CS 211: Intro to Computer Architecture

14.1: Direct-Mapped Caches (In Detail)

Minesh Patel

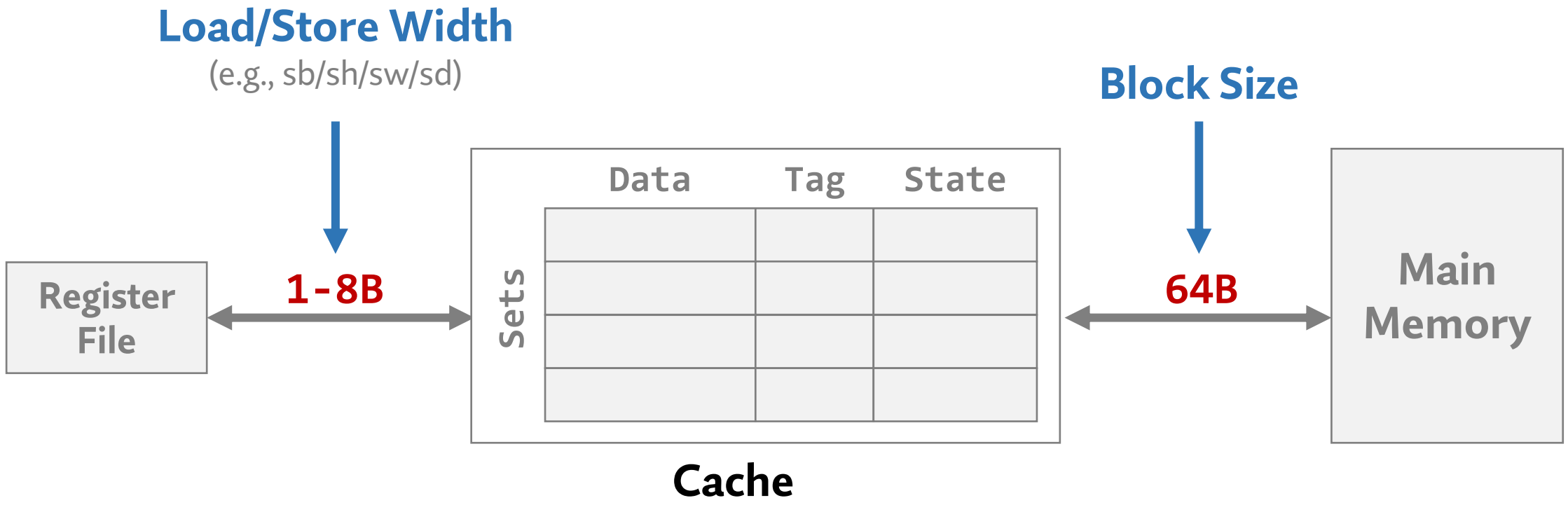
Spring 2025 – Tuesday 29 April

Announcements

- Ongoing
  - **WA10:** due **Friday (May 2) @ 11:59 pm**
  - **PA5:** due **Monday (May 5) @ 11:59 pm**
    - Pending: Autograder ☹
- Upcoming
  - **Final Exam:** May 14th
    - Pending: List of topics
    - Pending: Practice questions for this half of the semester

Recall: Cache Blocks

- Cache blocks are **bigger** than what we ask for
  - **Spatial locality:** assumes we will also want nearby memory
  - **Temporal locality**: holds the data as long as it can

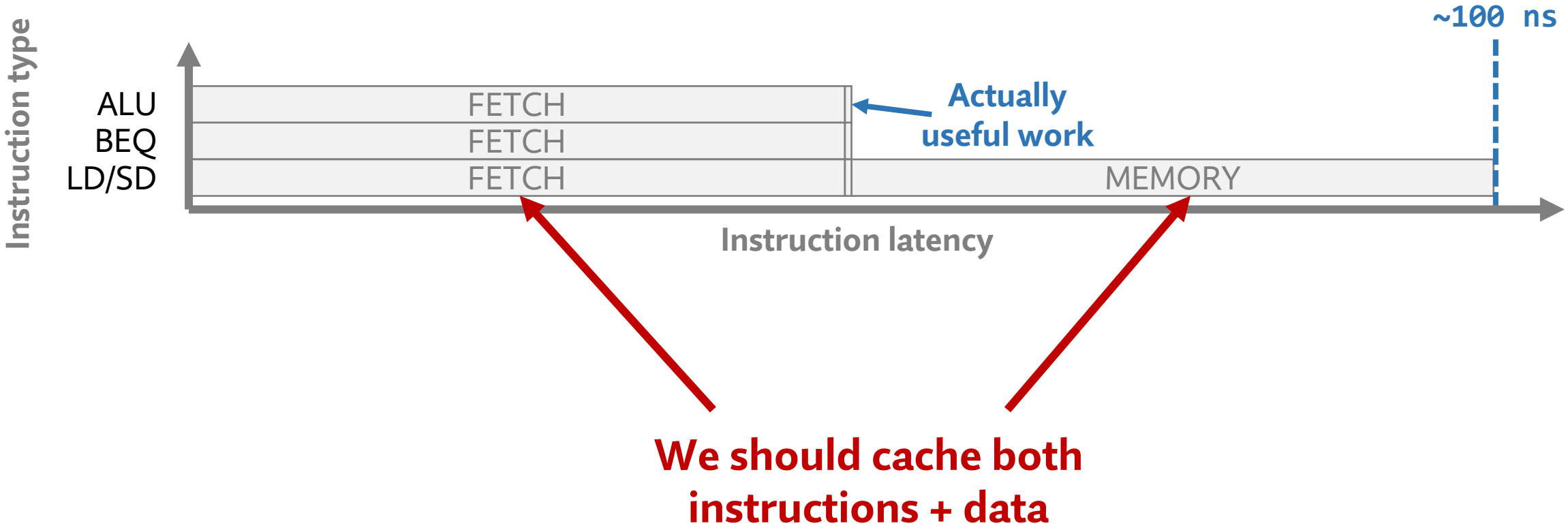


Agenda

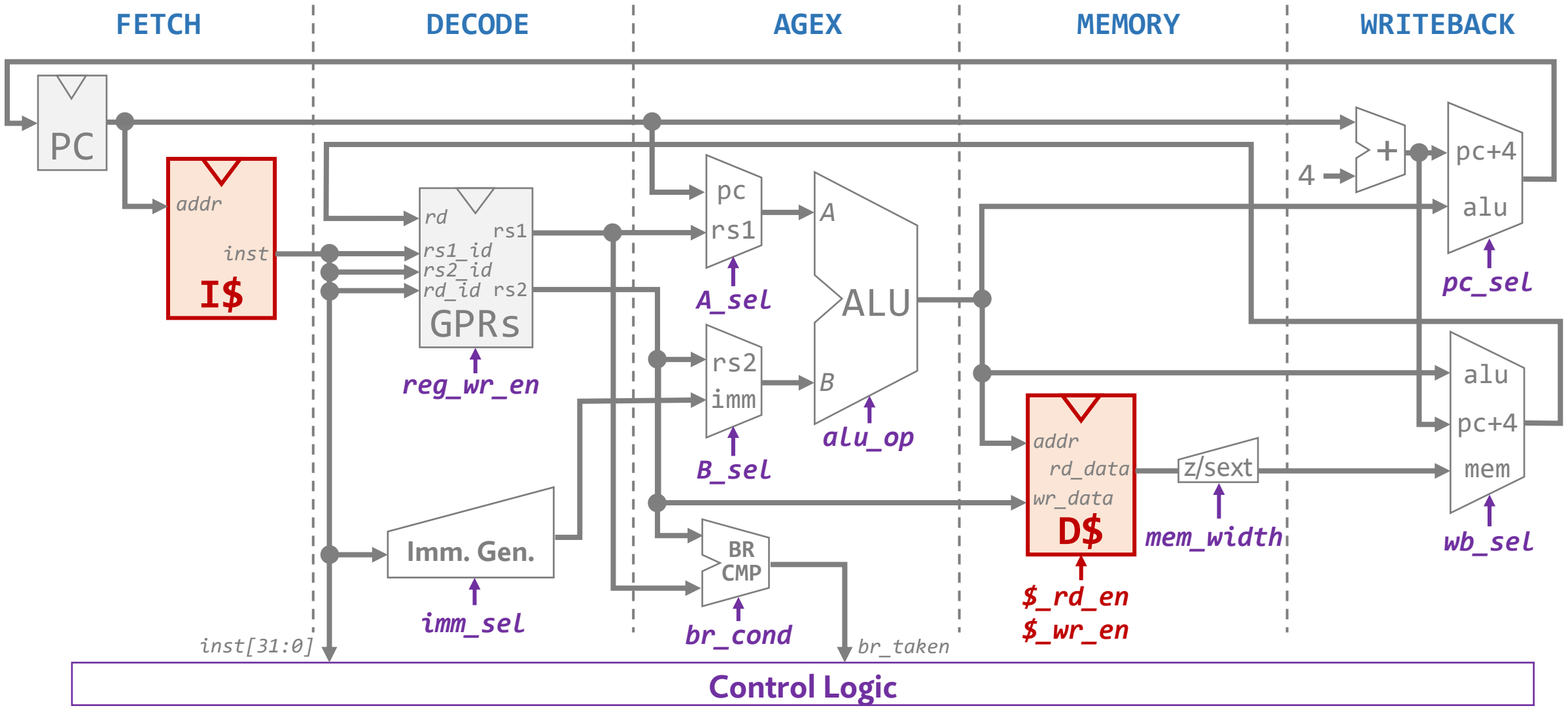
- **Instruction Cache and Data Cache**
  - Set Index Hashing
  - Simulating Cache Behavior
- Types of Cache Misses

Recall: Main Memory Kills Performance

- **Every instruction** needs fetching
- Every load/store **ALSO** needs memory access

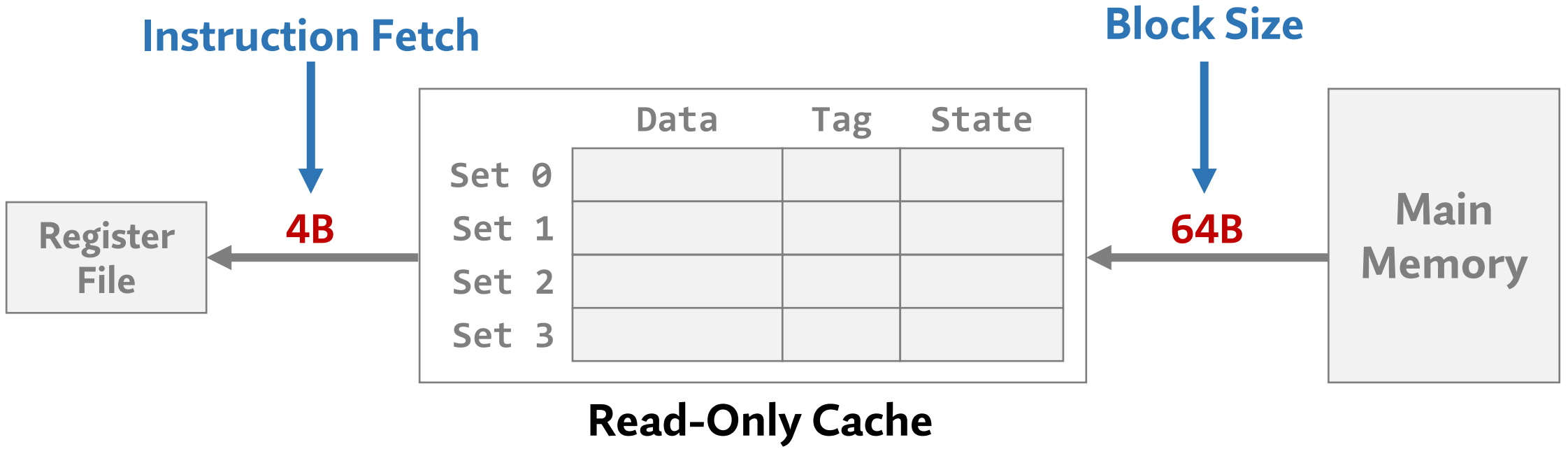


Instruction Cache (I$) and Data Cache (D$)



Instruction Cache

- Instructions have **very good** spatiotemporal locality
  - **Spatial:** We almost always want **PC + 4**
  - **Temporal:** All loops re-use instructions



Example: Instruction Cache Locality

- How much does the following code benefit from an **instruction cache**?
  - Assume 64B cache blocks

C Code

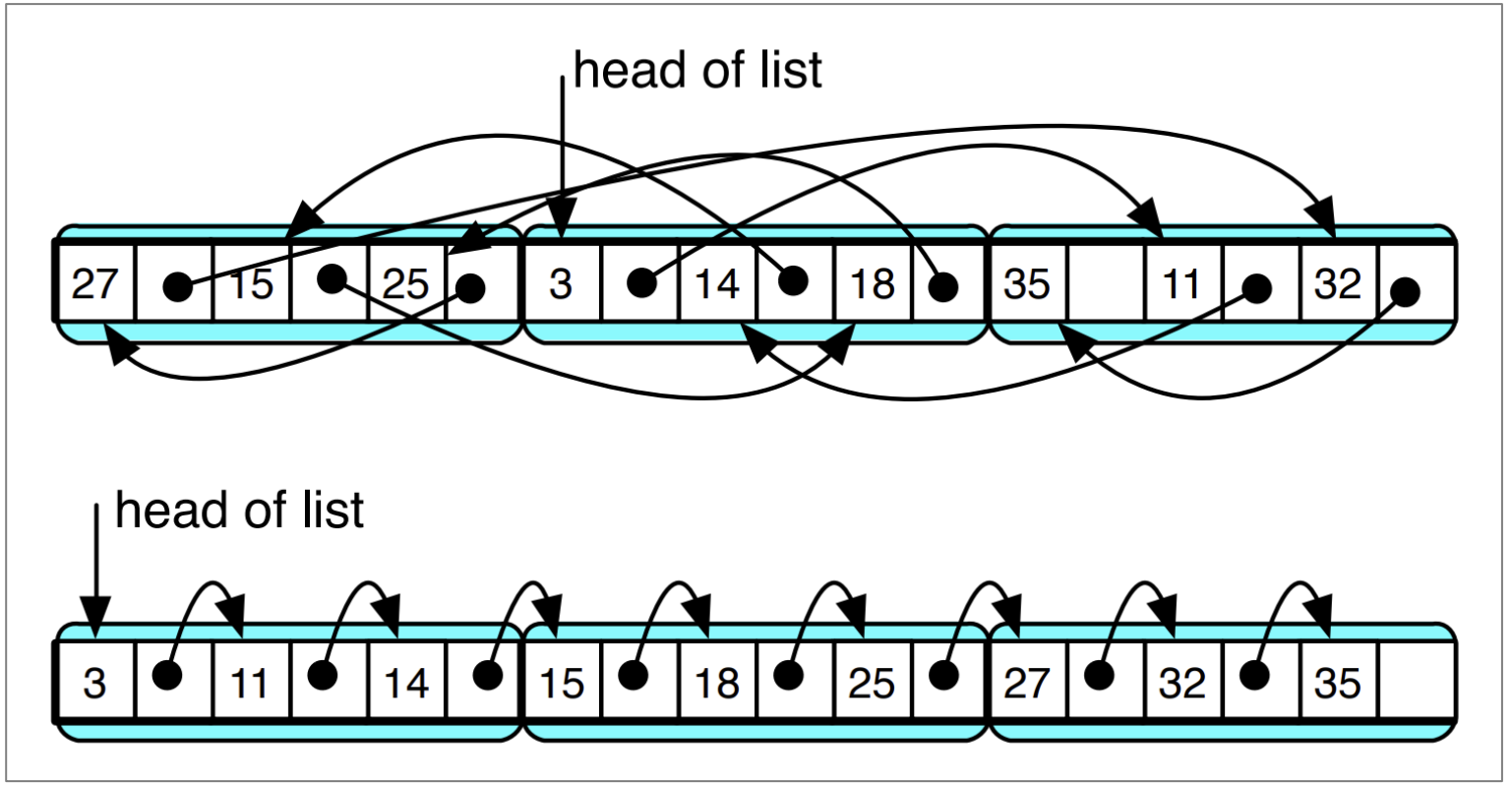
```
void func(uint64_t *a, uint64_t N)
{
    for(int i = 0; i < N; i++)
        a[i]++;
}
```

ASM Code

```
func:
    slli a1, a1, 3
    add  a1, a0, a1
.done:
    bne  a0, a1, .loop
    ret
.loop:
    ld   a5, 0(a0)
    addi a0, a0, 8
    addi a5, a5, 1
    sd   a5, -8(a0)
    j    .done
```

Data Cache Locality

- Data locality **depends on the application**
- If possible: prefer **cache-friendly** implementations



Blelloch and Harper, "Cache Efficient Functional Algorithms," CACM, 2014.

Cache-Friendly vs. Unfriendly

- C's 2D arrays are stored in **row-major order**

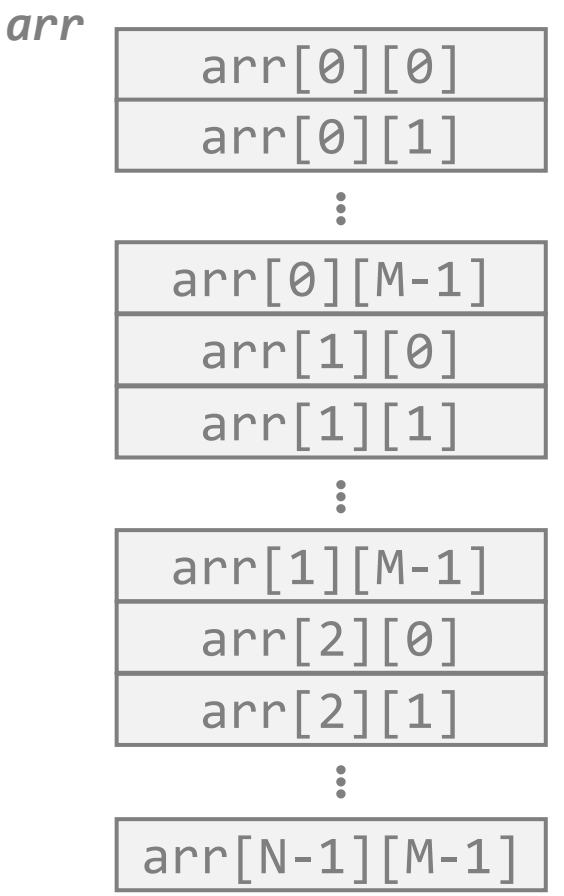
```
uint64_t arr[N * M]; // 1D version
```

	j = 0	j = 1	j = 2	...	j = M - 1
i = 0	arr[0]	arr[1]	arr[2]	...	arr[M-1]
i = 1	arr[M]	arr[M+1]	arr[M+2]	...	arr[2M-1]
i = 2	arr[2M]	arr[2M+1]	arr[2M+2]	...	arr[3M-1]
...	...	...	...	...	...
i = N-1	arr[(N-1)M]	arr[(N-1)M + 1]	arr[(N-1)M + 2]	...	arr[NM - 1]

```
for (i = 0; i < N; i++)
    for (j = 0; j < M; j++)
        sum += arr[i][j];
```

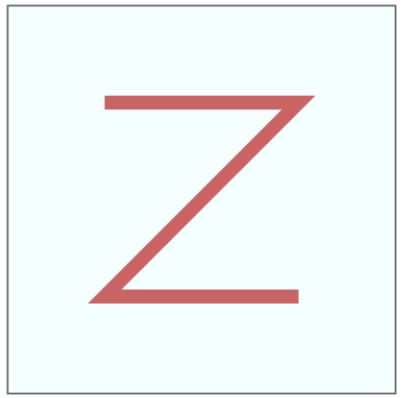
```
for (j = 0; j < M; j++)
    for (i = 0; i < N; i++)
        sum += arr[i][j];
```

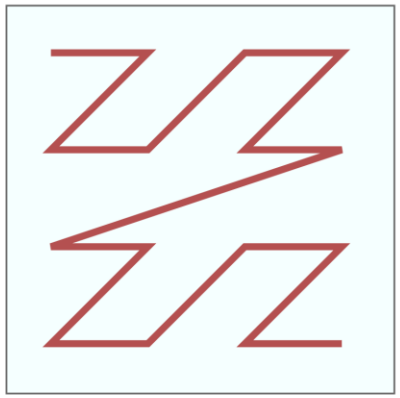
```
uint64_t arr[N][M];
    // 2D version
```

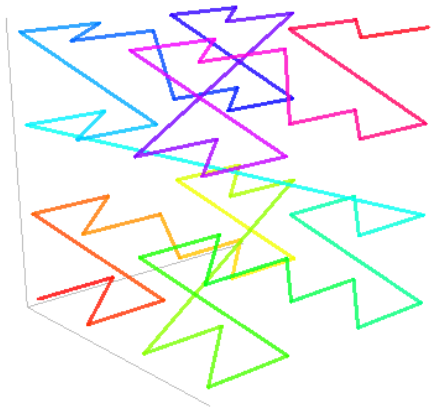


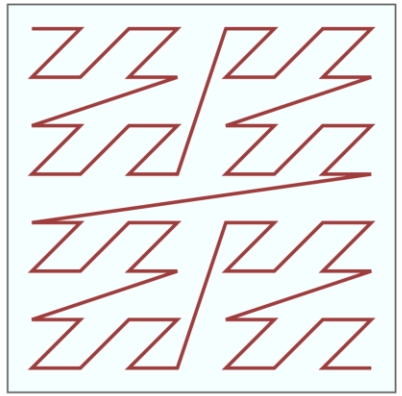
Example: Graphics Hardware

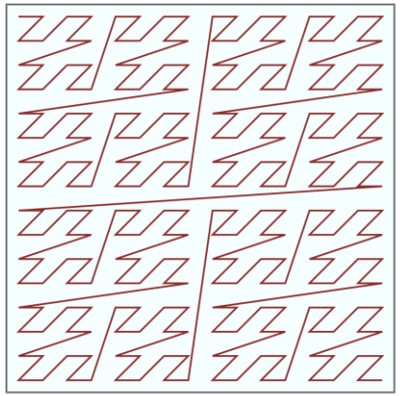
- GPU caches (often) optimize for **spatial locality** in higher dimensions

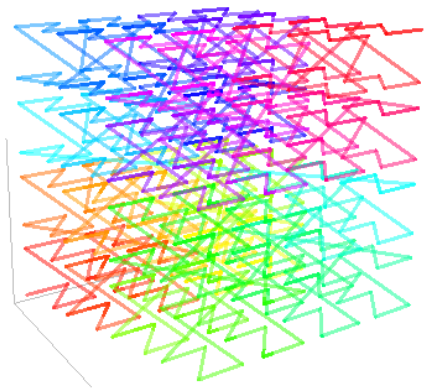


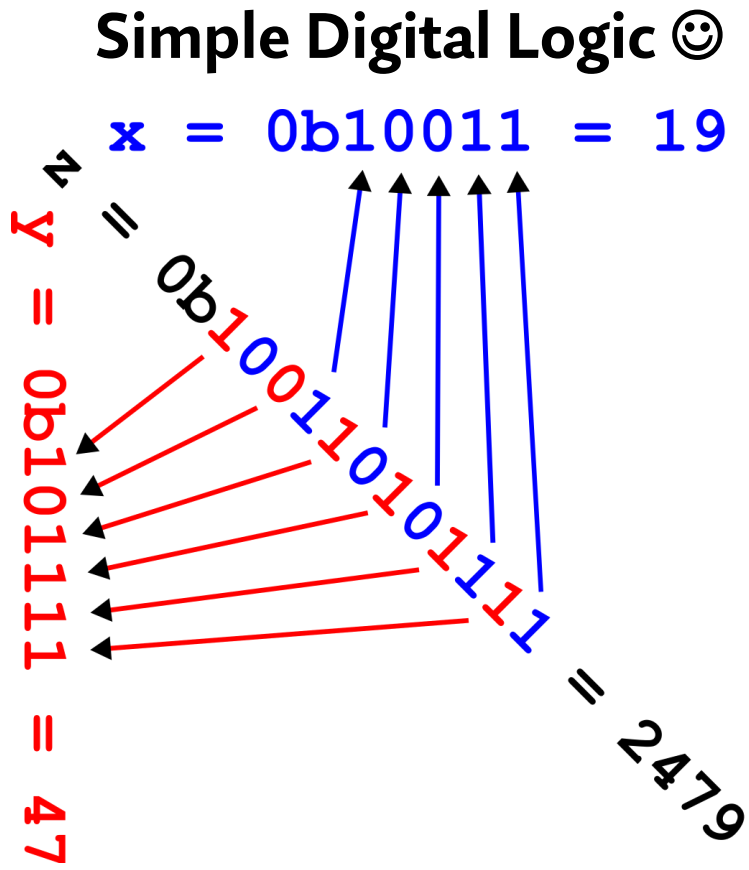








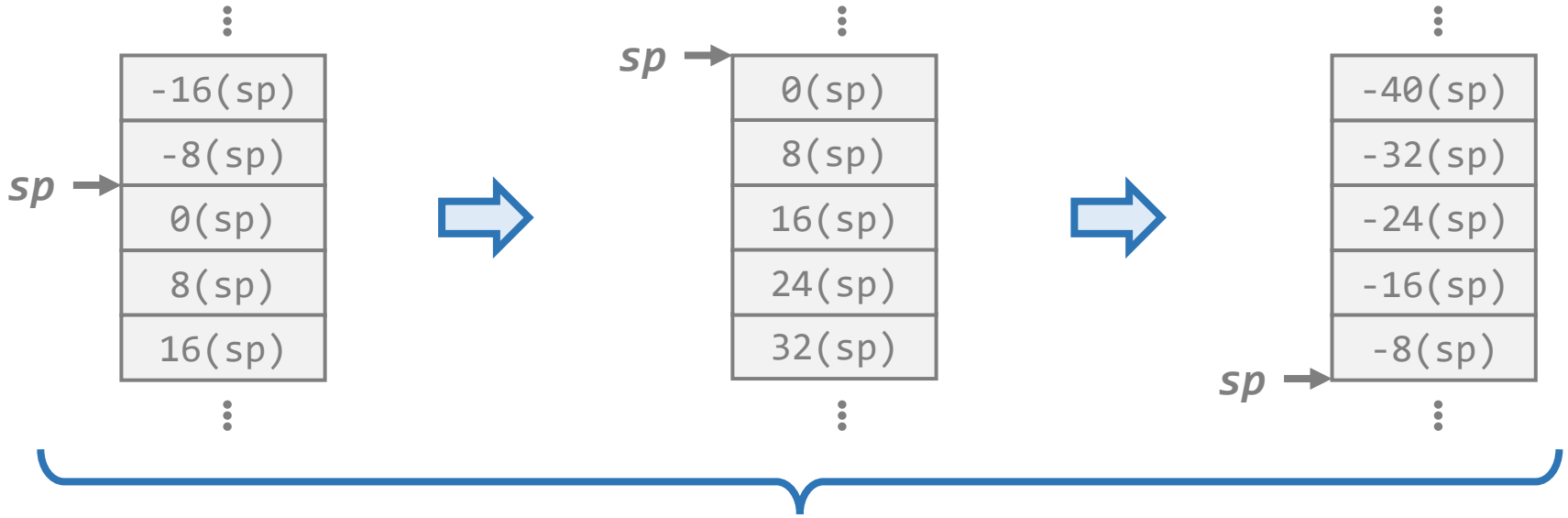




https://en.wikipedia.org/wiki/Z-order_curve

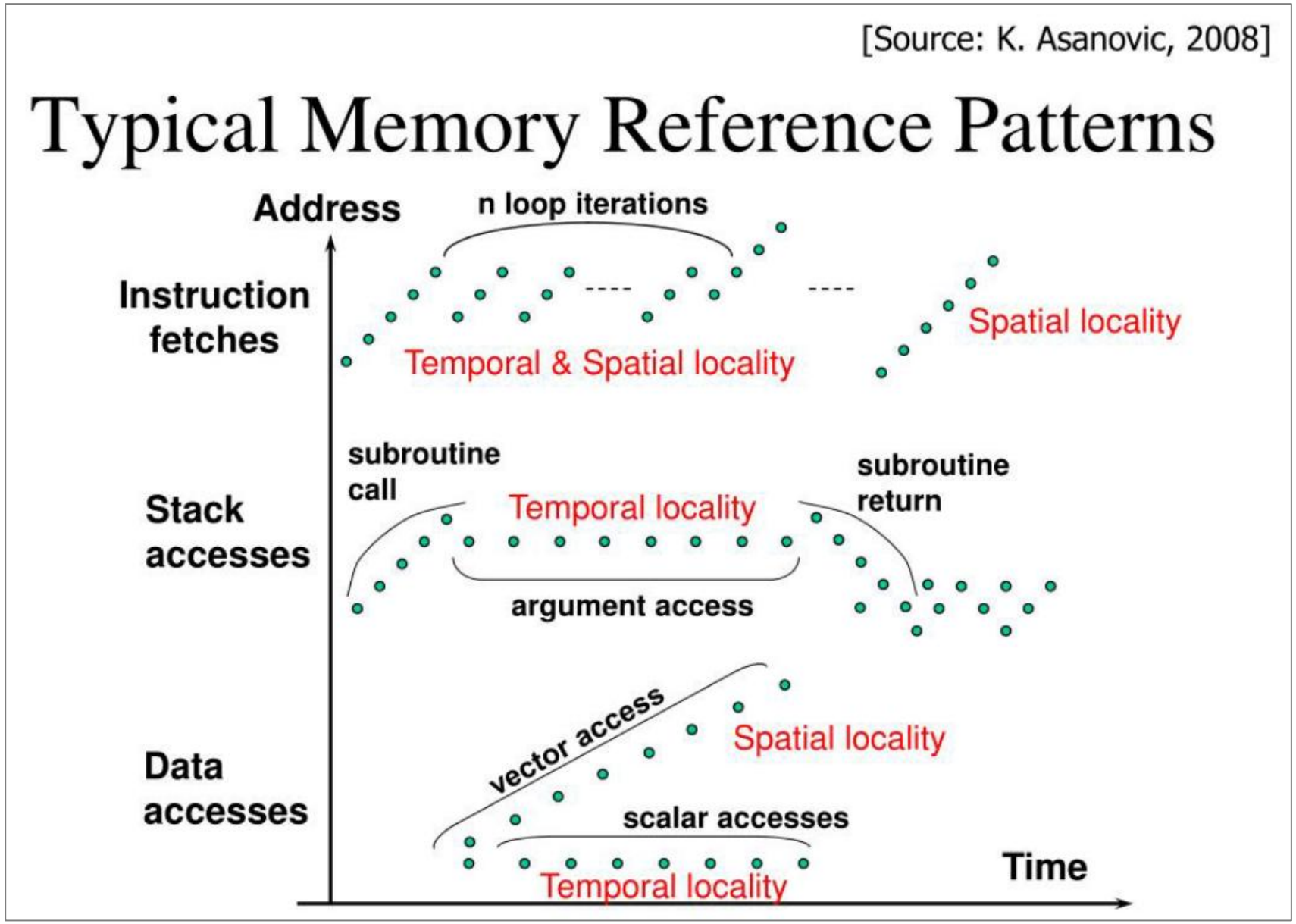
Cache-Friendly vs. Unfriendly

- The **call stack** is quite cache-friendly
  - **Spatial**: push/pop will go to the next/previous memory location
  - **Temporal**:
    - Every push has a pop
    - We will re-use stack locations as the stack grows/shrinks

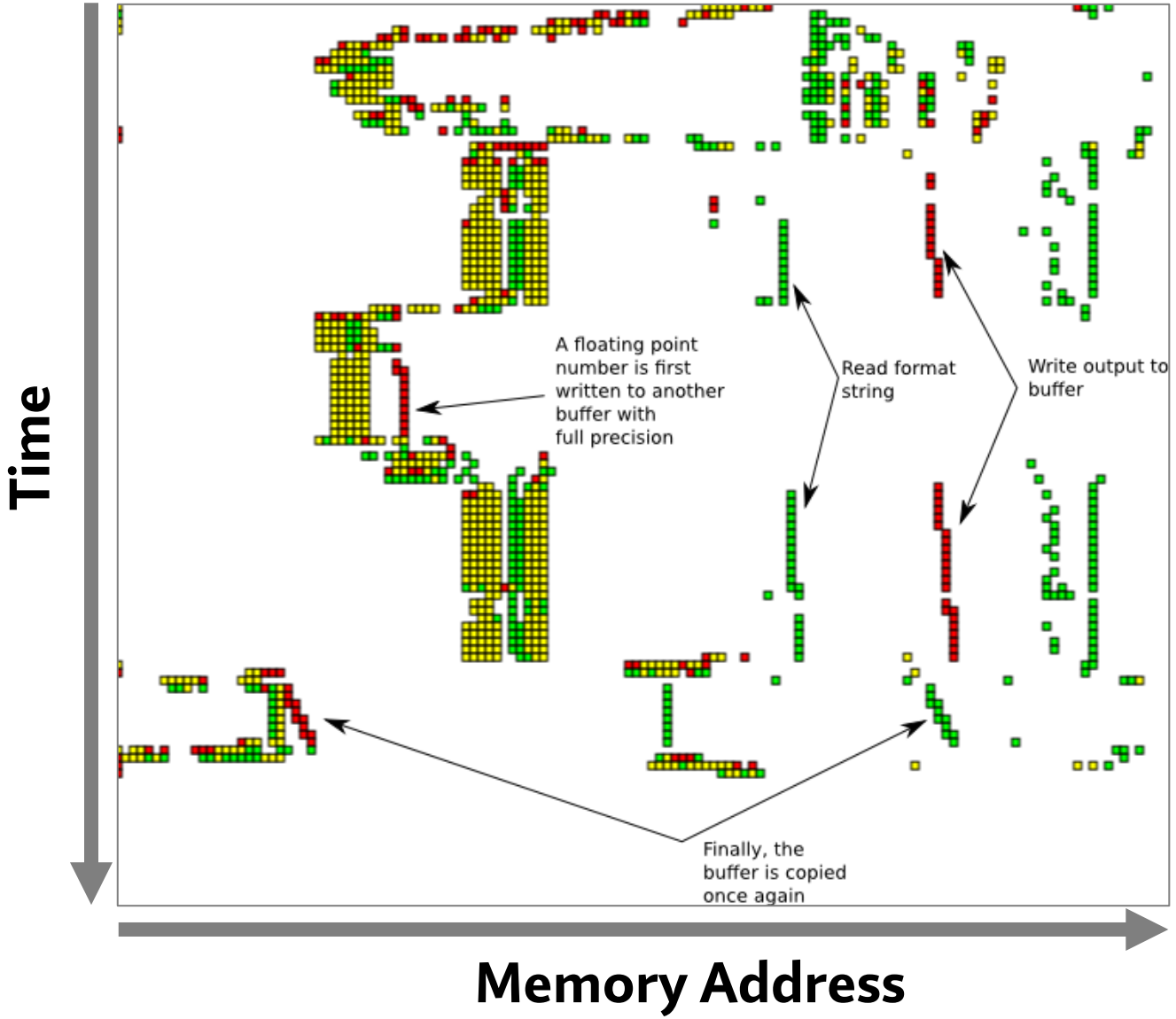


Same memory addresses are used over and over

Example Memory Reference Patterns



Example Memory Reference Pattern: printf()



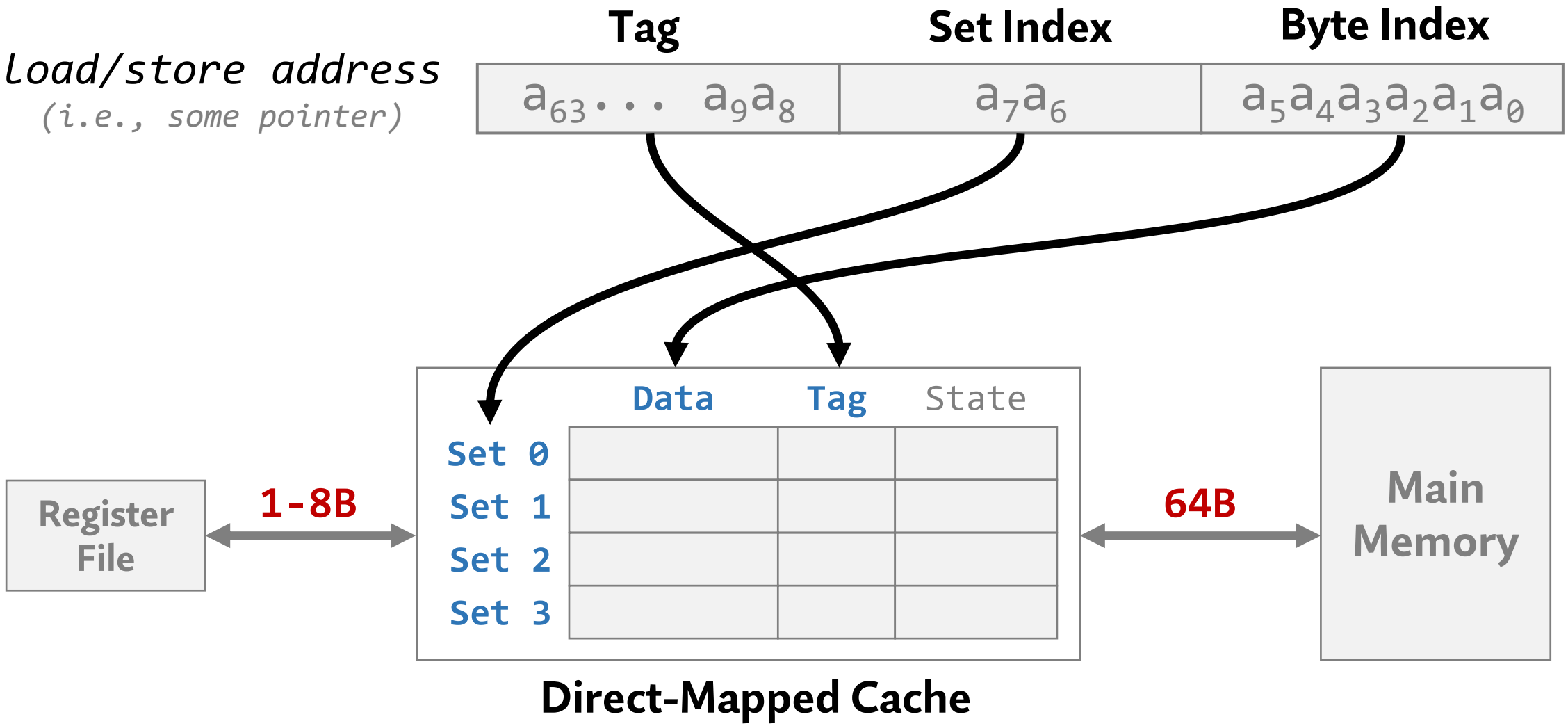
- red = write
- green = read
- yellow = read + write

https://bling.kapsi.fi/blog/x86-memory-access-visualization.html

Agenda

- Instruction Cache and Data Cache
  - **Set Index Hashing**
  - Simulating Cache Behavior
- Types of Cache Misses

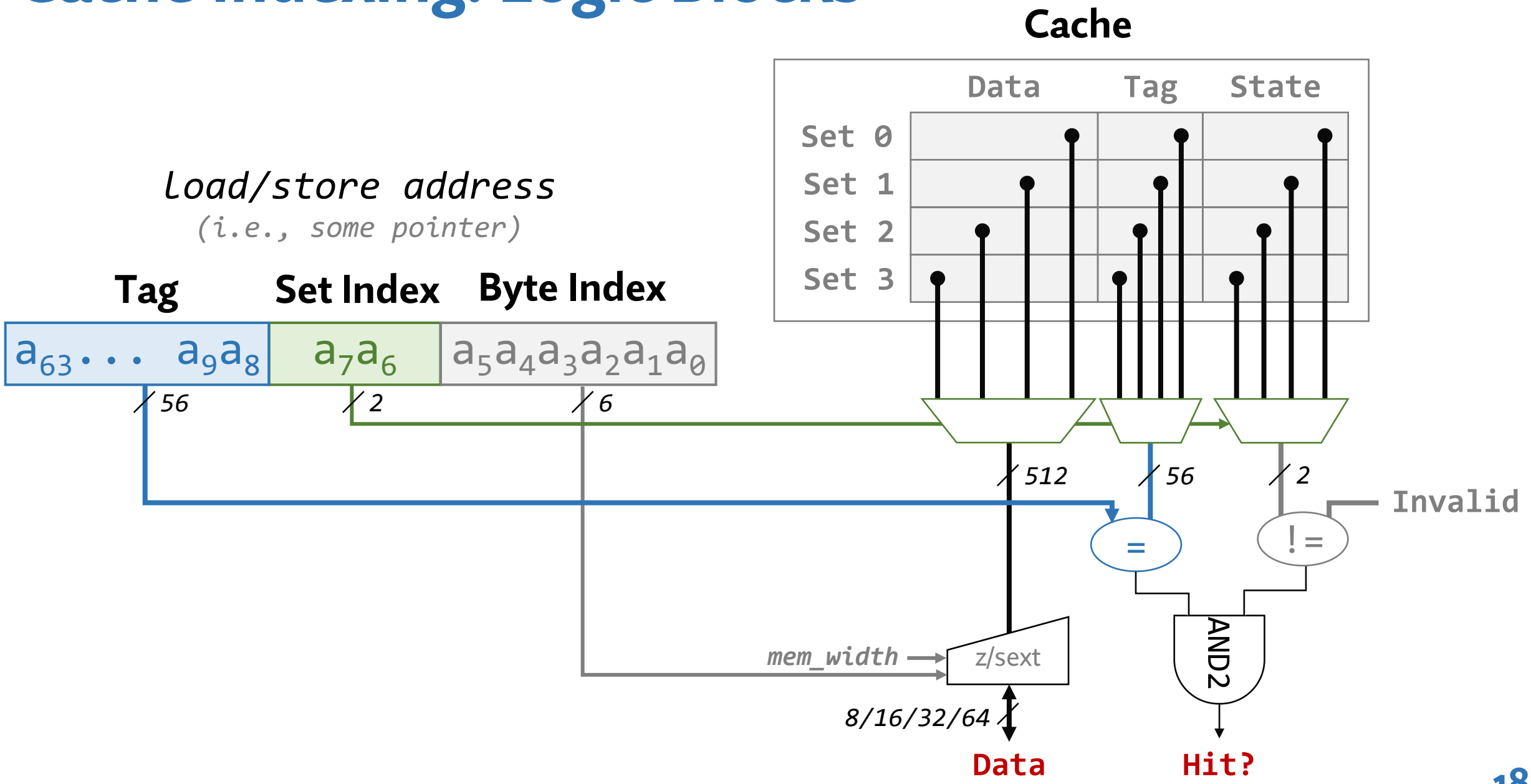
CPU-Cache-Memory Interface



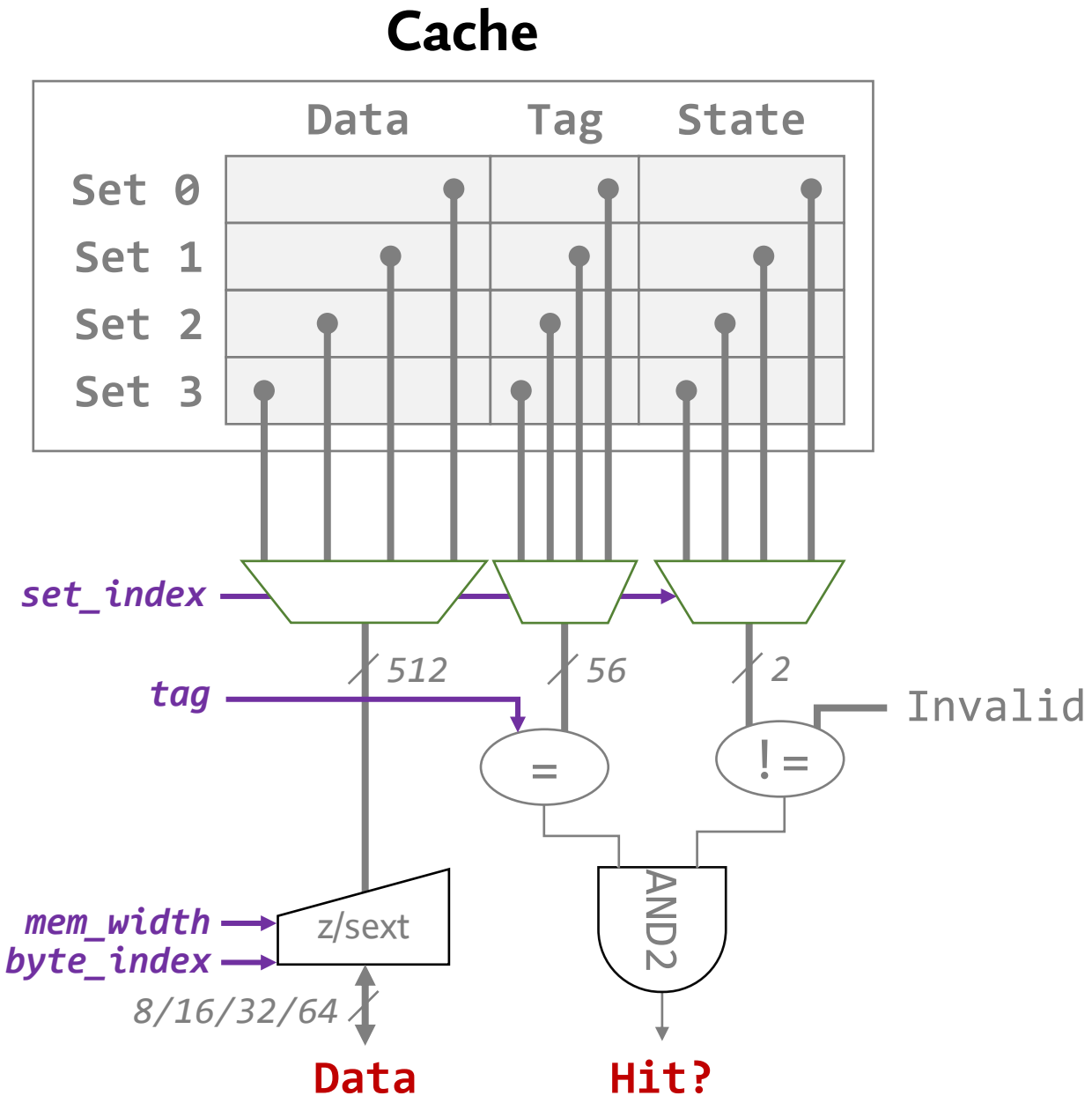
Example Problem: Cache Sizing

- Assume a **512 KB cache** with **128B blocks**:
  - How many sets are there?
  - How is an address mapped to a set?

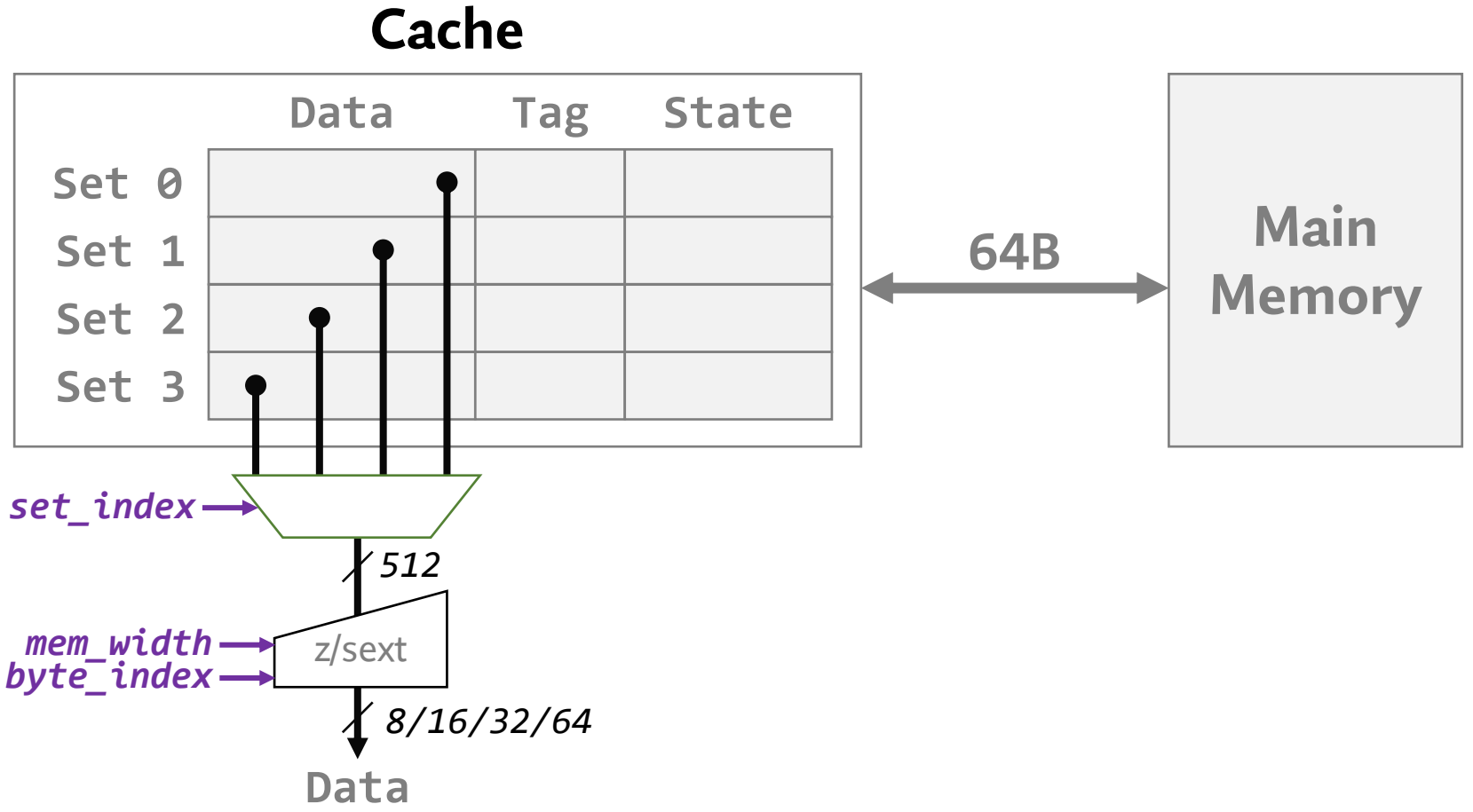
Cache Indexing: Logic Blocks



Cache Indexing is Logic Blocks + Control Signals

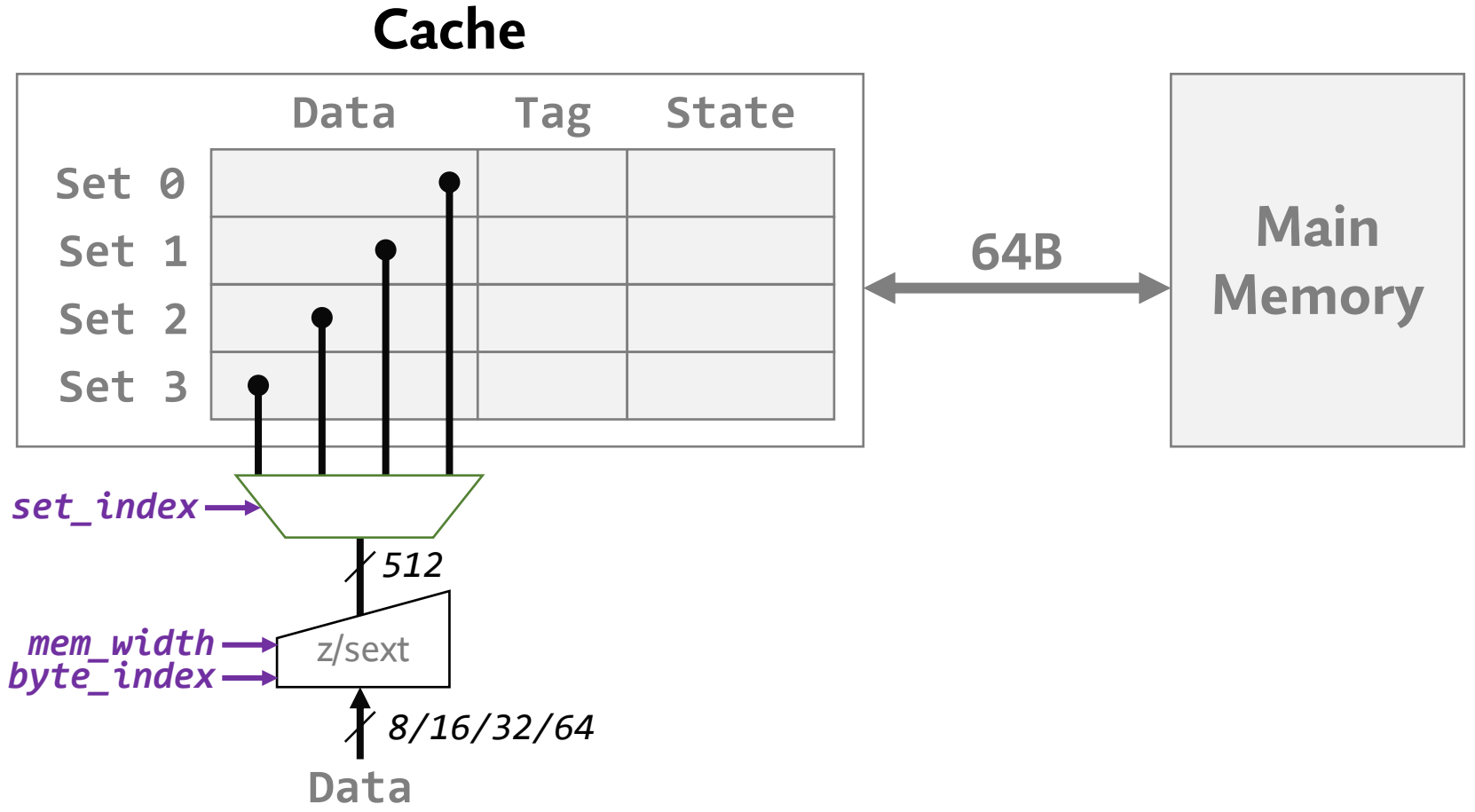


Example 1: Byte Index (ld)




```
li    a0, 0x3c0
ld    a5, 0(a0)
```

Example 1: Byte Index (sb)

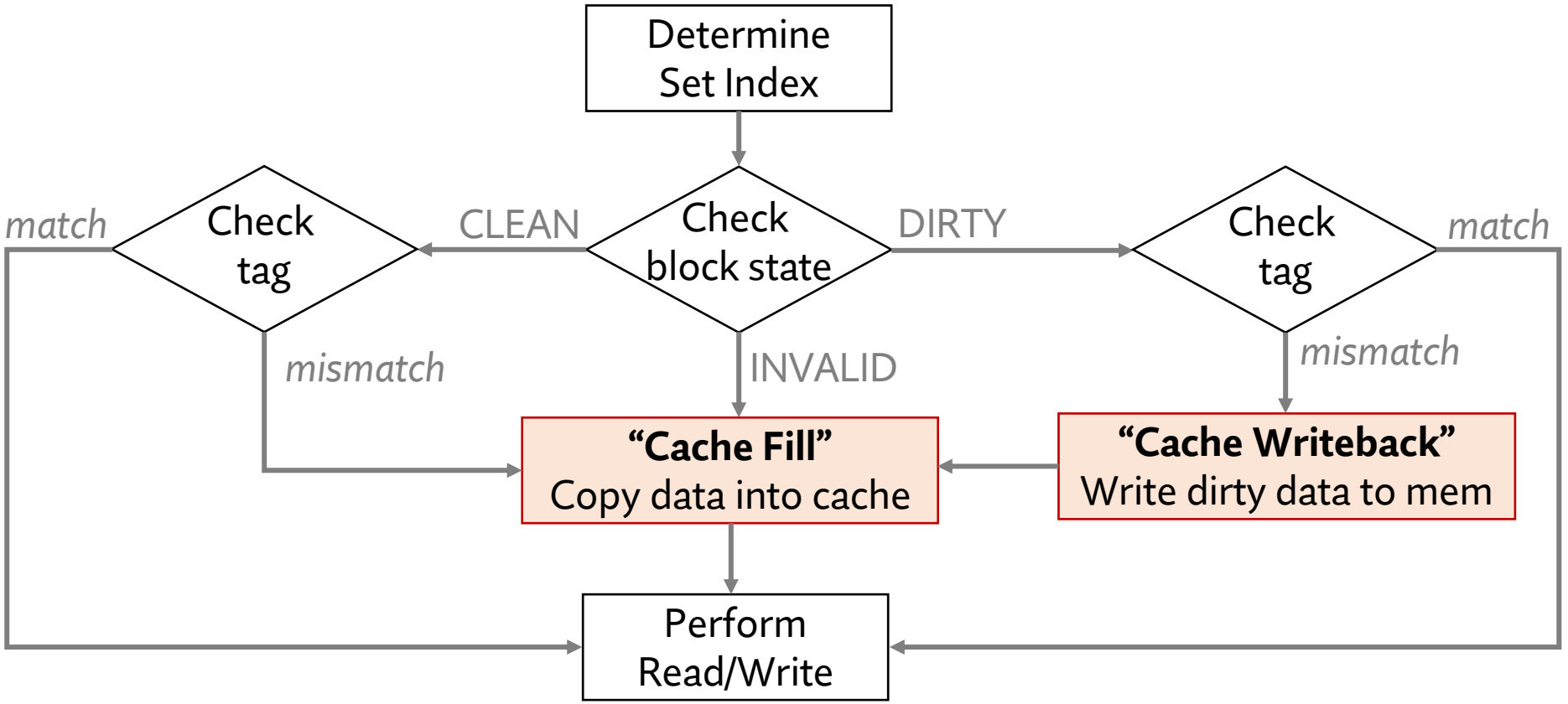



```
li    a0, 0xf83
sb    a5, 0(a0)
```

Agenda

- Instruction Cache and Data Cache
  - Set Index Hashing
  - **Simulating Cache Behavior**
- Types of Cache Misses

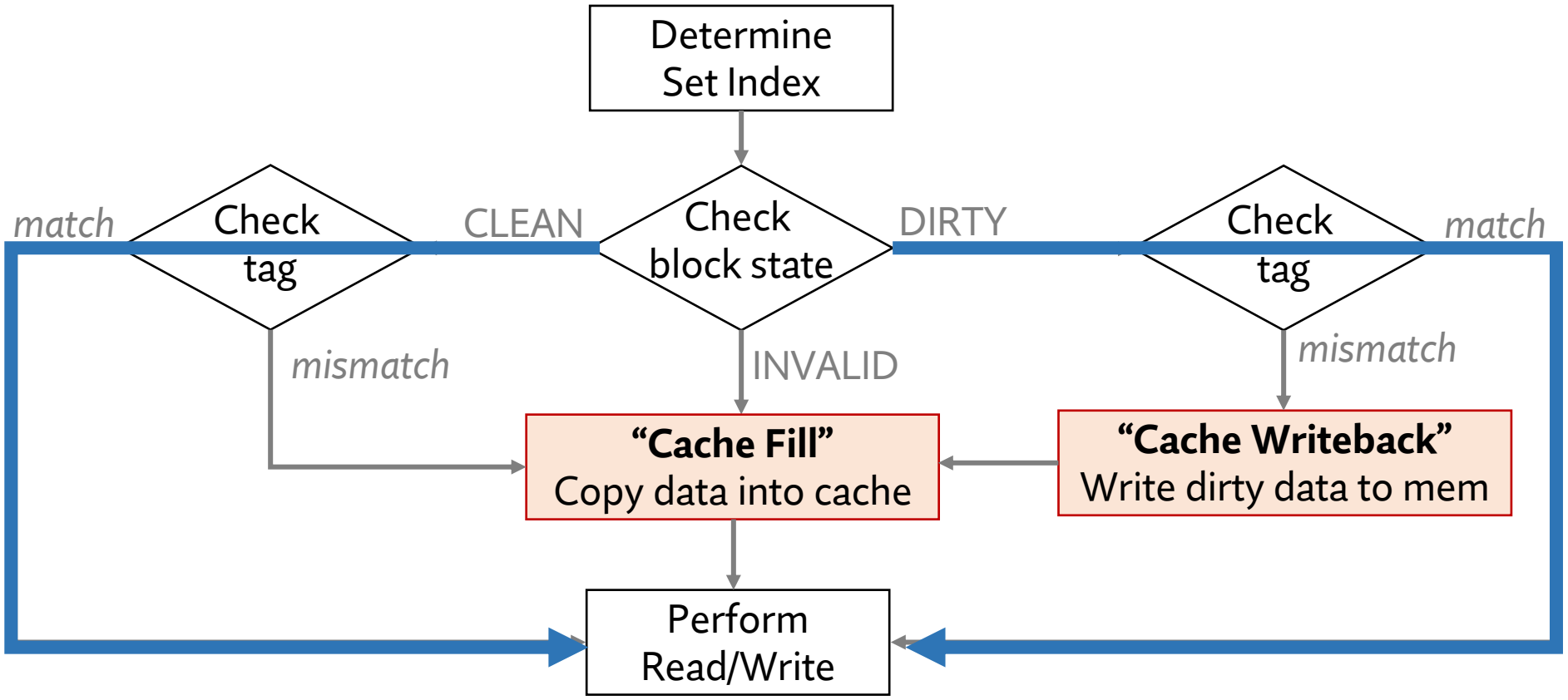
Recall: Cache Control Operations



- The cache **holds on to data** until there is a conflict for that cache block
  - **Spatial locality:** cache assumes we will also need nearby memory
  - **Temporal locality:** cache assumes we will re-use data

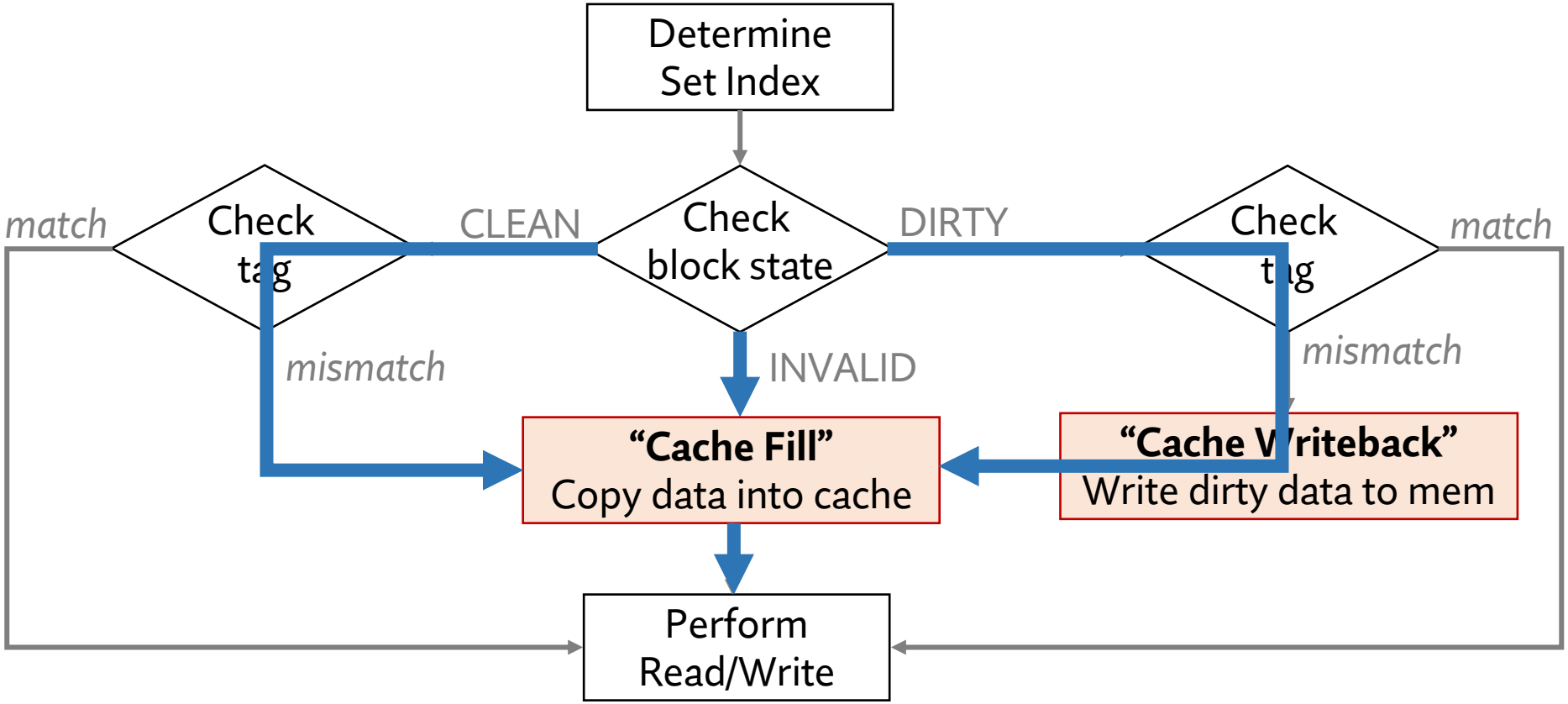
Cache Hit Operations

- **Cache Hit**: simply perform the read/write to the target block



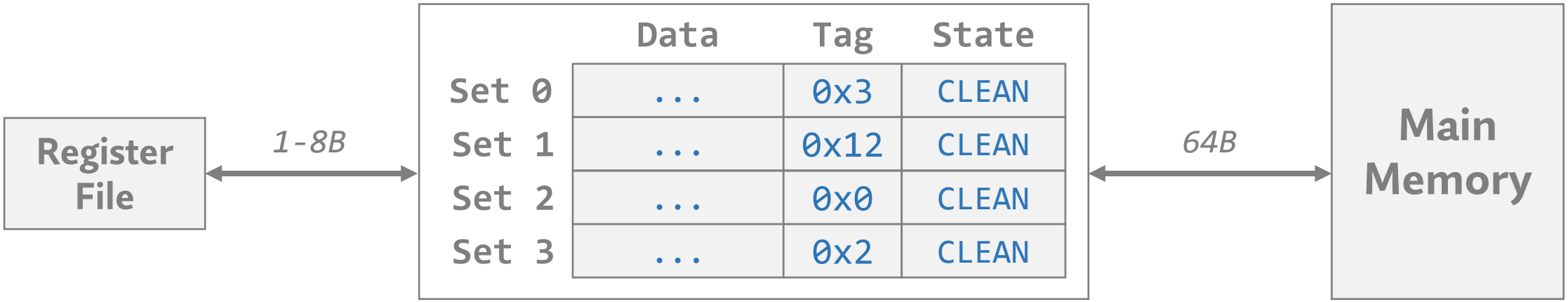
Cache Miss Operations

- **Cache miss**: always need to get the block from main memory
  - If dirty: also **write back** the previous block



Cache Conflicts

- **Conflict**: different addresses map to the same set (**hash collision**)



Cache

```
li   a0, 0x1c0
ld   a5, 0(a0)
```

```
li   a0, 0x2c0
ld   a5, 0(a0)
```

```
li   a0, 0x13f
sb   a5, 0(a0)
```

```
li   a0, 0x800
sw   a5, 0(a0)
```

Cache Simulation Example

- Assume 8-bit pointers

Tag	Set Idx	Byte Idx
a_3a_2	a_1a_0	-

Cache (4 Sets; 1B blocks)

	Data	Tag	State
Set 0			
Set 1			
Set 2			
Set 3			

Memory

0x00	A
	B
	C
	D
	E
	F
	G
	H
	I
	J
	K
	L
	M
	N
	O
0x0f	P

Memory Access Trace

```
lb    @ 0xf
sb Q @ 0x1
sb R @ 0xf
lb    @ 0x9
sb S @ 0x7
sb T @ 0x2
lb    @ 0x7
```

Cache Simulation Example

- Assume 8-bit pointers

Tag	Set Idx	Byte Idx
a_3a_2	a_1a_0	-

Cache (4 Sets; 1B blocks)

	Data	Tag	State
Set 0			
Set 1			
Set 2			
Set 3			

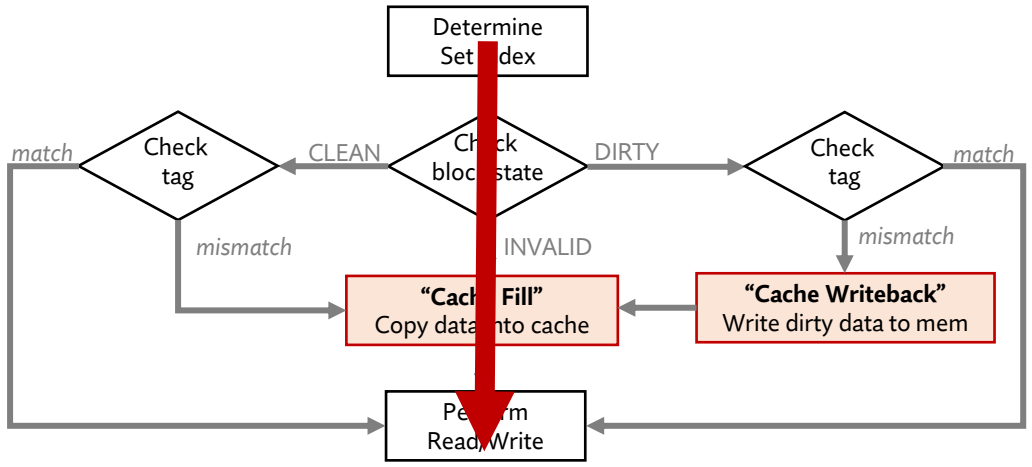
Memory

0x00	A
	B
	C
	D
	E
	F
	G
	H
	I
	J
	K
	L
	M
	N
0x0f	O
	P

Memory Access Trace

```
lb    @ 0xf
sb Q  @ 0x1
sb R  @ 0xf
lb    @ 0x9
sb S  @ 0x7
sb T  @ 0x2
lb    @ 0x7
```

MISS



Cache Simulation Example

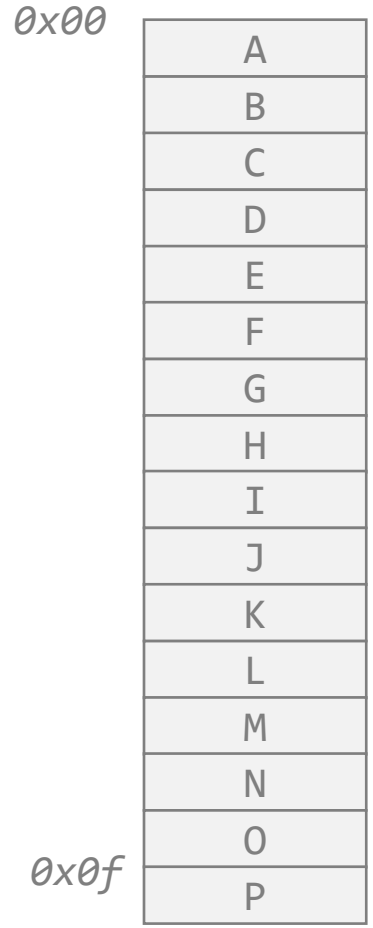
- Assume 8-bit pointers

Tag	Set Idx	Byte Idx
a_3a_2	a_1a_0	-

Cache (4 Sets; 1B blocks)

	Data	Tag	State
Set 0			
Set 1			
Set 2			
Set 3			

Memory



Memory Access Trace

```
lb    @ 0xf
sb Q @ 0x1
sb R @ 0xf
lb    @ 0x9
sb S @ 0x7
sb T @ 0x2
lb    @ 0x7
```

Cache Simulation Example: Nonzero Byte Index

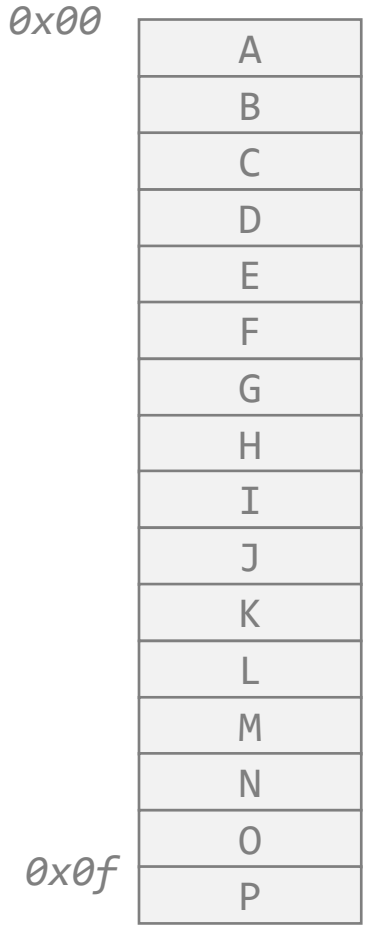
- Assume 8-bit pointers

Tag	Set Idx	Byte Idx
a_3a_2	a_1	a_0

Cache (4 Sets; 1B blocks)

	Data	Tag	State
Set 0			
Set 1			

Memory



Memory Access Trace

```
lb    @ 0xf
sb Q @ 0x1
sb R @ 0x4
```

Agenda

- Instruction Cache and Data Cache
  - Set Index Hashing
  - Simulating Cache Behavior
- **Types of Cache Misses**

Three Types of Cache Misses

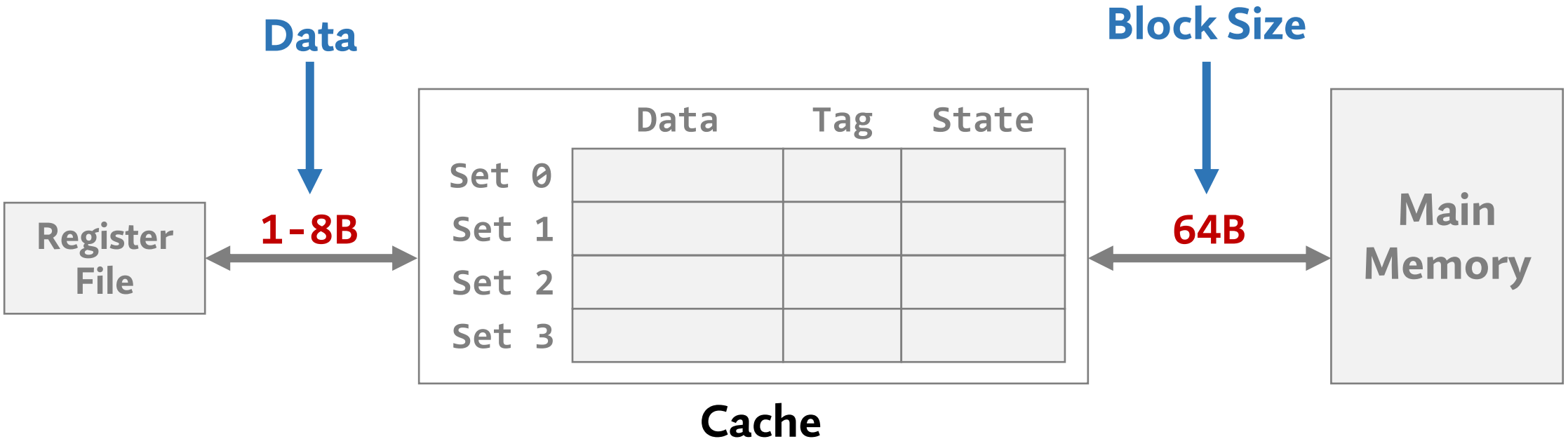
1. **Cold (compulsory) Miss:** never seen this block before
   - Unavoidable without *prediction*

2. **Conflict Miss**: blocks compete for a set
   - Can make the cache bigger
   - Can change the set index hash function
   - Can allow *(set associative caches – more on Thursday)*

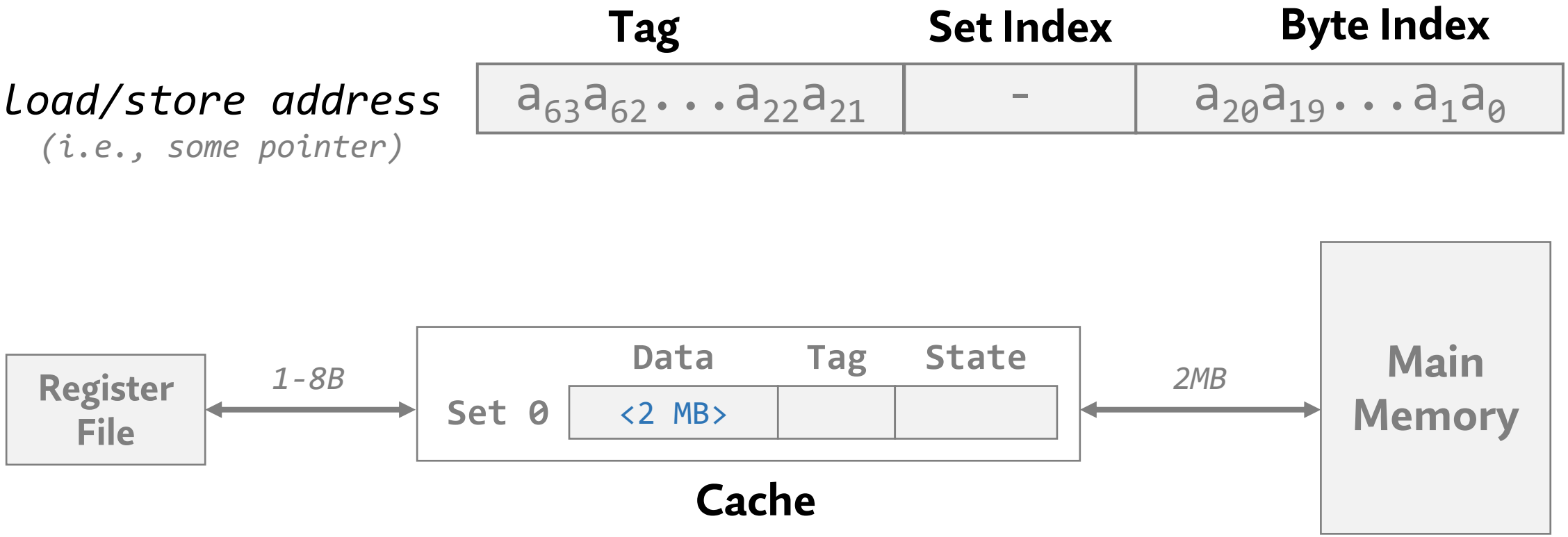
3. **Capacity Miss**: cache is too small

Mitigating Cold Misses

- Requires **predicting** that we need the block beforehand
  - i.e., **prefetching** the block
- We've actually already been prefetching!

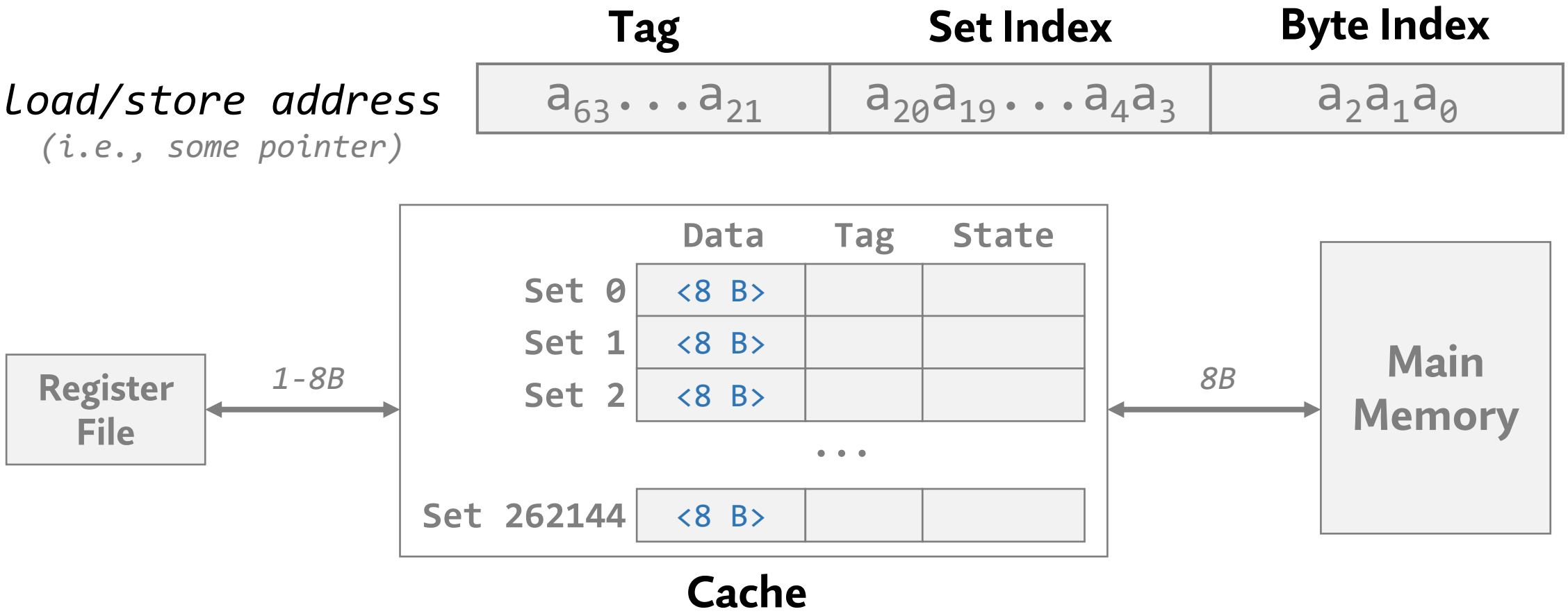


Extreme Example: One Giant Block



- We prefetch ~2MB of data for **any cold miss**
- **Q: Good or bad idea?**

Extreme Example: Many Tiny Blocks



- We **never prefetch** based on cold misses
- **Q: Good or bad idea?**

Thursday: Cache Design Tradeoffs

CS 211: Intro to Computer Architecture

14.1: Direct-Mapped Caches (In Detail)

Minesh Patel

Spring 2025 – Tuesday 29 April