UNIVERSITY of WASHINGTON | BOTHELL

CSS430 Operating-System Structures

Textbook Chapter 2

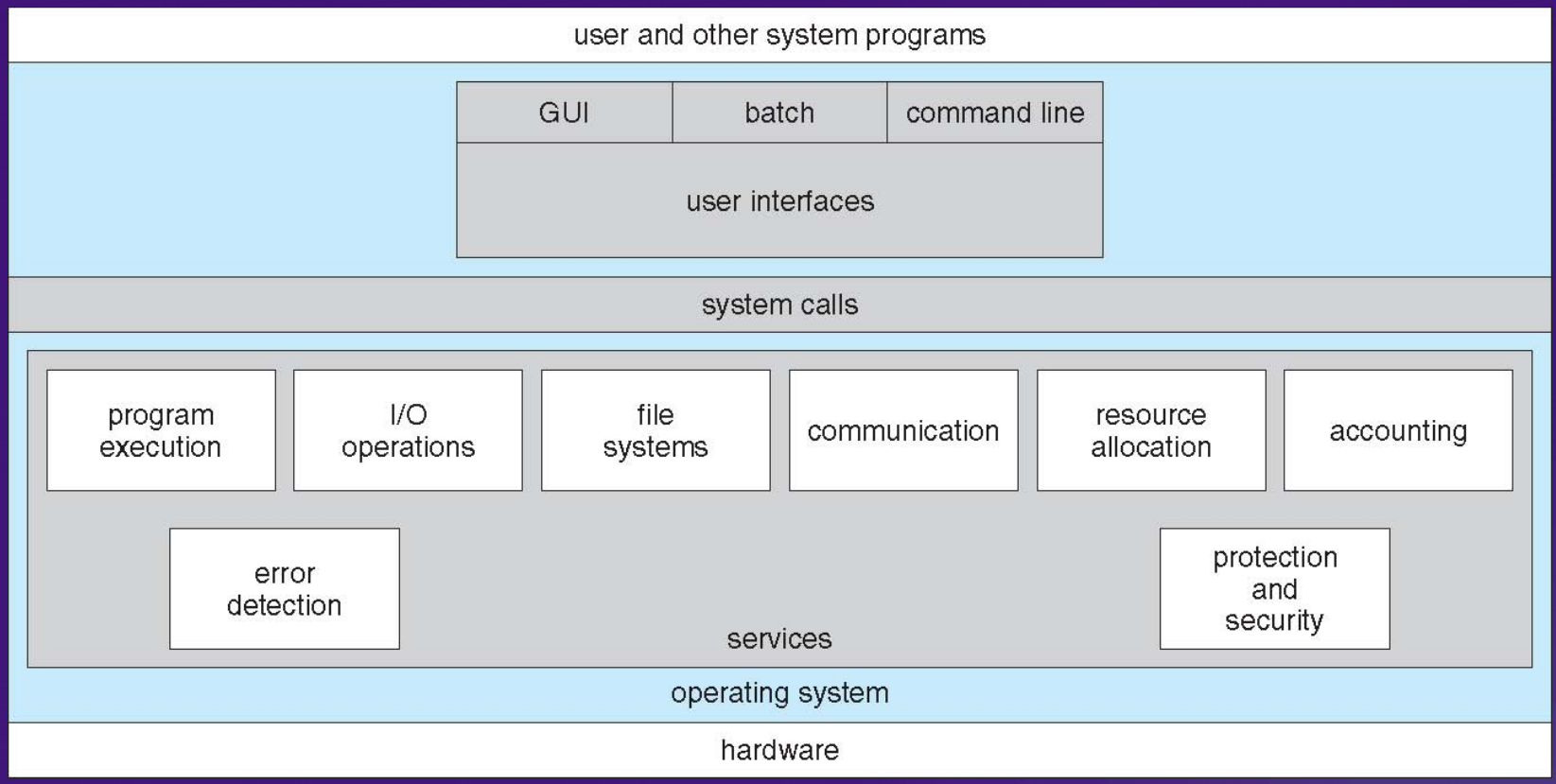
Instructor: Stephen G. Dame
e-mail: sdame@uw.edu

These slides were adapted from the OSC textbook slides (Silberschatz, Galvin, and Gagne), Professor Munehiro Fukuda and the instructor's class materials.

"People who are more than casually interested in computers should have at least some idea of what the underlying hardware is like. Otherwise the programs they write will be pretty weird." - D. Knuth

Donald Ervin Knuth (born 1938) is an American computer scientist, Professor Emeritus at Stanford University, and winner of the 1974 Turing Award.

A View of OS Services



OS Features

❖ Process Management	Week 2-5
❖ Main Memory Management	Week 6-7
❖ File Management	Week 8-9
❖ Secondary-Storage Management	if time allows
❖ I/O System Management	if time allows
❖ Networking	CSS432
❖ Protection System	Week 10
❖ Command-Interpreter System	Today

Process Management

- A *process* is a program in execution. A process needs **CPU time**, **memory**, **files**, and **I/O devices**, to accomplish its task.
- The operating system is responsible for
  1. Process **creation and deletion** (starting and terminating a program execution)
  2. Process **suspension and resumption** (letting a program wait for an I/O operation or a next turn)
  3. Process **synchronization** (letting a program wait for another program's termination)
  4. Process **communication** (allowing a program to send/receive data from another executing program)

Memory Management

- Memory is a large array of words or bytes, each with its own address.
- Main memory is a volatile data storage shared by the CPU and I/O devices.
- The operating system is responsible for:
  ① Keeping track of which parts of memory are currently being used and by whom.
  ② Deciding which processes to load when memory space becomes available.
  ③ Allocating and deallocating memory space as needed.

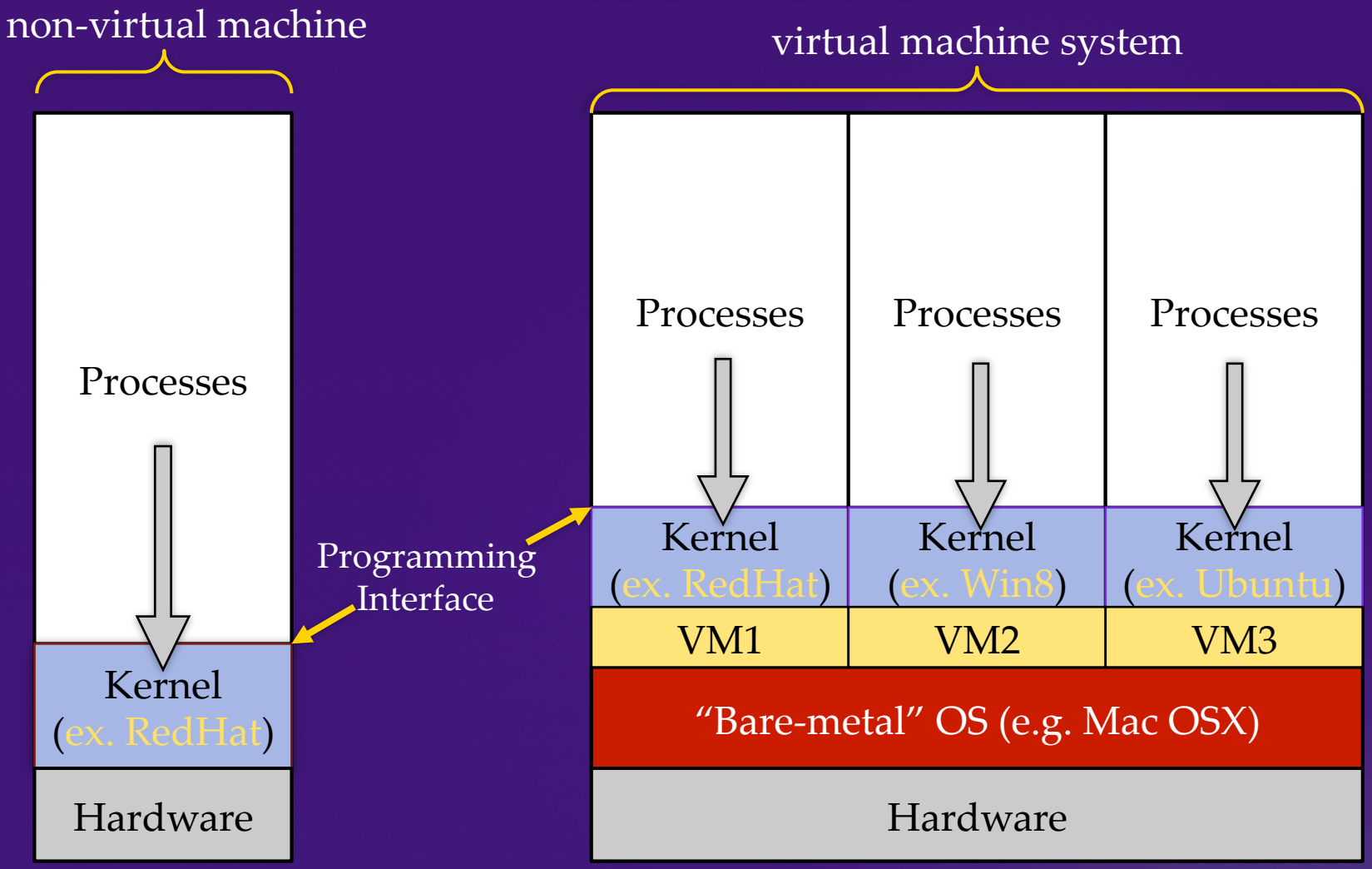
File Management

- Files represent programs and data.
- The operating system is responsible for:
  ① File creation and deletion.
  ② Directory creation and deletion.
  ③ Support of primitives for manipulating files and directories (open, read, write, seek, and close).
  ④ Mapping files to secondary storage (HDrive) and tertiary storage (R/W CDs, tapes, Flash Drv).
  ⑤ File backup on stable (nonvolatile) storage media.

Other Management Functions

- I/O Systems:
  - Buffering, caching, and spooling of I/O data (I/O devices are typically slow.)
  - Device drivers (program and control of peripheral devices)
- Secondary/Mass-Storages:
  - Disk management (for free and allocated spaces)
  - Disk scheduling (for an optical sequence of disk accesses)
  - Swap-space management (disk area used as virtual memory)
- Network:
  - Supporting various network protocols: TCP/IP, FTP, NFS, and HTTP, etc.
- Protection/Security:
  - Authentication (password, defending ext. I/O, bytecode verifier)
  - Access authorization (access mode, javsandbox a model)
  - Cryptography

Virtualization



VM Ware, Virtual Box, Parallels, etc.

Discussion 1

1. List at 3 or more reasons for why virtual machines are beneficial.

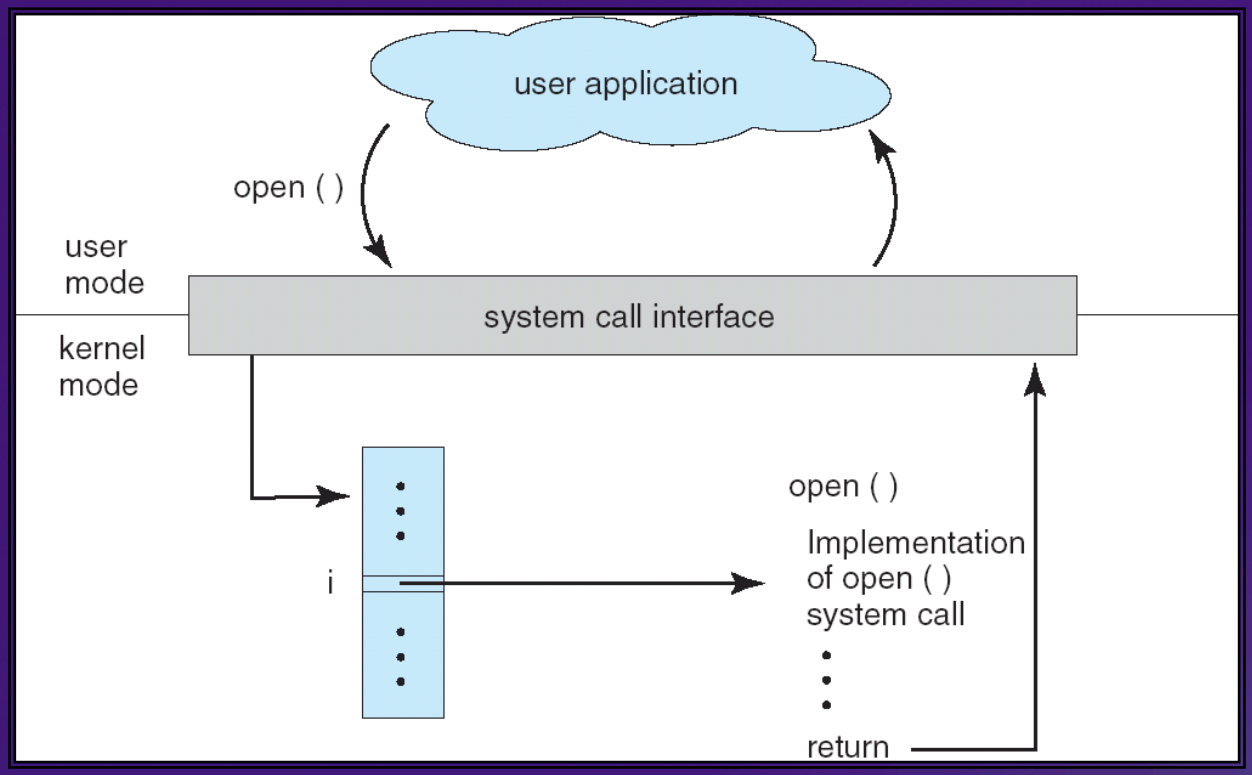
2. What would be a reason to share a file system between a host OS and VM OS?

3. What are some of the challenges of implementation of VMs?

4. Are VMs possible without some level of HW support from the host OS?

System Calls

All management functions in slides 4-7 must be performed through a **system call**.



System Calls (Continued)

Process control

- end, abort
- load, execute
- create process, terminate process
- get process attributes, set process attributes
- wait for time
- wait event, signal event
- allocate and free memory

File management

- create file, delete file
- open, close
- read, write, reposition
- get file attributes, set file attributes

Device management

- request device, release device
- read, write, reposition
- get device attributes, set device attributes
- logically attach or detach devices

Information maintenance

- get time or date, set time or date
- get system data, set system data
- get process, file, or device attributes
- set process, file, or device attributes

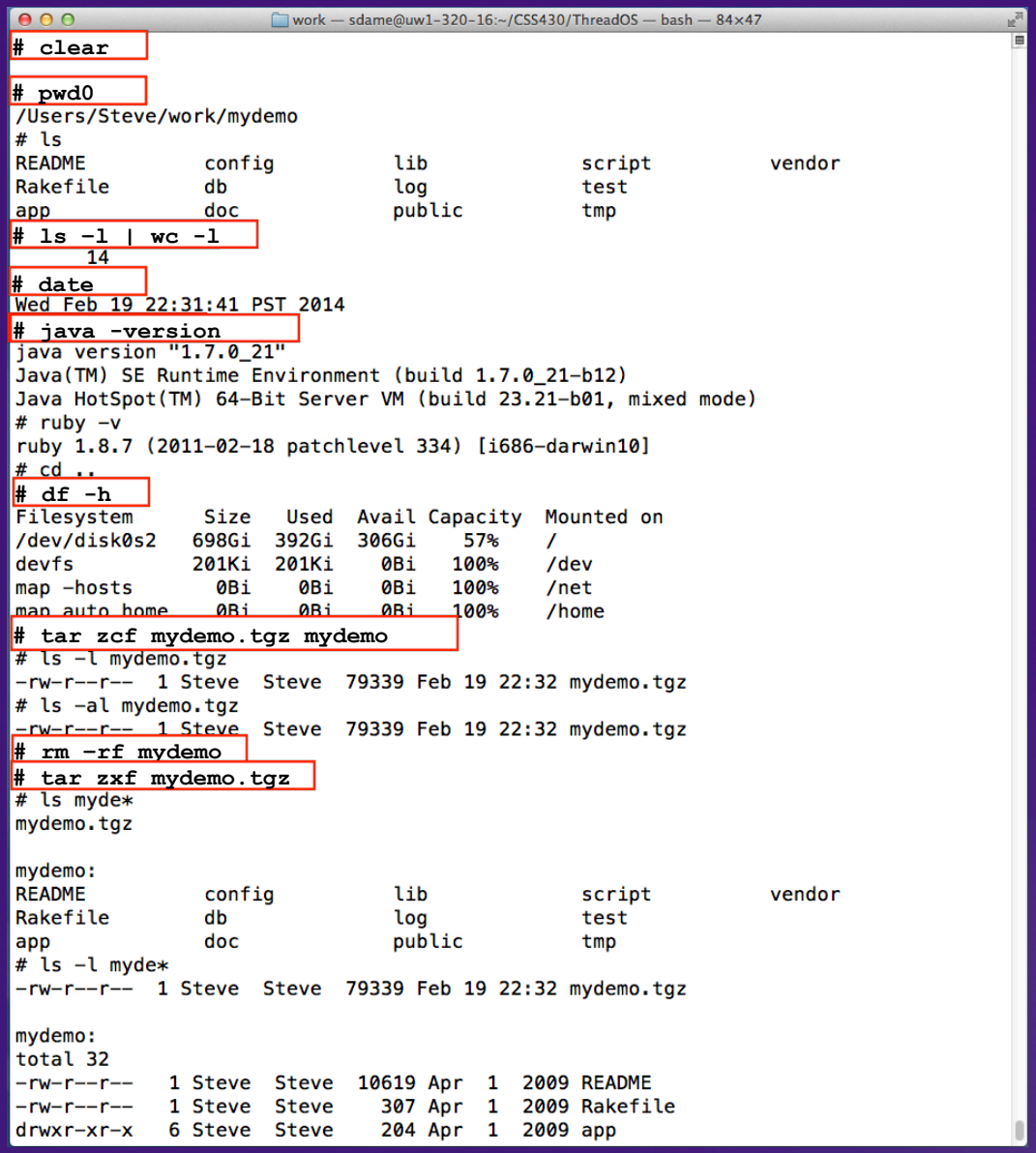
Communications

- create, delete communication connection
- send, receive messages
- transfer status information
- attach or detach remote devices

Command Interpreters

- The program that reads and interprets control statements
  - command-line interpreter (in DOS)
  - shell (in UNIX)
- What control statements can you pass the command interpreter?
  - Program execution: a.out, g++, vim, emacs
  - Process management: ps, kill, sleep, top, nice, pstack
  - I/O operations: lpr, clear, lprm, mt
  - File-system manipulation: ls, mkdir, mv, rm, chmod, [u]mount
  - Communication: write, ping, mesg

Bash Shell Command Interpreter



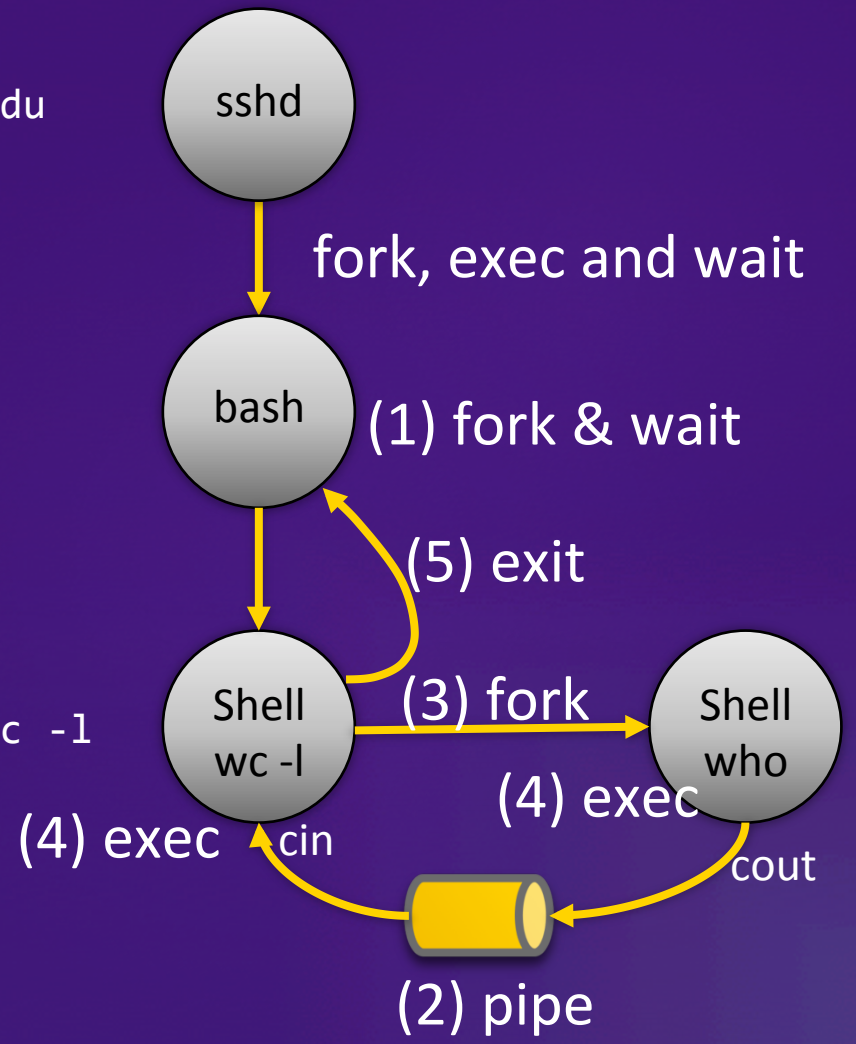
Shell (bash)

Fork, exec, wait and dup are System calls

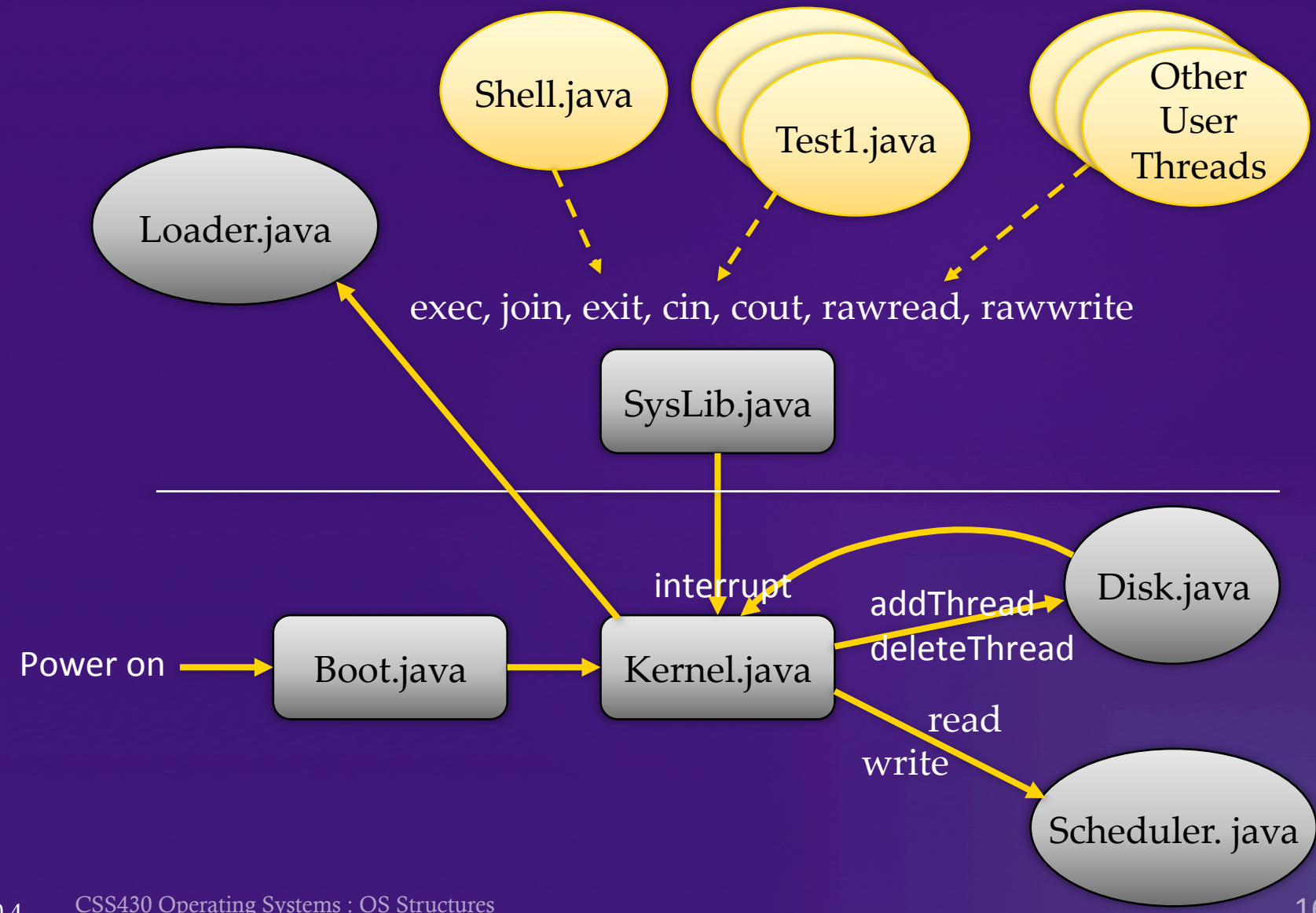
```
# ssh uwnetid@uw1-320-15.uwb.edu

uwnetid@uw1-320-15 ~:$

uwnetid@uw1-320-15 ~:$ who | wc -l
```



CSS430-Unique ThreadOS



LXC

Discussion 2

Class Discussion

- Compare Windows OS to Linux OS, from the following viewpoints?

  ① Windows temporarily keeps deleted files in Recycle Bin, while Linux rm deletes them instantly.

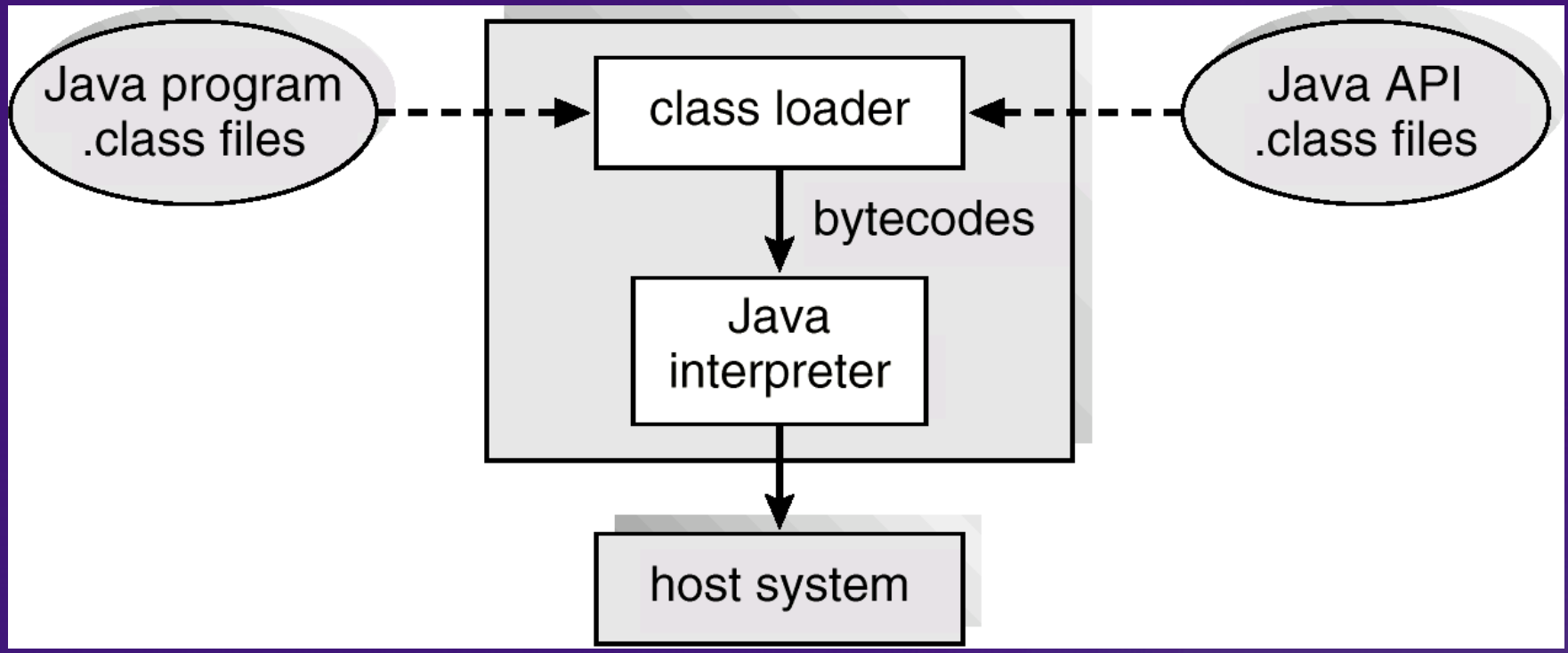
  ② Windows task manager allows us to kill processes with their program names, while Linux uses IDs to kill specific processes.

  ③ Windows launches most applications by double-clicking, while Linux needs a specific application to be typed from the command line (i.e. more command line script oriented).

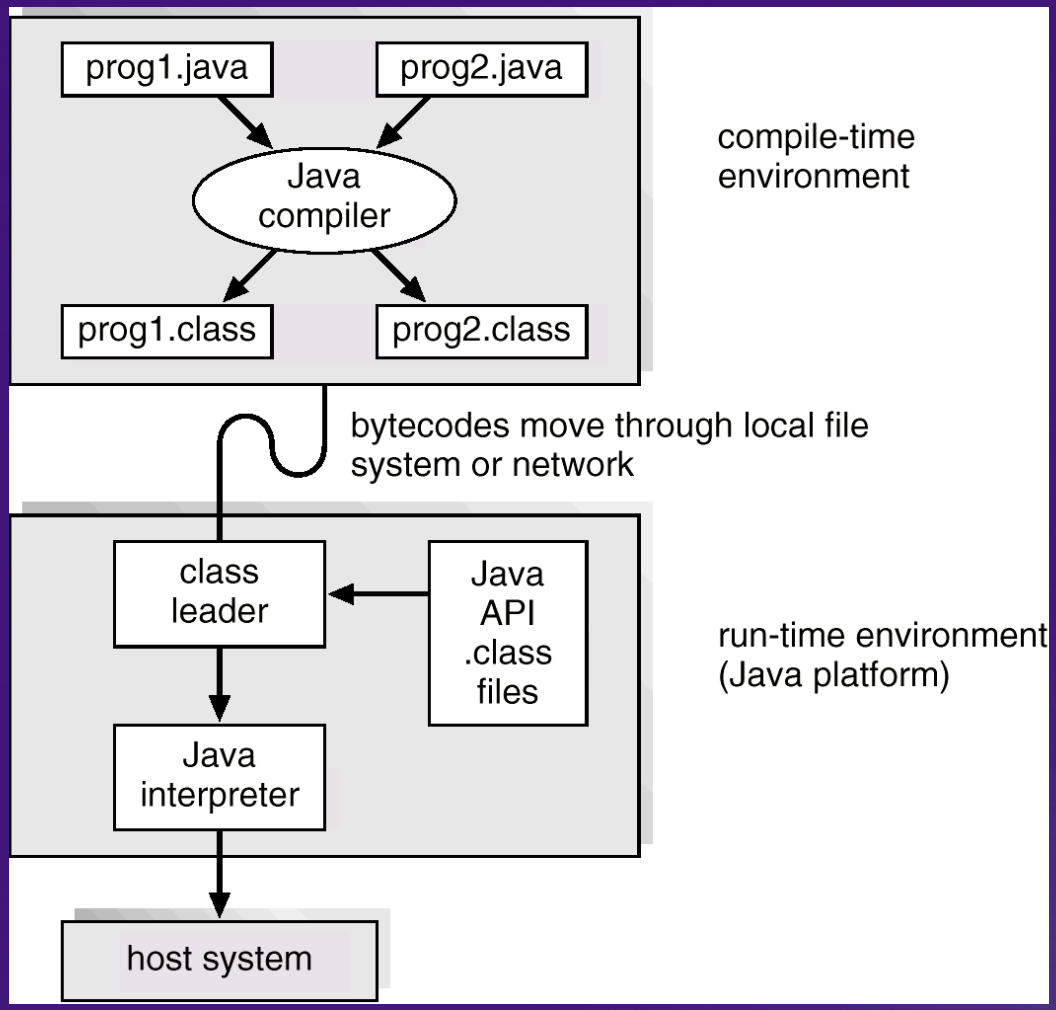
Java Technology

- Programming-language specification
  - C++-like object oriented programming language
  - No system-dependent descriptions
    - ✓ Variable sizes are universally defined over different machines
    - ✓ No system calls are supported
    - ✓ Automatic memory operations: no address concept and no delete
  - Multithreaded support
- Application-programming interfaces (API)
  - Various system-provided classes: graphics and I/O
- Virtual-machine specification
  - Interpretation of architecturally independent bytecode

Java Virtual Machine



Java Development Environment



Java Program

Hello — vim — 56×11

```
public class Hello {
   private String myName;
   Hello( String name ) { myName = name; }

   public void speak()
   {
      System.out.println("Hi! My name is " + myName);
   }
}
```

Hello class + a method

Hello — vim — 56×12

```
class HelloTest {
   public static void main( String[] args ) {
      Hello greeter = new Hello( "Von Neuman" );
      greeter.speak( );
   }
}
```

Test class has main()

Hello — bash — 56×5

```
# javac *.java
# java HelloTest
Hi! My name is Von Neuman
#
```

compile & execute HelloTest

Names and Packages

C++	Java
• No rules on class names • No correlation between class and file names • Headers predefine various useful class interfaces. • #include reads in a header file.	• Class names in `MixedCase` starting with a capital letter • Class and the corresponding file must have the same name. • Packages predefine various useful classes. • import omits the full package name.

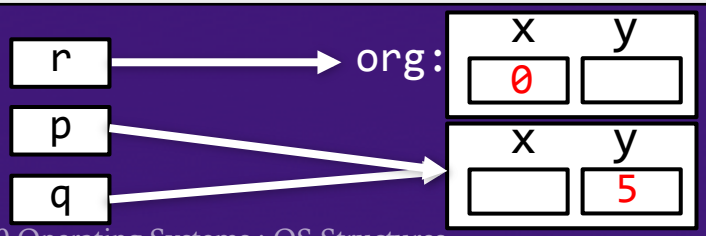
Filename: hello.cpp
`#include <cstdlib>`
`class hi {`
`  hi( ) {`
`  int r = rand( );`
`  }`
`}`
	Filename hello.java
	`import java.util.*;`
	`class Hello {`
	`  Hello( ) {`
	`  Random r = new Random( );`
	`  }`
	`}`

Values, Objects, and Pointers

C++

- Variables:
  - bool(true or false)
  - char(8bits), short(16bits), int(32bits), long(32bits),
  - float(32bits), double(64bits)
  - Pointers:
    *, &, and -> operators

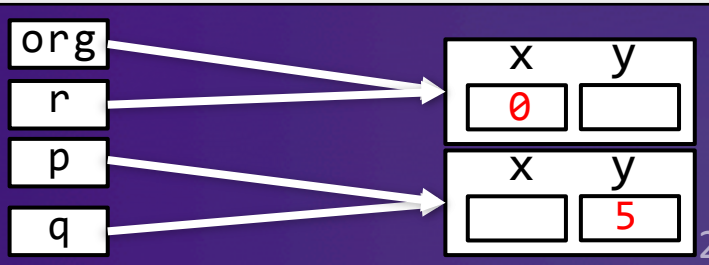
```
class Pair { int x, y };
Pair org;
Pair *p, *q, *r;
org.x = 0;
p = new Pair;
p->y = 5;
q = p;
r = &org;
```



Java

- Variables:
- boolean(true or false)
- byte(8bits), char(16bits), short(16bits), int(32bits), long(64bits)
- float(32bits), and double(64bits)
- Pointers: NO *, &, and ->

```
class Pair { int x, y };
Pair org = new Pair( );
Pair p, q, r;
org.x = 0;
p = new Pair( );
p.y = 5;
q = p;
r = org;
```



Pointers (Continued…)

C++	Java
➢ Function arguments: ▪ Primitive types: call by value ▪ Objects: call by reference (& needed) ➢ Garbage collection: ▪ delete needed (memory leaks!)	➢ Function arguments: ▪ Primitive types: call by value ▪ Objects: call by reference (**no** & needed) ➢ Garbage collection: ▪ **no** delete needed

`p = new Pair(2,7);`
`foo(&p);`
`//...`
`delete p;`
`P = new Pair(0,1);`
	`p = new Pair(2,7);`
	`//...`
	`foo(p);`
	`p = new Pair(0,1);`
	`// previous object p is deleted by system`

Public, Protected, Private, Static, and Final

C++	Java
▪ public ▪ protected ▪ private (default) ▪ Static (used as shared variables/functions) ▪ const	▪ public (default) ▪ protected ▪ Private ▪ static (used as shared and global variables/functions) ▪ final

Arrays and Strings

C++	Java

▪ Array name
✓ Points to address of the 1st Elem
▪ Array size
✓ Have to memorize how long it is.
✓ Cannot change the size.
 	
`int a[], *b;`
`a = new int[10];`
`b = a;`
 	
▪ `string class`
	▪ Array name
	✓ Points to the entire array object
	▪ Array size
	✓ Final field length returns the size.
	✓ Cannot change the size.
	
	`int a[];`
	`a = new int[10];`
	`int[] b = a;`
	
	▪ `String class`

Visit java.sun.com for details

Constructors and Overloading

C++	Java
▪ Object construction using new ✓ No parentheses needed if no arguments given ▪ Multiple constructors ✓ Allowed ▪ Overloading ✓ Including operators	▪ Object construction using new ✓ Parentheses always needed even if no arguments given ▪ Multiple constructors ✓ Allowed ▪ Overloading ✓ Operators are not overloaded

Inheritance, Interfaces, and Casts

C++

- Inheritance

```
class Derived : public Base {…}
```

  - ✓ Multiple inheritance allowed
  - ✓ Pure virtual functions for abstract classes

```
class Abstract {
   virtual func() = 0;
}
```

  - ✓ Constructors called from the base class
- Cast
  (typeName)var or typeName(var)

Java

- Inheritance

```
class Derived extends Base {…}
```

  - ✓ Single inheritance only (all objects are derived from Object class)
  - ✓ Methods without a body can be described in an interface

```
interface Runnable {
   void run();
}
```

  - ✓ Methods without a body can be described in an interface
- Multiple interfaces are inherited.

```
class Derived implements
      Runnable {…}
```

- Cast: (typeName)var

Exceptions

- No core dump but exceptions occur in Java.
- Some API methods request you to catch exceptions

- Catching Exceptions: → recommended technique

```
public Disk( int blocks ) {
   try {
      FileInputStream ifstream = new FileInputStream("DISK");
   } catch ( FileNotFoundException e) {
      System.out.println(e);
   {
}
```

- Throwing Exceptions:

```
public Disk( int blocks ) throws FileNotFoundException {
      FileInputStream ifstream = new FileInputStream("DISK");
}
```

SEE: Throwing and Catching Java Exceptions

Threads

- Threads are independent execution entities which run concurrently but share the same code and variables.

- Definition:

```
public class ThreadName extends Thread {
   ThreadName( String[] arg ) { ... }  // constructor
   public void run() {
      ...
      while(true) {
         ...
      }
   }
}
```

- Invocation:

```
ThreadName t1 = new ThreadName("Thread 1");
ThreadName t2 = new ThreadName("Thread 2");
t1.start();  // without waiting for t1's term, start t2
t2.start();
```

Vector

- Vector is a list of Objects
- Declaration:

```
Vector v = new Vector();
```

- Any type of objects are inserted:

```
v.add(new Integer(10));
v.add(0, new Integer(5));
```

- When retrieved, values must be converted from Object to an appropriate type:

```
Integer i1 = (Integer)v.get(0);
Integer i2 = (Integer)v.lastElement();
```

Exercises:

- Programming Assignment 1:
  - ✓ Check the syllabus for its due date.
- (Canvas) Turn-in problems (due Sunday @ midnight):
  - ✓ Textbook Exercises: 2.2, 2.6, 2.15, 2.25
  - ✓ List five commands and systems calls with regard to process management, file management, and I/O management respectively. Explain each of their behaviors.

	Process management	File management	I/O management
Commands			
System calls			

IDE Demonstration…

IDE Build Sample

```
public class MyThread {
   public static void main( String args[] ) {
      String arg = args[0];
      ThreadFunc child = new ThreadFunc( arg );
      child.start( );
      for ( int i = 0; i < 10; i++ ) {
         try {
            Thread.sleep( 1000 );
         } catch ( InterruptedException e ) { };
         System.out.println( "Master["+ i +"]: " + arg );
      }
      try {
         child.join( );
      } catch ( InterruptedException e ) { };
      System.out.println( "Master synched with slave" );
   }
}
```

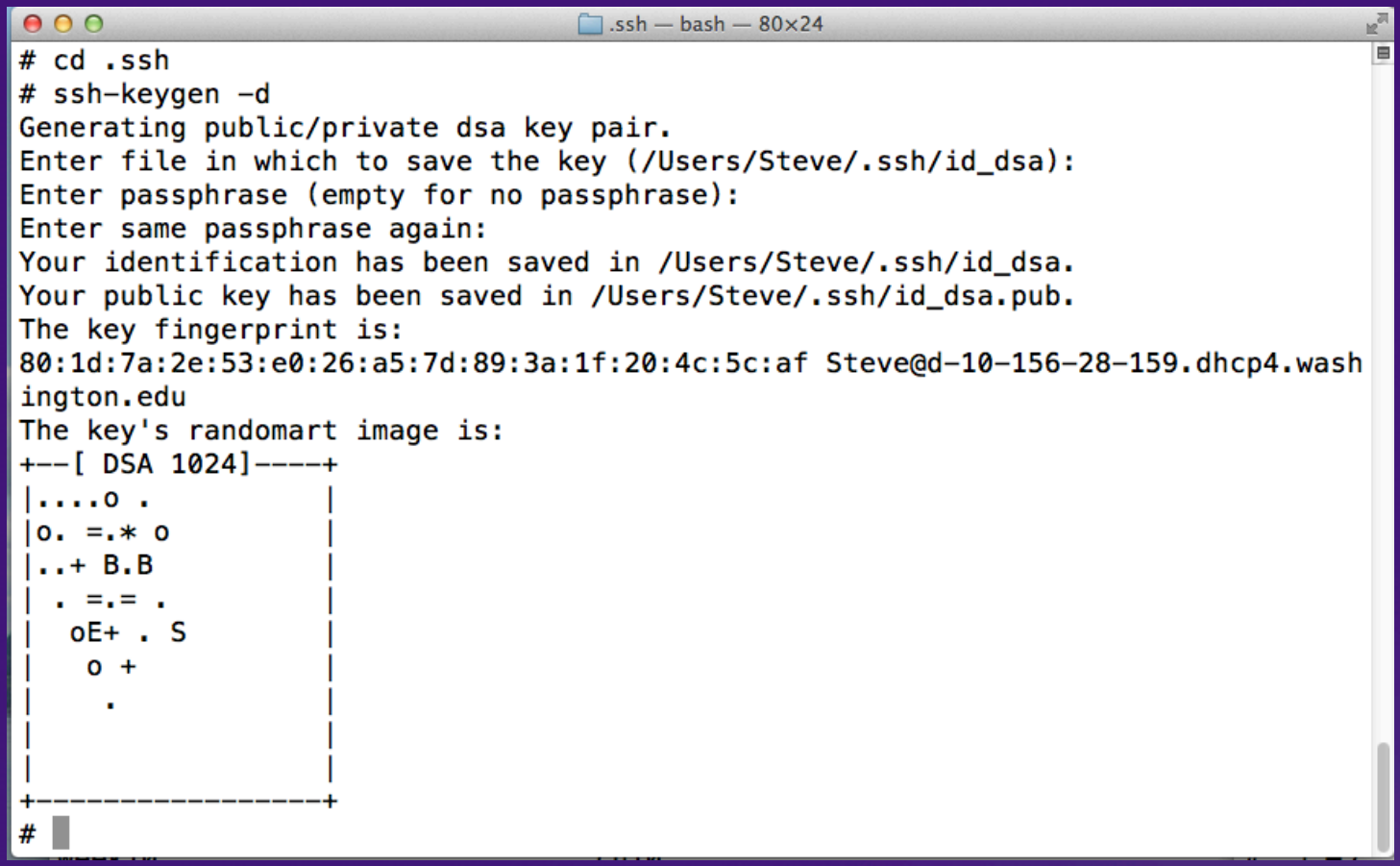
```
public class ThreadFunc extends Thread {
   private String param;
   public ThreadFunc( String strInit ) {
      param = strInit;
   }
   public void run( ) {
      for ( int i = 0; i < 3; i++ ) {
         try {
            Thread.sleep( 2000 );
         } catch ( InterruptedException e ) { };
         System.out.println( " Slave["+ i + "]: " + param );
      }
   }
}
```

SSH Keys

- Checklist/Instructions
  - Change directory to your home "ssh" folder
  - Generate public and private key
  - Rename your public key file to something memorable
  - Copy the public key to the host's ssh folder
  - Log in to the host and navigate to the ssh folder
  - APPEND! This public key to the end of the authorized_keys file
  - Log out and re-login to test

```
$ ssh uwnetid@uw1-320-lab.uwb.edu  OR
$ ssh –l uwnetid uw1-320-lab.uwb.edu
```

Generate with ssh-keygen




```
# ssh-keygen  → RSA key
# ssh-keygen -d → DSA key
What's the difference?  CLICK
```

Copy public key to host