school of Computer Science & Statistics

CS2031
Telecommunications II

Datagram Sockets

TRINITY COLLEGE DUBLIN
COLÁISTE NA TRÍONÓIDE, BAILE ÁTHA CLIATH
THE UNIVERSITY OF DUBLIN

Nodes & Ports

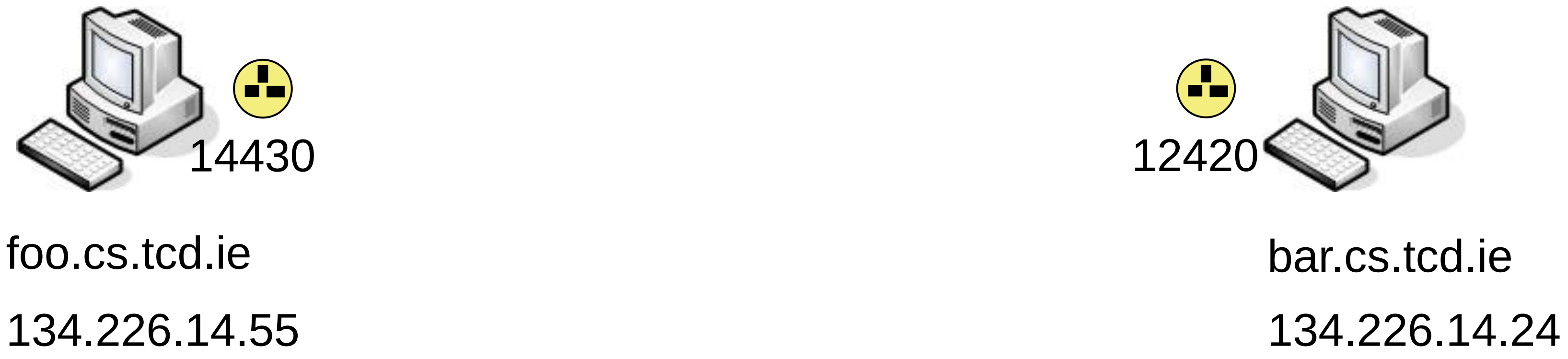
foo.cs.tcd.ie

134.226.14.55

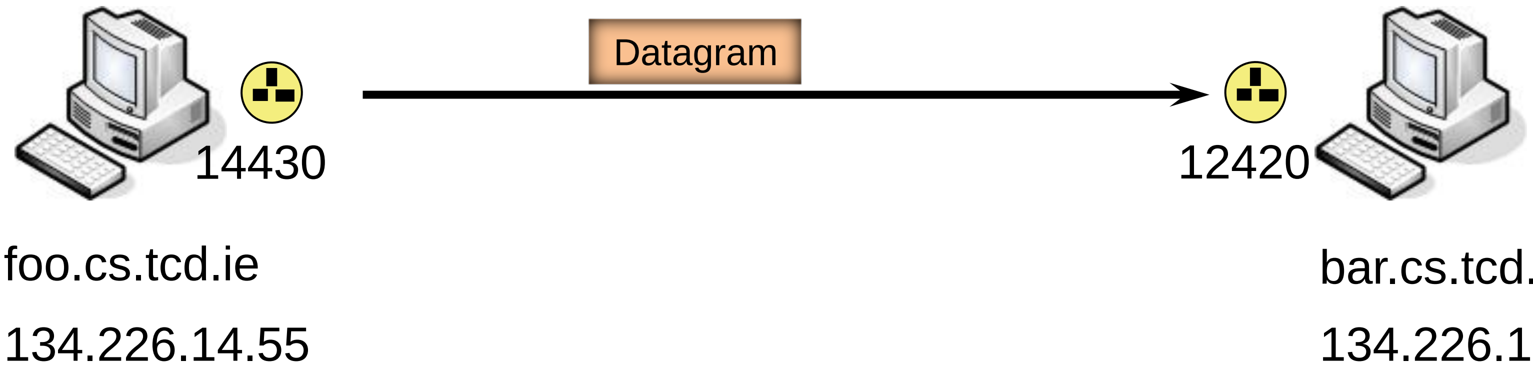
bar.cs.tcd.ie

134.226.14.24

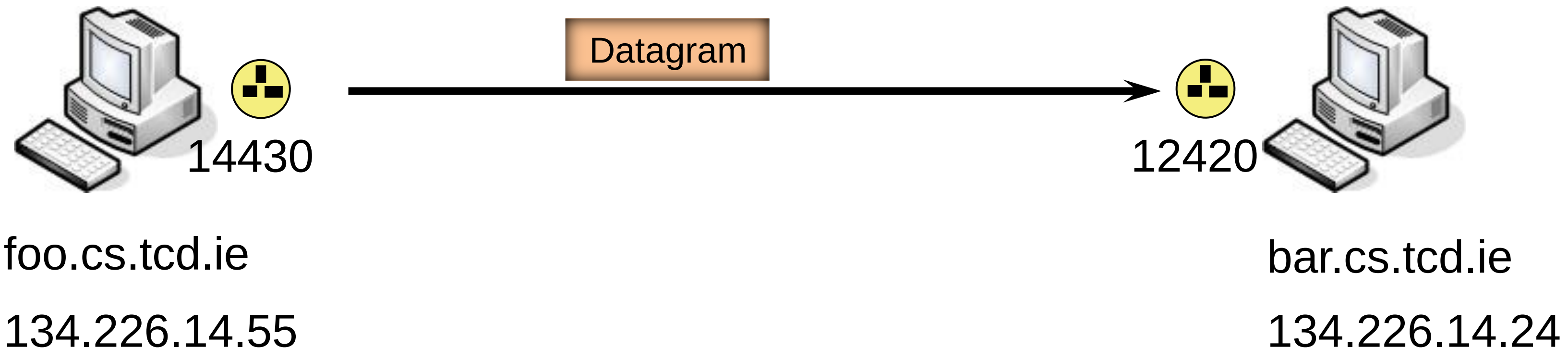
Sockets & Ports



Sockets & Ports

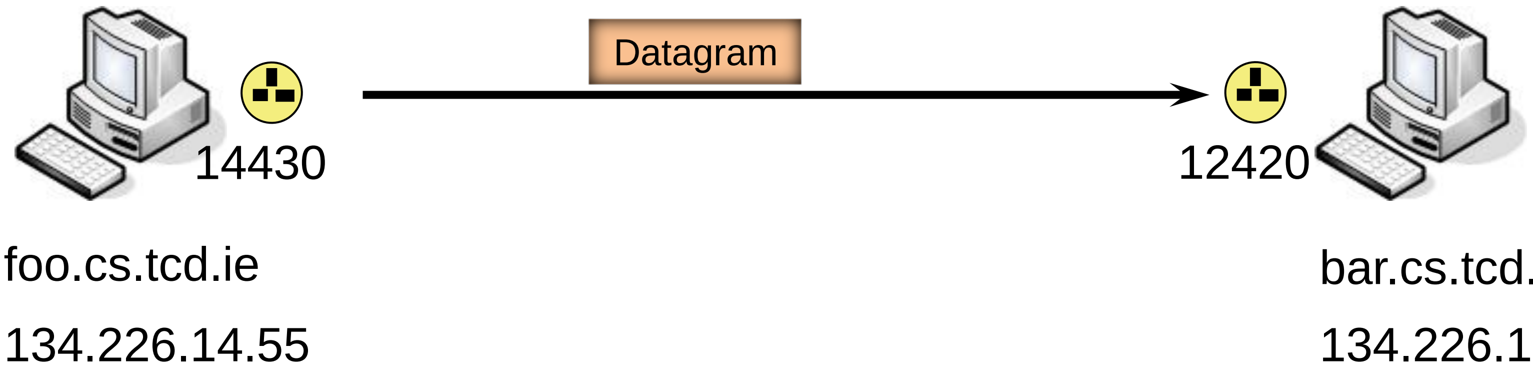


Sockets & Ports



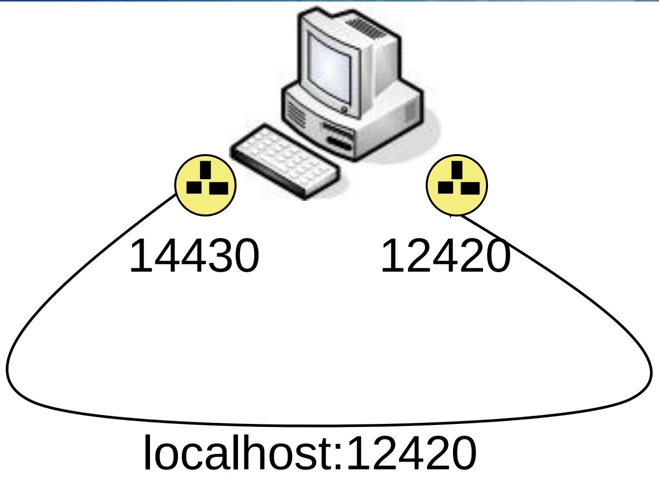
```
socket= new DatagramSocket(14430);

dstAddress= new InetSocketAddress(“bar.cs.tcd.ie”, 12420);
packet= new DatagramPacket(data, data.length, dstAddress);
socket.send(packet);
```

Sockets & Ports



```
socket= new DatagramSocket(12420);

packet = new DatagramPackte(new byte[SIZE], SIZE);
socket.receive(packet);
```

Sockets & Ports



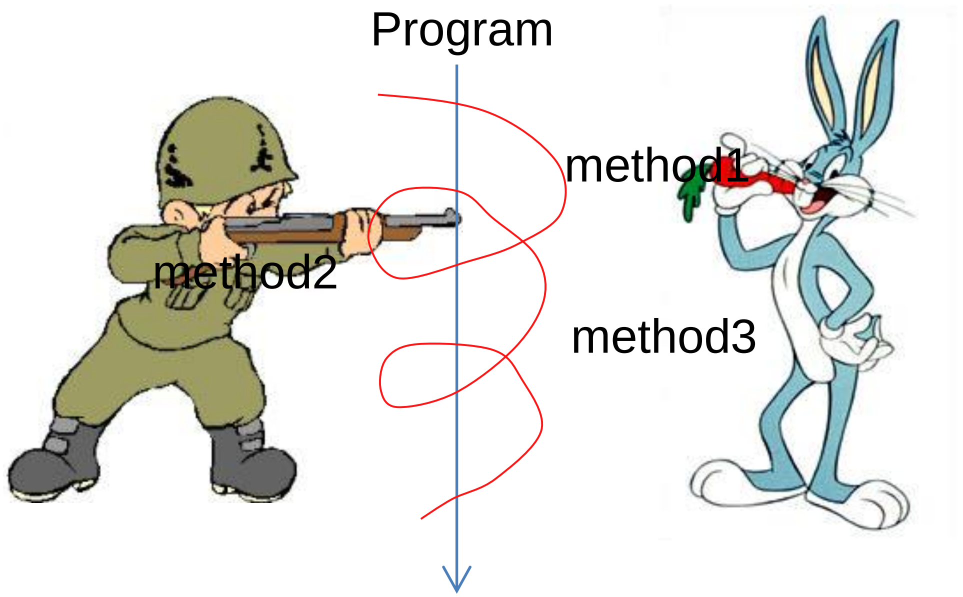
```
socket= new DatagramSocket(14430);

dstAddress= new InetSocketAddress("localhost", 12420);
packet= new DatagramPacket(data, data.length, dstAddress);
socket.send(packet);
```

Threads

…and now to something completely different

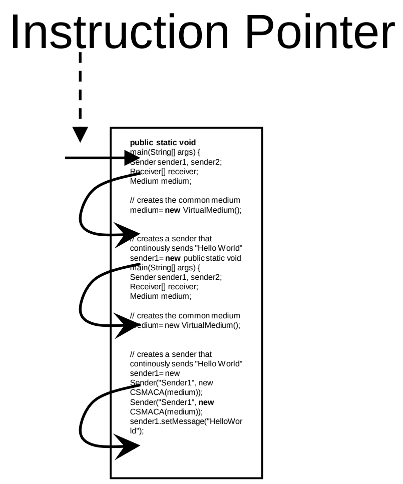
Threads

- Threa~~t~~ds of Execution
  - Lightweight Processes



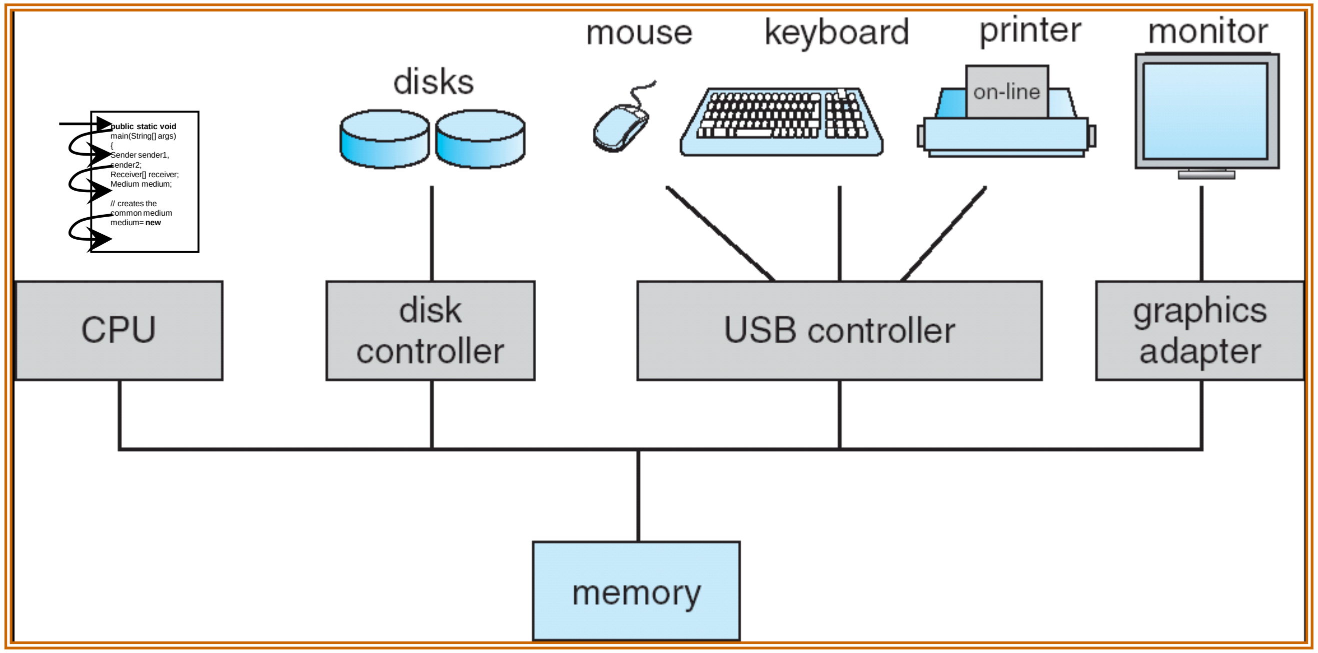
Single-Process System

- One process



```
public static void
main(String[] args) {
Sender sender1, sender2;
Receiver[] receiver;
Medium medium;

// creates the common medium
medium= new VirtualMedium();

// creates a sender that
continously sends "Hello World"
sender1= new public static void
main(String[] args) {
Sender sender1, sender2;
Receiver[] receiver;
Medium medium;

// creates the common medium
medium= new VirtualMedium();

// creates a sender that
continously sends "Hello World"
sender1= new
Sender("Sender1", new
CSMACA(medium));
Sender("Sender1", new
CSMACA(medium));
sender1.setMessage("HelloWor
ld");
```

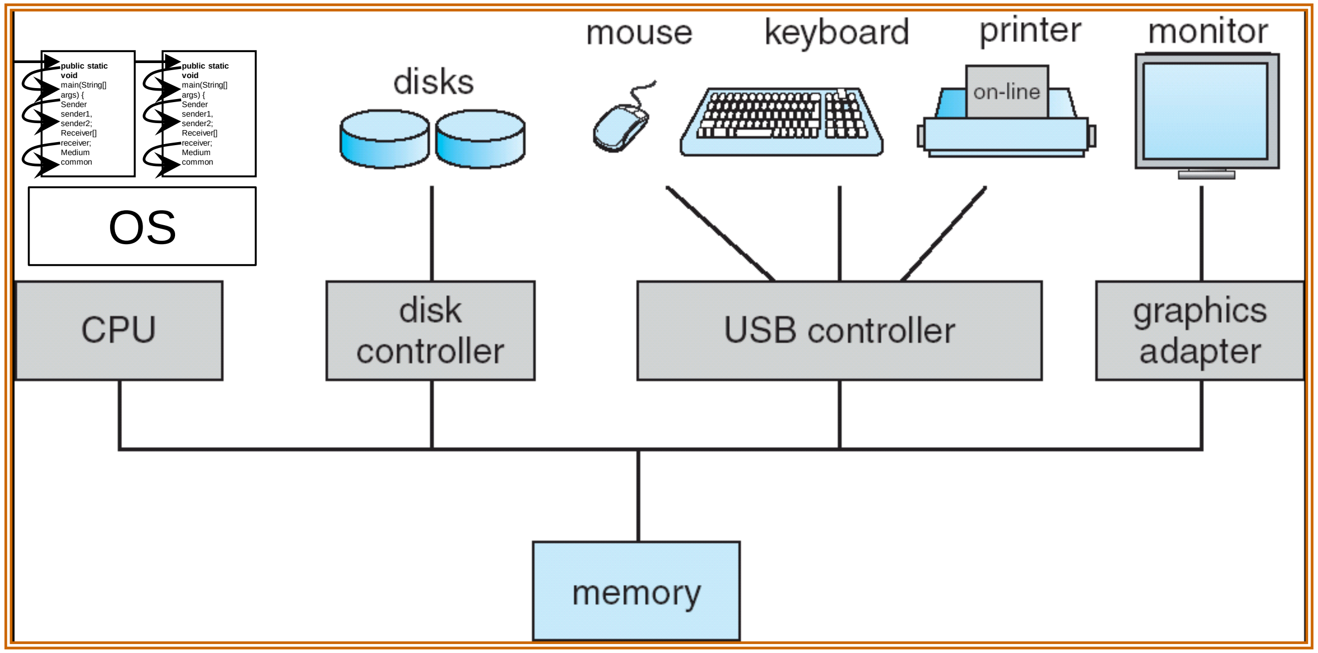
Single Program – Complete Control



```
public static void
main(String[] args)
{
Sender sender1,
sender2;
Receiver[] receiver;
Medium medium;

// creates the
common medium
medium= new
```

* Figure is courtesy of Silberschatz, Galvin, Gagne

OS & Multiple Programs → Chaos



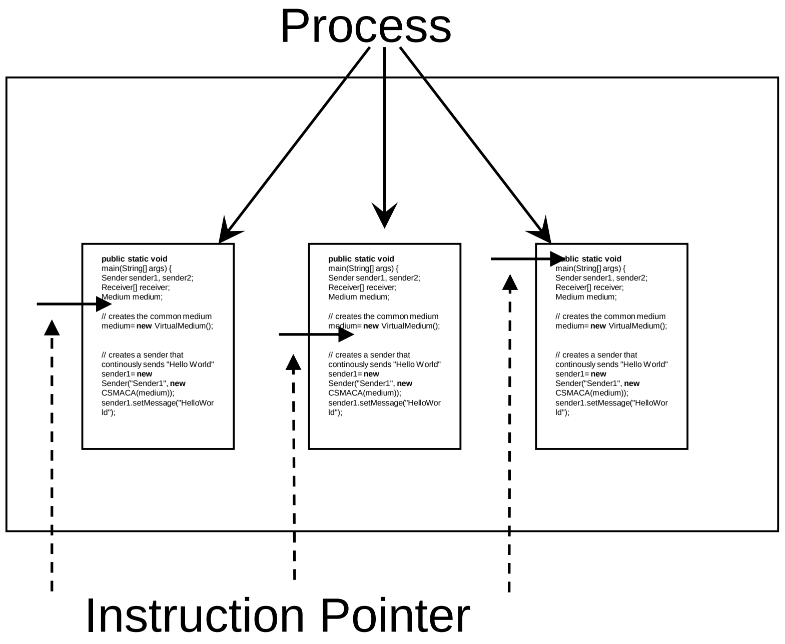
```
public static void main(String[] args) {
Sender sender1, sender2;
Receiver[] receiver;
Medium common
```

* Figure is courtesy of Silberschatz, Galvin, Gagne

Processes

- Separate address spaces
- Registers per process

- Problem:
  - Switching between processes

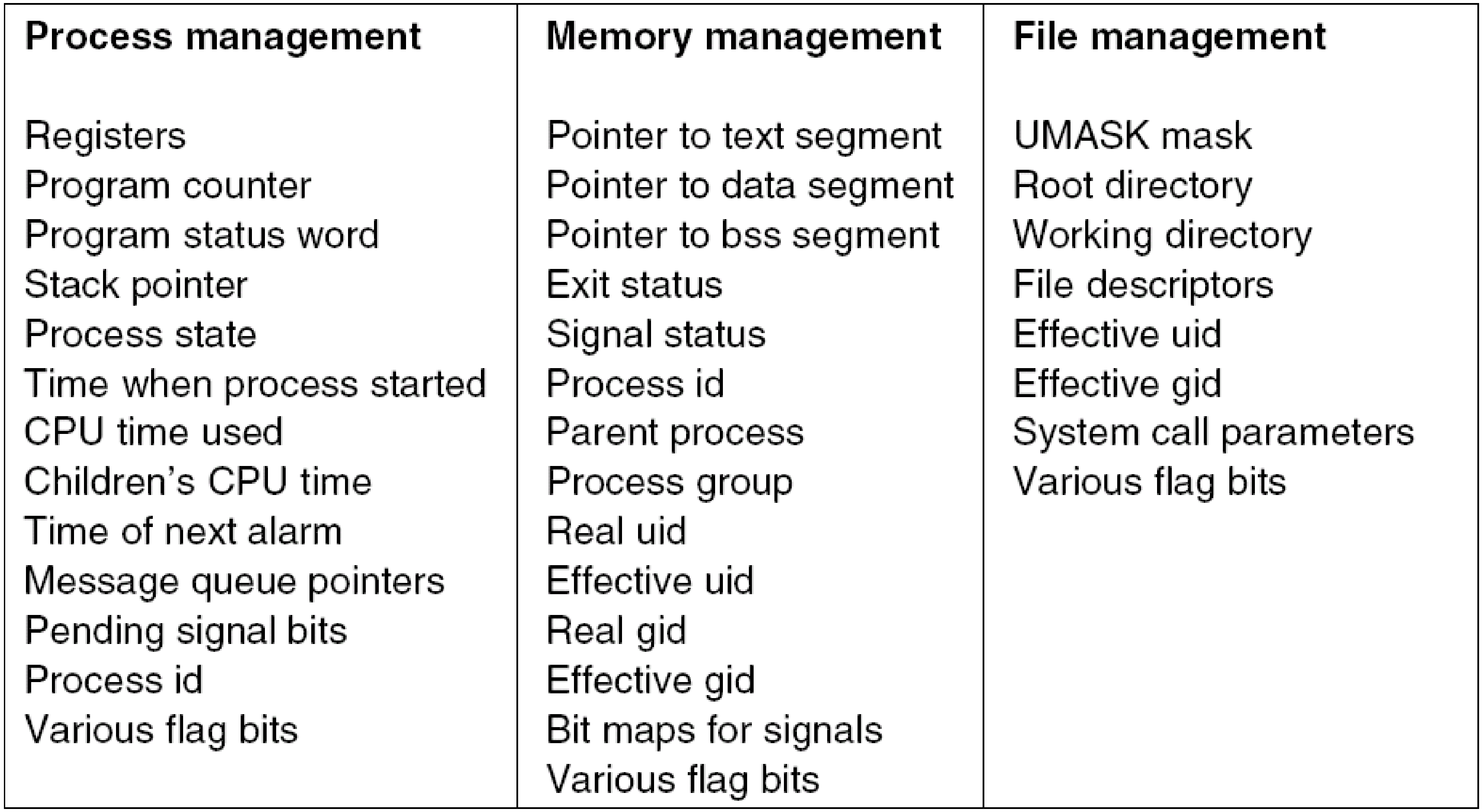


```
public static void
main(String[] args) {
Sender sender1, sender2;
Receiver[] receiver;
Medium medium;

// creates the common medium
medium= new VirtualMedium();

// creates a sender that
continously sends "Hello World"
sender1= new
Sender("Sender1", new
CSMACA(medium));
sender1.setMessage("HelloWor
ld");
```

```
public static void
main(String[] args) {
Sender sender1, sender2;
Receiver[] receiver;
Medium medium;

// creates the common medium
medium= new VirtualMedium();

// creates a sender that
continously sends "Hello World"
sender1= new
Sender("Sender1", new
CSMACA(medium));
sender1.setMessage("HelloWor
ld");
```

```
public static void
main(String[] args) {
Sender sender1, sender2;
Receiver[] receiver;
Medium medium;

// creates the common medium
medium= new VirtualMedium();

// creates a sender that
continously sends "Hello World"
sender1= new
Sender("Sender1", new
CSMACA(medium));
sender1.setMessage("HelloWor
ld");
```

Per-Process Details



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

* Figure is courtesy of A. Tanenbaum

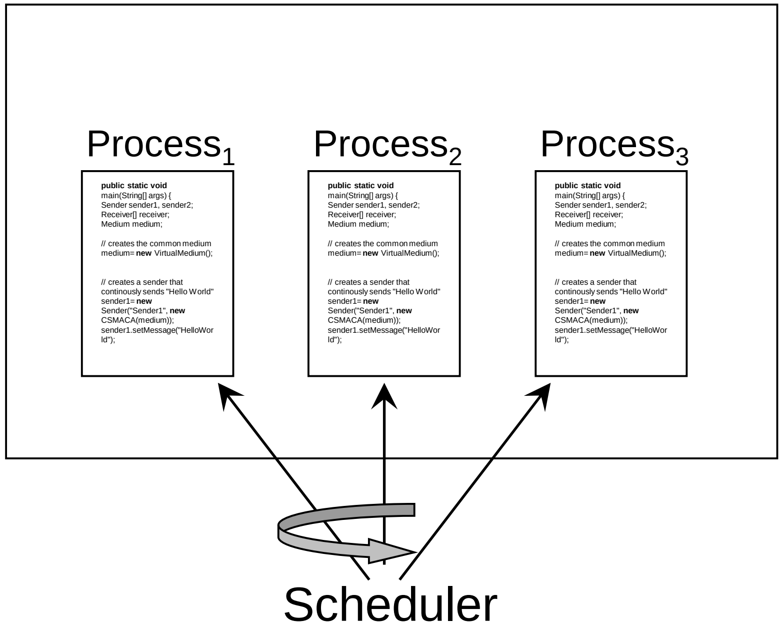
Process Switching

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

- Switching Virtual Memory

  ↓

  Overhead!



Process$_1$

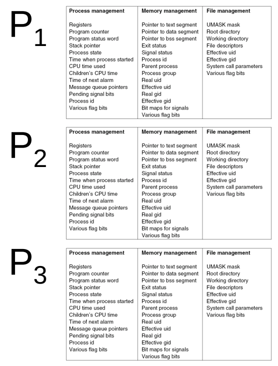
```
public static void
main(String[] args) {
Sender sender1, sender2;
Receiver[] receiver;
Medium medium;

// creates the common medium
medium= new VirtualMedium();


// creates a sender that
continously sends "Hello World"
sender1= new
Sender("Sender1", new
CSMACA(medium));
sender1.setMessage("HelloWor
ld");
```

Process$_2$

```
public static void
main(String[] args) {
Sender sender1, sender2;
Receiver[] receiver;
Medium medium;

// creates the common medium
medium= new VirtualMedium();


// creates a sender that
continously sends "Hello World"
sender1= new
Sender("Sender1", new
CSMACA(medium));
sender1.setMessage("HelloWor
ld");
```

Process$_3$

```
public static void
main(String[] args) {
Sender sender1, sender2;
Receiver[] receiver;
Medium medium;

// creates the common medium
medium= new VirtualMedium();


// creates a sender that
continously sends "Hello World"
sender1= new
Sender("Sender1", new
CSMACA(medium));
sender1.setMessage("HelloWor
ld");
```



P$_1$

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

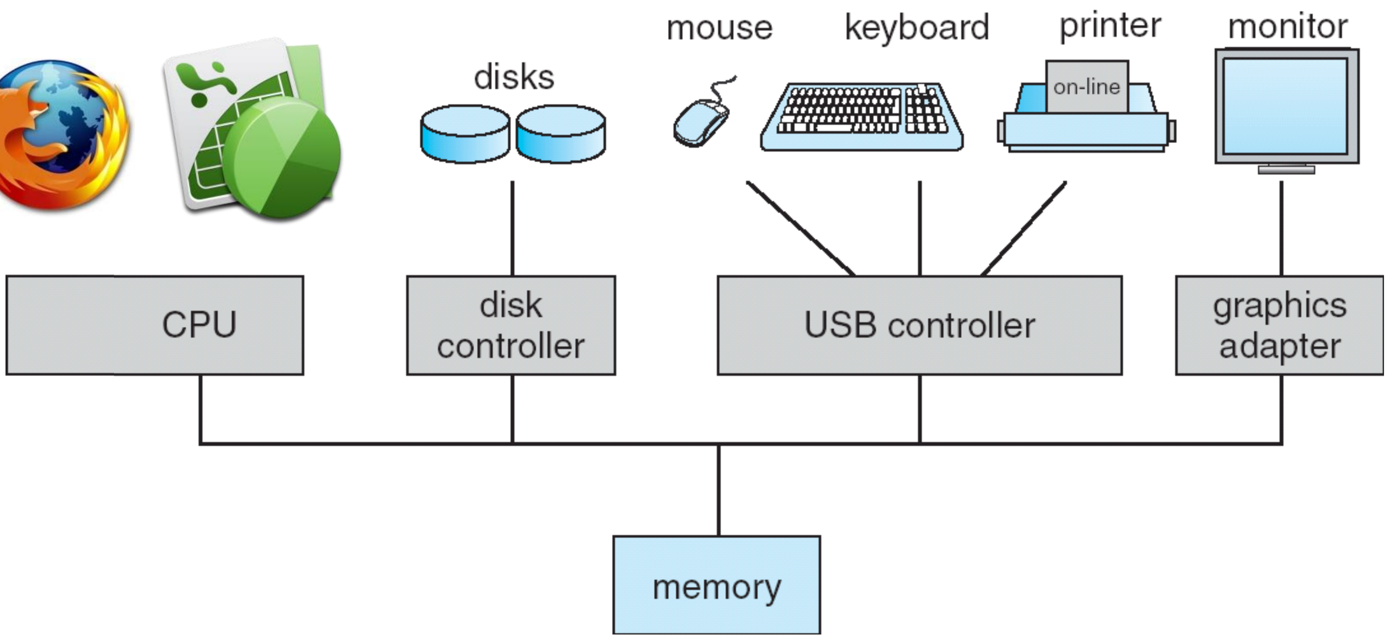
P$_2$

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

P$_3$

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

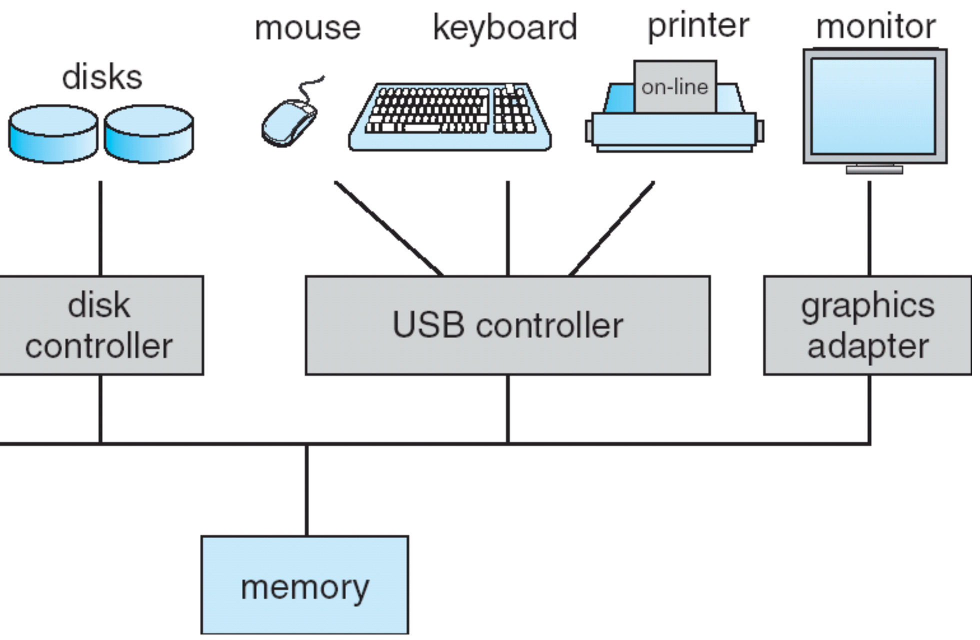
Switching Programs

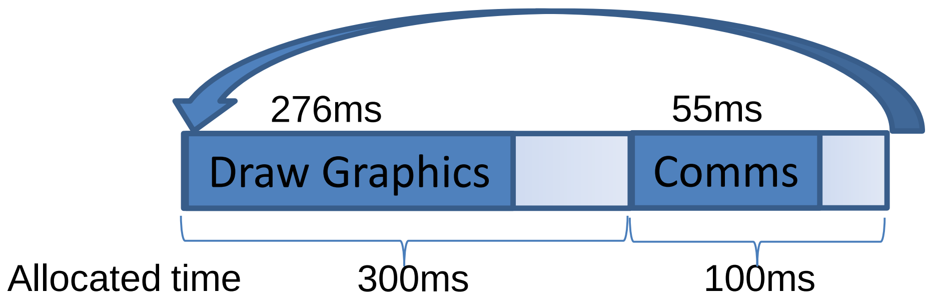


Switching Tasks in a Program

Comms with other players

Redrawing graphics



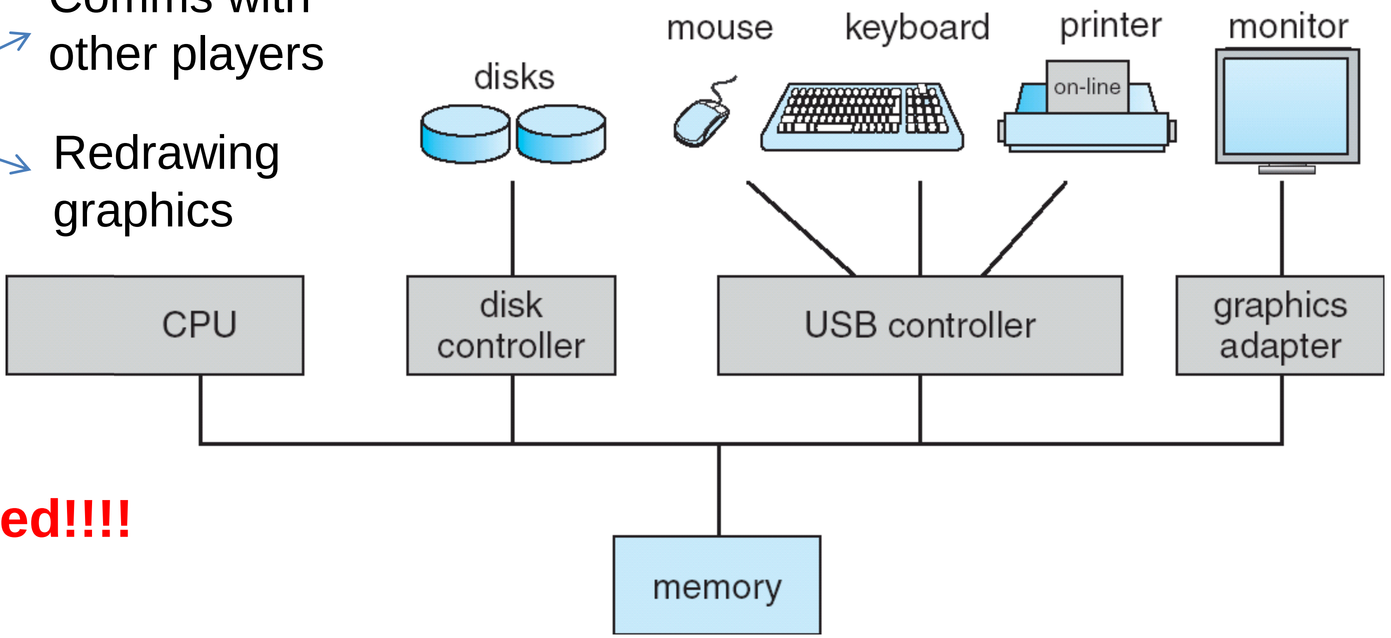


- Every 400ms the graphic will be redrawn
- It is important that the tasks are shorter than the allocated time

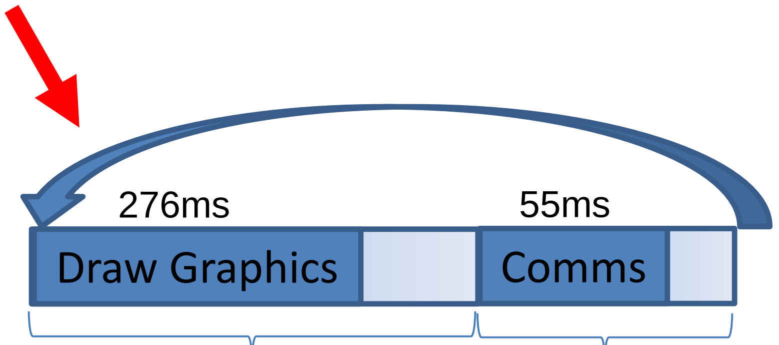
Switching Tasks in a Program

Comms with other players

Redrawing graphics



Tightly coupled!!!!

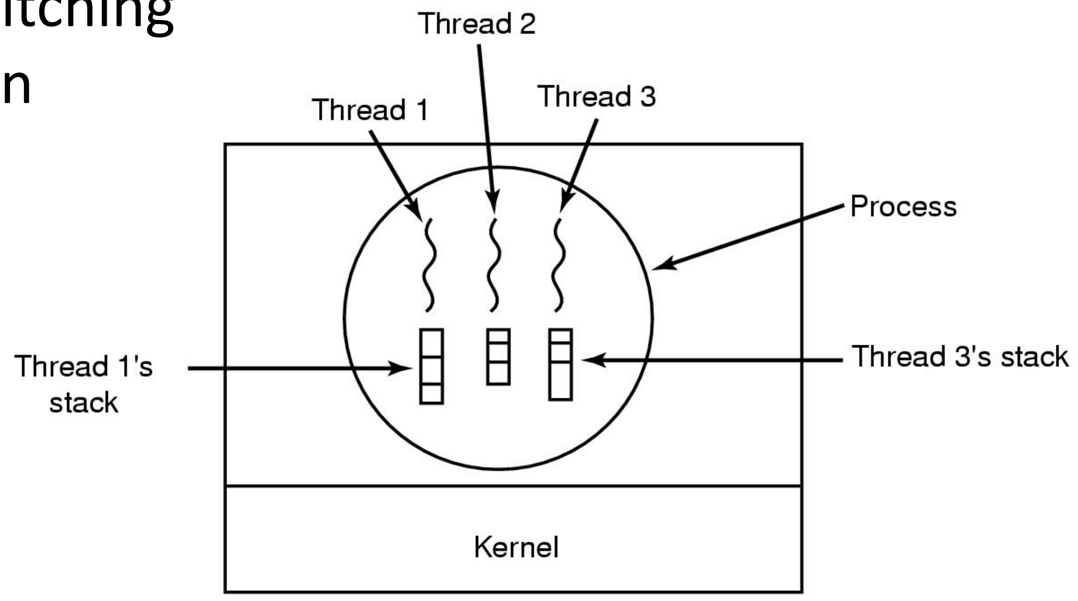


Allocated time 300ms 100ms

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

Threads

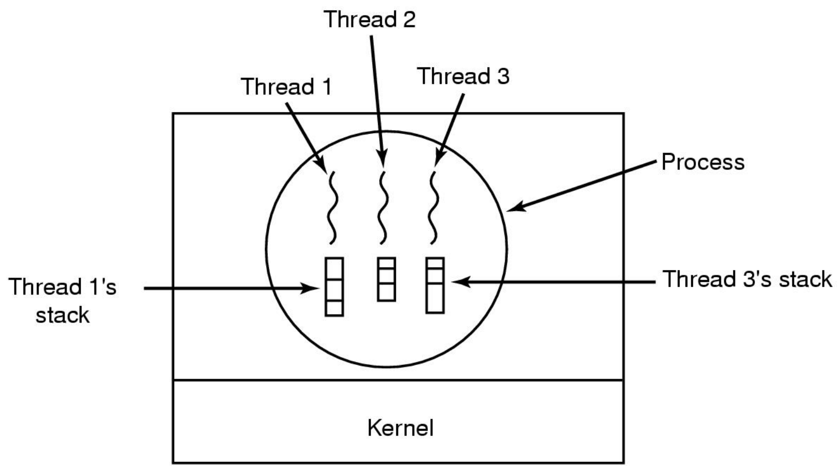
- Lightweight processes
- Share same address space
- Less overhead for switching between threads than between processes



* Figure is courtesy of A. Tanenbaum

Threads

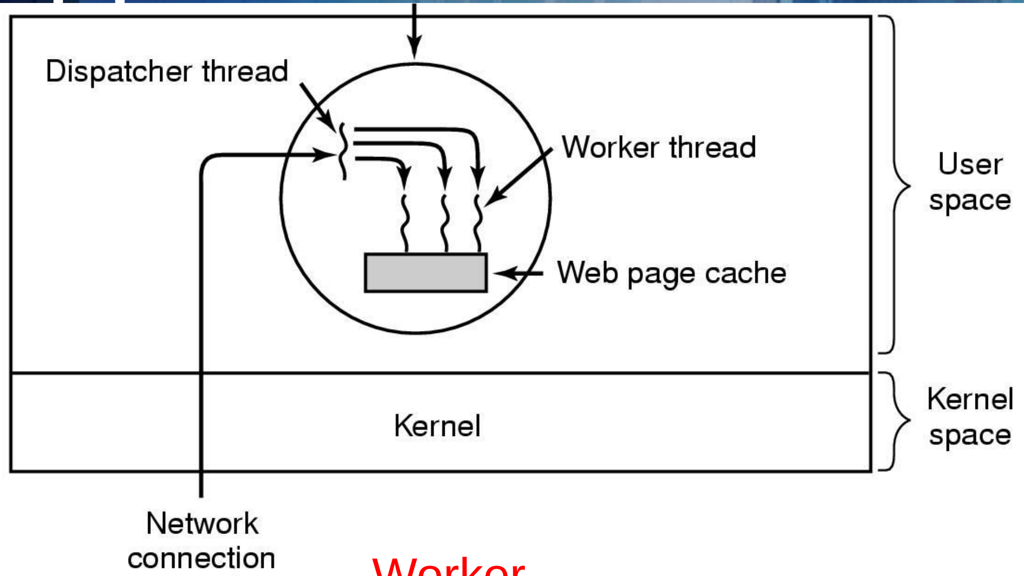
- Lightweight processes
- Share same address space
- Less overhead for switching between threads than between processes



Per process items	**Per thread items**
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

* Figure is courtesy of A. Tanenbaum

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)

* Figure is courtesy of A. Tanenbaum

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

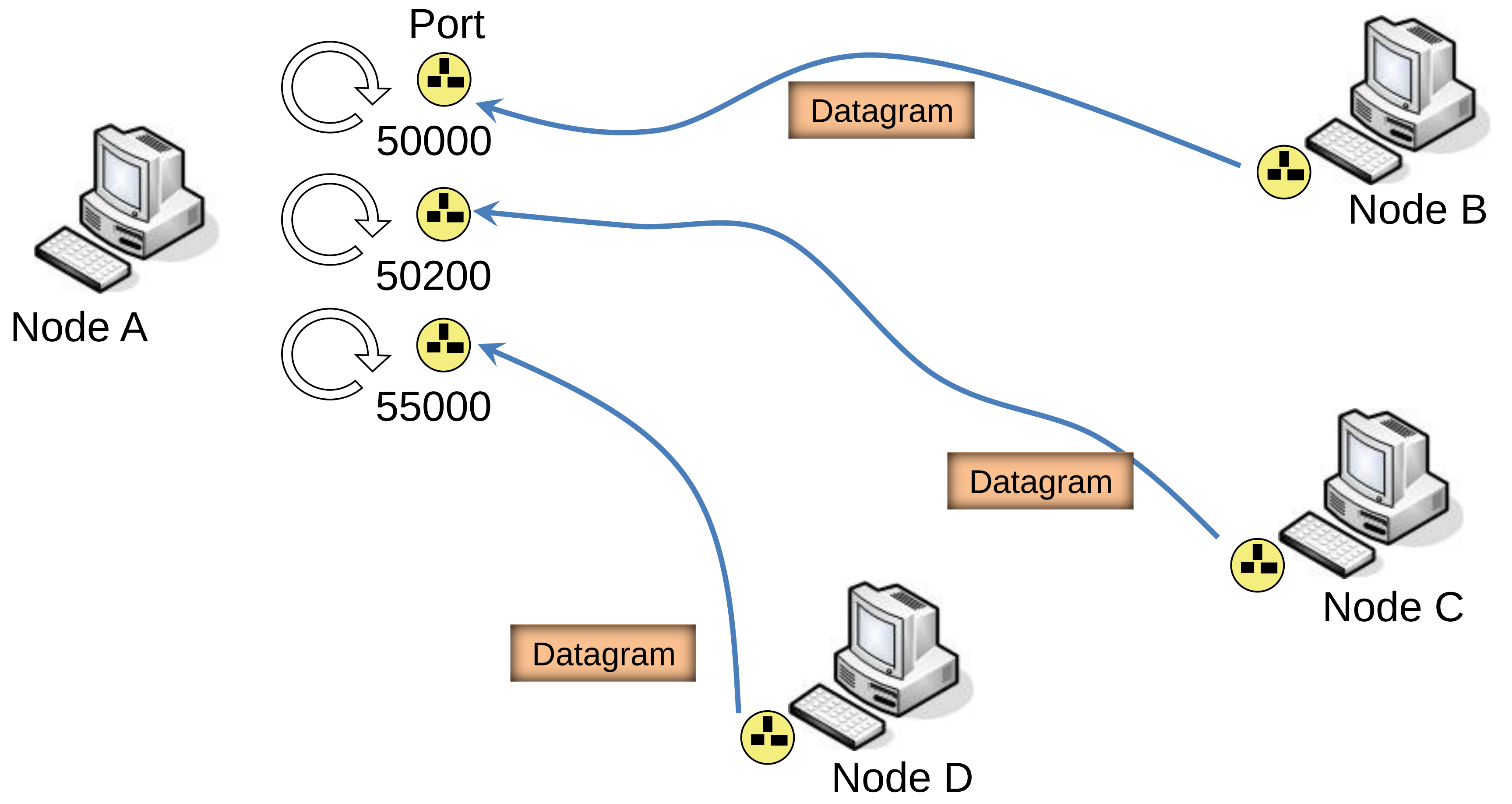
```
SocketThread t1, t2, t3;

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();
```

$\rbrace \longrightarrow$ Insert thread into list of running threads and execute “run” method

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();
```

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

Possible Output

time

```
T1: 11
T2: 11
T3: 11
T1: 12
T2: 12
T3: 12
...
```

or

time

```
T1: 11
T1: 12
T3: 11
T3: 12
T2: 11
T3: 13
...
```

or

time

```
T1: 11
T3: 11
T3: 12
T2: 11
T1: 12
T3: 13
...
```

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 {...
```

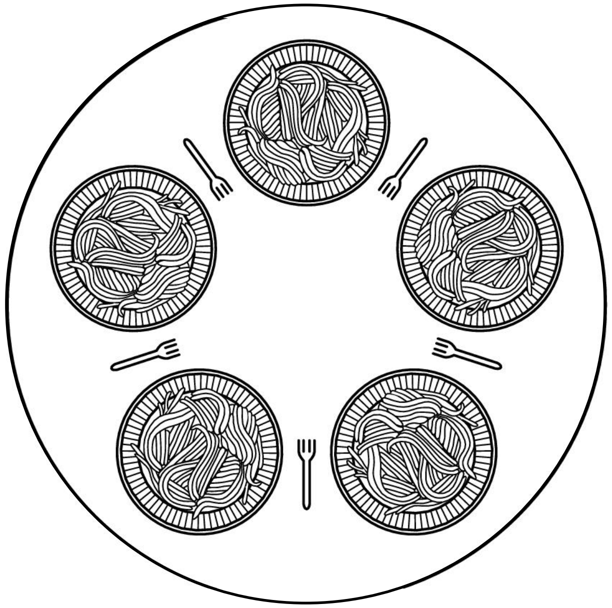
ERROR

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
Consumer

- 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--;
        }
    }
}
```

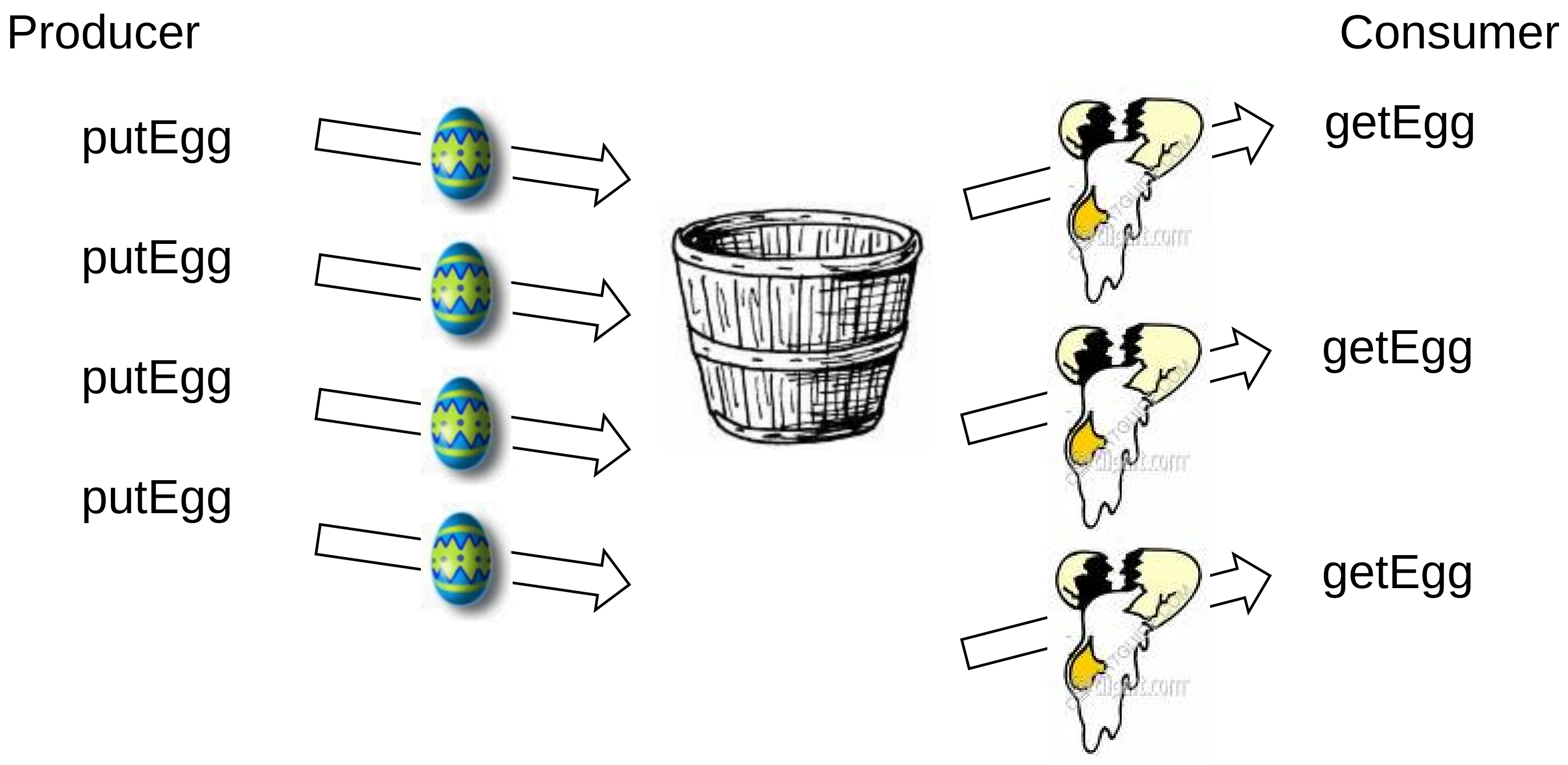
Producer-Consumer in Java III

```
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

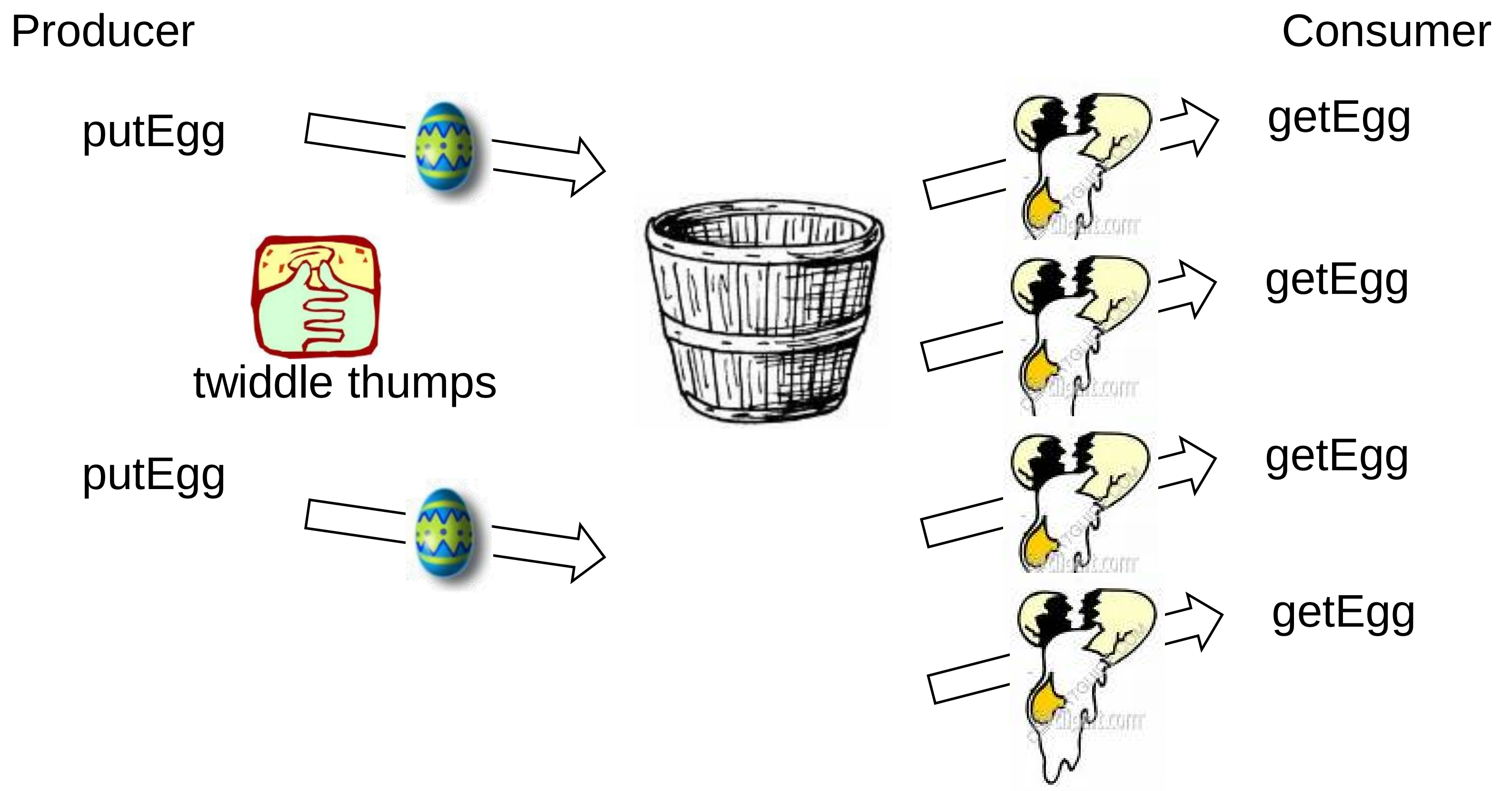
```
class TestSystem {

    public static void main (String[] args) {
            Producer producer;
            Consumer consumer;

            producer= new Producer();
            consumer= new Consumer();

            producer.start();
            consumer.start();
    }
}
```

Problem???



Execution is non-deterministic!

Problem???



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

Producer
Consumer

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

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

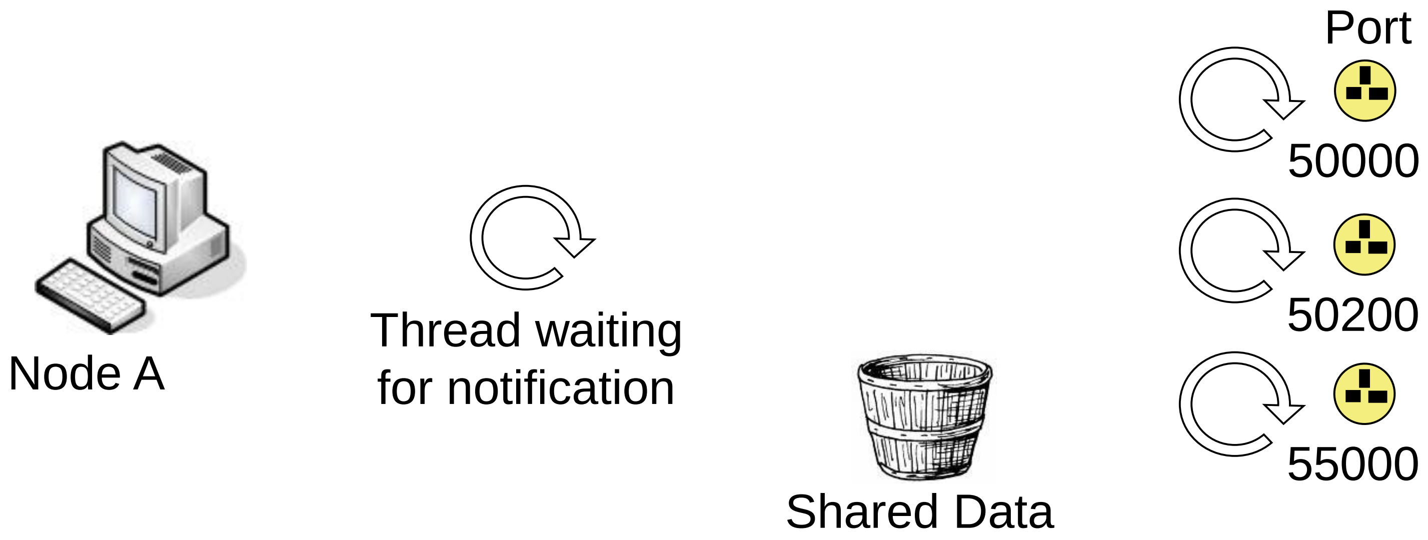
Producer-Consumer Problem

```
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!

Monitors in Java



```
synchronized void getSharedData() {
        wait();
        //do something
}
```

```
synchronized void changeSharedData() {
        // change data
        notifyAll();
}
```

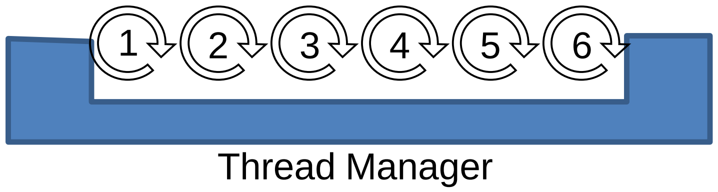
Only one thread can be in a synchronized method of a class at a given time.

Synchronized Methods

```
class SharedData {
   synchronized void put(Object o) {...}


   synchronized Object get() {...}


   synchronized void printContent() {...}
}
```

A Word of Warning

- Costs associated with Threads
  - Time for creation
  - Memory allocation
  - Garbage collection
  - etc
- Moderation is the key
- Thread pools
- Deadlocks!



Summary: Threads

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

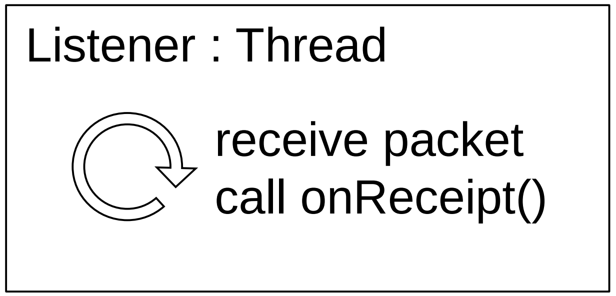
CS2031 Telecommunications II

Event-based Programming

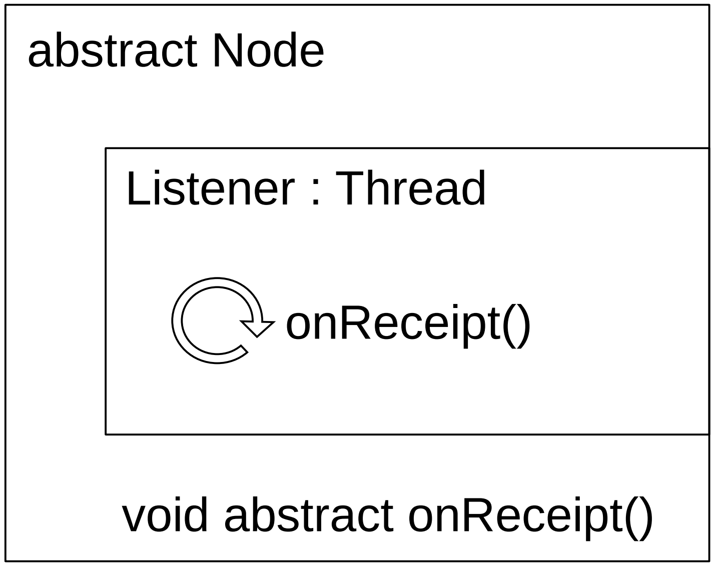
“Event-based Programming”

```
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();}
}
```

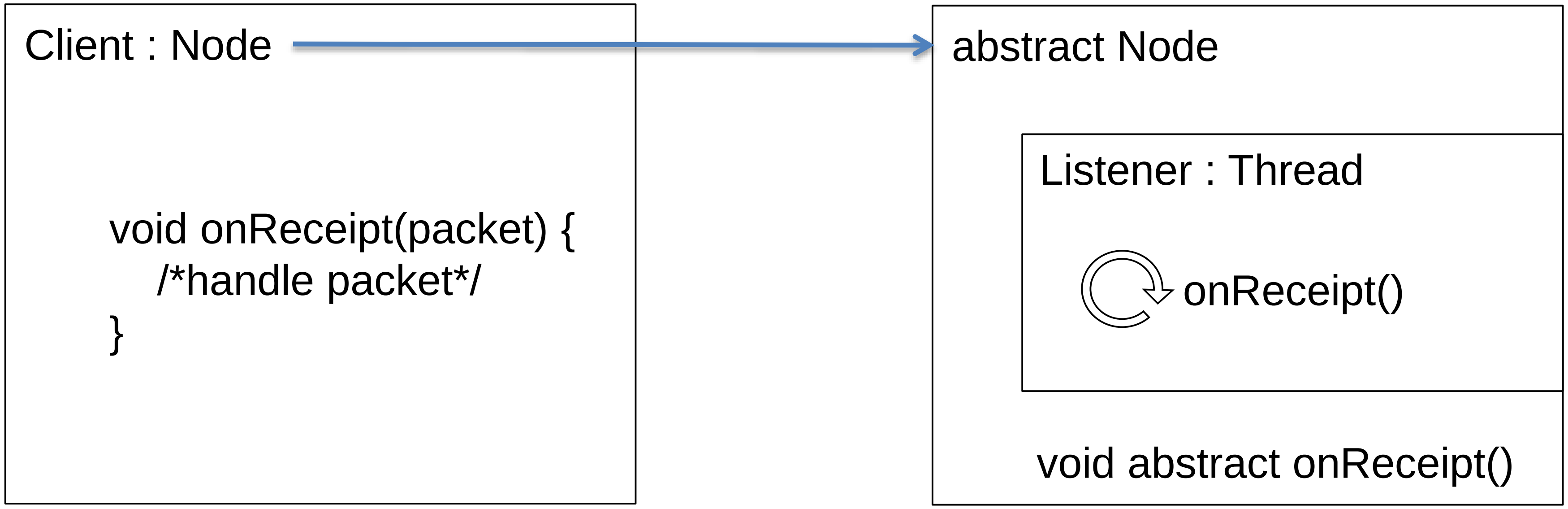
“Event-based Programming”



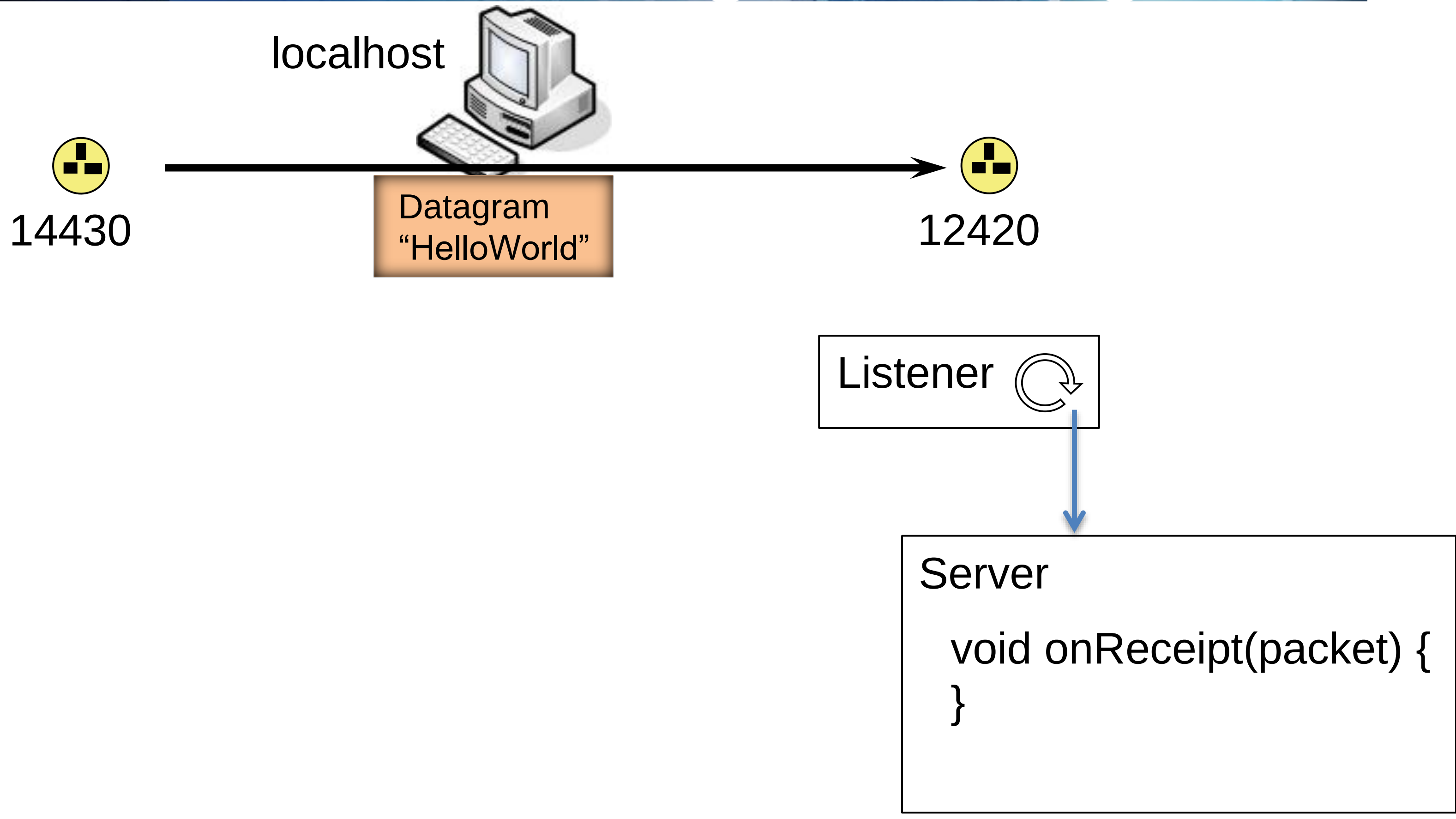
“Event-based Programming”



“Event-based Programming”



"Event-based Programming"



"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
Telecomms II