Today

- Synchronization Mechanisms – tradeoffs
- File Systems
  - Disk Storage

Review

- The Dining Philosophers is a classic synchronization problem
  - What issues did it bring up?
  - What were our goals for a solution?
  - What ideas for solutions did we have?

Review: Conditions for Deadlock

- Four conditions must be present for deadlock to occur:
  1. Non-preemption of ownership. Resources are never taken away from the holder.
  2. Exclusion. A resource has at most one holder.
  3. Hold-and-wait. Holder blocks to wait for another resource to become available.
  4. Circular waiting. Threads acquire resources in different orders.

Review: Dealing with Deadlock

1. Ignore it. Do you feel lucky?
2. Detect and recover. Check for cycles and break them by restarting activities (e.g., killing threads).
3. Prevent it. Break any precondition.
   - Keep it simple. Avoid blocking with any lock held.
   - Acquire nested locks in some predetermined order.
   - Acquire resources in advance of need; release all to retry.
   - Avoid "surprise blocking" at lower layers of your program.
4. Avoid it.
   - Deadlock can occur by allocating variable-size resource chunks from bounded pools
     - Google "Banker's algorithm".

Review: Possible Solutions to Dining Philosophers

- Asymmetric solution
  - Some pick up left chopstick first, some pick up right
    - How does that play out?
- Don't pick up either chopstick until *both* are free
  - How would you implement this?
- Allow a philosopher to take a chopstick from another philosopher who isn't yet eating
- Not ideal
  - Reduce the number of philosophers or increase the number of resources

Still issues with starvation--
Need guarantee of locks being acquired in order

Review: Deadlock vs. Starvation

- A ***deadlock*** is a situation in which a set of threads are all waiting for another thread to move.
  - But none of the threads can move because they are all waiting for another thread to do it.
- Deadlocked threads sleep "forever": the software "freezes".
  - It stops executing, stops taking input, stops generating output. There is no way out.
- ***Starvation*** (also called ***livelock***) is different:
  - Some schedule exists that can exit the livelock state, and the scheduler may select it, even if the probability is low.

SYNCHRONIZATION WRAPUP

Comparing Synchronization Mechanisms

- Synchronization Mechanisms
  - Lock
  - CV
  - Semaphore
  - Monitors
- Compare and Contrast
  - When to use

Synchronization Mechanisms

- Mutex/lock
  - Mutual exclusion: only one thread can access a resource at a time
- Signaling mechanisms:
  - Condition Variable
  - Semaphore
- Monitor: lock/CV combo

Semaphores vs. Mutex

- A binary semaphore is similar to a mutex, but …

We talked about correct use, but …
what could we do with semaphores that is prevented with a mutex?

Semaphores vs. Mutex

- A binary semaphore is similar to a mutex, but …
- Mutex has an *owner*
  - Only the owner can acquire/release the lock
- Semaphores: anyone *could* release the lock

Semaphores vs. Condition Variables

- Semaphores are like CVs with an atomic integer
- V(Up) differs from signal (notify) in that …?
- P(Down) differs from wait in that …?

Semaphores vs. Condition Variables

- Semaphores are like CVs with an atomic integer.
- V(Up) differs from signal (notify) in that:
  - Signal has no effect if no thread is waiting on the condition
    - Condition variables are not variables! They have no value!
  - Up has the same effect whether or not a thread is waiting
    - Semaphores retain a "memory" of calls to Up.
- P(Down) differs from wait in that:
  - Down checks the condition and blocks only if necessary.
    - No need to recheck the condition after returning from Down
    - The wait condition is defined internally, but is limited to a counter
  - Wait is explicit: it does not check the condition itself, ever
    - Condition is defined externally and protected by integrated mutex

Monitors vs. semaphores

- Monitors
  - Separate mutual exclusion and wait/signal operations
- Semaphores
  - Provide both with same mechanism
- Semaphores are more "elegant"
  - At least for producer/consumer (counted resources)
  - Can be harder to program

Monitors vs. semaphores

```
// Monitors
lock (mutex)

while (condition) {
  cv.wait (mutex)
}

unlock (mutex)
```

```
// Semaphores
down (semaphore)
```

- Where are the conditions in both?
- Which is more flexible?
- Why do monitors need a lock, but not semaphores?

Monitors vs. semaphores

```
// Monitors
lock (mutex)

while (condition) {
  cv.wait (CV, mutex)
}

unlock (mutex)
```

```
// Semaphores
down (semaphore)
```

- When are semaphores appropriate?
  - When shared integer maps naturally to problem at hand
    - when the condition involves a count of one thing

Where We Are …

- We've talked about
  - Kernel
  - Processes, process management
  - Synchronization
- Moving toward storage
  - File systems
    - Disk management, storage
  - Memory management

Why Do We Want File Systems?

- What are the goals of file systems?
  - What adjectives do you want to use to describe file systems?

Goals for File Systems

- Long-term storage
  - Persistent: remains "forever"
- Reliable
- Large capacity, low cost
- High performance
- Named data (lookup/query)
- Controlled sharing
- Security: protecting information

DISK STORAGE

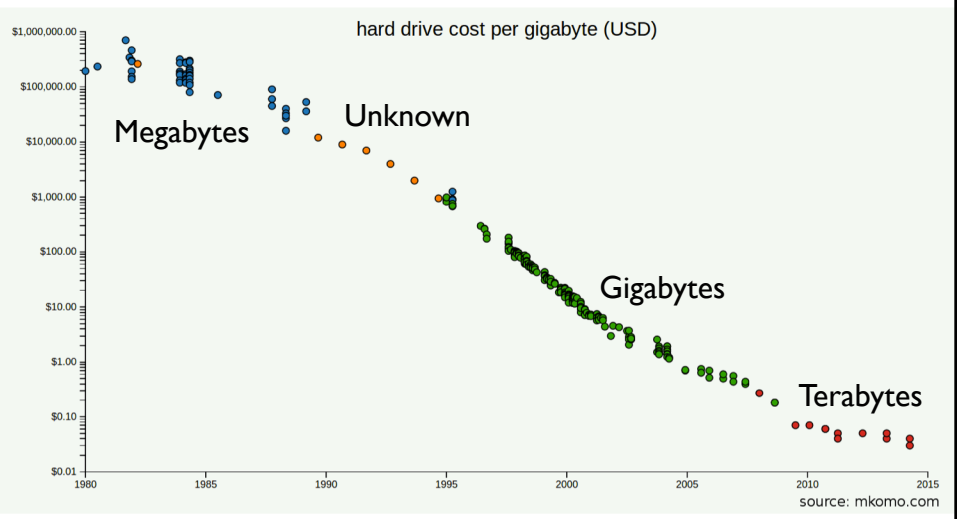
Using Disks for Storage

Why disks? persistent, random access, cheap

The First Commercial Disk Drive

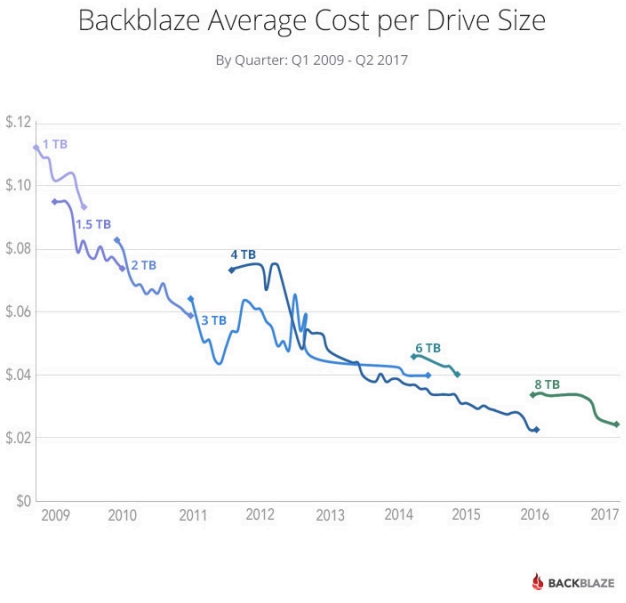
1956
IBM RAMDAC computer included the IBM Model 350 disk storage system

5M (7 bit) characters
50 x 24" platters
Access time = < 1 second



Point colors relate to the size of the drive

Source: http://www.mkomo.com/cost-per-gigabyte-update



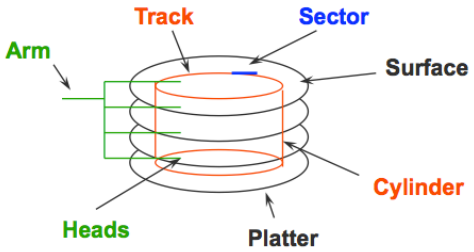
Source: https://www.backblaze.com/blog/hard-drive-cost-per-gigabyte/

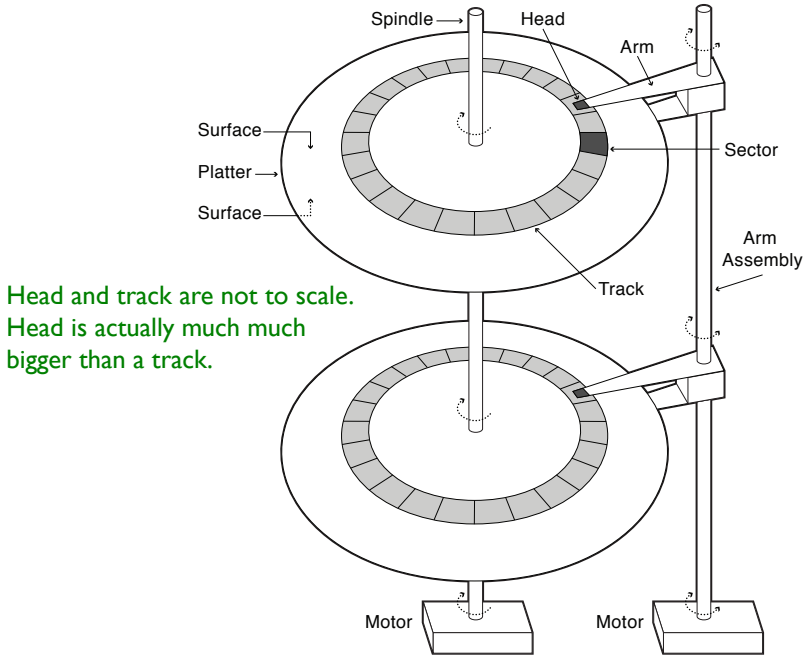
Using Disks for Storage

- Why disks? persistent, random access, cheap
- Biggest hurdle to OS: disks are [relatively] slow

Disk Geometry

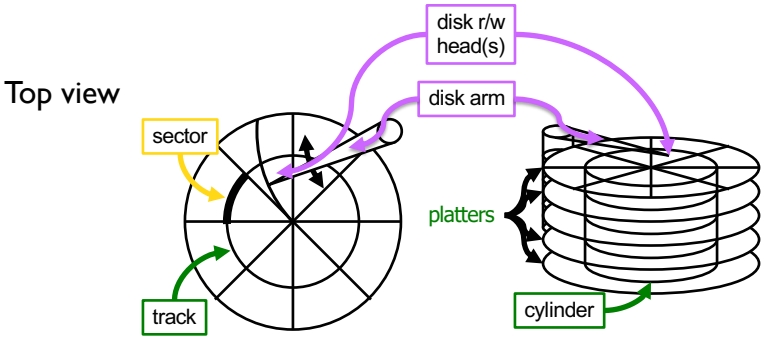
- Disk components
  - Platters
  - Surfaces
  - Tracks
  - Sectors
  - Cylinders
  - Arm
  - Heads





Head and track are not to scale. Head is actually much much bigger than a track.

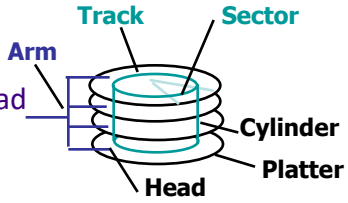
Hard Disk Performance: Moving Parts



- Moving parts: spinning platters, disk actuator arm
- Much more likely to fail than most other components

How Long to Access Data on Disk?

- 5-15 ms on average for access to random location
  - Includes seek time to move head to desired track
    - Roughly linear with radial distance
  - Includes rotational delay
    - Time for sector to rotate under head
- Times depend on drive model:
  - platter width (e.g., 2.5 in vs. 3.5 in)
  - rotation rate (5400 RPM vs. 15K RPM).
  - Enterprise drives use more/smaller platters spinning faster.
- These properties are mechanical and improve as technology advances over time



Disk Interaction: 1960s – 80s

- Specifying disk requests required a lot of info:
  - Cylinder #, head #, sector # (CHS)
  - Disks did not have controller hardware built-in
- Early OSes needed to know this info to make requests but didn't optimize data storage for it
- ~mid 80's: "fast file system" emerged, which took disk geometry into account.
  - Paper: "A Fast File System for Unix"
    - https://dl.acm.org/citation.cfm?doid=989.990
  - Example disk in paper is 150 MB

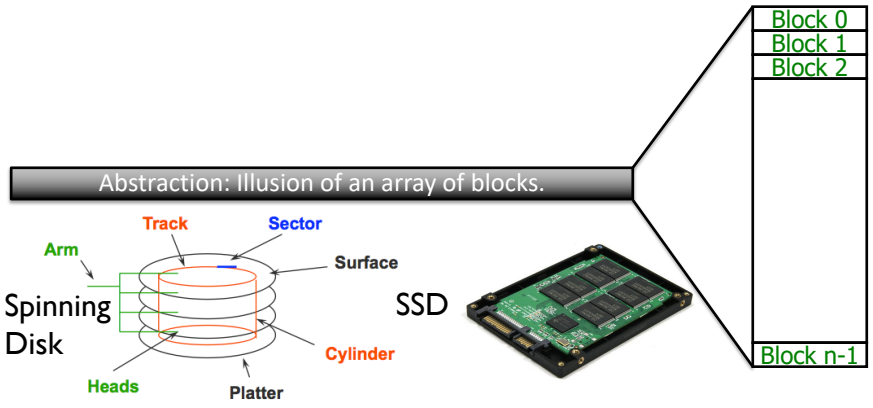
A few words about SSDs

- Solid State Drives (e.g., Flash memory):
  - No spinning platter, no arm to move, no mechanicals.
  - Faster than disk (at least for reads), slower than DRAM.
  - **No seek cost**. But writes require slow block erase and/or limited # of writes to each cell before it fails.
  - Technology is advancing rapidly; costs are dropping
- How should we use them? Are they just fast/expensive disks? Or can we use them like memory that is persistent? Open research question.
- **Trend**: use them as block storage, and/or combine them with HDDs to make hybrids optimized for particular uses.

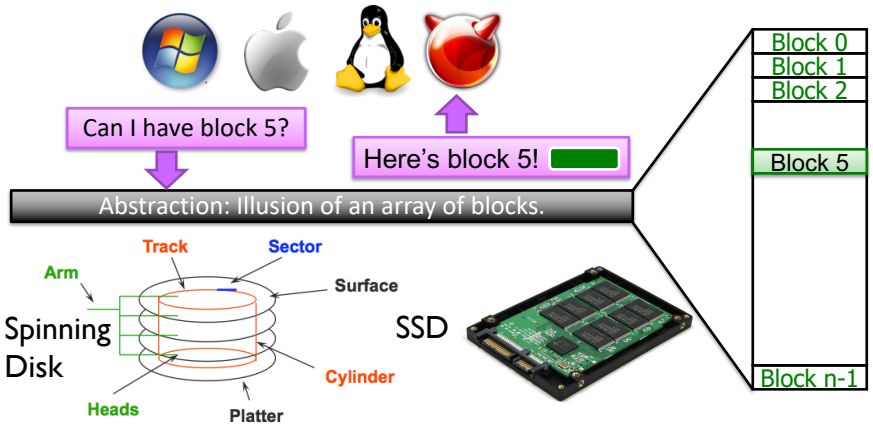
Modern Disk Interaction

- Very simple block number interface:
  - Disk is divided into N abstract blocks (traditionally 512 B, today often 4 KB)
  - `read(block #)`
  - `write(block #, data)`
- Trust the disk controller
  - Convert block number to the "right" place in disk geometry.
  - For some disks (SSDs), this may not even be the same location every time!
- Significant research happening now in new types of storage

Storage Abstraction for File System



Storage Abstraction for File System



Looking Ahead

- Synchronization Assignment due Monday
- File Systems!