SPP2037
SCALABLE DATA MANAGEMENT
FOR FUTURE HARDWARE

MxKernel

A System Software Architecture for Modern Hardware

mxkernel.org

Jan Mühlig
Jens Teubner

Databases and Information Systems

technische universität dortmund

Michael Müller
Olaf Spinczyk

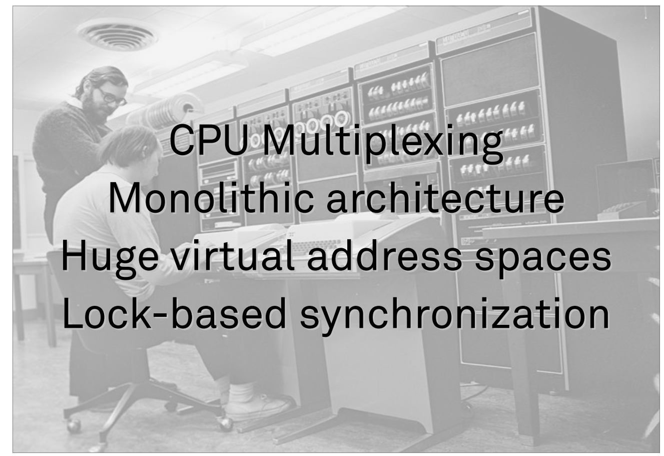
Embedded Software Systems

UNIVERSITÄT OSNABRÜCK

Traditional Operating Systems

... in the context of modern multicore and manycore systems

	Hardware		OS Support
Past	Single CPU User/supervisor mode Uniform physical memory MMU: Virtual memory Global I/O controllers	➔	CPU Multiplexing Monolithic architecture Huge virtual address spaces Lock-based synchronization
Future	Many CPU cores Heterogeneous cores Complex NUMA architecture Non-volatile memory Non-uniform I/O architecture Voltage/frequency islands Aging effects	➔	?



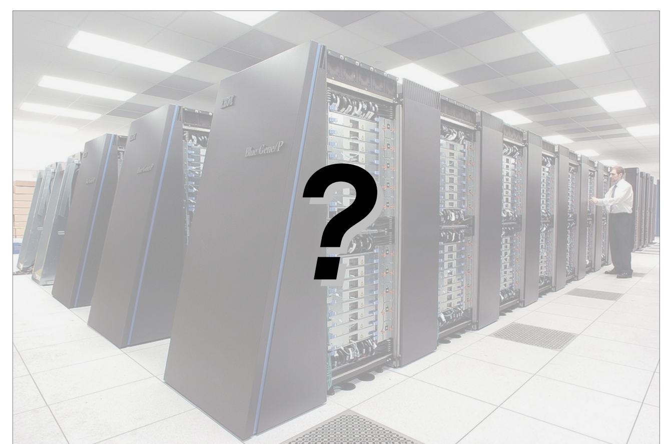
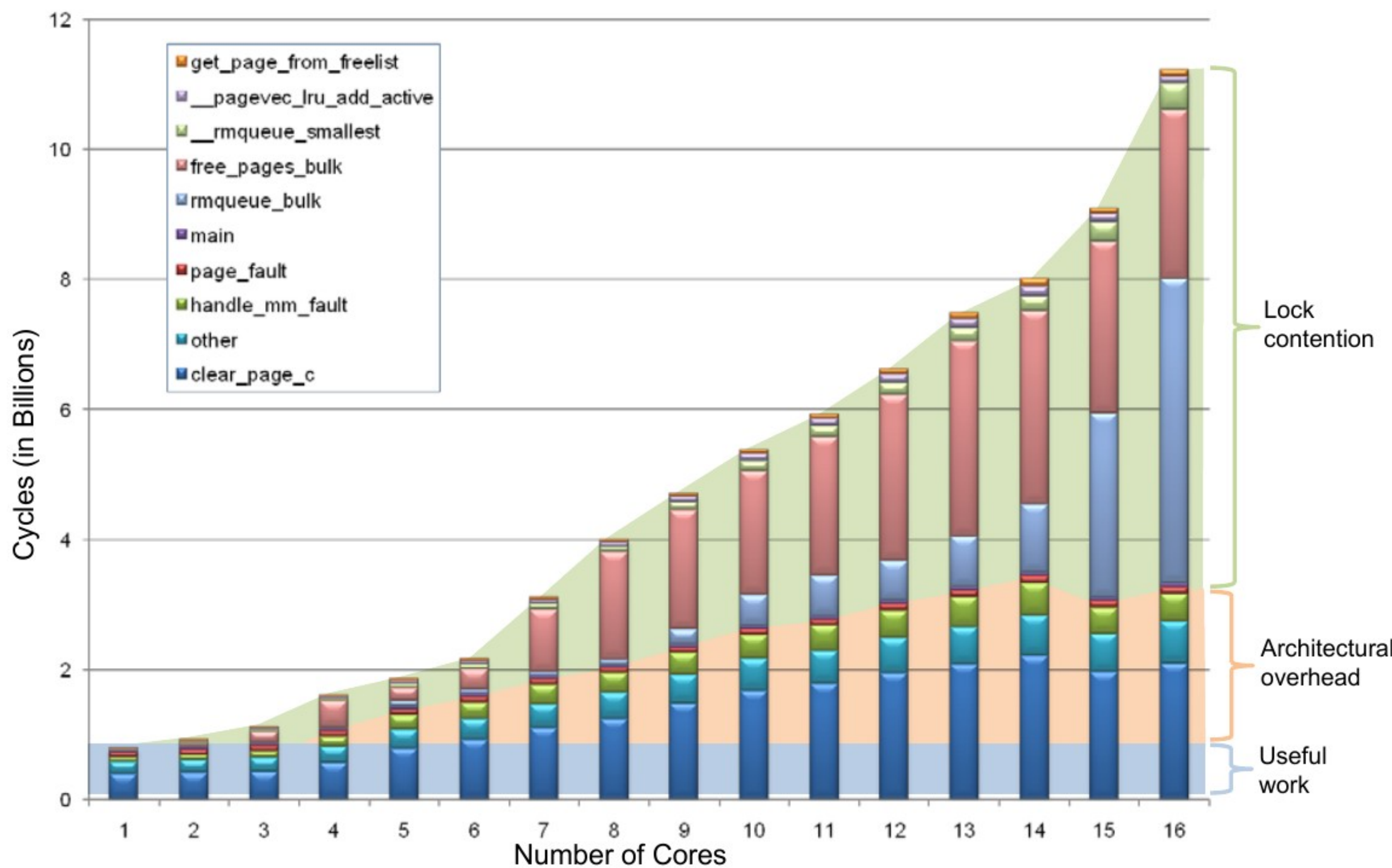


Image sources (aspect ratios slightly modified):
https://en.wikipedia.org/wiki/Unix
https://en.wikipedia.org/wiki/Supercomputer

Traditional Operating Systems

- Example: Linux memory allocation benchmark [1]



Outline

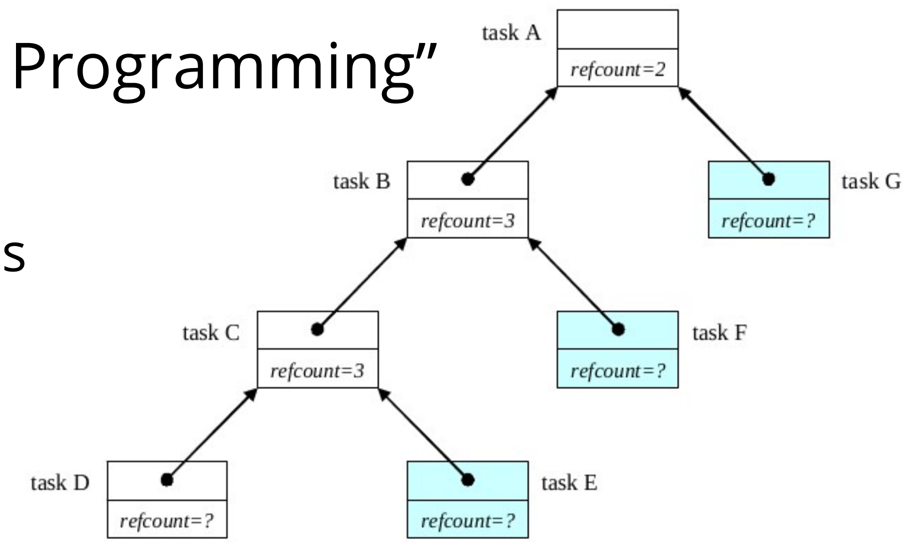
- Manycore Programming
- Manycore OS Research
- MxKernel Architecture
- Preliminary Results
- Conclusions

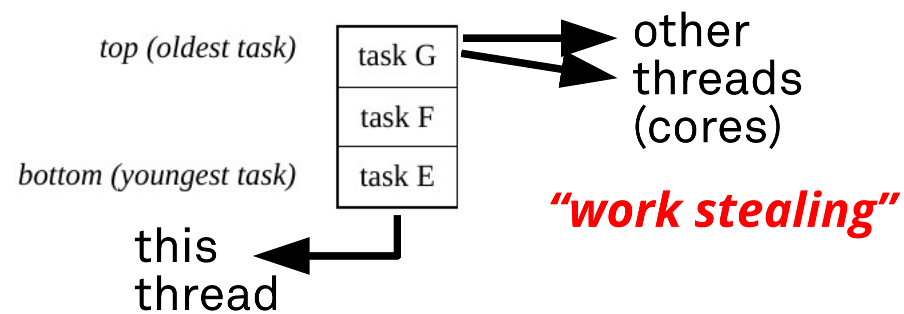
Outline

- **Manycore Programming**
- Manycore OS Research
- MxKernel Architecture
- Preliminary Results
- Conclusions

Manycore Programming: Intel® TBB [2]

- Instead of threads: "***Task***-based Programming"
  - Fine-grained units of work: functions, functors, or C++ lambdas
  - Lightweight: No separate stack, register set, etc.
- Task scheduler
  - Efficiently executes tasks from double ended queues
  - Automatic load balancing
- Problems
  - Inefficient if tasks perform blocking operations
  - Tasks must be synchronized by classic mechanisms → locks





… Programming: HyPer Morsels [3]

- Instead of threads: “***Morsel***-driven query execution”
  - Small DB operator pipelines, JIT compiled
  - Small chunks of input data
  - Input and output are NUMA-local
- Scheduler (in user space)
  - Fixed number of pinned threads
  - Load balancing by work stealing
  - Excellent scalability:
    30x performance on 32 core system
- Problems
  - Special purpose solution; Does not re-use OS features

The HyPer DBMS

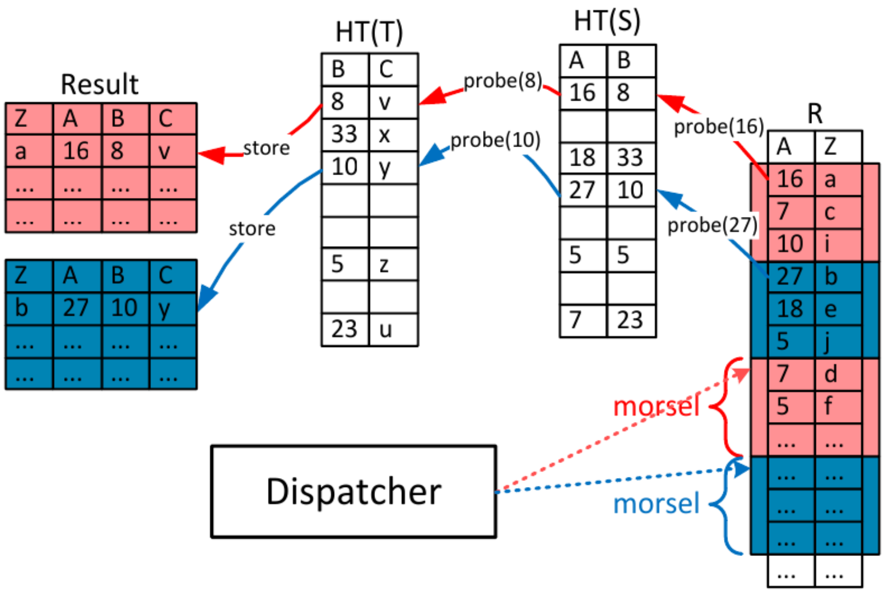


Figure 1: Idea of morsel-driven parallelism: $R \bowtie_A S \bowtie_B T$

Outline

- Manycore Programming
- **Manycore OS Research**
- MxKernel Architecture
- Preliminary Results
- Conclusions

Manycore OS: State-of-the-Art

- Barrelfish [4]
  - Multikernel architecture

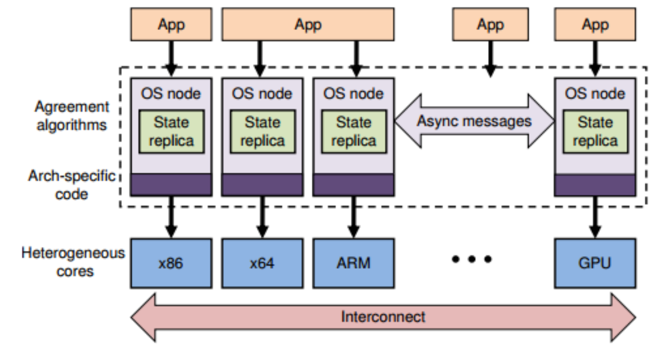
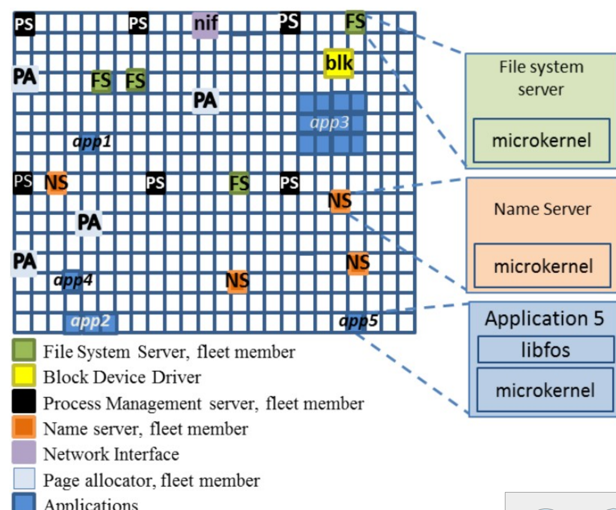
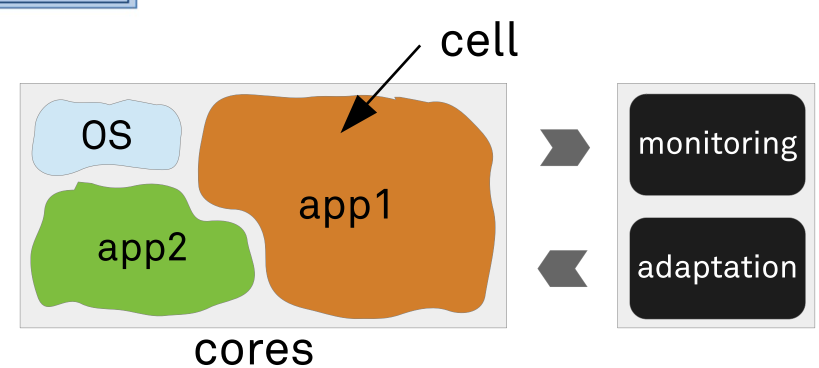


Figure 1: The multikernel model.

- fos [1]
  - Microkernel
  - Server threads (or "fleets")



- Tesselation [5]
  - Cell concept
  - Gang scheduling

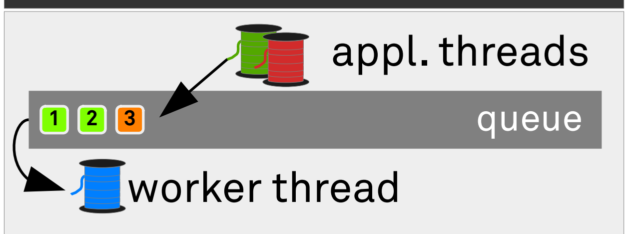


➔ Still using threads. Optimizations done by app. programmer.

… OS: Apple's GCD Kernel Support

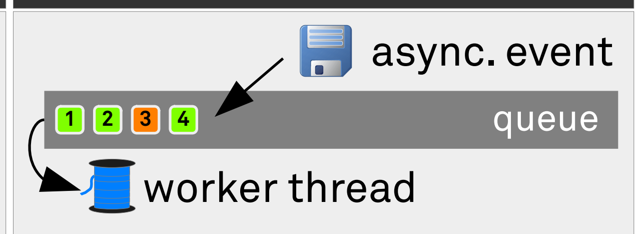
- ***"Grand Central Dispatch"***
  - Resembles TBB, but MacOS provides kernel-level support

Serial dispatch queue



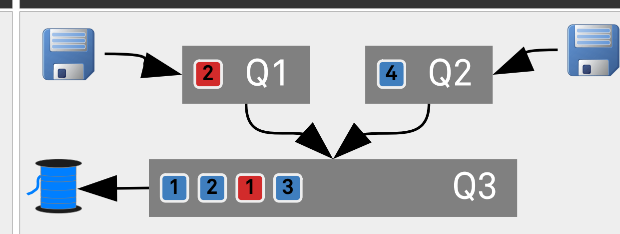
- Implicit serialization
- Worker thread creation on demand

Dispatch source



- Seamless I/O integration
- Automatic triggering of success/ failure handler

Queue hierarchy



- Restricted number of threads
- Guaranteed partial order

- Problems
  - Context switches for simple queue operations
    - Necessary to avoid priority inversion (task vs. thread priorities)
  - No clean layer structure in the kernel

Outline

- Manycore Programming
- Manycore OS Research
- **MxKernel Architecture**
- Preliminary Results
- Conclusions

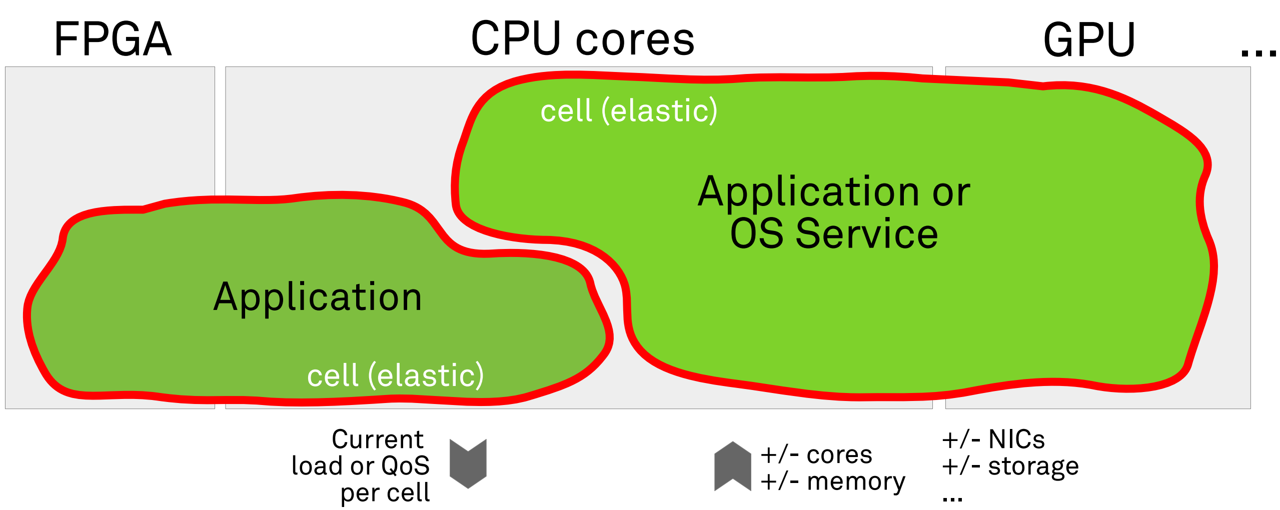
The MxKernel: Design Goals

1) Fair and optimized **partitioning** of heterogeneous resources between multiple applications and OS components
2) Handle **global concerns**, such as power management, in a central component
3) Topology-aware placement of control flows and data to **optimize performance**
4) Global as well as application-specific (tailored) **OS services** that can benefit from accelerators and many-core CPUs

The MxKernel: Key Features (1)

Goal 1:
- Fair and optimized partitioning of heterogeneous resources between multiple applications and OS components

Solution: Elastic cells
- Provide *spatial* isolation of applications and global OS services (based on priorities)
- Optimized mapping (e.g. NUMA-aware)
- Span over CPU cores, but also FPGA and GPU resources, etc.



The MxKernel: Key Features (2)

Goal 2:
- Handle global concerns, such as power management, in a central component

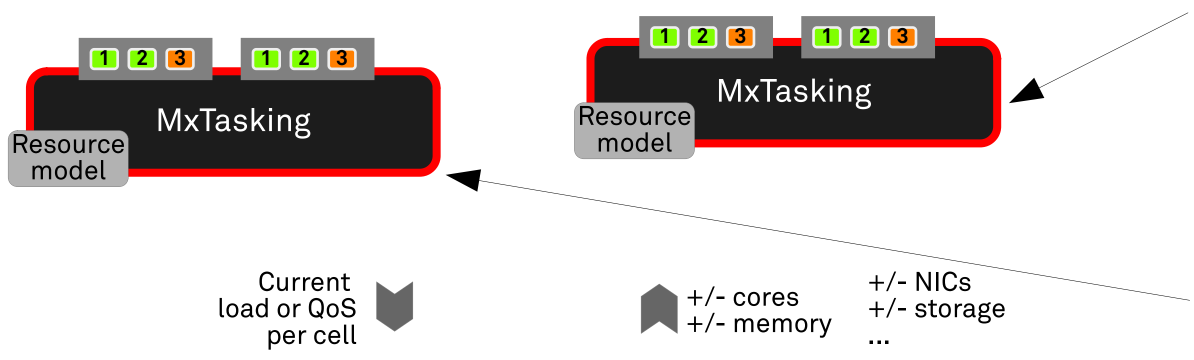
Solution: Global resource management
- Provisioning, monitoring and adaptation of cells (cores, memory, clock speed, etc.)
- Enforcement of system-wide policies (low-power, anti aging, etc.)

MxVisor
- isolation of cells
- priorities of applications
- optimized app-to-core mapping (NUMA-aware)
- power management
- anti-aging
- fault tolerance, e.g. app replication, handling damaged components)

The MxKernel: Key Features (3)

Goal 3:
- Topology-aware placement of control flows and data to optimize performance

Solution: Task-based programming model
- Simplifies development of parallel programs
- Unified programming model for heterogeneous compute units
- Helps to avoid lock-based synchronization
- Supports automatic load balancing, optimized task placement, and cell elasticity based on (physical) resource model

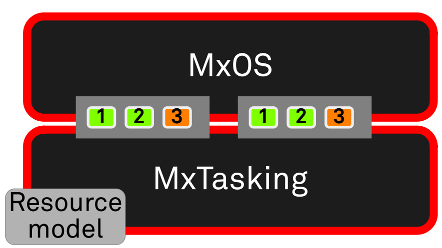


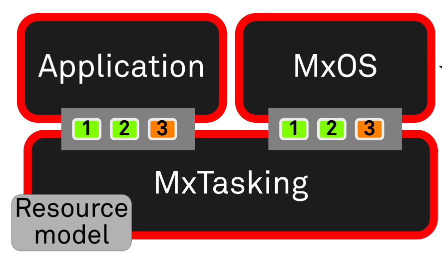
MxTasking
- task-based API
- handles adaptations
- topology-aware optimizations (e.g. NUMA)
- fine-grained application-specific mapping decisions
- exploit heterogeneous computing resources
- multiple specialized instances possible

The MxKernel: Key Features (4)

Goal 4:
- Global as well as application-specific (tailored) OS services that can benefit from accelerators and many-core CPUs

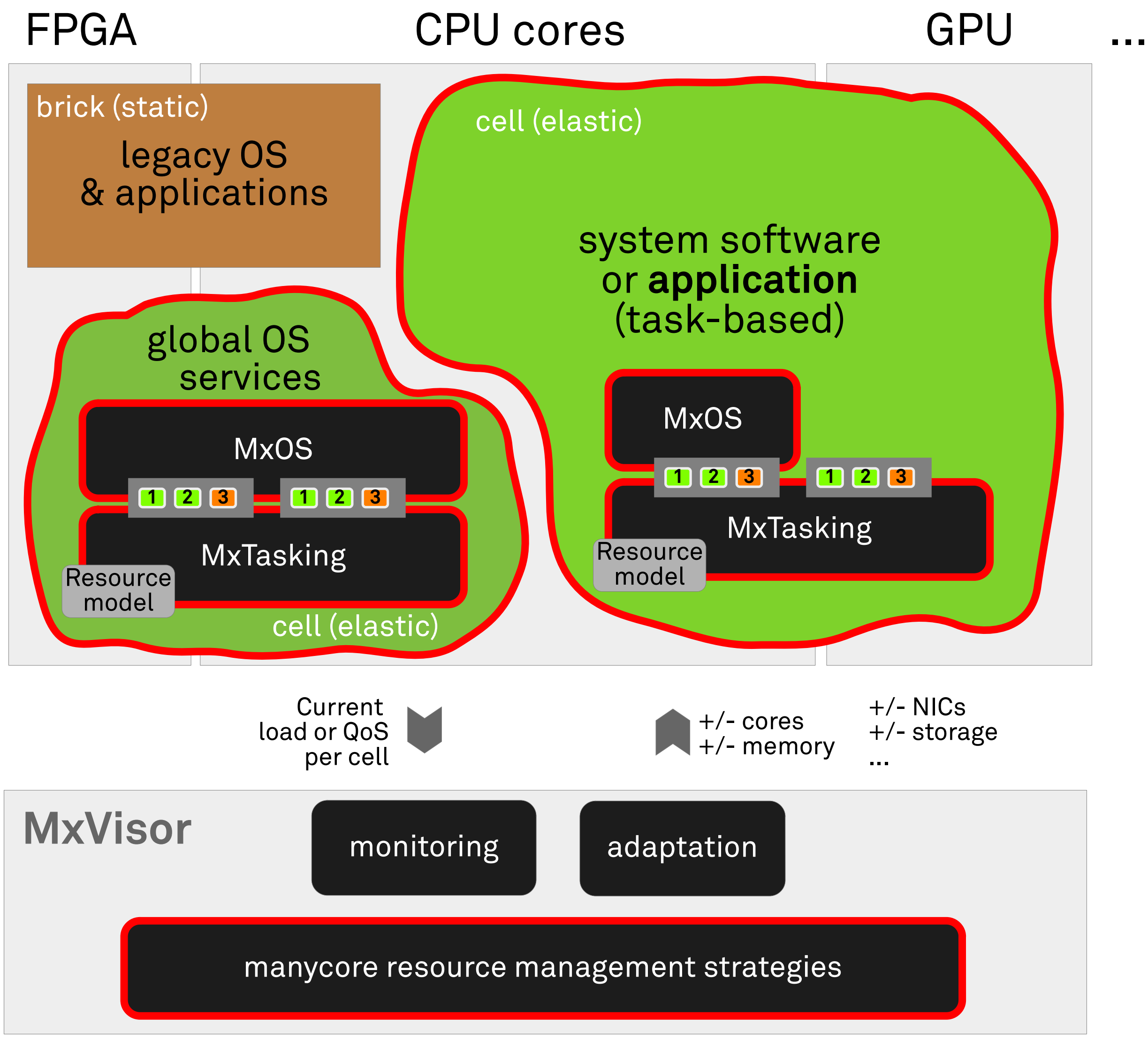
Solution: Global/local OS services built <u>on top</u> of MxTasking
- Scalability provided automatically by MxTasking/MxVisor
- Localized state
- Thread model can be provided for legacy applications
- Family-based design for code reuse





MxOS
- device drivers
- OS services, e.g. network protocols, filesystems, etc.

The MxKernel: Architecture

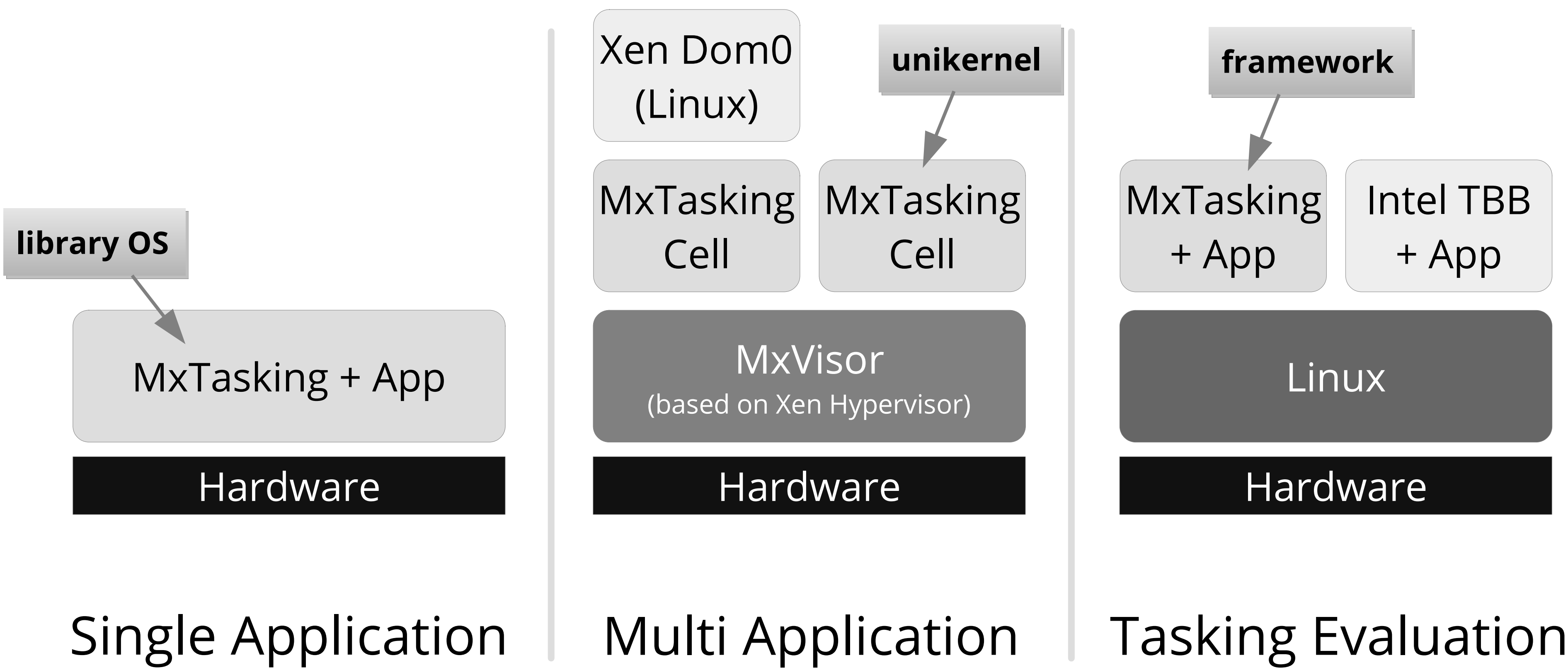


Outline

- Manycore Programming
- Manycore OS Research
- MxKernel Architecture
- **Preliminary Results**
- Conclusions

Prototype Implementation

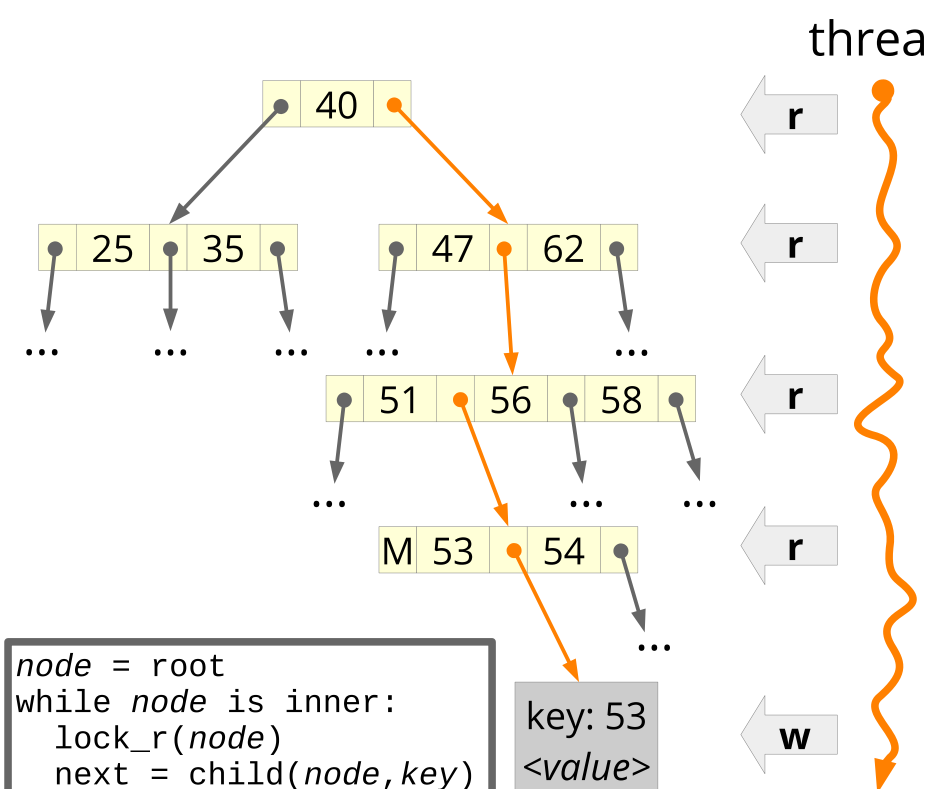
- … comes in three flavors (all x86/64, all experimental):



(link)

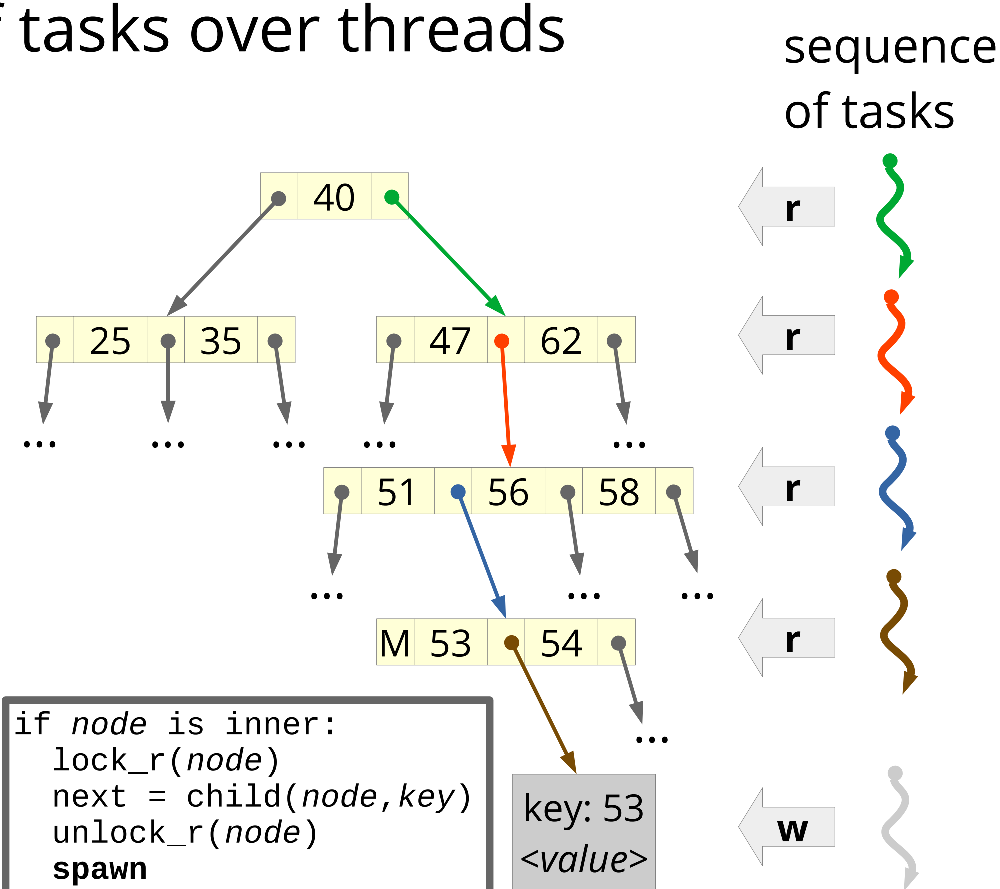
Favorite Benchmark: B-Tree Operations

- Demonstrates advantages of tasks over threads




```
node = root
while node is inner:
  lock_r(node)
  next = child(node,key)
  unlock_r(node)
  node = next
lock_w(node)
insert(node,key,value)
unlock_w(node)
```

“long” thread with unforeseeable access pattern

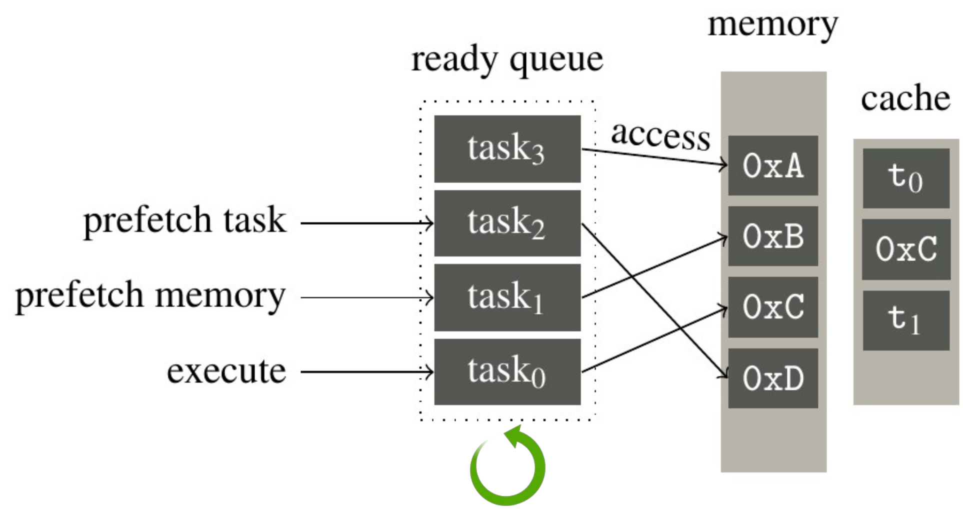


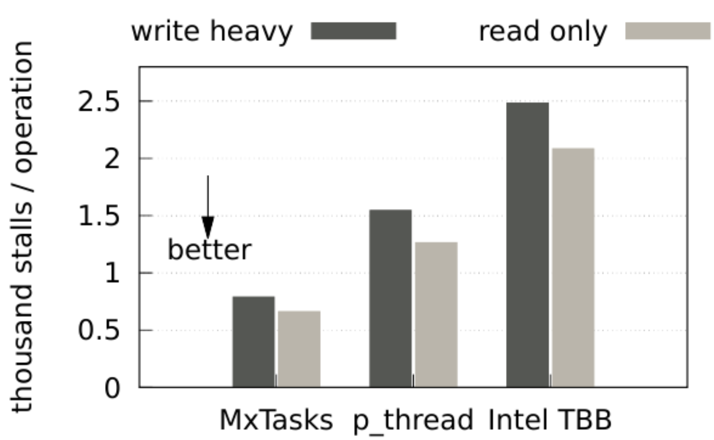

```
if node is inner:
  lock_r(node)
  next = child(node,key)
  unlock_r(node)
  spawn
    task(next,key,value)
else:
  lock_w(node)
  insert(node,key,value)
  unlock_w(node)
```

“handy” tasks with known access and locking requirements

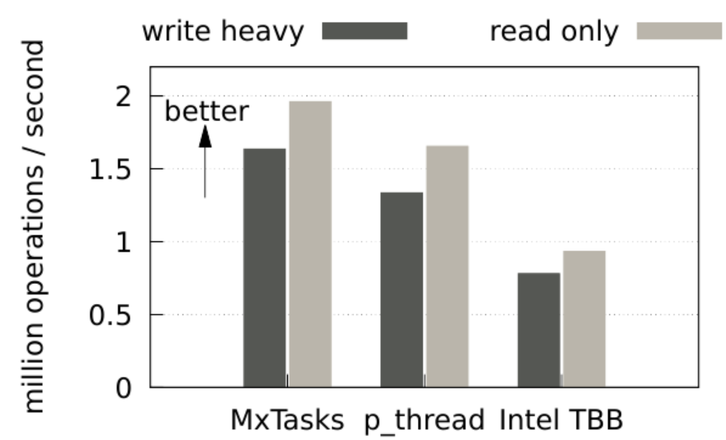
Task Metadata Pays Off: Prefetching [6]

- A glance into the future of memory accesses
- Optimization is fully **automatic**
- Performance impact:





(a) Memory Stalls

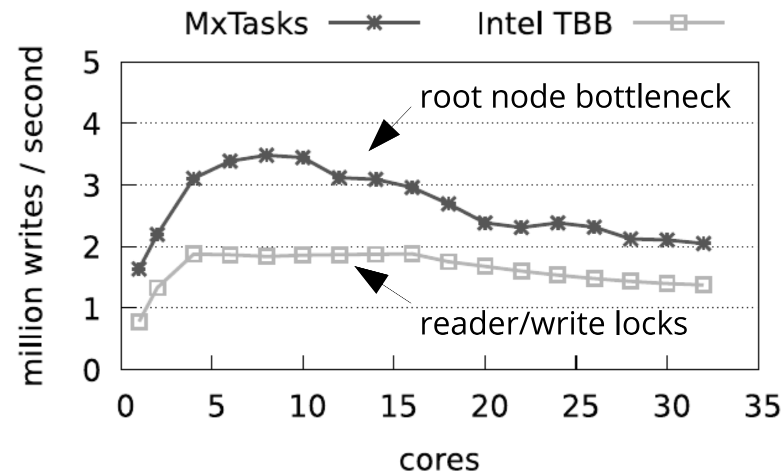


(b) Throughput

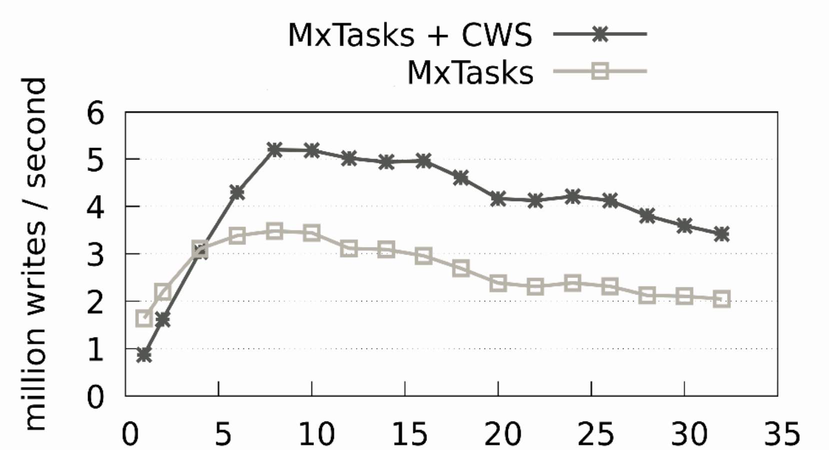
- B^{link}-tree insert and lookup operations on a 32 core Intel Xeon E5-2690
- measured with MxTasking on top of Linux

… Pays Off: Task Synchronization [6]

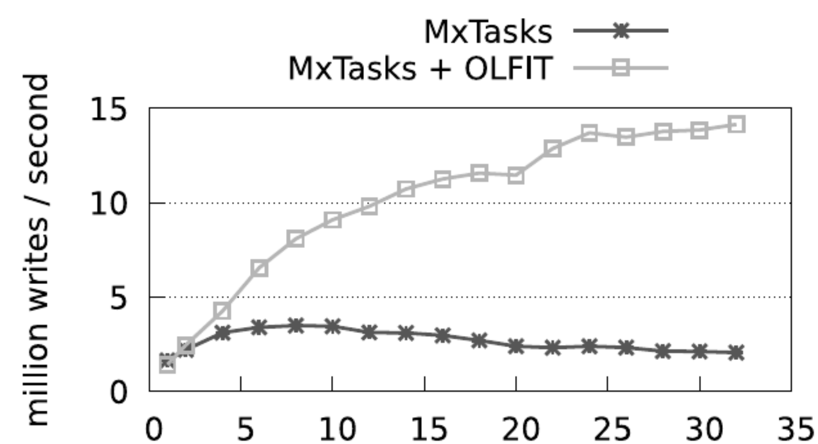
- Task-to-core-mapping can implicitly avoid locking
  - Objects are assigned to cores
  - Tasks accessing an object are spawned on that core



- Problem: Load balancing not trivial, but MxKernel can help



"Coarse-grained Work Stealing"



Optimistic Locking with OLFIT [7]

Outline

- Manycore Programming
- Manycore OS Research
- MxKernel Architecture
- Preliminary Results
- **Conclusions**

Conclusions

- Fine-grained control flow abstractions: good for optimization
  - Challenge: Minimize overhead
  - Challenge: Exploit application knowledge
  - Challenge: Many mapping strategies possible, good theory missing
- Heterogeneous computing resources can be integrated
  - Challenge: Lack of low-level hardware documentation
- Hypervisor technology and OS might converge
- It’s more than just fun: Compatibility layer possible!

References (1)

[1] A. Agarwal, J. Miller, D. Wentzlaff, H. Kasture, N. Beckmann, C. Gruenwald III, and C. Johnson, *FOS: A factored operating system for high assurance and scalability on multicores*. Massachusetts Institue of Technology. Technical Report AFRL-RI-RS-TR-2012-205, August 2012.

[2] Intel® Threading Building Blocks – Tutorial, Document Number 319872-009US, http://www.intel.com

[3] V. Leis, P. Boncz, A. Kemper, and T. Neumann. 2014. *Morsel-driven parallelism: a NUMA-aware query evaluation framework for the many-core age*. In Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD '14). ACM, New York, NY, USA, 743-754. DOI: https://doi.org/10.1145/2588555.2610507

References (2)

[4] A. Baumann, P. Barham, P.-E. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, and A. Singhania. *The Multikernel: A new OS architecture for scalable multicore systems*. In Proceedings of the 22nd ACM Symposium on OS Principles, Big Sky, MT, USA, October 2009.

[5] J. A. Colmenares, G. Eads, S. Hofmeyr, S. Bird, M. Moretó, D. Chou, B. Gluzman, E. Roman, D. B. Bartolini, N. Mor, K. Asanović, and J. D. Kubiatowicz. 2013. *Tessellation: refactoring the OS around explicit resource containers with continuous adaptation*. In Proceedings of the 50th Annual Design Automation Conference (DAC '13). ACM, New York, NY, USA, Article 76, 10 pages. DOI: https://doi.org/10.1145/2463209.2488827

[6] J. Mühlig, M. Müller, O. Spinczyk, and J.Teubner. *A novel System Software Stack for Data Processing on Modern Hardware*. Datenbank-Spektrum, 20(3):223-230, 2020.

References (3)

[7] Cha SK, Hwang S, Kim K, Kwon K. *Cache-conscious concurrency control of main-memory indexes on shared-memory multiprocessor systems*. In: Proceedings of the 27th International Conference on Very Large Databases (VLDB). Morgan Kaufmann Publishers Inc, San Francisco, CA, USA, pp 181–190, 2001.