Kubernetes Workload Resiliency in Action - Beyond Basics

Nabarun Pal
Broadcom

Akhil Mohan
Broadcom

It can run.

It can run.

Can it run reliably?

BEFORE WE START

Three questions for the room

BEFORE WE START

Three questions for the room

? Who runs AI / ML on Kubernetes today?

BEFORE WE START

Three questions for the room

? Who runs AI / ML on Kubernetes today?

? Who has lost a long job to an eviction?

BEFORE WE START

Three questions for the room

? Who runs AI / ML on Kubernetes today?

? Who has lost a long job to an eviction?

? Who shares one cluster across many teams?

Story Time

Kubernetes is becoming

Kubernetes is becoming

THE OS for AI

THE GAP BETWEEN

“Runs” and “runs reliably”.

THE GAP BETWEEN

“Runs” and “runs reliably”.

66%

can run it

THE GAP BETWEEN

“Runs” and “runs reliably”.

66%
can run it

7%
deploy daily

— CNCF 2025 Annual Survey

THE FIX IS UNIVERSAL

The stakes are loud in AI. The fix is universal.

LOUD IN AI

AI workloads make the cost of getting it wrong obvious: expensive, long-running, bursty.

THE FIX IS UNIVERSAL

The stakes are loud in AI. The fix is universal.

LOUD IN AI

AI workloads make the cost of getting it wrong obvious: expensive, long-running, bursty.

UNIVERSAL FIX

The mechanisms that fix it protect **any** workload, in **any** production cluster, today.

THE REAL LESSON

THE REAL LESSON

No single mechanism is enough.

THE REAL LESSON

No single mechanism is enough.

Resilience is how you layer them.
The interactions are where teams break.

THE RESILIENCE STACK

Resilience is a stack, not a setting.

Scheduler + Kubelet

↓ foundation

THE RESILIENCE STACK

Resilience is a stack, not a setting.

Requests / Limits + QoS

PriorityClass

ResourceQuota

PodDisruptionBudget

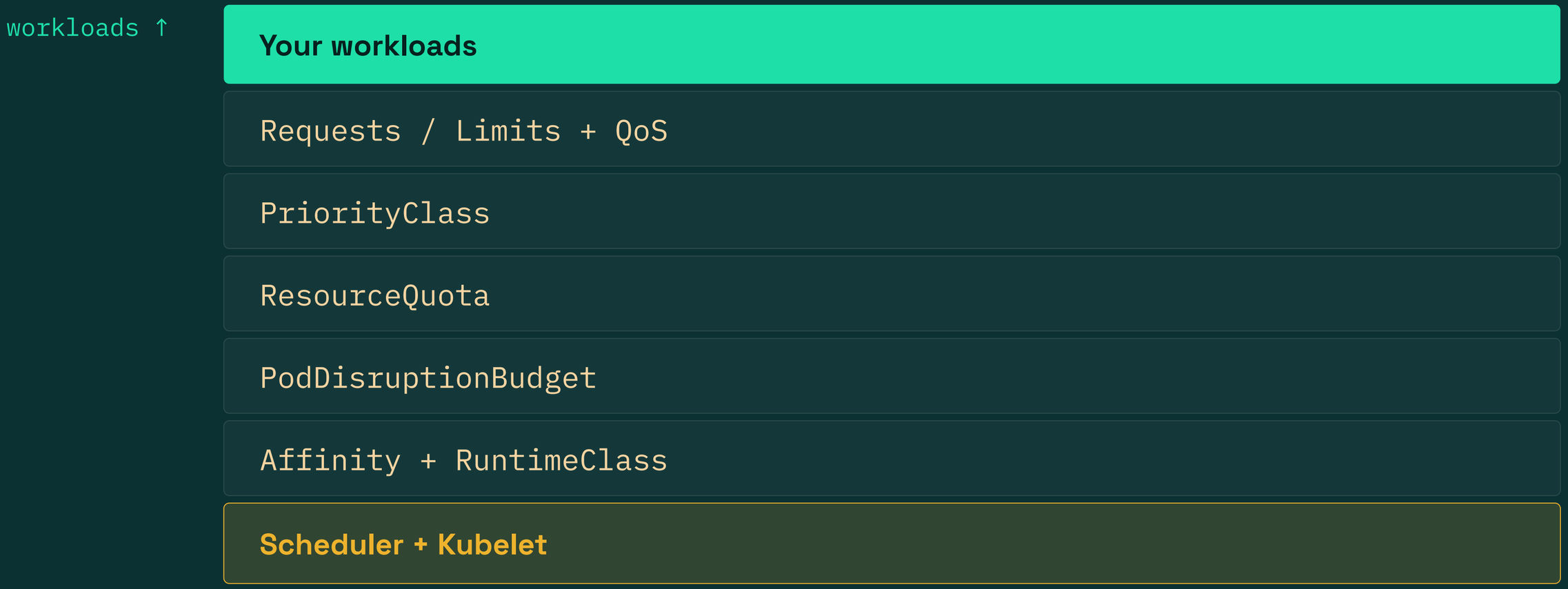
Affinity + RuntimeClass

Scheduler + Kubelet

↓ foundation

THE RESILIENCE STACK

Resilience is a stack, not a setting.



THE PLAN

Where the next thirty minutes go.

01 3 workloads, 3 failure shapes	02 The 6 levers each one pulls	03 How the levers interact	04 Where it all breaks at scale

STORY 1

Training that must not break.

The long-running workload pattern

STORY 1 · THE SHAPE OF THE PROBLEM

Not stateless. Kill one rank, the collective stalls.

STORY 1 · THE SHAPE OF THE PROBLEM

Not stateless. Kill one rank, the collective stalls.

- **Long-running:** hours to days on scarce accelerators.

STORY 1 · THE SHAPE OF THE PROBLEM

Not stateless. Kill one rank, the collective stalls.

- **Long-running:** hours to days on scarce accelerators.
- **Gang-scheduled:** every rank runs, or none make progress.

STORY 1 · THE SHAPE OF THE PROBLEM

Not stateless. Kill one rank, the collective stalls.

- **Long-running:** hours to days on scarce accelerators.
- **Gang-scheduled:** every rank runs, or none make progress.
- An interruption wastes **everything** since the last checkpoint.

STORY 1 · THE LEVERS

Four levers. PDB is the hero.

`PriorityClass`	don’t preempt training for low-value work
`PodDisruptionBudget`	**turn a hard kill into a graceful drain, so the job checkpoints**
`Grace + preStop`	actually use that drain window to checkpoint
`Gang + topology`	place ranks together, or hold GPUs for nothing

STORY 1 · THE LEVERS

Four levers. PDB is the hero.

`PriorityClass`	don’t preempt training for low-value work
`PodDisruptionBudget`	**turn a hard kill into a graceful drain, so the job checkpoints**
`Grace + preStop`	actually use that drain window to checkpoint
`Gang + topology`	place ranks together, or hold GPUs for nothing

THE KILLER STAT

5 of 6 ranks

= 0% progress

83% of GPUs held, doing nothing.

`At scale: 400 / 512 bound, 112 pending → 400 GPUs idle`

REFERENCE

developer.nvidia.com

AI Workloads on Rack-Scale Supercomputers

From hardware to topology-aware scheduling.

TAKE IT HOME

A PDB is how a training job survives the cluster doing its job.

Drains, upgrades, normal maintenance.
Go check yours! Most jobs don't have one.

STORY 2

Inference that stays up

Mission-critical services

STORY 2 · THE SHAPE OF THE PROBLEM

STORY 2 · THE SHAPE OF THE PROBLEM

- Sessions are long-running and resource-intensive.

STORY 2 · THE SHAPE OF THE PROBLEM

- Sessions are long-running and resource-intensive.
- Partially stateful: in-memory **KV / token caches**.

STORY 2 · THE SHAPE OF THE PROBLEM

- Sessions are long-running and resource-intensive.
- Partially stateful: in-memory **KV / token caches**.
- Routing must be model-aware and criticality-aware.

STORY 2 · THE SHAPE OF THE PROBLEM

A warm KV cache means ”just restart it” isn’t free.

- Sessions are long-running and resource-intensive.
- Partially stateful: in-memory **KV / token caches**.
- Routing must be model-aware and criticality-aware.

STORY 2 · WHAT IT DEMANDS

Three demands the demo never tests.

Elastic scaling

pods **and** nodes — HPA/KEDA + Cluster Autoscaler/Karpenter

STORY 2 · WHAT IT DEMANDS

Three demands the demo never tests.

Elastic scaling

pods **and** nodes — HPA/KEDA + Cluster Autoscaler/Karpenter

Low latency

held steady under bursty load, not just at rest

STORY 2 · WHAT IT DEMANDS

Three demands the demo never tests.

Elastic scaling pods **and** nodes — HPA/KEDA + Cluster Autoscaler/Karpenter	**Low latency** held steady under bursty load, not just at rest	**Automated failover** no human in the loop when a replica dies

STORY 2 · THE LEVERS

Same shape.
Graceful degradation is the idea.

`Requests/Limits + QoS`	Guaranteed for latency-critical serving
`PriorityClass`	graceful degradation - shed batch before interactive
`PodDisruptionBudget`	hold minimum replicas through disruptions
`DRA + RuntimeClass`	GPU allocation. Isolation where serving needs it

STORY 2 · THE KEY INTERACTION

Under node pressure - who gets evicted first?

STORY 2 · THE KEY INTERACTION

Under node pressure - who gets evicted first?

first	BestEffort, any priority
next	Burstable, low priority
later	Burstable, high priority
last	Guaranteed, high priority

STORY 2 · THE KEY INTERACTION

Under node pressure - who gets evicted first?

first	BestEffort, any priority
next	Burstable, low priority
later	Burstable, high priority
last	Guaranteed, high priority

THE TRAP

A high-priority BestEffort pod dies before a low-priority Guaranteed pod.

QoS is checked first, **then** priority. Priority is not a force field.

STORY 2 · ECOSYSTEM

Where inference work now lives.

Gateway API Inference Extension

Kueue · elastic workloads

NVIDIA Grove

WG Serving concluded · Feb 2026 →

autoscaling → SIG Node / Scheduling

multi-host (LWS) → SIG Apps

DRA → WG Device Management

STORY 3

Multi-tenancy: one cluster, many teams

Shared clusters

STORY 3 · THE SHAPE OF THE PROBLEM

STORY 3 · THE SHAPE OF THE PROBLEM

- Many teams, **one** expensive pool.

STORY 3 · THE SHAPE OF THE PROBLEM

- Many teams, **one** expensive pool.
- One tenant bin-packs to the edge and starves the rest.

STORY 3 · THE SHAPE OF THE PROBLEM

- Many teams, **one** expensive pool.
- One tenant bin-packs to the edge and starves the rest.
- Noisy neighbors, manual pod-herding, over-provisioning ”just in case”.

STORY 3 · THE SHAPE OF THE PROBLEM

One tenant bin-packs to the edge, the rest starve.

- Many teams, **one** expensive pool.
- One tenant bin-packs to the edge and starves the rest.
- Noisy neighbors, manual pod-herding, over-provisioning ”just in case”.

STORY 3 · THE LEVERS

Two jobs: fairness and isolation.

`ResourceQuota`	fair caps per tenant, scoped by PriorityClass
`PriorityClass`	system components outrank every tenant
`RuntimeClass`	isolate untrusted or specialized workloads
`Affinity/spread/taints`	carve pools, spread replicas for blast radius

STORY 3 · THE INTERACTION

Quota is a dial, not a switch.

too tight →

legitimate jobs sit **Pending**

STORY 3 · THE INTERACTION

Quota is a dial, not a switch.

too tight →

legitimate jobs sit **Pending**

← too loose

one tenant **hoards** the pool

STORY 3 · THE INTERACTION

Quota is a dial, not a switch.

too tight →	balance point	← too loose
legitimate jobs sit **Pending**	scope **by PriorityClass** : generous for prod, tight for dev	one tenant **hoards** the pool

STORY 3 · THE INTERACTION

STORY 3 · THE INTERACTION

The cluster itself.

STORY 3 · THE INTERACTION

The cluster itself.

- Bin-pack to the edge and you starve the node’s own machinery.

STORY 3 · THE INTERACTION

The cluster itself.

- Bin-pack to the edge and you starve the node’s own machinery.
- Pod parts (CNI, CSI): requests + system-node-critical

STORY 3 · THE INTERACTION

The cluster itself.

- Bin-pack to the edge and you starve the node's own machinery.
- Pod parts (CNI, CSI): requests + system-node-critical
- Non-pod parts (kubelet, runtime, OS): reserve with Node Allocatable

STORY 3 · THE INTERACTION

The cluster itself.

- Bin-pack to the edge and you starve the node’s own machinery.
- Pod parts (CNI, CSI): requests + system-node-critical
- Non-pod parts (kubelet, runtime, OS): reserve with Node Allocatable

You can’t give the OS a PriorityClass. You reserve for it.

STORY 3 · SPECIALIZED CASE — HPC & SCIENTIFIC COMPUTING

The failure is silent: no error, just slower every run.

- Tightly coupled MPI / collective jobs.
- Performance dominated by interconnect and NUMA / device locality.
- Topology Manager (NUMA alignment), gang scheduling, DRA + ComputeDomains for multi-node NVLink.

THE TOOLBOX · SIX LEVERS

What each really does

LEVER	WHAT IT ACTUALLY CONTROLS	COMMON MISREAD
`Requests/Limits + QoS`		*”limits = guarantee”*
`PriorityClass`		*”scheduled first”*
`ResourceQuota`		*”per-user”*
`RuntimeClass`		*”only for sandboxing”*
`PodDisruptionBudget`		*”stops all failure”*
`Affinity / topology spread`		*”scheduler reads my mind”*

THE TOOLBOX · SIX LEVERS

What each really does

LEVER	WHAT IT ACTUALLY CONTROLS	COMMON MISREAD
`Requests/Limits + QoS`	scheduling + eviction class	*”limits = guarantee”*
`PriorityClass`		*”scheduled first”*
`ResourceQuota`		*”per-user”*
`RuntimeClass`		*”only for sandboxing”*
`PodDisruptionBudget`		*”stops all failure”*
`Affinity / topology spread`		*”scheduler reads my mind”*

THE TOOLBOX · SIX LEVERS

What each really does

LEVER	WHAT IT ACTUALLY CONTROLS	COMMON MISREAD
`Requests/Limits + QoS`	scheduling + eviction class	*”limits = guarantee”*
`PriorityClass`	eviction / preemption order	*”scheduled first”*
`ResourceQuota`		*”per-user”*
`RuntimeClass`		*”only for sandboxing”*
`PodDisruptionBudget`		*”stops all failure”*
`Affinity / topology spread`		*”scheduler reads my mind”*

THE TOOLBOX · SIX LEVERS

What each really does

LEVER	WHAT IT ACTUALLY CONTROLS	COMMON MISREAD
`Requests/Limits + QoS`	scheduling + eviction class	*”limits = guarantee”*
`PriorityClass`	eviction / preemption order	*”scheduled first”*
`ResourceQuota`	per-namespace caps	*”per-user”*
`RuntimeClass`		*”only for sandboxing”*
`PodDisruptionBudget`		*”stops all failure”*
`Affinity / topology spread`		*”scheduler reads my mind”*

THE TOOLBOX · SIX LEVERS

What each really does

LEVER	WHAT IT ACTUALLY CONTROLS	COMMON MISREAD
`Requests/Limits + QoS`	scheduling + eviction class	*”limits = guarantee”*
`PriorityClass`	eviction / preemption order	*”scheduled first”*
`ResourceQuota`	per-namespace caps	*”per-user”*
`RuntimeClass`	runtime + OCI spec, ulimits	*”only for sandboxing”*
`PodDisruptionBudget`		*”stops all failure”*
`Affinity / topology spread`		*”scheduler reads my mind”*

THE TOOLBOX · SIX LEVERS

What each really does

LEVER	WHAT IT ACTUALLY CONTROLS	COMMON MISREAD
`Requests/Limits + QoS`	scheduling + eviction class	*”limits = guarantee”*
`PriorityClass`	eviction / preemption order	*”scheduled first”*
`ResourceQuota`	per-namespace caps	*”per-user”*
`RuntimeClass`	runtime + OCI spec, ulimits	*”only for sandboxing”*
`PodDisruptionBudget`	voluntary disruption only	*”stops all failure”*
`Affinity / topology spread`		*”scheduler reads my mind”*

THE TOOLBOX · SIX LEVERS

What each really does

LEVER	WHAT IT ACTUALLY CONTROLS	COMMON MISREAD
`Requests/Limits + QoS`	scheduling + eviction class	*”limits = guarantee”*
`PriorityClass`	eviction / preemption order	*”scheduled first”*
`ResourceQuota`	per-namespace caps	*”per-user”*
`RuntimeClass`	runtime + OCI spec, ulimits	*”only for sandboxing”*
`PodDisruptionBudget`	voluntary disruption only	*”stops all failure”*
`Affinity / topology spread`	placement + spreading	*”scheduler reads my mind”*

MATCH THE LEVER TO THE WORKLOAD'S LIFE

Anchored and protected vs disposable and retryable.

	Long-running (training, serving)	**Ephemeral** (batch, Jobs)
Disruption	PDB, checkpoints	preemptible, just retry
Placement	anti-affinity, spread	pack, backfill
QoS	Guaranteed / Burstable	Burstable / BestEffort

WHERE IT BREAKS

Three failure modes

WHERE IT BREAKS

At scale, the abstractions leak

Three failure modes

FAILURE MODE 1 · EVICTIONS

Push the node past the edge, the accounting bites.

FAILURE MODE 1 · EVICTIONS

Push the node past the edge, the accounting bites.

- Over-subscription and aggressive bin-packing push nodes past the edge.
- At large pools, even the accounting bites: the **”96 CPU” rounding error**.

FAILURE MODE 1 · EVICTIONS

Push the node past the edge, the accounting bites.

- Over-subscription and aggressive bin-packing push nodes past the edge.
- At large pools, even the accounting bites: the **”96 CPU” rounding error**.

Fixed upstream. We hit it at scale before it was.

FAILURE MODE 2 · PERFORMANCE DEGRADATION

The scariest failures don’t page you. They tax you.

- NUMA / topology misalignment costs you on every run.
- No error, no alert, just slower.
- Topology Manager is opt-in. Off by default

AT 8 NUMA NODES

>4.2B

hint combinations

~21 min to schedule

k8s #131738

FAILURE MODE 3 · CASCADING FAILURES

One node tips the next, which is also at the edge.

node A

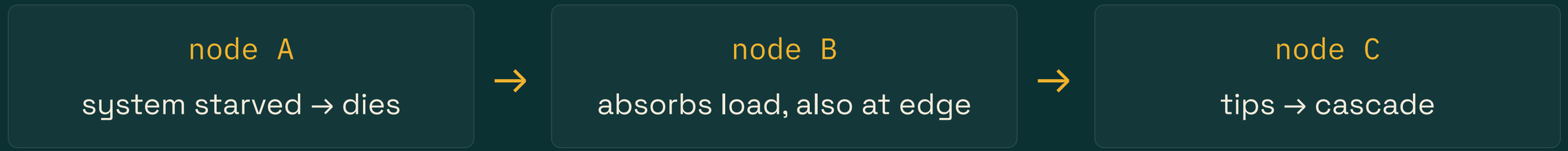
system starved → dies

FAILURE MODE 3 · CASCADING FAILURES

One node tips the next, which is also at the edge.

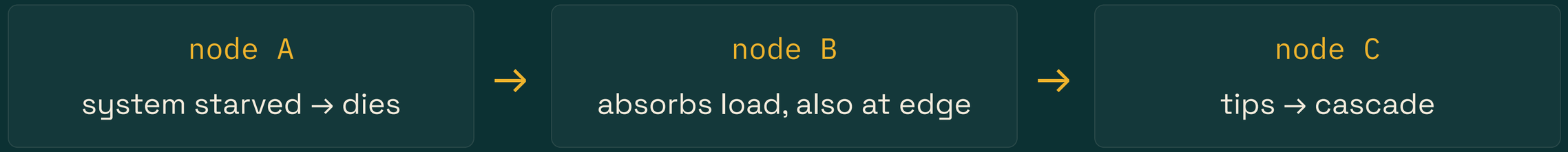
FAILURE MODE 3 · CASCADING FAILURES

One node tips the next, which is also at the edge.



FAILURE MODE 3 · CASCADING FAILURES

One node tips the next, which is also at the edge.



Reserve node headroom before you pack, not after.

It can run.

It can run.

Now make it run reliably.

TAKE THIS HOME

Five things to live by.

TAKE THIS HOME

Five things to live by.

- Layered beats any single mechanism.

TAKE THIS HOME

Five things to live by.

- Layered beats any single mechanism.
- Match the lever to the failure shape.

TAKE THIS HOME

Five things to live by.

- Layered beats any single mechanism.
- Match the lever to the failure shape.
- Protect the cluster’s own components first.

TAKE THIS HOME

Five things to live by.

- Layered beats any single mechanism.
- Match the lever to the failure shape.
- Protect the cluster’s own components first.
- The skill is the **interactions**, not the settings.

TAKE THIS HOME

Five things to live by.

- Layered beats any single mechanism.
- Match the lever to the failure shape.
- Protect the cluster’s own components first.
- The skill is the **interactions**, not the settings.
- PDBs turn maintenance from outage into a non-event.

GO LOOK. TODAY.

Three checks that find real gaps.

GO LOOK. TODAY.

Three checks that find real gaps.

? Do your long-running jobs have **PDBs**?

GO LOOK. TODAY.

Three checks that find real gaps.

? Do your long-running jobs have **PDBs**?

? Do your latency-critical pods have **Guaranteed QoS**?

GO LOOK. TODAY.

Three checks that find real gaps.

? Do your long-running jobs have **PDBs**?

? Do your latency-critical pods have **Guaranteed QoS**?

? Do your system daemons **outrank** user workloads?

KubeCon CloudNativeCon India 2026

Thank you

Nabarun Pal
@palnabarun

Akhil Mohan
@akhilerm

KubeCon CloudNativeCon India 2026

Thank you

Questions?

Nabarun Pal
@palnabarun

Akhil Mohan
@akhilerm