Distributed Systems

- Are ubiquitous
  - share resources
  - communicate
  - increase performance (speed, fault tolerance)
- Are characterized by
  - independent activities (concurrency)
  - loosely coupled parallelism (heterogeneity)
  - inherent uncertainty
  - need for synchronization
Example I: Coordinated clubbing

Coordinate meeting in a club by sending SMS

– Only one club & one time to go

It is absolutely bad if only one party shows up

Theorem: If message delivery is not guaranteed, then coordinated clubbing cannot be achieved

Example I: Coordinated clubbing

Ping-pong execution w/o message loss

k smallest number of messages s.t.

some participant, e.g., $p_0$, decides go

Agreement $\Rightarrow$ $p_1$ also decides go

Remove last message, from $p_1$ to $p_0$

$p_1$ still decides go

$\Rightarrow$ Execution with k-1 messages!

Theorem: If message delivery is not guaranteed, then coordinated clubbing cannot be done
Uncertainty in Distributed Systems

• Uncertainty comes from
  – differing processor speeds
  – varying communication delays
  – (partial) failures

• To ensure that a system is correct (despite uncertainty)
  – identify and abstract fundamental problems
  – state problems precisely
  – design algorithms to solve problems
  – prove correctness of algorithms
  – analyze complexity of algorithms (e.g., time, space, messages)
  – prove impossibility results and lower bounds

Application Areas

Classic problems in distributed computing are from:

- multi-threaded operating systems
- communication networks
- multiprocessor architectures
- (distributed) database systems
- software fault-tolerance
Example II: Finding Primes

On an asynchronous multicore processor
- Cross off multiples of each integer
- Need to maintain a **counter** (providing an **increment** operation)
- Simple **counter++** will not work if implemented by (separate) atomic read and write, i.e.,
  
  ```
  lval = read(counter)
  lval++
  write(counter, lval)
  ```

When the following execution happens

**process A**
- lval = read(counter)
- lval++
- write(counter, lval)

**process B**
- val = read(counter)
- lval++
- write(counter, lval)

The counter grows by 1 although incremented twice

An aside: how bad is it for the primes algorithm? Sometimes need to reconsider the specification
Course Overview: Models

- Introduce two basic communication models:
  - message passing
  - shared memory
- and two basic timing models:
  - synchronous
  - asynchronous

<table>
<thead>
<tr>
<th></th>
<th>Message passing</th>
<th>Shared memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>synchronous</td>
<td></td>
<td>PRAM</td>
</tr>
<tr>
<td>asynchronous</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Course Overview

Covers the following problems:
- mutual exclusion (Chapter 4)
- fault-tolerant consensus (Chapter 5)
- causality and time (Chapter 6)
- concurrent data structures

and failure models:
- **crash**: faulty processor just stops.
  Idealization of reality.
- **Byzantine** (arbitrary): conservative assumption, fits when failure model is unknown or malicious
Course Overview: Simulations

Simplify arguments about distributed systems

**Abstractions:** making one model appear to be an easier model
- more synchrony
- more benign faults
- stronger kinds of shared variables
- distributed shared memory
- broadcast and multicast

Relationship of Theory to Practice

Operating systems / distributed systems have issues relating to (virtual) concurrency of processes, e.g.,
- mutual exclusion
- deadlock

**Multi-processor** and multi-core architectures
- no common clock ⇒ asynchronous SM model
- common clock ⇒ synchronous SM model

Loosely coupled networks, such as the Internet ⇒ asynchronous MP model
Potential Payoff of Theoretical Approach

- careful specifications clarify intent
- increased confidence in correctness
- if abstracted well then results are relevant in multiple situations
- indicate inherent limitations
  - cf. NP-completeness