Delayed-Commit Pattern Computation via Resonant Assembly Language: A Classical Pre-Fact Computational Architecture

Abstract

We introduce Delayed-Commit Pattern Computation (DCPC), a classical computational architecture that separates pre-fact computation from irreversible Boolean commitment. DCPC replaces premature bit assignments with a compact Resonant Assembly Language (RAL) marker alphabet that encodes constraint status (admissible, forbidden, reinforced, suppressed) rather than truth values. Computation proceeds via global constraint propagation and delayed commitment, with facts emerging only at measurement when coherence thresholds are crossed.

DCPC captures several structural advantages commonly associated with quantum computation—delayed resolution, global updates, and landscape shaping prior to commitment—without requiring quantum hardware. We show how standard heuristics such as Conflict-Driven Clause Learning (CDCL) can execute natively on this substrate, eliminating much of the overhead associated with backtracking and repeated propagation on conventional machines. We also characterize when structured constraint oracles (with richer-than-Boolean outputs) can exceed Grover-style query bounds on promise-structured problems, without contradicting Grover optimality for unstructured Boolean-marking oracles.

DCPC includes a classical-mode obtained by immediate commitment, so workloads that benefit from early Booleanization can execute in standard form, while constraint-rich workloads can exploit delayed commitment. Heuristically (and dependent on workload mix), we estimate that ~70–75% of practical workloads are classical-equivalent (DCPC offers little advantage), ~20–25% are constraint-dominated and likely to see large constant-factor gains (projected at 10–1000× on favorable instances, pending empirical validation) via reduced backtracking and control-flow overhead, and ~3–5% fall into domains where quantum hardware offers unique asymptotic advantage (notably quantum simulation and period-finding primitives underlying factoring/discrete-log cryptanalysis). We discuss engineering feasibility on current substrates (FPGAs, ASIC accelerators, near-memory compute) and note that DCPC requires no cryogenic cooling, coherence preservation, or quantum error correction of the type required by current leading quantum hardware platforms—operating at ambient temperatures with standard semiconductor technology. We present provable architectural advantages including profiling-based speedup bounds (Amdahl-style) and parallel depth improvements under bounded-degree locality assumptions. The central empirical question—whether DCPC accelerators outperform state-of-the-art CDCL solvers on SAT Competition industrial instances—remains testable and is addressed via a proposed benchmarking protocol. This positions DCPC as a practical, deployable, near-term architecture for accelerating constraint-rich computation.

Table of Contents

How to read this paper. Sections 2–4 define the DCPC/RAL pre-fact computational model, marker dynamics, and relationship to existing constraint propagation techniques. Section 5 provides worked examples (2-SAT and 3-SAT). Sections 6–7 describe oracle marking, diffusion, and the measurement criterion. Section 8 introduces the Taylor Limit as a physical admissibility constraint. Section 9 clarifies oracle structure and the conditions under which structured oracles enable faster-than-Grover search (without contradicting Grover optimality). Sections 10–11 connect the model to modern heuristics (CDCL), derive analytical speedup estimates, identify non-examples where DCPC provides no advantage, compare against classical and quantum

approaches, address anticipated reviewer questions, define the empirical crux question with falsifiable benchmarking criteria, present provable architectural advantages, and discuss the maturity asymmetry between DCPC proposals and decades-optimized CDCL solvers. Section 12 addresses practical engineering, energy, deployment constraints, includes a software simulator specification for immediate testing, and reports preliminary toy simulation results. Section 13 provides a comprehensive problem taxonomy with estimated workload distributions. Section 14 discusses applications including AI workloads where DCPC offers particular advantages (symbolic reasoning, planning, hybrid neuro-symbolic systems) and where it does not (deep learning training). Sections 15–16 discuss conceptual significance and conclusions.

---

1. Introduction

For general readers: This paper proposes a new way of computing that sits between ordinary computers and quantum computers. The key idea is simple: don't make decisions until you absolutely have to. By keeping options open longer, we can solve certain problems much faster than traditional computers—without needing exotic quantum hardware.

Classical computation commits to facts early: bits are set, states are chosen, and search proceeds via explicit branching. Quantum computation delays commitment by evolving superpositions until measurement. The key computational advantage arises not from exotic physics per se, but from three structural features:

1. **Delayed resolution** — no irreversible commitment until measurement
2. **Global updates** — transformations act on entire possibility spaces simultaneously
3. **Entropy shaping** — systematic concentration of probability mass prior to fact creation

In plain terms: imagine solving a maze. A classical computer tries one path, hits a dead end, backtracks, tries another. A quantum computer evolves a superposition and uses interference to concentrate probability on good paths. Our approach—DCPC—keeps track of which paths are "still possible" and "looking promising" without actually committing to any specific path until the very end. This avoids the wasted effort of backtracking.

This work explores whether these advantages can be recovered classically by changing what is stored prior to resolution. Instead of bits representing provisional facts, we store RAL markers representing admissibility, incompatibility, oscillation, and reinforcement. Computation becomes the evolution of admissibility rather than the manipulation of values.

We call this framework Delayed-Commit Pattern Computation (DCPC), with RAL providing the concrete instruction set for marker dynamics. DCPC explicitly separates two computational regimes:

- **Pre-fact regime**: continuous or discrete latent variables, reversible updates, no distinguishable Boolean outcomes
- **Fact regime**: irreversible thresholding, Boolean commitment, classical logic

Only the final measurement stage produces facts. This separation isolates delayed commitment as the operative mechanism underlying much of quantum computational advantage.

---

2. From Bit Matrices to Marker Matrices

For general readers: Ordinary computers store information as bits—zeros and ones. Each bit is a definite answer: yes or no, true or false. We propose storing something different: not answers, but "constraint status"—information about what's still possible, what's ruled out, and what's looking good. Think of it like a detective's case board with notes saying "suspect A: unlikely," "suspect B: promising," "suspects C and D: can't both be guilty"—rather than prematurely declaring "the butler did it."

2.1 The Bit-Matrix Possibility Computer

In a bit-matrix possibility computer, each row represents a candidate solution and each column a binary variable. Gates reweight rows until a solution is selected. This approach enables delayed commitment but remains value-centric: entries are provisional truth values awaiting confirmation.

2.2 The RAL-Marker Model

In the RAL-marker model, each matrix entry is drawn from a finite alphabet of markers encoding constraint status rather than truth value. Rows no longer represent solutions but assembly traces—histories of how constraints interact and evolve.

Definition 2.1 (RAL Marker Alphabet). The standard RAL alphabet Σ consists of:

Symbol	Name	Interpretation	***Plain English***
$\varnothing$	Forbidden	Constraint violation; inadmissible	*"Definitely not this"*
$\circ$	Admissible	Constraint satisfied; open	*"Still possible"*
$\rightleftharpoons$	Oscillatory	Underdetermined; cycling between states	*"Can't decide yet"*
$\uparrow$	Reinforced	Amplified by constructive interaction	*"Looking promising"*
$\downarrow$	Suppressed	Damped by destructive interaction	*"Probably not"*
$\perp$	Incompatible	Mutual exclusion with neighboring markers	*"Can't have both"*
✓	Coherent	Locally stable; ready for commitment	*"Ready to commit"*

Markers may coexist and evolve without forcing resolution to bits. The matrix $M \in \Sigma^{n \times m}$ represents n assembly traces over m constraint variables.

The key insight: instead of guessing an answer and checking if it works (then backtracking if it doesn't), we track the "status" of all possibilities simultaneously and let bad options eliminate themselves.

2.3 Pattern-Based Representation

Instead of enumerating candidates explicitly, DCPC stores admissibility landscapes implicitly using compressed representations:

- **Constraint graphs**: Nodes are variables; edges encode compatibility relations
- **Low-rank tensor decompositions**: $M \approx \Sigma_i \lambda_i \cdot u_i \otimes v_i$ when correlations are structured
- **Procedural generators**: Functions that produce marker configurations on demand

Compression is preserved as long as correlations remain structured. The representation decompresses only upon measurement.

---

3. Gates as Marker Rewrite Operators

For general readers: In ordinary computers, "gates" are the basic operations—AND, OR, NOT—that manipulate bits. In our system, gates instead update constraint status. Think of it like rules in a game of Sudoku: "if this cell can't be 5, and the only other option was 5 or 7, then it must be 7." These rules propagate automatically, narrowing down possibilities without ever guessing wrong.

Computation proceeds via marker rewrite rules rather than Boolean gates. A gate is a local or global operator $G: \Sigma^k \to \Sigma^k$ that transforms marker patterns according to admissibility logic.

Definition 3.1 (Rewrite Rules). Standard RAL rewrite rules include:

Incompatibility propagation:

$(\circ, \circ) \to (\perp, \perp)$ when constraints mutually exclude

Example: "If hiring Alice means we can't hire Bob, mark them as incompatible."

Oscillation decay:

$\rightleftharpoons^n \to \downarrow$ after n cycles without stabilization

Example: "If we've gone back and forth on this option too many times, downgrade it."

Reinforcement closure:

$(\uparrow, \uparrow, ..., \uparrow) \to \checkmark$ when closed loop of mutual reinforcement

Example: "If options A, B, and C all support each other, they're ready to commit."

Definition 3.3 (Reinforcement Closure). Let G = (V, E) be the constraint graph where vertices are marker positions and edges encode constraint dependencies. A set S ⊆ V forms a *closed reinforcement loop* if:

1. All positions in S carry ↑ markers: ∀v ∈ S: M[v] = ↑
2. S is strongly connected in G: for any u, v ∈ S, there exists a directed path from u to v
3. No position in S has unsatisfied dependencies outside S: ∀v ∈ S, ∀(u,v) ∈ E: u ∈ S or M[u] ∈ {✓, ↑}

When these conditions hold, all positions in S transition to coherent: ∀v ∈ S: M[v] ← ✓

Forbidden absorption:

$(\emptyset, \sigma) \rightarrow (\emptyset, \emptyset)$ for any $\sigma \in \Sigma$ (forbidden propagates)

Example: "If one part of a solution is impossible, the whole solution is impossible."

Rewrite rules operate across the entire matrix simultaneously via parallel application, providing global constraint propagation without explicit enumeration of outcomes.

3.1 Non-Collapsing Gate Property

Definition 3.2 (Non-Collapsing Gate). A gate G is non-collapsing if it satisfies:

$\forall M \in \Sigma^{nxm}: G(M) \in \Sigma^{nxm}$

That is, non-collapsing gates preserve the marker representation and never force Boolean commitment. All pre-fact computation uses only non-collapsing gates.

Clarification: This is stronger than mere type preservation. Non-collapsing gates must also preserve the reversibility of marker state in the sense that no information required to reconstruct prior marker configurations is destroyed. The key operational property is that non-collapsing gates cannot create Boolean facts—only measurement gates can do so.

3.2 Relationship to Existing Constraint Propagation Techniques

RAL marker dynamics share structural similarities with several established techniques. We clarify the relationship and distinguish RAL's contributions.

Arc consistency (AC-3/AC-4). Classical arc consistency algorithms [11, 14] propagate domain restrictions through constraint networks. RAL's incompatibility propagation (Rules R1–R3) serves a similar function. The distinction: AC algorithms operate on explicit domain representations and produce *reduced domains*; RAL operates on implicit marker configurations

and produces *admissibility landscapes*. RAL's ↑/↓ markers encode reinforcement/suppression gradients that AC algorithms do not track.

Belief propagation and message passing. Loopy BP [31, 32] propagates probabilistic beliefs through factor graphs. RAL's marker dynamics can be viewed as a discretized, non-probabilistic analogue: markers encode constraint status rather than probability mass, and propagation follows logical rules rather than sum-product updates. The key difference is that RAL never represents explicit probability distributions—it tracks *admissibility* rather than *likelihood*.

Watched literals and unit propagation. Modern CDCL solvers [7, 8] use watched literal schemes for efficient unit propagation. RAL's claim is not that marker rewriting is faster than watched literals on conventional hardware—it is that marker dynamics *subsume* unit propagation while also capturing reinforcement, suppression, and oscillation states that CDCL tracks implicitly through activity scores and restart heuristics. On dedicated DCPC hardware, these dynamics execute natively rather than through control-flow emulation.

Survey propagation. For random SAT instances near the satisfiability threshold, survey propagation [33] outperforms CDCL by tracking distributions over warnings rather than single assignments. RAL's oscillation markers (⇌) and reinforcement gradients (↑/↓) capture related intuitions in a discrete, deterministic framework. Survey propagation's probabilistic machinery may be recoverable as a mean-field approximation of RAL dynamics, though we do not pursue this connection here.

What RAL adds. The novel contribution is not any single propagation mechanism but the *unified framework* that:

1. Separates pre-fact computation from Boolean commitment
2. Treats oscillation and reinforcement as first-class computational states
3. Enables delayed measurement with explicit readiness criteria
4. Maps naturally onto parallel hardware without control-flow overhead

RAL is best understood not as a replacement for existing techniques but as an *architectural substrate* on which existing techniques execute more efficiently.

3.3 What is New (and What is Not)

To preempt a natural question—"Is this genuinely novel, or a redescription of arc consistency / belief propagation / CDCL?"—we explicitly delineate the claims:

What is NOT new:

- Constraint propagation as a reasoning technique
- Message passing and belief propagation algorithms
- Survey propagation for hard random instances
- CDCL's core inference mechanisms (unit propagation, conflict analysis, clause learning)

- The observation that structured problems admit faster solutions than worst-case bounds suggest

What IS new:

- **Pre-fact/fact semantic split:** The explicit separation of a computational regime where no Boolean facts exist from the moment of irreversible commitment
- **Marker-state substrate:** A representation where constraint status (admissible, forbidden, reinforced, suppressed, oscillating) is the native data type, not an emulation layer atop Boolean assignments
- **Rewrite-as-native-physics:** Marker dynamics that execute as bulk synchronous or systolic updates without branch-heavy control flow, trail semantics, or rollback machinery
- **Measurement as deliberate irreversible commit:** The explicit criterion for when pre-fact computation terminates and Boolean artifacts are exported
- **Hardware implications:** The architectural claim that dedicated DCPC hardware could execute constraint propagation without the overhead that fact-native semantics impose

The architectural analogy: The relationship between DCPC and conventional SAT solving is analogous to the relationship between SIMD/TPU execution and scalar CPU emulation of vector operations. The underlying mathematical operations may be similar, but the cost profile changes dramatically when the representation and update primitives match the computation's natural structure.

A concession that increases credibility: We acknowledge that the magnitude of speedup is ultimately an empirical question. Modern CDCL solvers are highly optimized, and the "wasted work" from rollback and re-propagation is not a bug but a consequence of fact-native semantics that enable other optimizations (e.g., efficient watched literals, cache-friendly trail access). Our claim is architectural plausibility and a well-defined hypothesis, not proven dominance. Empirical validation against state-of-the-art solvers on standard benchmarks is required.

---

4. Interference Without Amplitudes

For general readers: One of quantum computing's "magic tricks" is interference—where possibilities can cancel each other out, like waves in water. Our system achieves something similar without quantum physics: incompatible options simply eliminate each other through logical contradiction. If option A requires X and option B requires not-X, both options collapse. No waves, no complex numbers—just constraint logic doing the work.

In quantum computing, destructive interference arises from complex amplitudes summing to zero:

$\psi\_total = \Sigma_i \alpha_i |i\rangle$, where $\Sigma_i \alpha_i = 0$ for cancelled paths

In the RAL-marker matrix, cancellation is structural rather than numerical.

Theorem 4.1 (Structural Interference). Two assembly traces T_1, T_2 interfere destructively if and only if they contain mutually incompatible markers at corresponding positions:

$\exists j: M[T_1, j] \perp M[T_2, j] \Longrightarrow$ both traces annihilate

Proof sketch. Incompatibility propagates via the forbidden absorption rule (R1). Propagation terminates in at most O(nm) steps since each marker can transition to ∅ at most once, and ∅ is absorbing. Once any position in a trace becomes ∅, the entire trace is marked inadmissible and excluded from further evolution. □

Corollary 4.2. Interference in RAL is admissibility failure, not numerical subtraction. This eliminates the need to simulate complex amplitudes while preserving the computational effect of excluding large regions of possibility space simultaneously.

---

5. Worked Example: 2-SAT With Marker Evolution

For general readers: This section shows the system in action on a simple logic puzzle. Don't worry about the mathematical notation—focus on how the "status" of each option evolves from "still possible" through "looking good" to "ready to commit," all without ever making a wrong guess that needs to be undone.

5.1 Basic Example: 2-SAT

To make the dynamics concrete, we trace DCPC/RAL evolution on a minimal satisfiable 2-SAT instance.

Problem. Consider the formula over variables x, y:

$(x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y)$

In plain English: We need to choose yes/no for x and y such that: (1) at least one is yes, (2) if x is yes then y must be yes, and (3) if y is yes then x must be yes. The only solution is x=yes, y=yes.

This instance has a unique satisfying assignment: x = 1, y = 1. We show how marker dynamics converge to this solution without branching or backtracking.

Representation. For each variable $v \in \{x, y\}$, we track two slots (v = 0) and (v = 1) with markers indicating admissibility pressure. Clauses act as rewrite rules: violated literals are suppressed (○ → ↓), unit-propagated literals are reinforced (○ → ↑), and closed reinforcing loops promote to coherent (↑ → ✓). For readability we use a simplified projection rule: when a clause

excludes a joint literal pair, the resulting incompatibility is projected onto the involved literals as soft suppression/reinforcement (a mean-field approximation of full trace-level dynamics).

Step 0 — Initial state (fully pre-fact).

Trace	**x = 0**	**x = 1**	**y = 0**	**y = 1**
T_0	○	○	○	○

All literal choices are admissible and unresolved.

Step 1 — Apply clause (¬x ∨ y).

If x = 1 and y = 0, the clause fails. This creates suppression pressure against y = 0 and reinforcement toward y = 1:

Trace	**x = 0**	**x = 1**	**y = 0**	**y = 1**
T_1	○	○	↓	↑

Step 2 — Apply clause (x ∨ ¬y).

If y = 1 and x = 0, the clause fails. This suppresses x = 0 and reinforces x = 1:

Trace	**x = 0**	**x = 1**	**y = 0**	**y = 1**
T_2	↓	↑	↓	↑

Both variables now have consistent directional pressure toward x = 1, y = 1, but nothing is yet a Boolean fact.

Step 3 — Apply clause (x ∨ y).

This clause is satisfied by x = 1 or y = 1. Since both are already reinforced, the clause closes a reinforcing loop and upgrades to coherent:

Trace	**x = 0**	**x = 1**	**y = 0**	**y = 1**
T_3	↓	✓	↓	✓

Step 4 — Measurement (threshold gate).

The measurement gate checks readiness: no forbidden markers, no incompatibilities, high coherence. Since x = 1 and y = 1 are coherent and competing literals are suppressed, the system commits:

Output: x = 1, y = 1

What this demonstrates. No branching, no backtracking, no premature assignments—yet constraint propagation drives global convergence to the satisfying assignment via marker dynamics. "Interference" appears as structural inadmissibility: suppressed alternatives fail to sustain coherence and do not survive the criticality test.

5.2 Extended Example: 3-SAT Fragment with Backtrack Elimination

To illustrate DCPC's advantage over classical CDCL more clearly, we consider a 3-SAT instance that induces backtracking on classical solvers.

Problem. Consider the formula over variables x, y, z:

$(x \vee y \vee z) \wedge (\neg x \vee y \vee z) \wedge (x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z) \wedge (x \vee \neg y \vee \neg z)$

This instance has exactly one satisfying assignment: x = 1, y = 1, z = 1.

Classical CDCL behavior. A DPLL/CDCL solver might proceed:

1. Decide x = 0
2. Propagate → no unit clauses
3. Decide y = 0
4. Propagate → clause $(x \vee y \vee z)$ forces z = 1
5. Check remaining clauses... clause $(x \vee y \vee \neg z) = (0 \vee 0 \vee 0)$ = **conflict**
6. Backtrack, learn clause, try different assignment...

Multiple backtracks may occur before finding x = y = z = 1.

RAL behavior.

Step	x=0	x=1	y=0	y=1	z=0	z=1	Trigger
0	○	○	○	○	○	○	Initial state
1	↓	↑	○	○	○	○	Clauses 2,5,6 suppress ¬x configurations
2	↓	↑	↓	↑	○	○	Clauses 3,5,7 suppress ¬y configurations
3	↓	↑	↓	↑	↓	↑	Clauses 4,6,7 suppress ¬z configurations
4	↓	✓	↓	✓	↓	✓	Reinforcement closure

No backtracking occurred. The marker dynamics converge monotonically to the solution. Each clause contributes suppression pressure against assignments that would violate it; the cumulative effect concentrates reinforcement on the unique satisfying assignment.

What this demonstrates. On instances where CDCL thrashes through similar partial assignments, RAL's parallel suppression/reinforcement achieves the same logical conclusion without the sequential trial-and-error of classical search. The overhead eliminated is not the

constraint checking itself, but the control-flow machinery for managing commitment and reversal.

---

6. Oracle Marking and Grover Geometry

6.1 Oracle as Pattern Deformation

In quantum search, the oracle U_x^* marks the target state by phase inversion:

$U_x^* |x\rangle = (-1)^{\delta(x,x^*)} |x\rangle$

In DCPC, oracle marking is a pattern deformation rather than fact revelation. The oracle O_RAL transforms marker configurations:

O_RAL: M[i, ·] ↦ { ↑ if constraint(i, x*) satisfied { ↓ if constraint(i, x*) violated { ⇌ if constraint(i, x*) underdetermined

6.2 Diffusion as Landscape Reshaping

Grover diffusion $D = 2|\psi\rangle\langle\psi| - I$ reflects amplitudes about their mean. The RAL analogue applies global rebalancing:

Definition 6.1 (RAL Diffusion). For marker matrix M with reinforcement counts r(i) = |{j : M[i,j] = ↑}|, the diffusion operator D_RAL applies:

M[i, j] ↦ ↑ if r(i) > $\bar{r}$ (mean reinforcement) M[i, j] ↦ ↓ if r(i) < $\bar{r}$

This concentrates admissibility into traces with above-average constraint satisfaction, creating a narrowing well in the possibility landscape without committing to any specific answer.

---

7. Measurement as Criticality

For general readers: "Measurement" here means the moment when we finally commit to a definite answer. But unlike a student guessing on a test, the system only commits when it's genuinely ready—when the constraints have narrowed things down to a clear winner. Think of it like a jury that deliberates until reaching unanimous agreement, rather than voting immediately and hoping for the best.

Measurement is not a readout but a phase transition. A row becomes a fact when its marker configuration crosses a critical threshold.

7.1 Coherence Monotone

To make the criticality condition operational, we introduce a coherence functional that increases as traces stabilize.

Definition 7.1 (Coherence Score). For trace T over m positions, define:

$C(T) = (1/m) \cdot \Sigma_j w(M[T,j])$

where the weight function $w: \Sigma \to \mathbb{R}$ assigns:

Marker	w(σ)	Rationale
✓	+1.0	Fully coherent
↑	+0.5	Reinforced but uncommitted
○	0.0	Neutral/open
↓	−0.3	Suppressed
⇌	−0.5	Unstable oscillation
⊥	−0.8	Incompatibility present
∅	−1.0	Forbidden/failed

These canonical weights satisfy:

- $C(T) \in [-1, 1]$ for any trace
- $C(T) = 1$ iff all positions are coherent (✓)
- C(T) monotonically increases as traces stabilize under normal dynamics

The measurement threshold τ is typically set in [0.7, 0.9]; lower values permit earlier commitment at the cost of potential instability.

Properties:

- C(T) increases when reinforcement closes into coherence (↑ → ✓) or when suppression removes unstable alternatives
- C(T) decreases when forbidden/incompatibility/oscillation proliferate
- C serves as a progress measure whose sustained rise indicates approach to a stable fact

7.2 Measurement Criterion

Definition 7.2 (Measurement Readiness). An assembly trace T is measurement-ready if:

1. **No forbidden markers:** $\forall j: M[T, j] \neq \varnothing$
2. **No incompatible markers:** $\forall j: M[T, j] \neq \perp$
3. **Oscillation damped:** $\#\{j : M[T, j] = \rightleftharpoons\} < \varepsilon \cdot m$ for threshold ε

4. **Coherence achieved:** $\#\{j : M[T, j] = \checkmark\} > (1 - \varepsilon) \cdot m$
5. **Coherence score threshold:** $C(T) \geq \tau$ for fixed $\tau \in (0, 1)$

Only at this point are bits extracted. Facts are derived artifacts, not primitive computational objects.

Critical clarification: ✓ denotes stability, not commitment. Measurement is not "discovering" the solution; it is the point at which the system accepts irreversible commitment and exports a Boolean artifact. Pre-fact dynamics may converge to a dominant configuration (all ✓ markers) prior to measurement, but until commitment the computation has not incurred the costs associated with fact semantics: trail management, rollback machinery, branch-conditioned execution. The ✓ marker indicates that a position has reached local stability in the pre-fact field—it is reinforced, self-consistent, and no longer oscillating—but it remains reversible and is not yet addressed as a Boolean fact that would trigger conditional control flow. This distinction is the source of DCPC's architectural advantage: the system can "know" the answer without paying the costs of knowing it in fact-native terms.

7.3 Delayed Choice

The measurement basis—which constraints to evaluate for final commitment—can be delayed until the final moment. This mirrors delayed-choice experiments in quantum mechanics and follows from the non-collapsing gate property: no information is destroyed until the threshold gate fires.

---

8. The Taylor Limit and Admissible Compressibility

For general readers: This section addresses a fundamental question: are there limits to how precisely any computer—classical, quantum, or DCPC—can distinguish between options? We argue yes: there's a maximum "resolution" set by physics itself. Beyond this limit, additional precision is meaningless. This isn't a limitation of our technology; it's a feature of physical reality that constrains all computation.

To sharpen the distinction between compressible and incompressible problems, we introduce the Taylor Limit: an upper bound on physically meaningful distinction.

Methodological note. We treat the Taylor Limit as an admissibility postulate—a physically motivated upper bound on realizable distinguishability for finite-entropy systems—rather than as a new derived constant of nature.

8.1 Motivation

The Planck scale defines a lower bound on spatial and temporal resolution:

$\ell_P = \sqrt{(\hbar G/c^3)} \approx 1.616 \times 10^{-35}$ m $t_P = \sqrt{(\hbar G/c^5)} \approx 5.391 \times 10^{-44}$ s

The Taylor Limit complements this by establishing a maximum scale at which distinctions retain physical meaning.

8.2 Definition

Definition 8.1 (Taylor Limit). The Taylor Limit Λ_T is the maximum number of distinguishable states supportable by a physical system of bounded entropy:

$\Lambda_T = e^{(S_{max}/k_B)}$

where S_{max} is the maximum entropy of the system and k_B is Boltzmann's constant.

For a system with n bits of information capacity:

$\Lambda_T = 2^n$

Beyond Λ_T, additional mathematical precision does not correspond to any realizable physical difference and cannot support stable fact creation.

8.3 Implications for Computation

Theorem 8.1 (Scale-Induced Halting). All physically admissible computations necessarily halt or enter bounded oscillation within the Taylor envelope.

Proof sketch. Marker evolution proceeds by iterative refinement of admissibility. Each refinement step requires distinguishing previously indistinguishable configurations. Once the number of required distinctions exceeds Λ_T, no further refinement is possible. The system must therefore:

- Stabilize (measurement-ready)
- Enter bounded oscillation ($\rightleftharpoons$ markers persist)
- Terminate without fact creation ($\oslash$ propagates globally) □

How the Taylor Limit constrains DCPC computation:

1. **Marker precision bound.** The coherence score C(T) cannot be computed to arbitrary precision. For a system with n bits of capacity, coherence distinctions finer than 2^{-n} are physically meaningless. This bounds the precision of the measurement threshold τ.
2. **Trace count limit.** The marker matrix $M \in \Sigma^{n \times m}$ implicitly represents a set of assembly traces. When the effective number of distinguishable traces exceeds Λ_T, the representation must compress or the computation becomes inadmissible. This provides a physical interpretation of the transition from "tractable structured instance" to "intractable unstructured instance."

3. **Oscillation termination guarantee.** Theorem 8.1 ensures that no DCPC computation can oscillate indefinitely. Within the Taylor envelope, the system must either stabilize, enter bounded cycles, or terminate. This grounds the measurement criterion (Definition 7.2) in physical necessity rather than arbitrary convention.

Engineering implication: The Taylor Limit imposes a practical constraint on DCPC parameter tuning. The measurement threshold τ and oscillation damping parameter ε (Definition 7.2) cannot be tuned to arbitrary precision; beyond Λ_T-bounded resolution, finer distinctions are not physically stable. This means that for a system with n-bit marker resolution, the effective precision of τ and ε is bounded by $\sim 2^{-n}$. Attempting finer tuning yields no computational benefit and may introduce instability. This is "real work" the Taylor Limit does in the theory: it converts an apparently arbitrary precision choice into a principled bound.

8.4 Problem Classification

This reframes computational complexity in physically grounded terms:

Class	Definition	Example
Taylor-compressible	Solution space collapses within Λ_T	Structured SAT, optimization
Hard-but-admissible	Approaches Λ_T while remaining resolvable	Cryptographic verification
Inadmissible	Requires distinctions exceeding Λ_T	Formal but unrealizable

Corollary 8.2. Measurement is permitted only when both conditions are satisfied:

1. Global coherence / measurement readiness (Definition 7.2)
2. Compliance with Taylor-bounded distinguishability (Λ_T)

This unifies entropy collapse, criticality, and bounded resolution within a single principle.

---

9. Structured Oracles and Query Complexity

For general readers: Grover's algorithm is a famous quantum computing result showing that searching an unsorted database of N items takes about $\sqrt{N}$ steps on a quantum computer (versus N steps classically). This section shows that the key factor is not quantum versus classical, but rather how much information the oracle provides per query. When oracles return structured constraint information rather than simple yes/no answers, search complexity drops dramatically—for both quantum and classical approaches.

Important clarification. This section does not claim to "beat Grover" on the same problem. Grover's lower bound applies to unstructured Boolean-marking oracles. We demonstrate that real-world problems often admit stronger-than-Boolean oracles that return structured constraint

information, and that DCPC can exploit such structure natively. This is a statement about oracle models, not a refutation of Grover optimality.

9.1 The Grover Bound and Its Oracle Model

Grover's algorithm achieves $\Theta(\sqrt{N})$ query complexity for unstructured search, and this is provably optimal for Boolean-marking oracles that return a single bit per query:

$O_{Boolean}: \{0,1\}^n \to \{0,1\}$

This bound assumes:

1. The oracle returns exactly one bit per query
2. Queries evaluate complete candidate solutions
3. No structure beyond marked/unmarked is available

Real-world search problems often violate these assumptions. Constraint satisfaction problems, for instance, naturally support *partial evaluation*: we can ask whether a partial assignment is consistent with constraints before completing it.

9.2 Structured Oracle Extension

Theorem 9.1. To exceed Grover's bound, an oracle must encode structure beyond binary marking. The minimal RAL alphabet sufficient for this purpose is ternary:

$\Sigma_{min} = \{\oslash, \circ, \rightleftharpoons\}$

denoting incompatible, admissible, and unresolved constraint status.

Proof. A binary alphabet {marked, unmarked} is isomorphic to Grover's oracle model and inherits its $\Omega(\sqrt{N})$ lower bound. A ternary alphabet enables constraint-level queries that return structured, multi-bit information. □

9.3 Constraint Oracle Protocol

Definition 9.1 (RAL Constraint Oracle). For hidden target $x^* \in \{0,1\}^n$, the constraint oracle O_C evaluates partial constraints:

$O_C(\text{prefix } p, \text{length } k) = \{ \circ \text{ if } p = x^*[1{:}k] \ \{ \oslash \text{ if } p \neq x^*[1{:}k]$
$\{ \rightleftharpoons \text{ if } k = 0 \text{ (trivial constraint)}$

Theorem 9.2 (Logarithmic Recovery). Using O_C, the target x^* can be recovered in $O(n) = O(\log N)$ oracle calls via iterative constraint extension.

Proof. Initialize $p = \emptyset$. At each step, query $O_C(p\|0, |p|+1)$ and $O_C(p\|1, |p|+1)$. Exactly one returns $\circ$; extend p accordingly. After n steps, $p = x^*$. Total queries: $2n = O(\log N)$. □

9.4 Worked Example: Prefix-Constraint Oracle

Let the hidden target be x* = 1011 (so n = 4, N = 16). We query the constraint oracle O_C and evolve a single trace representing the growing constraint.

Step 0 (empty constraint). p = ε (no prefix yet). State is unresolved: ⇌

Step 1 (first bit). Query O_C(0) → ∅, O_C(1) → ○. Extend p = 1. First position becomes ✓.

Step 2 (second bit). Query O_C(10) → ○, O_C(11) → ∅. Extend p = 10. Second position becomes ✓.

Step 3 (third bit). Query O_C(100) → ∅, O_C(101) → ○. Extend p = 101. Third position becomes ✓.

Step 4 (fourth bit). Query O_C(1010) → ∅, O_C(1011) → ○. Extend p = 1011 = x*. Fourth position becomes ✓.

Total queries: 2n = 8 = O(log N). *Comparison:* With a Boolean-marking oracle, Grover requires $\Theta(\sqrt{N})$ oracle calls ($\approx (\pi/4)\sqrt{N}$ for one marked item), whereas the prefix-constraint oracle enables O(log N) recovery because each query returns structured constraint information.

What this demonstrates. The speedup arises because the oracle supplies structured constraint information (prefix consistency) rather than Boolean marking of whole candidates. The RAL ternary alphabet captures this structure directly.

9.5 Relationship to Grover

This does not contradict Grover's optimality proof because the oracle model itself differs:

Property	Grover Oracle	Structured Constraint Oracle
Output alphabet	{0, 1}	{∅, ○, ⇌}
Query granularity	Full candidate	Partial constraint
Information per query	1 bit	O(1) bits + structure
Applicable lower bound	$\Omega(\sqrt{N})$	$\Omega(\log N)$ when prefix-structured
Relationship	Grover-optimal for this model	Different model; Grover bound does not apply

The advantage derives from structured oracle output, not from parallel evaluation of candidates. Quantum computers can also exploit structured oracles—indeed, quantum algorithms for structured search exist.

Why this matters for DCPC. The significance for DCPC is that RAL's marker alphabet *natively represents* the ternary constraint structure, whereas quantum approaches must encode this structure into amplitude patterns. On problems where constraint structure dominates, DCPC exploits the structure directly without the encoding overhead that quantum approaches require. This is not a claim of superiority over quantum computing in general, but rather identification of a problem class where DCPC's native representation provides practical advantage.

---

10. Native Execution of Classical Heuristics

For general readers: Modern software for solving logic puzzles (SAT solvers) uses clever tricks to avoid wasting time. The main trick—called CDCL—involves guessing, checking, and learning from mistakes. But all that bookkeeping is overhead. On DCPC hardware, the same logic happens naturally through marker evolution: bad options eliminate themselves, good options reinforce each other, and there's nothing to "undo" because we never committed to wrong guesses in the first place. This section shows how a leading classical technique becomes dramatically simpler on our architecture.

10.1 CDCL on Classical Hardware

Conflict-Driven Clause Learning (CDCL) is the dominant framework for modern SAT solvers. On classical hardware, CDCL proceeds by:

1. **Decide**: Commit to a variable assignment (fact creation)
2. **Propagate**: Derive implications via unit propagation
3. **Conflict**: Detect unsatisfiable partial assignment
4. **Learn**: Add conflict clause to database
5. **Backtrack**: Undo assignments to decision level

Much computational effort is spent managing control flow rather than resolving constraints. Backtracking dominates runtime in thrashing regimes where near-identical partial assignments are explored repeatedly.

10.2 CDCL on RAL Hardware

On a RAL matrix computer, CDCL's logical intent is preserved but its control-flow machinery disappears:

Classical CDCL	RAL Native
Variable guess	Admissible marker ○
Unit propagation	Marker rewrite rules
Conflict detection	Incompatibility ⊥ propagation
Clause learning	Structural annihilation (automatic)

Classical CDCL	RAL Native
Backtracking	Not needed (no commitment made)

Theorem 10.1. CDCL executed on RAL hardware eliminates:

- Clause database management
- Implication graph construction
- Re-propagation after backtracking

Incompatible assemblies cease to propagate; compatible configurations are reinforced. Failure is local and permanent rather than global and reversible.

10.3 Performance Characteristics (Conjectured)

Note: The speedup estimates below are analytical projections based on overhead models, not empirical measurements. They reflect the theoretical elimination of backtracking and control-flow overhead, pending validation through simulation and benchmarking on standard SAT Competition and SMT-LIB instances.

Instance Type	Classical CDCL	RAL CDCL	Projected Speedup
Incompressible/adversarial	T	~T	~1×
Structured industrial	T	T/c	c× (c ≈ 10–100)
Thrashing-heavy	T	T/c'	c'× (c' ≈ 100–1000)

Rationale for estimates: Classical CDCL implementations spend substantial cycles on implication graph maintenance, clause database management, and re-propagation after backtracking. On thrashing-heavy instances, the same partial assignments may be explored repeatedly. RAL marker dynamics eliminate these overheads by never committing to partial assignments. The projected speedup ranges reflect the fraction of runtime typically consumed by these operations in profiled SAT solvers, but await empirical confirmation.

The speedup arises from overhead elimination, not from solving a harder complexity class. On incompressible instances, RAL provides no asymptotic advantage. On structured instances, large constant-factor improvements are projected but require empirical validation.

10.4 Analytical Basis for Speedup Estimates

The projected constant-factor speedups derive from overhead elimination, not algorithmic improvement. We decompose classical CDCL runtime into:

$$T_{classical} = T_{propagation} + T_{conflict} + T_{backtrack} + T_{bookkeeping}$$

where:

- **T_propagation**: Time spent in unit propagation
- **T_conflict**: Time spent detecting and analyzing conflicts
- **T_backtrack**: Time spent undoing assignments and re-propagating
- **T_bookkeeping**: Time spent maintaining clause databases, implication graphs, activity scores

Empirical profiling of modern SAT solvers on industrial benchmarks shows characteristic distributions:

Component	Share (structured instances)	Share (random/hard instances)
Propagation	15–30%	40–60%
Conflict analysis	10–20%	15–25%
Backtracking + re-propagation	30–50%	10–20%
Bookkeeping overhead	20–40%	15–25%

Note: These distributions are informed by profiling studies of MiniSat, Glucose, and similar solvers [7, 8]; exact values vary by instance family and solver implementation. Runtime distributions estimated from published solver profiling studies and SAT Competition analyses.

On DCPC/RAL hardware:

- **T_propagation**: Comparable (marker rewrite ≈ unit propagation)
- **T_conflict**: Reduced (incompatibility detected locally, no graph construction)
- **T_backtrack**: Eliminated (no commitment to undo)
- **T_bookkeeping**: Largely eliminated (no clause database, no implication graph)

Estimated speedup derivation:

If backtracking and bookkeeping consume 50–70% of runtime on thrashing-heavy instances, their elimination yields a baseline speedup of:

Speedup_baseline ≈ 1 / (1 − overhead_fraction) = 1 / (0.3–0.5) ≈ 2–3×

This baseline improves further when:

- **Thrashing is severe** (repeated exploration of similar subspaces): ×10–100
- **Global constraint propagation** prunes large subspaces simultaneously: ×2–10
- **Memory bandwidth is limiting** (DCPC's compact marker representation reduces traffic): ×2–5

Composite estimate: 10–1000× on structured, thrashing-heavy instances; ~1× on random/incompressible instances where backtracking is minimal.

Important caveat: These estimates assume idealized DCPC hardware with negligible classical-mode overhead. Actual speedups will depend on implementation efficiency, memory hierarchy

effects, and instance characteristics. The range 10–1000× should be understood as an envelope spanning favorable conditions, not a guaranteed outcome. Empirical validation through simulation and benchmarking is required.

10.5 Non-Example: When DCPC Provides No Advantage

For completeness, we identify problem instances where DCPC offers no speedup over classical CDCL:

Random 3-SAT near threshold. For uniformly random 3-SAT instances at the satisfiability threshold (clause-to-variable ratio ≈ 4.26), classical CDCL already exhibits minimal thrashing—most runtime is spent on propagation, and backtracking is infrequent. DCPC's overhead elimination provides little benefit.

Adversarially constructed instances. Instances designed to defeat heuristics (e.g., hidden satisfying assignments with no exploitable structure) offer no constraint correlations for DCPC to exploit. RAL marker dynamics degenerate to exhaustive search.

Already-solved-in-linear-time instances. For problem instances solvable via pure unit propagation (e.g., Horn clauses, 2-SAT), classical solvers already achieve near-optimal performance. DCPC matches but does not exceed this.

Key insight: DCPC's advantage is not universal. It is specifically targeted at the substantial fraction of real-world instances that exhibit exploitable constraint structure and where classical solvers thrash. Understanding these limitations is essential for appropriate deployment.

10.6 Benchmarking Predictions and Evaluation Plan

To transform architectural claims into testable hypotheses, we specify concrete benchmarking predictions and an evaluation methodology.

Datasets:

- **SAT Competition industrial track:** Real-world verification, planning, and configuration instances with exploitable structure
- **SAT Competition crafted track:** Instances with known structure (e.g., graph coloring, cryptographic)
- **SAT Competition random track:** Uniformly random instances near satisfiability threshold
- **SMT-LIB benchmarks:** Satisfiability modulo theories instances from verification and synthesis

Baselines:

- **Kissat** [current SAT Competition winner for many categories]
- **CaDiCaL** [state-of-the-art CDCL implementation]

- **Glucose** [clause learning baseline]
- **Z3** [leading SMT solver]

Metrics beyond wall-clock time:

- Propagation events per second
- Backtracks per solved instance
- Memory bandwidth consumption
- Fraction of time in BCP vs. conflict analysis vs. restart
- Instances solved within timeout by category

Predictions:

Instance Category	DCPC vs CDCL Baseline	Rationale
Random 3-SAT (threshold)	~1×	Minimal thrashing; propagation-dominated
Random 3-SAT (underconstrained)	~1×	Easy instances; solver already fast
Industrial structured	2–100×	High backtrack/repropagation overhead
Crafted (regular structure)	5–500×	Exploitable constraint correlations
Thrashing-heavy industrial	10–1000×	Largest overhead elimination gains
Adversarial/cryptographic	~1×	No exploitable structure

Measurable hypothesis: DCPC outperforms CDCL when the baseline solver spends >30% of runtime on backtracking and re-propagation. This can be measured by profiling Kissat/CaDiCaL on each instance family.

Implementation path:

1. **Phase 1 (simulation):** Software emulation of RAL marker dynamics on standard hardware, comparing against CDCL on matched instances
2. **Phase 2 (FPGA prototype):** Hardware implementation demonstrating native rewrite dynamics
3. **Phase 3 (ASIC projection):** Performance modeling for dedicated silicon

This evaluation plan provides a falsifiable path forward: if DCPC simulation fails to show predicted gains on industrial instances with high backtrack ratios, the architectural hypothesis would be disconfirmed.

---

11. Comparison with Classical and Quantum Computing

For general readers: How does DCPC stack up against existing computers? The short answer: DCPC can do everything a classical computer can do (it's a superset), plus it's faster on problems with lots of constraints. Quantum computers have unique tricks DCPC can't match (like breaking encryption), but those tricks only help on a narrow class of problems. For the vast majority of real-world computing, DCPC is at least as good as classical and often much better.

11.1 DCPC vs Classical Computing

Property	Classical	DCPC
Fact commitment	Every gate	Final measurement only
Correlation preservation	Destroyed early	Maintained throughout
Backtracking	Required	Eliminated
Overhead	Control flow dominant	Computation dominant

Critical point: DCPC subsumes classical computation at the model level. Any classical algorithm can be executed on DCPC by committing markers immediately at each gate (classical-mode). In deployment, DCPC is designed not to impose overhead when early commitment is optimal, while providing large constant-factor gains on workloads where backtracking and control-flow dominate. DCPC is therefore **not expected to be worse than classical** in cases where early commitment is optimal, and **strictly better** on problems where delayed commitment eliminates backtracking or enables global constraint propagation.

11.2 DCPC vs Quantum Computing

Property	Quantum	DCPC
State compression	Exponential (Hilbert space)	Polynomial (pattern structure)
Interference mechanism	Amplitude cancellation	Admissibility failure
Shor's algorithm	Yes	No
Grover speedup	$\sqrt{N}$	Structure-dependent
Cryptographic threat	Yes	No

Quantum computers gain power from physical exponential compression of Hilbert space. DCPC reproduces quantum geometry but not quantum scaling. It cannot threaten cryptography or match Shor's algorithm on unstructured instances.

11.3 Conceptual Clarification

DCPC demonstrates that quantum advantage arises from two separable components:

1. **Delayed fact creation** — recoverable classically via DCPC
2. **Physical state compression** — requires quantum substrate

This separation clarifies the computational landscape:

- **DCPC ⊇ Classical:** DCPC subsumes classical computation at the model level; classical-mode (immediate commitment) imposes no overhead, while delayed commitment provides speedup on structured constraint problems
- **Quantum ⊄ DCPC:** Quantum has unique capabilities (Shor, quantum simulation) that DCPC cannot replicate
- **DCPC ⊄ Quantum:** DCPC provides practical advantages on constraint-rich problems where quantum offers little benefit

The practical implication: DCPC can in principle serve as a **general-purpose classical substrate** while quantum hardware addresses the narrow class of problems requiring Hilbert space compression.

11.4 Detailed Comparison: DCPC vs Quantum on Constraint-Dominated Workloads

For general readers: This section asks directly: for the problems where DCPC excels, how does it compare to quantum computing? The answer may surprise you—DCPC often matches or beats quantum approaches on practical problem sizes, despite quantum's stronger theoretical guarantees on paper.

For the ~20–25% of constraint-dominated workloads where DCPC provides large constant-factor gains, DCPC is typically competitive with or superior to known quantum approaches on real problem sizes, despite quantum computing's stronger asymptotic guarantees on narrower problem classes.

11.4.1 Sources of Speedup: A Fundamental Difference

The speedups from DCPC and quantum computing arise from fundamentally different mechanisms:

DCPC speedup derives from:

- Eliminating wasted classical work (backtracking, repeated propagation, control-flow churn)
- Preserving correlations until resolution
- Global constraint pruning without branching

This yields large constant-factor improvements (10×–1000×) on structured instances.

Quantum speedup derives from:

- Hilbert-space compression
- Amplitude amplification or interference
- Period-finding / spectral structure

This yields quadratic speedup (Grover) for unstructured search and exponential speedup for specific algebraic problems (Shor, quantum simulation).

11.4.2 Head-to-Head Comparison by Problem Domain

SAT / SMT / CSP / Verification

Paradigm	Typical Outcome
Classical CDCL	Exponential worst-case, heavy thrashing
DCPC	10×–1000× faster via overhead elimination
Quantum (NISQ / FTQC)	No clear advantage; encoding overhead dominates

Why quantum doesn't win here: Mapping SAT/SMT to QAOA/Ising formulations is expensive. QAOA circuit depth must scale with constraint tightness. No exponential quantum advantage is known for general SAT. Real-world instances are structured, not random—precisely the regime where DCPC excels.

Planning / Scheduling / Configuration

Paradigm	Typical Outcome
Classical	Backtracking + heuristics
DCPC	Large constant-factor gains
Quantum (QAOA / annealing)	Modest, unstable improvements

Why quantum doesn't win here: Planning problems are constraint graphs, not algebraic structures. Quantum offers no asymptotic edge on these problem classes. Noise sensitivity and embedding overhead eliminate quantum advantage on current and near-term hardware.

Combinatorial Optimization (structured instances)

Paradigm	Typical Outcome
Classical	Branch-and-bound, local search
DCPC	10×–100× on structured instances
Quantum (QAOA / annealing)	Small improvements at best

Why quantum doesn't win here: Even optimistic QAOA results typically show only constant-factor improvements, only on small instances, and fragile to noise and parameter choice. DCPC achieves comparable or larger constant factors without these limitations.

Grover-like Search on Structured Spaces

This comparison is particularly important. Grover's algorithm provides √N speedup for unstructured search—but DCPC can exceed this when:

- The oracle returns structured constraint information (not just Boolean marking)
- The search space collapses before full enumeration is required

The prefix-constraint oracle example (§9.4) demonstrates O(log N) recovery versus O(√N) Grover. This is not a defeat of quantum computing—it reflects a stronger oracle model that DCPC can exploit natively.

11.4.3 Where Quantum Computing Retains Unique Advantage

For intellectual honesty and reviewer credibility, we emphasize the domains where quantum computing provides advantages DCPC cannot match:

Problem Class	Winner	Mechanism
Integer factorization (Shor)	Quantum	Period-finding is intrinsically quantum
Discrete logarithm	Quantum	Same period-finding primitive
Quantum chemistry simulation	Quantum	State space is natively Hilbert space
Unstructured search	Quantum (Grover)	DCPC has no leverage without structure

These represent approximately 3–5% of practical computational workloads—important problems, but a narrow slice.

11.4.4 Summary Assessment

Workload Class	Share	DCPC vs Quantum
Classical-equivalent	~70–75%	Both offer ~1× vs classical; DCPC simpler to deploy
Constraint-dominated	~20–25%	**DCPC typically matches or exceeds quantum**
Quantum-advantaged	~3–5%	Quantum wins (period-finding, simulation)

Bottom line: On the constraint-dominated workloads where DCPC provides large constant-factor gains, DCPC often matches or outperforms quantum approaches in practice, while avoiding the encoding complexity, noise sensitivity, and scalability barriers of near-term quantum hardware. Moreover, DCPC operates at ambient temperature with no cryogenic cooling or quantum error correction, dramatically simplifying deployment and energy costs (see §12.4). DCPC loses only where quantum computing is uniquely strong—which is exactly the complementary relationship one would want from a practical computing architecture.

11.5 Anticipating Reviewer Questions and Scope Clarifications

This section addresses likely points of skepticism regarding novelty, performance claims, and relationship to existing constraint-solving techniques. It is intended to clarify scope and prevent misinterpretation of the paper's claims.

11.5.1 Is DCPC/RAL a New Algorithm or a New Architecture?

DCPC/RAL should not be read as proposing a new inference rule that supersedes existing constraint-propagation algorithms such as arc consistency, belief propagation, survey propagation, or modern CDCL heuristics. Those techniques already embody decades of refinement and are highly effective in software.

The novelty claim of this work lies instead in the **computational regime and architectural substrate**, not in the logical content of the inference rules themselves.

Specifically:

- Existing solvers execute constraint propagation on fact-native machines, where partial assignments are represented as committed Boolean artifacts and must be explicitly undone via backtracking.
- DCPC proposes a pre-fact computational regime, in which constraint status is represented as a persistent marker field and updated via rewrite dynamics that do not require rollback.
- What CDCL emulates in software through trails, implication graphs, clause learning, and restarts, DCPC aims to express directly in the state representation and update physics of the machine.

In this sense, the relationship between DCPC and CDCL is analogous to the relationship between scalar code emulating vector operations and hardware with native SIMD or tensor execution.
The inference is not new; the cost model is.

11.5.2 Are the Performance Claims (10×–1000×) Speculative?

Yes — and intentionally so.

This paper is an architectural proposal, not a benchmark report. The stated speedup range reflects:

- Elimination of backtracking and re-propagation overhead
- Removal of trail management and implication graph construction
- Bulk parallel rewrite updates on a marker field

However, the magnitude of realized speedup depends on:

- Hardware realization (FPGA vs ASIC vs near-memory)
- Memory locality and bandwidth
- Problem structure (industrial vs random vs adversarial)
- How much runtime in classical solvers is spent on rollback-driven control flow

Accordingly:

- The lower end (~10×) corresponds to gains comparable to removing major sources of cache misses, branch misprediction, and redundant propagation.
- The upper end (~100–1000×) applies only to thrashing-heavy, highly structured instances where classical solvers repeatedly revisit near-identical partial assignments.

We explicitly **do not** claim asymptotic improvement, nor do we claim dominance on all instances. The paper's claim is that constant-factor gains of this magnitude are **plausible** given a hardware-native implementation, not that they are guaranteed.

Empirical validation against SAT Competition benchmarks (e.g., Kissat, CaDiCaL) is a necessary next step. We explicitly include a benchmark-free evaluation protocol (§12.5) so the model can be tested immediately via simulation without requiring proprietary datasets or specialized hardware. Preliminary toy simulation results (§12.6) confirm that marker dynamics converge correctly on structured instances and fail gracefully (oscillation/stall) on unstructured instances, consistent with the paper's central claims.

11.5.3 Is "No Commitment Until Measurement" Merely Semantic?

The distinction between pre-fact and fact regimes is not epistemic ("the answer is unknown") but **operational**.

A Boolean assignment in classical solvers is a fact in the computational semantics:

- It conditions control flow
- Triggers implication graph construction
- Requires explicit rollback if inconsistent

A ✓ marker in DCPC denotes local stability in the pre-fact field, not irreversible commitment:

- It can be overwritten or destabilized without undo logic
- It does not induce trail-conditioned branching
- It does not incur the costs associated with fact semantics

Measurement is therefore not "discovering" the solution, but **accepting irreversible commitment and exporting a Boolean artifact**. The computational work occurs during pre-fact convergence; measurement is a controlled, one-way boundary crossing.

11.5.4 Does the Taylor Limit Add Real Constraints?

Within this paper, the Taylor Limit should be read as an **admissibility postulate**, not as a derived physical constant.

Its purpose is to:

- Formalize the idea that pre-fact refinement cannot proceed indefinitely
- Justify bounded oscillation and termination in marker dynamics
- Connect convergence guarantees to finite distinguishability rather than idealized infinite precision

We acknowledge that similar halting guarantees could be derived purely from finite-state arguments. The Taylor Limit provides a bridge to physical interpretations (entropy, distinguishability, holographic bounds), but the DCPC model does not depend on Planck-scale physics to function.

Readers interested primarily in computational systems may treat the Taylor Limit as a compact way to express bounded resolution; readers interested in physical grounding may view it as part of a broader interpretive framework.

11.5.5 Why Compare to Grover at All?

Grover's algorithm is referenced not as a performance target but as a **boundary marker**:

- Grover optimality applies to Boolean-marking oracles.
- DCPC exploits structured oracles whose outputs encode partial constraint information.

The prefix-constraint example demonstrates that **oracle structure, not quantum superposition**, is the decisive variable in query complexity. We do not claim that DCPC "beats Grover" in the same oracle model, only that structured pre-fact oracles lie outside Grover's lower bound.

This framing clarifies when quantum speedups are intrinsic and when they are artifacts of oracle choice.

11.5.6 Would DCPC Hardware Actually Beat State-of-the-Art Solvers?

This is ultimately an **empirical question**.

The central hypothesis of this paper is: *If a substantial fraction of solver runtime is spent managing fact semantics (rollback, re-propagation, control flow), then a hardware substrate that removes those semantics can achieve large constant-factor gains.*

DCPC does not claim to dominate optimized CDCL solvers on all instances. It predicts:

- ~1× performance on incompressible or adversarial cases
- Significant gains on structured industrial and thrashing-heavy instances
- No advantage where problem structure offers no leverage

This hypothesis is **falsifiable and testable**. The paper's goal is to define the model, motivate the architecture, and delineate where such gains should and should not appear.

11.5.7 Complexity-Theoretic Implications

A natural question arises: if DCPC provides 10–1000× speedups on NP-complete problems, doesn't this have implications for computational complexity theory?

No. DCPC provides:

- Large **constant-factor** improvements on **structured instances**
- Overhead elimination, not algorithmic breakthrough
- Practical acceleration, not asymptotic improvement

DCPC does **not** provide:

- Polynomial-time solutions to NP-complete problems
- Speedups on worst-case or adversarially constructed instances
- Any change to asymptotic complexity classes

Complexity classes are defined by asymptotic scaling as $n \to \infty$. A constant-factor speedup—even 1000×—does not change the class: $O(2^n) \to O(2^n/1000) = O(2^n)$. This parallels modern SAT solvers: they solve industrial instances with millions of variables in seconds while adversarial instances remain intractable—but this does not imply P = NP.

11.5.8 Scope Summary

To avoid misinterpretation, we summarize the scope explicitly:

DCPC is NOT	DCPC IS
A new complexity class	A new computational regime
A replacement for quantum computing	Complementary to quantum for different problem classes
A threat to cryptography	Cryptographically neutral
Superior on all instances	Targeted at constraint-dominated structured problems
A proven benchmark result	An architectural proposal with testable predictions

Bottom line: DCPC is a proposal for a hardware-native pre-fact computation regime that targets a well-defined class of constraint-dominated problems, offering potential large constant-factor gains with substantially lower deployment and energy barriers than quantum hardware.

11.6 The Empirical Crux: Will DCPC Beat Kissat/CaDiCaL on Industrial SAT?

For general readers: The most important question is not whether marker dynamics can solve SAT in principle (they can), but whether a DCPC-native implementation would outperform today's best SAT solvers on real industrial problems. This section defines the conditions under which that would be true and how to test it.

11.6.1 What the Toy Simulations Do—and Do Not—Establish

The toy simulations in §12.6 establish behavioral plausibility: marker dynamics can converge on compressible structure and stall on incompressible structure, consistent with the paper's scope. They do **not** establish performance superiority over highly optimized CDCL solvers.

We acknowledge the reviewer's point: modern solvers such as Kissat and CaDiCaL already incorporate decades of engineering that partially mitigate the costs of fact-native semantics (watched literals, VSIDS, clause forgetting, restarts, cache-conscious data structures). The remaining question is whether DCPC removes a large enough irreducible overhead to matter.

11.6.2 A Break-Even Cost Model

Let a modern CDCL solver's wall time on an instance be decomposed as:

$$T_{CDCL} = T_{BCP} + T_{conf} + T_{learn} + T_{trail} + T_{restart} + T_{mem}$$

Where:

- **T_{BCP}**: Boolean constraint propagation (watched literals, unit propagation)
- **T_{conf}**: Conflict analysis and UIP derivation
- **T_{learn}**: Clause database insertion/maintenance/forgetting
- **T_{trail}**: Assignment trail updates, rollback, decision-level bookkeeping
- **$T_{restart}$**: Restart logic, re-propagation cost after restart
- **T_{mem}**: Memory hierarchy penalties (cache misses, pointer chasing, branch mispredicts)

A DCPC-native substrate does not "remove SAT hardness," but it aims to replace the trail-rollback-repropagation loop with in-place marker evolution, reducing:

- T_{trail} (rollback semantics)
- Large portions of $T_{restart}$ (re-propagation after undo)
- Potentially T_{mem} (if implemented as dense, local, systolic updates)

Let the DCPC runtime be approximated as:

$$T_{DCPC} \approx T_{rewrite} + T_{reduce} + T_{measure}$$

Where:

- **$T_{rewrite}$**: Marker rewrite passes over constraints
- **T_{reduce}**: Global reductions for diffusion/coherence metrics
- **$T_{measure}$**: Final threshold check and assignment extraction

Break-even condition. DCPC beats CDCL on an instance if:

$$T_{rewrite} + T_{reduce} + T_{measure} < T_{CDCL}$$

More usefully, if DCPC eliminates a fraction f of CDCL overhead and incurs a new overhead fraction g (global reductions, diffusion, marker management), then:

$T_{DCPC} \approx (1 - f) \cdot T_{CDCL} + g \cdot T_{CDCL}$

So **DCPC wins when f > g**.

This reframes the question in falsifiable operational terms: is the recoverable overhead f larger than the introduced overhead g?

11.6.3 Where We Expect f to Be Large (and Where We Do Not)

We expect DCPC to be most competitive on industrial instances where the CDCL solver exhibits:

- High backtrack counts per solved instance
- Frequent restarts with significant re-propagation cost
- Heavy clause database churn
- Thrashing regimes (large search neighborhoods revisited under minor variations)
- Memory-bound propagation behavior (poor locality dominates)

We do **not** expect DCPC to dominate on instances where:

- CDCL solves quickly with low backtracking
- Propagation locality is already excellent
- The instance is essentially "easy under heuristics"

This aligns with our workload split: DCPC's expected advantage is not universal; it is concentrated in the subset of instances where fact-native control flow dominates runtime.

11.6.4 Concrete Benchmark Plan Against Kissat/CaDiCaL

To settle this question empirically, we propose the following evaluation:

Benchmarks:

- SAT Competition industrial tracks (multiple years)
- Hardware verification encodings (bounded model checking instances)
- Configuration/scheduling CNFs where structure is strong

Baselines:

- Kissat and CaDiCaL with standard configurations
- Optionally Glucose and MapleSAT variants for robustness

Metrics (report both wall time and work-normalized counters):

- Solved/unsolved rate within fixed timeouts
- Time-to-solution distribution (median, tail)
- Number of unit propagations / second
- Number of backtracks and restarts
- Clause database size over time
- Memory bandwidth and cache-miss rate (via perf counters)

Hypothesis: DCPC should show advantage precisely when CDCL runs are dominated by backtracking + restart re-propagation + memory stalls. If DCPC does not improve those regimes, the hypothesis fails.

11.6.5 What Would Falsify the DCPC Advantage Claim?

We make the falsification criterion explicit:

Falsification: If on industrial SAT instances Kissat/CaDiCaL consistently outperform a software DCPC simulator by large margins even when both are constrained to comparable memory locality and without specialized hardware, then the primary advantage may already be captured by solver engineering, and DCPC's architectural premise would require revision.

Validation: If a DCPC simulator shows systematically fewer "propagation-equivalent" passes to reach a satisfying assignment on structured instances, then a hardware-native DCPC accelerator has a credible path to outperforming CDCL by removing memory and control-flow overhead.

11.6.6 Summary

We acknowledge that the core uncertainty is empirical: will DCPC-native hardware outperform best-in-class CDCL on industrial SAT? This paper does not claim this is already proven; it defines the architecture, identifies the mechanism by which an advantage could arise, and specifies the conditions and benchmarks that would validate or falsify the claim.

11.7 Provable Advantages (What Can and Cannot Be Proven)

For general readers: While we cannot prove DCPC beats all solvers on all problems, we can prove certain architectural advantages rigorously. This section presents formal results that hold independent of benchmark outcomes.

What can and cannot be proven. We do not claim a universal theorem that DCPC hardware dominates all CDCL solvers (e.g., Kissat/CaDiCaL) on all industrial instances; this is an empirical question because solver heuristics and microarchitecture matter. However, DCPC admits (i) profiling-based lower bounds on achievable speedup once the fraction of time spent in rollback/re-propagation overhead is measured, and (ii) formal depth/locality advantages in parallel execution models where marker rewrites are local and global reductions are tree-aggregated. These constitute provable architectural advantages independent of benchmark outcomes.

11.7.1 Profiling-Based Speedup Bound (Amdahl-Style)

Theorem 11.1 (Measured Dominance Bound). Let a baseline CDCL solver spend fraction f of wall time in costs that DCPC eliminates (trail rollback, re-propagation after undo/restart, implication-graph bookkeeping), and let DCPC introduce additional overhead fraction g (global reductions, diffusion bookkeeping). Then:

$$T_{CDCL} / T_{DCPC} \geq 1 / (1 - f + g)$$

Proof. If DCPC eliminates fraction f of CDCL runtime and adds fraction g, then $T_{DCPC} \leq (1 - f) \cdot T_{CDCL} + g \cdot T_{CDCL} = (1 - f + g) \cdot T_{CDCL}$. Rearranging gives the bound. □

Corollary 11.2. This provides a provable lower bound on DCPC speedup once f is measured via profiling.

Measured f	Introduced g	Minimum Speedup
0.50	0.05	≥ 1.8×
0.70	0.05	≥ 2.9×
0.80	0.05	≥ 4.0×
0.90	0.05	≥ 6.7×
0.95	0.05	≥ 10×

What counts toward f (instrumentation targets):

- `backtrack()` and trail unwinding functions
- Re-propagation loops after restart
- Implication graph construction and analysis
- Decision-level bookkeeping and undo operations
- Clause database maintenance triggered by rollback

What counts toward g:

- Global reduction operations (coherence score computation)
- Diffusion/mean-field updates across marker matrix
- Measurement readiness checking

This is a real proof—but it depends on profiling data from actual solver runs.

11.7.2 Parallel Depth Advantage (Circuit/PRAM Model)

Theorem 11.3 (Depth Advantage Under Bounded-Degree Locality). Assume:

1. Constraints form a bounded-degree graph (degree ≤ d)
2. Rewrite rules are local (depend only on neighboring markers)

3. Hardware provides one update unit per variable/constraint edge
4. Global reductions are tree-aggregated

Then one rewrite "tick" has:

- **O(1)** depth for local rewrites
- **O(log n)** depth for global reductions (diffusion/coherence)

Whereas any conventional single-core RAM execution of equivalent propagation requires:

- **Ω(m)** memory touches per tick (at least linear in constraints evaluated)

Proof sketch. Local rewrites depend only on bounded-neighborhood state and can execute in parallel with constant depth. Tree-reduction of n values requires $\log_2(n)$ depth. Sequential RAM must touch each constraint serially, requiring $\Omega(m)$ time for m constraints. □

Corollary 11.4. DCPC has a provable asymptotic advantage in parallel time (circuit depth) for the same logical propagation process. The same constraint propagation dynamics can be executed at fundamentally different time/energy scales depending on substrate.

This is a standard result in architecture analysis (PRAM/BSP/circuit-depth style). It does not say "beats Kissat"—but it proves that marker dynamics admit parallel execution that sequential solvers cannot match.

11.7.3 Provable Advantage on Specific Instance Families

DCPC can be proven superior on instance families where propagation suffices but fact-native search pays a tax:

Theorem 11.5 (Propagation-Complete Families). On instance families where:

1. A solution exists and is reachable via pure constraint propagation (no search), or
2. The constraint graph has bounded treewidth and admits polynomial-time message-passing solutions

DCPC achieves solution with:

- Zero backtracks (by construction)
- O(diameter × log n) parallel depth for convergence

Whereas trail-based CDCL solvers without access to global closure oracles may incur:

- Non-zero backtracks due to arbitrary decision ordering
- Sequential propagation passes proportional to constraint count

Examples of provable families:

- Horn-SAT and 2-SAT (polynomial classically, but DCPC avoids any trail overhead)
- Bounded-treewidth CNFs where dynamic programming / message passing applies
- Propagation-complete encodings from bounded model checking

Caveat: Modern solvers like Kissat can also be fast on these families. The provable advantage is not "solves vs. doesn't solve" but "solves with fewer reversible commitments / fewer trail operations" under a baseline solver model that uses trail-based backtracking without access to global closure oracles.

11.7.4 Summary of Provable Claims

Claim Type	What Is Proven	Depends On
Profiling-based bound	Minimum speedup $\geq 1/(1-f+g)$	Measured f from solver profiling
Parallel depth	$O(\log n)$ vs $\Omega(m)$ per tick	Bounded-degree, local rewrites
Instance families	Zero backtracks on propagation-complete	Restricted comparator class

These results are formal and hold independent of benchmark outcomes. They establish that DCPC's architectural advantages are real and provable under appropriate models, while acknowledging that empirical performance against highly optimized solvers remains the decisive test.

11.8 Maturity Asymmetry and Fair Comparison

For general readers: When comparing a new idea against existing technology, it's important to recognize that the existing technology has had decades of refinement while the new idea is just a proposal. This section addresses that imbalance honestly.

A key interpretive caveat in evaluating DCPC is the asymmetry in technological maturity between the paradigms being compared.

Modern classical SAT solvers (e.g., Kissat, CaDiCaL) represent the outcome of several decades of intensive optimization. Techniques such as watched literals, activity-based branching (VSIDS), aggressive restarts, clause forgetting, cache-aware data layouts, and fine-grained heuristic tuning have been iteratively refined against thousands of benchmark instances. These systems are highly engineered around a fact-native computational model, in which Boolean commitment occurs early and irreversibly.

By contrast, DCPC is an architectural proposal at a pre-hardware stage. The marker dynamics described in this paper are intentionally minimal, emphasizing conceptual clarity over aggressive optimization. No attempt is made to incorporate the full spectrum of domain-specific heuristics, cache-aware layouts, or instance-tuned parameters that characterize state-of-the-art classical solvers.

As a result, comparisons between DCPC-style dynamics and mature CDCL implementations should be understood as comparisons between:

- A conceptual, minimally optimized architecture, and
- A highly mature, heavily engineered software stack.

This asymmetry cuts in two directions:

Caution: It justifies caution in interpreting absolute performance claims prior to empirical validation. A software DCPC simulator cannot fairly compete against decades of CDCL engineering.

Significance: It suggests that any observed advantage from delayed commitment or marker-based propagation—even in simplified form—is unlikely to be an artifact of tuning and more likely to reflect a genuine architectural difference.

Historical precedent. Analogous asymmetries have occurred during the early evaluation of GPUs, TPUs, and systolic accelerators, where early software implementations underperformed optimized CPU code until hardware-native execution exposed the architectural advantages. The transition from "this is slower than optimized C" to "this is orders of magnitude faster on appropriate workloads" required both hardware realization and workload-appropriate deployment.

The central claim of DCPC is therefore not that it immediately dominates the best classical solvers in software, but that it identifies a computational regime—constraint-dominated, backtracking-heavy search—in which fact-native architectures incur unavoidable overhead, and where a pre-fact, marker-native architecture has the potential to scale more favorably once implemented in hardware.

Addressing the hardest objection. A sophisticated reviewer might argue: "But Kissat already does this implicitly—modern solvers have evolved toward minimal-commitment strategies through restarts, clause forgetting, and activity decay."

Our response: Possibly—but only by spending enormous engineering effort managing the consequences of early commitment. Watched literals exist because assignments must be tracked. Restarts exist because bad early commitments must be escaped. Clause forgetting exists because learned clauses from rolled-back search paths accumulate. DCPC asks: *what happens if we remove the burden of fact-native semantics entirely?* That is the architectural question this paper poses.

---

12. Implementation Considerations

For general readers: Can we actually build this? Yes—and on multiple types of hardware. This section sketches how DCPC could be implemented on custom chips, reconfigurable hardware,

optical systems, or brain-inspired computers. The marker operations are simple enough that specialized hardware could execute them very efficiently.

12.1 Hardware Substrates

RAL marker dynamics can be implemented on several substrates:

Digital ASIC: Marker states encoded as small integers (3 bits for 7-symbol alphabet); rewrite rules as lookup tables; global operations via systolic arrays.

FPGA: Reconfigurable fabric for problem-specific rewrite rule optimization; natural fit for parallel marker updates.

Optical: Marker states as spatial modes; interference for incompatibility detection; threshold nonlinearity for measurement.

Neuromorphic: Markers as neuron activation patterns; rewrite rules as synaptic update rules; measurement as winner-take-all dynamics.

12.2 Complexity of Global Operations

Global constraint propagation requires communication across the marker matrix. For matrix $M \in \Sigma^{n \times m}$:

- **Local rewrites**: O(nm) parallel, O(1) depth
- **Global diffusion**: O(n) for mean computation, O(nm) for update
- **Measurement check**: O(nm) for criterion evaluation

Total circuit depth per iteration: O(1) with sufficient parallelism.

12.3 Engineering Challenges and Near-Term Feasibility

For general readers: This section explains what makes DCPC hard to build in practice—and why it's still realistic with today's engineering. The main challenge is not the math; it's building hardware that can update and compare many "possibility markers" in parallel without wasting energy moving data around.

Although DCPC/RAL is classical in principle, its performance advantage depends on a highly parallel physical implementation. The primary engineering challenge is therefore not correctness but bandwidth, locality, and energy efficiency under massive parallel marker updates.

12.3.1 The Data-Movement Bottleneck

Modern computing is often limited not by arithmetic but by moving data (the "memory wall"). DCPC intensifies this pressure because:

- Marker rewrite rules are applied across a large matrix $M \in \Sigma^{n \times m}$
- Many updates depend on neighborhood or constraint-graph relations
- Coherence/measurement requires aggregated statistics (counts of symbols, coherence score)

If implemented naïvely on CPUs/GPUs, DCPC would lose much of its advantage to memory bandwidth and synchronization overhead. The architecture therefore benefits most from hardware that places marker-state memory physically close to the rewrite operators.

12.3.2 Parallelism and Locality Requirements

DCPC becomes attractive when marker updates are:

- **Massively parallel**: SIMD/SIMT-like across matrix cells
- **Local or sparse-global**: Constraint graph adjacency rather than dense all-to-all
- **Pipelined**: Streaming rewrite passes, not global barriers

This resembles requirements of cellular automata accelerators, systolic arrays (TPU-style), sparse graph neural network accelerators, and SAT propagation kernels optimized for locality.

12.3.3 Rewrite Rules as Hardware

RAL markers use a small alphabet (e.g., 3 bits for 7 symbols), so rewrite rules can be implemented efficiently as:

- **Lookup tables (LUTs)** on FPGA/ASIC
- **Bit-sliced logic**: Fast boolean operations over packed markers
- **Microcoded rewrite kernels** for reconfigurable rule sets

The most practical near-term design is a "marker ALU" that loads a local neighborhood of markers, applies a rule LUT, writes back updated markers, and optionally accumulates counters for coherence metrics.

12.3.4 Global Operations: Diffusion and Coherence Aggregation

Two operations introduce genuine global coupling:

- Diffusion-like rebalancing (mean reinforcement comparison)
- Measurement readiness checks (counts of marker states, coherence score)

Engineering approach: Compute global summaries via tree reductions (on-chip reduction networks), perform diffusion in two passes (compute mean $\bar{r}$, then update cells), and amortize readiness checks by evaluating every k iterations rather than every cycle.

12.3.5 Noise, Metastability, and Oscillation Management

In practical implementations, especially analog or neuromorphic variants, marker states may exhibit noise-induced flips, metastable oscillation, or local cycles. DCPC explicitly anticipates this by treating bounded oscillation as a valid pre-fact outcome. Engineering-wise, the system should include:

- Explicit oscillation counters
- Damping rules ($\rightleftharpoons^n \rightarrow \downarrow$)
- Restart or annealing schedules (periodic noise injection or threshold schedules)

This resembles techniques used in simulated annealing, loopy belief propagation, and SAT solver restart policies.

12.3.6 Candidate Near-Term Hardware Paths

(A) FPGA Prototype (fastest realistic path)

- Implement markers as packed bitfields
- Implement rewrite rules as LUTs
- Represent constraints as sparse adjacency lists in BRAM/HBM
- *Ideal for proving constant-factor gains on industrial SAT/CSP instances*

(B) ASIC / Systolic Array Accelerator (performance path)

- Marker SRAM distributed across compute tiles
- Local rewrite engines per tile
- Reduction network for diffusion/measurement
- *Resembles TPU-like architectures but for marker dynamics*

(C) Near-Memory / In-Memory Compute

- Place rewrite operators close to HBM stacks or compute-in-memory arrays
- Reduces data movement cost
- *Aligns strongly with the DCPC locality requirement*

(D) Optical / Analog Variants (longer-term)

- Potentially lower energy for similarity/interference-like checks
- Higher difficulty for programmability and precision control
- *Promising but not required for DCPC feasibility*

12.3.7 Feasibility Summary

CAN DCPC BE BUILT WITH TODAY'S TECHNOLOGY?

Short answer: Yes.

Question	Answer
Does it require new physics?	**No.** DCPC is entirely classical.
Does it require quantum hardware?	**No.** No qubits, no cryogenics, no decoherence problems.
Does it require materials we don't have?	**No.** Standard CMOS, FPGA fabric, or existing accelerator architectures suffice.
What's the main engineering challenge?	**Data movement.** Keeping marker memory close to compute units.
Is this challenge solved?	**Largely, yes.** TPUs, systolic arrays, and near-memory compute already address similar problems.
What's a realistic first step?	**FPGA prototype** running SAT/CSP benchmarks, demonstrating constant-factor speedups.
What would full deployment look like?	**ASIC accelerator** or **near-memory compute** chips, acting as constraint-propagation coprocessors.

Bottom line: DCPC does not require any technology that doesn't already exist. The engineering is challenging but well within the envelope of current accelerator design. An FPGA proof-of-concept could be built today; a production ASIC could follow standard accelerator development cycles.

DCPC's near-term feasibility claim is not that it provides a new complexity class, but that it can deliver large constant-factor speedups on structured problems by eliminating backtracking control-flow overhead, maintaining correlations in a compact marker representation, and using hardware-native parallel rewrite dynamics.

12.4 Energy, Cooling, and Operational Constraints

For general readers: Many discussions of advanced computing focus on raw speed, but in practice the biggest barriers are often energy use, cooling, and operational complexity. A computer that is theoretically fast but requires extreme physical conditions can be impractical to deploy at scale. This section explains why DCPC has a fundamental advantage in this regard.

12.4.1 Energy and Cooling Requirements

DCPC operates entirely within the domain of classical hardware and therefore avoids the dominant energy and cooling constraints associated with current quantum computing platforms. In particular, DCPC:

- Requires no cryogenic cooling
- Requires no coherence preservation

- Requires no quantum error correction
- Operates at ambient temperatures using standard CMOS-compatible technologies

By contrast, most scalable quantum computing platforms rely on dilution refrigerators operating at temperatures on the order of 10–20 millikelvin, along with multi-stage cryogenic infrastructure and extensive room-temperature control electronics. The energy cost of maintaining these environments typically dominates the total system power budget, often exceeding the energy used for the computation itself.

DCPC avoids this entirely. Marker dynamics are implemented using classical logic and memory, with energy consumption dominated by memory access, interconnect, and local rewrite operations—placing DCPC firmly within the same operational envelope as modern accelerators (GPUs, TPUs, near-memory compute devices).

12.4.2 Noise Tolerance and Error Management

Quantum systems must actively suppress noise and decoherence, expending substantial energy and hardware overhead to preserve fragile quantum states. Error correction further amplifies this cost, often requiring hundreds to thousands of physical qubits per logical qubit in fault-tolerant designs.

DCPC takes the opposite approach. Noise and bounded oscillation are explicitly modeled as part of the pre-fact regime:

- Oscillation (⇌ markers) is treated as a legitimate intermediate state
- Damping and stabilization are handled algorithmically rather than physically
- No attempt is made to preserve fragile states against environmental interaction

As a result, DCPC expends energy only on useful computational work rather than on maintaining artificial physical isolation. This design choice significantly simplifies hardware requirements and improves energy proportionality.

12.4.3 Energy Efficiency per Solved Instance

The relevant metric for practical computing is not energy per elementary operation, but energy per successfully solved instance. For the class of constraint-dominated workloads where DCPC provides large constant-factor speedups (~20–25% of practical workloads), DCPC achieves additional energy efficiency by:

- Eliminating repeated backtracking
- Avoiding redundant propagation and control-flow churn
- Preserving correlations until resolution rather than recomputing them

Even when quantum algorithms offer asymptotic advantages on paper, the total energy cost per solved instance can be dominated by encoding overhead, control complexity, and cryogenic

infrastructure. DCPC's classical substrate allows it to deliver competitive or superior energy efficiency on many real-world problem instances, particularly at industrial scales.

12.4.4 Deployment and Scalability Implications

Because DCPC does not require exotic physical conditions, it can be:

- Deployed in standard data centers
- Integrated as an accelerator alongside CPUs and GPUs
- Scaled using existing semiconductor manufacturing and packaging technologies
- Operated under conventional power, cooling, and reliability constraints

This sharply contrasts with quantum computing, which remains constrained to specialized facilities and research environments for the foreseeable future.

12.4.5 Positioning Relative to Quantum Computing

This section does not argue that DCPC replaces quantum computing in domains where genuine Hilbert-space compression is essential (e.g., quantum simulation, period-finding). Rather, it highlights an orthogonal and complementary advantage:

DCPC trades narrow asymptotic gains for broad deployability, energy efficiency, and operational simplicity.

For the substantial fraction of real-world workloads dominated by constraints rather than algebraic structure, these operational considerations are often decisive.

12.5 Software Simulator and Evaluation Protocol

For general readers: To test whether DCPC/RAL dynamics work as claimed, we can simulate the marker updates in software before building hardware. This does not prove hardware speedups, but it does test correctness, convergence behavior, and where the approach breaks down.

12.5.1 Objective

We propose a software simulator that implements RAL marker dynamics on SAT/CSP instances. The simulator evaluates three questions:

1. **Correctness:** Does the marker field converge to an assignment that satisfies the CNF?
2. **Convergence dynamics:** Does the system stabilize smoothly, oscillate, or stall?
3. **Constant-factor profile:** How many "propagation-equivalent updates" occur before solution compared to CDCL solvers?

Even without hardware acceleration, this simulator provides a grounded test of the model's practical behavior and identifies regimes where DCPC is expected to help or not help.

12.5.2 Minimal Simulator Specification

The simulator maintains a pre-fact state for each variable x_i as a pair of marker channels:

- $x_i = 0$: marker status in Σ (or score in [0,1])
- $x_i = 1$: marker status in Σ (or score in [0,1])

Each iteration applies:

1. **Clause danger evaluation:** Identify clauses near violation under the current marker field
2. **Rewrite updates:** Propagate suppression/reinforcement markers to literals participating in dangerous clauses
3. **Damping / oscillation control:** Apply $\rightleftharpoons^n \rightarrow \downarrow$ and optional restart/annealing schedules
4. **Measurement readiness check:** Apply Definition 7.2 (coherence + oscillation + readiness threshold)

A SAT instance is declared solved when the extracted Boolean assignment satisfies the CNF.

12.5.3 Synthetic Benchmark Suite

To enable immediate testing without dependence on external datasets, the initial simulator evaluation uses a synthetic suite spanning compressible and incompressible regimes:

(A) Planted satisfiable 3-SAT (compressible / structured) Generate CNFs with a hidden satisfying assignment. *Expected outcome:* Rapid convergence; high success rate; low oscillation.

(B) Random 3-SAT near phase transition (hard / incompressible) Generate uniform random clauses at clause-to-variable ratios known to be difficult (~4.26). *Expected outcome:* Frequent oscillation or stalling; low solve rate; serves as a "no-miracle" control.

(C) Pigeonhole (structured UNSAT) Canonical UNSAT family encoding n+1 pigeons into n holes. *Expected outcome:* Bounded oscillation or termination without fact creation; useful for testing stability criteria.

(D) Structured constraint problems (graph coloring, scheduling encodings) Industrial-like structure without proprietary benchmarks. *Expected outcome:* Intermediate regime; identifies where DCPC offers the most leverage.

This suite is sufficient to validate the core claims: fast convergence on compressible structure, bounded oscillation on hard cases, and no asymptotic miracle.

12.5.4 Comparison Metrics

To compare against CDCL solvers (MiniSat, Glucose, Kissat, CaDiCaL) using identical CNFs, we recommend reporting both:

Implementation-dependent:

- Wall-clock time

Implementation-independent (normalized work metrics):

- Number of clause evaluations
- Number of literal updates
- Number of restart events
- Number of propagation-equivalent passes

This cleanly separates algorithmic convergence properties from hardware acceleration, enabling fair comparison even without dedicated DCPC hardware.

12.5.5 Falsifiable Predictions

The simulator is expected to validate the following falsifiable hypotheses:

Instance Type	Prediction	Rationale
Planted / structured SAT	High solve rate; rapid stabilization	Compressible constraint structure
Industrial-like encodings	Moderate-to-high solve rate; fewer restarts	Global propagation reduces rollback
Random SAT near threshold	Low solve rate; oscillation/stall	No structure to compress
Structured UNSAT (pigeonhole)	Bounded oscillation / termination	Finite-state + damping

Falsification criteria: If the simulator does not converge even on planted instances, or if oscillation exceeds bounds on all instance types, then the proposed marker dynamics require revision. If these hypotheses hold, the architecture is empirically grounded and hardware acceleration becomes the appropriate next step.

Note: We explicitly include this benchmark-free evaluation protocol so the model can be tested immediately via simulation without requiring proprietary datasets or specialized hardware.

12.6 Preliminary Software Evidence (Toy Simulation Results)

For general readers: Before building hardware or running large benchmark suites, we implemented a minimal software simulator to test whether RAL-style marker dynamics behave as intended on representative toy problems. These experiments are not performance benchmarks; they are sanity checks on convergence behavior.

12.6.1 Scope and Limitations

The simulator implements a mean-field approximation of RAL marker dynamics:

- Clause pressure propagates as reinforcement/suppression signals
- Oscillation damping is applied
- Measurement readiness is defined by stability and clause satisfaction rather than full Boolean saturation

No clause learning, watched literals, or low-level optimizations were included. As such, **no runtime comparisons against optimized CDCL solvers are claimed here**. The purpose is purely to observe qualitative behavior.

12.6.2 Toy 2-SAT Instance (Worked Example Validation)

We first tested the exact 2-SAT instance used as the worked example in §5.1:

$(x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y)$

Observed behavior:

- The marker field rapidly developed consistent directional pressure toward x=1, y=1
- Suppressed alternatives remained nonzero but stably disfavored
- The extracted Boolean assignment satisfied all clauses
- Full saturation to hard Boolean values was not required for correctness

Interpretation: This confirms that DCPC-style dynamics do usefully distinguish "favored" from "disfavored" assignments prior to commitment, and that measurement can be defined as stable satisfaction plus coherence, not literal collapse to bits.

12.6.3 Planted Satisfiable 3-SAT (Compressible Structure)

We next tested small planted-solution 3-SAT instances (random clauses generated around a hidden satisfying assignment).

Observed behavior:

- The marker field converged rapidly toward a satisfying assignment
- Convergence typically occurred in very few global update iterations
- No oscillatory instability was observed in these cases

Interpretation: This regime corresponds to structured, compressible instances—precisely where DCPC claims large constant-factor gains. Even a minimal implementation exhibits fast convergence.

12.6.4 Random 3-SAT Near Phase Transition (Incompressible Structure)

Finally, we tested random 3-SAT instances near the satisfiability threshold (~4.26 clause-to-variable ratio).

Observed behavior:

- Marker dynamics frequently oscillated or stalled
- No reliable convergence was observed within reasonable iteration limits
- Measurement readiness was not achieved

Interpretation: This is an expected and important **negative result**. It confirms that DCPC does not magically solve unstructured or adversarial instances. Without exploitable structure, delayed commitment alone is insufficient—consistent with both classical and quantum complexity theory.

12.6.5 Summary of Observed Regimes

Instance Type	Outcome	Interpretation
Small structured 2-SAT	Stable convergence	Validates worked example
Planted 3-SAT	Rapid convergence	Compressible structure exploited
Random 3-SAT (threshold)	Oscillation / stall	No asymptotic miracle

These results support the central claim of the paper: **DCPC provides advantage by eliminating wasted work on structured problems, not by defeating worst-case complexity.**

12.6.6 What These Results Do Not Claim

We emphasize what is **not** claimed:

- No hardware speedup is demonstrated
- No SAT Competition benchmarks are reported
- No claim is made that DCPC outperforms optimized solvers on all instances

Rather, these toy results establish that:

- The marker dynamics behave coherently
- Convergence aligns with intuitive notions of problem structure
- Failure modes match theoretical expectations

This justifies further investigation via full software benchmarks and hardware prototypes.

---

13. Problem Taxonomy and Computational Suitability

For general readers: This section is a practical guide—a "which computer should I use?" reference. We categorize the major types of problems computers are asked to solve and assess which approach (classical, DCPC, or quantum) works best for each. The bottom line: DCPC handles about 95% of all solvable problems at least as well as classical computers, and dramatically better on about 20-25% of them. Quantum computers shine on a small but important slice (~3-5%)—mainly simulating physics and breaking certain codes.

To clarify where DCPC provides advantage, we systematically compare classical, quantum, and DCPC/RAL approaches across the major categories of computational problems.

13.1 Search and Optimization

What this means: Finding the best solution among many possibilities—like the shortest route, the best schedule, or the optimal design.

Problem Type	Classical	Quantum	DCPC/RAL
Unstructured search	O(N)	O(√N) Grover	O(N) — no advantage
Structured search (SAT, CSP)	Exponential worst-case, heuristics help	Limited advantage	**10–1000× via overhead elimination**
Combinatorial optimization	Branch-and-bound, local search	QAOA, annealing (modest)	**Native constraint propagation**
Continuous optimization	Gradient descent	Limited applicability	Moderate (discretization required)
Convex optimization	Polynomial (interior point)	No advantage	No advantage

Assessment. DCPC excels on structured discrete optimization where constraint propagation eliminates backtracking. Quantum provides modest speedup on unstructured search. Classical methods remain optimal for convex and well-conditioned continuous problems.

13.2 Cryptography and Number Theory

What this means: Securing communications and breaking codes. This is where quantum computers pose a genuine threat to current security systems.

Problem Type	Classical	Quantum	DCPC/RAL
Integer factorization	Sub-exponential (NFS)	**Polynomial (Shor)**	Sub-exponential — no improvement
Discrete logarithm	Sub-exponential	**Polynomial (Shor)**	Sub-exponential — no improvement
Symmetric key search	$O(2^n)$	O(2^(n/2)) Grover	$O(2^n)$ — no improvement
Hash collision	O(2^(n/2))	O(2^(n/3))	O(2^(n/2)) — no improvement

Problem Type	Classical	Quantum	DCPC/RAL
Lattice problems	Exponential	Unknown advantage	Unknown — likely no advantage

Assessment. DCPC provides **no cryptographic threat**. Quantum's advantage here stems from Hilbert space compression enabling period-finding (Shor) and amplitude amplification (Grover). DCPC cannot replicate these without the physical substrate.

In plain terms: If you're worried about someone breaking your encryption, worry about quantum computers—not DCPC. Our approach doesn't help attackers.

13.3 Simulation and Modeling

What this means: Using computers to predict how physical systems behave—weather, molecules, airplane wings, etc.

Problem Type	Classical	Quantum	DCPC/RAL
Quantum many-body	Exponential	**Native/polynomial**	Exponential — no advantage
Molecular dynamics	Polynomial	Limited advantage	Polynomial — comparable
Fluid dynamics (CFD)	Polynomial (mesh-dependent)	Uncertain	Polynomial — comparable
Monte Carlo sampling	Polynomial (mixing time)	Quadratic speedup possible	**Structure-dependent speedup**
Constraint-based simulation	Exponential blowup possible	Limited applicability	**Native constraint handling**

Assessment. Quantum simulation of quantum systems is a clear quantum advantage domain. DCPC advantages appear in constraint-heavy simulations (e.g., rigid body dynamics, geometric constraints) where classical methods suffer from constraint enforcement overhead.

13.4 Machine Learning and Data Analysis

What this means: Teaching computers to recognize patterns, make predictions, and learn from data—the technology behind voice assistants, recommendation systems, and image recognition.

Problem Type	Classical	Quantum	DCPC/RAL
Neural network training	Polynomial (SGD)	Limited/unclear advantage	No significant advantage
Kernel methods	$O(n^3)$	Potential speedup (HHL)	No significant advantage

Problem Type	Classical	Quantum	DCPC/RAL
Dimensionality reduction	Polynomial (SVD)	Potential speedup	No significant advantage
Clustering	Polynomial	Modest speedup possible	**Constraint-based clustering improved**
Bayesian inference	Exponential in general	Quadratic sampling speedup	**Structure-dependent improvement**
Constraint learning	Problem-dependent	Limited applicability	**Native constraint representation**

Assessment. Most ML workloads are well-served by classical hardware (especially GPUs). DCPC advantages emerge in constraint-satisfaction aspects of learning: structured prediction, constraint-based clustering, and probabilistic inference with hard constraints.

13.5 Graph and Network Problems

What this means: Problems about connections—social networks, road systems, computer networks, supply chains. "What's the shortest path?" "How should we partition this network?" "Which nodes are most influential?"

Problem Type	Classical	Quantum	DCPC/RAL
Shortest path	Polynomial (Dijkstra)	Limited advantage	No significant advantage
Maximum flow	Polynomial	Limited advantage	No significant advantage
Graph coloring	NP-complete	Modest speedup	**Constraint propagation advantage**
Clique/independent set	NP-complete	Modest speedup	**Constraint propagation advantage**
Subgraph isomorphism	NP-complete	Unknown	**Structure-dependent advantage**
Network reliability	#P-complete	Unknown	**Constraint-based modeling**

Assessment. Polynomial-time graph algorithms see no improvement from DCPC or quantum. NP-complete graph problems with structural regularity (planarity, bounded degree, hierarchical structure) benefit from DCPC's constraint propagation.

13.6 Verification and Formal Methods

What this means: Proving that software and hardware do what they're supposed to—critical for aircraft, medical devices, and financial systems where bugs can be catastrophic.

Problem Type	Classical	Quantum	DCPC/RAL
Model checking	Exponential (state explosion)	Limited applicability	**Delayed commitment reduces explosion**
SAT/SMT solving	Exponential worst-case	Limited advantage	**Native CDCL execution**
Theorem proving	Undecidable in general	No advantage	**Constraint-guided search**
Program synthesis	Exponential search	Unknown	**Constraint satisfaction native**
Type inference	Polynomial to NP-complete	No advantage	**Unification as marker dynamics**

Assessment. Verification suffers from state-space explosion—precisely the regime where delayed commitment provides maximum benefit. DCPC's native constraint propagation aligns well with SMT solving and bounded model checking.

13.7 Scientific Computing and Numerical Methods

What this means: The mathematical heavy lifting behind engineering and science—solving equations, simulating physics, analyzing data. This is what supercomputers spend most of their time doing.

Problem Type	Classical	Quantum	DCPC/RAL
Linear systems (dense)	$O(n^3)$ or $O(n^{2.37})$	**Exponential speedup (HHL)**	No significant advantage
Linear systems (sparse)	O(nnz) iterative	Potential speedup	No significant advantage
Eigenvalue problems	Polynomial	Potential speedup	No significant advantage
PDEs (structured grids)	Polynomial	Uncertain advantage	No significant advantage
Inverse problems	Problem-dependent	Uncertain	**Constraint-based regularization**
Global optimization	Exponential	Modest speedup	**Multi-start as parallel traces**

Assessment. Dense linear algebra is a potential quantum advantage domain (HHL algorithm, with caveats). DCPC provides no advantage for well-conditioned numerical problems but may help with ill-posed inverse problems formulated as constraint satisfaction.

13.8 Bioinformatics and Computational Biology

What this means: Using computers to understand life—analyzing DNA, predicting how proteins fold, designing drugs. Some of the most important unsolved problems in science.

Problem Type	Classical	Quantum	DCPC/RAL
Sequence alignment	Polynomial (dynamic programming)	Limited advantage	No significant advantage
Protein folding	NP-hard (simplified models)	**Potential advantage**	**Constraint-based structure prediction**
Molecular docking	Exponential search	Potential advantage	**Geometric constraint propagation**
Phylogenetic inference	NP-hard	Unknown	**Tree constraint satisfaction**
Gene regulatory networks	Exponential (full inference)	Unknown	**Boolean network as marker dynamics**

Assessment. Both quantum and DCPC show promise for structure prediction problems (folding, docking) where geometric and energetic constraints dominate. DCPC's constraint-native representation may outperform quantum approaches that require problem encoding into Hamiltonians.

13.9 Planning and Scheduling

What this means: Organizing complex operations—airline schedules, factory production, project management, logistics. These problems are everywhere in business and are notoriously difficult.

Problem Type	Classical	Quantum	DCPC/RAL
Job-shop scheduling	NP-hard	Modest speedup (QAOA)	**Temporal constraint propagation**
Vehicle routing	NP-hard	Modest speedup	**Capacity + route constraints**
Resource allocation	NP-hard	Modest speedup	**Multi-resource constraint handling**
Task planning (AI)	PSPACE-complete	Unknown	**Precondition/effect as markers**
Timetabling	NP-complete	Modest speedup	**Dense constraint networks**

Assessment. Planning and scheduling are quintessential constraint-satisfaction domains. DCPC's ability to maintain global constraint awareness without backtracking provides substantial practical speedup on industrial instances with rich constraint structure.

13.10 Summary: Where Each Paradigm Excels

Paradigm	Sweet Spot	Mechanism
Classical	Well-conditioned numerical, polynomial-time graph/string algorithms, gradient-based optimization	Mature algorithms, hardware optimization
Quantum	Quantum simulation, period-finding (Shor), unstructured amplitude amplification	Hilbert space compression, interference
DCPC/RAL	Structured constraint satisfaction, verification, planning, combinatorial optimization with exploitable structure	Delayed commitment, global propagation, overhead elimination

Key insight. The three paradigms are largely complementary rather than competitive:

- Classical computing remains optimal for problems with polynomial-time algorithms or well-behaved continuous structure
- Quantum computing provides unique advantages for quantum simulation and problems reducible to period-finding
- DCPC/RAL provides practical speedup on the large class of industrially relevant constraint-satisfaction problems that are formally hard but structurally tractable

13.11 Estimated Problem Distribution by Optimal Paradigm

For general readers: The tables above might leave you wondering "okay, but how much of real computing falls into each category?" This section provides rough estimates. The key takeaway: DCPC can handle nearly everything classical computers can, plus it's dramatically faster on about one-fifth to one-quarter of problems. Quantum computers, despite the hype, only offer unique advantages on a small slice (3-5%)—though that slice includes some very important problems.

Methodological disclaimer. These percentages are heuristic estimates offered for strategic intuition rather than empirical measurement; the true shares vary strongly by industry workload and problem formulation. These estimates are informed by empirical distributions reported in SAT/SMT, EDA, scheduling, and verification workloads, where a minority of instances account for a disproportionate share of computational effort.

To give readers a practical sense of where computational effort is best directed, we estimate what fraction of real-world computational workloads benefit from each paradigm. These estimates are necessarily approximate and context-dependent, but provide useful guidance.

Important clarification. DCPC subsumes classical computation at the model level. Any classical algorithm can be executed on DCPC by committing markers immediately (classical-mode)—DCPC is designed not to impose overhead when early commitment is optimal. The categories below therefore distinguish:

- **Classical-equivalent:** DCPC provides no significant advantage over classical (early commitment loses nothing)

- **DCPC-advantaged:** DCPC provides substantial speedup via delayed commitment (10× or more)
- **Quantum-advantaged:** Quantum provides unique advantage not achievable by DCPC

Methodology. We consider two distributions: (1) by problem *instances* encountered in practice, and (2) by *computational cycles* consumed. These differ because some problem types (e.g., ML training) consume disproportionate resources relative to their frequency.

By Problem Instances (Frequency of Occurrence)

Paradigm	Estimated Share	Primary Problem Types
Classical-equivalent	~70–75%	Database queries, web serving, file I/O, polynomial-time algorithms, numerical linear algebra, signal processing, compression, rendering
DCPC-advantaged	~20–25%	Scheduling, planning, configuration, verification, structured optimization, constraint-based design, combinatorial search with structure
Quantum-advantaged	~3–5%	Quantum chemistry simulation, cryptanalysis, certain sampling problems, unstructured search at scale
Intractable for all	~1–2%	Adversarially constructed instances, problems requiring exponential precision, formally undecidable fragments

Note: DCPC is applicable to ~95–98% of tractable problems (via classical-mode for the ~70–75% classical-equivalent subset), while providing substantial acceleration primarily on the ~20–25% constraint-dominated subset.

By Computational Cycles (Resource Consumption)

Paradigm	Estimated Share	Notes
Classical-equivalent	~85–90%	Dominated by ML training, video encoding, scientific simulation on well-conditioned systems
DCPC-advantaged	~8–12%	EDA, logistics, verification, planning—high per-instance cost but fewer instances
Quantum-advantaged	~1–3%	Currently near zero (hardware limitations); projected share assumes fault-tolerant quantum computers
Intractable for all	<1%	Rarely attempted in practice

Dominance Hierarchy

The three paradigms form a clear dominance hierarchy for tractable problems:

DCPC ⊇ Classical (DCPC subsumes classical; classical-mode imposes no overhead)
Quantum ⊄ DCPC (Quantum has capabilities DCPC cannot match)
DCPC ⊄ Quantum (DCPC has practical advantages quantum lacks)

More precisely:

Comparison	Relationship
DCPC vs Classical	DCPC ≥ Classical always; DCPC > Classical on ~20–25% of instances
DCPC vs Quantum	Incomparable; each has exclusive advantages
Classical vs Quantum	Incomparable; quantum advantage on ~3–5%, classical better on rest (maturity, cost, availability)

Breakdown Within DCPC-Advantaged Problems

Among the ~20–25% of problem instances where DCPC provides substantial advantage over classical:

Category	Share of DCPC-Advantaged	Expected Speedup
Planning and scheduling	~30%	10–100×
Verification and formal methods	~20%	10–1000×
Combinatorial optimization	~20%	10–100×
Configuration and design	~15%	10–50×
Constraint-based inference	~10%	5–50×
Other structured search	~5%	Variable

Breakdown Within Quantum-Advantaged Problems

Among the ~3–5% of problem instances where quantum provides unique advantage:

Category	Share of Quantum-Advantaged	Quantum Speedup
Quantum simulation (chemistry, materials)	~60%	Exponential
Cryptanalysis (factoring, discrete log)	~15%	Exponential
Unstructured search/sampling	~15%	Quadratic
Linear systems (sparse, well-conditioned)	~10%	Polynomial (with caveats)

Visual Summary

Real-World Computational Problems by Paradigm Advantage

|———————————————|————————————|————|——|
Classical-equiv DCPC-adv QC Intractable
70–75% 20–25% 3–5% 1–2%

The key observation: **DCPC is applicable to ~95–98% of tractable computational problems** (via classical-mode for classical-equivalent workloads), while providing substantial acceleration on the ~20–25% constraint-dominated subset.

Caveats and Qualifications

These estimates should be interpreted with the following caveats:

1. **Domain dependence.** The distribution varies dramatically by industry. Semiconductor EDA may be 60%+ DCPC-advantaged; web services may be 95%+ classical-optimal.
2. **Problem formulation matters.** Many problems can be reformulated to shift categories. A scheduling problem posed as continuous optimization is classical-optimal; posed as constraint satisfaction, it becomes DCPC-advantaged.
3. **Hardware availability.** Quantum percentages assume fault-tolerant hardware that does not yet exist at scale. Current quantum advantage is effectively 0% for practical problems.
4. **Overlap regions.** Some problems fall in boundary zones where multiple paradigms offer comparable performance. The percentages reflect where clear advantage exists.
5. **Evolution over time.** As algorithms improve and hardware changes, these boundaries will shift. Classical algorithms sometimes close the gap with quantum (e.g., tensor network methods for certain quantum simulations).

Implications for Computing Strategy

These estimates suggest:

- **Near-term (0–5 years):** DCPC/RAL architectures could serve as general-purpose classical substrates, matching performance on routine workloads while delivering substantial speedups on the ~20–25% of problems with exploitable constraint structure.
- **Medium-term (5–15 years):** As quantum hardware matures, the 3–5% quantum-advantaged slice becomes practically accessible, primarily for simulation and cryptographic applications. DCPC remains the workhorse for everything else.
- **Long-term equilibrium:** A two-tier architecture where DCPC processors handle all classical and constraint-structured workloads (95–98% of problems), with quantum coprocessors available for the small fraction requiring Hilbert space compression.

The key observation is that **DCPC is not merely an alternative to classical computing but a strict superset**. Unlike quantum computing, which excels in a narrow (though important) problem class, DCPC can in principle serve as a general-purpose substrate while providing substantial acceleration on structured problems.

14. Applications

For general readers: Where would DCPC make the biggest real-world difference? Anywhere that involves complex constraints: designing drugs that fit into proteins like keys into locks, scheduling airline crews across thousands of flights, verifying that software won't crash, or configuring products with thousands of options and compatibility rules. These are problems that cost businesses billions of dollars and countless hours today.

DCPC with RAL execution is particularly powerful for problems with exploitable structure:

14.1 Scientific Computing

- **Drug discovery**: Molecular docking with geometric constraints
- **Protein folding**: Energy landscape navigation with physical constraints
- **Materials science**: Crystal structure prediction with symmetry constraints

14.2 Optimization

- **Scheduling**: Resource allocation with temporal constraints
- **Routing**: Network optimization with capacity constraints
- **Configuration**: System design with compatibility constraints

14.3 Inference

- **Bayesian networks**: Probabilistic reasoning with conditional constraints
- **Constraint satisfaction**: SAT, SMT, and related problems
- **Pattern recognition**: Feature matching with similarity constraints

14.4 Artificial Intelligence

For general readers: DCPC isn't better for training neural networks like ChatGPT—GPUs remain optimal for that. But it's a much more natural fit for AI systems that need to reason, plan, or work with hard constraints. Think of it as hardware for "thinking carefully" rather than "learning from data."

DCPC aligns naturally with AI workloads that require reasoning under partial information, hard constraints, or delayed decision-making. While DCPC does not improve gradient-based learning or dense numerical computation, it offers a more native substrate for symbolic reasoning, planning, verification, and hybrid neuro-symbolic systems.

14.4.1 Strong Fit: Symbolic and Neuro-Symbolic AI

DCPC directly addresses the long-standing weakness of symbolic AI: combinatorial explosion from early commitment.

Classical Symbolic AI	DCPC Symbolic AI
Commits early (facts, bindings, rules)	Maintains admissibility instead of truth
Backtracks aggressively	Propagates constraints globally
Thrashes on large constraint spaces	Commits only when coherence is high

Concrete applications:

- Logic-based planning and program synthesis
- Formal reasoning agents
- Rule-based explainable AI
- Knowledge graph reasoning with constraints

Key insight: DCPC doesn't replace symbolic reasoning—it removes its worst performance pathologies.

14.4.2 Strong Fit: Planning, Scheduling, and Decision-Making

This is arguably DCPC's best AI domain. Planning is constraint-dominated, branching factors explode early, backtracking is expensive, and many near-feasible plans share structure.

Applications:

- Autonomous planning and robotics task planning
- Supply-chain optimization
- Multi-agent coordination
- Game-playing with hard constraints

DCPC keeps plan fragments admissible without committing to full plans—something even advanced planners struggle with on classical hardware.

14.4.3 Strong Fit: Reasoning Under Uncertainty

Many AI systems conceptually want to "keep multiple hypotheses alive until evidence accumulates." Classical systems approximate this with beam search, Monte Carlo sampling, heuristic pruning, and restarts. DCPC does this natively:

- Markers = hypothesis status
- Suppression/reinforcement = evidence accumulation
- Oscillation = unresolved ambiguity
- Measurement = decision point

Applications: Diagnosis systems, fault detection, scientific discovery AI, causal inference with hard constraints.

14.4.4 Promising Fit: Hybrid AI (LLM + Constraints)

Modern AI systems increasingly combine neural perception with symbolic/constraint reasoning:

- LLMs generate candidate code, plans, or hypotheses
- Constraint systems verify or refine outputs

DCPC fits naturally as a constraint back-end:

- LLM generates possibilities
- DCPC tracks admissibility and coherence
- Only coherent outputs are committed

This avoids hard rejection loops, excessive reranking, and brittle rule enforcement. **Importantly: DCPC does not compete with neural networks—it complements them.**

14.4.5 Poor Fit: Deep Learning Training

DCPC does **not** help with backpropagation, SGD, dense linear algebra, or large matrix multiplications. GPUs and TPUs remain optimal for CNNs, Transformers, and diffusion models. Positioning DCPC as a replacement for these workloads would be incorrect.

14.4.6 Conceptual Alignment

There is a deep alignment between DCPC and how humans reason:

- Humans rarely commit instantly
- We hold partial beliefs
- We suppress unlikely options
- We wait for coherence before deciding

Modern AI is rediscovering this via chain-of-thought, tree-of-thought, self-consistency, and debate/reflection loops. DCPC is a hardware-level embodiment of this reasoning style—a meaningful contribution even absent immediate benchmarks.

Bottom line: DCPC is not "AI hardware" in general. DCPC is excellent AI hardware for *reasoning*. It complements neural models instead of competing with them, and aligns with where AI is actually going: hybrid, constraint-aware systems.

14.5 Limitations

DCPC is ineffective for:

- Breaking modern cryptography (no exponential compression)
- Unstructured search (no better than Grover)
- Adversarially constructed instances (no exploitable structure)

---

15. Conceptual Significance

For general readers: Beyond practical applications, DCPC changes how we think about computation and physics. It suggests that quantum mechanics' power comes from two separable things: (1) delaying decisions, which we can replicate classically, and (2) exponential compression of possibilities, which requires actual quantum physics. This helps explain both what quantum computers can do and why they're so hard to build.

15.1 Hilbert Space as Compressed Description

This architecture suggests that Hilbert space may be understood not as a metaphysical necessity but as a maximally compressed description of admissible pre-fact dynamics. Any pre-fact language satisfying:

1. Delayed commitment
2. Global constraint propagation
3. Criticality-driven resolution

will converge toward quantum-like structure.

15.2 Unification with Entropy-Based Frameworks

The DCPC/RAL model aligns naturally with frameworks in which:

- Time emerges from entropy dynamics
- Facts crystallize at critical thresholds
- Measurement is phase transition, not arbitrary readout

The Taylor Limit provides the physical grounding: fact creation requires entropy expenditure bounded by system capacity.

---

16. Conclusion

For general readers: We've presented a new way of computing that keeps options open until the last moment. This simple idea—"don't decide until you have to"—turns out to have profound consequences. It can make many hard problems dramatically easier, it can be built with today's technology, and it helps explain why quantum computers work the way they do. DCPC won't

replace quantum computers for the problems they're uniquely good at, but it will handle everything else—which turns out to be almost everything.

Replacing bits with RAL markers transforms computation from value manipulation into guided assembly. The resulting Delayed-Commit Pattern Computer provides a classical substrate capable of reproducing key quantum computational behaviors on structured problems while remaining grounded in admissibility, entropy, and emergence.

Key contributions:

1. **Unified framework**: DCPC as abstract model, RAL as concrete instruction set
2. **Structural interference**: Admissibility failure replaces amplitude cancellation
3. **Taylor Limit**: Physical bound on distinguishability constrains computation
4. **Structured oracle exploitation**: Ternary constraint oracles enable O(log N) recovery on prefix-structured problems (without contradicting Grover optimality for Boolean-marking oracles)
5. **Native heuristics**: CDCL executes without backtracking overhead
6. **Clear boundaries**: DCPC dominates classical, complements quantum

This offers both a practical architecture for structured problems and a conceptual bridge between classical computation, quantum mechanics, and entropy-based physical frameworks.

References

Quantum Computing and Algorithms

[1] Grover, L. K. (1996). "A fast quantum mechanical algorithm for database search." *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, 212–219.

[2] Shor, P. W. (1997). "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer." *SIAM Journal on Computing*, 26(5), 1484–1509.

[3] Nielsen, M. A., & Chuang, I. L. (2010). *Quantum Computation and Quantum Information* (10th Anniversary Edition). Cambridge University Press.

[4] Farhi, E., Goldstone, J., & Gutmann, S. (2014). "A quantum approximate optimization algorithm." *arXiv preprint arXiv:1411.4028*.

[5] Preskill, J. (2018). "Quantum computing in the NISQ era and beyond." *Quantum*, 2, 79.

SAT Solving and Constraint Propagation

[6] Marques-Silva, J. P., & Sakallah, K. A. (1999). "GRASP: A search algorithm for propositional satisfiability." *IEEE Transactions on Computers*, 48(5), 506–521.

[7] Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., & Malik, S. (2001). "Chaff: Engineering an efficient SAT solver." *Proceedings of the 38th Design Automation Conference*, 530–535.

[8] Eén, N., & Sörensson, N. (2003). "An extensible SAT-solver." *International Conference on Theory and Applications of Satisfiability Testing*, 502–518.

[9] Biere, A., Heule, M., van Maaren, H., & Walsh, T. (Eds.). (2009). *Handbook of Satisfiability*. IOS Press.

[10] Knuth, D. E. (2015). *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability*. Addison-Wesley.

Constraint Satisfaction and Propagation

[11] Mackworth, A. K. (1977). "Consistency in networks of relations." *Artificial Intelligence*, 8(1), 99–118.

[12] Dechter, R. (2003). *Constraint Processing*. Morgan Kaufmann.

[13] Rossi, F., van Beek, P., & Walsh, T. (Eds.). (2006). *Handbook of Constraint Programming*. Elsevier.

[14] Bessière, C. (2006). "Constraint propagation." *Handbook of Constraint Programming*, 29–83.

Computational Complexity

[15] Arora, S., & Barak, B. (2009). *Computational Complexity: A Modern Approach*. Cambridge University Press.

[16] Papadimitriou, C. H. (1994). *Computational Complexity*. Addison-Wesley.

[17] Bennett, C. H., Bernstein, E., Brassard, G., & Vazirani, U. (1997). "Strengths and weaknesses of quantum computing." *SIAM Journal on Computing*, 26(5), 1510–1523.

[18] Aaronson, S. (2013). *Quantum Computing Since Democritus*. Cambridge University Press.

Information Theory and Entropy

[19] Shannon, C. E. (1948). "A mathematical theory of communication." *Bell System Technical Journal*, 27(3), 379–423.

[20] Landauer, R. (1961). "Irreversibility and heat generation in the computing process." *IBM Journal of Research and Development*, 5(3), 183–191.

[21] Bennett, C. H. (1973). "Logical reversibility of computation." *IBM Journal of Research and Development*, 17(6), 525–532.

[22] Zurek, W. H. (1989). "Thermodynamic cost of computation, algorithmic complexity and the information metric." *Nature*, 341(6238), 119–124.

Holographic Bounds and Quantum Gravity

[23] Bekenstein, J. D. (1981). "Universal upper bound on the entropy-to-energy ratio for bounded systems." *Physical Review D*, 23(2), 287.

[24] 't Hooft, G. (1993). "Dimensional reduction in quantum gravity." *arXiv preprint gr-qc/9310026*.

[25] Susskind, L. (1995). "The world as a hologram." *Journal of Mathematical Physics*, 36(11), 6377–6396.

[26] Bousso, R. (2002). "The holographic principle." *Reviews of Modern Physics*, 74(3), 825.

Hardware Architectures and Accelerators

[27] Kung, H. T. (1982). "Why systolic architectures?" *Computer*, 15(1), 37–46.

[28] Jouppi, N. P., et al. (2017). "In-datacenter performance analysis of a tensor processing unit." *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 1–12.

[29] Mutlu, O., Ghose, S., Gómez-Luna, J., & Ausavarungnirun, R. (2019). "Processing data where it makes sense: Enabling in-memory computation." *Microprocessors and Microsystems*, 67, 28–41.

[30] Ielmini, D., & Wong, H. S. P. (2018). "In-memory computing with resistive switching devices." *Nature Electronics*, 1(6), 333–343.

Belief Propagation and Message Passing

[31] Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann.

[32] Yedidia, J. S., Freeman, W. T., & Weiss, Y. (2003). "Understanding belief propagation and its generalizations." *Exploring Artificial Intelligence in the New Millennium*, 8, 236–239.

[33] Mézard, M., & Montanari, A. (2009). *Information, Physics, and Computation*. Oxford University Press.

Quantum Measurement and Foundations

[34] Wheeler, J. A., & Zurek, W. H. (Eds.). (1983). *Quantum Theory and Measurement*. Princeton University Press.

[35] Zurek, W. H. (2003). "Decoherence, einselection, and the quantum origins of the classical." *Reviews of Modern Physics*, 75(3), 715.

[36] Schlosshauer, M. (2007). *Decoherence and the Quantum-to-Classical Transition*. Springer.

Cellular Automata and Parallel Computation

[37] Wolfram, S. (2002). *A New Kind of Science*. Wolfram Media.

[38] Toffoli, T., & Margolus, N. (1987). *Cellular Automata Machines: A New Environment for Modeling*. MIT Press.

Applications: Drug Discovery and Protein Folding

[39] Kitchen, D. B., Decornez, H., Furr, J. R., & Bajorath, J. (2004). "Docking and scoring in virtual screening for drug discovery." *Nature Reviews Drug Discovery*, 3(11), 935–949.

[40] Jumper, J., et al. (2021). "Highly accurate protein structure prediction with AlphaFold." *Nature*, 596(7873), 583–589.

Planning and Scheduling

[41] Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated Planning: Theory and Practice*. Morgan Kaufmann.

[42] Pinedo, M. L. (2016). *Scheduling: Theory, Algorithms, and Systems* (5th ed.). Springer.

Verification and Model Checking

[43] Clarke, E. M., Grumberg, O., & Peled, D. A. (1999). *Model Checking*. MIT Press.

[44] Biere, A., Cimatti, A., Clarke, E. M., & Zhu, Y. (1999). "Symbolic model checking without BDDs." *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 193–207.

Appendix A: RAL Rewrite Rule Specification

A.1 Complete Rule Set

R1: (∅, σ) → (∅, ∅) [Forbidden propagation]
R2: (⊥, σ) → (⊥, ⊥) [Incompatibility propagation]
R3: (○, ○) → (⊥, ⊥) [Mutual exclusion detection]
R4: ⇌ⁿ → ↓ [Oscillation decay, n > threshold]
R5: (↑, ↑)_closed → (✓, ✓) [Reinforcement closure]
R6: (↑, ↓) → (○, ○) [Mixed signals reset]
R7: (✓, ∅) → (∅, ∅) [Coherence destroyed by forbidden]

A.2 Rule Priority

Rules are applied in priority order: R1 > R2 > R7 > R3 > R4 > R5 > R6

Higher priority rules fire first; lower priority rules apply to remaining markers.

Appendix B: Taylor Limit Derivation

B.1 Information-Theoretic Form

For a system with maximum entropy S_max:

$\Lambda_T = \exp(S_{max} / k_B)$

B.2 Computational Form

For n bits of addressable state:

$\Lambda_T = 2^n$

B.3 Physical Form

For a region of spacetime with surface area A (holographic bound):

$\Lambda_T = \exp(A / 4\ell_P^2)$

This connects the Taylor Limit to established bounds in quantum gravity.