信息科学与技术学院
School of Information Science and Technology

CS 110
Computer Architecture
Fault-tolerance

Instructors: Siting Liu & Yuan Xiao
Course website: https://faculty.sist.shanghaitech.edu.cn/liust/courses/CS110.html
School of Information Science and Technology (SIST)
ShanghaiTech University

2026/6/16

Administratives

- Final exam, June 25th 8am-10am; you can bring **3**-page A4-sized double-sided cheat sheet, **handwritten** only! (Teaching center 102/202); the whole course will be covered. No electronic devices (no smart watches, no calculators, no phones, etc.)
- All the labs have been released! ✌️
- Project 4 released, ddl June 18th. **Will be checked during lab sessions June 18th/22nd/23rd.** ✌️
- HW 8 will be released today, ddl June 21st. ✌️
- Discussion June 12th on *Final Review*.

Recall: Great Idea: Dependability via Redundancy

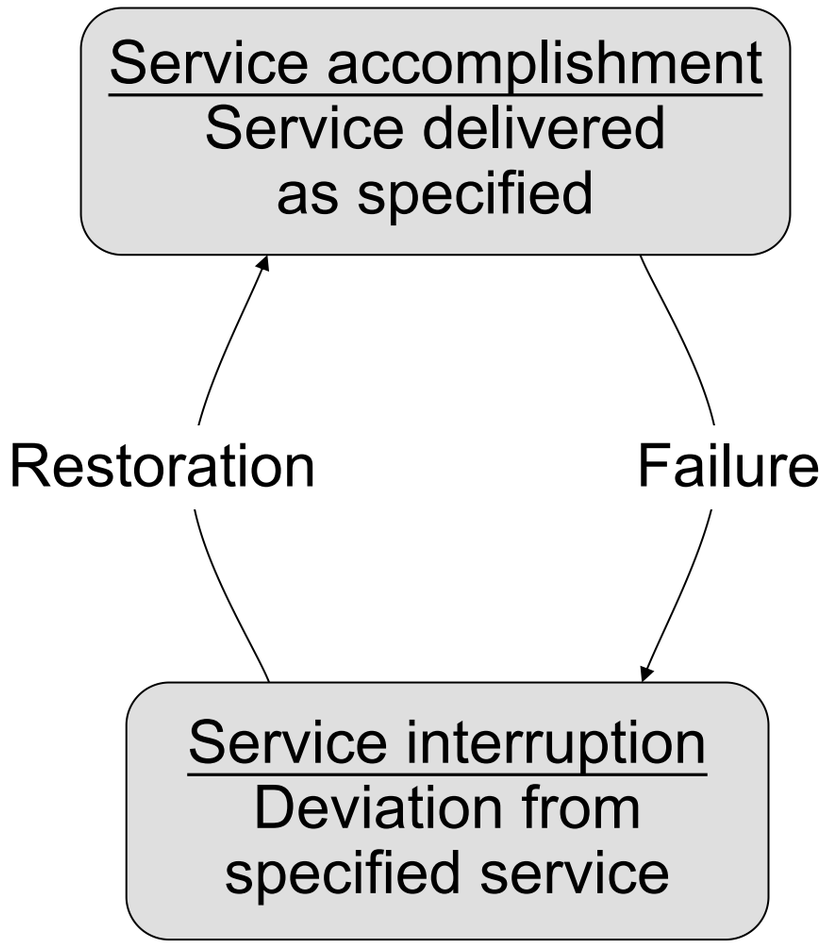
- Applies to everything from datacenters to memory
  - **Redundant datacenters** so that can lose 1 datacenter but Internet service stays online
  - **Redundant routes** so can lose nodes but Internet doesn't fail
  - **Redundant memory bits** of so that can lose 1 bit but no data (Error Correcting Code/ECC Memory)
  - **Redundant disks** so that can lose 1 disk but not lose data (Redundant Arrays of Independent (Inexpensive) Disks/RAID)

Dependability via Redundancy

- Spatial Redundancy – replicated data or check information or hardware to handle hard and soft (transient) failures;
  - information redundancy: error correction code (ECC and RAID, later this lecture);
  - hardware redundancy: triple modular redundancy (TMR); RAID;
- Temporal Redundancy – redundancy in time (retry) to handle soft (transient) failures

Dependability

- Fault: failure of a component
  - May or may not lead to system failure



Dependability Measures

- Reliability: Mean Time To Failure (MTTF)
- Service interruption: Mean Time To Repair (MTTR)
- Mean time between failures (MTBF)
  - MTBF = MTTF + MTTR
  - Availability = MTTF/(MTTF+MTTR)
- Improving Availability
  - Increase MTTF: More reliable hardware/software + Fault tolerance
  - Reduce MTTR: Improved tools and processes for diagnosis and repair

Availability Measures

- Availability = MTTF/(MTTF+MTTR) as %
- MTTF, MTBF usually measured in hours
- Since hope rarely down, shorthand is “number of 9‘s of availability per year”
  - 1 nine: 90% => 36 days of repair/year
  - 2 nines: 99% => 3.6 days of repair/year
  - 3 nines: 99.9% => 526 minutes of repair/year
  - 4 nines: 99.99% => 53 minutes of repair/year
  - 5 nines: 99.999% => 5 minutes of repair/year

Reliability Measures

- Another is average number of failures per year: Annualized Failure Rate (AFR)
- E.g., 1000 disks with 100,000 hours MTTF
- 365 days * 24 hours = 8760 hours
- (1000 disks * 8760 hrs/year) / 100,000 = 87.6 failed disks per year on average
- 87.6/1000 = 8.76% annual failure rate
- Google's 2007 study* found that actual AFRs for individual drives ranged from 1.7% for first year drives to over 8.6% for three-year old drives

*research.**google**.com/archive/disk_failures.pdf

Dependable Design Principle

- Design Principle: No single points of failure
  - "Chain is only as strong as its weakest link"
  - Achilles' Heel
- Dependability Corollary of Amdahl's Law
  - Doesn't matter how dependable you make one portion of system
  - Dependability limited by part you do not improve

Error Detection/Correction Codes

- Memory systems generate errors (accidentally flipped-bits)
  - DRAMs store very little charge per bit
  - "Soft" errors occur occasionally when cells are struck by alpha particles or other environmental upsets
  - "Hard" errors occur when chips permanently fail
    - Problem gets worse as memories get denser and larger
- Memories protected against failures with EDC/**ECC**
- Extra bits are added to each data-word
  - Used to detect and/or correct faults in the memory system
  - Each dataword value mapped to unique codeword
  - A fault changes valid codeword to invalid one, which can be detected

Block Code Principles

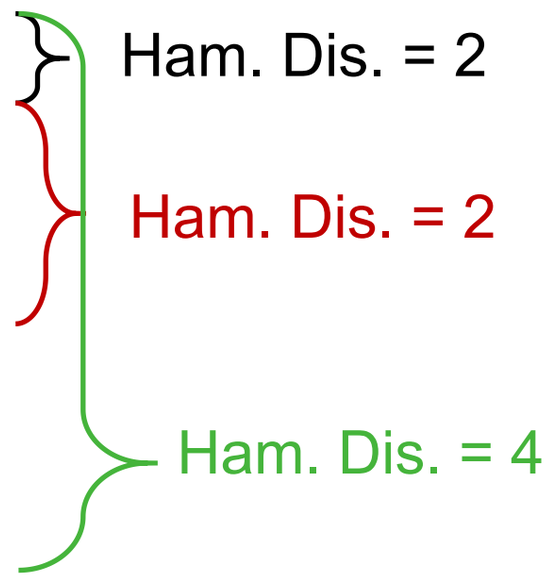
- Hamming distance = difference in # of bits
- E.g., p = 011011, q = 001111,
  - Ham. distance (p,q) = 2
- p = 011011, q = 110001, distance (p,q) =
- Can think of extra bits as creating a code with the data (a.k.a. information redundancy)
- There would be Ham. distance>1 between different valid codewords

Richard Hamming, 1915-98
Turing Award Winner

A Simple Implementation: Parity Bit

- Parity bits are added to a word to make it
  - either odd: odd number of ‘1’s
  - or even: even number of ‘1’s
- Let us add one parity bit to three-bit word
  - Odd parity: append a ‘1’ if having even number of ‘1’s in the original dataword, so that the whole codeword contains odd number of ‘1’s

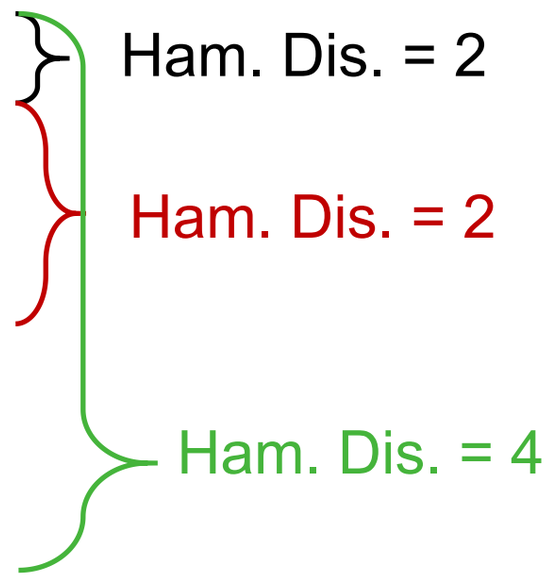
Dataword	Even number of ‘1’s?	Codeword
000	True	000 1
001	False	001 0
010	False	010 0
011	True	011 1
100	False	100 0
101	True	101 1
110	True	110 1
111	True	111 0



A Simple Implementation: Parity Bit

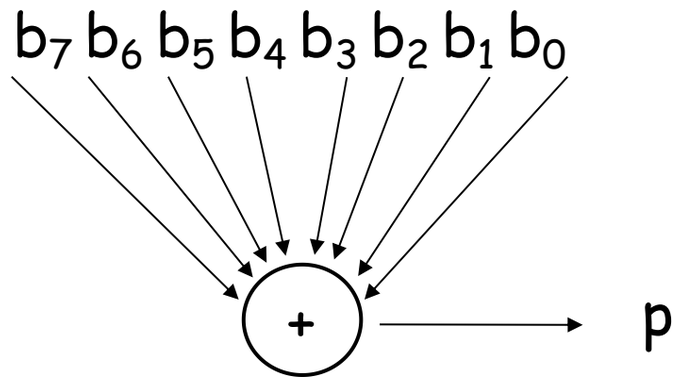
- Parity bits are added to a word to make it
  - either odd: odd number of ‘1’s
  - or even: even number of ‘1’s
- Let us add one parity bit to three-bit word
  - Even parity: append a ‘1’ if having odd number of ‘1’s in the original dataword, so that the whole codeword contains even number of ‘1’s

Dataword	Even number of ‘1’s?	Codeword
000	True	000 0
001	False	001 1
010	False	010 1
011	True	011 0
100	False	100 1
101	True	101 0
110	True	110 0
111	True	111 1



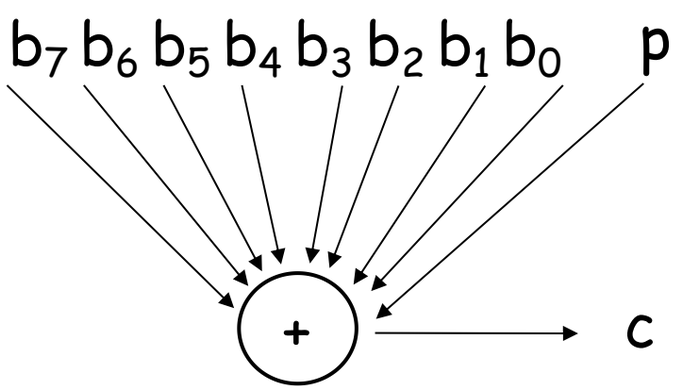
Parity Bit

- Each data value, before it is written to memory is "tagged" with an extra bit to force the stored word to have even parity.



- Requires extra one bit storage

- Each word, as it is read from memory is "checked" by finding its parity (including the parity bit).



- *c=1* indicates single error may occur (single error detection);
- Double (even)-bit error cannot be detected

"+" can be implemented by XOR

Parity Example

- Data 0101 0101
- 4 ones, even parity now
- Write to memory:0101 0101 0 to keep parity even

➡

- Read from memory 0101 0101 0
- 4 ones => even parity, detect no error

- Data 0101 0111
- 5 ones, odd parity now
- Write to memory: 0101 0111 1 to make parity even

➡

- Read from memory 0101 0101 1
- 5 ones => odd parity, detect error
- What if error in parity bit?

What about Error Correction?

- Richard Hamming came up with simple to understand mapping to allow single error correction (SEC) at minimum Ham. distance of 3
- Single error correction, double error detection (SEC/DED)
- Called “Hamming ECC”
- Worked weekends on relay computer with unreliable card reader, frustrated with manual restarting
- Got interested in error correction; published 1950
- R. W. Hamming, “Error Detecting and Correcting Codes,” The Bell System Technical Journal, Vol. XXVI, No 2 (April 1950) pp 147-160.

Detecting/Correcting Code Concept

- Error detection: bit pattern fails codeword check
- Error correction: map to the nearest valid codeword

Space of possible bit patterns (2^N)

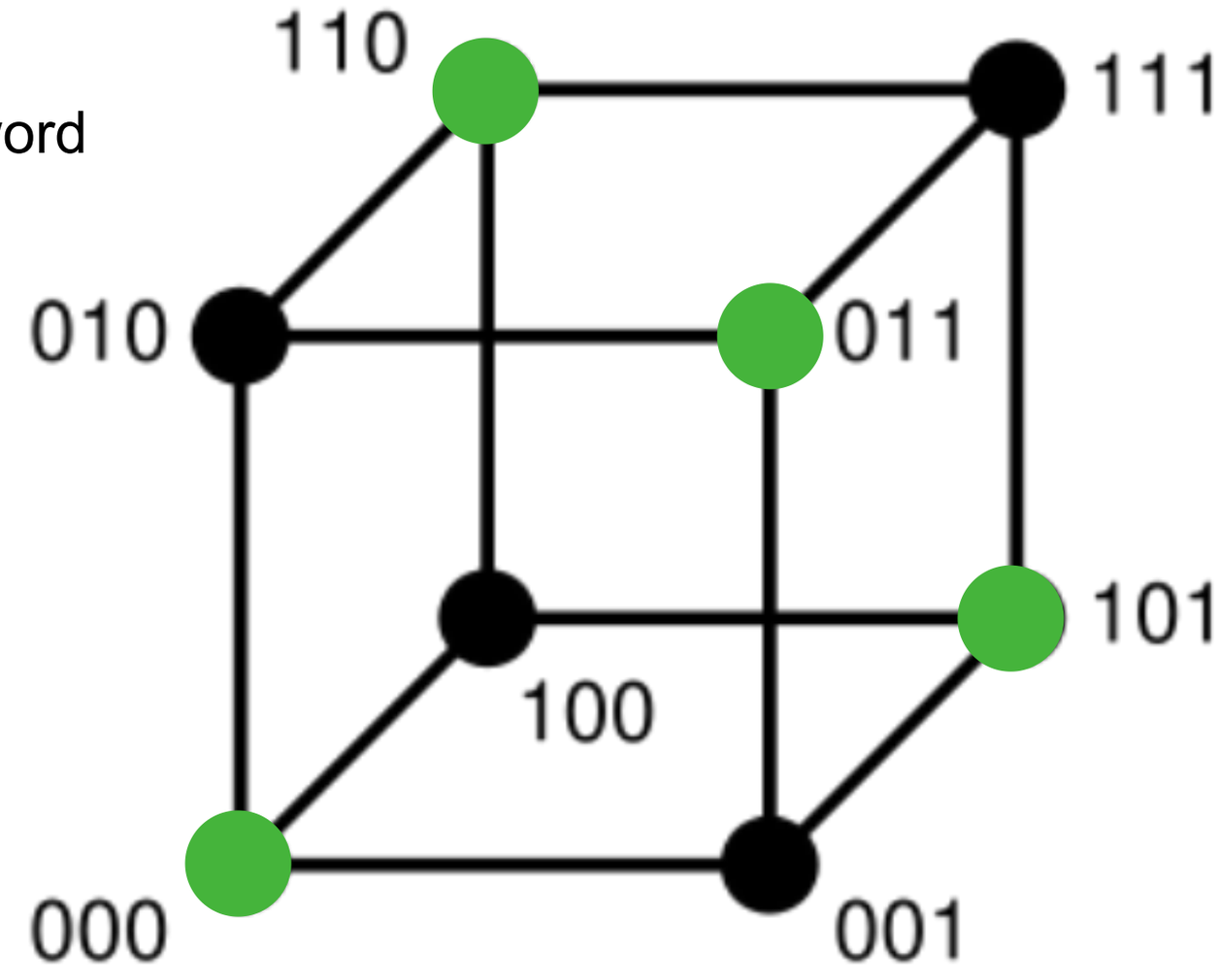
Error changes bit pattern to non-code (invalid codeword)

Sparse population of valid codewords ($2^M << 2^N$)
with identifiable signature

Hamming Distance: 8 Codeword

- 1 Parity bit: single error detection

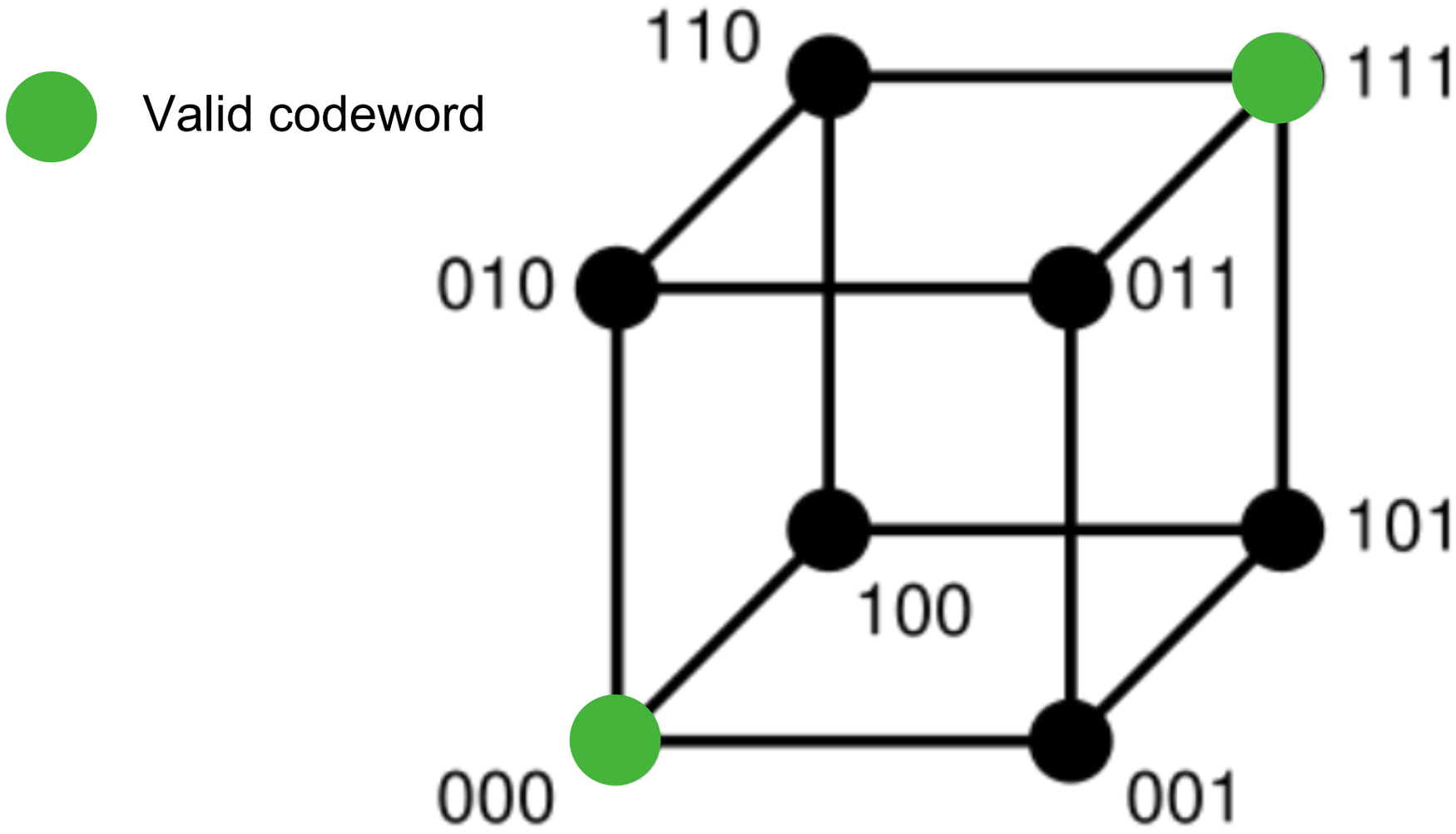
Valid codeword



No single-error codeword goes to another valid codeword;
1/2 codewords are valid

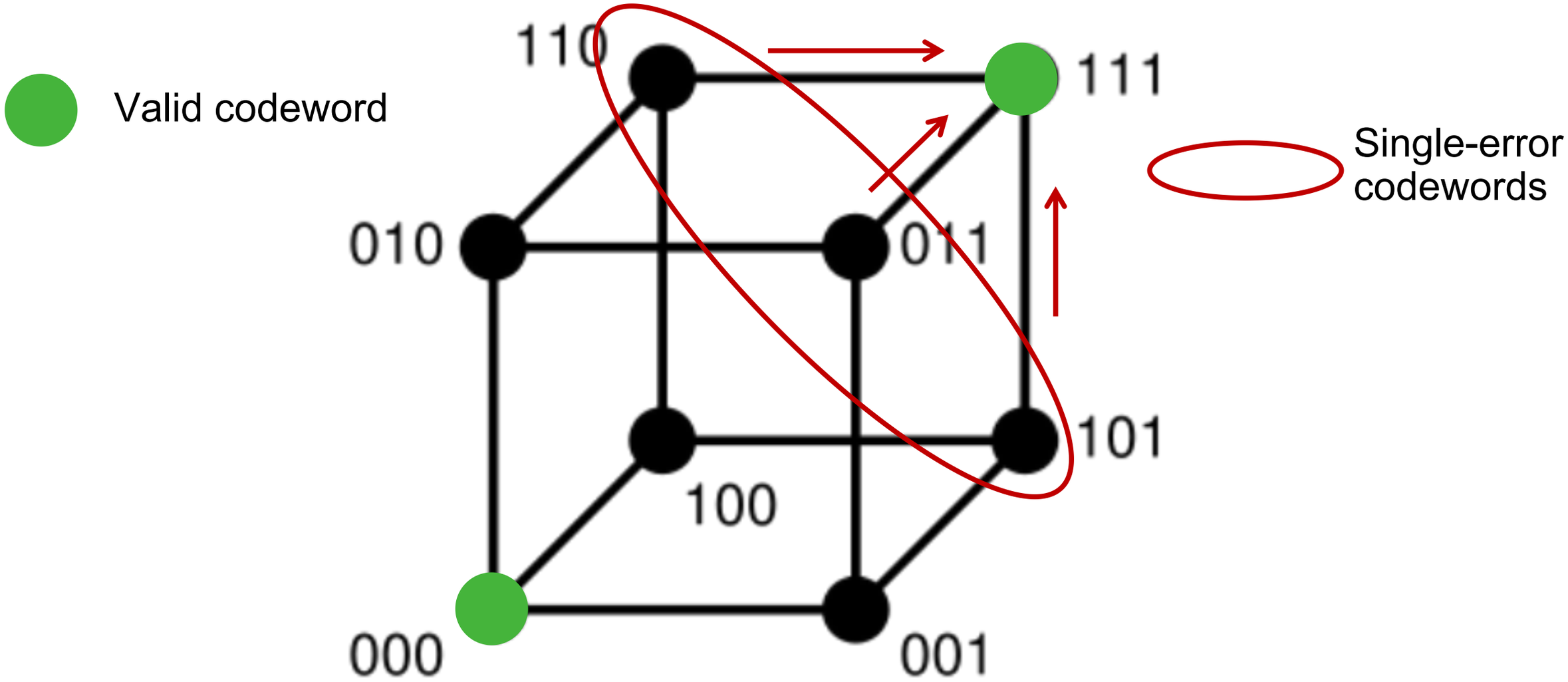
Hamming Distance of 3 Leads to SEC

- 2 Parity bits: single error correction (SEC); double error detection (DED)



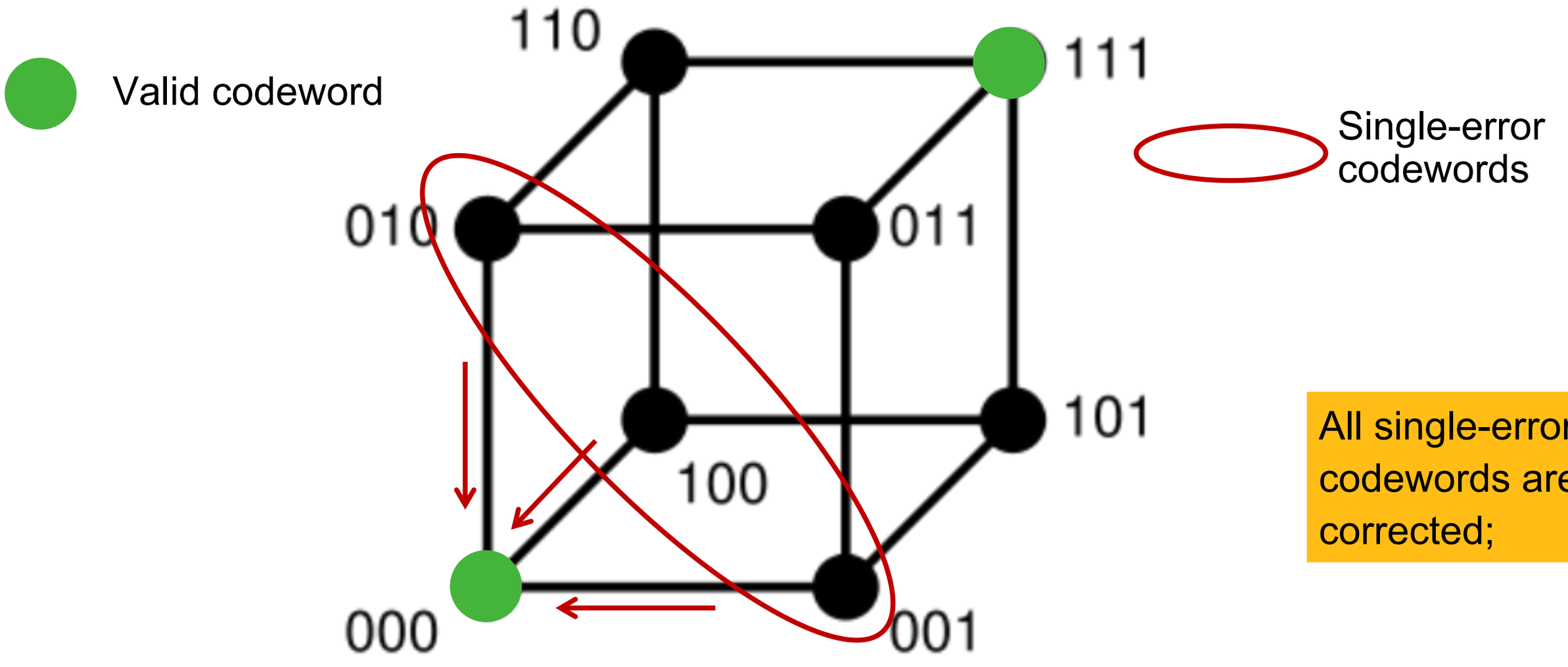
Hamming Distance of 3 Leads to SEC

- 2 Parity bits: single error correction (SEC); double error detection (DED)



Hamming Distance of 3 Leads to SEC

- 2 Parity bits: single error correction (SEC); double error detection (DED)

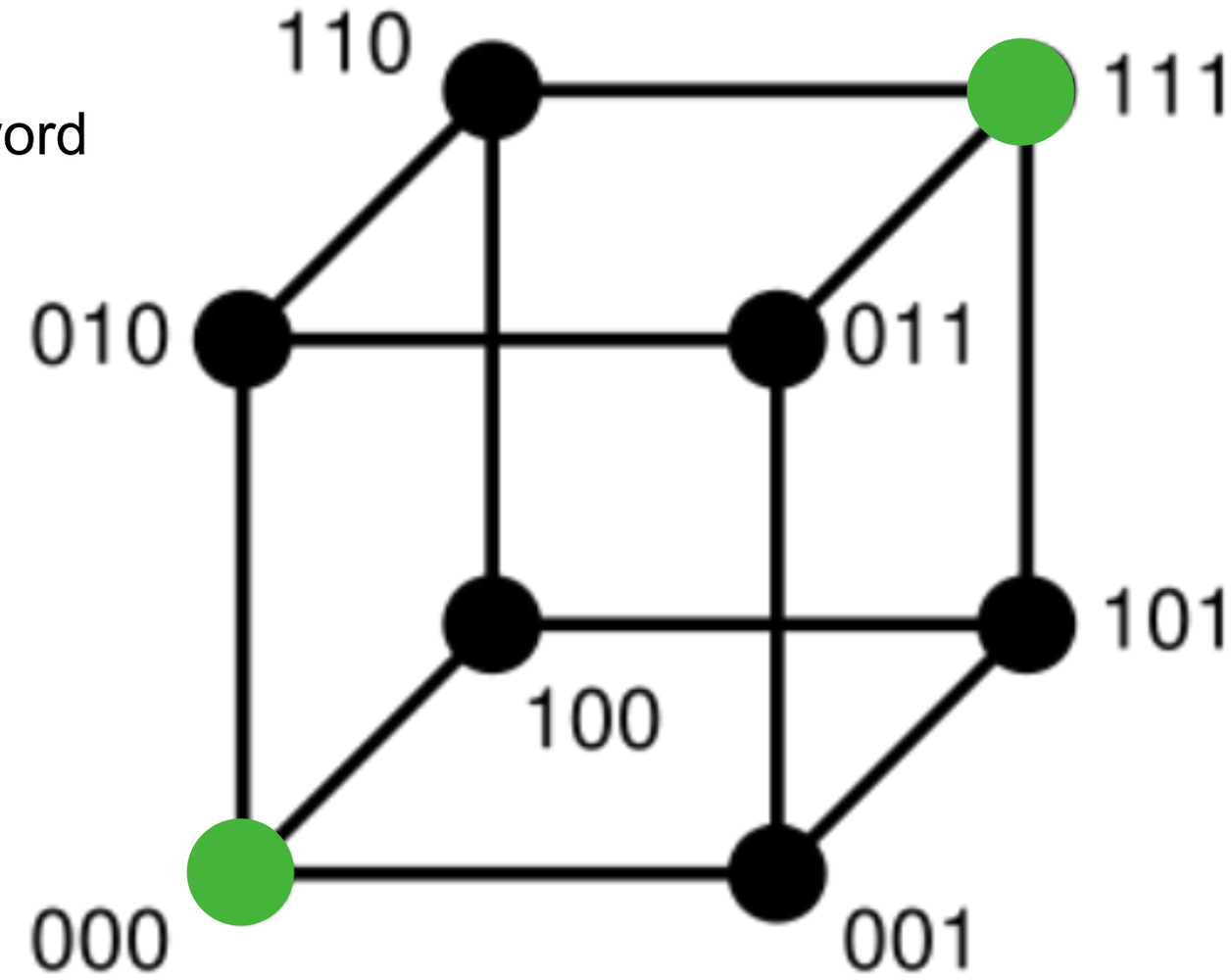


All single-error codewords are corrected;

Hamming Distance of 3 Leads to SEC

- 2 Parity bits: single error correction (SEC); double error detection (DED)

Valid codeword



No 2 bit error goes to another valid codeword; 1/4 codewords are valid

General Method for Hamming ECC

- Extra parity bits allow the position identification of a single error
- 1. Insert all bit positions that are powers of 2 as parity bits (positions 1, 2, 4, 8, 16, …)
  - Start numbering bits at 1 at left (not at 0 on right)
  - All other bit positions are data bits (positions 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, …)
- 2. The position of parity bit determines sequence of data bits that it checks
  - Bit 1: checks bits (1,3,5,7,9,11,...)
    - Bits with least significant bit of address = 1
  - Bit 2: checks bits (2,3,6,7,10,11,14,15,…)
    - Bits with 2nd least significant bit of address = 1
  - Bit 4: checks bits (4-7, 12-15, 20-23, ...)
    - Bits with 3rd least significant bit of address = 1
  - Bit 8: checks bits (8-15, 24-31, 40-47 ,...)
    - Bits with 4th least significant bit of address = 1
  - ... ...

Dataword

0 1 1 0 0 1 0 1

P P 0 P 1 1 0 P 0 1 0 1

Bit 1

P P ***0*** P ***1*** 1 ***0*** P ***0*** 1 ***0*** 1

1 P ***0*** P ***1*** 1 ***0*** P ***0*** 1 ***0*** 1

General Method for Hamming ECC (Cont'd)

Dataword

0 1 1 0 0 1 0 1

P P 0 P 1 1 0 P 0 1 0 1

Bit 1

P P ***0*** P ***1*** 1 ***0*** P ***0*** 1 ***0*** 1

1 P ***0*** P ***1*** 1 ***0*** P ***0*** 1 ***0*** 1

Bit 2

1 ***P*** 0 P 1 ***1 0*** P 0 ***1 0*** 1

1 ***0*** 0 P 1 ***1 0*** P 0 ***1 0*** 1

Bit 4

1 0 0 ***P 1 1 0*** P 0 1 0 ***1***

Bit 8

1 0 0 ***1 1 1 0*** P 0 1 0 ***1***

1 0 0 1 1 1 0 0 0 1 0 1

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11
Parity bit coverage	p1	X		X		X		X		X		X		X		X
	p2		X	X			X	X			X	X			X	X
	p4				X	X	X	X					X	X	X	X
	p8								X	X	X	X	X	X	X	X

Hamming ECC Error Check & Correction

- Suppose receive 1 0 0 1 1 1 0 0 0 1 **1** 1

	1	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**Error**
Parities	1	0		1				0					
Bit 1 check	1		0		1		0		0		1		Error
Bit 2 check		0	0				1	0			1	1	Error
Bit 4 check				1	1	1	0					1	√
Bit 8 check								0	0	1	1	1	Error

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11
Parity bit coverage	p1	X		X		X		X		X		X		X		X
	p2		X	X			X	X			X	X			X	X
	p4				X	X	X	X					X	X	X	X
	p8								X	X	X	X	X	X	X	X

Bit 1 + Bit 2 + Bit 8 = Bit 11

1 0 0 1 1 1 0 0 0 1 **1** 1

Error correction

1 0 0 1 1 1 0 0 0 1 **0** 1

Hamming ECC Error Detection

- Suppose double error 1 0 **1** 1 1 1 0 0 0 1 **1** 1

	1	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**Error**
Parities	1	0		1				0					
Bit 1 check	1		1		1		0		0		1		√
Bit 2 check		0	1			1	0			1	1		√
Bit 4 check				1	1	1	0					1	√
Bit 8 check								0	0	1	1	1	Error

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11
Parity bit coverage	p1	X		X		X		X		X		X		X		X
	p2		X	X			X	X			X	X			X	X
	p4				X	X	X	X					X	X	X	X
	p8								X	X	X	X	X	X	X	X

Double error detected, but cannot decide which bit is wrong

How about ≧ 3 bit errors?

Hamming ECC Error Detection

- An example: valid codeword 0 1 1 1 0 0 1 0 1 0 1 0

- Double error: 0 1 1 1 1 1 1 0 1 0 1 0
- Checking
  - Bit 1: 0 1 1 1 1 1 X
  - Bit 2: 1 1 1 1 0 1 X
  - Bit 4: 1 1 1 1 0 ✓
  - Bit 8: 0 1 0 1 0 ✓

- Single error: 0 1 0 1 0 0 1 0 1 0 1 0
- Checking
  - Bit 1: 0 0 0 1 1 1 X
  - Bit 2: 1 0 0 1 0 1 X
  - Bit 4: 1 0 0 1 0 ✓
  - Bit 8: 0 1 0 1 0 ✓

- Lead to the same syndrome: add an extra parity bit for all the bits at the MSB to distinguish single/double errors
  - i.e., **0** 0 1 1 1 0 0 1 0 1 0 1 0
- **Extended Hamming Code**

RAID: Redundancy for Disk

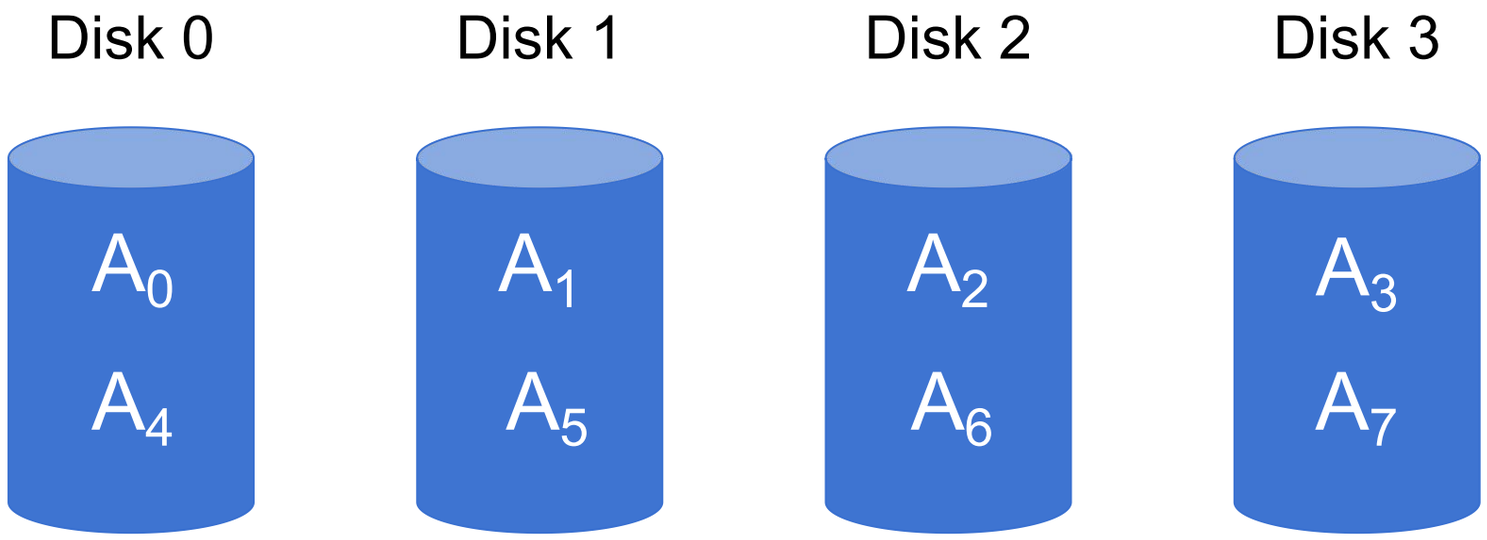
- Trade-off: price, capacity, density, etc.
- When you need storage space in petabytes (PB) or exabytes (EB)
  - 1 PB = 1024 TB
  - 1 EB = 1024 PB
- Do not forget that flash-based SSDs also fail
  - Limited program/erase cycles→wear leveling

RAID: Redundant Arrays of (Independent) Disks

- Files are “striped” across multiple disks
- Redundancy yields high data availability
  - Availability: service still provided to user, even if some components failed
- Disks will still fail
- Contents reconstructed from data redundantly stored in the array
  - Capacity penalty to store redundant info
  - Bandwidth penalty to update redundant info

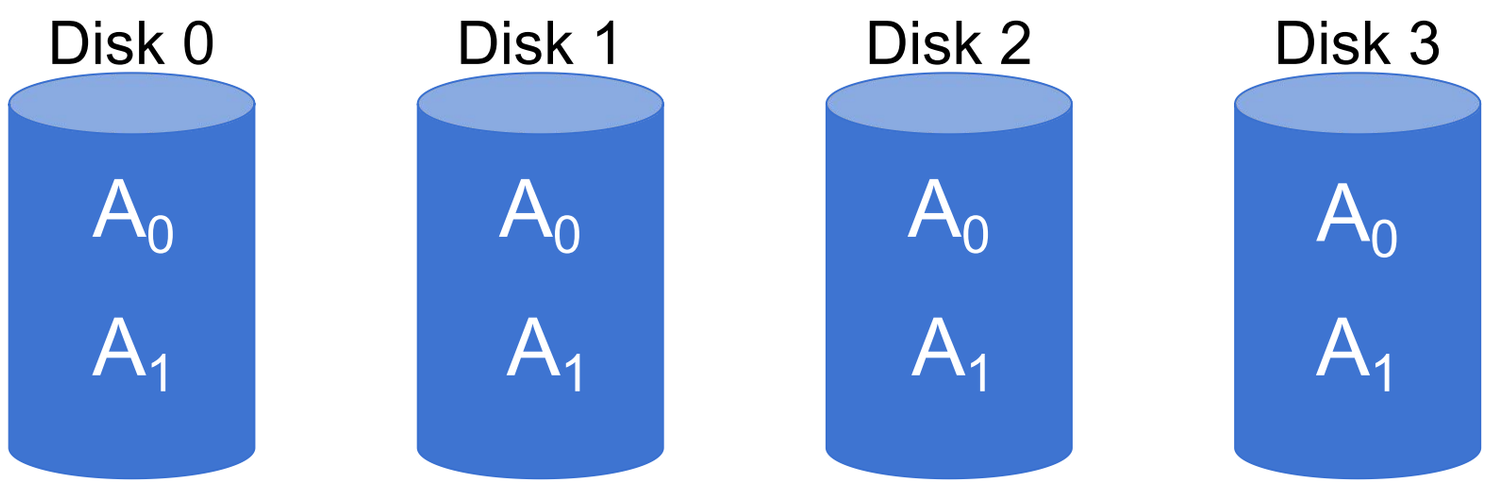
Various Configurations: RAID 0 (Striping)

- RAID 0 provides no fault tolerance or redundancy
  - Striping, or disk spanning
  - High performance



RAID 1 (Mirroring & Shadowing)

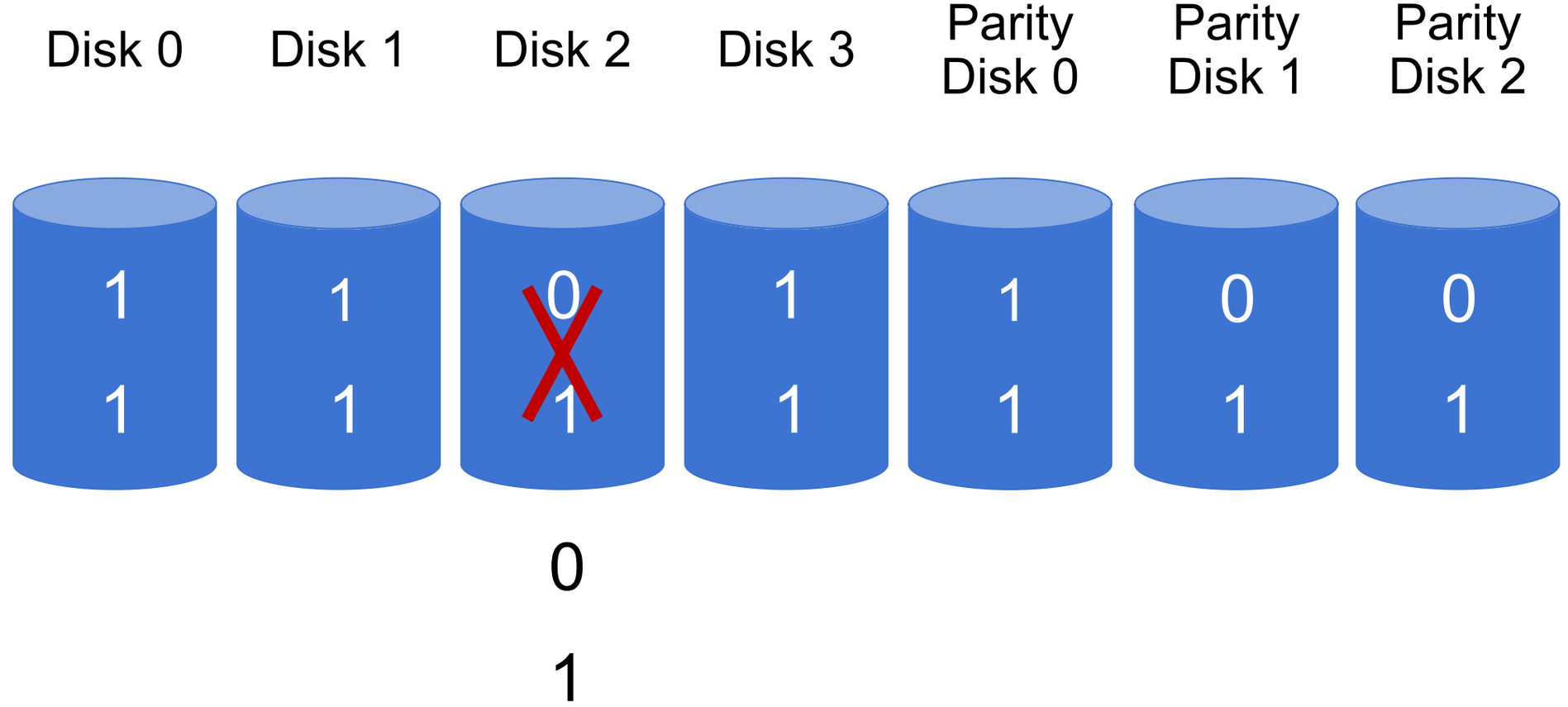
- Each disk is fully duplicated onto its "mirror(s)"
  - Hot spare, can support hot swap
  - Very high availability can be achieved
  - Bandwidth sacrifice on write: logical write = N physical writes
  - Reads may be optimized
- Most expensive solution: 100% capacity overhead
- RAID 10 (striped mirrors), RAID 01 (mirrored stripes):
  - Combinations of RAID 0 and 1.



RAID 2 (Parity Disk)

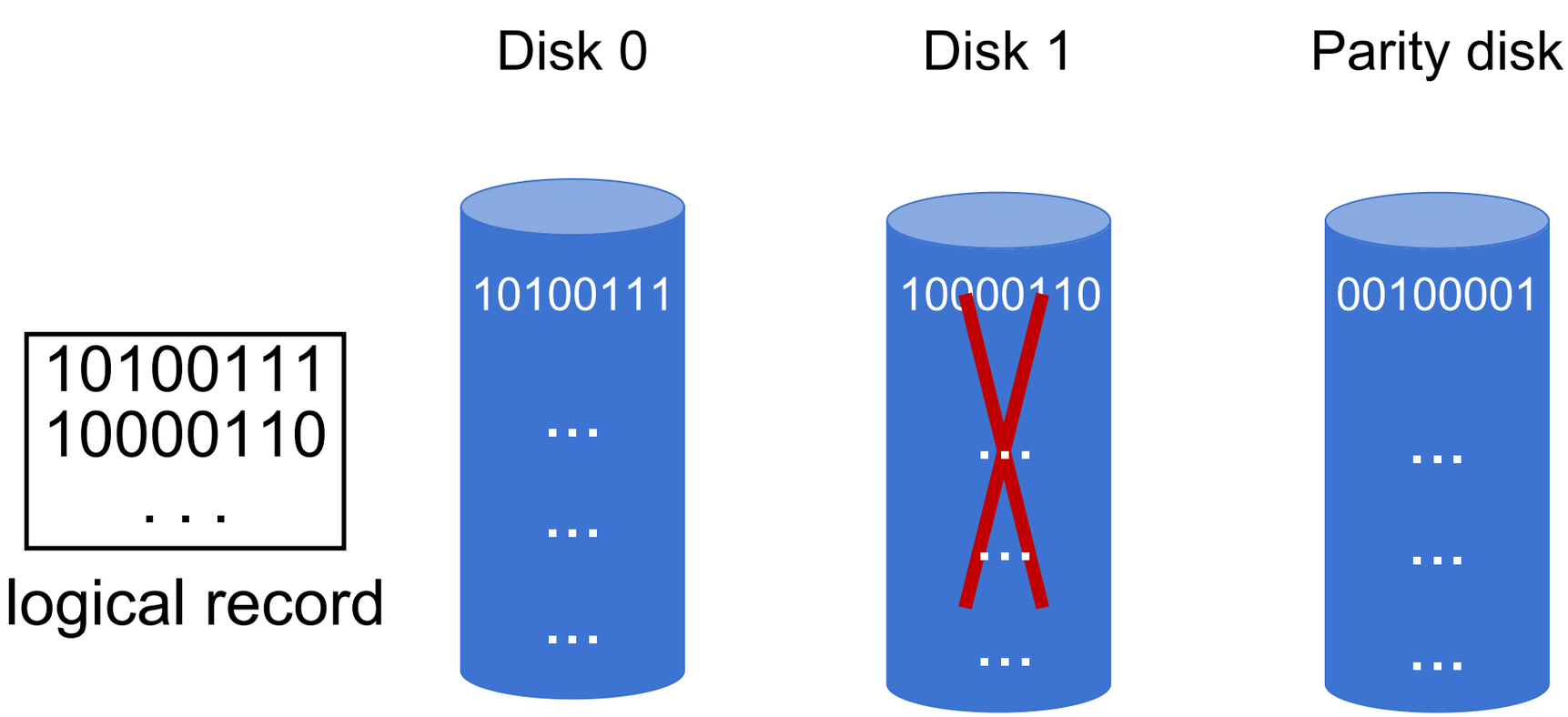
- P contains sum mod 2 (XOR) of other disks per stripe (“parity”)
- If disk fails, subtract P from sum of other disks to find missing information
- Bit-level + Hamming code

For example:



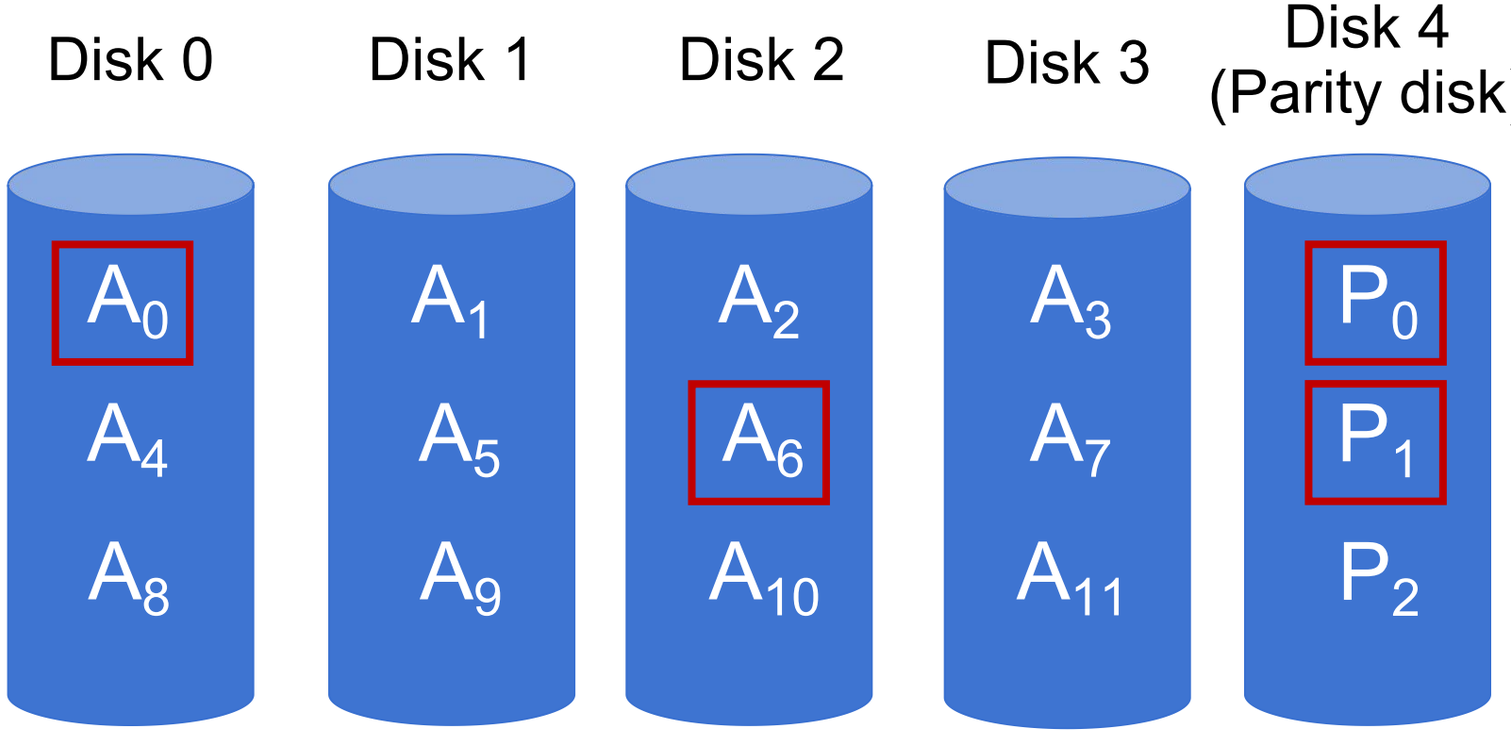
RAID 3 (Parity Disk)

- P contains sum mod 2 (XOR) of other disks per stripe (“parity”)
- If disk fails, subtract P from sum of other disks to find missing information
- Byte-level + parity disk



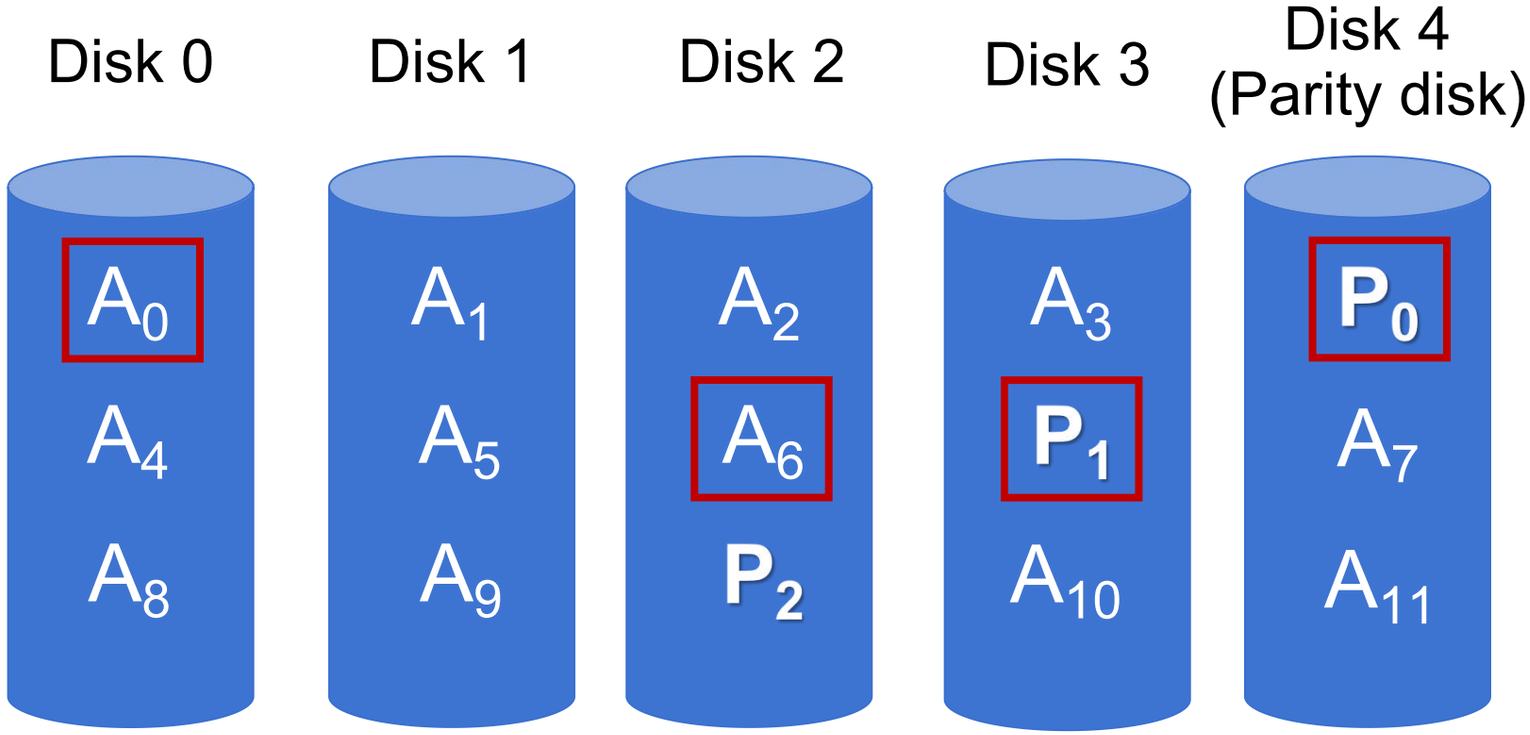
RAID 4 (Parity Disk, Block-level Striped)

- Good for sequential reads
- Bad for small writes
  - E.g. writes to A_0 & A_6 (disks 0 & 2)
  - Both write to disk 4 for changed parities, may lead to congestion



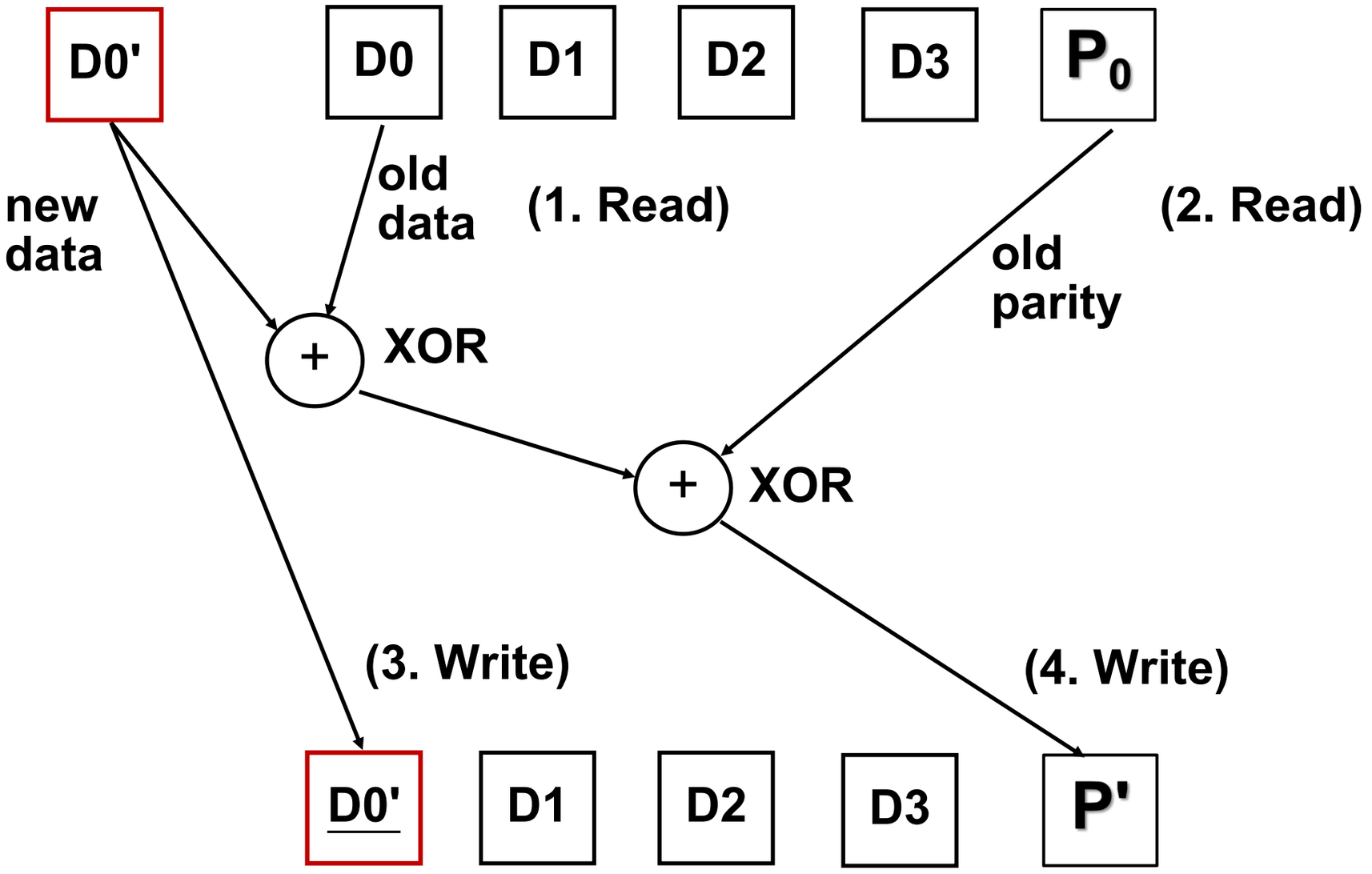
RAID 5 (Parity Disk, Striped & Interleaved)

- Good for small writes: high I/O rate interleaved parity, no specific parity disk
  - E.g. writes to A_0 & A_3 (disks 0 & 3)
  - Independent writes possible because of interleaved parity
    - Writes to disks 0, 2, 3 & 4



RAID 5 Small Write Algorithm

1 Logical Write = 2 Physical Reads + 2 Physical Writes



Conclusion

- Great Idea: Redundancy to Get Dependability
  - Spatial (extra hardware/information) and Temporal (retry if error)
- Reliability: MTTF & Annualized Failure Rate (AFR)
- Availability: % uptime
- Memory
  - Hamming ECC: correct single, detect double
- RAID
  - Interleaved data and parity

信息科学与技术学院
School of Information Science and Technology

CS 110
Computer Architecture
Security

Instructors: Siting Liu & Yuan Xiao
Course website: https://faculty.sist.shanghaitech.edu.cn/liust/courses/CS110.html
School of Information Science and Technology (SIST)
ShanghaiTech University

2026/6/16

Outline

- The Lives of Others
- Inception
  - Plant a value with a hammer
- Mission Impossible
  - When, or where
- The Water Horse
  - Flush the Loch Ness
- Meltdown and Spectre

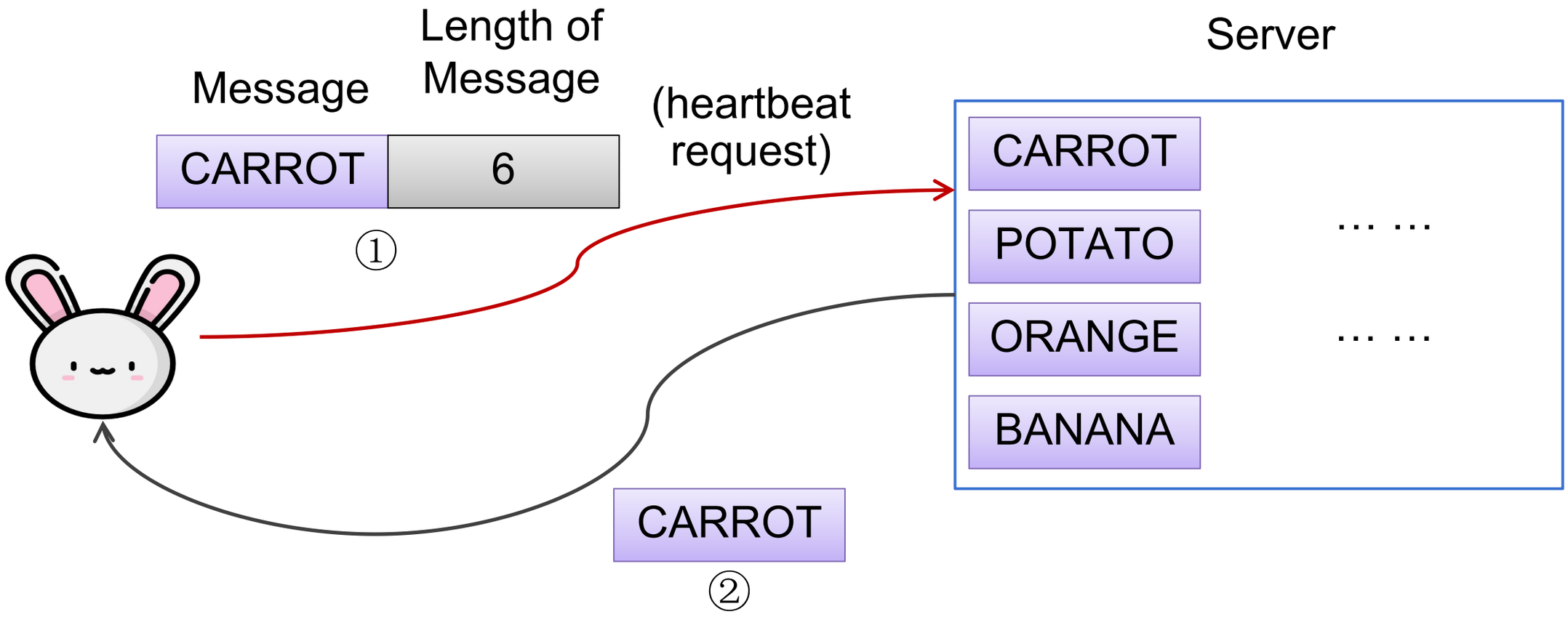
Over the Air: Heartbleed

- A network vulnerability uncovered in 2014 [1]
- Found in popular OpenSSL cryptographic software library
  - SSL/TLS: Secure Socket Layer (deprecated), and Transport Layer Security
  - Widely used in many applications: email, VPN, WeChat, Taobao, 12306, Momo, etc. [1, 2]

[1] https://heartbleed.com/
[2] https://daily.zhihu.com/story/3824566

How Heartbleed Works?

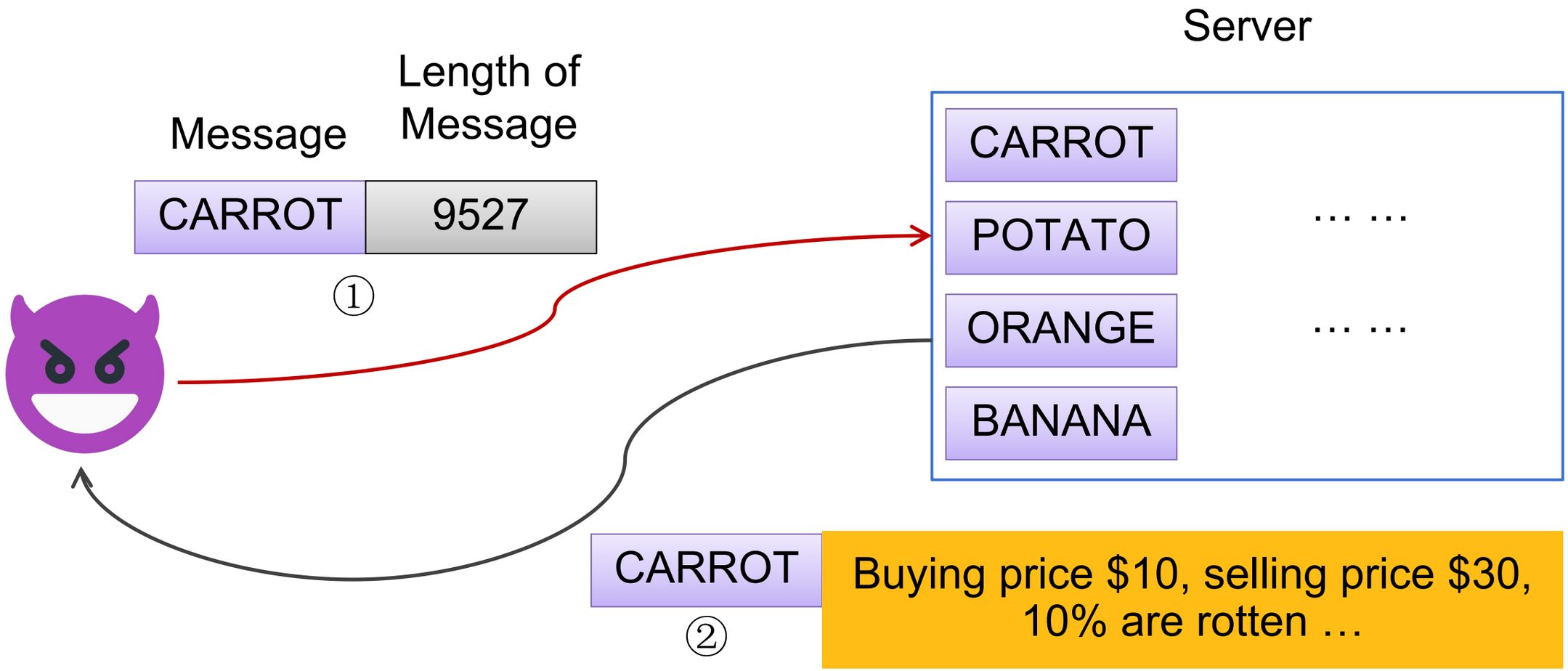
- The rule: one side sends a message with a length, and the other side replies with the same message according to the length.



[1] https://heartbleed.com/
[2] https://daily.zhihu.com/story/3824566

How Heartbleed Works?

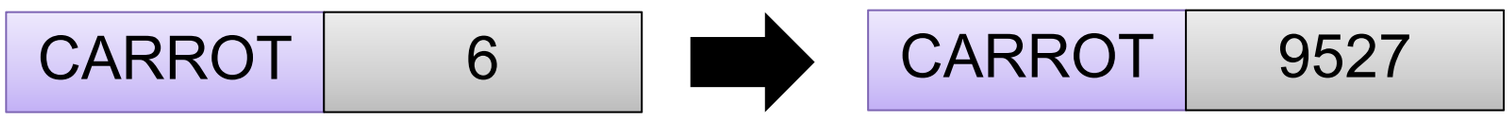
- The rule: one side sends a message with a length, and the other side replies with the same message according to the length.



[1] https://heartbleed.com/
[2] https://daily.zhihu.com/story/3824566

Why Heartbleed?

- Heartbleed happens if the packet's length is not validated.
- Found in TLS Heartbeat extension
  - To check whether a connection is still alive
  - One device confirms the other's continued presence by sending a specific payload in a packet that the other device sends back
    - Not only user to server, also possible server to user



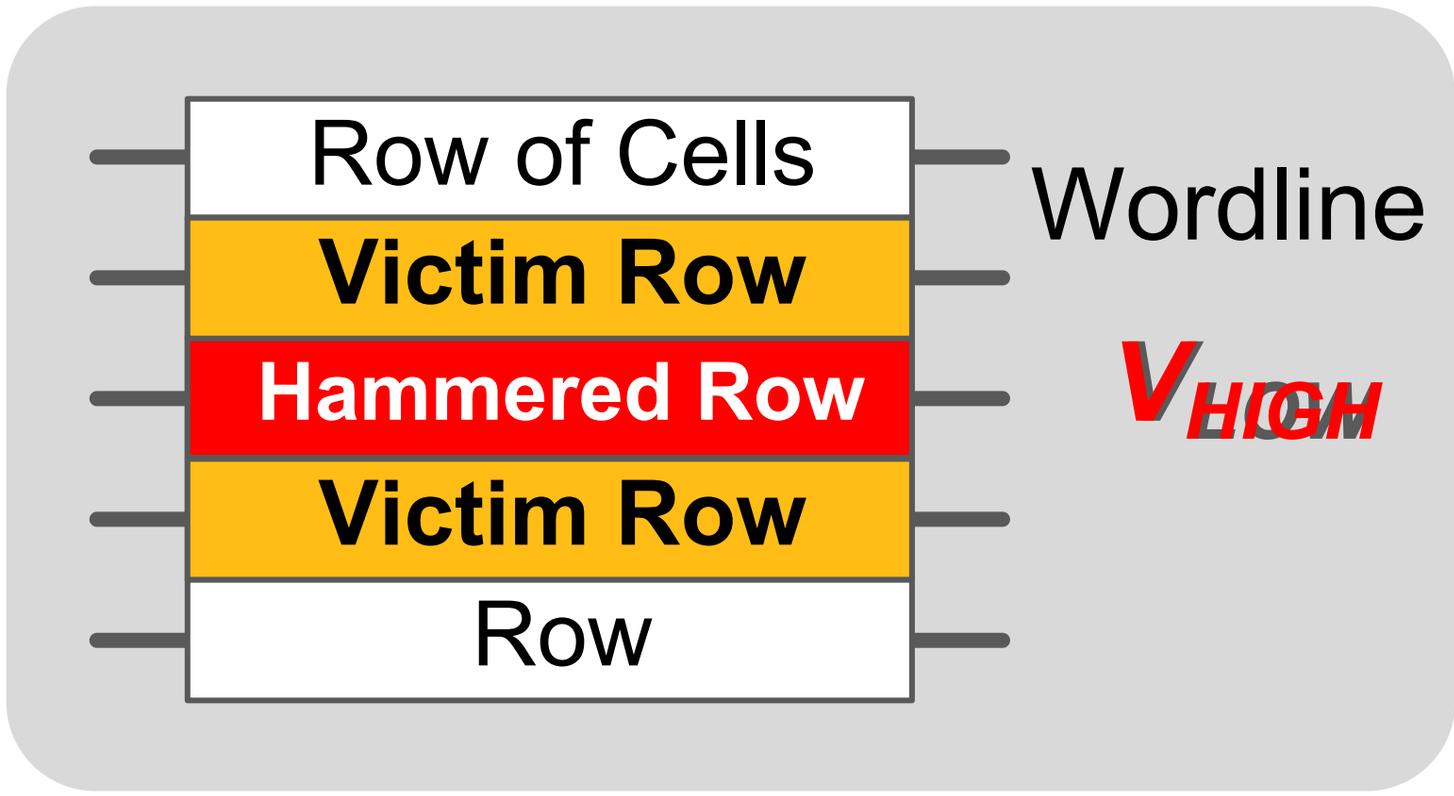
[1] https://heartbleed.com/
[2] https://daily.zhihu.com/story/3824566

Impact of Heartbleed

- Without using any privileged information or credentials, attackers can steal the victim's secrets, including user names and passwords, instant messages, emails and business critical documents and communication.
- Is it a design flaw of SSL/TLS?
  - No. It is an implementation flaw.
- Fix: length checking in the implementation
  - Developers/programmers implement them for networking
  - A problematic implementation causes problems
  - e.g., Wi-Fi and Bluetooth Low Energy
    - Wi-Fi: https://ieeexplore.ieee.org/document/9160891
    - BLE: https://dl.acm.org/doi/abs/10.5555/3489146.3489209

Inception: *Plant a Value with a Hammer*

- DRAM is prone to disturbance errors;
- Repeatedly reading a row enough times (before memory gets refreshed) induces disturbance errors in adjacent rows in most real DRAM chips you can buy today



Slides based on Onur Mutlu's slides https://people.inf.ethz.ch/omutlu/talks.htm
Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors, (Kim et al., ISCA 2014)

Why Rowhammer Happens?

- DRAM cells are too close to each other!
  - They are not electrically isolated from each other
- Access to one cell affects the value in nearby cells
  - Due to electrical interference between
    - the cells
    - wires used for accessing the cells
  - Also called cell-to-cell coupling/interference
- Example: When we activate (apply high voltage) to a row, an adjacent row gets slightly activated as well
  - Vulnerable cells in that slightly-activated row lose a little bit of charge
  - If row hammer happens enough times, charge in such cells gets drained, and bit may flip

How Rowhammer Becomes an Attack

- Picking a memory location
- Repeatedly accessing it without caching
- Adjacent rows in the same bank to be flipped
- An example to be exploited
  - Page table entry, containing a physical page no.
  - A bit of physical page no. flipped → another page

It sounds straightforward, but demands a lot of designs and tests.

The Impact of Rowhammer

Project Zero

[Exploiting the DRAM rowhammer bug to gain kernel privileges (Seaborn, 2015)]

News and updates from the Project Zero team at Google

https://github.com/google/rowhammer-test

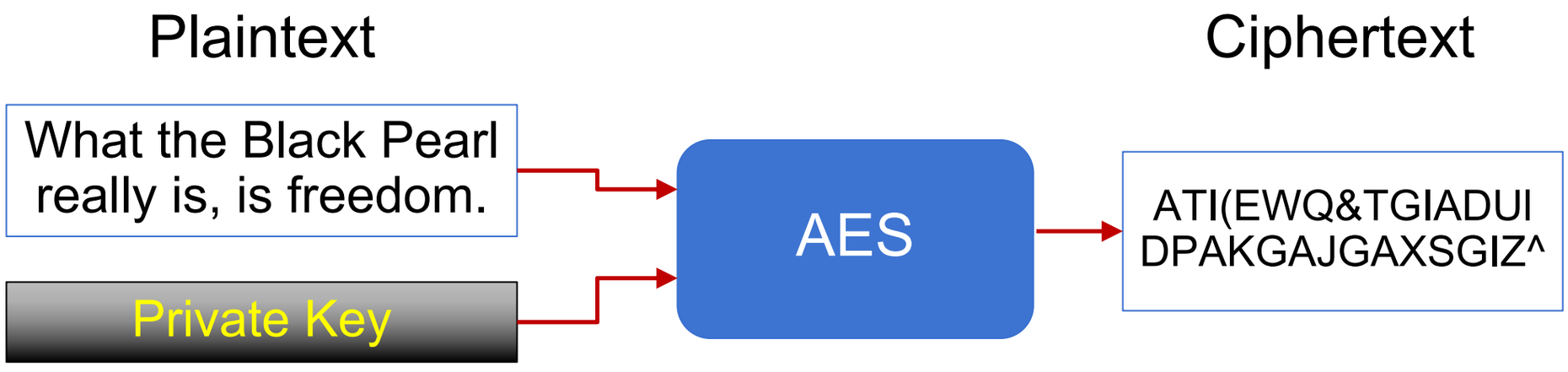
Monday, March 9, 2015

Exploiting the DRAM rowhammer bug to gain kernel privileges

ANVIL: Software-Based Protection Against Next-Generation Rowhammer Attacks (ASPLOS 2016)
http://dl.acm.org/citation.cfm?doid=2872362.2872390
CAn't Touch This: Software-only Mitigation against Rowhammer Attacks targeting Kernel Memory (USENIX Secuirty 2017) https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-brasser.pdf
Another Flip in the Wall of Rowhammer Defenses (IEEE S&P 2018) https://ieeexplore.ieee.org/document/8418607
Throwhammer: Rowhammer Attacks over the Network and Defenses (USENIX ATC 2018)
https://www.cs.vu.nl/~herbertb/download/papers/throwhammer_atc18.pdf
SpecHammer: Combining Spectre and Rowhammer for New Speculative Attacks (S&P 2022)
https://ieeexplore.ieee.org/abstract/document/9833802
CSI:Rowhammer - Cryptographic Security and Integrity against Rowhammer (S&P 2023)
https://www.computer.org/csdl/proceedings-article/sp/2023/933600a236

Mission Impossible: *When, or where*

- Encryption and decryption: e.g., advanced encryption standard (AES)
  - Used in OpenSSL and other applications
  - A key can be 128, 192, or 256 bits.



- Is it possible to guess a key?
  - Brute-force?

Side-channel Attack

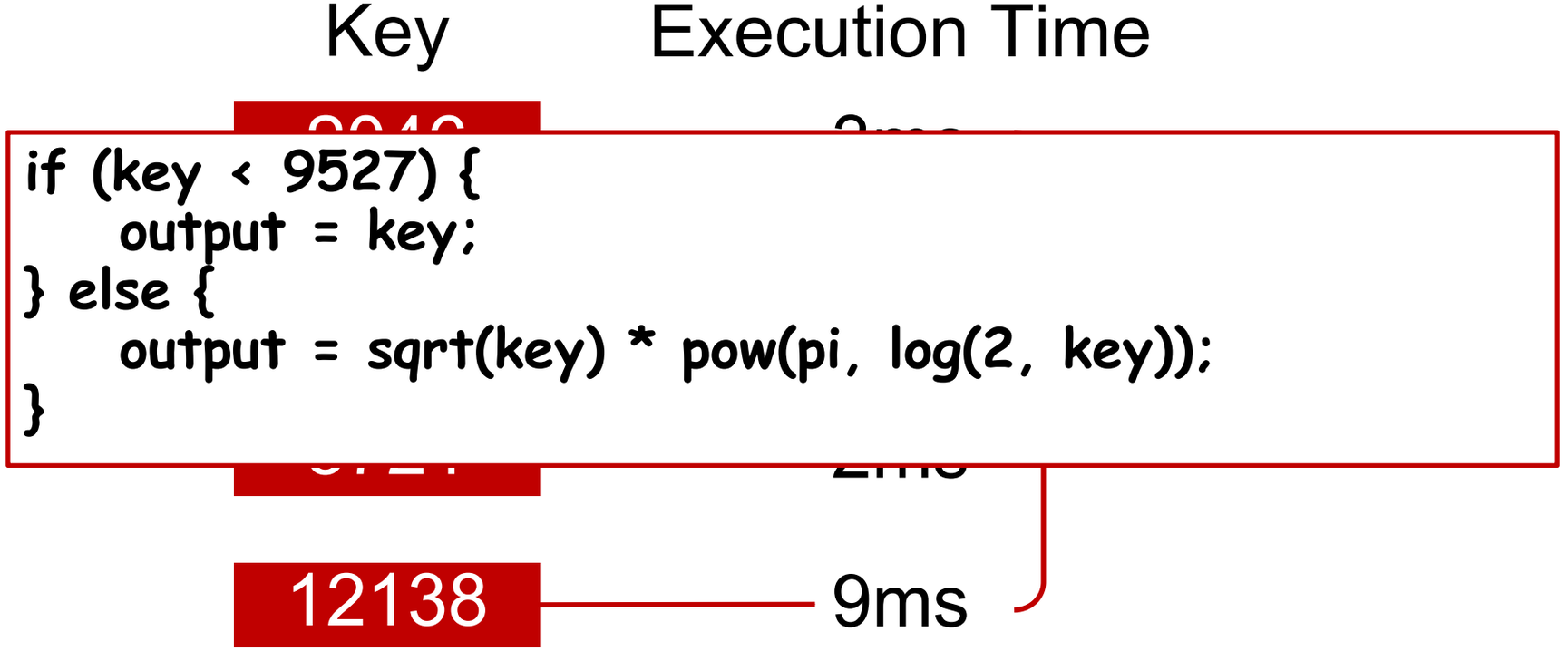
- Implementations have behaviours (observations) regarding different inputs
  - Timing
  - Accessed CPU cache lines
  - Power consumption
  - Sound

Side channels

- Not theorical properties of algorithm
- How to leverage a side channel?
  - First, to instrument, e.g., timing, power, etc.
  - More important, to figure out the relationship between observations via the side channel and inputs to the running program

Timing-based Attack

- When an implementation has data-dependent variations on its execution, it is prone to timing-based attacks.



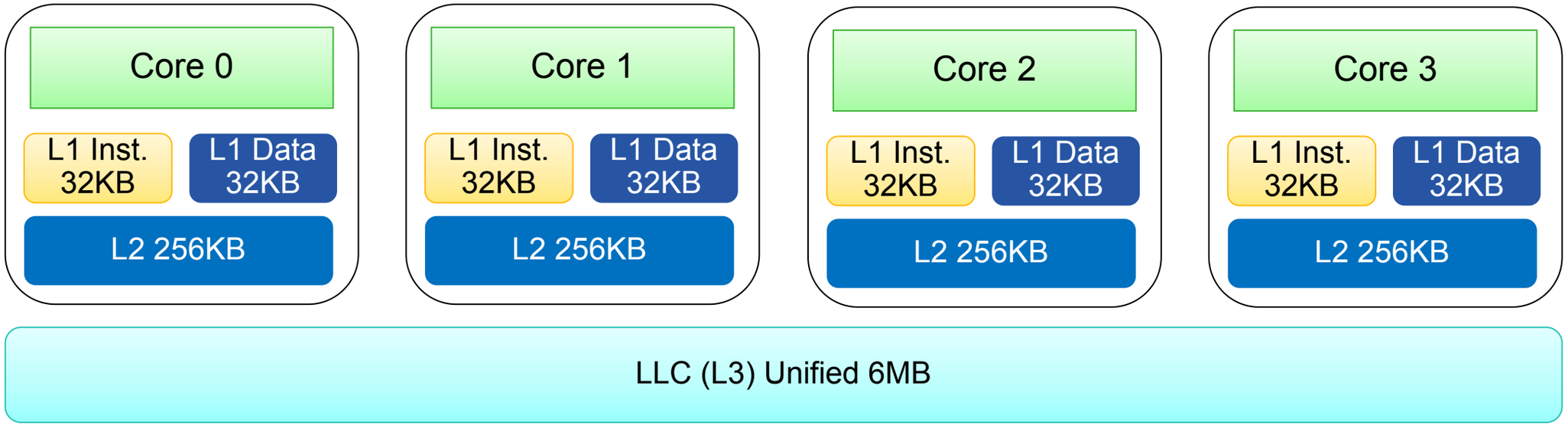
An Example

- Cachebleed
  - Reported in March 2016
  - Timing-based attack
    - On RSA in OpenSSL, RSA is a public-key cryptographic system for encryption and decryption
    - By detecting cache-bank conflicts via timing variations to reconstruct a key

Cachebleed https://ts.data61.csiro.au/projects/TS/cachebleed/

The Water House: *Flush the Loch Ness*

- Inclusive cache and shared pages
  - For an inclusive cache, a cache line in higher-level caches is in LLC.
  - Programs may share memory pages, e.g., the same executable files.



Intel Ivy Bridge Cache Architecture (Core i5-3470)

Flush & Reload

- To build a side channel
  - Manually share memory pages between attacker's and victim's programs
  - Attacker forcefully flushes and reloads cache lines
    - To observe the victim's cache misses/hits
    - To speculate the victim's secret
- Why inclusive cache for study?
  - All levels would be flushed

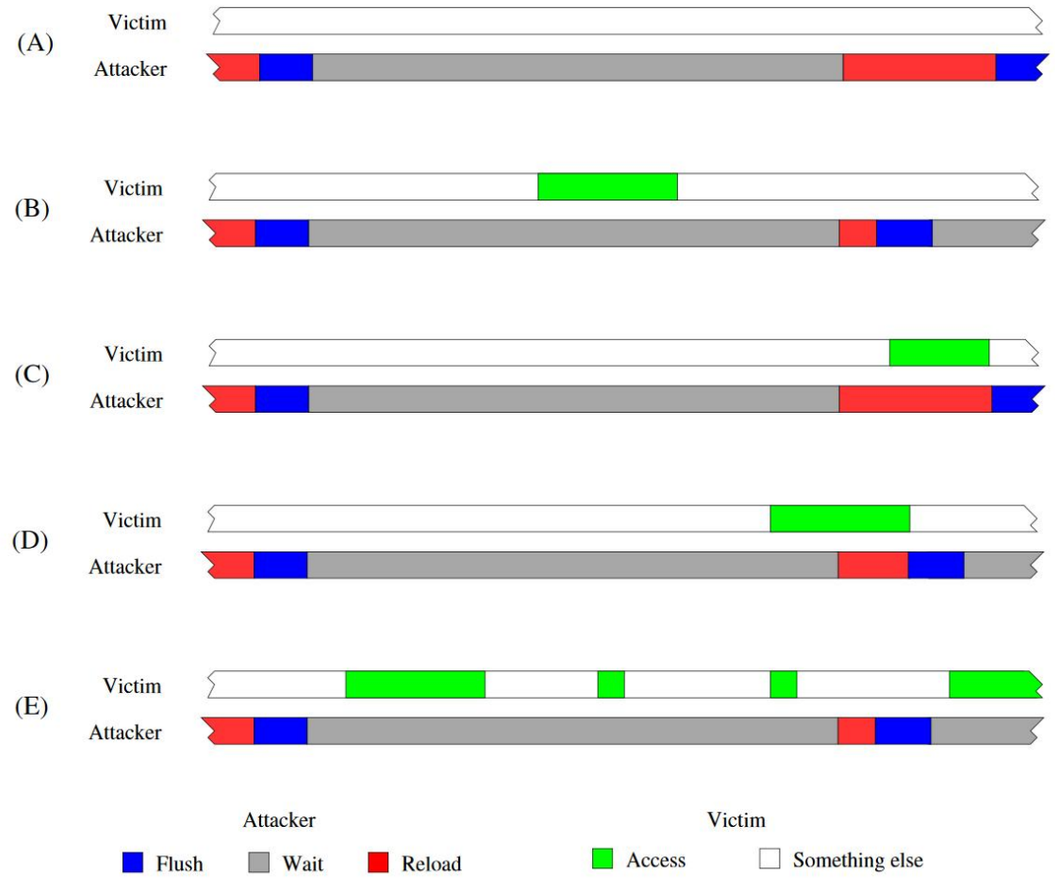


Figure 3: Timing of FLUSH+RELOAD. (A) No Victim Access (B) With Victim Access (C) Victim Access Overlap (D) Partial Overlap (E) Multiple Victim Accesses

https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-yarom.pdf

Flush & Reload Example

- GNU Privacy Guard (GnuPG) with RSA
  - GnuPG is open-source
  - But key for encryption/decryption is private
- Attacker memory-maps victim's executable file to her/his memory space, to share pages
  - Flush and reload cache lines to observe
  - `clflush` for x86: cache line flush

Flush & Reload Defense

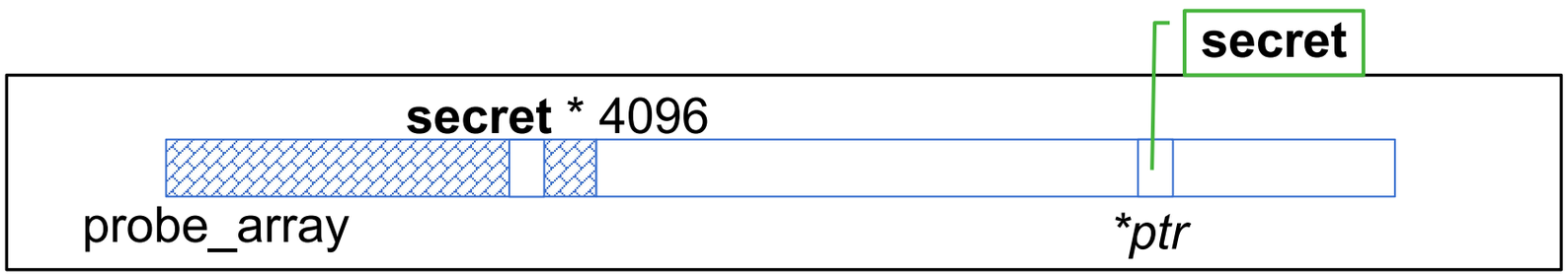
- Permission check for `clflush`
- Disallowance of memory sharing
  - Contradictory to the increase of sharing in OS and virtual machines
- Tuning software programs

Meltdown & Spectre

- Hardware vulnerability
  - Affecting Intel x86 microprocessors, IBM POWER processors, and some ARM-based microprocessors
- All Operating Systems affected!
- They are considered “catastrophic”!
- Allow to read all memory (e.g. from other process or other Virtual Machines (e.g. other users data on Amazon cloud service!))
- How Meltdown and Spectre work covers all knowledge of CA course:
  - Virtual Memory; Protection Levels; Instruction Pipelining; Out-of-order Execution; Speculative Execution; CPU Caching.

Meltdown: Out-of-order Execution

- Out of order execution: some instructions executed in advance



// secret is one-byte. probe_array is an array of char.

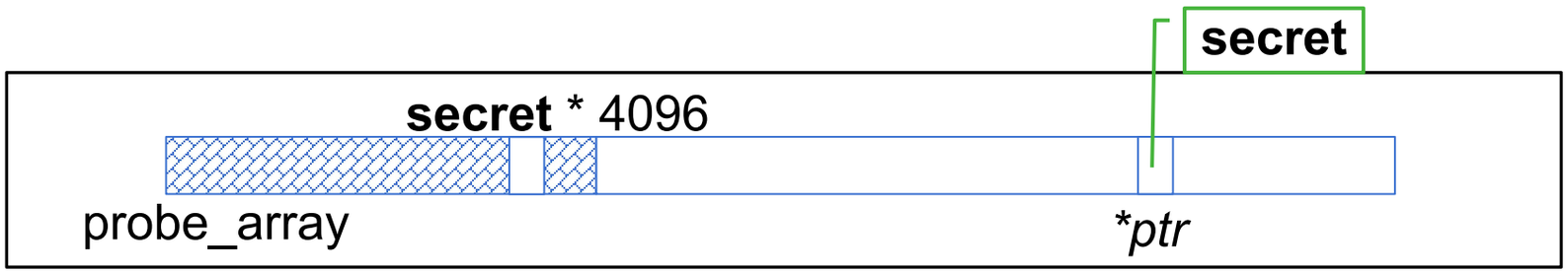
1. raise_exception();
2. // the line below is never reached
3. access(probe_array[secret * 4096])

The aim of Meltdown: to leak/dump memory

- probe_array should never be accessed, but accessed at some location probe_array + secret * 4096.
- probe_array is fully controlled by attacker who can use Flush+Reload to see which cache line of probe_array is hit, so as to figure out the value of secret.
- secret can be the value at any memory location, i.e., *ptr

The Impact of Meltdown

- The researchers put a value of 84 in `secret` and managed to use Flush+Reload to get a cache hit at the 84th page.



- The researchers developed competent programs to read memory locations that should be inaccessible to their program. They managed to dump the entire physical memory, for kernel and users.

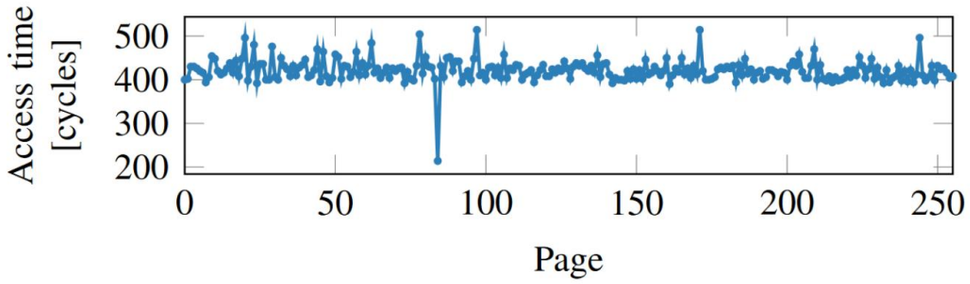


Figure 4: Even if a memory location is only accessed during out-of-order execution, it remains cached. Iterating over the 256 pages of `probe_array` shows one cache hit, exactly on the page that was accessed during the out-of-order execution.

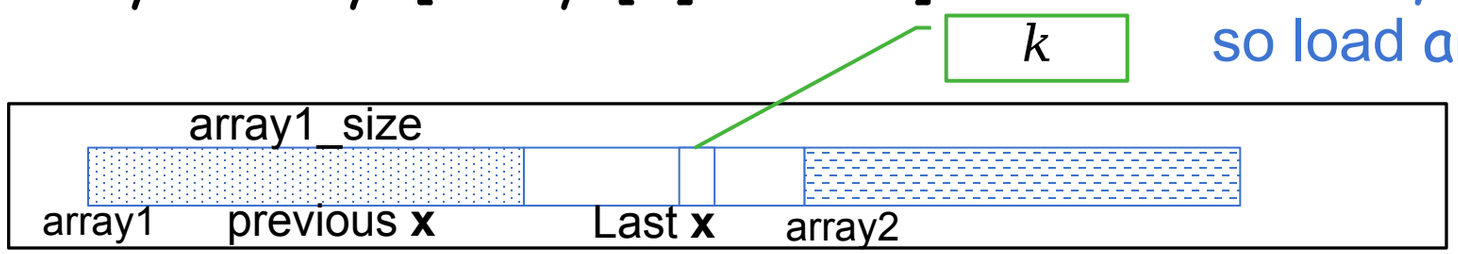
Meltdown https://meltdownattack.com/meltdown.pdf

Spectre: Speculative Execution

- Speculative execution, e.g., branch prediction
- Prerequisites:
  - i. array1[x], with an out-of-bound x larger than array1_size, resolves to a secret byte k that is cached;
  - ii. array1_size and array2 uncached.
  - iii. Previous x values have been valid.

The aim of Spectre: to read out a victim's sensitive information

```
// x is controlled by attacker.
1.  if (x < array1_size)          ← cache miss, so run next line due to prediction
2.    y = array2[array1[x] * 4096]   ← array1[x] cache hit, as k is cached, so load array2[k * 4096]
```



- Regarding a misprediction with an illegal x, array2[k * 4096]will not be used, but has been loaded into CPU cache.
- We can use Flush+Reload to guess k with array2.

The Impact of Spectre

- Processors can be tricked in speculative execution to modify cache state
  - Leaving attackers an exploitable opportunity
- Sensitive information of a victim program may be leaked
- Speculative Store Bypass
  - A newer variant of Spectre (v4) could allow an attacker to retrieve older but stale values in a CPU's stack or other memory locations.
  - https://software.intel.com/security-software-guidance/software-guidance/speculative-store-bypass

Spectre https://spectreattack.com/spectre.pdf

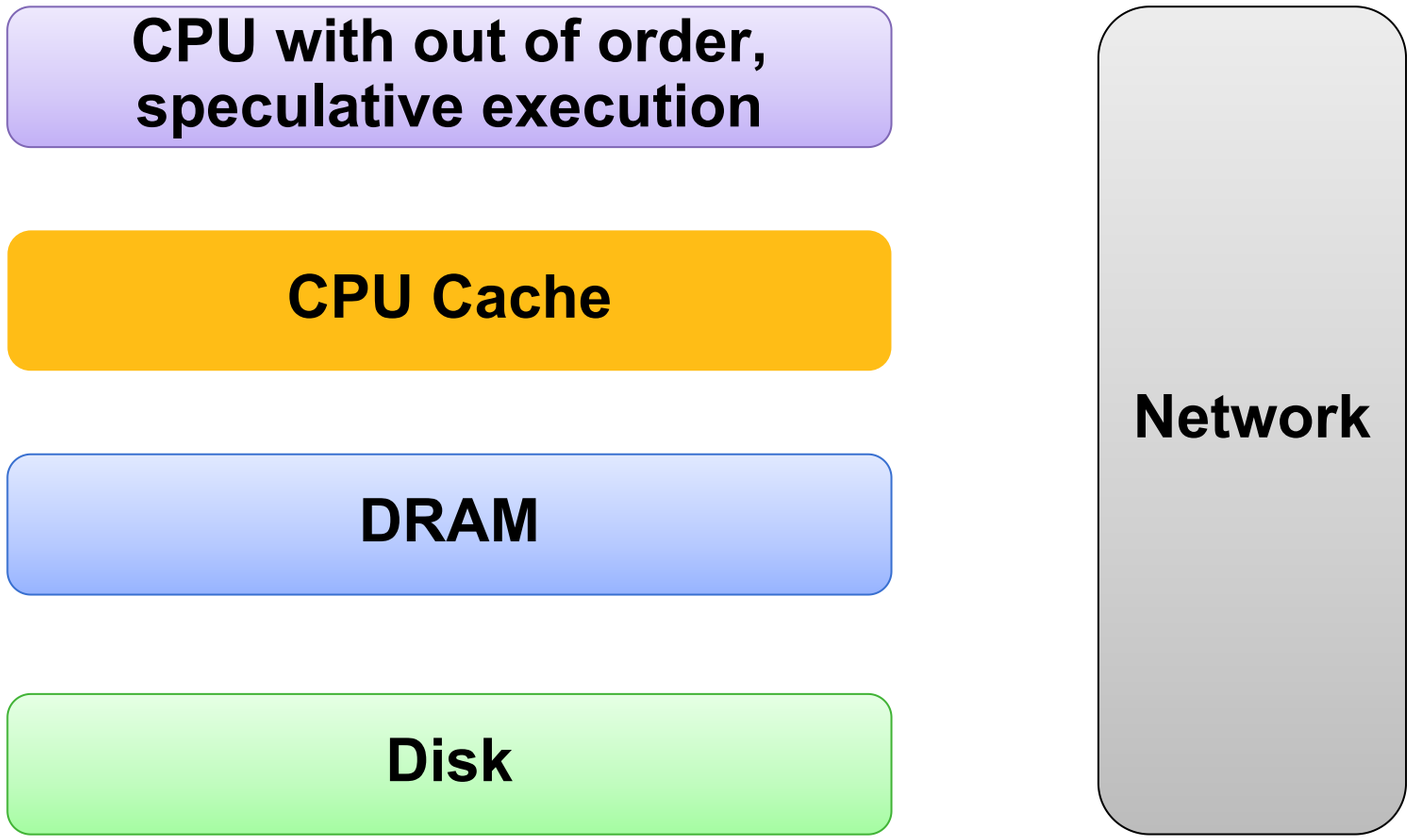
Meltdown & Spectre

- More complicated than examples here
- Multiple variants today
- Many processors, OSes, applications affected
  - PC, mobile devices, cloud
- Many proposals to mitigate their impacts

No announced RISC-V silicon is susceptible, and the popular open-source RISC-V Rocket processor is unaffected as it does not perform memory accesses speculatively. https://riscv.org/2018/01/more-secure-world-risc-v-isa/

However, there is a workshop paper "*Replicating and Mitigating Spectre Attacks on a Open-Source RISC-V Microarchitecture*" https://carrv.github.io/2019/papers/carrv2019_paper_5.pdf

Vulnerable Architecture



Conclusion

- Every part of a computer can be vulnerable
  - Be vigilant and attentive in designing and programming
- Security and privacy have different presences
  - More than DoS, DDoS, virus, Trojan, ransomware, spyware, and phishing emails
- The challenges for a computer architect
  - To rule out any possibility of vulnerabilities
  - To achieve both high performance and security