6.5810: Datacenter Software

Adam Belay <abelay@mit.edu>

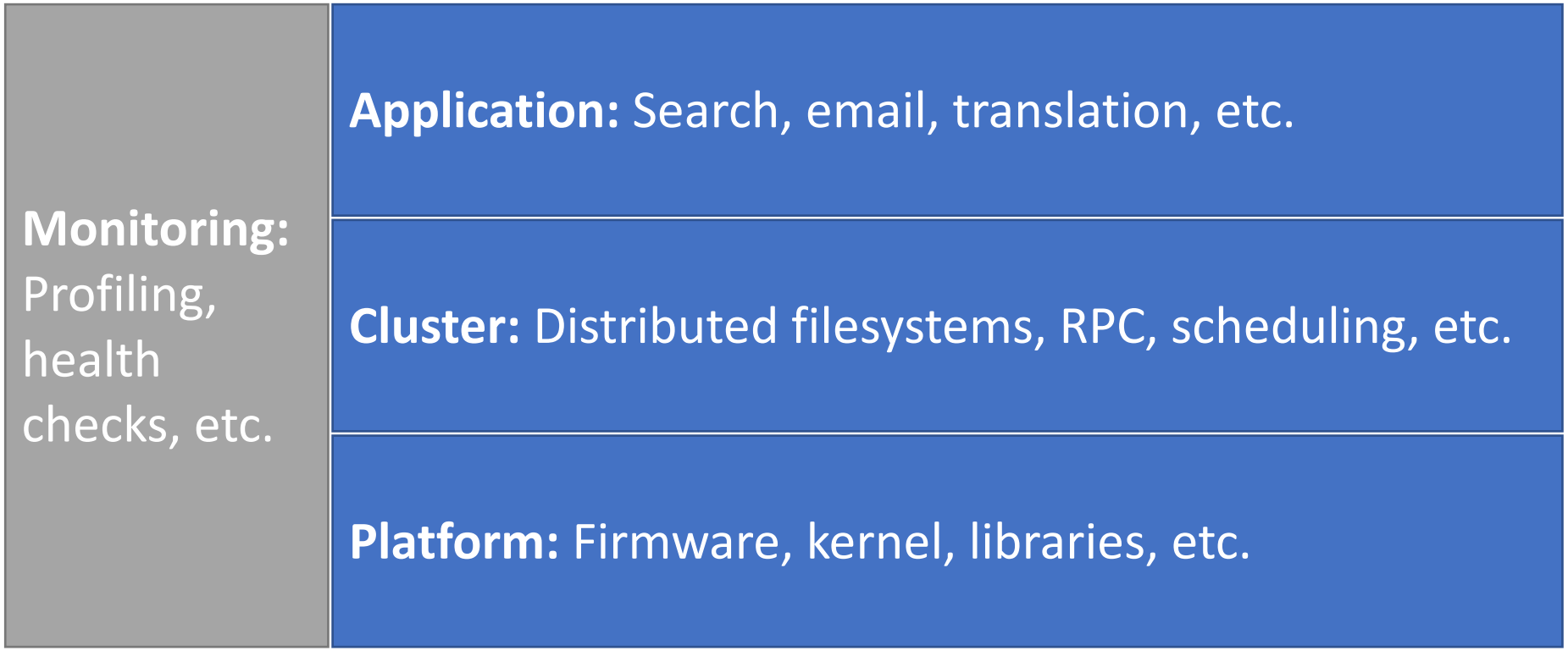
Class logistics

- Please sign up for a presentation slot by the end of the week
  - Sign up sheet will be posted later today
- Reminder: Post your questions about paper readings the night before
  - (no more reading assignments this week)
- Agenda today:
  - Datacenter software stack
  - CloudLab

Warehouse-scale computing

- A single “service” can rely on thousands of machines
- Failures are the common case, services must survive them and maintain high uptime (e.g., 99.99th percentile)
- OS functions are no longer about single machines; think of the entire cluster as an OS
  - Today’s focus

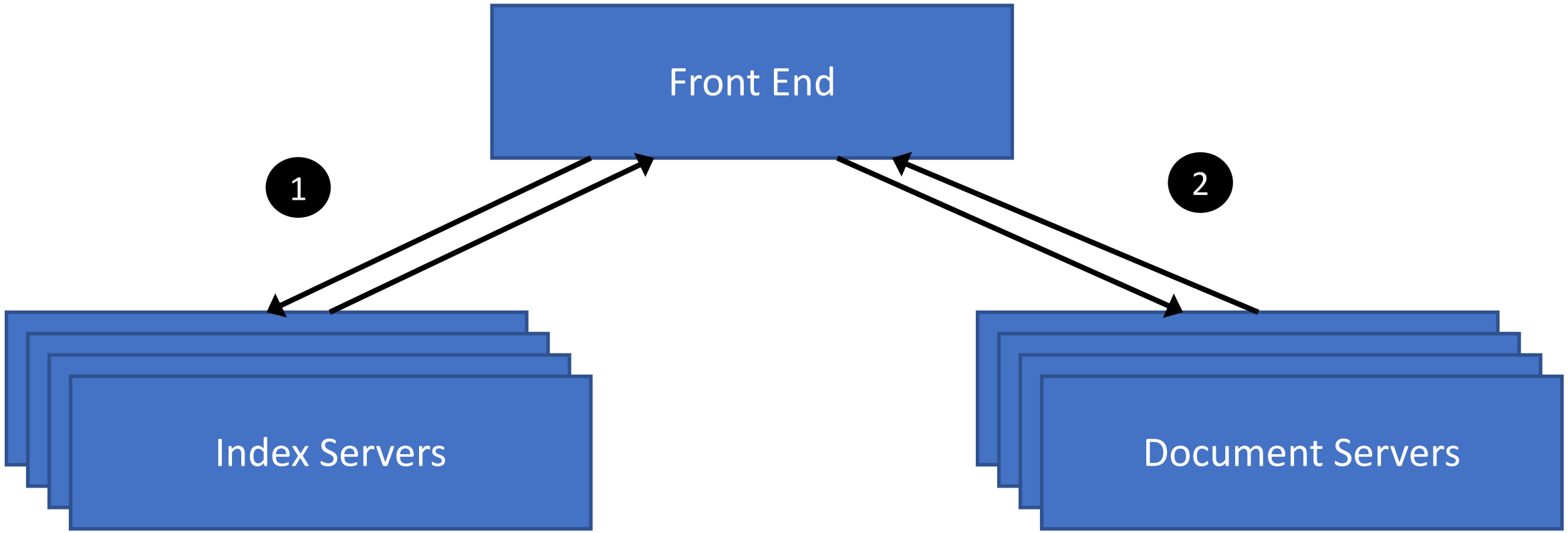
WSC Software Stack



Example application: Search

- A single query reads hundreds of megabytes of data and consumes 10’s of billions of compute cycles
- Example of *fan-out* access pattern
- Latency sensitive workload
- See: Web Search for a Planet: The Google Cluster Architecture. Barosso et. Al. for more details

Search architecture (simplified)



Lifecycle of a search request

1. DNS load balancing steers to a cluster (globally distributed)
2. Hardware load balancer chooses a web server (see Maglev NSDI'16)
3. Web server receives HTTP request
4. Index Phase: Search words -> list of document IDs
   - Intersect document ID lists per word, sort by relevance score
5. Document Phase: document IDs -> {title, summary, URL}
6. Web server generates HTTP response

Challenges for search

- Tens of terabytes of uncompressed data (Back in 2003)
  - Can’t fit in a single server
- Solution sharding
  - Divide index and documents into pools
  - Each pool contains replicas of machines with the same data
  - Provides throughput and fault tolerance
- Today: Enormous fan-out needed for search
  - Tail latency impacts search response and quality

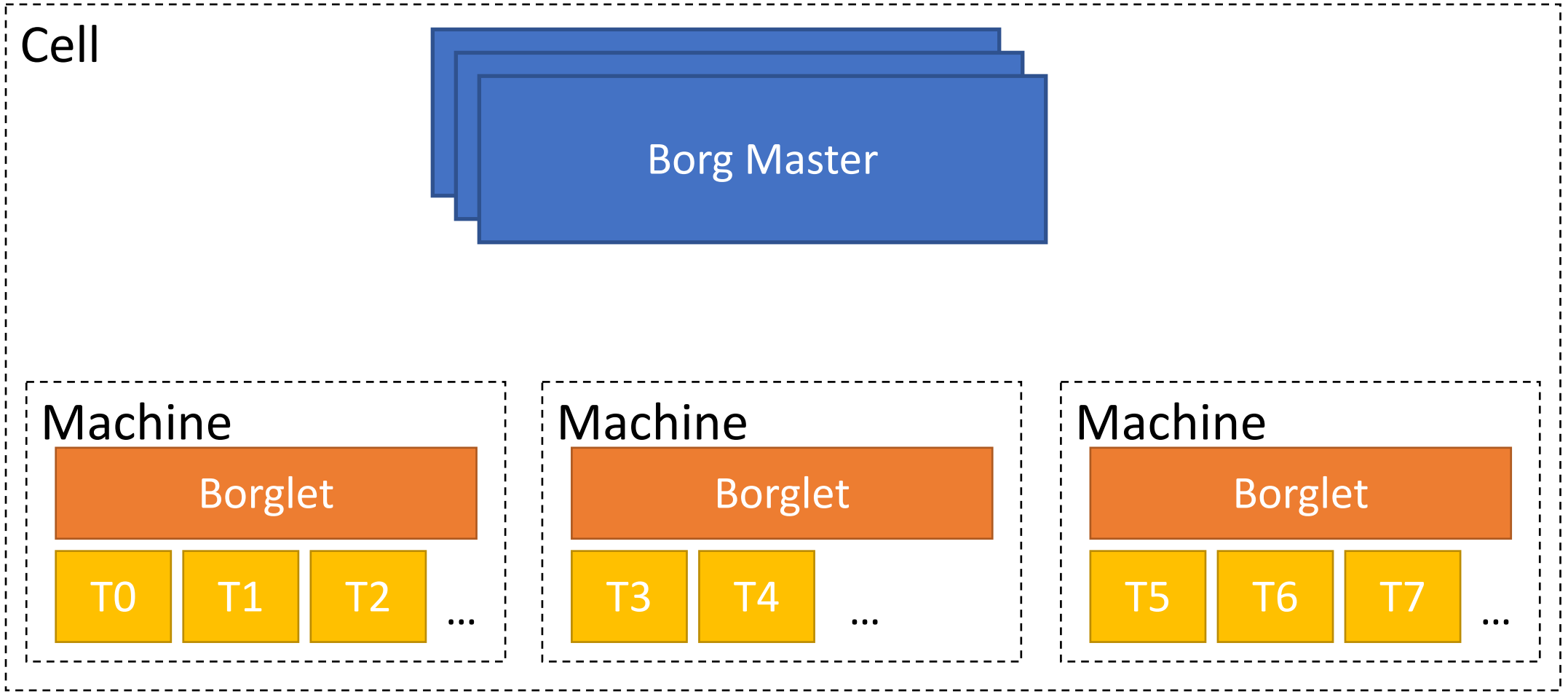
Example cluster: Borg (Cluster scheduling)

- Goal: Hide complexity of resource management and fault tolerance from users across thousands of machines
- Users submit *jobs* to Borg, each job is a collection of *tasks*, that all run the same binary across machines
- Each job runs in a *cell*, a collection of machines
- Each cell runs a combination of best effort and latency critical tasks
- See: Large-scale cluster management at Google with Borg. Verma et. Al. EuroSYS'15 for more details

How Borg selects machines?

- Machines vary in CPU, disk, memory, external IP access, flash, etc.
- Each Job has constraints on which type of machine a task can use
  - Includes CPU cores, RAM bytes, disk space and bandwidth etc.
  - Best effort jobs often request 0.1 CPU cores
- Borg must resolve this scheduling problem within each cell
- Job configuration changes are possible; often result in restarted tasks

Borg architecture



Borg components

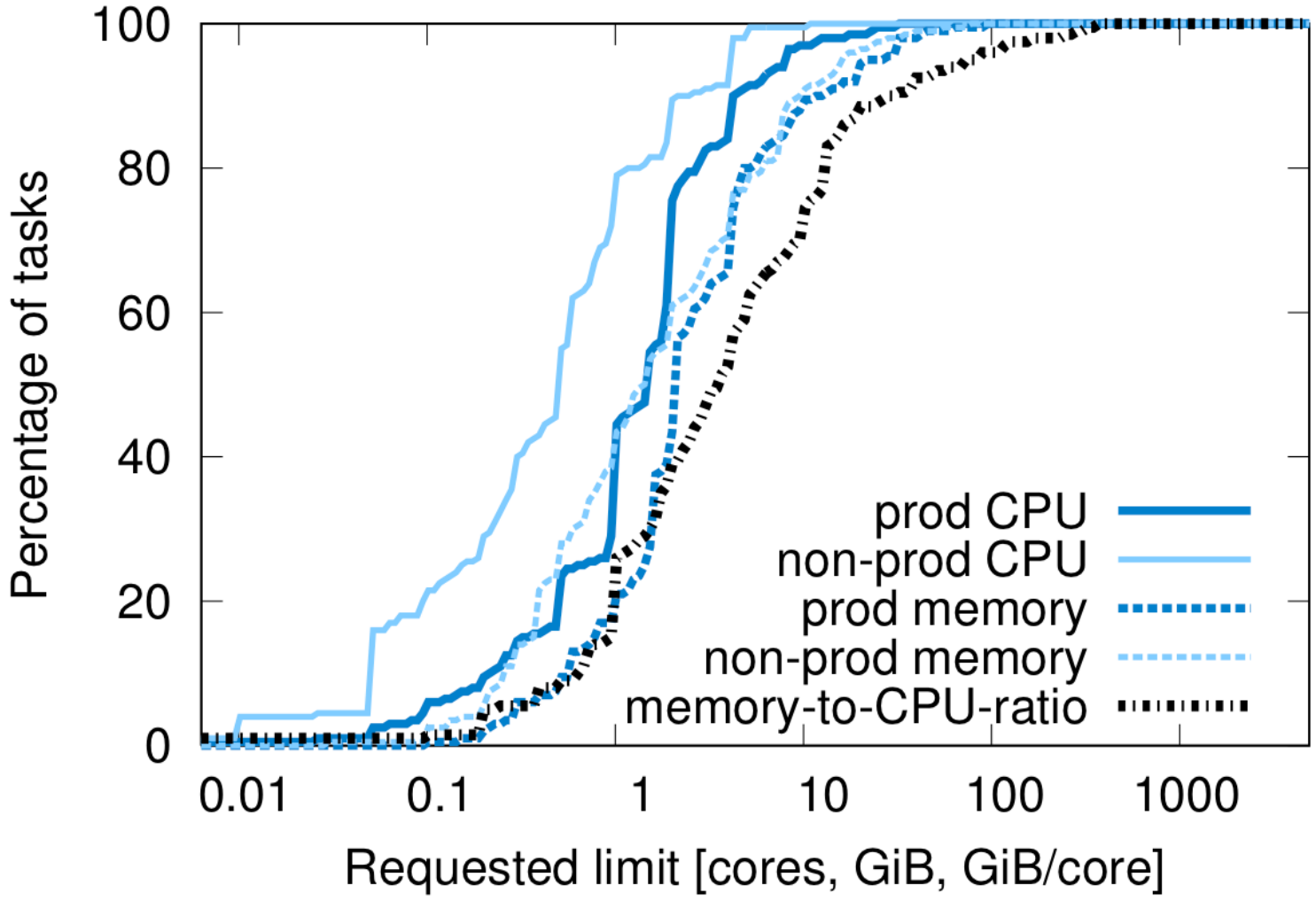
Borgmaster:

- Maintains a replicated task queue
- Places tasks on available machines
- Maintains queues of different priorities

Borglet:

- Launches tasks on machines (in containers)
- Restarts tasks if they fail
- Monitors state of machine

Q: How many resources do tasks request?



Large-scale cluster management at Google with Borg. Verma et. Al. EuroSYS'15

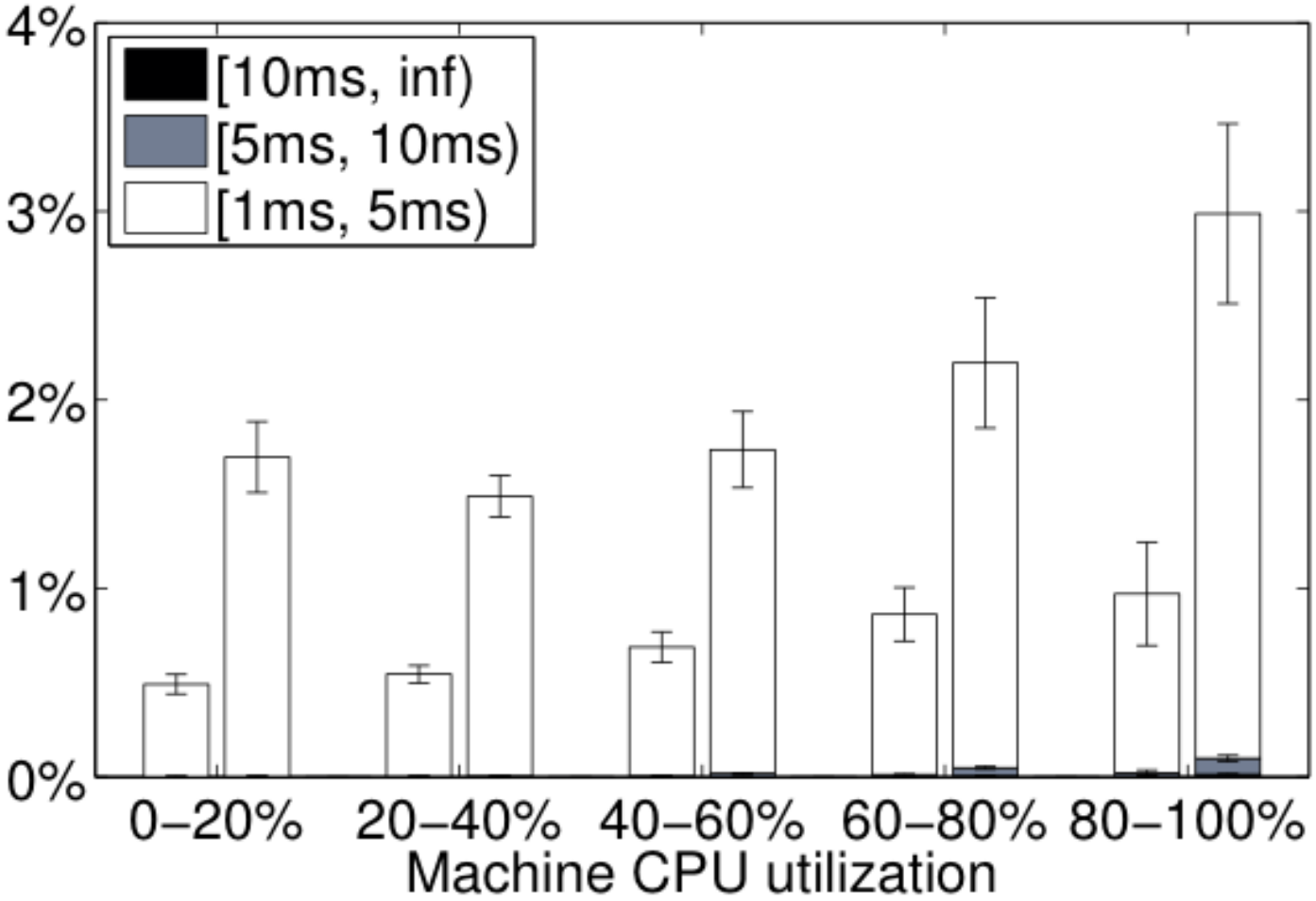
Resource reclamation

- Each task requests a resource *limit*
  - Task is killed if it uses more RAM or throttled if it uses more CPU
- Issue: Utilization: Large gap between limit and actual use
- Borglet monitors actual use and tells borgmaster
- Borgmaster places best effort tasks in gap between limit and use of latency critical tasks (with a safety margin)

Platform example: Linux Kernel

- Linux is a popular OS platform for datacenters
- A typical machine handles more than 9 tasks, more than 4500 threads
- Linux cgroups used to provide memory and performance isolation
- Compressible resources (e.g., CPU cycles, disk bandwidth)
  - Can be rate limited for degraded performance
- Uncompressible resources (e.g., RAM and disk storage)
  - Task must be killed if they run out

Linux isolation: Good but some interference

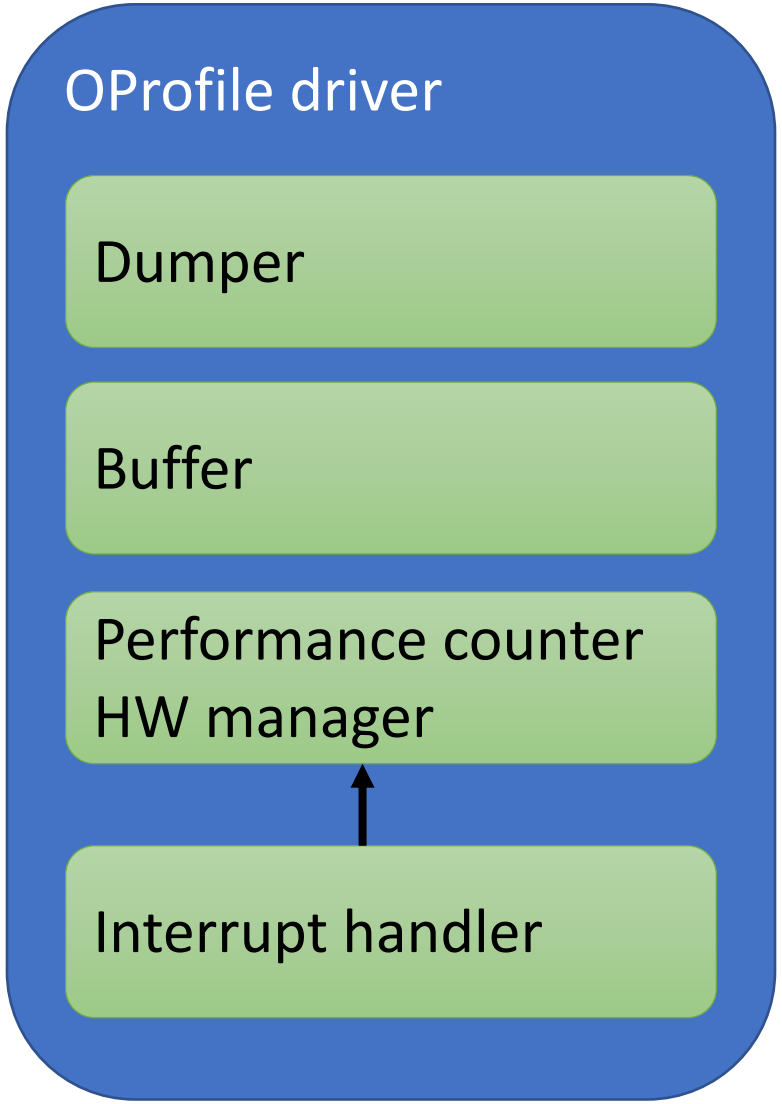


Large-scale cluster management at Google with Borg. Verma et. Al. EuroSYS'15

Monitoring example: Clusterwide profiling

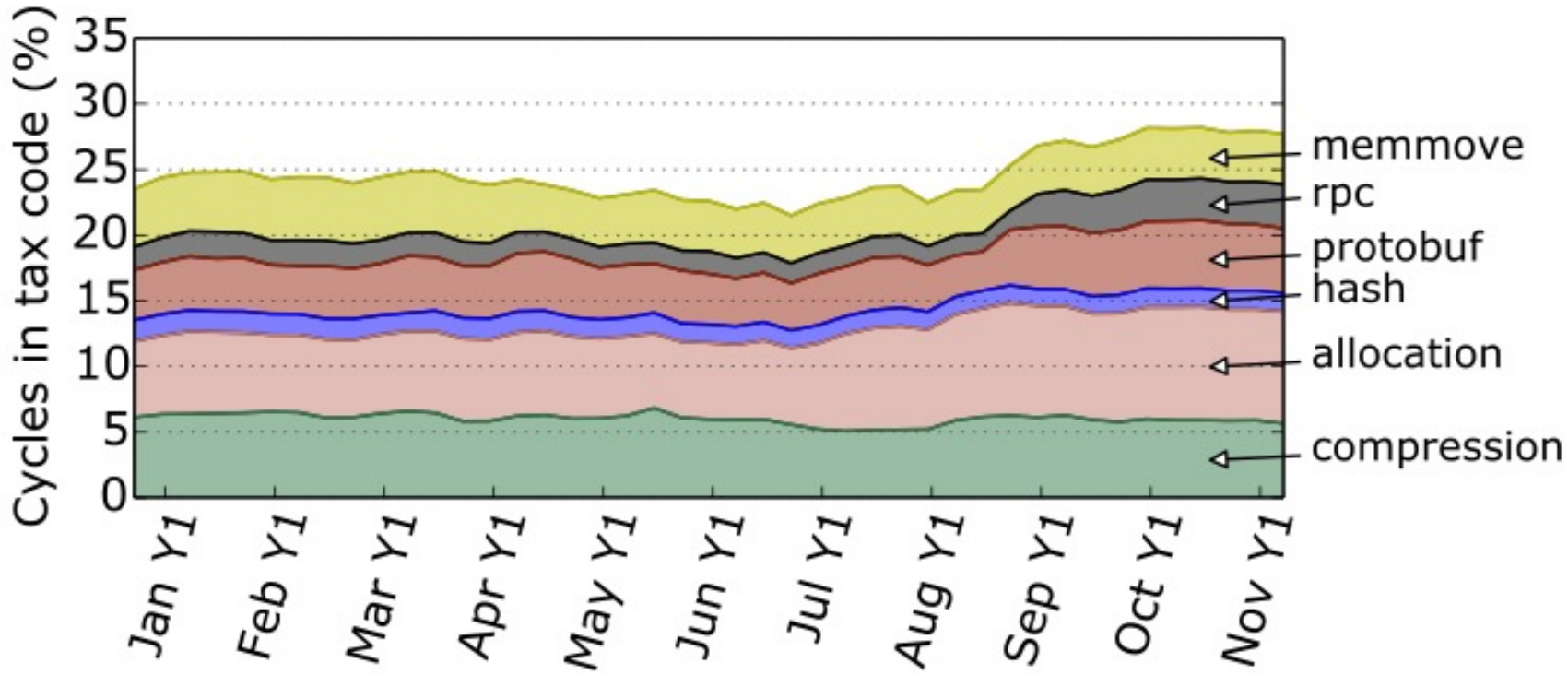
- Challenge: What are the hottest code regions? What locks are most contended? What processes are hogging memory?
- Solution periodically sample resource use on all machines
- Allows for code optimization at a datacenter level
- See: GOOGLE-WIDE PROFILING: A CONTINUOUS PROFILING INFRASTRUCTURE FOR DATA CENTERS. Ren et. Al. for more details

Statistical profiling



- Timer interrupt periodically fires
- Measure current IP, performance counters, memory use, etc.
- Record to a log
- Use debug symbol data to convert to functions in code

Library overhead (from last lecture)



Svilen Kanev et. Al. Profiling a warehouse-scale computer. ISCA'15

Conclusion

- Datacenter software components: Applications, Cluster, Platform, and Monitoring
- Saw just one example from each today
- Many large research challenges for all four layers

MIT 6.5810: Cloudlab Overview

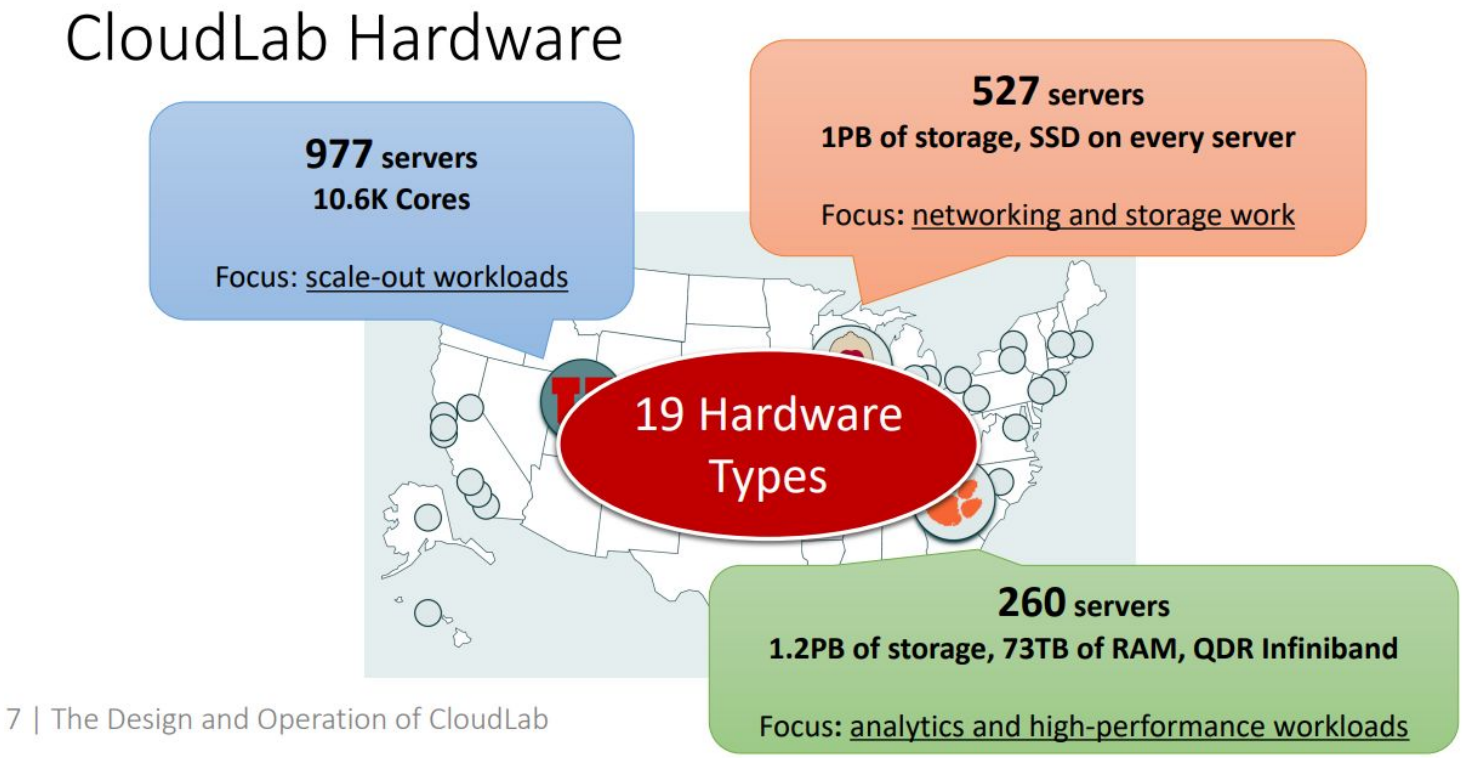
Zain Ruan (zainruan@mit.edu)

Office Hours

- For lab assignments.
- Time: 4:00 pm-6:00 pm Monday, starting next week.
- Location: 32-G9 lobby.

What Is Cloudlab?

- https://www.cloudlab.us/
- A shared cloud infrastructure for research and education in computer systems.
- The CloudLab clusters have almost 15,000 cores distributed across six sites in the United States.



The Design and Operation of CloudLab [ATC' 19]

For more information: https://docs.cloudlab.us/hardware.html

Who Use Cloudlab?

- Course instructors and students.
  - For doing lab assignments and final projects.
- System researchers.

Networking	30%
Security	16%
Storage	11%
Applications	10%
Computing	9%
Virtualization	8%
Databases	7%
Middleware	4%
Energy & Power	2%
Other	15%

* Based on 93 papers from 2017-2018

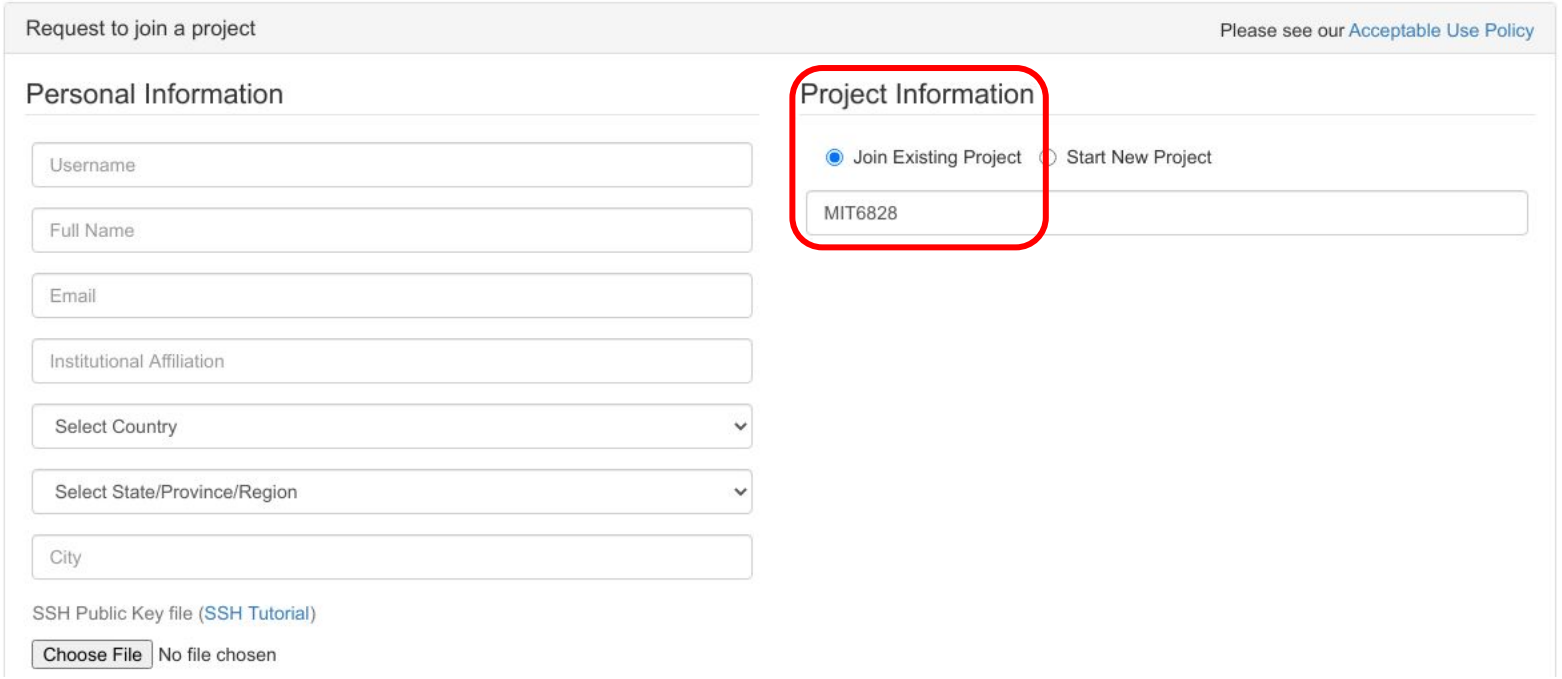
The Design and Operation of CloudLab [ATC' 19]

Why Use Cloudlab?

- Cost reasons.
  - No access to enough private servers.
  - No access to specific hardwares (e.g., GPUs).
- Performance isolation.
  - System research requires accurate measurements.
  - Cloudlab provides access to **bare-metal** instances instead of VMs.
- Full control.
  - Able to customize library, OS, and even BIOS.

Create Cloudlab Account

- https://www.cloudlab.us/signup.php



Launch A Cloudlab Instance

- Decide your instance type and check its availability
  - https://www.cloudlab.us/resinfo.php
- “Start Experiment” (i.e., ASAP)
  - The default expiration time is 16 hours.
    - But extensions can be requested.
  - Once expired, old data are discarded.
    - Backup data. Write a script to rebuild environment automatically.
    - Or create your own disk image (snapshot).
- “Reserve Nodes”
  - For the machine not available right now.
  - For a longer machine time, e.g., >one week.
  - Some reservations need cloudlab administrator’s approval.

Demo

- Apply for an account.
- Create experiment profile.
- Launch an instance.
- Create disk image.
- SSH example.
- Use web serial console.