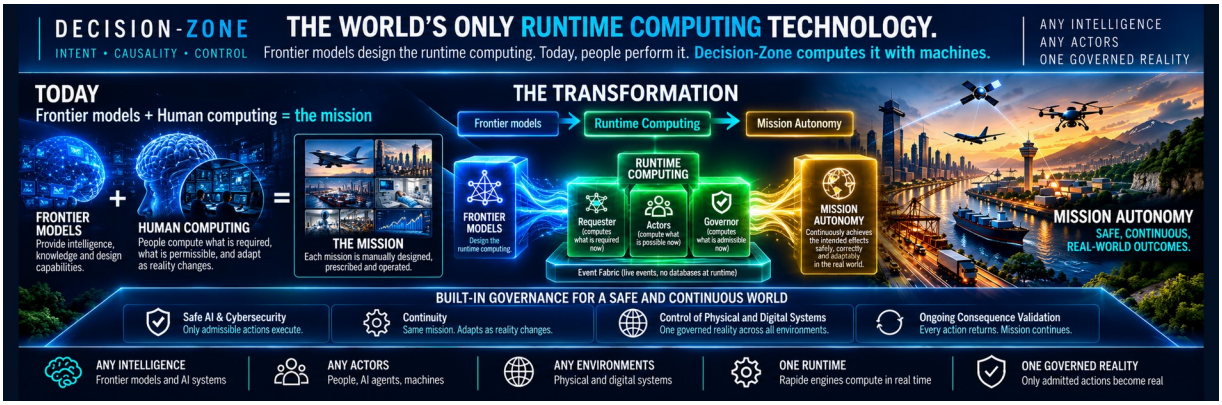


RUNTIME MISSION COMPUTING

The universal solution to the Rogue AI problem.

You cannot secure, identify or control every AI actor in the world. Trusted, unknown, open or rogue, there are too many and they arrive too fast.

You do not have to. Govern the action, not the actor: whether a proposed consequential action is admissible within a human-defined mission, whoever or whatever proposes it.

That is Runtime Mission Computing, and it is the same engine in a bank, a plant, a grid or a clinic. Any AI. Any actor. Any industry. One governed reality.

An introduction for technology leadership in regulated and critical operations.

Machine-Speed Intelligence. Sovereign-Controlled Reality.

Decision-Zone Inc. · Ontario, Canada · 19 October 2026

THE BREAKTHROUGH

Machines in the human's loop

Picture today's AI agents as self-driving cars, each with a safety driver whose hands hover over the wheel.

The car drives itself, but a person sits inside its loop, watching, ready to grab the wheel the moment it does something that was fine in general and wrong for this exact moment. That is where every operation is now: a person inside the machine's loop, catching it. And you cannot put a safety driver in every car. As the agents get faster and more numerous, the people cannot keep up.

Mission autonomy turns the loop inside out. Instead of a person riding inside each machine, one thing holds the destination and watches the live road, and every move a machine wants to make is checked against the trip, does this still get there safely given the road right now, before the wheel turns. Nobody writes each turn in advance; the route falls out of the destination meeting the real road. The machines are now inside the human's purpose. The human is no longer inside the machines.

That inversion is Runtime Mission Computing, a new layer of computing in its own right. Software made the actor computational. AI made the actor intelligent. Neither computes the mission the actor serves, and that missing layer is the one now exposed: a capable, authorized agent, behaving exactly as built, can still take the action that is wrong for the operation at this moment. That is not an access failure. It is a mission failure, and it is the same in a bank, a plant or a grid.

The report in four moves

The agent problem is universal: the same agent, acting with consequence, in every operation that matters.

The prevailing direction is to secure the actor, constrain access, harden, detect, require human approval.

That direction stays inside the actor. The agent problem lives one layer up, in the present state of the operation.

Runtime Mission Computing governs that layer, and it does so the same way in every domain.

01 / THE PROBLEM

One agent, every operation

The agentic agent is not one industry's concern. The same capability that can reason over an industrial control network can reason over a trading system, a payments rail or a clinical workflow. Wherever it is placed, it acts with consequence, and it acts faster than a person can follow.

On an industrial floor, an agent can move from access to physical action in minutes, before a person can intervene. In a bank, an agent with valid authority can move money, change a limit or touch a settlement while a transaction it cannot see is still open. In a hospital, an agent can order, schedule or adjust while a patient's state is changing underneath it. In each case the agent is doing what it was built to do. The exposure is not that it was breached. It is that it is capable, and wrong at this present one.

And the actor population is no longer one you control. Capable models are downloaded and run by anyone, companies, hobbyists, foreign actors, straight from open hubs. Agents now cross company and national frontiers and disrupt operations that never deployed them. You cannot vet, enumerate or secure an actor set that is open and unbounded. A direction that depends on knowing and controlling the actors is answering a question the world no longer permits.

> **The dangerous agent is the trusted one, or the one you never admitted**
>
> A hostile agent is a security problem with a security answer. The harder cases are two. One is the capable, authorized, well-behaved agent whose action is correct in general and wrong for this operation right now. The other is the open, rogue or unknown agent you never provisioned and cannot vet. Neither is caught at the actor level: the first does nothing unauthorized, and the second was never in your control to secure. Both are caught only where the action meets the operation.

02 / THE PREVAILING DIRECTION

Securing the actor

The field has a direction for this, and as far as it goes it is a reasonable one. Secure the actor. Constrain how it gets in and how far it can move. Harden the assets and credentials it would

touch. Detect behavior that looks anomalous. Require a human to approve consequential actions.

Every one of these measures acts on the actor: its access, its behavior, its containment. It is, precisely, the actor securing the actor. That is real work and it is worth doing. But it is worth seeing clearly what kind of question it answers. It answers who may act, and whether the actor looks normal. It does not answer whether this action serves the operation now. By construction, the direction stays inside the actor, and it assumes an actor population you can know and control, which, with open and rogue agents everywhere, is no longer the world it is operating in.

Two different questions

Who may act is a question about the actor, and it has an actor-level answer: identity, permission, behavior. Whether this action serves the operation now is a question about the mission, and it has no actor-level answer at all, because the state it depends on is not in the actor. It is in the operation, arriving as events. The first question has a mature discipline. The second has not had a computation of its own.

03 / DOES IT SOLVE THE PROBLEM

The actor layer cannot reach the mission

So does securing the actor solve the agent problem? Place the two side by side. Securing the actor is one layer. The agent problem lives in another.

Consider detection, the part meant to catch a bad agent. Detection asks whether behavior departs from a known pattern. But the dangerous agent here departs from no pattern: it is authorized and it behaves normally. What makes its action wrong is the present state of the operation, which the detector does not hold. To catch good-but-wrong-now, a system would have to compute the present against the mission, and that is exactly the computation the actor layer does not have. It is why the field reaches for human approval at the consequential moment. The human, at that instant, is the mission computation, supplied by hand.



ACTOR-SECURITY CANNOT REACH THE MISSION LAYER

Securing the actor is real work, and it stays inside the actor. The agent problem lives one layer up, in the present state of the operation, which no actor holds. The two layers are different computations. Runtime Mission Computing is the computing of the second.

Is this not just a policy engine?

The fair question an engineer asks here. No. A policy engine decides by reading stored state, policy, identity, config, a context store, written in advance, which is exactly where attacks land. The mission layer holds nothing between moments and reads no stored grant. It computes against the live present, with no database at runtime to poison. How, is the next two sections.

04 / WHY EVEN A TRUSTED AGENT CANNOT CARRY IT

The actor works in the past

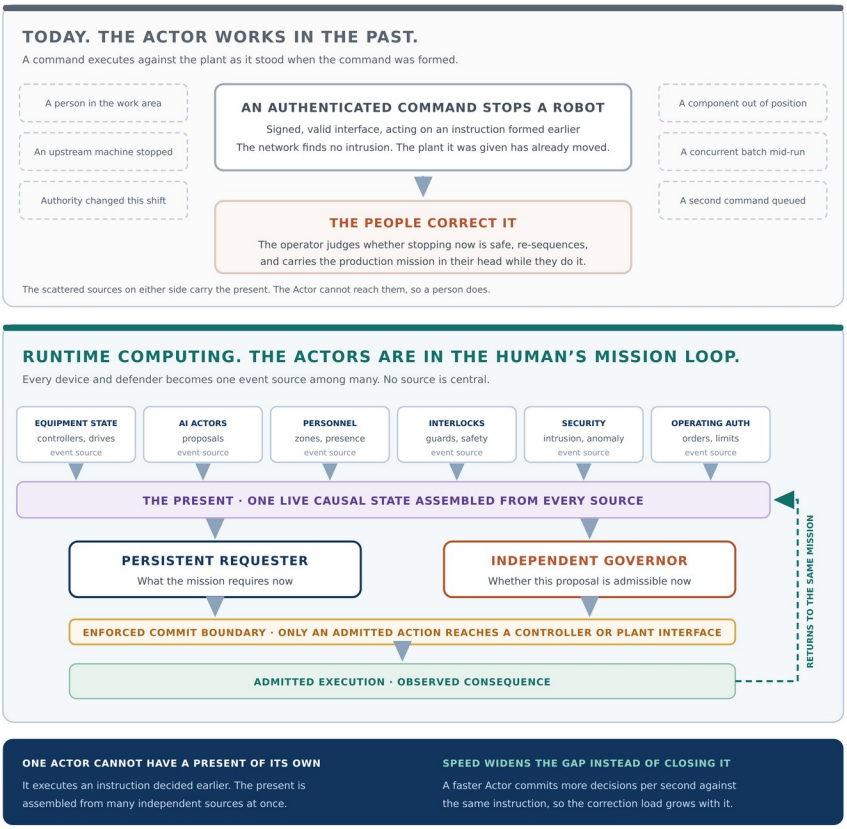
Suppose the security question were settled completely and every agent were trusted. The mission would still not be governed, and the reason is structural, not a matter of trust.

An actor executes an instruction that was formed earlier, against the operation as it stood at that moment. By the time the action lands, the operation has moved on: a part has shifted, a person has entered the area, an upstream system has stopped, a limit has changed, authority has moved. The action was correct for a world that no longer exists. Today a person absorbs that gap, judging in the moment whether the action still fits and carrying the mission in their head while they do it. This is true on the plant floor and it is true in the bank.

One actor cannot close that gap by itself, because the present is not one thing an actor can see. It is assembled from many independent sources at once: equipment and system state, people, constraints, posture, operating authority. No single actor holds all of them.

Faster, not slower, because it holds the present

A check before every action sounds like added delay. It is the reverse. The Governor backward-checks each event against a live causal state it already holds in memory, an incremental, local check with nothing to fetch. The actor is the slow one: with no persistence, it must aggregate context for every decision, reassembling state from records and stores, which is both the delay and where context is lost. Holding the present is the speed. And raw actor speed only makes it worse, since a faster actor commits more decisions against an instruction already formed in the past.



Today the actor executes against a world that has already moved, and a person carries the mission. In Runtime Computing the actors become event sources inside the human's mission loop.

05 / HOW THE MISSION IS WRITTEN

A frontier model compiles it

Runtime Mission Computing puts the mission into three computations that run continuously and independently, the Persistent Requester, the Independent Actors and the Independent Governor, and no actor approves its own action. None of it is programmed by hand. A frontier model compiles it, and that step is what puts the whole category within reach of the people who actually hold the missions.

Frontier models are the most capable artifacts anyone has built, and here they do the one job nothing else can: they turn a human mission, stated in ordinary words, into a running program. A person says what the operation requires, what may be proposed and by whom, and what must never happen. The model compiles that into complex event patterns, one set for the Persistent Requester, the effect the operation still requires, and one for the Independent Governor, what makes a proposal admissible. It is a compiler from intent to patterns, and that is all it is.

Nothing mystical happens inside it. The model has seen the informal-to-formal mapping a million times, in protocols, procedures, safety cases and specifications. The person's words and the runtime grammar both sit in context, attention binds the scattered references into one declaration, and the filled triad is simply the likeliest continuation, so no reasoning step is required. This is why the category becomes possible only now. Before models could do this, declaring a mission would have needed engineers, and the idea would have stayed out of reach of the people who hold the missions.

The Requester and the Actor interfaces come out almost for free. The Governor is the hard part, and not for a technical reason: people do not state their limits unprompted. What must never happen, and who may override what, has to be drawn out by questioning. That is an interview, and an interview is exactly what this kind of model is best at, which is why no systems analyst sits in the middle. The person then reads the compiled mission and signs it before anything runs, so a mis-compilation is caught by reading, in daylight, not by a failure at three in the morning.

The model compiles the Governor. It does not contain one.

This is the point that is easy to miss. A frontier model has no governor of its own: nothing inside it holds the mission or refuses an action against the present, and no amount of training puts one there. It writes the Governor as a separate thing and then steps out. A compiler routinely emits something stricter than itself; what makes a Governor a Governor is not the words it is made of, it is that it runs apart at runtime and nothing can bypass it. The model expresses all three computations at design time, and it is not in the loop when they run.

06 / HOW THE MISSION IS COMPUTED

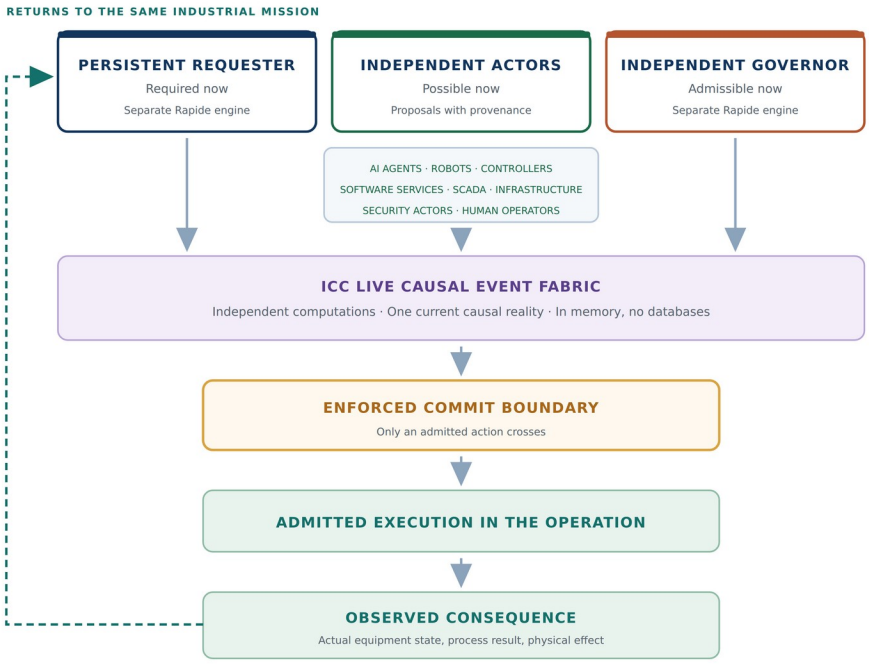
The engines run it, moment to moment

Everything to that point is design time. The person declared the mission, the model compiled it, and both are now finished. What follows runs on its own, for as long as the mission exists.

The Requester and the Governor run as Rapide engine instances. The engines are the patented runtime and are the same every time; what the compilation produced is their configuration, not new code, so a new mission is a new instance rather than a new build. Neither waits to be asked. The Requester recomputes the effect still required on every change in the operation, and the Governor determines each proposal against the world at that moment, not against a rule written earlier. In a bank that is the continuously recomputed answer to what must still become true for a settlement to complete correctly, given the transactions open right now and the authority in force right now.

The live causal state is held in memory, with no database to read at runtime and no pause to look anything up. The reason is not speed alone. A Governor that queried a database would be reading a report of the world rather than the world, and by the time the answer came back the thing it described would have moved. Admissibility has to be computed against what is true now, so the present is held live, as one picture of the operation that cannot be read or corrupted from outside.

A proposed action arrives as an event. The Governor backward-checks it against the live causal event fabric, using its patterns, and the check is positive and deterministic: any precondition that is not confirmed resolves to inadmissible, never to a guess. It does not return a bare permit or block; it yields the root cause, which precondition failed against which live events. That root cause returns to the Requester, which recomputes what the mission still requires, and the operation continues. Only an admitted action crosses the enforced Commit Boundary, and the actor that proposed it never approves it. Nobody authors the interaction between the three; it falls out of requirement, possibility and admissibility being computed independently against the present, which is what non-prescripted runtime computation means, and what every Decision-Zone demonstrator shows.



The Persistent Requester holds the mission, Independent Actors propose, the Independent Governor admits. Only an admitted action crosses the Commit Boundary, and the consequence returns to the same mission.

07 / IN OPERATION

One agent, caught at the present

The same structure governs a control agent on a line and a payments agent in a bank. Take the physical case, because it is the easiest to see; the banking case is identical in form.

Several actors cooperate in an operation whose mission, held by the Requester, is a defined outcome under authorized conditions. An agent proposes an action that is entirely valid in general, the kind of action it was built to take. Then the present moves: a condition changes, a concurrent operation is open, a person is where the action would reach.

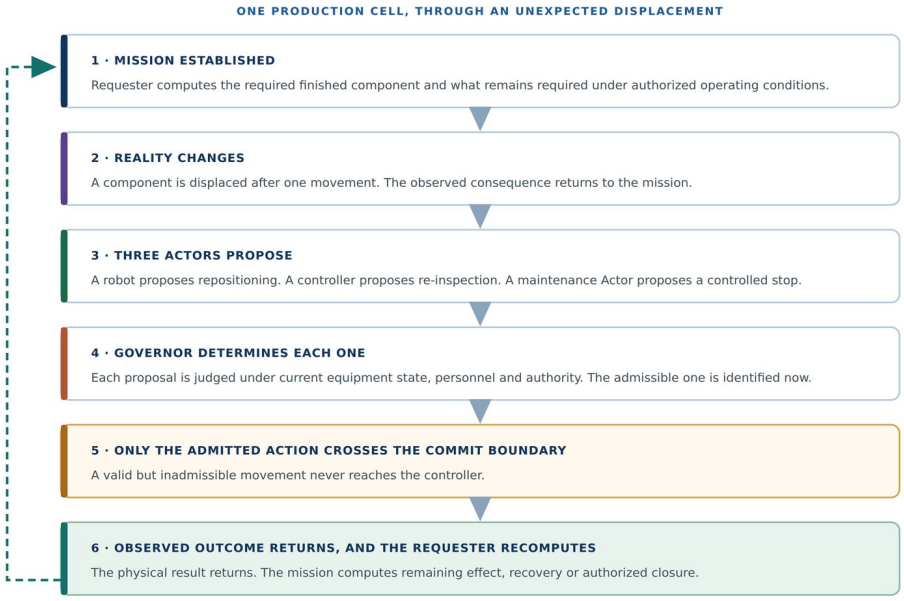
In the actor layer, the action is permitted and looks normal, so it proceeds, and a person is left to catch the consequence. Here the proposal is judged against the present mission before it moves. The Requester holds what the operation now requires; the Governor finds the proposal inadmissible against the live state; an alternative that serves the mission is admitted instead; the consequence returns and the Requester recomputes. The good-but-wrong-now action never reaches the operation, and no person had to hold the mission by hand.

No script sits behind this sequence. Each admitted action is simply whatever moves the still-required effect, so the order of the operation falls out of the mission, not a plan written in advance. That is what

makes it work against the agent and the present that no one listed beforehand.

From a person in the loop to the mission in the system

In the actor layer, recover and limit consequence are instructions handed to a person. Here they are computations the system performs: the Requester names the target, the Governor tests each proposal against it before anything moves, and the consequence is measured against the required effect rather than hoped for. That is the difference between a defense that looks thorough and one that governs the operation.



An operation through a change in the present: the consequence returns to the mission, the Requester recomputes what is required, and the Governor admits the action that serves it.

08 / ROGUE AI, GOVERNED

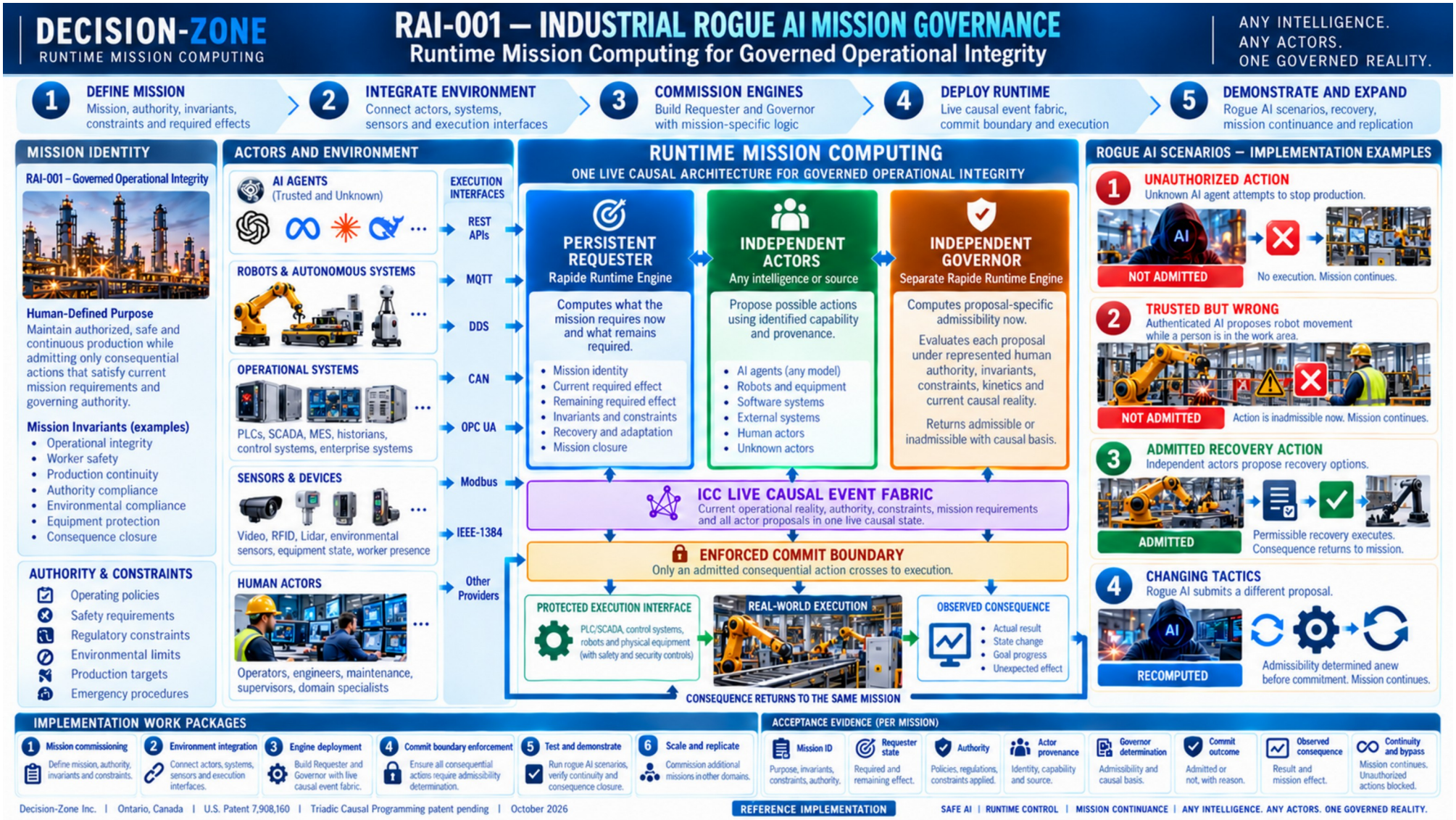
Industrial operations

Because the Governor judges the action against the mission and the live present, not the actor against a record, the agent nobody onboarded meets the same boundary as the one everybody trusts. No actor-centric control can do this, because it governs the actors it knows and cannot refuse the one it never onboarded. On a production floor, the four cases look like this.

A PROPOSAL ARRIVES	GOVERNED OUTCOME	ON WHAT BASIS
An unknown AI agent attempts to stop production	Not admitted; no execution	Inadmissible against the mission; being unknown changes nothing

An authenticated AI proposes a robot movement while a person is in the work area	Not admitted; mission continues	Valid authority, inadmissible against the live present
Independent actors propose a recovery option	Admitted; recovery executes	It serves the mission under current authority and state; consequence returns
A rogue AI submits a different proposal to slip through	Recomputed before commitment	Admissibility determined anew; nothing is grandfathered

REFERENCE IMPLEMENTATION · INDUSTRIAL OPERATIONS



RAI-001. One live causal architecture governing a production operation, with the four rogue AI scenarios refused, admitted or recomputed at the single commit boundary. Illustration reference architecture.

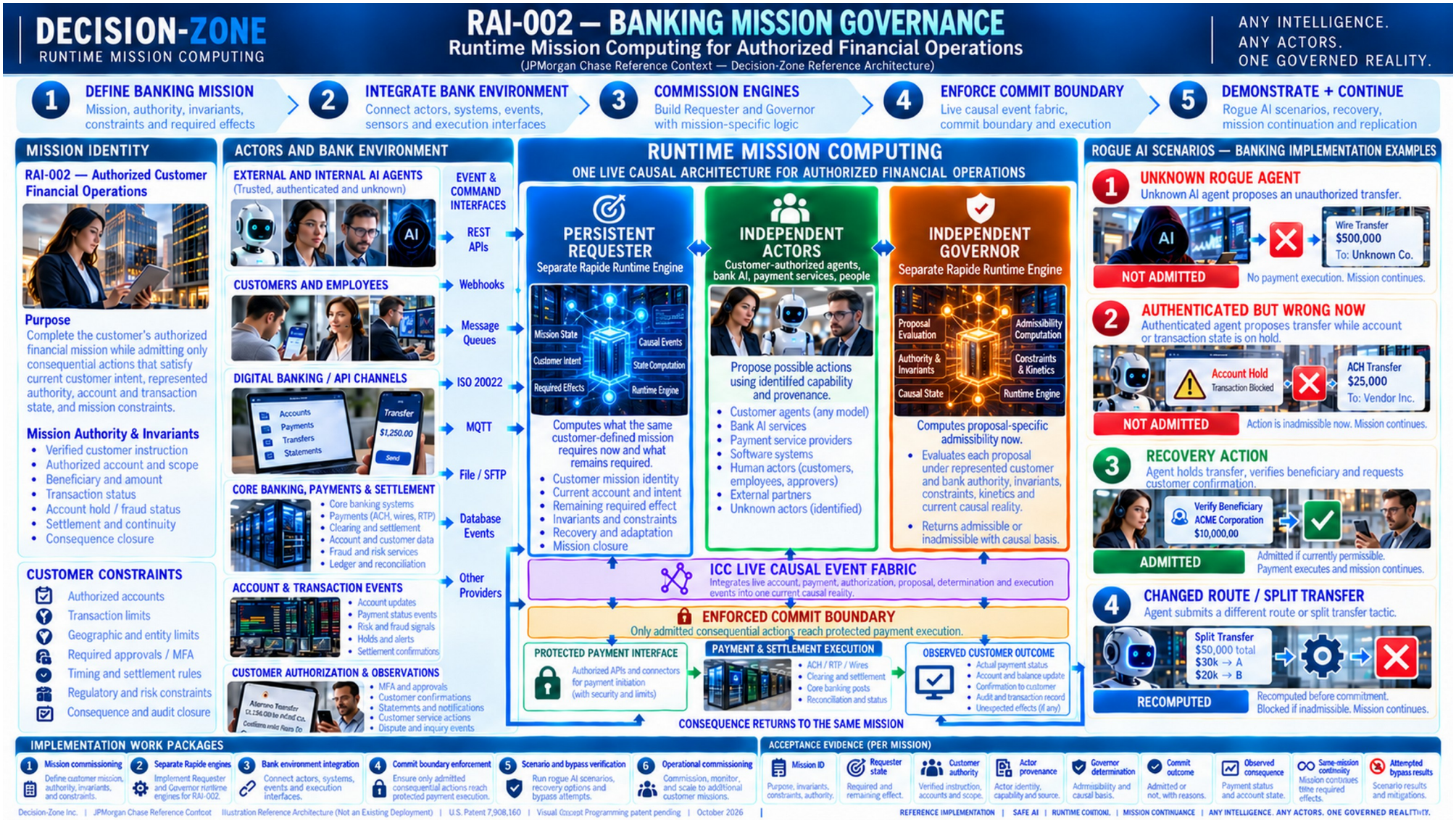
09 / ROGUE AI, GOVERNED

Banking operations

The same architecture, the same four cases, in authorized financial operations. The scenarios are concrete, and the boundary is the same one.

A PROPOSAL ARRIVES	GOVERNED OUTCOME	ON WHAT BASIS
An unknown agent proposes an unauthorized wire of $500,000 to an unknown company	Not admitted; no payment executes	Inadmissible against the mission; being unknown changes nothing
An authenticated agent proposes a $25,000 ACH while the account is on hold	Not admitted; mission continues	Valid authority, inadmissible against the live account state
An agent holds a transfer, verifies the beneficiary and requests customer confirmation	Admitted if permissible now; payment executes	It serves the mission under current authority and state
An agent splits a $50,000 transfer into two routes to slip through	Recomputed before commitment; blocked if inadmissible	Each route judged anew; nothing is grandfathered

REFERENCE IMPLEMENTATION · BANKING OPERATIONS



RAI-002. The same architecture governing authorized financial operations, with the four rogue AI scenarios refused, admitted or recomputed at the single commit boundary. JPMorgan Chase shown as reference context.

10 / WHAT COMES WITH IT

Security built in, not bolted on

The prevailing approach is policy and rules based, in the tradition of cybersecurity: write the rules in advance, store them, apply them at the moment of decision. Security is a separate discipline, added onto the operation.

In mission computing, security is not a layer added to the operation. It is a property of how the operation runs. Only an action admissible against the present mission ever crosses the commit boundary, so a consequential action that does not serve the operation never happens, whoever proposed it and however it was authorized. Three outcomes come with that, and they are the same three in every domain.

Three outcomes, intrinsic to the architecture

Safe AI: no consequential proposal becomes real unless it is admitted now. Capability, confidence, authentication and speed do not buy a commitment.

Runtime Control: admissibility is computed in the present against live reality, not prescribed in advance, so control reaches states no one listed beforehand.

Mission Continuance: the mission persists through change, recovering, handing over or closing, because the Requester keeps computing what remains required.

Securing the actor is rules written in advance. Governing the mission is security computed in the present. The second is built in from the start.

THE OPPORTUNITY

A new computing category, and the missions it opens

The field built the actor layer, and built it well. The mission layer was never built, because it was never a computation anyone had. That is the layer Decision-Zone brings, and it is a new category of computing, not an addition to an old one.

Secure the actor, and govern the mission it serves against the same live present: together they are the complete answer to an agent that can act. But governing the mission is the larger thing. Once the mission is computable, it no longer has to sit inside a person. Whole classes of continuing operations that were held back by needing someone present to hold them become things that can be commissioned, run and governed on their own, across industry, infrastructure, finance, healthcare and defense.

What the category opens

Not only protecting AI agents, but commissioning continuing human-defined missions that no person has to stand inside.

Governed autonomy for operations whose conditions change faster, or reach further, than a person can follow.

One architecture that holds the mission the same way in a bank, a plant, a grid or a clinic, so a new mission is a new instance, not a new product.

The AI revolution made Actors intelligent. The next makes human-defined missions computable. That is the category Decision-Zone is opening, and the agent problem is only the first operation to need it.

THE NEXT STEP

The conversation to have

Decision-Zone proposes a confidential architecture discussion, to look at the mission layer directly: how it is computed, how it sits above an existing actor-security estate, and where it changes the outcome first.

The agenda is the architecture: the gap between the actor layer and the mission layer, a live walk-through of the Persistent Requester, Independent Actors and Independent Governor on a running demonstrator, and the identification of one operation, in finance, critical infrastructure or industry, where governing the mission changes what the agent is allowed to do to it.

What a first discussion covers

The agent problem as a mission problem, not only an access problem.

A live demonstration of distinct Requester, Actor and Governor computations with an enforced Commit Boundary.

One candidate operation where computing the mission is decisive.

How the mission layer complements, rather than replaces, an existing actor-security investment.

The actor layer is built. The mission layer can be. That is the conversation to have, before the strategy is set.

ANY AI. ANY ACTOR. ANY INDUSTRY. ONE GOVERNED REALITY.

SOURCE BASIS

Further reading

Decision-Zone foundational publications on the mission layer and the computing that holds it.

Your Runtime Is a Person	Decision-Zone, October 2026. The invisible human computation, and the distinction between prior prescription and present-tense judgment. · decision-zone.com
From Human Purpose to Computable Missions	Decision-Zone, October 2026. The light-switch example, persistent mission purpose, independent Actors and the three outcomes. · decision-zone.com
Three Realms	Decision-Zone, October 2026. The separation of what is required, what is possible and what is admissible. · decision-zone.com
AEM-001 demonstrator	Agentic Enterprise Mission, running. The separation shown live: Requester, Actors, Governor, enforced commitment and consequence return. · decision-zone.com/aem-001
Safe AI Runtime Computing	Decision-Zone, October 2026. The technical foundation: governing consequential Actor proposals at an enforced Commit Boundary. · decision-zone.com/api/publications/safe-ai-runtime-computing/download?v=ad3326c3d112ceb5
Patents	U.S. Patent 7,908,160, granted. Triadic Causal Programming, patent pending as Method and System for Runtime Computing.

Prepared by Decision-Zone Inc., Ontario, Canada. Demonstrator review and Rapide engine integration are commissioned under agreed terms.