CPSC 416 Distributed Systems

Winter 2023 Term 1 (November 23, 2023)

Tony Mason (fsgeek@cs.ubc.ca), Lecturer

Logistics

Teaching Assistants

Andy Hsu (andy.hsu@alumni.ubc.ca)

Hamid Ramezanikebrya (hamid@ece.ubc.ca)

Jonas Tai (jonastai@student.ubc.ca)

Cathy Yang (kaiqiany@student.ubc.ca)

Office Hours

Remember: Use Piazza for **all** official course-related communications

- Not on Piazza? Not official.
- Canvas “comments/messages” **are not monitored**

Office Hours:

Who	When	Where
Tony	Monday 14:00-15:00 Wednesday 16:00-17:00	Discord
Andy	Thursday 19:00-20:30	Discord
Hamid	Friday 16:30-18:00	Kaiser 4075
Jonas	Thursday 13:00-14:00	Online (see Piazza)
Cathy	Friday 09:00-10:30	X237

Assessment

This week

- No (more) deadlines

Next week

- Capstone Status Report (Tue @ 17:00)
- Design Recipe Self-Assessment (Thu @ 17:00)

Note:

- You are strongly encouraged to collaborate with others on this
- You should use tools at your disposal to answer these questions
- **Do not forget to submit it.**

Today’s Failure

Microsoft Azure

Date: September 16, 2023

Time: 07:24 UTC

Source: https://azure.status.microsoft/en-us/status/history/

US East Region scale units in a single availability zone lost power, rebooted.

A *subset* failed to come back online after reboot.

Impact:

- “[S]ome customers using proxy mode connection may have experienced impact, due to one connectivity gateway not being configured with zone-resilience.”

Microsoft Azure (September 16, 2023)

Impact:

- “SQL Databases with ‘auto-failover groups’ enabled were failed out of the region, incurring approximately eight hours of downtime prior to the failover completing.
- “SQL Databases with ‘active geo-replication’ were able to self-initiate a failover to an alternative region manually to restore availability.”

Root Cause: “this incident was initially triggered by a UPS rectifier failure on a Primary UPS.”

Microsoft Azure September 16, 2023

Secondary Failure(s):

The UPS was connected to three Static Transfer Switches (STS) – which are designed to transfer power loads between independent and redundant power sources, without interruption. The STS is designed to remain on the primary source whenever possible, and to transfer back to it when stable power is available again. When the UPS rectifier failed, the STS successfully transferred to the redundant UPS – but then the primary UPS recovered temporarily, albeit in a degraded state. In this degraded state, the primary UPS is unable to provide stable power for the full load. So, after a 5-second retransfer delay, when the STS transferred from the redundant UPS back to the primary UPS, the primary UPS failed completely.

While the STS should then have transferred power back to the redundant UPS, the STS has logic designed to stagger these power transfers when there are multiple transmissions (to and from primary and redundant UPS) happening in a short period of time. This logic prevented the STS from transferring back to the redundant power, after the primary UPS failed completely, which ultimately caused a power loss to a subset of the scale units within the datacenter – at 07:24 UTC, for 1.9 seconds. This scenario of load transfers, to and from degraded UPS, over a short period of time, was not accounted for in the design. After 1.9 seconds, the load moved to the redundant source automatically for a final time. Our onsite datacenter team validated that stable power was feeding all racks immediately after the event, and verified that all devices were powered on.

A previously discovered bug that applied to some of our BIOS software led to several hosts not retrying to connect to a PXE server, and remaining in a stuck state. Although this was a known issue, the initial symptoms led us to believe that there was a potential issue with the network and/or our PXE servers – troubleshooting these symptoms led to significant delays in correlating to the known BIOS issue. While multiple teams were engaged to help troubleshoot these issues, our attempts at force rebooting multiple nodes were not successful. As such, a significant amount of time was spent exploring additional mitigation options. Unbeknownst to our on call engineering team, these bulk reboot attempts were blocked by an internal approval process, which has been implemented as a safety measure to restrict the number of nodes that are allowed to be forced rebooted at one time. Once we understood all of the factors inhibiting mitigation, at around 16:30 UTC we proceeded to reboot the relevant nodes within the safety thresholds, which mitigated the BIOS issue successfully.

Microsoft Azure September 16, 2023

Monitoring Deficiency:

> Throughout this incident, we did not have adequate alerting in place, and could not determine which specific VMs were impacted, because our assessment tooling relies on a heartbeat emitted from the compute nodes, which were stuck during the boot up process. Unfortunately, the time taken to understand the nature of this incident meant that communications were delayed. For customers using Service Bus and Event Hubs, this was multiple hours. For customers using Virtual Machines, this was multiple days. As such, we are investigating several communications related repairs, including why automated communications were not able to inform customers with impacted VMs in near real time, as expected.

Post Incident Review Video: https://www.youtube.com/watch?v=VU0XttRVyOg

Lesson Goals

Data Center Services

Trends and Services

High-speed RDMA and programmable networks

Resource heterogeneity

Resource disaggregation

Resource management and orchestration

Trends

Moore's Law:

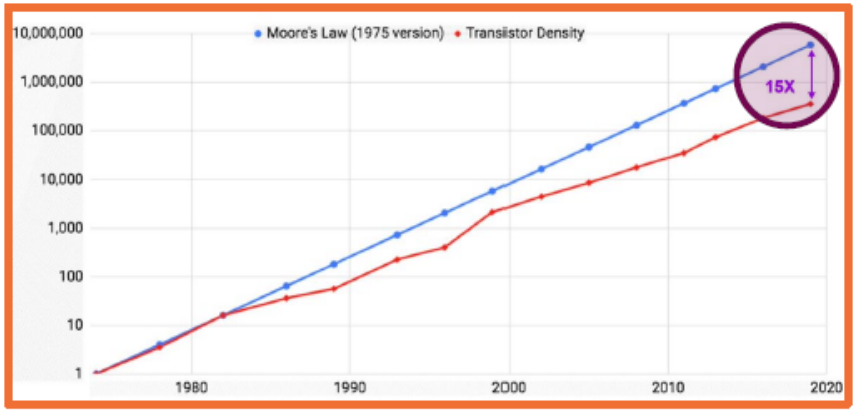
- Economy of scale
- Performance and scale with commodity components

Specialization/heterogeneity

- GPUs, TPUs, etc.
- New memory
- New Storage classes

Disaggregation

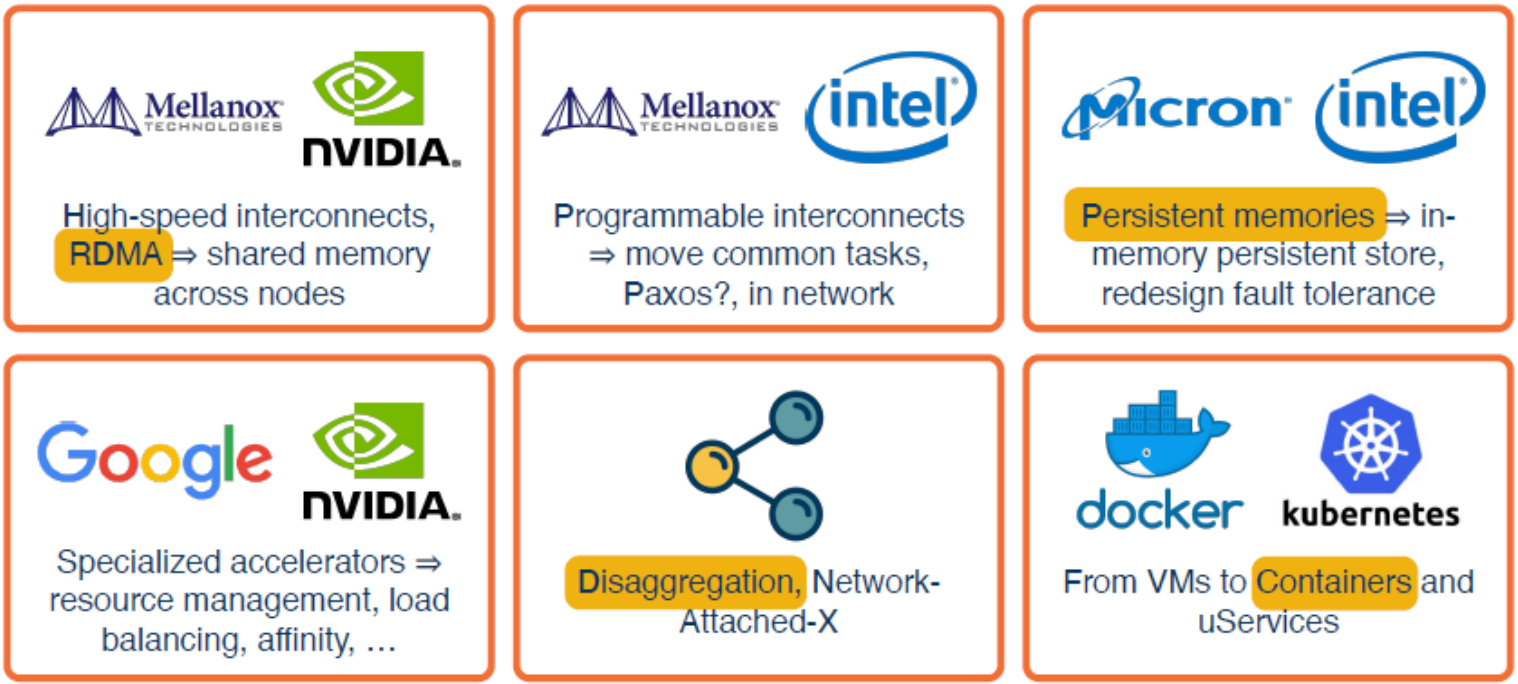
- Independently scaled tiers of different resources



Moore, Gordon E. "No exponential is forever: but 'Forever' can be delayed!" Solid-State Circuits Conference, 200.

New Technologies

Limitation of (x86+DRAM+Ethernet) + scale and requirements of emerging workloads:



Remote Direct Memory Access (RDMA)

Remote DMA

Bypass CPU

- Data access/communications via interconnect support (“network”)

Benefits (versus commodity Ethernet):

- Higher Bandwidth
- Lower latency

Tradeoffs:

- Costs

RDMA

Original idea: 1990s research

Virtual Interconnect Architecture (VIA): 2000s

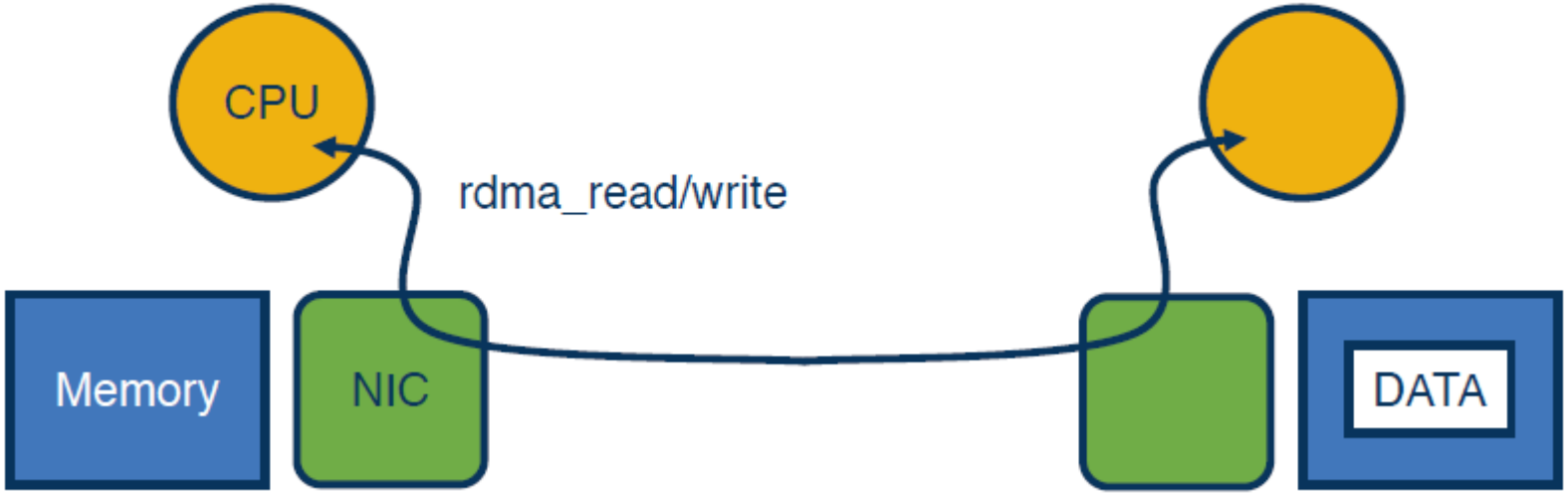
Mellanox: Infiniband

- Today: approx. 50% of top 500 machines, many data centers

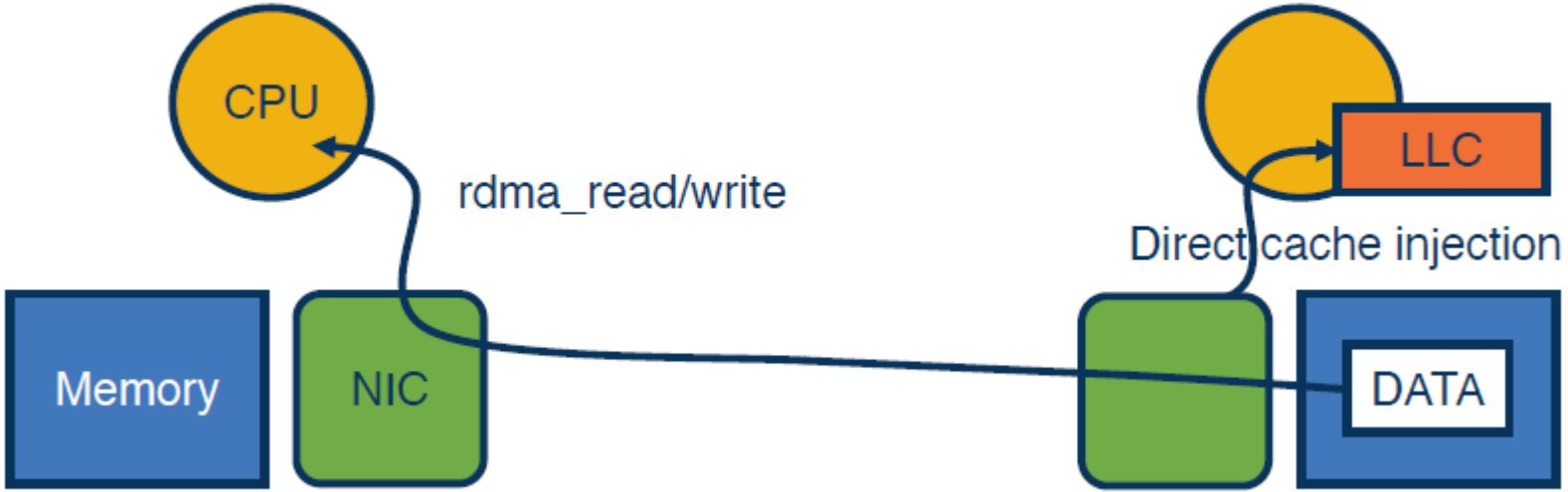
Remote DMA over Converged Ethernet (RoCE)

Internet Wide Area RDMA Protocol (iWARP)

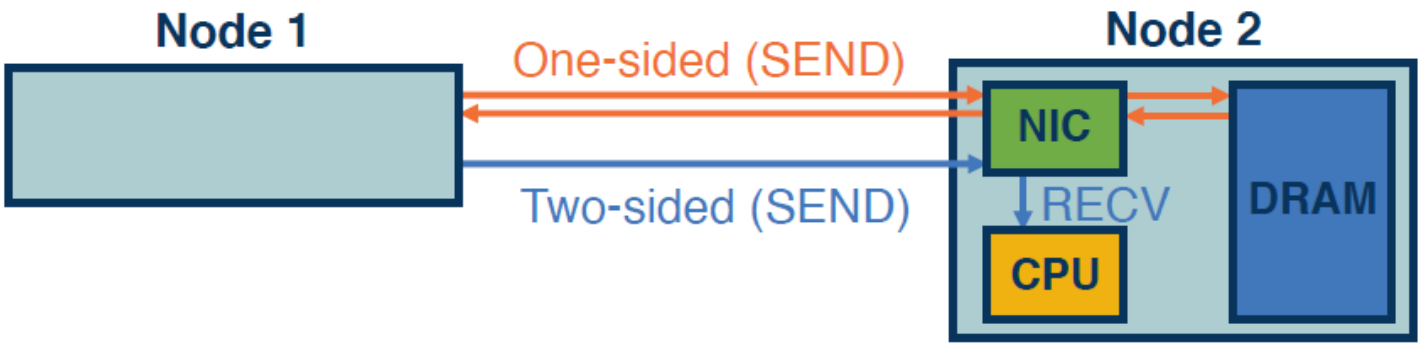
Two-sided RDMA



One-sided RDMA



RDMA Specialized Remote Procedure Call (RPC)



One-sided:

- Low CPU utilization, multiple RTTs
- Redesign to match new API

Two-sided:

- Single RTT, simple integration in existing stack

Figure adapted from: https://www.usenix.org/sites/default/files/conference/protected-files/osdi16_slides_kalia.pdf

RPC with RDMA Options

Leverage RDMA features:

- Connection vs. connection-less protocol
- Shared Receive Queues
- Datagram communication for small RPCs
- …

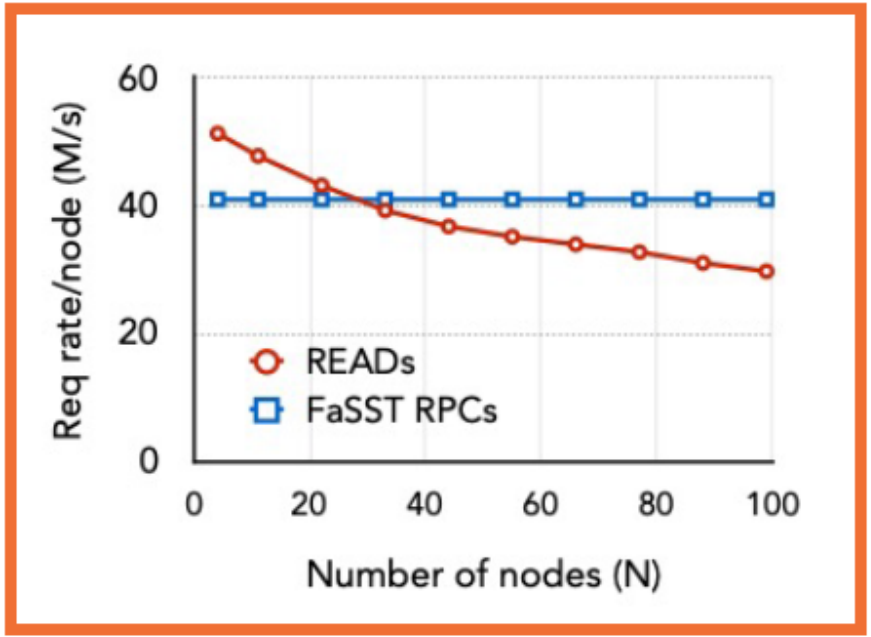
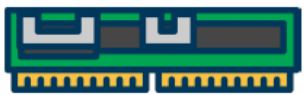
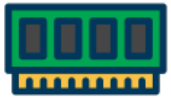


Figure from: FaSST Paper, OSDI'16

Persistent Memory

	PCIe SSD	Byte-addressable Persistent Memory (PMEM)	Main Memory (DRAM)
		PCM FeRAM MRAM 	
Interface	block I/O	**byte**	byte
Durability	yes	**yes**	no
Latency	30us	~100ns -- 300ns	60ns
Bandwidth	-	$\frac{1}{8}x\ to\ \frac{1}{4}x$	$1x$
Capacity	Terabytes	Terabytes	Gigabytes

RPC + Persistent Memory

Persistent data operations require **flush** to persistent memory

Must complete **before client is Acknowledged**

Removes advantage of RDMA over send/receive RPC

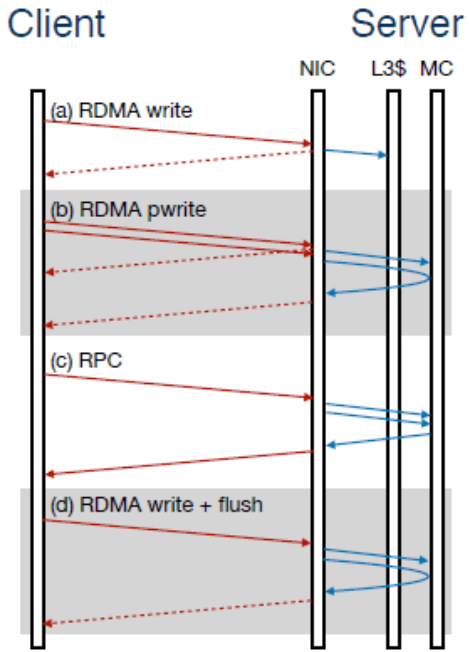


Figure 2 from SOCC'20 paper: *http://anujkalia.com/doc/socc20/kalia.pdf*

Network and PCIe operations involved in writing to remote NVMM with different methods. Red arrows from the client to the server's NIC are network packets. The dotted arrows are NIC-generated RDMA acknowledgments. Straight blue arrows between the server's NIC and its cache (L3) or memory controller (MC) are PCIe DMA or MMIO writes; the curved ones are DMA reads. The server's CPU (not shown) is in-volved in persisting RPC requests.

Disaggregation

Server configurations

- Different memory components
- Different compute components
- Storage
- Etc.

Choose specific amounts based on workload

- Workloads evolve

Monolithic server configurations

- Inflexible, cannot elastically scale
- Imbalances lead to inefficiency

Monolithic servers with fixed configuration of CPU, memory, storage, devices, PCIe slots, …

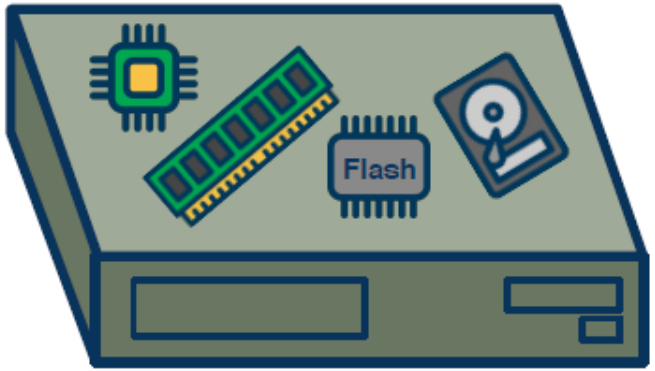


Figure adapted from the LegoOS presentation: https://www.usenix.org/sites/default/files/conference/protected-files/osdi18_slides_shan.pdf

Disaggregation

Resource Disaggregation

- Pools of different resource types
- Network attached
- Independently scaled

Motivation

- Fast Networks
- Integrate compute with devices
  - Smart NICs
  - In-memory compute

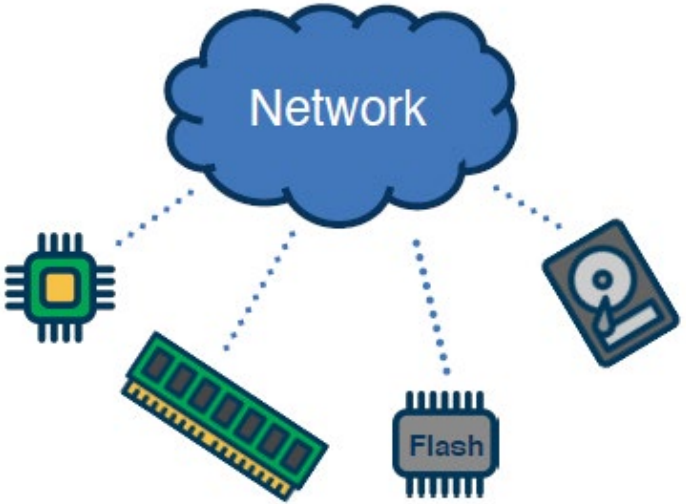
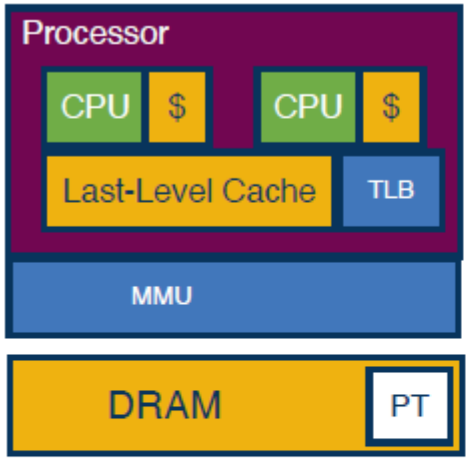


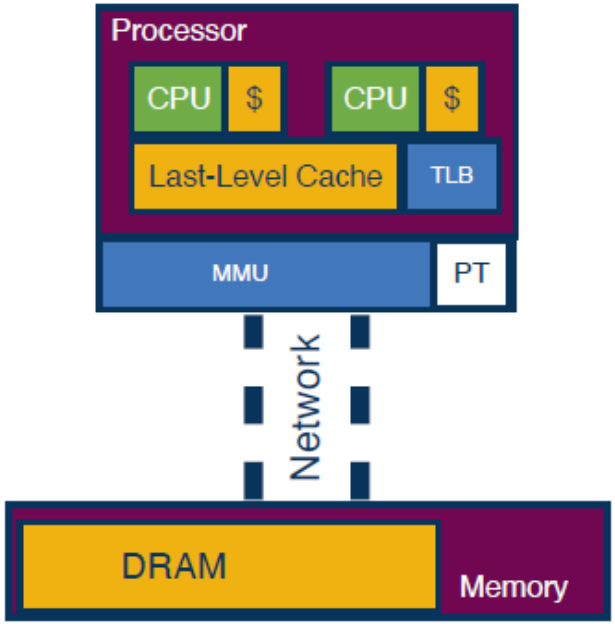
Figure adapted from the LegoOS presentation: https://www.usenix.org/sites/default/files/conference/protected-files/osdi18_slides_shan.pdf

Separating Processor and Memory



Figures adapted from the LegoOS Presentation at OSDI'20: https://github.com/WukLab/LegoOS

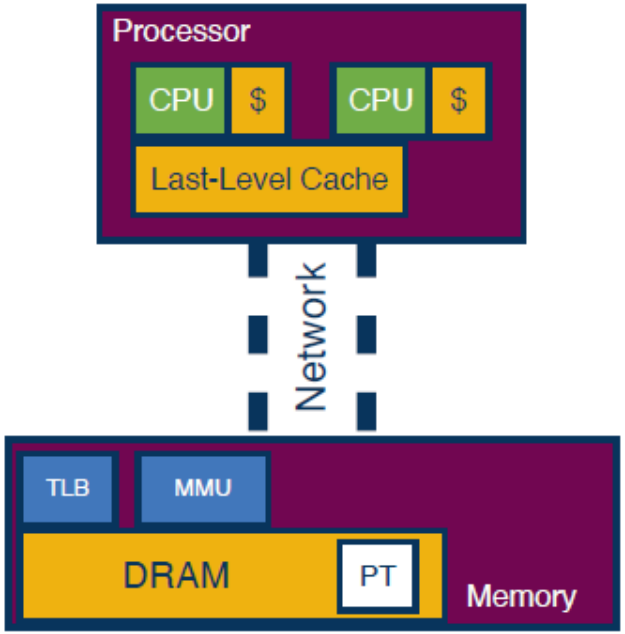
Separate Processor and Memory



Disaggregating DRAM

Figures adapted from the LegoOS Presentation at OSDI'20: https://github.com/WukLab/LegoOS

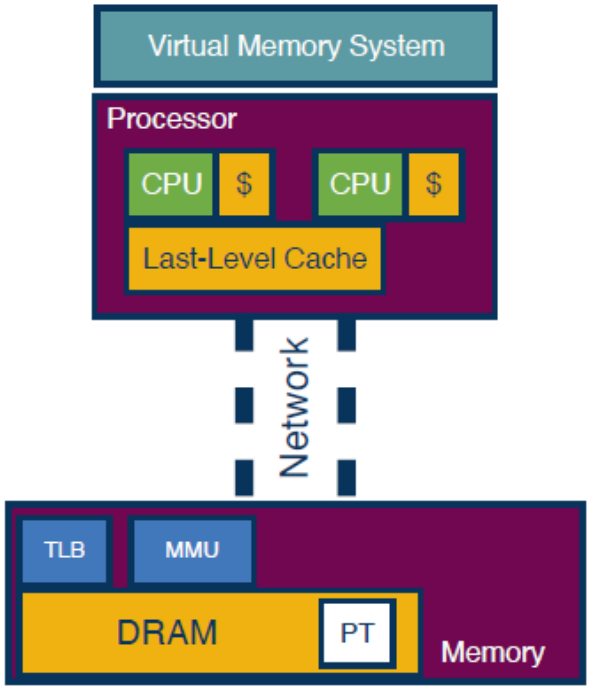
Separate Processor and Memory



Separate and move
hardware units
to memory component

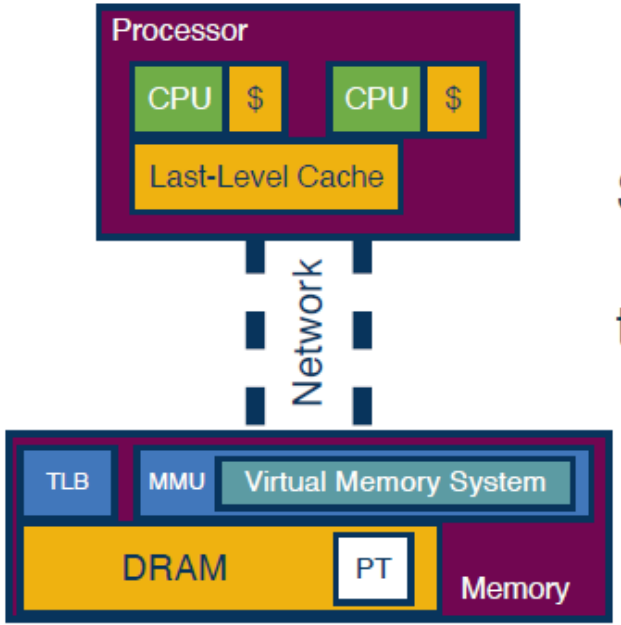
Figures adapted from the LegoOS Presentation at OSDI'20: https://github.com/WukLab/LegoOS

Separate Processor and Memory



Figures adapted from the LegoOS Presentation at OSDI'20: https://github.com/WukLab/LegoOS

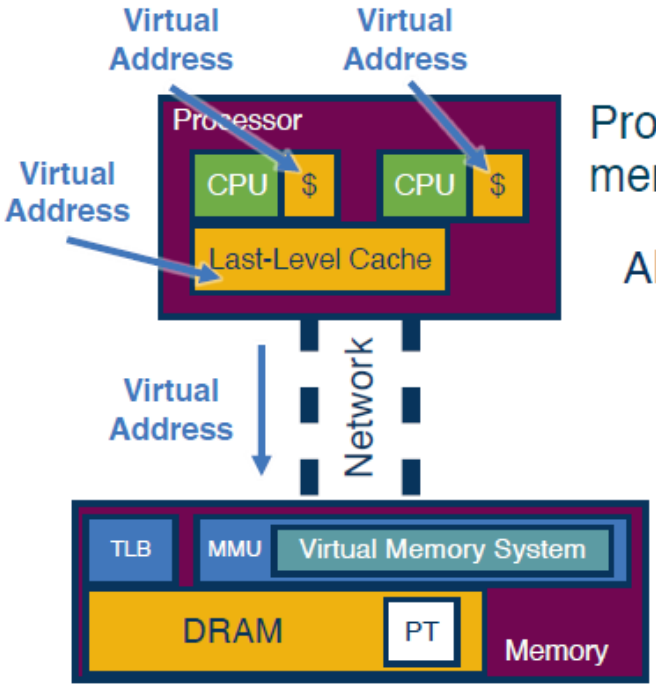
Separate Processor and Memory



Separate and move
virtual memory system
to memory component

Figures adapted from the LegoOS Presentation at OSDI'20: https://github.com/WukLab/LegoOS

Separate Processor and Memory



Processor components only see virtual memory addresses

All levels of cache are ***virtual cache***

Memory components manage virtual and physical memory

Figures adapted from the LegoOS Presentation at OSDI'20: https://github.com/WukLab/LegoOS

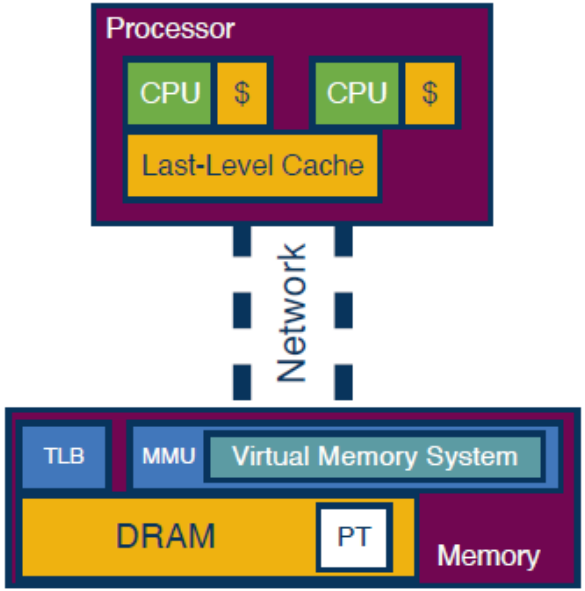
Disaggregation Challenges

Network is slower than local memory bus

- Bandwidth: 25-50% capacity
- Improving quickly (800Gb/s Ethernet now exists)

- Latency: 12x longer
- Improving slowly (“speed of light”)

Use Extended Cache in CPU

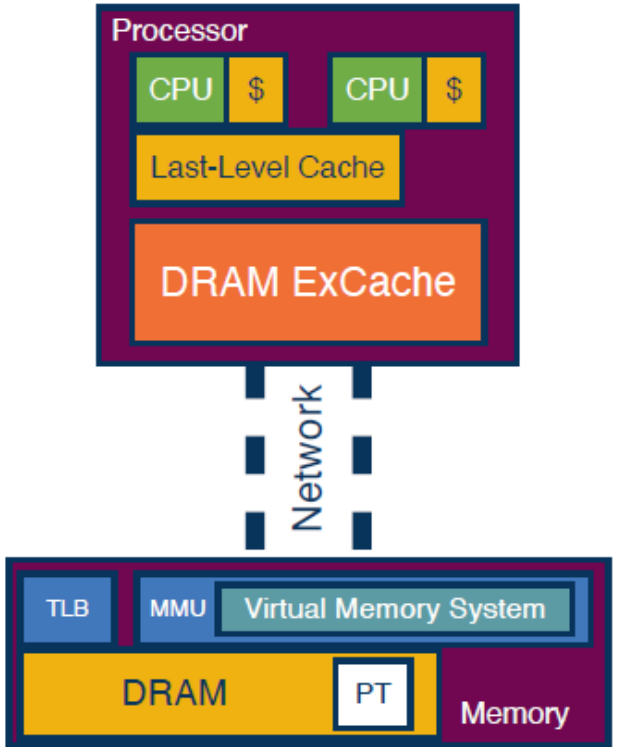


Figures adapted from the LegoOS Presentation at OSDI'20: https://github.com/WukLab/LegoOS

Add small DRAM/HBM with CPU

Extend Cache

- Software/Hardware co-managed
- Inclusive
- Virtual memory cache



Figures adapted from the LegoOS Presentation at OSDI'20: https://github.com/WukLab/LegoOS

Datacenter Scale

Thousands of components

General purpose server components

Specialized configurations for specific workloads

Hyperscale sizes (order of magnitude larger!)

https://blog.google/around-the-globe/google-asia/growing-our-data-center-in-singapore

Management Challenges

Application

Long running services
Batch jobs
Production vs. non-production

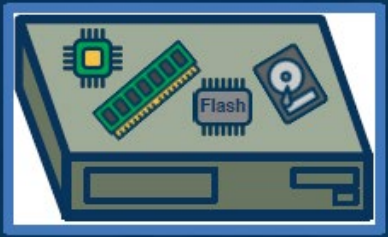
Multi-tenancy

Processes/tasks

Differ in end-to-end metrics:
- Latency-sensitive
- Throughput-intensive

Differ in resource requirements:
- Compute-intensive
- Demand accelerators
- Data-intensive
- Data access speed vs. data capacity

Orchestrate resource allocation and deployment



Service-level Agreement (SLA)
Service-level Objective (SLO)

Managing Resources

Omega (Eurosys 2013)

Borg (Eurosys 2015)

Kubernetes

Borg Terminology

Cell: a collection of machines

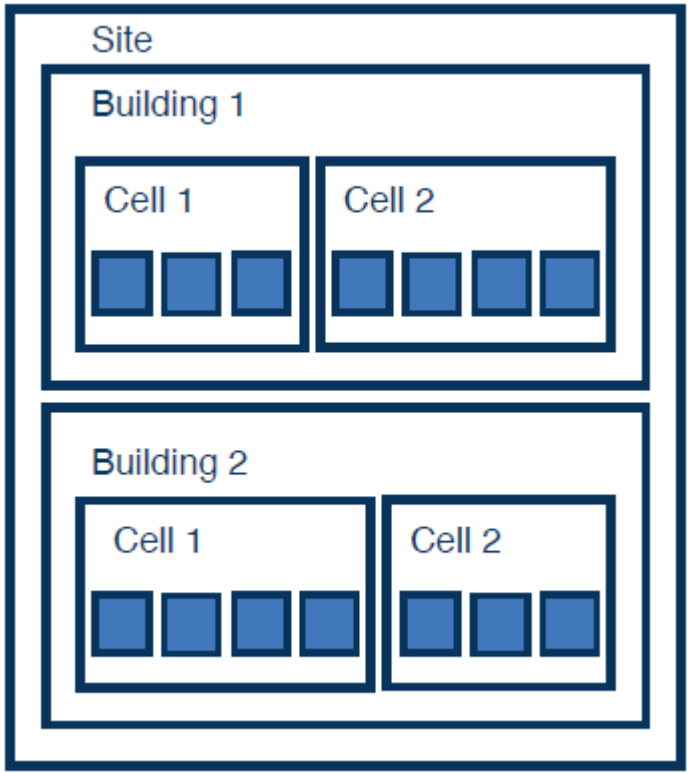
- Unit of management

Machines in a cell are part of one cluster

- High-performance “network fabric”

Cluster: set of machines in a single datacenter building

Site: a set of datacenter buildings



Application Job & Task States

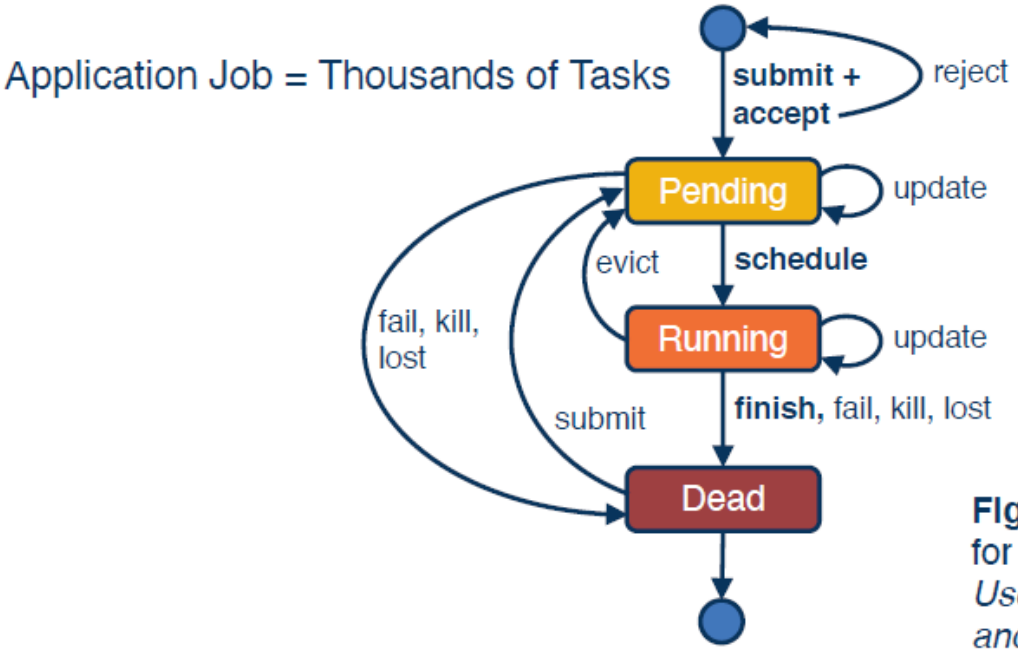


Figure 2: The state diagram for both jobs and tasks. *Users can trigger submit, kill, and update transitions.*

Adapted from: https://storage.googleapis.com/pub-tools-public-publication-data/pdf/43438.pdf

Borg Architecture

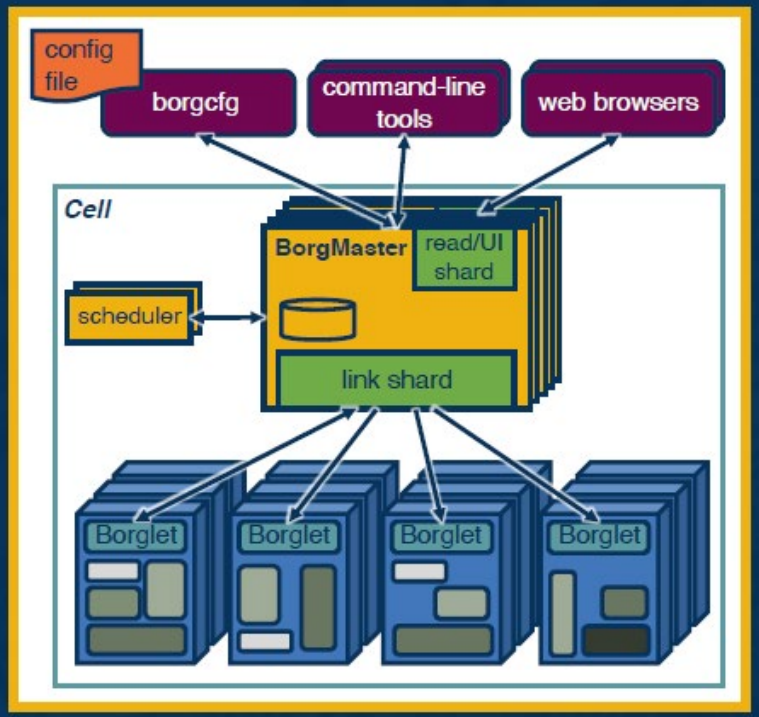


Figure 1: The high-level architecture of Borg. *Only a tiny fraction of the thousands of worker nodes are shown.*

Figure adapted from: https://res.infoq.com/news/2015/04/google-borg/en/resources/borg.png

Borg Architecture

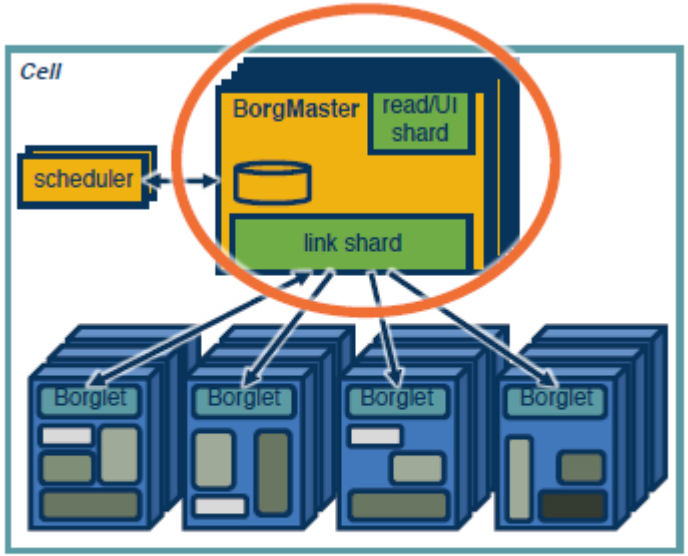


Figure adapted from: https://res.infoq.com/news/2015/04/google-borg/en/resources/borg.png

- Borgmaster is the brain of the Borg system
- One Borgmaster per cell
- Handles requests to execute/check job status
- Cell state maintained in memory
- Maintains **pending queue**
  - If job exceeds quota, not admitted
  - Quota assignment is *policy,* not Borg
- Assigns **tasks** to machines
- Monitors all machine state within the cell

Borg Architecture

Task Scheduler:

- Scans pending queue in priority order
- Checks feasibility, finds set of machines where tasks can run
- Finds best fit
- Forwards assignment to Borgmaster

Borgmaster:

- May pre-empt lower priority tasks
- Pre-empted tasks are returned to pending queue
- **Production** priority tasks not pre-empted.

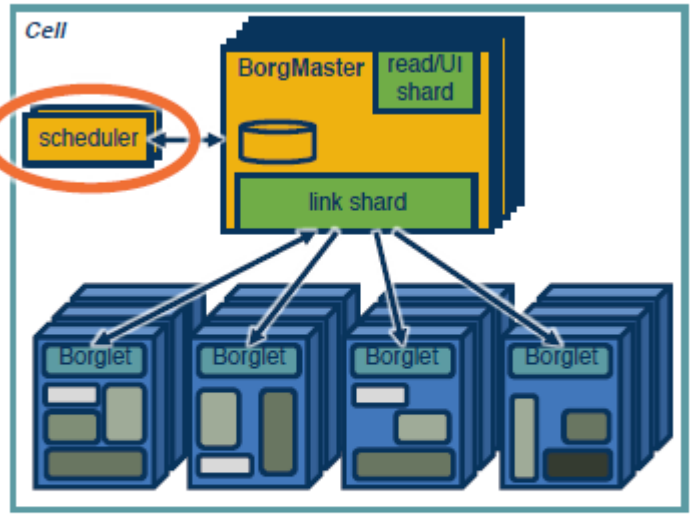


Figure adapted from: https://res.infoq.com/news/2015/04/google-borg/en/resources/borg.png

Borg Architecture

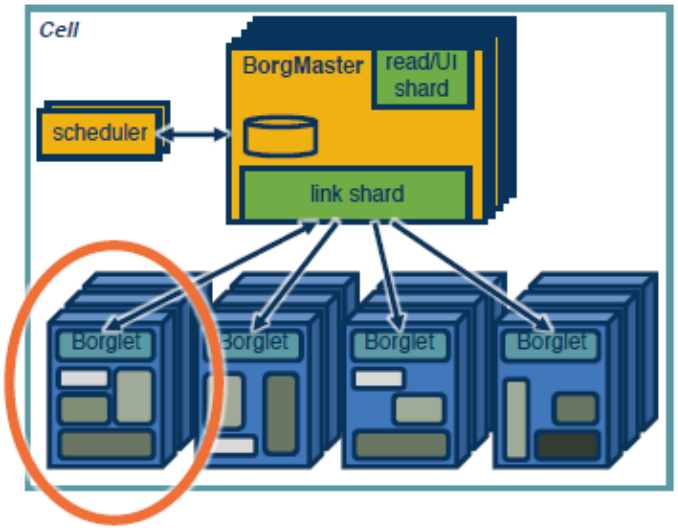


Figure adapted from: https://res.infoq.com/news/2015/04/google-borg/en/resources/borg.png

Borglet: local Borg agent on machine

- Start/stop tasks
- Restart tasks after failure
- Manages local resources
- Reports machine state to Borgmaster

Borgmaster polls Borglet

- Machine info
- Updated cell state
- No response = machine marked down

Borgmaster Reliability

Borgmaster has 5 replicas

Chubby lock acquire by leader; other replicas use lock collision info to find leader.

Only leader changes cell state.

Cell leader is *also* Paxos leader for replicated data store

Each replica contains cell state (Paxos-based key-value store)

Failover time is approximately **10 seconds**.

Borgmaster Availability

Reschedule pre-empted tasks

Reduces correlated failures

- Spreads job tasks across failure domains
  - Separate machines
  - Separate racks
  - Separate power domains

Limits task disruption rates

Avoids repating task/machine pairings that cause task/machine crashes

- “Learning”

Borg Scalability

Decouples task assignment from scheduling

- Asynchronous update/read from pending queue
- Permits different schedulers

Efficient communications

- Separate threads for RPCs and Borglet communications
- Use link shards to summarize information from Borglets

Optimize scoring of machine/task pairs

Efficient resource utilization

- Spread job tasks across machines
- Allow mixing production/non-production workloads

Scheduler Optimizations

Score Caching:

- Scores for task assignment cached until machine/task priorities change
- Small changes in resource quantities on machines are ignored

Equivalence classes:

- Group tasks with identical requirements
- Score computed once **per equivalence class**

Relaxed randomization:

- Scheduler examines machines in random order to finds enough to score
- Reduces amount of scoring/cache-invalidations when tasks enter/leave
- Speeds up assignment of tasks to machines

Performance Isolation

Tasks run in containers

Borglets manipulate container properties

Borg tasks have an application class

- Latency sensitive application classes: high priority
- Batch application classes: low priority

Compressible resources

- CPU cycles, rate based disk I/O bandwidth
- May be reclaimed with lower QoS, but continues running

Non-compressible resources

- Memory, disk space
- Reclaim requires halting task

Lesson Review

Data Center Services

Trends and Services

High-speed RDMA and programmable networks

Resource heterogeneity

Resource disaggregation

Resource management and orchestration

Questions?


THE UNIVERSITY OF BRITISH COLUMBIA
THE UNIVERSITY OF BRITISH