KUBECON · LINKEDIN

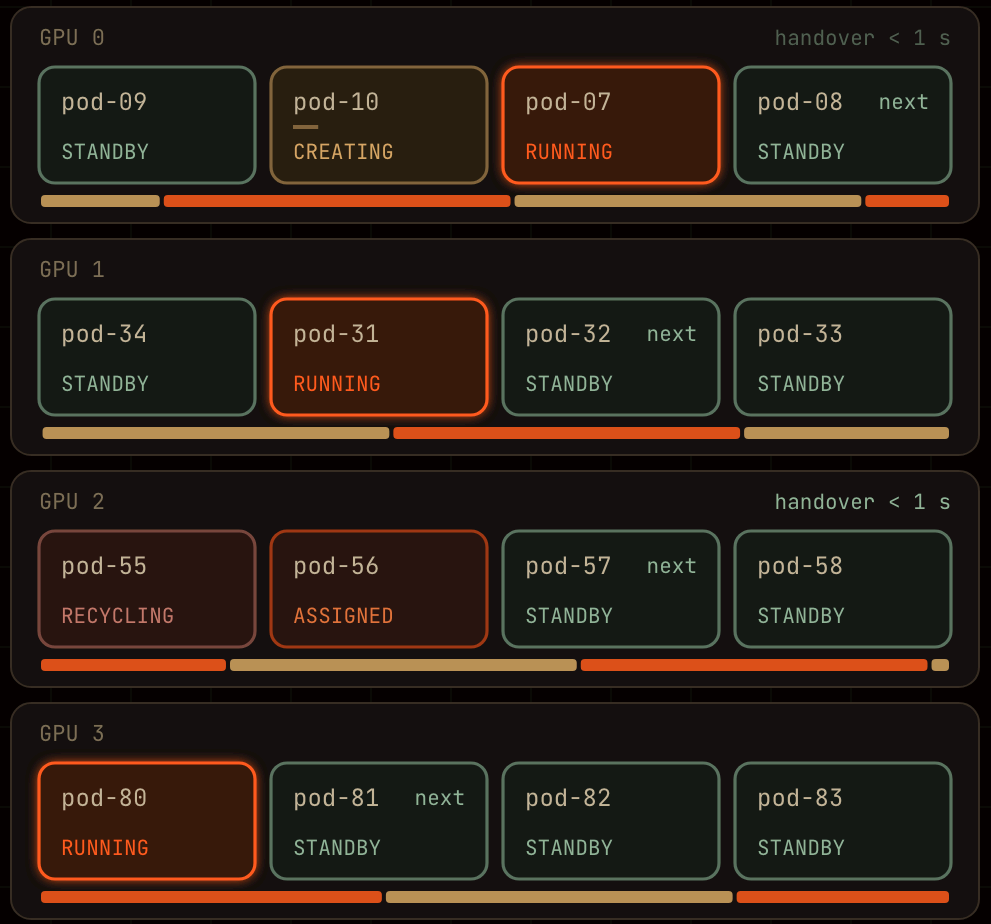
Firepool

Zero-Gap GPU Execution

Building Secure, Pre-Warmed Pod Runtimes for High-Velocity AI Agents

Ankit Goyal
Principal Staff Software Engineer, AI Platform, LinkedIn

Frank Gu
Manager, AI Platform, LinkedIn



AGENDA

What we'll cover

01 Motivation

02 How Firepool works

03 Warm pools and reuse

04 Zero gap

05 Sandbox

MOTIVATION

Our agents run thousands of short GPU jobs

Autopilot

AGENT LOOP

Migrates TensorFlow models and datasets to PyTorch, in a small loop that improves an objective on every iteration.

Autoresearch

AGENT LOOP

A user defines a research objective; the agent proposes experiments, runs them and evaluates the results.

Agentic RL

RL LOOP

Many rollouts execute in parallel inside the main training loop, behind a barrier at every iteration.

GPU JOBS OVER TIME



the same caller, again and again · around the clock



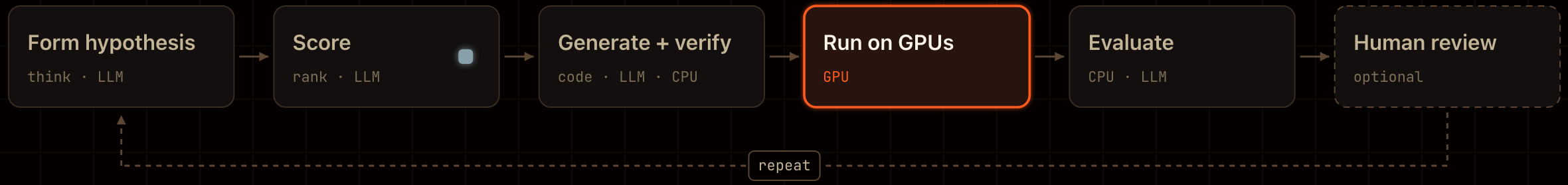
variable runtime long jobs



bursty, parallel fan-out

MOTIVATION

The GPU is a thin slice of every iteration



ONE ITERATION · WALL CLOCK



A DAY OF ITERATIONS



The GPU is busy only in the ember slivers. Everything else is an LLM, a CPU or a person.

What we tried

A · Agent on a GPU node

the GPU sits next to the agent for its whole run



\+ instant starts

\- GPU held idle through every pause

B · Agent on CPU, a GPU pod per step

create a pod whenever a step needs a GPU



\+ GPU held only while a job runs

\- a pod cold start on every step

WHAT WE TRIED

Why a fresh pod is slow

ONE GPU POD IN PRODUCTION · SUBMIT TO FIRST INSTRUCTION

MEASURED



1 scheduling + GPU binding

2 eight platform init containers

3 mounts, image check

4 pod agent ready

AGENDA

What we'll cover

01 Motivation

02 How Firepool works

03 Warm pools and reuse

04 Zero gap

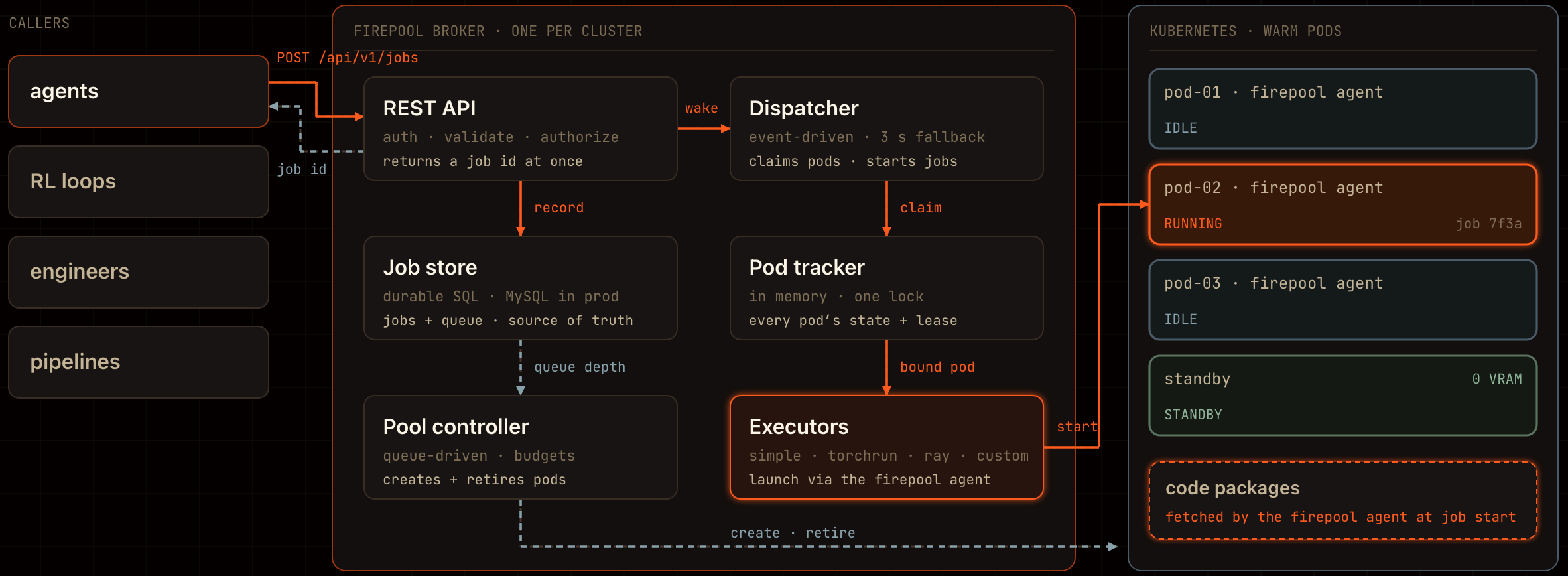
05 Sandbox

Firepool puts jobs on pods that are already warm



A broker in front of Kubernetes: each job is bound to a warm pod instead of waiting for a new one

Inside the broker: from submit to a running job



AGENDA

What we'll cover

01 Motivation

02 How Firepool works

03 Warm pools and reuse

04 Zero gap

05 Sandbox

Agent-first APIs

AGENT · SHELL

```
$ firepool submit train.py -u h200_1
[firepool] job 7f3a… submitted
  (warm_pool_unit=h200_1, executor=simple, workers=1)
$ firepool status 7f3a…
status:  RUNNING
unit:    h200_1
$ firepool logs 7f3a… --follow
step 0  loss 2.31
step 1  loss 1.87
$ firepool wait 7f3a…
status:  SUCCEEDED
```

REST · /API/V1

POST	/jobs	submit → job id
GET	/jobs/{id}	status, events, exit code
GET	/jobs/{id}/logs?follow	SSE log stream
GET	/jobs/{id}	poll until terminal
POST	/jobs/{id}/cancel	cancel

BUILT FOR PROGRAMS

- Unknown request fields are rejected, not ignored
- Every terminal failure carries an error class and exit code

HOW IT WORKS

Allocating from the warm pool

WARM POOL · H200_1 · BUILT BEFORE ANYONE ASKS

pod-00 IDLE	pod-01 IDLE	pod-02 IDLE
pod-03 IDLE	pod-04 same caller RUNNING agent-A	pod-05 IDLE

agent-A's next step reuses its own warm pod — no pod is created

WHICH POD A CLAIM GETS · ONE STEP, ONE LOCK

prefer	Same caller's idle pod	HIT
else	A never-used idle pod	clean · already running
else	Promote a zero-VRAM standby	standby pools · shares a busy GPU
else	Create a new pod	the cold path

MULTI-POD JOBS · ALL PODS OR NONE

executor: simple | torchrun | ray | custom

pod-0 RUNNING rank 0	pod-1 RUNNING rank 1	pod-2 RUNNING rank 2	pod-3 RUNNING rank 3

One claim for every pod · ranks and addresses injected at start

MEASURED no partial claims in 50 gang jobs · warm gang claim 0.27–0.45 s

HOW IT WORKS

Interchangeable pods: bind the job late



HOW IT WORKS

Reuse a warm pod only for the caller who used it

1 · VERIFIED RESET

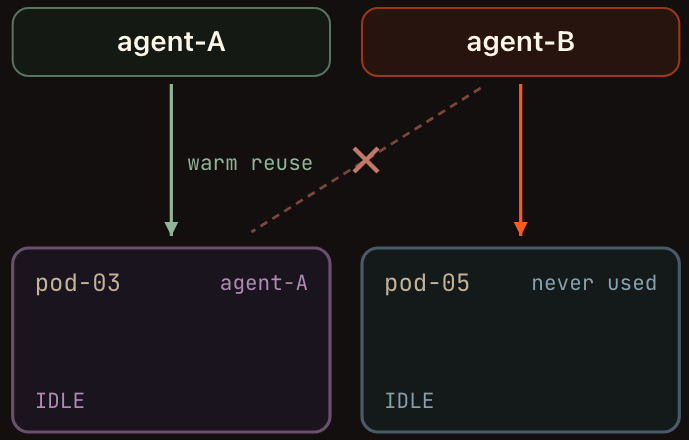
pod-03 — held for agent-A

IDLE — job done

- ✓ process tree killed
- ✓ gone from /proc
- ✓ package dir + /dev/shm emptied

reset verified

2 · OWNER-ONLY REUSE



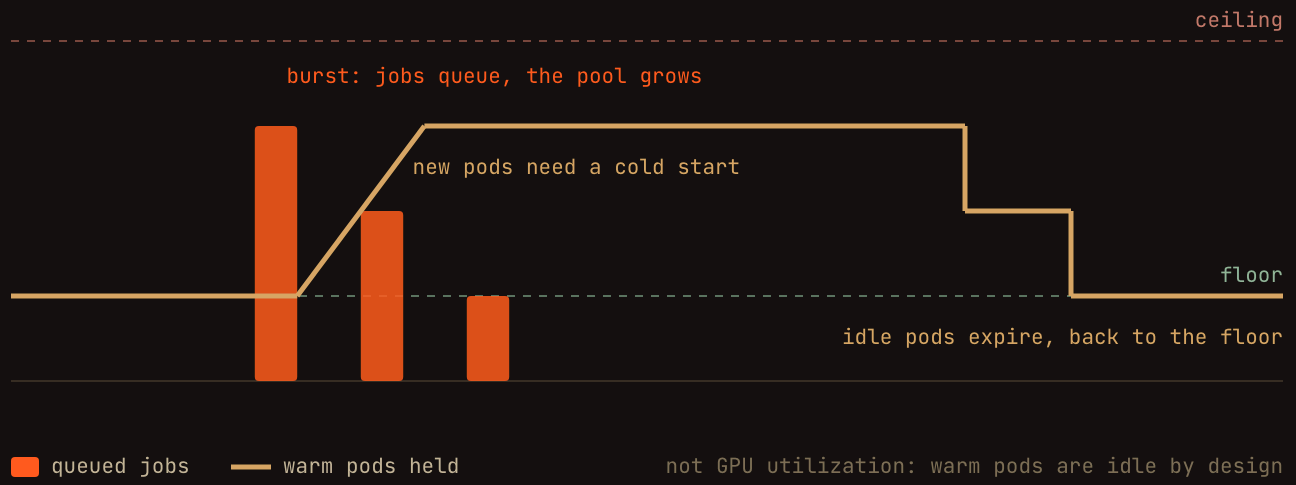
3 · RETIRE AT A REUSE-TAG BOUNDARY



HOW IT WORKS

Scale warm pools on the queue, inside GPU budgets

WARM PODS FOLLOW THE QUEUE



FOUR KNOBS PER POOL

floor
pods kept ready when nothing is queued

ceiling
the most GPUs the pool may hold

keep-alive
how long an idle pod waits for work

idle cap
idle pods kept past their keep-alive

Every pool fits inside a GPU budget.

BORROW IDLE GPUS FROM OTHER TEAMS



Use them while they are idle
advertised by the cluster's capacity controller

Give them back on demand
returned as soon as the owner reclaims its quota

AGENDA

What we'll cover

01 Motivation

02 How Firepool works

03 Warm pools and reuse

04 Zero gap

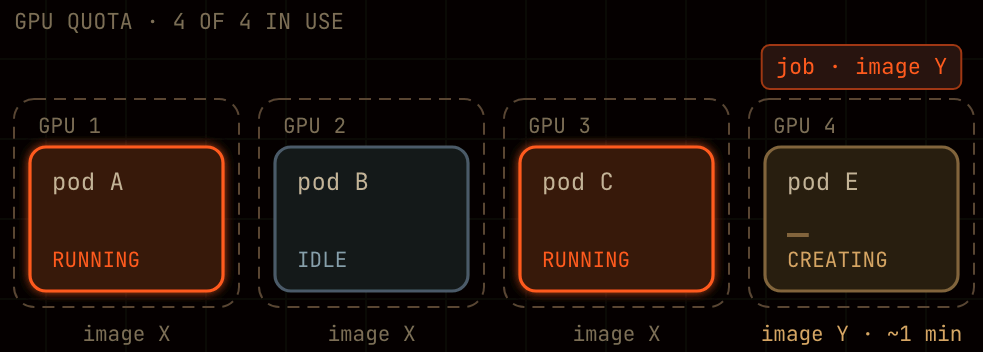
05 Sandbox

THE CATCH

GPU capacity is finite, and warm pods are expensive

1 At capacity, new pods replace old ones

When the cluster is full or a pool reaches its GPU quota, we can't just add warm pods. Each new one replaces an old one: delete, then create, about a minute on the critical path.



2 Real jobs are heterogeneous

Different images, tenants and reuse tags need different pods. Any mismatch can trigger a pod refresh, and the GPU sits idle while the job waits.



Can we keep more kinds of warm pods without more GPUs?

Zero Gap

1. **Create scheduling slots**
   HAMi + Volcano · 4 slots per card

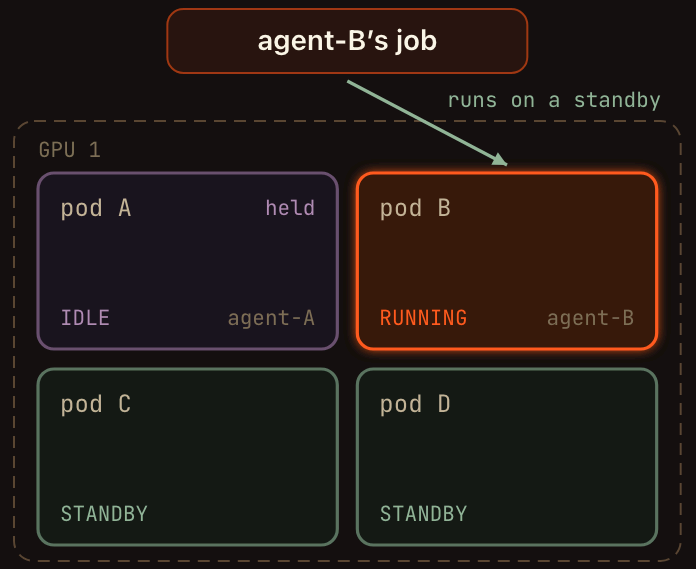
2. **Park zero-VRAM standbys**
   no CUDA context · no GPU memory

3. **Hand over on the next claim**
   B gets the whole card · A stays held



What zero gap buys

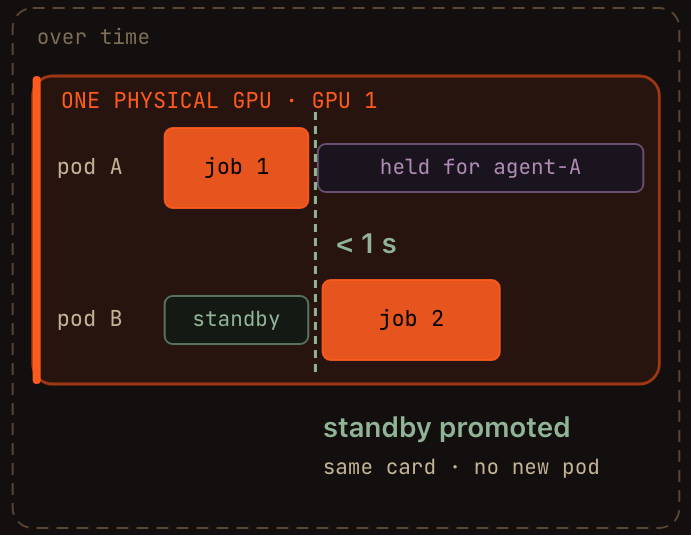
1 · HELD PODS SHARE THE CARD



Keep A warm, run B

one job runs on the card at a time

2 · NO GAP BETWEEN TENANTS OR TAGS



Median wait: 51 s → 0.94 s

H200 benchmark · a new reuse tag every job

3 · MANY IMAGES, ONE GPU

FUTURE



Keep several images warm on one GPU

not shipped today

AGENDA

What we'll cover

01 Motivation

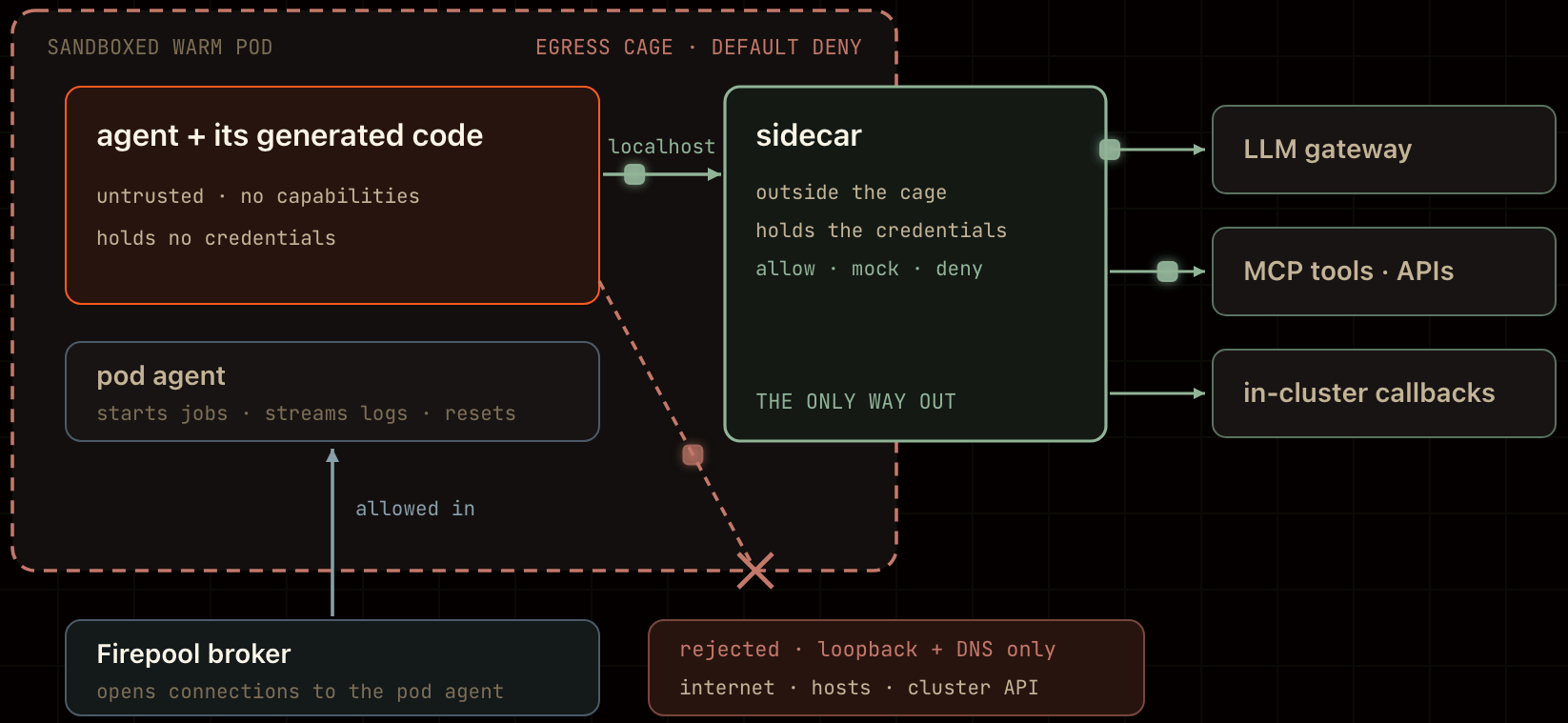
02 How Firepool works

03 Warm pools and reuse

04 Zero gap

05 Sandbox

Firepool pods run in a Sandbox



SANDBOX RULES

Least privilege
no capabilities

Deny egress by default
loopback + DNS only

No credentials in the agent
the sidecar attaches them

Policy on every call
allow · mock · deny

Any pool can be sandboxed
set per pool · annotations + a webhook

RESULTS

MEASURED

What warm pods buy

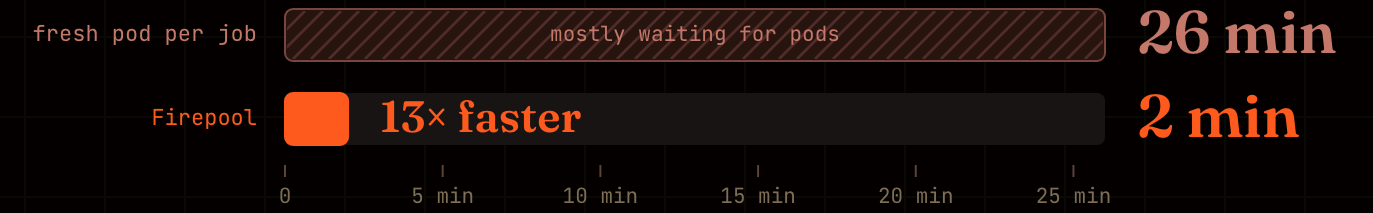
Code starts in half a second

Production medians · from getting a pod to the job's first instruction



Short agent loops finish sooner

Single-node H200 benchmark · fixed 0.5–4 s jobs · cached image · two-GPU cap



Most requests get a ready pod

Production · 3,206 pod requests · sampled 14-hour peak · warm includes standbys



OPEN SOURCE · SOON

Firepool is going open source

We’re looking for collaborators

- Run agents or RL loops on Kubernetes GPUs? Try it early.
- Work with us on warm pools, zero gap and sandboxing.
- Tell us what your workloads need.

Thank you Questions?

CONNECT ON LINKEDIN



Ankit Goyal
Principal Staff Engineer
AI Platform, LinkedIn

linkedin.com/in/goyalankit3

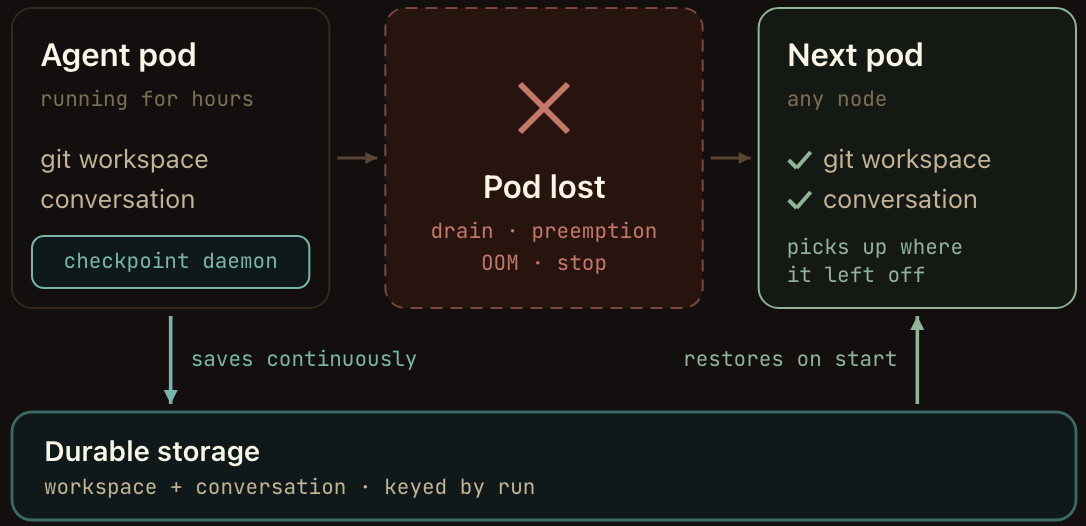


Cong (Frank) Gu
Manager
AI Platform, LinkedIn

linkedin.com/in/frank-cong-gu-1059838b

Agents survive pod loss, and learn across runs

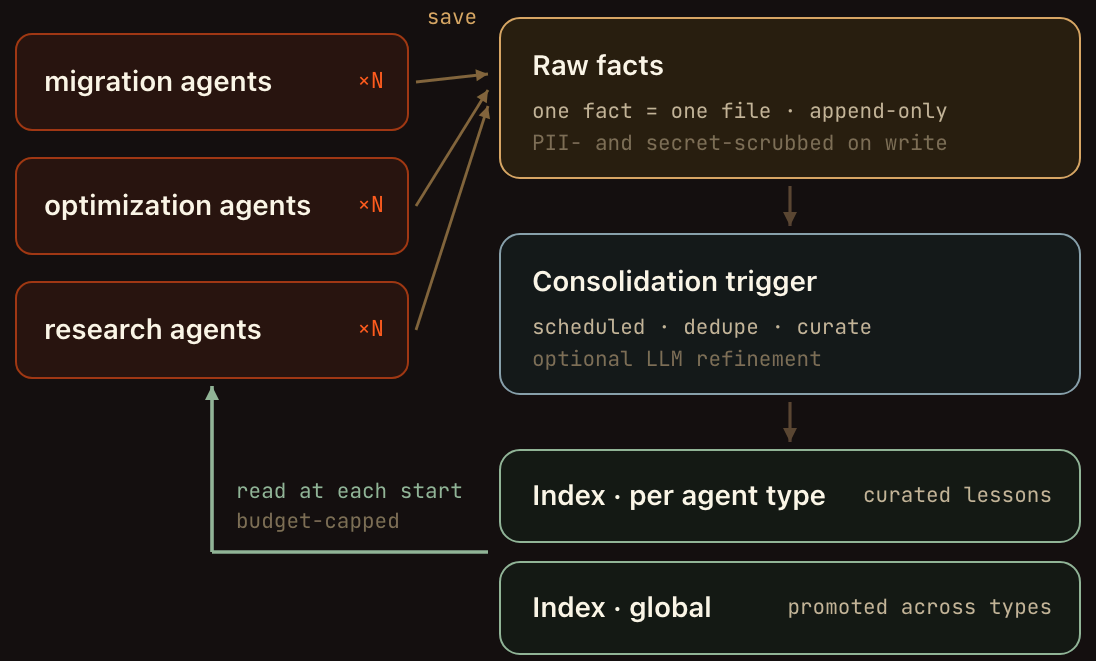
FAULT-TOLERANT PERSISTENCE



RESUMED WITH A RECOVERY PROMPT

"Re-run your setup, check for Firepool GPU jobs still running, then carry on."

CROSS-RUN AGENT MEMORY



Each new session starts from what earlier runs learned.