Database Reliability in the Context of Distributed Databases

Physics, consensus, concurrency and disaster recovery — the
— the trade-offs that decide whether your data survives,
survives, presented without a vendor scorecard.

Francisco Munoz Alvarez · Miracle · linkedin.com/in/franciscomunozalvarez
e

Francisco Munoz Alvarez

- Over Three decades keeping mission-critical Oracle platforms alive — Maximum Availability Architecture
- Advisor to enterprises worldwide on high availability and disaster recovery
- International keynote speaker, MBA — author of the forthcoming book Engineering Resilience: The High Availability Mindset
- Races for Australia at the Dragon Boat World Championships

francisco.munoz.alvarez@miracleltd.com

https://www.linkedin.com/in/franciscomunozalvarez

fcomunoz

Contents

01

The concepts — one vocabulary

Four building blocks — every vendor assembles the same kit

SHARDING

Split data horizontally; each shard owns a slice. Scale out by adding shards — at the cost of cross-shard coordination.

SHARED DISK / CACHE

Several compute nodes over one coherent copy — instance failure survivable without moving data.

REPLICATION / REPLICAS

Full copies, sync or async — reads scale out; writes pay the coordination or lag cost.

STANDBYS

Full copies whose job is takeover — the disaster-recovery block, measured in RPO and RTO.

Real systems **combine** the blocks. The interesting question is never “which block is modern” — it’s which combination meets **your** requirements.

Three replication levels — three different promises

STORAGE LEVEL

Blocks, beneath the DB

Transparent to the database — and blind to it. Consistency is whatever the array promises. E.g. storage mirroring, Aurora's storage tier.

LOGICAL / DATABASE LEVEL

One writer, replicated changes

The database ships changes it understands: redo, WAL, change records. Read-write primary, readable or standby copies. E.g. Data Guard, PostgreSQL streaming, AlloyDB.

DISTRIBUTED LEVEL

Consensus in the write path

Every commit clears a quorum — writes spread across nodes, latency coupled to the slowest majority member. E.g. Spanner, CockroachDB, YugabyteDB, TiDB.

Each level trades something. **Transparent but blind · aware but single-writer · multi-writer but quorum-bound.** There is no free level.

The terminology map

ROLE	CONSENSUS WORLD (RAFT)	PRIMARY/STANDBY WORLD	WHAT IT MEANS
Leader	Leader / RU leader	Primary	The one place writes are accepted
Voting copy	Follower (voting)	Sync standby · cluster node	Acknowledges before commit completes
Readable copy	Read-only / non-voting replica	Active (read) standby	Serves reads, never votes
Multi-writer	Rare — single leader per range	Logical replication (e.g. GoldenGate)	Several writers, conflict handling required

With the map in hand, every datasheet claim reduces to two precise questions: **who accepts the write, and who must acknowledge it?**

02

Consensus — RAFT, honestly

RAFT: an elected leader, a replicated log, a majority

LEADER

Accepts all client writes, appends entries to the log, replicates to followers. One per RAFT group (or per range/region of data).

FOLLOWERS

Replicate the leader's entries. A write is **committed when a majority acknowledges** — the quorum is the guarantee.

CANDIDATE

A follower that stops hearing the leader: increments the term, requests votes, wins with a majority — and the log resumes.

Elegant, automatic, arbiter-free — and honestly priced: **during an election, writes stop**, and every commit pays a majority round-trip.

Election time: the number vs the outage

Raw election vs the whole client-visible disruption — detection, election, lease transfer, retry

SYSTEM	RAFT IMPLEMENTATION	RAW ELECTION	CLIENT-VISIBLE WORST CASE
CockroachDB	Custom, per-range	~300 ms (tunable)	up to ~9 s
YugabyteDB	DocDB, strong sync	~150–300 ms (adaptive)	up to ~3 s (+TCP timeouts to 15 s)
TiDB	Per-region groups	~200 ms	up to ~20 s

The marketing number is the election; **the outage your user feels is the whole process** — and short-vs-long timeout tuning is a genuine trade every consensus system faces.

“Consistent” is three different contracts

CONTRACT	COCKROACHDB	YUGABYTEDB	TIDB
Write durability	Majority write	Majority write	RAFT-commit based
Default isolation	Serializable	Snapshot isolation	Snapshot isolation
Read replicas	Geo-partitioned — eventually consistent	Follower reads — timeline consistent	TiFlash — strong or stale-timestamp

Every row is documented behavior, not a defect. The point: **“replicas for DR reads” signs a different staleness contract in each system** — know which one your application signed.

03

Four factors that govern reliability

Latency: “I want RPO = 0” meets the speed of light

c ≈ 299,792 km/s — under 300 km per millisecond, before fiber, routers and TCP

Rule of thumb: every 150 km of distance ≥ 1 ms of round-trip

Consensus or sync standby, same bill: a cross-region acknowledgment taxes **every commit**

RPO = 0 at distance is always a choice about **who pays**: commit latency, or async copies

---

No distributed database defeats the speed of light. This constraint is identical for every architecture in this talk — the honest differences are in how each one lets you place the cost.

Optimistic vs pessimistic — a contract with your application

PESSIMISTIC

Lock first, conflicts wait

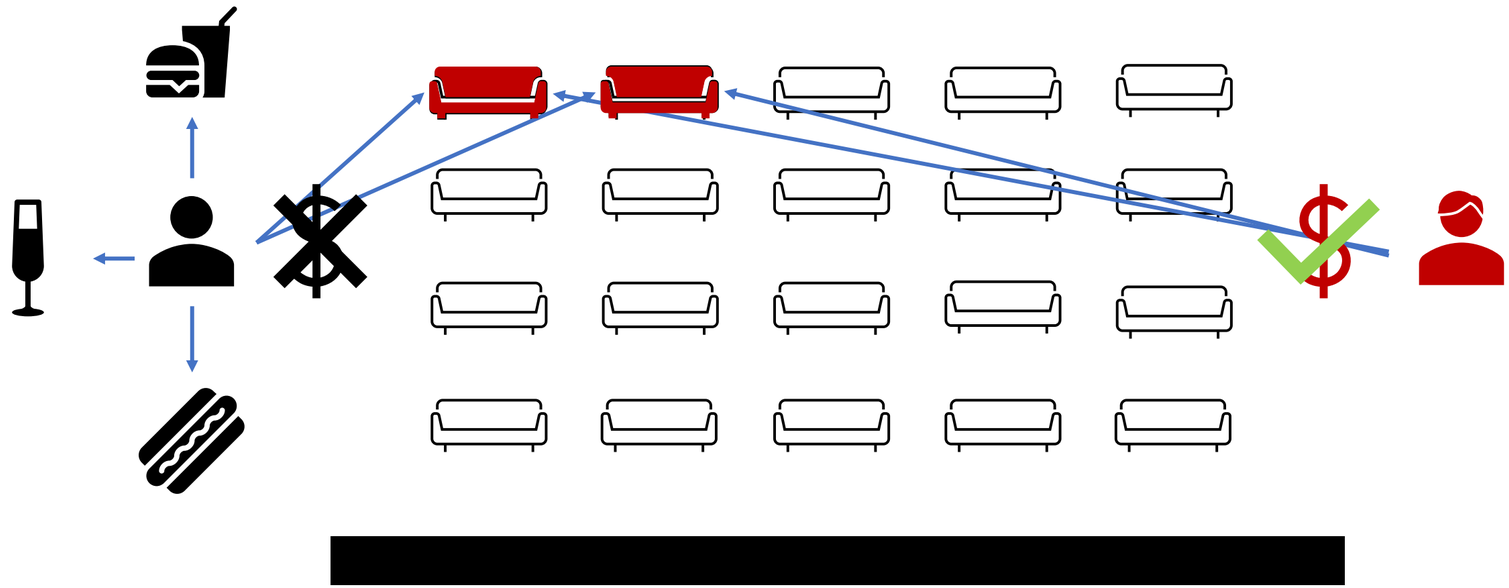
Contention serializes quietly; the app never sees “try again”. Cost: locks held, throughput under heavy contention. Default in TiDB and traditional engines (with MVCC reads).

OPTIMISTIC

Proceed, validate at commit

Excellent when conflicts are rare. Under hot-row contention the loser gets a serialization error — **and the retry is your application’s job** . Default in CockroachDB; isolation-dependent in YugabyteDB.

The migration trap: a pessimistic-era application moved onto an optimistic-default engine doesn’t fail in testing — **it fails under production contention, as retry storms nobody coded for** . Budget the redesign, or match the model.



DISTRIBUTED DATABASES – COMPARISON MATRIX

	On-Premises Availability	Distribution Layer	Data Modeling	Converged Database	Triggers, Cursors and Stored Procedures within DB	Distributed Capabilities Since	CC (*) Mode (Default)	Popularity Ranking**
CockroachDB	Yes	Raft with Range Sharding (Hash-Sharded Indexes)	Limited Data and Partition Types	No	**No**	2015	Optimistic	**69**
YugabyteDB	Yes	Raft with Hash and Range Sharding	Limited Data and Partition Types	No	Yes	2016	Optimistic	**118**
Google Spanner	No	Raft with Range Sharding	Limited Data Types	No	**No**	2017	Pessimistic	**94**
Oracle Database	Yes	In-Memory Redo (Physical and Logical) Replication, Shard (*), and Raft NEW IN 23ai	Huge amount of Data and Partition Types	Yes	Yes	1986 with version 5 **Many more years of innovation than all other competitors together!**	Pessimistic& Optimistic	**1**

(*) **User-Defined**, **Range-Based**, **List-Based**, **and** **Composite Sharding**.

(*) Concurrence Control ** Reference: https://db-engines.com/en/ranking

MIRACLE

Partitions: did it crash, stall — or keep committing?

THE CONSENSUS ANSWER

Majority side keeps writing; minority refuses. Automatic and clean — but no quorum means **no writes at all** , and lease/clock edge cases are where the post-mortems live.

THE ARBITER ANSWER

Primary/standby + observer or witness decides decides who lives. Works — **if the arbiter sits in sits in a third failure domain** ; placed wrong, it wrong, it becomes the liar in chief.

THE STORAGE ANSWER

Array-level mirroring mostly defers the question question to the storage layer — transparent, and transparent, and blind to database semantics. semantics.

Split-brain is not a vendor defect — **it is the distributed-systems problem** . Architectures differ only in which failure mode you pre-select. Select it knowingly.

Where the logic lives is an availability decision

IN THE DATABASE

Procedures, triggers, constraints, queues — one round-trip, one consistency domain. Cost: engine coupling. **Several distributed SQL systems don't offer this layer** — a migration line-item.

IN THE APPLICATION

Services own logic; portable, testable. Cost: cross-service consistency is now yours — the problem SAGA patterns (sequences of local transactions + compensation) exist to manage.

CONVERGED VS SPECIALIZED

One engine for relational, JSON, graph, vectors, queues — or one store per workload. Fewer moving parts vs per-workload optimization. Both are legitimate.

No universally right answer — but a wrong one: pretending the choice wasn't made. **Every arrow between two data stores is a consistency contract someone must own.**

04

Designing for RPO/RTO — and choosing

RPO and RTO — business numbers, per scenario

RPO — RECOVERY POINT

How much data can you lose?

Seconds, minutes, hours — this decides sync vs async replication and backup cadence. RPO = 0 at distance always costs commit latency (see: physics).

RTO — RECOVERY TIME

How long until you serve again?

This decides failover automation — and client configuration, which is usually the forgotten half of the outage.

Set them per scenario

instance failure · site loss · region loss · cyber recovery

One global number **overspends on easy scenarios or lies about hard ones**

The business sets the numbers; engineering meets them or reports the gap. And **an unmeasured number is a wish** — only a drill makes it a fact.

High availability is not disaster recovery

LOSE 1 OF 3

Quorum holds. Writes continue. Nobody notices. **This is HA — and it works.**

LOSE 2 OF 3

The survivor refuses writes to protect consistency. Correct behavior — **and not a database anymore** . This is where HA ends.

QUORUM ACROSS REGIONS?

East + West majority: an East outage can strand West without quorum — **no writes anywhere** , from a one-region event.

Sound design in **every** family: quorum tight (3 zones, one metro) · non-voting replicas or async standbys in the far region · far-region failover an **explicit** decision, automatic or human.

Three families, honestly compared

FAMILY	GENUINELY STRONG	THE HONEST COSTS
Distributed SQL CockroachDB · Yugabyte · TiDB · Spanner	Elastic write scaling · automatic node failover · no external arbiter	Quorum-bound commit latency · DR must be designed around the quorum · optimistic-leaning contracts · thinner in-DB logic layer
Clustered primary/standby Oracle RAC + Data Guard · PG + Patroni	DR at any distance (incl. sync zero-loss where latency allows) · rich in-DB logic · multi-active via logical replication	Single write master per DB · scale-up more than write scale-out · operational depth to staff
Storage / managed array mirroring · Aurora-style tiers	Operationally simplest · provider handles the plumbing	Blind to DB semantics · staleness/failover on the provider’s terms · lock-in

Pick the column whose **costs** you can live with. Every column has them — a comparison that shows one column without costs is marketing.

Six questions before any engine gets named

1 · RPO/RTO per scenario from the business — instance, site, region, cyber	The requirements everything else serves
2 · Latency budget where are users; where can voters / sync copies live?	Physics prunes the options fast
3 · Concurrency profile hot rows? retry-tolerant code?	Optimistic vs pessimistic — match, or budget the redesign
4 · Logic in the database count procedures, triggers, queues before assuming they port	The quiet migration line-item
5 · Region-scale behavior whiteboard the partitions — who writes, who arbitrates?	Pre-select your failure mode knowingly
6 · Proof drills on a cadence, measured at the client	**An unrehearsed DR is a hypothesis**

What to carry out of the room

Physics is undefeated — RPO 0 at distance costs commit latency, in every architecture

Consensus is engineering, not magic — know the client-visible election numbers

Concurrency contracts are real — optimistic vs pessimistic decides who writes the retry logic

HA is not DR — quorum survival ≠ region survival; design both, explicitly

Prove, don’t assume — requirements from the business, drills measured at the client

---

Choose with requirements. Verify with rehearsals. That discipline is vendor-neutral — and it is the entire talk.

Choose with requirements. Verify with rehearsals.

Deeper reading: “The Fundamentals of Reliability in the Context of Distributed Databases”, parts 1–4. Questions welcome.

Francisco Munoz Alvarez · Miracle · linkedin.com/in/franciscomunozalvarez

MIRACLE

Together we make miracles

Thank You!

Francisco Munoz Alvarez
Global CTO & Chief Evangelist

francisco.munoz.alvarez@miracleltd.com
https://www.linkedin.com/in/franciscomunozalvarez
fcomunoz

MIRACLE