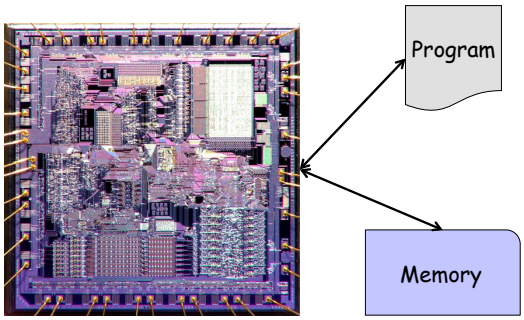
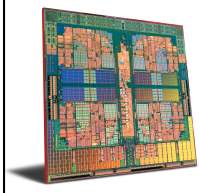
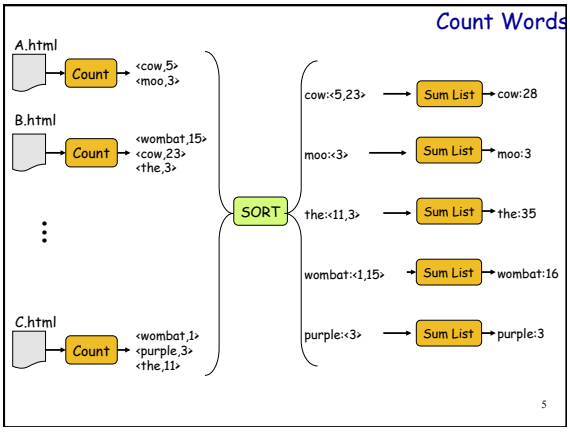
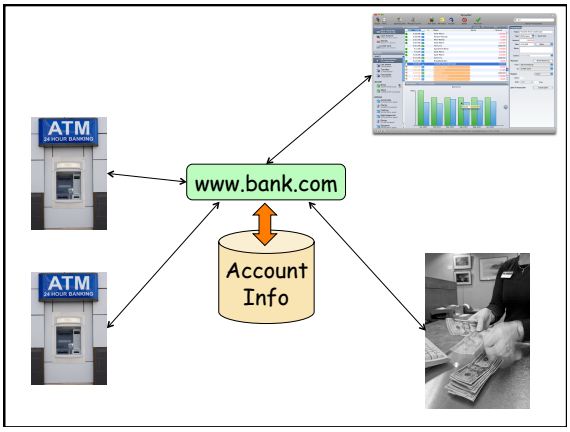
Parallel & Concurrent Programming

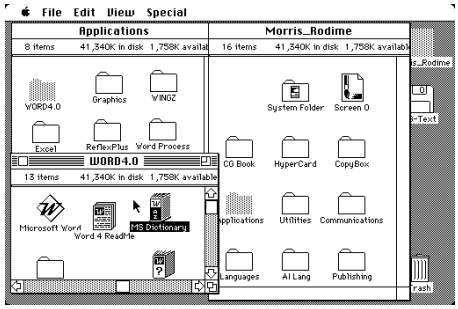
CSCI 334
Stephen Freund





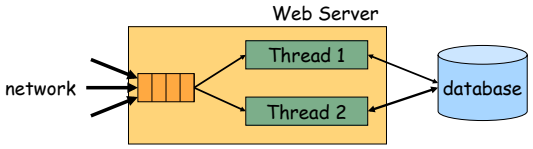






Concurrency

- Benefits
  - Speed
  - Availability
  - Distribution
  - Code structure
- Challenges
  - Hard to write
  - Not always possible (circuit evaluation)
  - Specifics
    - communication: send/receive info
    - synchronization: wait for another process
    - atomicity: don't stop in middle



Basic Question for Us

- How can programming languages make concurrent and distributed programming easier?

"PhotoShop" Demo

```
do i=1, n
     z(i) = x(i) + y(i)
enddo
```

```
z(1) = 1
do i=2, n
     z(i) = z(i-1)*2
enddo
```

```
z(1) = 1
do i=2, n
    z(i) = z(1)*2**(i-1)
enddo
```

cobegin/end

```
cobegin
  x = x + 1   ||   y = y + 1
end
```

MATLAB parfor Loop

```
clear A
for i = 1:8
    A(i) = i;
end
```

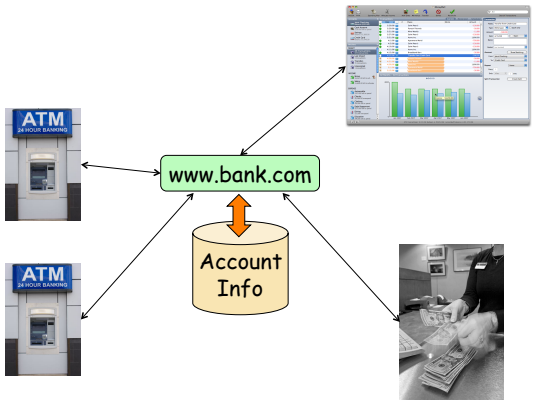
```
clear A
parfor i = 1:8
    A(i) = i;
end
```

```
clear A
parfor i = 1:8
    A(i) = A(i-1)+1;
end
```

?

Nondeterministic Concurrency

- Concurrency Control
  - mutual exclusion
  - semaphores
  - monitors
  - transactions
- Communication Abstractions
  - message passing
  - Actors



Race Condition Demo

Concurrency and Race Conditions

```
int bal = 0;
```

Thread 1

```
t1 = bal
bal = t1 + 10
```

Thread 2

```
t2 = bal
bal = t2 - 10
```

Thread 1 | Thread 2

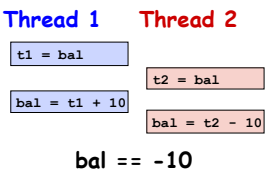
```
t1 = bal
bal = t1 + 10
```

```
t2 = bal
bal = t2 - 10
```

bal == 0

Concurrency and Race Conditions

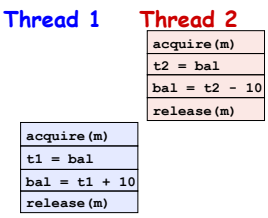
```
int bal = 0;

Thread 1
 t1 = bal
 bal = t1 + 10

Thread 2
 t2 = bal
 bal = t2 - 10
```



bal == -10

Concurrency and Race Conditions

```
Lock m = new Lock();
int bal = 0;

Thread 1
 synchronized(m) {
   t1 = bal
   bal = t1 + 10
 }

Thread 2
 synchronized(m) {
   t2 = bal
   bal = t2 - 10
 }
```



Account Monitor

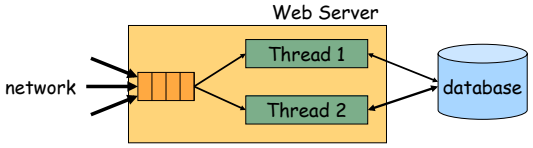
acquire lock of receiver object

```
class Account {
  private int balance;

  public synchronized void add(int n) {
    balance += n;
  }

  public synchronized String toString() {
    return "balance = " + balance;
  }
}
```

http://www.docjar.com/html/api/java/util/Vector.java.html

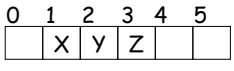
Producer-Consumer Buffers

- Buffer with finite size
  - Producers add values to it
  - Consumers remove values from it
- Used "everywhere"
  - buffer messages on network, OS events, events in simulation, messages between threads...



Java Buffer

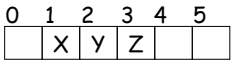
```
public class Buffer<T> {

    private T[] elementData;
    private int elementCount;
    private int start;
    private int end;
```



elementCount = 3

Java Buffer

```
public class Buffer<T> {

    private T[] elementData;
    private int elementCount;
    private int start;
    private int end;
```

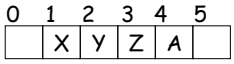


elementCount = 3

b.insert("A");

Java Buffer

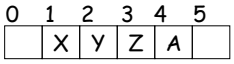
```
public class Buffer<T> {

    private T[] elementData;
    private int elementCount;
    private int start;
    private int end;
```



start end elementCount = 4

Java Buffer

```
public class Buffer<T> {

    private T[] elementData;
    private int elementCount;
    private int start;
    private int end;
```

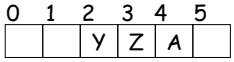


start end elementCount = 4

s = b.delete();

Java Buffer

```
public class Buffer<T> {

    private T[] elementData;
    private int elementCount;
    private int start;
    private int end;
```



start end elementCount = 3

Consumers

```
class Consumer extends Thread {
    private final Buffer<Character> buffer;

    public Consumer(Buffer<Character> b) {
        buffer = b;
    }

    public void run() {
        while (true) {
            char c = buffer.delete();
            System.out.print(c);
        }
    }
}
```

Producers

```
class Producer extends Thread {
    private final Buffer<Character> buffer;

    public Producer(Buffer<Character> b) {
        buffer = b;
    }

    public void run() {
        while (moreData()) {
            char c = next();
            buffer.insert(c);
        }
    }
}
```

Using Buffers

```
class Example {
  public static void main(String[] args) {
    Buffer<String> buffer = new Buffer<String>(5);
    Producer prod = new Producer(buffer);
    Consumer cons = new Consumer(buffer);
    prod.start();
    cons.start();
  }
}
```

Unsafe Buffer Ops

```
public class Buffer<T> {

    private T[] elementData;
    private int elementCount;
    private int start;
    private int end;

   public void insert(T t) {

     end = (end + 1) % elementData.length;
     elementData[end] = t;
     elementCount++;

   }

   public T delete() {

     T elem = elementData[start];
     start = (start + 1) % elementData.length;
     elementCount--;
     return elem;

   }
```

Safe Buffer Ops

```
lic class Buffer<T> {

 private T[] elementData;
 private int elementCount;
 private int start;
 private int end;

public synchronized void insert(T t) throws InterruptedException {
  while (elementCount == elementData.length) wait();
  end = (end + 1) % elementData.length;
  elementData[end] = t;
  elementCount++;
  notifyAll();
}

public synchronized T delete() throws InterruptedException {
  while (elementCount == 0) wait();
  T elem = elementData[start];
  start = (start + 1) % elementData.length;
  elementCount--;
  notifyAll();
  return elem;
}
```

Consumers With Handler

```
class Consumer extends Thread {
    private final Buffer<Character> buffer;

    public Consumer(Buffer<Character> b) {
        buffer = b;
    }

    public void run() {
        try {
            while (true) {
               char c = buffer.delete();
               System.out.print(c);
            }
        } catch (InterruptedException e) {
            // thread interrupted, so stop loop
        }
    }
}
```

Interrupting Threads

```
class Example {
  public static void main(String[] args) {
    Buffer<String> buffer = new Buffer<String>(5);
    Producer prod = new Producer(buffer);
    Consumer cons = new Consumer(buffer);
    prod.start();
    cons.start();
    try {
        prod.join();
        cons.interrupt();
    } catch (InterruptedException e) {
        System.out.println("...");
    }
  }
}
```

Account Monitor, redux

```
class Account {
  int balance;

  synchronized void add(int n) {
    balance += n;
  }

  synchronized void transfer(Account other,
                             int n) {
    balance -= n;
    other.add(n);
  }
```

Deadlock

Thread 1	Thread 2
a.transfer(b,n)	b.transfer(a,n)
acquire(a)	acquire(b)
a.bal -= n	
wait for b	b.bal -= n
	wait for a

```
class Account {
  int balance;

  synchronized void add(int n) {
    balance += n;
  }

  synchronized void transfer(Account other,
                             int n) {
    balance -= n;
    other.add(n);
  }
```

java.lang.StringBuffer...

```
public final class StringBuffer {
  int count;
  char chars[];
  synchronized int length() { return count; }
  synchronized int getChars(...) { ... }

  synchronized StringBuffer append(StringBuffer sb) {
    int len = sb.length();
    ...
    sb.getChars(0, len, value, count);
    ...
 }
}
```

Atomicity Demo

java.util.StringBuffer...

```
public final class StringBuffer {
  int count;
  char chars[];
  synchronized int length() { return count; }
  synchronized int getChars(...) { ... }

  synchronized StringBuffer append(StringBuffer sb) {
    int len = sb.length();
    ...
    sb.getChars(0, len, value, count);
    ...
 }
}
```

http://www.docjar.com/html/api/java/lang/StringBuffer.java.html

Atomic As a Language Feature

```
class Account {
  int balance;

  atomic void add(int n) {
    balance += n;
  }

  atomic void transfer(Account other, int n) {
    balance -= n;
    other.add(n);
  }
```

Pessimistic Atomicity Implementation

```
class Account {
  int balance;

  void add(int n) {
    synchronized(global_lock) {
      balance += n;
    }
  }

  void transfer(Account other, int n) {
    synchronized(global_lock) {
      balance -= n;
      other.add(n);
    }
  }
```

What's good? What's bad? Alternatives?