Hardware Vulnerabilities

CS-576 Systems Security

Instructor: Georgios Portokalidis

Fall 2018

Status Quo … Some Years Ago

Software is faulty and includes vulnerabilities

- Lack of memory safety, insecure design, programmer errors, …
- Integrity can be compromised, and confidential data leaked

Hardware is correct and can be trusted

- Main problem is the outsourcing of manufacturing
- Can we trust that the chips we import contain only the logic that was part of the original design?

New Attitude Towards HW

Hardware complexity has risen significantly

- To deliver more performance despite the end of Moore's law
  - *Moore's law is the observation that the number of transistors in a dense integrated circuit doubles about every two years. --wikipedia*
- To deliver new features
  - Virtualization, security, and other extensions

```
$ cat /proc/cpuinfo
…
flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2
ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl xtopology
nonstop_tsc aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid
sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch epb
invpcid_single kaiser tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms
invpcid rtm mpx rdseed adx smap clflushopt intel_pt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp
hwp_notify hwp_act_window hwp_epp
…
```

- To increase performance despite other limitations
  - Memory bandwidth, non-parallelized workloads

New Attitude Towards HW

Hardware complexity has risen significantly

- To deliver more performance despite the end of Moore's law
  - *Moore's law is the observation that the number of transistors in a dense integrated circuit doubles about every two years. --wikipedia*
- To deliver new features
  - Virtualization, security, and other extensions

```
$ cat /proc/cpuinfo
…
flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2
ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl xtopology
nonstop_tsc aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid
sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch epb
invpcid_single kaiser tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms
invpcid rtm mpx rdseed adx smap clflushopt intel_pt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp
hwp_notify hwp_act_window hwp_epp
…
```

- To increase performance despite other limitations
  - Memory bandwidth, non-parallelized workloads

Hardware bugs are now a reality!

Multiple Different Bugs Discovered

Rowhammer: corruption of bits (bit flips) in modern DRAM

Cache-based side channels

Speculative execution bugs

Memory Hierarchies

Random-Access Memory (RAM)

FROM: Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Key features

- RAM is traditionally packaged as a chip.
- Basic storage unit is normally a cell (one bit per cell).
- Multiple RAM chips form a memory.

RAM comes in two varieties:

- SRAM (Static RAM)
- DRAM (Dynamic RAM)

DRAM

SRAM vs DRAM Summary

	Transistors per bit	Access time	Needs refresh?	Needs EDC*?	Cost	Applications
SRAM	4 or 6	1X	No	Maybe	100x	Cache memories
DRAM	1	10X	Yes	Yes	1X	Main memories, frame buffers

*Error detection and correction

Nonvolatile Memories

DRAM and SRAM are volatile memories

- Lose information if powered off.

Nonvolatile memories retain value even if powered off

- Read-only memory (ROM): programmed during production
- Programmable ROM (PROM): can be programmed once
- Eraseable PROM (EPROM): can be bulk erased (UV, X-Ray)
- Electrically eraseable PROM (EEPROM): electronic erase capability
- Flash memory: EEPROMs. with partial (block-level) erase capability
  - Wears out after about 100,000 erasings

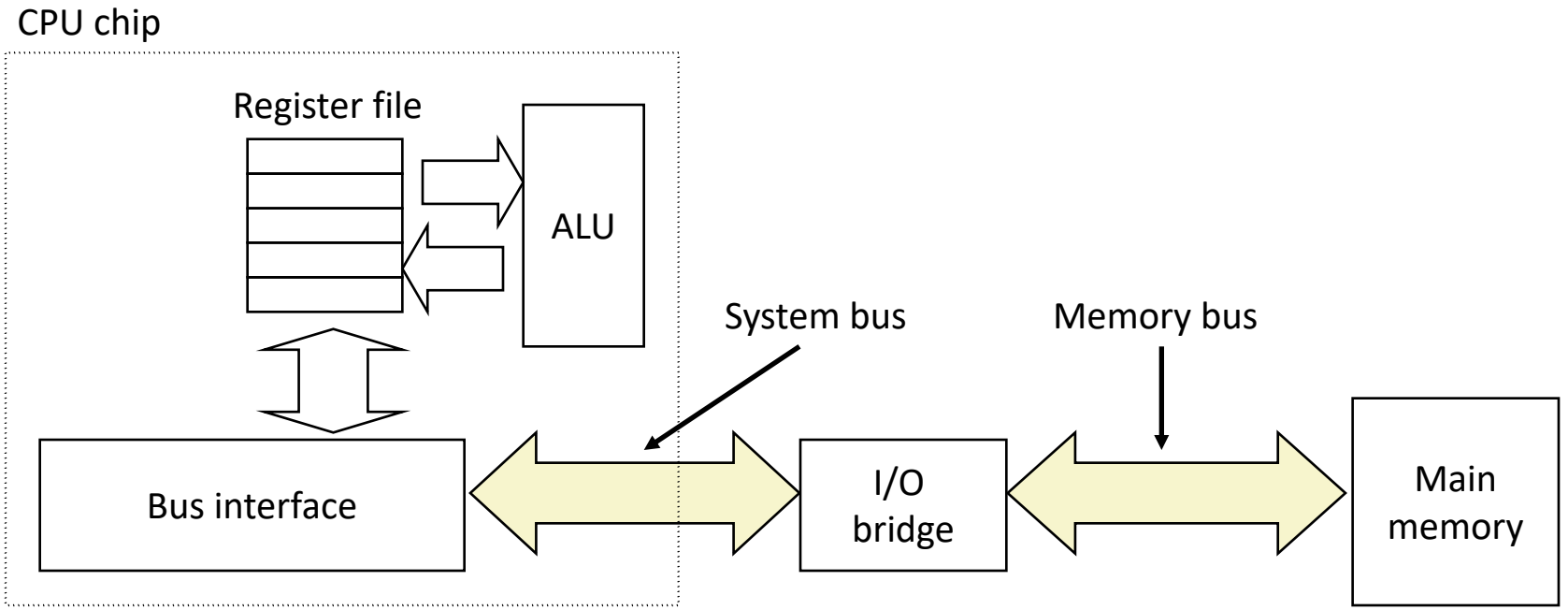
Uses for Nonvolatile Memories

- Firmware programs stored in a ROM (BIOS, controllers for disks, network cards, graphics accelerators, security subsystems,…)
- Solid state disks (replace rotating disks in thumb drives, smart phones, mp3 players, tablets, laptops,…)
- Disk caches

Connecting CPU and Memory

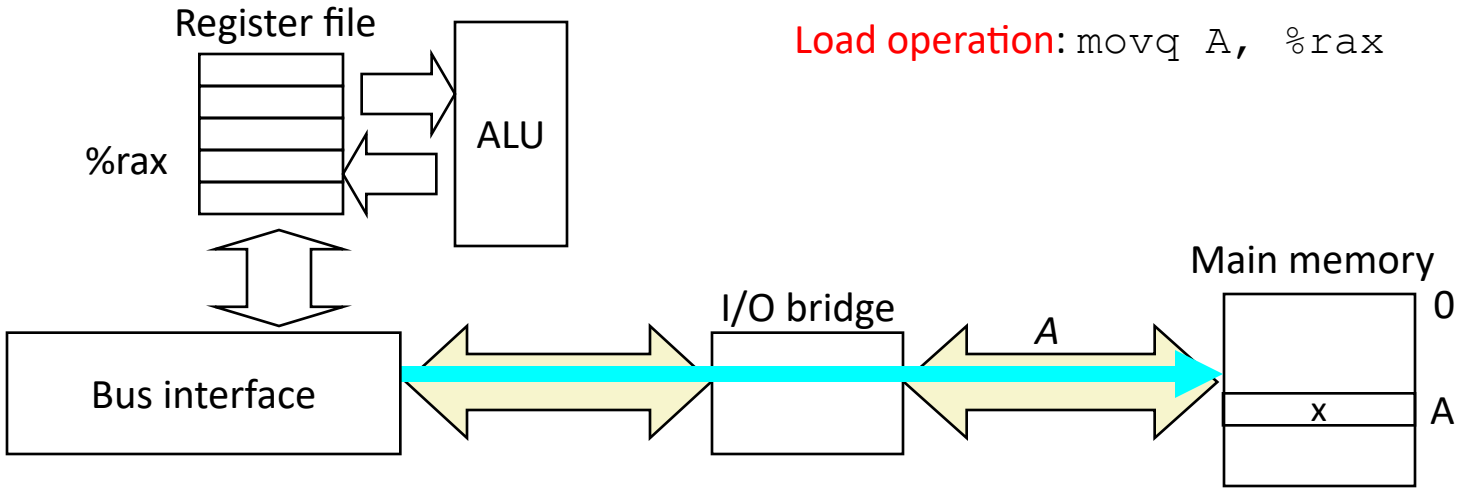
A bus is a collection of parallel wires that carry address, data, and control signals.

Buses are typically shared by multiple devices.



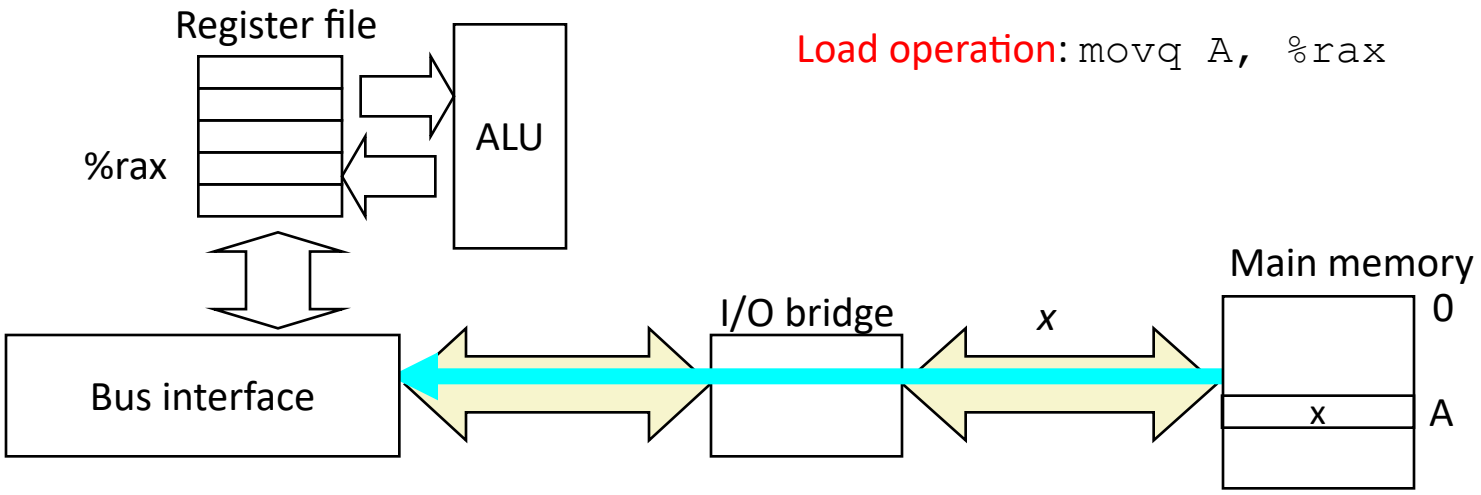
Memory Read Transaction (1)

CPU places address A on the memory bus.



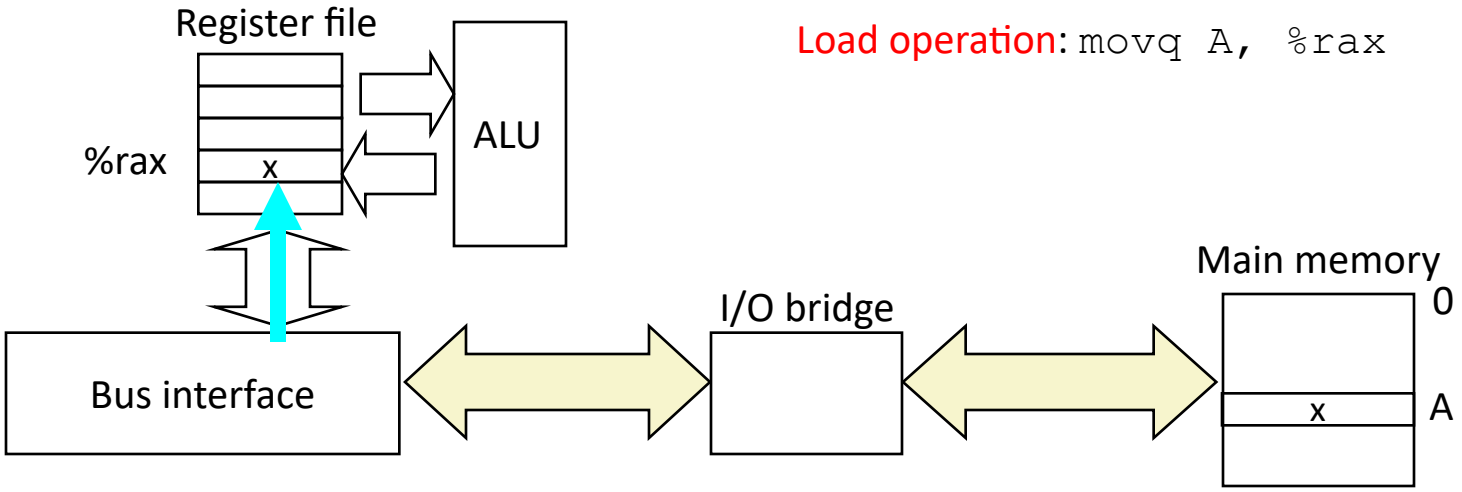
Memory Read Transaction (2)

Main memory reads A from the memory bus, retrieves word x, and places it on the bus.



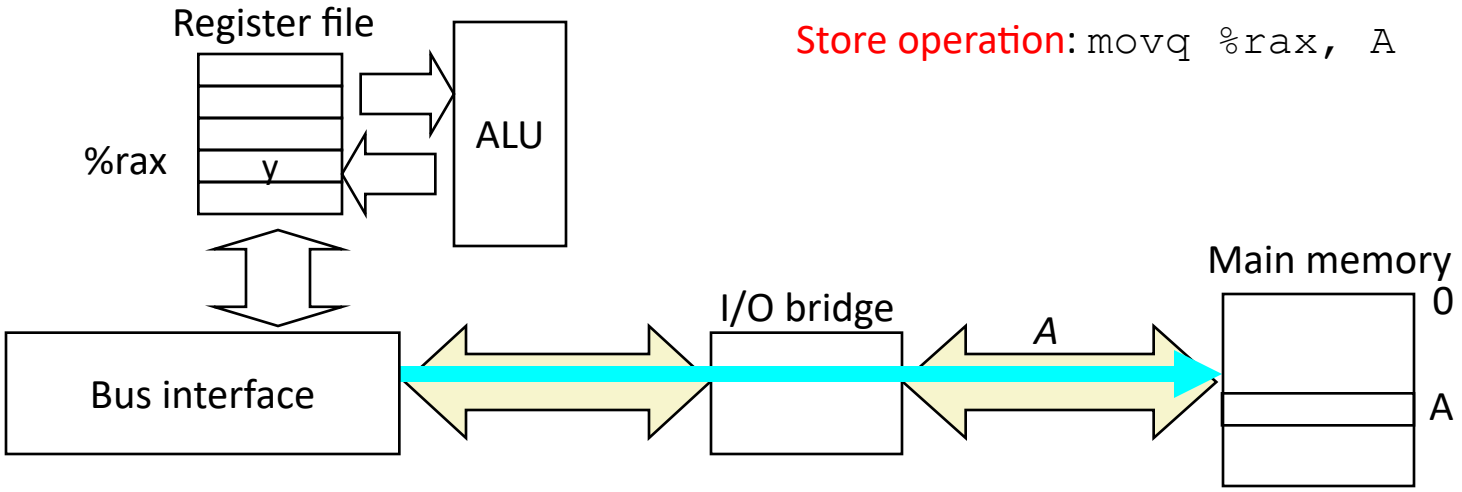
Memory Read Transaction (3)

CPU read word x from the bus and copies it into register %rax.



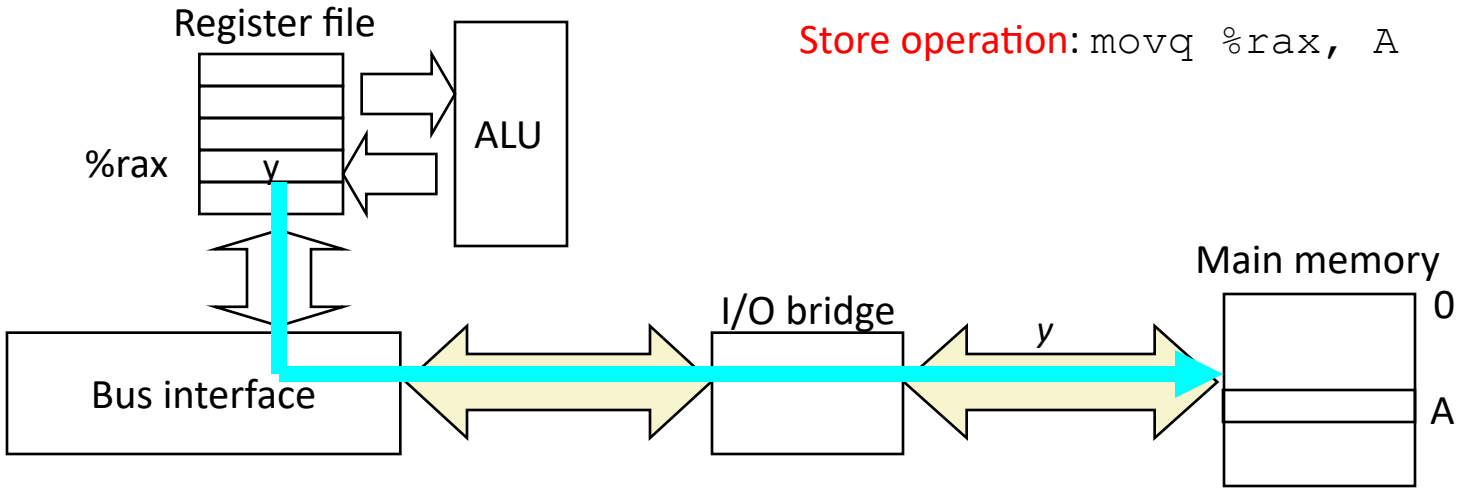
Memory Write Transaction (1)

CPU places address A on bus. Main memory reads it and waits for the corresponding data word to arrive.



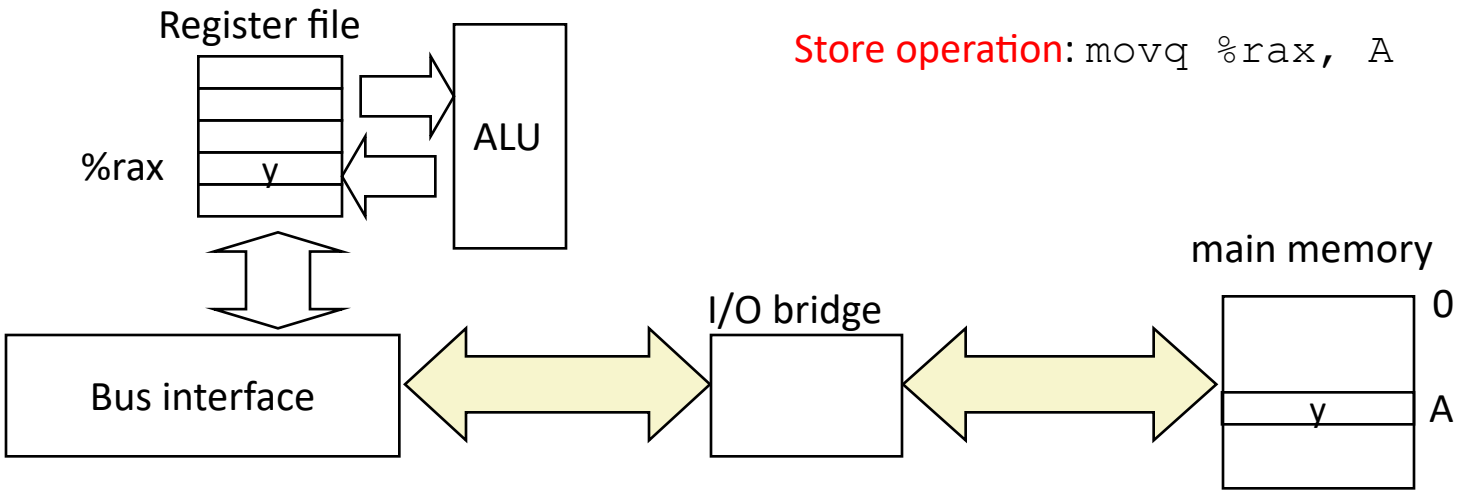
Memory Write Transaction (2)

CPU places data word y on the bus.



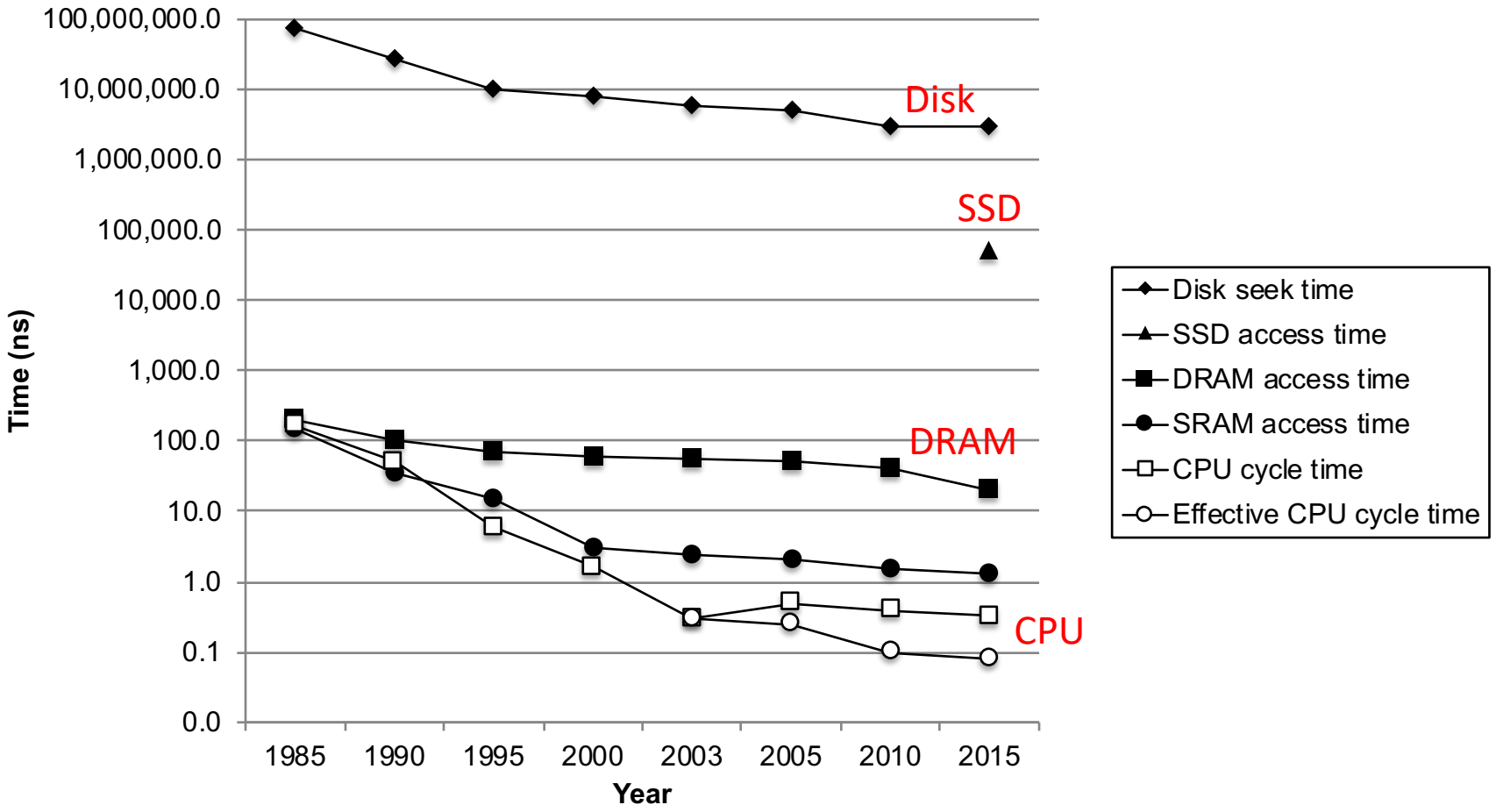
Memory Write Transaction (3)

Main memory reads data word y from the bus and stores it at address A.



The CPU-Memory Gap

The gap widens between DRAM, disk, and CPU speeds.

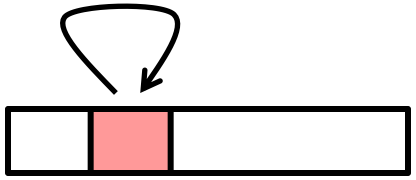


Locality to the Rescue!

The key to bridging this CPU-Memory gap is a fundamental property of computer programs known as locality

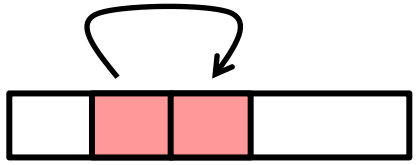
Locality

Principle of Locality: Programs tend to use data and instructions with addresses near or equal to those they have used recently



Temporal locality:

- Recently referenced items are likely to be referenced again in the near future



Spatial locality:

- Items with nearby addresses tend to be referenced close together in time

Caching in the Memory Hierarchy

Caches

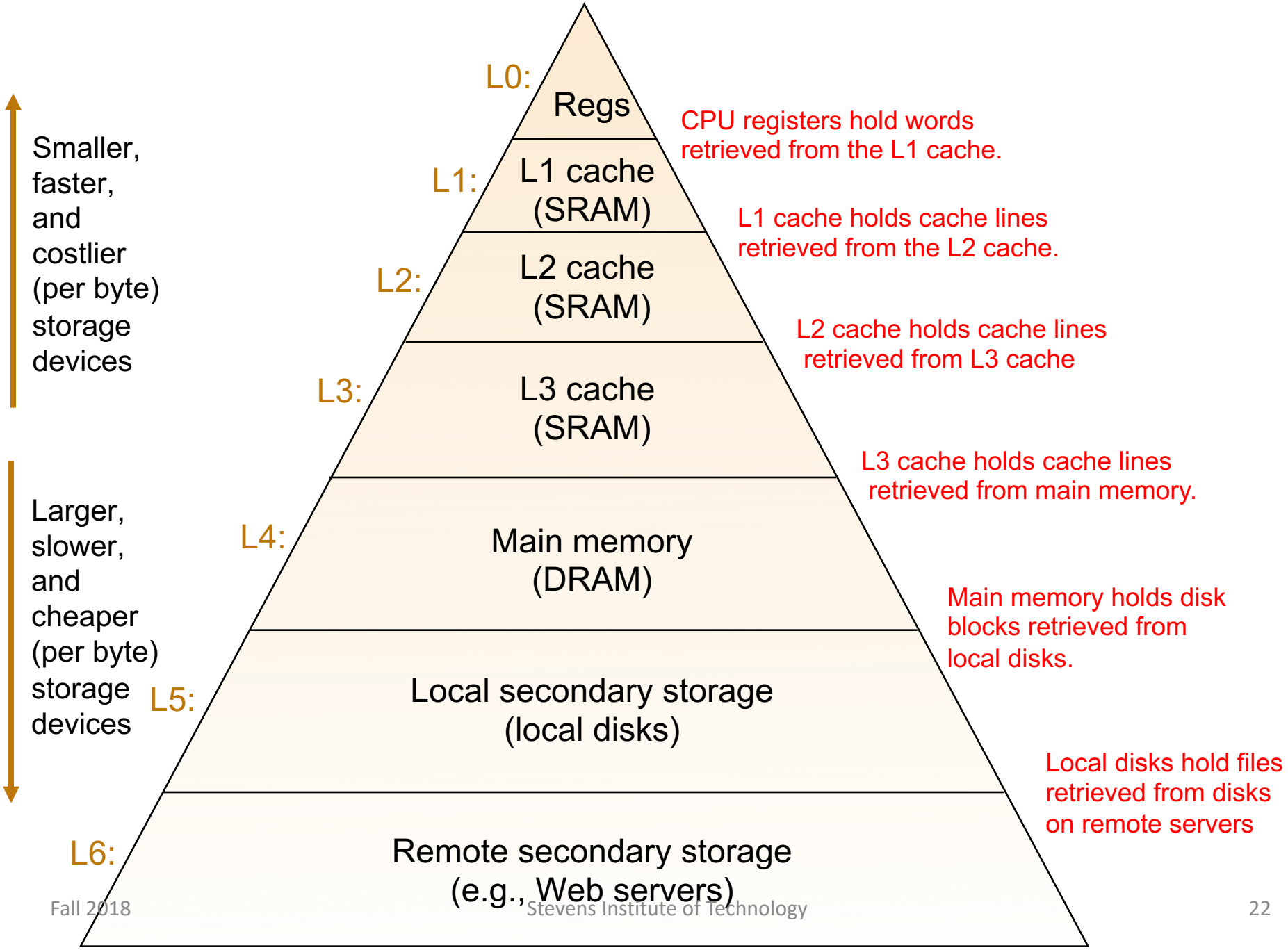
Cache: A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.

Fundamental idea of a memory hierarchy:

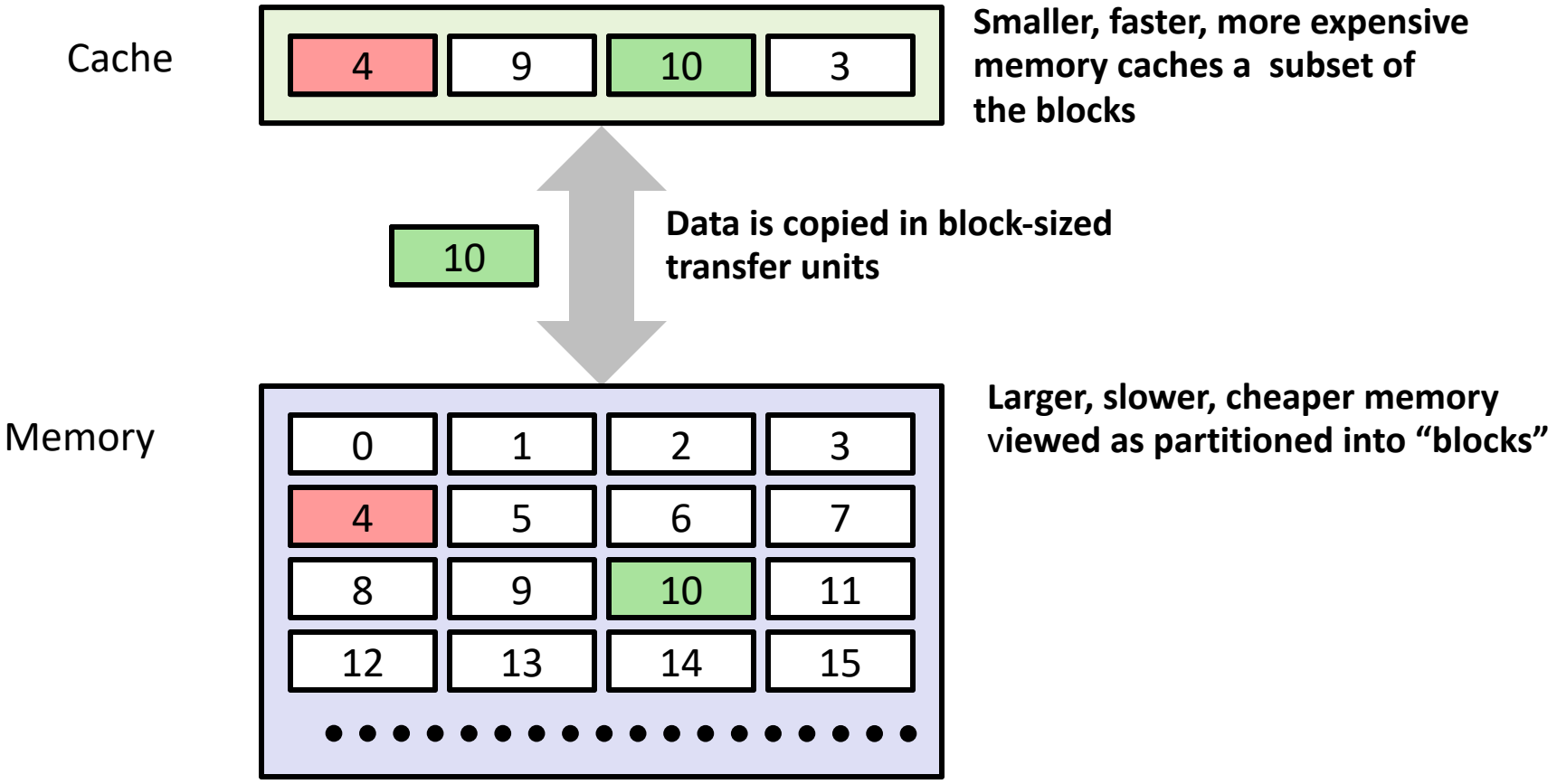
- For each k, the faster, smaller device at level k serves as a cache for the larger, slower device at level k+1.

Why do memory hierarchies work?

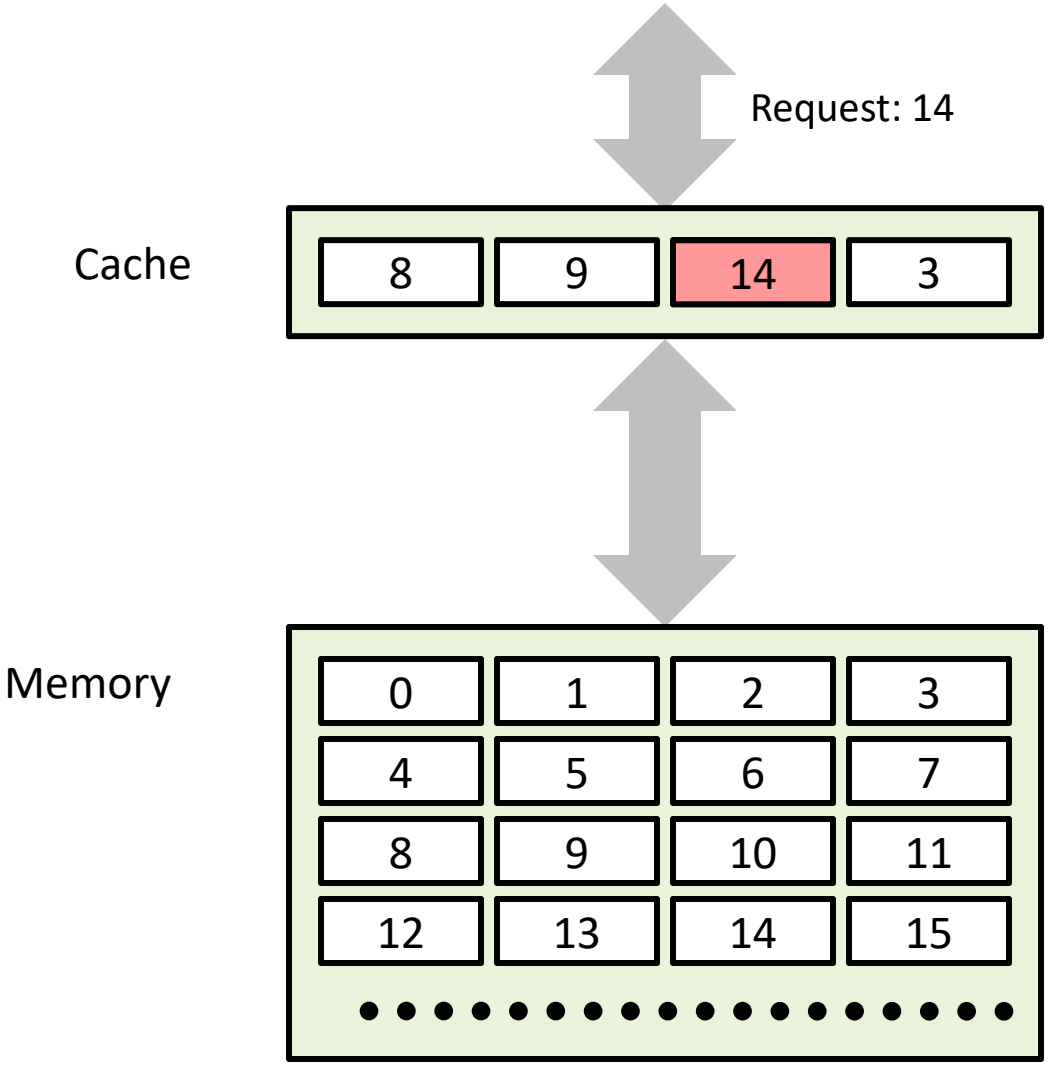
- Because of locality, programs tend to access the data at level k more often than they access the data at level k+1.
- Thus, the storage at level k+1 can be slower, and thus larger and cheaper per bit.


Smaller, faster, and costlier (per byte) storage devices
Larger, slower, and cheaper (per byte) storage devices
L0:
Regs
CPU registers hold words retrieved from the L1 cache.
L1:
L1 cache (SRAM)
L1 cache holds cache lines retrieved from the L2 cache.
L2:
L2 cache (SRAM)
L2 cache holds cache lines retrieved from L3 cache
L3:
L3 cache (SRAM)
L3 cache holds cache lines retrieved from main memory.
L4:
Main memory (DRAM)
Main memory holds disk blocks retrieved from local disks.
L5:
Local secondary storage (local disks)
Local disks hold files retrieved from disks on remote servers
L6:
Remote secondary storage (e.g., Web servers)

General Cache Concepts



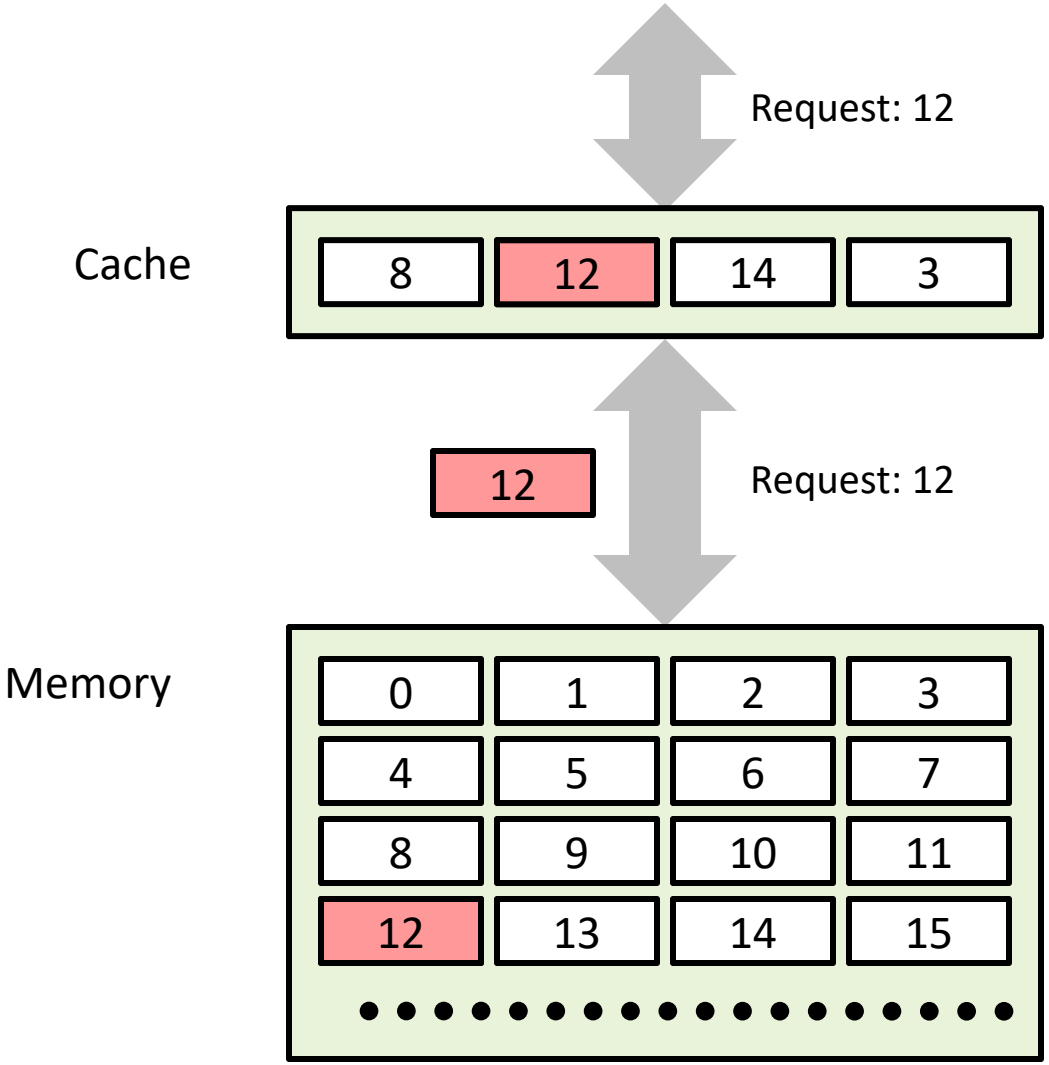
General Cache Concepts: Hit



Data in block b is needed

Block b is in cache:
Hit!

General Cache Concepts: Miss



Data in block b is needed

Block b is not in cache:
Miss!

Block b is fetched from *memory*

Block b is stored in cache

- Placement policy: determines where b goes
- Replacement policy: determines which block gets evicted (victim)

Types of Cache Misses

Cold (compulsory) miss

- Cold misses occur because the cache is empty.

Conflict miss

- Most caches limit blocks at level k+1 to a small subset (sometimes a singleton) of the block positions at level k.
  - E.g. Block i at level k+1 must be placed in block (i mod 4) at level k.
- Conflict misses occur when the level k cache is large enough, but multiple data objects all map to the same level k block.
  - E.g. Referencing blocks 0, 8, 0, 8, 0, 8, ... would miss every time.

Capacity miss

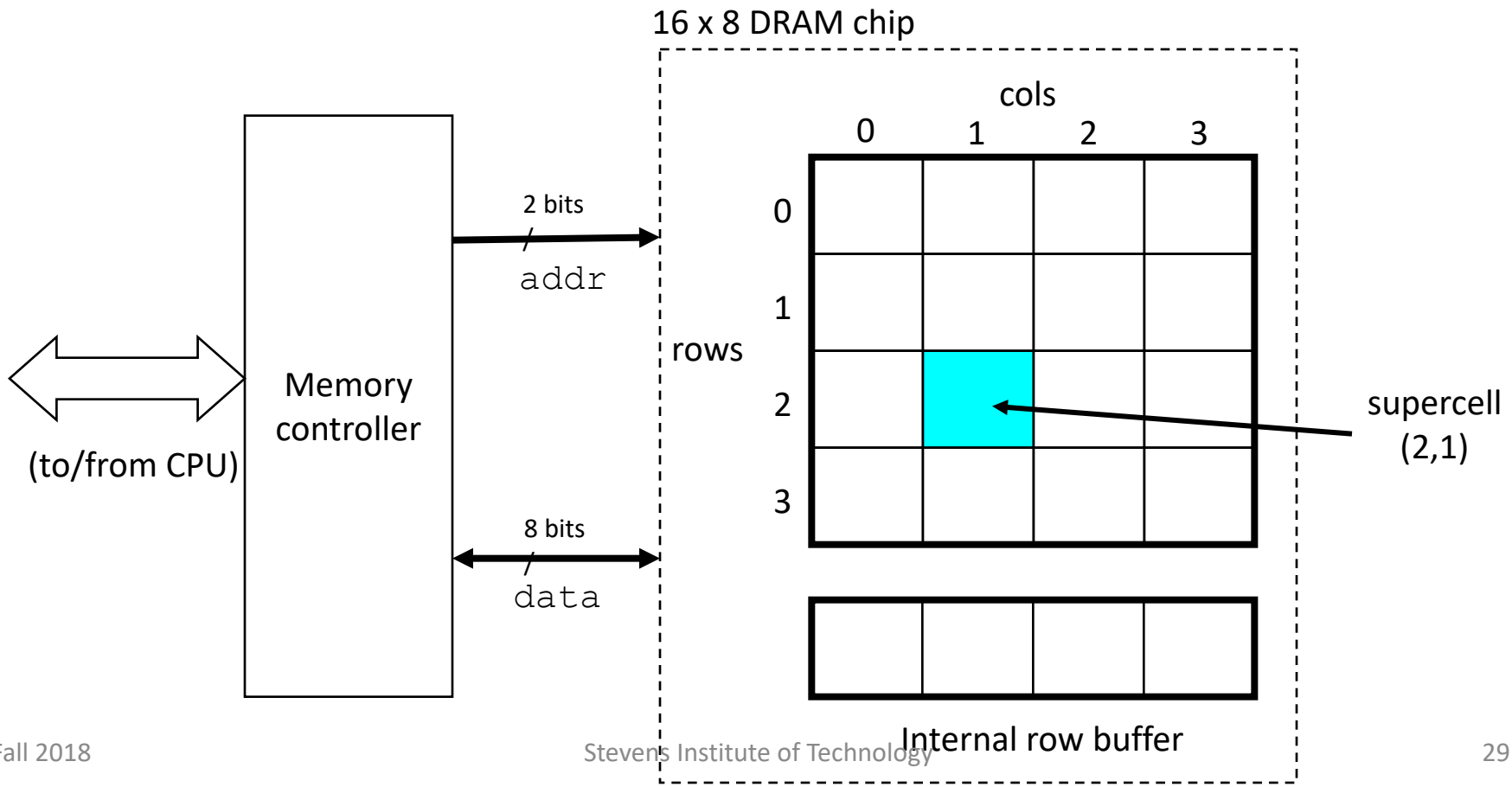
- Occurs when the set of active cache blocks (working set) is larger than the cache.

Dynamic RAM

Conventional DRAM Organization

d x w DRAM:

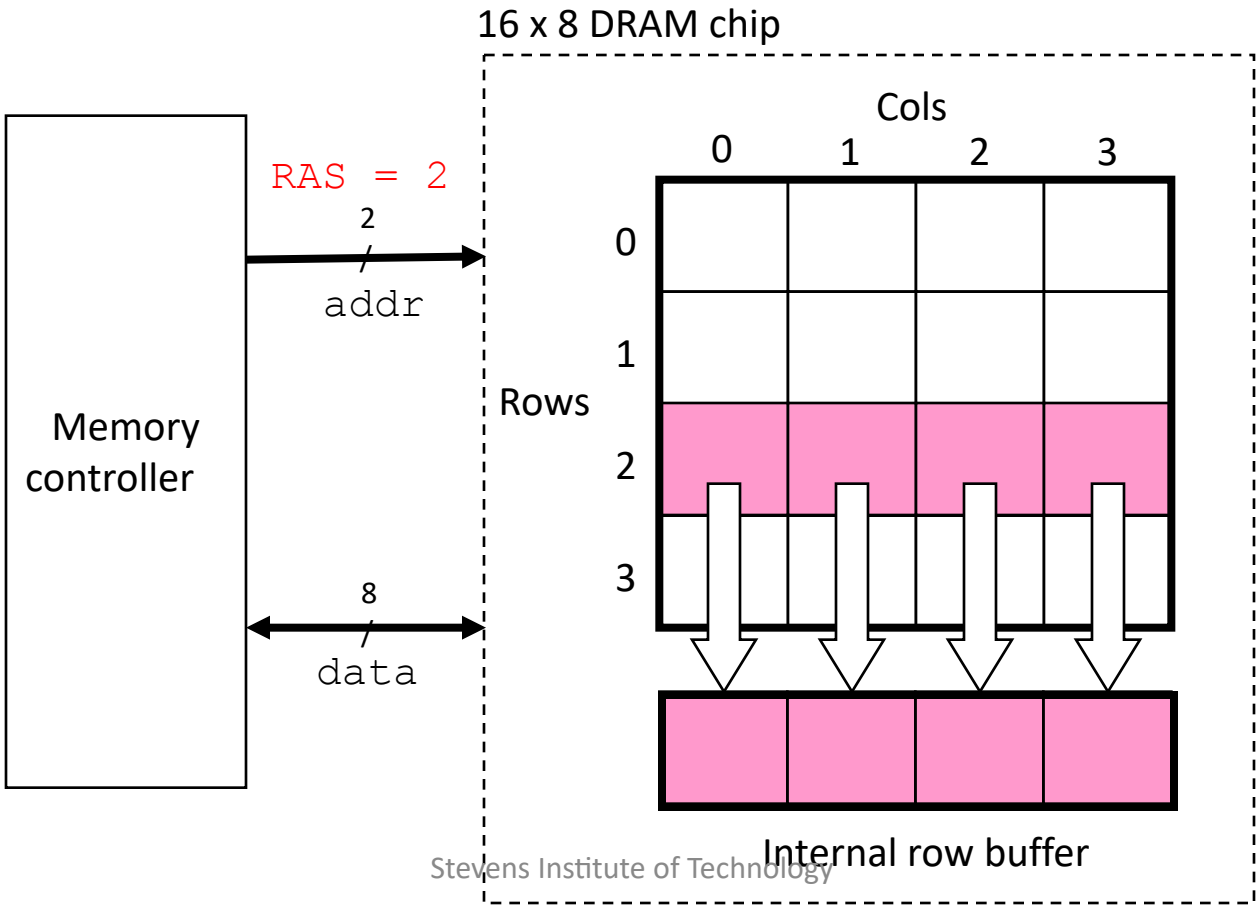
- dw total bits organized as d supercells of size w bits



Reading DRAM Supercell (2,1)

Step 1(a): Row access strobe (RAS) selects row 2.

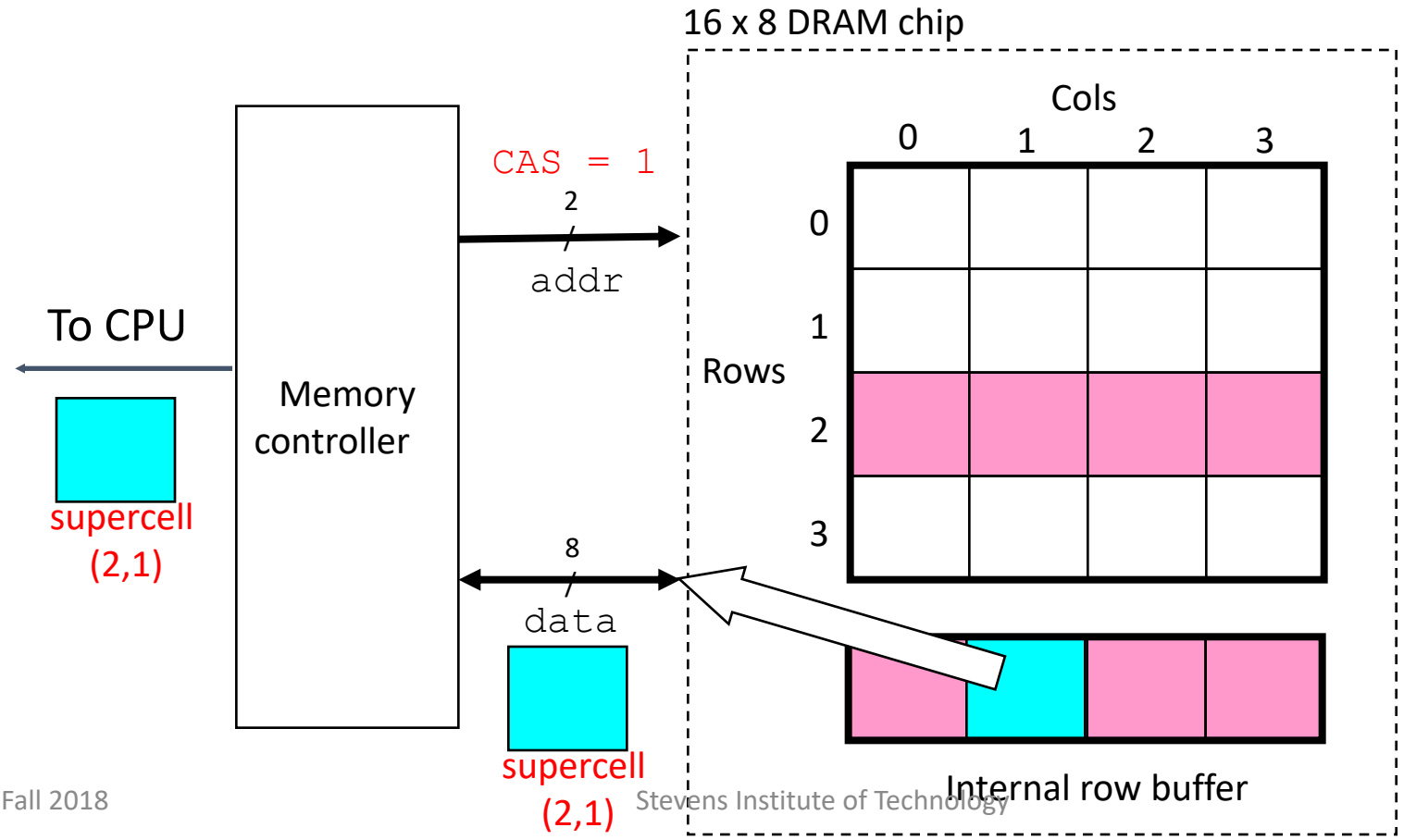
Step 1(b): Row 2 copied from DRAM array to row buffer.



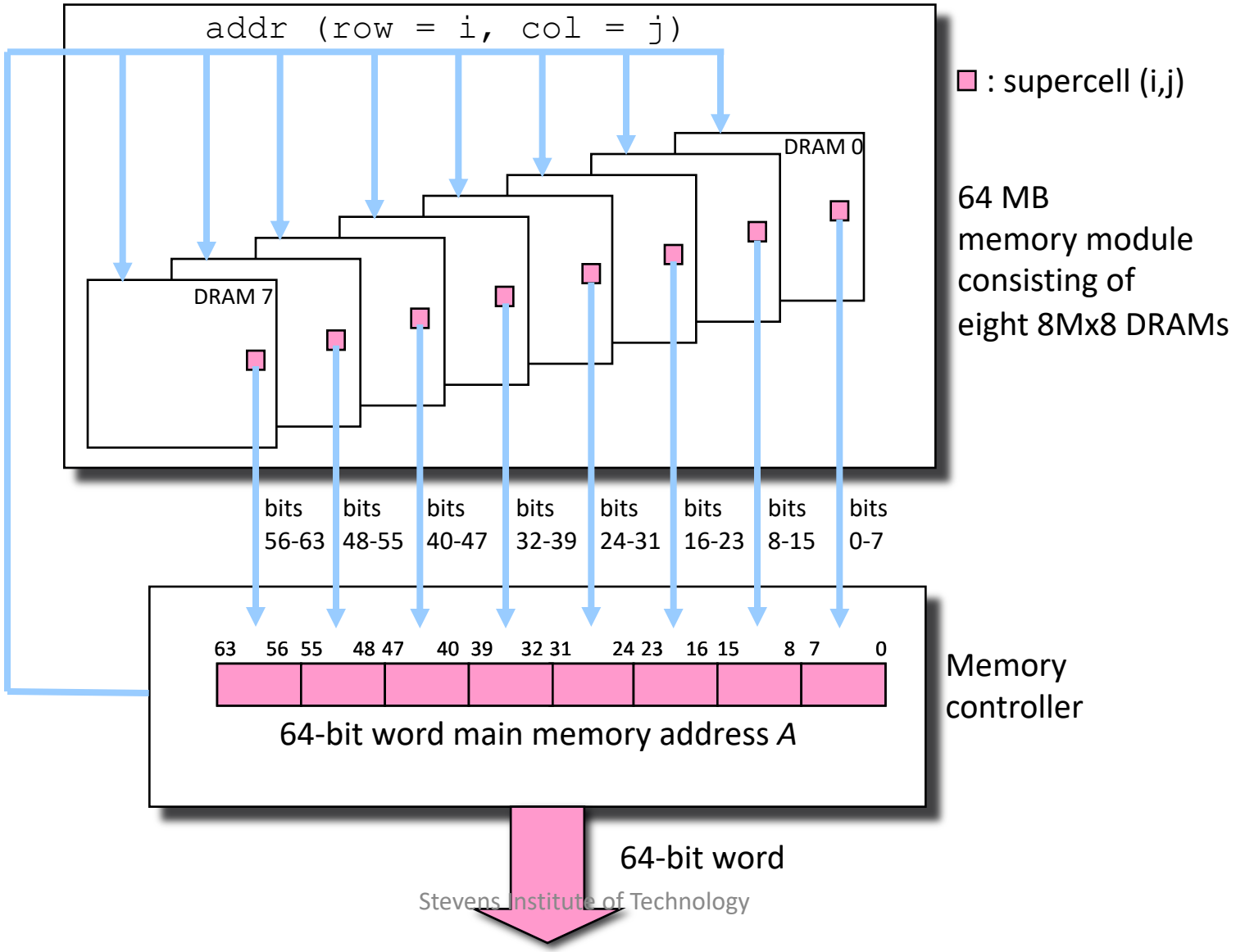
Reading DRAM Supercell (2,1)

Step 2(a): Column access strobe (CAS) selects column 1.

Step 2(b): Supercell (2,1) copied from buffer to data lines, and eventually back to the CPU.



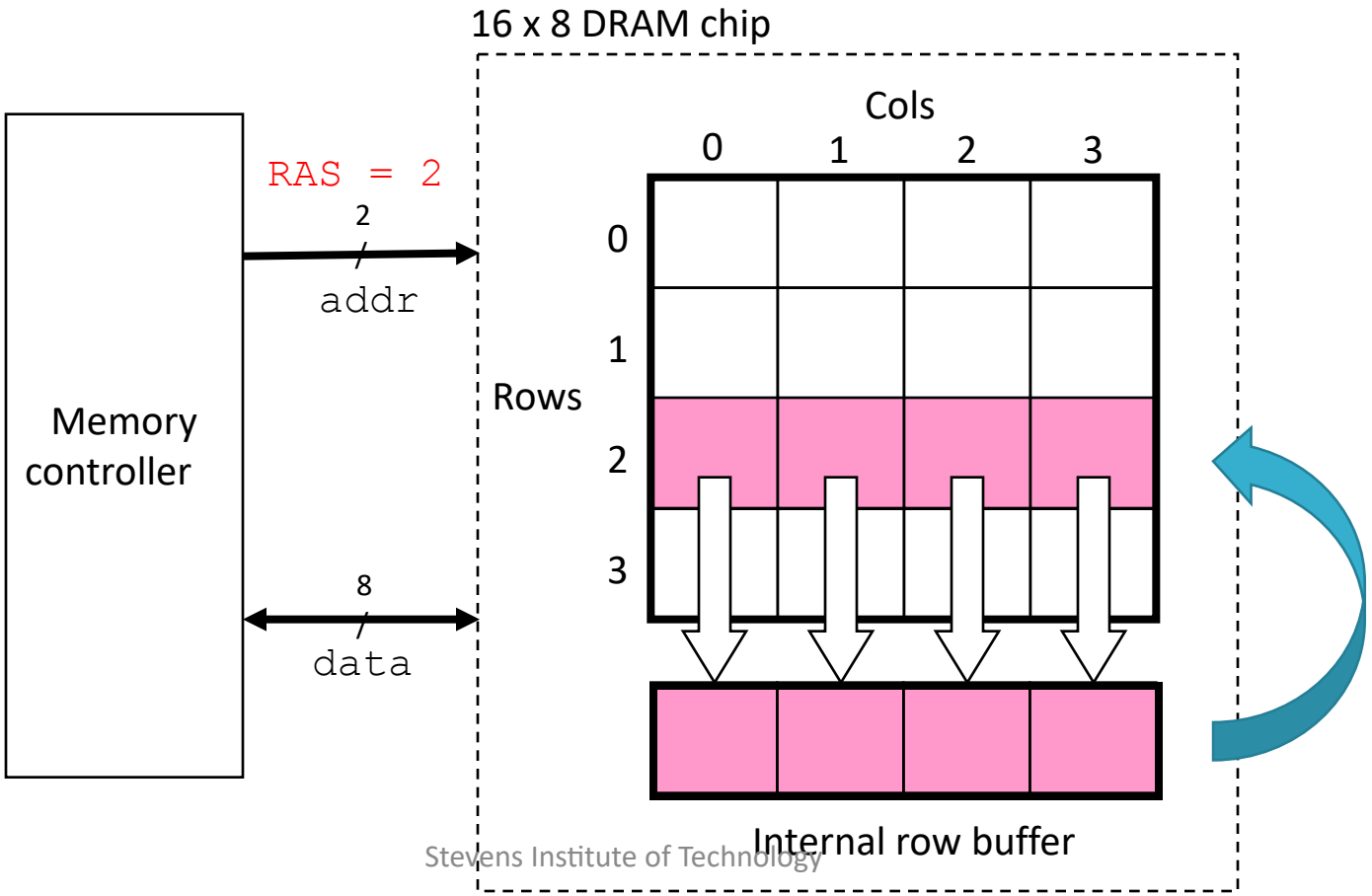
Memory Banks



DRAM Refresh

Electric charge is lost when reading

- Data need to be re-written in the cells



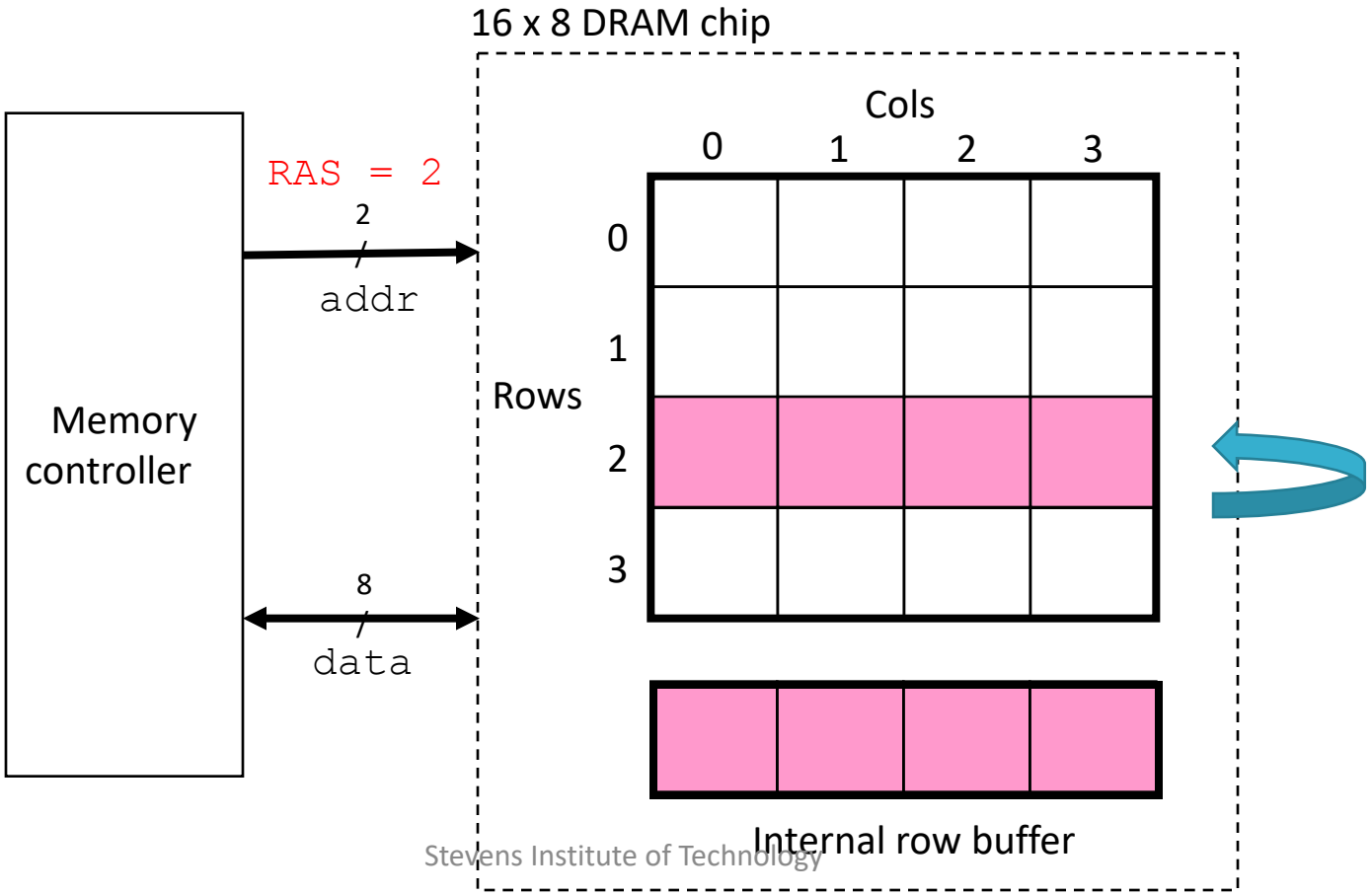
DRAM Refresh

Electric charge is lost when reading

- Data need to be re-written in the cells

Charge is also lost over time

- Data need to be refreshed



DRAM Disturbance Errors

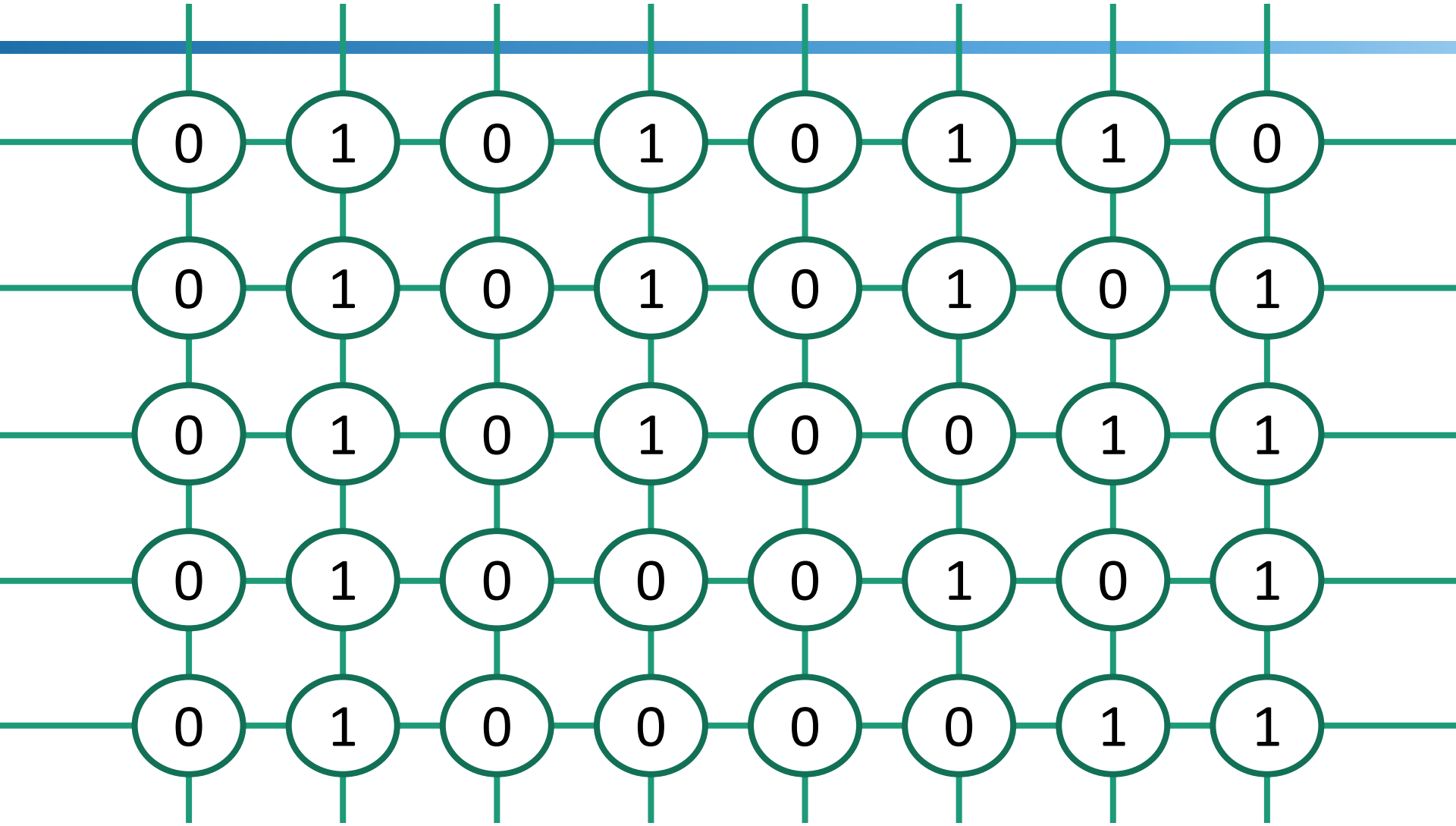
Memory cells (capacitors) have a natural discharge rate

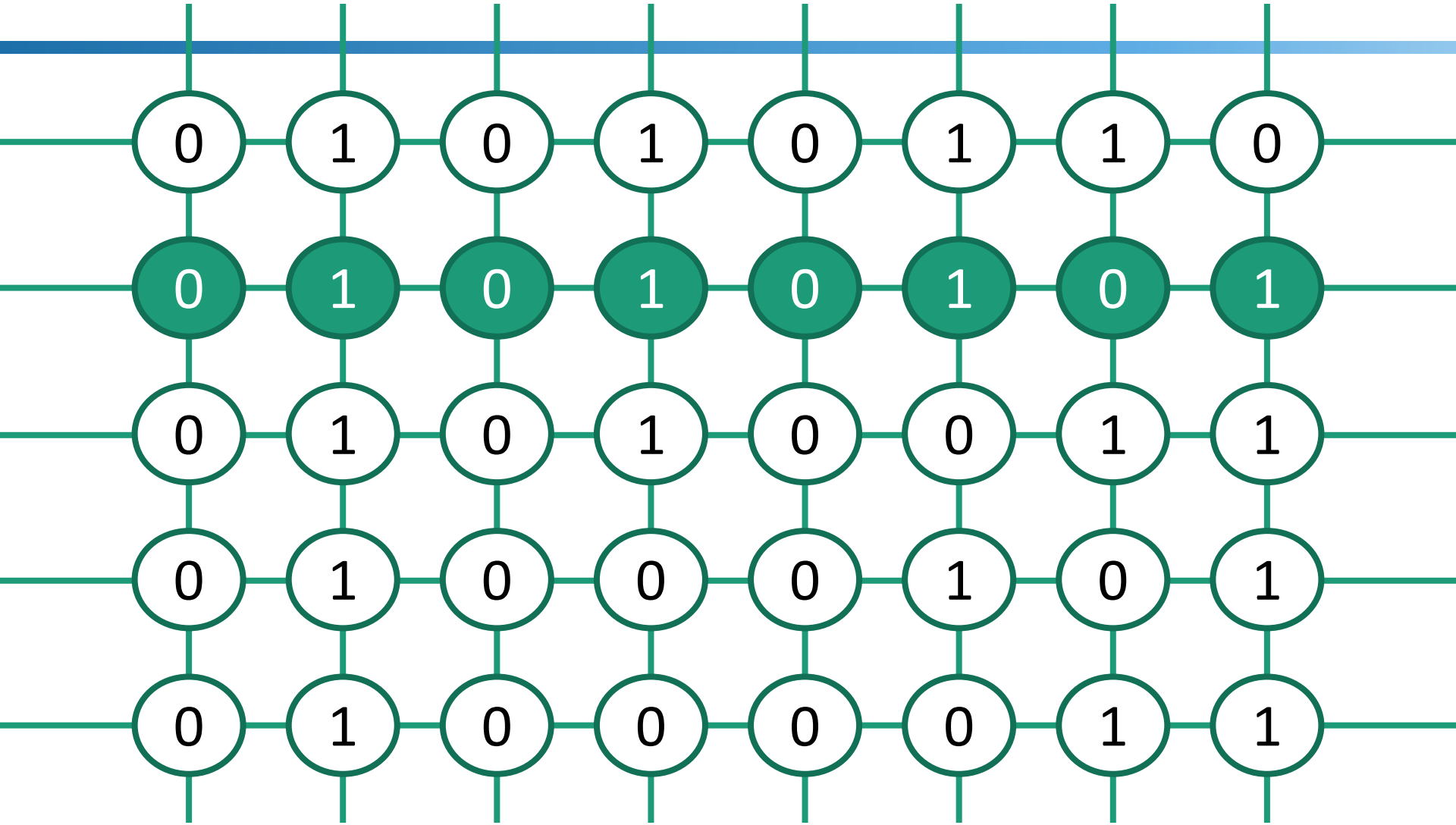
- Refresh every 64ms

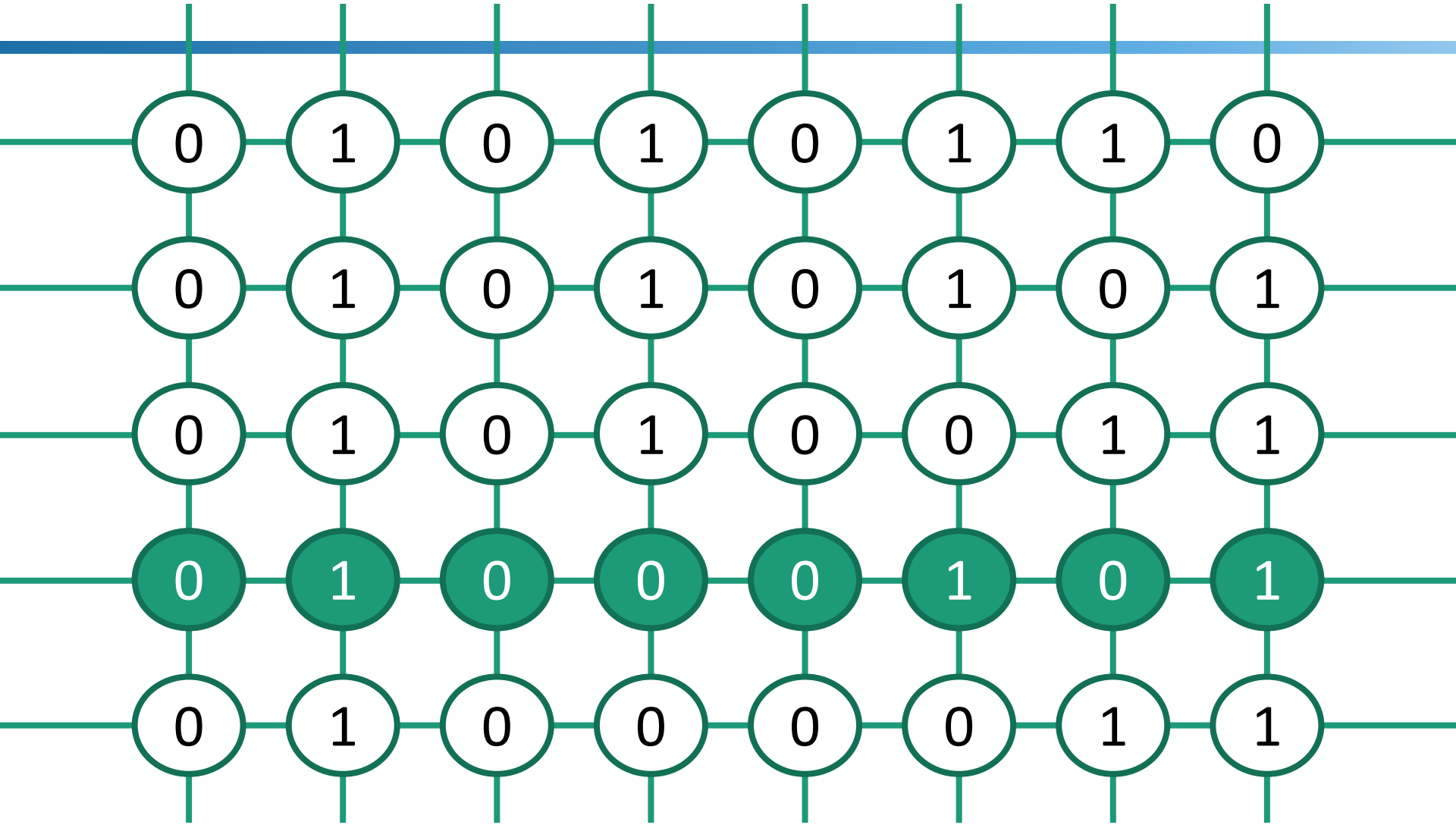
Activating neighboring cells increases the discharge rate

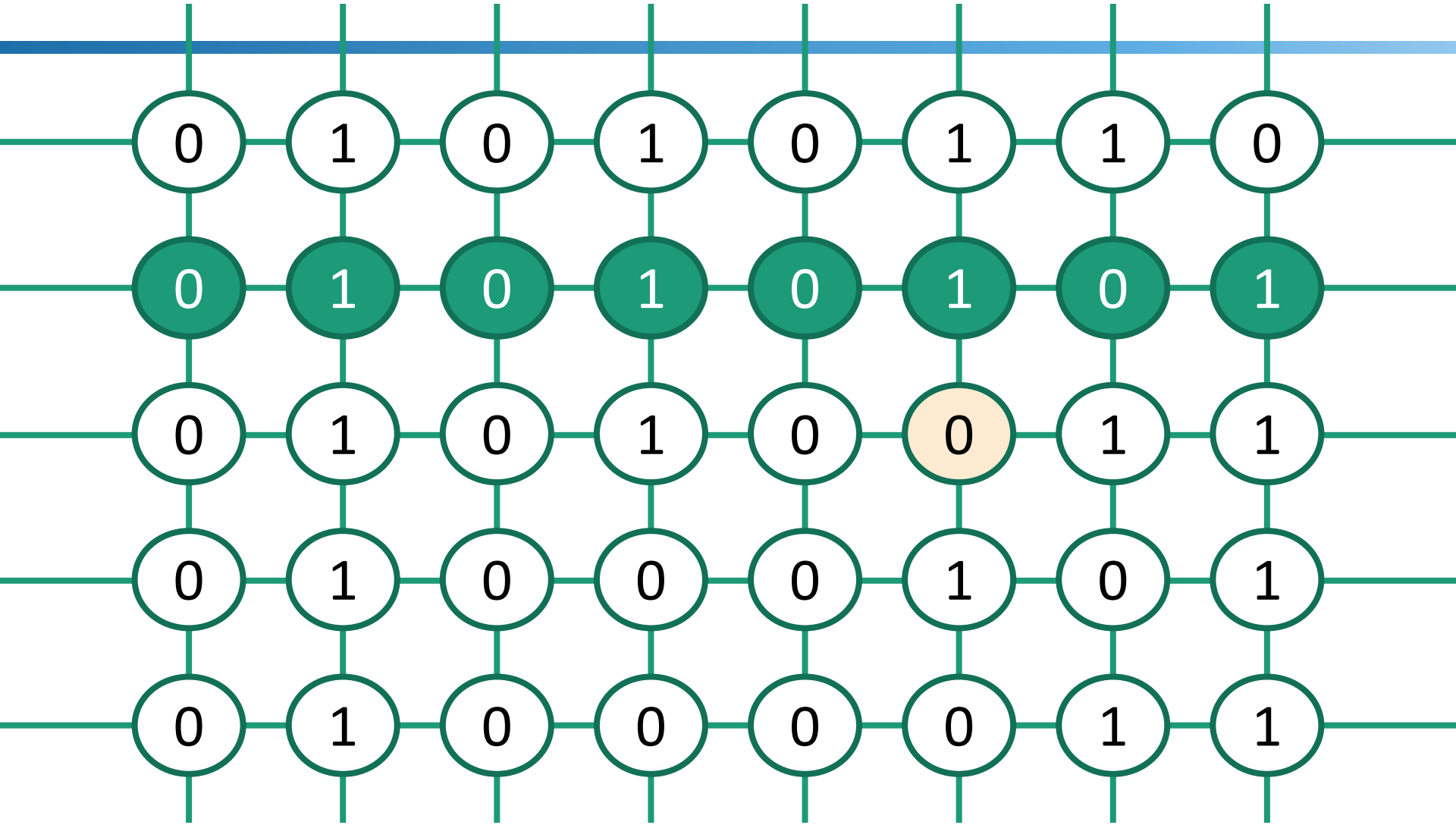
- Victim cell is charged to represent 1
- Neighboring cells are accessed frequently
- Victim cell leaks charge below a certain threshold
- When read, victim cell is interpreted 0

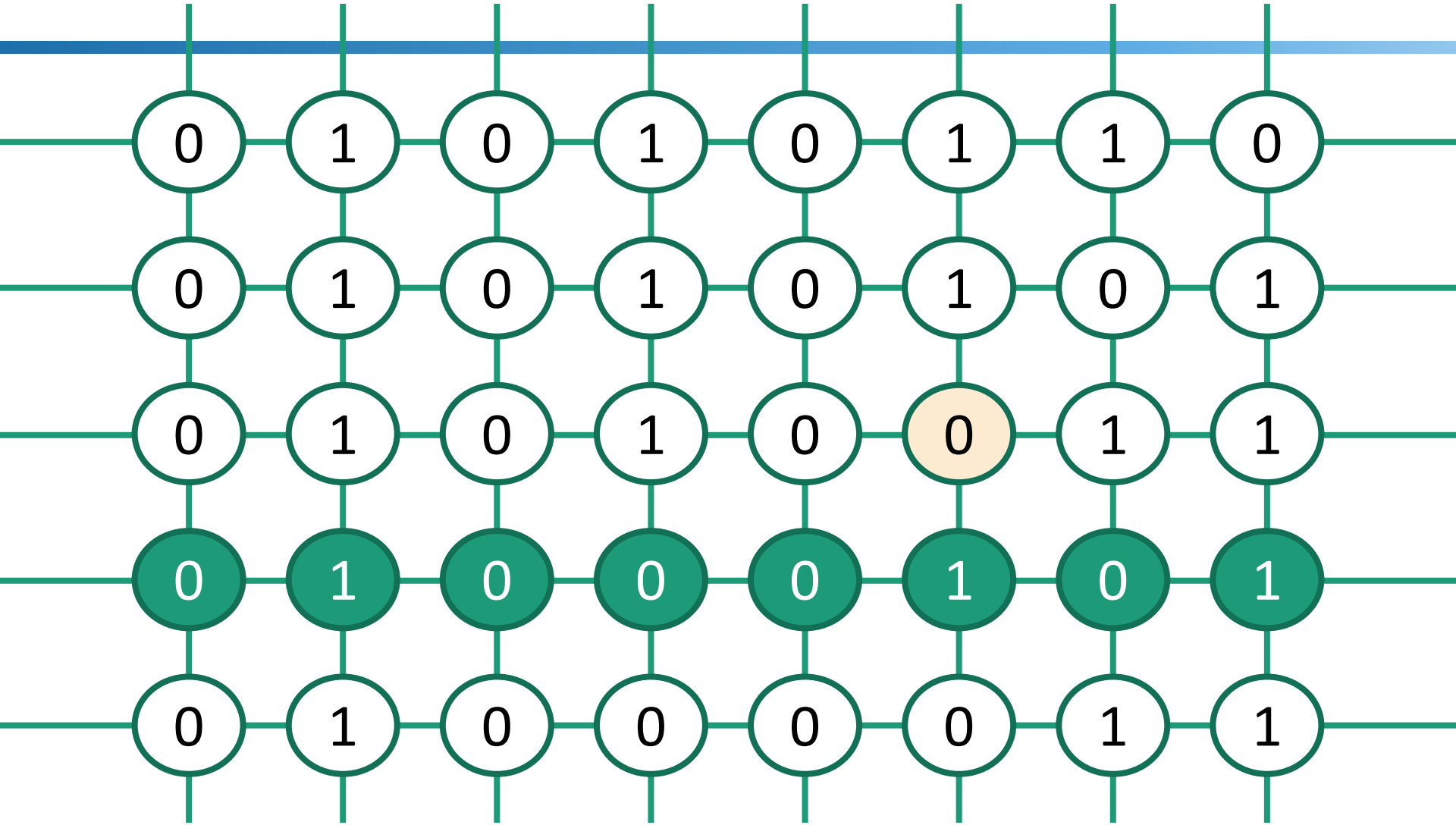
FROM: van der Veen et al., "GuardION: Practical Mitigation of DMA-based Rowhammer Attacks on ARM"

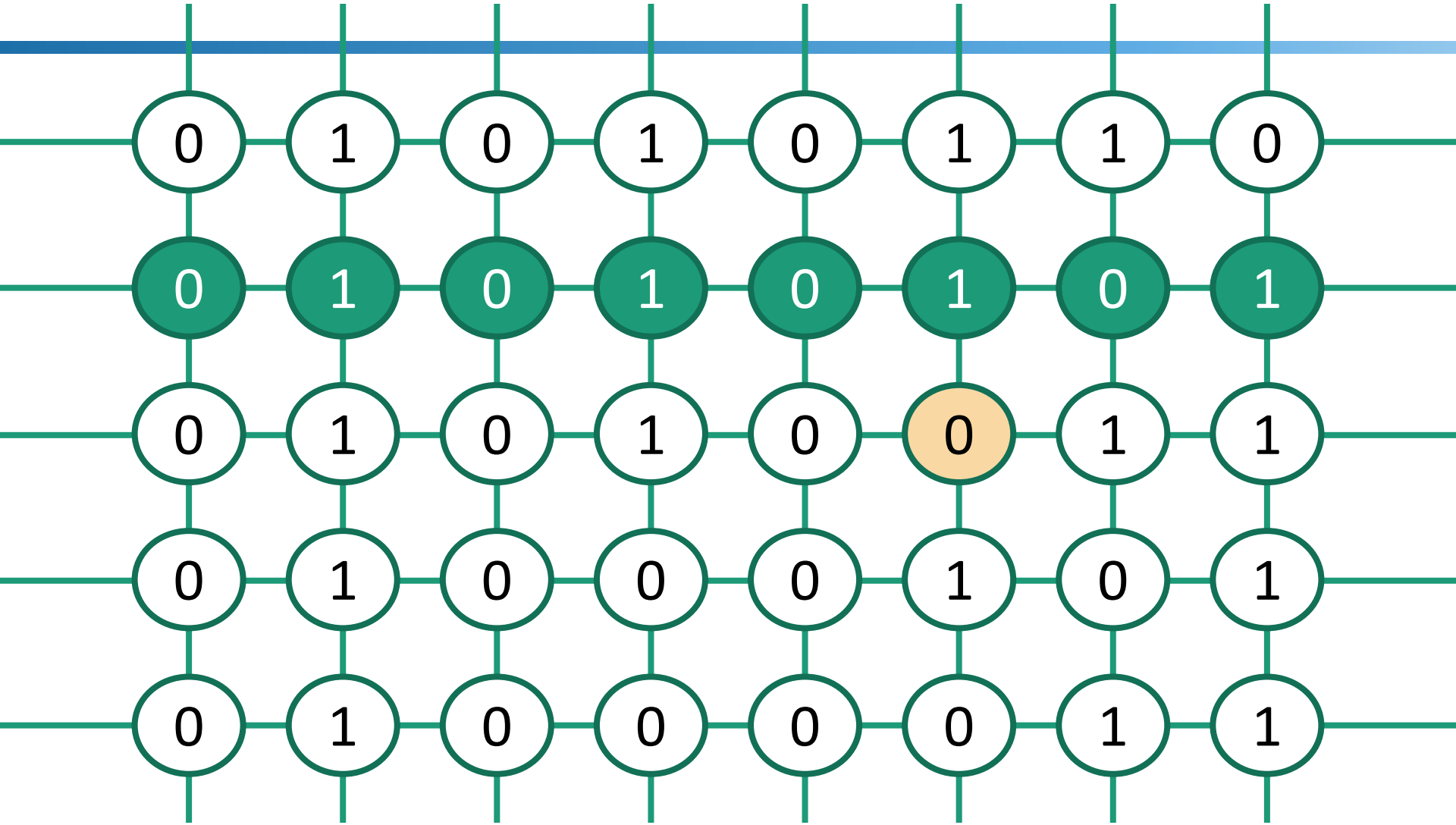

0 1 0 1 0 1 1 0
0 1 0 1 0 1 0 1
0 1 0 1 0 0 1 1
0 1 0 0 0 1 0 1
0 1 0 0 0 0 1 1

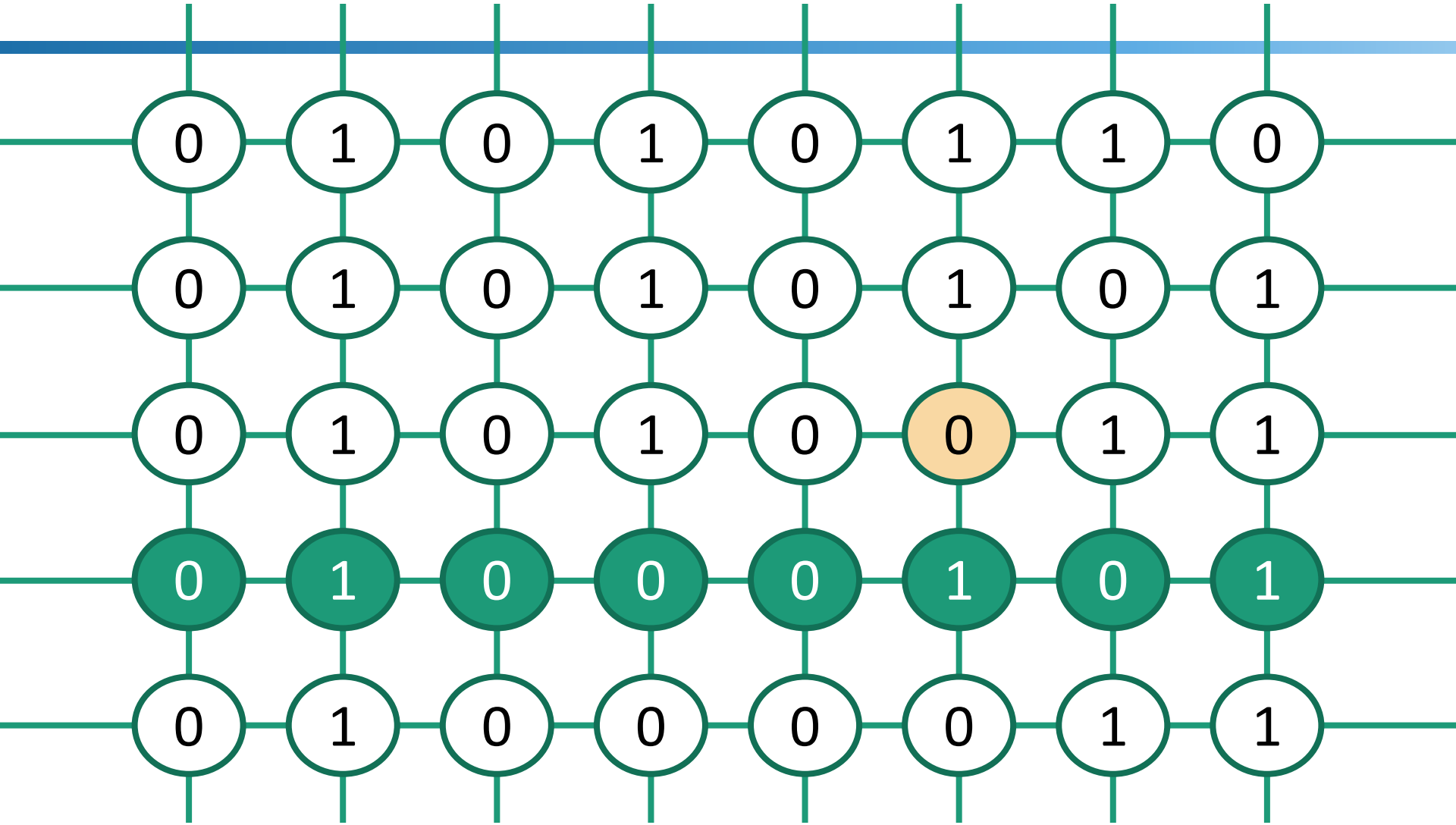

0 1 0 1 0 1 1 0
0 1 0 1 0 1 0 1
0 1 0 1 0 0 1 1
0 1 0 0 0 1 0 1
0 1 0 0 0 0 1 1

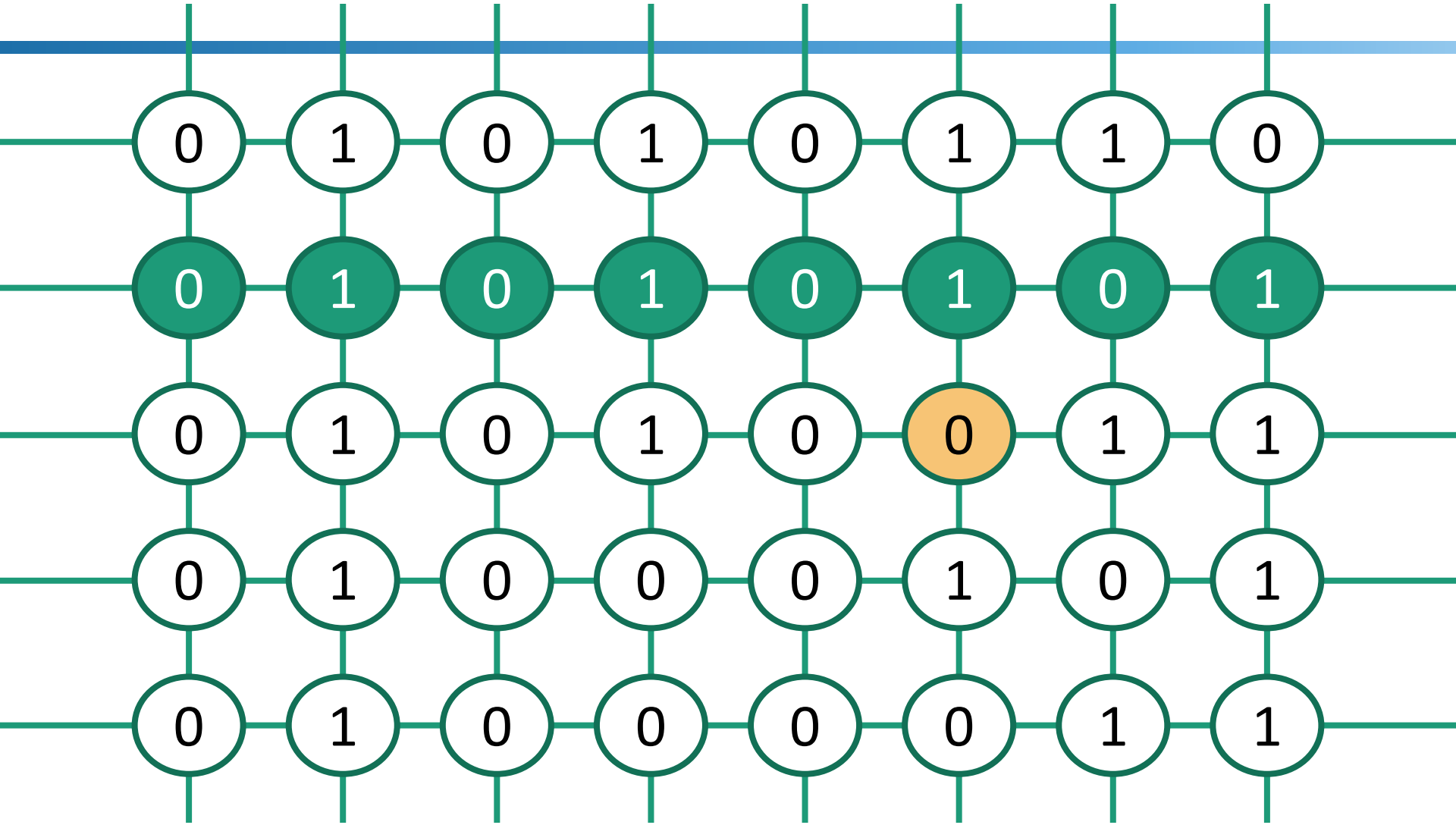

0 1 0 1 0 1 1 0
0 1 0 1 0 1 0 1
0 1 0 1 0 0 1 1
0 1 0 0 0 1 0 1
0 1 0 0 0 0 1 1

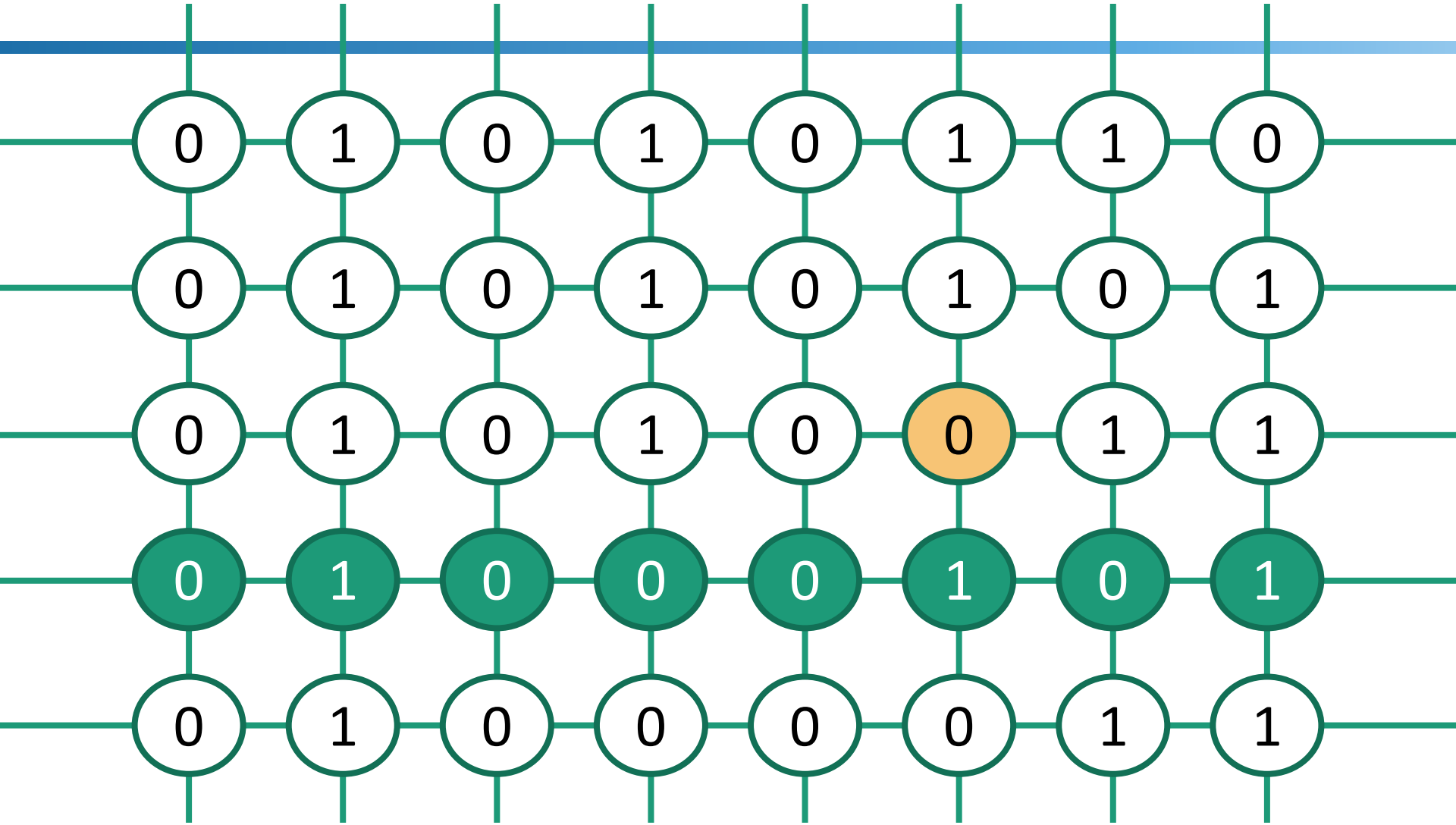

0 1 0 1 0 1 1 0
0 1 0 1 0 1 0 1
0 1 0 1 0 0 1 1
0 1 0 0 0 1 0 1
0 1 0 0 0 0 1 1

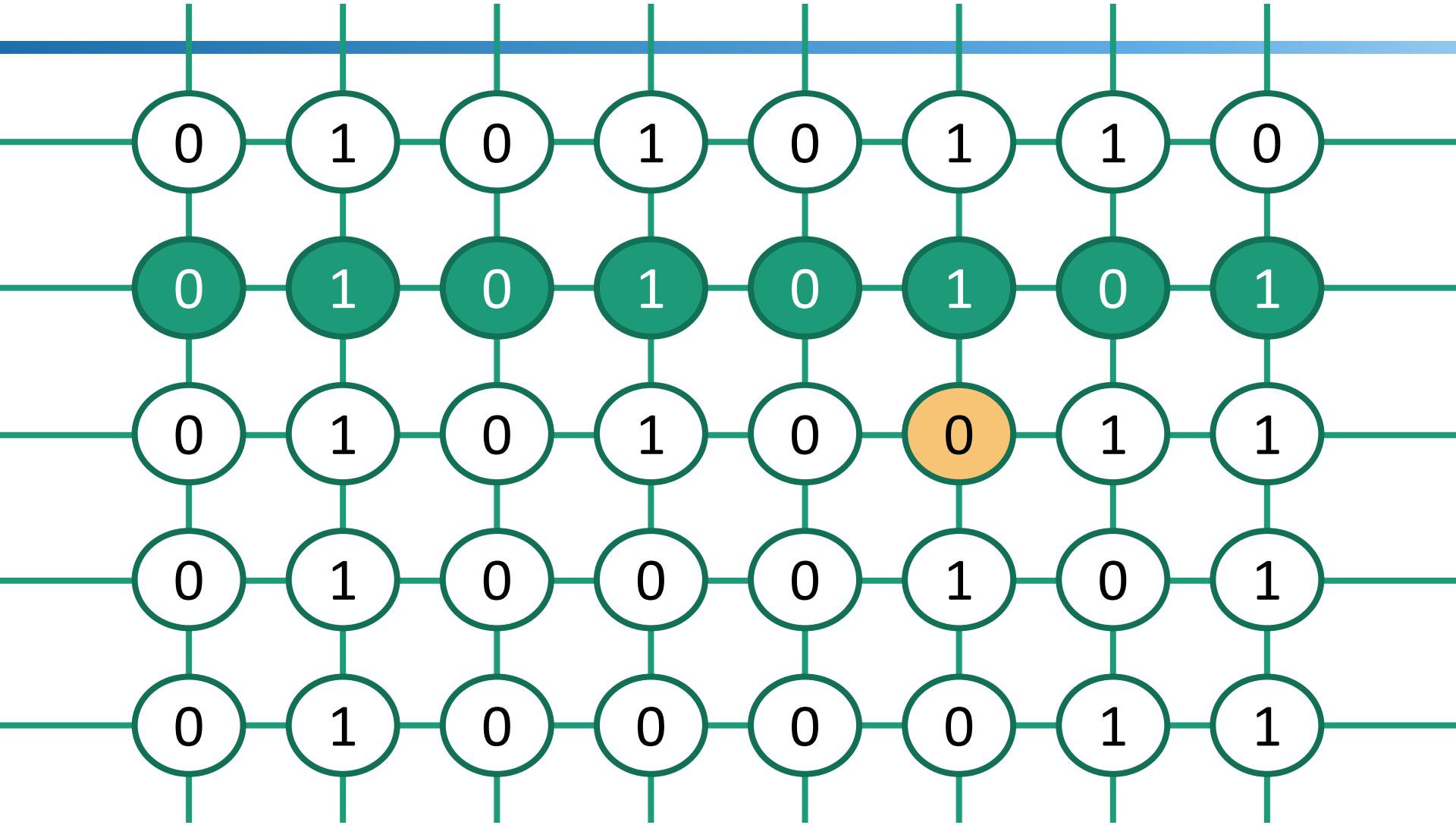

0 1 0 1 0 1 1 0
0 1 0 1 0 1 0 1
0 1 0 1 0 0 1 1
0 1 0 0 0 1 0 1
0 1 0 0 0 0 1 1

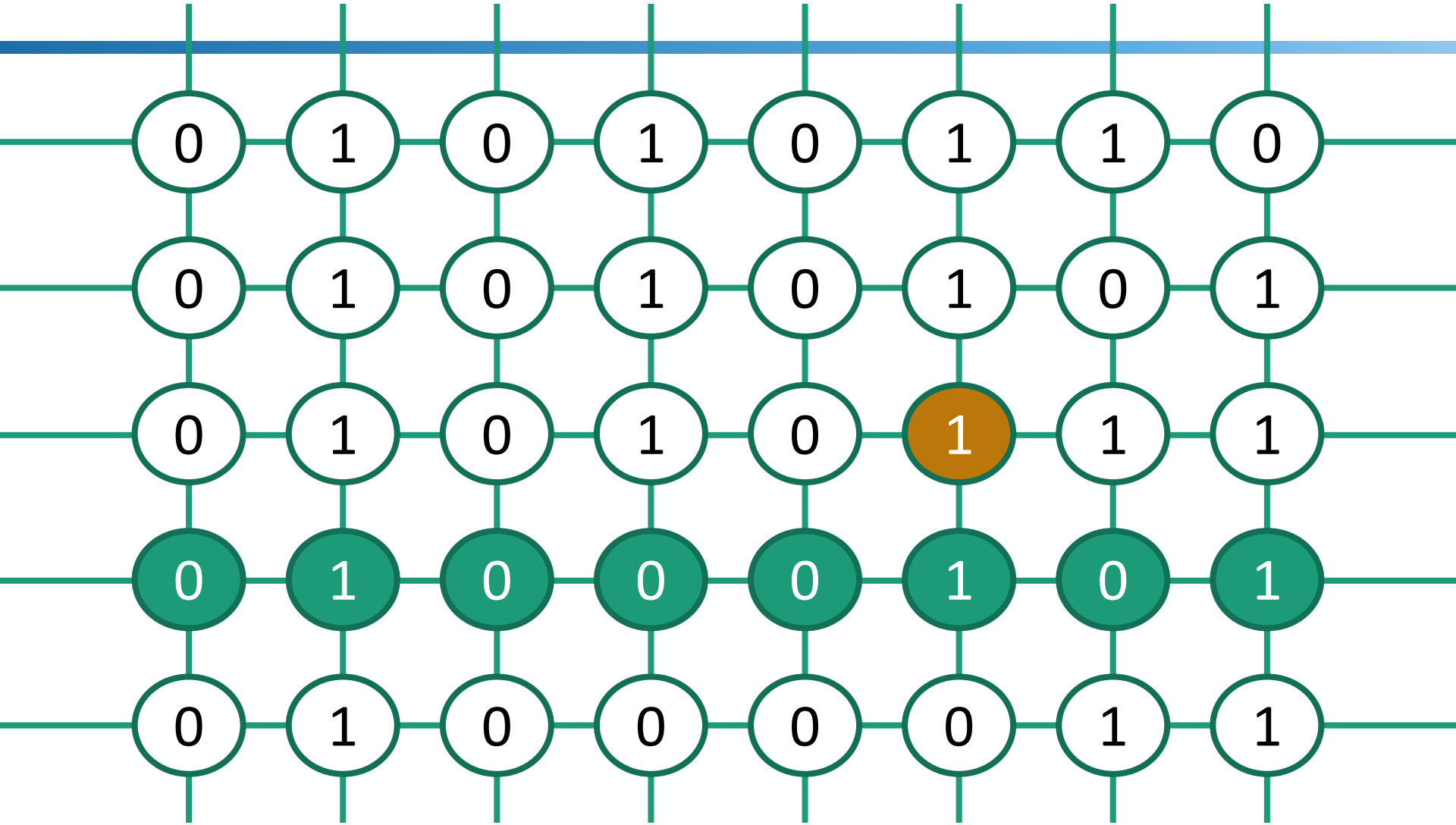

0 1 0 1 0 1 1 0
0 1 0 1 0 1 0 1
0 1 0 1 0 0 1 1
0 1 0 0 0 1 0 1
0 1 0 0 0 0 1 1


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


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


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


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


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

Rowhammer

Flip a bit in a **victim** row by reading from two **aggressor** rows

Not every bit may flip

Bit flips are **reproducible**

Challenges

1. Bypass the CPU cache
2. Get large contiguous chunks of memory

High-level Steps for Exploitation

Memory Templating

- Scan memory for useful bit flips

Land sensitive data

- Store a crucial data structure on a vulnerable page

Reproduce the bit flip

- Modify the data structure and get root access

Memory Templating

Un-cached memory access

- cache eviction with clflush

```
code1a:
  mov (X), %eax  // Read from address X
  mov (Y), %ebx  // Read from address Y
  clflush (X)  // Flush cache for address X
  clflush (Y)  // Flush cache for address Y
  jmp code1a
```

Find pairs of addresses that are on a different row, but on the same bank

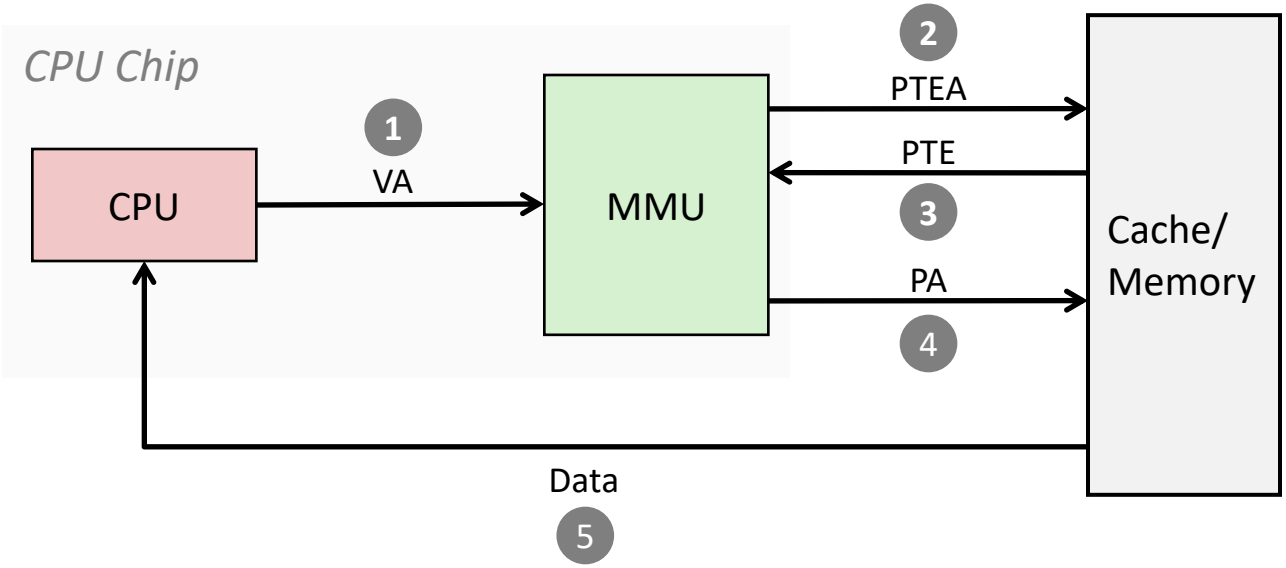
- Use knowledge of how the CPU's memory controller maps physical addresses to DRAM's
- Virt-to-phys page mappings /proc/self/pagemap
- Use 2MB huge pages (4K is smaller than a typical DRAM row)

Land Sensitive Data

Store a page table on a vulnerable page

Virtual Memory Basics

Address Translation Overview



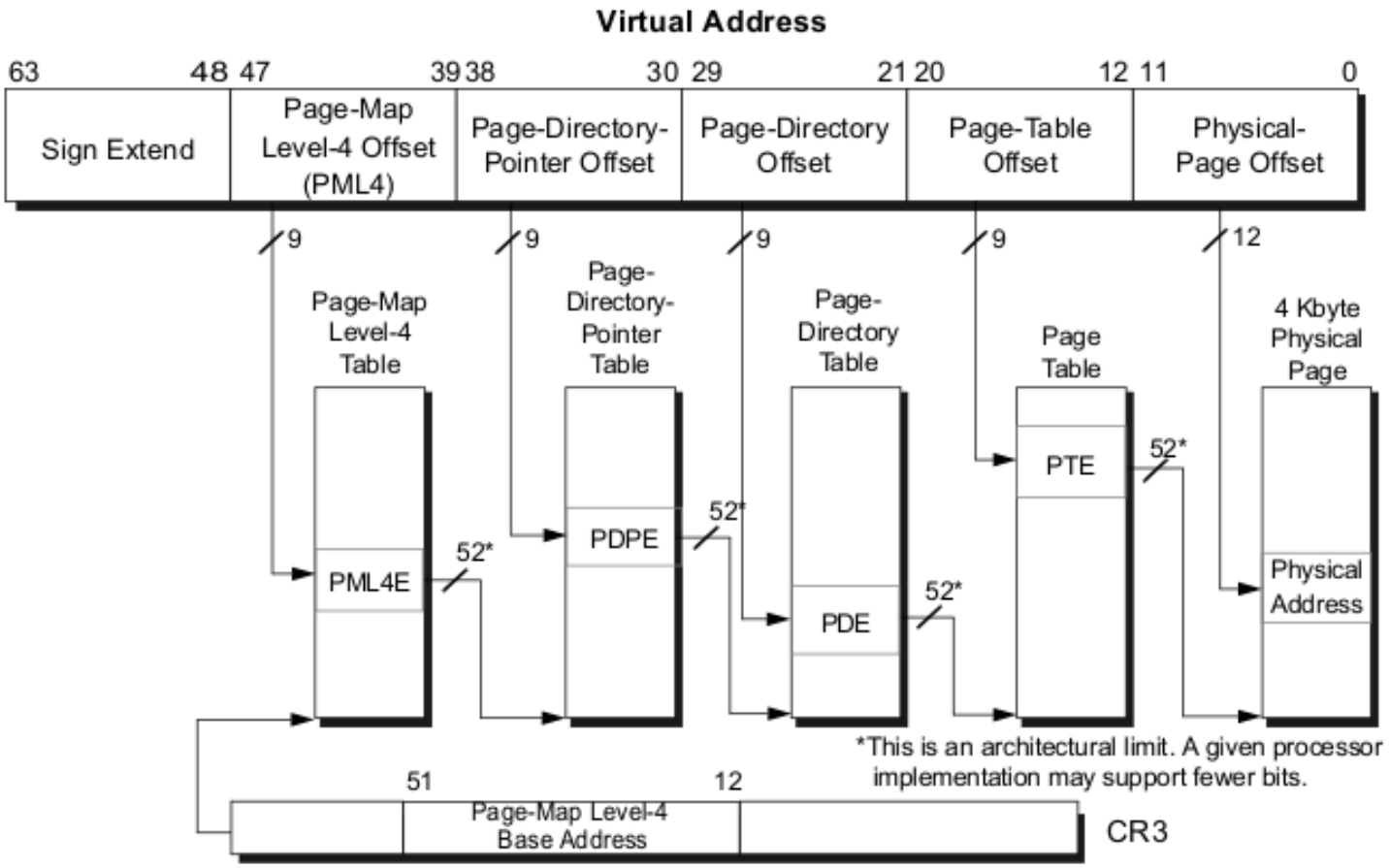
1) Processor sends virtual address to MMU

2-3) MMU fetches PTE from page table in memory

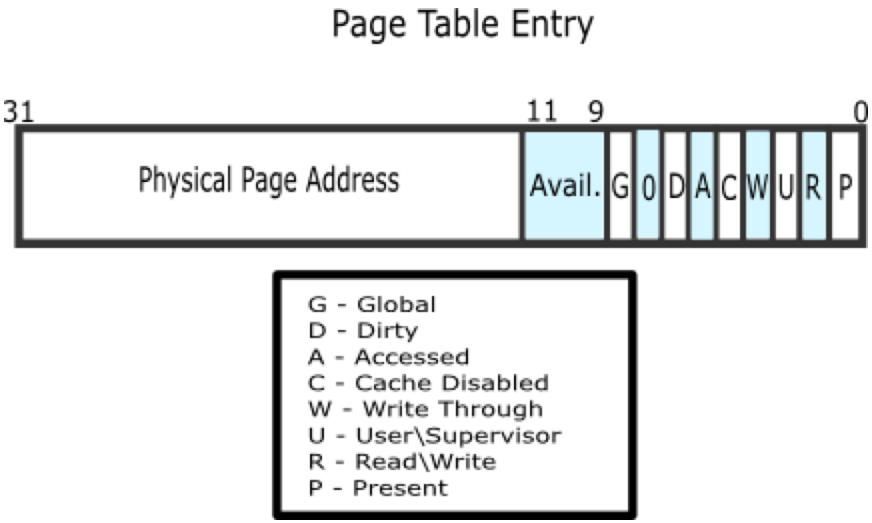
4) MMU sends physical address to cache/memory

5) Cache/memory sends data word to processor

VM in x86


Virtual Address
63
48 47
39 38
30 29
21 20
12 11
0
Sign Extend
Page-Map Level-4 Offset (PML4)
Page-Directory-Pointer Offset
Page-Directory Offset
Page-Table Offset
Physical-Page Offset
9
9
9
9
12
Page-Map Level-4 Table
Page-Directory-Pointer Table
Page-Directory Table
Page Table
4 Kbyte Physical Page
PML4E
PDPE
PDE
PTE
Physical Address
52*
52*
52*
52*
*This is an architectural limit. A given processor implementation may support fewer bits.
51
12
Page-Map Level-4 Base Address
CR3

Page Table Entry



Land Sensitive Data

Store a page table on a vulnerable page

To:

- Map a kernel page containing privileged data to user space
  - By creating a new mapping to its physical page
- Make a kernel page writable from user space

Hard problem: need to massage memory to look exactly right

- Potentially no access to pagemap (virtual – physical address mapping)

Massaging Memory Tricks

Allocate all physical memory

De-allocate memory corresponding to vulnerable bits

Cause allocation of new page table entry

Requires knowledge of the allocator

Reproduce the Bit Flip

Demonstrated to be able to

- Compromise OpenSSH PK authentication on VM co-location scenario
  - Two VMs (attacker and victim) running on the same host
- apt-get compromise by GPG signature forgery

No need for a software bug

Cache side channel attacks: CPU Design as a security problem

FROM: Anders Fogh, Protect Software

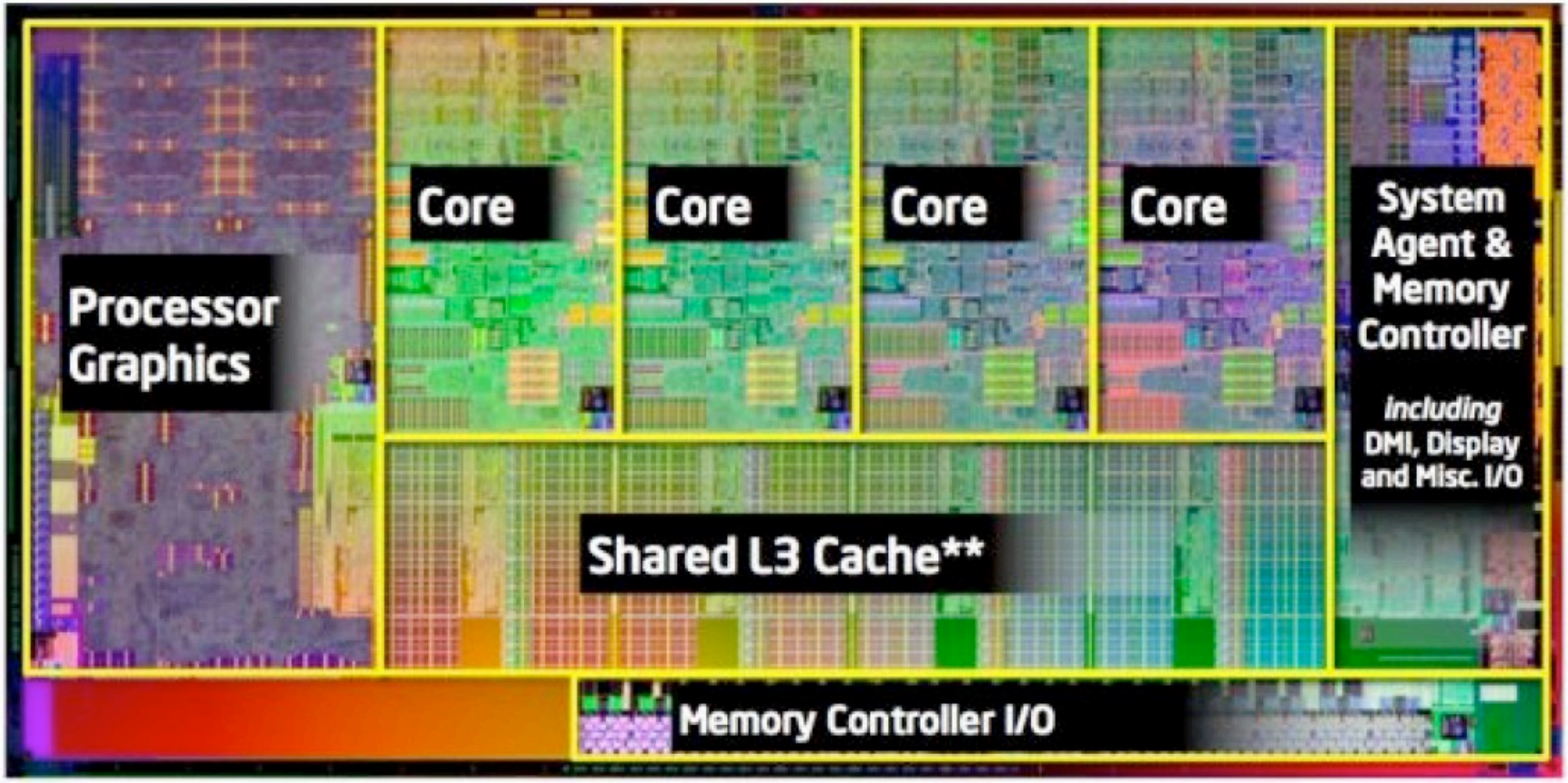
Cache Side Channel Attacks

Cache side channel attacks are attacks enabled by the micro architectural design of the CPU

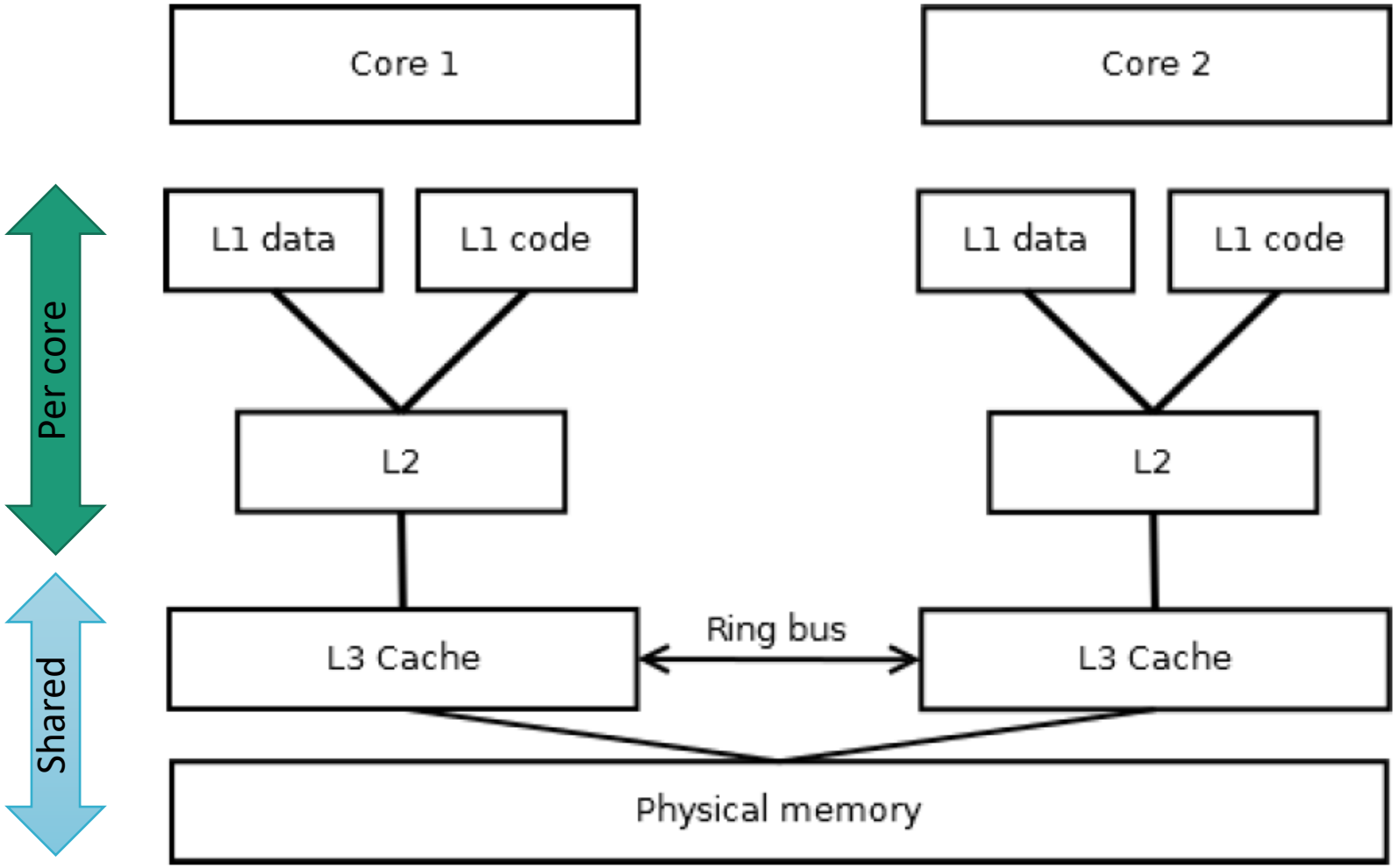
Because these side channels are part of hardware design they are notoriously difficult to defeat

Probably most important side channel because of bandwidth, size and central position in the architecture

Modern Processor



Cache Hierarchy



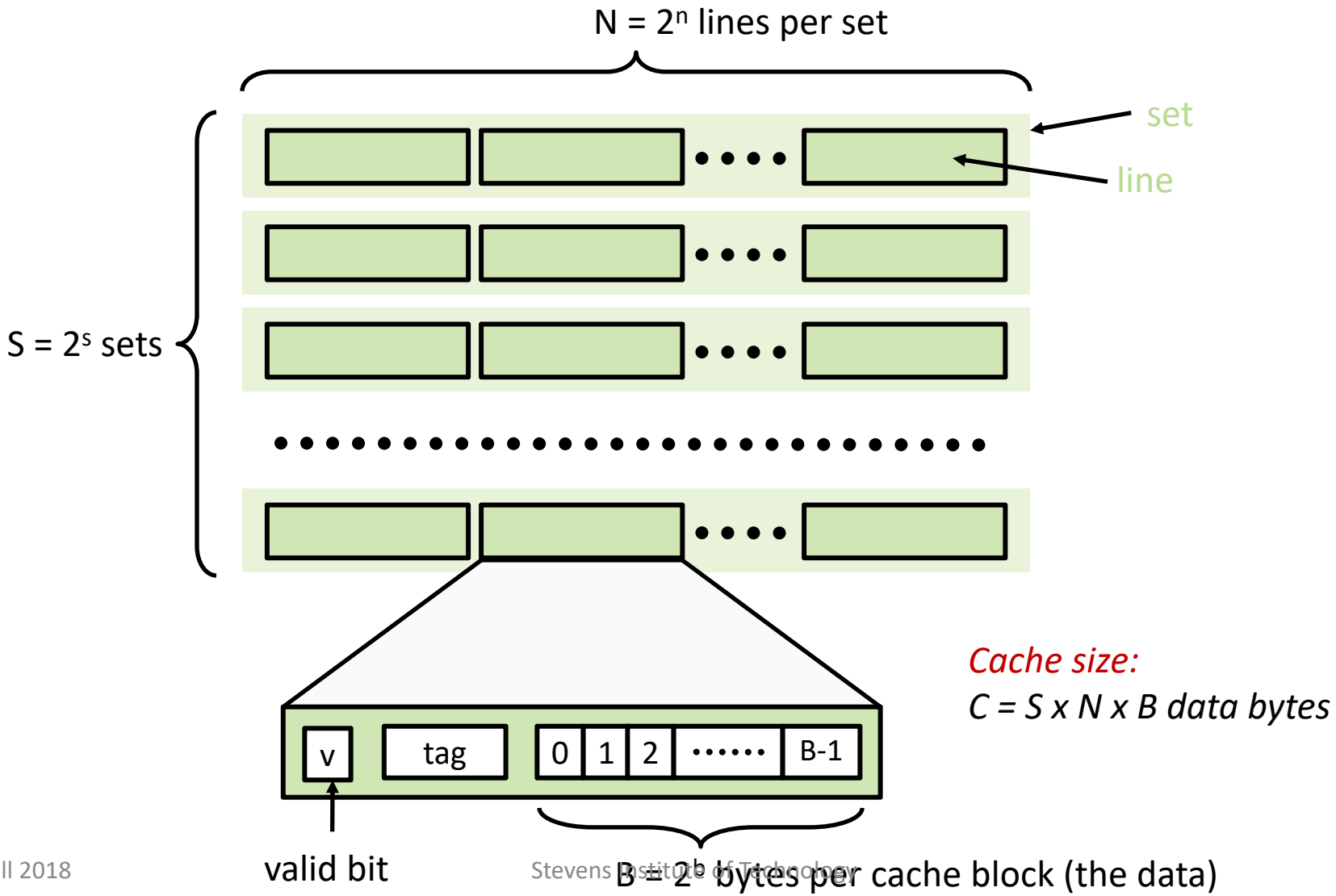
Cache Features

Almost any memory read/write is placed in the cache: The cache is a mirror image of memory activity on the computer.

L3 is shared globally across all users and privilege levels

Inclusive cache hierarchy: If we remove memory from L3, we remove it from all caches: We can manipulate the cache!

General Cache Organization

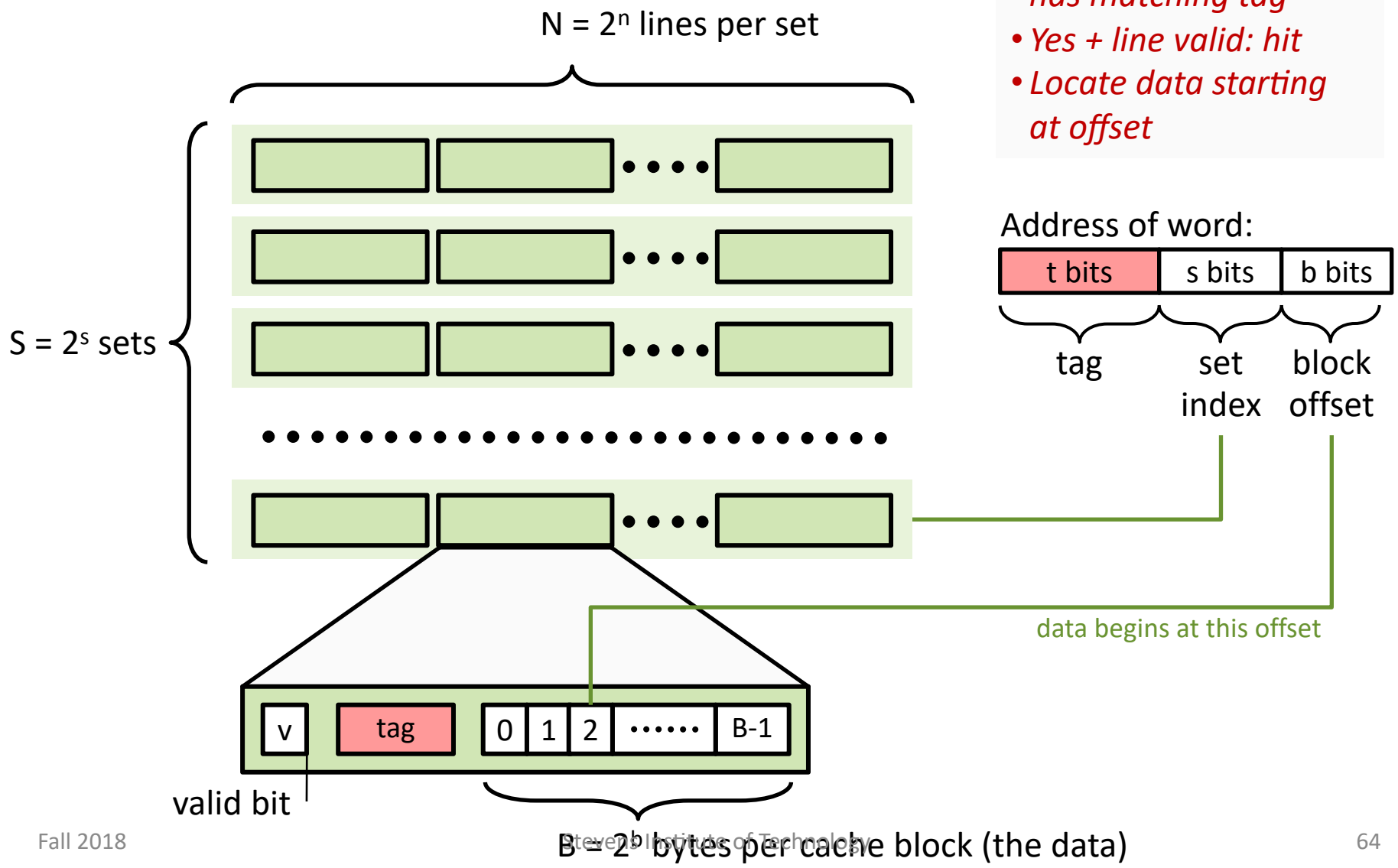


Cache size:

C = S x N x B data bytes

Cache Read

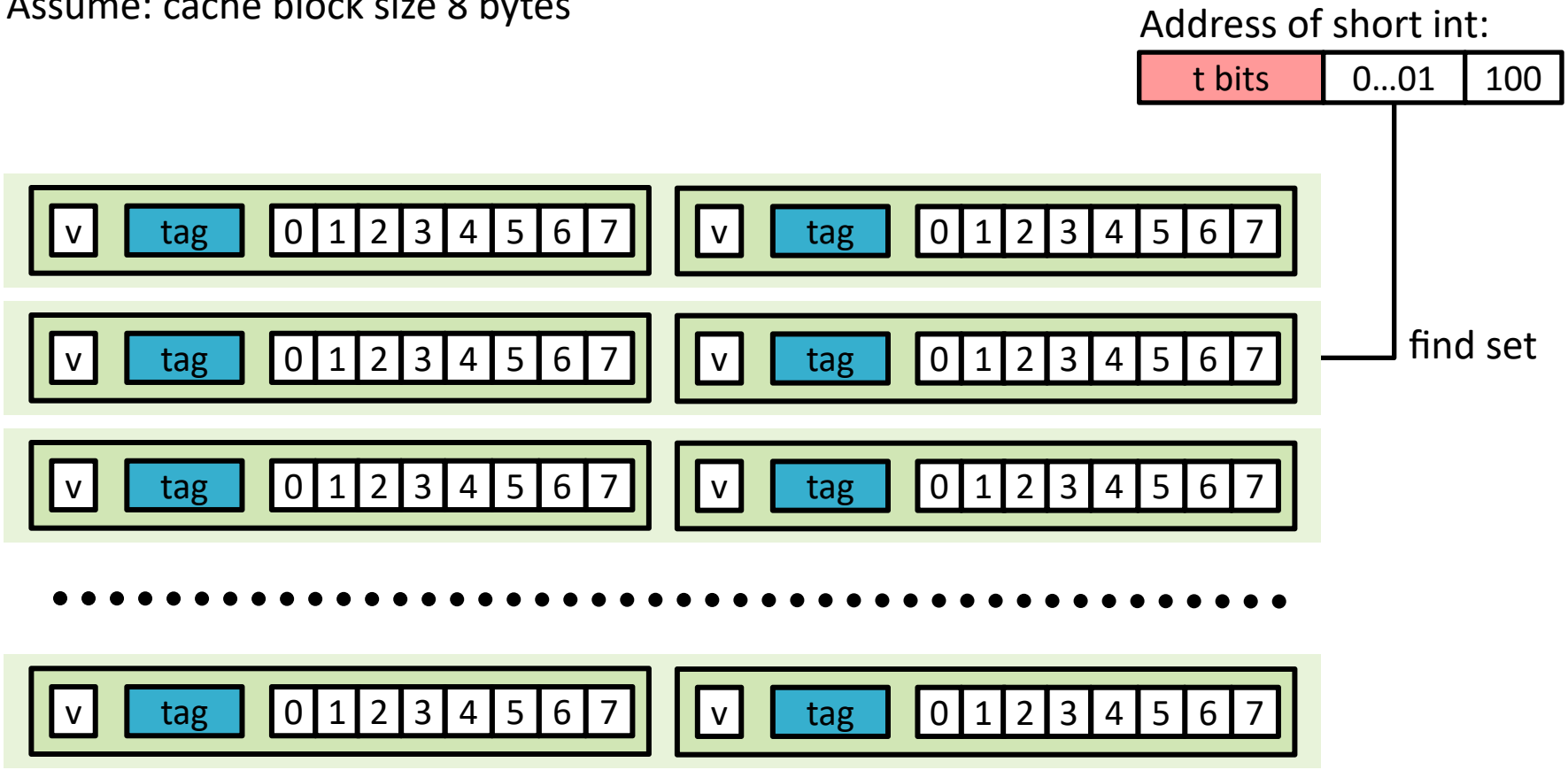
- *Locate set*
- *Check if any line in set has matching tag*
- *Yes + line valid: hit*
- *Locate data starting at offset*



N-way Set Associative Cache (Here: N = 2)

N = 2: Two lines per set

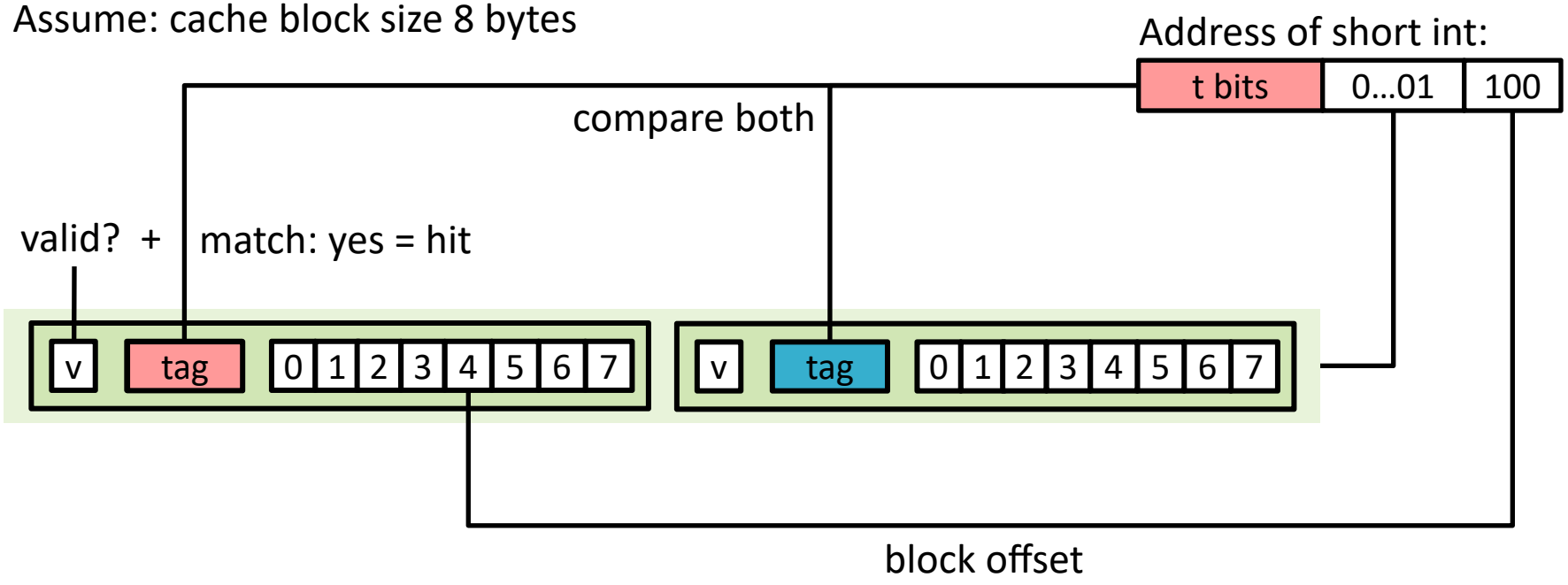
Assume: cache block size 8 bytes



N-way Set Associative Cache (Here: N = 2)

N = 2: Two lines per set

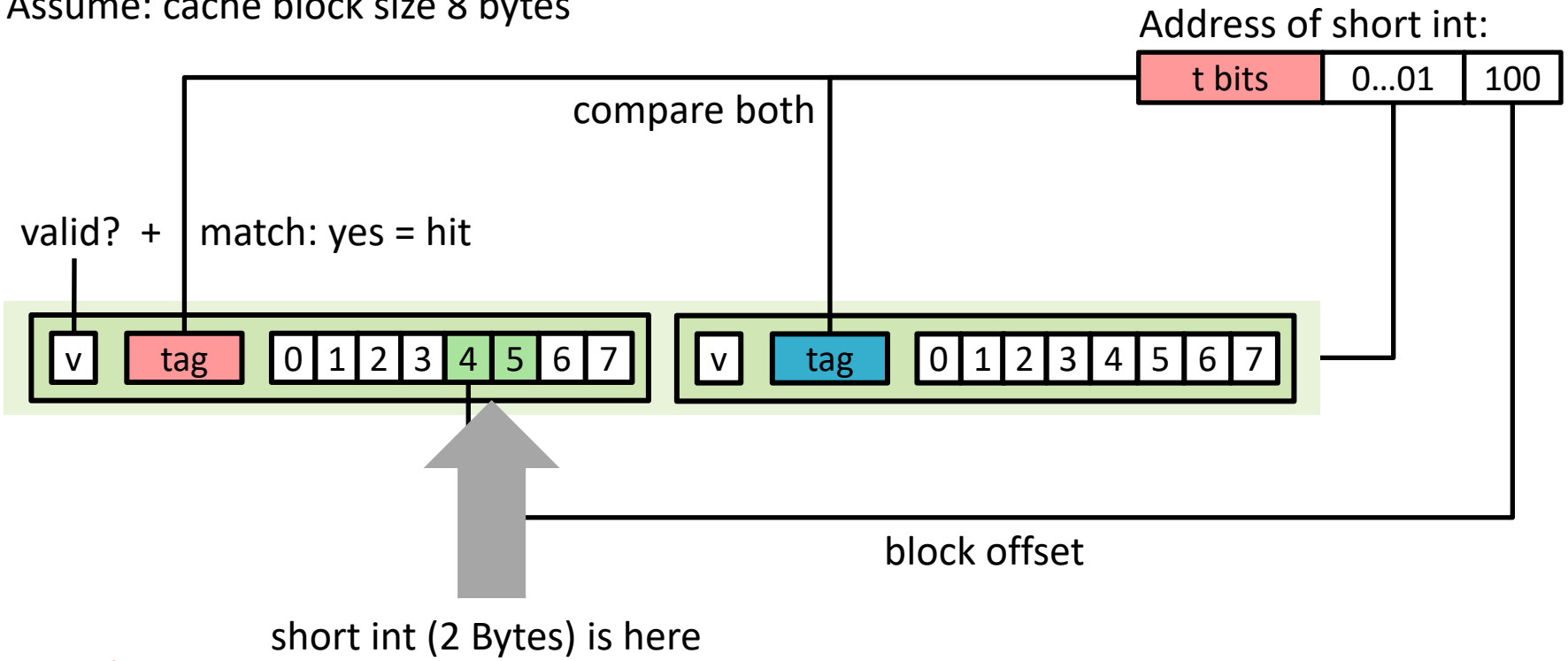
Assume: cache block size 8 bytes



N-way Set Associative Cache (Here: N = 2)

N = 2: Two lines per set

Assume: cache block size 8 bytes



No match:

- One line in set is selected for eviction and replacement
- Replacement policies: random, least recently used (LRU), …

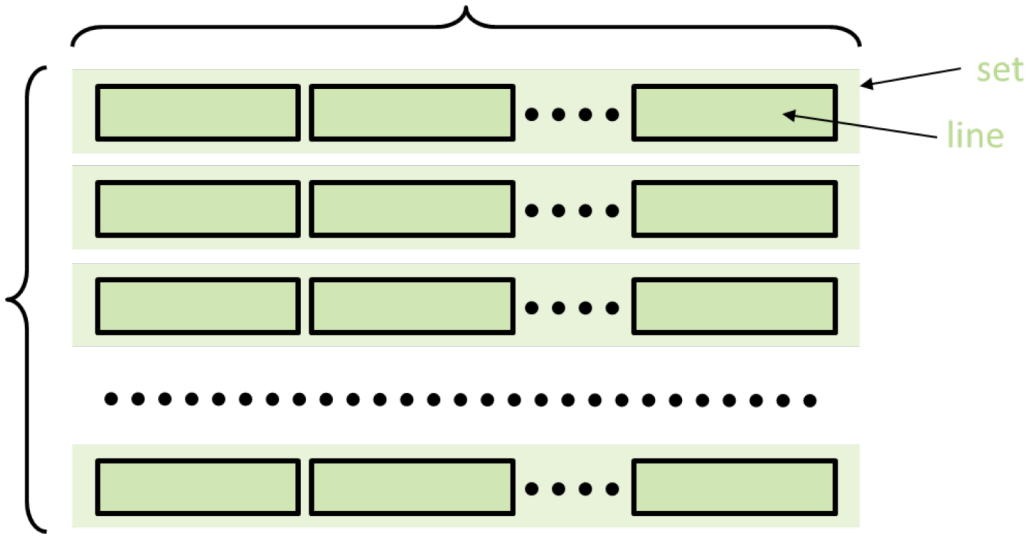
https://en.wikipedia.org/wiki/Cache_replacement_policies

Typical L3 Cache

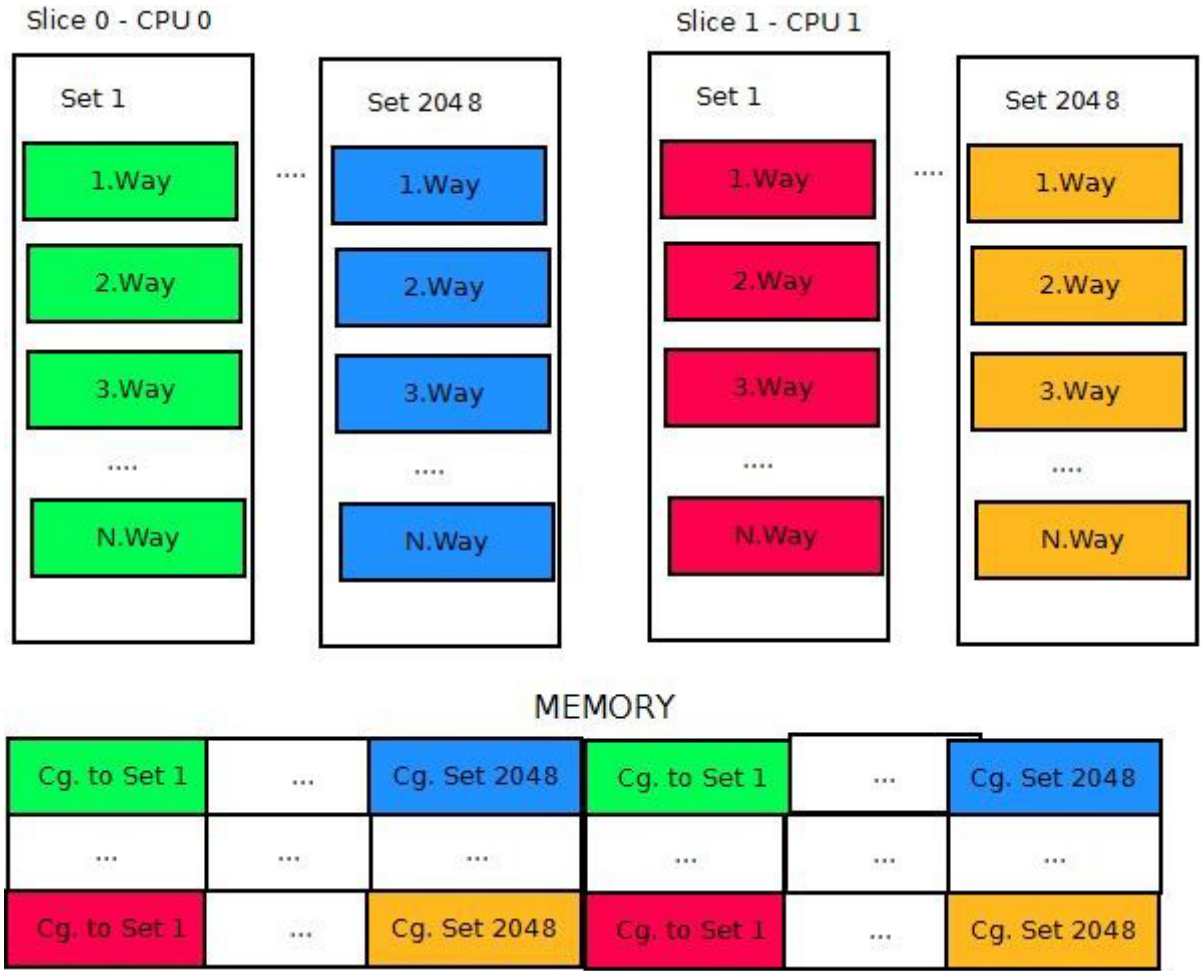
Cache lines are typically 64 bytes long

N-Way set associative cache

- 2048 cache sets per core
- Each set can store N (typically 12-20) cache lines, depending on total cache size



Typical L3 Cache



Common Aspects of Atacks

We can determine if something is cached

- Speed difference: 80 CLK vs. 200 CLK

We can manipulate the cache

- Evict: Access memory until a given address is no longer cached
- Flush: Remove a given address using clflush instruction
- Prime: Place known addresses in the cache

Big 3 Cache Side Channel Attacks

Evict + Time

Prime + Probe

Flush + Reload

They all work this way: Manipulate cache to known state, „wait“ for victim activity and examine what has changed

Evict + Time

Step 1. Execute a function to prime cache

Step 2. Time the function

Step 3. Evict a cache set

Step 4. Time the function

If Step 2 was faster than step 4 the function probably used an address congruent to the cache set in step 3

Prime + Probe

Step 1. Prime a cache set to contain known attacker addresses

Step 2. Wait for victim activity

Step 3. Time accessing address from step 1.

If accessing memory in step 3 is slow (cache miss)

victim used memory congruent with cache set in

step 1

Flush + Reload

Step 1. Flush: Flush shared address from cache

Step 2. Wait for victim

Step 3. Reload: Time access for accessing the shared address:

If fast timing in 3 was fast it was placed in cache by victim. If slow victim did not use the address

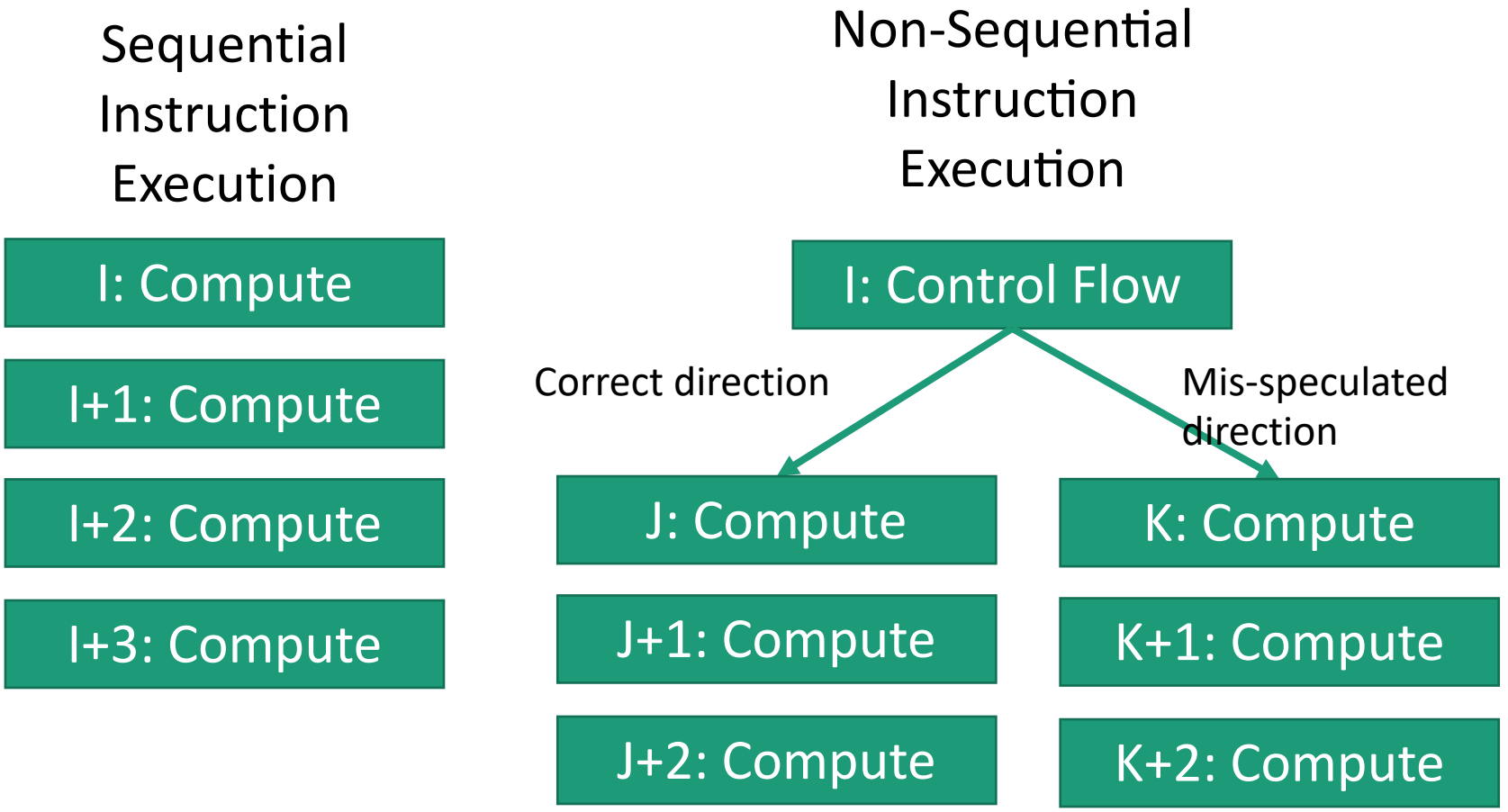
Example: Code Vulnerable to a Side Channel

```
WCHAR gdk_keysym_to_unicode(gkeysym Input)
{
    if (IsUpper(Input)) {
        return gkeysym2unicode_UpperCase(Input);
    } else {
        return gkeysym2unicode_LowerCase(Input);
    }
}
```

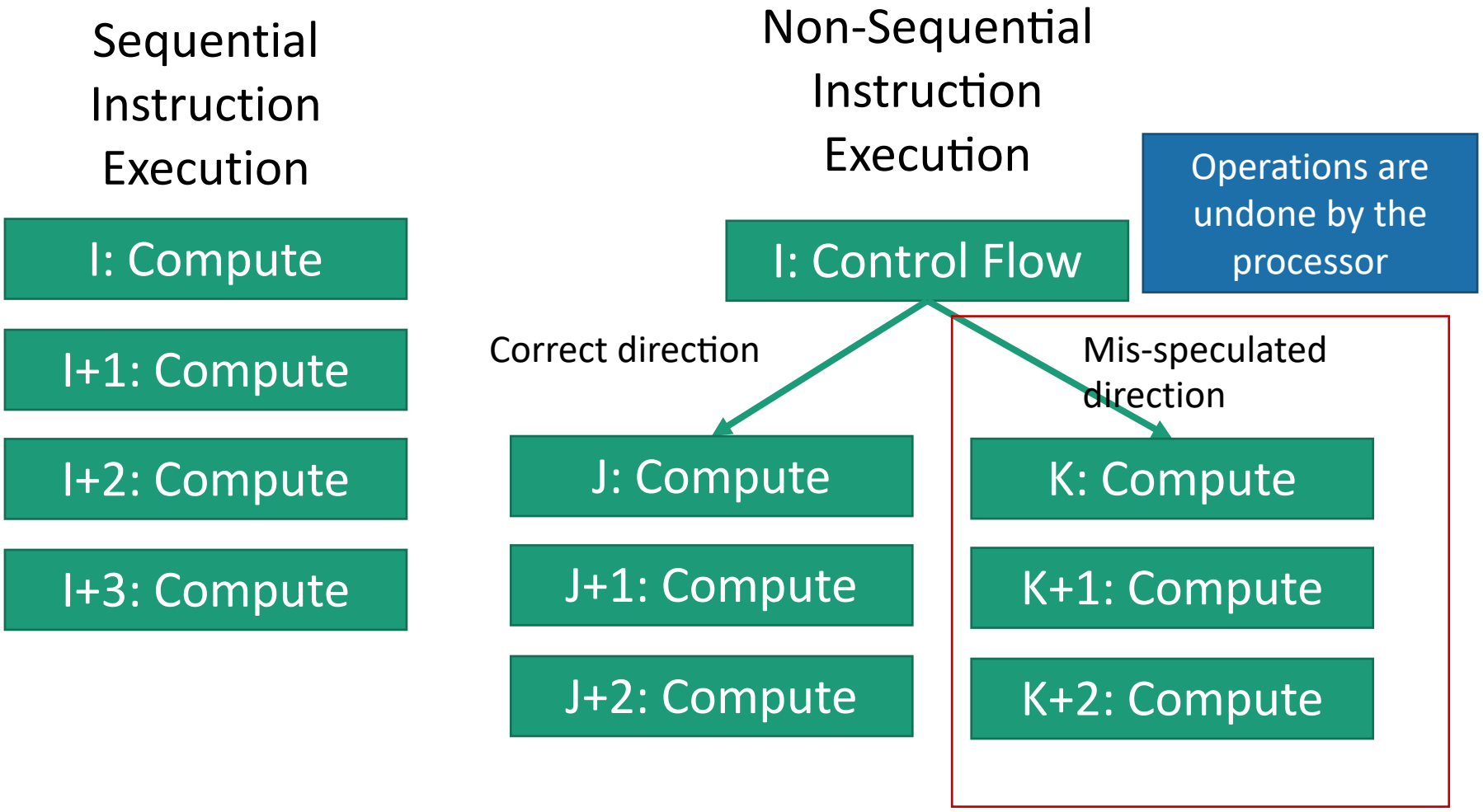
Spectre and Meltdown

FROM: Adam Belay, Srini Devadas, and Joel Emer

Control Speculation



Control Speculation



Meltdown

Problem: Attacker can influence speculative control flow

Bug:

- Speculative execution not subject to page permission checks
- Data remain in cache

Attack: User code can read kernel data (secret)

Three steps:

- Setup: flush the cache
- Force speculation that depends on secret
- Measure cache timings

Meltdown example

Setup:
clflush(timing_ptr[guess]);

Transmit:
timing_ptr[*kernel_addr];

Page Fault

May still read *kernel_addr (speculatively)

Receive:
mfence();
s = rdtsc(); *timing_ptr[guess];
e = rdtscp();
if (e - s < CACHE_MISS_THRESHOLD)
  printf("guess was right!\n");

Spectre

Problem: Attacker can influence speculative control flow

Bug:

- Speculative execution leads to out-of-bounds memory access
- Data remain in cache

Attack: User code can read data in the same process

- For example, through JS executing on the same host

Same steps

Spectre Example

Transmit - Bounds Check Bypass:

```
if (x < array1_size)
  array2[array1[x] * 256];
```

May execute for x >= array_size