NC STATE UNIVERSITY
Department of Computer Science

CSC501
Operating Systems Principles

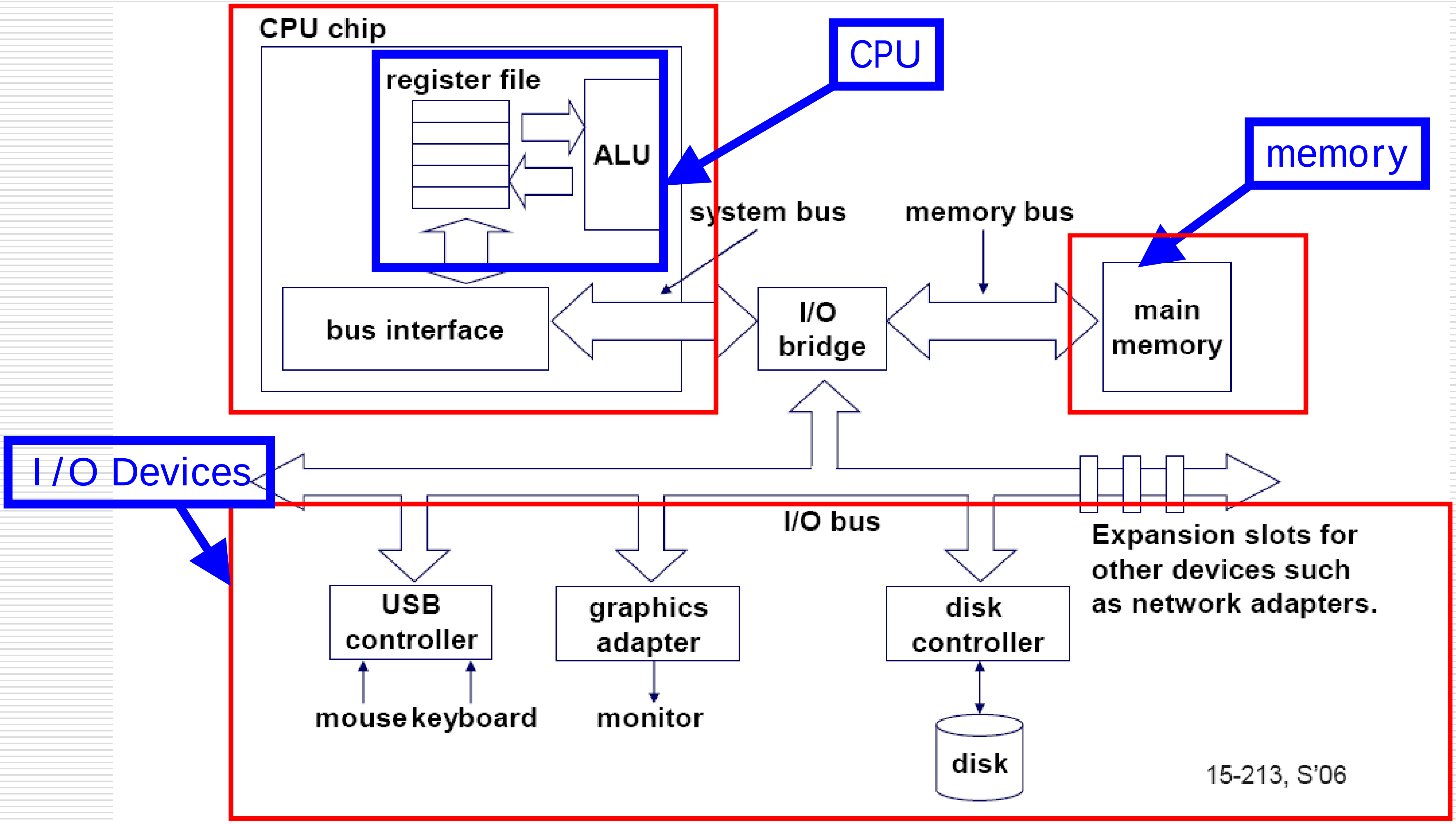
I/O Devices

Previous Lectures

- Abstraction
  - Process
  - Memory

- Now
  - I/O Devices

Basic Computer Architecture



Intel SR440BX Motherboard

CPU

System Bus & MMU/AGP/PCI Controller

I/O Bus

IDE Disk Controller

USB Controller

Serial & Parallel Ports

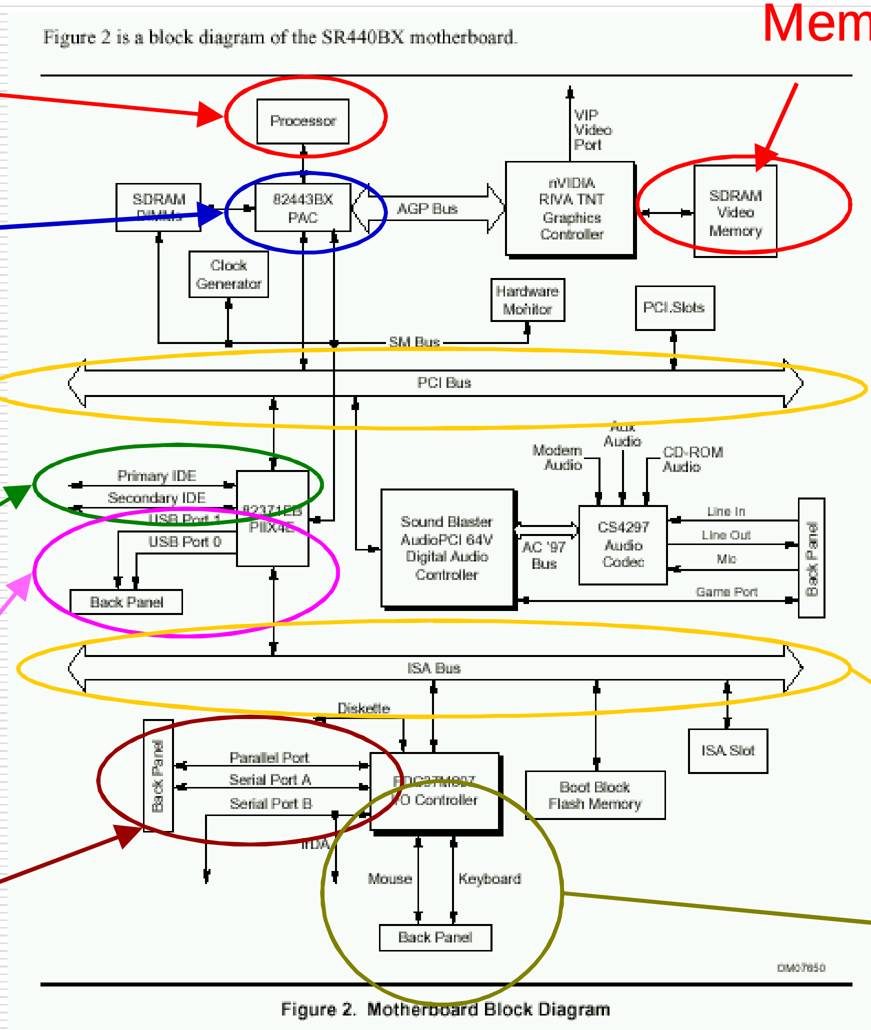


Figure 2. Motherboard Block Diagram

Memory

Another I/O Bus

Keyboard & Mouse

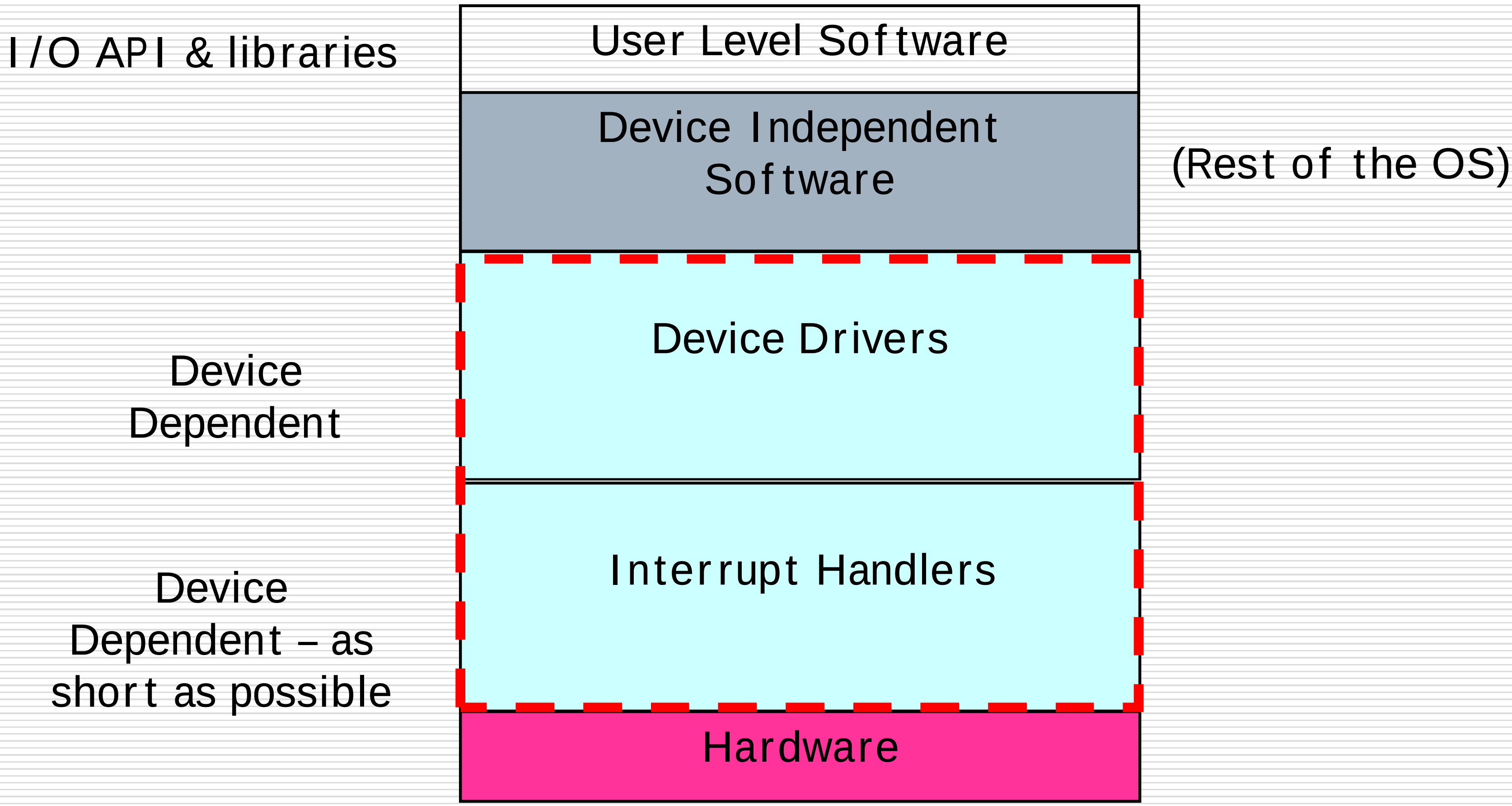
OS Designer's Perspective

- The largest, most complex subsystem in OS
- Most lines of code with the highest rate of code changes
  - Where OS engineers most likely to work
  - Difficult to test thoroughly
  - Big impact on performance and perception
  - Bigger impact on acceptability in market

Principles of I/O Software

- **Efficiency** - Do not allow I/O operations to become system bottleneck, especially for slow devices
- **Generality** - Provide a common API to access I/O devices
  - Uniform naming/interface
    - open, close read, write…
  - Device independence
    - isolate OS and application programs from device specific details and peculiarities
    - isolate the impact of device errors, retry where possible
      - Errors are abundant in I/O

I/O Software "Stack"



Communication with I/O Devices

- Programmed I/O (a.k.a., polling)
- Interrupt-Driven I/O
- DMA-based I/O

Programmed I/O (Polling)

- Used when device and controller are relatively quick to process an I/O operation
  - Device driver
    - Gains access to device
    - Initiates I/O operation
    - Loops testing for completion of I/O operation
    - If there are more I/O operations, repeat
  - Used in following kinds of cases
    - Service interrupt time > Device response time
    - Device has no interrupt capability
    - Embedded systems where CPU has nothing else to do

Any Example?

Interrupt-Driven I/O

- Interrupts occur on I/O events
  - Operation completion
  - Error or change of status
- Interrupt handling
  - Step 1: Save the current machine state
  - Step 2: Load the machine state for interrupt handling
  - Step 3: Invoke the corresponding ISR
  - Step 4: Resume the program execution

Any Example?

Question:

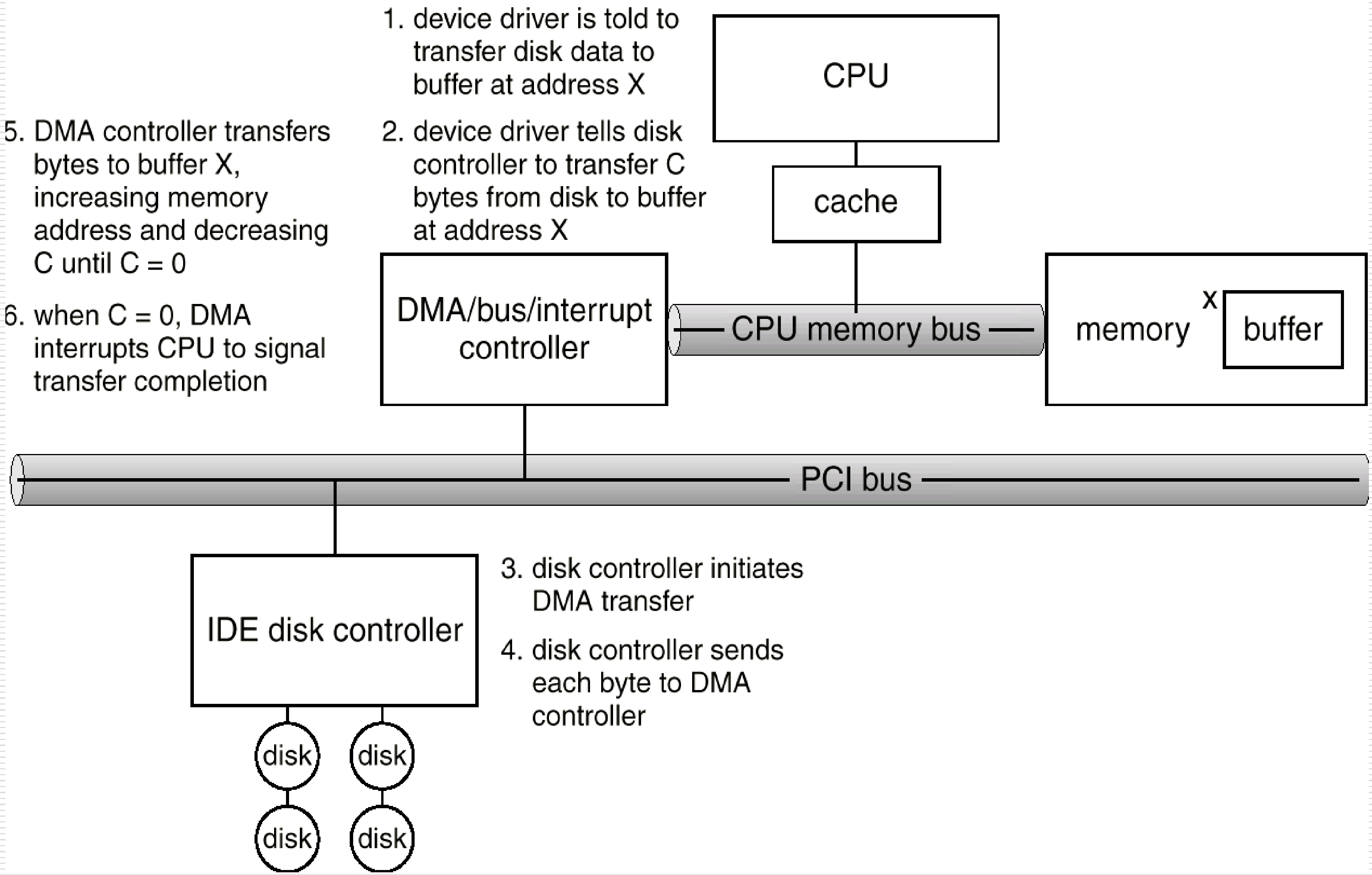
Where do we save the current machine states?

DMA-Based I/O

- Direct Memory Access
  - Device read/write directly from/to memory
  - Transfer from memory to device typically initiated from CPU
  - Transfer from device to memory can be initiated by the device or the CPU

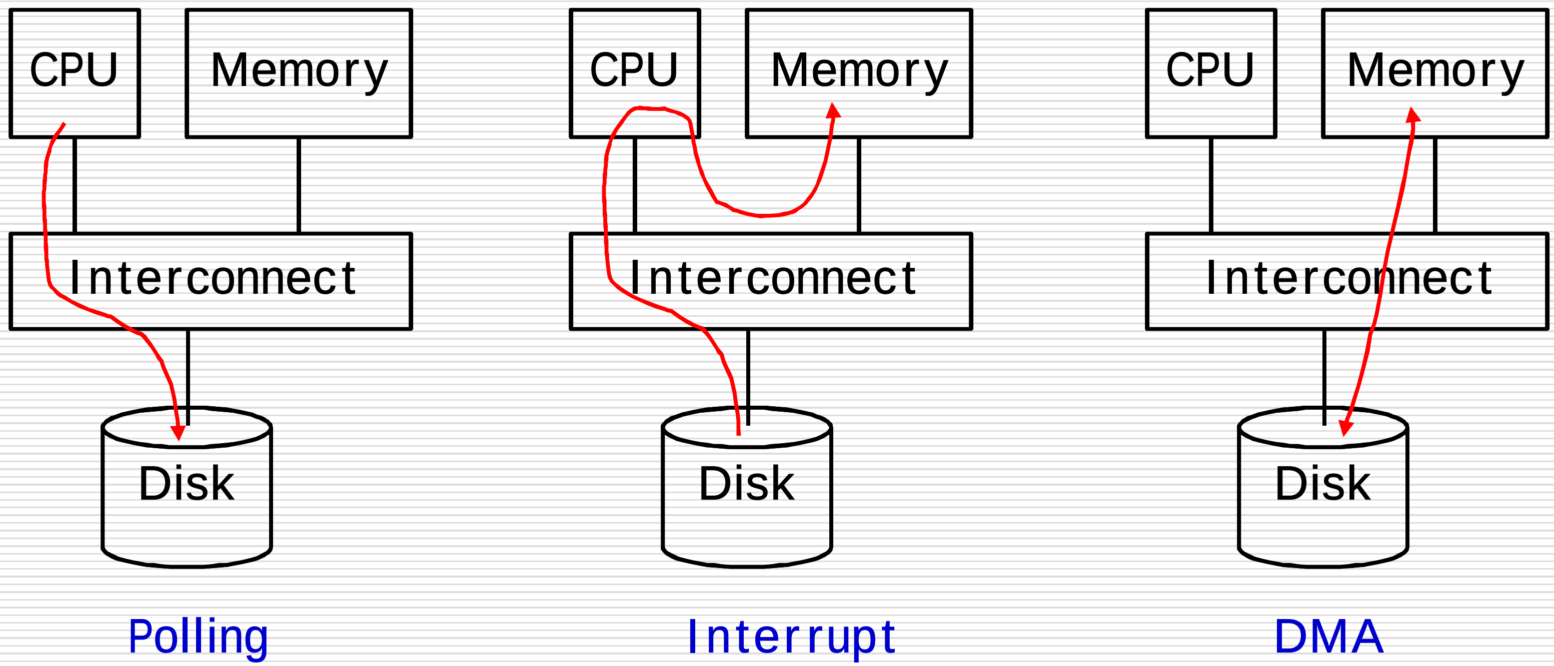
Any Example?

DMA-Based I/O


1. device driver is told to transfer disk data to buffer at address X
2. device driver tells disk controller to transfer C bytes from disk to buffer at address X
3. disk controller initiates DMA transfer
4. disk controller sends each byte to DMA controller
5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0
6. when C = 0, DMA interrupts CPU to signal transfer completion
CPU
cache
CPU memory bus
memory
X
buffer
DMA/bus/interrupt controller
PCI bus
IDE disk controller
disk
disk
disk
disk

Communication with I/O Devices

- Comparison: Polling, Interrupt, and DMA
  - Polling and Interrupt = word at a time
  - DMA = block at a time



Computer Science NC STATE UNIVERSITY

Question:

How should the device driver be designed?

Device Driver

- Hides the device specifics from the above layers
  - Supporting a common API
- Translates logical I/O into device I/O
  - E.g., logical disk blocks into {head, track, sector}
  - Performs data buffering and scheduling of I/O operations
- Structure
  - Several synchronous entry points: device initialization, queue I/O requests, state control, read/write
  - An asynchronous entry point to handle interrupts

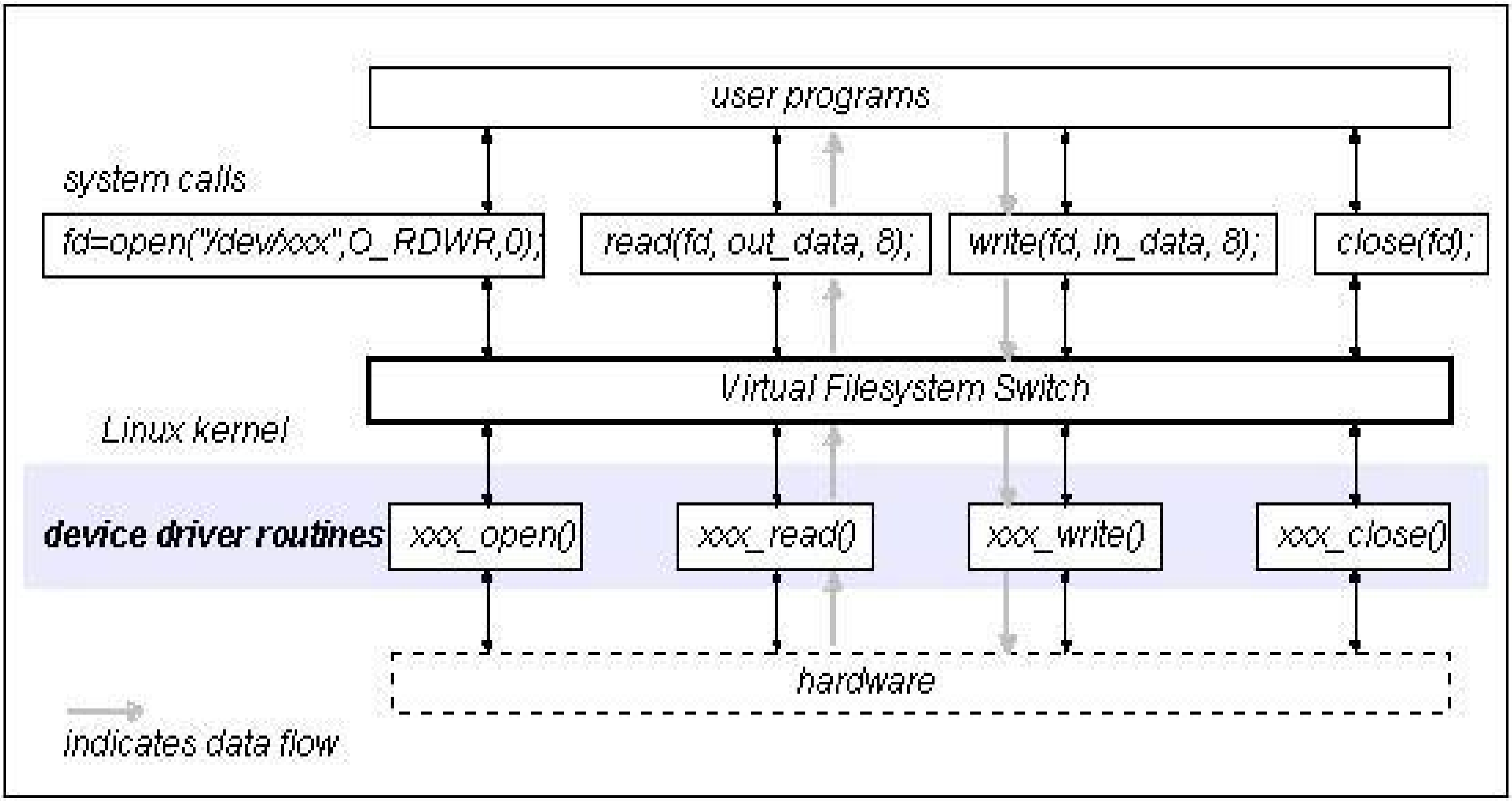
Device Driver – A Typical Design

- OS and driver communication: OS-dependent
  - Commands and data between OS and drivers
    - Meet the interface specs of the device independent layer
    - Utilize the facilities supplied by the OS – buffers, error codes, etc.
- Driver and hardware comm.: Device-dependent
  - Commands and data between driver and hardware
    - Actions during interrupt handling

Device Driver – A Typical Design

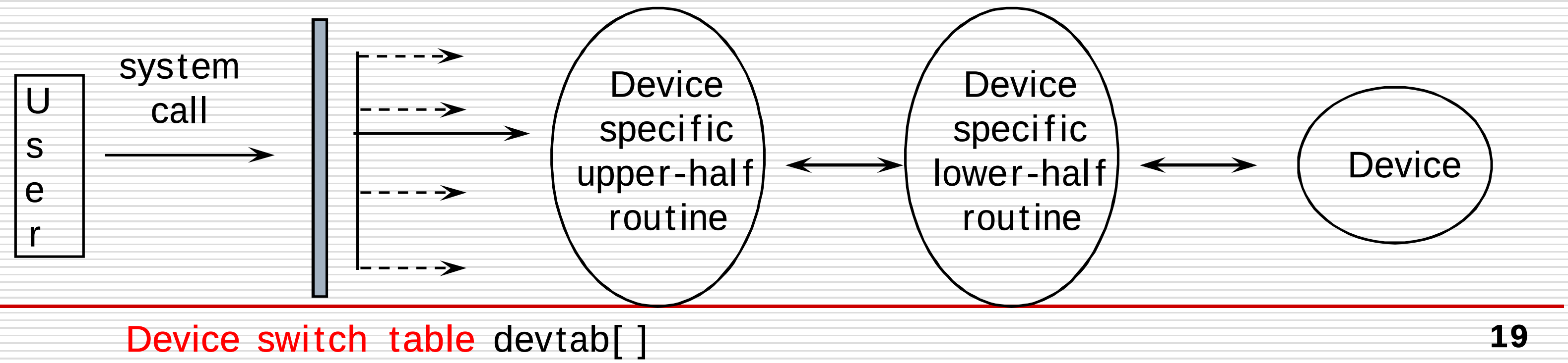
- Driver operations
  - Initialize devices
  - Interpret commands from OS
  - Schedule multiple outstanding requests
  - Manage data transfer
  - Accept and process interrupts
  - …
- Typically use open-close-read-write paradigm
  - Support *open, close, read, write, seek* functions
  - Use *ioctl* function as escape mechanism

Uniform API (e.g., in Linux)



Case Study w/ Xinu

- Code divided into two parts
  - Upper Half
    - Defines abstract operations
    - Executed by application to request I/O
  - Lower Half
    - Device-specific interrupt routine
    - Executed by arbitrary (current) process
    - Invoked when operation completes



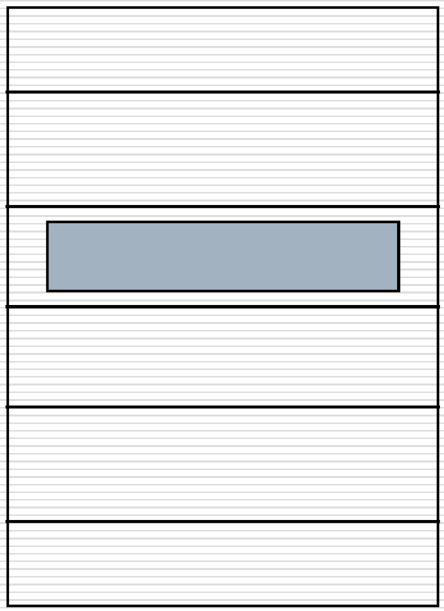
Case Study w/ Xinu

- Device switch table
  - One row per device
  - One column per operation
  - Entry points to a procedure

device ↓

operation →

	open	read	write	. . .
CONSOLE	&ttyopen	&ttyread	&ttywrite	
SERIAL0	&ionull	&comread	&comwrite	
SERIAL1	&ionull	&comread	&comwrite	
ETHER	&ethopen	&ethread	&ethwrite	
⋮				



Device Switch Table

```
int         dvnum;
long int    (*dvinit)();
long int    (*dvshutdown)();
long int    (*dvopen)();
long int    (*dvclose)();
long int    (*dvread)();
long int    (*dvwrite)();
long int    (*dvseek)();
long int    (*dvgetc)();
long int    (*dvputc)();
long int    (*dvcntl)();
long int    (*dviint)();
long int    (*dvoint)();
long int    dvcsr;
long int    dvivec;
long int    dvovec;
char        *dvioblk;
int         dvminor;
```

Ptrs to device specific high-level routines

Ptr to device control block

device switch table entry
(struct devsw)

The structure of the control block varies from device to device. The control block contains information associated with the device.

Case Study w/ Xinu

- Device switch table
  - Device-independent operations "generic"
  - Given operation may not apply to given device
    - Example: *seek on keyboard or* network
  - All entries in device switch table must be valid

Any solution?

Case Study w/ Xinu

- Device switch table
  - Device-independent operations “generic”
  - Given operation may not apply to given device
    - Example: *seek on keyboard or* network
  - All entries in device switch table must be valid
- Special Entries
  - *ionull*
    - Used for innocuous operation, always returns *OK*
  - *ioerr*
    - Used for incorrect operation, always returns *SYSERR*

Example Device Driver *tty* in Xinu

- Used for I/O to console
- Separate input and output buffers
- Use semaphores to synchronize

- Upper-half routines
  - ttyinit, ttyopen, ttyclose, ttyread, ttywrite, ttyputc, ttygetc, ttypcntl
- Lower-harf routines
  - ttyiin (input interrupt handler)
  - ttyoin (output interrupt handler)

Example Device Driver *tty* in Xinu

- Actions during a character output
  - Upper-half
    - Waits for buffer space
    - Deposits character
    - Triggers device
  - Lower-half
    - Extracts character and pass to device
    - Signals space semaphore

Next Lecture

- Hard Disk
  - Scheduling and RAID