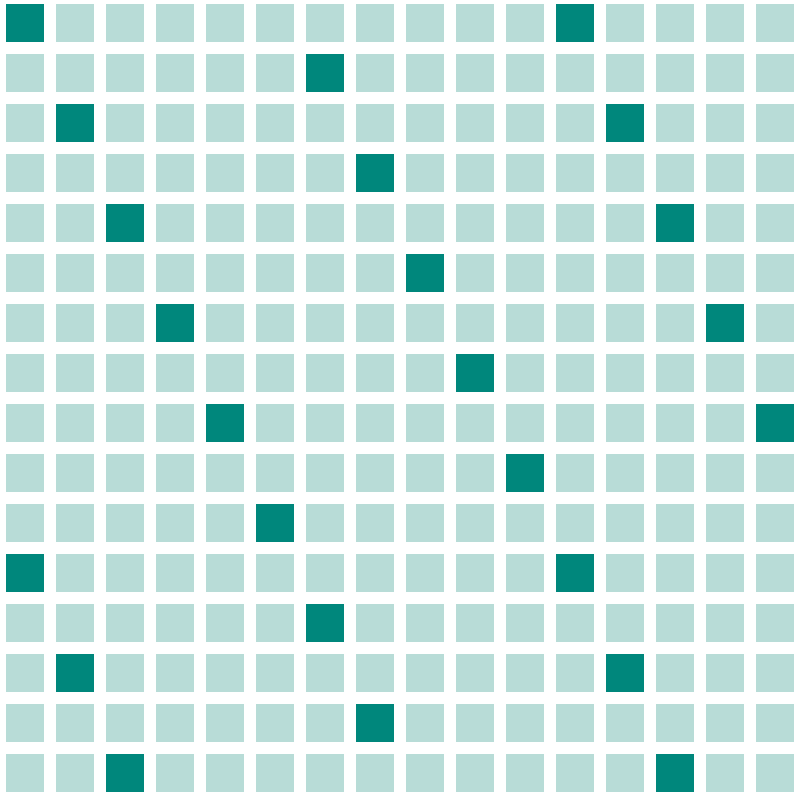
Fine-Tuning a Vision Language Model Across Hundreds of GPUs on Kubernetes

Daniel Bichuetti · Forlex
Phelipe Fabres · AWS



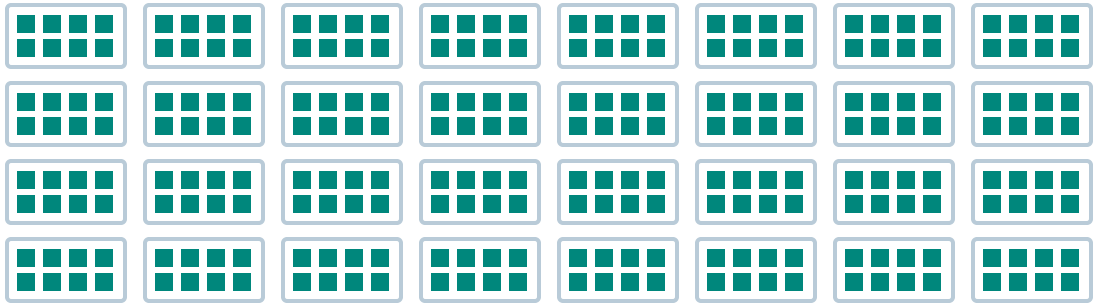
256 GPUs

256 GPUs, one coordinated training job

256

B200 GPUs

32 nodes, 8 GPUs per node



Document VLM pipeline

Pages, images and text

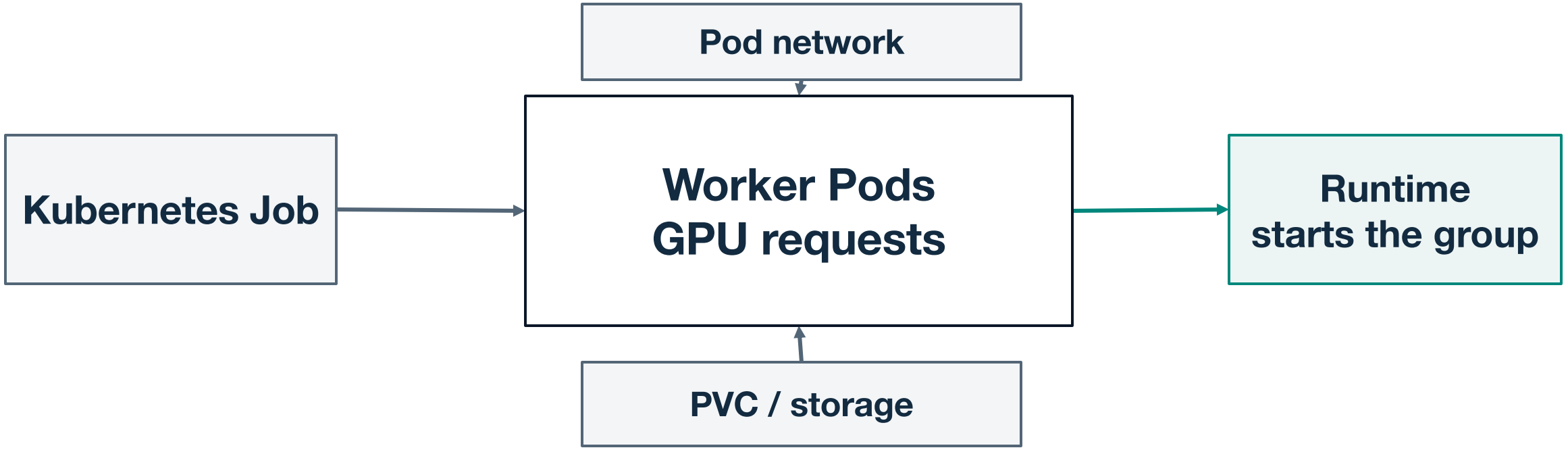
50–80 TB
Document corpus

Preparation and training run at different rates

A useful step depends on the whole group

Four contracts organize the Kubernetes stack

Jobs, resource requests, networking and storage become one operating path



ADMIT **COMMUNICATE** **FEED** **RECOVER**

One pending worker can hold the group

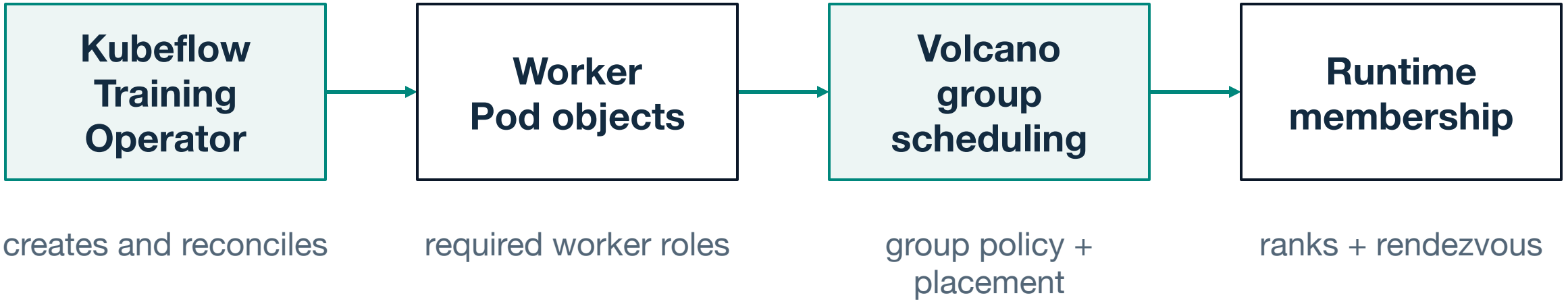
Fixed-group example: 32 required workers

w00	w01	w02	w03	w04	w05	w06	w07
w08	w09	w10	w11	w12	w13	w14	w15
w16	w17	w18	w19	w20	w21	w22	w23
w24	w25	w26	w27	w28	w29	w30	pending

Rendezvous waits for the required membership

Kubeflow, Volcano and the training runtime

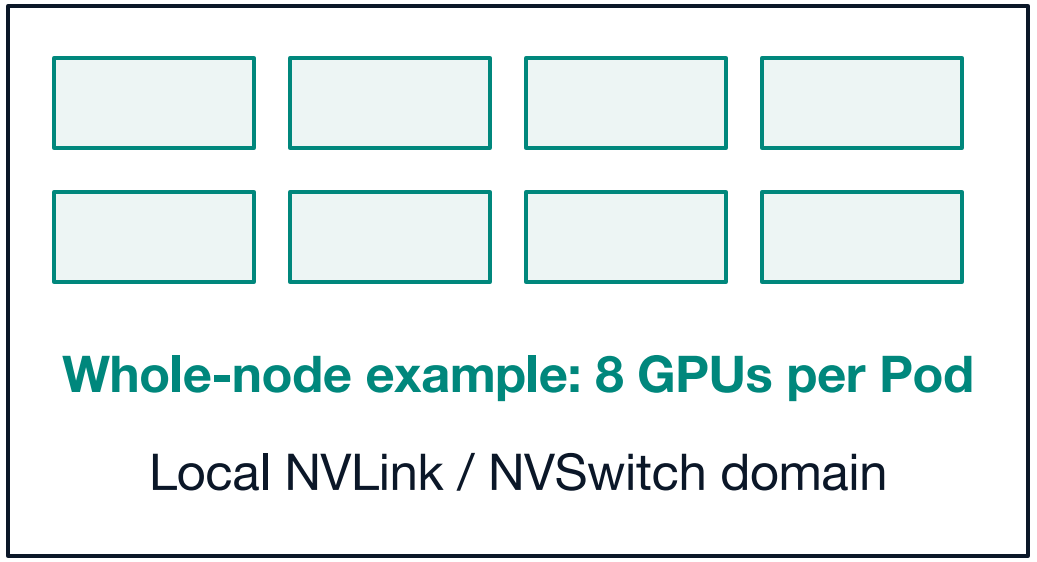
Worker ownership, admission and membership



One controller owns worker lifecycle and retry policy

Whole-node allocation and fabric placement

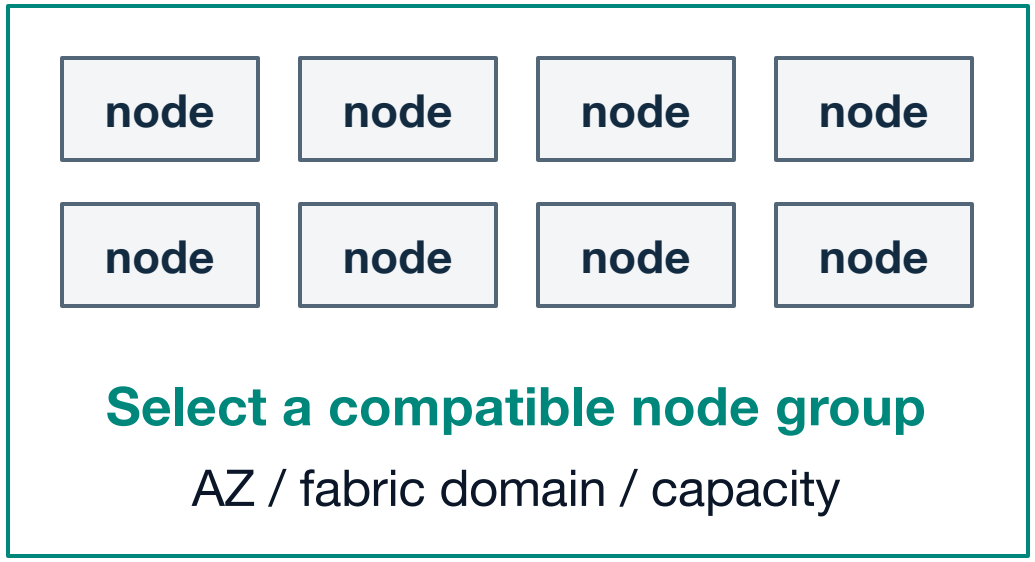
Within one node



Whole-node example: 8 GPUs per Pod

Local NVLink / NVSwitch domain

Between nodes



Select a compatible node group

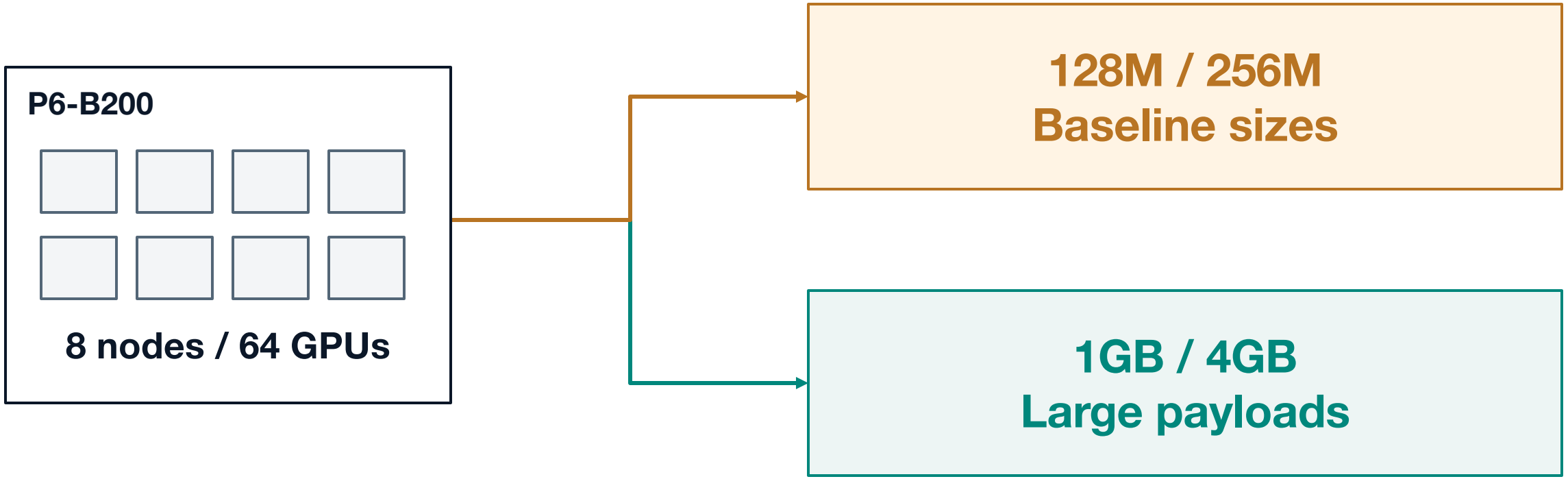
AZ / fabric domain / capacity

CPU / GPU / NIC alignment

Scheduler selects nodes; runtime maps ranks

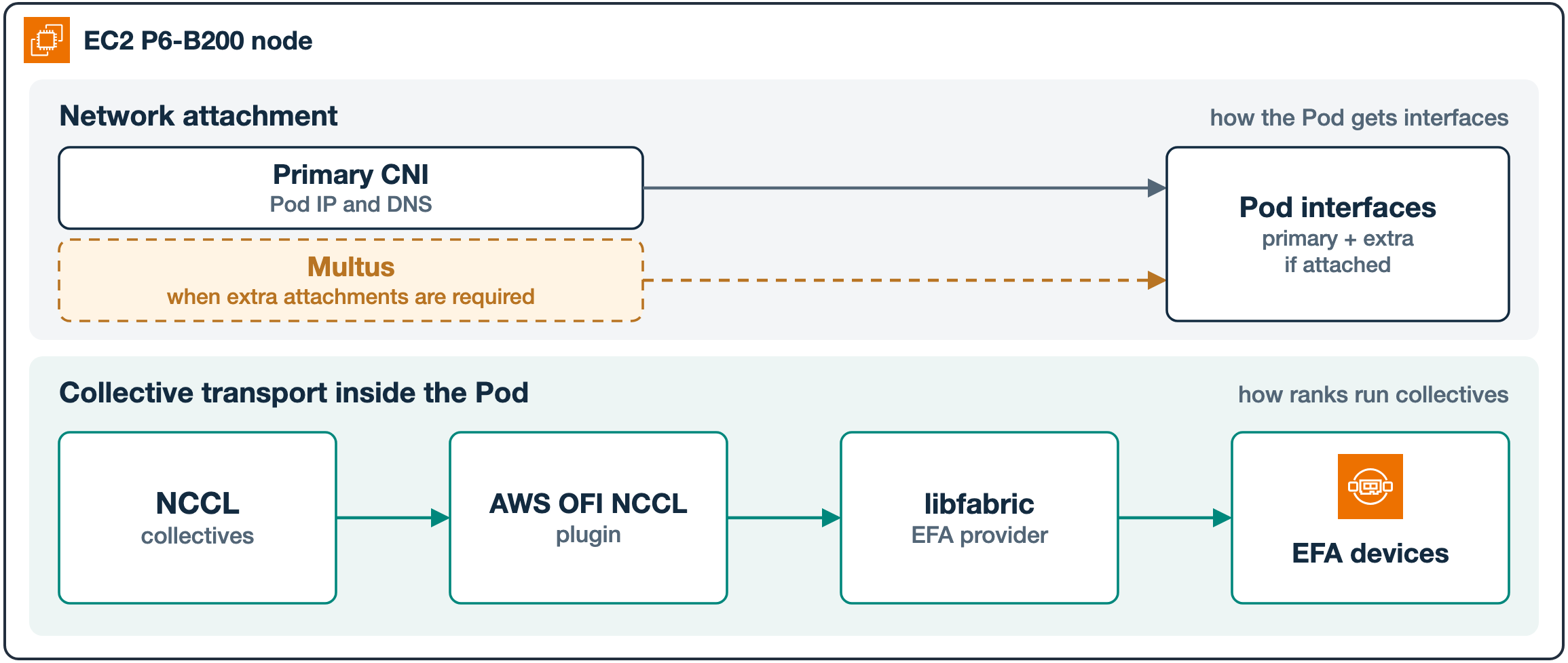
Message size changes the communication test

April network investigation: 8 P6-B200 nodes / 64 GPUs



Test representative payloads, then measure the application step

Multus and the EFA stack solve different layers



Fabric qualification inside the workload Pod



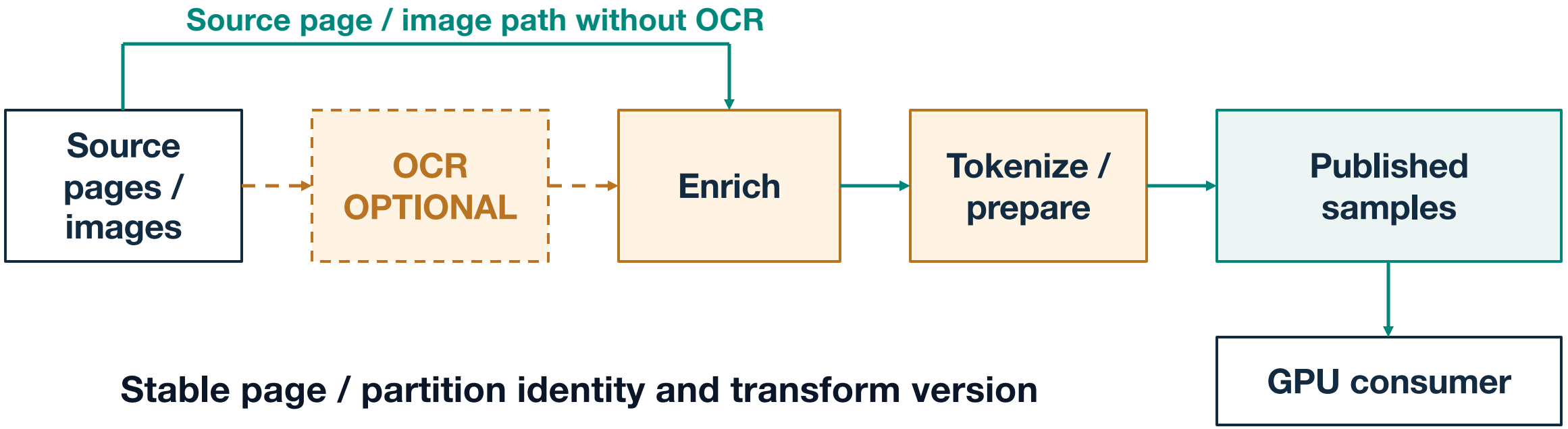
Devices and libraries

Selected provider

Collective + application step

Same image. Same placement. Representative payloads.

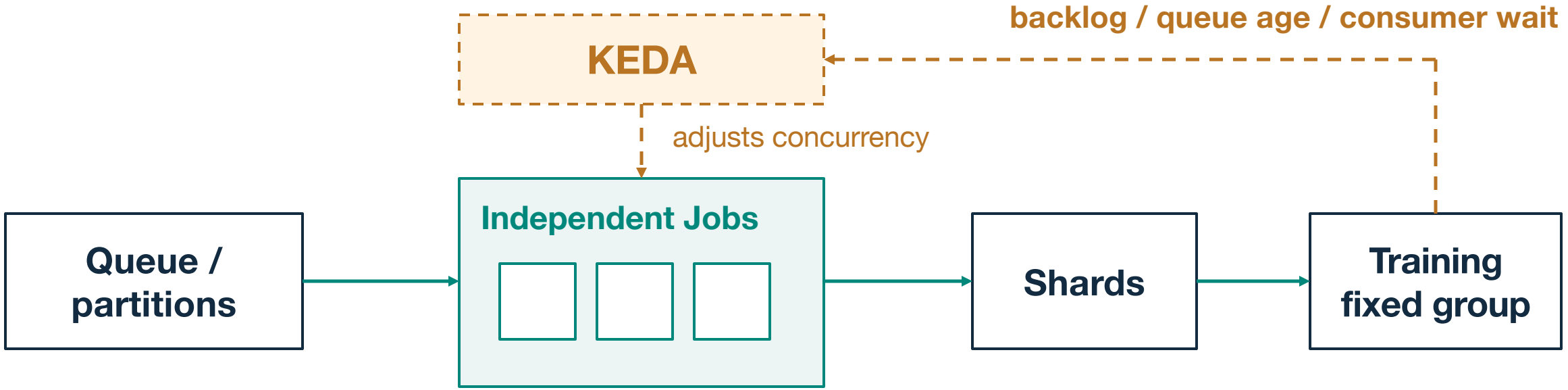
Independent Jobs prepare the VLM input



Stable page / partition identity and transform version

Publish complete, versioned samples before the consumer reads them

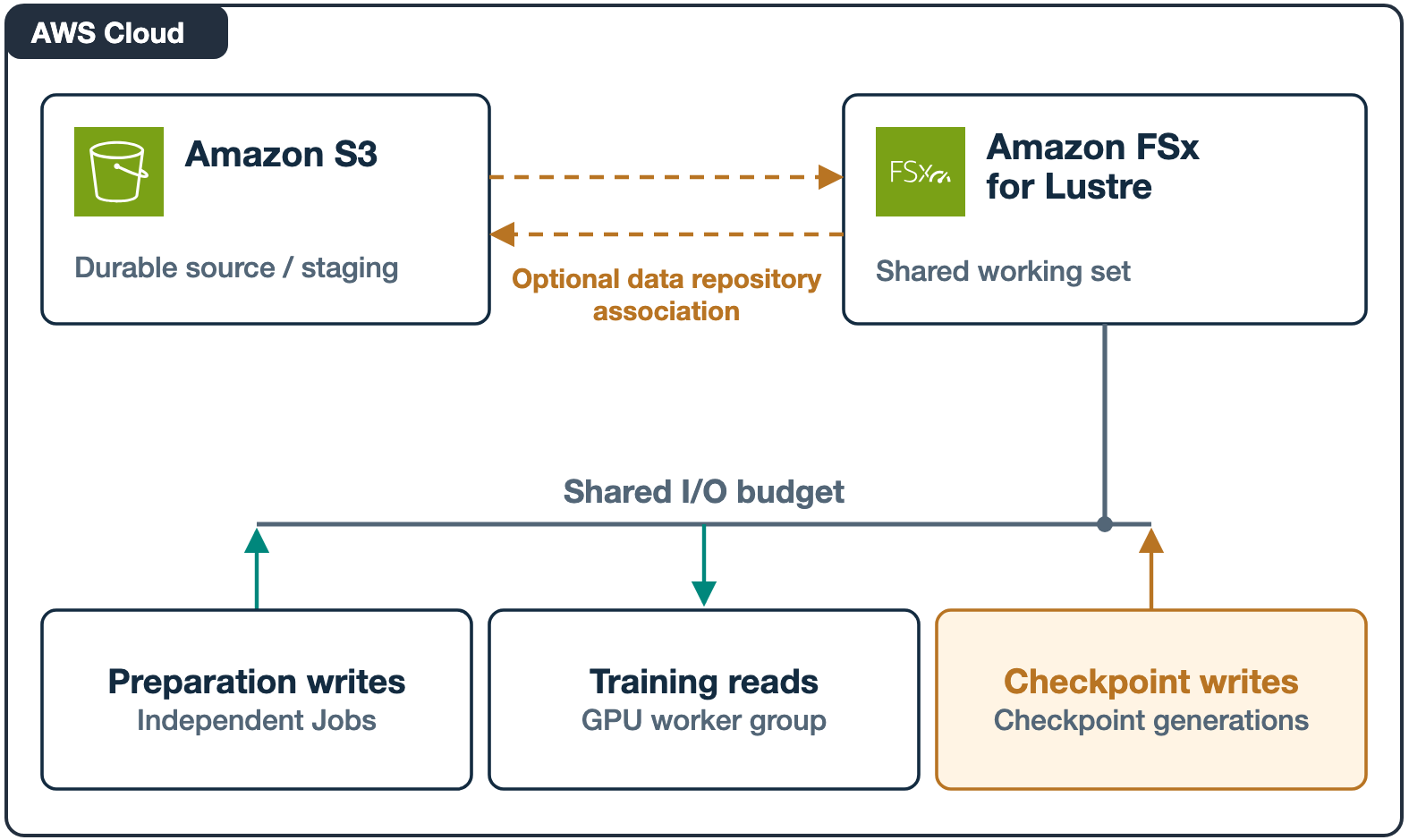
KEDA scales independent preparation Jobs



Bounded cache and idempotent publication

Preparation concurrency follows the I/O budget

Input and checkpoints share an I/O budget



Three sizing questions

Capacity

Working set + generations

Bandwidth

Overlapping reads and writes

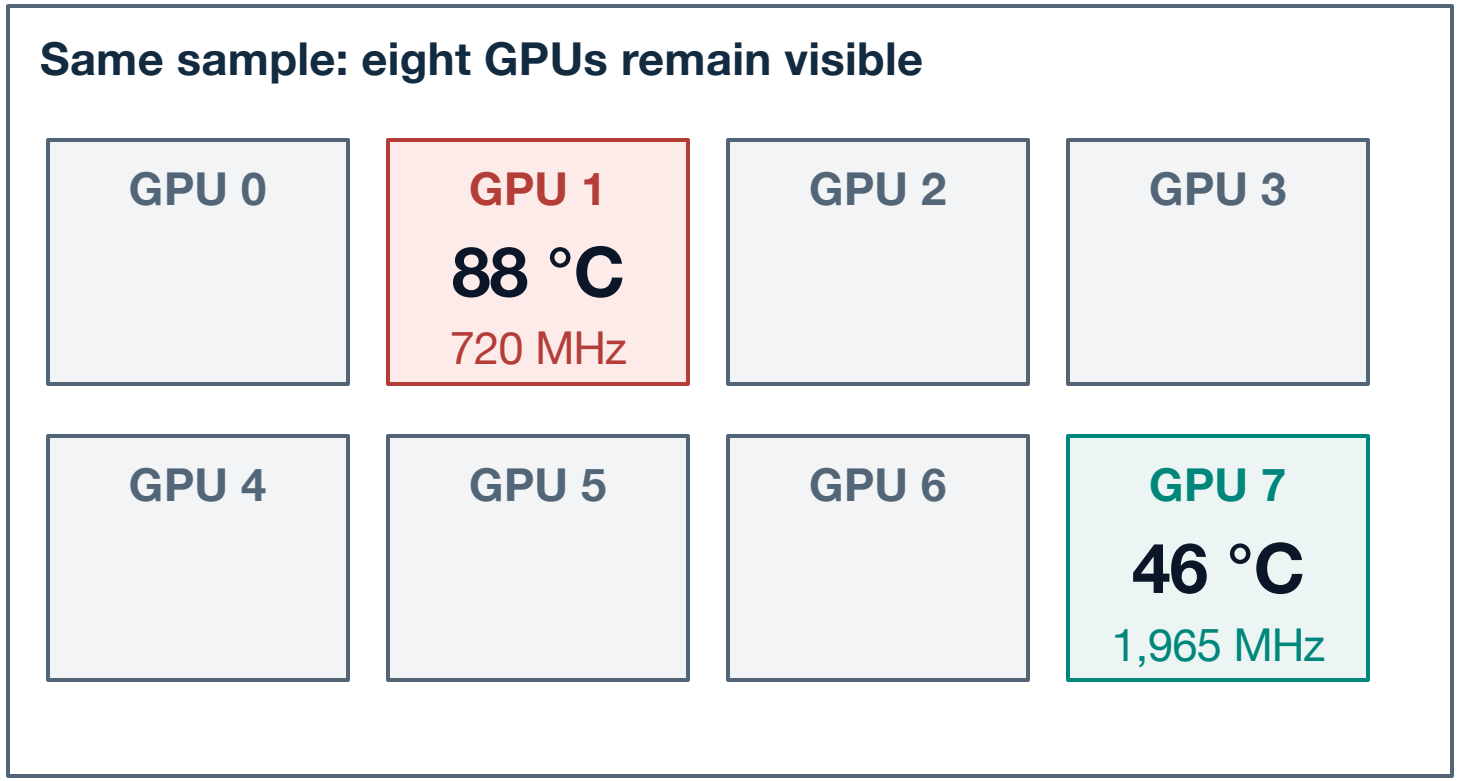
Concurrency

Useful input delivered

Tune concurrency while input reads and checkpoint writes overlap

GPU health can diverge from EC2 health

August thermal incident

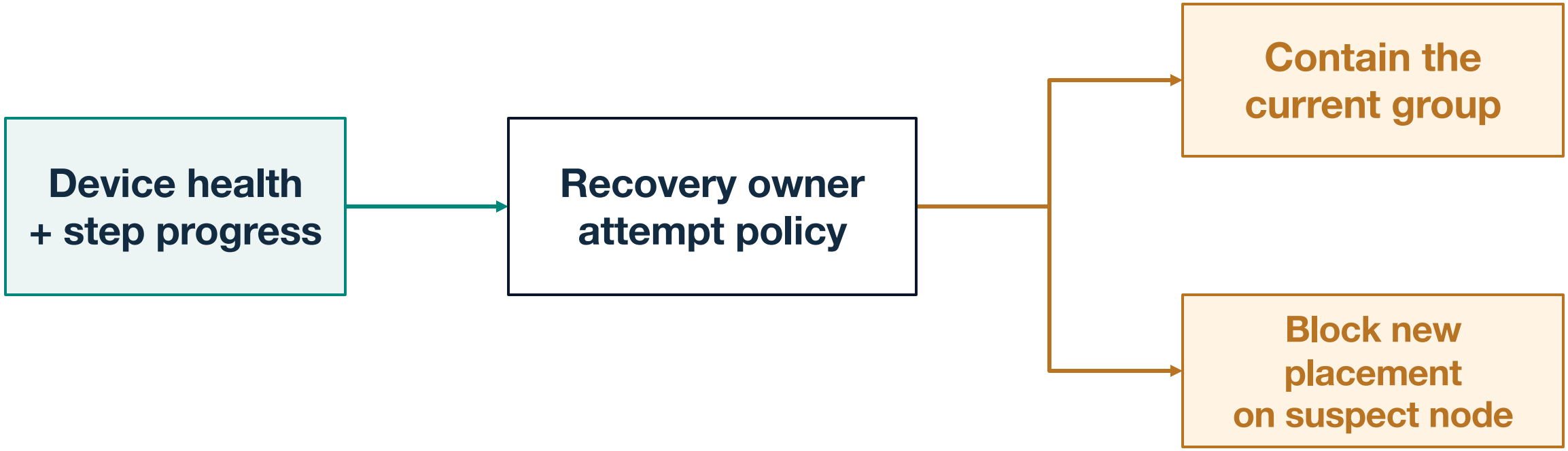


EC2 checks passed

Node removed from trusted capacity

Node eligibility includes accelerator health and application progress

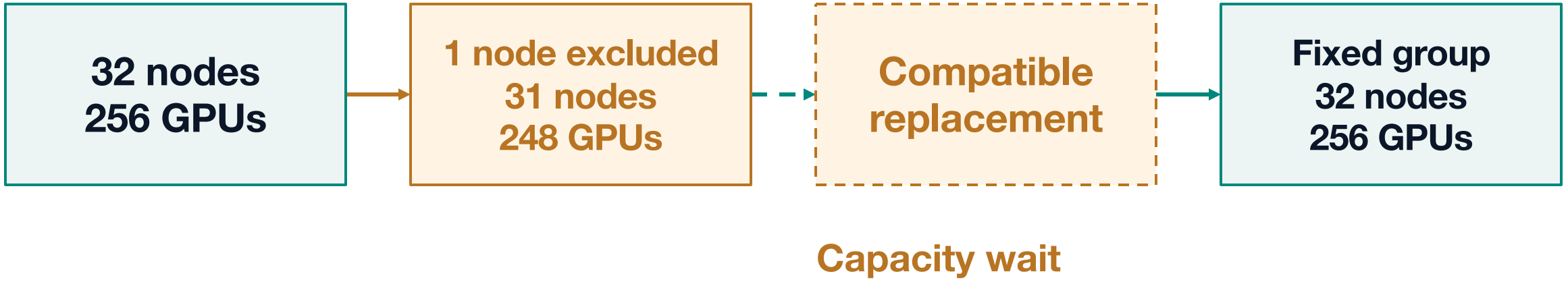
Health signals need a recovery owner



NoSchedule blocks new placement; containment stops the current attempt

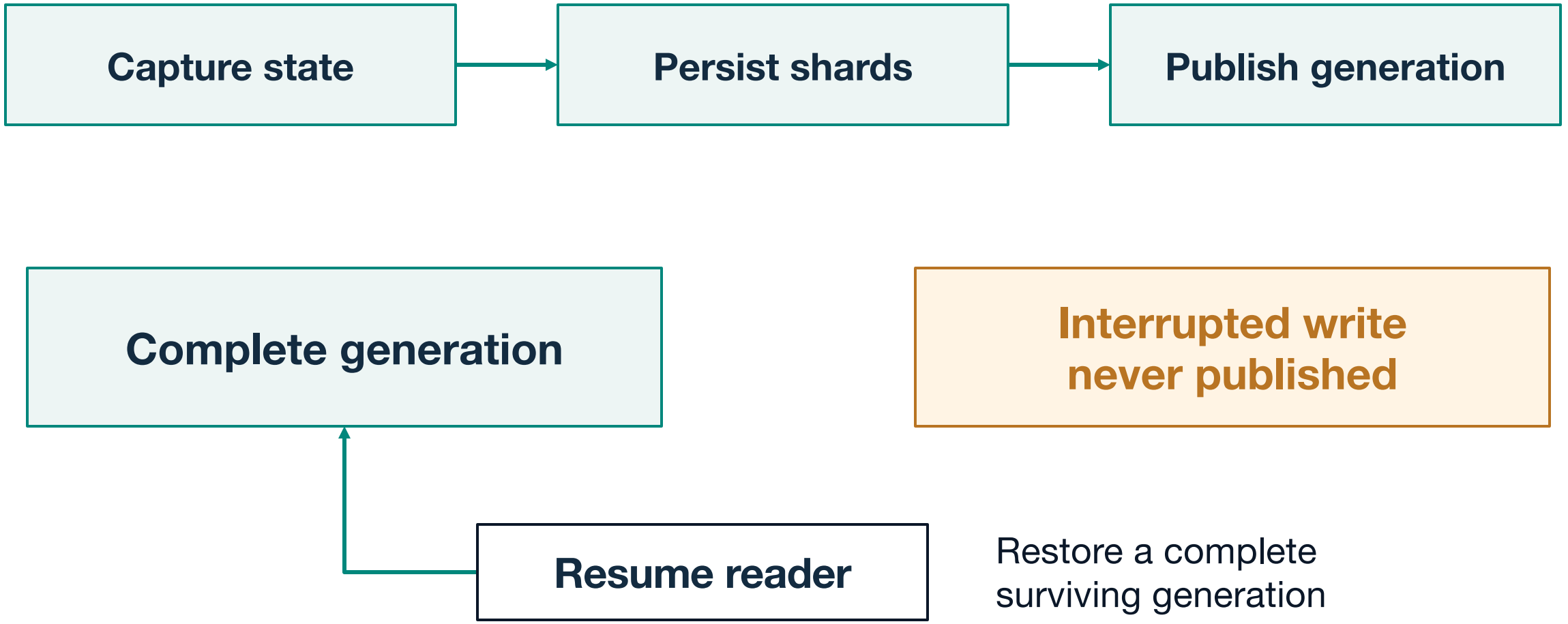
Recovery needs healthy replacement capacity

Fixed-world-size example: 32 nodes × 8 GPUs

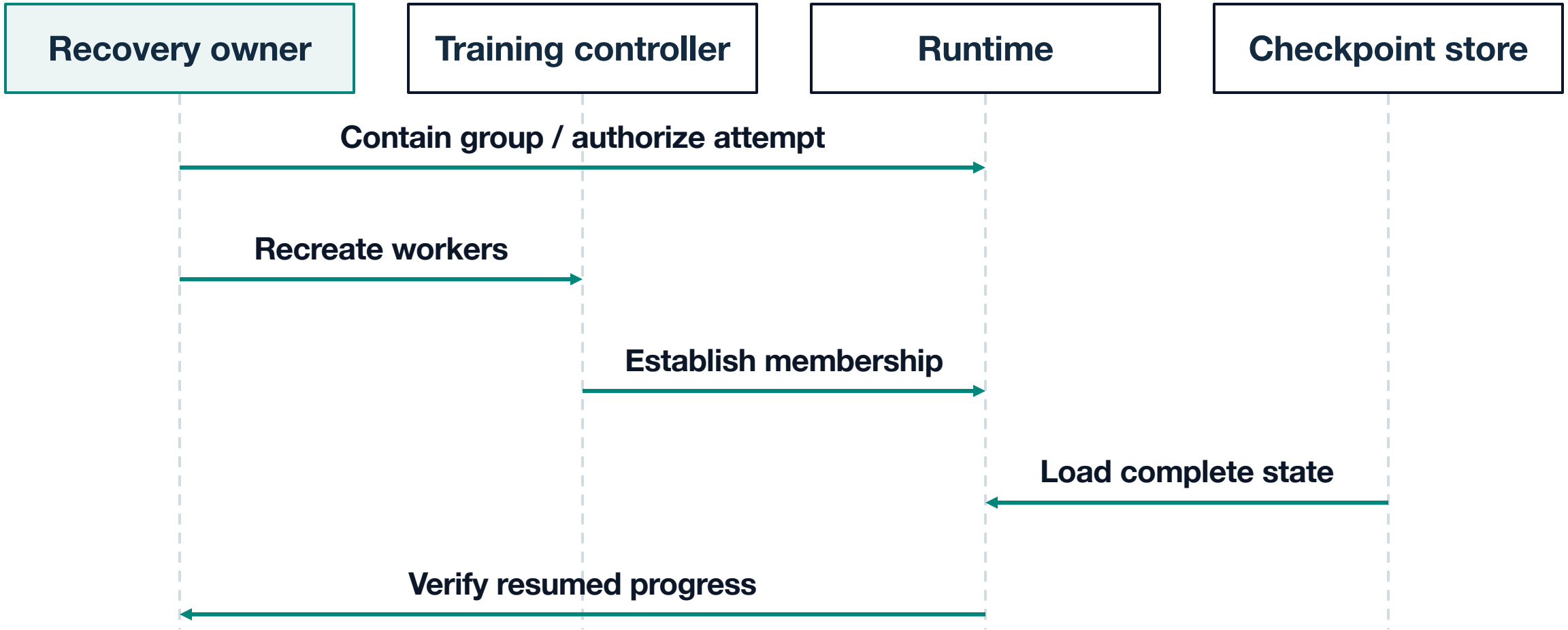


Healthy eligible capacity is required before membership and restore

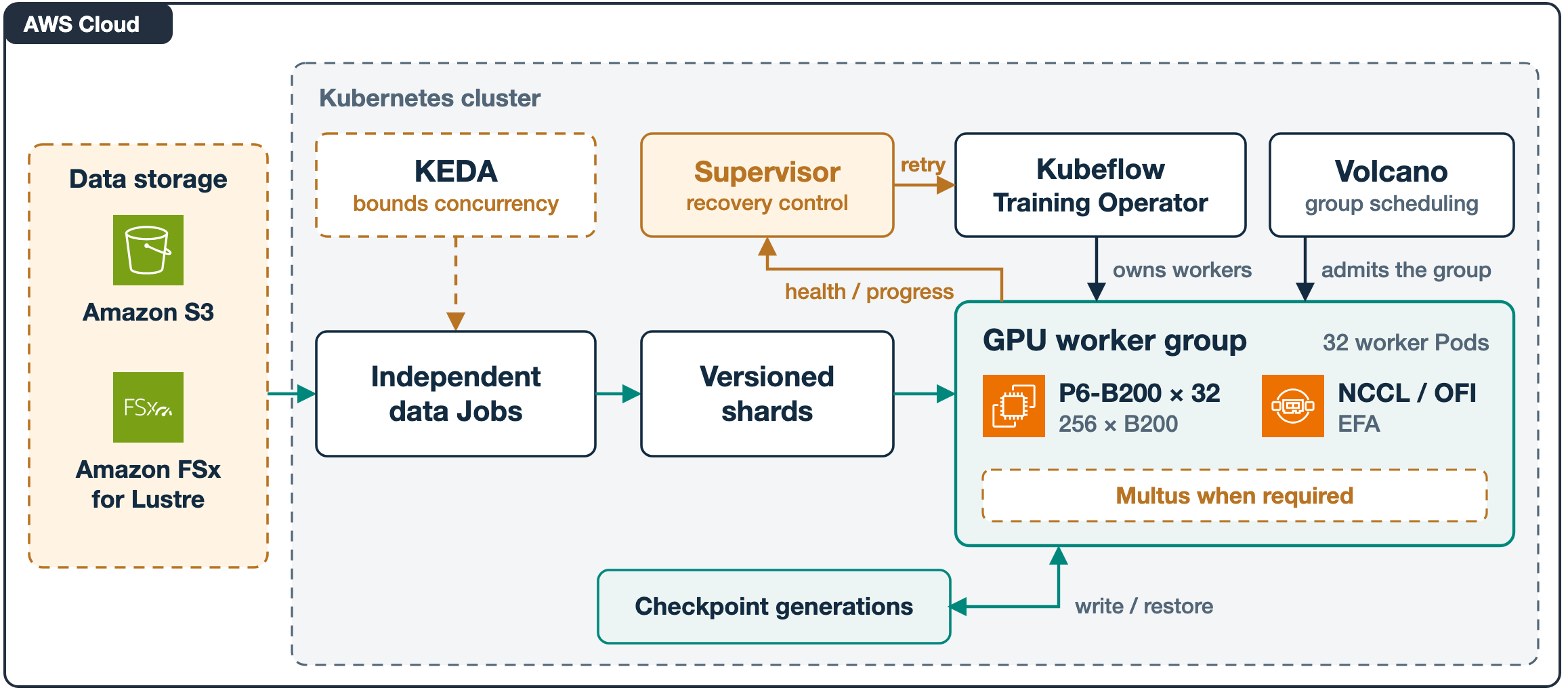
Complete checkpoints define the restore point



Recovery finishes when useful work resumes



One architecture connects the operating loop



Four rehearsals before a long run

Group startup	All required ranks reach rendezvous under competing allocations
Collective transport	Representative payloads connect to an application step
Input delivery	Complete samples reach the consumer during overlapping I/O
Recovery	A complete generation supports the first resumed step

Four contracts for one worker group

ADMIT	COMMUNICATE	FEED	RECOVER
Controller + scheduler	NCCL / OFI / EFA Attachment when needed	Jobs + bounded scaling	Healthy capacity + complete checkpoint

Questions

A1 Implementation map

A2 Scheduling choices

A3 Worker APIs

A4 Topology & placement

A5 Fabric checks

A6 Preparation retries

A7 Recovery control

A8 Checkpoint publication

A9 Research & methods

Implementation map: tools, owners and interfaces

Boundary	Tools / owner	Implementation decision
ADMIT	Training controller + admission system	Required roles, group size and startup policy
COMMUNICATE	CNI / Multus + NCCL / OFI / EFA	Attachments, allocated devices and selected transport
FEED	Independent Jobs + KEDA + storage	Work identity, complete publication and concurrency
RECOVER	Recovery owner + runtime + checkpoints	Healthy capacity, generation selection and resumed work

Choosing group admission and scheduling

Volcano

Batch scheduler

Queues, gang policy and placement

Kueue

Workload admission

Quota, topology and readiness policy

Native gang

Kubernetes v1.37 beta

Opt-in PodGroup API and scheduler configuration

Disabled by default

Choose by controller integration, queue policy and topology requirements

← Questions

Worker ownership across training APIs

Kubeflow Trainer TrainJob	Training Operator PyTorchJob	VolcanoJob
Runtime definitions + Trainer controller → JobSet	Framework worker lifecycle (legacy API)	Volcano job controller and tasks

Choose one worker owner and align admission, startup and retries

← Questions

Topology: allocation, placement and runtime

Local allocation

8 GPUs requested by one Pod in this example
Align CPU / GPU / NIC using the actual map

Group placement

Select compatible nodes and a network domain
EFA communicating nodes: same AZ

Runtime + capacity

Map ranks; measure collectives and application step
A compatible replacement must satisfy placement too

Collective transport checks inside the Pod

Inside the actual workload Pod




```
fi_info -p efa -t FI_EP_RDM
NCCL_DEBUG=INFO
```

fi_info: provider availability
Workload logs: selected transport
Application step: useful progress

Compare the same image, placement, operation, dtype, units and ranks

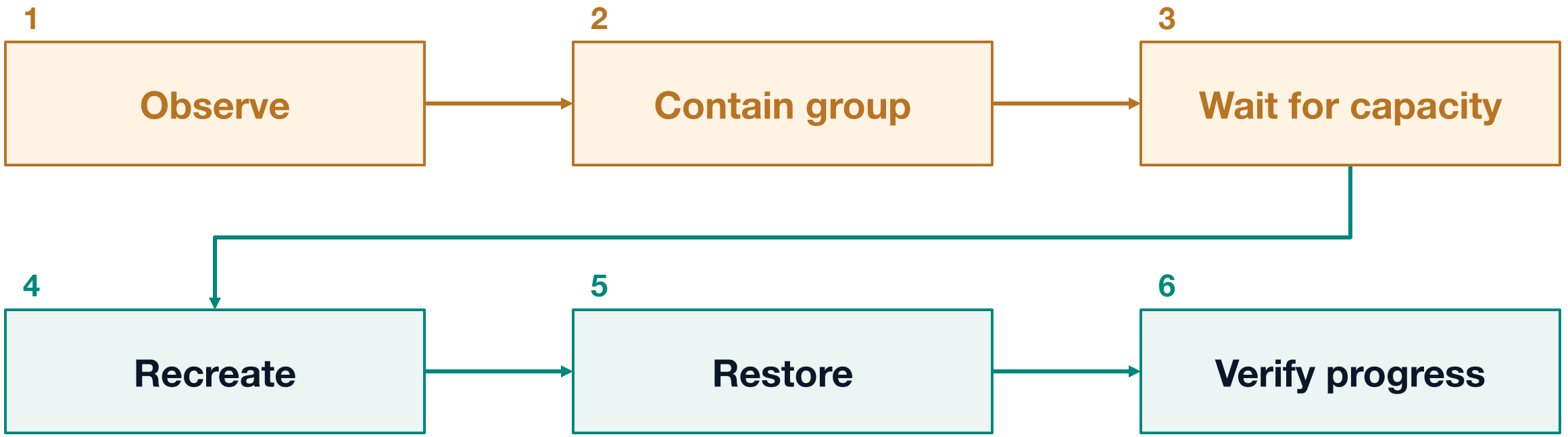
Idempotent preparation Jobs



Application publication makes retries safe

Bound independent Job concurrency by storage and consumer demand

A single recovery authority coordinates attempts



One retry authority per attempt. NoSchedule does not evict running Pods.

Checkpoint publication and restore protocol



Reject partial generations and stale attempts

Recoverable state

Model / optimizer / scheduler
RNG + input state when required

Measure the continuation

State age / capacity wait / restore
First verified resumed step

Research behind the operating methods

01 Tiresias, NSDI 2019

02 Pollux, OSDI 2021

03 TACCL, NSDI 2023

04 TopoOpt, NSDI 2023

05 Plumber, MLSys 2022

06 Cachew, USENIX ATC 2022

07 CheckFreq, FAST 2021

08 ByteCheckpoint, NSDI 2025

09 Gemini, SOSP 2023

10 MegaScale, NSDI 2024, section 4

ADMIT

Queue and placement policy

COMMUNICATE

Topology and collective design

FEED

Input profiling and caching

RECOVER

State persistence and coordinated restart