Disaggregating Memory with Software-Managed Virtual Cache

Yizhou Shan, Yiying Zhang

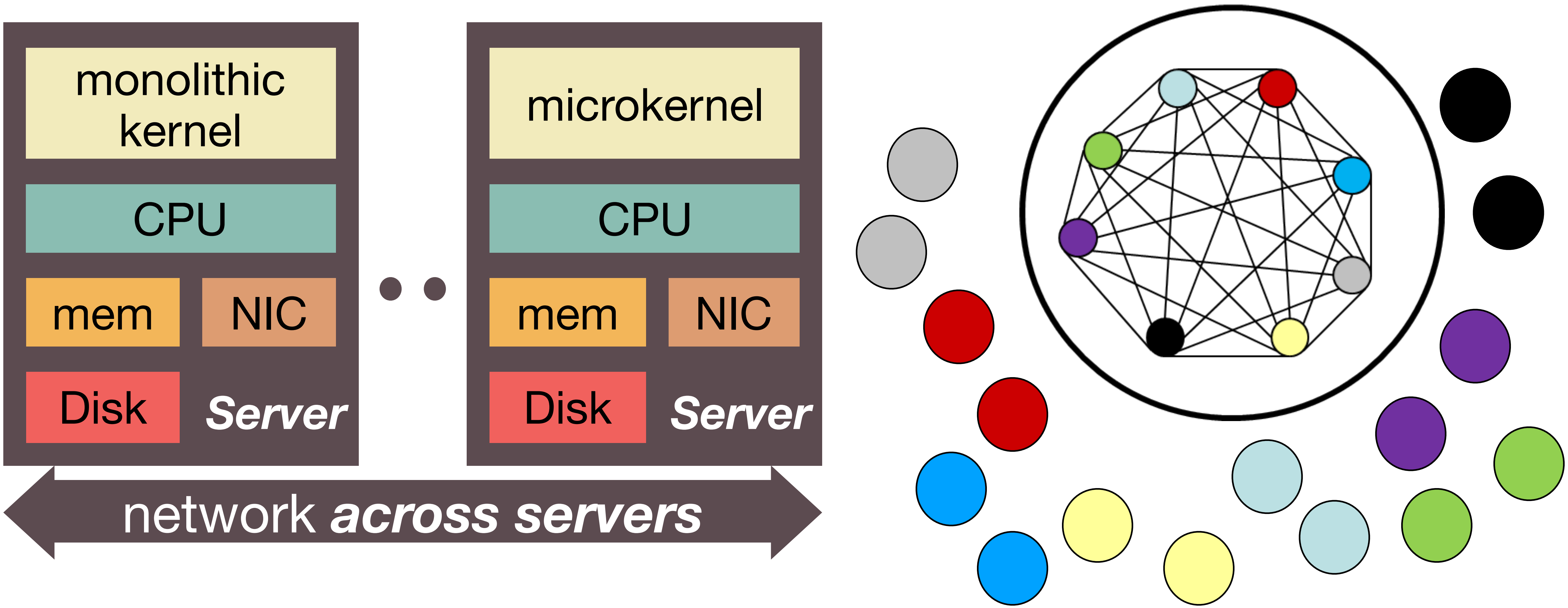
WukLab

Purdue University

Resource Disaggregation:

Breaking monolithic servers into network-attached, independent hardware components

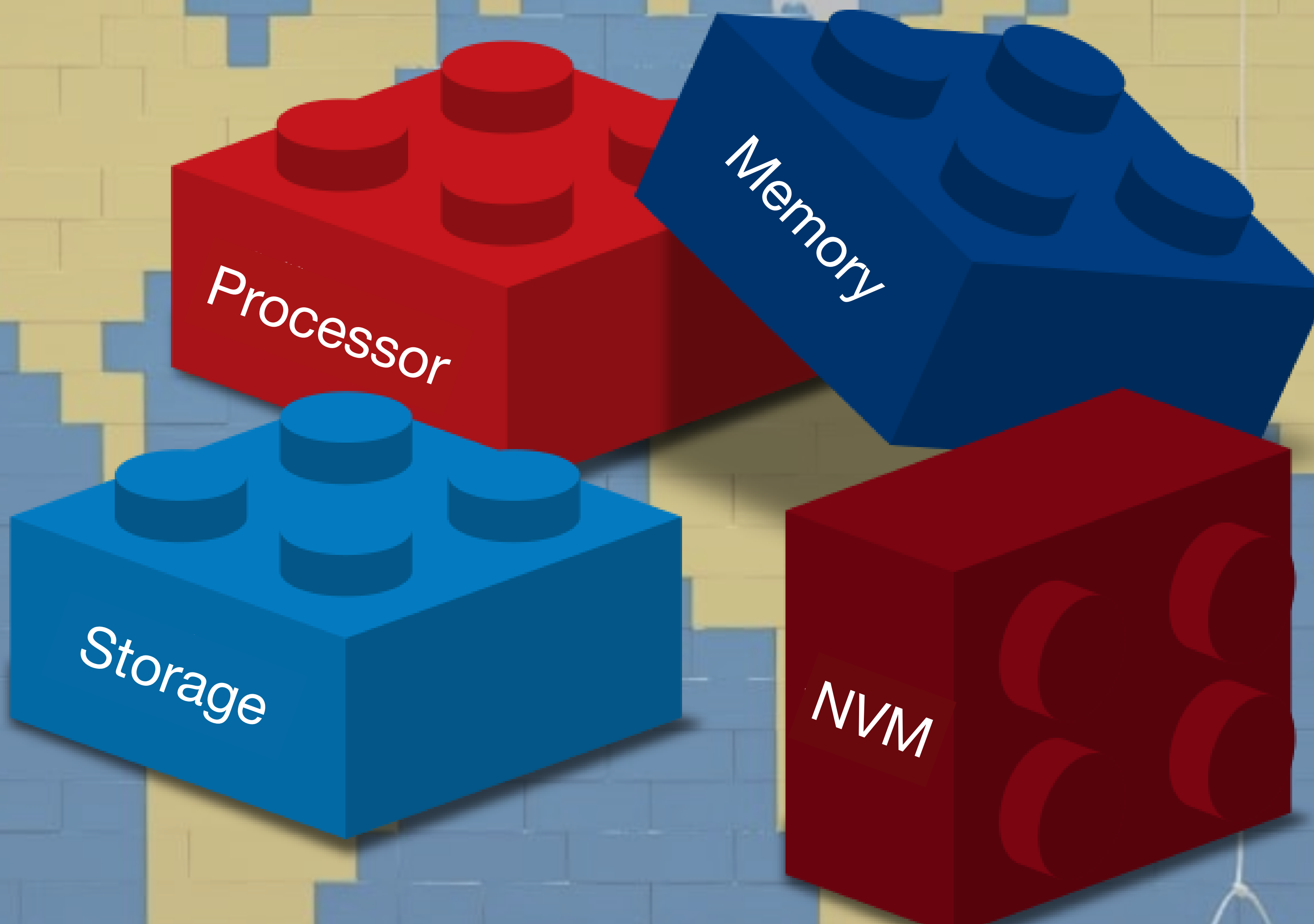
Traditional OSes

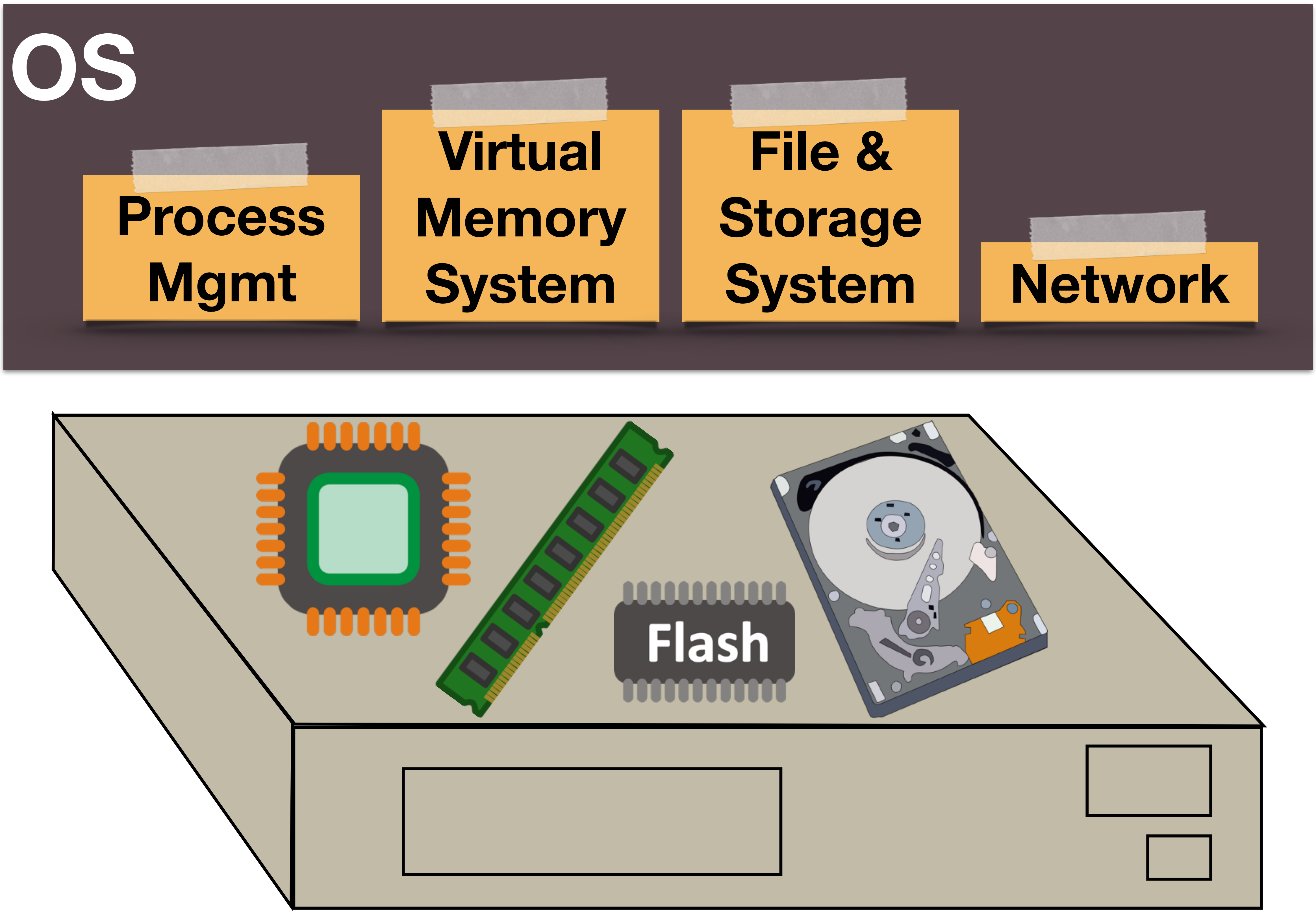


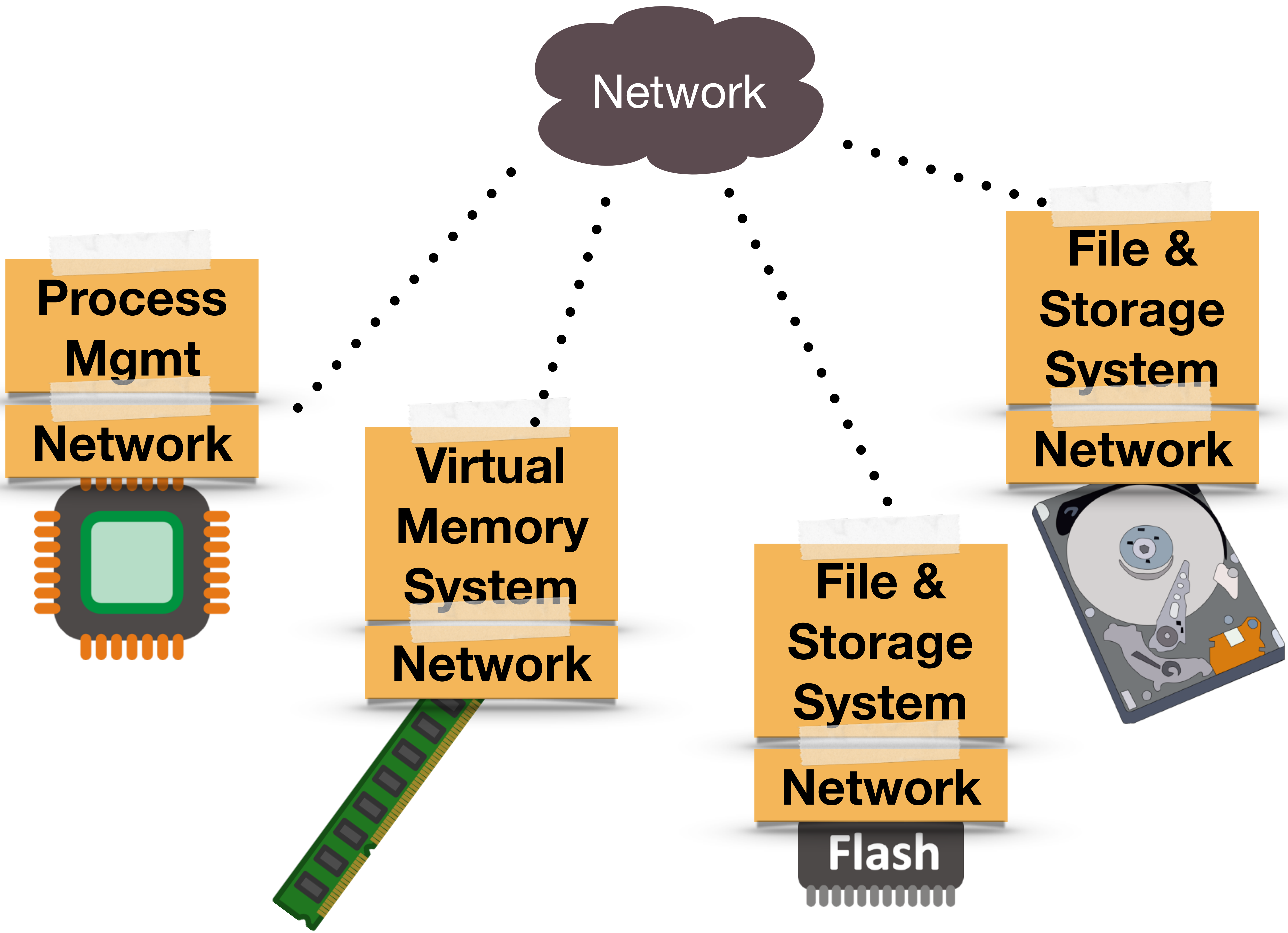
- Manages single node and all hardware resources in it
- Bad for hardware heterogeneity and hotplug
- Does not handle component failure

When hardware is disaggregated, the OS should be also!

Lego: the *First* Disaggregated OS




OS
Process Mgmt
Virtual Memory System
File & Storage System
Network
Flash


Network
Process Mgmt
Network
Virtual Memory System
Network
File & Storage System
Network
Flash
File & Storage System
Network

Key Challenge: Cost of Crossing Network

	Bandwidth	Latency
Mem Bus	50-100 GB/s	~50ns
PCIe 3.0 (x16)	16 GB/s	~700ns
InfiniBand (EDR)	12.5 GB/s	500ns
InfiniBand (HDR)	25 GB/s	<500ns
GenZ	32-400 GB/s	<100ns

- Network **hardware** is much faster than before
- Current network still slower than local memory bus

Observations in Memory Access and Hardware Trends

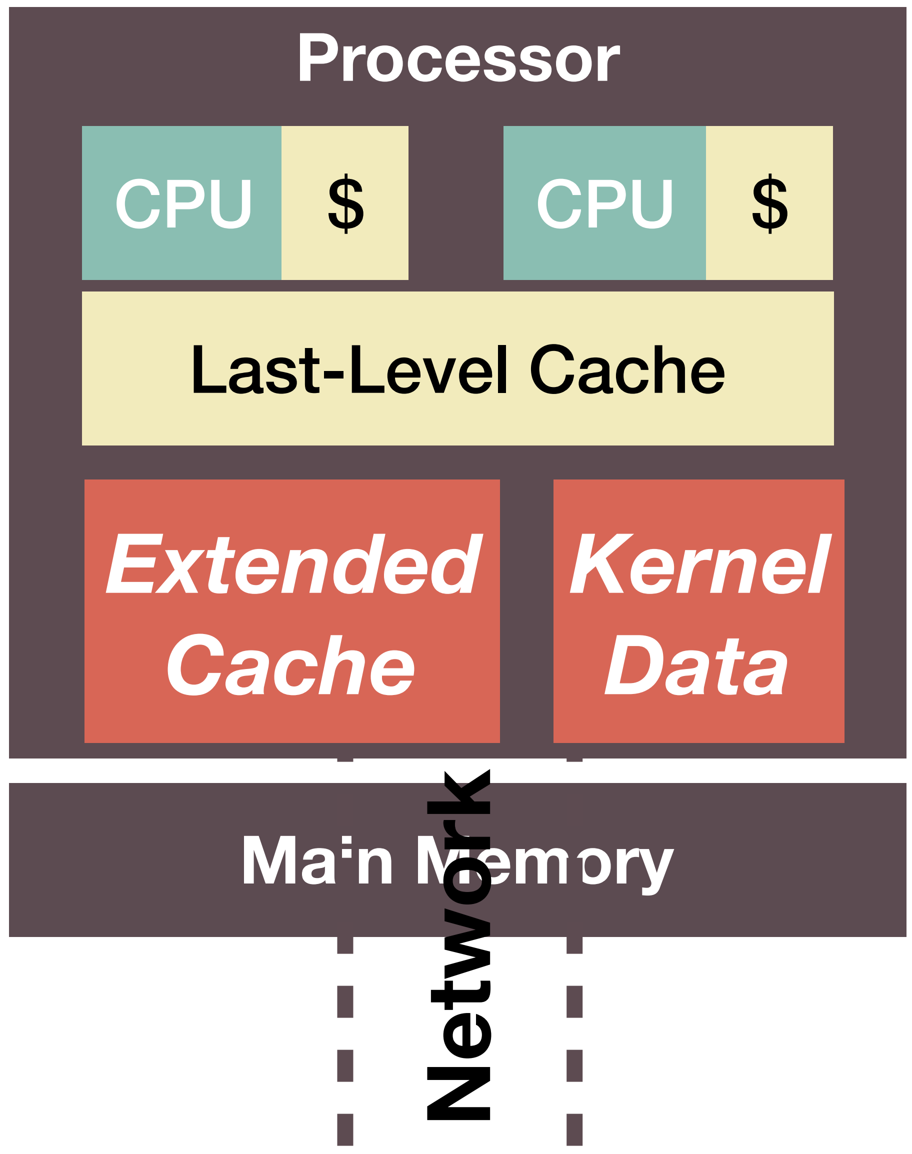
- Total memory footprint can be large,
- but most accesses go to a small portion
  - 90% => 7MB / 9.6GB (pagerank)
  - 95% => 300MB / 9.6GB

- Faster, smaller memory close to CPU
  - HBM, 3D-stacked

- More computation power at memory
  - PIM/PNM

Our Solution: Separate Memory Perf and Capacity

- **Bigger memory behind network**
- **Extended cache at processor**
  - HBM or regular DRAM
  - Can be software-managed
- Separate physical mem for kernel

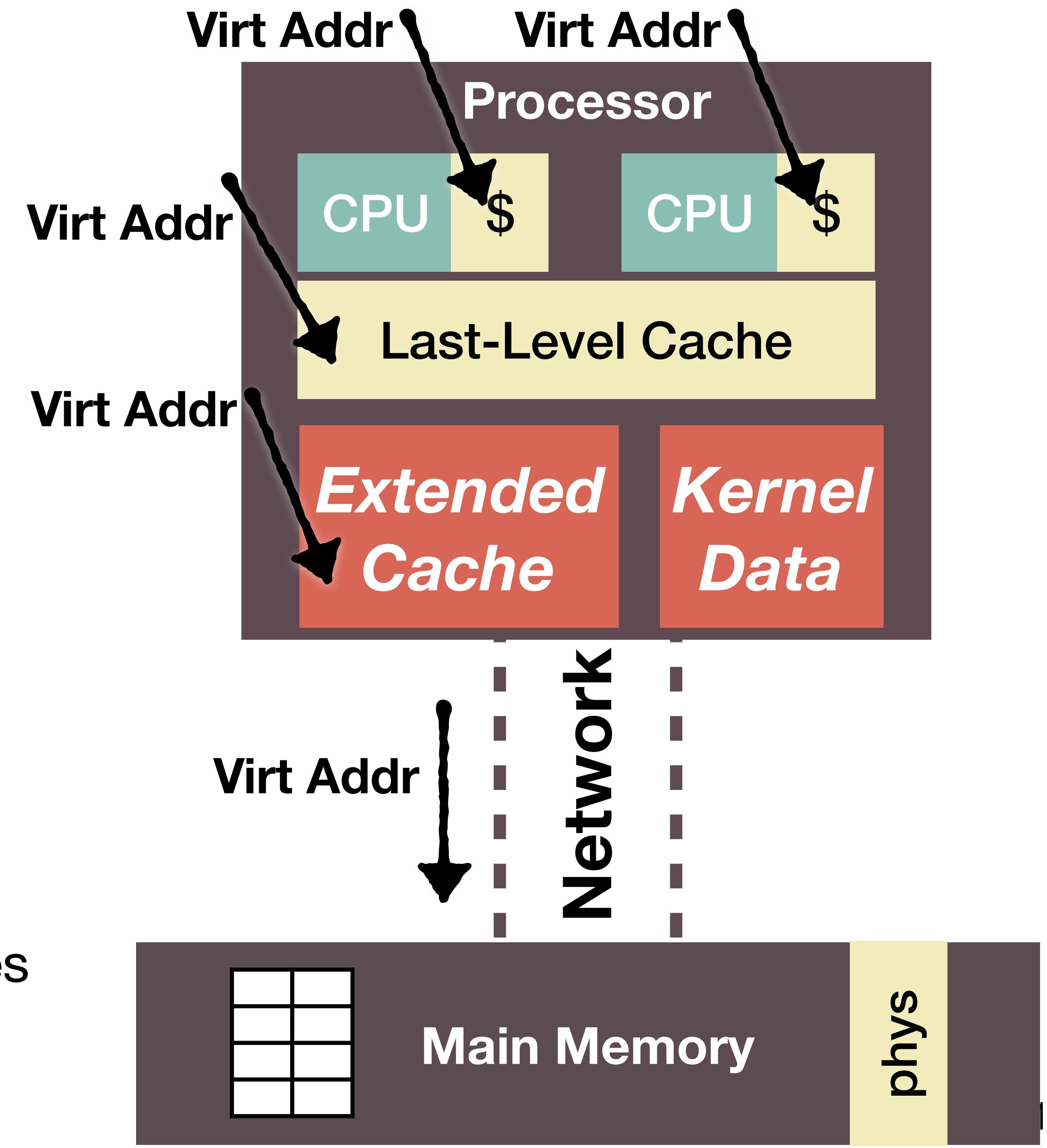


Clean Separation of Processor and Memory Functionalities

- Important for heterogeneity, flexibility, and failure independence

- Memory components manage
  - Virtual and physical memory spaces
  - Virtual to physical memory mapping

- Processors
  - Only see virtual memory addresses
  - Software-managed virtual cache



Other Challenges

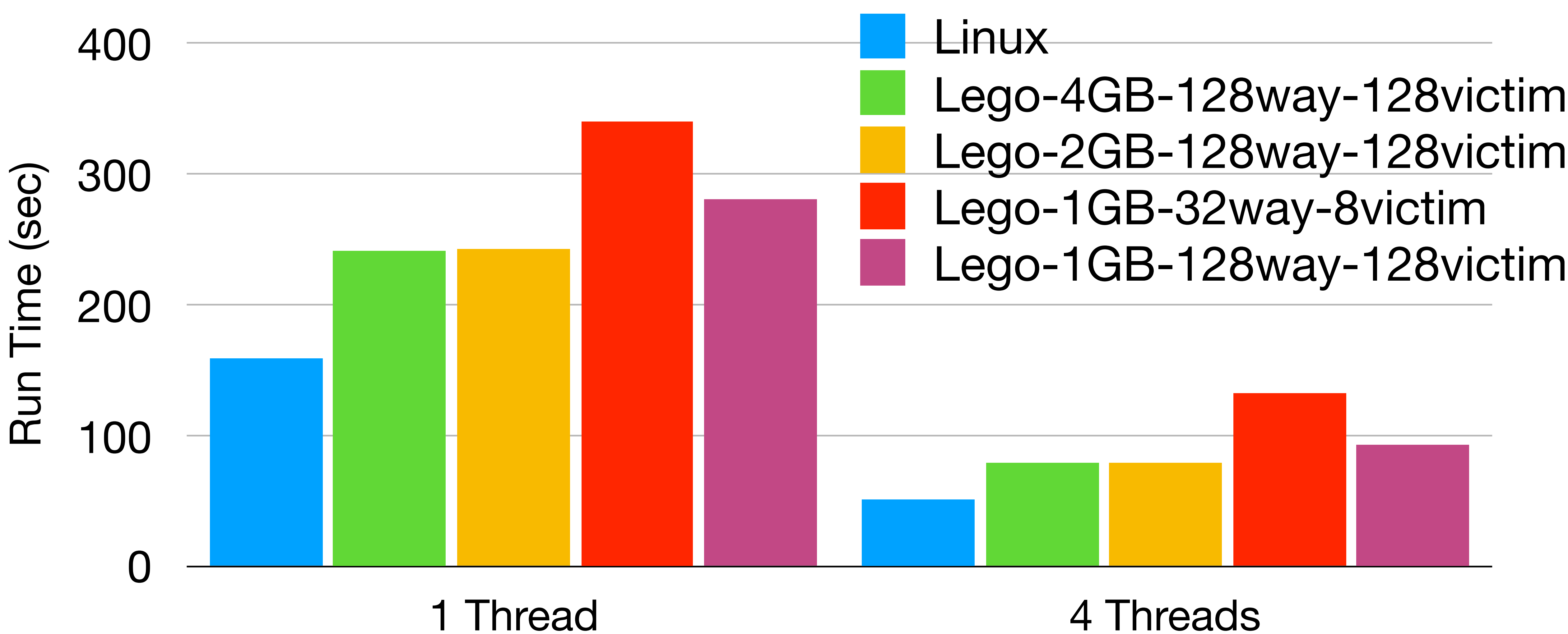
- Handling component failure
- Manage distributed, heterogeneous resources
- Fitting micro-OS services in hardware controller
- Implementing Lego on current servers

Implementation and Emulation

- Lego built from scratch, >200K LOC and growing
  - Runs all Linux ABIs and unmodified binaries
  - Manages disaggregated processor, memory, storage
  - Global resource manager
- Hardware emulation
  - Use regular servers to emulate hardware devices
  - DRAM organized as extended cache, managed by page fault handler

Preliminary Results

- One processor, one memory, one storage, connected with 40Gbps InfiniBand
- ***Phoenix*** [1]: single-node MapReduce implementation, running word count of 2GB file



[1]: Ranger etal., “Evaluating MapReduce for Multi-core and Multiprocessor Systems”. (HPCA 07)”

Conclusion

- Resource disaggregation calls for new system
- ***Lego***: new OS designed and built for datacenter resource disaggregation
- Separating memory performance and capacity, and processor and memory functionalities
- Many challenges and many potentials

Thank you
Questions?

WukLab

wuklab.io

PURDUE UNIVERSITY

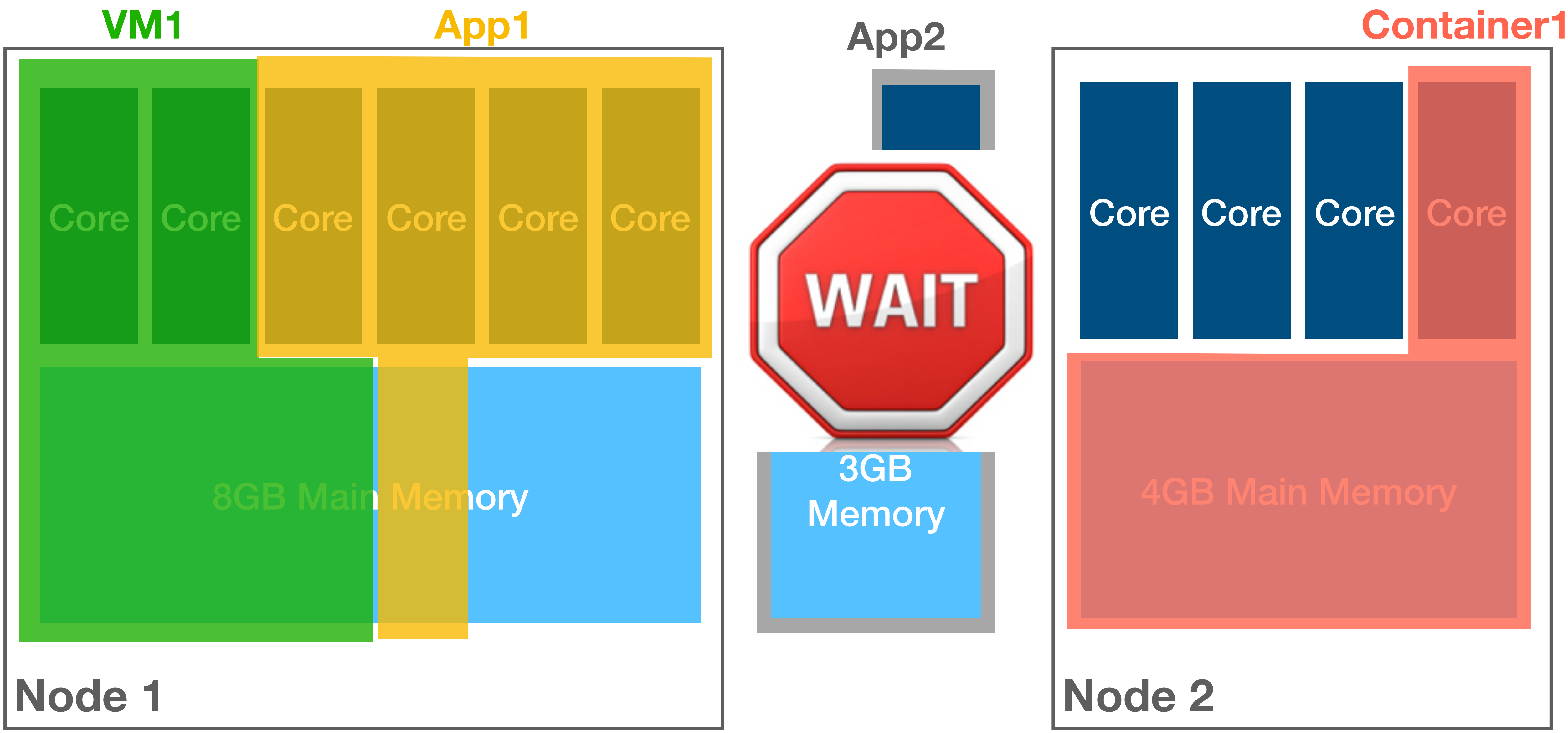
Split Container: Running Containers beyond Physical Machine Boundaries

Yilun Chen, Yiying Zhang

WukLab

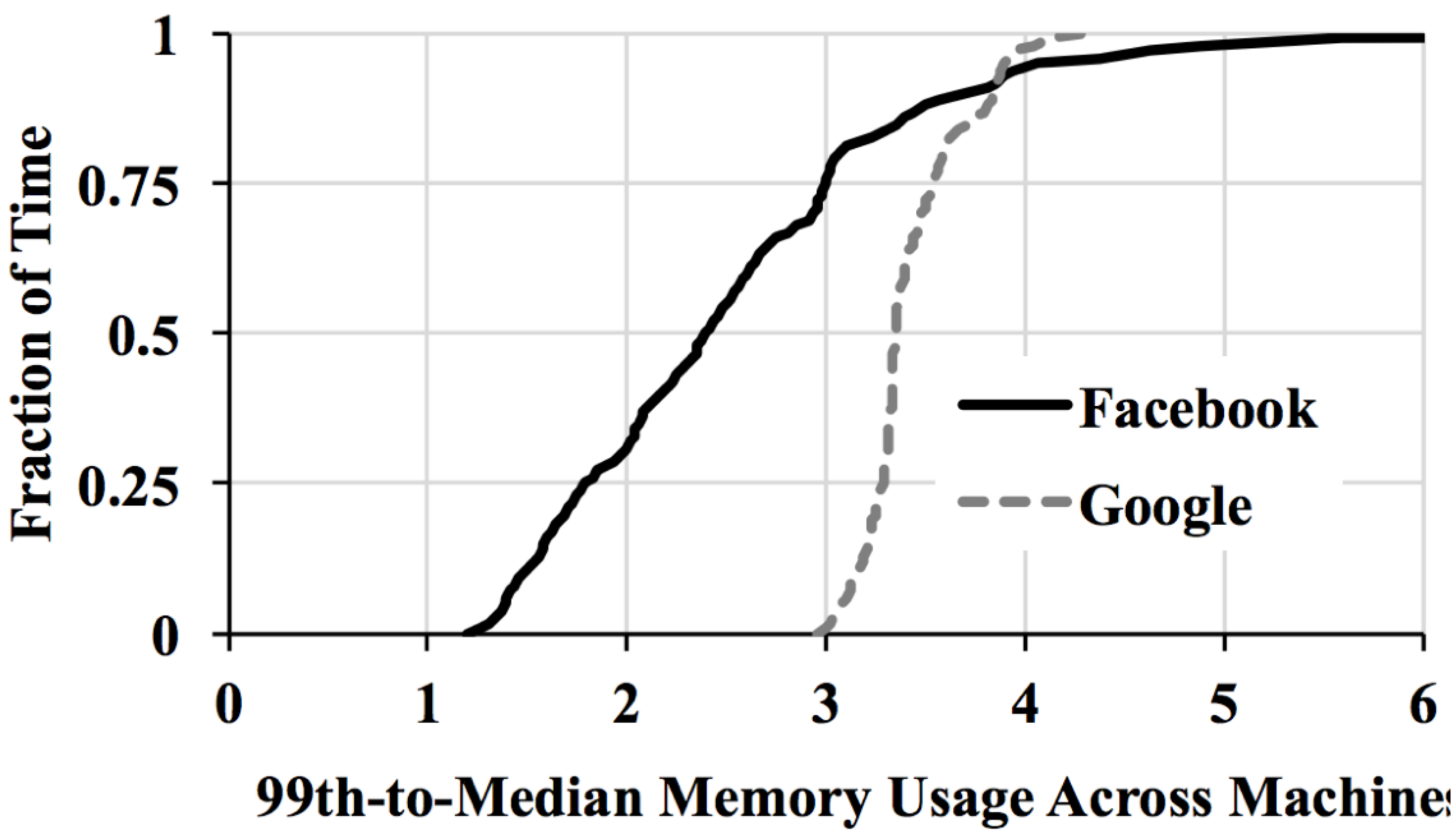
PURDUE UNIVERSITY

Resource Allocation in Datacenters

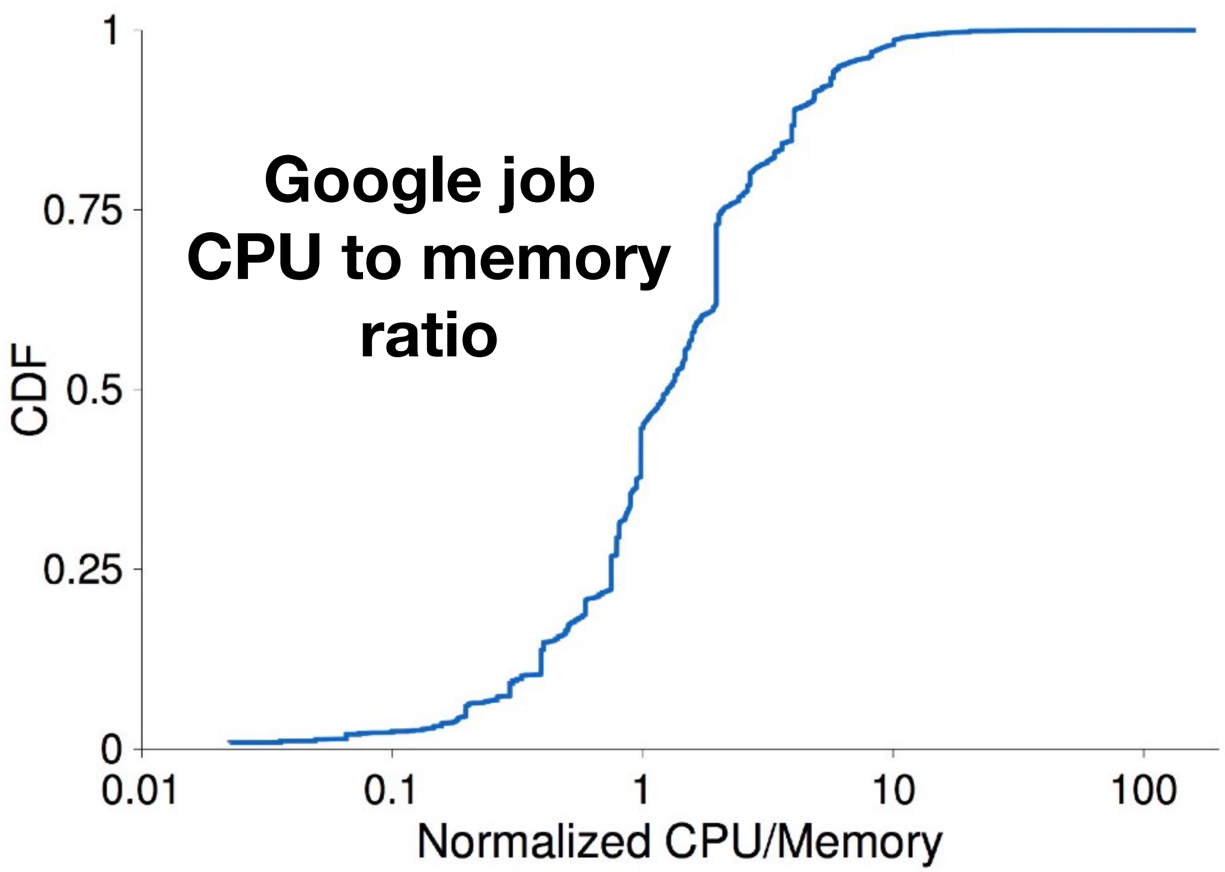


Physical Machine Boundary!

CPU/Memory Usages across Machines and across Jobs



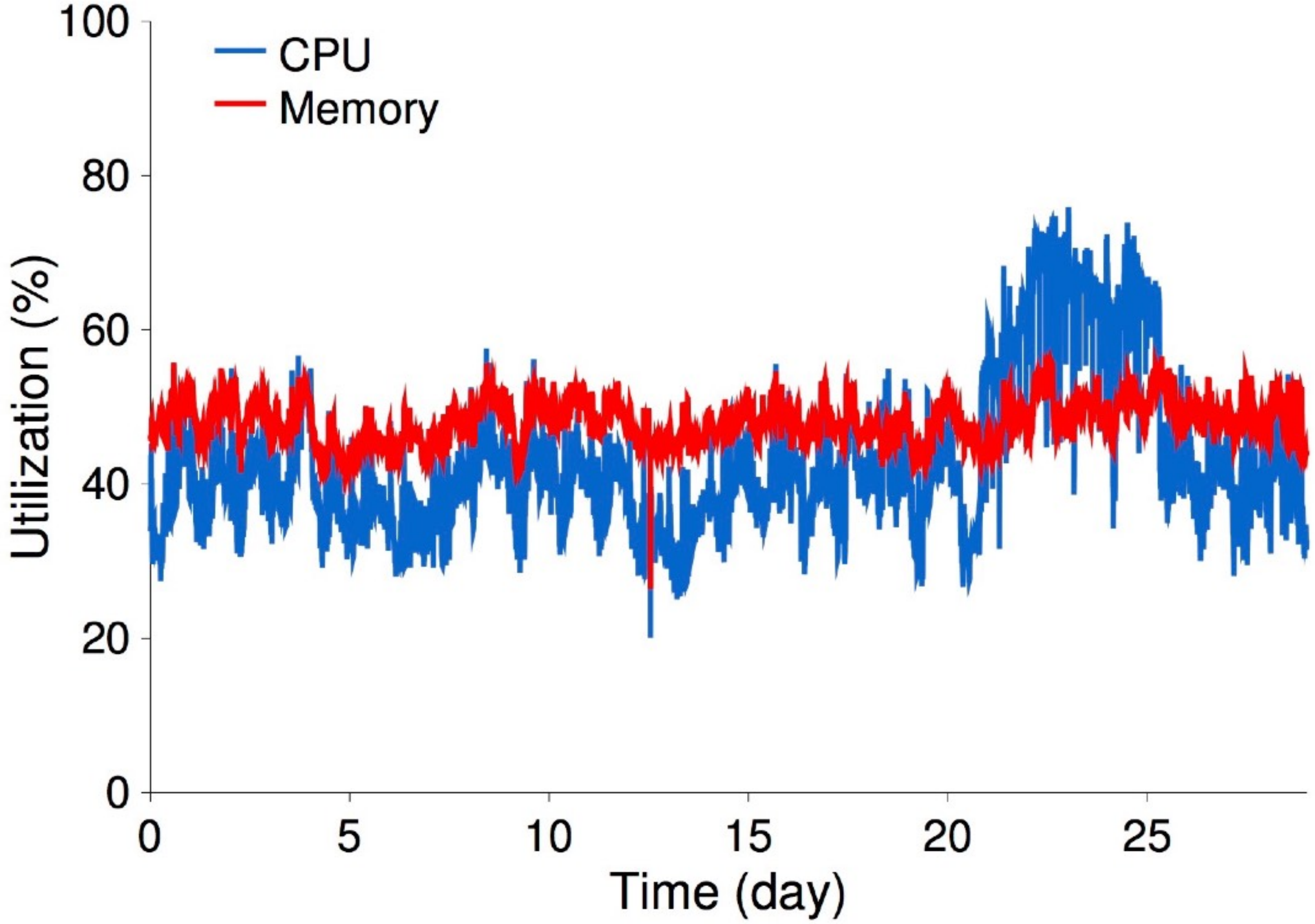
Source: Gu et al. “Efficient Memory Disaggregation with Infiniswap” NSDI’17

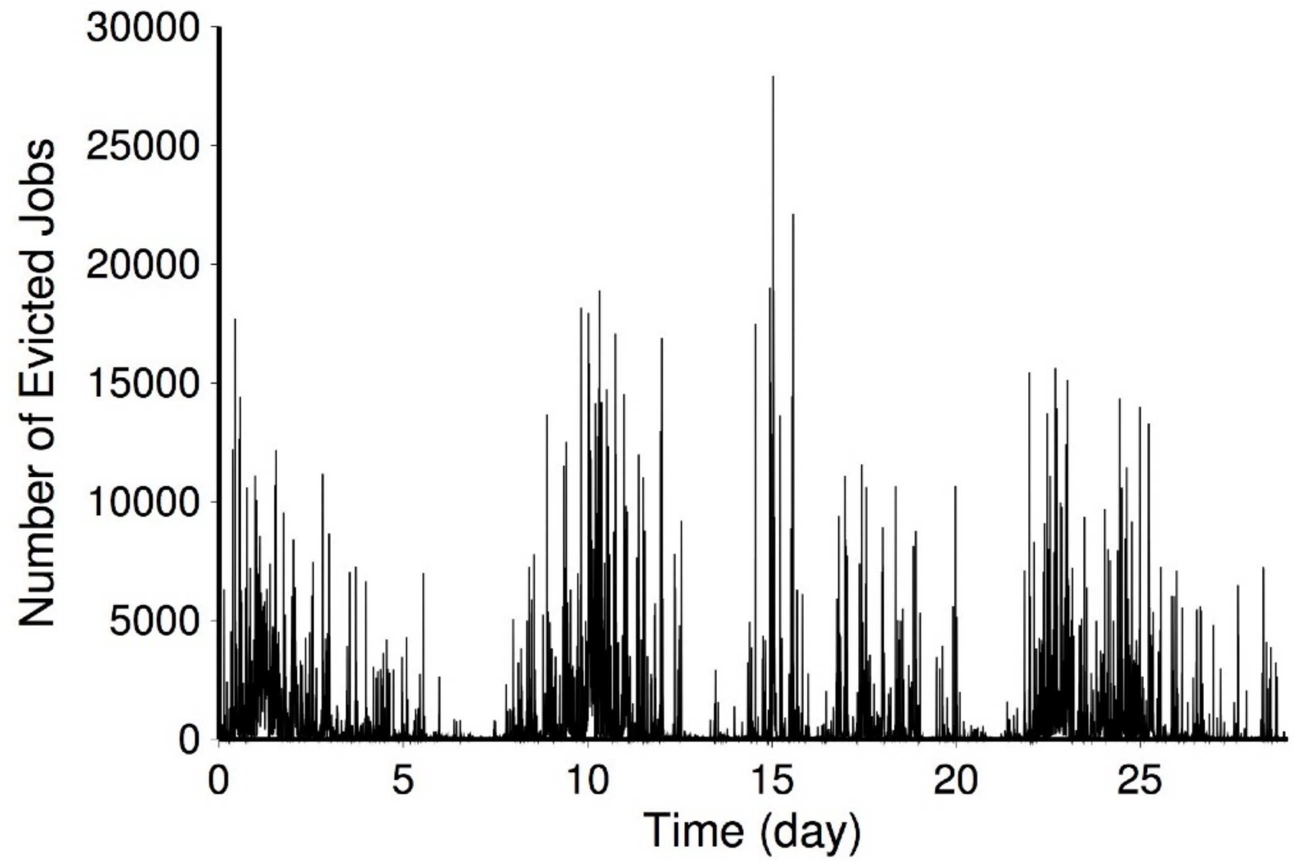


Google Cluster Trace “https://github.com/google/cluster-data”

Modern Datacenter Applications Have Heterogeneous CPU/Memory Requirements

Resource Utilization in Production Clusters





* Google Production Cluster Trace Data. "https://github.com/google/cluster-data"

Unused Resource + Waiting/Killed Jobs Because of Physical-Node Constraints

Physical Resource Disaggregation

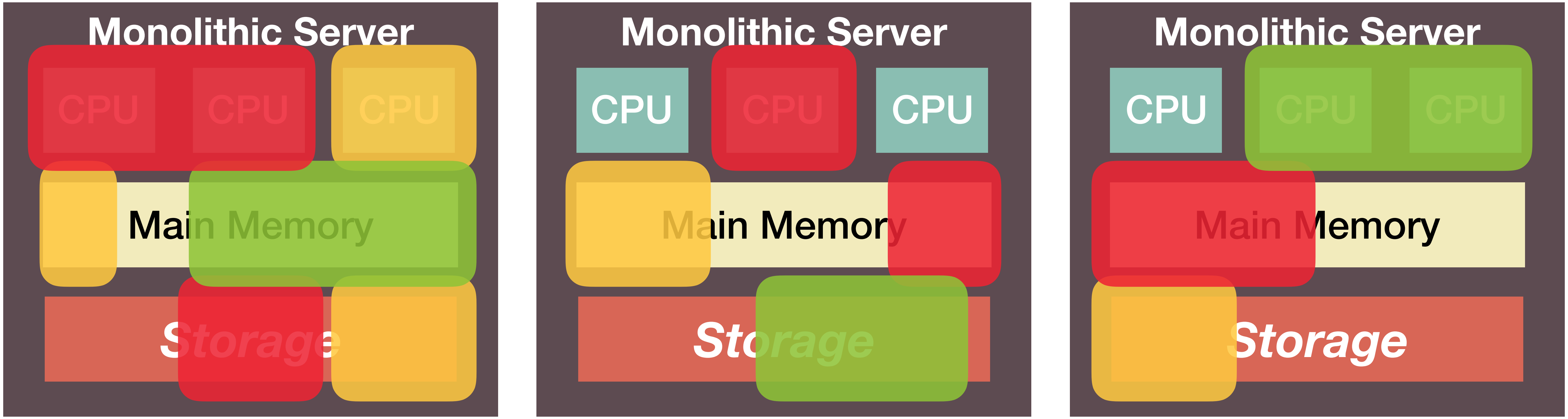
- Great support of heterogeneity
- Very flexible in resource management

- But needs hardware, network, and OS changes

Is there any less disruptive way to achieve better resource utilization and elasticity?

Virtually Disaggregated Datacenter

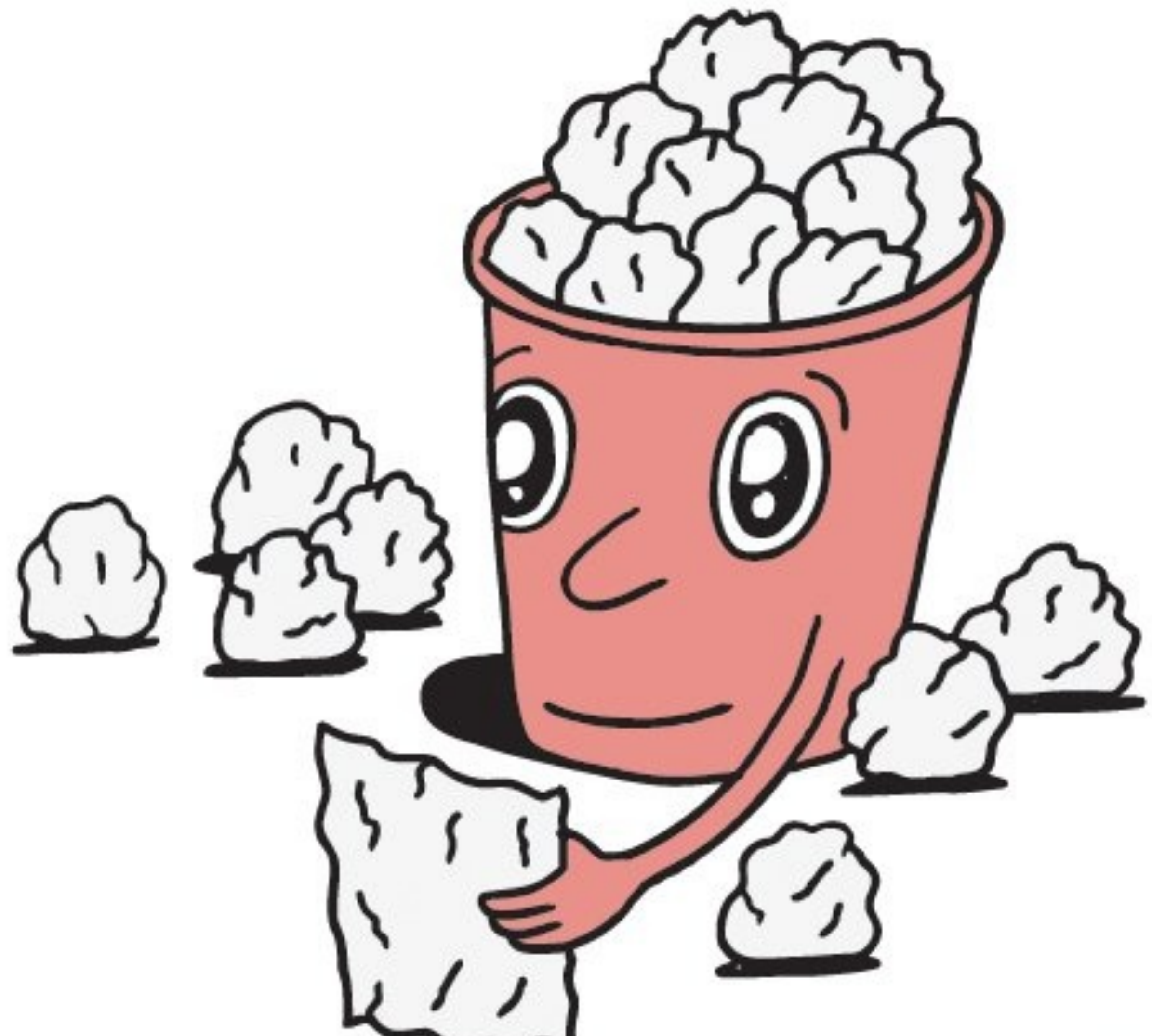
- Use resources on remote (distributed) machines



Using Remote/Distributed Resources

- Was a popular idea in 90s
  - Remote memory/paging/swap
  - Network block device
  - Distributed shared memory (DSM)

- No production-scale adoption
  - Cost of network communication
  - Coherence traffic



REVISIT YOUR OLD IDEAS WITH NEW EYES.

Remote/Distributed Memory in Modern Times

- New application trends
  - Large parallelism
  - New computation and memory requirements
  - New programming models

- Network is 10x-100x faster
  - InfiniBand: 200Gbps, <500ns
  - GenZ: 32-400GB/s, <100ns

Recent New Attempts

- Distributed Shared Memory
  - Grappa
  - Hotpot (Distributed Shared Persistent Memory)
- Network swapping
  - InfiniSwap
- Non-coherent distributed memory
  - VMware
- How to communicate across nodes?
  At what level of transparency?

Message Passing

- New programming languages
  - Use message passing instead of shared memory to do thread communication
  - Golang channel, Erlang actors, Akka library

- New application development practices
  - Use multiple processes instead of multithreading
  - Use message passing instead of shared memory
  - Nginx, Apache Web Server, Node.js

Splitting Containers with Message Passing

- Splitting a container across physical machines
  - Both processes and memory
  - Elastic, good resource utilization

- Use message passing for both inter- and intra-process communication
  - No coherence traffic
  - Easy to track and optimize inter-node communication

Challenges in Splitting Containers

- Performance overhead of crossing network
- Performance imbalance
- Management of split resources
- QoS
- Failure handling

Conclusion

- Splitting computation and memory not a new idea
- But new application and network properties
- Splitting container across physical machines
- Use message passing for all inter- and intra-process communication
- More challenges and opportunities

Thank you
Questions?

WukLab

wuklab.io

PURDUE UNIVERSITY

Mitsume: an Object-Based Remote Memory System

Shin-Yeh Tsai, Yiying Zhang

One-Sided Remote Memory/NVM

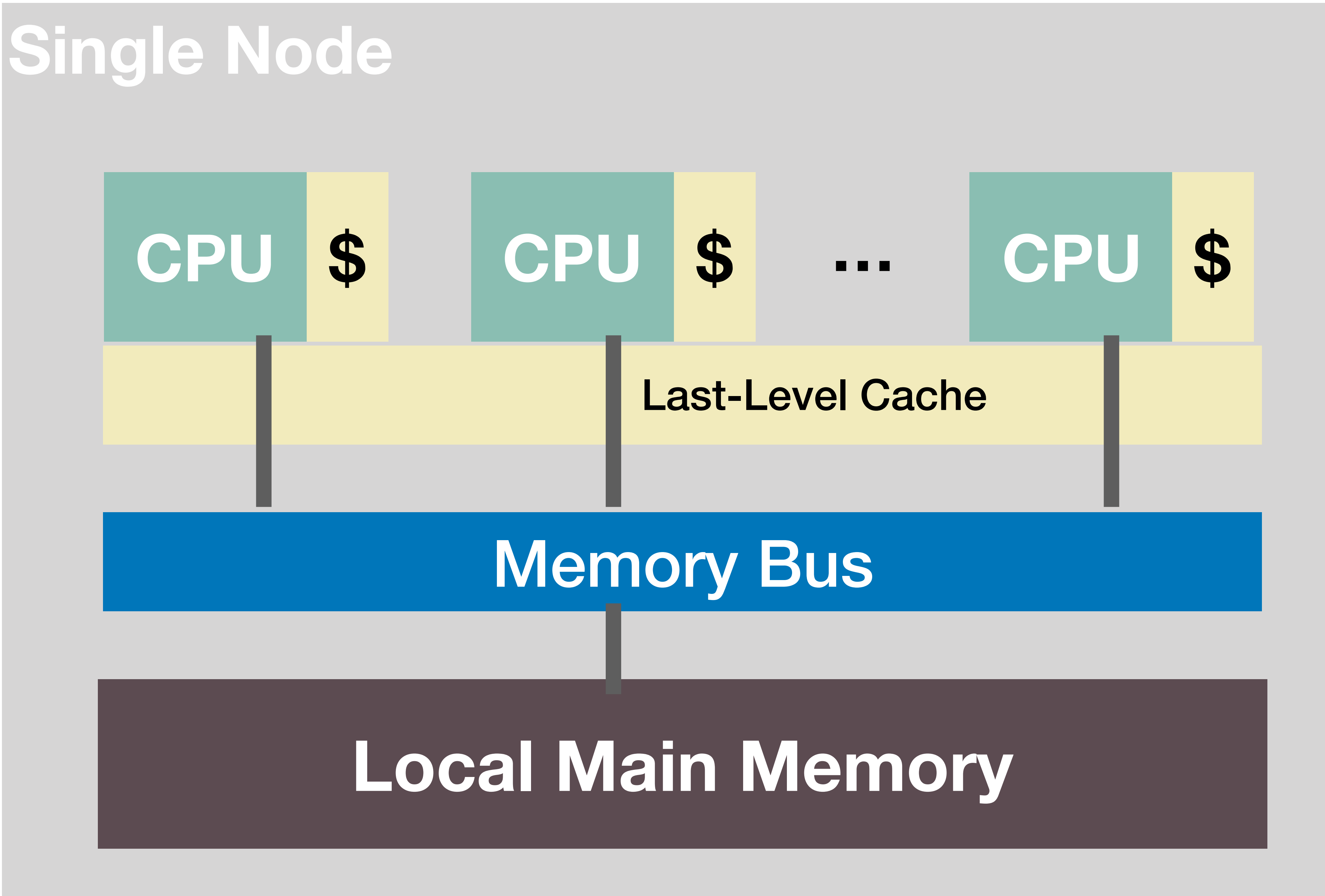
- One-sided devices
  - Devices without (general) computation power
  - Can only be read and written to with limited, low-level APIs
  - Disaggregated memory
  - NVMe over fabrics
- Cheap, low energy

Remote Memory Challenges and Opportunities

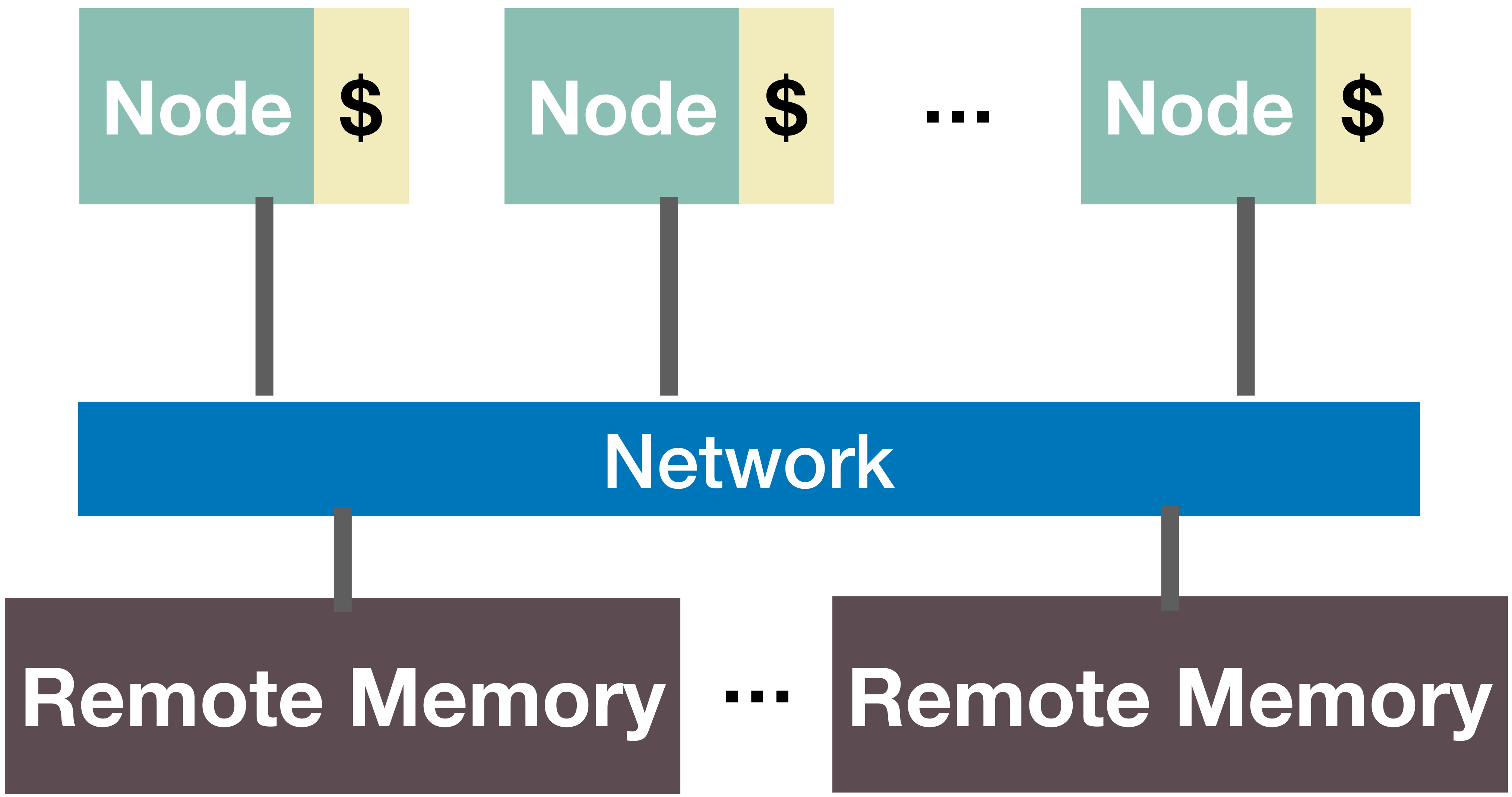
- Challenges
  - No computation power at memory
  - Remote memory can fail independently
- Opportunities
  - Can trace and control (remote) memory accesses
  - No local accesses that can violate atomicity

How to use one-sided remote memory/NVM devices?

Local Multiprocessor Shared Memory



Remote Memory



Remote Memory and Local Memory Comparison

- Similarities
  - No computation power
  - Multiple processors (cores) can read and write to

- Differences (remote memory)
  - No hardware coherence
  - Can fail independently (and more often)
  - Larger but slower than local memory

Our View of Remote Memory

- Treat remote memory as a raw data-store hardware
  - Similar to DRAM chips in local main memory
  - Fast, cheap data store

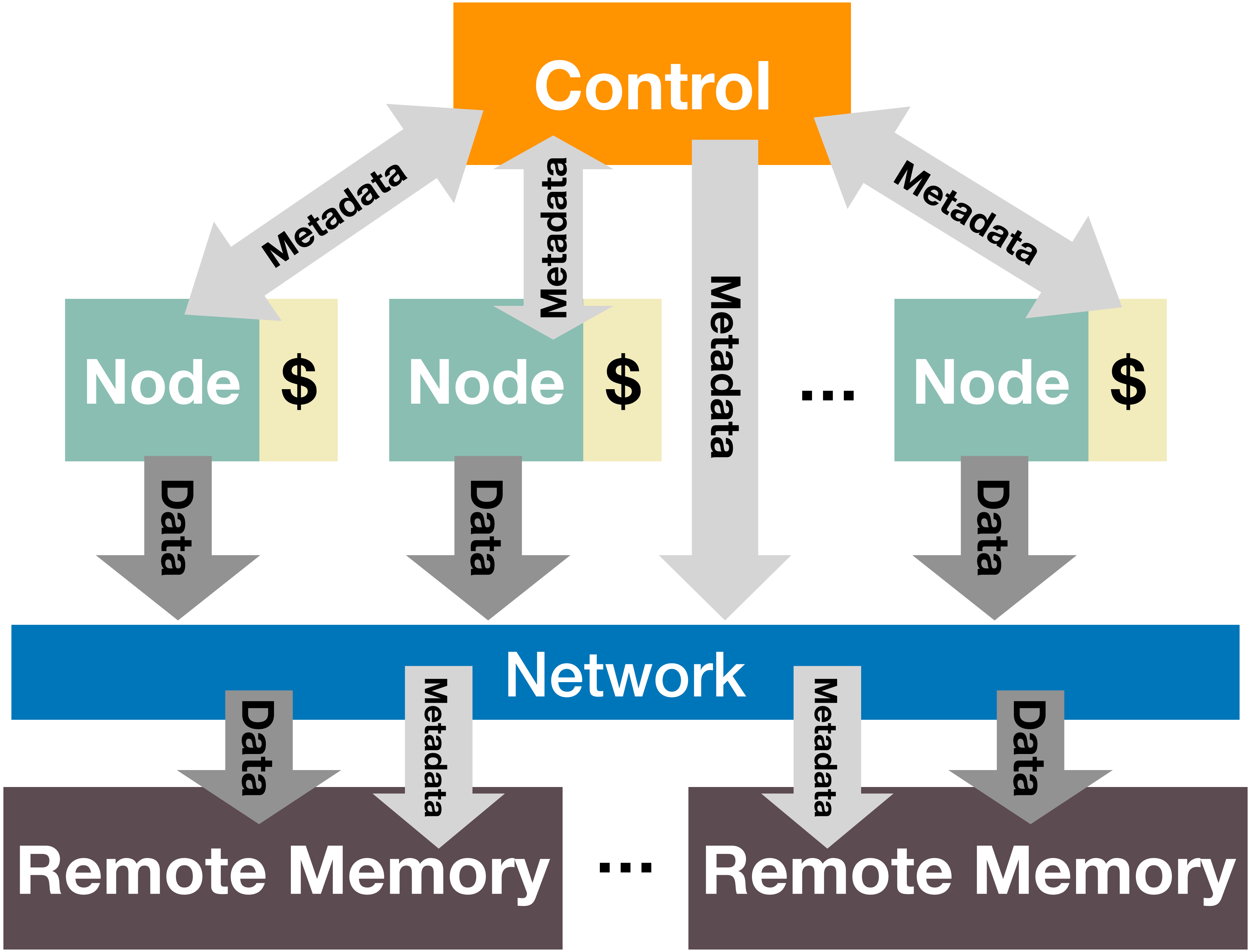
- Extract the control and intelligence apart
  - Similar to memory controller in local main memory
  - But managed in software

Mitsume*: an Object-Based Remote Memory System

- Separate data and control path
- Data: one-sided
  - Client nodes read/write to remote memory
  - Multiple processors (cores) can read and write to
- Control: two-sided
  - Global software controller manages remote memory and talks to clients via two-sided operations

* Mitsume means three eyes in Japanese and is from the manga and anime Mitsume ga Tooru (the Three-Eyed One)

Mitsume Architecture



Mitsume Data Organization

- Data stored and located by “object”
- Updates to an object guaranteed atomic and append-only
- Each object can have multiple versions
- Flexible physical locations of (versions) objects
- Each object can have their own replication factor

Global Control

- Allocate physical memory at remote memory
- Garbage collection
- Ensures QoS for different clients
- Resource management and load balancing
- Failure handling
- Security

Usage Models

- Key-value store
- Version system
- Remote swap
- Messaging system
- Pub/Sub

Conclusion

- One-sided memory/NVM devices are useful
- Learn from local memory system
- Separate data and control of remote memory
- Many usage possibilities of remote memory

Thank you
Questions?

WukLab

wuklab.io

PURDUE UNIVERSITY