CS 630 - 002
Operating Systems Design

Lecture 2
OS Designs

Jing Li

jingli@njit.edu

GITC 4106

New Jersey Institute of Technology

In This Lecture

- Different Operating Systems Designs

- Administrivia
  - Assignment 1 will be released on Feb 5 and due on Feb 19 23:55

- OS Examples:
  - Unix, Linux, Window, Mac

- Background
  - How to gradually add abstraction to multiplex resources

Slides courtesy of Hung Daochuan, Chris Gill, David Ferry, Tarek Abdelzaher, Ion Stoica, John Kubiatowicz, Peter Dennings, Anthony Joseph, Jonathan Ragan-Kelley, Peter Troger.

Recap

Abstraction, Arbitration & Protection

- OS offers abstractions, arbitration and protection

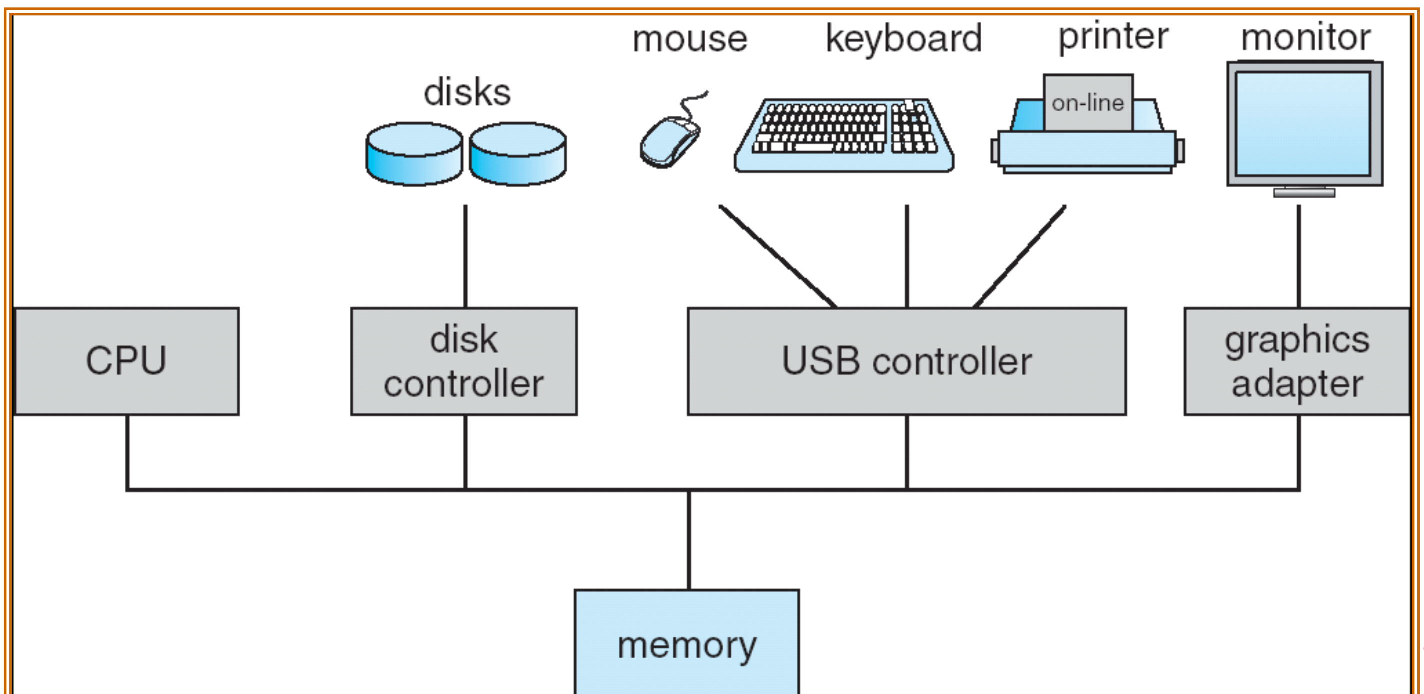
Application

Virtual Machine Interface

Operating System

Physical Machine Interface

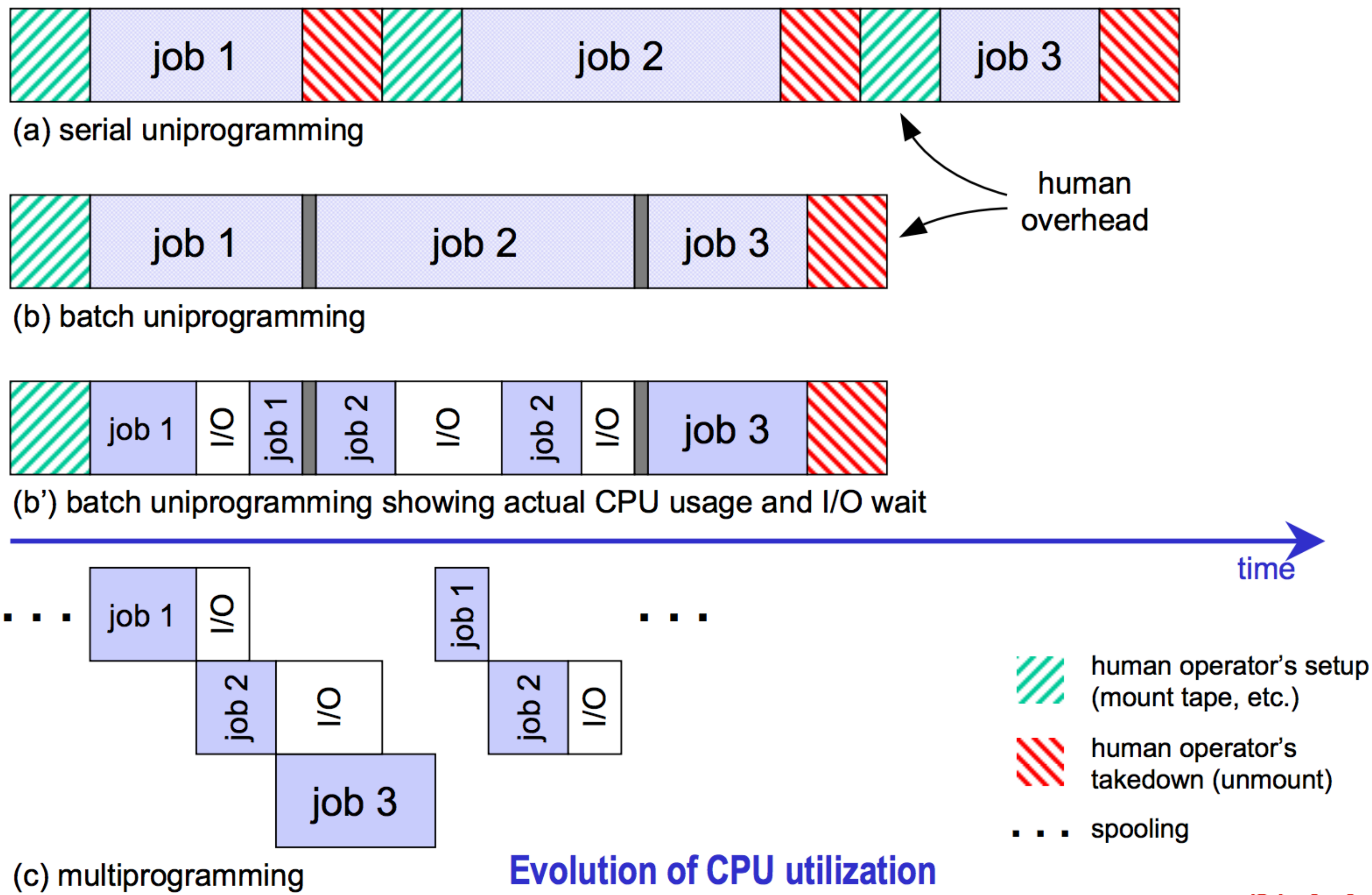
Hardware



Common Operating System Components

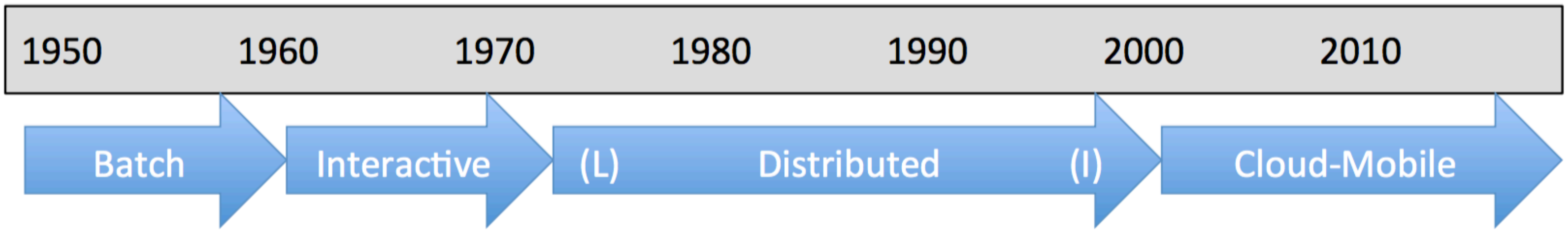
- Process Management:
  - "process" is a program in execution. OS responsible for creation, deletion, scheduling, communication
- Main-memory Management: allocation, protection
- File Management:
  - creation, deletion, directory structures, mapping files to hardware
- I/O System Management: device driver interface, buffering
- Secondary Storage Management
  - free space management, storage allocation, disk scheduling
- Networking: another device to manage - high-speed information flow
- Protection System: specification and enforcement of access controls
- Command-interpreter:
  - e.g. UNIX command line or MS-DOS prompt. A windowing system is just a way to issue the same commands without knowing what they are.

Evolution of CPU utilizations



Eras of Operating Systems

- Time sharing systems (multitasking)



	Batch Multiprogramming	**Time Sharing**
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system	Job control language commands provided with the job	Commands entered at the terminal

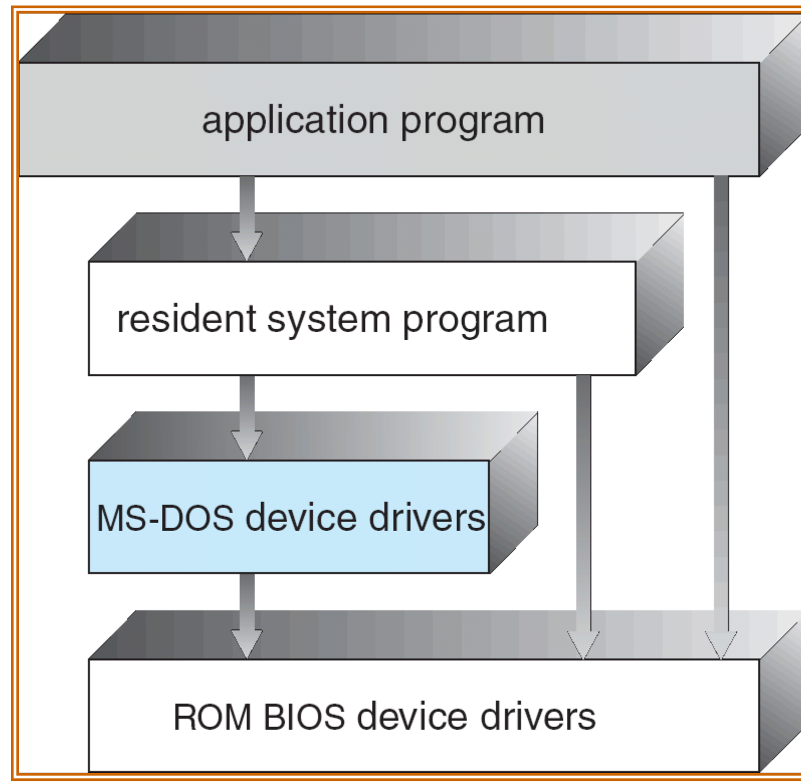
- This is done by switching user tasks or processes transparently. We want each interactive user to get a turn on the CPU quickly - good response time. Need to switch among processes quickly - context switching.

Different Operating Systems Designs

New Jersey Institute of Technology

Simple Structure

- All aspects of the OS linked together in one binary
  - APIs not carefully designed (and/or lots of global variables)
  - Interfaces and levels of functionality not well separated
  - No address protection
- Example: MS-DOS
  - provide the most functionality in the least space
  - Made sense in early days of personal computers with limited processors (e.g. 6502)

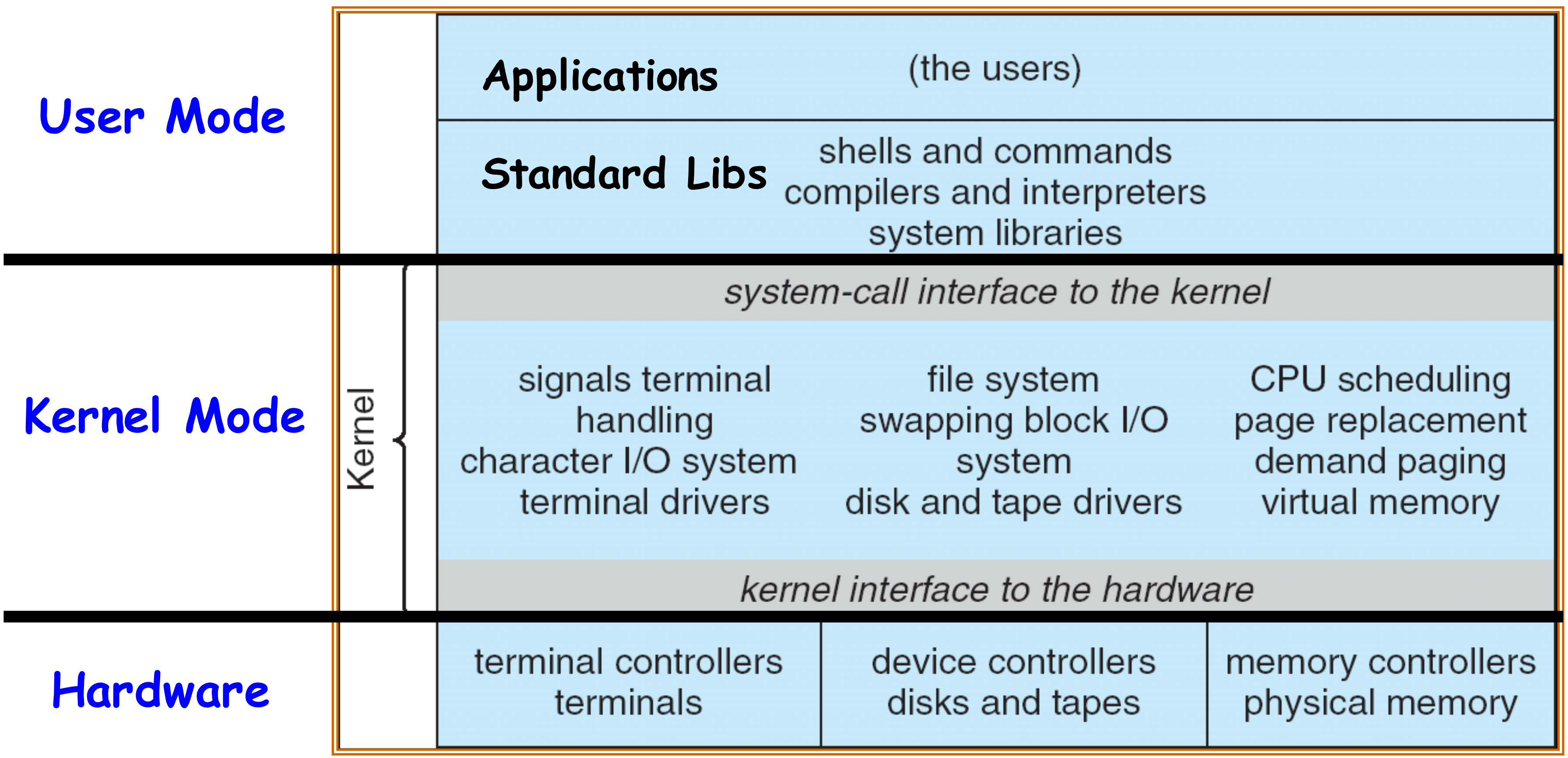


- Advantages?
  - Low memory footprint
- Disadvantages?
  - Very fragile, no enforcement of structure/boundaries

NJIT

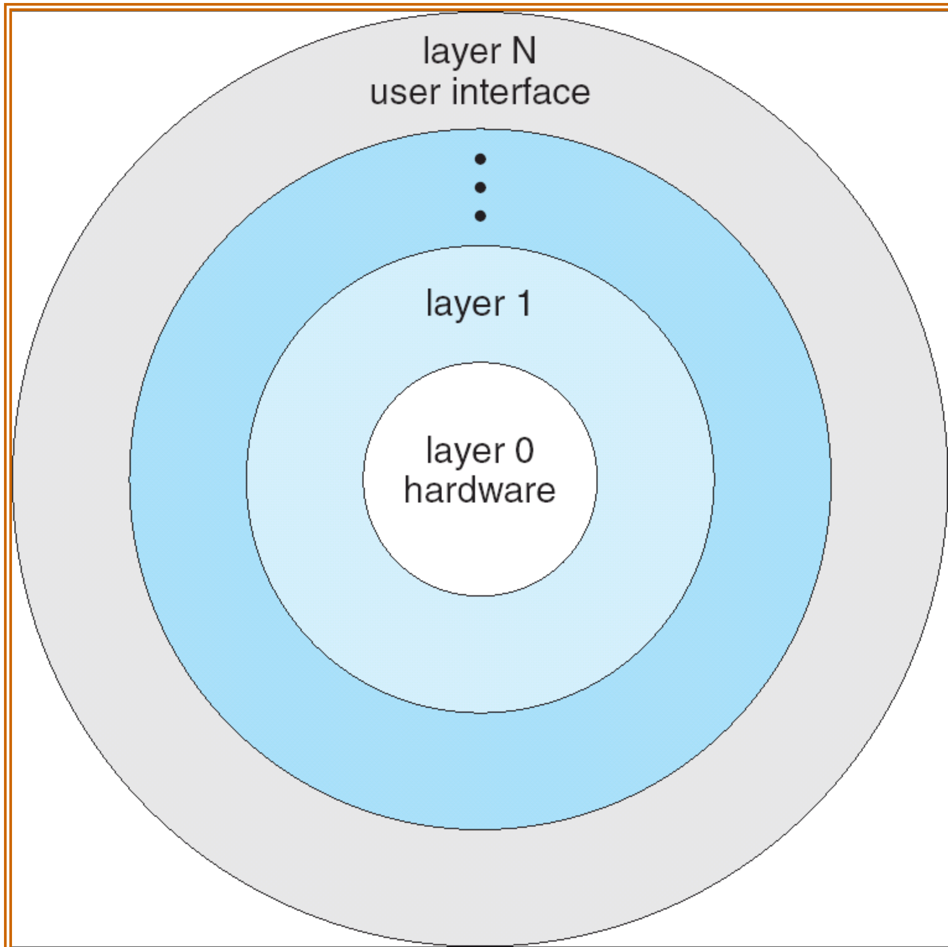
Monolithic Structure: UNIX System Structure

- Two-Layered Structure: User vs Kernel



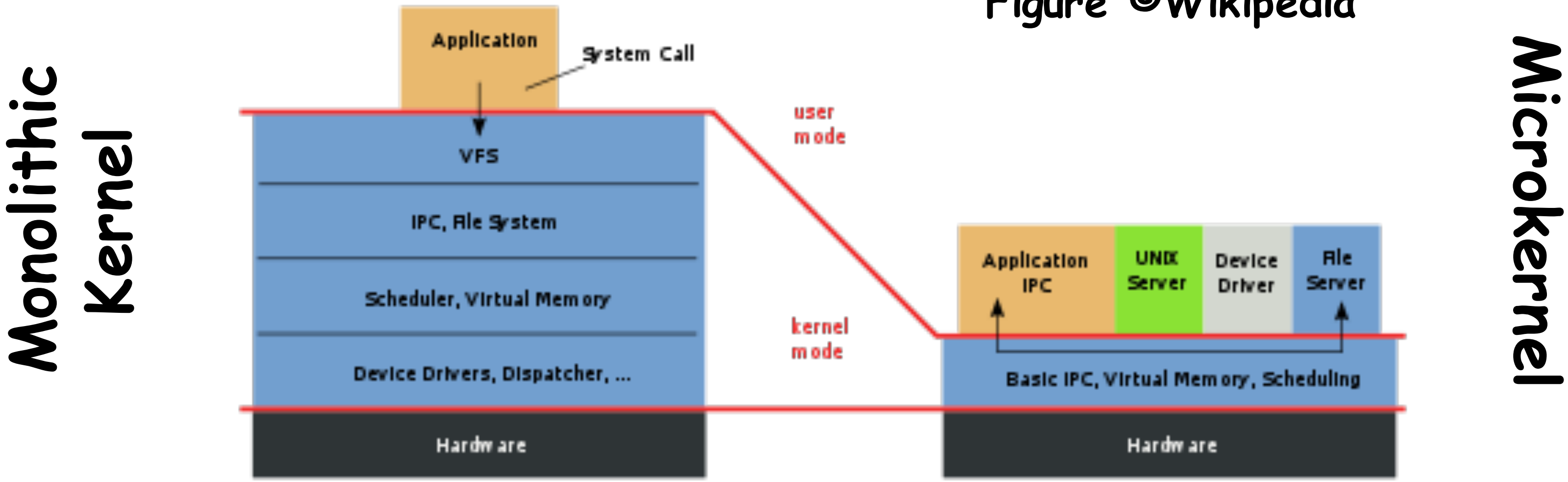
Layered Operating System

- Operating system is divided into many layers (levels)
  - Each built on top of lower layers
  - Bottom layer (layer 0) is hardware
  - Highest layer (layer N) is the user interface
- Each layer uses functions (operations) and services of only lower-level layers
- Machine-dependent vs. independent layers



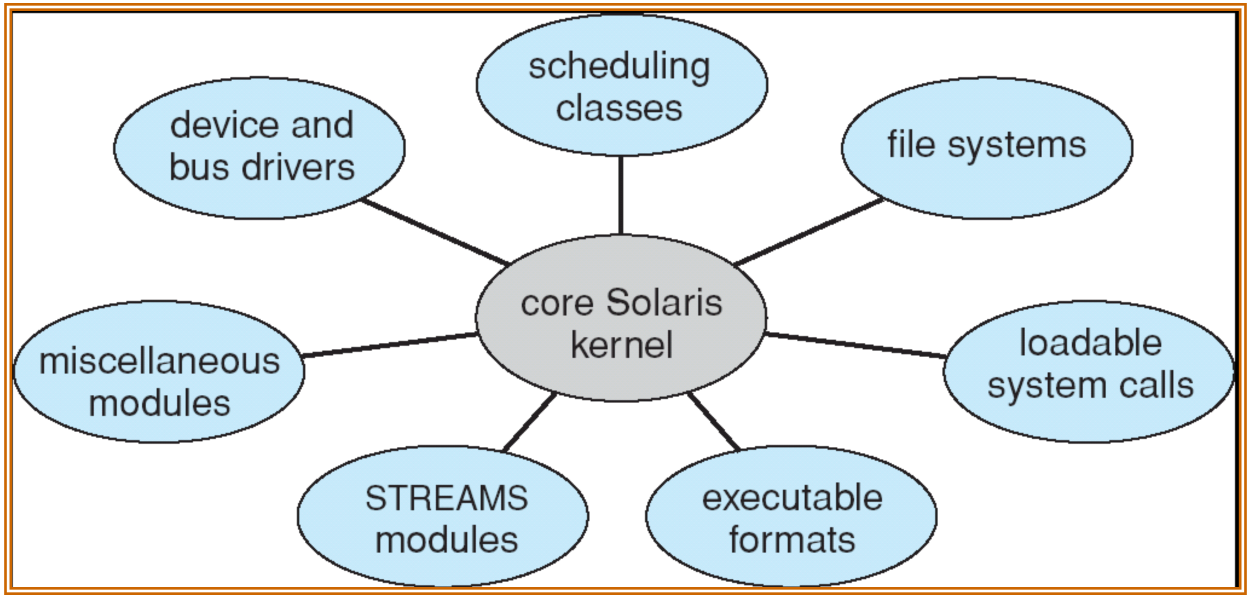
Microkernel Structure

Figure ©Wikipedia



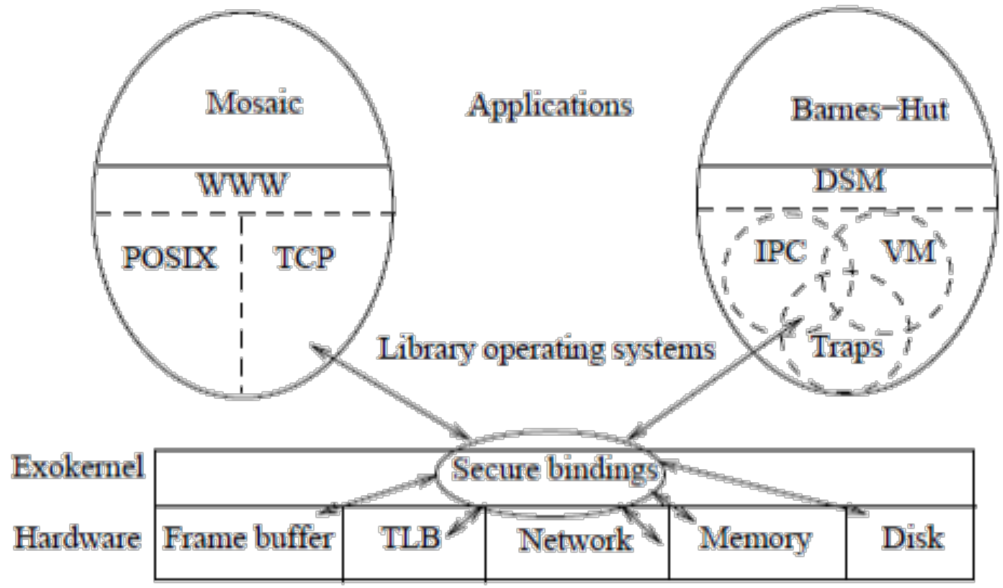
- Moves functionality from the kernel into "*user*" space
  - Small core OS running at kernel level
  - OS Services built from many independent user-level processes
  - Communication between modules with message passing
- Benefits:
  - Easier to extend a microkernel
  - Easier to port OS to new architectures
  - More reliable (less code is running in kernel mode)
  - Fault Isolation (parts of kernel protected from other parts)
  - More secure
- Detriments:
  - Performance overhead can be severe for naïve implementation

Modules-based Structure



- Most modern operating systems implement modules
  - Uses object-oriented approach
- Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Overall, similar to layers but with more flexible
  - May or may not utilize hardware enforcement

ExoKernel: Separate Protection from Management



- Thin layer exports hardware resources directly to users
  - As little abstraction as possible
  - Secure Protection and Multiplexing of resources
- LibraryOS: traditional OS functionality at User-Level
  - Customize resource management for every application
  - Is this a practical approach?
- Very low-level abstraction layer
  - Need extremely specialized skills to develop LibraryOS

Different Operating Systems Designs

- Simple:
  - Only one or two levels of code
- Layered:
  - Lower levels independent of upper levels
- Microkernel:
  - OS built from many user-level processes
- Modular:
  - Core kernel with Dynamically loadable modules
- ExoKernel:
  - Separate protection from management of resources
- Cell-based OS:
  - Two-level scheduling of resources

NJIT

Videos

[History of Apple (OS & Others)]

[History of Mac OS (UI & Features)]

Administrivia

Office hour update: Shaoze Fan, Wednesday from 10:00 AM to 11:30 AM GITC 4325.

NJIT
New Jersey Institute of Technology

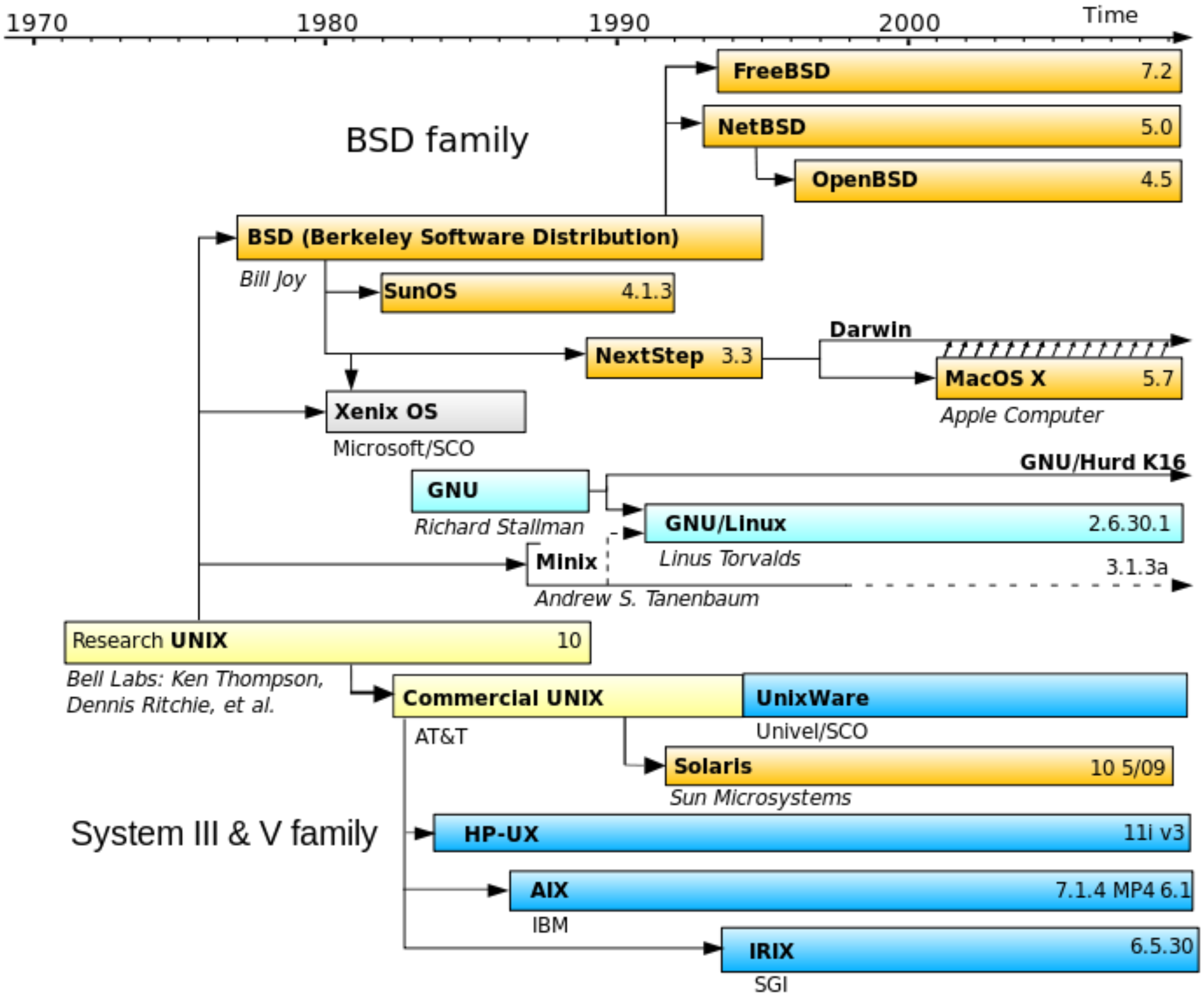
Labs

- Remember concepts is different from using/implementing
- If time allows, we will have in-class demo/lab time
- Form your team ASAP:
  - Each lab will be completed in a team of up to 3 students, and teams may be different for each lab.
- Lab 1 will be released on Feb 5 and due on Feb 19 23:55
  - Your first trial of forming your group
  - Ease you into programming in Linux
  - Bases for a later Lab – writing a shell program

Example Operating Systems

New Jersey Institute of Technology

History of Unix



Videos

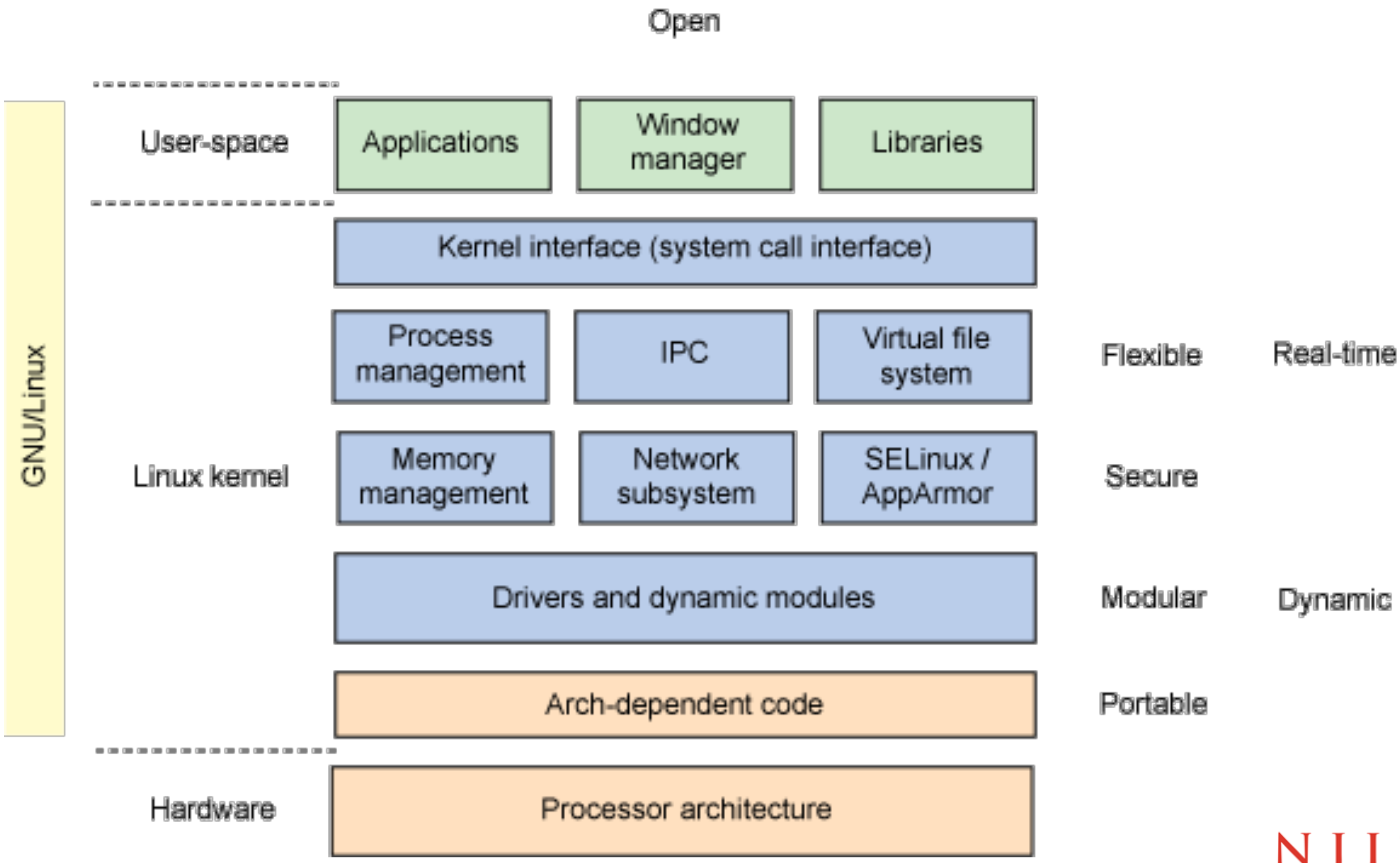
Unix to Linux (Part 1: first 3min & last 1min)

(Part 2: first 3min)

Linux: A Short Documentary

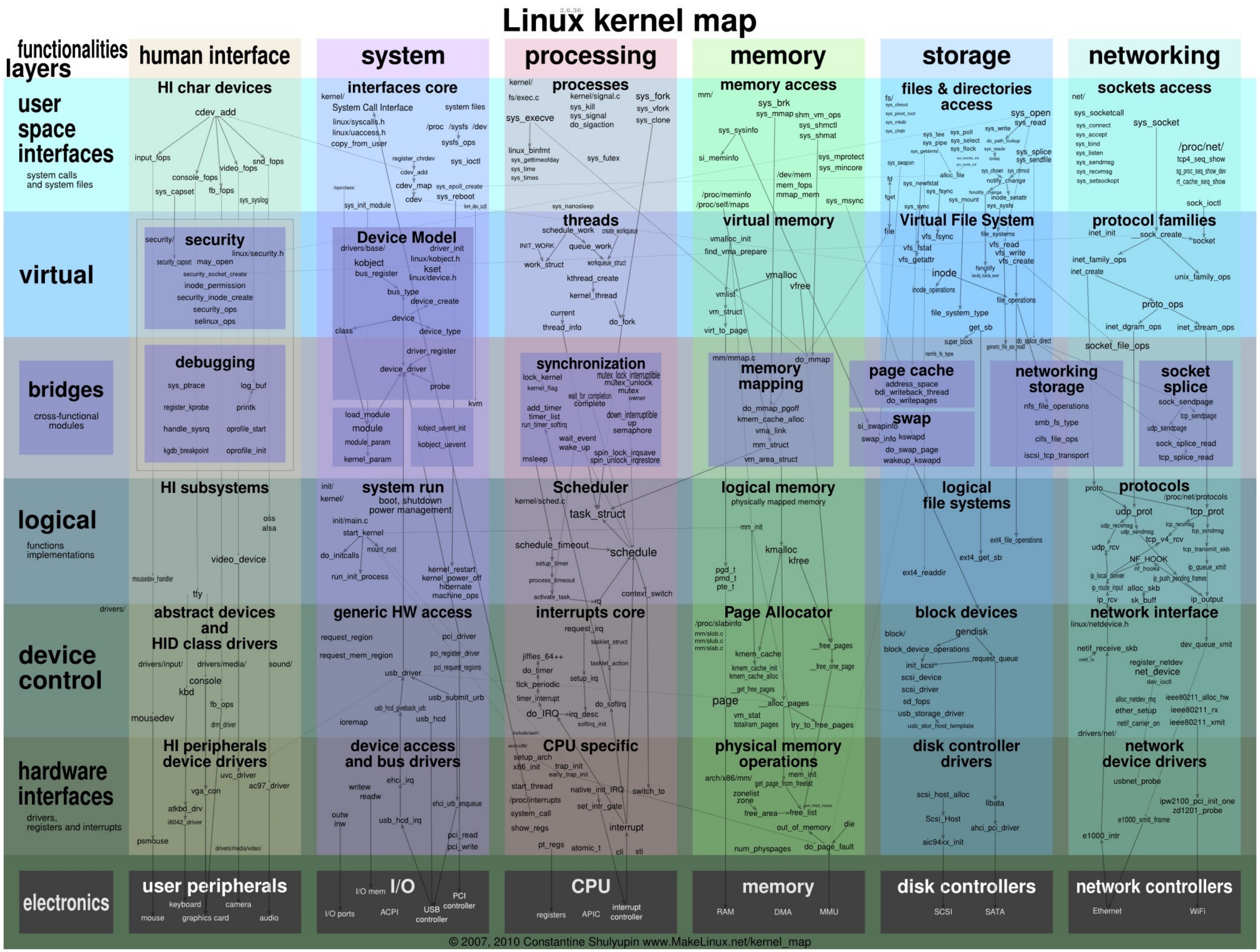
New Jersey Institute of Technology

Linux Structure (Simplified Figure)

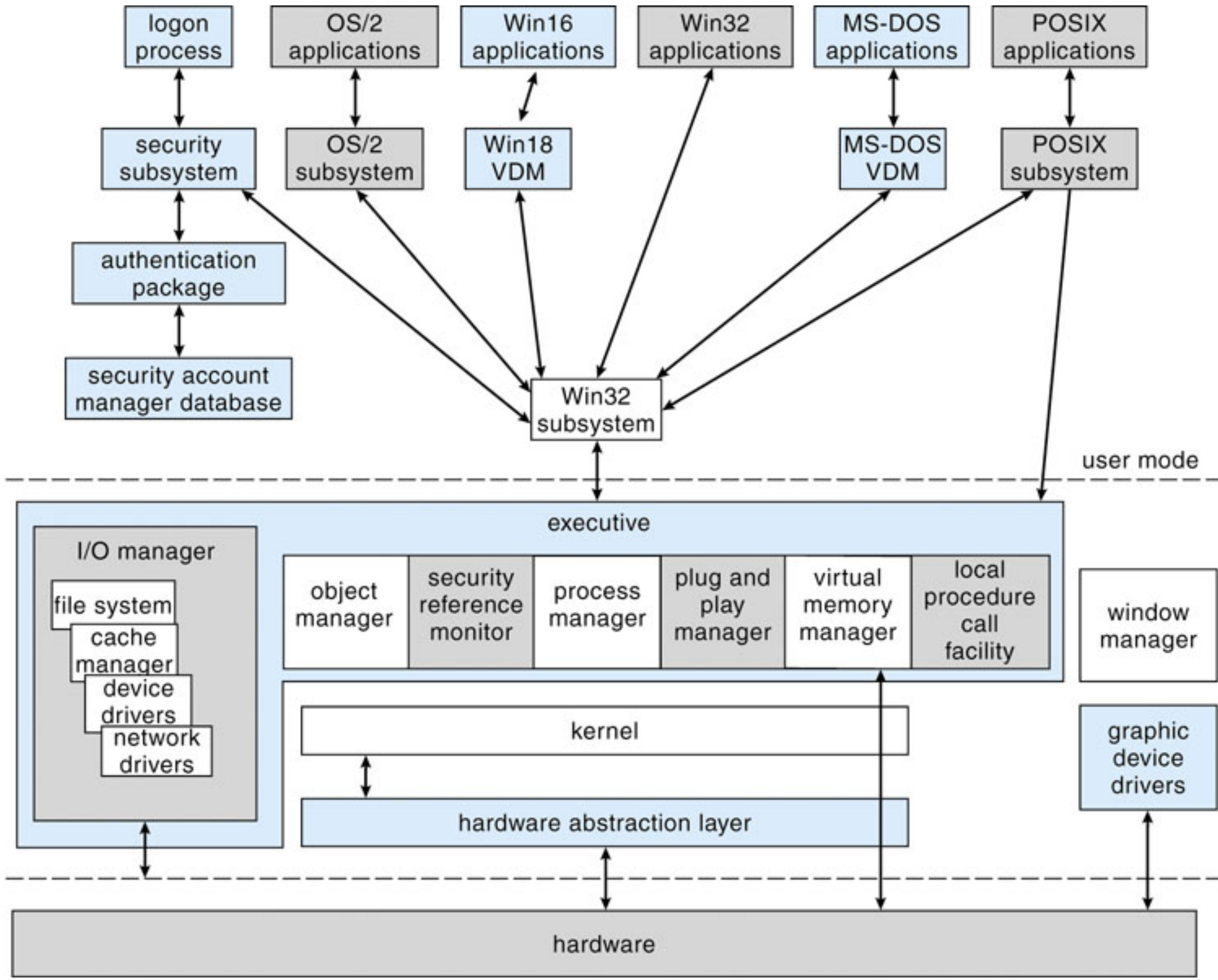


Map of Linux Components

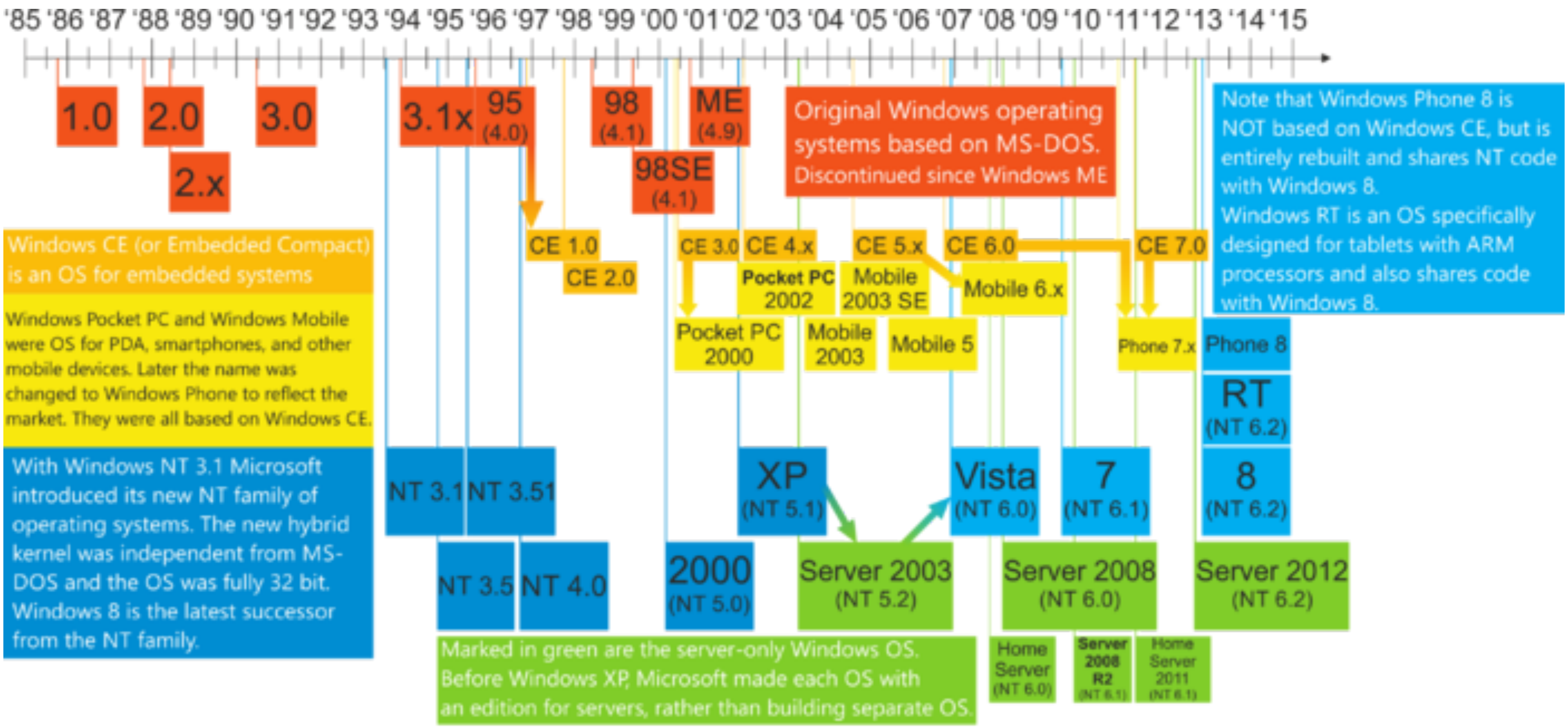
- http://www.makelinux.net/kernel_map/



Microsoft Windows Structure



History of Windows



Explanation of arrows: I. Windows CE is based on code from Windows 95. II. Windows Pocket PC 2000 is based on Windows CE 3.0. III. Windows Mobile 6.x is based on Windows CE 5.x, rather than CE 6.0. IV. Windows Phone 7 is based on code from both Windows CE 6.0 and CE 7.0. V. Windows Vista was built on code from Windows Server 2003, rather than Windows XP.

Videos

[The Evolution of Windows® – From Windows® 1.0 to Windows® 10]

Background

Concurrency

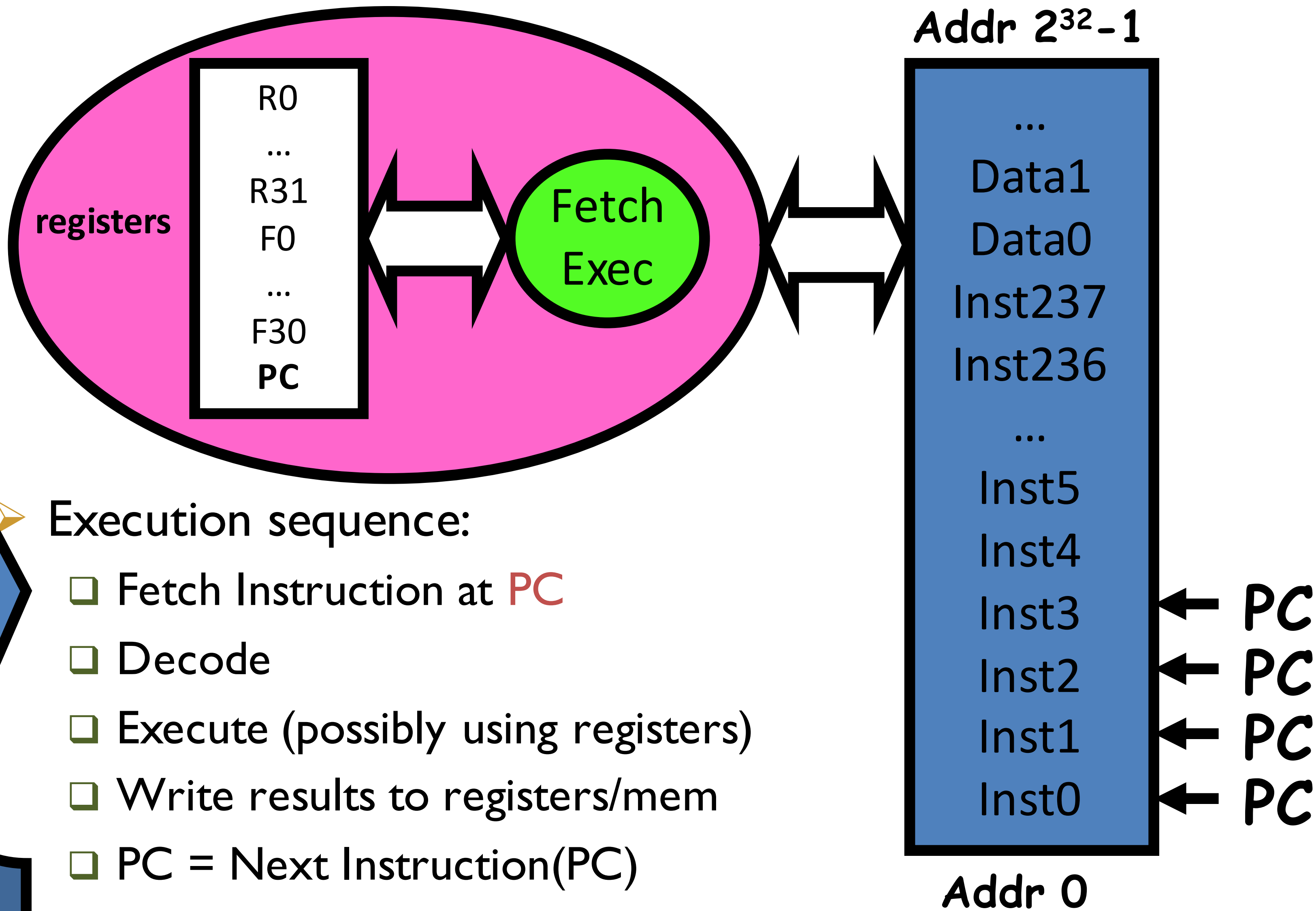
- "Thread" of execution
  - Independent Fetch/Decode/Execute loop
  - Operating in some Address space
- Uniprogramming: *one thread at a time*
  - MS/DOS, early Macintosh, Batch processing
  - Easier for operating system builder
  - Get rid of concurrency by defining it away
  - Does this make sense for personal computers?
- Multiprogramming: *more than one thread at a time*
  - Multics, UNIX/Linux, OS/2, Windows NT/2000/XP, Mac OS X
  - Often called "multitasking", but multitasking has other meanings
- Multicore $\Rightarrow$ Multiprogramming, right?

The Basic Problem of Concurrency

- The basic problem of concurrency involves resources:
  - Hardware (simple): single CPU, single DRAM, single I/O devices
  - Multiprogramming API: users think they have exclusive access to shared resources
- OS Has to coordinate all activity
  - Multiple users, I/O interrupts, …
  - How can it keep all these things straight?
- Basic Idea: Use abstraction
  - Decompose hard problem into simpler ones
  - Abstract the notion of an executing program
  - Then, worry about multiplexing these abstract machines

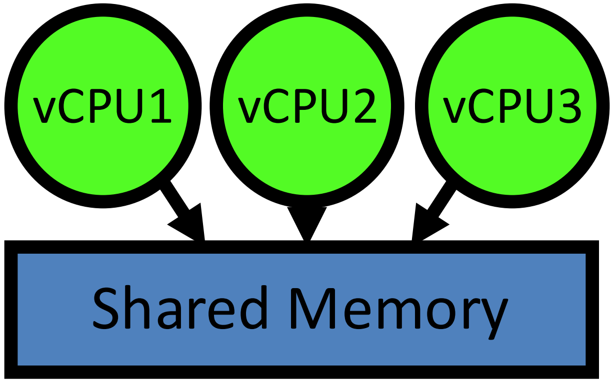
NJIT

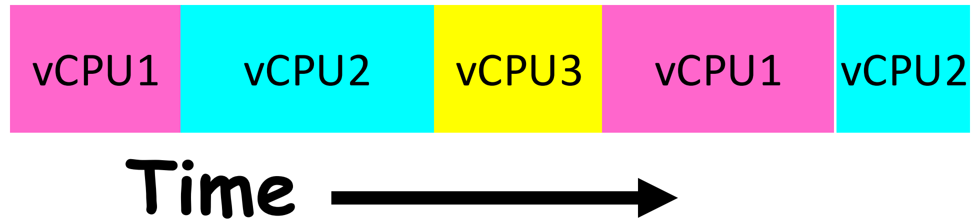
Recall: What happens during program execution?



- Execution sequence:
  - Fetch Instruction at PC
  - Decode
  - Execute (possibly using registers)
  - Write results to registers/mem
  - PC = Next Instruction(PC)
  - Repeat

How to give the illusion of multiple processors?



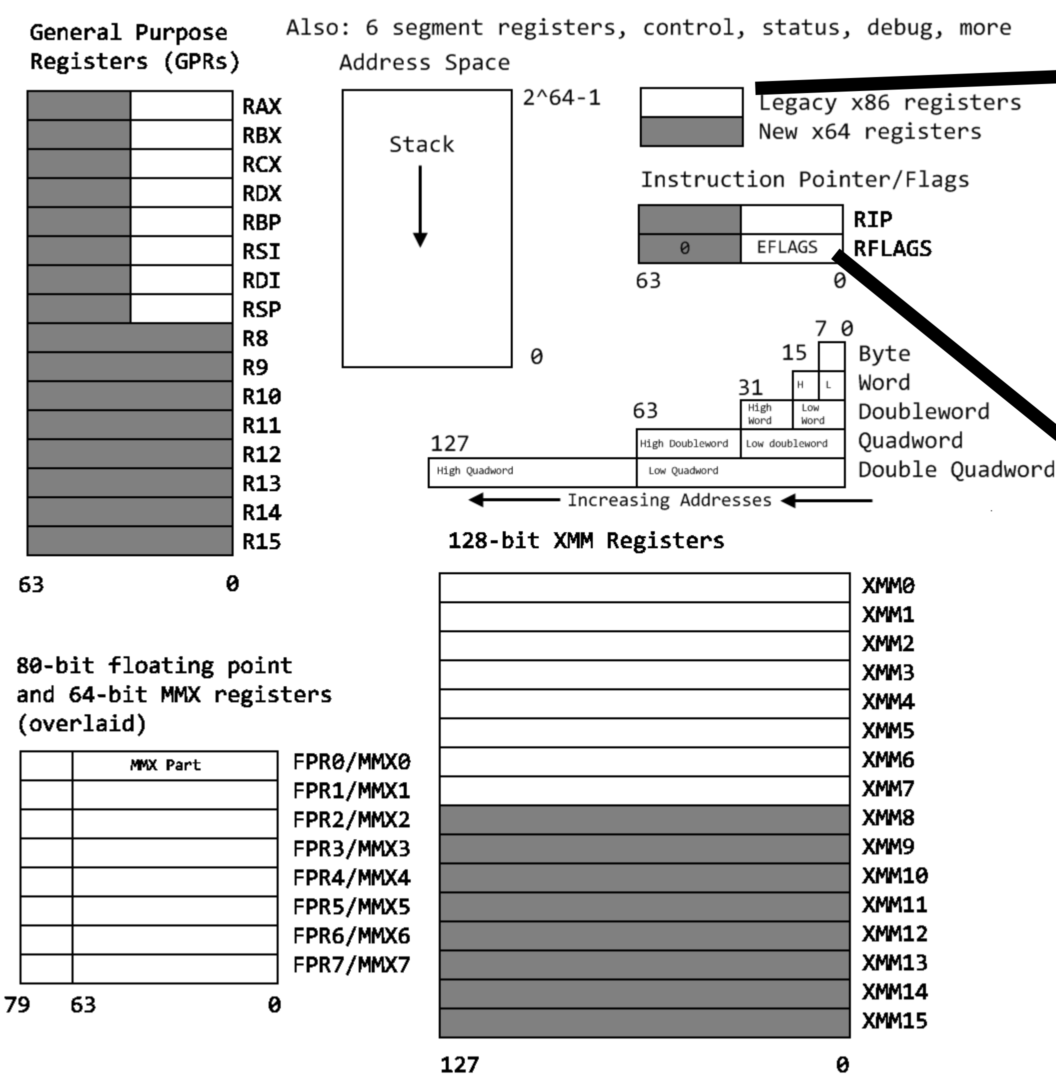


- Assume a single processor. How do we provide the illusion of multiple processors?
  - Multiplex in time!
- Each virtual "CPU" needs a structure to hold:
  - Program Counter (PC), Stack Pointer (SP)
  - Registers (Integer, Floating point, others…?)
  - Call result a "Thread" for now…
- How to switch from one CPU to the next?
  - Save PC, SP, and registers in current state block
  - Load PC, SP, and registers from new state block
- What triggers switch?
  - Timer, voluntary yield, I/O, other things

NJIT

What needs to be saved in Modern X86?

64-bit Register Set

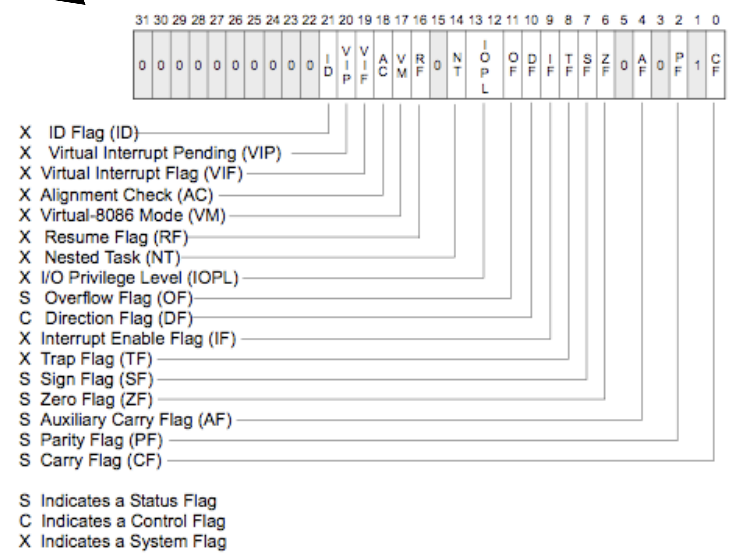


Traditional 32-bit subset

General-Purpose Registers

31 … 16	15 … 8	7 … 0	16-bit	32-bit
	AH	AL	AX	EAX
	BH	BL	BX	EBX
	CH	CL	CX	ECX
	DH	DL	DX	EDX
	BP			EBP
	SI			ESI
	DI			EDI
	SP			ESP

EFLAGS Register



Property of the simple multiprogramming technique

- All virtual CPUs share same non-CPU resources
  - I/O devices the same
  - Memory the same
- Consequence of sharing:
  - Each thread can access the data of every other thread (good for sharing, bad for protection)
  - Threads can share instructions (good for sharing, bad for protection)
  - Can threads overwrite OS functions?
- This (unprotected) model common in:
  - Embedded applications
  - Windows 3.1/Machintosh (switch only with yield)
  - Windows 95 (switch with both yield and timer)

NJIT

How to protect threads from one another?

Need three important things:

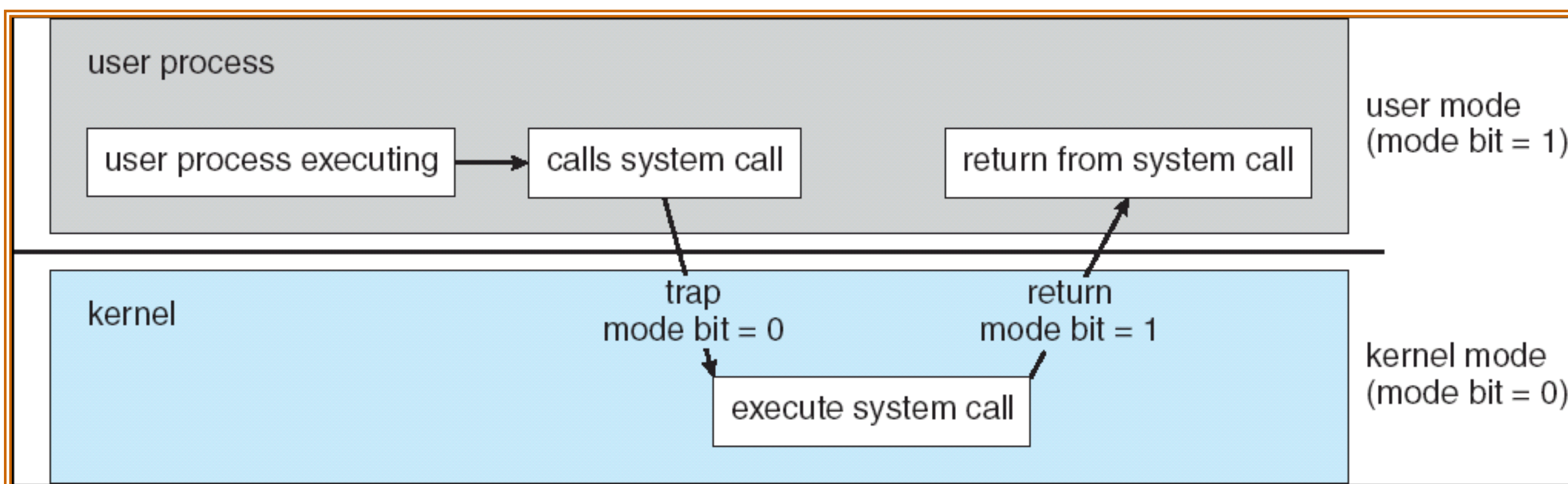
- Protection of memory
  - Every task does not have access to all memory
- Protection of I/O devices
  - Every task does not have access to every device
- Protection of Access to Processor: Preemptive switching from task to task
  - Use of timer
  - Must not be possible to disable timer from user code

Protecting Threads from Each Other

- Problem: Run multiple applications in such a way that they are protected from one another
- Goal:
  - Keep User Programs from Crashing OS
  - Keep User Programs from Crashing each other
  - [Keep Parts of OS from crashing other parts?]
- (Some of the required) Mechanisms:
  - Address Translation
  - Dual Mode Operation
- Simple Policy:
  - Programs are not allowed to read/write memory of other Programs or of Operating System

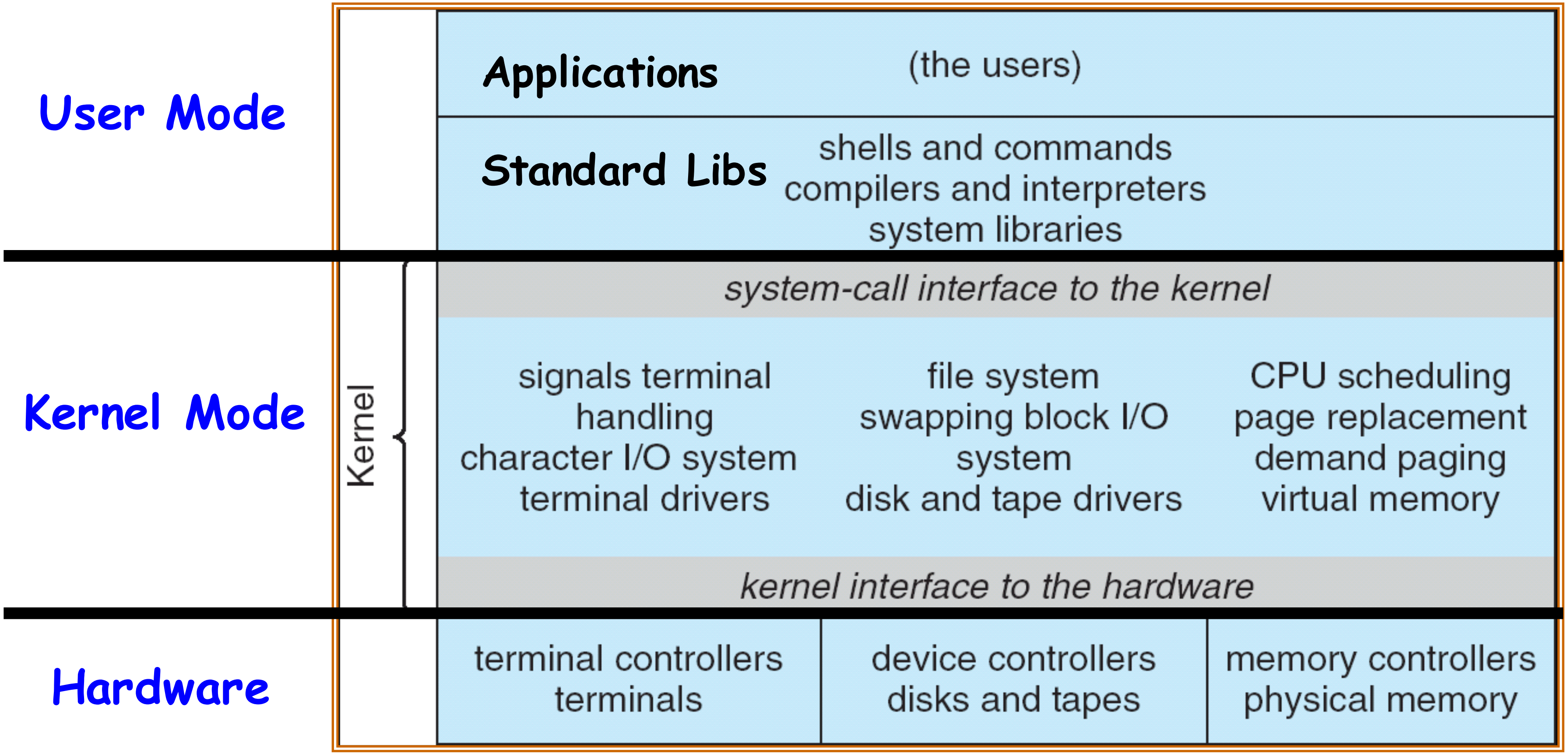
Dual Mode Operation

- Hardware provides at least two modes:
  - "Kernel" mode (or "supervisor" or "protected")
  - "User" mode: Normal programs executed
- Some instructions/ops prohibited in user mode:
  - Example: cannot modify page tables in user mode
    - Attempt to modify ⇒ Exception generated
- Transitions from user mode to kernel mode:
  - System Calls, Interrupts, Other exceptions



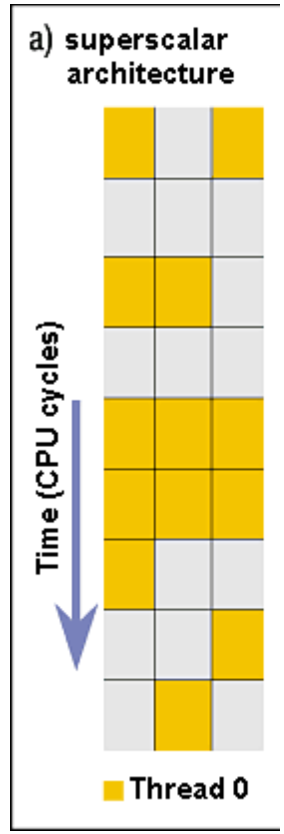
UNIX System Structure

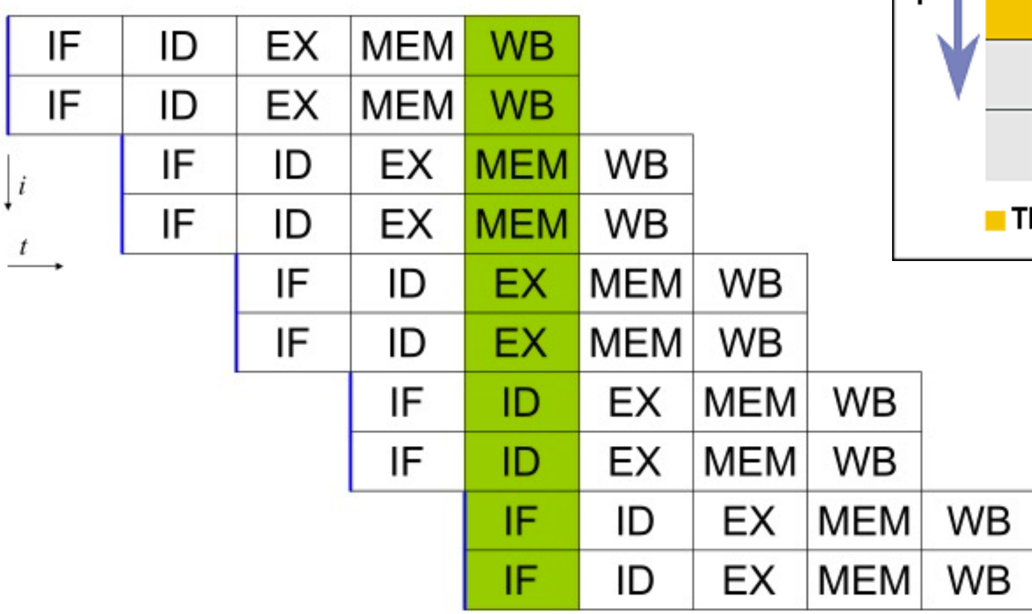
- Dual mode: User & Kernel Modes



Modern Technique: SMT/Hyperthreading

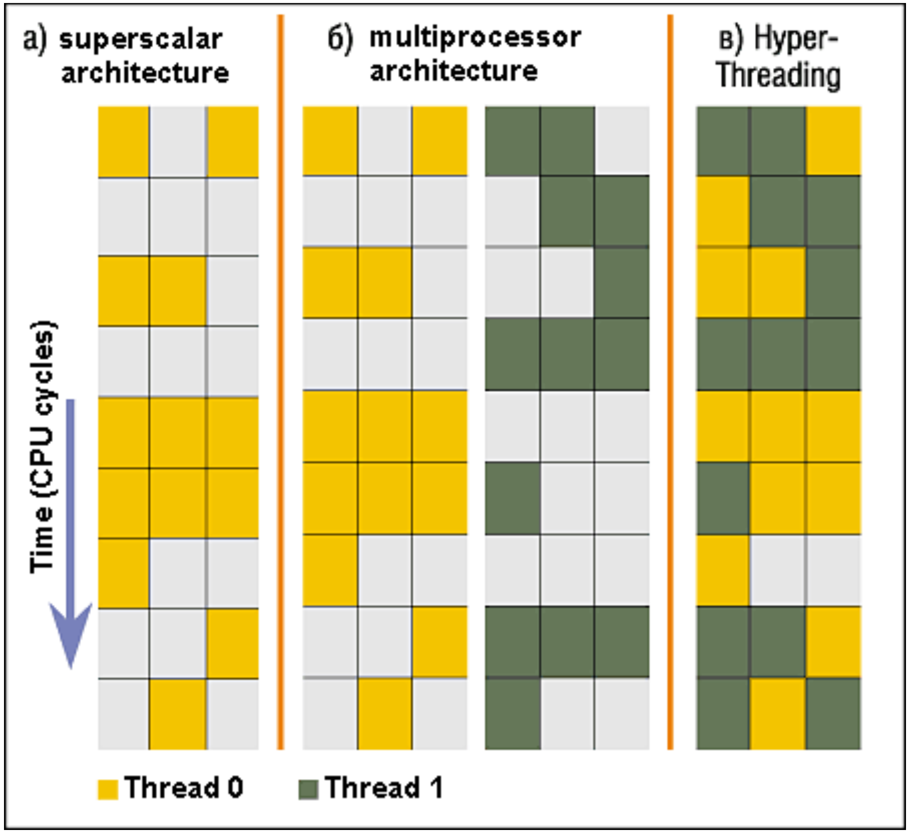
- Superscalar processor
  - instruction-level parallelism within a single processor





Modern Technique: SMT/Hyperthreading

- Hardware technique
  - Exploit natural properties of superscalar processors to provide illusion of multiple processors
  - Higher utilization of processor resources
- Can schedule each thread as if were separate CPU
  - However, not linear speedup!
  - If have multiprocessor, should schedule each processor first
- Original technique called "Simultaneous Multithreading"
  - See http://www.cs.washington.edu/research/smt/
  - Alpha, SPARC, Pentium 4 ("Hyperthreading"), Power 5



NJIT