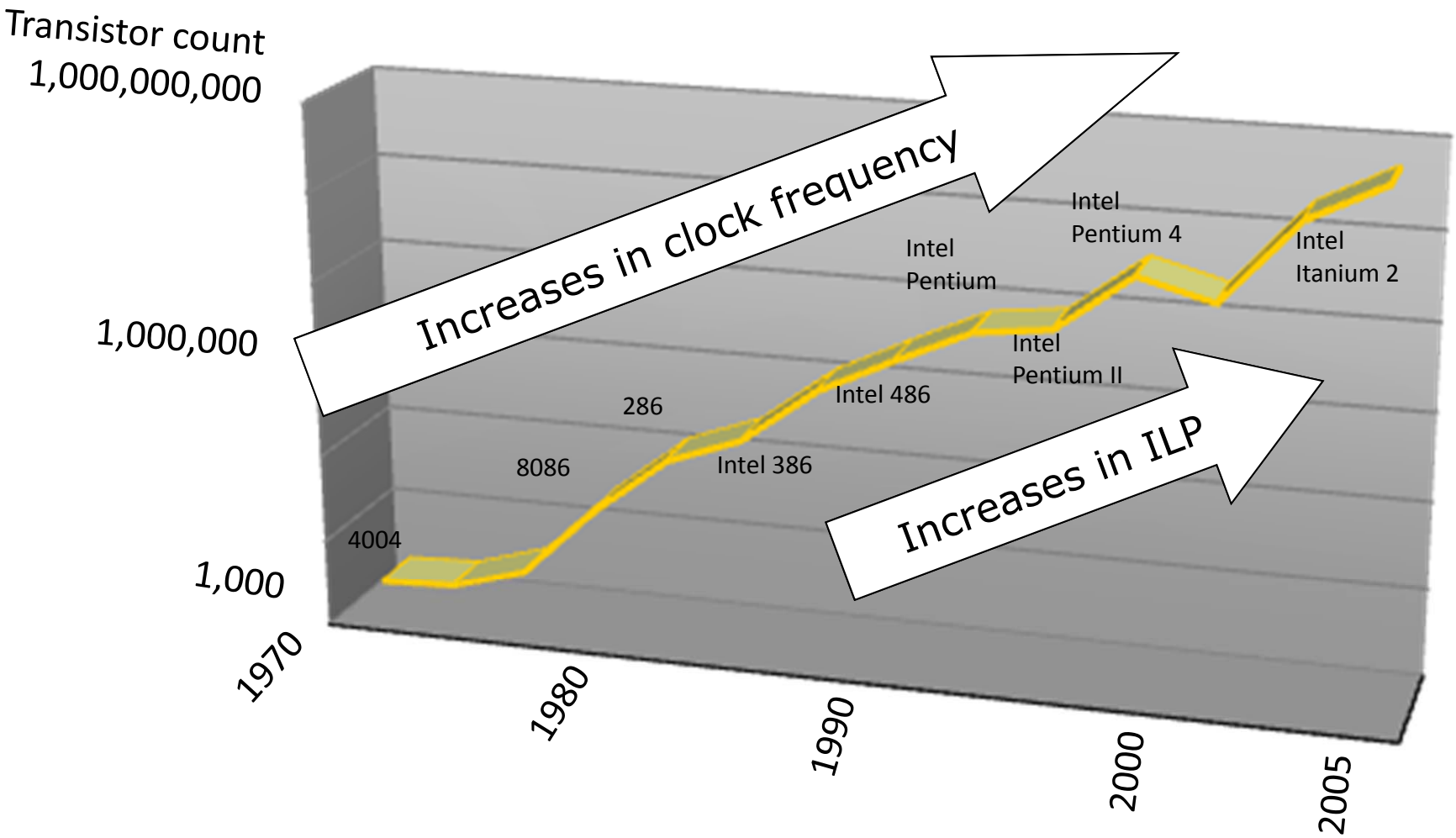
Chapter 9:
Multicore and Multikernels
Advanced Operating Systems
(263-3800-00L)

Timothy Roscoe

Herbstsemester 2012

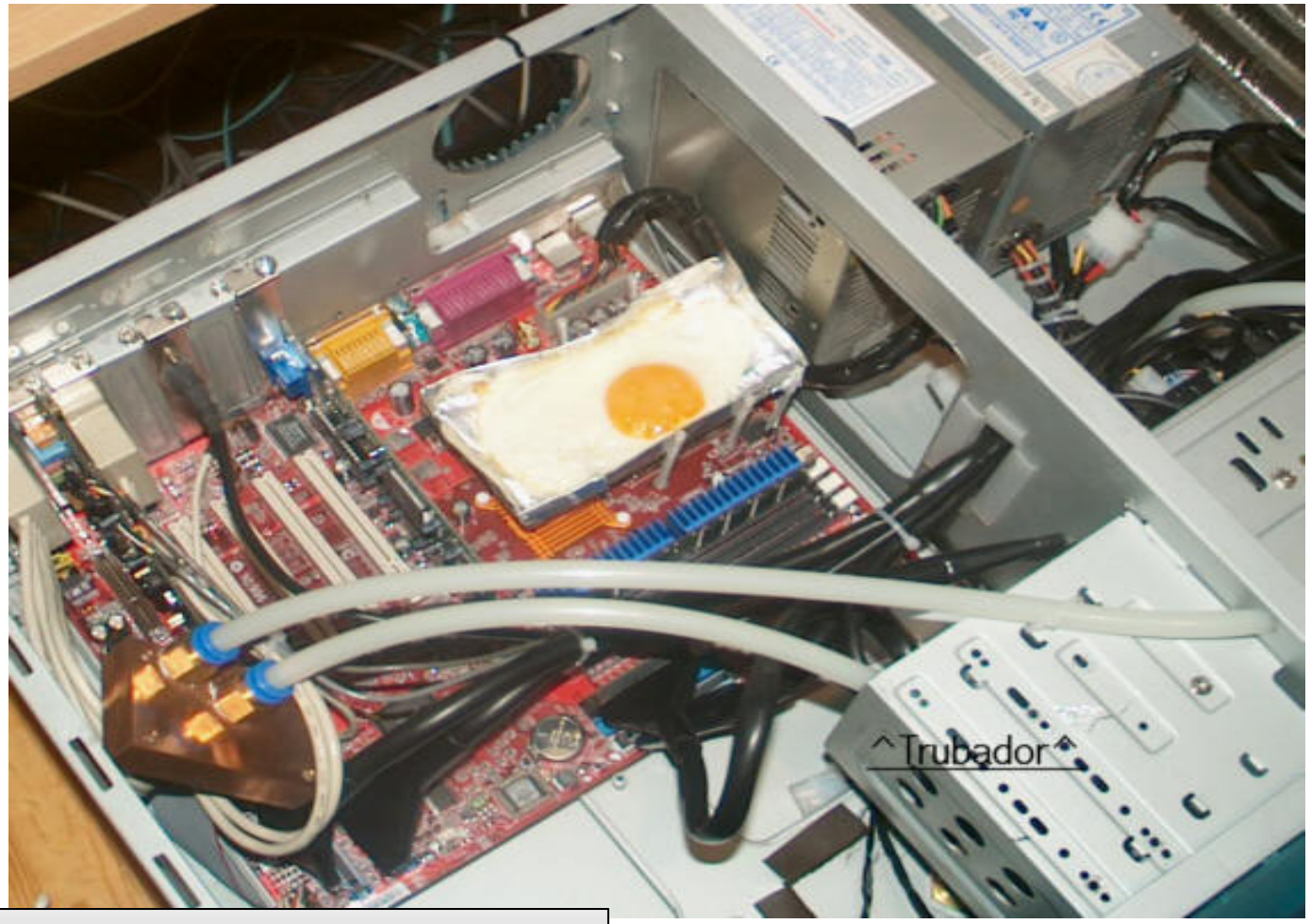
http://www.systems.ethz.ch/courses/fall2012/AOS

Moore's law: the free lunch



Source: table from http://download.intel.com/pressroom/kits/events/moores_law_40th/MLTimeline.pdf

The power wall

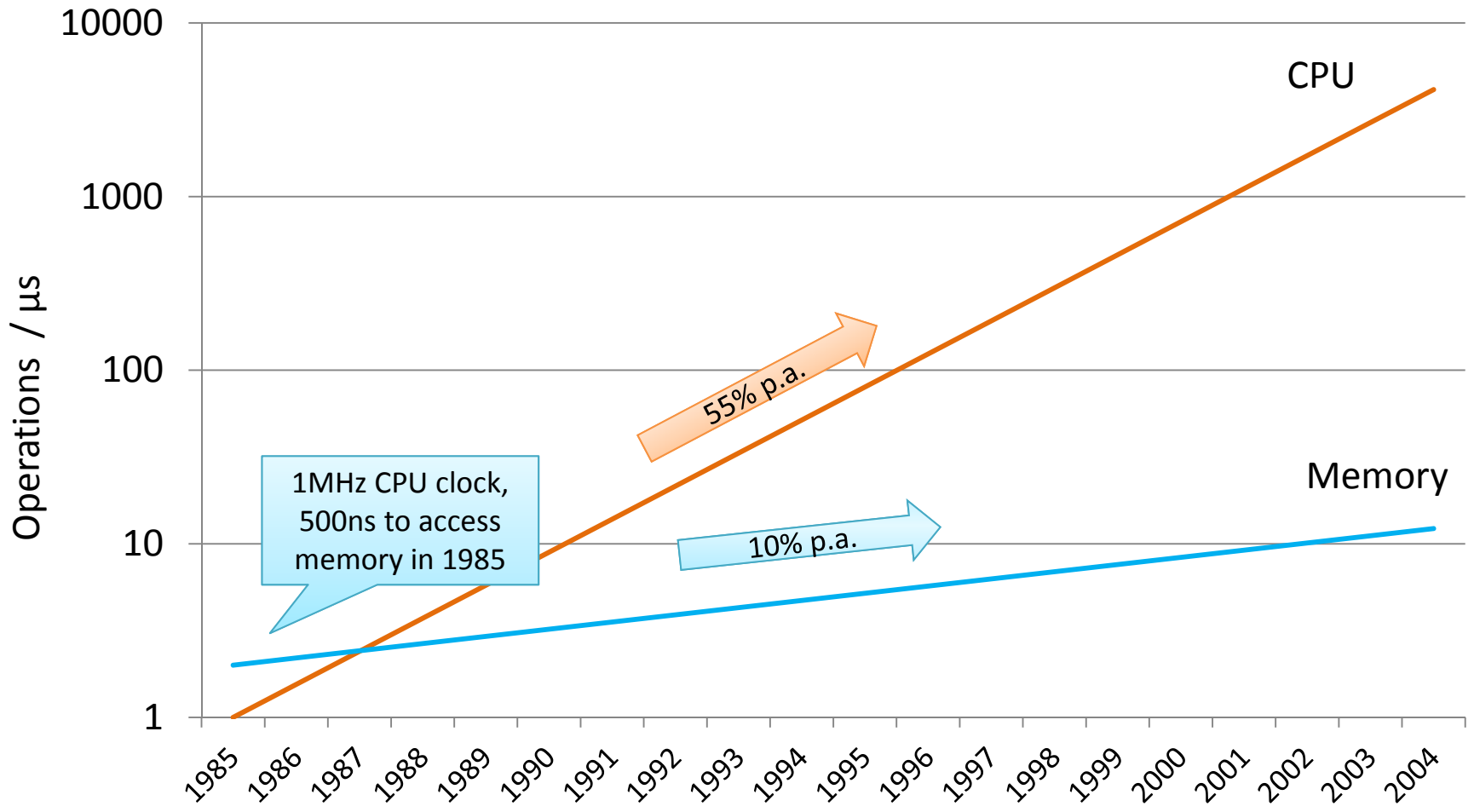


AMD Athlon 1500 processor
http://www.phys.ncku.edu.tw/~htsu/humor/fry_egg.html

The power wall

- Power dissipation depends on clock rate, capacitive load and voltage
  - Increases in clock frequency mean more power dissipated and so more cooling required
  - Decreases in voltage reduce dynamic power consumption but increase the static power leakage from transistors
- We've reached the practical power limit for cooling commodity microprocessors
  - Can't increase clock frequency without expensive cooling

The memory wall



The ILP wall

Systems@ETH Zürich

- ILP = “Instruction level parallelism”
- Implicit parallelism between instructions in 1 thread
- Processor can re-order and pipeline instructions, split them into microinstructions, do aggressive branch prediction etc.
  - Requires hardware safeguards to prevent potential errors from out-of-order execution
- Increases execution unit complexity and associated power consumption
  - Diminishing returns
- Serial performance acceleration using ILP has stalled

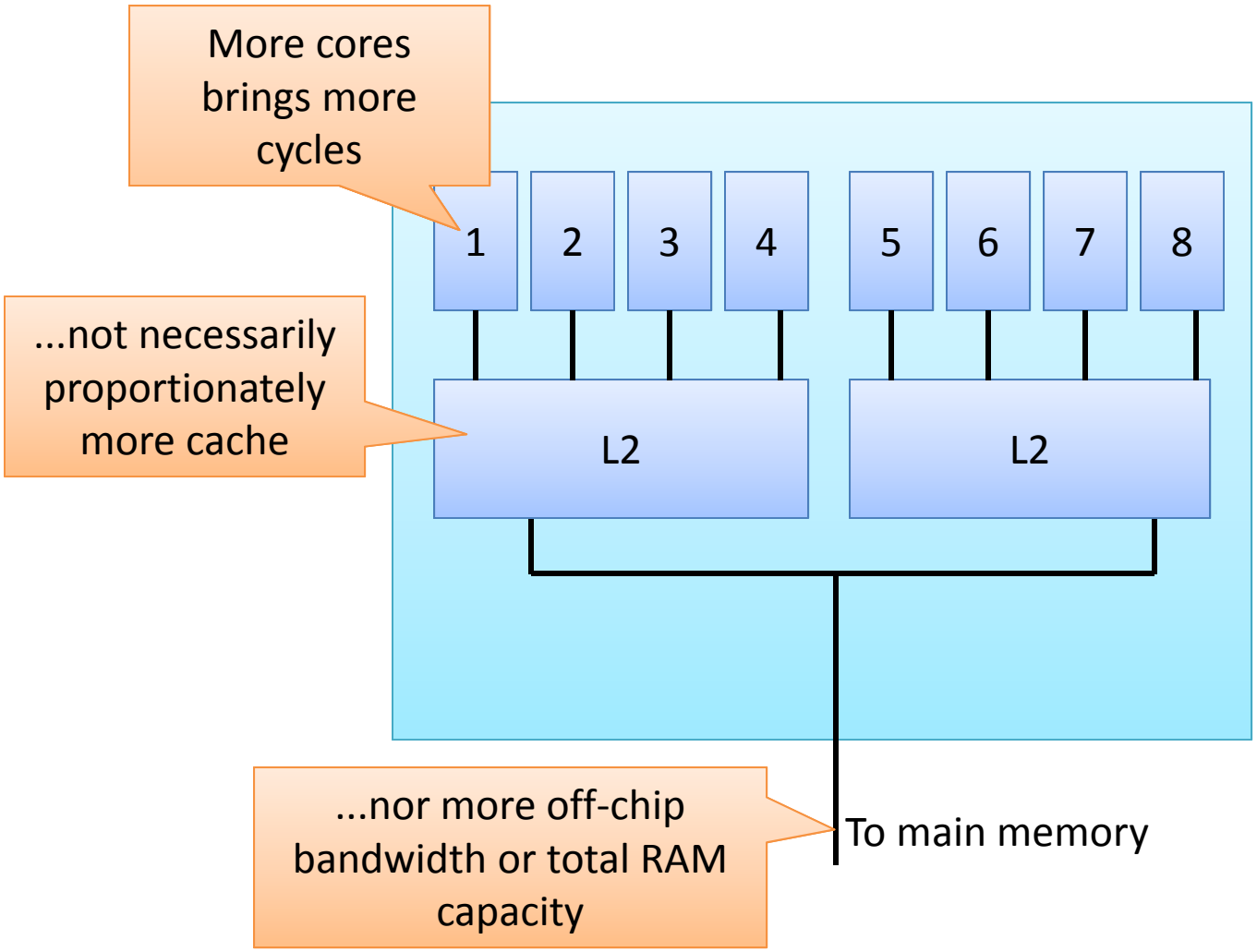
End of the road for serial hardware

- Power wall + ILP wall + memory wall = brick wall
  - Power wall $\Rightarrow$ can't clock processors any faster
  - Memory wall $\Rightarrow$ for many workloads performance dominated by memory access times
  - ILP wall $\Rightarrow$ can't keep functional units busy while waiting for memory accesses
- There is also a complexity wall, but chip designers don't like to talk about it…

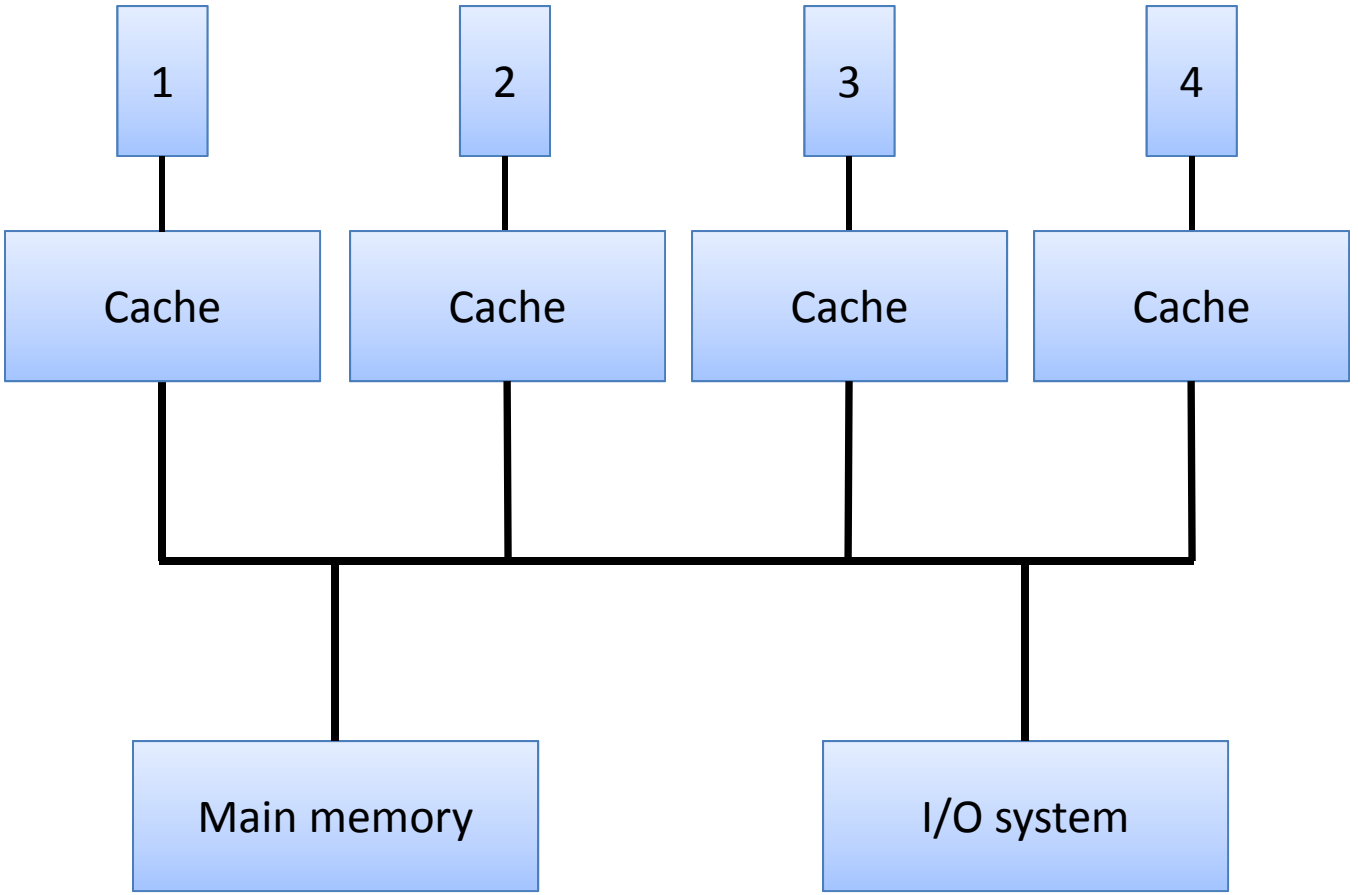
Multicore processors

- Multiple processor cores per chip
  - This is the future (and present) of computing
- Most multicore chips *so far* are shared memory multiprocessors (SMP)
  - Single physical address space shared by all processors
  - Communication between processors happens through shared variables in memory
  - Hardware typically provides cache coherence

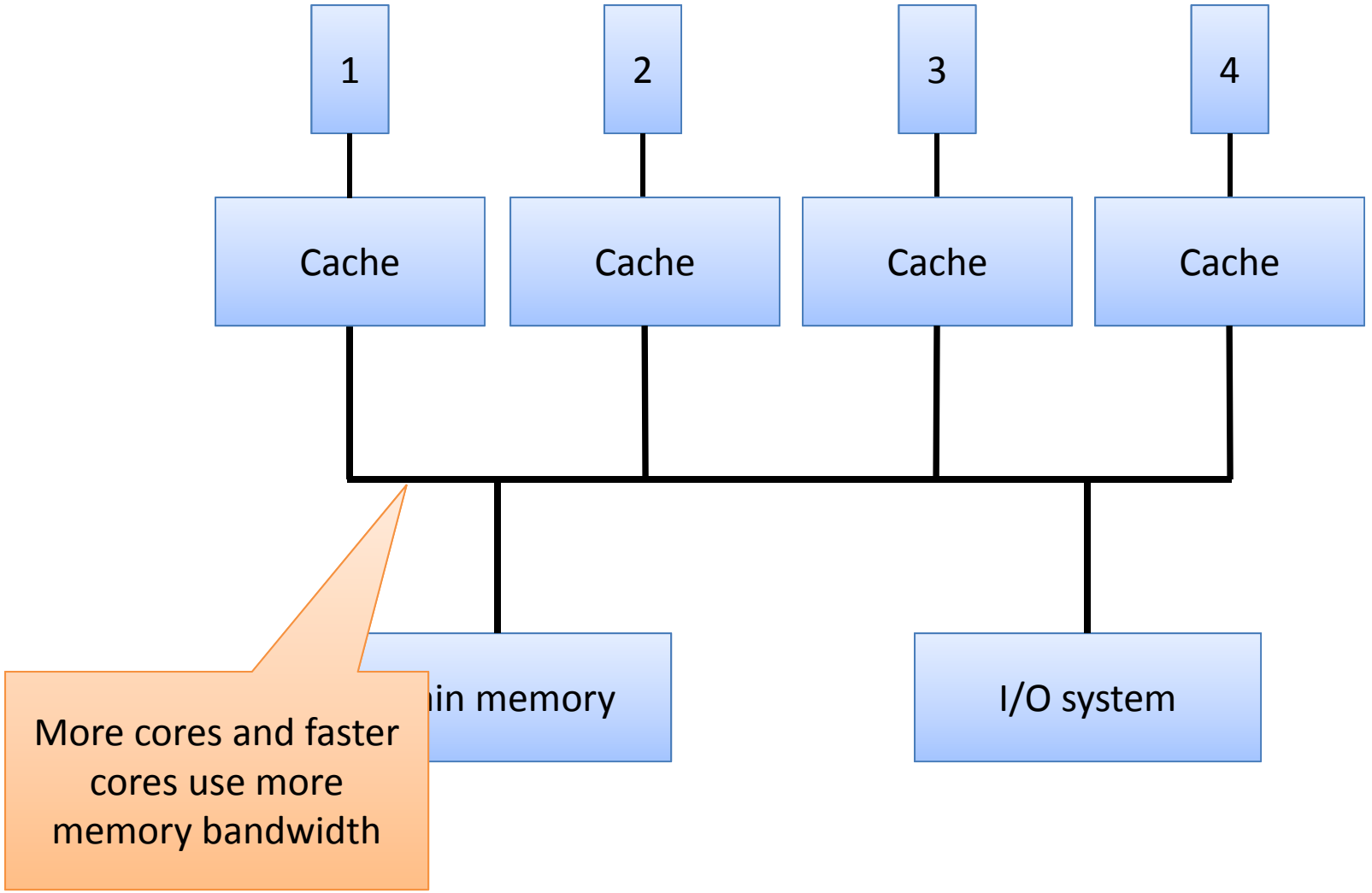
SMP architecture



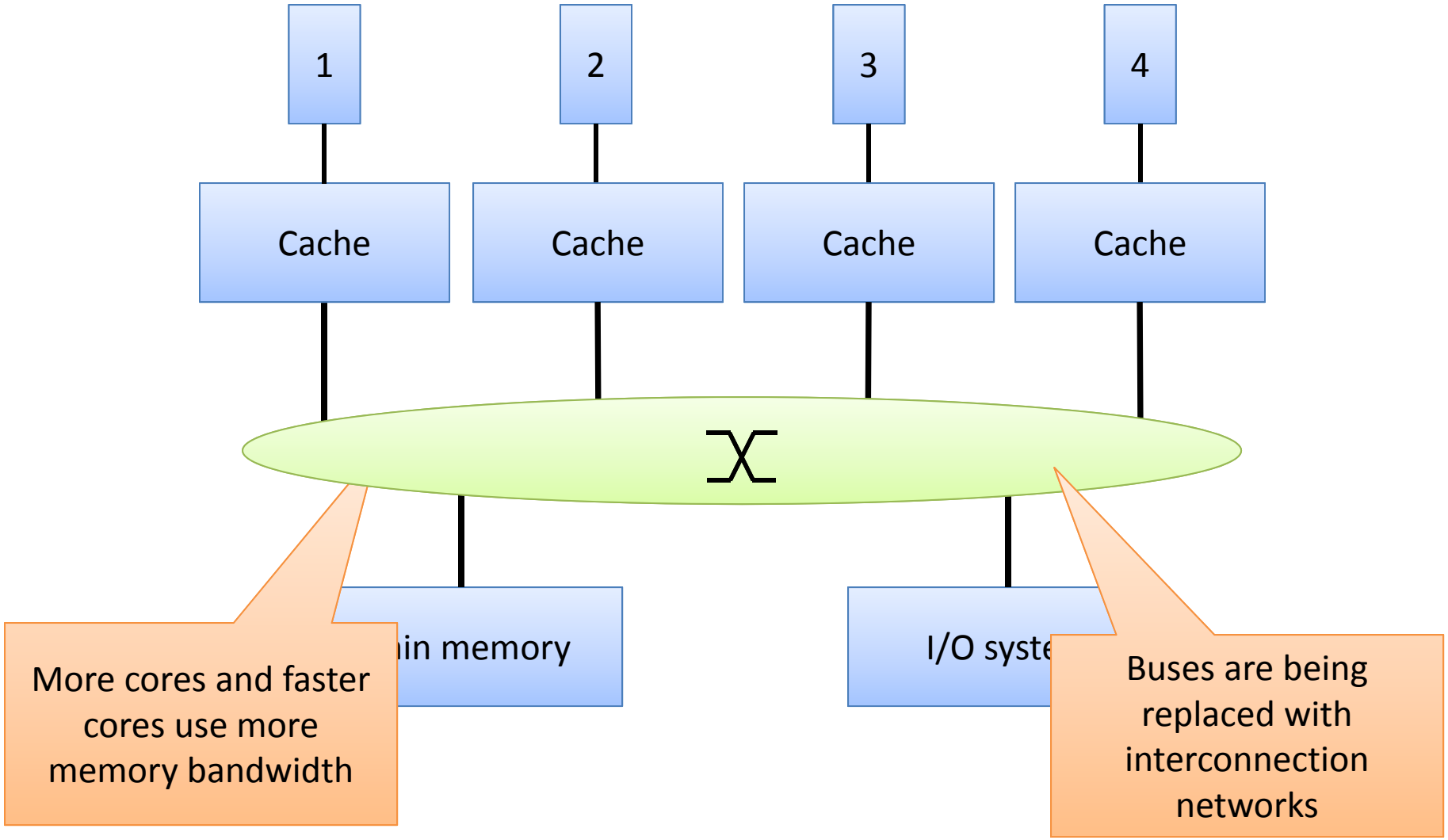
SMP architecture



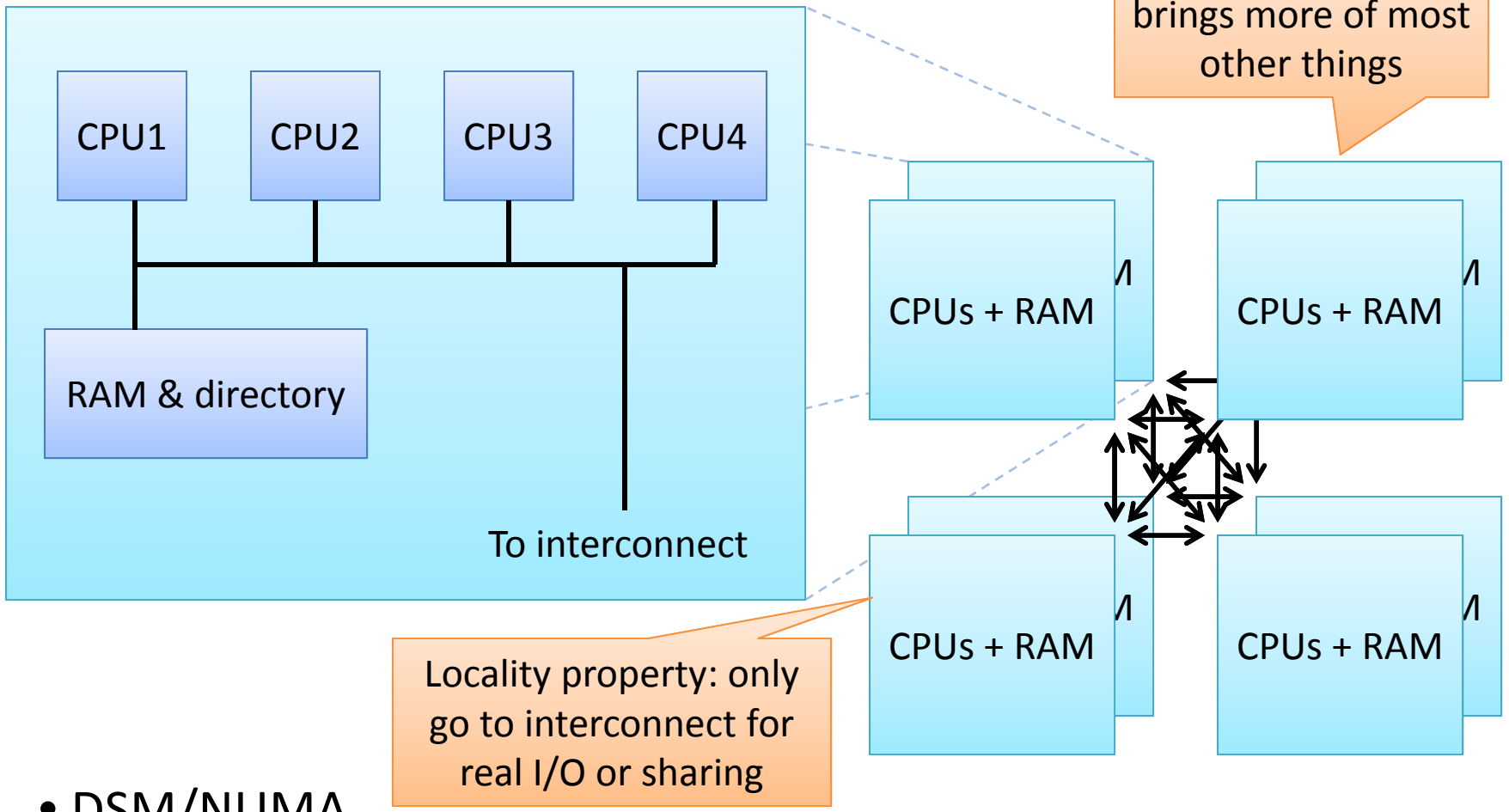
SMP architecture



SMP architecture

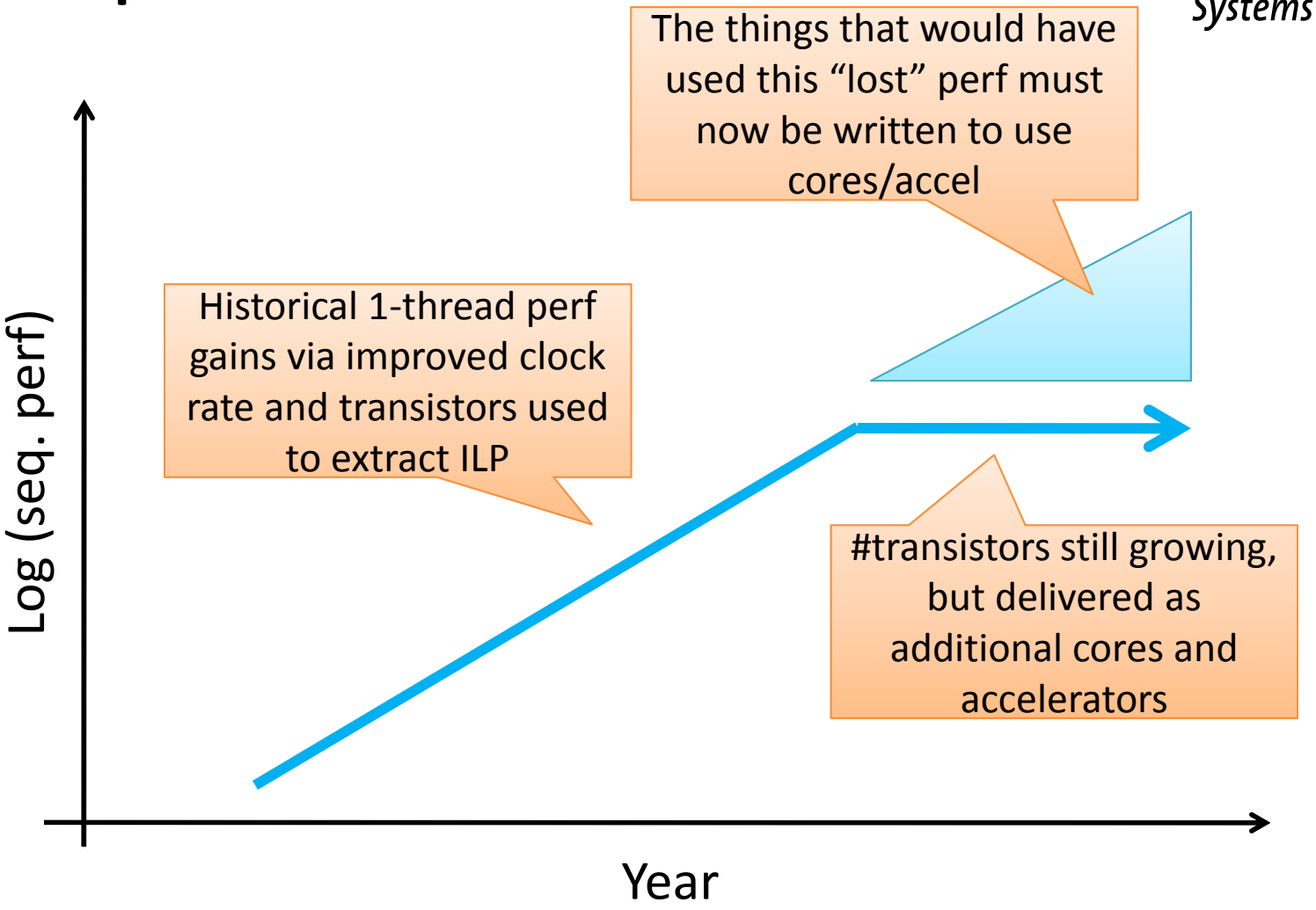


Distributed memory architecture



- DSM/NUMA
- Message-passing, eg clusters
- Could scale to 100s or 1000s of cores

Implications for software



Further reading

- “Challenges and Opportunities in Many-Core Computing”, John L. Manferdelli et al, Proceedings of the IEEE, 96(5), May 2008
- “Hitting the Memory Wall: Implications of the Obvious”, W.A. Wulf and Sally A. McKee, Computer Architecture News, 23(1), December 1994

Section 3

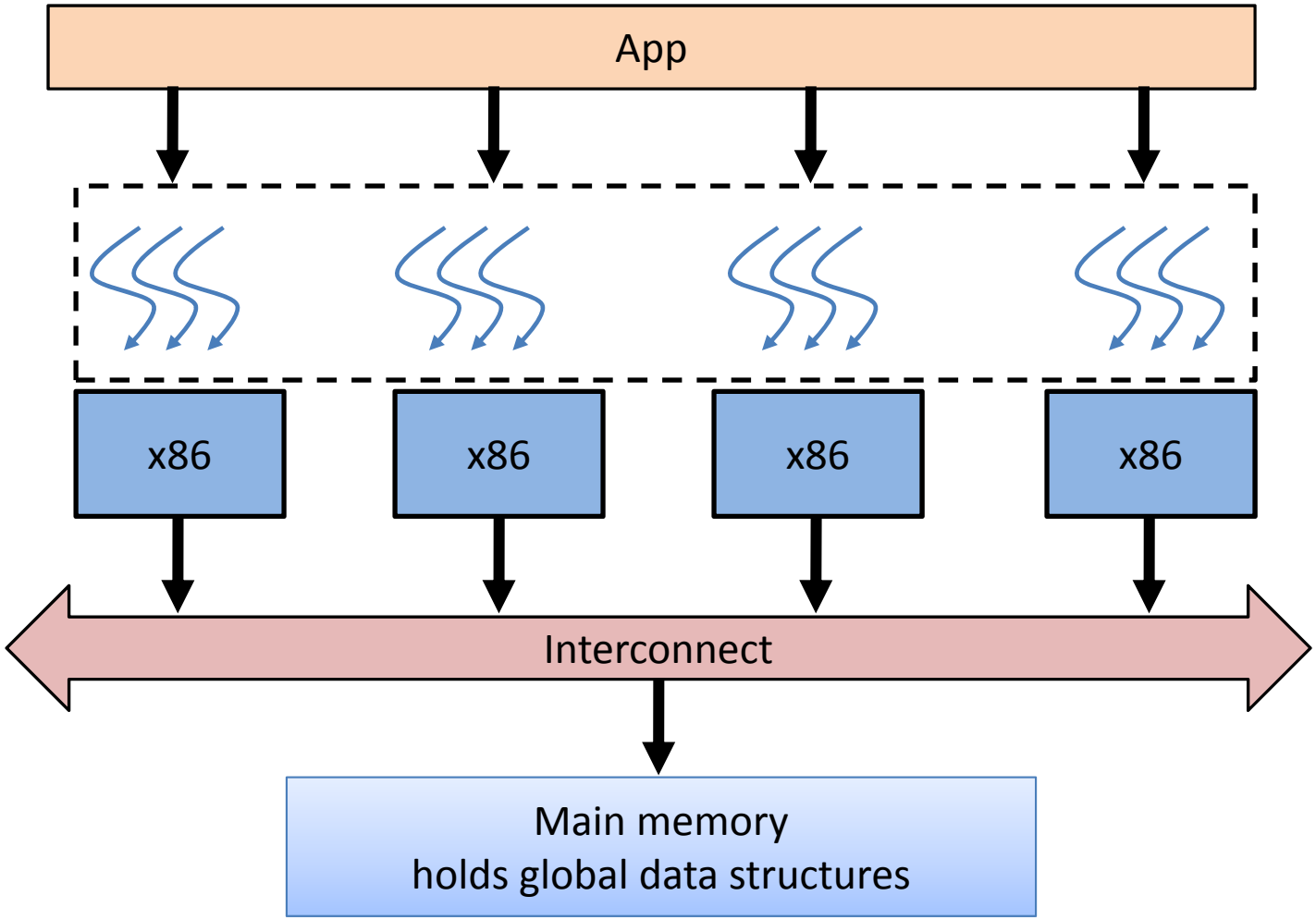
MULTICORE CHALLENGES FOR CURRENT OPERATING SYSTEMS

The Multicore OS Challenge

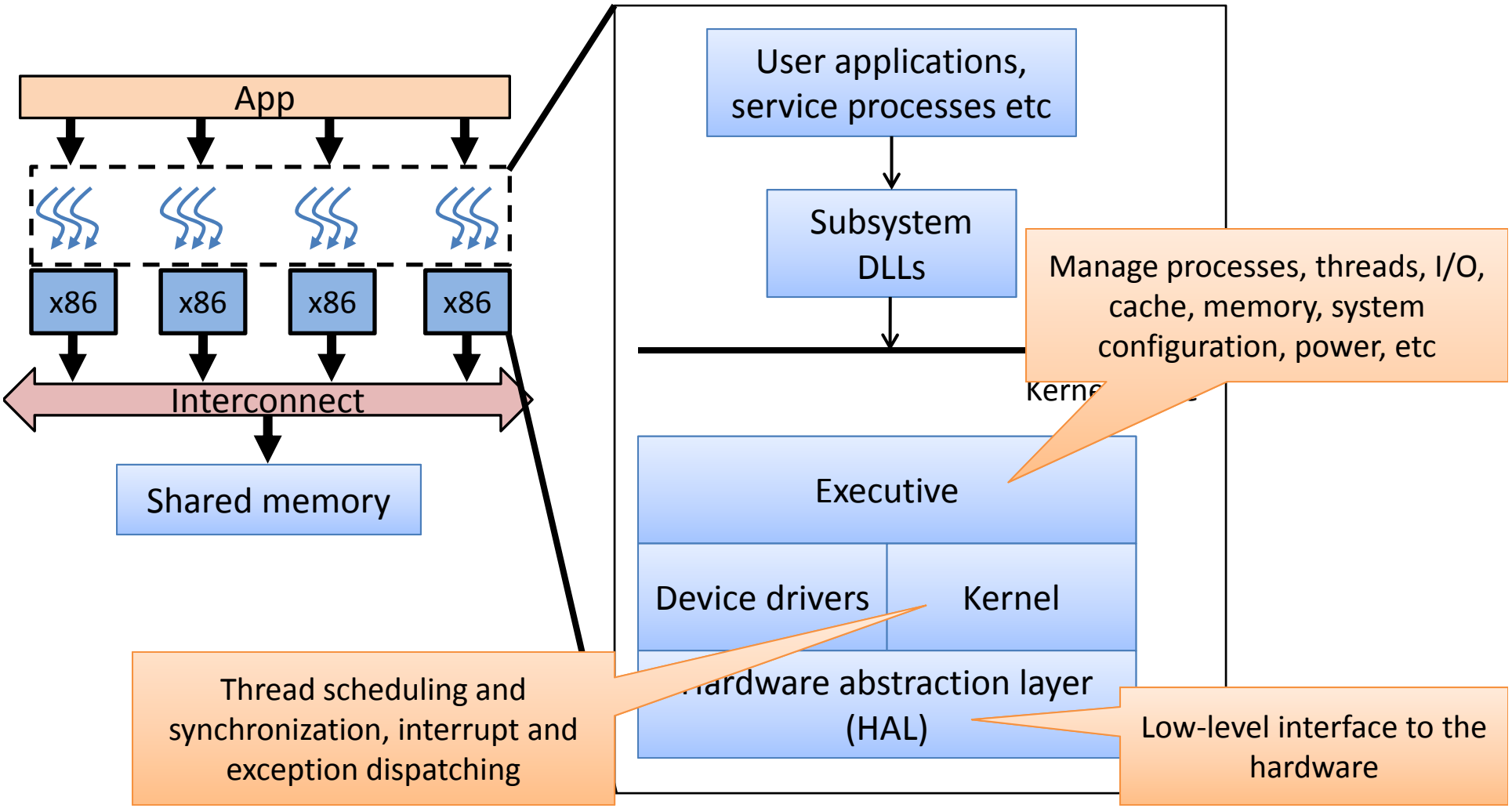
- Today's OS will not work with tomorrow's hardware.
  - Too slow as the number of cores increases
  - Can't handle the diversity of hardware
  - Can't keep up as hardware changes

Monolithic OS structure

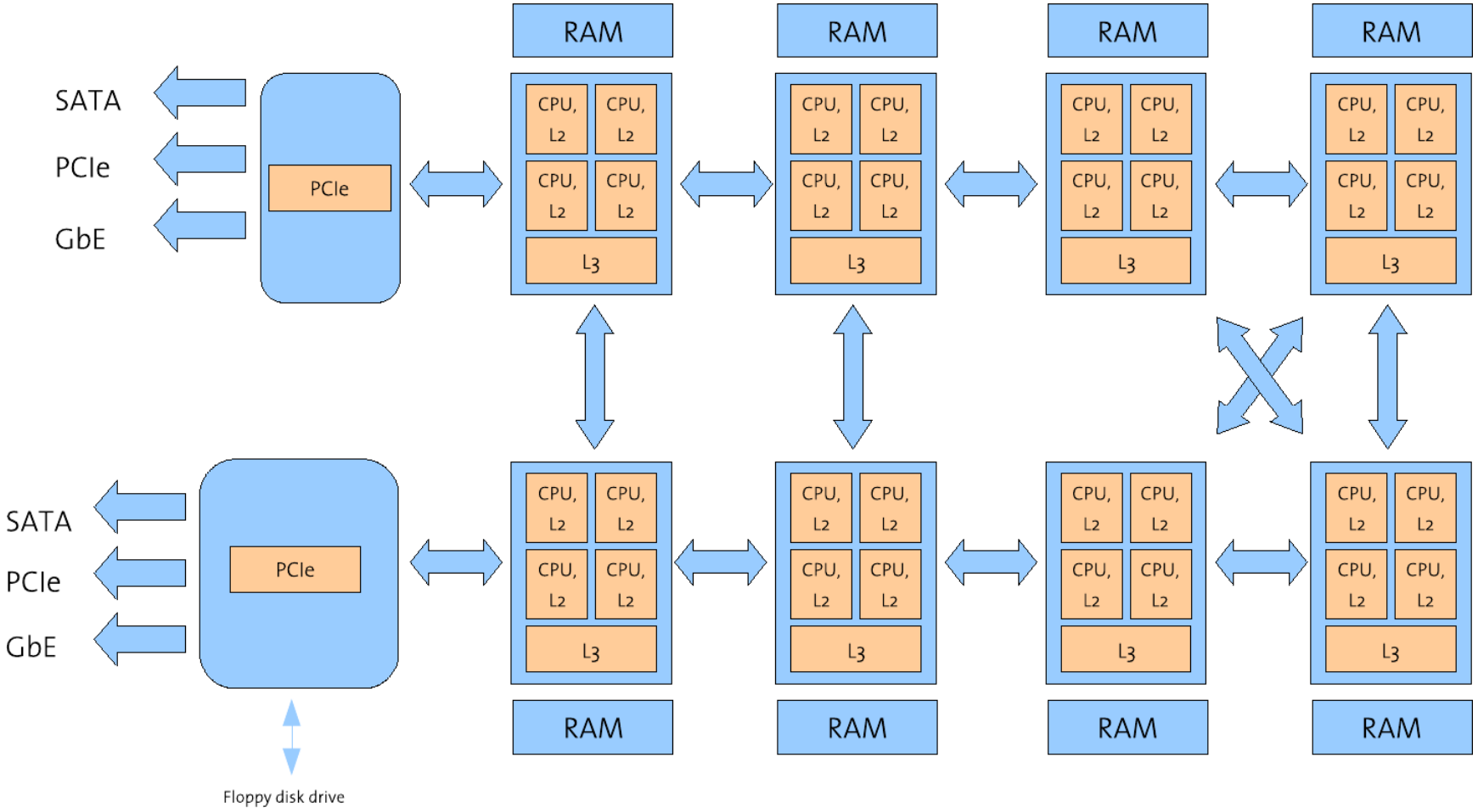
Systems@ETH Zürich



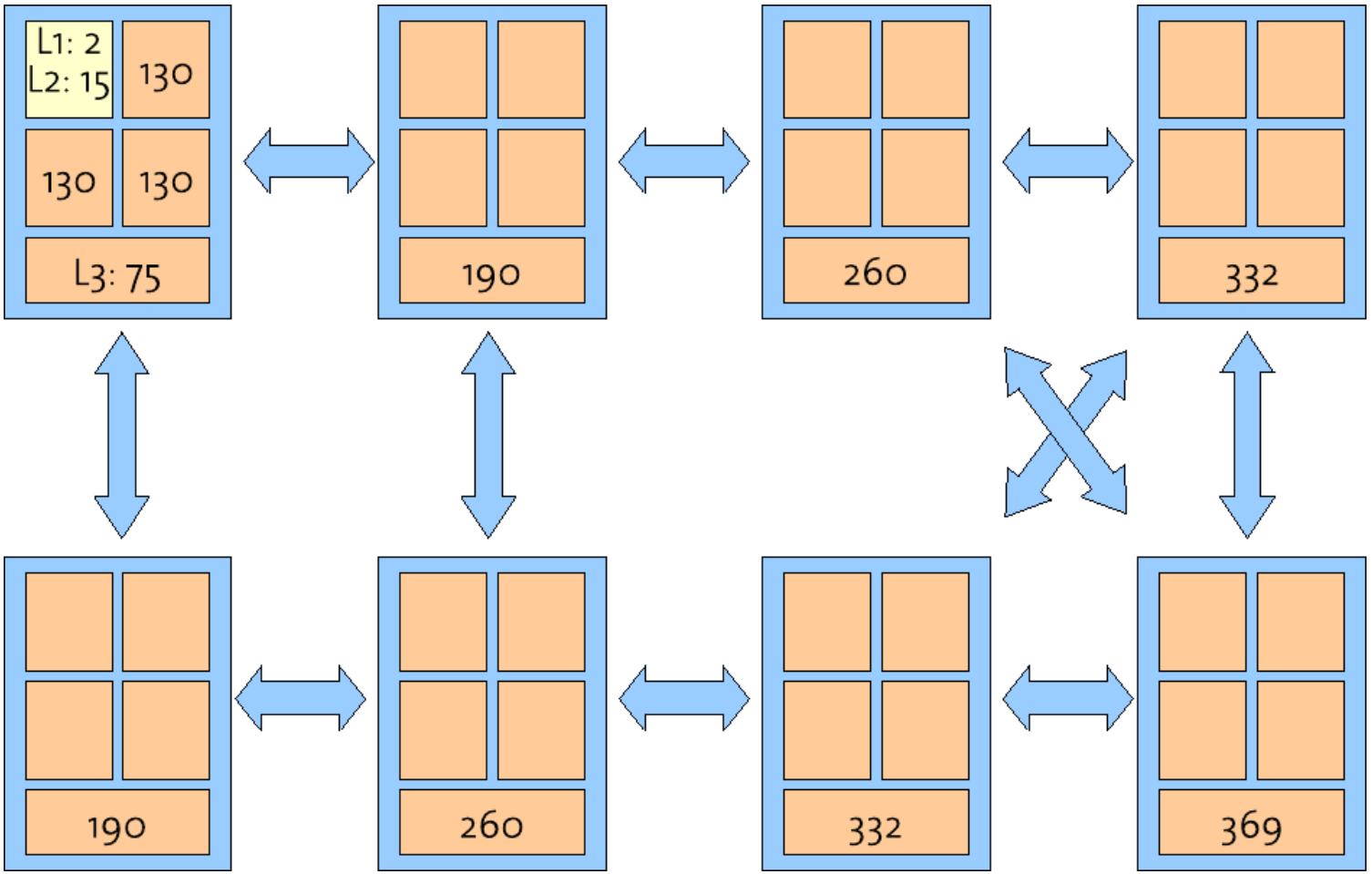
Windows: example monolithic OS



Cost of communication: 8-socket 32-core AMD Barcelona



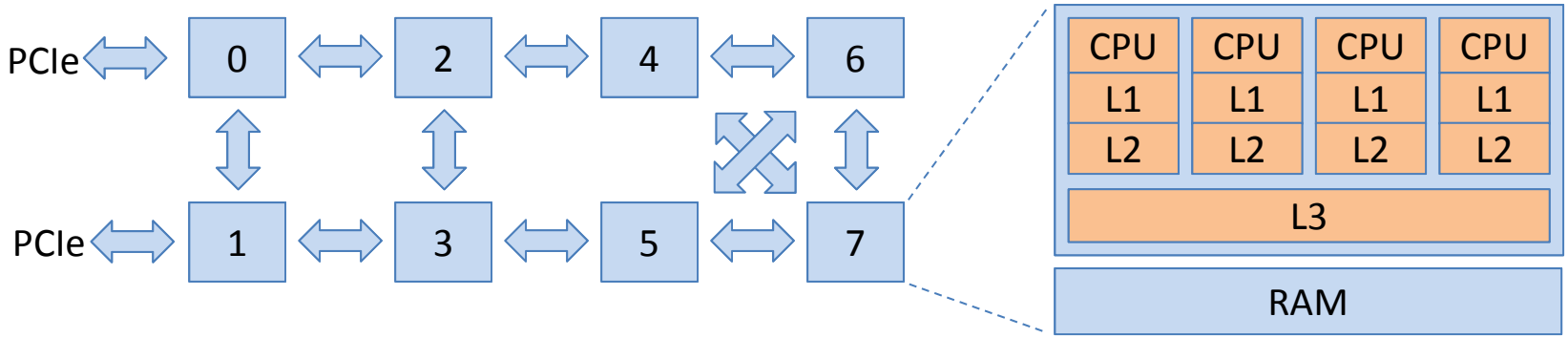
Memory access latency



Communication latencies

Systems@ETH Zürich

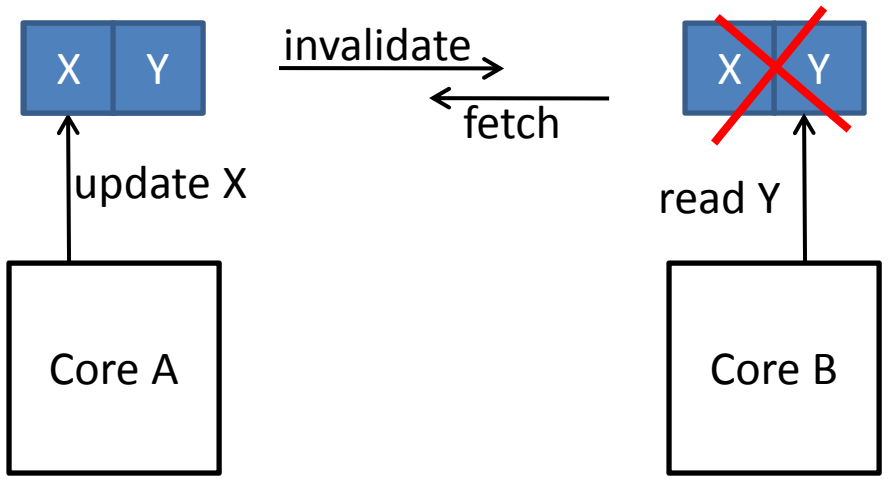
Example: 8 * quad-core AMD Opteron



Access	cycles	normalized to L1	per-hop cost
L1 cache	2	1	-
L2 cache	15	7.5	-
L3 cache	75	37.5	-
Other L1/L2	130	65	-
1-hop cache	190	95	60
2-hop cache	260	130	70

False sharing

- Two unrelated shared variables are located in the same cache line
- Accessing the variables on different processors causes the entire cache line to be exchanged between the processors



The cost of shared data

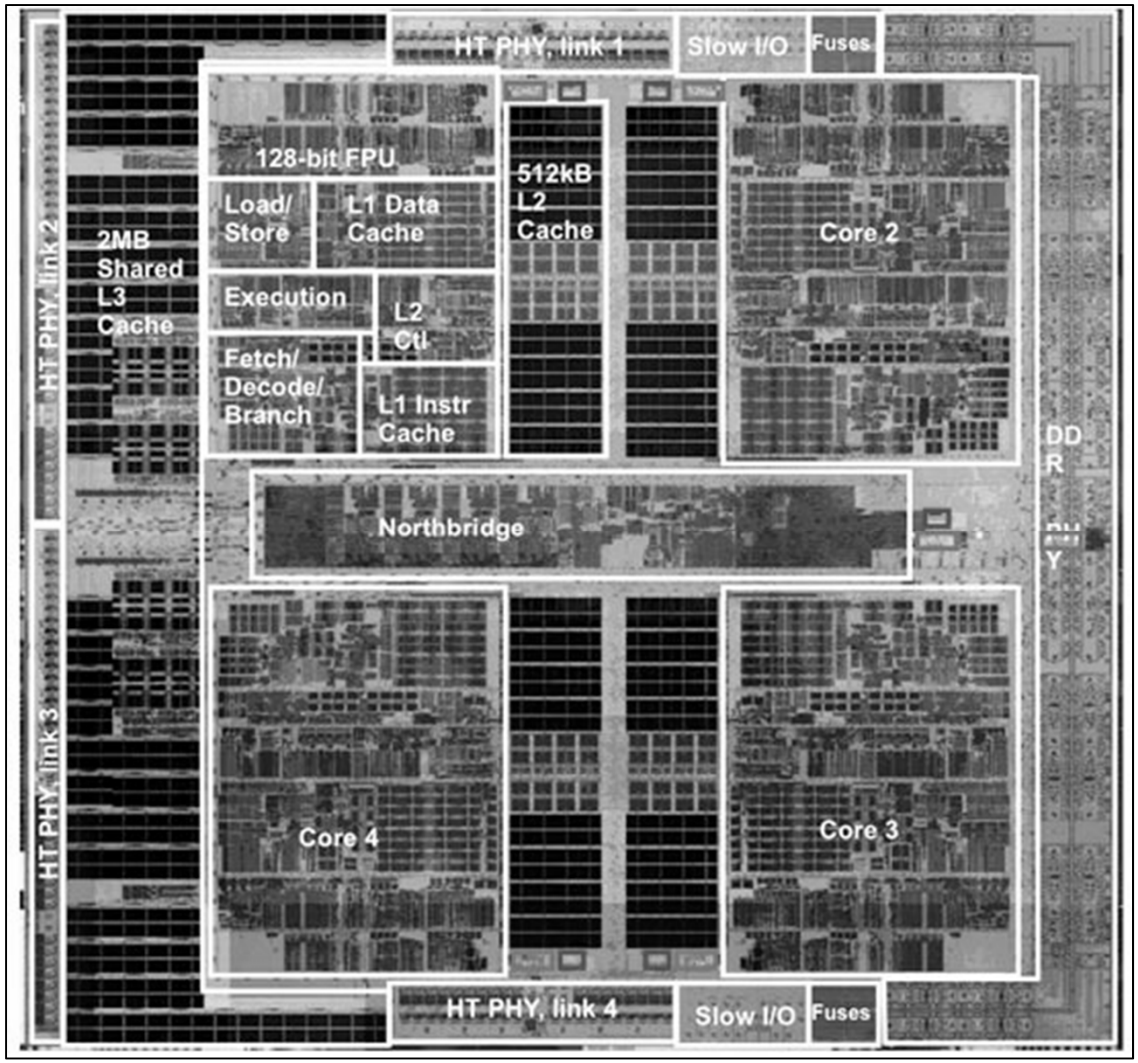
- Generates unnecessary interconnect traffic and processor stalls
- Need to lay out data structures very carefully
  - False sharing $\Rightarrow$ serialization
  - Can (often does) outweigh theoretical complexity!
- This fine-tuning increases:
  - code complexity and engineering effort
  - reduces portability to other hardware

Hardware diversity

- Hardware is changing faster than system software
  - Engineering effort to fix scaling problems is becoming overwhelming
- Hardware is already diverse
  - Can't tune OS design to any one machine architecture
- Cores will not all be the same
  - Different performance characteristics
  - Different instruction set variants
  - Different architectures (GPUs, NICs, etc.)

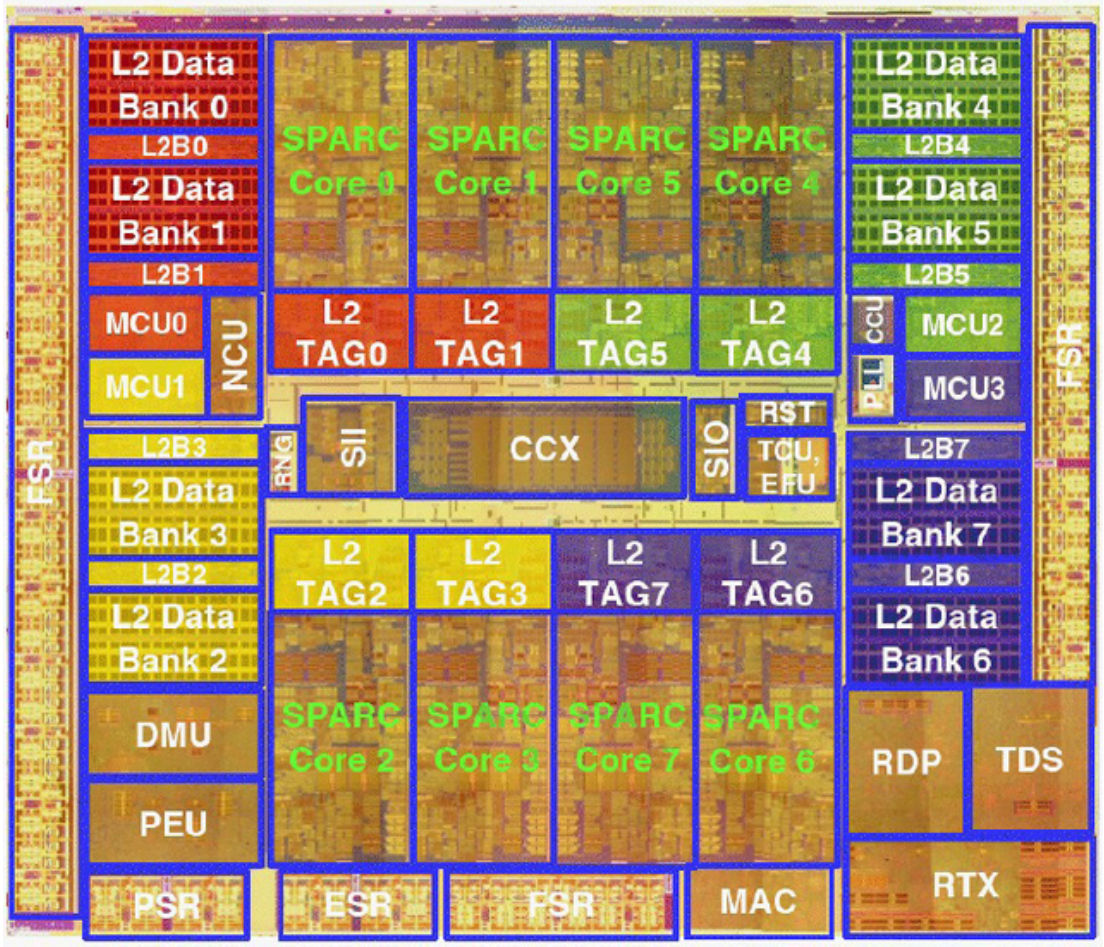
From AMD Barcelona...

Systems@ETH Zürich



… to Sun Niagara-2

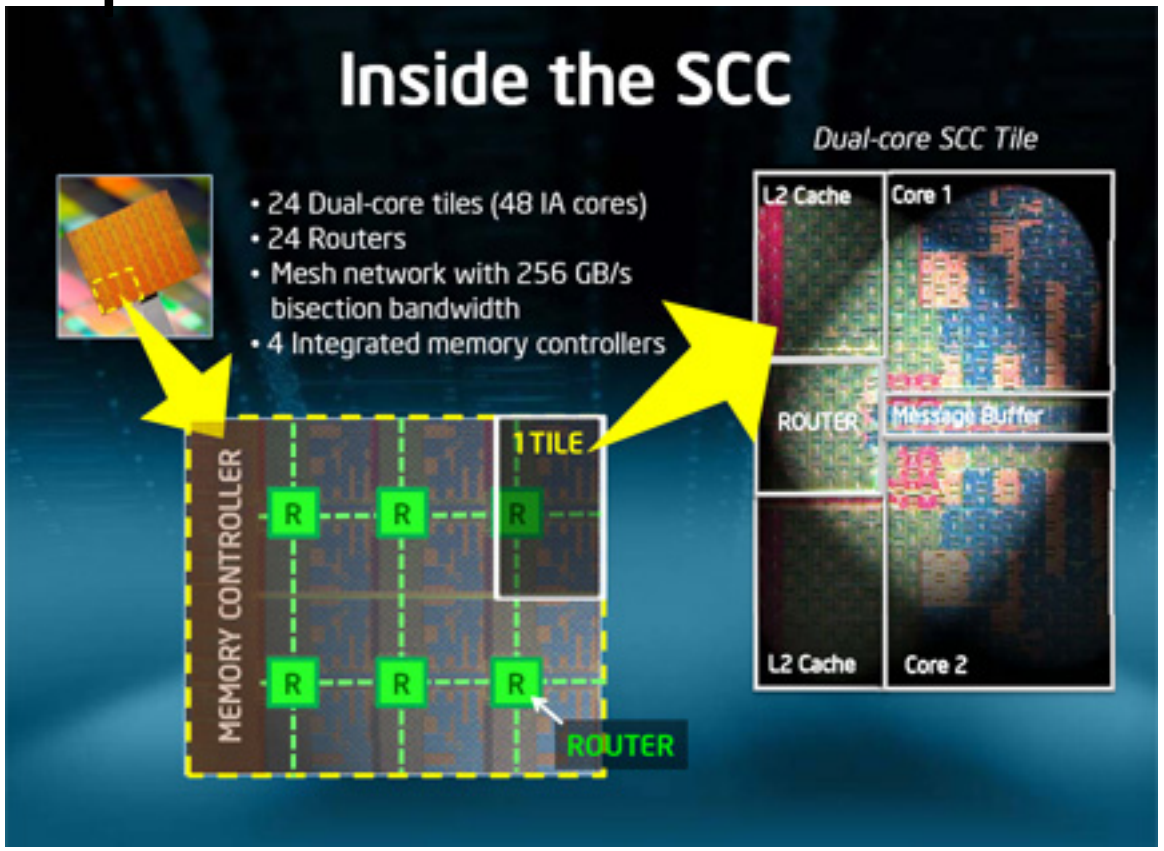
Systems@ETH Zürich



Intel Single-chip Cloud Computer

Systems@ETH Zürich

Announced December 2009



"In a sense, the SCC is a microcosm of cloud datacenter. Each core can run a separate OS and software stack and act like an individual compute node that communicates with other compute nodes over a packet-based network."

Summary:
Modern operating systems are

- Large, multi-threaded shared-memory programs
  - Tuned to particular architectural assumptions
  - Very different to most computational workloads!
- Not designed as scalable, parallel programs
  - Hard to scale (too many bottlenecks)
  - Hard to adapt to new hardware
    - And there's too much of it!
  - Don't work at all on non-cache coherent machines

Section 3

THE MULTIKERNEL MODEL

What is “The Multikernel Model”?

- A reference model for operating systems on ***multicore*** computers
- Premise:

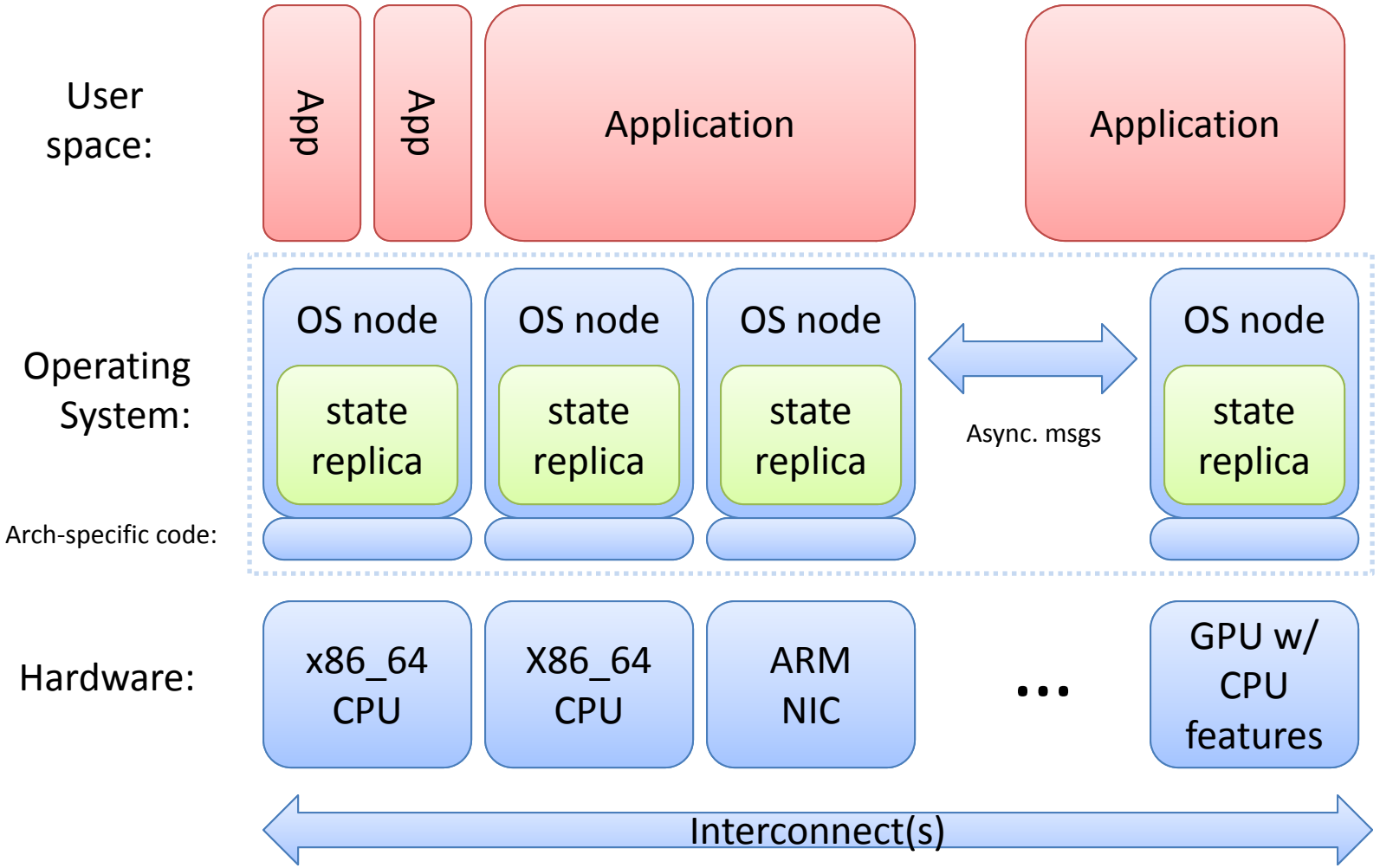
  Computer hardware looks increasingly like a network...

  ... so the operating system should look like a ***distributed system***

Properties of a distributed system

- Multiple independent and concurrently executing nodes
- Communication by message-passing
- Replicated state
- Leading to:
  - Scalability
  - Fault tolerance
  - Fast local operations
  - Explicit sharing

The multikernel model



Explicit inter-core communication

- All communication with messages
- Decouples system structure from inter-core communication mechanism
- Communication patterns explicitly expressed
- Better match for future hardware
  - Naturally supports heterogeneous cores, non-coherent interconnects (PCIe)
  - with cheap explicit message passing
  - without cache-coherence
- Allows split-phase operations

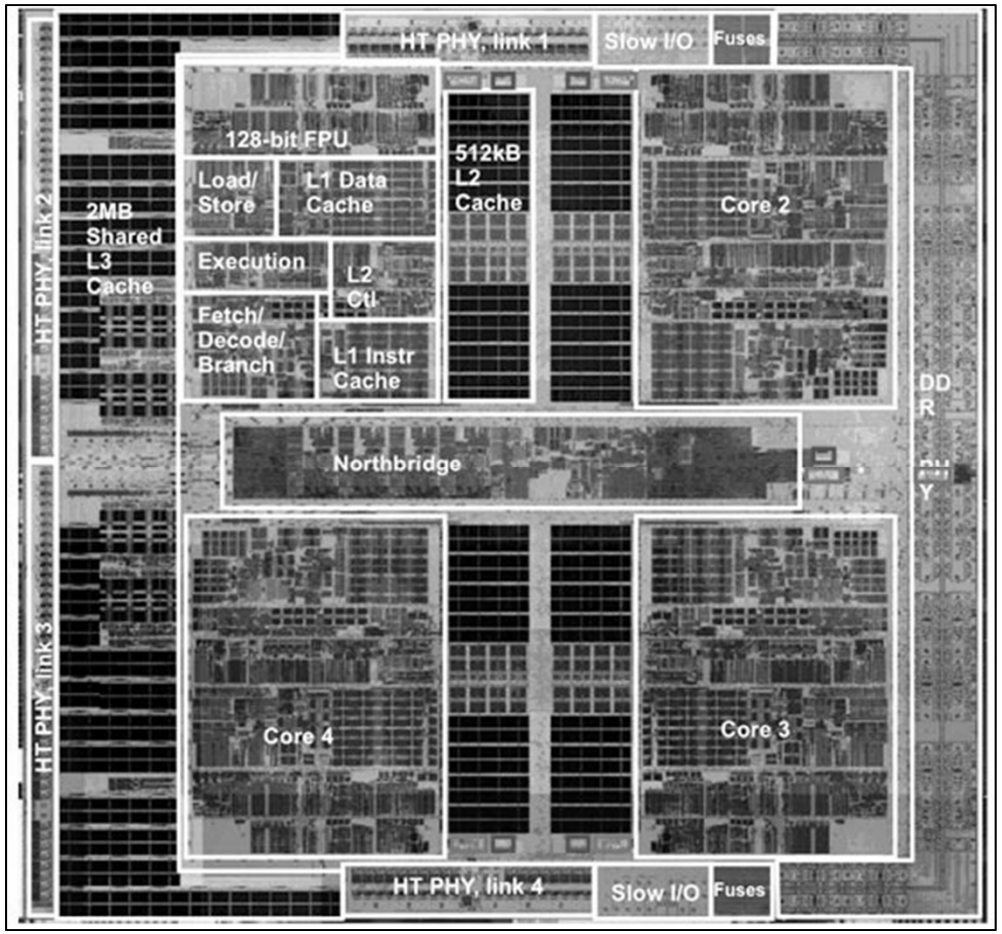
Message passing vs. shared memory

- Structures are duals (Lauer & Needham, 1978)
  - Choice depends on machine architecture
- Shared memory has been favoured until now
- What are the trade-offs?
  - Depends on data size and amount of contention

Experiment: shared memory vs message-passing

Systems@ETH Zürich

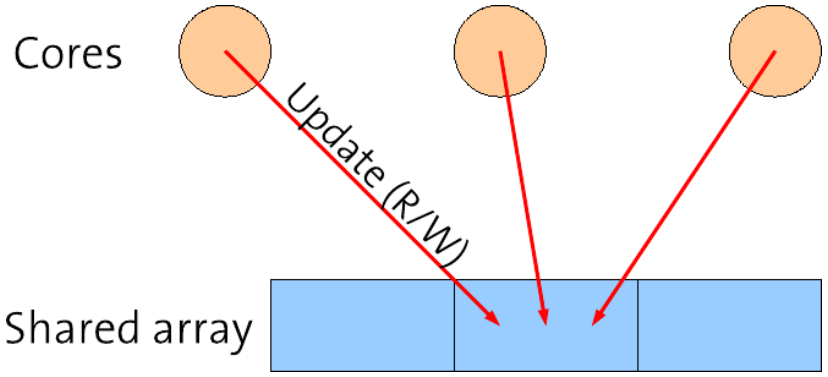
- Measure costs (latency per operation) of updating a shared data structure
- Hardware: 4*quad-core AMD Opteron



Message passing vs. shared memory: experiment

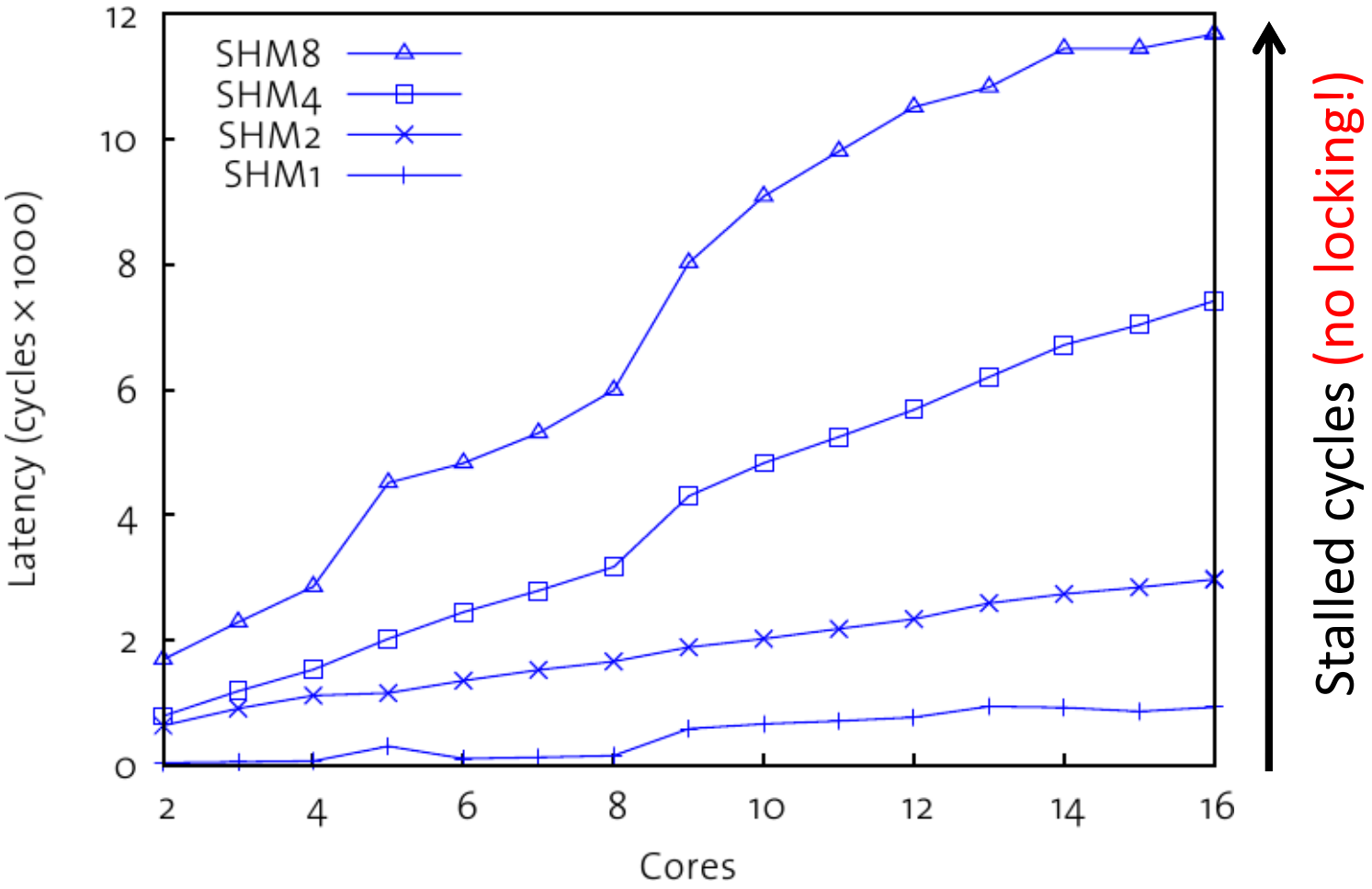
Systems@ETH Zürich

- Shared memory (move the data to the operation):
- Each core updates the same memory locations (no locking)
- Cache-coherence protocol migrates modified cache lines
  - Processor stalled while line is fetched or invalidated
  - Limited by latency of interconnect round-trips
  - Performance depends on data size (cache lines) and contention (number of cores)



Shared memory results

4x4-core AMD system

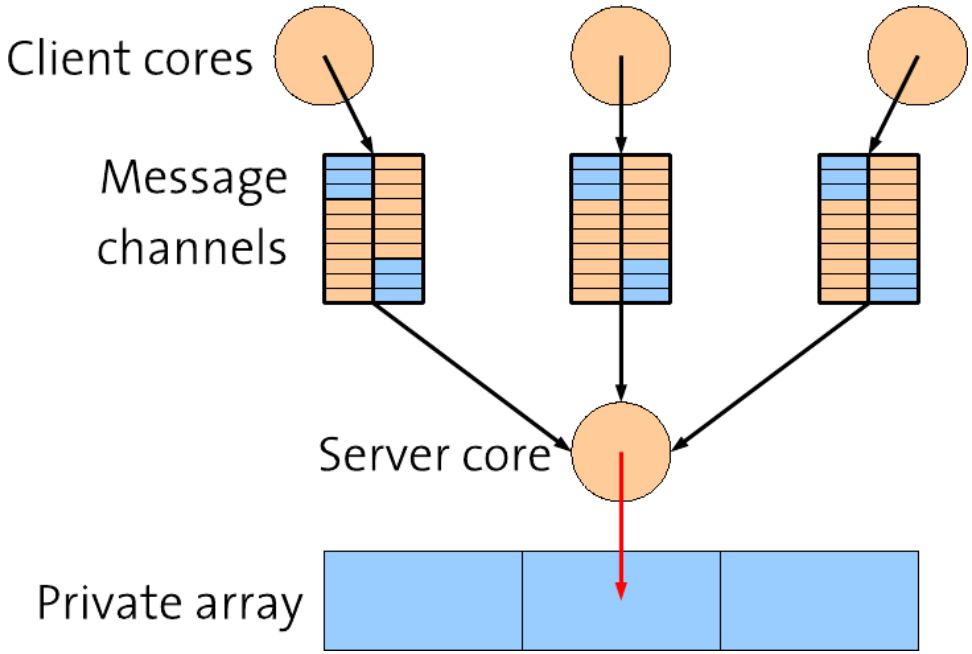


Message passing vs. shared memory: experiment

Systems@ETH Zürich

Message passing (move the operation to the data):

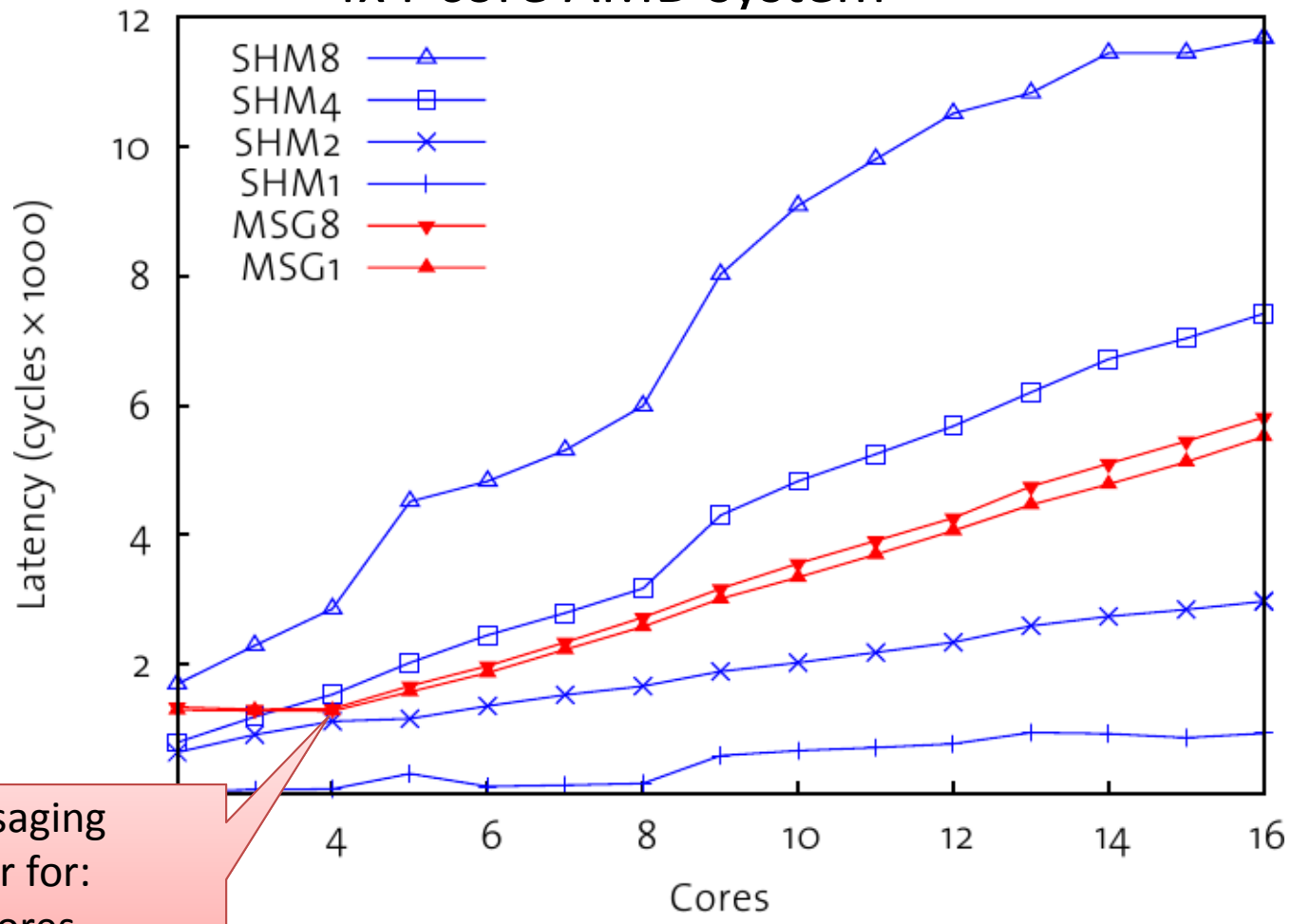
- A single server core updates the memory locations
- Each client core sends RPCs to the server
  - Operation and results described in a single cache line
  - Block while waiting for a response (in this experiment)



Message passing vs. shared memory: tradeoff

Systems@ETH Zürich

4x4-core AMD system

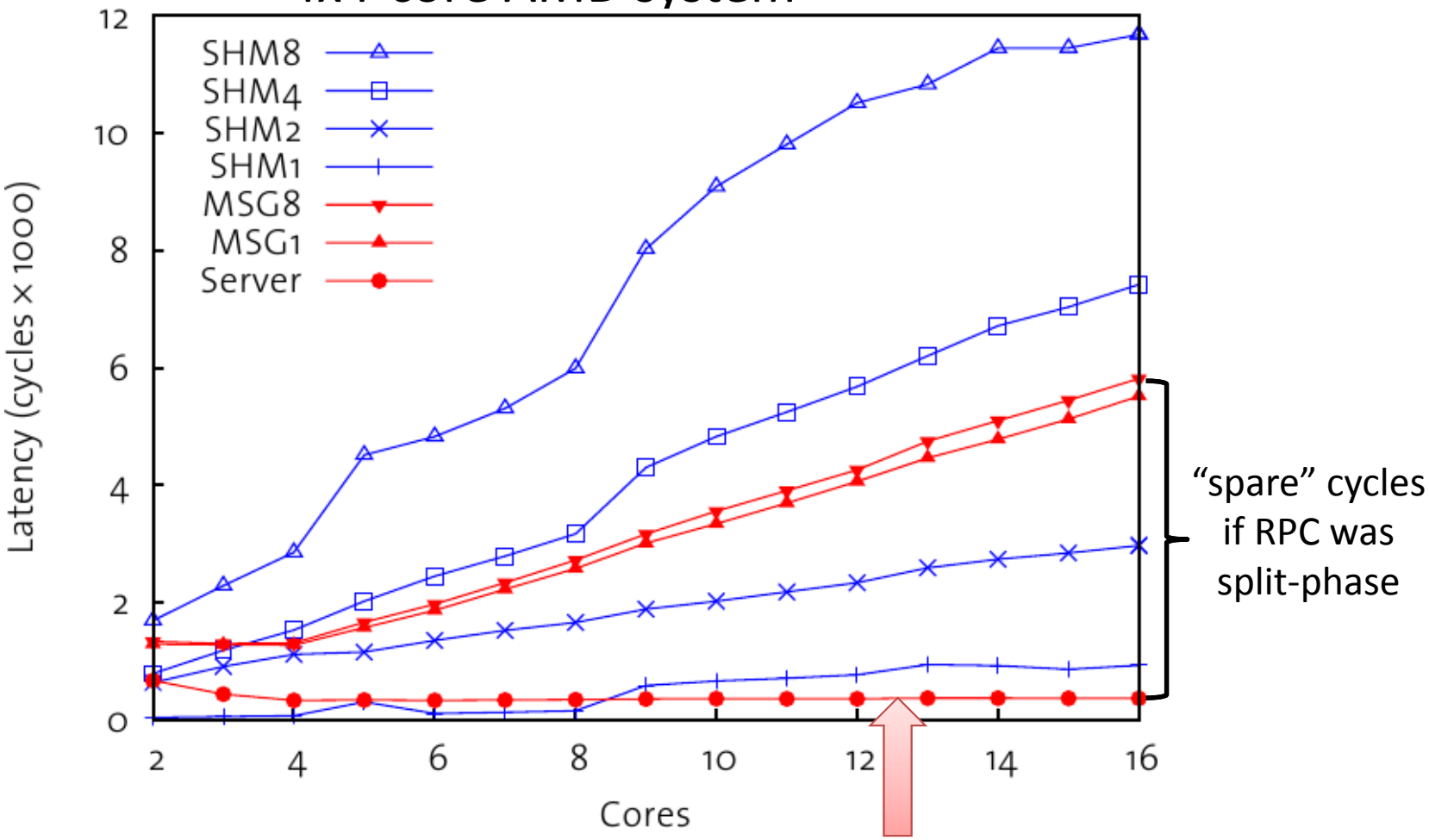


Messaging faster for:
≥ 4 cores
≥ 4 cache lines

Message passing vs. shared memory: tradeoff

Systems@ETH Zürich

4x4-core AMD system



"spare" cycles if RPC was split-phase

Actual cost of update at server

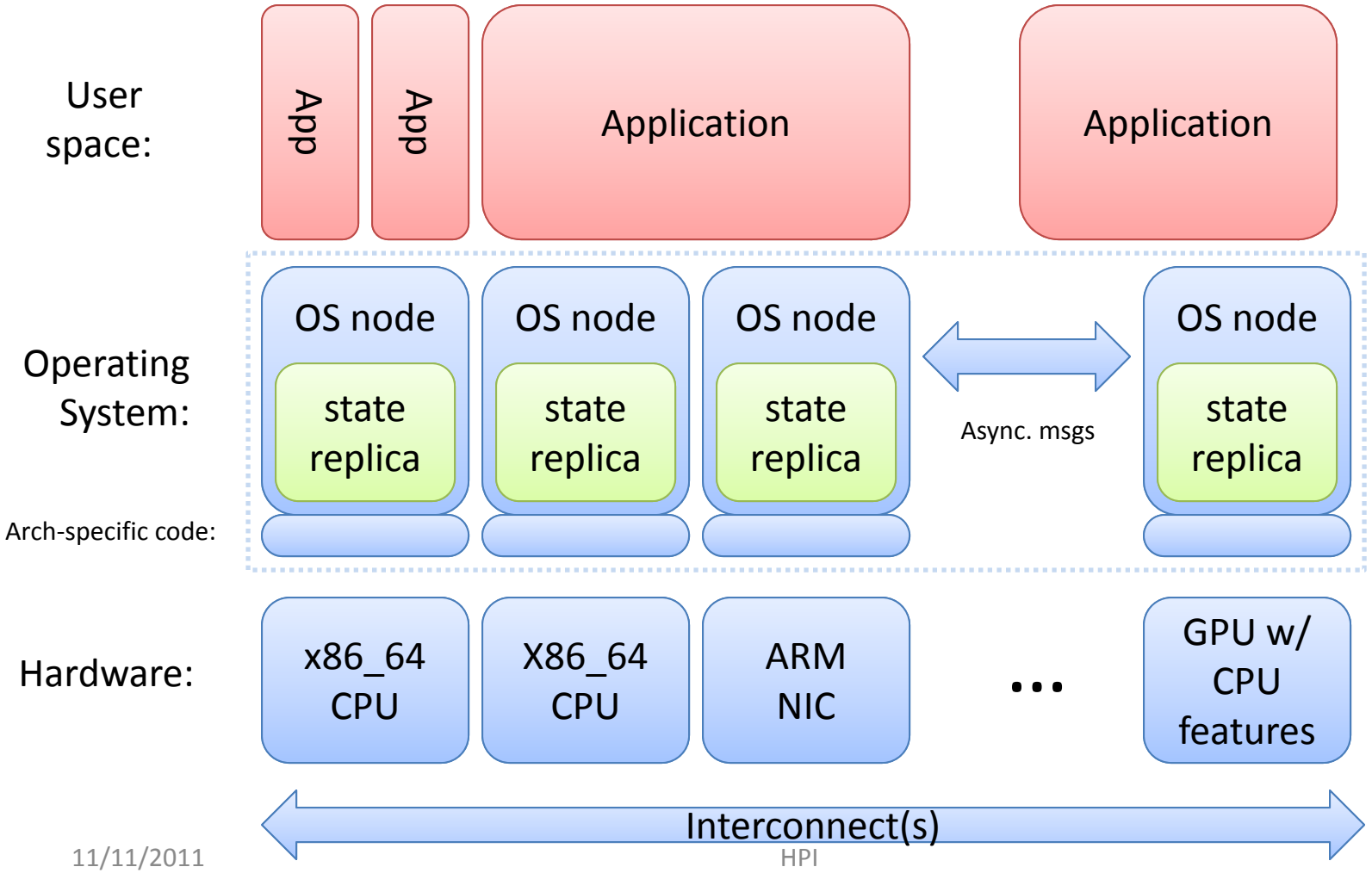
Hardware-neutral structure

- Separate OS structure from hardware
- Only hardware-specific parts:
  - Message transports (highly optimised / specialised)
  - CPU / device drivers
- Adaptability to changing performance characteristics
  - Late-bind protocol and message transport implementations

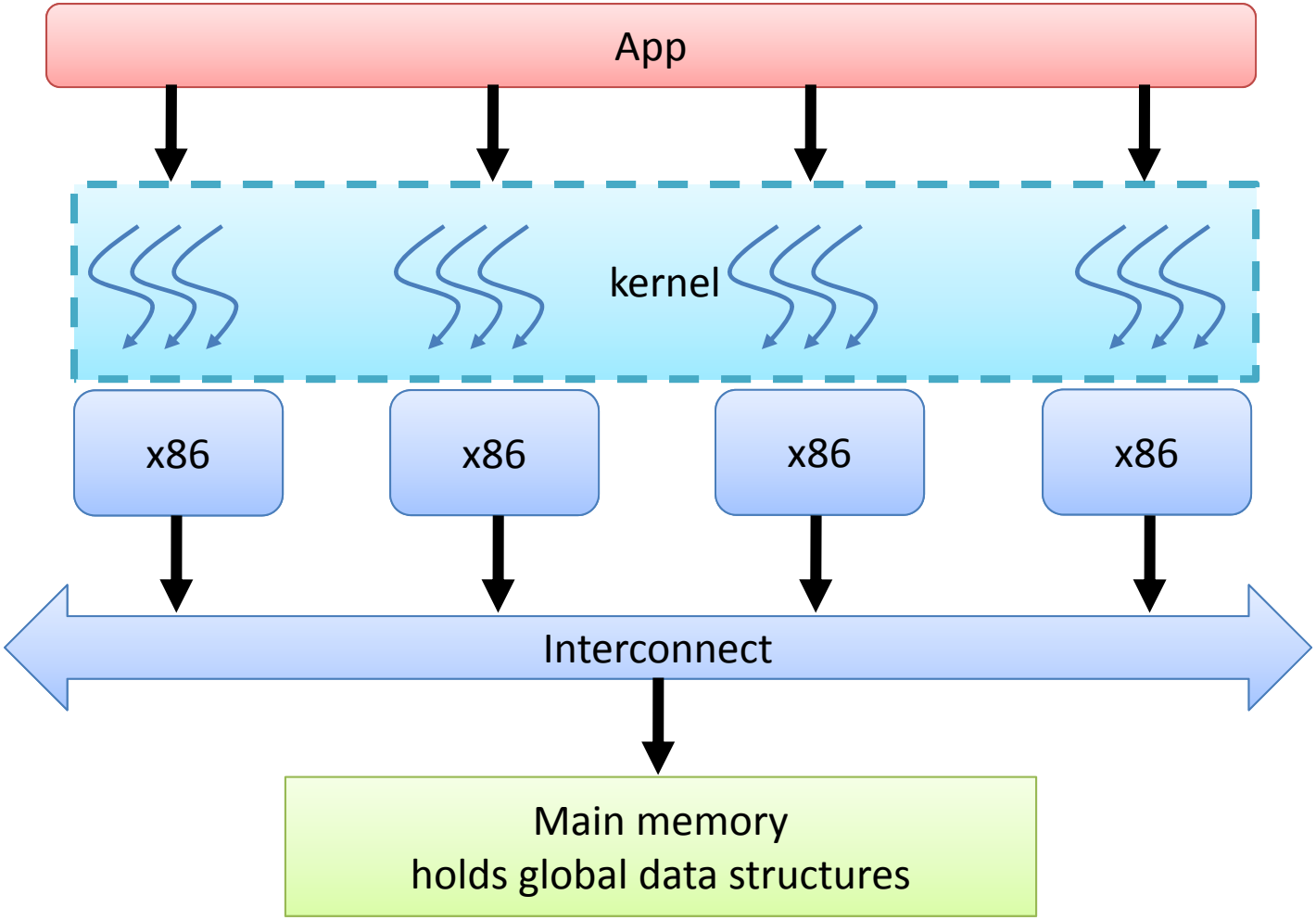
Replicate common state

- Potentially-shared state accessed *as if it were a local replica*
  - Scheduler queues, process control blocks, etc.
  - Required by message-passing model
- Naturally supports domains that do not share memory
- Naturally supports changes to the set of running cores
  - Hotplug, power management

The multikernel model

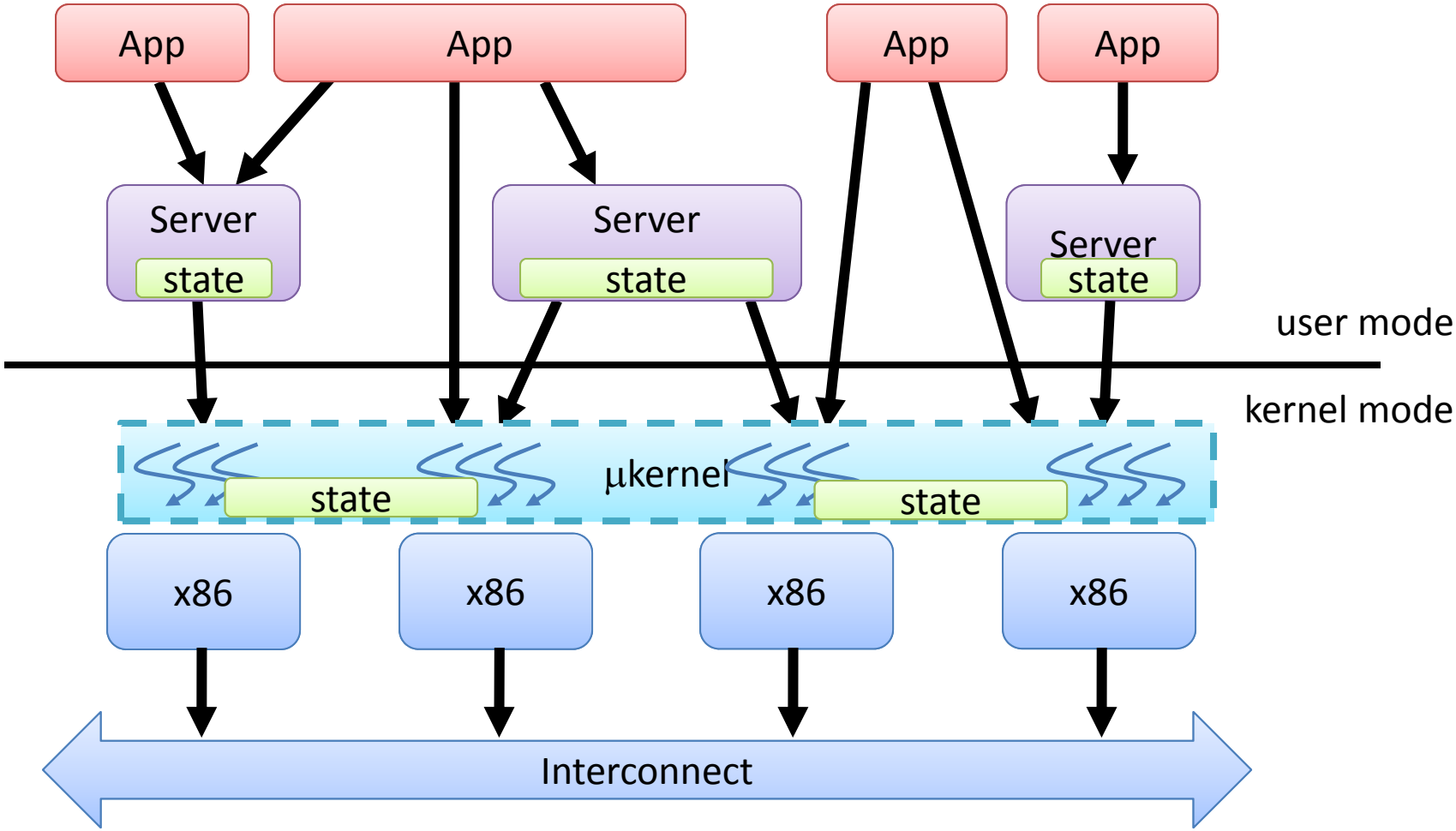


...vs a monolithic OS on multicore

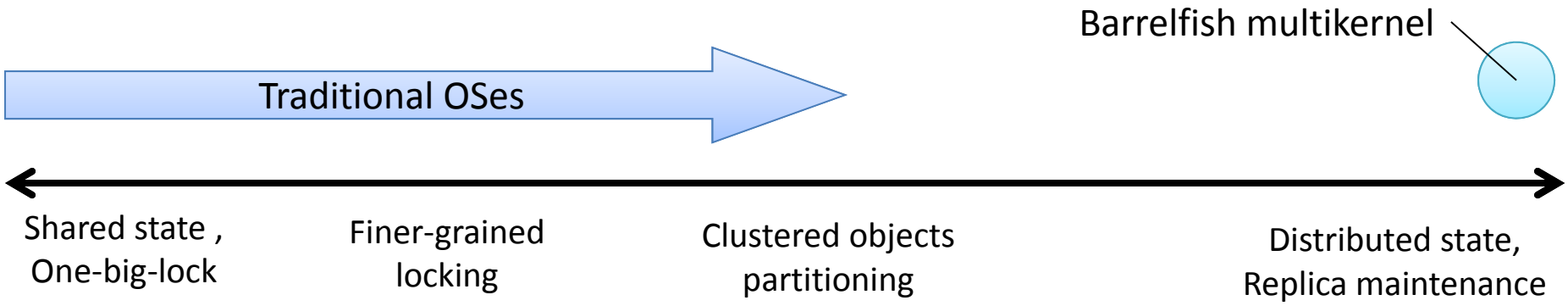


...vs a µkernel OS on multicore

Systems@ETH Zürich

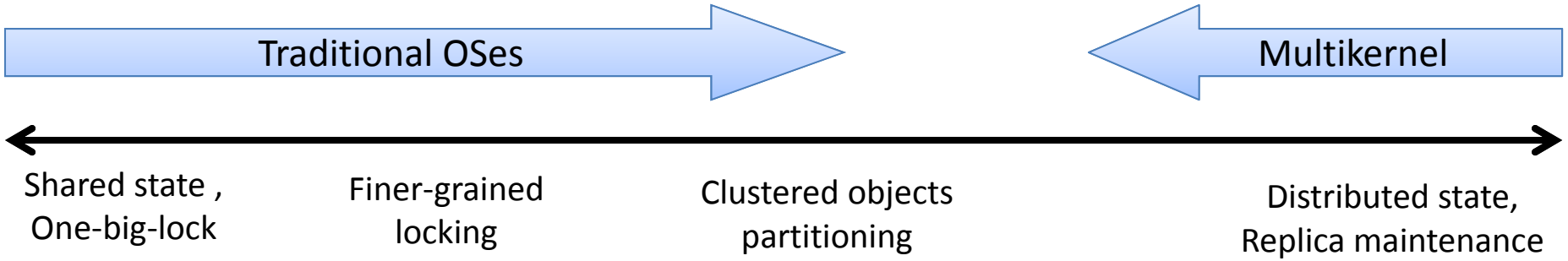


Replication vs sharing as the default



- Replicas used as an optimization in other systems

Replication vs sharing as the default



- Replicas used as an optimization in other systems
- In a multikernel, sharing is a local optimisation
  - Shared (locked) replica on closely-coupled cores
  - Only when faster, as *decided at runtime*
- Basic model remains split-phase messaging

Section 4

BARRELFISH

Barrelfish: our multikernel

Systems@ETH Zürich

- ETH Zurich + Microsoft Research
- Currently supports:
  - 32- and 64-bit x86
  - Intel Single-chip Cloud Computer
  - ARM (& Xscale)
  - Beehive (experimental softcore)
- Published 2009, available now
  - MIT open source licence

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Microsoft Research

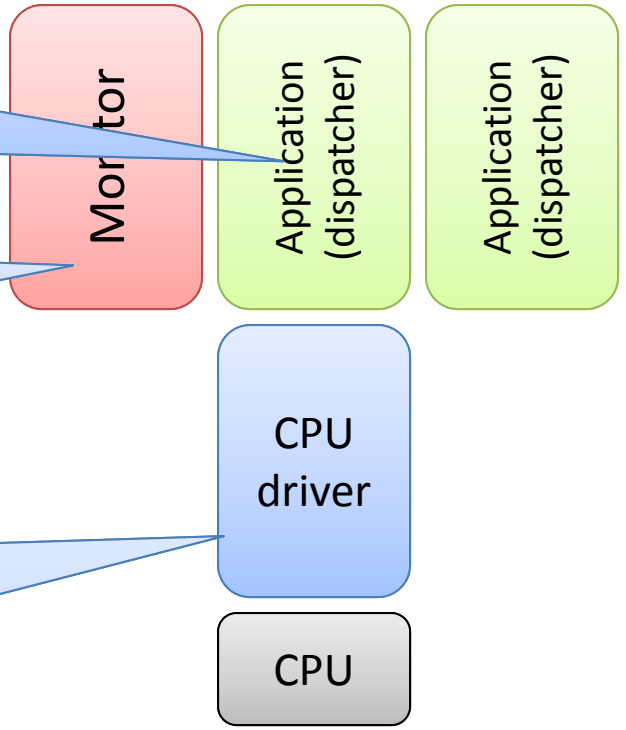
Per-core architecture

Systems@ETH Zürich

- Representative of app. on each core (including drivers and services)
- Upcalled from CPU driver
- Local thread scheduling
- Communicates with peer dispatchers

- User space (extra privilege)
- Communicates with other monitors
- Manages distributed operations
- Performs long-running operations

- Kernel space
- Manages CPU, MMU, APIC
- Multiplexes core btw. *dispatchers*
- Handles interrupts
- Implements protection via *capability* validation
- Single-threaded, non-preemptable



Examples of agreement protocols in Barrelfish

- Keeping TLBs consistent
  - Each core holds a cache of virtual memory mappings, known as the TLB
  - Requires 1-phase commit
- Keep the capability database consistent
  - Virtual memory protection is enforced with capabilities
  - The capability database is replicated on every core
  - Requires 2-phase commit
- If cores can sleep (eg to save power), we may need a group membership protocol and more sophisticated consensus algorithms

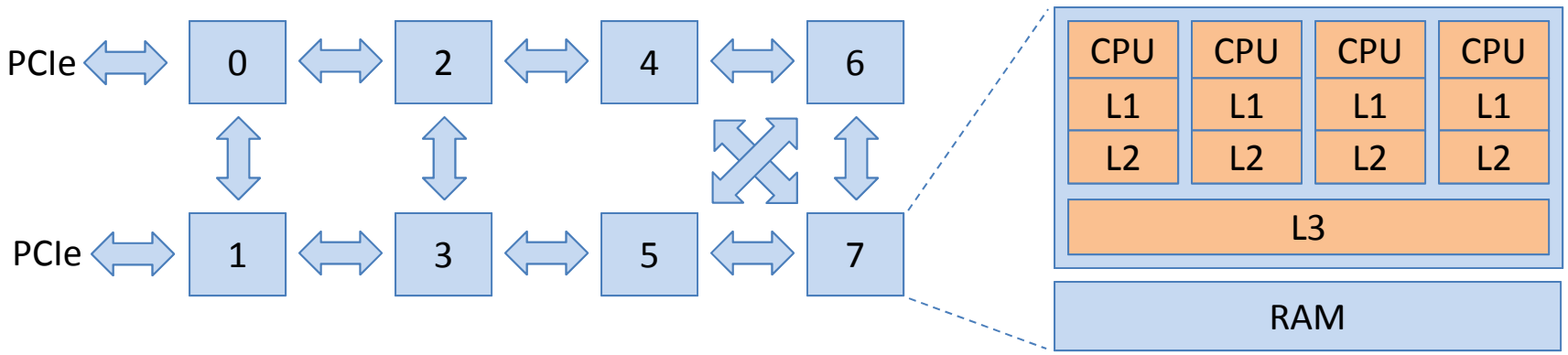
TLB shootdown

- When a mapping changes, the TLB must be flushed
  - On a multi-core machine, the TLB of every core that might contain the mapping must be flushed
- Requires global coordination (on every OS)
  - Send a message to every core with a mapping
  - Wait for acks (must be short!)
- Linux/Windows:
  - Send IPI (interprocessor interrupt)
  - Spin on shared ack count
- Barrelfish:
  - Monitor runs 1-phase commit protocol to remote cores
  - Can exploit knowledge of interconnect topology to improve performance

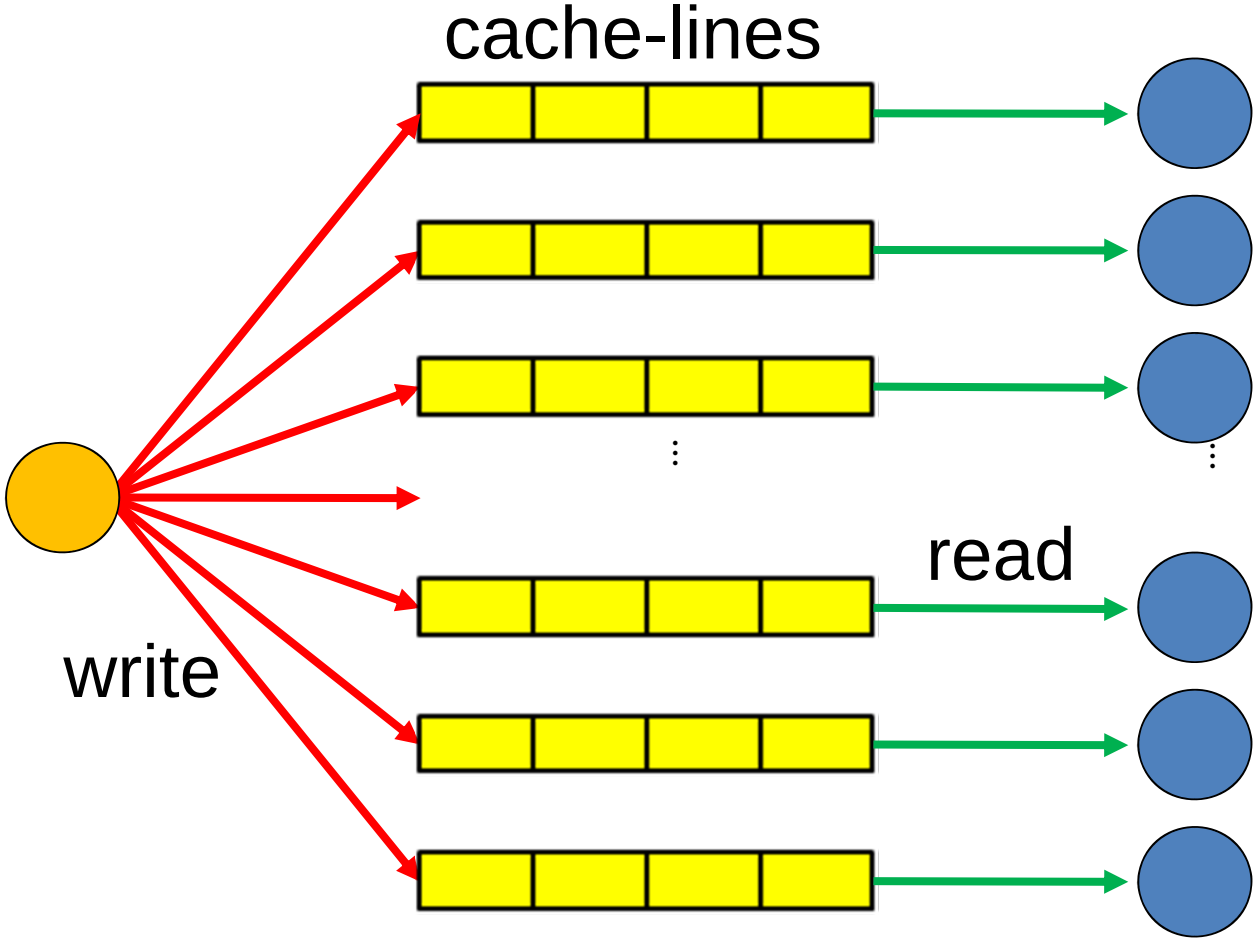
Case study for TLB shootdown

Systems@ETH Zürich

Hardware: 8 * quad-core AMD Opteron

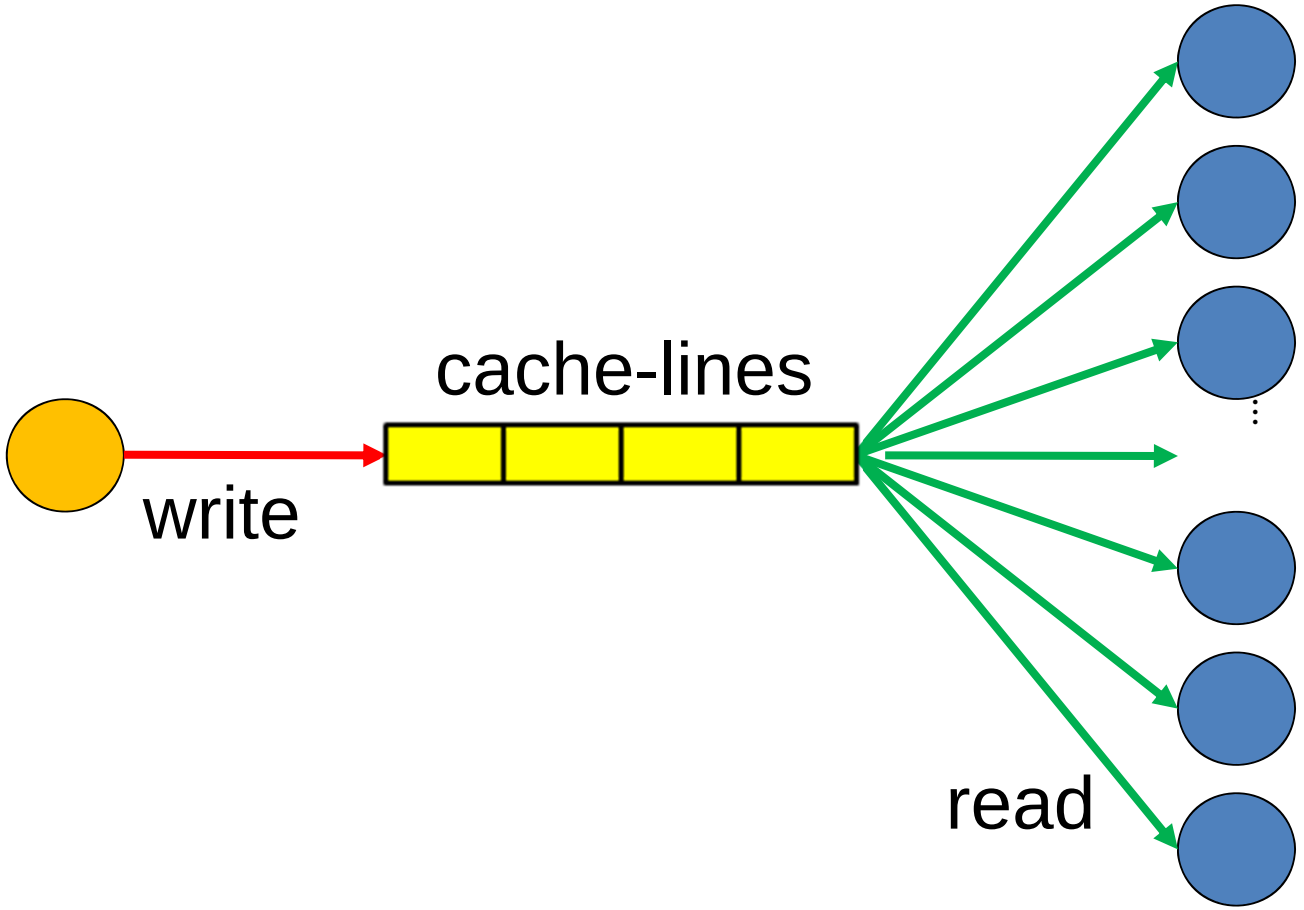


TLB shootdown: n*unicast



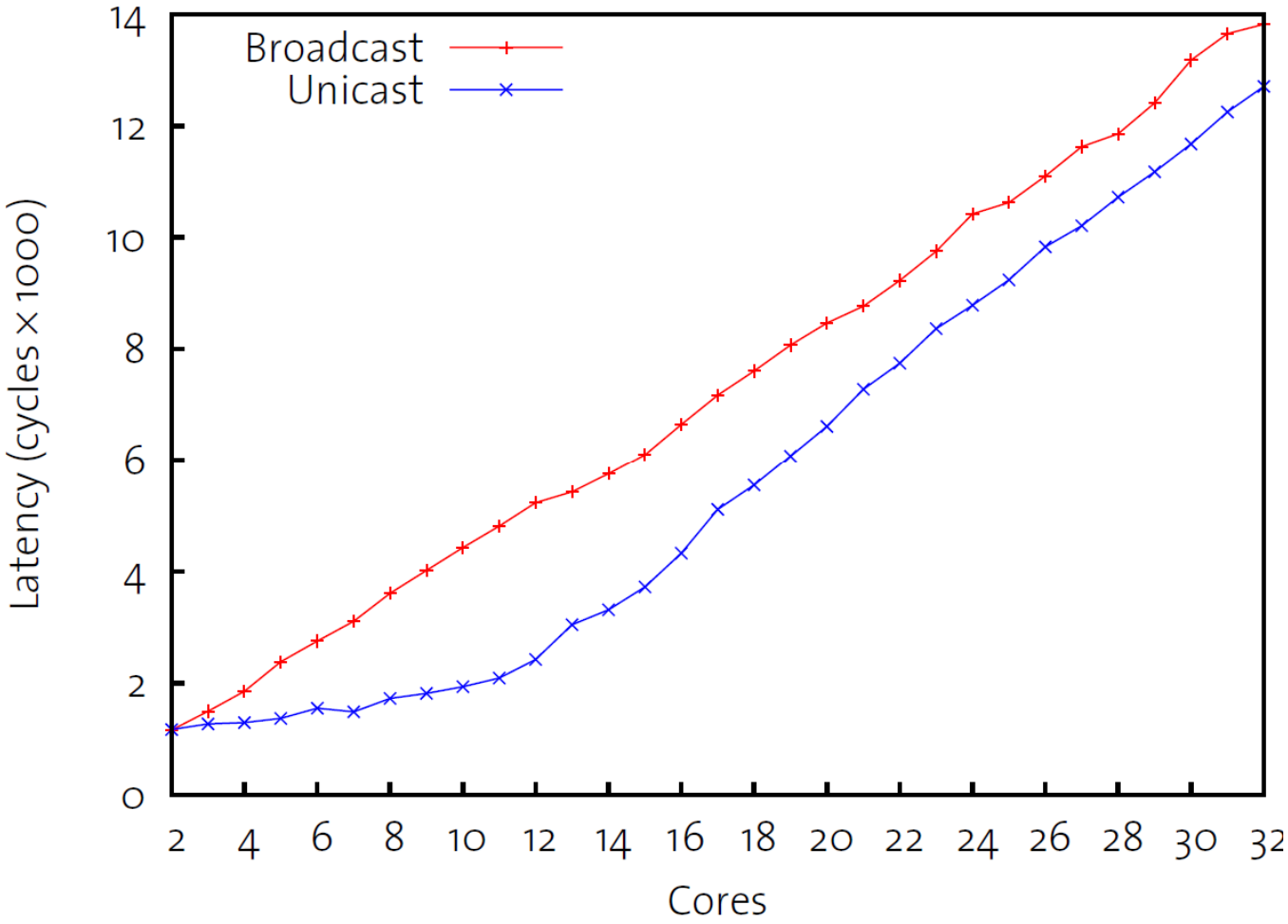
TLB shootdown: 1*broadcast

Systems@ETH Zürich



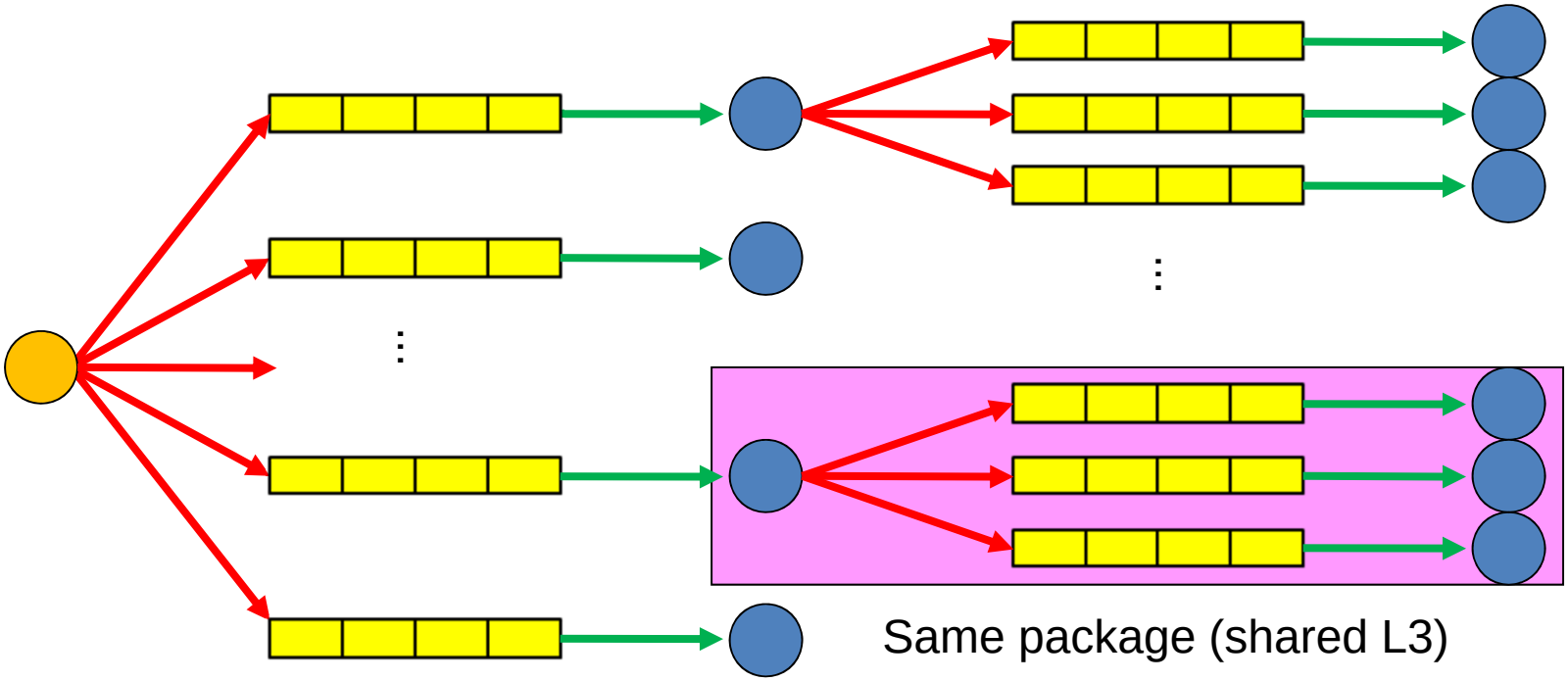
Messaging costs

Systems@ETH Zürich

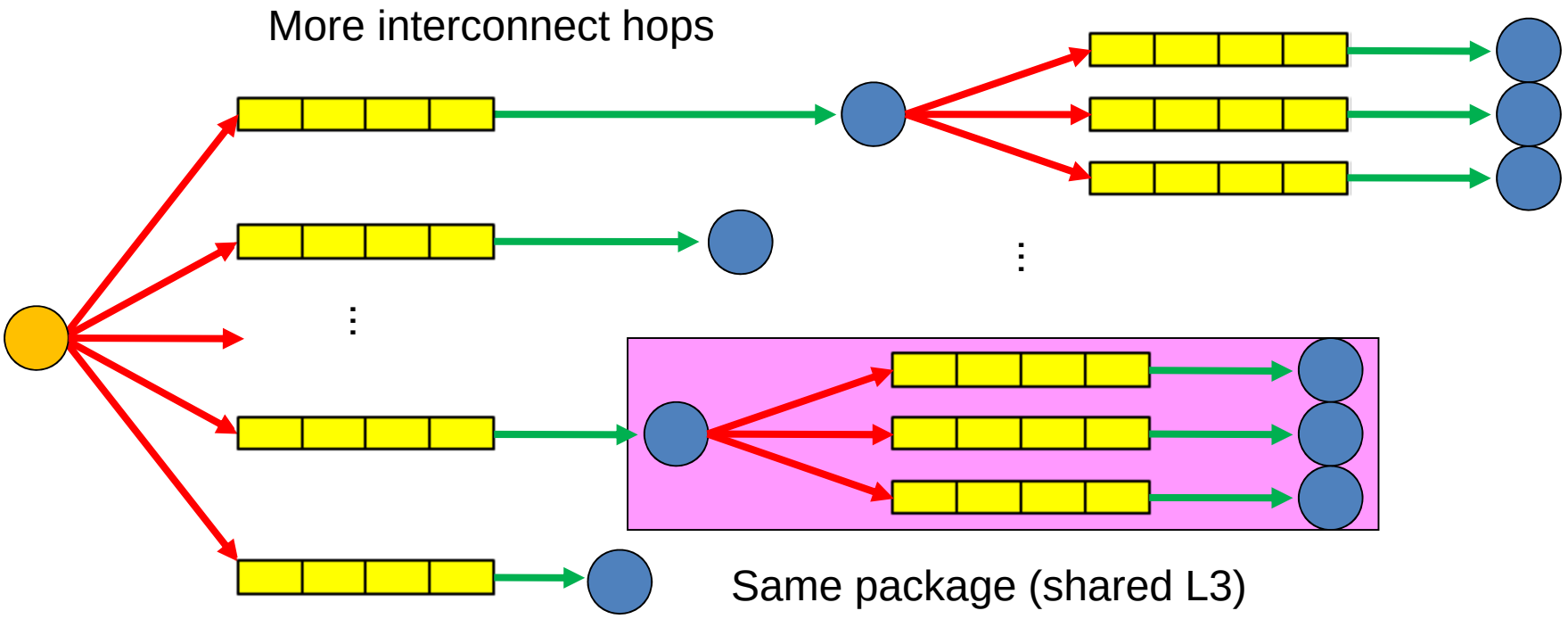


TLB shootdown: multicast

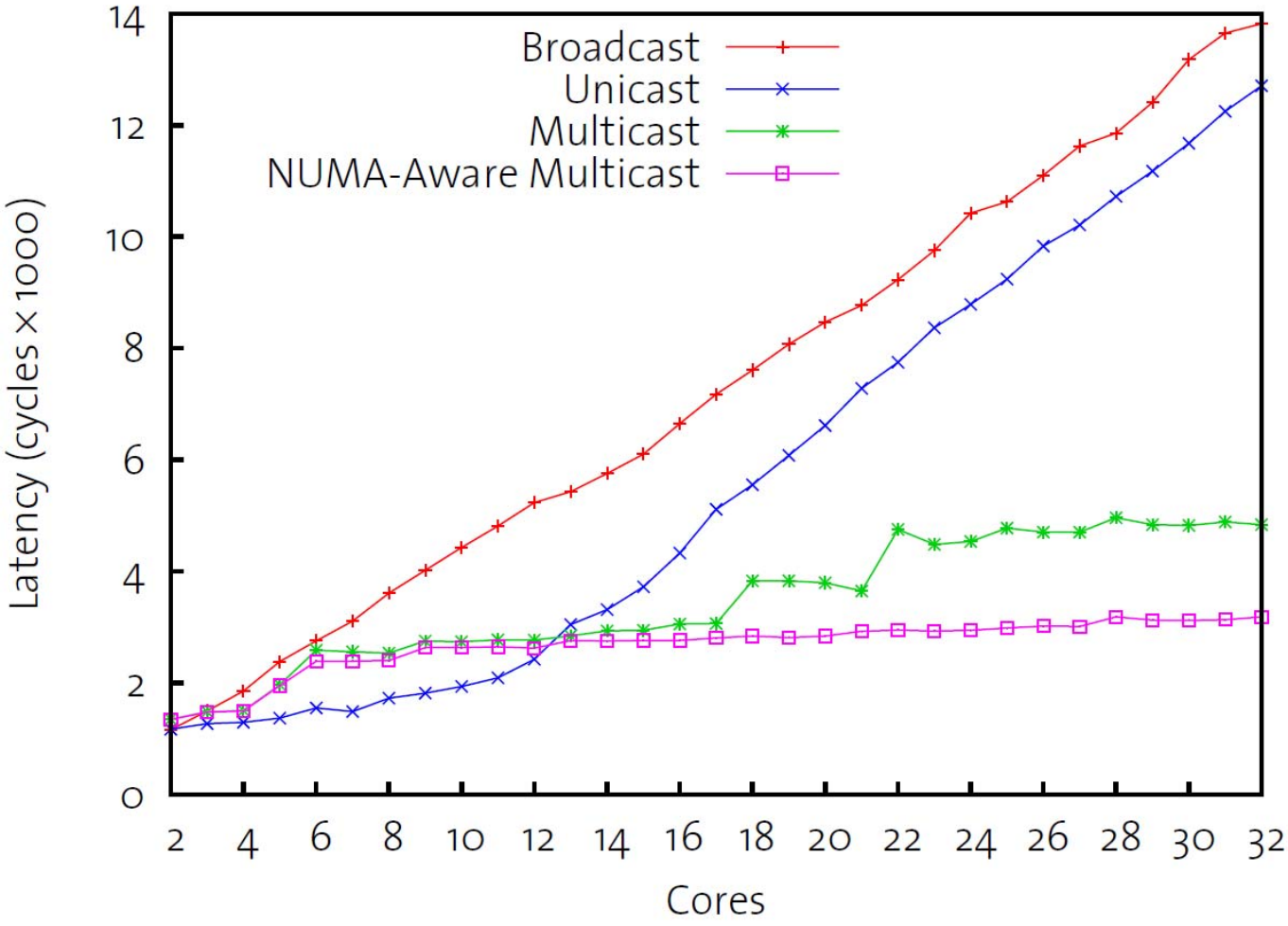
Systems@ETH Zürich



TLB shootdown: NUMA-aware multicast



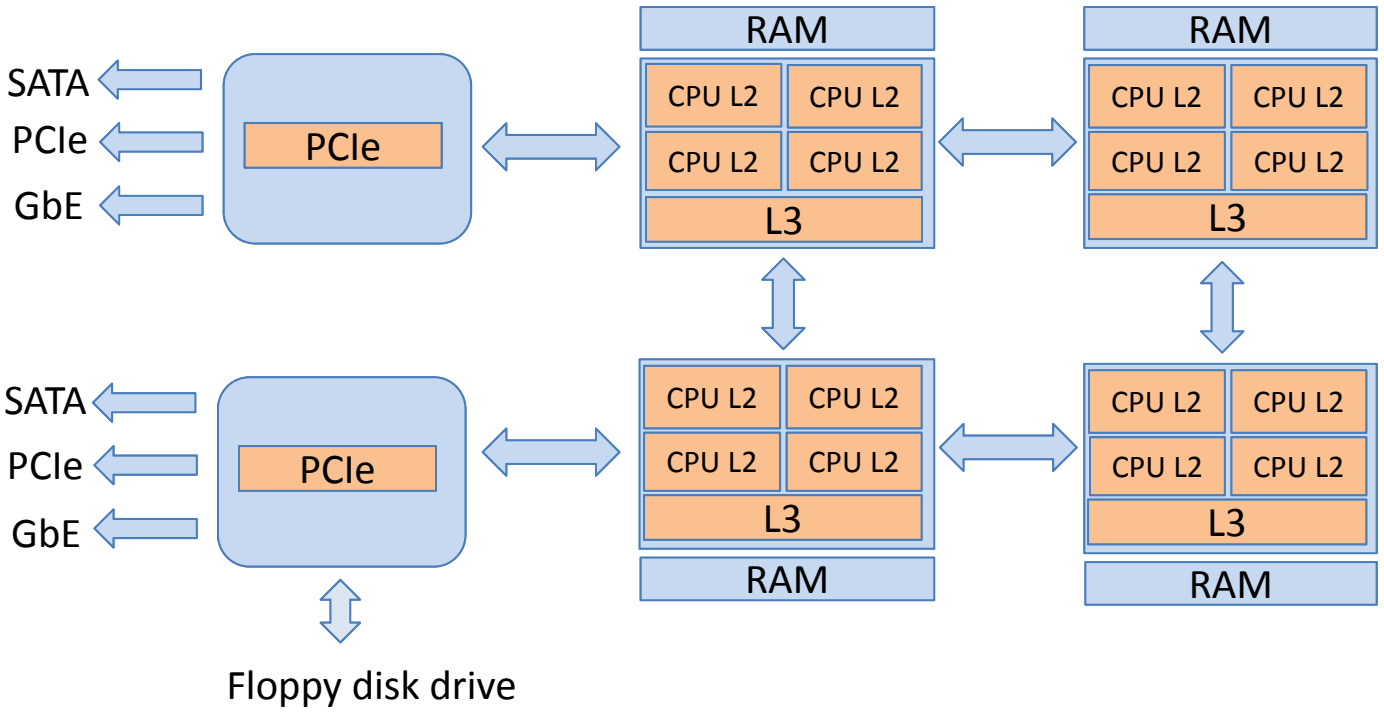
Messaging costs



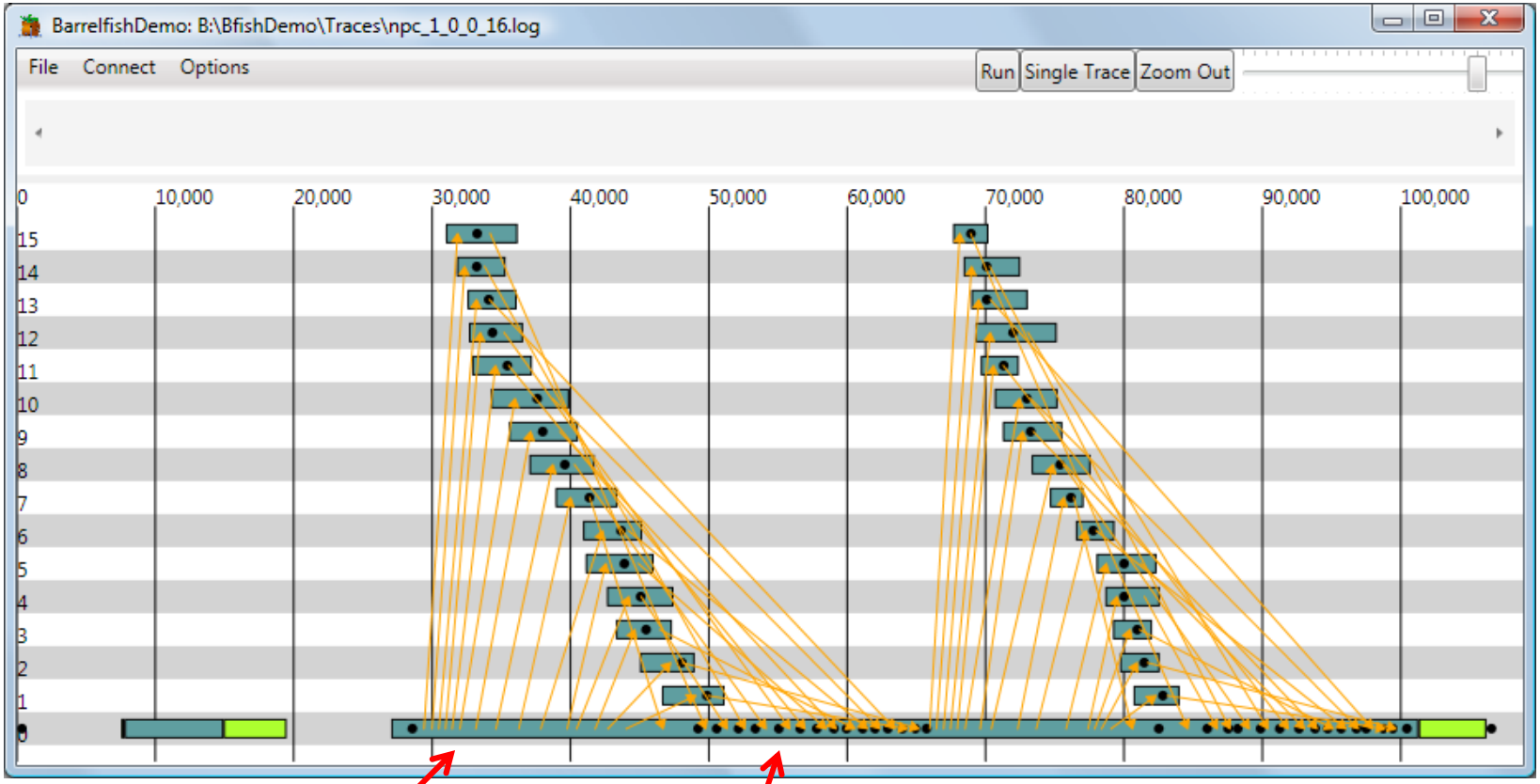
2 Phase Commit on a quad-core Opteron

Systems@ETH Zürich

Dell PowerEdge R905
4 * quad-core AMD Opteron



Trace: 2PC unicast



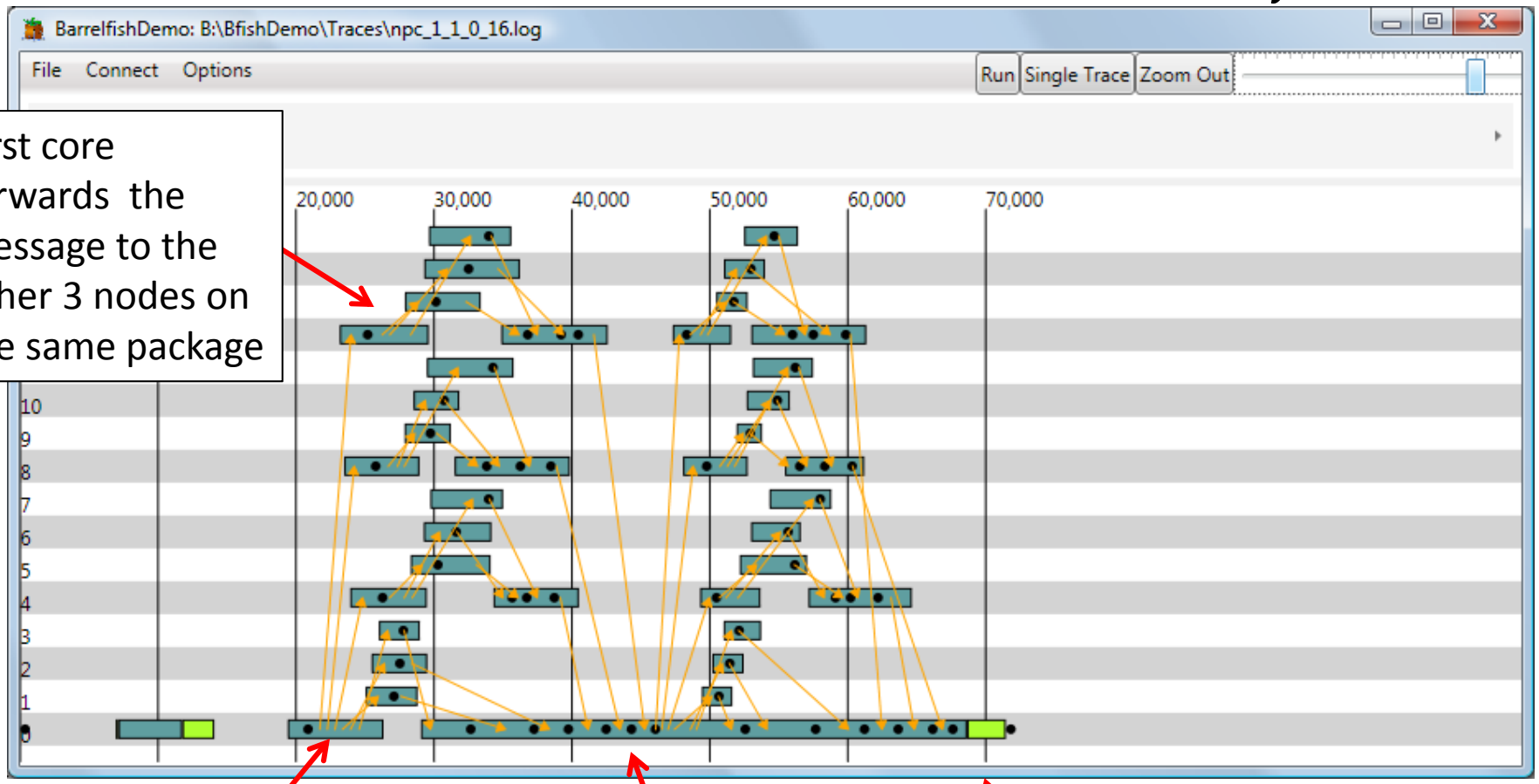
Core 0 sends a message to every other core

Waits for all the replies

Total time is 100,000 cycles (~40us)

Trace: 2PC NUMA-aware multicast

Systems@ETH Zürich



First core forwards the message to the other 3 nodes on the same package

Core 0 sends a message to a core on each remote package

Gathers the replies

Time reduced to 70,000 cycles