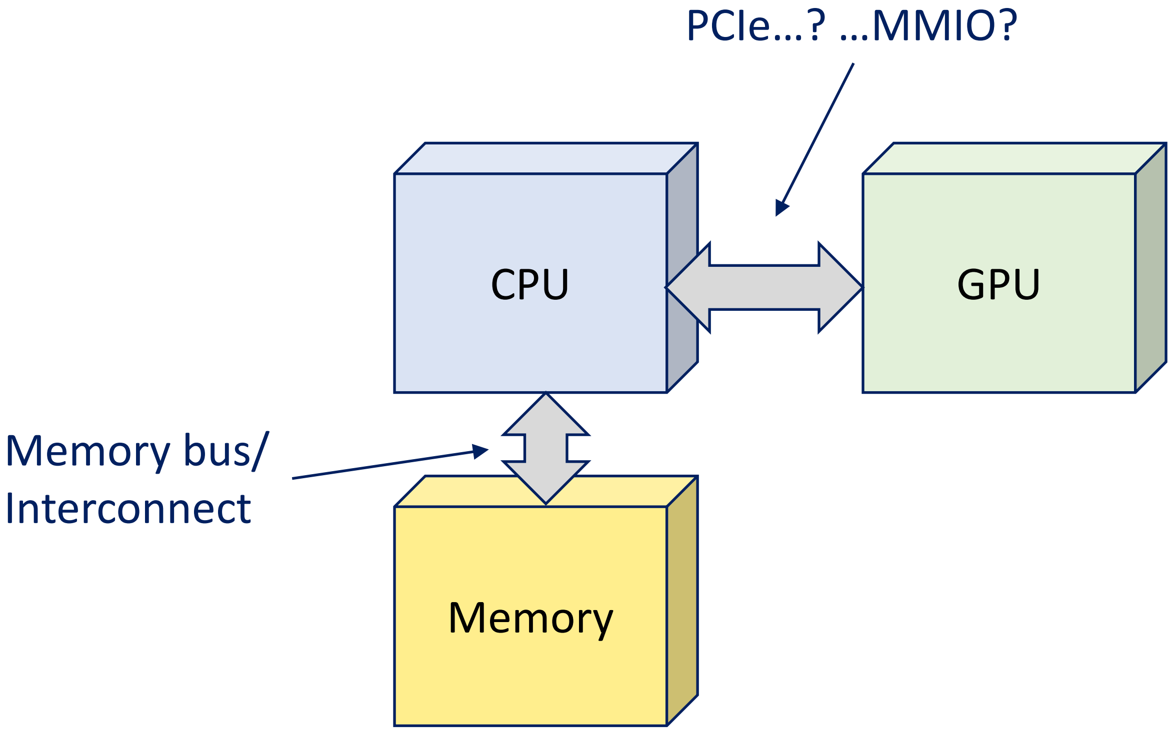
CS152: Computer Systems Architecture
System Bus Architecture

Sang-Woo Jun

Winter 2021

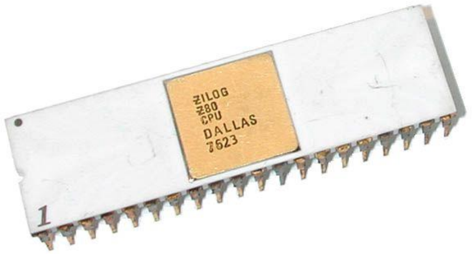
Large amount of material adapted from MIT 6.004, “Computation Structures”,
Morgan Kaufmann “Computer Organization and Design: The Hardware/Software Interface: RISC-V Edition”,
and CS 152 Slides by Isaac Scherson

Covered computer architecture so far

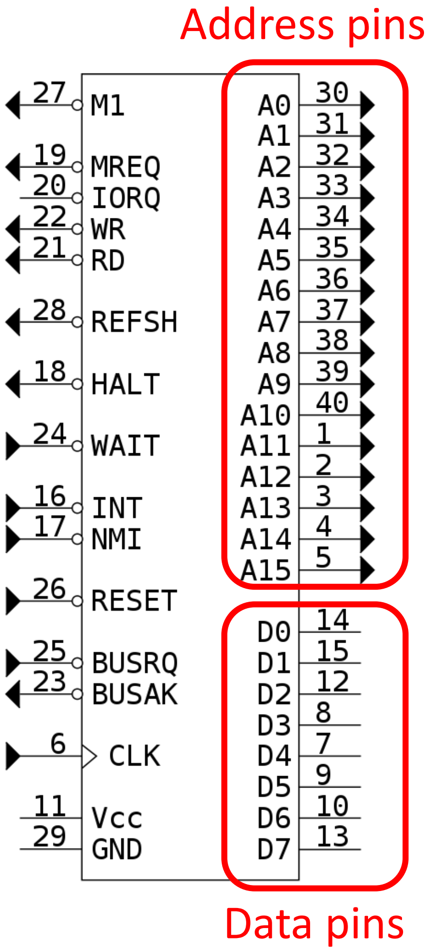


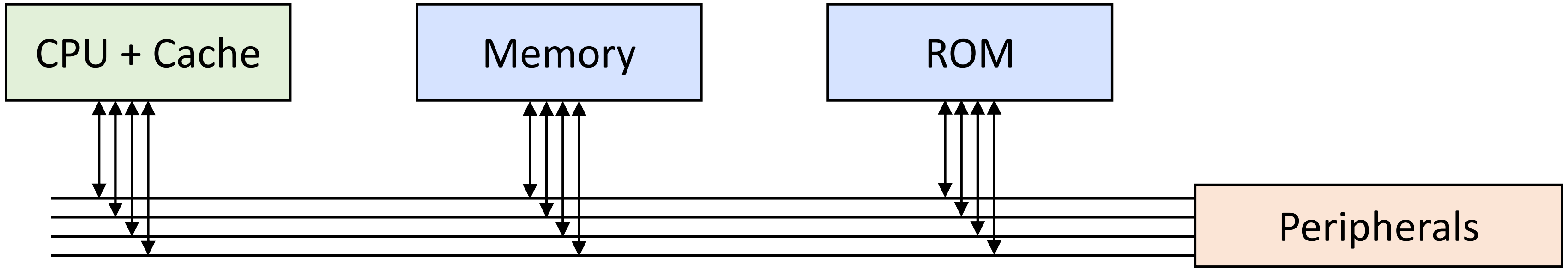
“A very early version of the Zilog Z80,”Damicatz, Wikimedia commons

At a high level: The system bus



- A “system bus” connects cpu, memory, and I/O
- Historically, this used a be an actual bus
  - Bundle of shared wires!
  - Still used in embedded systems, (I2C, SPI, …)
  - “Slaves” (not CPU) snoop the address pins, and respond when address is directed to itself
  - Cooperation/Agreement critical!

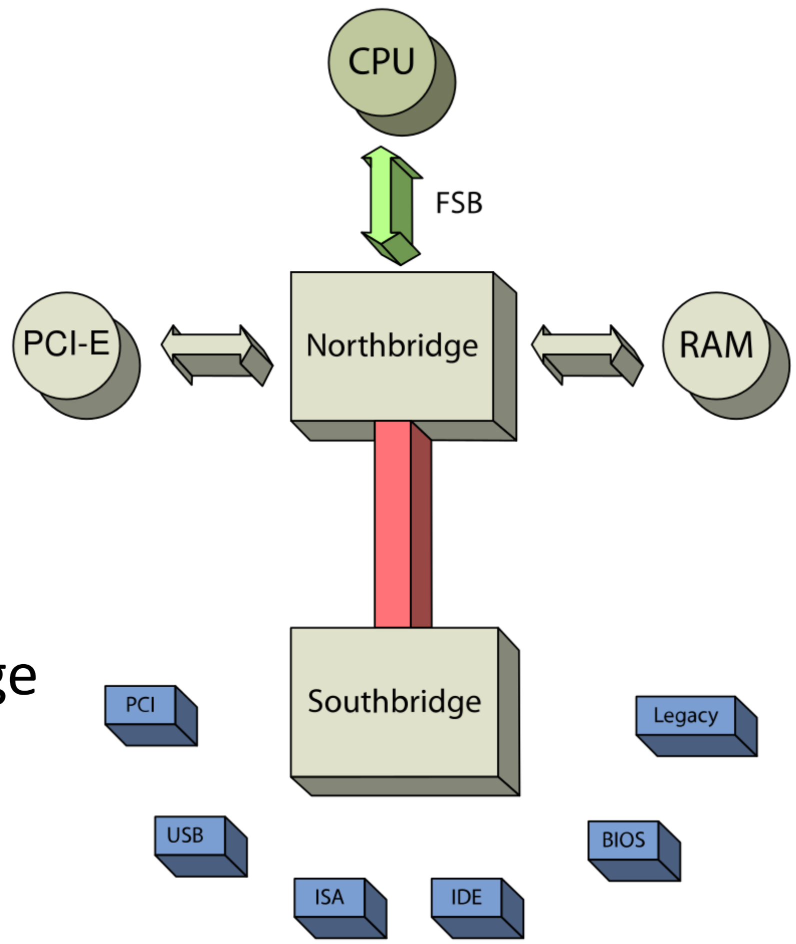




“The Z80's original DIP40 chip package pinout,” Sakurambo, Wikimedia commons

Modern system busses are multi-tiered

- Conceptually divided into two clusters
  - Fast devices connected via “North bridge”
    - Memory, PCIe, …
  - Slow devices connected via “South bridge”
    - SATA, USB, Keyboard, …
  - Simplifies design, saves resources
    - Keyboard doesn’t need as much bandwidth as memory!
- Originally used to be two separate chips
  - North bridge is now often integrated into CPU package



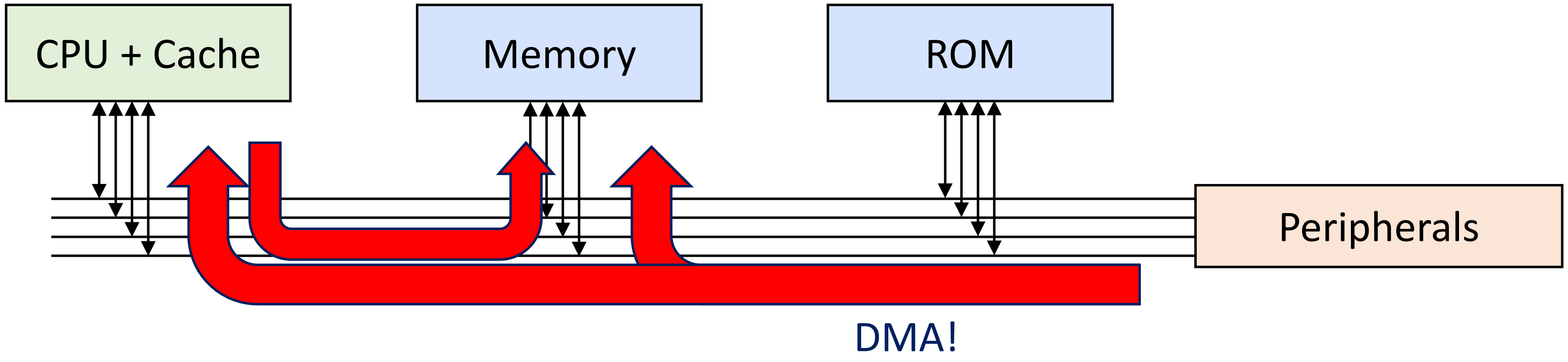
“A typical north/southbridge layout,”Fred the Oyster, Wikimedia commons

Communicating with peripherals

- From the processor perspective, interface has not changed much
- Default operation is still memory-mapped I/O
  - CPU writes to a special address region
  - Memory requests get translated to requests to peripheral device
  - Device responses get translated to memory responses
- MMIO not treated specially by CPU
  - Except, mapped region is not cacheable
  - E.g., If peripheral omits a read response, CPU hangs
  - BIG problem: Peripheral access is SLOW!
    - LW instruction waiting forever… We should be doing something else while we wait

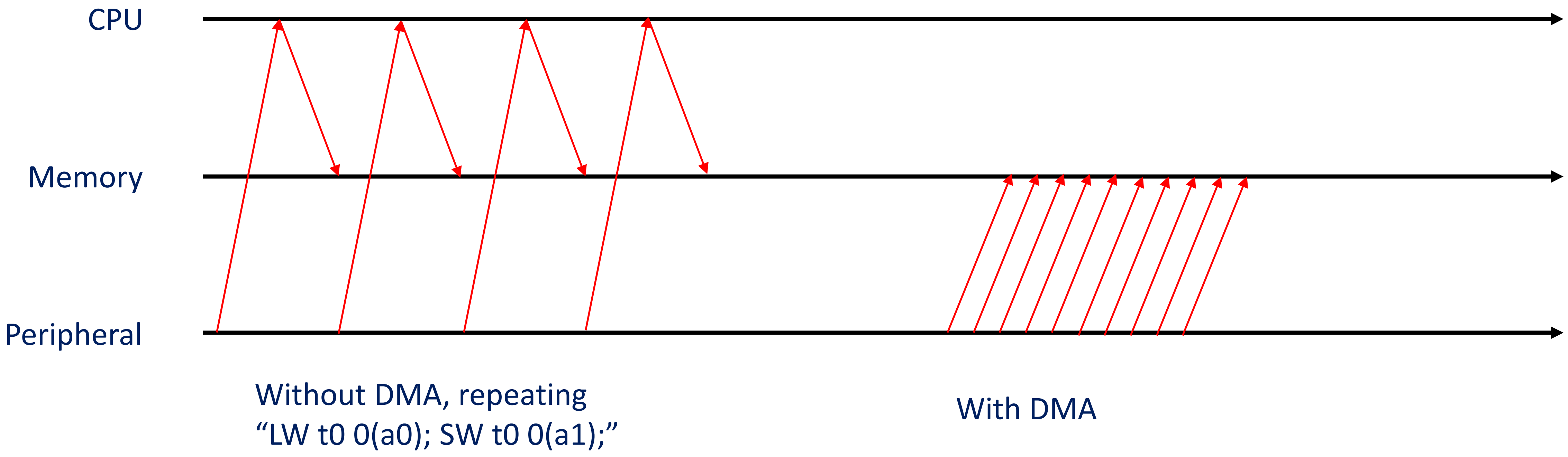
Introducing Direct Memory Access (DMA)

- To solve the problem of high-latency, synchronous peripheral access
- The CPU delegates memory access
  - Either to peripheral device, or to a separate “DMA controller”
  - Copying 4 KB from disk to memory no longer requires 4K+ CPU instructions
  - CPU asks disk to initiate DMA, and can move on to other things



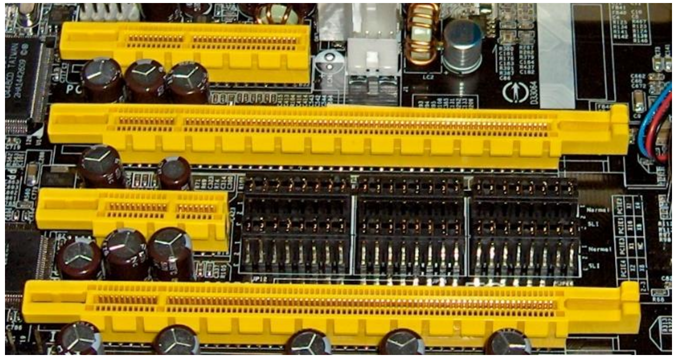
Introducing Direct Memory Access (DMA)

- High performance with DMA, by overlapping high-latency access



Peripheral Component Interconnect Express

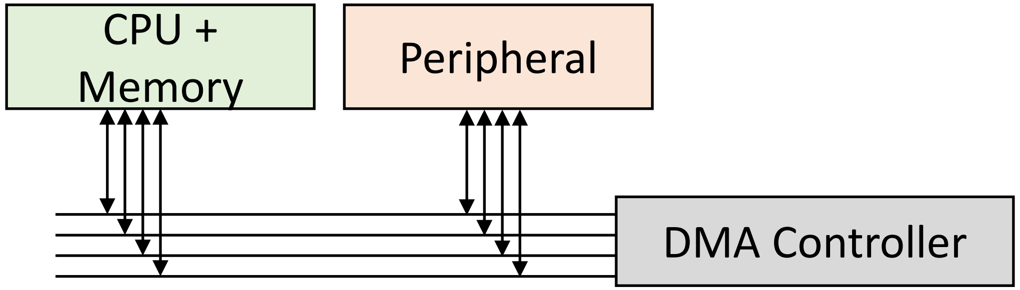
- Newest in a long line of expansion bus standards
  - ISA, AGP, PCI, …
- PCIe is currently de-facto standard for high-performance peripherals
  - GPUs, NVMe storage, Ethernet, …
  - Classified into “Generations”, organized into multiple “Lanes”
    - E.g., Single Gen 3 lane capable of ~1 GB/s, 16 lane device capable of ~16 GB/s
    - Currently migrating into ~2 GB/s/lane Gen 4 and ~4 GB/s/lane Gen 5



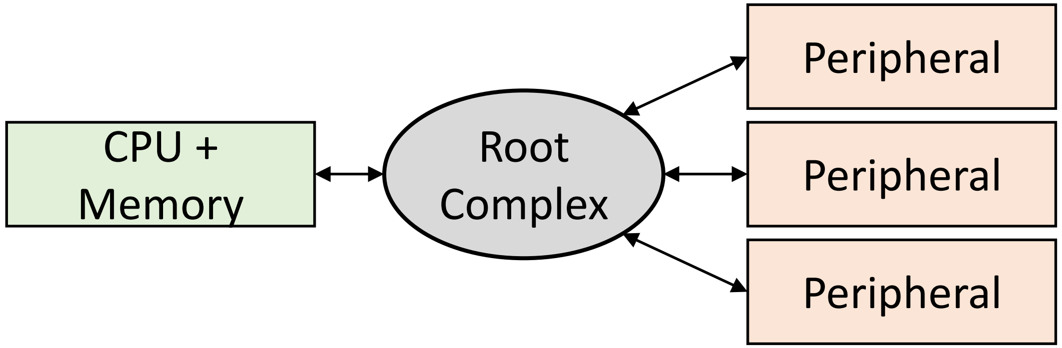
“Various slots on a computer motherboard,” w:user:snickerdo , Wikimedia commons

PCIe "bus" is not a bus

- A true bus architecture saves silicon, but silicon is cheap now!
  - Moore's law…
- Despite the "bus" name, PCIe implements point-to-point connection
  - Multiple peripherals can transmit data at once
    - Subject to CPU-side bandwidth limitations
  - Also supports peer-to-peer communication
    - Doesn't eat into CPU-side bandwidth budget
    - Needs agreement and support from both devices
    - E.g., Ethernet to storage, GPU to GPU, …



Vs.



CS 152: Computer Systems Architecture
Storage Technologies

Sang-Woo Jun

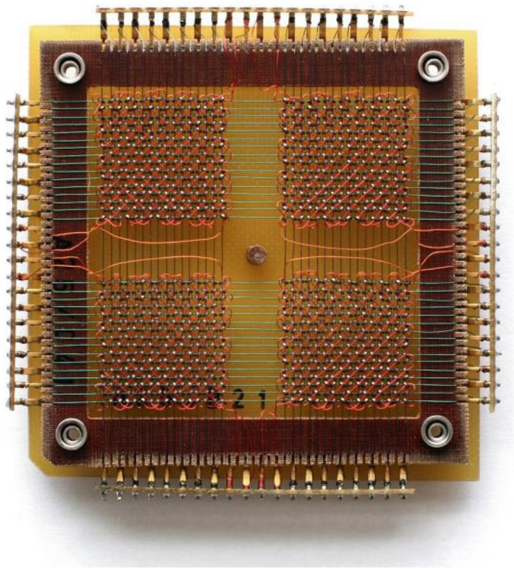
Winter 2021

UCI

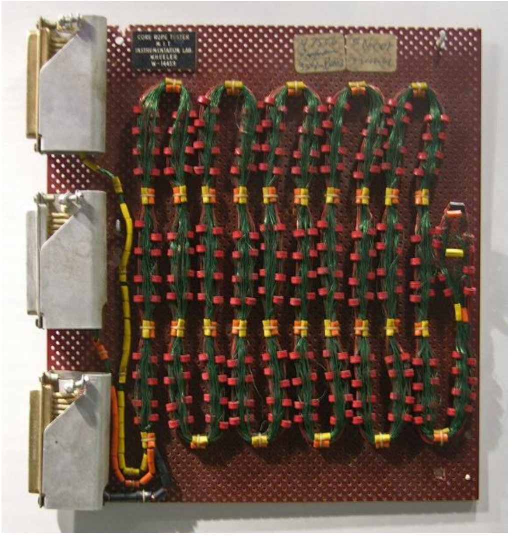
Storage Used To be a Secondary Concern

- Typically, storage was not a first order citizen of a computer system
  - As alluded to by its name “secondary storage”
  - Its job was to load programs and data to memory, and disappear
  - Most applications only worked with CPU and system memory (DRAM)
  - Extreme applications like DBMSs were the exception
- Because conventional secondary storage was very slow
  - Things are changing!

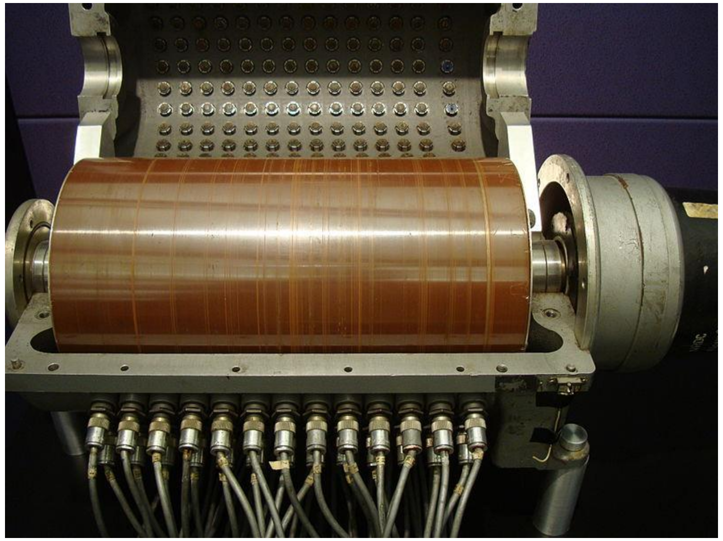
Some (Pre)History



Magnetic core memory
1950~1970s
(1024 bits in photo)



Rope memory (ROM) 1960's
72 KiB per cubic foot!
Hand-woven to program the
Apollo guidance computer



Drum memory
100s of KiB
1950's

Photos from Wikipedia

Some (More Recent) History

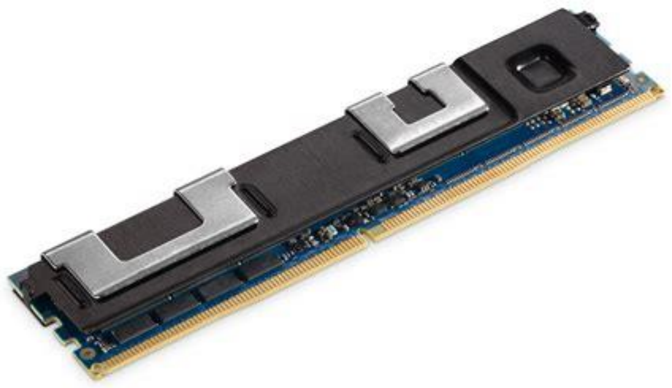
Floppy disk drives
1970's~2000's
100 KiBs to 1.44 MiB

Hard disk drives
1950's to present
MBs to TBs

Photos from Wikipedia

Some (Current) History

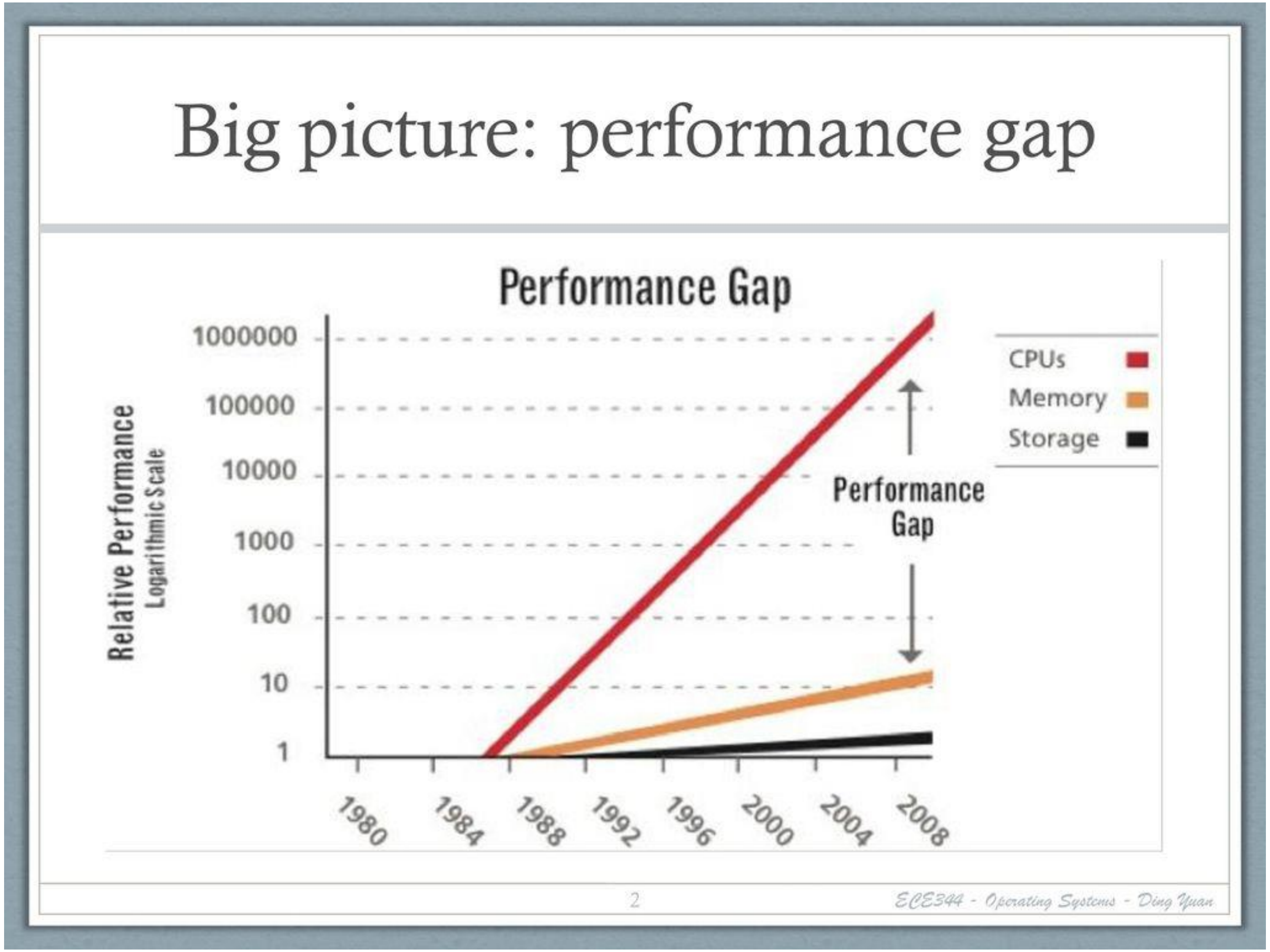
Solid State Drives
2000's to present
GB to TBs



Non-Volatile Memory
2010's to present
GBs

Hard Disk Drives

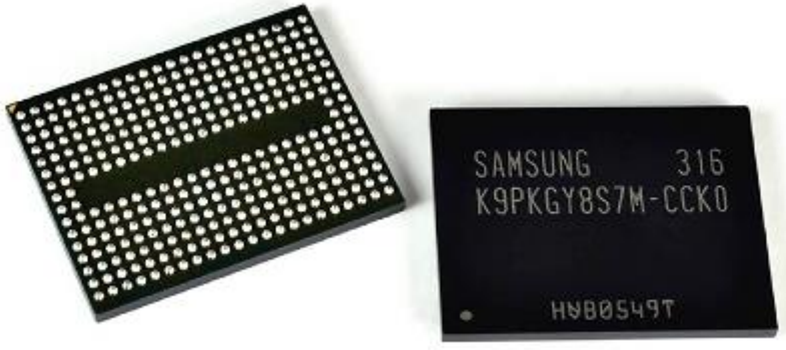
- Dominant storage medium for the longest time
  - Still the largest capacity share
- Data organized into multiple magnetic platters
  - Mechanical head needs to move to where data is, to read it
  - Good sequential access, terrible random access
    - 100s of MB/s sequential, maybe 1 MB/s 4 KB random
  - Time for the head to move to the right location (“seek time”) may be ms long
    - 1000,000s of cycles!
- Typically “ATA” (Including IDE and EIDE), and later “SATA” interfaces
  - Connected via “South bridge” chipset

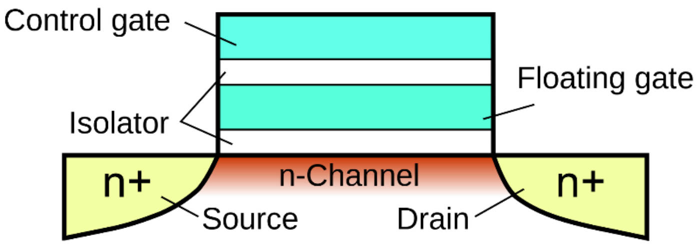


Ding Yuan, “Operating Systems ECE344 Lecture 11: File System”

Solid State Drives

- "Solid state", meaning no mechanical parts, addressed much like DRAM
  - Relatively low latency compared to HDDs (10s of us, compared to ms)
  - Easily parallelizable using more chips – Multi-GB/s
- Simple explanation: flash cells store state in a "floating gate" by charging it at a high voltage
  - High voltage acquired via internal charge pump (no need for high V input)



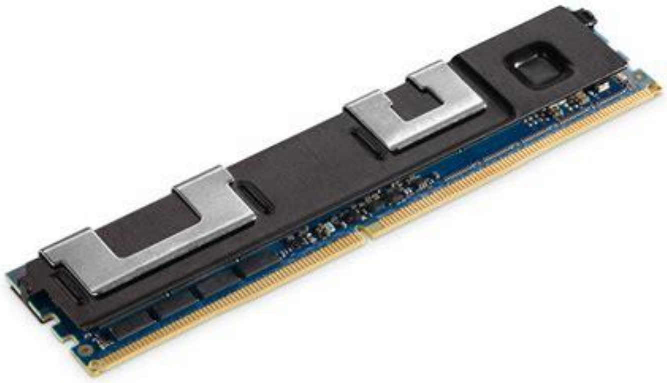


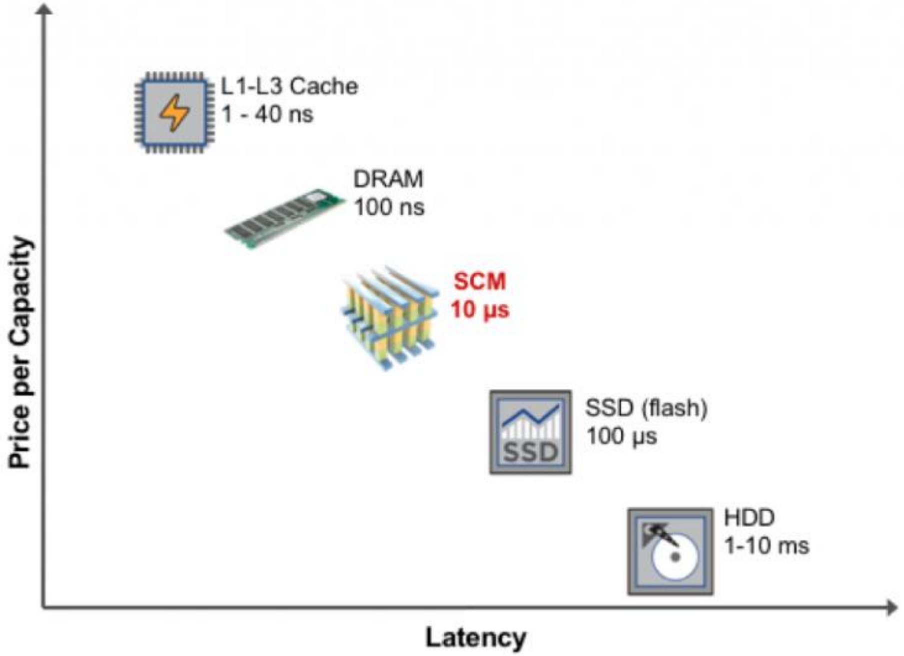
Solid State Drives

- Serial ATA (SATA) interface, over Advanced Host Controller Interface (AHCI) standard
  - Used to be connected to south bridge,
  - Up to 600 MB/s, quickly became too slow for SSDs
- Non-Volatile Memory Express (NVMe)
  - PCIe-attached storage devices – multi-GB/s
  - Redesigns many storage support components in the OS for performance

Non-Volatile Memory

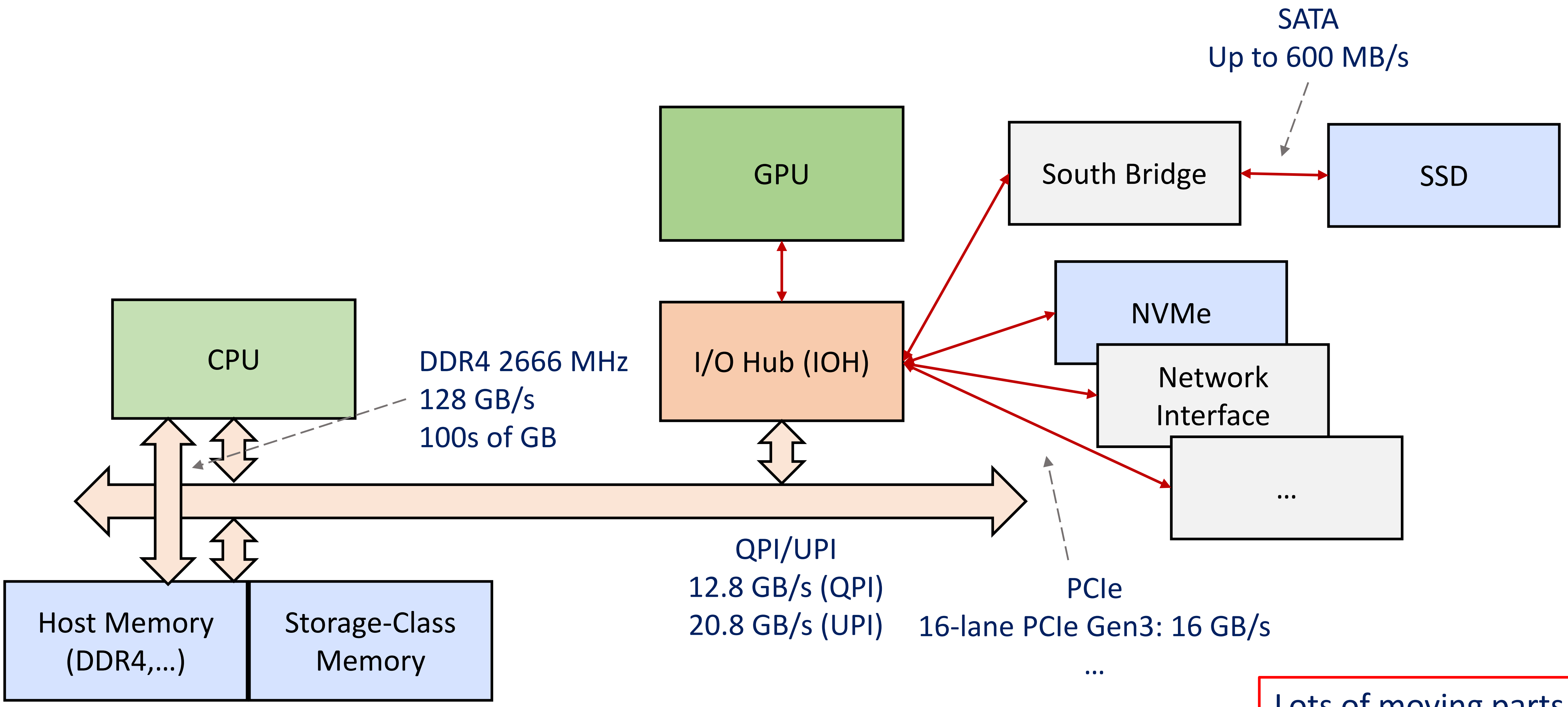
- Naming convention is a bit vague
  - Flash storage is also often called NVM
    - Storage-Class Memory (SCM)?
  - Anything that is non-volatile and fast?
- Too fast for even PCIe/NVMe software
  - Plugged into memory slots, accessed like memory
- But not quite as fast as DRAM
  - Latency/Bandwidth/Access granularity
  - Usage under active research!





Souce: NetApp blog, “Storage Class Memory: What’s Next in Enterprise Storage,” 2018

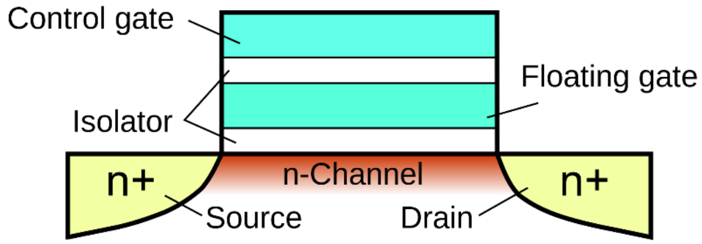
System Architecture Snapshot (2021)



Lots of moving parts!

Flash Storage

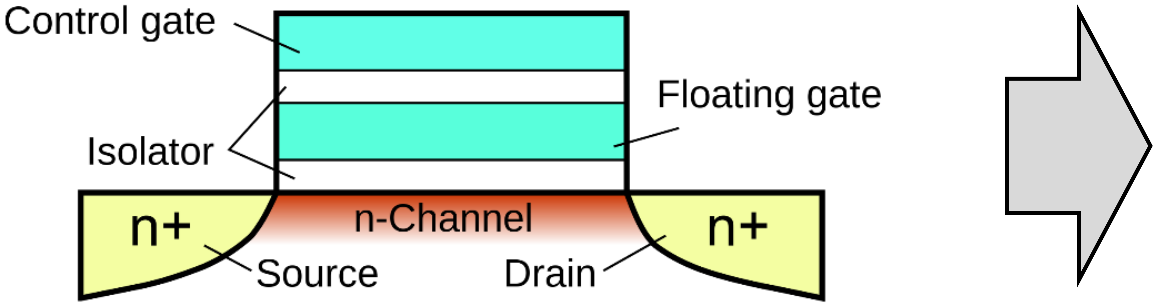
- Most prominent solid state storage technology
  - Few other technologies available at scale (Intel X-Point one of few examples)
- Flash cells store data in “floating gate” by charging it at high voltage*
- Cells configured into NOR-flash or NAND-flash types
  - NOR-flash is byte-addressable, but costly
  - NAND-flash is “page” addressable, but cheap
- Many bits can be stored in a cell by differentiating between the amount of charge in the cell
  - Single-Level Cell (SLC), Multi (MLC), Triple (TLC), Quad (QLC)
  - Typically cheaper, but slower with more bits per cell

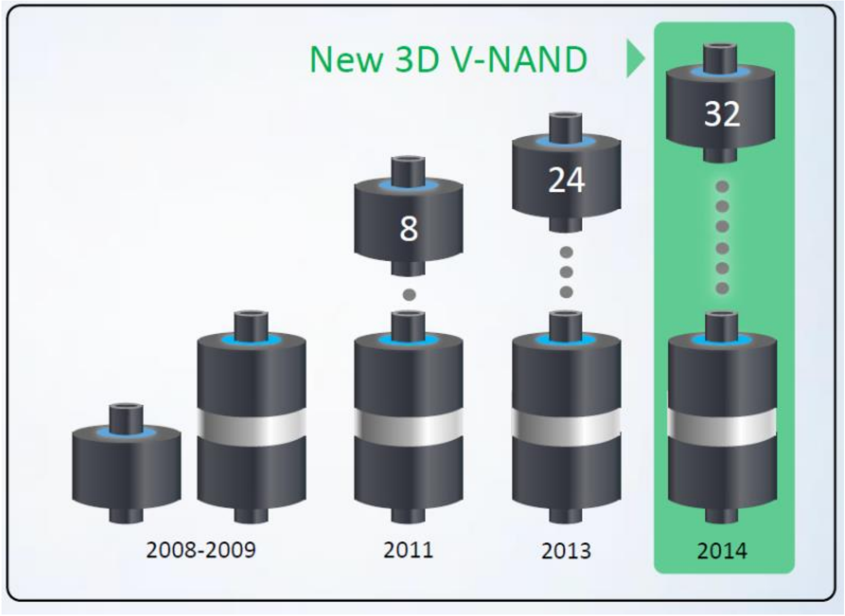


*Variations exist, but basic idea is similar

3D NAND-Flash

- NAND-Flash scaling limited by charge capacity in a floating gate
  - Only a few hundred can fit at current sizes
  - Can’t afford to leak even a few electrons!
- Solution: 3D stacked structure… For now!



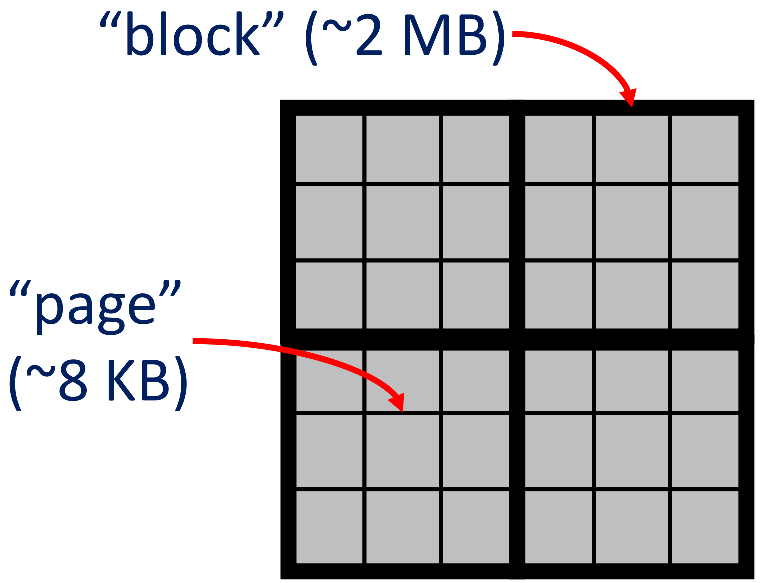


NAND-Flash Fabric Characteristics

- Read/write in “page” granularity
  - 4/8/16 KiB according to technology
  - Corresponds to disk “sector” (typically 4 KiB)
  - Read takes 10s of us to 100s of us depending on tech
  - Writes are slower, takes 100s of us depending on tech
- A third action, “erase”
  - A page can only be written to, after it is erased
  - Under the hood: erase sets all bits to 1, write can only change some to 0
  - **Problem :** Erase has very high latency, typically ms
  - **Problem :** Each cell has limited program/erase lifetime (thousands, for modern devices) – Cells become slowly less reliable

NAND-Flash Fabric Characteristics

- Performance impact of high-latency erase mitigated using large erase units (“blocks”)
  - Hundreds of pages erased at once
- What these mean: in-place updates are no longer feasible
  - In-place write requires whole block to be re-written
  - Hot pages will wear out very quickly
- People would not use flash if it required too much special handling

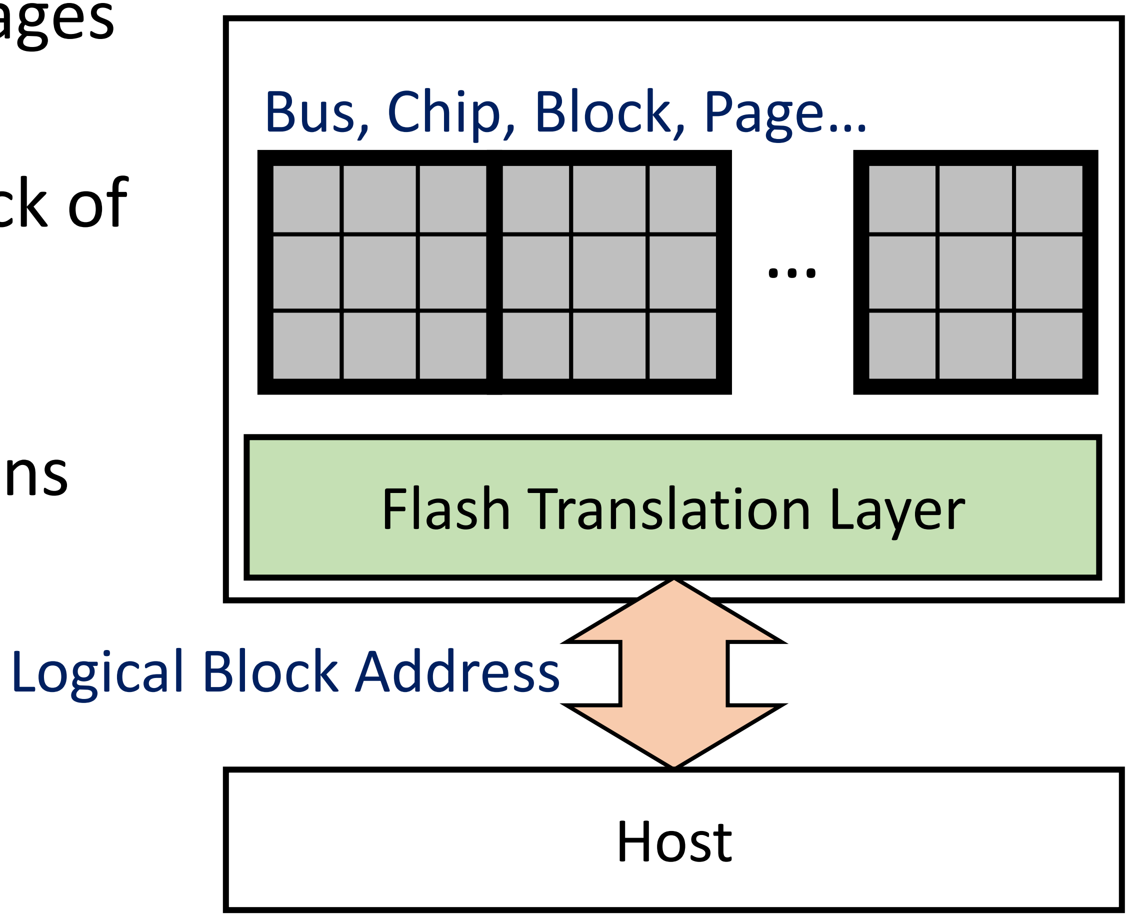


NAND-Flash SSD Architecture

- High bandwidth achieved by stringing organizing many flash chips into many busses
  - Enough chips on a bus to saturate bus bandwidth
  - More busses to get more bandwidth
- Many dimensions of addressing!
  - Bus, chip, block, page

The Solution: Flash Translation Layer (FTL)

- Exposes a logical, linear address of pages to the host
- A “Flash Translation Layer” keeps track of actual physical locations of pages and performs translation
- Transparently performs many functions for performance/durability



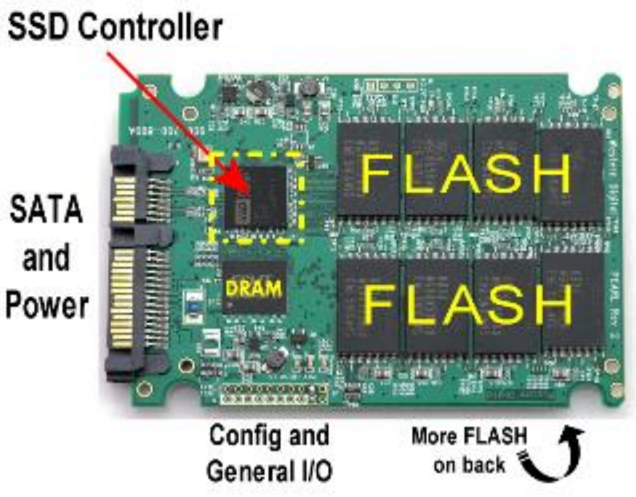
Some Jobs of the Flash Translation Layer

- Logical-to-physical mapping
- Bad block management
- Wear leveling: Assign writes to pages that have less wear
- Error correction: Each page physically has a few more bits for error codes
  - Reed-Solomon, BCH, LDPC, …
- Deduplication: Logically map pages with same data to same physical page
- Garbage collection: Clear stale data and compact pages to fewer blocks
- Write-ahead logging: Improve burst write performance
- Caching, prefetching,…

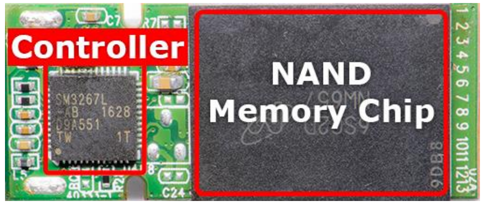
That's a Lot of Work for an Embedded System!

- ❑ Needs to maintain multi-GB/s bandwidth
- ❑ Typical desktop SSDs have multicore ARM processors and gigabytes of memory to run the FTL
  - o FTLs on smaller devices have sacrifice various functionality

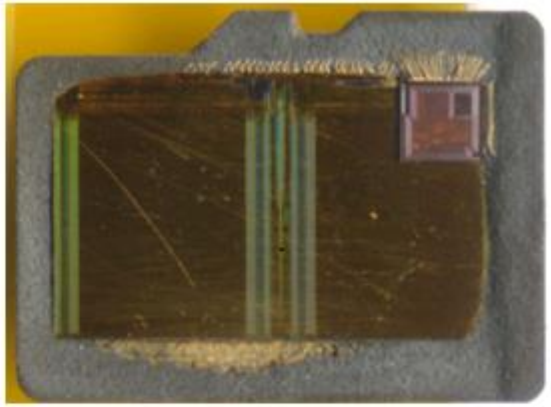
SATA SSD



USB Thumbdrive



MicroSD



Thomas Rent, "SSD Controller," storagereview.com
Jeremy, "How Flash Drives Fail," recovermyflashdrive.com
Andrew Huang, "On Hacking MicroSD Cards," bunniestudios.com

Some FTL Variations

- Page level mapping vs. Block level mapping
  - 1 TB SSD with 8 KB blocks need 1 GB mapping table
  - But typically better performance/lifetime with finer mapping
- Wear leveling granularity
  - Honest priority queue is too much overhead
  - Many shortcuts, including group based, hot-cold, etc
- FPGA/ASIC acceleration
- Open-channel SSD – No FTL
  - Leaves it to the host to make intelligent, high-level decisions
  - Incurs host machine overhead

Managing Write Performance

- Write speed is slower than reads, especially if page needs to be erased
- Many techniques to mitigate write overhead
  - Write-ahead log on DRAM
  - Pre-erased pool of pages
  - For MLC/TLC/QLC, use some pages in “SLC mode” for faster write-ahead log – Need to be copied back later

Flash-Optimized File Systems

- Try to organize I/O to make it more efficient for flash storage (and FTL)
- Typically “Log-Structured” File Systems
  - Random writes are first written to a circular log, then written in large units
  - Often multiple logs for hot/cold data
  - Reading from log would have been very bad for disk (gather scattered data)
- JFFS , YAFFS, F2FS, NILFS, …

Storage in the Network

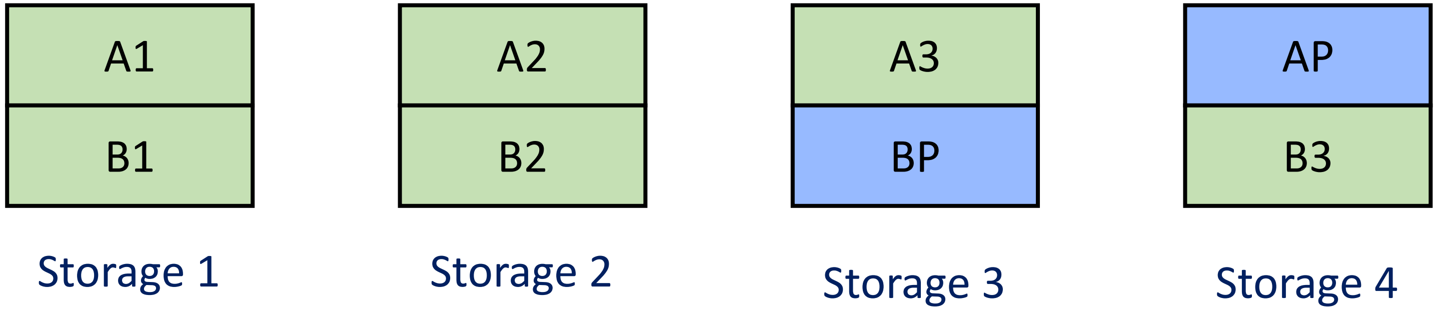
- Prepare for lightning rounds of very high-level concepts!

Redundant Array of Independent Disks (RAID)

- Technology of managing multiple storage devices
  - Typically in a single machine/array, due to limitations of fault-tolerance
- Multiple levels, depending on how to manage fault-tolerance
  - RAID 0 and RAID 5 most popular right now
- RAID 0: No fault tolerance, blocks striped across however many drives
  - Fastest performance
  - Drive failure results in data loss
  - Block size configurable
  - Similar in use cases to the Linux Logical Volume manager (LVM)

Fault-Tolerance in RAID 5

- RAID 5 stripes blocks across available storage, but also stores a parity block
  - Parity block calculated using xor (A1^A2^A3=AP)
  - One disk failure can be recovered by re-calculating parity
    - A1 = AP^A2^A3, etc
  - Two disk failure cannot be recovered
  - Slower writes, decreased effective capacity



Degraded Mode in RAID 5

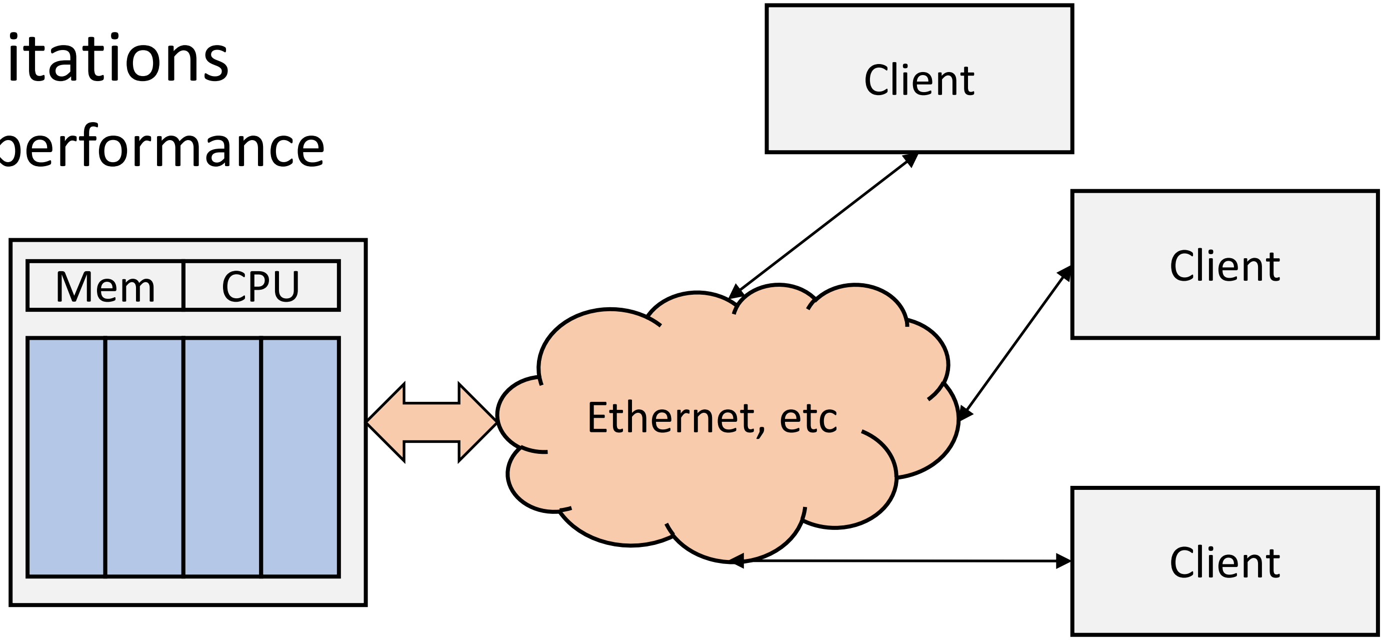
- In case of a disk failure it enters the “degraded mode”
  - Accesses from failed disk is served by reading all others and xor’ing them (slower performance)
- The failed disk must be replaced, and then “rebuilt”
  - All other storages are read start-to-finish, and parity calculated to recover the original data
  - With many disks, it takes long to read everything – “Declustering” to create multiple parity domains
  - Sometimes a “hot spare” disk is added to be idle, and quickly replace a failed device

Network-Attached Storage (NAS)

- Intuition: Server dedicated to serving files “File Server”
  - File-level abstraction
  - NAS device own the local RAID, File system, etc
  - Accessed via file system/network protocol like NFS (Network File System), or FTP
- Fixed functionality, using embedded systems with acceleration
  - Hardware packet processing, etc
- Regular Linux servers also configured to act as NAS
- Each NAS node is a separate entity – Larger storage cluster needs additional management

Network-Attached Storage (NAS)

- Easy to scale and manage compared to direct-attached storage
  - Buy a NAS box, plug it into an Ethernet port
  - Need more storage? Plug in more drives into the box
- Difficult to scale out of the centralized single node limit
- Single node performance limitations
  - Server performance, network performance



Storage-Area Networks (SAN)

- In the beginning: separate network just for storage traffic
  - Fibre Channel, etc, first created because Ethernet was too slow
  - Switch, hubs, and the usual infrastructure
- Easier to scale, manage by adding storage to the network
  - Performance distributed across many storage devices
- Block level access to individual storage nodes in the network
- Controversial opinion: Traditional separate SAN is dying out
  - Ethernet is unifying all networks in the datacenter
    - 10 GbE, 40 GbE slowly subsuming Fibre Channel, Infiniband, …

Converged Infrastructure

- Computation, Memory, Storage converged into a single unit, and replicated
- Became easier to manage compared to separate storage domains
  - Software became better (Distributed file systems, MapReduce, etc)
  - Decreased complexity – When a node dies, simply replace the whole thing
- Cost-effective by using commercial off-the-shelf parts (PCs)
  - Economy of scale
  - No special equipment (e.g., SAN)

Chris von Nieda, “How Does Google Work,” 2010

Hyper-Converged Infrastructure

- Still (relatively) homogenous units of compute, memory, storage
- Each unit is virtualized, disaggregated via software
  - E.g., storage is accessed as a pool as if on a SAN
  - Each unit can be scaled independently
  - A cloud VM can be configured to access an arbitrary amount of virtual storage
  - Example: vmware vSAN

Object Storage

- Instead of managing content-oblivious blocks, the file system manages objects with their own metadata
  - Instead of directory/file hierarchies, each object addressed via global identifier
  - Kind of like key-value stores, in fact, the difference is ill-defined
  - e.g., Lustre, Ceph object store
- An “Objest Storage Device” is storage hardware that exposes an object interface
  - Still mostly in research phases
  - High level semantics of storage available to the hardware controller for optimization