ECE/CS 757: Advanced Computer Architecture II

Instructor:Mikko H Lipasti

Spring 2017

University of Wisconsin-Madison

Lecture notes based on slides created by John Shen, Mark Hill, David Wood, Guri Sohi, Jim Smith, Natalie Enright Jerger, Michel Dubois, Murali Annavaram, Per Stenström and probably others

Computer Architecture

- Instruction Set Architecture (IBM 360)
  - *… the attributes of a [computing] system as seen by the programmer. I.e. the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls, the logic design, and the physical implementation. -- Amdahl, Blaaw, & Brooks, 1964*
- Machine Organization (microarchitecture)
  - ALUS, Buses, Caches, Memories, etc.
- Machine Implementation (realization)
  - Gates, cells, transistors, wires

757 In Context

- Prior courses
  - 352 – gates up to multiplexors and adders
  - 354 – high-level language down to machine language interface or instruction set architecture (ISA)
  - 552 – implement logic that provides ISA interface
  - CS 537 – provides OS background (co-req. OK)
- This course – 757 – covers parallel machines
  - Multiprocessor systems
  - Data parallel systems
  - Memory systems that exploit MLP
  - Etc.
- Additional courses
  - ECE 752 covers advanced uniprocessor design (not a prereq)
    - Will review key topics in next lecture
  - ECE 755 covers VLSI design
  - ME/ECE 759 covers parallel programming
  - CS 758 covers special topics (recently parallel programming)

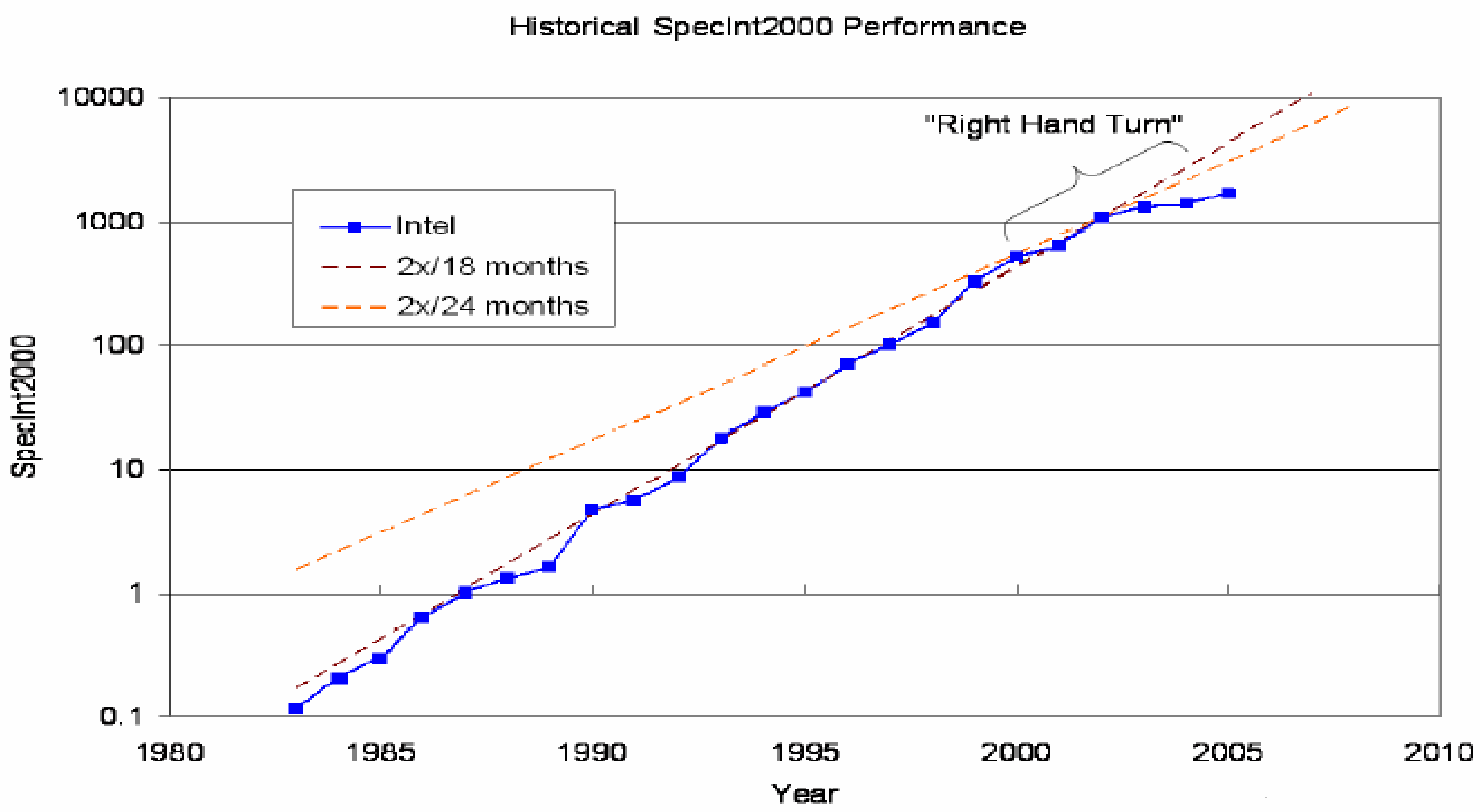
Why Take 757?

- To become a computer designer
  - Alumni of this class helped design your computer
- To learn what is *under the hood* of a computer
  - Innate curiosity
  - To better understand when things break
  - To write better code/applications
  - To write better system software (O/S, compiler, etc.)
- Because it is intellectually fascinating!
- Because multicore/parallel systems are ubiquitous

Computer Architecture

- Exercise in engineering tradeoff analysis
  - Find the fastest/cheapest/power-efficient/etc. solution
  - Optimization problem with 100s of variables
- All the variables are changing
  - At non-uniform rates
  - With inflection points
  - Only one guarantee: Today's right answer will be wrong tomorrow
- Two high-level effects:
  - Technology push
  - Application Pull

Trends



- Moore’s Law for device integration
- Chip power consumption
- Single-thread performance trend

[source: Intel]

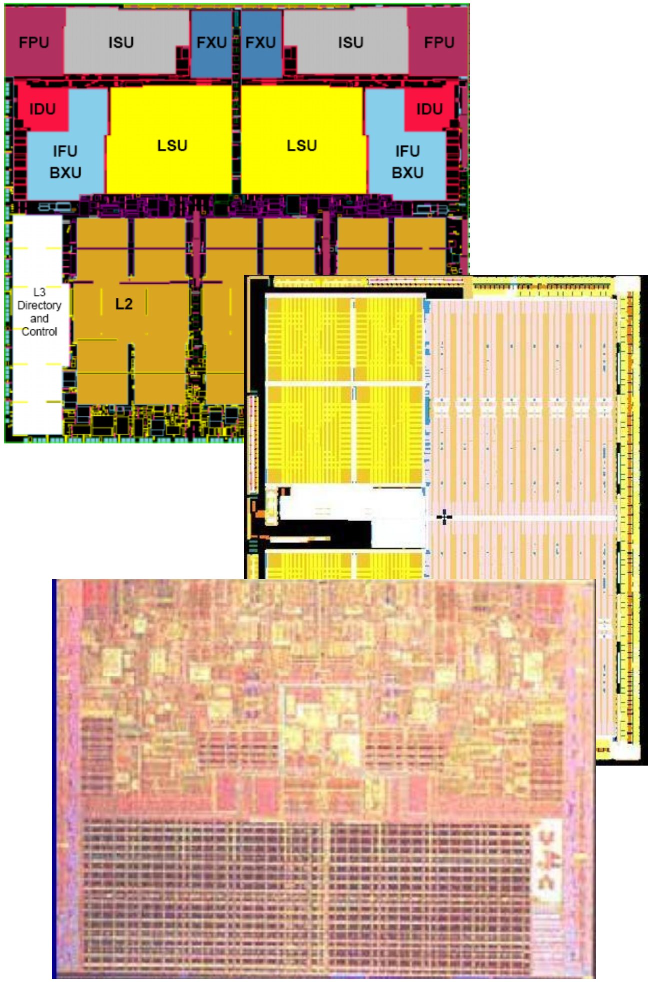
Dynamic Power

$$P_{dyn} \approx \sum_{i \in units} k \overset{\Uparrow}{C_i} \overset{\Downarrow}{V^2} \overset{\Uparrow}{A_i} \overset{\Uparrow}{f}$$

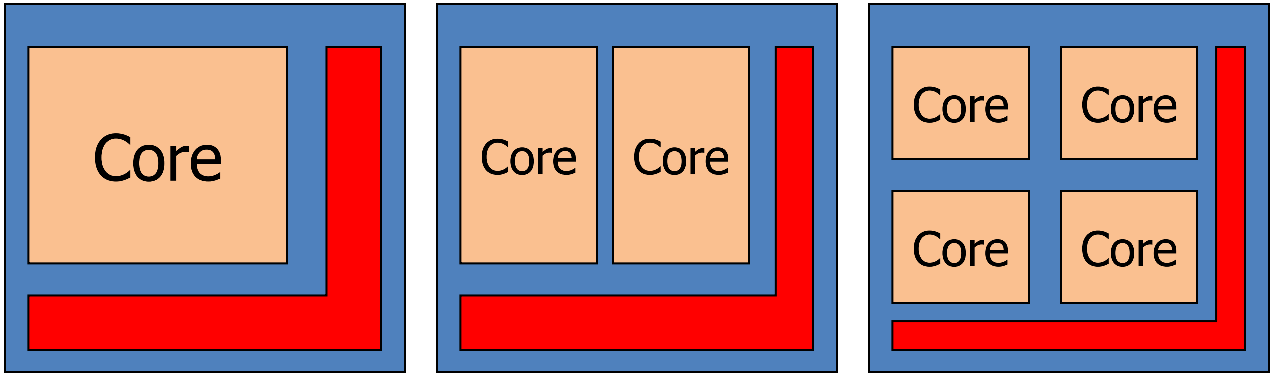
- Static CMOS: current flows when active
  - Combinational logic evaluates new inputs
  - Flip-flop, latch captures new value (clock edge)
- Terms
  - C: capacitance of circuit
    - wire length, number and size of transistors
  - V: supply voltage
  - A: activity factor
  - f: frequency
- Future: Fundamentally power-constrained

Multicore Mania

- First, servers
  - IBM Power4, 2001
- Then desktops
  - AMD Athlon X2, 2005
- Then laptops
  - Intel Core Duo, 2006
- Now, cellphone & tablet
  - Qualcomm, Nvidia Tegra, Apple A6, etc.

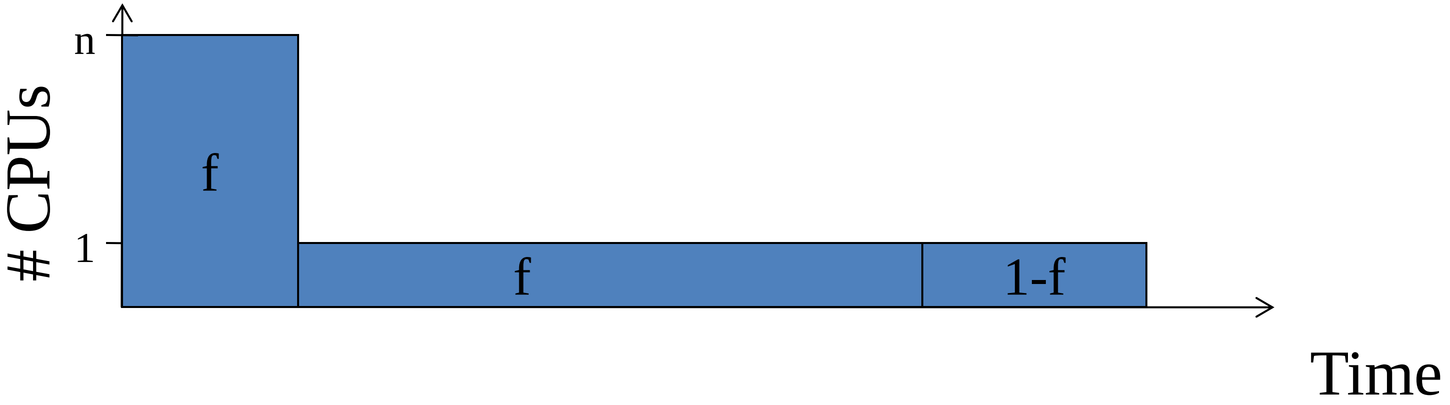


Why Multicore



	Single Core	Dual Core	Quad Core
Core area	A	~A/2	~A/4
Core power	W	~W/2	~W/4
Chip power	W + O	W + O’	W + O’’
Core performance	P	0.9P	0.8P
Chip performance	P	1.8P	3.2P

Amdahl’s Law



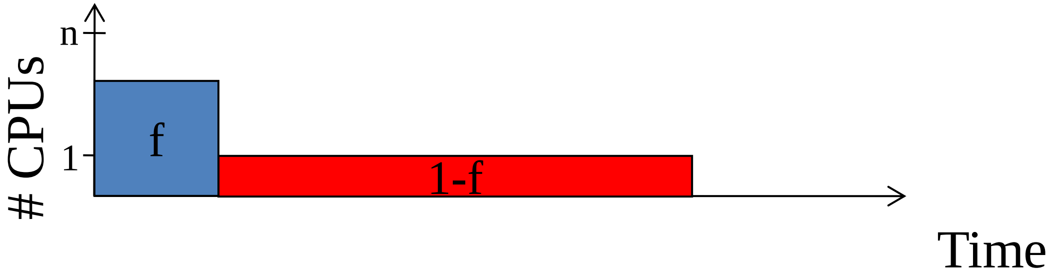
f – fraction that can run in parallel

1-f – fraction that must run serially

$$Speedup = \frac{1}{(1-f)+\frac{f}{n}}$$

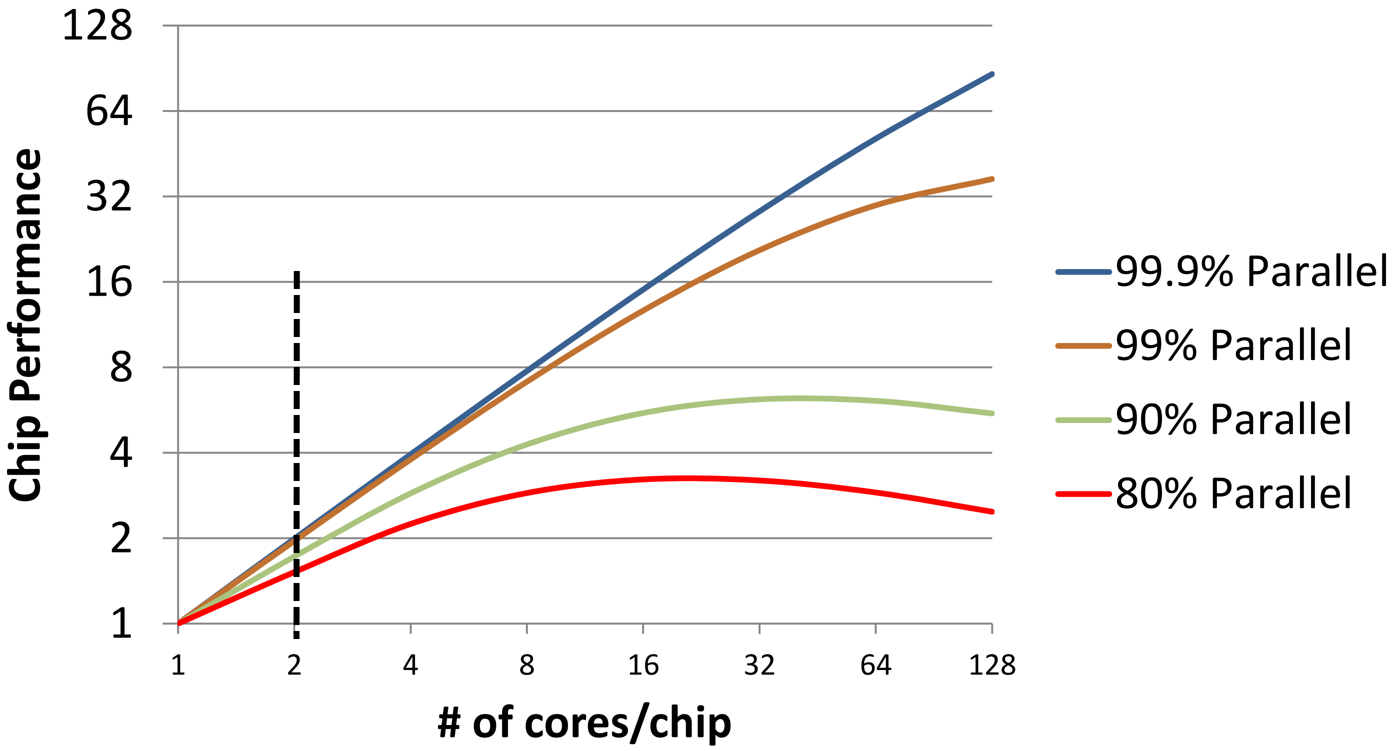
$$\lim_{n\to\infty}\frac{1}{1-f+\frac{f}{n}} = \frac{1}{1-f}$$

Fixed Chip Power Budget



- Amdahl's Law
  - Ignores (power) cost of n cores
- Revised Amdahl's Law
  - More cores → each core is slower
  - Parallel speedup < n
  - Serial portion (1-f) takes longer
  - Also, interconnect and scaling overhead

Fixed Power Scaling



- Fixed power budget forces slow cores
- Serial code quickly dominates

Challenges

- Parallel scaling limits *many-core*
  - >4 cores only for well-behaved programs
  - Optimistic about *new* applications
- Interconnect overhead
- Single-thread performance
  - Will degrade unless we innovate
- Parallel programming
  - Express/extract parallelism in new ways
  - Retrain programming workforce

Finding Parallelism

1. Functional parallelism
   - Car: {engine, brakes, entertain, nav, …}
   - Game: {physics, logic, UI, render, …}
2. Automatic extraction
   - Decompose serial programs
3. Data parallelism
   - Vector, matrix, db table, pixels, …
4. Request parallelism
   - Web, shared database, telephony, …

Balancing Work

- Amdahl’s parallel phase f: all cores busy
- If not perfectly balanced
  - (1-f) term grows (f not fully parallel)
  - Performance scaling suffers
- Manageable for data & request parallel apps
- Very difficult problem for other two:
  - Functional parallelism
  - Automatically extracted

Coordinating Work

- Synchronization
  - Some data somewhere is shared
  - Coordinate/order updates and reads
  - Otherwise → chaos
- Traditionally: locks and mutual exclusion
  - Hard to get right, even harder to tune for perf.
- Research to reality: Transactional Memory
  - Programmer: Declare potential conflict
  - Hardware and/or software: speculate & check
  - Commit or roll back and retry
  - IBM, Intel, others, now support in HW

Single-thread Performance

- Still most attractive source of performance
  - Speeds up parallel and serial phases
  - Can use it to buy back power
- Must focus on *power consumption*
  - Performance benefit ≥ Power cost
- Focus of 752; brief review coming up

Focus of this Course

- How to minimize these overheads
  - Interconnect
  - Synchronization
  - Cache Coherence
  - Memory systems
- Also
  - How to write parallel programs (a little)
  - Non-cache coherent systems (clusters, MPP)
  - Data-parallel systems

Expected Background

- ECE/CS 552 or equivalent
  - Design simple uniprocessor
  - Simple instruction sets
  - Organization
  - Datapath design
  - Hardwired/microprogrammed control
  - Simple pipelining
  - Basic caches
  - Some 752 content (optional review)
- High-level programming experience
  - C/UNIX skills – modify simulators

About This Course

- Readings
  - Posted on website later this week
  - Make sure you keep up with these! Often discussed in depth in lecture, with required participation
  - Subset of papers must be reviewed in writing, submitted through learn@uw
- Lecture
  - Attendance required, pop quizzes
- Homeworks
  - Not collected, for your benefit only
  - Develop deeper understanding, prepare for midterms

About This Course

- Exams
  - Midterm 1: Friday 3/3 in class
  - Midterm 2: Monday 4/17 in class
  - Keep up with reading list!
- Textbook
  - For reference:
    - Dubois, Annavaram, Stenström, Parallel Computer Organization and Design, Cambridge Univ. Press, 2012.
    - 4 beta chapters from Jim Smith posted on course web site
  - Additional references available as well
    - Morgan Kauffman synthesis lectures (UW access only)

About This Course

- Course Project
  - Research project
    - Replicate results from a paper
    - Or attempt something novel
    - Parallelize/characterize new application
  - Proposal due 3/17, status report 4/21
- Final project includes a written report and an oral presentation
  - Written reports due 5/8
  - Presentations during class time 5/1, 5/3

About This Course

- Grading

– Homework, quizzes, paper reviews	20%
– Midterm 1	25%
– Midterm 2	25%
– Project	30%

- Web Page (check regularly)
  - http://ece757.ece.wisc.edu

About This Course

- Office Hours
  - Prof. Lipasti: EH 3621, TBD, or by appt.
- Communication channels
  - E-mail to instructor, class e-mail list
    - ece757-1-s17@lists.wisc.edu
  - Web page
    - http://ece757.ece.wisc.edu
  - Office hours

About This Course

- Other Resources
  - Computer Architecture Colloquium – Tuesday 4-5PM, 1240 CSS
  - Computer Engineering Seminar – Friday 12-1PM, EH4610
  - Architecture mailing list:
    - http://lists.cs.wisc.edu/mailman/listinfo/architecture
  - WWW Computer Architecture Page http://www.cs.wisc.edu/~arch/www

About This Course

- Lecture schedule:
  - MWF 1-2:15pm
  - Cancel 1 of 3 lectures (on average)
  - Free up several weeks near end for project work

Tentative Schedule (1st half)

Week	Dates	Topic
1	1/18, 1/20	Introduction 752 review
2	**1/23, 1/25** 1/27	**Class cancelled** Cores, multithreading, multicore
3	1/30, 2/1, 2/3	MP Software Memory Systems
4	**2/6, 2/8** 2/10	**Class cancelled** MP Memory Systems
5	2/13, 2/15, 2/17	Coherence & consistency
6	2/20, 2/22, 2/24	Coherence & consistency cont'd
7	2/27, 3/1 3/3	Catch up / midterm review **Midterm 1 in class on 3/3**
8	**3/6** 3/8, 3/10	**Class cancelled** Simulation Methodology Transactional Memory
9	3/13, 3/15, 3/17	Interconnection Networks **Project proposal due 3/17**
N/A	3/20, 3/22, 3/24	Spring break

Tentative Schedule (2nd half)

Week	**Dates**	**Topic**
10	3/27, 3/29, 3/31	SIMD MPP
11	4/3, 4/5 **4/7**	Clusters, GPGPUs **Class cancelled**
12	4/10, 4/12, 4/14	Catch up and review
13	4/17 **4/19, 4/21**	**Midterm 2 in class 4/17** **No lecture; project work** **Project status report due 4/21**
14	**4/24, 4/26, 4/28**	**No lecture; project work**
15	5/1, 5/3	Project talks, course Evaluation
16	5/8	**No final exam** **Project reports due 5/8**

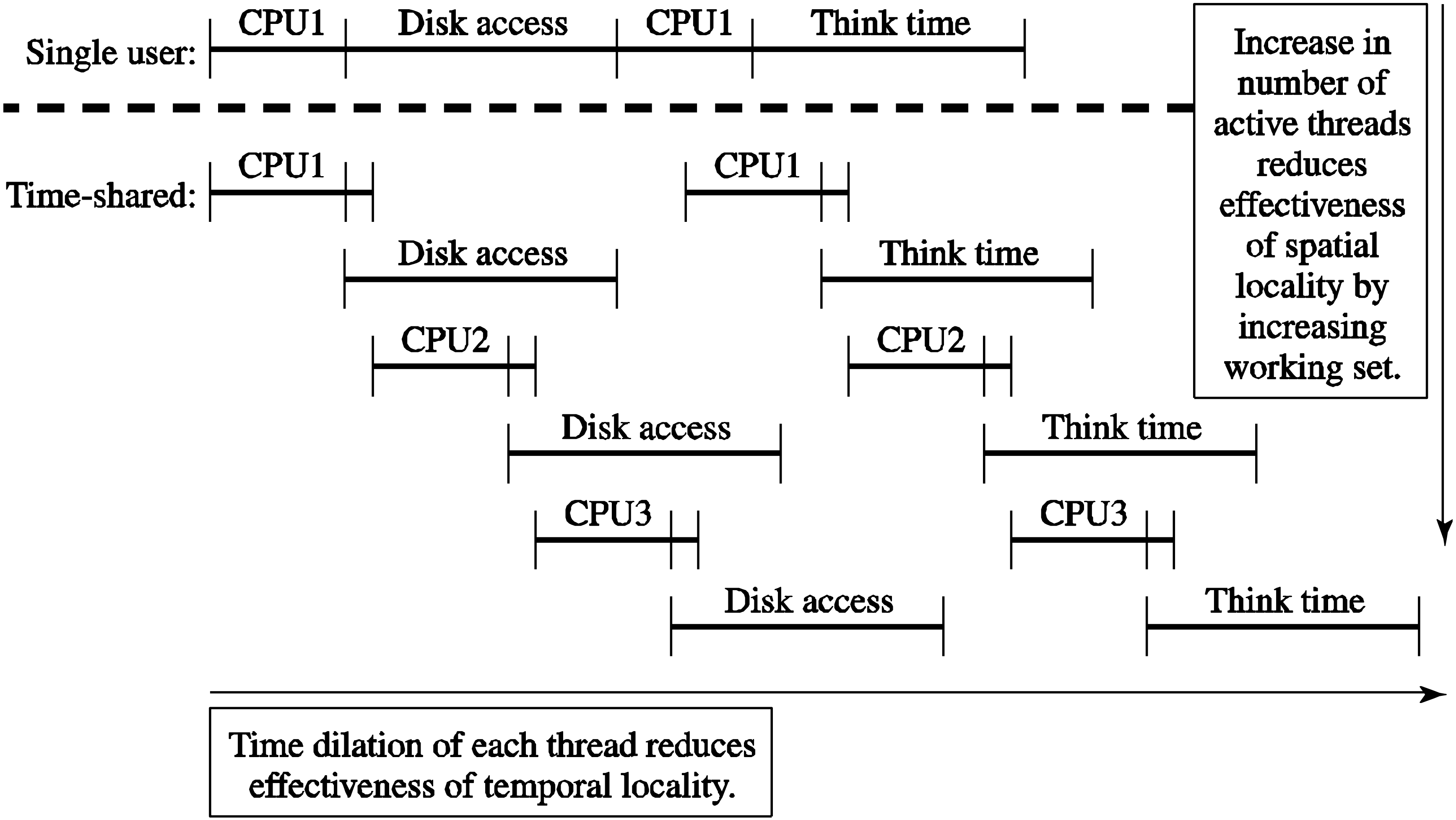
Brief Introduction to Parallel Computing

- Thread-level parallelism
- Multiprocessor Systems
- Cache Coherence
  - Snoopy
  - Scalable
- Flynn Taxonomy
- UMA vs. NUMA

Thread-level Parallelism

- Instruction-level parallelism (752 focus)
  - Reaps performance by finding independent work in a single thread
- Thread-level parallelism
  - Reaps performance by finding independent work across multiple threads
- Historically, requires explicitly parallel workloads
  - Originates from mainframe time-sharing workloads
  - Even then, CPU speed >> I/O speed
  - Had to overlap I/O latency with “something else” for the CPU to do
  - Hence, operating system would schedule other tasks/processes/threads that were “time-sharing” the CPU

Thread-level Parallelism



Increase in number of active threads reduces effectiveness of spatial locality by increasing working set.

Time dilation of each thread reduces effectiveness of temporal locality.

- Reduces effectiveness of temporal and spatial locality

Thread-level Parallelism

- Initially motivated by time-sharing of single CPU
  - OS, applications written to be multithreaded
- Quickly led to adoption of multiple CPUs in a single system
  - Enabled scalable product line from entry-level single-CPU systems to high-end multiple-CPU systems
  - Same applications, OS, run seamlessly
  - Adding CPUs increases throughput (performance)
- More recently:
  - Multiple threads per processor core
    - Coarse-grained multithreading (aka “switch-on-event”)
    - Fine-grained multithreading
    - Simultaneous multithreading
  - Multiple processor cores per die
    - Chip multiprocessors (CMP)

Multiprocessor Systems

- Primary focus on shared-memory symmetric multiprocessors
  - Many other types of parallel processor systems have been proposed and built
  - Key attributes are:
    - Shared memory: all physical memory is accessible to all CPUs
    - Symmetric processors: all CPUs are alike
  - Other parallel processors may:
    - Share some memory, share disks, share nothing
    - Have asymmetric processing units
- Shared memory idealisms
  - Fully shared memory
  - Unit latency
  - Lack of contention
  - Instantaneous propagation of writes

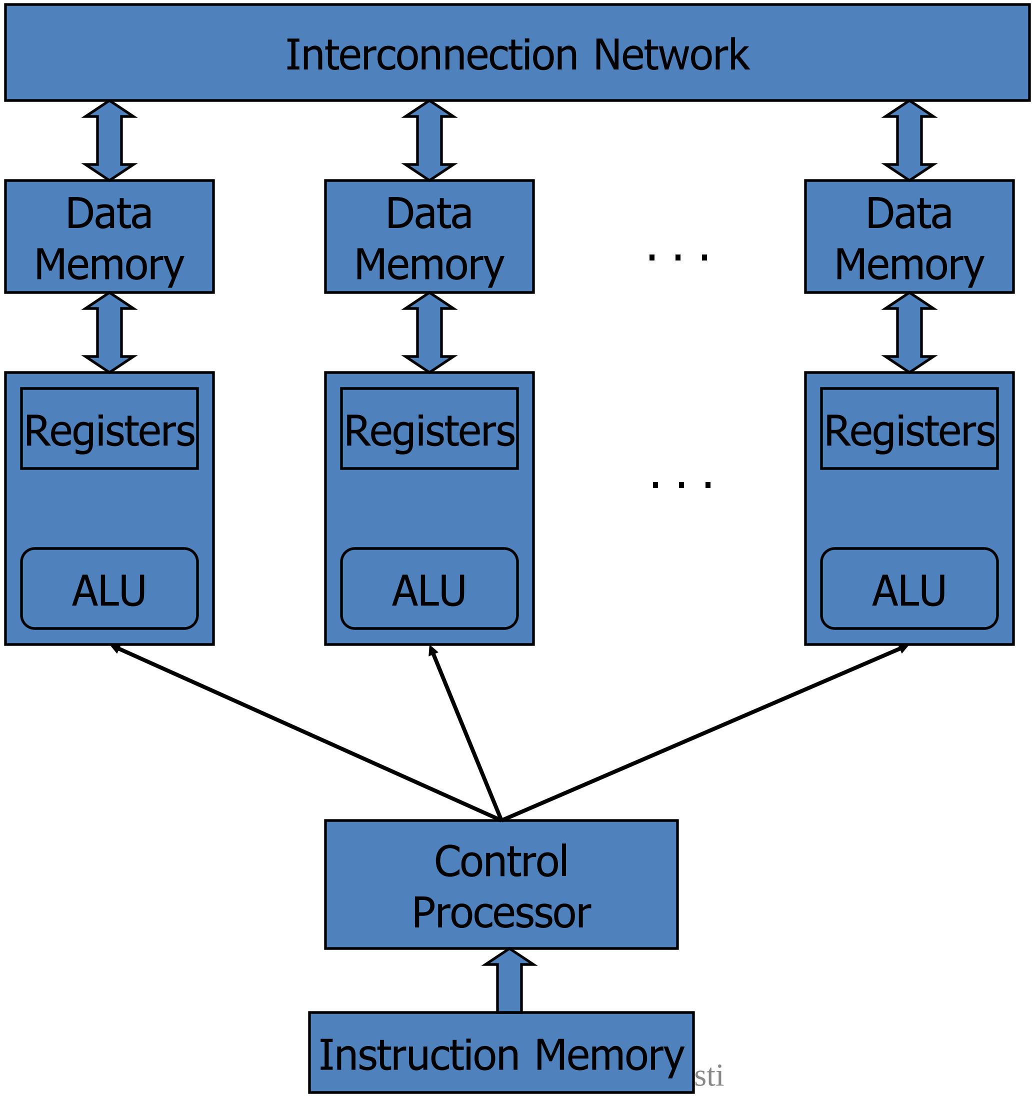
Motivation

- So far: one processor in a system
- Why not use N processors
  - Higher throughput via parallel jobs
  - Cost-effective
    - Adding 3 CPUs may get 4x throughput at only 2x cost
  - Lower latency from multithreaded applications
    - Software vendor has done the work for you
    - E.g. database, web server
  - Lower latency through parallelized applications
    - Much harder than it sounds

Where to Connect Processors?

- At processor?
  - Single-instruction multiple data (SIMD)
- At I/O system?
  - Clusters or multicomputers
- At memory system?
  - Shared memory multiprocessors
  - Focus on Symmetric Multiprocessors (SMP)

Connect at Processor (SIMD)



sti

Connect at Processor

- SIMD Assessment
  - Amortizes cost of control unit over many datapaths
  - Enables efficient, wide datapaths
  - Programming model has limited flexibility
    - Regular control flow, data access patterns
- SIMD widely employed today
  - MMX, SSE, 3DNOW vector extensions
  - GPUs from Nvidia and AMD

Connect at I/O

- Connect with standard network (e.g. Ethernet)
  - Called a cluster
  - Adequate bandwidth (GB Ethernet, going to 10GB)
  - Latency very high
  - Cheap, but “get what you pay for”
- Connect with custom network (e.g. IBM SP1,SP2, SP3)
  - Sometimes called a multicomputer
  - Higher cost than cluster
  - Poorer communication than multiprocessor
- Internet data centers built this way

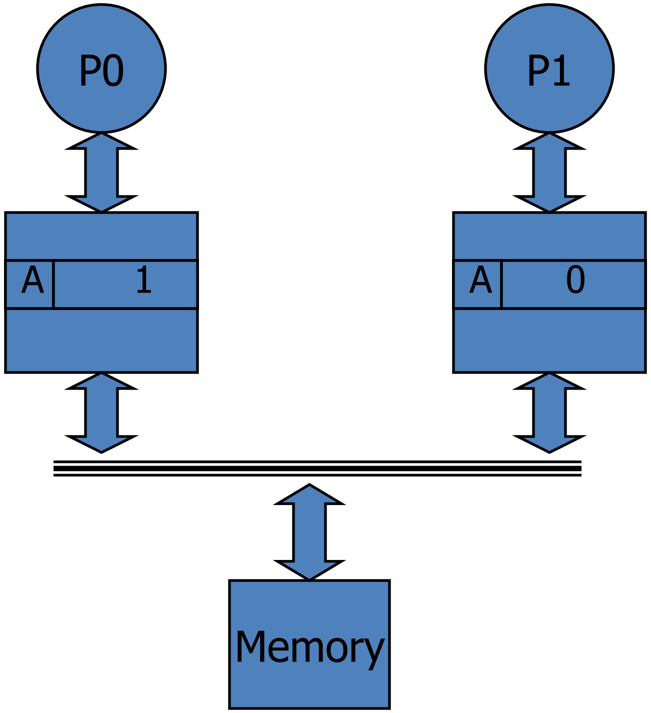
Connect at Memory: Multiprocessors

- Shared Memory Multiprocessors
  - All processors can address all physical memory
  - Demands evolutionary operating systems changes
  - Higher throughput with no application changes
  - Low latency, but requires parallelization with proper synchronization
- Most successful: Symmetric MP or SMP
  - 2-64 microprocessors on a “bus”
  - Still use cache memories

Cache Coherence Problem

Load A

Store A<= 1



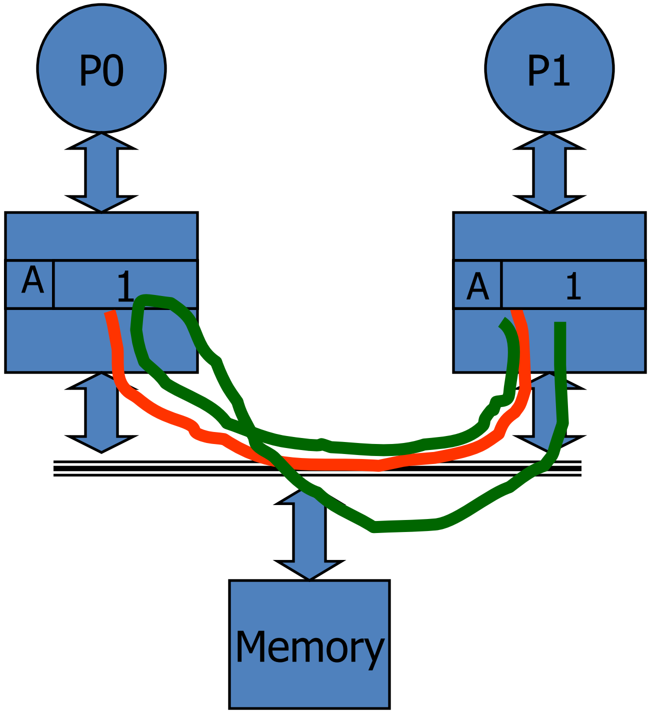
Load A

Load A

Cache Coherence Problem

Load A

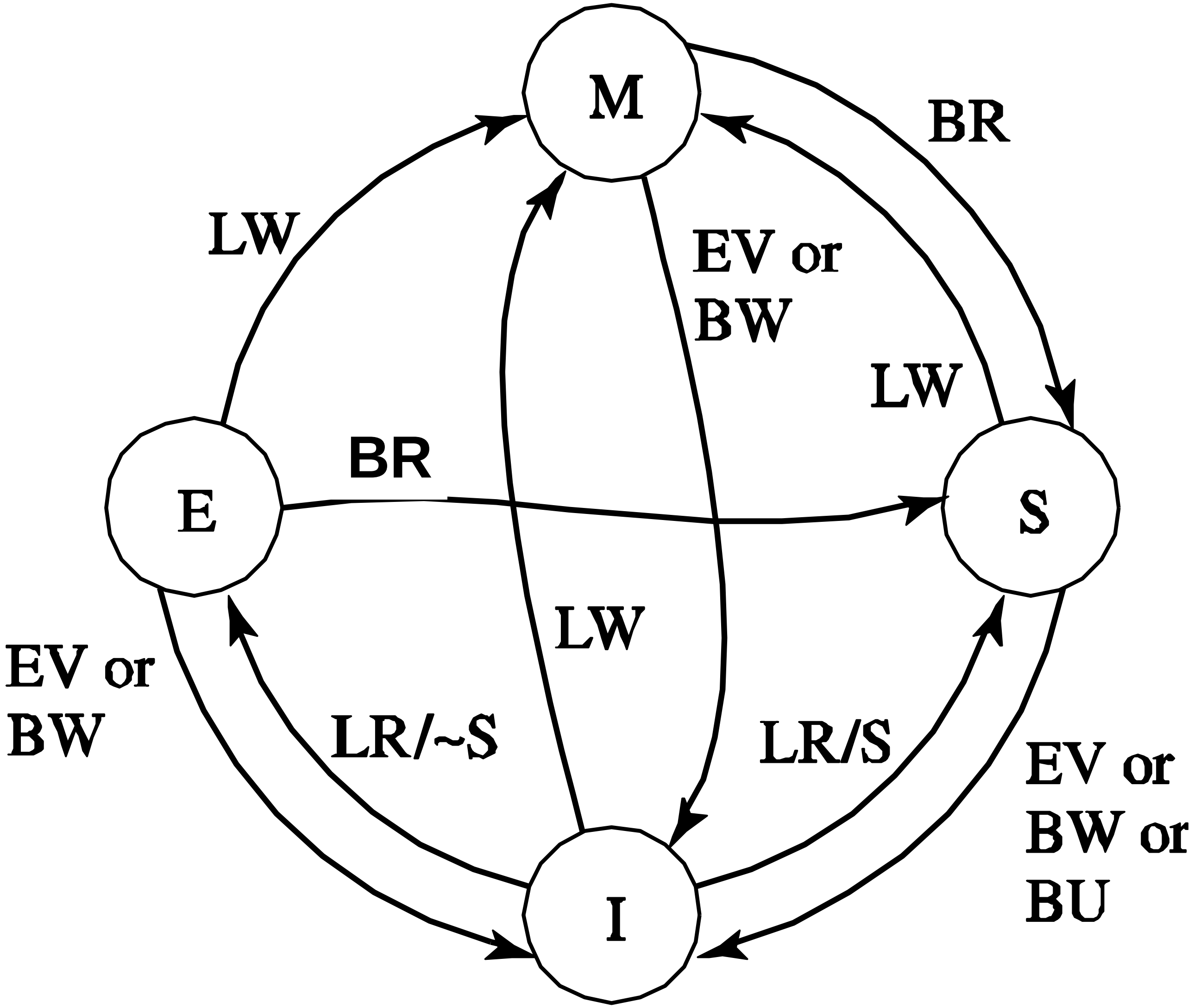
Store A<= 1



Load A

Load A

Sample Invalidate Protocol (MESI)



Sample Invalidate Protocol (MESI)

Current State s	**Event and Local Coherence Controller Responses and Actions (s' refers to next state)**					
	Local Read (LR)	**Local Write (LW)**	**Local Eviction (EV)**	**Bus Read (BR)**	**Bus Write (BW)**	**Bus Upgrade (BU)**
Invalid (I)	Issue bus read if no sharers then s' = E else s' = S	Issue bus write s' = M	s' = I	Do nothing	Do nothing	Do nothing
Shared (S)	Do nothing	Issue bus upgrade s' = M	s' = I	Respond shared	s' = I	s' = I
Exclusive (E)	Do nothing	s' = M	s' = I	Respond shared s' = S	s' = I	Error
Modified (M)	Do nothing	Do nothing	Write data back; s' = I	Respond dirty; Write data back; s' = S	Respond dirty; Write data back; s' = I	Error

Snoopy Cache Coherence

- All requests broadcast on bus
- All processors and memory snoop and respond
- Cache blocks writeable at one processor or read-only at several
  - Single-writer protocol
- Snoops that hit dirty lines?
  - Flush modified data out of cache
  - Either write back to memory, then satisfy remote miss from memory, or
  - Provide dirty data directly to requestor
  - Big problem in MP systems
    - Dirty/coherence/sharing misses

Scaleable Cache Coherence

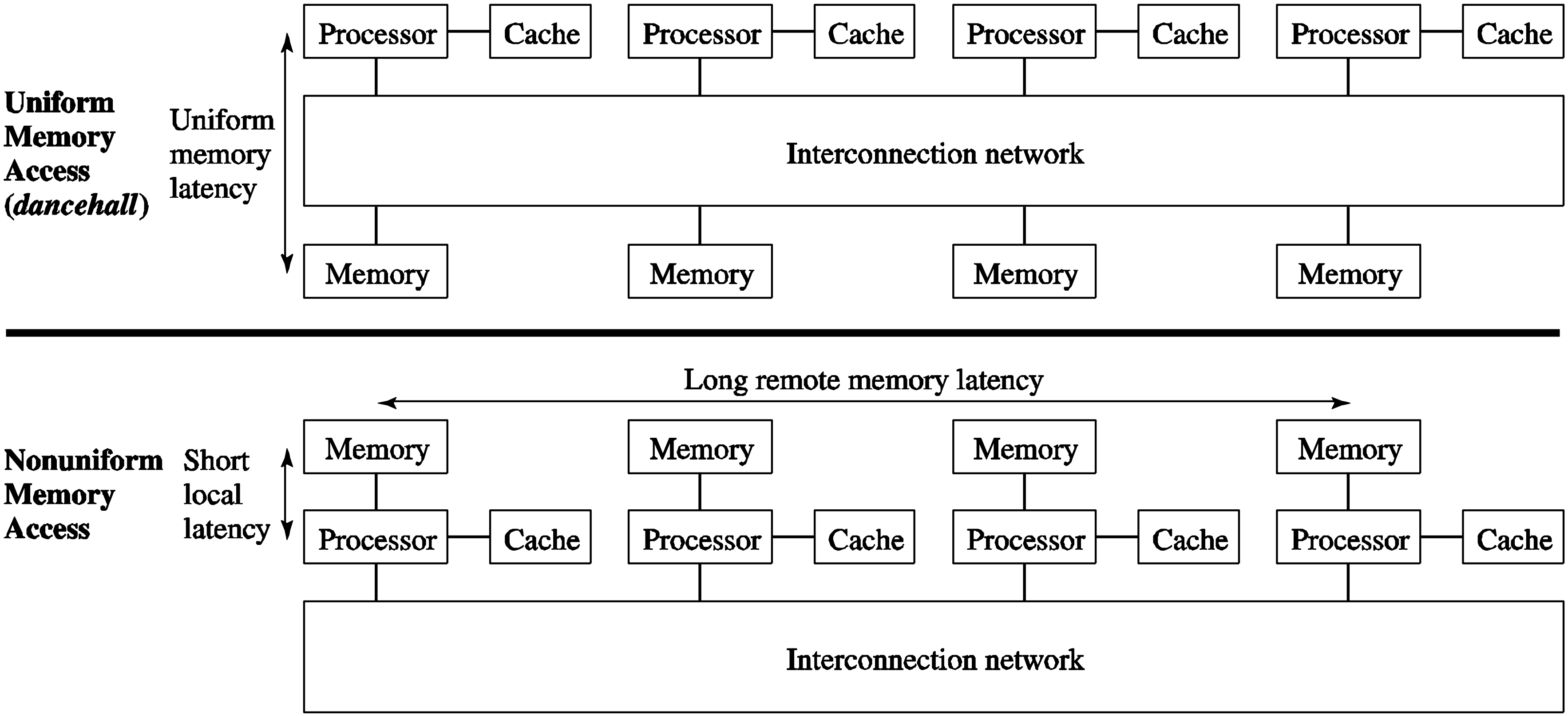
- Eschew physical bus but still snoop
  - Point-to-point tree structure
  - Root of tree provides ordering point
- Or, use level of indirection through directory
  - Directory at memory remembers:
    - Which processor is “single writer”
      - Forwards requests to it
    - Which processors are “shared readers”
      - Forwards write permission requests to them
  - Level of indirection has a price
    - Dirty misses require 3 hops instead of two
      - Snoop: Requestor->Owner->Requestor
      - Directory: Requestor->Directory->Owner->Requestor

Flynn Taxonomy

Flynn (1966)	Single Data	Multiple Data
Single Instruction	SISD	SIMD
Multiple Instruction	MISD	MIMD

- MISD
  - Fault tolerance
  - Pipeline processing/streaming or systolic array
- Now extended to SPMD
  - single program multiple data

Memory Organization: UMA vs. NUMA



Memory Taxonomy

For Shared Memory	Uniform Memory	Nonuniform Memory
Cache Coherence	CC-UMA	CC-NUMA
No Cache Coherence	NCC-UMA	NCC-NUMA

- NUMA wins out for practical implementation
- Cache coherence favors programmer
  - Common in general-purpose systems
- NCC widespread in *scalable* systems
  - CC overhead is too high, not always necessary

Example Commercial Systems

- CC-UMA (SMP)
  - Sun E10000: http://doi.ieeecomputersociety.org/10.1109/40.653032
- CC-NUMA
  - SGI Origin 2000: The SGI Origin: A ccnuma Highly Scalable Server
- NCC-NUMA
  - Cray T3E: http://www.cs.wisc.edu/~markhill/Misc/asplos96_t3e_comm.pdf
- Clusters
  - ASCI: https://str.llnl.gov/str/April05/Seager.html

Summary

- Thread-level parallelism
- Multiprocessor Systems
- Cache Coherence
  - Snoopy
  - Scalable
- Flynn Taxonomy
- UMA vs. NUMA