Today's Objectives

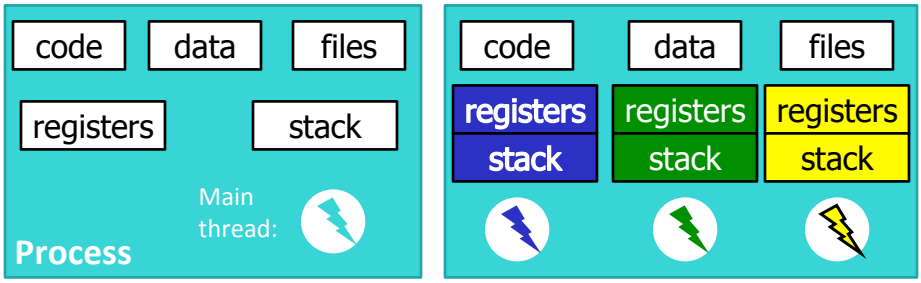
- Threads and Synchronization Wrap up
- Services

Review

- What is the goal of TCP?
  - What are some of the approaches it uses to meet these goals?
- Compare and contrast threads and processes
- What problem is caused by light-weight threads?

Review: Threads and Processes: Execution Environments

- Execution environments
  - address space (memory)
  - thread synchronization and communication resources (sockets)
  - higher-level resources likes open files
- Each process has its own separate execution environment
- Threads ***share*** a single execution environment



Threads vs. Processes

- Benefits of using threads instead of processes
  - Low communication/context switching overhead since execution environment/address space is shared
  - Easy to take advantage of parellelism multi-processor computers
- Disadvantages of threads
  - Shared resources can lead to synchronization problems
  - Need to be careful to avoid deadlocks
  - Need to provide atomic operations

Exploring Issues with Shared Data

- Shared Data: an array list (representation: Deck of Cards)
- Discussion
  - What are the problematic situations?
  - What causes them?
  - How can you prevent problematic situations?
    - Motivates synchronization mechanisms

Critical Sections and Synchronization Mechanisms

- ***Critical section***: Sections of code that have to happen uninterrupted or ***atomically***
  - Only one thread can execute at a time
  - Often issue of *writes* among *reads*
- ***Synchronization mechanism: Lock***
  - Prevent more than one thread from accessing critical section at same time
  - If thread is interrupted while in the synchronized critical section, no other thread can start on the (synchronized) critical section

Problems

- Interrupted writes to ***shared*** data
  - Allows threads to access inconsistent data
- (Not an issue if the data is local to the thread)
- Example: Multiple threads run the following code on the shared variable deck

```
myCard = deck.peek()
// check, do I want that card?
deck.pop()
print(myCard)
```


Interleaving Example Problem

```
myCard = deck.peek()
deck.pop()
print(myCard)
```

```
myCard = deck.peek()
deck.pop()
print(myCard)
```

Threads interleaved:

```
myCard = deck.peek()
myCard = deck.peek()
deck.pop()
print(myCard)
deck.pop()
print(myCard)
```

What card does the blue thread *pop*?

What is the critical section?
(i.e., what code needs to executed *atomically* (without interruption)?)

Interleaving Example Problem

```
myCard = deck.peek()
deck.pop()
print(myCard)
```

Critical Section:
Atomic operation that needs to be done together

Printing does not need to be part of the critical section.
Why? While `deck` is a shared variable, `myCard` is a *local* variable.

Synchronization in Java

```
// Only one thread can execute at a time.
// The thread executing the block is said
// to hold the lock/monitor of the deck
// object
synchronized(deck) {
   // Access shared variables and other
   // shared resources
   myCard = deck.peek()
   deck.pop()
}
System.out.println(myCard)
```

210 in 325

Is there shared data in the web server?

- Even if only one Java statement in critical section, synchronize it!
- One high-level Programming Language statement probably translates into multiple VM language statements
  - Prevent interruption at low level

High-level:

```
count++;
```

Virtual Machine level:

```
Retrieve count
Add 1 to count
Store count
```


How does this relate to the Web Server?

- We have multiple threads
- Is there shared data? Is that data ever written?

INTERNET SERVICES

Discussion

- What is the focus of the paper?
  - What isn't?
- What is the author's experience?
- What did the author identify as challenges to giant-scale services?
- What are the benefits of networked solutions?
- What are the components of the network service system?

Benefits of Network Services

- Access anywhere, anytime
  - This is even more true now than it was in 2001
  - Cloud computing?
- Availability via multiple devices
  - Also more true…
- Groupware support
  - Calendaring, teleconferencing, messaging, etc.
- Lower overall cost
  - Multiplex infrastructure over active users
  - Dedicated resources are typically at least 96% idle
  - Central administrative burden, simplified end devices
- Simplified service updates
  - Update the service in one place, or 100 million?

Giant-Scale Services

- Paper does not address
  - Service monitoring, configuration, QoS, security, logging, and log analysis
  - Wide-area replicated services
  - Write-intensive services
  - Database management systems
- Challenges for network services:
  - High availability
    - Critical in today's Internet-dependent society
    - Each second of downtime = lots of lost money and revenue
  - Evolution
  - Growth

Buzzword Bingo: ISP

- + Net Neutrality

ISP: Internet Service Provider

- Provides connections to the rest of the Internet
- Net neutrality
  - Providers want to give preference to certain types of traffic
    - Ex: Slow down Netflix traffic on Comcast network so that people will prefer Comcast On-Demand
  - Net neutrality advocates want all traffic to be seen as equal – no distinction
    - Way it has always been, your expectation

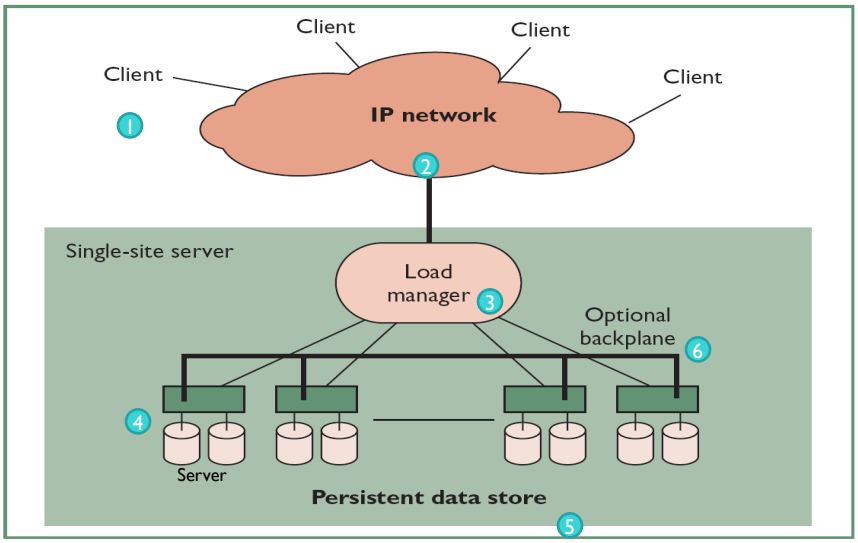
Cluster Discussion

- What is the case for clusters?
  - Is it still true?
- What do they consist of?

Buzzword: Commodity Components

- Defn: [relatively] cheap components, widely available, interchangeable
  - "Off-the-shelf" components
- Allow creation of [relatively] cheap clusters of machines that work together
- In contrast to specialty hardware, e.g., supercomputers

Network Service Components



Clusters as Building Blocks

- No alternative to clusters for building network services that can scale to global use
- Cluster benefits:
  - Incremental scalability
    - Adding one machine typically linearly improves performance
  - Independent components
  - Cost and performance

Looking Ahead

- Web Server Project!