Computer Organization and Architecture

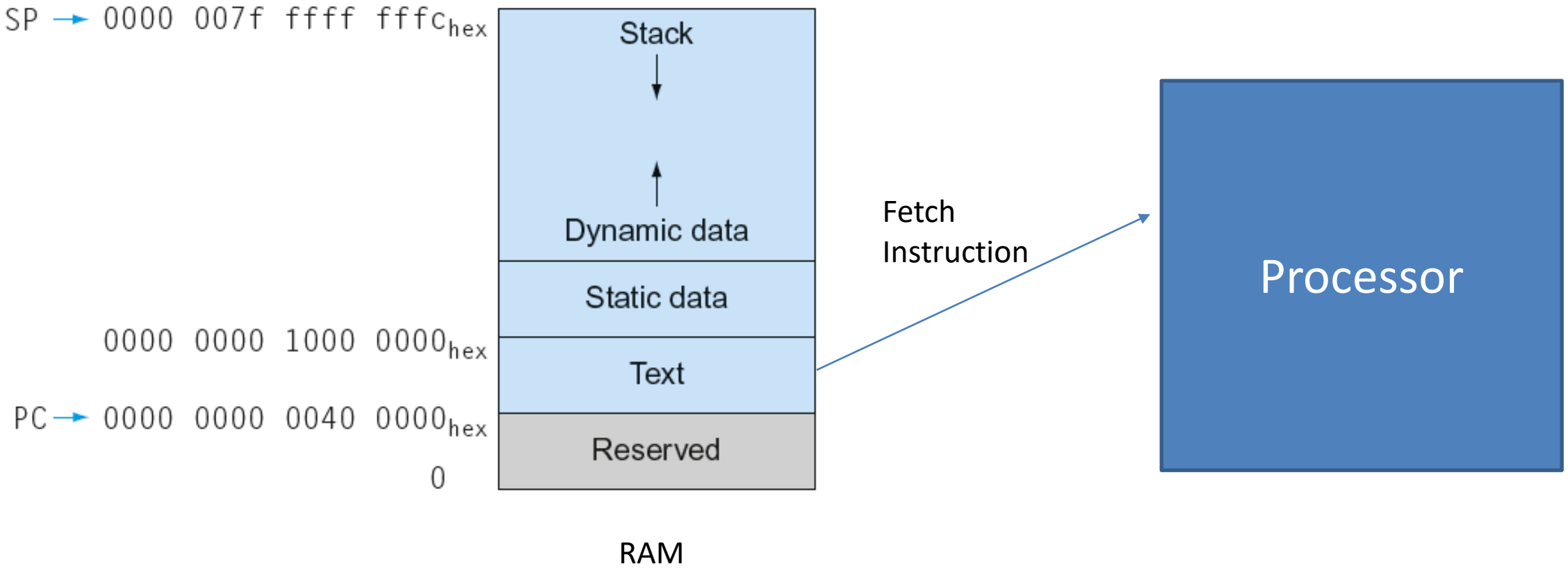
Lecture – 22

Nov 2nd, 2022

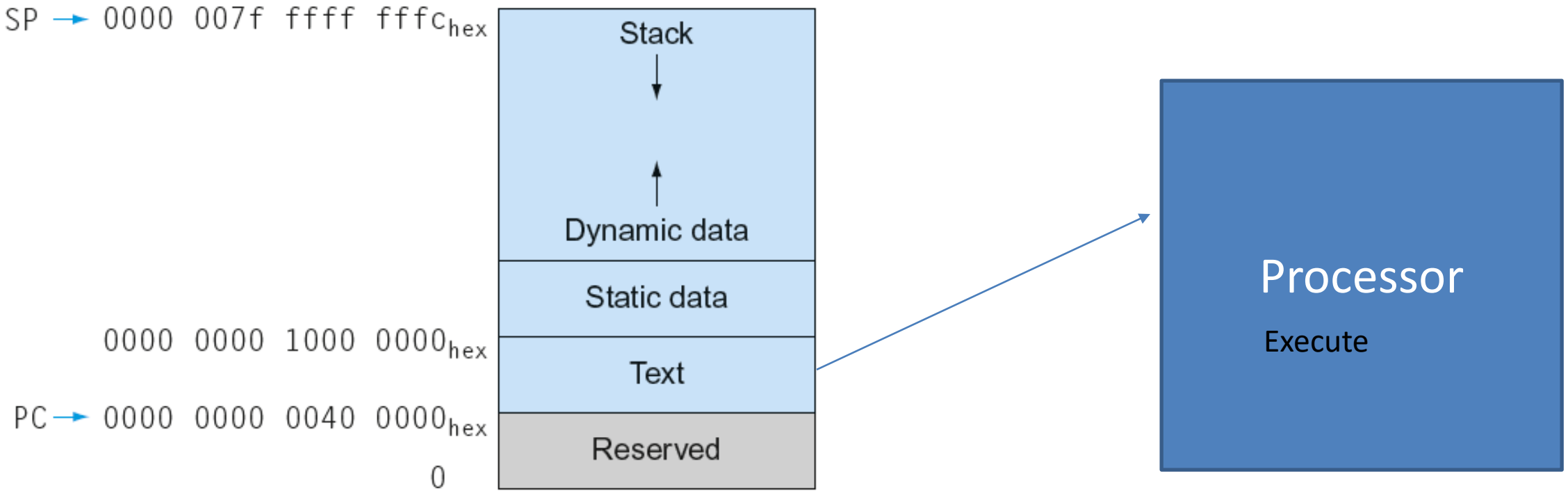
Chapter 5

Large and Fast: Exploiting Memory Hierarchy

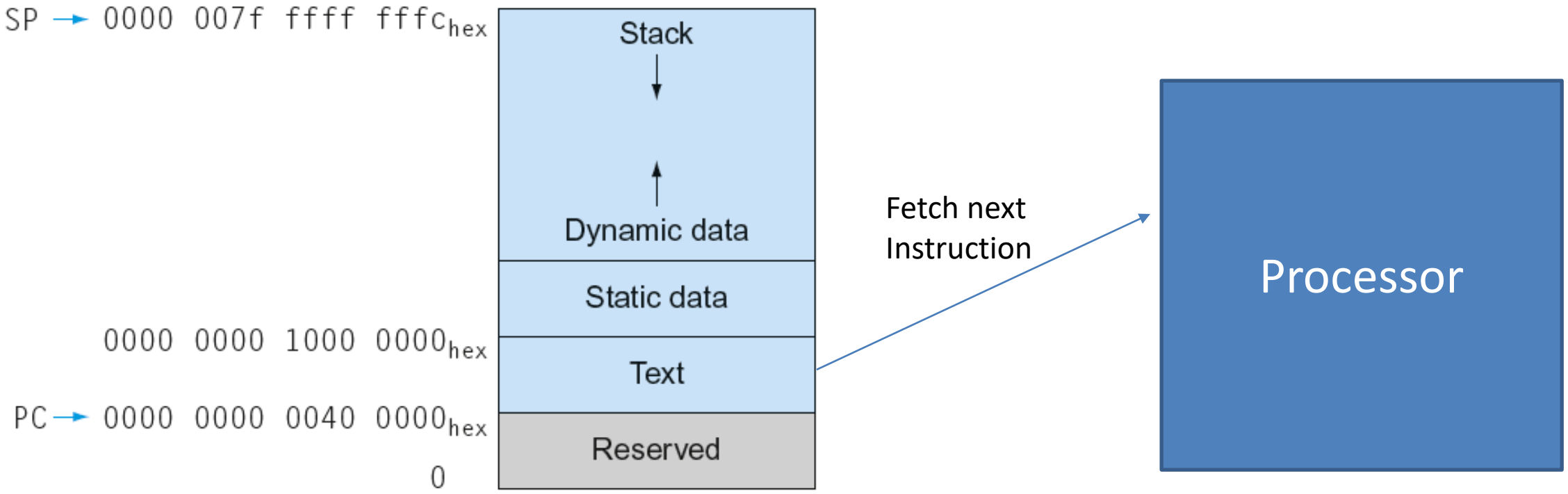
Executing instructions



Executing instructions

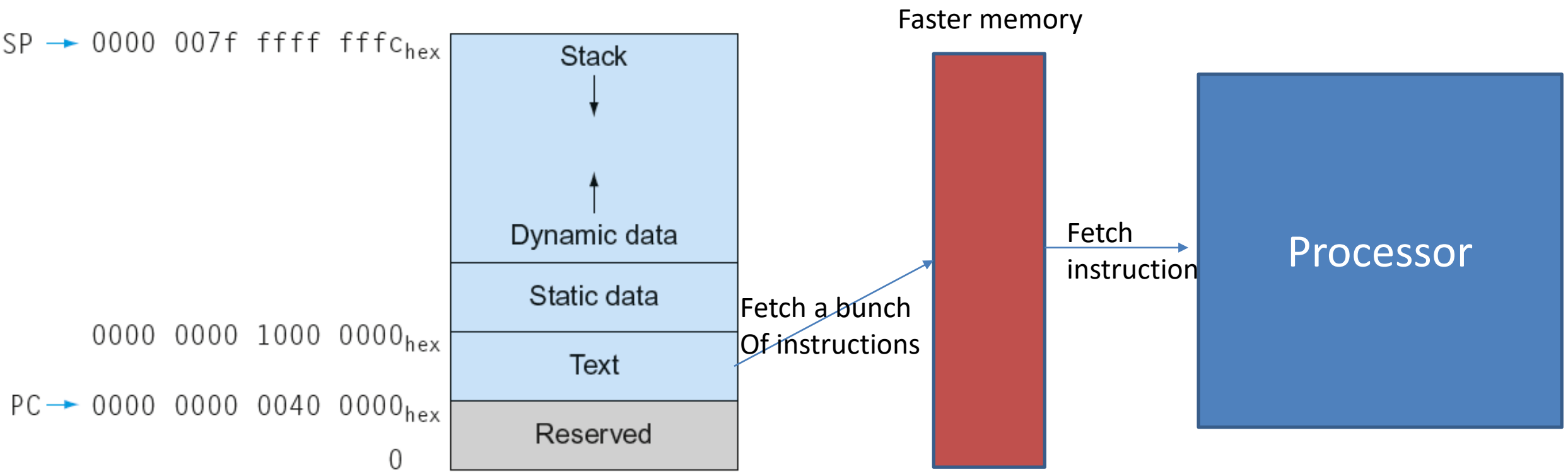


Executing instructions



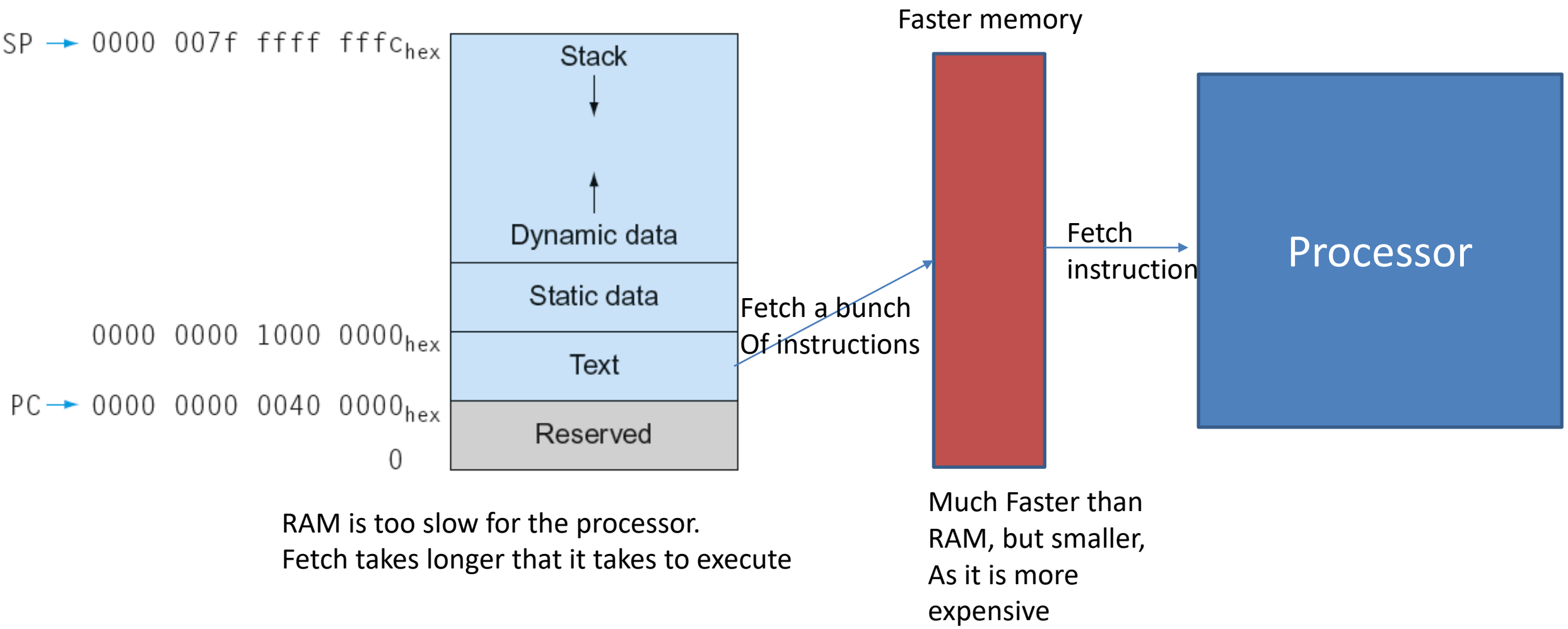
RAM is too slow for the processor.
Fetch takes longer that it takes to execute

Executing instructions



RAM is too slow for the processor.
Fetch takes longer that it takes to execute

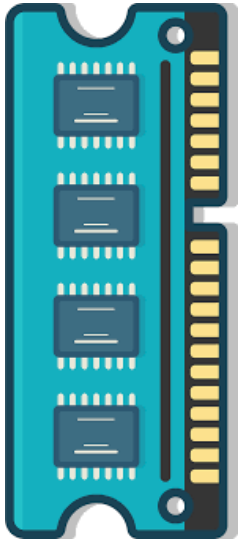
Executing instructions



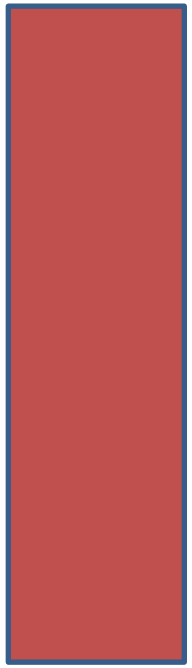
Hierarchy of memories

Stored in memory
HDD

RAM – Random
access memory



Faster memory

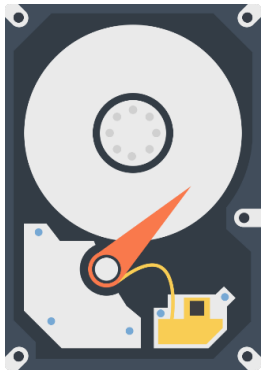


Memories closer to processor are faster but smaller in size
and more expensive

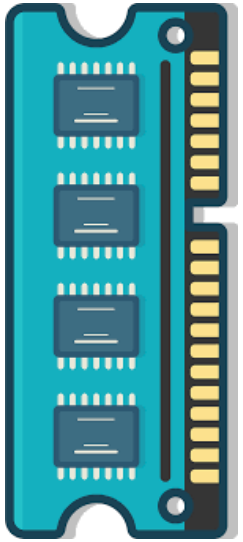
Hierarchy of memories

1-2 TB

Stored in memory
HDD

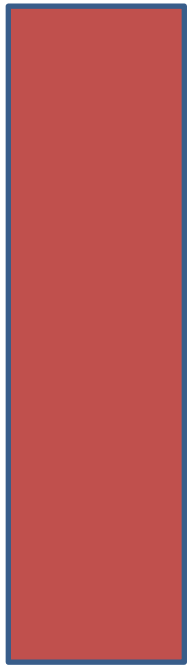


8-32 GB
RAM – Random
access memory



16-30 MB

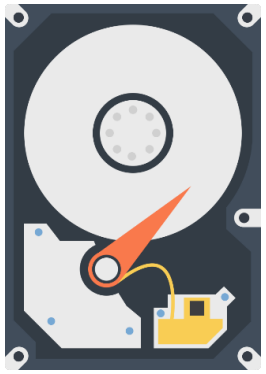
Faster memory



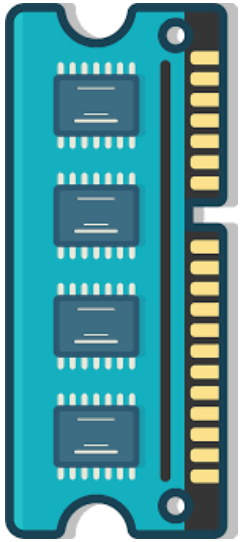
As memories closer are smaller, not all data can be accommodated.

Hierarchy of memories

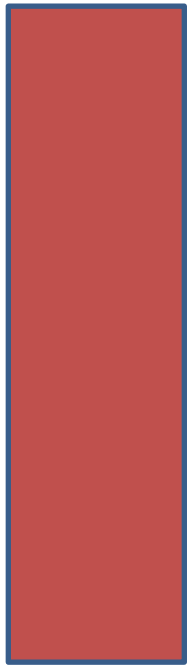
Stored in memory
HDD



RAM – Random access memory



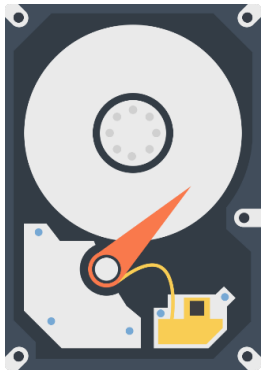
Faster memory



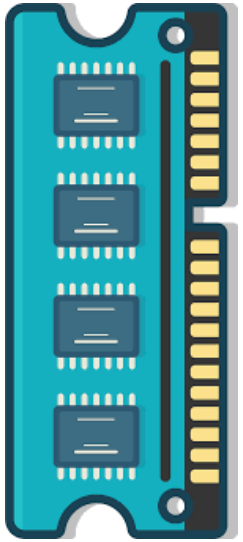
As memories closer are smaller, not all data can be accommodated.
On each fetch, If we fetch each instruction one at a times all the way from the SSD, the intermediate memories are redundant

Hierarchy of memories

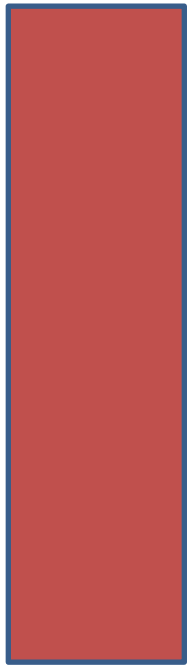
Stored in memory
HDD



RAM – Random
access memory



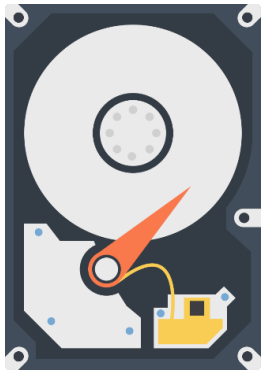
Faster memory



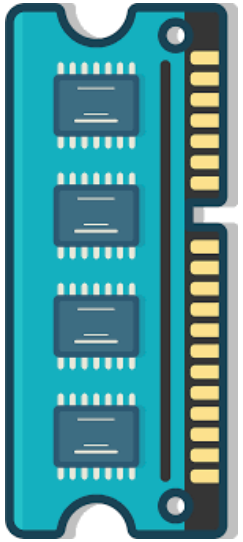
How to fetch data efficiently so that relevant information is more often available in the memories closer to the Processor.
(or) create an illusion that all the data is always available in the faster memories.

Hierarchy of memories

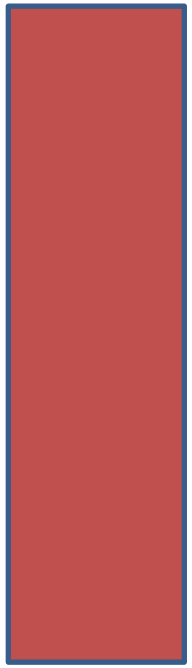
Stored in memory
HDD



RAM – Random
access memory



Faster memory



How to fetch a head of time?

Example

```
LDUR    X1, [X0,#0]
LDUR    X2, [X0,#8]
ADD     X3, X1, X2
STUR    X3, [X0,#24]
LDUR    X4, [X0,#16]
ADD     X5, X1, X4
STUR    X5, [X0,#32]
```

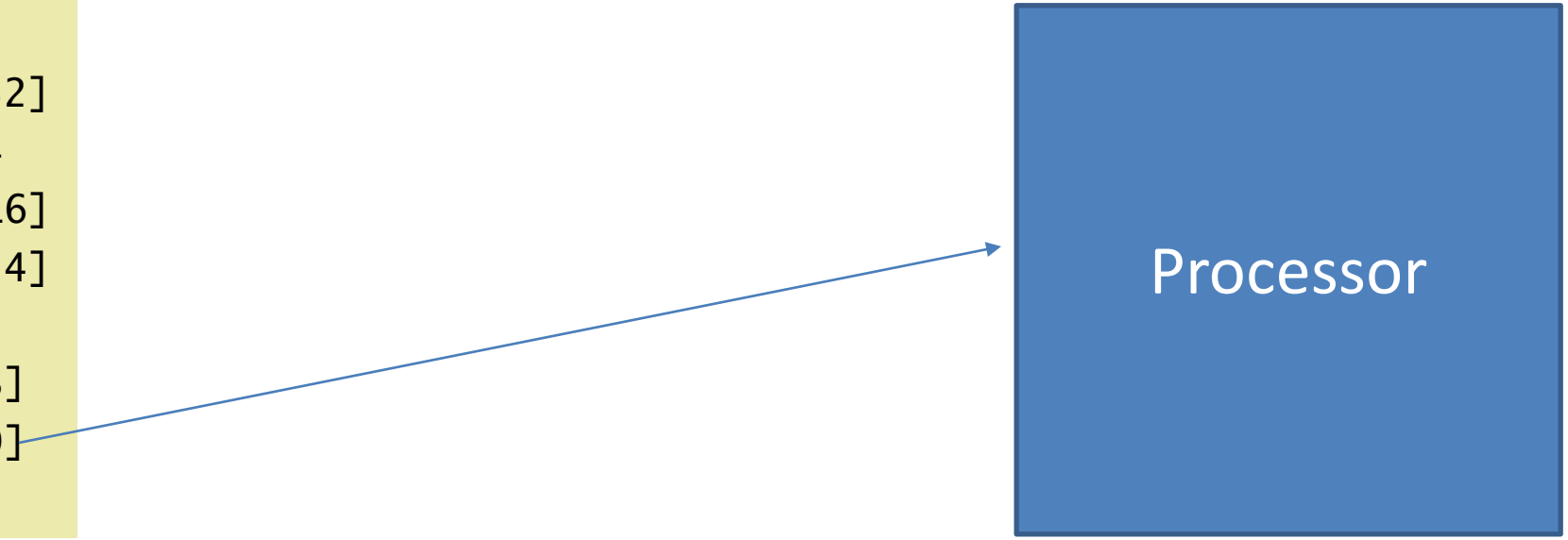
Example

```
LDUR    X1, [X0,#0]
LDUR    X2, [X0,#8]
ADD     X3, X1, X2
STUR    X3, [X0,#24]
LDUR    X4, [X0,#16]
ADD     X5, X1, X4
STUR    X5, [X0,#32]
```

```
ADDR
    7 STUR    X5, [X0,#32]
    6 ADD     X5, X1, X4
    5 LDUR    X4, [X0,#16]
    4 STUR    X3, [X0,#24]
    3 ADD     X3, X1, X2
    2 LDUR    X2, [X0,#8]
    1 LDUR    X1, [X0,#0]
```

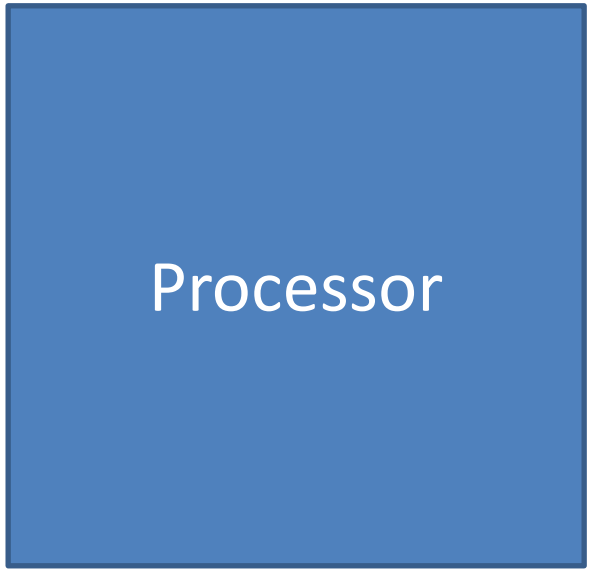
Example

```
ADDR
    7 STUR      X5, [X0,#32]
    6 ADD       X5, X1, X4
    5 LDUR      X4, [X0,#16]
    4 STUR      X3, [X0,#24]
    3 ADD       X3, X1, X2
    2 LDUR      X2, [X0,#8]
    1 LDUR      X1, [X0,#0]
```



Example

```
ADDR
    7 STUR      X5, [X0,#32]
    6 ADD       X5, X1, X4
    5 LDUR      X4, [X0,#16]
    4 STUR      X3, [X0,#24]
    3 ADD       X3, X1, X2
    2 LDUR      X2, [X0,#8]
    1 LDUR      X1, [X0,#0]
```



Example

```
ADDR
    7 STUR      X5, [X0,#32]
    6 ADD       X5, X1, X4
    5 LDUR      X4, [X0,#16]
    4 STUR      X3, [X0,#24]
    3 ADD       X3, X1, X2
    2 LDUR      X2, [X0,#8]
    1 LDUR      X1, [X0,#0]
```

Instruction closely located in memory are more likely to be accessed

Processor

Example

```
ADDR
    7 STUR      X5, [X0,#32]
    6 ADD       X5, X1, X4
    5 LDUR      X4, [X0,#16]
    4 STUR      X3, [X0,#24]
    3 ADD       X3, X1, X2
    2 LDUR      X2, [X0,#8]
    1 LDUR      X1, [X0,#0]
```

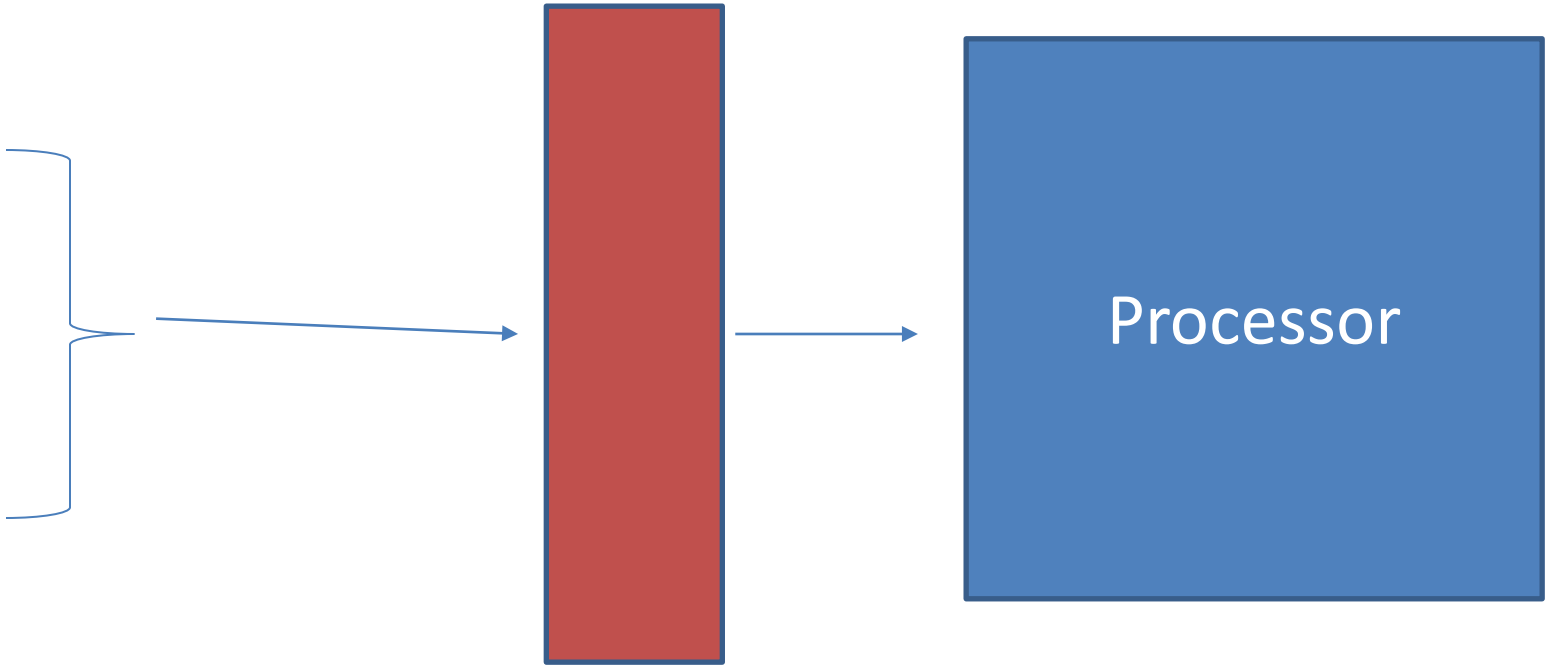
Processor

Instruction closely located in memory are more likely to be accessed

Spatial Locality

Example

```
ADDR
    7  STUR      X5, [X0,#32]
    6  ADD       X5, X1, X4
    5  LDUR      X4, [X0,#16]
    4  STUR      X3, [X0,#24]
    3  ADD       X3, X1, X2
    2  LDUR      X2, [X0,#8]
    1  LDUR      X1, [X0,#0]
```



Instruction closely located in memory are more likely to be accessed

Spatial Locality

Example - 2

```
for (i =0; i< 100, i++){
        a += 1
}
```

i in register X0

a in register X1

```
        ADD XO, XZR, XZR
Loop: ADDI X1, X1, #1
        ADDI X0, X0, #1
        SUBI X2, X0, #100
        CBNZ X2, Loop
```

Example - 2

```
for (i =0; i< 100, i++){
        a += 1
}
```

i in register X0

a in register X1

```
        ADD XO, XZR, XZR
Loop: ADDI X1, X1, #1
        ADDI X0, X0, #1
        SUBI X2, X0, #100
        CBNZ X2, Loop
```

ADDR

5 *CBNZ X2,* ***2***

4 SUBI X2, X0, #100

3 ADDI X0, X0, #1

2 *ADDI X1, X1, #1*

1 ADD XO, XZR, XZR

Example - 2

ADDR

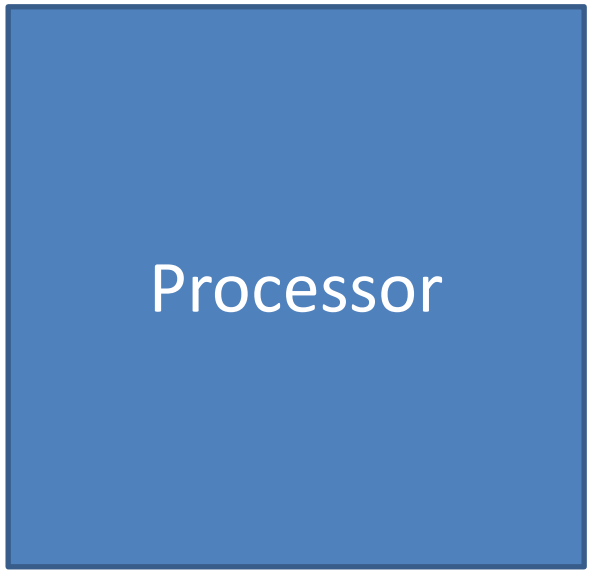
5 *CBNZ X2,* ***2***

4 SUBI X2, X0, #100

3 ADDI X0, X0, #1

2 *ADDI X1, X1, #1*

1 ADD XO, XZR, XZR



Instructions 2, 3, 4, and 5 are accessed a 100 times repeatedly.

Example - 2

ADDR

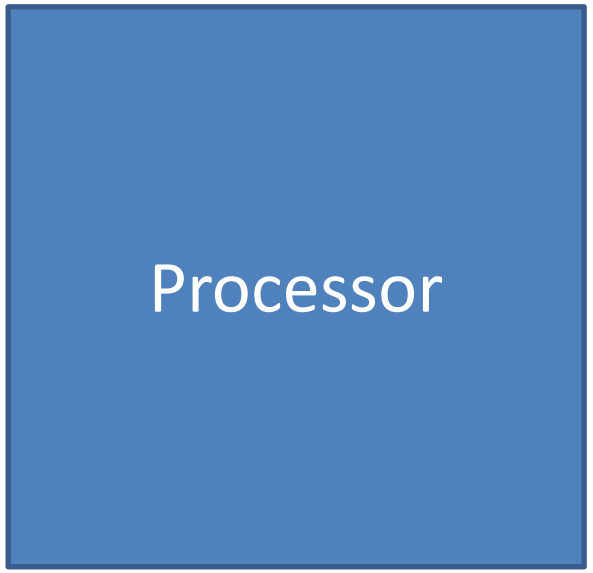
5 *CBNZ X2,* ***2***

4 SUBI X2, X0, #100

3 ADDI X0, X0, #1

2 *ADDI X1, X1, #1*

1 ADD XO, XZR, XZR



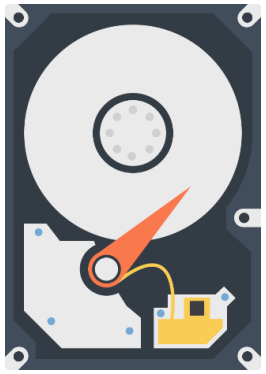
Instructions 2, 3, 4, and 5 are accessed a 100 times repeatedly.
Instructions accessed once, tend to be accessed again
Temporal Locality

Principle of Locality

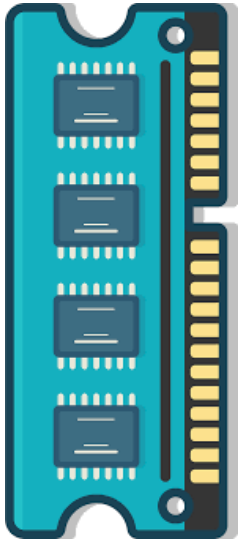
- Programs access a small proportion of their address space at any time
- Temporal locality
  - Items accessed recently are likely to be accessed again soon
  - e.g., instructions in a loop, induction variables
- Spatial locality
  - Items near those accessed recently are likely to be accessed soon
  - E.g., sequential instruction access, array data

Hierarchy of memories

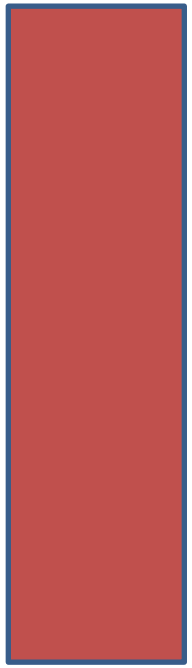
Stored in memory
HDD



RAM – Random
access memory



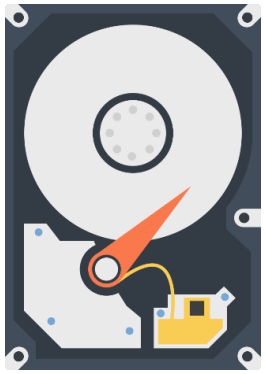
Faster memory



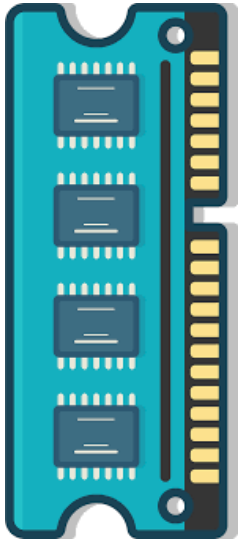
Fetch data using the principle of Locality

Hierarchy of memories

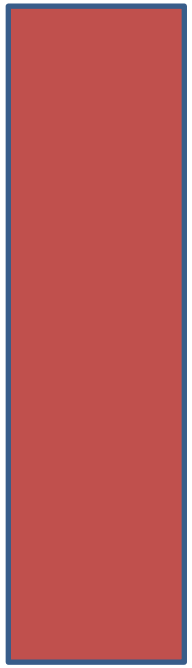
Stored in memory
HDD



RAM – Random access memory
D(ynamic)RAM



Faster memory
S(tatic)RAM



Fetch data using the principle of Locality

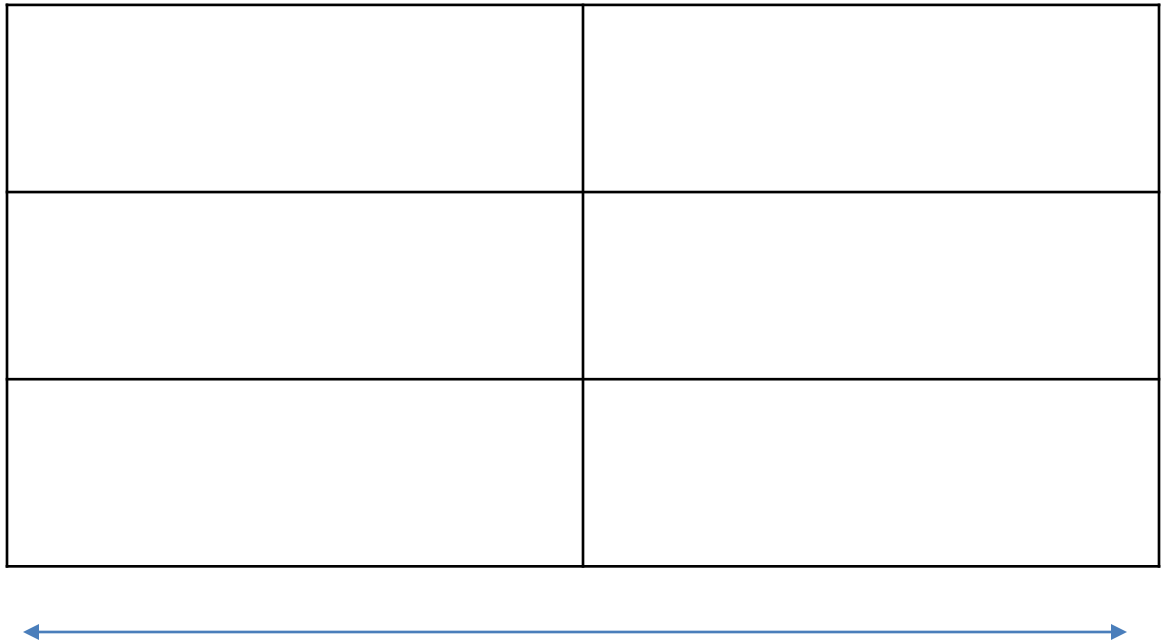
Taking Advantage of Locality

- Memory hierarchy
- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
  - Main memory
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory
  - Cache memory attached to CPU

Memory Hierarchy Levels

```
ADDR
    7 STUR      X5, [X0,#32]
    6 ADD       X5, X1, X4
    5 LDUR      X4, [X0,#16]
    4 STUR      X3, [X0,#24]
    3 ADD       X3, X1, X2
    2 LDUR      X2, [X0,#8]
    1 LDUR      X1, [X0,#0]
```

DRAM



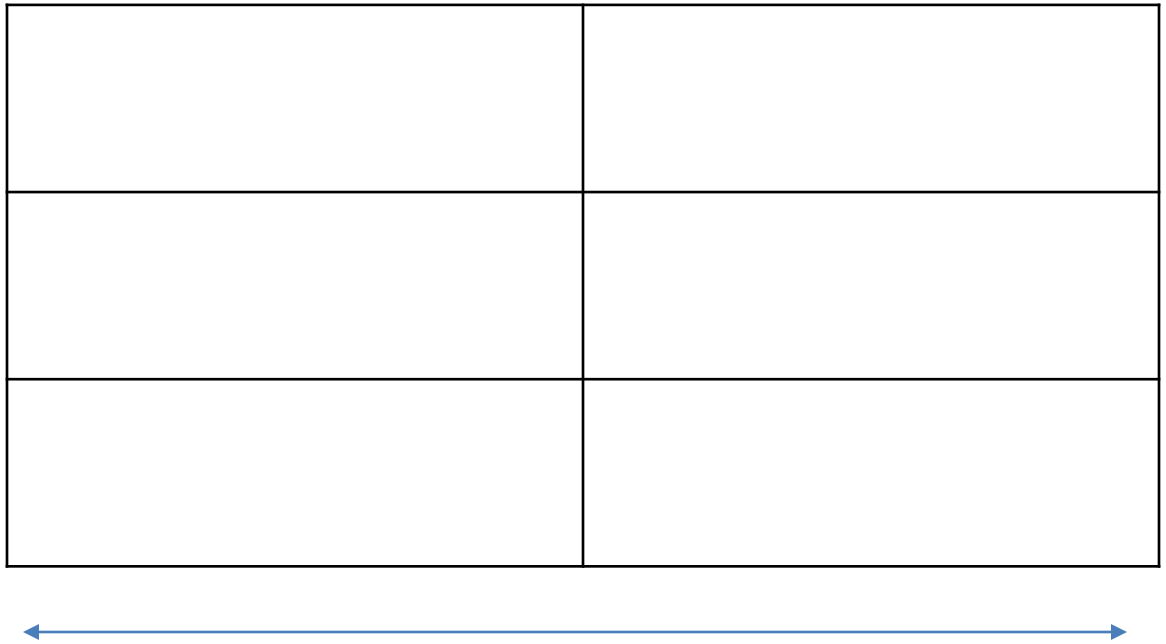
64 Bits (2 Words)

Memory Hierarchy Levels

ADDR

```
7 STUR    X5, [X0,#32]
6 ADD     X5, X1, X4
5 LDUR    X4, [X0,#16]
4 STUR    X3, [X0,#24]
3 ADD     X3, X1, X2
2 LDUR    X2, [X0,#8]
1 LDUR    X1, [X0,#0]
```

DRAM



64 Bits (2 Words)

Request for address 1
Address 1 is not available in SRAM.
This request is called a **Miss**

Memory Hierarchy Levels

ADDR

```
7 STUR    X5, [X0,#32]
6 ADD     X5, X1, X4
5 LDUR    X4, [X0,#16]
4 STUR    X3, [X0,#24]
3 ADD     X3, X1, X2
2 LDUR    X2, [X0,#8]
1 LDUR    X1, [X0,#0]
```

Spatial locality

DRAM

Request for address 1

Memory Hierarchy Levels

ADDR

```
7 STUR    X5, [X0,#32]
6 ADD     X5, X1, X4
5 LDUR    X4, [X0,#16]
4 STUR    X3, [X0,#24]
3 ADD     X3, X1, X2
2 LDUR    X2, [X0,#8]
1 LDUR    X1, [X0,#0]
```

DRAM

Spatial locality

LDUR X1, [X0,#0]	LDUR X2, [X0,#8]

64 Bits (2 Words)

Request for address 1

Memory Hierarchy Levels

ADDR

```
7 STUR    X5, [X0,#32]
6 ADD     X5, X1, X4
5 LDUR    X4, [X0,#16]
4 STUR    X3, [X0,#24]
3 ADD     X3, X1, X2
2 LDUR    X2, [X0,#8]
1 LDUR    X1, [X0,#0]
```

DRAM

Spatial locality

LDUR X1, [X0,#0]	LDUR X2, [X0,#8]

64 Bits (2 Words)

Request for address 1

Minimum information that can be saved to SRAM or removed from SRAM, called **block or line**

Each block can be indexed or addressed.

Memory Hierarchy Levels

ADDR

```
7 STUR    X5, [X0,#32]
6 ADD     X5, X1, X4
5 LDUR    X4, [X0,#16]
4 STUR    X3, [X0,#24]
3 ADD     X3, X1, X2
2 LDUR    X2, [X0,#8]
1 LDUR    X1, [X0,#0]
```

DRAM

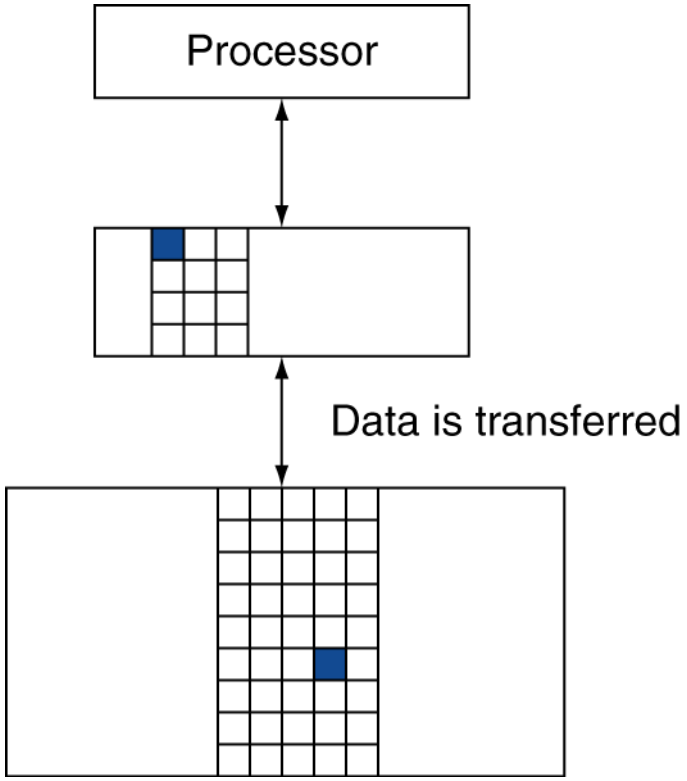
Spatial locality

LDUR X1, [X0,#0]	LDUR X2, [X0,#8]

64 Bits (2 Words)

Request for address 2
Already available in SRAM
This request is called a **HIT**

Memory Hierarchy Levels



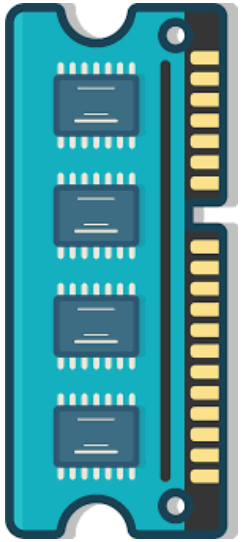
- Block (aka line): unit of copying
  - May be multiple words
- If accessed data is present in upper level
  - Hit: access satisfied by upper level
    - Hit ratio: hits/accesses
- If accessed data is absent
  - Miss: block copied from lower level
    - Time taken: miss penalty
    - Miss ratio: misses/accesses
      = 1 – hit ratio
  - Then accessed data supplied from upper level

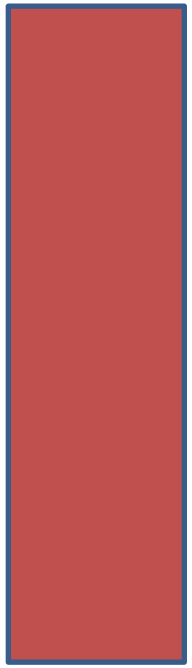
Memory Technologies

DRAM

SRAM

Stored in memory
HDD





Memories closer to processor are faster but smaller in size and more expensive

Memory Technology

- Magnetic disk
  - 5ms – 20ms, $0.01 – $0.02 per GB

Memory Technology

- Magnetic disk
  - 5ms – 20ms, $0.01 – $0.02 per GB
- Dynamic RAM (DRAM)
  - 50ns – 70ns, $3 – $6 per GB

Memories closer to processor are
faster but smaller in size and more expensive

Memory Technology

- Magnetic disk
  - 5ms – 20ms, $0.01 – $0.02 per GB
- Dynamic RAM (DRAM)
  - 50ns – 70ns, $3 – $6 per GB
- Static RAM (SRAM)
  - 0.5ns – 2.5ns, $500 – $1000 per GB

Memories closer to processor are
faster but smaller in size and more expensive

Memory Technology

- Magnetic disk
  - 5ms – 20ms, $0.01 – $0.02 per GB
- Dynamic RAM (DRAM)
  - 50ns – 70ns, $3 – $6 per GB
- Static RAM (SRAM)
  - 0.5ns – 2.5ns, $500 – $1000 per GB
- Ideal memory
  - Access time of SRAM
  - Capacity and cost/GB of disk

Memory Technology

- Magnetic disk
  - 5ms – 20ms, $0.01 – $0.02 per GB
- Dynamic RAM (DRAM)
  - 50ns – 70ns, $3 – $6 per GB
- Static RAM (SRAM)
  - 0.5ns – 2.5ns, $500 – $1000 per GB
- Ideal memory
  - Access time of SRAM
  - Capacity and cost/GB of disk

Volatile Memory:
Data and instructions that computer needs in real time.
Data lost once the power to the computer is turned off

Memory Technology

- Magnetic disk
  - 5ms – 20ms, $0.01 – $0.02 per GB
- Dynamic RAM (DRAM)
  - 50ns – 70ns, $3 – $6 per GB
- Static RAM (SRAM)
  - 0.5ns – 2.5ns, $500 – $1000 per GB
- Ideal memory
  - Access time of SRAM
  - Capacity and cost/GB of disk

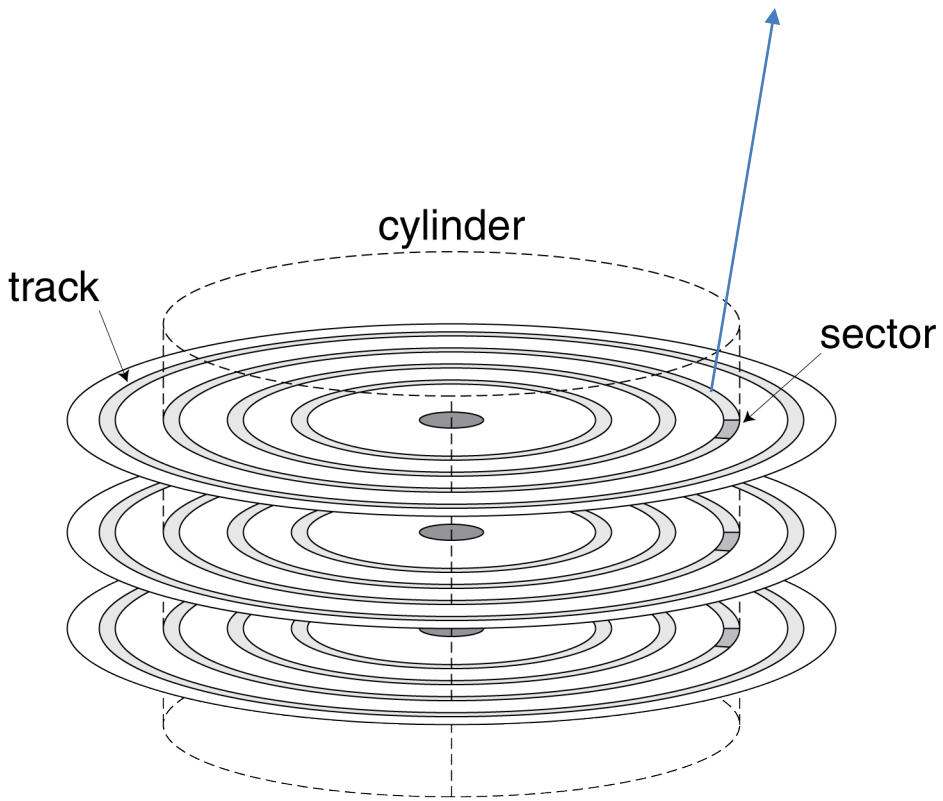
Non- Volatile Memory:
Contains all data and information
Data is not lost when the power to the computer is turned off

Disk Storage

- Nonvolatile, rotating magnetic storage

Disk Storage

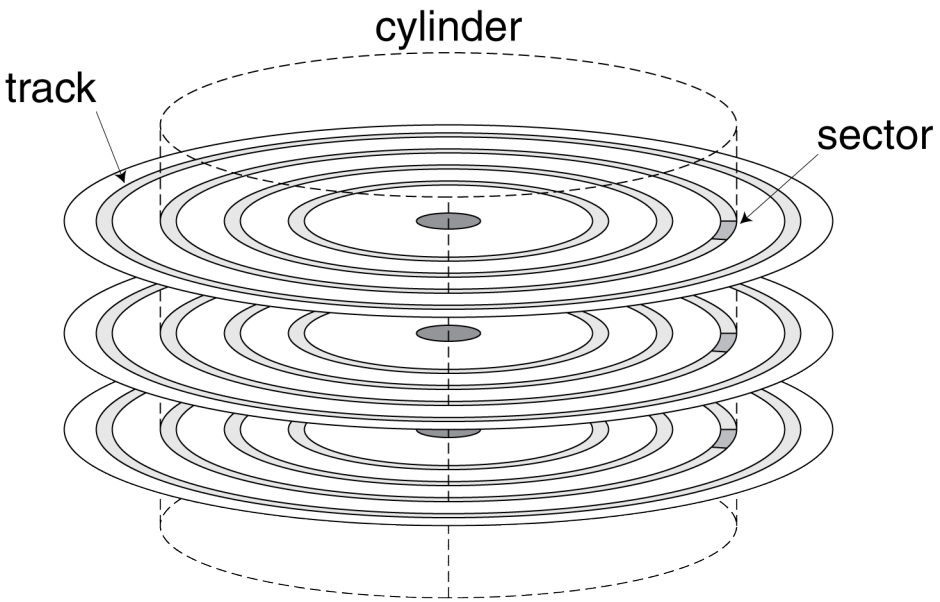
- Nonvolatile, rotating magnetic storage



Disk Storage

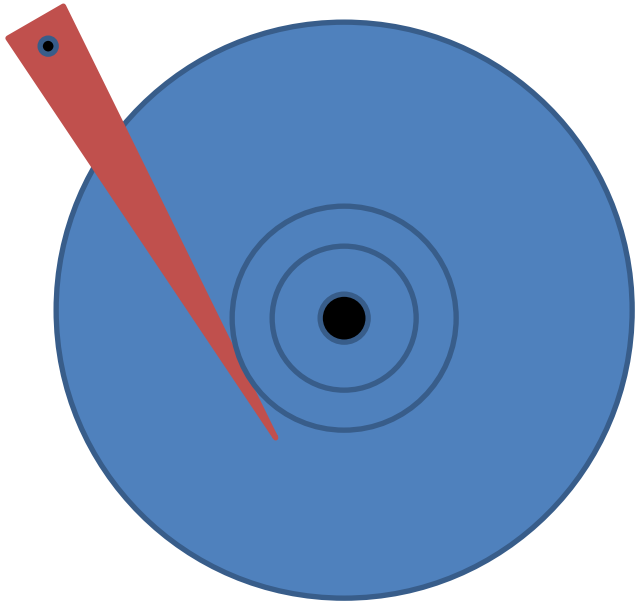
- Nonvolatile, rotating magnetic storage

Head: To read and write data



Disk Sectors and Access

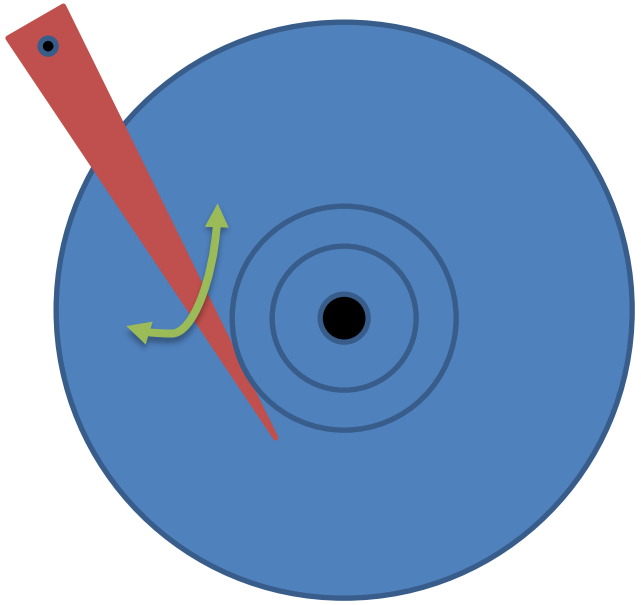
To access data:



Disk Sectors and Access

To access data:

Move head to correct track



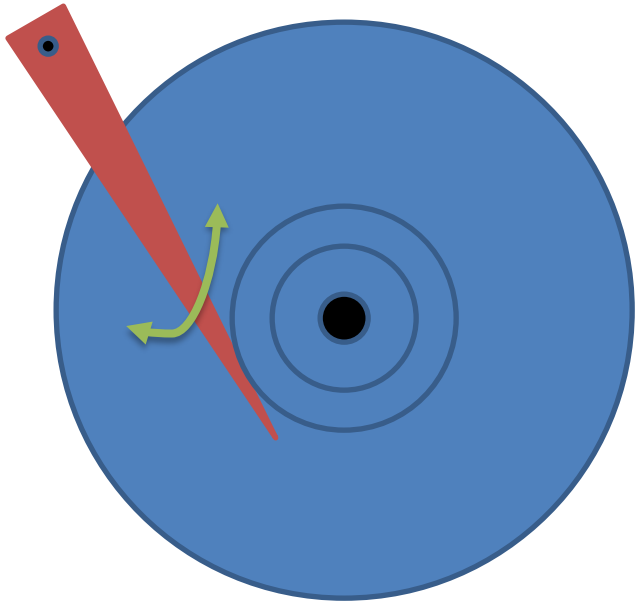
Disk Sectors and Access

To access data:

Move head to correct track (**Seek**)

Rotate disk until the required sector is under the head

Read data



Disk Sectors and Access

- Access to a sector involves
  - Queuing delay if other accesses are pending
  - Seek Time: time to move the heads
  - Rotational latency: time to rotate the disk
  - Data transfer: Time to read/write and transfer
  - Controller overhead: Other overhead

Disk Access Example

- Given
  - 512B sector, 5400 RPM (rotations per minute), 4ms average seek time, 100MB/s transfer rate, 0.2ms controller overhead, idle disk

Disk Access Example

- Given
  - 5400 RPM (rotations per minute), 4ms average seek time, 100MB/s transfer rate, 0.2ms controller overhead, idle disk
- Average rotational latency time

  5400 RPM = 5400/60 RPS

  time for 1 rotation = 1/(5400/60) s

  time for 0.5 rotation

0.5 rotation/ (5400/60) = 0.0056s = 5.6ms rotational latency

Disk Access Example

- Given
  - 512B sector, 5400 RPM (rotations per minute), 4ms average seek time, 100MB/s transfer rate, 0.2ms controller overhead, idle disk
- Average rotational latency time

0.5 rotation/ (5400/60) = 0.0056s = 5.6ms rotational latency

- Average read time
  - 4ms seek time
    + ½ / (5400/60) = 5.6ms rotational latency
    + 512 / 100MB/s = 0.005ms transfer time
    + 0.2ms controller delay
    = 9.805ms

Disk Access Example

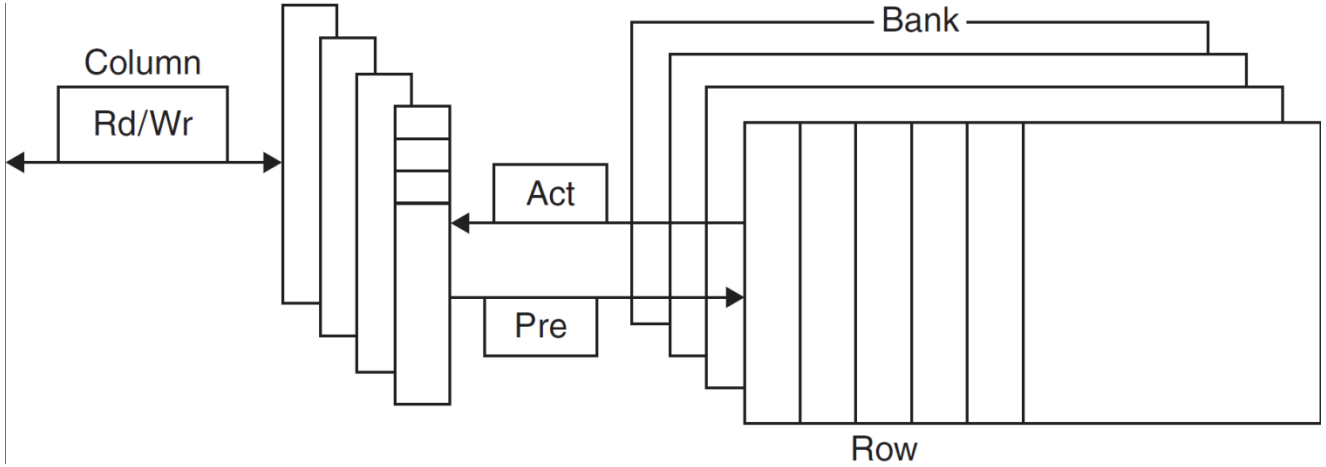
- Given
  - 512B sector, 5400 RPM (rotations per minute), 4ms average seek time, 100MB/s transfer rate, 0.2ms controller overhead, idle disk
- Average rotational latency time

0.5 rotation/ (5400/60) = 0.0056s = 5.6ms rotational latency

- Average read time
  - 4ms seek time
    + ½ / (5400/60) = 5.6ms rotational latency (15000 RMP 2ms)
    + 512 / 100MB/s = 0.005ms transfer time
    + 0.2ms controller delay
    = 9.805ms

D(ynamic)RAM Technology

- Data stored as a charge in a capacitor
  - Single transistor used to access the charge
  - Must periodically be refreshed
    - Read contents and write back
    - Performed on a DRAM “row”

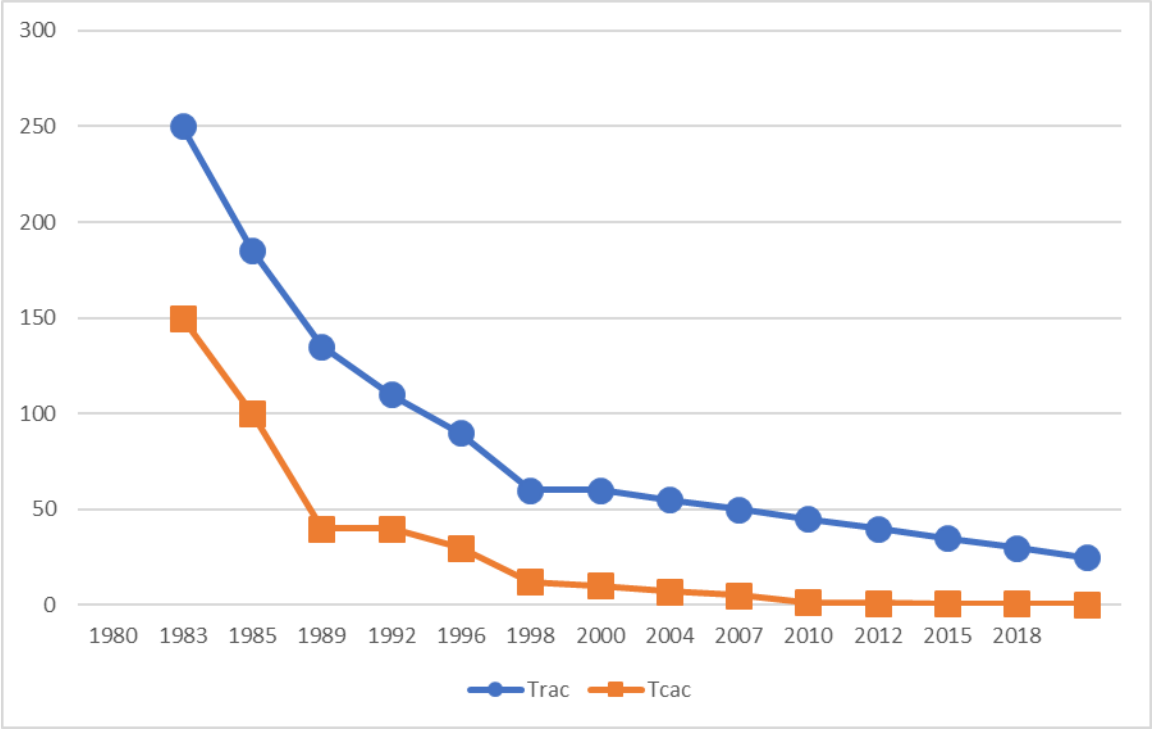


Advanced DRAM Organization

- Bits in a DRAM are organized as a rectangular array
  - DRAM accesses an entire row
- Double data rate (DDR) DRAM
  - Transfer on rising and falling clock edges
- Quad data rate (QDR) DRAM
  - Separate DDR inputs and outputs

DRAM Generations

Year	Capacity	$/GB
1980	64 Kibibit	$6,480,000
1983	256 Kibibit	$1,980,000
1985	1 Mebibit	$720,000
1989	4 Mebibit	$128,000
1992	16 Mebibit	$30,000
1996	64 Mebibit	$9,000
1998	128 Mebibit	$900
2000	256 Mebibit	$840
2004	512 Mebibit	$150
2007	1 Gibibit	$40
2010	2 Gibibit	$13
2012	4 Gibibit	$5
2015	8 Gibibit	$7
2018	16 Gibibit	$6



S(tatic)RAM Technology

- *Static Random Access Memory (SRAM)*
  - Requires low power to retain bit
  - Fixed access time to data
  - No need to refresh (unlike DRAM)
  - Requires 6 transistors/bit
    - Expensive

Taking Advantage of Locality

- Memory hierarchy
- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
  - Main memory
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory
  - **Cache memory** attached to CPU

Cache Memory

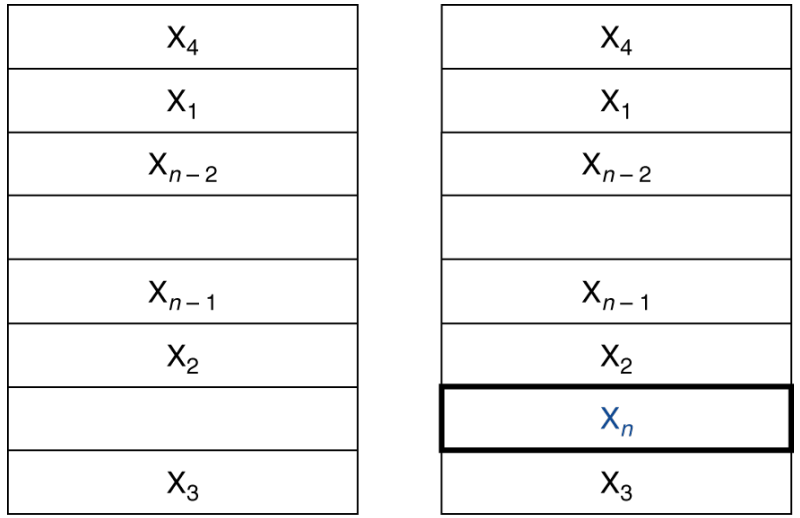
- Cache memory
  - The level of the memory hierarchy closest to the CPU
- Given accesses $X_1, \ldots, X_{n-1}, X_n$

X_4
X_1
X_{n-2}
X_{n-1}
X_2
X_3

a. Before the reference to X_n

Cache Memory

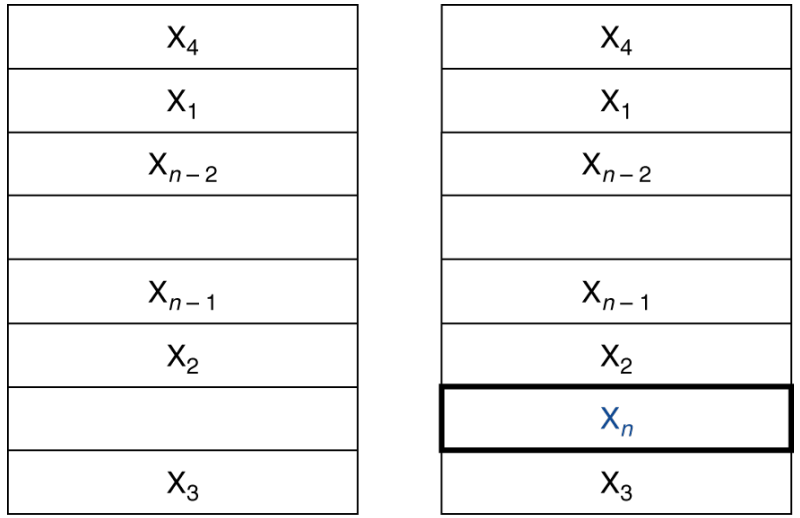
- Cache memory
  - The level of the memory hierarchy closest to the CPU
- Given accesses $X_1, \ldots, X_{n-1}, X_n$



a. Before the reference to X_n b. After the reference to X_n

Cache Memory

- Cache memory
  - The level of the memory hierarchy closest to the CPU
- Given accesses $X_1, \ldots, X_{n-1}, X_n$

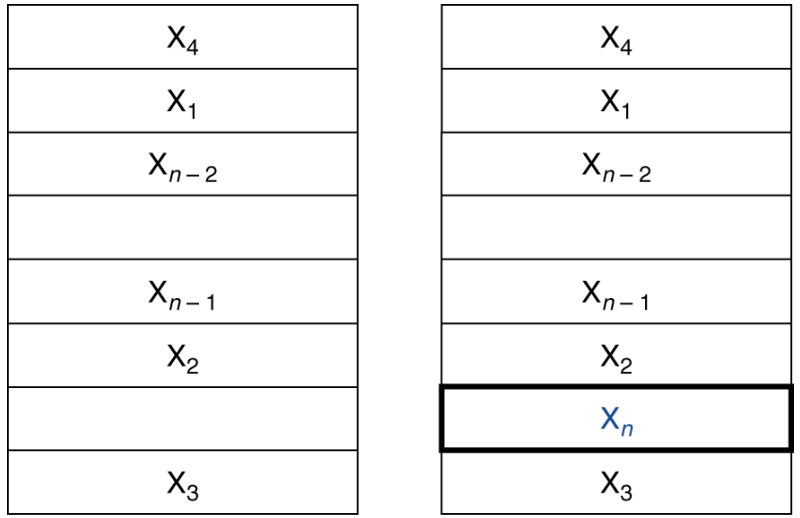


a. Before the reference to X_n b. After the reference to X_n

- How do we know if the data is present?
  - Do we search each block in SRAM?

Cache Memory

- Cache memory
  - The level of the memory hierarchy closest to the CPU
- Given accesses $X_1, \ldots, X_{n-1}, X_n$

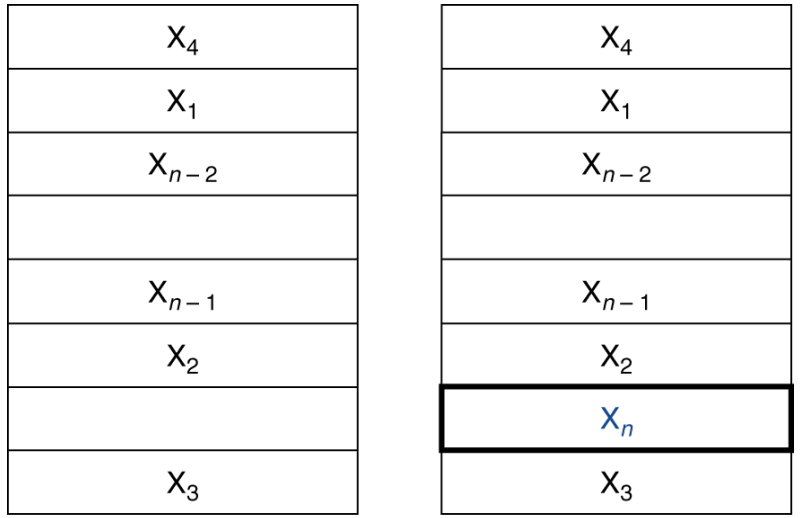


a. Before the reference to X_n b. After the reference to X_n

- How do we know if the data is present?
  - Do we search each block in SRAM?
  - Takes too long

Cache Memory

- Cache memory
  - The level of the memory hierarchy closest to the CPU
- Given accesses $X_1, \ldots, X_{n-1}, X_n$

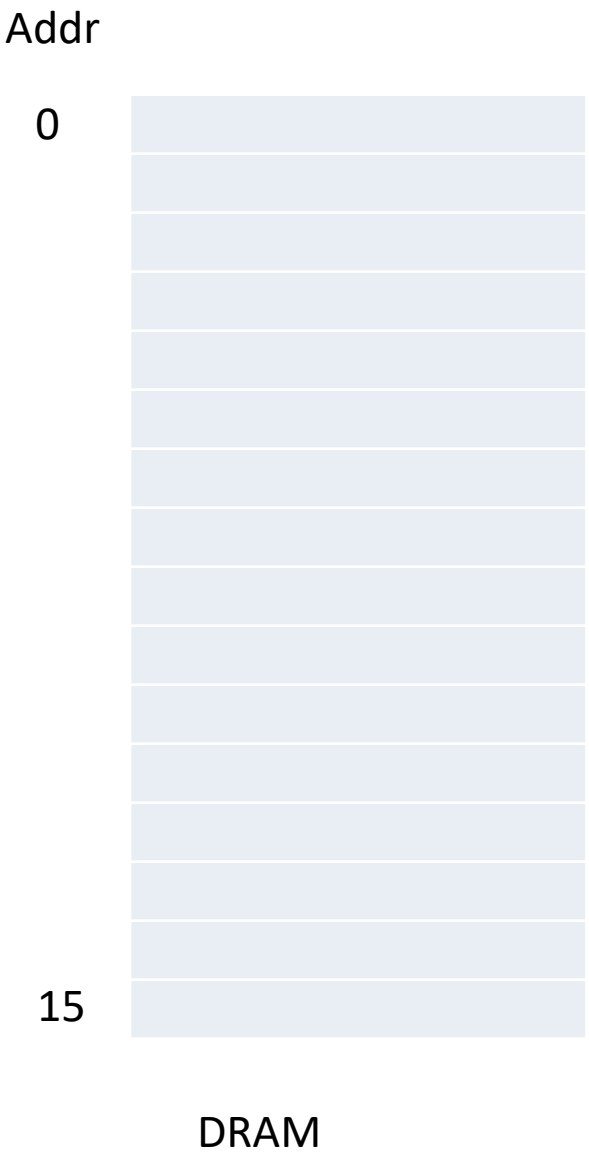


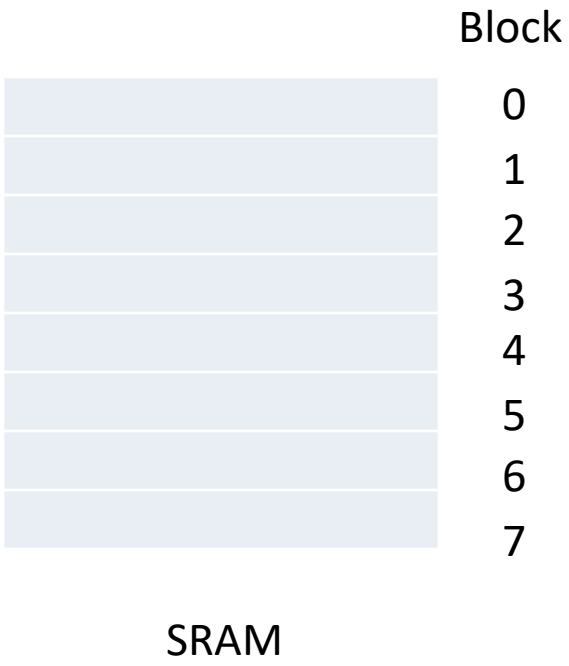
a. Before the reference to X_n b. After the reference to X_n

- Make an agreement between processor and SRAM, that if a certain address is available, it will be stored in a particular block.

Example

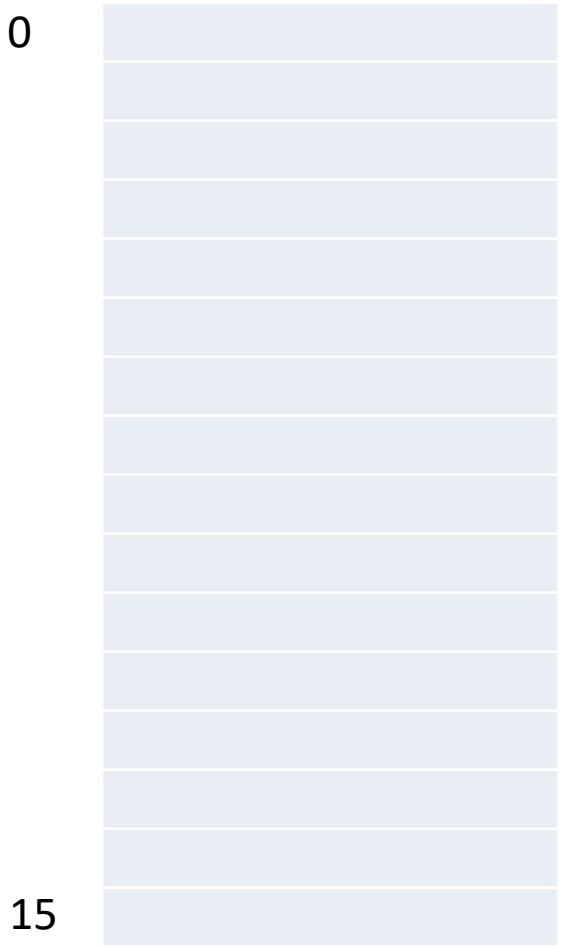
Example





Example

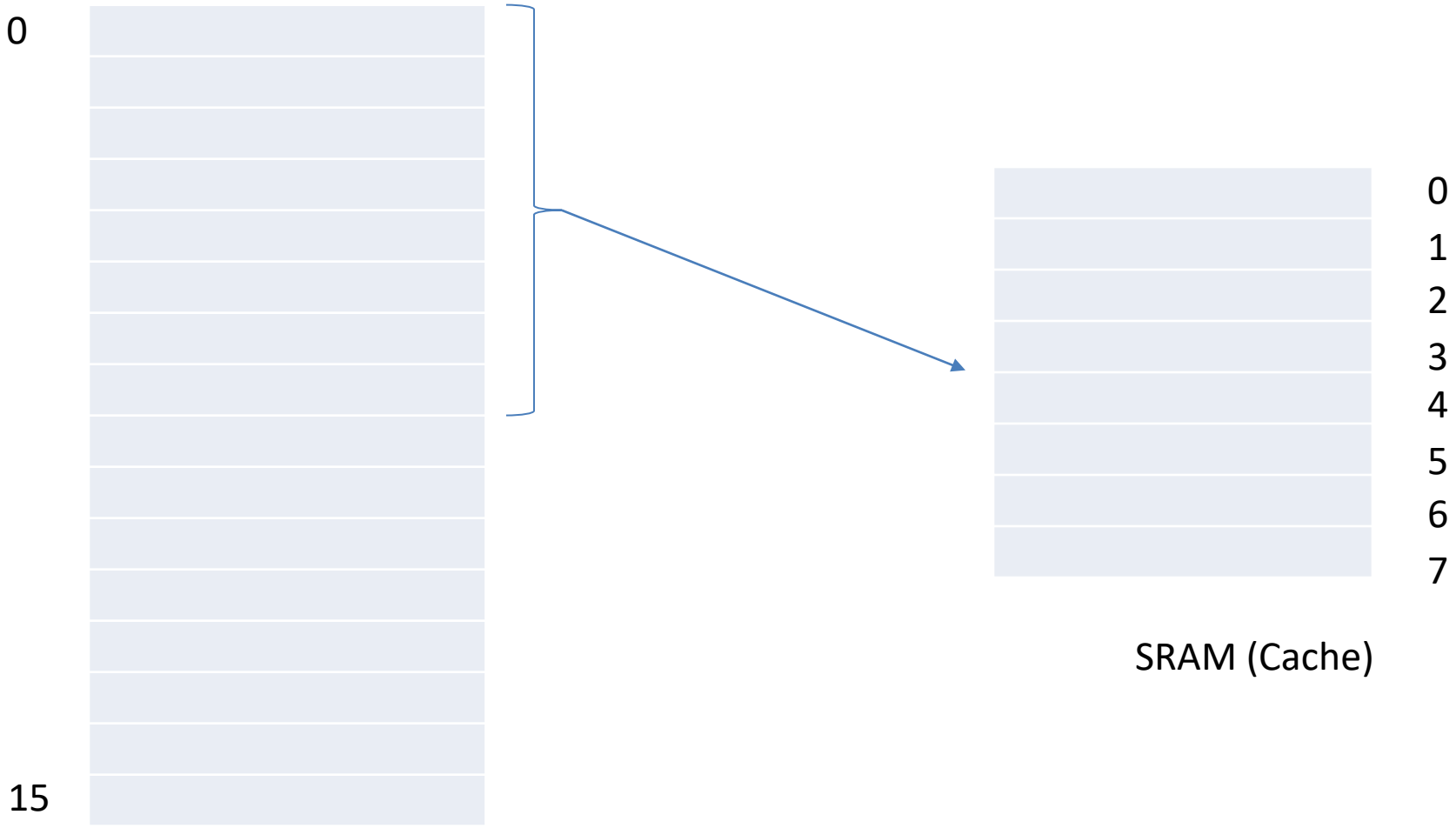
Executing instructions sequentially, no branches
Fetch 0:
Not available in cache so fetch form main memory



DRAM (Main memory)

SRAM (Cache)

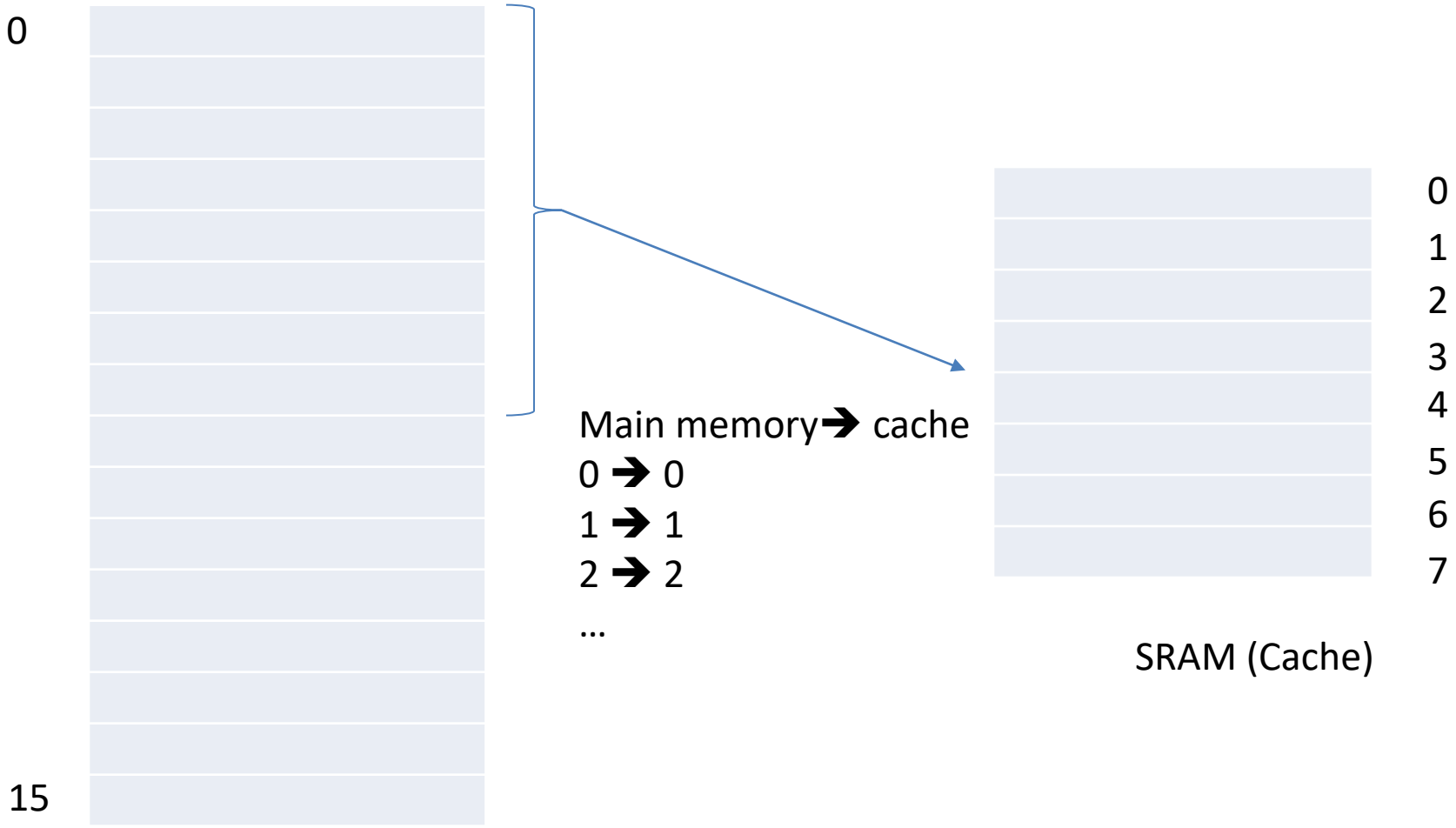
Example



DRAM (Main memory)

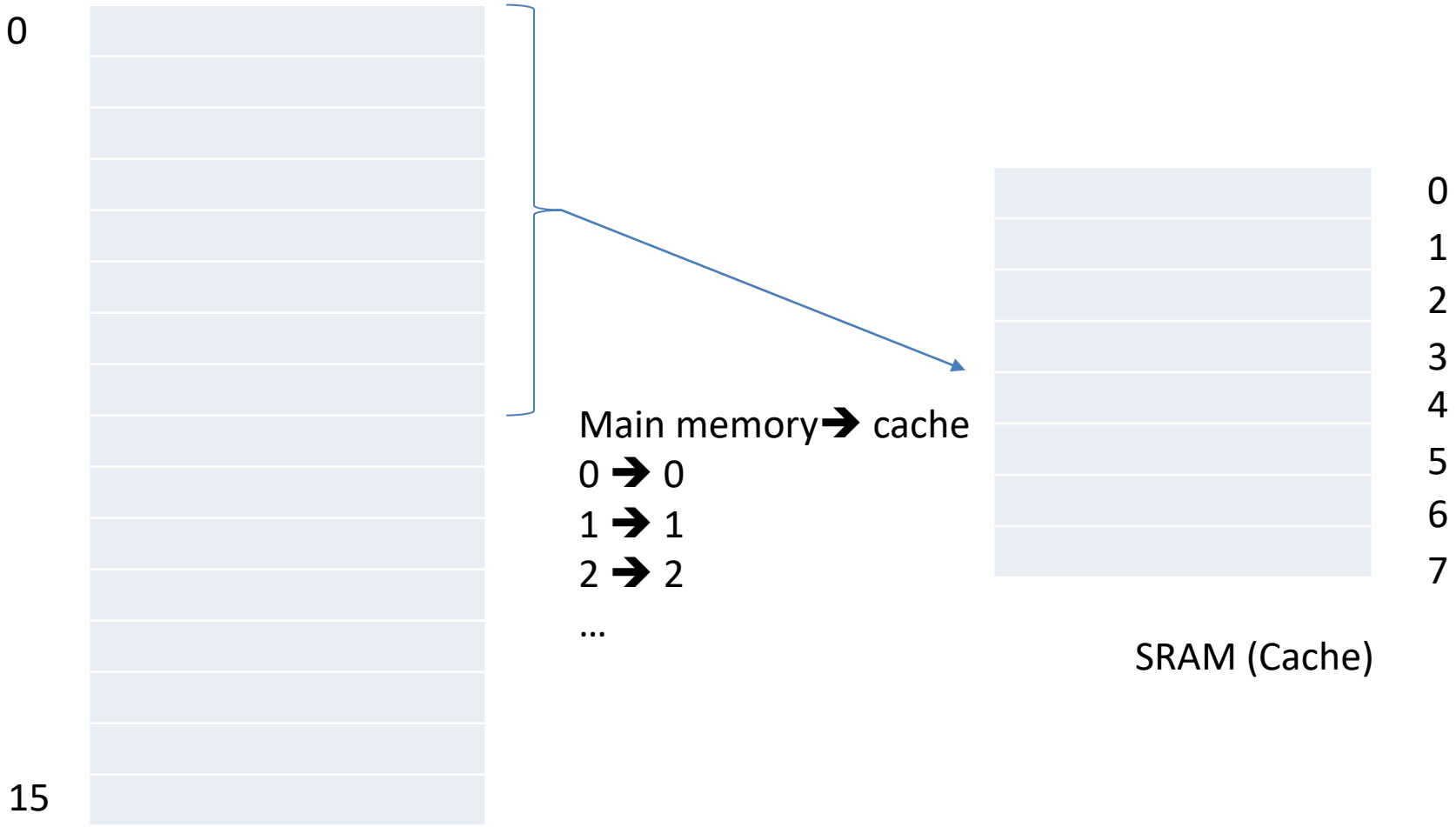
Fetch 0:
Not available in cache, so fetch form main memory
Spatial locality
Fetch next 7 as well

Example



Fetch 0:
Not available in cache, so fetch form main memory
Spatial locality
Fetch next 7 as well

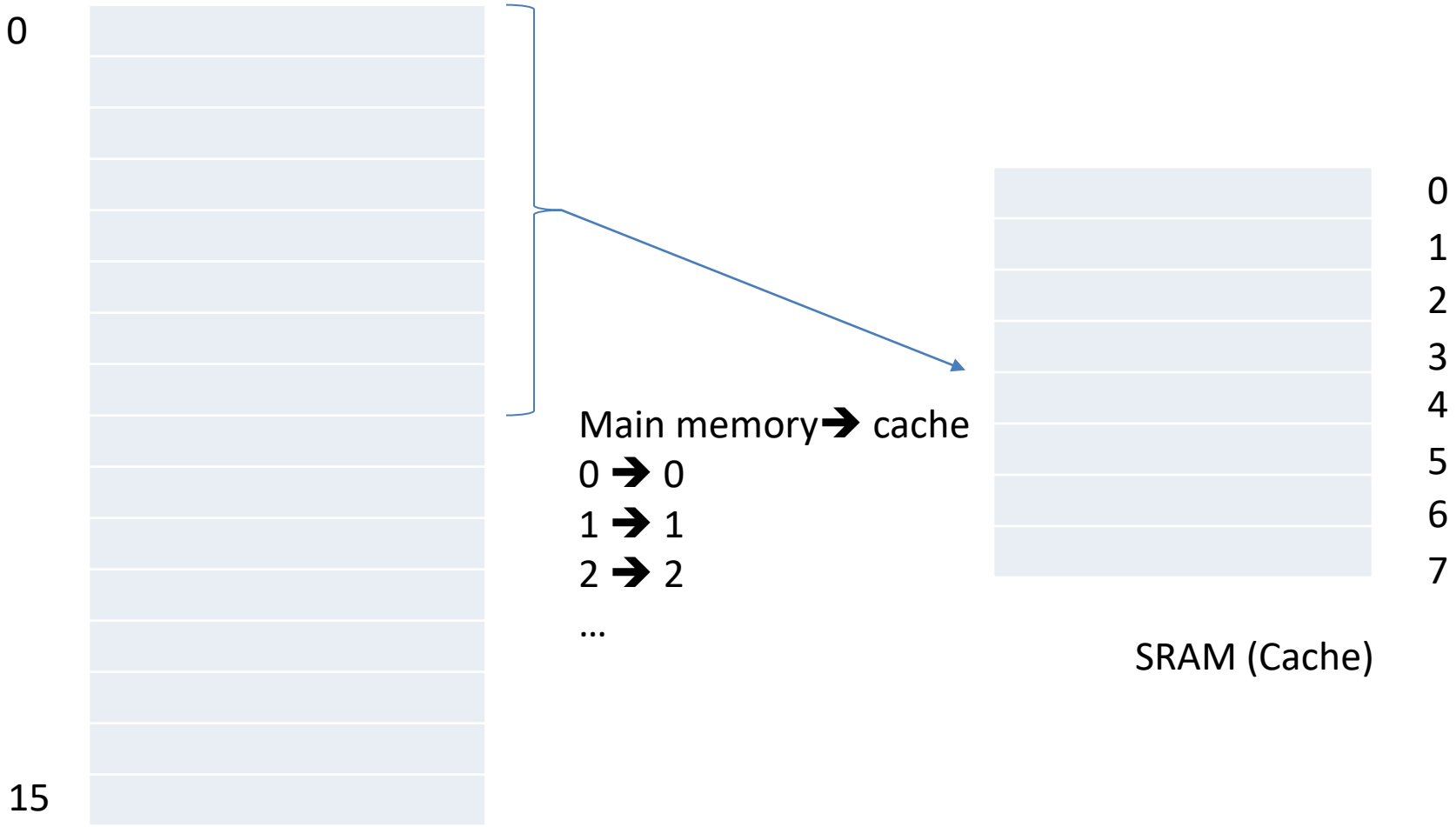
Example



DRAM (Main memory)

Fetch 8:
Not available in cache, so fetch form main memory

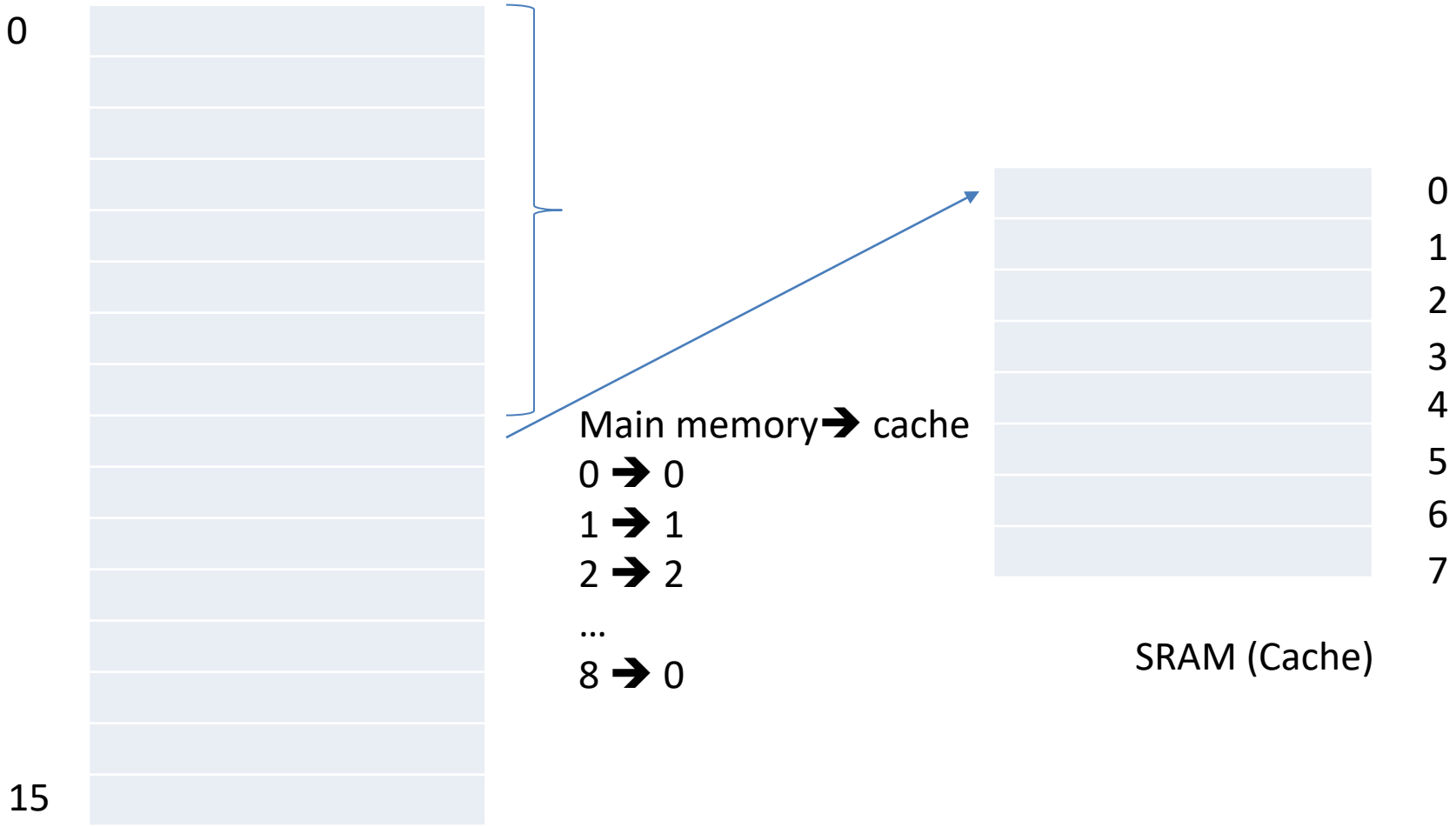
Example



DRAM (Main memory)

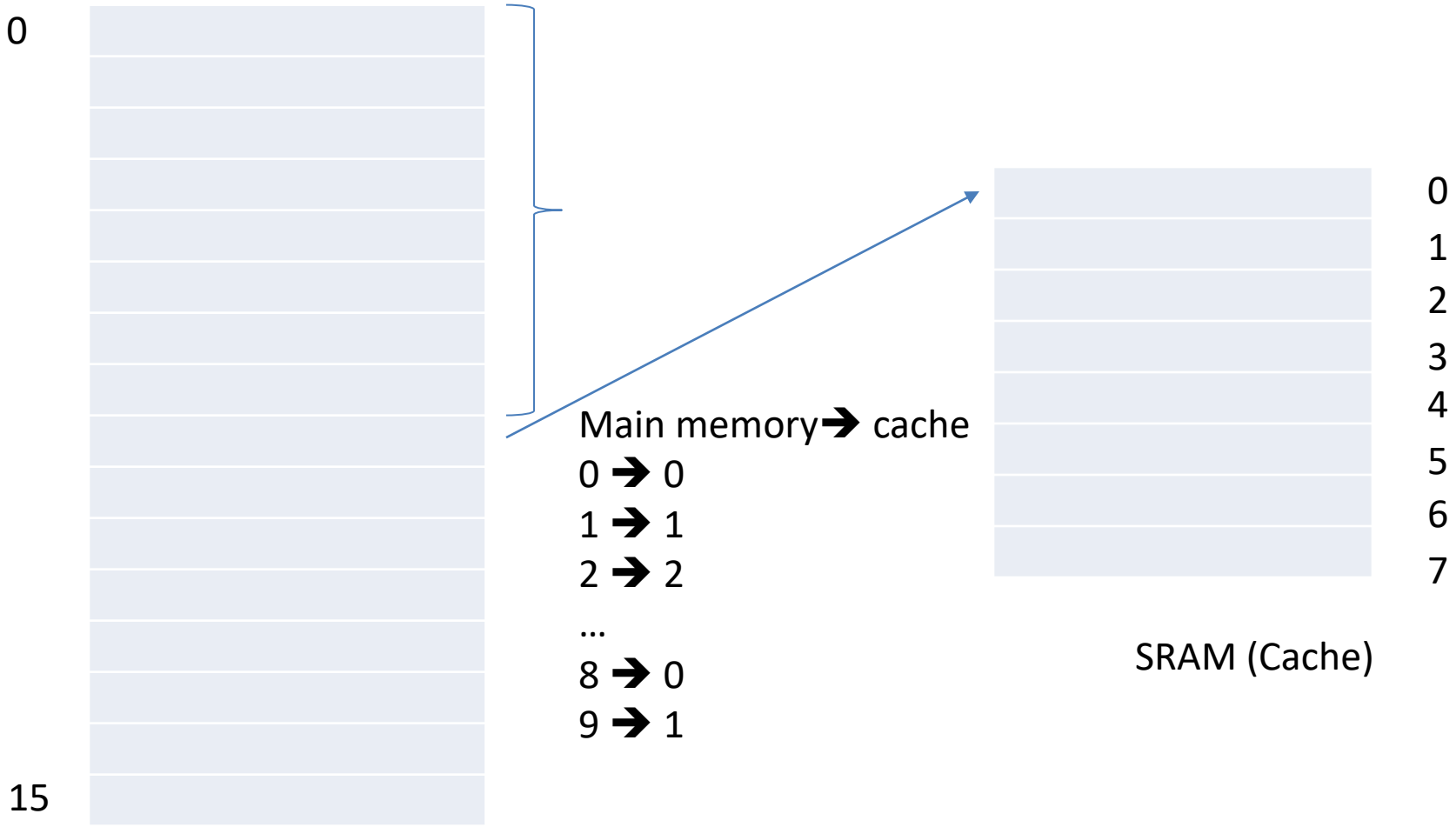
Fetch 8:
Not available in cache, so fetch form main memory
Temporal locality:
Retail the latest accessed ones and replace the oldest one

Example



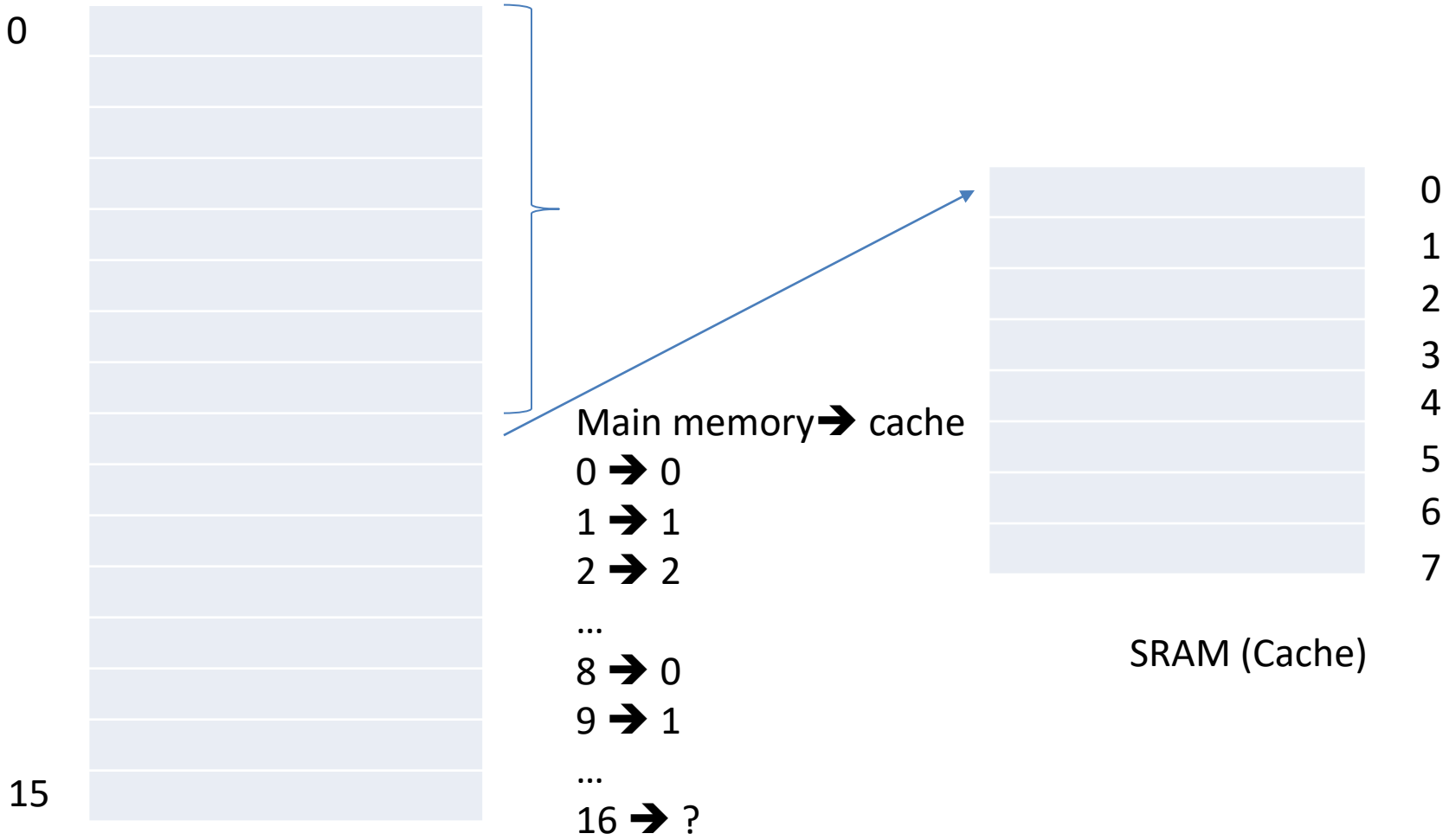
Fetch 8:
Not available in cache, so fetch form main memory
Temporal locality:
Retain the latest accessed ones and replace the oldest one

Example



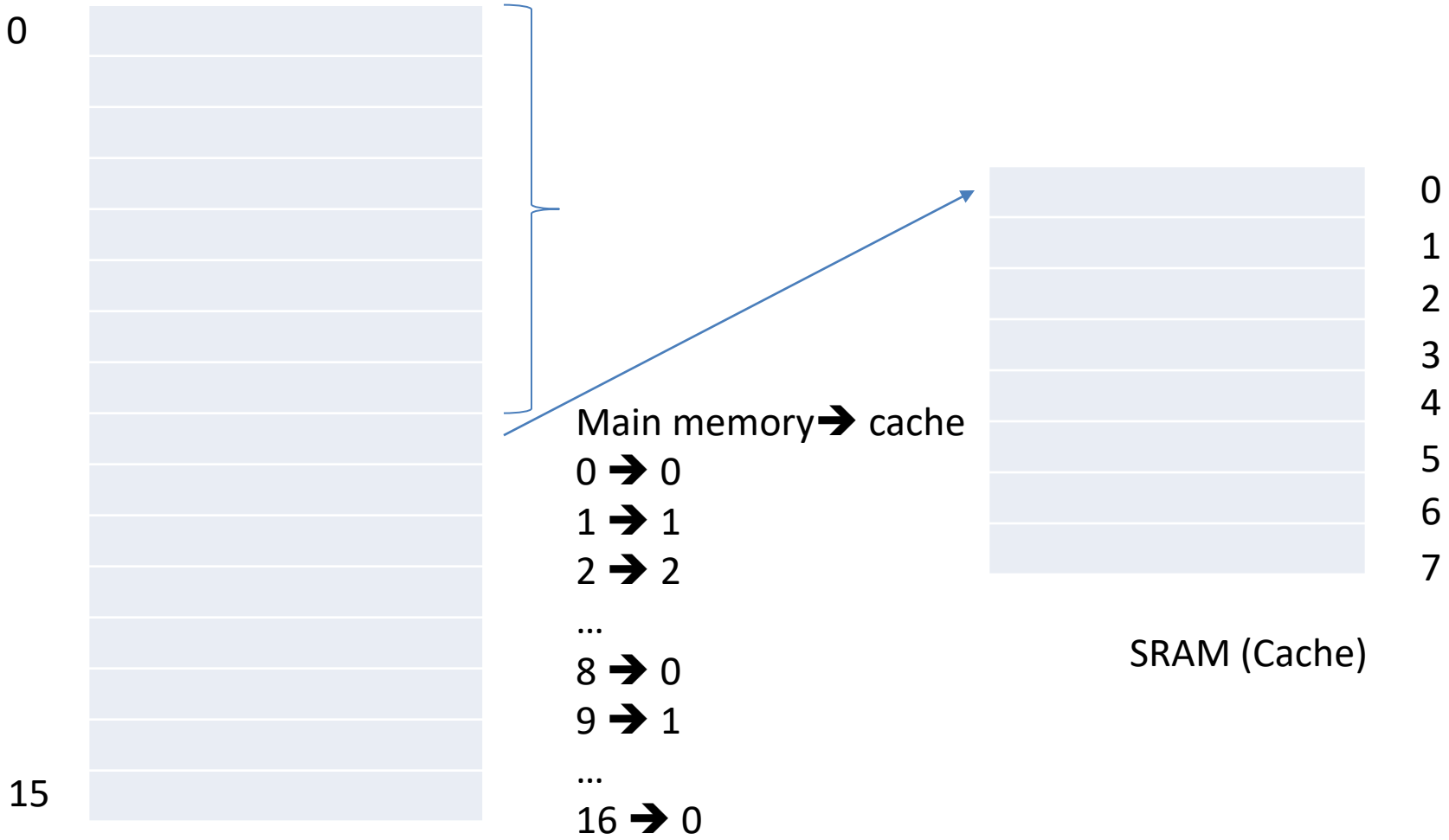
Fetch 9:
Not available in cache, so fetch form main memory
Temporal locality:
Retain the latest accessed ones and replace the oldest one

Example



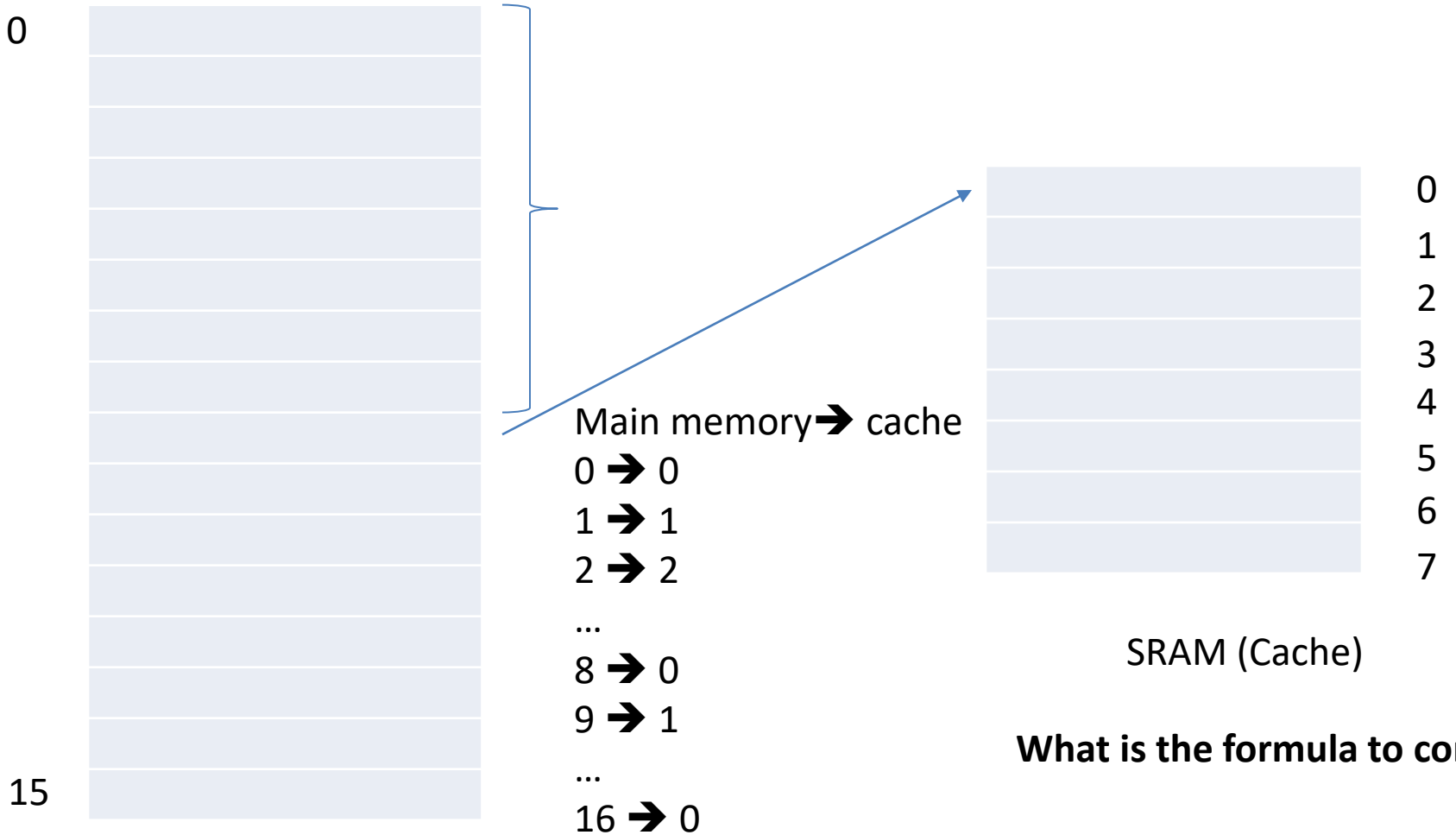
Fetch 9:
Not available in cache, so fetch form main memory
Temporal locality:
Retain the latest accessed ones and replace the oldest one

Example



Fetch 9:
Not available in cache, so fetch form main memory
Temporal locality:
Retain the latest accessed ones and replace the oldest one

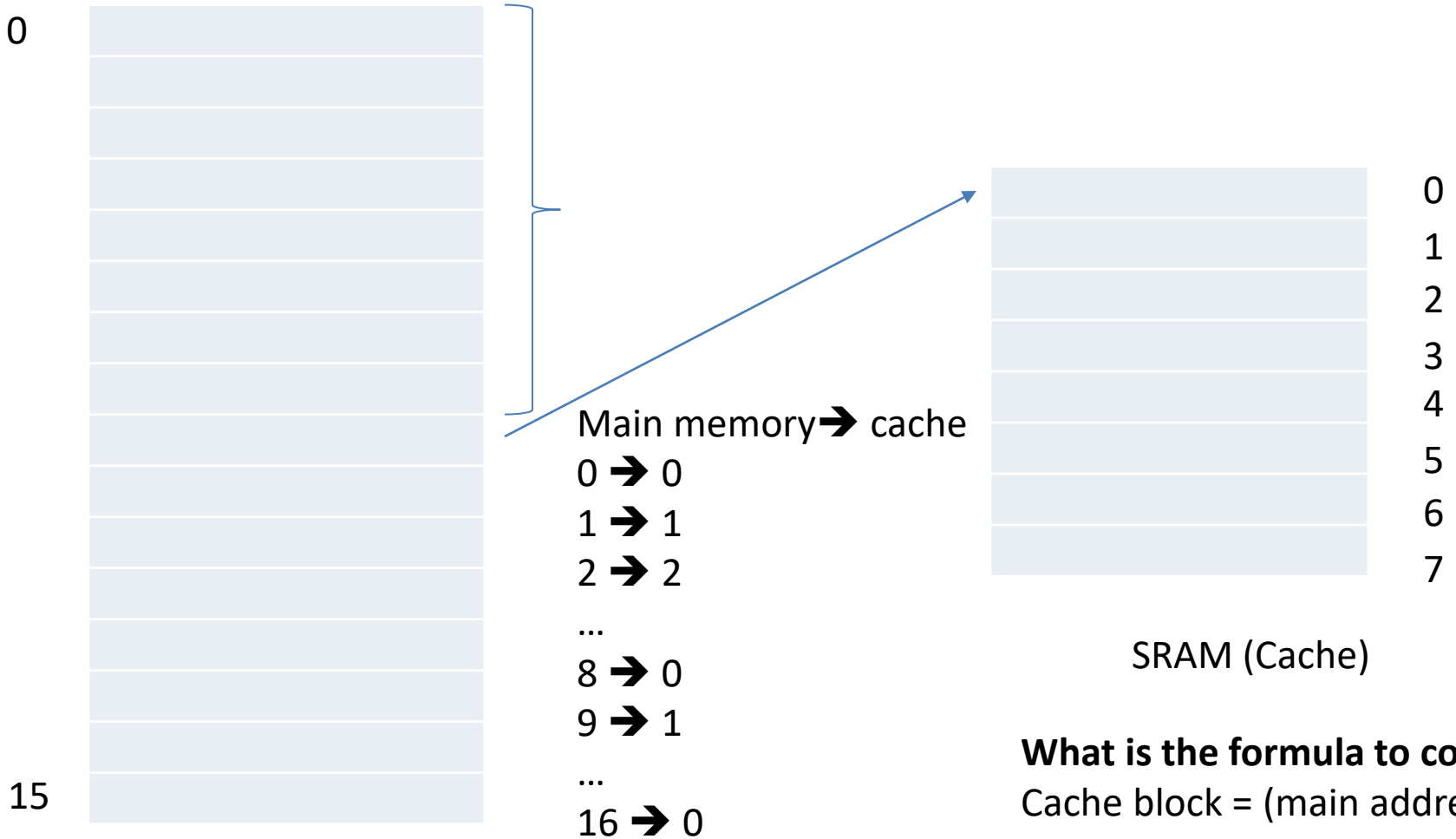
Example



Fetch 9:
Not available in cache, so fetch form main memory
Temporal locality:
Retain the latest accessed ones and replace the oldest one

What is the formula to compute mapping?

Example



DRAM (Main memory)

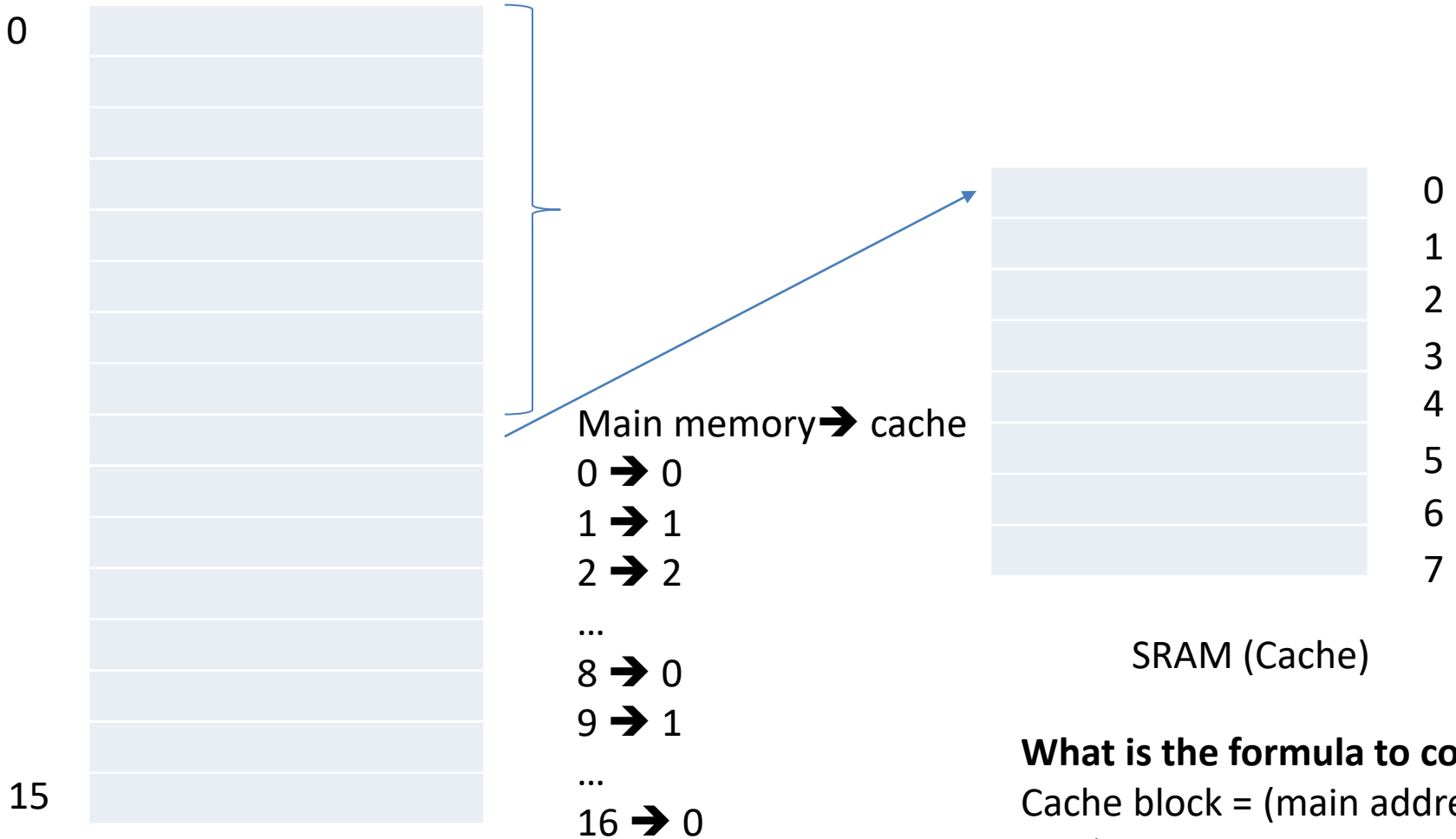
Main memory ➔ cache
0 ➔ 0
1 ➔ 1
2 ➔ 2
…
8 ➔ 0
9 ➔ 1
…
16 ➔ 0
17 ➔ 1

SRAM (Cache)

Fetch 9:
Not available in cache, so fetch form main memory
Temporal locality:
Retain the latest accessed ones and replace the oldest one

What is the formula to compute mapping?
Cache block = (main address) modulo (number of blocks)

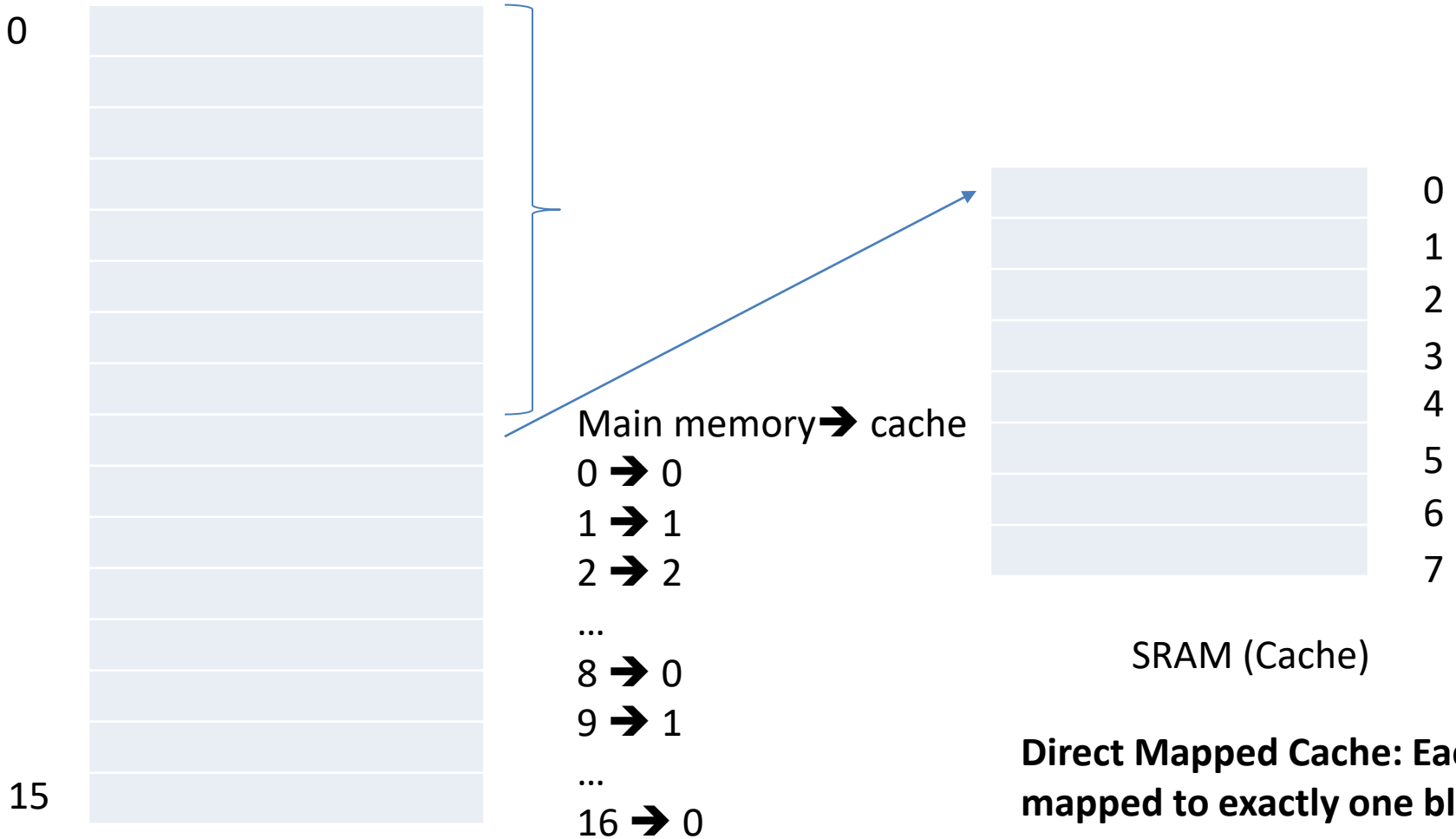
Example



Fetch 9:
Not available in cache, so fetch form main memory
Temporal locality:
Retain the latest accessed ones and replace the oldest one

What is the formula to compute mapping?
Cache block = (main address) modulo (number of blocks)
16 % 8 = 0
17 % 8 = 1

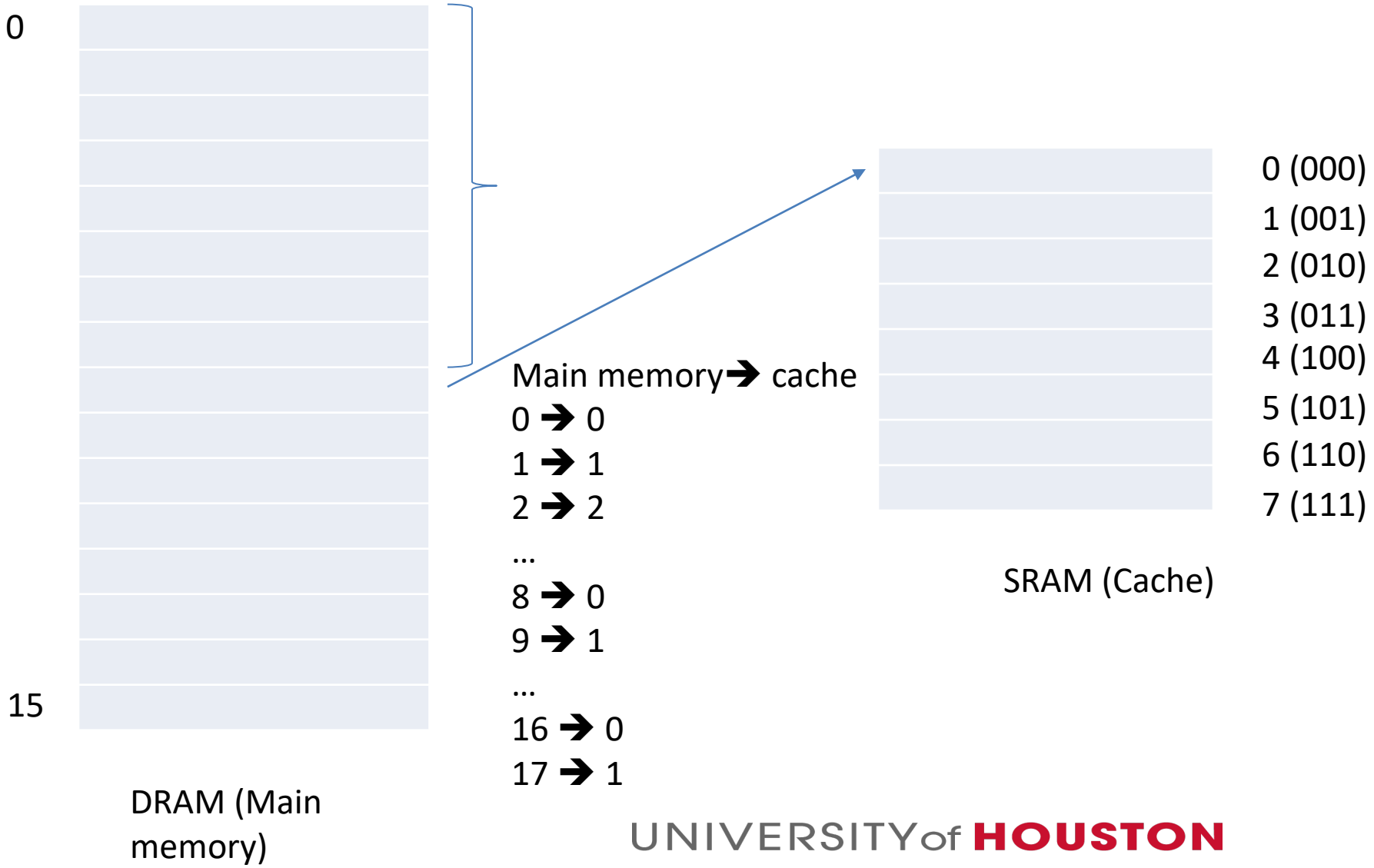
Example



Fetch 9:
Not available in cache, so fetch form main memory
Temporal locality:
Retain the latest accessed ones and replace the oldest one

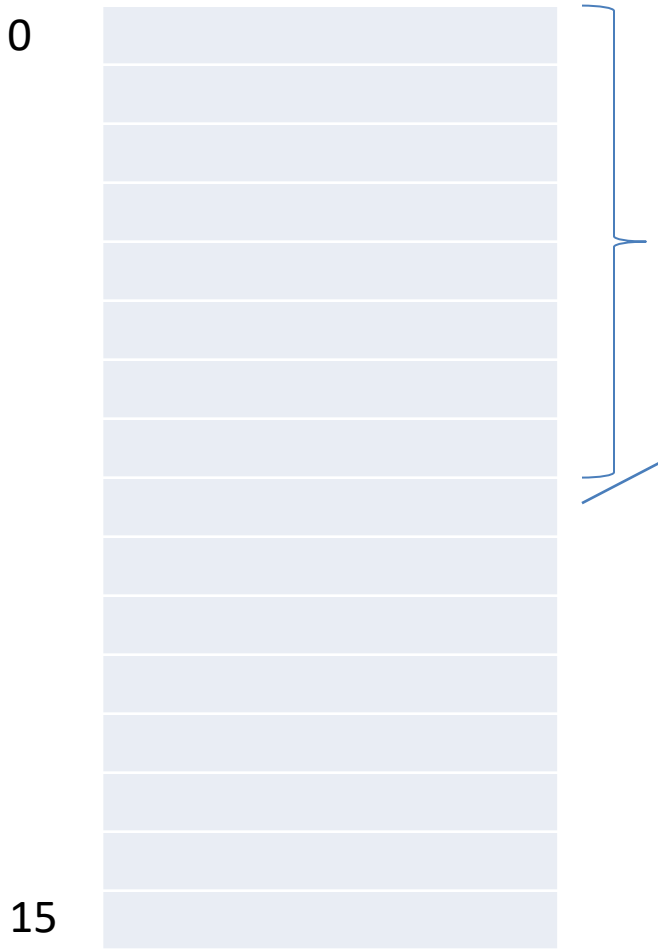
Direct Mapped Cache: Each address in main memory is mapped to exactly one block

Example



8 blocks need 3 bits to index them)

Example



DRAM (Main memory)

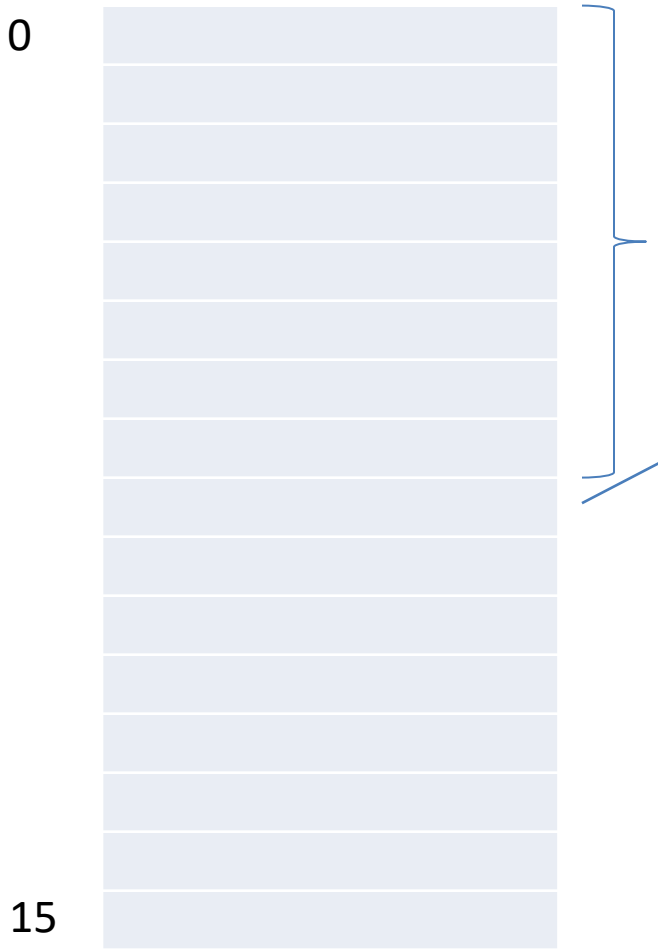
Main memory➔ cache
0 (00000) ➔ 0
1 (00001) ➔ 1
2 (00010) ➔ 2
…
8 (01000) ➔ 0
9 (01001) ➔ 1
10 (01010) ➔ 2
…
16 (10000) ➔ 0
17 (10001) ➔ 1
18 (10010) ➔ 2

SRAM (Cache)

0 (000)
1 (001)
2 (010)
3 (011)
4 (100)
5 (101)
6 (110)
7 (111)

8 blocks need 3 bits to index them)

Example



DRAM (Main memory)

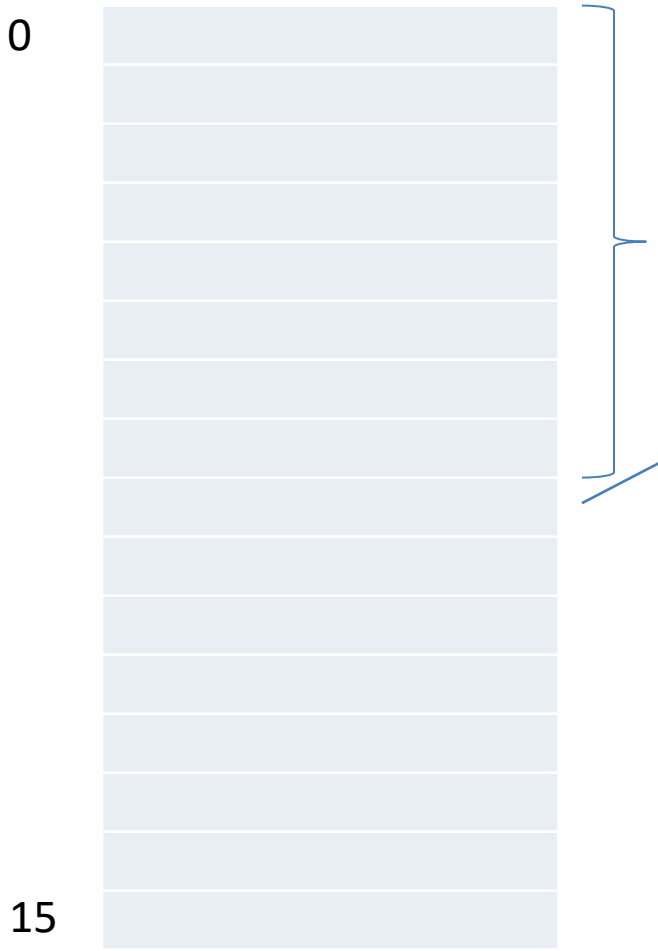
Main memory➔ cache
0 (00**000**) ➔ 0
1 (00001) ➔ 1
2 (00010) ➔ 2
…
8 (01000) ➔ 0
9 (01001) ➔ 1
10 (01010) ➔ 2
…
16 (10000) ➔ 0
17 (10001) ➔ 1
18 (10010) ➔ 2

SRAM (Cache)

0 (**000**)
1 (001)
2 (010)
3 (011)
4 (100)
5 (101)
6 (110)
7 (111)

8 blocks need 3 bits to index them)

Example



DRAM (Main memory)

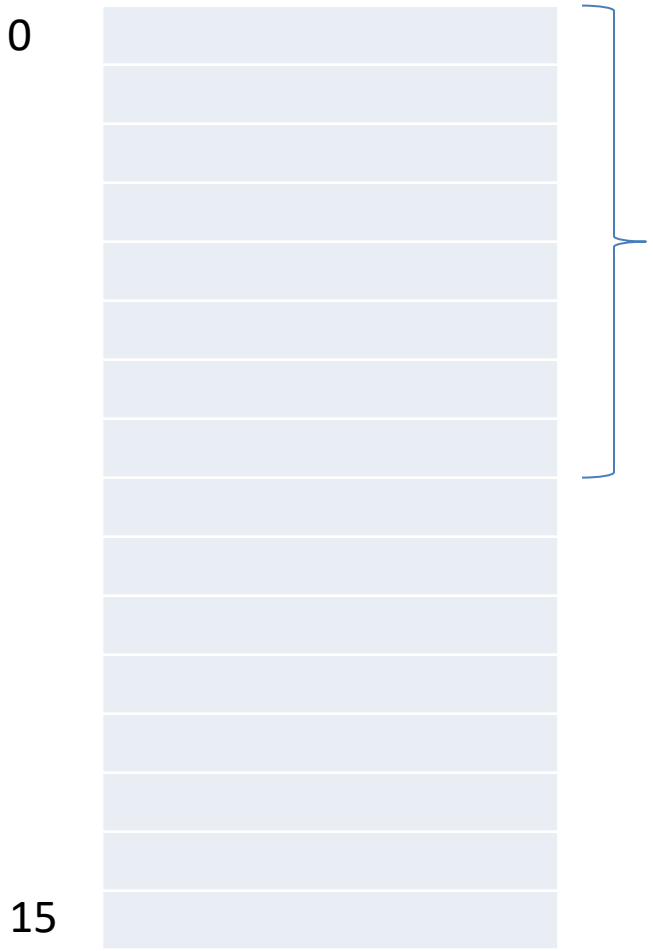
Main memory → cache
0 (00000) → 0
1 (00001) → 1
2 (00010) → 2
…
8 (01000) → 0
9 (01001) → 1
10 (01010) → 2
…
16 (10000) → 0
17 (10001) → 1
18 (10010) → 2

SRAM (Cache)

0 (000)
1 (001)
2 (010)
3 (011)
4 (100)
5 (101)
6 (110)
7 (111)

8 blocks need 3 bits to index them)

Example



DRAM (Main memory)

Main memory ➔ cache

0 (00000) ➔ 0
1 (00001) ➔ 1
2 (00010) ➔ 2
…
8 (01000) ➔ 0
9 (01001) ➔ 1
10 (01010) ➔ 2
…
16 (10000) ➔ 0
17 (10001) ➔ 1
18 (10010) ➔ 2

0 (000)
1 (001)
2 (010)
3 (011)
4 (100)
5 (101)
6 (110)
7 (111)

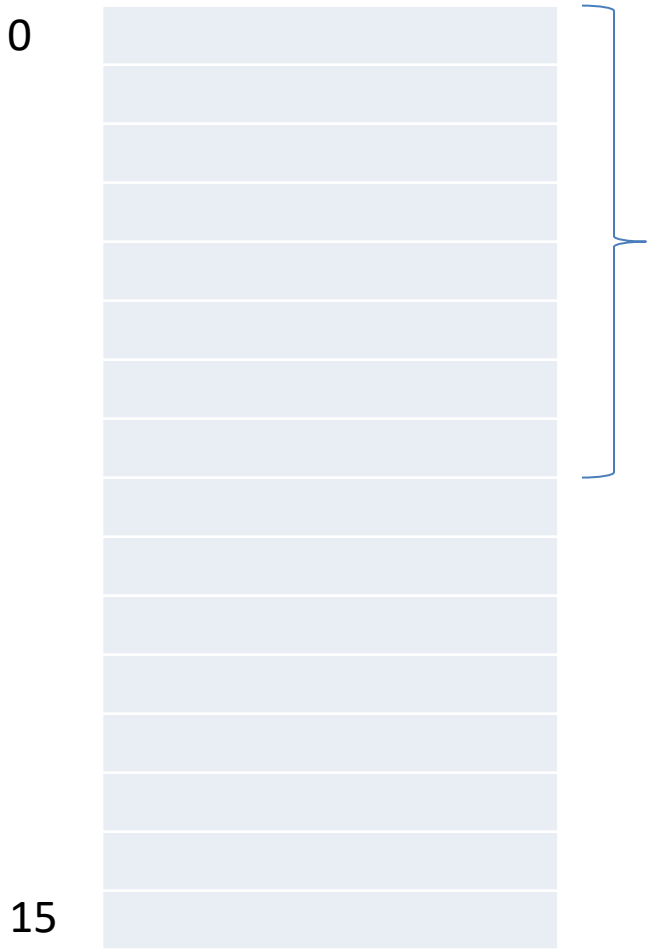
SRAM (Cache)

$2^3 = 8$ blocks
need 3 bits to index them)

Lower order

$Number\ of\ blocks\ (2^m) = 8$

Example



DRAM (Main memory)

Main memory ➔ cache

0 (00000) ➔ 0
1 (00001) ➔ 1
2 (00010) ➔ 2
…
8 (01000) ➔ 0
9 (01001) ➔ 1
10 (01010) ➔ 2
…
16 (10000) ➔ 0
17 (10001) ➔ 1
18 (10010) ➔ 2

0 (000)
1 (001)
2 (010)
3 (011)
4 (100)
5 (101)
6 (110)
7 (111)

SRAM (Cache)

$2^3 = 8$ blocks
need 3 bits to index them)

Lower order
$Number\ of\ blocks\ (2^m) = 8$
$\log_2(number\ of\ blocks) = \log_2 8 = 3$ are used as the cache index

Example

0

15

DRAM (Main memory)

Main memory➔ cache

0 (00**000**) ➔ 0
1 (00**001**) ➔ 1
2 (00**010**) ➔ 2
…
8 (01**000**) ➔ 0
9 (01**001**) ➔ 1
10 (01**010**) ➔ 2
…
16 (10**000**) ➔ 0
17 (10**001**) ➔ 1
18 (10**010**) ➔ 2

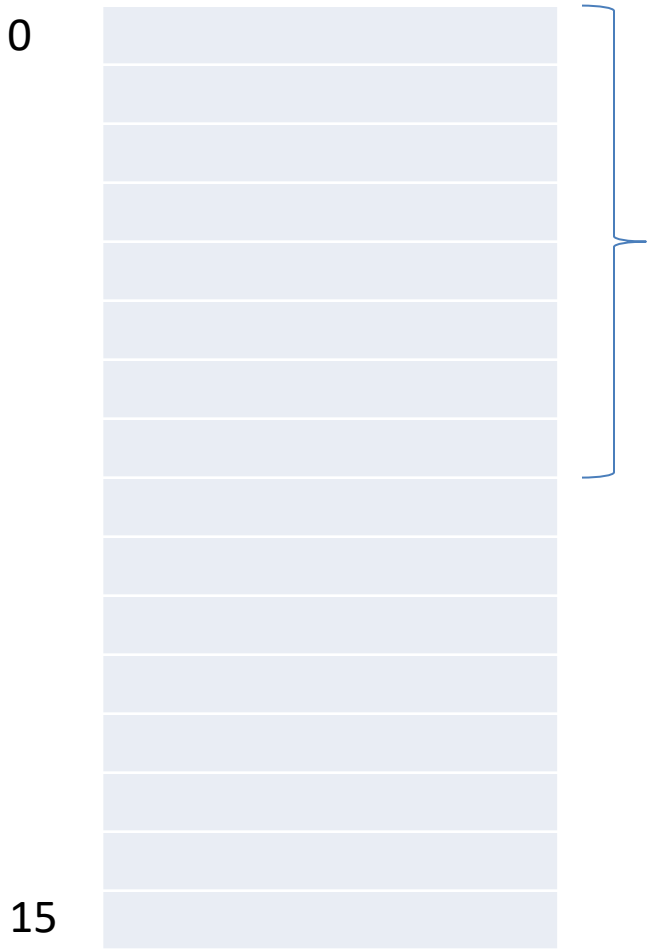
0 (000)
1 (001)
2 (010)
3 (011)
4 (100)
5 (101)
6 (110)
7 (111)

SRAM (Cache)

Fetch 19:
Which block in SRAM to check

Example

Fetch 19:
19 in binary 10011



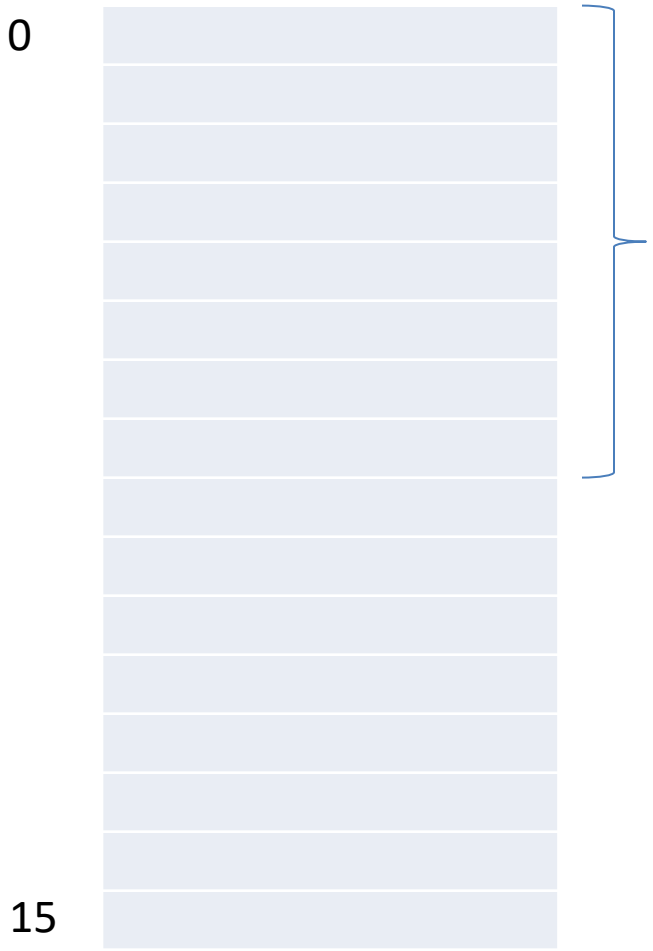
DRAM (Main memory)

Main memory → cache
0 (00**000**) → 0
1 (00**001**) → 1
2 (00**010**) → 2
…
8 (01**000**) → 0
9 (01**001**) → 1
10 (01**010**) → 2
…
16 (10**000**) → 0
17 (10**001**) → 1
18 (10**010**) → 2

SRAM (Cache)

0 (000)
1 (001)
2 (010)
3 (011)
4 (100)
5 (101)
6 (110)
7 (111)

Example



DRAM (Main memory)

Main memory➔ cache

0 (00**000**) ➔ 0
1 (00**001**) ➔ 1
2 (00**010**) ➔ 2
…
8 (01**000**) ➔ 0
9 (01**001**) ➔ 1
10 (01**010**) ➔ 2
…
16 (10**000**) ➔ 0
17 (10**001**) ➔ 1
18 (10**010**) ➔ 2

SRAM (Cache)

0 (000)
1 (001)
2 (010)
3 (011)
4 (100)
5 (101)
6 (110)
7 (111)

Fetch 19:
19 in binary 10**011**
Block 3
19 modulo 8 = 3

Example

0

15

DRAM (Main memory)

Main memory➔ cache

0 (00**000**) ➔ 0
1 (00**001**) ➔ 1
2 (00**010**) ➔ 2
…
8 (01**000**) ➔ 0
9 (01**001**) ➔ 1
10 (01**010**) ➔ 2
…
16 (10**000**) ➔ 0
17 (10**001**) ➔ 1
18 (10**010**) ➔ 2

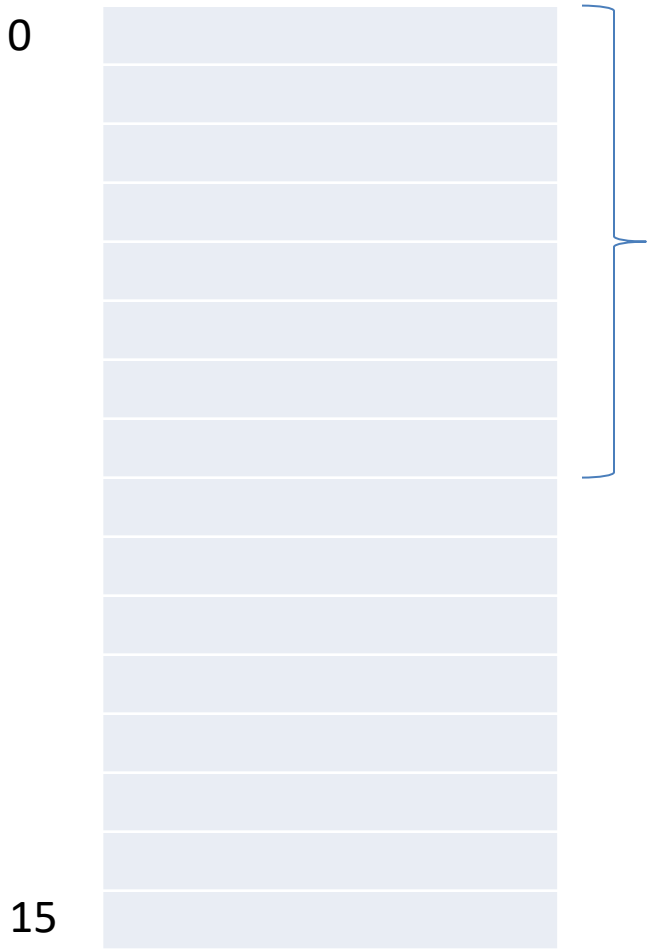
0 (000)
1 (001)
2 (010)
3 (011)
4 (100)
5 (101)
6 (110)
7 (111)

SRAM (Cache)

Fetch 19:
19 in binary 10**011**
Block 3
19 modulo 8 = 3

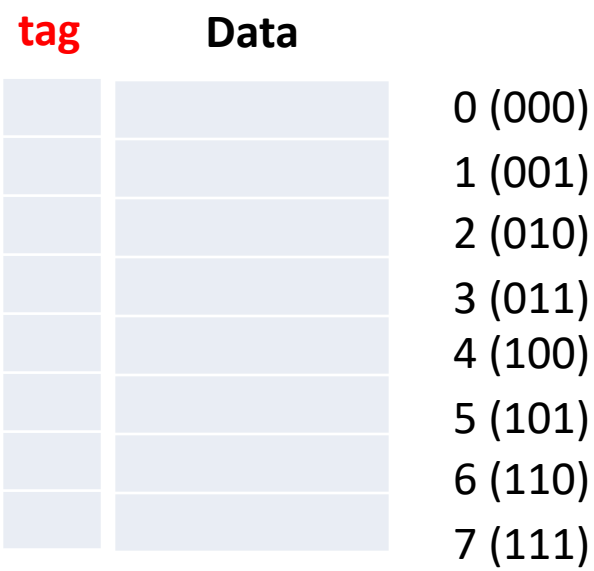
But the data in 011 could have come from address 3, 11 or from 19

Example



DRAM (Main memory)

Main memory➔ cache
0 (00**000**) ➔ 0
1 (00**001**) ➔ 1
2 (00**010**) ➔ 2
…
8 (01**000**) ➔ 0
9 (01**001**) ➔ 1
10 (01**010**) ➔ 2
…
16 (10**000**) ➔ 0
17 (10**001**) ➔ 1
18 (10**010**) ➔ 2



SRAM (Cache)

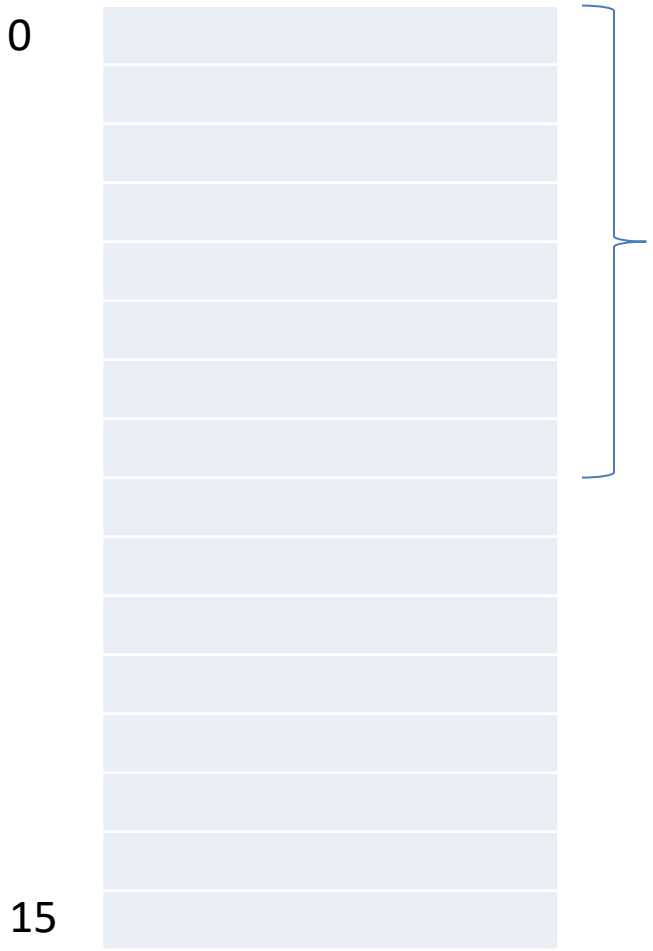
Fetch 19:
19 in binary 10**011**
Block 3
19 modulo 8 = 3

But the data in 011 could have come from address 3, 11 or from 19

Need to save the most significant bits of the address as well **10**011

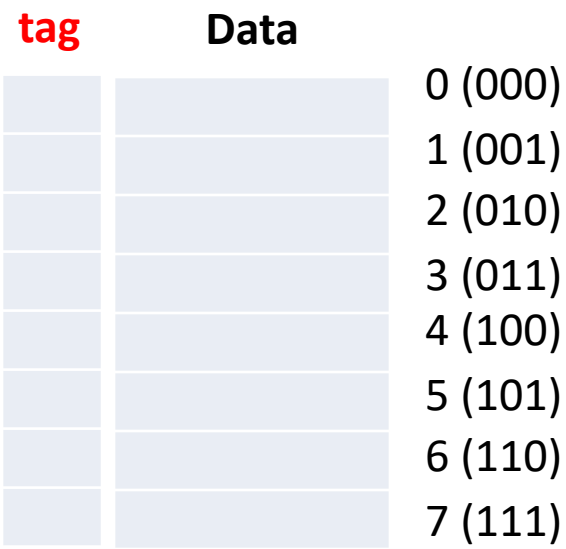
10 called the **tag**.
Now we can identify the data came form address 19 in main memory

Example



DRAM (Main memory)

Main memory➔ cache
0 (00**000**) ➔ 0
1 (00**001**) ➔ 1
2 (00**010**) ➔ 2
…
8 (01**000**) ➔ 0
9 (01**001**) ➔ 1
10 (01**010**) ➔ 2
…
16 (10**000**) ➔ 0
17 (10**001**) ➔ 1
18 (10**010**) ➔ 2



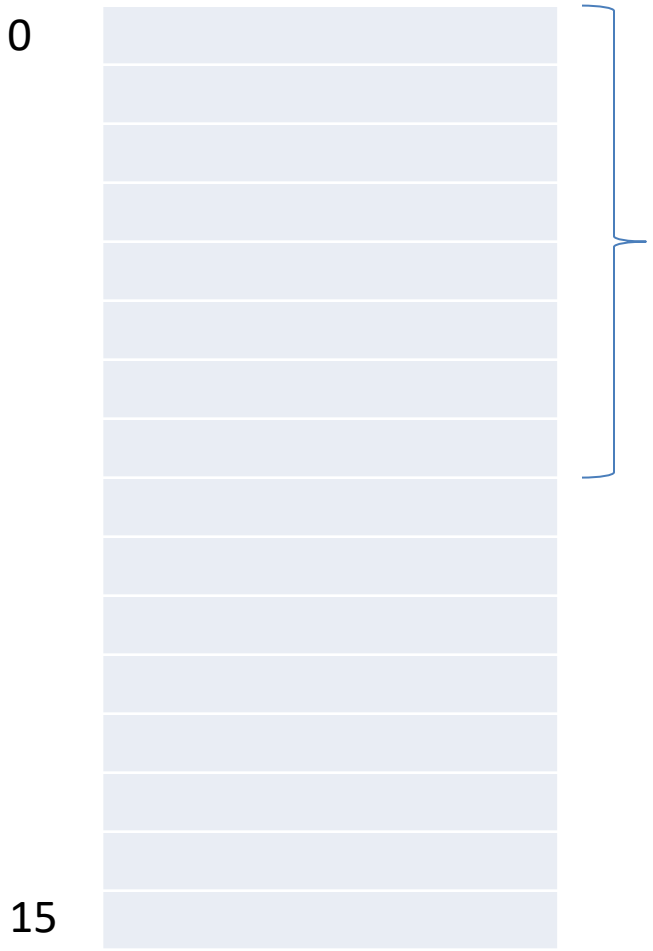
SRAM (Cache)

Fetch 19:
19 in binary 10**011**
Block 3
19 modulo 8 = 3

But the data in 011 could have come from address 3, 11 or from 19

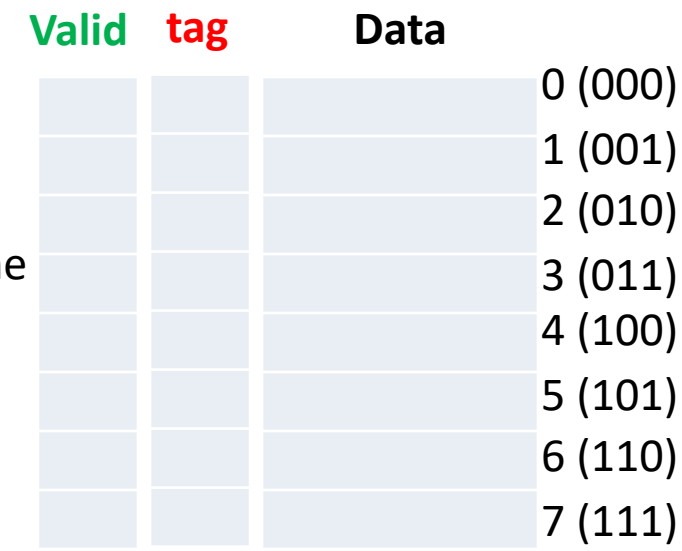
Make sure that the data in block is valid.
For example when processor starts, the cache does not have good values

Example



DRAM (Main memory)

Main memory ➔ cache
0 (00**000**) ➔ 0
1 (00**001**) ➔ 1
2 (00**010**) ➔ 2
…
8 (01**000**) ➔ 0
9 (01**001**) ➔ 1
10 (01**010**) ➔ 2
…
16 (10**000**) ➔ 0
17 (10**001**) ➔ 1
18 (10**010**) ➔ 2



SRAM (Cache)

Fetch 19:
19 in binary 10**011**
Block 3
19 modulo 8 = 3

But the data in 011 could have come from address 3, 11 or from 19

Make sure that the data in block is valid.
For example when processor starts, the cache does not have good values
Additional 1 bit (**valid bit**) to indicate validity for each block.

Cache Example

- 8-blocks, 1 word/block, direct mapped
- Initial state

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem[10110]**
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem[11010]**
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

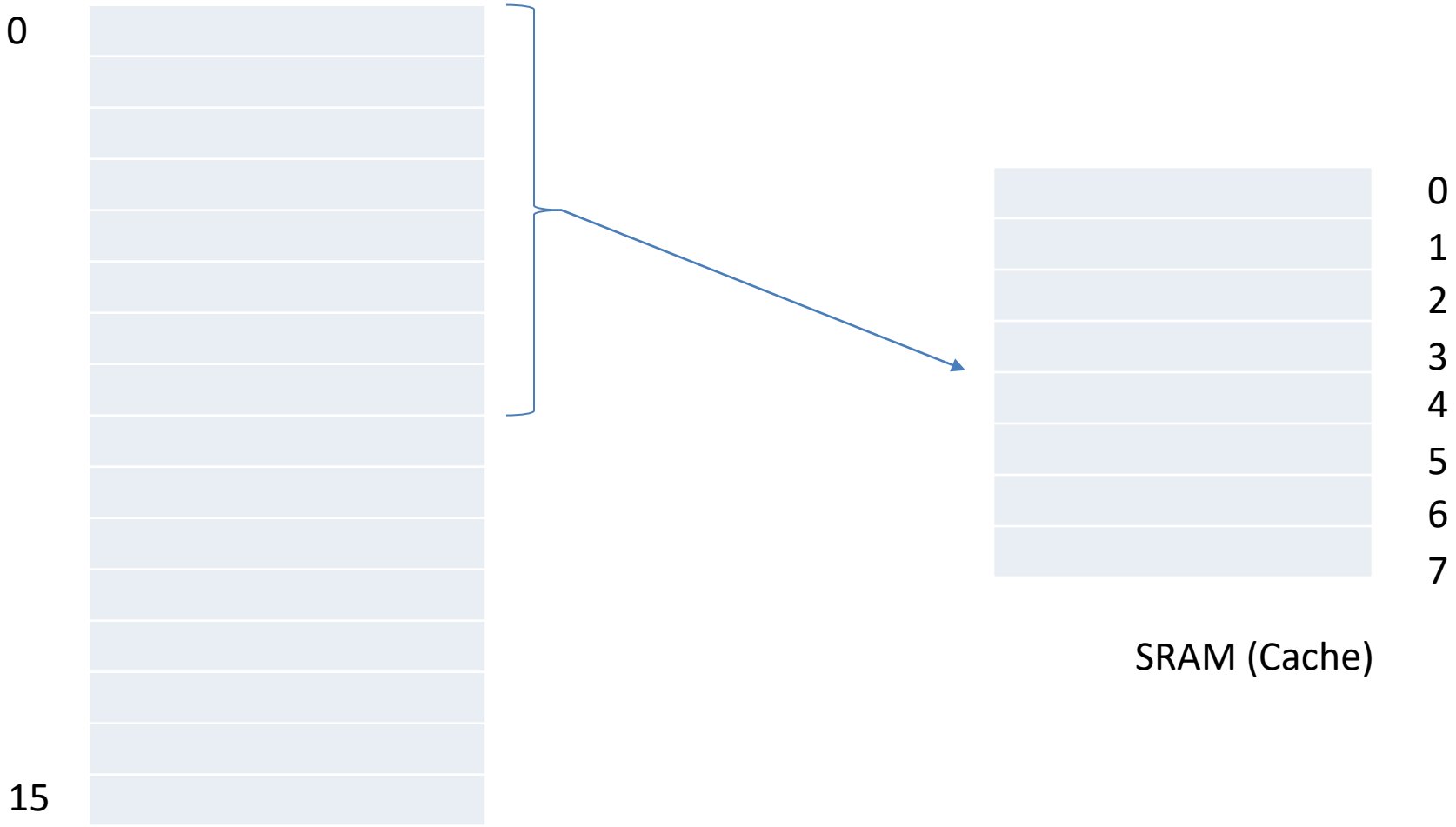
Index	V	Tag	Data
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	**Y**	**10**	**Mem[10010]**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

SRAM Data



DRAM (Main memory)

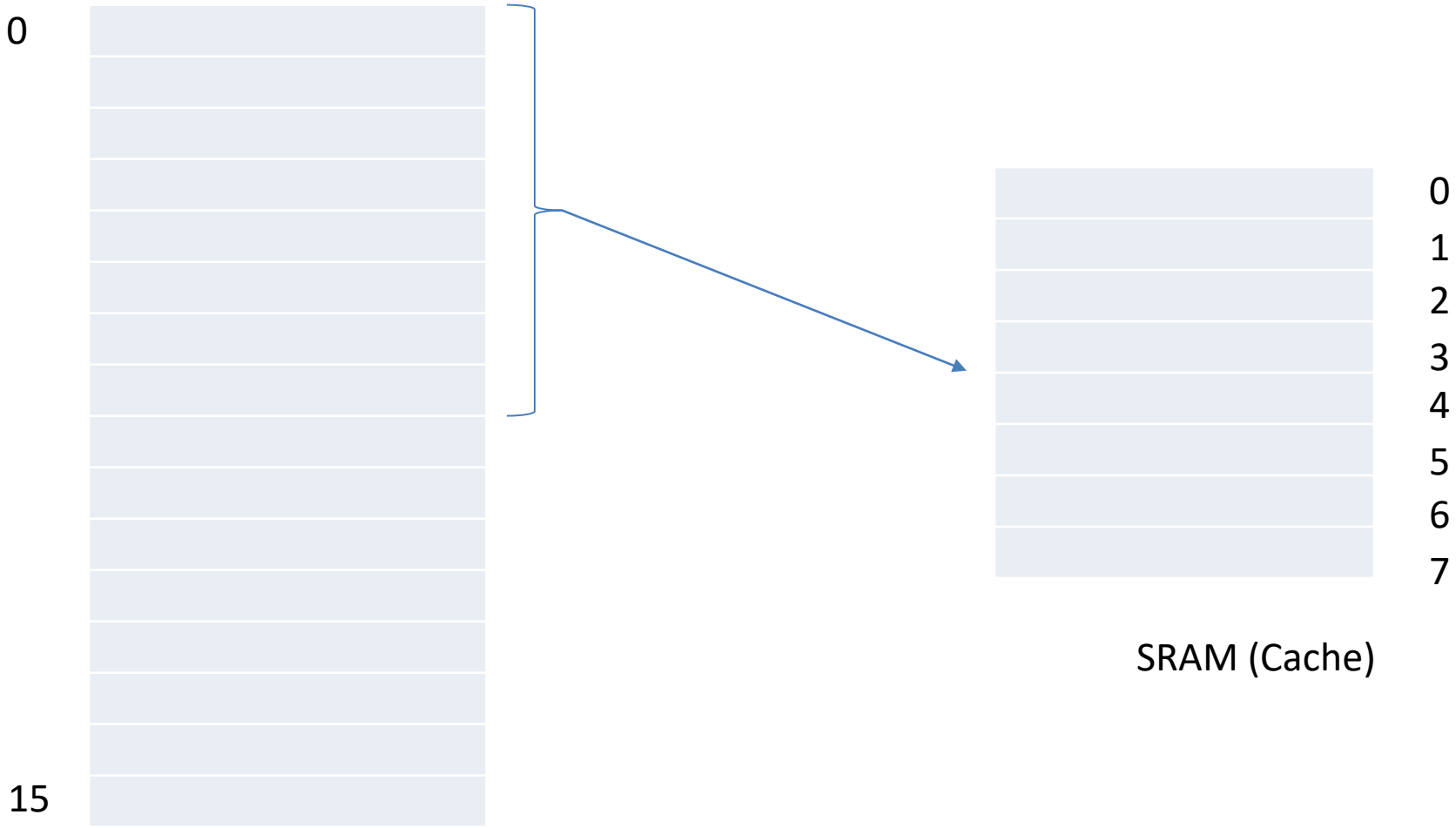
Fetch 0:
Not available in cache so fetch form main memory
Spatial locality
Fetch next 7 as well

In this Example,

Each address in Main memory is mapped to cache

SRAM Data

Byte addressed,
I byte in each
address



DRAM (Main memory)

SRAM (Cache)

Fetch 0:
Not available in cache so fetch form main memory
Spatial locality
Fetch next 7 as well

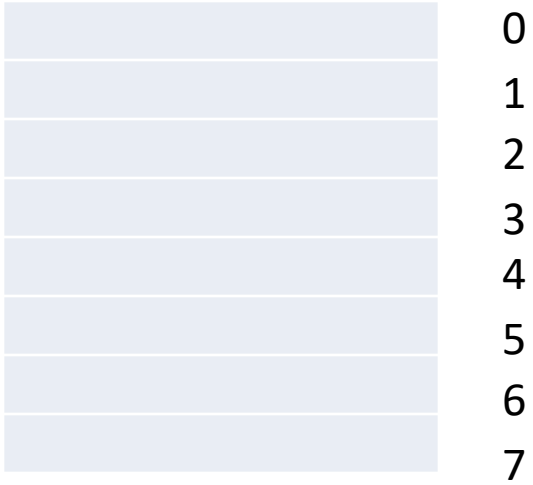
In this Example,

Each address in Main memory is mapped to cache

SRAM Data

Byte addressed,
I byte in each
address

DRAM (Main
memory)

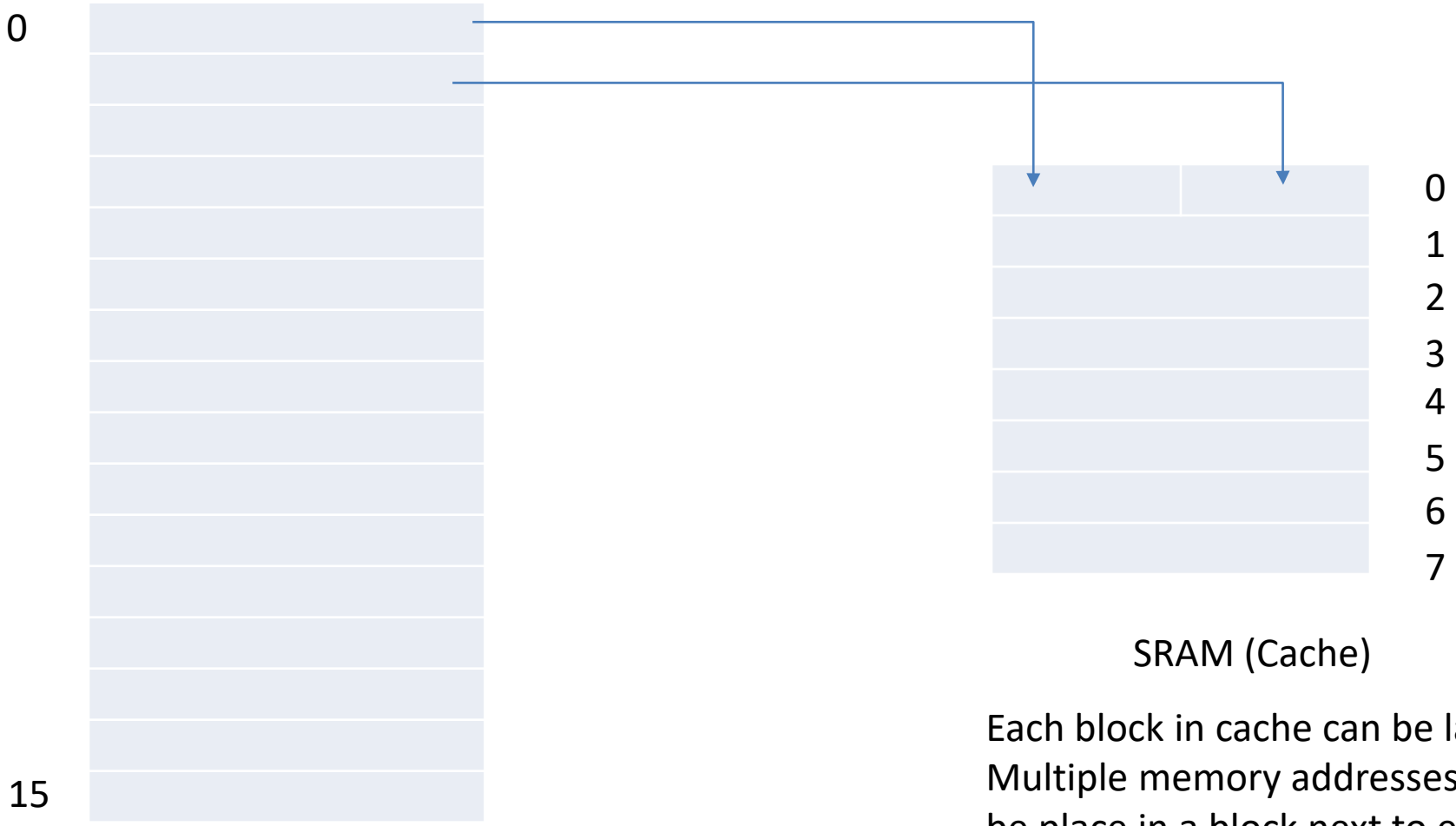


SRAM (Cache)

Each block in cache can be large than 1 byte.
Multiple memory addresses (spatially proximal)
addresses can be place in a block next to each other

SRAM Data

Byte addressed,
I byte in each
address



DRAM (Main memory)

SRAM (Cache)

Fetch 0:
Not available in cache so fetch form main memory
Spatial locality
Fetch address 1 as well, as the block can accommodate 2 bytes

Each block in cache can be large than 1 byte.
Multiple memory addresses (spatially proximal) can be place in a block next to each other
If each block is 16 bits (2 Bytes)

SRAM Data

Byte addressed,
I byte in each
address

0

15

DRAM (Main
memory)

0 (000)
1 (001)
2 (010)
3 (011)
4 (100)
5 (101)
6 (110)
7 (111)

SRAM (Cache)

SRAM Data

Byte addressed,
I byte in each
address

0

15

DRAM (Main
memory)

Fetch
0 ➔ miss

		0 (000)
		1 (001)
		2 (010)
		3 (011)
		4 (100)
		5 (101)
		6 (110)
		7 (111)

SRAM (Cache)

SRAM Data

Byte addressed,
I byte in each
address

0

15

DRAM (Main
memory)

Fetch
0 ➔ miss (load address 0, 1)

		0 (000)
		1 (001)
		2 (010)
		3 (011)
		4 (100)
		5 (101)
		6 (110)
		7 (111)

SRAM (Cache)

SRAM Data

Byte addressed,
I byte in each
address

0

15

DRAM (Main
memory)

Number of blocks (2^m) = 8
Mapped to the index matching
$\log_2(8) = 3$

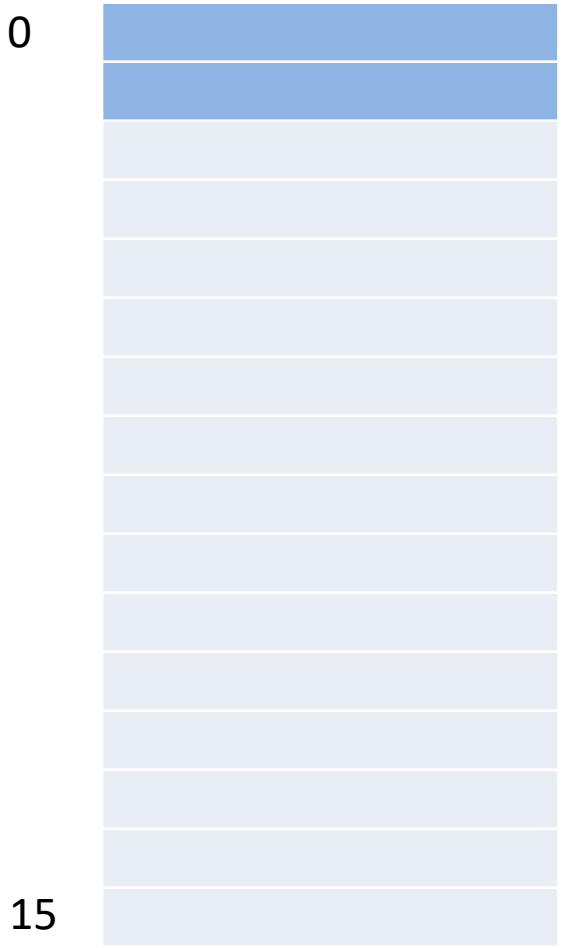
Fetch
0 (00**000**) ➔ miss (load address 0, 1)

		0 (000)
		1 (001)
		2 (010)
		3 (011)
		4 (100)
		5 (101)
		6 (110)
		7 (111)

SRAM (Cache)

SRAM Data

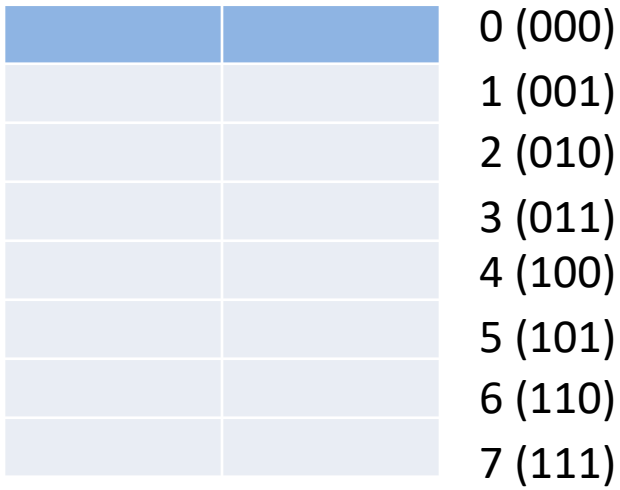
Byte addressed,
I byte in each
address



DRAM (Main memory)

Number of blocks (2^m) = 8
Mapped to the index matching
$\log_2(8) = 3$

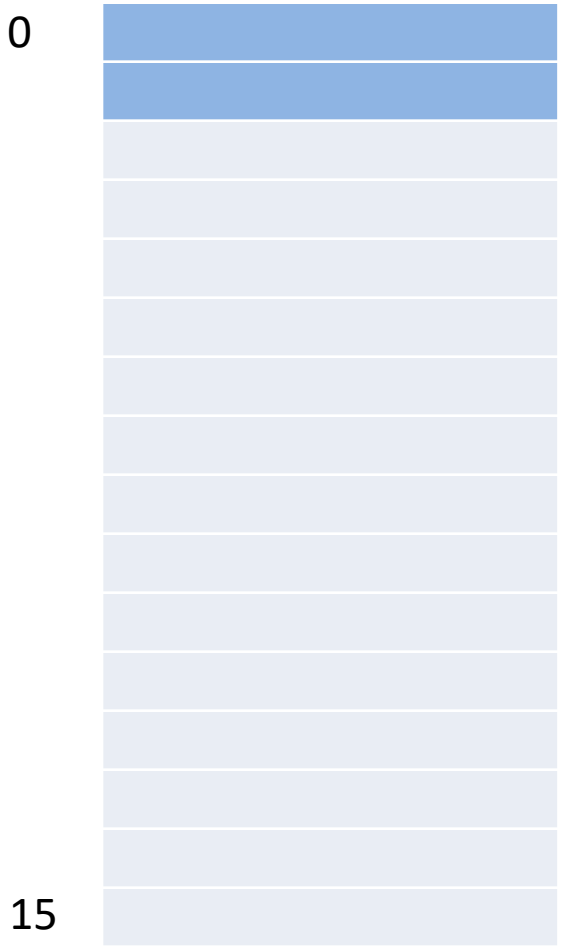
Fetch
0 (00**000**) ➔ miss (load address 0, 1)
1 (00**001**) ➔ hit



SRAM (Cache)

SRAM Data

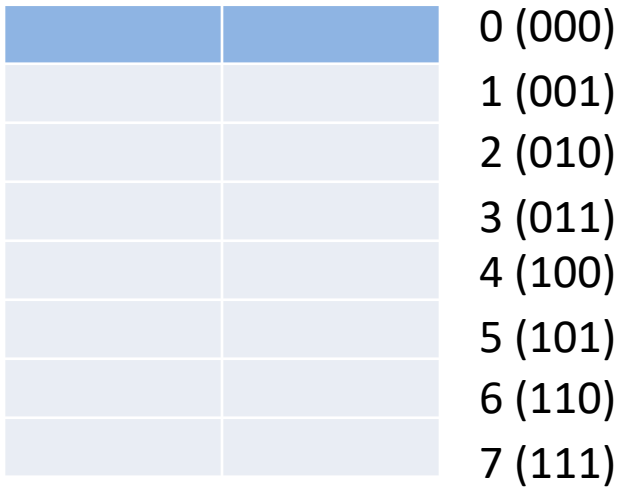
Byte addressed,
I byte in each
address



DRAM (Main memory)

Number of blocks (2^m) = 8
Mapped to the index matching
$\log_2(8) = 3$

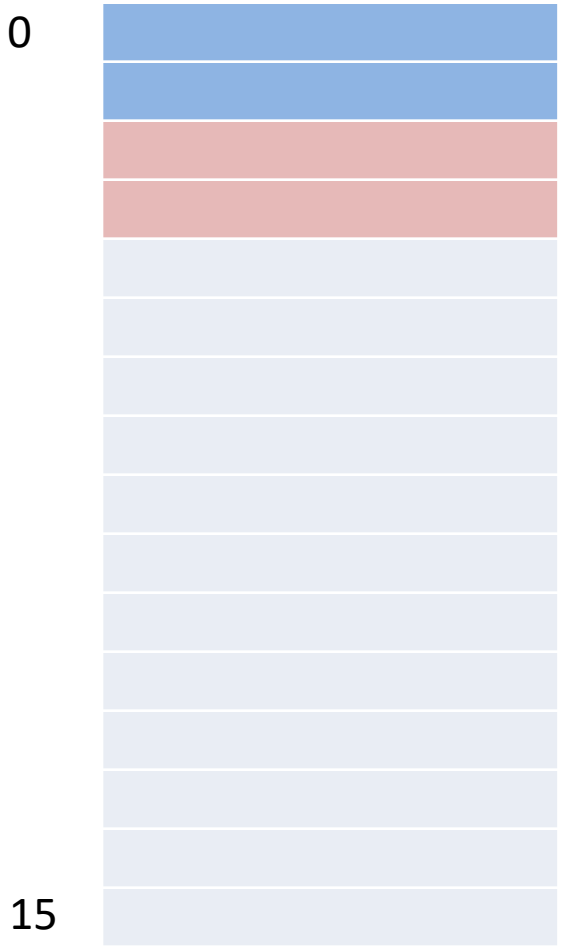
Fetch
0 (00**000**) ➔ miss (load address 0, 1)
1 (00**001**) ➔ hit
2 (00**010**) ➔ miss (load 2, 3)



SRAM (Cache)

SRAM Data

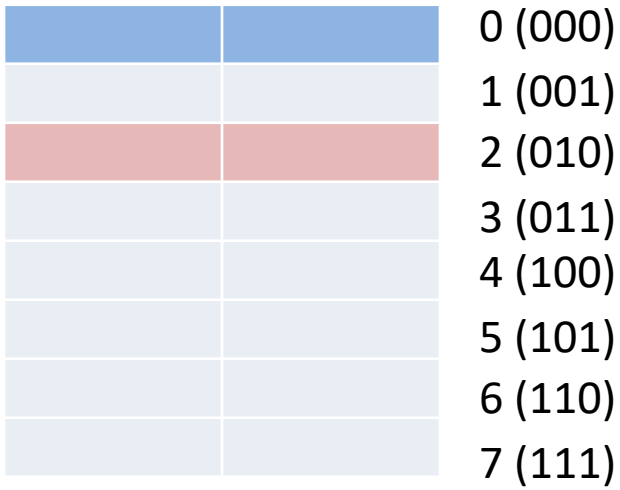
Byte addressed,
I byte in each
address



DRAM (Main memory)

Number of blocks (2^m) = 8
Mapped to the index matching
$\log_2(8) = 3$

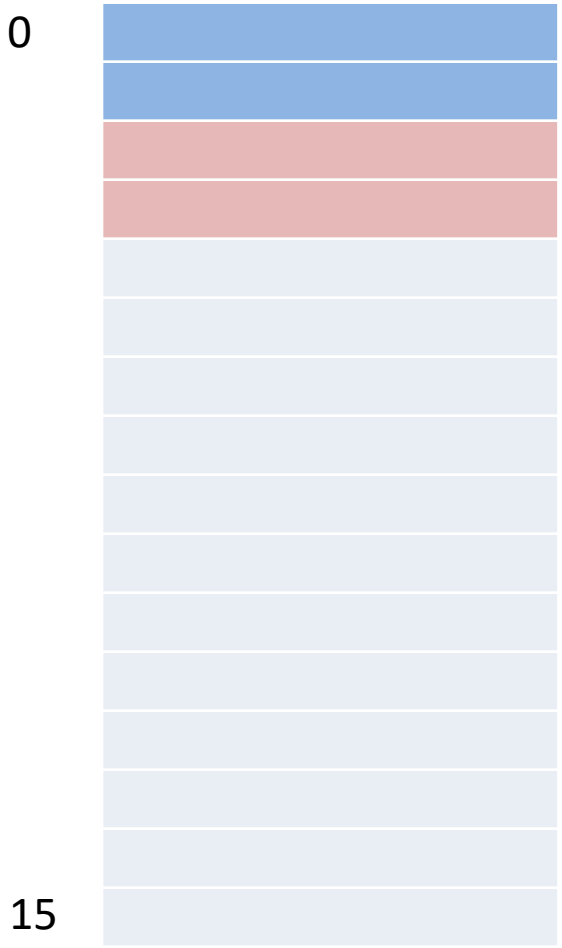
Fetch
0 (00**000**) ➔ miss (load address 0, 1)
1 (00**001**) ➔ hit
2 (00**010**) ➔ miss (load 2, 3)



SRAM (Cache)

SRAM Data

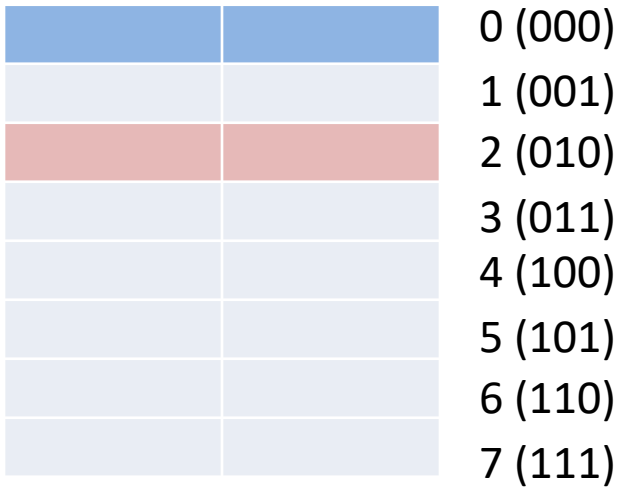
Byte addressed,
I byte in each
address



DRAM (Main memory)

Number of blocks (2^m) = 8
Mapped to the index matching
$\log_2(8) = 3$

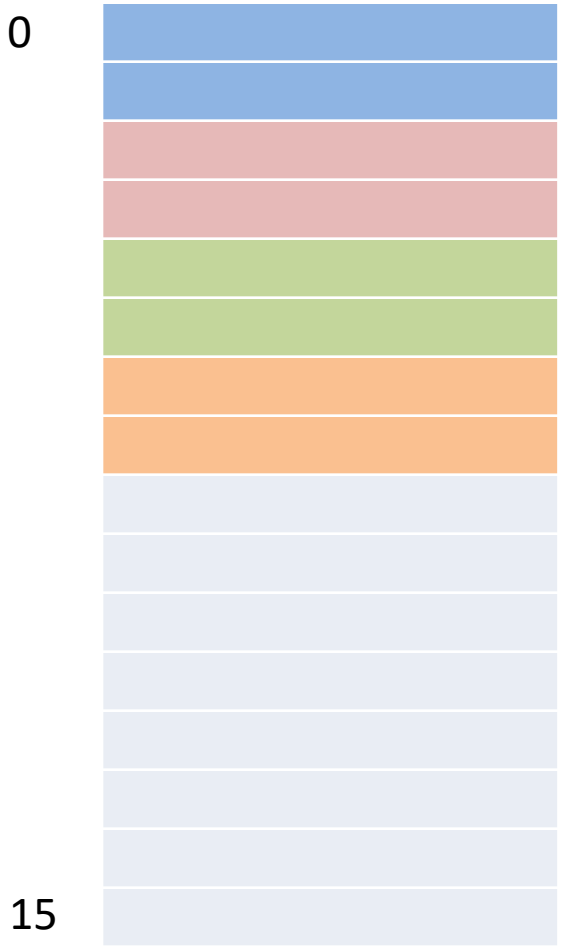
Fetch
0 (00**000**) ➔ miss (load address 0, 1)
1 (00**001**) ➔ hit
2 (00**010**) ➔ miss (load 2, 3)
3 (00**011**) ➔ hit



SRAM (Cache)

SRAM Data

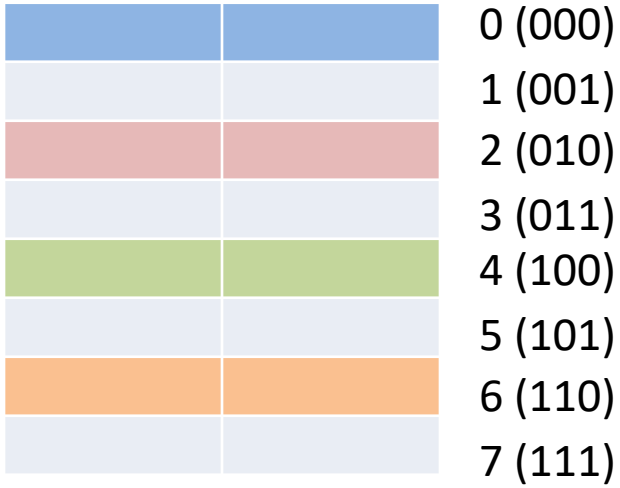
Byte addressed,
I byte in each
address



DRAM (Main memory)

Number of blocks (2^m) = 8
Mapped to the index matching
$\log_2(8) = 3$

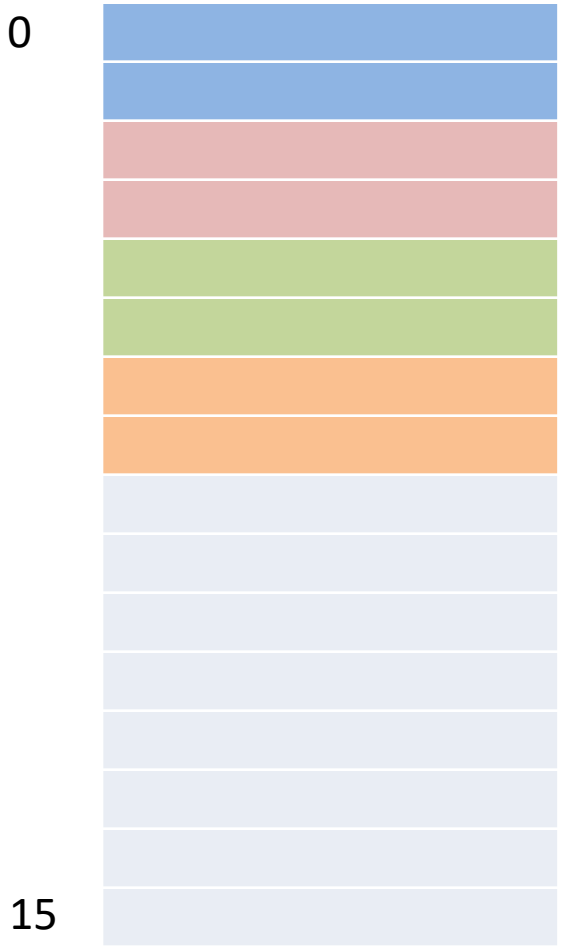
Fetch
0 (00**000**) ➔ miss (load address 0, 1)
1 (00**001**) ➔ hit
2 (00**010**) ➔ miss (load 2, 3)
3 (00**011**) ➔ hit
…



SRAM (Cache)

SRAM Data

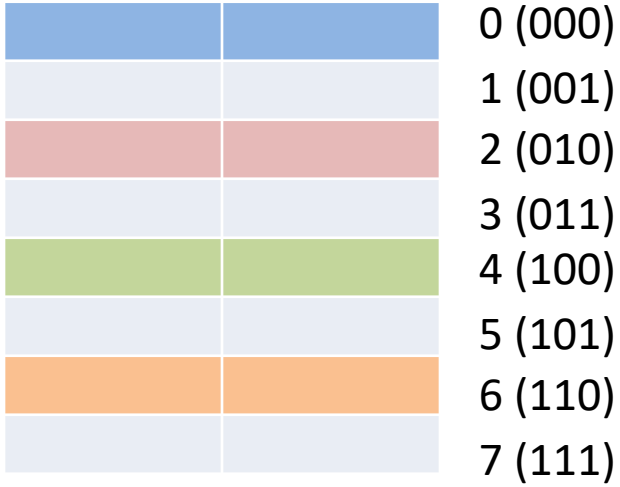
Byte addressed,
I byte in each
address



DRAM (Main memory)

Number of blocks (2^m) = 8
Mapped to the index matching
$\log_2(8) = 3$

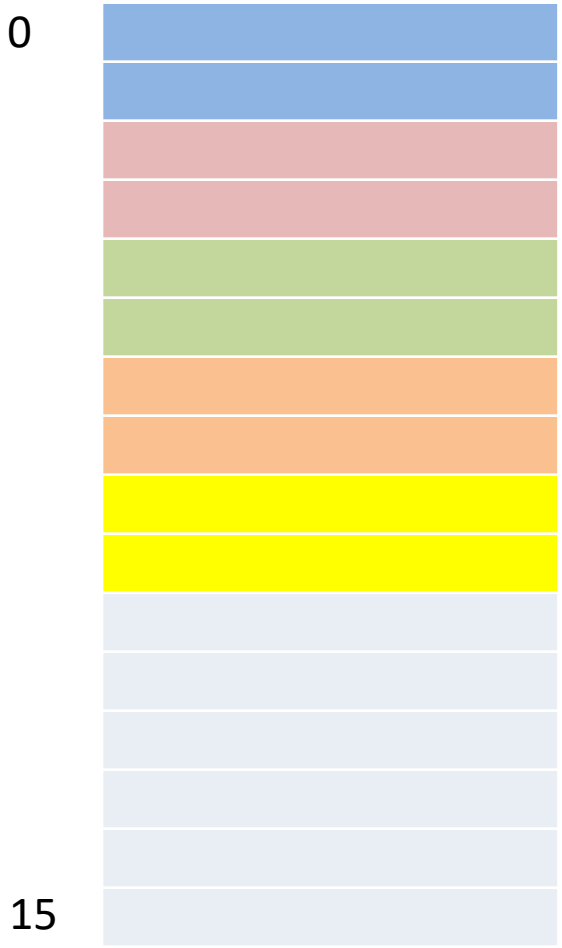
Fetch
0 (00**000**) ➔ miss (load address 0, 1)
1 (00**001**) ➔ hit
2 (00**010**) ➔ miss (load 2, 3)
3 (00**011**) ➔ hit
…
8 (01**000**) ➔ miss (load 8, 9)



SRAM (Cache)

SRAM Data

Byte addressed,
I byte in each
address

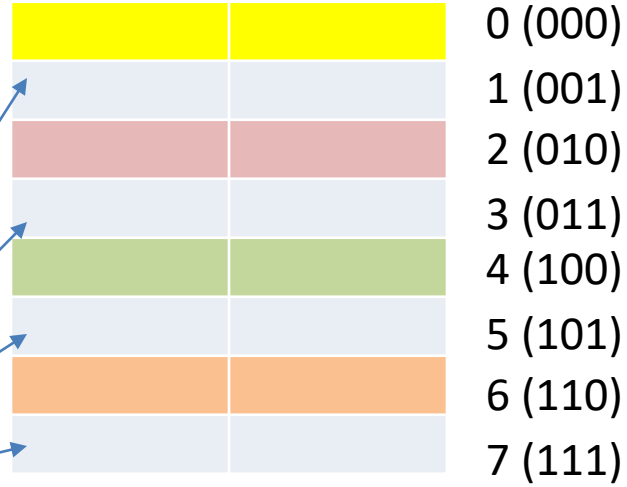


DRAM (Main memory)

Number of blocks (2^m) = 8
Mapped to the index matching
$\log_2(8) = 3$

Fetch
0 (00**000**) ➔ miss (load address 0, 1)
1 (00**001**) ➔ hit
2 (00**010**) ➔ miss (load 2, 3)
3 (00**011**) ➔ hit
…
8 (01**000**) ➔ miss (load 8, 9)

Blocks 1, 3, 5, 7 are never mapped



SRAM (Cache)

SRAM Data

Byte addressed,
I byte in each
address

0

15

DRAM (Main
memory)

Number of blocks (2^m) = 8
Mapped to the index matching
$\log_2(8) = 3$

Fetch
0 (00000) ➔

Memory Alignment:
Make sure that each block in a cache stores 2^n bytes (power of 2).
In this example n = 1
Each block store 2^1bytes

0 (000)
1 (001)
2 (010)
3 (011)
4 (100)
5 (101)
6 (110)
7 (111)

SRAM (Cache)

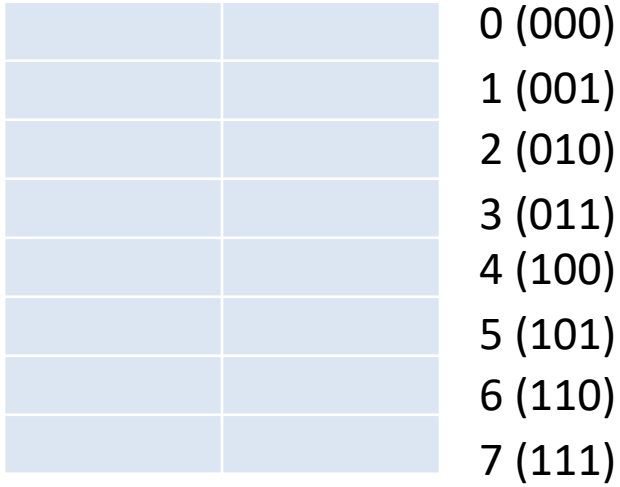
SRAM Data

Byte addressed,
I byte in each
address

DRAM (Main memory)

Number of blocks (2^m) = 8
Mapped to the index matching
$\log_2(8) = 3$

Fetch
0 (00000) ➔



SRAM (Cache)

Memory Alignment:
Make sure that each block in a cache stores 2^n bytes (power of 2).

Ignore last n bits in mapping computation

SRAM Data

Byte addressed,
I byte in each
address

0

15

DRAM (Main
memory)

Number of blocks (2^m) = 8
Mapped to the index matching
$\log_2(8) = 3$

Fetch
0 (00000) ➔

Memory Alignment:
Make sure that each block in a cache stores 2^n bytes (power of 2).

Ignore last n bits in mapping computation
Here n = 1

0 (000)
1 (001)
2 (010)
3 (011)
4 (100)
5 (101)
6 (110)
7 (111)

SRAM (Cache)

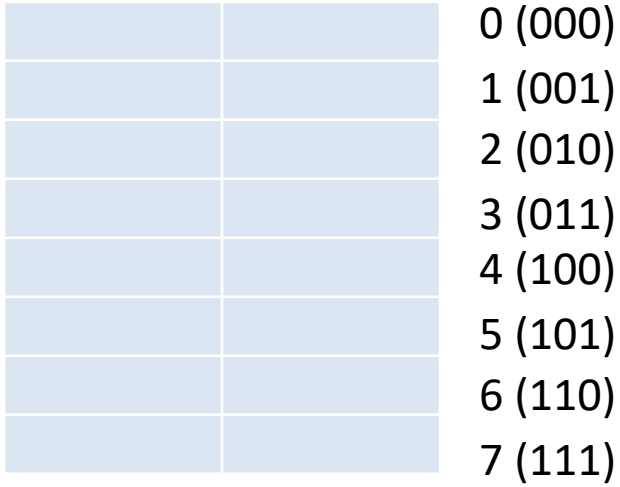
SRAM Data

Byte addressed,
I byte in each
address

DRAM (Main memory)

Number of blocks (2^m) = 8
Mapped to the index matching
$\log_2(8) = 3$

Fetch
0 (00000) ➔



SRAM (Cache)

Memory Alignment:
Make sure that each block in a cache stores 2^n bytes (power of 2).

Ignore last n bits in mapping computation
Here n = 1

SRAM Data

Byte addressed,
I byte in each
address

0

15

DRAM (Main memory)

Number of blocks (2^m) = 8
Mapped to the index matching
$\log_2(8) = 3$

Fetch
0 (0**000**0) ➔ miss

Memory Alignment:
Make sure that each block in a cache stores 2^n bytes (power of 2).

Ignore last n bits in mapping computation
Here n = 1

0 (000)
1 (001)
2 (010)
3 (011)
4 (100)
5 (101)
6 (110)
7 (111)

SRAM (Cache)

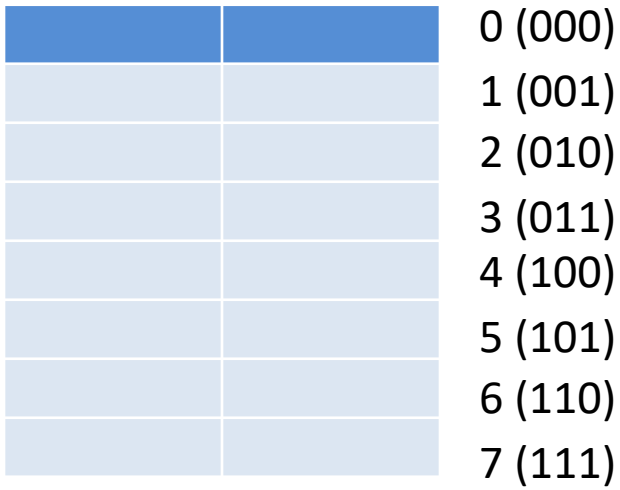
SRAM Data

Byte addressed,
I byte in each
address

DRAM (Main memory)

Number of blocks (2^m) = 8
Mapped to the index matching
$\log_2(8) = 3$

Fetch
0 (0**000**0) ➔ miss (load)



SRAM (Cache)

Memory Alignment:
Make sure that each block in a cache stores 2^n bytes (power of 2).

Ignore last n bits in mapping computation
Here n = 1

SRAM Data

Byte addressed,
I byte in each
address

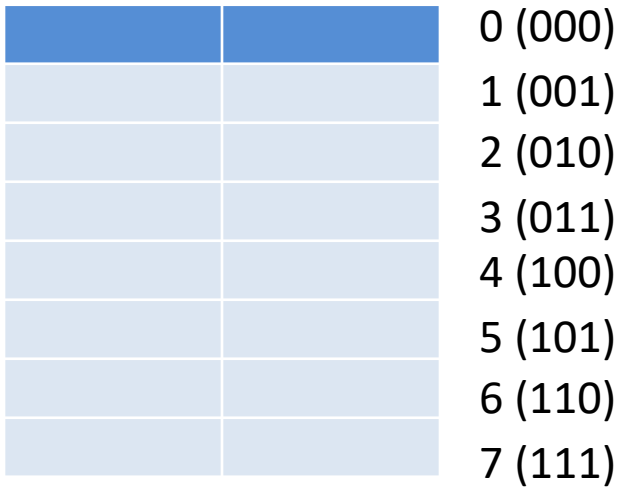
DRAM (Main memory)

Number of blocks (2^m) = 8
Mapped to the index matching
$\log_2(8) = 3$

Fetch
0 (0**000**0) ➔ miss (load)
1 (0**000**1) ➔ hit

Memory Alignment:
Make sure that each block in a cache stores 2^n bytes (power of 2).

Ignore last n bits in mapping computation
Here n = 1



SRAM (Cache)

SRAM Data

Byte addressed,
I byte in each
address

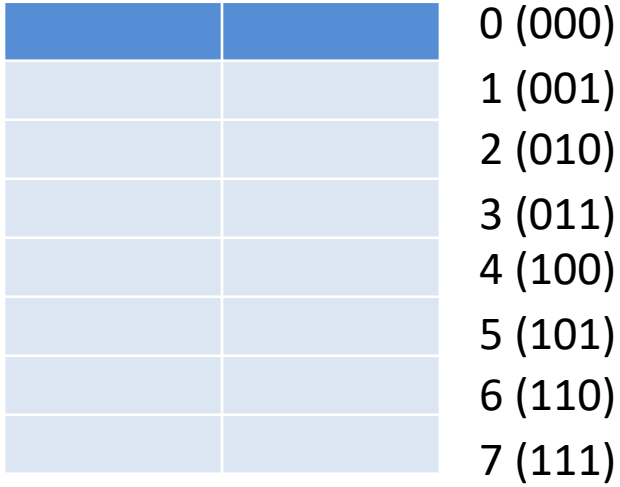
DRAM (Main memory)

Number of blocks (2^m) = 8
Mapped to the index matching
$\log_2(8) = 3$

Fetch
0 (0**0000**0) ➔ miss (load)
1 (0**0000**1) ➔ hit
2 (00010) ➔ miss

Memory Alignment:
Make sure that each block in a cache stores 2^n bytes (power of 2).

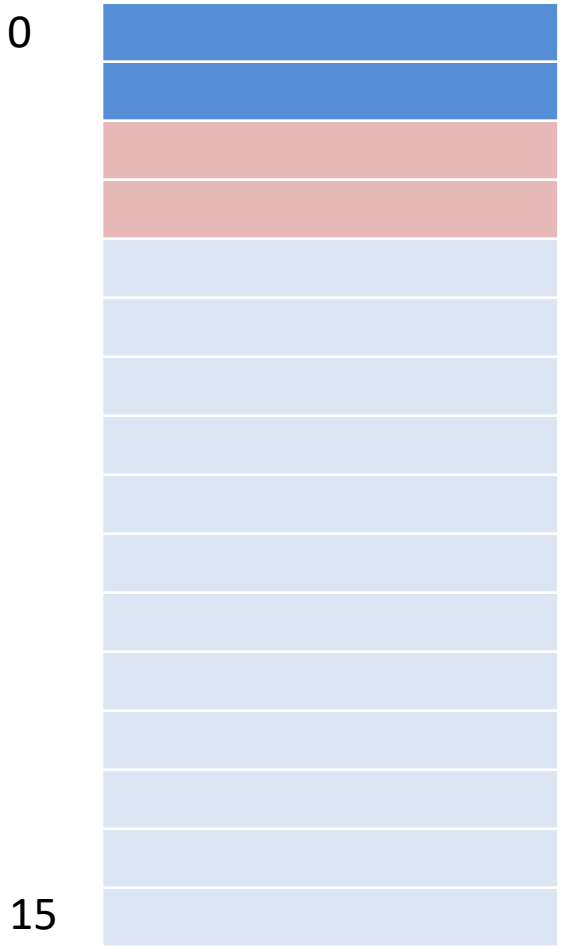
Ignore last n bits in mapping computation
Here n = 1



SRAM (Cache)

SRAM Data

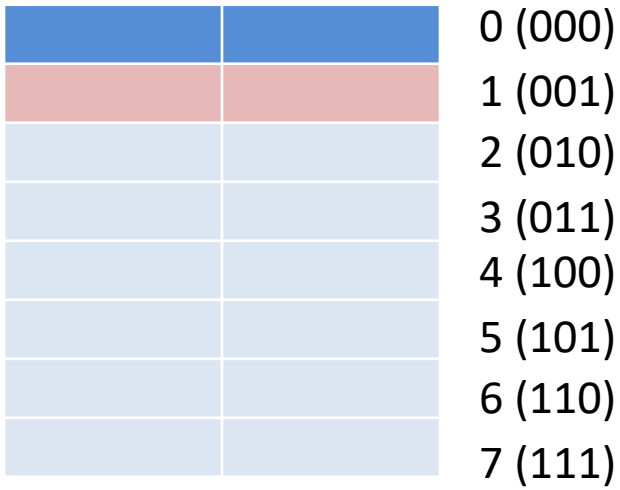
Byte addressed,
I byte in each
address



DRAM (Main memory)

Number of blocks (2^m) = 8
Mapped to the index matching
$\log_2(8) = 3$

Fetch
0 (0**0000**0) ➔ miss (load)
1 (0**0001**) ➔ hit
2 (0**0010**) ➔ miss (load)



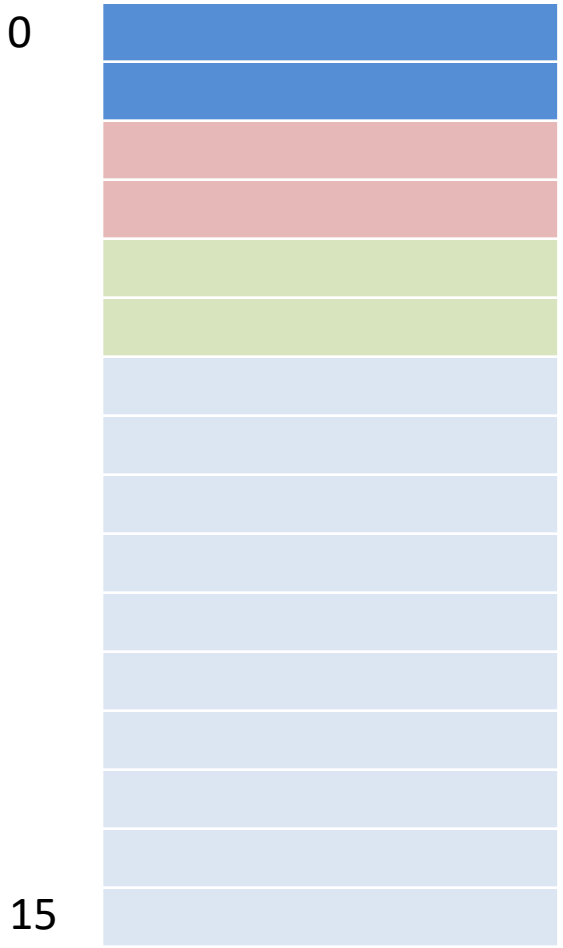
SRAM (Cache)

Memory Alignment:
Make sure that each block in a cache stores 2^n bytes (power of 2).

Ignore last n bits in mapping computation
Here n = 1

SRAM Data

Byte addressed,
I byte in each
address



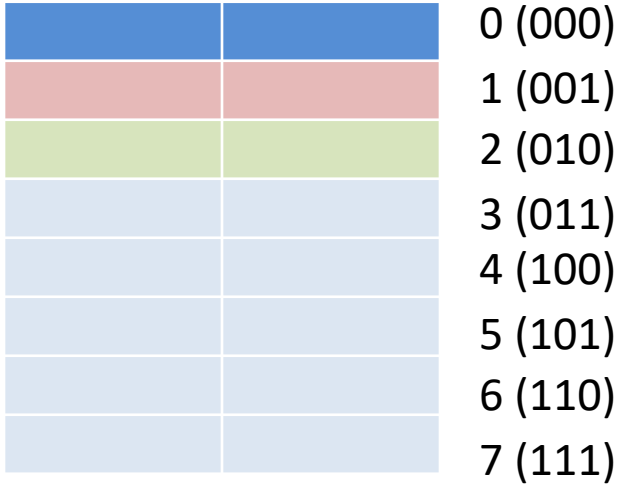
DRAM (Main memory)

Number of blocks (2^m) = 8
Mapped to the index matching
$\log_2(8) = 3$

Fetch
0 (0**0000**0) ➔ miss (load)
1 (0**0000**1) ➔ hit
2 (0**0010**0) ➔ miss (load)
3 (0**0011**) ➔ hit
4 (0**0100**) ➔ miss (load)

Memory Alignment:
Make sure that each block in a cache stores 2^n bytes (power of 2).

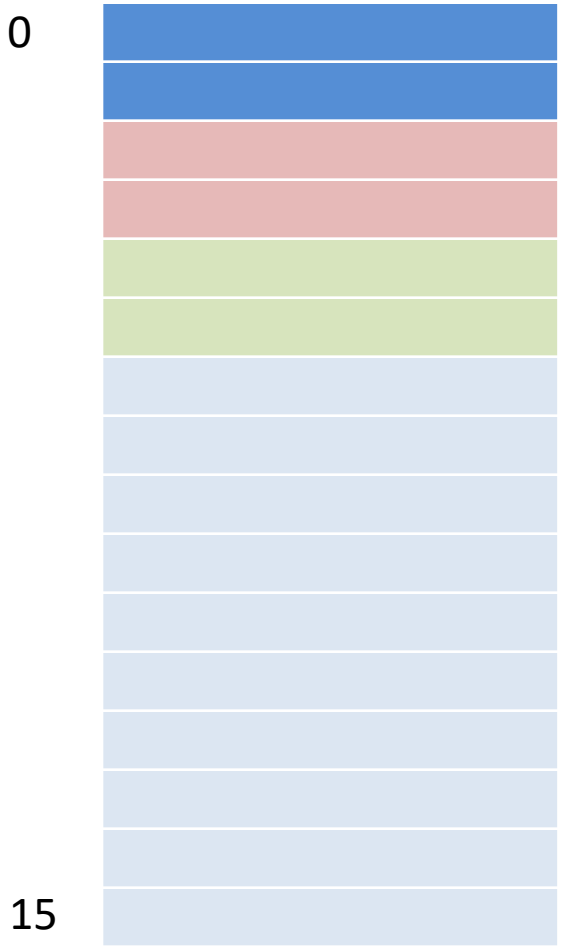
Ignore last n bits in mapping computation
Here n = 1



SRAM (Cache)

SRAM Data

Byte addressed,
I byte in each
address



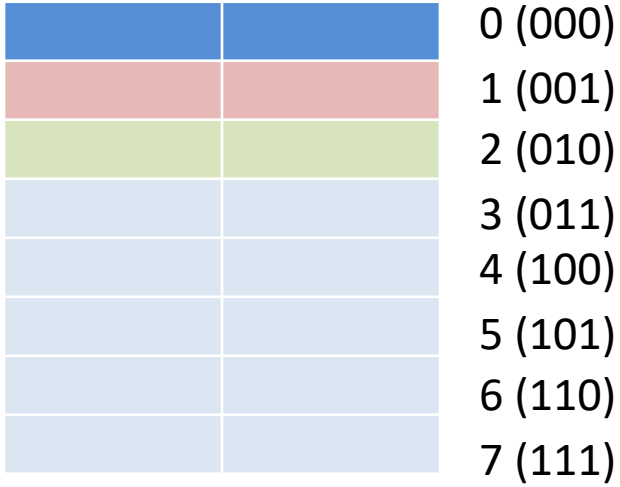
DRAM (Main memory)

Number of blocks (2^m) = 8
Mapped to the index matching
$\log_2(8) = 3$

Fetch
0 (0**0000**) ➔ miss (load)
1 (0**0001**) ➔ hit
2 (0**0010**) ➔ miss (load)
3 (0**0011**) ➔ hit
4 (0**0100**) ➔ miss (load)

Memory Alignment:
Make sure that each block in a cache stores 2^n bytes (power of 2).

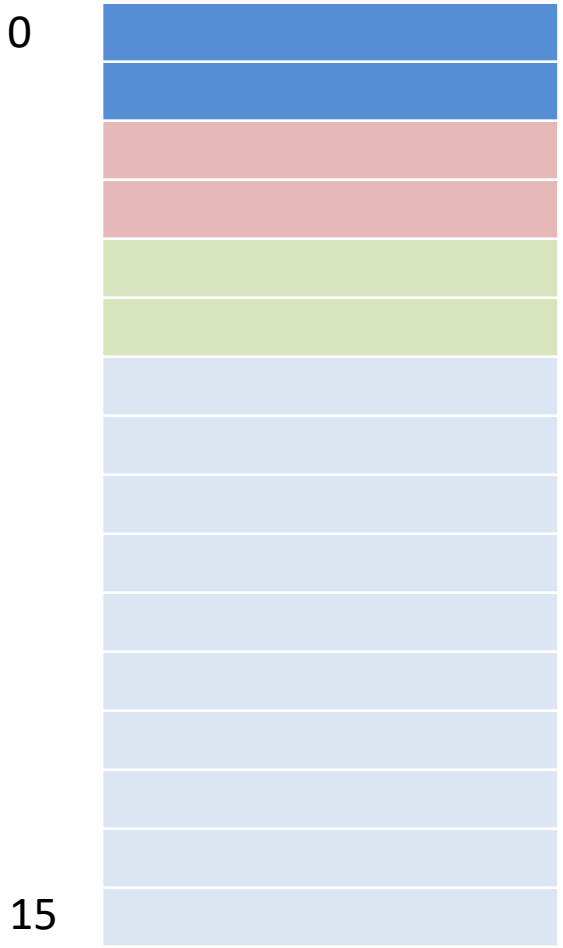
Last n bits used as offset to locate bytes in block



SRAM (Cache)

SRAM Data

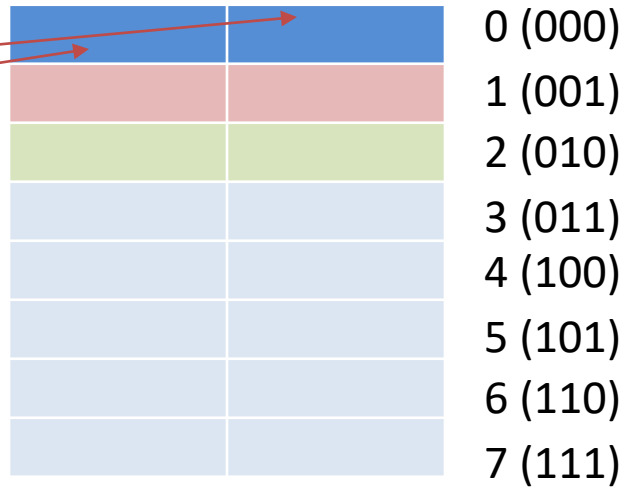
Byte addressed,
I byte in each
address



DRAM (Main memory)

Number of blocks (2^m) = 8
Mapped to the index matching
$\log_2(8) = 3$

Fetch
0 (0**0000**) ➔ miss (load)
1 (0**0001**) ➔ hit
2 (0**0010**) ➔ miss (load)
3 (0**0011**) ➔ hit
4 (0**0100**) ➔ miss (load)



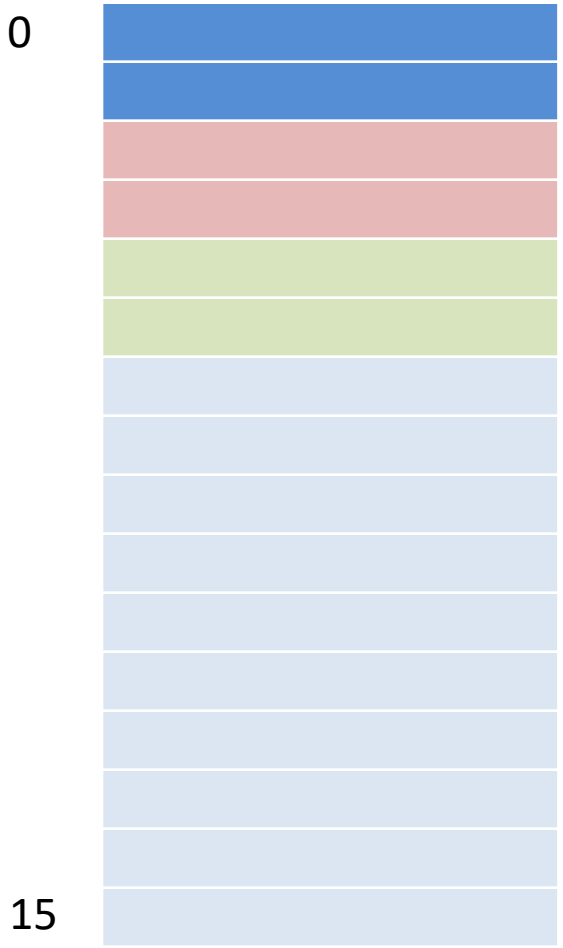
SRAM (Cache)

Memory Alignment:
Make sure that each block in a cache stores 2^n bytes (power of 2).

Last n bits used as offset to locate bytes in block (**Byte Offset**)

SRAM Data

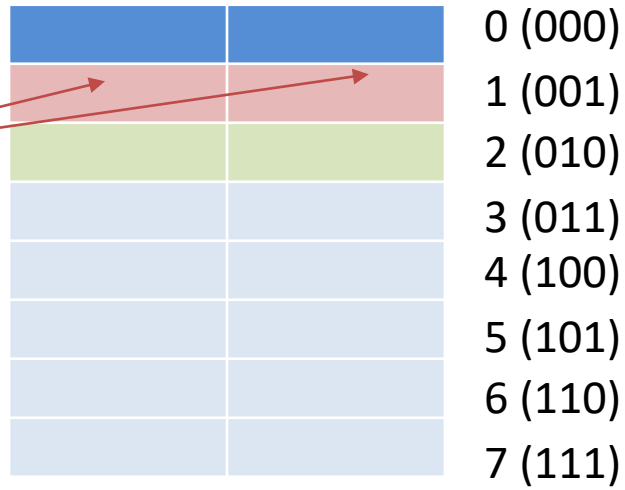
Byte addressed,
I byte in each
address



DRAM (Main memory)

Number of blocks (2^m) = 8
Mapped to the index matching
$\log_2(8) = 3$

Fetch
0 (00000) ➔ miss (load)
1 (00001) ➔ hit
2 (00010) ➔ miss (load)
3 (00011) ➔ hit
4 (00100) ➔ miss (load)



SRAM (Cache)

Memory Alignment:
Make sure that each block in a cache stores 2^n bytes (power of 2).

Last n bits used as offset to locate bytes in block

SRAM Data

Byte addressed,
I byte in each
address

0

15

DRAM (Main
memory)

Number of blocks (2^m) = 8
Mapped to the index matching
$\log_2(8) = 3$

Fetch
0 (00000) ➔ miss (load)
1 (00001) ➔
2 (00010) ➔
3 (00011) ➔
4 (00100) ➔

Memory Alignment:
Make sure that each block in a cache stores 2^n bytes.

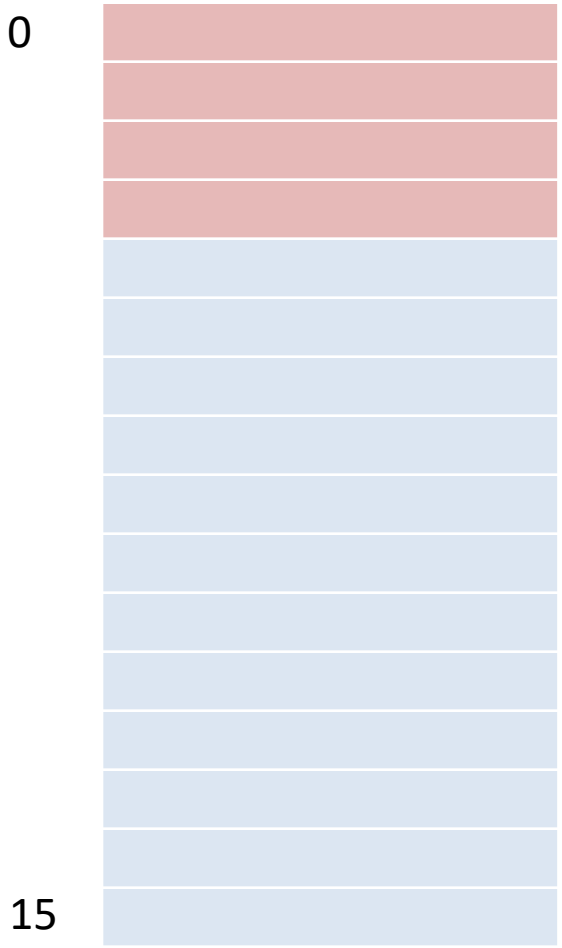
If n = 2, each block has 4 bytes (1 Word)

0 (000)
1 (001)
2 (010)
3 (011)
4 (100)
5 (101)
6 (110)
7 (111)

SRAM (Cache)

SRAM Data

Byte addressed,
I byte in each
address



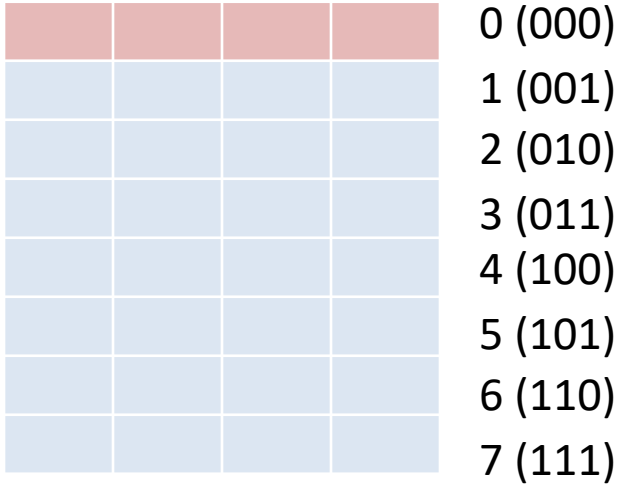
DRAM (Main memory)

Number of blocks (2^m) = 8
Mapped to the index matching
$\log_2(8) = 3$

Fetch
0 (**000**00) ➔ miss (load)
1 (00001) ➔
2 (00010) ➔
3 (00011) ➔
4 (00100) ➔

Memory Alignment:
Make sure that each block in a cache stores 2^n bytes.

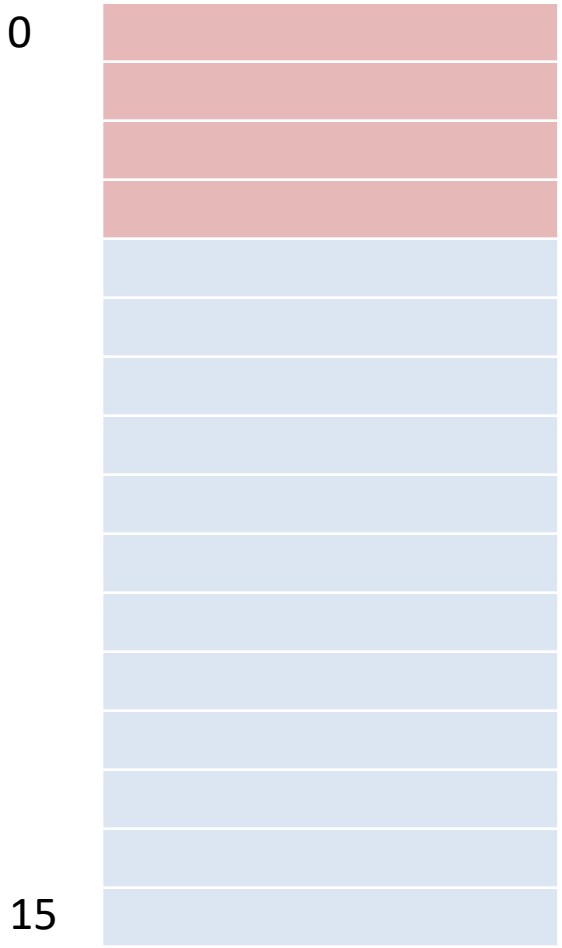
If n = 2, each block has 4 bytes (1 Word)



SRAM (Cache)

SRAM Data

Byte addressed,
I byte in each
address



DRAM (Main memory)

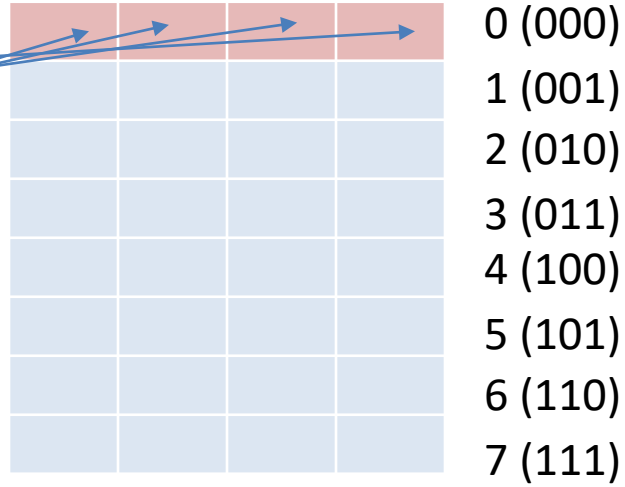
Number of blocks (2^m) = 8
Mapped to the index matching
$\log_2(8) = 3$

Fetch
0 (**000**00) ➔ miss (load)
1 (**000**01) ➔ hit
2 (**000**10) ➔ hit
3 (**000**11) ➔ hit
4 (00100) ➔

Memory Alignment:
Make sure that each block in a cache stores 2^n bytes.

If n = 2, each block has 4 bytes (1 Word)

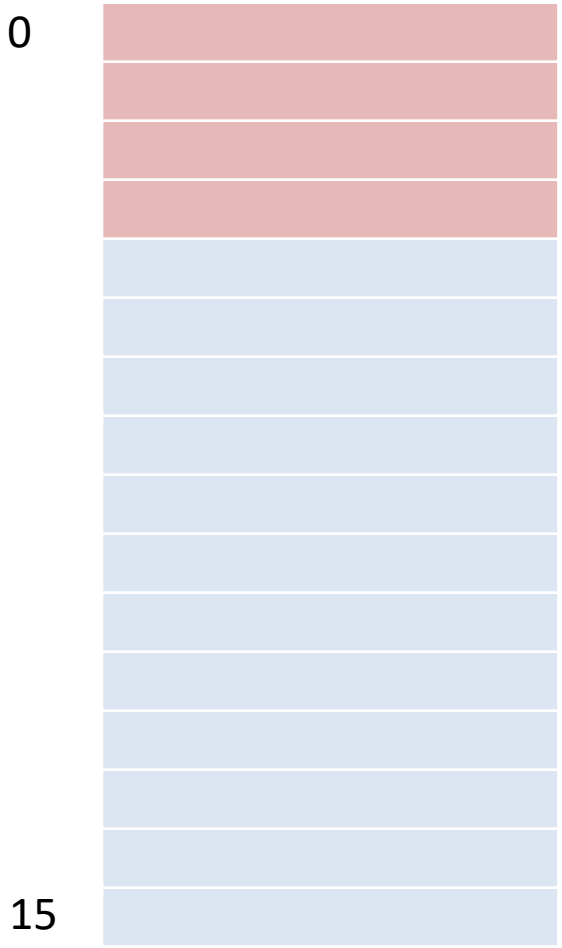
Byte
Offset



SRAM (Cache)

SRAM Data

Byte addressed,
I byte in each
address



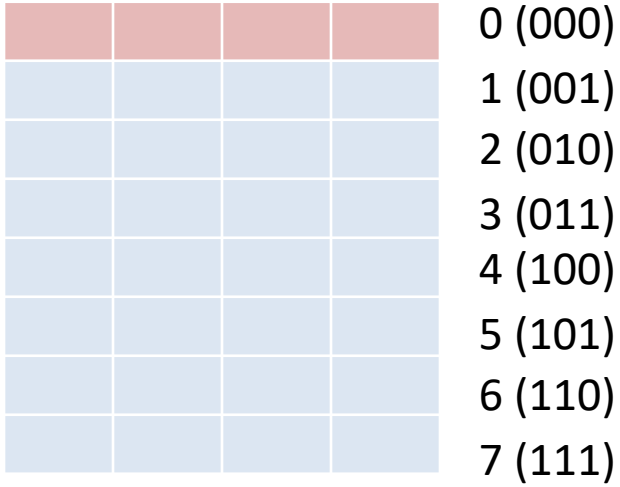
DRAM (Main memory)

Number of blocks (2^m) = 8
Mapped to the index matching
$\log_2(8) = 3$

Fetch
0 (**00**00) ➔ miss (load)
1 (**00**01) ➔ hit
2 (**00**10) ➔ hit
3 (**00**11) ➔ hit
4 (00100) ➔

n=1 ➔n=2
More hits

Is larger block size better?



SRAM (Cache)

Block Size Considerations

- Larger blocks should reduce miss rate
  - Due to spatial locality
- But in a fixed-sized cache
  - SRAM is expensive
  - Larger blocks $\Rightarrow$ fewer of them
    - More competition $\Rightarrow$ increased miss rate
  - Larger blocks $\Rightarrow$ pollution
- Larger miss penalty
  - Can override benefit of reduced miss rate
  - Early restart and critical-word-first can help

A more realistic example

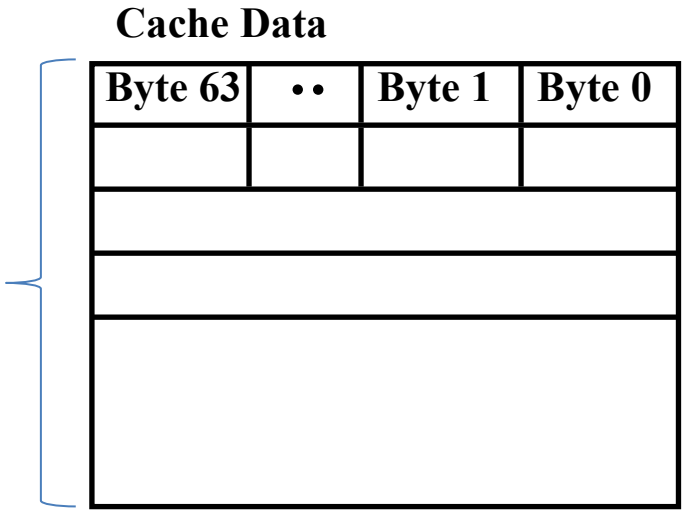
- 48 bit addresses
- 64 KB (64 * 1024 Bytes) of direct access cache
- Cache block size of 64 Bytes

- How many cache blocks does this cache have ?

A more realistic example

- 48 bit addresses
- 64 KB (64 * 1024 Bytes) of direct access cache
- Cache block size of 64 Bytes

- How many cache blocks does this cache have ?



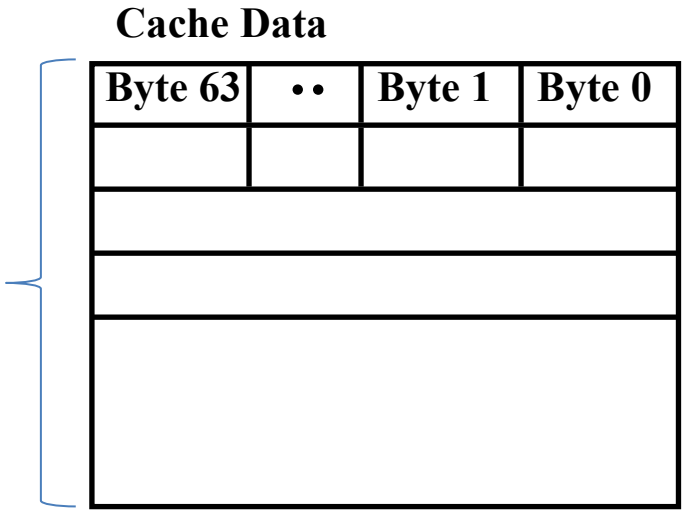
A more realistic example

- 48 bit addresses
- 64 KB (64 * 1024 Bytes) of direct access cache
- Cache block size of 64 Bytes

- How many cache blocks does this cache have ?

No. of cache blocks: 64 * 1024 Bytes /64 Bytes = 1024 Blocks

- No. of bits required for the cache index:



A more realistic example

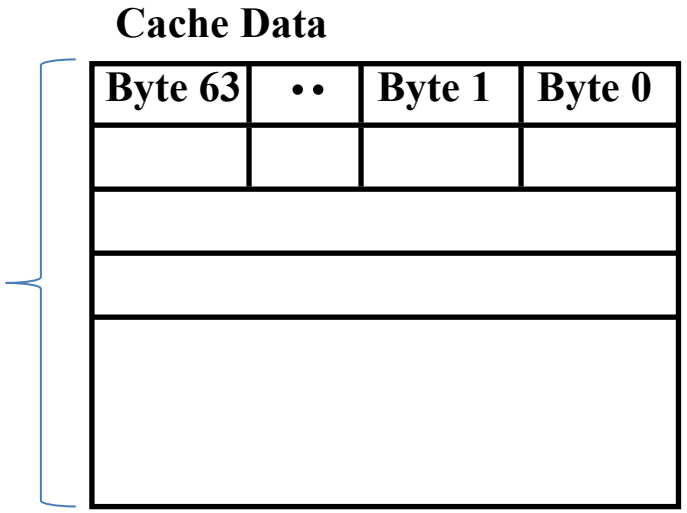
- 48 bit addresses
- 64 KB (64 * 1024 Bytes) of direct access cache
- Cache block size of 64 Bytes

- How many cache blocks does this cache have ?

No. of cache blocks: 64 * 1024 Bytes /64 Bytes = 1024 Blocks

- No. of bits required for the cache index:

$$\log_2(1024) = 10 \;\; \Rightarrow m = 10$$



A more realistic example

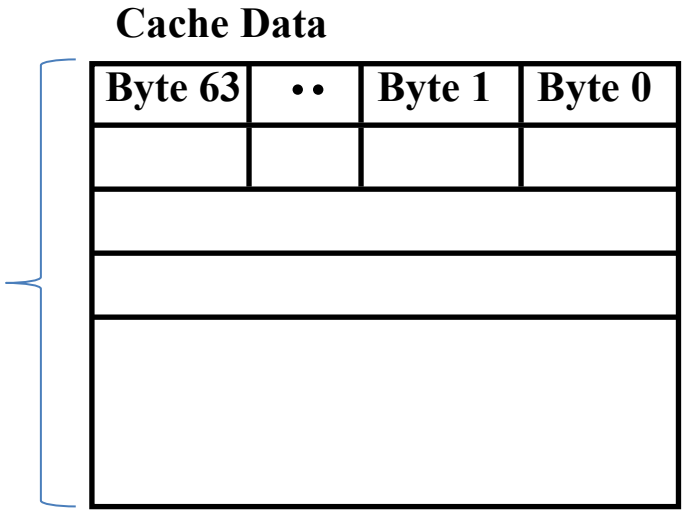
- 48 bit addresses
- 64 KB (64 * 1024 Bytes) of direct access cache
- Cache block size of 64 Bytes

- How many cache blocks does this cache have ?

No. of cache blocks: 64 * 1024 Bytes /64 Bytes = 1024 Blocks

- No. of bits required for the cache index:
- $\log_2(1024) = 10$ => m = 10

- How many bits do we need to ignore end of the address because the cache block size is 64 Bytes?



A more realistic example

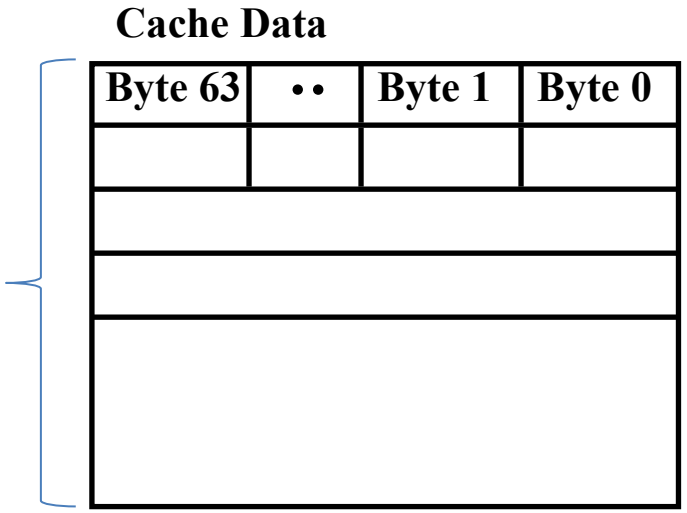
- 48 bit addresses
- 64 KB (64 * 1024 Bytes) of direct access cache
- Cache block size of 64 Bytes

- How many cache blocks does this cache have ?

No. of cache blocks: 64 * 1024 Bytes /64 Bytes = 1024 Blocks

- No. of bits required for the cache index:
- $\log_2(1024) = 10$ => m = 10

- How many bits do we need to ignore end of the address because the cache block size is 64 Bytes?
- $2^w = 64$ => w = 6



A more realistic example

48 bit address 0100 1101 1001 0000 0010 0100 1101 1001 0000 0010 0100 1101