Kernel Threads

David E. Culler

CS162 – Operating Systems and Systems Programming

Lecture 7

Sept 15, 2014

Reading: A&D Ch4.4-10
HW 1 due today
Proj. 1 Pintos Threads out

Objectives

- Solidify your understanding of threads as a concept.
- Use of threads
  – in user level programs
  – in the kernel
    - Support processes and OS concurrency
    - Support user level threads
- Develop your understanding of the implementation of threads in the kernel
  – You will develop it much further through project 1

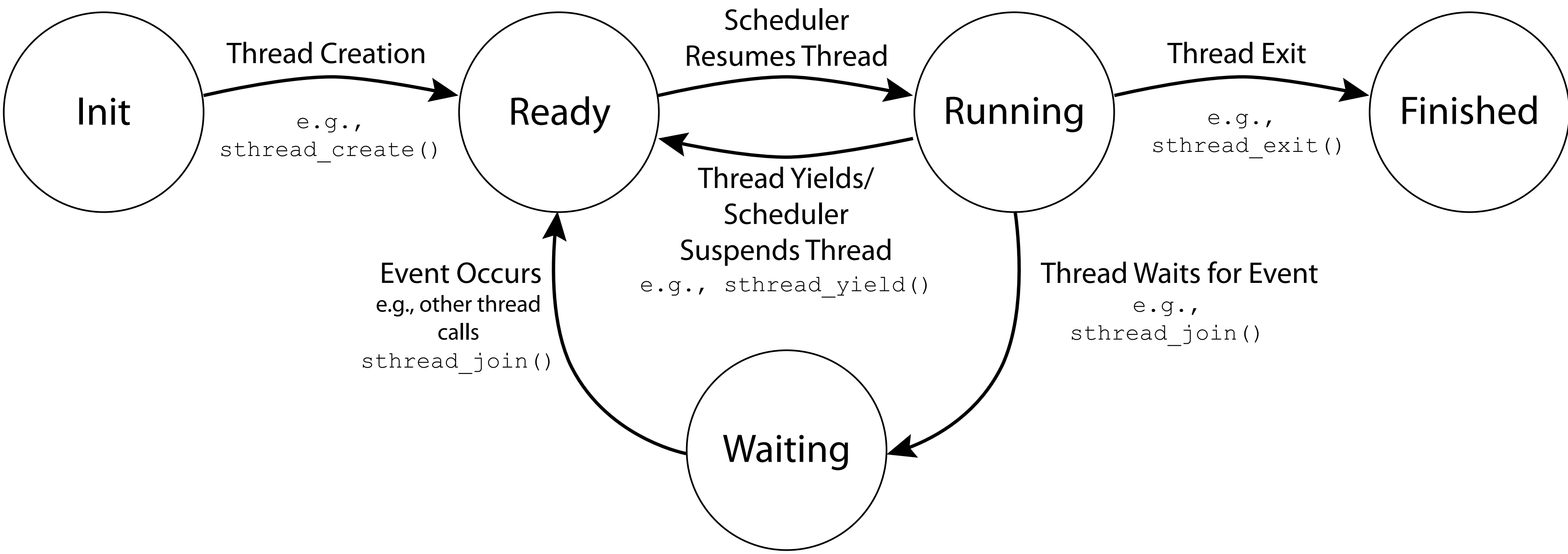
Threads

- Independently schedulable entity
- Sequential thread of execution that runs concurrently with other threads
  - It can block waiting for something while others progress
  - It can work in parallel with others (ala cs61c)
- Has local state (its stack) and shared (static data and heap)

Thread State

- State shared by all threads in process/addr space
  - Content of memory (global variables, heap)
  - I/O state (file system, network connections, etc)
- Execution Stack (logically private)
  - Parameters, temporary variables
  - Return PCs are kept while called procedures are executing
- State "private" to each thread
  - CPU registers (including, program counter)
  - Ptr to Execution stack
  - Kept in TCB ≡ Thread Control Block
    - When thread is not running
- Scheduler works on TCBs

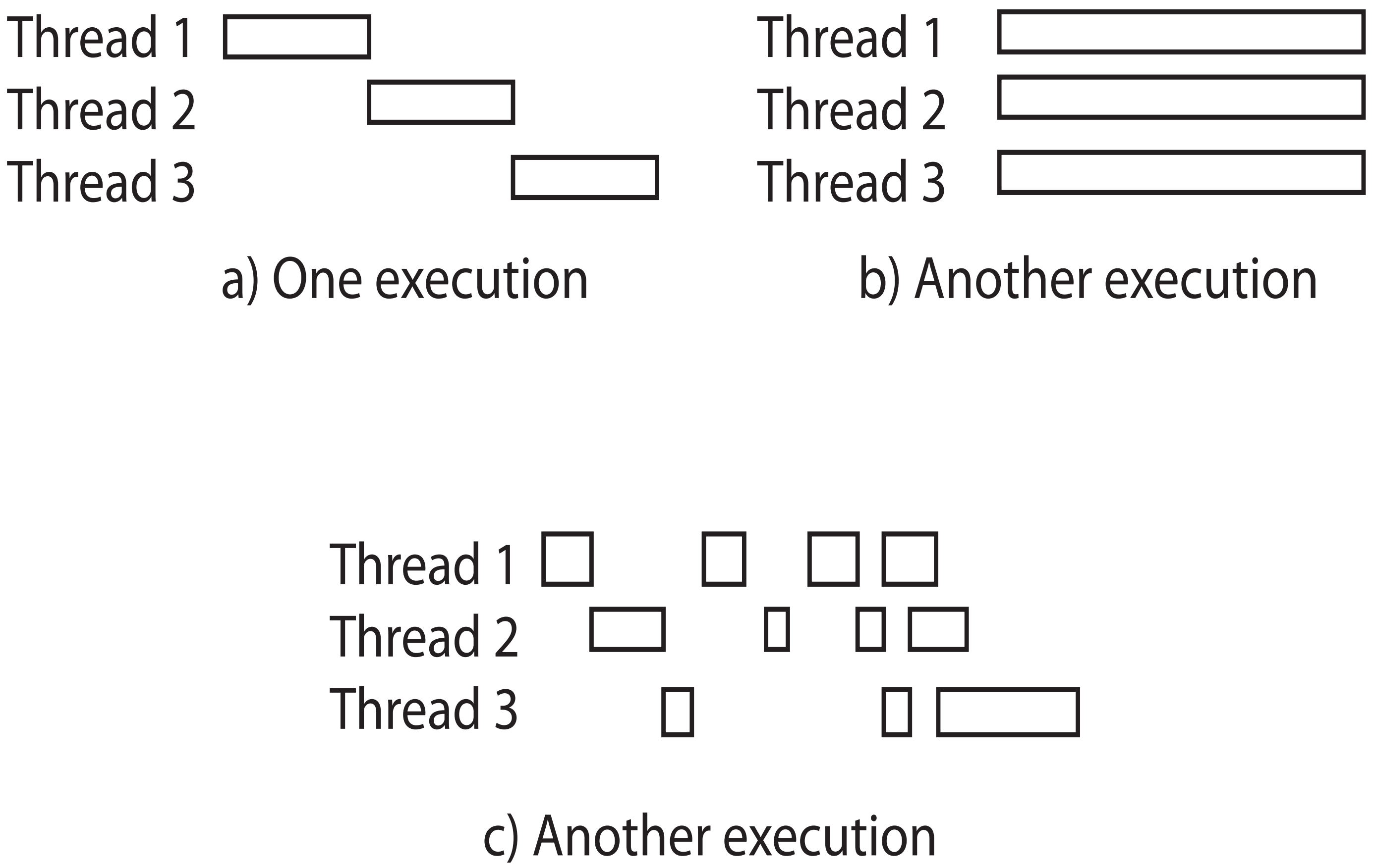
Thread Lifecycle



Programmer vs. Processor View

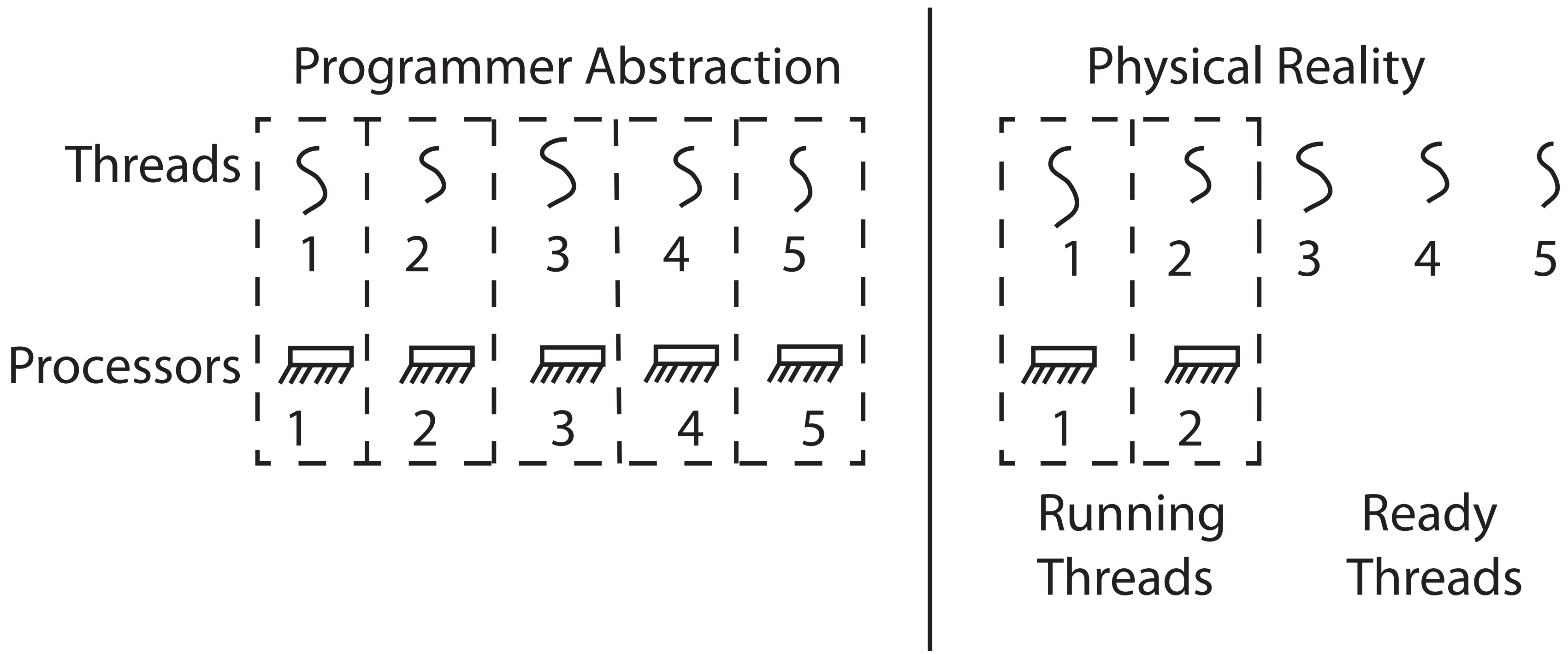
Programmer's View	Possible Execution #1	Possible Execution #2	Possible Execution #3
.	.	.	.
.	.	.	.
.	.	.	.
x = x + 1;	x = x + 1;	x = x + 1	x = x + 1
y = y + x;	y = y + x;	....	y = y + x
z = x +5y;	z = x + 5y;	thread is suspended	....
.	.	other thread(s) run	thread is suspended
.	.	thread is resumed	other thread(s) run
.	.	....	thread is resumed
		y = y + x	....
		z = x + 5y	z = x + 5y

Possible Executions



Thread Abstraction

- Infinite number of processors
- Threads execute with variable speed
  – Programs must be designed to work with any schedule



A typical use case

Client Browser
- process for each tab
- thread to render page
- GET in separate thread
- multiple outstanding GETs
- as they complete, render portion

Web Server
- fork process for each client connection
- thread to get request and issue response
- fork threads to read data, access DB, etc
- join and respond

Kernel Use Cases

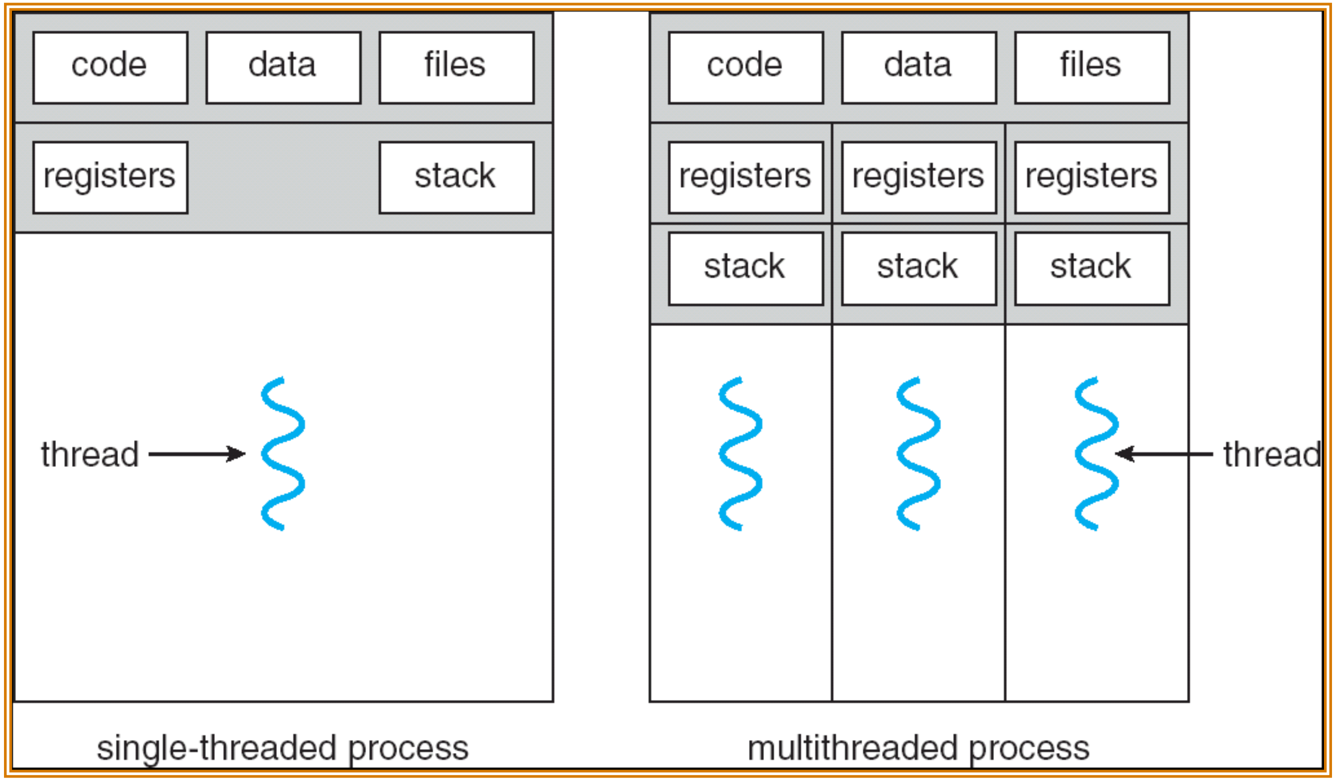
- Thread for each user process
- Thread for sequence of steps in processing I/O
- Threads for device drivers
- …

Per Thread State

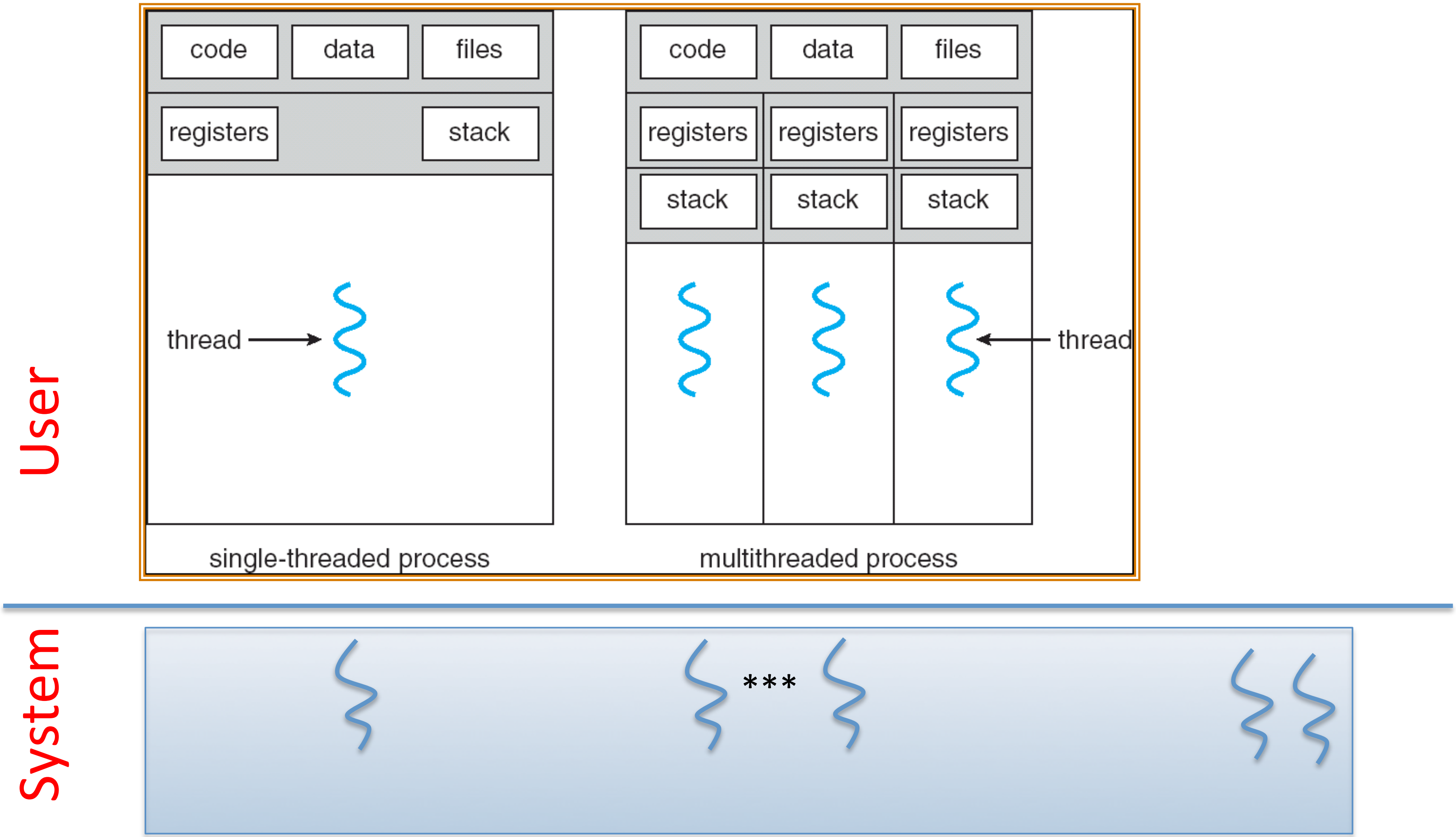
- Each Thread has a *Thread Control Block* (TCB)
  - Execution State: CPU registers, program counter (PC), pointer to stack (SP)
  - Scheduling info: state, priority, CPU time
  - Various Pointers (for implementing scheduling queues)
  - Pointer to enclosing process (PCB) – user threads
  - Etc (add stuff as you find a need)

- OS Keeps track of TCBs in "kernel memory"
  - In Array, or Linked List, or …

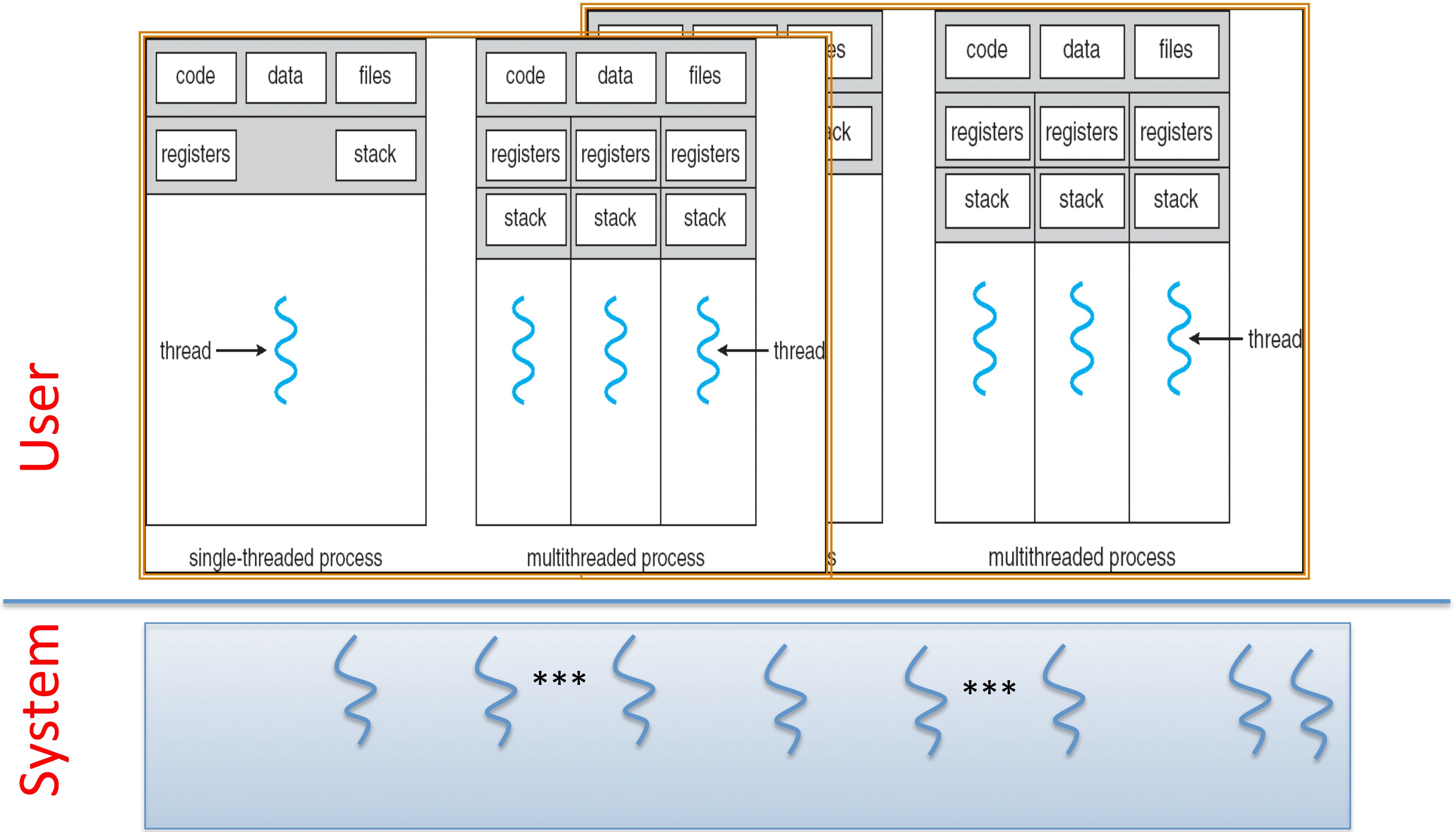
Single and Multithreaded Processes



Supporting 1T and MT Processes

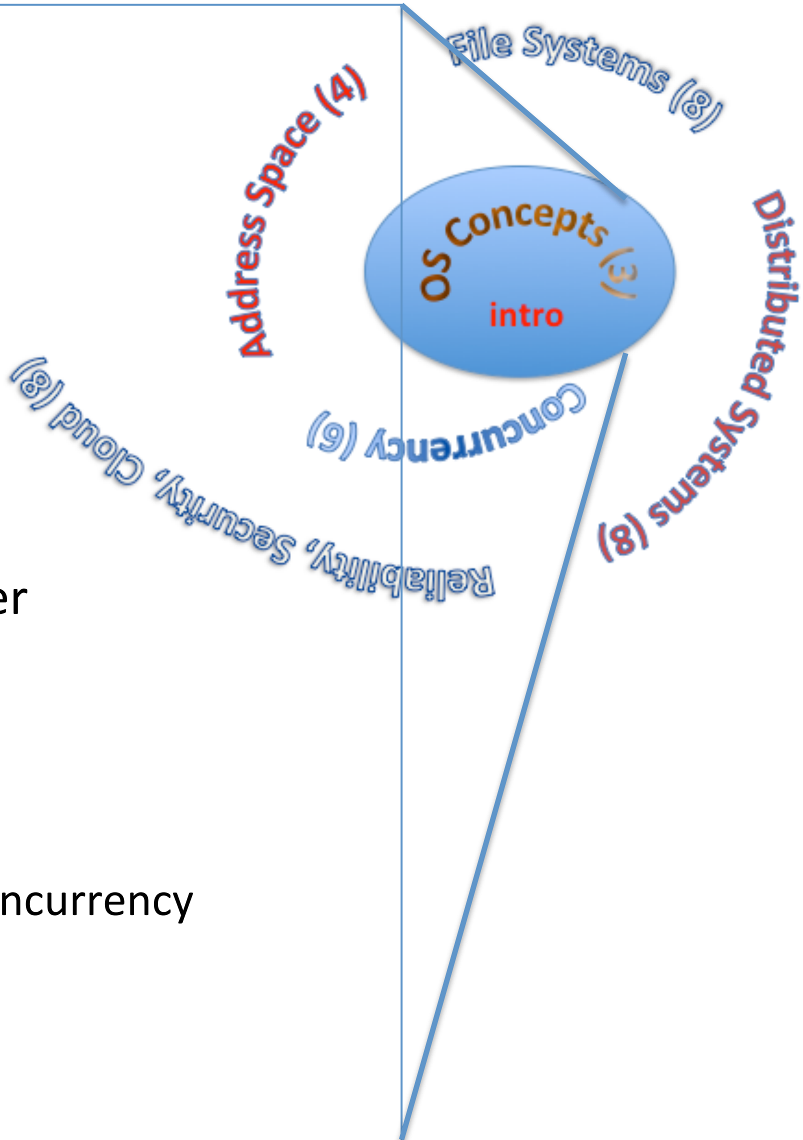


Supporting 1T and MT Processes



You are here… why?

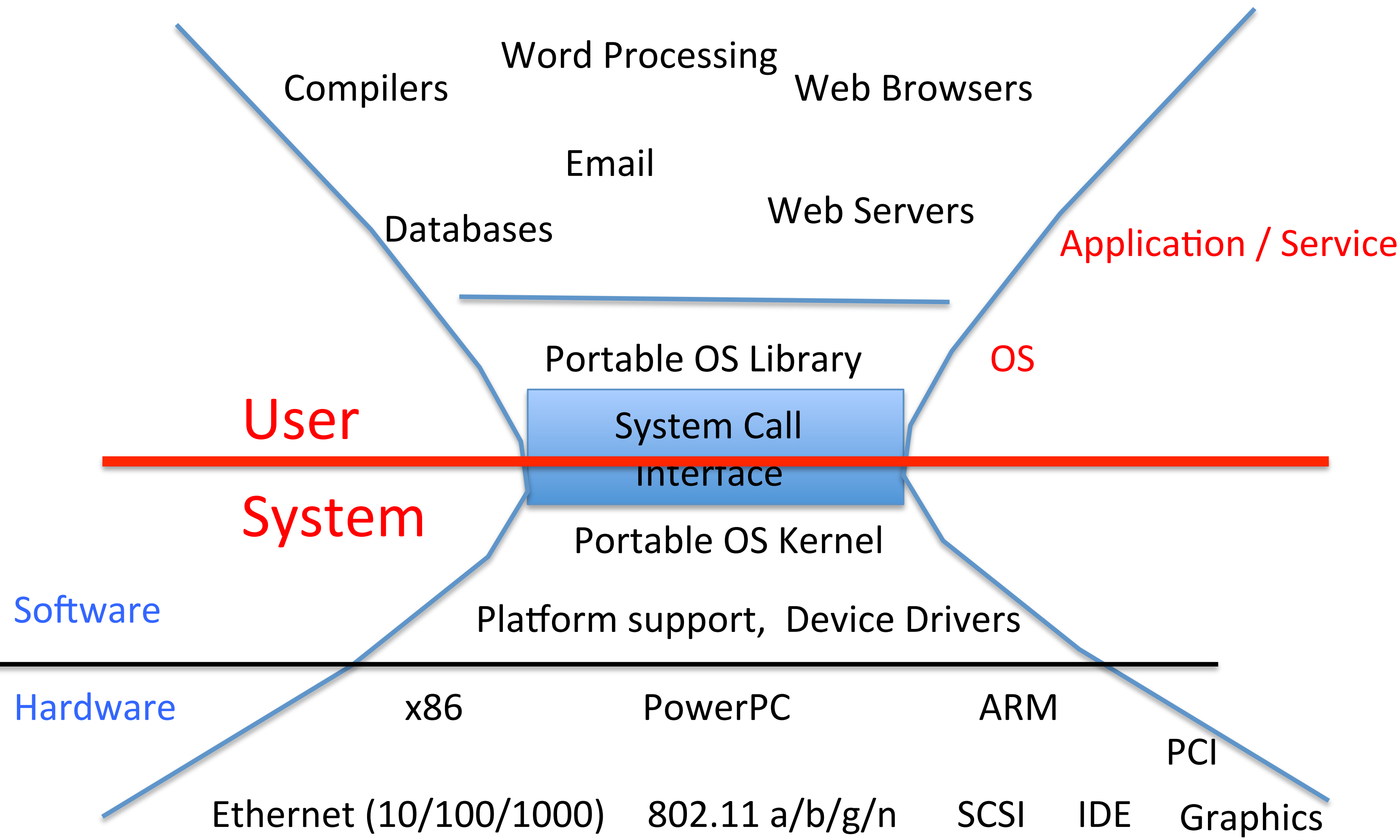
- Processes
  - Thread(s) + address space
- Address Space
- Protection
- Dual Mode
- Interrupt handlers
  - Interrupts, exceptions, syscall
- File System
  - Integrates processes, users, cwd, protection
- Key Layers: OS Lib, Syscall, Subsystem, Driver
  - User handler on OS descriptors
- Process control
  - fork, wait, signal, exec
- Communication through sockets
  - Integrates processes, protection, file ops, concurrency
- Client-Server Protocol
- Concurrent Execution: Threads
- Scheduling



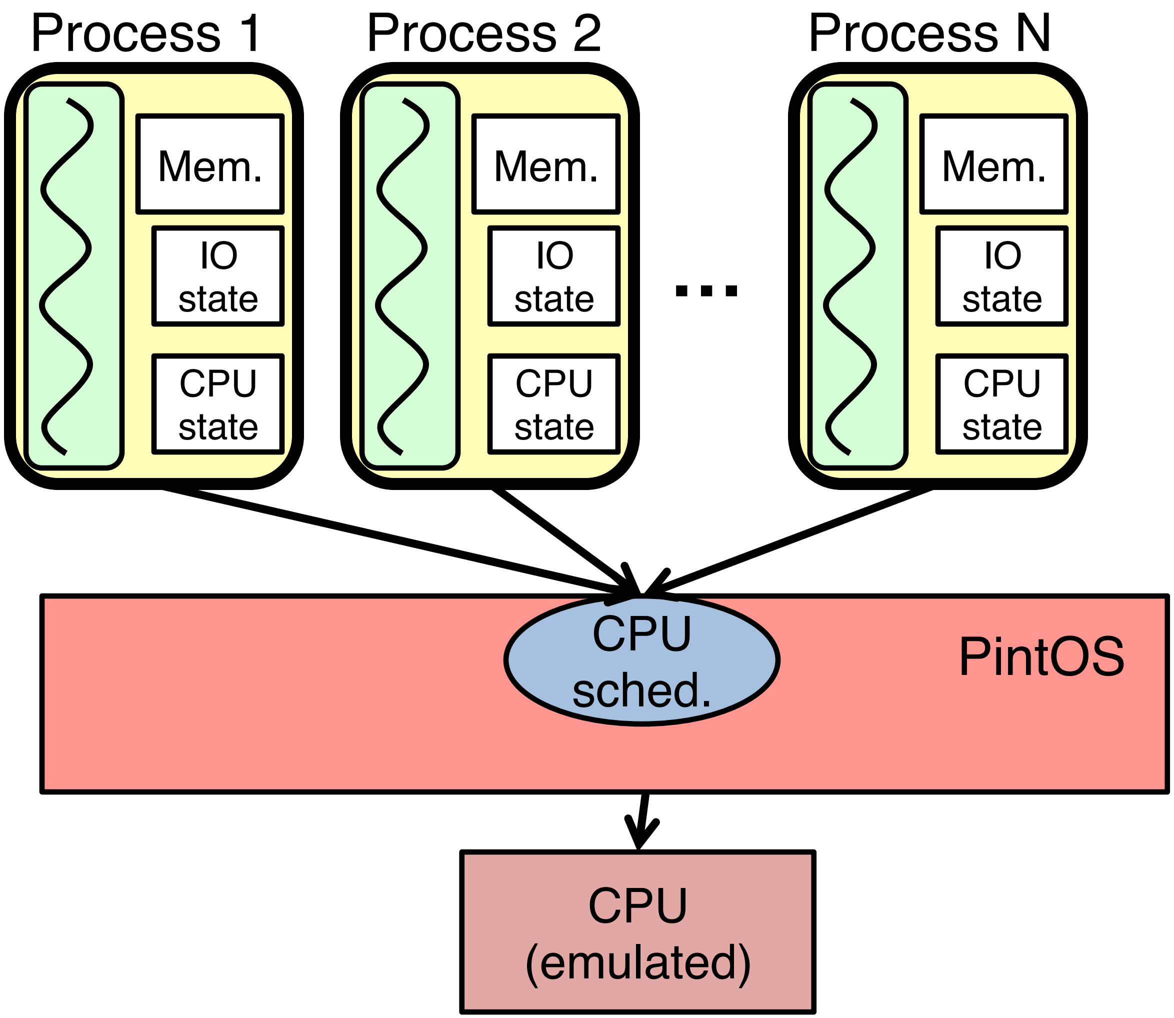
Perspective on 'groking' 162

- Historically, OS was the most complex software
  - Concurrency, synchronization, processes, devices, communication, …
  - Core systems concepts developed there
- Today, many "applications" are complex software systems too
  - These concepts appear there
  - But they are realized out of the capabilities provided by the operating system
- Seek to understand how these capabilities are implemented upon the basic hardware.
- See concepts multiple times from multiple perspectives
  - Lecture provides conceptual framework, integration, examples, …
  - Book provides a reference with some additional detail
  - Lots of other resources that you need to learn to use
    - man pages, google, reference manuals, includes (.h)
- Section, Homework and Project provides detail down to the actual code AND direct hands-on experience

Operating System as Design

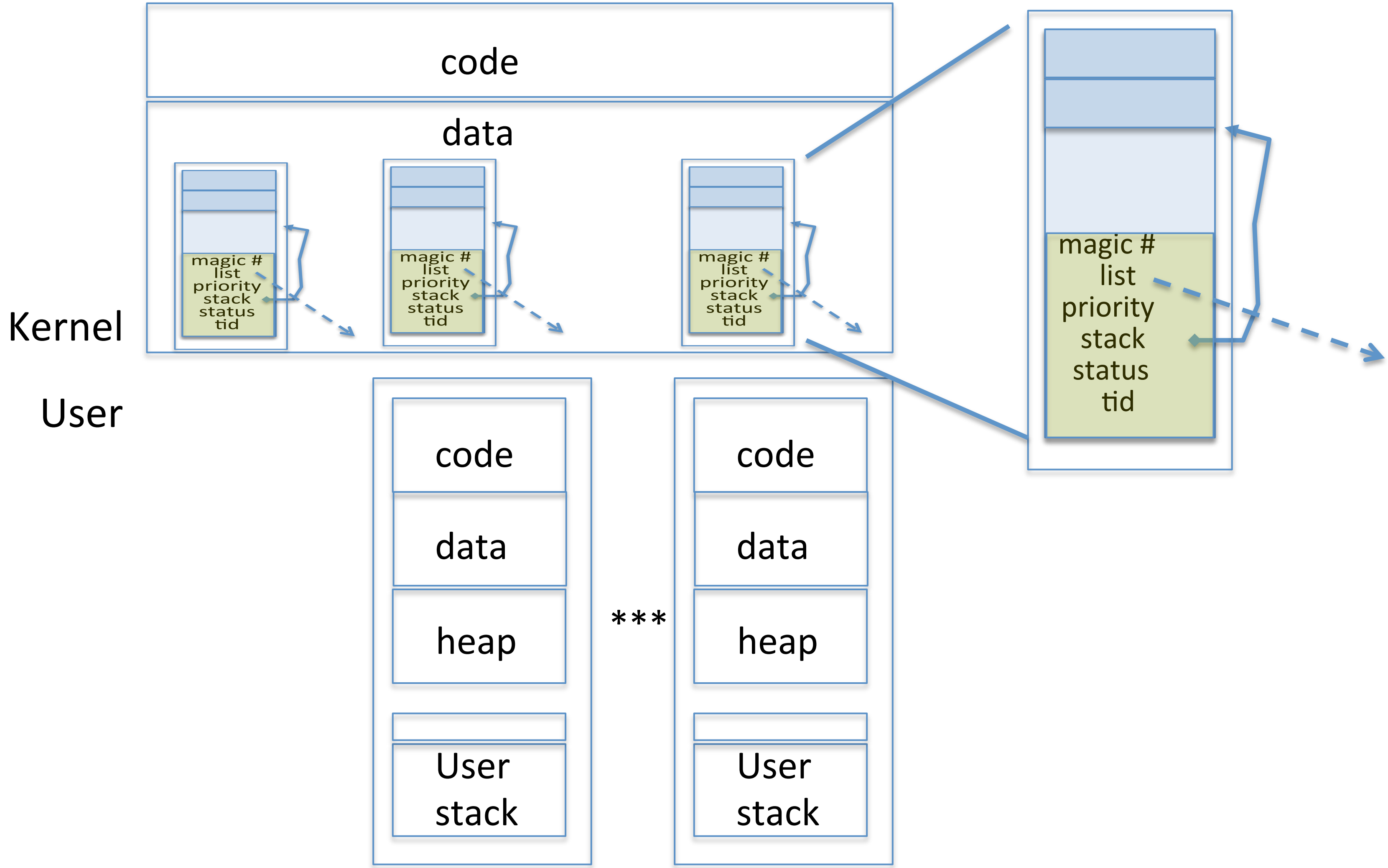


Starting today: Pintos Projects



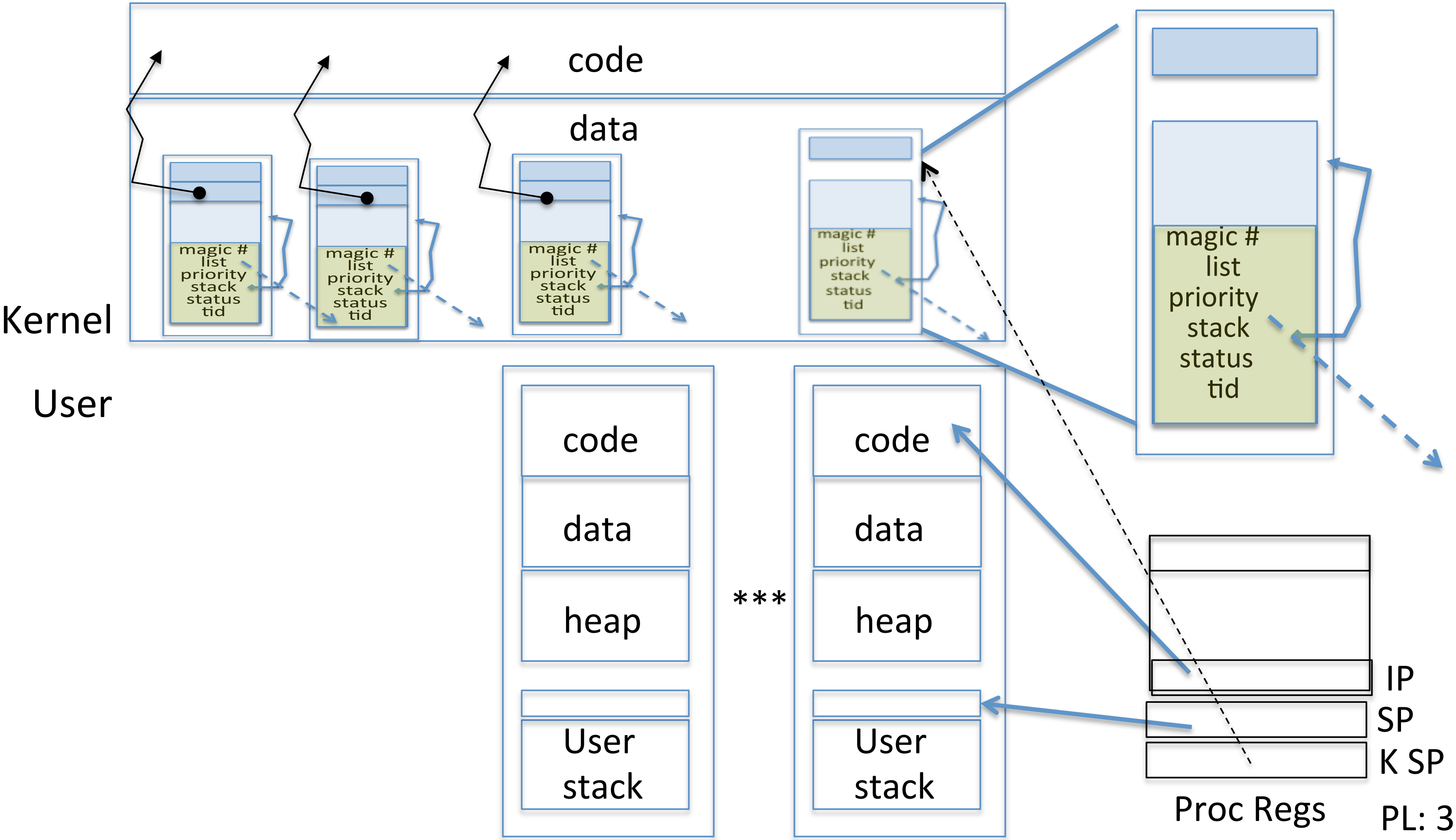
- Groups almost all formed
- Work as one!
- 10x homework
- P1: threads & scheduler
- P2: user process

MT Kernel 1T Process ala Pintos/x86



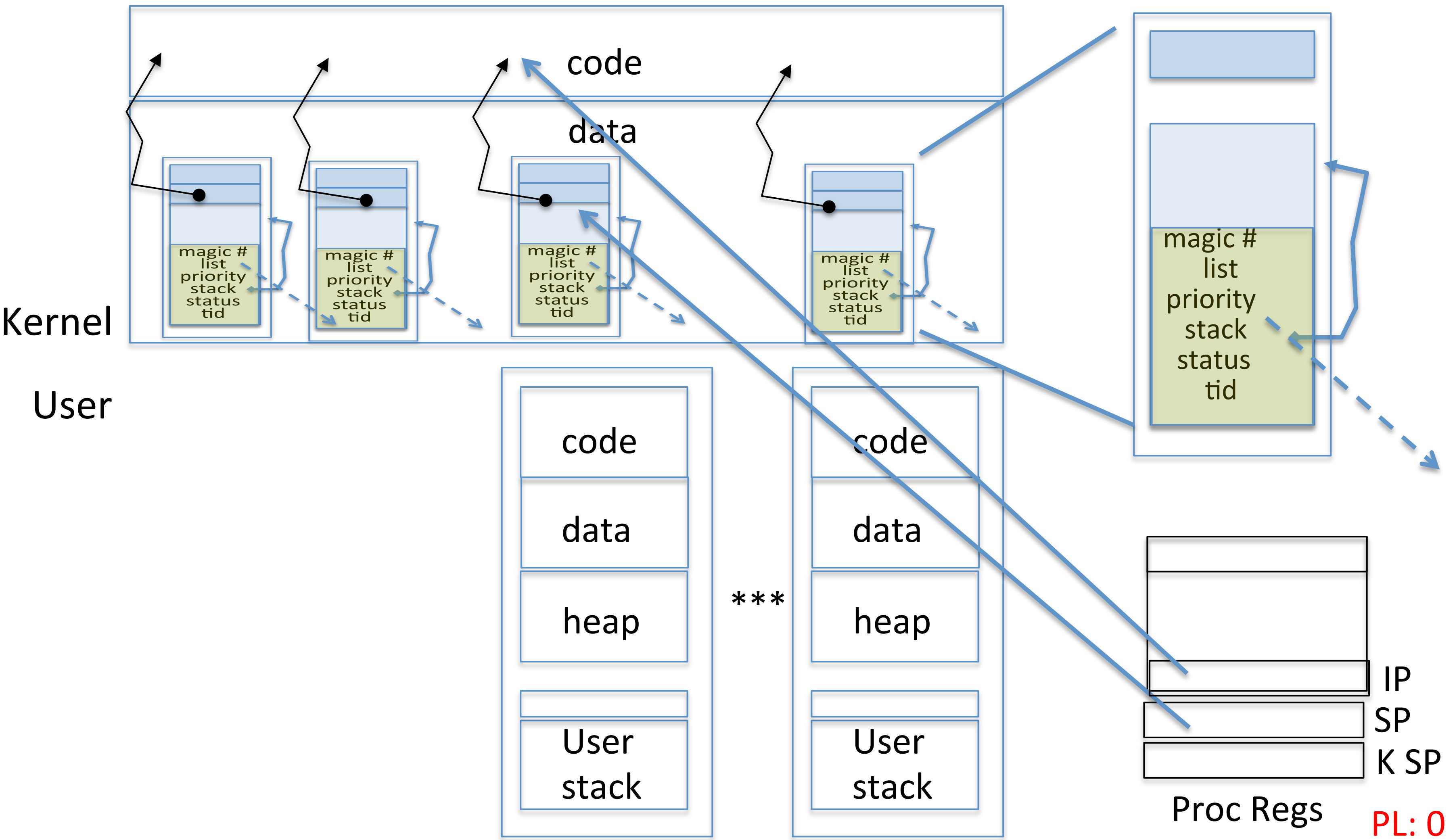
- Each user process/thread associated with a kernel thread, described by a 4kb Page object containing TCB and kernel stack for the kernel thread

In User thread, w/ k-thread waiting



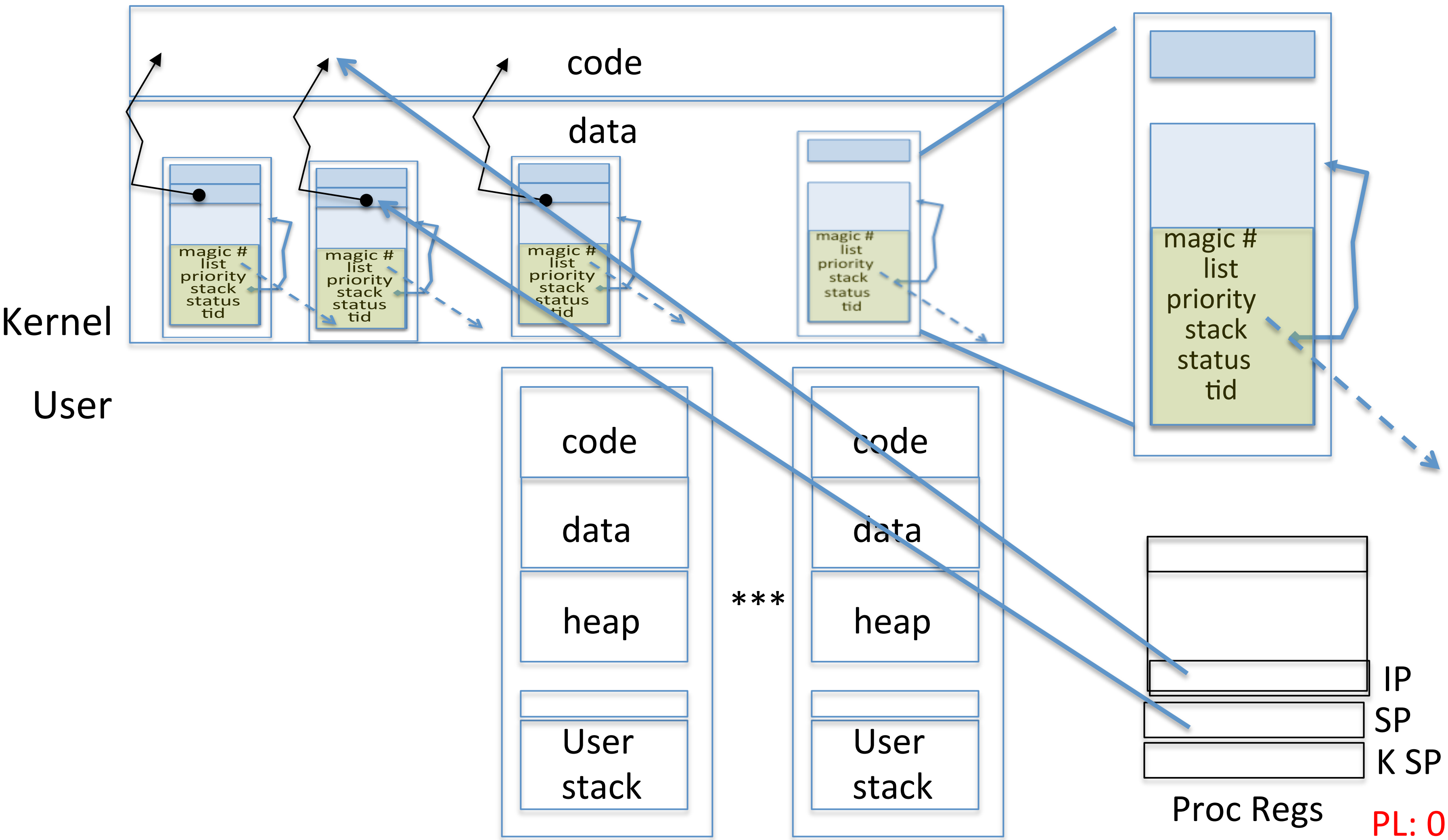
- x86 proc holds interrupt SP high system level
- During user thread exec, associate kernel thread is “standing by”

In Kernel thread



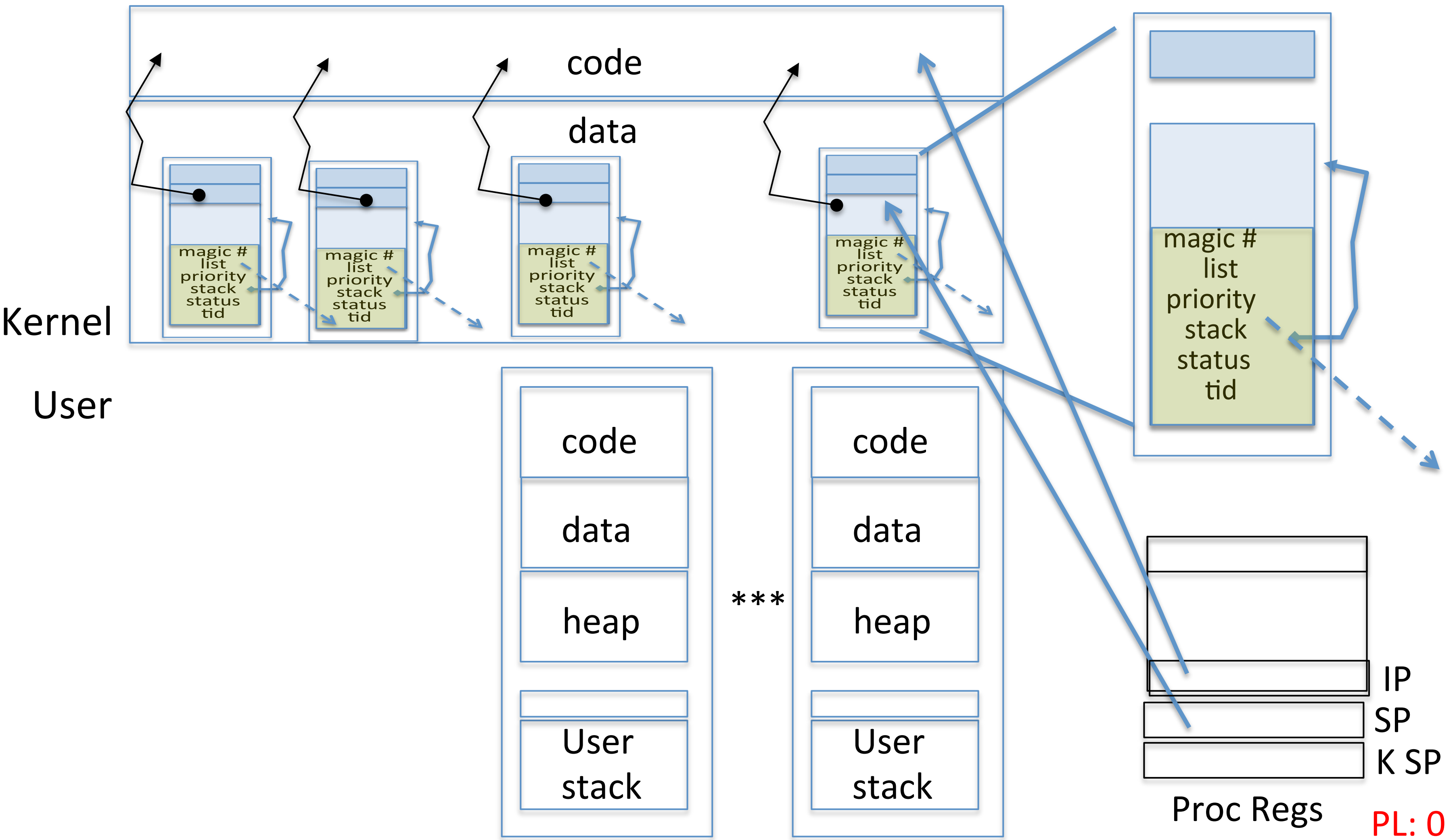
- Kernel threads execute with small stack in thread struct
- Scheduler selects among ready kernel and user threads

Thread Switch (switch.S)

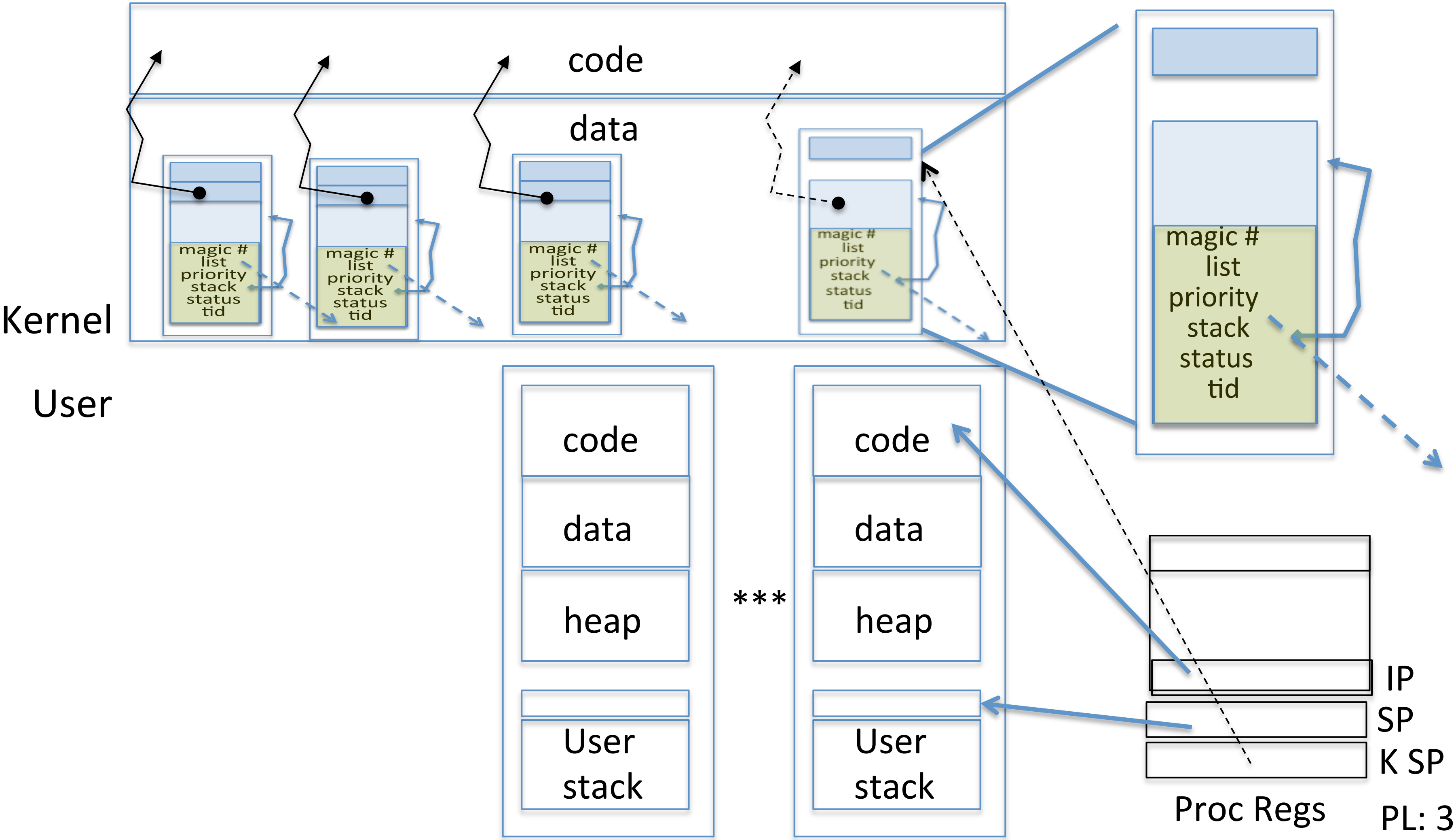


- switch_threads: save regs on current small stack, change SP, return from destination threads call to switch_threads

Switch to Kernel Thread for Process

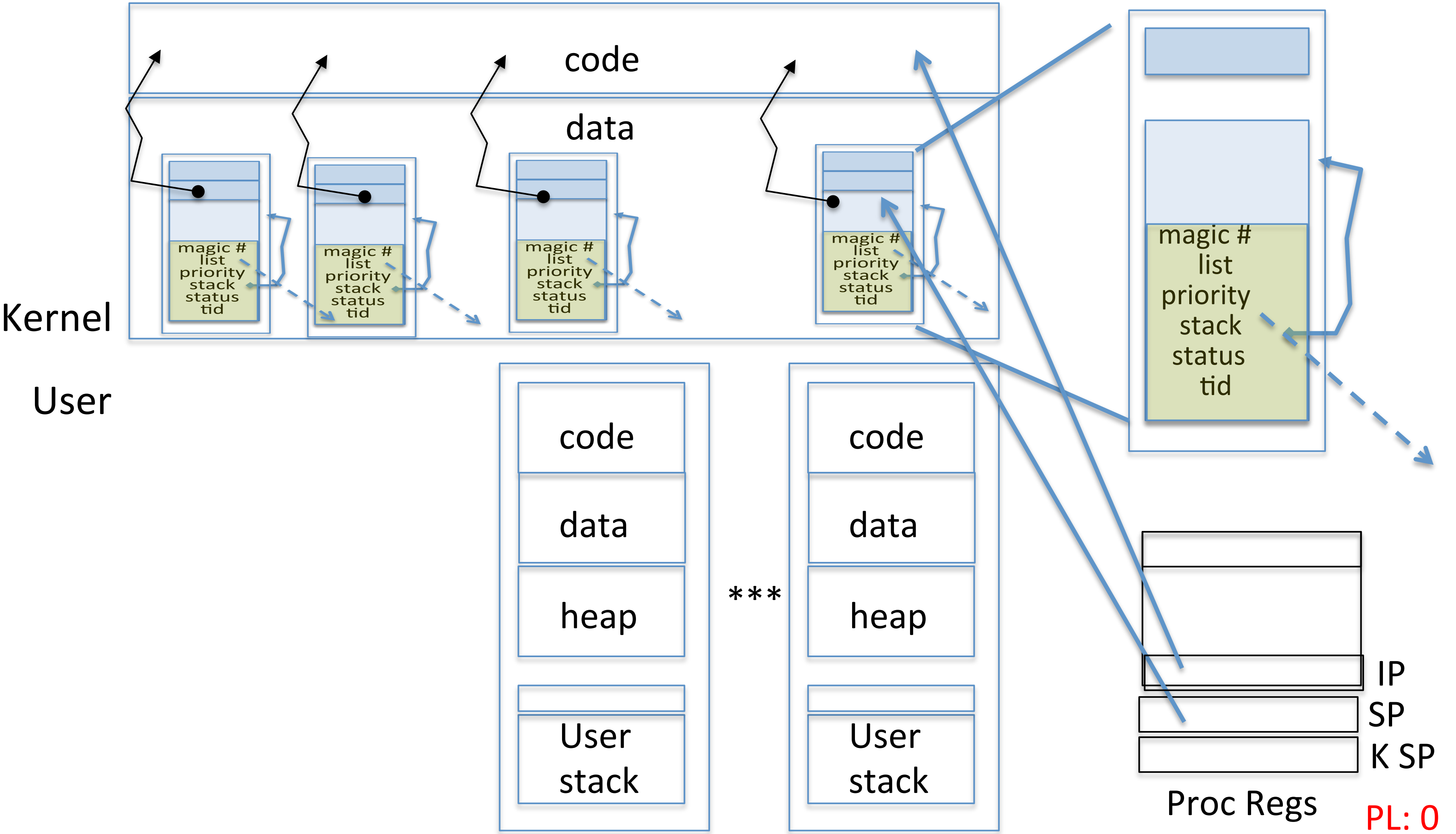


Kernel->User



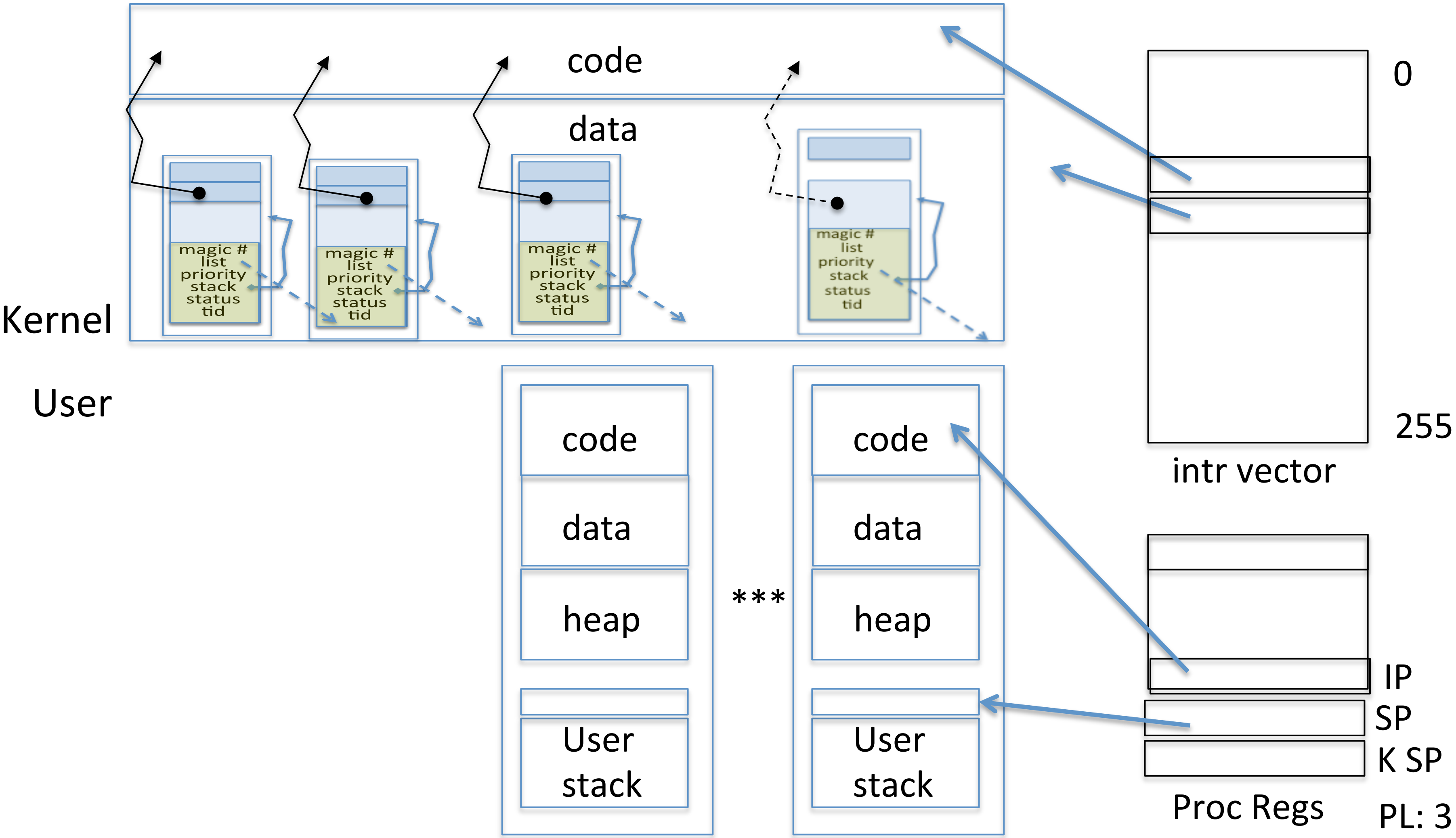
- iret restores user stack and PL

User->Kernel



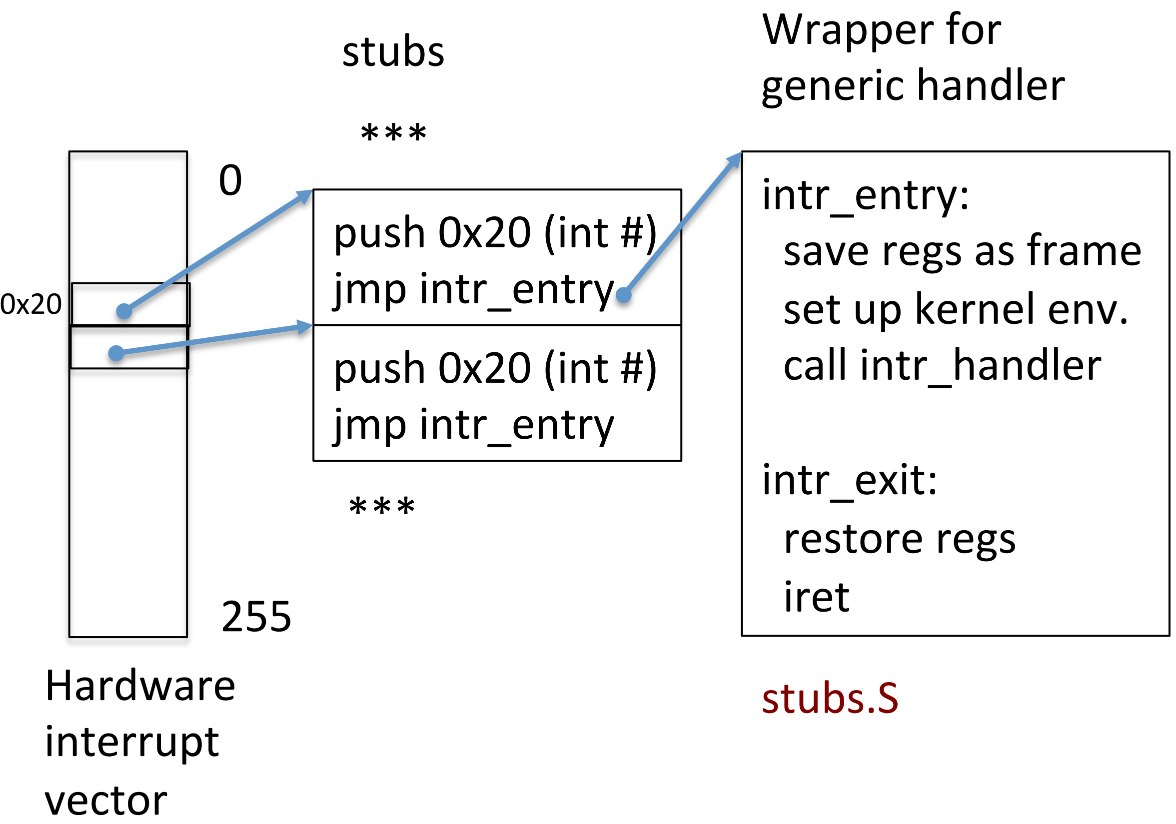
- Mechanism to resume k-thread goes through interrupt vector

User->Kernel via interrupt vector



- Interrupt transfers control through the IV (IDT in x86)
- iret restores user stack and PL

Pintos Interrupt Processing



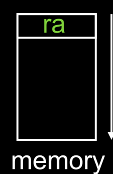
Recall: cs61C THE STACK FRAME

Basic Structure of a Function

Prologue

```
entry_label:
addi $sp,$sp, -framesize
sw $ra, framesize-4($sp)   # save $ra
save other regs if need be
```

Body . . .   **(call other functions…)**



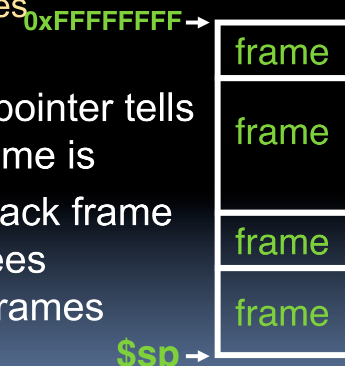
Epilogue

```
restore other regs if need be
lw $ra, framesize-4($sp)   # restore $ra
addi $sp,$sp, framesize
jr $ra
```

CS61C L10 Introduction to MIPS : Procedures I (18)   Garcia, Spring 2014

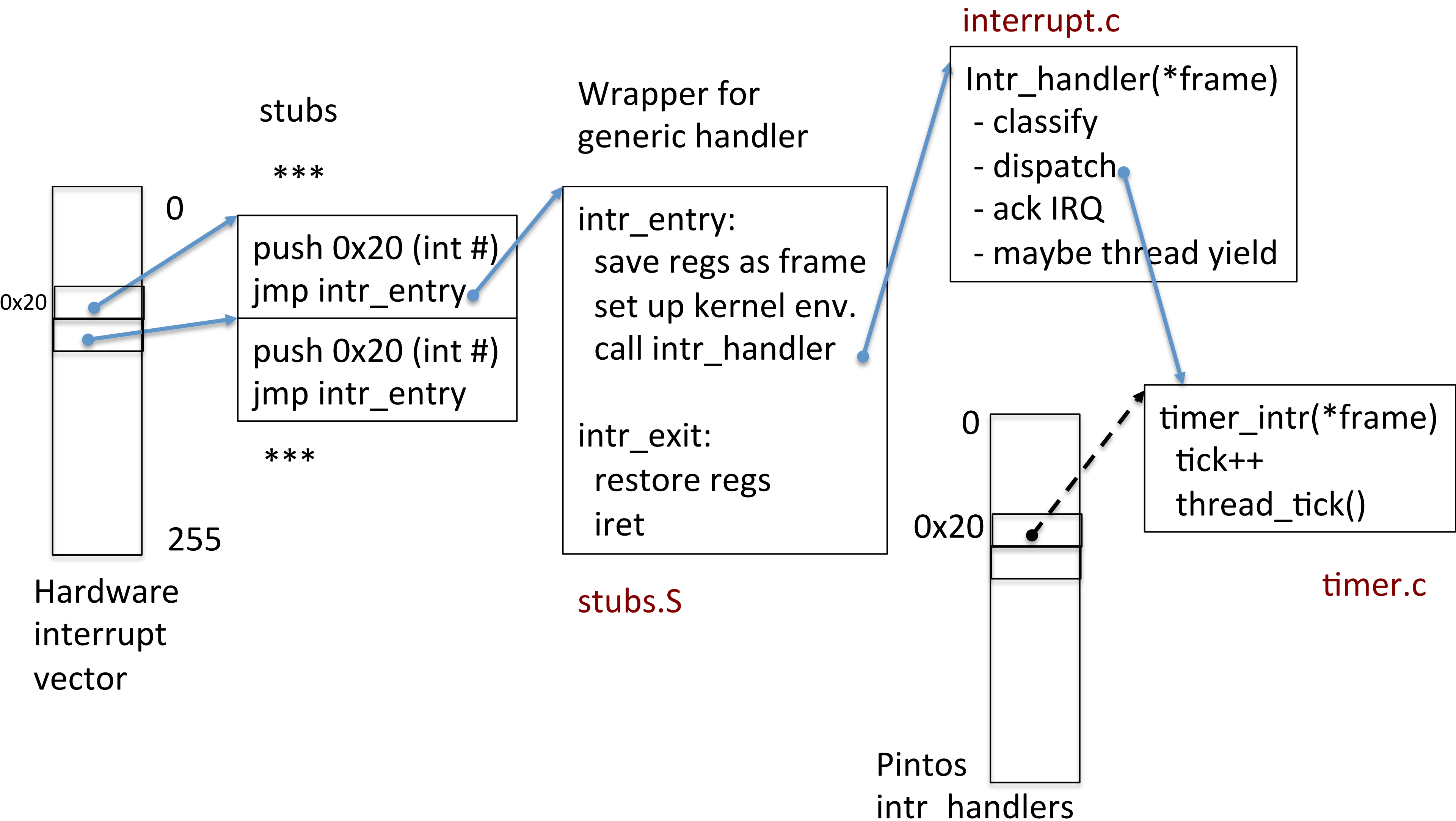
The Stack (review)

- Stack frame includes:
  - Return "instruction" address
  - Parameters
  - Space for other local variables
- Stack frames contiguous blocks of memory; stack pointer tells where bottom of stack frame is
- When procedure ends, stack frame is tossed off the stack; frees memory for future stack frames



CS61C L11 Introduction to MIPS : Procedures II & Logical Ops (3)   Garcia, Spring 2014 © UCB

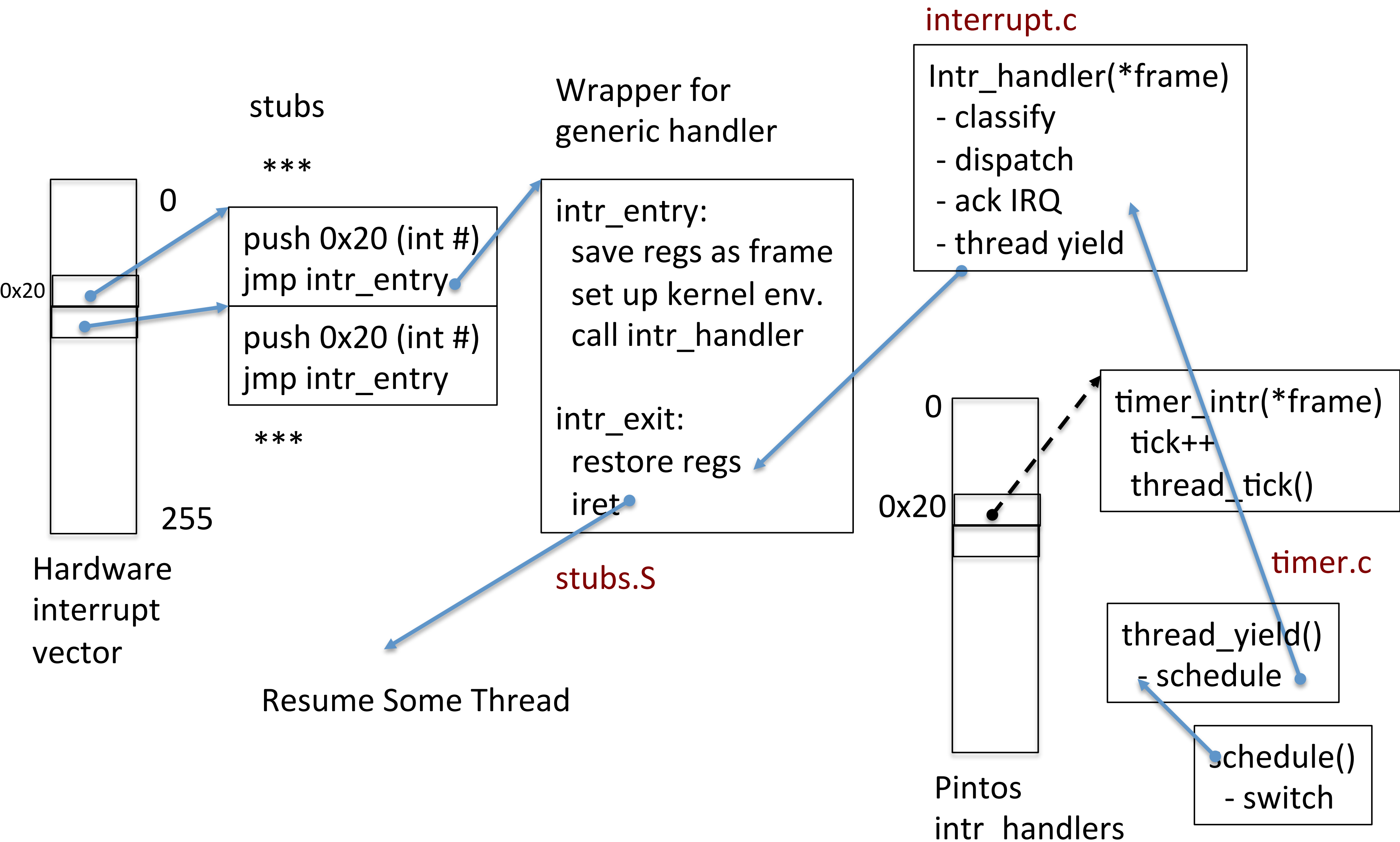
Pintos Interrupt Processing



Timer may trigger thread switch

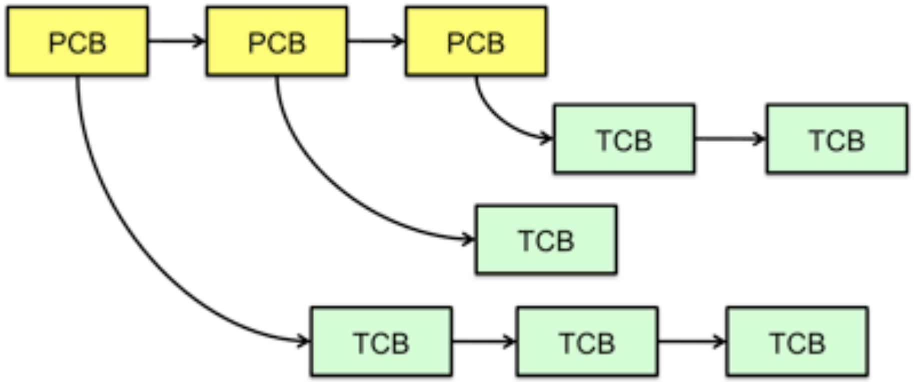
- thread_tick
  - Updates thread counters
  - If quanta exhausted, sets yield flag
- thread_yield
  - On path to rtn from interrupt
  - Sets current thread back to READY
  - Pushes it back on ready_list
  - Calls schedule to select next thread to run upon iret
- Schedule
  - Selects next thread to run
  - Calls switch_threads to change regs to point to stack for thread to resume
  - Sets its status to RUNNING
  - If user thread, activates the process
  - Returns back to intr_handler

Pintos Return from Processing



Multithreaded Processes

- PCB may be associated with multiple TCBs:



- Switching threads within a process is a simple thread switch
- Switching threads across blocks requires changes to memory and I/O address tables.

The Next Big Question

- So how do threads cooperate & coordinate?

- Synchronization operations
  - High level structured to low level unstructured
  - Disabling interrupts is the lowest and most brute force
    - Eliminates interleaving in short sections of OS code

Perspectives

The Numbers

Context switch in Linux: 3-4 μsecs (Current Intel i7 & E5).

- Thread switching faster than process switching (100 ns).
- But switching across cores about 2x more expensive than within-core switching.
- Context switch time increases sharply with the size of the working set*, and can increase 100x or more.

* The working set is the subset of memory used by the process in a time window.

Moral: Context switching depends mostly on cache limits and the process or thread's hunger for memory.

The Numbers

- Many process are multi-threaded, so thread context switches may be either **within-process** or **across-processes**.

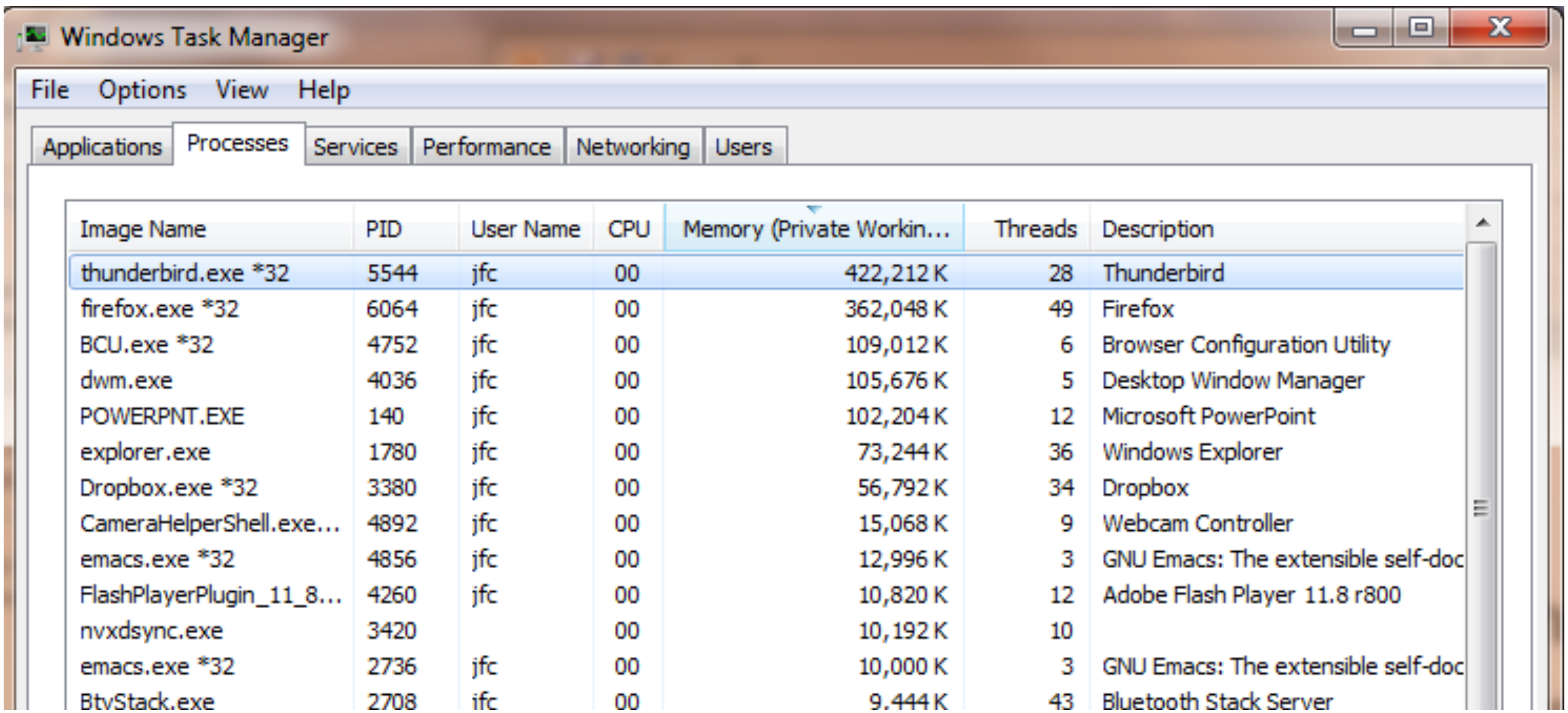


Image Name	PID	User Name	CPU	Memory (Private Workin...	Threads	Description
thunderbird.exe *32	5544	jfc	00	422,212 K	28	Thunderbird
firefox.exe *32	6064	jfc	00	362,048 K	49	Firefox
BCU.exe *32	4752	jfc	00	109,012 K	6	Browser Configuration Utility
dwm.exe	4036	jfc	00	105,676 K	5	Desktop Window Manager
POWERPNT.EXE	140	jfc	00	102,204 K	12	Microsoft PowerPoint
explorer.exe	1780	jfc	00	73,244 K	36	Windows Explorer
Dropbox.exe *32	3380	jfc	00	56,792 K	34	Dropbox
CameraHelperShell.exe...	4892	jfc	00	15,068 K	9	Webcam Controller
emacs.exe *32	4856	jfc	00	12,996 K	3	GNU Emacs: The extensible self-doc
FlashPlayerPlugin_11_8...	4260	jfc	00	10,820 K	12	Adobe Flash Player 11.8 r800
nvxdsync.exe	3420		00	10,192 K	10	
emacs.exe *32	2736	jfc	00	10,000 K	3	GNU Emacs: The extensible self-doc
BtvStack.exe	2708	jfc	00	9,444 K	43	Bluetooth Stack Server

Threads in a Process

- Threads are useful at user-level
  - Parallelism, hide I/O latency, interactivity
- Option A (early Java): user-level library, within a single-threaded process
  - Library does thread context switch
  - Kernel time slices between processes, e.g., on system call I/O
- Option B (Linux, MacOS, Windows): use kernel threads
  - System calls for thread fork, join, exit (and lock, unlock,…)
  - Kernel does context switching
  - Simple, but a lot of transitions between user and kernel mode
- Option C (Windows): scheduler activations
  - Kernel allocates processors to user-level library
  - Thread library implements context switch
  - System call I/O that blocks triggers upcall
- Option D: Asynchronous I/O

Classification

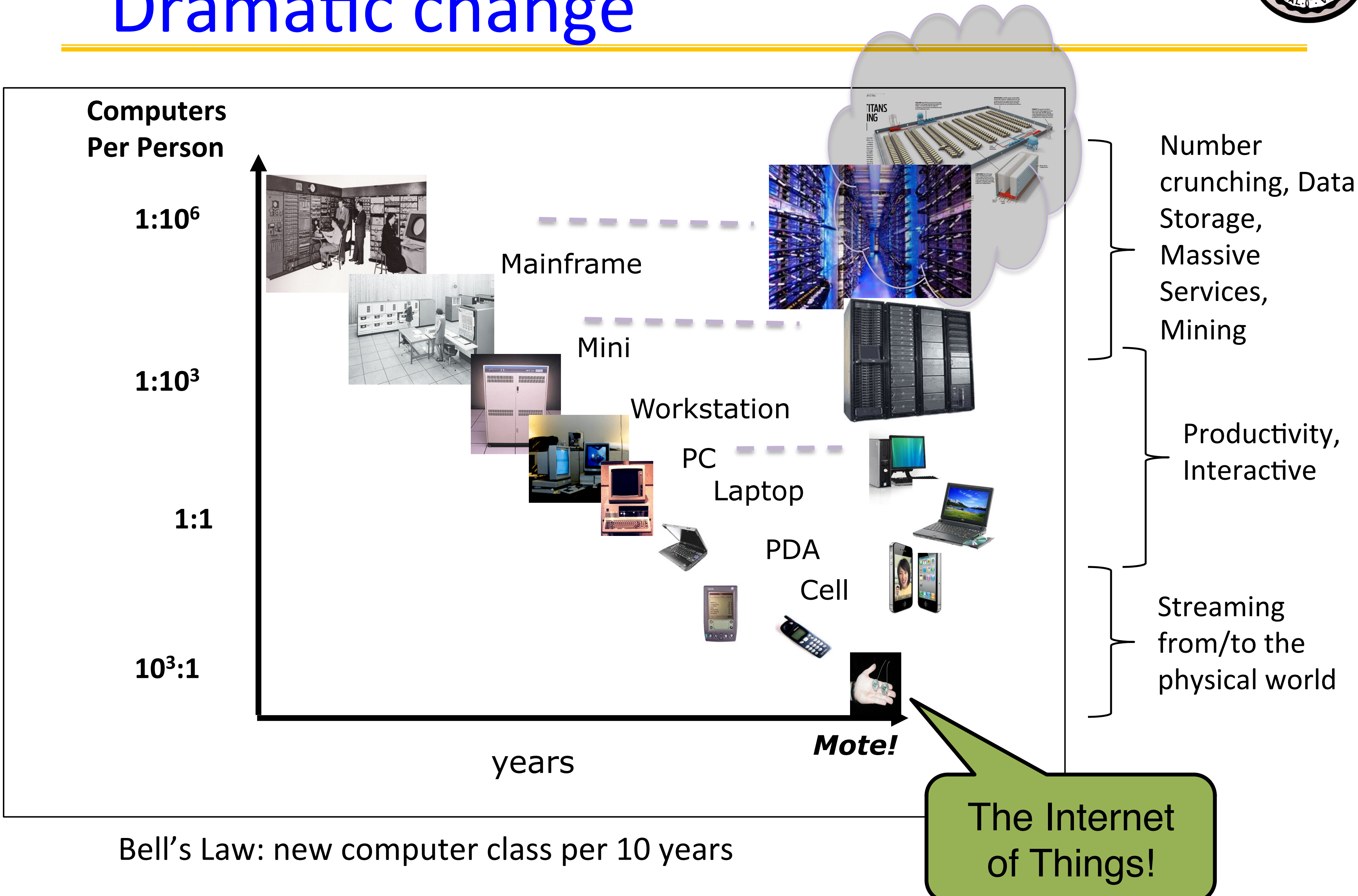
# threads Per AS: / # of addr spaces:	One	Many
One	MS/DOS, early Macintosh	Traditional UNIX
Many	Embedded systems (Geoworks, VxWorks, JavaOS,etc) JavaOS, Pilot(PC)	Mach, OS/2, HP-UX, Win NT to 8, Solaris, OS X, Android, iOS

- Real operating systems have either
  - One or many address spaces
  - One or many threads per address space

OS Archaeology

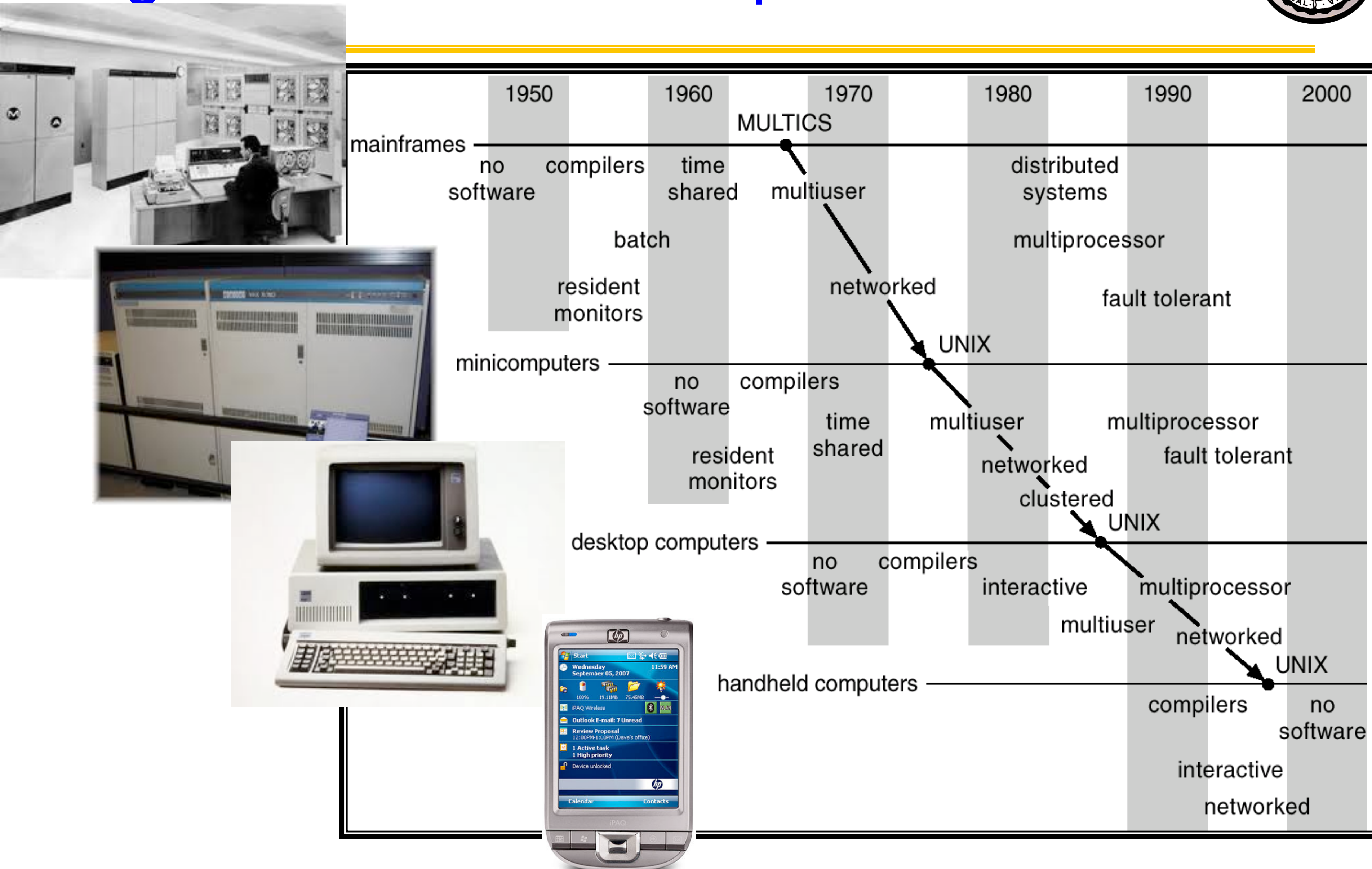
- Because of the cost of developing an OS from scratch, most modern OSes have a long lineage:
- Multics → AT&T Unix → BSD Unix → Ultrix, SunOS, NetBSD,…
- Mach (micro-kernel) + BSD → NextStep → XNU → Apple OSX, iphone iOS
- Linux → Android OS
- CP/M → QDOS → MS-DOS → Windows 3.1 → NT → 95 → 98 → 2000 → XP → Vista → 7 → 8 → phone → …
- Linux → RedHat, Ubuntu, Fedora, Debian, Suse,…

Dramatic change



Bell's Law: new computer class per 10 years

Migration of OS Concepts and Features



Recall: (user) Thread Operations

- thread_fork(func, args)
  - Create a new thread to run func(args)
  - Pintos: thread_create
- thread_yield()
  - Relinquish processor voluntarily
  - Pintos: thread_yield
- thread_join(thread)
  - In parent, wait for forked thread to exit, then return
- thread_exit
  - Quit thread and clean up, wake up joiner if any
  - Pintos: thread_exit

http://cs162.eecs.berkeley.edu/static/lectures/code06/pthread.c

Example: pthreads.c