

CS2031

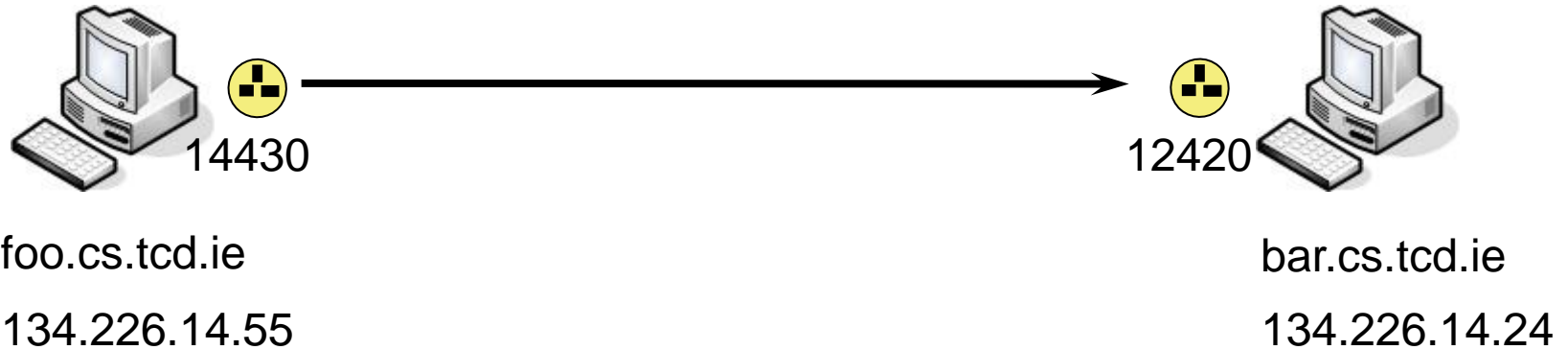
Telecommunications II

Datagram Sockets

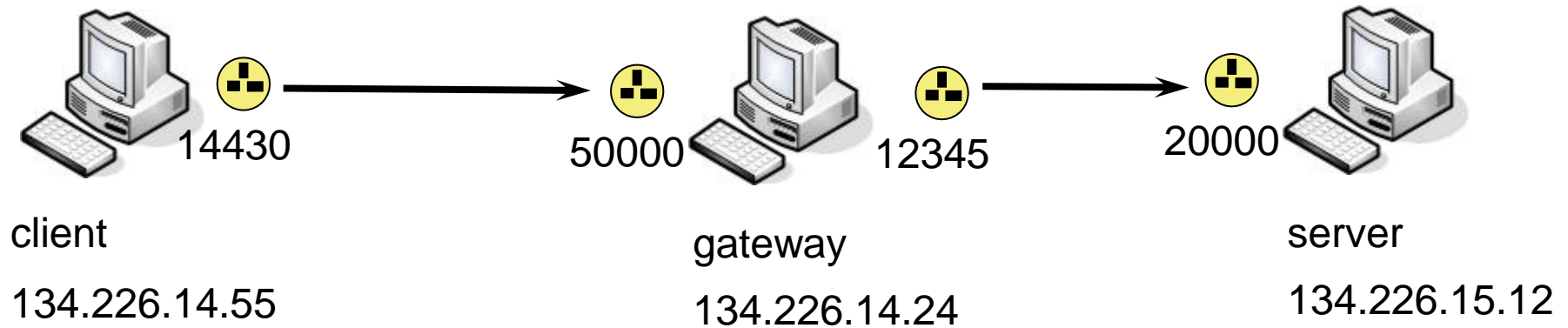
Programming Concepts

- Multi-threading
- Event-based Programming
 - Callbacks

Sockets & Ports

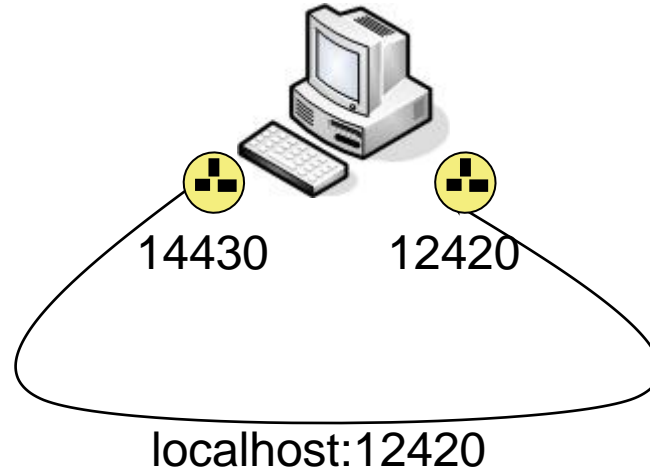


Sockets & Ports



Incoming Outgoing

Sockets & Ports



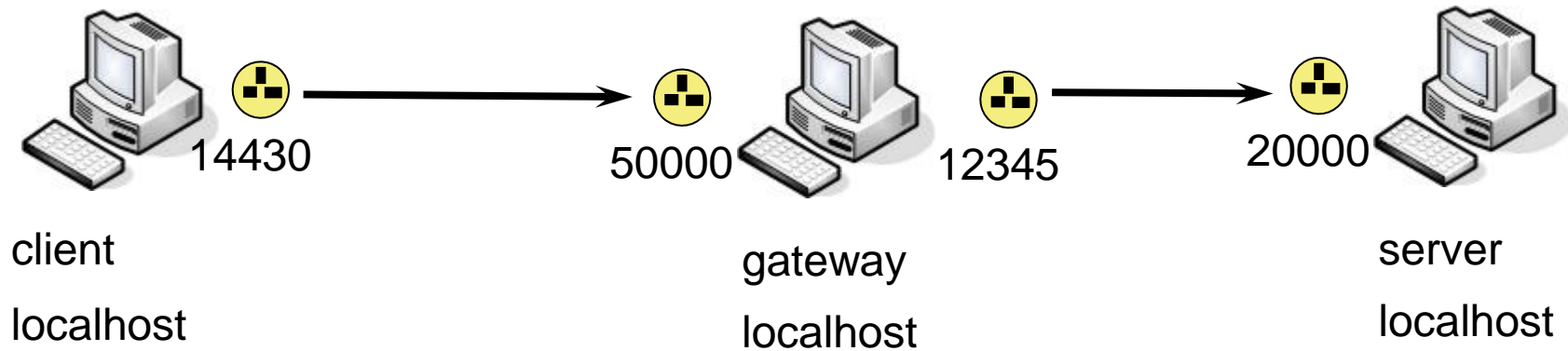
```
socket= new DatagramSocket(14430);
```

```
dstAddress= new InetSocketAddress("localhost", 12420);
```

```
packet= new DatagramPacket(data, data.length, dstAddress);
```

```
socket.send(packet);
```

Assignment on Local Machine



Threads

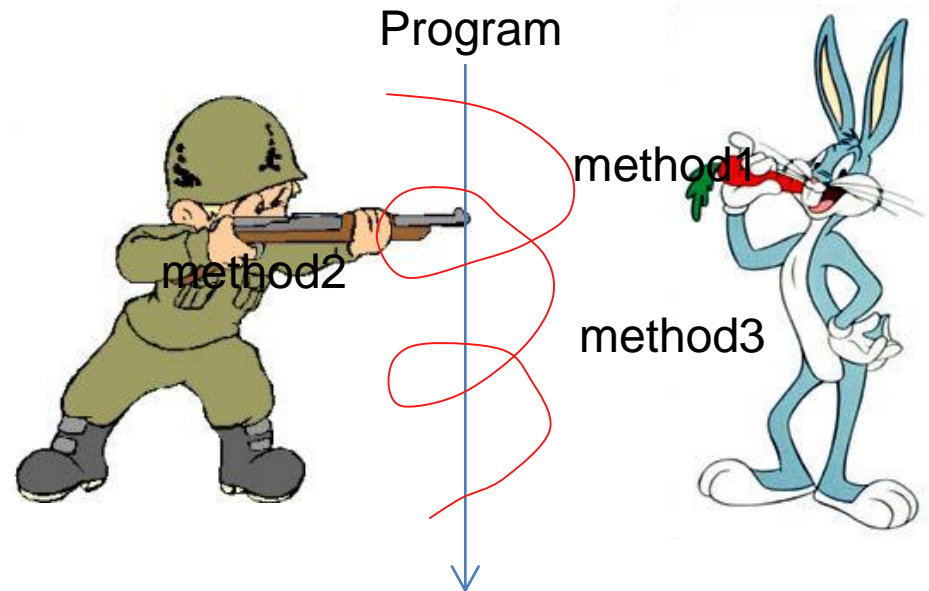


**...and now to something
completely different**

Threads

- Threats^d of Execution

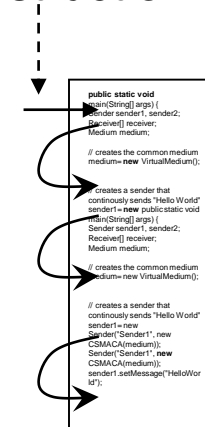
- Lightweight Processes



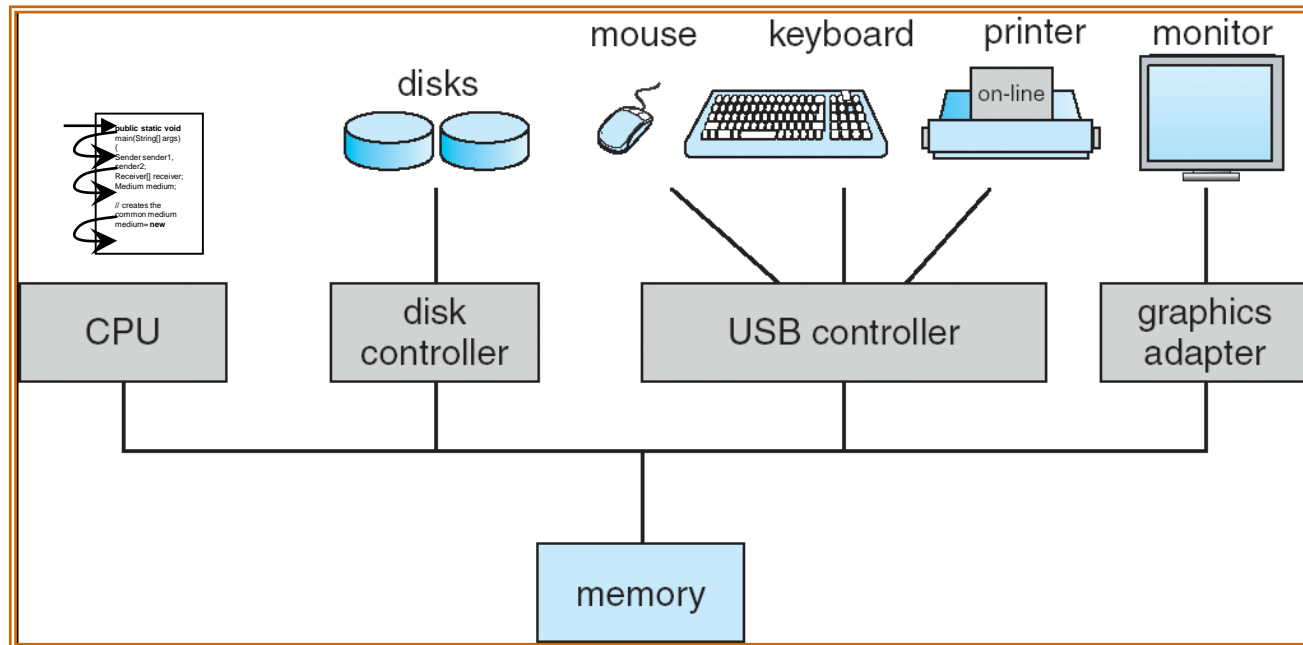
Single-Process System

- One process

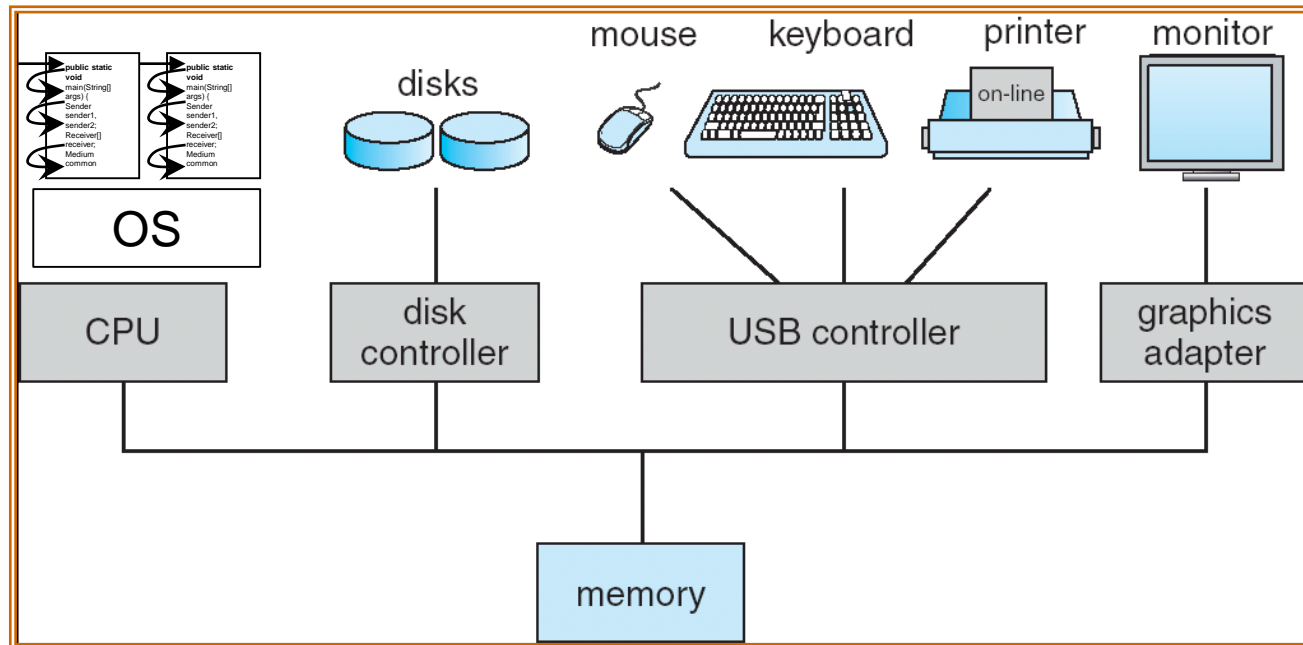
Instruction Pointer



Single Program – Complete Control

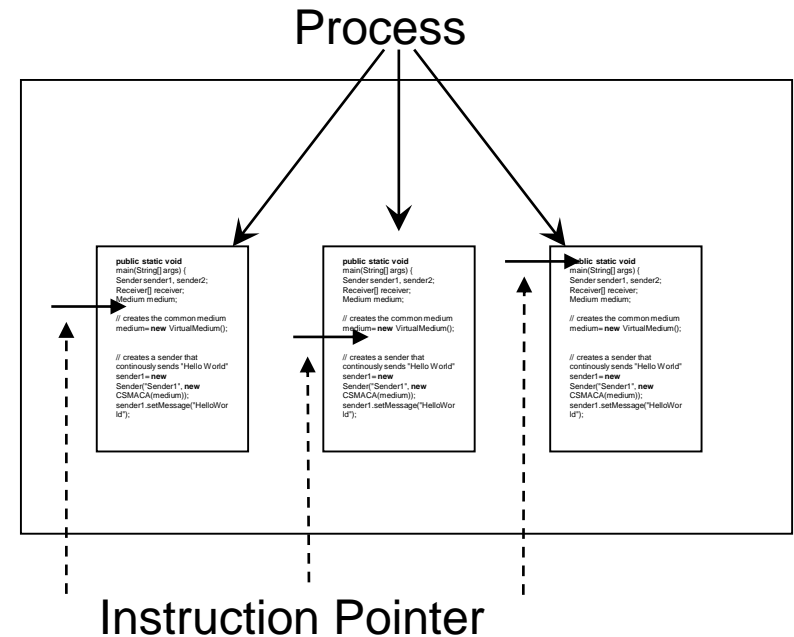


OS & Multiple Programs → Chaos



Processes

- Separate address spaces
- Registers per process
- Problem:
 - Switching between processes

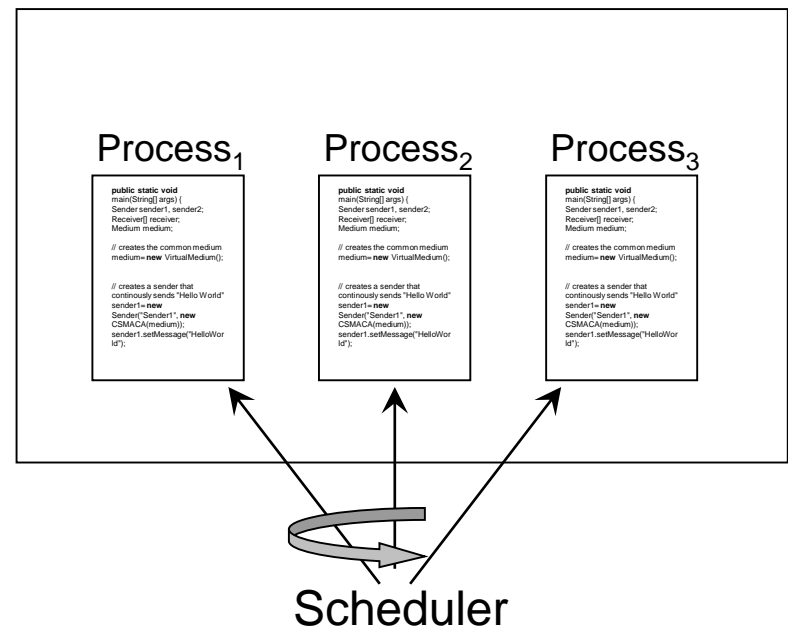


Per-Process Details

Process management	Memory management	File management
Registers Program counter Program status word Stack pointer Process state Time when process started CPU time used Children's CPU time Time of next alarm Message queue pointers Pending signal bits Process id Various flag bits	Pointer to text segment Pointer to data segment Pointer to bss segment Exit status Signal status Process id Parent process Process group Real uid Effective uid Real gid Effective gid Bit maps for signals Various flag bits	UMASK mask Root directory Working directory File descriptors Effective uid Effective gid System call parameters Various flag bits

Process Switching

- Saving of registers
 - Instruction pointer
 - Stack pointers
 - Other registers



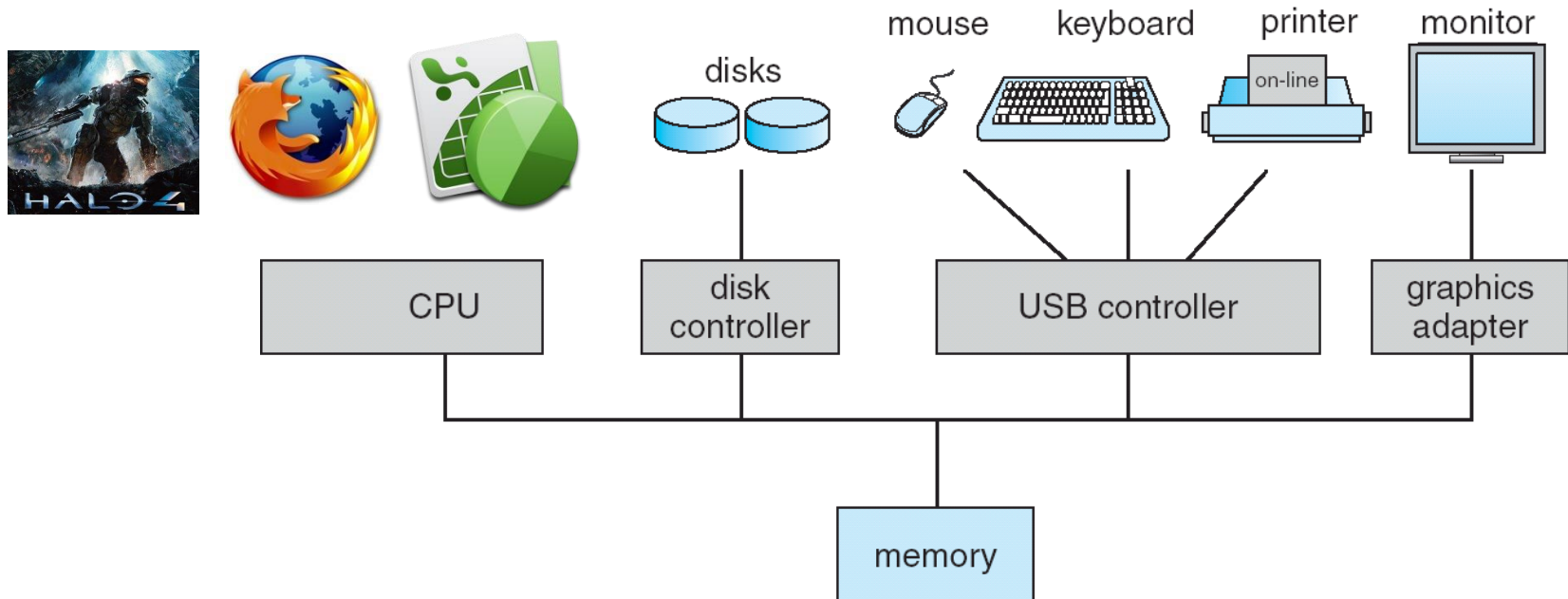
- Switching Virtual Memory



Overhead!

P ₁	Process management Registers Program counter Program status word Stack pointer Process state Time when process started CPU time used Message queue pointers Pending signals Process ID Various flag bits	Memory management Pointer to last segment Pointer to data segment Pointer to free segment Exit status Signal status Process ID Parent process Process group Real uid Effective uid Real gid Effective gid Set maps for signals Various flag bits	File management UID/GID mode Root directory Working directory File descriptors Effective uid Effective gid System call parameters Various flag bits
P ₂	Process management Registers Program counter Program status word Stack pointer Process state Time when process started CPU time used Message queue pointers Pending signals Process ID Various flag bits	Memory management Pointer to last segment Pointer to data segment Pointer to free segment Exit status Signal status Process ID Parent process Process group Real uid Effective uid Real gid Effective gid Set maps for signals Various flag bits	File management UID/GID mode Root directory Working directory File descriptors Effective uid Effective gid System call parameters Various flag bits
P ₃	Process management Registers Program counter Program status word Stack pointer Process state Time when process started CPU time used Message queue pointers Pending signals Process ID Various flag bits	Memory management Pointer to last segment Pointer to data segment Pointer to free segment Exit status Signal status Process ID Parent process Process group Real uid Effective uid Real gid Effective gid Set maps for signals Various flag bits	File management UID/GID mode Root directory Working directory File descriptors Effective uid Effective gid System call parameters Various flag bits

Switching Programs

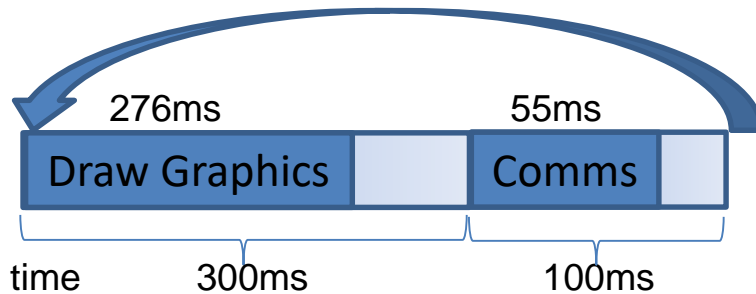
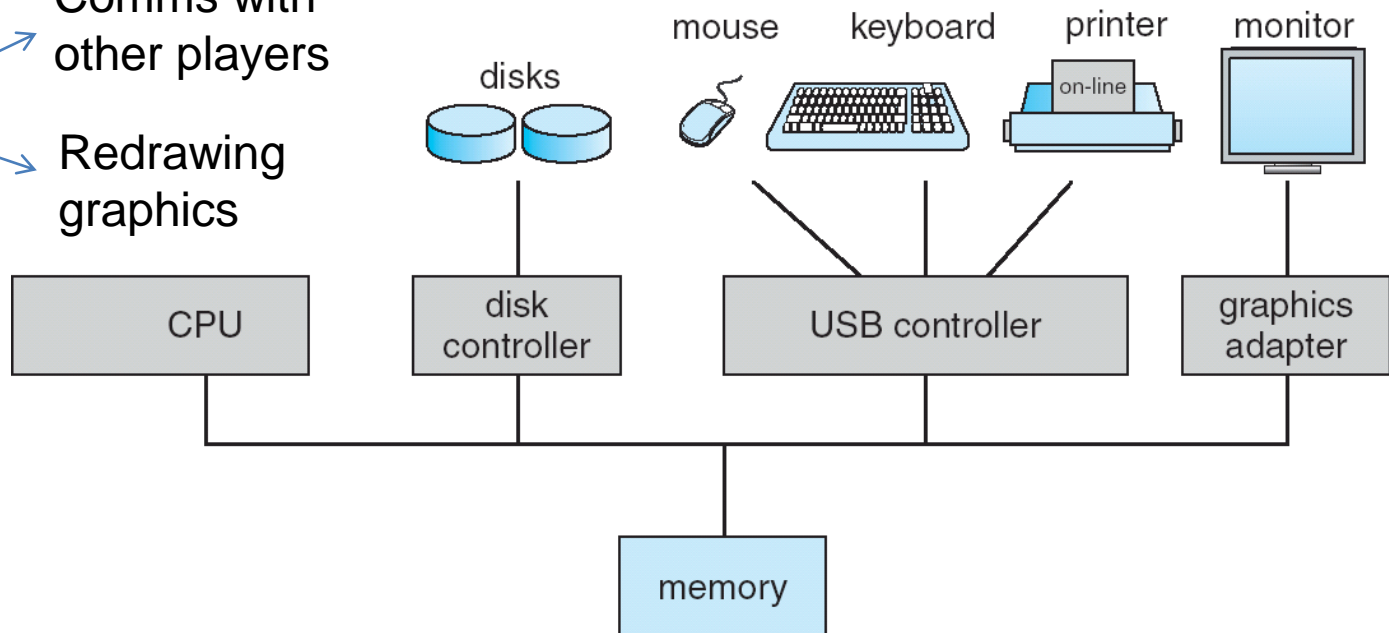


Switching Tasks in a Program



Comms with
other players

Redrawing
graphics



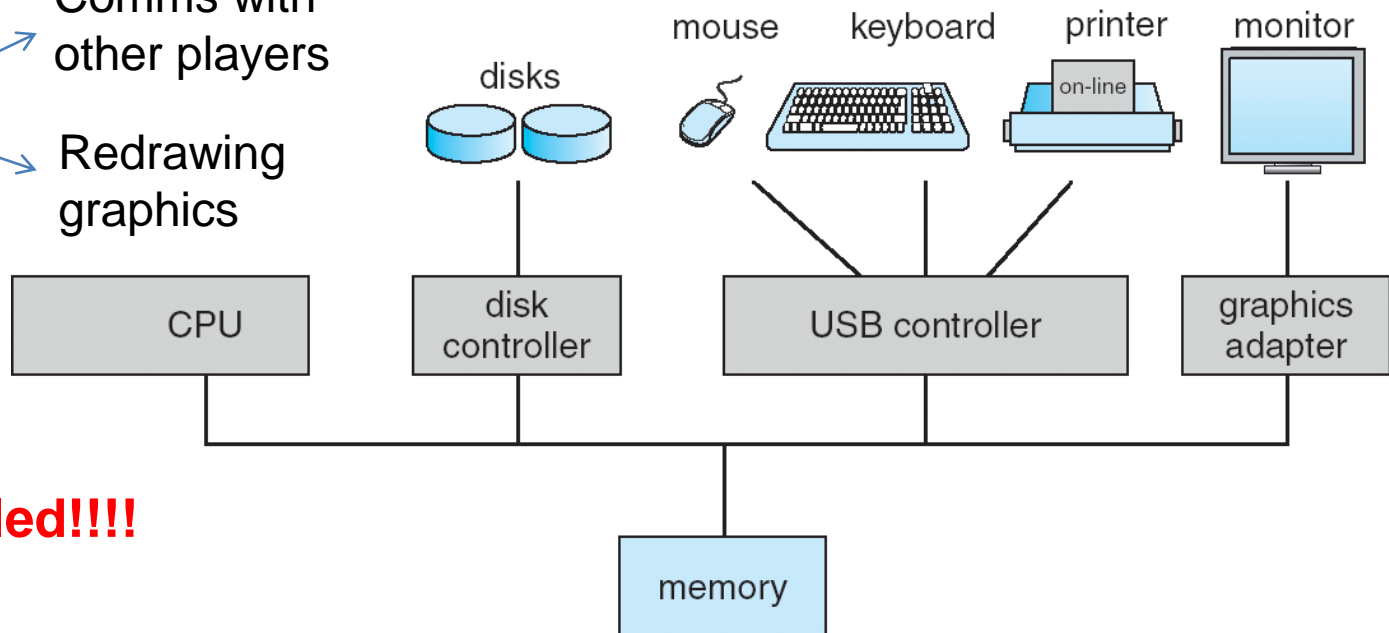
- Every 400ms the graphic will be redrawn
- It is important that the tasks are shorter than the allocated time

Switching Tasks in a Program

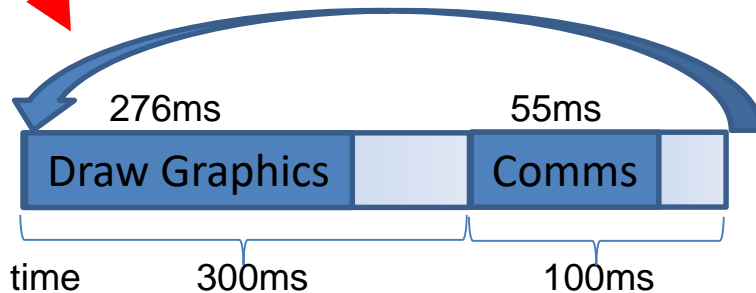


Comms with
other players

Redrawing
graphics



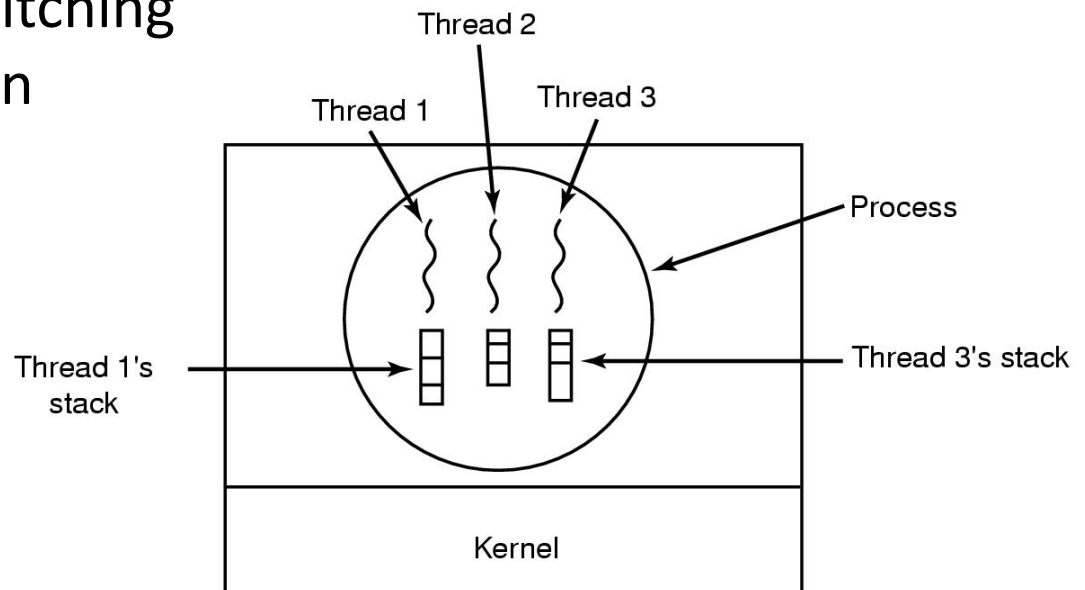
Tightly coupled!!!!



- Every 400ms the graphic will be redrawn
- It is important that the tasks are shorter than the allocated time

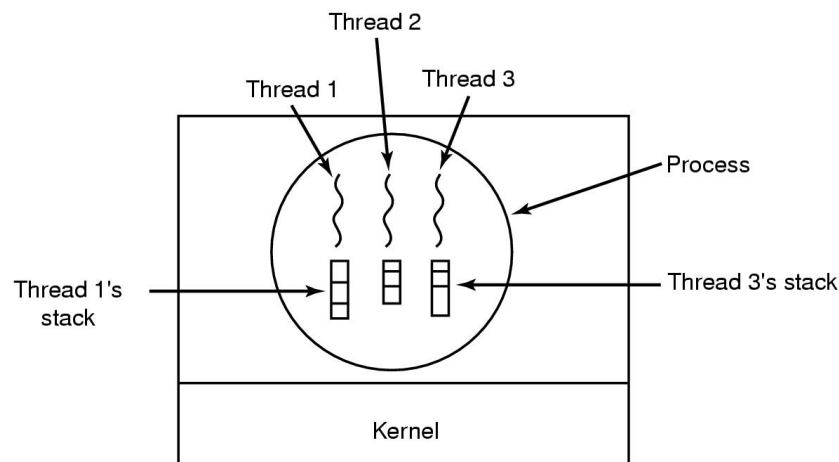
Threads

- Lightweight processes
- Share same address space
- Less overhead for switching between threads than between processes



Threads

- Lightweight processes
- Share same address space
- Less overhead for switching between threads than between processes



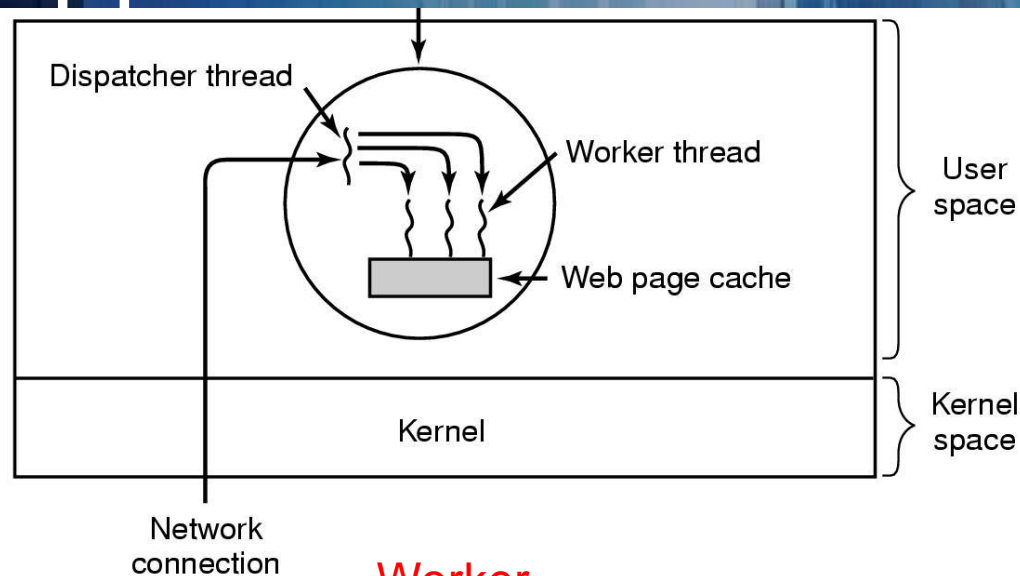
Per process items

Address space
Global variables
Open files
Child processes
Pending alarms
Signals and signal handlers
Accounting information

Per thread items

Program counter
Registers
Stack
State

Application of Threads



Dispatcher

```
while (TRUE) {  
    get_next_request(&buf);  
    handoff_work(&buf);  
}
```

(a)

Worker

```
while (TRUE) {  
    wait_for_work(&buf)  
    look_for_page_in_cache(&buf, &page);  
    if (page_not_in_cache(&page))  
        read_page_from_disk(&buf, &page);  
    return_page(&page);  
}
```

(b)

Java Threads

```
class Thread {  
    public Thread (String name);  
    public Thread (Runnable target)  
    ...  
    public void start ();  
    static void sleep (long millis)  
}
```

Selection of methods of
class "Thread"

Java Threads

```
class Thread {  
    public Thread (String name);  
    ...  
    public void start ();  
    public void run();  
}
```

Selection of methods of
class “Thread”

```
class XYZ extends Thread {  
    public void run() {  
    }  
}
```

← Class that extends
“Thread” needs to
implement the **run**
method

Java Thread – Socket Example I

```
class SocketThread extends Thread {  
    DatagramSocket socket;  
  
    SocketThread (String name, int port) {  
        super (name);  
        socket= new DatagramSocket(port);  
    }  
}  
  
t1 = new SocketThread ("Socket1", 50000);
```

Java Thread – Socket Example II

```
class SocketThread extends Thread {
```

```
    DatagramSocket socket;
```

```
    SocketThread (String name, int port) {  
        super (name);  
        socket= new DatagramSocket(port);  
    }
```

```
    public void run() {  
        while(TRUE) {  
            packet= socket.receive();  
            System.out.println (name + ":" + packet.getData());  
        }  
    }  
}
```


Creating & Starting Threads I

SocketThread t1, t2, t3;

t1 = new SocketThread ("Socket1", 50000);

t2 = new SocketThread ("Socket2", 50200);

t3 = new SocketThread ("Socket3", 55000);

Creating & Starting Threads II

SocketThread t1, t2, t3;

t1 = new SocketThread ("Socket1", 50000);

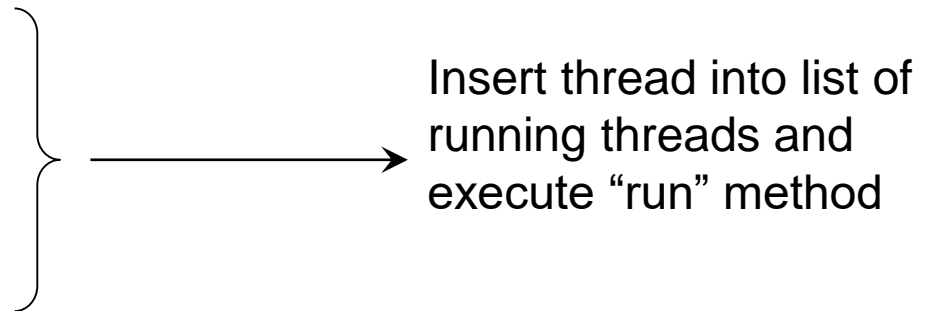
t2 = new SocketThread ("Socket2", 50200);

t3 = new SocketThread ("Socket3", 55000);

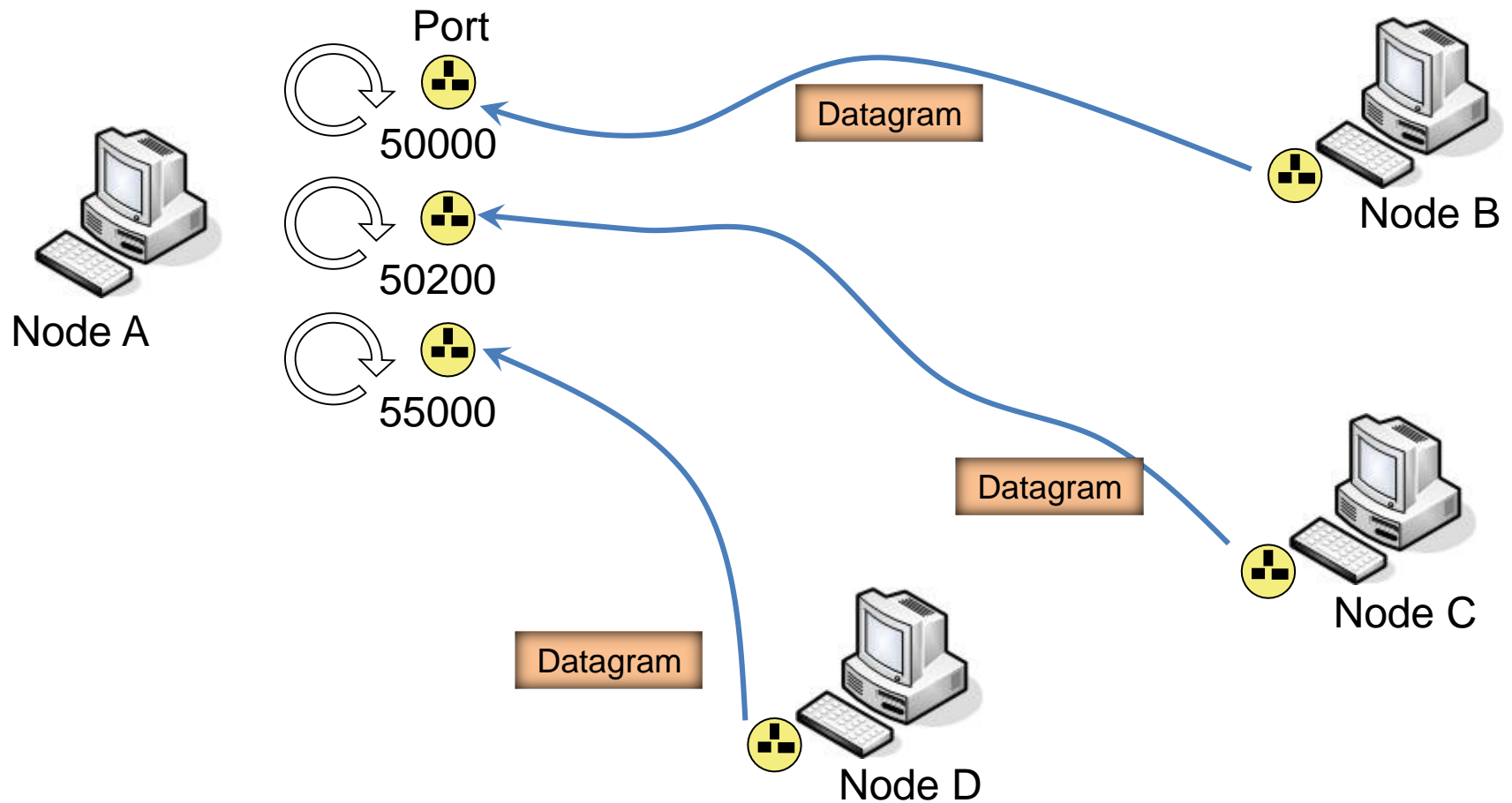
t1.start();

t2.start();

t3.start();



Concurrent Communication



Thread Execution Example I

```
class CounterThread extends Thread {  
    long counter;  
  
    CounterThread (String name, long counter) {  
        super (name);  
        this.counter = counter;  
    }  
}
```

```
t1 = new CounterThread ("T1", 10);
```

Thread Execution Example II

```
class CounterThread extends Thread {  
    long counter;  
  
    CounterThread (String name, long counter) {  
        super (name);  
        this.counter = counter;  
    }  
  
    public void run() {  
        while(TRUE) {  
            counter++;  
            System.out.println (name + ": " + counter);  
            Thread.sleep (Math.random() * 5000);  
        }  
    }  
}
```

Thread Execution Example III

CounterThread t1, t2, t3;

```
t1 = new CounterThread ("T1", 10);
```

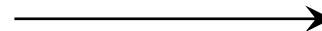
```
t2 = new CounterThread ("T2", 10);
```

```
t3 = new CounterThread ("T3", 10);
```

```
t1.start();
```

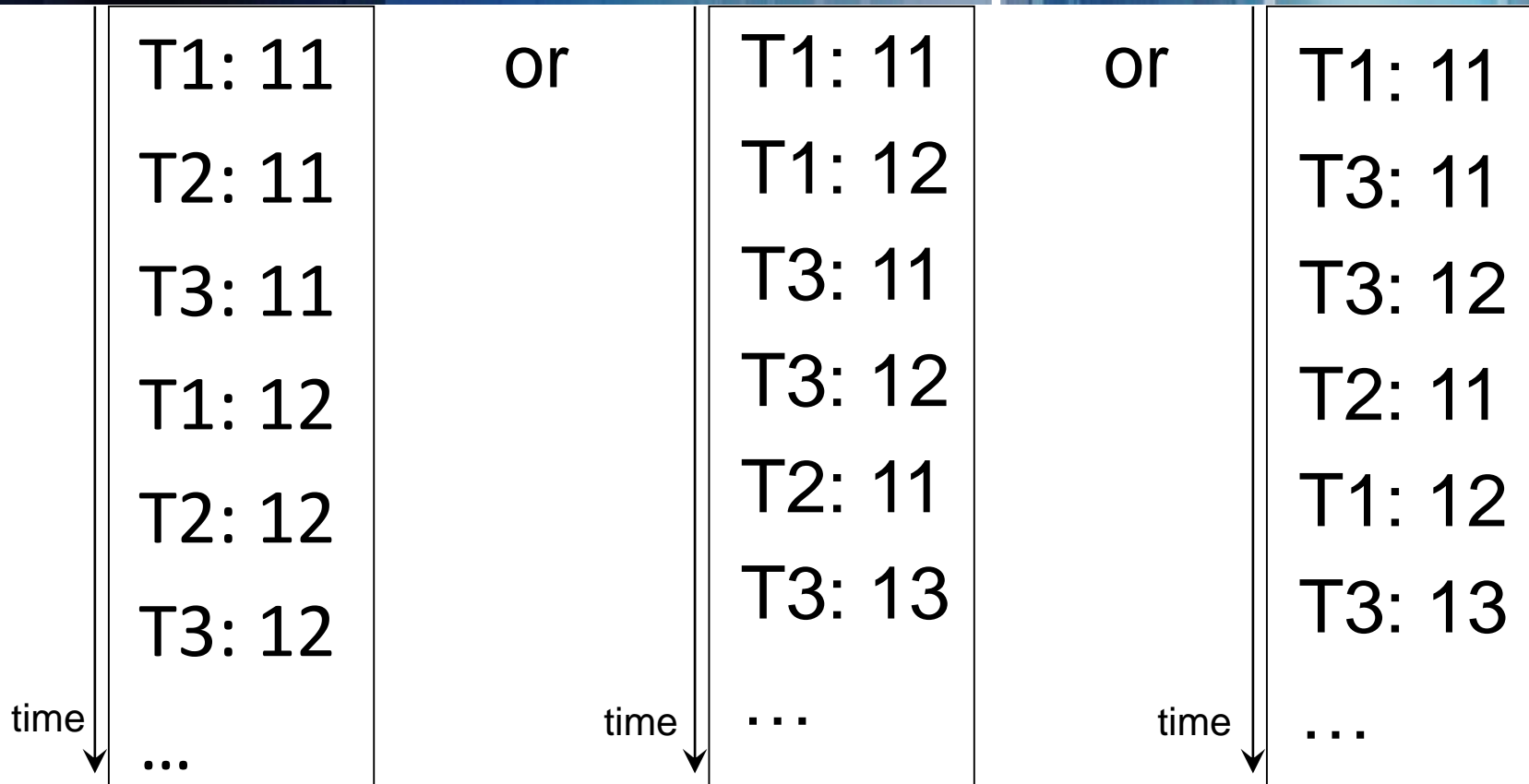
```
t2.start();
```

```
t3.start();
```



Insert thread into list of running threads and execute "run" method

Possible Output



Execution is **non-deterministic**!

Interface: `java.lang.Runnable`

Java doesn't support Multiple Inheritance:

`class AccountThread extends Thread, Account {...`

← **ERROR**

**Java doesn't support
multiple inheritance**

Interface: java.lang.Runnable

Java doesn't support Multiple Inheritance:

```
class AccountThread extends Thread, Account {...
```

← ERROR
**Java doesn't support
multiple inheritance**

```
class CounterThread implements Runnable {
```

```
...
```

```
    public void run() {
```

```
}
```

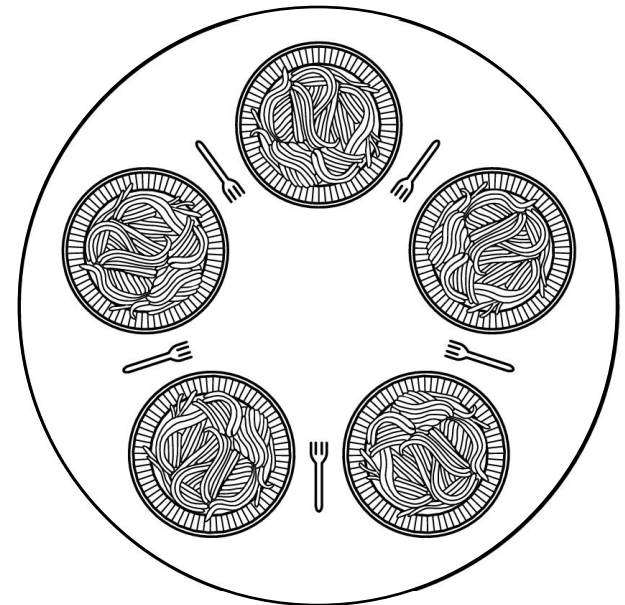
```
new Thread (new CounterThread("T1", 10)).start;
```

Problems with Concurrency

- Concurrent access to global variables, etc

- Requires synchronization

- Approaches
 - Monitors
 - Semaphores
 - Barriers



Dining Philosophers

(see Principles of Concurrent Programming, M. Ben-Ari)

Producer-Consumer Problem



- Producer delivers 1 egg at a time
- Basket can hold exactly 1 egg
- Consumer can only consume an egg if an egg is in the basket

Producer-Consumer in Java I

```
class TestSystem {  
    Basket basket;  
  
    TestSystem() {  
        basket= new Basket(0);  
    }  
  
    class Basket {  
        int content;  
  
        public Basket (int content) {  
            this.content= content;  
        }  
    }  
}
```

Producer-Consumer in Java II

```
class TestSystem {  
    ...  
    class Basket {  
        int content;  
        ...  
  
        public void putEgg () {  
            content++;  
        }  
  
        public void takeEgg() {  
            content--;  
        }  
    }  
}
```

Producer-Consumer in Java III

```
class TestSystem {  
    Basket basket;  
  
    class Producer extends Thread {  
        public void run() {  
            while (true) basket.putEgg();  
        }  
    }  
  
    class Consumer extends Thread {  
        public void run() {  
            while (true) basket.takeEgg();  
        }  
    }  
}
```

Producer-Consumer in Java IV

```
class TestSystem {  
  
    public static void main (String[] args) {  
        Producer producer;  
        Consumer consumer;  
  
        producer= new Producer();  
        consumer= new Consumer();  
  
        producer.start();  
        consumer.start();  
    }  
}
```

Problem???

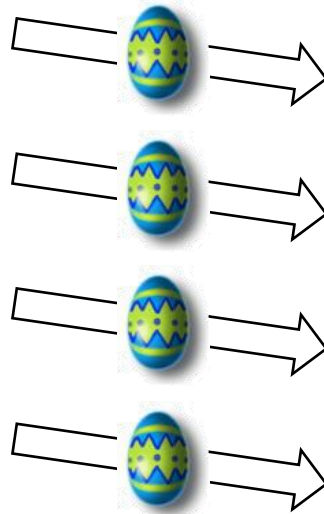
Producer

putEgg

putEgg

putEgg

putEgg

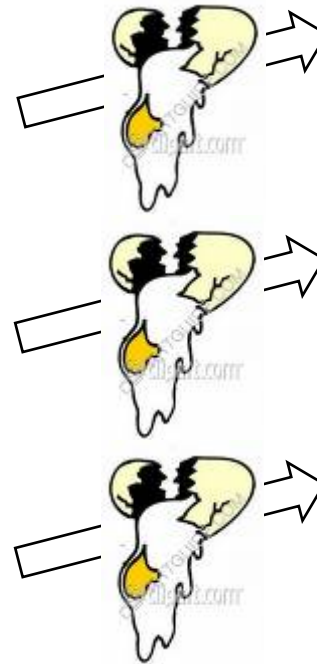


Consumer

getEgg

getEgg

getEgg

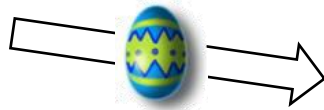


Execution is non-deterministic!

Problem???

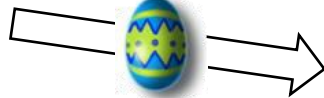
Producer

putEgg



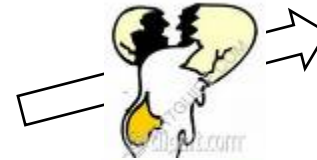
twiddle thumbs

putEgg

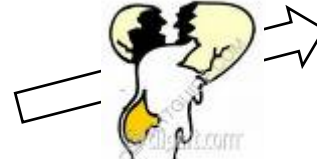


Consumer

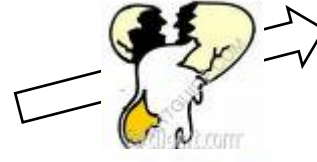
getEgg



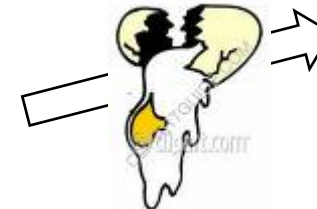
getEgg



getEgg



getEgg



Execution is non-deterministic!

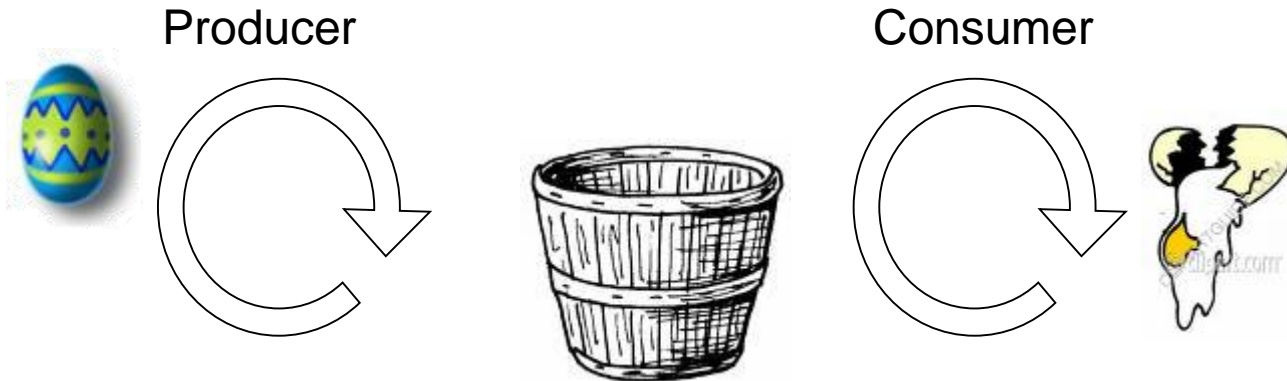
Producer-Consumer in Java V

```
class TestSystem {  
    ...  
    class Basket {  
        int content;  
        ...  
        public synchronized void putEgg () {  
            while (content!=0) wait();  
            content++;  
            notify();  
        }  
    }  
}
```

Producer-Consumer in Java VI

```
class TestSystem {  
    ...  
    class Basket {  
        int content;  
        ...  
        public synchronized void takeEgg () {  
            while (content!=1) wait();  
            content--;  
            notify();  
        }  
    }  
}
```

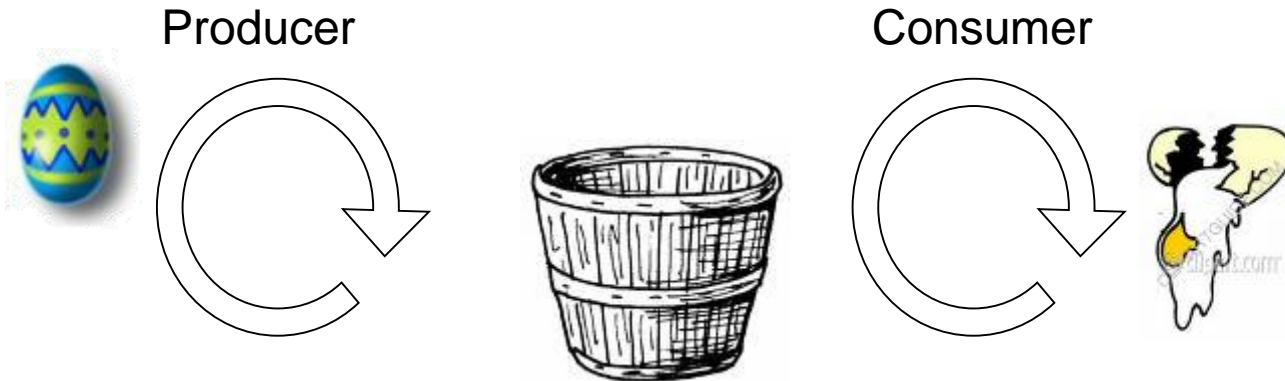
Producer-Consumer Problem



```
public synchronized void putEgg () {  
    while (content!=0) wait();  
    content++;  
    notify();  
}
```

```
public synchronized void takeEgg () {  
    while (content!=1) wait();  
    content--;  
    notify();  
}
```

Producer-Consumer Problem



```
public synchronized void putEgg () {  
    while (content!=0) wait();  
    content++;  
    notify();  
}
```

```
public synchronized void takeEgg () {  
    while (content!=1) wait();  
    content--;  
    notify();  
}
```

Monitor in Java: One active thread in method per instance!

Summary: Threads

- Concurrent Execution
 - Non-deterministic Execution
- Java
 - Inherit from Thread class
 - Implement Runnable interface
- Synchronization
 - wait() & notifyAll() / notify()

CS2031

Telecommunications II

Event-based Programming

“Event-based Programming”

```
public void run() {  
    DatagramPacket packet;  
  
    try {  
        while(true) {  
            packet = new DatagramPacket(new byte[PACKETSIZE], PACKETSIZE);  
            socket.receive(packet);  
            onReceipt(packet);  
        }  
    } catch (Exception e) {e.printStackTrace();}  
}
```


“Event-based Programming”

Listener : Thread



receive packet
call onReceipt()

“Event-based Programming”

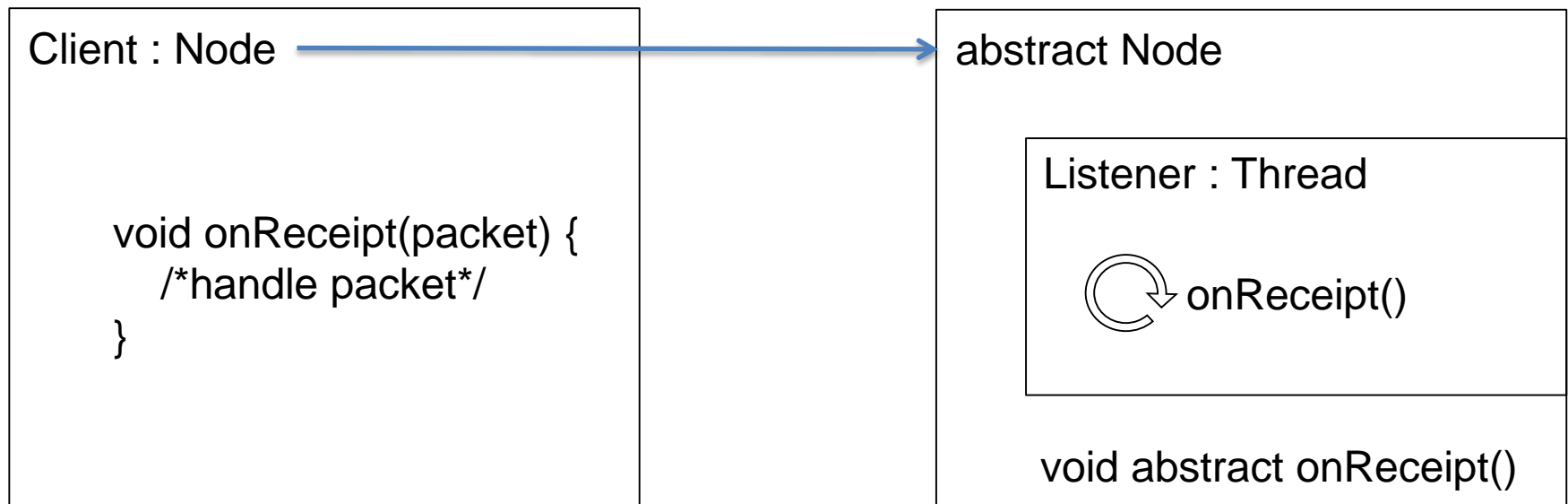
abstract Node

Listener : Thread

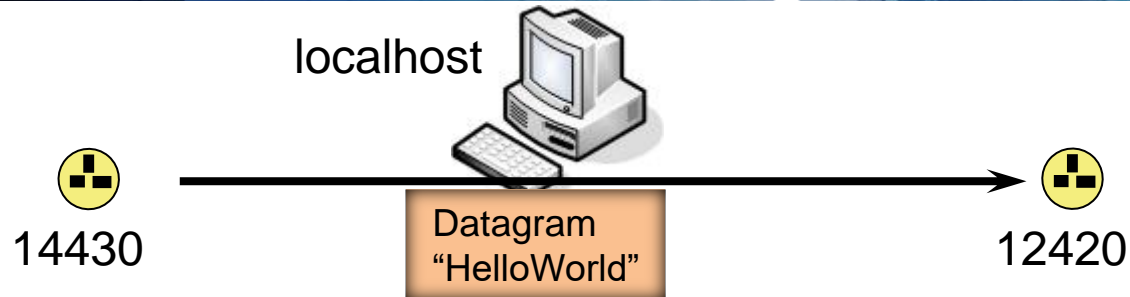
 onReceipt()

void abstract onReceipt()

“Event-based Programming”



“Event-based Programming”



Server

```
void onReceipt(packet) {  
}
```

“Event-based Programming”

```
public void go() {latch.countDown();}  
public void run() {  
    DatagramPacket packet;  
  
    try {  
        latch.await();  
        while(true) {  
            packet = new DatagramPacket(new byte[PACKETSIZE], PACKETSIZE);  
            socket.receive(packet);  
            onReceipt(packet);  
        }  
    } catch (Exception e)  
        {if (!(e instanceof SocketException)) e.printStackTrace();}  
}
```



That's all
folks