Multi-core Operating Systems

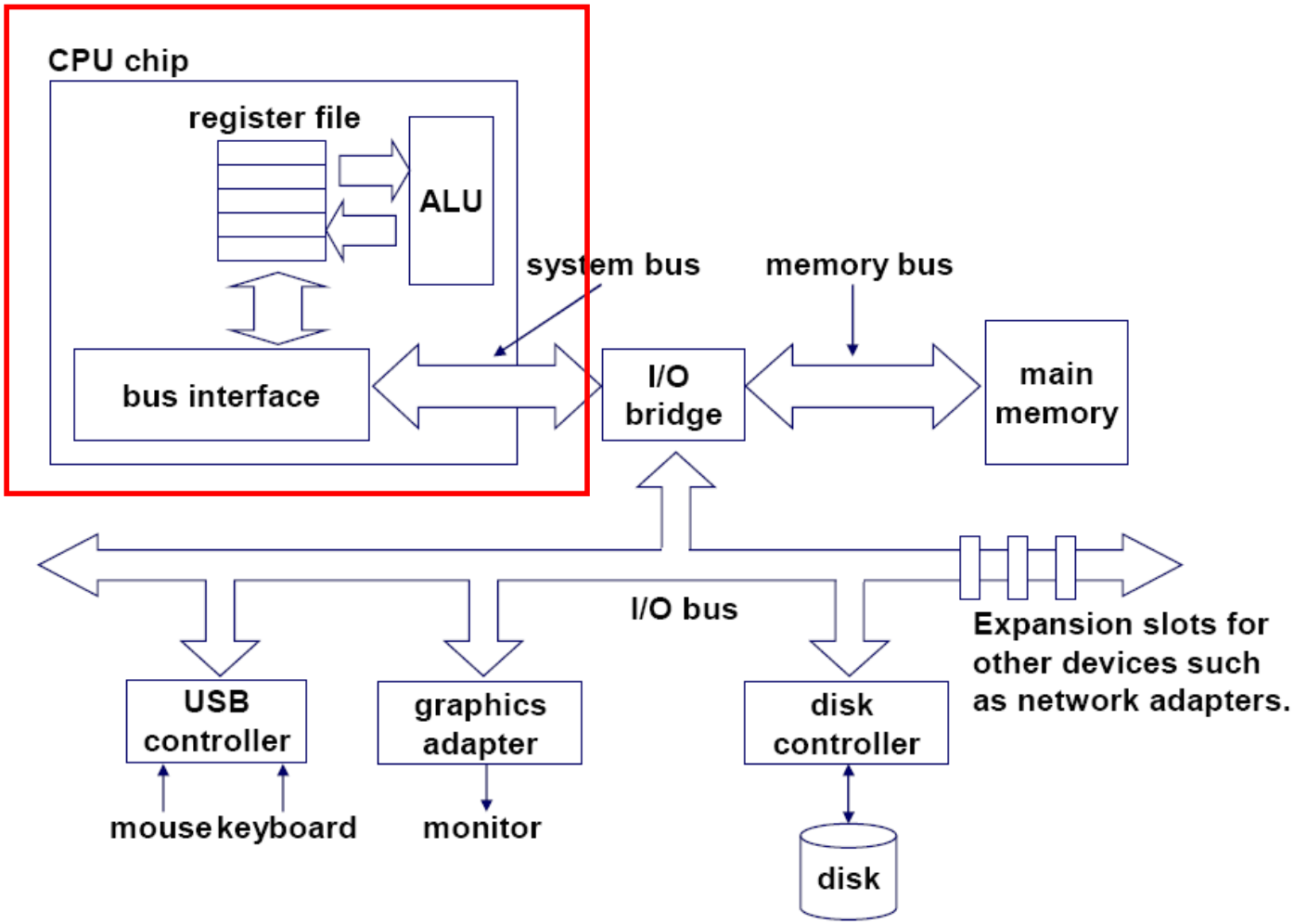
Daniel Shih
National Taiwan University

Outline

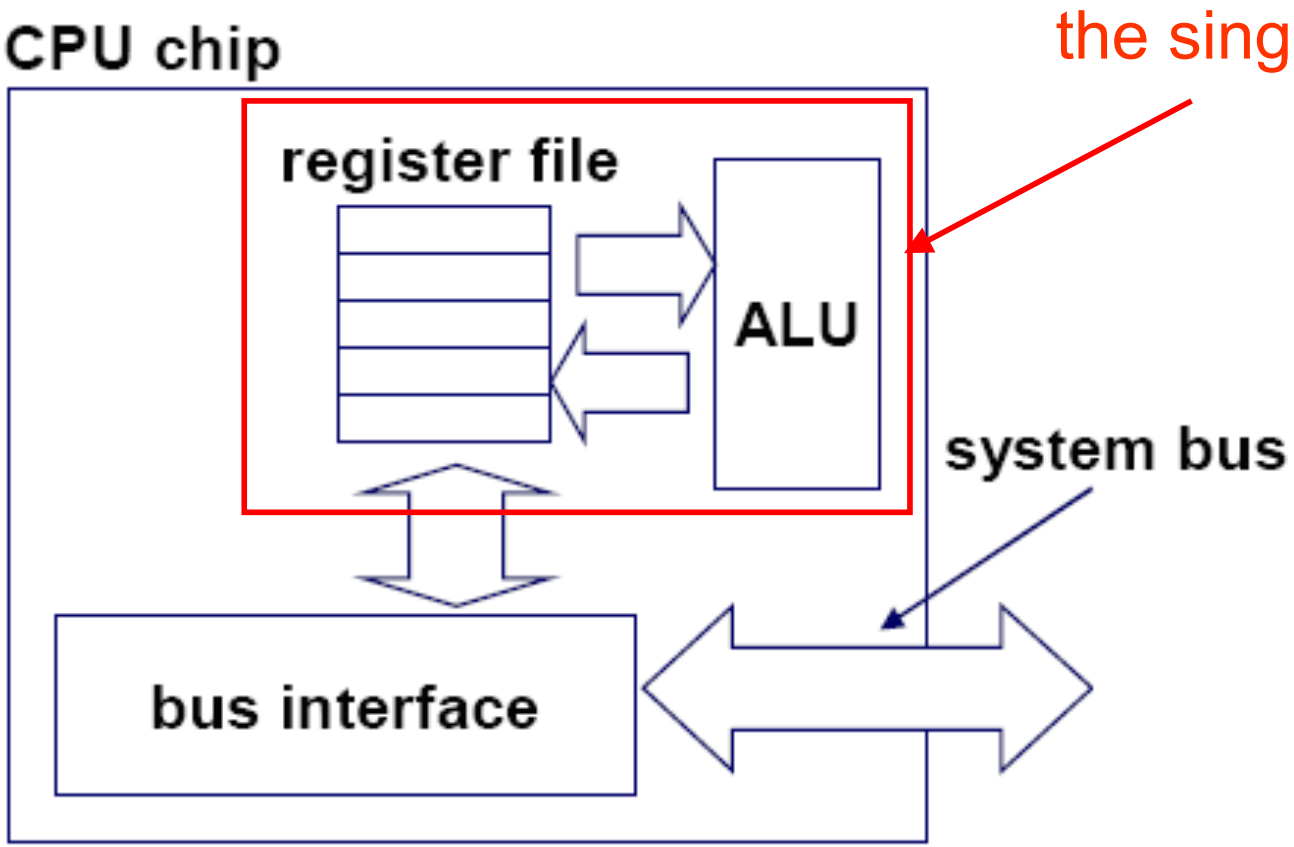
- Multi-core Architectures
- Desired features
- Multi-kernel Model
- FlexDCP: a QoS framework for CMP architectures

Multi-core Architecture

Single-core computer



Single-core CPU chip



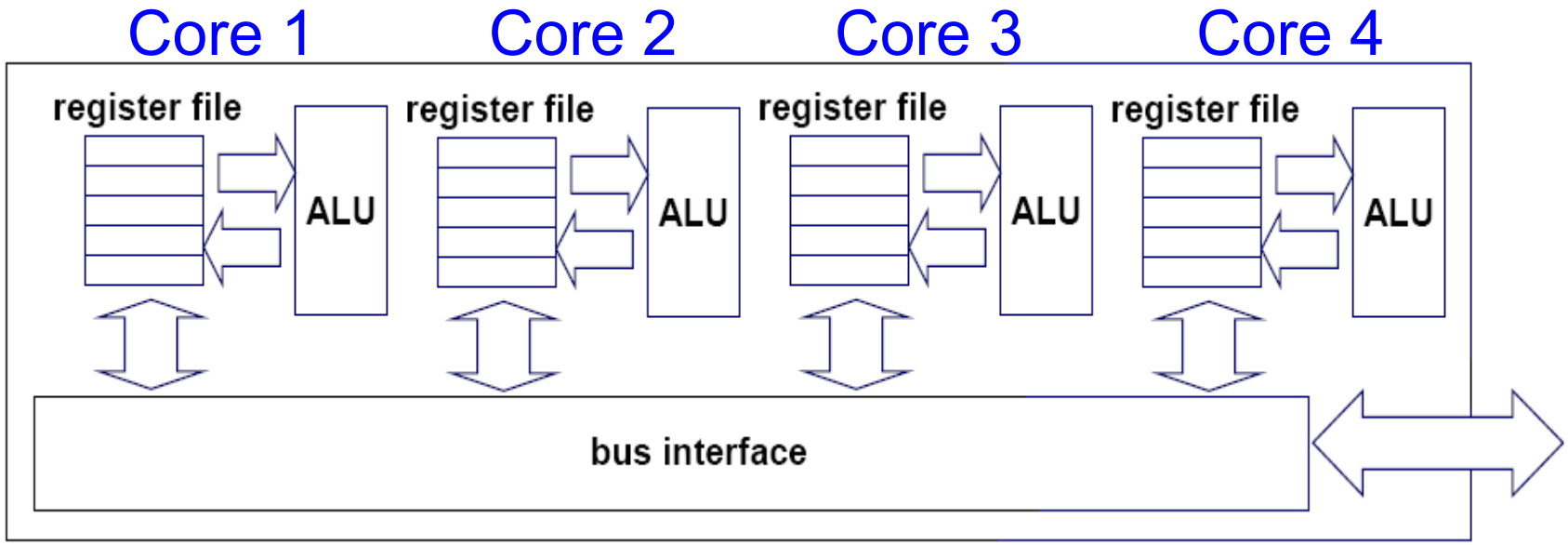
Why multi-core ?

- Difficult to make single-core clock frequencies even higher
- Deeply pipelined circuits:
  - heat problems
  - speed of light problems
  - difficult for design and verification
  - large design teams necessary
  - server farms need expensive air-conditioning

- Many new applications are multithreaded
- General trend in computer architecture (shift towards more parallelism)

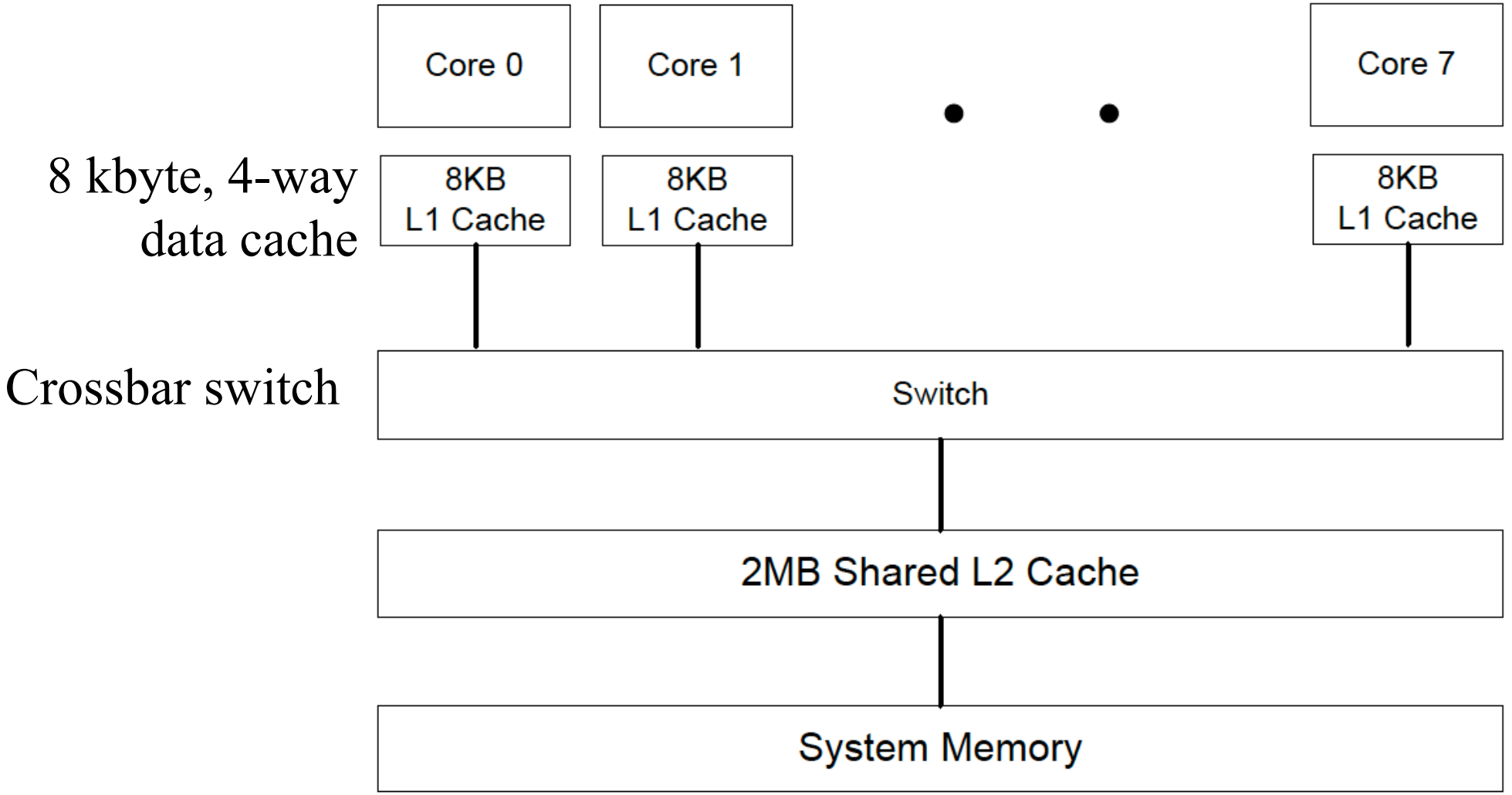
Multi-core architectures

- Multi-core Architecture: Replicate multiple processor cores on a single die.

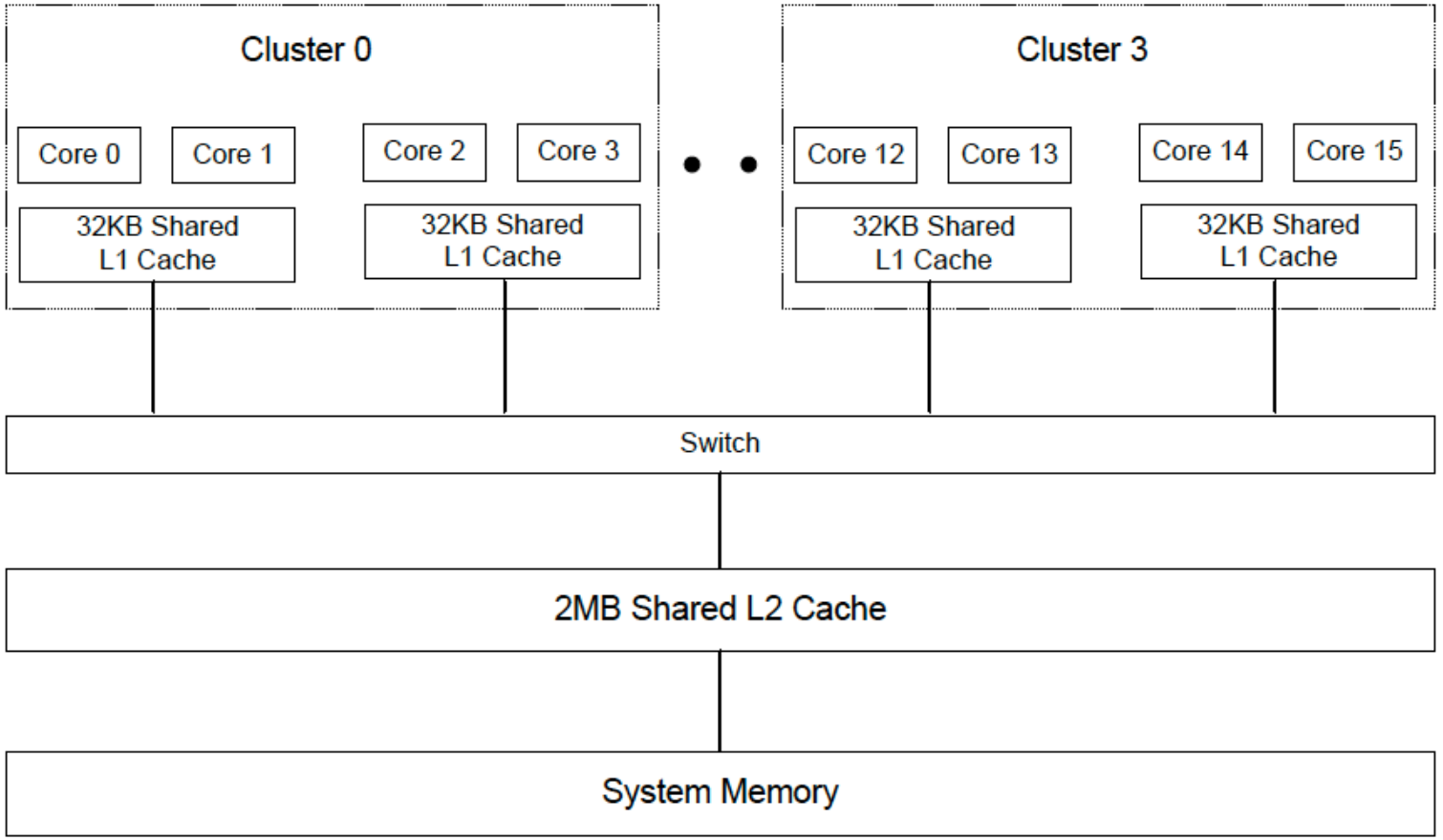


Multi-core CPU chip

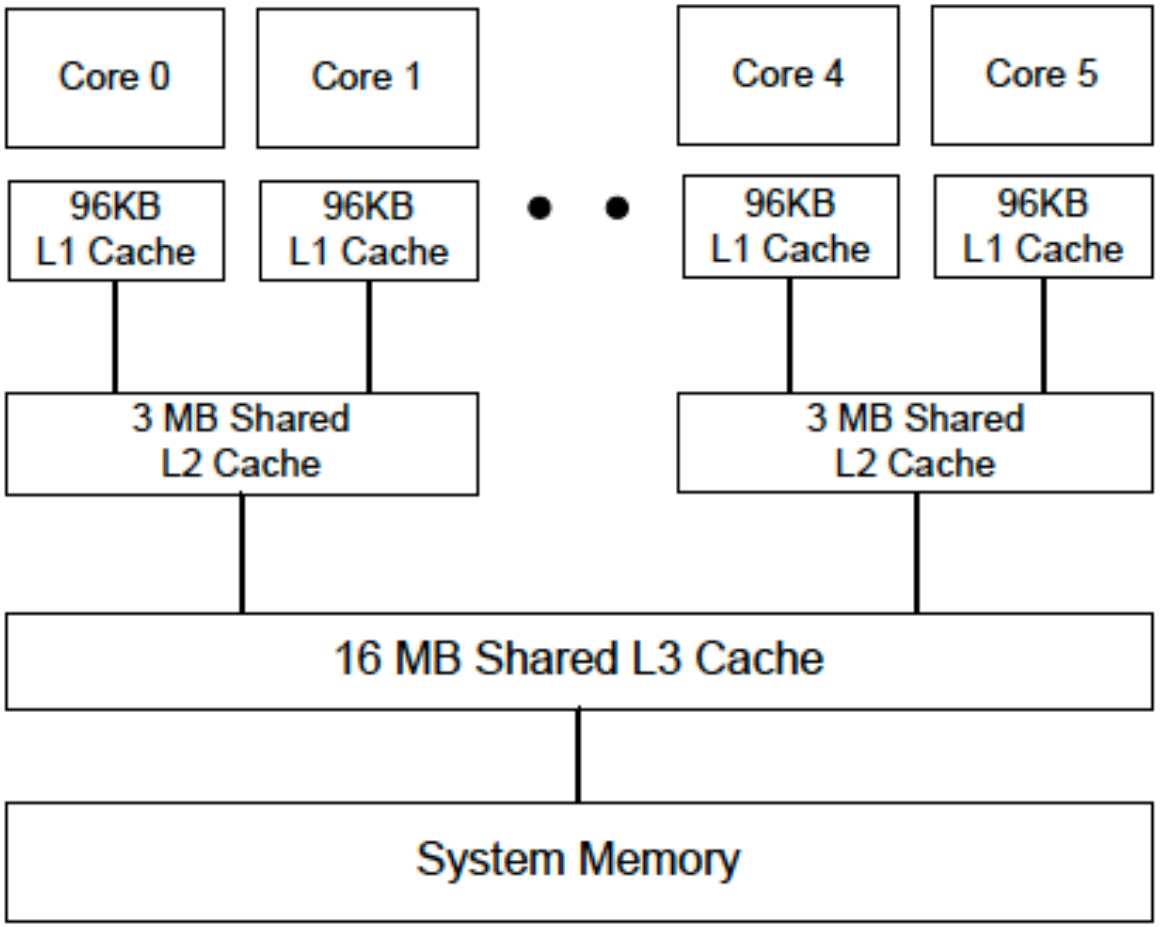
Sun UltraSPARC T2 Processors



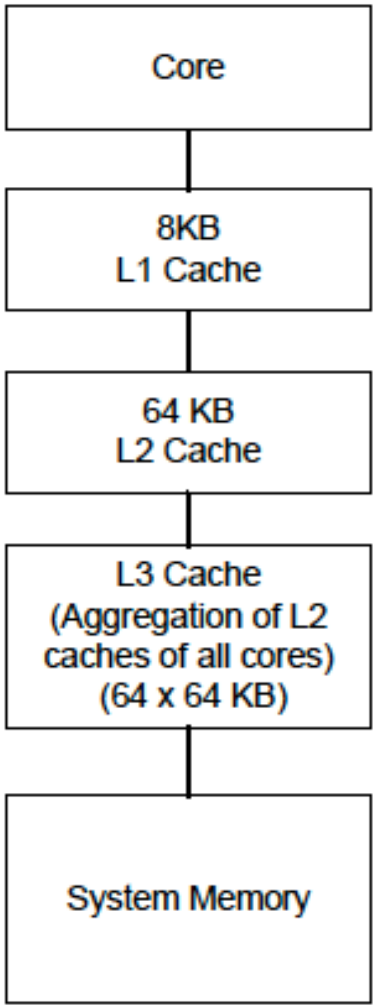
Sun 16-core Rock Processor



Intel 6-core Dunnington Processors

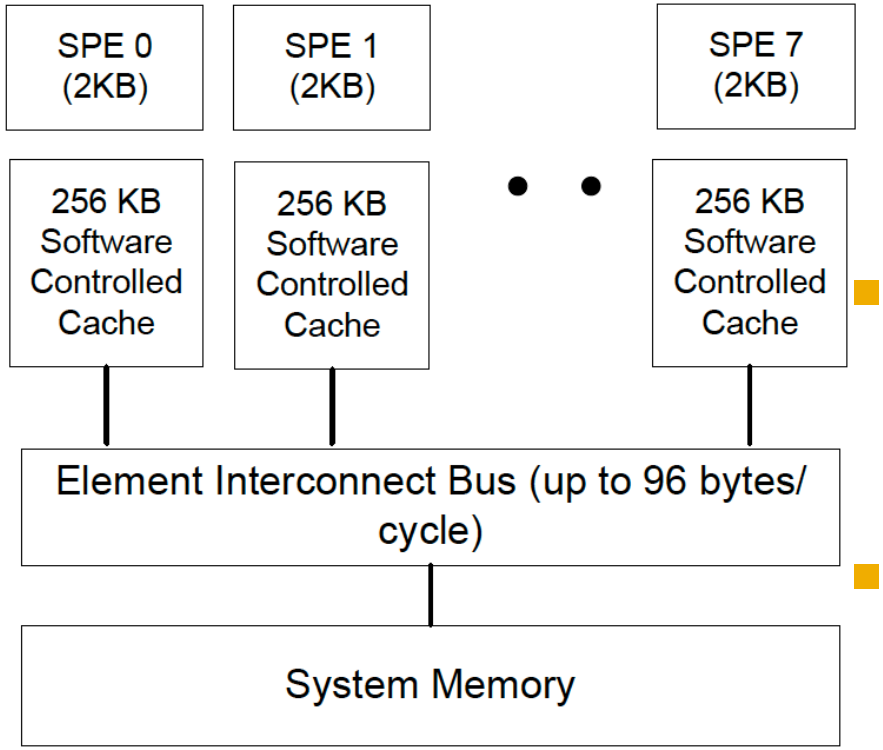


Tilera Tile64 Processors and Intel 80-core Tera-Scale Processors



- Tile64 has a 8x8 grid of processor cores.
- The Tile64 architecture eliminates on bus chip interconnect.
- Each processor core has a communication switch that connects it to a two dimensional on-chip mesh network called the iMesh.

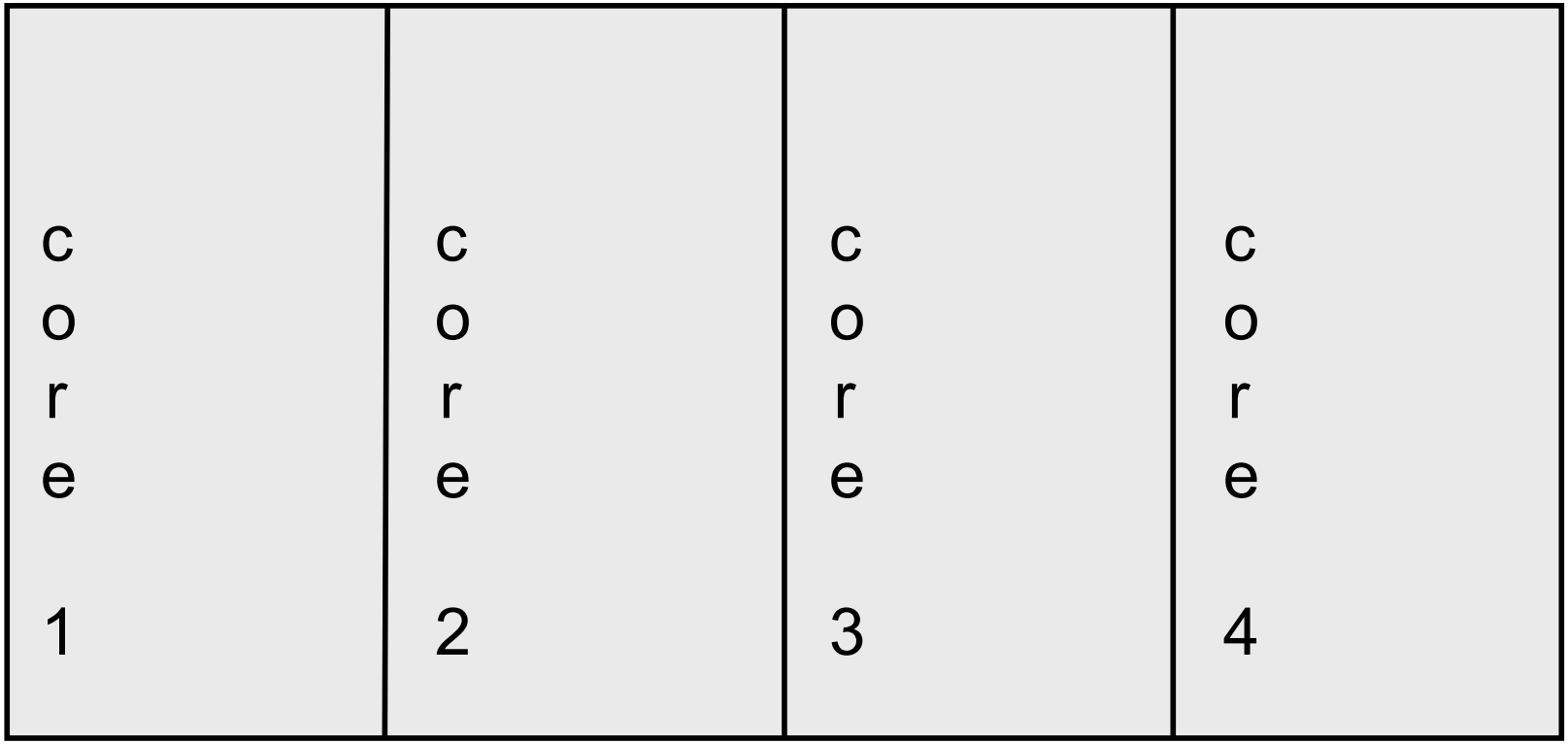
IBM, Sony, and Toshiba cell broadband engine multi-core architecture



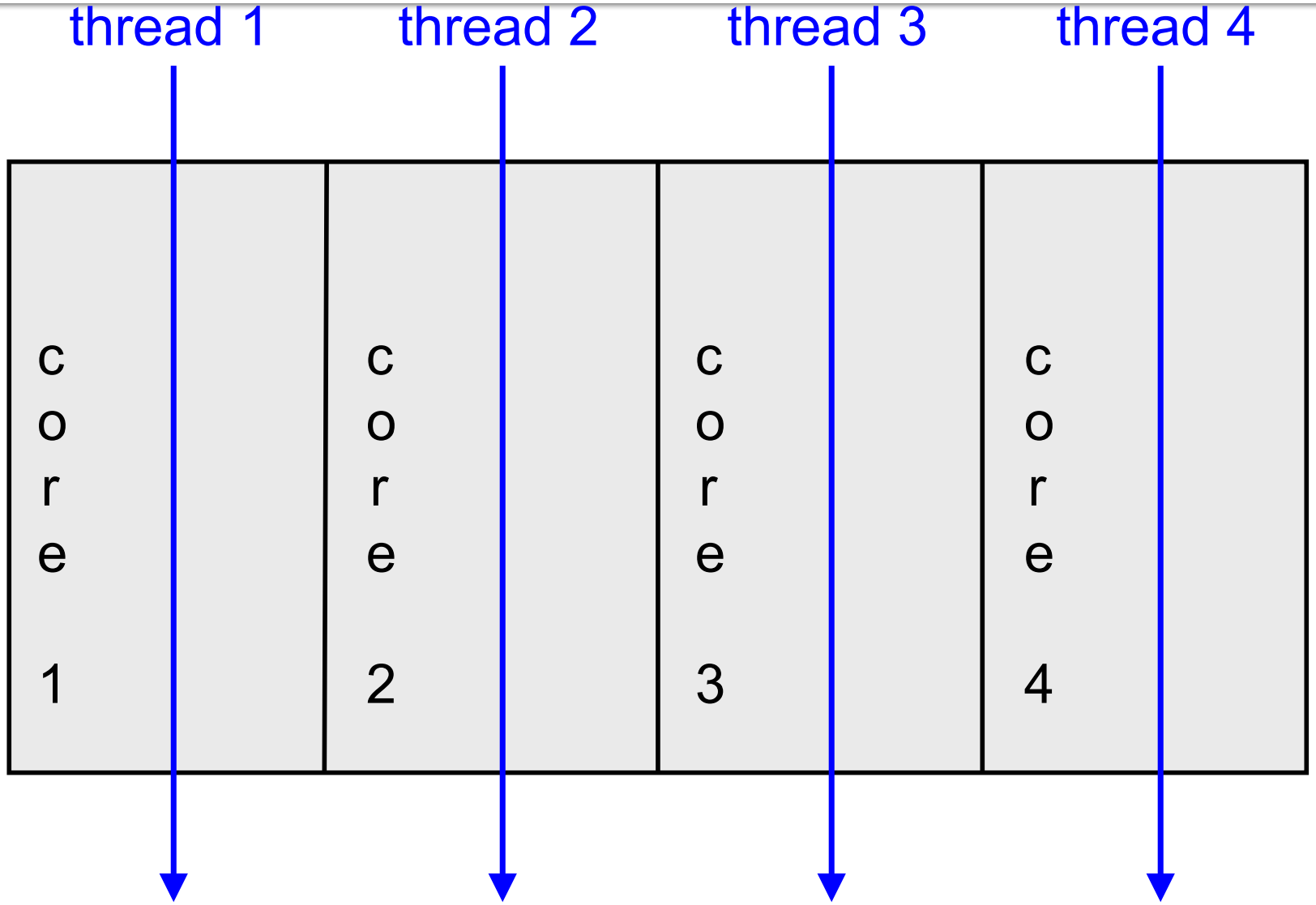
- SPE: An SPE has a register consisting of 128 registers of 128-bits each (2 KB of storage), and a local store of 256 KB.
- The architecture does not support a traditional cache-based memory hierarchy.
- The complexity of cache coherence is moved to the programmer/compiler for managing local store.

Multi-core CPU chip

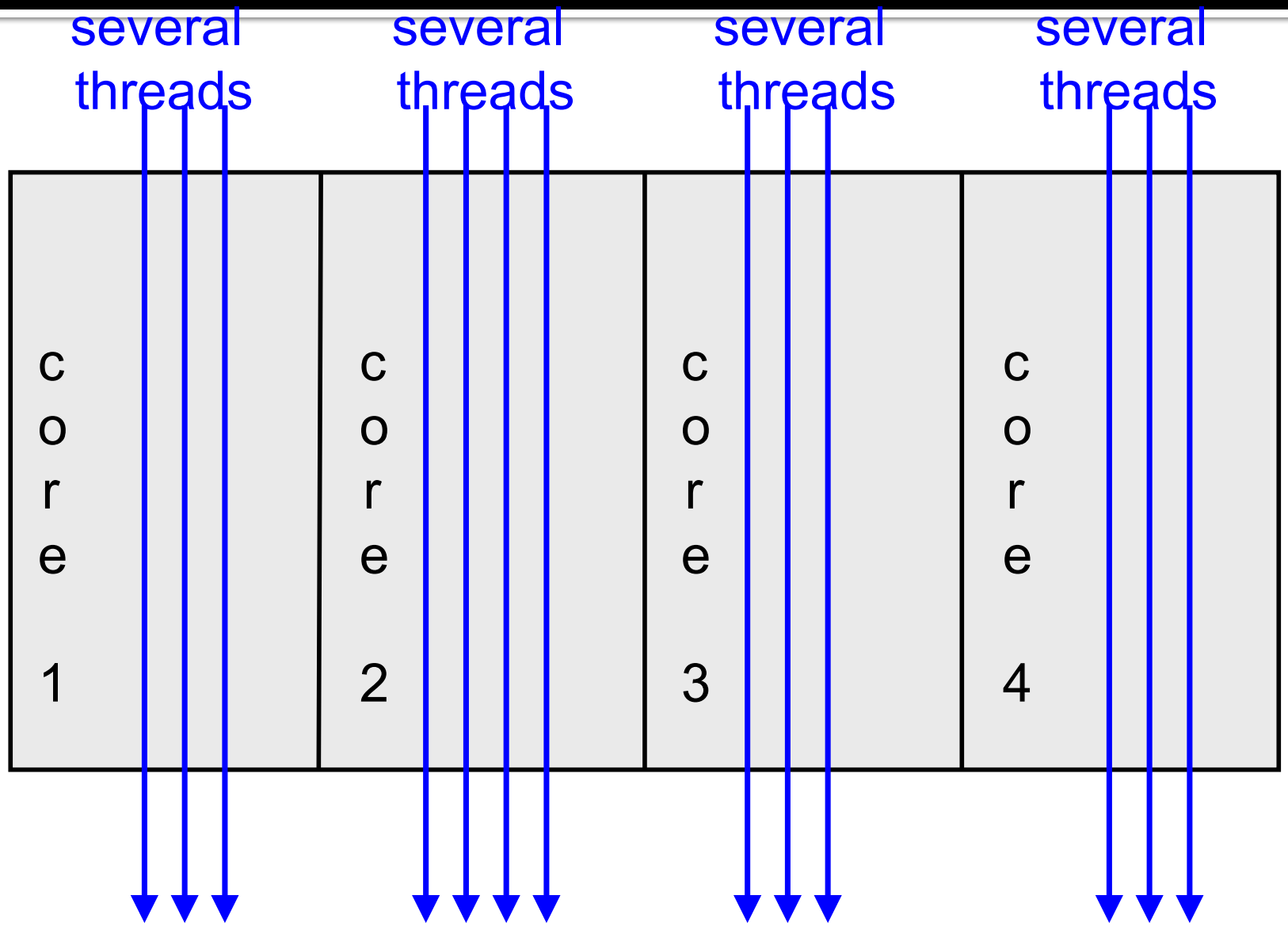
- The cores fit on a single processor socket
- Also called CMP (Chip Multi-Processor), Symmetric Multi-Core Processor (SMP)



The cores run in parallel



Within each core, threads are time-sliced (just like on a uniprocessor)



Instruction-level parallelism

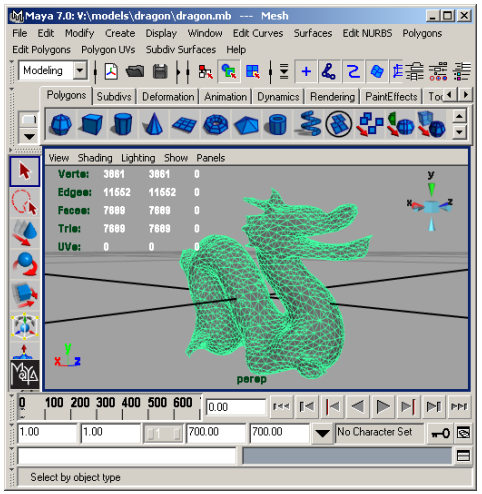
- Parallelism at the machine-instruction level
- The processor can re-order, pipeline instructions, split them into microinstructions, do aggressive branch prediction, etc.
- Instruction-level parallelism enabled rapid increases in processor speeds over the last 15 years

Thread-level parallelism (TLP)

- This is parallelism on a more coarser scale
- Server can serve each client in a separate thread (Web server, database server)
- A computer game can do AI, graphics, and physics in three separate threads
- Single-core superscalar processors cannot fully exploit TLP
- Multi-core architectures are the next step in processor evolution: explicitly exploiting TLP

What applications benefit from multi-core?

- Database servers
- Web servers (Web commerce)
- Compilers
- Multimedia applications
- Scientific applications, CAD/CAM
- In general, applications with *Thread-level parallelism* (as opposed to instruction-level parallelism)



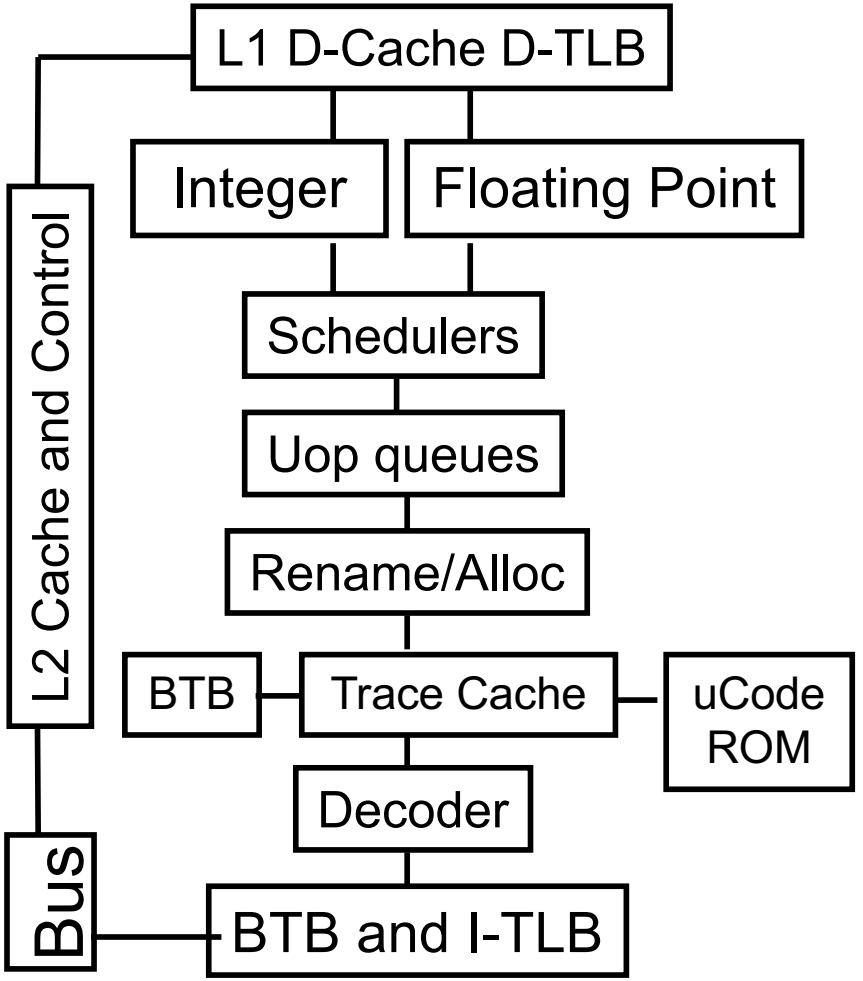
Each can run on its own core

More examples

- Editing a photo while recording a TV show through a digital video recorder
- Downloading software while running an anti-virus program
- "Anything that can be threaded today will map efficiently to multi-core"
- BUT: some applications difficult to parallelize

A technique complementary to multi-core: Simultaneous multithreading

- Problem addressed: The processor pipeline can get stalled:
  - Waiting for the result of a long floating point (or integer) operation
  - Waiting for data to arrive from memory
- Other execution units wait unused



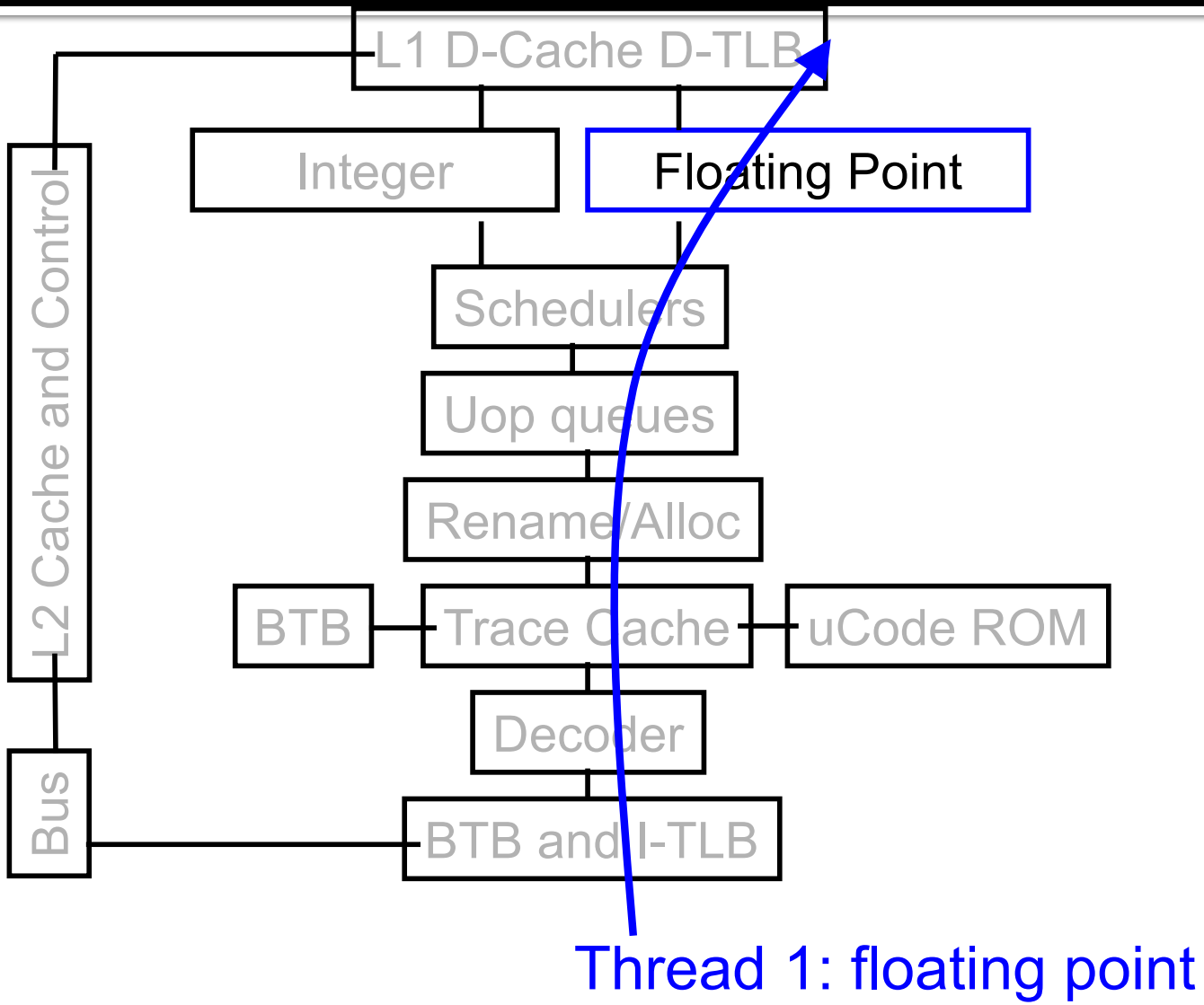
Source: Intel

Simultaneous multithreading (SMT)

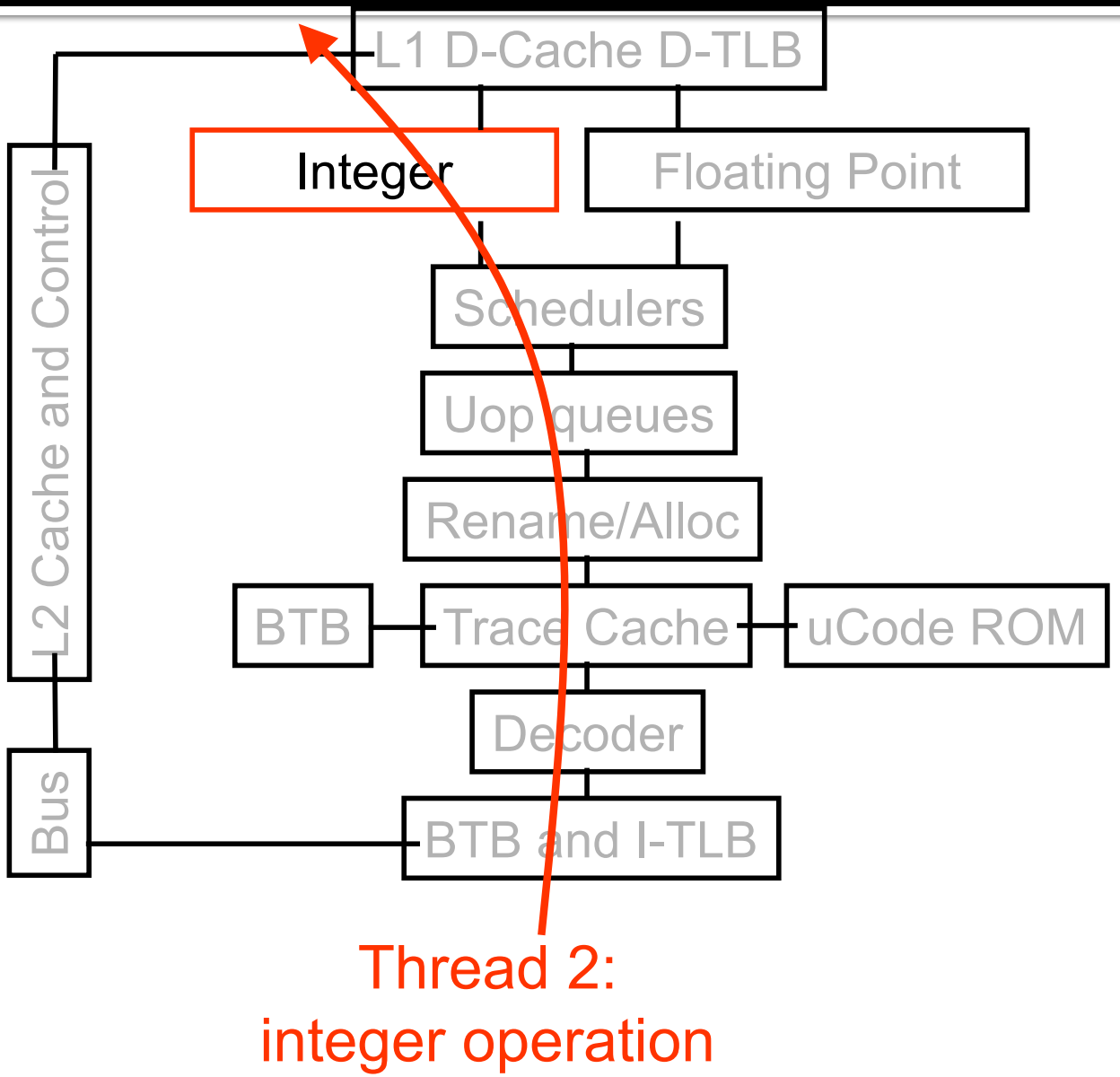
- Permits multiple independent threads to execute SIMULTANEOUSLY on the SAME core
- Weaving together multiple "threads" on the same core

- Example: if one thread is waiting for a floating point operation to complete, another thread can use the integer units

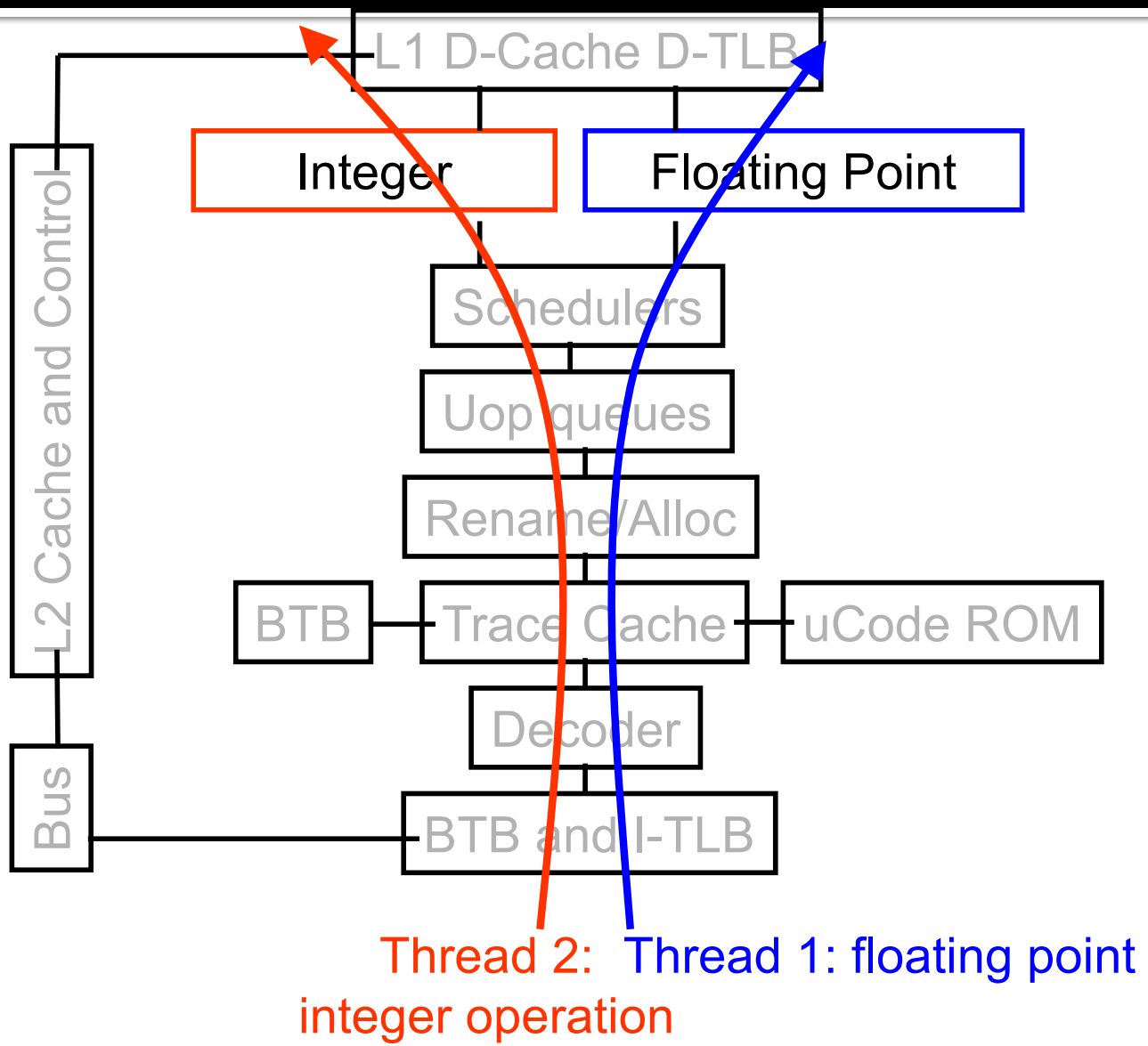
Without SMT, only a single thread can run at any given time



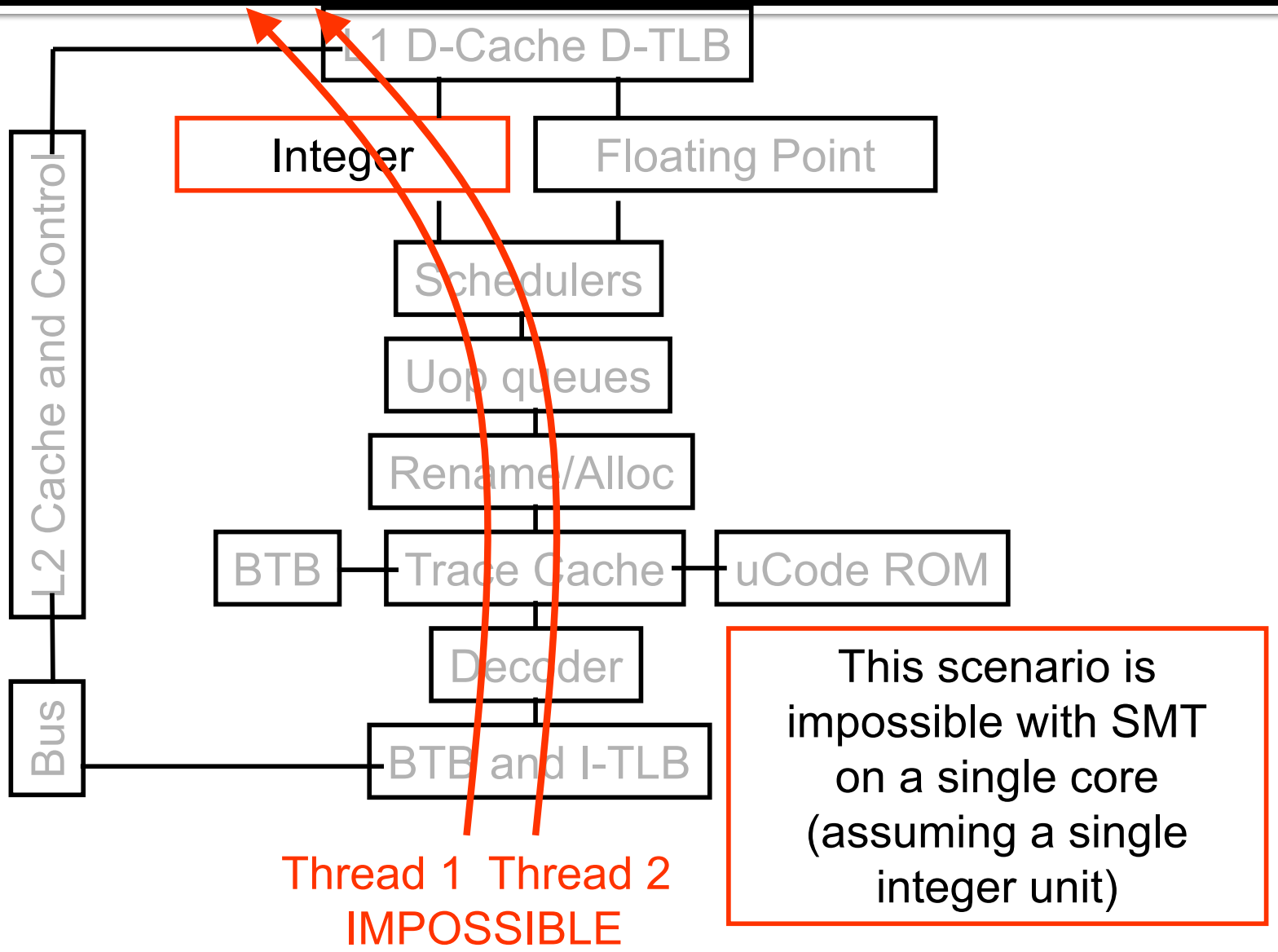
Without SMT, only a single thread can run at any given time



SMT processor: both threads can run concurrently



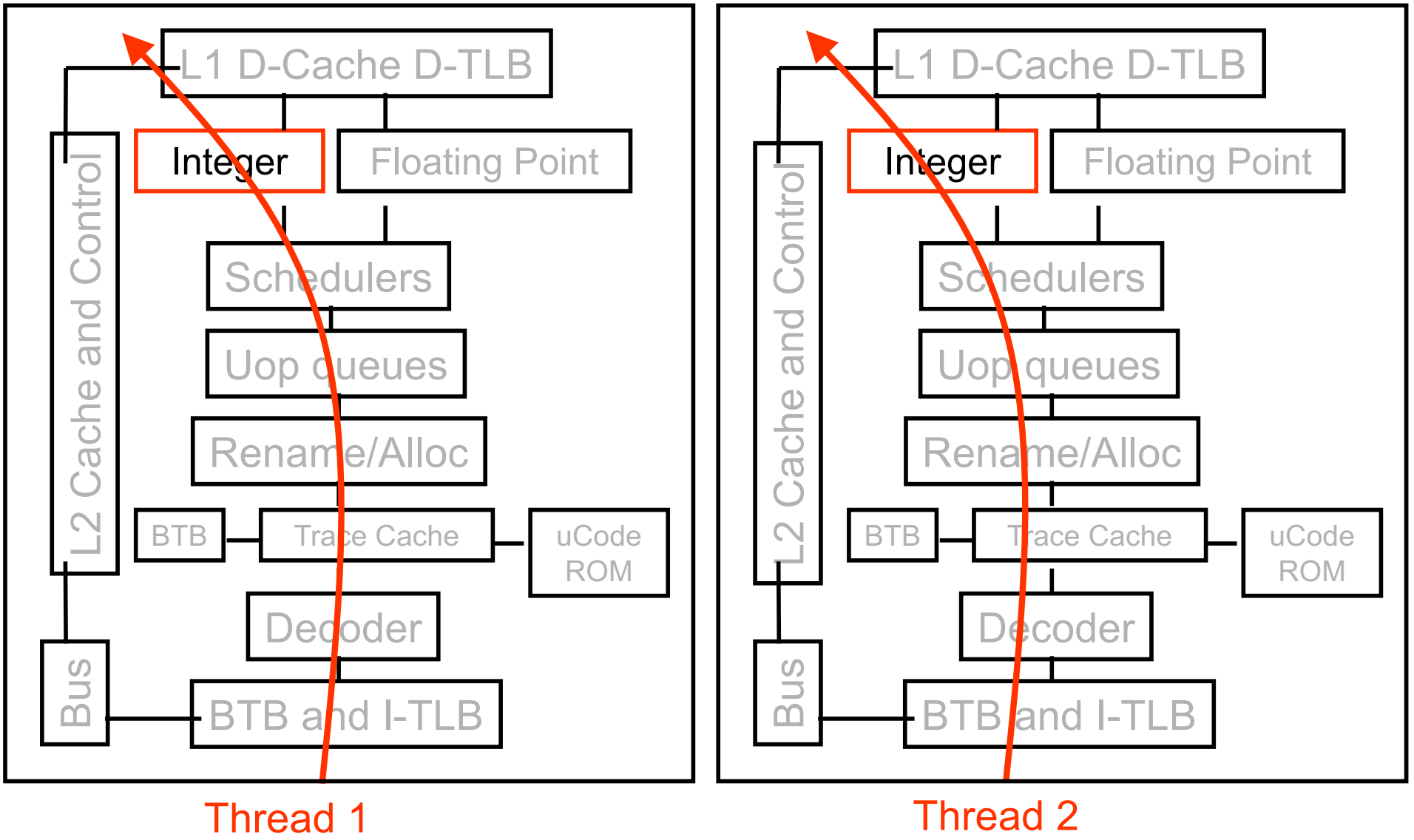
But: Can't simultaneously use the same functional unit



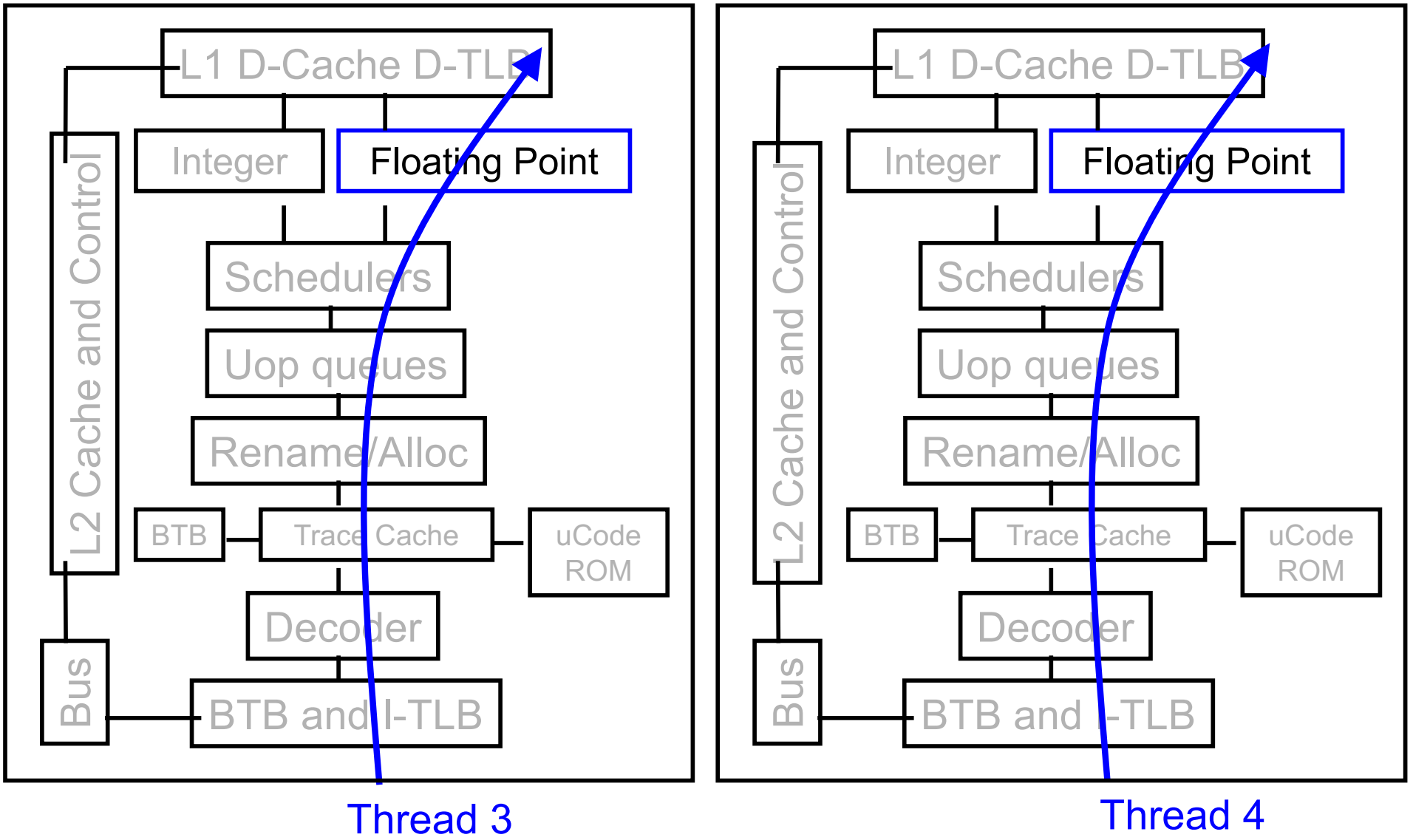
SMT not a "true" parallel processor

- Enables better threading (e.g. up to 30%)
- OS and applications perceive each simultaneous thread as a separate "virtual processor"
- The chip has only a single copy of each resource
- Compare to multi-core: each core has its own copy of resources

Multi-core:
threads can run on separate cores



Multi-core:
threads can run on separate cores

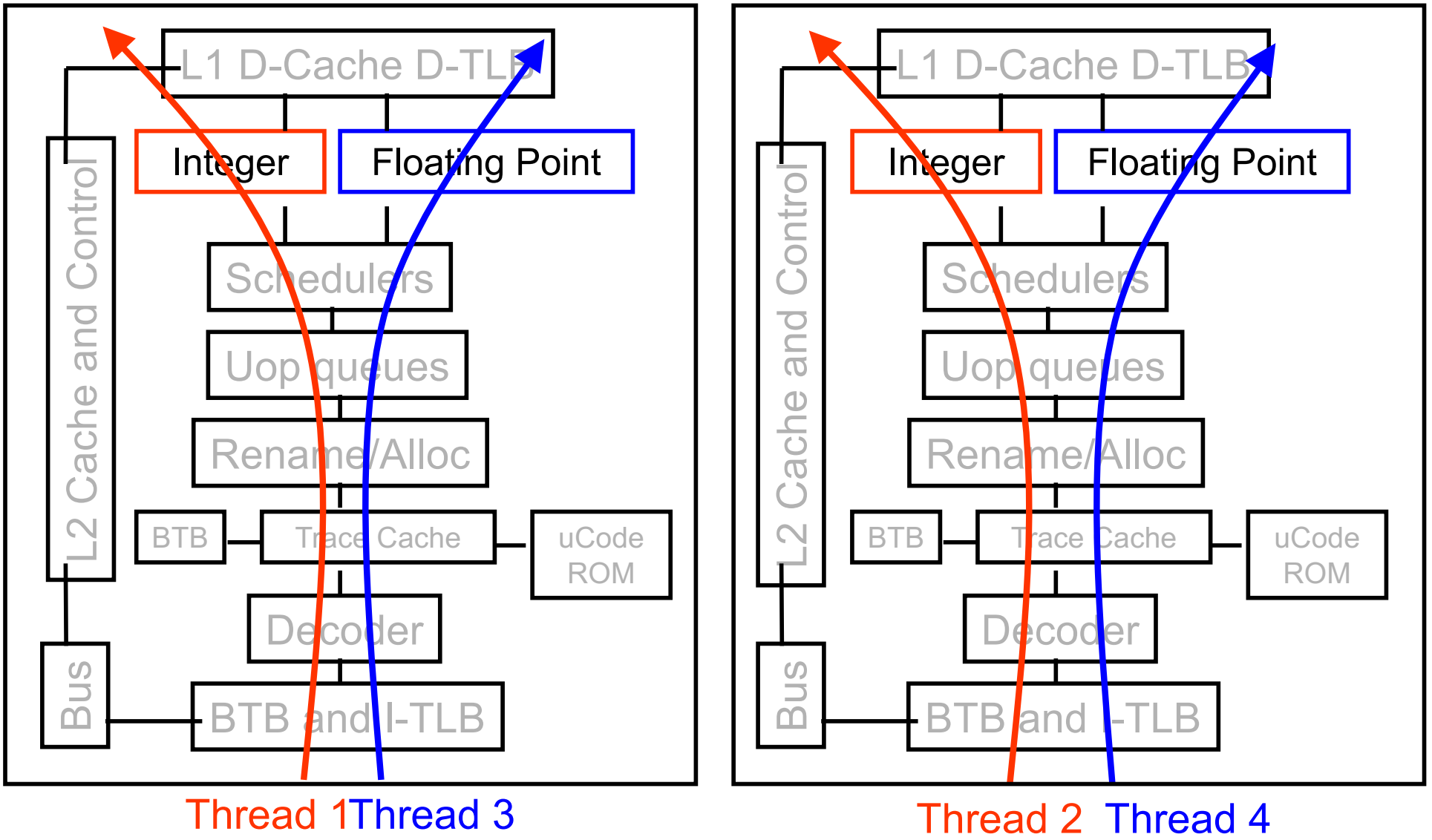


Combining Multi-core and SMT

- Cores can be SMT-enabled (or not)
- The different combinations:
  - Single-core, non-SMT: standard uniprocessor
  - Single-core, with SMT
  - Multi-core, non-SMT
  - Multi-core, with SMT
- The number of SMT threads:

  2, 4, or sometimes 8 simultaneous threads
- Intel calls them “hyper-threads”

SMT Dual-core: all four threads can run concurrently



Comparison: multi-core vs SMT

- Advantages/disadvantages?

Comparison: multi-core vs SMT

- Multi-core:
  - Since there are several cores,
    each is smaller and not as powerful
    (but also easier to design and manufacture)
  - However, great with thread-level parallelism
- SMT
  - Can have one large and fast superscalar core
  - Great performance on a single thread
  - Mostly still only exploits instruction-level parallelism

Multi-core vs. Multi-processors

- Contention on Shared Resources
  - More shared resource on the common path
  - Having unregulated contention in any of these shared resources may lower system throughput and hinder scalable performance.
- Non-uniform Inter-Core Communication Latency:
  - Hierarchical nature of multicore systems lead to non-uniform communication latency:
    - Inter-core communication via shared-cache vs. that via bus
    - One order of magnitude difference

Multi-core Operating Systems

- Most modern operating systems such as Microsoft Windows, Linux and Unix support multi-core processor platforms.
- Do we need a special designed operating systems for multi-core processors?

Troubles for Modern OSs

- Single authority
  - Modern and future computing systems operate in multiple trusted models.
  - No single authority can possibly uniformly and safely manage them all and a buggy device driver can jeopardize all programs.
  - Different applications/programs need different levels of protection and security policies.
  - High performance applications prefer to manage the device directly.

Troubles for Modern OSs

- Uniform device driver model:
  - Single address space for all devices makes it easier to program and to optimize the performance.
  - When the number of devices increase, OS becomes the performance bottleneck of the system.

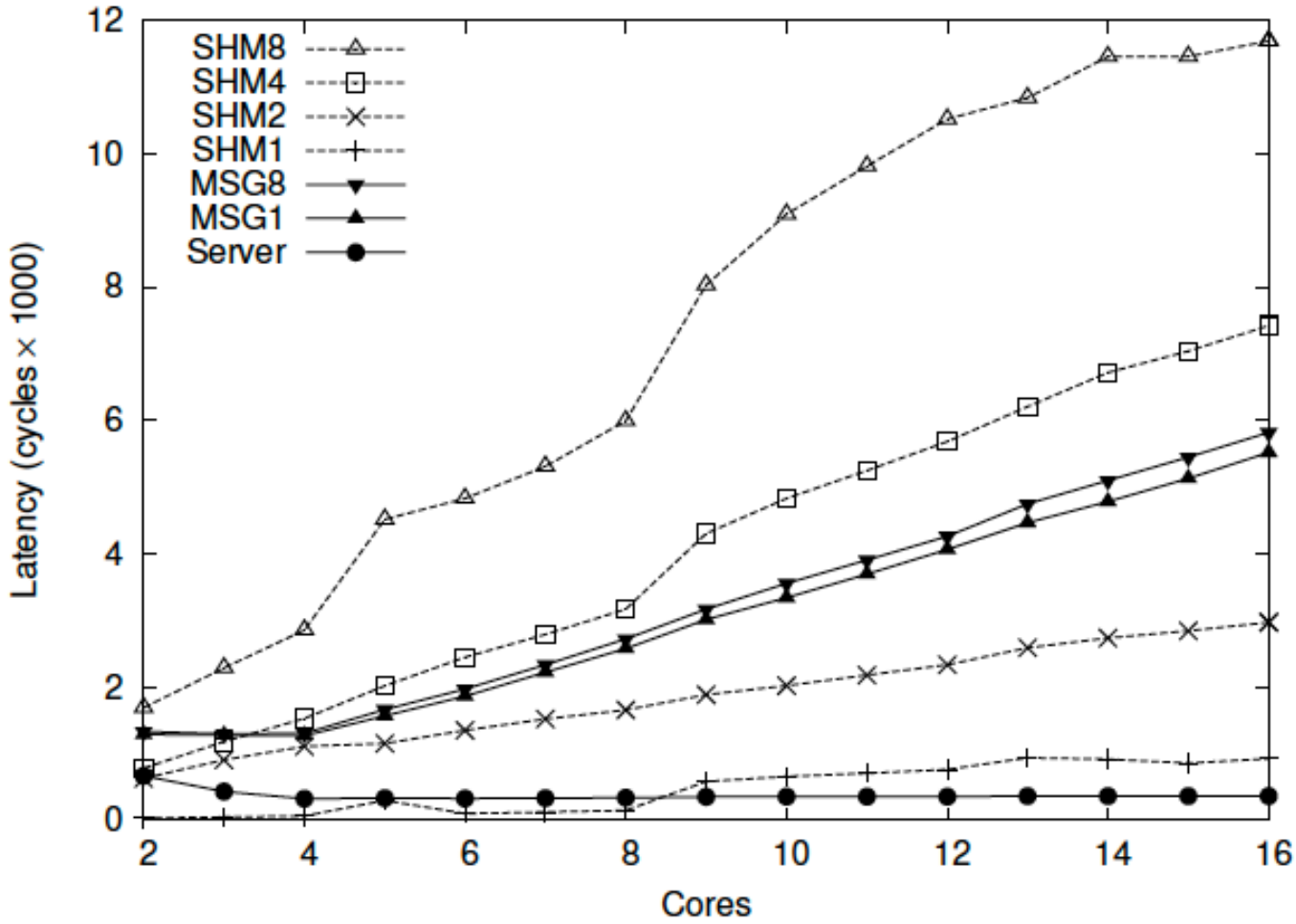
Troubles for Modern OSs

- Single performance mode:
  - Modern OSs are designed for *either* high throughput, high reliability, *or* high real-time guarantees.
  - Future computing systems will have applications with all these requirements running simultaneously.
- One resource management policy for all applications
  - Modern OSs try to provide everything any application could want.
  - Specialized operating systems for relational DBMS are more scalable and exploit domain knowledge.

Troubles for Modern OSs

- Scalability for cache-coherence
  - Shared-memory based cache-coherence mechanism suffers the performance when the number of core/processors increases.
  - Treating a multicore system as a message passing distributed system helps to lower the overhead of cache update.

Shared-memory vs. Message passing



Latency against number of cores for updates of various sizes on the 4x4-core AMD system

Desired Features

- Transparency
- Memory management
- High system throughput
- Application specific thread scheduling
- Isolation
- Dynamic hardware reconfiguration

Desired Features

- Tolerant to hardware fault or run-time hardware reconfiguration
  - Multi-core processors are designed to tolerate the faulty of individual cores.
  - Hardware resources including processing cores, memory, and bus may be partitioned at run-time to meet the needs.
- Operating systems should be able to tolerate such fault and dynamic configuration.

Multikernel: An OS architecture for scalable multicore Systems

The Multikernel: A New OS Architecture for Scalable Multicore Systems, Andrew Baumann, Paul Barhamy, Pierre-Evariste Dagandz, Tim Harrisy, Rebecca Isaacsy, Simon Peter, Timothy Roscoe, Adrian Schüpbach, and Akhilesh Singhania, SOSP 2009.

One solution for all questions?

- Multicore systems are used in a variety of environments ranging from embedded systems, personal computing platforms to data centers.
- Workloads are less predictable and more OS-intensive.
- No longer acceptable to tune a general-purpose OS design for a particular hardware platform.
- Scalability problem must affect a substantial group of users.

Architecture

- Rethinking the structure of the OS as a distributed system of functional units communicating via explicit messages.
- Three design principles:
  - Make all inter-core communication explicit
  - Make OS structure hardware-neutral
  - View state as replicated instead of shared.

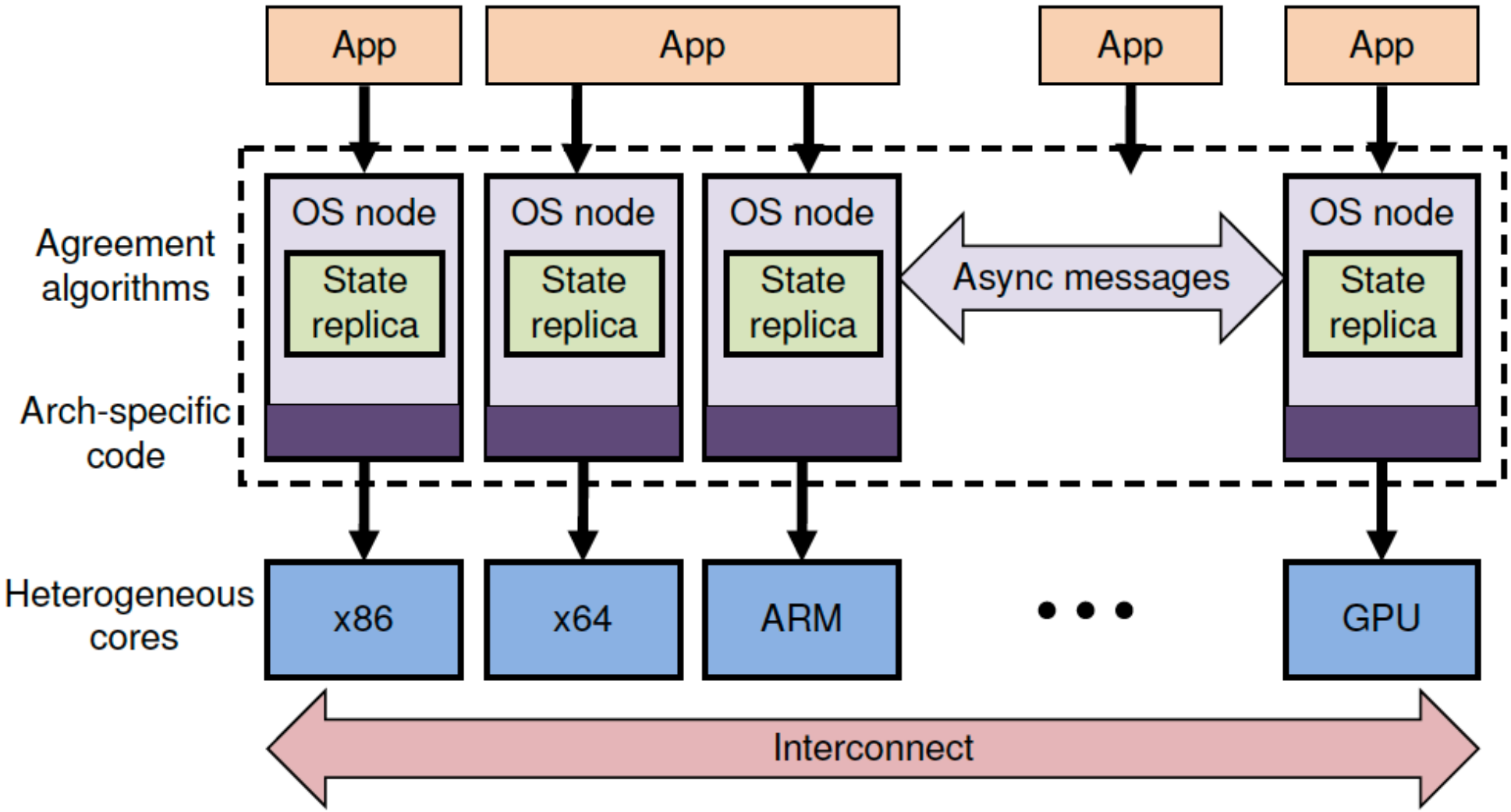
Explicit inter-core communication

- Implicit inter-core communication (e.g., shared memory update) suffers system performance when the number of shared copies increase.
- Explicit inter-core communication (e.g., message passing) allows the OS to make more efficient use of the interconnect
  - Pipelining scheduling
  - Batch process
  - Message routing for network system can be reused.

Explicit Inter-core Communication

- It allows the OS to
  - provide isolation and resource management on heterogeneous cores, or to
  - schedule jobs effectively on arbitrary inter-core topologies by placing tasks with reference to communication patterns and network effects.
- Requests and responses are decoupled and allow
  - Asynchronous operations and
  - Wakeup when needed to save power.

Multikernel Model



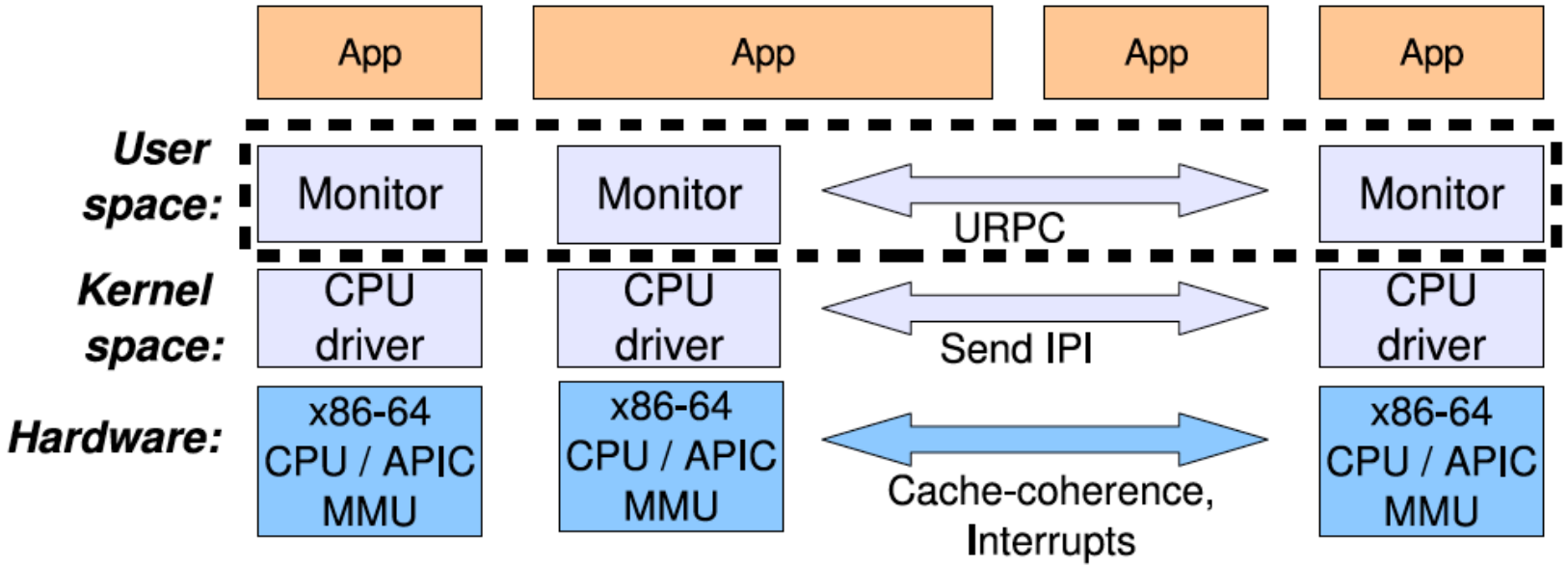
Hardware-neutral OS

- OS only targets two aspects:
  - Messaging transport mechanism and
  - Interface to hardware (CPUs and devices)
- Advantage:
  - Easy to port for difference performance characteristics
  - Message passing mechanism can be optimized for hardware-specific technique
  - Easy to adjust the message passing parameters (e.g., queue length and block size) in the late development stage.

View state as replicated, not shared

- Shared global state such as process dispatch queue requires mutual exclusive access.
- Replication is well known for scalability.
  - It reduces the contention on memory, load on system interconnect and overhead of synchronization.
- Each OS owns a replicated local copy of the data including TLB and consistency is maintained by exchanging messages.
  - Consistence semantics determine the overhead
  - Well-designed mechanism can overcome the overhead.

Implementation



- Platforms:
  - 2x4-core Intel system: 2 quad-core 2.66 Ghz processors and a single memory controller.
  - 2x2-core AMD system: 2 dual-core 2.6 Ghz AMD Opteron 2220
  - 4x4-core AMD system: 4 quad-core 2.5GHz AMD Opterson 8380
  - 8x4 core AMD system: q quad-core 2GHz AMD Operton 8350

System Structure

- CPU Driver:
  - CPU drivers are purely local to a core
  - CPU drivers encapsulate the functionality found in a typical monolithic microkernel: scheduling, communication, and low-level resource allocation.
- Monitor:
  - a distinguished user-mode process
  - all inter-core coordination is performed by monitors

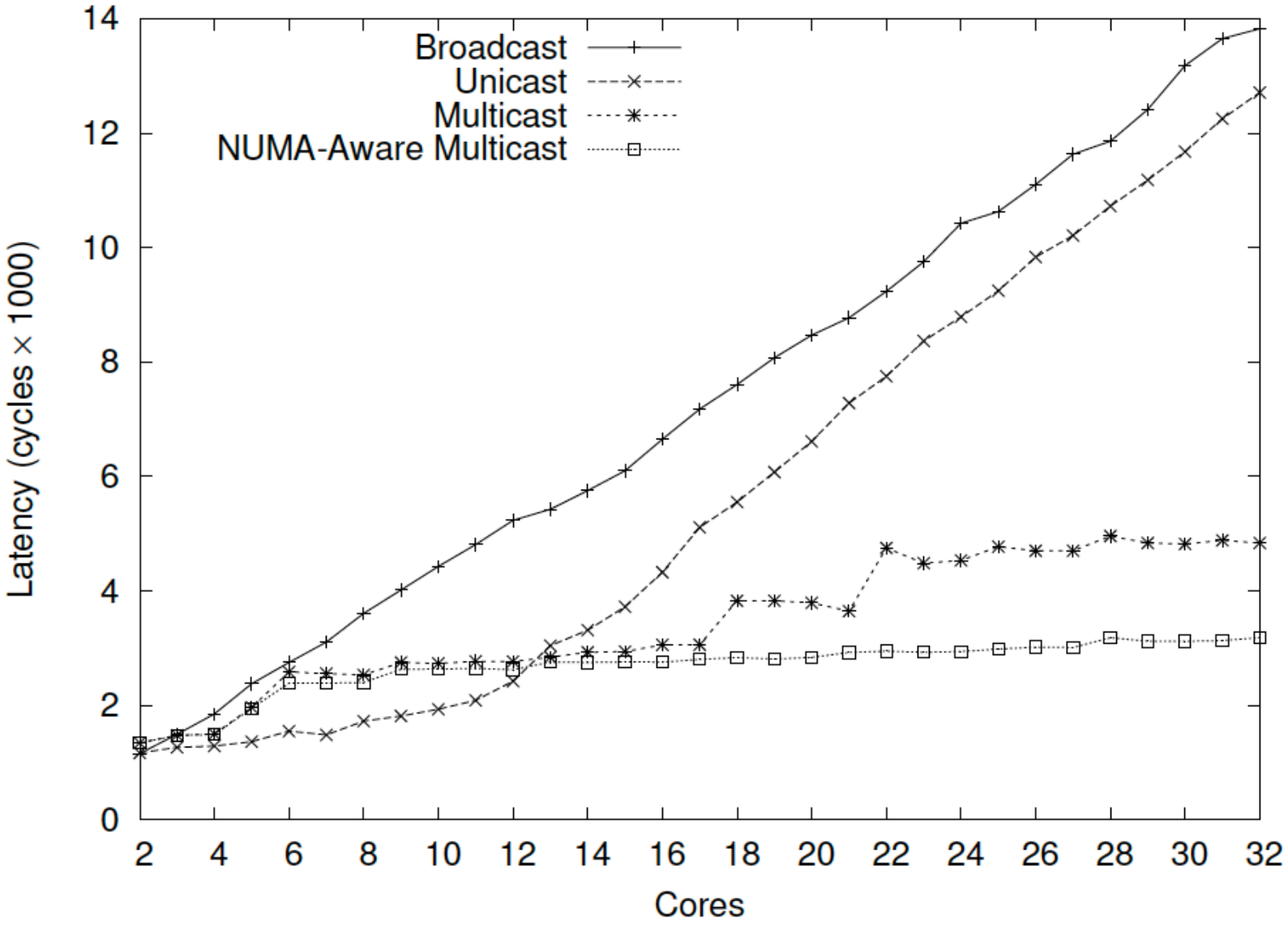
Process Structure

- A process is represented by a collection of dispatcher objects, one on each core on which it might execute.
- Communication is not actually between processes but between dispatchers and hence cores.
- Dispatchers on a core are scheduled by the local CPU driver

Inter-core Communication

- All inter-core communication occurs with messages.
- Different transport implementations for different hardware scenarios.
- User-level RPC (URPC) between cores: a region of shared memory is used as a channel to transfer cache-line-sized messages point-to-point between single writer and reader cores.
- Receiving URPC messages is done by polling memory
- All message transports are abstracted behind a common interface, allowing messages to be marshaled, sent and received in a transport-independent way.

Inter-core Communication: Broadcast, Unicast, Multicast, vs. NUMA aware multicast



Memory Management

- All virtual memory management, including allocation and manipulation of page tables, is performed entirely by user-level code.
  - To allocate and map in a region of memory, a user process must first acquire capabilities to sufficient RAM to store the required page tables
  - These RAM capabilities are mapped into page table capabilities, allowing it to insert the new page tables into its root page table
  - The CPU driver performs the actual page table and capability manipulations

Memory Management

- Page mapping and remapping requires global coordination and is implemented by one-phase commit operation between all the monitors.
- Capability retyping, of which revocation is a special case.
  - Changing the usage of an area of memory and requires global coordination.
  - All cores must agree on a single ordering of the operations to preserve safety
  - The monitors initiate a two-phase commit protocol to ensure that all changes to memory usage are consistently ordered across the processors.

Reference

- Enhancing Operating System Support for Multicore Processors by Using Hardware Performance Monitoring, Reza Azimi, David K. Tam, Livio Soares, and Michael Stumm
- Challenges and Opportunities in Many-Core Computing, John L. Manferdelli, Naga K. Govindaraju, and Chris Crall, Proceedings of the IEEE, Vol. 96, No. 5, May 2008.
- The Multikernel: A New OS Architecture for Scalable Multicore Systems, Andrew Baumann, Paul Barhamy, Pierre-Evariste Dagandz, Tim Harrisy, Rebecca Isaacsy, Simon Peter, Timothy Roscoe, Adrian Schüpbach, and Akhilesh Singhania, SOSP 2009.