CMPE436: Concurrent and Distributed Programming
Lecture 1

Alper Şen

Today's Class

- Course Syllabus
- Sequential and Concurrent Programs
- Concurrent and Distributed Systems
- Concurrent Programming

CMPE436: Concurrent and Distributed Programming

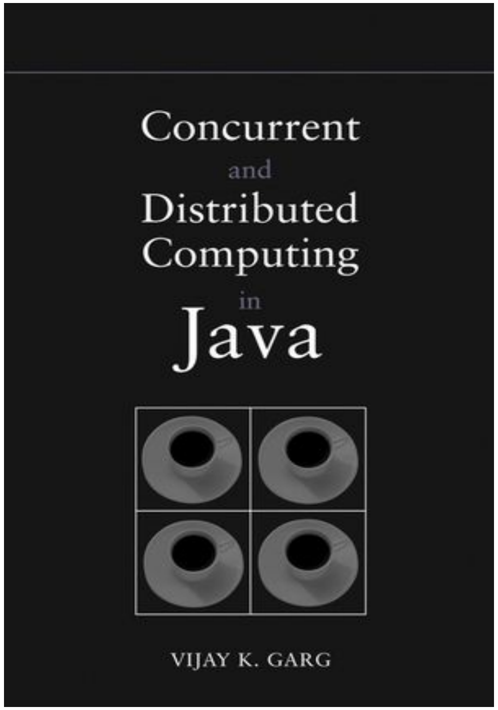
- Fall 2017, TThTh 578

- **Instructor:** Assoc. Prof. Alper Sen
- **Email:** alper.sen@boun.edu.tr
- **Office Hours:** After class

- **Course Web:**

Course Syllabus

- **Reference Textbook:** Concurrent and Distributed Computing in Java by Vijay K. Garg, Wiley & Sons, 2004. http://users.ece.utexas.edu/~garg/jbk.html



- **Prerequisites:** Programming skills. Background in Algorithms or Operating Systems will be helpful, but is not essential. See me if you have concerns!

Course Syllabus

- **Course Organization:** instructor lectures, assignments, term project and exams.

- **Tentative Grading:** 20% Assignments, 30% Exam, 25% Term project and presentation, 20% Final Exam, 5% participation + attendance.

- **Right to take final exam:** Midterm grade > 20, Term project and presentation grade > 50

- **Term Project:** Major component of the course.
  - **e.g.** Android based concurrent applications.

Course Syllabus

- **Assignments:** Programming in Java as well as non-programming assignments.
  - Students must submit their assignments through email only.
  - Assignments are due at their specified date and time. We will reduce assignment grades by 25 points (out of 100) for each day that they are late.

  - No cheating allowed. Programs will be checked for similarity, including programs from previous years.

Course Goals

- Learn fundamentals of concurrent and distributed systems
- Appreciation of the problems of concurrent/distributed programming
  - Classic synchronization problems and solutions
- Develop practical skills writing concurrent/distributed programs

Concurrent Programming Topics

- (Chap 1) Process, thread, race conditions
- (Chap 2) Mutual Exclusion: Software solutions
- (Chap 2) Hardware solutions
- (Chap 3) Synchronization Primitives: Semaphores
- (Chap 3) Monitors
- Deadlocks
- Data Race Detection
- Verifying Concurrent Programs (Model Checking, Promela)

Distributed Programming Topics

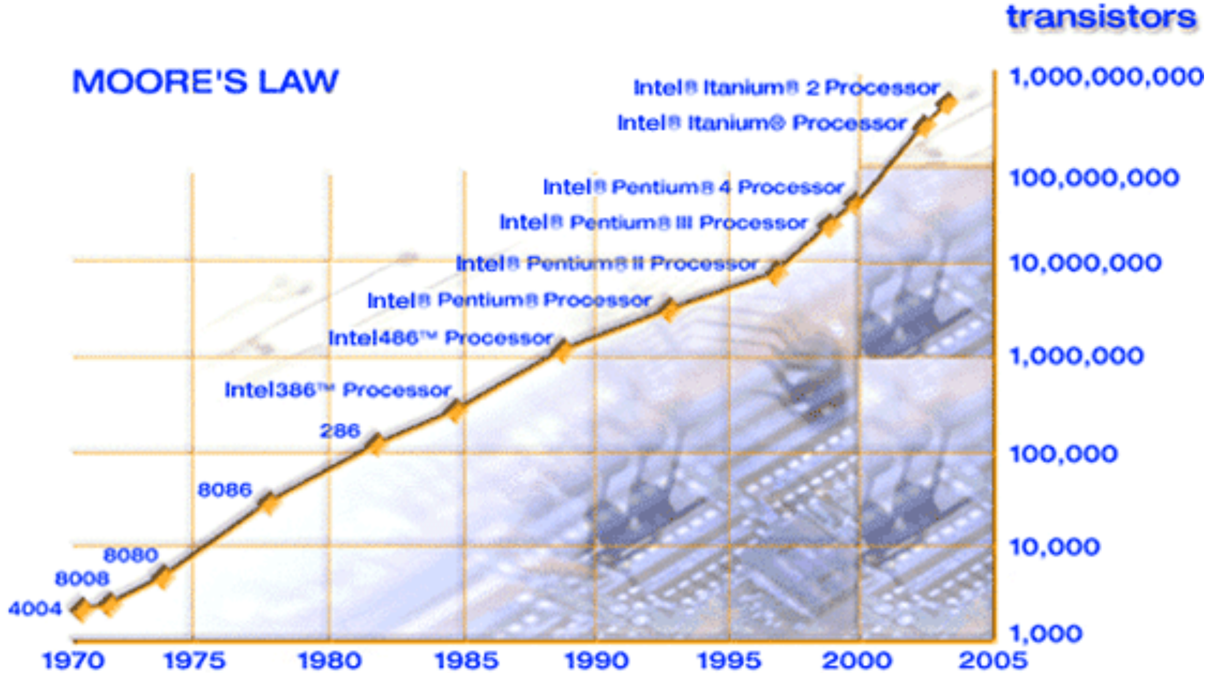
- (Chap 6) Sockets, Java RMI
- (Chap 7) Models, Logical Clocks
- (Chap 8) Resource Allocation: Mutual Exclusion, Dining Philosophers
- (Chap 9) Global Snapshot
- (Chap 10) Global Property Detection
- (Chap 11) Detecting Termination and Deadlocks

Questions?

- Your feedback is essential for the course.
- Feel free to ask questions.
- Attend office hours.

Concurrency in Hardware

- Moores' s Law: Number of transistors double every 18 months



- Increasing problems with heat and power consumption as clock speeds increase.

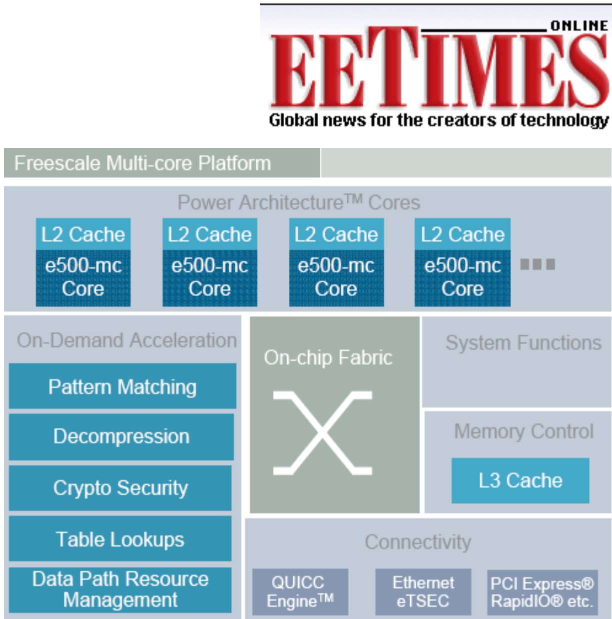
Concurrency in Hardware

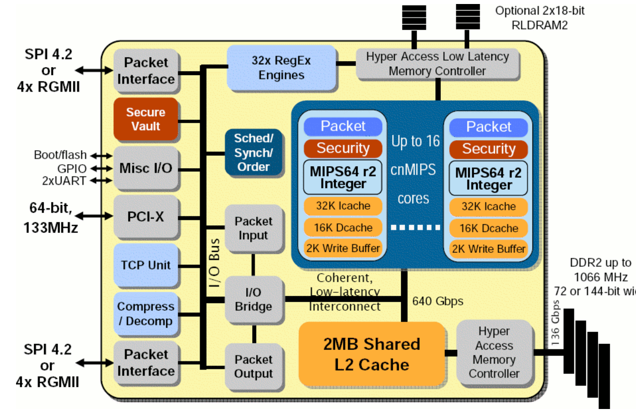
- We have lived in a sequential world relying on processor speed ups (and instruction level parallelism, pipelines etc.) to come to our rescue.
- Those days appear to be over.
- Processor vendors have turned to multi-core technology to continue the performance march.
- Software needs to catch up.

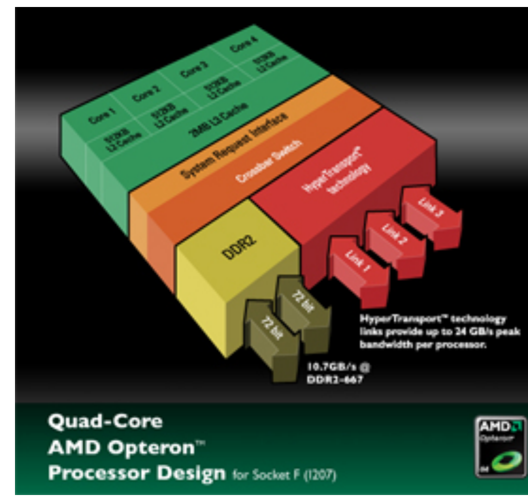
Concurrency in Hardware

- Multiple nodes (older)
  - » Clusters, Supercomputers
- Multiple threads per core (new)
  - » Simultaneous Multithreading, Hyperthreading
- Multi-processor, Multi-core Systems (new)
- The Free Lunch Is Over: Concurrent hardware demands concurrent software

Multi-cores are Here







Concurrency Examples

- Playstation 4 – now Pro
  - Effectively use 8 cores + GPU
- PCs
  - Effectively use 4 processors and a graphics adapter to generate graphics
- Multithreaded Java program on a multiprocessor system
  - Access to shared data structures
  - Synchronization between threads
- Web server (Cloud, internet applications)
  - How to serve 1000 or 10000 concurrent requests with 100 file servers: P2P
  - How to guarantee consistent reads with simultaneous writes?

Advantages of Concurrent/Distributed Programming

- **Reactive programming**
  - User can interact with applications while tasks are running.

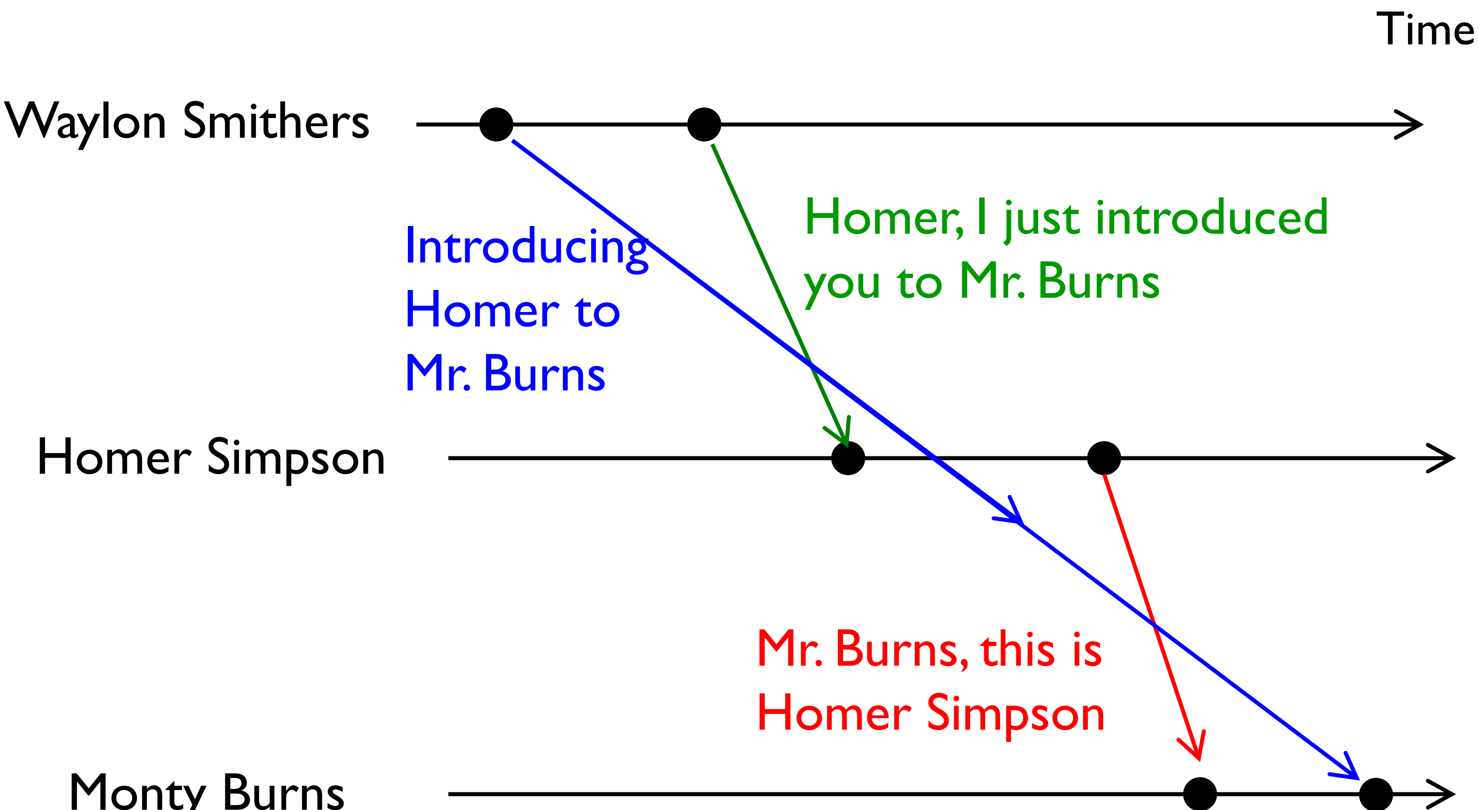
- **Availability of services**
  - Long-running tasks need not delay short-running ones, e.g., a web server can serve an entry page while at the same time processing a complex query.
- **Parallelism**
  - multiple things running at the same time
  - Complex programs can make better use of concurrent hardware in new multi-core processor architectures, SMPs, LANs or WANs,

e.g., scientific applications/algorithms, games, video etc.

Challenges of Concurrent/Distributed Programming

- **Non-determinism**
  - Mastering exponential number of behaviors due to different schedules, message delays.
- **Synchronization problems**
  - Concurrent tasks should not corrupt consistent state of program (need mutual exclusion)
  - Tasks should not suspend and indefinitely wait for each other (need progress)
- **Resource consumption**
  - Threads can be expensive. Overhead of scheduling, context-switching, and synchronization.

Synchronization Problem



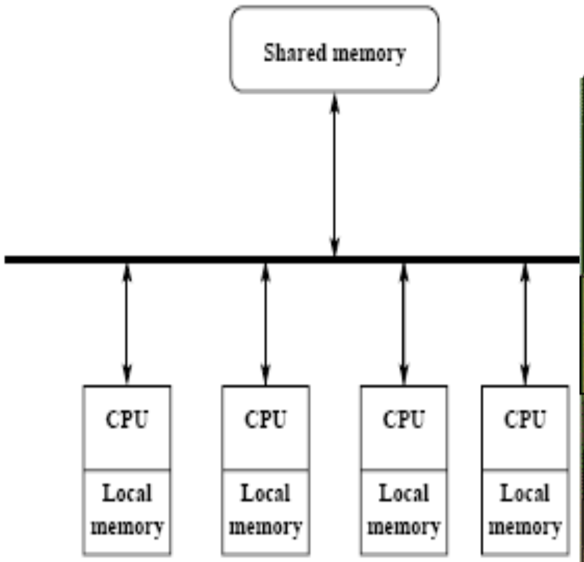
???

Simpson, eh? New man?

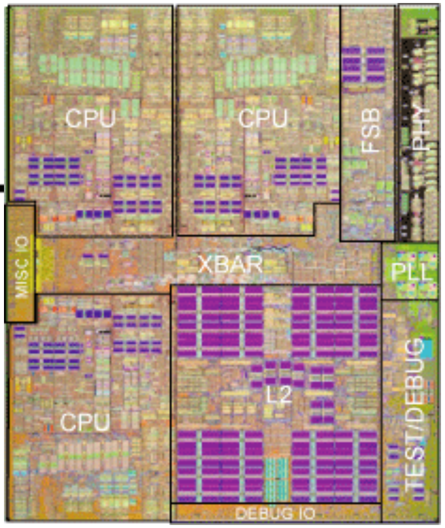
Distributed Chat Application

Concurrent Systems

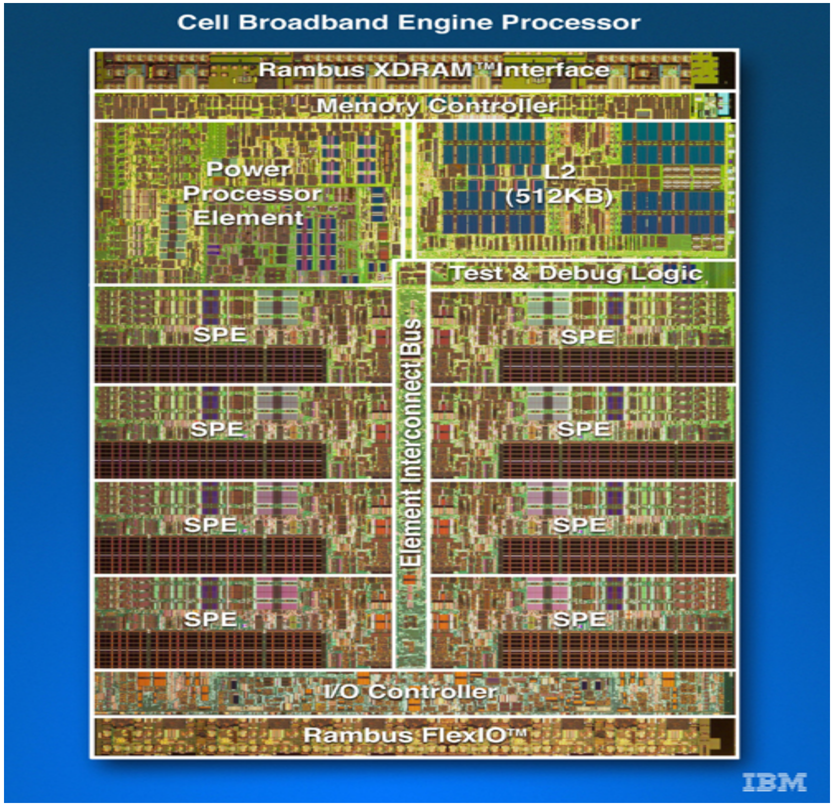
- Concurrent (Parallel) Systems: processors/multi-cores
  - Shared memory
  - Global Clock
  - Communication through shared memory



High Level View



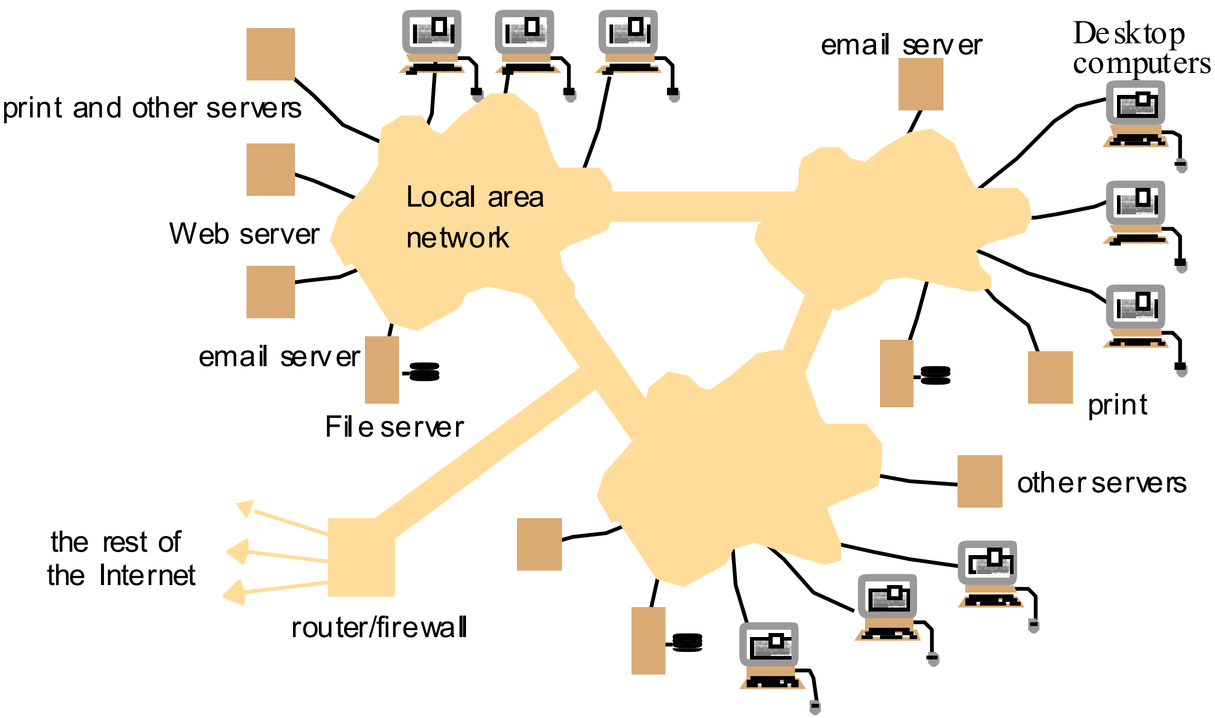
Xbox 360



PS3

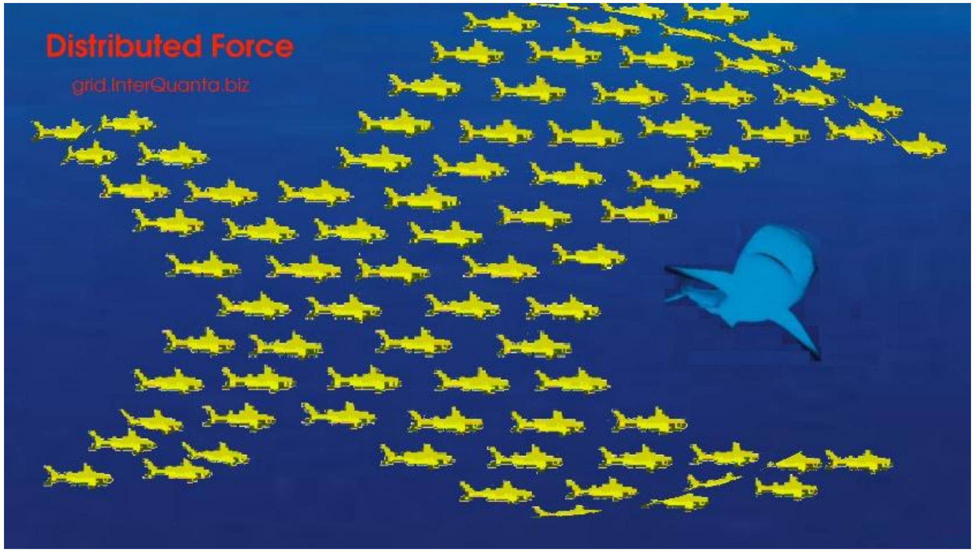
Distributed Systems

- Distributed Systems: computers + network
  - No shared memory: no computer has up-to-date knowledge about the overall system
  - No global clock: there is no common notion of time
  - Communication through messages



Concurrent versus Distributed Systems

- Concurrent Systems
  - faster due to shared memory
  - fine-grain parallelism
  - easy access
  - Java, C#, Pthreads, OpenMP, Intel TBB



- Distributed Systems
  - scalable, heterogeneous
  - data/resource sharing
  - geographic structure
  - reliable
  - Java, MPI, CORBA

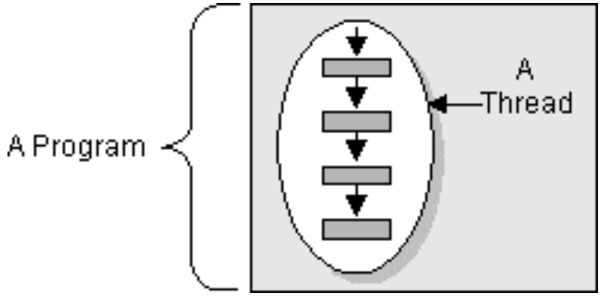
Characteristics of Concurrent and Distributed Systems

- **Tightly-coupled:** shared physical clock, synchronization overhead
- **Loosely-coupled:** no shared clock. Distributed systems use causality instead of physical clock.
- **Synchronous** versus **Asynchronous** Distributed Systems: upper-bound on communication.
  - Asynchronous systems are difficult when processors or links can fail.

Sequential versus Concurrent/Distributed Programming

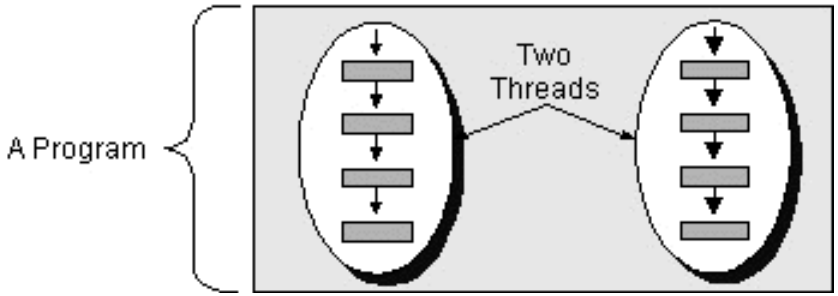
- A **computer program** is a set of instructions in a high-level or a machine-level language.

- The world runs in parallel, but our usual model of software does not.
- In a sequential program, a single stream of operations executes one instruction after another.
- For example, merge sort, matrix multiplication (could be parallelized).



Sequential versus Concurrent/Distributed Programming

- In a concurrent/distributed program, several streams of operations may execute concurrently. Each stream of operations executes as it would in a sequential program *except for the fact that streams can communicate and interfere with one another.*
- For example, a GUI application.
- Parallel matrix multiplication



- When we execute a program we get one or more processes depending if it is a sequential or concurrent program.

Processes

- **Process** is an execution context of a running program
- Process **memory** state consists of:
  - **Code**: machine instructions in memory
  - **Data**: memory used by static global vars + runtime allocated memory (heap)
  - **Stack**: local vars + function call activation records

- Communication between processes through explicit IPC mechanisms (Linux)
  - signals (send & receive ints)
  - pipes (a blocking FIFO between two processes)
  - sockets (explicit message passing, remote control of processes)

Threads

- **Threads** are lightweight processes that share address space (code + data) but have their own stack
  - if not properly synchronized, a race condition occurs.
- User threads: Thread management done by user threads library, e.g. Java JVM, Pthreads
  - Faster to create, manipulate, and synchronize
- Kernel threads: Linux, Solaris, Windows
  - Threads can exploit multiprocessors
- User threads map to Kernel threads

Context Switch

- A **context switch** is the switching of the CPU from one process or thread to another.
- For context switch; save memory state, register states, program counter etc.
- Single core: threads share the core via time-slicing
- Multiple cores: many threads run simultaneously on separate cores

Processes versus Threads

- Processes
  - share nothing, which makes them very scalable
  - hard to create
  - costly context switching
  - flexible communication

- Threads
  - all memory shared except for stack
  - easy to create
  - cheaper context switching
  - communicate through memory, must be on same machine.
  - requires properly synchronized (thread-safe) code

Race Condition

- In shared memory (concurrent) systems, a race condition occurs if two threads simultaneously access the same shared variable and one of the accesses is write.
- Interleaving of multi-thread programs generate multiple execution traces.
- Depending on the order in which the shared variable is accessed by threads, data may be corrupted or updates may be lost.

Race Condition: Example 1

Each thread executes

$x = x + 1;$

or similarly

$t = x;$
$x = t + 1;$

Initially x = 0

Expect x = 2

Trace1	
Thread1	Thread2
t1 = x;	
x = t1+1;	
	t2 = x;
	x = t2+1;
Final x = 2	

Race Condition: Example 1

Initially x = 0

Trace2	
Thread1	Thread2
t1 = x;	
	t2 = x;
x = t1+1;	
	x = t2+1;
Final x = 1 !!	

Race Condition: Example 1

Initially x = 0

Trace3	
Thread1	Thread2
t1 = x;	
	t2 = x;
	x = t2+1;
x = t1+1;	
Final x = 1 !!	

Race Condition: Example 1

x=x+1 Initially x = 0 Can have 6 different interleavings !

Trace1		Trace2		Trace3	
Thread1	Thread2	Thread1	Thread2	Thread1	Thread2
t1 = x;		*t1 = x;*		*t1 = x;*	
x = t1+1;			*t2 = x;*		*t2 = x;*
	t2 = x;	*x = t1+1;*			*x = t2+1;*
	x = t2+1;		*x = t2+1;*	*x = t1+1;*	
Final x = 2		Final x = 1		Final x = 1	

Summary

- Course Syllabus
- Sequential versus Concurrent/Distributed Programs
- Concurrent versus Distributed Systems
- Concurrent Programming:
  - Process, Thread
  - Race Conditions

For now

- Start with a Java tutorial. Available on the course web.

- Setup Eclipse IDE or IntelliJ for Java.
  http://www.eclipse.org/

- HW 1: Familiarize with sequential aspects of Java and Eclipse IDE.

- Start thinking about project topics.