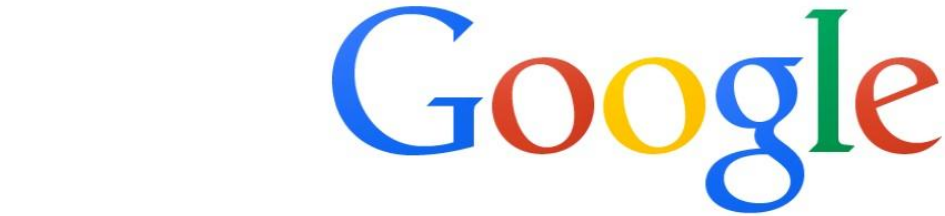


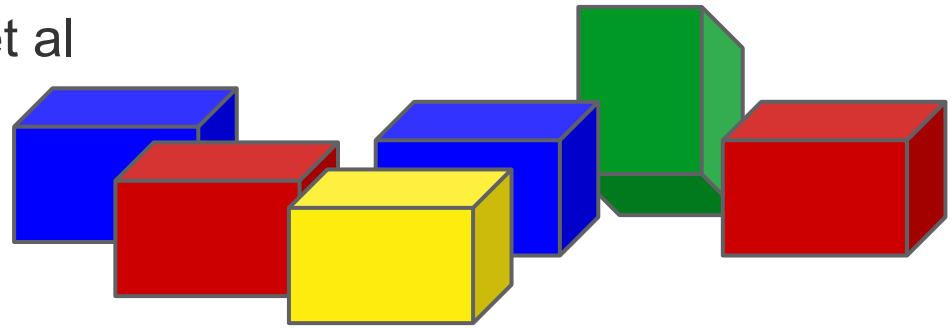
Let Me Contain That For You

Google Search I'm Feeling Lucky

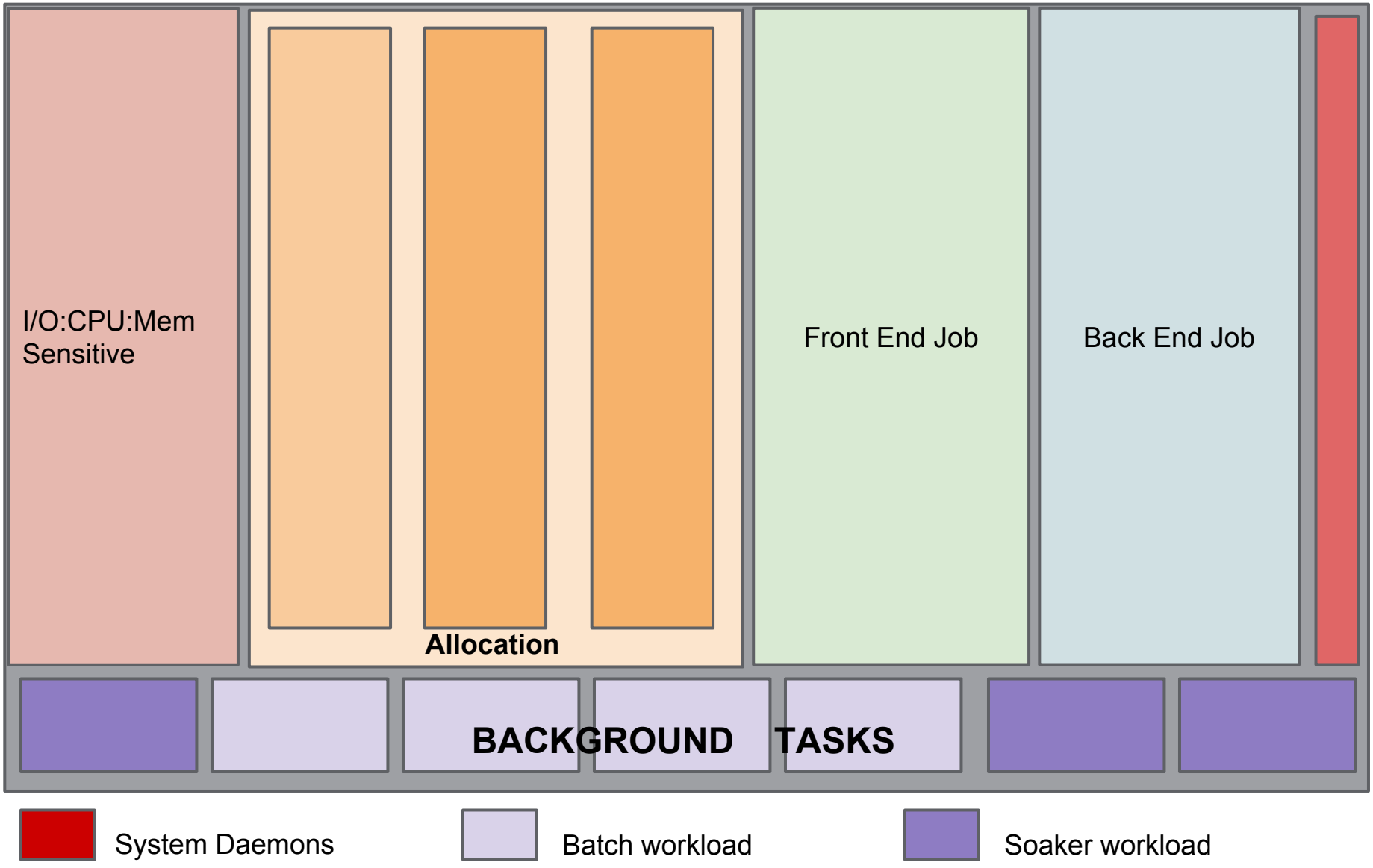
Victor Marmol (vmarmol@google.com)
Rohit Jnagal (jnagal@google.com)

Containers @ Google

- Early users: Scaling process management and isolation.
- What: Linux cgroups + user-space policies and monitoring.
- Everywhere: SaaS, PaaS, IaaS; Private and Public clouds.
- Containerizing shared machines
  - Asymmetric workloads : Latency, bandwidth, and priority
  - Asymmetric Isolation
  - High churn
- Goals:
  - Performance guarantees.
  - High utilization across resources.
  - Shared resources.
  - Overcommitment: Invisible workload from reclaimed resources.
  - Near zero overhead.
- Other use cases: ChromeOS et al



A Shared Google Machine



Google

Resource Isolation

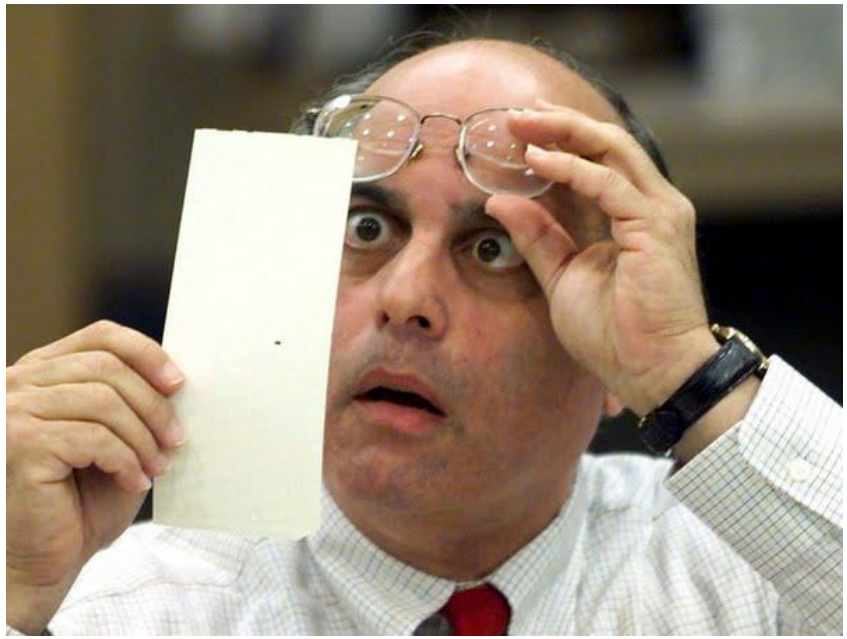
- Quality of service
  - Bandwidth - Fair share, progress guarantees, availability.
  - Latency - wakeup, allocation, access times
  - Priority - Order of importance.
  - Performance: Microarchitecture interference (CPI[2]); Locality
- Solution:
  - Scheduling a good mix.
  - Hierarchical resource management for effective sharing.
  - Maximize utilization across all dimensions.
  - Cgroup-aware tasks:
    - User subcontainers [eg. Query management]
    - User schedulers.
    - Self-correcting tasks: Notifications

image credit

Scalability

- Churn
  - 1 Creation/Deletion per 10 seconds
- Per Container
  - Read: O(10) cgroup-based stats per second
  - Write: O(1) cgroup-based param per second
- Per Machine
  - O(100) containers
  - Looks to grow dramatically
- Overall
  - Read: 1000's per second
  - Write: 100's per second
- Users can do a lot more.

- Precise accounting for chargeback
- Monitoring built in at multiple layers
- Extremely low overhead



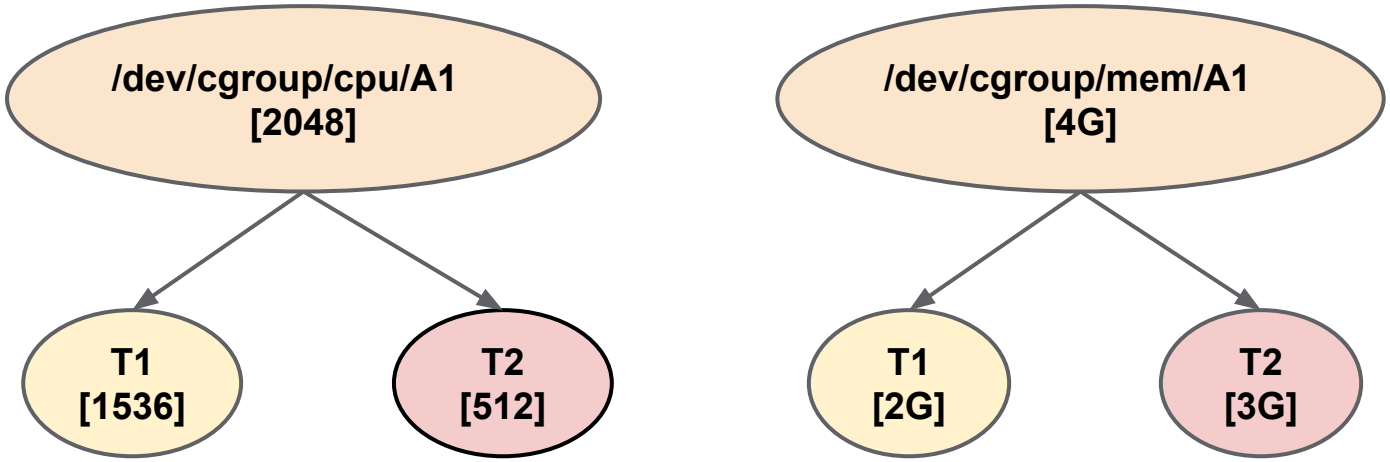
Google

Let Me  That For You

- Revised container management
  - Separate cgroup abstraction from policies.
  - Configuring cgroups with an intent-based resource specification.
- Built for scalability and parallel access.
- Also includes extra kernel patches for:
  - Improving resource isolation.
  - Providing tighter performance guarantees.
  - Precise accounting in face of sharing.
  - Cap for global resources.
- Allow users to create subcontainers with restrictions.
- Open-source: Sharing use-cases, problems, and benchmarks.
- Implement policies in a higher layer:
  - Continuous monitoring and fine-tuning.
  - No critical loops [Remember LPC2011?]
  - Machine-level utilization and isolation management.
  - Isolated from system APIs.

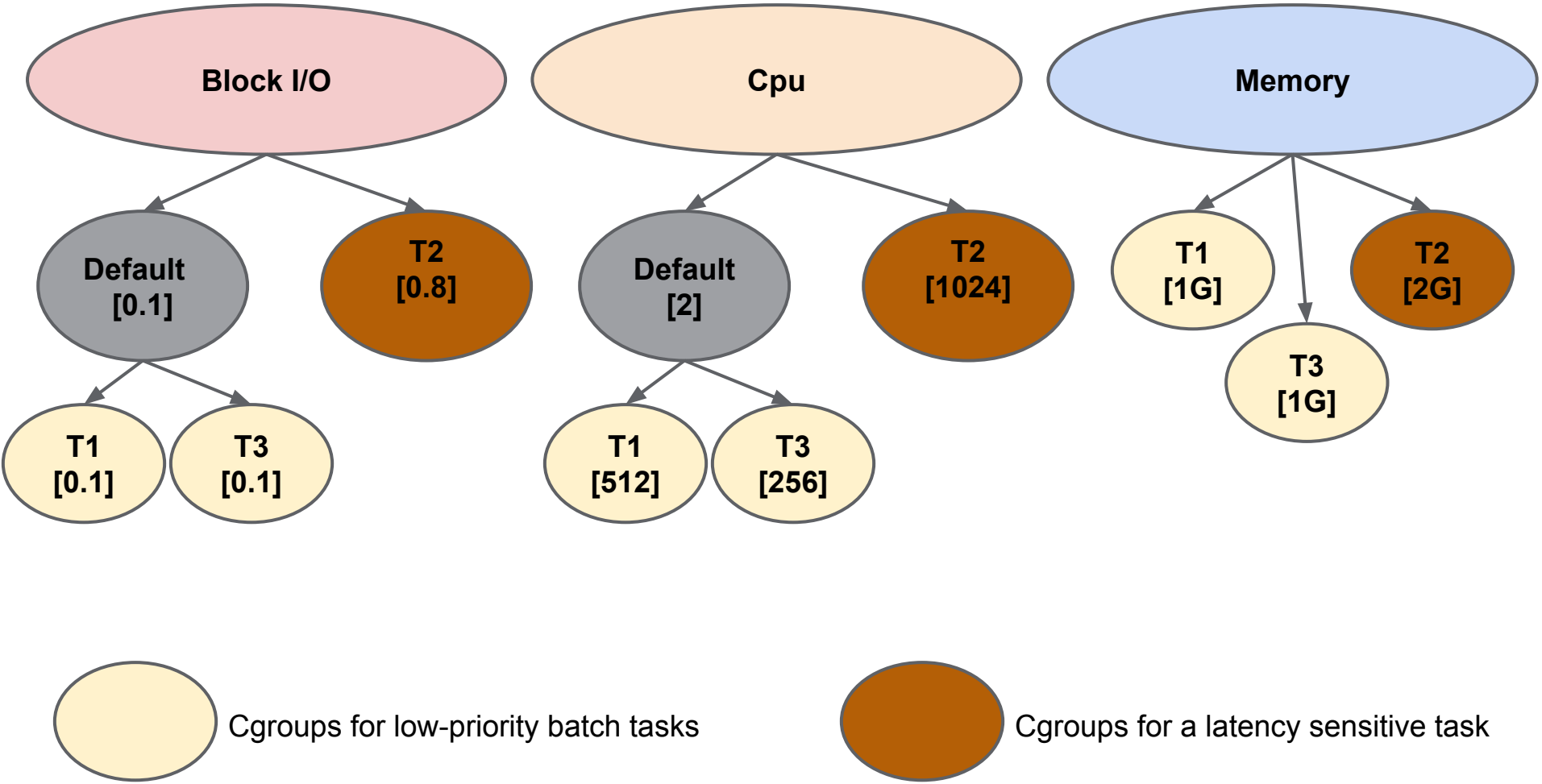
Hierarchical Sharing

An allocation **A1** with two tasks **T1** and **T2**

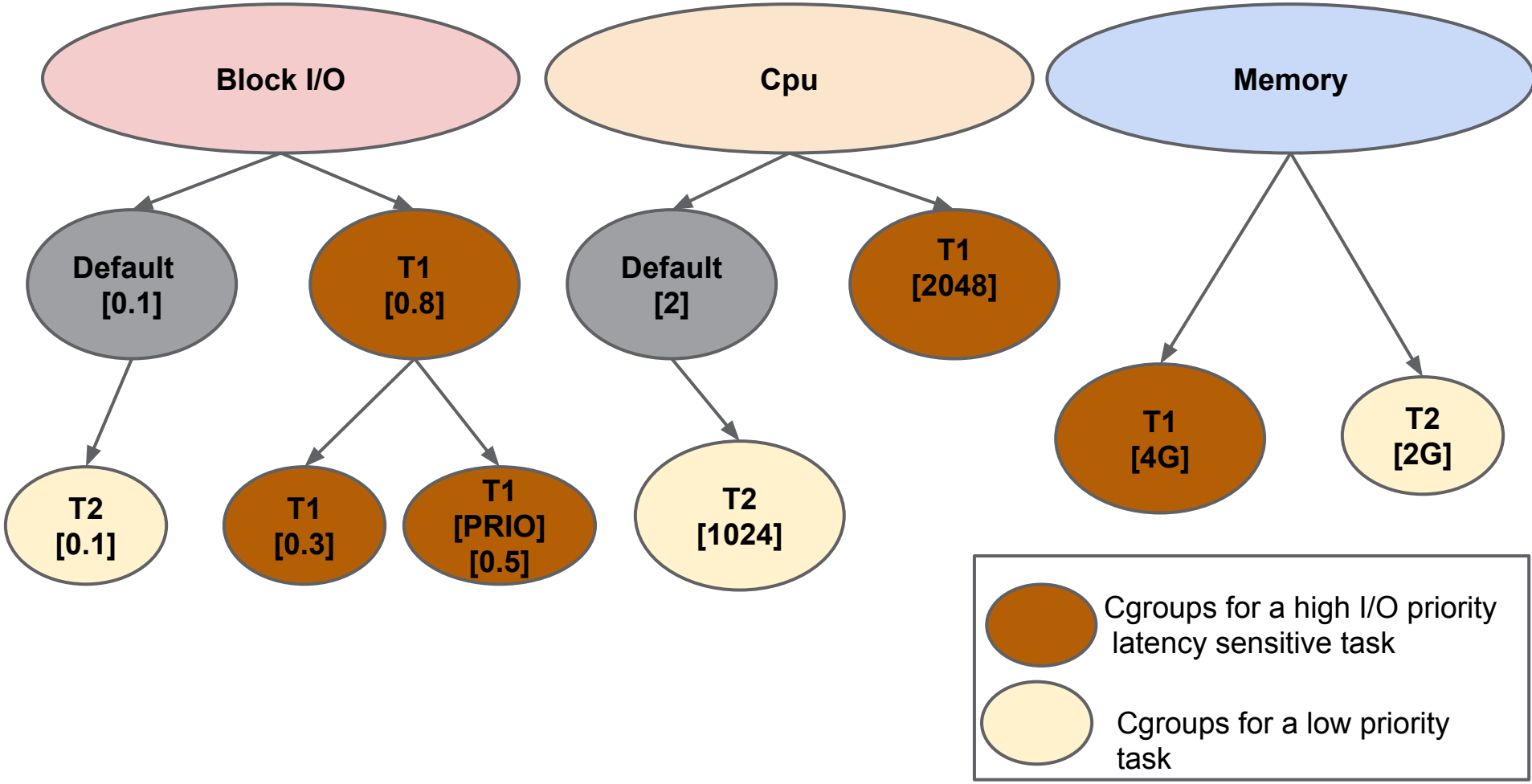


Task running in an allocation sharing resources with co-located siblings.

Managing priority across resources

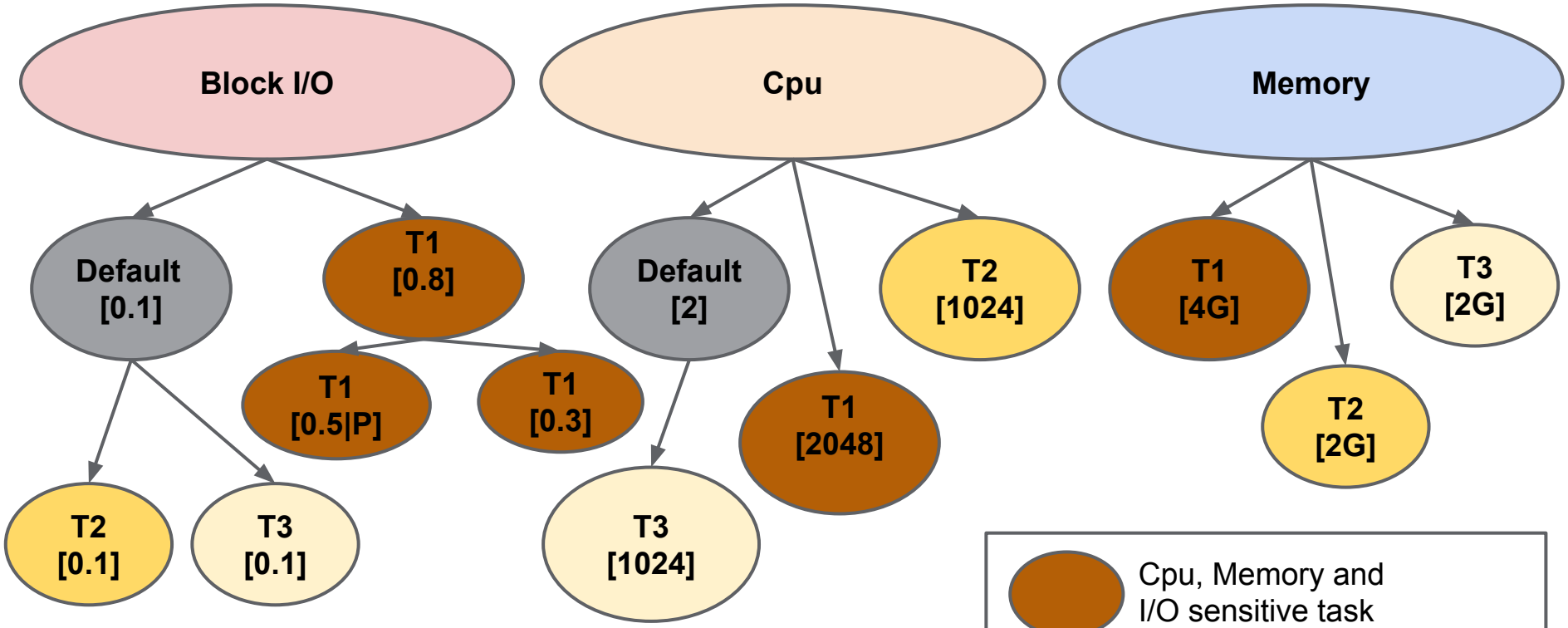


Google Managing priority across resources



A task may require multiple containers for the same resource to balance its workload priorities. I/O server T1 uses two subcontainers to differentiate incoming I/O requests and moves threads to the right subcontainer.

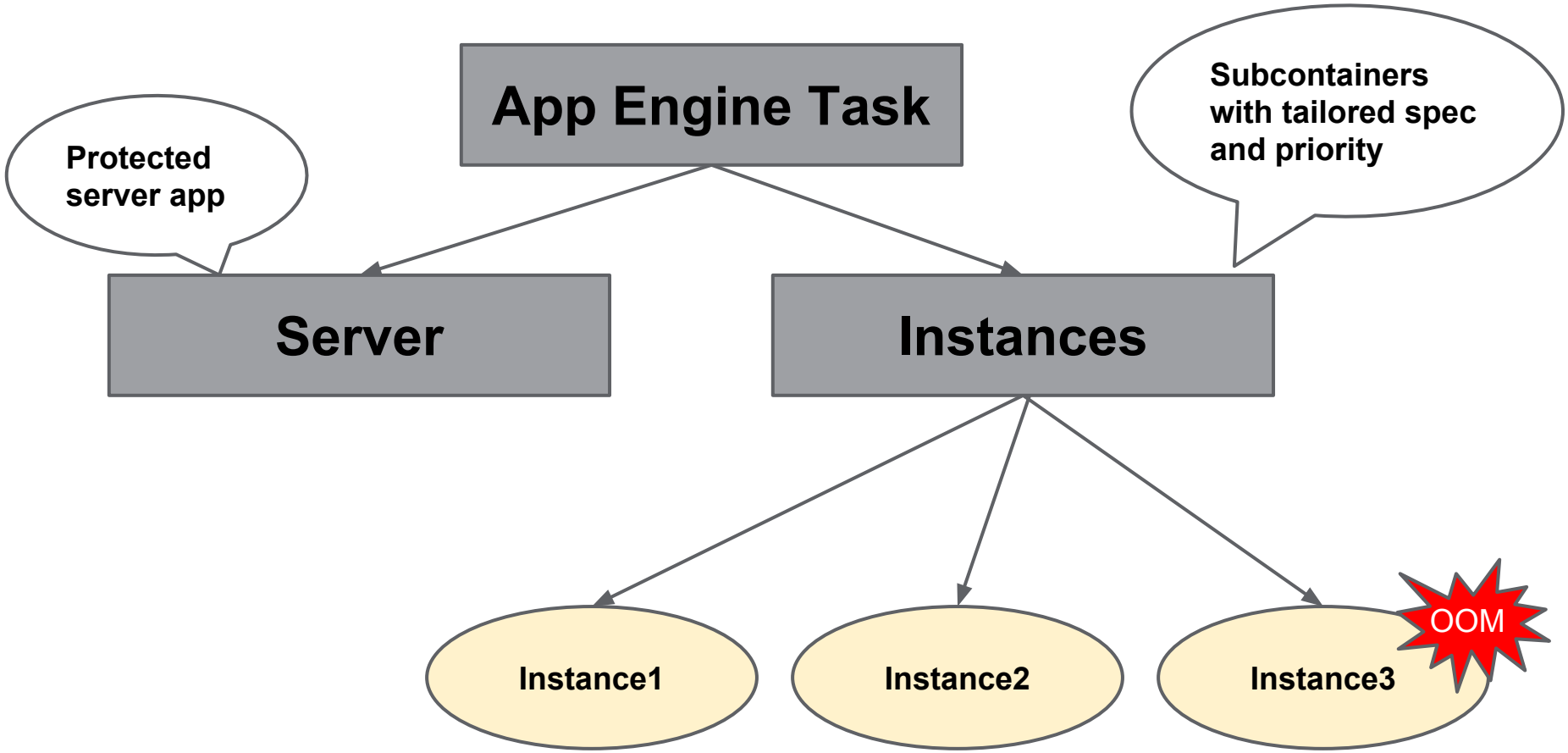
Splitting hierarchies for performance



Splitting hierarchies reduces stranded resources and improves performance for highly sensitive tasks.

Google

User Subcontainers



App Engine uses on-demand container creation:
fair sharing, notifications, and isolation of misbehaving apps

Takeaways

- Cgroups support goes beyond containerized VMs.
- Sharing and overcommitment is a key to higher utilization.
- Managing each resource separately helps fine-tune utilization and performance.
- More power to users means better flexibility and scalability.

Come find us for chat, discussions, BoF, and drinks.
Or virtually:

jnagal@google.com
vmarmol@google.com