Automated Resource Allocation Driven by Service Level Objectives

Developers define SLOs.

The infrastructure automatically adapts.



Julia Furst Morgado

Principal Developer Relations Engineer at Dash0

Pedro Célestin

Head of Platforms and Infrastructure at CLDF

The Sizing Guessing Game



Developer
no idea how much CPU and memory to ask for

Under-sized

Devs guess low ➡Workload breaks in production

Over-sized

Devs guess high ➡ Massive waste, no actual need

Devs focus on application logic, Ops focuses on saturation metrics, and nobody talks.

Who we are



CNCF Ambassador

AWS Container Hero

Docker Captain

Merge Forward Ambassador

KCD NYC and CNCF NYC Meetup Organizer

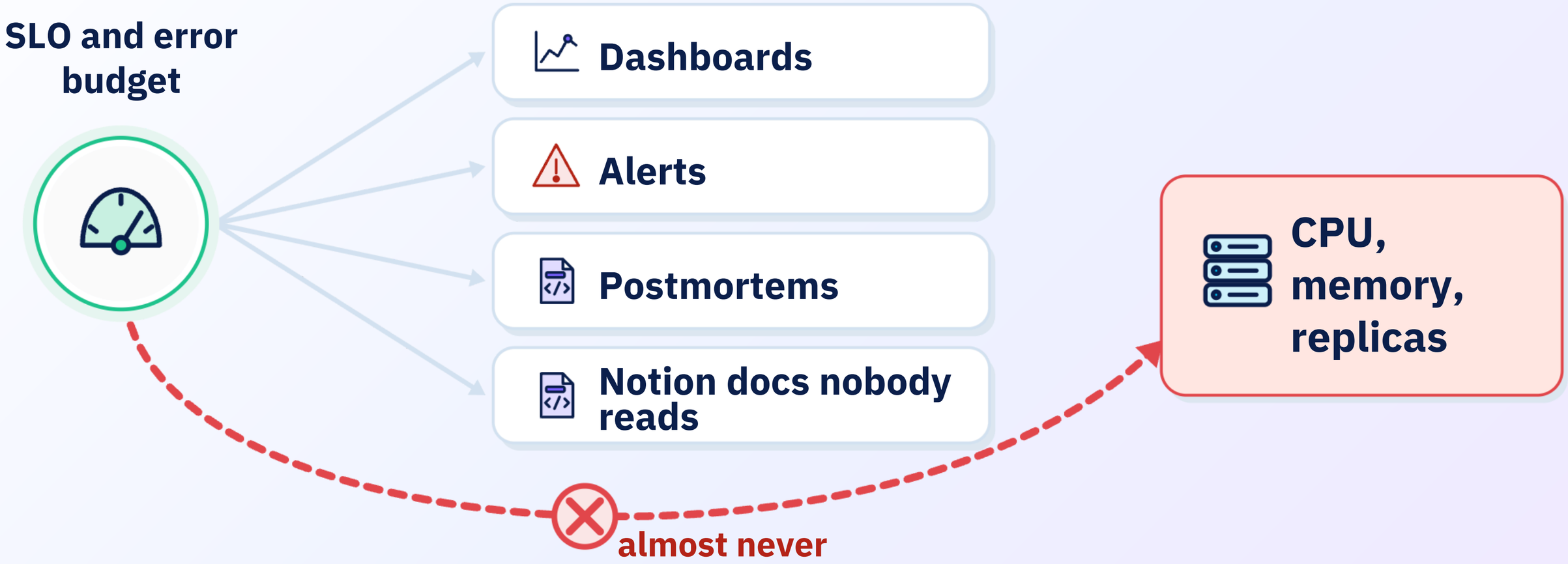


CNCF Ambassador

Golden Kubestronaut

Cloud Native Brazilia Organizer

SLOs already hold the answer



SLOs feed people and documents. They almost never drive infrastructure.

SLIs, SLOs, and the error budget

SLI

Good events over total

Availability and latency, as users see it.

SLO

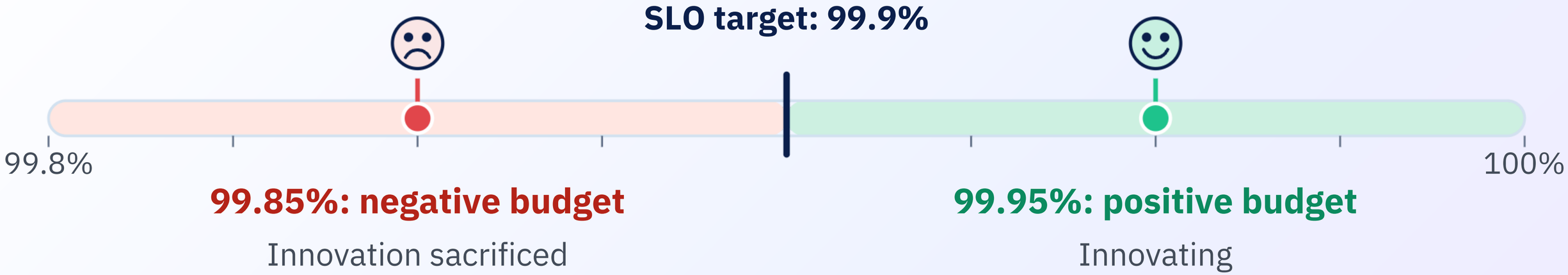
The objective

100% costs too much, so the business sets a target.

ERROR BUDGET

SLI minus SLO

How much the SLI can burn and still meet the objective.



The SLO sets the cost limit

Business impact: a missed SLO justifies more investment.

Waste: gains above the SLO have no business value.



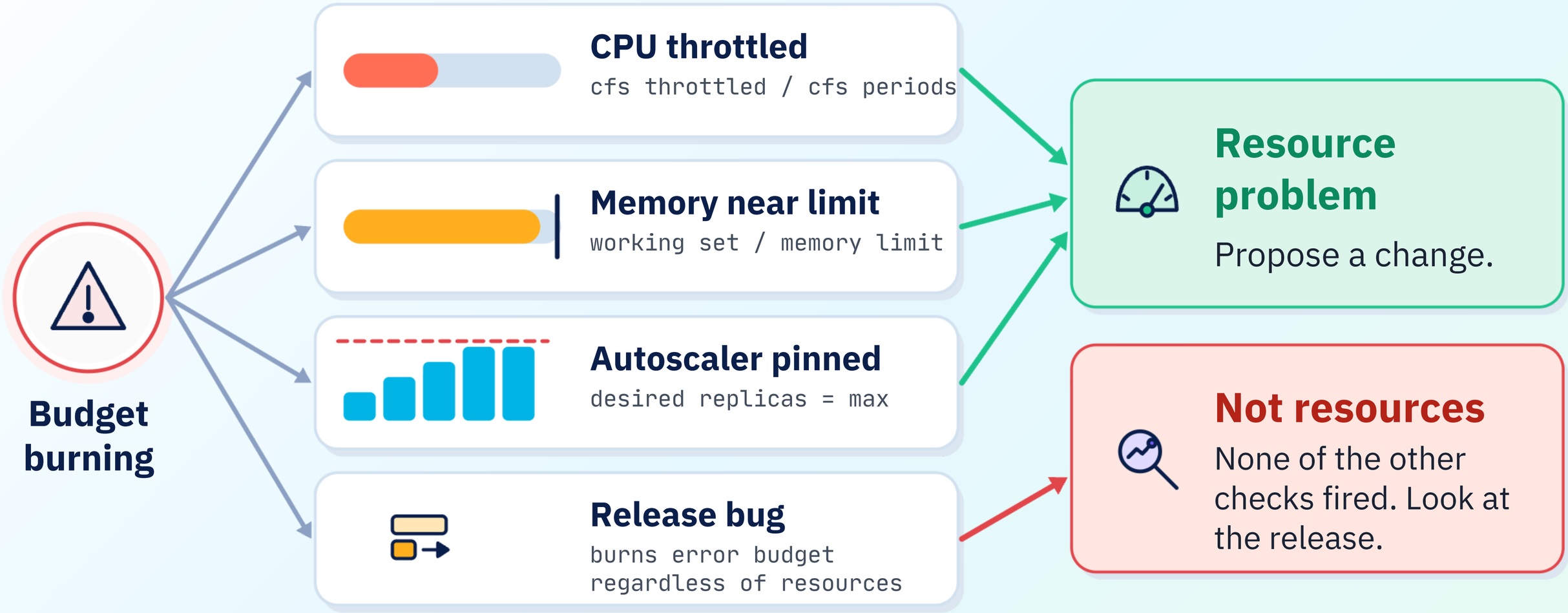
Reliability only creates business value up to your SLO.

Monitor, decide, adjust, validate



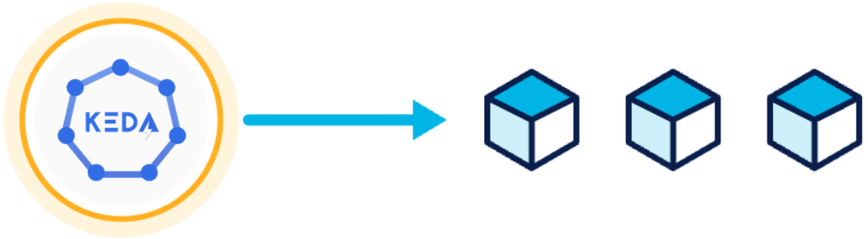
Every adjustment is a hypothesis, validated by SLI data.

Resource problem or bad release?



Two levers: replicas and requests

Horizontal: replicas



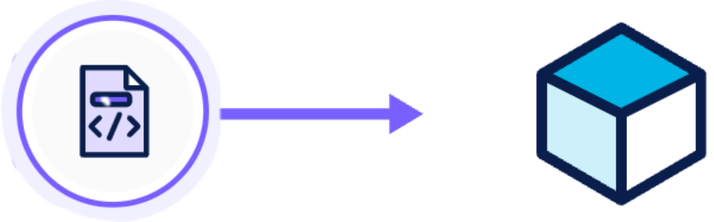
ScaledObject: min, max, threshold

KEDA creates and manages an HPA.

Pods are added or removed.

*HPA also works without keda

Vertical: requests and limits



Rollout pod template: resources

A new template starts a new ReplicaSet, which a canary can test.

Or resize in place: GA in Kubernetes 1.35.

HPA or KEDA owns replicas. The planner owns requests, in Git.

Do not run VPA in Auto mode on the same workload.

GitOps wiring: who owns which field

apps/my-service/

base/rollout.yaml	developers
base/slo.yaml	developers
tuning/scaledobject.yaml	planner
tuning/resources.patch.yaml	planner
tuning/guardrails.yaml	humans only

Argo CD ignoreDifferences

spec.replicas

Git: 3 | Live: 7 | ignored

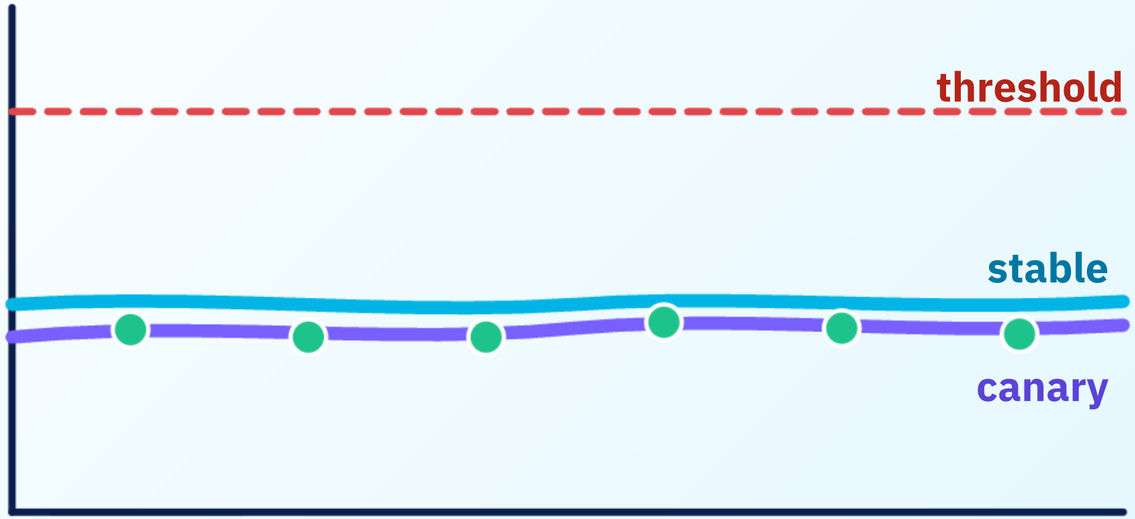
resources.requests

Git: 500m | Live: 250m | applied

Ignore replicas. Never ignore the fields the planner owns.

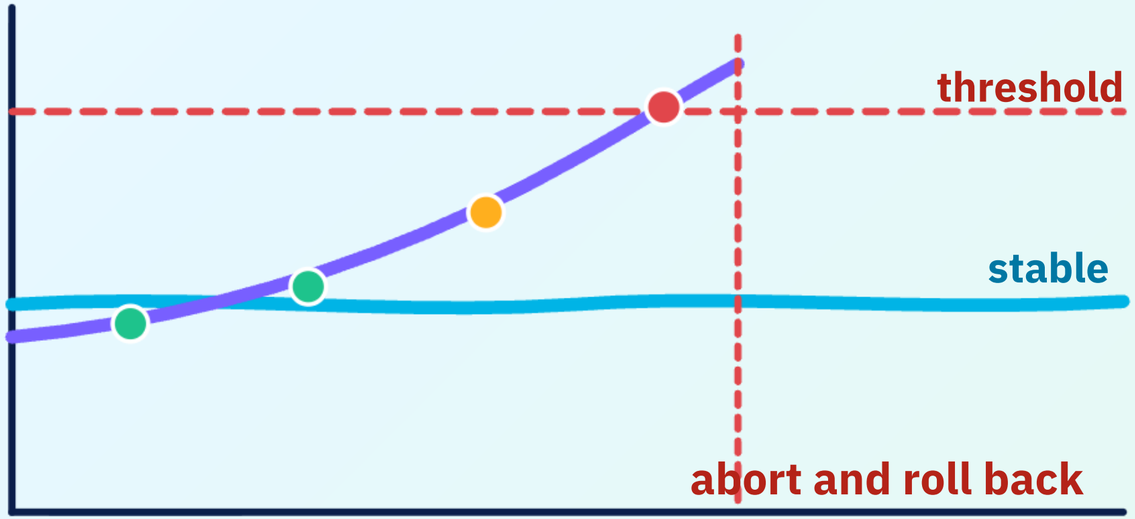
A resource canary in Argo Rollouts

Accepted: every check passes



one check every 5 minutes

Rolled back: a check fails



Analysis knobs: `interval: 5m` `count: 6` `failureLimit: 1`

A resource change is a pod template change, so a canary can test it.

The Turning Point: Why Math Wasn't Enough



Pure Math

Static formulas and linear trends.

Failed to adapt to spiky, non-linear microservice traffic.



Raw AI

LLMs reasoned across context.

But stochastic outputs led to hallucinated CPU specs and erratic scaling.



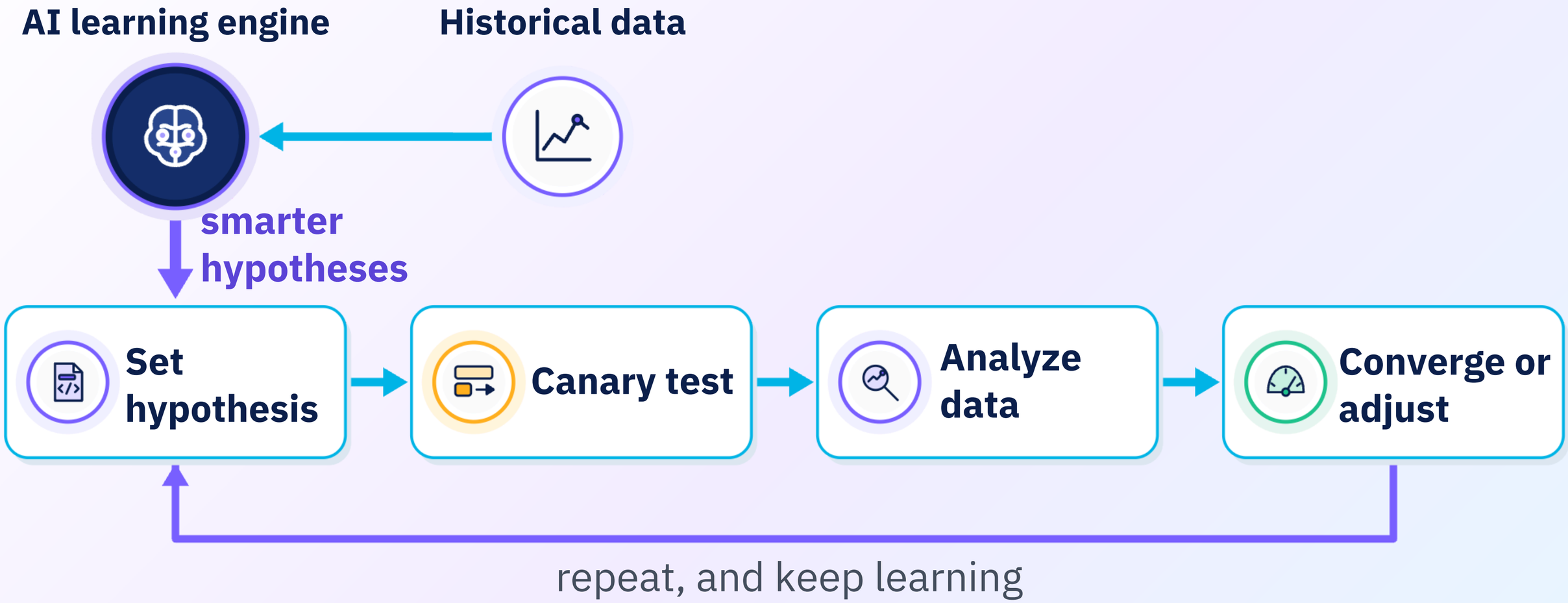
Governed AI

AI proposes hypotheses; deterministic code enforces boundaries.

Safe, adaptive scaling without direct production access.

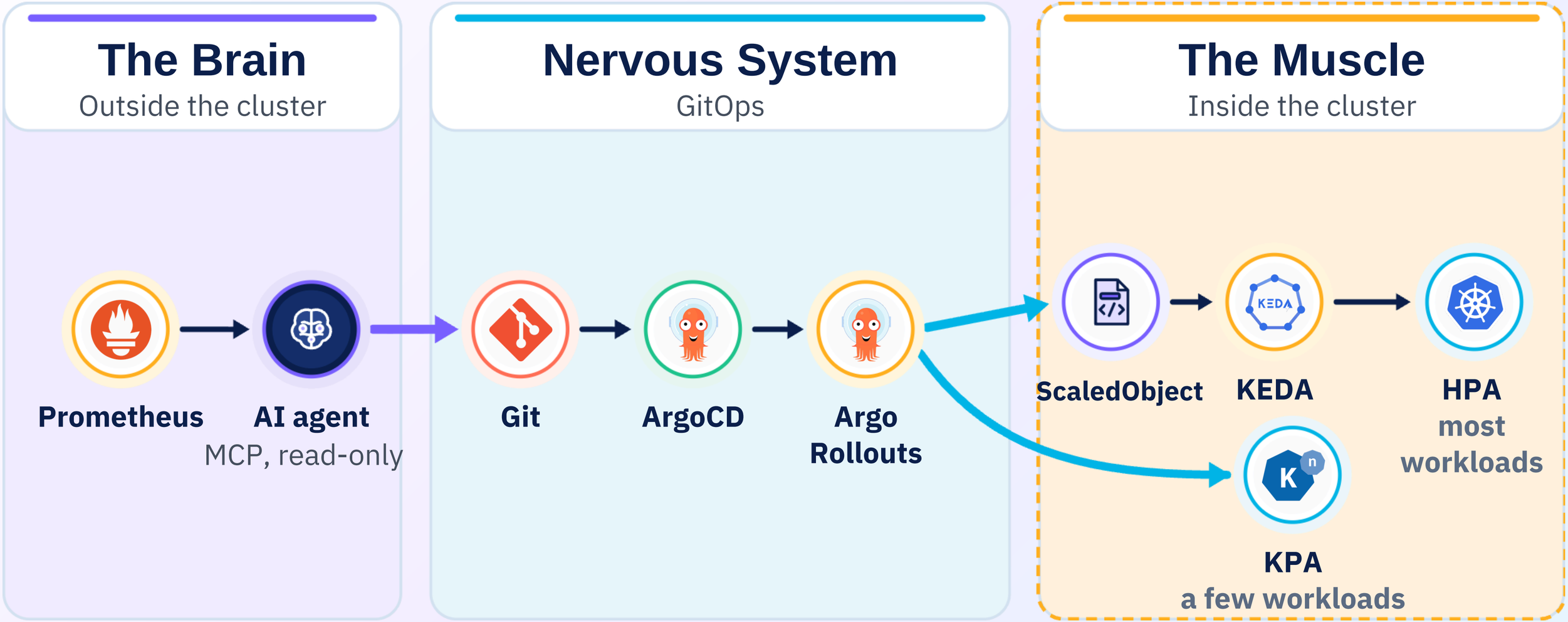
The Rule: Never let an LLM touch production directly. AI proposes; guardrails and canaries decide.

Why add AI at all?

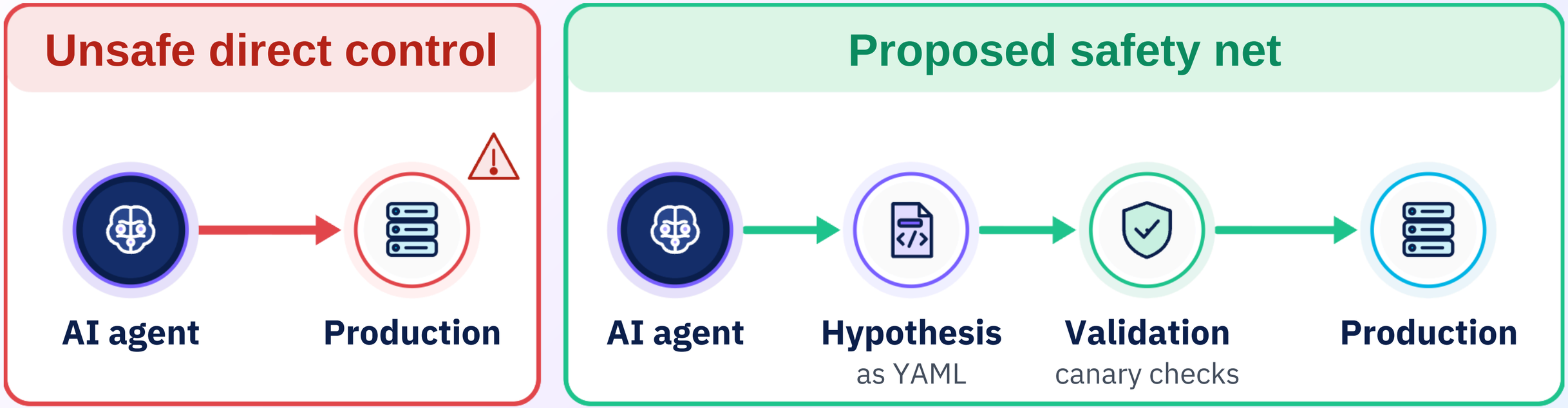


Each hypothesis is smarter, so convergence time drops as the system runs.

Brain, Nervous System, and Muscle

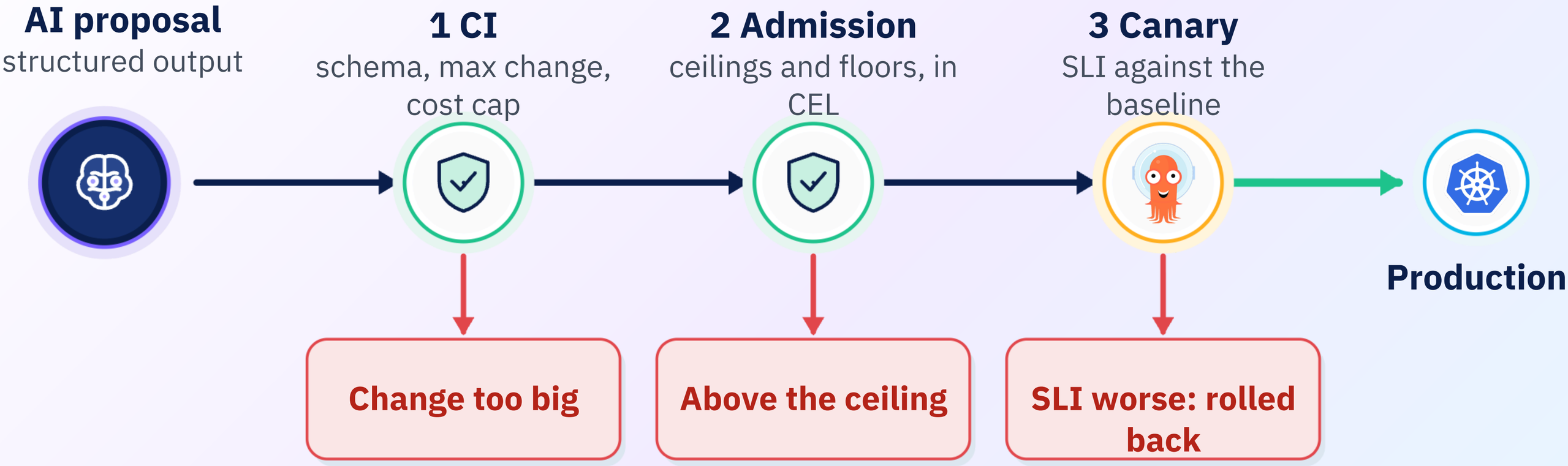


AI must not touch production directly



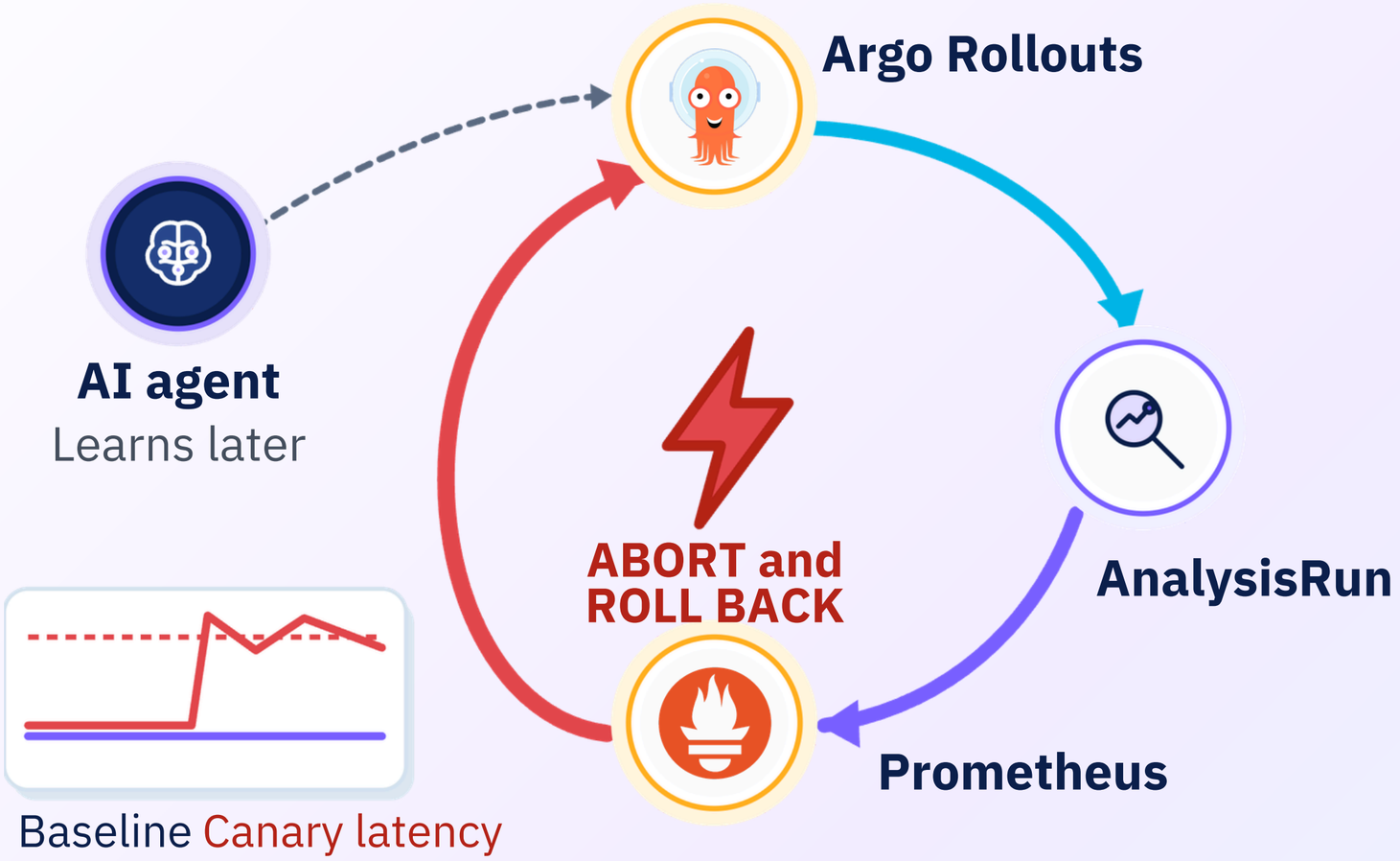
LLMs are stochastic text predictors, so raw metrics lead to hallucinated scaling. The AI proposes hypotheses. Validation decides.

Guardrails as code, in three layers



A hallucinated number has to get past all three.

If the AI is wrong, the canary aborts



Real-time protection cannot wait for AI inference.

1. The canary starts with the AI's parameters.

2. AnalysisRun polls Prometheus.

3. Worse than baseline: abort and roll back.

4. The AI logs the failure and learns.

Human guardrails, stateless scope



Humans set the limits

A maximum spend.

A hard minReplicaCount of 1 where cold starts breach the SLO.

The AI can propose changes only as pull requests.



Out of scope

Stateful workloads: databases, caches, brokers.

Node-level autoscaling and provisioning.

Designed for stateless services and async workers.

Early results

MVP Running on a few services, still calibrating.

Mistake types **few** devs copy one mistake to many workloads

Workloads hit **many**

Detects very obvious sizing mistakes.

Found absurd cases and optimized them.

Not yet: fine tuning and forecasting.

Three things to take home

1

SLOs are the north star for resource allocation.

2

AI proposes. GitOps and canaries validate.

3

Start simpler. Less complexity up front.

Pedro Célestin

Why Raw LLMs Fail at Capacity Planning (And How GitOps Fixes It), Medium, June 2026.

THANK YOU

Julia Furst Morgado

linkedin.com/in/juliafmorgado

Pedro Célestin

linkedin.com/in/pedrocrc