Today

- File System Reliability
  - RAID
- Memory Management
  - Virtual Memory

> I just spent 20 minutes trying to checkout with my Walmart cart online to have it not work on a door buster item.
> I thought maybe it's an off my one bug so I added a can of spam to the order.
> Yep no problem checking out then.

Review: Spinning Disk Scheduling

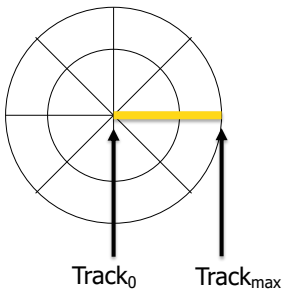
- Why does disk scheduling matter?
- What are the costs in disk scheduling?
- What are our goals for disk scheduling?
- What are some algorithms for disk scheduling?
  - What are their tradeoffs?

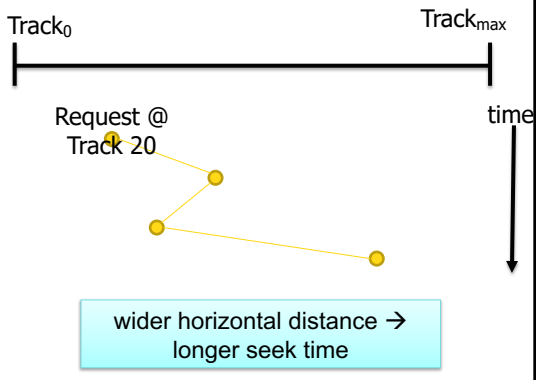
Review: Disk Scheduling

- The operating system is responsible for using hardware *efficiently*
  - For the disk drives: having a fast access time and disk bandwidth
- Minimize seek time
  - Seek time ≈ seek distance
- **Disk bandwidth** is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer

Review: Disk Arm

- Assume the disk arm can move back and forth from left to right and right to left.





Note we didn't talk about rotation time; that's hard for the OS to calculate

Review: Disk Scheduling

- Like CPU scheduling, disk scheduling is a policy decision
  - What should happen if multiple processes all want to access disk?
- Like CPU scheduling, the choice of metric influences the policy decision:
  - Priority: if a process is important, give execute its requests first
  - Wait time/latency, from request to completion
  - Throughput: maximize the data transfer from the disk
  - Fairness: give each process the same opportunity to access the disk

In some cases, trade-off between last two.

Review: Linux's Default Scheduler: CFQ

- Completely Fair Queueing (CFQ)
  - Not to be confused with the "completely fair scheduler (CFS)" for the CPU…

- Keep a disk request queue for each process
- Move requests from process queues to dispatch queue in round-robin fashion (if equal priority)

Can reorder these requests to improve disk performance within some constraints

Review:
Selecting a Disk-Scheduling Algorithm

- D-S algorithm should be written as a separate module of the operating system
  - Can be replaced with a different algorithm if necessary
- Performance depends on the number and types of requests
  - If only one request in queue – becomes FCFS
- SCAN is common
  - Less computational cost
- SCAN and C-SCAN perform better for systems with heavy load on the disk
  - Less starvation
- Add a deadline scheduler
  - if a request hasn't been fulfilled within some period of time, service it

David Patterson

Garth Gibson

Randy Katz

Redundant Array of Inexpensive Disks

RAID

Idea: Replace Small Number of Large Disks with Large Number of Small Disks! (1988 Disks)

	Big, Expensive IBM 3390K	Small, Cheap IBM 3.5" 0061	Small, Cheap x70	
Capacity	20 GBytes	320 MBytes	23 GBytes	
Volume	97 cu. ft.	0.1 cu. ft.	11 cu. ft.	9X
Power	3 KW	11 W	1 KW	3X
Data Rate	15 MB/s	1.5 MB/s	120 MB/s	8X
I/O Rate	600 I/Os/s	55 I/Os/s	3900 IOs/s	6X
MTTF	250 KHrs	50 KHrs	??? Hrs	
Cost	$250K	$2K	$150K	

Disk Arrays have potential for large data and I/O rates, high MB per cu. ft., high MB per KW

But what about reliability?

Array Reliability

- Reliability of N disks = Reliability of 1 Disk ÷ N
  - 50,000 Hours ÷ 70 disks = 700 hours
  - Disk system MTTF: drops from 6 years → 1 month!
- Arrays (without redundancy) too unreliable to be useful!

Hot spares support reconstruction in parallel with access: very high media availability can be achieved

Redundant Arrays of (Inexpensive →Independent) Disks (RAID)

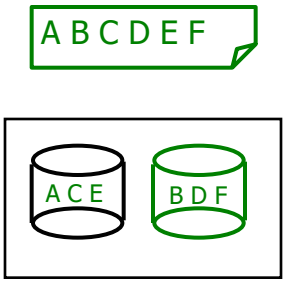
- Key idea: files are ***striped*** across multiple disks
  - Can do reads in parallel on the multiple disks
- Redundancy yields high data availability
  - **Availability**: service still provided to user, even if some components failed

Redundant Arrays of (Inexpensive →Independent) Disks (RAID)

- Disks will still fail
- Contents reconstructed from data redundantly stored in the array
  - ***Capacity penalty*** to store redundant info
  - ***Bandwidth penalty*** to update redundant info
- Multiple schemes
  - Provide different balance between data reliability and input/output performance

Redundant Arrays of Independent Disks
RAID 0: Striping

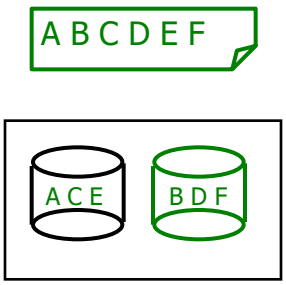
- Stripe data at the block level across multiple disks



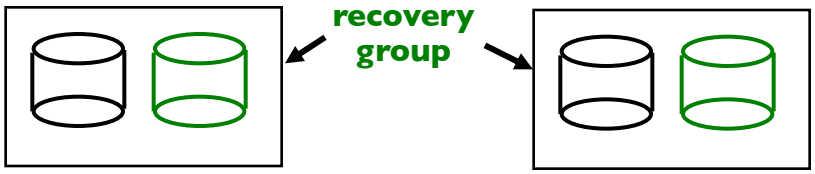
What are the outcomes?
- Expected behavior on read, write?
- On failure?

Redundant Arrays of Independent Disks
RAID 0: Striping

- Stripe data at the *block* level across multiple disks
- High read and write bandwidth
- Not a true **R**AID since no redundancy
- Failure of any one drive will cause the entire array to become unavailable



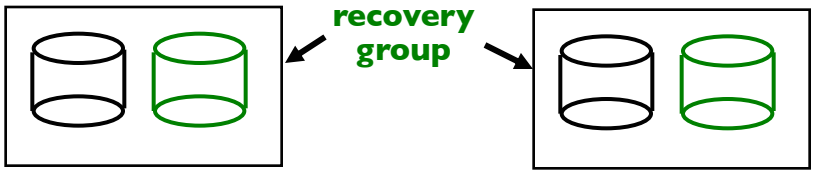
Redundant Arrays of Independent Disks
RAID 1: Disk Mirroring/Shadowing



- Each disk is fully duplicated onto its **mirror**

What are the outcomes?
- Expected behavior on read, write?
- On failure?

Redundant Arrays of Independent Disks
RAID 1: Disk Mirroring/Shadowing



- Each disk is fully duplicated onto its **mirror**
  - Very high availability can be achieved
- Bandwidth sacrifice on write:
  - Logical write = two physical writes
  - Reads may be optimized
- Most expensive solution: 100% capacity overhead

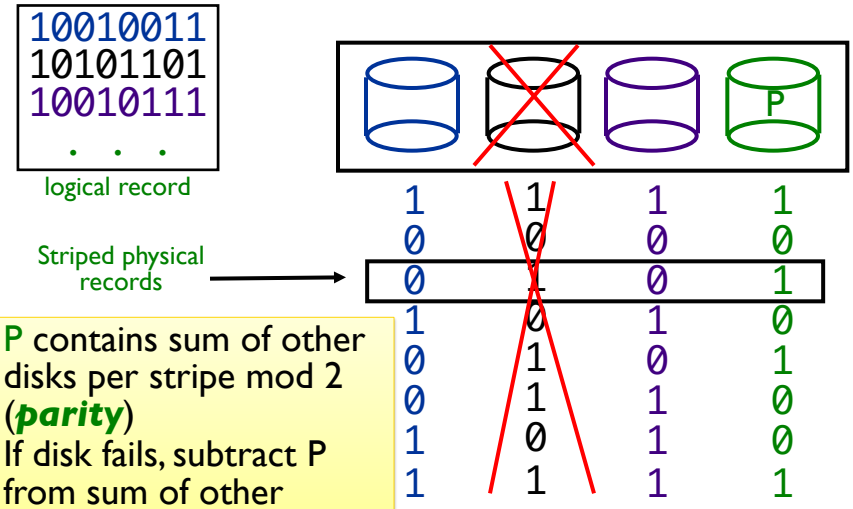
Prefer reliability & performance over low data storage

RAID-I (1989)

- Consisted of a Sun 4/280 workstation with
  - 128 MB of DRAM
  - 4 dual-string SCSI controllers
  - 28 5.25-inch SCSI disks
  - specialized disk striping software

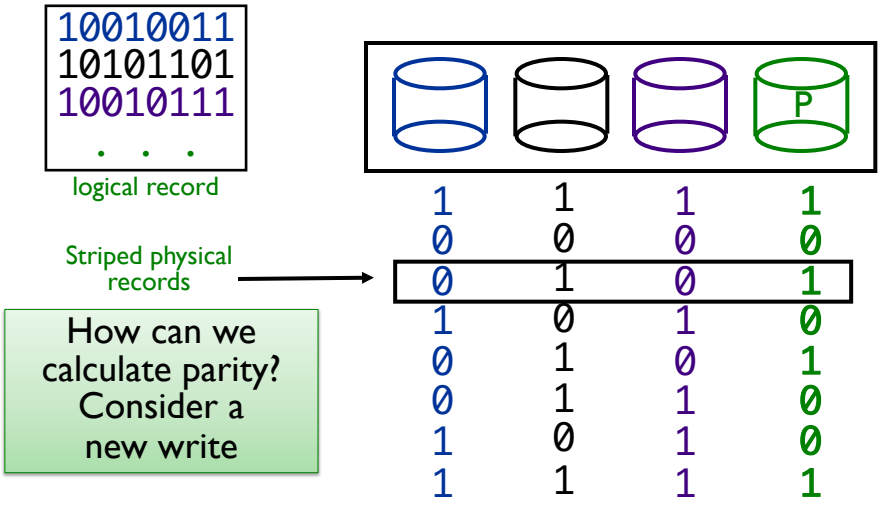
(RAID 2 not interesting, so skip… involves Hamming codes)

Redundant Array of Independent Disks
RAID 3: Parity Disk



- P contains sum of other disks per stripe mod 2 (***parity***)
- If disk fails, subtract P from sum of other disks to find missing information

Redundant Array of Independent Disks
RAID 3: Parity Disk



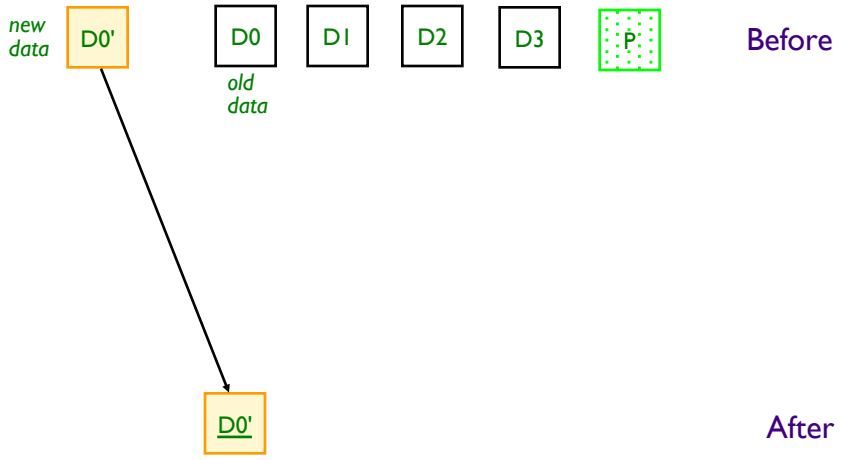
Calculating Parity?

- Option 1: read all data disks, create new sum, and write to Parity Disk
  - Read each disk
  - 2 writes (new data, parity)
- Option 2: since P has old sum, compare old data to new data, add the difference to P
  - 2 reads
  - 2 writes

Problems of Disk Arrays: Small Writes

RAID-3: Small Write Algorithm

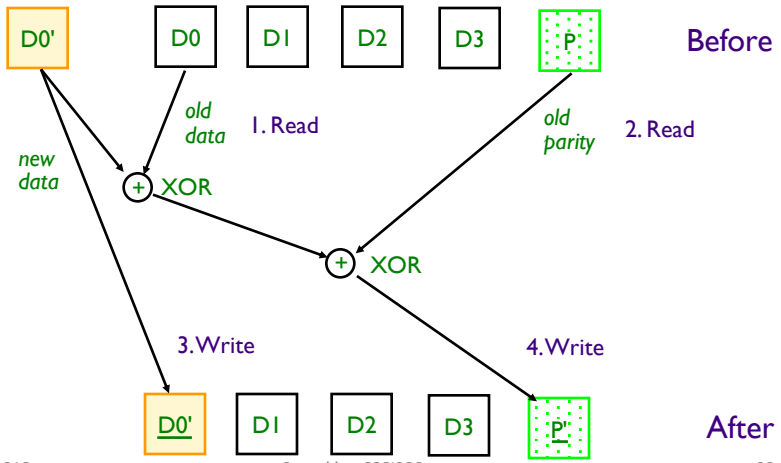
1 Logical Write = 2 Physical Reads + 2 Physical Writes



Problems of Disk Arrays: Small Writes

RAID-3: Small Write Algorithm

1 Logical Write = 2 Physical Reads + 2 Physical Writes



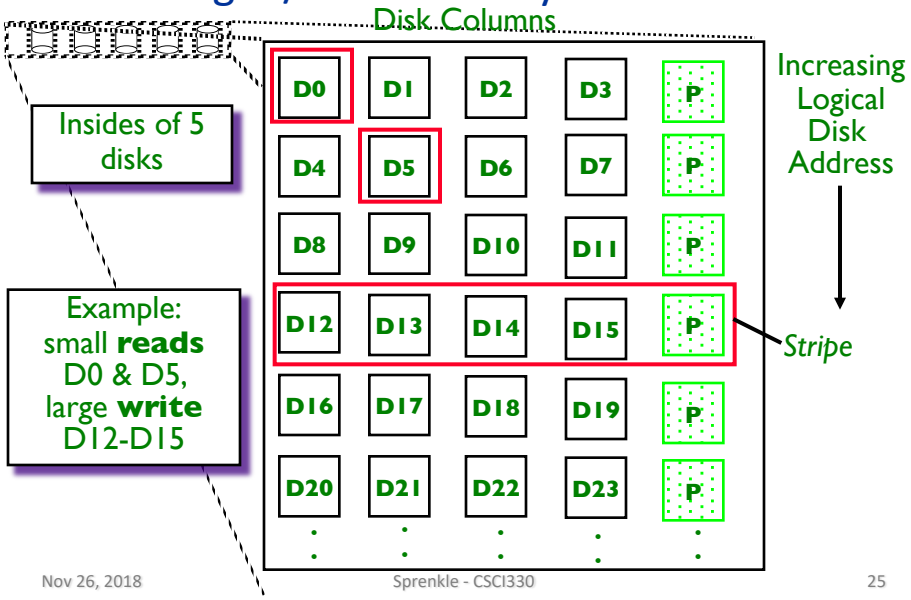
RAID 3

- Sum computed across recovery group to protect against hard disk failures, stored in P disk
- Logically, a single high-capacity, high-transfer-rate disk: good for large transfers
- But byte-level striping is bad for most files
  - all disks involved, even for small writes
- Parity disk is still a bottleneck

Inspiration for RAID 4

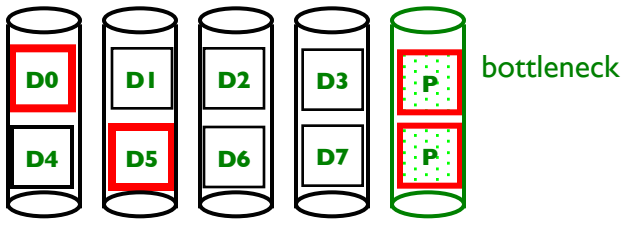
- RAID 3 stripes data at the *byte* level
  - RAID 3 relies on parity disk to discover errors on read
  - But every sector on disk has an error detection field
- Idea: *Block-level* striping
  - Rely on error detection field to catch errors on read, not on the parity disk
- Goals:
  - Allow independent reads to different disks simultaneously
  - Increase read I/O rate since only one disk is accessed rather than all disks for a small read

Redundant Arrays of Independent Disks
RAID 4: High I/O Rate Parity



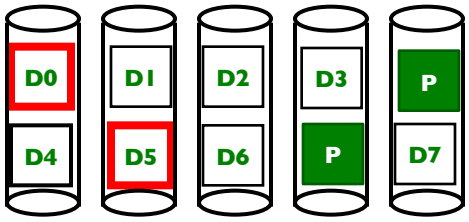
Inspiration for RAID 5

- RAID 4 works well for small reads
- Small writes are still limited by Parity Disk:
  - Write to D0, D5, both also write to P disk



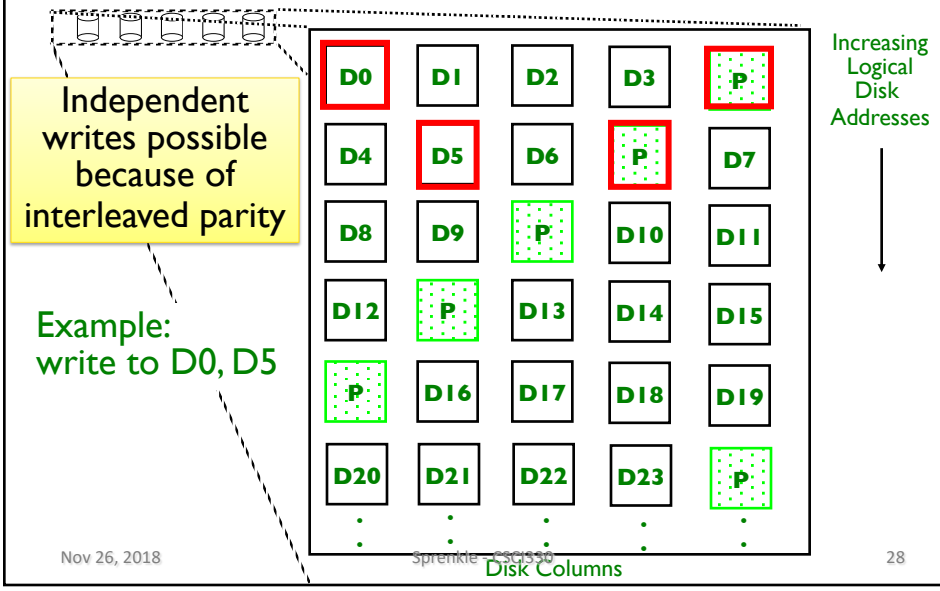
RAID 5

- RAID 4 works well for small reads
- Small writes are still limited by Parity Disk:
  - Write to D0, D5, both also write to P disk
- Idea: Rotate the Parity disk

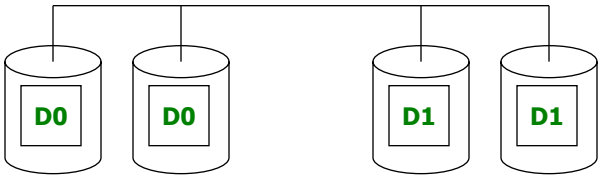


Result: same disk isn't a bottleneck for *all writes*

Redundant Arrays of Independent Disks
RAID 5: High I/O Rate Interleaved Parity



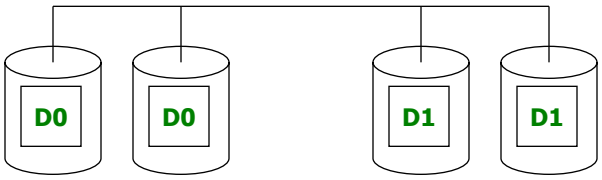
RAID-10 (0+1)



- Striping + mirroring
  - Stripes (RAID-0) across reliable logical disks, implemented as mirrored disks (RAID-1)

What's the impact?
(What are the costs?)

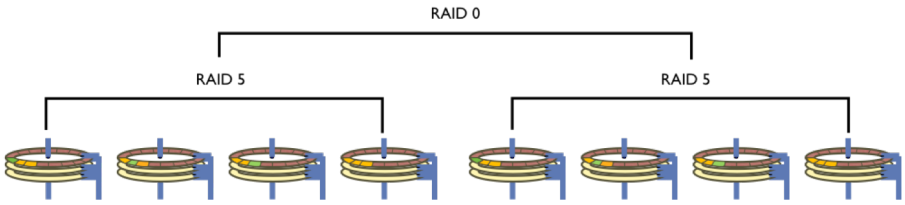
RAID-10 (0+1)



- Striping + mirroring
  - Stripes (RAID-0) across reliable logical disks, implemented as mirrored disks (RAID-1)
- High storage overhead/cost
- For small write-intensive apps, may be better than RAID-5
  - Write data twice but no reads or XORs required

RAID-50

- Stripes (RAID-0) across groups of disks with block interleaved distributed parity (RAID-5)
  - Write is striped (RAID-0) to two sets of disks implemented RAID-5
- Increased write performance and better data protection



Who is in control?

- Hardware
  - Pros:
    - Tends to be reliable
      - hardware implementers test
    - Offloads parity computation from CPU
  - Cons
    - Dependent on card for recovery (replacements?)
    - Must buy card (for the PCI bus)
- Software
  - Pros
    - Other OS instances might be able to recover
    - No additional cost
      - Part of OS
  - Cons
    - Software has bugs
    - Ties up CPU to compute parity

RAID Weaknesses

- Disks tend to be the same age
  - Similar failure times
- Inflexible organization
  - Disks, file system requirements change over time
- Rebuild time is expensive
  - Disk capacity has increased
  - Transfer speed has increased a little
  - Error rates decreased a little

Another process resource

MEMORY MANAGEMENT

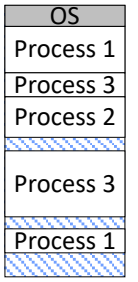
Memory Management

- Basic hardware capabilities
  - Logical vs. Physical addresses
  - Address binding
- Multiprogramming and Memory
- Virtual Memory

Ties in with Project 5: Processes & Multiprogramming

Memory

- Reality
  - there's only so much memory to go around
  - no two processes should use the same (physical) memory addresses.



Physical Memory

Drab and ugly

- Abstraction goal: make every process think it has the same memory layout

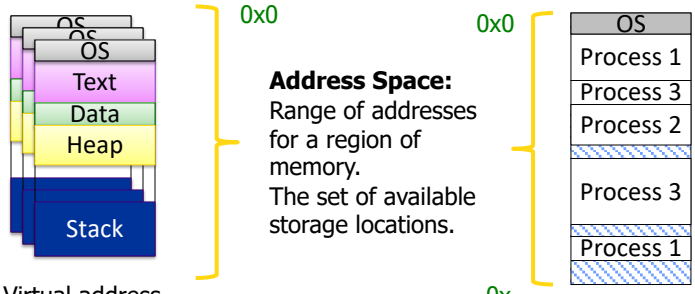
Abstraction

Colorful and happy

Memory Terminology

Virtual (logical) Memory:
The abstract view of memory given to processes.
Each process gets an independent view of the memory

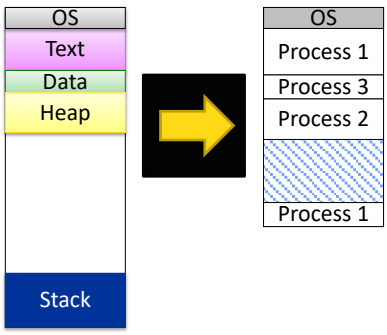
Physical Memory:
The contents of the hardware (RAM) memory. Managed by OS.
Only **one** of these for the entire machine!



Virtual address space (VAS): fixed size (CPU)

(Determined by amount of installed RAM.)

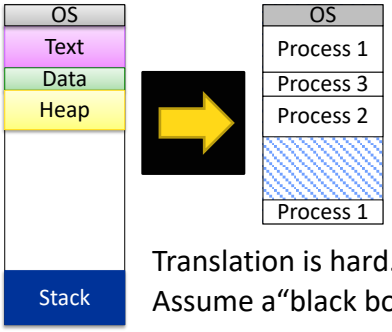
Address Translation: Wish List



- Map virtual addresses to physical addresses

Address Translation

- Virtual addresses must be translated to physical addresses

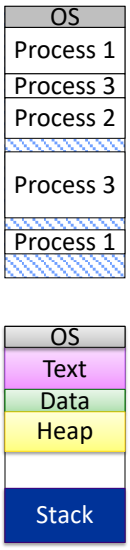


Translation is hard.
Assume a"black box" mechanism does it.

- Is logical addressing worth it/necessary?
- What do we want it to provide for us?

How does each benefit from having a logical memory abstraction?

- The user
- The programmer
- The compiler
- The OS / OS designer
- The hardware / hardware designer



Looking Ahead

- Project 4 due *tomorrow* night