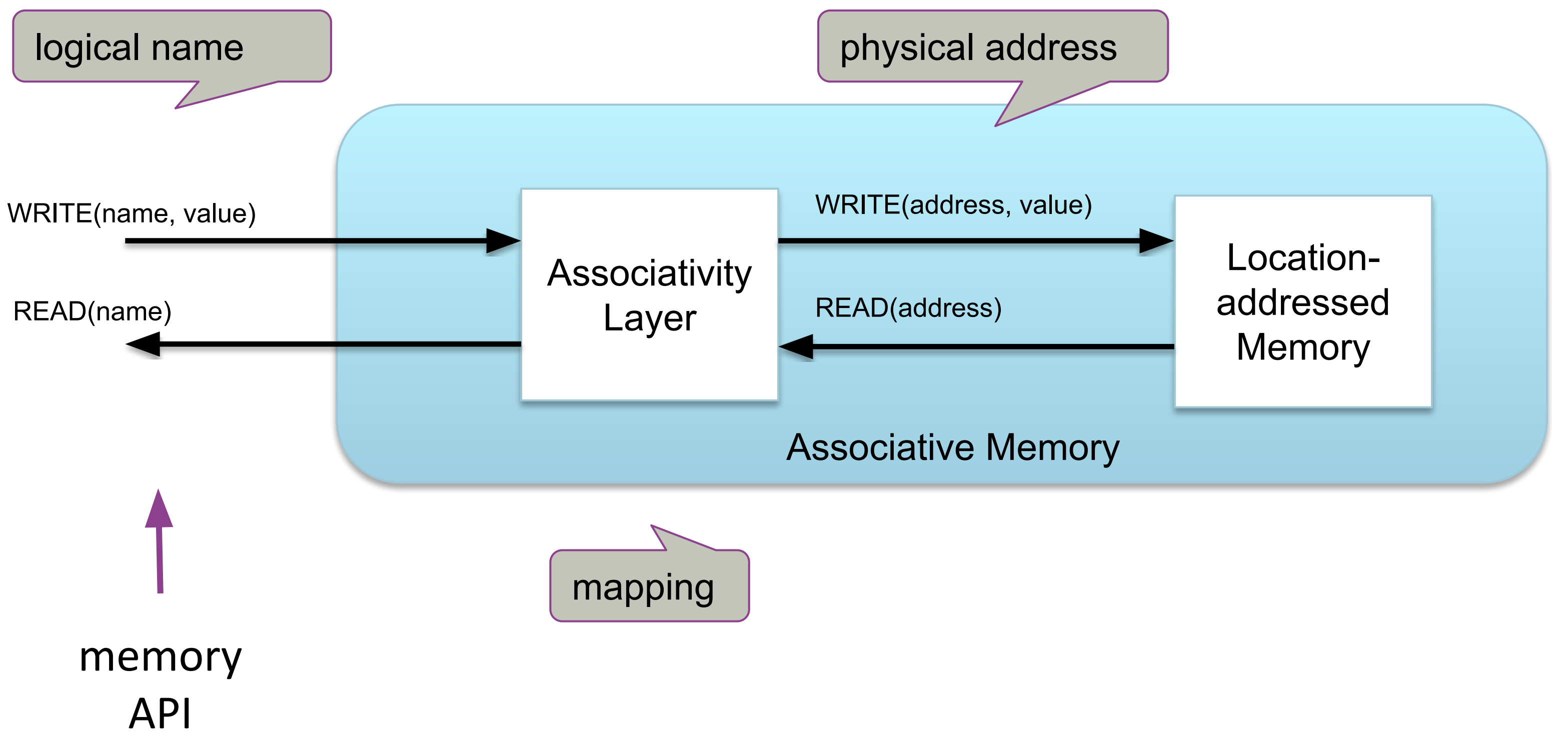
Operating Systems and C
Fall 2022
11. File

Memory Abstraction

recall fundamental abstractions

- interpreter — during lecture 4
- **memory** — **today (again)!**
- communication — at end of course

still a memory abstraction



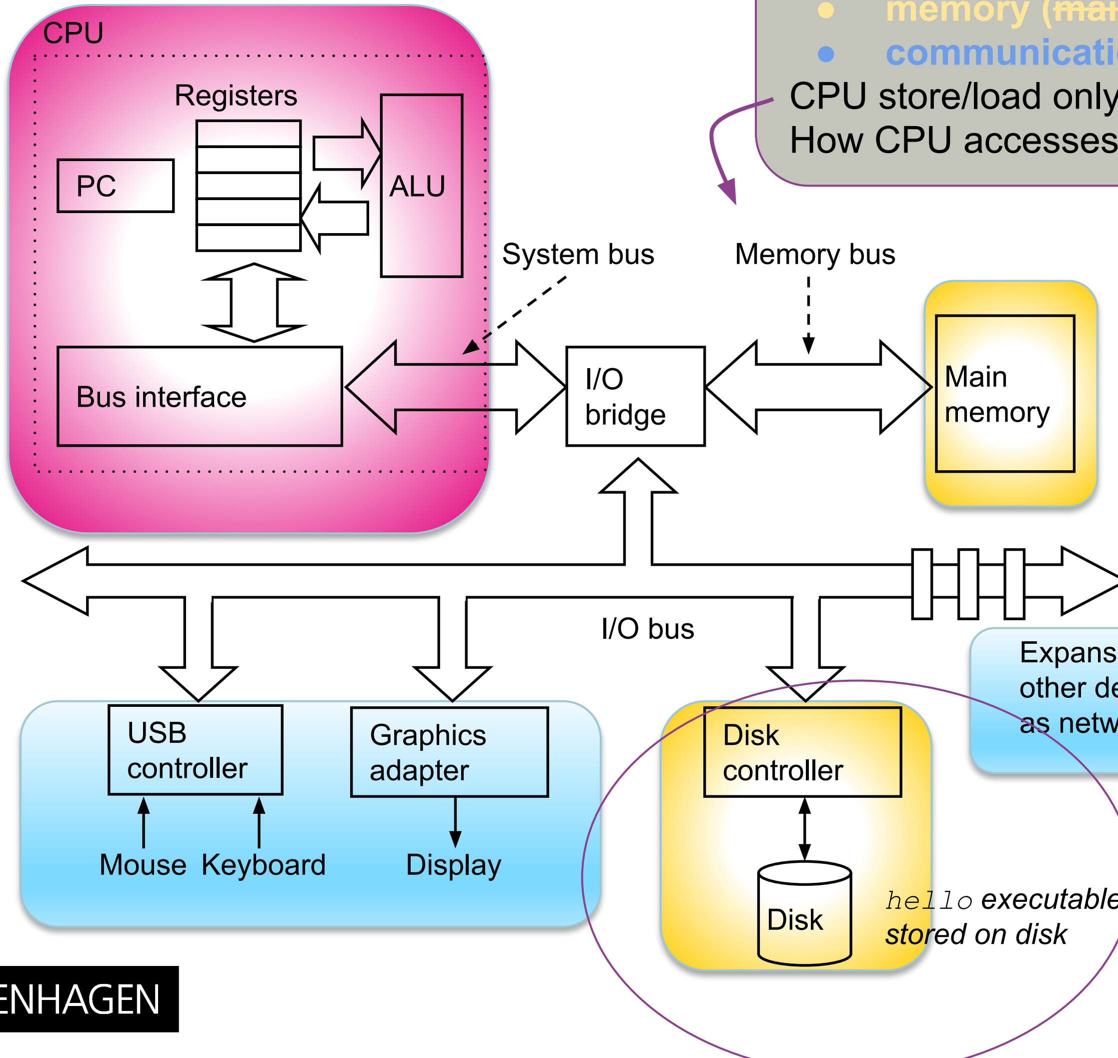
Source: Saltzer and Kaashoek

Computer Hardware

20s

recall hardware:
- ~~CPU~~
- memory (~~main memory~~, disk)
- communication

CPU store/load only via. Main memory.
How CPU accesses data on disk?

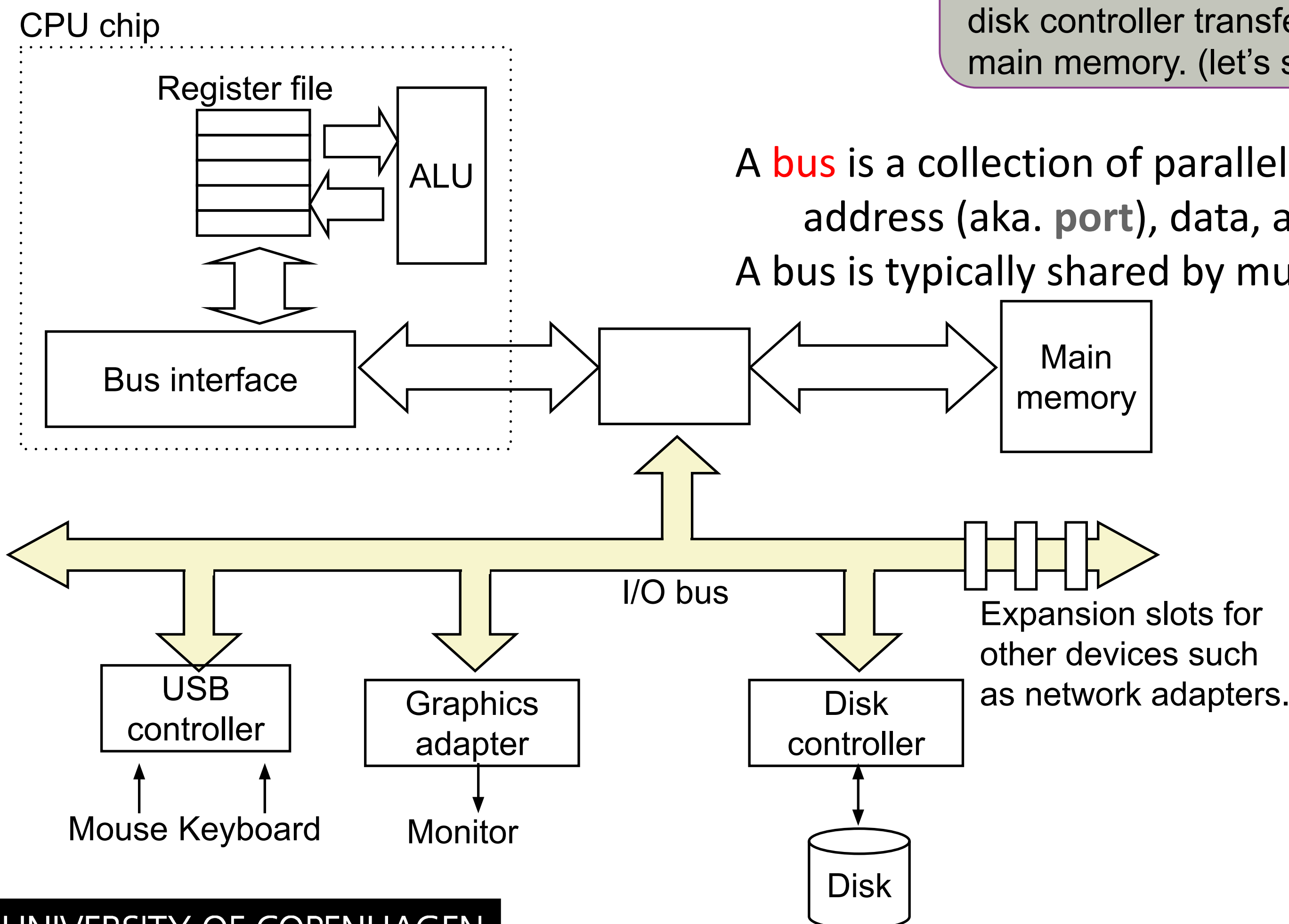


focus today

I/O Bus

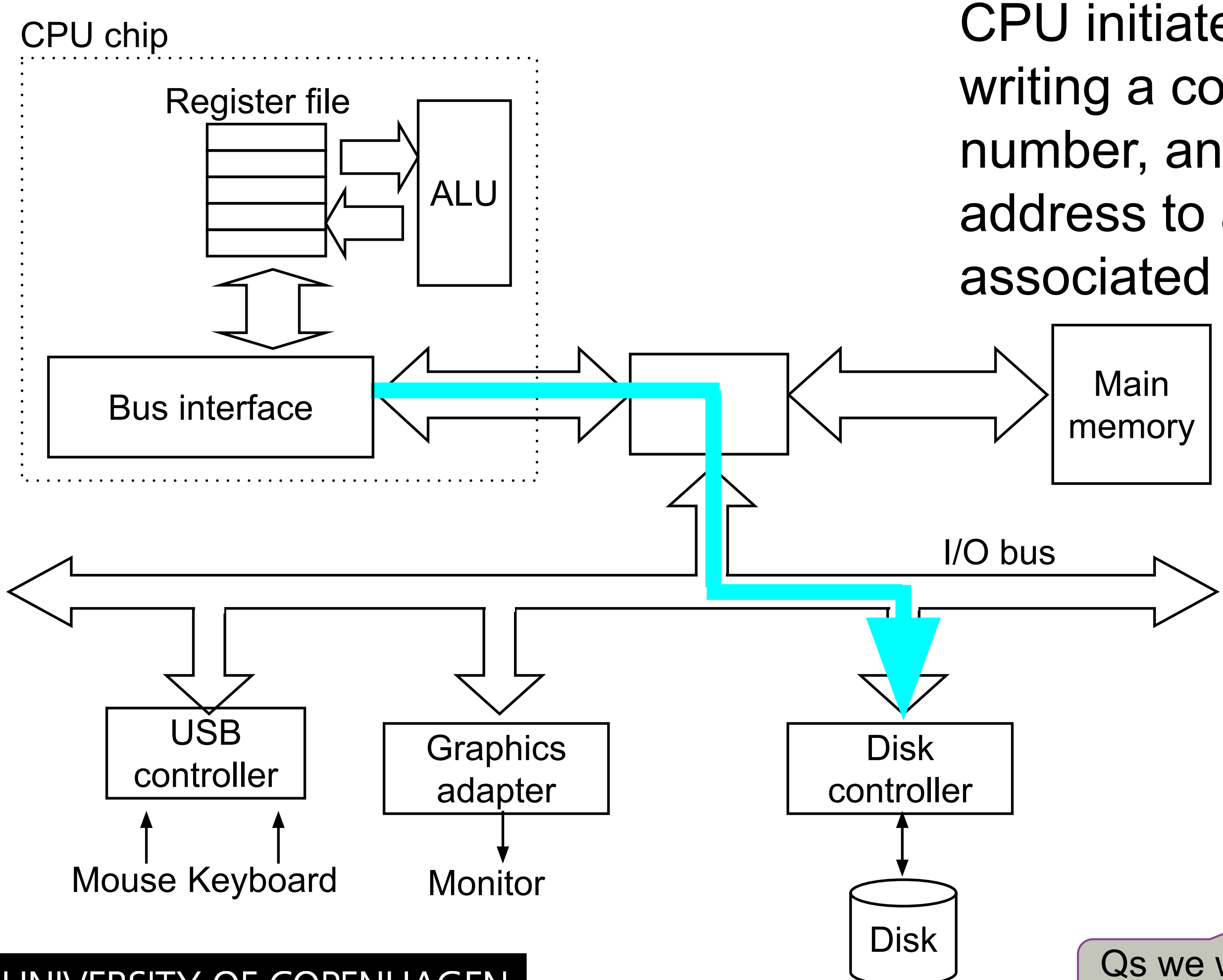
answer:
CPU sends instruction to disk controller.
(which implements **memory abstraction**).
disk controller transfers to (read) / from (write) main memory. (let's see this in action)

A bus is a collection of parallel wires that carry address (aka. **port**), data, and control signals.
A bus is typically shared by multiple devices.



Reading a Disk Sector (1)

10s



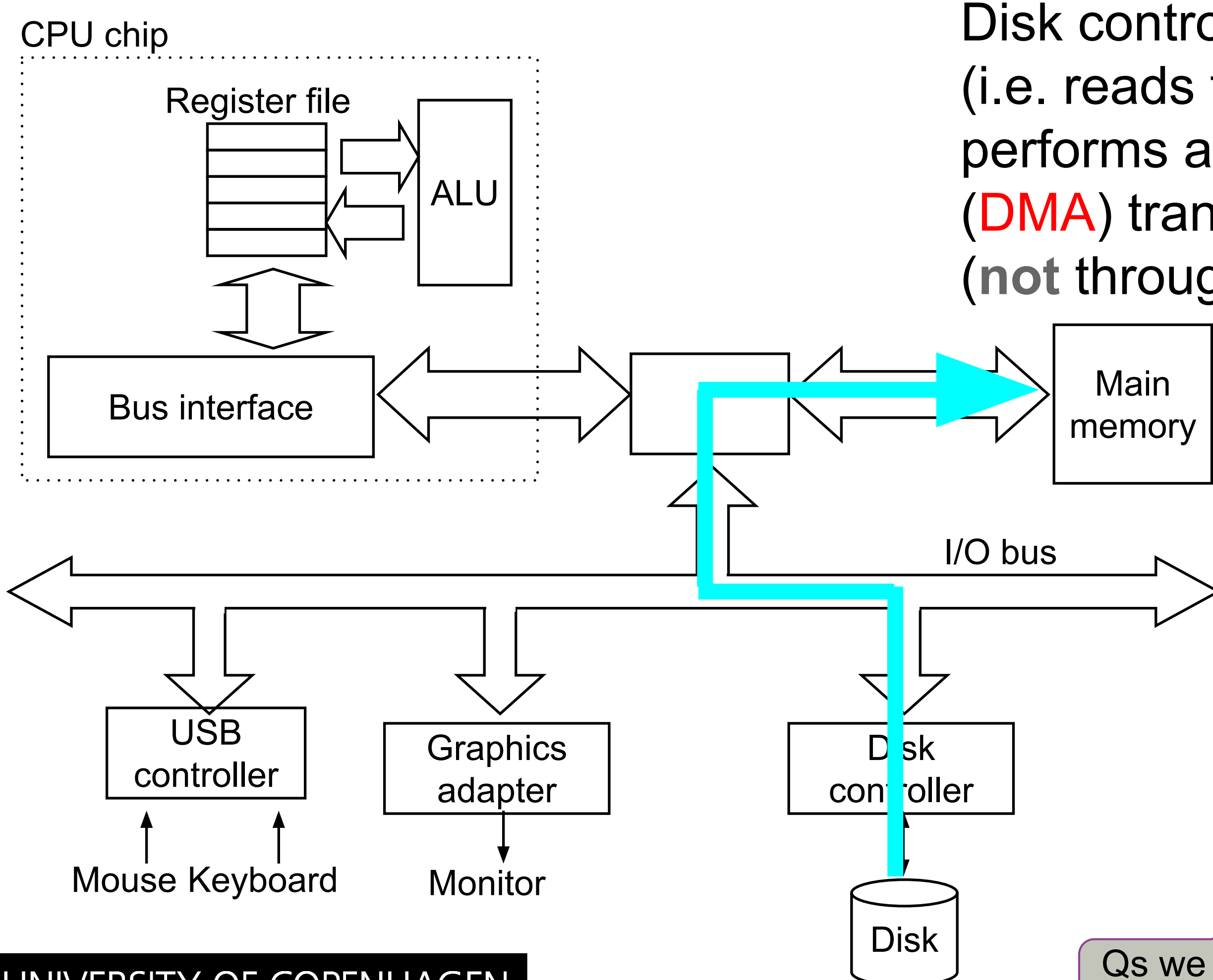
CPU initiates a disk read by writing a command, logical block number, and destination memory address to a port (address) associated with disk controller.

What does a disk controller do?
How do hosts interact with disk controllers?

Qs we will address

Reading a Disk Sector (2)

10s



Disk controller does its work (i.e. reads the sector) and performs a direct memory access (DMA) transfer into main memory. (**not** through the CPU)

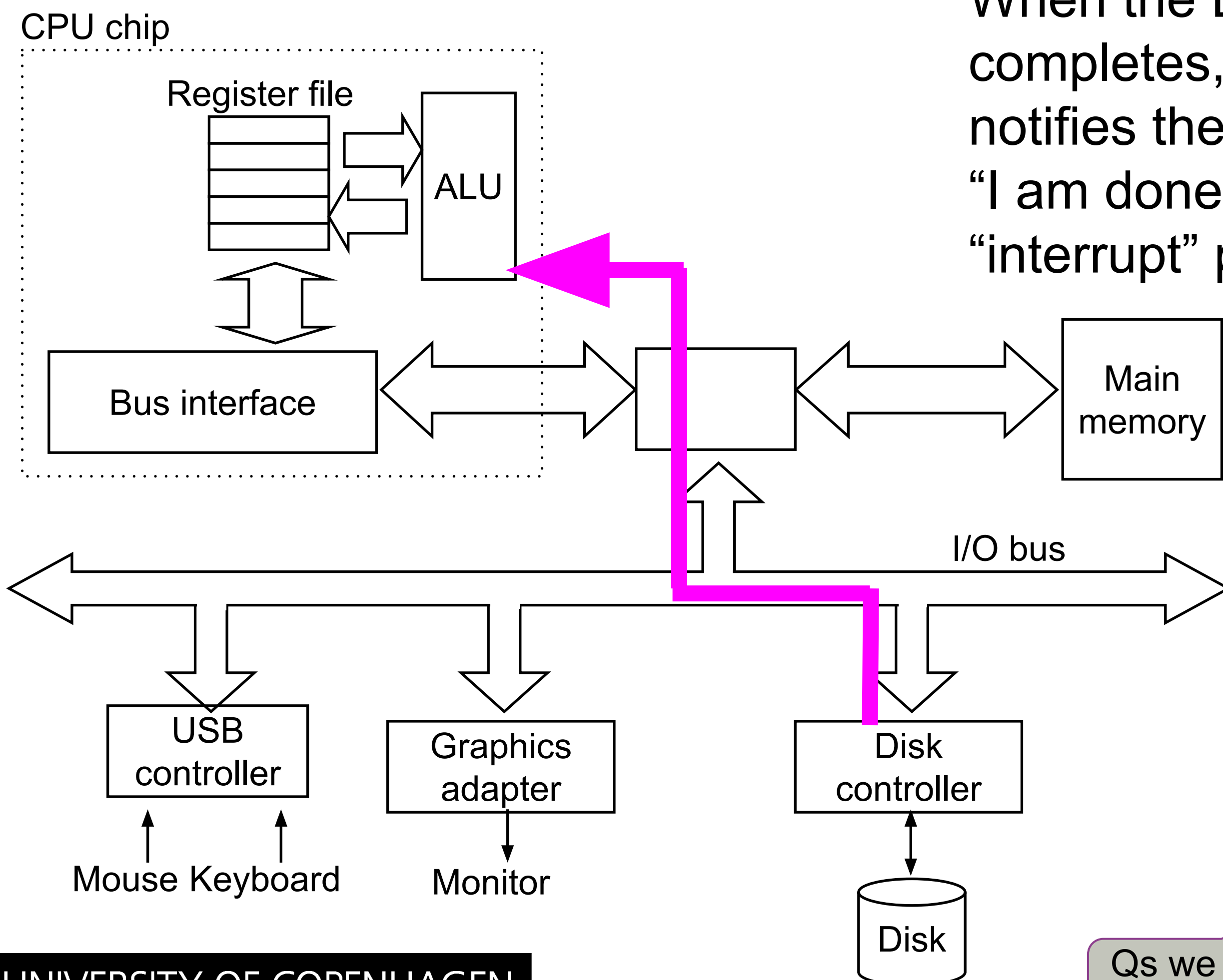
How does a disk controller perform a DMA?

Qs we will address

Reading a Disk Sector (3)

10s

When the DMA transfer completes, the disk controller notifies the CPU with an *interrupt* "I am done" (i.e., asserts a special "interrupt" pin on the CPU)



How are I/Os handled on the host?
How are I/Os exposed to programmers?

Qs we will address

Outline

10s

file is the main abstraction for data that is stored

1. **File System**
   - How are I/Os exposed to programmers?
   - How are I/Os handled on the host?
2. Storage devices
   - What does a disk controller do?
   - How do hosts interact with disk controllers?
   - How does a disk controller perform a DMA?
3. Computational Storage

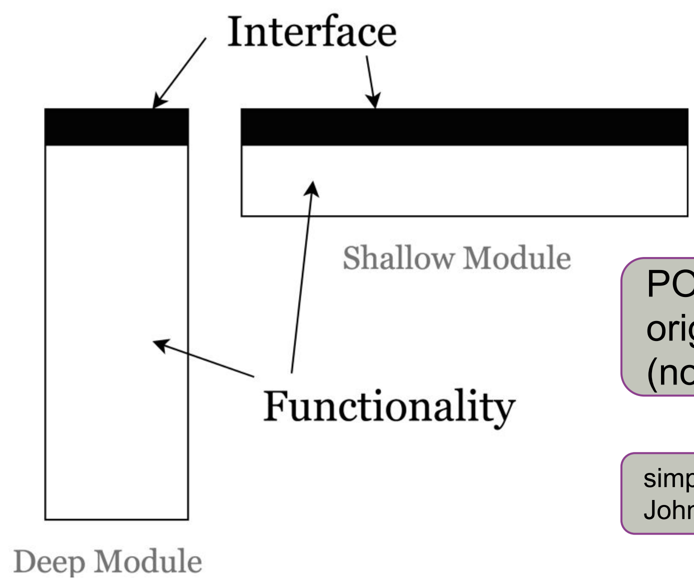
because they are cool

Files

like memory!

A file is an array of bytes

File interface: create/delete, open/close, read/write



POSIX: unified unix specification.
original POSIX interface is beautiful.
(not part of C language, but std lib)

simple interface, hiding a lot of complexity.
John Ousterhout, professor at Stanford.

https://www.youtube.com/watch?v=bmSAYlu0NcY&feature=youtu.be

Outline

we'll go through

- different layers of the file system, and
- how it works in Linux.

if you are serious about programming, then you'll be applying for a job where you have to *program during the interview*. one vanilla question: "what happens when you read (or write) to (from) a file?"

File System Layers

What happens when you open a file or when you read from a file?

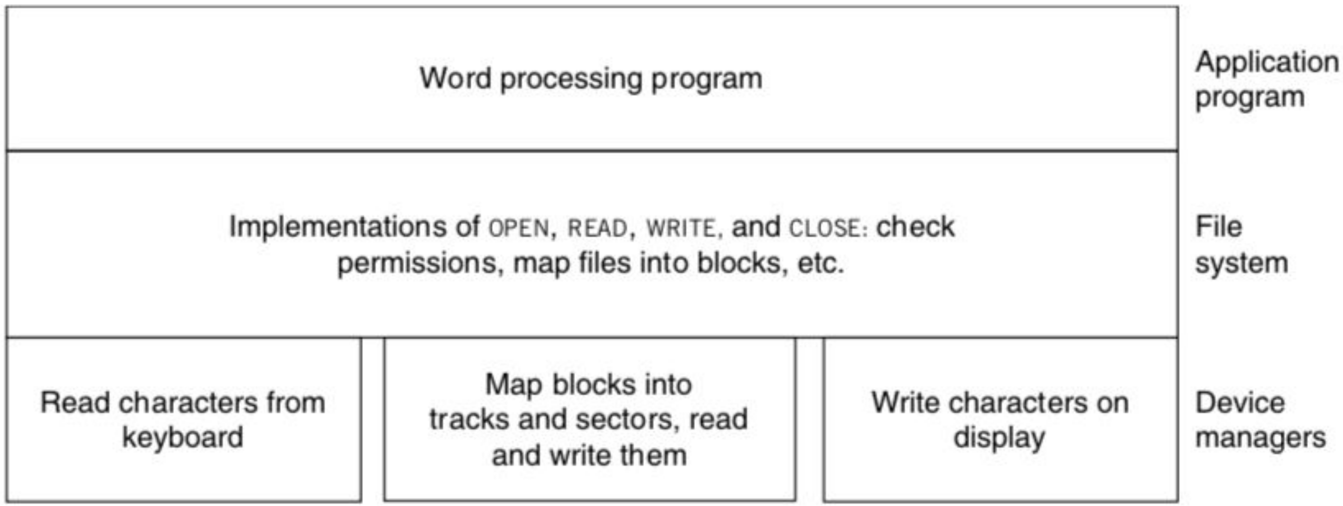
Linux File System Components

How is the Linux file system organized?

Layering and Naming

just an exercise in:

10s



block devices

Figure from Principles of Computer System Design, Saltzer /Kaashoek

Block Layer

we start with:

30s

A **block device** is an **array of blocks**.
To each block is associated a number,
a Logical Block Address (LBA)

sound familiar?
virtual memory! pages!
(quantized data)
block is a unit of transfer.
(associativity layer maps block number to actual block)

```
procedure BLOCK_NUMBER_TO_BLOCK (integer b) returns block
    return device[b]
```

hard disk drives today
(incl. SSDs) are block devices.

File Layer

How to represent files?
Each **file** is a **collection of disk blocks**
(more abstractly (haha), an **array of bytes**)

```
structure inode {
        integer block_numbers[N];   // the numbers of the blocks that constitute the file
        integer size;               // the size of the file in bytes
}
```

inode (“index node”) is a collection of block numbers (associated to the file), and their collective size.
(we need this level of indirection)

```
procedure INDEX_TO_BLOCK_NUMBER (instance of inode i, integer index) returns integer {
        return i.block_numbers[index];
}
```

```
procedure INODE_TO_BLOCK ( integer offset, instance of inode i) returns instance of block
        o ← offset / BLOCKSIZE;
        b ← INDEX_TO_BLOCK_NUMBER(inode, o);
        return BLOCK_NUMBER_TO_BLOCK(b);
}
```

Inode Name Layer

File system state:
inode_table

How to avoid carrying inodes around?

level of indirection

number the inodes!
inode_number to inode table (map),
carry this table around.

```
procedure INODE_NUMBER_TO_INODE(integer inode_number) returns instance of inode{
        return inode_table[inode_number];
}
```

```
procedure INODE_NUMBER_TO_BLOCK (integer offset, integer inode_number)
                                        returns instance of block {
        structure inode i ← INODE_NUMBER_TO_INODE (inode_number);
        o ← offset / BLOCKSIZE;
        b ← INDEX_TO_BLOCK_NUMBER (i, o);
        return BLOCK_NUMBER_TO_BLOCK (b);
}
```

```
return INODE_TO_BLOCK(offset,i);
```

File Name Layer

File system state: inode_table

Representing directories

directory is also an inode.
now we have 2 types of inodes.

```
structure inode {
    integer block_numbers[N];   // the numbers of the blocks that constitute the file
    integer size;               // the size of the file in bytes
    integer type;               // type of file: regular file, directory,...
}
```

each block stores many inode nums

User-friendly names

File name	Inode number
program	10
Paper	12

when you work with files, you don't work with inode numbers. you work with filenames. need mapping from filename to inode number.

in dir, we store, **alongside an inode number**, the *filename* of that inode.

File Name Layer

File system state:
inode_table

Directory lookup

```
procedure NAME_TO_INODE_NUMBER (character string filename, integer dir) returns integer {
    return LOOKUP (filename, dir);
}

procedure LOOKUP (character string filename, integer dir) returns integer{
    instance of block b;
    instance of inode i ← INODE_NUMBER_TO_INODE (dir);
    if i.type ≠ DIRECTORY then return FAILURE;
    for offset from 0 to i.size – 1 do {
        b ← INODE_NUMBER_TO_BLOCK (offset, dir);
        if STRING_MATCH (filename, b) then {
            return INODE_NUMBER (filename, b);// return inode number for filename
        }
        offset ← offset + BLOCKSIZE;        // increase offset by block size
    }
    return FAILURE;
}
```

(inode number)

if filename occurs in b,

then return the inode num that's written next to the filename

`STRING_MATCH, INODE_NUMBER` implementation not shown

Path Name Layer

File system state:
inode_table

Hierarchy of Directories

```
procedure PATH_TO_INODE_NUMBER (character string path, integer dir) returns integer{
        if (PLAIN_NAME (path)) return NAME_TO_INODE_NUMBER (path, dir);
        else {
                dir ← LOOKUP (FIRST (path), dir);
                path ← REST (path);
                return PATH_TO_INODE_NUMBER (path, dir);
        }
}
```

Absolute Path Name Layer

File system state:
inode_table
Process state:
wd

Change working directory

```
procedure CHDIR (path character string) { wd ← PATH_TO_INODE_NUMBER (path, wd); }
```

How to name a file regardless of the current working directory?

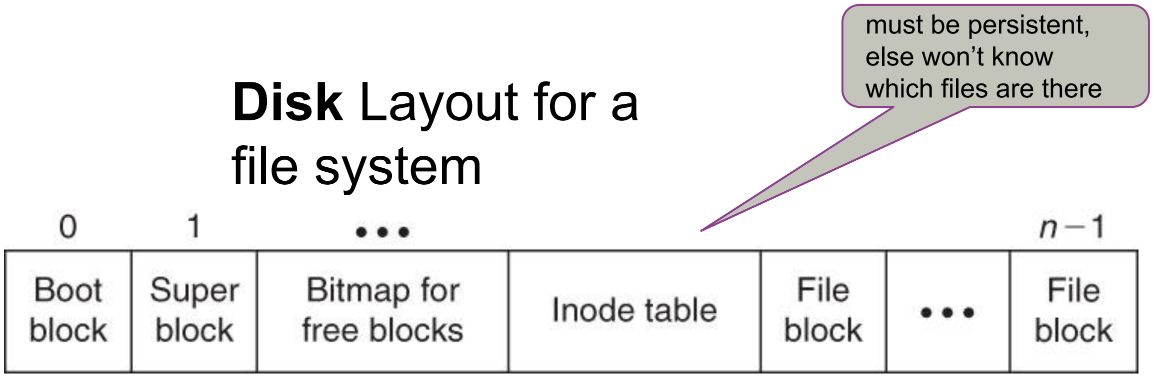
```
procedure GENERALPATH_TO_INODE_NUMBER (character string path) returns integer {
    if (path[0] = "/") return PATH_TO_INODE_NUMBER(path, 1);
    else return PATH_TO_INODE_NUMBER(path, wd);
}
```

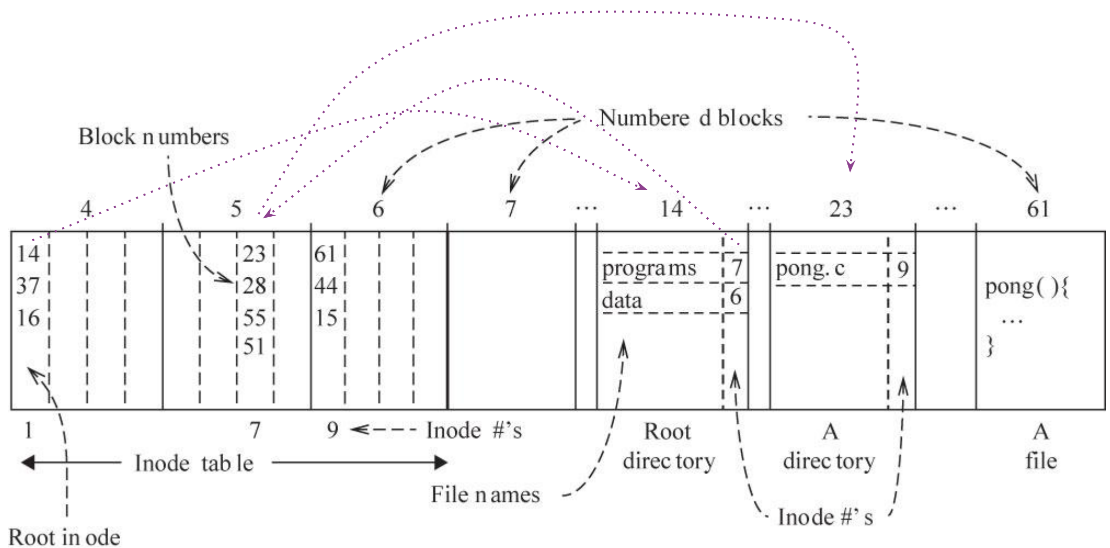
(root inode number)

Unix File System Naming Scheme

File system state:
inode_table
Process state:
wd

Disk Layout for a file system





Symbolic Link Layer

we can have multiple paths to the same inode.

How about flexible management of files?

LINK (*from_name*, *to_name*);
UNLINK (*from_name*);

```
structure inode {
    integer block_numbers[N];
    integer size;
    integer type;
    integer refcnt;
}
```

once no path refers to inode, it can be garbage collected.

Symbolic Link Layer

How to create links across file systems (where the inode numbers are not unique)?

```
procedure PATHNAME_TO_INODE (character string filename) returns instance of inode{
        instance of inode i;
        inode_number ← GENERALPATH_TO_INODE_NUMBER (filename);
        i ← INODE_NUMBER_TO_INODE (inode_number);
        if i.type = SYMBOLIC then
                i = GENERALPATH_TO_INODE_NUMBER (COERCE_TO_STRING (i.block_numbers))
        return i;
}
```

Symbolic Link Layer

in Linux, you can mount a file system

10s

How to attach new disks to a file system?

MOUNT ("/dev/fd1", "/floppy")

disk

mount point

(1) represents a file system
(2) device and root inode for the given file system

Inode pinned in memory for <u>usb</u>

(1) name of parent inode, i.e., usb

Inode pinned in memory for <u>/dev/usb1</u>

Naming Layers in Unix File System

10s

Layer	Names	Values	Context	Name-mapping algorithm	
Symbolic link	Path names	Path names	The directory hierarchy	PATHNAME_TO_GENERAL_PATH	
Absolute path name	Absolute path names	Inode numbers	The root directory	GENERALPATH_TO_INODE_NUMBER	user-oriented names
Path name	Relative path names	Inode numbers	The working directory	PATH_TO_INODE_NUMBER	
File name	File names	Inode numbers	A directory	NAME_TO_INODE_NUMBER	machine-user interface
Inode number	Inode numbers	Inodes	The inode table	INODE_NUMBER_TO_INODE	
File	Index numbers	Block numbers	An inode	INDEX_TO_BLOCK_NUMBER	machine-oriented names
Block	Block numbers	Blocks	The disk drive	BLOCK_NUMBER_TO_BLOCK	

Source: Saltzer and Kaashoek

API: State

File system state:
- inode_table
- file_table

Process state:
- fd_table
- wd

Which files are in use?
`file_table`

File name	Inode number	cursor
program	10	64
Paper	12	0

Cursor is the first byte that will be accessed by the next read or write operation.

Which files is each process using?
`fd_table`

Mapping from file descriptors into the **file_table**.
(file descriptors are per-process. natural numbers; 0 is stdin, 1 is stdout, 2 is stderr, …)

Multiple processes can have a file open with different cursors, and
Multiple processes can have a file open sharing a cursor (fork; **fd_table** shared)

API: inode

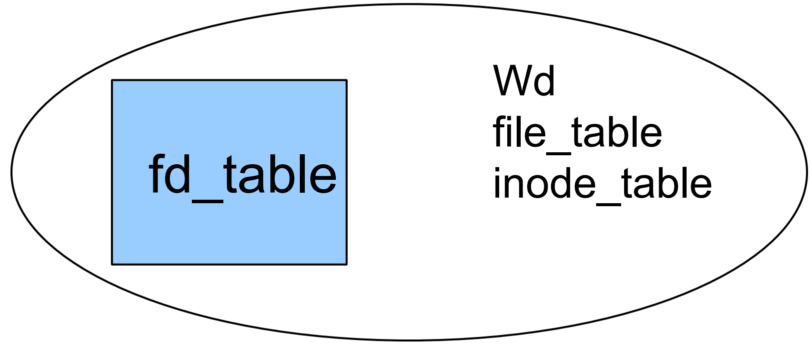
```
structure inode {
    integer block_numbers[N];   // the number of blocks that constitute the file
    integer size;               // the size of the file in bytes
    integer type;               // type of file: regular file, directory, symbolic link
    integer refcnt;             // count of the number of names for this inode
    integer userid;             // the user ID that owns this inode
    integer groupid;            // the group ID that owns this inode
    integer mode;               // inode's permissions
    integer atime;              // time of last access (READ, WRITE,...)
    integer mtime;              // time of last modification
    integer ctime;              // time of last change of inode
```

we did not talk about e.g. access control

API Calls: Open

skip

Process state

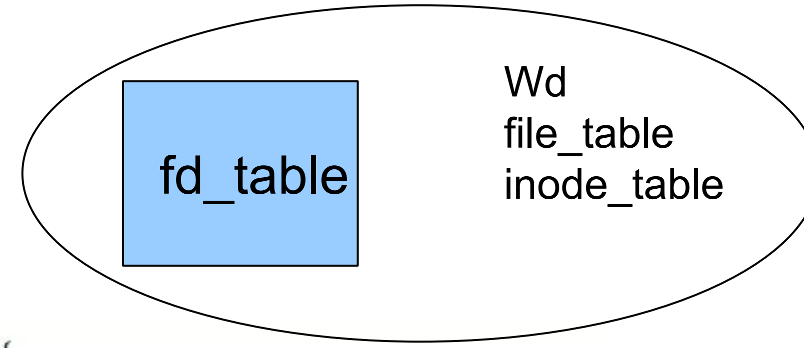



```
procedure OPEN (character string filename, flags, mode) {
    inode_number ← PATH_TO_INODE_NUMBER (filename, wd);
    if inode_number = FAILURE and flags = O_CREATE then { // Create the file?
        inode_number ← CREATE (filename, mode);           // Yes, create it.
    } else return FAILURE;
    inode ← INODE_NUMBER_TO_INODE (inode_number);
    if PERMITTED (inode, flags) then {   // Does this user have the required permissions?
        file_index ← INSERT (file_table, inode_number);
        fd ← FIND_UNUSED_ENTRY (fd_table);// Yes, find entry in file descriptor table
        fd_table[fd] ← file_index;             // Record file index for the file descriptor
        return fd;                        // Return fd
    } else return FAILURE;                // No, return a failure
}
```

API Calls: Read

skip

Process state



File name	Inode number	cursor
program	10	64
Paper	12	0

```
procedure READ (fd, reference buf, n) {
    file_index ← fd_table[fd];
    cursor ← file_table[file_index].cursor;
    inode ← INODE_NUMBER_TO_INODE (file_table[file_index].inode_number);
    m = MINIMUM (inode.size – cursor, n);
    atime of inode ← NOW ();
    if m = 0 then return END_OF_FILE;
    for i from 0 to m – 1 do {
        b ← INODE_NUMBER_TO_BLOCK (i, inode_number);
        COPY (b, buf, MINIMUM (m – i, BLOCKSIZE));
        i ← i + MINIMUM (m – i, BLOCKSIZE);
    }
    file_table[file_index].cursor ← cursor + m;
    return m;
}
```

Outline

a word about the Linux file system.

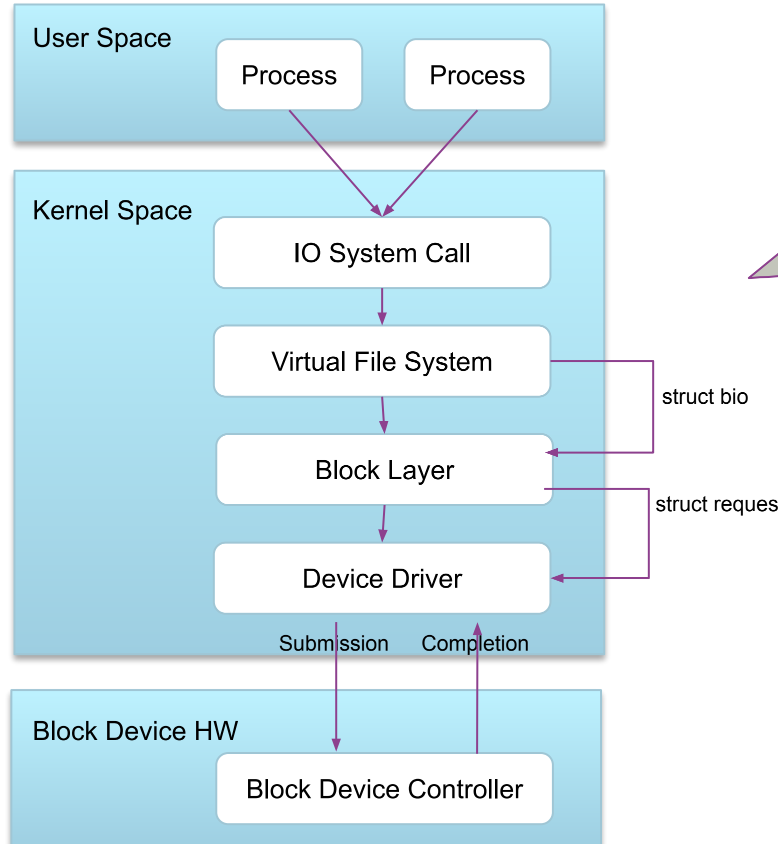
File System Layers

What happens when you open a file or when you read from a file?

Linux File System Components

How is the Linux file system organized?

Linux File System



file system is part of the operating system.

uniform way for different file systems to hook up to Linux. (POSIX)

Linux I/O System Calls

- creat, open, read, write, close, lseek
- fsync
- link, unlink
- stat, lstat, fstat
- access, umask, chmod, chown, utime
- ioctl

libraries for this

there are also *async I/O* system calls. (e.g. aio_read)
and ways to batch system calls (io_submit, …)

Linux Virtual File System

The virtual file system defines the generic file system interface and data structures:

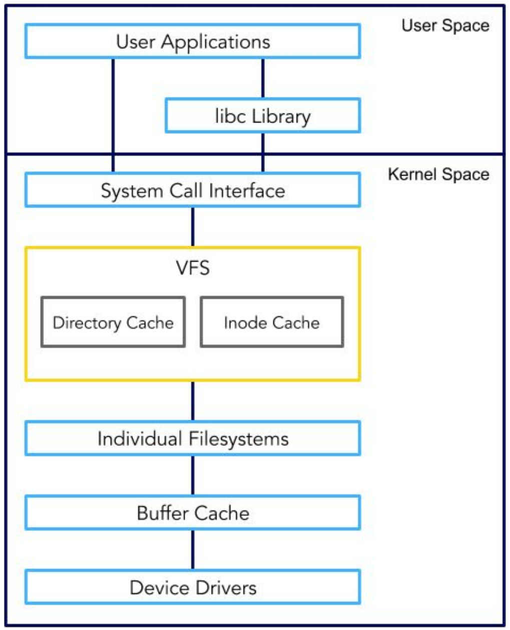
file, dentry, inode, vfsmount, super_block.

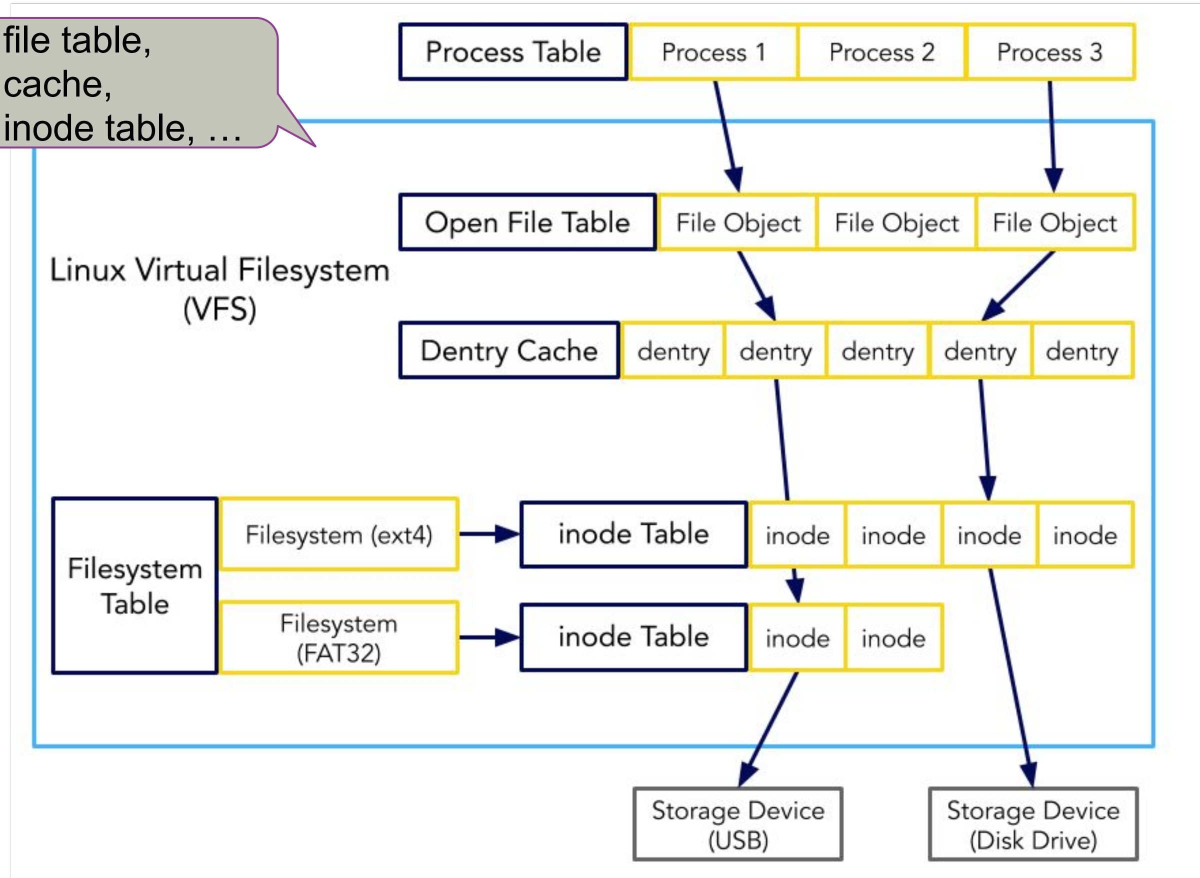
Each specific file system provides a specific implementation:

block-based FS (ext4, btrfs), network FS (NFS, ceph),
stackable FS, pseudo FS (sysfs),
special purpose FS (tmpfs)

Linux VFS

they all respect the VFS setup.





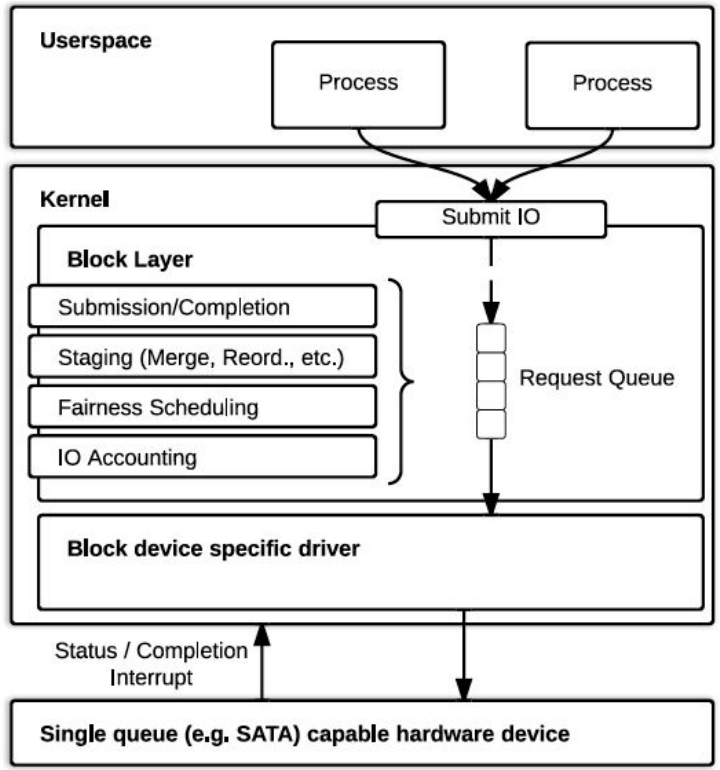
file table,
cache,
inode table, …

don't be afraid of VFS; it's just inodes, inode table, etc.

https://www.starlab.io/blog/introduction-to-the-linux-virtual-filesystem-vfs-part-i-a-high-level-tour

Linux Legacy Block Layer

block layer is taking block I/O requests, and issuing those to HW. how hard can that be?



in 2013, they started making block layer smart: disk receives many random I/Os ⇒ disk wants to reorder them to be sequential (lump them together to be smart about how you access disk).

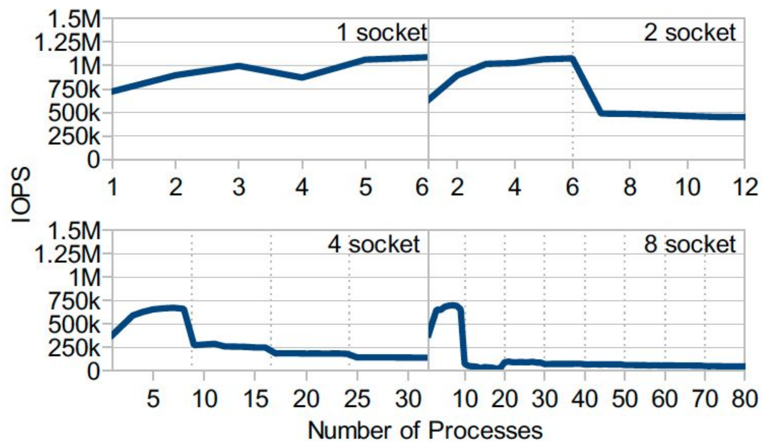
devices are async, yet I/O from FS is sync.
mapping from synchronous to asynchronous was done by the block layer.

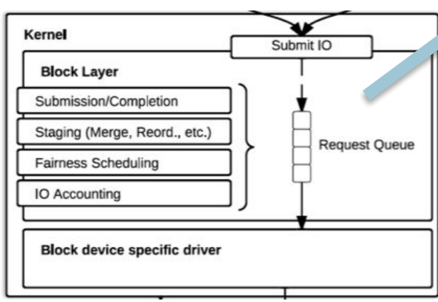
https://kernel.dk/blk-mq.pdf

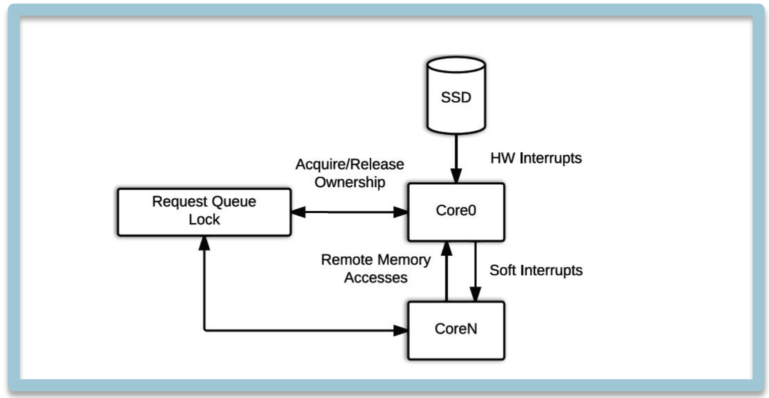
Scalability Problem

suddenly, we have disks that are real fast,
CPUs that are fast, but
CPUs cannot access disks fast.

Platform (Intel)	Sandy Bridge-E	Westmere-EP	Nehalem-EX	Westmere-EX
Processor	i7-3930K	X5690	X7560	E7-2870
Num. of Cores	6	12	32	80
Speed (Ghz)	3.2	3.46	2.66	2.4
L3 Cache (MB)	12	12	24	30
NUMA nodes	1	2	4	8



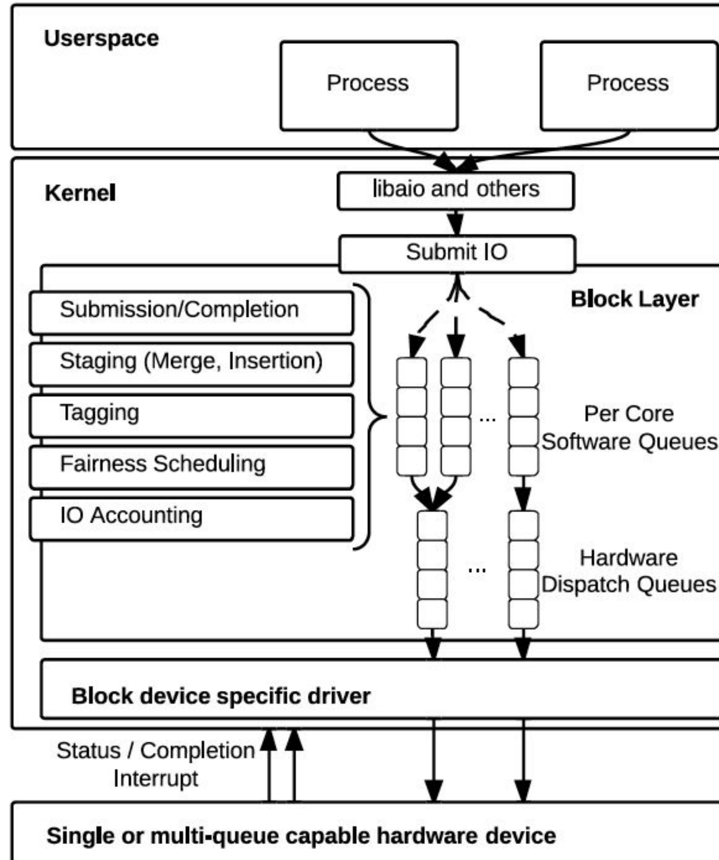




https://kernel.dk/blk-mq.pdf

Linux mlqblk Block Layer

https://kernel.dk/blk-mq.pdf



multi-queue block layer.
i.e. per-core queue in software.
sound good?

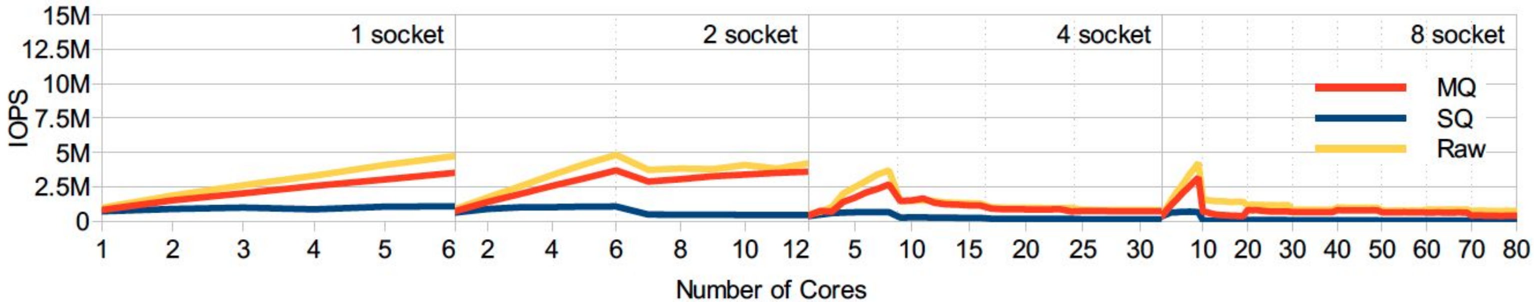
block layer maintainer

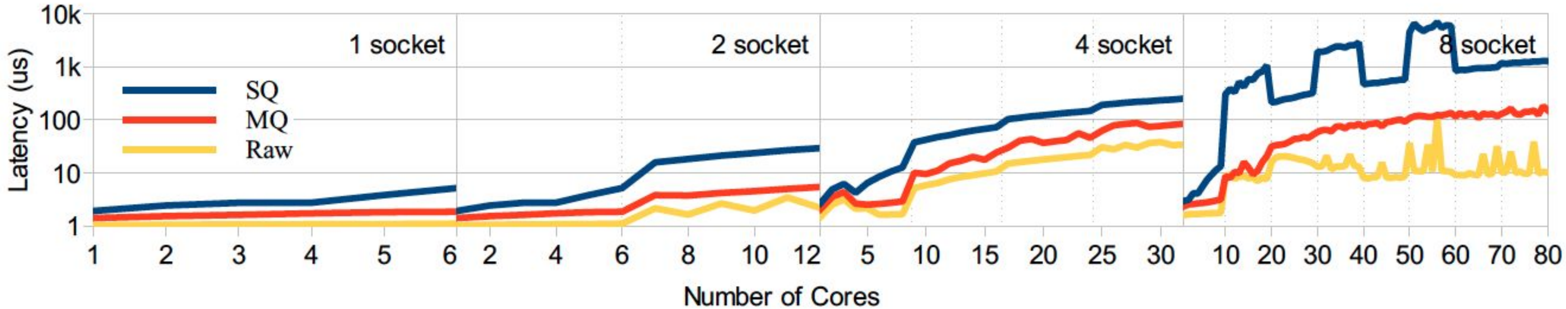
Jens Axboe's design

Experimental Results [Systor13]

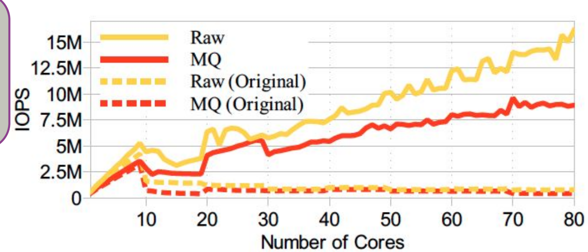
https://kernel.dk/blk-mq.pdf

hmm, that yielded no speedup.



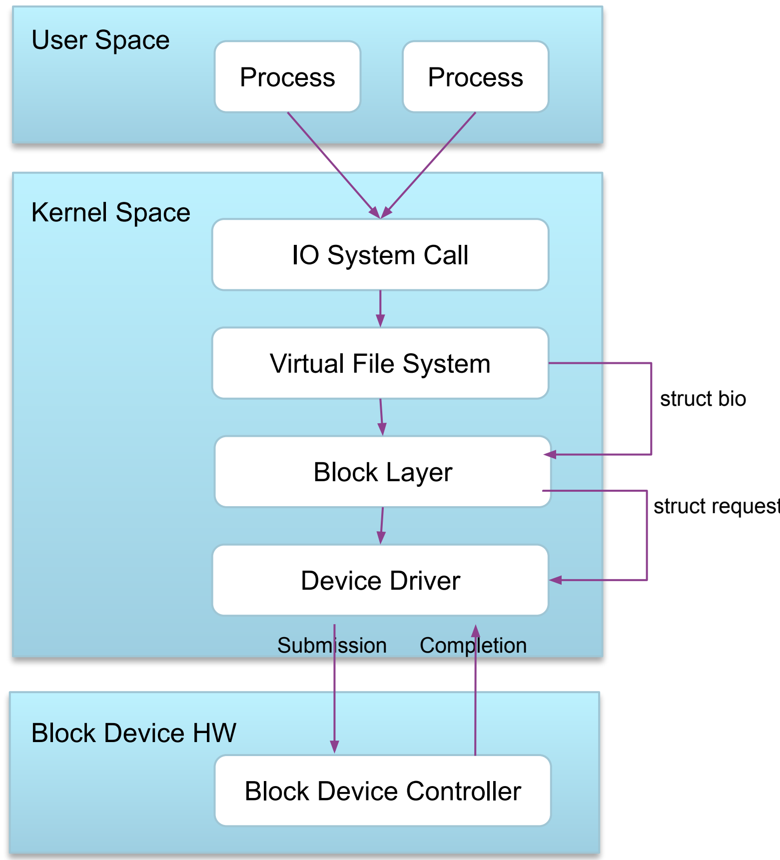


last 10 years of work in Linux IO stack: remove latency to increase throughput to IO devices



Linux File System

1s



Device Drivers

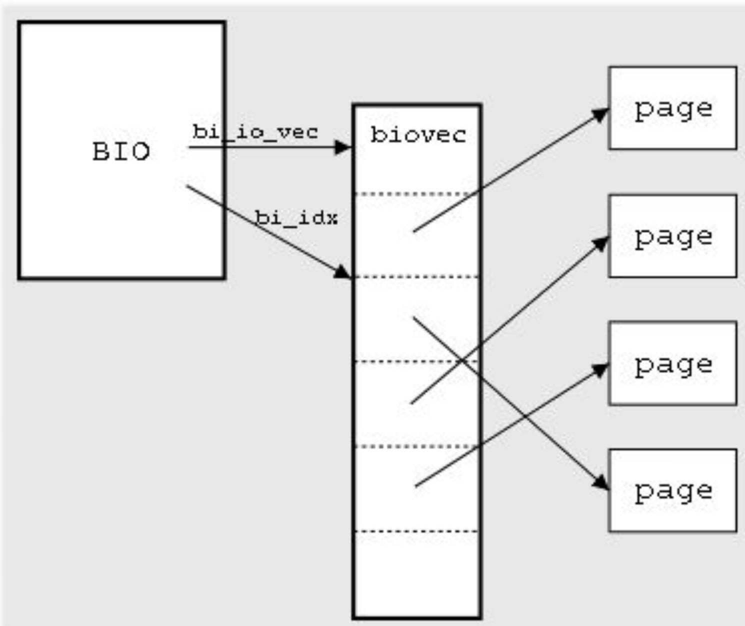
10s

- How are I/Os represented?
  - Data structures: bios and requests
- How are I/Os submitted?
  How are I/O completions handled?
  - What is the storage interface?
  - Put differently: What is the abstraction of the underlying storage devices?

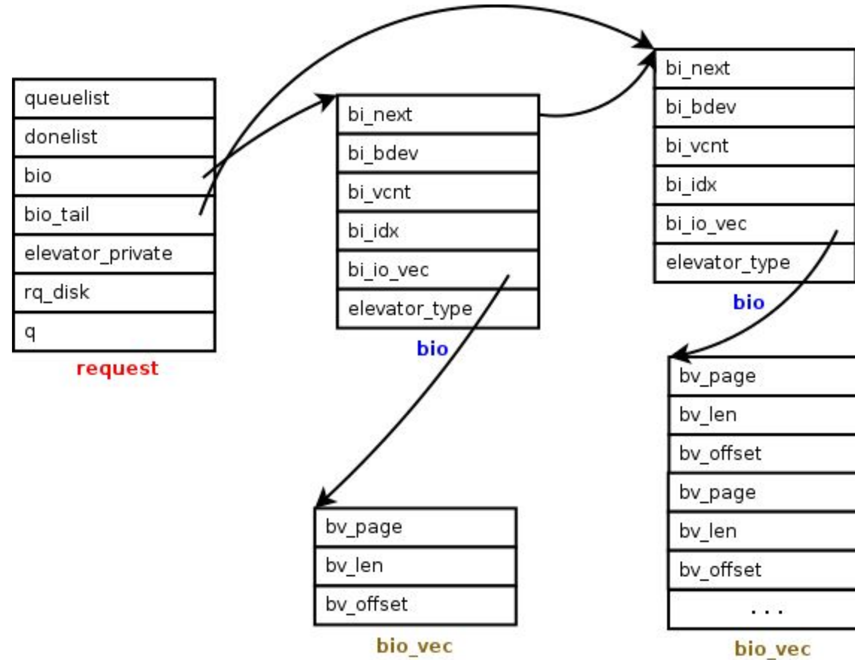
Data structures: bio and requests

(just wanted to mention this, not dwell on it)

10s



http://elixir.free-electrons.com/linux/latest/source/Documentation/block/biodoc.txt



https://www.kernel.org/doc/Documentation/block/request.txt

Outline

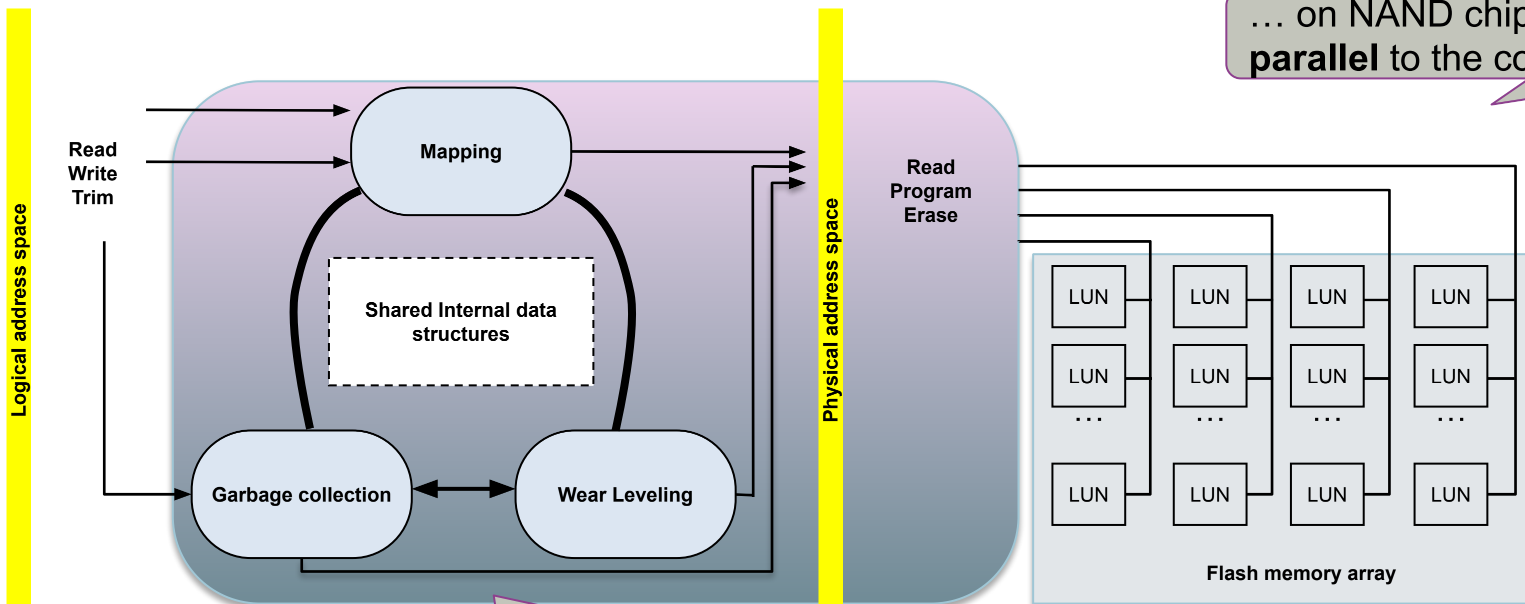
1. File System
   - How are I/Os exposed to programmers?
   - How are I/Os handled on the host?
2. **Storage devices**
   - What does a disk controller do?
   - How do hosts interact with disk controllers?
   - How does a disk controller perform a DMA?
3. Computational Storage

What does a SSD controller do?

1. Handles interactions with host
2. Maps logical ops onto physical reads, writes, erase

… on NAND chips, wired in **parallel** to the controller

(LUN = logical unit number)



(added sophistication)

(wear leveling = equi-distribute the wear of disk)

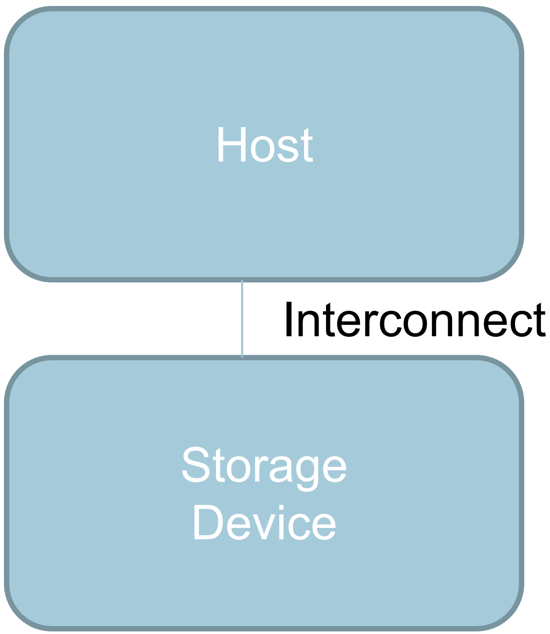
Flash Translation Layer (FTL)

(Copenhagen!)

much work on clever mapping (grouping) that lowers need for garbage collection

(in SSD, random access is just as fast as sequential access, assuming well-managed parallelism)

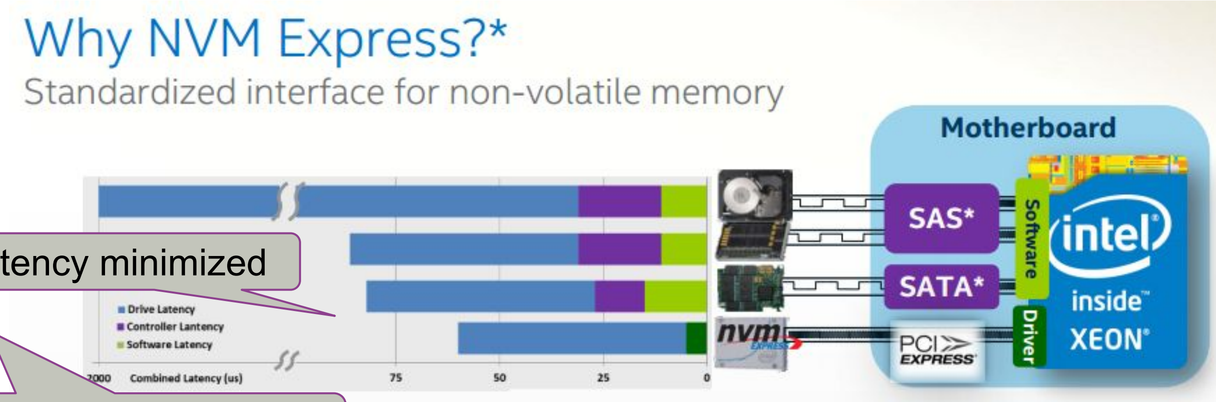
Device-Host Interconnect



- Physical Interconnect
  - SATA / AHCI
  - PCIe
  - Ethernet
  - …
- Protocol
  - SATA
  - **NVMe**
  - NVMf

NVMe

(v2.0 introduced in July 2021)

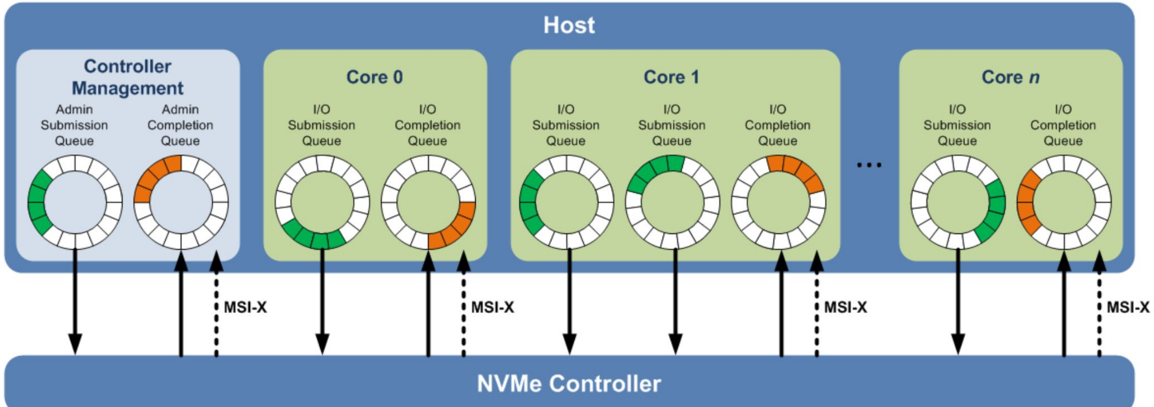


latency minimized

(why: with NVMe, memory is shared on the PCIe bus)

(driver on host)

queues (submission, completion) shared between host and device
(access memory on host from device)



(on the device)

Simple Command Set – Optimized for NVM

Admin Commands
Create I/O Submission Queue
Delete I/O Submission Queue
Create I/O Completion Queue
Delete I/O Completion Queue
Get Log Page
Identify
Abort
Set Features
Get Features
Asynchronous Event Request
Firmware Activate (optional)
Firmware Image Download (opt)
Format NVM (optional)
Security Send (optional)
Security Receive (optional)

NVM I/O Commands
Read
Write
Flush
Write Uncorrectable (optional)
Compare (optional)
Dataset Management (optional)
Write Zeros (optional)
Reservation Register (optional)
Reservation Report (optional)
Reservation Acquire (optional)
Reservation Release (optional)

Only 10 Admin and 3 I/O commands required

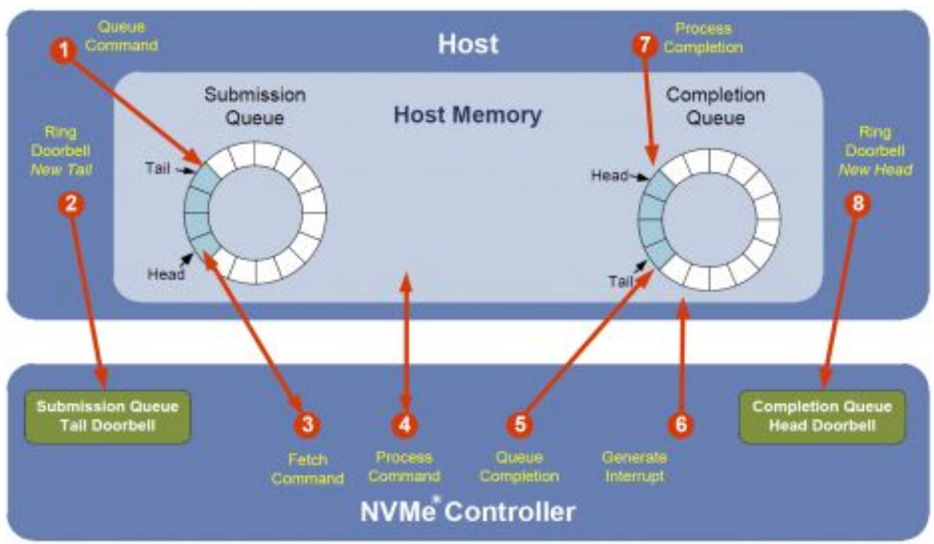
IDF14

45

Interconnect - NVMe

the process:

30s



	AHCI	NVMe
Maximum Queue Depth	1 command queue 32 commands per Q	64K queues 64K Commands per Q
Un-cacheable register accesses (2K cycles each)	6 per non-queued command 9 per queued command	2 per command
MXI-X and Interrupt Steering	Single interrupt; no steering	2K MSI-X interrupts
Parallelism & Multiple Threads	Requires synchronization lock to issue command	No locking
Efficiency for 4KB Commands	Command parameters require two serialized host DRAM fetches	Command parameters in one 64B fetch
Driver Support	Typically in-box	Installed with device

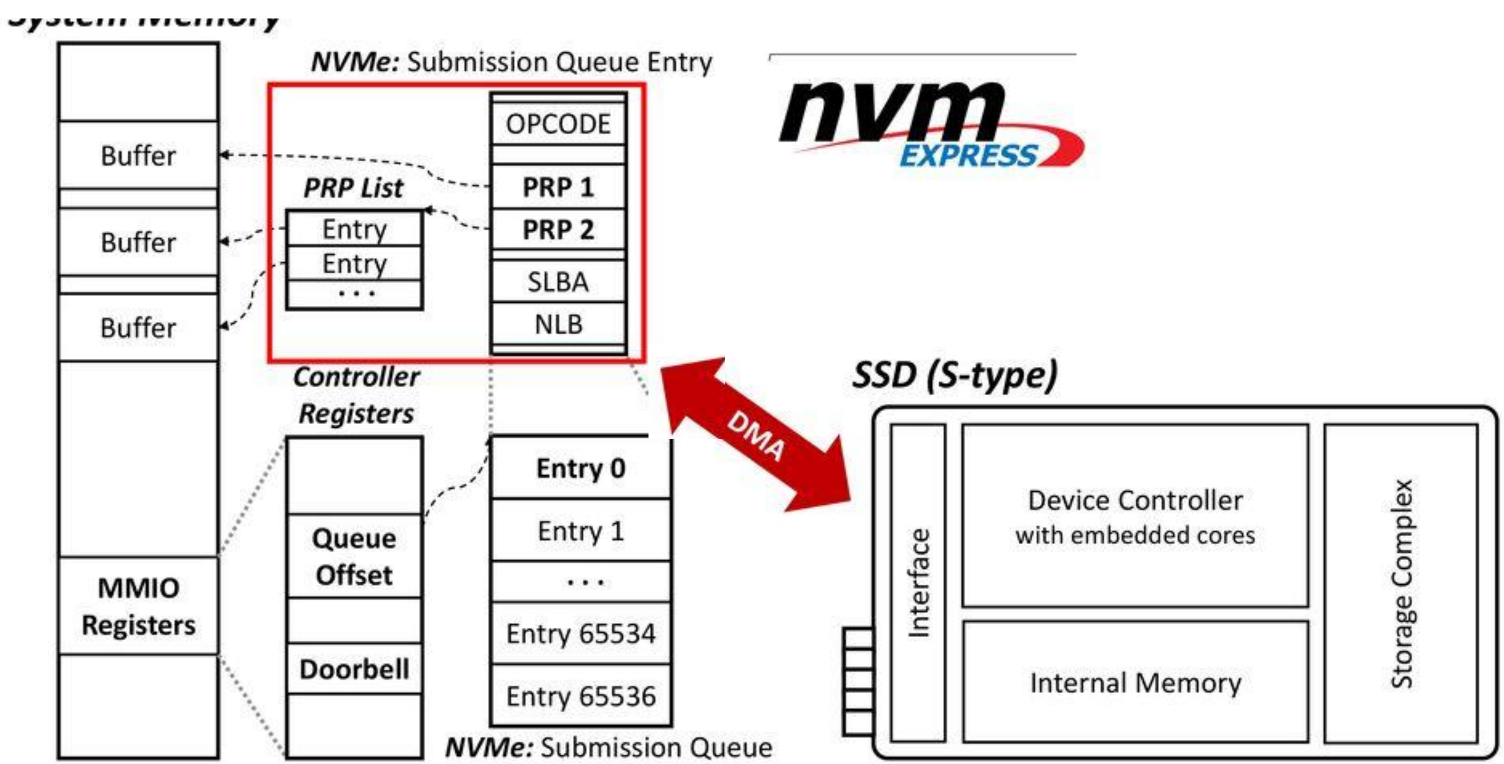
if you want to be fast with how you do IO, then you have to manipulate, from your program, submissions and completions.

https://www.flashmemorysummit.com/English/Collaterals/Proceedings/2013/20130812_PreConfD_Marks.pdf

How to deal with data transfers?

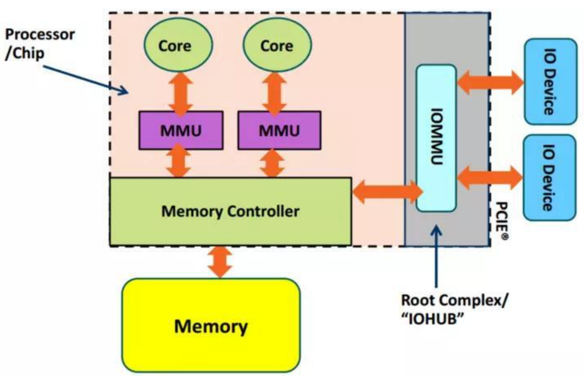
where things are stored
(recall virtual memory)

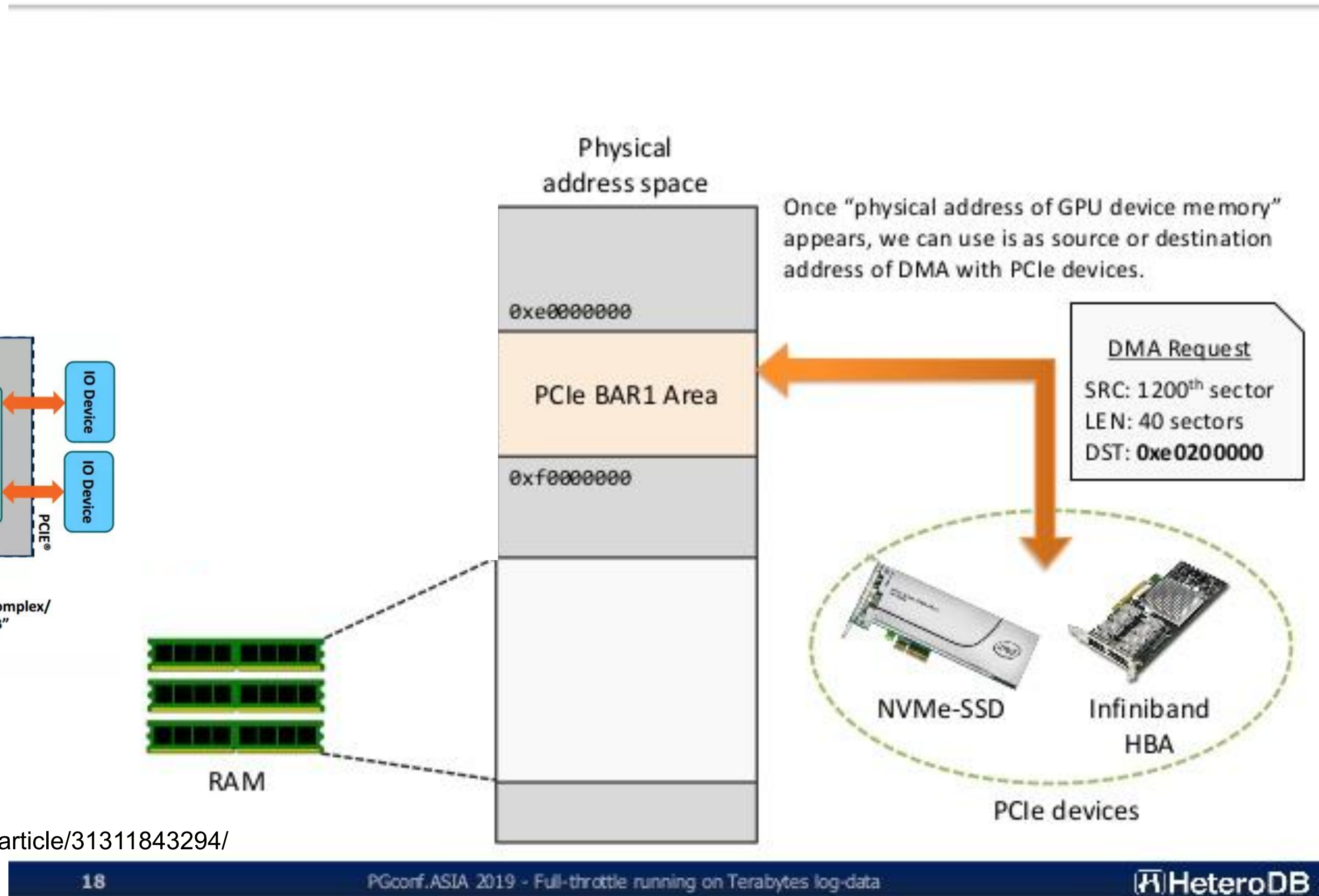
30s



https://slideplayer.com/slide/14772624/

How to deal with data transfers?





https://www.programmersought.com/article/31311843294/

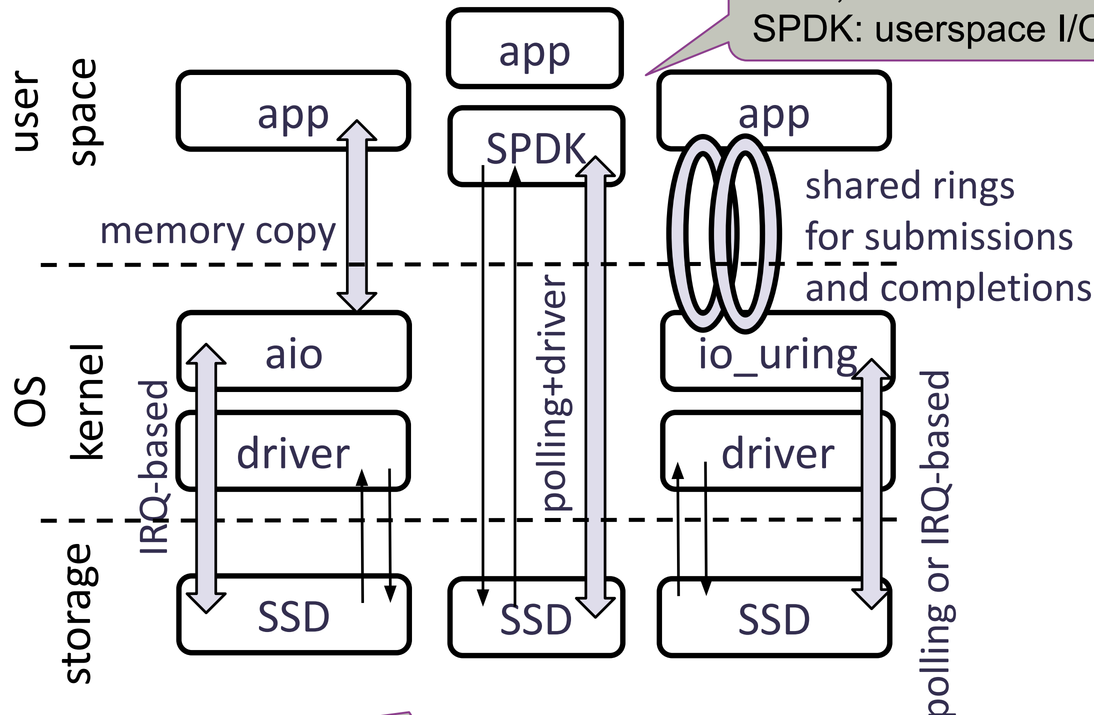
https://www.slideshare.net/Equnix/pgconfasia-2019-bali-fullthrottle-running-on-terabytes-logdata-kohei-kaigai

NVMe Interfaces

usually, data must be transferred from userspace to kernelspace before it can be transferred to a device. (copy, ctx switch, …)

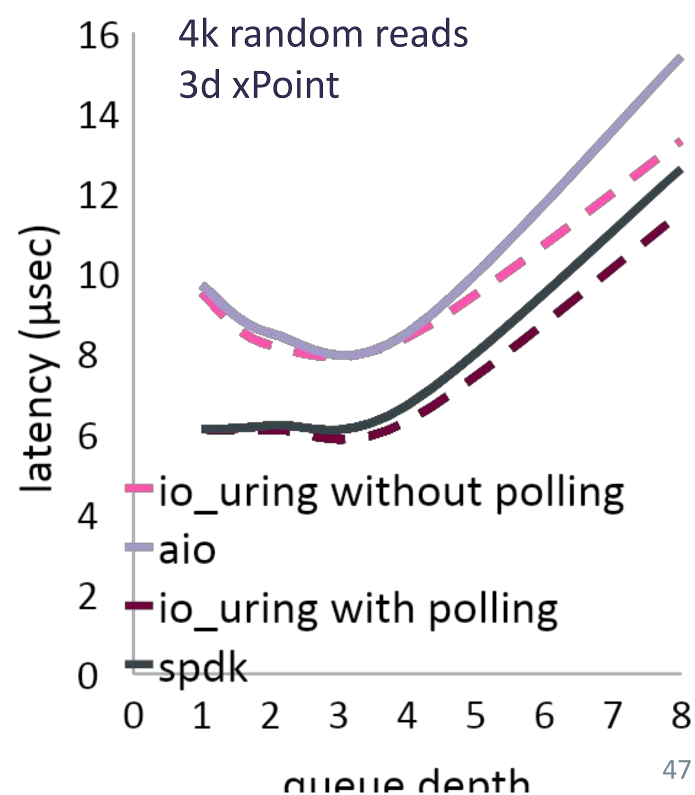
sources: [Faster IO through io_uring] & [Efficient I/O with io_uring] & [J.Axboe]

2015, Intel introduced SPDK: userspace I/O.



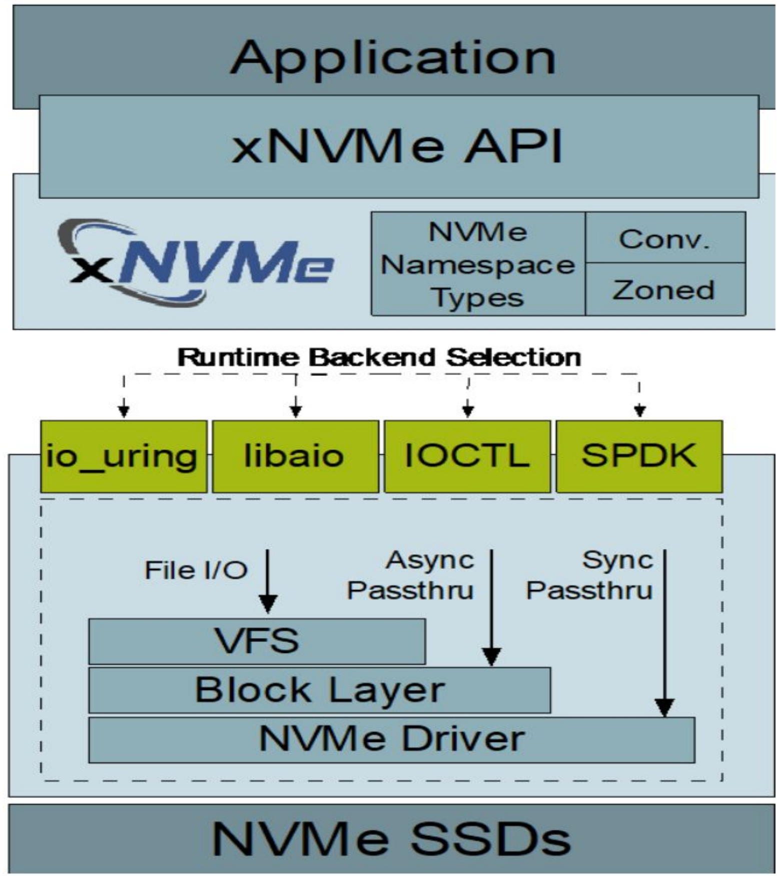
direct mapping from userspace to datastructures on device.

SPDK matches io_uring & aio in performance.



NVMe Interfaces

https://xnvme.io/



Samsung (Copenhagen)

abstracts over these; uniform API

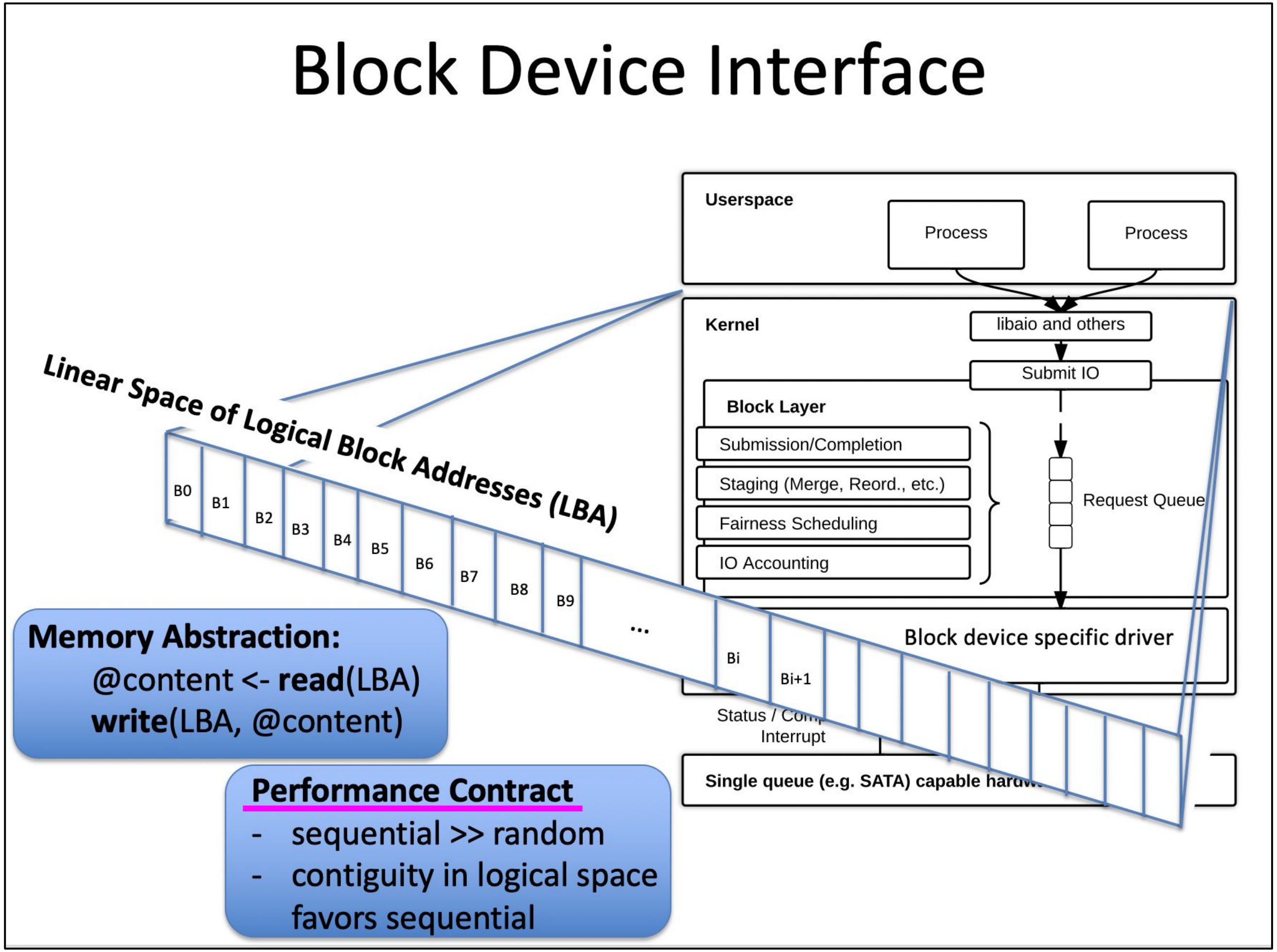
Logical Address Space

Recall, the file systems slides a few slides ago …

A block device is an array of blocks.
To each block is associated a number,
a Logical Block Address (LBA)

Block device

CIDR 2013

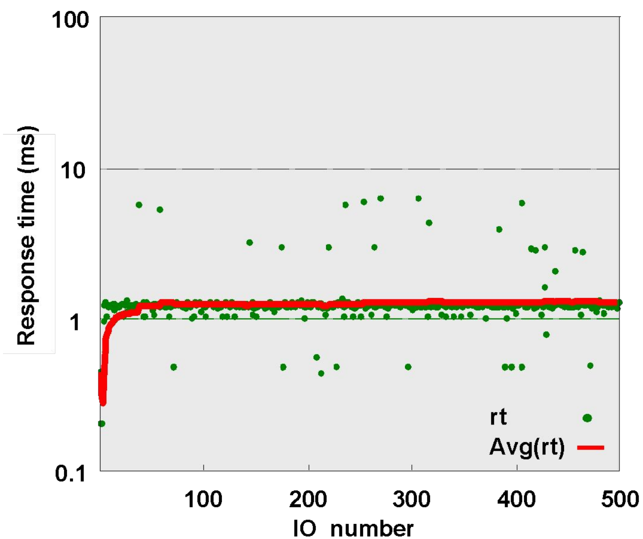


Performance contract?

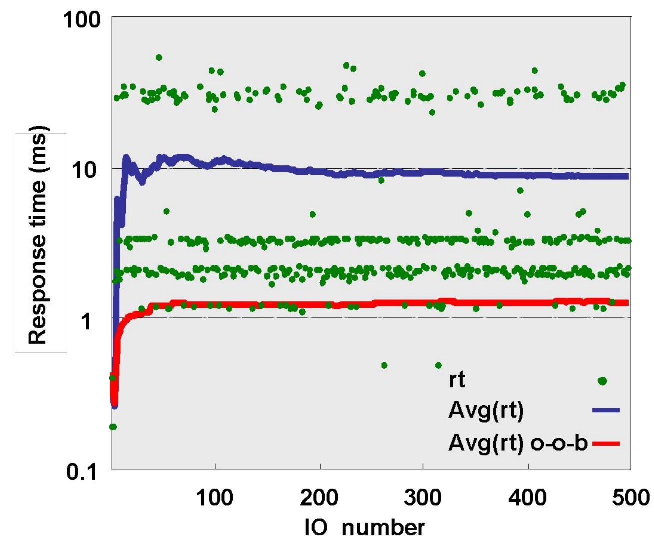
CIDR 2009

Measuring Samsung SSD RW performance

- Out-of-the-box … and after filling the device!!! (similar behavior on Intel SSD)



Random Writes – Samsung SSD
Out of the box



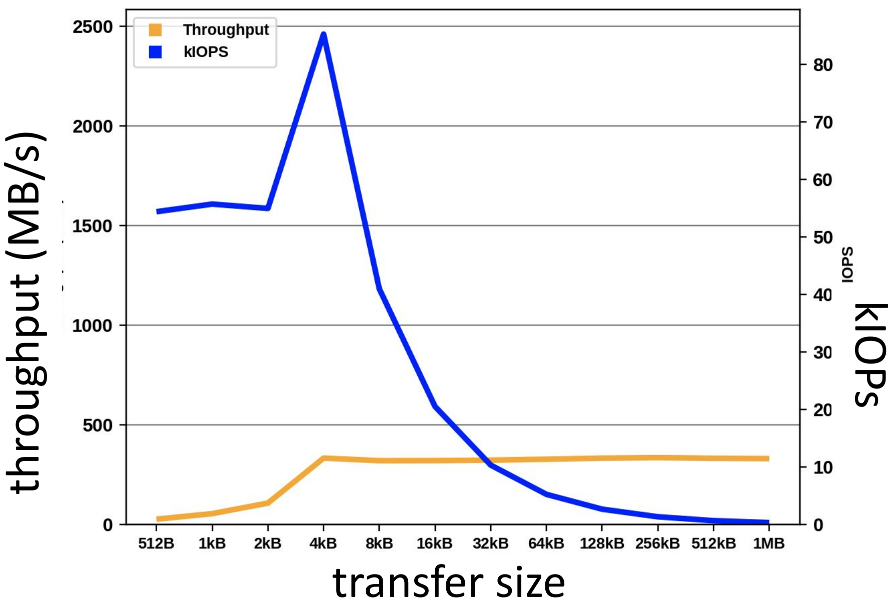
Random Writes – Samsung SSD
After filling the device

Performance contract?

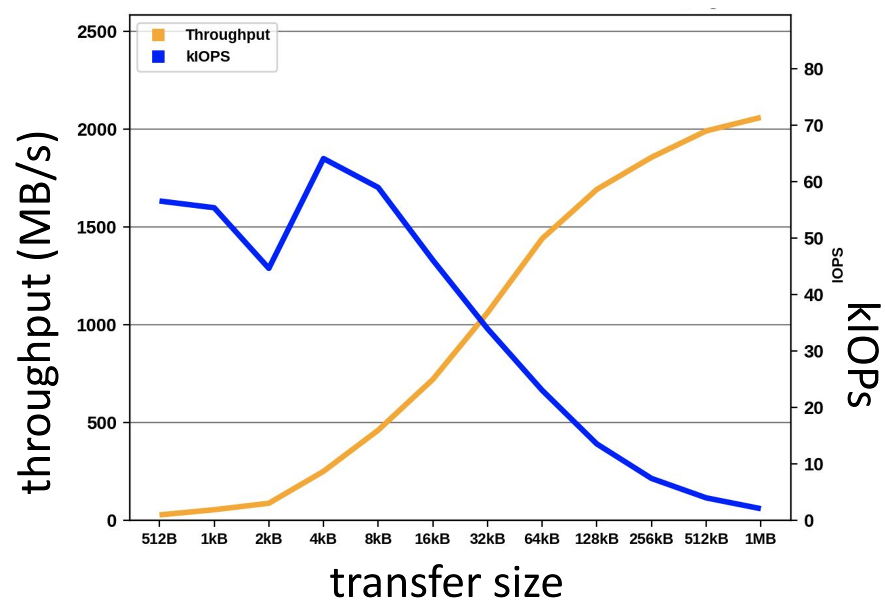
there is nothing interesting about the design of an SSD; it's all about the design of the FTP, and how things are mapped from interface to device.

random writes- source: AnandTech 2019

Samsung SSD with Z-NAND



Intel Optane



No intrinsic performance characteristics for SSDs (equipped with a generic FTL)

(flash translation layer)

Open-Channel

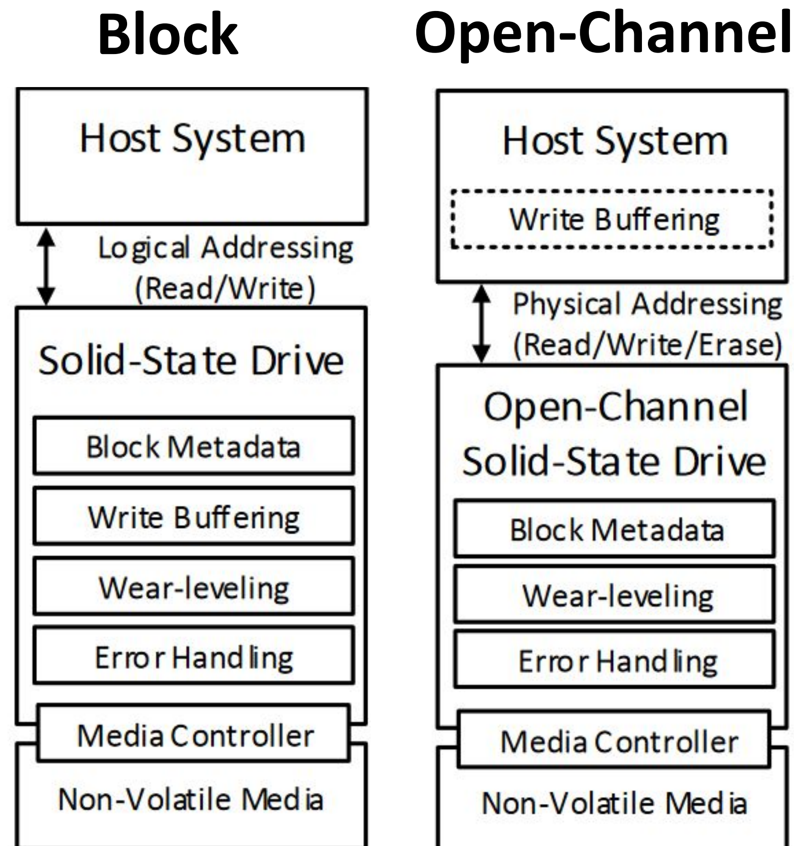
“Let’s make an SSD with no FTL”.
(you don’t want the FTL in your way)
(if you can manage all on machine)

Physical address space exposed

- host can make decisions about ***data placement*** & ***I/O scheduling***

SSD management split between

- back-end (embedded on SSD) ***block metadata*** & ***wear levelling*** (for warrantee)
- front-end (host-based) FTL ***mapping of logical to physical address spaces, overprovisioning,*** & ***garbage collection***



Open-Channel

alibaba, Western Digital, Microsoft, …
but didn't become part of NVMe standard.

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

LightNVM: The Linux Open-Channel SSD Subsystem

Matias Bjørling, *CNEX Labs, Inc. and IT University of Copenhagen;* Javier Gonzalez, *CNEX Labs, Inc.;* Philippe Bonnet, *IT University of Copenhagen*

https://www.usenix.org/conference/fast17/technical-sessions/presentation/bjorling

Matias Bjørling · 1st
Director, Emerging System Architectures at Western Digital
Copenhagen, Capital Region, Denmark · 500+ connections ·

LWN.net
News from the source

Taking control of SSDs with LightNVM

Principal Software Engineer | SSDR R&D Center Lead
Samsung Electronics
Jan 2019 – Present · 10 mos
Copenhagen Area, Capital Region, Denmark
I established and lead Samsung Semiconductor Denmark Research (SSDR) - Samsung's Memory Solutions first R&D center in Europe and fifth worldwide.

Javier González

CNEX LABS AND BROADCOM WIN MOST INNOVATIVE FLASH MEMORY TECHNOLOGY AWARD AT FLASH MEMORY SUMMIT 2017

CNEX Labs Open-Channel SSD technology and Broadcom's NetXtreme S-Series SOC provide unprecedented IO isolation and scalability for hyperscale and cloud service providers

Flash Memory Summit

Alibaba Open Channel Ecosystem

As Alibaba's **strategic partner** on Open Channel SSDs, Intel has worked with Alibaba extensively since 2017 to co-develop and co-validate this innovative solution. Alibaba's strength as a leading cloud service provider combined with Intel's strength as the leading memory and storage innovator puts us in a position to deliver the industry's 1st Open Channel SSD product.

intel · 紫光存储 UNIC² · Micron · SK hynix · Shannon Systems · CNEXLABS

- Alibaba is collaborating with major vendors in industry to build an ecosystem for Open Channel SSD
- Share development & debug resources
- Reduce time & complexity for SSD qualification in Alibaba
- Massive deployment in 2019

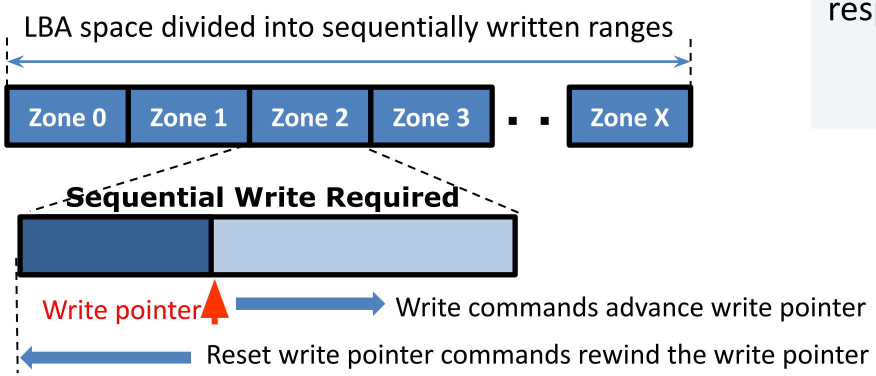
Zoned Namespaces (ZNS)

successor:

30s

Mode	Host managed	Host aware
Host responsibility	Explicit zone transition Write command Write pointer per zone	Implicit zone transition Append command at large queue depth
SSD responsibility	Zone placement I/O scheduling	Zone placement I/O scheduling sequential writes within a zone

you: how to map workload onto zones,
disk: manage parallelism best possible

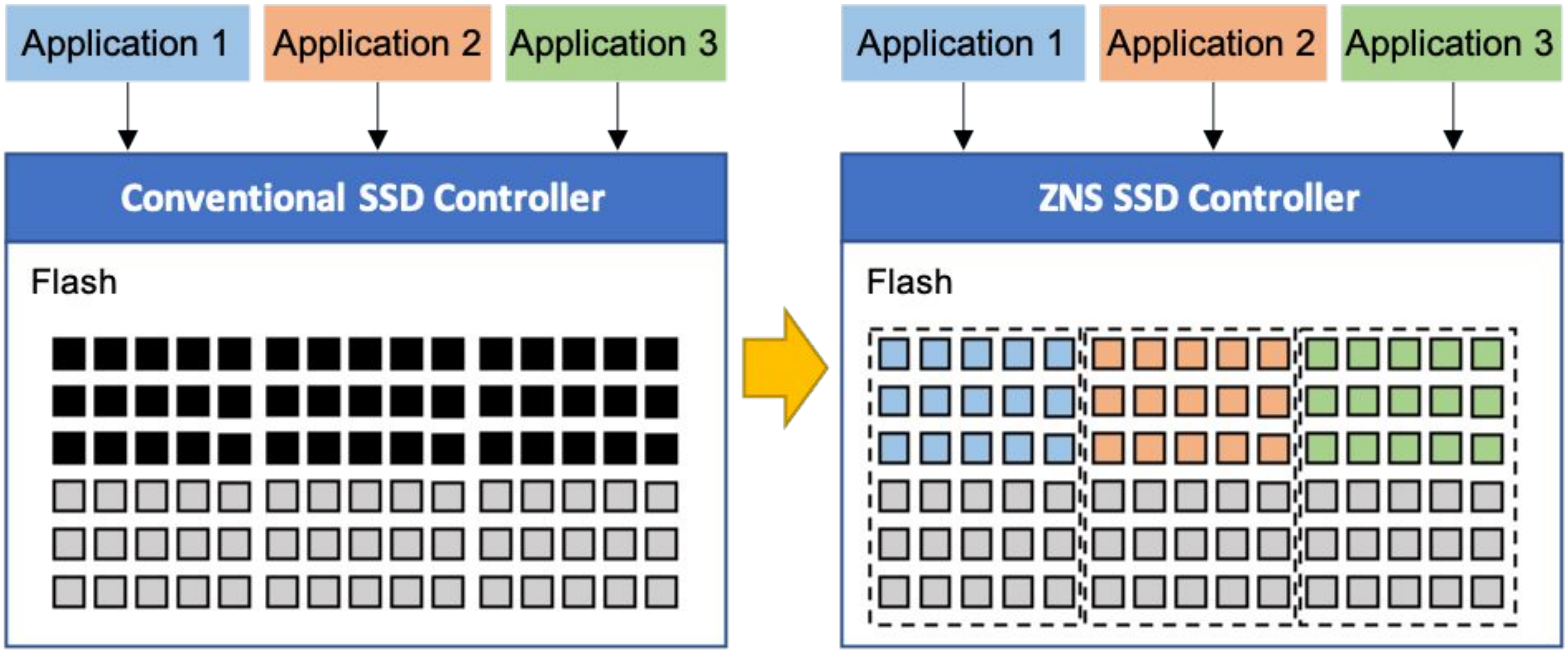


one level of control on device.
(not as good as open-channel)

NVMe: Block and Zoned namespaces

30s

https://nvmexpress.org/new-nvmetm-specification-defines-zoned-namespaces-zns-as-go-to-industry-technology/



Regular SSD: Device controls data placement

ZNS SSD: Applications control data placement in zones

Outline

1. File System
   - How are I/Os exposed to programmers?
   - How are I/Os handled on the host?
2. Storage devices
   - What does a disk controller do?
   - How do hosts interact with disk controllers?
   - How does a disk controller perform a DMA?
3. Computational Storage

conventional SSD: a lot of complexity on device; gets in way of host. open-channel: host did too much work. ZNS is a compromise, still not perfect.

idea: you *should* be able to program your SSDs.

Computational Storage

Put Everything in Future (Disk) Controllers (it's not "if", it's "when?")

Jim Gray

http://www.research.Microsoft.com/~Gray

Acknowledgements:
Dave Patterson explained this to me a year ago
Kim Keeton, **Erik Riedel**, **Catharine Van Ingen** — **Helped me sharpen these arguments**

1

Basic Argument for x-Disks

- Future disk controller is a super-computer.
  - 1 bips processor
  - 128 MB dram
  - 100 GB disk plus one arm
- Connects to SAN via high-level protocols
  - RPC, HTTP, DCOM, Kerberos, Directory Services,….
  - Commands are RPCs
  - management, security,….
  - Services file/web/db/… requests
  - Managed by general-purpose OS with good dev environment
- Move apps to disk to save data movement
  - need programming environment in controller

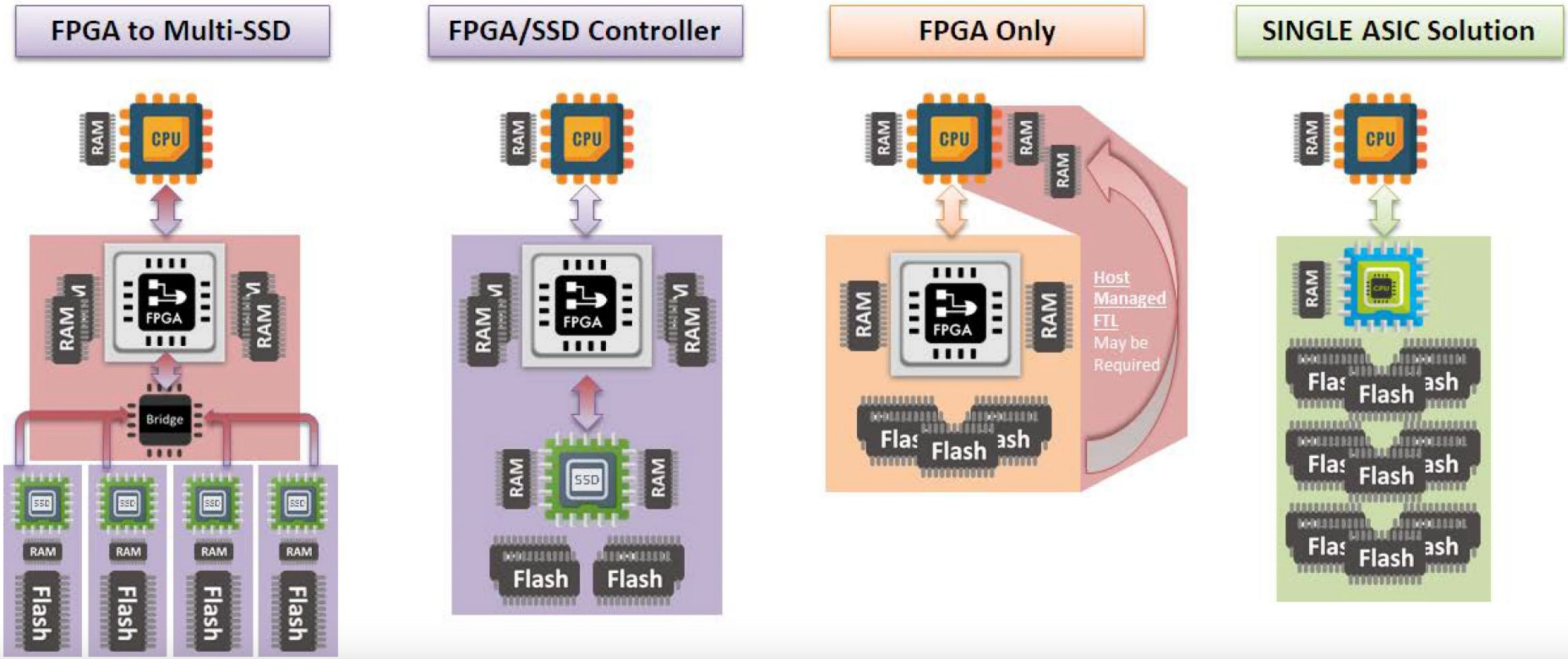
Jim Gray, NASD Talk, 6/8/98
http://jimgray.azurewebsites.net/jimgraytalks.htm

Computational storage = Computation on the IO Path

Computational storage

https://www.youtube.com/watch?v=3CeOIY1PO-Y

storage industry has defined an architecture for computational storage. NVMe will be standard.



Specialized storage interface + functionality offload

Communication Abstraction

what we get to… is a **communication** abstraction.
not just blocks, but e.g. 8MB object, transactions, …

Source: Saltzer and Kaashoek

Take-Aways

File abstraction is one of Unix enduring contribution. Beautiful example of a deep module. You should be able to describe the data structures and name mapping steps involved in file system operations.

NVMe as host/storage interface. NVMe manages completion/submission queues. NVMe namespaces include block device and zones.

With computational storage, storage devices are moving from a memory to a communication abstraction.