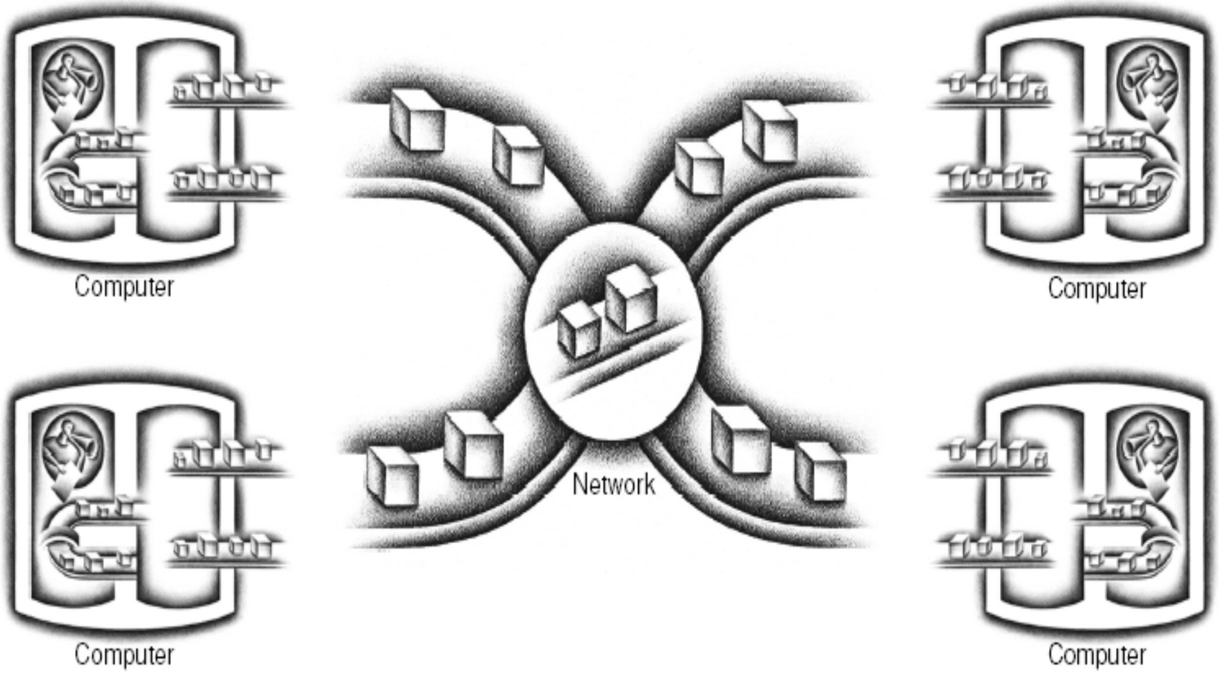
Multicores, Multiprocessor (Part I)

The Five Classic Components of a Computer



Parallel Computers

- Goal: connecting multiple computers to get higher performance
  - Multiprocessors
  - Scalability, availability, power efficiency
- Parallel Workloads
  - Job-level (process-level) parallelism
    - High throughput for independent jobs
  - Parallel processing program
    - Single program run on multiple processors

Challenge of Parallel Processing

- **Challenges**
  - **Partitioning**
  - **Coordination**
  - **Communication overheads**
- **Available Parallelism in applications**
  - **Amdahl's Law (FracX: original % to be speed up)**
    Speedup = 1 / [(FracX/SpeedupX + (1-FracX)]

What fraction of the original computation can be sequential to get 80X speedup from 100 processors?

Assume either 1 processor or 100 fully used

80 = 1 / [(FracX/100 + (1-FracX)]

0.8*FracX + 80*(1-FracX) = 80 - 79.2*FracX = 1

FracX = (80-1)/79.2 = 0.9975

•Only 0.25% sequential!

Scaling Example

- Workload: sum of 10 scalars, and 10 × 10 matrix sum
  - Speed up from 10 to 100 processors
- Single processor: Time = $(10 + 100) \times t_{add}$
- 10 processors
  - Time = $10 \times t_{add} + 100/10 \times t_{add} = 20 \times t_{add}$
  - Speedup = 110/20 = 5.5 (55% of potential)
- 100 processors
  - Time = $10 \times t_{add} + 100/100 \times t_{add} = 11 \times t_{add}$
  - Speedup = 110/11 = 10 (10% of potential)
- Assumes load can be balanced across processors

Scaling Example (cont)

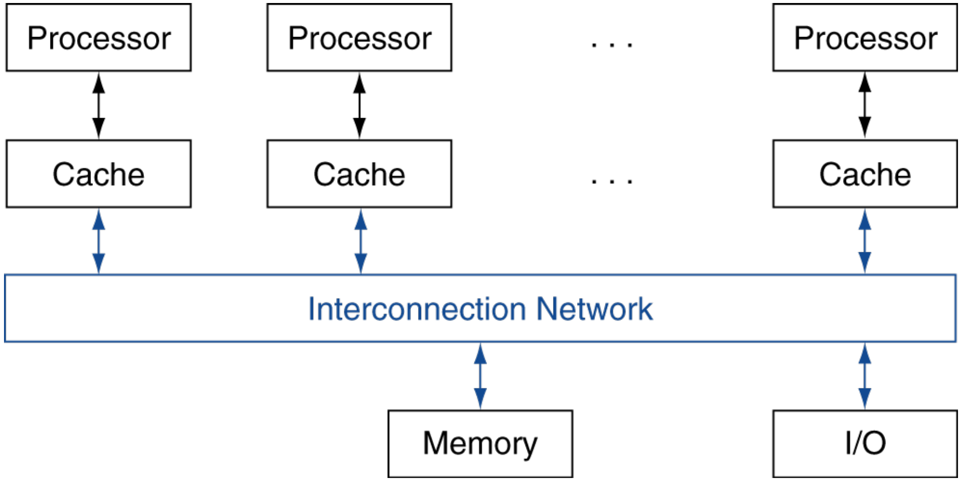
- What if matrix size is 100 × 100?
- Single processor: Time = $(10 + 10000) \times t_{add}$
- 10 processors
  - Time = $10 \times t_{add} + 10000/10 \times t_{add} = 1010 \times t_{add}$
  - Speedup = 10010/1010 = 9.9 (99% of potential)
- 100 processors
  - Time = $10 \times t_{add} + 10000/100 \times t_{add} = 110 \times t_{add}$
  - Speedup = 10010/110 = 91 (91% of potential)
- Assuming load balanced

Strong vs Weak Scaling

- Strong scaling: problem size fixed
- Weak scaling: problem size proportional to number of processors

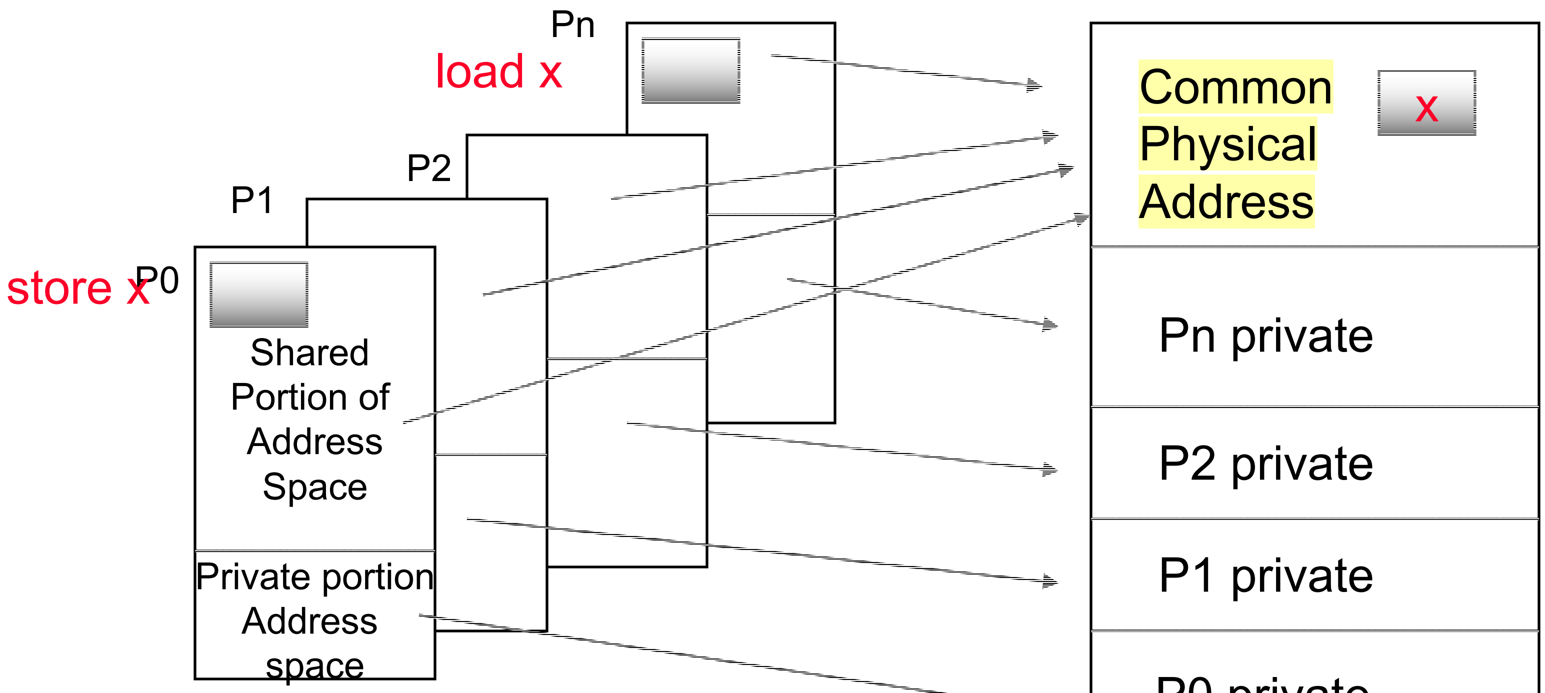
Shared Memory

- SMP: shared memory multiprocessor
  - Hardware provides single physical address space for all processors
  - Synchronize shared variables using locks
  - Memory access time
    - » UMA (uniform) vs. NUMA (nonuniform)



Communication Models

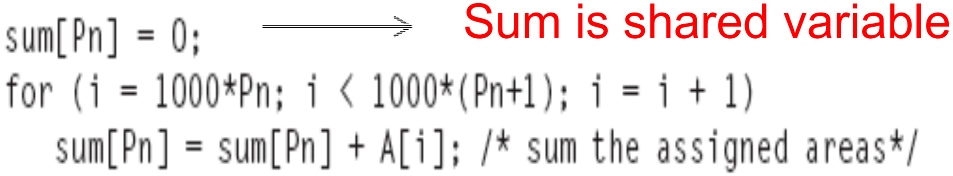
- **Single Address Space: load/store**



Program Example – Single-Address Space

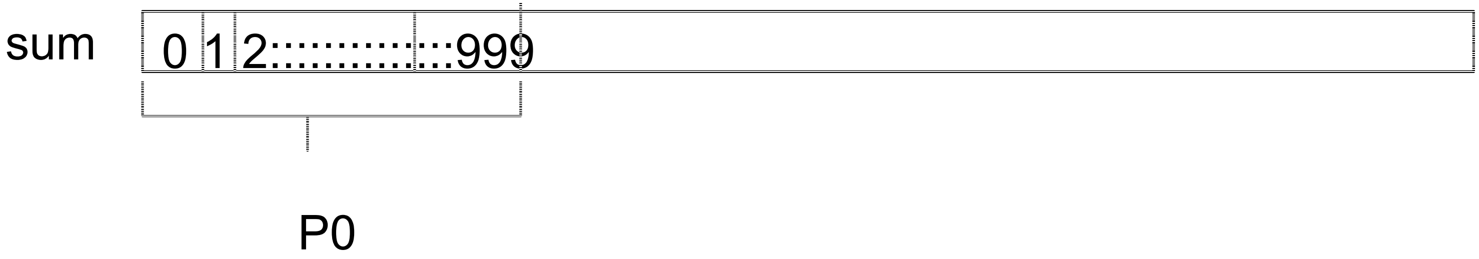
- **sum 100,000 numbers & 100 processors (load & store)**

First Step: each processor (Pn) sums his subset of numbers



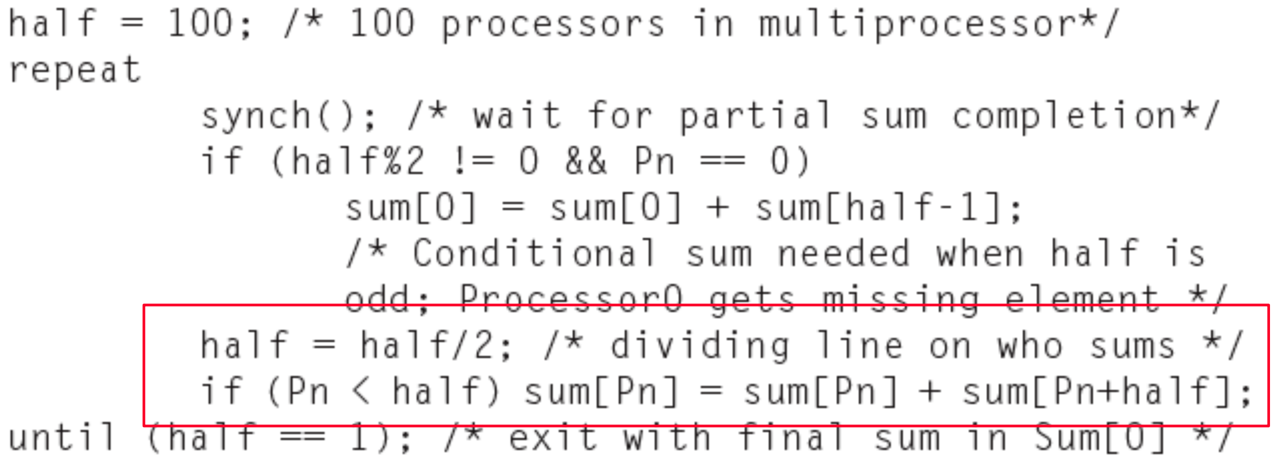
```
sum[Pn] = 0;
for (i = 1000*Pn; i < 1000*(Pn+1); i = i + 1)
   sum[Pn] = sum[Pn] + A[i]; /* sum the assigned areas*/
```

→ Sum is shared variable

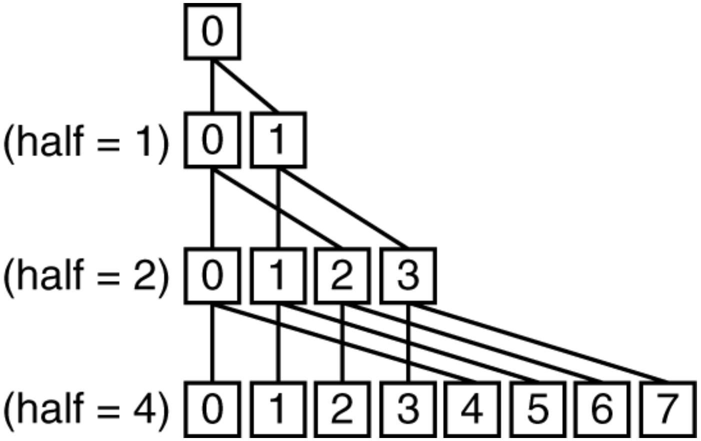


Program Example – Single-Address Space

Second Step: Add partial sums via divide-and-conquer

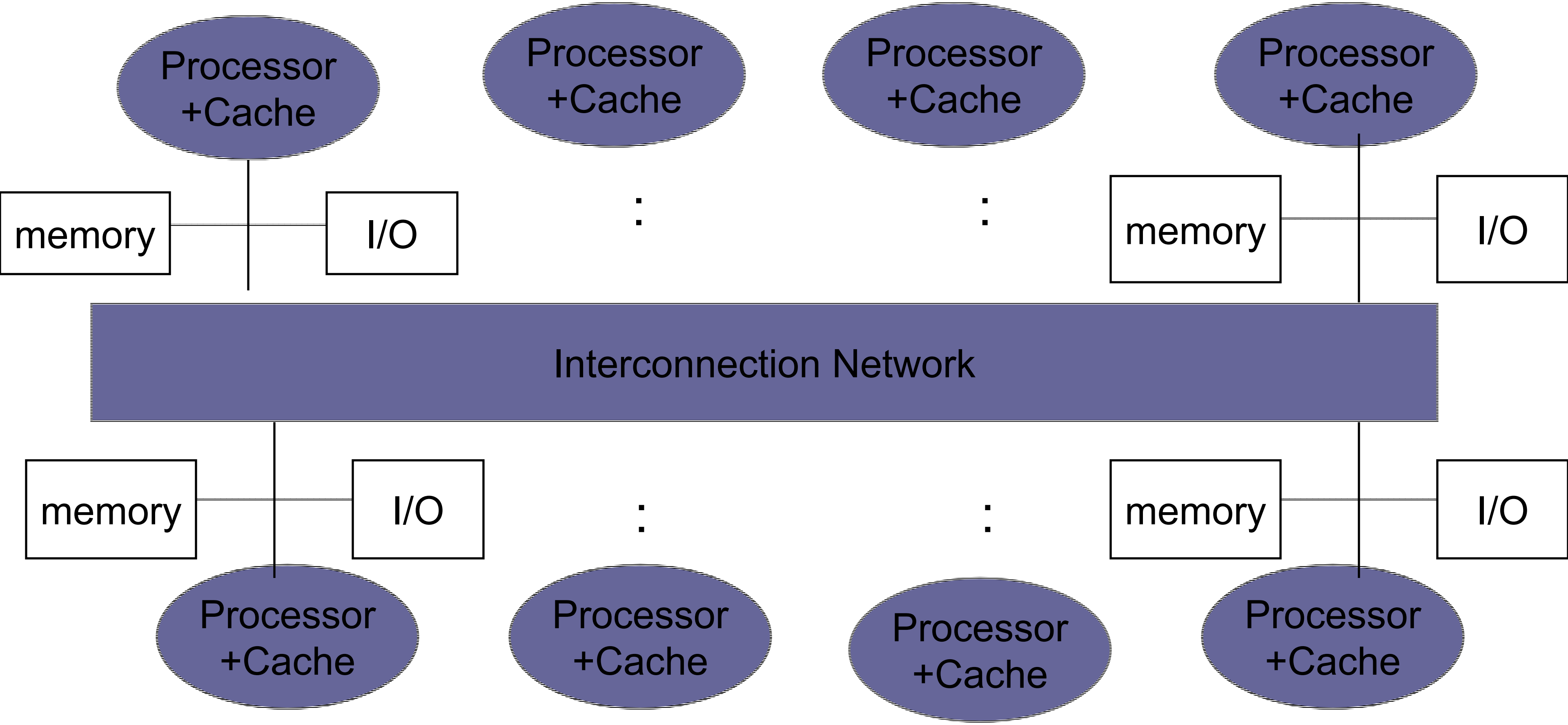


```
half = 100; /* 100 processors in multiprocessor*/
repeat
        synch(); /* wait for partial sum completion*/
        if (half%2 != 0 && Pn == 0)
              sum[0] = sum[0] + sum[half-1];
              /* Conditional sum needed when half is
              odd; Processor0 gets missing element */
        half = half/2; /* dividing line on who sums */
        if (Pn < half) sum[Pn] = sum[Pn] + sum[Pn+half];
until (half == 1); /* exit with final sum in Sum[0] */
```



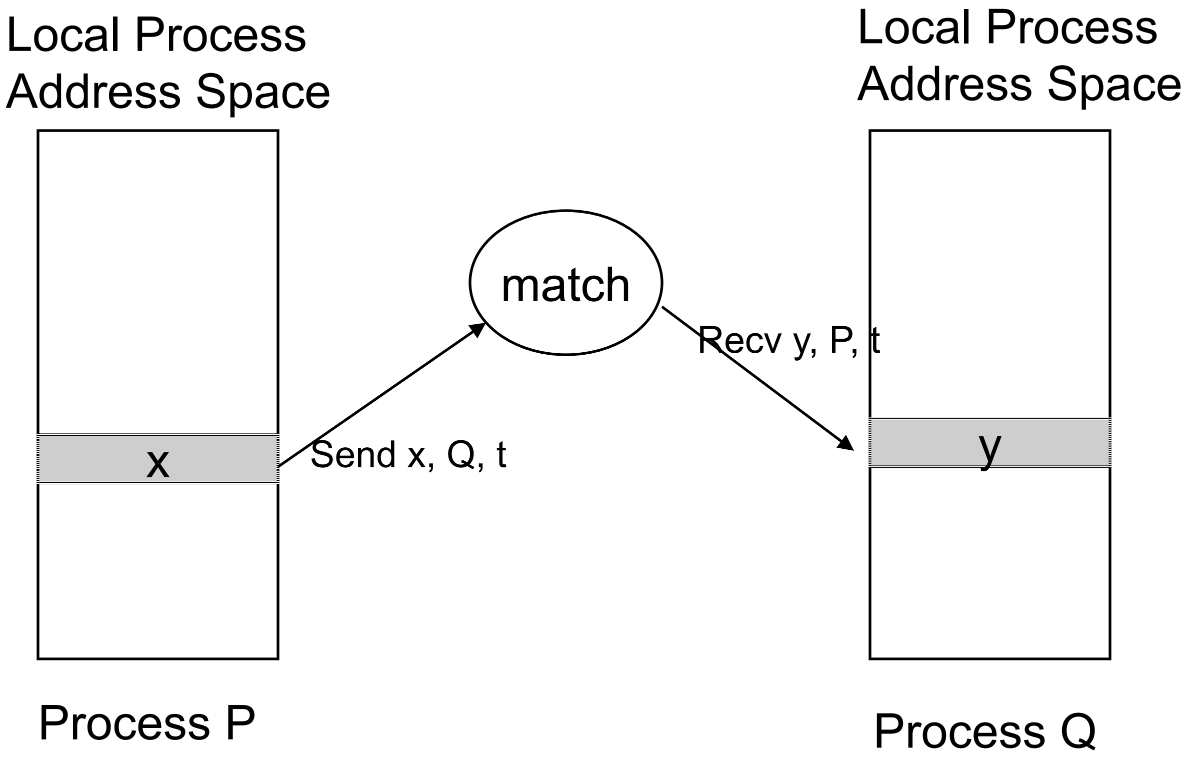
Message Passing Multiprocessors

- **Clusters: Collections of computers connected via I/O over standard network switches to form a message-passing multiprocessors**



Communication Models

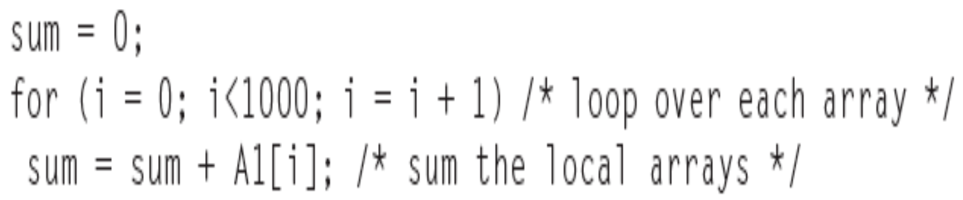
- **Multiple address spaces: Message Passing**



Parallel Program – Message Passing

- **sum 100,000 numbers & 100 processors (send & receive)**

First Step: each processor (Pn) sums his subset of numbers

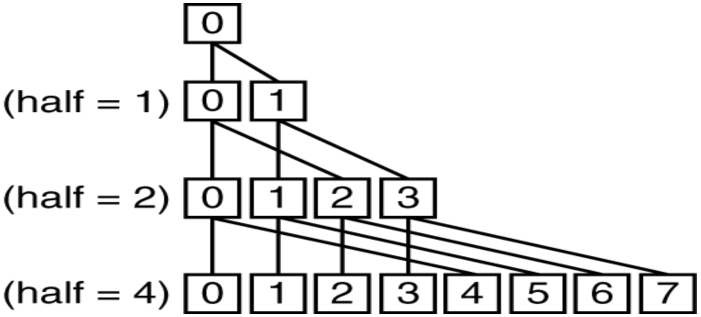


```
sum = 0;
for (i = 0; i<1000; i = i + 1) /* loop over each array */
 sum = sum + A1[i]; /* sum the local arrays */
```

Parallel Program – Message Passing

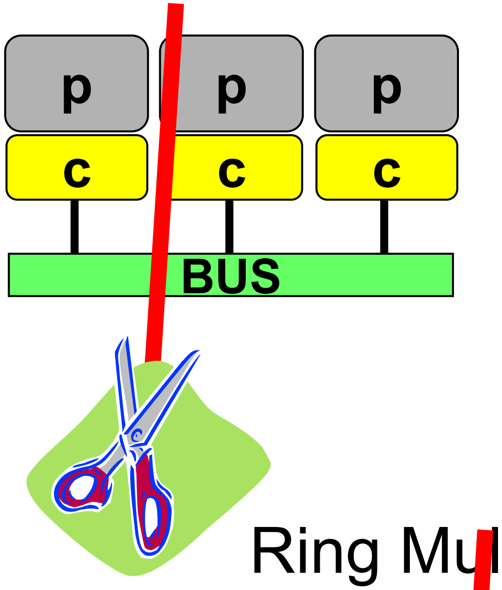
Second Step: Add partial sums via divide-and-conquer

```
limit = 100; half = 100;/* 100 processors */
repeat
  half = (half+1)/2; /* send vs. receive dividing line*/
  if (Pn >= half && Pn < limit) send(Pn - half, sum);
  if (Pn < (limit/2)) sum = sum + receive();
  limit = half; /* upper limit of senders */
until (half == 1); /* exit with final sum */
```



Bisection Bandwidth is Important

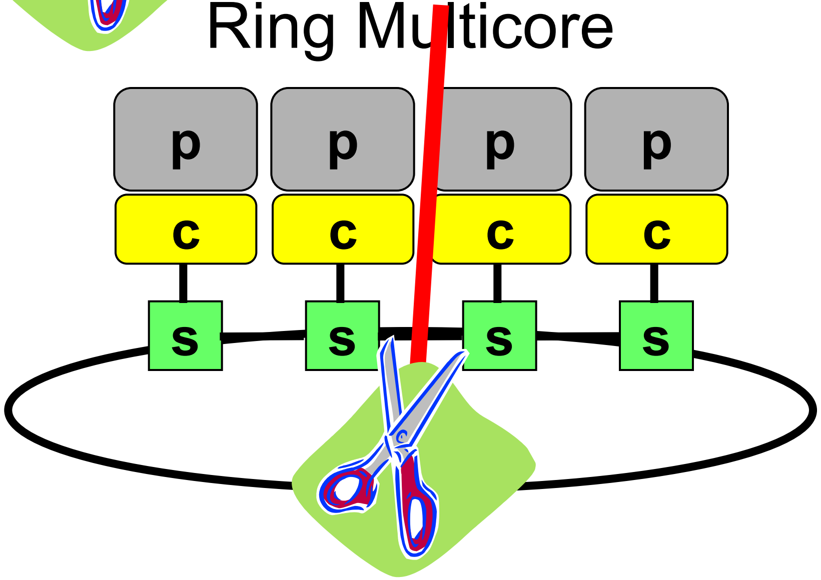
Bus Multicore



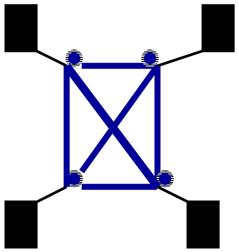
Total network bandwidth =
bandwidth-per-link x link_no

Bisection bandwidth =
the bandwidth between two equal parts of a multiprocessor

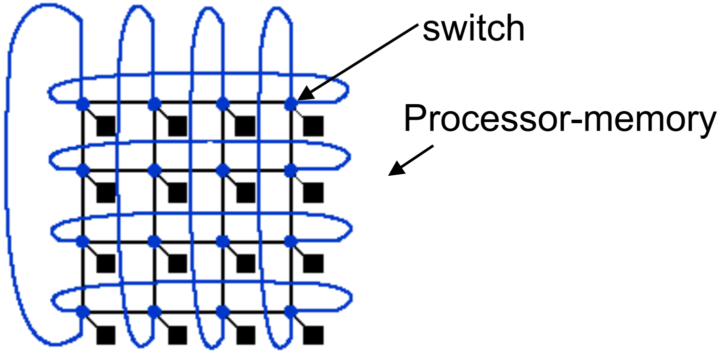
Ring Multicore



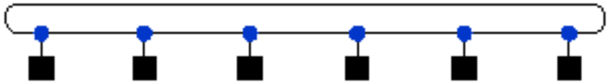
Network Topology



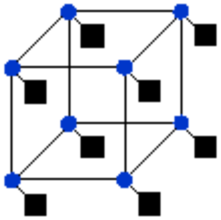
Fully-connected



2D torus

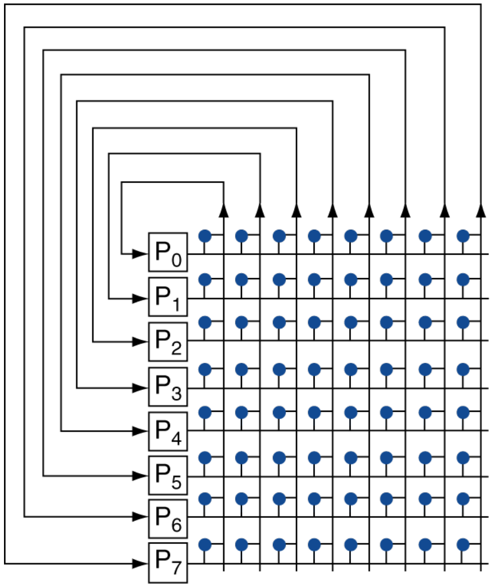


Ring

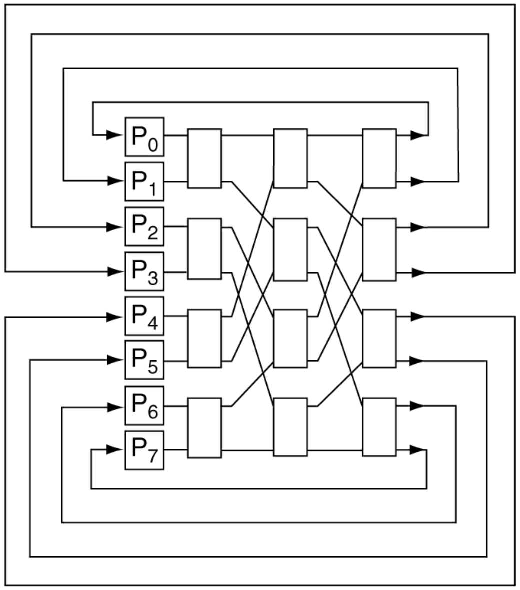


Cube

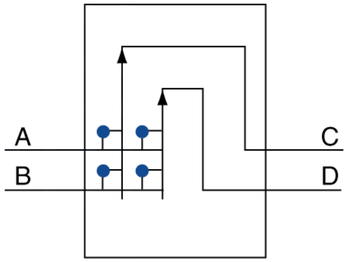
Multistage Networks



a. Crossbar



b. Omega network



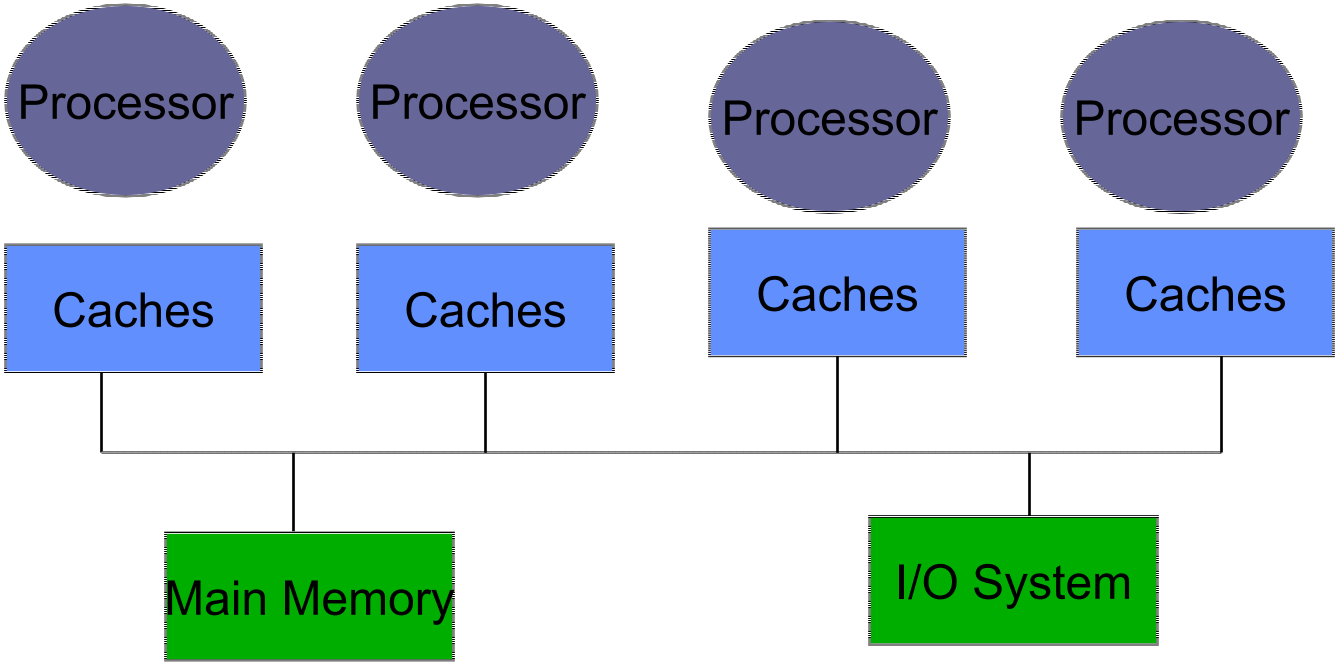
c. Omega network switch box

Network Characteristics

- Performance
  - Latency per message (unloaded network)
  - Throughput
    - Link bandwidth
    - Total network bandwidth
    - Bisection bandwidth
  - Congestion delays (depending on traffic)
- Cost
- Power
- Routability in silicon

Cache Coherency Problem

- Traffic per processor and the bus bandwidth determine the # of processors
- Caches can lower bus traffic
  - Cache coherency problem



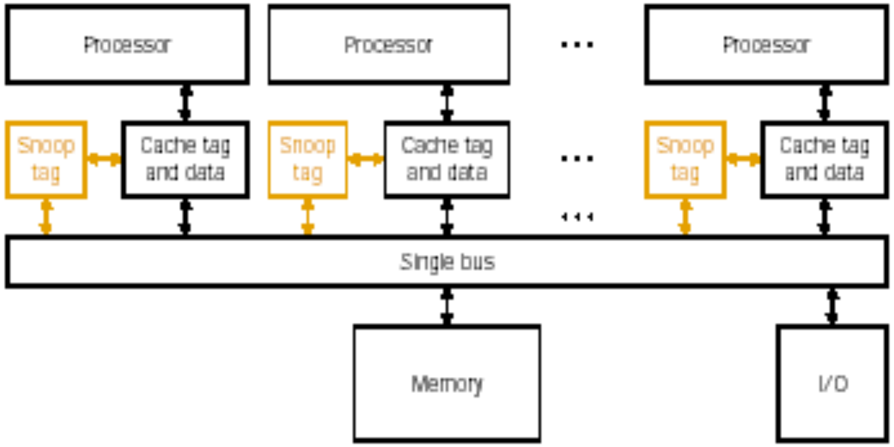
Cache Coherency

Time	Event	$ A	$ B	X (memory)
0				1
1	CPU A reads X	1		1
2	CPU B reads X	1	1	1
3	CPU A stores 0 into X	0	1	0

Cache Coherency Protocol

5.10

- Snooping Solution (Snoopy Bus):
  - Send all requests for data to all processors
  - Processors snoop to see if they have a copy and respond accordingly
  - Requires broadcast, since caching information is at processors
  - Works well with bus (natural broadcast medium)
  - Dominates for small scale machines (most of the market)



Basic Snoopy Protocols

- **Write Invalidate Protocol:**
  - Multiple readers, single writer
  - Write to shared data: an invalidate is sent to all caches which snoop and *invalidate* any copy
  - Read Miss:
    - » Write-through: memory is always up-to-date
    - » Write-back: snoop in caches to find most recent copy
- **Write Update Protocol:**
  - Write to shared data: broadcast on bus, processors snoop, and *update* copies
  - Read miss: memory is always up-to-date
- **What happens if two processors try to write to the same shared data word in the same clock cycle?**
  - Write serialization: bus serializes requests

Basic Snoopy Protocols

- **Invalidation** write back

Processor activity	Bus activity	Contents of CPU A' cache	Contents of CPU B's cache	Contexts of memory location X
				0
CPU A reads X	Cache miss for X	0		
CPU B reads X	Cache miss for X	0	0	0
CPU A writes a 1 to X	Invalidation for X	1		0
CPU B reads X	Cache miss for X	1	1	1

- **Update**

Processor activity	Bus activity	Contents of CPU A' cache	Contents of CPU B's cache	Contexts of memory location X
				0
CPU A reads X	Cache miss for X	0		
CPU B reads X	Cache miss for X	0	0	0
CPU A writes a 1 to X	Write broadcast of X	1	1	1
CPU B reads X		1	1	1

Basic Snoopy Protocols

- **Write Invalidate versus Broadcast:**
  - Invalidate requires one transaction per write-run
  - Invalidate uses spatial locality: one transaction per block
  - Broadcast: BW (increased) vs. latency (decreased) tradeoff

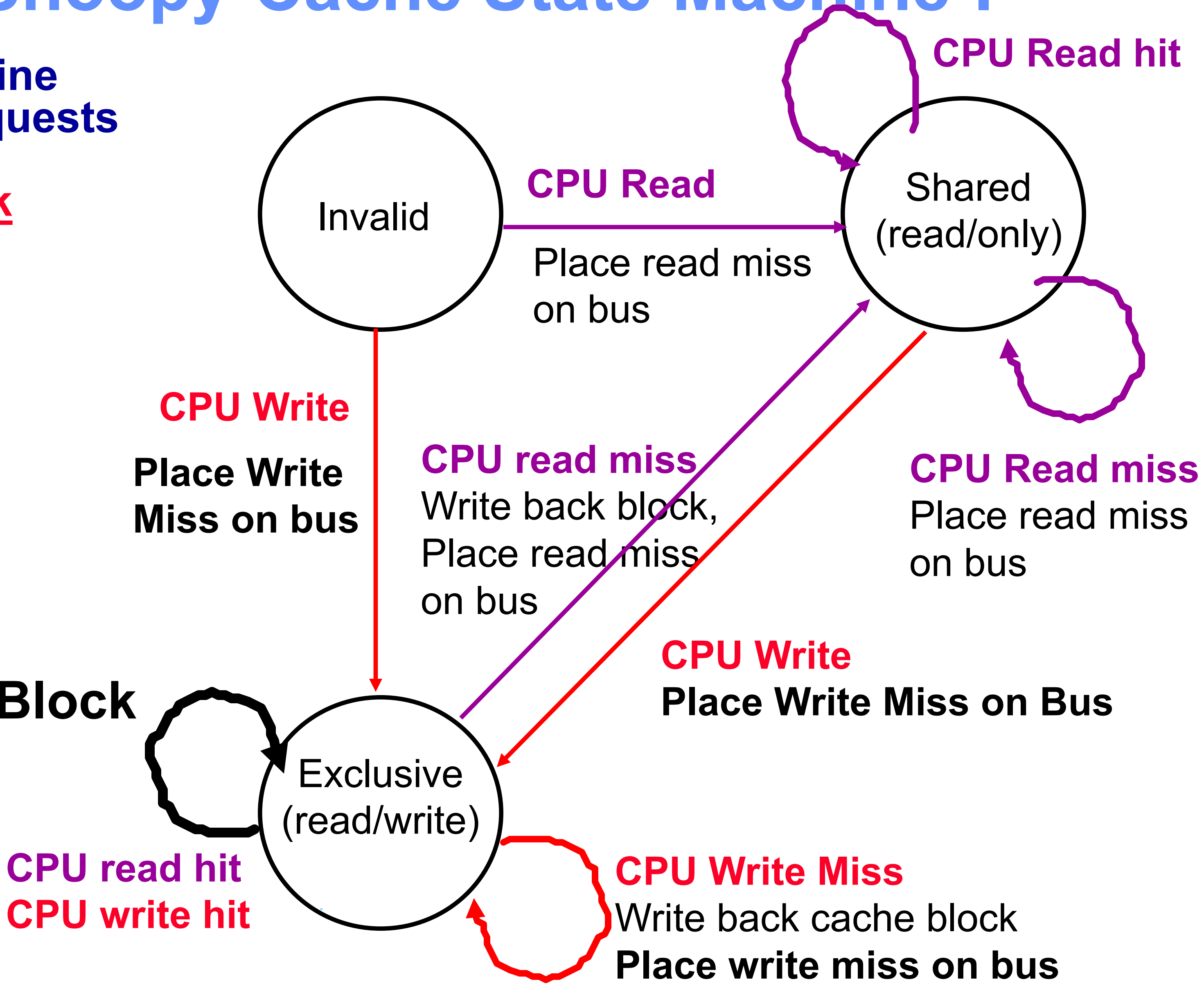
Invalidate protocol is more popular than update !

An Example Snoopy Protocol

- Invalidation protocol, write-back cache
- Each block of memory is in one state:
  - Clean in all caches and up-to-date in memory
  - OR Dirty in exactly one cache
  - OR Not in any caches
- Each cache block is in one state:
  - Shared: block can be read
  - OR Exclusive: cache has only copy, its writeable, and dirty
  - OR Invalid: block contains no data
- Read misses: cause all caches to snoop
- Writes to clean line are treated as misses

Snoopy-Cache State Machine-I

- **State machine for *CPU* requests for each cache block**



Invalid

Shared (read/only)

Exclusive (read/write)

CPU Read hit

CPU Read

Place read miss on bus

CPU Write

Place Write Miss on bus

CPU read miss

Write back block,

Place read miss on bus

CPU Read miss

Place read miss on bus

CPU Write

Place Write Miss on Bus

Cache Block State

CPU read hit

CPU write hit

CPU Write Miss

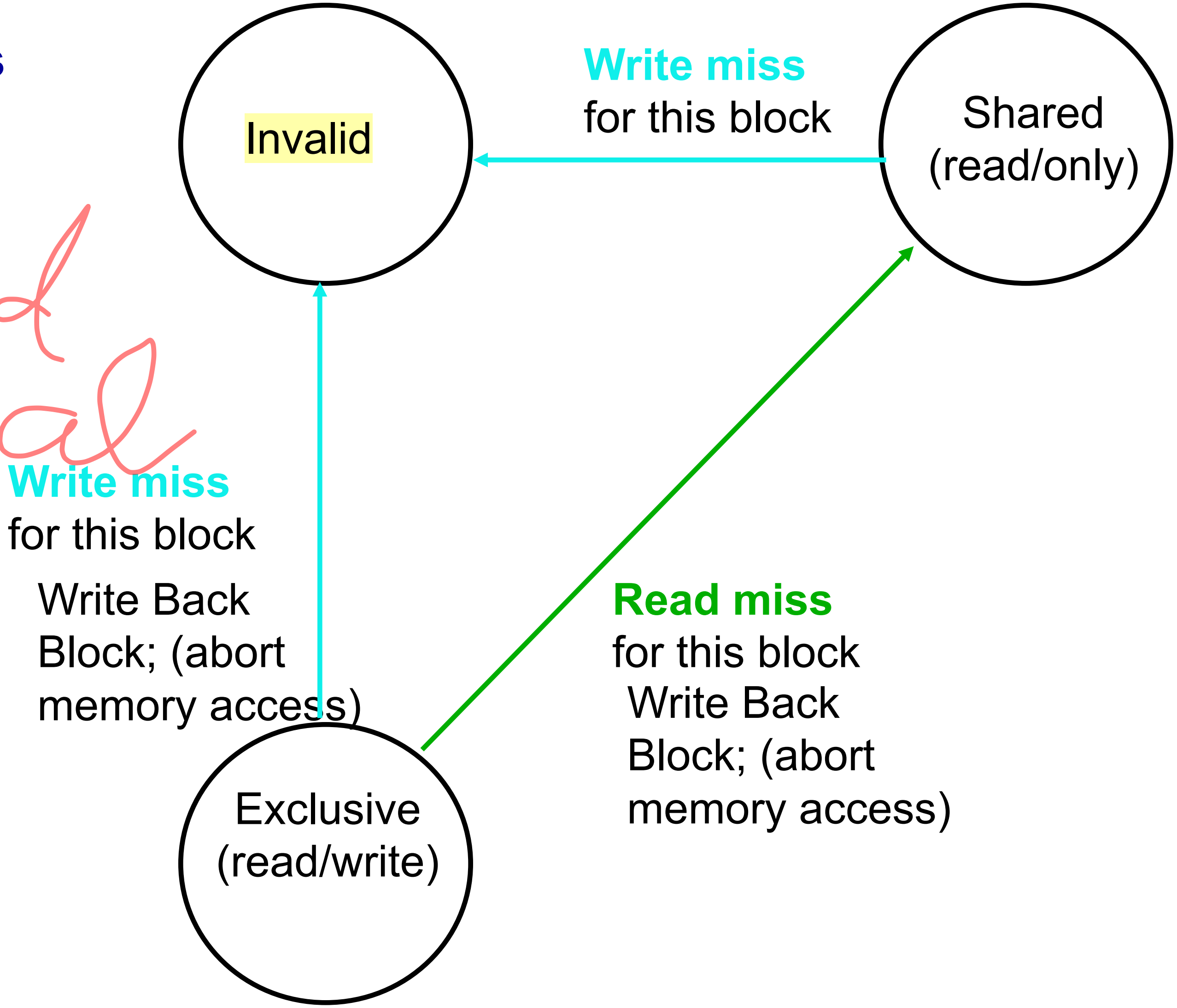
Write back cache block

Place write miss on bus

Snoopy-Cache State Machine-II

- **State machine for *bus* requests for each cache block**

invalid protocal



Invalid

Shared
(read/only)

Exclusive
(read/write)

Write miss
for this block

Write miss
for this block

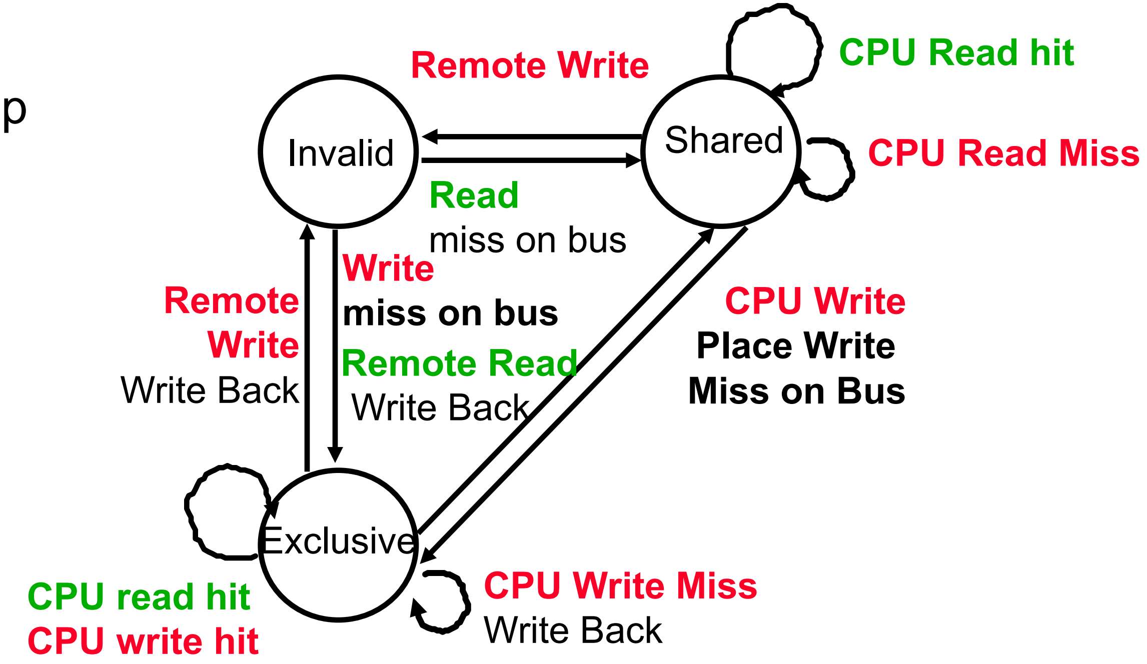
Write Back
Block; (abort
memory access)

Read miss
for this block
Write Back
Block; (abort
memory access)

Example

	Processor 1			Processor 2			Bus				Memory	
	P1			*P2*			*Bus*				*Memory*	
step	*State*	*Addr*	*Value*	*State*	*Addr*	*Value*	*Action*	*Proc.*	*Addr*	*Value*	*Addr*	*Value*
P1: Write 10 to A1												
P1: Read A1												
P2: Read A1												
P2: Write 20 to A1												
P2: Write 40 to A2												

Assumes initial cache state is invalid and A1 and A2 map to same cache block, but A1 != A2

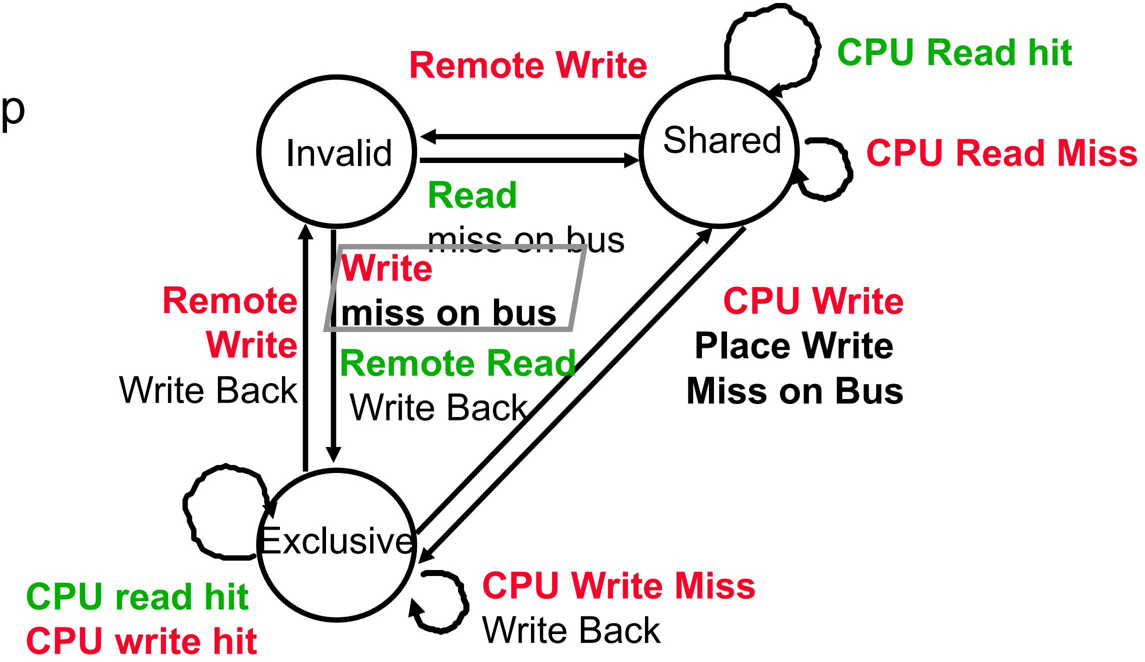


Example: Step 1

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1: Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1												
P2: Read A1												
P2: Write 20 to A1												
P2: Write 40 to A2												

Assumes initial cache state is invalid and A1 and A2 map to same cache block, but A1 != A2.

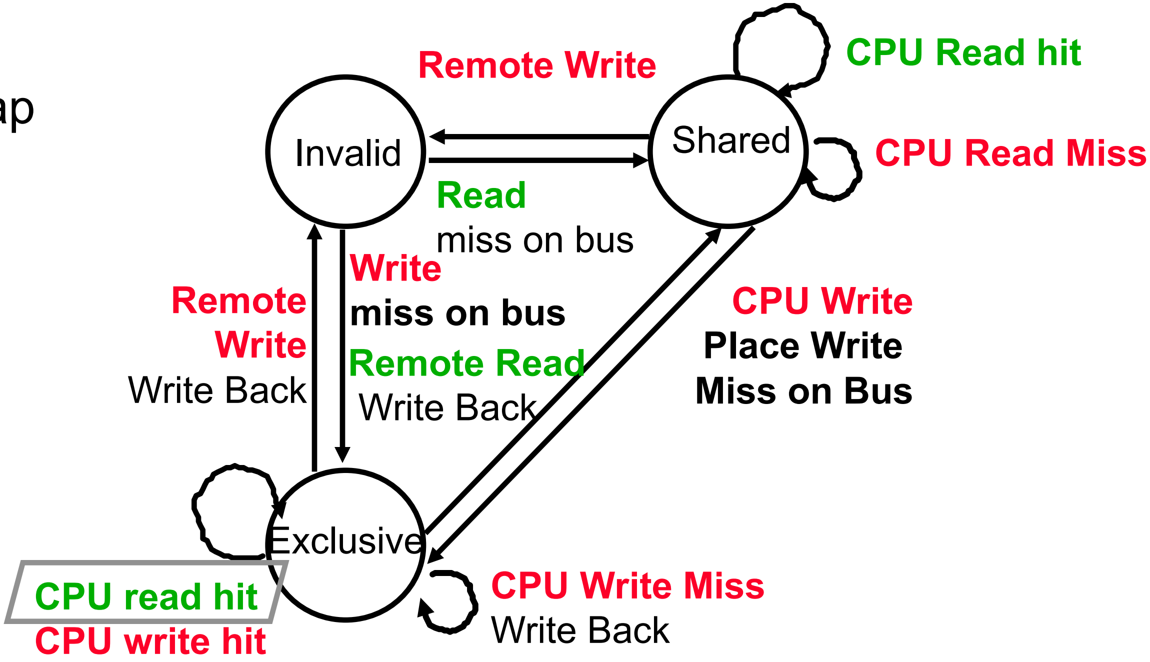
Active arrow =



Example: Step 2

	P1			P2			Bus				Memory	
step	*State*	*Addr*	*Value*	*State*	*Addr*	*Value*	*Action*	*Proc.*	*Addr*	*Value*	*Addr*	*Value*
P1: Write 10 to A1	*Excl.*	*A1*	*10*				*WrMs*	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1												
P2: Write 20 to A1												
P2: Write 40 to A2												

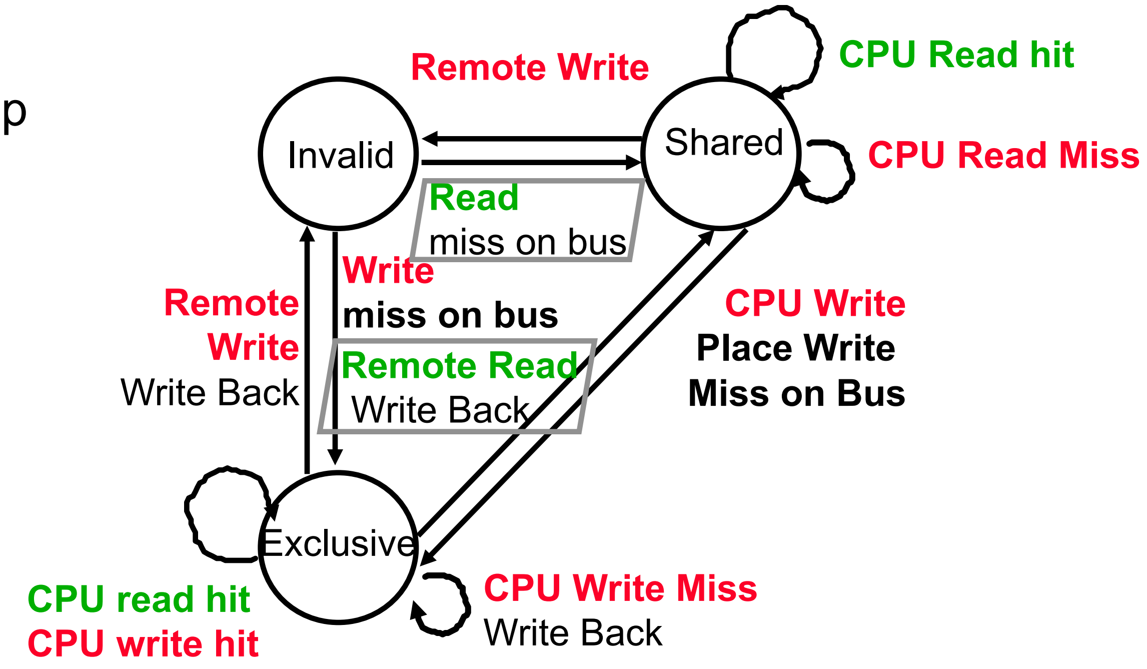
Assumes initial cache state
is invalid and A1 and A2 map
to same cache block,
but A1 != A2



Example: Step 3

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1: Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1				Shar.	A1		RdMs	P2	A1			
	Shar.	A1	10				WrBk	P1	A1	10	A1	10
				Shar.	A1	10	RdDa	P2	A1	10	A1	10
P2: Write 20 to A1												
P2: Write 40 to A2												

Assumes initial cache state
is invalid and A1 and A2 map
to same cache block,
but A1 != A2.

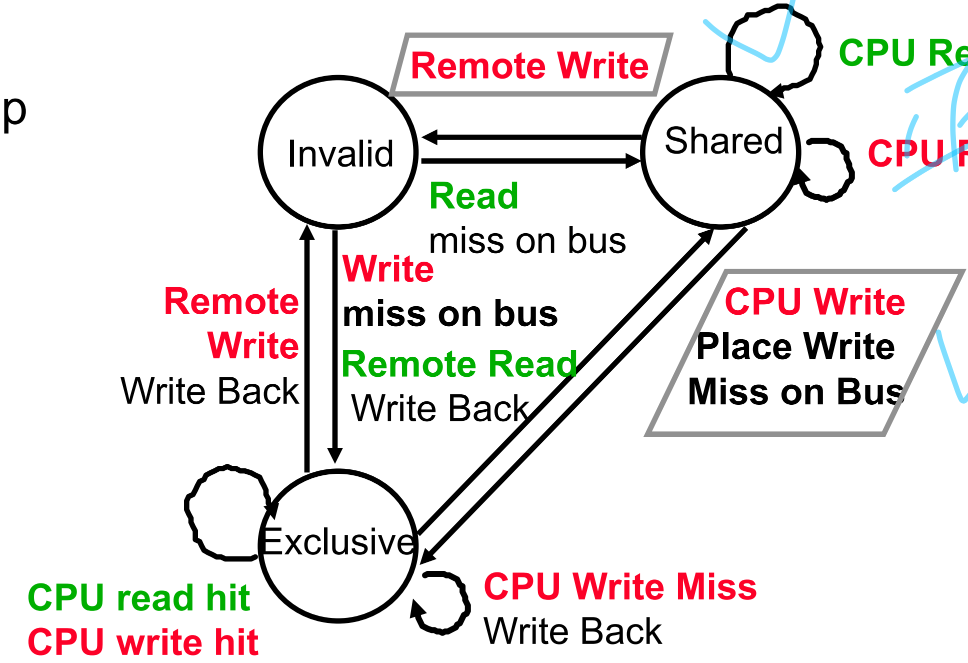


Example: Step 4

	P1			*P2*			*Bus*				*Memory*	
step	*State*	*Addr*	*Value*	*State*	*Addr*	*Value*	*Action*	*Proc.*	*Addr*	*Value*	*Addr*	*Value*
P1: Write 10 to A1	*Excl.*	*A1*	*10*				*WrMs*	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1				*Shar.*	*A1*		*RdMs*	P2	A1			
	Shar.	A1	10				*WrBk*	P1	A1	10	*A1*	*10*
				Shar.	A1	*10*	*RdDa*	P2	A1	10	A1	10
P2: Write 20 to A1	*Inv.*			*Excl.*	A1	*20*	*WrMs*	P2	A1		A1	10
P2: Write 40 to A2												10
												10

memory 没有 修 value

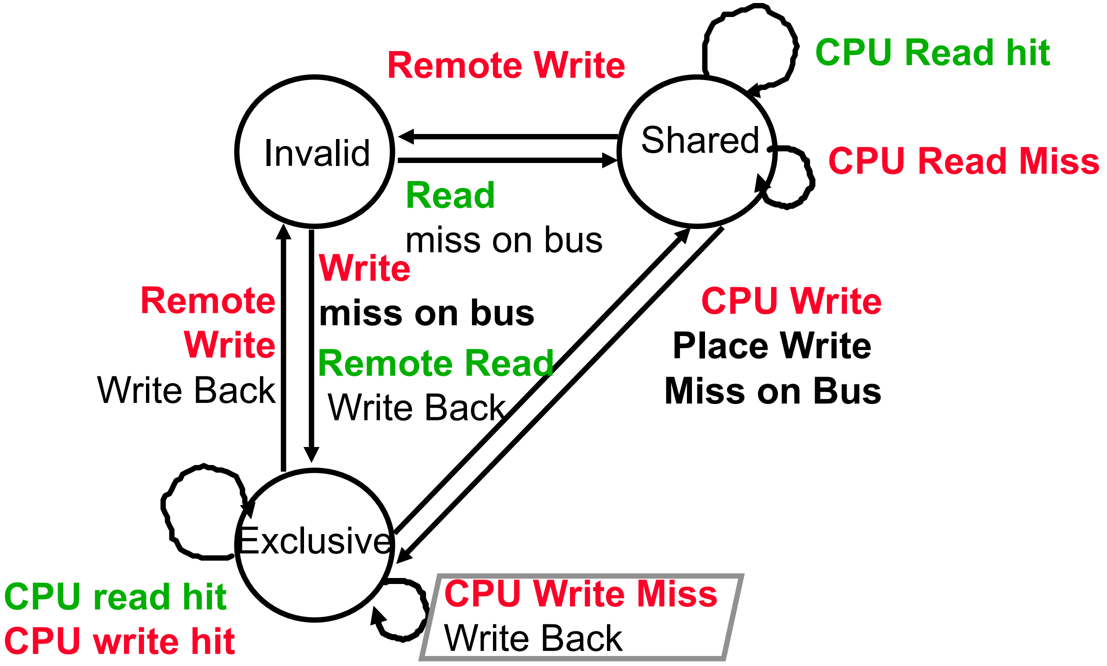
Assumes initial cache state is invalid and A1 and A2 map to same cache block, but A1 != A2



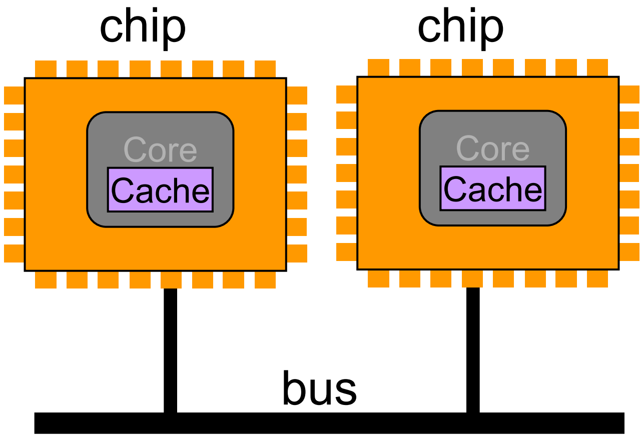
Example: Step 5

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1: Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1				Shar.	A1		RdMs	P2	A1			
	Shar.	A1	10				WrBk	P1	A1	10	A1	10
				Shar.	A1	10	RdDa	P2	A1	10	A1	10
P2: Write 20 to A1	Inv.			Excl.	A1	20	WrMs	P2	A1		A1	10
P2: Write 40 to A2							WrMs	P2	A2		A1	10
				Excl.	A2	40	WrBk	P2	A1	20	A1	20

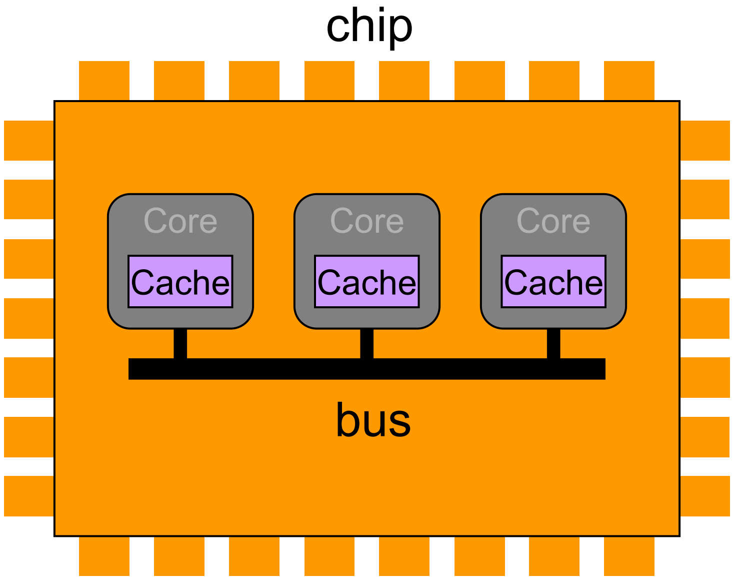
Assumes initial cache state
is invalid and A1 and A2 map
to same cache block,
but A1 != A2



What is multi-core?



Off-chip bus



On-chip bus

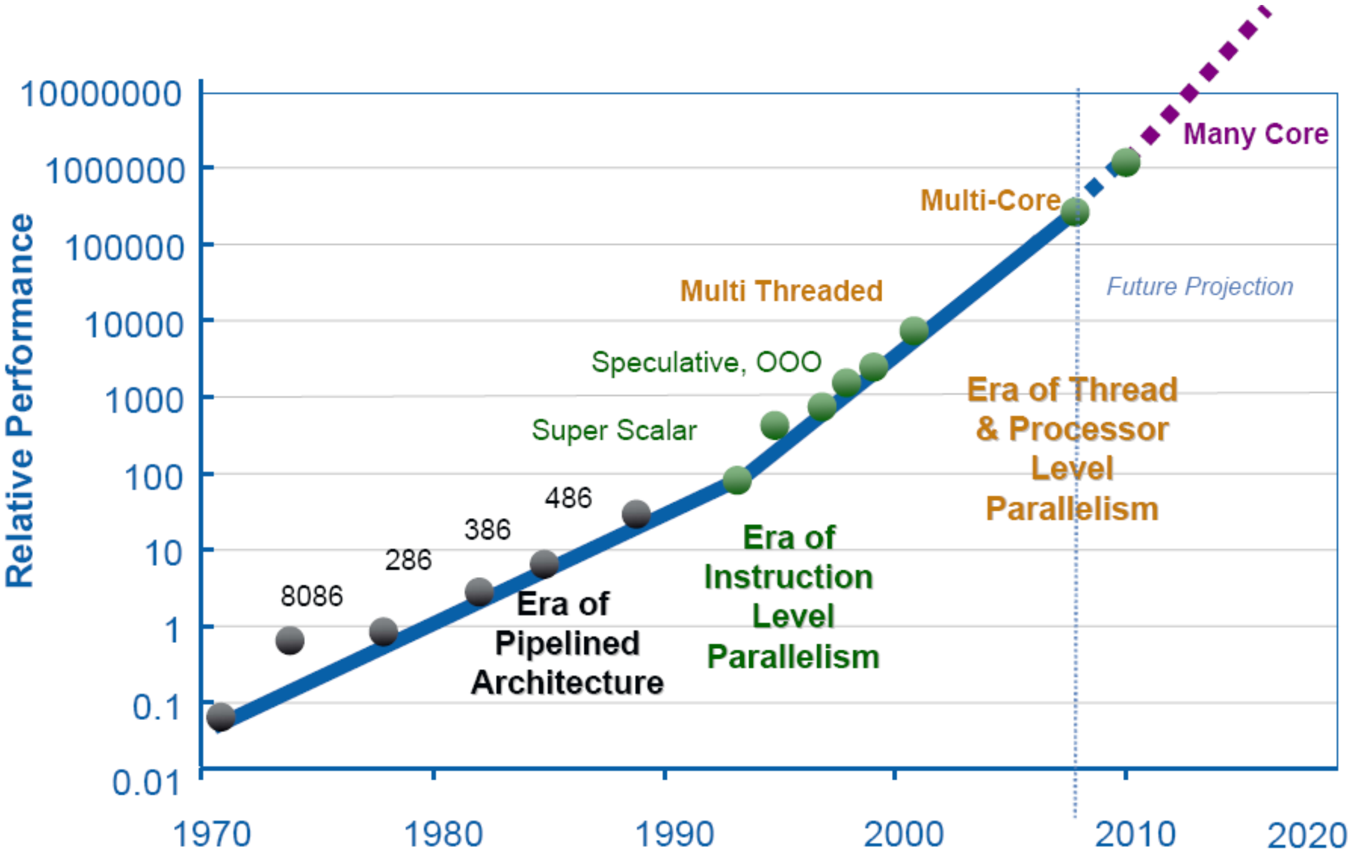
Why Multi-core?

- Old Conventional Wisdom: Power is free, Transistors expensive
- New Conventional Wisdom: "Power wall" Power expensive, Xtors free
  (Can put more on chip than can afford to turn on)
- Old CW: Sufficiently increasing Instruction Level Parallelism via compilers, innovation (Out-of-order, speculation, VLIW, …)
- New CW: "ILP wall" law of diminishing returns on more HW for ILP
- Old CW: Multiplies are slow, Memory access is fast
- New CW: "Memory wall" Memory slow, multiplies fast
  (200 clock cycles to DRAM memory, 4 clocks for multiply)
- Old CW: Uniprocessor performance 2X / 1.5 yrs
- New CW: Power Wall + ILP Wall + Memory Wall = Brick Wall
  - Uniprocessor performance now 2X / 5(?) yrs

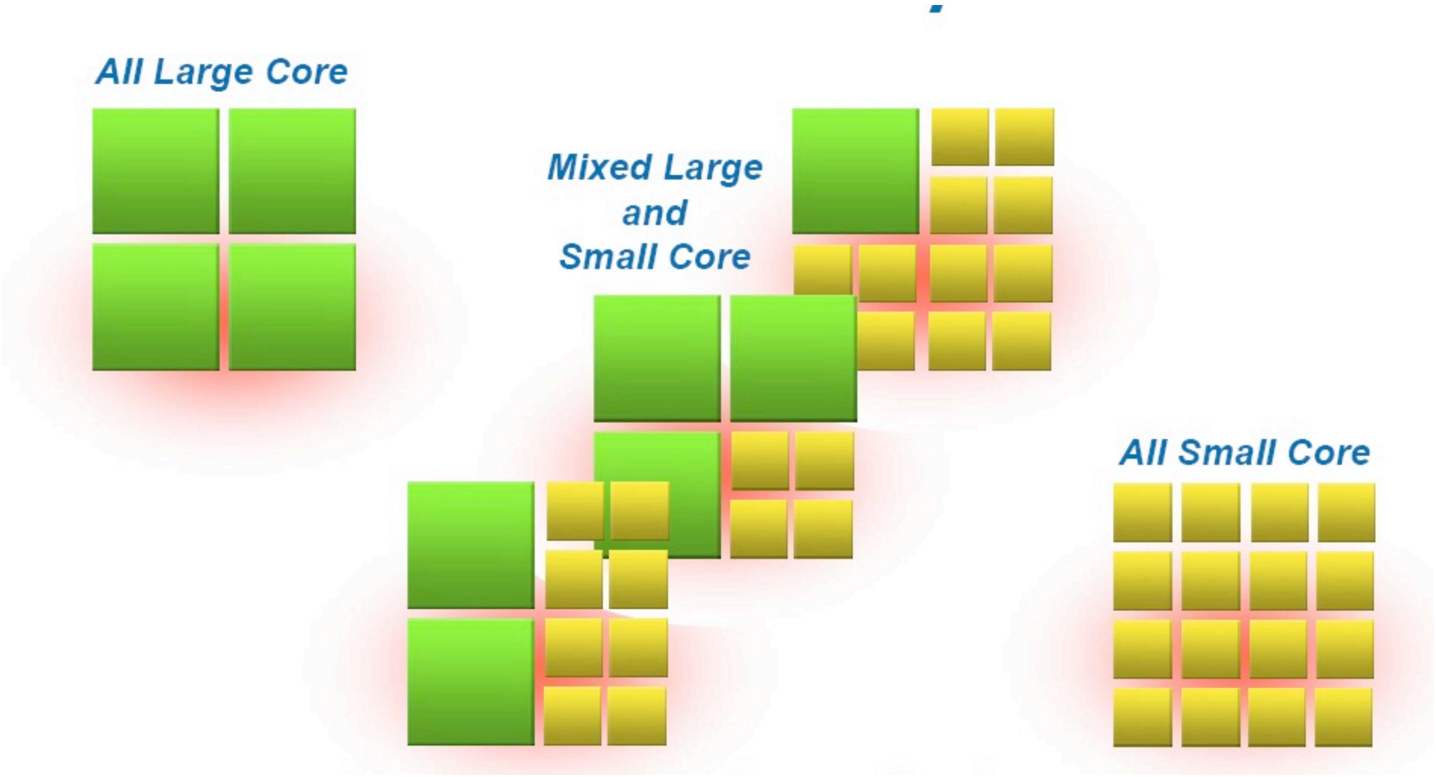
⇒ Sea change in chip design: multiple "cores"
(2X processors per chip / ~ 2 years)

» More simpler processors are more power efficient

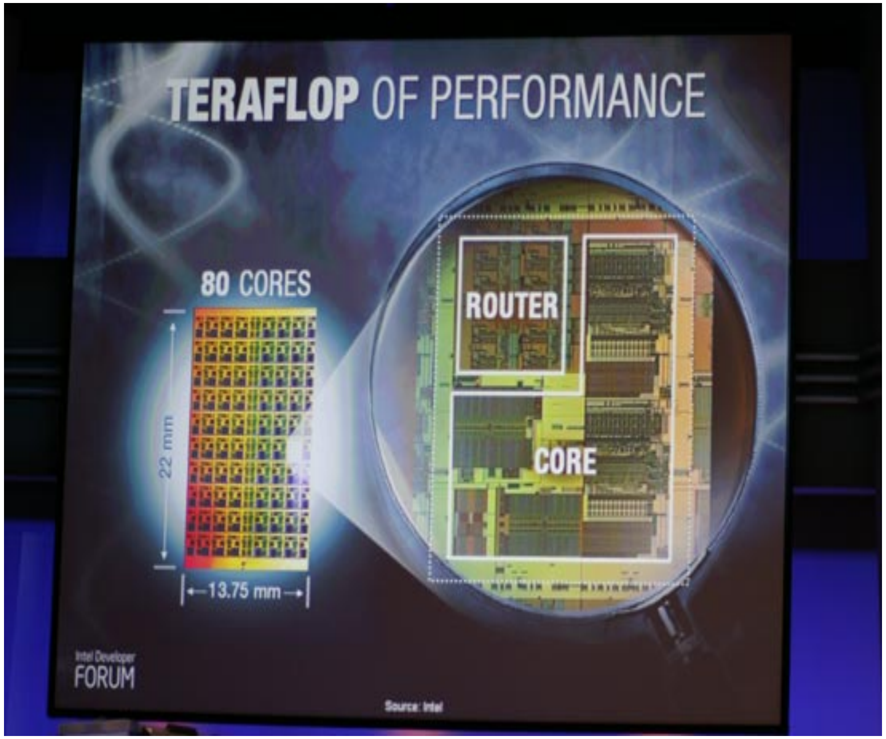
Parallelism for Energy Efficiency Present and Future



From Multicore to Manycore



Tera-Scale Computing

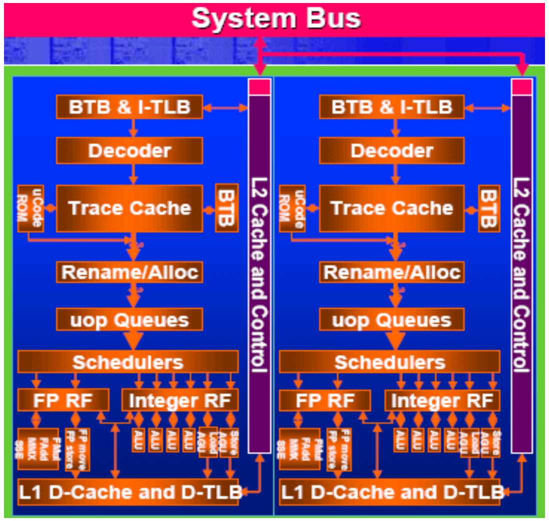


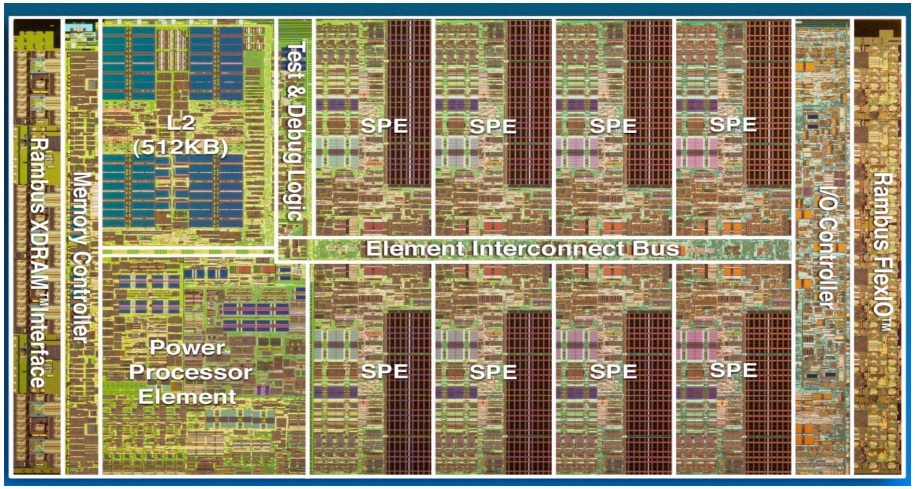
TERAFLOP OF PERFORMANCE

Source: Intel

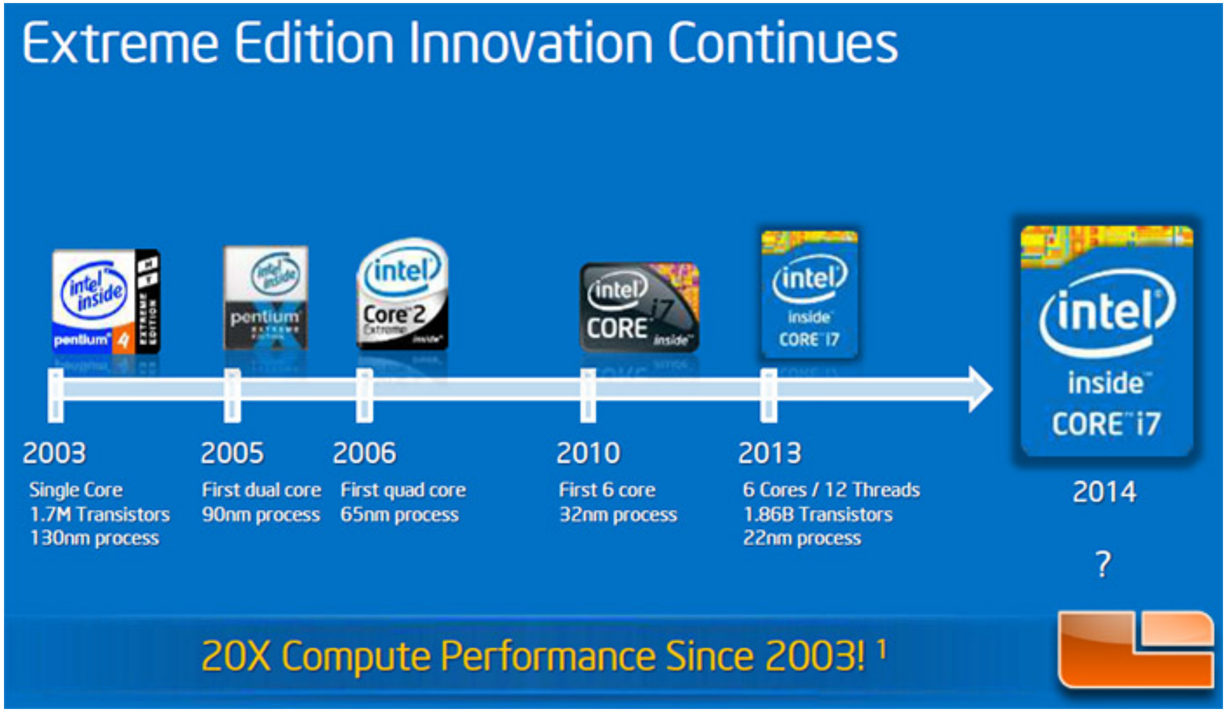
Chip-Multiprocessor

- **Chip-Multiprocessor (2005 ~)**
  - A chip contains multiple cores
  - Example: Intel Xeon, AMD Opteron, Sony Playstation, Sun T1 Niagara





Intel’s Multi-Core Roadmap



Extreme Edition Innovation Continues

2003
Single Core
1.7M Transistors
130nm process

2005
First dual core
90nm process

2006
First quad core
65nm process

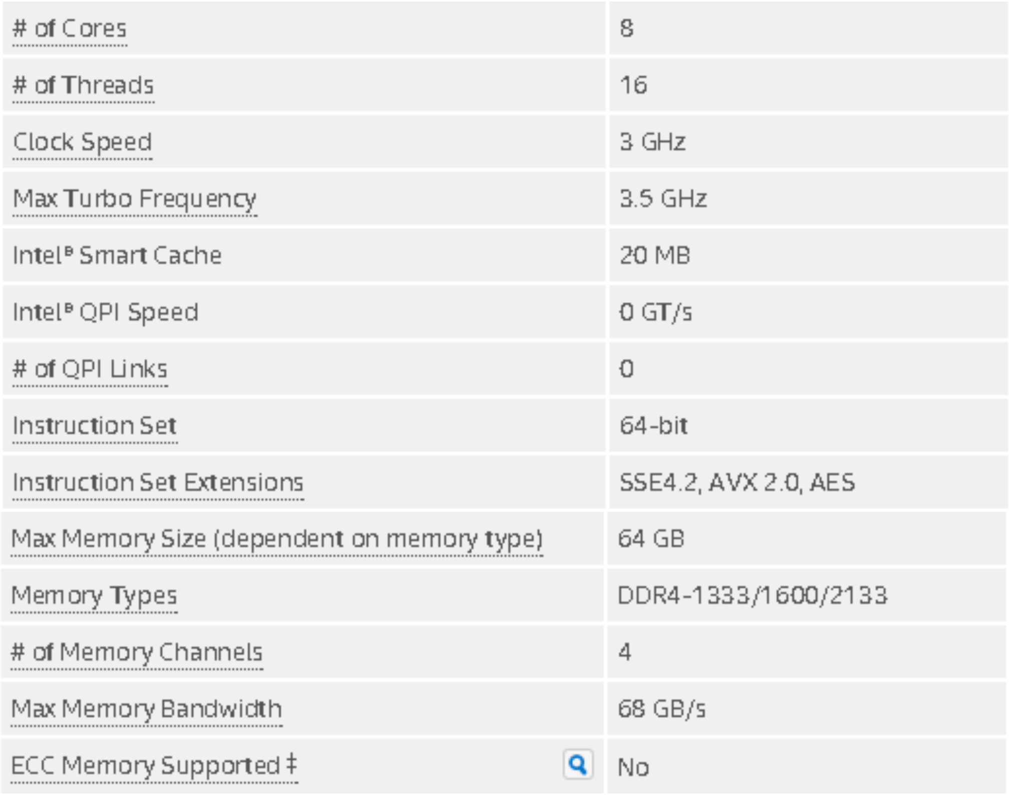
2010
First 6 core
32nm process

2013
6 Cores / 12 Threads
1.86B Transistors
22nm process

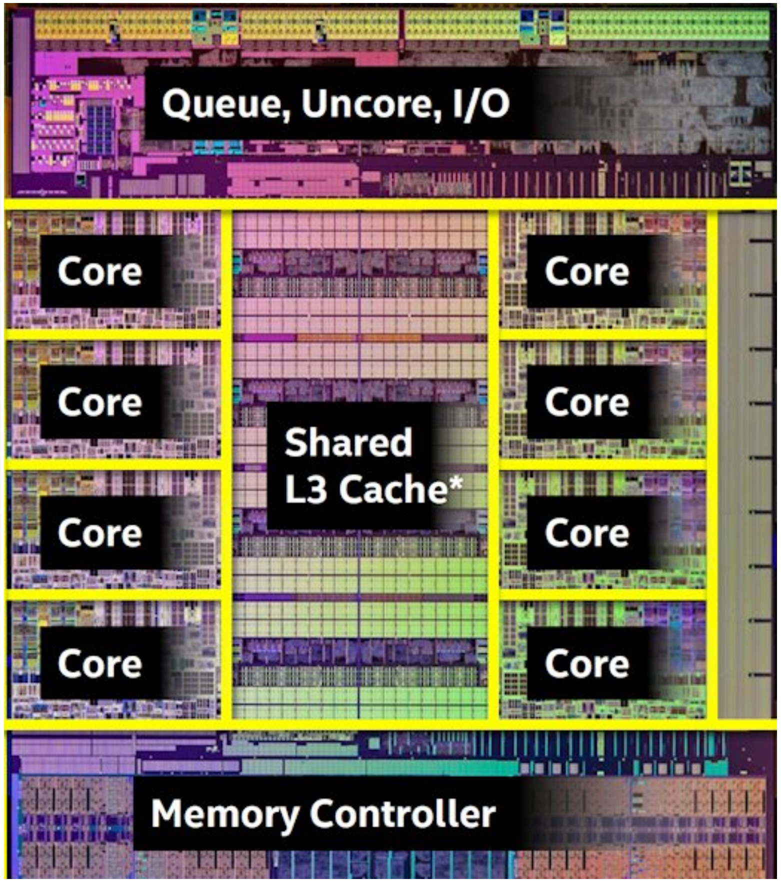
2014
?

20X Compute Performance Since 2003! [1]

Intel® Core™ i7-5960X Processor

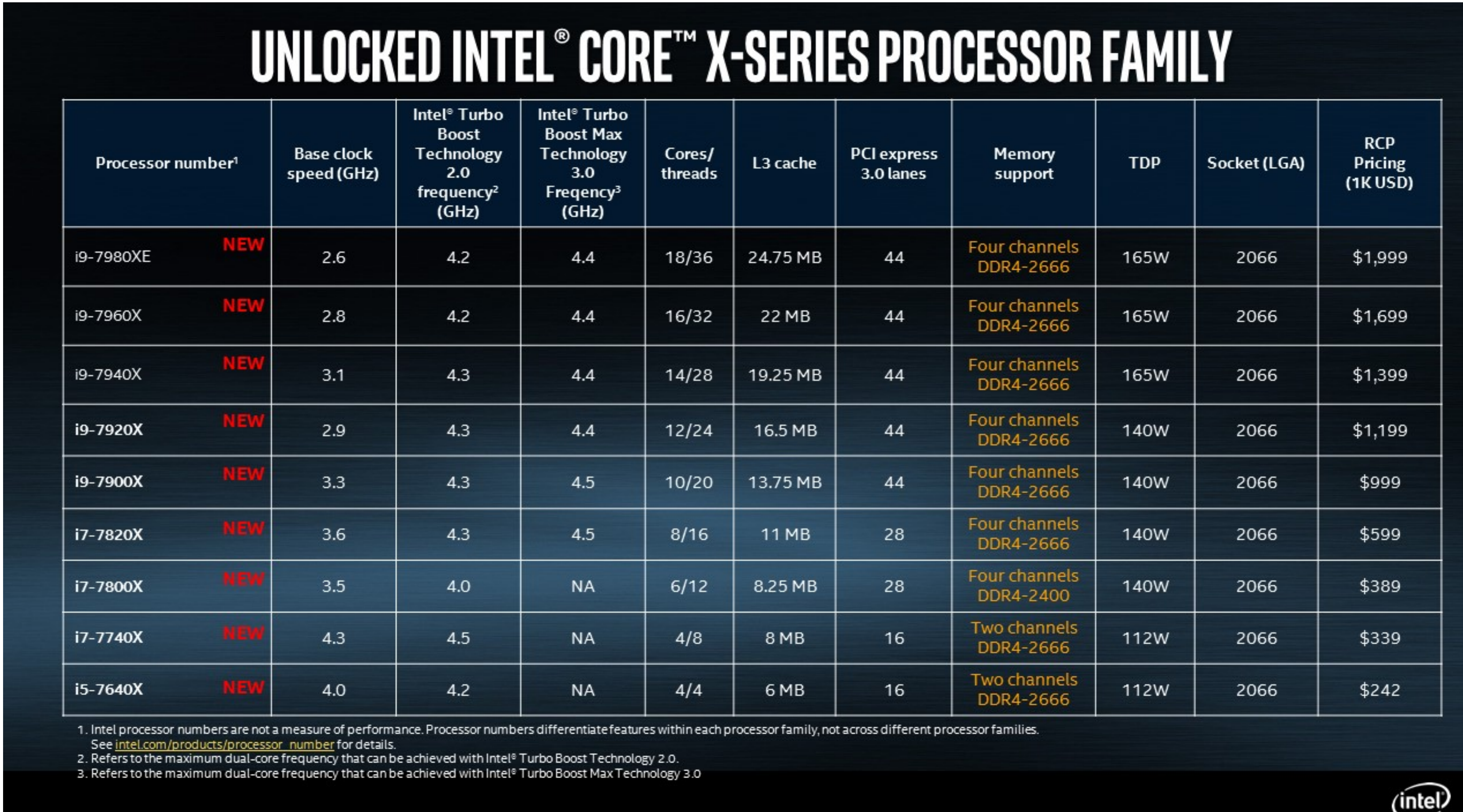


# of Cores	8
# of Threads	16
Clock Speed	3 GHz
Max Turbo Frequency	3.5 GHz
Intel® Smart Cache	20 MB
Intel® QPI Speed	0 GT/s
# of QPI Links	0
Instruction Set	64-bit
Instruction Set Extensions	SSE4.2, AVX 2.0, AES
Max Memory Size (dependent on memory type)	64 GB
Memory Types	DDR4-1333/1600/2133
# of Memory Channels	4
Max Memory Bandwidth	68 GB/s
ECC Memory Supported ‡	No



http://ark.intel.com/products/82930/Intel-Core-i7-5960X-Processor-Extreme-Edition-20M-Cache-up-to-3_50-GHz#@specifications

Intel Core I9 ~ announced in 2017

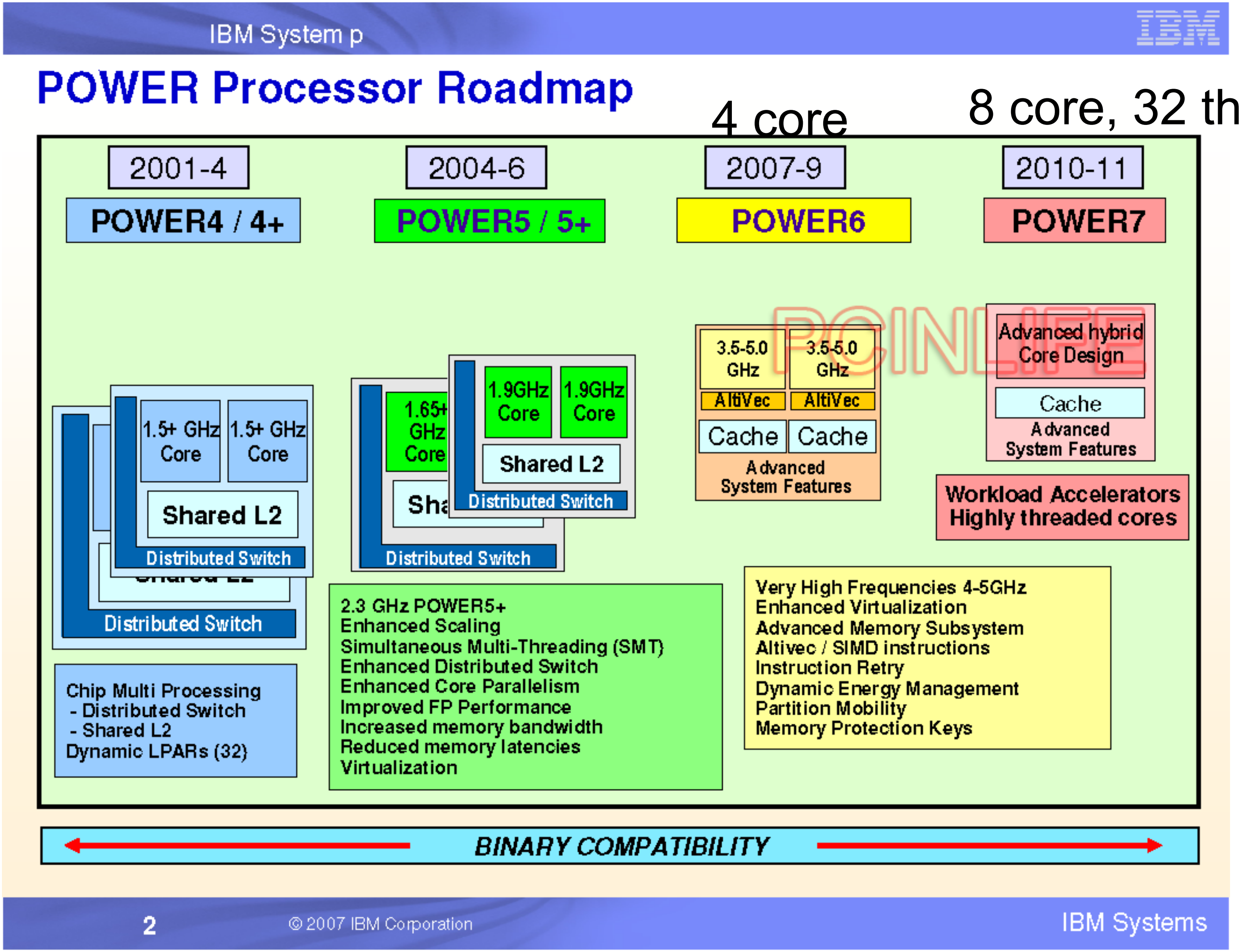


UNLOCKED INTEL® CORE™ X-SERIES PROCESSOR FAMILY

Processor number[1]		Base clock speed (GHz)	Intel® Turbo Boost Technology 2.0 frequency[2] (GHz)	Intel® Turbo Boost Max Technology 3.0 Freqency[3] (GHz)	Cores/ threads	L3 cache	PCI express 3.0 lanes	Memory support	TDP	Socket (LGA)	RCP Pricing (1K USD)
i9-7980XE	NEW	2.6	4.2	4.4	18/36	24.75 MB	44	Four channels DDR4-2666	165W	2066	$1,999
i9-7960X	NEW	2.8	4.2	4.4	16/32	22 MB	44	Four channels DDR4-2666	165W	2066	$1,699
i9-7940X	NEW	3.1	4.3	4.4	14/28	19.25 MB	44	Four channels DDR4-2666	165W	2066	$1,399
i9-7920X	NEW	2.9	4.3	4.4	12/24	16.5 MB	44	Four channels DDR4-2666	140W	2066	$1,199
i9-7900X	NEW	3.3	4.3	4.5	10/20	13.75 MB	44	Four channels DDR4-2666	140W	2066	$999
i7-7820X	NEW	3.6	4.3	4.5	8/16	11 MB	28	Four channels DDR4-2666	140W	2066	$599
i7-7800X	NEW	3.5	4.0	NA	6/12	8.25 MB	28	Four channels DDR4-2400	140W	2066	$389
i7-7740X	NEW	4.3	4.5	NA	4/8	8 MB	16	Two channels DDR4-2666	112W	2066	$339
i5-7640X	NEW	4.0	4.2	NA	4/4	6 MB	16	Two channels DDR4-2666	112W	2066	$242

1. Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. See intel.com/products/processor_number for details.
2. Refers to the maximum dual-core frequency that can be achieved with Intel® Turbo Boost Technology 2.0.
3. Refers to the maximum dual-core frequency that can be achieved with Intel® Turbo Boost Max Technology 3.0

IBM: Power Processor



IBM System p

POWER Processor Roadmap

4 core

8 core, 32 threads

2001-4	2004-6	2007-9	2010-11
POWER4 / 4+	POWER5 / 5+	POWER6	POWER7

Workload Accelerators
Highly threaded cores

Chip Multi Processing
- Distributed Switch
- Shared L2

Dynamic LPARs (32)

2.3 GHz POWER5+
Enhanced Scaling
Simultaneous Multi-Threading (SMT)
Enhanced Distributed Switch
Enhanced Core Parallelism
Improved FP Performance
Increased memory bandwidth
Reduced memory latencies
Virtualization

Very High Frequencies 4-5GHz
Enhanced Virtualization
Advanced Memory Subsystem
Altivec / SIMD instructions
Instruction Retry
Dynamic Energy Management
Partition Mobility
Memory Protection Keys

BINARY COMPATIBILITY

POWER8 Processor

Technology

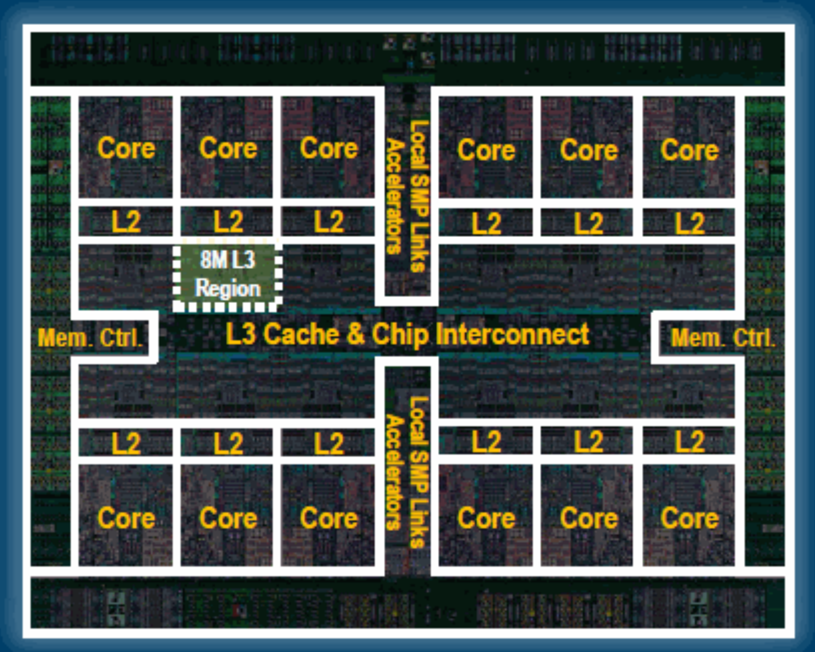
- 22nm SOI, eDRAM, 15 ML 650mm2

Cores

- 12 cores (SMT8)
- 8 dispatch, 10 issue, 16 exec pipe
- 2X internal data flows/queues
- Enhanced prefetching
- 64K data cache, 32K instruction cache

Accelerators

- Crypto & memory expansion
- Transactional Memory
- VMM assist
- Data Move / VM Mobility



Energy Management

- On-chip Power Management Micro-controller
- Integrated Per-core VRM
- Critical Path Monitors

Caches

- 512 KB SRAM L2 / core
- 96 MB eDRAM shared L3
- Up to 128 MB eDRAM L4 (off-chip)

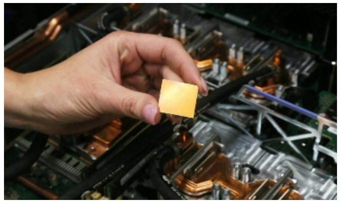
Memory

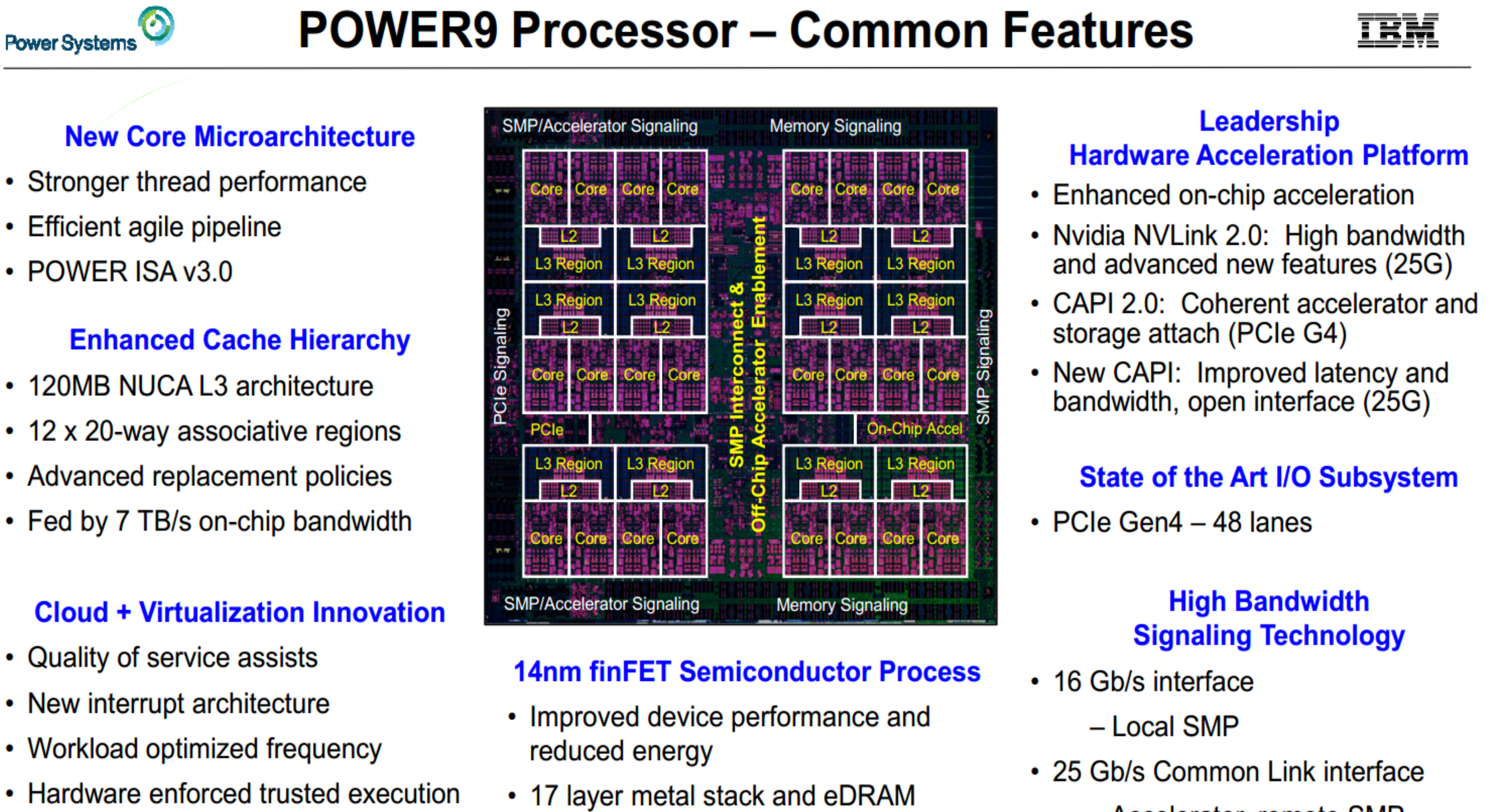
- Up to 230 GB/s sustained bandwidth

Bus Interfaces

- Durable open memory attach interface
- Integrated PCIe Gen3
- SMP Interconnect
- CAPI (Coherent Accelerator Processor Interface)

IBM Power 9 (announced in Dec. 2017)





Power Systems

POWER9 Processor – Common Features

IBM

New Core Microarchitecture

- Stronger thread performance
- Efficient agile pipeline
- POWER ISA v3.0

Enhanced Cache Hierarchy

- 120MB NUCA L3 architecture
- 12 x 20-way associative regions
- Advanced replacement policies
- Fed by 7 TB/s on-chip bandwidth

Cloud + Virtualization Innovation

- Quality of service assists
- New interrupt architecture
- Workload optimized frequency
- Hardware enforced trusted execution

SMP/Accelerator Signaling
Memory Signaling
Core
L2
L3 Region
PCIe Signaling
SMP Interconnect & Off-Chip Accelerator Enablement
SMP Signaling
PCIe
On-Chip Accel
SMP/Accelerator Signaling
Memory Signaling

14nm finFET Semiconductor Process

- Improved device performance and reduced energy
- 17 layer metal stack and eDRAM

Leadership Hardware Acceleration Platform

- Enhanced on-chip acceleration
- Nvidia NVLink 2.0: High bandwidth and advanced new features (25G)
- CAPI 2.0: Coherent accelerator and storage attach (PCIe G4)
- New CAPI: Improved latency and bandwidth, open interface (25G)

State of the Art I/O Subsystem

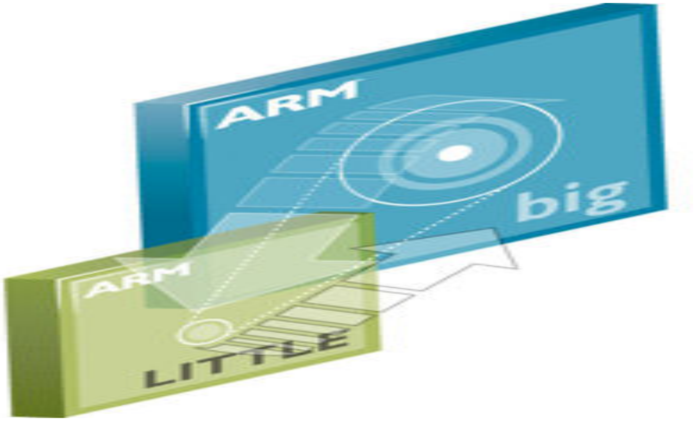
- PCIe Gen4 – 48 lanes

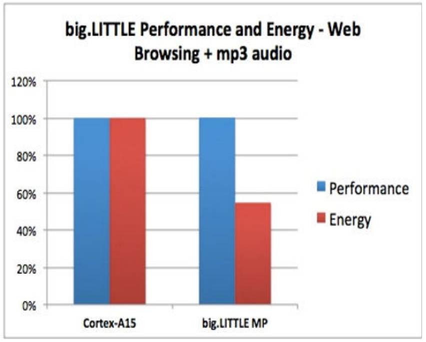
High Bandwidth Signaling Technology

- 16 Gb/s interface
  - – Local SMP
- 25 Gb/s Common Link interface
  - – Accelerator, remote SMP

ARM Big.Little Technology

- ARM big.LITTLE processing is designed to deliver the vision of the right processor for the right job. In current big.LITTLE system implementations a ‘big’ ARM Cortex™-A15 processor is paired with a ‘LITTLE’ Cortex™-A7 processor to create a system that can accomplish both high intensity and low intensity tasks in the most energy efficient manner. For example, the performance capabilities of the Cortex-A15 processor can be utilized for heavy workloads, while the Cortex-A7 can take over to process most efficiently majority of smartphone workloads. These include operating system activities, user interface and other always on, always connected tasks.





Eight-core in mobile device

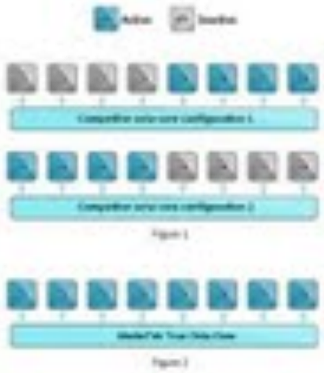
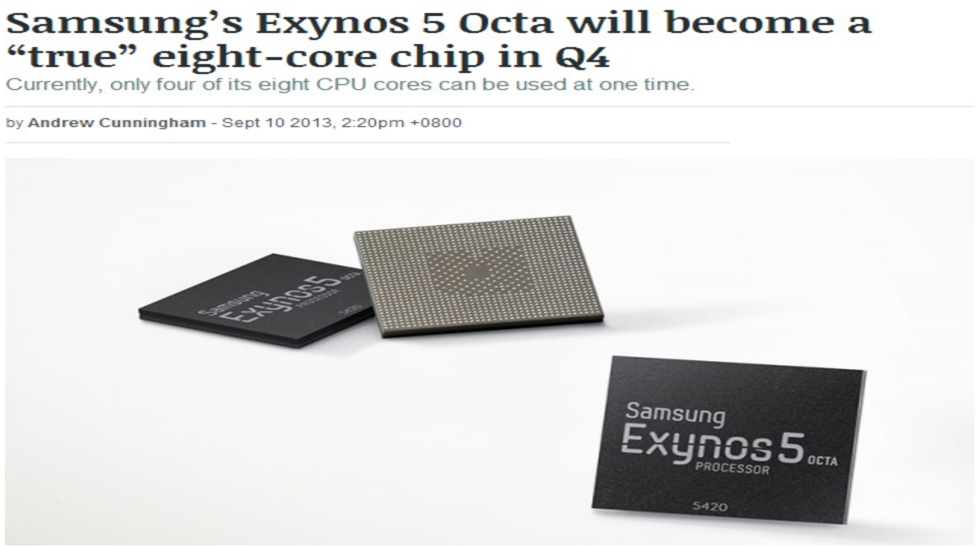

Active
Inactive
Competitor octa-core configuration 1
Competitor octa-core configuration 2
MediaTek True Octa-Core

Figure 1

Figure 2

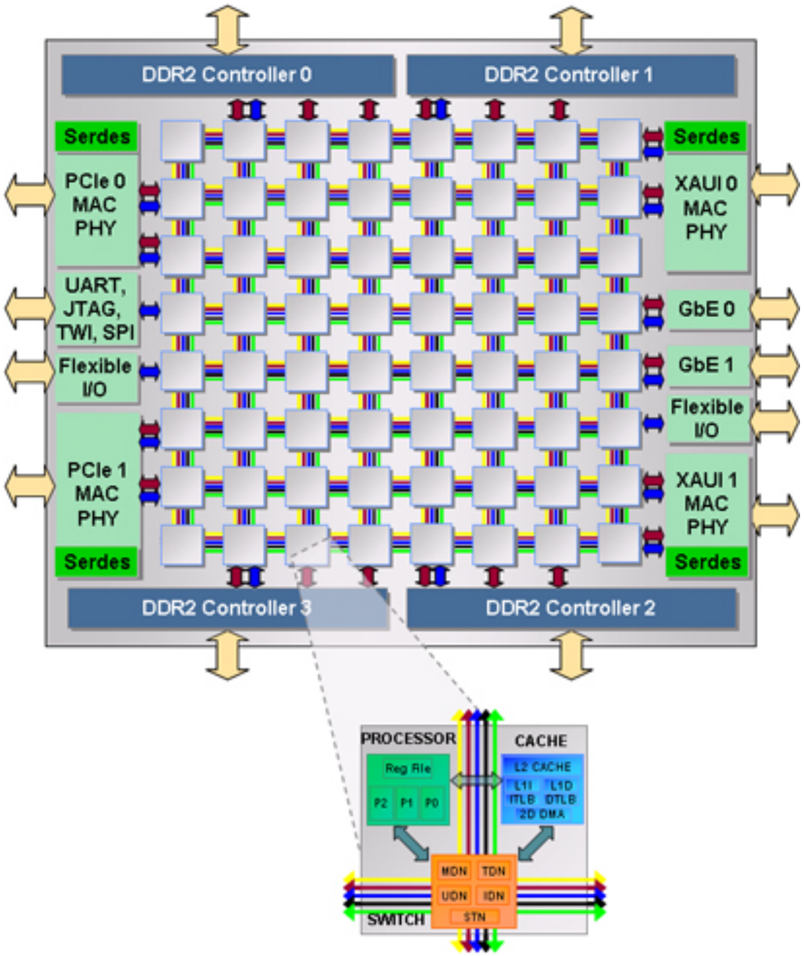


Samsung’s Exynos 5 Octa will become a “true” eight-core chip in Q4

Currently, only four of its eight CPU cores can be used at one time.

by Andrew Cunningham - Sept 10 2013, 2:20pm +0800

Tilera 64-core processor



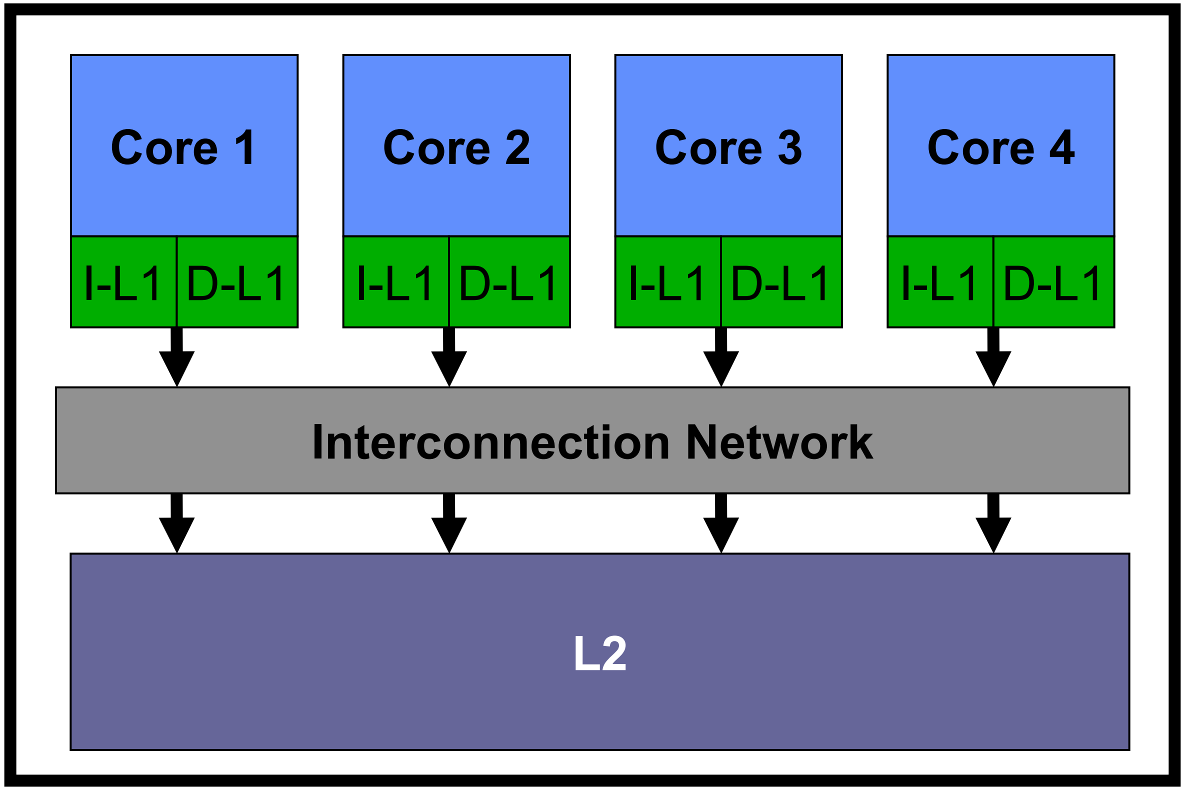
Router

“Tilera® Corporation provides multicore processors that deliver the world's highest performance computing for a broad range of applications such as networking, wireless infrastructure, digital multimedia, and cloud computing. Tilera’s processors are based on an intelligent mesh (iMesh™) architecture and provide far greater scalability than any other processor on the market.”

“The Processor is the new Transistor” [Rowen]

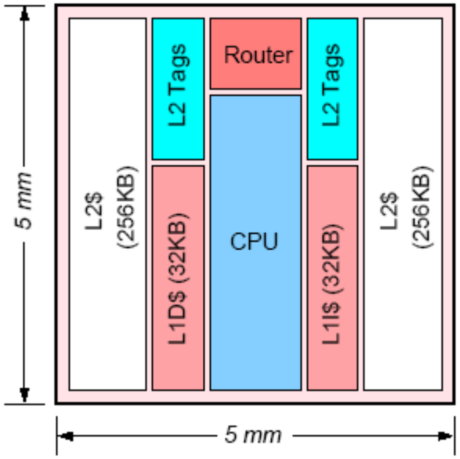
Basic CMP Architecture

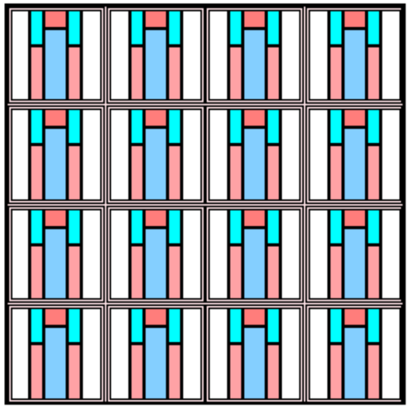
- **Shared last level cache**



Scalable CMP Architecture

- **Tiled CMP**
  - **Each tile includes processor, L1, L2, and router**
  - **Physically distributed last level cache**



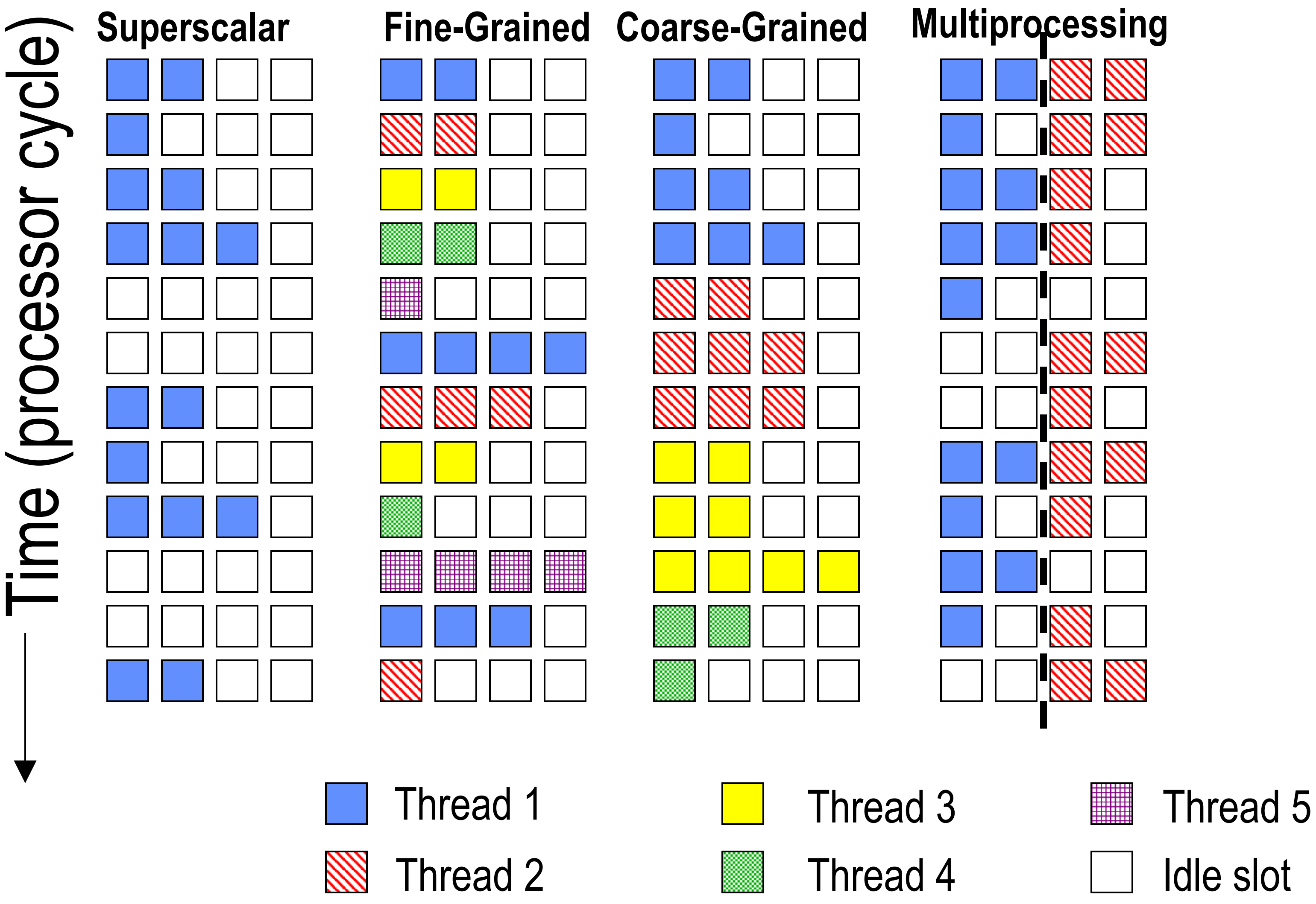


A 4 X 4 CMP (20 mm X 20 mm)

Multithreading

- Performing multiple threads of execution in parallel
  - Replicate registers, PC, etc.
  - Fast switching between threads
- Fine-grain multithreading
  - Switch threads after each cycle
  - Interleave instruction execution
  - If one thread stalls, others are executed
- Coarse-grain multithreading
  - Only switch on long stall (e.g., L2-cache miss)
  - Simplifies hardware, but doesn’t hide short stalls (eg, data hazards)

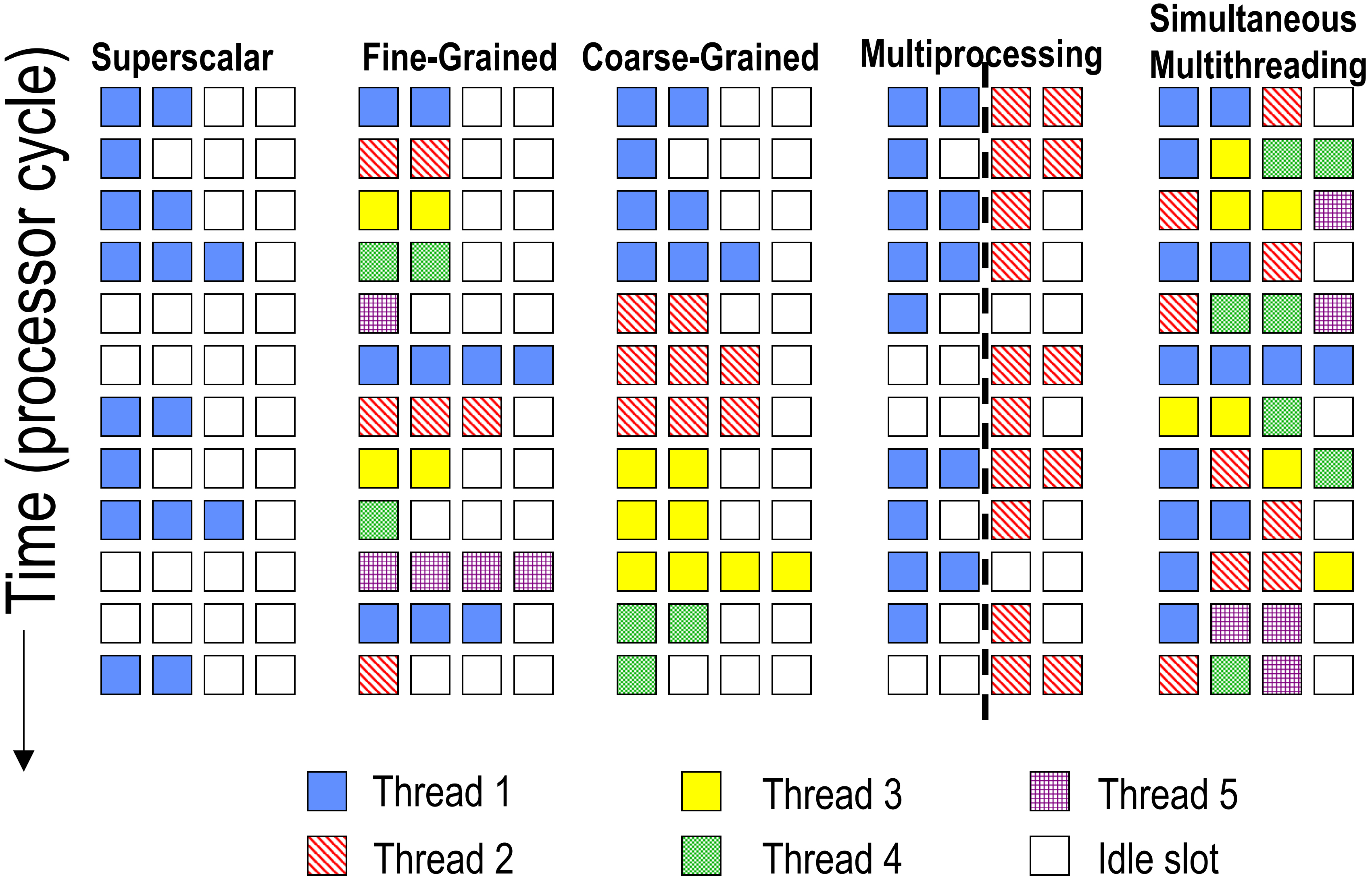
Multithreaded Categories



Simultaneous Multithreading

- In multiple-issue dynamically scheduled processor
  - Schedule instructions from multiple threads
  - Instructions from independent threads execute when function units are available
  - Within threads, dependencies handled by scheduling and register renaming
- Example: Intel Pentium-4 HT
  - Two threads: duplicated registers, shared function units and caches

Multithreaded Categories


Superscalar
Fine-Grained
Coarse-Grained
Multiprocessing
Simultaneous Multithreading
Time (processor cycle)
Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Idle slot