ARCHITECTURE GUIDE

Real-Time Decisioning for Telco

How to architect for millisecond-scale, ACID-grade decision authority across charging, policy control, mediation, network automation, and agentic AI use cases

An architecture reference for telco technology vendors, systems integrators, and CSP in-house engineering teams

Executive Summary

Telco vendors, the systems integrators who deploy on their behalf, and the CSPs who build their own platforms are all under pressure to make charging, policy, and network operations decisions faster, at greater scale, and increasingly with AI in the loop. Fragmented architectures struggle to do this correctly under sustained load. This guide sets out the architectural requirements for closing that gap, and where deterministic decisioning, machine learning, and agentic AI each belong.

The Core Problem

The gap in most telco stacks isn't in detection; it's in decision authority. Detection, state management, decision logic, and enforcement typically live in separate systems with different consistency models. Individually sound, together they produce architectures that move data quickly but decide too late, too inconsistently, or in the wrong place. The result: revenue leakage, reconciliation overhead treated as normal operating cost, and remediation that fires after the network condition it was meant to fix has already changed.

Critically, this is a problem that arrives with scale. A single national market with a modest subscriber base can often be served correctly from one central system. Large geographies and large subscriber bases cannot, and the distribution that follows is what turns consistency from a solved problem into an architectural one.

Independent research from STL Partners points the same way. STL's April 2026 analysis, which tracked 64 agentic AI demonstrations and announcements at Mobile World Congress 2026, found only two live deployments using sub-second data, both in-call fraud detection, with network operations agents largely confined to trials and proofs of concept. STL's August 2026 research program, based on interviews with 11 senior operator and vendor executives, then framed the cause as a structural data-architecture gap rather than a model-maturity problem: centralized data lakes remain right for enterprise analytics and long-term AI training, but were not designed for low-latency operational decisions, and the sampling and aggregation used to manage their cost strips out exactly the detail an agent needs to catch a genuine anomaly.

The Architectural Filter

Real-time decisioning architecture is only the right answer when three requirements apply simultaneously: continuous scale (sustained high-volume load, not occasional peaks), ACID-grade consistency (atomic, consistent, isolated, and durable decisions made against current state, never "eventually consistent"), and low latency (typically single-digit to tens of milliseconds, with no meaningful concept of "later"). If a system can batch, tolerate inconsistency, or rely on reconciliation, simpler tools remain the right choice.

Two Decisioning Patterns

- **On-line decisioning:** the subscriber’s transaction is waiting on a synchronous answer (charging, policy control, online signaling routing). The risk here is a consistency problem at scale: concurrent requests racing the same state across a distributed footprint.
- **Event-driven decisioning:** the event has already happened (offline mediation, network automation). The risk here is a staleness problem: state has moved on by the time a recommendation is evaluated.

The Three-Tier Decision Model

- **Tier 1, System Normal** (99%+ of decisions): deterministic logic against current state, answering in single-digit milliseconds end to end. No AI involved.
- **Tier 2, Known Abnormal:** trained ML models provide scores as input; deterministic logic still makes the final call, inside the same latency budget.
- **Tier 3, New Abnormal:** genuinely novel cases escalate out of the transaction path to an agent and then a human reviewer, with the agent querying live operational state rather than stale batch data. The agent recommends; a human decides; deterministic checks govern anything that executes.

Shorthand: the "DAD sandwich". Deterministic control at the entry point, Agentic interpretation in the middle for novel cases, Deterministic enforcement and recording at the exit.

Where This Shows Up in Telco Stacks

- **Mediation:** online signaling routing across multiple charging platforms (on-line), plus high-volume offline event consolidation for revenue assurance and billing (event-driven).
- **Converged online charging and policy control:** in-line allow, block, grant, and throttle decisions with the transaction waiting, under a fixed network SLA where a missed deadline commonly defaults to "allow" and silently shifts cost onto the operator (on-line).
- **Network automation:** closed-loop remediation before subscriber impact (event-driven).

Operational Requirements That Get Overlooked

A decisioning layer that is architecturally correct but operationally fragile still fails in production. Four requirements recur in vendor evaluations: active-everywhere geographic resilience (regional failure becomes capacity loss, not downtime), in-service upgrades that don't depend on routing traffic to a distant cluster, five-nines availability through redundant rather than hopeful recovery, and elastic scaling that holds latency and consistency guarantees under traffic spikes.

What This Isn't

This architecture is not a general-purpose streaming platform, an analytics system, or an AI product. It is not the right fit for systems that can tolerate batching, eventual consistency, or manual reconciliation without material cost. Honest scoping is intentional: it's what makes the rest of this guide usable as engineering input rather than a sales pitch.

The full guide (following this summary) covers each of the above in implementation detail, including a reference architecture diagram, application-area breakdowns, and buyer-language signals for qualifying whether a given system genuinely needs this architecture.

Table of Contents

1. Purpose & Audience

This guide is written for the engineering audience that designs and builds real-time telco systems: BSS/OSS vendors, mediation platform providers, charging and policy vendors, network automation and AIOps vendors, the systems integrators who deploy them, and the in-house engineering organizations at CSPs that build their own platforms rather than buying them. Several tier-1 operator groups fall firmly into that last category, and the architectural requirements are identical whichever side of the buy/build line a team sits on.

It is not a product pitch. Its purpose is to lay out the architectural requirements for real-time decisioning at telco scale, explain why fragmented architectures fail under those requirements, and describe the reference architecture, including where ML and agentic AI fit, that closes the gap. Section 12 addresses Volt specifically, and is clearly separated from the vendor-neutral material that precedes it.

> *Use this guide to evaluate your own architecture, to brief engineering teams on requirements before a build, or to assess a decisioning-layer vendor against a consistent set of criteria.*

The guide draws on three recurring telco application areas: mediation, converged online charging and policy control, and network automation. The architectural requirements are common across all three; the decisioning gap shows up differently in each, which is why Section 4 introduces a foundational split between on-line and event-driven decisioning before the application areas are covered individually.

2. The Core Problem: The Decisioning Gap

Modern telco architectures are built to handle scale. Online systems respond to transactions in milliseconds. Event-driven pipelines move usage records, alarms, and network signals at massive volume. The tooling is mature and the components, individually, do what they were designed to do.

And yet, charging systems still leak revenue. Fraud still clears before it's blocked. Network degradation still reaches subscribers before remediation fires. Reconciliation jobs still run nightly to clean up what the system got wrong during the day.

The tools aren't failing. The architecture has a structural gap, and it isn't in detection.

Detection Isn't the Problem. Decision Authority Is.

When a telco system fails to prevent revenue leakage, the instinct is to inspect the detection layer: was the signal too slow, was the model wrong? More often, the real question is what happens after detection: can the system translate what it knows into an authoritative decision fast enough to matter, against state that's actually current right now?

This is the decisioning gap: the architectural distance between knowing something and deciding something, and between deciding something and that decision becoming authoritative truth that downstream systems act on immediately.

In most telco stacks, this gap exists because detection, state management, decision logic, and downstream action live in separate systems, each with its own consistency model and its own view of "what's happening right now." Individually sound, together they produce a system that is fast at moving data and slow, or simply wrong, at deciding what to do with it.

Why the Gap Opens at Scale

It is worth being precise about when this becomes an architectural problem, because it is not universal and claiming otherwise invites justified skepticism from anyone who has run a well-behaved charging platform.

In a single small market, the problem is largely tractable. A compact geography keeps network round trips well inside the decision budget from anywhere to anywhere, and a subscriber base of a few million with complex hierarchical and shared balances can be held and served correctly from one central system. Ireland is a fair example: one location, tight SLAs, consistency maintained without much architectural effort.

Large markets break both of those assumptions at once. A geography the size of the United States or a pan-European group footprint cannot serve every request from a single location inside a millisecond-scale SLA, because the physics of the round trip alone consumes the budget. And a subscriber base in the tens or hundreds of millions, with shared data pools, family and enterprise account hierarchies, and multi-device accounts, will not fit in a single node. Distribution is therefore not a design preference; it is forced.

Once state is distributed both locally and geographically, concurrent access to the same shared balance from more than one place becomes routine rather than exceptional. That is the point at which consistency stops being free and architectures start trading correctness against latency. The decisioning gap is, in practice, what that trade looks like in production.

Independent Confirmation: The STL Partners Findings

This isn't a theoretical concern. Two pieces of STL Partners research from 2026 bear on it directly, and they are cited separately here because they establish different things.

The first, published in April 2026, tracked 64 agentic AI demonstrations and announcements at Mobile World Congress 2026. Only two live deployments used sub-second data, both customer-facing in-call fraud detection. Network operations agents were largely confined to trials and proofs of concept, and none of the live network-operations deployments depended on sub-second data.

The second, an August 2026 research program based on in-depth interviews with 11 senior individuals across telco operators and their vendor partners, examined AI readiness where real-time decision-making is critical to network operations. It frames the gap as a structural data-architecture problem, not a model-maturity problem. Centralized data lakes and warehouses remain the right approach for enterprise analytics, governance, and long-term AI training, but they were not designed for low-latency operational decisions. The sampling and aggregation telcos rely on to manage the cost of processing petabytes of daily network data strips out exactly the individual spikes, retries, and correlated failures an agent needs to distinguish a genuine anomaly from noise. Transporting complete event data to a central platform before reasoning can begin introduces latency that real-time decisions can't absorb.

The gap between what's live today and what autonomous operations require isn't a model problem waiting on the next generation of LLMs. It's an architecture problem: agents need thorough, low-latency live event data alongside trusted, governed context, delivered by an execution layer that was not designed primarily for batch analytics.

What the Gap Costs

- **Revenue leakage,** often attributed to billing complexity, but frequently caused by quota decisions that are not made atomically against current state, while concurrent requests modify what others are simultaneously reading.
- **Reconciliation overhead,** the most visible symptom. When systems cannot guarantee correct real-time decisions, they compensate by verifying outcomes after the fact. Reconciliation doesn't fix the gap; it's evidence the gap exists.
- **Remediation lag,** commonly treated as a pipeline performance problem, but more often a decision-authority problem: the recommendation is generated quickly, but conditions have changed by the time it's evaluated and recorded.

3. The Architectural Filter: When Real-Time Decisioning Applies

Not every telco system needs this architecture. The filter is a non-negotiable, simultaneous requirement for three properties. If any one of them doesn't genuinely apply, a simpler approach, batching, eventual consistency, standard streaming, is usually the right call.

Continuous Scale

The system processes tens of thousands to millions of events per second under sustained load, not just occasional peak spikes. It cannot pause to catch up.

ACID-Grade Consistency

Decisions must be ACID in the standard sense: atomic, consistent, isolated, and durable. In a decisioning layer they must also be made against state that is current rather than a delayed approximation of it, even under extreme load. Isolation is the property most often traded away quietly at scale, and it is the one Section 12 leans on when it cites strict serializability. "Fast but inconsistent" and "eventually consistent" are both unacceptable. This requirement becomes more critical, not less, when agentic AI is part of the decision path: an agent reasoning from stale state will reason correctly from incorrect premises, and its output cannot be allowed to control mission-critical outcomes directly.

Low Latency

There is no meaningful concept of "later." Decisions are typically needed in single-digit to tens of milliseconds. Raw speed alone isn't the gating factor; what matters is whether the SLA is met at required scale, with ACID guarantees, in the tail as well as the mean. A system with an excellent average and a bad p99.9 fails a fixed network deadline just as surely as a slow one.

All three must matter together. A system that only needs scale (but can reconcile later), or only needs speed (but can tolerate inconsistency), or only needs insight (analytics or AI alone) does not need a real-time decisioning layer. This architecture is only the right answer when correctness, scale, and latency are all hard constraints simultaneously.

4. Two Patterns: On-Line vs. Event-Driven Decisioning

Telco systems operate in two fundamentally different real-time patterns, and the decisioning gap manifests differently in each. Architects should identify which pattern a given system falls into before designing the fix; the requirements diverge in important ways. Note that several application areas contain both: mediation, in particular, spans the two.

	On-Line Decisioning	Event-Driven Decisioning
Trigger	The subscriber's transaction is waiting, pending a synchronous answer	The event has already happened; there's no transaction waiting
Examples	Charging, policy control, quota and balance checks, online signaling routing across multiple charging platforms	Offline mediation and event consolidation, network automation, remediation, fraud scoring pipelines
Client pattern	Bidirectional transactional clients requiring a millisecond synchronous response	Streaming pipelines processing continuous signal volume
Nature of the gap	A consistency problem at scale: once the subscriber base and geography force distribution, concurrent requests read and write the same shared state from more than one place at a time	A staleness problem: by the time a recommendation is evaluated, the state it was based on may have changed
Failure mode	Over-consumption (revenue leakage) or wrongful denial of a paying subscriber (worse commercially)	Remediation that conflicts with in-flight changes, or fires too late to prevent subscriber impact
What "late" means	Network failure handling is commonly configured to continue service when no decision arrives within SLA, so the operator silently absorbs the cost	The issue has either resolved itself or escalated into an outage by the time the decision lands

A Note on Geography and Geo-Redundancy

The consistency-at-scale point in the table above deserves expanding, because it is the single most common reason a well-designed platform stops behaving well.

Architectures that achieve geographic reach by partitioning subscribers across regional clusters introduce a specific failure: a subscriber who roams, or whose traffic is rerouted, may briefly sit outside their assigned region, creating a window where decisions are made on stale state or not made at all. Shared and

hierarchical balances make this worse, because the balance being decremented may be authoritative in one region while the request arrives in another.

The naive remedy, routing such requests back to the owning region, reintroduces the round trip that the regional split existed to avoid. Across a large national or group footprint that round trip can consume most of a single-digit-millisecond budget on its own, which means the fallback path breaks the SLA that the primary path was designed to meet.

What is actually required for on-line decisioning at this scale is not simply low latency, but low latency with ACID guarantees sustained across all sites simultaneously, without partitioning the subscriber base by region to achieve it. This is covered further in Section 9.1 (Cross Data Center Replication).

5. Reference Architecture

The Fragmented Architecture (What Most Stacks Look Like Today)

Events are ingested into streaming platforms, processed by stream processors, written to operational data stores, interpreted by rules engines or microservices, analyzed by analytics/ML systems, and finally acted upon by application code. Each layer is individually reasonable and often best-of-breed, but decisions emerge only after events pass through multiple hops, each with its own state, timing, and failure semantics.

This fragmentation creates predictable problems: latency becomes structural rather than incidental; state is duplicated and drifts; failure handling becomes ambiguous because no single layer has clear authority. Under load, these issues compound, typically requiring reconciliation jobs and compensating logic just to preserve acceptable behavior.

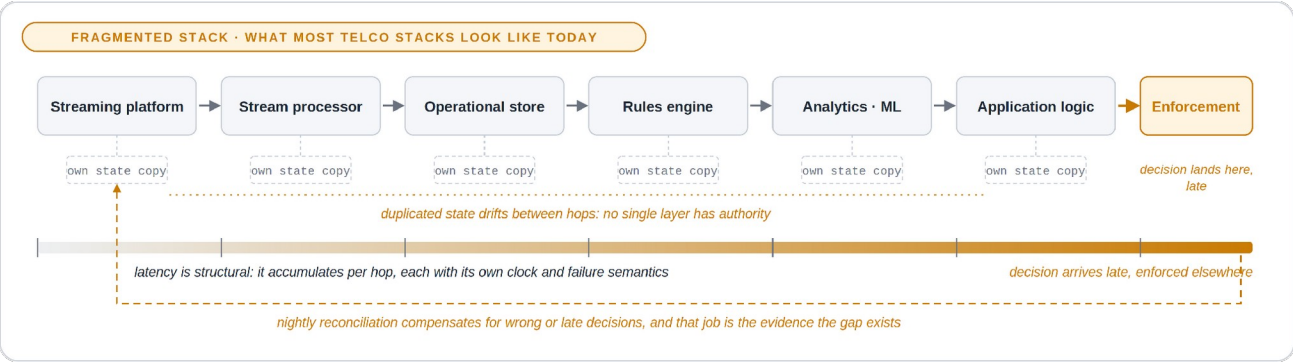

Figure 1: The fragmented architecture most telco stacks run today. Each hop holds its own copy of state and its own failure semantics; latency accumulates structurally, the decision is enforced late and elsewhere, and the nightly reconciliation job is the evidence that the gap exists.

The Consolidated Architecture (The Decisioning Layer)

The structural fix is architectural consolidation, not a new tool bolted onto the existing stack. State management, decision logic, and atomic recording of outcomes need to happen within a single execution and consistency model, not scattered across service boundaries with different guarantees.

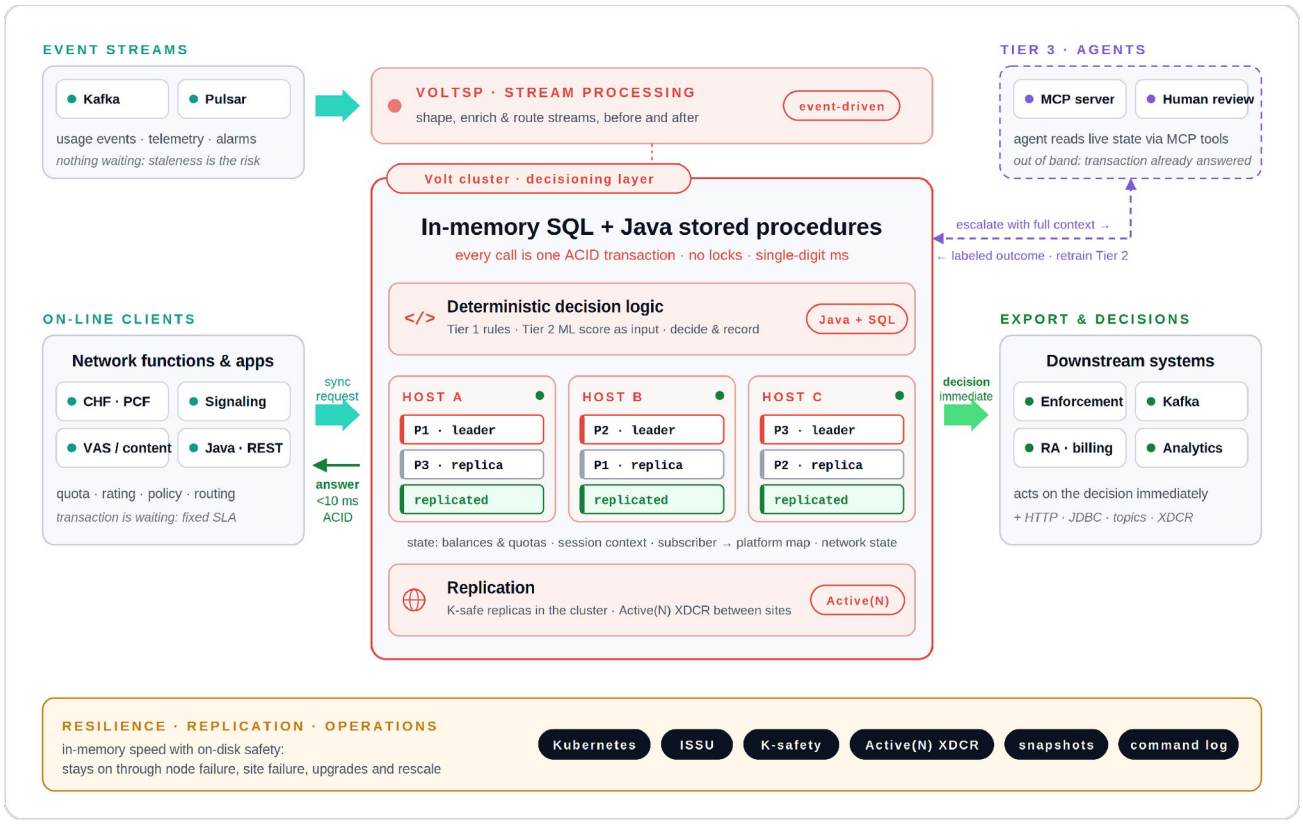

Figure 2: The consolidated decisioning layer. Both ingress paths resolve against one authoritative state: on-line clients call the cluster directly and receive an ACID answer in single-digit milliseconds, while event streams enter through stream processing. Tier 3 escalation runs out of band and feeds back into the scoring path. Leader and replica are per-partition roles; there is no cluster-level master (Section 9.3). K-safe replicas and Active(N) XDCR keep the layer on through failure, upgrade, and rescale.

Two Ingress Paths, One Authoritative State

A point often lost in reference diagrams: the decisioning layer must serve both patterns from Section 4, against the same state. These are not two systems, and they are not two copies of the data.

- **The on-line path** accepts synchronous requests directly from network functions and application clients, over the relevant transactional interfaces, and returns an authoritative answer inside a fixed SLA while the caller blocks. There is no queue and no stream in front of it; introducing one would break the pattern.
- **The event-driven path** consumes continuous signal volume from streaming platforms (Kafka, Pulsar, and similar), applies decision logic against the same state, and triggers downstream action or remediation.

Both paths read and write the same authoritative operational state within one consistency model. That is the whole point: an online quota grant and an event-driven usage update are not allowed to hold different views of the same balance. An architecture that satisfies only the event-driven path is a stream processing design, not a decisioning layer, and it will not support charging or policy control.

What the Decisioning Layer Does

- Accepts synchronous transactional requests and answers them authoritatively within a fixed SLA
- Ingests events continuously from streaming platforms
- Maintains the authoritative operational state required to make decisions, not a delayed approximation of it
- Performs real-time computation, detection, and deterministic decision logic against that state
- Records outcomes atomically as authoritative truth, immediately
- Exposes that operational state as trusted, reusable data products, including as MCP tools, backed by stored procedures, that agentic AI systems can query during reasoning

Downstream systems enforce these decisions immediately; the decisioning layer is the authority on what should happen next. Streaming and AI remain essential inputs, but neither is asked to do a job it wasn't designed for: streaming moves data, it doesn't decide; AI generates insight and recommendations, it doesn't guarantee correctness under load.

Design principle: the path from signal to authoritative decision should be short, explicit, and predictable. If it isn't, the system will be fast at moving data and slow, or inconsistent, at deciding what to do with it.

6. The Three-Tier Decision Framework

Not all real-time decisions are equal. A useful architectural model separates decisions into three tiers by complexity, and assigns each tier the right kind of intelligence, never more than is needed, never less than correctness requires.

Tier	Volume	Nature	What Handles It	Latency
1, System Normal	99%+	Routine, known patterns, no ambiguity	Deterministic logic against current state. No AI, no humans.	Single-digit ms end to end
2, Known Abnormal	Small %	Unusual but recognizable edge cases	Trained ML models provide scores as input; the decisioning layer applies deterministic logic to make the final call	Single-digit to tens of ms
3, New Abnormal	Very small %	Truly novel, conflicting signals, no existing rule covers it	Handled out of band: an agent queries operational state via MCP tools and assembles the case; a human decides; deterministic checks govern anything that executes; full chain recorded	Hundreds of ms to seconds (agent), seconds to minutes (human)

Tier 3 Happens Out of Band

Tier 3 cannot sit inside an on-line transaction, and it is important not to imply otherwise. A network function waiting on a charging or policy answer will time out long before an agent has assembled a case, and vastly before a human has looked at it. Human review is a seconds-to-minutes activity, not a millisecond one.

The workable pattern is therefore to decide deterministically now and escalate in parallel. The decisioning layer returns a provisional but authoritative answer inside the SLA, chosen to bound exposure rather than to be optimal: a reduced quota grant, a lower reauthorization threshold, a service allowed but flagged. The full case then goes to the agent and the human asynchronously, and whatever they conclude applies to subsequent decisions rather than retroactively to the one already returned. In event-driven flows the constraint is looser, because no transaction is blocked, but the same separation applies.

Why the Split Matters

Most decisions are routine and should never touch a model of any kind; they should execute deterministically in single-digit milliseconds using established rules. Reserving ML for genuinely ambiguous cases, and agentic AI plus human judgment for genuinely novel ones, keeps the system fast where it can be and intelligent where it must be. It also keeps cost proportionate: per-transaction inference economics that are trivial at Tier 3 volumes are prohibitive at Tier 1 volumes.

This also gives the architecture a feedback loop, and the loop is the most valuable part. Tier 3 escalations, once resolved by a human with agent assistance, produce labeled explainability chains: what was queried, what was recommended, what was decided, and why. That output is a labeled training example and often a candidate feature, which flows back into ML retraining and into the Tier 2 scoring path. "New abnormal" gradually becomes "known abnormal" (Tier 2), and eventually "system normal" (Tier 1). Escalation rates fall as the system accumulates experience.

Closing that loop is a data-architecture problem rather than a model problem. It requires the operational state, the decision record, and the escalation outcome to be capturable together, in a form that can be fed to training without a separate reconciliation exercise, and the resulting model deployed back into a path that still has to answer in single-digit milliseconds.

Shorthand worth adopting in design reviews: the "DAD sandwich": deterministic control at the entry point, agentic interpretation in the middle for novel cases, and deterministic enforcement and recording at the exit. AI recommends and interprets; the decisioning layer decides and enforces, every time.

7. Telco Application Areas

The three application areas below are where this architecture shows up most consistently in telco stacks today. Each is mapped to its decisioning pattern (Section 4) so the requirements are unambiguous before implementation begins.

7.1 Mediation: Three Different Jobs Under One Name

"Mediation" is an overloaded term covering at least three architecturally distinct jobs, and conflating them is the fastest route to designing the wrong system. It is worth stating plainly what mediation is not: any competent online charging product already maintains authoritative session context, rates in real time, and controls balance in the online path before the transaction completes. That is its job, and it does it. Mediation exists to handle what charging does not.

Online signaling routing (on-line pattern)

Operators rarely run a single charging platform. Low-cost brands, the flagship brand, enterprise and consumer segments, and platforms inherited through acquisition commonly coexist, each authoritative for a different slice of the subscriber base. The core network does not know which platform owns a given subscriber. Something has to sit in the online call flow, resolve the subscriber to the correct platform against an authoritative mapping, and route the request, all inside the same SLA the charging system itself has to meet.

This is on-line decisioning in the strictest sense: the transaction is waiting and the routing decision is on the critical path. It is also less forgiving than it looks, because a misroute is not a slow decision but a wrong one, resolved against the wrong balance on the wrong platform. Multi-brand and post-acquisition operator groups are where this pattern shows up most acutely.

Charging for services with no standard online interface (on-line pattern)

Value-added services, content and digital goods purchases, and partner-billed services often have no well-defined 3GPP online interface to the charging system. Mediation supplies the real-time balance authority, event construction, and policy application these flows require without a standard Gy or Nchf path to lean on, while still needing an answer while the purchase is in progress.

Offline event consolidation (event-driven pattern)

Mediation also collects, normalizes, correlates, and consolidates usage detail for downstream consumers: revenue assurance, billing, partner and interconnect settlement, and analytics. This is event-driven at very high volume, where completeness and correctness matter more than milliseconds, but it draws on the same subscriber and session state as the online paths above.

The architectural consequence is the important part. One application area contains both a hard-SLA synchronous path and a high-volume event path, operating on shared state. Split them across separate stacks and the reconciliation between them becomes a permanent operating cost, which is precisely the pattern described in Section 2. This is the clearest case in telco for a decisioning layer that serves both ingress paths from one consistency model, and the reason Section 5 insists on that shape.

7.2 Converged Online Charging and Policy Control

5G separates these into distinct network functions: the Policy Control Function governs bandwidth allocation, throttling, and QoS enforcement, while the Charging Function governs quota, rating, and balance. In 4G the equivalents were the PCRF and the OCS. That separation matters for interface design and for which vendor owns which box, but it does not change the decisioning requirement, which is materially

identical on both sides: a synchronous request from the network, a fixed SLA, an answer that must be correct against current authoritative state, and a configured fallback when no answer arrives in time. This guide therefore treats them as a single application area.

The transaction is waiting

Allow this session. Grant this quota. Apply this QoS profile. Block for exceeded allowance. There is no meaningful concept of "later," and no opportunity to reconcile before the subscriber has been served.

The failure mode is scale, not round trips

It is tempting to characterize the problem as charging systems making external calls for balance or policy mid-transaction and blowing their SLA. That is not how competent platforms behave; they hold the state they need in the decision path. The genuine exception is a deployment where the system of record sits outside the charging platform, occasionally in billing, but that is a legacy shape which was largely addressed in 4G and is designed out by 5G converged charging.

What competent platforms struggle with is the scale problem from Section 2: a subscriber base too large for one node and a geography too large to serve from one location inside the SLA. Distribution follows, and with it concurrent access to shared and hierarchical state, family and enterprise account structures, shared data pools, multi-device accounts, from more than one place at once. That is where correctness begins to cost latency, and where architectures start quietly trading one for the other.

A missed SLA converts directly into leakage

When no decision arrives in time, the configured failure-handling behavior on the network interface decides what happens next. Many operators deliberately choose to continue service rather than deny it, on the entirely rational grounds that wrongly blocking a paying subscriber is commercially worse than a period of unmetered usage. The consequence is that an SLA miss under load does not surface as an error or an alarm. It surfaces as revenue the operator has already agreed to absorb.

This is why tail latency, not average latency, is the number that matters here, and why a decisioning layer that cannot hold its SLA at peak is better described as expensive than as slow. It also means the network is handed a structural excuse to bypass the decisioning layer entirely under exactly the conditions where decisions matter most.

Complexity only ever increases

More concurrent services, more connected devices, more shared and hierarchical accounts, more mobility across regions. Every one of those raises decision complexity. None of them relaxes the latency requirement. For a published production example of exactly this problem, concurrent deductions from shared household and MVNO bundles at operator scale, see the i2i Systems case study referenced in Section 12.

7.3 Network Automation: From Detection to Closed-Loop Remediation

No transaction is waiting, but timing is still critical: the goal is to prevent degradation before subscribers notice or SLAs are breached. Monitoring, AIOps correlation, and ML predictions stream continuously, but without a decisioning layer, detection lives in one system, network and service-commitment state lives in

another, and remediation orchestration attempts to coordinate across domains. By the time the pieces align across potentially thousands of network elements, the issue has resolved itself or escalated into an outage.

Remediation recommendations, whether from ML models or from agentic AI systems, need deterministic evaluation against authoritative network state before execution: is this safe to execute now, does it conflict with in-flight changes, is it within operational bounds, and can execution be atomic with rollback? An action that was correct when it was recommended is not necessarily correct when it lands.

8. ML, Agentic AI, and Where Each Belongs

AI is increasingly present across all three application areas above: scoring fraud, interpreting anomalies, reasoning through network conditions, assisting escalations. Its value is real. But "AI" is too coarse a word to architect with, and treating it as one thing is how systems end up with a probabilistic component in an enforcement path.

Trained ML Models and LLM-Based Agents Have Different Properties

Trained ML models are repeatable. A fixed model artifact is a function: the same feature vector in produces the same score out, and it changes only when someone deliberately releases a new version. That is precisely why an ML score can sit in the in-line path at Tier 2. It is fast, it is reproducible, it is auditable, and, critically, it is calibratable, so the score can be thresholded against the commercial cost of a false decline versus a missed detection, and that threshold means the same thing next month as it does today.

One caveat worth stating precisely, because it is where audit trails usually fail: repeatability holds for a given feature vector, not for a given transaction. Where features include windowed aggregates, velocity, counts in the last hour, distance from the last event, scoring the same transaction at two different moments will legitimately produce different answers, because the inputs genuinely differ. Reproducing a historical decision therefore requires persisting the feature vector alongside the score and the outcome, not just the raw event.

LLM output is not reproducible in production. Generation samples from a probability distribution by default. Even with sampling effectively disabled, hosted inference does not guarantee identical output: execution varies with batching, and the served model can change without notice. More significant than either is calibration: the confidence an LLM attaches to a judgment is not a probability, it moves with context ordering, and it cannot be reliably tuned to a chosen operating point. That is a deeper obstacle to putting an LLM in an enforcement path than latency is, and latency alone already rules it out of a single-digit-millisecond budget.

The practical consequence is the tiering in Section 6. Deterministic logic and repeatable ML scores in the transaction path; language-model reasoning out of band, on cases where the system has already bounded its exposure.

Why Agents Reason Badly in Telco Systems

When an agent queries operational data to inform its reasoning, in most current architectures it is querying state from some point in the recent past, reasoning from yesterday's truth. Failures in production are frequently not model failures at all: the model reasoned correctly from incorrect premises, because the operational context it was given was not current. Sampling and aggregation compound this, stripping out precisely the individual spikes and correlated failures that distinguish a genuine anomaly from noise.

Fixing the model does not fix this. Fixing the data path does.

The Fix: An Operational Intelligence Layer

ML scores and agent recommendations are inputs to decision logic, not replacements for it.

- Agents query current operational state directly through MCP tools, each backed by a stored procedure that returns current balance, transaction history, active alerts, or congestion levels, instead of running queries against a stale warehouse.
- When an ML score is inconclusive, the decisioning layer escalates out of band: it packages the event, the score, and the full operational context, and passes it to the agent for interpretation, having already returned a bounded deterministic answer to the waiting transaction.
- The agent returns a structured recommendation. The decisioning layer evaluates it deterministically against current state: is it valid, does it conflict with actions already in progress, is it within operational bounds?
- If it passes, it becomes an authoritative decision, recorded immediately. If not, the case escalates further to a human reviewer with full causal context.

The agent reasons. The decisioning layer decides. This is the same tiered model from Section 6: agents operate at Tier 3, always evaluated by deterministic logic before anything executes.

Governance, Concretely

"Governance" in this context means a specific set of deterministic checks applied to every model or agent recommendation before action:

- Evaluate the recommendation against business policy and current system state.
- Enforce thresholds and boundaries (transaction amount caps, rate limits, credit policy).
- Validate context and timing: is this still the right action, are there conflicting actions in flight?
- Constrain the action space: an agent recommendation should select from a bounded set of permitted operations, not compose an arbitrary one.
- Treat subscriber-supplied and third-party text as untrusted input. Telco events carry free-text fields, merchant descriptors, payment references, subscriber-entered strings. Placing those directly into an agent prompt creates an injection surface that an adversary controls. A deterministic scoring path is indifferent to what the text says; a language model is not.

- Capture complete auditability: what was recommended, what was decided, why, and what the outcome was, together with the state the decision was made against.
- Provide fail-safe handling for invalid or out-of-bounds recommendations, including escalation paths.

Scoping distinction worth being precise about: a trained ML model in a fixed release is repeatable, and changes only when you deliberately ship a new one. A language model in production is not, and no architecture makes it so. What an architecture can guarantee is that every recommendation is evaluated by deterministic logic against current authoritative state before it affects production, and that the full chain is recorded and reproducible. Evaluate vendors on that. Treat "deterministic AI" as a claim to interrogate rather than a feature.

9. Resilience & Operational Requirements

Telco-grade deployments carry operational requirements beyond the core decisioning logic itself. A decisioning layer that is architecturally correct but operationally fragile will still fail CSPs in production. Four requirements recur across vendor evaluations.

9.1 Cross Data Center Replication (Active-Everywhere)

Regional failure must become capacity loss, not downtime, and every region should process production traffic simultaneously rather than following a primary/replica model. This directly closes the geography gap described in Section 4: a subscriber who roams or whose traffic reroutes should never fall into a region-shaped blind spot where no authoritative decision is possible, and no request should have to make a long round trip to its home region to get an answer. Deterministic multi-region consistency ensures that whether two regions or ten process conflicting updates, all sites converge to the same final state.

9.2 In-Service Software Upgrade

The historical approach to upgrades, taking a cluster out of service and routing its traffic to a remote cluster for the duration, no longer holds up at 5G response SLAs across large geographies. If the remote cluster is far enough away that the round trip alone consumes the decision budget, the fallback breaks the SLA it was meant to protect. In a configuration that continues service when no decision arrives, that does not show up as an outage; it shows up as leakage, for the duration of the maintenance window.

Upgrades therefore have to happen in service: nodes upgraded individually while the cluster continues processing production traffic, with new decision logic activating only once every node has completed, so the cluster never applies two versions of the rules at the same time.

With K-safety redundancy this is a routine operation rather than a risk event. Taking a node out of the cluster leaves the partitions it hosted served by their surviving replicas, at one level lower redundancy than normal running. The node is upgraded, rejoins, and resynchronizes, restoring full redundancy before the next node is taken. Traffic is never interrupted, correctness is never compromised, and decision latency is unaffected, because the replicas now answering were already serving. Combined with active-everywhere replication across two or three sites (Section 9.1), even a site-level maintenance event becomes a capacity

reduction rather than an outage. Zero-downtime operation is an achievable property of this architecture, not an aspiration; it asks only that the cluster be operated with its designed redundancy intact.

9.3 High Availability (Five Nines)

Infrastructure failure at telco scale is constant, not exceptional. Availability should come from redundant, decentralized operation: every partition has a leader and K replicas, and there is no cluster-level master node or coordinator. Fast failure detection and automatic failover, rather than hopeful recovery procedures.

9.4 Elastic Scalability

Decisioning systems can't queue or batch when overwhelmed the way analytics workloads can. Clusters should scale up and down without service disruption, and decision latency and consistency guarantees should hold whether the cluster is running on three nodes or thirty. Adding capacity means redistributing partitions onto the new nodes, and that redistribution has to happen online, with traffic continuing and the SLA held throughout, rather than inside a maintenance window the network can no longer afford.

When evaluating a vendor or designing in-house: ask what happens to decision latency and consistency during a regional failover, a rolling upgrade, and a traffic spike, and ask for the tail, not the average. Steady-state numbers are the easy case.

10. What This Architecture Is Not

Clarity here prevents over-scoping. A real-time decisioning layer, correctly deployed, is deliberately narrow in what it claims to be.

- **Not a general-purpose streaming platform.** It doesn't replace Kafka, Pulsar, or Flink. Streaming moves events; the decisioning layer decides what to do with them.
- **Not an analytics or reporting system.** It isn't built for offline analysis, historical reporting, or exploratory queries.
- **Not an AI system.** It does not claim to "decide intelligently" on its own. Models and agents provide scores and recommendations; the decisioning layer applies deterministic authority.
- **Not a charging or policy product.** It is the layer such products are built on, not a replacement for the rating, session, and product-catalog logic that makes one a charging system.
- **Not a rules engine bolted onto an existing stack downstream.** It replaces fragmented decision paths rather than sitting after them.
- **Not the right fit for systems that can tolerate batching, eventual consistency, or manual reconciliation without material cost.** Simpler tools remain the better choice there (see Section 3).

11. Buyer & Market Signals

Teams evaluating whether a system genuinely needs this architecture, or listening for it in a customer conversation, can watch for language like:

- “We can't explain why a decision was made.”
- “Our agents are confidently wrong because they can’t see live state.”
- “It works in one market but not across the group footprint.”
- “Charging behavior breaks down at scale.”
- “We rely too heavily on reconciliation.”
- “Why does behavior change under load?”
- “We can’t take a maintenance window any more.”
- “We can see problems coming but can’t prevent them.”

Practical use: these signals work in both directions. They are useful for recognizing the pattern in your own systems (tie back to the filter in Section 3), and equally useful when qualifying a decisioning-layer vendor; ask them directly how their architecture addresses each signal.

12. How Volt Implements This Architecture

Everything above describes the architecture on its own terms, independent of any single vendor. This section covers how Volt Active Data implements it, for readers evaluating Volt against the requirements laid out in Sections 1 through 11.

Mapping Volt to the Reference Architecture

- **The decisioning layer (Section 5).** Volt maintains authoritative operational state, executes deterministic decision logic, and records outcomes atomically as authoritative truth, all within a single execution and consistency model. Both ingress paths are served from that one state: synchronous transactional clients on the on-line path, and continuous streaming ingest, shaped and enriched by Volt stream processing (VoltSP), on the event-driven path.
- **The three-tier framework (Section 6).** Tier 1 decisions execute deterministically in single-digit milliseconds end to end. Tier 2 incorporates ML model scores as input inside the same budget. Tier 3 exposes operational state to agentic AI systems as MCP tools backed by stored procedures, so agents reason from live, authoritative data rather than stale batch snapshots, out of band from the waiting transaction.
- **Both decisioning patterns (Section 4).** For on-line decisioning, Volt returns ACID-compliant decisions in-line with the transaction inside a single-digit-millisecond budget. For event-driven decisioning, it consumes streaming signals continuously and executes operational control or remediation logic against current state.
- **Scale in both dimensions (Section 2).** Cross Data Center Replication is active-everywhere (Volt's Active(N) XDCR) rather than primary/replica, so a large subscriber base can be distributed and a large

geography served locally without partitioning subscribers into region-shaped blind spots or paying a home-region round trip.

- **Resilience (Section 9).** In-Service Software Upgrade proceeds node by node with the cluster in service: K-safety means the partitions on the node being upgraded continue to be served from their replicas, with no interruption and no effect on decision latency. K-safety redundancy and automatic failover provide high availability without a master or coordinator, and Cross Data Center Replication across two or more sites makes site-level maintenance and regional failure a capacity reduction rather than an outage. Clusters scale elastically online, redistributing partitions with traffic continuing.

In Production

i2i Systems built its Frontiers-CBS online charging system on Volt as the decisioning core, replacing Oracle TimesTen. Charging decisions that previously took seconds complete in under 10 milliseconds at 99.999% availability, and concurrent deductions from shared household and MVNO bundles are serialized per partition, removing race conditions and overspend risk. The deployment is triple-active across sites for a Turkish operator with around 25 million subscribers, which is the shared-balance-at-scale problem from Section 7.2 in production.

Mahindra Comviva's MobiLytix real-time marketing platform runs on Volt to act on recharge, usage, and customer-interaction events inside the short window in which they still matter: an event-driven deployment of the same layer.

A global tier 1 operator runs Volt for live sub-second mediation, routing online signaling across multiple charging platforms (Section 7.1).

What Backs the Latency Claim

Volt's published online-charging benchmark, whose benchmark application is public on GitHub (VoltDB/app-telco-charging), ran the charging workload on Google Cloud at cluster sizes from 3 to 27 nodes. Each operation allocates quota against a subscriber balance across multiple tables with full ACID guarantees. Throughput scaled near-linearly with node count to over 3.2 million operations per second at 27 nodes, with 99th-percentile latency under five milliseconds throughout and per-node throughput essentially constant across cluster sizes. A newer benchmark report on the Volt resource library, Why TPS Benchmarks Lie and What Real VoltDB Performance Looks Like, complements this with results at the configuration a production deployment would actually run. The mediation workload, with deduplication, enrichment, and aggregation against a subscriber table of over five million rows, sustained around 240,000 transactions per second at K=1 with command logging enabled, across 48 vCPUs including the replicas, with 99th-percentile latency under five milliseconds. The charging workload held around 3,500 transactions per second per vCPU at K=1 with replica cores counted, again with p99 under five milliseconds.

Volt's standing performance position for production deployments is an average latency of one to two milliseconds and a 99th percentile under ten milliseconds, with strict serializability, the highest isolation level, on every transaction. That is the basis for the single-digit-millisecond figures used in this guide.

Two things are worth being explicit about, because a careful reader will ask. First, K-safety replication is synchronous within a site. It has a cost, roughly half the per-core throughput at K=1 as the report quantifies,

and a small addition to every write. What it does not do is move the tail out of budget: the published p99 at K=1 is still under five milliseconds. It changes how many nodes you provision, not whether the SLA holds. Second, replication between sites over XDCR is asynchronous and conflict-resolved, so every site answers its own traffic at local latency and the SLA does not depend on the distance to another data center. The latency claim holds at each site of an active-everywhere deployment, which is the configuration that matters for the geography argument in Section 2.

13. Next Steps

If the architecture in this guide matches a system you're currently building or evaluating, there are a few practical ways to go deeper:

- Start on the Developer Hub at developers.voltactivedata.com: the interactive architecture explorer, the getting-started paths, and the sandbox for evaluating your own architecture against the requirements in Section 3.
- Read the benchmark: Why TPS Benchmarks Lie and What Real VoltDB Performance Looks Like: Volt's performance results for mediation and charging workloads at the realistic high-availability configuration, on the Volt resource library.
- Watch for the next asset in this series: a decision framework for choosing between architectural approaches based on your specific latency, scale, and consistency requirements.

Appendix: Glossary

Term	Definition
Decisioning layer	The architectural component that maintains authoritative state, applies deterministic decision logic, and records outcomes as authoritative truth immediately.
Decision authority	The system or role responsible for making a decision final and enforceable, as distinct from the system that merely detects or recommends.
On-line decisioning	A pattern where a transaction is held pending a synchronous decision (e.g. charging, policy control, online signaling routing).
Event-driven decisioning	A pattern where the triggering event has already occurred and the decision governs what happens next (e.g. offline mediation, remediation).
CHF / PCF	5G Charging Function and Policy Control Function: the separated network functions governing quota and rating, and bandwidth and QoS, respectively. The 4G equivalents were the OCS and the PCRF.
CCS	Converged Charging System: the 5G charging architecture in which the charging function holds authoritative balance and rating state rather than depending on an external system of record.
Mediation	An overloaded term covering online signaling routing across multiple charging

Term	Definition
	platforms, charging for services with no standard online interface, and offline event consolidation for revenue assurance and billing. See Section 7.1.
Fail-open / continue	Configured network failure-handling behavior in which service continues when no charging or policy decision arrives within SLA, so a missed deadline becomes absorbed cost rather than a denied subscriber.
Tier 1 / 2 / 3	The three-tier decision-complexity model: routine and deterministic, ML-assisted and ambiguous, and agentic plus human-in-the-loop for novel cases handled out of band.
DAD sandwich	Shorthand for Deterministic to Agentic to Deterministic: AI recommends and interprets in the middle, but deterministic control governs entry and exit.
MCP tools	Model Context Protocol tools: the standard interface through which agentic AI systems call functions exposed by other systems. In a decisioning layer, each tool is backed by a stored procedure, so an agent reads authoritative operational state through the same transactional path as the application, not a copy of it.
Repeatable (of a model)	A fixed model artifact returns the same output for the same input vector, changing only on deliberate release. True of trained ML models; not true of hosted LLM generation. Distinct from returning the same answer for the same transaction, since features may include time-windowed aggregates.
XDCR	Cross Data Center Replication; an active-everywhere architecture where every region processes production traffic and decision authority concurrently. Volt's implementation is Active(N).
ISSU	In-Service Software Upgrade; upgrading cluster nodes individually with the cluster in service.