I've got a bunch of virtual Windows machines networked together, hooked up to an incoming pipe from the net. They execute email attachments, share files, and have no security patches. Between them they have practically every virus.

There are mail trojans, Warhol worms, and all sorts of exotic polymorphs. A monitoring system adds and wipes machines at random. The display shows the viruses as they move through the network, growing and struggling.

You know, normal people just have aquariums. Good morning, Blaster. Are you and W32.Welchia getting along?

Pretty, isn't it?

What is it?

Who's a good virus? You are! Yes, you are!
Flash Memory

Flash Memory describes a particular hardware *implementation* the Storage abstraction.
Flash Memory Specifications

- Digital (no mechanical components)
- Non-Volatile (Like Disk)
- 100X Faster than Disk (1000x slower than RAM)
- 5-10X MORE expensive than Disk (5-10X Less expensive than RAM)
Flash Memory Specifications

- Digital (no mechanical components)
- Non-Volatile (Like Disk)
- 100X Faster than Disk (1000x slower than RAM)
- 5-10X MORE expensive than Disk (5-10X Less expensive than RAM)

This sounds great! What’s the catch?
Flash Memory Specifications

- Digital (no mechanical components)
- Non-Volatile (Like Disk)
- 100X Faster than Disk (1000x slower than RAM)
- 5-10X MORE expensive than Disk (5-10X Less expensive than RAM)

This sounds great! What’s the catch?

It’s not easy to make non-volatile memory that is digital (i.e. not mechanical)
Flash Memory Quirks

In order to write to any page, you must ERASE the entire block (called an erase unit), which is usually much larger than the page itself (~100x).

What’s worse, a specific location on the flash memory can only be erased a certain number of times (called wear-out) before it is unusable. This means that flash capacity has a finite amount of usage.

Let’s say that the user very frequently accesses certain locations on the flash chip (let’s say, for the ROOT inode). How might we go about preventing early wear-out for that region?
Think Pair Share

Warranties for Flash Storage devices will often say something like:

“Protected for the first 4 years or 6TB worth of writes”

Why is that?
FTL

Much like how we handled *virtual memory*, Flash Devices come packaged with special software called the Flash Translation layer -> it maps storage addresses the OS uses to actual locations in the flash chip.
FTL

Much like how we handled virtual memory, Flash Devices come packaged with special software called the Flash Translation layer - it maps storage addresses the OS uses to actual locations in the flash chip.

The implication here is that we need to store a block map that maps virtual blocks to physical blocks.

While the computer is on, we can store the block map in memory or entirely in the flash itself. What are benefits and drawbacks of doing either?
FTL - Garbage Collection

A nice thing about the FTL is that you can move pages around within blocks to combat fragmentation within erase units.

Recall that erasure units contain many (10-100X) pages per block. If we want to discard garbage pages from a unit, this requires:
- Reading the whole erasure unit into memory
- Figuring out what is garbage and what is not (in memory - basically free)
- Write out the valid pages elsewhere (costly!) and erase the original block

To write the valid pages elsewhere, we need to maintain some number of unused erasure units at all times!
FTL - Garbage Collection

Important math:

If I need space for an entire erase unit (which is a bunch of pages):

How many writes does it take if I have ample space and many completely empty erase units?

How about if my device is 95% utilized (meaning that, on average, 5% of pages within the block are free, and the other 95% contain live data?)
FTL - Garbage Collection

Important math:

If I need space for an entire erase unit (which is a bunch of pages):

How many writes does it take if I have ample space and many completely empty erase units?

How about if my device is 95% utilized (meaning that, on average, 5% of pages within the block are free, and the other 95% contain live data?)

We call this *unexpected cost increase* **write amplification**.
Flash - Final Thoughts

The issue with the FTL (basically the virtual memory mapping controller for flash) is that our OS does not control it (like it controls virtual memory or the file system). This has a few interesting consequences:

- We will **duplicate** structures that have the same overall affect (FTL block map is a lot like the inode table structure). This wastes space in our storage and memory!
- The OS does not tell the FTL when it frees pages (i.e. truncates or deletes a file). This means that the FTL might think a page in a block is live but in reality it should be reclaimed (causing artificially higher device utilization, implying worse write amplification!!!)
Announcements

Please fill out the when2meet so that we can get a time that works for as many people as possible!

Office hours will be held Thursday in OU (2nd floor), but I am cutting them to 9AM-10AM because I am out most of Thursday.
Virtual Machines

High Level Idea -> Emulate an operating system of your choice (i.e. MACOSX Big Sur 5.03) that just looks like a normal user process on your computer.
Virtual Machines

High Level Idea -> Emulate an operating system of your choice (i.e. MACOSX Big Sur 5.03) that just looks like a normal user process on your computer.

For our computers, this means that we will have 2 operating systems running at the same time. The software that services the VM operating system (called the guest OS) is called the **hypervisor**. You can think of it as the **foundation** that you need to build a virtual machine on.
Virtual Machines

High Level Idea -> Emulate an operating system of your choice (i.e. MACOSX Big Sur 5.03) that just looks like a normal user process on your computer.
Running Virtual Machines

In practice, we run guest operating systems as *user-mode* processes. This means that the computations issued by the virtual system get directly executed by our CPU without emulation.

Issues:
What happens when the guest OS tries to issue a privileged instruction (like HALT?)
In practice, we run guest operating systems as *user-mode* processes. This means that the computations issued by the virtual system get directly executed by our CPU without emulation.

**Issues:**
What happens when the guest OS tries to issue a privileged instruction (like HALT?)
- The user process will *trap* into the hypervisor, which simulates the correct behavior.
In practice, we run guest operating systems as *user-mode* processes. This means that the computations issued by the virtual system get directly executed by our CPU without emulation.

**Issues:**

What happens when the guest OS tries to issue a privileged instruction (like HALT?)

- The user process will *trap* into the hypervisor, which simulates the correct behavior.

What happens when a program running on the virtual machine issues a privileged instruction that requires kernel intervention (like a system call?)
Running Virtual Machines

In practice, we run guest operating systems as *user-mode* processes. This means that the computations issued by the virtual system get directly executed by our CPU without emulation.

**Issues:**

What happens when the guest OS tries to issue a privileged instruction (like HALT?)
- The user process will *trap* into the hypervisor, which simulates the correct behavior.

What happens when a program running on the virtual machine issues a privileged instruction that requires kernel intervention (like a system call?)
- Issue -> it needs to look like the system call is being handled by the guest OS, but in reality the hypervisor is handling it.
Running Virtual Machines

In practice, we run guest operating systems as *user-mode* processes. This means that the computations issued by the virtual system get directly executed by our CPU without emulation.

Issues:
What happens when the guest OS tries to issue a privileged instruction (like HALT?)
- The user process will *trap* into the hypervisor, which simulates the correct behavior.

What happens when a program running on the virtual machine issues a privileged instruction that requires kernel intervention (like a system call?)
- Issue -> it needs to look like the system call is being handled by the guest OS, but in reality the hypervisor is handling it.
- It is messy, but the system call traps into the hypervisor, which gives the guest OS the tools it needs to complete the syscall before letting it run. The guest OS traps to the hypervisor when it completes the syscall so that the hypervisor can return control to the user program.
Running Virtual Machines

Privileged instructions:
Running Virtual Machines

In practice, we run guest operating systems as *user-mode* processes. This means that the computations issued by the virtual system get directly executed by our CPU without emulation.

Issues:
- System calls are slow, and some, like **page faults** are incredibly slow due to VM / hypervisor switching and emulation.
Running Virtual Machines

In practice, we run guest operating systems as *user-mode* processes. This means that the computations issued by the virtual system get directly executed by our CPU without emulation.

Issues:
- System calls are slow, and some, like **page faults** are incredibly slow due to VM / hypervisor switching and emulation.
- Fix #1 -> **Paravirtualization**: Let the VM know it is a VM and provide special instructions that call directly into the hypervisor rather than needing to be perfectly simulated as user-level code. (This breaks the virtualization contract!)
Running Virtual Machines

In practice, we run guest operating systems as *user-mode* processes. This means that the computations issued by the virtual system get directly executed by our CPU without emulation.

Issues:
- System calls are slow, and some, like **page faults** are incredibly slow due to VM / hypervisor switching and emulation.
- Fix #1 -> **Paravirtualization**: Let the VM know it is a VM and provide special instructions that call directly into the hypervisor rather than needing to be perfectly simulated as user-level code. (This breaks the virtualization contract!)
- Fix #2 -> **Dynamic Binary Translation**: This approach requires at-runtime evaluation of the code being run. If any suspicious instructions are being executed, the translator inserts traps and dynamically rewrites the instructions to comply better with the underlying system.
Virtual (Virtual) Memory

Virtual Memory is another significant issue for virtual machines:

We now have 3 kinds of addresses: guest virtual addresses, guest physical addresses, and host physical addresses!
Virtual (Virtual) Memory

2 implementations for virtual memory:

1 -> Shadow Page Tables (SPT)
   - The hypervisor maintains mappings from guest virtual memory to host physical memory.
   - These were basically a second set of page tables just for the hypervisor to store and update.
   - Key drawback -> every time we enter the hypervisor, we take a serious performance hit. Doing this for every memory address translation is really really bad for performance.
Virtual (Virtual) Memory

2 implementations for virtual memory:

2 -> Extended Page Tables (EPT)
- Extended page tables also maintain mappings from guest virtual memory to host physical memory.
- The key difference between EPTs and SPTs is that the guest OS can interact with the EPT directly, meaning it does not need to call into the hypervisor (which is very slow).
  - As a finer note (not important), this is because computer hardware has become more supportive of virtualization, so it has hardware that facilitates structures like EPTs.
Virtual (Virtual) Memory

Why do we use VMs?
At one time, virtual memory system designers were advised to bias page replacement algorithms against modified pages in favor of those that have not been modified. The result of this suggestion was the unfortunate behavior of program code pages (which tend to be some of the only pages that are not modified) being kicked out of memory before other pages. Describe one reason as to why swapping out a read-only page might be beneficial.
Think-Pair-Share on Links

1. I create 1 **hard** link to “Minecraft.exe”. I then create 1 **soft** link to “Minecraft.exe.” How many new **inumbers** do I need to make these links?

2. Can you make a **soft** link to a **hard** link?
   a. Can you make a **soft** link to a **soft** link?
   b. Can you make a **hard** link to a **soft** link?

3. If I make a soft link and then *change its location* (i.e. put it in another directory), what will happen? Will it still work?
Concept Check Time! (Think Pair Share)

Spend some time thinking about these by yourselves, then discuss with your table. Finally, we’ll reconvene to discuss! **Recall that for performance-related questions, disk seeks are the most important metric!**

1. Assuming I have *lots* of disk space (disk space >> files allocated; no fragmentation), how would Linked Files perform vs. Contiguous Allocation *(performance in terms of # of disk seeks)*

2. Now assume I have a file system with somewhat limited disk space (fragmentation could be a problem) and *very limited* memory (RAM). Between Contiguous Alloc, Linked Files, and Windows FAT, which would you recommend and why?

3. For a random access (i.e. any byte of any file), rank the 4 approaches in terms of performance (consider disk seeks first, then memory). Ties are allowed if justified.
If we have time: Exam Review