Semantic Sharding and Distributed Query Planning: A Typed, Versioned Routing Contract for CatDB

Matthew Long
The YonedaAI Collaboration
YonedaAI Research Collective
Chicago, IL
matthew@yonedaai.com · https://yonedaai.com

23 July 2026

Provisional research specification: OBSTRUCTED

Abstract

Putting related records on the same machine can reduce communication, but "related" has several meanings in a semantic database. Two records may share a tenant, workspace, source, semantic object, provenance authority, region, time interval, or physical key. Those scopes can disagree. A low-latency replica may violate residency policy. A path whose endpoints are local may still depend on a remote edge. A shard advertised as irrelevant under one mapping may contain qualifying rows under another.

This paper gives a provisional placement and routing contract for CatDB. It separates logical partitions from physical placements, records exact or conservative coverage predicates, binds every route to query, mapping, snapshot, policy, consistency, and topology revisions, and carries explicit obligations into distributed join, exchange, index, and materialization alternatives. A partition may be pruned for logical disjointness only when a checkable certificate proves that its coverage and the rewritten query predicate cannot overlap under the same semantics. Policy exclusion and snapshot incompatibility remain distinct outcomes. Complete selection must also preserve ordinary bag multiplicity after replica and intentional-overlap handling.

The paper states twelve proposed invariants, supplies eighteen finite counterexamples, and fixes fifteen claim anchors. It compares the proposed contract with established work on distributed query processing, workload and graph partitioning, geographic and tenant placement, logical data skipping, provenance, and materialized computation. The possible CatDB contribution is narrow: a versioned contract that keeps semantic scopes and physical obligations together, followed eventually by controlled ablation against physical-only and communication-aware baselines.

This is not a sharding system or a distributed planner result. The repository contains no P12 routing runtime, exchange executor, distributed join planner, pruning oracle, migration protocol, cross-shard benchmark, vector-locality result, or full-text locality result. Papers 2, 5, 10, and 11 are internally accepted paper contracts, but the

repository supplies no executable interfaces or independent oracles that instantiate their query denotation, source coverage, topology/migration, and observation-consistency boundaries. CatDB-specific correctness, locality, performance, and scale claims therefore remain OPEN, OBSTRUCTED, or REJECTED.

Contents

1 Introduction

1.1 One request, several boundaries

Consider a tenant that has a published customer model, a private workspace with an uncommitted mapping, orders in two regional PostgreSQL shards, a rate-limited source of support cases, and a materialized revenue view. An analyst asks for customers whose June orders and support activity satisfy a typed path. The response must include complete lineage and obey a policy that keeps raw support text in one region.

Several attractive shortcuts are wrong. Routing by tenant alone can expose the private workspace. Routing by a semantic `Customer` label can overload one hot shard. Sending the whole query to the closest region can violate residency or use a connector with different scalar semantics. Reading the materialized view because its refresh timestamp is current can cross a mapping revision. Pruning a June shard because it was marked closed can miss a late correction. Replicating a small reference relation can multiply an ordinary bag result if the replicas are treated as distinct logical inputs.

These are not edge cases that a cost model can repair later. They determine which physical alternatives are admissible. Cost-based selection belongs after query meaning, coverage, policy, snapshot, replica, and migration obligations have been made explicit.

1.2 Research question

The bounded question is:

> Can a versioned semantic placement contract improve shard selection, cross-shard plan enumeration, and plan explanation over a physical-statistics baseline without changing query denotation, hiding incomplete coverage, or erasing network and consistency costs?

The strongest defensible target is a finite profile with a frozen relational fragment, exact result-bag comparison, explicit source coverage, conventional hash and range baselines, a fixed topology and consistency profile, and feature-by-feature ablation. No such implementation or evaluation exists today.

1.3 Contributions and boundaries

This provisional manuscript contributes:

1. a typed separation among partition coverage, physical placement, routing epochs, requests, decisions, and obligations;
2. a partition-dimension analysis for workspace, tenant, semantic object, source, provenance domain, geography, time, and physical key;
3. soundness and completeness conditions for correctness-preserving pruning under exact or conservative coverage;
4. admissibility contracts for paths, distributed joins, query and data shipping, reference replicas, indexes, and materializations;
5. twelve proposed invariants and eighteen finite counterexamples;
6. a primary-source synthesis, fifteen claim anchors, and a falsifiable future evaluation contract.

The paper does not claim a running router, a distributed executor, a benchmark win, native pruning superiority, cross-shard performance, vector locality, or full-text locality. It

does not call an alternative optimal. P12 may enumerate admissible alternatives; Paper 14 owns later cost-based choice.
Evidence status. PROPOSED manuscript contract; implementation and evaluation OBSTRUCTED..

2 Evidence Discipline and Dependency Boundary

2.1 Status vocabulary

Label	Meaning in this paper
SUPPORTED BY PRIOR LITERATURE	A primary source establishes the cited result under its own assumptions. It is not CatDB evidence.
PAPER REASONING	A finite argument or counterexample establishes the bounded statement in this manuscript. It does not verify a runtime.
ACCEPTED PAPER CONTRACT	An internally accepted predecessor manuscript fixes a paper-level boundary. It is not an executable interface, oracle, or runtime result.
PROPOSED	A typed contract is specified but not accepted or executed.
OPEN	A claim has a defined evidence path but no accepted result.
OBSTRUCTED	A required upstream definition, implementation, or evidence path is absent.
REJECTED	A claim conflicts with prior work, a counterexample, or the current repository cut.

Table 1: Evidence labels. No P12 claim is currently computational, experimentally supported, machine checked, or an implemented guarantee.

Evidence does not transfer automatically between layers. A distributed join paper does not implement CatDB. A typed path does not reveal a placement. A coverage predicate does not prove that routing metadata is current. A locally sound prune does not prove a complete multi-source answer. An immutable replicated target does not make a mutable active pointer consistent. A query equivalence proof does not verify exchange code.

2.2 Direct dependency blockers

P6, P9, and P13 also affect materialization, workspace isolation, and provenance completeness. Those secondary dependencies cannot replace the four direct boundaries. P12 must not treat accepted manuscript definitions as implemented behavior or invent router-local substitutes for their missing executable profiles.

2.3 Current repository cut

The current repository has no P12 partition descriptor, placement registry, routing table, exchange operator, distributed join planner, pruning certificate checker, topology-epoch

Paper	Required paper boundary	Status	Consequence for P12
P2	Query denotation, bag/null/scalar semantics, typed physical-plan boundary, and result equivalence.	ACCEPTED PAPER CONTRACT	No executable query oracle or physical-plan interface exercises routing and join equivalence.
P5	Source interpretation, coverage, capabilities, pushdown, source snapshots, and partial-failure semantics.	ACCEPTED PAPER CONTRACT	No executable coverage or capability interface exercises source pruning and query shipping.
P10	Operation envelope, topology epoch, membership, replica identity, migration, and replay.	ACCEPTED PAPER CONTRACT	No distribution runtime or history oracle tests route stability and shard movement.
P11	Snapshot, visibility, consistency, freshness, stale/tentative/error, and active-reference profiles.	ACCEPTED PAPER CONTRACT	No consistency runtime or history oracle tests multi-placement reads and mutable references.

Table 2: The four accepted predecessor contracts. Their executable instantiation and the P12 evidence that would consume it remain absent.

implementation, migration protocol, distributed materialization placement service, cross-shard fixture, Haskell oracle, Lean P12 theorem, or benchmark result. The source package accompanying this manuscript therefore reproduces a LaTeX specification and citation audit, not a system result.

3 Typed Partition and Placement Contract

3.1 Primitive identities

The contract uses stable, version-bearing identities. None implies locality.

Listing 1: Primitive P12 identities.

```
PartitionId           := stable logical-partition identity
ReplicaId             := stable physical-replica identity
PlacementId           := stable physical-placement record
RoutingEpochId        := immutable topology revision
TenantId              := authorization and billing domain
WorkspaceId           := immutable workspace or overlay identity
SemanticObjectId      := versioned canonical object or family
SourceId              := versioned physical-source identity
ProvenanceDomainId    := provenance authority or retention domain
RegionId              := geography or residency region
TimeDomainId          := valid, system, or source-frontier time
SnapshotId            := immutable source or catalog snapshot
ConsistencyProfileId  := immutable P11 profile identity
CapabilityProfileId   := immutable P5 capability identity
QueryProfileId        := immutable P2 query-semantics identity
```

```
MappingVersionId      := immutable mapping revision
PolicyVersionId       := immutable policy revision
MaterializationId     := immutable definition or instance
IndexId               := immutable index definition and build
Digest                := canonical-content digest
```

A `SemanticObjectId` names meaning under a revision. A `PartitionId` names logical coverage. A `PlacementId` states where that logical coverage is realized. A `ReplicaId` names one physical copy. Conflating these identities is enough to create routing, multiplicity, and migration errors.

3.2 Partition dimensions

Listing 2: Partition dimension sum type.

```
PartitionDimension :=
    Workspace
  | Tenant
  | SemanticObject
  | Source
  | ProvenanceDomain
  | Geography
  | Time
  | PhysicalKey
  | Composite(list<PartitionDimension>)
```

`PhysicalKey` remains explicit because semantic dimensions must be compared and composed with hash, range, list, graph-derived, and workload-derived placement. Semantic dimensions do not replace ordinary physical partition keys.

3.3 Coverage predicate

Listing 3: Versioned coverage predicate.

```
CoveragePredicate := {
  language: PredicateLanguageId,
  expression: CanonicalPredicateIR,
  query_profile: QueryProfileId,
  mapping_scope:
      BoundTo(MappingVersionId)
    | MappingIndependent(MappingIndependenceWitness),
  exactness: Exact | ConservativeSuperset | SampledHint,
  authority: AuthorityRecord,
  valid_from: LogicalTime?,
  valid_to: LogicalTime?,
  recorded_at: SystemTime,
  digest: Digest
}
```

Only Exact and ConservativeSuperset coverage may participate in correctness-preserving exclusion. A SampledHint may affect estimates, probe priority, or physical-plan ranking. It cannot remove a partition from a complete exact answer.

`mapping_scope` has no omitted or default case. `BoundTo(v)` may support pruning only when the request rewrite and partition descriptor bind the same mapping version v. `MappingIndependent(w)` requires a checkable witness that the predicate uses only source-native or raw fields whose denotation is independent of every mapping admitted by the request. A missing, unknown, or invalid mapping scope fails closed: the partition remains a candidate, or the request is rejected, but the partition is never excluded by logical disjointness.

3.4 Partition descriptor

Listing 4: Logical partition descriptor.

```
PartitionDescriptor := {
  partition_id: PartitionId,
  logical_relation: LogicalRelationRef,
  dimensions: nonempty list<PartitionDimension>,
  coverage: CoveragePredicate,
  partition_key: PhysicalPartitionKey?,
  logical_owner: AuthorityRecord,
  topology_epoch: RoutingEpochId,
  placements: nonempty list<PlacementId>,
  replica_policy: ReplicaPolicy,
  residency_policy: PolicyVersionId,
  consistency_profile: ConsistencyProfileId,
  source_snapshots: map<SourceId, SnapshotId>,
  mapping_versions: set<MappingVersionId>,
  workspace_scope: WorkspaceScope,
  tenant_scope: TenantScope,
  provenance_scope: ProvenanceScope,
  temporal_scope: TemporalScope,
  indexes: set<IndexId>,
  materializations: set<MaterializationId>,
  migration_state:
      Stable | Copying | DualReadable | Cutover | Retiring,
  digest: Digest
}
```

The descriptor separates logical coverage from placement. Two physical replicas of one partition do not create two logical copies. Two overlapping logical partitions can create duplicate logical coverage, so they require an explicit overlap and multiplicity rule.

A row update is a cross-shard migration whenever it changes the effective partition key, even if the topology epoch is otherwise unchanged. This includes updates to a region, semantic-object assignment, workspace scope, or other semantic dimension used by the key. The transition must bind one read-snapshot or cut, old and new coverage predicates, write ownership for each phase, and duplicate suppression keyed by stable row identity and mutation identity. During copying or dual readability, reads use the named migration protocol. The source coverage may exclude the row only after the destination coverage is sound for the cut and the protocol has prevented loss or double visibility. Without those witnesses, the update is deferred or rejected rather than treated as an in-place mutation.

3.5 Placement descriptor

Listing 5: Physical placement descriptor.

```
PlacementDescriptor := {
  placement_id: PlacementId,
  partition_id: PartitionId,
  replica_id: ReplicaId,
  source_id: SourceId,
  region: RegionId,
  endpoint: EndpointRef,
  capability_profile: CapabilityProfileId,
  observed_snapshot: SnapshotId,
  consistency_profile: ConsistencyProfileId,
  health: Healthy | Degraded | Unreachable | Unknown,
  lag: FreshnessObservation?,
  routing_epoch: RoutingEpochId,
  indexes: set<IndexId>,
  materializations: set<MaterializationId>,
  digest: Digest
}
```

Health and lag are observations with timestamps and authorities. They are not timeless facts, and they do not override policy or snapshot requirements.

3.6 Routing epoch

Listing 6: Versioned routing epoch.

```
RoutingEpoch := {
  epoch_id: RoutingEpochId,
  parent: RoutingEpochId?,
  partitions: map<PartitionId, PartitionDescriptor>,
  placements: map<PlacementId, PlacementDescriptor>,
  effective_frontier: OperationFrontier,
  transition: Stable | FromParent(MigrationProtocolRef),
  consistency_profile: ConsistencyProfileId,
  created_by: AuthorityRecord,
  digest: Digest
}
```

A request must not silently combine unrelated epochs. A migration read needs a P10/P11 rule for old and new coverage, duplicate suppression, visibility, and write ownership.

4 Partition Dimensions and Their Failure Modes

No one dimension is the semantic partitioning scheme. One deployment may use region, then tenant, then time ranges, with source-specific physical shards. Another may partition object families and replicate a small immutable reference version. The contract retains the dimensions because flattening them into one label hides conflicts among isolation, completeness, locality, and policy.

Table 3: Partition dimensions, possible uses, and correctness risks.

Dimension	Candidate selector	Possible use	Main correctness risk
Workspace	Base commit, overlay, branch, immutable commit	Isolate experimental or uncommitted state while sharing immutable base data	One workspace observes another overlay or a mixed base/overlay snapshot
Tenant	Tenant identity and policy scope	Isolation, quota accounting, and tenant-local work	Cross-tenant leakage through references, caches, pooled indexes, or shared materializations
Semantic object	Canonical object, relation, typed object family, or path neighborhood	Co-locate bounded typed traversals	Semantic adjacency does not predict frequency; a central object can become a hotspot, and reassignment is a row migration
Source	Source identity and declared coverage	Source-native pushdown and avoidance of irrelevant calls	Coverage may be partial or overlapping; failure can be mistaken for emptiness
Provenance domain	Authority, retention, trust, source, or evidence class	Local lineage retention and policy evaluation	Derivation crosses domains and an incomplete explanation is labeled complete
Geography	Region, jurisdiction, or residency rule	Policy-admissible placement, user locality, regional availability	Location, authority, residency, network distance, and replica freshness disagree; region-key changes are row migrations
Time	Valid time, system time, source frontier, or event range	Range pruning, archival tiers, incremental refresh	Late data, corrections, overlapping intervals, or the wrong temporal interpretation

Dimension	Candidate selector	Possible use	Main correctness risk
Physical key	Hash, range, list, graph cut, or workload group	Balance, parallelism, and conventional pruning	Skew, correlated hot keys, and workload drift; mutable keys require cross-shard row migration

4.1 Workspace and tenant

Workspace placement must distinguish immutable shared base state, workspace-local overlay state, workspace-visible materializations, mutable publication references, merge/rebase operations, and authority to read another workspace. A tenant may own several isolated workspaces. A workspace is not a tenant.

Tenant routing is security-sensitive. A tenant key can be useful, but it does not prove isolation. Every tenant-derived exclusion must use authenticated context and a versioned policy. A missing tenant constraint is a rejected request, not permission for a global scan.

4.2 Semantic object and source

A semantic-object partition may group a canonical entity with total functional attributes, a bounded family traversed together, a path neighborhood, or a schema module with declared crossing edges. Such a partition remains a placement hypothesis. Cardinality, skew, update rate, data size, workload frequency, and network topology remain physical inputs.

A source partition is interpreted through P5 coverage and capability contracts. It may be complete, partial, overlapping, stale, materialized, or temporarily unavailable. Source selection asks which sources may contribute to a canonical query. Shard selection asks which placements of their data should be read. The two decisions are related but not interchangeable.

4.3 Provenance, geography, and time

A provenance-domain partition can reduce unnecessary lineage scans or keep regulated evidence in approved locations. If a result crosses provenance domains, the result contract must distinguish complete lineage, domain-scoped lineage, accepted summaries, policy withholding, failure, and unknown coverage. Withheld evidence is not "no lineage."

Geographic routing separates legal residency, physical location, coordinator location, failure domain, network distance, and authority jurisdiction. A nearby replica may be stale or unauthorized. A distant replica may be the only admissible source.

Temporal routing separates valid time, system time, source frontier, materialization refresh time, and metadata observation time [22]. A valid-time predicate cannot automatically prune a system-time partition. Late arrivals and corrections can modify an older valid-time range.

4.4 Composite, hierarchical, and overlapping coverage

A composite selector may use

$$C_s(x) = C_{\text{tenant}}(x) \wedge C_{\text{region}}(x) \wedge C_{\text{time}}(x).$$

The conjunction does not define precedence, overlap, fallback, replication, or migration.

One candidate evaluation order is:

Listing 7: Proposed policy-first routing order.

```
request
  -> policy-admissible regions
  -> tenant and workspace scopes
```

```
  -> semantic and source coverage
  -> time and physical-key partitions
  -> healthy, version-compatible placements
  -> admissible distributed-plan alternatives
```

An implementation may reorder pure metadata checks if it preserves the same admissible set and diagnostics. The overlap class is explicit:

Listing 8: Logical overlap classes.

```
OverlapClass :=
    DisjointLogicalCoverage
  | SameLogicalPartitionReplicas
  | IntentionalLogicalOverlap(DeduplicationRule)
  | TransitionalMigrationOverlap(MigrationProtocolRef)
  | UnknownOverlap
```

`UnknownOverlap` cannot support a complete exact bag result without an accepted identity and multiplicity rule.

5 Route Requests, Decisions, and Obligations

5.1 Route request

Listing 9: Route request.

```
RouteRequest := {
  request_id: RequestId,
  query: TypedQueryRef,
  query_profile: QueryProfileId,
  workspace: WorkspaceId?,
  tenant: TenantId,
  semantic_scope: set<SemanticObjectId>,
  allowed_sources: set<SourceId>?,
  required_provenance: ProvenanceRequirement,
  geography: GeographyRequirement,
  time_window: TemporalPredicate?,
  source_snapshots: map<SourceId, SnapshotRequirement>,
  consistency: ConsistencyProfileId,
  freshness: FreshnessRequirement,
  residency_policy: PolicyVersionId,
  mapping_versions: set<MappingVersionId>,
  topology_epoch: RoutingEpochId,
  max_fanout: nonnegative integer?,
  execution_budget: ExecutionBudget?,
  failure_policy: Strict | ExplicitPartial | DeclaredStale,
  trace_level: TraceLevel
}
```

Tenant is required because a request without an authorization domain is not admissible. Workspace is optional because a request may target a published immutable commit.

5.2 Route decision

Listing 10: Typed routing outcome.

```
RouteDecision :=
  Admitted {
    selected_partitions: nonempty set<PartitionId>,
    selected_placements: nonempty set<PlacementId>,
    excluded_partitions:
        map<PartitionId, PruneCertificate>,
    alternatives:
        nonempty list<DistributedPlanAlternative>,
    obligations: set<RoutingObligation>,
    expected_completeness:
        Complete | ExplicitlyPartial | Unknown,
    topology_epoch: RoutingEpochId,
    decision_digest: Digest
  }
| Deferred {
    missing_metadata: set<MetadataRequirement>,
    retry_condition: RetryCondition
  }
| Rejected {
    code: RoutingDiagnosticCode,
    reasons: nonempty list<RoutingDiagnostic>,
    violated_requirements: set<RequirementRef>
  }
| Unsupported {
    code: RoutingDiagnosticCode,
    unsupported_features: nonempty set<FeatureRef>
  }
```

A `max_fanout` limit cannot justify an incomplete ordinary answer. If a complete result exceeds the limit, routing must reject, defer, or return an explicitly partial result under a profile that allows partiality.

5.3 Routing obligations

Listing 11: Obligations retained by admitted plans.

```
RoutingObligation :=
    CoverageSound(PartitionId, CoveragePredicate)
  | PruningSound(PartitionId, PruneCertificate)
  | CompleteSelection(set<PartitionId>, CompletenessWitness)
  | TenantIsolation(TenantId, PolicyVersionId)
  | WorkspaceIsolation(WorkspaceId, SnapshotId)
  | ResidencyAllowed(PlacementId, PolicyVersionId)
  | CapabilitySatisfied(
        PlacementId, CapabilityProfileId, PlanFragment)
  | SnapshotCompatible(
        set<PlacementId>, ConsistencyProfileId)
  | MappingCompatible(set<MappingVersionId>)
  | ReplicaMultiplicityPreserved(ReplicaEquivalenceWitness)
```

```
  | JoinSemanticsPreserved(JoinEquivalenceWitness)
  | PathCoveragePreserved(PathRoutingWitness)
  | MaterializationValid(
        MaterializationId, DependencyWitness)
  | RoutingEpochCoherent(RoutingEpochId)
  | MigrationReadSafe(MigrationWitness)
  | ProvenanceSufficient(ProvenanceRequirement)
```

These are proof or validation requests, not explanatory decoration. An executor that discards them changes the admitted-plan contract.

6 Sound Pruning and Complete Selection

6.1 Coverage soundness

Let I_s be the logical contents of partition s at the accepted snapshot. Let $C_s(x)$ be its versioned coverage predicate and $\phi_q(x)$ the query predicate after accepted P2/P5 rewriting. Coverage soundness requires

$$x \in I_s \implies C_s(x). \tag{1}$$

Logical-disjointness pruning is admissible only if

$$C_s(x) \wedge \phi_q(x) \quad \text{is unsatisfiable} \tag{2}$$

under the same scalar, null, mapping, time, and policy semantics used by the query.

Proposition 6.1 (Conservative coverage permits false positives, not false negatives)**.** *Assume Equation (1) and that the decision procedure for Equation (2) is sound under the frozen query semantics. If C_s is an exact predicate or a conservative superset, pruning s after a valid unsatisfiability result removes no qualifying occurrence from I_s.*

Proof. Suppose a qualifying $x \in I_s$ were removed. By coverage soundness, $C_s(x)$. Because x qualifies, $\phi_q(x)$. Their conjunction would therefore be satisfiable, contradicting the valid unsatisfiability result. The conservative superset may retain extra partitions, but it cannot justify removing an actual qualifying occurrence. □

Evidence status. PAPER REASONING under explicit assumptions; P2/P5 instantiation, decision procedure, metadata maintenance, and execution remain OBSTRUCTED..

6.2 Prune certificate

Listing 12: Prune certificate.

```
PruneCertificate := {
  partition_id: PartitionId,
  query_fragment: PredicateRef,
  coverage_predicate: CoveragePredicate,
  rewrite_trace: RewriteTrace,
  decision:
```

```
      Disjoint | PolicyExcluded | SnapshotIncompatible,
  decision_engine: DecisionEngineId,
  assumptions: set<AssumptionRef>,
  mapping_evidence:
      BoundMappings(nonempty set<MappingVersionId>)
    | MappingIndependent(MappingIndependenceWitness),
  policy_version: PolicyVersionId,
  topology_epoch: RoutingEpochId,
  checked_at: SystemTime,
  witness_or_core:
      UnsatWitness | PolicyDecision | CompatibilityFailure,
  digest: Digest
}
```

`PolicyExcluded` and `SnapshotIncompatible` do not establish that a partition contains no matching data. They establish that a complete answer is not currently admissible from the placement. The request may need a typed rejection or an explicitly partial result.

6.3 Safe and unsafe inputs

Candidate correctness inputs include authenticated tenant and workspace scope, exact canonical-object coverage, mapping and source-coverage predicates, enumerated domains, exact equalities, temporal ranges, provenance requirements, residency policy, materialization dependency scope, exact path reachability summaries, and exact min/max or set summaries.

Embeddings, sampled histograms, stale statistics, learned source-selection scores, observed affinity, semantic names, unverified mappings, best-effort capabilities, health guesses, and natural-language policy labels are cost inputs or hints. They do not correctness-prune by themselves. In particular, this manuscript reports no vector or full-text pruning or locality result.

6.4 Complete selection

For a request requiring a complete exact result:

$$\mathsf{Ans}(q, I) = \mathsf{Ans}\left(q, \biguplus_{s \in S_{\text{selected}}} I_s\right), \tag{3}$$

after applying the declared replica and overlap quotient without changing logical bag semantics.

Equation (3) is a target, not a current theorem. It depends on P2 query denotation and P5 source completeness. A correct per-shard prune can coexist with an incomplete global answer if a contributing source or logical partition never entered the candidate cover.

7 Cross-Shard Paths

Let a typed path be

$$p = f_1; f_2; \cdots; f_n.$$

Typing establishes composability. It does not establish placement, index availability, replication, snapshot compatibility, source coverage, policy admissibility, or remote-call count.

Listing 13: Path placement record.

```
PathPlacement := {
  path_ref: TypedPathRef,
  edge_realizations: nonempty list<EdgePlacement>,
  endpoint_partitions: set<PartitionId>,
  required_indexes: set<IndexId>,
  source_fragments: list<SourcePlanFragment>,
  exchange_points: list<ExchangePoint>,
  snapshot_obligations: set<SnapshotObligation>,
  provenance_obligations: set<RoutingObligation>,
  topology_epoch: RoutingEpochId
}
```

P12 may enumerate local indexed navigation, a local join chain, source-native query shipping, batched remote lookup, edge or endpoint replication, data shipping, repartition on a path key, and reuse of a valid path materialization. Each alternative must preserve definedness, missing-value behavior, multiplicity, endpoint identity, mapping and source revisions, snapshot visibility, policy, and required provenance. Co-locating endpoints is insufficient when an intermediate edge or authority decision remains remote.

8 Distributed Joins and Exchanges

Distributed query processing, semijoins, relation shipping, parallel exchange, and split execution are established work [3, 4, 14, 15, 18, 26, 28]. P12's role is to enumerate admissible alternatives without changing the P2/P5 logical join.

Table 4: Distributed join alternatives.

Alternative	Shape	Preconditions	Main costs and risks
Co-located local	Both inputs already partitioned compatibly	Exact compatible keys, equality, mapping, and snapshot	Skew and hidden collation or mapping mismatch
Source-native pushdown	Connector executes the join	Exact P5 capability and source interpretation	Source load, dialect, failure, and returned bytes
Coordinator	Ship one or both inputs to a coordinator	Residency permits movement and coordinator preserves semantics	Bytes, memory, spill, and single-site bottleneck
Broadcast	Copy a small side to all partitions	Size bound, pinned replica version, bag and identity rule	Fanout, staleness, and write amplification
Repartition	Hash or range both sides by a join key	Compatible equality, hash/range, null, and scalar semantics	Network, spill, skew, and partition count
Semijoin	Send projected keys, filter remotely, return matches	Exact key projection and equality	Extra rounds, duplicate keys, and large key sets
Bloom-assisted semijoin	Send probabilistic membership filter	False positives allowed, false negatives forbidden	Build and transfer cost, saturation, residual traffic
Remote lookup	Issue batched key probes	Remote index, bounded outer cardinality, capability, and rate limits	Calls, throttling, and tail latency
Materialized join	Read a precomputed result	Dependency, frontier, snapshot, policy, and version validity	Freshness, storage, invalidation, refresh
Replicated-reference join	Join local facts with a pinned small reference replica	Exact version, digest, bag, equality, and snapshot	Active-pointer staleness and replica maintenance

Listing 14: Distributed join alternative.

```
DistributedJoinAlternative := {
  join_ref: LogicalJoinRef,
  strategy: JoinStrategy,
  input_partitions: map<InputSide, set<PartitionId>>,
  execution_sites: set<PlacementId>,
  exchanges: list<Exchange>,
  required_capabilities: set<CapabilityRequirement>,
  required_indexes: set<IndexId>,
  snapshot_contract: SnapshotContract,
  equality_profile: EqualityProfileId,
  multiplicity_profile: MultiplicityProfileId,
  null_profile: NullProfileId,
  residency_check: PolicyDecision,
  provenance_plan: ProvenancePlan,
  obligations: set<RoutingObligation>,
  cost_inputs: CostInputBundle
}
```

A cheap join is inadmissible if it changes equality or null semantics, reads incompatible snapshots, duplicates replicated rows, drops a contributing source, crosses a residency boundary, cannot produce required provenance, or uses a stale mapping or materialization. Only admissible alternatives reach the P14 cost model.

8.1 Query shipping, data shipping, and split execution

Query shipping sends executable work to the data. Data shipping moves data or intermediates to another execution site. Split execution combines remote fragments, local residuals, exchanges, and materialized stages. They are plan properties, not backend-wide labels.

Listing 15: Exchange contract.

```
Exchange := {
  exchange_id: ExchangeId,
  mode:
      QueryShipping | DataShipping |
      KeyShipping | FilterShipping,
  from: PlacementId,
  to: PlacementId,
  schema: PhysicalSchemaRef,
  partitioning: ExchangePartitioning,
  ordering: OrderingProperty?,
  estimated_rows: Estimate?,
  estimated_bytes: Estimate?,
  compression: CompressionProfile?,
  encryption: EncryptionProfile?,
  retry_profile: RetryProfile,
  snapshot_token: SnapshotToken?,
  provenance_mode: ProvenanceTransferMode,
  residency_decision: PolicyDecision,
  materialization:
      Temporary | Durable(MaterializationId) | Streaming,
  cost_inputs: CostInputBundle
```

```
}
```

The planner first establishes whether code and data may cross a proposed boundary. It can then compare CPU, I/O, messages, bytes, serialization, remote calls, throttling, spill, coordination, provenance capture, and failure exposure. Semantic metadata may change admissibility or improve an estimate. It cannot erase the transfer.

9 Reference Replicas, Indexes, and Materializations

9.1 Reference data

Reference data includes schemas, mappings, code sets, taxonomies, policies, dimension tables, and small entity dictionaries. An immutable, content-addressed version can support a local join only if the request pins that version, the digest matches, policy permits the placement, equality and bag semantics match, all dependencies exist, and result provenance records the version.

A mutable active pointer such as `current_mapping`, `main`, or `approved_policy` requires a P10/P11 observation contract. A route must name an exact active reference at an accepted linearization point, a causally compatible reference, a bounded-stale reference with observed lag, an explicit conflict or tentative state, or unavailability. It may not label all of those states "current."

Replicating one logical reference row to n placements must not multiply a logical join result by n. A plan chooses one compatible replica per logical partition or retains a replica-equivalence witness.

9.2 Index placement

Listing 16: Local index placement.

```
IndexPlacement := {
  index_id: IndexId,
  logical_relation: LogicalRelationRef,
  covered_partition: PartitionId,
  placement: PlacementId,
  key_semantics: KeySemanticsId,
  predicate: PredicateRef?,
  ordering: OrderingProperty?,
  snapshot: SnapshotId,
  build_frontier: OperationFrontier,
  mapping_versions: set<MappingVersionId>,
  topology_epoch: RoutingEpochId,
  health: IndexHealth,
  digest: Digest
}
```

An index definition and an index build are distinct identities. Candidate patterns include tenant-leading composites, workspace-overlay indexes, typed path edge indexes, source-native indexes, provenance-domain indexes, spatial indexes within region partitions,

temporal indexes, RDF permutations, and materialization-specific exchange keys. RDF-3X and Hexastore already demonstrate rich RDF index permutations [30, 36]; the existence of such indexes is not a P12 novelty claim.

A global secondary index is itself distributed, replicated, and consistency-sensitive. It may reduce fanout while adding write amplification, index lag, migration, rebuild, residency, and a new snapshot relation to base data. No locality advantage is reported here. Semantic adjacency and path indexes remain hypotheses to be measured against equivalent physical indexes.

Vector and full-text locality require separate physical baselines. SPANN uses balanced vector partitions, replicated closure assignments, and query-aware machine selection for billion-scale approximate nearest-neighbor search [9]. Earlybird hash-partitions documents across servers and maintains concurrently readable, real-time inverted indexes [7]. These systems establish baseline mechanisms and metrics; they do not establish exact semantic pruning or a CatDB locality result.

9.3 Distributed materialization

Listing 17: Distributed materialization descriptor.

```
DistributedMaterialization := {
  materialization_id: MaterializationId,
  definition: TypedQueryRef,
  output_schema: LogicalRelationRef,
  partitioning: PartitioningContract,
  placements: nonempty set<PlacementId>,
  dependency_versions: set<ArtifactVersionRef>,
  source_frontiers: map<SourceId, OperationFrontier>,
  topology_epoch: RoutingEpochId,
  consistency_profile: ConsistencyProfileId,
  provenance_profile: ProvenanceRequirement,
  residency_policy: PolicyVersionId,
  refresh_mode: Full | Incremental | Lazy | OnDemand,
  visibility: PrivateWorkspace | Tenant | Published,
  freshness: FreshnessObservation,
  indexes: set<IndexId>,
  state: Building | Ready | Stale | Invalid | Failed | Retiring,
  digest: Digest
}
```

P12 may enumerate shard-local partial views, partition-aligned joins, region-local replicas, a coordinator aggregate, replicated reference views, source-local pushed views, workspace-private or tenant-shared views, and partitioned dataflow state. Reuse is admissible only when definition coverage, schema and mapping versions, source frontiers, visibility, policy, residency, provenance, partition overlap, ready state, and topology epoch all match.

Partial aggregates require an admitted merge algebra. `SUM` and `COUNT` under fixed bag and numeric semantics differ from `AVG`, distinct aggregates, order-sensitive aggregates, top-k, and nonassociative functions. There is no generic "materialize locally and merge" rule.

Incremental view maintenance and distributed dataflow are established areas [5, 6, 17, 27]. Refresh and movement also cross P6, P10, and P11 boundaries. Copy completion alone does

not prove freshness or safe cutover.

10 Proposed Correctness Invariants

The following IDs are local to P12. They are acceptance obligations, not current system guarantees.

Table 5: Twelve proposed P12 invariants.

ID	Proposed invariant	Deps	Status
SHARD-01	Every logical partition descriptor over-approximates its actual contents at the named snapshot.	P2, P5	OBSTRUCTED
SHARD-02	Every excluded partition has a valid disjointness, policy, or compatibility certificate; only disjointness proves no matching tuple.	P2, P5	OBSTRUCTED
SHARD-03	Selected partitions cover every qualifying logical occurrence for a complete request.	P2, P5	OBSTRUCTED
SHARD-04	Replica selection and intentional overlap preserve logical bag multiplicity.	P2, P10, P11	OBSTRUCTED
SHARD-05	Every admitted local, broadcast, repartition, semijoin, lookup, or materialized join equals the frozen logical join.	P2, P5	OBSTRUCTED
SHARD-06	Cross-shard path routing preserves endpoint results, definedness, multiplicity, versions, and provenance.	P2, P5	OBSTRUCTED
SHARD-07	A route uses one coherent topology epoch or an accepted migration-read protocol, including row-level effective shard-key changes.	P10, P11	OBSTRUCTED
SHARD-08	Every mutable-reference read satisfies the named P11 profile or returns a typed stale, conflict, or unavailable result.	P10, P11	OBSTRUCTED
SHARD-09	Materialization reuse satisfies dependency, frontier, policy, workspace, tenant, and topology conditions.	P6, P10, P11	OBSTRUCTED
SHARD-10	Residency and tenant/workspace isolation are checked before cost-based route selection.	P5, P9, P11	OPEN design; runtime OBSTRUCTED

ID	Proposed invariant	Deps	Status
SHARD-11	Routing explanations are deterministic for the same versioned metadata and request.	P2, P5, P10, P11	OPEN
SHARD-12	Semantic-feature benefit is reported only when a named ablation improves a preregistered endpoint after overhead.	Bench. harness	OPEN

11 Counterexamples

Each counterexample is a required future fixture family. The consequence is PAPER REASONING for the bounded history. No CatDB runtime is proved.

Counterexample 11.1 (CE-01: semantic object creates a hotspot). Partition all customer facts and edges by semantic object. Ninety percent of queries traverse `Customer`, and one enterprise customer owns forty percent of orders. The grouping is well typed but overloads one partition. A hash or workload-derived subpartition can do better.

Consequence. Semantic coherence does not imply balance.

Counterexample 11.2 (CE-02: mapping rewrite invalidates pruning). Partition S_1 advertises `region = US` under mapping m_1. Mapping m_2 interprets source code `NA` as both US and Canada under a reconciliation rule. A Canadian query compiled with m_2 is evaluated against old m_1 coverage and S_1 is pruned.

Consequence. Coverage and query predicates must share mapping semantics.

Counterexample 11.3 (CE-03: tenant and workspace are conflated). Tenant T owns workspaces W_1 and W_2. W_1 contains an uncommitted mapping and overlay row. Routing by tenant alone exposes the row to a query in W_2.

Consequence. Tenant routing does not establish workspace isolation.

Counterexample 11.4 (CE-04: mutable reference replicas diverge). Two regions cache `active_mapping`. During a partition, one points to m_7 and one to m_8. A global query joins results compiled under both.

Consequence. Mutable reference replication requires a named observation contract.

Counterexample 11.5 (CE-05: local endpoints hide a remote path edge). `Order` and `Country` endpoints are local, but the `Order -> Customer -> Address -> Country` realization stores `Address` in another region.

Consequence. Endpoint co-location does not prove path locality.

Counterexample 11.6 (CE-06: local uniqueness is not global uniqueness). Two tenant shards each accept customer key 42 because uniqueness is enforced locally. A global identity query assumes one row.

Consequence. Local constraints do not imply global constraints.

Counterexample 11.7 (CE-07: late data defeats a closed time partition). A June valid-time partition is marked closed on 1 July. On 5 July a source corrects a June event. A router that trusts only the closed marker excludes June from a history query.

Consequence. Valid time, system time, and source frontier remain distinct.

Counterexample 11.8 (CE-08: geography and residency disagree). The lowest-latency replica is in region R_2, but policy p_9 permits the tenant's raw records only in R_1. The planner ships data to R_2.

Consequence. Cost does not override residency.

Counterexample 11.9 (CE-09: shipping cost reverses with selectivity). A coordinator expects a remote predicate to select 0.1% of a table and ships the query. The source cannot push the predicate exactly and returns the whole table. Key shipping would have transferred less.

Consequence. Capability and residual work are explicit cost inputs.

Counterexample 11.10 (CE-10: broadcast changes bag multiplicity). A dimension row exists in three replicas. The coordinator scans all three as logical partitions and joins them with one fact row, returning three copies.

Consequence. Replica identity must be quotiented without removing ordinary logical duplicates.

Counterexample 11.11 (CE-11: stale materialization crosses mappings). A revenue view is at source frontier 100 under `gross-revenue@4`. A query asks for frontier 100 under `net-revenue@5`. The router sees only the matching frontier and reuses the view.

Consequence. Freshness without semantic dependency versions is insufficient.

Counterexample 11.12 (CE-12: routing-epoch cutover loses a read). Range $[0, 100)$ moves from shard A to shard B. The routing table switches to B before the last copied delta is visible there. A post-cutover read misses a record.

Consequence. Cutover needs an accepted P10/P11 migration protocol.

Counterexample 11.13 (CE-13: provenance partition hides a derivation). A result is stored in the `finance` provenance domain, but an input mapping and reconciliation decision live in `identity`. The router reads only `finance` and labels the lineage complete.

Consequence. Provenance locality cannot redefine completeness.

Counterexample 11.14 (CE-14: graph cut fails under workload drift). A graph partition minimizes crossing edges for workload W_0. A new query family traverses a previously cold, high-degree relation, and nearly every query crosses shards.

Consequence. Graph partition quality is workload-relative.

Counterexample 11.15 (CE-15: local top-k is incomplete). Each of ten shards returns its local top ten. The global score uses a cross-shard authority weight that local shards do not have.

Consequence. Local top-k merge needs explicit score and bound assumptions.

Counterexample 11.16 (CE-16: Bloom filter has a false negative). An incorrectly resized Bloom filter drops a present join key and the remote side removes its matching row.

Consequence. Bloom-assisted semijoins may admit false positives, not false negatives.

Counterexample 11.17 (CE-17: global index is ahead of base data). A global index routes key k to shard B after a migration record is published, but the base row is visible only on A at the requested snapshot.

Consequence. An index route needs a snapshot relation to base data.

Counterexample 11.18 (CE-18: policy change leaves a reusable cache). A tenant materialization was built under policy p_3, which allowed a sensitive field. Policy p_4 removes that authority. The physically fresh view remains in cache and is served.

Consequence. Policy revision participates in materialization validity.

12 Prior Art and the Narrow Candidate Delta

12.1 Distributed planning and physical execution

Cost-based access-path and join-order selection predates P12 [31]. SDD-1 and semijoin work made communication part of distributed query processing [3, 4]. R* compared estimated and actual messages, I/O, and CPU across relation shipping, fetch-matches, semijoins, and Bloom joins [26]. Shared-nothing execution, exchange operators, split execution, and modern exchange placement are also established [14, 15, 18, 28].

The consequence is direct: P12 cannot claim novelty for local, broadcast, repartition, lookup, coordinator, semijoin, Bloom-assisted, query-shipping, or data-shipping alternatives. It must preserve and compare them.

12.2 Partitioning, elasticity, tenancy, and geography

Workload-driven placement, single-partition transaction design, deterministic ordering, and elastic hot/cold movement are established [12, 23, 34, 35]. Consistent hashing, high-availability partitioning, global replication, range tablets, tenant-guided distribution, and multi-tenant schema techniques likewise precede P12 [1, 8, 11, 13, 24, 32].

These systems show that keys, workload, geography, tenancy, replication, skew, and transaction boundaries already guide placement. Any P12 result must show what versioned semantic metadata adds after those inputs receive an equal baseline.

12.3 Graph placement, logical skipping, and completeness

RDF-3X, Hexastore, distributed RDF engines, graph partitioning, and graph-aware query decomposition are close prior art for semantic-object and path claims [20, 21, 25, 30, 36]. Graph cuts can reduce communication for selected workloads, but overlap, replication, skew, and drift remain.

Pando is a direct logical-partitioning and data-skipping baseline, while completeness metadata constrains what source exclusion can mean [29, 33]. R*-tree work and temporal data management provide established geographic, multidimensional, and time boundaries [2, 22].

12.4 Provenance, materialization, and federation

Provenance semirings and database-provenance surveys establish the compositional and interpretive boundaries for lineage [10, 19]. Classic view maintenance, Differential Dataflow, Noria, and DBSP establish materialized and incremental state [5, 6, 17, 27]. BigDAWG illustrates cross-engine islands and movement [16].

P12 therefore cannot claim novelty for provenance, materialization, incremental maintenance, distributed dataflow, or polystore movement.

12.5 Candidate delta

The plausible CatDB contribution is a typed, versioned placement and routing contract whose coverage predicates can name workspace, tenant, semantic object, source, provenance domain, geography, time, and physical key; whose logical exclusions carry checkable certificates; and whose alternatives retain mapping, snapshot, authority, residency, provenance, consistency, replica, routing-epoch, and materialization obligations. Its added value would have to survive feature ablation against conventional physical and communication-aware baselines.

That contribution is PROPOSED. It becomes a systems result only after correctness checks, implementation, controlled evaluation, and independent review.

13 Claim Ledger

The fifteen claim anchors are deliberately uneven. `P12-C14` and `P12-C15` are rejected, so their inclusion prevents later prose from quietly reviving them.

Table 6: P12 claim anchors and promotion gates.

ID	Candidate claim	Current status	Required evidence
P12-C01	P12 can represent the seven requested semantic dimensions plus physical keys in one typed descriptor.	OPEN design; implementation OBSTRUCTED	Accepted schema, fixtures, and cross-language round trip
P12-C02	Versioned coverage predicates can support sound shard pruning.	OBSTRUCTED	Executable P2 query oracle, P5 coverage evaluator, and SHARD-01/02
P12-C03	Selected shards can be proved complete for the bounded profile.	OBSTRUCTED	Executable P2/P5 denotation and completeness oracle plus SHARD-03
P12-C04	Cross-shard paths can be decomposed without changing results.	OBSTRUCTED	Executable P2 path oracle, P5 realizations, and SHARD-06
P12-C05	P12 can enumerate the named distributed join alternatives with explicit obligations.	OPEN design; execution OBSTRUCTED	Typed plan schema and correctness fixtures
P12-C06	Query and data shipping can respect capability, residency, and provenance policy.	OBSTRUCTED	Executable P5/P11 profiles plus policy fixtures
P12-C07	Version-pinned immutable reference replicas can support exact local joins.	OPEN	Digest, placement, equality, bag, and snapshot tests
P12-C08	Mutable active references can be read without silent consistency downgrade.	OBSTRUCTED	Executable P10/P11 profiles and histories
P12-C09	Semantic-object, path, vector, or full-text indexes improve locality.	OPEN	Matched physical indexes, SPANN/Earlybird-style baselines, retrieval-specific workloads, ablation, skew, and drift
P12-C10	Distributed materialization placement preserves validity.	OBSTRUCTED	P6/P10/P11 frontiers, dependencies, recovery, and SHARD-09
P12-C11	Routing-epoch migration preserves reads and writes under a named profile.	OBSTRUCTED	P10/P11 migration model and fault histories
P12-C12	Semantic metadata improves a preregistered routing or runtime endpoint.	OPEN	Baseline-controlled ablations including metadata overhead

ID	Candidate claim	Current status	Required evidence
P12-C13	Routing explanations are deterministic and auditable.	OPEN	Canonical metadata, traces, and insertion-order tests
P12-C14	Semantic metadata eliminates network and consistency costs.	REJECTED	Conflicts with scope, prior work, and counterexamples
P12-C15	P12 currently has an executable implementation or benchmark.	REJECTED	Current repository audit

14 Evaluation Contract, Not Results

14.1 Safety gate

No timing result may be interpreted until:

1. query and result semantics are frozen;
2. source coverage and failure semantics are accepted;
3. every treatment returns the required result bag and provenance;
4. bounded exhaustive and generated pruning fixtures observe no false negatives;
5. tenant, workspace, residency, snapshot, and materialization gates pass;
6. migration histories pass for the named epoch protocol.

This manuscript executes none of those system checks and reports no benchmark.

14.2 Baselines and treatments

Baseline	Comparison purpose
Hash and range/time partitioning	Balance, fanout, conventional pruning, and repartition joins
Physical statistics and capabilities only	Isolate the effect of semantic metadata
R*-style communication alternatives	Messages, I/O, CPU, shipping, fetch-matches, semijoin, and Bloom
Schism and E-Store style placement	Workload graph, balance, skew, movement, and drift
Pando-style logical skipping	Metadata-rich logical predicate pruning
Distributed RDF and graph partitioning	Semantic-object and path workloads
SPANN-style distributed vector ANNS	Recall at k, machine fanout, partition balance, closure replication, memory/disk cost, and index build
Earlybird-style sharded inverted index	Term and posting placement, query fanout, ingestion/indexing latency, concurrent updates, and freshness
PolyBase and exchange placement	Pushdown, split execution, and movement
Full recomputation plus IVM/dataflow	Distributed materialization

Table 7: Minimum baseline families. A reconstruction does not inherit the original system's published performance.

Treatments add one feature at a time: tenant, workspace, semantic object/path, source coverage, provenance domain, geography/residency, time/frontier, and semantic material-

ization metadata. The full treatment comes last. Feature order must be randomized or preregistered when interactions matter.

Vector workloads must pin dimension, distance, target recall, dataset size, partition count, closure replication, update rate, and index-build resources. Full-text workloads must pin corpus, tokenizer, term and document skew, partitioning, posting replication, ingestion rate, freshness, top-k semantics, and query fanout. Neither workload may reuse relational completeness language for an approximate retrieval result.

14.3 Workloads and metrics

The workload matrix varies partition and replica count, regions, tenants, workspace depth and overlay fraction, semantic-object skew, source type and overlap, provenance completeness, temporal width and late arrivals, join selectivity and skew, path length, network behavior, snapshot compatibility, routing-epoch phase, materialization validity, index state, and strict, partial, or stale failure policy.

Correctness metrics include result-bag and provenance equality, pruning false negatives and false positives, policy leaks, fractured reads, migration loss/duplication, and invalid materialization reuse. Planning metrics include alternatives, metadata bytes, prediction error, certificate size, and maintenance cost. Execution metrics include first-row and total latency, throughput, bytes, messages, calls, CPU, I/O, memory, spill, retries, rejections, and movement.

14.4 Falsifiers

Table 8: Exact future falsifiers.

Hypothesis	Falsifier
Pruning is sound	Any qualifying occurrence on a disjointness-pruned partition
Selection is complete	Any required occurrence absent from the selected logical union
Replica handling preserves bags	Any multiplicity change caused by physical replica count
Path routing preserves semantics	Any endpoint, missingness, multiplicity, version, or provenance mismatch
A join alternative is exact	Any normalized result or provenance mismatch
Epoch migration is safe	Any lost, duplicated, fractured, or unauthorized observation outside the accepted profile
Materialization reuse is valid	Reuse after an invalidating dependency, policy, frontier, workspace, or topology change
Semantic metadata improves pruning	No benefit over physical metadata after overhead, or the benefit vanishes under ablation
Semantic metadata improves planning	No preregistered reduction in cost error or admissible execution cost

Hypothesis	Falsifier
Semantic placement improves locality	No reduction in crossings, bytes, or latency, or reversal under declared drift

Future statistical reporting must pin source, compiler, connector, database, topology, and protocol versions; preserve failures and rejected plans; use repeated randomized schedules; separate cold and warm states; report per-workload results; retain seeds and minimized counterexamples; count metadata maintenance; and never substitute extrapolation for an unexecuted scale.

15 Formalization and Implementation Roadmap

15.1 Finite oracle and proof boundary

A future independent finite oracle should enumerate typed partition covers, evaluate frozen queries before and after partitioning, check disjointness pruning, quotient replicas while preserving bags, compare join alternatives, evaluate paths, check admitted aggregate algebras, and minimize false-negative or mixed-epoch histories.

Proposed Lean targets are:

Listing 18: Proposed finite Lean target names.

```
Shard.coverage_sound
Shard.prune_sound
Shard.selection_complete
Shard.replica_quotient
Shard.local_join_equiv
Shard.semijoin_equiv
Shard.path_route_equiv
Shard.partial_aggregate_merge
Shard.epoch_coherence
```

Each theorem must state bag, null, scalar, mapping, completeness, and snapshot assumptions.

Finite tests would not prove all sizes. Lean query equivalence would not verify a connector or router. Semijoin equivalence would not make a semijoin cheaper. Coverage proof would not establish metadata currency. Epoch coherence would not establish liveness.

15.2 Staged implementation

1. **R0, executable dependency profiles:** bind the accepted P2 query, P5 source, P10 topology/migration, and P11 observation contracts to normalized executable interfaces and independent oracles. Status: OBSTRUCTED.
2. **R1, language-neutral schemas:** define canonical envelopes for descriptors, epochs, requests, decisions, certificates, exchanges, joins, indexes, materializations, and counterexamples. Status: OPEN after R0.

3. **R2, independent finite oracle:** implement the bounded model without importing Rust decisions. Status: OBSTRUCTED.
4. **R3, Lean core:** prove only the frozen `SHARD-*` subset and publish axiom and coverage tables. Status: OBSTRUCTED.
5. **R4, metadata and routing layer:** implement descriptor, coverage, pruning, topology, route, explanation, and diagnostic modules. Status: OBSTRUCTED; no crate exists.
6. **R5, physical alternatives:** add typed local, broadcast, repartition, semijoin/Bloom, lookup, shipping, and materialization alternatives. Status: OBSTRUCTED.
7. **R6, topology and fault simulator:** exercise skew, lag, source failure, epoch transition, late updates, and policy/mapping change. Status: OBSTRUCTED.
8. **R7, backend execution:** integrate only P5-conformant connectors and compare every distributed result with the independent oracle. Status: OBSTRUCTED.
9. **R8, benchmark harness:** implement the baseline, ablation, measurement, and falsification contract. Status: OBSTRUCTED.

16 Exact Nonclaims and Limitations

For avoidance of doubt:

1. CatDB does not currently implement semantic sharding.
2. CatDB does not currently implement distributed query planning.
3. No P12 route, join, pruning, materialization, or benchmark result has been measured.
4. No semantic metadata feature has shown a performance benefit.
5. No P12 pruning rule has been machine checked.
6. A semantic object is not a physical partition.
7. A typed path is not a placement map.
8. A source partition is not necessarily complete or disjoint.
9. A tenant key does not by itself prove isolation.
10. A workspace is not a tenant.
11. A provenance domain is not necessarily a complete derivation boundary.
12. Geography, residency, authority, and network distance are distinct.
13. Valid time, system time, source frontier, and refresh time are distinct.
14. Replication does not imply freshness or authority.
15. An active reference is not immutable because its targets are immutable.

16. A global index is not free of replication and consistency cost.
17. A materialization is not valid merely because its rows are present.
18. Graph partitioning does not eliminate edge cuts.
19. A semijoin or Bloom filter is not always cheaper than relation shipping.
20. A distributed join is not correct because it is cheap.
21. Semantic metadata does not eliminate network transfers.
22. Semantic metadata does not eliminate consistency or coordination cost.
23. Category theory does not change CAP, consensus, failure, or latency boundaries.
24. P12 enumerates admissible alternatives; P14 owns cost-based choice.
25. External system results do not establish CatDB behavior.
26. This paper reports no vector-search locality result.
27. This paper reports no full-text locality result.
28. A row-level update is not local when it changes the effective shard key.
29. An absent or unknown mapping binding does not establish mapping-independent coverage.

16.1 Open and obstructed questions

Blocked questions include the exact P2 query denotation, P5 source-coverage language and complete-answer rule, P10 epoch and migration operations, P11 snapshot cut and active-reference profiles, P6 materialization frontier, P9 workspace isolation, and P13 provenance completeness.

Open design questions include a practical predicate language for certificates, maintenance of conservative coverage under mapping change, composite hierarchies, high-degree object handling, exact path summaries, semijoin crossover points, zero-false-negative Bloom implementations, index lag, admitted partial aggregates, explicit incomplete results, metadata overhead, workload drift, and useful operator explanations.

17 Conclusion

Semantic metadata can inform distributed placement, but it does not turn meaning into locality. The same object can span sources, regions, workspaces, policies, mappings, and snapshots. The contract in this paper keeps those scopes separate, requires checkable exclusion certificates, and retains physical and consistency obligations in every candidate plan.

The current result is a falsifiable specification. It says what an eventual router must prove, reject, expose, and measure. It does not show that CatDB routes a query, prunes a shard, executes a distributed join, preserves a migration read, or improves locality. Those

claims remain blocked until the accepted P2, P5, P10, and P11 boundaries are instantiated as executable profiles and a bounded implementation passes correctness gates before performance evaluation.

A Disputed Claims

Table 9: Claims rejected by the specification.

ID	Disputed claim	Reason for rejection
P12-D01	A typed path is physically local.	Typing has no placement semantics.
P12-D02	Semantic-object partitions are naturally balanced.	CE-01 and workload/skew prior art.
P12-D03	Semantic metadata eliminates cross-shard joins.	Semantic relations commonly cross partitions.
P12-D04	A replicated reference is current everywhere.	Mutable references require P10/P11 guarantees.
P12-D05	A freshness timestamp proves materialization validity.	Dependencies, mappings, policies, and frontiers also matter.
P12-D06	A Bloom join can safely lose positives.	False negatives change the result.
P12-D07	Tenant partitioning proves isolation.	Caches, indexes, references, and policy checks may cross tenants.
P12-D08	Region-local execution is policy-admissible.	Location, authority, and residency are distinct.
P12-D09	Global indexes remove fanout without consistency cost.	The index is a distributed replicated structure.
P12-D10	Local uniqueness implies global uniqueness.	CE-06.
P12-D11	Closed time partitions never change.	Late arrivals and corrections.
P12-D12	Graph partitioning removes network communication.	Cuts, overlap, skew, and workload drift remain.
P12-D13	Category theory changes CAP or consensus boundaries.	No message or ordering mechanism follows from functoriality.

ID	Disputed claim	Reason for rejection
P12-D14	The route with minimum estimated bytes is admissible.	Semantics, snapshots, residency, and provenance precede cost.
P12-D15	The current repository implements P12.	The local audit found no P12 runtime surface.
P12-D16	A row-level shard-key update is an in-place local mutation.	Changing the effective partition key is a cross-shard migration.
P12-D17	An omitted mapping version means coverage is mapping-independent.	Mapping independence requires a checkable witness; missing or invalid evidence fails closed.

B Review and Reproduction Gate

A future accepted manuscript must receive distributed-query, database-systems, graph/RDF, consistency, security/residency, implementation, reproducibility, and adversarial claim review. Reviewers should ask whether:

1. query semantics and source coverage are explicit;
2. every excluded partition has the correct certificate class;
3. the eight partition dimensions remain distinct;
4. replicas, overlap, and bag duplicates remain distinct;
5. all movement and snapshot obligations are visible;
6. mutable references use a named P11 profile;
7. migration histories cover loss, duplication, and fractured reads;
8. policy checks precede cost;
9. physical-only baselines receive the same ordinary statistics;
10. semantic features are ablated individually;
11. maintenance and planning overhead are included;
12. failures, timeouts, partial answers, and stale answers are retained;
13. correctness failure blocks performance interpretation;
14. no sentence implies that network or consistency cost disappeared;

15. row-level shard-key changes carry cross-shard migration obligations;
16. mapping-independent pruning has a checkable independence witness;
17. current evidence labels match actual artifacts.

This correction draft follows an independent round-one review that requested minor revisions. It has not undergone terminal acceptance review. Its companion reproducibility record covers only the isolated LaTeX build, citation closure, source hashes, and rendered-page inspection.

References

[1] Stefan Aulbach, Torsten Grust, Dean Jacobs, Alfons Kemper, and Jan Rittinger. Multi-tenant databases for software as a service: Schema-mapping techniques. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 1195–1206, 2008. doi: 10.1145/1376616.1376736. URL https://doi.org/10.1145/1376616.1376736.

[2] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331, 1990. doi: 10.1145/93597.98741. URL https://doi.org/10.1145/93597.98741.

[3] Philip A. Bernstein and Dah-Ming W. Chiu. Using semi-joins to solve relational queries. *Journal of the ACM*, 28(1):25–40, 1981. doi: 10.1145/322234.322238. URL https://doi.org/10.1145/322234.322238.

[4] Philip A. Bernstein, Nathan Goodman, Eugene Wong, Christopher L. Reeve, and James B. Rothnie. Query processing in a system for distributed databases (SDD-1). *ACM Transactions on Database Systems*, 6(4):602–625, 1981. doi: 10.1145/319628.319650. URL https://doi.org/10.1145/319628.319650.

[5] Jose A. Blakeley, Per-Åke Larson, and Frank Wm. Tompa. Efficiently updating materialized views. In *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data*, pages 61–71, 1986. doi: 10.1145/16894.16861. URL https://doi.org/10.1145/16894.16861.

[6] Mihai Budiu, Tej Chajed, Frank McSherry, Leonid Ryzhyk, and Val Tannen. DBSP: Automatic incremental view maintenance for rich query languages. *Proceedings of the VLDB Endowment*, 16(7):1601–1614, 2023. doi: 10.14778/3587136.3587137. URL https://doi.org/10.14778/3587136.3587137.

[7] Michael Busch, Krishna Gade, Brian Larson, Patrick Lok, Samuel Luckenbill, and Jimmy Lin. Earlybird: Real-time search at Twitter. In *2012 IEEE 28th International Conference on Data Engineering*, pages 1360–1369, 2012. doi: 10.1109/ICDE.2012.149. URL https://doi.org/10.1109/ICDE.2012.149.

[8] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems*, 26(2):1–26, 2008. doi: 10.1145/1365815.1365816. URL https://doi.org/10.1145/1365815.1365816.

[9] Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. SPANN: Highly-efficient billion-scale approximate nearest neighborhood search. In *Advances in Neural Information Processing Systems*, volume 34, pages 5199–5212, 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/hash/299dc35e747eb77177d9cea10a802da2-Abstract.html. Official NeurIPS proceedings record.

[10] James Cheney, Laura Chiticariu, and Wang-Chiew Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4):379–474, 2009. doi: 10.1561/1900000006. URL https://doi.org/10.1561/1900000006.

[11] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. Spanner: Google's globally-distributed database. In *10th USENIX Symposium on Operating Systems Design and Implementation*, pages 251–264, 2012. URL https://www.usenix.org/conference/osdi12/technical-sessions/presentation/corbett.

[12] Carlo Curino, Evan Jones, Yang Zhang, and Sam Madden. Schism: A workload-driven approach to database replication and partitioning. *Proceedings of the VLDB Endowment*, 3(1–2):48–57, 2010. doi: 10.14778/1920841.1920853. URL https://doi.org/10.14778/1920841.1920853.

[13] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. In *Proceedings of the Twenty-First ACM Symposium on Operating Systems Principles*, pages 205–220, 2007. doi: 10.1145/1294261.1294281. URL https://doi.org/10.1145/1294261.1294281.

[14] David DeWitt and Jim Gray. Parallel database systems: The future of high performance database systems. *Communications of the ACM*, 35(6):85–98, 1992. doi: 10.1145/129888.129894. URL https://doi.org/10.1145/129888.129894.

[15] David J. DeWitt, Alan Halverson, Rimma Nehme, Srinath Shankar, Josep Aguilar-Saborit, Artin Avanes, Miro Flasza, and Jim Gramling. Split query processing in PolyBase. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 1255–1266, 2013. doi: 10.1145/2463676.2463709. URL https://doi.org/10.1145/2463676.2463709.

[16] Jennie Duggan, Aaron J. Elmore, Michael Stonebraker, Magda Balazinska, Bill Howe, Jeremy Kepner, Sam Madden, David Maier, Tim Mattson, and Stan Zdonik. The BigDAWG polystore system. *ACM SIGMOD Record*, 44(2):11–16, 2015. doi: 10.1145/2814710.2814713. URL https://doi.org/10.1145/2814710.2814713.

[17] Jon Gjengset, Malte Schwarzkopf, Jonathan Behrens, Lara Timbó Araújo, Martin Ek, Eddie Kohler, M. Frans Kaashoek, and Robert Morris. Noria: Dynamic, partially-stateful data-flow for high-performance web applications. In *13th USENIX Symposium on Operating Systems Design and Implementation*, pages 213–231, 2018. URL https://www.usenix.org/conference/osdi18/presentation/gjengset.

[18] Goetz Graefe. Volcano: An extensible and parallel query evaluation system. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):120–135, 1994. doi: 10.1109/69.273032. URL https://doi.org/10.1109/69.273032.

[19] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 31–40, 2007. doi: 10.1145/1265530.1265535. URL https://doi.org/10.1145/1265530.1265535.

[20] Sairam Gurajada, Stephan Seufert, Iris Miliaraki, and Martin Theobald. TriAD: A distributed shared-nothing RDF engine based on asynchronous message passing. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, pages 289–300, 2014. doi: 10.1145/2588555.2610511. URL https://doi.org/10.1145/2588555.2610511.

[21] Jiewen Huang, Daniel J. Abadi, and Kun Ren. Scalable SPARQL querying of large RDF graphs. *Proceedings of the VLDB Endowment*, 4(11):1123–1134, 2011. doi: 10.14778/3402707.3402747. URL https://doi.org/10.14778/3402707.3402747.

[22] Christian S. Jensen and Richard T. Snodgrass. Temporal data management. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):36–44, 1999. doi: 10.1109/69.755613. URL https://doi.org/10.1109/69.755613.

[23] Robert Kallman, Hideaki Kimura, Jonathan Natkins, Andrew Pavlo, Alexander Rasin, Stanley Zdonik, Evan P. C. Jones, Samuel Madden, Michael Stonebraker, Yang Zhang, John Hugg, and Daniel J. Abadi. H-Store: A high-performance, distributed main memory transaction processing system. *Proceedings of the VLDB Endowment*, 1(2):1496–1499, 2008. doi: 10.14778/1454159.1454211. URL https://doi.org/10.14778/1454159.1454211.

[24] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 654–663, 1997. doi: 10.1145/258533.258660. URL https://doi.org/10.1145/258533.258660.

[25] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998. doi: 10.1137/S1064827595287997. URL https://doi.org/10.1137/S1064827595287997.

[26] Lothar F. Mackert and Guy M. Lohman. R* optimizer validation and performance evaluation for distributed queries. In *Proceedings of the 12th International Conference on Very Large Data Bases*, pages 149–159, 1986. URL https://www.vldb.org/conf/1986/P149.PDF. Official VLDB proceedings paper.

[27] Frank McSherry, Derek G. Murray, Rebecca Isaacs, and Michael Isard. Differential dataflow. In *Sixth Biennial Conference on Innovative Data Systems Research*, 2013. URL https://www.microsoft.com/en-us/research/publication/differential-dataflow/. Official Microsoft Research publication record.

[28] Abhishek Modi, Kaushik Rajan, Srinivas Thimmaiah, Prakhar Jain, Swinky Mann, Ayushi Agarwal, Ajith Shetty, Shahid K. I, Ashit Gosalia, and Partho Sarthi. New query optimization techniques in the Spark engine of Azure Synapse. *Proceedings of the VLDB Endowment*, 15(4):936–948, 2022. doi: 10.14778/3503585.3503601. URL https://doi.org/10.14778/3503585.3503601.

[29] Amihai Motro. Completeness information and its application to query processing. In *Proceedings of the 12th International Conference on Very Large Data Bases*, pages 170–178, 1986. URL https://www.vldb.org/conf/1986/P170.PDF. Official VLDB proceedings paper.

[30] Thomas Neumann and Gerhard Weikum. RDF-3X: A RISC-style engine for RDF. *Proceedings of the VLDB Endowment*, 1(1):647–659, 2008. doi: 10.14778/1453856.1453927. URL https://doi.org/10.14778/1453856.1453927.

[31] P. Griffiths Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, pages 23–34, 1979. doi: 10.1145/582095.582099. URL https://doi.org/10.1145/582095.582099.

[32] Alexander Shraer, John Quinn, Jonathan Ruben, Michael Ford, Mike McMahon, Nathan Williams, Nicolas Favre-Felix, Nihar Sharma, Ori Herrnstadt, Paul Seligman, Raghav Pisolkar, Alexandre Aybes, Scott Dugas, Scott Gray, Shirley Lu, Sytze Harkema, Valentin Kravtsov, Vanessa Hong, Yizuo Tian, Wan Ling Yih, Bryan Davis, Christos Chrysafis, Dave Browning, Eric Krugler, Eric Stone, Harrison Chandler, and Jacob Farkas. CloudKit: Structured storage for mobile applications. *Proceedings of the VLDB Endowment*, 11 (5):540–552, 2018. doi: 10.1145/3187009.3164138. URL https://doi.org/10.1145/3187009.3164138.

[33] Sivaprasad Sudhir, Wenbo Tao, Nikolay Laptev, Cyrille Habis, Michael Cafarella, and Samuel Madden. Pando: Enhanced data skipping with logical data partitioning. *Proceedings of the VLDB Endowment*, 16(9):2316–2329, 2023. doi: 10.14778/3598581.3598601. URL https://doi.org/10.14778/3598581.3598601.

[34] Rebecca Taft, Essam Mansour, Marco Serafini, Jennie Duggan, Aaron J. Elmore, Ashraf Aboulnaga, Andrew Pavlo, and Michael Stonebraker. E-Store: Fine-grained elastic partitioning for distributed transaction processing systems. *Proceedings of the VLDB Endowment*, 8(3):245–256, 2014. doi: 10.14778/2735508.2735514. URL https://doi.org/10.14778/2735508.2735514.

[35] Alexander Thomson, Thaddeus Diamond, Shu-Chun Weng, Kun Ren, Philip Shao, and Daniel J. Abadi. Calvin: Fast distributed transactions for partitioned database systems. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 1–12, 2012. doi: 10.1145/2213836.2213838. URL https://doi.org/10.1145/2213836.2213838.

[36] Cathrin Weiss, Panagiotis Karras, and Abraham Bernstein. Hexastore: Sextuple indexing for semantic web data management. *Proceedings of the VLDB Endowment*, 1(1): 1008–1019, 2008. doi: 10.14778/1453856.1453965. URL https://doi.org/10.14778/1453856.1453965.