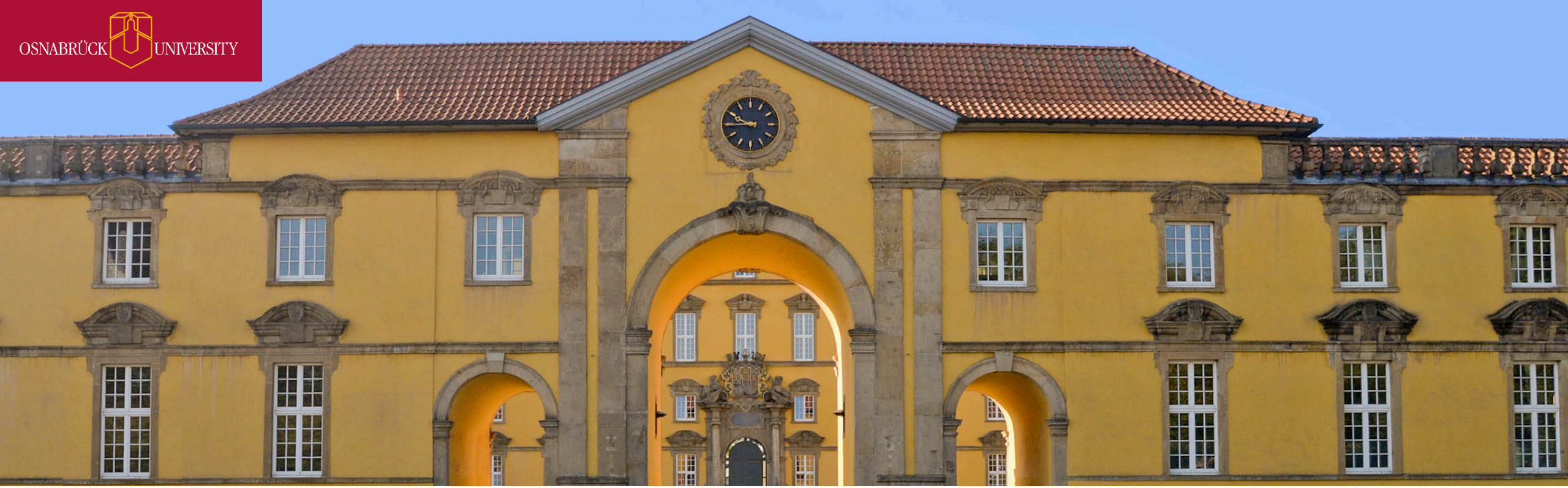
OSNABRÜCK UNIVERSITY

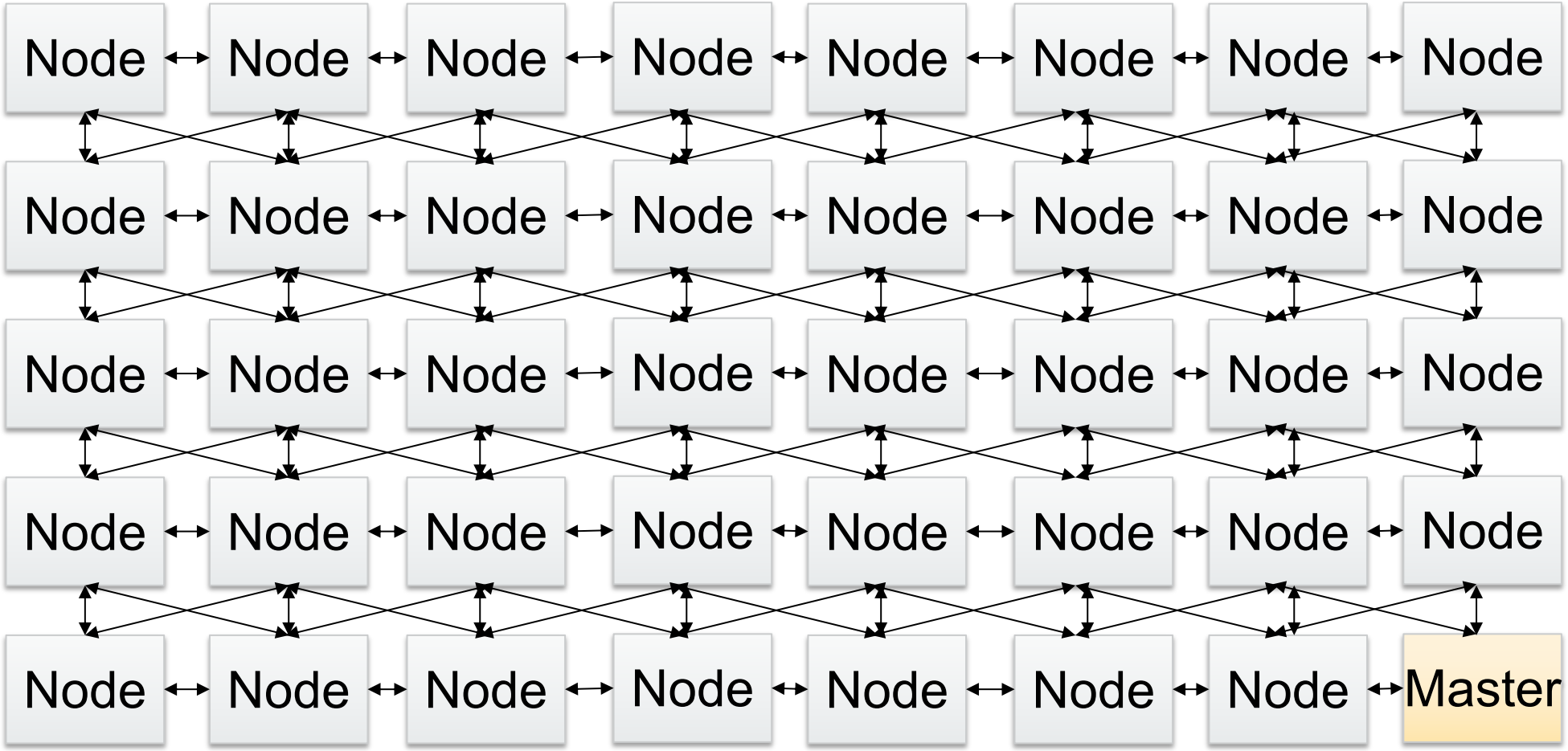


Managing Resources in the Data Center with the MxKernel

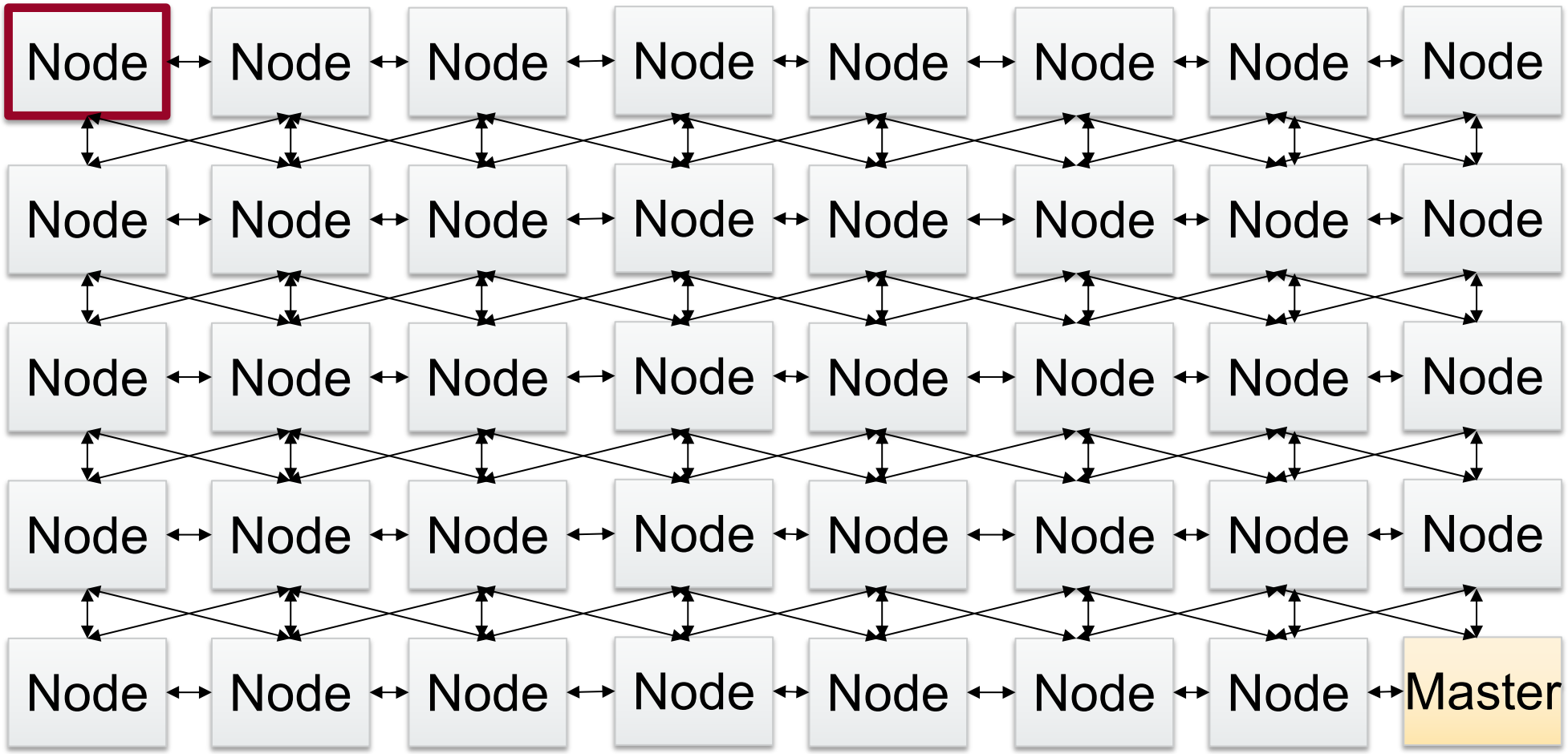
Bamberg, Sep 28, 2023

Michael Müller (michael.mueller@uos.de)

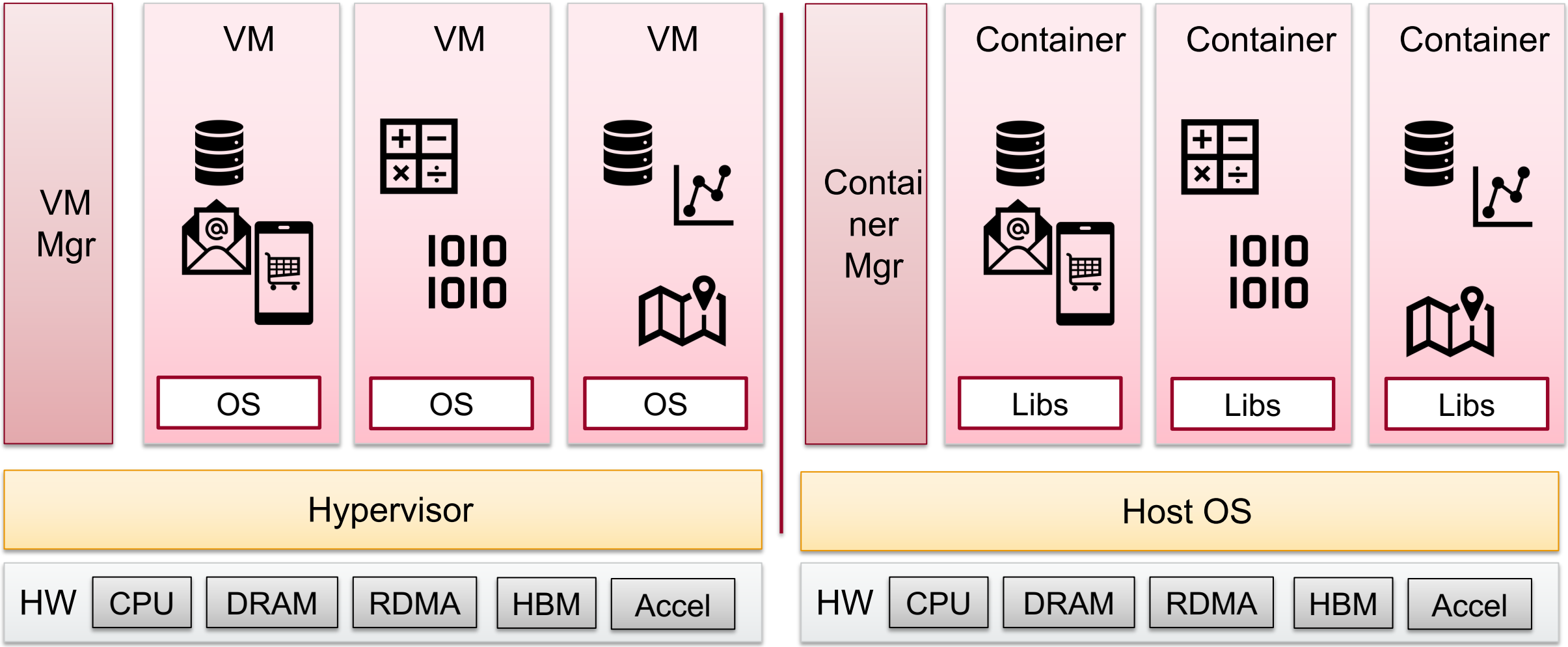
Today's Data Center



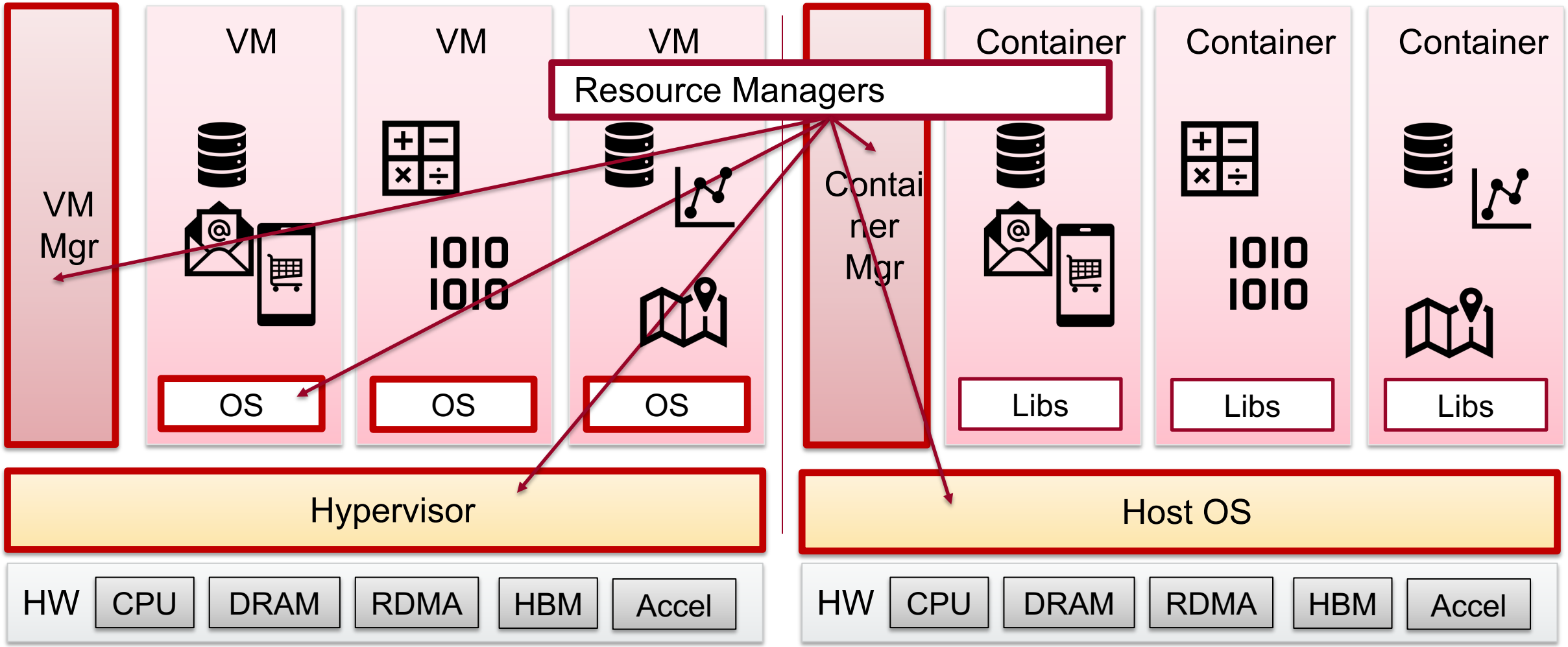
Today's Data Center



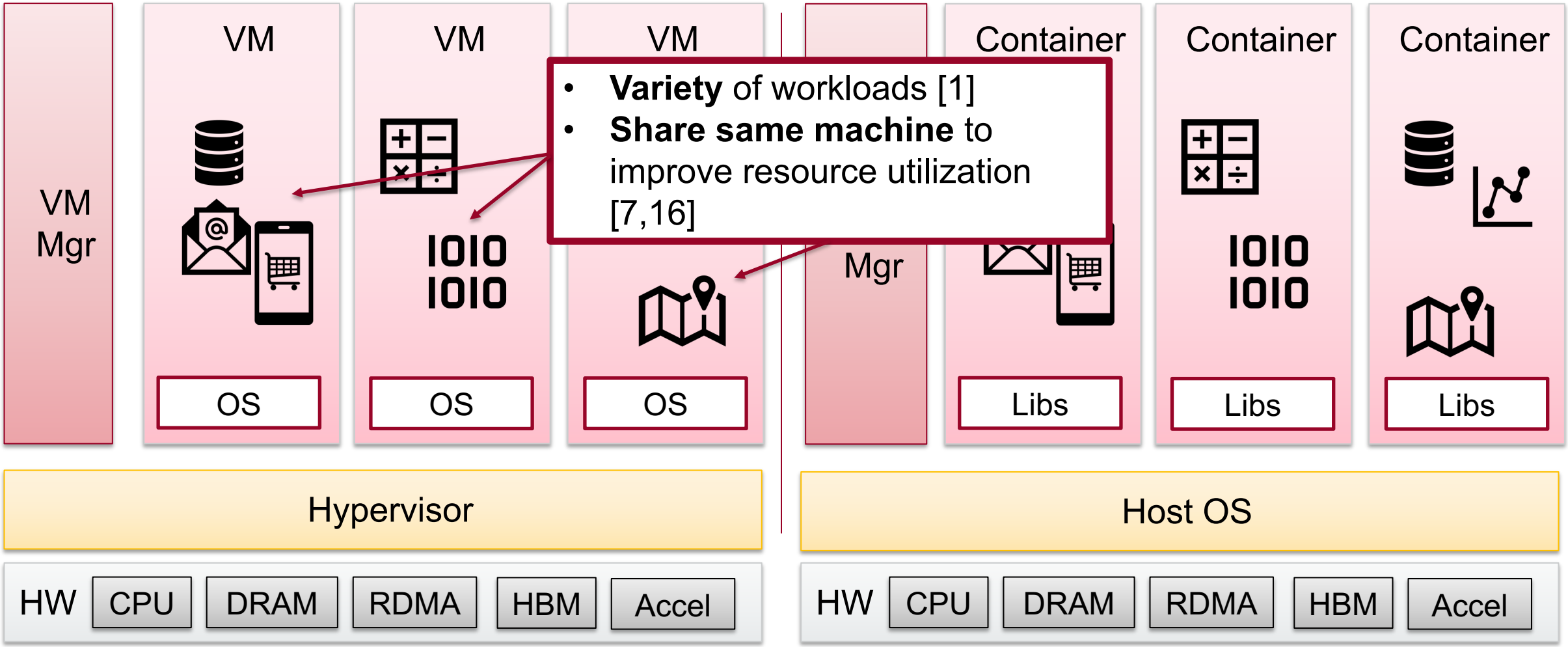
Today's Data Center Software Stack



Today's Data Center Software Stack

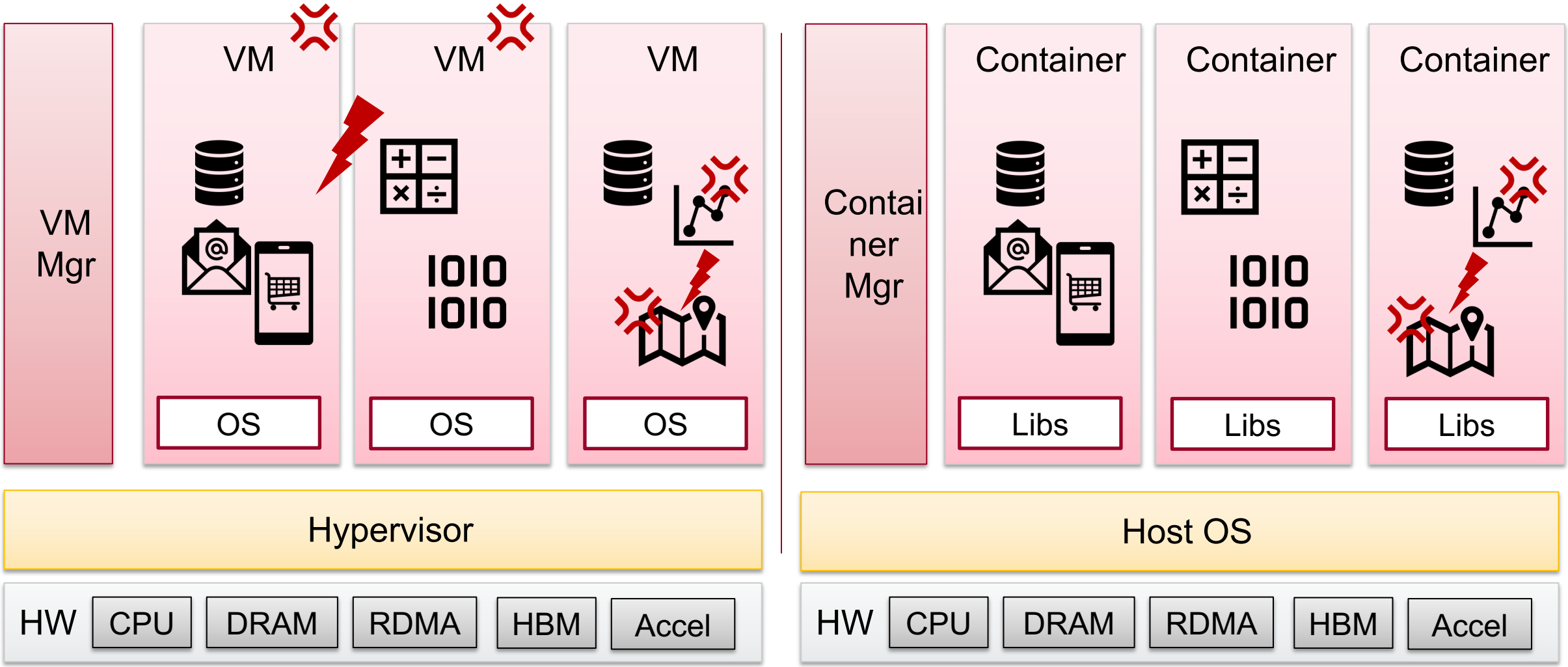


Today's Data Center Software Stack

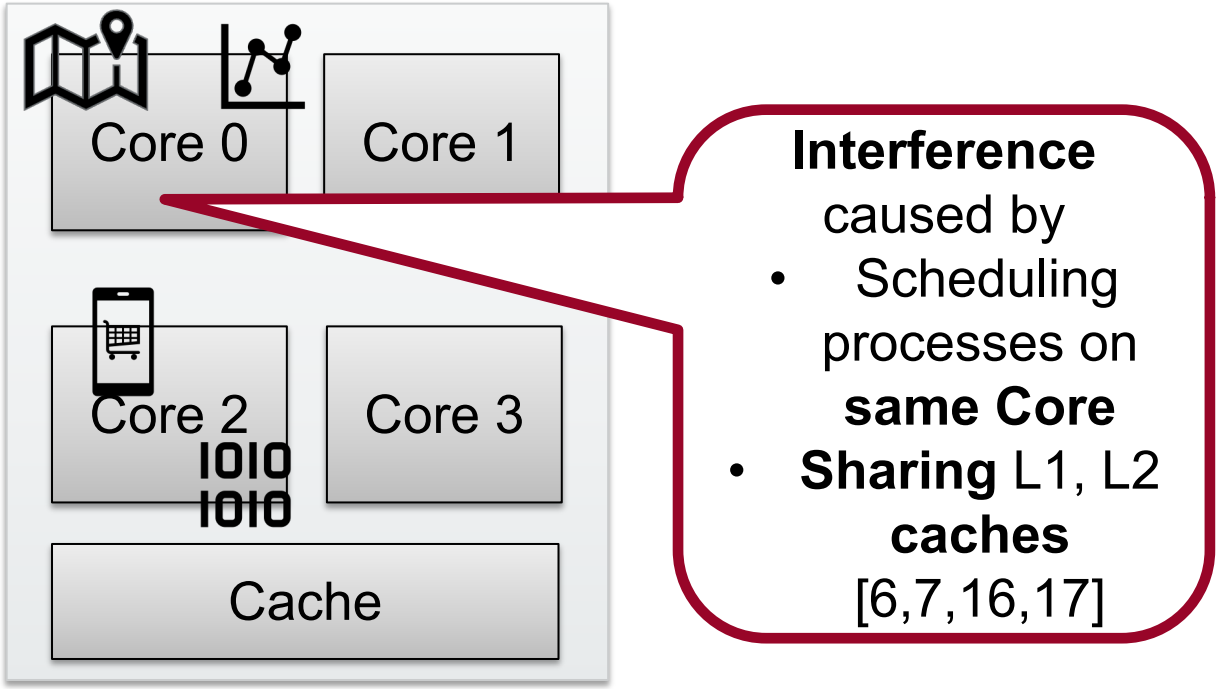


- **Variety** of workloads [1]
- **Share same machine** to improve resource utilization [7,16]

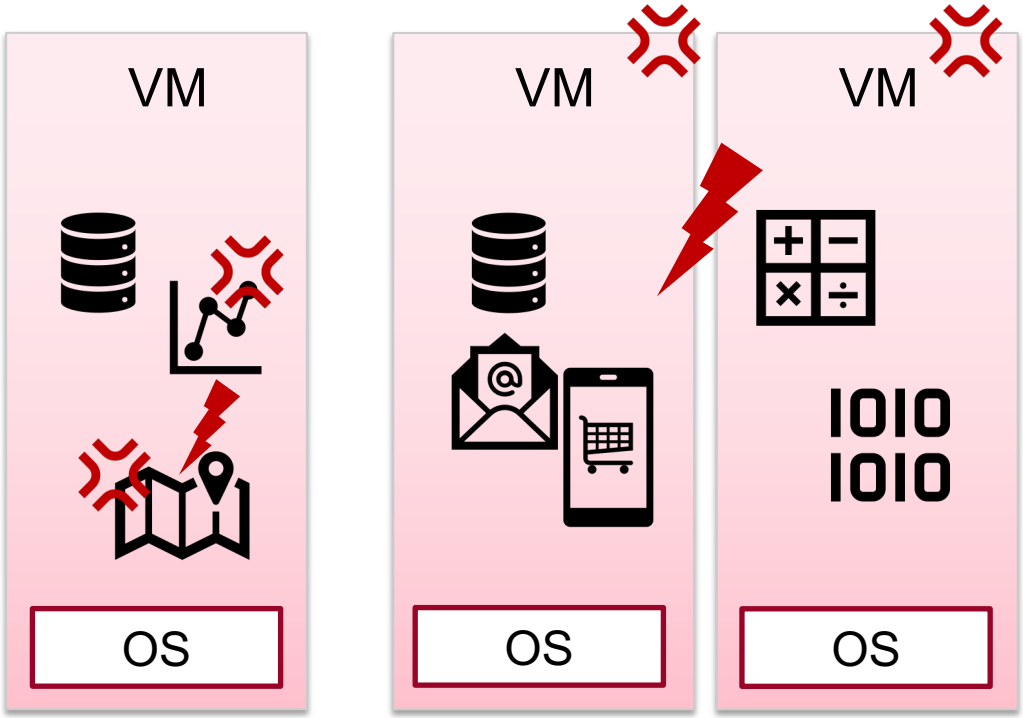
Challenges in Resource Management



Challenges in Resource Management: Interferences



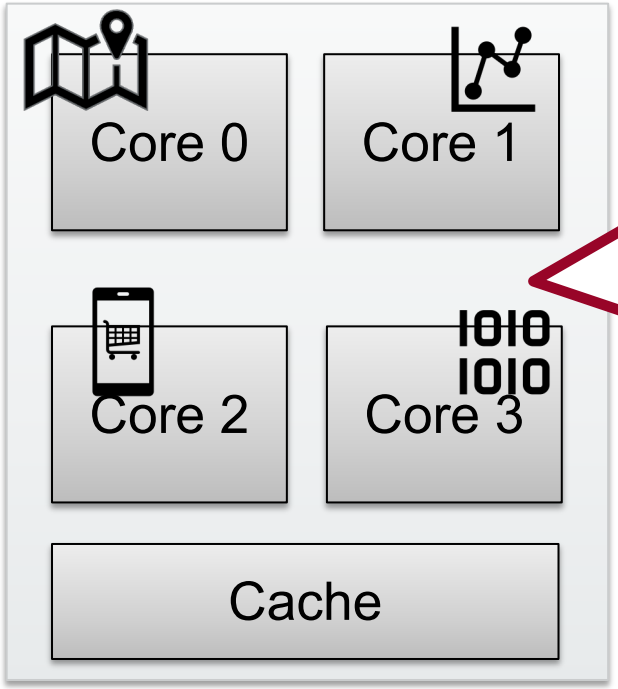
Interference caused by
- Scheduling processes on **same Core**
- **Sharing** L1, L2 **caches** [6,7,16,17]



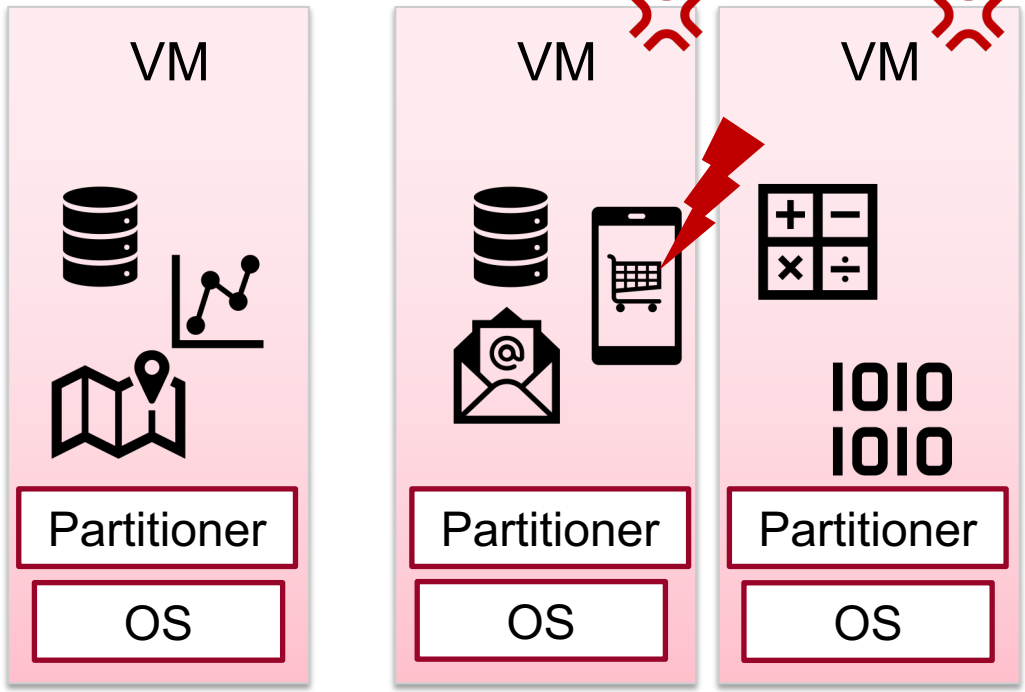
Consequences:
- **High** tail latencies
- **Degraded** throughput

⇒ SLA **penalties** for operator

Challenges in Resource Management: Interferences

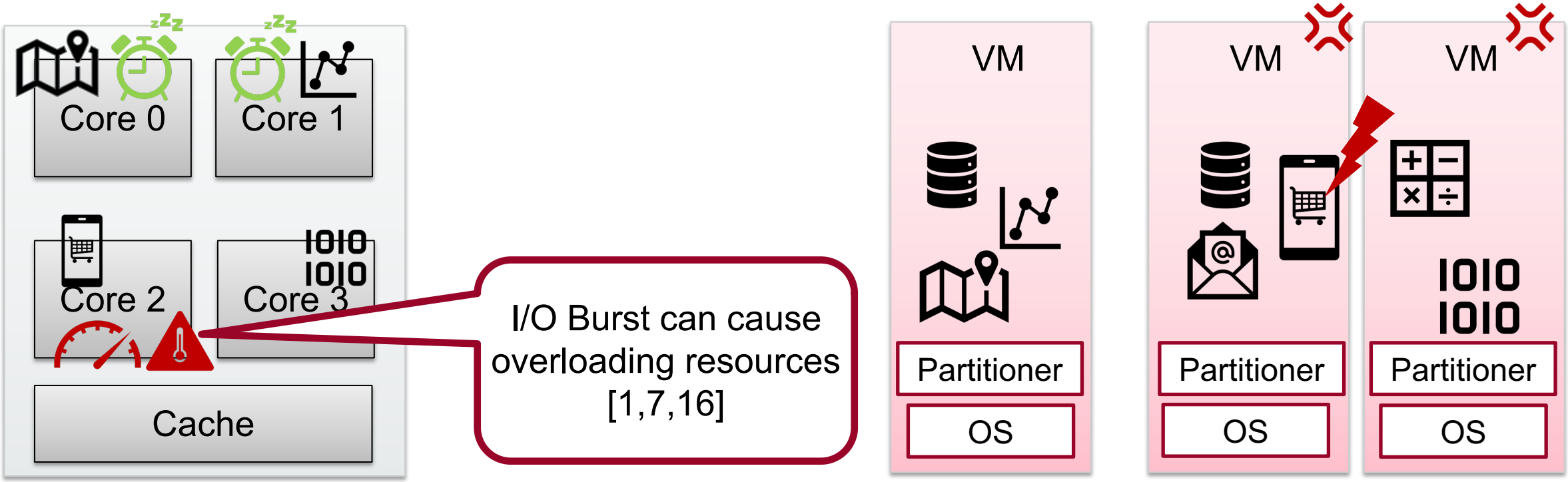


Common **Solution: Resource Partitioning** [7,15,17]



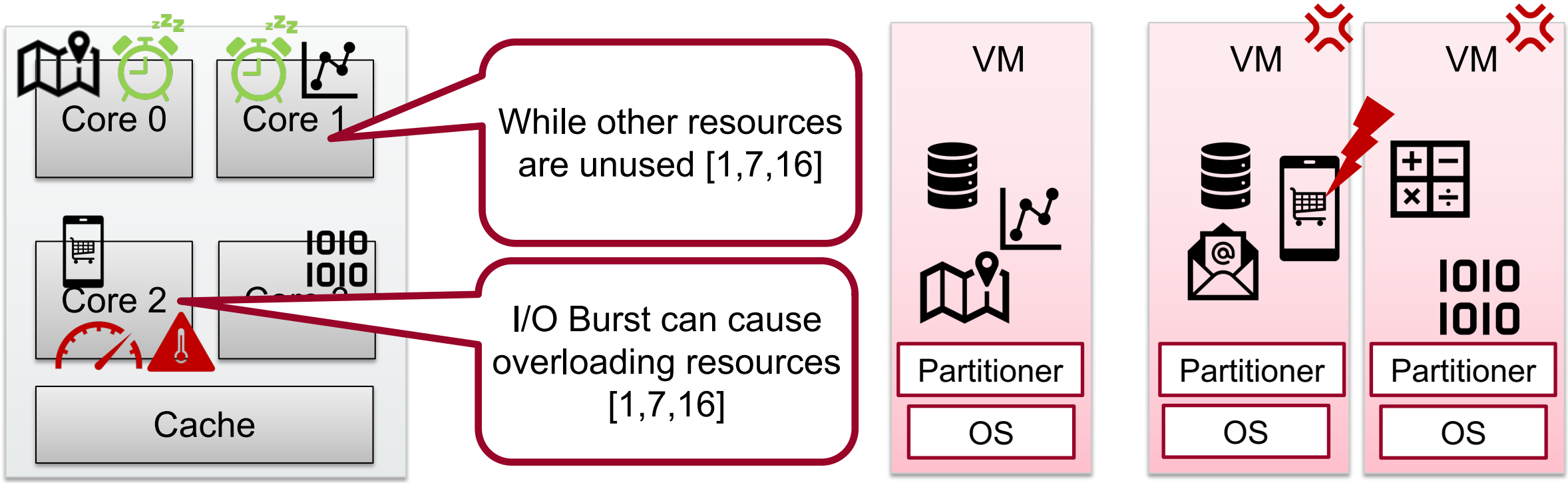
Pitfall: To **rigid** partitioning may cause **overloading** VMs/containers **and** under-utilization of resources **simultaneously** [6,7,11,17]

Challenges in Resource Management: Allocation



Pitfall: To **rigid** partitioning may cause **overloading** VMs/containers **and** under-utilization of resources **simultaneously** [6,7,11,17]

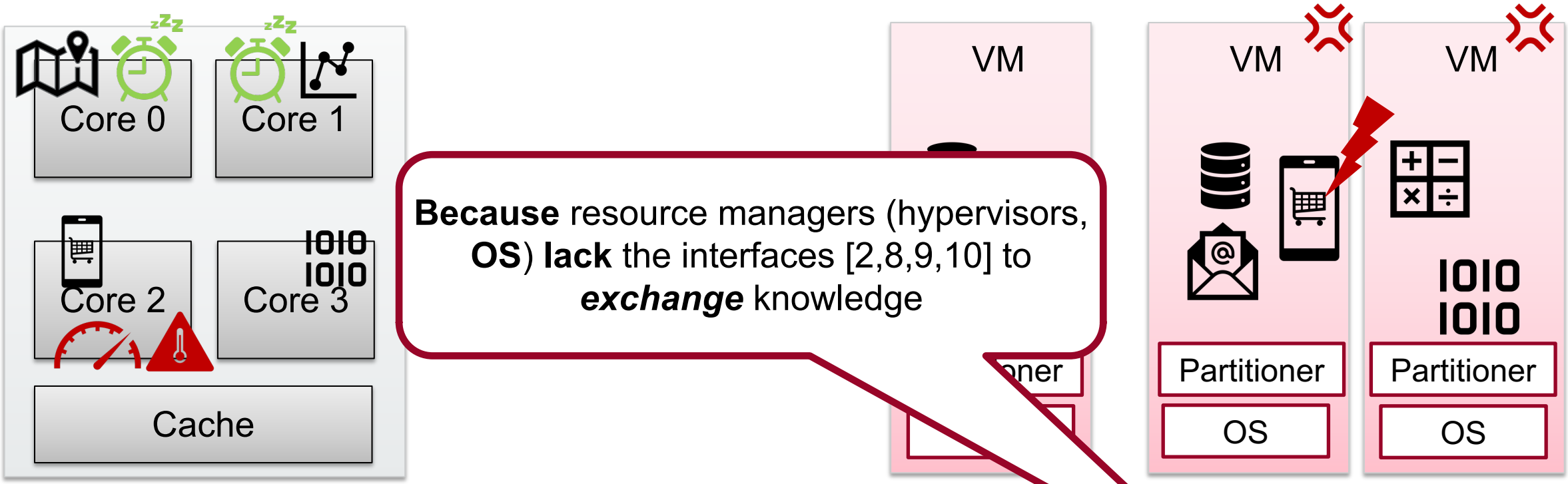
Challenges in Resource Management: Allocation



Solution: **Dynamic** Resource Partitioning **or Allocation** [4,6,7,17]

But, determining resource **allocation** for VMs and application partitions is **difficult** without **application knowledge** [6,11] **and** decisions must be made **fast** [6,11]

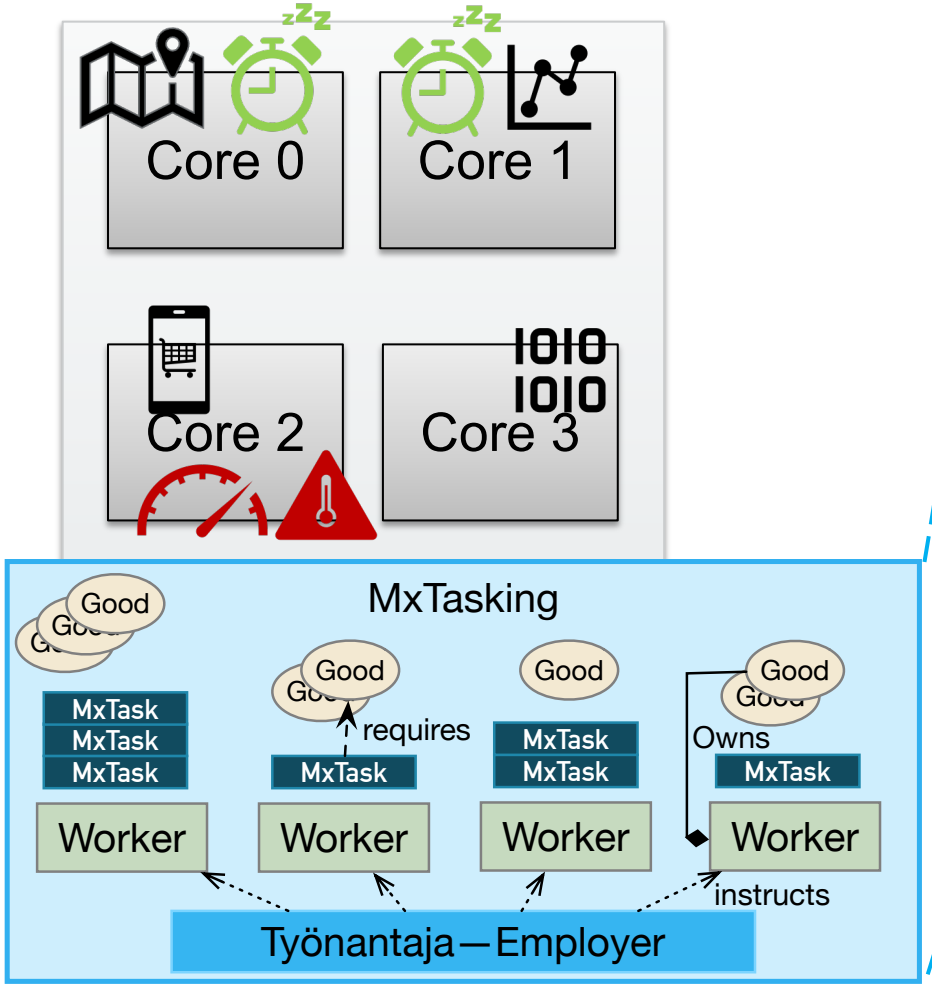
Challenges in Resource Management: Allocation

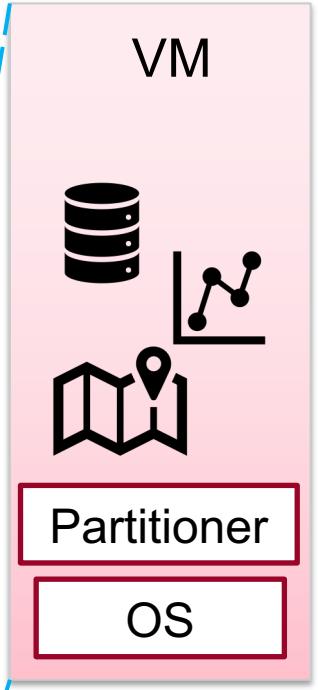


Solution: **Dynamic** Resource Partitioning **or Allocation** [4,6,7,17]

But, determining resource **allocation** for VMs and application partitions is **difficult** without **application knowledge** [6,11] **and** decisions must be made **fast** [6,11]

MxKernel Concepts



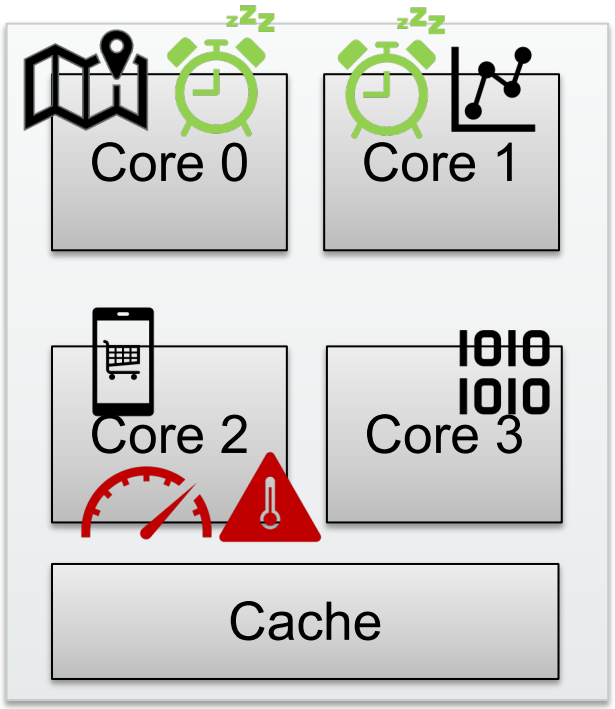


MxTasks

- Provide **annotations** to **predict future** resource demands
- Are **non-preemptable, light-weight closed units** of work
- Provide **task-parallel** programming
- **Benefits** for **automatic prefetching** and **synchronization** already shown*

*) Mühlig, J. and Teubner, J. 2021. MxTasks: How to Make Efficient Synchronization and Prefetching Easy. *Proceedings of the 2021 International Conference on Management of Data* (New York, NY, USA, Jun. 2021), 1331–1344.

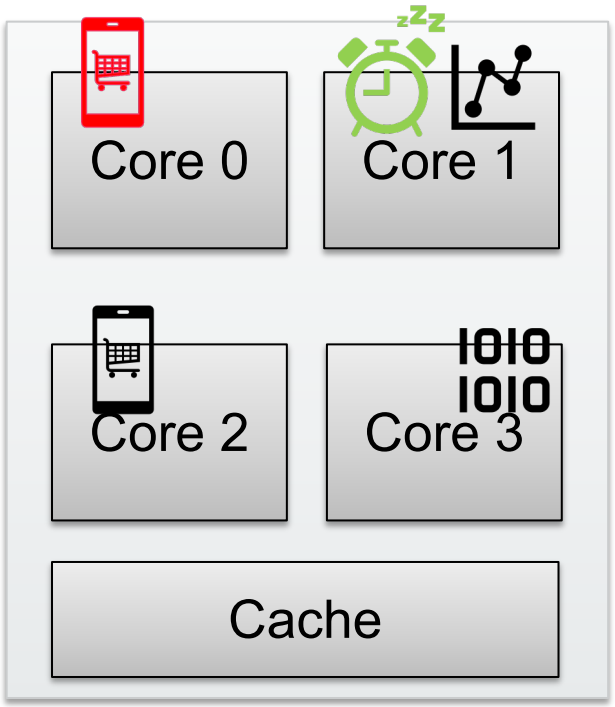
MxKernel Concepts



Cells

- Represent **processes** of a workload
- **Minimize interference** between **online** and **offline services** among workloads
- **Fast** reaction to **load changes**
- **By**
  - ***Elastic*** resource domains
  - Featuring **task-based** runtime environment
  - **Managed** by **the zookeeper (Hoitaja)**

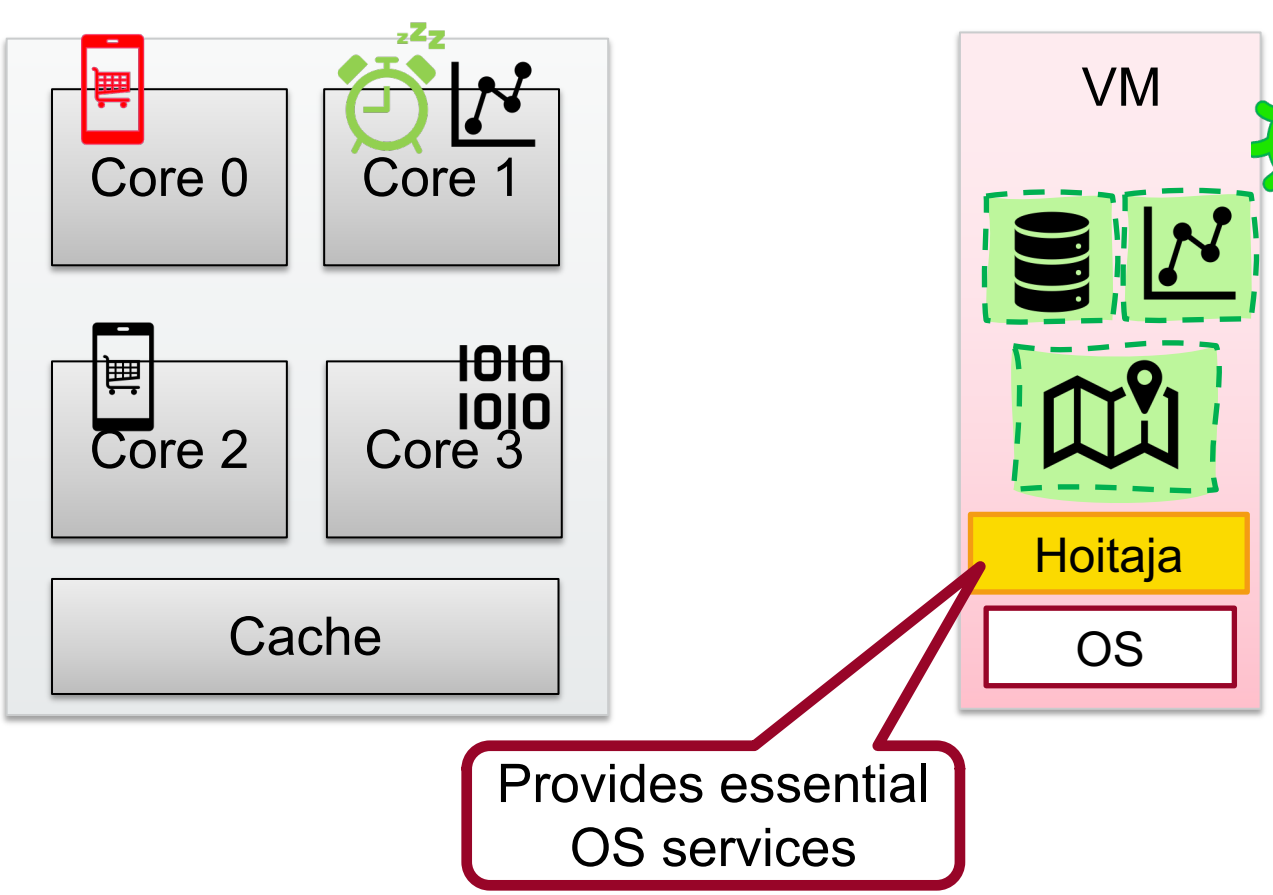
MxKernel Concepts



Cells

- Represent **processes** of a workload
- **Minimize interference** between **online** and **offline services** among workloads
- **Fast** reaction to **load changes**
- **By**
  - ***Elastic*** resource domains
  - Featuring **task-based** runtime environment
  - **Managed** by **the zookeeper (Hoitaja)**

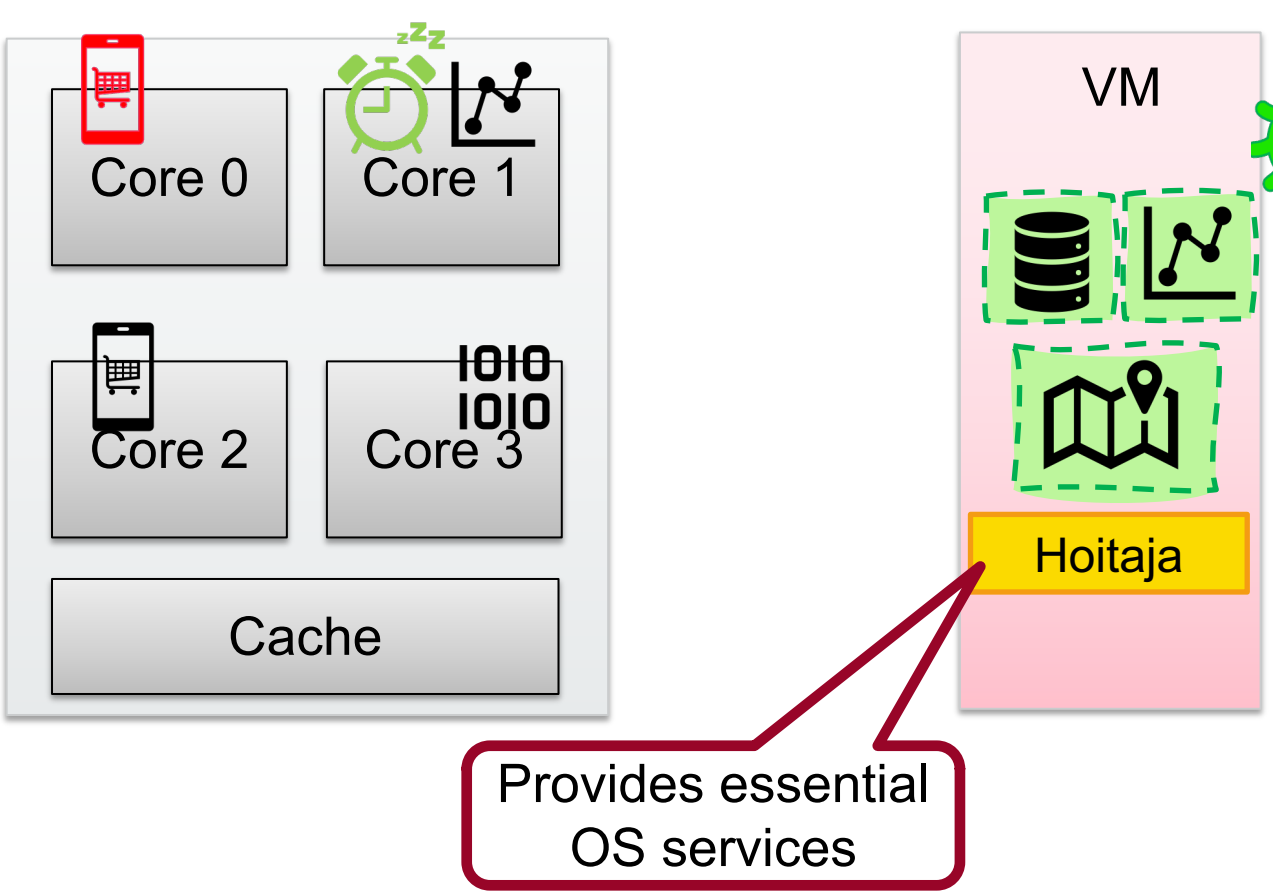
MxKernel Concepts



Cells

- Represent **processes** of a workload
- **Minimize interference** between **online** and **offline services** among workloads
- **Fast** reaction to **load changes**
- **By**
  - ***Elastic*** resource domains
  - Featuring **task-based** runtime environment
  - **Managed** by **the zookeeper (Hoitaja)**

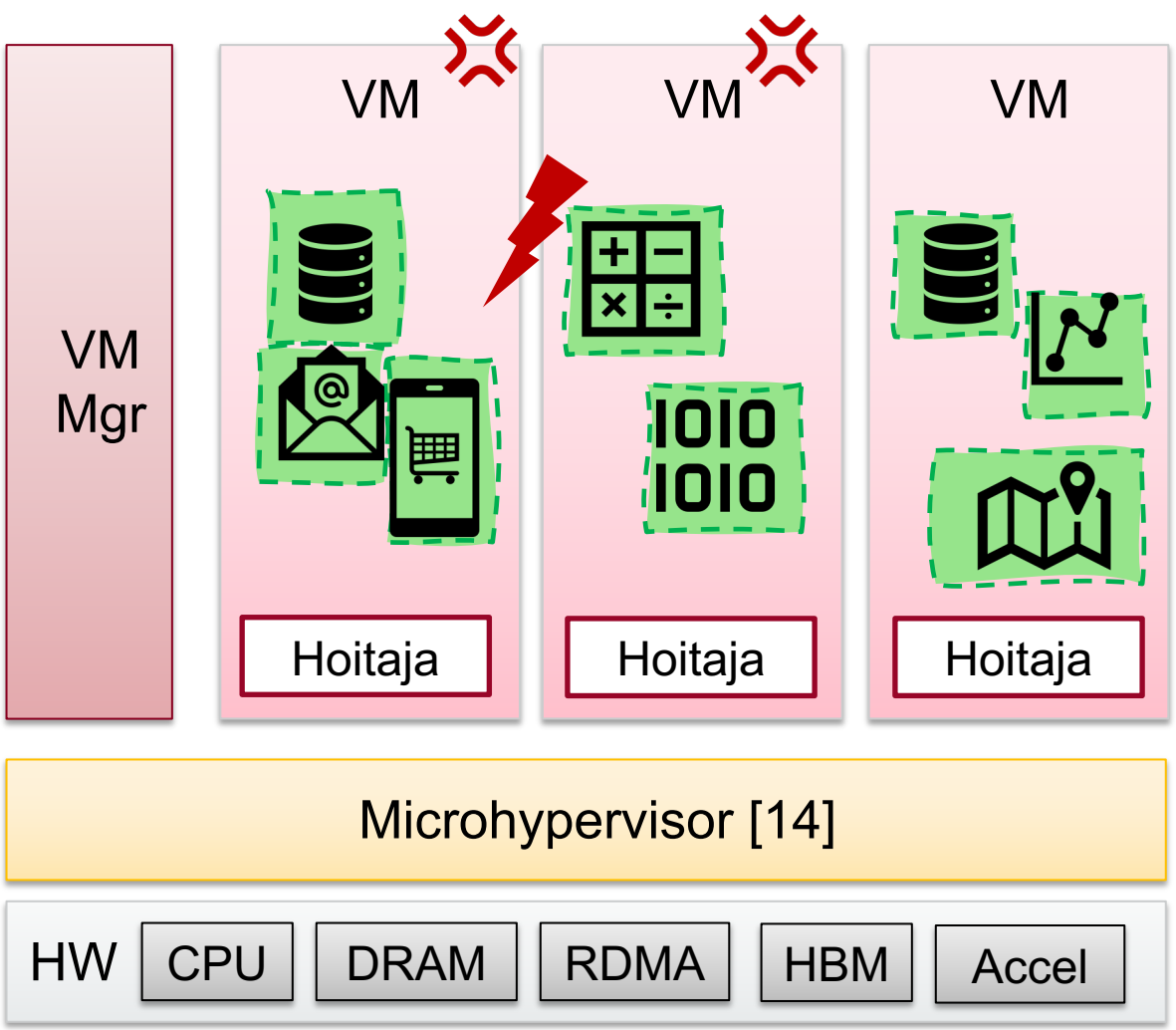
MxKernel Concepts



Cells

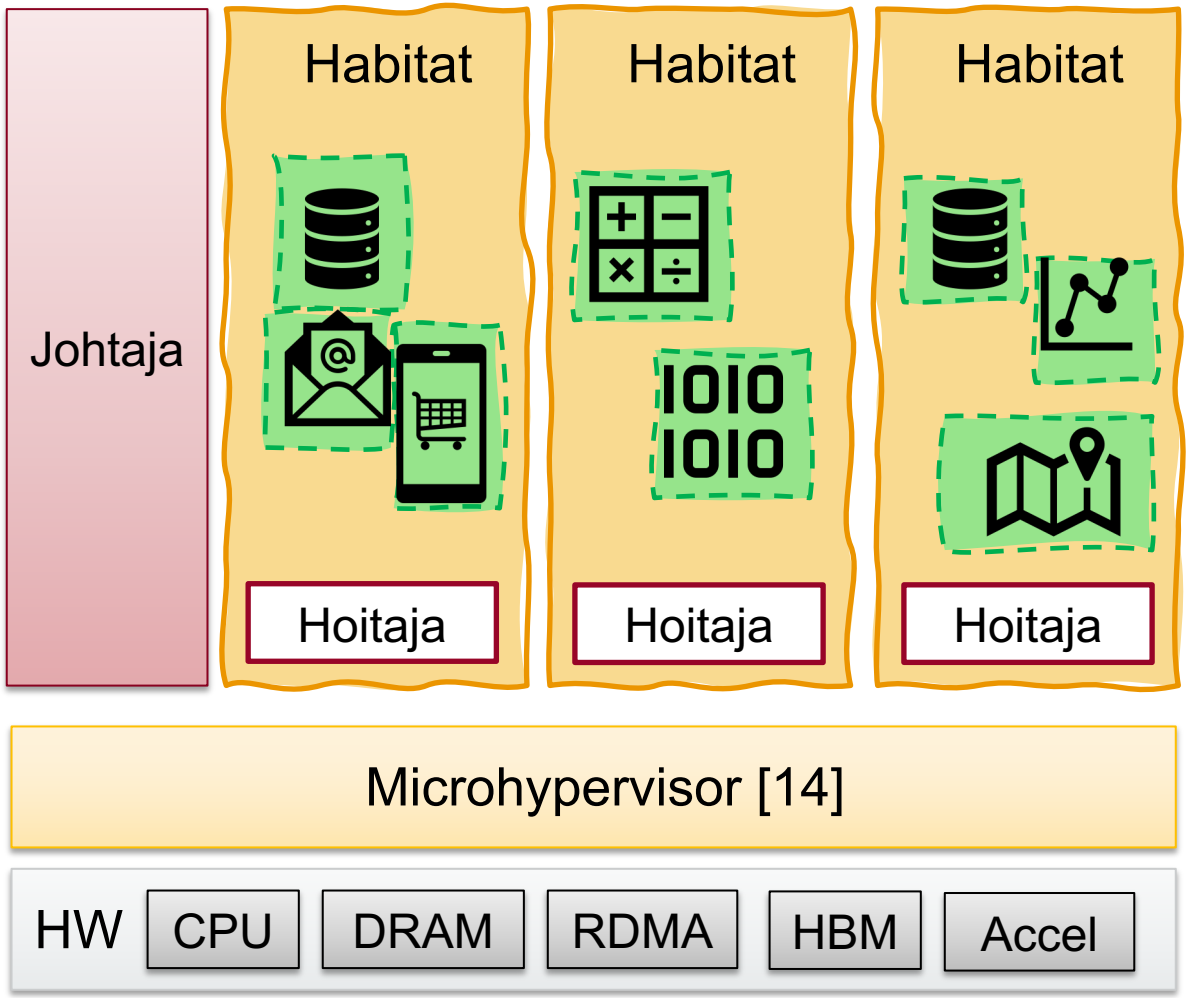
- Represent **processes** of a workload
- **Minimize interference** between **online** and **offline services** among workloads
- **Fast** reaction to **load changes**
- **By**
  - ***Elastic*** resource domains
  - Featuring **task-based** runtime environment
  - **Managed** by **the zookeeper (Hoitaja)**

MxKernel Concepts



Habitats

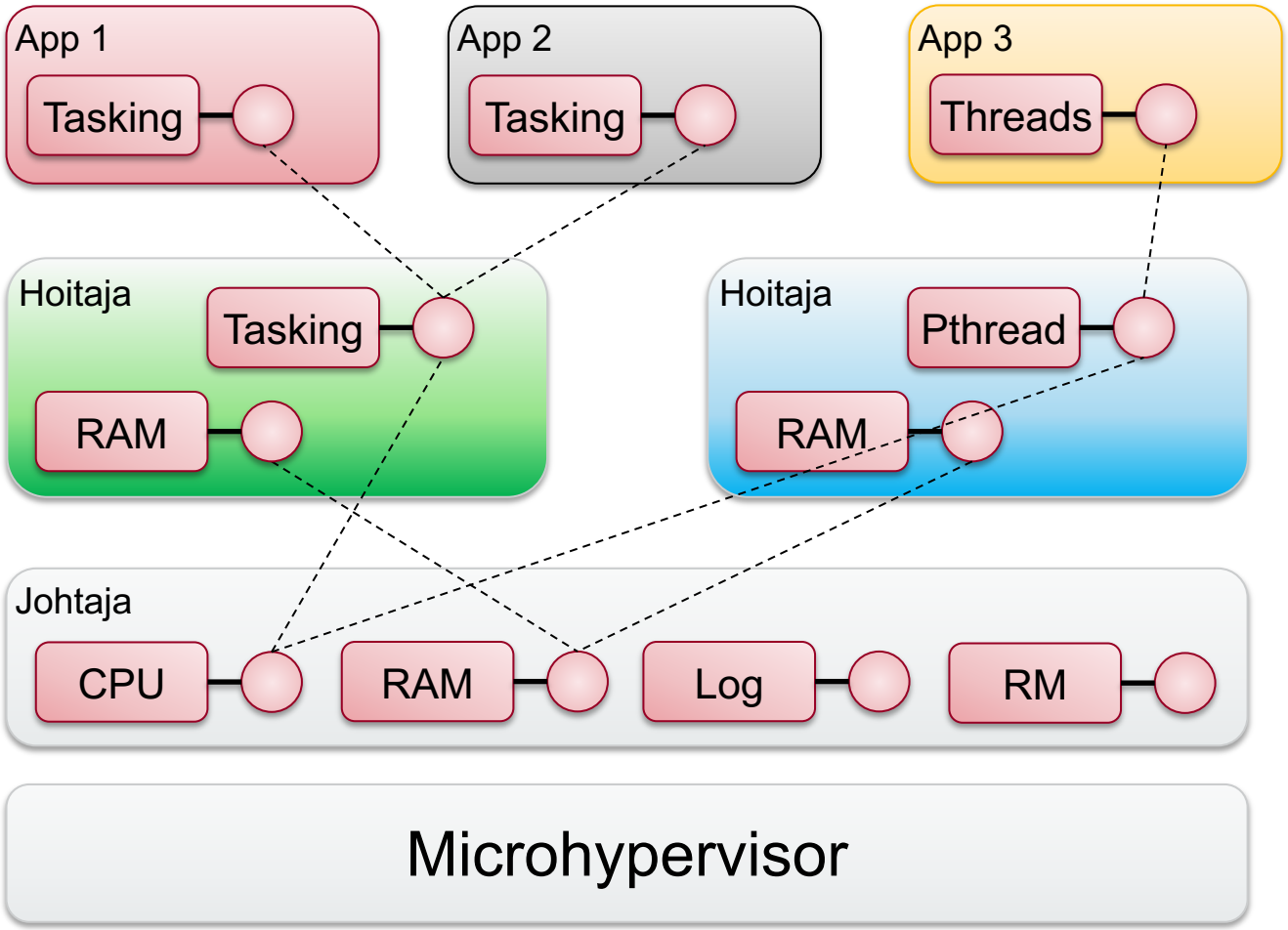
MxKernel Concepts



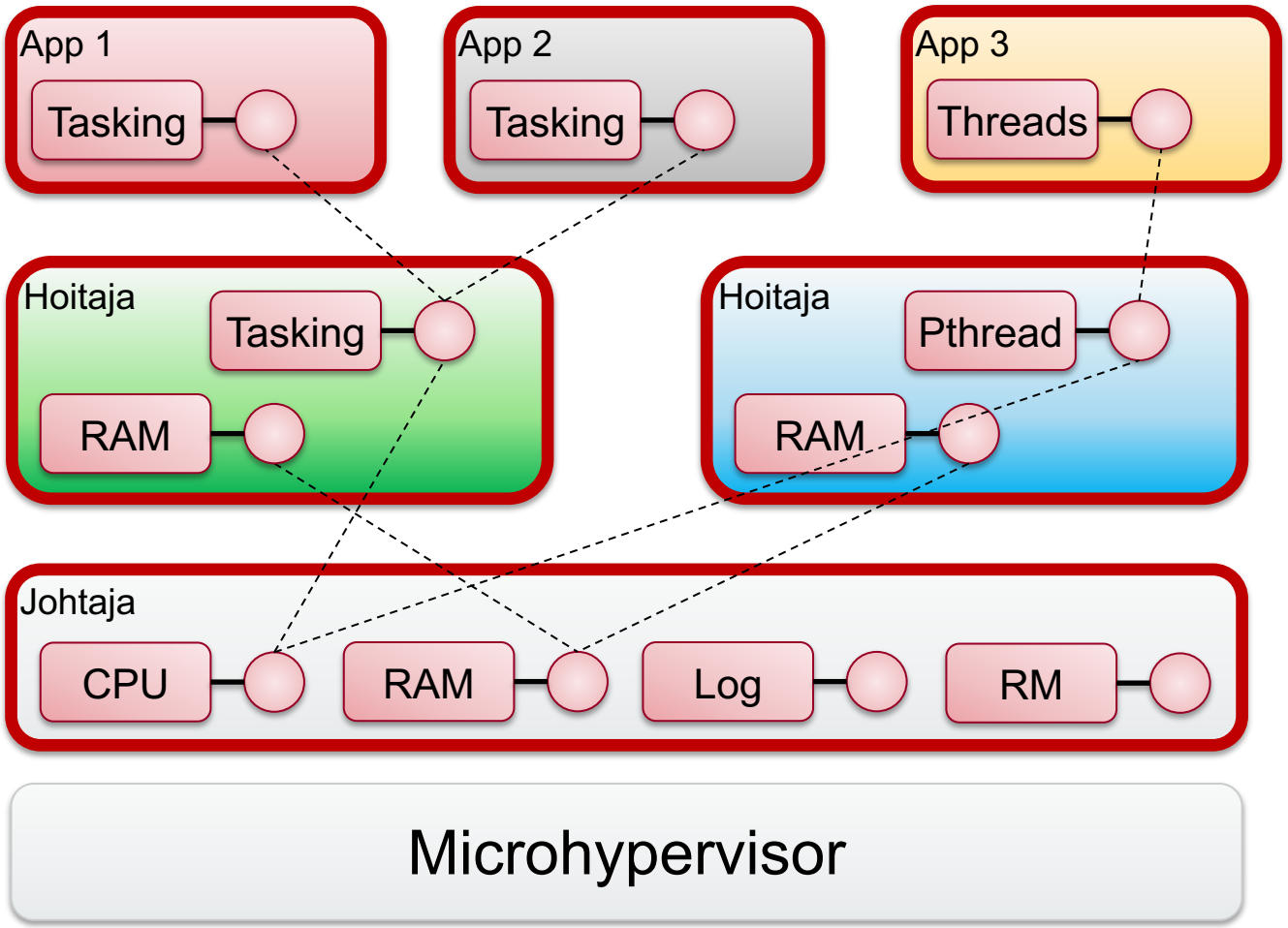
Habitats

- **Address**
  - **Fairness and Interferences** between workloads
- **By** providing
  - An **elastic** resource **container**
  - A **local** tailorable **resource manager, the zookeeper**
- **Managed** by **global** resource manager **(Johtaja, the director)**

MxKernel Architecture



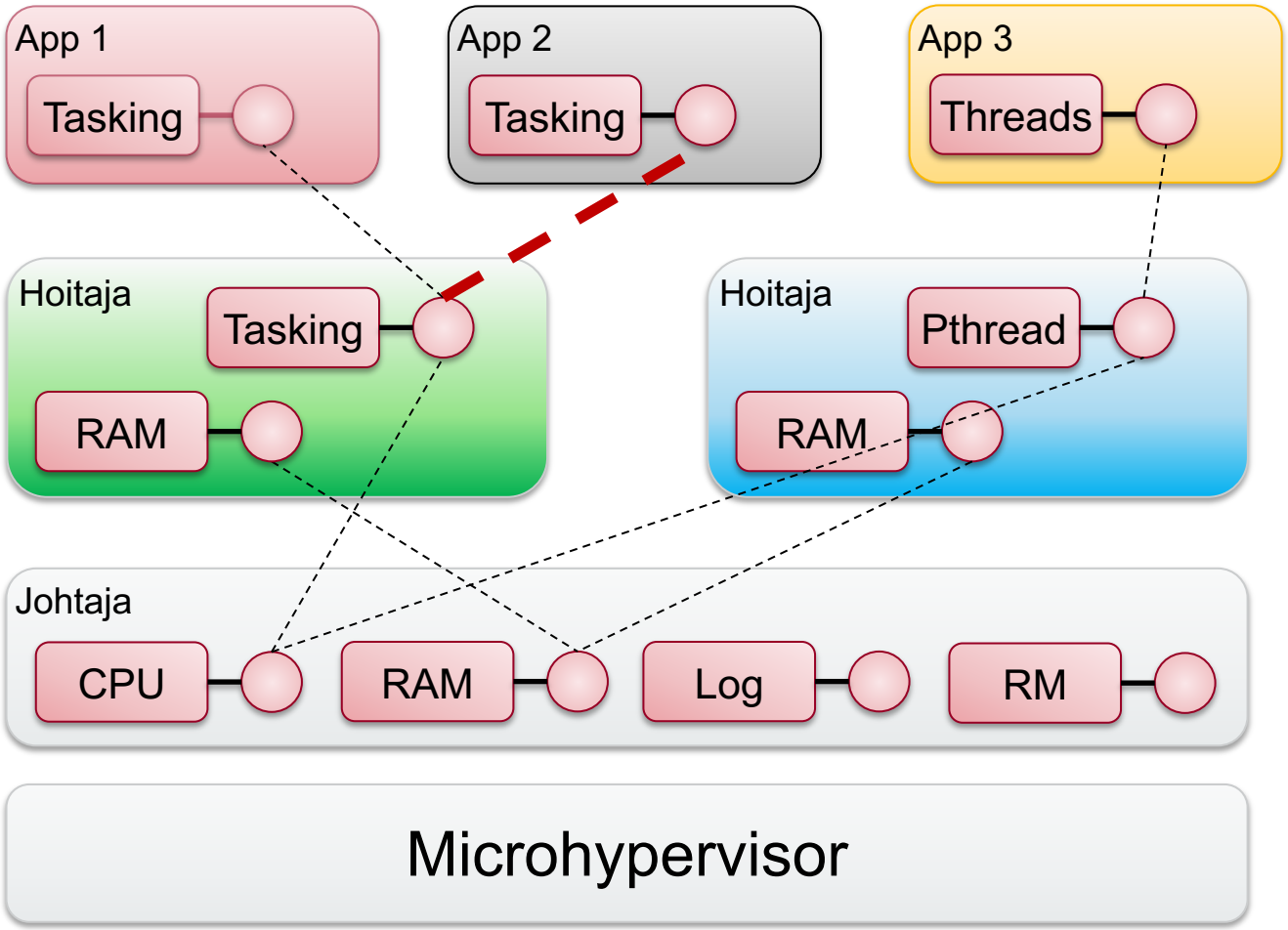
MxKernel Architecture



- Composed of components*

*) Based on the architecture of the Genode OS Framework (www.genode.org)

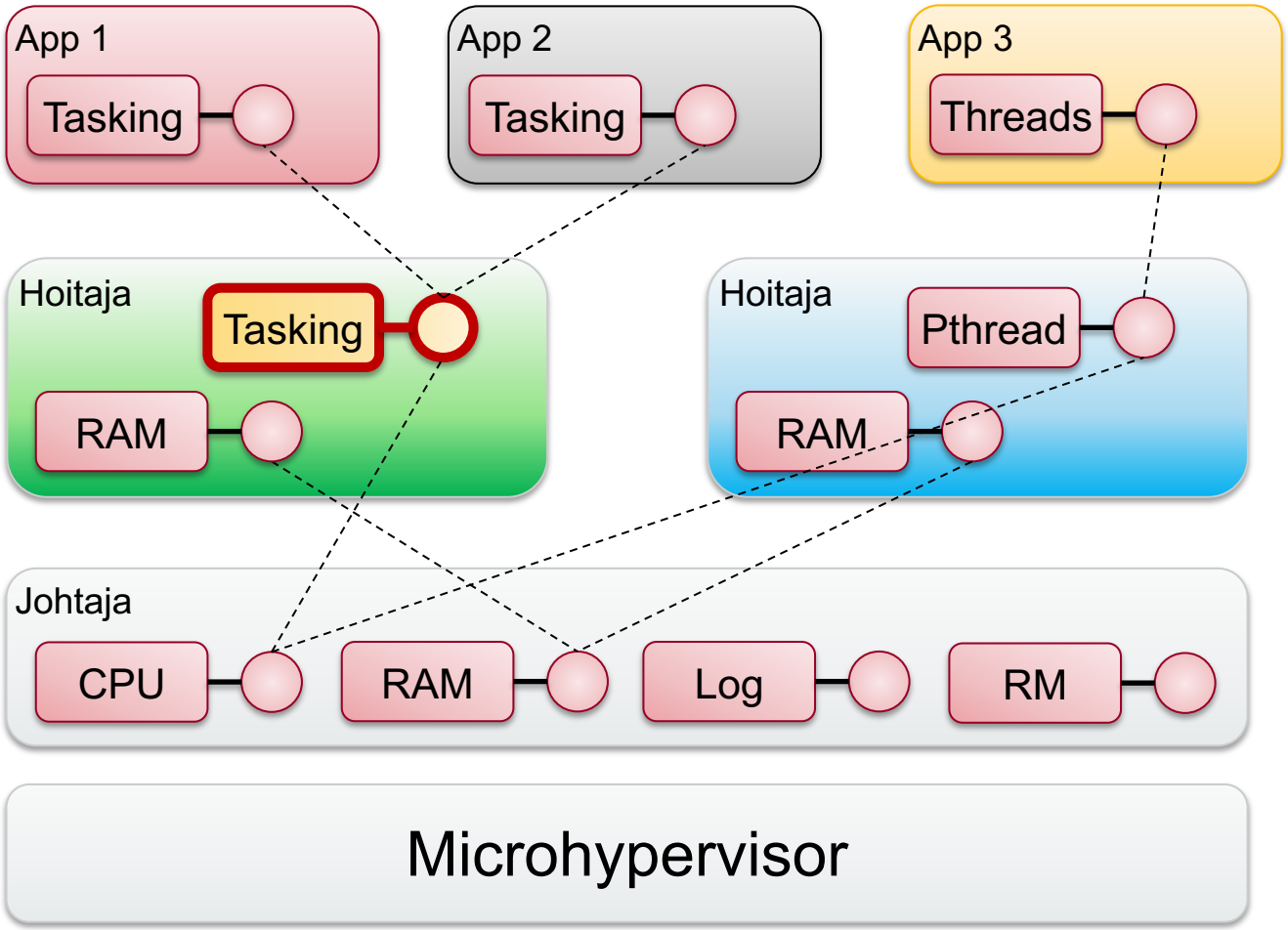
MxKernel Architecture



- Composed of components*
- Components interact via RPC

*) Based on the architecture of the Genode OS Framework (www.genode.org)

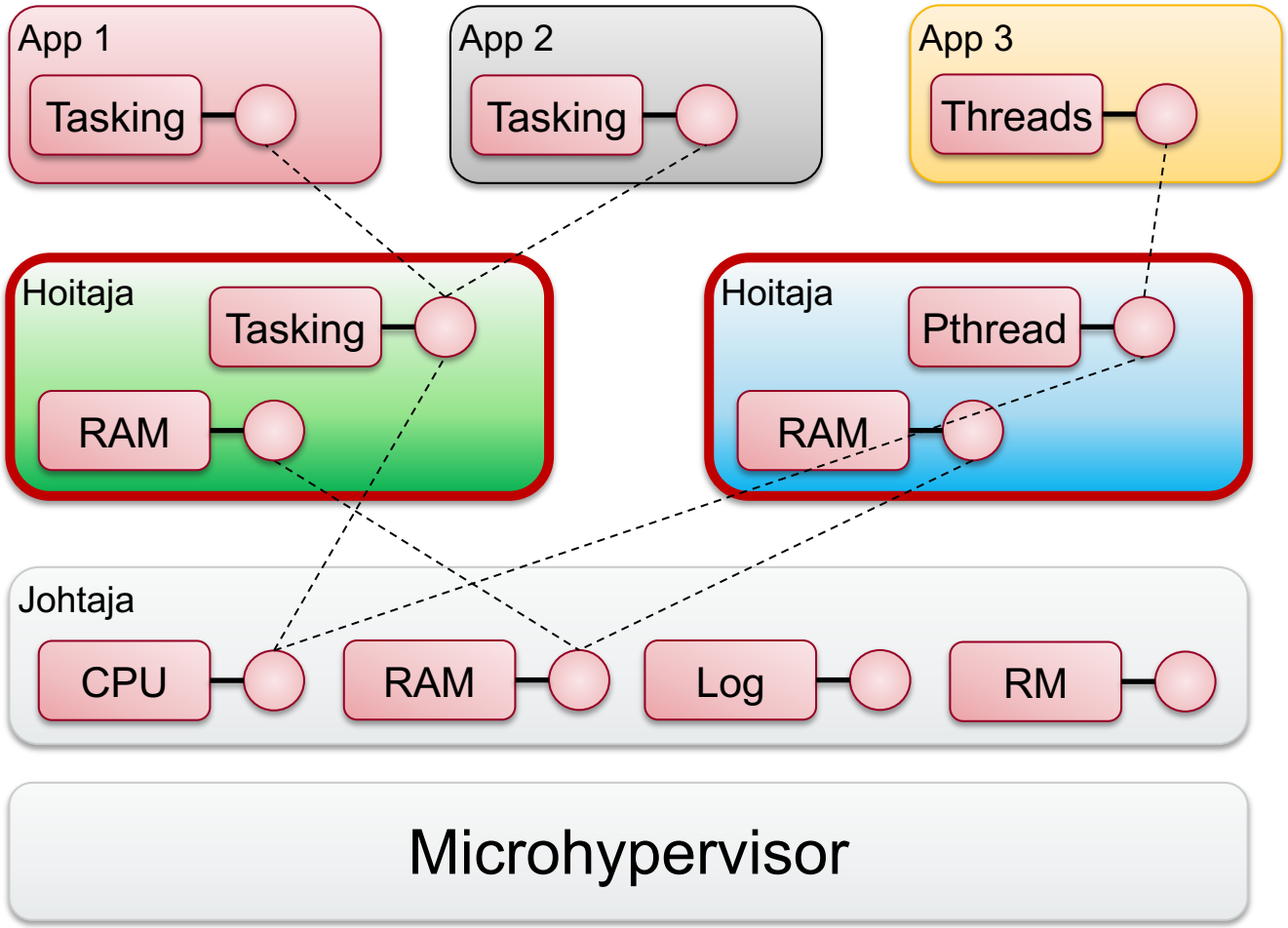
MxKernel Architecture



- Composed of components*
- Components interact via RPC
- And can offer (OS) services

*) Based on the architecture of the Genode OS Framework (www.genode.org)

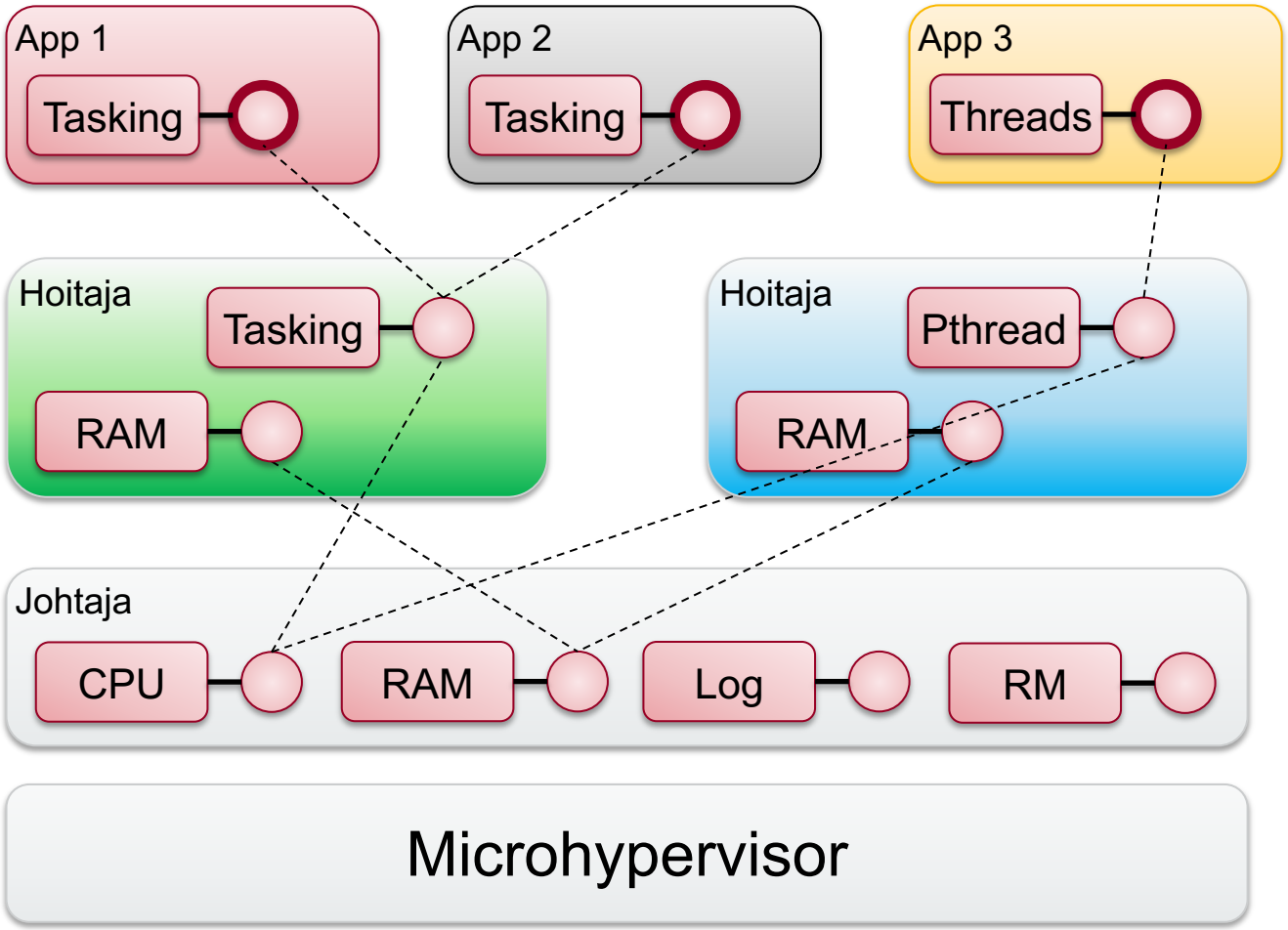
MxKernel Architecture



- Composed of components*
- Components interact via RPC
- And can offer (OS) services
- Parents control access to services and resources

*) Based on the architecture of the Genode OS Framework (www.genode.org)

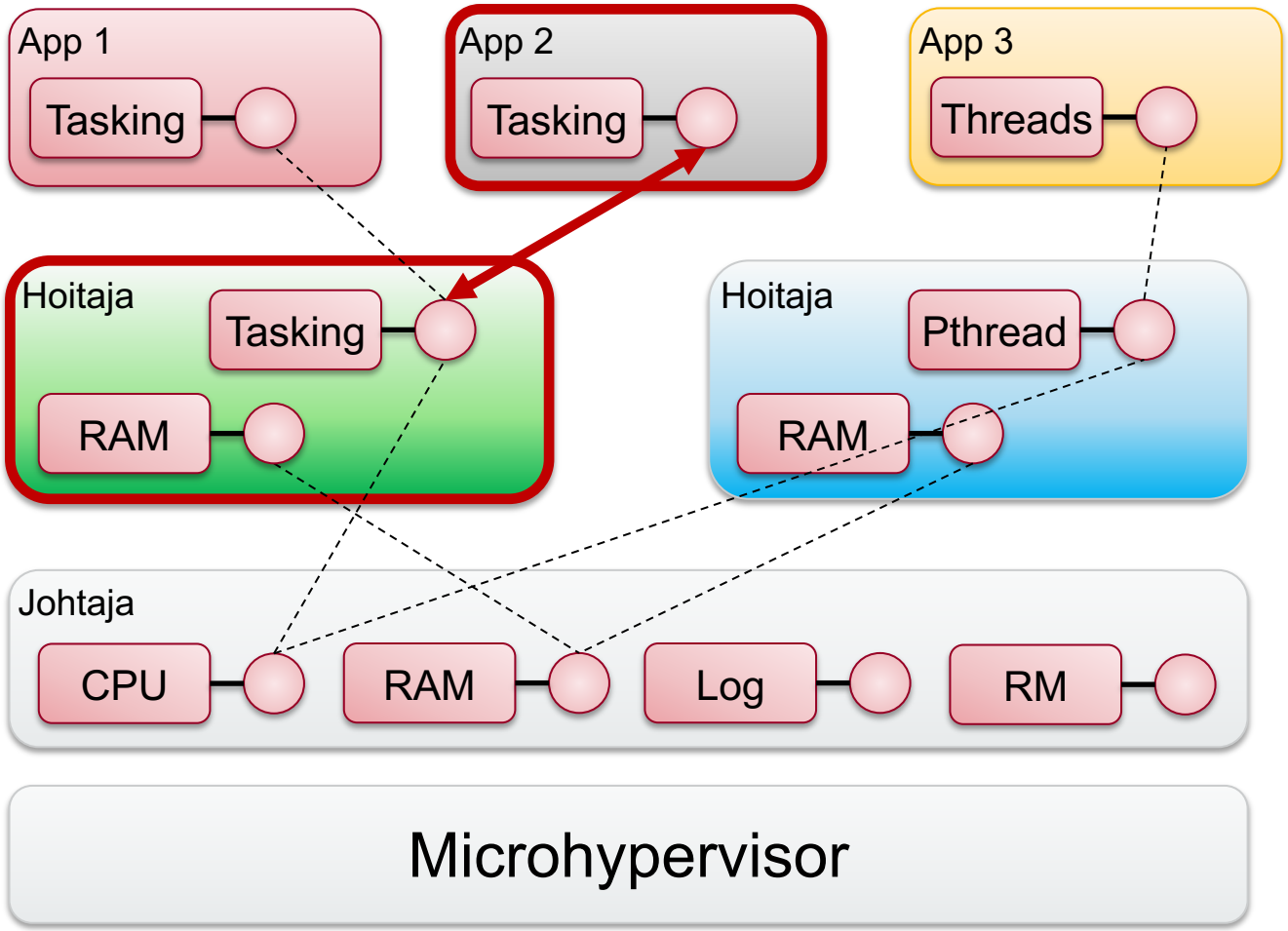
MxKernel Architecture



- Composed of components*
- Components interact via RPC
- And can offer (OS) services
- Parents control access to services and resources
- Using capabilities

*) Based on the architecture of the Genode OS Framework (www.genode.org)

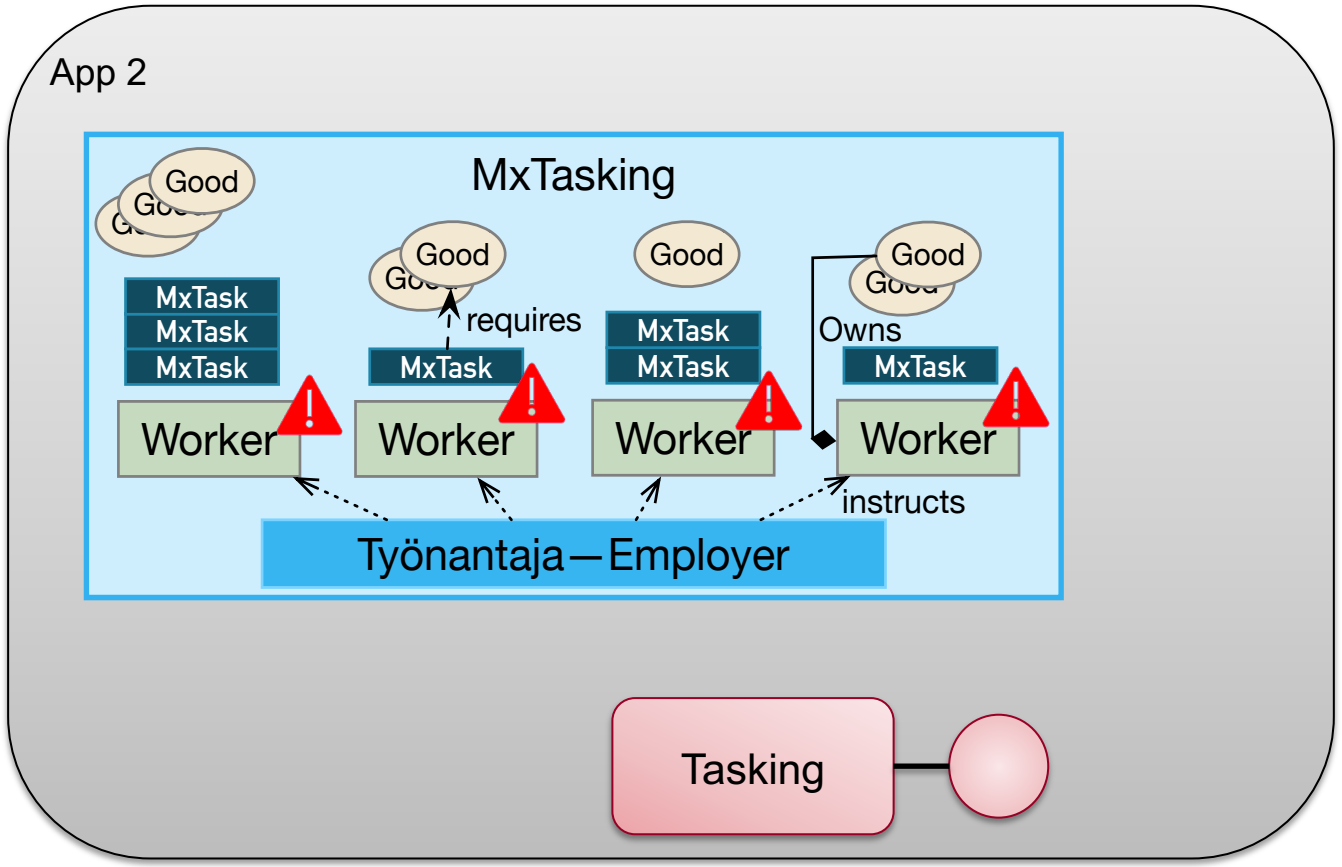
MxKernel Architecture

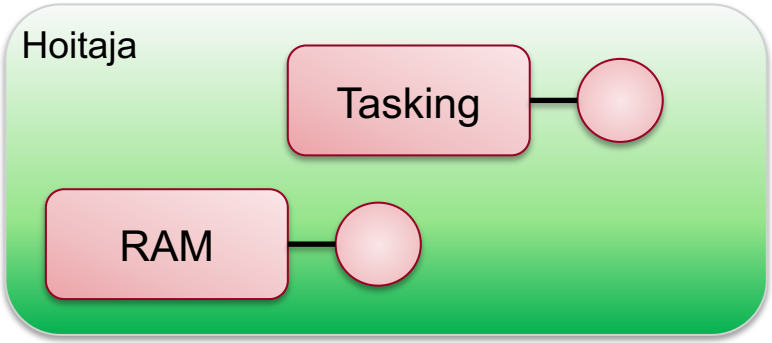


- Composed of components*
- Components interact via RPC
- And can offer (OS) services
- Parents control access to services and resources
- Using capabilities
- Children negotiate resource allocations with parents

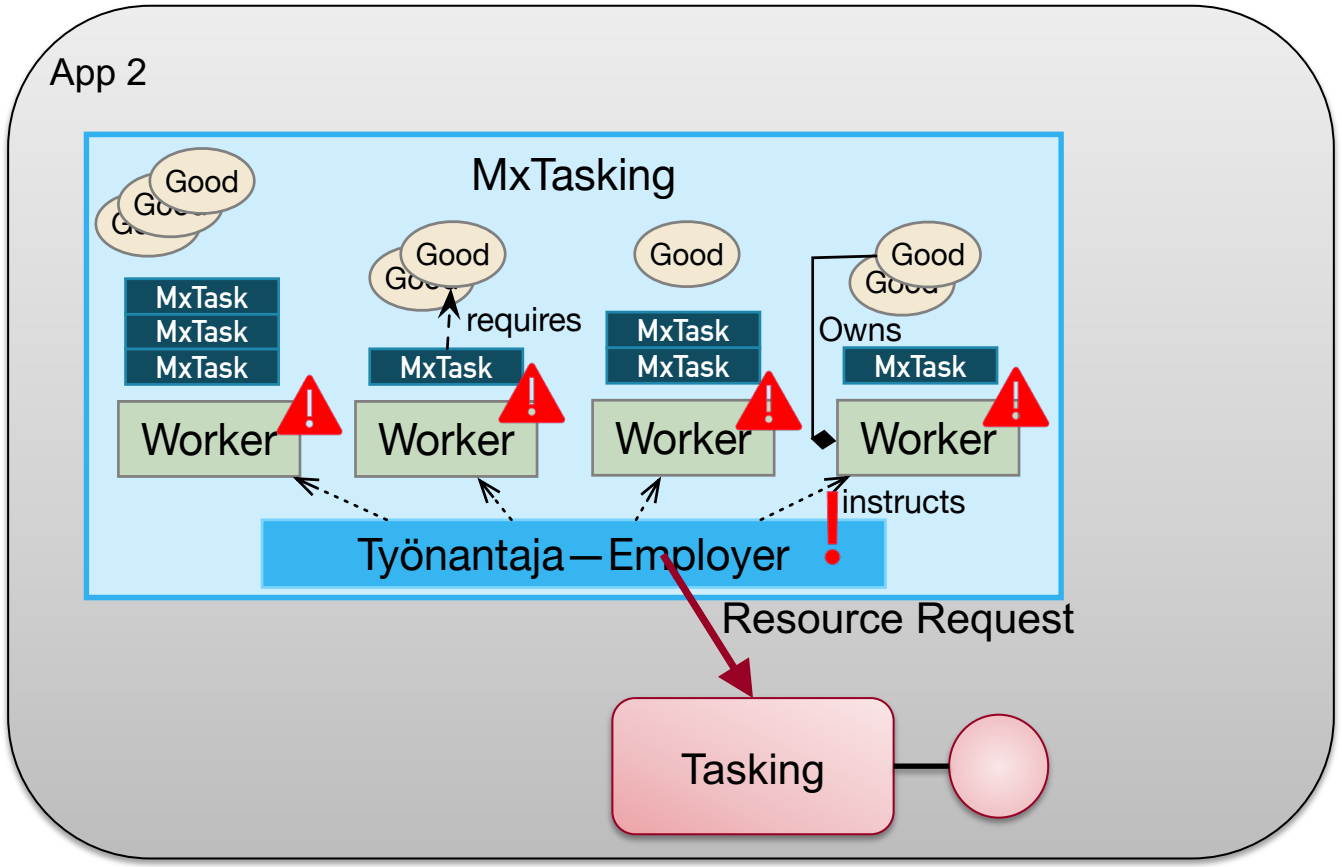
*) Based on the architecture of the Genode OS Framework (www.genode.org)

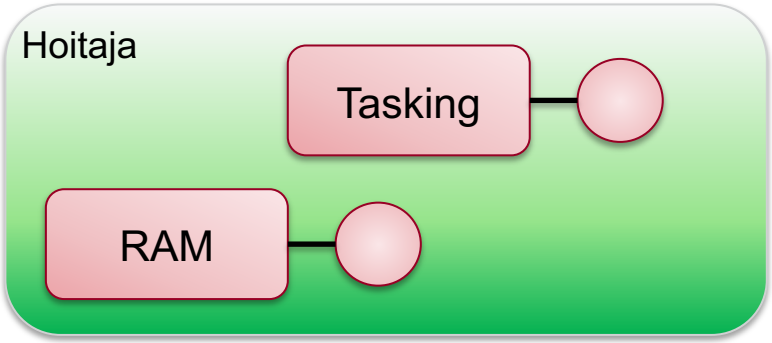
Resource Allocation Negotiation



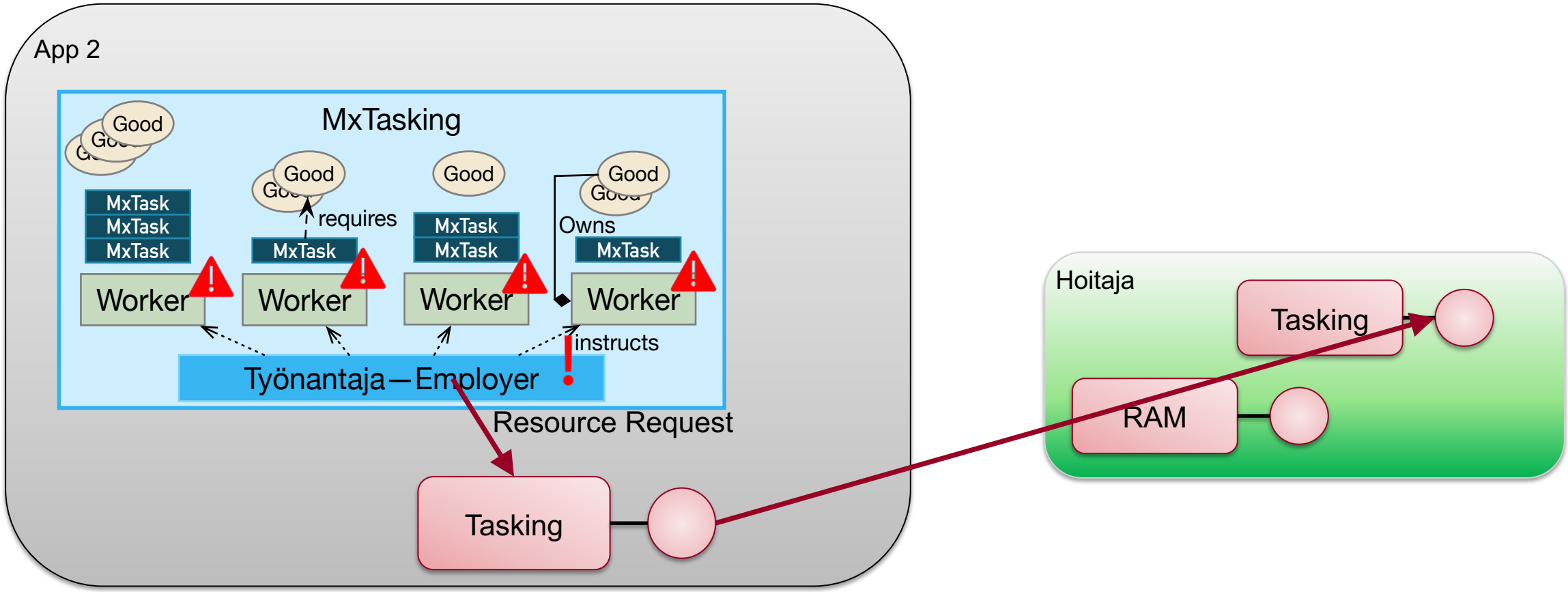


Resource Allocation Negotiation

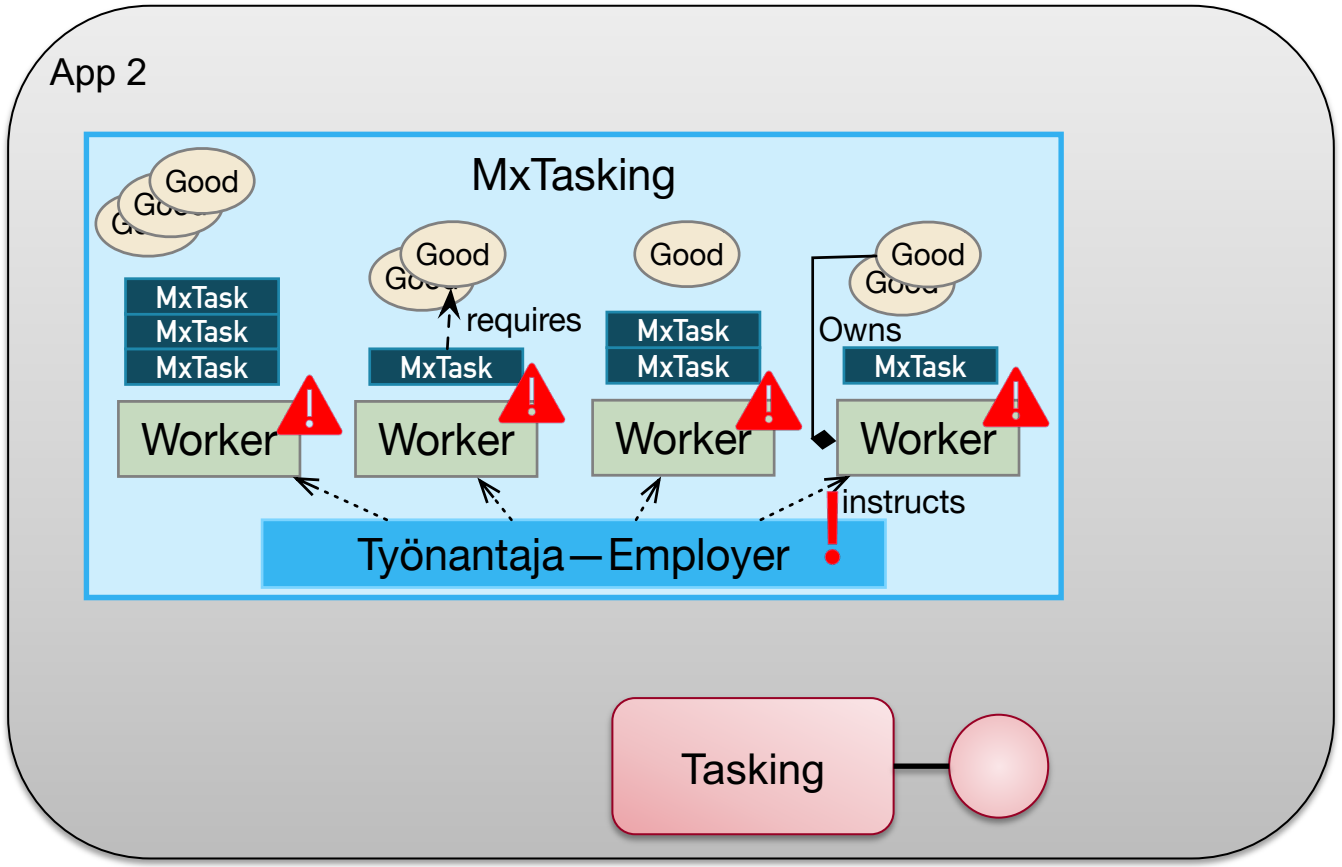


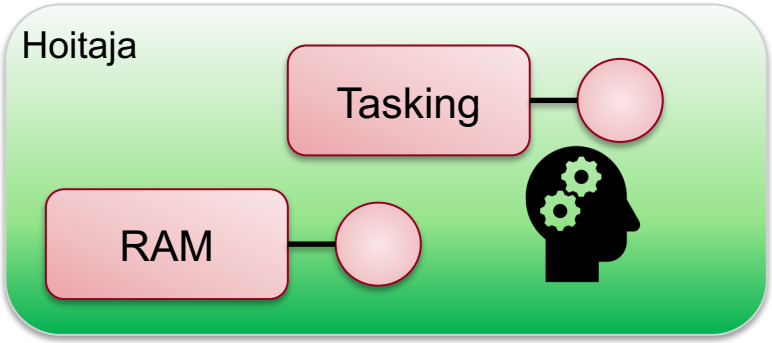


Resource Allocation Negotiation



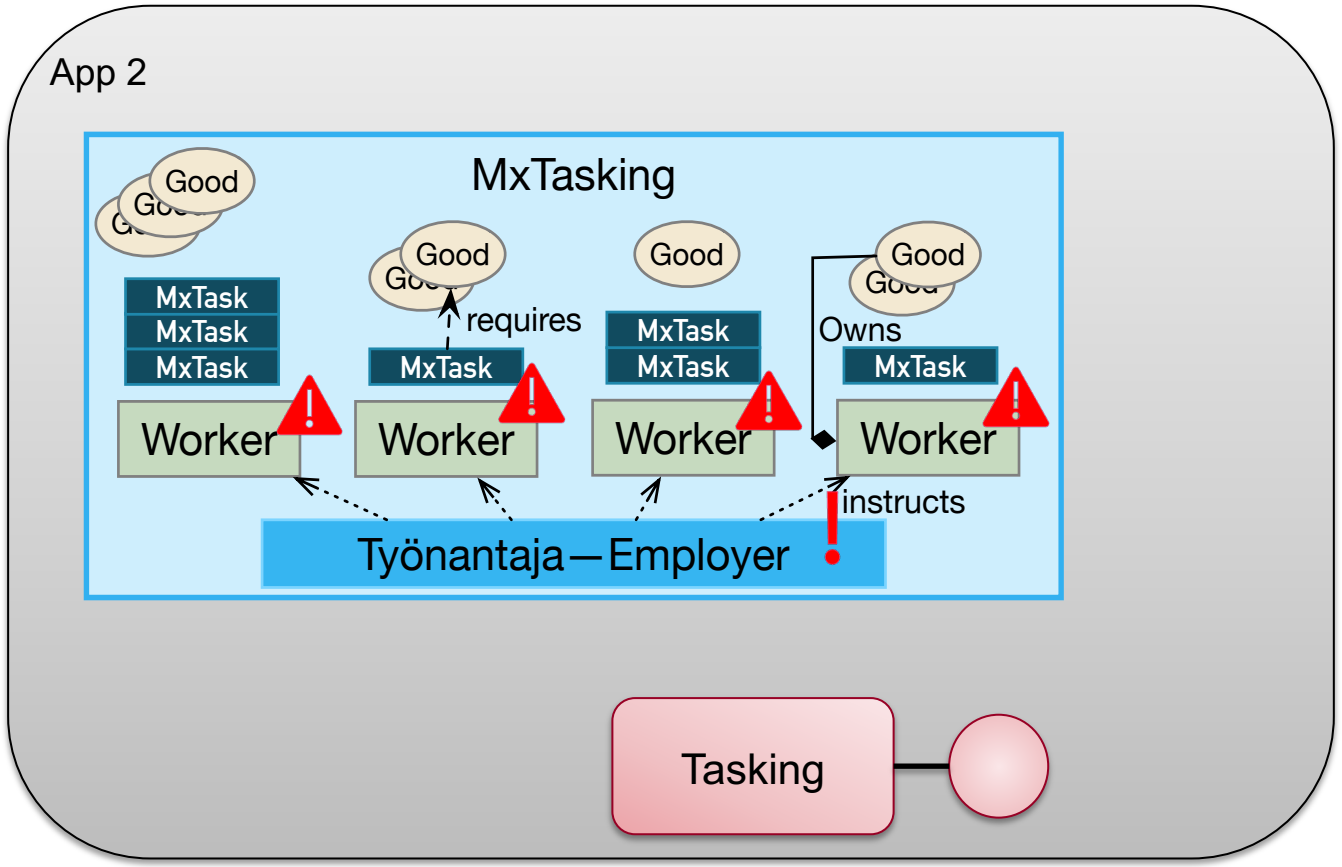
Resource Allocation Negotiation

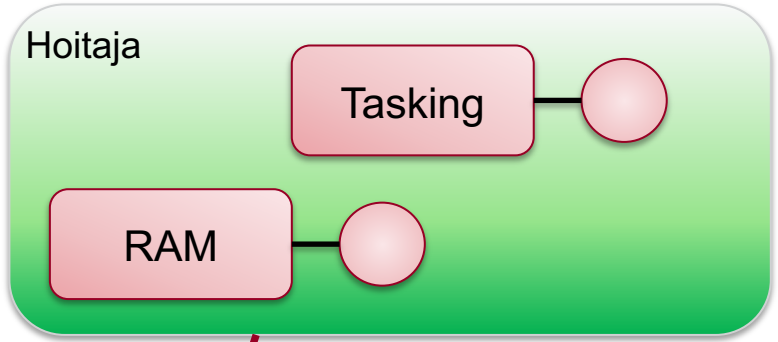




Compute new allocation

Resource Allocation Negotiation

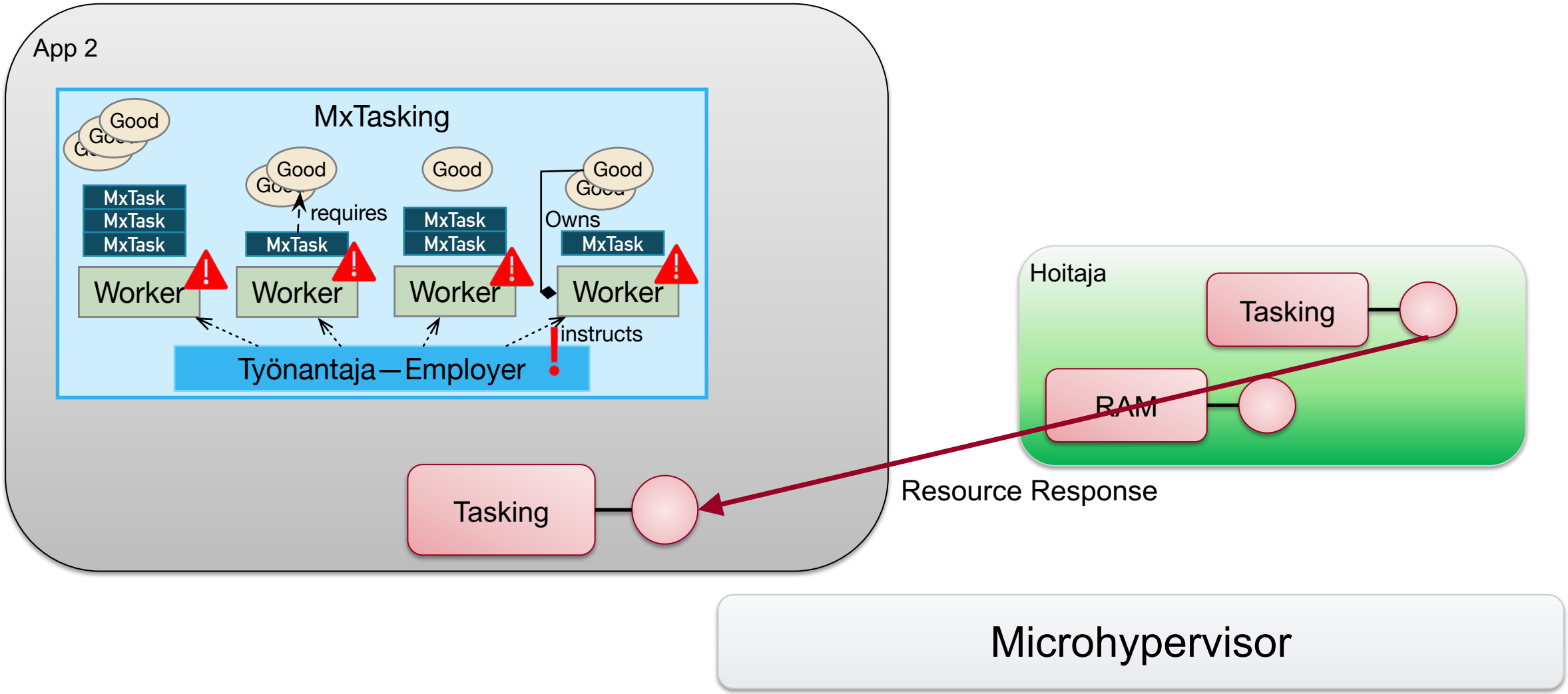




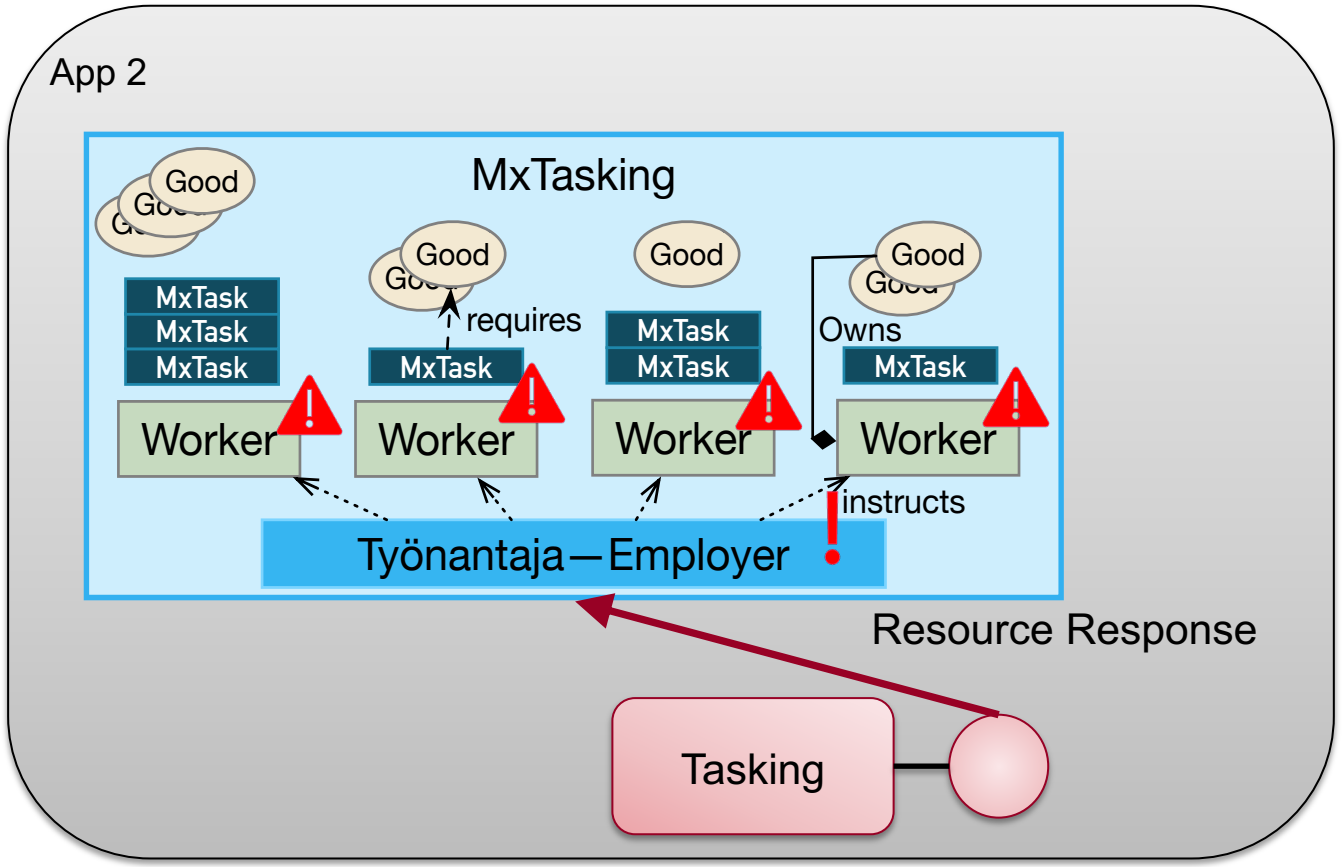
Change allocation

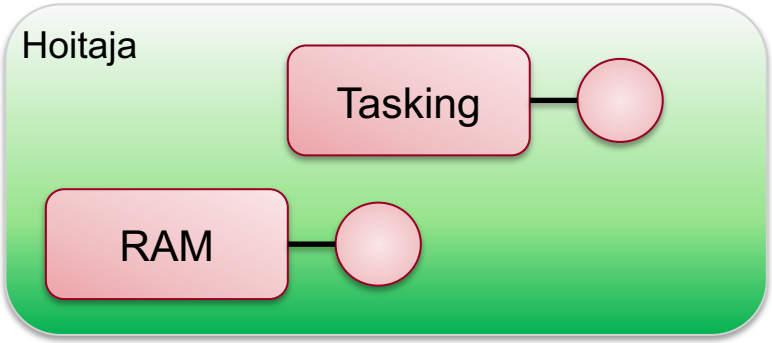
Microhypervisor

Resource Allocation Negotiation



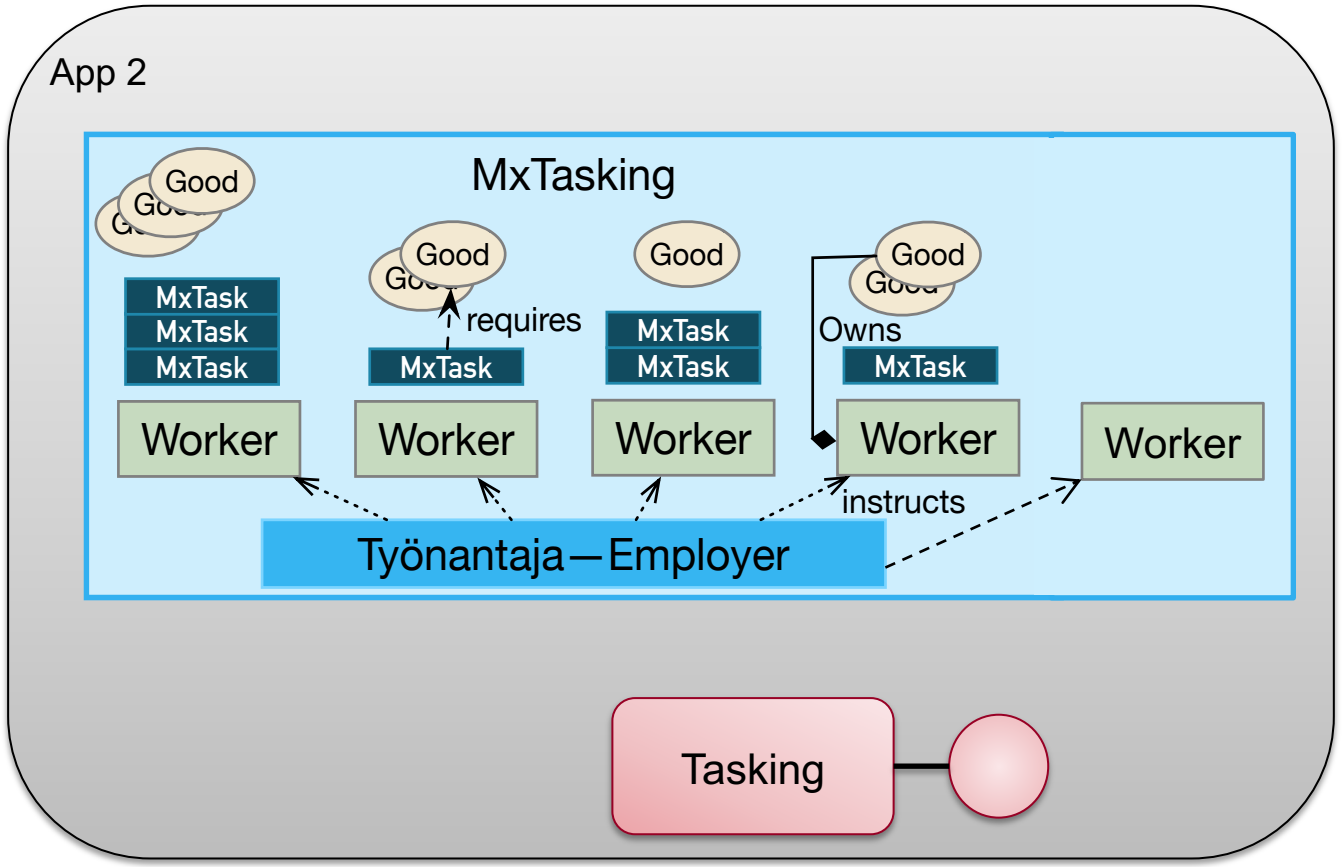
Resource Allocation Negotiation

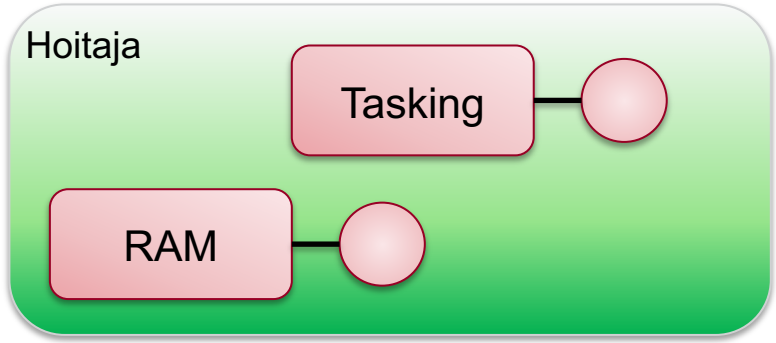




Microhypervisor

Resource Allocation Negotiation





Microhypervisor

Summary

- Resource Management difficult challenge for DC operators
  - Must optimize performance and minimize interference of workloads
  - While increasing resource utilization to save costs

Summary

- Resource Management difficult challenge for DC operators
  - Must optimize performance and minimize interference of workloads
  - While increasing resource utilization to save costs
- Traditional OSes lack interface to communicate resource demands of workloads accurately
  - => Often, over- and under-provisioning of resources leading to waste and overloads

Summary

- Resource Management difficult challenge for DC operators
  - Must optimize performance and minimize interference of workloads
  - While increasing resource utilization to save costs
- Traditional OSes lack interface to communicate resource demands of workloads accurately
  - => Often, over- and under-provisioning of resources leading to waste and overloads
- The MxKernel architecture provides a remedy to these challenges
  - With dynamic resource containers for workloads and their tasks
  - MxTasks as verbose control flow abstraction
  - Component-based architecture allowing for tailored resouce management within workloads

Thank you for your attention

Prototypical implementation of the MxKernel Architecture

- github.com/mmueller41/genode
- github.com/mmueller41/NOVA

EalánOS

- github.com/jmuehlig/mxtasking
- dbis.cs.tu-dortmund.de

MxTasking

Get in touch

Embedded Software Systems

- ess.cs.uos.de
- www.uni-osnabrueck.de

SPP2037 SCALABLE DATA MANAGEMENT FOR FUTURE HARDWARE

- mxkernel.org
- dfg-spp2037.org

References (1)

[1] Barroso, L.A., Hölzle, U. and Ranganathan, P. 2019. *The Datacenter as a Computer: Designing Warehouse-Scale Machines*. Springer International Publishing.

[2] Baumann, A., Appavoo, J., Krieger, O. and Roscoe, T. 2019. A fork() in the road. *Proceedings of the Workshop on Hot Topics in Operating Systems* (New York, NY, USA, May 2019), 14–22.

[3] Bruno, J., Brustoloni, J., Gabber, E., Silberschatz, A. and Small, C. Pebble: A Component-Based Operating System for Embedded Applications.

[4] Colmenares, J.A., Eads, G., Hofmeyr, S., Bird, S., Moretó, M., Chou, D., Gluzman, B., Roman, E., Bartolini, D.B., Mor, N., and others 2013. Tessellation: refactoring the OS around explicit resource containers with continuous adaptation. *Proceedings of the 50th Annual Design Automation Conference* (2013), 76.

[5] Engler, D.R. The Exokernel Operating System Architecture. 120.

[6] Fried, J., Ruan, Z., Ousterhout, A. and Belay, A. 2020. Caladan: Mitigating Interference at Microsecond Timescales. (2020), 281–297.

[7] Iorgulescu, C., Azimi, R., Kwon, Y., Elnikety, S., Syamala, M., Narasayya, V., Herodotou, H., Tomita, P., Chen, A., Zhang, J. and Wang, J. 2018. PerfIso: performance isolation for commercial latency-sensitive services. *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference* (USA, Jul. 2018), 519–531.

References (2)

[8] Kim, H.-J., Lee, Y.-S. and Kim, J.-S. 2016. NVMeDirect: A User-space I/O Framework for Application-specific Optimization on NVMe SSDs. *8th USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage 2016, Denver, CO, USA, June 20-21, 2016.* (2016).

[9] Li, J., Sharma, N.Kr., Ports, D.R.K. and Gribble, S.D. 2014. Tales of the Tail: Hardware, OS, and Application-level Sources of Tail Latency. *Proceedings of the ACM Symposium on Cloud Computing* (New York, NY, USA, Nov. 2014), 1–14.

[10] Marty, M. et al. 2019. Snap: a microkernel approach to host networking. *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (New York, NY, USA, Oct. 2019), 399–413.

[11] McClure, S., Ousterhout, A., Shenker, S. and Ratnasamy, S. 2022. Efficient Scheduling Policies for {Microsecond-Scale} Tasks. (2022), 1–18.

[12] Olivier, P., Barbalace, A. and Ravindran, B. 2020. The Case for Intra-Unikernel Isolation. (Mar. 2020).

[13] Rizzo, L. netmap: a novel framework for fast packet I/O.

[14] Steinberg, U. and Kauer, B. 2010. NOVA: a microhypervisor-based secure virtualization architecture. *Proceedings of the 5th European conference on Computer systems - EuroSys '10* (Paris, France, 2010), 209.

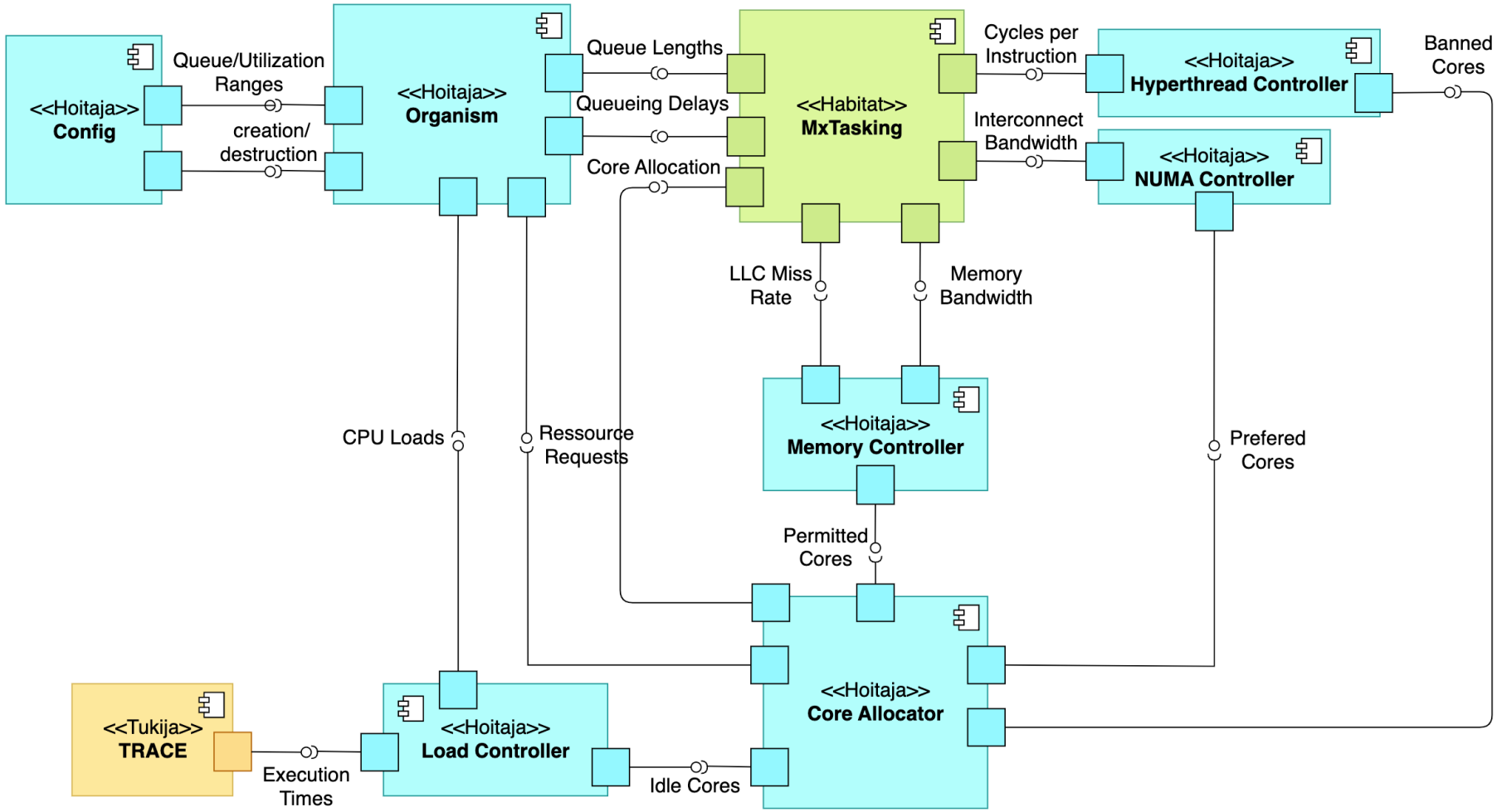
References (3)

[15] Talbot, J., Pikula, P., Sweetmore, C., Rowe, S., Hindy, H., Tachtatzis, C., Atkinson, R. and Bellekens, X. 2020. A Security Perspective on Unikernels. *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)* (Jun. 2020), 1–7.

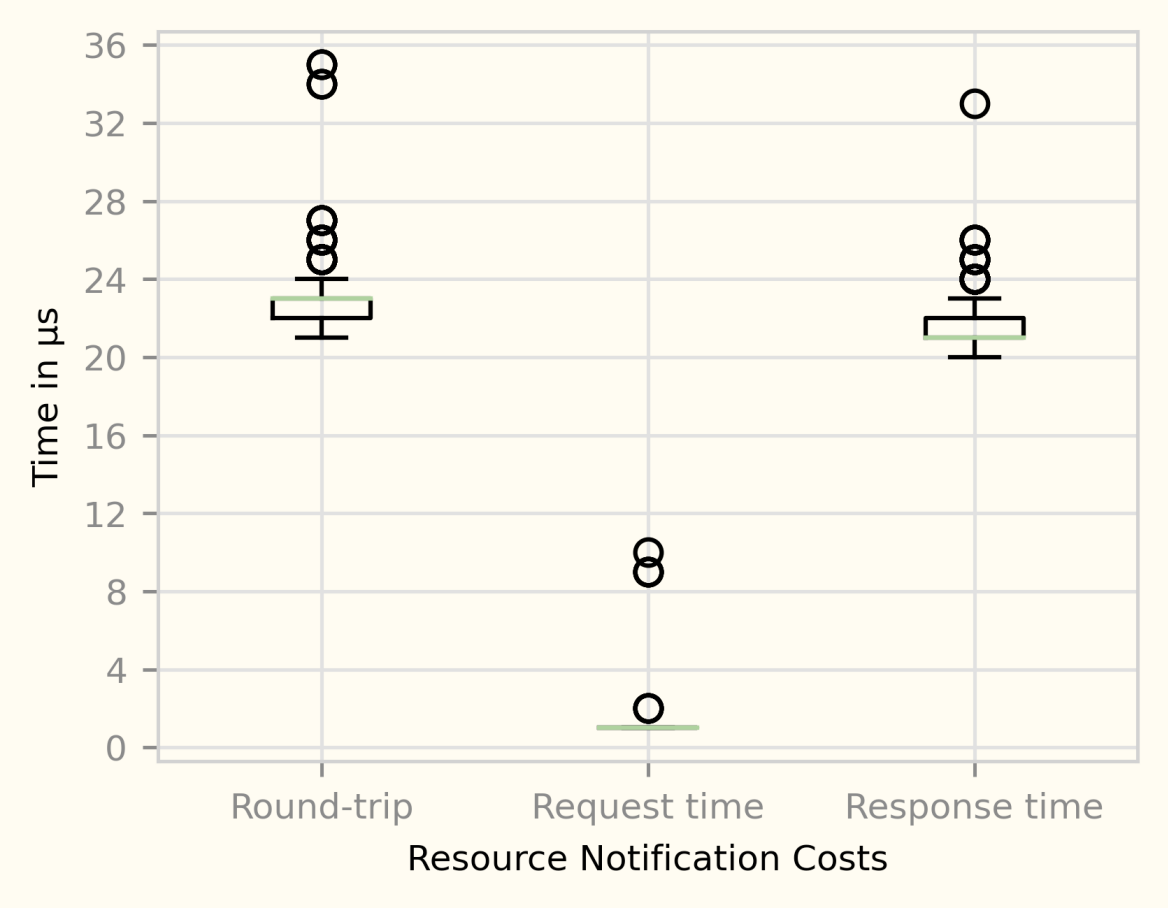
[16] Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E. and Wilkes, J. 2015. Large-scale cluster management at Google with Borg. *Proceedings of the Tenth European Conference on Computer Systems* (New York, NY, USA, Apr. 2015), 1–17.

[17] Zhang, X., Tune, E., Hagmann, R., Jnagal, R., Gokhale, V. and Wilkes, J. 2013. CPI2: CPU performance isolation for shared compute clusters. *Proceedings of the 8th ACM European Conference on Computer Systems* (New York, NY, USA, Apr. 2013), 379–391.

Hoitaja's Architecture



Preliminary Evaluation: Notification Mechanism



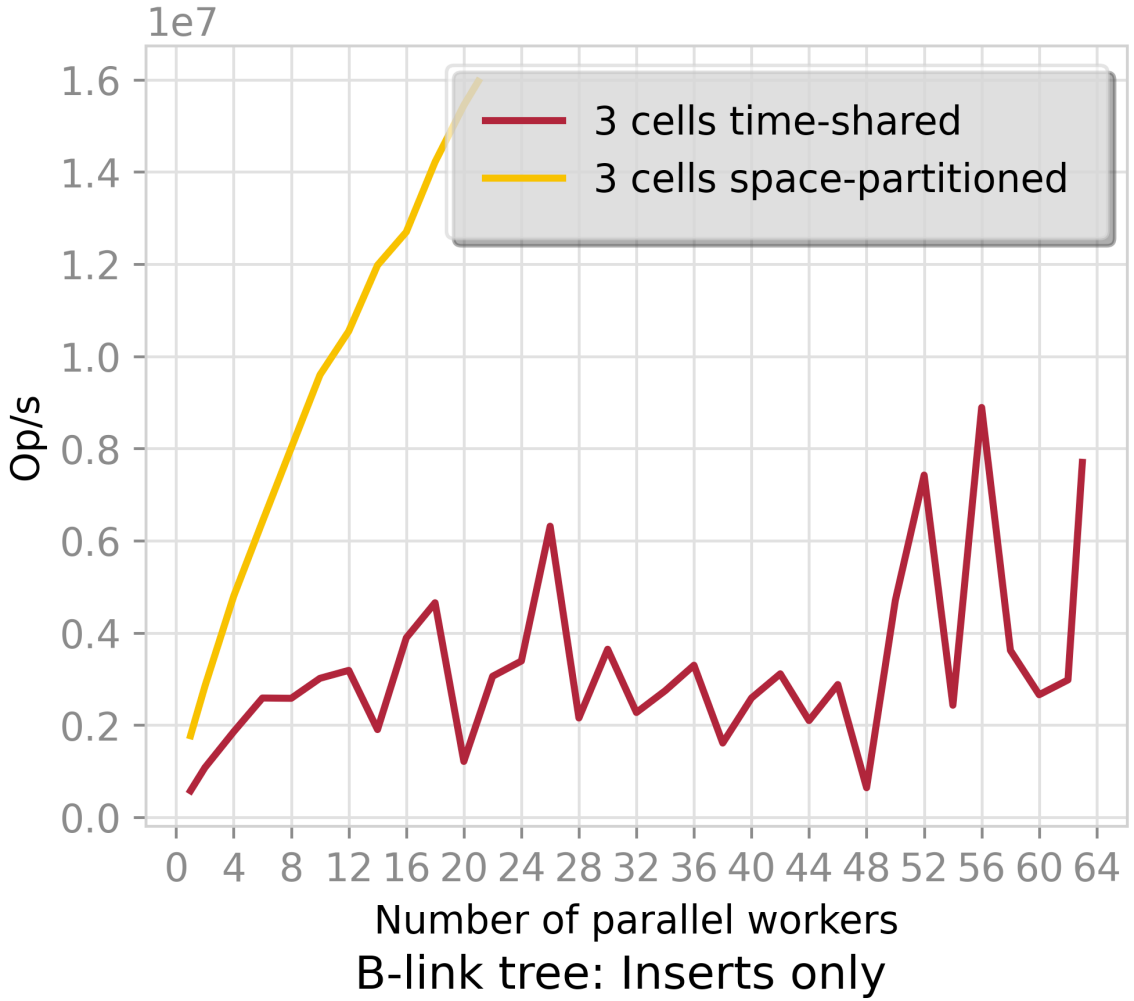
- Hoitaja sends notification to cell, cells sends acknowledege when notification received
- 5000 notifications total
- 10ms interval between notifications

Example: B-link tree benchmark

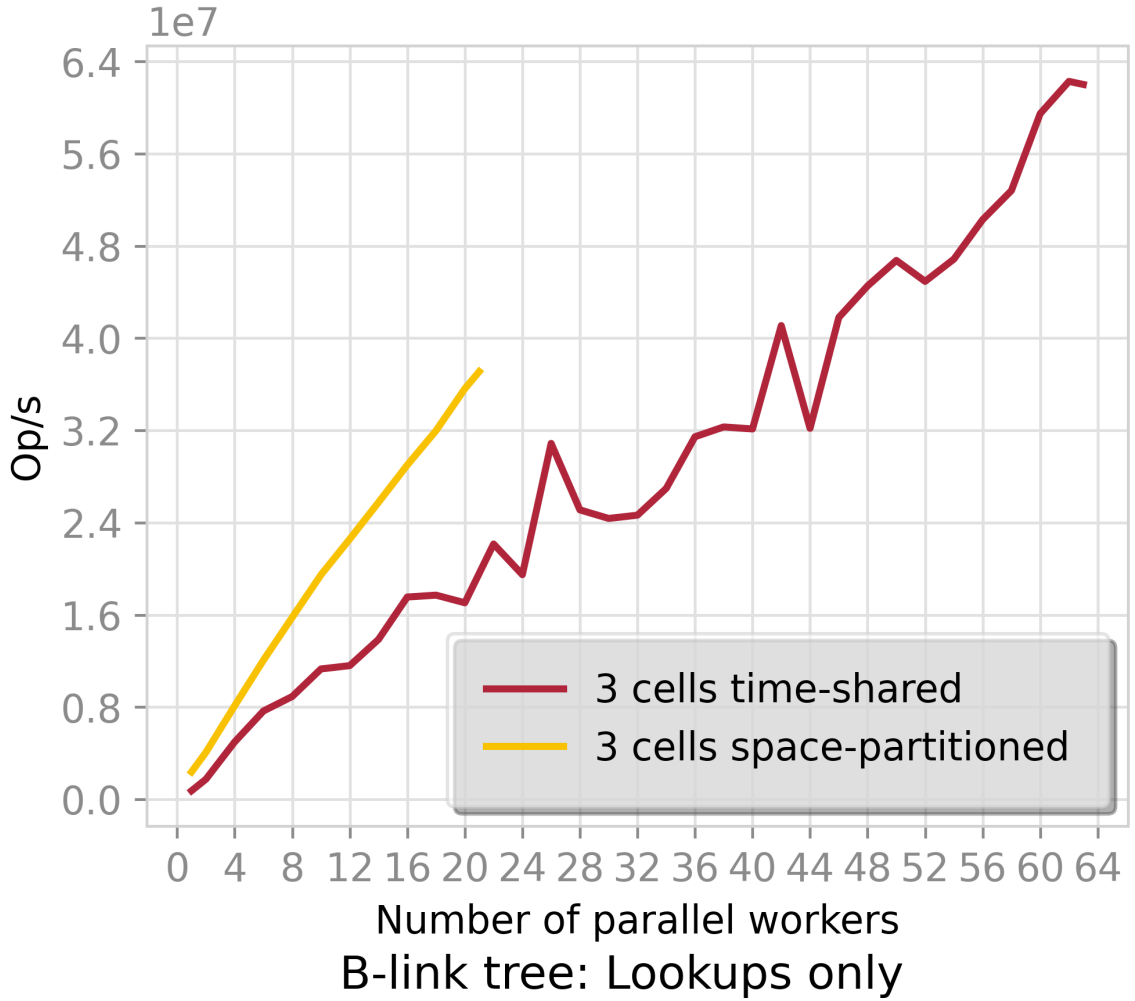
- Investigate how **throughput** of benchmark is **affected** when we run **multiple instances**
  - On the **same** set of CPU cores
  - On a **disjunct** set of CPU cores
- **Which** scenario will yield the **higher** throughput at the respective **maximum** of cores?

- B-link trees are a wide-spread **datastructure** for **indexing** in **database** systems.
- The B-link tree benchmark is based on **YCSB**, a **common benchmark** for **key-value stores**.

Example: Results



B-link tree: Inserts only



B-link tree: Lookups only