UNSW
THE UNIVERSITY OF NEW SOUTH WALES

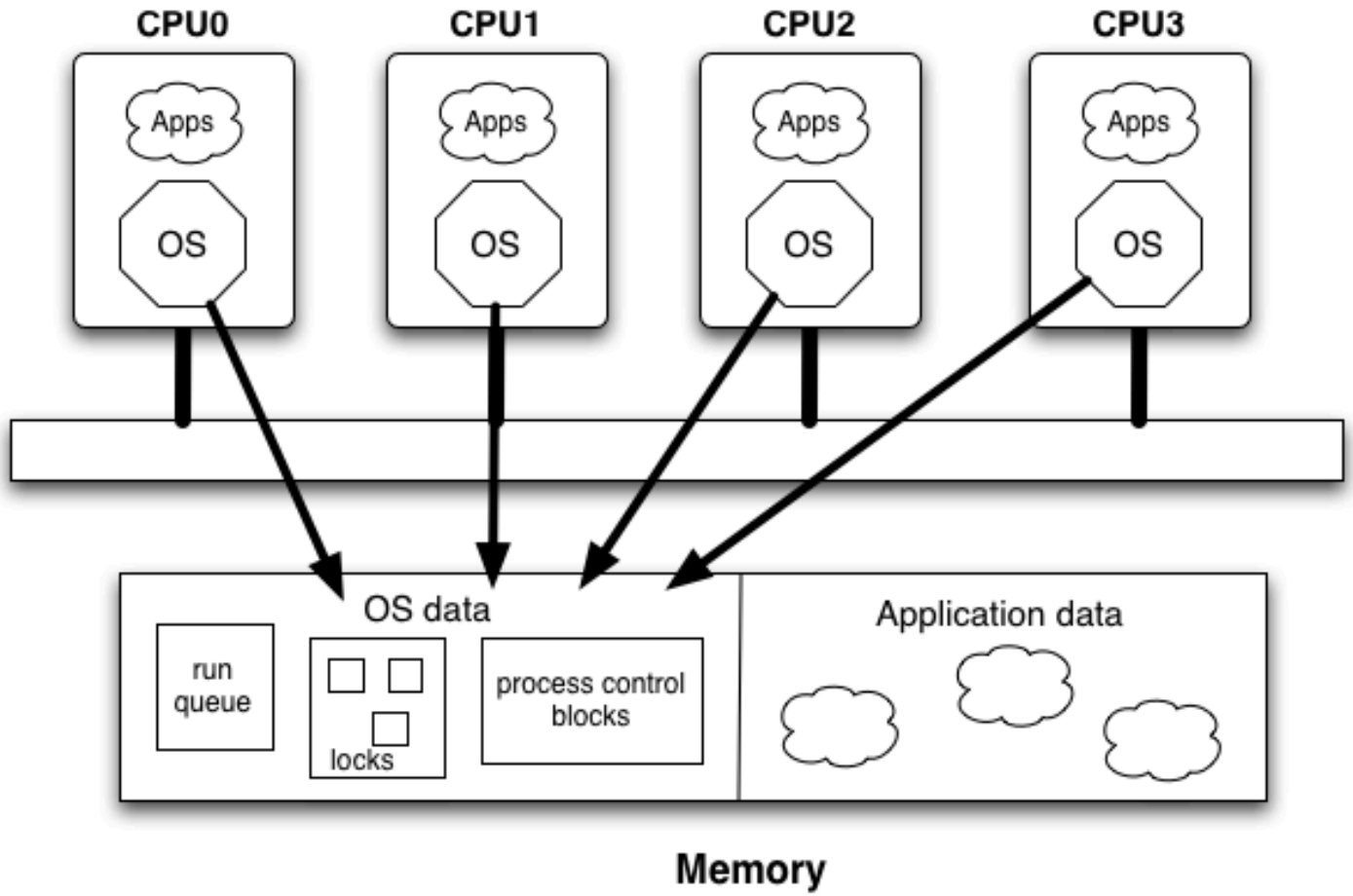
COMP9242
Advanced Operating Systems

S2/2011 Week 7:
Multiprocessors – Part 2

Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Funding and Supporting Members and Partners
Australian National University
UNSW THE UNIVERSITY OF NEW SOUTH WALES
NSW GOVERNMENT
State Government Victoria
THE UNIVERSITY OF MELBOURNE
THE UNIVERSITY OF SYDNEY
Queensland Government
Griffith UNIVERSITY
QUT Queensland University of Technology
THE UNIVERSITY OF QUEENSLAND AUSTRALIA

Multiprocessor OS



- Key design challenges:
  - Correctness of (shared) data structures
  - Scalability

Scalability of Multiprocessor OS

Remember Amdahl's law

- Serialisation prevents scalability
- Whenever application not running on core, scalability reduced

Sources of Serialisation:

- Locking
  - Waiting for a lock ➔ stalls self
  - Lock implementation:
    - Atomic operations lock bus ➔ stalls everyone
    - Cache coherence traffic loads bus ➔ slows down others
- Memory access
  - Relatively high latency to memory ➔ stalls self
- Cache
  - Processor stalled while cache line is fetched or invalidated
  - Limited by latency of interconnect round-trips
  - Performance depends on data size (cache lines) and contention (number of cores)

More Cache Issues

- False sharing
  - Unrelated data structs share the same cache line
  - Accessed from different processors
  - ➔ Cache coherence traffic and delay
- Cache line bouncing
  - Shared R/W on many processors
  - E.g: bouncing due to locks: each processor spinning on a lock brings it into its own cache
  - ➔ Cache coherence traffic and delay
- Cache misses
  - Potentially direct memory access
  - When does cache miss occur?
    - Application runs on new core
    - Cached memory has been evicted

Optimisation for Scalability

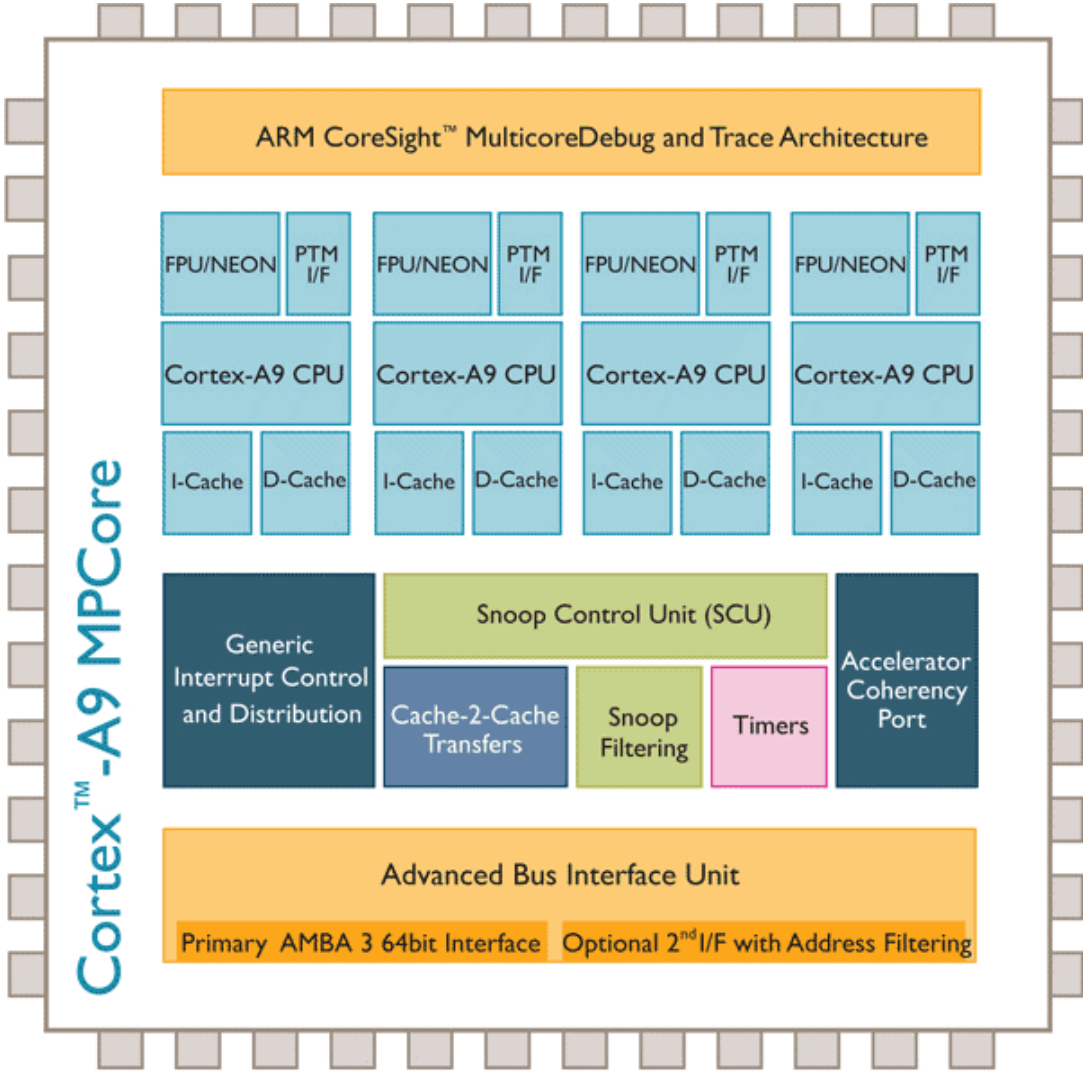
- Reduce amount of code in critical sections
  - Increases concurrency
  - Fine grained locking
    - Lock data not code
    - Tradeoff: more concurrency but more locking (and locking causes serialisation)
  - Lock free data structures
- Reduce false sharing
  - Pad data structures to cache lines
- Reduce cache line bouncing
  - Reduce sharing
  - E.g: MCS locks use local data
- Reduce cache misses
  - Affinity scheduling: run process on the core where it last ran.
  - Avoid cache pollution

Contemporary Multiprocessor Hardware

- Intel Nehalem: Beckton, Westmere
- AMD Opteron: Barcelona, Magny Cours
- ARM Cortex A9, A15 MPCore
- Oracle (Sun) UltraSparc T1,T2,T3,T4 (Niagara)

Scale and Structure

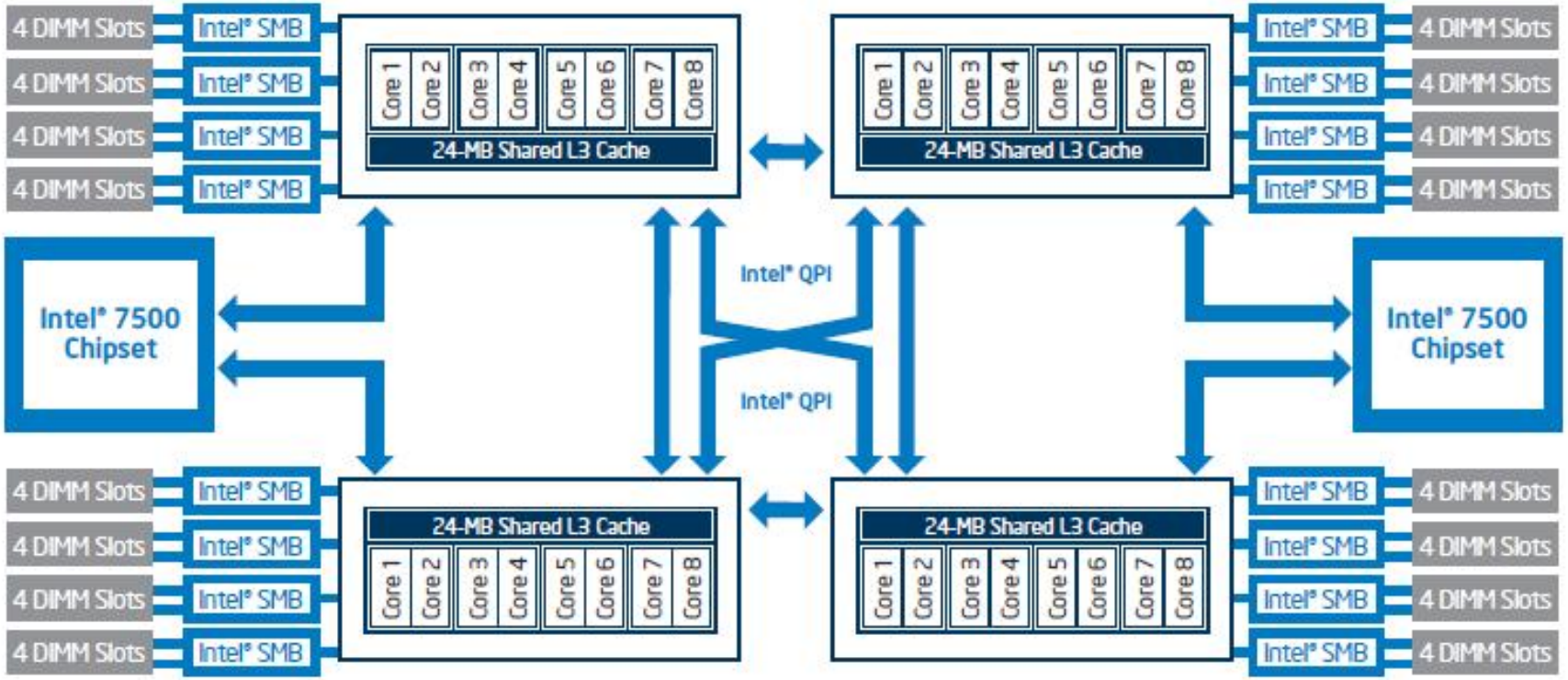
- ARM Cortex A9 MPCore



From http://www.arm.com/images/Cortex-A9-MP-core_Big.gif

Scale and Structure

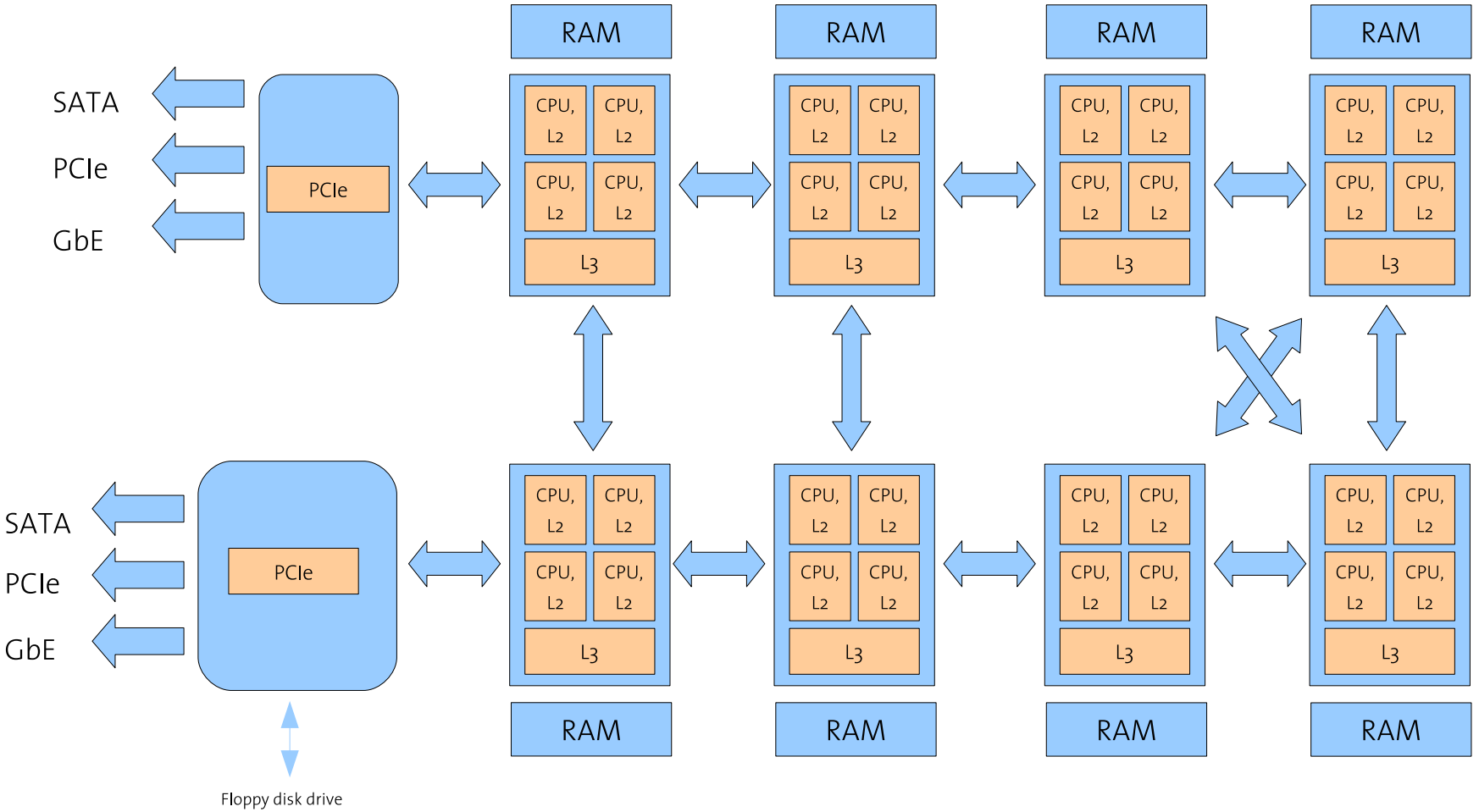
- Intel Nehalem



From www.dawnofthered.net/wp-content/uploads/2011/02/Nehalem-EX-architecture-detailed.jpg

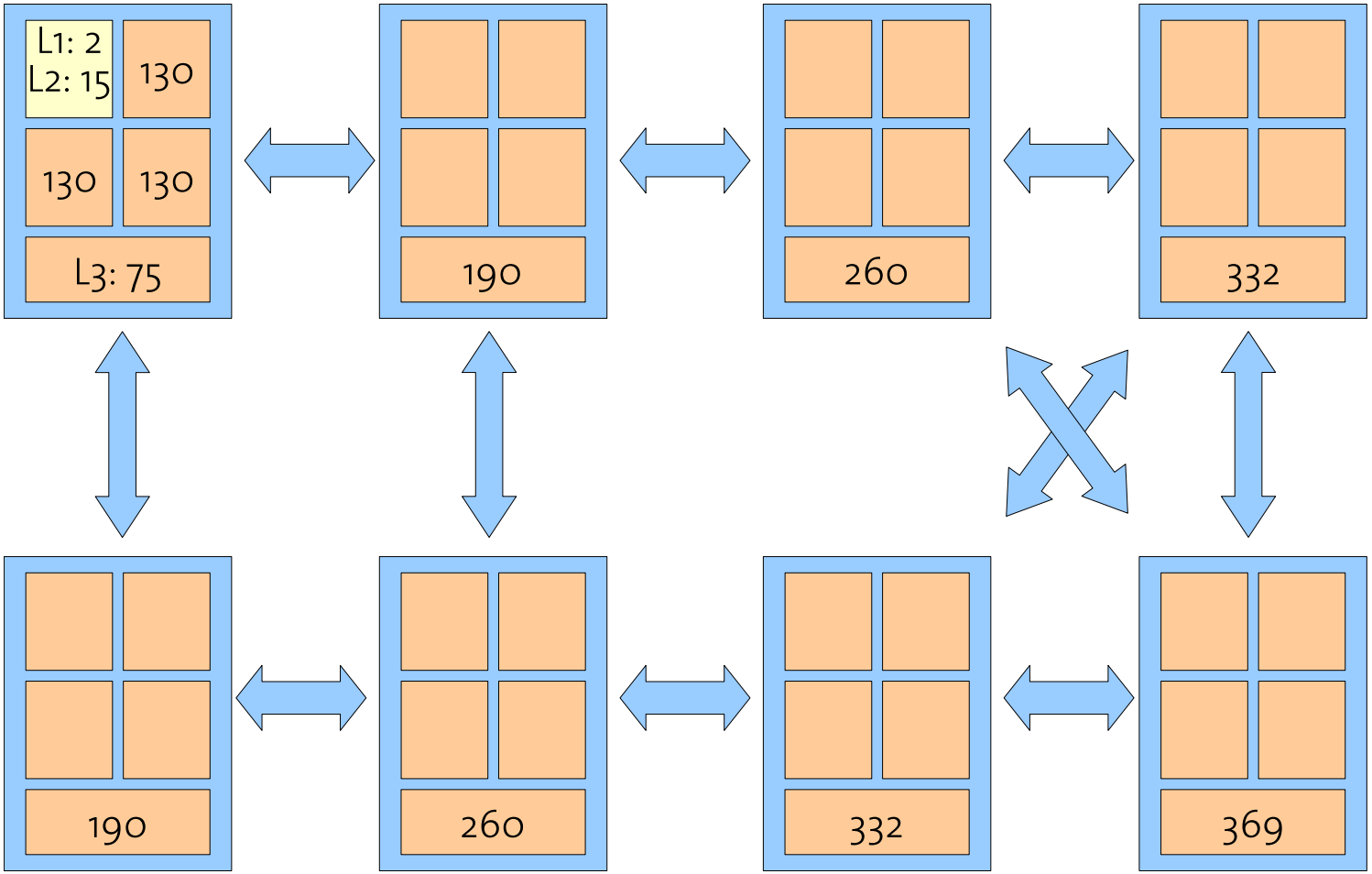
Interconnect

- AMD Barcelona



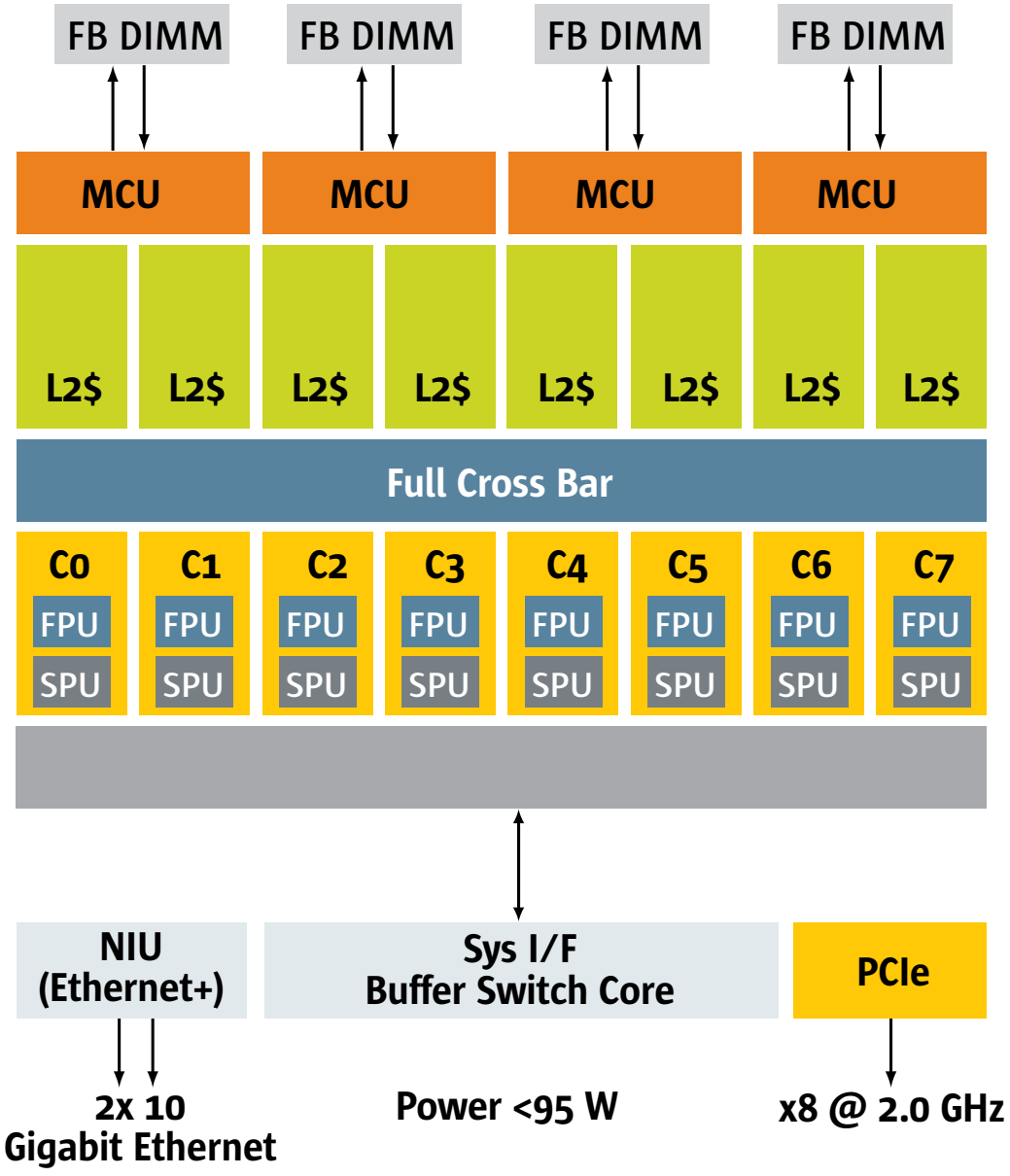
From www.sigops.org/sosp/sosp09/slides/baumann-slides-sosp09.pdf

Memory Locality and Caches



From www.systems.ethz.ch/education/past-courses/fall-2010/aos/lectures/wk10-multicore.pdf

Interprocessor Communication



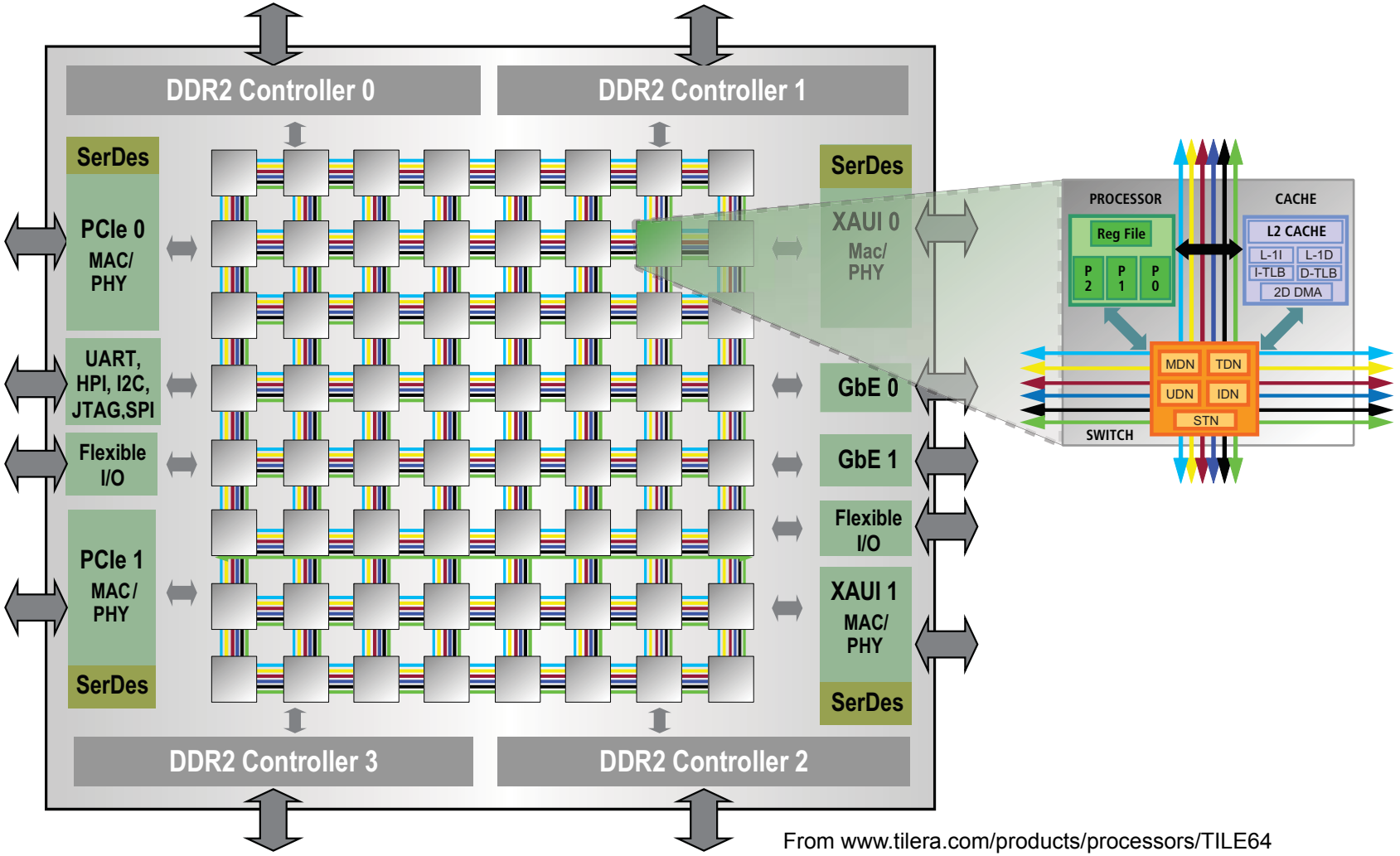
From Sun/Oracle

Experimental/Future Multiprocessor Hardware

- Intel SCC
- Microsoft Beehive
- Intel Polaris
- Tilera Tile64

Scale and Structure

- **Tilera Tile64**, Intel Polaris

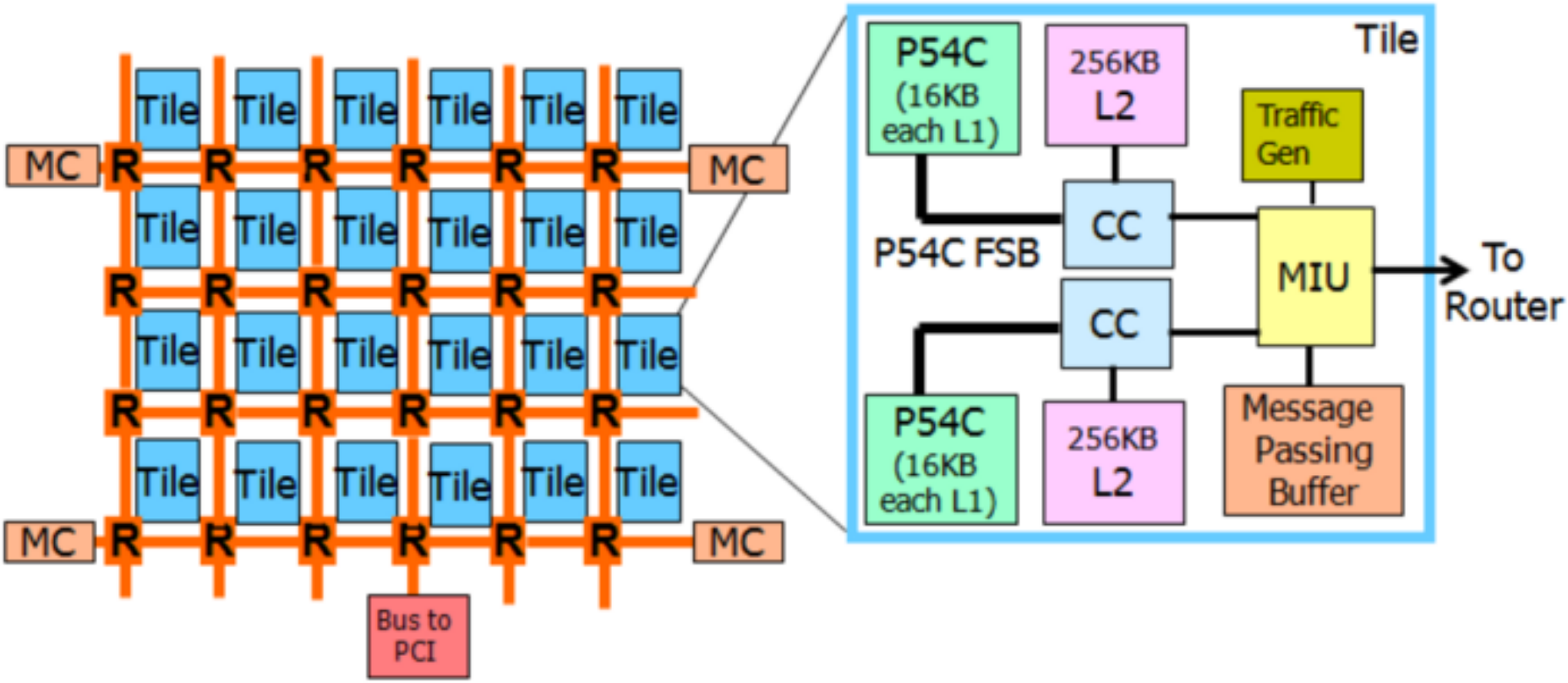


From www.tilera.com/products/processors/TILE64

Cache and Memory

NICTA

- Intel SCC

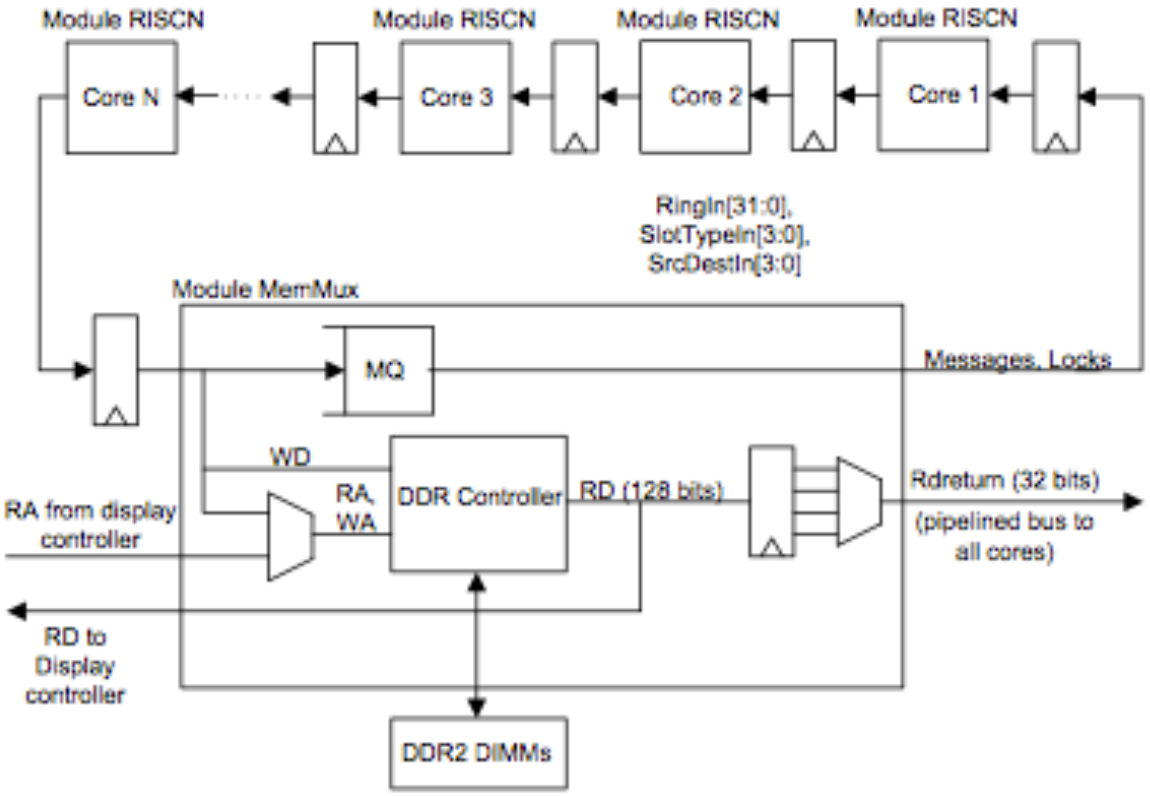


From techresearch.intel.com/spaw2/uploads/files/SCC_Platform_Overview.pdf

Interprocessor Communication

NICTA

- Beehive



From projects.csail.mit.edu/beehive/BeehiveV5.pdf

Summary

- Scalability
  - 100+ cores
  - Amdahl's law really kicks in
- NUMA
  - Also variable latencies due to topology and cache coherence
- Cache coherence may not be possible
  - Can't use it for locking
  - Shared data structures require explicit work
- Computer is a distributed system
  - Message passing
  - Consistency and Synchronisation
  - Fault tolerance
- Heterogeneity
  - Heterogeneous cores, memory, etc.
  - Properties of similar systems may vary wildly (e.g. interconnect topology and latencies between different AMD platforms)

OS Design for Modern (and future) Multiprocessors

- Avoid shared data
  - Performance issues arise less from lock contention than from data locality
- Explicit communication
  - Regain control over communication costs
  - Sometimes it's the only option
- Tradeoff: parallelism vs synchronisation
  - Synchronisation introduces serialisation
  - Make concurrent threads independent
- Allocate for locality
  - E.g. provide memory local to a core
- Schedule for locality
  - With cached data
  - With local memory
- Tradeoff: uniprocessor performance vs scalability

Design approaches

- Divide and conquer
  - Using virtualisation
  - Using exokernel
- Reduced sharing
  - By design
  - Brute force
- No sharing
  - Computer is a distributed system

Divide and Conquer

Disco

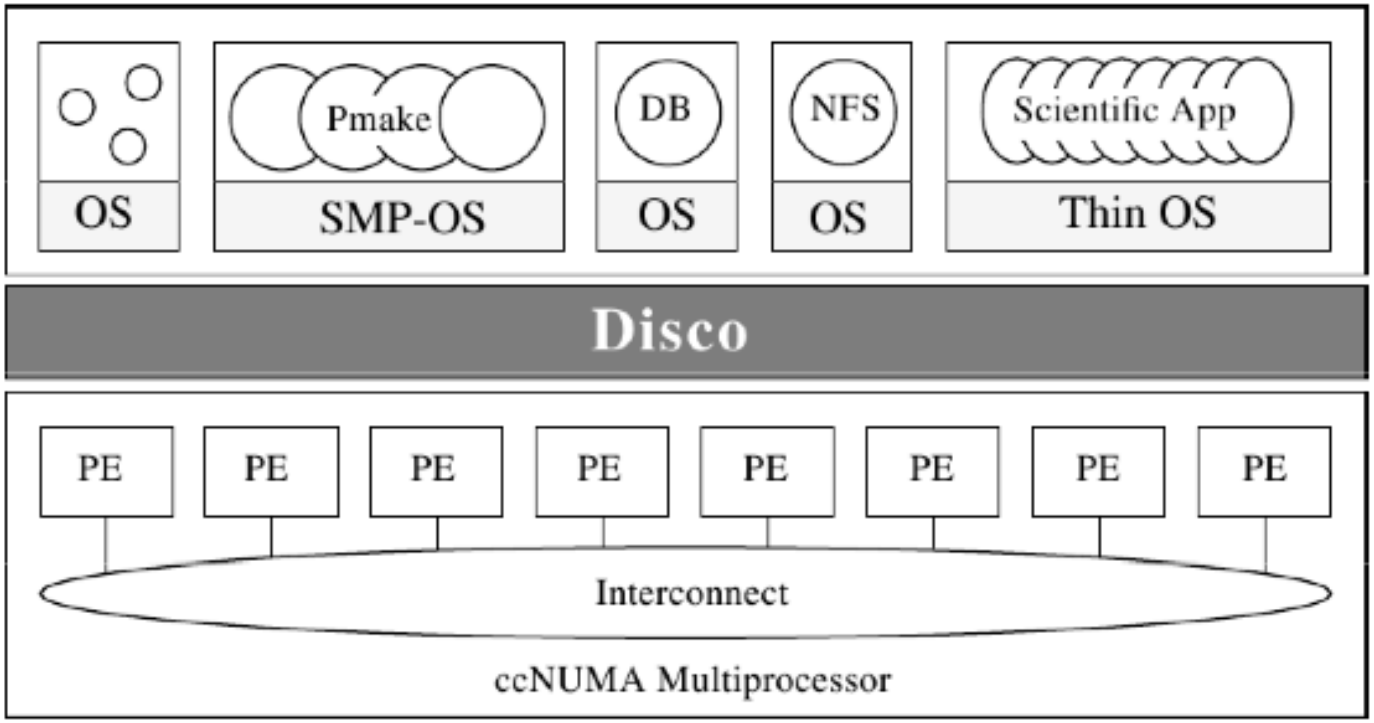
Running commodity OSes on scalable multiprocessors [Bugnion et al., 1997]
http://www-flash.stanford.edu/Disco/

- Scalability is too hard!

- Context:
  - ca. 1995, large ccNUMA multiprocessors appearing
  - Scaling OSes requires extensive modifications
- Idea:
  - Implement a scalable VMM
  - Run multiple OS instances
- VMM has most of the features of a scalable OS:
  - NUMA aware allocator
  - Page replication, remapping, etc.
- VMM substantially simpler/cheaper to implement
- Modern incarnations of this
  - Virtual servers (Amazon, etc.)
  - Research (Cerebrus)

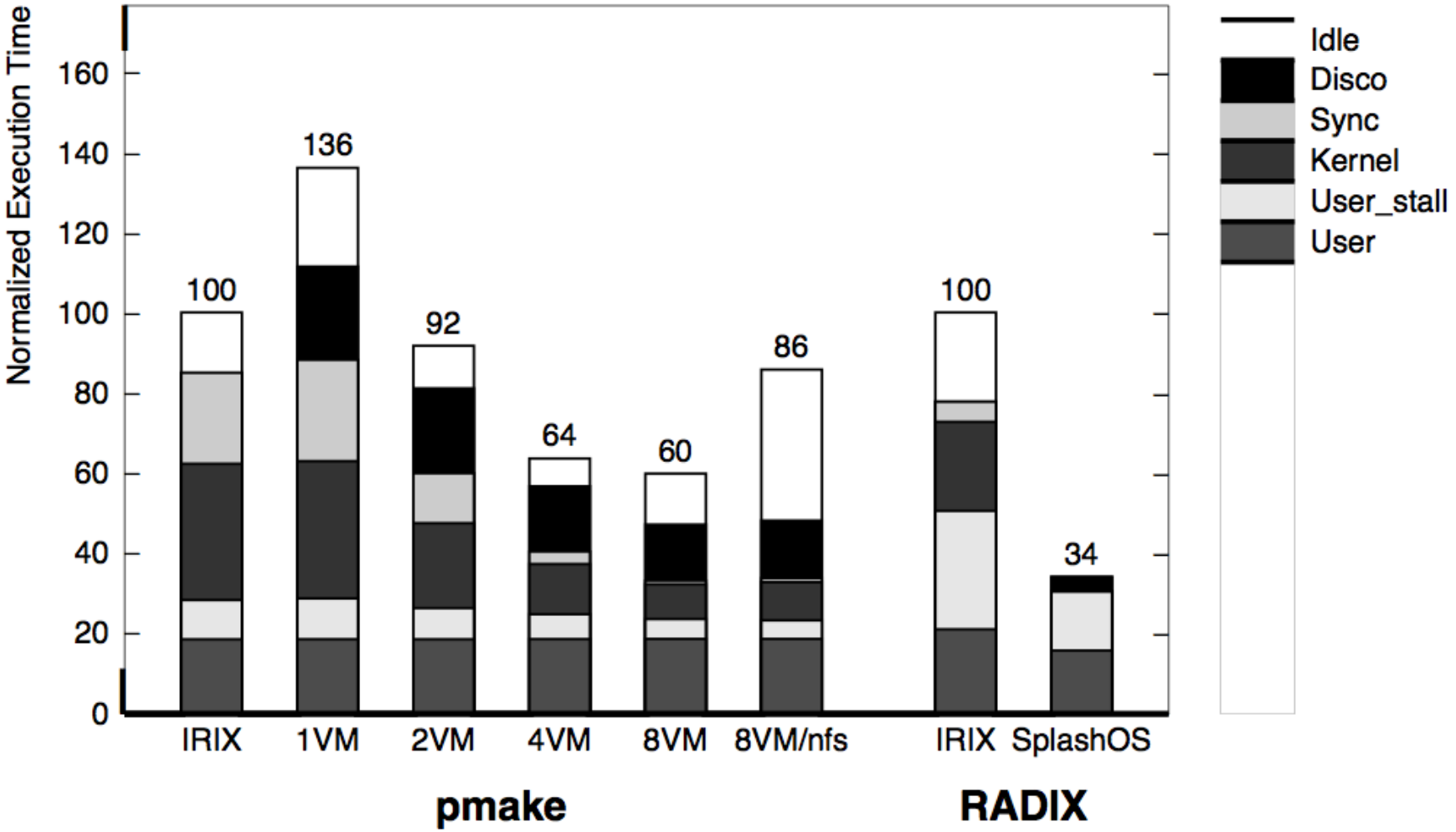
Disco Architecture

NICTA



[Bugnion et al., 1997]

Disco Performance



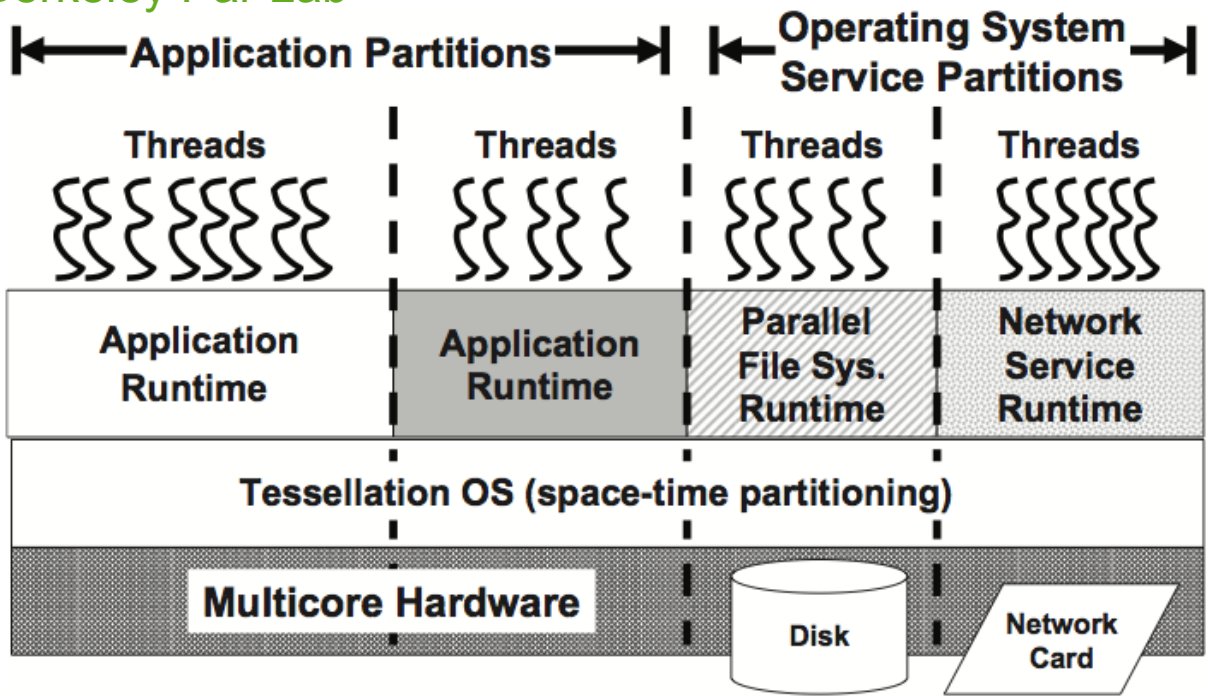
Space-Time Partitioning

Tessellation

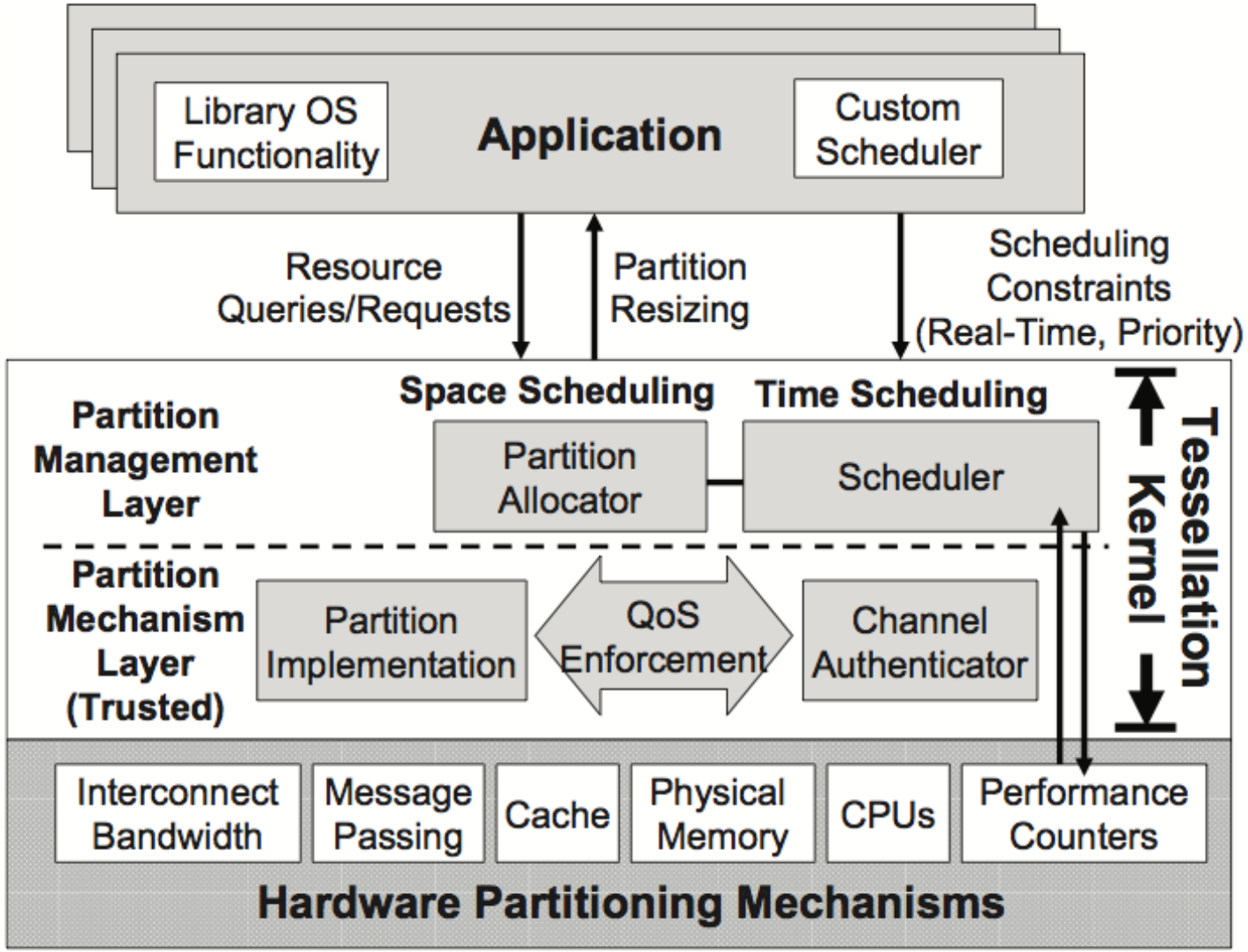
Tessellation: Space-Time Partitioning in a Manycore Client OS [Liu et al., 2010]
http://tessellation.cs.berkeley.edu/

- Space-Time partitioning
- 2-level scheduling

- Context:
  - 2009-… highly parallel multicore systems
  - Berkeley Par Lab



Tessellation



Reduce Sharing

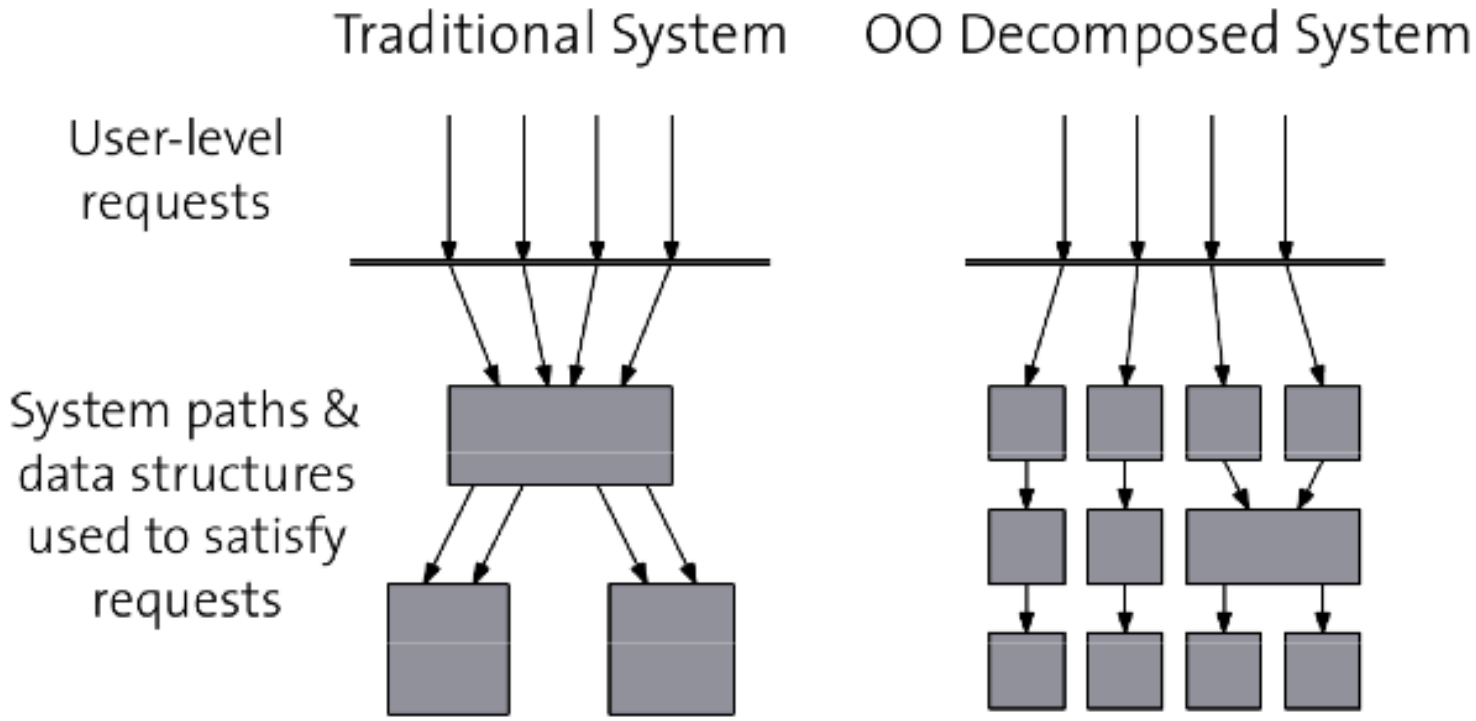
K42

Clustered Objects, Ph.D. thesis [Appavoo, 2005]
http://www.research.ibm.com/K42/

- Context:
  - 1997-2006: OS for ccNUMA systems
  - IBM, U Toronto (Tornado, Hurricane)
- Goals:
  - High locality
  - Scalability
- Object Oriented
  - Fine grained objects
- Clustered (Distributed) Objects
  - Data locality
- Deferred deletion (RCU)
  - Avoid locking
- NUMA aware memory allocator
  - Memory locality

K42: Fine-grained objects

NICTA

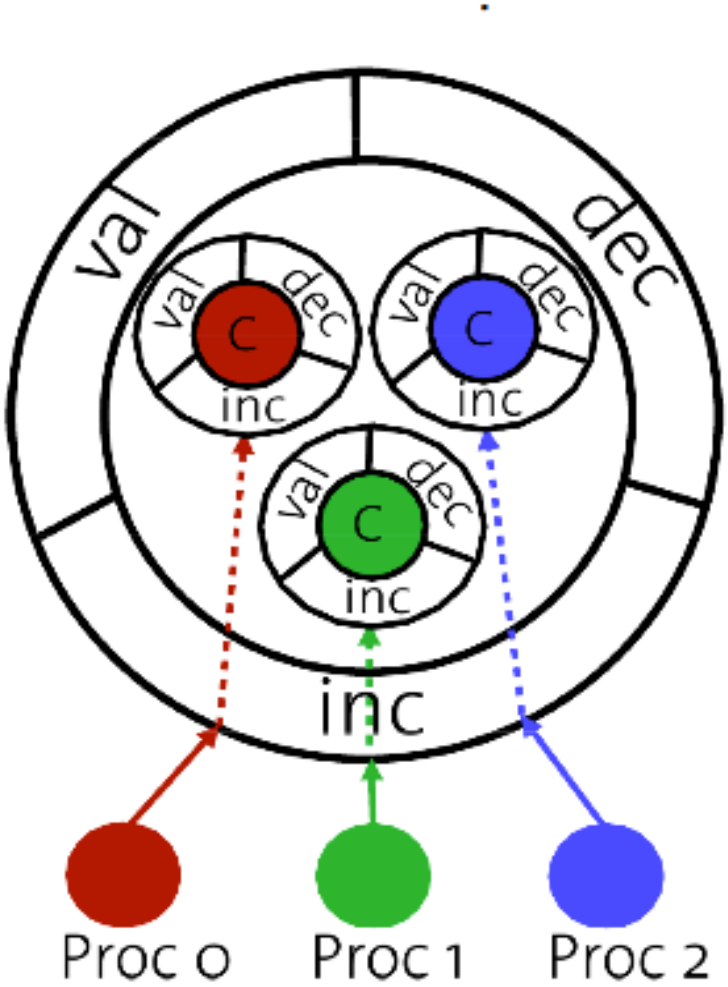


- much sharing

- much **less** sharing
- better performance

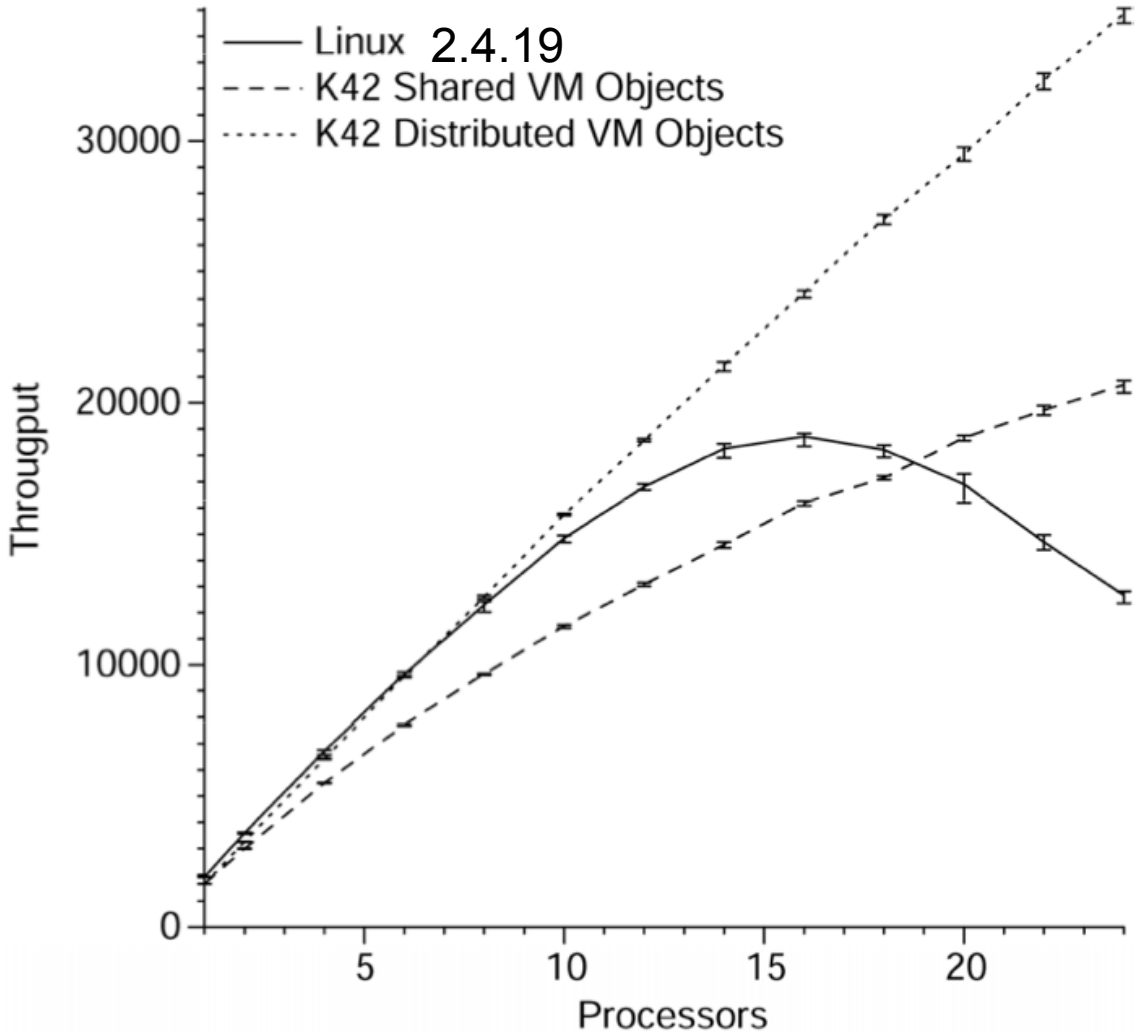
[Appavoo, 2005]

K42: Clustered objects



- Globally valid object reference
- Resolves to
  - Processor local representative
- Sharing, locking strategy local to each object
- Transparency
  - Eases complexity
  - Controlled introduction of locality
- Shared counter:
  - *inc*, *dec*: local access
  - *val*: communication
- Fast path:
  - Access mostly local structures

K42 Performance



Corey

- Context — Corey: An Operating System for Many Cores [Boyd-Wickizer et al., 2008] http://pdos.csail.mit.edu/corey
  - 2008, high-end multicore servers, MIT
- Goals:
  - Application control of OS sharing
- Address Ranges
  - Control private per core and shared address spaces
- Kernel Cores
  - Dedicate cores to run specific kernel functions
- Shares
  - Lookup tables for kernel objects allow control over which object identifiers are visible to other cores.
- Linux scalability (2010 – scale Linux to 48 cores)
  - sloppy counters, per-core data structs, fine-grained lock, lock free, cache lines : 3002 lines of code changed
  - no scalability reason to give up on traditional operating system organizations just yet.

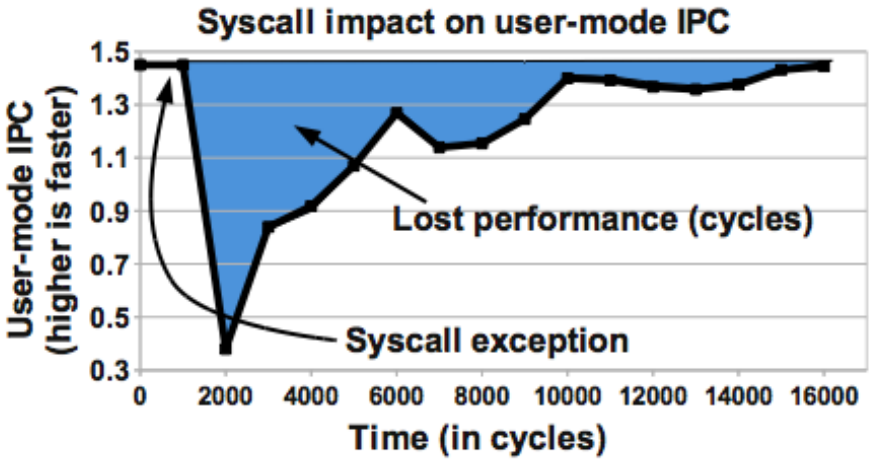
An Analysis of Linux Scalability to Many Cores [Boyd-Wickizer et al., 2010]

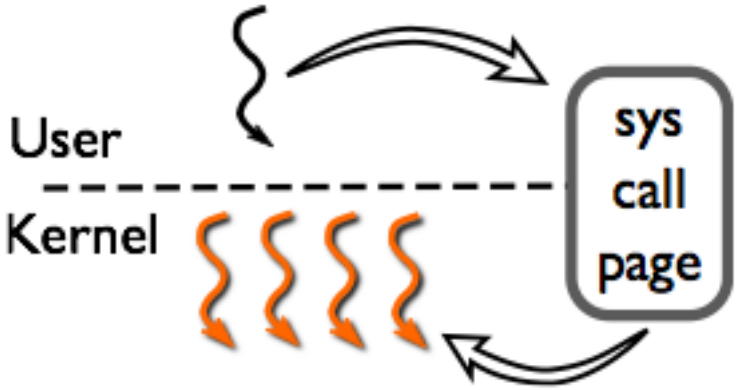
FlexSC

- Context:

  FlexSC: Flexible System Call Scheduling with Exception-Less System Calls [Soares and Stumm., 2010]

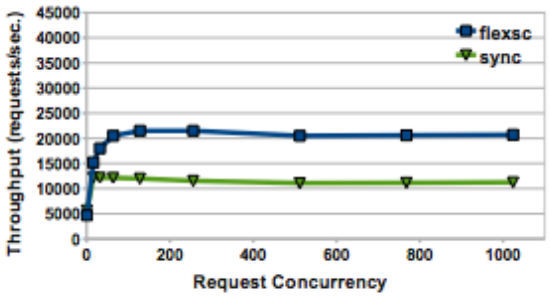
  - 2010, commodity multicores
  - U Toronto
- Goal:
  - Reduce context switch overhead of system calls
- Syscall context switch:
  - Usual mode switch overhead
  - But: cache and TLB pollution!
- Asynchronous system calls
  - Batch system calls
  - Run them on dedicated cores
- FlexSC-Threads
  - M on N
  - M >> N



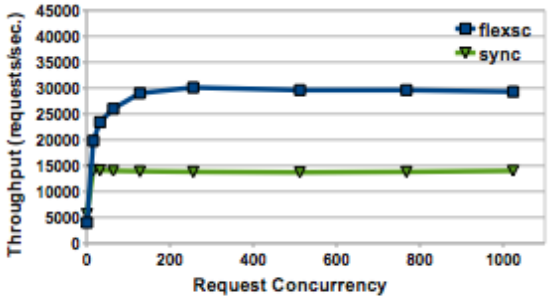


FlexSC Results

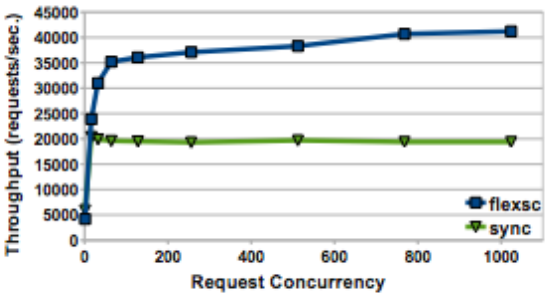
Syscall	Instructions	Cycles	IPC	i-cache	d-cache	L2	L3	d-TLB
`stat`	4972	13585	0.37	32	186	660	2559	21
`pread`	3739	12300	0.30	32	294	679	2160	20
`pwrite`	5689	31285	0.18	50	373	985	3160	44
`open+close`	6631	19162	0.34	47	240	900	3534	28
`mmap+munmap`	8977	19079	0.47	41	233	869	3913	7
`open+write+close`	9921	32815	0.30	78	481	1462	5105	49



(a) 1 Core



(b) 2 Cores

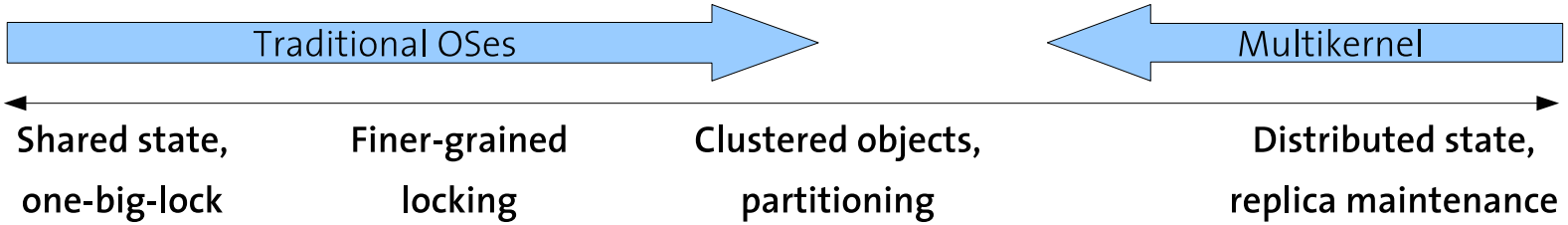


(c) 4 Cores

No sharing

NICTA

- Multikernel
  - Barrelfish
  - fos: factored operating system

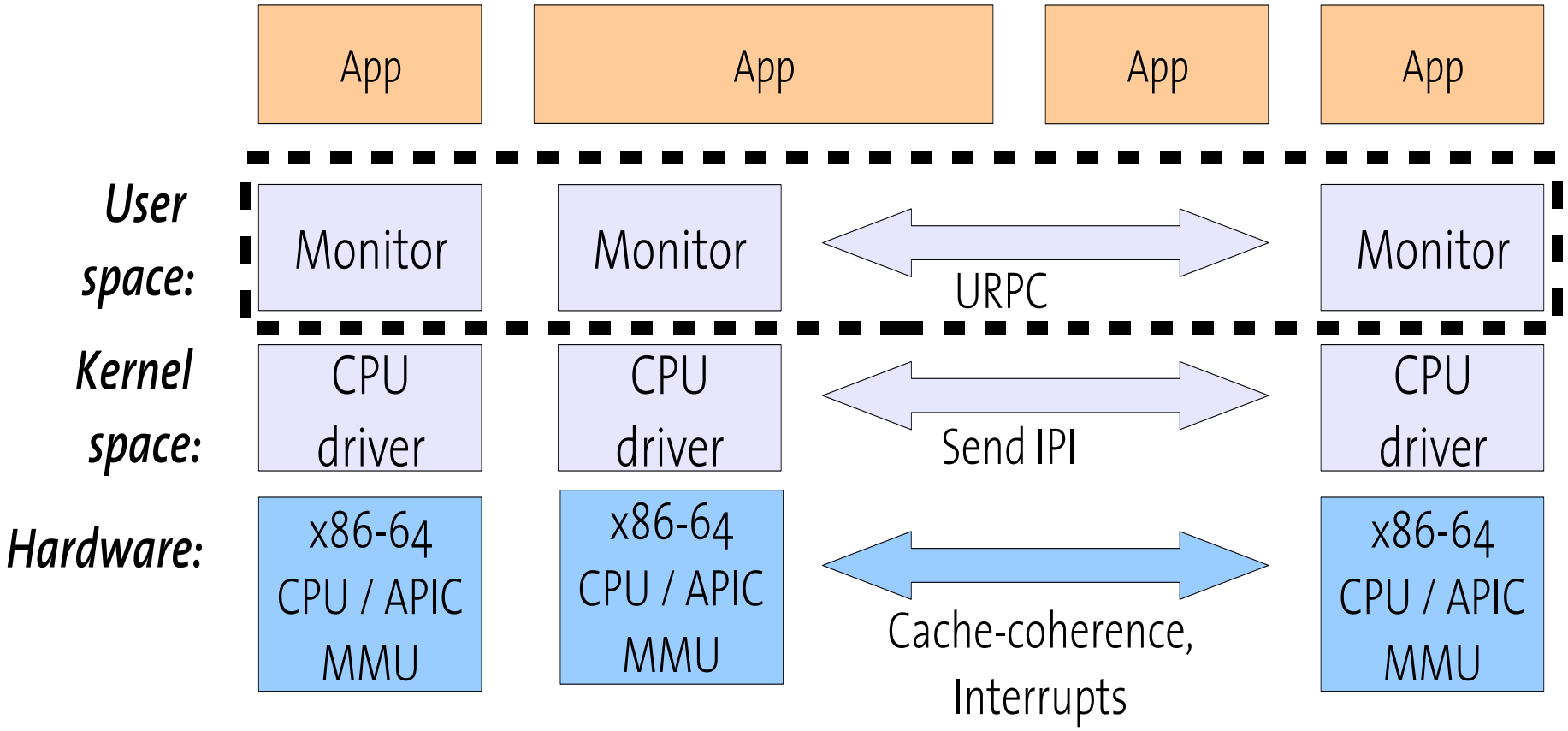


The Multikernel: A new OS architecture for scalable multicore systems [Baumann et al., 2009]
http://www.barrelfish.org/

Barrelfish

- Context: The Multikernel: A new OS architecture for scalable multicore systems [Baumann et al., 2009] http://www.barrelfish.org/
  - 2007 large multicore machines appearing
  - 100s of cores on the horizon
  - NUMA (cc and non-cc)
  - ETH Zurich and Microsoft
- Goals:
  - Scale to many cores
  - Support and manage heterogeneous hardware
- Approach:
  - Structure OS as *distributed system*
- Design principles:
  - Interprocessor communication is explicit
  - OS structure hardware neutral
  - State is replicated
- Microkernel
  - Similar to seL4: capabilities

Barrelfish


App
App
App
App
User space:
Monitor
Monitor
URPC
Monitor
Kernel space:
CPU driver
CPU driver
Send IPI
CPU driver
Hardware:
x86-64 CPU / APIC MMU
x86-64 CPU / APIC MMU
Cache-coherence, Interrupts
x86-64 CPU / APIC MMU

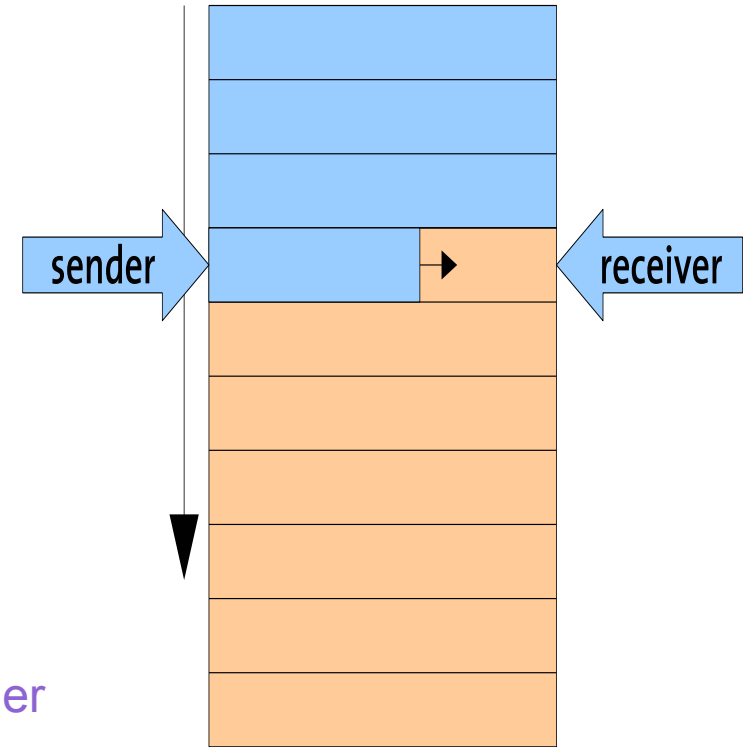
Barrelfish: Replication

- Kernel + Monitor:
  - Only memory shared for message channels
- Monitor:
  - Collectively coordinate system-wide state
- System-wide state:
  - Memory allocation tables
  - Address space mappings
  - Capability lists
- What state is replicated in Barrelfish
  - Capability lists
- Consistency and Coordination
  - Retype: two-phase commit to globally execute operation in order
  - Page (re/un)mapping: one-phase commit to synchronise TLBs

Barrelfish: Communication

NICTA

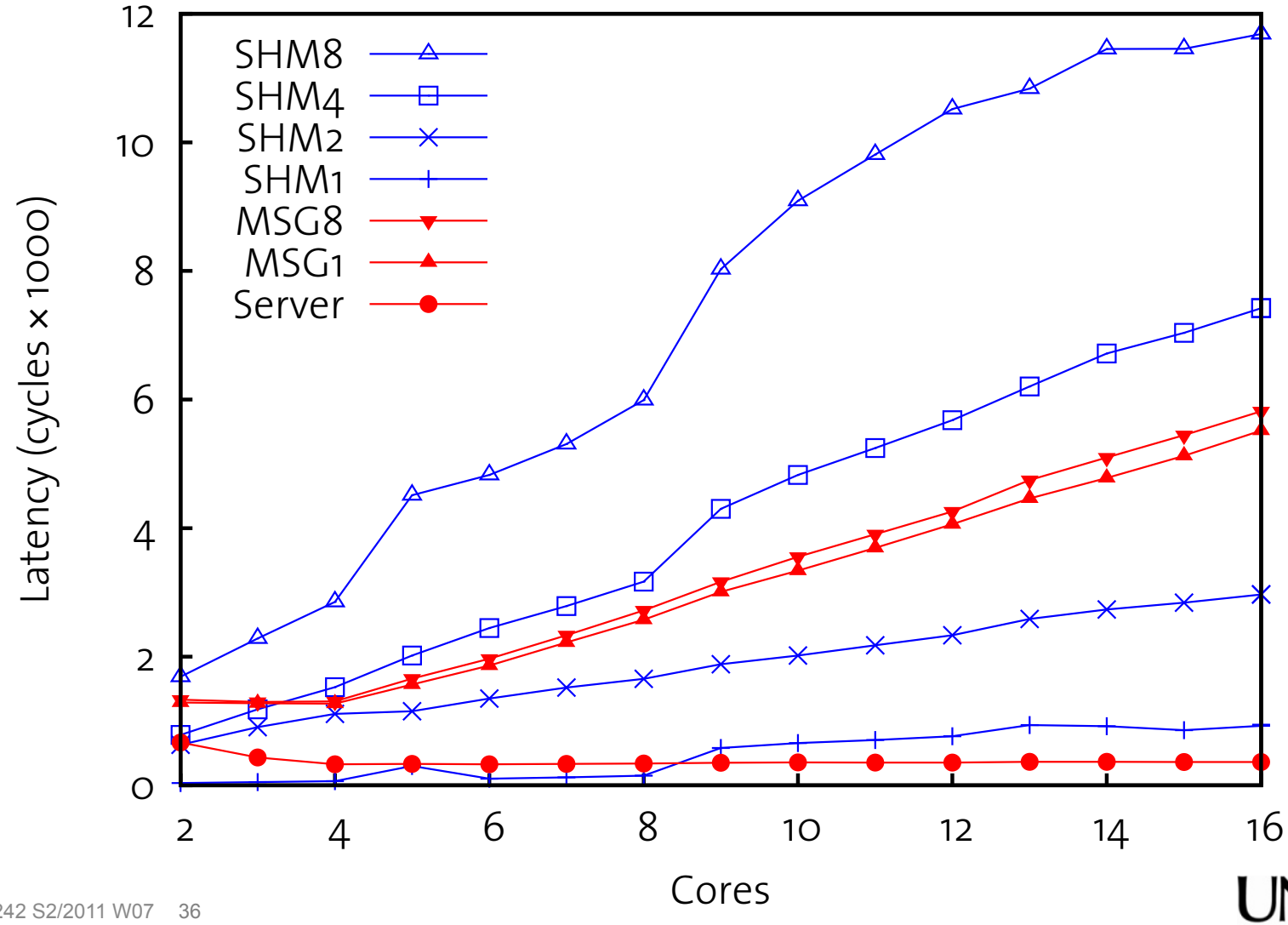
- Different mechanisms:
  - Intra-core
    - Kernel endpoints
  - Inter-core
    - URPC
- URPC
  - Uses cache coherence + polling
  - Shared bufffer
    - Sender writes a cache line
    - Receiver polls on cache line
    - (last word so no part message)
  - Polling?
    - Cache only changes when sender writes, so poll is cheap
    - Switch to block and IPI if wait is too long.



Barrelfish: Results

NICTA

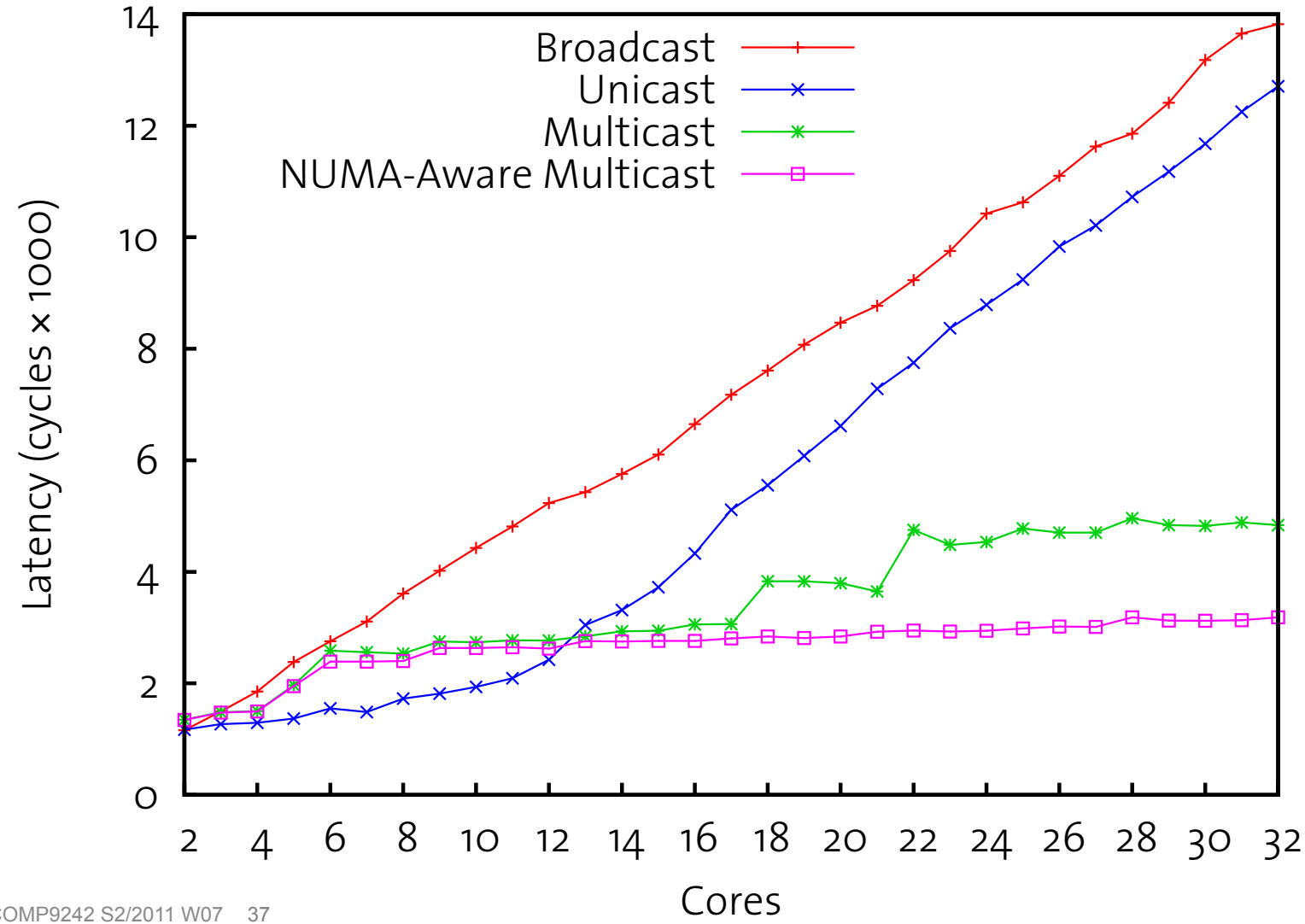
- Message passing vs caching



Barrelfish: Results

NICTA

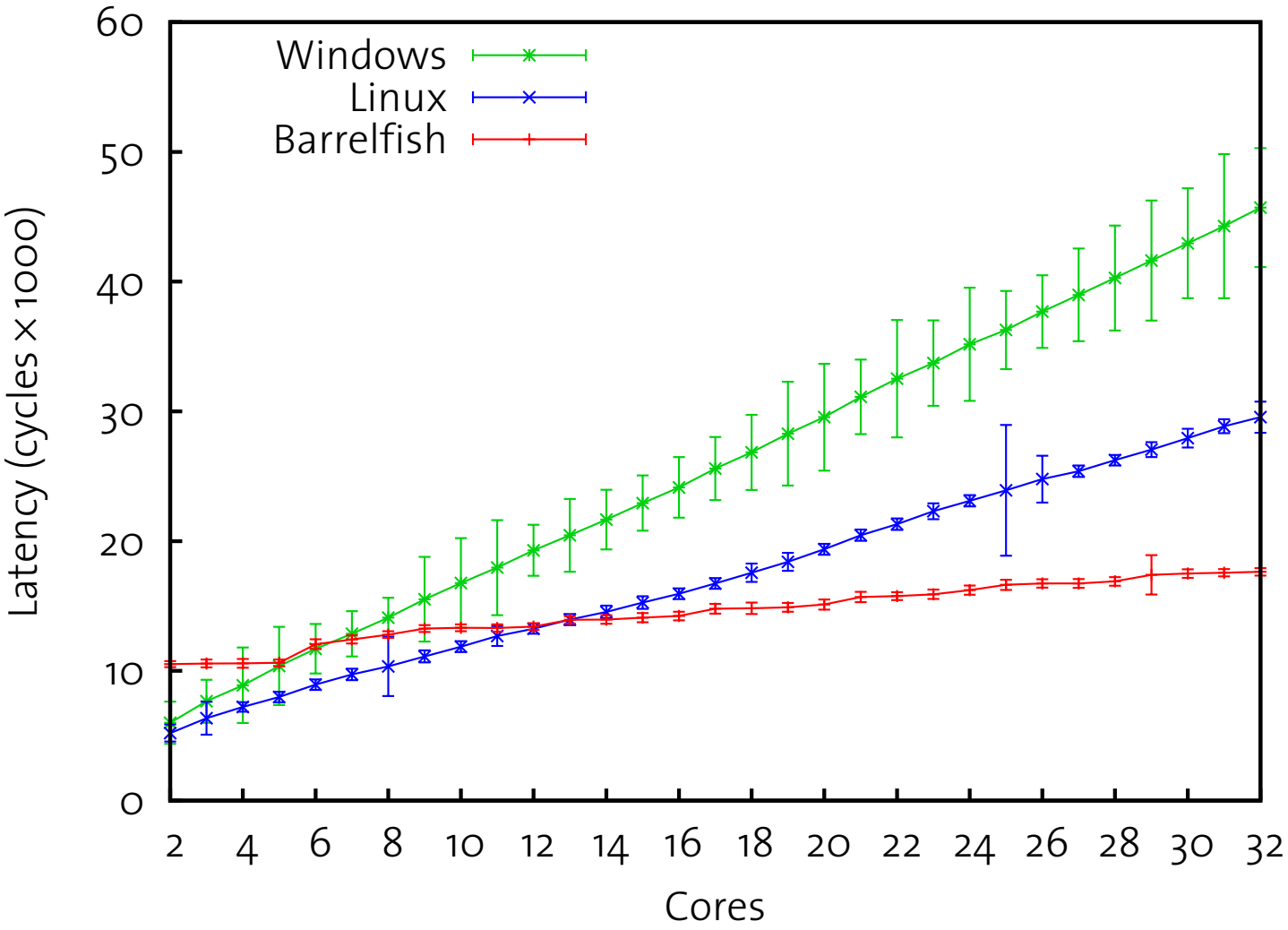
- Broadcast vs Multicast



Barrelfish: Results

NICTA

- TLB shootdown



UNSW
THE UNIVERSITY OF NEW SOUTH WALES

Summary

- Trends in multicore
  - Scale (100+ cores)
  - NUMA
  - No cache coherence
  - Distributed system
  - Heterogeneity
- OS design guidelines
  - Avoid shared data
  - Explicit communication
  - Locality
- Approaches to multicore OS
  - Partition the machine (Disco, Tessellation)
  - Reduce sharing (K42, Corey, Linux, FlexSC)
  - No sharing (Barrelfish, fos)