

A vibrant scene of a tulip field in a Dutch town square. The foreground is filled with numerous bright orange and yellow tulips. In the background, a wide pedestrian square is visible with many people walking. Historic Dutch-style buildings line the square, and a tall church spire with a red flag is prominent in the distance. The sun is low in the sky, creating a warm, golden light.

When Software Starts Talking to Itself

Adi Polak

LIFECYCLE OF INNOVATION

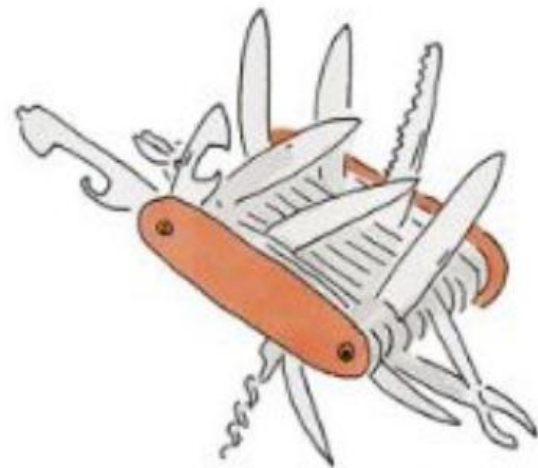
TOM
FISH
BURNÉ



CUSTOMER NEED



LAUNCH PRODUCT



NEW FEATURES



NOW WITH AI



RAISE PRICES



DISRUPTION



A DEMO THAT LOOKS LIKE MAGIC. UNTIL IT RETRIES.

ONE REQUEST

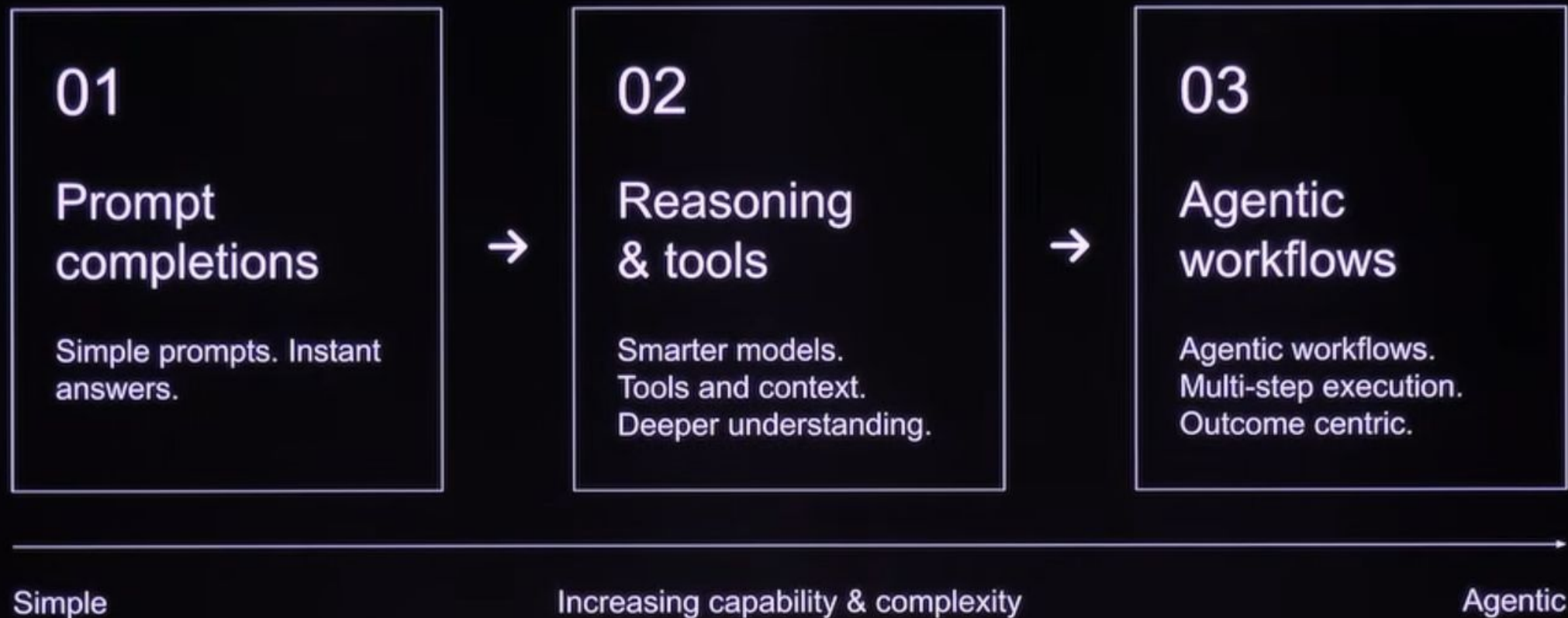
"Secure my upgrade."

TWO SIDE EFFECTS

Retry → two booking attempts

A broker can redeliver. Your agent must not re-purchase tickets.

Product evolution



AI AGENTS CAN

- **interpret events**
- **make decisions**
- **call tools**
- **create new work**
- **communicate with other agents**
- **change behavior based on what they observe**

are we
DOOMED?

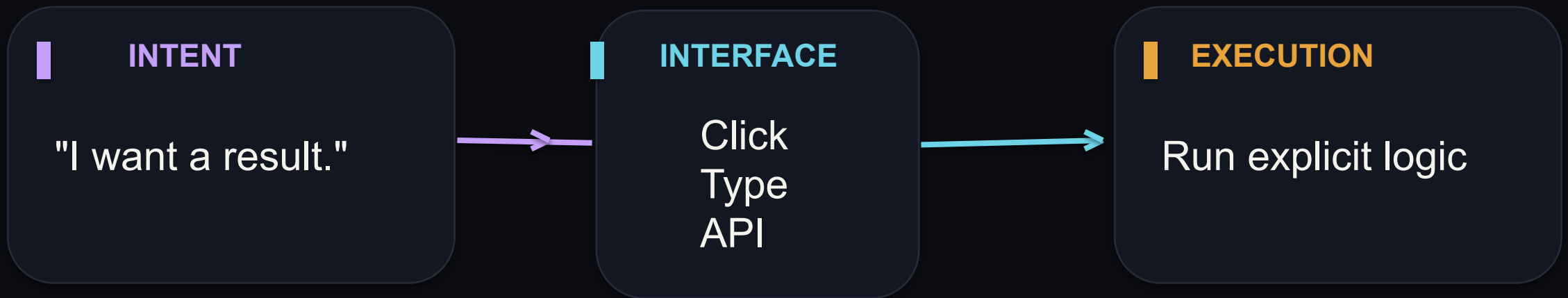
A message gets delivered.

A consumer runs code.

But who chooses the next action?

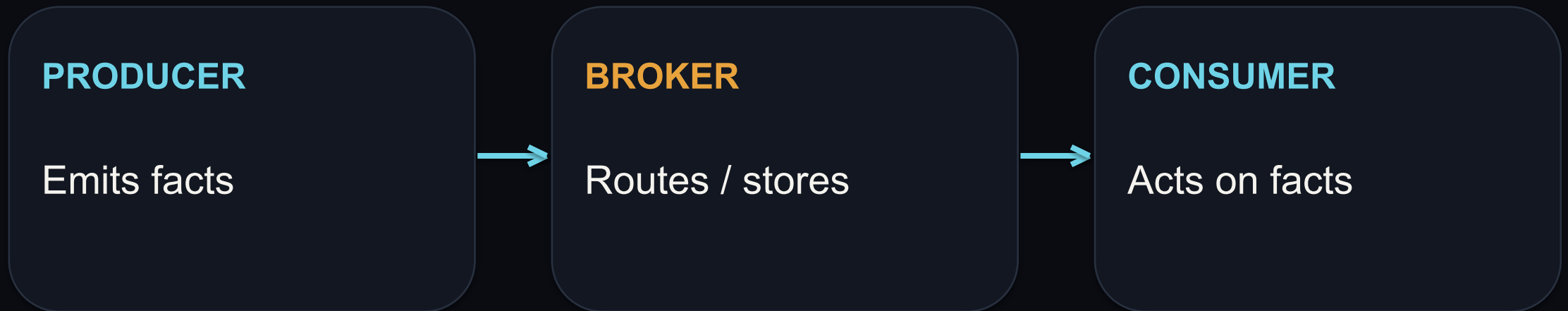
3 Layers

LAYER 1 / Humans talk to software



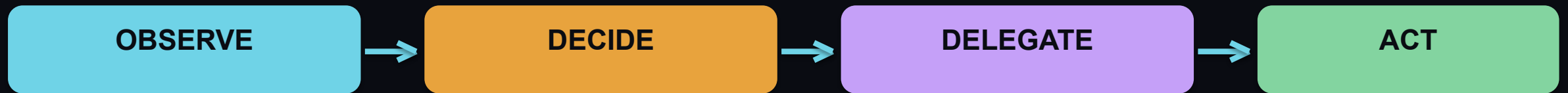
Human intent → software behavior → human observes the outcome

LAYER 2 / Software talks to software



Event streams · Work queues · Pub/sub · RPC / APIs

LAYER 3 / Software starts talking to itself



The code defines the boundaries. The agent chooses the path.

THREE LAYERS. ONE SYSTEM.

03

AGENT DECISIONS

Goals · planning · delegation · tools

02

DISTRIBUTED COMMUNICATION

Event streams · queues · durable messages

01

HUMAN INTENT & OVERSIGHT

Interfaces · approvals · policies



So what is the unit of work, today?

THE UNIT OF WORK



A message is a delivery unit. A task can be much larger.

THREE EXPECTATIONS WE COULD REASON ABOUT

EXPLICIT PATHS

**The code defines
the branches.**

MEASURABLE

**Work can be
measured and
capped.**

SCOPED EFFECTS

**Permissions
govern
what it may touch.**

What happens when the consumer is no longer a deterministic service?

What happens when the message reaches an agent whose choice depends on changing observations?

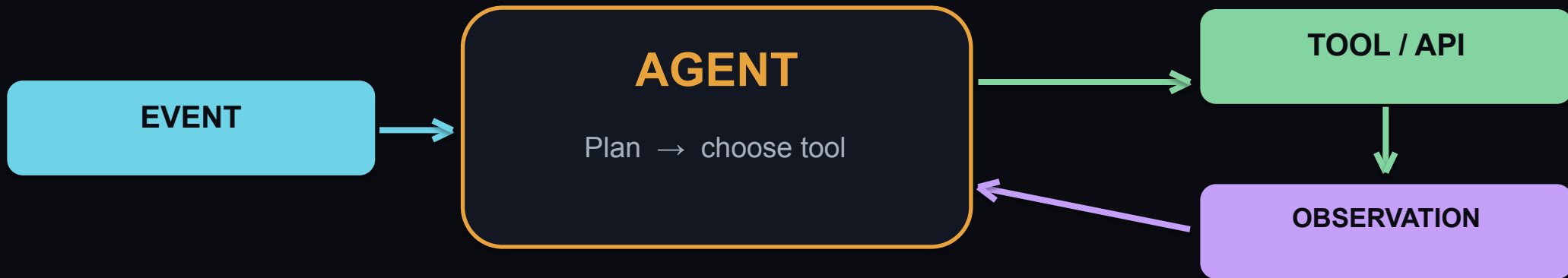


NEXT To answer, we have to see the patterns emerging in actual agent architectures.



AGENTIC SYSTEMS PATTERNS

PATTERN 01 / The tool-use loop



RELIABILITY QUESTION:

**How do you bound iterations
and authorize the next tool call?**

PATTERN 01 - agent call APIs

A flight-upgrade agent can observe the world and take real actions.

01

READ

Flight search

`search_available_upgrades()`

Find available seats

02

READ

Preferences DB

`get_traveler_preferences()`

Check passenger preferences

03

READ

Calendar API

`check_calendar()`

Check schedule conflicts

04

WRITE

Booking API

`upgrade_seat()`

Reserve a new seat

05

WRITE

Payment API

`charge_card()`

Process payment

06

WRITE

Email API

`send_confirmation()`

Notify the passenger

THE IMPORTANT DISTINCTION

A search reads state. A booking or charge creates an external side effect.

PATTERN 01 -One request. Multiple decisions. Multiple tool calls.

The agent selects what to invoke, in what order, and whether to try again.



PATTERN 02 / Sequential chain

EXT

Age

AFT

nt C

RELIABILITY QUESTION

what if the intermediate errors propagate as plausible-looking decisions

LAYER 2 AN

ons.

A workflow chooses the order; each agent owns a step.

PATTERN 02 / Sequential chain

EXT

Age

AFT

nt C

RELIABILITY QUESTION

What happens when intermediate errors propagate as plausible-looking decisions?

LAYER 2 AN

ons.

A workflow chooses the order; each agent owns a step.

PATTERN 03 / Router + handoff

TASK

RELIABILITY QUESTION

ING AGENT

EL AGENT

ESCALATION

PATTERN 03 / Router + handoff

TASK

RELIABILITY QUESTION

Who owns the task after a handoff? Who retries?

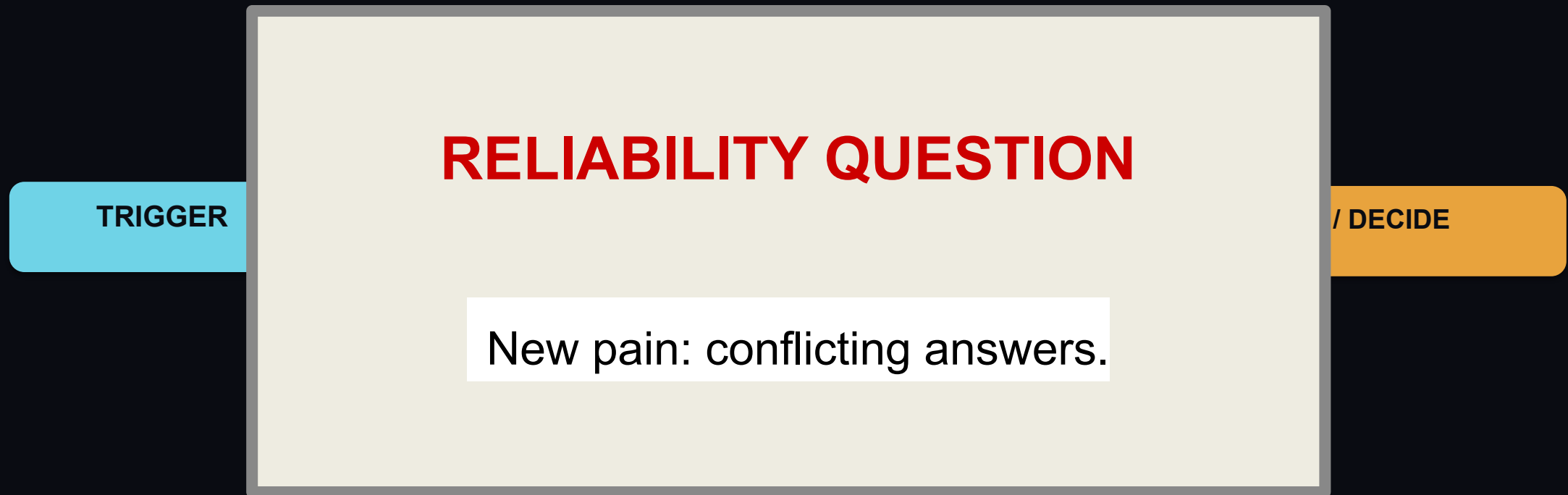
ING AGENT

EL AGENT

ESCALATION

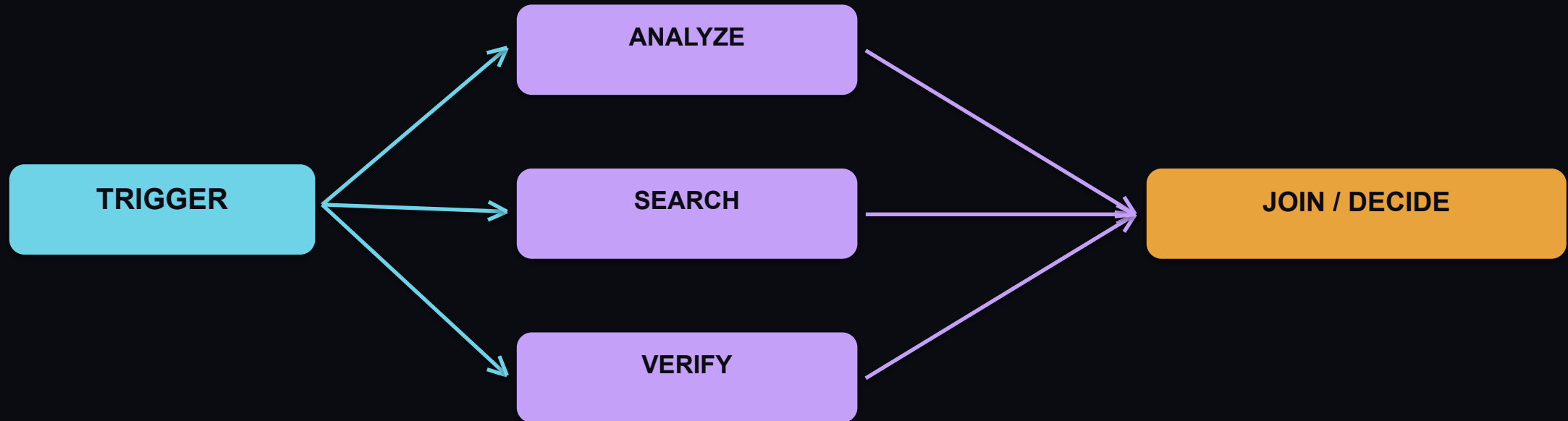
The next consumer is selected from the task context.

PATTERN 04 / Fan-out + fan-in



Layer 2 ancestor: pub/sub fan-out, scatter/gather.

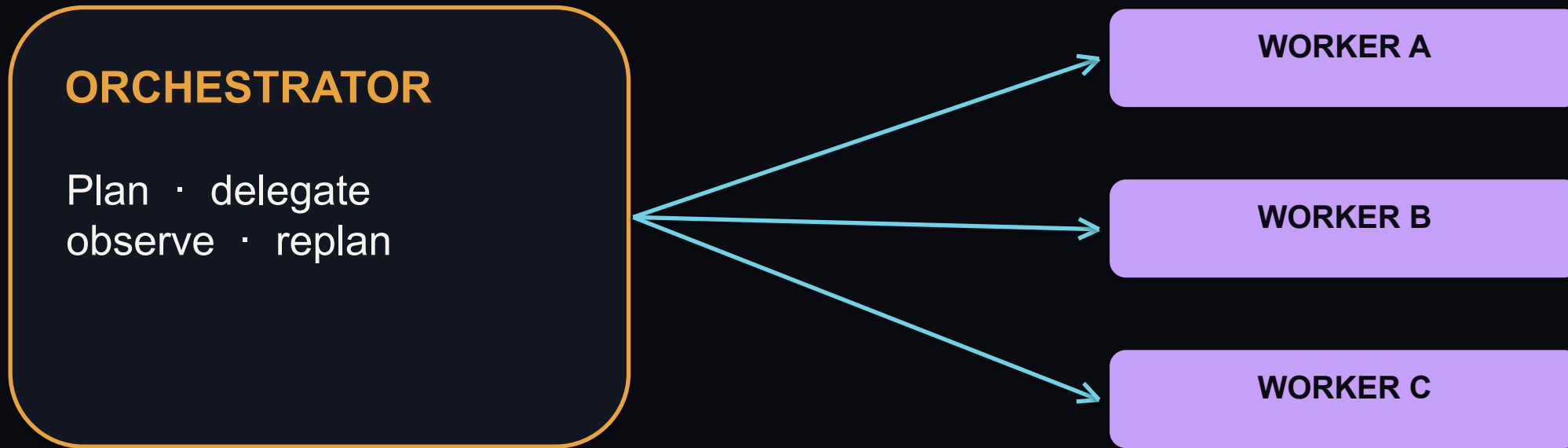
PATTERN 04 / Fan-out + fan-in



Layer 2 ancestor: pub/sub fan-out, scatter/gather.

Parallelism buys time, then creates a consistency problem.

PATTERN 05 / Orchestrator + workers



The task graph is discovered while the system is running.

PATTERN 05 / Orchestrator + workers

ORCHESTRATOR

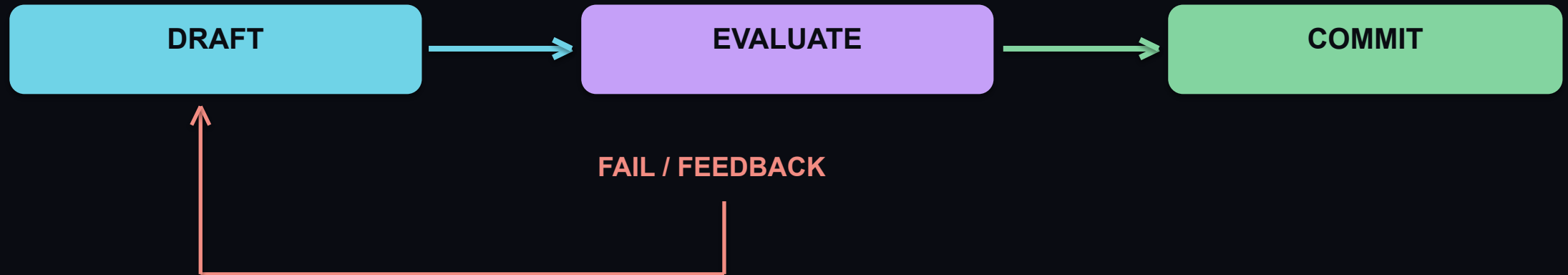
Plan · delegate
observe ·

RELIABILITY QUESTION

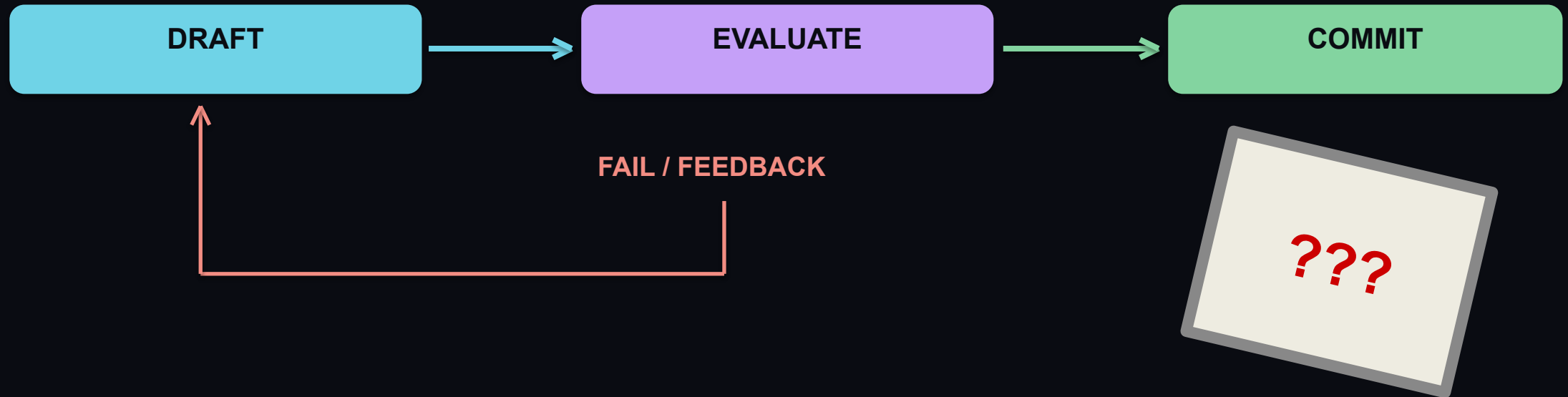
How do we cap work that did
not exist in the original plan?

The task graph is discovered while the system is running.

PATTERN 06 / Evaluate, retry, escalate



PATTERN 06 / Evaluate, retry, escalate



Max iterations · acceptance criteria · human approval for irreversible actions

A feedback loop needs a stopping condition.



EVENT DRIVEN AGENT PATTERNS

PATTERN 07 / Reactive agent consumer

Events become observations. Decisions become commands or new events.

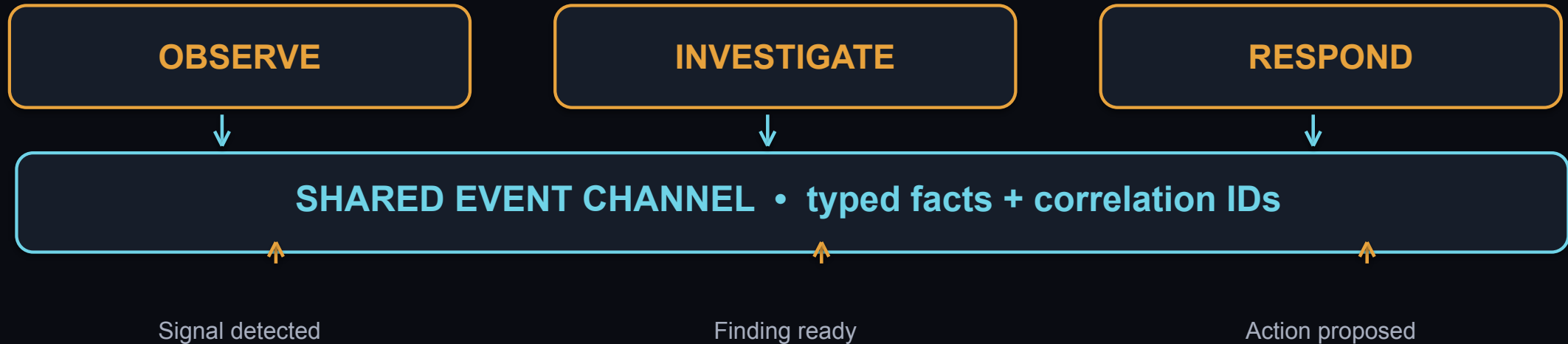


An agent is an event consumer - - - but its next action is selected at runtime.

QUESTION Which events deserve reasoning—and who is authorized to act?

PATTERN 08 / Agent choreography

No central manager. Specialized agents react to each other's events.

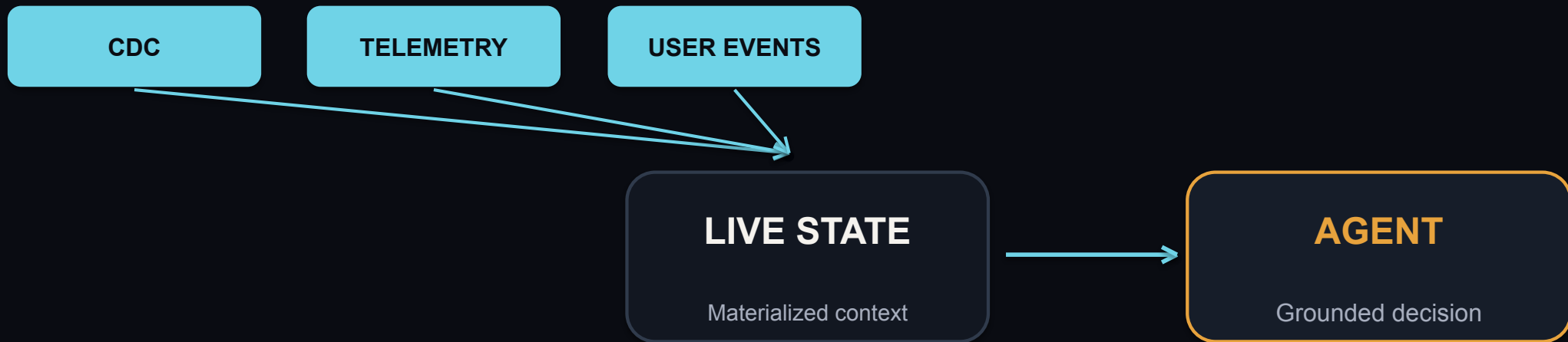


Each agent owns its capability; nobody owns the whole story by default.

RELIABILITY QUESTION What stops an event storm or cyclic delegation?

PATTERN 09 / Streaming context

The agent reasons over live state—not just the last message it received.



Events → stateful processing → fresh context → decisions

KEY DISTINCTION A durable log is history. A stateful projection is current context.

PATTERN 10 / Durable, event-waiting agents

A task can pause for hours, survive a crash, and resume on a new event.



An agent is not a process that must stay awake.

RELIABILITY QUESTION Who owns the wait, timeout, dedupe and cancellation?

PATTERN 11 / Asynchronous agent delegation

Agent-to-agent exchange is a task lifecycle.

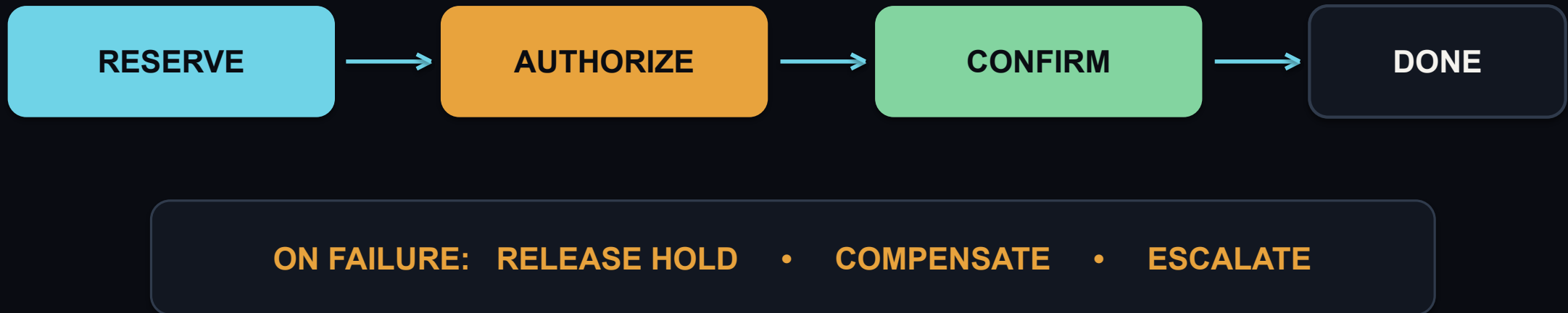


A2A-style task semantics complement durable messaging; neither replaces the other.

DESIGN CONTRACT Task ID · owner · progress · deadline · completion event

PATTERN 12 / Agentic saga

Autonomy crosses systems. Business actions do not share one transaction.



A model proposes; policy and application logic commit.

RELIABILITY QUESTION What can be safely reversed? and what cannot?

PATTERN 13 / Closed-loop autonomous control

Agents can react to the outcomes of their own previous actions.



THE DANGER Need a damping and a kill switch, without it the feedback loop is unpredictable

**Can the System
handle this?**



What changes when the consumer is an agent?

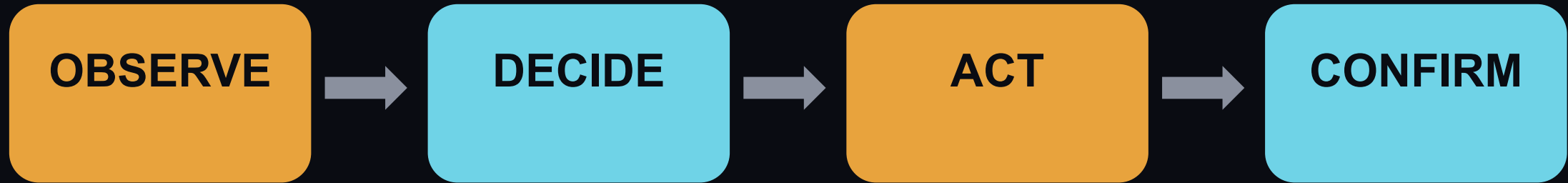
The infrastructure is familiar. The failure modes are not.

FOUNDATIONAL GUARANTEE	WHAT CHANGES WITH AGENTS	THE FLIGHT UPGRADE BOT
01 ORDERING	Events arrive in order, but decisions also depend on external observations.	Seat available when checked, gone before the bot books.
02 EXACTLY-ONCE	Processing guarantees don't automatically cover external business actions.	Booking succeeds → timeout → retry risks a second upgrade.
03 BACKPRESSURE	One message can spawn an expanding graph of tool calls, tasks, and retries.	One request fans out into searches, checks, and delegated work.
04 REPLAY	The same input may produce different decisions, or repeat side effects.	A replay must not trigger another purchase or booking.

THE SHIFT **Reliable message delivery no longer guarantees reliable outcomes.**

Accountability

THE TASK IS THE UNIT OF ACCOUNTABILITY



Trace tasks across events, tools and decisions.

RELIABILITY BOUNDARIES.

MESSAGING LAYER

Deliver and route events

Ordering · retries · flow control

AGENT / TASK LAYER

Own tasks and decisions

State · budgets · handoffs

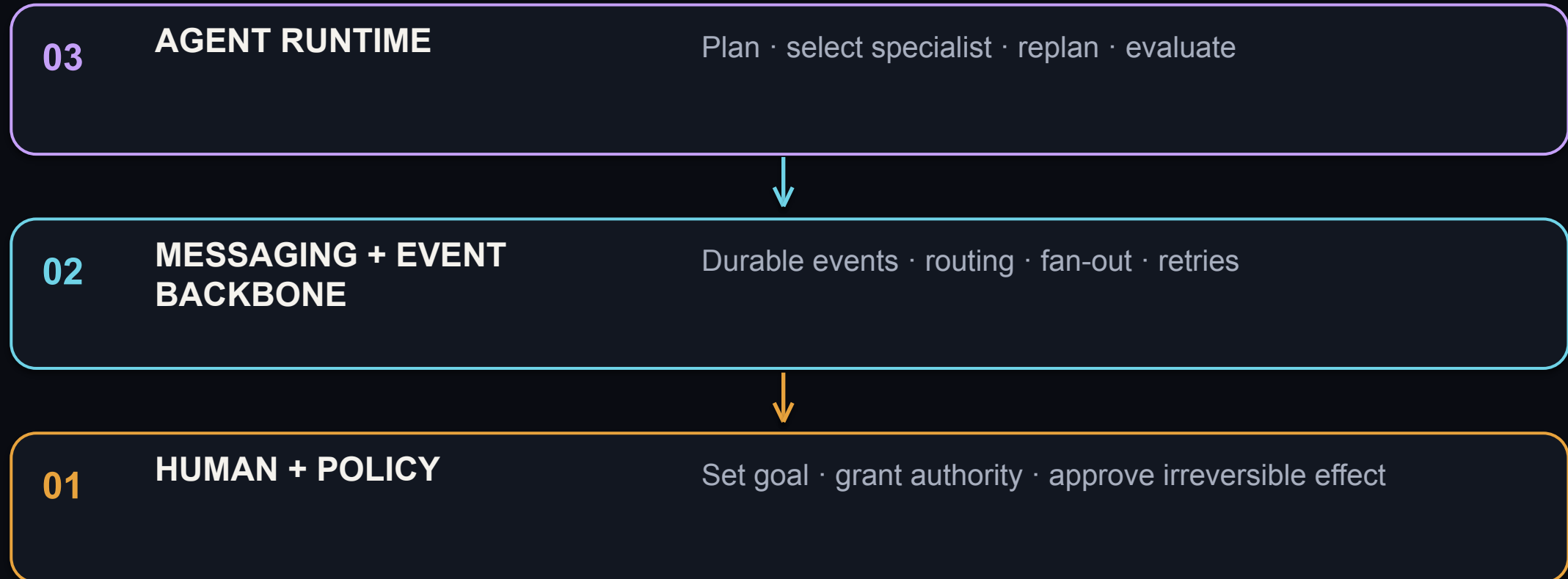
EFFECT / POLICY LAYER

Control external actions

Idempotency · approvals · audit

Tool and agent protocols complement—not replace—durable messaging.

PUT THEM TOGETHER / One durable task



ORDERING within defined scope

EXACTLY-ONCE in scoped
guarantees

BACKPRESSURE with
task quotas limiting chains

REPLAY evidence before replay
the world

THE NEW RELIABILITY MODEL

One system. **Three promises.**

01

MESSAGE DELIVERY



DELIVER THE EVENT

Events are durably delivered
and can be processed again.



02

TASK EXECUTION



OWN THE TASK

Work is tracked, recovered,
and brought to completion.



03

SAFE SIDE EFFECTS



PROTECT THE ACTION

Retries do not duplicate
external business actions.

THE SHIFT

Reliable delivery is only the beginning.

A coordination layer is taking shape.

MEDIATE

Who may act?

AUDIT

What was decided?

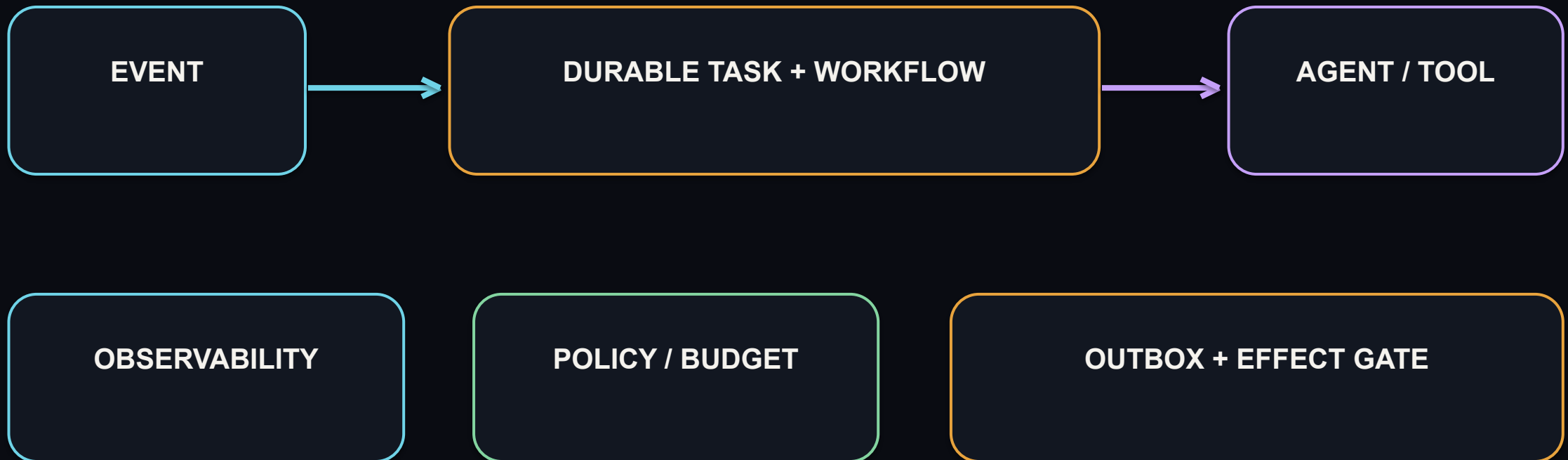
THROTTLE

What is the budget?

INTERVENE

Where can a human step in?

THE REFERENCE DESIGN / Durable autonomy



Correlate every event, decision and side effect with a task ID.

The transport stays. The trust boundary becomes explicit.

***Trace decisions,
task IDs and tool outcomes.***

***Budget the whole task,
not just the queue.***

***Gate irreversible actions;
dedupe effects.***

***Treat each agent as a principal
with least privilege.***

***Test cancel, replay
and human recovery.***

The image is a composite background. The lower half shows a curved horizon of Earth at night, with a dense network of glowing orange, blue, and purple lines representing satellite orbits or data paths. The upper half is a deep space scene with a dark blue and black background, filled with numerous stars and two prominent spiral galaxies with pinkish-purple and blue hues. The text is centered in the upper half.

This is our moment to lead.

The infrastructure we built is the foundation of autonomy.



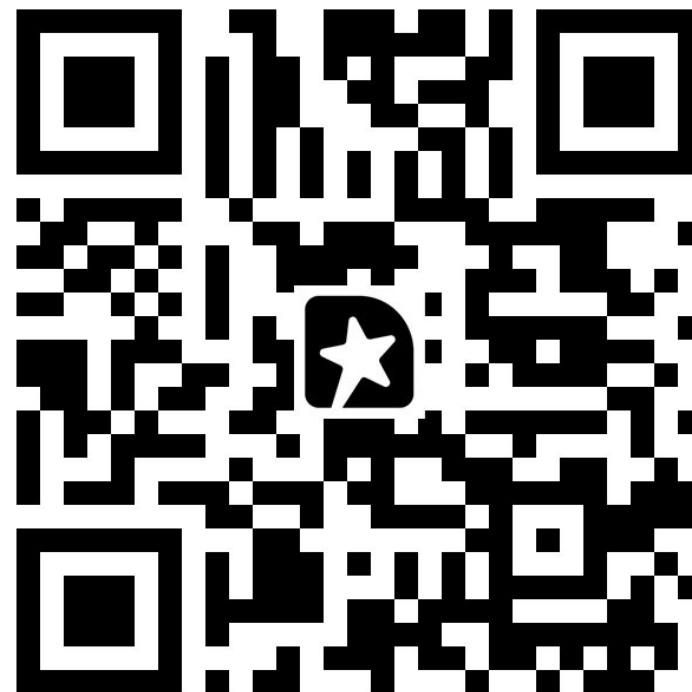
Learn More!

- <https://www.anthropic.com/engineering/building-effective-agents>
- <https://learn.microsoft.com/en-us/azure/architecture/ai-ml/guide/ai-agent-design-patterns>
- <https://docs.aws.amazon.com/prescriptive-guidance/latest/agentless/event-driven-architecture.html>
- <https://learn.microsoft.com/en-us/azure/architecture/patterns/choreography>
- <https://go.temporal.io/platform-hub/ai-engineering/ai-reference-architecture>
- <https://docs.aws.amazon.com/prescriptive-guidance/latest/agentless/prompt-chaining-saga-patterns.html>
- <https://docs.aws.amazon.com/prescriptive-guidance/latest/agentless/evaluator-reflect-refine-loop-patterns.html> ;
- <https://docs.aws.amazon.com/prescriptive-guidance/latest/agentless/observer-and-monitoring-agents.html>



Adi Polak

Data & AI @ Confluent • Keynote Speaker •
Databricks MVP • Best-Selling Author • Technic...



Keynote: When Software Starts Talking to Itself