

Coordination

Concurrent Systems as a Foundation for Artificial Organizations

Dr. Paulo Salem · Research report · 15 September 2026

Information researched and text written by AI under the general direction of Dr. Paulo Salem. This report complements the Artificial Organizations survey-tutorial. It examines how concurrent participants communicate, synchronize, claim resources, and recover, so readers can select a computational substrate without confusing it with an organization's authority, duties, or acceptance rules.

Finding. The existing organization-theory and multi-agent account needs a third foundation: **coordination in concurrent and distributed systems**. A role defines responsibility; a protocol defines interaction; a coordination mechanism determines which concurrent transitions may actually occur. All three must agree at consequential boundaries.

1. Executive Conclusions

- 1. Recover Linda's abstraction, not just its terminology.** A tuple space is an associative coordination medium with consuming and non-consuming operations, not a synonym for any shared database, vector store, or group chat.
- 2. Atomic claim is narrower than reliable work.** Removing one task tuple can choose one consumer. It does not recover a crashed consumer, deduplicate repeated publication, persist an outcome, or prevent a repeated external effect.
- 3. Separate work distribution from authority.** A participant able to match or remove a record is not thereby entitled to perform the act represented in it.
- 4. Put agreement where the invariant needs it.** Evidence accumulation can often proceed with little synchronization. Exclusive admission, budget limits, revocation, and final closure need more careful coordination or explicit rights partitioning. Neither global consensus everywhere nor eventual convergence everywhere is an adequate default.
- 5. Use deployed lessons about ambiguity.** Chubby's sequencers address stale lock holders at receiving services. ZooKeeper's recipes address lost replies after successful creation. Those failures also occur when the client is an agent.
- 6. Measure the real path.** Validate claims, retries, stale actors, resource limits, and recovery against the service that owns the effect. A successful local tuple-space demonstration is not a distributed reliability evaluation.

2. Scope, Method, and Evidence

This is a **focused narrative review**, not a systematic review with exhaustive database coverage. It follows Linda's primary literature into coordination languages and related models, then triangulates with fault-tolerant services, language/runtime documentation, and work on coordination avoidance. The research prioritizes semantic mechanisms and counterexamples over current popularity.

The accompanying research index records source access and integration locations. Full-text passages were inspected for the 1992 and 1994 Linda papers, Hoare's CSP paper, Chubby, CALM, invariant confluence, and the CRDT report. Gelernter's 1985 abstract establishes the original generative-communication claim; the later primary Linda paper supplies the detailed operations. Reo's abstract supports the exogenous-connector comparison. Some extension publications were identified through bibliographic metadata only and remain explicit deeper-reading leads. No access-blocked paper is represented as a completed implementation audit.

Versioned Apache River and Ruby documentation establish concrete API behavior. The retained experiment uses the machine's **Ruby 2.6.10 Rinda**, not a new tuple space written for this report. Its local findings are narrower than the Ruby 3.1 documentation used as an additional reference. No cloud infrastructure, live company data, paid model service, or business mutation was used.

3. Four Meanings of Coordination

LAYER	QUESTION	REPRESENTATIVE OBJECT	WHAT SUCCESS AT THIS LAYER CANNOT ESTABLISH
Organizational	Who is responsible, permitted, or obliged?	Role, mission, directed commitment	That concurrent processes implement the duty safely
Protocol	Which messages and interaction states are meaningful?	KQML act, MCP call, A2A task	That the receiver acted only once or had authority
Computational	Which accesses, claims, and transitions synchronize?	Tuple operation, channel rendezvous, transaction	That the resulting act is legitimate or useful
Operational	What survives delay, loss, crash, and restart?	Durable record, lease, epoch, recovery procedure	That the business outcome met its acceptance criteria

Gelernter and Carriero's separation of **computation** from **coordination** offers an important architectural principle [R2]: the numerical or symbolic work of a component need not encode the entire interaction topology. But separation does not mean independence of contracts. Changing a template, buffer, or connector can change which participant receives work, when it can proceed, and whether it ever finishes. The coordination medium is executable behavior, not passive plumbing.

In an artificial organization, coordination includes both immutable evidence exchange and mutually exclusive rights to change the world. These have different requirements. A common record format is useful; a single consistency policy for every record is often not.

4. Linda: Generative Communication

Linda introduces a logically shared **tuple space**, commonly understood as a multiset of typed tuples. A tuple is an ordered collection of fields. Participants publish and retrieve tuples by matching field values and formal placeholders rather than by addressing a receiving process [R1–R3]. The 1994 paper explicitly distinguishes the language model from different implementation strategies.

Spatial decoupling means a producer need not name its consumer. **Temporal decoupling** means the publication and retrieval need not happen simultaneously: the tuple remains available until consumed under the space's semantics. Neither property means the space survives a machine crash. Logical residence and physical durability are separate claims.

OPERATION	CORE PURPOSE	WAITING AND MUTATION	IMPORTANT LIMIT
<code>out</code>	Publish an evaluated tuple	Adds a tuple; no matching receiver need be ready	Repeating publication may create duplicates
<code>in</code>	Retrieve and consume a matching tuple	Waits for a match; removes one selected tuple	Claim does not acknowledge finished work
<code>rd</code>	Observe a matching tuple	Waits for a match; leaves it present	Observation does not reserve it
<code>eval</code>	Create a tuple through concurrent evaluation	Active computations eventually yield a passive tuple	Not every Linda-like implementation provides this operation
<code>inp</code> , <code>rdp</code>	Probe variants in relevant dialects	Report failure without waiting for a future tuple	Local absence is not global completion or durable absence

Matching normally considers tuple length, types, and specified values. If several tuples match, the abstract model does not entitle the application to assume oldest-first delivery, priority, fairness, or a particular consumer. An implementation may add policies; those policies must be documented and tested. Duplicates matter because a multiset is not a uniqueness-constrained table.

A Primary-Source Operation

The 1994 Linda paper gives the following expression [R3, p. 636]:

```
out('table entry', i, j, f(i,j))
```

This is the source's illustrative Linda notation, not portable modern C syntax. The fields are evaluated before publication as a passive tuple. Its neighboring `eval` discussion distinguishes concurrent evaluation from publishing an already computed value. The field variables and function belong to the paper's example; the excerpt was inspected, not compiled with a historical C-Linda toolchain.

The distinction becomes practical when work contains expensive computation. Publishing a description of a task, spawning its computation, and publishing its result are different operations. A queue of descriptions does not become an execution engine merely because it is called a tuple space.

Where Linda Helps

Linda is attractive for dynamically sized worker pools, associative work discovery, and applications in which participants should not depend on each other's identities. The source literature discusses distributed data structures, parallel computation, and implementations on different machine architectures. Its central gain is a stable interaction vocabulary while the participating computation changes.

The cost moves into matching, tuple placement, indexing, contention, and recovery. Broad wildcard reads can conceal accidental coupling. A central implementation can be a bottleneck; distributing matching introduces further consistency and placement decisions. A logical single space is not evidence of a single physical server, nor proof that a distributed implementation is efficient.

5. A Real Implementation: Rinda

Masatoshi Seki's *The dRuby Book*, Section 6.2, demonstrates this exchange:¹

```
$ts.write(["take-test", 1])
$ts.take(["take-test", nil])
```

The returned tuple is `["take-test", 1]`. `nil` is a wildcard in this template. The original walkthrough uses a dRuby service and multiple terminals; this report's retained harness executes the operations against an in-process `Rinda::TupleSpace`. It does not run the book's network topology.

The same source publishes a factorial request, consumes it at a worker, and publishes a result with a different tuple tag. That concrete separation is more instructive than a generic arrow marked “collaboration”: request and answer are different objects, and the client waits for the answer. The harness reproduces the calculation yielding `['fact-answer', 1, 5, 120]`. Identical requests without distinct correlation identifiers still need a policy for identifying results.

1. Masatoshi Seki, *The dRuby Book*, online Section 6.2, “How Rinda Works,” including `take-test`, duplicate tuples, class matching, and factorial examples. Inspected 15 September 2026. <https://www.druby.org/sidruby/6-2-how-rinda-works.html>. Ruby 3.1 API reference: <https://ruby-doc.org/stdlib-3.1.0/libdoc/rinda/rdoc/Rinda/TupleSpace.html>.

What We Actually Tested

CHECK	OBSERVED RESULT	CLAIM JUSTIFIED
Repeated non-destructive read	Same tuple remains readable	Reading does not consume that tuple
Two threads attempt zero-wait take on one tuple	One success, one <code>RequestExpiredError</code>	One tuple is not handed to both takers in this test
Publish the same tuple twice	Two takes succeed	Publication does not deduplicate these equal values
Match integer fields against integer and float candidates	Integer tuple selected; float tuple remains	This concrete Rinda template distinguishes these field types
Factorial request and answer	Result is 120	The local source-derived request/result workflow works
Consumer started before later publication	Consumer returns the published tuple	Producer/consumer startup order need not be hardwired
Inspect completion records after a bare take	No completion tuple appears	Claim alone does not manufacture a result

The harness has a ten-second outer deadline and uses explicit thread/queue synchronization, not arbitrary sleeps. It does **not** prove fairness, model all interleavings, kill the tuple-space process, test a partition, or exercise disk persistence. A thread no longer doing useful work and a dead remote process are not interchangeable failure models. The harness is a semantic regression aid, not a production queue recommendation.

6. Related Models: Change the Mechanism, Not Just the Name

FAMILY	COORDINATION MEDIUM	CHARACTERISTIC SYNCHRONIZATION	PREFER IT WHEN	DO NOT ASSUME
Linda / tuple spaces	Associatively matched shared tuples	Blocking read; exclusive consuming operation	Work discovery should not name recipients	FIFO, durable delivery, completion acknowledgement
Blackboard architecture	Shared partial problem solution and control policy	Scheduler chooses useful knowledge-source activations	Diagnosis or synthesis is opportunistic	A tuple store supplies the scheduler or acceptance rule
CSP-style channels	Explicit communication topology	In the original model, matching input/output rendezvous	Protocol sequencing and backpressure matter	All practical channel libraries are unbuffered or identical to CSP
Actors	Addressed participants with private state	Asynchronous messages; local handling discipline	Ownership and encapsulated state are central	Global order, durable mailboxes, or exactly-once effects
Kahn process networks	Directed FIFO streams	Blocking reads; idealized unbounded channels	Deterministic stream transformation fits the task	Arbitrary mutable agents preserve network determinism
Reo	Composed connectors outside components	Connector-specific coordination constraints	Interaction policy should be composed independently	Every connector is a queue or a Linda space
Transactional database / queue	Records and acknowledged work	Transactions, compare-and-swap, visibility or acknowledgement policy	Durable ownership and queryable lifecycle dominate	Any queue automatically covers the remote side effect
Replicated coordination service	Small ordered metadata	Atomic updates, sessions, version checks	Leader identity, configuration, membership, or locks	It should transport every large work artifact
CRDT / monotonic dataflow	Mergeable facts or lattice state	Progress without a global barrier where assumptions permit	Independently accumulated information is useful	Convergence enforces every business invariant

Hoare's original CSP paper makes input/output synchronization a first-class structuring mechanism [R7]. This is a different decoupling choice from Linda: the rendezvous constrains when participants proceed. A buffered channel changes that timing; a finite buffer can provide backpressure but can also introduce deadlock if a cycle is incorrectly designed.

Kahn's model provides a semantic foundation for deterministic process networks [R8]. Its useful lesson is to make data dependencies explicit. The deterministic result depends on the process and stream model, not merely drawing arrows. Finite-memory implementations and processes that test absence or race for

shared mutable data need additional analysis. The retrieved Kahn PDF is image-based; this report treats the precise formal development as a primary-reading lead, not a reproduced proof.

Actors shift the coordination boundary to addressed, state-owning participants. Erlang's actual documentation makes the operational limits tangible: signal delivery across a failed distribution channel can be lost, and its ordering guarantee concerns one sending entity and one receiving entity, not a global order. A connection-loss monitor does not prove the remote process stopped.² Supervision can restart a participant; it does not roll back an external payment.

Reo makes **exogenous coordination** explicit: connectors impose interaction patterns on components through their input/output ends [R6]. This is especially relevant to agent systems that otherwise bury coordination decisions in prompts. But connector composition has its own semantics; naming a Python router “Reo-like” does not establish equivalence to the model.

7. The Tuple-Space Research Lineage

BRANCH	WHAT CHANGES RELATIVE TO THE BASIC SPACE	EVIDENCE STRENGTH IN THIS REVIEW
JavaSpaces	Distributed entries with transaction, lease, wait, and notification contracts	Versioned API inspected
IBM TSpaces	Data management, communication, and computation combined in middleware	IBM publication abstract inspected
Tuple centres	Programmable coordination behavior in the medium	Bibliographic verification; runtime details remain a lead
KLAIM	Locality and mobile-agent interaction enter the language design	Bibliographic verification; no type-system audit
LIME	Mobility makes shared coordination state a distinct concern	Paper identified; project access blocked
Reo	Coordination expressed as composed connectors	Primary abstract inspected

IBM TSpaces is more than a renamed operation set. The 1998 IBM paper frames its middleware around data management, computation, and communication [R19]. That motivates inspecting storage and execution services independently of tuple matching. The abstract's broad claims about filling a middleware void are authors' positioning, not a historical priority judgment adopted here. No current TSpaces deployment or supported distribution was verified.

The survey by Papadopoulos and Arbab [R4] is an important route into the wider coordination-language literature; it prevents a false history in which everything between Linda and contemporary agent orchestration disappears. Its bibliographic record was checked, but its full taxonomy was not reproduced in this review.

2. Erlang/OTP, “Processes,” especially “Delivery of Signals,” “Monitors,” and “Receiving Signals,” inspected 15 September 2026. This living reference includes release-specific behavior; ordinary signal order and mailbox selection should not be conflated. https://www.erlang.org/doc/system/ref_man_processes.html.

JavaSpaces adds a concrete distributed-object interface with entry matching, transactions, leasing, and notification. The Apache River 2.2.2 interface shows these as separate parameters and methods, rather than implied properties of a generic `take`.³

```
Lease write(Entry entry, Transaction txn, long lease)
Entry take(Entry tmpl, Transaction txn, long timeout)
```

These are signature reductions from the API, with declared exceptions omitted; they are not standalone Java declarations. An entry's requested lease is its residence duration under the service contract. It is **not a worker claim lease** and not a mandate to perform a business act. A transaction can group participating space operations, but an unrelated API does not become a transaction participant because its invocation occurs between a take and a commit. The nonblocking-looking `takeIfExists` can still wait for transactional state to settle; API names alone are not a latency contract. No River service was deployed here.

Tuple centres make coordination behavior programmable rather than leaving the tuple medium fixed [R5]. This connects to the TuCSoN/ReSpecT research line: reactions can control the consequences of tuple operations. This report identifies that direction, not a current release's reaction syntax or production guarantees; the verified record for R5 is bibliographic, not a full runtime audit.

KLAIM identifies localities and mobile interaction as a distinct branch [R9]. Naming and access across local spaces matter when there is no permanently shared global medium. Picco, Murphy, and Roman's LIME paper [R20] and related mobile-space work are further research leads: changing connectivity and shared-space availability need explicit treatment, but their detailed semantics were not validated here.

Shared-dataspace and chemical-style models, including Swarm and Gamma, widen the question from message delivery to guarded transformations of a shared collection. They are useful follow-up families, but this report does not equate their rewrite rules with Linda's operations or claim to have reproduced their operational semantics. The central reusable question is: **which coordination policy belongs in the medium, and who is allowed to change it?**

LVars connects coordination to lattice-based shared data and determinism [R10]. It is a useful bridge to the monotonicity discussion below, not a synonym for arbitrary CRDT objects. Only its bibliographic identity was verified here; threshold-read and determinism proofs need the original paper before a detailed implementation claim is made.

8. Atomicity, Safety, and Progress

An atomic operation behaves as an indivisible transition in its specified model. **Linearizability** further places each completed operation at a point between its invocation and response, respecting real-time precedence [R17]. Neither means that a sequence of operations is atomic. Two reads followed by a write can race even when every individual call is linearizable.

Safety asks whether something forbidden can happen. **Liveness** asks whether required progress eventually occurs under specified assumptions. Two workers cannot take one tuple twice in the basic claim test, yet the system can still wait forever for a result that a failed consumer never publishes. The first property

3. Apache River 2.2.2, `net.jini.space.JavaSpace` API, methods `write`, `read`, `take`, `takeIfExists`, and `notify`, inspected 15 September 2026. <https://river.apache.org/release-doc/2.2.2/api/net/jini/space/JavaSpace.html>.

does not imply the second.

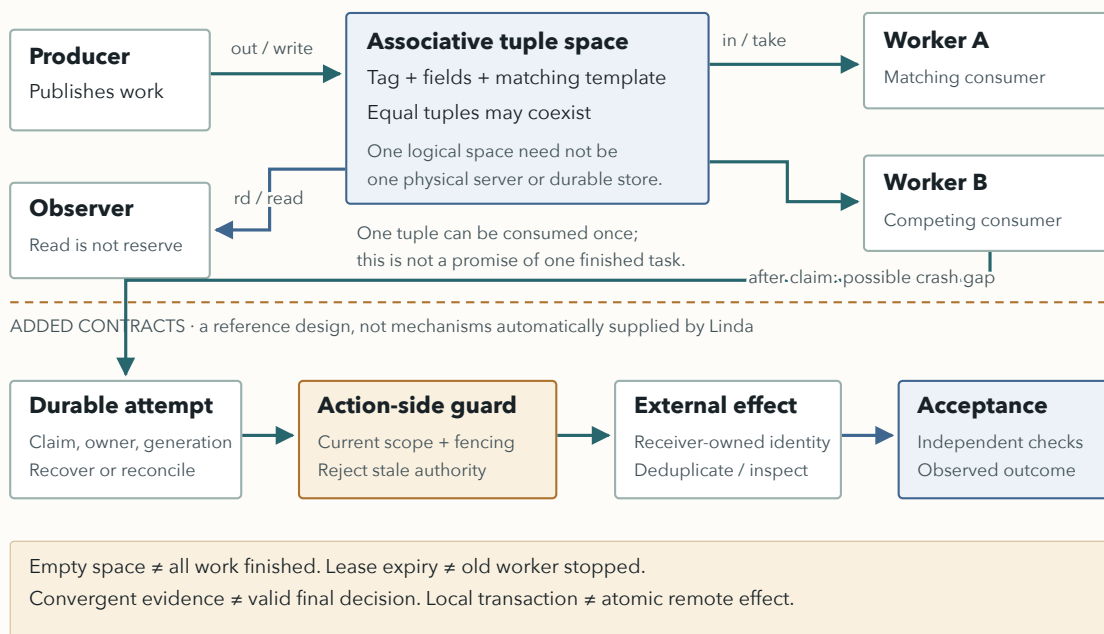
Absence tests are particularly dangerous. “No matching task now” can mean idle work, an in-flight claim, delayed publication, a disconnected producer, a failed monitor, or finished work. Distributed termination requires a completion protocol that accounts for producers, in-flight messages, and active work. A dashboard that displays an empty queue as success has confused a storage observation with a workflow invariant.

Similarly, barrier arrival does not imply the legitimacy of a decision. A barrier can wait for the wrong participants, hang after failure, or release on technical completion without checking evidence. The membership set, failure behavior, and meaning of arrival must all be explicit.

9. The Failure Windows That Matter

COORDINATION IS NOT COMPLETION

Linda-like space operations above; additional reliability and institutional contracts below.



Coordination boundaries: tuple claim, durable attempt, external effect, and observed completion.

WINDOW	WHAT CAN HAPPEN	NECESSARY DESIGN QUESTION
Publication succeeds; reply is lost	Retrying creates duplicate work	How is operation identity reconciled?
Tuple is taken; worker stops before recording an attempt	Work disappears from the available set	Is claim plus attempt creation transactional or otherwise recoverable?
Attempt recorded; result absent	Worker may be slow, stopped, or disconnected	Who owns retry, and what evidence permits reassignment?
Claim expires; old worker resumes	Old and new attempts overlap	Does the effect-owning service reject stale generations?
External effect succeeds; acknowledgement is lost	Retrying may duplicate the effect	Can the receiver deduplicate by a stable business-operation key?
Result published; verifier has not checked it	"Done" overstates the outcome	Which state distinguishes submission from acceptance?
Evidence collected; policy changes	Work proceeds under stale authority	Where is the active authority rechecked?

Fencing makes stale ownership visible to the resource being modified. A monotonically ordered epoch or another validated sequencer accompanies the request; the recipient rejects obsolete authority under its contract. A token merely attached to a log is not fencing. Checking a lease once and then calling an uncontrolled remote API leaves a race between the check and the effect.

There is no general exactly-once external-effect guarantee obtainable by attaching an idempotency key to a local queue. The receiving service must implement the key's scope, retention, conflict behavior, and atomic association with the effect. An outbox couples local state to publication when they share a transaction; an inbox/deduplication record can help on receipt. Neither retroactively enrolls an unrelated external system in that transaction. Compensation in a saga is a new business act, not time reversal [R18].

10. Operational Evidence: Chubby and ZooKeeper

Security boundary. A wildcard matching all tenants can leak or consume another tenant's work; a caller-supplied namespace is not authorization. Constrain access at the service boundary, validate schemas, restrict executable reactions, and keep sensitive payloads out of broadly discoverable tuples. Rinda's local convenience does not justify exposing a dRuby endpoint to untrusted clients. None of the local tests evaluates authentication, authorization, or arbitrary object deserialization.

Chubby at Google

Burrows reports Chubby use by GFS and Bigtable for such tasks as master election, discovery, and small metadata [R11, introduction]. This is a real operational case, not an invented agent company. The design emphasizes coarse-grained locks and reliable, low-volume storage rather than transporting all application data.

The paper's **sequencer** contains the lock name, acquisition mode, and generation number. A client sends it to the protected service, which is expected to reject invalid or inappropriate sequencers [R11, Section 2.4]. The same section discusses lock-delay as a less satisfactory alternative for services unable to validate sequencers. This supplies a precise lesson: recovery of lock ownership is not enough if an old holder's delayed request can still mutate the resource.

The organizational analogy is limited but useful. A new employee or agent can take over a role while the prior occupant still has in-flight requests. The institution must bind consequential operations to current authority, not just update the name displayed in a directory.

ZooKeeper at Yahoo and Beyond

Hunt et al.'s ZooKeeper work [R12] provides a distinct coordination-service model: clients build higher-level recipes over small hierarchical records. The official recipes demonstrate locks, elections, barriers, and membership. These are client protocols over service primitives, not business workflows built into a server.⁴

The lock recipe uses ephemeral sequential nodes, with contenders generally watching the predecessor rather than all watching one owner. This reduces the herd effect. Importantly, the recipe discusses **create succeeded but reply was lost** and uses a GUID to identify the node after reconnection. An absent response is not evidence of an absent transition.

The recipe's global-lock wording should not be read as a promise that a paused or partitioned client instantly knows its session expired. External resources still need protection from stale requests. Likewise, a watch is a prompt to re-read applicable state, not a general durable audit log of all intermediate changes. The documentation's ephemeral queue illustration should not be copied as a durable business-job queue without analyzing what session loss deletes.

The paper's read/write guarantees are not interchangeable: a read served locally is not automatically the latest globally committed observation. Designs that need fresh authoritative state must use the service's actual synchronization and versioning contract. This report does not claim to have run a ZooKeeper cluster or verified partition behavior.

11. When Coordination Can Be Avoided

Hellerstein and Alvaro explain the CALM result as a connection between logical monotonicity and consistent coordination-free distributed computation [R13]. **Monotonicity** means that adding inputs does not invalidate previously justified outputs. The result is about a program's semantics in a specified distributed model, not a guarantee of instantaneous agreement, zero communication, or an absence of runtime synchronization.

Their contrast between detecting a cycle and establishing non-reachability is instructive. Additional edges can preserve a discovered cycle, while a conclusion that no path exists can be invalidated by a late edge. The source discussion assumes the relevant fact model; a changing real-world graph needs explicit time or version semantics. "No evidence of a problem" is similarly not a monotonic certificate that the problem does not exist.

4. Apache ZooKeeper 3.9 documentation, "Recipes and Solutions," particularly "Important Note About Error Handling," "Locks," and "Leader Election"; inspected 15 September 2026. <https://zookeeper.apache.org/doc/current/recipes.html>. Publication and official paper route: <https://www.usenix.org/conference/usenix-atc-10/zookeeper-wait-free-coordination-internet-scale-systems>.

CRDTs provide sufficient algebraic and delivery conditions for convergence [R14]. For a state-based design, a join-semilattice with inflationary updates and a commutative, associative, idempotent merge can make merge order irrelevant. Operation-based designs require their own operation and delivery conditions; “CRDT” does not mean unreliable arbitrary operations will eventually work. Convergence concerns replicas' agreement, not the truth of input claims. The inspected report assumes non-Byzantine behavior and memory that survives process crashes [R14, Section 2]. Its introduction also notes that some designs grow without bound and that reclaiming metadata can require synchronization. Do not discard those qualifications when applying the result to evidence from untrusted or potentially incorrect agents.

Invariant confluence asks a different, application-centred question: can locally valid concurrent states be merged while preserving the chosen invariant [R15]? Some operations and constraints permit coordination avoidance; others do not. For bounded spending or limited stock, independently using the same remaining capacity can violate the global limit. **Preallocated rights or escrow** can permit local operations within disjoint allocations, but allocation and transfer of those rights must themselves preserve the invariant. Renaming a budget counter a CRDT does not solve overspending.

Transfer to artificial organizations: accumulate immutable observations, source hashes, and submitted claims without making every contributor wait for global agreement. Coordinate when granting scarce rights, revoking authority, selecting the active policy, or declaring final acceptance under a fixed evidence set. Store both “this claim was submitted” and “this claim is currently accepted” if needed: the former can remain true when the latter changes. This is a design synthesis, not a theorem that every evidence workflow is monotonic.

12. Selection and Rejection Rules

REQUIREMENT	STARTING POINT	REJECT OR EXTEND IT WHEN
Associative discovery among interchangeable workers	Tuple-space semantics	Matching scope, durable recovery, or priority cannot be expressed or verified
Ordered stream with bounded producer pressure	Channels or stream processing	Cross-stream joins, shared invariants, or durable replay need more mechanisms
Encapsulated state with addressed requests	Actor/runtime service	Mailbox growth, lifecycle persistence, or cross-actor invariants remain unspecified
Durable task lifecycle and auditable ownership	Transactional work records or established queue/workflow engine	Claim and completion state are app-private conventions with no recovery tests
Small shared authority/configuration metadata	Replicated coordination service	It is being used as a bulk artifact transport or as a substitute for resource fencing
Mergeable evidence accumulation	CRDT or monotonic representation	Deletion, absence, revocation, or exclusive rights violate the assumptions
Reusable interaction constraints outside participants	Connector or programmable-medium approach	Semantics, tooling, and deployment support cannot justify the integration cost

The smallest suitable design can use more than one mechanism. A durable work record may use a transactional claim; artifact content may live in a separate store; observational facts may replicate independently; one guarded boundary may serialize a scarce right. The aim is **minimal necessary coordination**, not maximum architectural variety.

Performance claims need the correct denominator: useful accepted outcomes per unit cost, not tuple operations or messages per second alone. Measure matching selectivity, payload size, active participants, contention, tail latency, queue age, retry amplification, fairness, and restoration time. The 1994 Linda paper reports machine- and workload-specific results and explicitly notes a small-data latency disadvantage on one platform [R3, p. 653]. It supports neither “Linda is always slow” nor “tuple spaces outperform message passing everywhere.”

13. Integration into the Artificial Organizations Survey

The new foundation changes existing explanations, not merely the reference list:

- **Field map and history:** concurrent-system coordination is a separate lineage alongside organization science, MAS, and current agent runtimes.
- **Topology:** distinguish organizational patterns from their synchronization substrate; blackboard, tuple space, actor, and queue are not synonyms.
- **Delegation and policy:** role replacement and lease expiry require action-side enforcement; matching a task does not grant authority.
- **Evidence and closure:** queue emptiness, process termination, and protocol completion are not independent acceptance. Preserve attempts and external ambiguity.
- **Memory:** distinguish append-only evidence from non-monotonic active decisions; replica convergence does not establish policy validity.
- **Runtime:** place claim, fencing, idempotency, and persistence at the owning service boundary; use existing infrastructure rather than a bespoke “agent queue.”
- **Construction and learning checks:** test lost replies, stale holders, duplicate publication, worker failure after claim, and replay after effects.

14. Validation Agenda and Open Questions

The Rinda harness is a completed local semantic check. The next appropriate deployment tests require an isolated environment for the chosen real substrate:

1. Crash after successful claim but before completion; reconnect and inspect durable ownership and recovery eligibility.
2. Drop the success reply after publication or effect execution; retry the same operation identity and verify deduplication or explicit ambiguity.
3. Pause an owner beyond lease expiry, admit a successor, then resume the old owner; verify that the resource rejects its stale write.

- 4.** Change authority while work is live; prove that new operations use the active rules without pretending prior external effects vanished.
- 5.** Reorder and duplicate notifications; reconstruct state from authoritative records rather than notification counts.
- 6.** Withhold one producer or result; verify that empty work state does not cause premature closure and that deadlines produce explicit failure or uncertainty.
- 7.** Compare a simpler single-worker path with the coordinated one on accepted outcomes, time, cost, and harm, not merely aggregate throughput.

Open questions specific to artificial organizations include how much shared coordination logic can be generated safely from natural-language policy; how to preserve task semantics during schema and role migrations; how expensive pattern matching interacts with large agent populations; and how to audit claims that cross heterogeneous substrates. There is no evidence here that choosing Linda alone resolves these questions.

References

- R1.** Gelernter, D. (1985). "Generative Communication in Linda." *ACM TOPLAS*, 7(1), 80–112. DOI: <https://doi.org/10.1145/2363.2433>. Abstract inspected; detailed operation account cross-checked against R3.
- R2.** Gelernter, D., & Carriero, N. (1992). "Coordination Languages and Their Significance." *Communications of the ACM*, 35(2), 97–107. <https://doi.org/10.1145/129630.129635>. [Inspected full-text copy](#).
- R3.** Carriero, N., Gelernter, D., Mattson, T., & Sherman, A. (1994). "The Linda Alternative to Message-Passing Systems." *Parallel Computing*, 20, 633–655. [https://doi.org/10.1016/0167-8191\(94\)90032-9](https://doi.org/10.1016/0167-8191(94)90032-9). [Inspected academic copy](#).
- R4.** Papadopoulos, G. A., & Arbab, F. (1998). "Coordination Models and Languages." *Advances in Computers*, 46, 329–400. [https://doi.org/10.1016/S0065-2458\(08\)60208-9](https://doi.org/10.1016/S0065-2458(08)60208-9). Bibliographic check only.
- R5.** Omicini, A., & Denti, E. (2001). "From Tuple Spaces to Tuple Centres." *Science of Computer Programming*, 41(3), 277–294. [https://doi.org/10.1016/S0167-6423\(01\)00011-9](https://doi.org/10.1016/S0167-6423(01)00011-9). Bibliographic check; detailed reaction semantics and implementation remain further-reading work.
- R6.** Arbab, F. (2004). "Reo: A Channel-Based Coordination Model for Component Composition." *Mathematical Structures in Computer Science*, 14(3), 329–366. <https://doi.org/10.1017/S0960129504004153>. Abstract inspected; no connector engine run.
- R7.** Hoare, C. A. R. (1978). "Communicating Sequential Processes." *Communications of the ACM*, 21(8), 666–677. [Inspected academic copy](#).
- R8.** Kahn, G. (1974). "The Semantics of a Simple Language for Parallel Programming." *Information Processing 74*, 471–475. [Academic scan](#). Image-based scan located; the formal proof was not independently reproduced.
- R9.** De Nicola, R., Ferrari, G. L., & Pugliese, R. (1998). "KLAIM: A Kernel Language for Agents Interaction and Mobility." *IEEE TSE*, 24(5), 315–330. <https://doi.org/10.1109/32.685256>. Bibliographic check only.
- R10.** Kuper, L., & Newton, R. R. (2013). "LVars: Lattice-Based Data Structures for Deterministic Parallelism." *Haskell Symposium*, 71–84. <https://doi.org/10.1145/2502323.2502326>. Bibliographic check only.
- R11.** Burrows, M. (2006). "The Chubby Lock Service for Loosely-Coupled Distributed Systems." *OSDI*, 335–350. [Inspected paper](#).
- R12.** Hunt, P., Konar, M., Junqueira, F. P., & Reed, B. (2010). "ZooKeeper: Wait-free Coordination for Internet-scale Systems." *USENIX ATC*. [Official publication and paper route](#). Publication page and official recipes inspected; direct full-paper download returned an access error in this session.
- R13.** Hellerstein, J. M., & Alvaro, P. (2019). "Keeping CALM: When Distributed Consistency Is Easy." <https://arxiv.org/abs/1901.01930v2>. Full-text passages inspected.
- R14.** Shapiro, M., Preguiça, N., Baquero, C., & Zawirski, M. (2011). "A Comprehensive Study of Convergent and Commutative Replicated Data Types." INRIA RR-7506. [Open report](#).
- R15.** Bailis, P., Fekete, A., Franklin, M. J., Ghodsi, A., Hellerstein, J. M., & Stoica, I. (2014). "Coordination Avoidance in Database Systems." Extended version of *PVLDB* 8(3). <https://arxiv.org/abs/1402.2237v4>.
- R16.** Seki, M. *The dRuby Book*. Chapter 6, "Coordinating Processes Using Rinda." [Author's online edition](#). Concrete source examples and implementation documentation inspected; local behavior tested with the installed Ruby 2.6.10.
- R17.** Herlihy, M. P., & Wing, J. M. (1990). "Linearizability: A Correctness Condition for Concurrent Objects." *ACM TOPLAS*, 12(3), 463–492. <https://doi.org/10.1145/78969.78972>. Foundational definition; not a new proof of any implementation's linearizability.
- R18.** Garcia-Molina, H., & Salem, K. (1987). "Sagas." *ACM SIGMOD*, 249–259. <https://doi.org/10.1145/38714.38742>. Existing survey source, reused to connect local transactions to long-lived business processes.
- R19.** Wyckoff, P., McLaughry, S. W., Lehman, T. J., & Ford, D. A. (1998). "T Spaces." *IBM Systems Journal*, 37(3), 454–474. <https://doi.org/10.1147/sj.373.0454>. [IBM publication abstract](#) inspected; no current product-status or performance claim established.
- R20.** Picco, G. P., Murphy, A. L., & Roman, G.-C. (1999). "LIME: Linda Meets Mobility." *ICSE*, 368–377. <https://doi.org/10.1109/ICSE.1999.841027>. Bibliographic record verified; project-site access blocked, detailed semantics remain further research.