GPU Programming

Dr. Farshad Khunjush

Department of Computer Science and Engineering
Shiraz University

Fall 2021

From Pipeline to Multi-core architectures

Dr. Farshad Khunjush

Some slides come from
Professor Saman Amarasinghe (MIT)
@ http://groups.csail.mit.edu/cag/ps3/
And Professor Arvind and Joe Elmer (MIT)
@ http://csg.csail.mit.edu/6.823/lecnotes.html

Outline

- Implicit Parallelism: Superscalar Processors
- Explicit Parallelism
- Shared Instruction Processors
- Shared Sequencer Processors
- Shared Memory Processors
- Multi-threaded Processors
- Multi-core Processors

Implicit Parallelism: Superscalar Processors

- Issue varying numbers of instructions per clock
  - statically scheduled
    - using compiler techniques
    - in-order execution
  - dynamically scheduled
    - Extracting ILP by examining 100’s of instructions
    - Scheduling them in parallel as operands become available
    - Rename registers to eliminate anti dependences
    - out-of-order (OOO) cexecution
    - Speculative execution

Pipeline Execution:

IF: Instruction fetch
ID : Instruction decode
EX : Execution
WB : Write back

Cycles

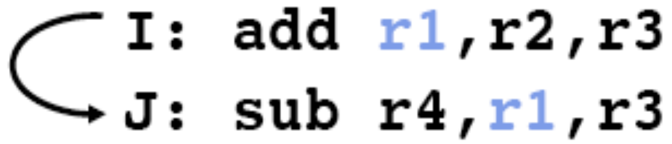
Instruction #	1	2	3	4	5	6	7	8
Instruction i	IF	ID	EX	WB				
Instruction i+1		IF	ID	EX	WB			
Instruction i+2			IF	ID	EX	WB		
Instruction i+3				IF	ID	EX	WB	
Instruction i+4					IF	ID	EX	WB

Superscalar Execution:

	Cycles						
Instruction type	1	2	3	4	5	6	7
Integer	IF	ID	EX	WB			
Floating point	IF	ID	EX	WB			
Integer		IF	ID	EX	WB		
Floating point		IF	ID	EX	WB		
Integer			IF	ID	EX	WB	
Floating point			IF	ID	EX	WB	
Integer				IF	ID	EX	WB
Floating point				IF	ID	EX	WB

Data Dependency & Hazards

- Instr J is data dependent (aka true dependence) on Instr I



```
I: add r1,r2,r3
J: sub r4,r1,r3
```

- If two instructions are data dependent, they cannot execute simultaneously, be completely overlapped or execute in out-of order
- If data dependence caused a hazard in pipeline, called a Read After Write (RAW) hazard

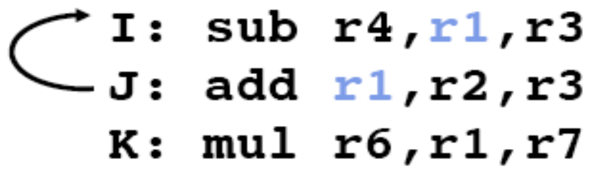
Data Dependencies, Hazards, and ILP

- HW/SW must preserve program order: order instructions would execute sequentially as determined by original source program
  - Dependences are a property of programs
- Importance of the data dependencies
  - 1) indicates the possibility of a hazard
  - 2) determines order in which results must be calculated
- Goal: exploit parallelism by preserving program order only where it affects the outcome of the program

Name Dependency #1: Anti-dependency

- Name dependency: when 2 instructions use same register or memory location, called a name, but no flow of data between the instructions associated with that name; 2 versions of name dependence

- Instr J writes operand *before* Instr I reads it



```
I: sub r4,r1,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

Called an "anti-dependence" by compiler writers.
This results from reuse of the name "r1"

- If anti-dependence caused a hazard in the pipeline, called a Write After Read (WAR) hazard

Name Dependency #2: Output dependency

- Instr J writes operand *before* Instr I writes it.

```
I: sub r1,r4,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

- Called an "output dependence" by compiler writers. This also results from the reuse of name "r1"
- If anti-dependence caused a hazard in the pipeline, called a Write After Write (WAW) hazard

- Instructions involved in a name dependence can execute simultaneously if name used in instructions is changed so instructions do not conflict
  - Register renaming resolves name dependence for registers
  - Renaming can be done either by compiler or by HW

Control Dependencies

- Every instruction is control dependent on some set of branches, and, in general, these control dependencies must be preserved to preserve program order

```
if p1 {
        S1;
};
if p2 {
        S2;
}
```

- S1 is control dependent on p1, and S2 is control dependent on p2 but not on p1.
- Speculative Execution

Speculation

- Greater ILP: Overcome control dependence by hardware speculating on outcome of branches and executing program as if guesses were correct
  - Speculation ⇒ fetch, issue, and execute instructions as if branch predictions were always correct
  - Dynamic scheduling ⇒ only fetches and issues instructions
- Essentially a data flow execution model: Operations execute as soon as their operands are available

Speculation in Modern Superscalars

- Different predictors
  - Branch Prediction
  - Value Prediction
  - Prefetching (memory access pattern prediction)
- Inefficient
  - Predictions can go wrong
  - Has to flush out wrongly predicted data
  - While not impacting performance, it consumes power

Pentium IV

- Pipelined
  - minimum of 11 stages for any instruction
- Instruction-Level Parallelism
  - Can execute up to 3 x86 instructions per cycle
- Data Parallel Instructions
  - MMX (MultiMedia eXtension) (64-bit) and SSE (Streaming SIMD Extensions) (128-bit) extensions
  - provide short vector support
- Thread-Level Parallelism at System Level
  - Bus architecture supports shared memory multiprocessing

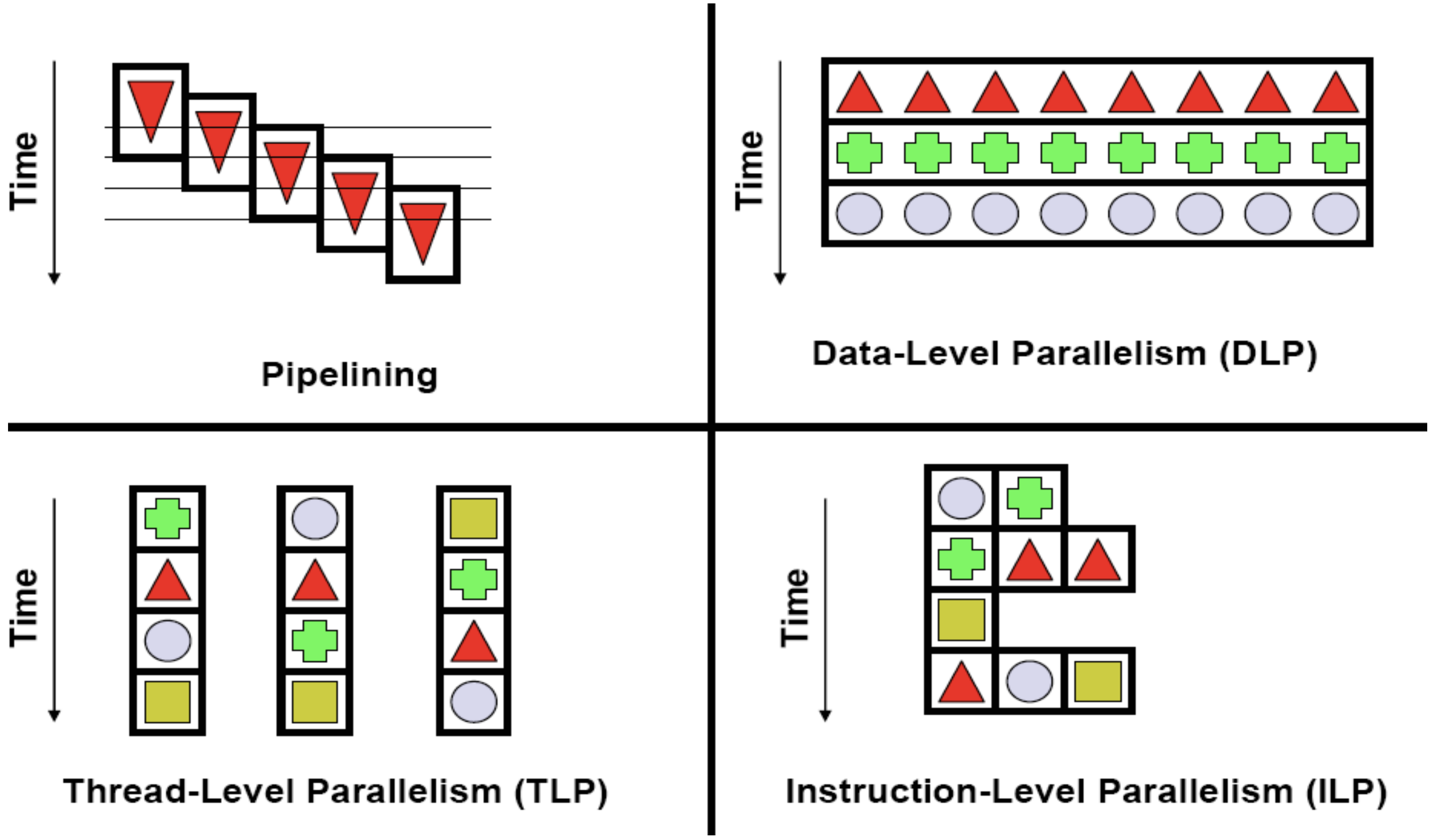
Outline

- Implicit Parallelism: Superscalar Processors
- **Explicit Parallelism**
- Shared Instruction Processors
- Shared Sequencer Processors
- Shared Memory Processors
- Multi-threaded Processors
- Multicore Processors

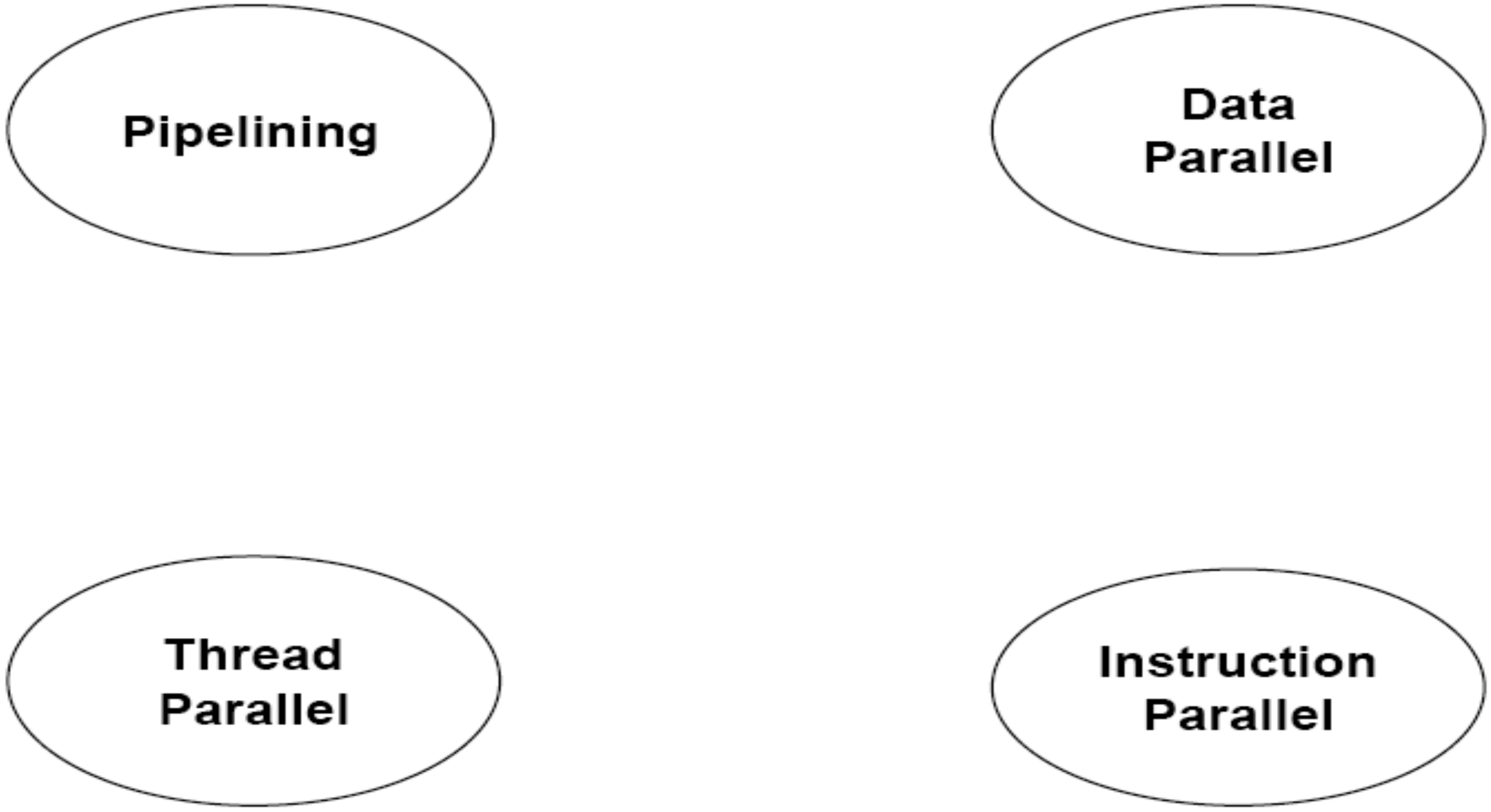
Explicit Parallel Processors

- Parallelism is exposed to software
  - Compiler or Programmer
- Many different forms
  - Loosely coupled Multiprocessors to tightly coupled VLIW

Types of Parallelism


Time
Pipelining
Time
Data-Level Parallelism (DLP)
Time
Thread-Level Parallelism (TLP)
Time
Instruction-Level Parallelism (ILP)

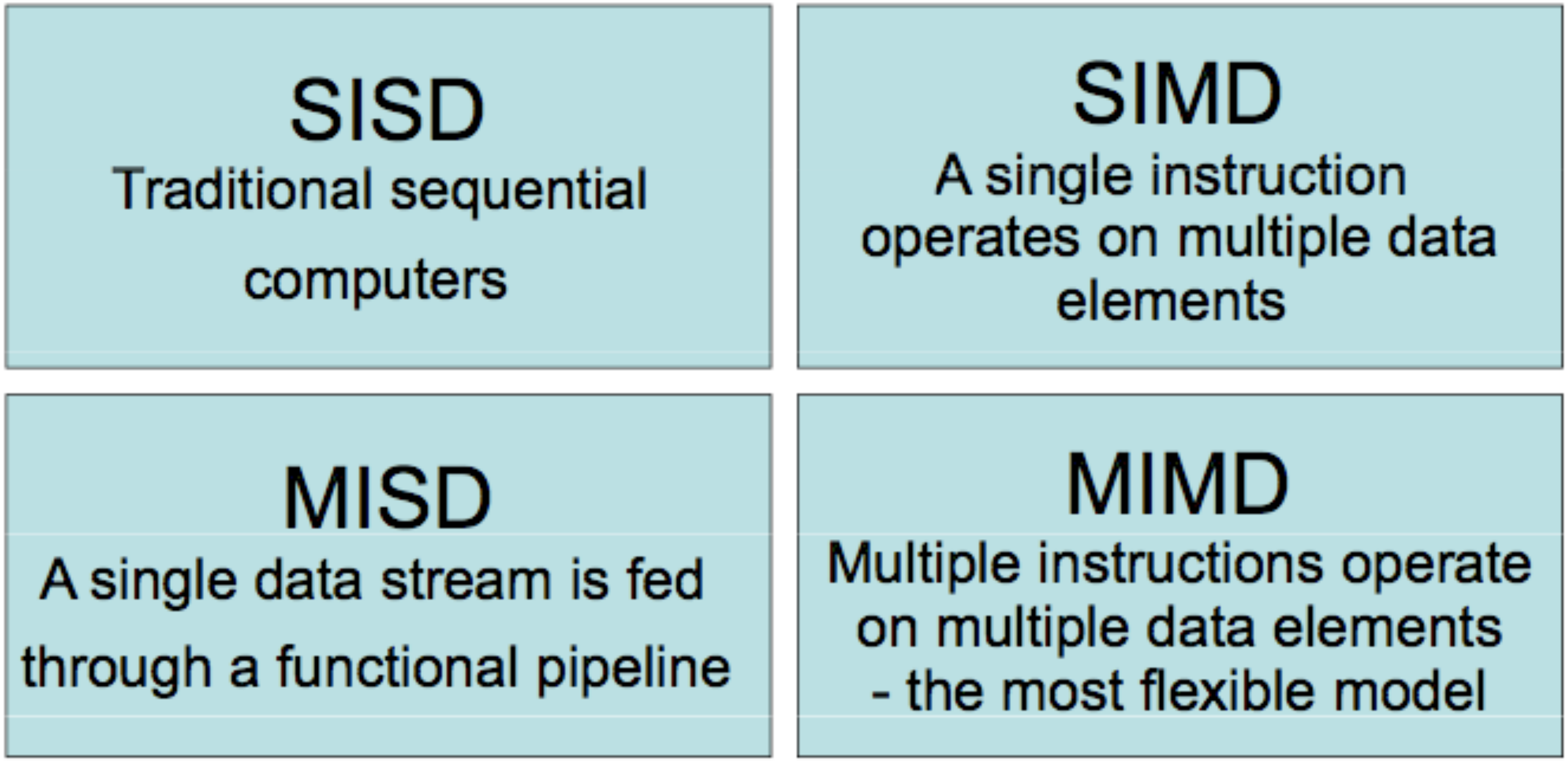
Translating Parallelism Types



Issues in Parallel Machine Design

- Communication
  - how do parallel operations communicate data results?
- Synchronization
  - how are parallel operations coordinated?
- Resource Management
  - how are a large number of parallel tasks scheduled onto finite hardware?
- Scalability
  - how large a machine can be built?

Flynn's Classification (1966)



SISD Traditional sequential computers	**SIMD** A single instruction operates on multiple data elements
MISD A single data stream is fed through a functional pipeline	**MIMD** Multiple instructions operate on multiple data elements - the most flexible model

Outline

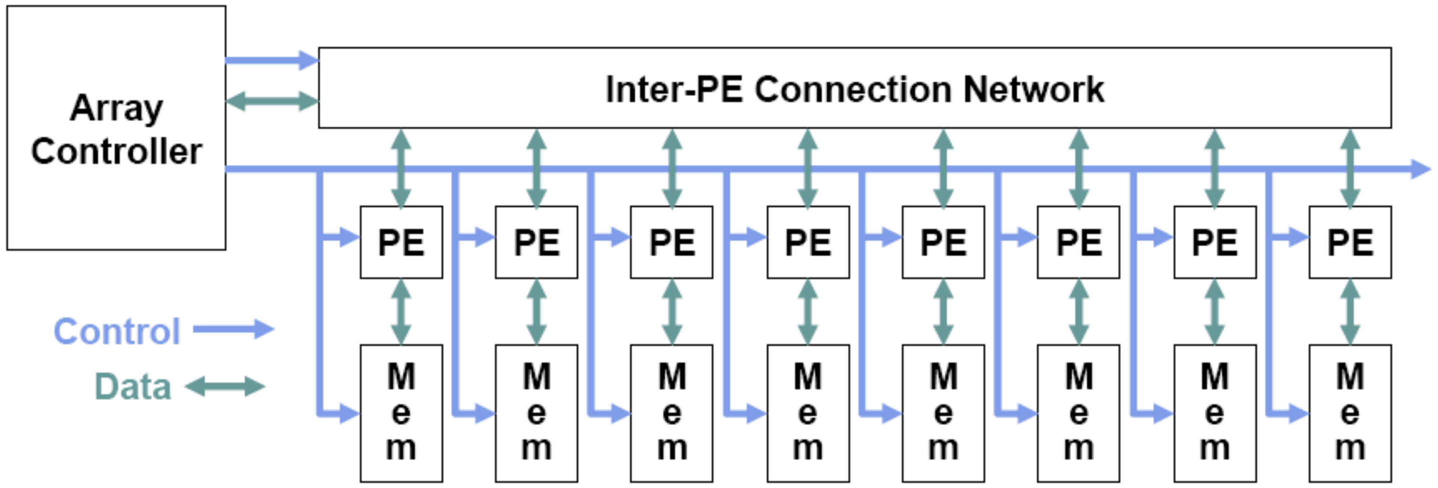
- Implicit Parallelism: Superscalar Processors
- Explicit Parallelism
- **Shared Instruction Processors**
- Shared Sequencer Processors
- Shared Memory Processors
- Multi-Threaded Processors
- Multicore Processors

Shared Instruction: SIMD Machines

- Illiac IV (1972)
  - 64 64-bit PEs, 16KB/PE, 2D network
- Goodyear STARAN (1972)
  - 256 bit-serial associative PEs, 32B/PE, multistage network
- ICL DAP (Distributed Array Processor) (1980)
  - 4K bit-serial PEs, 512B/PE, 2D network
- Goodyear MPP (Massively Parallel Processor) (1982)
  - 16K bit-serial PEs, 128B/PE, 2D network
- Thinking Machines Connection Machine CM-1 (1985)
  - 64K bit-serial PEs, 512B/PE, 2D + hypercube router
  - CM-2: 2048B/PE, plus 2,048 32-bit floating-point units
- Maspar MP-1 (1989)
  - 16K 4-bit processors, 16-64KB/PE, 2D + Xnet router
  - MP-2: 16K 32-bit processors, 64KB/PE

Shared Instruction: SIMD Architecture

- Central controller broadcasts instructions to multiple processing elements (PEs)

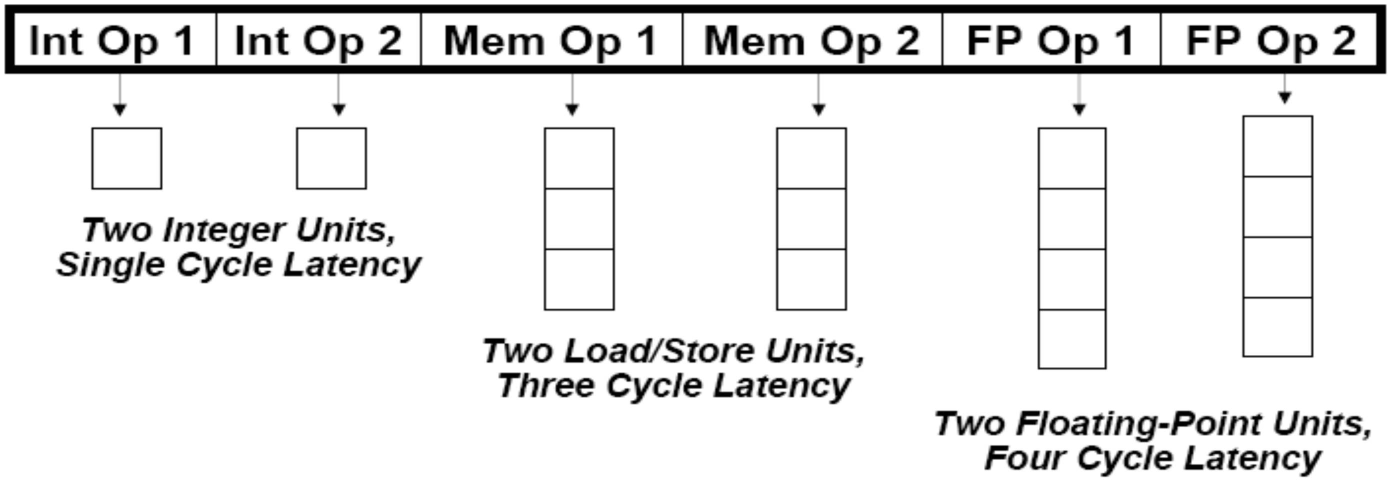


- **Only requires one controller for whole array**
- **Only requires storage for one copy of program**
- **All computations fully synchronized**

Outline

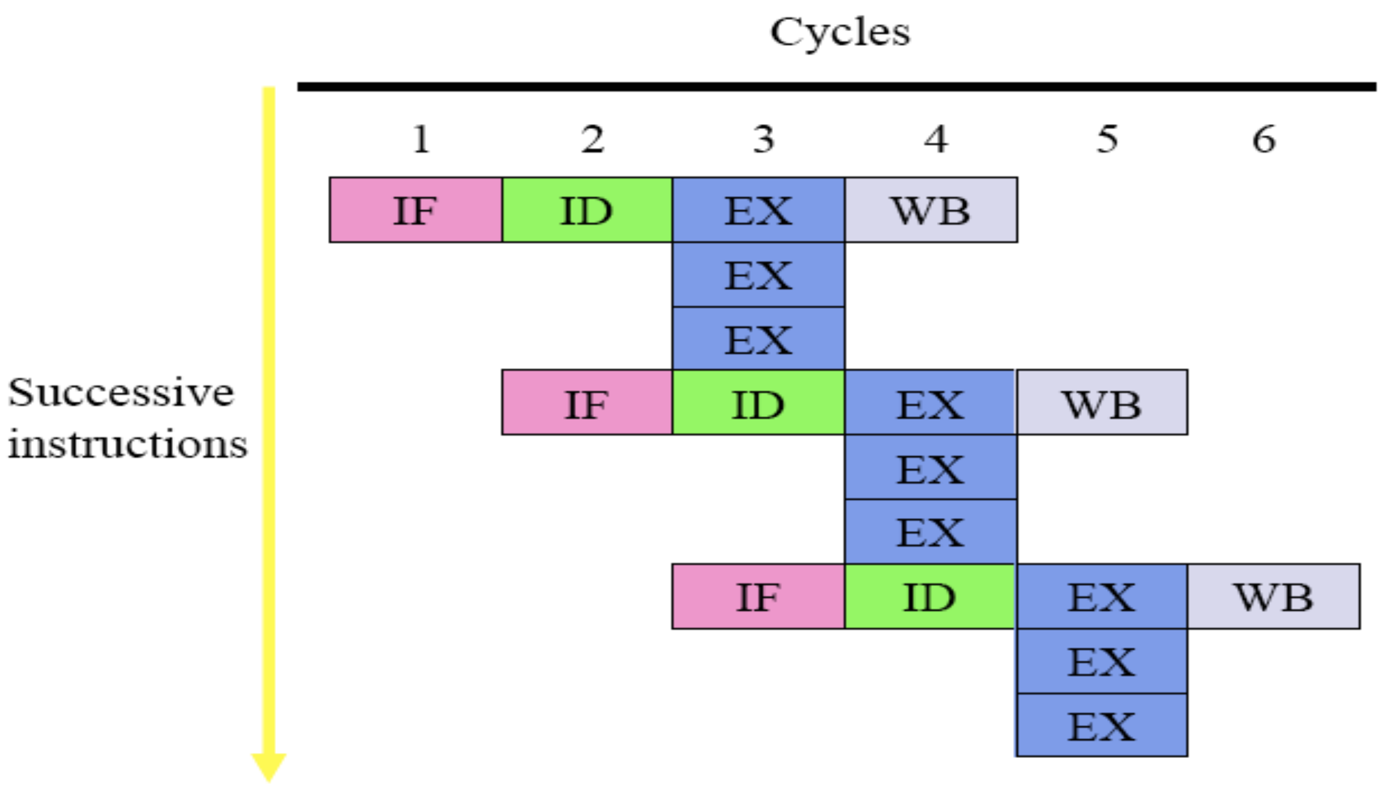
- Implicit Parallelism: Superscalar Processors
- Explicit Parallelism
- Shared Instruction Processors
- **Shared Sequencer Processors**
- Shared Memory Processors
- Multi-threaded Processors
- Multicore Processors

Shared Sequencer
VLIW: Very Long Instruction Word


Int Op 1
Int Op 2
Mem Op 1
Mem Op 2
FP Op 1
FP Op 2
Two Integer Units, Single Cycle Latency
Two Load/Store Units, Three Cycle Latency
Two Floating-Point Units, Four Cycle Latency

- Compiler schedules parallel execution (Static Scheduling)
- Multiple parallel operations packed into one long instruction word
- Compiler must avoid data hazards

VLIW Instruction Execution



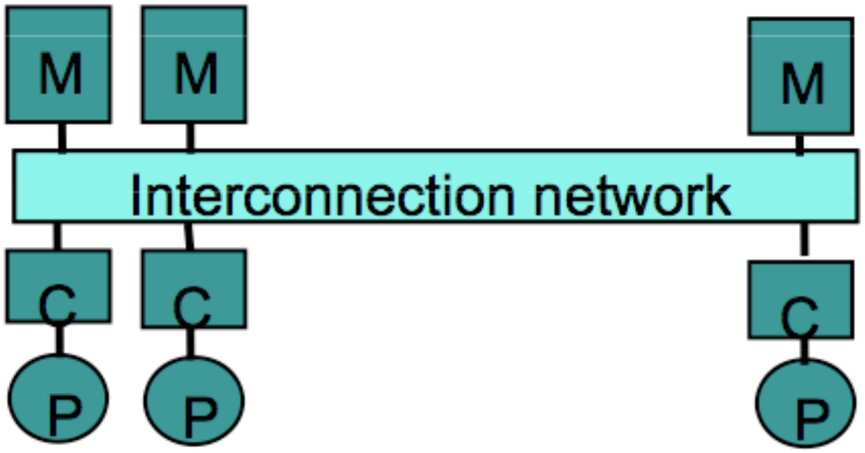
VLIW execution with degree = 3

Outline

- Implicit Parallelism: Superscalar Processors
- Explicit Parallelism
- Shared Instruction Processors
- Shared Sequencer Processors
- **Shared Memory Processors**
- Multi-Threade Processors
- Multicore Processors

Shared Memory Multiprocessors

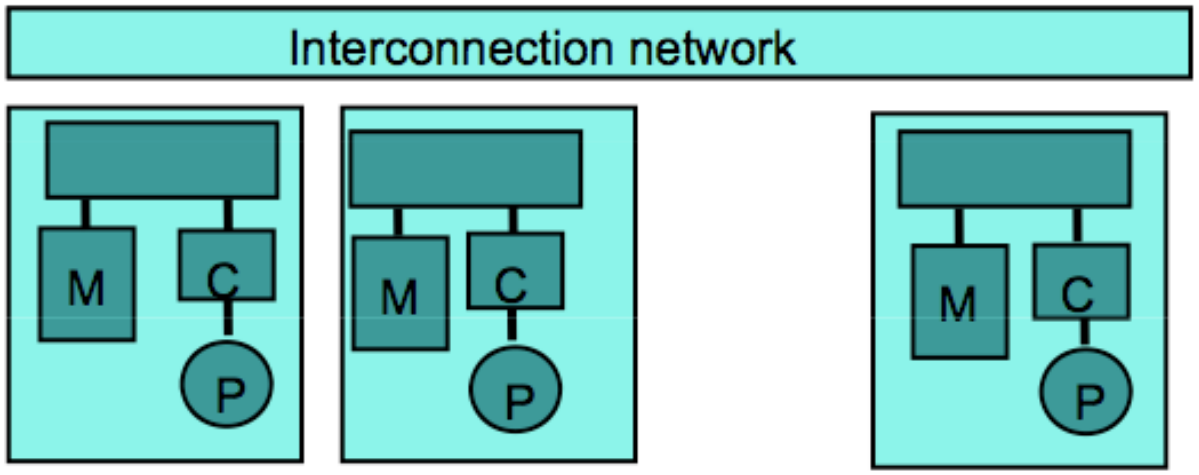
- Symmetric Shared-Memory Multiprocessor



- Load and store instructions used to communicate data between processes
  - no OS involvement
  - low software overhead
- Usually some special synchronization primitives
  - fetch&op
  - load linked/store conditional

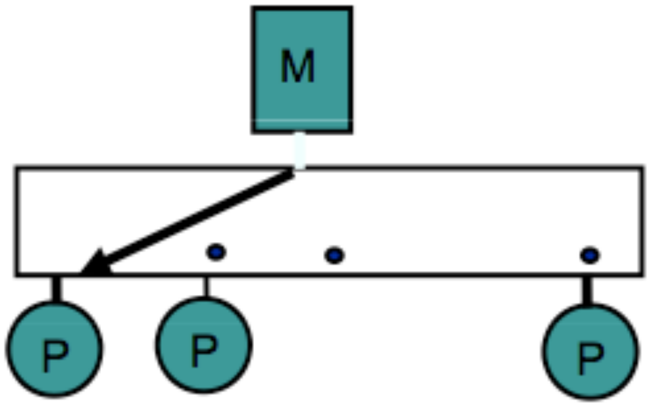
Shared Memory Multiprocessors

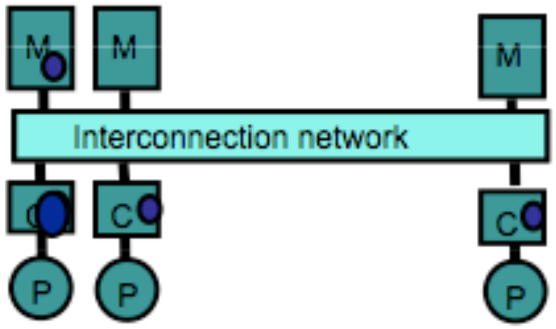
- In large scale systems, the logically shared memory is implemented as physically distributed memory modules
- Two main categories
  - non cache coherent
  - hardware cache coherent



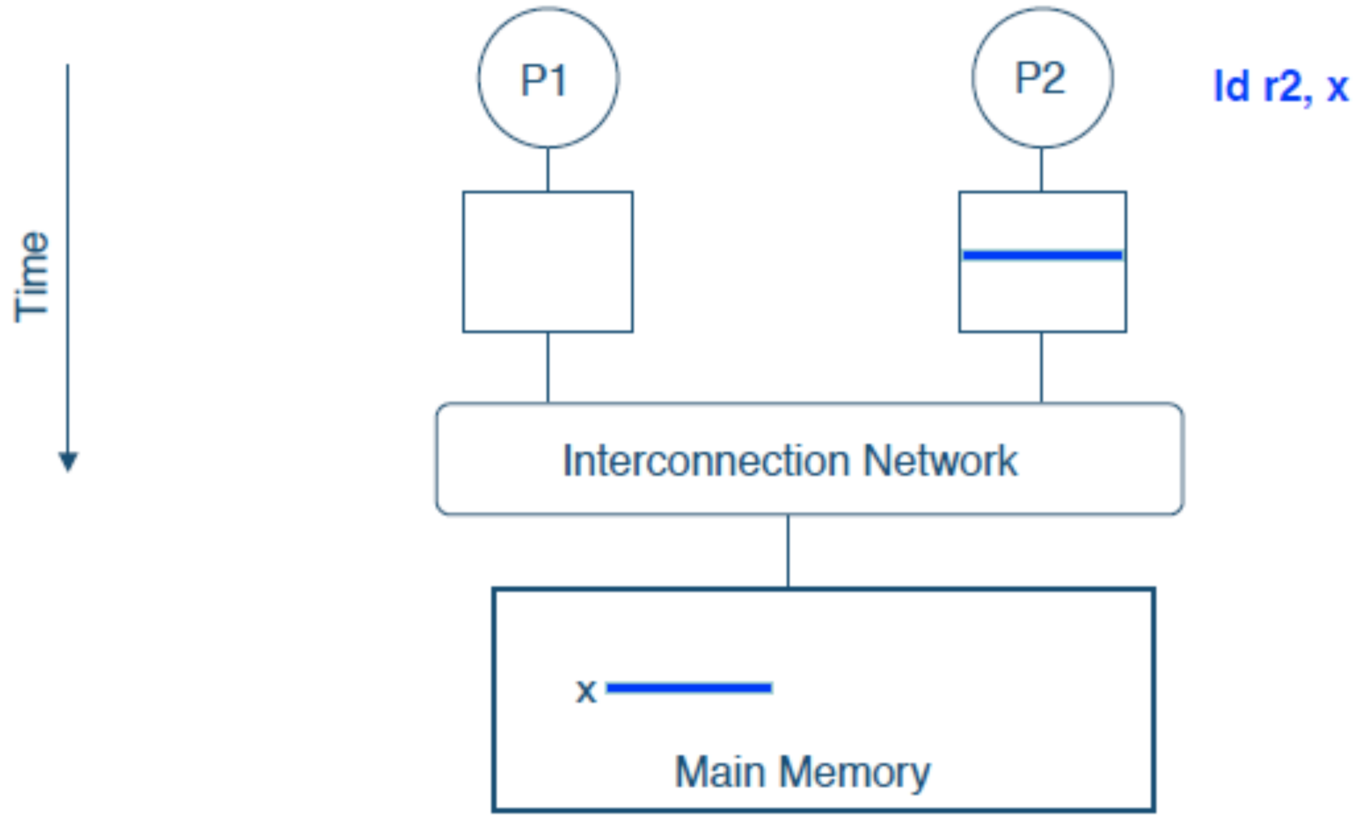
Shared Memory Multiprocessors

- The Cache Coherence problem
  - Definition (memory coherence): A read shall return the value of the latest write as defined by the partial order of memory operations in a valid execution.

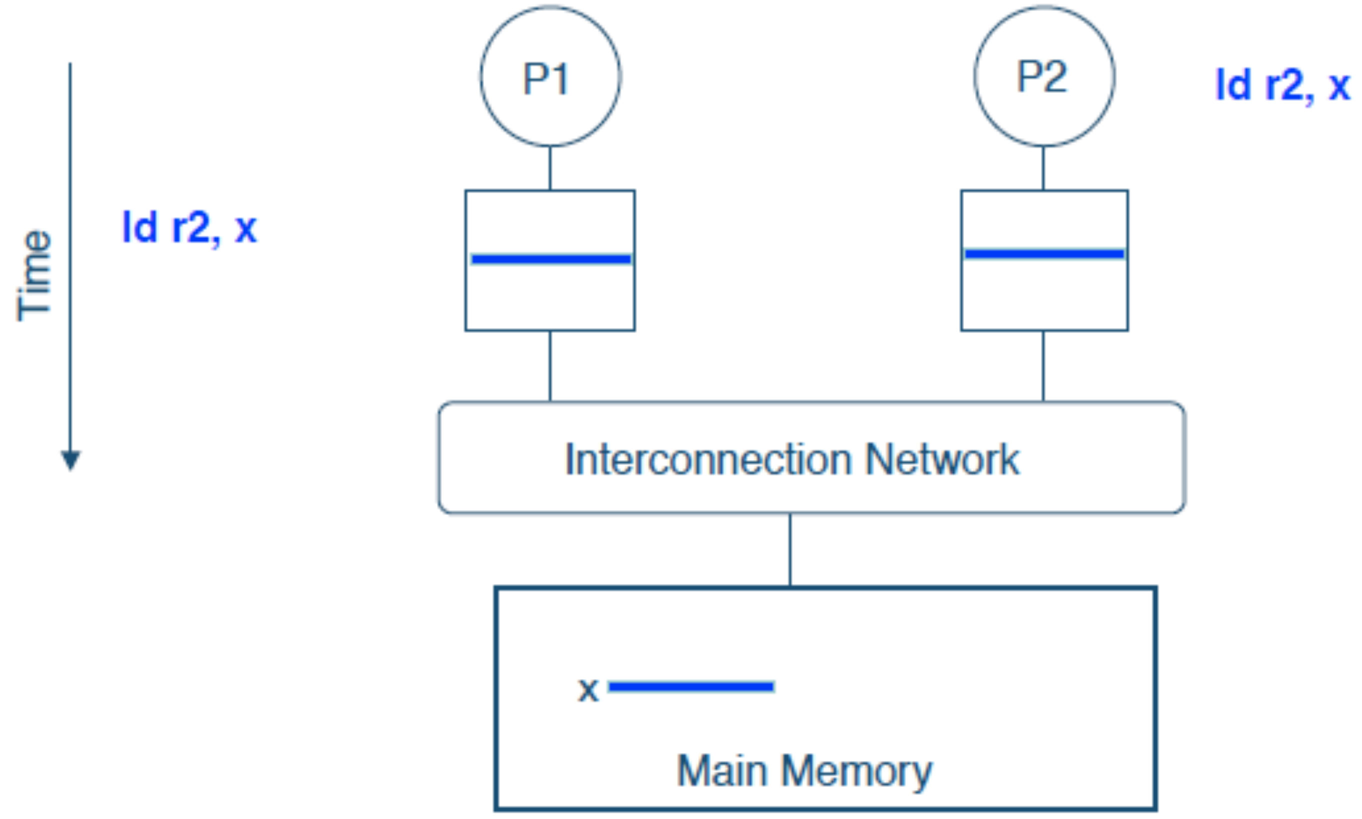




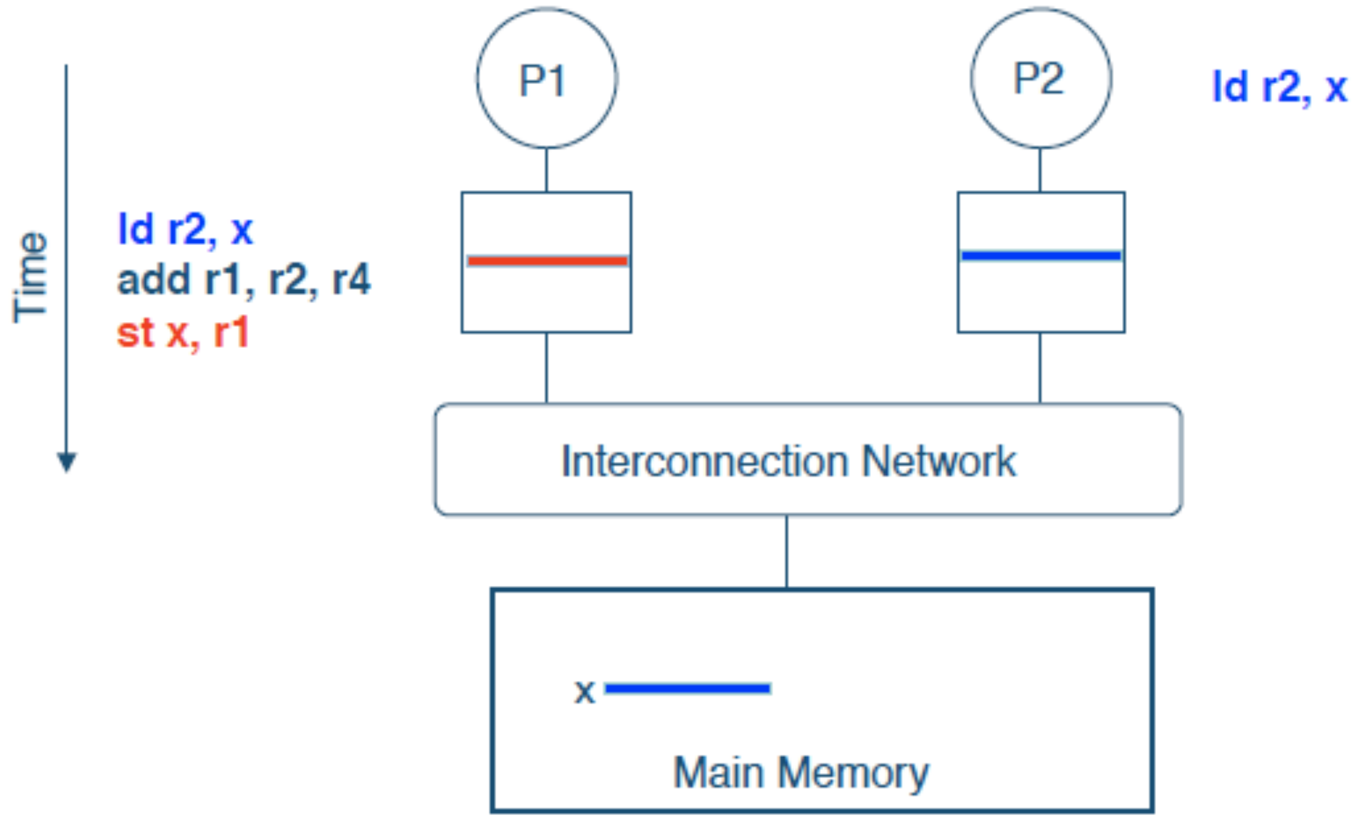
Cache Coherency Example



Cache Coherency Example



Cache Coherency Example

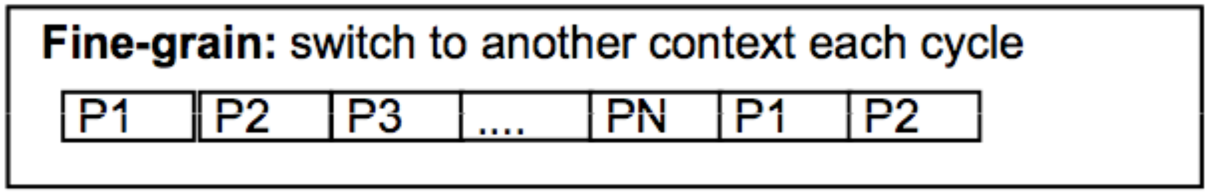


Shared Memory Multiprocessors

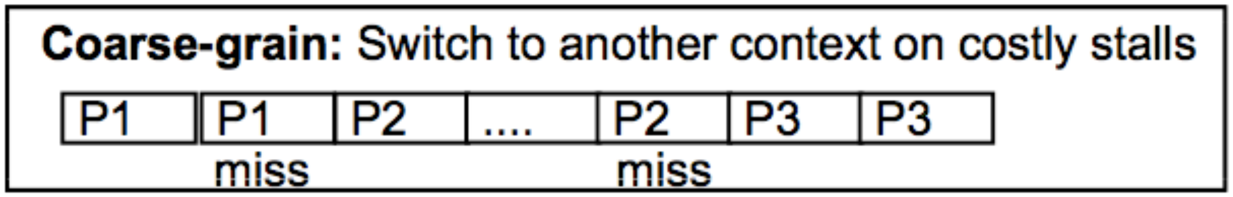
- No hardware cache coherence
  - IBM RP3
  - BBN Butterfly
  - Cray T3D/T3E
  - Parallel vector supercomputers (Cray T90, NEC SX-5)
- Hardware cache coherence
  - many small-scale SMPs (e.g. Quad Pentium Xeon systems)
  - large scale bus/crossbar-based SMPs (Sun Starfire)
  - large scale directory-based SMPs (SGI Origin)

Multi-Threading

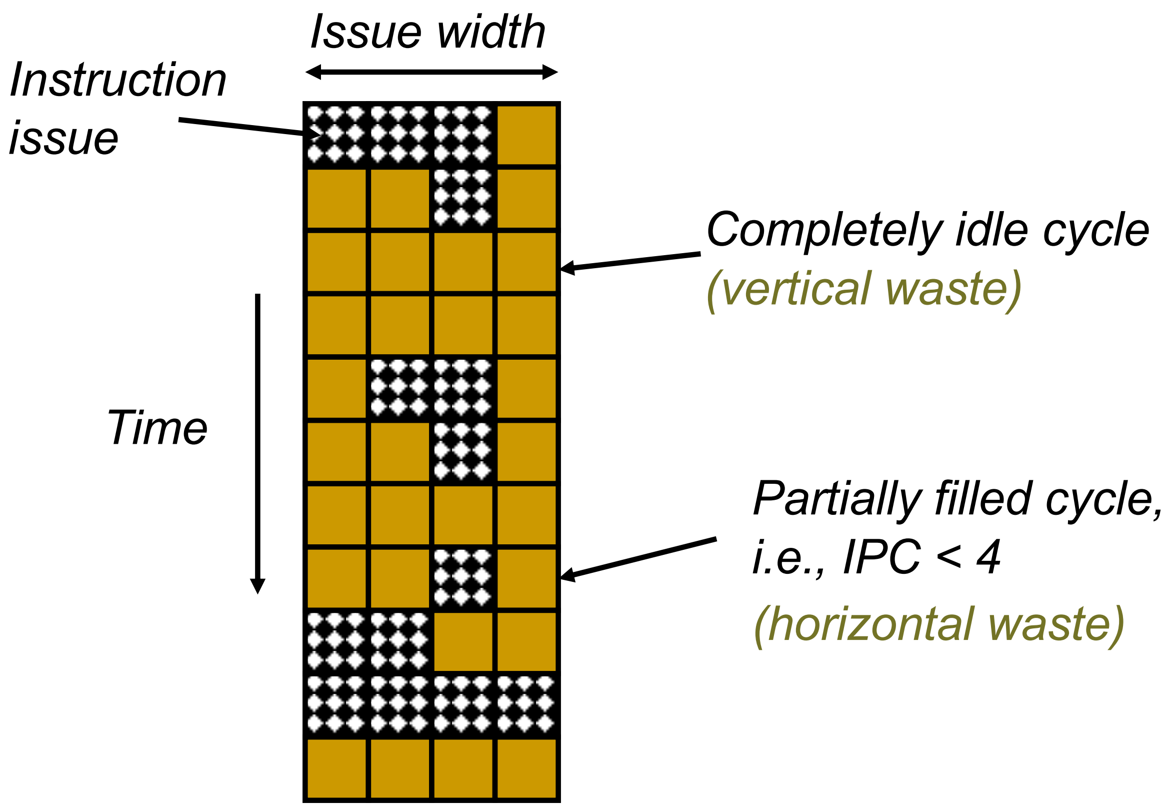
- Increase resource utilization by multiplexing the execution of multiple threads on the same pipeline
- Two Approaches:
  - Fine grain



  - Coarse grain

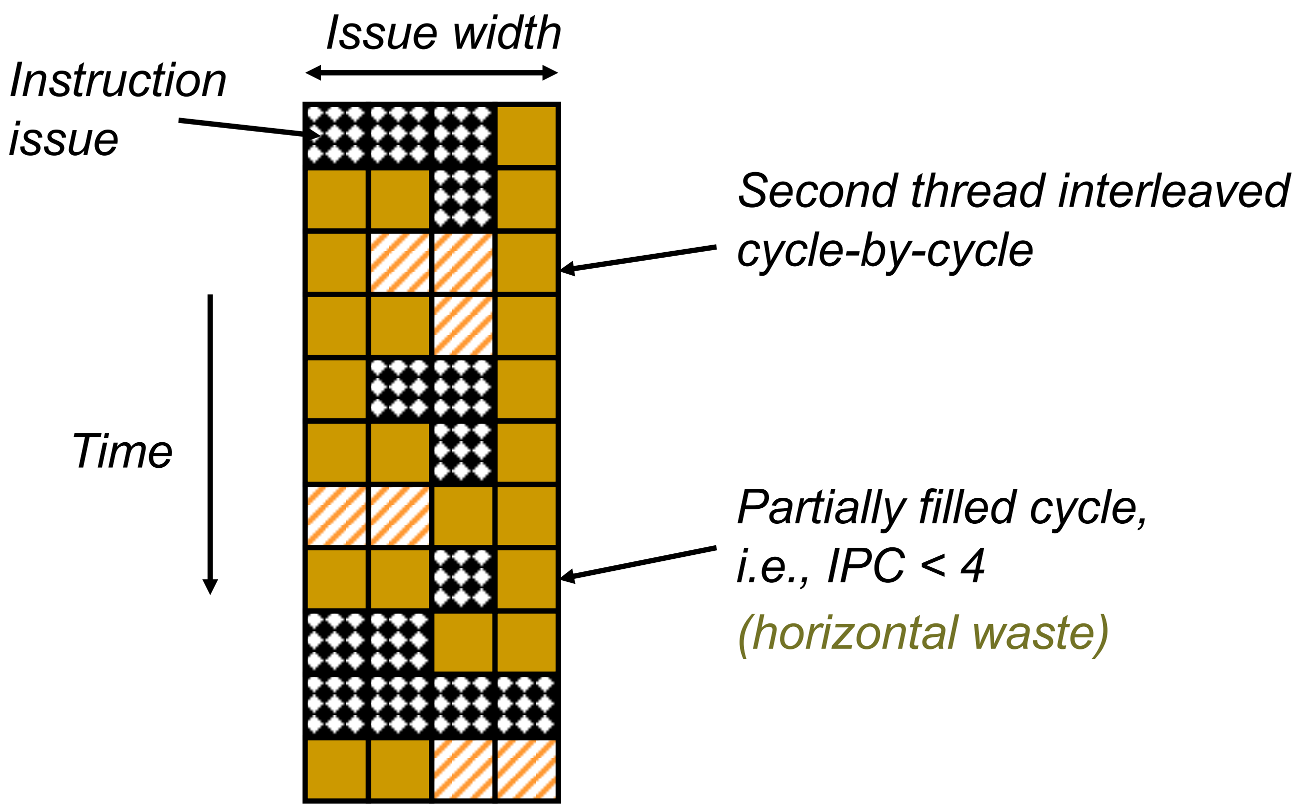

Coarse-grain: Switch to another context on costly stalls
P1
P1
P2
....
P2
P3
P3
miss
miss

Superscalar Machine Efficiency



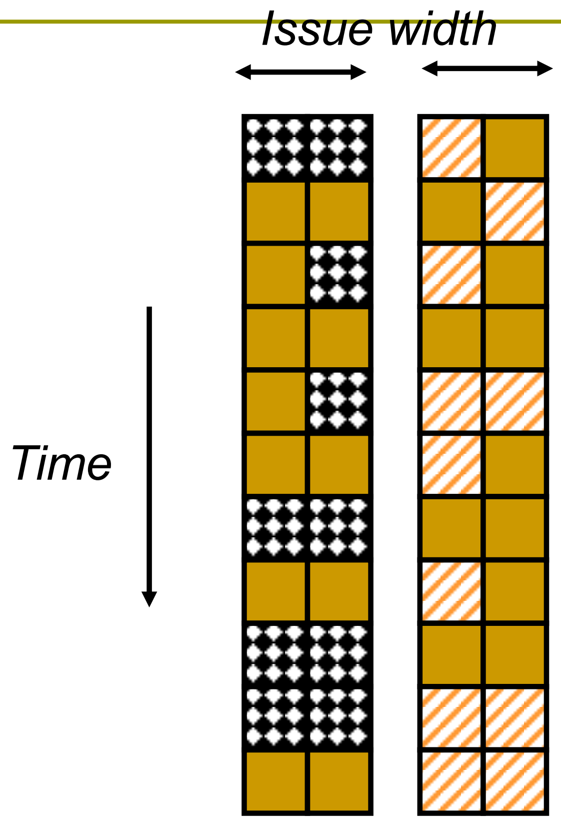
- *Why horizontal waste?*
- *Why vertical waste?*

Vertical Multithreading


Issue width
Instruction issue
Second thread interleaved cycle-by-cycle
Time
Partially filled cycle, i.e., IPC < 4
(horizontal waste)

- What is the effect of cycle-by-cycle interleaving?
  - removes vertical waste, but leaves some horizontal waste

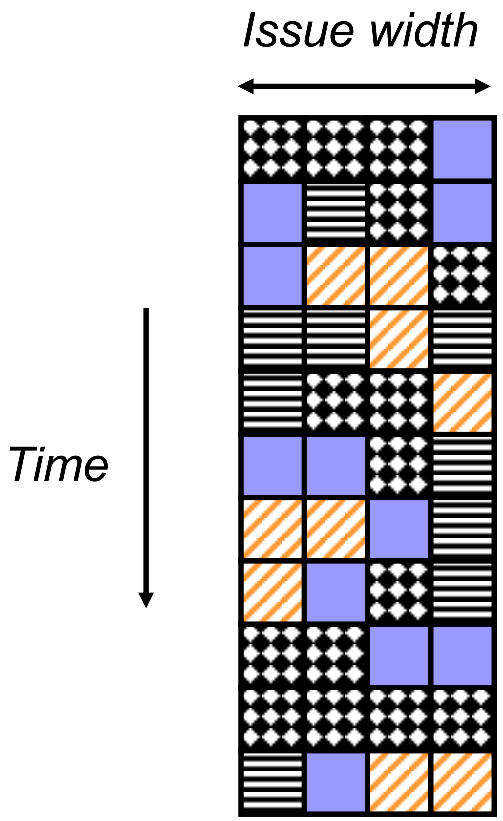
Chip Multiprocessing



- What is the effect of splitting into multiple processors?
  - eliminates horizontal waste,
  - leaves some vertical waste, and
  - caps peak throughput of each thread.

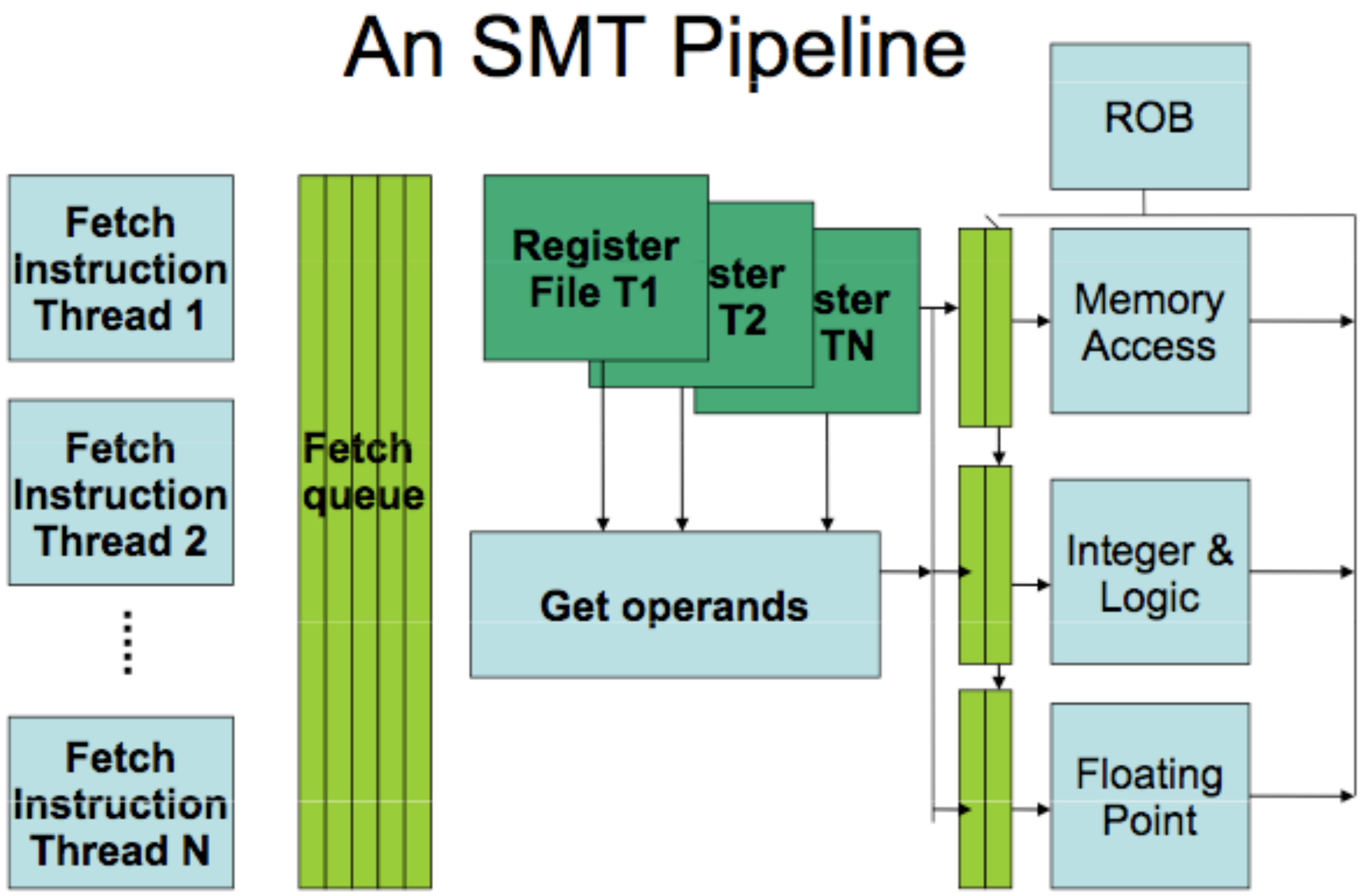
Ideal Superscalar Multithreading

[Tullsen, Eggers, Levy, UW, 1995]

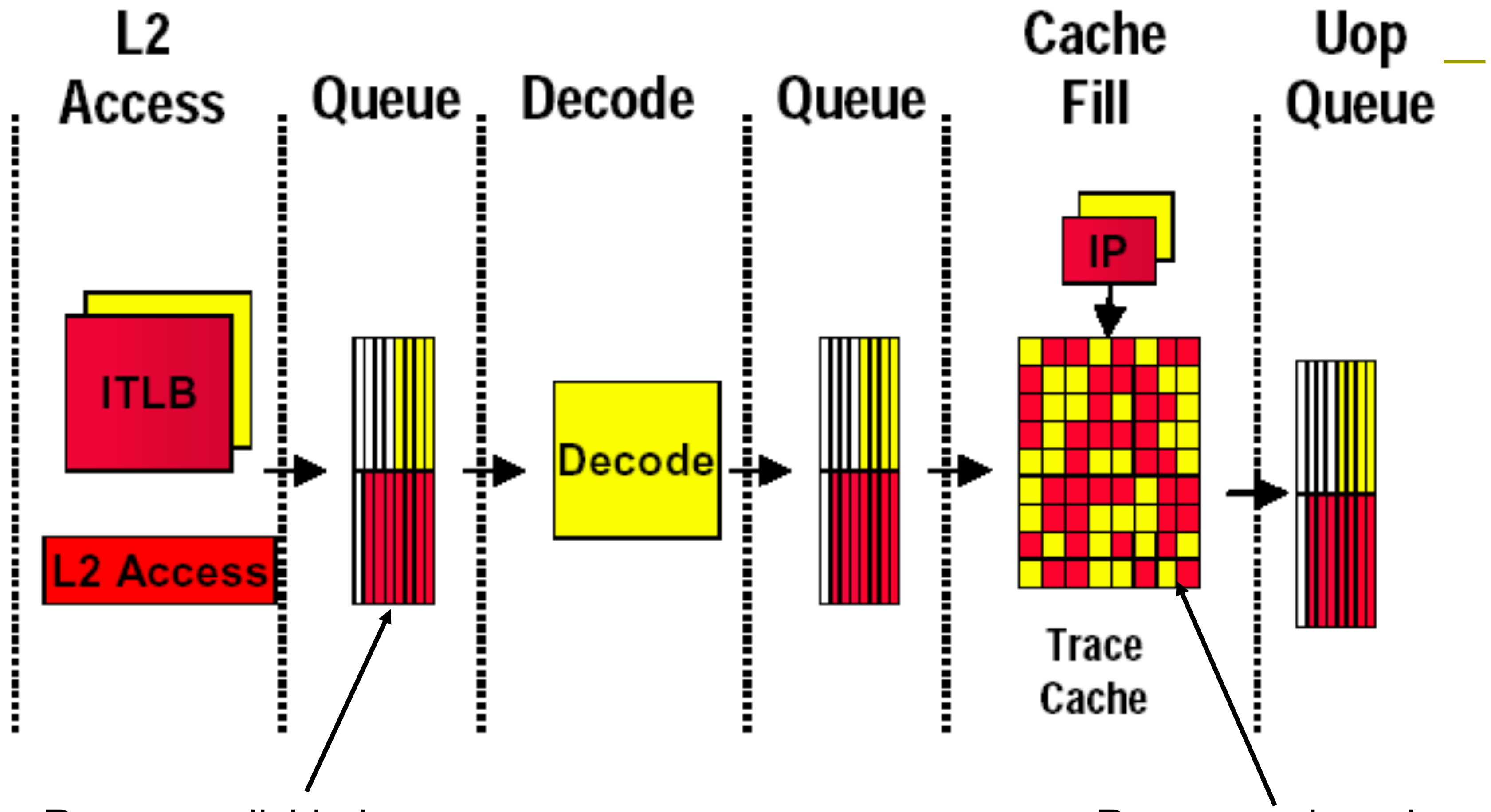


- Interleave multiple threads to multiple issue slots with no restrictions

A Typical Multi-threaded Architecture



Pentium-4 Hyper-threading (*Front End*)

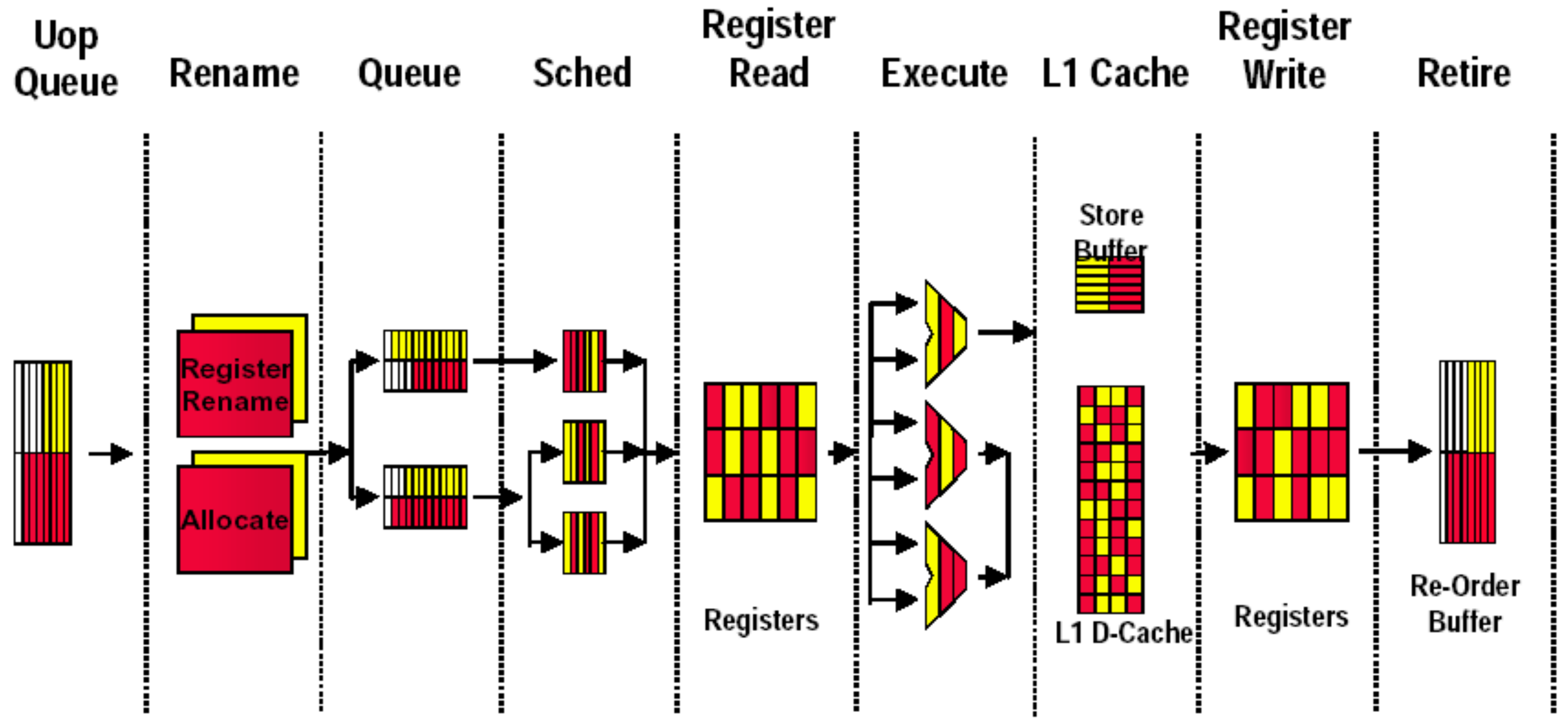


Resource divided between logical CPUs

Resource shared between logical CPUs

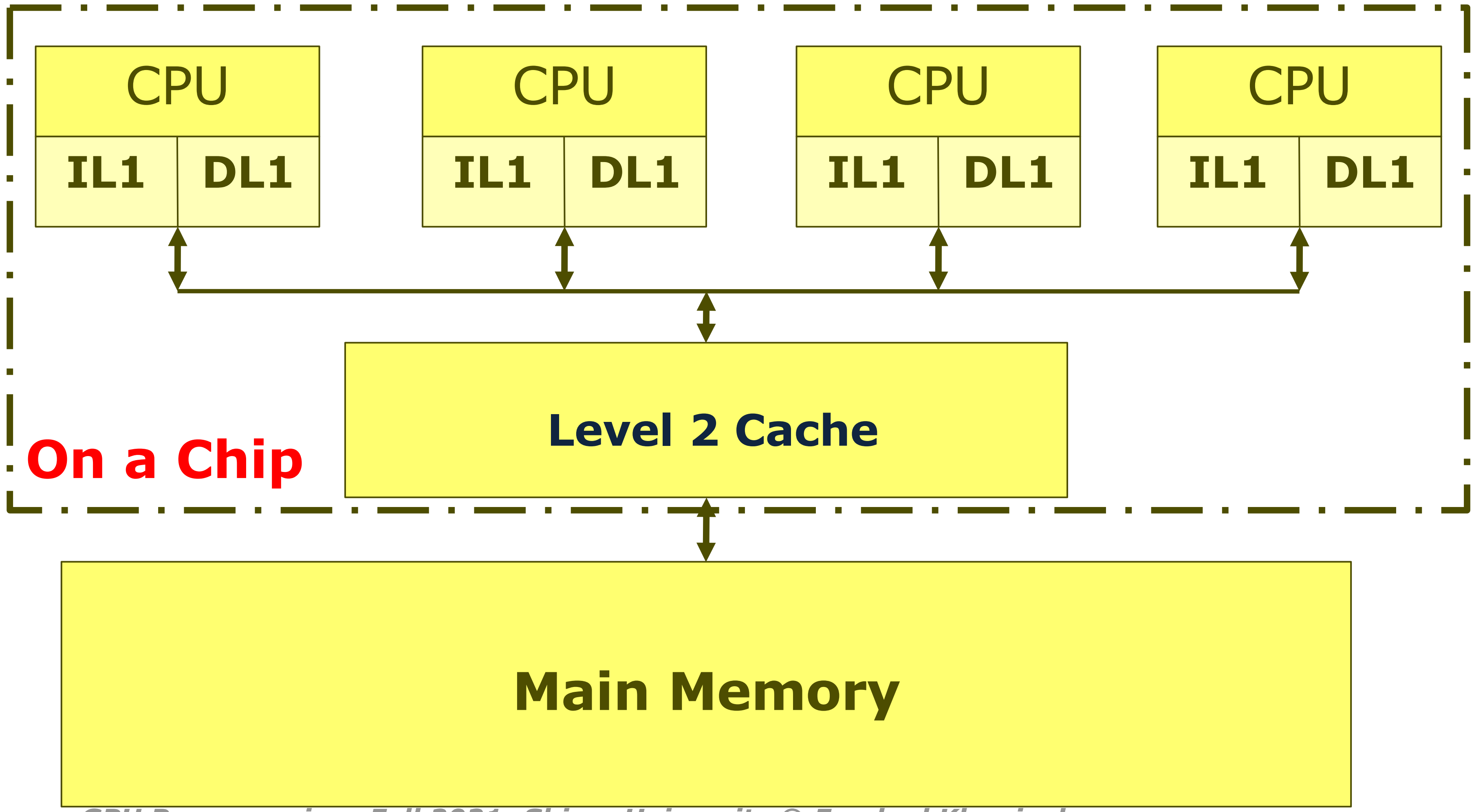
[Intel Technology Journal, Q1 2002]

Pentium-4 Hyperthreading (*Execution Pipeline)*



[Intel Technology Journal, Q1 2002]

A Typical Multi-Core Processor (aka CMP)



Designing a milticore platform

- Designers must confront single-core design options
  - Instruction fetch, wakeup, select
  - Execution unit configuration & operand bypass
  - Load/queue(s) & data cache
  - Checkpoint, log, runahead, commit.
- As well as additional degrees of freedom
  - ***How many cores? How big each?***
  - Shared caches: How many levels? How many banks?
  - On-chip interconnect: bus, switched?

Want Simple Multicore Hardware Model

To Complement Amdahl's Simple Software Mode

(1) Chip Hardware Roughly Partitioned into

- (I) Multiple Cores (with L1 caches)
- (II) The Rest (L2/L3 cache banks, interconnect, pads, etc.)
- Assume: Changing Core Size/Number does NOT change The Rest

(2) Resources for Multiple Cores Bounded

- Bound of N resources per chip for cores
- Due to area, power, cost ($$$), or multiple factors
- Bound = Power? (but pictures here use Area)

Simple Multicore Hardware Model, (cont)

(3) Architects can improve single-core performance using more of the bounded resource

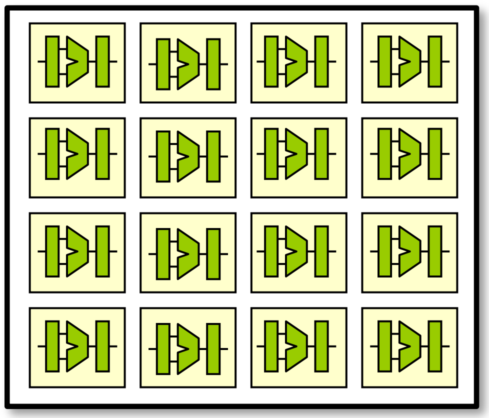
- A Simple Base Core
  - Consumes 1 Base Core Equivalent (BCE) resources
  - Provides performance normalized to 1
- An Enhanced Core (in same process generation)
  - Consumes R x BCEs
  - Performance as a function or R: Perf(R)
- What does function Perf(R) look like?

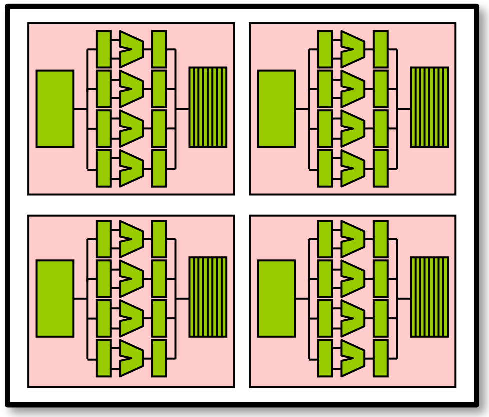
More on Enhanced Cores

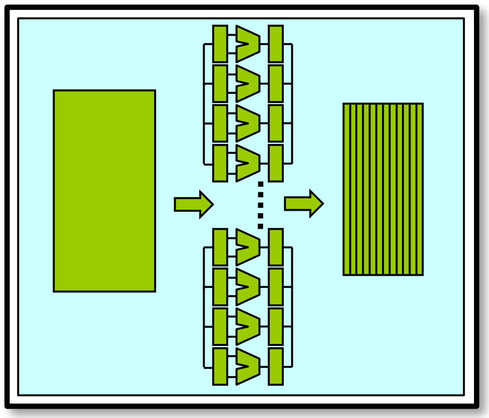
- (Performance Perf(R) consuming R BCEs resources)
- If Perf(R) > R ➔ Always enhance core
  - Cost-effectively speedups both sequential & parallel
- Therefore, equations assume Perf(R) < R
- Graphs Assume Perf(R) = square root of R
  - 2x performance for 4 BCEs, 3x for 9 BCEs, etc.
- Two options symmetric / asymmetric multicore chips

Q1: How Many (Symmetric) Cores per Chip?

- Each Chip Bounded to N BCEs (for all cores)
- Each Core consumes R BCEs
- Assume Symmetric Multicore = All Cores Identical
- Therefore, N/R Cores per Chip — (N/R)*R = N
- For an N = 16 BCE Chip:


Sixteen 1-BCE cores


Four 4-BCE cores


One 16-BCE core

Performance of Symmetric Multicore Chips

- Serial Fraction 1-F uses 1 core at rate Perf(R)
- Serial time = (1 – F) / Perf(R)

- Parallel Fraction uses N/R cores at rate Perf(R) each
- Parallel time = F / (Perf(R) * (N/R)) = F*R / Perf(R)*N

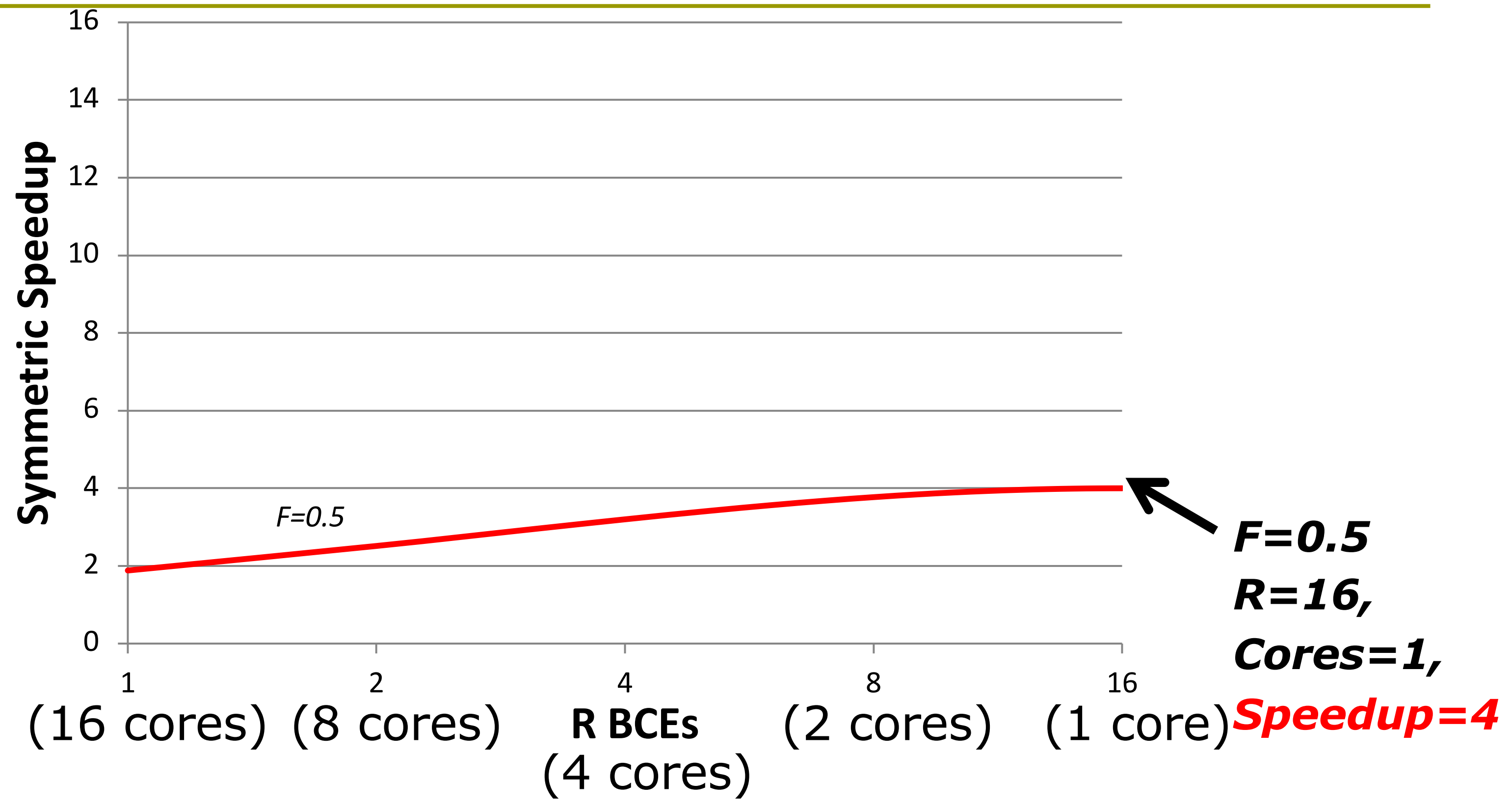
- Therefore, w.r.t. one base core:

$$\text{Symmetric Speedup} = \frac{1}{\underbrace{\dfrac{1 - F}{\text{Perf}(R)}}_{} + \underbrace{\dfrac{F * R}{\text{Perf}(R) * N}}_{}}$$

Enhanced Cores speed Serial & Parallel

- Implications?

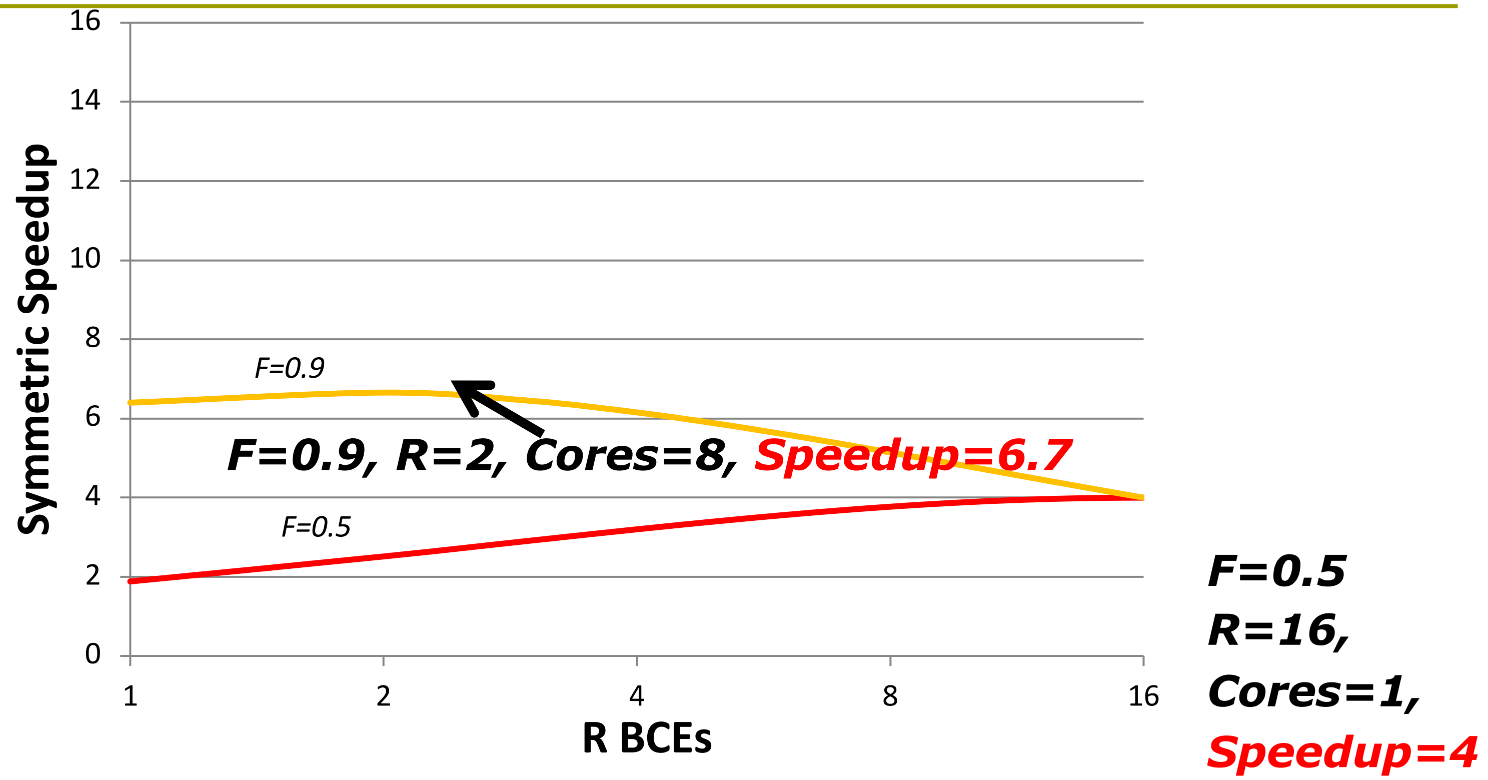
Symmetric Multicore Chip, N = 16 BCEs



F=0.5, Opt. Speedup S = 4 = 1/(0.5/4 + 0.5*16/(4*16))

Need more parallelism to have multicore optimal!

Symmetric Multicore Chip, N = 16 BCEs



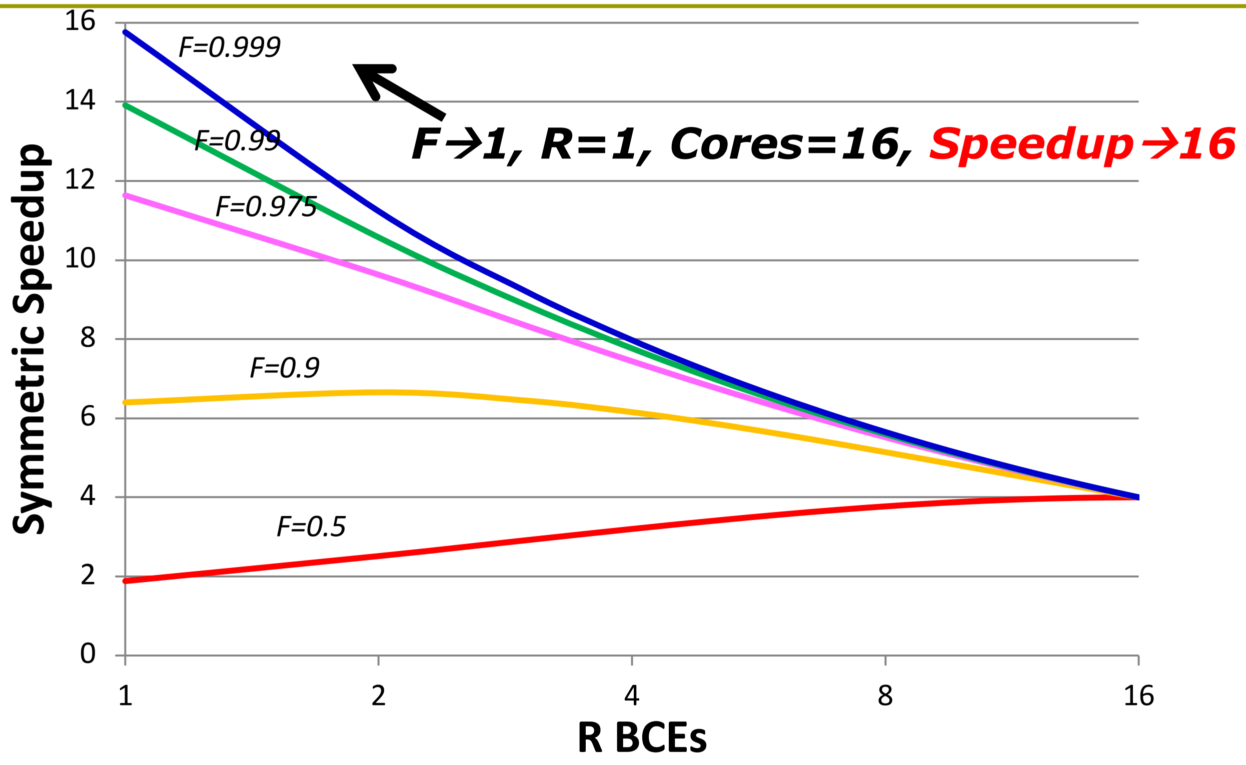
F=0.9, R=2, Cores=8, Speedup=6.7

F=0.5 R=16, Cores=1, Speedup=4

At F=0.9, Multicore optimal, but speedup limited

Need to obtain even more parallelism!

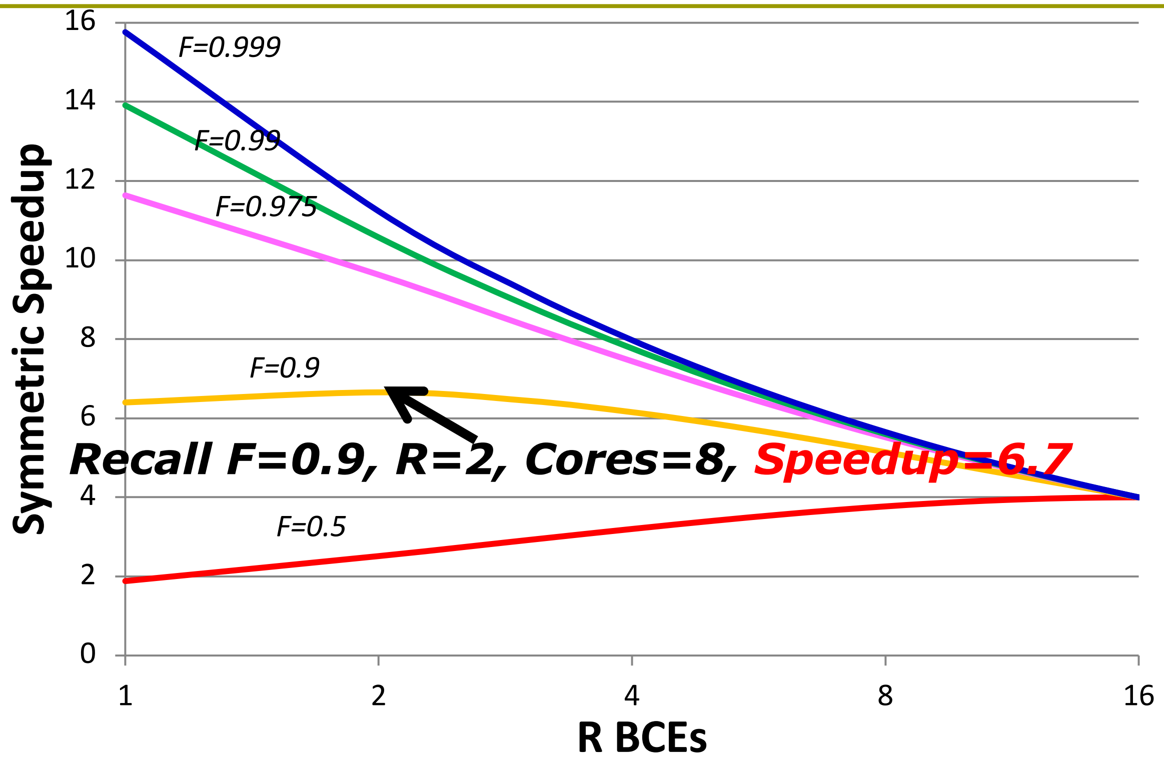
Symmetric Multicore Chip, N = 16 BCEs



F matters: Amdahl's Law applies to multicore chips

Researchers should target parallelism F first

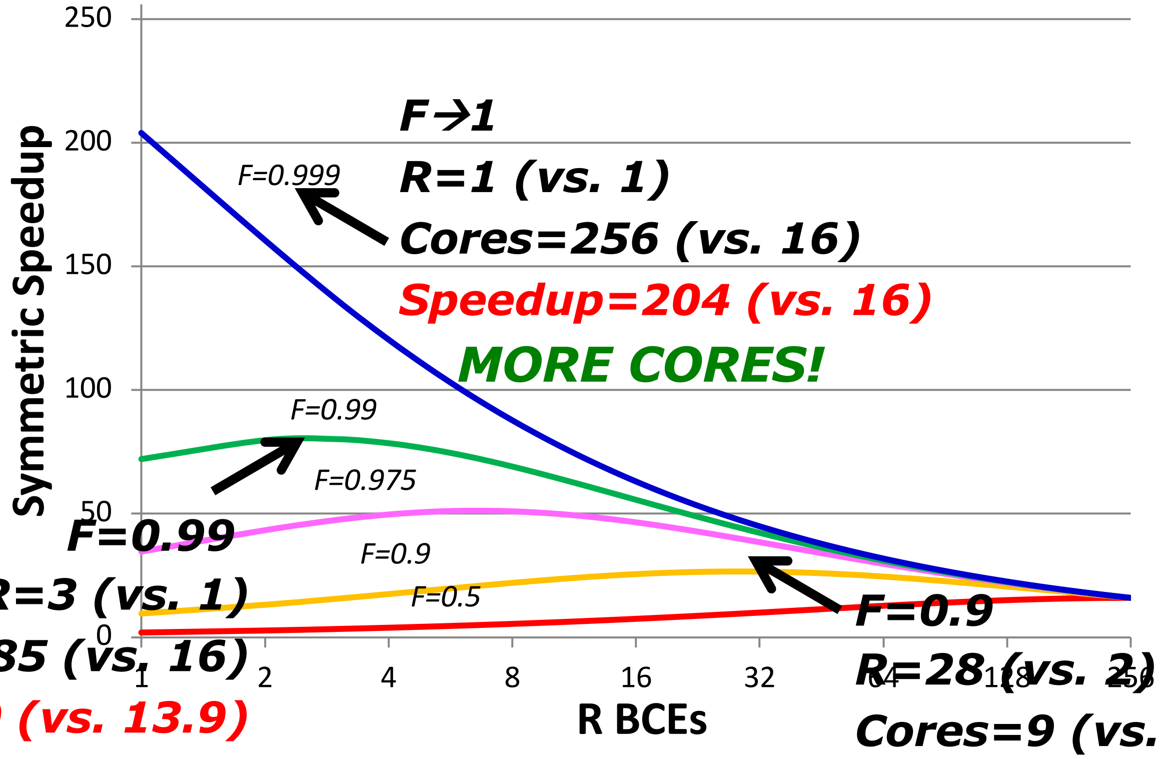
Symmetric Multicore Chip, N = 16 BCEs



As Moore's Law enables N to go from 16 to 256 BCEs,

More core enhancements? More cores? Or both?

Symmetric Multicore Chip, N = 256 BCEs



F→1
R=1 (vs. 1)
Cores=256 (vs. 16)
Speedup=204 (vs. 16)
MORE CORES!

F=0.99
R=3 (vs. 1)
Cores=85 (vs. 16)
Speedup=80 (vs. 13.9)
ORE ENHANCEMENTS
& MORE CORES!

F=0.9
R=28 (vs. 2)
Cores=9 (vs. 8)
Speedup=26.7 (vs. 6.7)
CORE ENHANCEMENTS!

As Moore's Law increases N, often need enhanced core designs

Researcher should target single core performance too

Outline

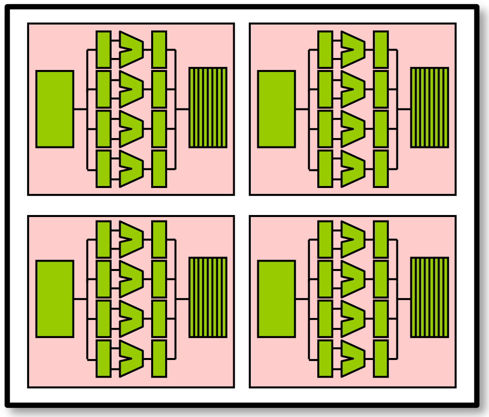
- Symmetric Multicore Chips
- **Asymmetric Multicore Chips**

Asymmetric (Heterogeneous) Multicore Chips

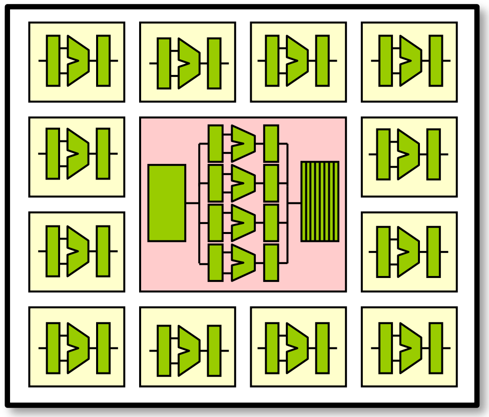
- Symmetric Multicore Required All Cores Equal
- Why Not Enhance Some (But Not All) Cores?
- For Amdahl's Simple Software Assumptions
  - One Enhanced Core
  - Others are Base Cores
- How does this affect our hardware model?

How Many Cores per Asymmetric Chip?

- Each Chip Bounded to N BCEs (for all cores)
- One R-BCE Core leaves N-R BCEs
- Use N-R BCEs for N-R Base Cores
- Therefore, 1 + N - R Cores per Chip
- For an N = 16 BCE Chip:



Symmetric: Four 4-BCE cores



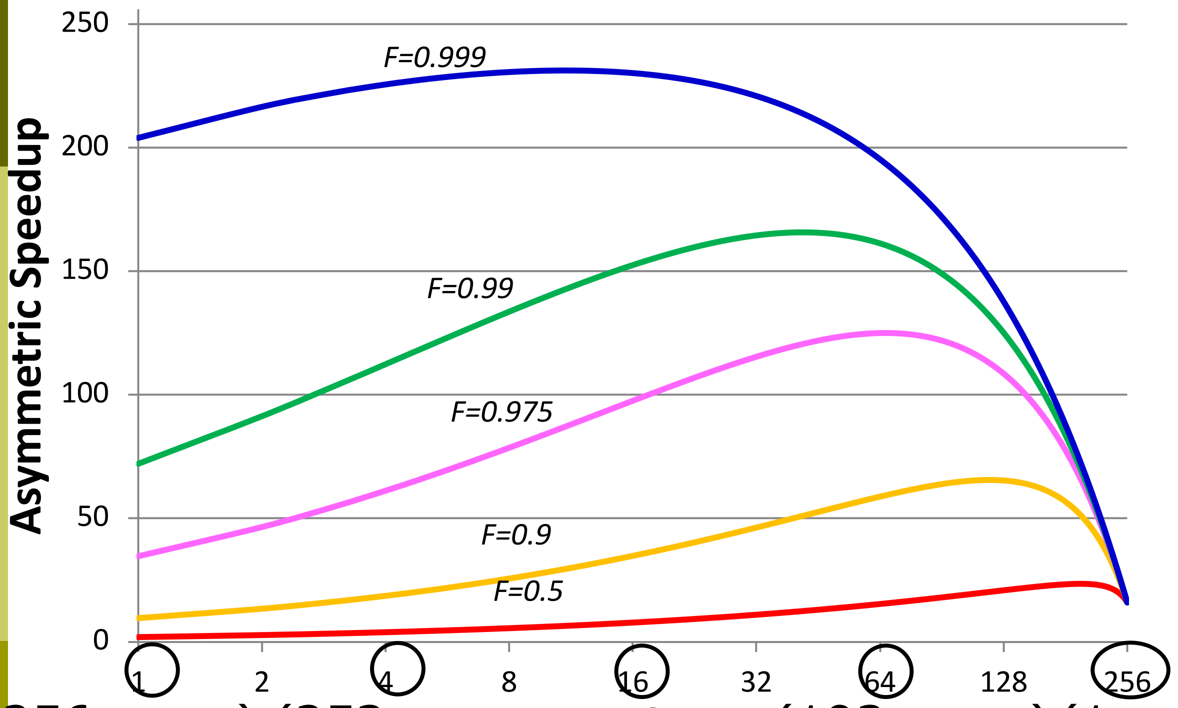
Asymmetric: One 4-BCE core & Twelve 1-BCE base cores

Performance of Asymmetric Multicore Chips

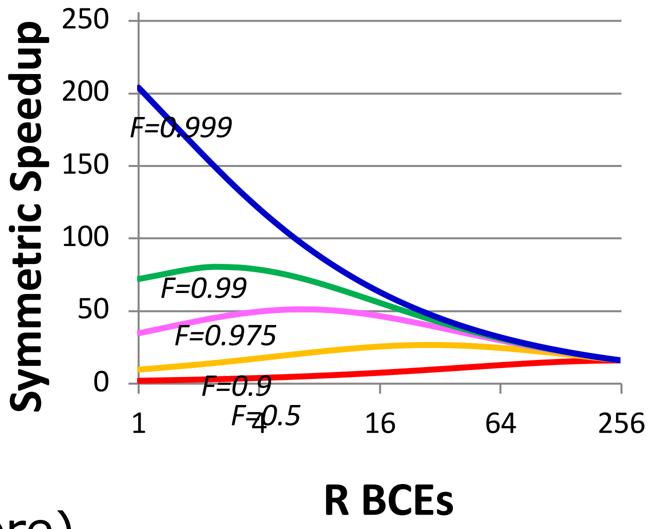
- Serial Fraction 1-F same, so time = (1 – F) / Perf(R)
- Parallel Fraction F
  - One core at rate Perf(R)
  - N-R cores at rate 1
  - Parallel time = F / (Perf(R) + N - R)
- Therefore, w.r.t. one base core:

$$\text{Asymmetric Speedup} = \frac{1}{\frac{1 - F}{\text{Perf}(R)} + \frac{F}{\text{Perf}(R) + N - R}}$$

Asymmetric Multicore Chip, N = 256 BCEs



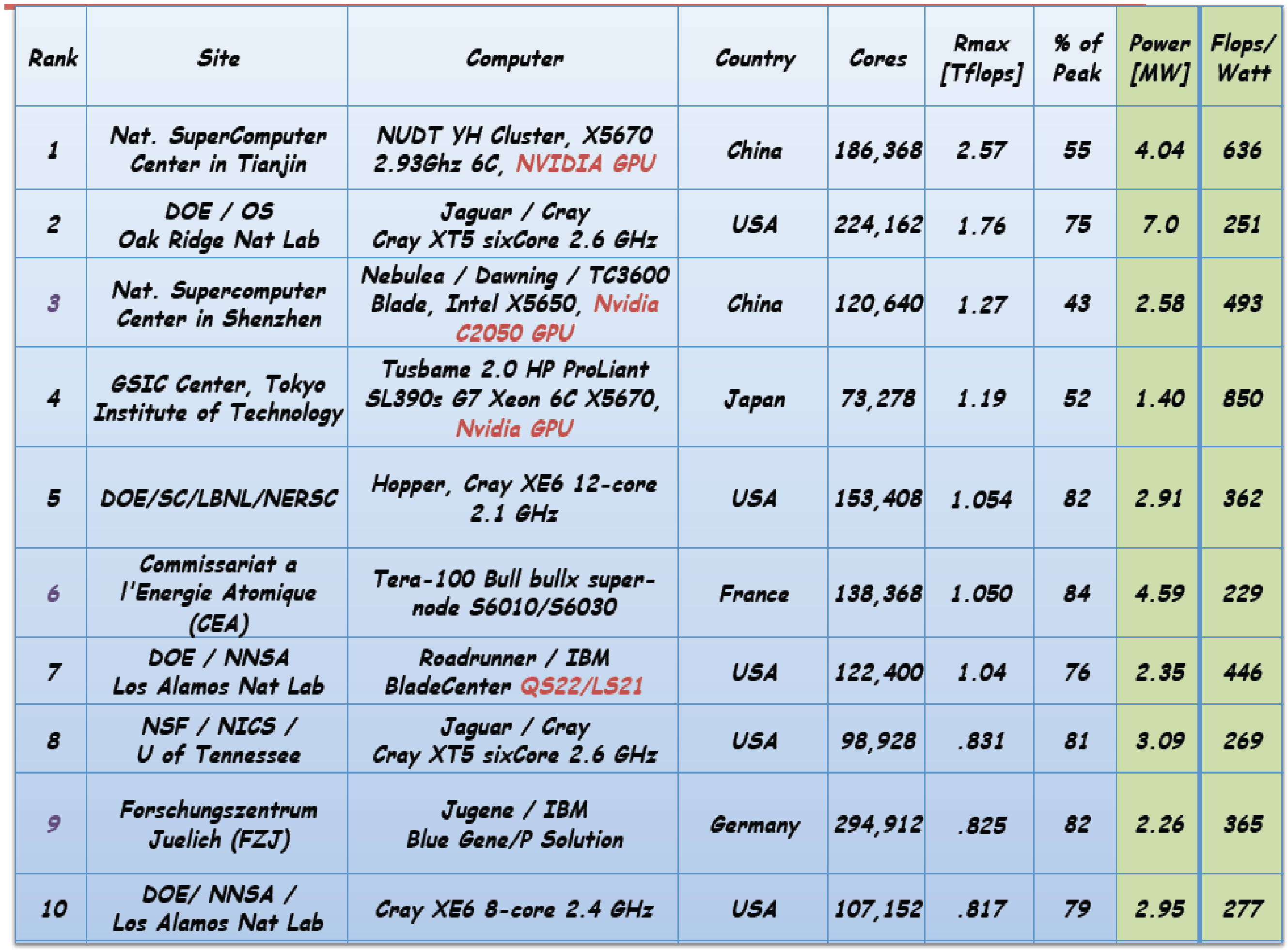
(256 cores) (253 cores) (241 cores) (193 cores) (1 core)



Number of Cores = 1 (Enhanced) + 256 – R (Base)

How do Asymmetric & Symmetric speedups compare?

Hybrid architectures in Top 500 sites



Rank	Site	Computer	Country	Cores	Rmax [Tflops]	% of Peak	Power [MW]	Flops/ Watt
1	Nat. SuperComputer Center in Tianjin	NUDT YH Cluster, X5670 2.93Ghz 6C, NVIDIA GPU	China	186,368	2.57	55	4.04	636
2	DOE / OS Oak Ridge Nat Lab	Jaguar / Cray Cray XT5 sixCore 2.6 GHz	USA	224,162	1.76	75	7.0	251
3	Nat. Supercomputer Center in Shenzhen	Nebulea / Dawning / TC3600 Blade, Intel X5650, Nvidia C2050 GPU	China	120,640	1.27	43	2.58	493
4	GSIC Center, Tokyo Institute of Technology	Tusbame 2.0 HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU	Japan	73,278	1.19	52	1.40	850
5	DOE/SC/LBNL/NERSC	Hopper, Cray XE6 12-core 2.1 GHz	USA	153,408	1.054	82	2.91	362
6	Commissariat a l'Energie Atomique (CEA)	Tera-100 Bull bullx super-node S6010/S6030	France	138,368	1.050	84	4.59	229
7	DOE / NNSA Los Alamos Nat Lab	Roadrunner / IBM BladeCenter QS22/LS21	USA	122,400	1.04	76	2.35	446
8	NSF / NICS / U of Tennessee	Jaguar / Cray Cray XT5 sixCore 2.6 GHz	USA	98,928	.831	81	3.09	269
9	Forschungszentrum Juelich (FZJ)	Jugene / IBM Blue Gene/P Solution	Germany	294,912	.825	82	2.26	365
10	DOE/ NNSA / Los Alamos Nat Lab	Cray XE6 8-core 2.4 GHz	USA	107,152	.817	79	2.95	277

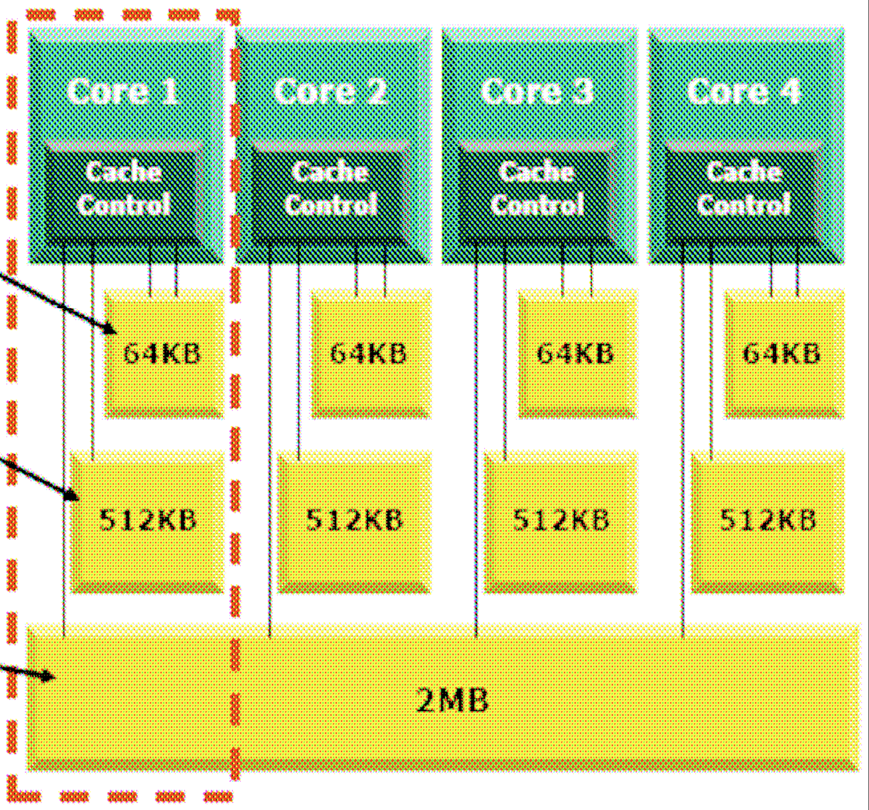
Some Modern CPUs

AMD Barcelona

Dedicated L1
- Locality keeps most critical data in the L1 cache
- Lowest latency
- 2 loads per cycle

Dedicated L2
- Sized to accommodate the majority of working sets today
- Dedicated to eliminate conflicts common in shared caches
  - Better for Virtualization

Shared L3 – NEW
- Victim-cache architecture maximizes efficiency of cache hierarchy
- Fills from L3 leave likely shared lines in the L3
- Sharing-aware replacement policy
- Ready for expansion at the right time for customers

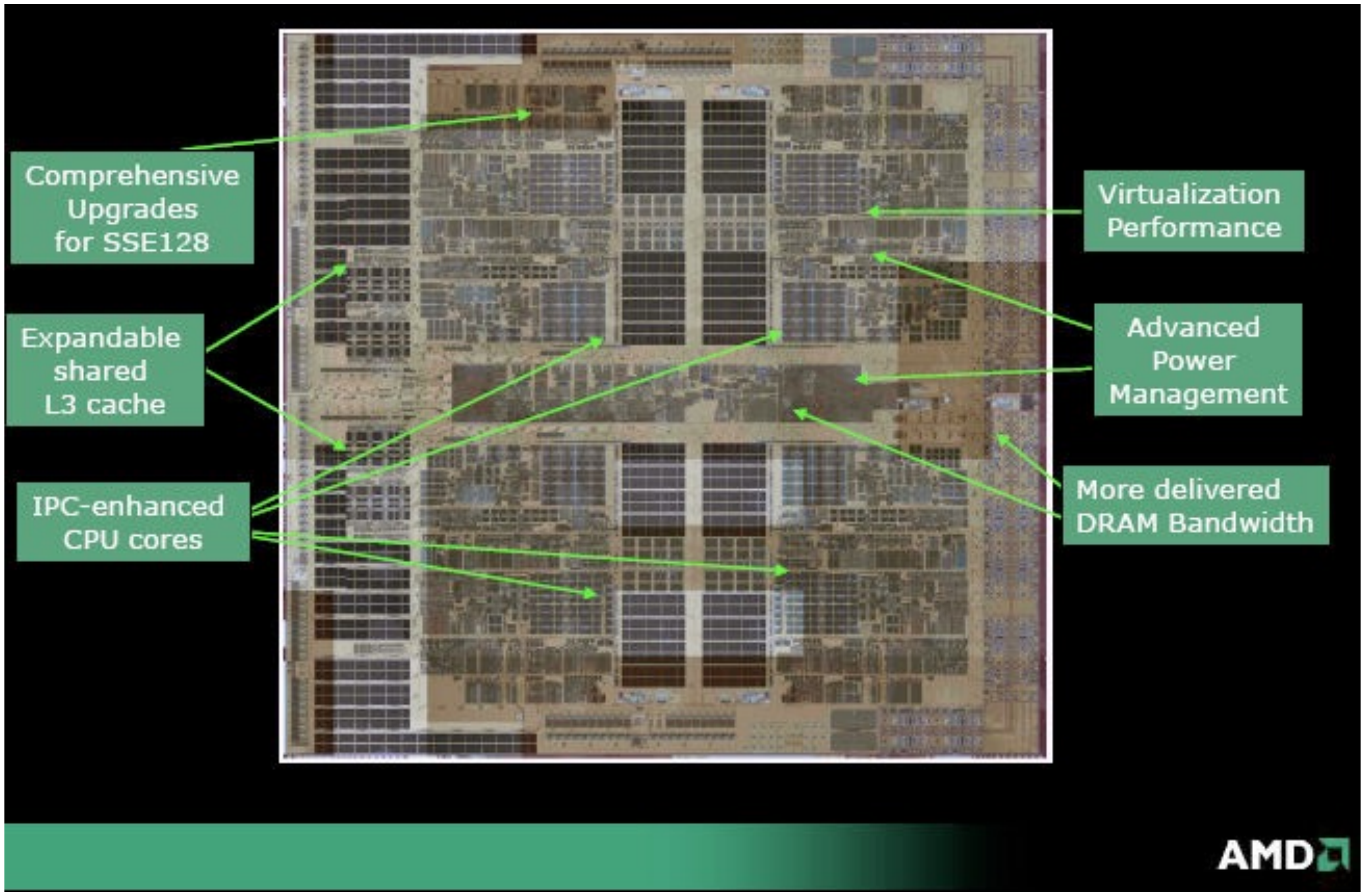


Source: AMD ©

AMD Barcelona

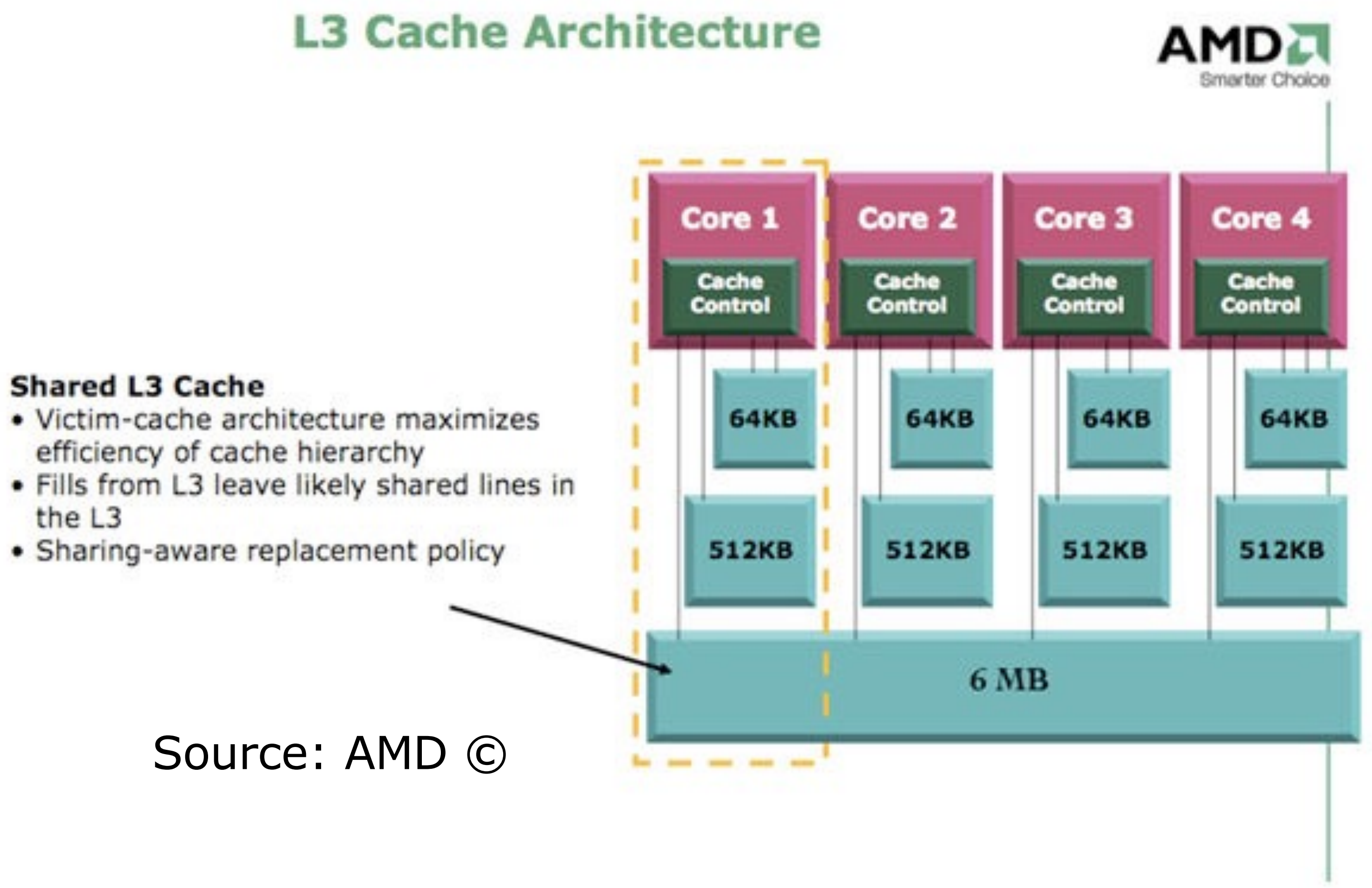
- L1: 2-way set associative, LRU replacement, block size 64 bytes, split I & D, write-back & write-allocate, 3 cycles latency
- L2: idem. except 9 cycles latency
- L3: idem. except evict block shared by fewest core and 34 cycles latency

AMD Barcelona (65 nm, < 3GHz)


Comprehensive Upgrades for SSE128
Expandable shared L3 cache
IPC-enhanced CPU cores
Virtualization Performance
Advanced Power Management
More delivered DRAM Bandwidth
AMD

Source: AMD ©

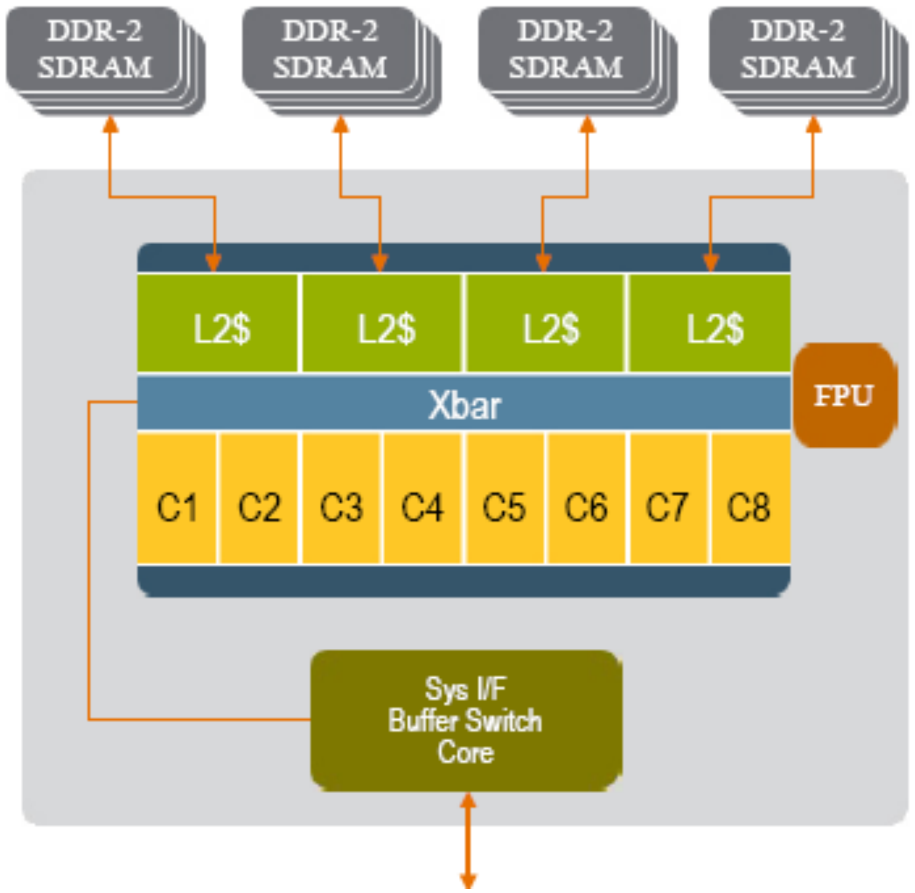
AMD Shanghai (45nm, < 2.6 GHz)



Shared L3 Cache
- Victim-cache architecture maximizes efficiency of cache hierarchy
- Fills from L3 leave likely shared lines in the L3
- Sharing-aware replacement policy

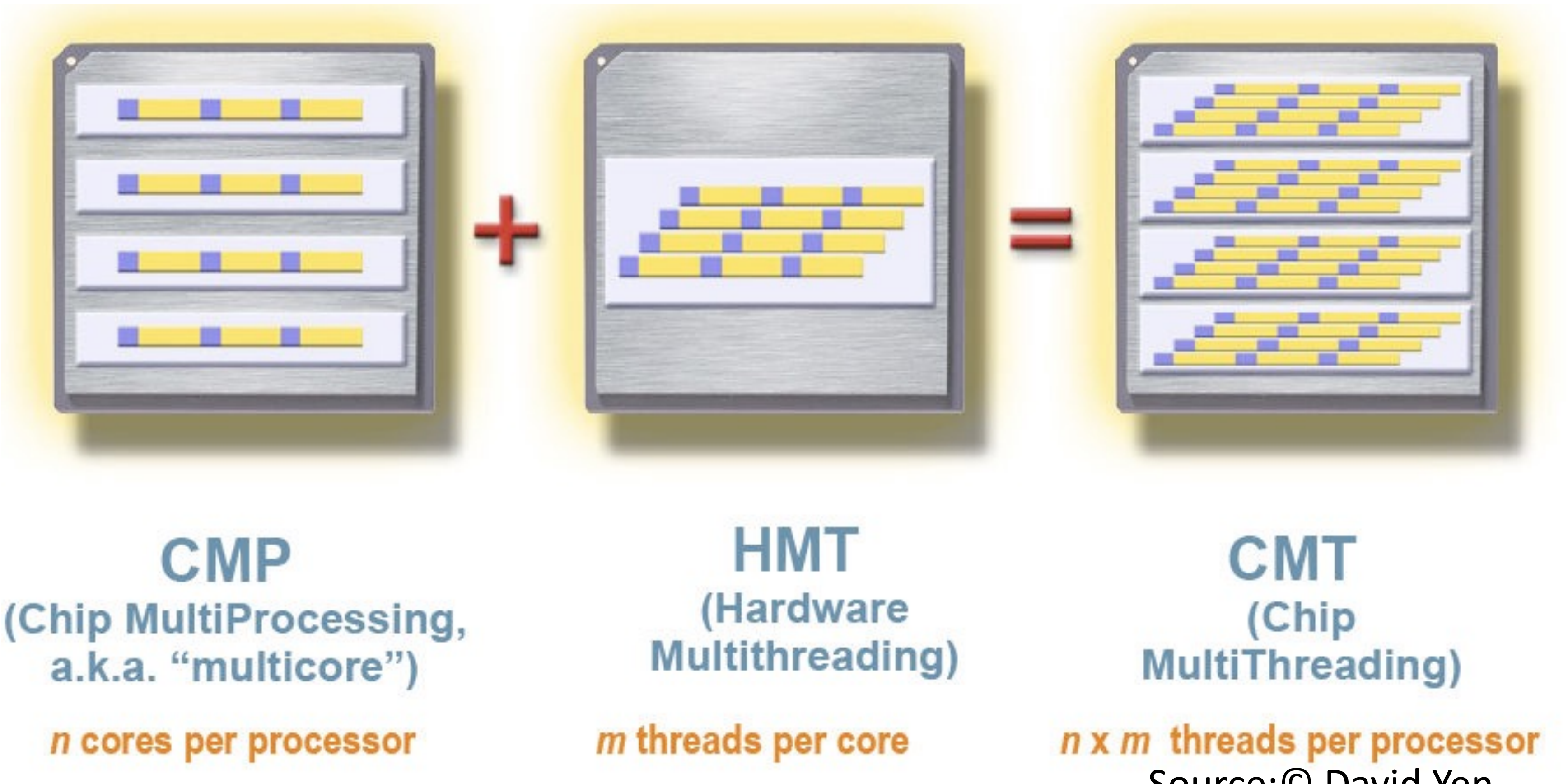
Source: AMD ©

SUN© Ultra-sparc T1 (Niagara 1)



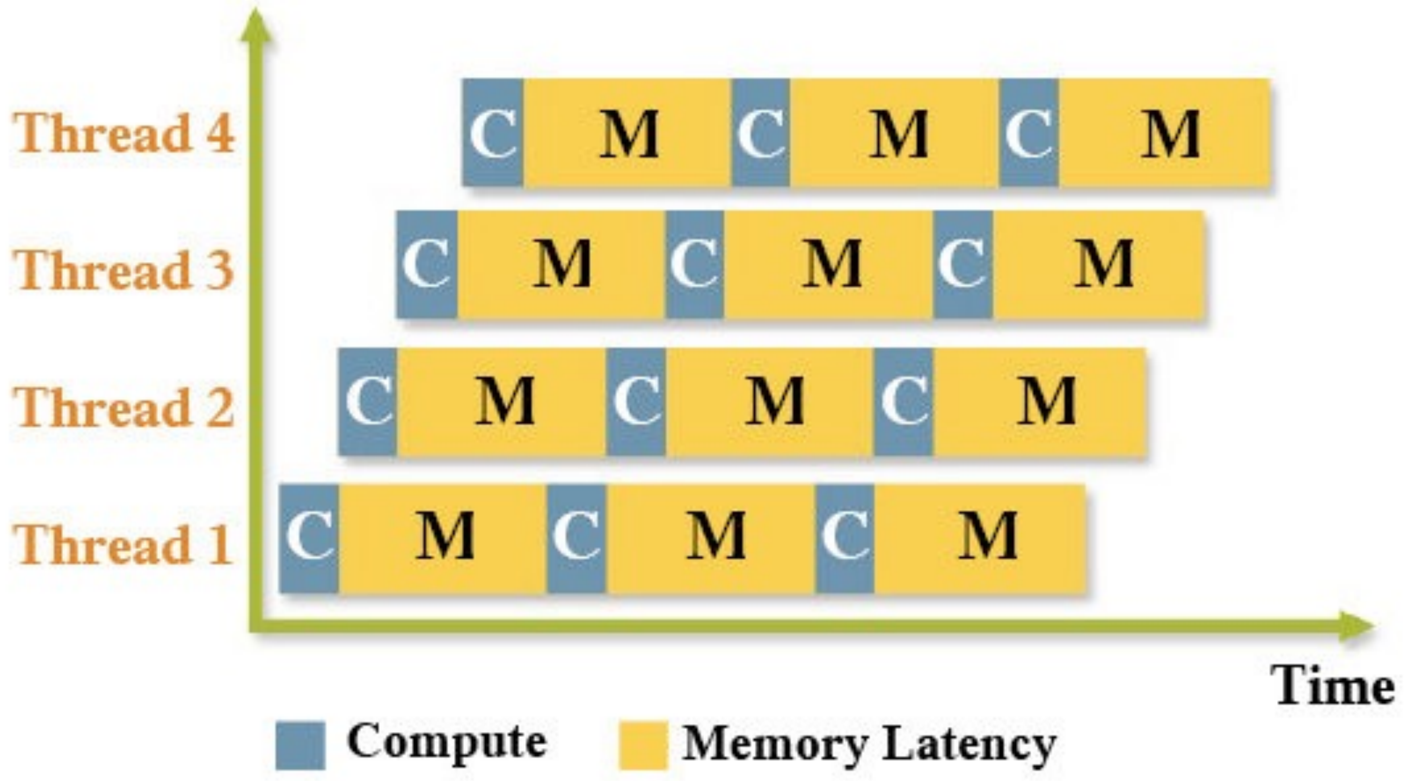
Source: Sun ©

Chip Multi-threading



Source:© David Yen

Niagara 1

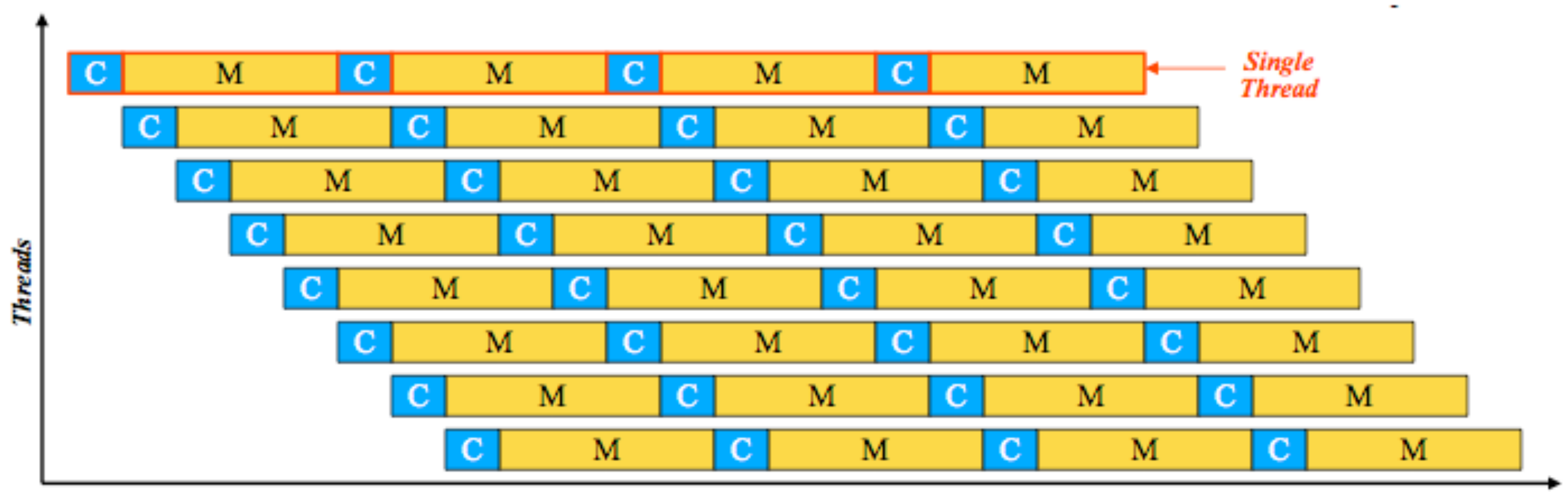


While one thread is blocked, other threads continue computing – results in higher IPC

Source:© David Yen

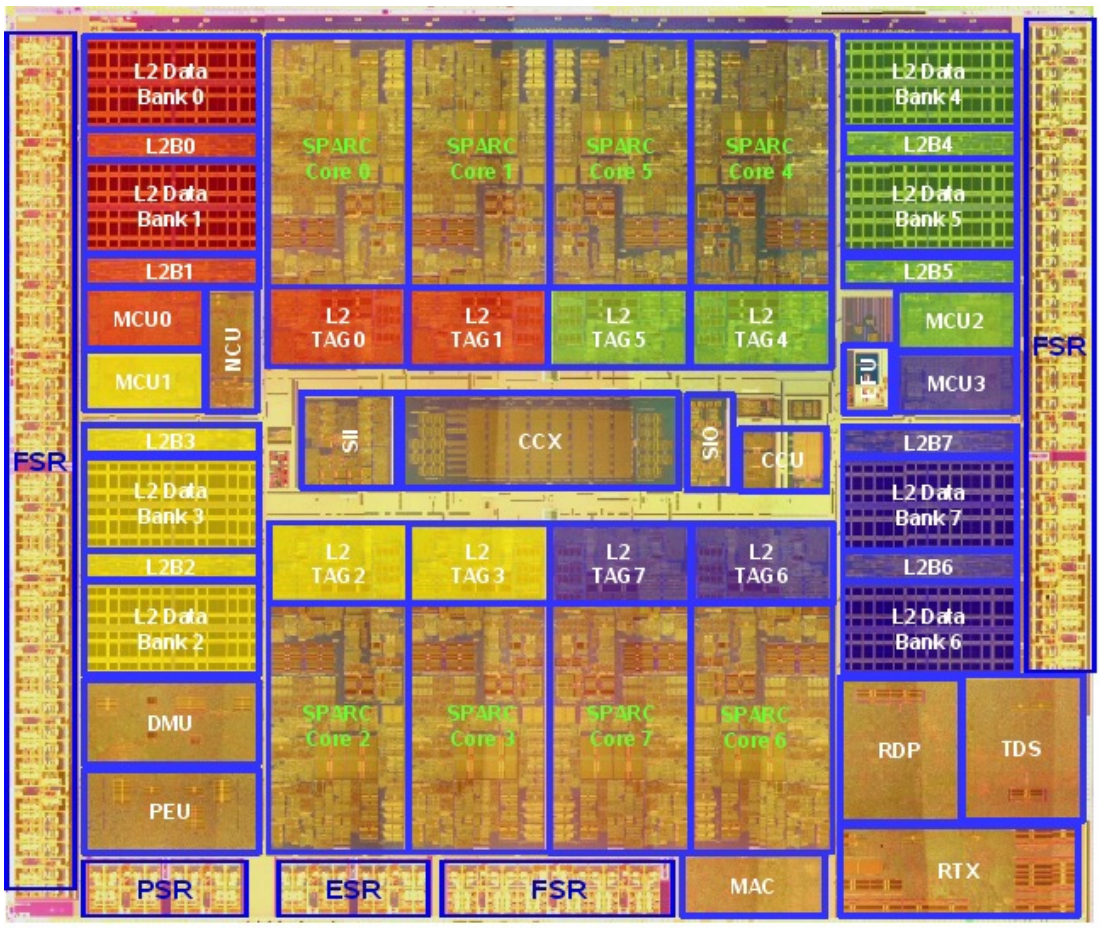
SUN© Ultra-sparc T2 (Niagara 2)

- 8 Cores (each 8 Threads) Total 64 Threads



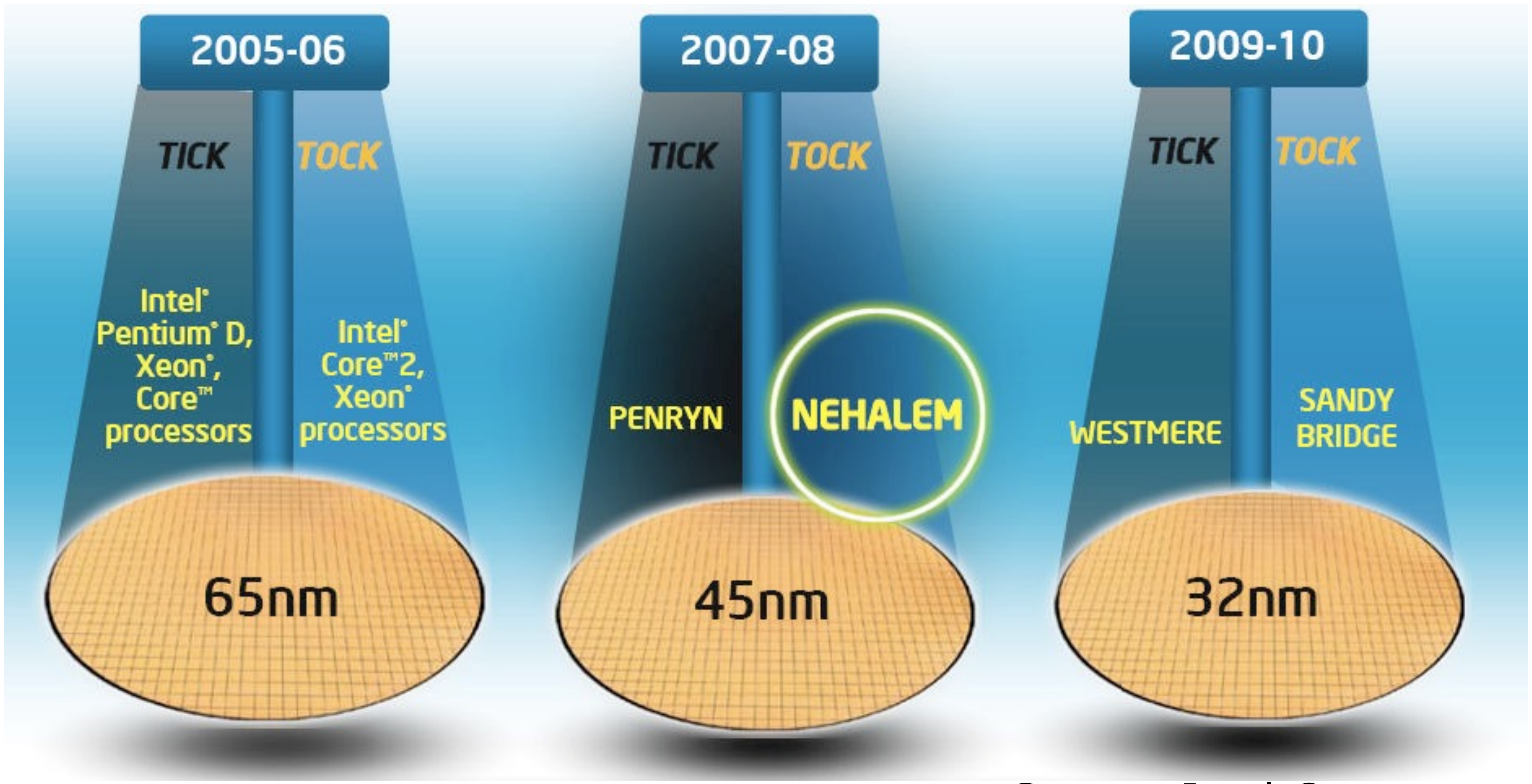
Source:© Robert Golla

Ultra-sparc T2 die



Source: SUN ©

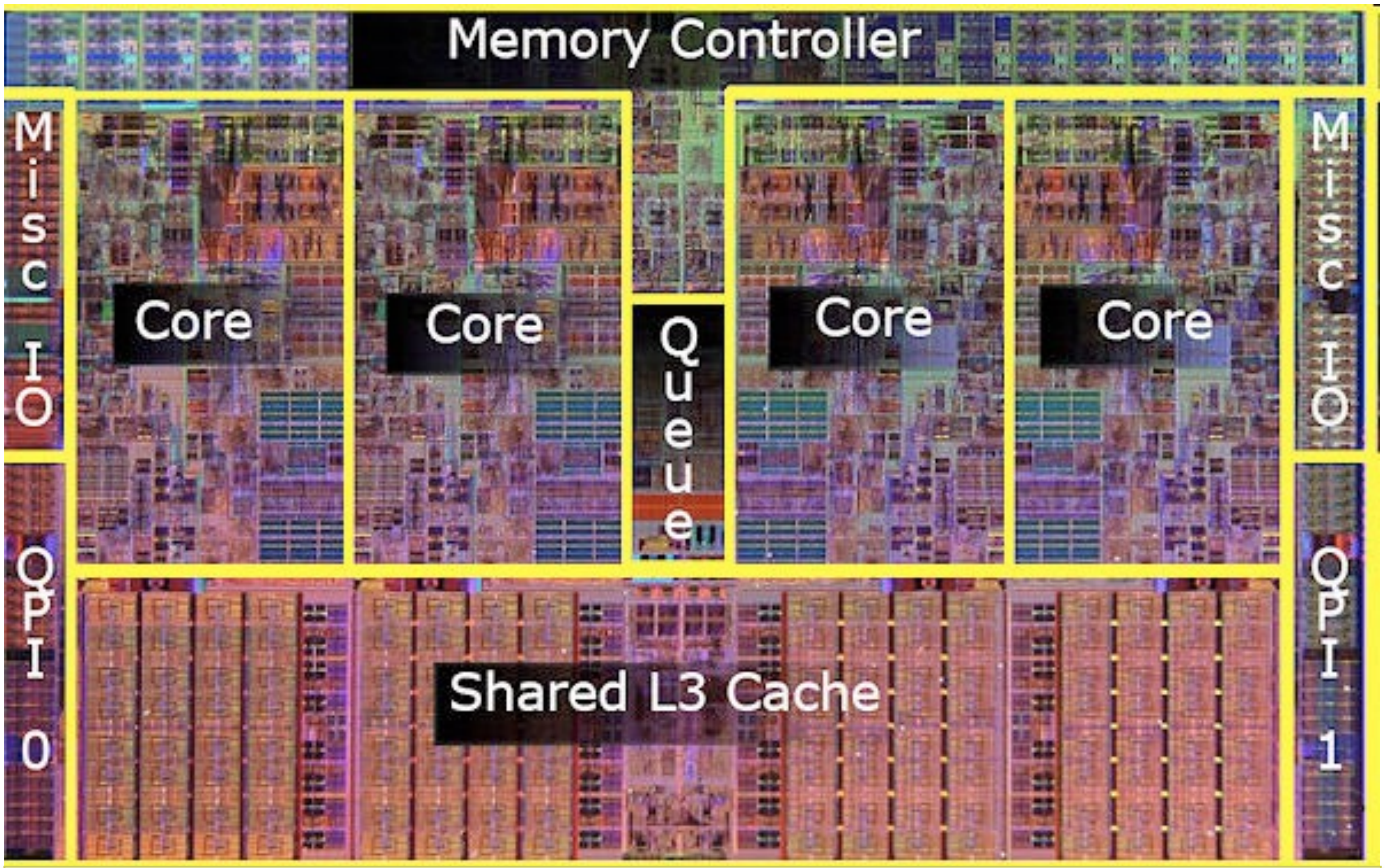
Intel© Tick Tock



Source: Intel ©

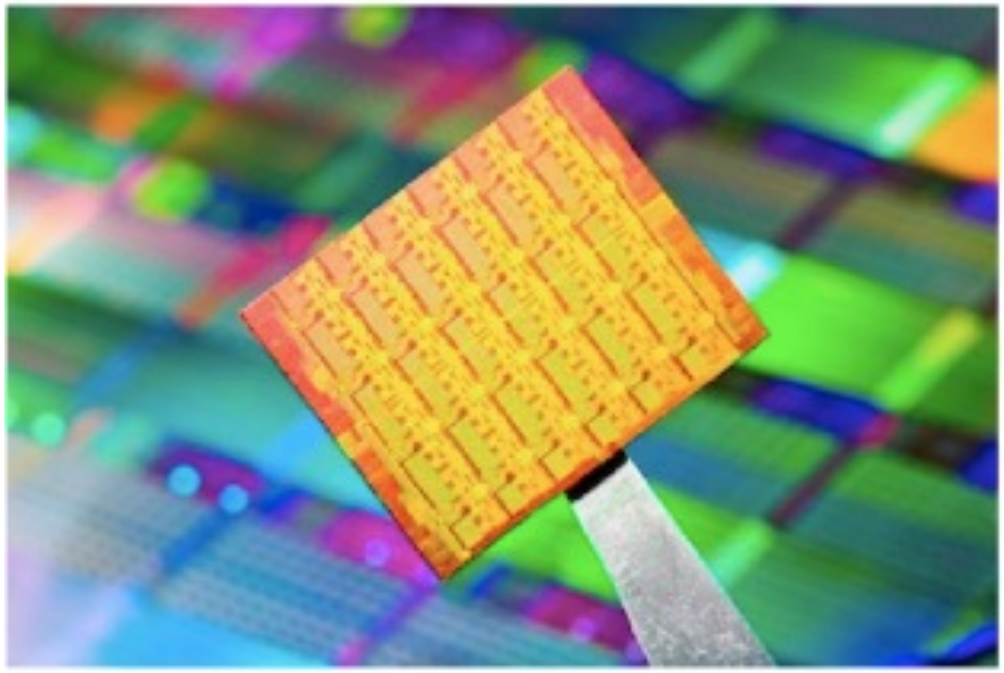
Intel© Core i7 (Nehalem, 2.6-3.2 GHz)

Source: Intel ©

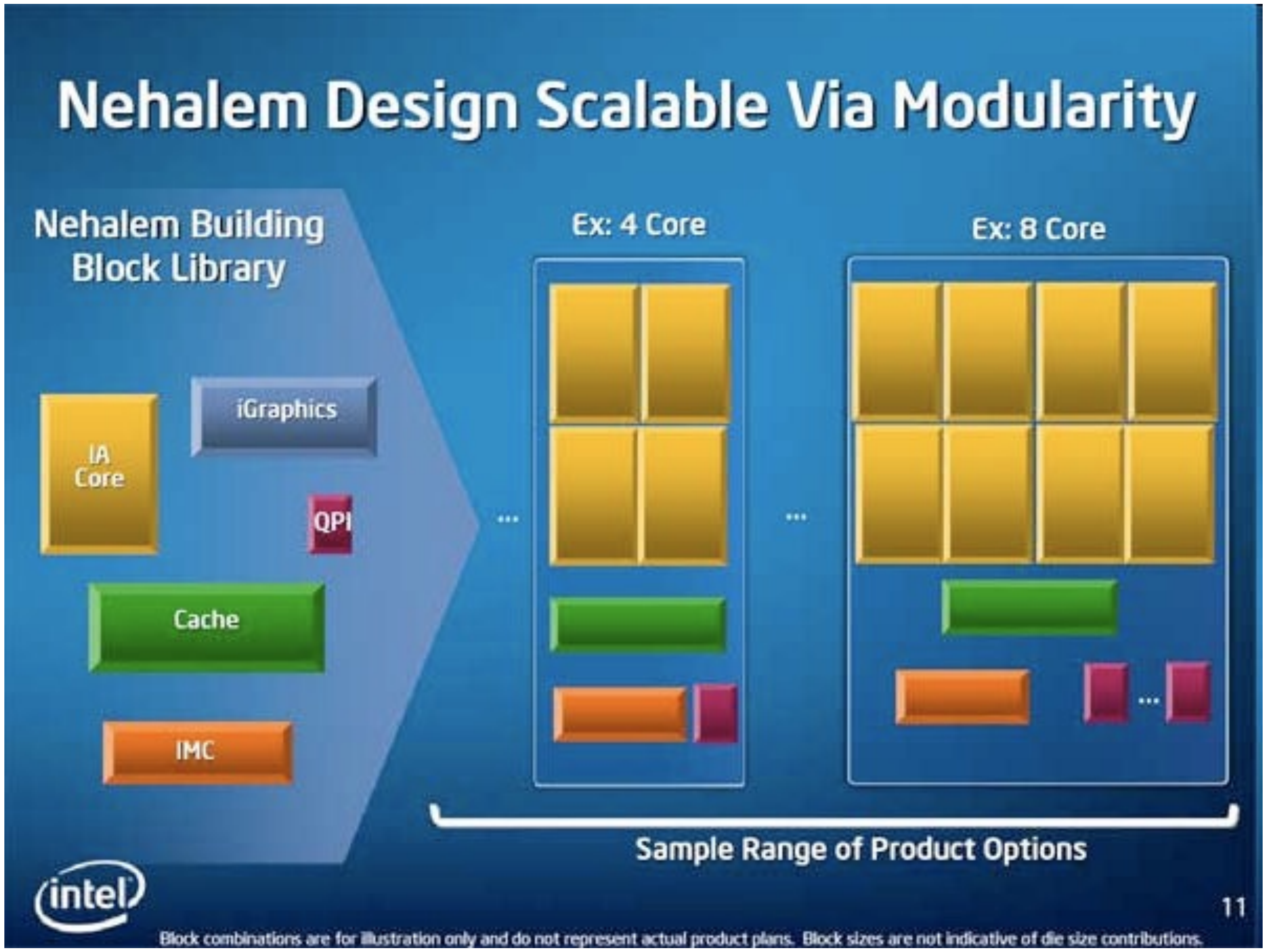


2 SMT per core

Intel Single Chip Cloud Computer 48 Cores On A Single Chip



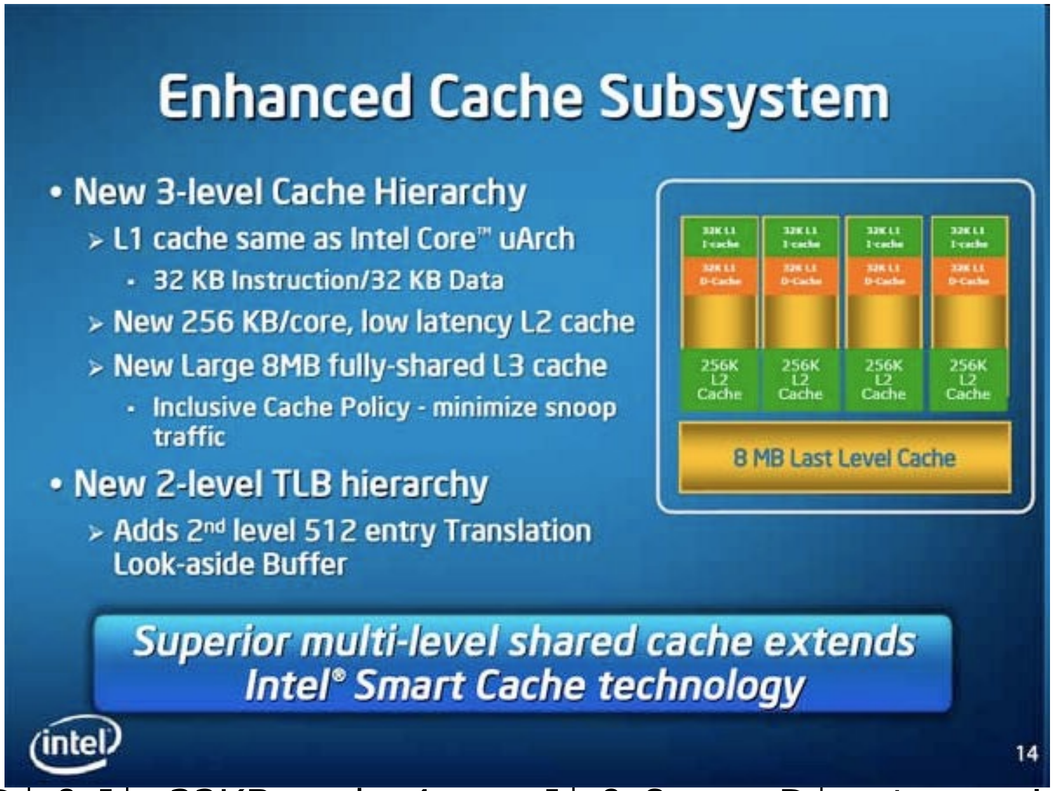
Nehalem


Nehalem Design Scalable Via Modularity
Nehalem Building Block Library
IA Core
iGraphics
QPI
Cache
IMC
Ex: 4 Core
Ex: 8 Core
Sample Range of Product Options
intel
11
Block combinations are for illustration only and do not represent actual product plans. Block sizes are not indicative of die size contributions.

Source: Intel ©

Nehalem Caches

Source: Intel ©



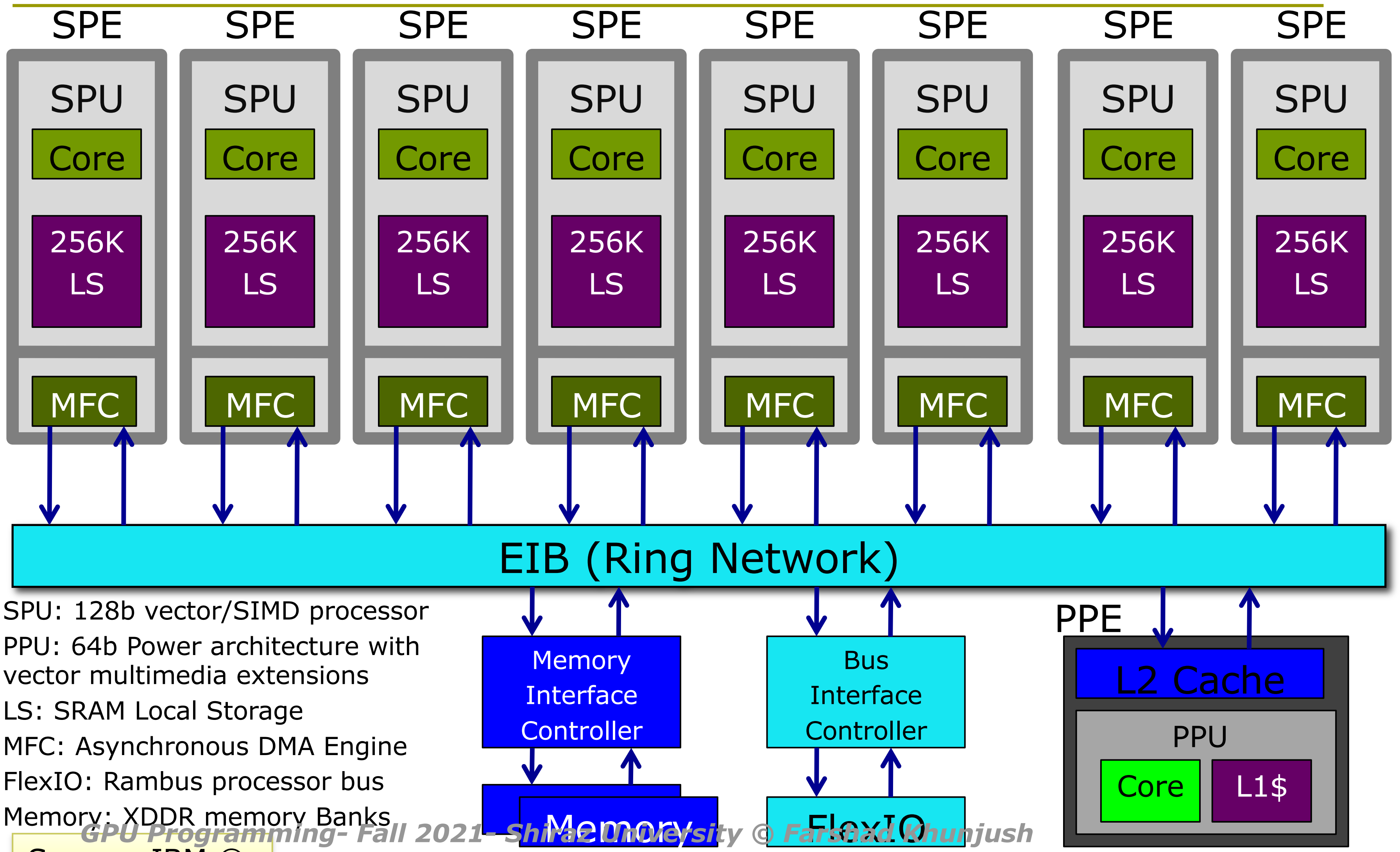
Enhanced Cache Subsystem

- New 3-level Cache Hierarchy
  - L1 cache same as Intel Core™ uArch
    - 32 KB Instruction/32 KB Data
  - New 256 KB/core, low latency L2 cache
  - New Large 8MB fully-shared L3 cache
    - Inclusive Cache Policy - minimize snoop traffic
- New 2-level TLB hierarchy
  - Adds 2nd level 512 entry Translation Look-aside Buffer

Superior multi-level shared cache extends Intel® Smart Cache technology

- L1: split D$ & I$, 32KB each, 4-way I$ & 8-way D$ set associative, approx. LRU, block size 64 bytes, write-back & write-allocate
- L2: 8-way set associative, idem.
- L3: 16-way set associative, idem

Cell BE© Architecture (Heterogeneous CMP)



SPU: 128b vector/SIMD processor
PPU: 64b Power architecture with vector multimedia extensions
LS: SRAM Local Storage
MFC: Asynchronous DMA Engine
FlexIO: Rambus processor bus
Memory: XDDR memory Banks

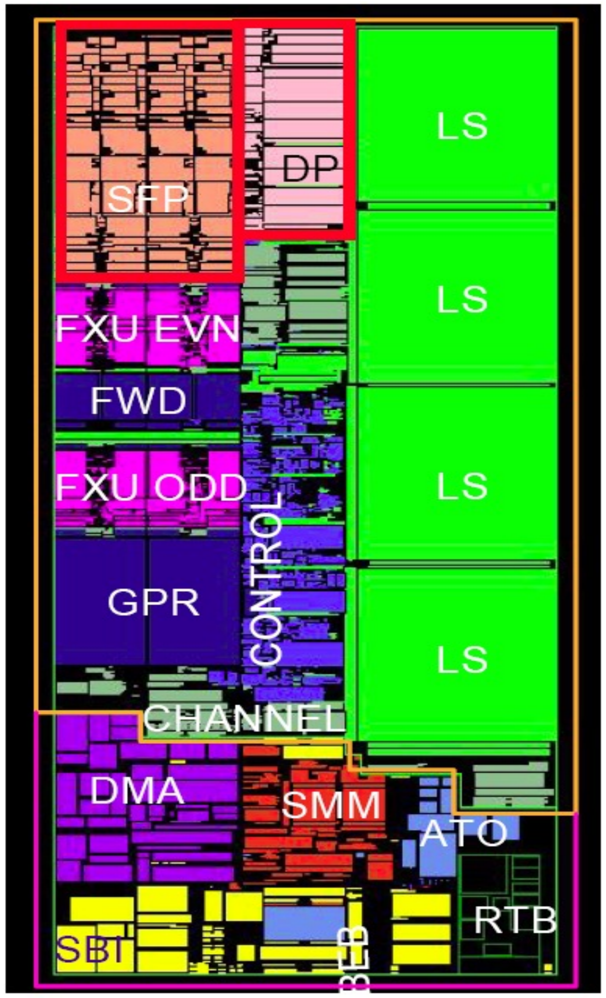
Source: IBM ©

PPE

- IBM 64-bit Power Architecture (Dual-threaded)
- 128-bit vector media extension
- Two Level cache hierarchy
  - L1 (32 KB I & D)
  - L2 (512 KB)
- Runs the OS
- Provides Application Control
- Handles Virtual Memory

SPE

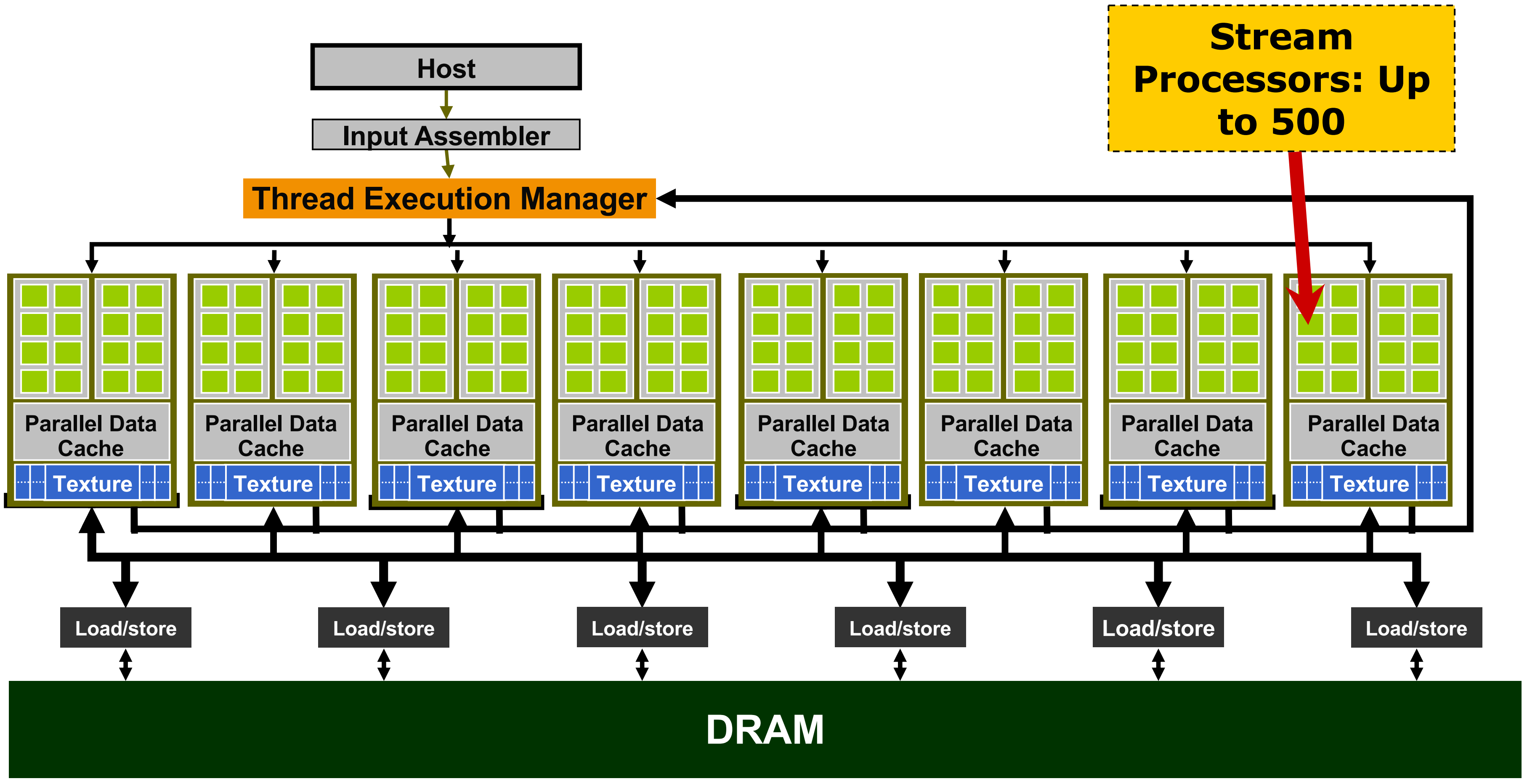
- A RISC architecture
  - 32-bit fixed instructions
  - Load/Store architecture
  - 128 unified registers (128b)
- User-mode architecture
  - No Page translation within SPU
- VMX-like SIMD dataflow
- 256KB Local store
  - Combined Instruction & Data
- DMA block transfer



14.5mm² (90nm SOI)

Source: Kahle, Spring Processor Forum 2005

GPU Architecture (NVIDIA)



Source:NVIDIA

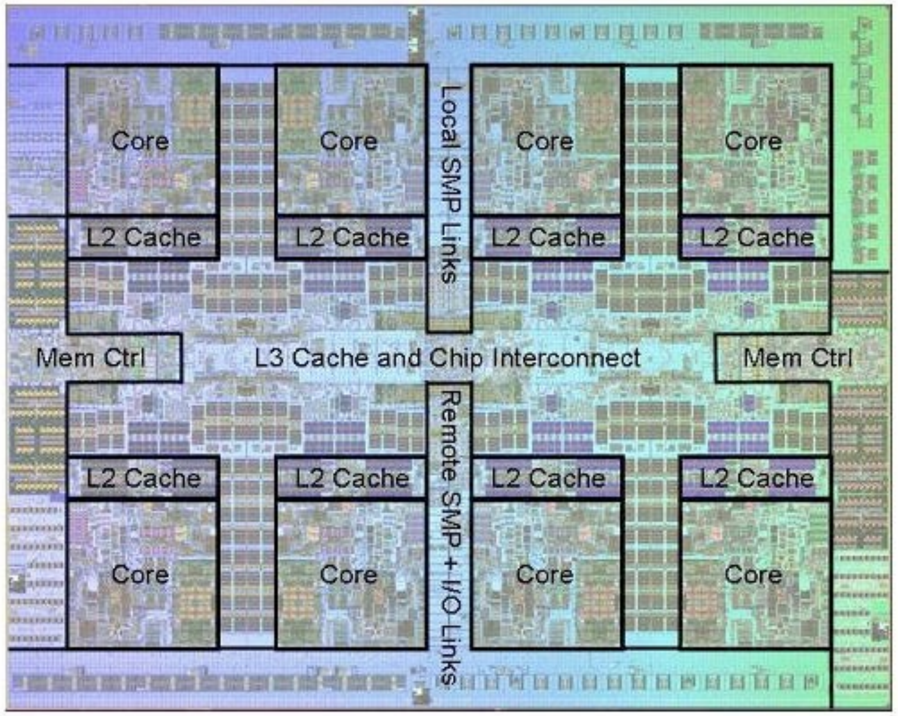
Trend in Multi-core processors

- AMD Opteron introduced in 2003
  - Hypertransport
  - On-chip memory controller
- In 2006
  - Sun Niagara I: 8 cores/32 thread & 4 on-chip memory controllers
  - Intel & AMD: dual core
  - IBM: dual core & Cell
  - Azul: 24 cores
- In 2007
  - Sun Niagara II: 8 cores/64 threads
  - Intel quad-core vs. AMD quad-core (Barcelona)
  - Azul: 48 cores
- In 2009
  - Sun Victoria Falls: 4 Niagara II+ (256 threads)
    - Server (web, DBs)
  - Intel Nehalem (core i7 & Xeon) vs. AMD quad-core (Shanghai) & six-core (Istanbul)
    - Desktop/workstations/Server
  - ARM: Cortex A9 4 cores
    - low-power embedded systems (e.g. mobile phones)
  - Nvidia: Tesla (HPC), GTX

2010- New Systems

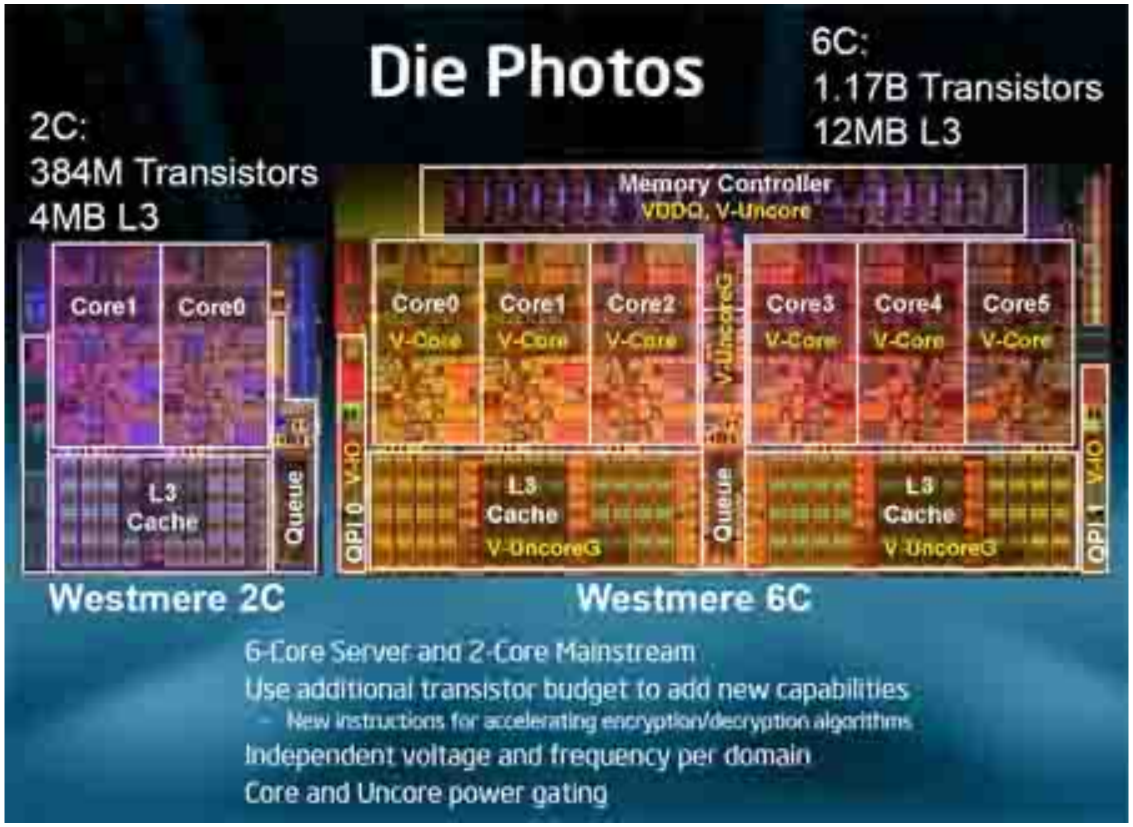
POWER7 Processor Chip

- 567mm^2 Technology: 45nm lithography, Cu, SOI, eDRAM
- 1.2B transistors
  - Equivalent function of 2.7B
  - eDRAM efficiency
- Eight processor cores
  - 12 execution units per core
  - 4 Way SMT per core
  - 32 Threads per chip
  - 256KB L2 per core
- 32MB on chip eDRAM shared L3
- Dual DDR3 Memory Controllers
  - 100GB/s Memory bandwidth per chip sustained
- Scalability up to 32 Sockets
  - 360GB/s SMP bandwidth/chip
  - 20,000 coherent operations in flight
- Advanced pre-fetching Data and Instruction
- Binary Compatibility with POWER6



Source: IBM ©

Intel Westmere (Gulftown)



6-Core Server and 2-Core Mainstream

Use additional transistor budget to add new capabilities

– New instructions for accelerating encryption/decryption algorithms

Independent voltage and frequency per domain

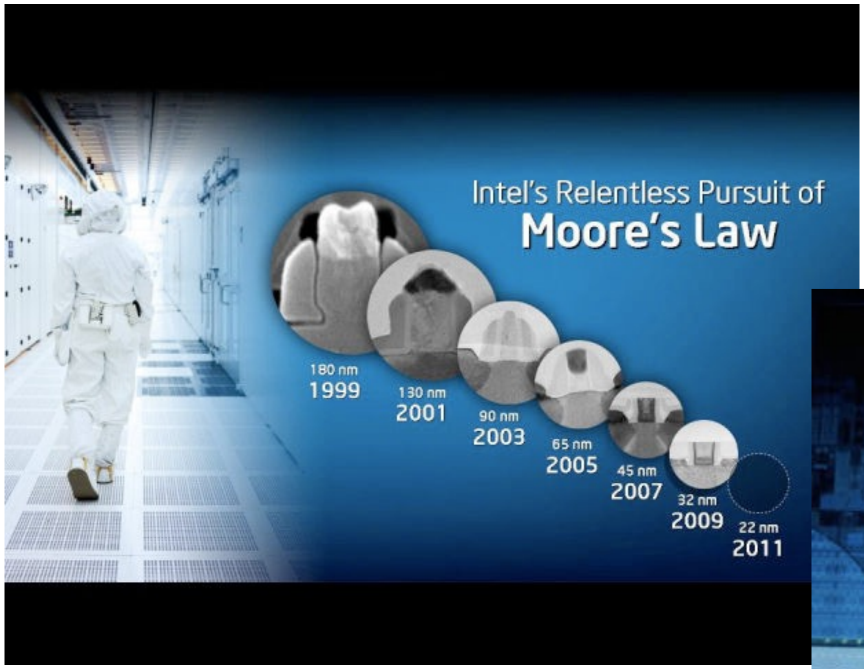
Core and Uncore power gating

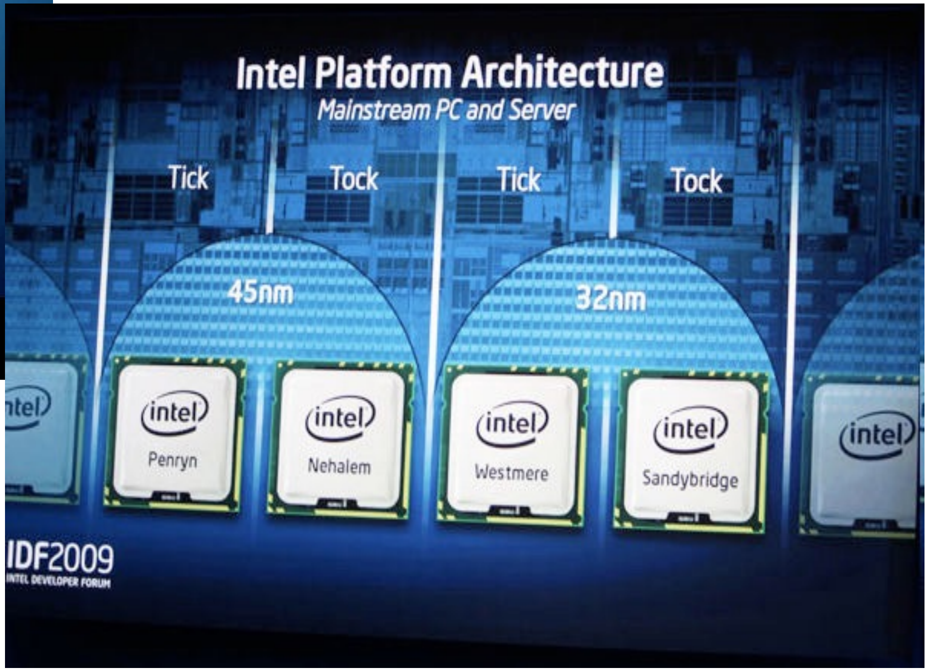
Source: Intel ©

2010 – New Systems

- AMD 12-core Opteron
- Sun Microsystems - Niagara 3 (or RainbowFalls)
  - 16-cores (8 or 16 threads per core?)
- Nvidia Fermi – 512 Cuda cores
- ARM – Cortex A9 at 2GHz

Looking at the future





Source: Intel ©

Papers to read

- Simultaneous multithreading: maximizing onchip parallelism. ISCA 1995.

http://doi.acm.org/10.1145/223982.224449

- The SGI Origin: a ccNUMA highly scalable server. ISCA 1997.

http://doi.acm.org/10.1145/264107.264206