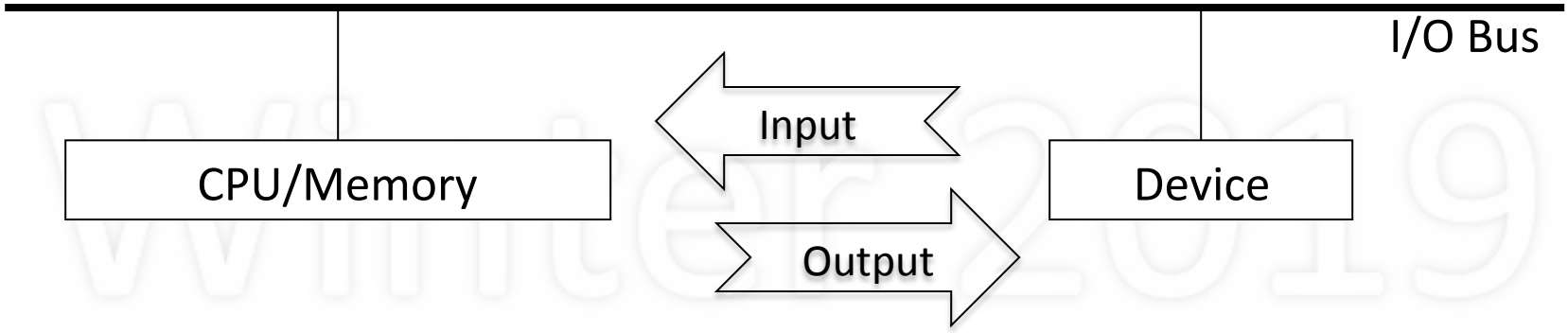
CSE 120: Principles of Operating Systems
Lecture 12: Input/Output System

Prof. Joseph Pasquale
University of California, San Diego

Feb 27, 2019

What is I/O

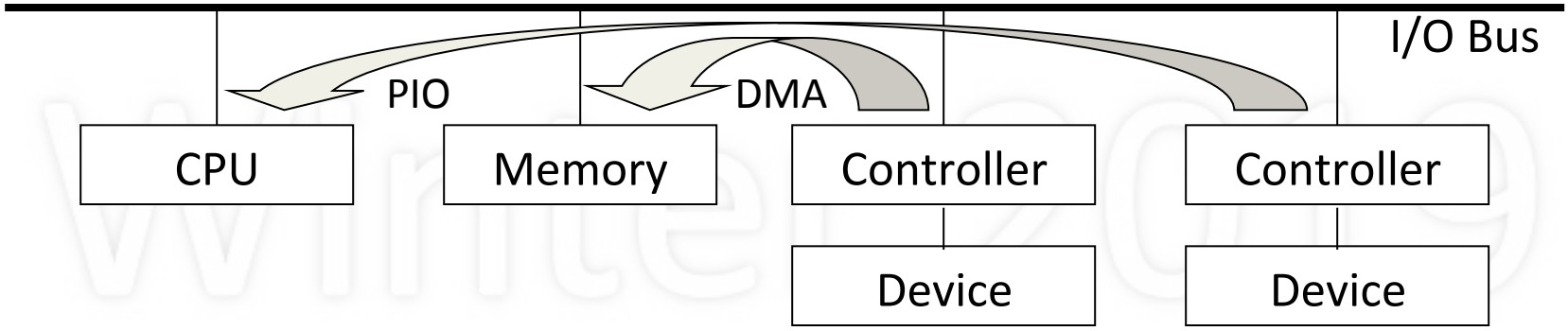


- I/O = Input/Output
  - Input from attached device to CPU/memory
  - Output from CPU/memory to device
- Synchronization and transferring data

The I/O System: Issues

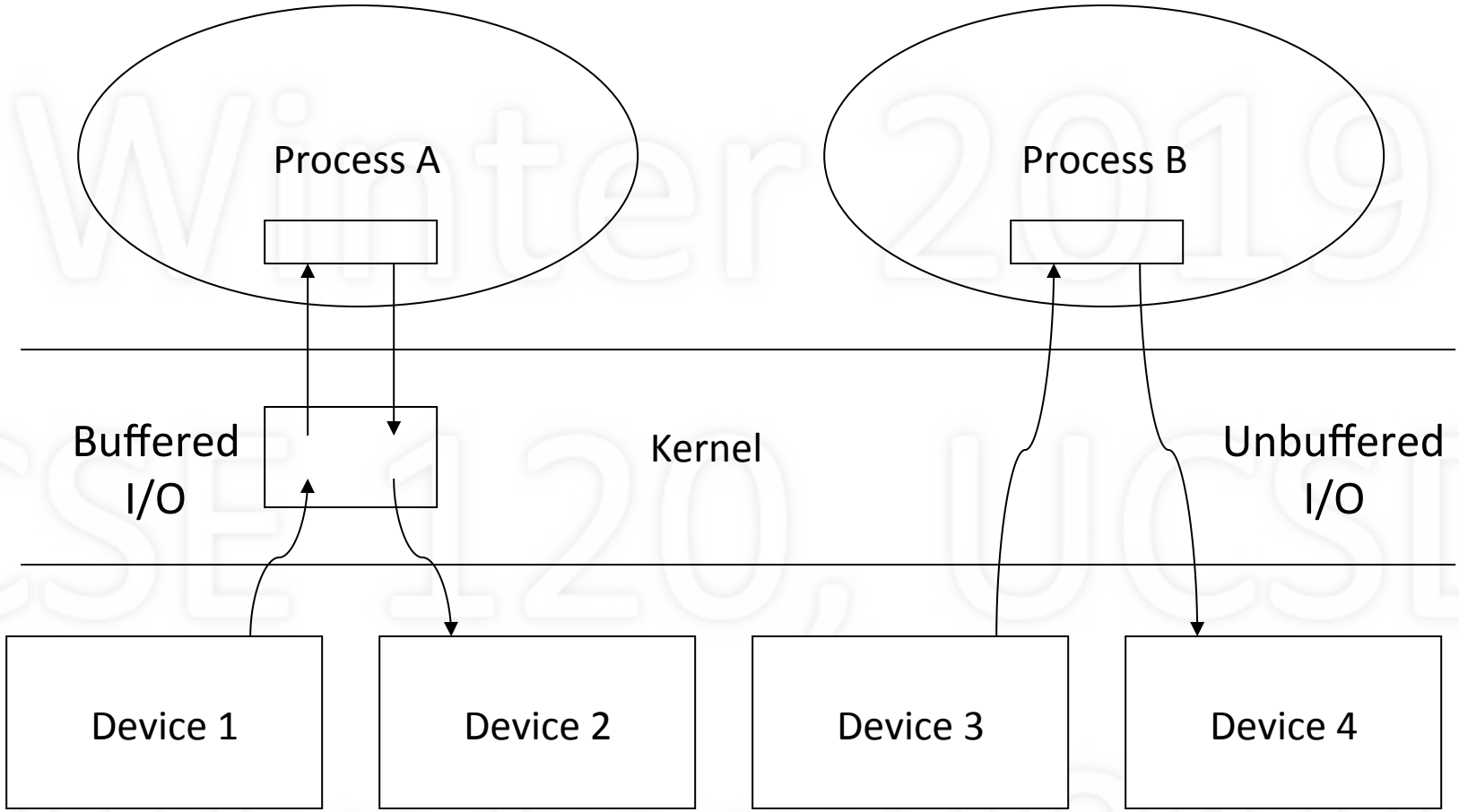
- Problems
  - So many different types of I/O devices
  - Wide range: speed, operation, data transfer units
- Questions
  - How does a process initiate I/O?
  - How is synchronization achieved?
  - How is data transferred?

Background: I/O Hardware

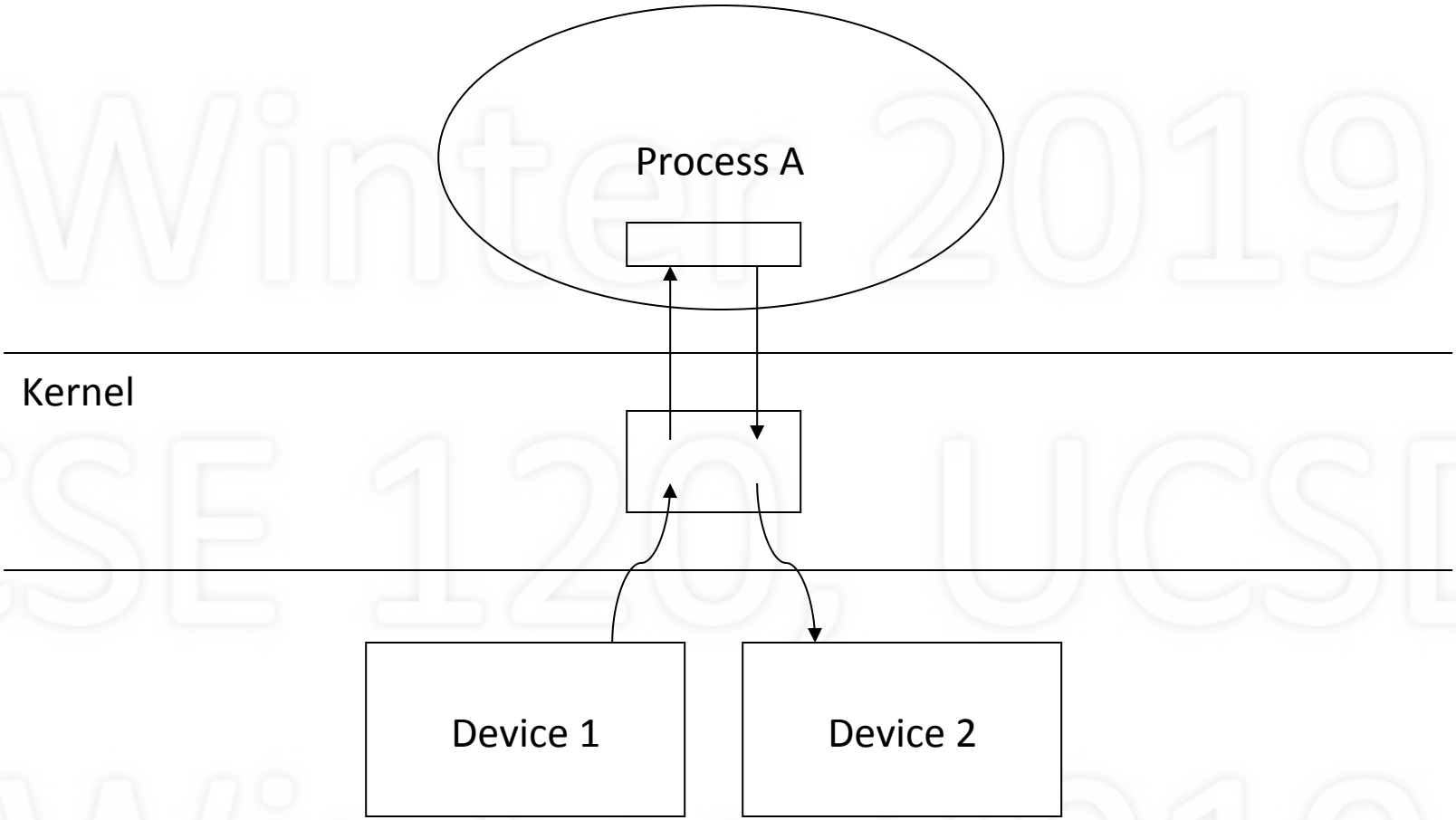


- CPU and device (controller) communicate via
  - I/O instructions
  - Memory instructions (memory-mapped)
- Data transfer: programmed I/O vs. DMA
- Synchronization: polling vs. interrupts

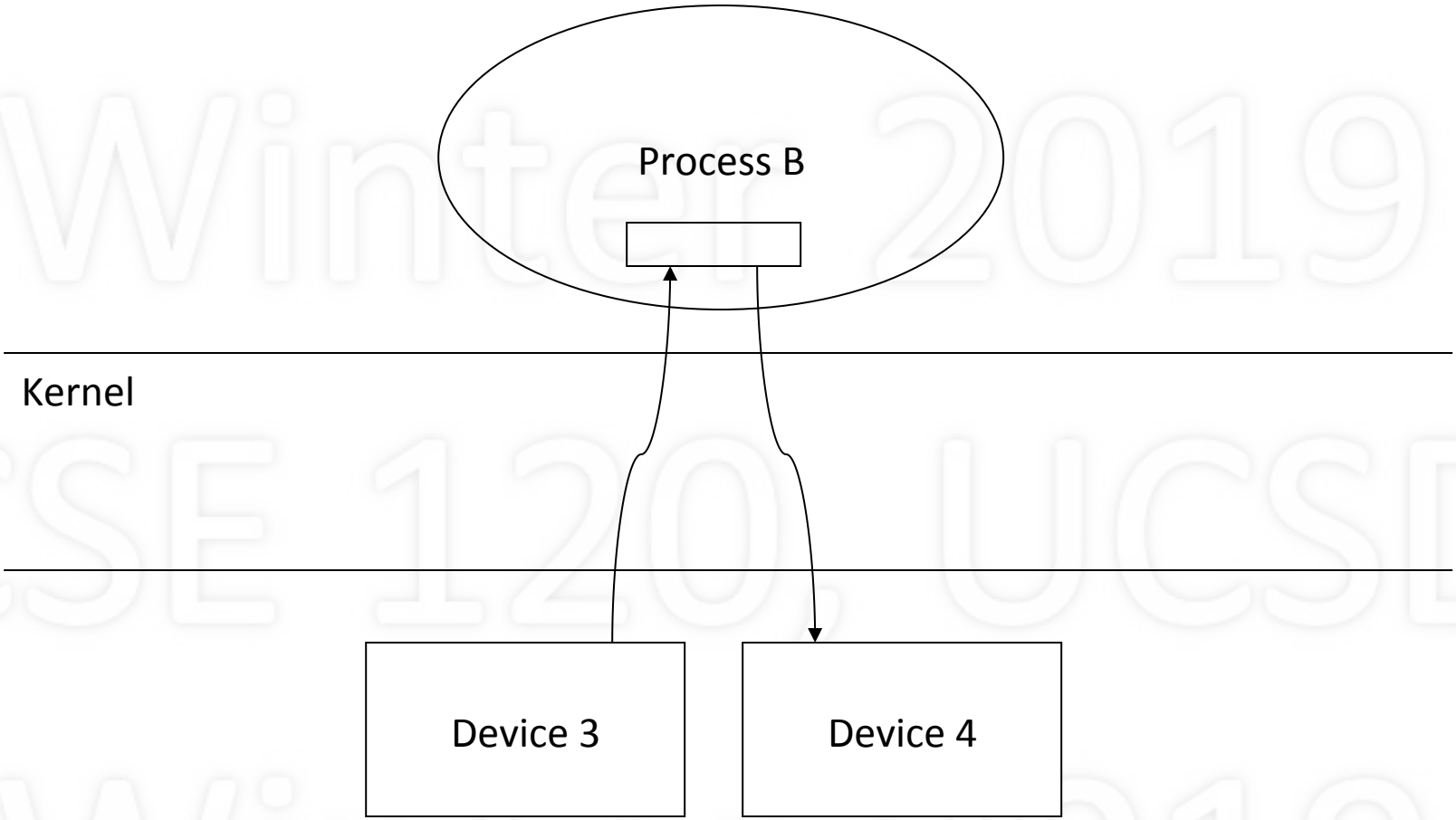
Buffered and Unbuffered I/O



Buffered I/O



Unbuffered I/O



Pros and Cons of Kernel Buffering

- Pros
  - What if pages containing buffer are paged out?
  - What if entire process is swapped out?
  - Can pin pages, but if too many processes do this …
- Cons
  - Memory copying is expensive!
  - Consider effect on caches

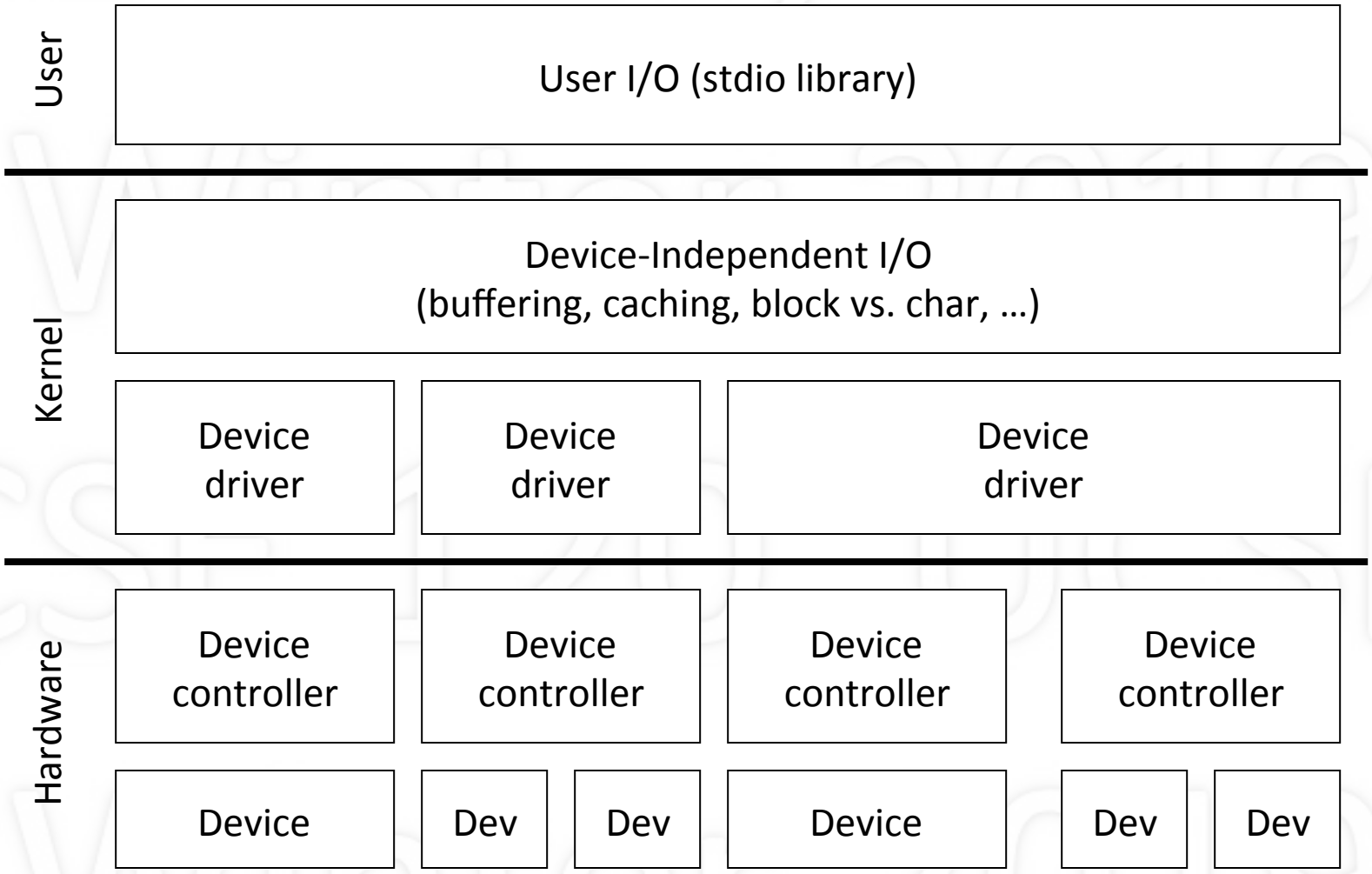
Dealing with Complexity of Devices

- Many different types of devices
  - Classify by shared characteristics
  - Imposes structure: shared code, lower complexity
- Dimensions
  - Variable vs. fixed size units
  - Sequential vs. random-access
  - Synchronous vs. asynchronous
  - Speed of operation

I/O System

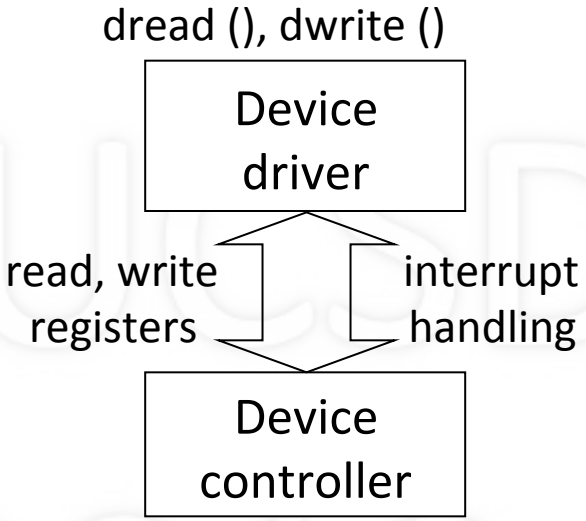
- Software that deals with I/O
  - Mostly in the kernel
  - Also in processes (in form of library, e.g., stdio)
- Separated into two portions
  - Device-dependent
  - Device-independent

I/O System Structure: Layered



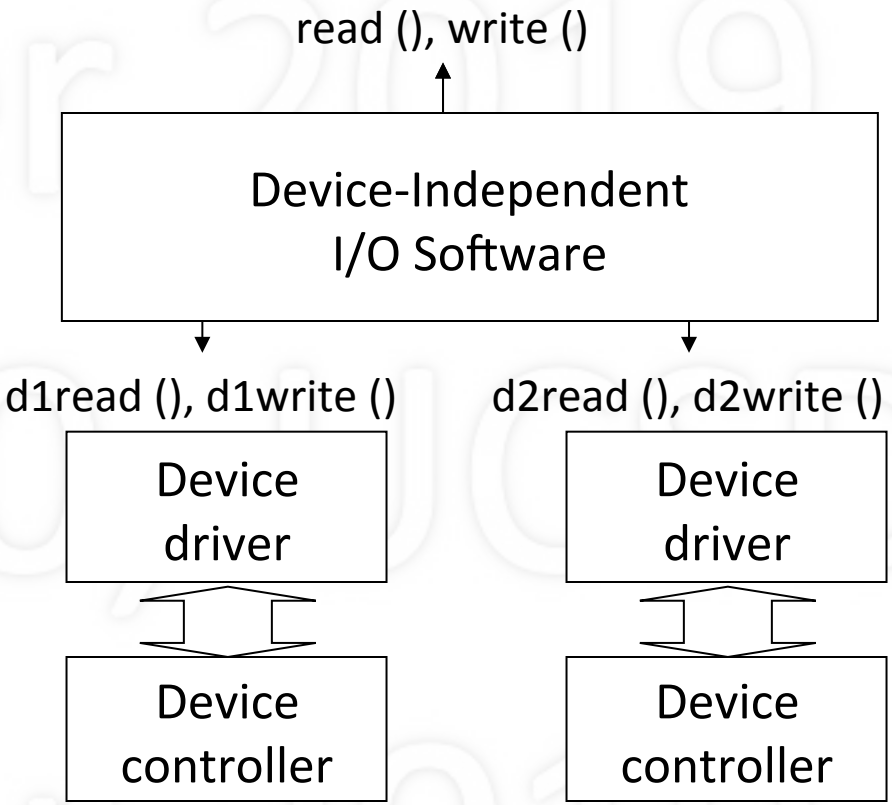
Device Dependent: Device Drivers

- Encapsulates device-dependent code
  - Contains device-specific register reads/writes
- Implements a standard interface
  - open (), close (), read (), write ()
- Interrupt handlers
  - Executes when I/O completes
  - Updates data structures
  - Wakes up waiting process



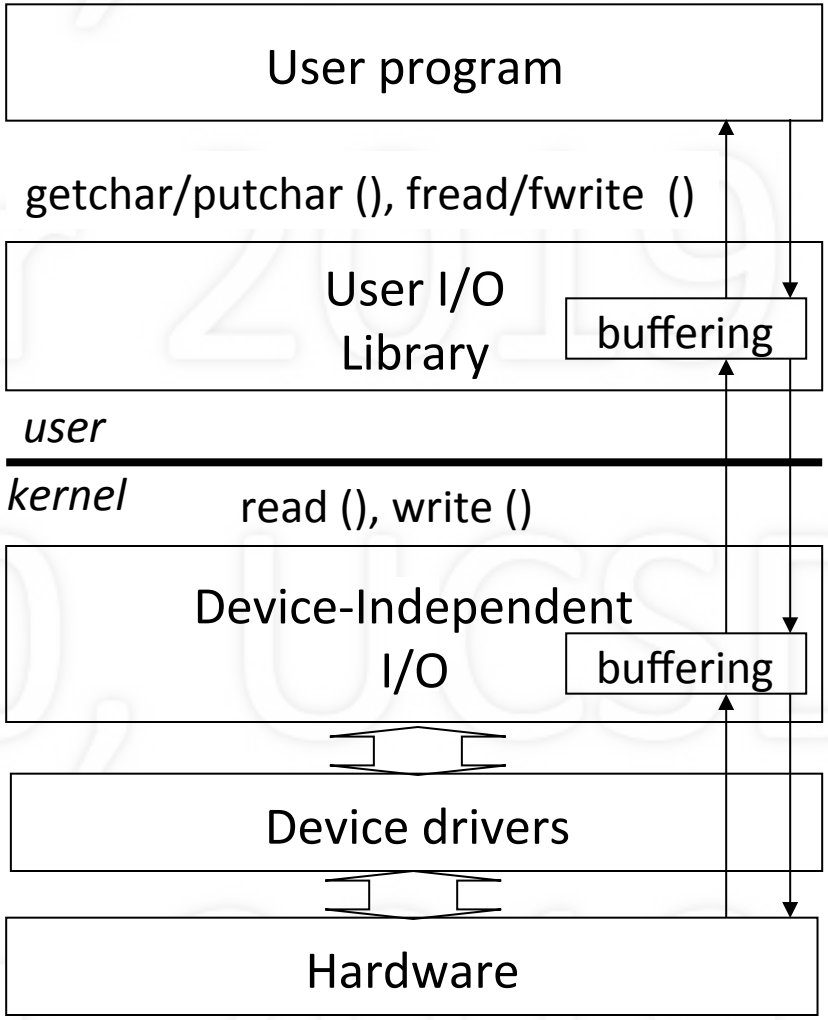
Device-Independent I/O

- Uniform interfacing for device drivers
- Naming, protection
- Uniform block size
- Buffering, caching
- Storage allocation
- Locking
- Error handling

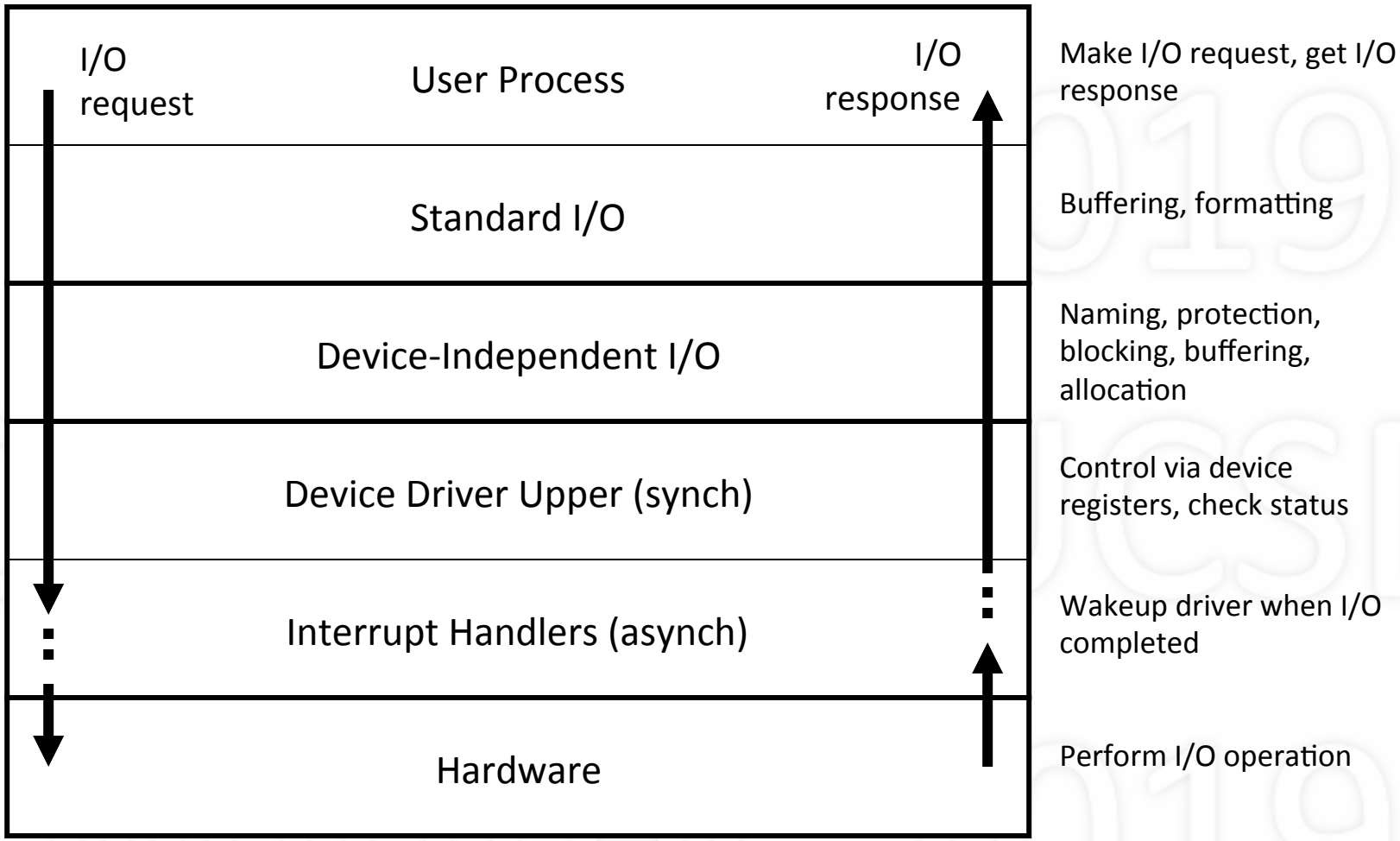


User-space I/O

- Convenient interface
  - printf () vs. write ()
- User-level buffering
  - Unix: stdio library
- Spooling daemons
  - Printer

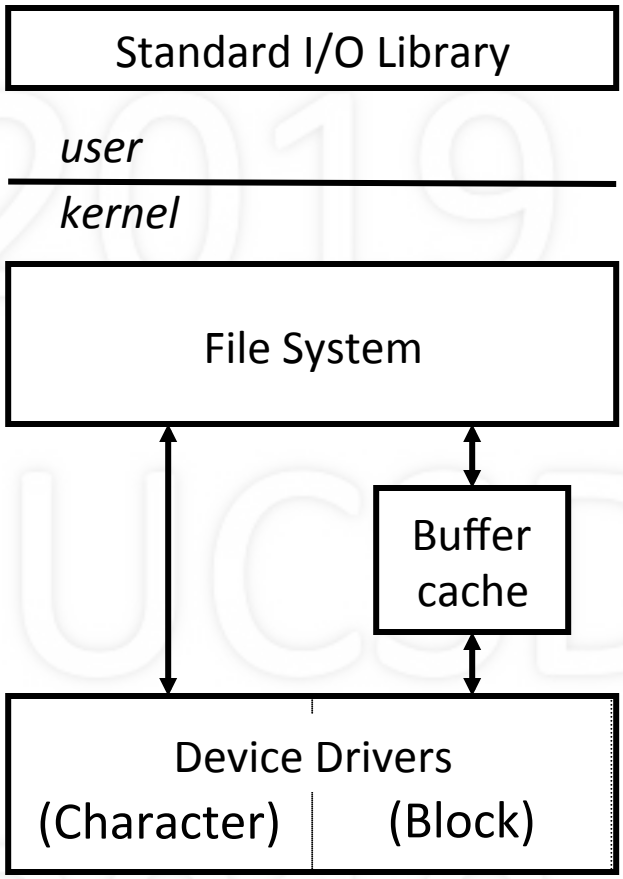


Overall Operation



Example: UNIX I/O Model

- Uses file system interface
- stdio.h: C standard I/O library
- Block devices
  - Fixed-size blocks
  - Randomly addressable
  - Uses buffer cache
- Character devices
  - Variable sequence of bytes
  - For non-block devices



UNIX: I/O System Call Interface

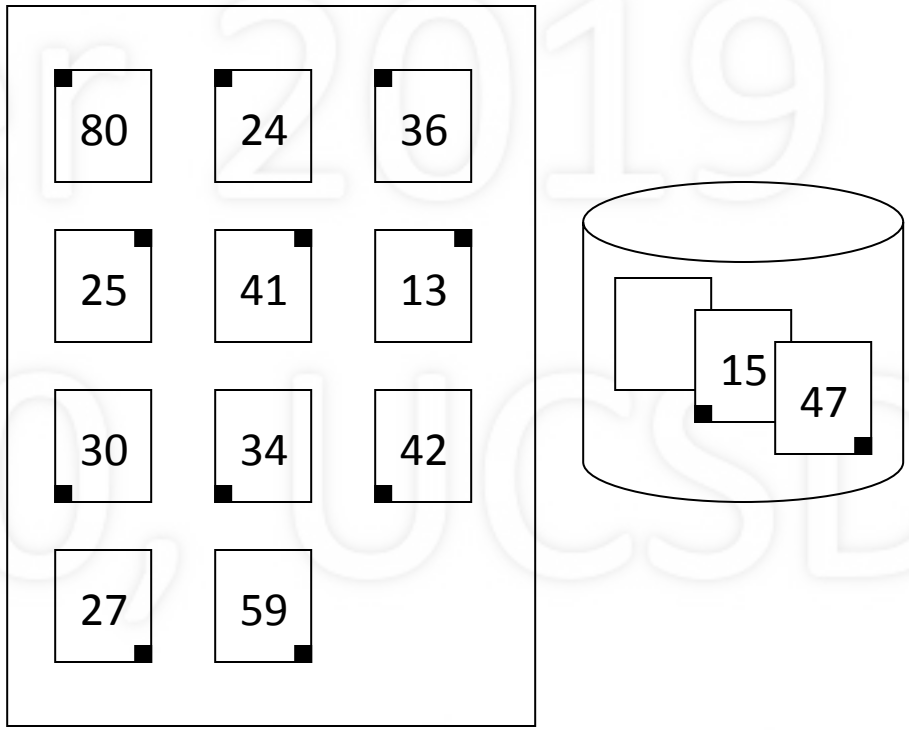
- fd = open (“/dev/devname”, …);
- close (fd);
- nr = read (fd, buf, n);
- nw = write (fd, buf, n);
- ioctl (fd, cmd, buf);

UNIX: Standard I/O Library

- fopen, fread, fwrite, fprintf, fscanf, fclose, …
- Private buffer kept in user space
- Minimizes the number of I/O system calls

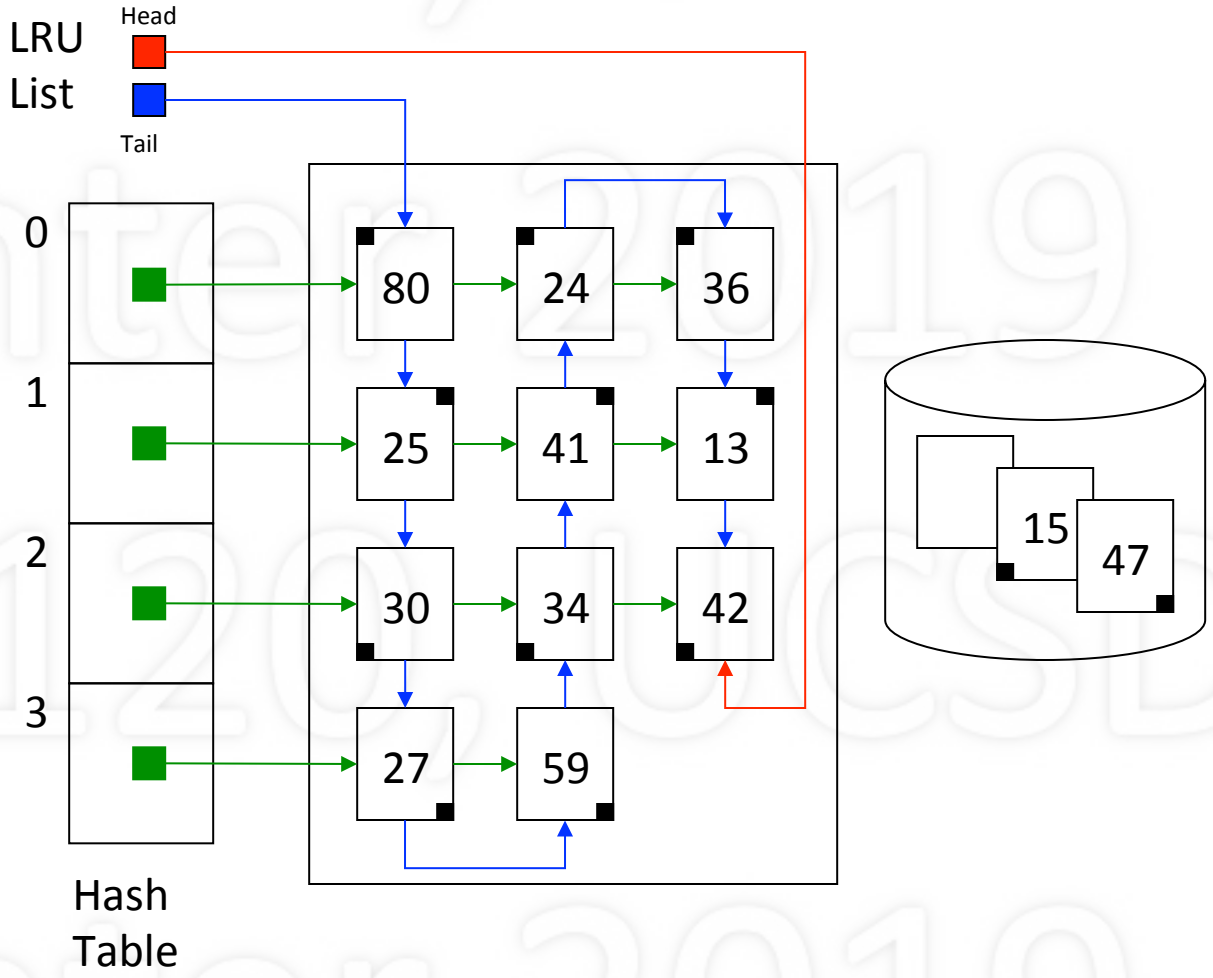
Software Block Cache Design

- Has copies of blocks that are also on disk
- Upon read or write
  - Check if a buffer contains the block
  - If not, get from disk
  - To make room, remove LRU block



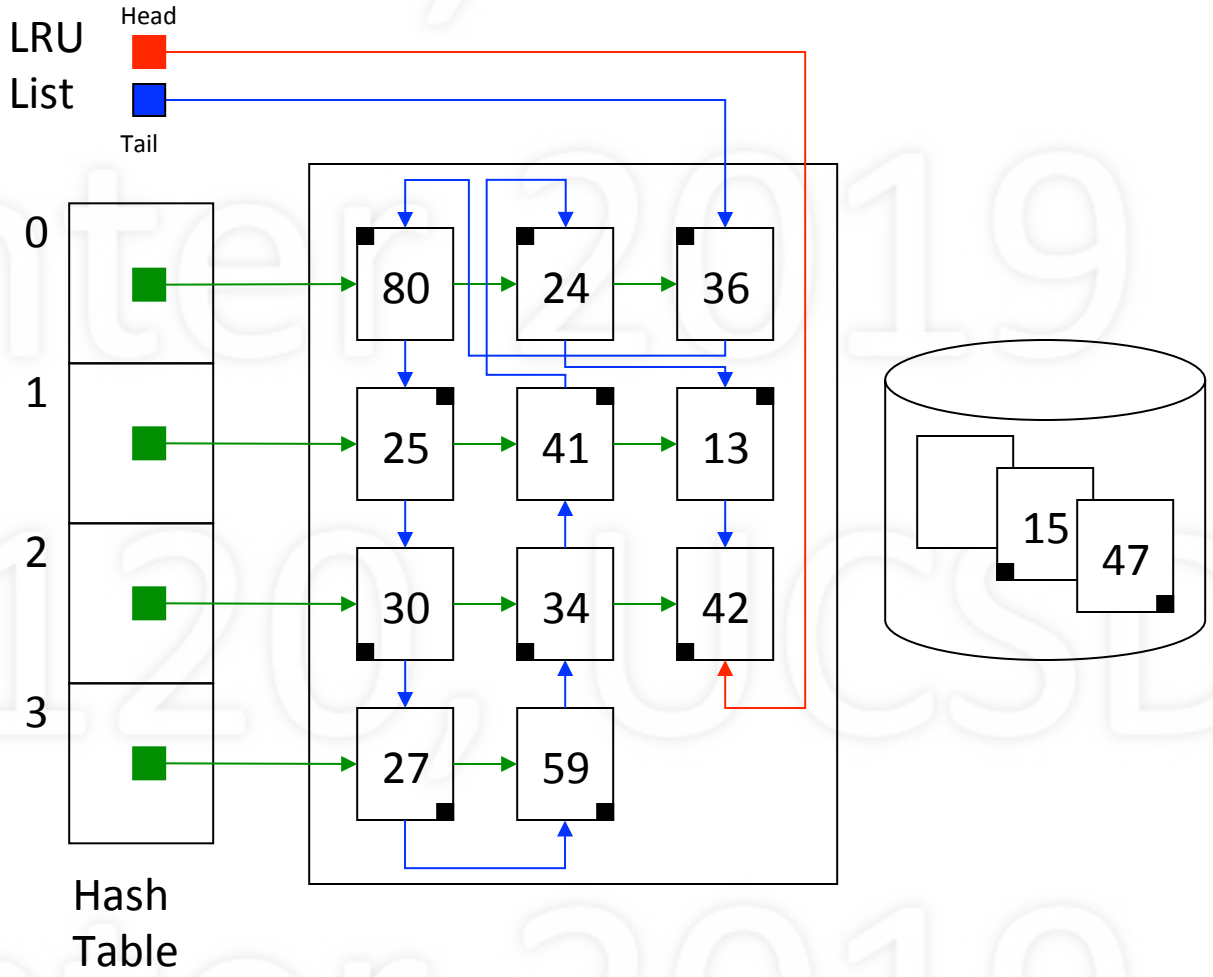
Searching the Buffer Cache

- Read (36)
- 36%4 = 0
- Search list 0 for 36
- Cache hit!



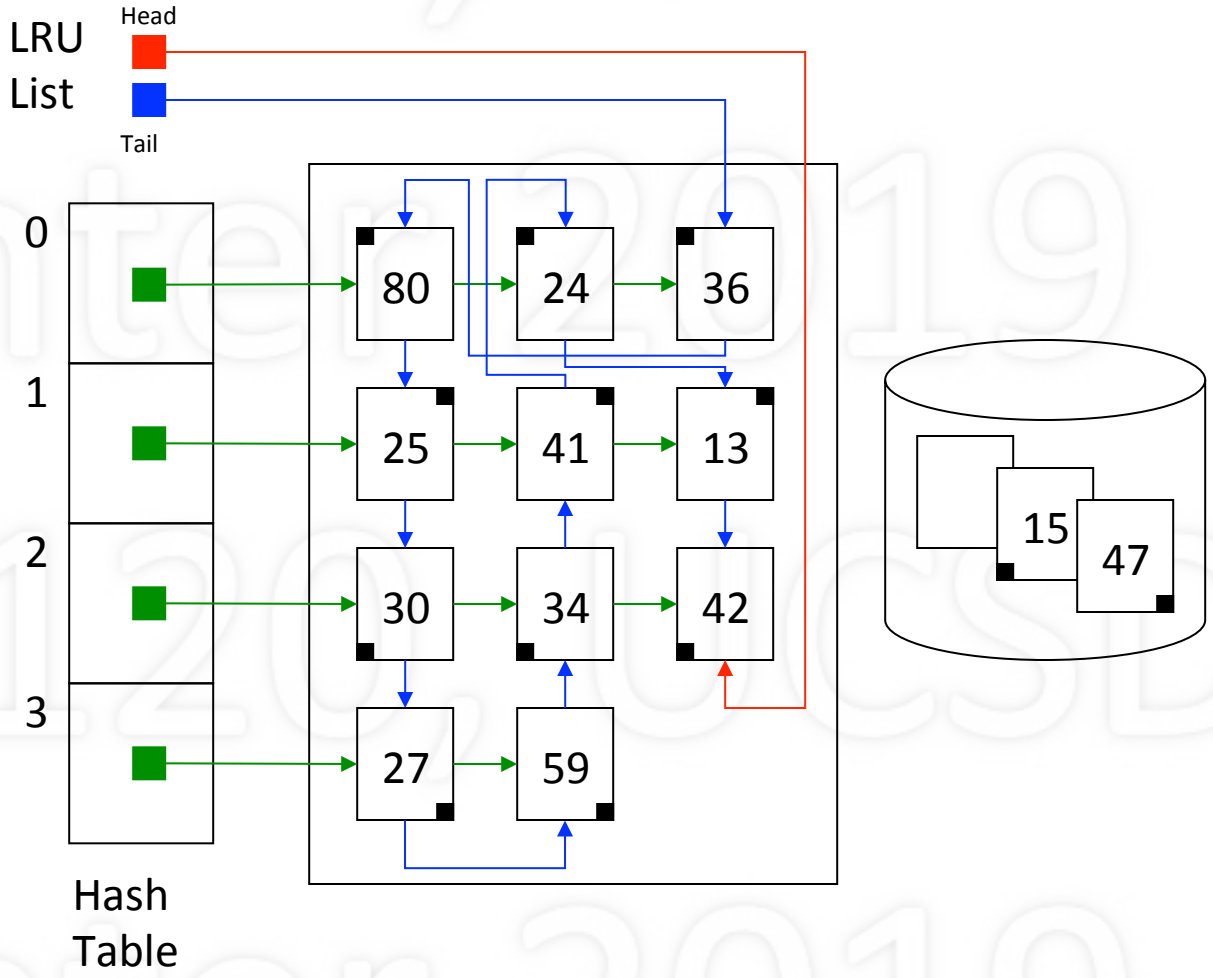
Searching the Buffer Cache

- Read (36)
- 36%4 = 0
- Search list 0 for 36
- Cache hit!
- *Update LRU list*



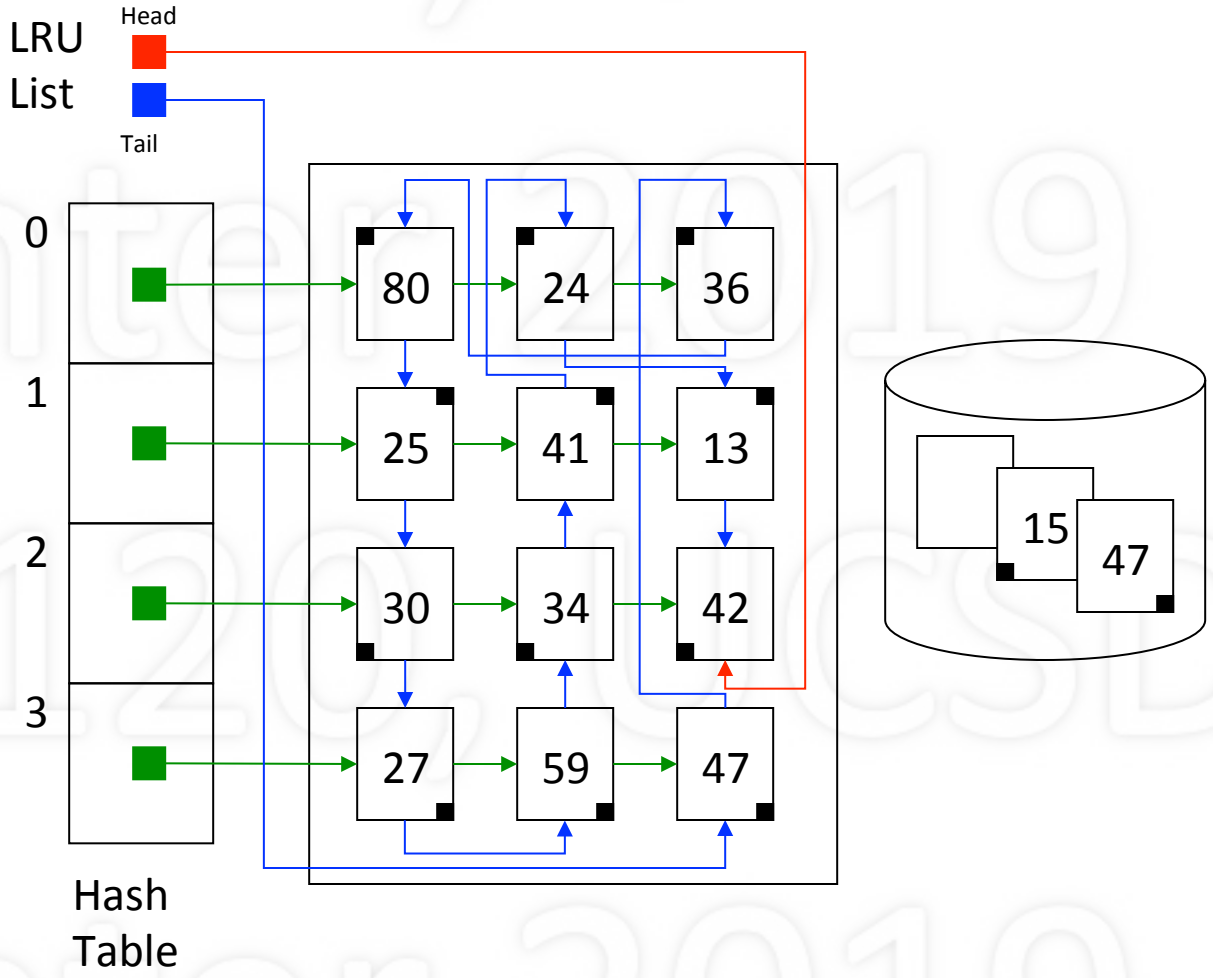
Searching the Buffer Cache

- Read (47)
- 47%4 = 3
- Search list 3 for 47
- Cache miss!



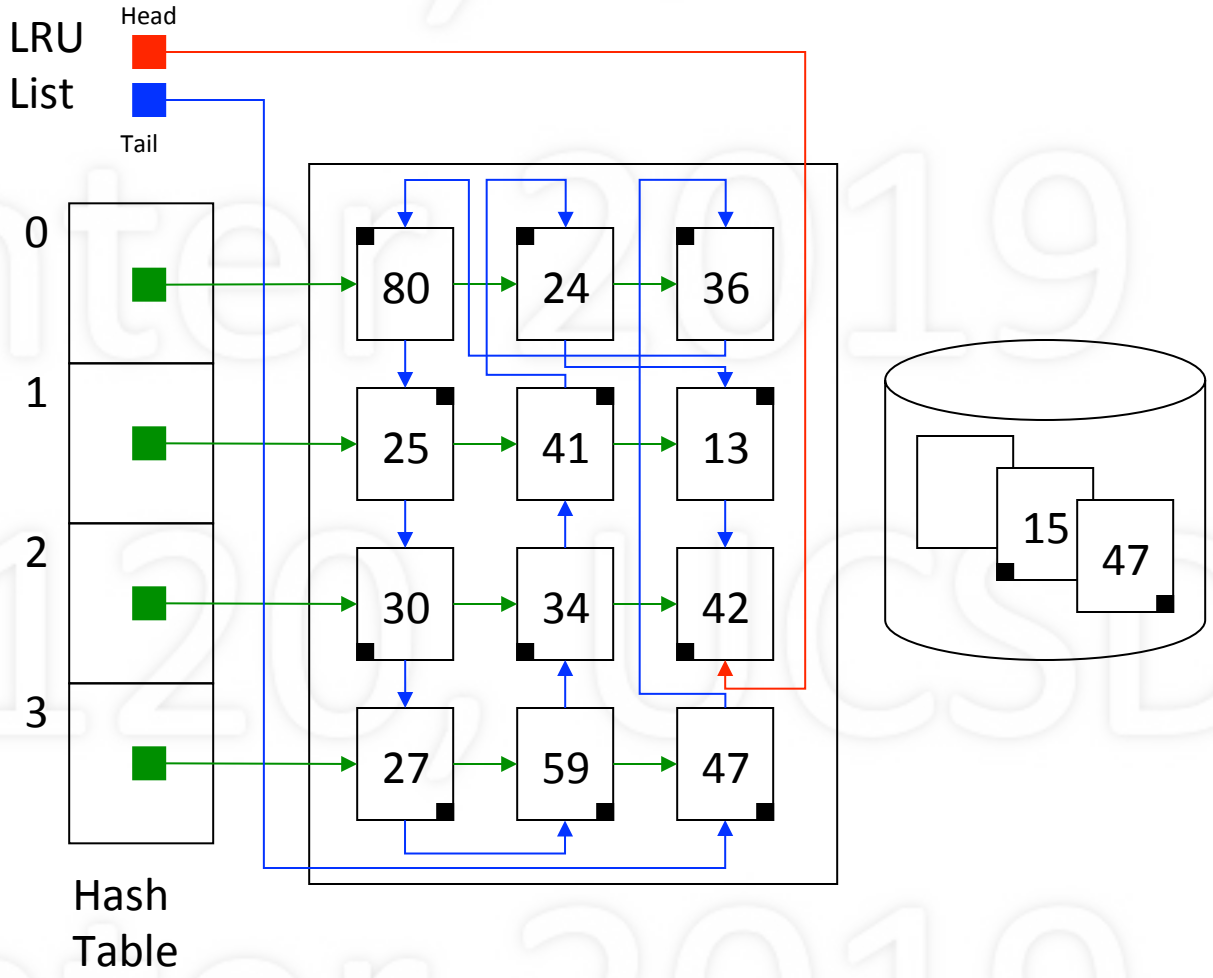
Searching the Buffer Cache

- Read (47)
- 47%4 = 3
- Search list 3 for 47
- Cache miss!
- *Retrieve 47*
- *Update LRU list*



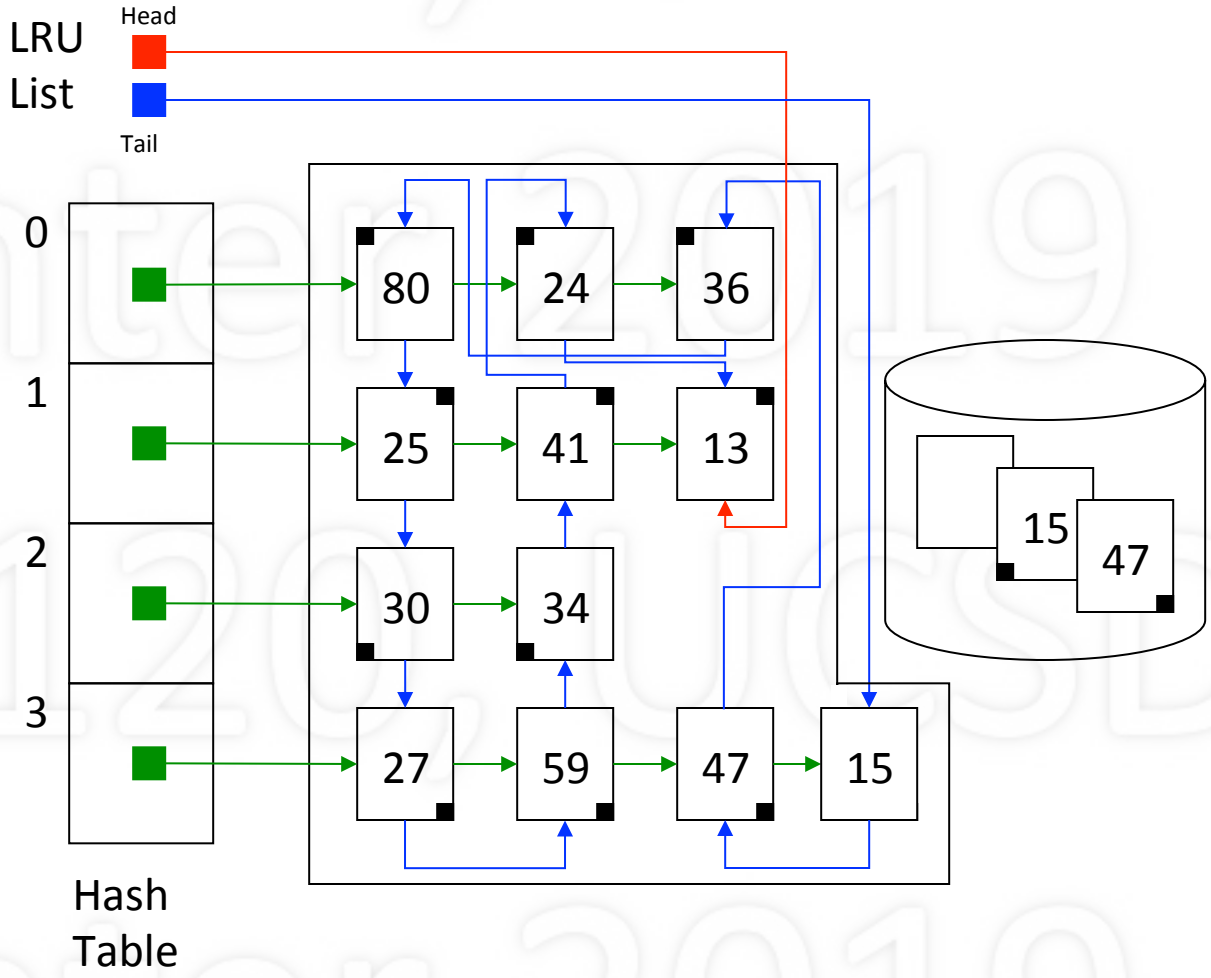
Searching the Buffer Cache

- Read (15)
- 15%4 = 3
- Search list 3 for 15
- Cache miss!



Searching the Buffer Cache

- Read (15)
- 15%4 = 3
- Search list 3 for 15
- Cache miss!
- *Remove 42*
- *Retrieve 15*



Summary

- I/O system structured in layers
  - User-level I/O libraries
  - Kernel-level device-independent
  - Kernel-level device-dependent
    - Upper-level device drivers
    - Lower-level interrupt handlers
- Hardware effect on device drivers
  - I/O vs. memory-mapped, PIO, interrupts, DMA, …

Textbook

- Read Chapter 12 (on I/O Systems)
  - Lecture-related: 12.1-12.5, 12.8
  - Recommended: 12.6-12.7

Review & Research

- What are the primary problems with supporting I/O in an operating system?
- What is the purpose of the I/O bus?*
- What is the difference between the I/O bus and the memory bus?**
- When a device generates I/O, as far as the operating system is concerned, where does it come from (for output) or go to (for input)?*

R&R

- What is the difference between an I/O instruction and memory-mapped I/O?*
- What is the difference between programmed I/O and DMA?*
- How does programmed I/O achieve data transfer and synchronization?**
- How does DMA work achieve data transfer and synchronization?**

R&R

- What is buffering?**
- What is user buffering?*
- What is kernel buffering?*
- What is the difference between buffered I/O and unbuffered I/O?*
- When would you use buffered rather than unbuffered I/O?** How about unbuffered rather than buffered I/O?**

R&R

- How does kernel buffering help if pages containing the user buffer are paged out?*
- How does kernel buffering help if the entire process is swapped out?*
- What is meant by “pinning pages”?**
- What would be the problem if too many processes pin pages?**

R&R

- How does buffering relate to memory copying?**
- Why is memory copying expensive?***
- What is the effect of buffering on caches?***

R&R

- How does the classifying of devices help in managing the complexity of I/O?**
- What are the various ways in which devices can be classified?*
- What is the I/O system (the portion of the OS devoted to I/O) structured in layers?**

R&R

- What is a device driver?*
- What is device-dependent code?*
- Why should device-dependent code be encapsulated in a device driver?*
- Why must a device driver implement a standard interface?**
- What must a device driver do when an interrupt for its device occurs?**

R&R

- What is device-independent I/O?*
- What is the advantage of separating I/O functionality into device-dependent and device-independent portions?**
- What are the types of operations that are characterized as device-independent, and why (as opposed to device-dependent I/O)?**

R&R

- What is user-space I/O?*
- What is the advantage of separating I/O functionality into user-space and kernel-space portions?**
- What are the types of operations that are characterized as user-I/O, and why (as opposed to device-independent I/O)?**

R&R

- On slide 15, if a user process makes an I/O request such as a fread() Standard I/O library call to read from a file located on disk, how does this get translated into calls at each lower level, and what is the functionality of each layer?***
- Continuing, how does an interrupt translate into function returns at the various levels?***

R&R

- In the UNIX I/O Model, what is the difference between a “block device” and a “character device”?*
- What criteria is used to categorize a device as a block device?*
- Is there any advantage to categorizing a device as a block device?**
- What is the buffer cache?*

R&R

- For each of the system calls comprising the UNIX I/O system call interface, what functionality does it provide?**
- What are the purposes of the UNIX Standard I/O library?**
- In what way does the UNIX Standard I/O library enhance system performance?***

R&R

- What is the purpose of the software block cache on slide 19, and how does it work?*
- Why is this cache implemented in software rather than hardware?**
- On slide 20, how does the cache help when block 36 is requested?*
- Why is a hash table used in the design of the cache?***

R&R

- On slide 21, what is the purpose of the LRU list, and how is it updated?**
- On slides 22-23, what happens with the cache when block 47 is requested?*
- On slides 24-26, what happens with the cache when block 15 is requested?**