On to I/O via Virtual Memory

David E. Culler

CS162 – Operating Systems and Systems Programming

Lecture 20

October 15, 2014

Reading: A&D 11.2 (OSC 13)
HW 4 out
Proj 2 out

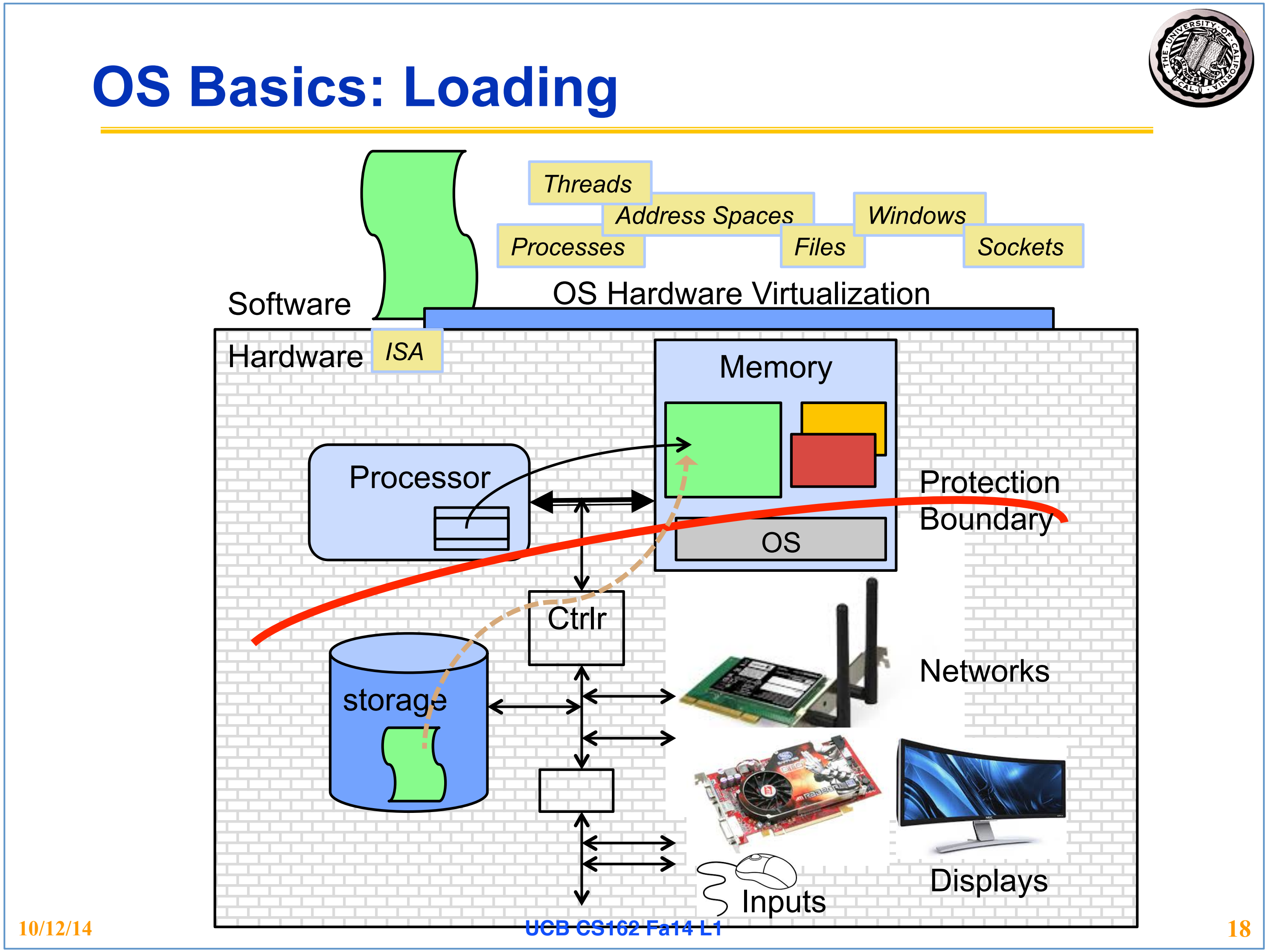
… bottom lines of the long road here

- Virtual-to-physical address translation provides the *illusion* of a large, sparsely occupied virtual address space for every process
  - Used to solve many OS requirements
    - Protection, memory allocation, multi-programming, sharing, fast IO (!!!)
  - Implemented through mapping structures
    - Exponents matter (2^20 millions, 2^30, 2^60 gazillions)
    - Has to be VERY FAST in the common (success) case
- Caches provide the *illusion* of a very large, fast physical memory
  - Essential to performance (100x) on modern machines
- Similar techniques and trade-offs in cache of translations and memory blocks
- All aspects of modern OS design must be cache friendly
  - Slow OS perceived as Slow Computer

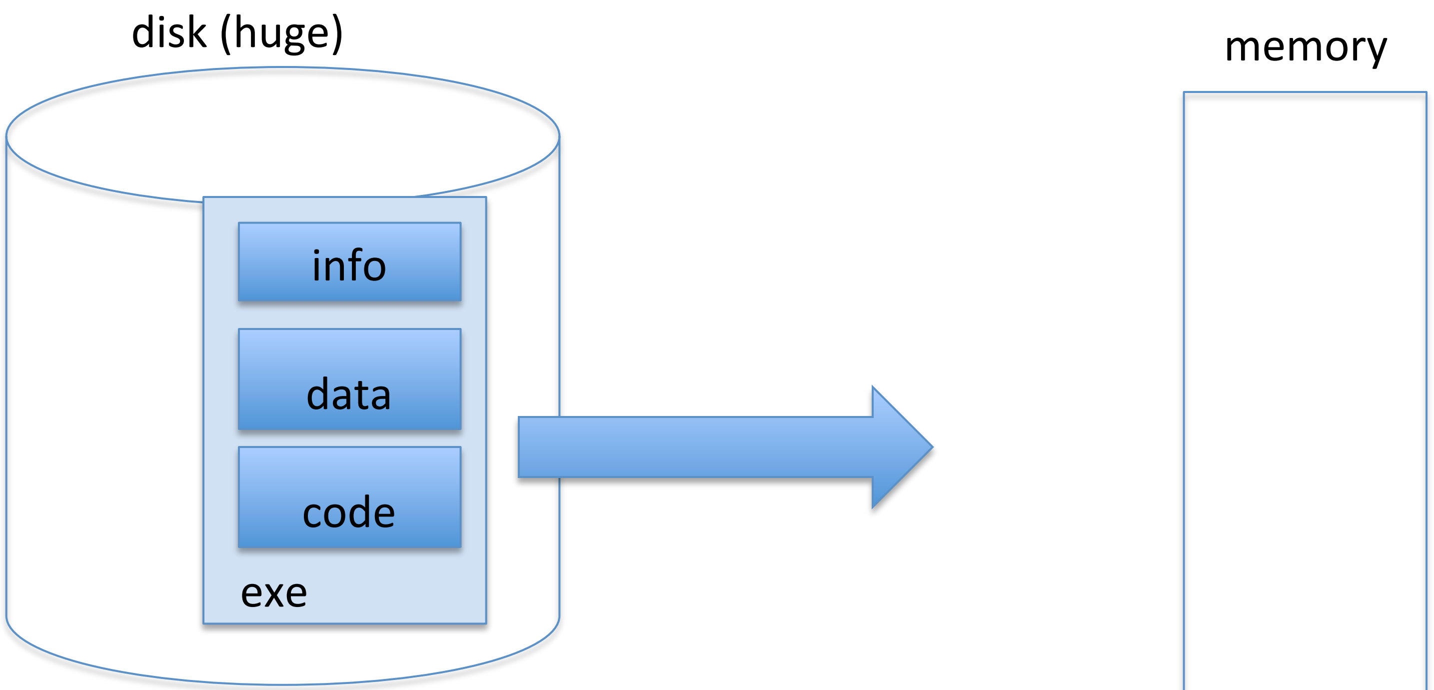
Objectives

- Recall and solidify understanding the concept and mechanics of caching.
- Understand how caching and caching effects pervade OS design.
- Put together all the mechanics around TLBs, Paging, and Memory caches
- Solidify understanding of Virtual Memory

Recall: the most basic OS function

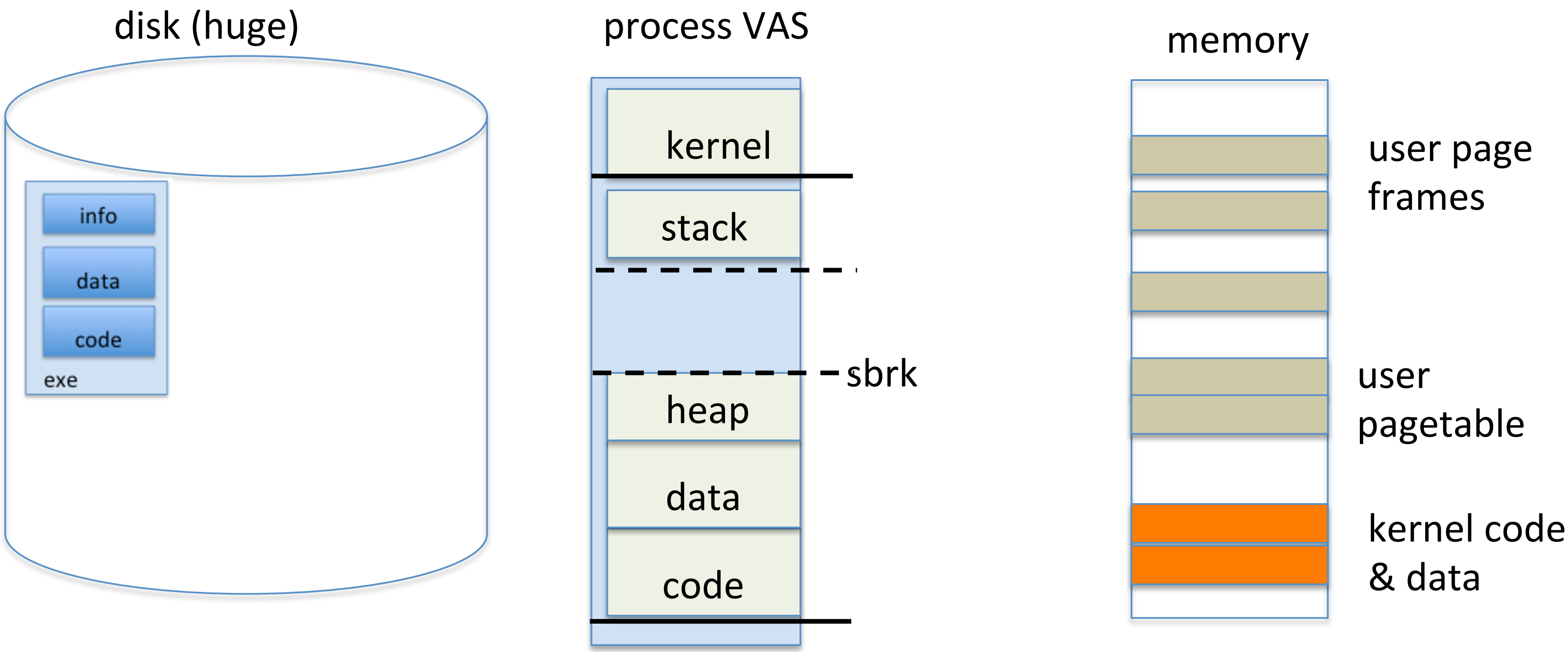


Loading an executable into memory



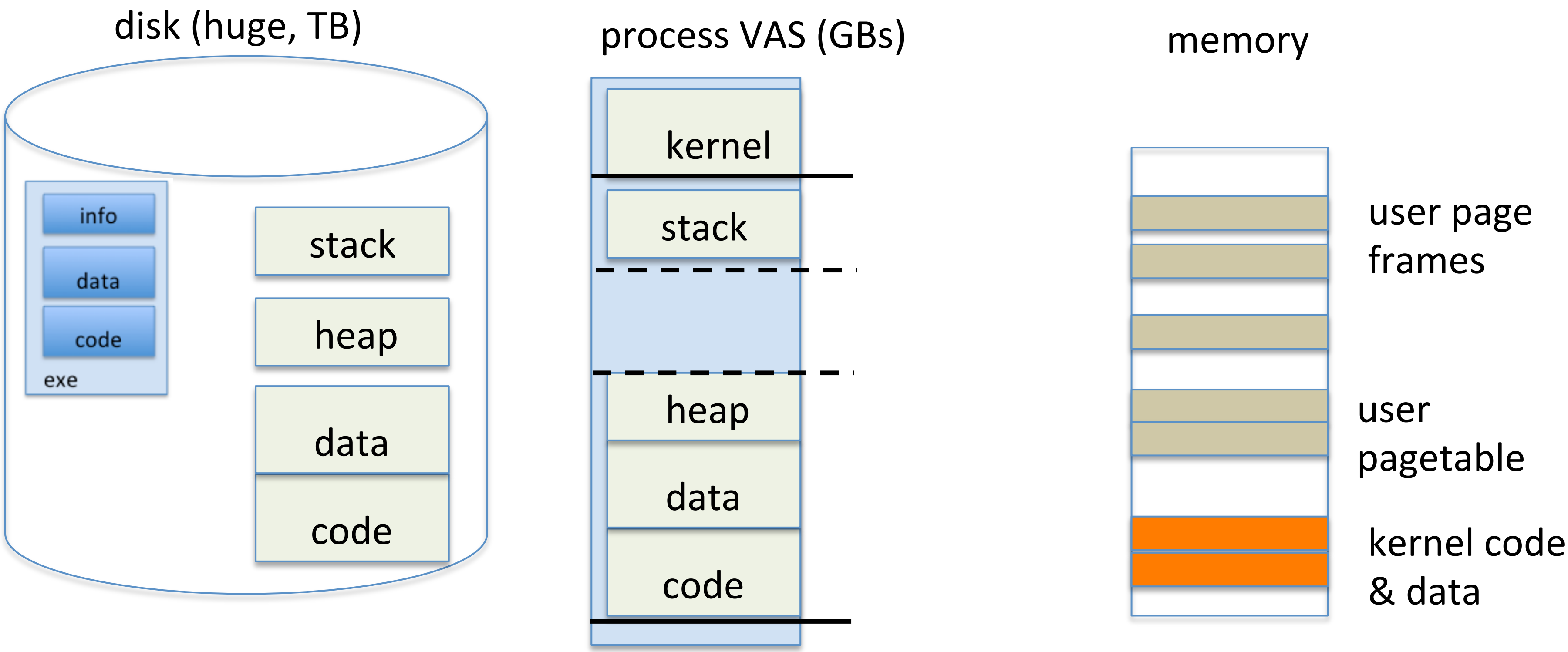
- .exe
  - lives on disk in the file system
  - contains contents of code & data segments, relocation entries and symbols
  - OS loads it into memory, initializes registers (and initial stack pointer)
  - program sets up stack and heap upon initialization: CRT0

Create Virtual Address Space of the Process



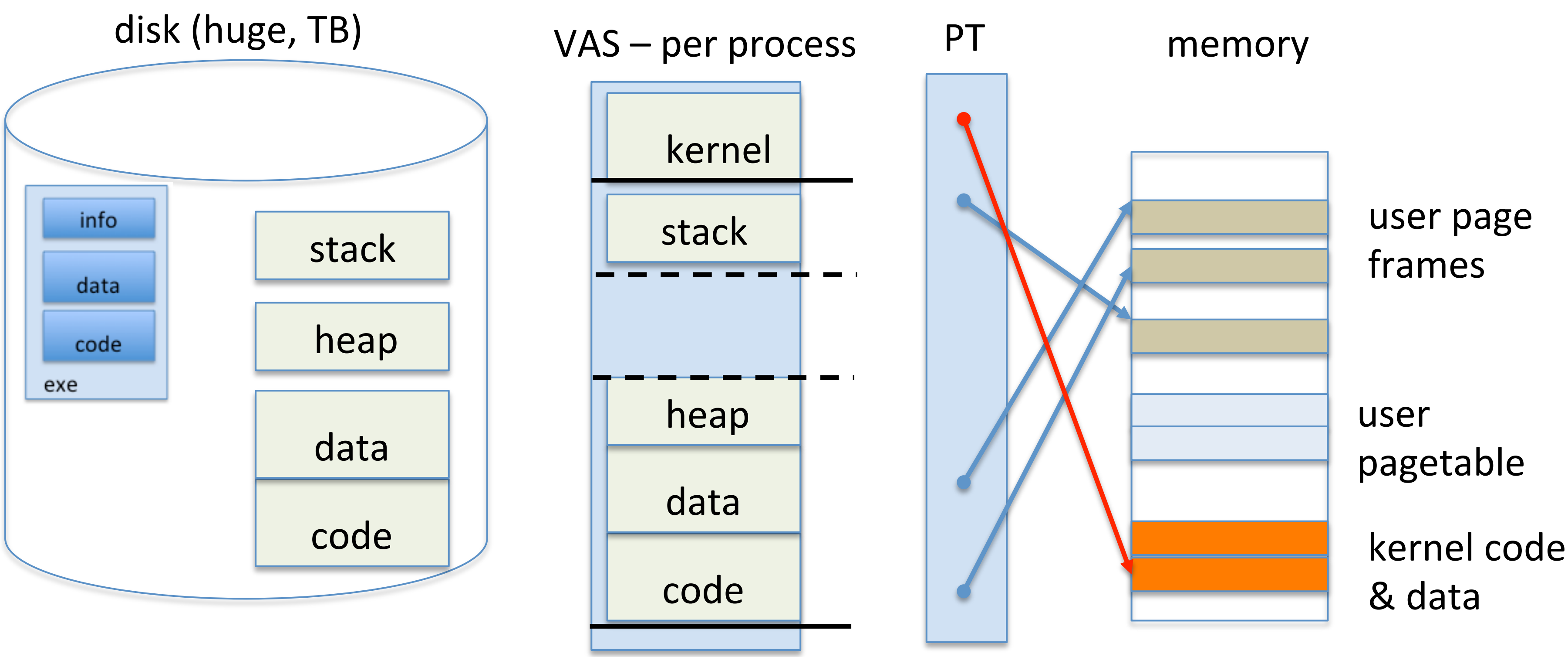
- Utilized pages in the VAS are backed by page blocks on disk
  - called the backing store
  - typically in an optimized block store, but can think of it like a file

Create Virtual Address Space of the Process



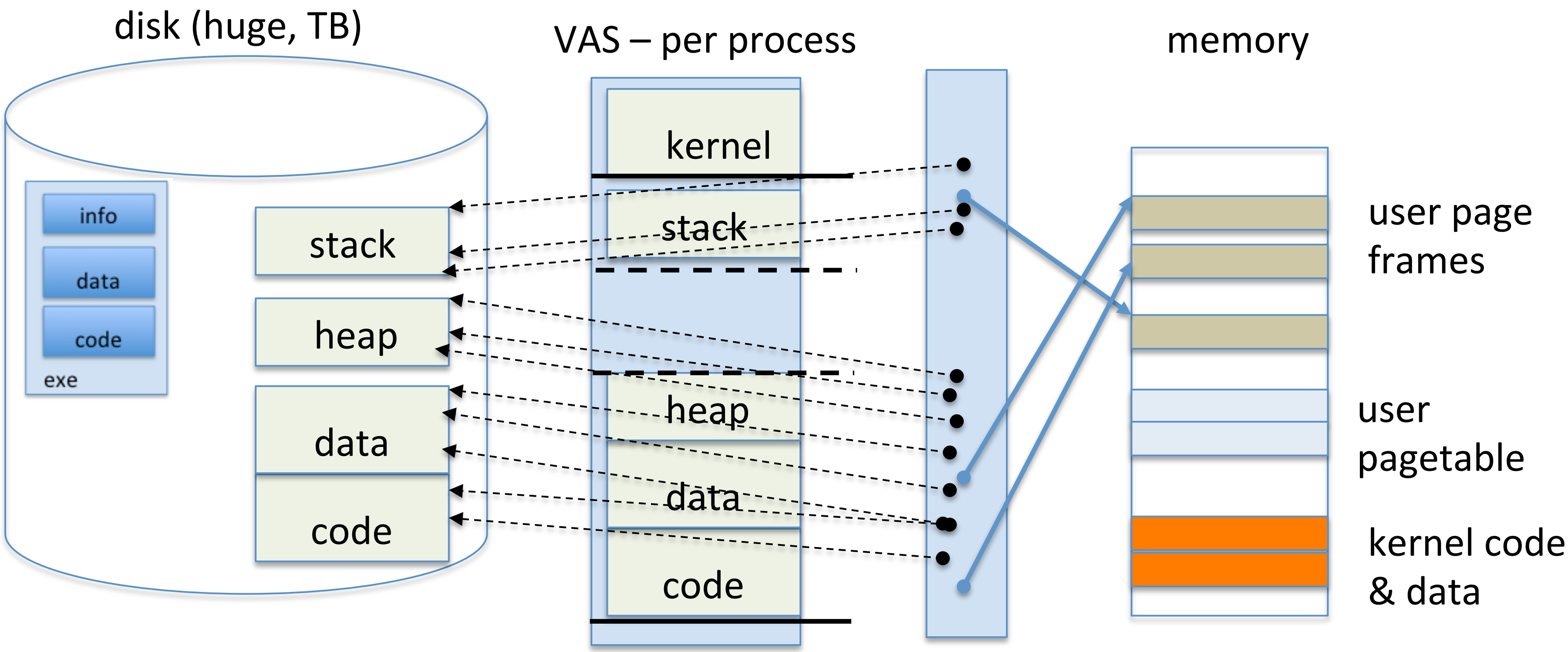
- User Page table maps entire VAS
- All the utilized regions are backed on disk
  - swapped into and out of memory as needed
- For *every* process

Create Virtual Address Space of the Process



- User Page table maps entire VAS
  – resident pages to the frame in memory they occupy
  – the portion of it that the HW needs to access must be resident in memory

Provide Backing Store for VAS



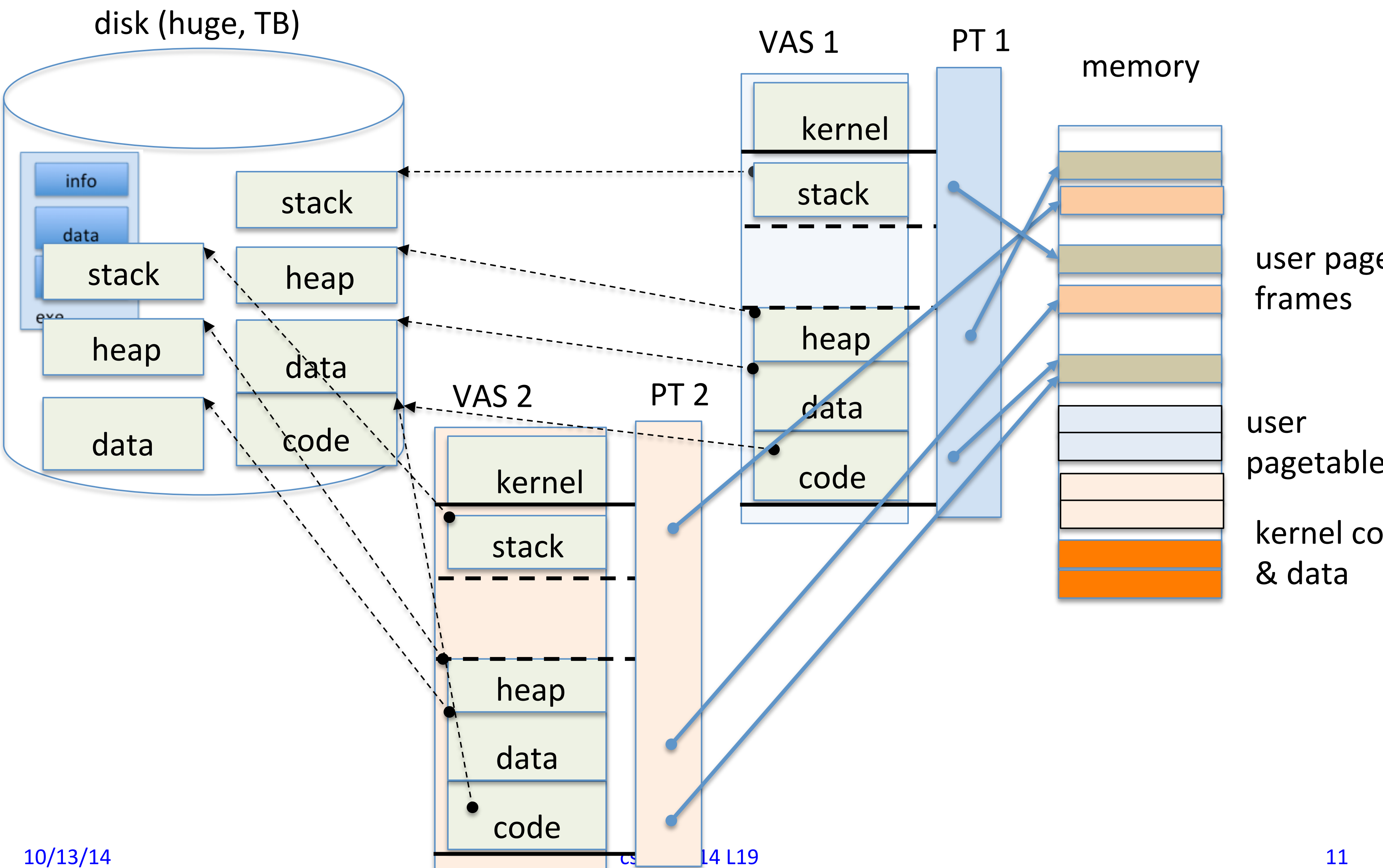
- User Page table maps entire VAS
- Resident pages mapped to memory frames
- For all other pages, OS must record where to find them on disk

What data structure is required to map non-resident pages to disk?

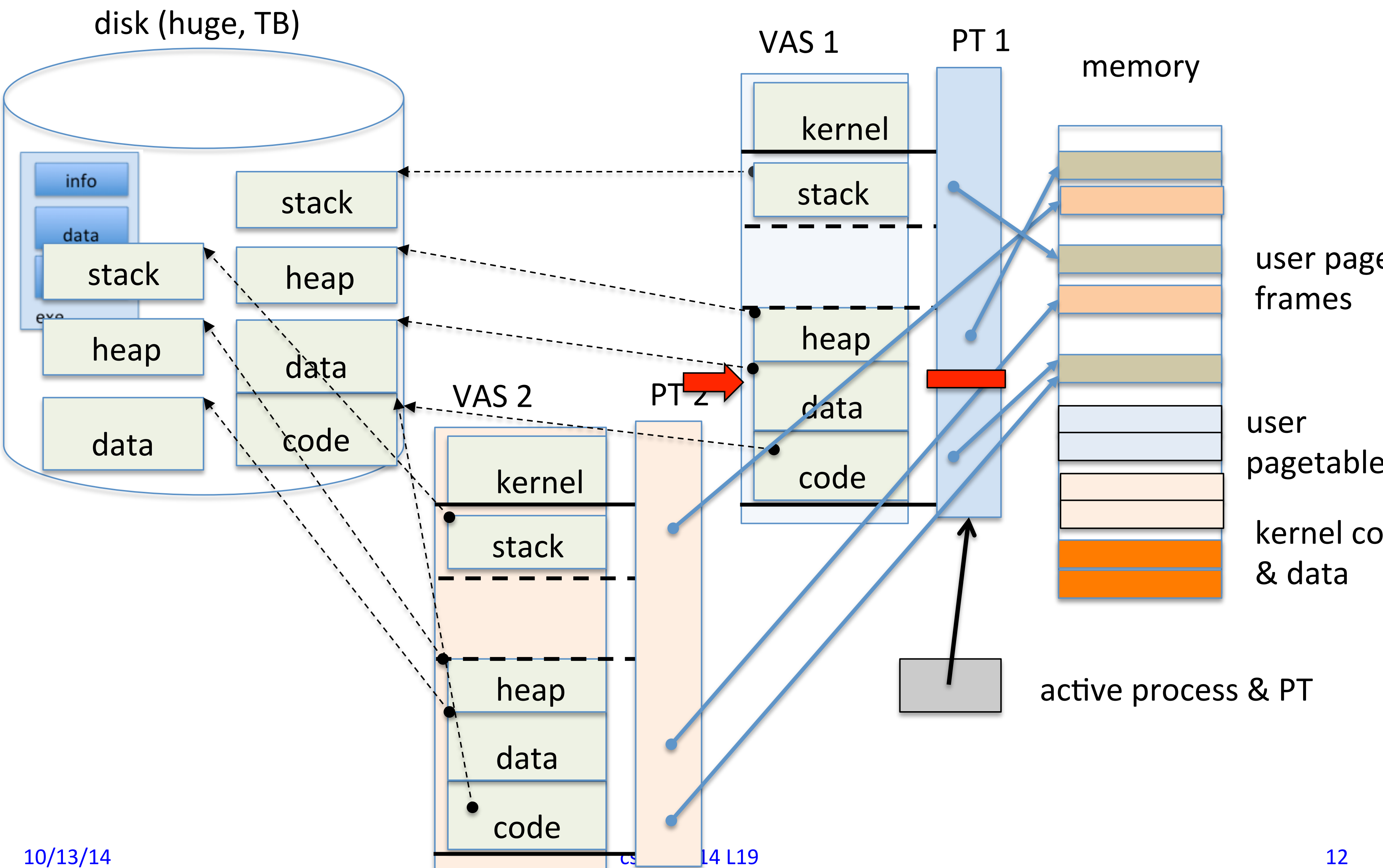
- FindBlock(PID, page#) => disk_block

- Like the PT, but purely software
- Where to store it?
- Usually want backing store for resident pages too.
- Could use hash table (like Inverted PT)

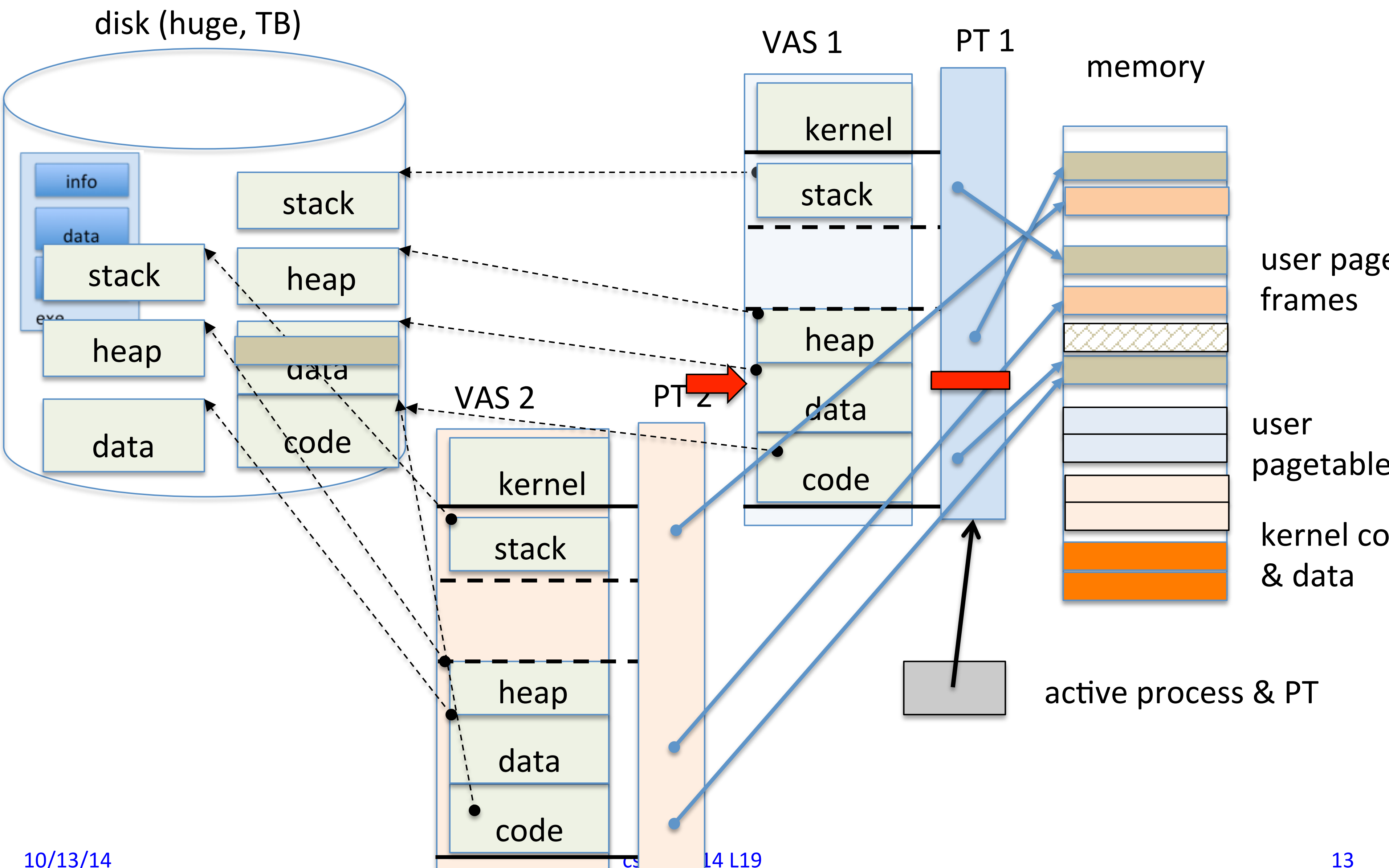
Provide Backing Store for VAS



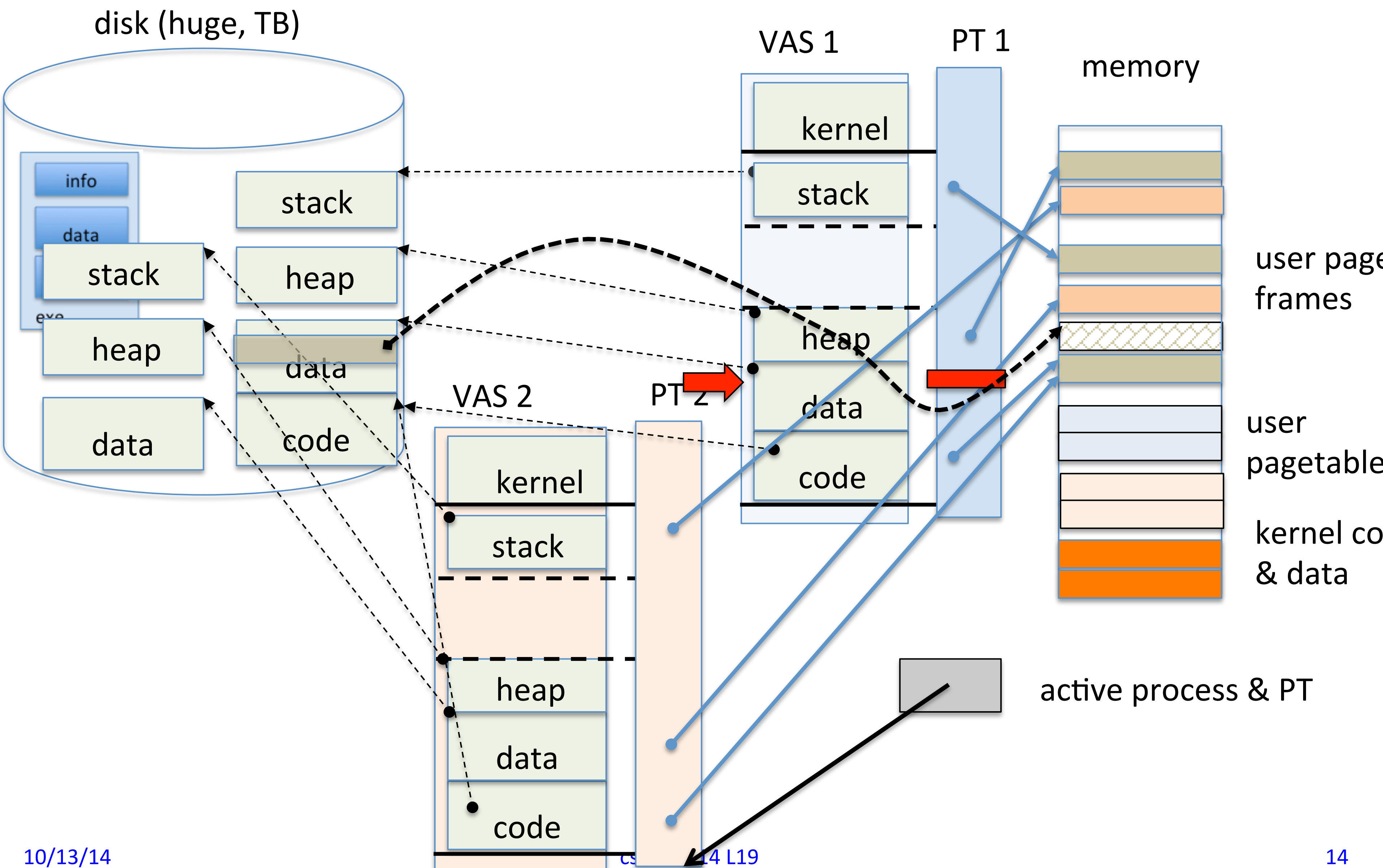
On page Fault …



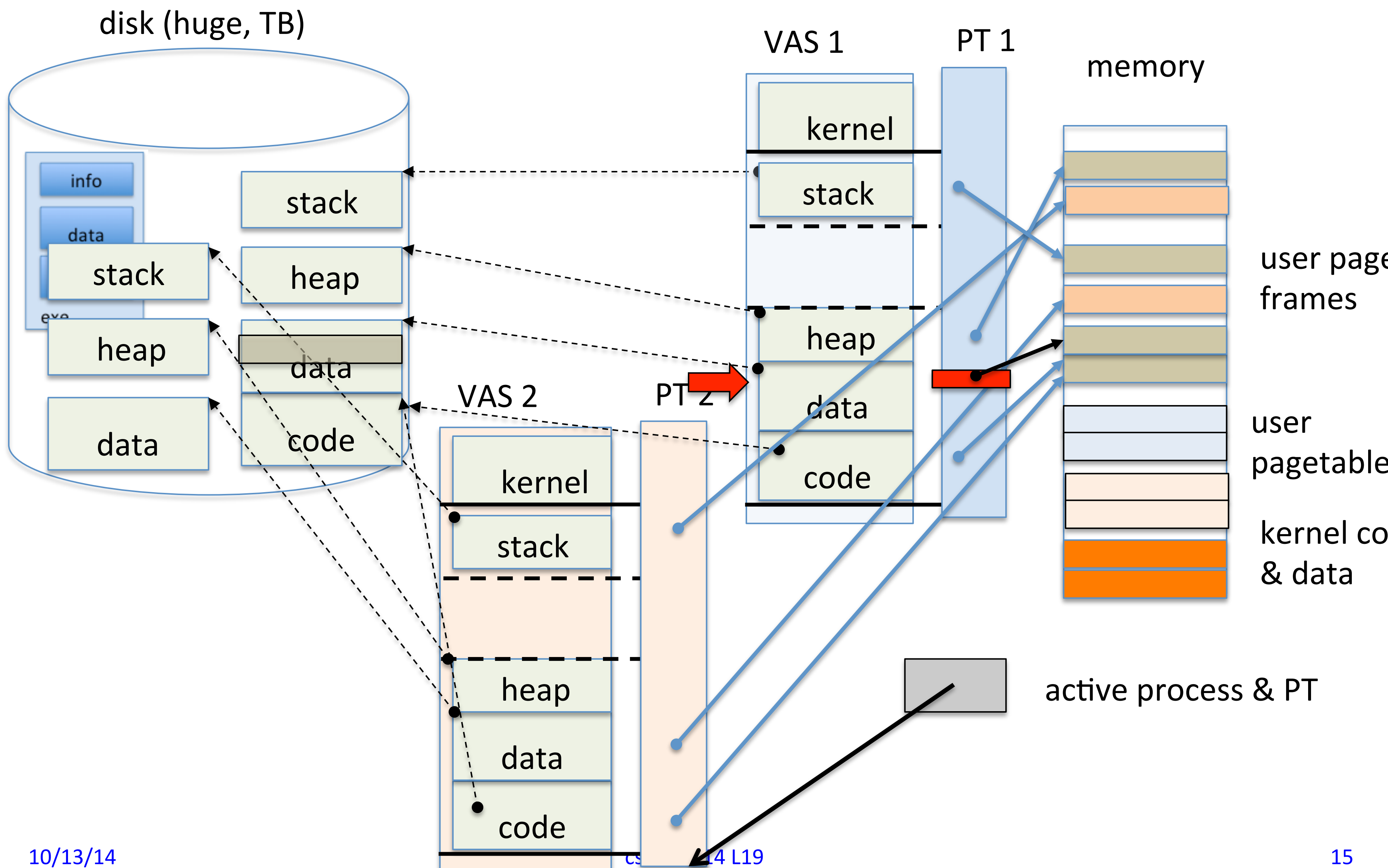
On page Fault … find & start load



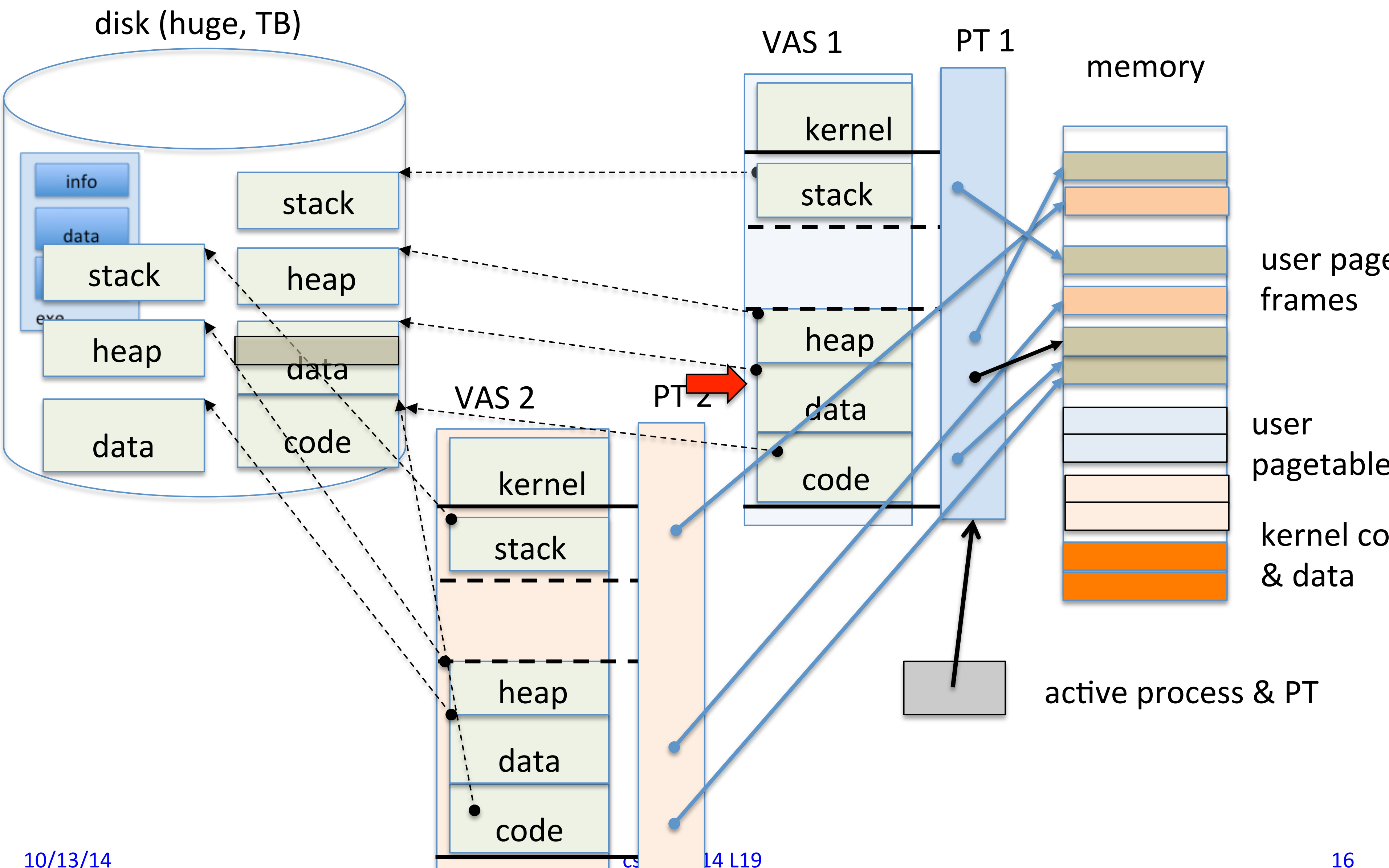
On page Fault … schedule other P or T



On page Fault … update PTE



Eventually reschedule faulting thread



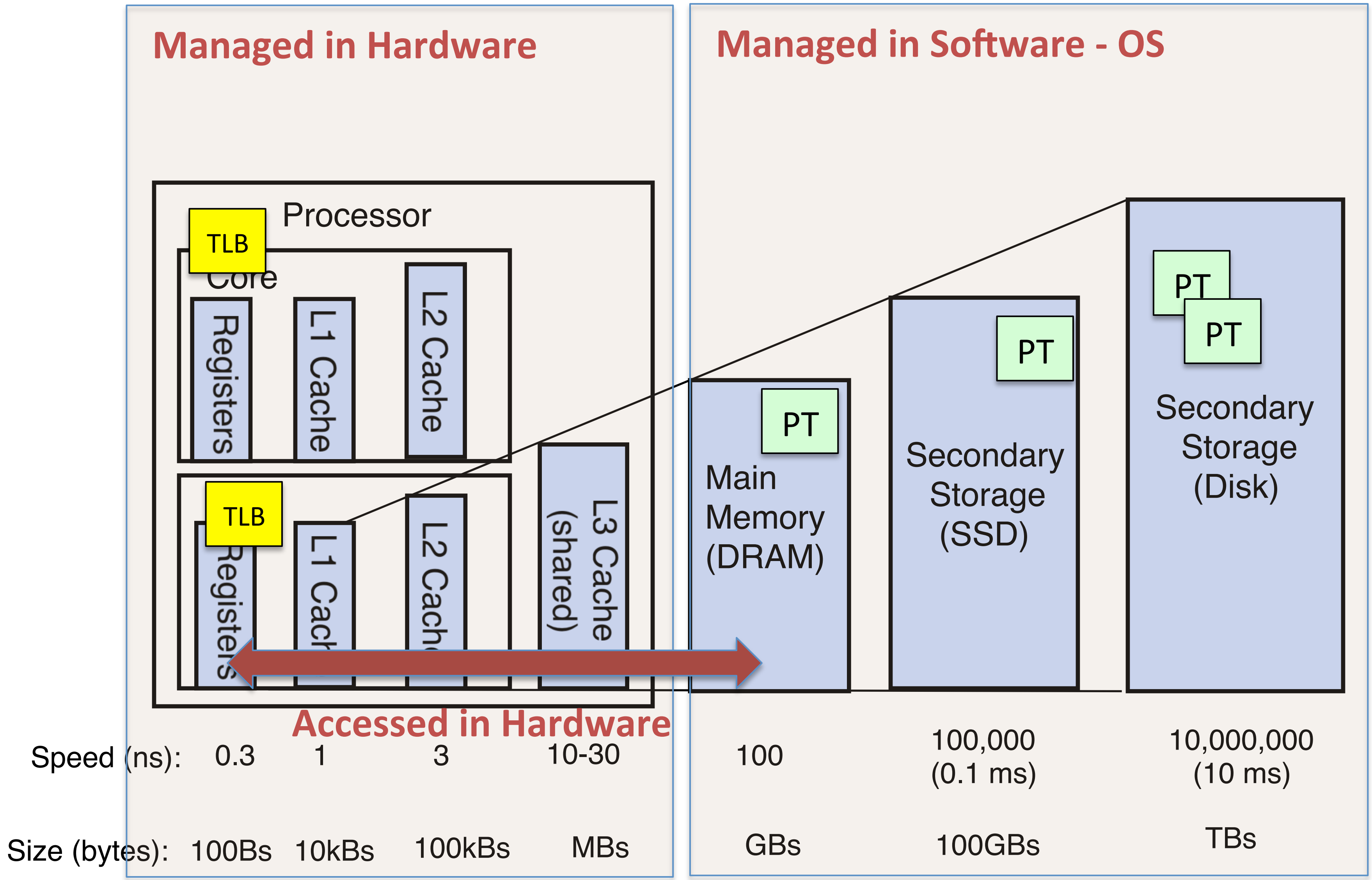
Where does the OS get the frame?

- Keeps a free list
- Unix runs a "reaper" if memory gets too full
- As a last resort, evict a dirty page first

How many frames per process?

- Like thread scheduling, need to “schedule” memory resources
  - allocation of frames per process
    - utilization? fairness? priority?
  - allocation of disk paging bandwith

Management & Access to the Memory Hierarchy



Summary

- Virtual address space for protection, efficient use of memory, AND multi-programming.
  - hardware checks & translates when present
  - OS handles EVERYTHING ELSE
- Conceptually memory is just a cache for blocks of VAS that live on disk
  - but can never access the disk directly
- Address translation provides the basis for sharing
  - shared blocks of disk AND shared pages in memory
- How else can we use this mechanism?
  - sharing ???
  - disks transfers on demand ???
  - accessing objects in blocks using load/store instructions

Historical Perspective

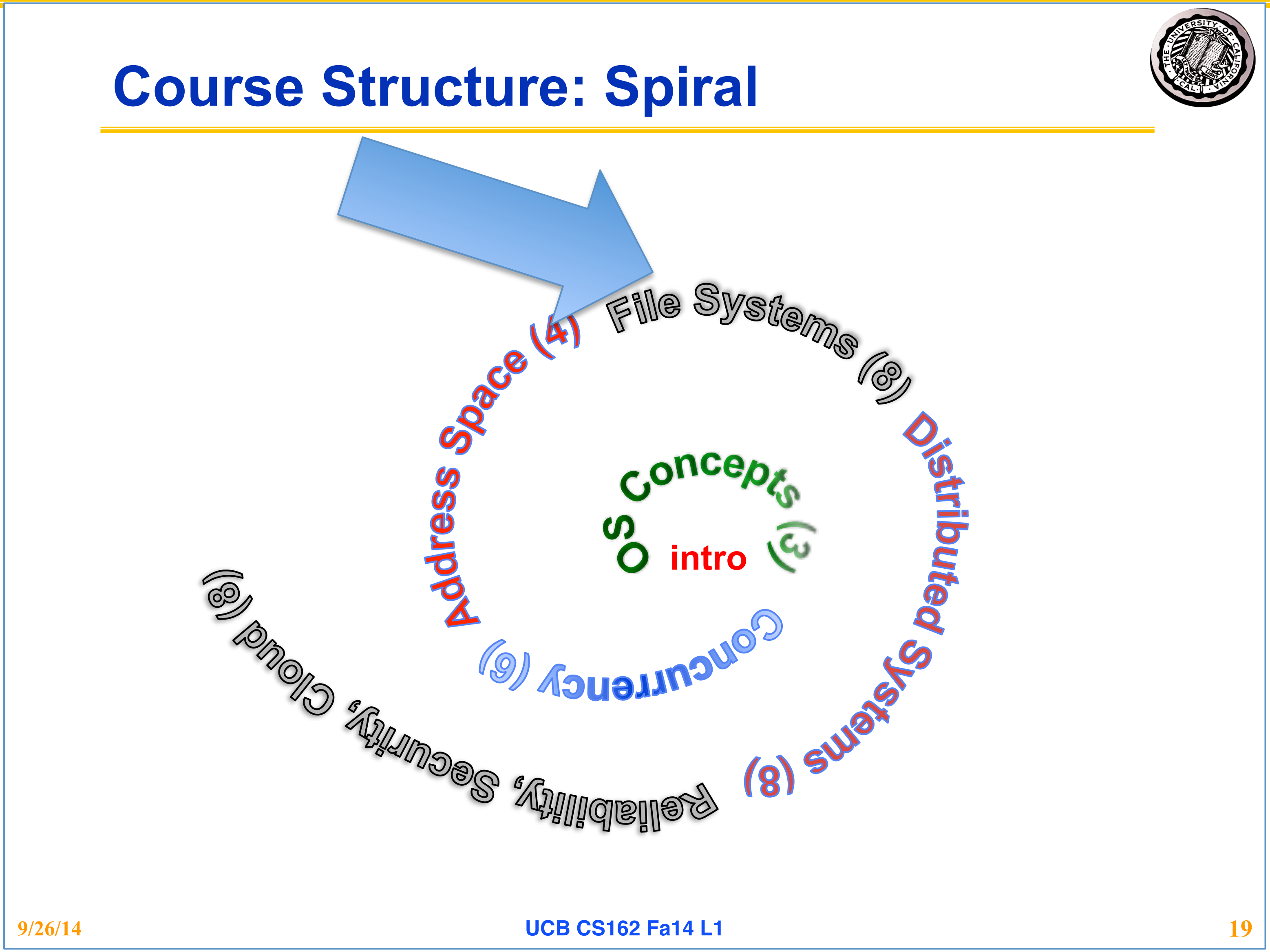
- Mainframes and minicomputers (servers) were "always paging"
  - memory was limited
  - processor rates <> disk xfer rates were much closer
- When overloaded would THRASH
  - with good OS design still made progress
- Modern systems hardly every page
  - primarily a safety net + lots of untouched "stuff"
  - plus all the other advantages of managing a VAS
- Effective use of the entire storage hierarchy is absolutely essential

Admin: Projects

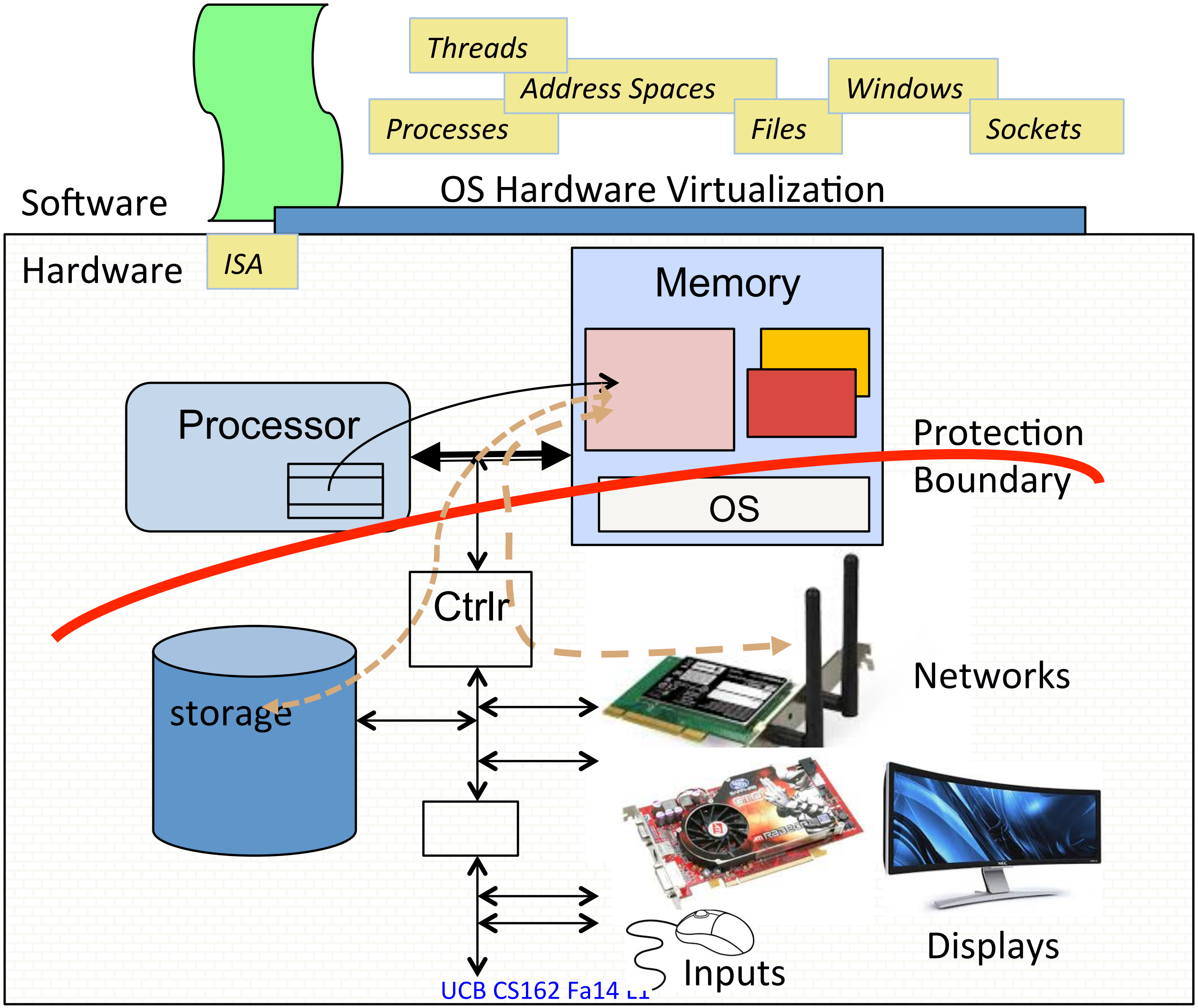
- Project 1
  - deep understanding of OS structure, interrupt, threads, thread implementation, synchronization, scheduling, and interactions of scheduling and synchronization
  - work effectively in a team
    - effective teams work together with a plan

    => schedule three 1-hour joint work times per week
- Project 2
  - exe load and VAS creation provided for you
  - syscall processing, FORK+EXEC, file descriptors backing user file handles, ARGV
    - registers & stack frames
  - two development threads for team
    - but still need to work together

You are here …


Course Structure: Spiral
OS Concepts (3)
intro
Concurrency (6)
Address Space (4)
File Systems (8)
Distributed Systems (8)
Reliability, Security, Cloud (8)
9/26/14
UCB CS162 Fa14 L1
19

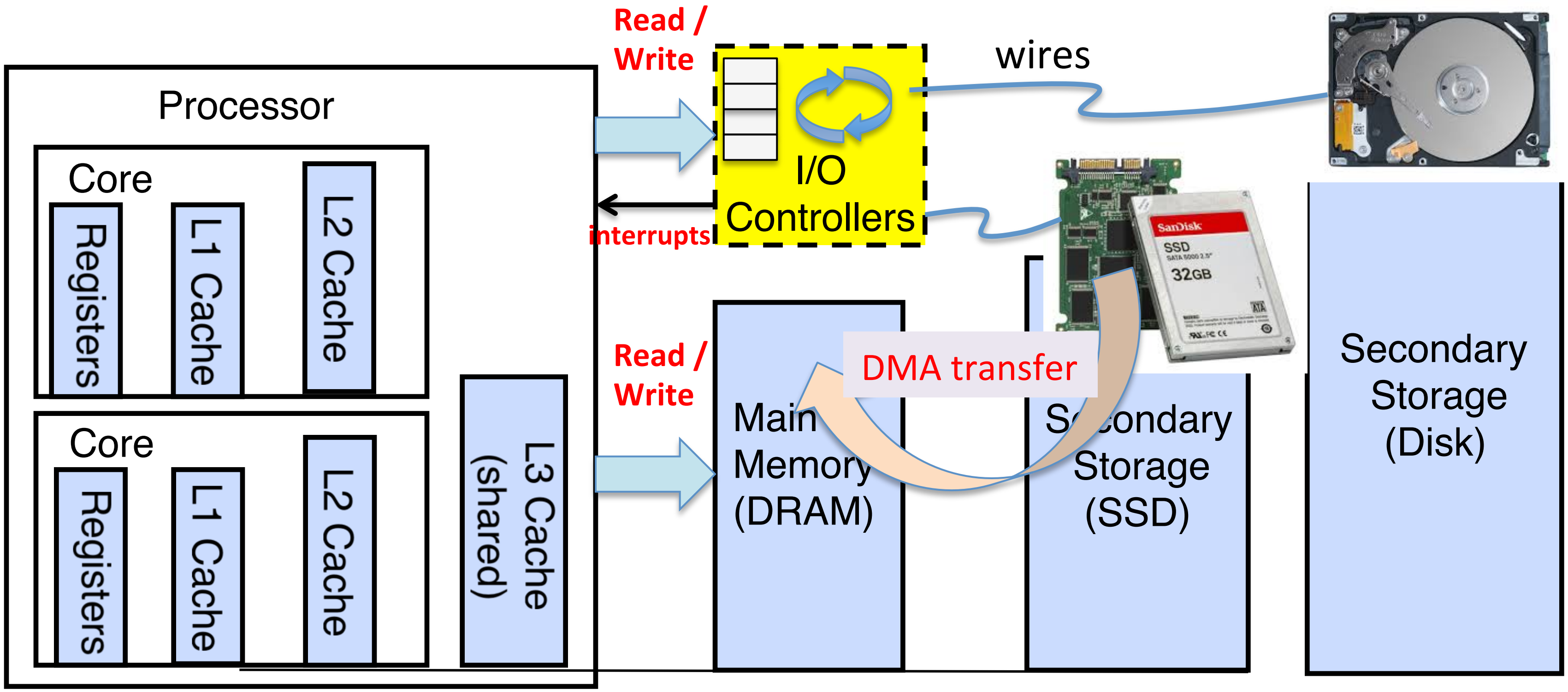
OS Basics: I/O


Threads
Address Spaces
Windows
Processes
Files
Sockets
Software
OS Hardware Virtualization
Hardware
ISA
Memory
Processor
Protection
Boundary
OS
Ctrlr
storage
Networks
Displays
Inputs

The Question of the Day

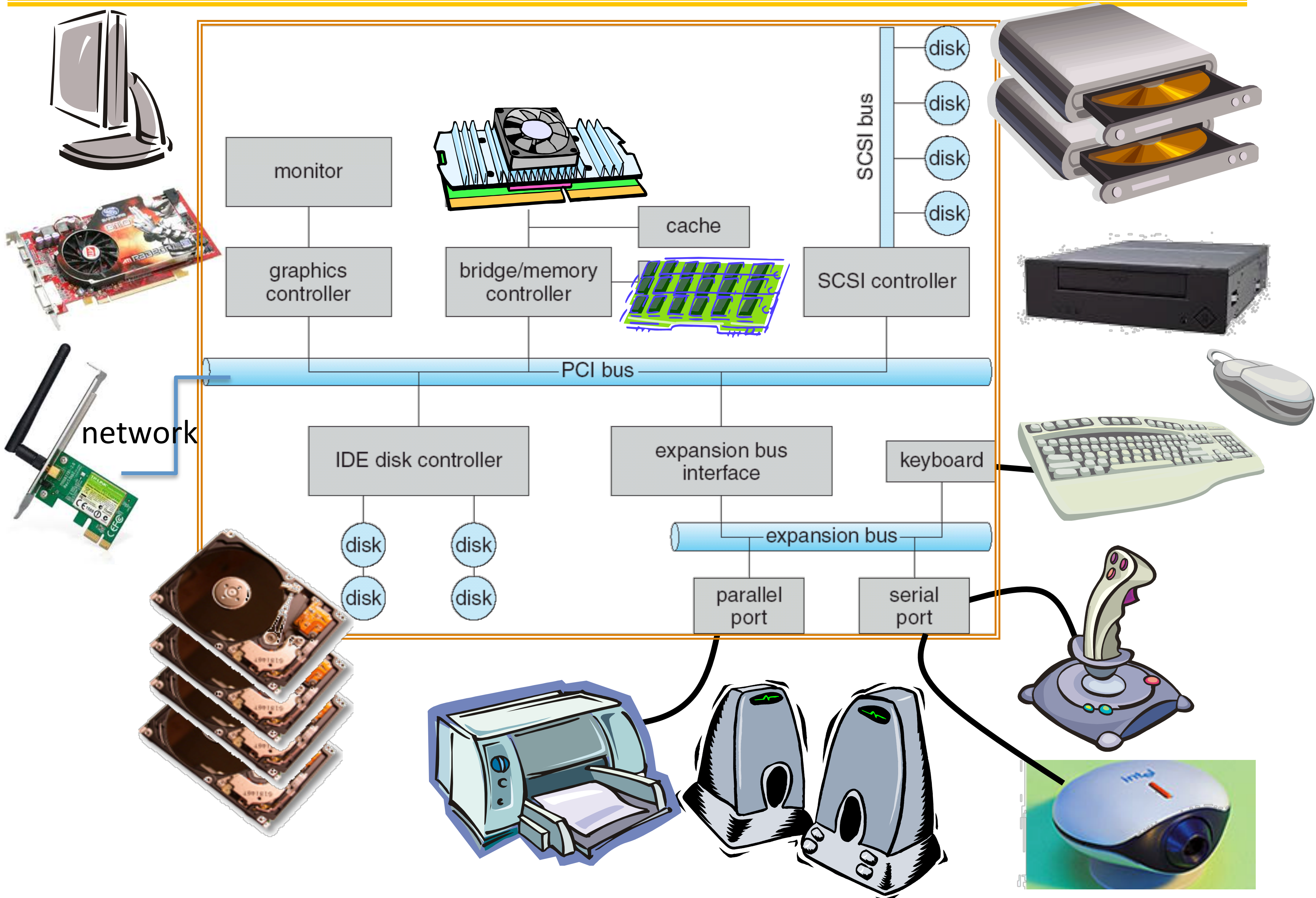
- The OS provides convenient, protected, high-level abstractions of shared physical resources
  - Processors => Threads
  - Memory => Address Space
  - Disk Blocks => Files
  - Network Packets => Messages
  - Keyboard, Mouse, Display => Windows
- So, how does it access the hardware to build these?

In a picture

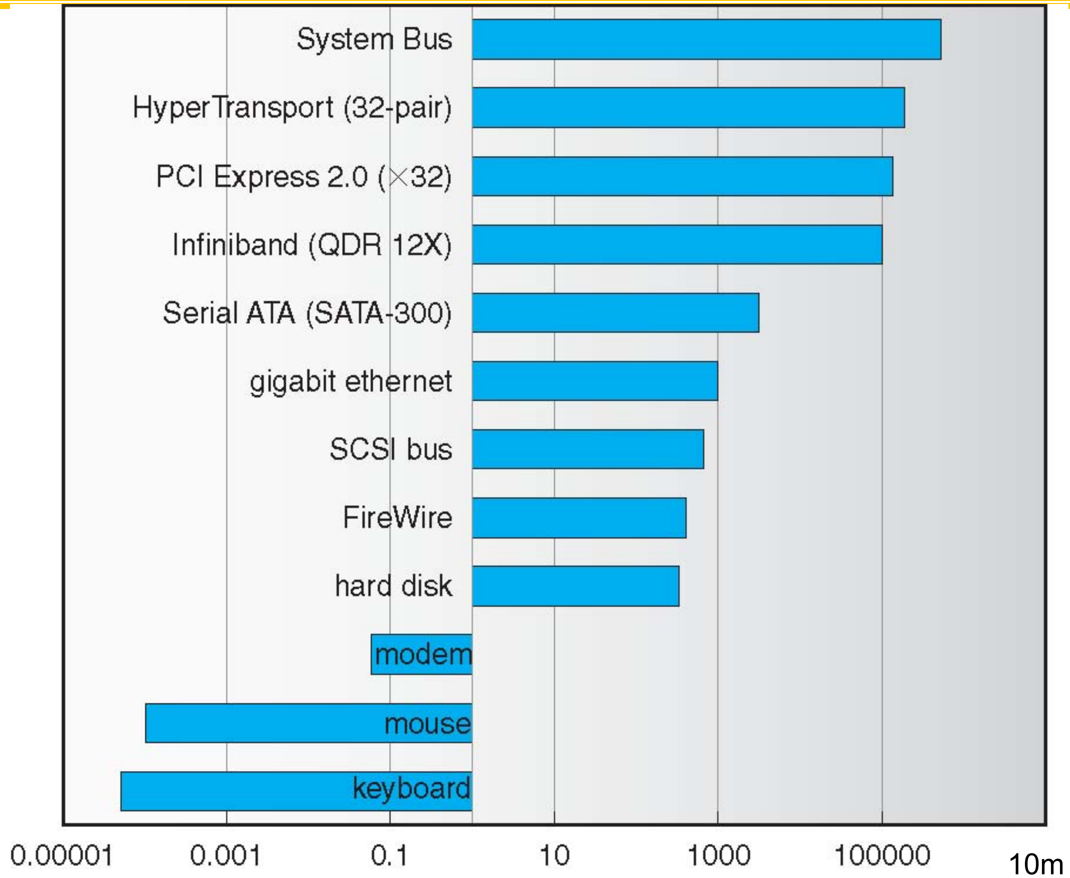


- I/O devices you recognize are supported by I/O Controllers
- Processors accesses them by reading and writing IO registers as if they were memory
  - Write commands and arguments, read status and results

Modern I/O Systems

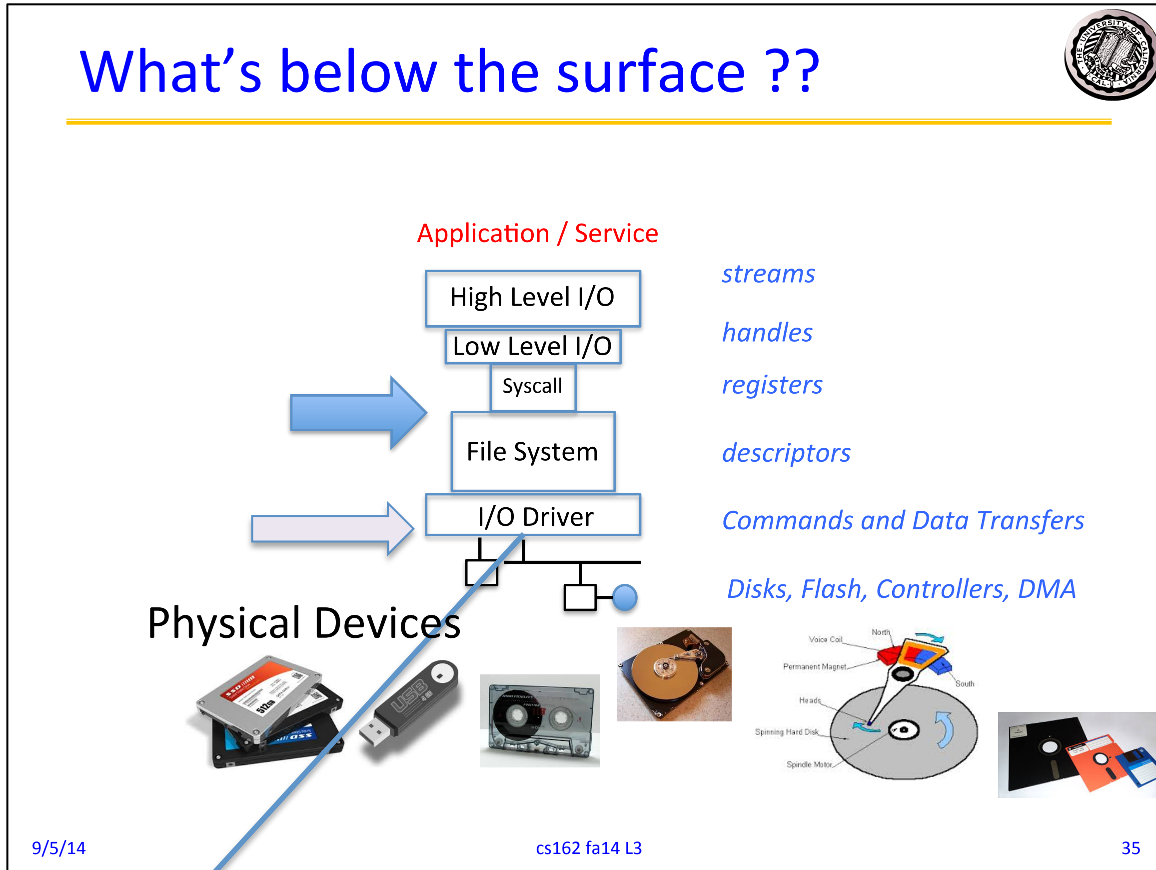


Example Device-Transfer Rates in Mb/s (Sun Enterprise 6000)



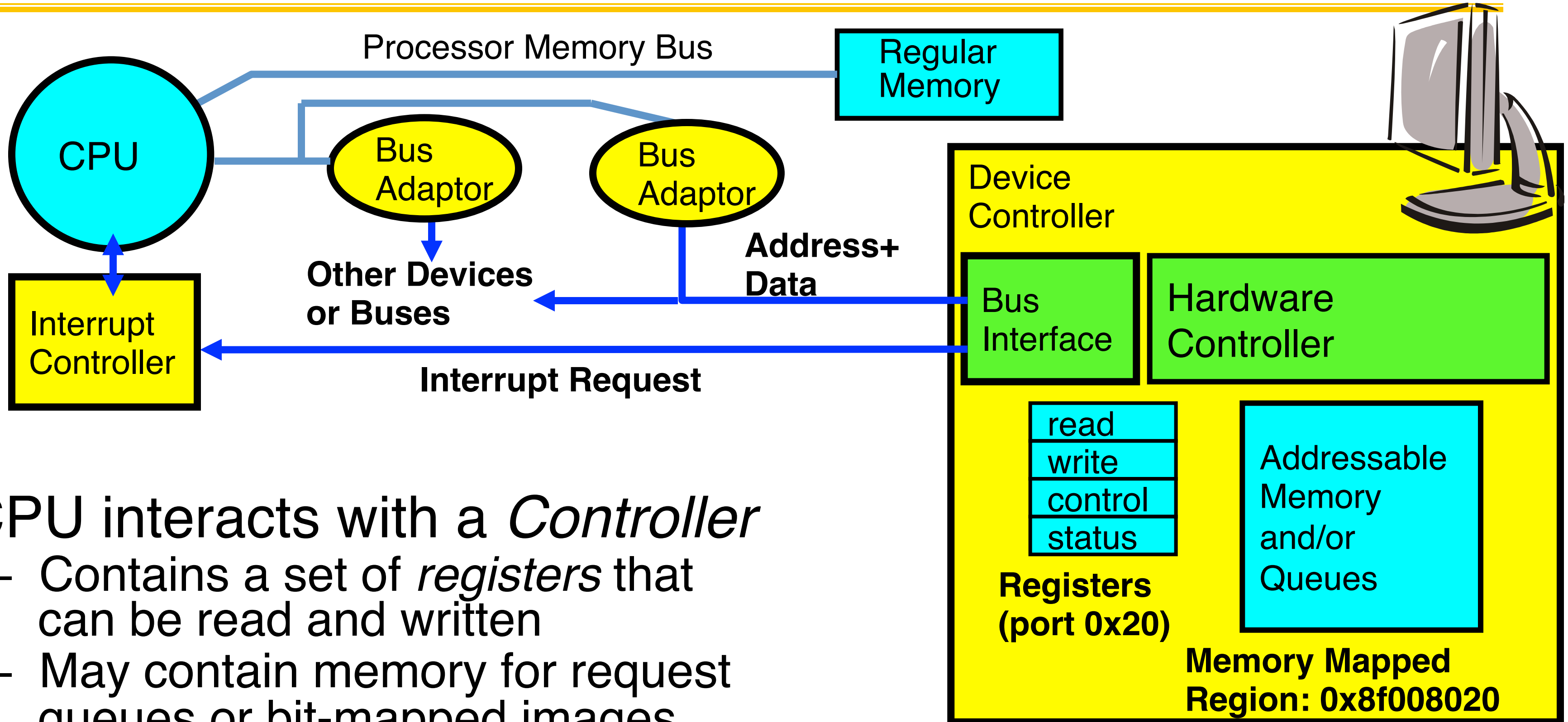
- Device Rates vary over 12 orders of magnitude !!!
  - System better be able to handle this wide range
  - Better not have high overhead/byte for fast devices!
  - Better not waste time waiting for slow devices

What does it mean for OS?



Memory mapped I/O
Direct Memory Access
Interrupts

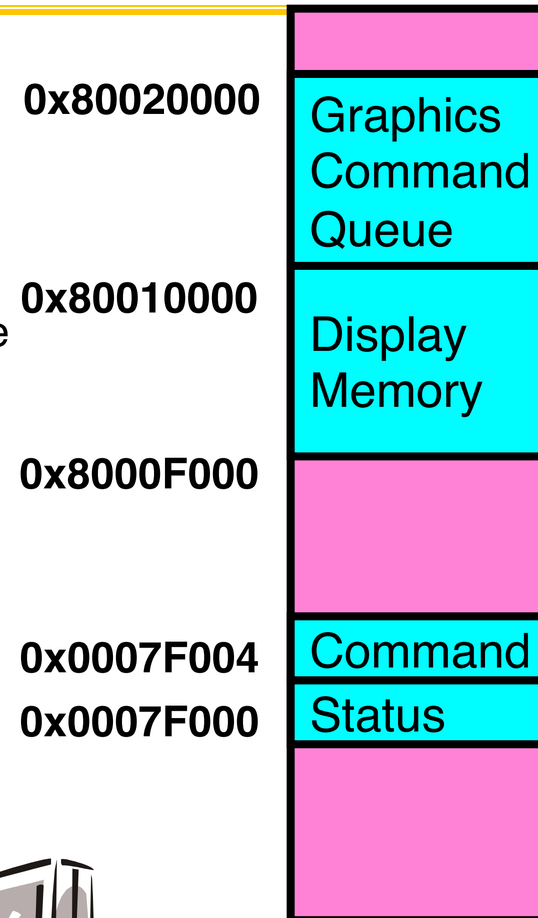
How Does the Processor Talk to Devices?



- CPU interacts with a *Controller*
  - Contains a set of *registers* that can be read and written
  - May contain memory for request queues or bit-mapped images
- Regardless of the complexity of the connections and buses, processor accesses registers in two ways:
  - I/O instructions: in/out instructions (e.g., Intel’s 0x21,AL)
  - Memory mapped I/O: load/store instructions
    - Registers/memory appear in physical address space
    - I/O accomplished with load and store instructions

Example: Memory-Mapped Display Controller

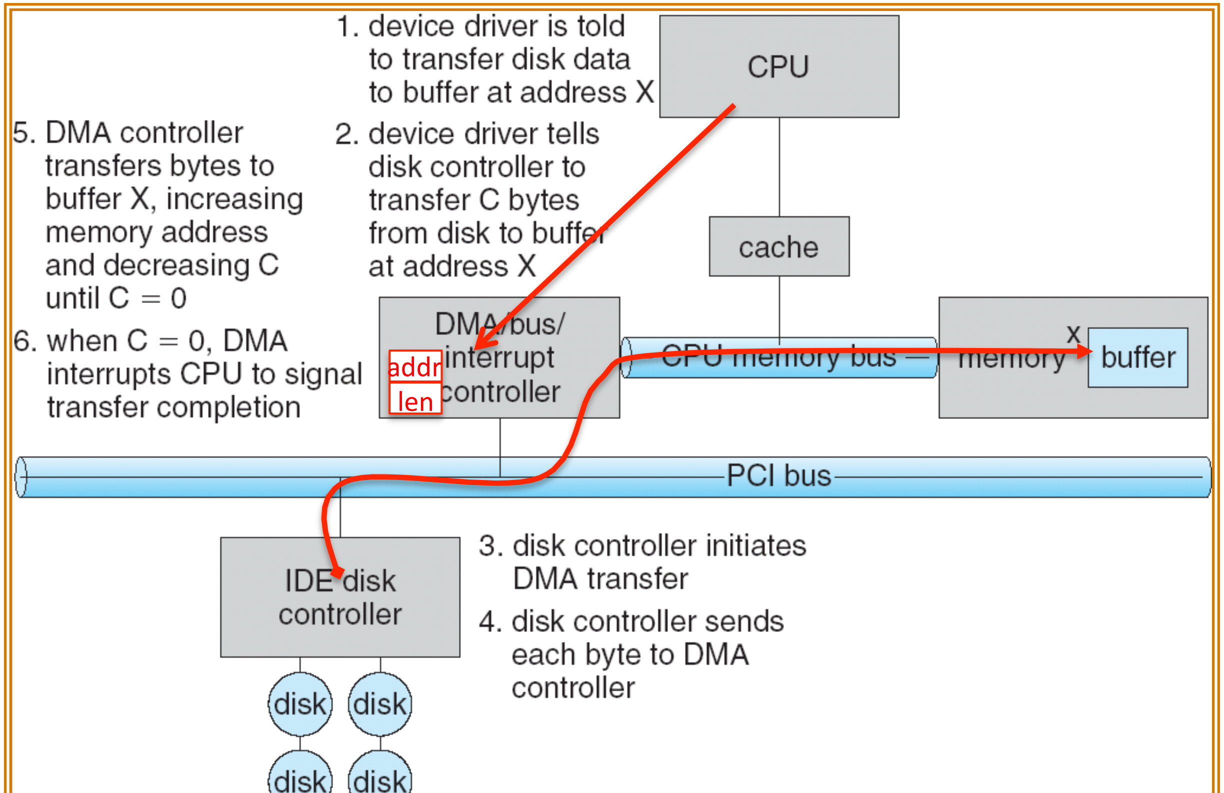
- Memory-Mapped:
  - Hardware maps control registers and display memory into physical address space
    - Addresses set by hardware jumpers or programming at boot time
  - Simply writing to display memory (also called the “frame buffer”) changes image on screen
    - Addr: 0x8000F000—0x8000FFFF
  - Writing graphics description to command-queue area
    - Say enter a set of triangles that describe some scene
    - Addr: 0x80010000—0x8001FFFF
  - Writing to the command register may cause on-board graphics hardware to do something
    - Say render the above scene
    - Addr: 0x0007F004
- Can protect with address translation



Physical Address Space

Transferring Data To/From Controller

- Programmed I/O:
  - Each byte transferred via processor in/out or load/store
  - Pro: Simple hardware, easy to program
  - Con: Consumes processor cycles proportional to data size
- Direct Memory Access:
  - Give controller access to memory bus
  - Ask it to transfer data blocks to/from memory directly
- Sample interaction with DMA controller (from OSC):



I/O Device Notifying the OS

- The OS needs to know when:
  - The I/O device has completed an operation
  - The I/O operation has encountered an error
- I/O Interrupt:
  - Device generates an interrupt whenever it needs service
  - Pro: handles unpredictable events well
  - Con: interrupts relatively high overhead
- Polling:
  - OS periodically checks a device-specific status register
    - I/O device puts completion information in status register
  - Pro: low overhead
  - Con: may waste many cycles on polling if infrequent or unpredictable I/O operations
- Actual devices combine both polling and interrupts
  - For instance – High-bandwidth network adapter:
    - Interrupt for first incoming packet
    - Poll for following packets until hardware queues are empty

What is the Role of I/O?

- Without I/O, computers are useless (disembodied brains?)
- But… thousands of devices, each slightly different
  - How can we standardize the interfaces to these devices?
- Devices unreliable: media failures and transmission errors
  - How can we make them reliable???
- Devices unpredictable and/or slow
  - How can we manage them if we don't know what they will do or how they will perform?

If time – take apart a new machine

My New MacPro

- Hardware
  - ATA
  - Audio
  - Bluetooth
  - Camera
  - Card Reader
  - Diagnostics
  - Disc Burning
  - Ethernet Cards
  - Fibre Channel
  - FireWire
  - Graphics/Displays
  - Hardware RAID
  - Memory
  - PCI Cards
  - Parallel SCSI
  - Power
  - Printers
  - SAS
  - SATA/SATA Express
  - SPI
  - Storage
  - Thunderbolt
  - USB
- Network
  - Firewall
  - Locations
  - Volumes
  - WWAN
  - Wi-Fi

Hardware Overview:

Model Name:	MacBook Pro
Model Identifier:	MacBookPro11,2
Processor Name:	Intel Core i7
Processor Speed:	2 GHz
Number of Processors:	1
Total Number of Cores:	4
L2 Cache (per Core):	256 KB
L3 Cache:	6 MB
Memory:	16 GB
Boot ROM Version:	MBP112.0138.B07
SMC Version (system):	2.18f10
Serial Number (system):	C02MX0M3FD58
Hardware UUID:	63B1A15F-36A2-533

MacBook Pro

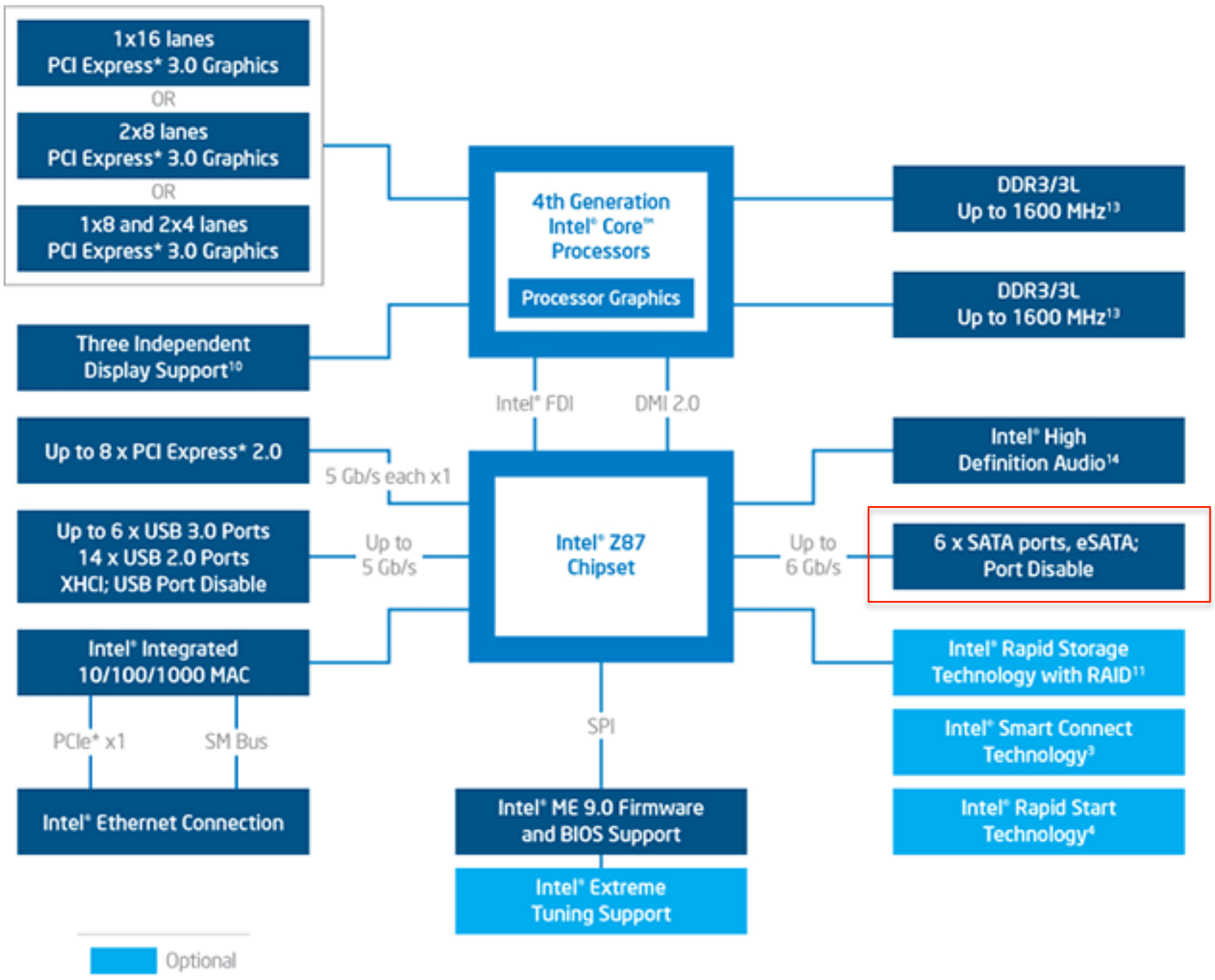
Retina, 15-inch, Late 2013

Processor 2 GHz Intel Core i7

Memory 16 GB 1600 MHz DDR3

Graphics Intel Iris Pro 1536 MB

Intel i7 Core …



My Disk

Apple SSD Controller:

Vendor: Apple
Product: SSD Controller
Physical Interconnect: PCI
Link Width: x2
Link Speed: 5.0 GT/s
Description: AHCI Version 1.30 Supported

APPLE SSD SM0256F:

Capacity: 251 GB (251,000,193,024 bytes)
Model: APPLE SSD SM0256F
Revision: UXM2JA1Q
Serial Number: S1K4NYAF592211
Native Command Queuing: Yes
Queue Depth: 32
Removable Media: No
Detachable Drive: No
BSD Name: disk0
Medium Type: Solid State
TRIM Support: Yes
Partition Map Type: GPT (GUID Partition Table)
S.M.A.R.T. status: Verified
Volumes:

EFI:

Capacity: 209.7 MB (209,715,200 bytes)
BSD Name: disk0s1
Content: EFI

Macintosh HD:

Capacity: 250.14 GB (250,140,434,432 bytes)
Available: 195.43 GB (195,428,974,592 bytes)
Writable: Yes
File System: Journaled HFS+
BSD Name: disk0s2
Mount Point: /
Content: Apple_HFS
Volume UUID: 90C81FF2-EED6-3FEE-BA72-294D2DBFB952

Recovery HD:

My Graphics

Intel Iris Pro:

Chipset Model:	Intel Iris Pro
Type:	GPU
Bus:	Built-In
VRAM (Dynamic, Max):	1536 MB
Vendor:	Intel (0x8086)
Device ID:	0x0d26
Revision ID:	0x0008
Displays:	

Color LCD:

Display Type:	Retina LCD
Resolution:	2880 x 1800
Retina:	Yes
Pixel Depth:	32-Bit Color (ARGB8888)
Main Display:	Yes
Mirror:	Off
Online:	Yes
Built-In:	Yes

- Haswell introduces configurations with large graphics & an on-package eDRAM cache
- Cache attributes
  - High throughput and low latency
  - Flat power vs. sustained bandwidth curve
  - Fully shared between Graphics, Media, and Cores
- Low latency on package interface to CPU