COMP9242
Advanced Operating Systems

S2/2014 Week 10:
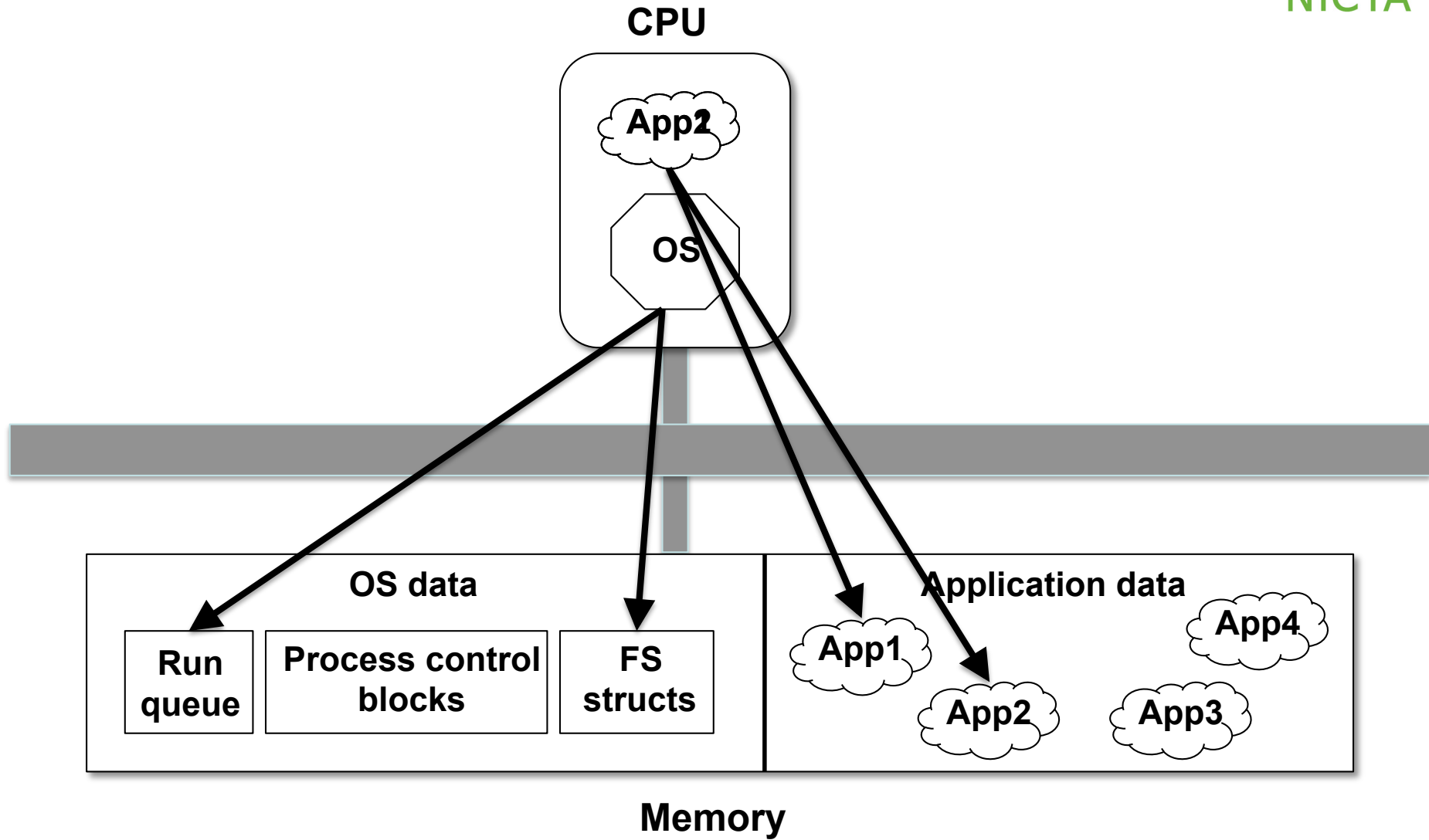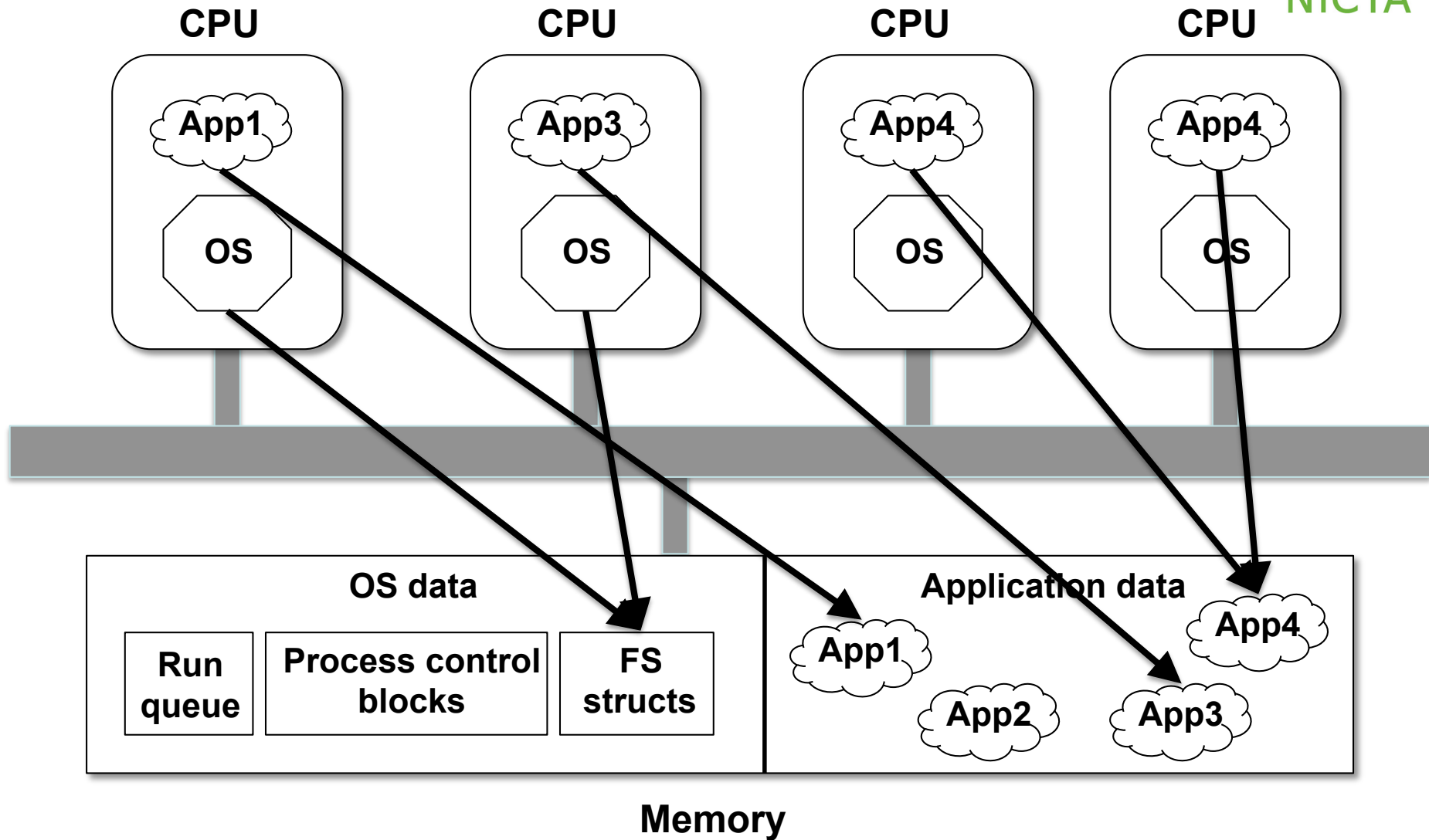# Multiprocessor OS

Overview

- Multiprocessor OS
  - Scalability (Review)

- Multiprocessor Hardware
  - Contemporary systems
  - Experimental and Future systems

- OS design for Multiprocessors
  - Guidelines
  - Design approaches
    - Divide and Conquer
    - Reduce Sharing
    - No Sharing

UNSW
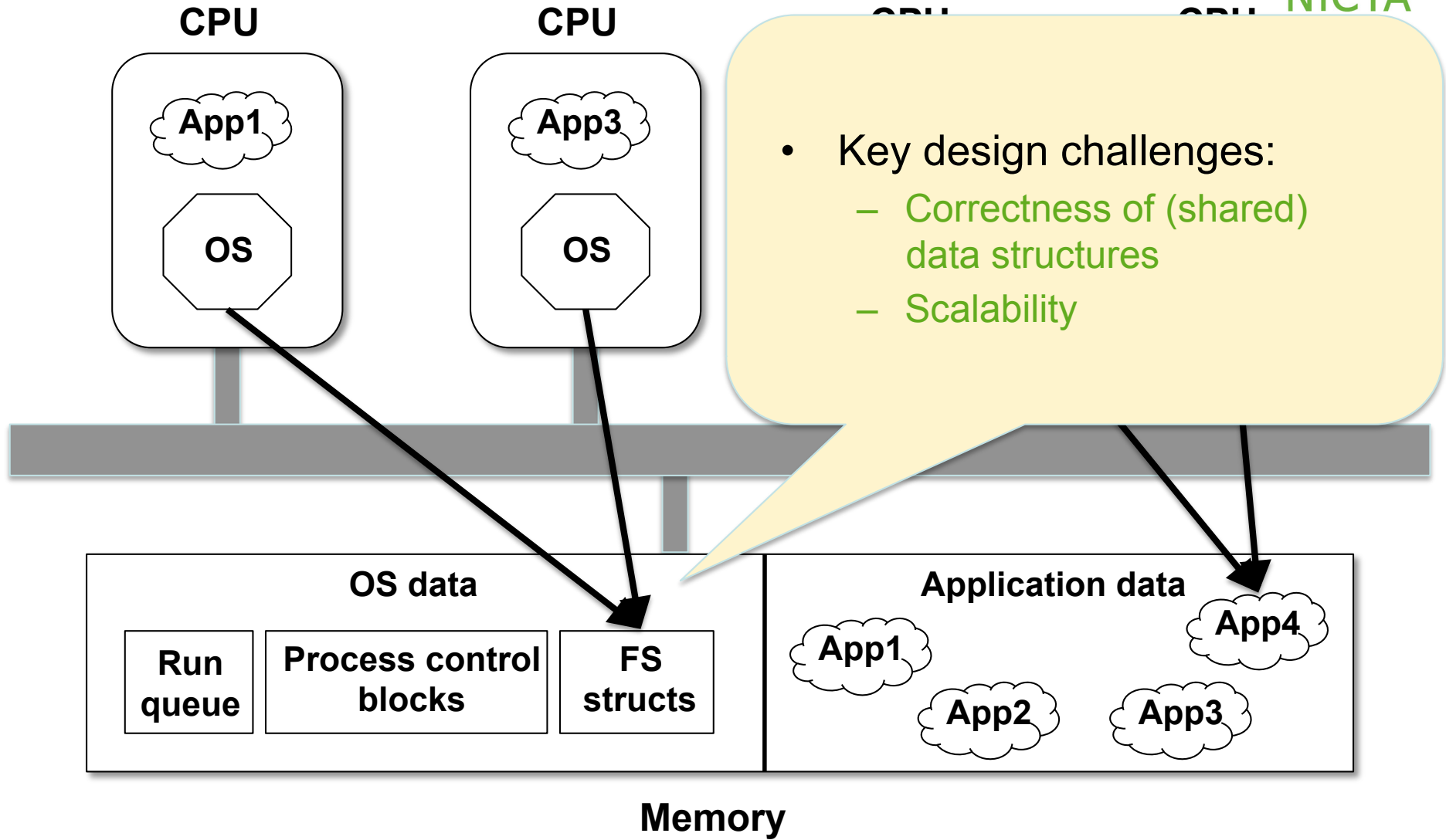THE UNIVERSITY OF NEW SOUTH WALES

MULTIPROCESSOR OS

Uniprocessor OS

CPU

App2

OS

OS data

Run queue	Process control blocks	FS structs

Application data

App1

App2

App3

App4

Memory

Multiprocessor OS

Multiprocessor OS

CPU   CPU

App1

OS

App3

OS

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

OS data

Run queue	Process control blocks	FS structs

Application data

App1

App2

App4

App3

Memory

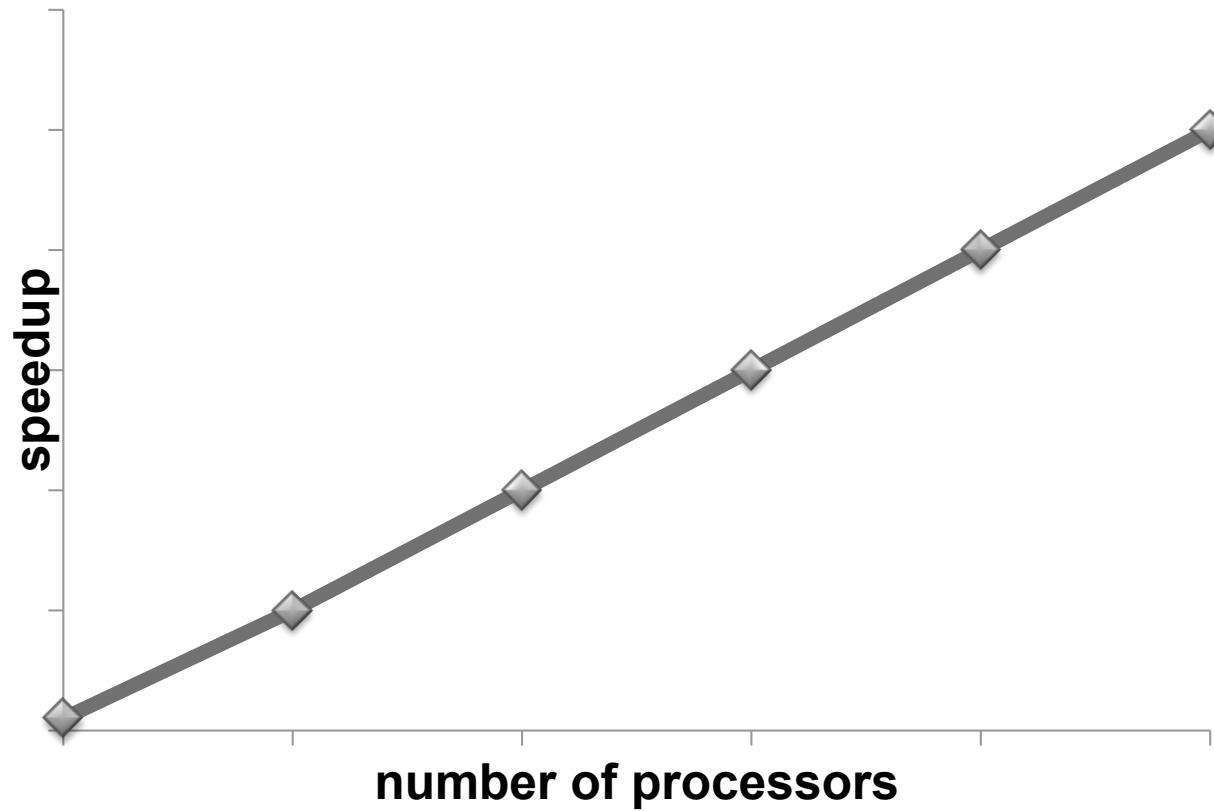# Correctness of Shared Data

- Concurrency control
  - Locks
  - Semaphores
  - Transactions
  - Lock-free data structures

- We know how to do this:
  - In the application
  - In the OS
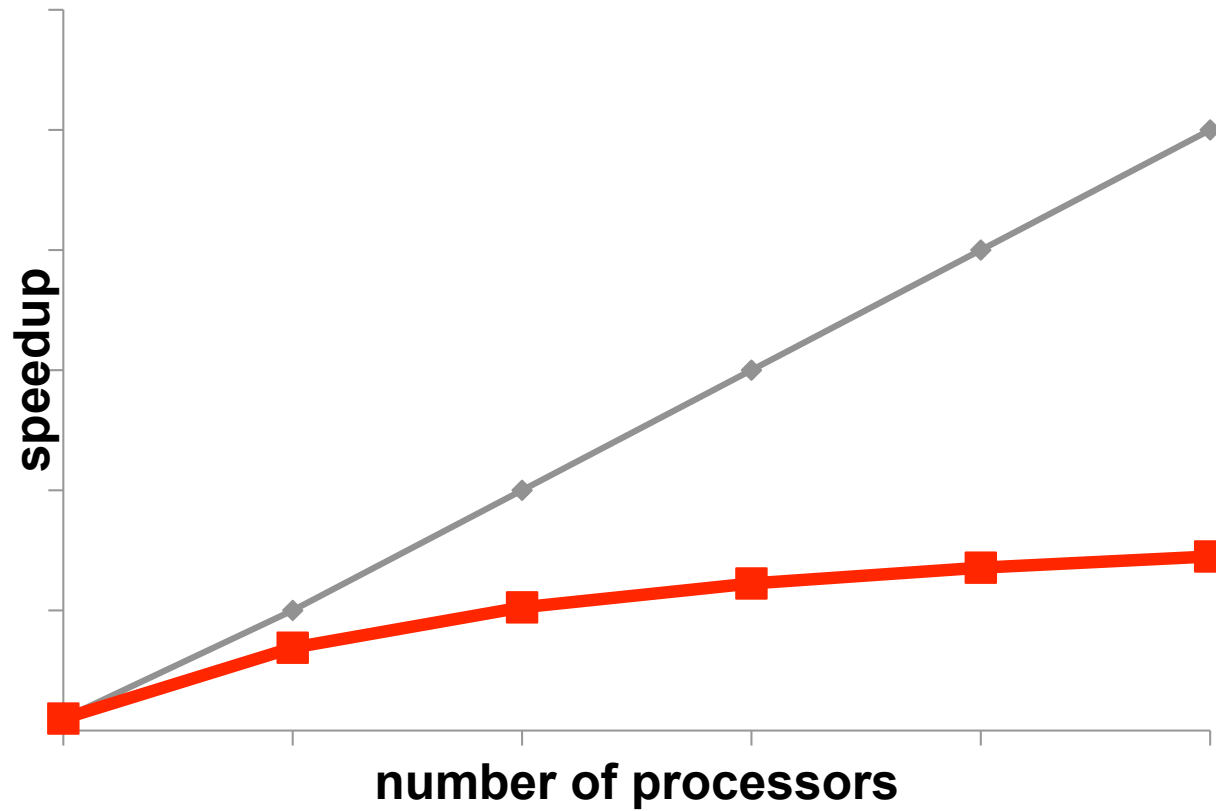
Scalability

Speedup as more processors added

Ideal

speedup vs number of processors

Scalability

Speedup as more processors added

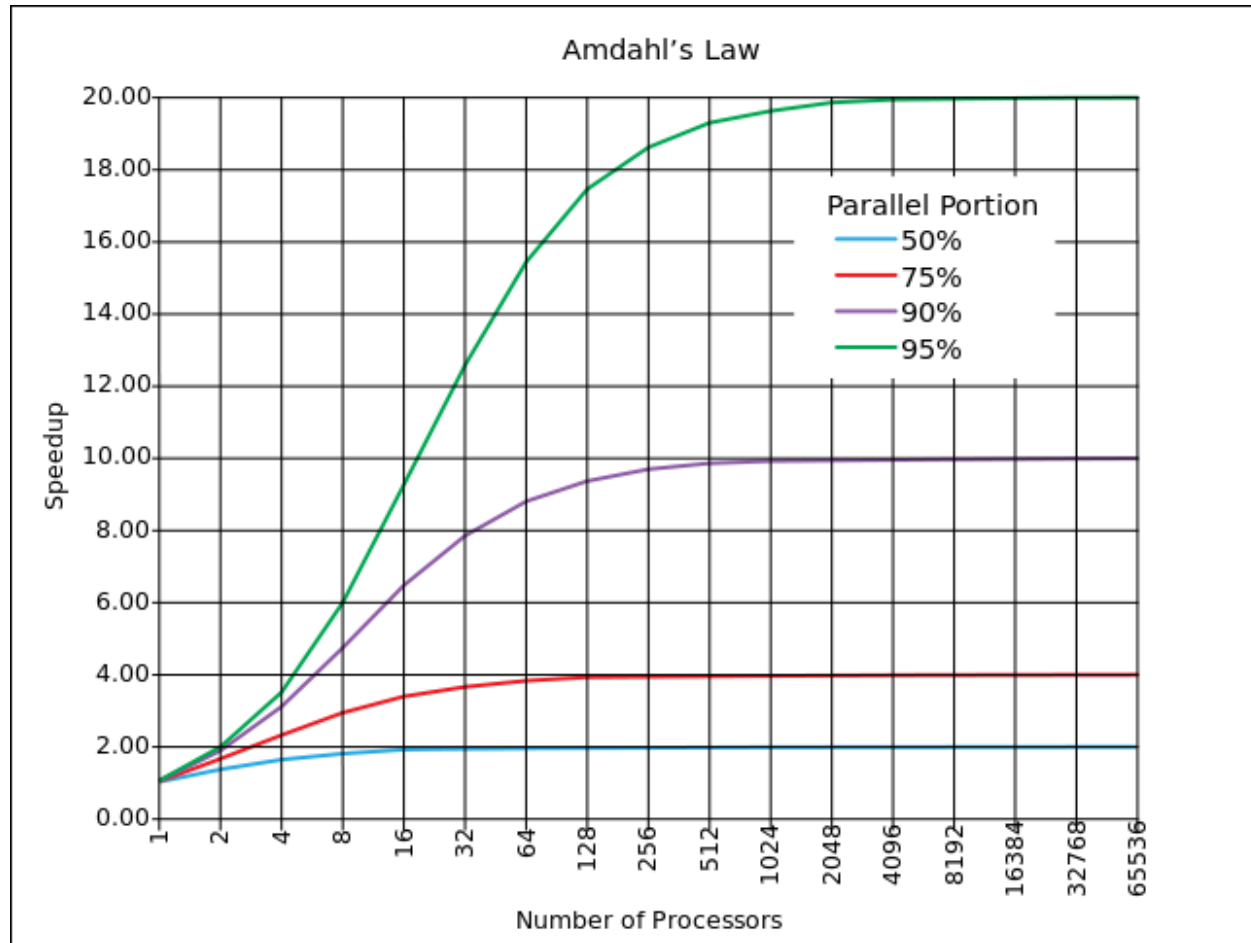**Reality**

speedup

number of processors

Scalability and Serialisation

Remember Amdahl's law

- – Serial (non-parallel) portion: when application not running on all cores
- – Serialisation prevents scalability

$$S(N) = \frac{1}{(1-P) + \dfrac{P}{N}}$$

From http://en.wikipedia.org/wiki/File:AmdahlsLaw.svg

Serialisation

Where does serialisation show up?

- Application (e.g. access shared app data)
- OS (e.g. performing syscall for app) How much time is spent in OS?

Sources of Serialisation:

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

UNSW
THE UNIVERSITY OF NEW SOUTH WALES

More Cache-related Serialisation

- 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 ➜ stalls self
  - When does cache miss occur?
    - Application runs on new core
    - Cached memory has been evicted

MULTIPROCESSOR HARDWARE

Multi-What?

- Multiprocessor, SMP
  - >1 separate processors, connected by off chip bus

- Multicore
  - >1 processing cores in a single processor, connected by on chip bus

- Multithread, SMT
  - >1 hardware threads in a single core

- Multicore + Multiprocessor
  - >1 multicore processors
  - >1 multicore dies in a package

NICTA

UNSW
THE UNIVERSITY OF NEW SOUTH WALES

Interesting Properties of Multiprocessors

- Structure
  - How many cores and processors are there
  - What kinds of cores and processors are there
  - How are they organised

- Interconnect
  - How are the cores and processors connected

- Memory Locality and Caches
  - Where is the memory
  - What is the cache architecture

- Interprocessor Communication
  - How do cores and processors send messages to each other
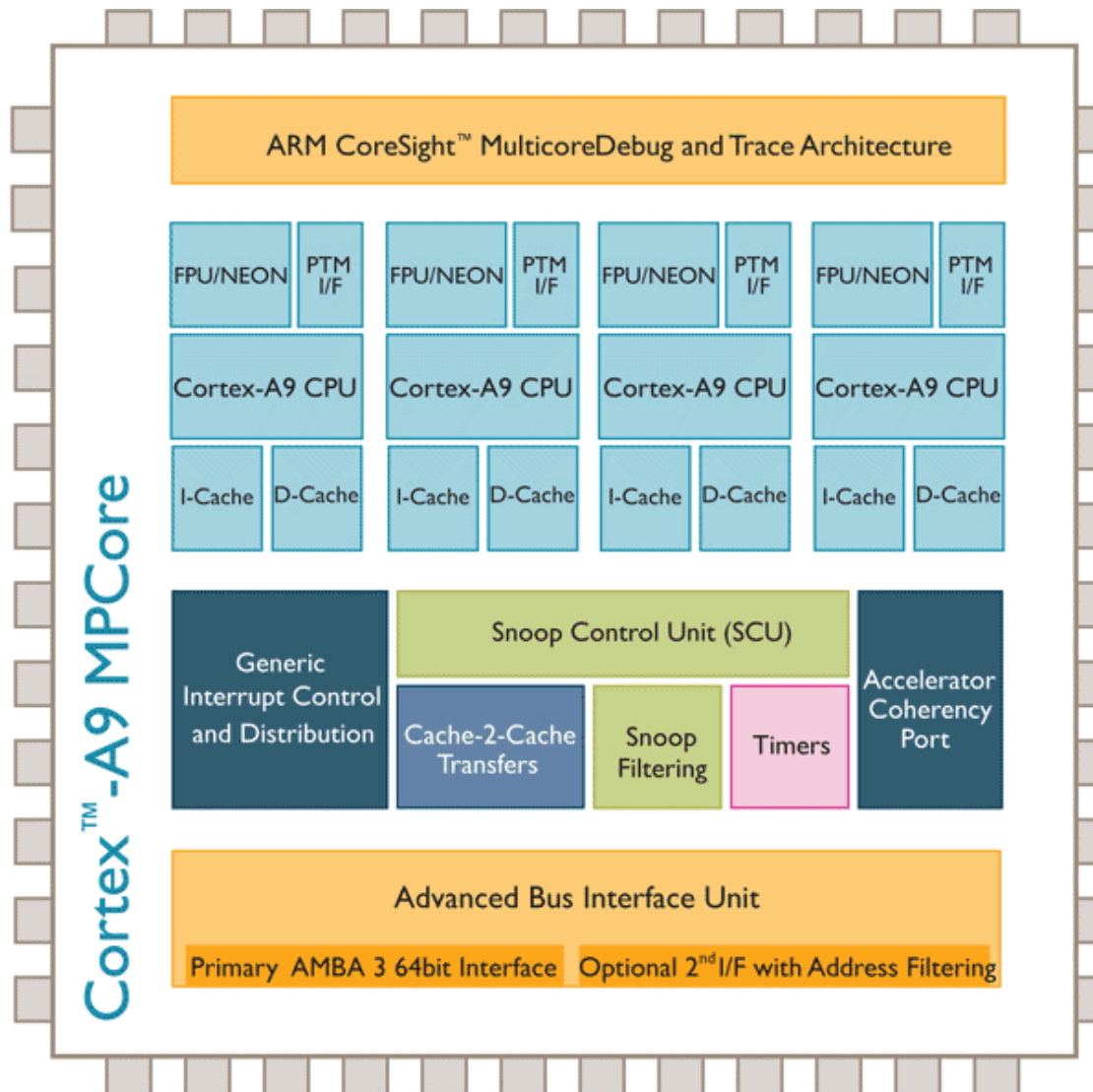
Contemporary Multiprocessor Hardware

- Intel:
  - Nehalem, Westmere: 10 core, QPI
  - Sandy Bridge, Ivy Bridge:
    - 5ore, ring bus, integrated GPU, L3, IO
  - Haswell (Broadwell):
    - 18 core, ring bus, transactional memory, slices (EP)
- AMD:
  - K10 (Opteron: Barcelona, Magny Cours)
    - 12 core, Hypertransport
  - Bulldozer, Piledriver, Steamroller (Opteron, FX)
    - 16 core, Clustered Multithread: module with 2 integer cores
- Oracle (Sun) UltraSparc T1,T2,T3,T4,T5 (Niagara)
  - 16 cores, 8 threads/core (2 simultaneous), crossbar, 8 sockets
- ARM Cortex A9, A15 MPCore, big.LITTLE
  - 4 -8 cores, big.LITTLE: A7 + A15

UNSW
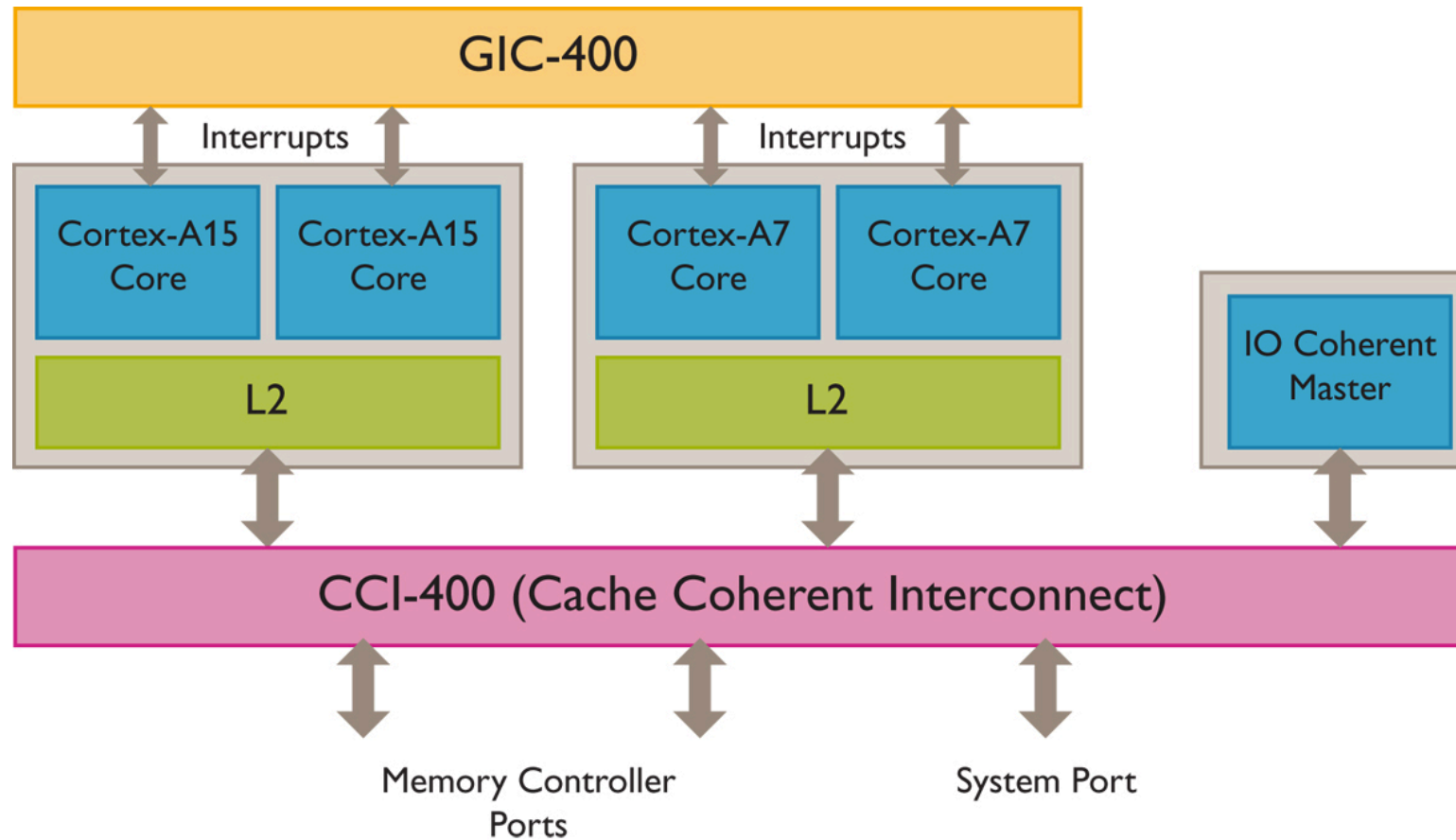THE UNIVERSITY OF NEW SOUTH WALES

Scale and Structure

- ARM Cortex A9 MPCore

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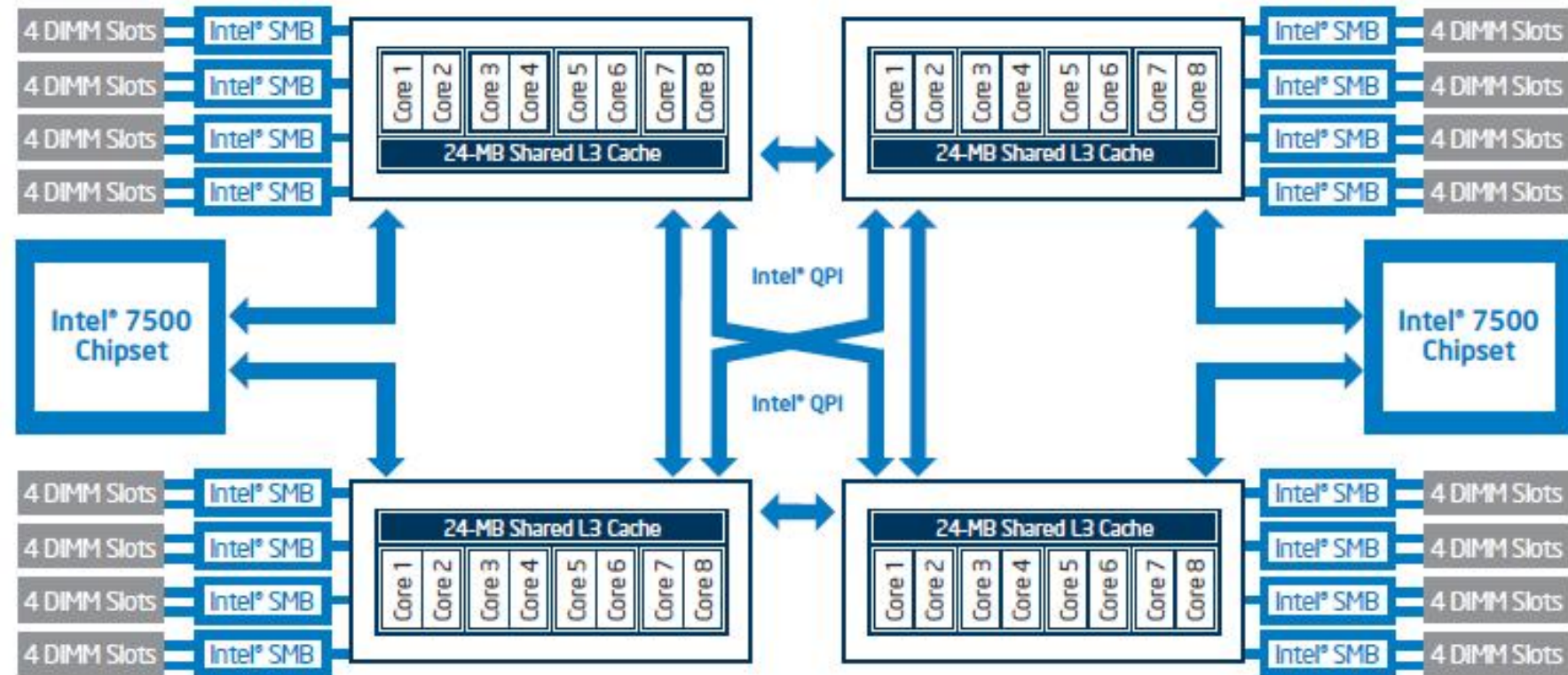
Scale and Structure

- ARM big.LITTLE

From http://www.arm.com/images/Fig_1_Cortex-A15_CCI_Cortex-A7_System.jpg
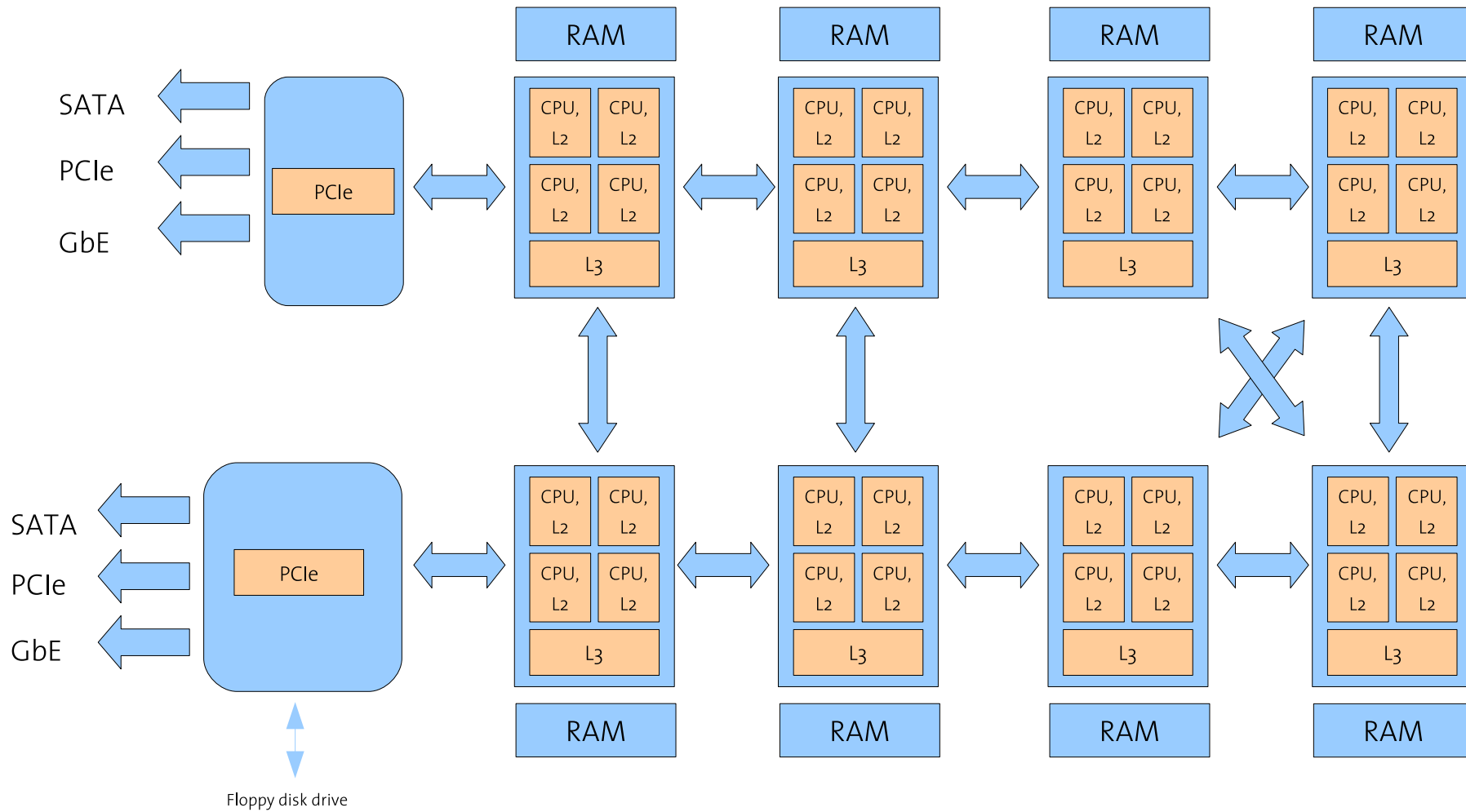
Scale and Structure

- Intel Nehalem

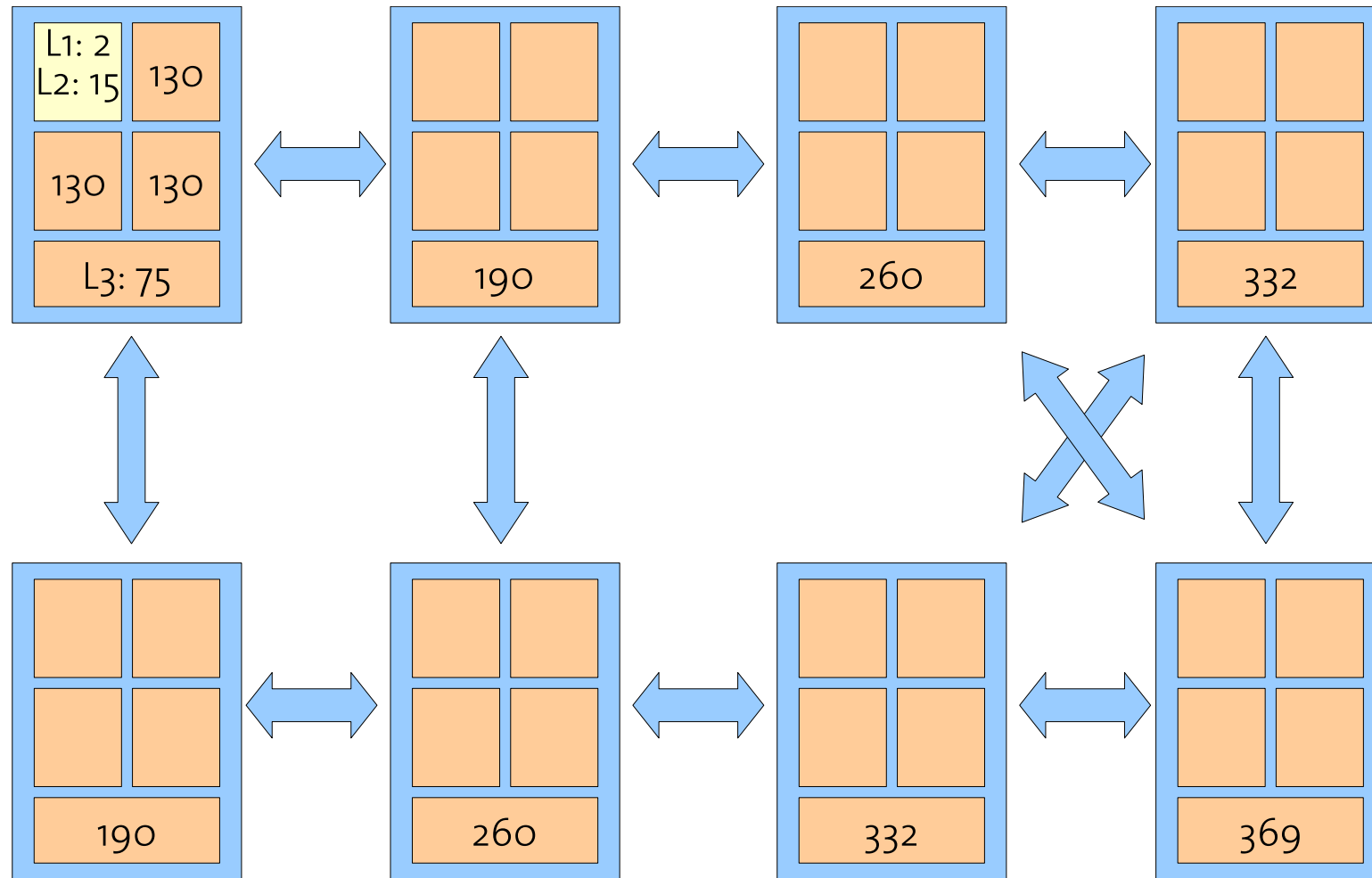From www.dawnofthered.net/wp-content/uploads/2011/02/Nehalem-EX-architecture-detailed.jpg
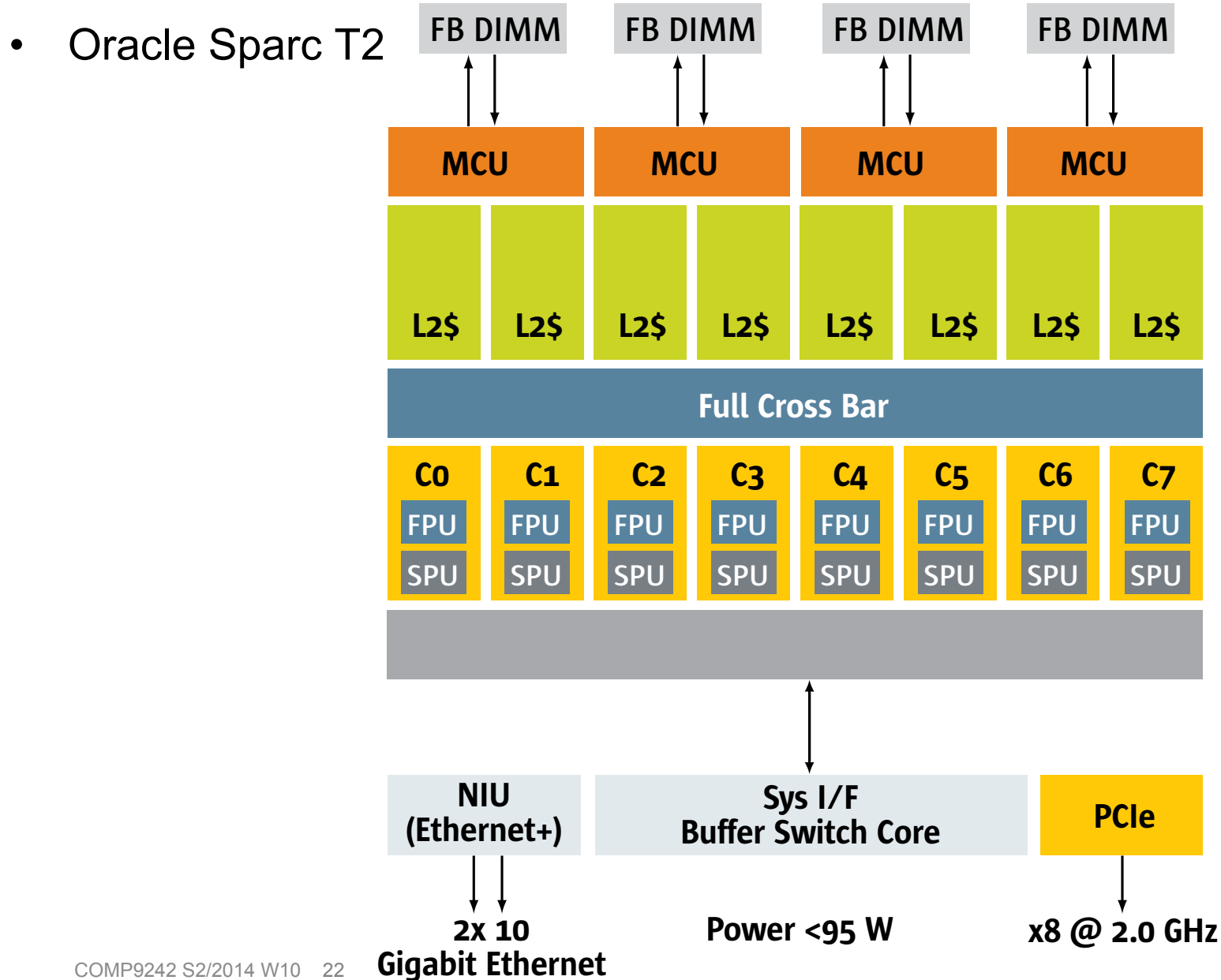
Interconnect

- ## AMD Barcelona

Memory Locality and Caches

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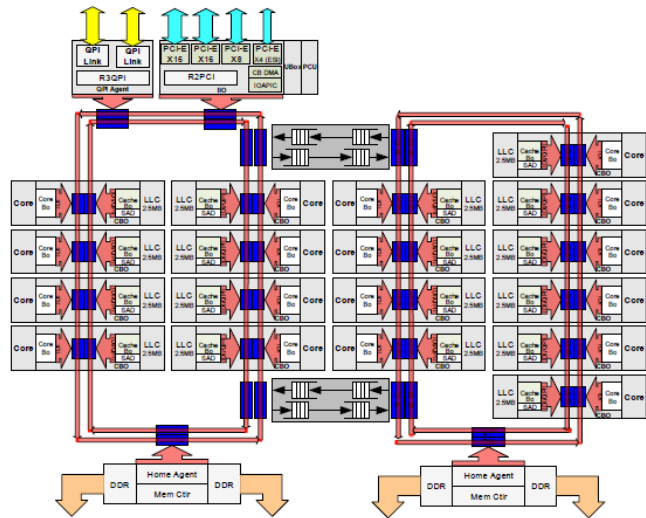
Interprocessor Communication

- Oracle Sparc T2

From Sun/Oracle
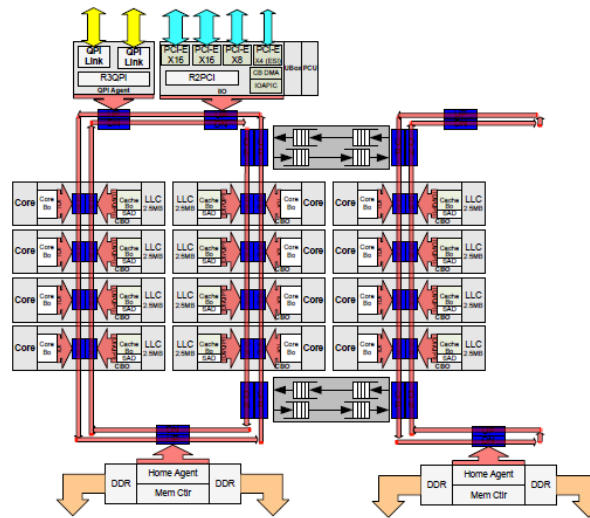
Interprocessor Communication
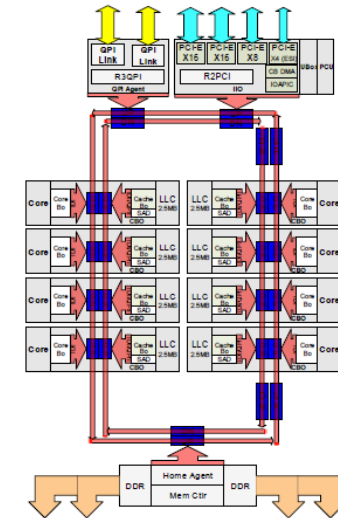
Haswell EP Die Configurations

14-18 Core (HCC)
10-12 Core (MCC)
4-8 Core (LCC)

Not representative of actual die-sizes, orientation and layouts – for informational use only.

Chop	Columns	Home Agents	Cores	Power (W)	Transitors (B)	Die Area (mm²)
HCC	4	2	14-18	110-145	5.69	662
MCC	3	2	6-12	65-160	3.84	492
LCC	2	1	4-8	55-140	2.60	354

From http://www.anandtech.com/show/8423/intel-xeon-e5-version-3-up-to-18-haswell-ep-cores-/4

Cluster on Die (COD) Mode

COD Mode for 18C E5-2600 v3

- Supported on 1S & 2S SKUs with 2 Home Agents (10+ cores)

- In memory directory bits & directory cache used on 2S to reduce coherence traffic and cache-to-cache transfer latencies

- Targeted at NUMA optimized workloads where latency is more important than sharing across Caching Agents

  - Reduces average LLC hit and local memory latencies

  - HA sees most requests from reduced set of threads potentially offering higher effective memory bandwidth

- OS/VMM own NUMA and process affinity decisions

From http://www.anandtech.com/show/8423/intel-xeon-e5-version-3-up-to-18-haswell-ep-cores-/4

UNSW
THE UNIVERSITY OF NEW SOUTH WALES

Experimental/Future Multiprocessor Hardware

- Microsoft Beehive

  - Ring bus, no cache coherence

- Tilera Tile64, Tile-Gx

  - 100 cores, mesh network

- Intel Polaris

  - 80 cores, mesh network

- Intel SCC

  - 48 cores, mesh network, no cache coherency

- Intel MIC (Multi Integrated Core) (Knight's Corner - Xeon Phi)

  - 60+ cores, ring bus

UNSW
THE UNIVERSITY OF NEW SOUTH WALES
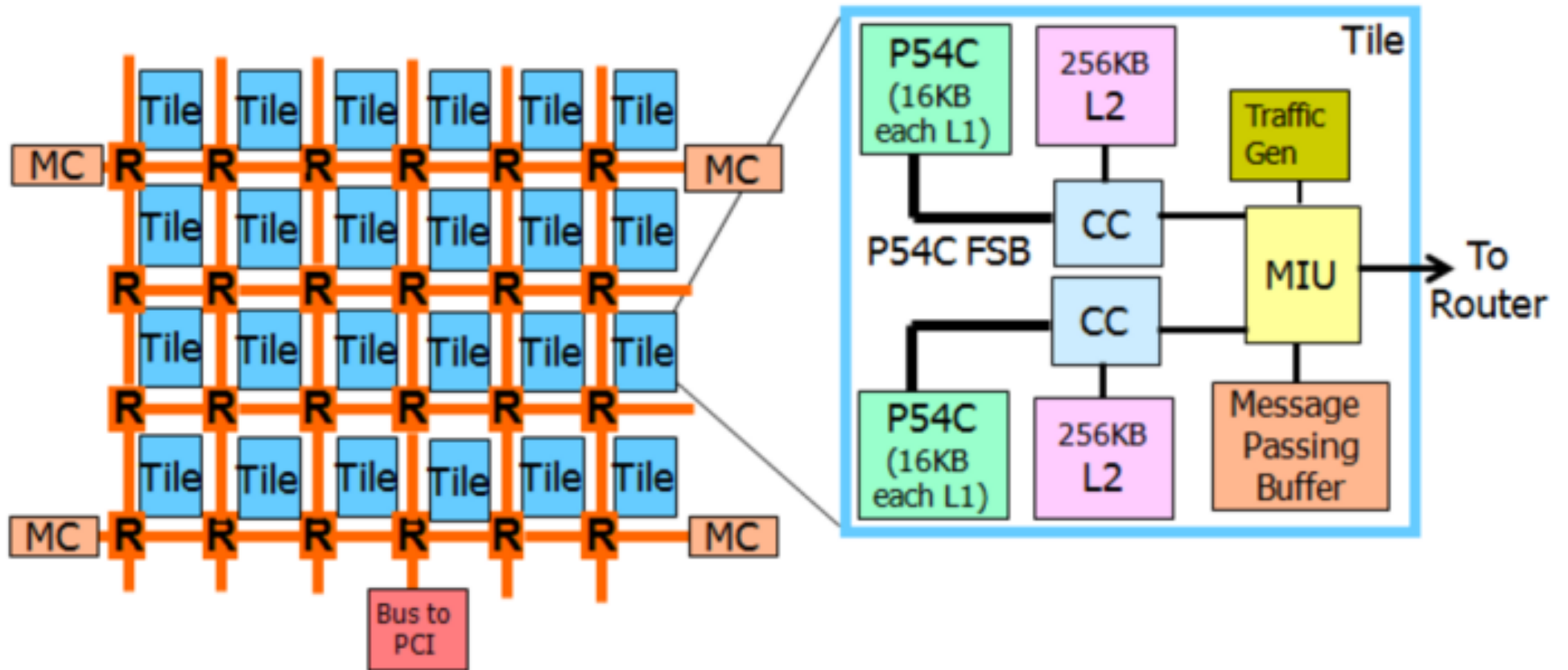
Scale and Structure

- **Tilera Tile64**, Intel Polaris
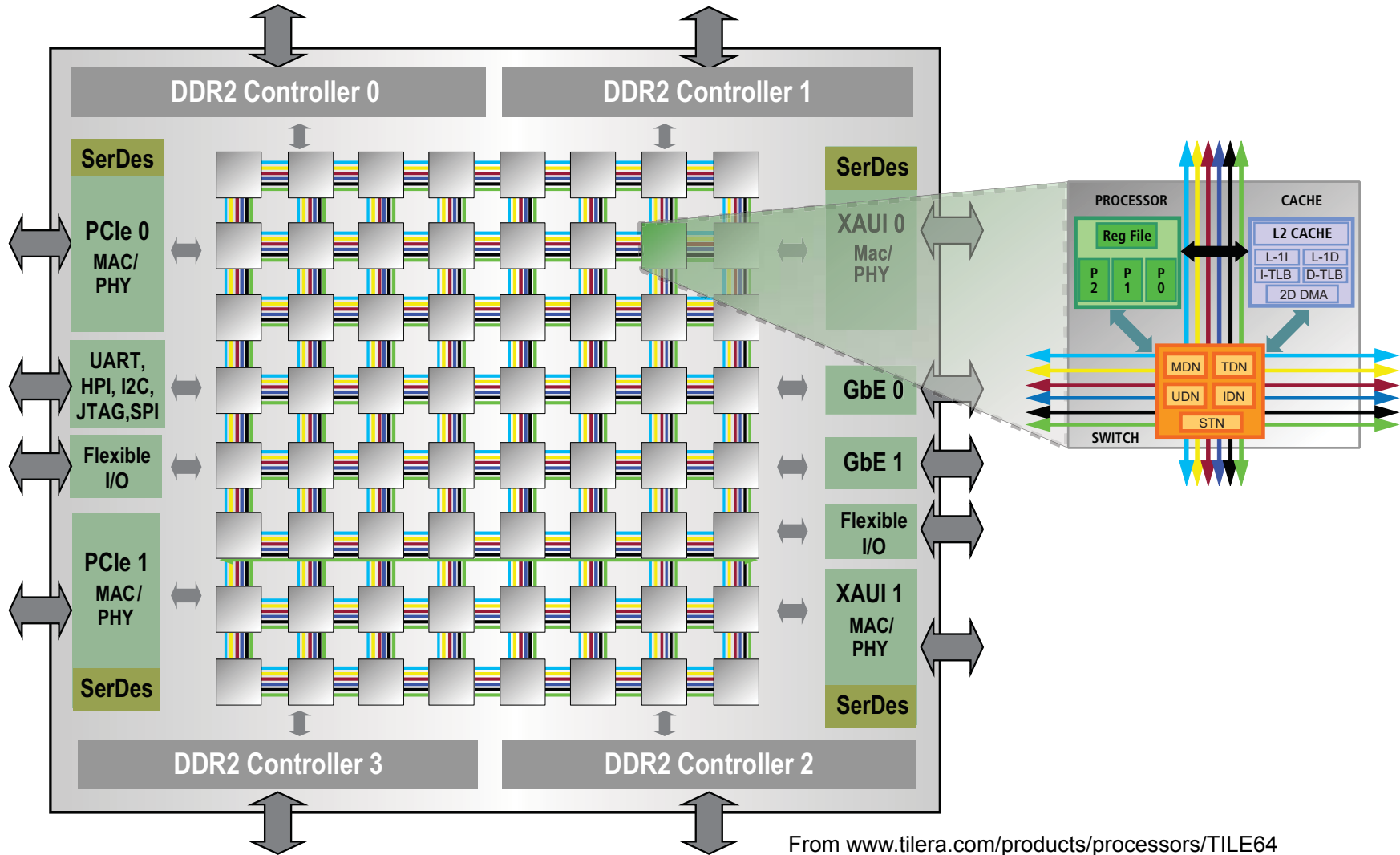
From www.tilera.com/products/processors/TILE64
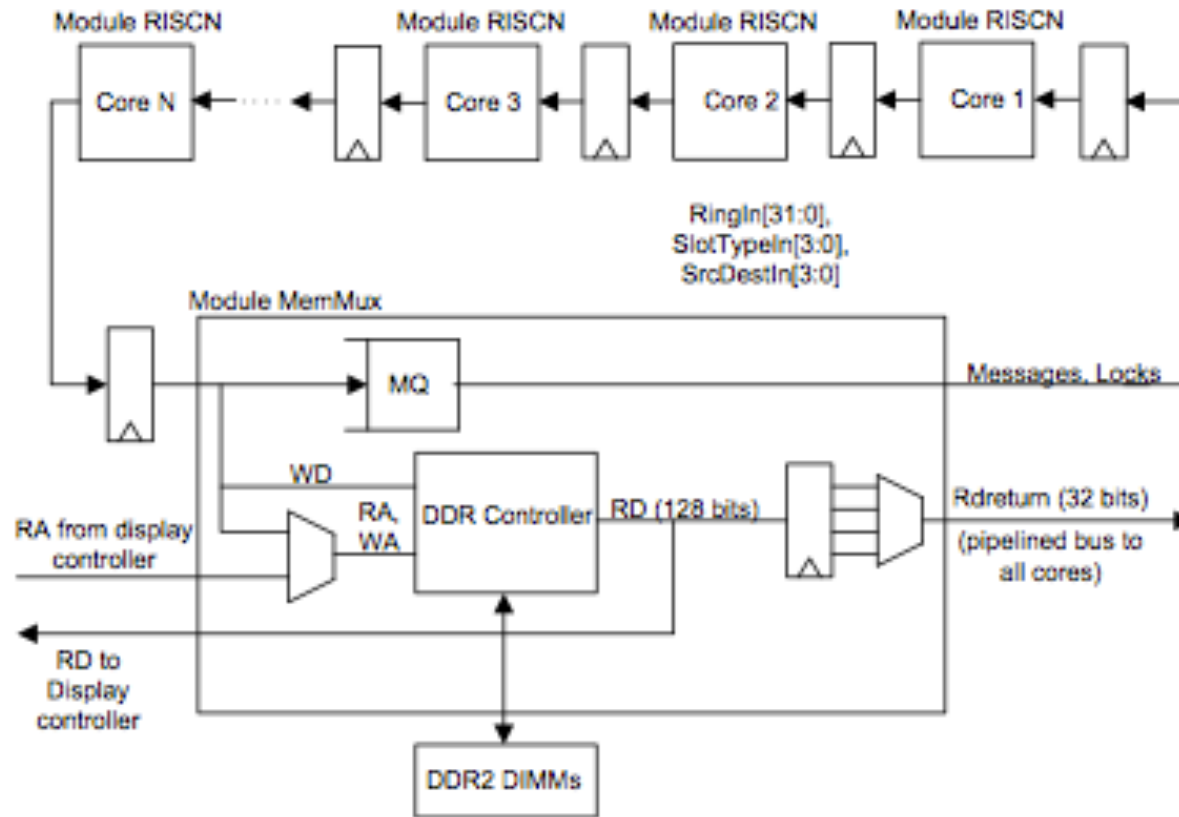
Cache and Memory

- Intel SCC

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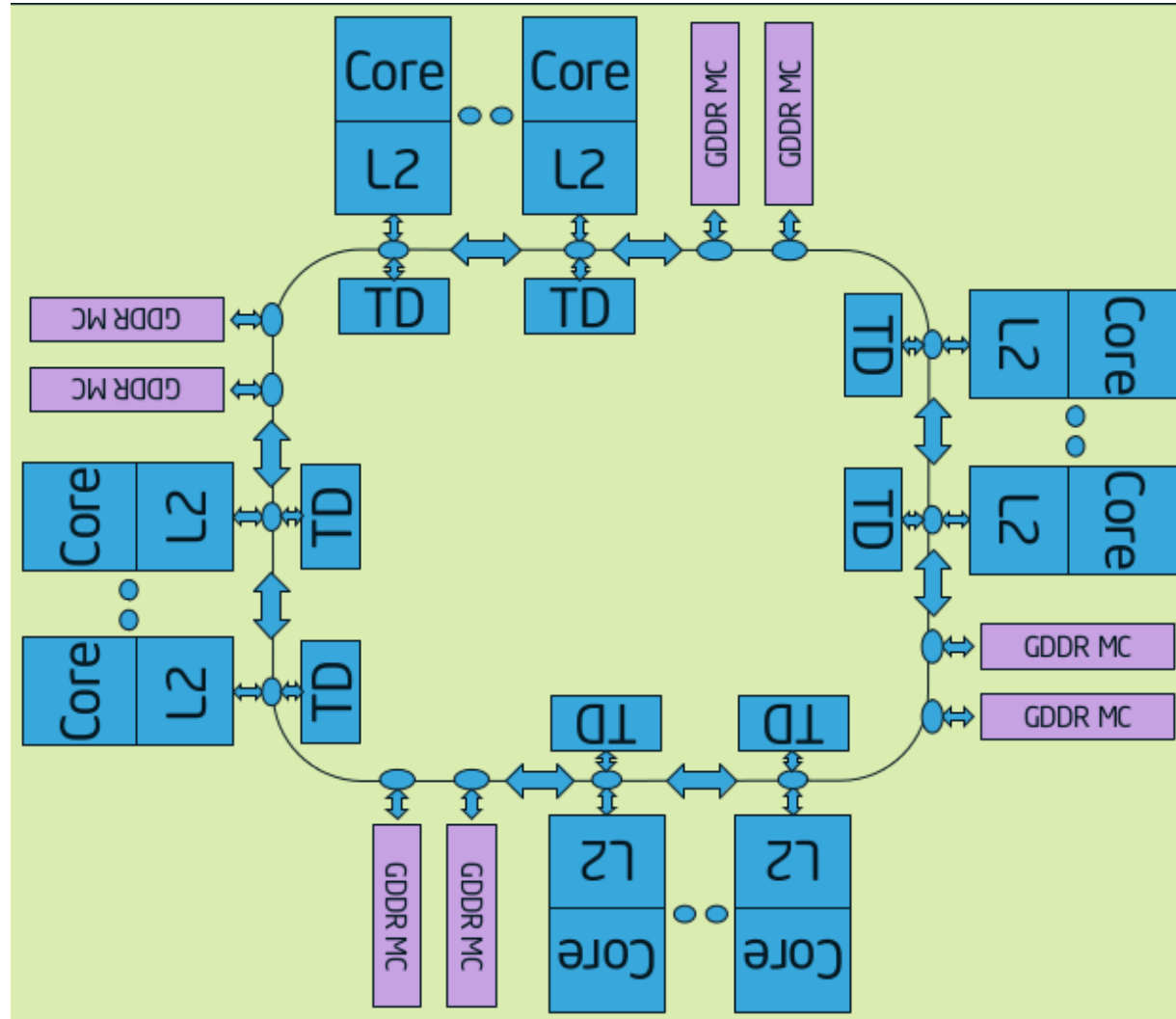
Interprocessor Communication

- Beehive

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

Interprocess Communication

- Intel MIC

From http://semiaccurate.com/2012/08/28/intel-details-knights-corner-architecture-at-long-last/

Summary

- Scalability
  - 100+ cores
  - Amdahl's law really kicks in

- Heterogeneity
  - Heterogeneous cores, memory, etc.
  - Properties of similar systems may vary wildly (e.g. interconnect topology and latencies between different AMD platforms)

- 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

OS DESIGN FOR MULTIPROCESSORS

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
- Avoid expensive memory access
  - Avoid uncached memory
  - Access cheap (close) memory

Optimisation for Scalability

- 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

OS Design Guidelines 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
  - Divide multiprocessor into smaller bits, use them as normal
  - Using virtualisation
  - Using exokernel

- Reduced sharing
  - Brute force & Heroic Effort
    - Find problems in existing OS and fix them
    - E.g Linux rearchitecting: BKL -> fine grained locking
  - By design
    - Avoid shared data as much as possible

- No sharing
 - Computer is a distributed system
    - Do extra work to share!

NICTA

UNSW
THE UNIVERSITY OF NEW SOUTH WALES

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 (Cerberus)

NICTA

UNSW
THE UNIVERSITY OF NEW SOUTH WALES
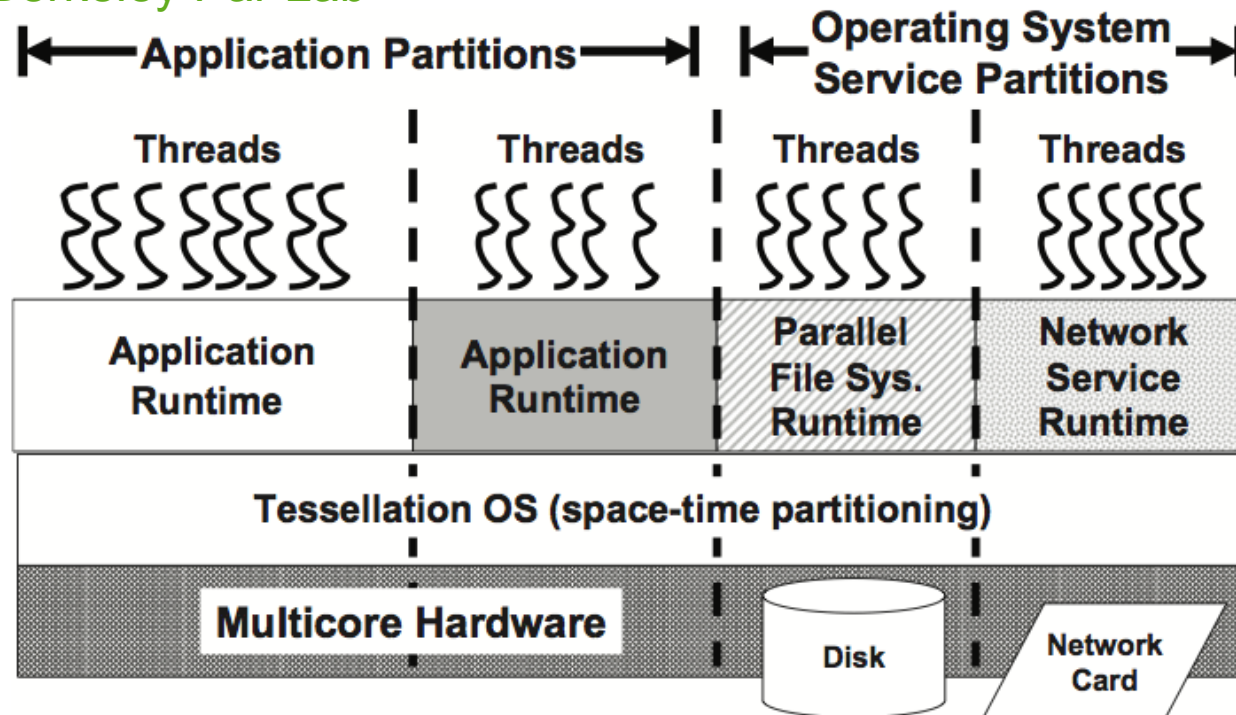
Disco Architecture
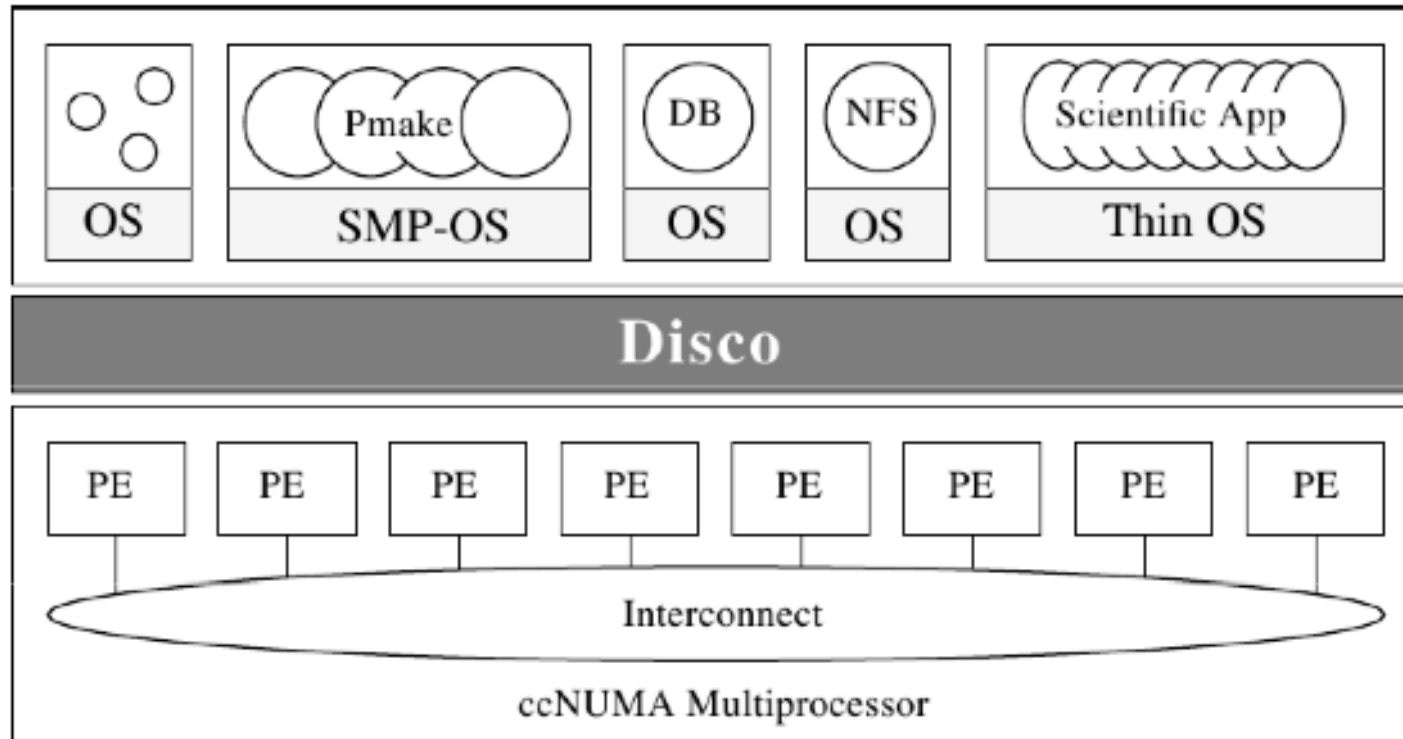
[Bugnion et al., 1997]

Disco Performance
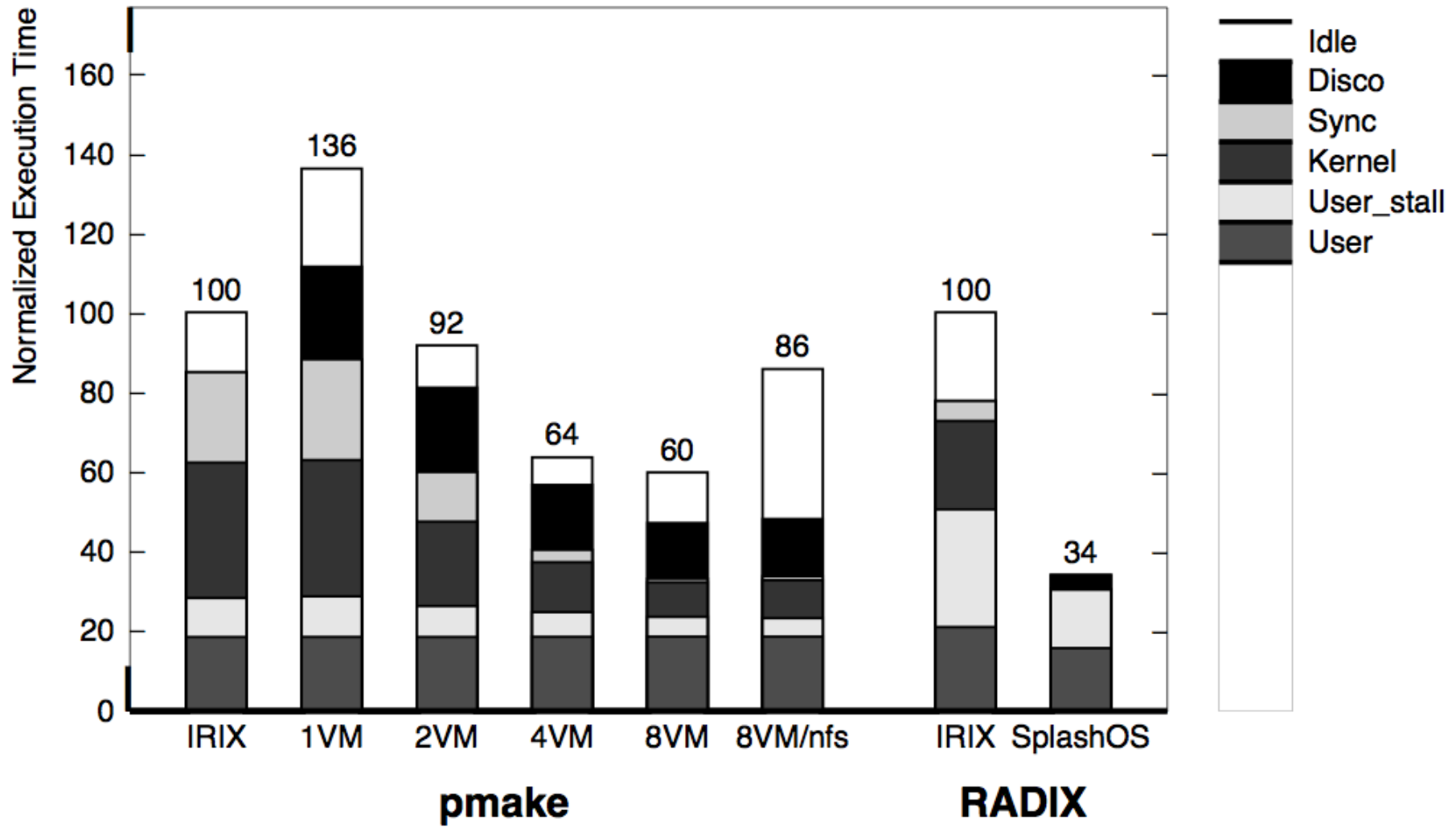
Space-Time Partitioning

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
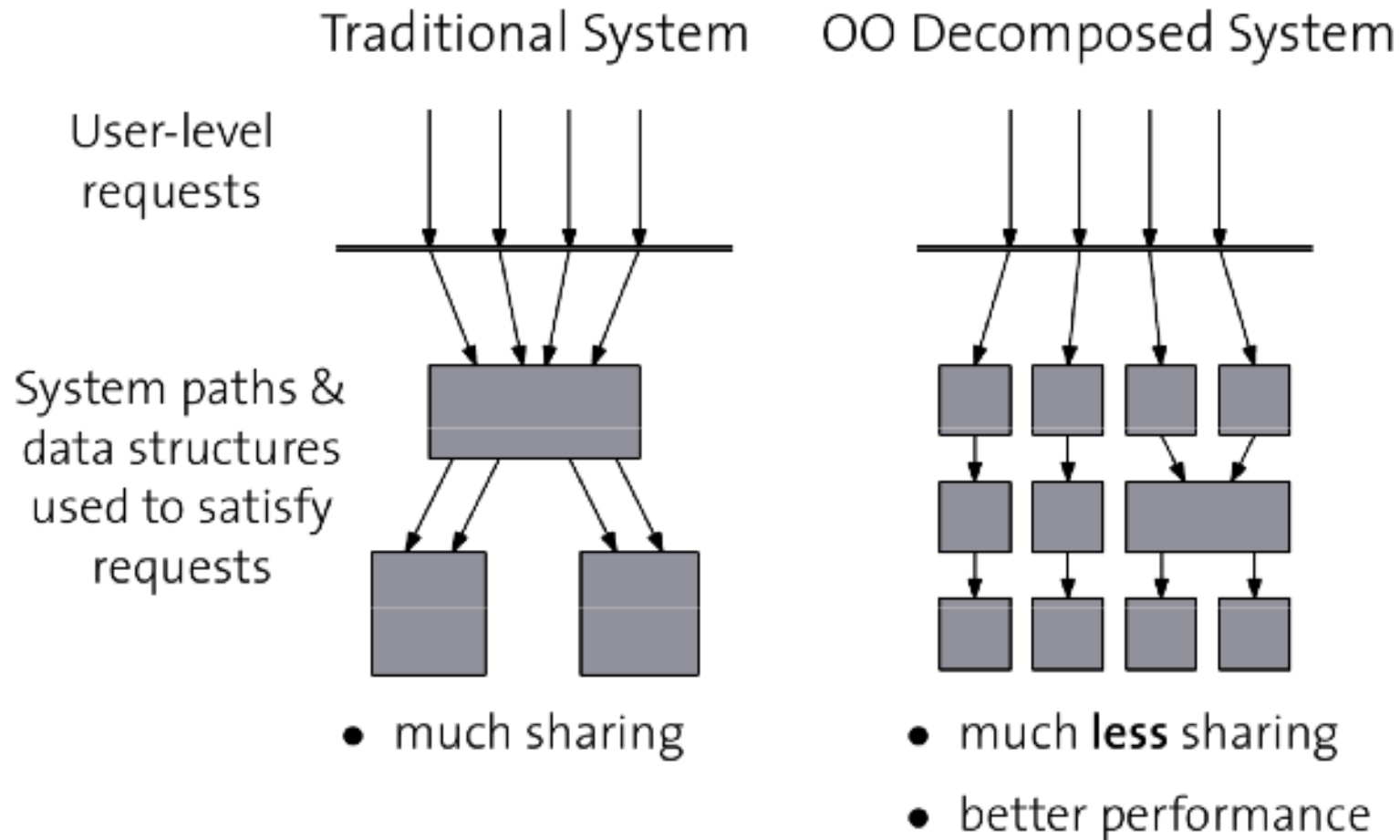
– Berkeley Par Lab

Tessellation
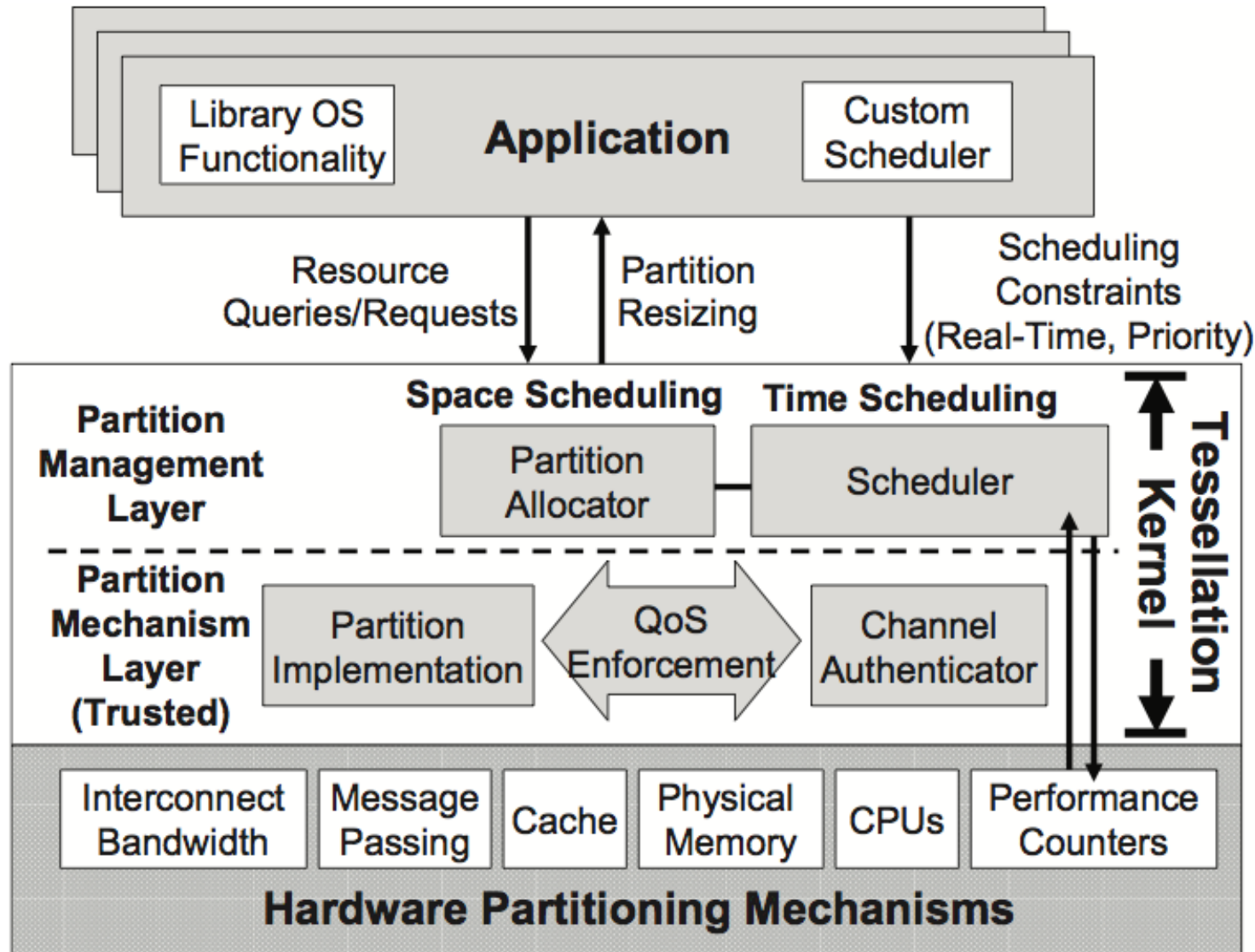
Reduce Sharing

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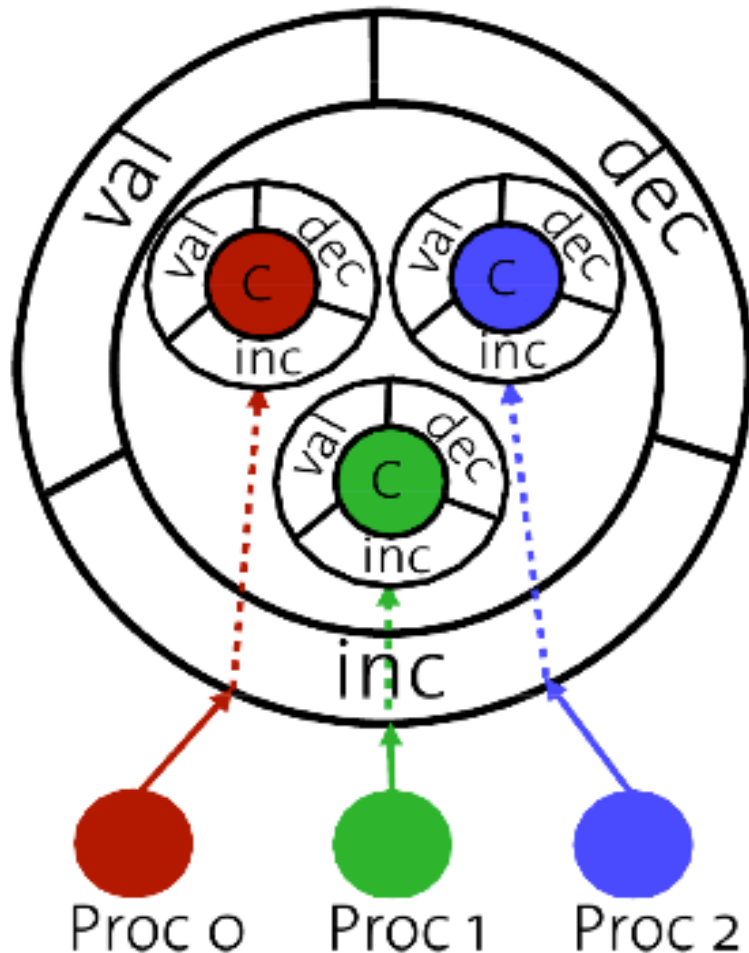http://www.research.ibm.com/K42/

- 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

Traditional System

OO Decomposed System

User-level requests

System paths & data structures used to satisfy requests

- 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
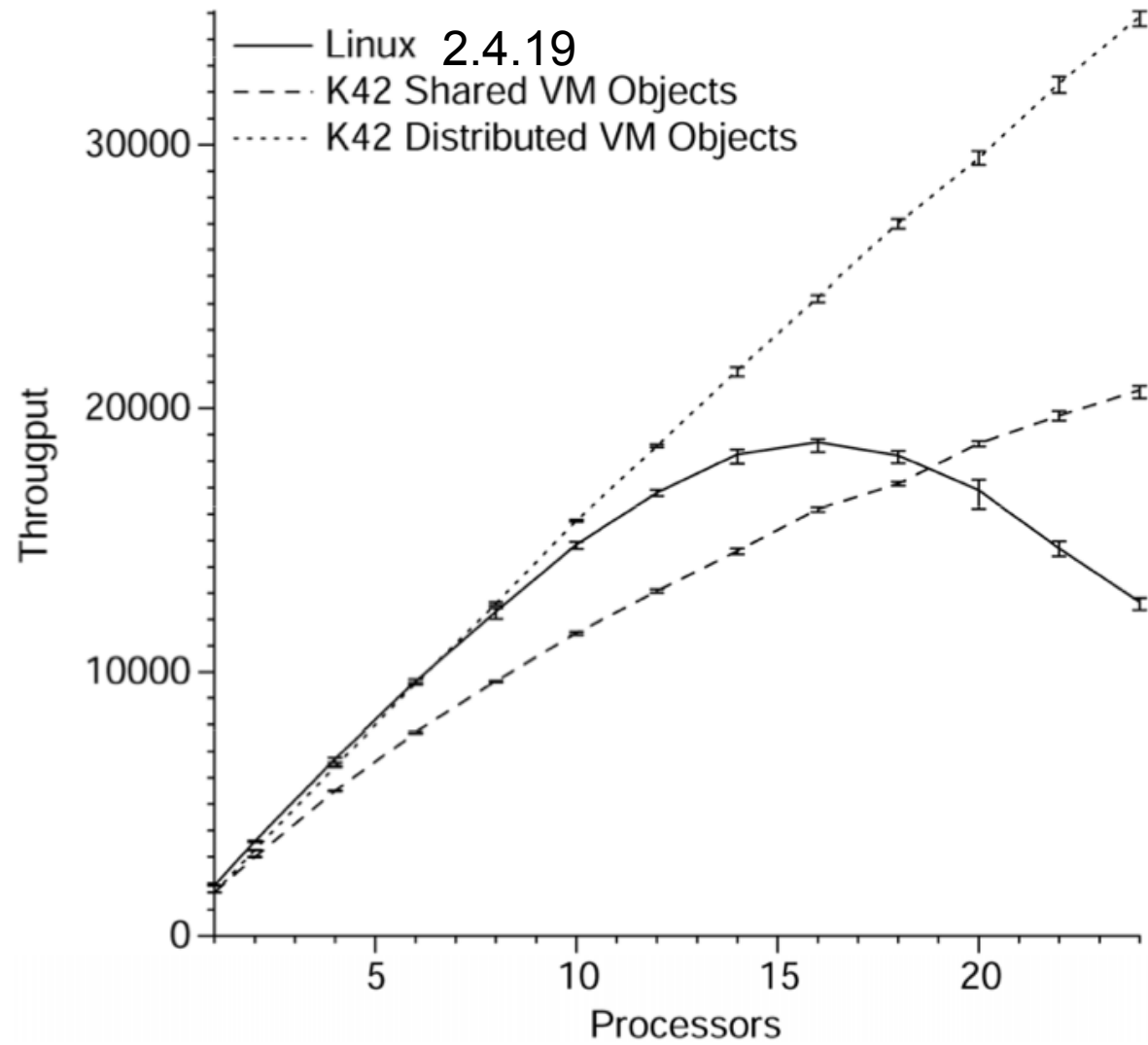
Corey

- Context
  - 2008, high-end multicore servers, MIT
- Goals:
  - Application control of OS sharing
- OS
  - Exokernel-like, higher-level services as libraries
  - By default only single core access to OS data structures
  - Calls to control how data structures are shared
- 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.

Corey: An Operating System for Many Cores [Boyd-Wickizer et al., 2008]
http://pdos.csail.mit.edu/corey

UNSW
THE UNIVERSITY OF NEW SOUTH WALES

Linux Brute Force Scalability

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

  - 2010, high-end multicore servers, MIT

- Goals:

  - Scaling commodity OS

- Linux scalability (2010 – scale Linux to 48 cores)
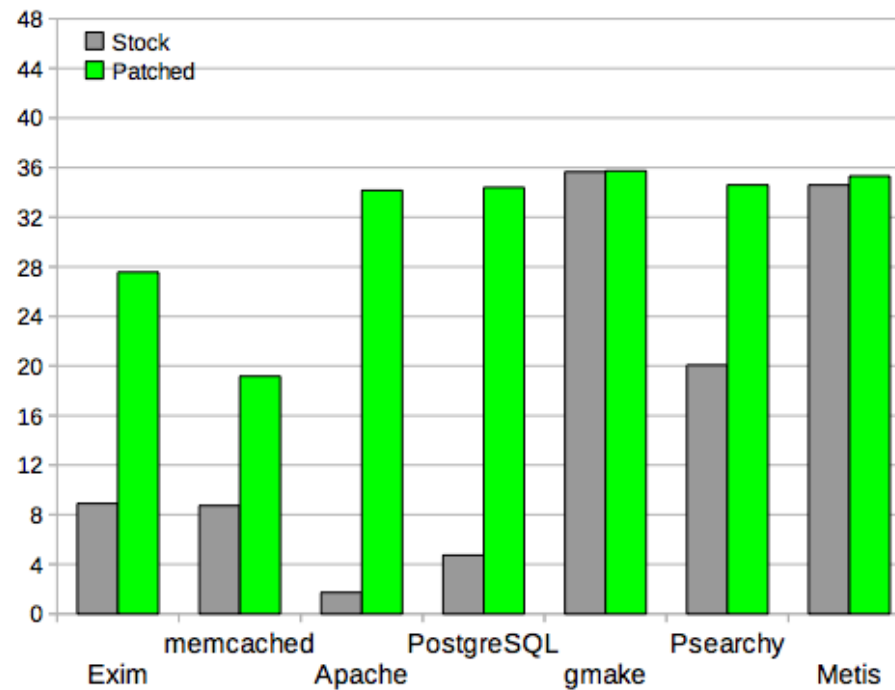
Y-axis: (throughput with 48 cores) / (throughput with one core)

Linux Brute Force Scalability

- Apply lessons from parallel computing and past research
  - sloppy counters,
  - per-core data structs,
  - fine-grained lock, lock free,
  - cache lines
  - 3002 lines of code changed

	memcached	Apache	Exim	PostgreSQL	gmake	Psearchy	Metis
Mount tables		X	X				
Open file table		X	X				
Sloppy counters	X	X	X				
inode allocation	X	X					
Lock-free dentry lookup		X	X				
Super pages							X
DMA buffer allocation	X	X					
Network stack false sharing	X	X		X			
Parallel accept		X					
Application modifications				X		X	X

- Conclusion:
  - no scalability reason to give up on traditional operating system organizations just yet.

Scalability of the API

- Context
  - 2013, previous multicore projects at MIT

- Goals
  - How to know if a system is really scalable?

- Workload-based evaluation
  - Run workload, plot scalability, fix problems
  - Did we miss any non-scalable workload?
  - Did we find all bottlenecks?

- Is there something fundamental that makes an system non-scalable?
 - The interface might be a fundamental bottleneck

The Scalable Commutativity Rule: Designing Scalable Software for Multicore Processors[Clements et al., 2013]

Scalable Commutativity Rule

- The Rule
  - *Whenever interface operations commute, they can be implemented in a way that scales.*

- Commutative operations:
  - Cannot distinguish order of operations from results
  - Example:
    - Creat:
      - Requires that lowest available FD be returned
      - Not commutative: can tell which one was run first

- Why are commutative operations scalable?
  - results independent of order ⇒ communication is unnecessary
  - without communication, no conflicts

- Informs software design process
  - Design: design guideline for scalable interfaces
  - Implementation: clear target
 - Test: workload-independent testing

# Commuter: An Automated Scalability Testing Tool
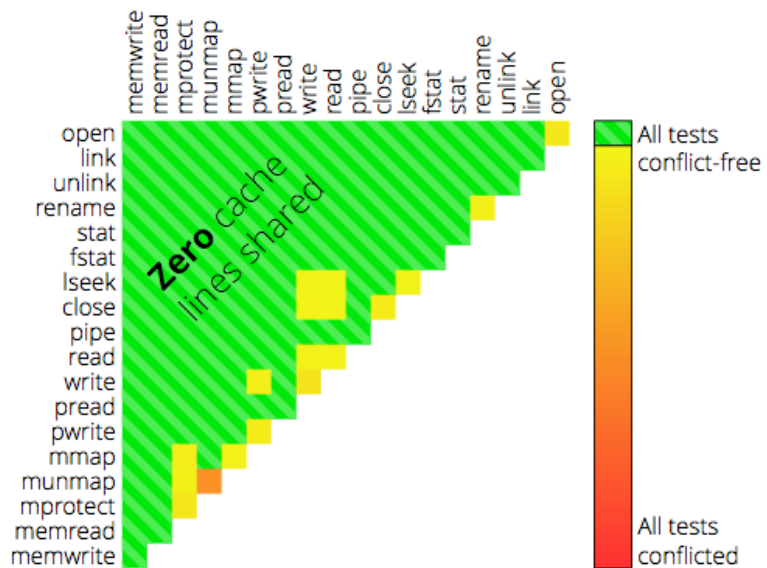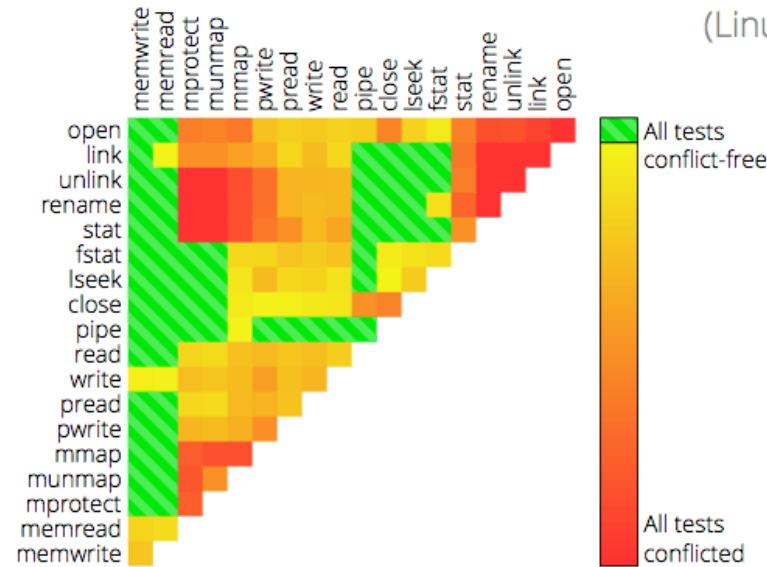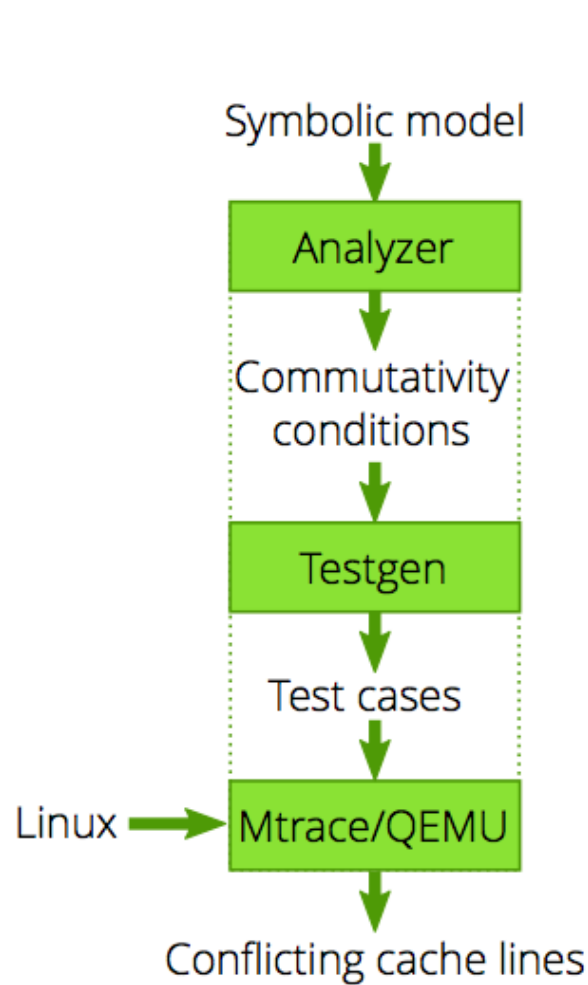
FlexSC

- 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!

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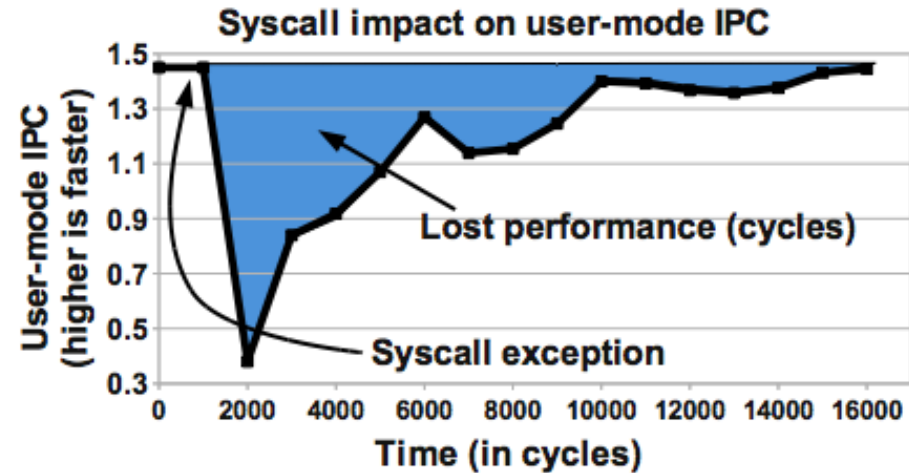
| Syscall	Instructions	Cycles	IPC	i-cache	d-cache	L2	L3	d-TLB
stat	4972	13585	0.37	32	186	660	2559	21
pread	3739	12300	0.30	32	294	679	2160	20
pwrite	5689	31285	0.18	50	373	985	3160	44
open+close	6631	19162	0.34	47	240	900	3534	28
mmap+munmap	8977	19079	0.47	41	233	869	3913	7
open+write+close	9921	32815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

FlexSC

- Asynchronous system calls
  - Batch system calls
  - Run them on dedicated cores
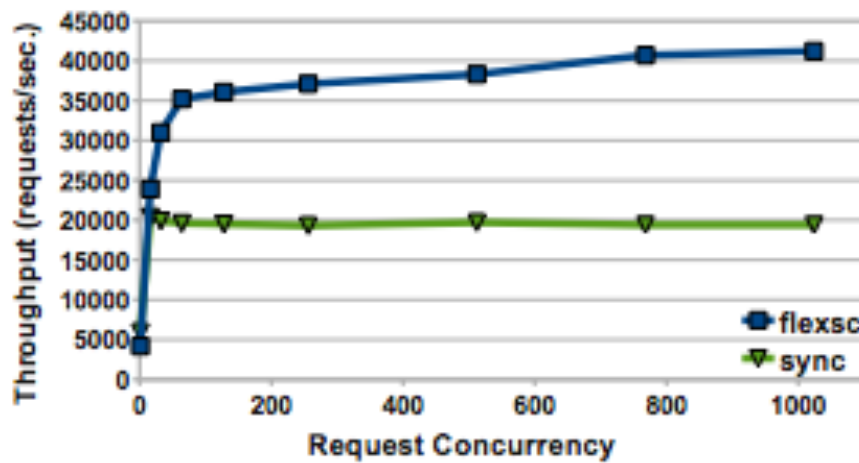- FlexSC-Threads
  - M on N
  - M >> N
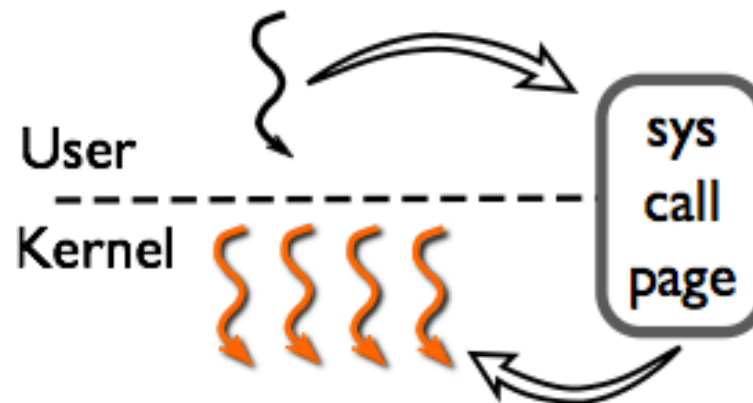
FlexSC Results
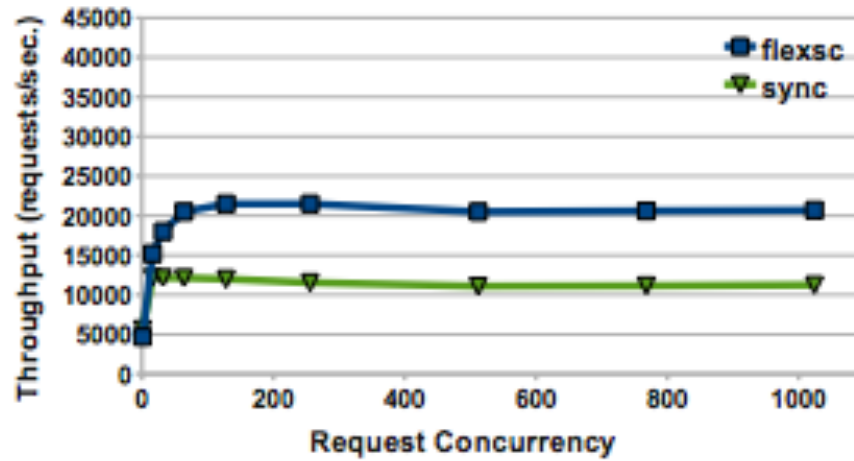
(a) 1 Core
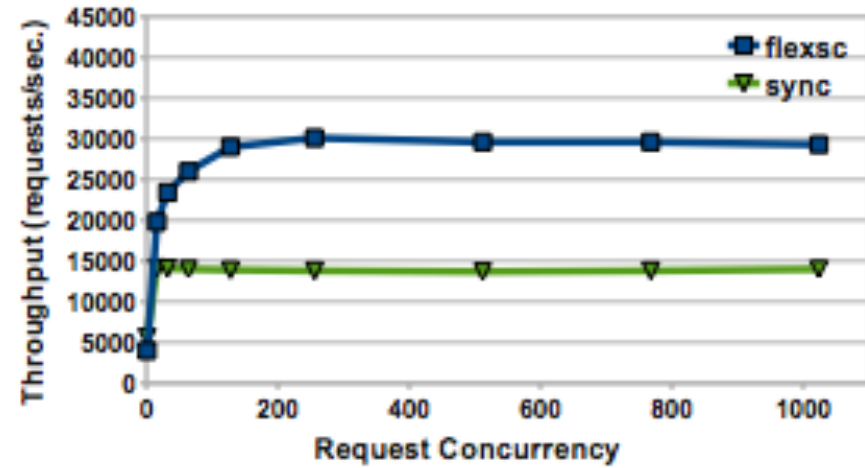
(b) 2 Cores
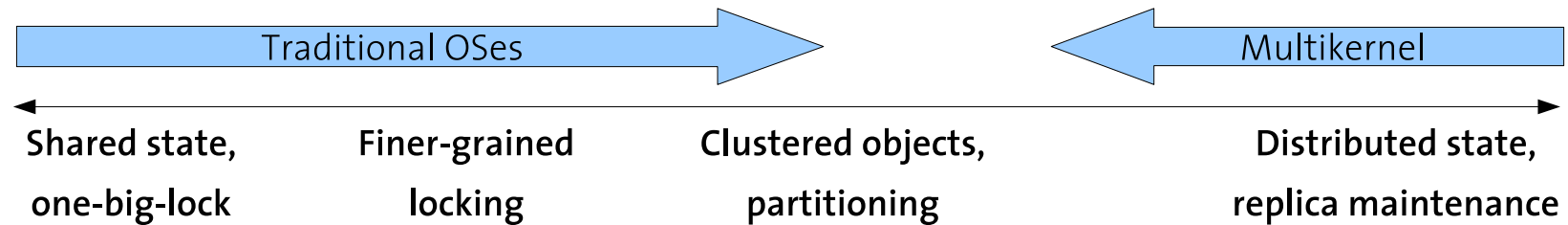
Apache
FlexSC: batching,
sys call core redirect

(c) 4 Cores

UNSW
THE UNIVERSITY OF NEW SOUTH WALES

No sharing

- 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

UNSW
THE UNIVERSITY OF NEW SOUTH WALES
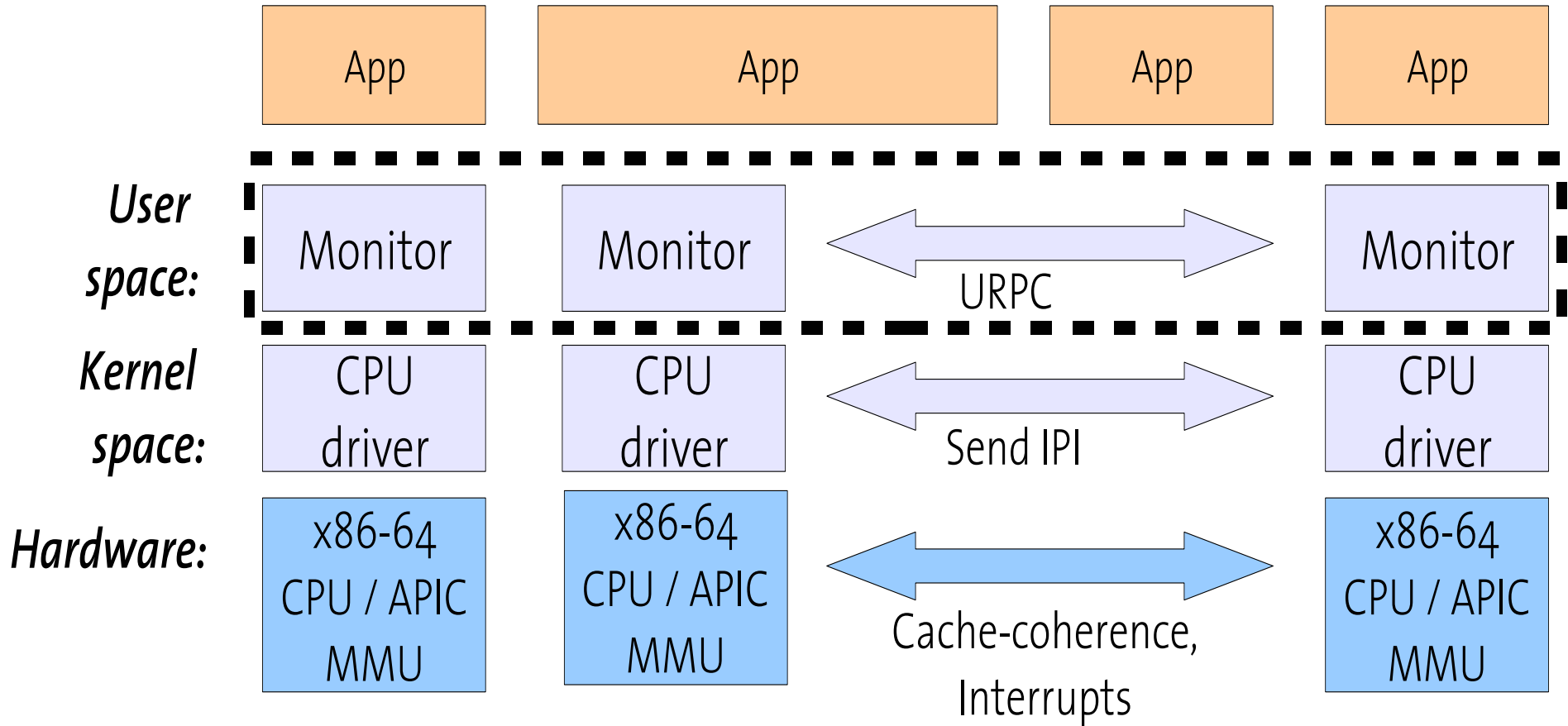
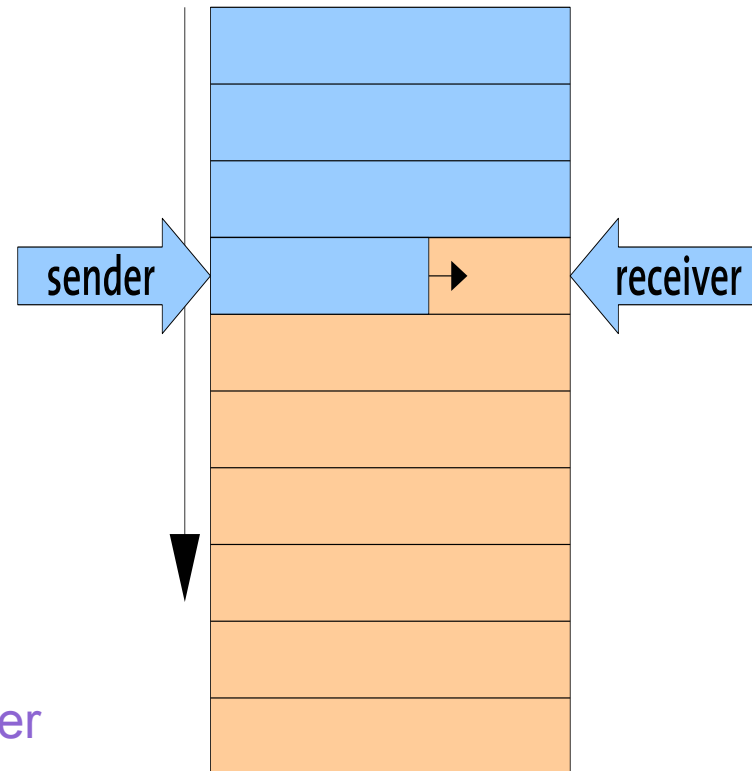# Barrelfish

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

UNSW
THE UNIVERSITY OF NEW SOUTH WALES

Barrelfish: Communication

- 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
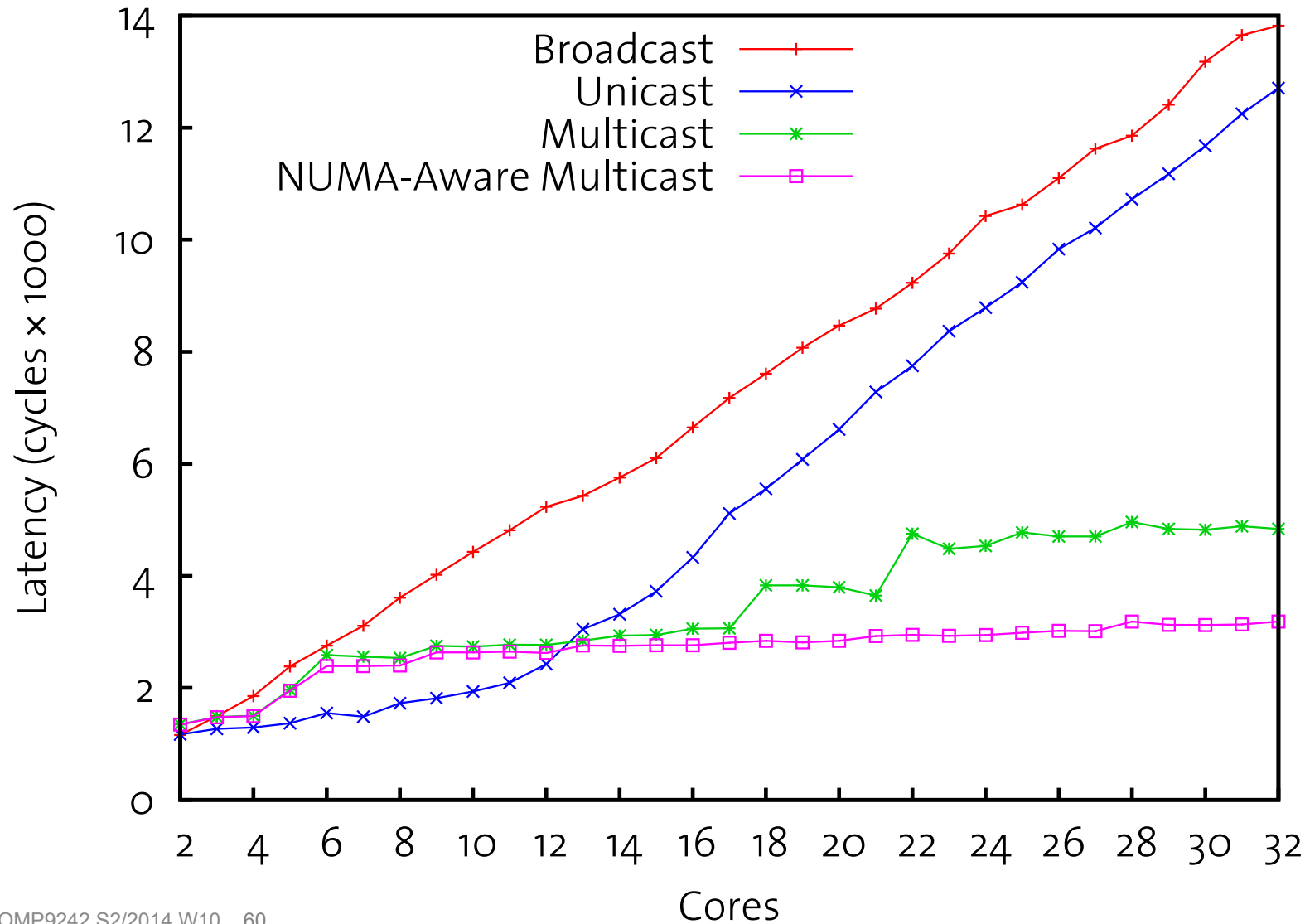   - Switch to block and IPI if wait is too long.

sender → | ← receiver

Barrelfish: Results

- Message passing vs caching
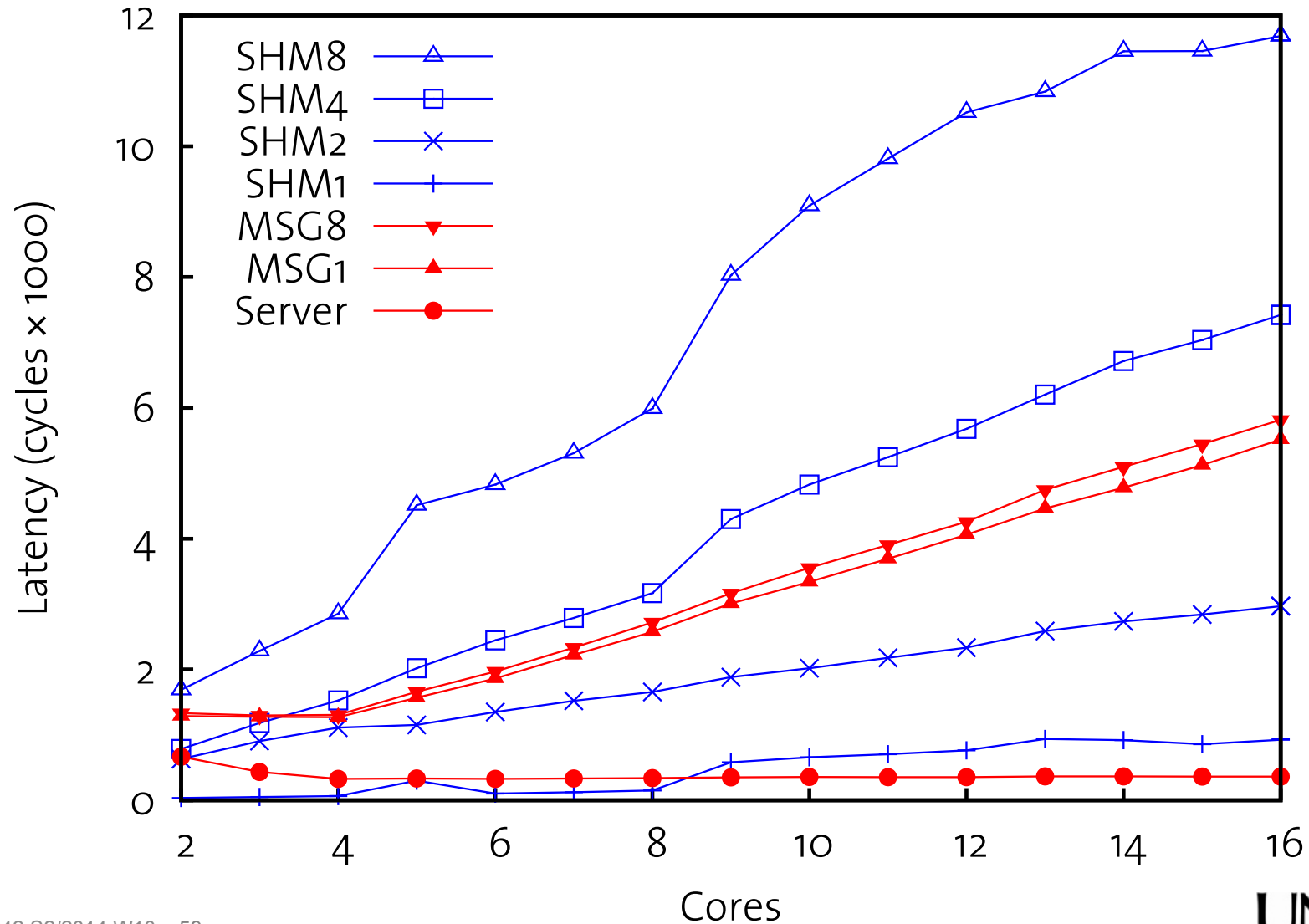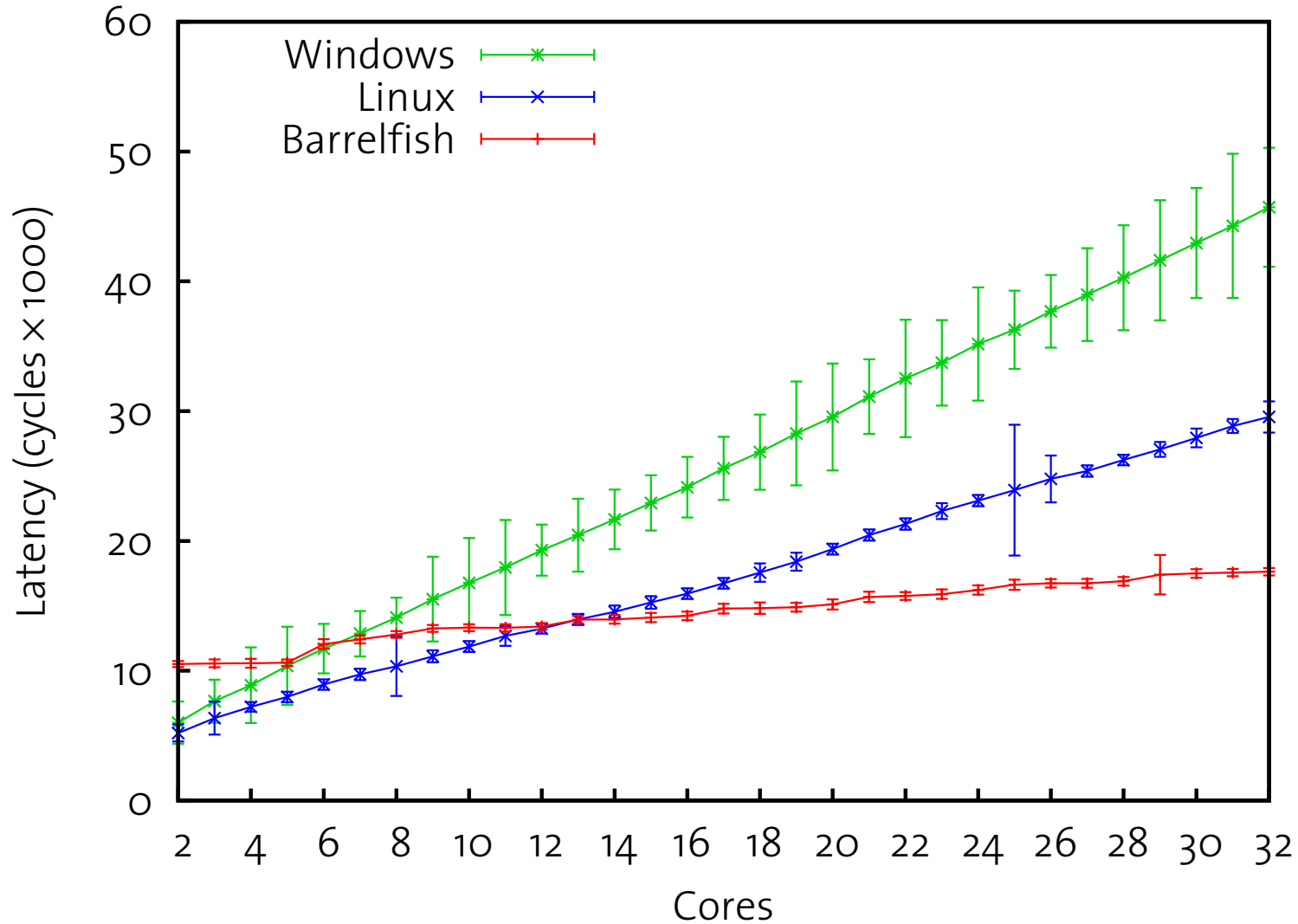
Barrelfish: Results

- Broadcast vs Multicast

Barrelfish: Results

- TLB shootdown

SUMMARY

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)