Real-Time Decision-Making in Distributed Systems

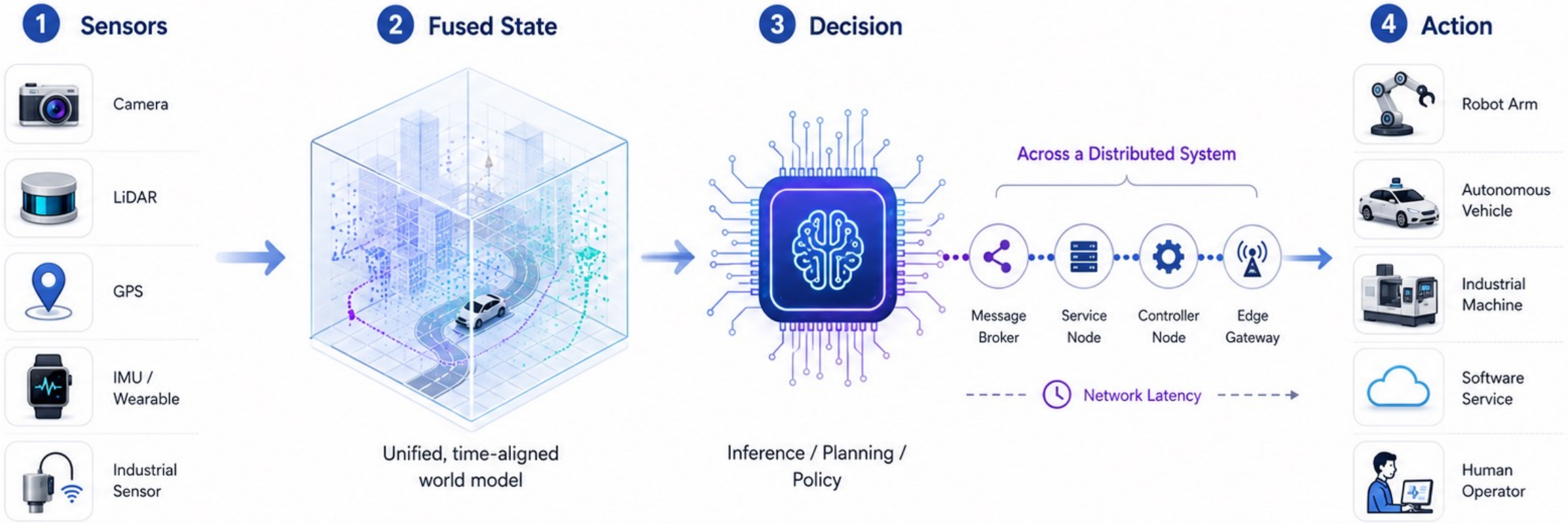
From sensor fusion to reliable actions under latency, uncertainty, and failure



Dr. Oleksandr Muliarevych
Associate Professor,
Lviv Polytechnic National University

Summer School on Sensor Fusion – SUT, Gliwice, Poland

1. From Sensor Fusion to Action



? What happens after the data is fused?

Sensor fusion **estimates** the world. Distributed systems must **act on it in time.**

2. The Core Idea

Sensor fusion creates a state estimate. A distributed decision system must transform it into a timely, safe, and reliable action.

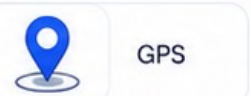
Nervous System Analogy



1 Sensors





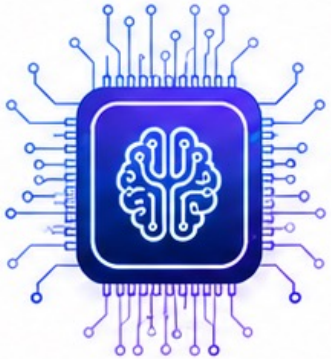


2 Fusion / State Estimate

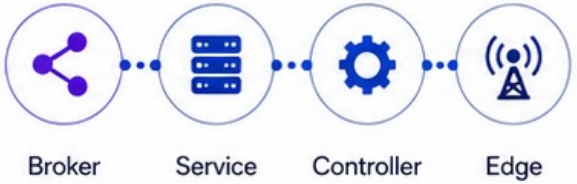


Unified world model

3 Decision Policy



4 Distributed Delivery



Latency / messaging / failures

5 Actuator / Action





A state estimate must become a **timely**, **safe**, and **reliable** action.

Where errors can appear



Sensor noise



Time mismatch



Model uncertainty



Network delay



Duplicate message



Processor restart



Stale command

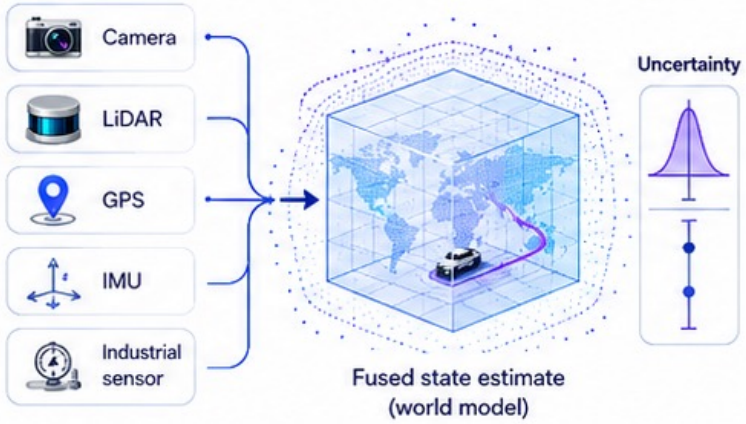
3. The Core Idea

A real-time decision = a sufficiently correct action delivered before its deadline.

We act under uncertainty. The goal is enough reliable information to choose a safe and useful action before the opportunity disappears.

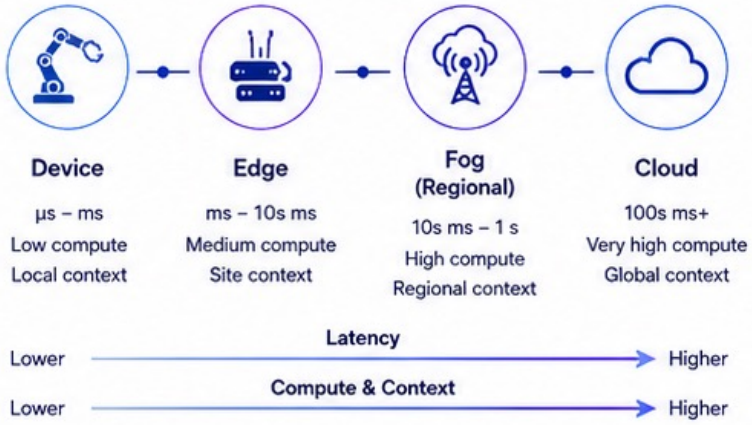
Three major tasks

1 Fuse data + preserve uncertainty



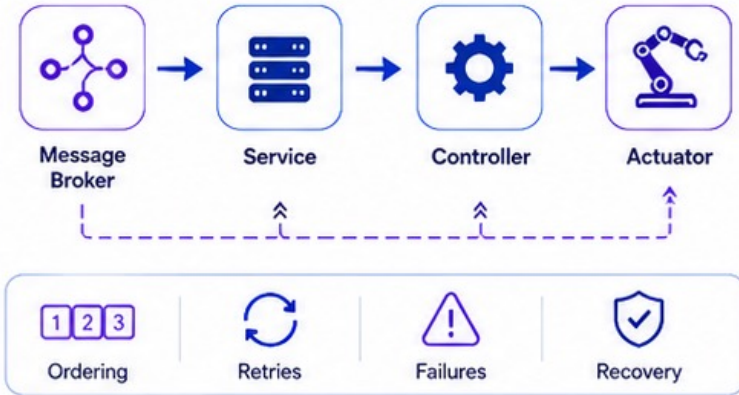
Fuse heterogeneous data without hiding uncertainty.

2 Place the decision at the right level



Choose the right location for the decision.

3 Engineer correctness in distribution



Ordering, retries, failures and recovery must not create wrong actions.

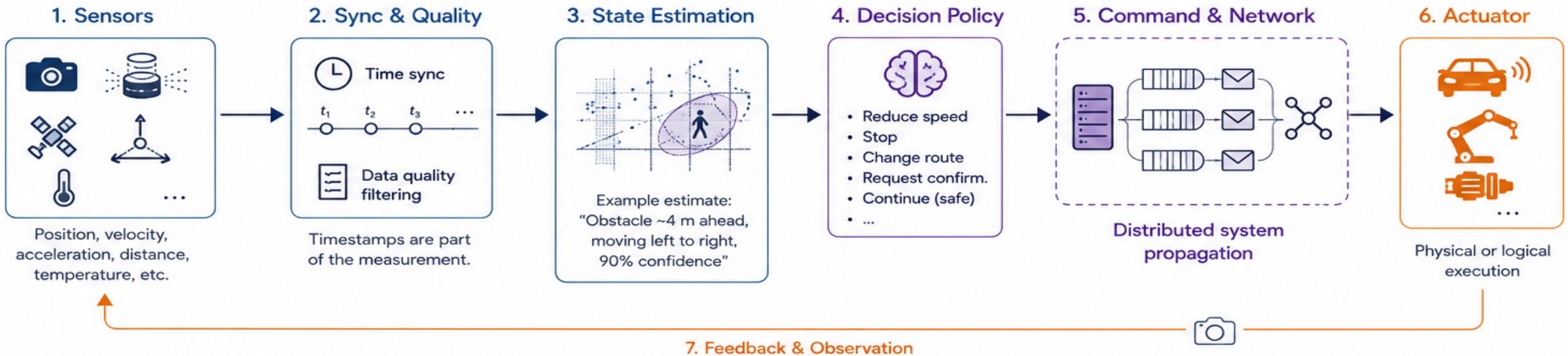
Edge-intelligence perspective

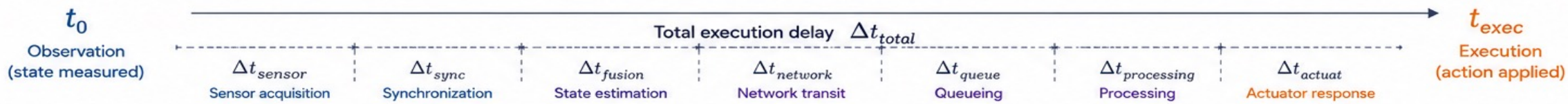
Fusion enables communication, computation, control and collaboration close to the data source.

Communication | Computation | Control | Collaboration

4. Observation vs. Execution: The State Staleness Problem

The state you act on was measured earlier.





Staleness Reality

$$X(t_0) \neq X(t_0 + \Delta t_{total})$$

The physical world changes while we compute.

A decision optimal at t_0 may be hazardous at $t_{exec} = t_0 + \Delta t_{total}$.

Latency Composition

$$\Delta t_{total} = \Delta t_{sensor} + \Delta t_{sync} + \Delta t_{fusion} + \Delta t_{network} + \Delta t_{queue} + \Delta t_{processing} + \Delta t_{actuat}$$

Engineering goal: Minimize Δt_{total} while preserving correctness, safety, and reliability.

Engineering Principle

A real-time control action must be valid for the state of the physical world at the exact moment of execution, not the state that existed when the computation cycle began.

A continuous feedback loop is non-negotiable. Verify the physical state transformation.

5. Real-time: Deadlines, Latency and Reliability

Real-time means meeting deadlines — consistently.

Hard Real-time

Missing a deadline = system failure.

Example: emergency stop after collision.



Firm Real-time

Late result has little or no value.

Example: route correction after the junction.



Soft Real-time

Late result still has value, but its value decreases.

Example: monitoring dashboard.

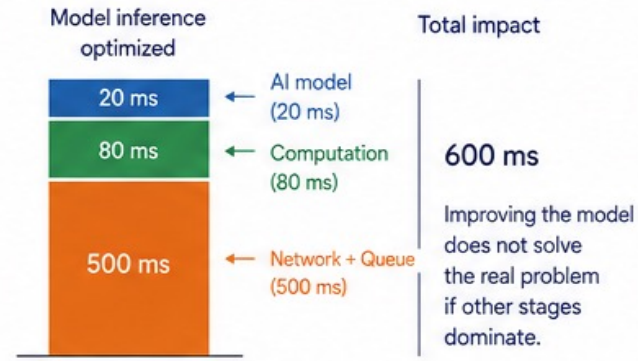


End-to-end Event-to-Action Latency

T_{acq} Acquire sensor data + T_{sync} Synchronize / resample + T_{fusion} State estimation + T_{net} Network transmission + T_{queue} Queueing / event stream + T_{state} Access state + T_{decide} Decision calculation + $T_{actuate}$ Apply action = T_{e2a} Event-to-action latency

Event observed (t_0) → Action applied (t_{exec})

Why End-to-End Matters



Deadline-Based Performance

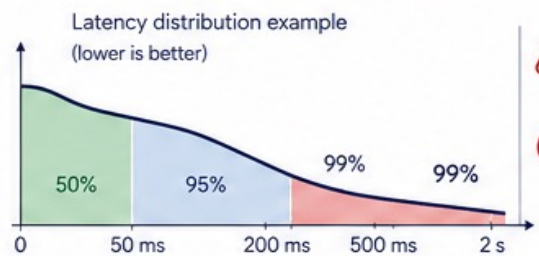
$$P(T_{e2a} \leq D) \geq R_{min}$$

D — deadline (e.g., 200 ms)

R_{min} — minimum required probability

Example: $P(T_{e2a} \leq 200\ ms) \geq 99.9\%$

Why Averages Are Dangerous

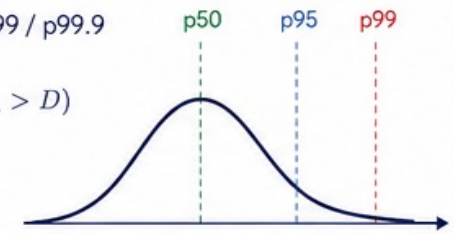


⚠ Average: 50 ms (looks excellent)

⊗ But 1% of operations take > 2 s (unsafe)

What We Must Measure

- ✓ Percentiles: p50 / p95 / p99 / p99.9
- ✓ Deadline miss rate: $P(T_{e2a} > D)$
- ✓ Not just averages



Key Takeaway: A real-time system is only as fast as its slowest stages — and only as good as its worst-case behavior.

6. Freshness-aware Fusion

Recent, reliable and low-noise observations should influence the estimate more than stale data.

Simple freshness-aware fusion

1 Observation age

$$\mathrm{age}_i(t) = t - \mathrm{timestamp}_i$$

2 Freshness-aware weight

$$w_i(t) = \frac{c_i}{\sigma_i^2} \exp\left(-\lambda \cdot \mathrm{age}_i(t)\right)$$

3 Fused state estimate

$$\hat{x}(t) = \frac{\sum_i w_i(t) z_i}{\sum_i w_i(t)}$$

Camera 10 Hz

IMU 100 Hz

GPS 1 Hz

Higher confidence, lower variance, and lower age => higher influence.

State estimation with traceability

1 Kalman-style update

$$x_{k|k} = x_{k|k-1} + K_k\left(z_k - H x_{k|k-1}\right)$$

2 Decision context vector

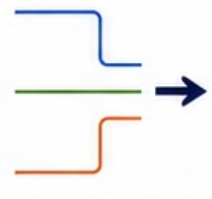
$$C = \{\hat{x}, \mathbf{P}, \Delta t_{\mathrm{age}}, \mathbf{Q}_{\mathrm{status}}\}$$

estimate | confidence | data age | source quality

Uncertainty metadata is part of the state, not optional diagnostics.

Example

Sensor A	10.0 m	(recent, reliable)
Sensor B	10.2 m	(recent, slightly noisier)
Sensor C	8.0 m	(2 s old)



Freshness-aware estimate ≈ 10.1 m

State estimates for decision-making should include uncertainty and traceability.

The decision service should know not only the value, but also confidence, age, and source quality.

7. Decision-making under Uncertainty and Deadlines

Choose the best action while balancing benefit, timeliness, risk, and coordination.

Decision as an Optimization Problem

$$a^* = \arg\max_{a \in \mathcal{A}} \left(\mathbb{E}\left[U(a, \hat{x})\right] - C_{\text{late}}\left(T_{\text{e2a}}, D\right) - C_{\text{risk}}\left(a, \gamma\right) - C_{\text{coord}}(a) \right)$$

1) Expected Utility

$\mathbb{E}\left[U(a, \hat{x})\right]$

Benefit / reward of action a in estimated state $\hat{x}$.

$\mathbb{E}$	Expected value (average over uncertainty)
U	Utility function (value, benefit)
a	Action being evaluated
$\hat{x}$	Estimated state of the world (from fusion layer)

2) Late Cost

$C_{\text{late}}\left(T_{\text{e2a}}, D\right)$

Penalty for delaying execution or missing a deadline.

C_{late}	Cost of latency
T_{e2a}	End-to-actuation time (observation → action)
D	Deadline (time threshold)

3) Risk Cost

$C_{\text{risk}}\left(a, \gamma\right)$

Penalty for potential unsafe or undesirable outcomes.

C_{risk}	Cost of physical risk
a	Action under evaluation
γ	Uncertainty / confidence level (reliability of $\hat{x}$)

4) Coordination Cost

$C_{\text{coord}}\left(a\right)$

Penalty for conflicts or negative impact on other agents/systems.

C_{coord}	Cost of coordination
a	Action under evaluation
—	Captures congestion, resource contention, and interference

What This Means

We do not maximize utility alone. We trade it off against latency, risk, and coordination.

The policy can be: rules, optimization, MPC, planning, RL policy, or a hybrid.

The core scientific question: how to balance utility, latency, uncertainty, safety, and coordination.

Goal:
Select the action with the highest expected value given deadlines, uncertainty, and system-wide impact.

Controlled Degradation: Move to Safer Modes

If global information is unavailable → use local avoidance, stop accepting complex routes.

If sensor confidence < threshold → reduce speed.

If AI recommendation is uncertain and time permits → request human confirmation.

If deadline is too tight for human involvement → execute autonomous fallback policy.

Safe mode does not always mean "stop".

It means choose behavior whose risk is acceptable under reduced information quality.

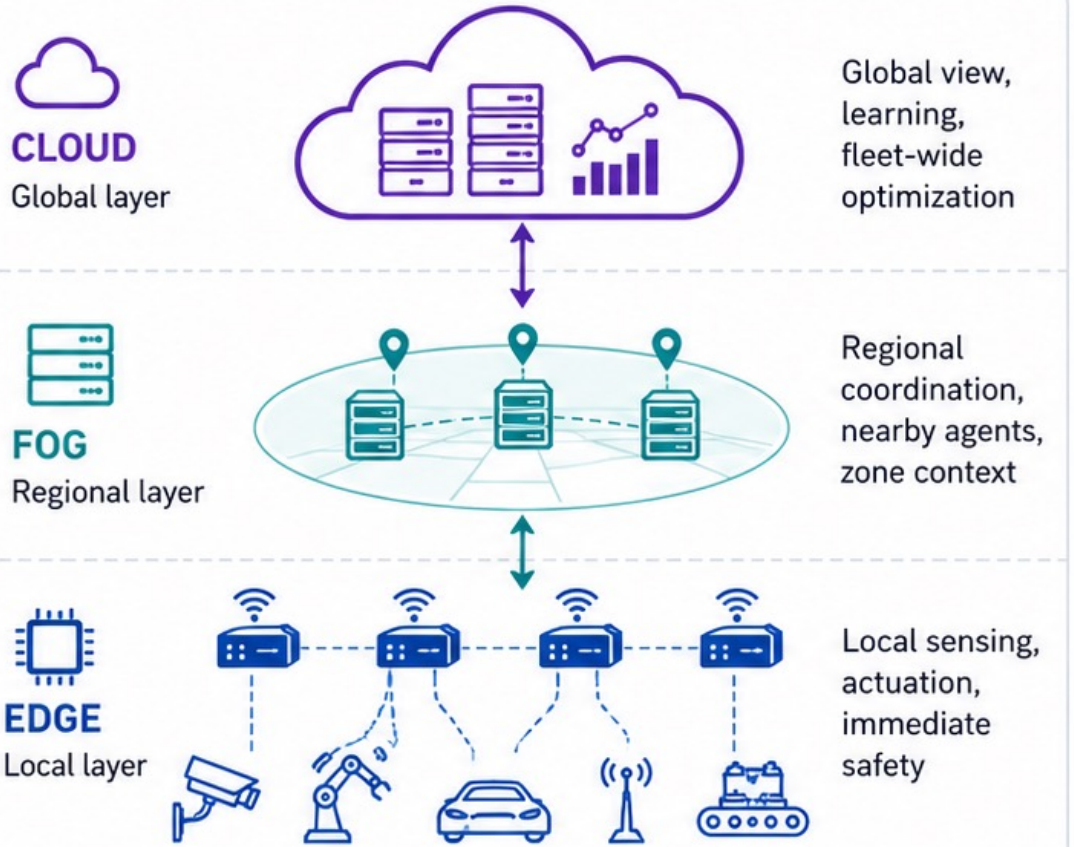
Safety Policy (Example)

$$\text{If } \left(\gamma < \theta\right) \wedge \left(R(a, \hat{x}) > \rho\right) \implies a \leftarrow a_{\text{safe}}$$

γ	→	state confidence / quality
θ	→	minimum confidence threshold
$R(a, \hat{x})$	→	estimated risk of action a in state $\hat{x}$
ρ	→	maximum acceptable risk threshold
a_{safe}	→	safe mode action (e.g., reduce speed, stop, yield)

8. Architectural Placement of Decision Authority

Where should the decision happen?



Edge
- milliseconds
- local safety
- privacy

Fog
- regional coordination
- nearby agents
- zone context

Cloud
- global policy
- learning
- fleet-wide optimization

Decision placement as optimization

$$\min_{\ell \in \mathcal{L}} J(\ell) = \alpha T_\ell + \beta E_\ell + \gamma C_\ell + \delta R_\ell$$

Subject to: latency < D; privacy & realiability

T = latency E = energy C = cost R = risk

Principle: Move the decision to the lowest layer that has enough information and authority to act safely.

Edge handles immediate safety;
Fog coordinates locally;
Cloud optimizes globally.

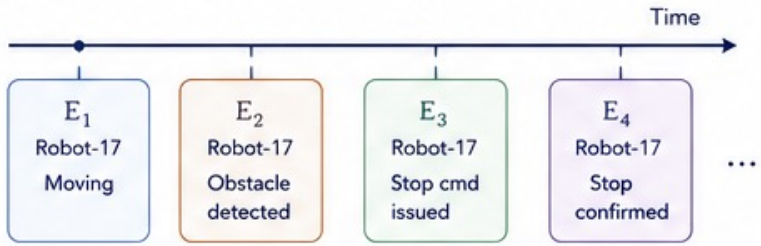
9. Event Streams: Ordered Facts, Stateful Decisions

Implement the architecture with event-stream processing.

1. Events: Facts About What Happened

- A sensor produced a measurement.
- A vehicle changed position.
- A model produced a new estimate.
- A battery level crossed a threshold.
- A route was updated.
- A command was accepted.
- An actuator completed an operation.
- ...

2. Event Log: Ordered, Append-Only



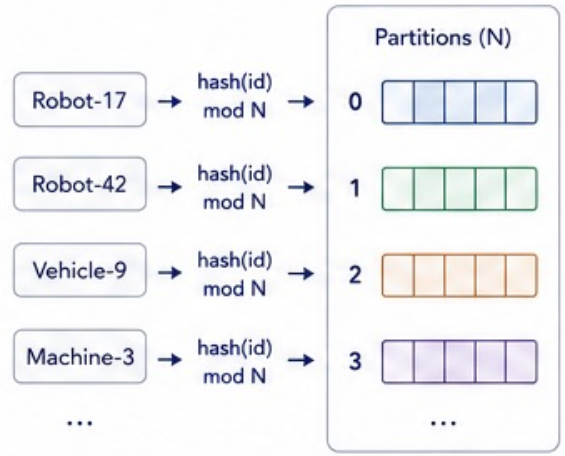
- Append-only log of immutable events.
- We preserve order per entity, not a single global order.
- All events for the same entity go to the same partition.

Partitioning

$$\text{partition} = \text{hash}(\text{entityId}) \bmod N$$

N = number of partitions

3. Partitioning by Entity



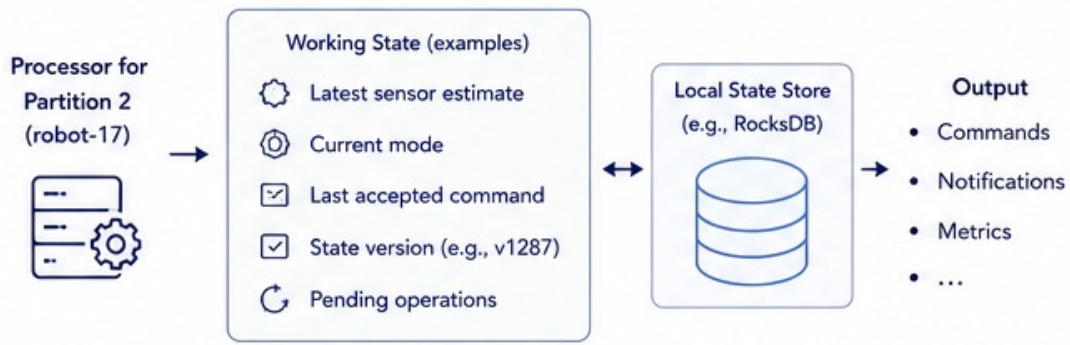
Each partition is processed by one logical processor, which sees that entity's events in order.

Example (One Robot)

1. Robot moving
2. Obstacle detected
3. Stop command issued
4. Stop confirmed

Correct order matters.

4. Stateful Processor



Local state = speed. No remote DB call for every event.

5. Operational Considerations

Process restarts:
How long does state restoration take?

Rebalancing:
What happens during movement?

State size:
How large can it grow?

Replayability:
Can the state be reconstructed from the event log?

6. Determinism, Replay, Explainability

If we have the same ordered input events, the same initial state, and the same decision logic → we should reproduce the same state transitions and explain why a decision was made.

This requires:

- Replayable, immutable event log
- Deterministic (or controlled) decision logic
- Versioned state and logic
- Complete metadata (timestamps, sources, ids)

10. Correctness Patterns in Distributed Decisions

Engineering patterns that make distributed decisions correct, reliable, and explainable.

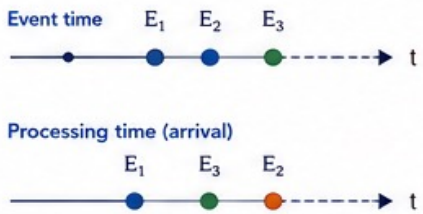
1 Ordering

Keep events for the same entity in order.

Partition by stable key (entity)

$$\text{partition} = \text{hash}(\text{entityId}) \bmod N$$

Use event-time with watermarks for delayed data.



Watermark
Wait up to W for late events; close windows after watermark passes.

2 Idempotency

Repeatable operations do not change final state.

Examples

Idempotent
- Set mode = SAFE
- Set target speed = 30 km/h

Not idempotent
- Decrease speed by 10% (applied twice → 20%)

Design commands as "set" or use idempotency keys.

3 Versioned State (Optimistic Concurrency)

Every state transition carries the expected version.

Current state: version = 15

↓

Command computed from version 15

↓

Accepted → apply → version = 16

↓

Late command from version 14
Reject or re-evaluate

Prevents stale updates from overwriting newer state.

4 Backpressure

When input > processing capacity, apply controlled response.

Without backpressure



Queues grow → memory exhausted → latency unbounded → system fails.

With backpressure

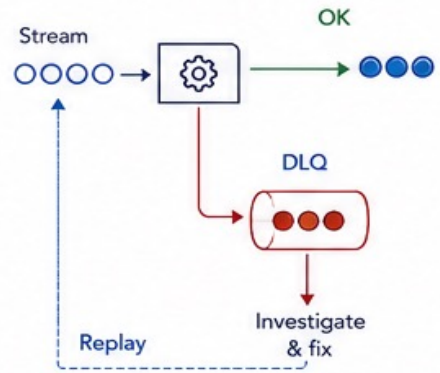


- Slow producers
- Drop or defer low-priority work
- Aggregate / sample events
- Switch to degraded mode

Keeps latency bounded and system stable.

5 Dead-Letter Queue & Replay

Isolate bad events without stopping the stream.



After fix, replay events (or range) to rebuild state and reprocess.

Idempotency Key

$$\text{idempotency_key} = H(\text{entityId}, \text{eventId}, \text{stateVersion}, \text{actionType})$$

- entityId: Who
- eventId: What event
- stateVersion: From which state
- actionType: What type of action

The receiver can answer: "Have I already applied this intent?"

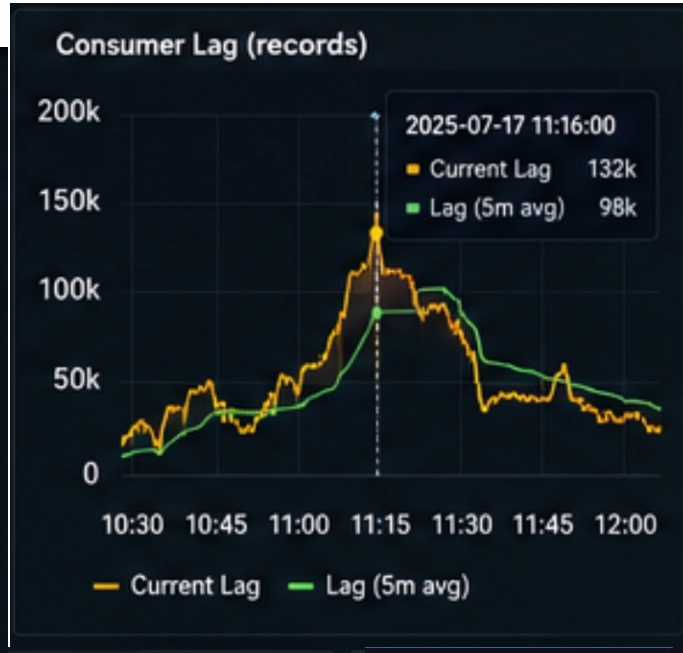
The Most Important Principle

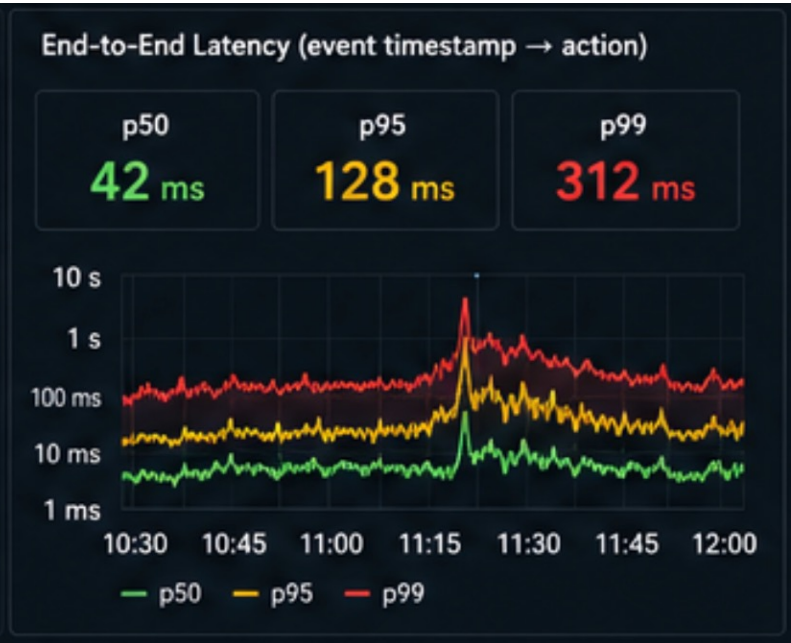
Exactly-once business behavior is often more important than exactly-once message delivery.

Messaging reduces duplicates, but only the application understands the meaning and can ensure side effects happen correctly.

11. Grafana metrics



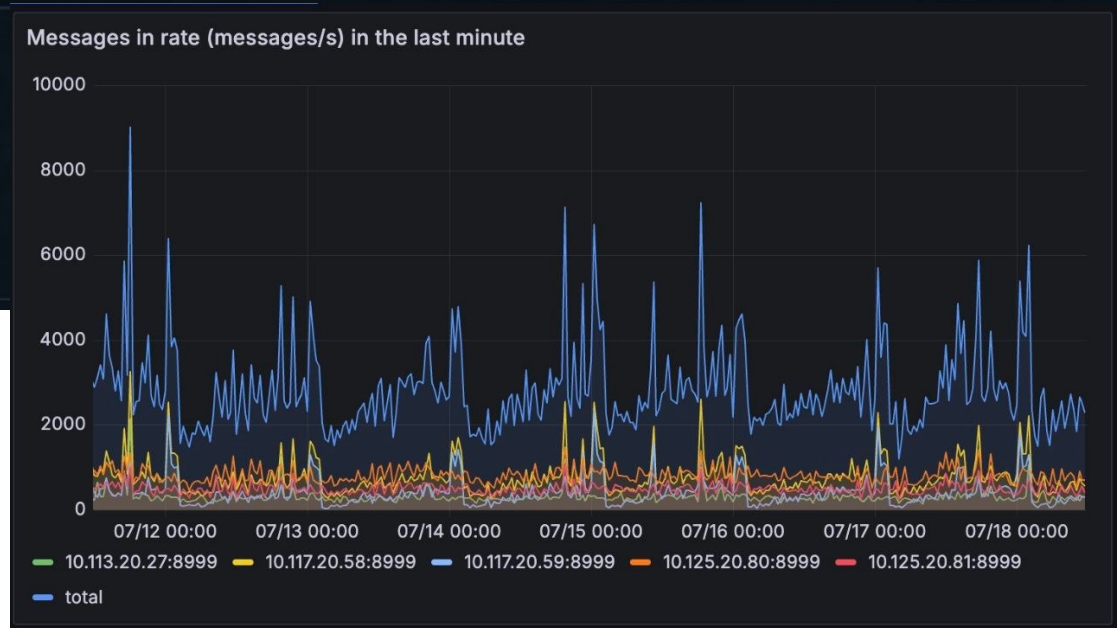












Key Point

Infrastructure metrics show platform health. Outcome metrics show decision correctness and safety.

12. Decision Reliability is Multi-Dimensional

A decision is reliable when it is correct, timely, once-only, recoverable, and explainable.

The Five Dimensions

Correct

The selected state transition is valid for the current context.

Timely

A correct decision delivered after its deadline may be operationally incorrect.

Once-Only Business Effect

Infrastructure may retry messages, but the action must not be repeated harmfully.

Recoverable

After a process failure, we can rebuild the necessary state and continue processing within objective.

Explainable

We can reconstruct why the decision was made and what the system knew.

Explainable Means We Can Reconstruct

- Which event initiated the decision?
- What data was available?
- What was its age?
- Which version of the state was used?
- Which model/policy version produced it?
- Was the command retried?
- Which service accepted it?
- Was fallback mode active?
- What was the final outcome?

Decision Audit Trail (Example)

- 10:15:21.123 Event received sensor.msg (entity=Robot-17)
- 10:15:21.125 State version v=3421
- 10:15:21.128 Estimate pos=(12.4, 5.1), σ=0.18, age=32ms
- 10:15:21.131 Policy decision policy=v3.7.1
- 10:15:21.134 Command issued cmd=STOP, id=9f13b..., retry=0
- 10:15:21.162 Command accepted actuator ACK, result=OK

From Definition to Measurable Objectives (SLOs)

Dimension	What We Measure	Example SLO / Objective	Why It Matters
Correct	Decision correctness (validation / rules / outcome)	Invalid decision rate $\leq 10^{-5}$	Prevents unsafe or invalid actions
Timely	End-to-end latency (event timestamp → action)	p99 latency $\leq D$ (deadline)	Ensures decision is still useful
Once-Only Effect	Duplicate commands & harmful duplicates	Harmful duplicate actions = 0	Prevents repeated side effects
Recoverable	Recovery time after failure	Recovery time $\leq R_{objective}$	Limits downtime and backlog accumulation
Explainable	Audit completeness (complete decision records)	Decisions with audit trail $\geq 99.9\%$	Enables root cause analysis and accountability

Key Takeaway

Reliability is multi-dimensional.
Operational excellence requires measuring both platform health and decision outcomes.

Platform Metrics (Health)
- CPU
- Memory
- Throughput
- Lag, Errors

+

Decision Metrics (Outcomes)
- Correctness
- Timeliness
- Once-only
- Recoverability
- Explainability

=

Reliable Decisions

→ Measure what matters. Improve what you measure.

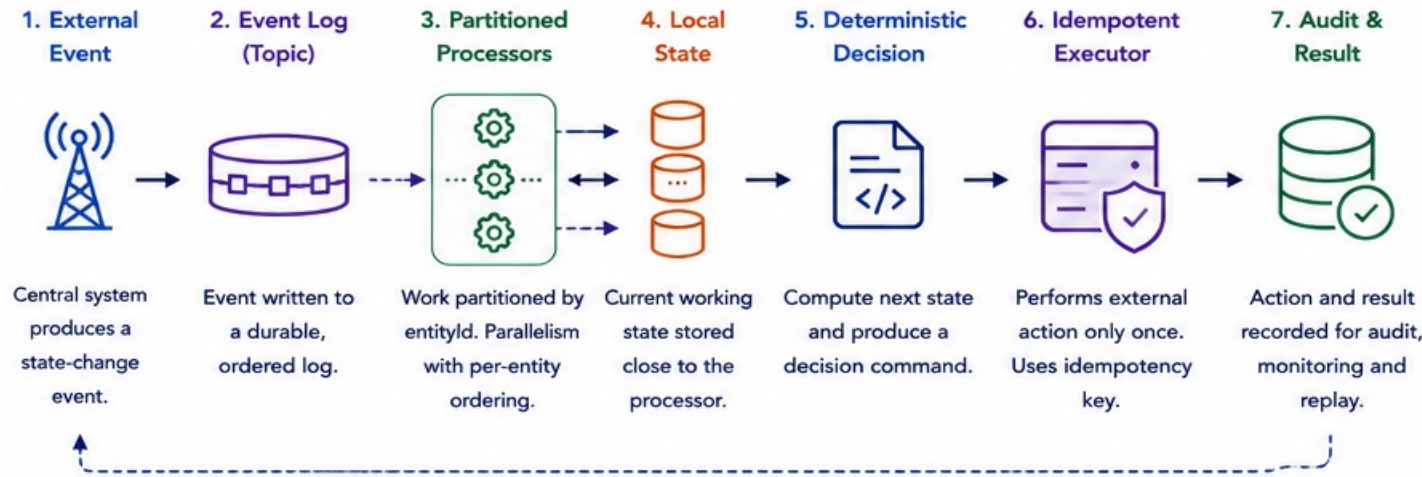
13. Anonymized Production Case Study

One external event can affect up to ~1,000,000 dependent entities.

The Problem with Naive Approaches

- Database becomes a bottleneck
- Sequential processing is too slow
- Downstream failures stop everything
- Retries may repeat actions
- Recovery is difficult and unclear

Event-Stream Architecture Pattern



All events, decisions, commands and results are recorded for audit and replay.

Why This Works

- Parallelism at scale (up to ~1M entities)
- Per-entity ordering
- Local state for speed
- Deterministic decisions
- Idempotent execution
- Clear recovery strategy
- Full auditability

Key Components and Responsibilities

#	Component	Role	Key Guarantees
1	External Event	Central system emits a state-change event.	Single source of truth for the trigger.
2	Event Log	Durable, ordered record of all events.	Durability, ordering per partition, replayability.
3	Partitioned Processors	Consume events by entity partition in parallel.	Per-entity order, horizontal scalability.
4	Local State	Keep current working state close to the processor.	Low latency, fast lookups, state versioning.
5	Deterministic Decision	Compute next state and create command.	Deterministic, explainable, versioned decisions.
6	Idempotent Executor	Perform external side effect safely.	Exactly-once business effect, safe retries.
7	Audit & Result Store	Record command, result, status and metadata.	Immutable audit trail, monitoring, replay.

Applies to Many Cyber-Physical Systems

AGV Fleet (Example)

- **Event:** Corridor blocked
- **Entities:** Affected vehicles
- **Decision:** Reacluulate route
- **Command:** New target route
- **Executor:** Vehicle controller

Smart Grid (Example)

- **Event:** Supply change
- **Entities:** Controllable loads
- **Decision:** Adjust limits
- **Command:** New limit setpoint
- **Executor:** Load controller

Industrial Line (Example)

- **Event:** Machine failure
- **Entities:** Tasks & robots
- **Decision:** Reassign / reroute / safe shutdown
- **Command:** New task / route / shutdown action
- **Executor:** PLC / robot ctrl.

14. My publications in the real-time decision pipeline

decision pipeline



01 Perception & state estimation

Motion detection, tracking and NN target recognition turn raw signals into state and confidence.

[P1] [P6]

02 Optimization under constraints

ACO searches for high-quality decisions while routes, resources and conditions change.

[P2] [P4]

03 Message representation

Serialization affects size, CPU cost, compatibility and high-throughput event latency.

[P3] [P5] [P7]

04 Cloud execution & logistics

Serverless/cloud warehouse optimization links computation with physical planning.

[P2]

05 Trustworthy AI decisions

External inputs must be separated from trusted context to resist prompt injection.

[P8] [P9]

Common research question: how to make reliable decisions when data, resources, latency and trust constraints change in real time.

References used as connection points

[P1] Tsiunyk & Muliarevych (2022). Software System for Motion Detection and Tracking. ACPS.
[P2] Kniazhyk & Muliarevych (2023). Cloud Computing With Resource Allocation Based on Ant Colony Optimization. ACPS.
[P3] Maltsev & Muliarevych (2024). Beyond JSON: Evaluating Serialization Formats for Space-Efficient Communication. ACPS.
[P4] Danyliuk & Muliarevych (2024). Ant colony algorithm in traffic flow control. ACPS.
[P5] Maltsev, Muliarevych & Razzaque (2024). Classifying serialization formats for inter-service communication in distributed systems. ACPS.
[P6] Tsiunyk & Muliarevych (2024). Performance analysis of different types of NN models for target recognition. ACPS.
[P7] Maltsev & Muliarevych (2025). Optimizing inter-service communication in latency-sensitive distributed systems. DESSERT.
[P8] Muliarevych (2024). LLM-driven defense against prompt injection vulnerabilities. TCSET.
[P9] Muliarevych (2024). Mitigating input prompt attack vulnerabilities in systems with a language model interface. DESSERT.

Q&A

Thank you for your attention!

Reliable decisions in distributed systems connect **perception**, **communication**, **optimization**, **execution** and **trust**.



