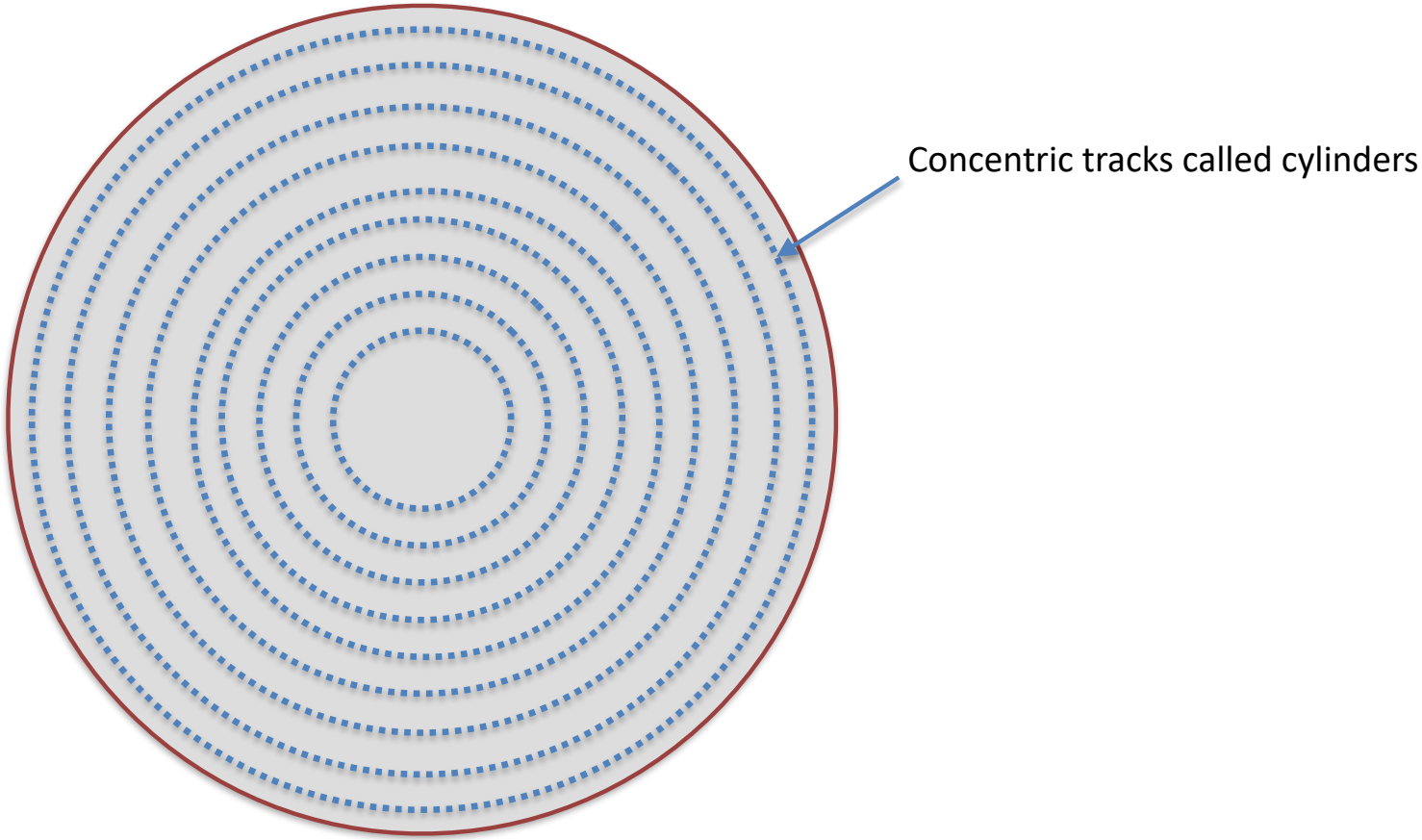
CS 310

FILESYSTEMS

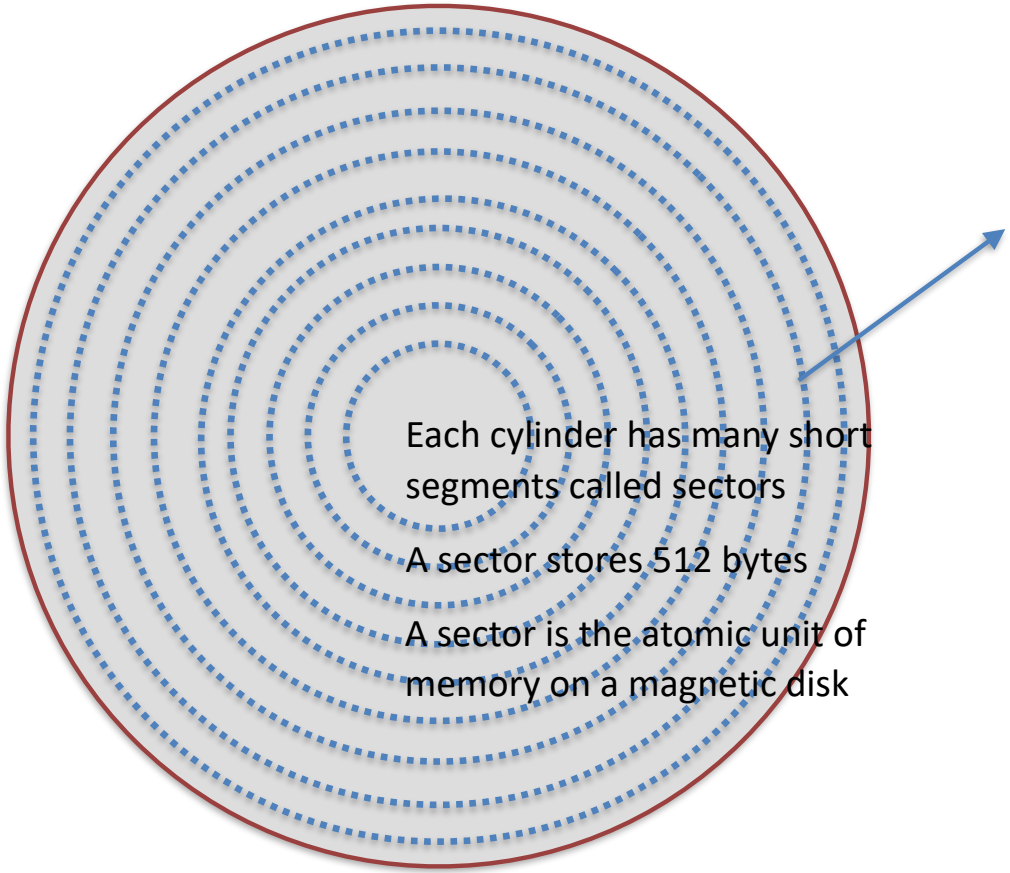
MAGNETIC DISKS


Seagate

HARD DISK PLATTER



HARD DISK PLATTER



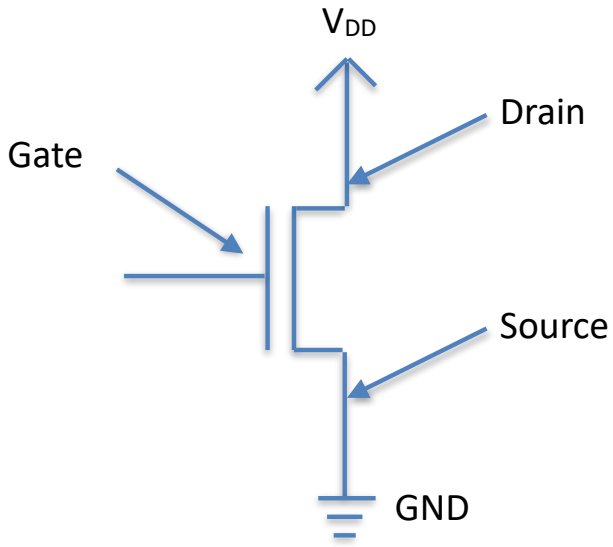
LOGICAL BLOCK ADDRESSES

Cylinder	Head	Sector	LBA
0	0	1	0
0	0	2	1
0	0	3	2
…	…	…	…
0	0	63	62
0	1	1	63
…	…	…	…

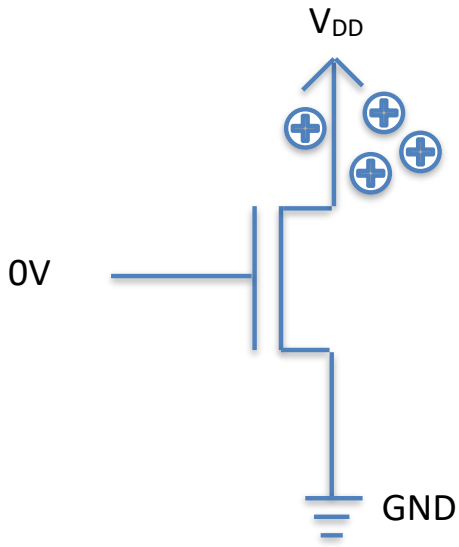
$$\mathbf{LBA} = (C \times HPC + H) \times SPT + (S - 1)$$

INTRO TO FLASH MEMORY CELLS

N-CHANNEL MOSFET

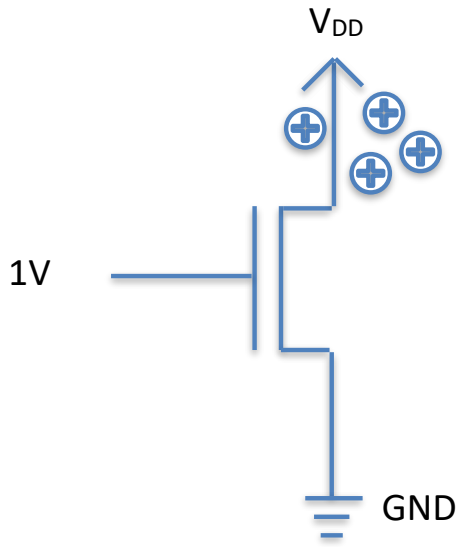


N-CHANNEL MOSFET



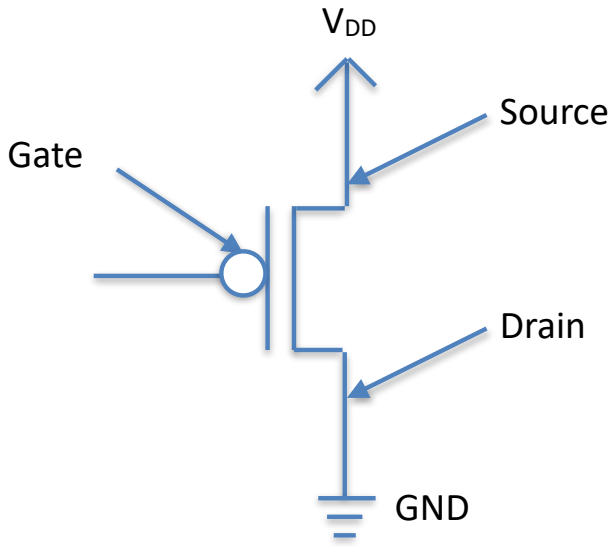
Charges get stuck @ drain

N-CHANNEL MOSFET

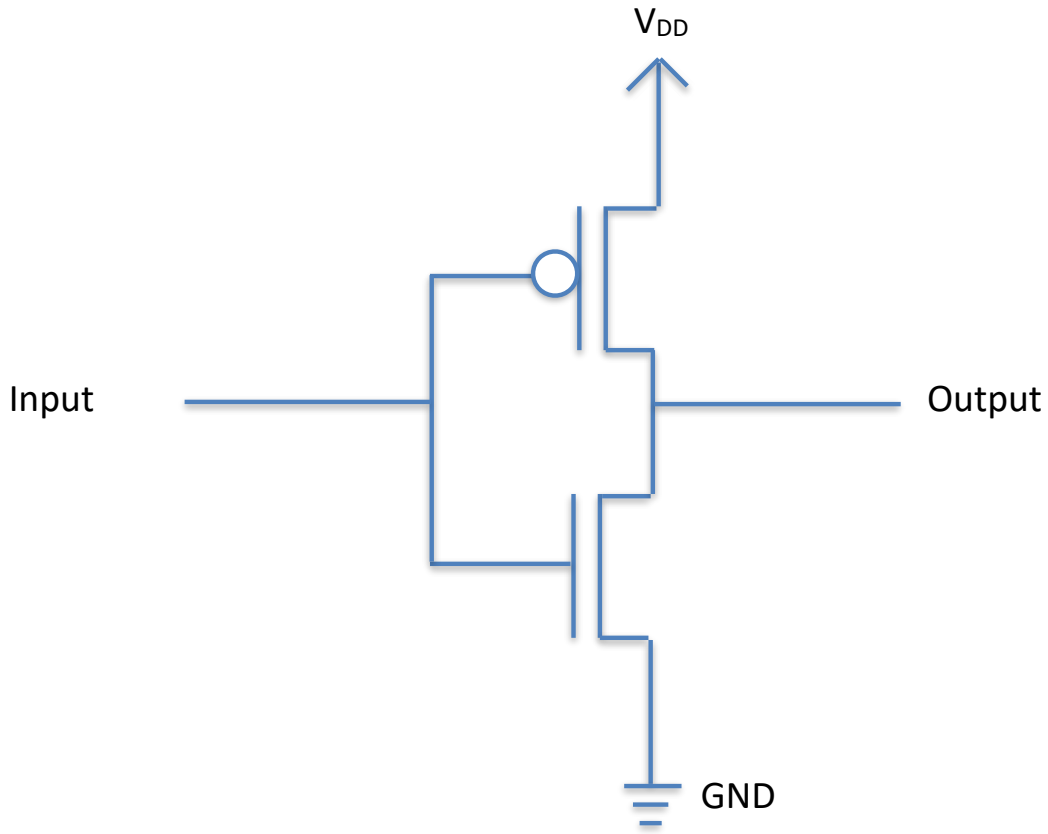


P-CHANNEL MOSFET

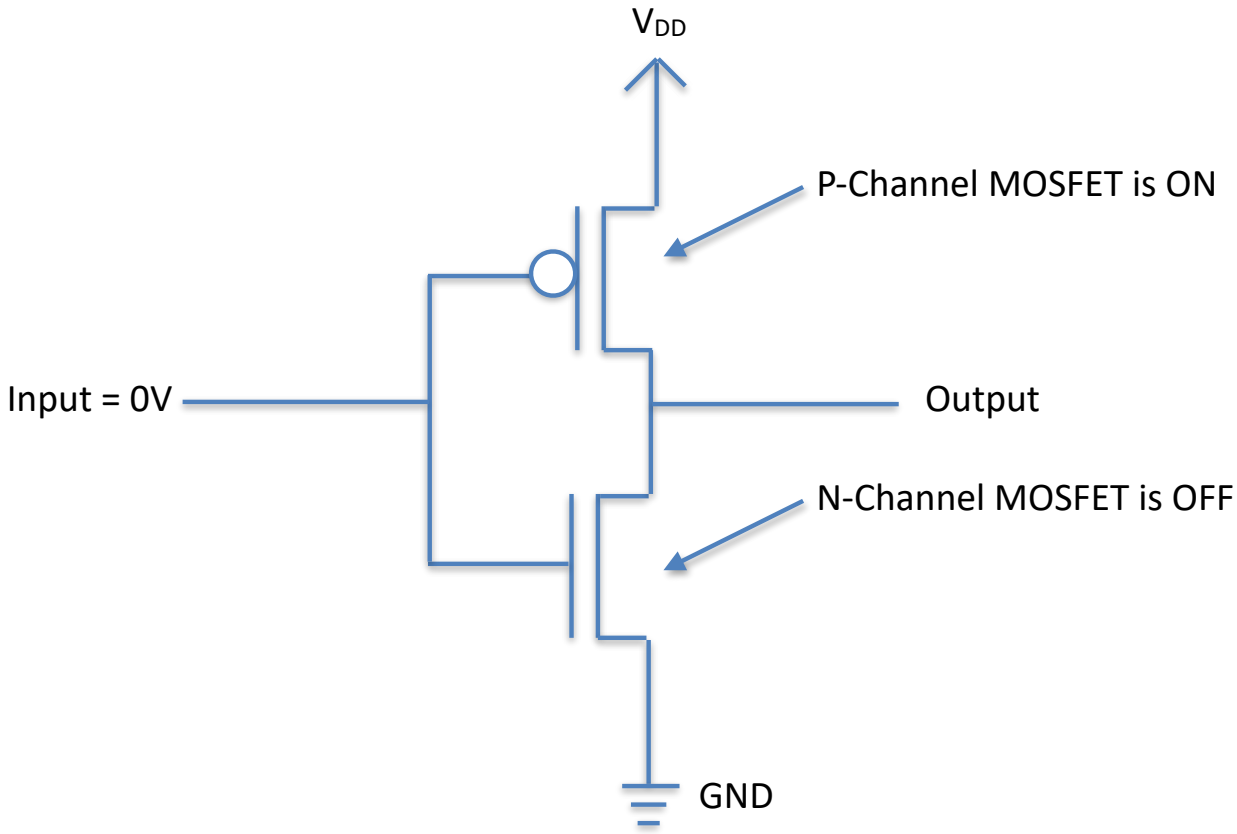
Same as N-Channel MOSFET, but you apply a voltage to turn it off.



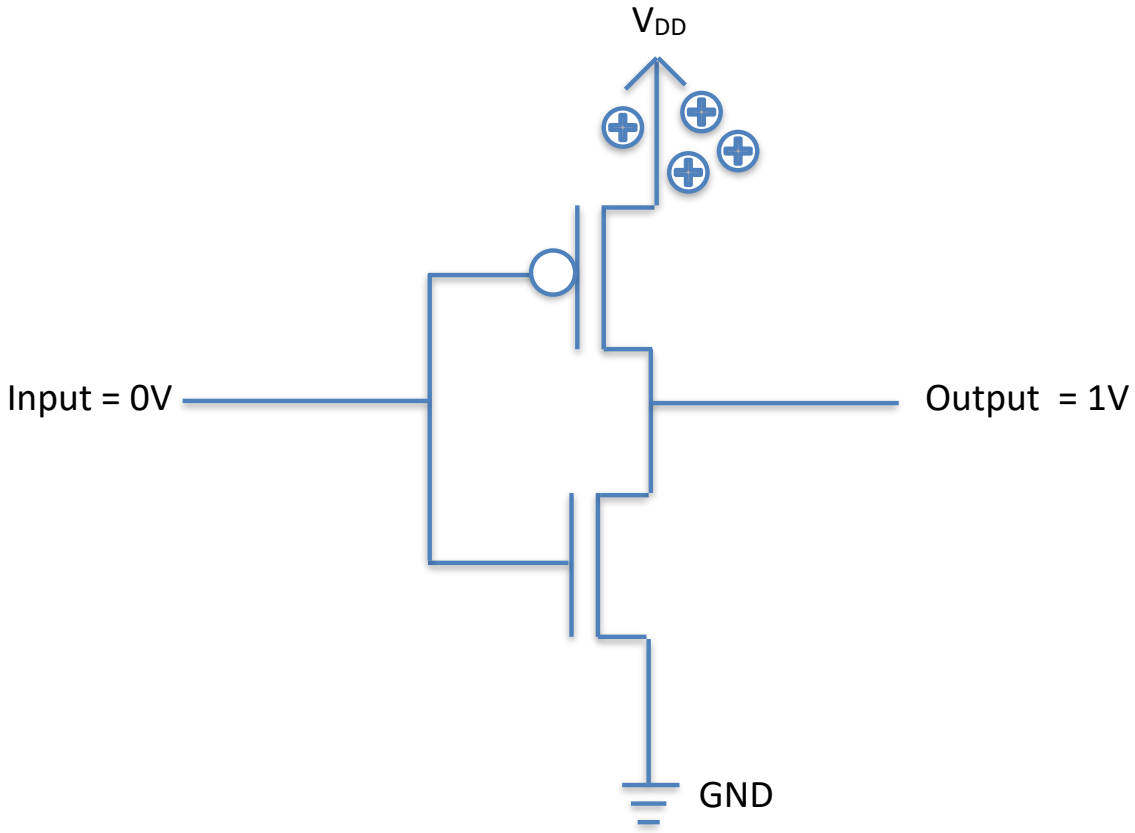
COOL THINGS YOU CAN MAKE: INVERTER



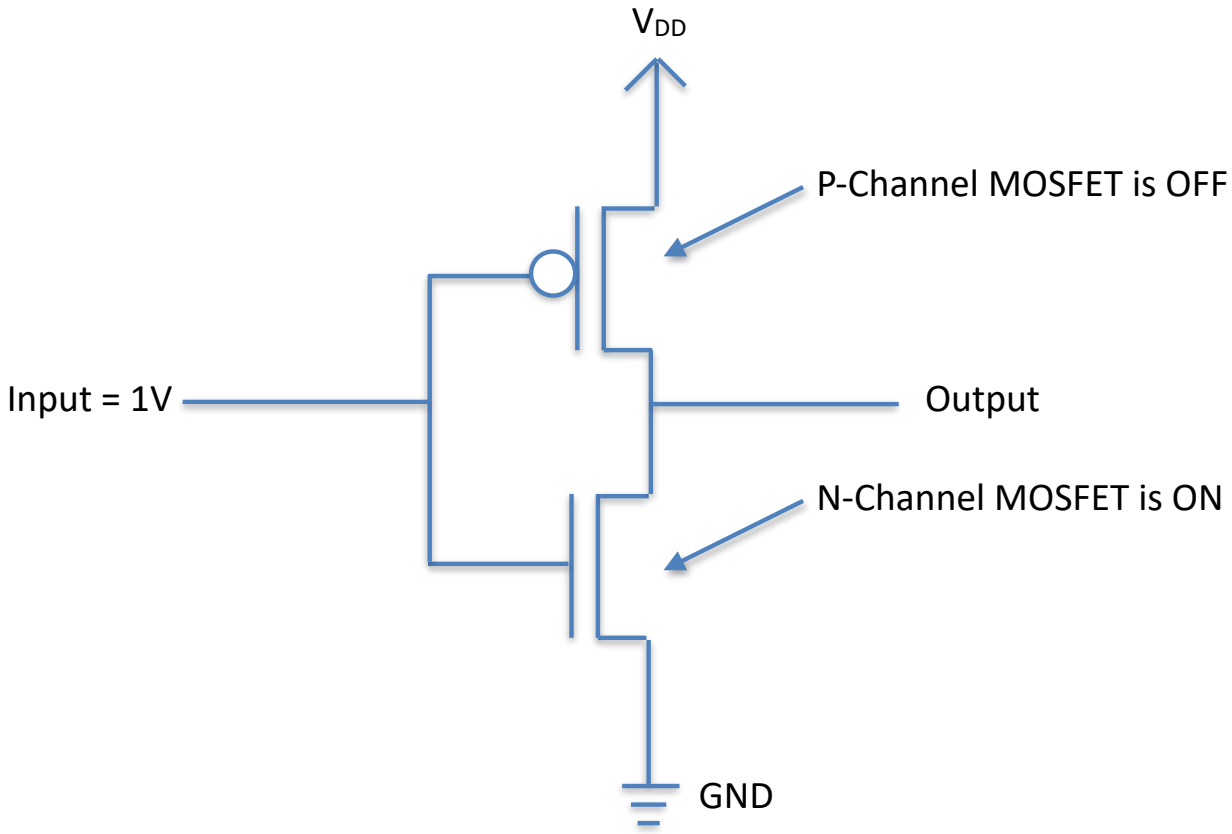
COOL THINGS YOU CAN MAKE: INVERTER



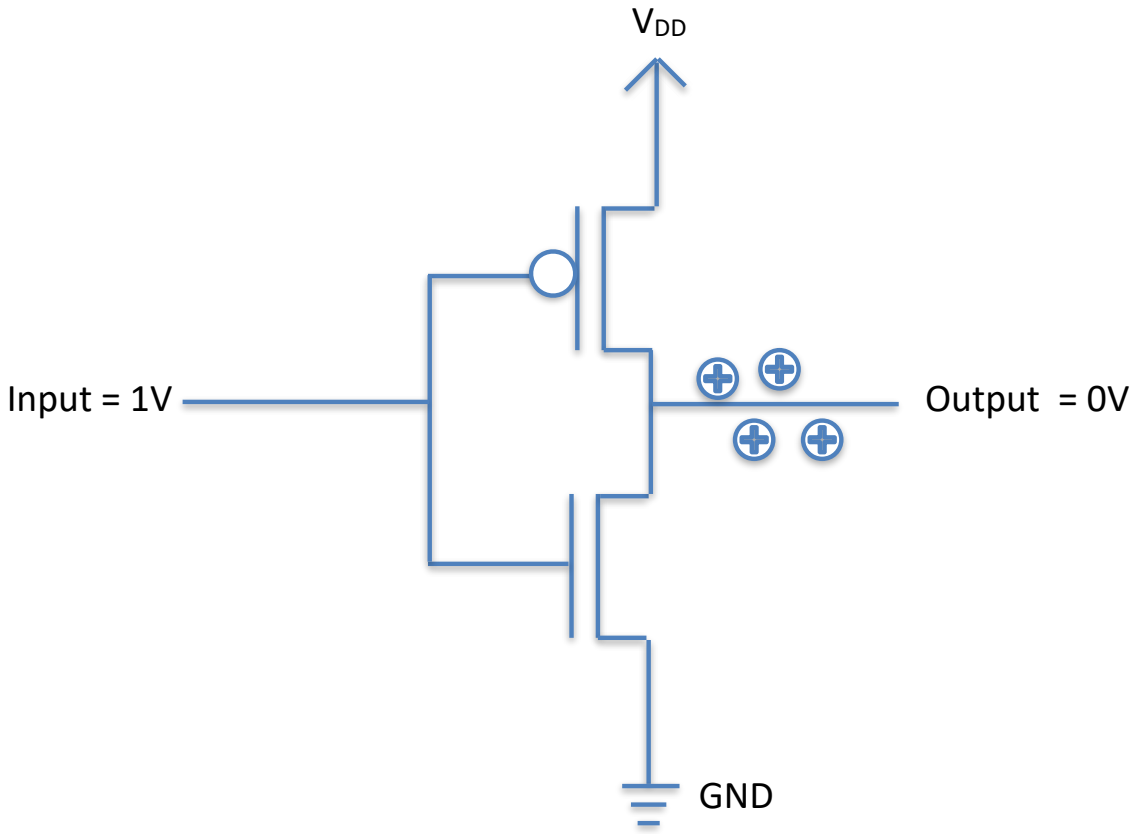
COOL THINGS YOU CAN MAKE: INVERTER



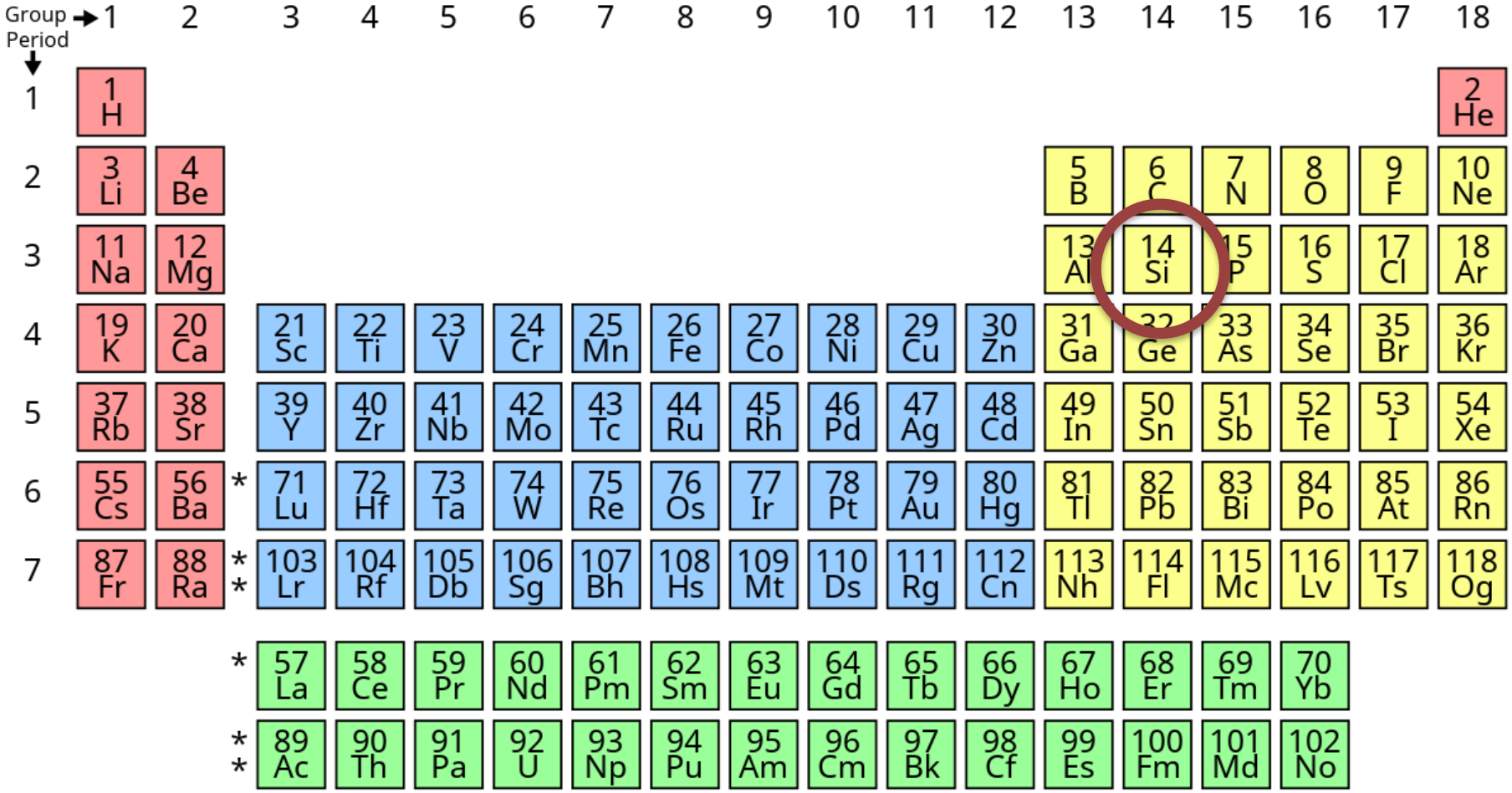
COOL THINGS YOU CAN MAKE: INVERTER



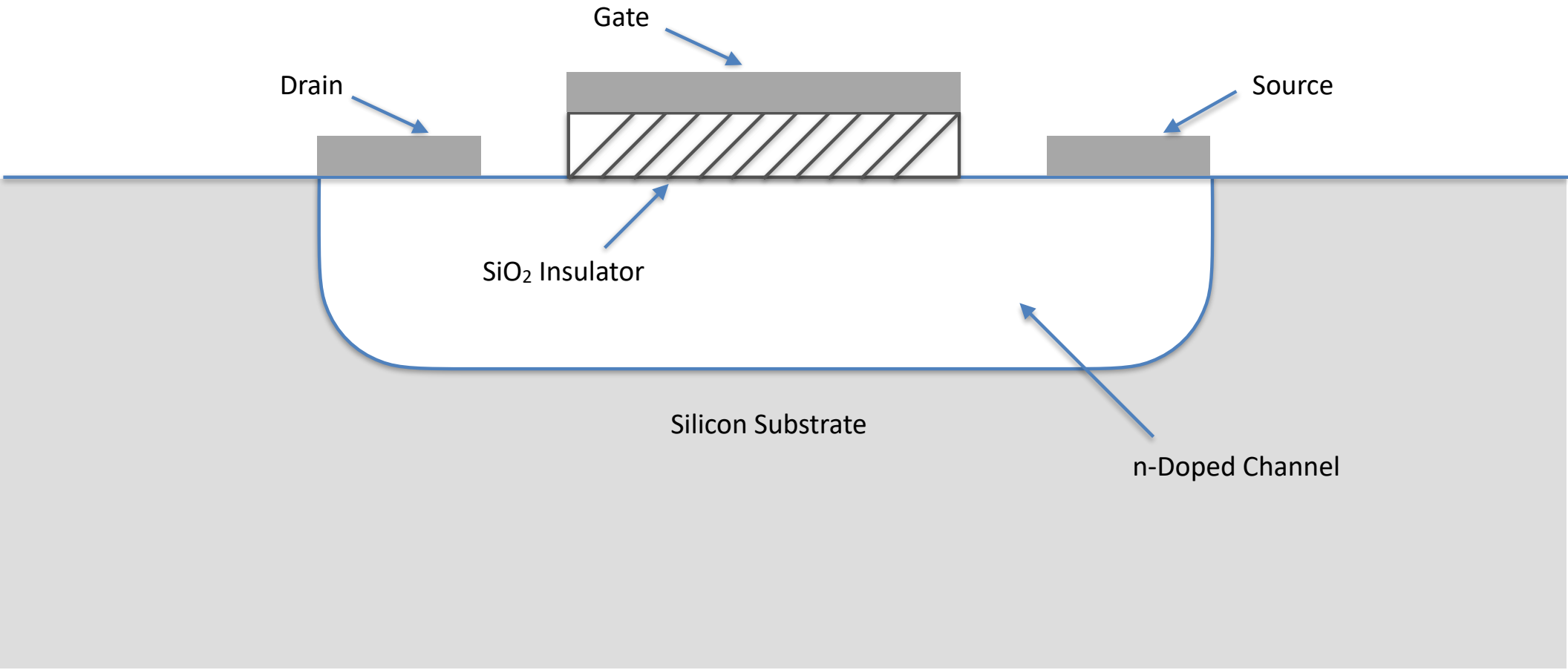
COOL THINGS YOU CAN MAKE: INVERTER



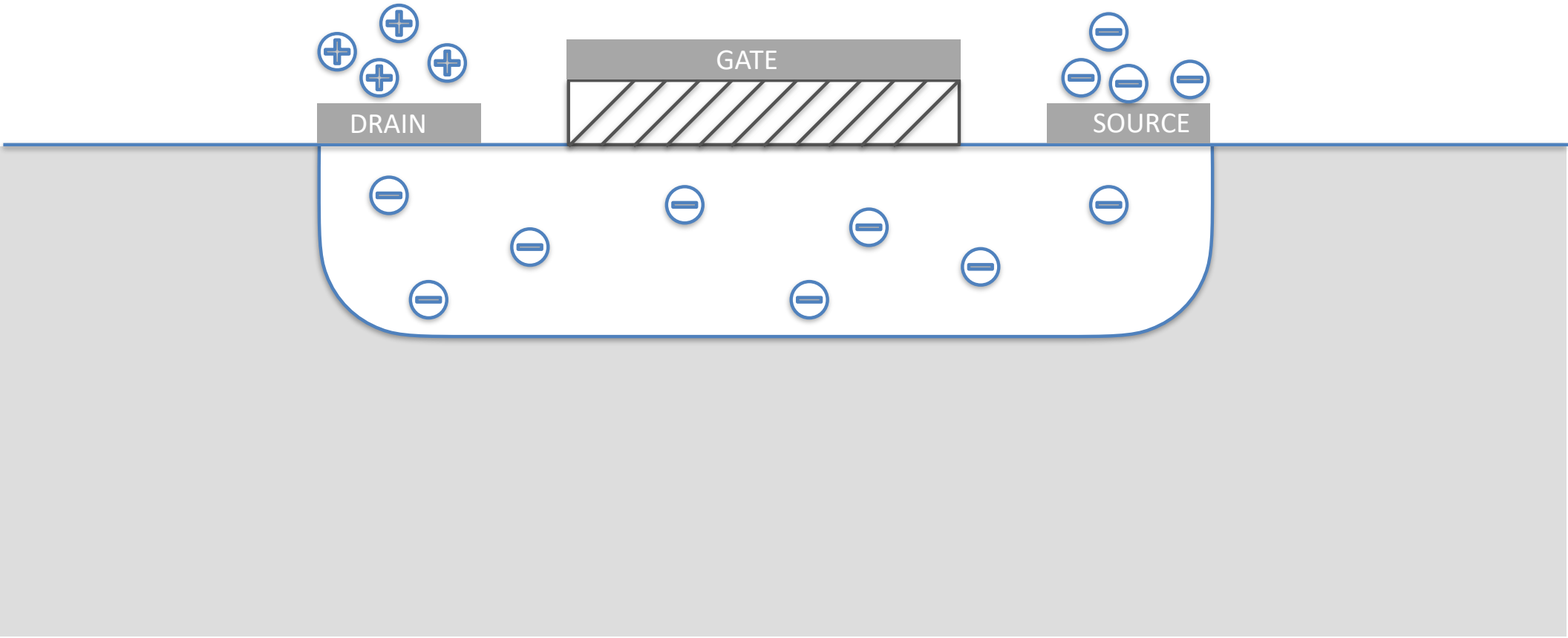
HOW AN N-CHANNEL MOSFET IS MADE



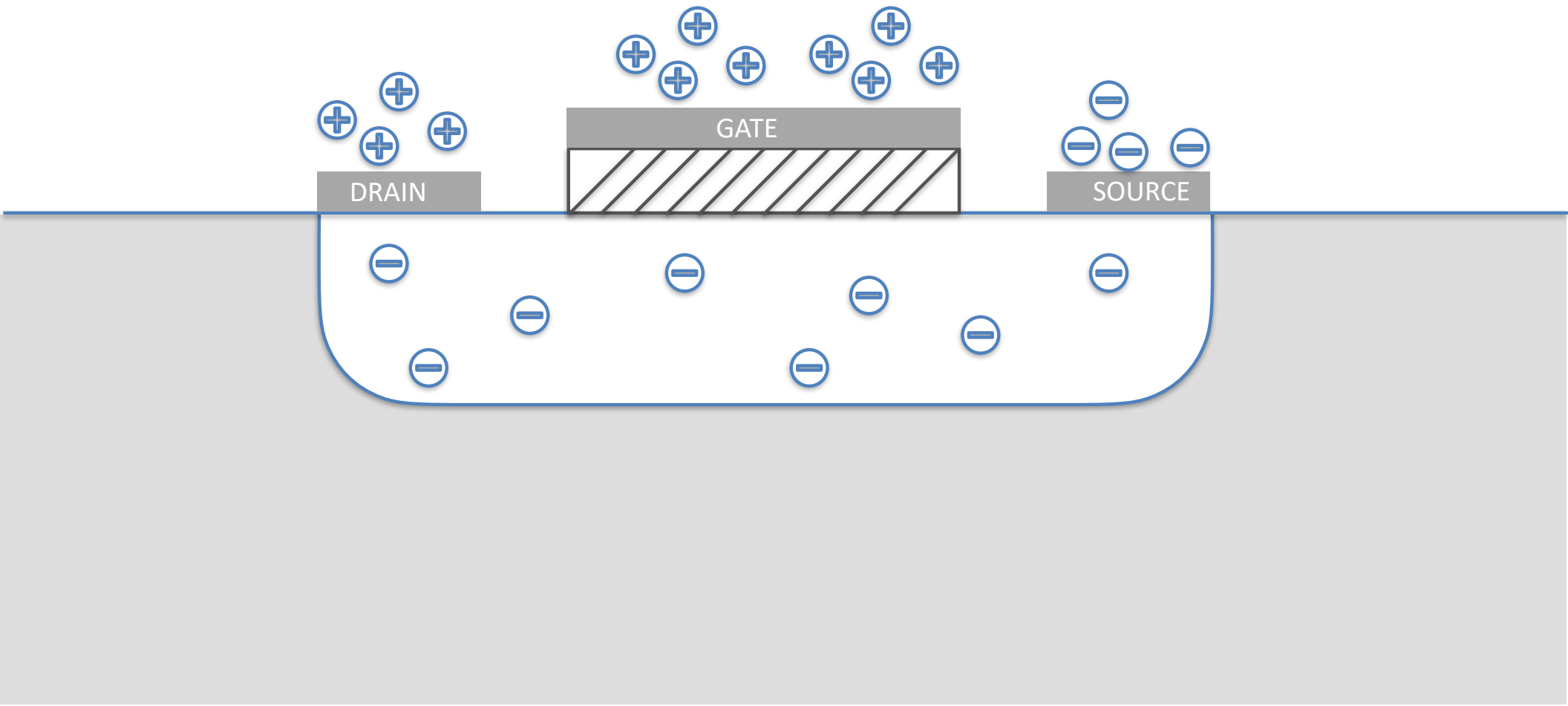
HOW AN N-CHANNEL MOSFET IS MADE

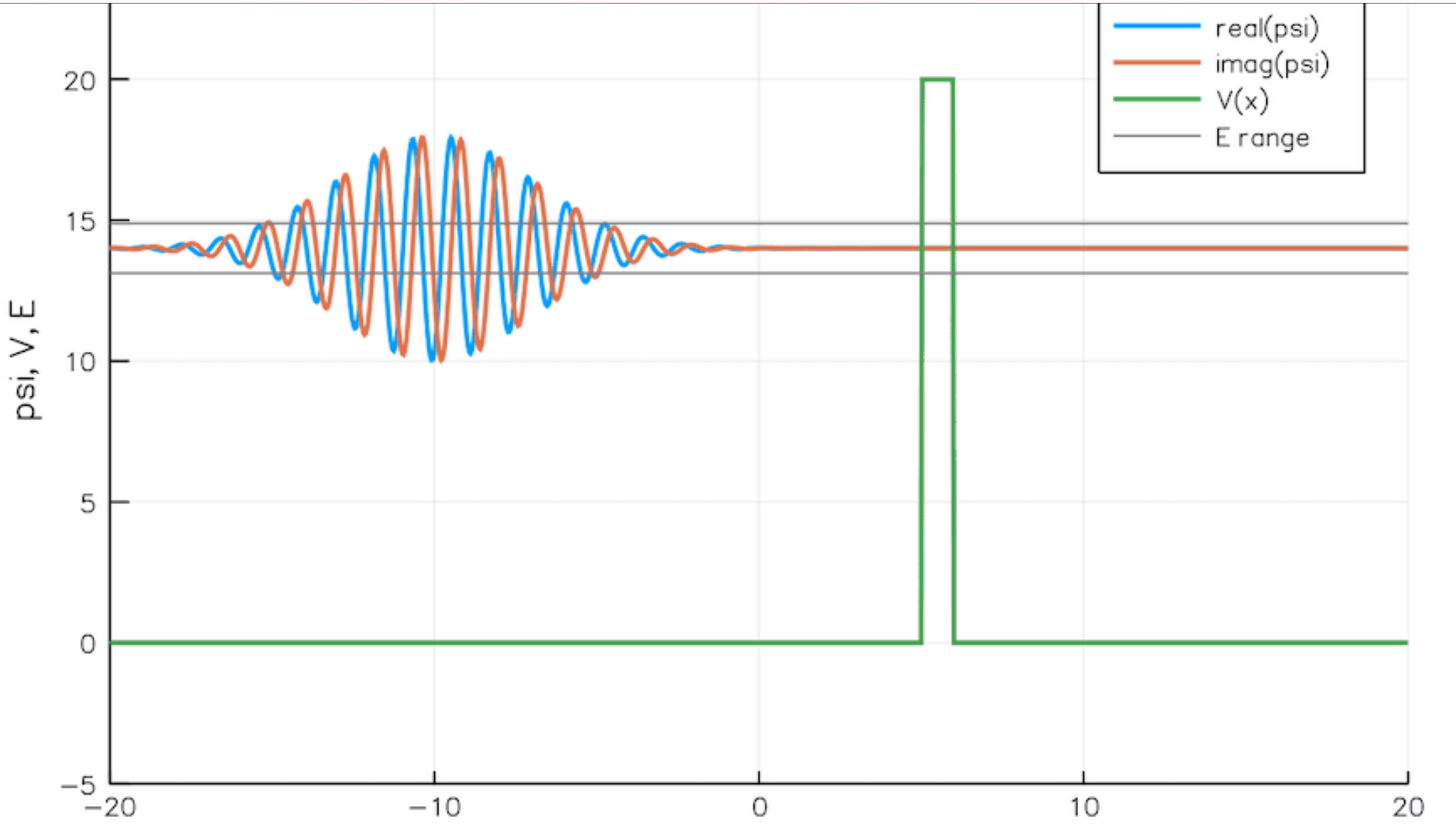


HOW AN N-CHANNEL MOSFET IS MADE

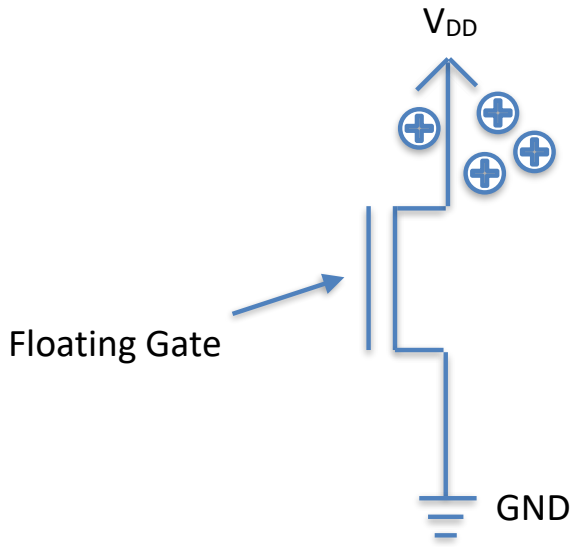


HOW AN N-CHANNEL MOSFET IS MADE

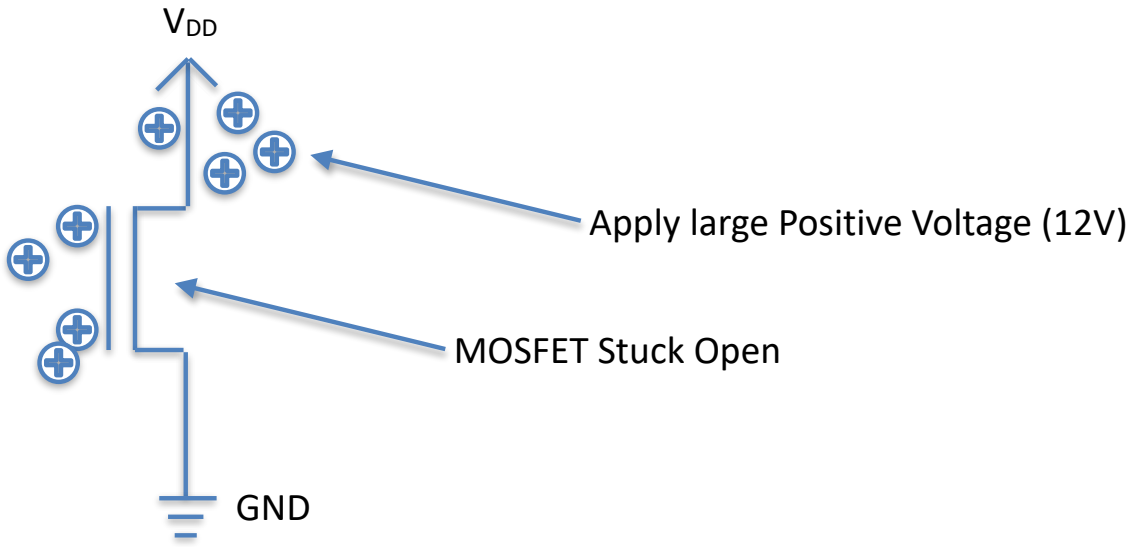



real(psi)
imag(psi)
V(x)
E range
psi, V, E
20
15
10
5
0
−5
−20
−10
0
10
20

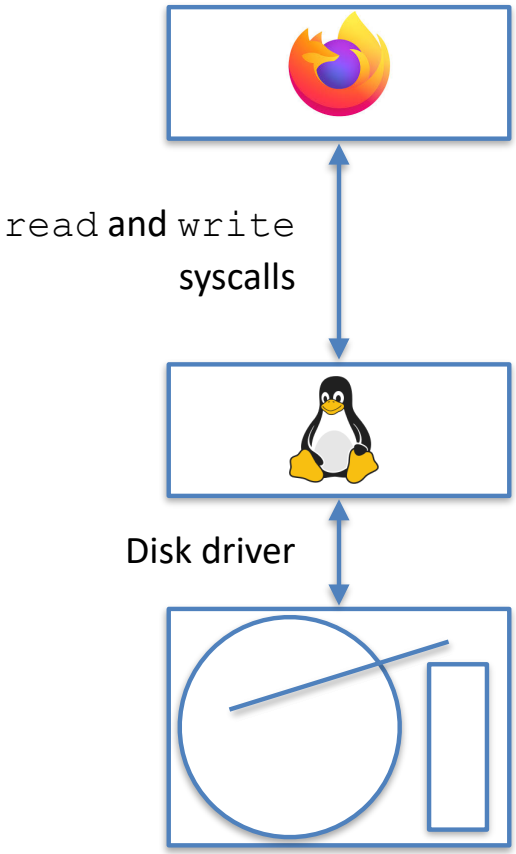
N-CHANNEL MOSFET FLASH MEMORY CELL



N-CHANNEL MOSFET FLASH MEMORY CELL



HOW FILESYSTEMS WORK



Userland: gets random access to data in files

OS: Provides filesystem abstraction so programs don't have to deal with sector-level storage

Disk: Provides storage at the granularity of a sector (512 bytes)

Have you noticed the OS has a very low opinion of you?

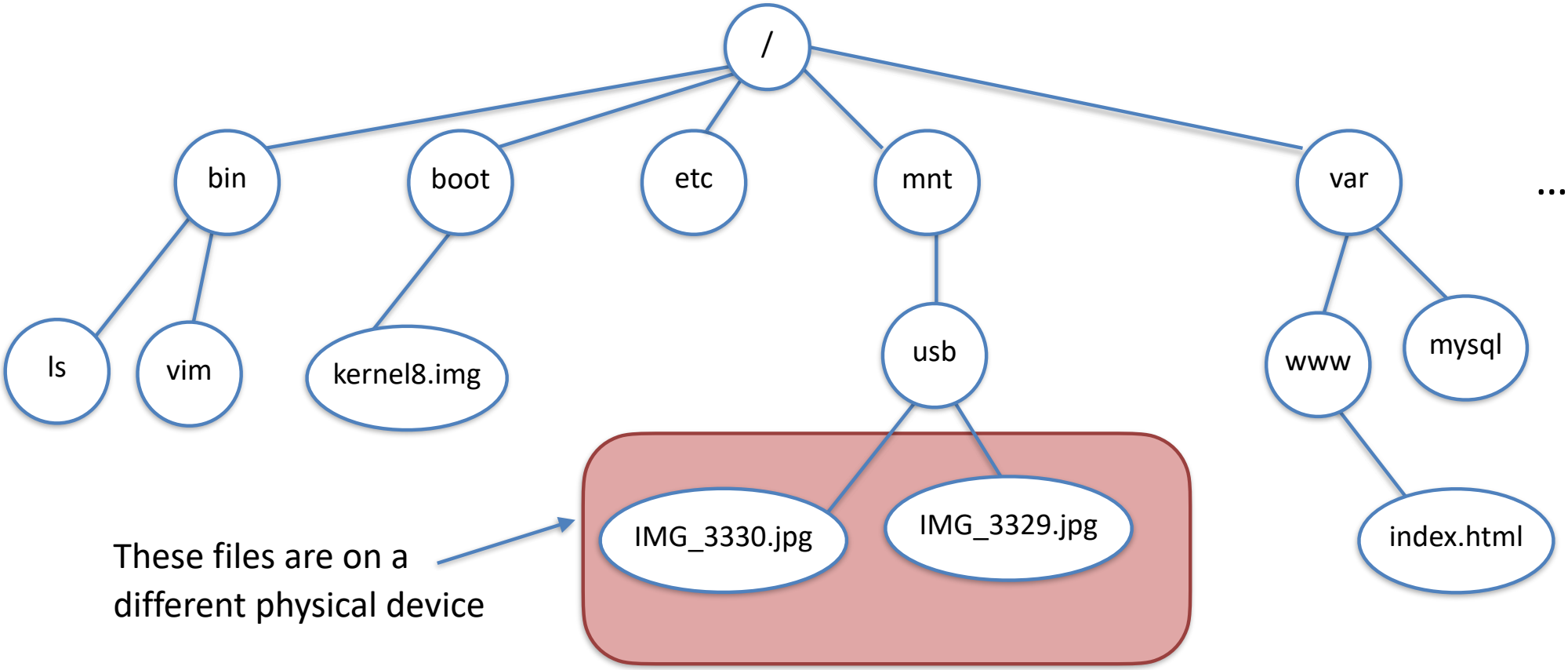
It doesn't think you can:

- manage your own memory
- keep track of your own persistent storage (files)
- deal with your own I/O
- etc. etc. etc.

Exterminate All Operating System Abstractions

Dawson R. Engler M. Frans Kaashoek
{engler, kaashoek}@lcs.mit.edu
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139

ABSTRACT INTERFACE THAT FILESYSTEMS PROVIDE

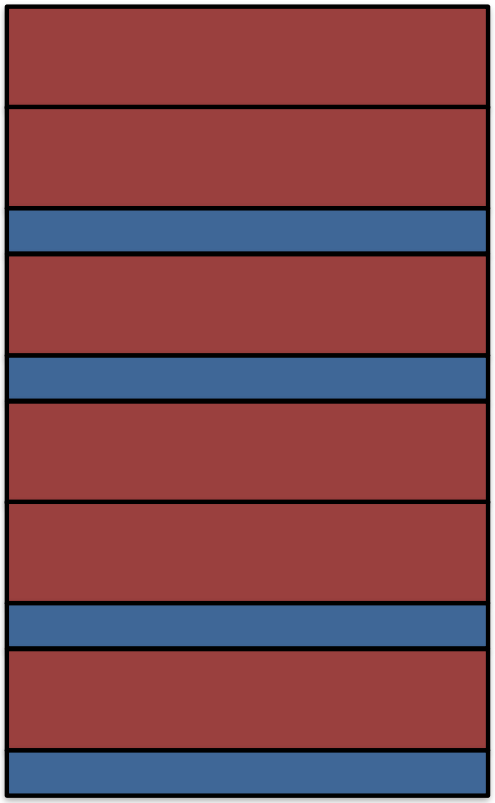


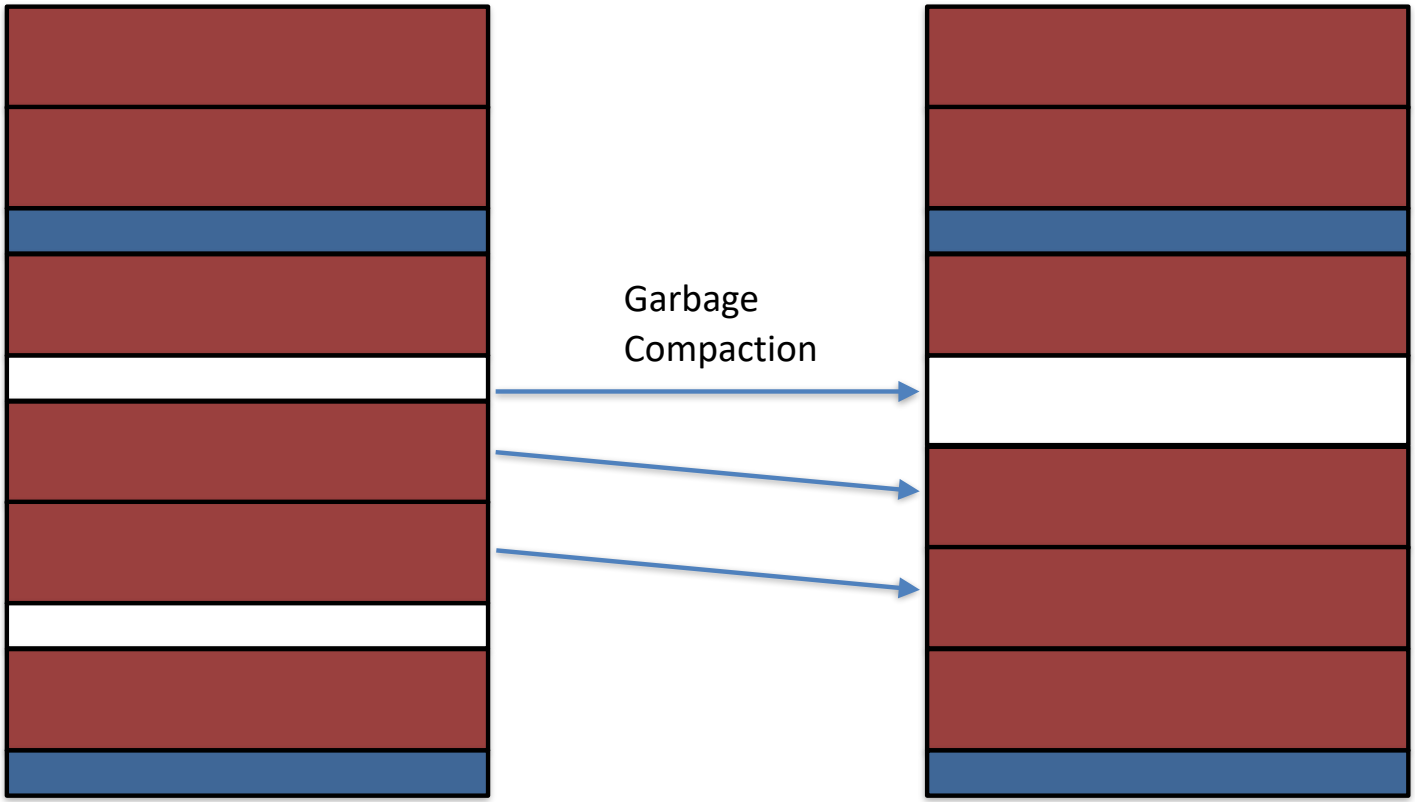
ADMINISTRIVIA

- Homework 7 Due Today
- Final Project Coming Up

ADMINISTRIVIA

- **Homework 7 Due Today**
- **Final Project Coming Up**



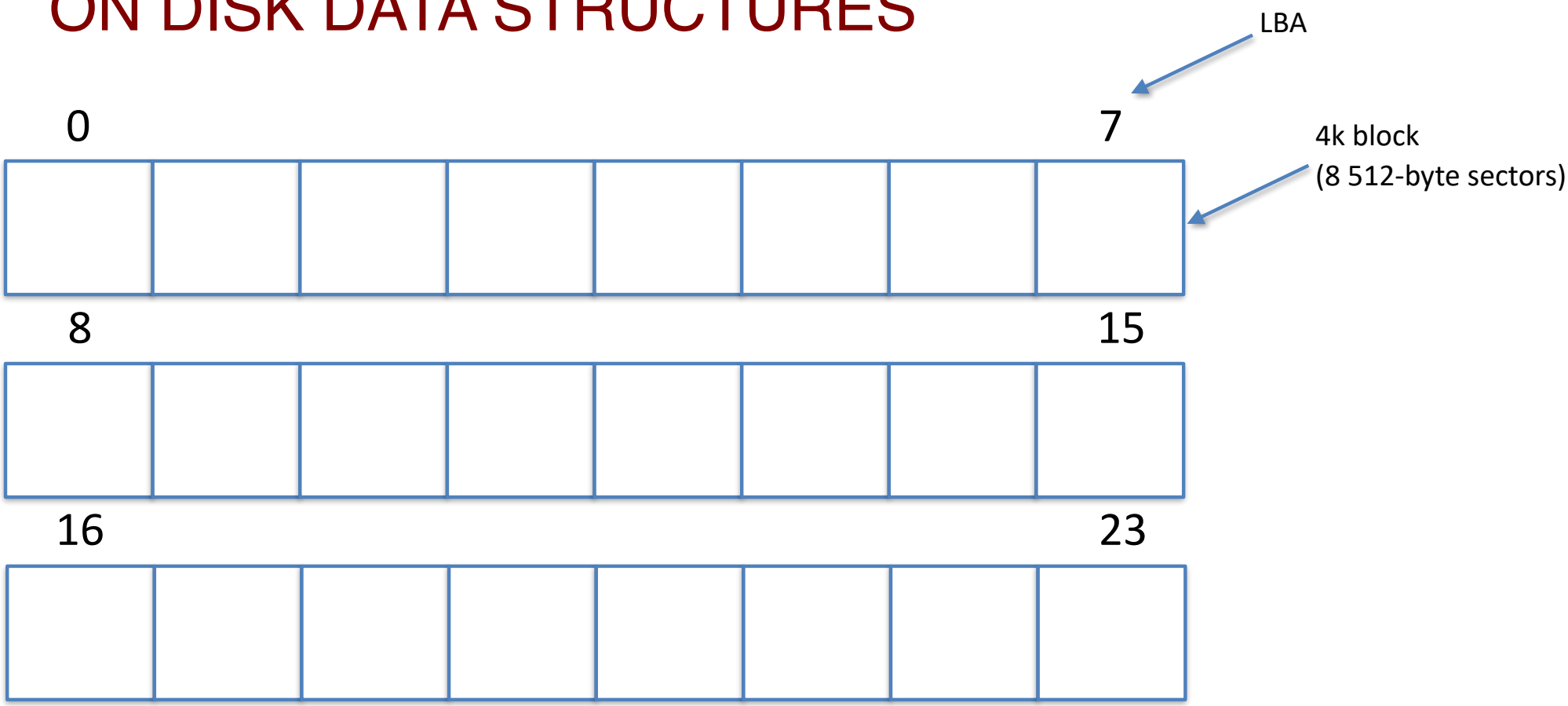

Garbage
Compaction

LINUX PERF

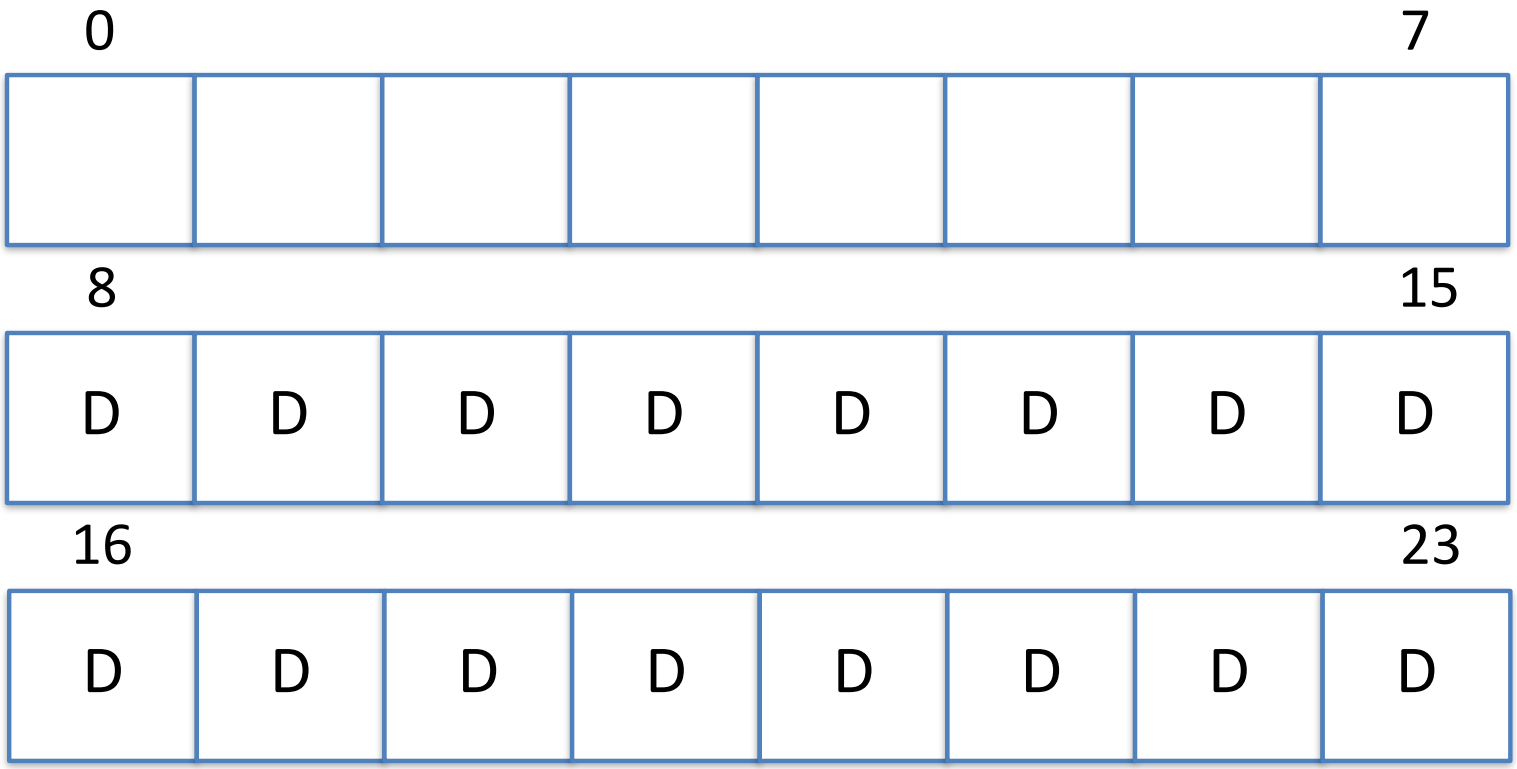
- **Tracks hardware events**

```
# perf record -e block:block_rq_issue -ag
^C
# ls -l perf.data
-rw------- 1 root root 3458162 Jan 26 03:03 perf.data
# perf report
[...]
# Samples: 2K of event 'block:block_rq_issue'
# Event count (approx.): 2216
#
# Overhead       Command      Shared Object                Symbol
# ........  ............  ..................  ......................
#
    32.13%            dd  [kernel.kallsyms]  [k] blk_peek_request
                      |
                      --- blk_peek_request
                          virtblk_request
                          __blk_run_queue
                          |
                          |--98.31%-- queue_unplugged
                          |          blk_flush_plug_list
                          |          |
                          |          |--91.00%-- blk_queue_bio
                          |          |          generic_make_request
                          |          |          submit_bio
                          |          |          ext4_io_submit
                          |          |          |
                          |          |          |--58.71%-- ext4_bio_write_page
                          |          |          |          mpage_da_submit_io
                          |          |          |          mpage_da_map_and_submit
                          |          |          |          write_cache_pages_da
                          |          |          |          ext4_da_writepages
                          |          |          |          do_writepages
                          |          |          |          __filemap_fdatawrite_range
                          |          |          |          filemap_flush
                          |          |          |          ext4_alloc_da_blocks
                          |          |          |          ext4_release_file
                          |          |          |          __fput
                          |          |          |          ____fput
                          |          |          |          task_work_run
                          |          |          |          do_notify_resume
                          |          |          |          int_signal
                          |          |          |          close
                          |          |          |          0x0
                          |          |          |
                          |          |          --41.29%-- mpage_da_submit_io
[...]
```

ON DISK DATA STRUCTURES



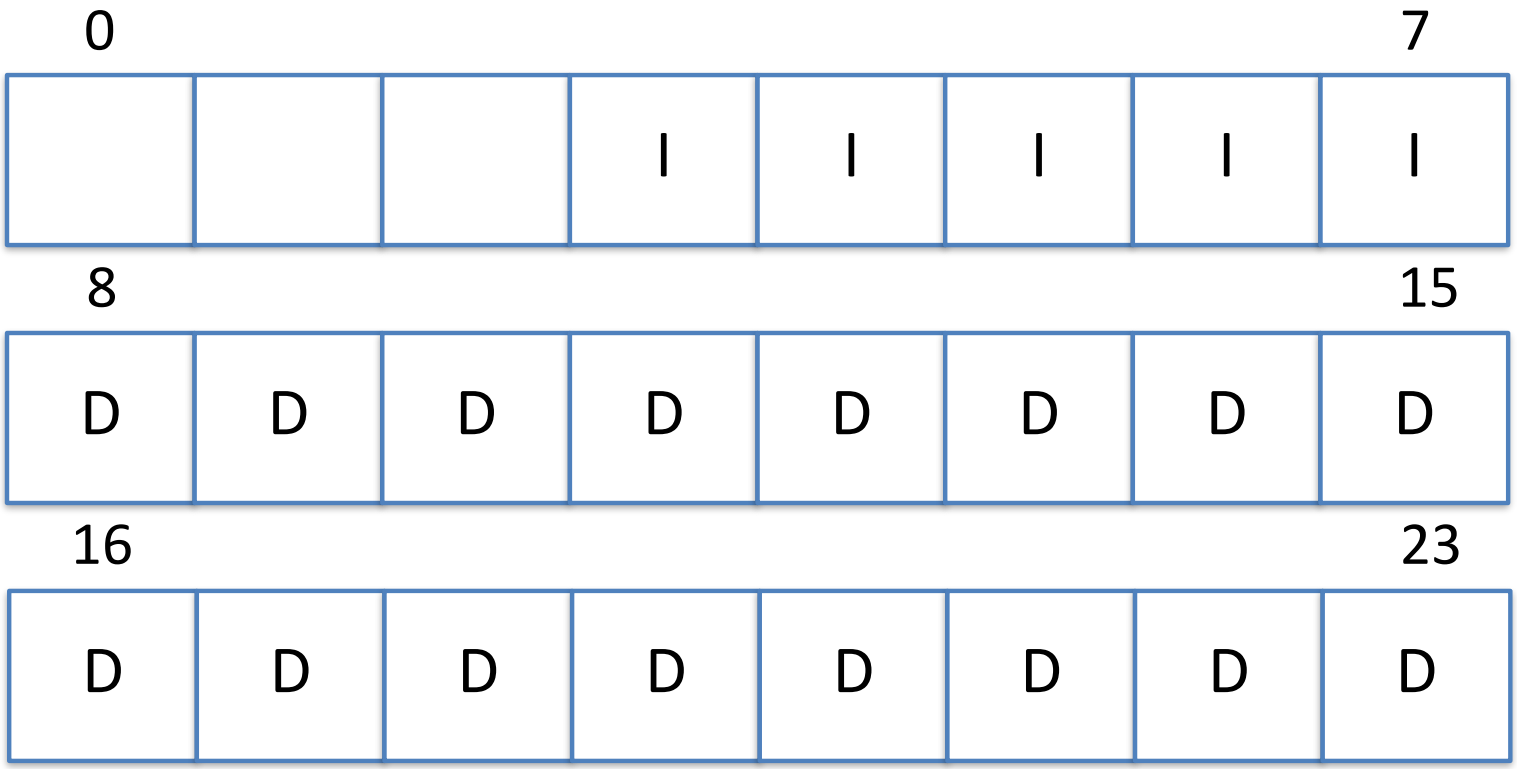
ON DISK DATA STRUCTURES



Data Region consists of blocks that can be allocated for file data.

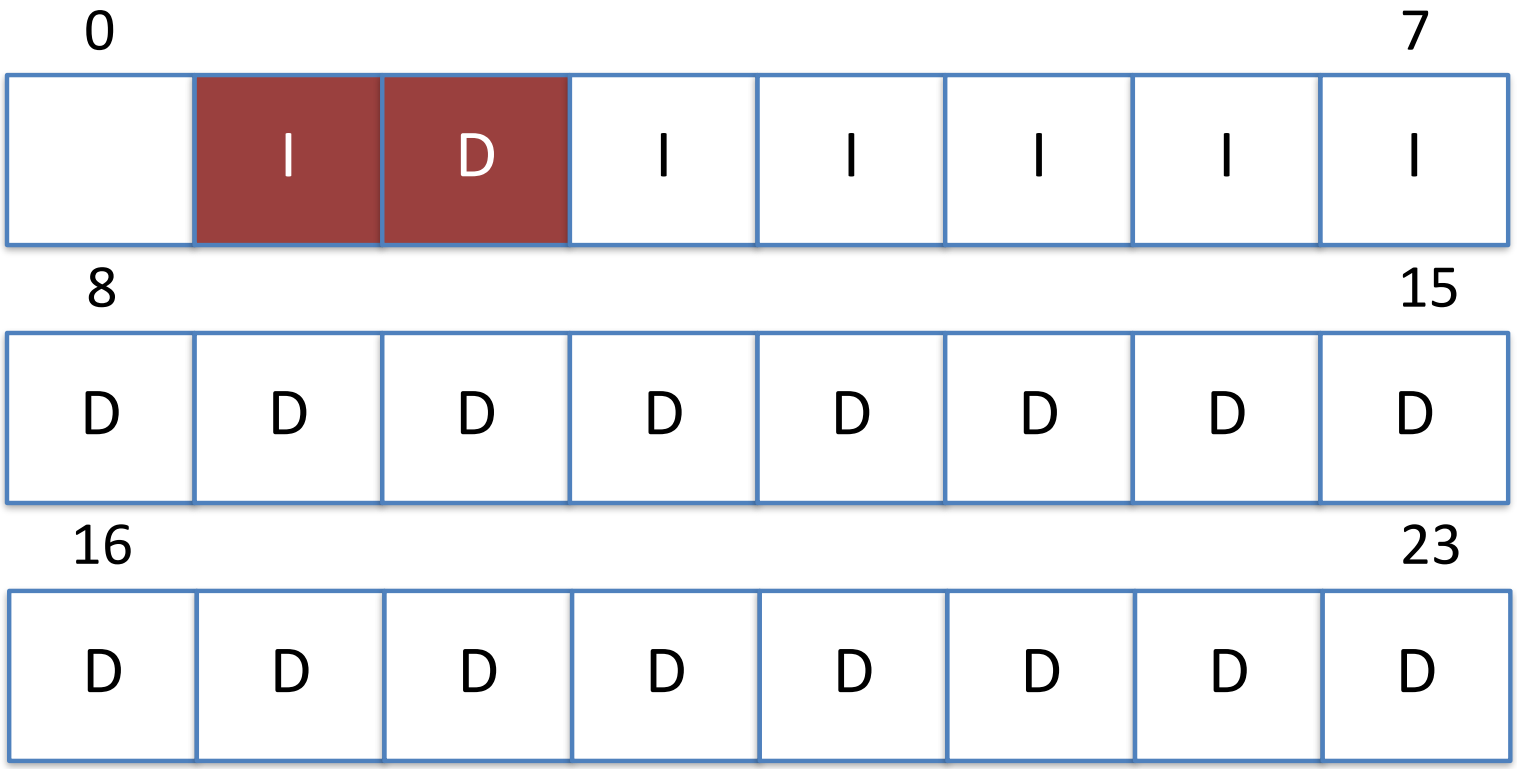
Data blocks can't be subdivided for small files.

ON DISK DATA STRUCTURES



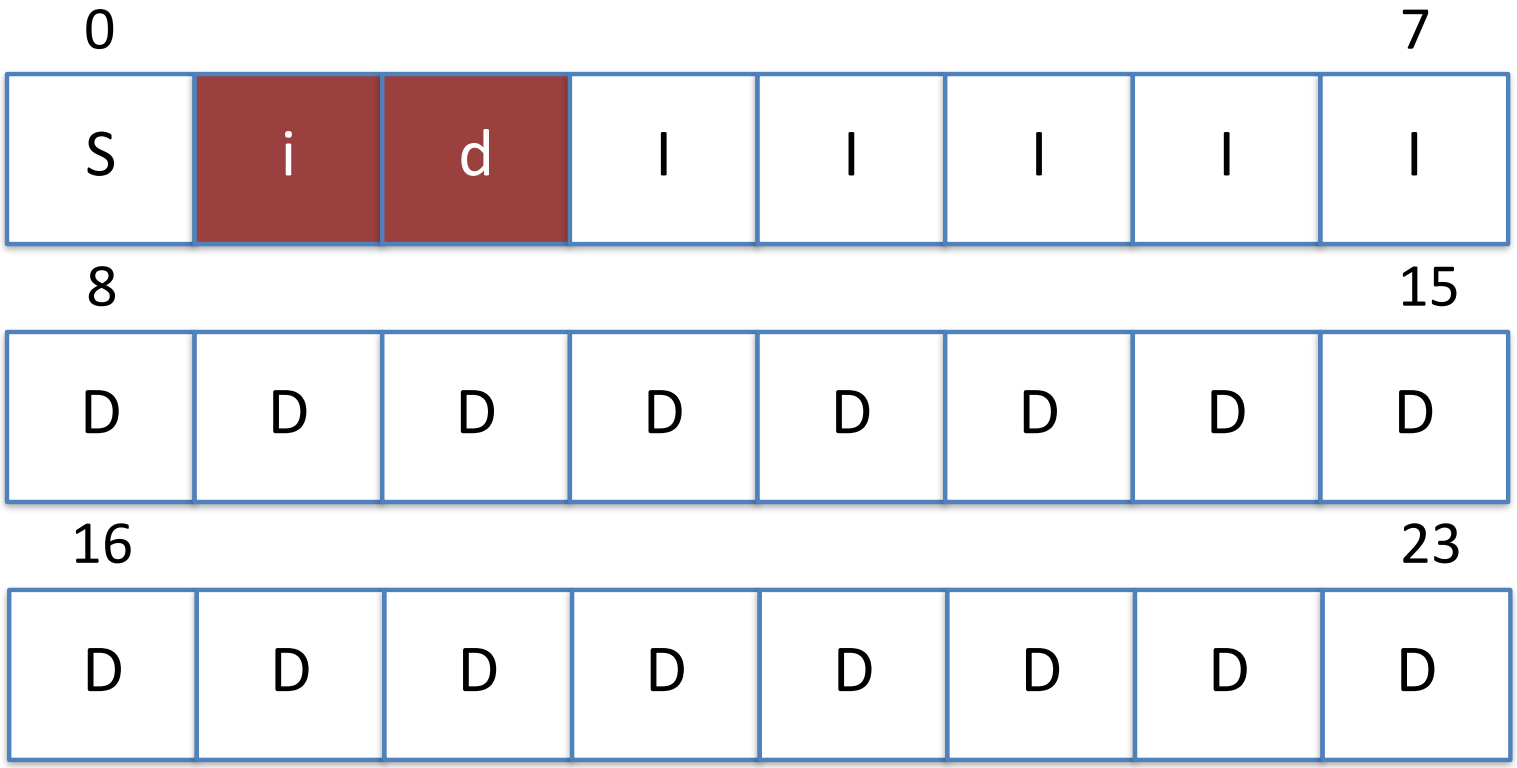
An inode tells us where to find the data blocks for a particular file.

ON DISK DATA STRUCTURES



Data bitmap and inode bitmaps tell us which data blocks and inode blocks are available.

ON DISK DATA STRUCTURES



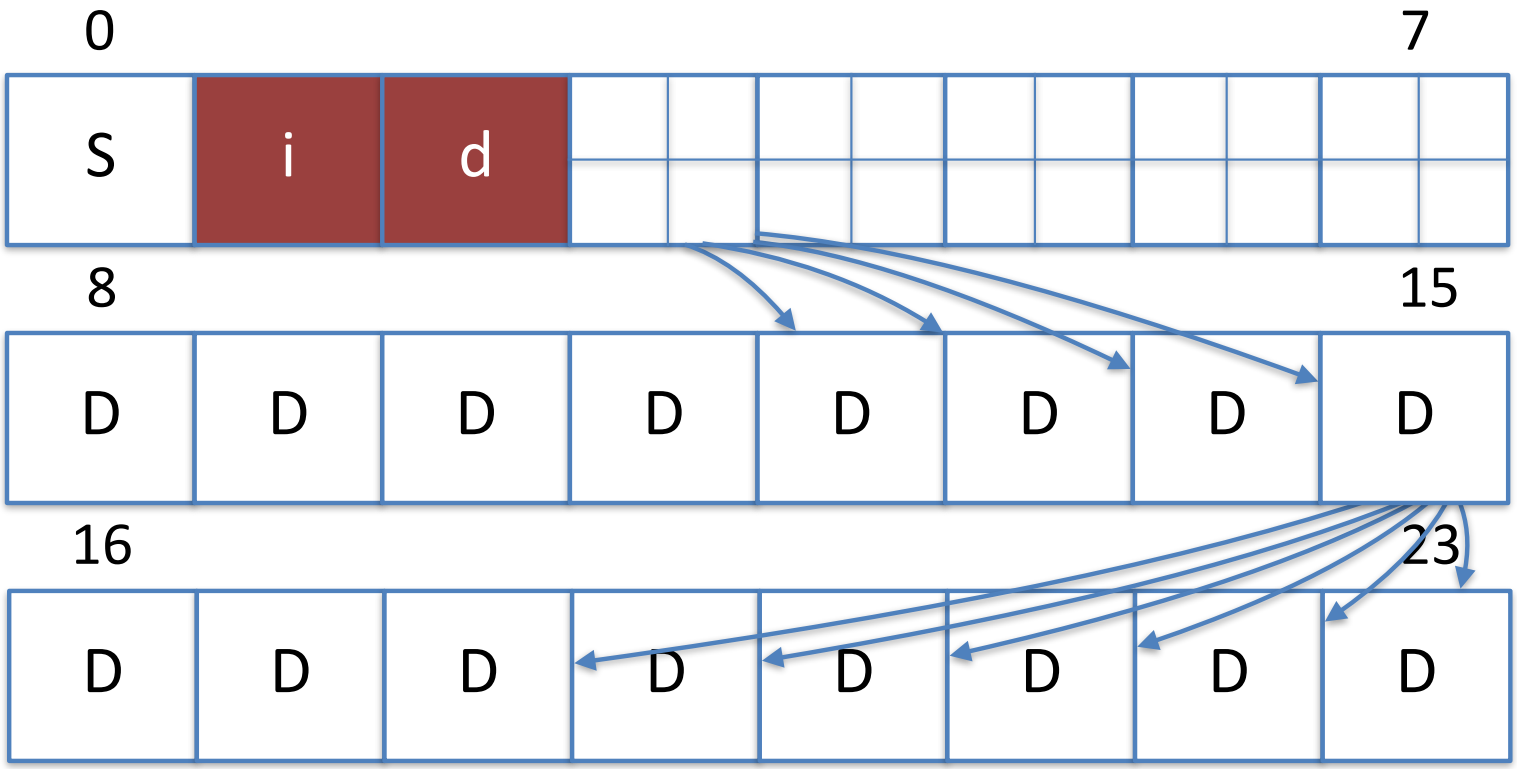
Superblock tells us parameters of the filesystem like how many inode blocks and data blocks there are and where to find the root inode.

When mounting a filesystem, the OS always reads the superblock first to find out where all the other data structures are.

INODES

Size	Name	Description
2 bytes	mode	can this file be read/written/executed?
2 bytes	uid	owner of this file
4 bytes	size	How many bytes in this file?
4 bytes	time	Time this file was last accessed
4 bytes	ctime	Time this file was created
4 bytes	mtime	Time this file was modified
4 bytes	dtime	What time was this inode deleted?
2 bytes	gid	Group that owns this file
2 bytes	links_count	How many hard links to this file
4 bytes	blocks	How many blocks allocated to this file
4 bytes	flags	How should ext2 use this inode?
4 bytes	osd1	Available for use by OS
60 bytes	block	Set of 15 disk pointers
4 bytes	generation	file version (used by NFS)
4 bytes	file_acl	used for permissions
4 bytes	dir_acl	permissions…

CREATING LARGE FILES

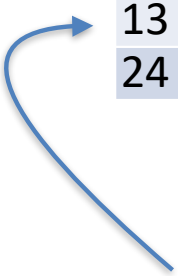


If an inode can only point to 4 data blocks, max file size is 16 kbytes (4 * 4k).

Multi-level index uses data blocks to hold extra index pointers.

If each data block can hold 1024 pointers, max file size with inode + 1 data block is (4+1024)*4k = 4112 k

inum	record length	string length	file name
5	12	2	.
2	12	3	..
12	12	4	foo
13	12	4	bar
24	36	29	foo_bar_version_12_27_20.txt



Each Row is called a directory entry

HARD LINK: MAKE ANOTHER DIRECTORY ENTRY POINT TO SAME INODE

inum	record length	string length	file name
5	12	2	.
2	12	3	..
12	12	4	foo
13	12	4	bar
24	36	29	foo_bar_version_12_27_20.txt
12	**15**	**9**	**foo_link**

foo_link and foo both point to inode 12

Limitations of hard links:

1. You can't create hard links to directories (to prevent cycles).
2. You can't create a hard link to a file on another partition.

SYMLINKS/SOFT LINKS/SYMBOLIC LINKS

inum	record length	string length	file name
5	12	2	.
2	12	3	..
12	12	4	foo
13	12	4	bar
24	36	29	foo_bar_version_12_27_20.txt
12	**15**	**9**	**foo_link**

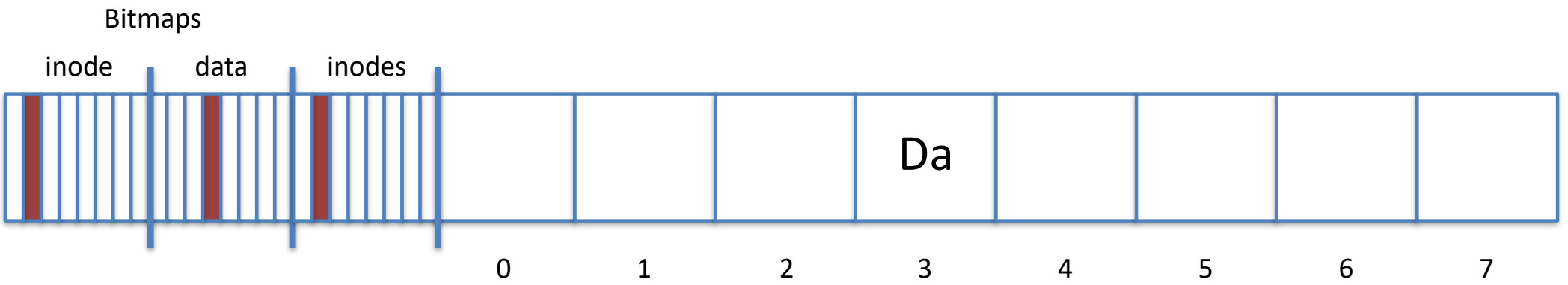
foo_link and foo both point to inode 12

Limitations of hard links:

1. You can't create hard links to directories (to prevent cycles).
2. You can't create a hard link to a file on another partition.

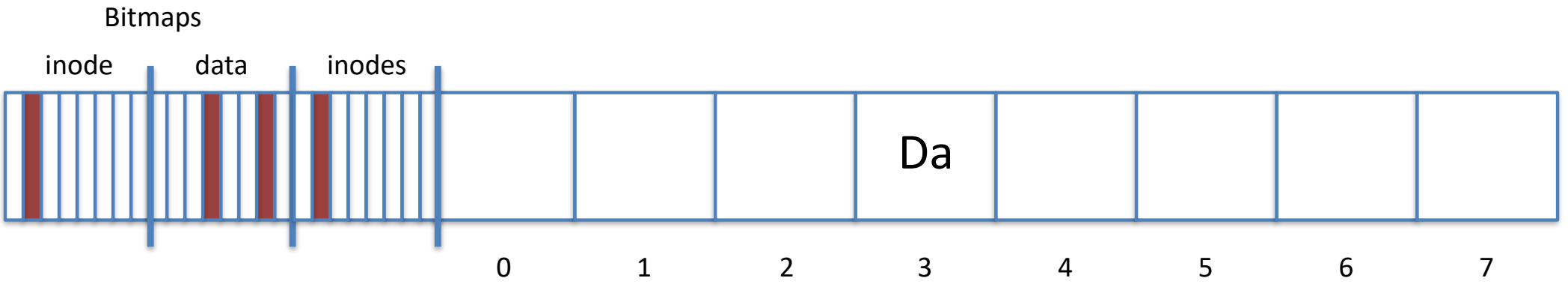
JOURNALING AND WRITE INCONSISTENCIES

WHAT HAPPENS WHEN POWER FAILS MID-WRITE?



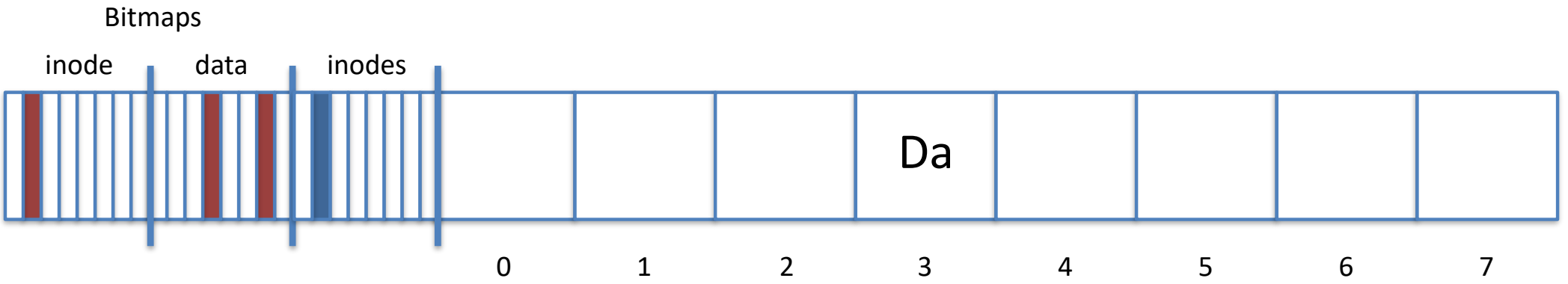
SUPPOSE WE WANT TO APPEND A 4K BLOCK TO FILE

1. Allocate a data block from the data bitmap.



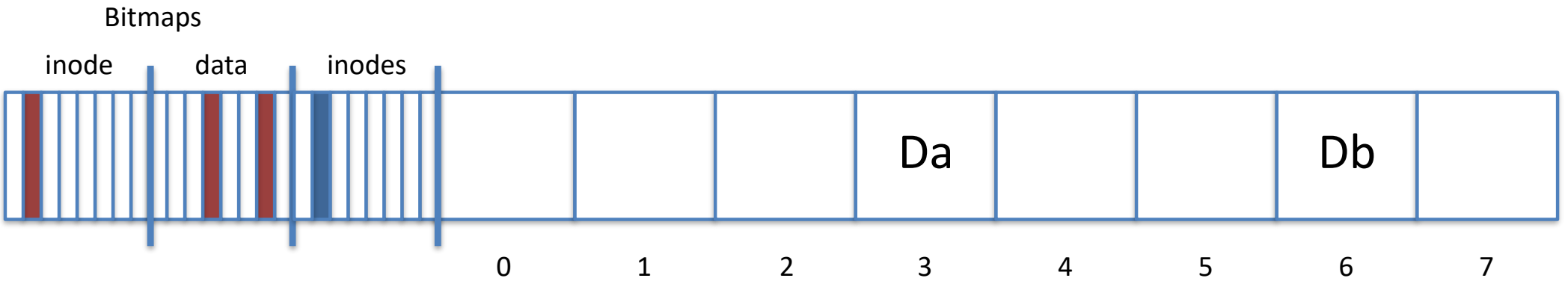
SUPPOSE WE WANT TO APPEND A 4K BLOCK TO FILE

1. Allocate a data block from the data bitmap.
2. Set the direct pointer in the file's inode to point to the new data block.



SUPPOSE WE WANT TO APPEND A 4K BLOCK TO FILE

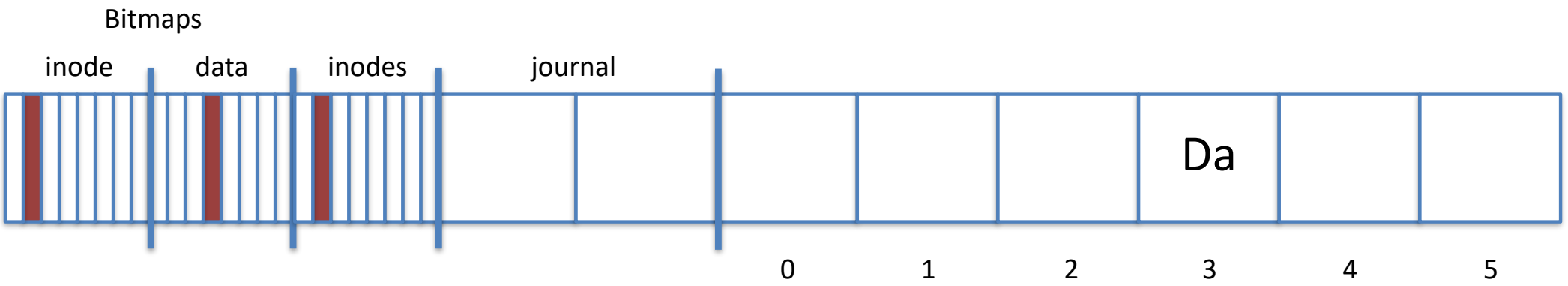
1. Allocate a data block from the data bitmap.
2. Set the direct pointer in the file's inode to point to the new data block.
3. Write the new data block.



POSSIBLE FAILURE SCENARIOS

1. Just the data block gets written, not the inode or bitmap
2. Just the inode gets written, not data or bitmap (inconsistency)
3. Just the bitmap gets written, not inode or data (inconsistency)
4. inode and bitmap are written, but not data (garbage data)
5. inode and data get written, but not bitmap (inconsistency)
6. bitmap and data get written but no inode (inconsistency)

JOURNALING ALLOWS US TO RECOVER



JOURNALING ALLOWS US TO RECOVER

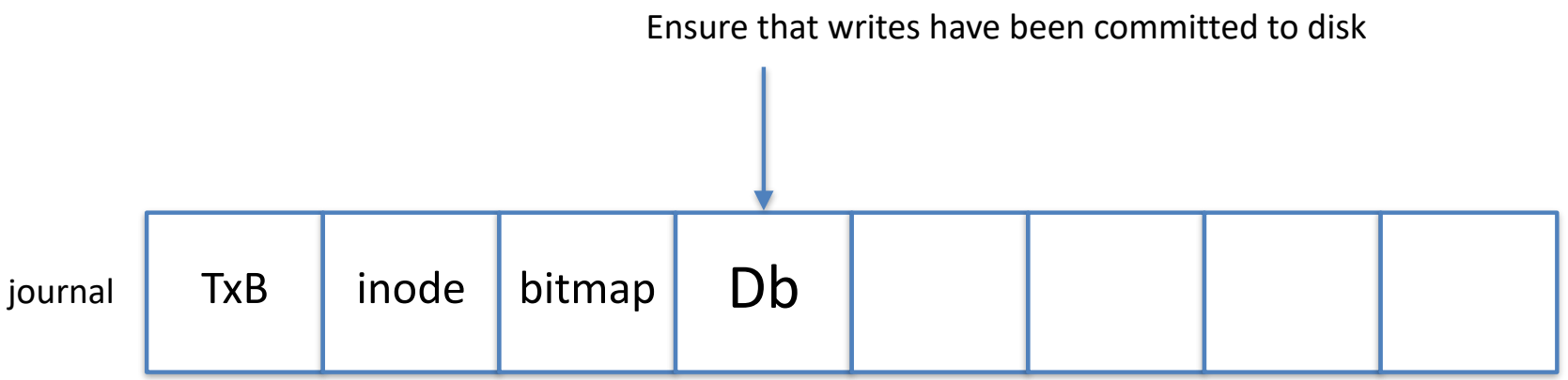
Note: writes of 512 byte sectors are atomic.

JOURNALING ALLOWS US TO RECOVER

JOURNALING ALLOWS US TO RECOVER

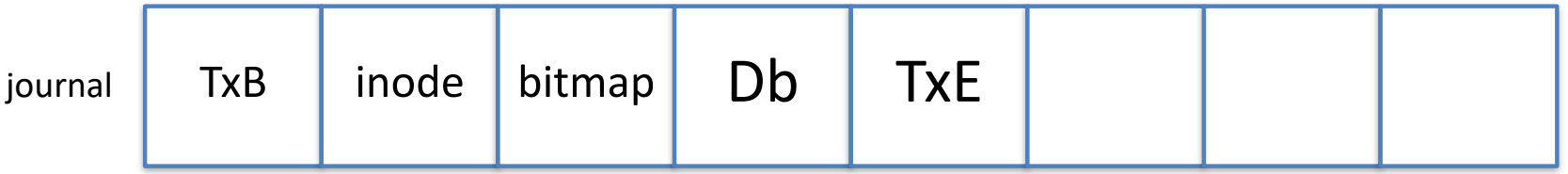
JOURNALING ALLOWS US TO RECOVER

JOURNALING ALLOWS US TO RECOVER



POSSIBLE FAILURE SCENARIOS: JOURNALING

1. Power fails during journaling before TxE commits: transaction is lost, but fs stays consistent.
2. Power fails after TxE commits: recover the transaction from journal.
3. Power fails after journal commits while updating on-disk structs: recover transaction from journal.



FAT FS

ADMINISTRIVIA

- Homework 8 (FAT FS) Due Next Wednesday 10/30
- Canonical Kernel Dev Screening Questions on website.
- Project Proposals Due 11/4
- Ignite Lab Networking Wednesday 10/23, Cuneo 111

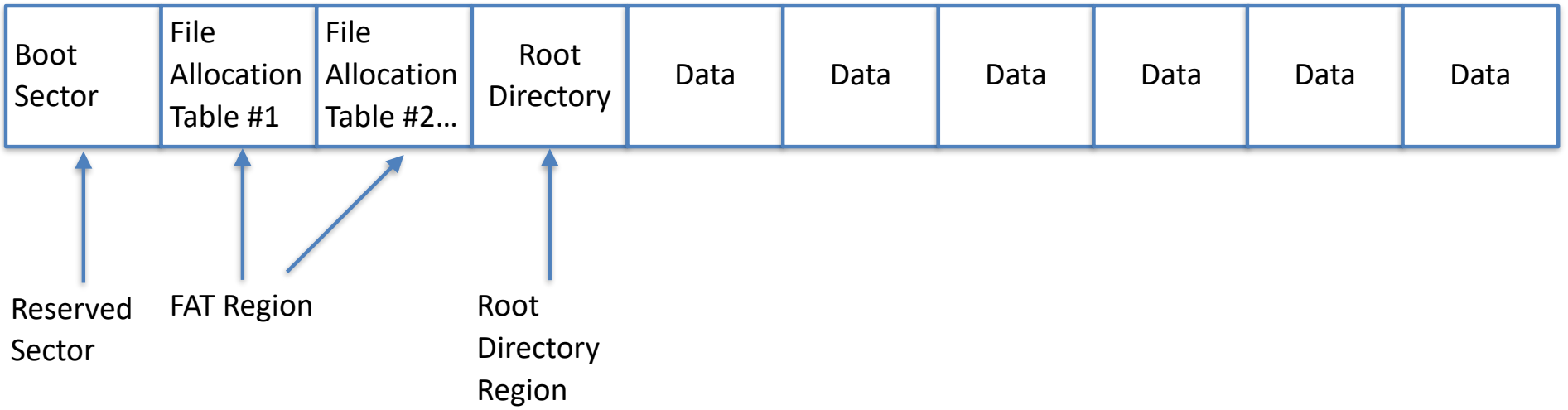
FINAL PROJECT IDEAS: FILESYSTEMS

- Filesystems tend to scatter data all over the place (inode table, inode bitmap, directory entries all in different locations).
  - Does this slow data accesses down? Benchmark to find out.
- Filesystems are needlessly complex for many applications.
  - Can we get away with no directories in some cases?
  - Is there a better way to organize data on disk (other than n-ary tree, which is inefficient)?

WHAT TO TAKE NEXT

- COMP 410 Advanced OS
  - Not just cross-listed combined section with grad students and undergrads.
  - Not offered next semester.
- COMP 445 IoT Device Application Security
  - Offered next semester, tenatively T/TH around noon

FAT FILESYSTEM



BOOT SECTOR (SUPERBLOCK)

Offset	Size	Description
0x00	3	Jump Instruction. Unused by you.
0x03	8	OEM Name (name of formatting program)
0x0B	2	Bytes per sector
0x0D	1	Sectors per cluster (cluster = block)
0x0E	2	Number of reserved sectors
0x10	1	Number of FATs
0x11	2	Number of root directory entries
0x13	2	Total sectors
0x15	1	Media descriptor. Unused by you
0x16	2	Sectors per FAT
0x18	2	Sectors per track
0x1A	2	Number of heads
0x1C	4	Number of hidden sectors
0x20	4	Total sectors in the FS
0x24	1	Logical Drive Number
0x25	1	Reserved
0x26	1	Extended Signature
0x27	4	Serial number
0x2B	11	Volume label
0x36	8	FS type

ROOT DIRECTORY ENTRY

Offset	Size	Description
0x00	8	Short file name
0x08	3	File extension
0x0B	1	File attributes
0x0C	1	Attributes. Not needed by you
0x0D	1	First character of a deleted file
0x0E	2	Create time. Not needed by you.
0x10	2	Create date. Not needed by you.
0x12	2	Last access date
0x14	2	File access rights bitmap. Not needed by you.
0x16	2	Last modified time. Not needed by you.
0x18	2	Last modified date. Not needed by you.
0x1A	2	Start of file in clusters.
0x1C	4	File size in bytes.

MOUNTING YOUR FAT FS

1. Read the boot sector. Use the info in it to find the root directory entry on disk.
2. Read the root directory entry. Iterate thru each RDE searching for a match with the filename you're looking for. When you find the match, the RDE will tell you the data cluster where you can find the file's data.
3. Read the FAT. If your file takes up more than one data cluster, the FAT will contain linkages to the other ones.

EXAMPLE: BOOT SECTOR (SUPERBLOCK)

What sector do we read to get the RDE?

Offset	Size	Value	Description
0x00	3	0xEB 0x3C 0x90	Jump Instruction. Unused by you.
0x03	8	“mkfs.fat”	OEM Name (name of formatting program)
0x0B	2	0x0200	Bytes per sector
0x0D	1	0x04	Sectors per cluster (cluster = block)
0x0E	2	0x0004	Number of reserved sectors
0x10	1	0x02	Number of FATs
0x11	2	0x0002	Number of root directory entries
0x13	2	0x0080	Total sectors
0x15	1	0xF8	Media descriptor. Unused by you
0x16	2	0x0020	Sectors per FAT
0x18	2	0x0020	Sectors per track
0x1A	2	0x0002	Number of heads
0x1C	4	0x00000000	Number of hidden sectors
0x20	4	0x00000000	Total sectors in the FS
0x24	1	0x80	Logical Drive Number
0x25	1	0x00	Reserved
0x26	1	0x29	Extended Signature
0x27	4	0xD52A5875	Serial number
0x2B	11	“NO NAME”	Volume label
0x36	8	“FAT16”	FS type

RDE Location = # FAT Tables * #Sectors/FAT + # Hidden Sectors + # Reserved Sectors

RDE Location = 2 * 32 + 0 + 4 = 68

Offset	Size	Value	Description
0x00	3	0xEB 0x3C 0x90	Jump Instruction. Unused by you.
0x03	8	“mkfs.fat”	OEM Name (name of formatting program)
0x0B	2	0x0200	Bytes per sector
0x0D	1	0x04	Sectors per cluster (cluster = block)
0x0E	2	0x0004	Number of reserved sectors
0x10	1	0x02	Number of FATs
0x11	2	0x0002	Number of root directory entries
0x13	2	0x0080	Total sectors
0x15	1	0xF8	Media descriptor. Unused by you
0x16	2	0x0020	Sectors per FAT
0x18	2	0x0020	Sectors per track
0x1A	2	0x0002	Number of heads
0x1C	4	0x00000000	Number of hidden sectors
0x20	4	0x00000000	Total sectors in the FS
0x24	1	0x80	Logical Drive Number
0x25	1	0x00	Reserved
0x26	1	0x29	Extended Signature
0x27	4	0xD52A5875	Serial number
0x2B	11	“NO NAME”	Volume label
0x36	8	“FAT16”	FS type

ROOT DIRECTORY ENTRY #1

Offset	Size	Value	Description
0x00	8	41 6a 00 75 00 6e 00 6b (“Aj.u.n.k”)	Short file name
0x08	3	00 2e 00	File extension
0x0B	1	0x0F	File attributes
0x0C	1	0x00	More Attributes. Not needed by you
0x0D	1	0x3C (’t’)	First character of a deleted file
0x0E	2	0x0074	Create time. Not needed by you.
0x10	2	0x0078	Create date. Not needed by you.
0x12	2	0x0074	Last access date
0x14	2	0x0000	File access rights bitmap. Not needed by you.
0x16	2	0xFFFF	Last modified time. Not needed by you.
0x18	2	0xFFFF	Last modified date. Not needed by you.
0x1A	2	0x0000	Start of file in clusters.
0x1C	4	0xFFFFFFFF	File size in bytes.

ROOT DIRECTORY ENTRY #2

Offset	Size	Value	Description
0x00	8	4a 55 4e 4b 20 20 20 20 (“JUNK “)	Short file name
0x08	3	54 58 54 (“TXT”)	File extension
0x0B	1	0x20	File attributes
0x0C	1	0x00	More Attributes. Not needed by you
0x0D	1	0x19	First character of a deleted file
0x0E	2	0x7CA9	Create time. Not needed by you.
0x10	2	0x5270	Create date. Not needed by you.
0x12	2	0x5270	Last access date
0x14	2	0x0000	File access rights bitmap. Not needed by you.
0x16	2	0x7CA9	Last modified time. Not needed by you.
0x18	2	0x5270	Last modified date. Not needed by you.
0x1A	2	0x0000	Start of file in clusters.
0x1C	4	0x00000000	File size in bytes.