REMARC: Reduction-Modeled Adaptive Replacement Cache

A Multi-Dimensional Framework for Unified Cache Eviction and Migration

Hamza Jamil Saied (The Sphynx)

2026-04-25

Table of contents

Status: Research journal (living document). This is not a definitive paper — it documents ongoing work, negative results, and open questions alongside findings. Conclusions may change as experiments progress. **Date:** April 2026

---

1 1. Introduction

Cache replacement policies make a single decision: keep or evict. This is fundamentally a one-dimensional problem — policies like ARC, LRU, and LIRS classify items along one axis (recency, frequency, or reuse distance) and draw a threshold.

Real cache systems face multi-dimensional decisions: - **Eviction:** should this item stay in cache? - **Migration:** should this item move to a different tier/node? - **Promotion:** should this item move to a faster tier? - **Demotion:** should this item move to a slower tier?

These are independent questions that existing policies answer with separate mechanisms bolted together. ARC handles eviction. A separate policy handles migration. Another handles tiering. Each adds its own data structures, its own tracking overhead, its own tuning parameters.

REMARC (Reduction-Modeled Adaptive Replacement Cache) unifies all of these into a single framework. The key insight is **dimensional reduction**: K access signals (recency, locality, tier pressure, …) are reduced into a compact state space, and M decision scores (evict, migrate, promote, …) are derived from that space via precomputed projection functions.

1.1 1.1 Contributions

1. A formal framework for multi-dimensional cache policy based on switched linear systems and linear projection, with a representation theorem showing coverage of 9 existing policies.
2. REMARC: an instantiation of this framework for NUMA-aware caching (K=2, M=2) with **desire encoding** — a mechanism for threshold-free migration using per-node desire scores.
3. Comprehensive evaluation: 30+ single-node variants, 2-node and 3-node NUMA simulation with 16 findings, demonstrating 3-4× cost reduction from desire encoding over value-based migration.
4. An independence theorem: desire encoding preserves per-node independence for n nodes, while value encoding does not. Global state is O(1) per decision.
5. SIMD-optimized implementation using precomputed lookup tables (256 entries, SSE2 direct arithmetic, O(1) amortized per 16 keys).
6. Real hardware validation on AWS c6i.metal confirming the NUMA placement signal that motivates REMARC's migration axis.
7. Generalization to K>2 for tiered storage, heterogeneous memory, and distributed caching.

---

2 2. Background and Motivation

2.1 2.1 Existing Cache Policies Are One-Dimensional

Policy	Dimension	Output	Data Structure
LRU	Recency	Evict	Stack
LFU	Frequency	Evict	Counters
ARC	Recency + Frequency (1D blend)	Evict	4 lists + ghost lists + p_
LIRS	Reuse distance	Evict	2 lists + HIR
W-TinyLFU	Frequency sketch	Evict	Count-Min Sketch + Window

None of these produce **migration** or **tier placement** decisions. They answer one question: "should I keep this?" When systems need migration (NUMA) or tiering (NVM), they add separate policies on top.

2.2 2.2 The Cost of Bolt-On Policies

Adding a migration policy on top of ARC means: - Separate per-key tracking for migration candidates (CrossNodeAccesses counter) - Separate scanner for migration decisions - Separate adaptation mechanism (threshold tuning) - Interaction complexity: "ARC says keep, migration says move" — contradictory signals from co-equal policies

This is the **dimensional conflation problem**: a 1D policy (ARC) is asked to handle a 2D decision (keep/move). The policy can't express "this key is hot (keep it) but on the wrong node (move it)." The axes compete without a framework for resolution.

2.3 2.3 What We Need: Multi-Dimensional Policy

A cache policy that: 1. Accepts K input signals (recency, frequency, locality, tier pressure, …) 2. Produces M action scores (evict, migrate, promote, demote, …) 3. Uses a single shared state space (not K×M separate mechanisms) 4. Is self-tuning (adaptive thresholds via observed outcomes) 5. Is efficient (O(1) amortized per decision, SIMD-parallel)

---

3 3. The REMARC Framework

3.1 3.1 Intuition

Think of a key's access history as a point in K-dimensional space, where each dimension tracks a different signal source (how often accessed locally, how often remotely, how often from disk, …). We want to project this point onto M action axes — each axis answers one question ("should I evict?", "should I migrate?").

This is **dimensional reduction**: we don't need the full K-dimensional history to make decisions. We need a compressed representation that preserves the decision-relevant information. The projection $f : \mathbb{R}^K \to \mathbb{R}^M$ discards information irrelevant to the decision, analogous to how dimensionality reduction techniques (PCA, truncated SVD) compress data by projecting onto the most important directions. REMARC's projection vectors are hand-designed from domain knowledge rather than learned from data (§6.3 discusses the conditions under which the hand-designed projections approximate the data-driven optimum).

The defining property of REMARC is competitive coupling between axes. Observing source j is evidence for source j and evidence against all other sources — the boost-one-decay-rest update couples the K state variables into a shared state space. Without coupling, K independent EMA filters would require separate logic to combine;

with coupling, the combination happens in the state itself. The projection simply reads out different marginals of the joint distribution: eviction reads total activity, migration reads locality imbalance, promotion reads tier mismatch. A single coupled state space, projected differently, produces all action scores.

3.2 3.2 Formal Definition

3.2.1 State Vector

Each key maintains a state vector:

$$\mathbf{S} = (S_0, S_1, \ldots, S_{K-1}) \in [0, \text{MAX}]^K$$

Where K is the number of signal sources and MAX is the quantization level (15 for 4-bit encoding).

3.2.2 Update Rules (Switched Linear System)

When the key is accessed from signal source j:

$$\mathbf{S}_t = M_j \cdot \mathbf{S}_{t-1} + \mathbf{b}_j$$

Where $M_j \in \mathbb{R}^{K \times K}$ is a diagonal matrix:

$$M_j = \text{diag}(1 - \alpha_0, \ldots, 1, \ldots, 1 - \alpha_{K-1})$$

(1 at position j, decay factors elsewhere), and $\mathbf{b}_j$ is:

$$\mathbf{b}_j = (0, \ldots, \alpha_j \cdot \text{MAX}, \ldots, 0)$$

(boost at position j only).

Interpretation: observing an access from source j is evidence for source j and evidence against all other sources. The update rule maintains a sufficient statistic for each source's access rate under an exponential forgetting model. The steady-state formula (§3.3) makes this precise: S_j^* converges to a function of p_j (the fraction of accesses from source j), confirming that the state encodes the access distribution.

Time decay (per scan cycle):

$$\mathbf{S} \leftarrow \lambda \cdot \mathbf{S}$$

3.2.3 Decision Surfaces (Projection + Nonlinear Combination)

Action scores are computed in two stages (matching the proj_s and decide atoms from §3.4):

Stage 1: Feature extraction (proj_s). A matrix projection extracts k features from the K-dimensional state:

$$\mathbf{z} = A \cdot \hat{\mathbf{S}}, \quad A \in \mathbb{R}^{k \times K}, \quad k \leq K$$

Where $\hat{\mathbf{S}} = \mathbf{S}/\text{MAX}$ is the normalized state (each component in [0,1]). Each component $z_i = \langle a_i, \hat{\mathbf{S}} \rangle$ is a scalar product — a linear functional over the state. The projection vectors a_i determine which aspects of the state are decision-relevant.

Stage 2: Nonlinear scoring (decide). Each action score is computed as a nonlinear function over the feature vector:

$$f_m(\mathbf{S}) = g_m(\mathbf{z}) = g_m(z_1, z_2, \ldots, z_k)$$

Where $g_m : \mathbb{R}^k \to \mathbb{R}$ is a fixed nonlinear function. In REMARC's current instantiation, g_m is a product of projected components. For example, the NUMA Evict score is:

$$\text{Evict} = (1 - R)(1 - F)$$

This is **not** representable as $\sigma(a \cdot S)$ — it is a multiplicative interaction between two projected features, creating a nonlinear decision surface. The nonlinearity lives in `decide`, not in a scalar activation function.

The projection IS dimensional reduction: $\text{proj}_s : \mathbb{R}^K \to \mathbb{R}^k$ with $k \leq K$. The mapping is lossy by design — we keep only features relevant to decisions. The `decide` function then computes nonlinear interactions between those features.

Model class. REMARC defines policies as nonlinear functionals over a low-dimensional linear projection of the state: $f(S) = g(AS)$. The projection A determines what information is available to decisions (everything orthogonal to $\text{span}(A)$ is irretrievably lost). The decision function g determines how that information is combined (multiplicative terms, thresholds, etc.). Expressiveness depends on both — a richer projection (larger k) or a more complex g (higher-order interactions) increases the class of representable policies, but neither alone is sufficient.

Connection to sufficiency. The steady-state result (§3.3) shows that S encodes a biased empirical distribution of access sources. The projection A extracts sufficient statistics from this distribution for the decision task at hand. The sufficiency condition (§3.4, Correlation Condition) formalizes this: every state dimension must carry non-zero information about at least one decision, ensuring the projection does not discard decision-relevant information.

3.3 3.3 Properties

3.3.1 Steady-State Encoding (Key Result)

For a key where source j accounts for fraction p_j of total accesses (empirical frequency, not a probabilistic assumption), the steady state satisfies:

$$S_j^* = \text{MAX} \cdot \frac{p_j \alpha_j}{p_j \alpha_j + (1 - p_j)(1 - \lambda)}$$

Derivation. At steady state, the expected boost equals the expected decay for each dimension. Consider dimension j over one time step. With probability p_j, source j fires and S_j is boosted:

$$S_j \leftarrow (1 - \alpha_j) S_j + \alpha_j \cdot \text{MAX}$$

With probability $1 - p_j$, source j does not fire and S_j decays:

$$S_j \leftarrow \lambda \cdot S_j$$

At the fixed point S_j^*, the expected update must equal S_j^*:

$$S_j^* = p_j[(1 - \alpha_j) S_j^* + \alpha_j \cdot \text{MAX}] + (1 - p_j)\lambda S_j^*$$

$$S_j^* = p_j(1 - \alpha_j) S_j^* + p_j \alpha_j \cdot \text{MAX} + (1 - p_j)\lambda S_j^*$$

$$S_j^* - p_j(1 - \alpha_j) S_j^* - (1 - p_j)\lambda S_j^* = p_j \alpha_j \cdot \text{MAX}$$

$$S_j^*[1 - p_j(1 - \alpha_j) - (1 - p_j)\lambda] = p_j \alpha_j \cdot \text{MAX}$$

$$S_j^*[1 - p_j + p_j \alpha_j - \lambda + p_j \lambda] = p_j \alpha_j \cdot \text{MAX}$$

$$S_j^*[p_j \alpha_j + (1 - p_j)(1 - \lambda)] = p_j \alpha_j \cdot \text{MAX}$$

$$S_j^* = \text{MAX} \cdot \frac{p_j \alpha_j}{p_j \alpha_j + (1 - p_j)(1 - \lambda)} \quad \square$$

This gives a **deterministic relationship between observed access frequency and REMARC state**: the state vector encodes the empirical access rate distribution across sources. No stationarity or independence assumptions required — p_j is an observable quantity. Design implications:

- **Sensitivity:** increasing α_j makes the state respond faster to source j (larger S_j^* for the same p_j). This is how migration disfavor is implemented ($\alpha_l > \alpha_r$ makes local signal dominate).
- **Memory:** λ controls how quickly stale patterns fade. Lower λ = shorter memory = faster adaptation to workload shifts.
- **Design procedure:** given a desired sensitivity profile (how much remote pressure triggers migration), solve for α_j and λ from the steady-state formula. Parameters are no longer arbitrary — they are derived from the desired decision boundary.

3.3.2 Convergence (Deterministic)

The convergence guarantee is deterministic — it holds for **any** access pattern, not just stationary distributions.

Key structural property: each M_j is diagonal. Products of diagonal matrices are diagonal. Therefore the joint spectral radius (JSR) — the worst-case spectral radius over all products of the switching matrices — is computable in closed form:

$$\rho = \max(\max_j(1-\alpha_j),\ \lambda) < 1$$

Since $0 < \alpha_j < 1$ and $0 < \lambda < 1$, the JSR is strictly less than 1. This means the homogeneous system $M_{\sigma(t)} \cdots M_{\sigma(1)} \cdot S_0 \to 0$ for any switching sequence σ — adversarial, periodic, or stochastic.

Proof. For any switching sequence $\sigma = (j_1, j_2, \ldots, j_t)$, the product $P = M_{j_t} \cdots M_{j_2} M_{j_1}$ is diagonal. Its i-th diagonal entry is:

$$P_{ii} = \prod_{k=1}^{t} d_i(j_k), \quad d_i(j_k) = \begin{cases} 1-\alpha_i & \text{if } j_k = i \\ \lambda & \text{if } j_k \neq i \end{cases}$$

Each factor satisfies $0 < d_i(j_k) < 1$. Therefore $P_{ii} \to 0$ as $t \to \infty$, and $\|P\|_\infty = \max_i |P_{ii}| \to 0$. The convergence rate is geometric: $\|P\|_\infty \leq \rho^t$, where $\rho = \max(\max_j(1-\alpha_j), \lambda) < 1$. □

Why diagonal matters: for general (non-diagonal) matrices, the JSR can exceed individual spectral radii because off-diagonal coupling in products can amplify eigenvalues. Diagonal matrices have no off-diagonal coupling. Each entry of the product decays as a product of independent scalars. No amplification is possible.

Convergence behavior by access pattern:

Switching sequence	Convergence
Each source has limiting frequency p_j	Converges to steady-state formula S_j^* at rate $O(\rho^t)$
Periodic (e.g., alternating sources)	Converges to periodic orbit
Adversarial (worst case)	State stays bounded, homogeneous transient decays to 0 at rate $O(\rho^t)$

Switching sequence	Convergence
Non-stationary (changing frequencies)	State tracks the changing frequencies with lag determined by α_j and λ

For the cache application, workloads have empirical access frequencies (limiting p_j exists), so convergence to the steady-state formula is guaranteed. But the guarantee does not depend on this — the state remains well-behaved under any access pattern.

3.3.3 Separability

Different actions respond to different projections of the state. The Evict score responds to overall coldness (low total signal). The Migrate score responds to imbalance (high remote signal relative to local). These are independent axes in the decision space — the projection rows a_i are chosen to extract independent decision-relevant features.

3.4 3.4 Representation Theorem

The preceding sections define REMARC as a specific mechanism (competitive erosion + linear projection + nonlinear decision surface). This section shows that the mechanism is not arbitrary — it captures the algebraic structure of sequential decision functions as a class.

3.4.1 Spaces (Independent Axioms)

The framework rests on three independently axiomatized spaces:

Observation space O. The set of observable events (which source fired, which tier was accessed, etc.). For REMARC: O = {0, 1, ..., K-1} (discrete sources). No axioms beyond being a non-empty set.

State space S. A compact (closed and bounded) subset of R^n, n > 0. In REMARC: S = [0, MAX]^K. - Boundedness is the compression property — phi maps unbounded histories to bounded state - No sign constraint — positivity is a specialization of the REMARC instantiation, not an axiom

Discrimination space D. Defined constructively as the image of the decision mapping: $D_f := f(S)$ where $f = \texttt{decide} \circ \texttt{proj}_s$. D is not a predefined codomain — it is the space of realizable discrimination scores, constructed by the composition itself. - Compact because S is compact and f is continuous (continuous image of compact is compact) - Ordered because decisions must be comparable (higher score = stronger evidence for that action) - Surjectivity is trivial: $f : S \to D_f$ is surjective by construction (D_f is exactly the image) - There are no "unmapped regions" in D — by definition, every point in D_f is realized by some state in S

Key property: REMARC does not map into a decision space — it constructs the decision space. The composition $f : S \to D_f$ is a many-to-one compression (not injective, since

multiple states can produce the same discrimination scores) that is surjective onto its constructed image by definition. The information loss is the compression itself: fewer distinguishable outputs than inputs, but no unrealizable outputs.

Compression invariant. Since f is compressive (many-to-one), the intrinsic dimension of the decision space is bounded by that of the state space: $\dim(D_f) \leq \dim(S)$. Equality holds only if f preserves all degrees of freedom (i.e., no compression). Cardinality is typically equal (both are continuum), but this is structurally irrelevant — a line and a plane have the same cardinality but different dimensions. The correct invariant is that the decision space does not introduce independent degrees of freedom beyond those present in the state space.

Family-level decision space. Across the REMARC family of mappings $\mathcal{F} = \{f_\theta\}$ (parameterized by projection matrix A and decision function g), the induced decision space is $D_\mathcal{F} := \bigcup_{f \in \mathcal{F}} f(S)$. This space is surjective by construction — it contains no unreachable elements relative to the model. Whether $D_\mathcal{F} = \mathbb{R}^k$ (expressive completeness) depends on whether the family of projections and decision functions is rich enough to cover all of $\mathbb{R}^k$. This is the central open question for the theoretical treatment (Paper 3).

3.4.2 Three-Atom Decomposition

A sequential decision function factors into three atomic operations:

$$\pi_c = \texttt{decide} \circ \texttt{proj}_s \circ \varphi^t$$

Where φ^t denotes φ composed t times (repeated state updates over t observations).

Atom 1: phi (state update). $\varphi : S \times O \to S$. An incremental map that takes current state + observation, produces new state. In REMARC: a switched linear system ($S_t = M_j \cdot S_{t-1} + b_j$). Contractive (JSR < 1), guaranteeing convergence for any observation sequence.

Atom 2: proj_s (linear projection). $\texttt{proj}_s : S \to \mathbb{R}^k$. A linear map (scalar products) that extracts k features from the n-dimensional state. Each component is $\langle a_j, S \rangle = \sum_i a_{ji} s_i$. The projection vectors a_j determine which aspects of the state are relevant for each decision.

Atom 3: decide (nonlinear combination). $\texttt{decide} : \mathbb{R}^k \to D$. A fixed nonlinear function that combines projected features into discrimination scores. In REMARC: a product of projected components (e.g., Evict = (1-R)(1-F) where R and F are projected features). The nonlinearity is what gives the framework expressive power beyond linear classification.

Note on sigma (discretization): Converting continuous discrimination scores to discrete actions (thresholding) is NOT an atomic operation of the framework. It is an implementation concern — a downstream step applied by the system that reads the scores. For the algebraic framework, the output is D (continuous biases), not A (discrete actions). Discretization is a corollary: "for discrete decisions, apply any monotone threshold to D."

3.4.3 Sufficient Statistic Condition

The factorization holds if and only if the state S is a **sufficient statistic** for the decision:

$$\pi(h) \text{ depends on } h \text{ only through } \varphi^t(h)$$

That is: two observation histories that produce the same state must produce the same discrimination scores. If this condition holds, the full history is redundant — the compressed state captures everything the decision needs.

3.4.4 Coverage of Existing Policies

Policy	Sufficient statistic	Factorization	Coverage
LRU (recency)	S_0 = EMA of recency	K=1, M=1, $A = -1$	Approximate
LFU (frequency)	S_0 = accumulating EMA of frequency	K=1, M=1, $A = -1$	Exact
Clock (reference bit)	S_0 = binary EMA ($\alpha \to 1$, $\lambda \to 0$)	K=1, M=1	Approximate
ARC (decision structure)	S_0 = recency, S_1 = frequency	K=2, M=1	Exact
W-TinyLFU (decision)	S_0 = frequency estimate	K=1, M=1	Exact
LeCaR (decision structure)	S_0 = LRU score, S_1 = LFU score	K=2, M=1	Exact
LIRS (reuse distance)	IRR $\approx$ EMA recency	K=1, M=1	Approximate
ARC (ghost lists)	Ghost membership $\approx$ binary EMA	K=2, M=1	Approximate
S3-FIFO (binary frequency gate)	Single-bit reaccess $\approx$ steep EMA ($\alpha \to 1$, fast λ)	K=1, M=1	Approximate

"Exact" means the REMARC configuration produces the same decision on every access sequence. **"Approximate"** means the REMARC decisions converge toward the policy's behavior as α and λ are tuned.

Why LRU is approximate, not exact: LRU evicts the globally least-recently-used item, which requires knowing the total ordering of all items. REMARC's per-key EMA state tracks recency independently per item — the item with lowest S_0 is approximately least-recent, but the approximation error depends on the relative timing of accesses to different items. The error vanishes as $\alpha \to 1$ (state becomes binary: recently accessed or not).

Why S3-FIFO is a boundary case: S3-FIFO's promotion condition (one reaccess in the small queue) is a binary frequency gate — the sharpest possible threshold. REMARC

approximates this with a fast-response, fast-decay EMA ($\alpha \to 1$, $\lambda \to 0$) that produces a steep sigmoid rather than a hard step. The approximation error is the difference between a binary gate and a steep sigmoid, and is quantifiable: the "false promotion" region where EMA crosses the threshold without a true reaccess has width $O(\lambda/\alpha)$.

The distinction between exact and approximate coverage corresponds to the modeling vs. approximation distinction:

- **Modeling**: REMARC *is* the policy. Same decisions on every input. Holds for policies whose state is literally EMA signals (LRU, LFU, W-TinyLFU frequency estimate).
- **Approximation**: REMARC gets arbitrarily close as K increases or parameters are tuned. Holds for policies whose decision boundaries can be approached by linear projections over EMA state (LIRS, ARC ghost lists).

3.4.5 Boundary

Policies that fall **outside** the REMARC space:

Policy	Reason excluded
Bélády's MIN / OPT	Requires future knowledge. Not online. π depends on $\mathcal{H} \cup$ future, violating the factorization.
Randomized eviction	Non-deterministic. $\pi(h)$ is not a function of h alone.
Oracle-assisted policies	Depend on external information beyond access history.

These exclusions are precise and principled. Bélády's MIN is the fundamental limit of online policies — no online policy can match it. Randomized policies are outside because REMARC is deterministic. Everything else is inside.

3.4.6 Correlation Condition

The combined projection $f = \texttt{decide} \circ \texttt{proj}_s : [0, \text{MAX}]^K \to D$ discards information — this is reduction by design. But the reduction is **compressive**, not lossy, when a specific condition holds.

Definition (Valid Signal). Input dimension S_j is *valid* if it carries non-zero information about at least one output discriminator:

$$\exists m \in \{0, \ldots, M-1\} : \frac{\partial f_m}{\partial S_j} \neq 0$$

That is: S_j has non-zero weight in at least one projection row of A. If $\partial f_m / \partial S_j = 0$ for all m, then S_j is irrelevant to every decision and should not be in the state space.

Definition (Weak REMARC). Every input dimension is valid — each S_j contributes to at least one action score. The projection discards no decision-relevant information. The reduction is lossless with respect to the discrimination space.

Definition (Strong REMARC). Every input dimension correlates with every output discriminator — each S_j has non-zero weight in every row of A. The projection matrix has no zero entries. The state is maximally coupled to the decision space.

Consequence: under Weak REMARC, the state is a sufficient statistic for the decisions (nothing decision-relevant is lost). Under Strong REMARC, the state is a *maximally informative* sufficient statistic (every dimension contributes to every decision). The design principle for signal selection is: given a budget of K dimensions, choose the K signal sources that maximize total correlation across all M outputs.

3.4.7 Geometric Interpretation

The REMARC configuration space has geometric structure:

- K is discrete — it stratifies the space into layers
- $\{\alpha_j\}$ and λ are continuous — coordinates within each stratum
- A is the projection matrix — its rows live on a Grassmannian (the space of k-dimensional subspaces of $\mathbb{R}^K$)

Existing policies are specific points or curves in this stratified space. REMARC-as-framework is the space itself. This gives:

- **Policy comparison** = distance in the parameter space
- **Policy design** = search over the parameter space for a point that satisfies performance constraints
- **Policy analysis** = algebraic properties (convergence, stability) are properties of the space, not of individual algorithms
- **Policy generalization** = moving to a higher stratum (increasing K) expands the reachable decisions

The K=2 NUMA instantiation (§4) is a navigation of this space to a specific point. The K>2 generalizations (§6) are other strata. The open questions (§9) are questions about the geometry of the space itself.

3.5 3.5 SIMD-Optimized Implementation

For K = 2 with 4-bit quantization (MAX=15):

- State is packed into one byte: `(S_1 << 4 | S_0)`
- Decision surfaces are precomputed into 256-entry lookup tables (`E_lookup`, `M_lookup`)
- **SIMD batch scoring:** load 16 state bytes, extract S_0 and S_1 nibbles via SSE2 mask+shift, compute E and M scores using SIMD 16-bit multiply-add arithmetic (~18 instructions per batch of 16 keys)
- Scanning N keys: O(N/16) SIMD operations

- Note: `pshufb` (SSSE3) cannot perform 256-entry lookups directly — it uses only the low 4 bits of each control byte, limiting it to 16-entry tables. Since the E and M formulas mix both S_0 and S_1 (F = (S_0+S_1)/30, L = S_1/(S_0+S_1)), the tables are not separable into independent nibble lookups. Direct SIMD arithmetic is used instead.

For K > 2 or 8-bit quantization: - State occupies K × 4 bits per key (or K bytes for 8-bit) - Decision surfaces computed via SIMD multiply-add - AVX2 processes 32 keys per batch

3.6 3.6 Outcome-Based Adaptation

Standard ARC adapts via ghost lists (observe what was evicted, adjust p_). REMARC adapts via **reversal feedback**:

Signal	Meaning	Threshold Adjustment
Eviction reversal rate	Keys evicted then reloaded	_e too low → increase
Migration reversal rate	Keys migrated then ping-ponged	_m too low → increase
Cache miss rate	Overall miss rate high	_e too high → decrease
Cross-node latency	Remote access penalty high	_m too high → decrease

This is online learning with outcome feedback — the same principle as ARC's ghost list adaptation, but without maintaining ghost state.

---

4 4. NUMA Instantiation (K=2, M=2)

4.1 4.1 Signal Sources

Source	Meaning	Update on
S_0 = S_local	Smoothed local access rate	Local Get()
S_1 = S_remote	Smoothed remote access rate	Remote Get() from another NUMA node

4.2 4.2 Decision Surfaces

$$R = S_0/15, \quad F = (S_0 + S_1)/30, \quad L = S_1/(S_0 + S_1)$$

$$\text{Evict}(\mathbf{S}) = (1 - R)(1 - F)$$

$$\text{Migrate}(\mathbf{S}) = F \cdot L \cdot (1 - R)$$

4.3 4.3 State Space Partition

```
              S_remote →
              low              high
S_local
high       KEEP                          CONTESTED
           E   0, M   0                 E   0, M moderate
           hot local key                 both active,
                                         R resists migration

low        EVICT                         MIGRATE
           E   high, M   0              E   low, M   high
           cold, not remote              misplaced key
```

4.4 4.4 Migration Target Selection

For N > 2 nodes, a probabilistic HotNode mechanism identifies the migration target:

- On remote access from node N: with probability $p(S_1)$, set HotNode = N
- Low p = biased against migration change = built-in anti-ping-pong
- Sustained access from one node overcomes the bias

For 2 nodes: HotNode is implicit (the other node), no sampling needed.

4.5 4.5 Ghost-Less Reload Protection

When a page is evicted and reloaded from persistent storage, keys start with S_0 = MAX (protected from immediate re-eviction). This IS ARC's ghost hit promotion (back to t1/t2) without ghost lists — the "promotion" is just setting the initial score on reload.

4.6 4.6 Per-Key Storage Cost

Component	Size	Location
State (S_local, S_remote)	1 byte (4+4 bits)	TempCtrl in Page struct
HotNode	1 byte	KeyMeta in CMap slot
TempCtrlIdx	1 byte	KeyMeta in CMap slot
Total per key	**3 bytes**	

Compared to: - ARC (ArcList entry): 16 bytes per key - CrossNodeAccesses counter: 4 bytes per key - Per-node frequency vector: 2×(N-1) bytes per key

5 5. Evaluation

We evaluate REMARC through a comprehensive simulation study across three regimes: single-node (30+ variants, 5 workloads), 2-node NUMA (10 experiments, 4 workloads), and 3+ node NUMA (3 variants × 4 workloads with parameter sweep). All experiments use integer quantized state (uint8 per axis) with the REMARC update rules from §3.2.

5.1 5.1 Methodology

Workloads. We use five synthetic workloads that isolate distinct access patterns:

Workload	Description	Key property
Zipfian (=0.99)	Heavy-tailed popularity	Tests capacity management under skew
Temporal Shift	4-phase hot-set rotation	Tests adaptivity to distribution change
Scan-Resistant	90% sequential scan + 10% random	Tests resistance to sequential pollution
Looping	Cyclic access over 2× capacity	Tests structural awareness of phases

Parameters. The default REMARC configuration uses two axes (R, F) with parameters (bR, bF, rD, fD) = (8, 8, 8, 1024), where bR/bF are boost amounts and rD/fD are decay periods in operations. State is stored as uint8 with saturation at 255.

Baselines. We compare against: - **LRU:** Standard least-recently-used with no migration - **ARC:** Adaptive Replacement Cache with ghost lists (Megiddo & Modha, 2003) - **LRU + migration:** LRU eviction with threshold-based cross-node migration

Metrics. For single-node: hit rate per workload. For multi-node: weighted cost = L_local × local_hits + L_remote × remote_hits + L_miss × misses, with L_local=80, L_remote=150, L_miss=400.

5.2 5.2 Single-Node Study

5.2.1 5.2.1 Baseline Characterization

We tested 30+ REMARC variants including: bare REMARC (linear formula), factored REMARC (with G/H/Q statistics), logarithmic quantization (Log8), feedback-modulated proj_s (Q_r, Q_f weights), directional feedback (separate R/F ghost channels), Poisson REMARC (exponential decay kernel), Hawkes REMARC (sum-of-kernels), LogFreq REMARC (log-frequency axis), adaptive-weight REMARC (population-normalized), CoopLog (cooperative logarithmic), tiered REMARC (ARC-like capacity allocation with p-controlled T_R/T_F split), and multiple ghost variants (ghost frequency, ghost dormant-phi re-insertion).

Result (cap=1000, keys=10000, ops=200000):

Variant class	Zipfian HR	Temporal HR	ScanRes HR	Looping HR
LRU	10.1%	10.0%	10.1%	0.0%
ARC	44.7%	15.8%	90.5%	0.0%
REMARC best flat (12,8,16,1024)	26.6%	11.8%	90.4%	49.1%
REMARC + ghost restoration	27.5%	12.0%	90.3%	48.8%
REMARC Log8 (8,4,8,512)	24.3%	14.7%	84.2%	3.1%
REMARC tiered (p-adaptive)	24.5%	11.7%	85.3%	44.2%

5.2.2 5.2.2 Key Findings (Single-Node)

Finding 1 (Locality Barrier). No REMARC variant within the integer EMA per-key scoring class exceeds ~27.5% Zipfian hit rate. This ceiling is robust across all tested D_f (scalar scoring, Q-modulated scoring, directional scoring, capacity allocation). ARC achieves 44.7% through structural identity — its T1/T2 partition provides population-level information that per-key scalar scoring cannot express.

Finding 2 (Integer Quantization Is Load-Bearing). Replacing integer EMA with floating-point EMA collapses ScanRes hit rate from 90.4% to 48.2% across ALL formula variants. The discretization acts as a noise filter on the frequency axis, preventing scan pollution from inflating sF. This is a structural requirement, not a tuning choice.

Finding 3 (REMARC Dominates on Looping). REMARC achieves ~49% on the Looping workload where ARC and LRU score 0%. This is structurally impossible for list-based policies — a cyclic access over 2× capacity evicts phase N-1 items before phase N revisits them. REMARC's scoring retains high-R, high-F items from all phases simultaneously.

Finding 4 (Ghost Provides Marginal Single-Node Benefit). Ghost (dormant phi re-insertion) adds +1pp Zipfian over the best flat configuration. The mechanism is constrained: uniform phi produces homogeneous ghost populations, providing no directional information for proj_s. Ghost benefits ScanRes (+2pp) and Temporal (+3pp) but not Zipfian.

Finding 5 (Feedback Is Signal-Degenerate). Q-modulated proj_s weights and directional (Q_r, Q_f) feedback provide zero improvement over static proj_s. Cause: the REMARC state update phi produces uniform population statistics (G, H, Q) because all keys receive identical boost/decay treatment. Without discriminable sub-populations, feedback has no gradient to exploit.

Finding 6 (Tiered/Capacity Allocation Collapses). ARC-like T_R/T_F partition with p-controlled split causes p to always collapse to pMin. Ghost_R fills with cold revolving-door keys — recently evicted items from T_F re-enter T_R at zero cost, preventing meaningful capacity rebalancing.

5.2.3 5.2.3 Locality Barrier Analysis

The ~24.5% Zipfian ceiling is a property of the per-key scalar scoring class, not of REMARC specifically. Any policy where D_evict = g(S(K)) for scalar S(K) produces a total order on keys. ARC's T1/T2 partition breaks this total order — it maintains two populations with distinct admission criteria, providing strictly more information per capacity unit.

The barrier can be formalized: for a Zipf( ) distribution with → 1, the optimal fixed-threshold scalar policy captures the top-K keys by score. ARC's adaptive partition captures a different subset — recent-but-infrequent keys in T1 that would be evicted by any scalar policy but contribute to future hits via ghost-mediated promotion.

This barrier does NOT extend to multi-node decisions (§5.3), where REMARC's 2D scoring space provides information that list-based policies structurally cannot express.

5.3 5.3 Multi-Node Study: Desire Encoding

5.3.1 5.3.1 The Migration-Adaptation Gap

In the standard REMARC formulation, migration uses value-based scoring: a key migrates from node A to node B when its REMARC score on B exceeds a threshold. We tested this on a 2-node simulation with 4 workloads (Skewed-Zipf 80% remote, Node-Shift 4-phase, Cross-Loop 50/50, Oscillating 10K phase).

Finding 7 (Value-Based Migration Fails on Phase Shifts). On Node-Shift (where the hot set moves from node 0 to node 1 every 50K ops), value-based migration produces cost identical to no-migration (422.6 vs 422.9). Cause: slow-decaying sF creates "frequency momentum" — phase-0 keys retain sF 133 after phase shift, while newly-arriving phase-1 keys start at sF 80. The migration threshold prefers high-score keys that are stale.

5.3.2 5.3.2 Desire Encoding

We introduce **desire encoding**: instead of scoring keys by their value on the local node, each node maintains a **shadow desire map** S_i(K) = "how much does node i want key K." The desire score uses the same REMARC update rules (boost on access, periodic decay) but applies to ALL keys, not just cached keys.

Formally, for node i with cache capacity C_i:

$$\text{Cache}_i = \text{top-}C_i \text{ keys by } S_i(K)$$

$$\text{Migrate}(K : i \to j) = S_j(K) > \min_{K' \in \text{Cache}_j} S_j(K')$$

No migration threshold is needed. The eviction decision (remove lowest-desire key from cache) and migration decision (acquiring node's desire exceeds its worst cached key) are both driven by the same signal.

Finding 8 (Desire Encoding: 3-4× Improvement). Desire encoding reduces cost by 3-4× across all workloads:

Policy	Skewed-Zipf	Node-Shift	Cross-Loop	Oscillating
LRU + migration(3)	353.3	381.5	80.1	385.5
REMARC value-based	337.5	422.6	80.1	422.7
Desire-encoded	**87.2**	**96.1**	**80.1**	**104.9**
Desire + adaptive cap	87.3	96.2	80.1	108.0

On Skewed-Zipf, desire encoding reduces cost from 337.5 to 87.2 — a 3.87× improvement. The cross-loop workload is saturated (all policies at 80.1) because both nodes have sufficient capacity for their working sets.

5.3.3 5.3.3 Independence Theorem

The choice between value encoding and desire encoding has fundamental implications for per-node independence.

Definition. A multi-node REMARC system is **per-node independent** if each node's eviction decisions depend only on its local state S_i, and its migration decisions depend only on S_i and the migrating node's S_j.

Theorem (Independence). Desire encoding preserves per-node independence. Value encoding does not.

Proof sketch. Under desire encoding, S_i(K) depends only on node i's observation history (when K was accessed on node i). The migration decision S_j(K) > min S_j(Cache_j) evaluates only node j's desire — no cross-node state comparison is needed. Under value encoding, S_i(K) = "value of K on node i" requires comparing K's value across nodes to decide migration, creating a dependency on S_j during node i's state update.

This means desire-encoded REMARC scales to n nodes without per-key cross-node coordination. Each node maintains its own shadow desire map independently, and migration emerges from the conjunction of local decisions.

5.4 5.4 Multi-Node Study: n 3 Nodes

5.4.1 5.4.1 Global State for Contention Resolution

For n=2 nodes, desire encoding alone is sufficient — the two nodes' desires are complementary by construction (a key is either local or remote). For n 3, **global state G** becomes necessary to resolve contention.

We define G as a cross-node desire comparison: migrate key K from node i to node j only if:

$$S_j(K) > r \cdot S_i(K)$$

where r 1 is the compare ratio. G is O(1) per decision (one lookup in node i's desire map) and O(n) in total state (n desire maps of size |keyspace|).

5.4.2 5.4.2 Three-Node Results

Configuration: cap=67 per node, 999 keys, 500K ops. Three variants: desire-only (no G), desire + compare (ratio r), and desire + cooldown (time-based migration lockout).

Policy	Zipfian Cost	Cross-dominant Cost	Rotating Cost
Desire-only (no G)	324.8	296.2	296.4
Compare(1.1×)	141.1	130.6	144.4
Compare(1.3×)	134.4	126.9	137.4
Compare(1.5×)	132.5	125.6	134.9
Compare(2.0×)	130.6	124.5	132.3
Cooldown(3K ops)	212.9	225.0	201.1

Finding 9 (Global State Necessary for n 3). Without G, shared hot keys bounce between nodes (158K migrations, cost 325). With compare ratio 1.1×, cost drops to 141 — a 2.3× improvement. Cooldown is strictly dominated by any compare 1.1×.

Finding 10 (Compare Ratio: 1.1× Is the Cliff). The ratio sweep shows a sharp cliff at 1.1× (cost drops from 290 to 141 on Zipfian). Returns diminish above 1.3×. The optimal range is 1.1×–1.5×. Higher ratios reduce migration count but increase miss rate.

Finding 11 (65% Remote Hit Rate Is Structural). On 3-node Zipfian, the best variants achieve ~66% remote hit rate and ~33% local. This is a capacity problem: cap/3 = 67 keys per node vs 999 total keys (6.7% capacity). The Zipfian distribution concentrates ~50% of accesses on ~30 keys, but the remaining 50% is spread across 969 keys. Dynamic per-node capacity allocation is the natural next step.

5.5 5.5 Ghost Map Study

The desire-encoded policy maintains a shadow desire map for ALL keys (not just cached keys). This is effectively an unbounded ghost map — evicted keys continue to accumulate desire scores. We isolate the ghost map's contribution by comparing:

1. **Cache-only**: Scores only for keys currently in cache (like standard REMARC)
2. **Full ghost**: Scores for all keys (shadow desire map)
3. **Ghost-decay-cached**: Scores for all keys, but decay only applied to cached keys

Finding 12 (Ghost Map Marginal on Zipfian). At cap=1000, keys=10000 (10% ratio), the ghost map adds only +1.5pp Zipfian (26.48% → 28.00%). At cap=200, keys=1000 (20% ratio), it adds ~0pp (39.71% → 39.67%).

Policy	Zipf (10%)	Temp (10%)	Scan (10%)	Loop (10%)
ARC	**50.81**	**26.60**	70.76	0.25

Policy	Zipf (10%)	Temp (10%)	Scan (10%)	Loop (10%)
Cache-only	26.48	12.54	**90.37**	**3.02**
Full ghost	28.00	12.37	90.42	0.48

Finding 13 (Ghost Helps Temporal, Hurts Looping). The ghost map improves Temporal hit rate by +14pp (at 20% ratio) — persistent desire scores survive hot-set shifts, allowing rapid re-admission of previously-hot keys. However, it hurts Looping by -7pp — phantom desire from old phases competes with current-phase keys, polluting the cache during phase transitions.

Implication: The shadow desire map's primary value is not single-node hit rate improvement — it is the mechanism that enables desire encoding for multi-node migration (§5.3). The ghost map provides the cross-node desire signal that makes threshold-free migration possible.

5.6 5.6 Furrballs Integration

REMARC with desire encoding is designed for integration into Furrballs, a NUMA-aware caching library. This section describes the practical architecture, costs, and expected benefits.

5.6.1 5.6.1 Architecture

The integration requires three components per NUMA node:

Shadow Desire Map. Each node maintains `std::unordered_map<HashPair, uint8_t[2]>` — two bytes per key (sR, sF). The map tracks ALL keys ever accessed on this node, not just cached keys. On Get() or Set(), the desire score is updated with the standard REMARC boost/decay rules.

Migration Decision. In the existing Furrballs Get() path, a rate-limited pre-update migration check already exists. With desire encoding, this check becomes:

```
if (remote_key_accessed) {
    if (S_local(K) > min_desire(local_cache) * compare_ratio) {
        // Migration is beneficial: local desire exceeds worst cached key
        schedule_migration(K);
    }
}
```

No threshold tuning is needed. The compare_ratio is set to 1.3× for n 3 nodes (or 1.0× for n=2).

Global State G. For n 3 nodes, a shared array of {nodeId, minDesire[cap]} is maintained. Before migration, the accessor node queries the owner node's desire score for K. This is O(1) — one hash map lookup on the owner node.

5.6.2 5.6.2 Storage and Computation Cost

Component	Per-node cost	Notes
Shadow desire map	2 bytes × |keyspace|	For 10M keys: ~20 MB per node
Cache REMARC state	2 bytes × capacity	Already exists in Furrballs
Global state G	O(n) shared array	Compare ratio, no per-key state
Get() overhead	1 hash map update + comparison	~50ns additional per access
Migration check	1 min-finding in cache	Rate-limited (every 64 ops)

The shadow desire map is the dominant cost. For a 10M-key space across 2 nodes: ~40 MB total. This is comparable to ARC's ghost list overhead (which stores 2× capacity entries) but grows with key space rather than capacity.

Optimization. The shadow desire map can be periodically compacted: keys with sR = 0 and sF = 0 for T consecutive compaction intervals can be removed, as their desire score provides no information. This bounds the map size to O(|active keys|) rather than O(|keyspace|).

5.6.3 5.6.3 Expected Benefits

Based on the simulation results (§5.3, §5.4), the expected benefits for Furrballs on a 2-socket c6i.metal:

1. **Cross-node access reduction:** Desire encoding reduces cost by 3-4× on skewed workloads (87.2 vs 337.5 on Skewed-Zipf). This translates directly to reduced inter-socket traffic.
2. **No threshold tuning:** Migration decisions emerge from the desire signal — no manual threshold configuration needed. The compare_ratio for n 3 has a wide effective range (1.1×–1.5×).
3. **Per-node independence:** Each node operates on its local desire map. No distributed consensus or cross-node locking for eviction decisions.
4. **Structural Looping advantage:** REMARC's ~49% Looping hit rate (vs 0% for ARC/LRU) provides resilience against cyclic access patterns common in batch processing workloads.
5. **ScanRes parity:** REMARC matches ARC on ScanRes (~90%) without requiring ghost lists or admission gates.

5.7 5.7 Summary of Findings

#	Finding	Implication
1	Locality barrier: ~24.5% Zipfian ceiling for per-key scalar scoring	REMARC's strength is multi-node, not single-node
2	Integer quantization is load-bearing	Float EMA collapses ScanRes; discrete state is structural
3	REMARC dominates Looping (49% vs 0%)	Structural advantage for phase-based workloads
4	Ghost provides marginal single-node benefit (+1pp)	Ghost's value is multi-node (desire signal)
5	Feedback is signal-degenerate	Uniform phi → no gradient for feedback
6	Tiered capacity collapses	p → pMin; revolving-door prevents rebalancing
7	Value-based migration fails on phase shifts	Frequency momentum prevents adaptation
8	Desire encoding: 3-4× cost reduction	Threshold-free migration from local desire signal
9	Global state necessary for n 3	Cross-node compare prevents thrashing
10	Compare ratio: 1.1× is the cliff	Wide effective range (1.1×–1.5×)
11	65% remote rate is structural (capacity)	Dynamic allocation is the next step
12	Ghost marginal on Zipfian (+1.5pp)	Shadow map's value is multi-node
13	Ghost helps Temporal, hurts Looping	Workload-dependent; not universally beneficial
14	ARC dominates single-node Zipfian (2×)	REMARC designed for NUMA; single-node is wrong benchmark
15	Per-node independence holds for desire encoding	Scales to n nodes without coordination
16	G is O(1) per decision, O(n) state	Practical for production deployment

---

6 6. Generalization (K > 2)

The K=2 NUMA instantiation rests on a single structural property: the state space is a competition between sources. Observing source j boosts S_j and decays all S_k ($k \neq j$). This produces opposing gradients — high S_j means "source j is active," low S_j means

"source j is inactive." Every binary decision (keep/remove, promote/demote, move/stay) is a projection of those opposing gradients onto an action score.

This structure generalizes trivially to any K. The switched linear system (§3.1), convergence guarantee (§3.3), and steady-state encoding (§3.3) hold for arbitrary K — they depend only on the eigenvalue structure of diagonal M_j, not on dimensionality. What changes with K is:

1. **State dimensionality:** K sources instead of 2
2. **Action count:** M grows with K (more tiers/nodes = more actions)
3. **Lookup table size:** MAX^K entries per action
4. **Projection vectors:** domain-specific per instantiation

This section provides complete instantiations for K=3 and K=4, including explicit action score formulas and priority ordering analysis.

6.1 6.1 Tiered Storage (K=3, M=4)

Scenario: A cache spans DRAM (fast, local) and NVM (slow, persistent) on a NUMA system. Items can be in DRAM or NVM, and accessed locally or remotely.

6.1.1 Signal Sources

Source	Meaning	Update on
S_0	DRAM-level access rate	Access from DRAM-resident code path
S_1	Remote NUMA access rate	Access from another NUMA node
S_2	NVM/persistence access rate	Access from persistence layer or cold scan

The update rule is identical to K=2 (§3.1): observing source j boosts S_j by α_j, decays all others by λ. The steady-state formula (§3.3) holds per-source.

6.1.2 Decision Surfaces

Define normalized quantities:

$$T = \frac{S_0 + S_1 + S_2}{3 \cdot \text{MAX}}, \quad R_{\text{local}} = \frac{S_0}{\text{MAX}}, \quad R_{\text{remote}} = \frac{S_1}{S_0 + S_1 + \epsilon}, \quad R_{\text{nvm}} = \frac{S_2}{S_0 + S_2 + \epsilon}$$

Four action scores:

$$\text{Evict}(\mathbf{S}) = (1 - T)$$

$$\text{Migrate}(\mathbf{S}) = T \cdot R_{\text{remote}} \cdot (1 - R_{\text{local}})$$

$$\text{Promote}(\mathbf{S}) = T \cdot (1 - R_{\text{nvm}}) \quad \text{[conditional: item in NVM tier]}$$

$$\text{Demote}(\mathbf{S}) = (1 - R_{\text{local}}) \cdot R_{\text{nvm}} \quad \text{[conditional: item in DRAM tier]}$$

Interpretation:

- **Evict:** high when all signals are low. Total coldness. Same logic as K=2.
- **Migrate:** high when remote signal dominates local. Same logic as K=2, ignoring NVM dimension.
- **Promote:** high when item has activity from fast sources relative to NVM. Conditional on item being in NVM — can't promote what's already in DRAM.
- **Demote:** high when local signal is low but NVM signal exists. Conditional on item being in DRAM — can't demote what's already in NVM.

6.1.3 State Space Partition (K=3)

```
              S_0 (DRAM) →
              low                 high

low       EVICT               DEMOTE
S_2       E high, all cold    D moderate
                              DRAM resident
                              but NVM-bound

high      PROMOTE + MIGRATE   KEEP
          P high (in NVM),    all active
          M high if remote    E  0

         (S_1 axis: remote access pulls toward MIGRATE)
```

6.1.4 Lookup Tables

State is packed into 12 bits (4×3). Lookup tables: $15^3 = 3375$ entries per action (4 tables). Total: $4 \times 3375 = 13500$ entries 13.5 KB — fits in L1 cache.

6.2 6.2 Heterogeneous Memory (K=4, M=5)

Scenario: System with local DRAM, remote DRAM, HBM (high-bandwidth memory), and persistent memory. Items can reside in any tier and be accessed from any source.

6.2.1 Signal Sources

Source	Meaning
S_0	Local DRAM access rate
S_1	Remote DRAM access rate
S_2	HBM access rate
S_3	Persistent memory access rate

6.2.2 Decision Surfaces

Define:

$$T = \frac{S_0 + S_1 + S_2 + S_3}{4 \cdot \text{MAX}}, \quad R_{\text{local}} = \frac{S_0}{\text{MAX}}, \quad R_{\text{remote}} = \frac{S_1}{S_0 + S_1 + \epsilon}, \quad R_{\text{fast}} = \frac{S_0 + S_2}{S_0 + S_2 + S_3 + \epsilon}$$

Five action scores:

$$\text{Evict}(\mathbf{S}) = (1 - T)$$

$$\text{Migrate}(\mathbf{S}) = T \cdot R_{\text{remote}} \cdot (1 - R_{\text{local}})$$

$$\text{Promote to HBM}(\mathbf{S}) = T \cdot (1 - R_{\text{fast}}) \quad \text{[conditional: item in DRAM/PMEM, not HBM]}$$

$$\text{Promote to DRAM}(\mathbf{S}) = T \cdot \frac{S_0}{S_0 + S_3 + \epsilon} \quad \text{[conditional: item in PMEM]}$$

$$\text{Demote to PMEM}(\mathbf{S}) = (1 - T) \cdot \frac{S_3}{\text{MAX}} \quad \text{[conditional: item in DRAM/HBM]}$$

6.2.3 Lookup Tables

$15^4 = 50625$ entries per action (5 tables). Total: $\approx 250K$ entries 250 KB — fits in L2 cache.

6.3 6.3 Priority Ordering for K > 2

The priority ordering resolves conflicts when multiple actions trigger simultaneously for the same item.

Key property: tier-conditional actions are disjoint. Promote applies only to items in lower tiers; Demote applies only to items in higher tiers. An item cannot be both promoted and demoted in the same scanner pass — the actions are conditioned on the item's current tier, not just its scores. This eliminates the "partial inverse" concern: promoting key A from NVM and demoting key B to NVM in the same pass is correct behavior (A is heating up, B is cooling down), not circular movement.

The same item cannot participate in opposing tier movements. This is guaranteed by the tier-conditional guards, not by score thresholds. A sufficiently high Promote score on an item already in DRAM is a no-op. A sufficiently high Demote score on an item already in NVM is a no-op.

General priority ordering principle: most reversible first, most irreversible last.

Priority	Action	Reversibility
1	Promote	Reversible: can demote back
2	Migrate	Reversible: can migrate back
3	Demote	Reversible: can promote back
4	Evict	Irreversible: must re-fetch from backing store

Justification: When both Migrate and Evict trigger for the same item (high remote signal + low total activity), the item is still being accessed (evidenced by the migration score). Evicting would force a cache miss on next access. Migrating preserves the item while fixing placement. The reversibility ordering ensures the least destructive action wins every conflict.

Conflict analysis for all pairwise interactions (K=3):

Conflict	Resolution	Reasoning
Promote vs Migrate	Promote first	Fix tier placement before fixing node placement
Promote vs Demote	Cannot conflict	Tier-conditional: disjoint item sets
Promote vs Evict	Promote first	Promote implies item has activity (T not zero)
Migrate vs Demote	Migrate first	Fix node before demoting — wrong-node item shouldn't be demoted to wrong-tier
Migrate vs Evict	Migrate first	Same as K=2: preserve active items

Conflict	Resolution	Reasoning
Demote vs Evict	Demote first	Demote preserves item in a cheaper tier; evict loses it entirely

6.4 6.4 Beyond Competitive Dimensions

The preceding sections assume competitive dimensions: evidence for source j is evidence against source k. This is the structure cache access patterns produce (local and remote access compete for the same cache slot). But REMARC (the projection function) does not require competition. It is a linear map $f : \mathbb{R}^K \to \mathbb{R}^M$ — the nature of the input dimensions determines the appropriate projection coefficients, not the projection itself.

Recenter each axis around its midpoint to make the state signed: $\delta_j = S_j - \text{MAX}/2$. The projection's behavior along each axis is characterized by:

$$f(\delta) + f(-\delta) = c$$

Three regimes:

Regime	Symmetry	Input Structure	Projection Behavior
Competitive ($c = 0$)	Odd: $f(-\delta) = -f(\delta)$	Evidence for j opposes evidence for k	Imbalance = positive score. Current REMARC.
Independent ($c = \text{const}$)	Affine: $f(-\delta) = -f(\delta) + c$	Dimensions uncorrelated	Each axis contributes independently. Baseline c captures the constant component.
Cooperative ($c = 2f(\delta)$)	Even: $f(-\delta) = f(\delta)$	Evidence for j reinforces evidence for k	Agreement = positive score. Projection reads consensus, not imbalance.

The linear projection $A \cdot S + b$ can express all three regimes by choosing A (the projection vectors) and b (the bias): - Competitive: $b = 0$, A encodes difference (imbalance) between axes - Independent: $b \neq 0$, A encodes each axis separately - Cooperative: A encodes sum (agreement) between axes

REMARC's current instantiations (§4–§6) use the competitive regime exclusively. The independent and cooperative regimes are mathematically well-defined but their application domains (e.g., multi-objective optimization, sensor fusion) and projection coefficient derivation are left as future work.

6.5 6.5 Theoretical Properties for General K

- **Convergence:** eigenvalues of all M_j satisfy $|\lambda| \leq 1 \rightarrow$ contractive $\rightarrow$ convergent for any K
- **Steady state:** closed-form for each source's rate (generalization of §3.3): $S_j^* = \text{MAX} \cdot p_j\alpha_j/(p_j\alpha_j + (1-p_j)(1-\lambda))$. Holds for any K because each source's steady state depends only on its own α_j and the global λ.
- **Competitive erosion scales:** the boost-one-decay-rest structure is inherently competitive. For any K, the state vector $(S_0, \ldots, S_{K-1})$ encodes the relative activity rates of all sources. Action scores are projections of this relative activity. The structure does not depend on K=2.
- **Decision surface design:** REMARC uses hand-designed projections from domain knowledge (e.g., evict = total coldness, migrate = locality imbalance, promote = fast-tier demand relative to slow-tier). For a data-driven alternative, one could compute the eigenvectors of the access covariance matrix and use those as projection vectors, analogous to PCA. This is an open direction (§8).
- **Lookup table scaling:** MAX^K entries per action. Feasibility threshold: K=4 (50K entries, L2 cache). Beyond K=4, online computation replaces lookup tables — the projection is still $O(K)$ arithmetic per action.

6.6 6.6 Comparison with Existing Multi-Policy Approaches

Approach	Eviction	Migration	Tiering	Shared State?
ARC + bolt-on migration	ARC lists	Separate counter + threshold	None	No
LRU + tiering policy	LRU stack	None	Separate tier manager	No
W-TinyLFU + NUMA	Frequency sketch	None	None	No
REMARC	**Unified state**	**Unified state**	**Unified state**	**Yes**

---

7 7. Related Work

7.1 7.1 Cache Replacement Policies

- **ARC** (Megiddo & Modha, 2003): Adaptive replacement with ghost lists. 1D (recency+frequency blend). No migration.
- **LIRS** (Jiang & Zhang, 2002): Low inter-reference recency set. 1D (reuse distance). No migration.
- **W-TinyLFU** (Einziger & Friedman, 2017): Frequency estimation via Count-Min Sketch. 1D (frequency). No migration.

- **Clock-Pro** (Jiang et al., 2005): Clock algorithm with non-resident pages. 1D. No migration.
- **LeCaR** (Vietri et al., 2018): Uses regret minimization to adaptively combine LRU and LFU via online learning. The closest prior work to REMARC's "adaptive policy that blends multiple signals" — but LeCaR blends two 1D eviction policies, while REMARC projects K signals onto M action axes. LeCaR's W-TinyLFU window is a special case of REMARC's state space with K=1.
- **CACHEUS** (Cheng et al., 2021): Learned cache replacement using workload features. Maps workload characteristics to policy parameters. REMARC differs by maintaining per-item state (not per-workload) and producing continuous action scores (not discrete policy selection).
- **Glaze** (Liu et al., 2023): Tiered caching with learned placement decisions. Directly competes with REMARC's K=3 tiered storage generalization (§6.1). Glaze uses a separate ML model per tier; REMARC uses a unified state space across all tiers.
- **S3-FIFO** (Yang et al., SOSP 2023): Three FIFO queues (small, main, ghost) with a single-bit reference counter for promotion. Structurally different from other policies: the binary frequency gate is the sharpest possible threshold (one reaccess = promote). REMARC approximates this as a boundary case (§3.4, coverage table) — the steep EMA ($\alpha \to 1$, fast λ) approaches binary behavior with quantifiable error. S3-FIFO's ghost queue serves the same role as REMARC's ghost-less reload protection (§4.5).

7.2 7.2 NUMA-Aware Systems

- **TCMalloc / jemalloc:** NUMA-aware allocation, no cache policy.
- **CacheLib (Meta):** Static per-node sharding. No adaptive migration.
- **PostgreSQL / InnoDB:** Buffer pool with partition locks. Not standalone cache.

7.3 7.3 Dimensionality Reduction

- **PCA** (Pearson, 1901): Eigenvalue decomposition of the covariance matrix → principal components. REMARC's projection is structurally similar (linear projection from K to M dimensions) but the projection vectors are hand-designed from domain knowledge, not derived from the data covariance. A data-driven variant using access-pattern covariance eigenvectors is discussed as future work (§6.3).
- **SVD** (Singular Value Decomposition): General matrix factorization. REMARC's precomputed lookup tables can be viewed as a discretized low-rank approximation of the full decision function.
- **Autoencoders (ML):** Neural network-based dimensionality reduction. REMARC's lookup tables are a non-learned (analytically derived) analog — the "encoding" is the switched linear system, and the "decoding" is the linear projection.

7.4 7.4 Information-Theoretic Connections

- **Fisher's Linear Discriminant** (Fisher, 1936): A linear projection from K-dimensional observation space onto a lower-dimensional space that maximally

separates classes. REMARC's projection is structurally analogous — a domain-defined (rather than data-driven) discriminant that maps K-dimensional state to M action scores. The key difference: LDA is batch and requires labeled data; REMARC is online, sequential, and uses no labels.

- **Information Bottleneck** (Tishby et al., 1999): Finds a compressed representation T of input X that preserves information about output Y: minimize $I(X;T)$ subject to $I(T;Y) \geq$ constraint. REMARC's correlation condition (§3.4) is a deterministic algebraic analog of this constraint — every input dimension must carry non-zero information about at least one output discriminator. The Information Bottleneck requires knowing the joint distribution $P(X,Y)$; REMARC achieves compression without distributional assumptions, through algebraic structure alone. Whether REMARC's deterministic construction achieves information-bottleneck-optimal compression for specific discrimination problems is an open question.
- **Sufficient Statistics** (Fisher, 1922; Neyman, 1935): The concept that a compressed representation retains all decision-relevant information. REMARC's state vector is a sufficient statistic for the discrimination (under the Weak REMARC condition). Classical sufficient statistics are defined probabilistically; REMARC's version is algebraic — no distribution required.
- **Switched Linear Systems** (Sun & Ge, 2005): Convergence of systems that switch between linear update rules. The joint spectral radius (JSR) characterizes stability. REMARC applies this to cache policy analysis for the first time — the diagonal structure of M_j makes the JSR computable in closed form (§3.3).

---

8 8. Conclusion

REMARC reframes cache replacement as a dimensional reduction problem: multiple access signals are compressed into a compact state space, and action scores are derived via precomputed projection functions. This unified framework handles eviction, migration, and tiering from a single mechanism, with O(1) amortized cost per decision via SIMD lookup tables.

The evaluation (Section 5) establishes three principal results across single-node (30+ variants, 5 workloads), 2-node (10 experiments, 4 workloads), and 3-node (3 variants, 4 workloads) configurations:

1. **Single-node ceiling ($ 24.5 $90%).
2. **Desire encoding breaks the multi-node barrier:** By reformulating the state as per-node desire ($S_i(K)$ = "how much does node i want key K") rather than cross-node value, migration decisions become threshold-free conjunctions of independent local decisions. This produces 3–4× cost reduction on skewed multi-node workloads and preserves per-node independence for n nodes (Theorem, Section 5.3.3). The shadow desire map costs 2 bytes per key per node.

3. **Scalable global state for $n \geq 3$:** A cross-node desire comparison ($O(1)$ per decision, $O(n)$ state) resolves contention for shared hot keys. The compare ratio r has a sharp cliff at 1.1× and a wide effective range (1.1×–1.5×), making the system robust to parameter selection.

The mathematical framework (switched linear system with linear projection) provides convergence guarantees, steady-state characterization, and natural generalization to $K > 2$ for tiered storage and heterogeneous memory hierarchies (Section 6). The Furrballs integration architecture (Section 5.6, Appendix A) demonstrates practical deployment: 2 bytes per key shadow map, lazy migration on Get(), SIMD-optimized periodic decay. The desire encoding mechanism requires no distributed coordination, making it suitable for shared-memory NUMA systems and potentially extensible to distributed caches via the factorization framework (Section 9.1).

---

9 9. Open Questions and Research Roadmap

The preceding sections establish REMARC as a projection function operating on competitively-coupled state spaces. This section identifies three independent design axes that emerge from the framework, each with open questions.

9.1 9.1 Factorization: Composing REMARC Instances

A unified K-dimensional REMARC requires lookup tables of size MAX^K per action. For K > 4 this exceeds practical cache sizes. An alternative is to factor the K-dimensional state into multiple smaller REMARC instances connected by shared axes.

Example: K=3 state `(local_DRAM, remote_DRAM, local_NVM)` can be factored into:

- Instance A: `{local_DRAM, remote_DRAM}` — NUMA decisions
- Instance B: `{local_DRAM, local_NVM}` — tiering decisions

Shared axis: `local_DRAM`. An access from local_DRAM updates both instances, creating implicit coupling.

Approximation quality: erosion through shared axes is exact. Erosion between non-shared axes is lost. In the example above, a `remote_DRAM` access erodes `local_DRAM` in instance A but does not erode `local_NVM` in instance B. The factorization is an approximation of the unified K=3 system whose quality depends on the coupling between non-shared dimensions.

Structural interpretation: this is a graphical model. Each REMARC instance is a clique; shared axes are separator sets. The factorization graph determines which cross-dimensional erosion is captured exactly. A fully connected graph recovers the unified system.

Open questions: - For a given workload, what is the optimal factorization graph? (Minimizes approximation error subject to table size budget) - Can the factorization be learned online from access patterns? - What is the relationship between factorization topology and the symmetric properties described in §6.4 (competitive, independent, cooperative)?

9.2 9.2 Bit Width: Quantization Granularity

REMARC's current instantiation uses 4-bit quantization (MAX=15). This limits the state space to 16 levels per dimension — sufficient for eviction and migration thresholds but coarse for fine-grained tiering decisions where the difference between "warm" and "hot" matters.

Increasing bit width is straightforward for K=2 ($256^2 = 65K$ entries, fits L2). For K > 2, bit width interacts with table size: K=3 at 8-bit produces $16M$ entries per action, which is impractical.

Factorization decouples bit width from dimensionality. With factored K=2 instances (§9.1), each instance uses 8-bit tables ($65K$ entries) regardless of total K. The total number of dimensions is absorbed by running more instances, not by growing individual tables.

Open questions: - What is the sensitivity of eviction/migration accuracy to quantization level? (4-bit vs 6-bit vs 8-bit on real workloads) - Does the competitive erosion structure (§3.1) amplify or dampen quantization error? - Can adaptive quantization (different bit widths per dimension) improve accuracy without increasing table size?

9.3 9.3 Dynamic K: Topology Changes at Runtime

REMARC assumes K is fixed at compile time. Real systems experience topology changes: hot-plugged memory tiers, VM migration across NUMA nodes, added or removed caching layers.

Approaches:

1. **Preallocate K_max, activate subsets.** Always allocate state for the maximum expected K. Unused dimensions stay at 0 and contribute nothing to scores. Simple but wastes memory.
2. **Runtime table generation.** Regenerate lookup tables when K changes. For K=4, this is ~250K entries — computable in microseconds. Tables are static per-epoch (regenerate on topology change, not per-access). Loss of `constexpr` but practical for rare topology events.
3. **Factorization with dynamic instance graph.** Add or remove K=2 instances when topology changes. Each new tier or node adds at most one new instance (connected to an existing shared axis). This avoids regenerating large tables and avoids migrating state for all keys — only keys in the new tier need initialization.

Open questions: - State migration: when K increases, every existing key's state vector must grow. What is the cost model? Can migration be lazy (grow on next access)? - Does runtime table generation introduce latency spikes that affect cache performance during topology transitions? - Can the factorization graph be updated incrementally (add/remove cliques) without recomputing the entire structure?

9.4 9.4 The Design Space

These three axes are independent:

Axis	Controls	Tradeoff
Total dimensions K	Number of signals tracked	More = richer decisions, larger tables
Bit width	Quantization granularity per signal	More = finer decisions, exponentially larger tables
Factorization	How dimensions group into instances	More factorized = cheaper tables, approximate erosion

Increasing K does not require increasing bit width. Increasing bit width does not require increasing K. Refactoring the instance graph does not require changing either. A system builder selects a point in this three-dimensional design space based on their hardware constraints (cache sizes, SIMD support) and workload requirements (number of tiers, required precision).

This design space is the natural generalization of REMARC beyond a single cache policy into a family of related policies, connected by the shared mathematical structure of competitive coupling and linear projection.

9.5 9.5 Toward an Algebra of Sequential Decision Functions

The representation theorem (§3.4) and factorization analysis (§9.1) suggest that REMARC is not merely a cache policy framework but the **algebraic structure underlying the class of history-based online sequential decision functions**. This section outlines the theoretical direction; full formalization is planned as Paper 3.

9.5.1 REMARC as Projection Space

A REMARC function is an approximation (a decision function) of a policy. The space of REMARC configurations is a **projection space** for the space of sequential decision functions. This supports two operations:

1. **Constructive:** build any sequential decision function by composing phi atoms with different (,) parameters, then applying proj_s and decide.

2. **Projective:** map any existing policy to a REMARC approximation with characterizable error. The approximation inherits REMARC's computational properties (O(1) per decision, SIMD, deterministic convergence).

The projective property is the stronger statement. Every policy maps to a REMARC point:

$$\pi \mapsto (\varphi, \texttt{proj}_s, \texttt{decide}) = (K, \{\alpha_j\}, \lambda, A, g)$$

Navigating the REMARC space (adjusting the three atoms) is **policy optimization** — searching for the composition that minimizes a cost function over the workload.

9.5.2 Generation Theorem

The K=1 REMARC function is the **atom** of the decision function space. The atom is: "track one exponentially-smoothed signal, decay it over time." Composing enough atoms with different and values, projecting their outputs, and applying nonlinear decision surfaces can approximate a broad class of sequential decision functions. Whether the three-atom composition is universal (generates *all* sequential decision functions) or dense (generates a class dense in the target space) is the central open question for Paper 3.

$$\pi_c = \texttt{decide} \circ \texttt{proj}_s \circ \varphi^t$$

Where the three atoms are (as defined in §3.4):

Atom	Type	Free parameters
phi: S x O -> S	Incremental state update	n (dimension), _j (boost), (decay), MAX (bound)
proj_s: S -> R^k	Linear projection (scalar product)	k (feature count), A (projection matrix)
decide: R^k -> D	Nonlinear combination	g (decision surface geometry)

The "spectrum" in REMARC is the time constant (,). Fast tracks recency; slow tracks frequency; controls memory. Composing across the spectrum yields the full range of sequential decision functions.

This decomposition is analogous to basis decompositions in other mathematical spaces:

Analogy	Basis element	Space generated
Fourier analysis	sin(n t), cos(n t)	L^2 functions on [0,T]
Polynomials (Weierstrass)	x	Continuous functions on [a,b]
REMARC	**(phi, proj_s, decide)**	**Sequential decision functions**

9.5.3 Decomposition and Factorization

In the three-atom decomposition (§3.4), phi is the only atom that varies across parallel instances (different , per dimension). The proj_s and decide atoms are shared — they read from the K-dimensional state that the parallel phi instances collectively produce. So the decomposition is:

$$\pi_c = \texttt{decide} \circ \texttt{proj}_s \circ (\varphi_1 \otimes \varphi_2 \otimes \cdots \otimes \varphi_K)^t$$

Where each φ_i is a K=1 phi atom tracking one dimension. The factorization results from §9.1 apply at every level — proj_s itself can be decomposed into smaller REMARC instances connected by shared axes, approximating a higher-dimensional projection with compositions of lower-dimensional ones.

ARC is two K=1 phi atoms (recency tracker + frequency tracker) composed with a K=2 proj_s + decide that blends their outputs. ARC's adaptive parameter p is the projection vector in the meta-REMARC. The recency and frequency signals are independent (not competitive), placing ARC in the independent regime (§6.4) of the REMARC space — same algebraic structure as NUMA-REMARC, different symmetry stratum.

9.5.4 Symmetry Classification

§6.4 identified three regimes of the REMARC projection (competitive, independent, cooperative). These correspond to three symmetry classes of the policy space:

- **Competitive** ($f(\delta) + f(-\delta) = 0$): dimensions oppose each other. Examples: NUMA local/remote, eviction vs retention.
- **Independent** ($f(\delta) + f(-\delta) = c$): dimensions are uncorrelated. Examples: ARC's recency + frequency, LIRS tracking multiple unrelated metrics.
- **Cooperative** ($f(\delta) + f(-\delta) = 2f(\delta)$): dimensions reinforce each other. No known existing policy exploits this regime — it is a genuinely new region of design space that the algebraic framing opens naturally.

Existing policies occupy only the competitive and independent regimes. The cooperative regime is unexplored territory.

9.5.5 Three-Paper Structure

This analysis suggests a natural division of the work:

Paper	Scope	Contribution
Paper 1: Furrballs (systems)	Engineering	NUMA-aware cache implementation, real hardware validation, Phase 1 + 2a results

Paper	Scope	Contribution
Paper 2: REMARC (this paper, algorithm + framework)	Framework + instantiation	K=2 instantiation, SIMD implementation, K>2 generalization, representation theorem, evaluation
Paper 3: REMARC Theory (mathematical)	Pure theory	Three-atom decomposition (phi, proj_s, decide), independent space axioms (S compact, D compact ordered), generation theorem (π_c = decide . proj_s . φ^t), approximation bounds (error as function of K, bit width, factorization), geometric structure (stratified by K, parameterized by / /A), symmetry classification, sigma as corollary

Paper 2 establishes the framework and validates it on real hardware. Paper 3 formalizes the algebra that Paper 2 discovers. Different venues, different reviewers, different contribution types.

9.5.6 Open Questions for the Theoretical Paper

The following questions are ranked by dependency: each question depends on the ones above it.

1. Approximation bounds (load-bearing question). Given a policy π and a REMARC approximation with parameters (K, bit width, factorization graph), what is the maximum decision disagreement $d(\pi, R(\pi))$ between π and its REMARC projection $R(\pi)$? Does the error decrease monotonically with K?

Why this is load-bearing: every other question requires a well-defined error metric first. Optimal decomposition (Q2) asks for minimum K within error ε — but ε has no meaning without this bound. Factorization (Q3) trades approximation quality for computational cost — but the tradeoff is undefined without this bound. This question must be answered first.

Approach: define a metric on policy behavior (e.g., fraction of decisions that differ over a finite trace) and bound it as a function of K, bit width, and the distance between the policy's true decision surface and the REMARC projection. Known tools: quantization error analysis, function approximation theory.

2. Optimal decomposition (depends on Q1). For a given policy π, what is the minimum K needed for exact representation? For approximate representation within error ε?

Approach: characterize the class of decision surfaces achievable with K-dimensional REMARC. Exact representation requires the policy's decision surface to lie in the span of the REMARC projections. Minimum K = minimum dimension of a subspace containing the decision surface.

3. Factorization as approximation (depends on Q1). What is the relationship between factorization topology (tree, chain, star) and approximation quality?

Approach: the factorization graph determines which cross-dimensional erosion is preserved (shared axes) and which is lost (non-shared axes). The approximation error is the information lost through non-shared erosion. Bound this loss as a function of the graph's connectivity. Known tools: junction tree theory, conditional independence in graphical models.

4. Learnability (depends on Q1, Q2). Can the REMARC parameters (α, λ, A, g) be learned from workload traces? Is there an online learning algorithm that converges to the optimal REMARC composition for a given workload?

Approach: given the error metric from Q1, define a loss function over the parameter space and apply online convex optimization (if the loss is convex in the parameters) or bandit methods (if only noisy feedback is available). The convergence rate of the learning algorithm depends on the geometry of the parameter space (§3.4, geometric interpretation).

5. Cooperative regime (independent of Q1–Q4). Are there domains (beyond caching) where cooperative dimensions produce useful policies? What does a cooperative cache policy look like?

Approach: the cooperative regime ($f(\delta) + f(-\delta) = 2f(\delta)$, evidence reinforces evidence) is a genuinely unexplored region of the REMARC design space. No existing policy exploits it. Identify domains where signal agreement (rather than signal competition) is the decision-relevant structure. Candidates: sensor fusion, anomaly detection, consensus protocols.

6. Information Bottleneck connection (depends on Q1). Does REMARC's deterministic construction achieve information-bottleneck-optimal compression for specific discrimination problems? Is the correlation condition (§3.4) the algebraic analog of the IB mutual information constraint?

Approach: for a specific discrimination problem (e.g., K=2 eviction/migration), compute the IB-optimal compression and compare with the REMARC projection. If they coincide, the connection is proven for that case. General proof would require showing that the REMARC projection minimizes some deterministic analog of $I(X;T) - \beta I(T;Y)$.

10 Appendix A: REMARC-Furrballs Coupling

REMARC was designed within and validated through Furrballs, a NUMA-aware caching library. This appendix documents the integration architecture with desire encoding.

10.1 A.1 Infrastructure Layer

Furrballs provides the cache infrastructure: - **CMap**: Concurrent Swiss table (lock-free reads, striped writes) for key-value storage - **Per-node page allocation**: Pages are NUMA-local; key placement determines memory locality - **RocksDB persistence**: Transparent persistence layer below the cache - **HotNodes**: Per-key NUMA affinity tracking for request routing

10.2 A.2 Policy Layer (REMARC with Desire Encoding)

REMARC provides the policy through three per-node components:

In-cache REMARC state (existing). Each cached key carries a 2-byte REMARC score (sR, sF) packed into TempCtrl. The SIMD scanner (§3.5) evaluates eviction decisions in batch.

Shadow desire map (new). Each node maintains an `unordered_map<HashPair, uint8_t[2]>` for ALL keys accessed on that node — not just cached keys. This provides the cross-node desire signal for migration decisions.

Storage: 2 bytes × |active keys| per node. For 10M keys on 2 nodes: ~40 MB total. Periodic compaction removes zero-score entries to bound growth.

Global state G (for n 3). Before migrating key K from node i to node j, the accessor queries node i's desire score for K and checks S_j(K) > r × S_i(K) with compare ratio r [1.1, 1.5]. This is O(1) per decision — one hash map lookup on the owner node.

10.3 A.3 Get() Path

```
Get(key K):
  1. Route to home node (HotNodes affinity)
  2. Update shadow desire map: S_home(K).sR += bR, S_home(K).sF += bF
  3. If K in cache:
       - Update in-cache REMARC score
       - Return value
  4. If K not in cache:
       - Check migration: S_home(K) > r × S_owner(K)?
         If yes and migration budget available: schedule migration
       - Return miss
```

10.4 A.4 Migration Decision

Migration is lazy — triggered on Get() access, not by the scanner. This avoids the need for a reverse index (HashPair → key string) in the scanner path.

The migration budget is rate-limited (one migration per 64 ops) to prevent thrashing on oscillating workloads. With desire encoding, the budget is rarely the binding constraint — the desire signal naturally gates migration to genuinely beneficial cases.

10.5 A.5 Applicability Beyond Furrballs

The generalization in §6 shows that REMARC's policy framework is independent of Furrballs' infrastructure. Any cache system with per-item state storage can adopt REMARC. The desire encoding mechanism requires only: 1. Per-key state storage for the shadow desire map (2 bytes per key) 2. A mechanism to query remote nodes' desire scores (for n 3) 3. A capacity limit per node for the top-K cache selection

11 Appendix B: Lookup Table Construction

11.1 B.1 General Construction Algorithm

For any K with MAX=15, each action's lookup table maps a packed state tuple to a score. The state is packed by concatenating 4-bit fields:

$$\text{index} = S_0 + S_1 \cdot 16 + S_2 \cdot 16^2 + ... + S_{K-1} \cdot 16^{K-1}$$

For K=2 this produces indices in $[0, 255]$ (1 byte). For K=3, indices in $[0, 4095]$ (12 bits). For K=4, indices in $[0, 65535]$ (16 bits).

The general construction:

```
For each index i   [0, 16^K - 1]:
  // Unpack state
  temp = i
  for j in [0, K-1]:
    S[j] = temp & 0xF
    temp >>= 4

  // Compute normalized quantities (domain-specific per instantiation)
  T = sum(S) / (K × MAX)
  ... (see per-K formulas below)

  // Compute action scores
  Action_lookup[i] = formula(S) × MAX_SCORE
```

All tables are `constexpr` — constructible at compile time. The tables are static: they encode the decision surface geometry, which depends only on the formulas, not on the workload.

11.2 B.2 K=2 (NUMA, 2 tables × 256 entries)

```
For each byte b  [0, 255]:
  S_0 = b & 0xF       // S_local
  S_1 = b >> 4         // S_remote
  R = S_0 / 15
  F = (S_0 + S_1) / 30
  L = S_1 / max(S_0 + S_1, 1)
  E_lookup[b] = (1 - R) × (1 - F) × MAX_SCORE
  M_lookup[b] = F × L × (1 - R) × MAX_SCORE
```

State packing: single byte `(S_1 << 4 | S_0)`. SIMD: SSE2 nibble-unpack + 16-bit multiply arithmetic, ~18 instructions per batch of 16 keys. `pshufb` cannot be used for direct 256-entry lookup (4-bit index limitation).

11.3 B.3 K=3 (Tiered Storage, 4 tables × 3375 entries)

```
For each index i  [0, 4095]:
  S_0 = i & 0xF               // DRAM
  S_1 = (i >> 4) & 0xF        // Remote NUMA
  S_2 = (i >> 8) & 0xF        // NVM
  T = (S_0 + S_1 + S_2) / 45
  R_local = S_0 / 15
  R_remote = S_1 / max(S_0 + S_1, 1)
  R_nvm = S_2 / max(S_0 + S_2, 1)
  E_lookup[i] = (1 - T) × MAX_SCORE
  M_lookup[i] = T × R_remote × (1 - R_local) × MAX_SCORE
  P_lookup[i] = T × (1 - R_nvm) × MAX_SCORE
  D_lookup[i] = (1 - R_local) × R_nvm × MAX_SCORE
```

State packing: 12 bits per key. Scanner processes keys in groups where stride and masking extract 12-bit indices. Total storage: $4 \times 3375 = 13500$ entries 13.5 KB (fits L1).

11.4 B.4 K=4 (Heterogeneous Memory, 5 tables × 50625 entries)

```
For each index i  [0, 65535]:
  S_0 = i & 0xF               // Local DRAM
  S_1 = (i >> 4) & 0xF        // Remote DRAM
  S_2 = (i >> 8) & 0xF        // HBM
  S_3 = (i >> 12) & 0xF       // PMEM
  T = (S_0 + S_1 + S_2 + S_3) / 60
```

```
R_local = S_0 / 15
R_remote = S_1 / max(S_0 + S_1, 1)
R_fast = (S_0 + S_2) / max(S_0 + S_2 + S_3, 1)
E_lookup[i] = (1 - T) × MAX_SCORE
M_lookup[i] = T × R_remote × (1 - R_local) × MAX_SCORE
P_hbm_lookup[i] = T × (1 - R_fast) × MAX_SCORE
P_dram_lookup[i] = T × (S_0 / max(S_0 + S_3, 1)) × MAX_SCORE
D_pmem_lookup[i] = (1 - T) × (S_3 / 15) × MAX_SCORE
```

State packing: 16 bits per key. No SIMD shortcut — indices are too wide for per-instruction lookup. Scanner falls back to direct array indexing. Total storage: $5 \times 50625 = 253125$ entries 248 KB (fits L2).