



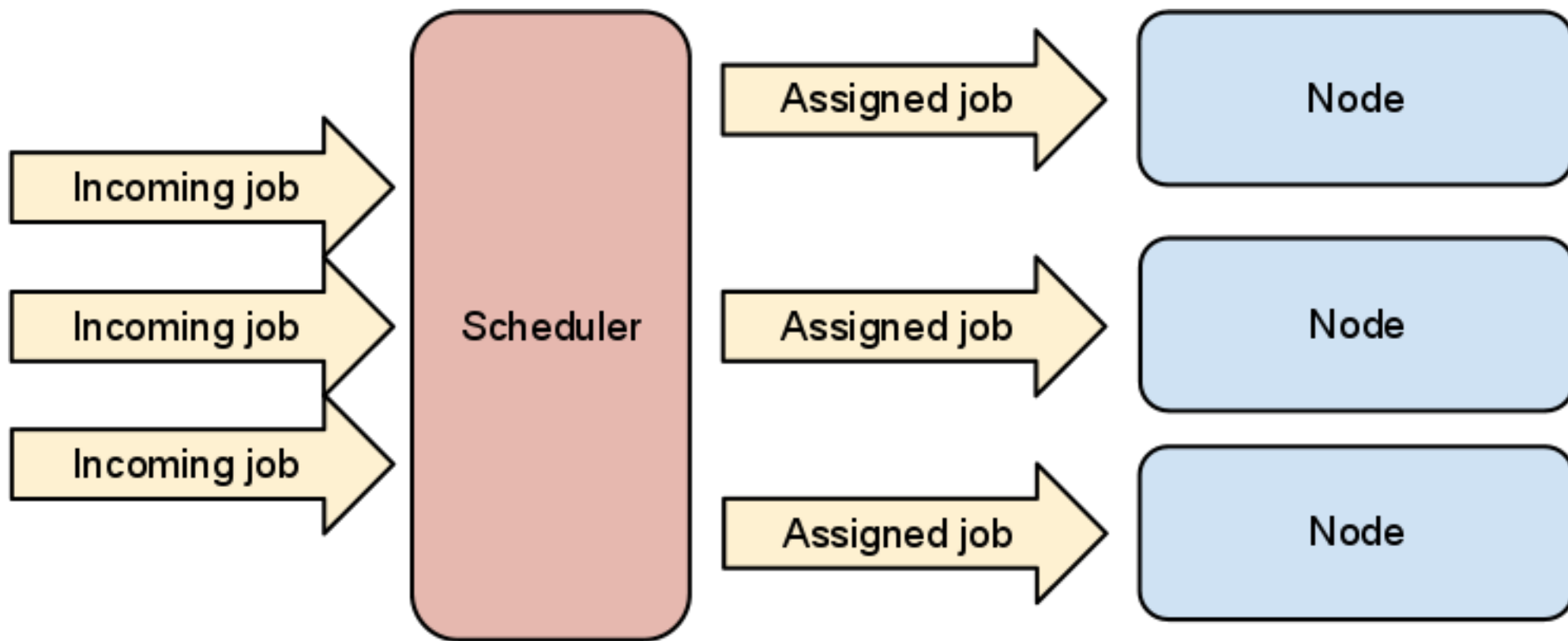
Control Loops In Userspace

Google's Cluster Management Problems

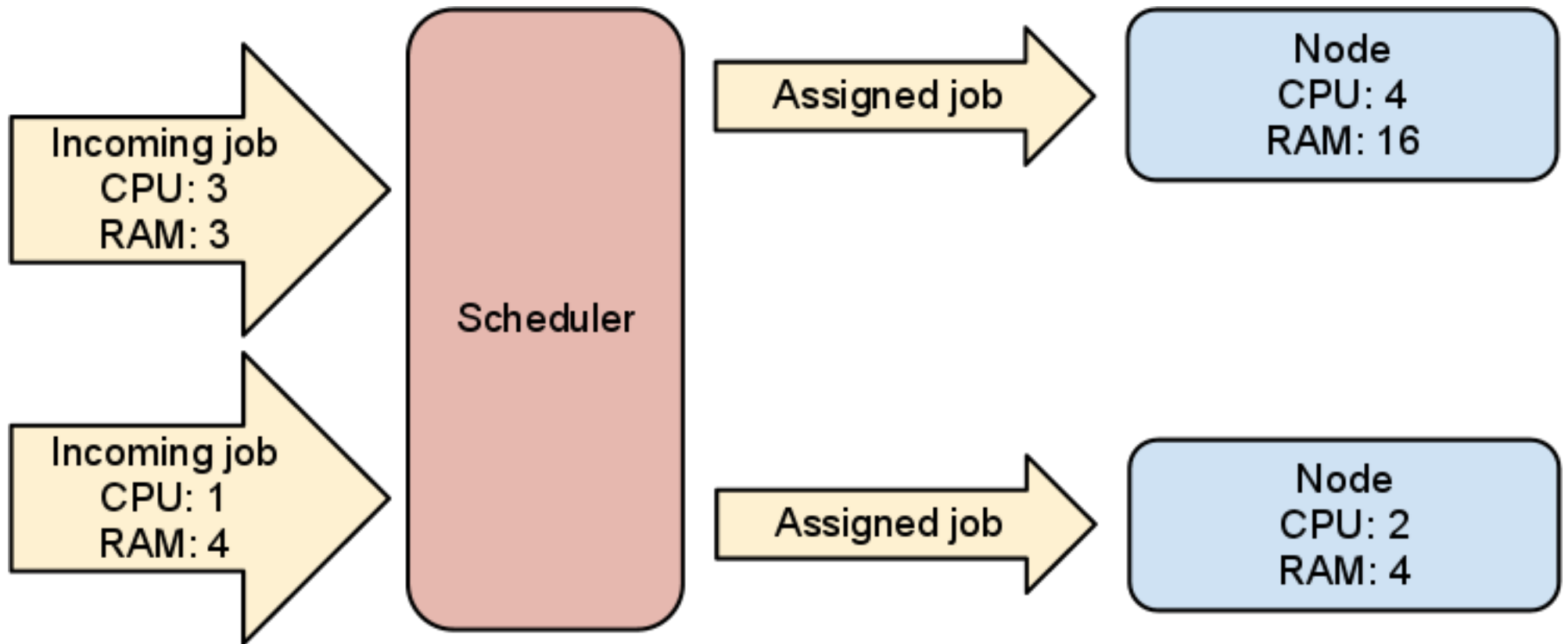
Tim Hockin



Overview



Resources



Schedulable entities

- Task (with a capital T)
 - "Run me now"
 - Consumes node resources immediately
 - Can be stopped
- Alloc (with a capital A)
 - "Make room for me"
 - Consumes no node resources
 - Can be stopped
 - Can have multiple Tasks scheduled "inside"
 - Has "child" resources and "shared" resources

Utilization

- Tasks rarely use 100% of their resources
- Allocs are, by definition, place holders
- Result: overall utilization is low
- Why can't we run more stuff?
- Over-commit is dangerous - some tasks are clearly more important than others (e.g. web search vs a map reduce)
- If we want to do overcommit, we need to be able to react to dynamic load changes

User-facing semantics



- Important jobs get guaranteed resources. Other jobs get best-effort resources
- We call this "tier 1" and "tier 2"
- Tier 2 is "second hand" resources that we have already sold as tier 1, but are not currently being used
- Tier 2 Tasks might be killed at any moment, but on average are not
- Tier 1 resources cost a lot more than tier 2 resources
- We never over-commit tier 1 resources
- We can pack machines with tier 2 resources

Containers

- Tasks and Allocs exist in Containers
- Container != cgroup
- Container == {CPU rate, memory space, disk space, disk bandwidth, net bandwidth, TCP/UDP ports, chroot, ...}
- A Container is a **concept** that achieves our goals

Goals

- Isolation
 - Tasks should not impact each other
 - Behavior of a task should be the same regardless of what else is on the machine
- Predictability
 - Tasks should behave the same each time they run
 - Unless they are specifically configured to use "slack"
- Minimal impact
 - Tasks should not need to know about what we do behind the curtain

Managing over-committed machines



- We watch all running Tasks in near-realtime
- We only sell resources to account for the whole machine at any given time
- If a tier 1 Task has a load spike, a tier 2 Task might be killed to cover it
 - Some resources are "compressible" (CPU) and some are not (RAM)
- We have to react quickly!
- Kernel support is critical, but not always available

Mechanisms

- We like cgroups
 - Easily group tasks together for resource management
 - Easily apply userland policy to manage different resources
 - The kernel can react to conditions very quickly.
 - Need hierarchical cgroups
- We implement control loops in userspace when required
 - These hurt. Complex, racy, inaccurate.
- Other control APIs, e.g. disk quota, /proc/pid/maps, etc.

Real issues

- Memory containment
 - We use fake NUMA for this today
 - Pack pages onto as few nodes as possible
 - Practical limits to how many nodes we can have
 - Node size wastes RAM
 - Requires lots of userspace management
 - Moving to memory cgroup
 - Almost no control loops!
 - Need full hierarchy support
- Shared resource control
 - We manually re-balance CPU time in Allocs
 - Moving toward hierarchical cgroups
 - We manually re-distributed IO bandwidth in Allocs
 - Moving toward hierarchical cgroups

Real issues

- OOM management
 - We have to bring down whole Containers, rather than individual processes.
 - Today we have to trap OOM and do the work in userspace.
- Accounting of shared resources
 - Multiple Tasks in an Alloc map a disk-page. Who gets the bill? We want the Alloc to get billed. We do manual accounting.
 - Multiple unrelated Tasks map a disk-page. Who gets the bill? e.g. libc

Wishlist



- OOM killing for whole cgroups
- Deterministic shared-page billing
- Reduced overhead for cgroups (scalability)
- Hierarchical memory cgroups in all aspects
 - Per-cgroup reclaim based on soft-limit
 - Per-cgroup working-set estimates
- Proportional IO (dist time) hard limits
- Atomic signalling and/or killing of cgroups
- cgroup control of *any* shared resource
 - thread limits, fd limits, ports, network interfaces, etc
 - anything with a system-global limit or impact
- More OOM adjustment granularity
- Highly accurate stats, e.g. wakeup times, per sched_entity
- Backend billing for things like interrupts
- Cache isolation
- Memory and cache bandwidth stats