



~~Everything You Ever Wanted~~ A Few Things To Know About Resource Scheduling

eBay Meetup
December 14, 2016

Tim Hockin <thockin@google.com>
Senior Staff Software Engineer, Google
@thockin



WARNING:
Some of this presentation
is aspirational

I posit:
Kubernetes is fundamentally ABOUT
resource management

- CPU
- Memory

- CPU
- Memory
- Disk space
- Disk time
- Disk “spindles”

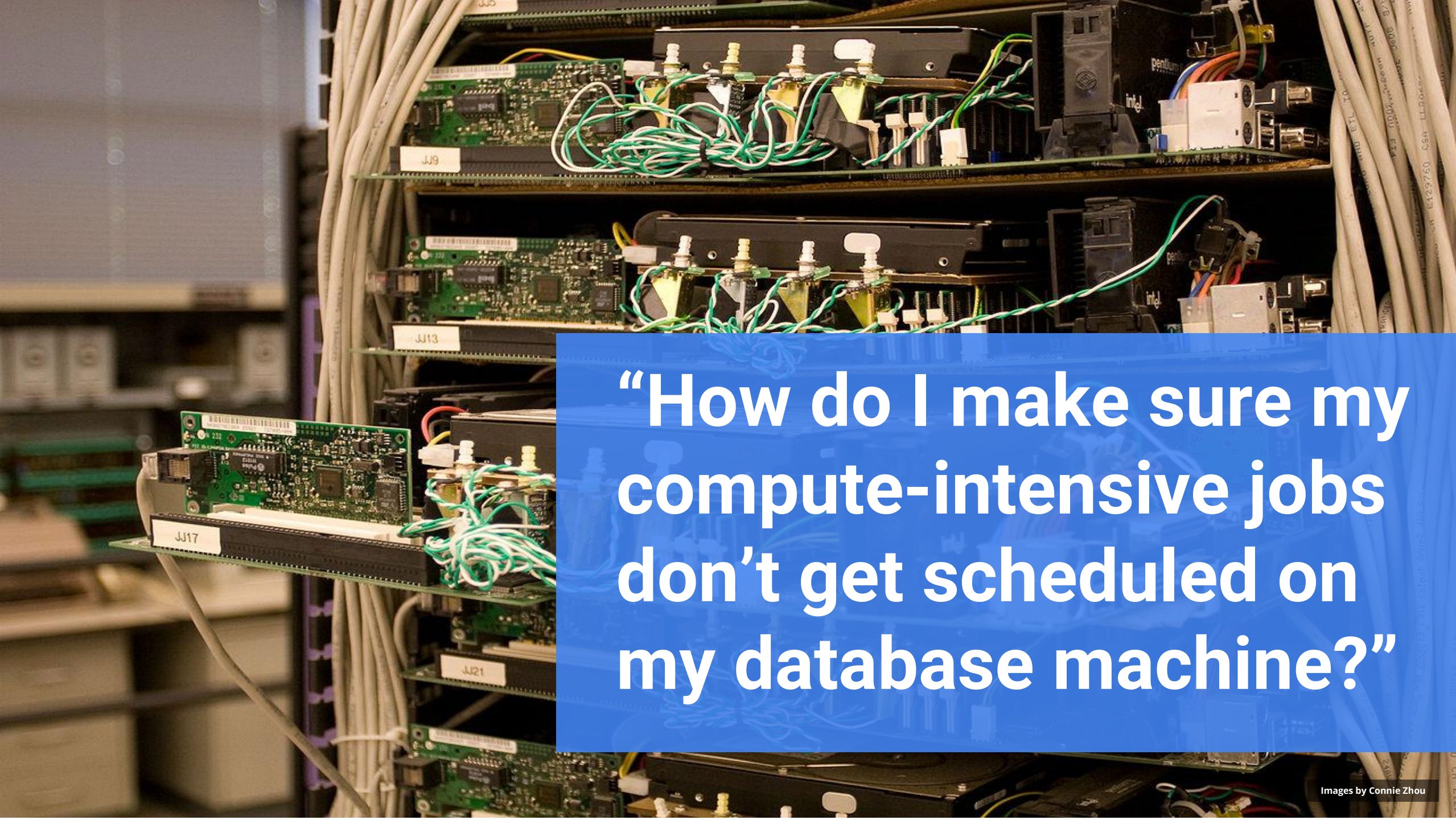
- CPU
- Memory
- Disk space
- Disk time
- Disk “spindles”
- Network bandwidth
- Host ports

- CPU
- Memory
- Disk space
- Disk time
- Disk “spindles”
- Network bandwidth
- Host ports

- Cache lines
- Memory bandwidth
- IP addresses
- Attached storage
- PIDs
- GPUs
- Power

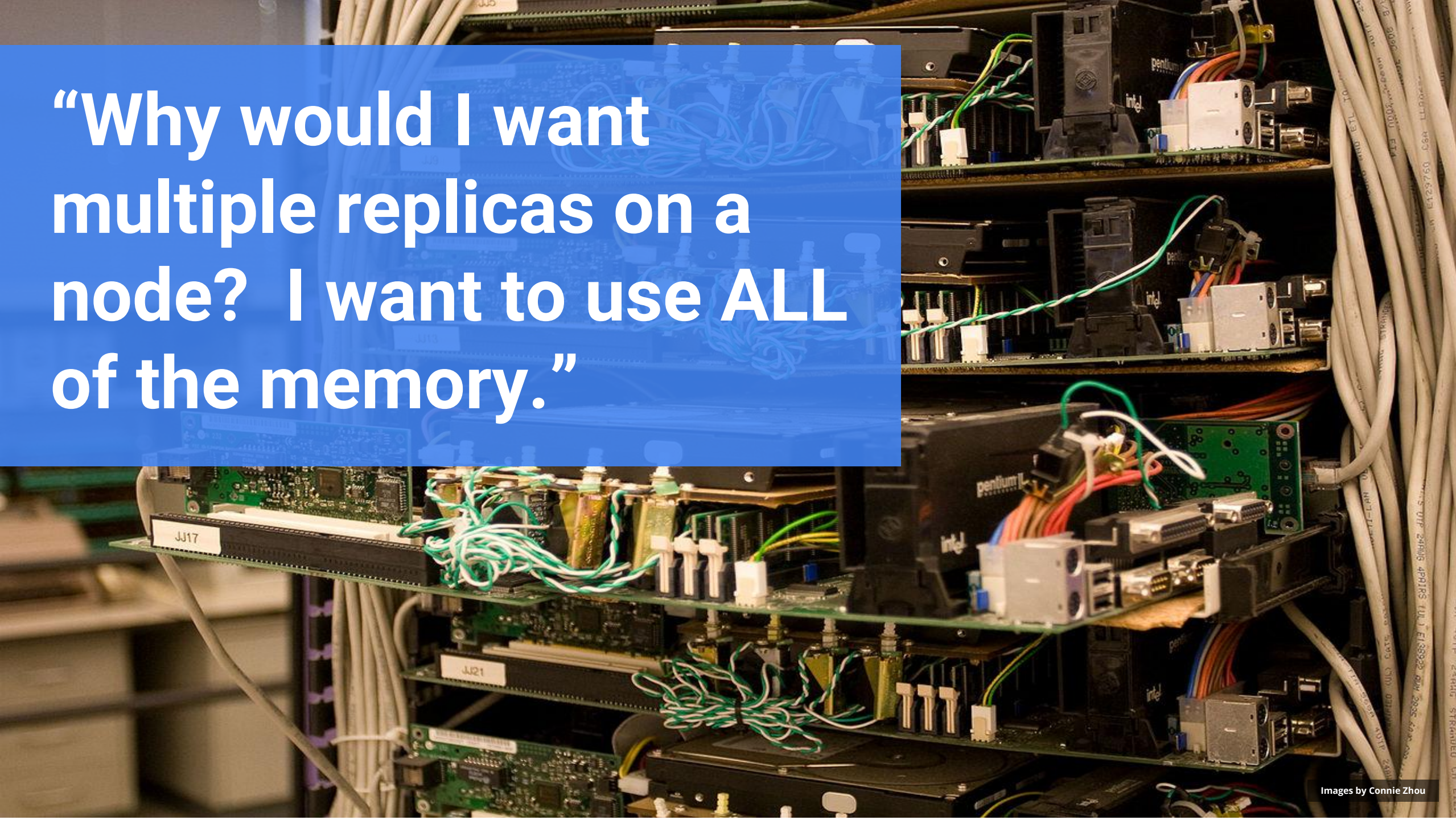
- CPU
 - Memory
 - Disk space
 - Disk time
 - Disk “spindles”
 - Network bandwidth
 - Host ports
 - Arbitrary, opaque third-party resources we can’t possibly predict
- Cache lines
 - Memory bandwidth
 - IP addresses
 - Attached storage
 - PIDs
 - GPUs
 - Power

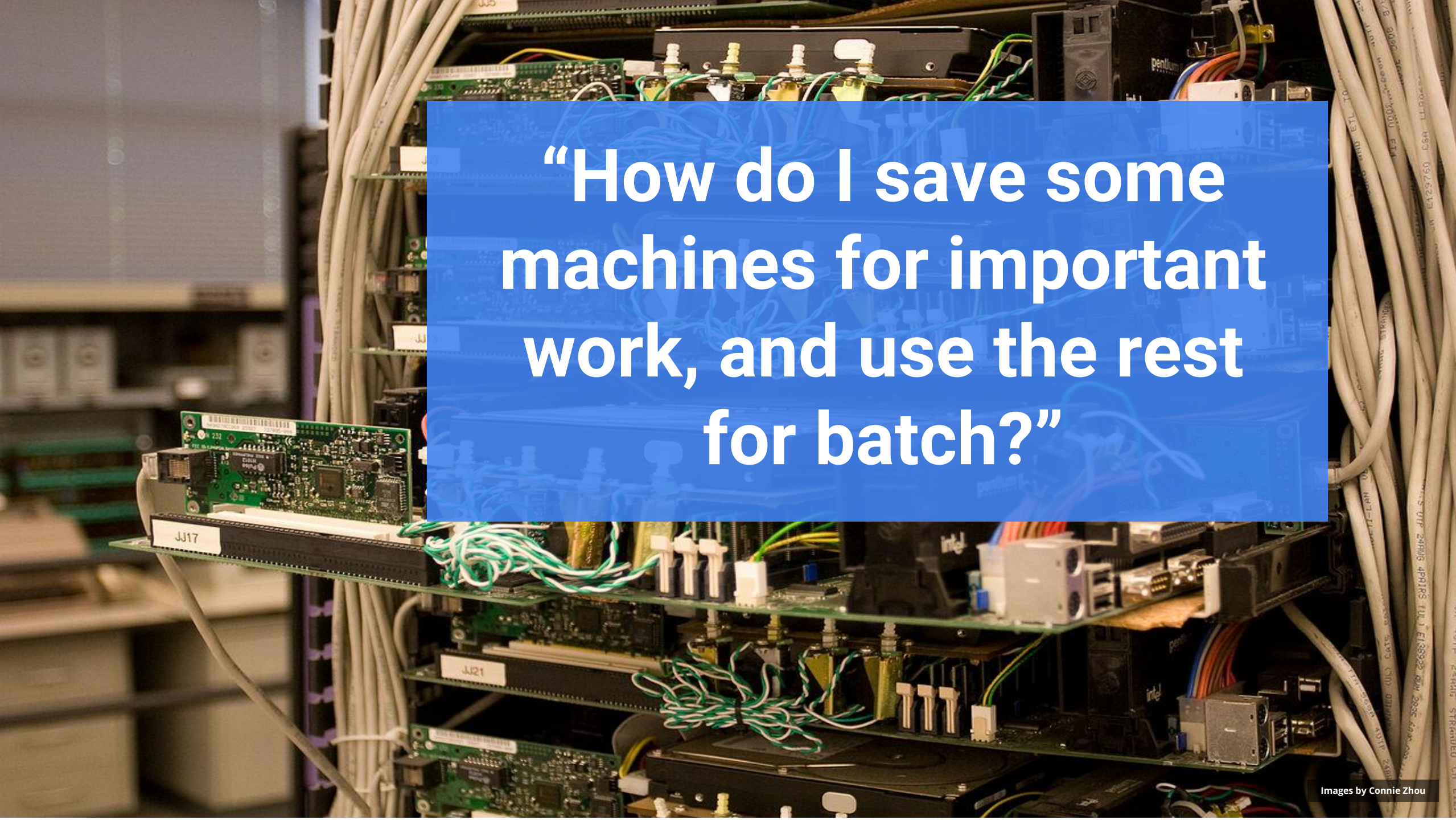
Many people are still asking the wrong questions.



**“How do I make sure my
compute-intensive jobs
don’t get scheduled on
my database machine?”**

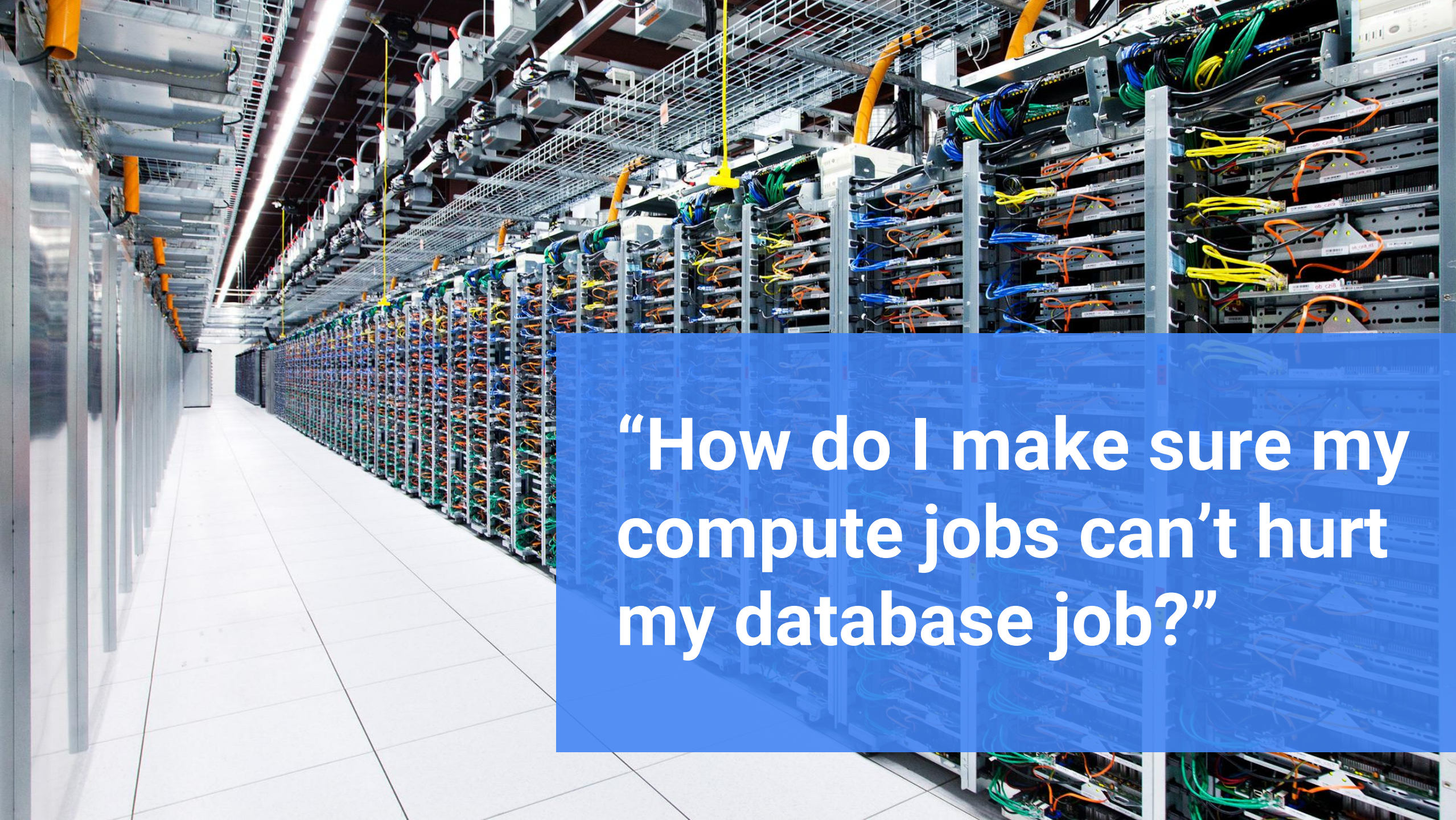
“Why would I want multiple replicas on a node? I want to use ALL of the memory.”



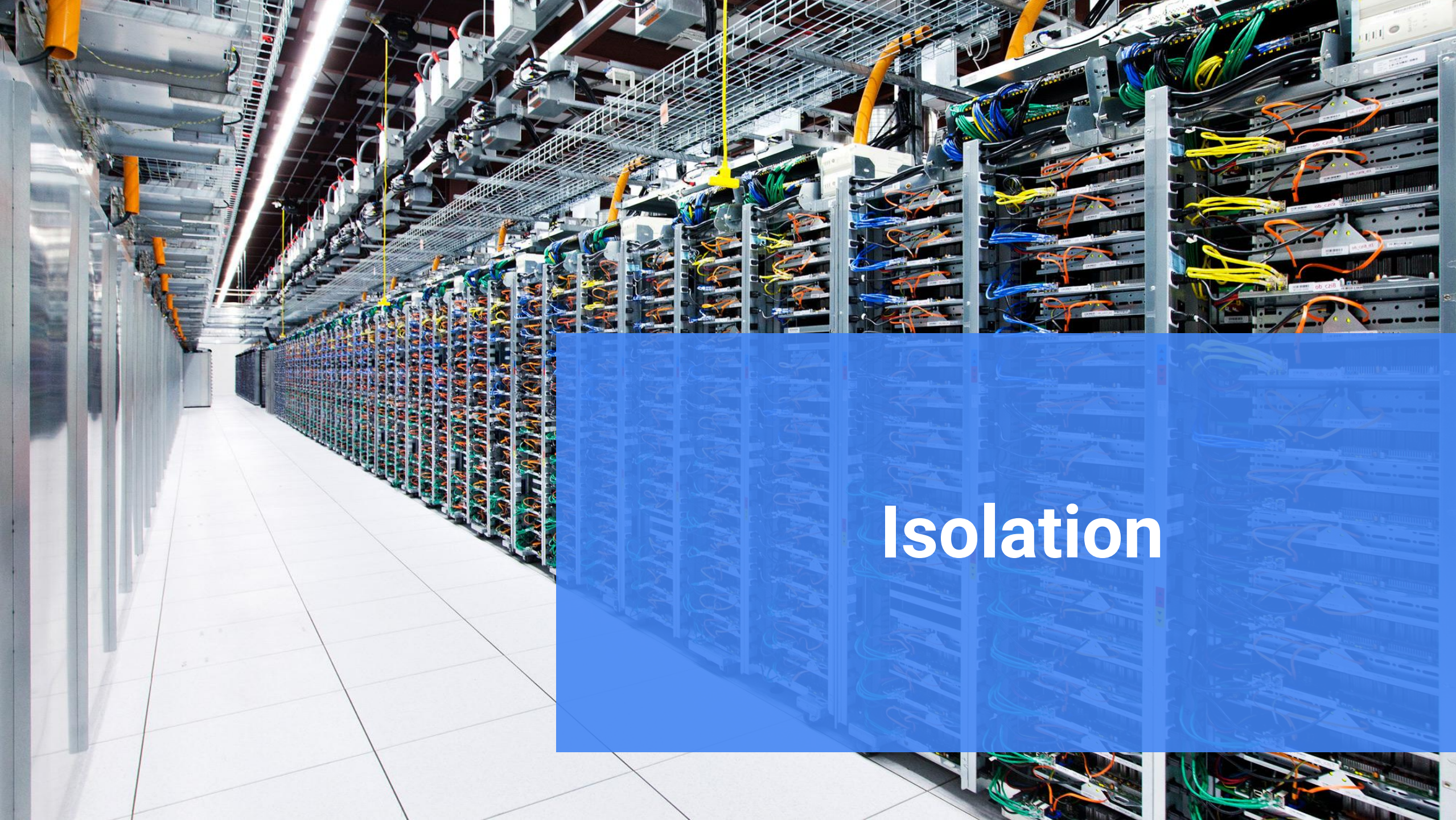


“How do I save some machines for important work, and use the rest for batch?”

So... what should they be asking?



**“How do I make sure my
compute jobs can’t hurt
my database job?”**




Isolation

**“How do I know how
much memory and CPU
my job needs?”**



Sizing





**“How do I safely pack
more work onto less
machines?”**

A photograph of a server room with rows of server racks. The racks are filled with server units, many of which have glowing yellow and blue lights. The perspective is looking down a long aisle between the racks. A large, semi-transparent blue rectangle is overlaid in the center of the image, containing the word "Utilization" in white text.

Utilization

Isolation

Isolation



Prevent apps from hurting each other

Make sure you actually get what you paid for

Kubernetes (and Docker) isolate CPU and memory

Don't handle things like memory bandwidth, disk time, cache, network bandwidth, ... (yet)

Predictability at the extremes is paramount

When does isolation matter?

Infinite loops

Memory leaks

Disk hogs

Fork bombs

Cache thrashing



Counter-measures

Infinite loops: CPU shares and quota

Memory leaks: OOM yourself

Disk hogs: Quota

Fork bombs: Process limits

Cache thrashing: LLC jails, cache segments



Counter-measures: work to do

Infinite loops: CPU shares and quota

Memory leaks: OOM yourself

Disk hogs: Quota

Fork bombs: Process limits

Cache thrashing: LLC jails, cache segments



Resource taxonomy



Compressible resources

- Hold no state
- Can be taken away very quickly
- “Merely” cause slowness when revoked
- e.g. CPU, disk time

Non-compressible resources

- Hold state
- Are slower to be taken away
- Can fail to be revoked
- e.g. Memory, disk space

Requests and limits

Request: amount of a resource allowed to be used, with a strong guarantee of availability

- CPU (seconds/second), RAM (bytes)
- Scheduler will not over-commit requests

Limit: max amount of a resource that can be used, regardless of guarantees

- scheduler ignores limits

Repercussions:

- $\text{request} < \text{usage} \leq \text{limit}$: resources *might* be available
- $\text{usage} > \text{limit}$: throttled or killed



Quality of service

Guaranteed: highest protection

- $\text{limit} == \text{request}$

Burstable: medium protection

- $\text{request} > 0 \ \&\& \ \text{limit} > \text{request}$

Best Effort: lowest protection

- $\text{request} == 0$

How is “protection” implemented?

- CPU: cgroup shares & quota
- Memory: OOM score + user-space evictions



Requests and limits

Behavior at (or near) the limit depends on the particular resource

Compressible resources: throttle usage

- e.g. No more CPU time for you!

Non-compressible resources: reclaim

- e.g. Write-back and reallocate dirty pages
- Failure means process death (OOM)

Being correct is more important than optimal



What if I don't specify?



You get best-effort isolation

You might get defaulted values

You might get OOM killed randomly

You might get CPU starved

You might get no isolation at all

Sizing

Sizing



How many replicas does my job need?

How much CPU/RAM does my job need?

Do I provision for worst-case?

- Expensive, wasteful

Do I provision for average case?

- High failure rate (e.g. OOM)

Benchmark it!

Benchmarks are hard.

Benchmarks are hard.

Accurate benchmarks are VERY hard.

Horizontal scaling

Add more replicas

Easy to reason about

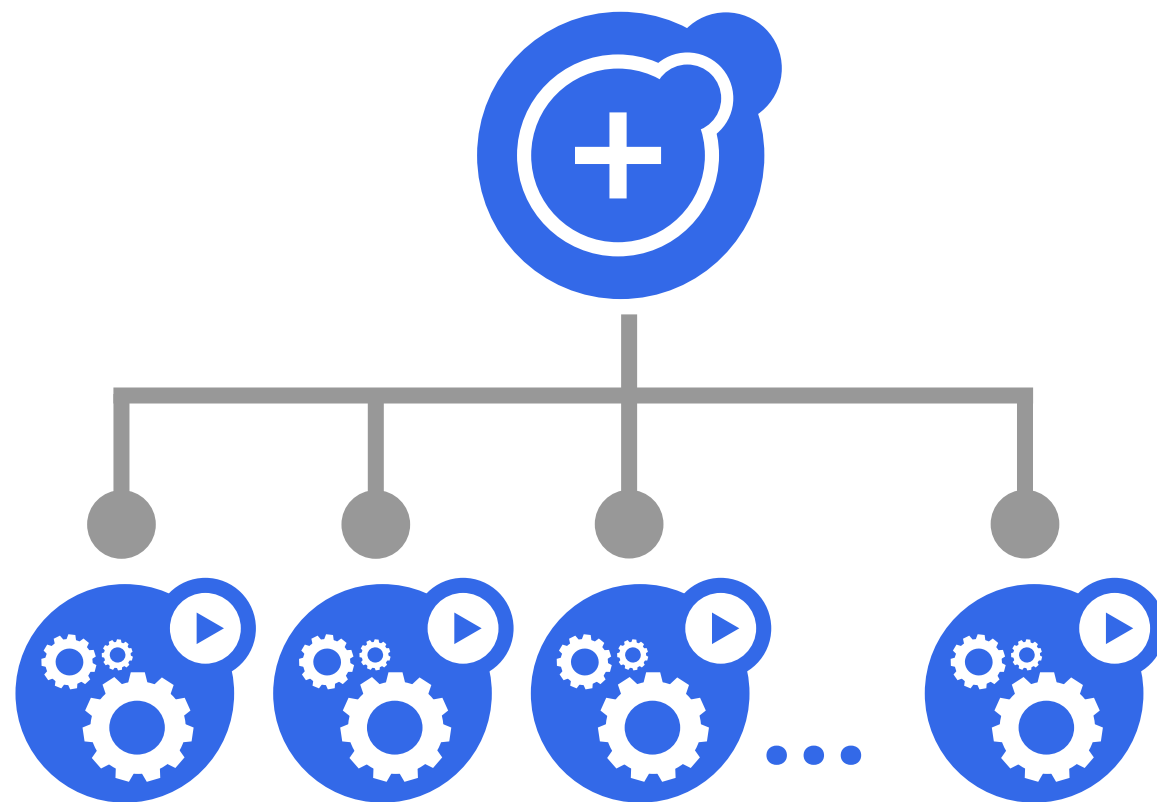
Works well when combined with resource isolation

- Having >1 replica per node makes sense

Not always applicable

- e.g. Memory use scales with cluster size

HorizontalPodAutoscaler



What can we do?



Horizontal scaling is not enough

Resource needs change over time

If only we had an “autopilot” mode...

- Collect stats & build a model
- Predict and react
- Manage Pods, Deployments, Jobs
- Try to stay ahead of the spikes

Autopilot in Borg



Most Borg users use autopilot

See earlier statement regarding benchmarks - even at Google

Kubernetes API is purpose-built for this sort of use-case

We need a **VerticalPodAutoscaler**

Utilization



Utilization

Resources cost money

Wasted resources == wasted money

You ~~want~~ NEED to use as much of your capacity as possible

Selling it is not the same as using it



How can we do better?



Utilization demands isolation

- If you want to push the limits, it has to be safe at the extremes

People are inherently cautious

- Provision for 90%-99% case

VPA & strong isolation should give enough confidence to provision more tightly

We need to do some kernel work, here

Siren's song: over-packing



Clusters need some room to operate

- Nodes fail or get upgraded

As you approach 100% bookings (requests), consider what happens when things go bad

- Nowhere to squeeze the toothpaste!

Plan for some idle capacity - it will save your bacon one day

- Priorities & rescheduling can make this less expensive

Wrapping up



WARNING:
Some of this presentation
was aspirational

We still have a LONG WAY to go.
Fortunately, this is a path we've been
down before.

Kubernetes is Open



open community



open source



open design



open to ideas

<https://kubernetes.io>

Code: github.com/kubernetes/kubernetes

Chat: slack.k8s.io

Twitter: [@kubernetesio](https://twitter.com/kubernetesio)