Node-Based Memory Management for Scalable NUMA Architectures

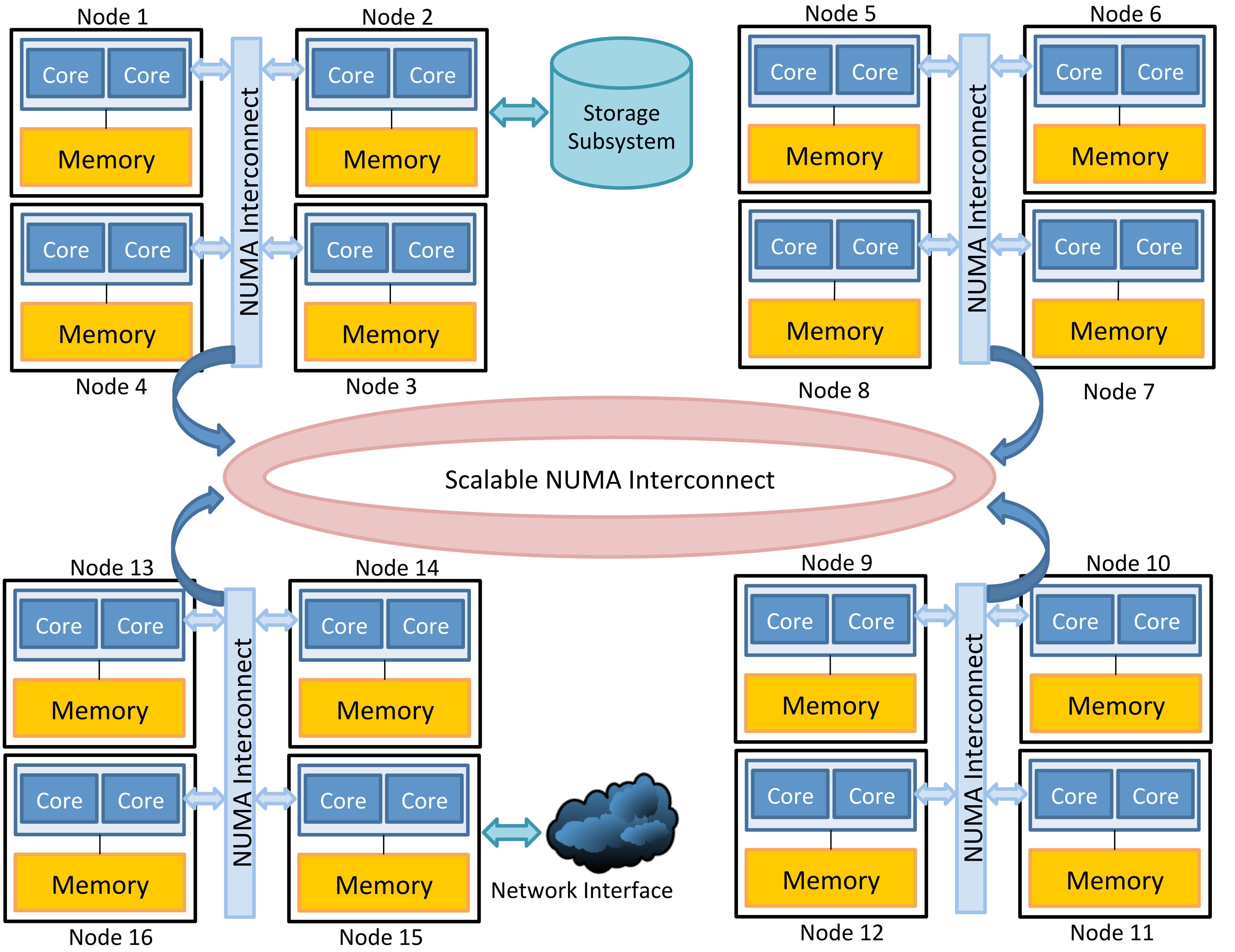
International Workshop on Runtime and Operating Systems for Supercomputers (ROSS 2012)

Stefan Lankes[1], Thomas Roehl[2], Christian Terboven[2], Thomas Bemmerl[1]

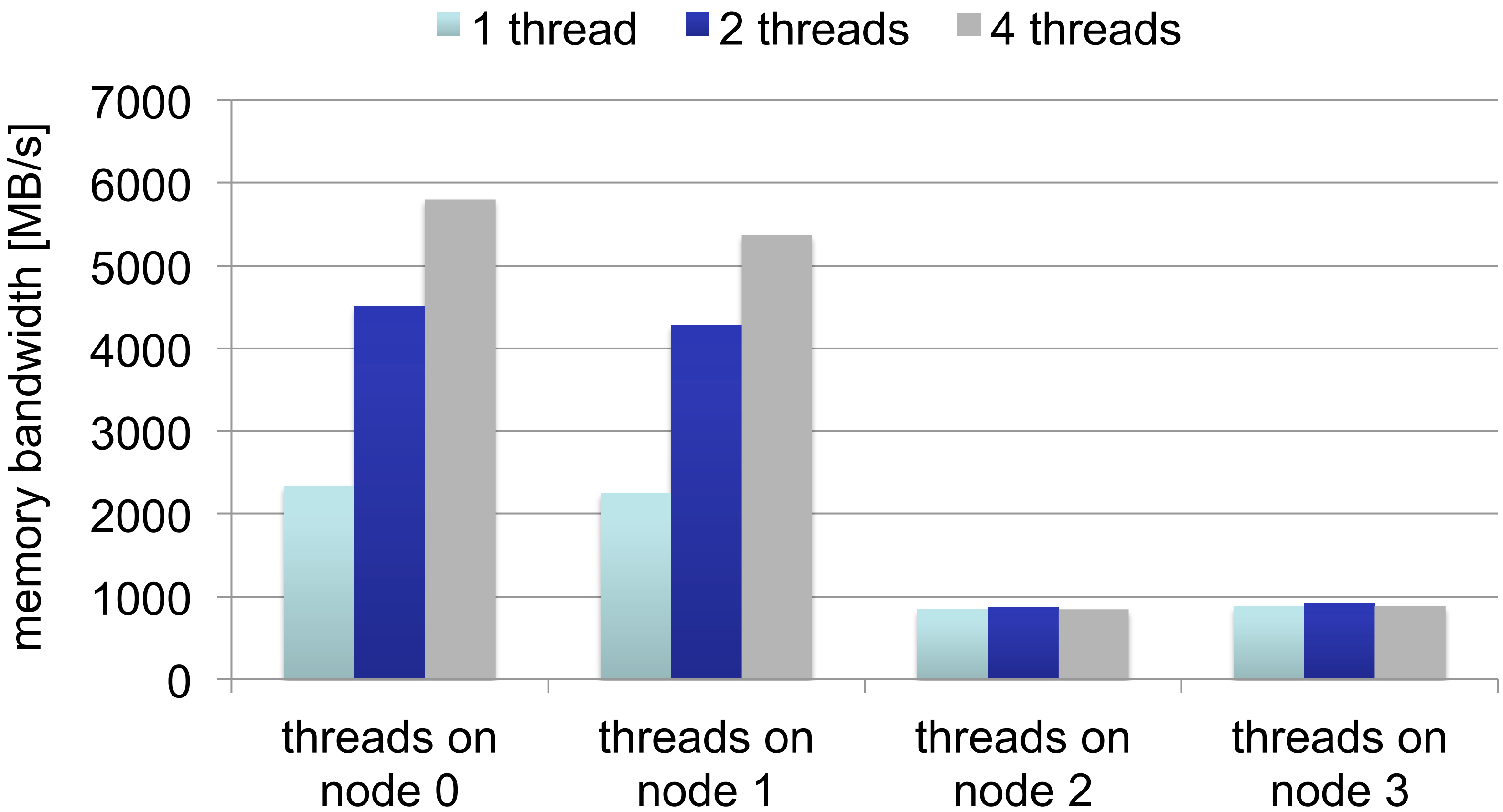
[1]Chair for Operating Systems, RWTH Aachen University
[2]Center for Computing and Communication, RWTH Aachen University

Outline

- Motivation
- Illustration of a common memory management
- Design of the node-based memory management
- Critical analysis
- Future prospects
- Benchmark results
- Conclusions and outlook

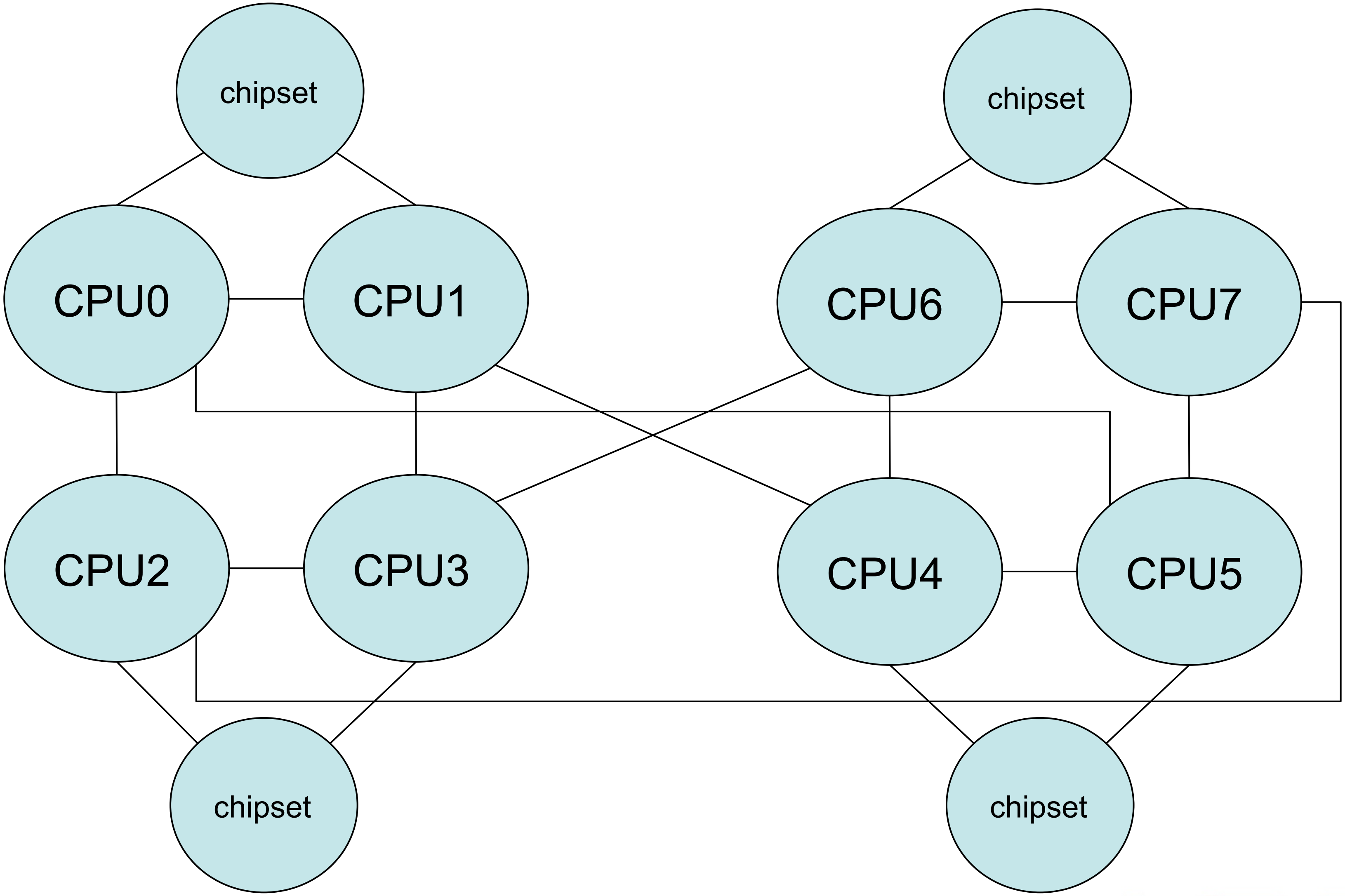

Node 1
Node 2
Node 3
Node 4
Node 5
Node 6
Node 7
Node 8
Node 9
Node 10
Node 11
Node 12
Node 13
Node 14
Node 15
Node 16
Core
Memory
NUMA Interconnect
Storage Subsystem
Scalable NUMA Interconnect
Network Interface

Performance Characteristics (NumaScale-Cluster)

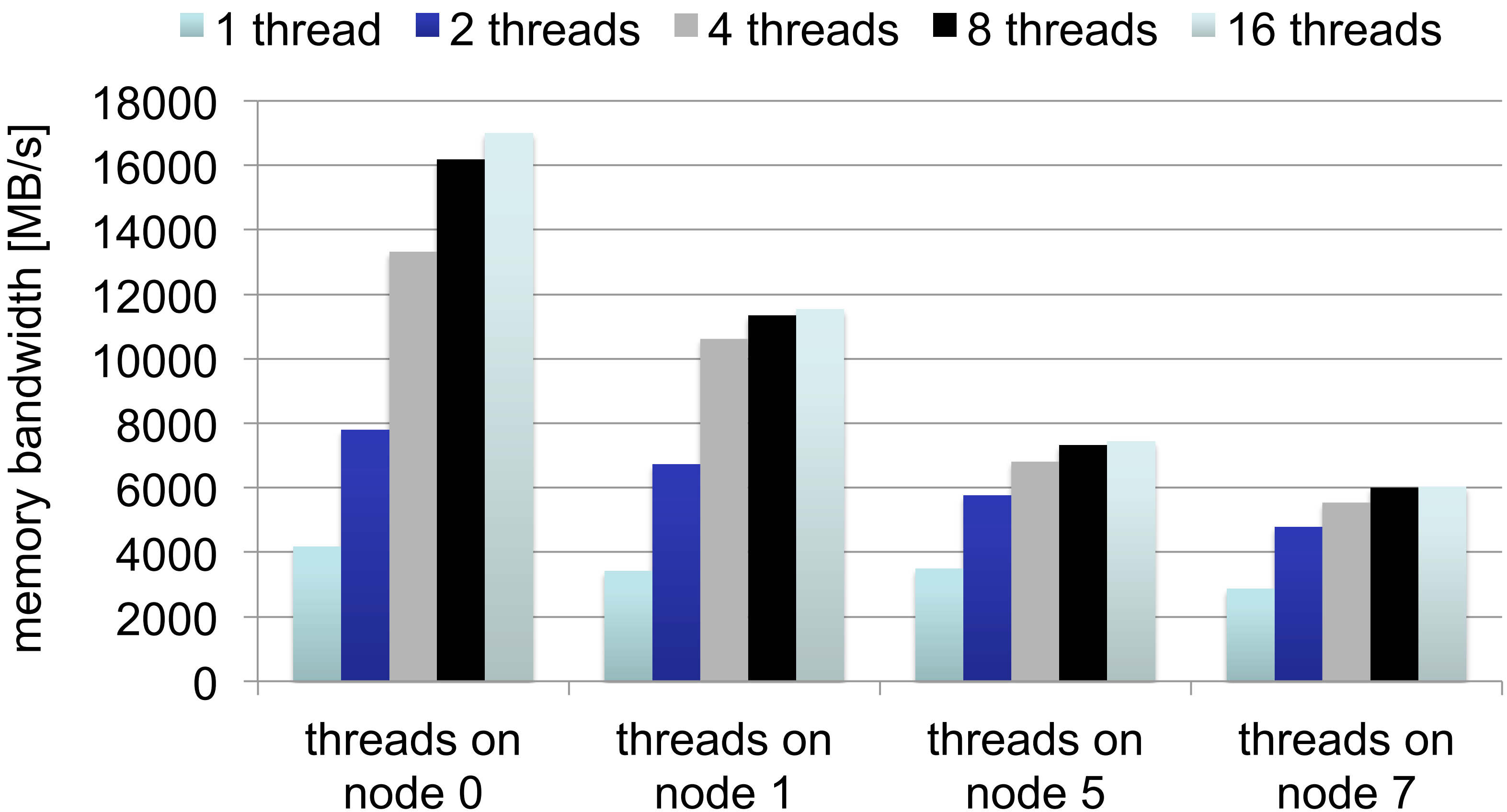


- 2 systems with 2 AMD QuadCores of type 8378 combined via NumaConnect
- all data on node 0

Eight-Socket Configuration
(Westmere-EX)

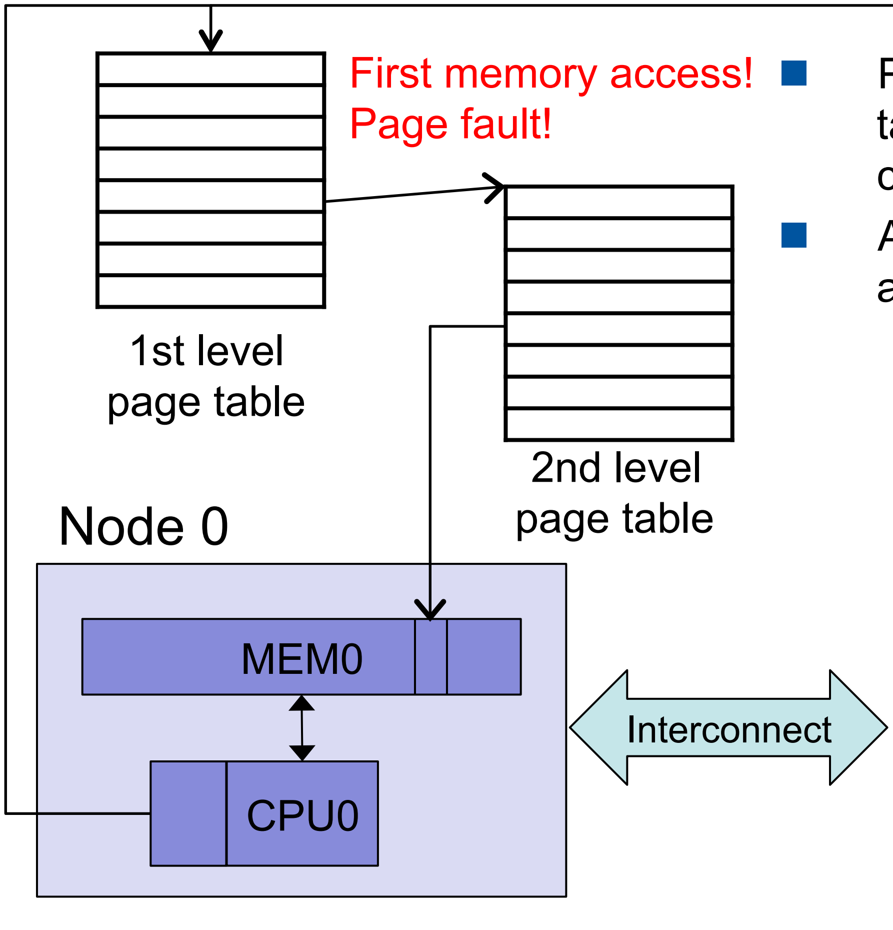


Performance Characteristics (Westemere-EX)



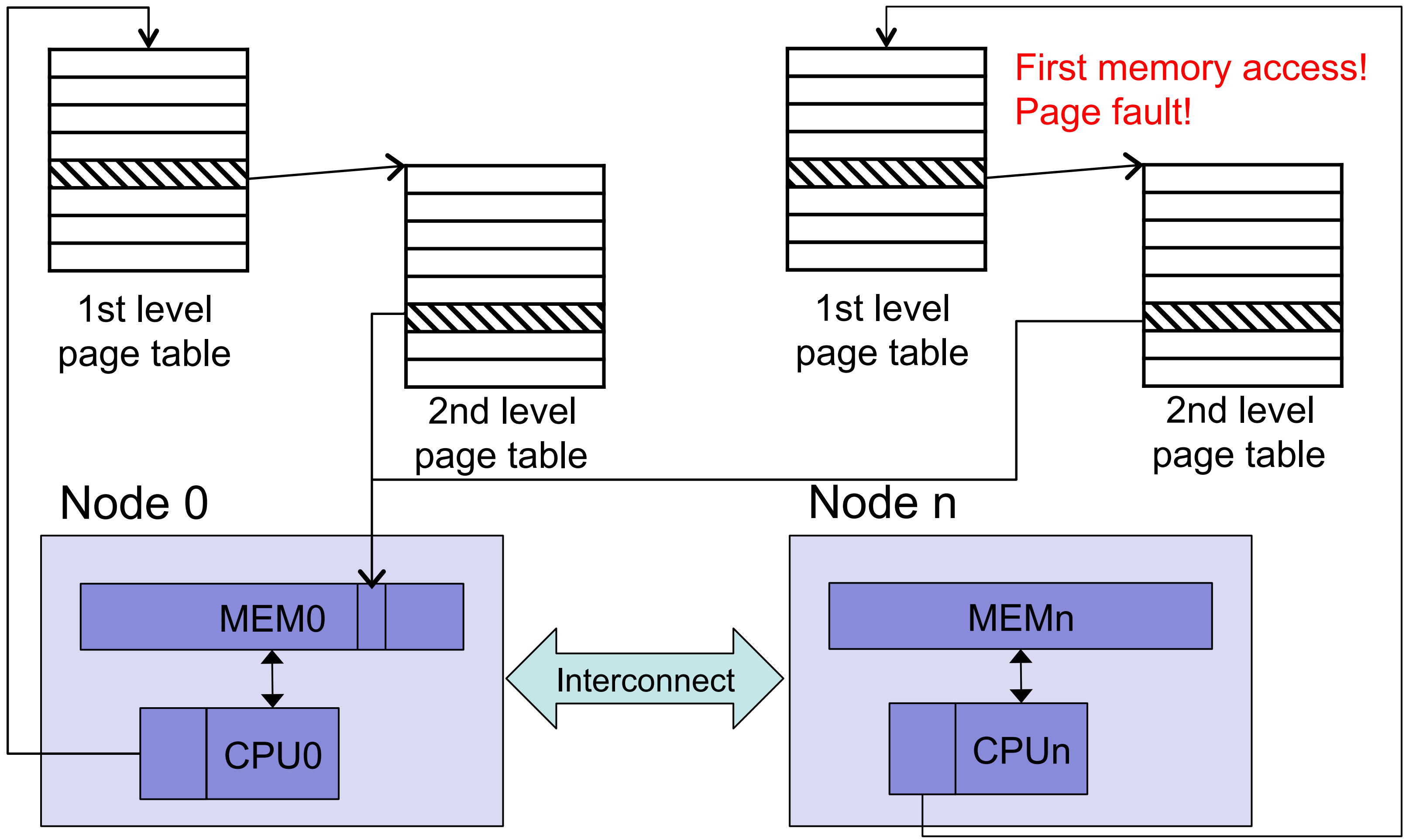
- 8 Intel Xeon CPU E7-8850 (Westmere-EX)
- 8 * 10 Cores / 8 * 20 Cores via HyperThreading
- all data on node 0

Common Memory Management
Process/thread creation

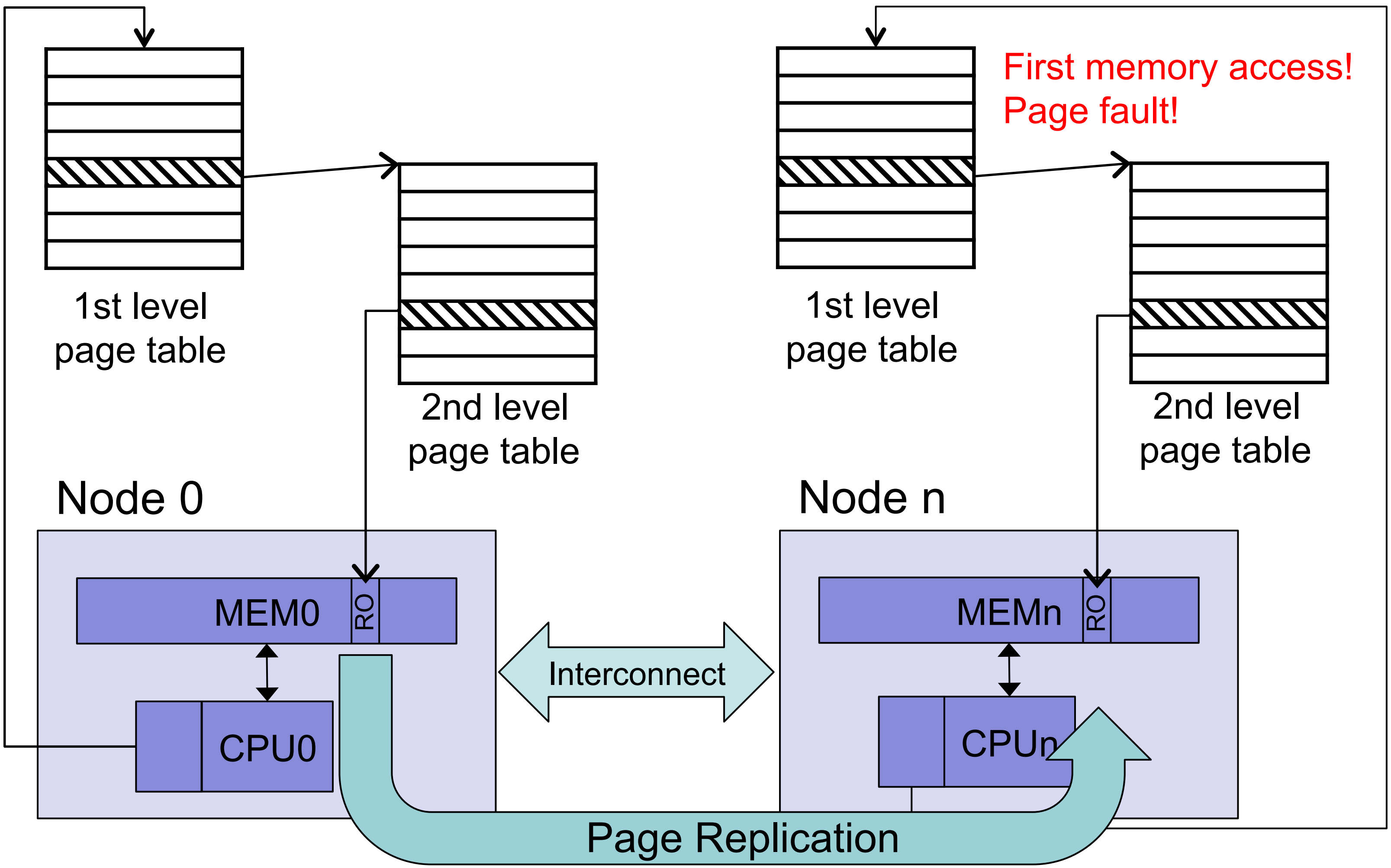


- Pointer to the 1st level page table is also part of the process control block
- All threads use the identical address space
  - → Same entry point on all cores

Page Table per Node
Basic idea



Page Table per Node
Replication of read-only regions



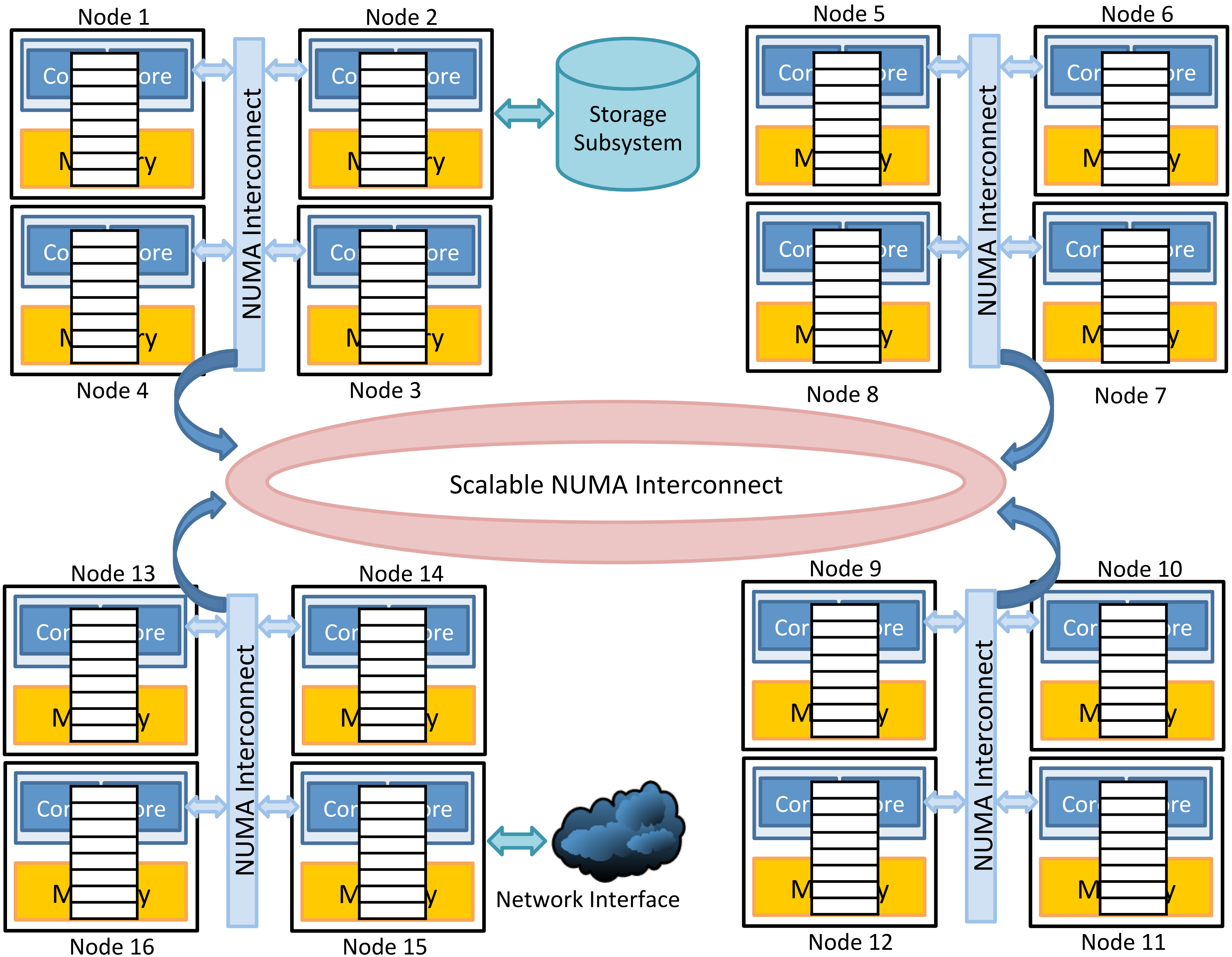
Advantages & Disadvantages

- Pro:
  - → Reflecting actual hardware at mapping layer
  - → After duplication only accesses to local memory
  - → Easy preparation of applications to use `mprotect()`

- Contra:
  - → Memory overhead
    - » One page table per NUMA node
    - » Duplicated pages
  - → Replication time
  - → Searching for mappings at all NUMA nodes (page fault, `mprotect()`, `free()`)

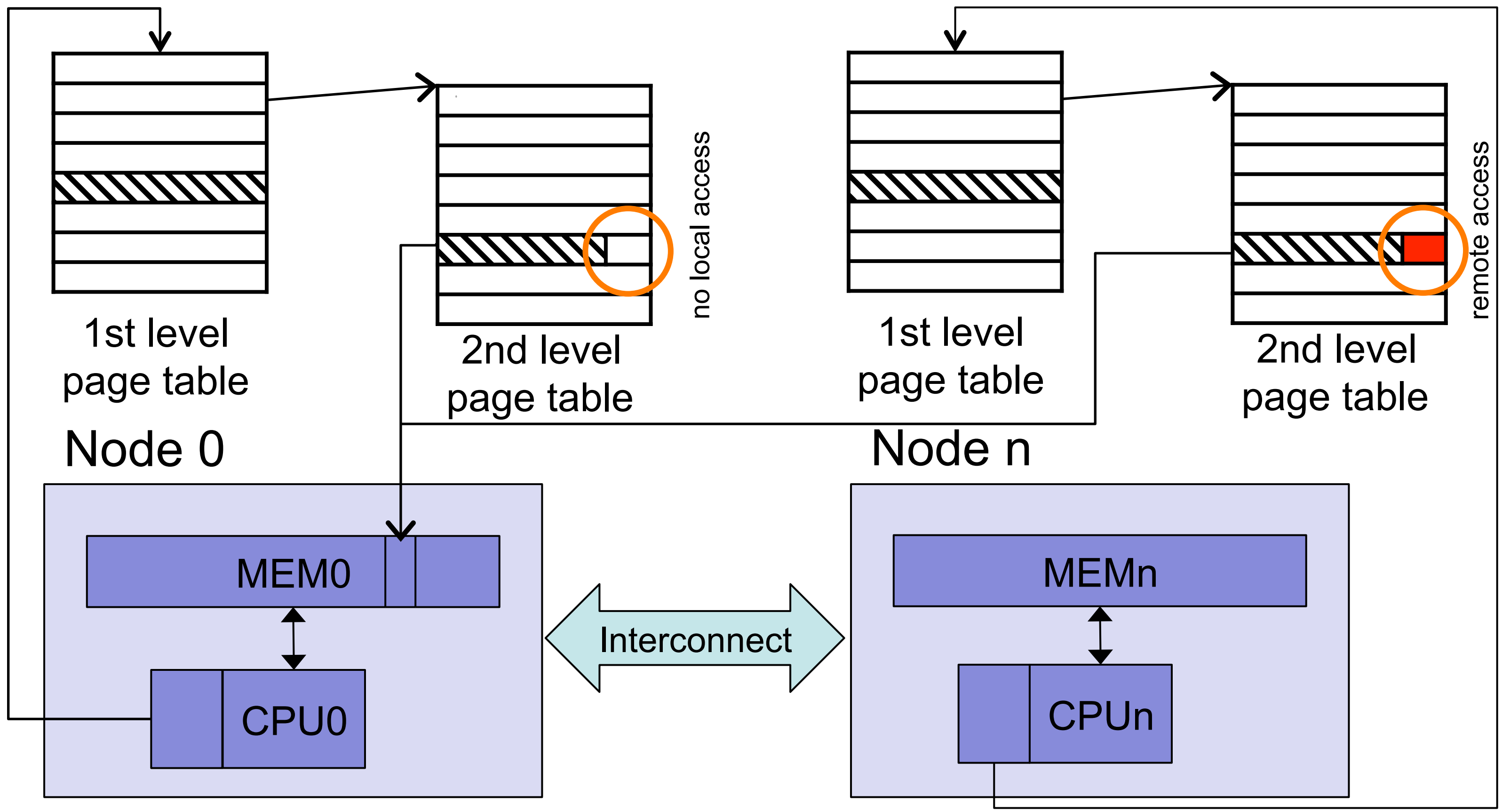
Avoid PGT-Traversal at Mapping Search

- Current Approach
  - → Searching for mappings at all NUMA nodes
  - → On which node should we start?
- Under development
  - → Use node-distance based search
    - » Does not guarantee less work
  - → Add new management structure
    - » Derived page table stores virtual address-to-nodemask mappings
    - » Needs 2 page table traversals per search,
    - » First resolve location, then address
    - » Increases memory footprint


Node 1
Node 2
Node 3
Node 4
NUMA Interconnect
Storage Subsystem
Node 5
Node 6
Node 7
Node 8
NUMA Interconnect
Scalable NUMA Interconnect
Node 13
Node 14
Node 15
Node 16
NUMA Interconnect
Network Interface
Node 9
Node 10
Node 11
Node 12
NUMA Interconnect

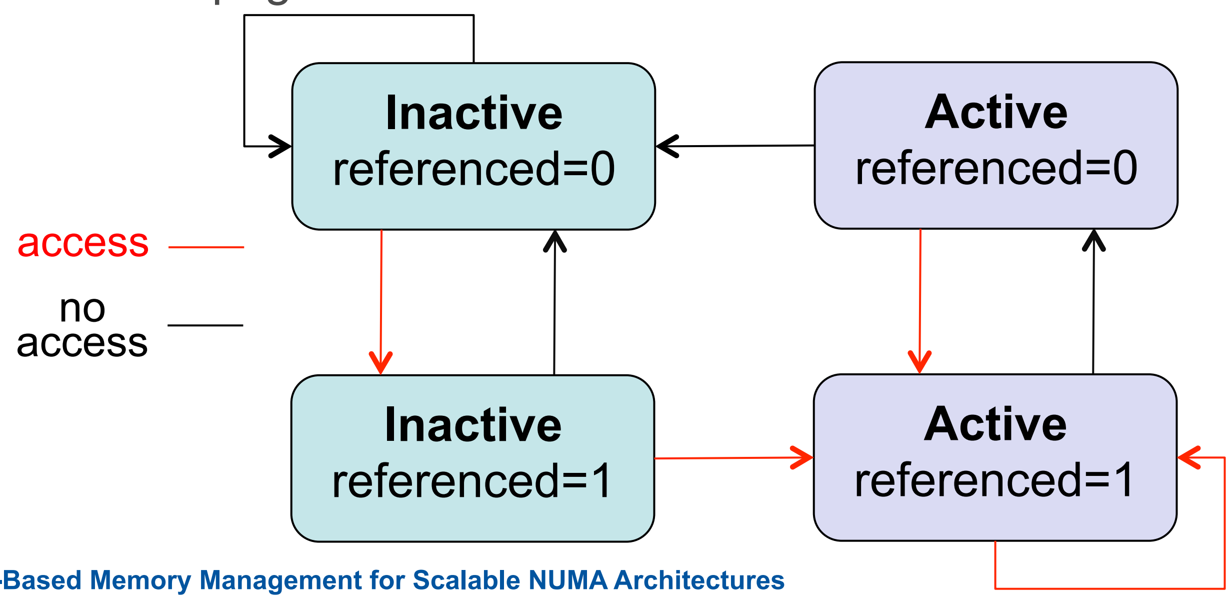
Detection of Performance Issues

- Page tables include access/dirty bits to record memory accesses.
  - → Usable to detect performance issues?



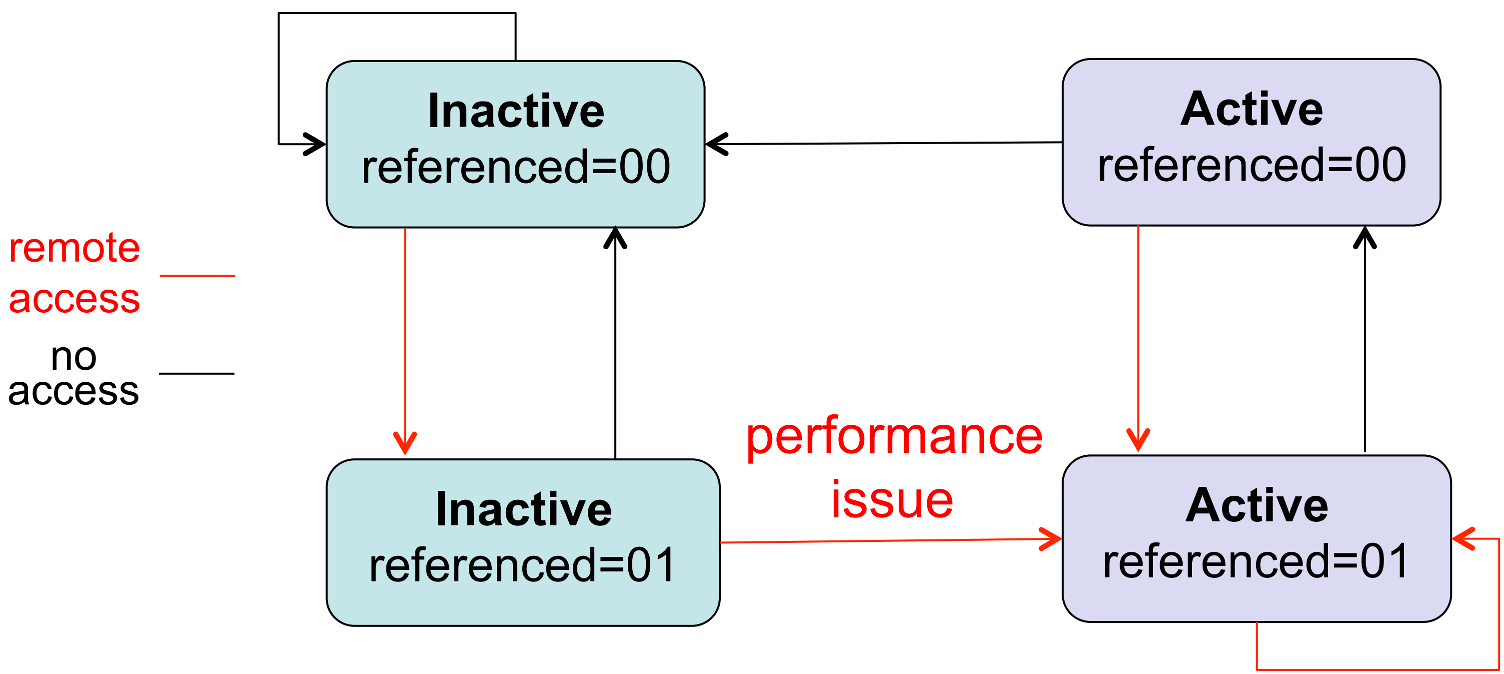
Common usage of the access / dirty bits

- Normally used to realize demand paging.
  - → Approximation of Least Recently Used (LRU)
  - → Classical concept
    - » Managing of two lists of active and inactive page frames
    - » State transition realized via access bits
    - » Doubling the number of accesses via a reference bit to move pages from the inactive to active list.



Transfer to the Node-based Memory Management

- Usage of two reference bits
  → One to signalize local and one to signalize remote memory accesses
- Abstract of the new state graph



Jacobi solver as Application Benchmark

- Solving of $A \cdot x = b, A \in R^{n \times n}, b \in R^n, x \in R^n$
- Iterative rule:

$$x_i^{m+1} = \frac{1}{a_{i,i}} \left(b_i - \sum_{j \neq i} a_{i,j} x_j^m \right)$$

- Abstract code for the new memory management

```
(sequential) initialization of A, b and x0
forbid write access to A and b
while(!found_solution)
   parallel for over the iterative rule
allow write access to A and b
```

- Straightforward implementation

Jacobi solver as Application Benchmark

- Solving of $A \cdot x = b, A \in R^{n \times n}, b \in R^n, x \in R^n$
- Iterative rule:

$$x_i^{m+1} = \frac{1}{a_{i,i}} \left(b_i - \sum_{j \neq i} a_{i,j} x_j^m \right)$$

- Abstract code

```
(sequential) initialization of A, b and x0
forbid write access to A and b
while(!found_solution)
   parallel for over the iterative rule
allow write access to A and b
```

Jacobi solver as Application Benchmark

- Solving of $A \cdot x = b, A \in R^{n \times n}, b \in R^n, x \in R^n$
- Iterative rule:

$$x_i^{m+1} = \frac{1}{a_{i,i}} \left(b_i - \sum_{j \neq i} a_{i,j} x_j^m \right)$$

- Abstract code

```
(sequential) initialization of A, b and x0
~~forbid write access to A and b~~ thread binding
while(!found_solution)
   parallel for over the iterative rule
~~allow write access to A and b~~
```

Jacobi solver as Application Benchmark

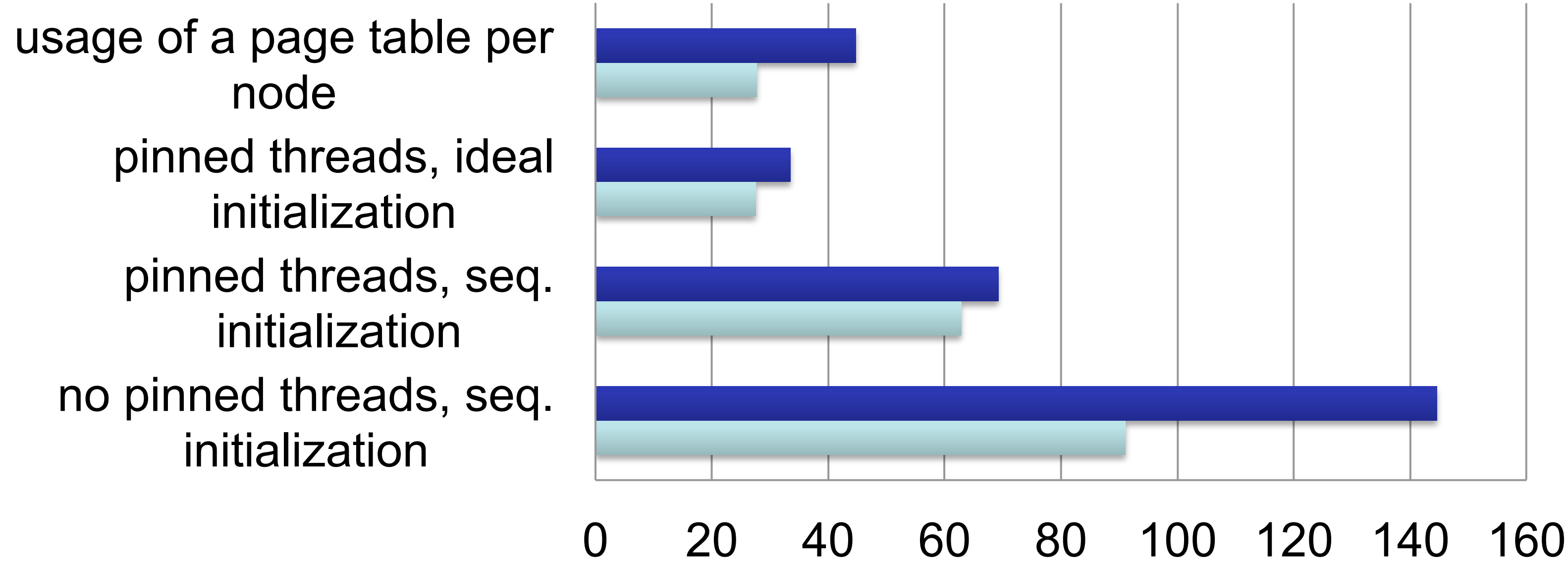
- Solving of $A \cdot x = b, A \in R^{n \times n}, b \in R^n, x \in R^n$
- Iterative rule:

$$x_i^{m+1} = \frac{1}{a_{i,i}}\left(b_i - \sum_{j \neq i} a_{i,j} x_j^m\right)$$

- Abstract code

```
(~~sequential~~ideal) initialization of A, b and x0
~~forbid write access to A and b~~ thread binding
while(!found_solution)
   parallel for over the iterative rule
~~allow write access to A and b~~
```

Jacobi solver (Westmere-EX)



	no pinned threads, seq. initialization	pinned threads, seq. initialization	pinned threads, ideal initialization	usage of a page table per node
80 threads	144,609	69,247	33,543	44,77
160 threads	91,067	62,864	27,517	27,746

```
matrix size: 5120 x 5120
iterations:  20000
```

Conclusions and Outlook

- Memory management can reflect the actual hardware
- First performance results are promising
- Reduction of overhead by
  - → usage of virtual address-to-node mapping
  - → bundling of NUMA nodes
- Introduce possibilities to detect performance issues
- Simple integration into existing programming models

```
#pragma  omp parallel for shared(A,B,C)
         readonly(A,B)
for (i=0; i<0; i++)
   C[i] = A[i] + B[i];
```

Thank you for your kind attention!

Stefan Lankes

Chair for Operating Systems
RWTH Aachen University
Kopernikusstr. 16
52056 Aachen, Germany

www.lfbs.rwth-aachen.de
contact@lfbs.rwth-aachen.de

Backup slides

Related Work

- Page placement strategies are extensively investigated
  - → Page placement via hints
    - » Affinity-On-Next-Touch
      - – Proposals: Nordergraaf & van der Pas
      - – Variations: Shermerhorn, Goglin et al., Bircsak at al.
    - » Template library of locality management (Majo & Gross)
  - → (Semi)automatic page placement
    - » profile-guided automatic page placement (Mueller et al.)
    - » dynamic page migration via counting remote memory accesses
      - – Memory controller extensions: SGI Origin
      - – Compiler extensions: Nikolopoulos et al.
- However, it exists room for optimizations.

Page Table per Node
Basic idea

- One page table per node
- Context switch: Load node-local page table
- Page fault
  - → Page not mapped: allocate new page and map locally
  - → Page mapped remotely:
    - » RW page: duplicate mapping
    - » RO page: duplicate page and map clone locally
- New system call to create a process, which uses our node-based memory management,
  - → Per default, the processes use the traditional concept.
- Via `mprotect` the page replication could be implicitly en- or disabled for certain memory regions.

Overhead (Westmere-EX)

	unmodified Linux kernel (3.3.8)	page table per node
time to allocate a page	1.666μs	6.671μs
time to protect a page	0.00005μs	0.032μs
time to replicate a page	___	4.479μs
time to unprotect a page	0.0001μs	0.148μs
time to replicate a reference	___	1.445μs

Test platform

- 8 Intel Xeon CPU E7-8850 (Westmere-EX)
- 8 * 10 Cores / 8 * 20 Cores via HyperThreading

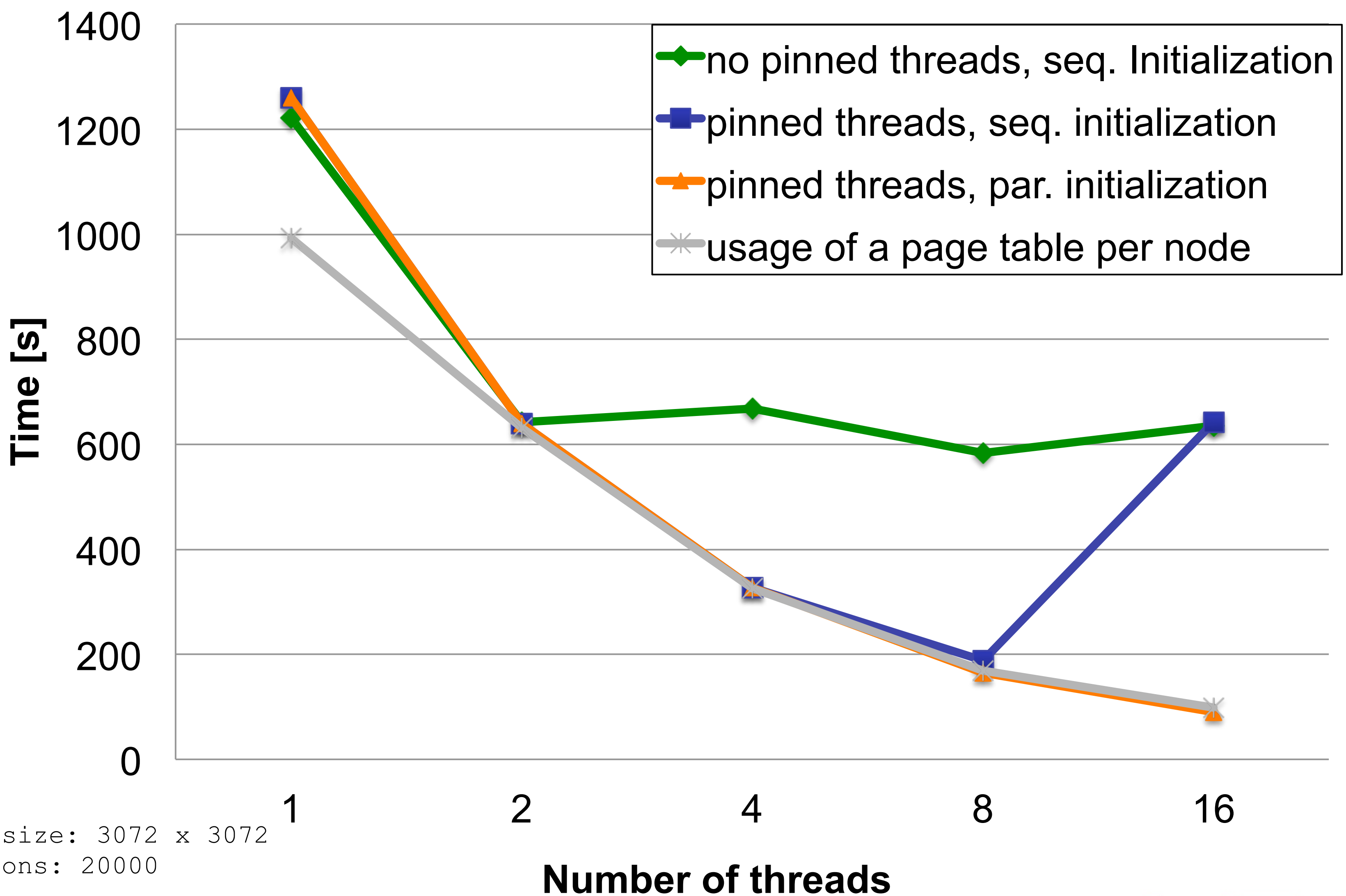
Overhead (NumaScale-Cluster)

	unmodified Linux kernel (2.6.37)	page table per node
time to allocate a page	2.810µs	3.143µs
time to protect a page	0.034µs	0.110µs
time to replicate a page	___	26.956µs
time to unprotect a page	0.195µs	2.787µs
time to replicate a reference	___	6.044µs

Test platform

- 2 systems with 2 AMD QuadCores of type 8378 combined via NumaConnect

Jacobi solver (NumaScale-Cluster)




```
matrix size: 3072 x 3072
iterations: 20000
```