You must be logged in PollEv to get credit

Participation

The “participation” segment of your grade has three main components:

- 4% for Lecture attendance, as measured by occasional Poll Everywhere polls. Starting on Oct. 20th, you need to be **logged into an account assoicated with your Cornell NetID** for your Poll Everywhere participation to count. This is the only way for us to reliably identify who participated.
- 4% for lab attendance, as recorded by the lab’s instructors.
- 2% for surveys:
    - The introduction survey (on Gradescope) in the first week of class.
    - The mid-semester feedback survey.
    - The semester-end course evaluation.

We know that life happens, so you can miss up to 3 lab sections and 5 lectures without penalty.

Cornell Bowers CIS
Computer Science

CS3410: Computer Systems and Organization

LEC16: Caches (Vol. II)

Professor Giulia Guidi
Wednesday, October 22, 2025

Credits: Bala, Bracy, Garcia, Guidi, Kao, Sampson, Sirer, Weatherspoon

Plan for Today

- Brief review of temporal and spatial locality
- Cache design: direct mapped, fully associative, set associative

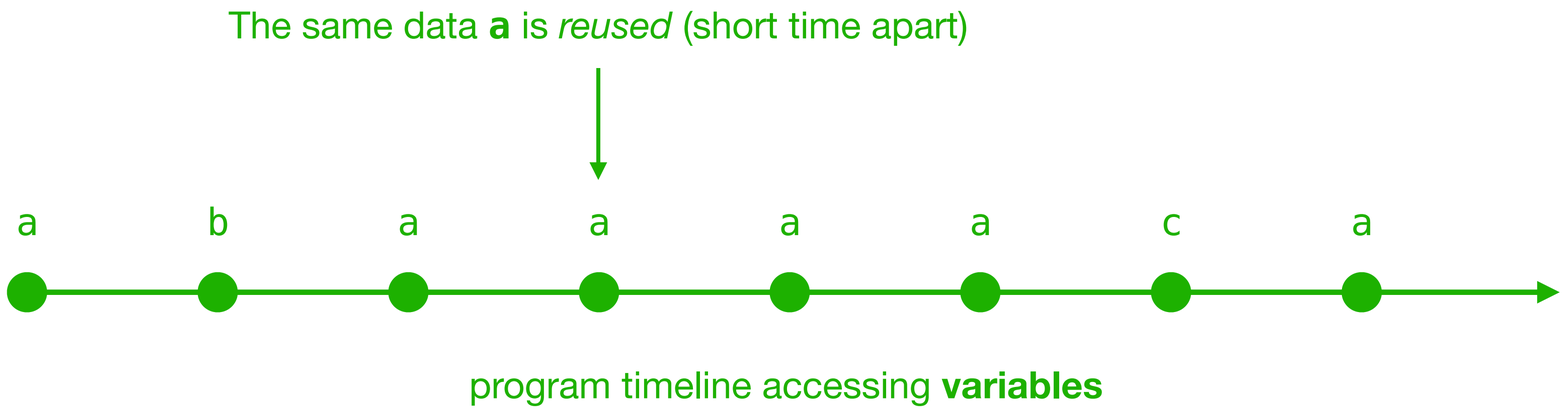
Locality, Locality, Locality

Locality Locality Locality

2 main categories!

If you ask for something, you're likely to ask for:

- The same piece of data again soon **temporal locality**

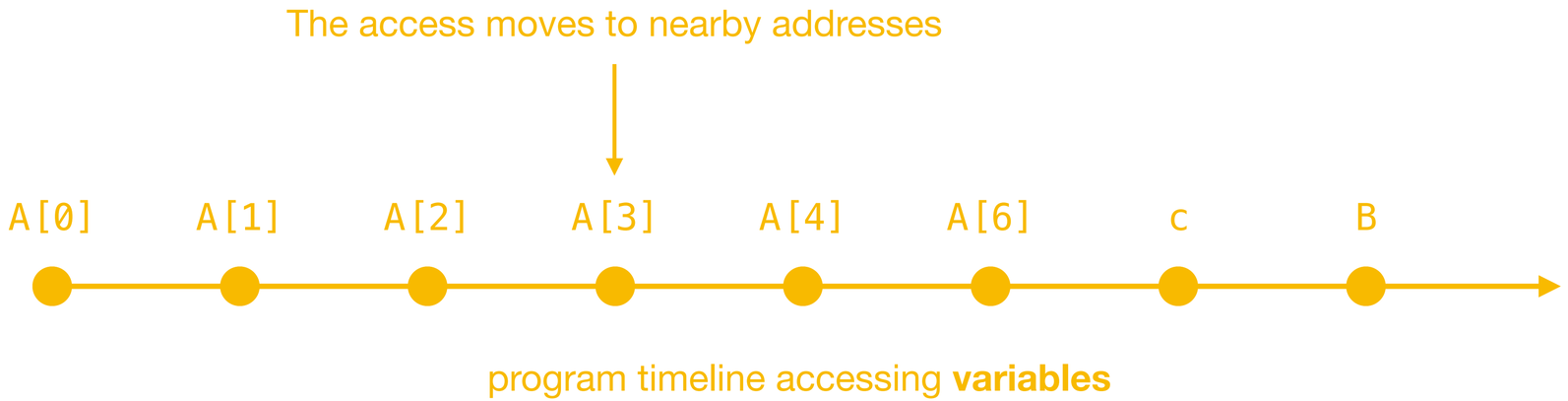


Locality Locality Locality

2 main categories!

If you ask for something, you're likely to ask for:

- Piece of data that's near the previous piece of data **spatial locality**



Locality (Errata Corrige)

Choose the variable that has good **spatial locality** and the one that has good **temporal locality:**

```
1 total = 0;
2 for (i = 1; i < n; i++)
{
3     n--;
4     total += a[i];
}
5 return total;
```

Variable	Temporal locality	Spatial locality
`total`	✅ High	❌ Low
`a[i]`	❌ Low	✅ High
`n`	✅ High	❌ Low
`i`	✅ High	❌ Low

`i`, `n`, and `total` all exhibit **temporal locality** because they are accessed every iteration of the loop, while `a[i]` instead shows **spatial locality** as we move sequentially through the array

Locality

C code:

```
int total = 0;
int temp;
int a[1000];

for (int i = 0; i < 1000; i++) {
    temp = i * 3;
    total += a[i * 10];
}
// 100 lines of code and temp is never used again
return total;
```

Variable	Temporal locality	Spatial locality
`total`	✅ High	❌ Low
`a[i * 10]`	❌ Low	❌ Low
`temp`	❌ Low	❌ Low
`i`	✅ High	❌ Low

`temp` has **bad** temporal locality because it is never reused beyond the loop, and the array has bad spatial locality because elements are accessed with a large stride instead of sequentially

Locality

C code:

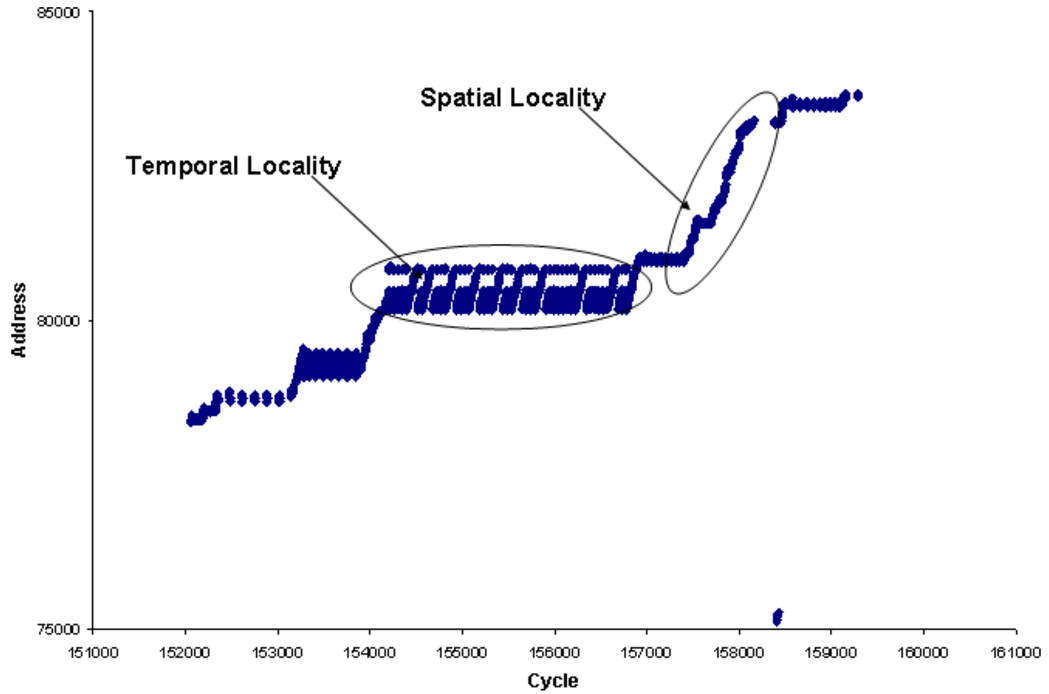
```
int total = 0;
int temp;
int a[1000];

for (int i = 0; i < 1000; i++) {
    temp = i * 3;
    total += a[i * 10];
}
// 100 lines of code and temp is never used again
return total;
```

Variable	Temporal locality	Spatial locality
`total`	✅ High	❌ Low
`a[i * 10]`	❌ Low	❌ Low
`temp`	❌ Low	❌ Low
`i`	✅ High	❌ Low

In the loop, `temp = i * 3` does have good temporal locality *as a variable access b*ut the **value** stored in `temp` has **bad** temporal locality

Locality

Locality is *not just about how often a variable appears*, but **about how the value is reused** over time or space **relative to the rest of the program**



https://stackoverflow.com/questions/7638932/what-is-locality-of-reference

Back to caches

Terminology

Cache hit

- Data is in the cache
- $\mathbf{t_{hit}}$ = time to access data in the cache
- **Hit Rate** (%) = # cache hits / # cache accesses

Cache miss

- Data is **not** in the cache
- $\mathbf{t_{miss}}$ = time to access and retrieve data
- **Miss Rate** (%) = # cache misses / # cache accesses

16 Byte Memory

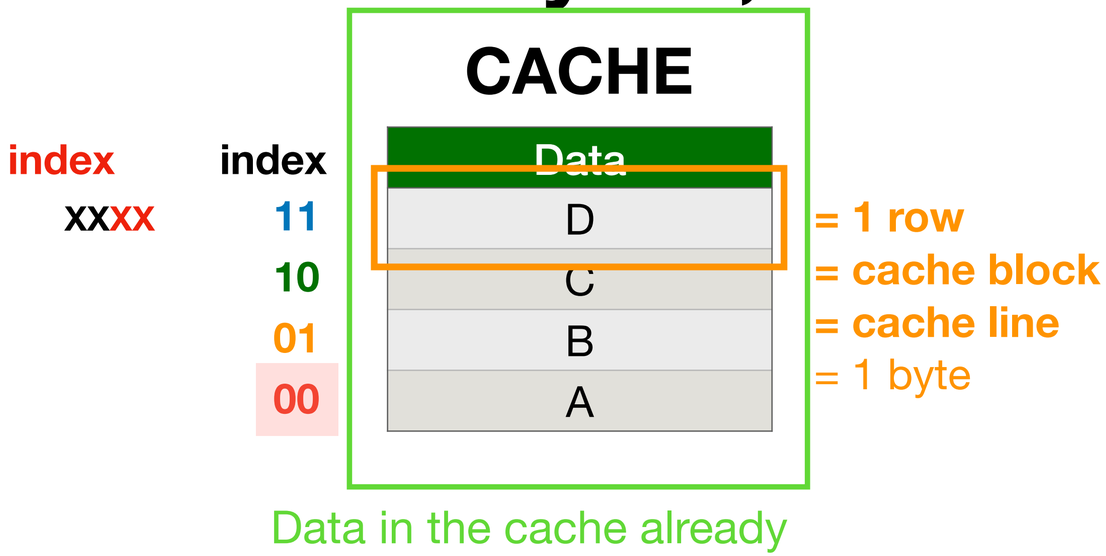
`load 1100 ⟶ r1`

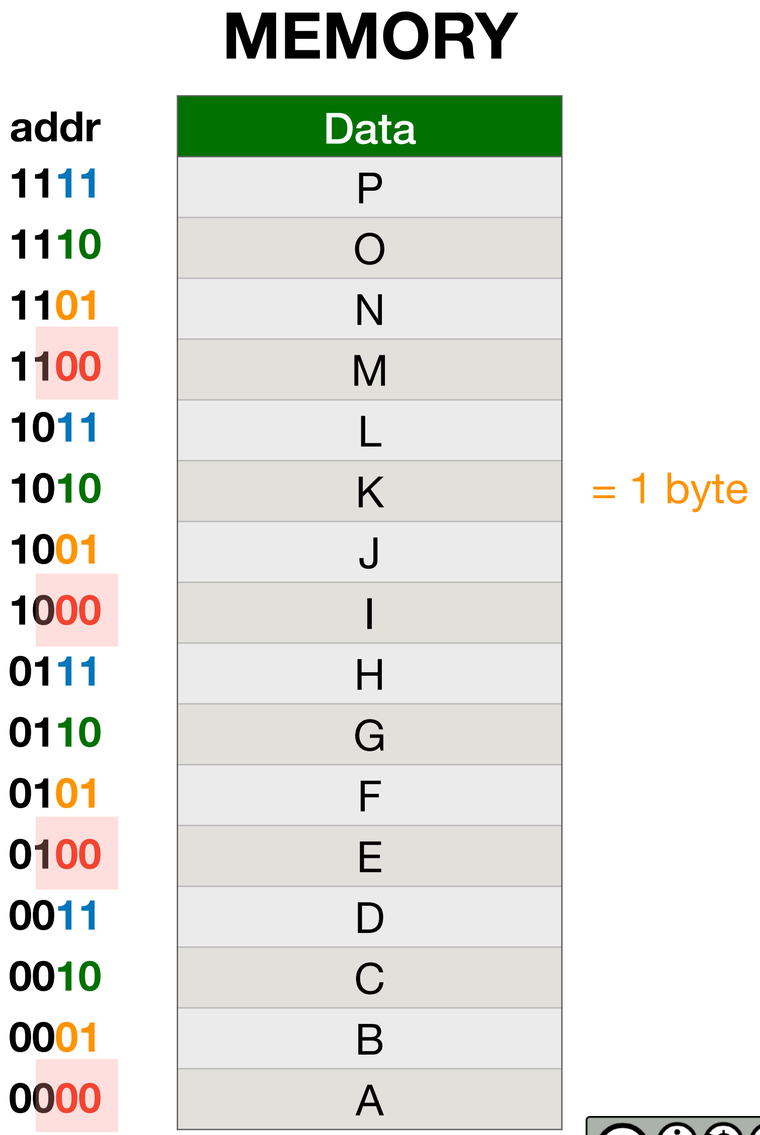
- **Byte-addressable** memory
- In this example, **4-bit address**

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

4 Bytes, Directed Mapped Cache





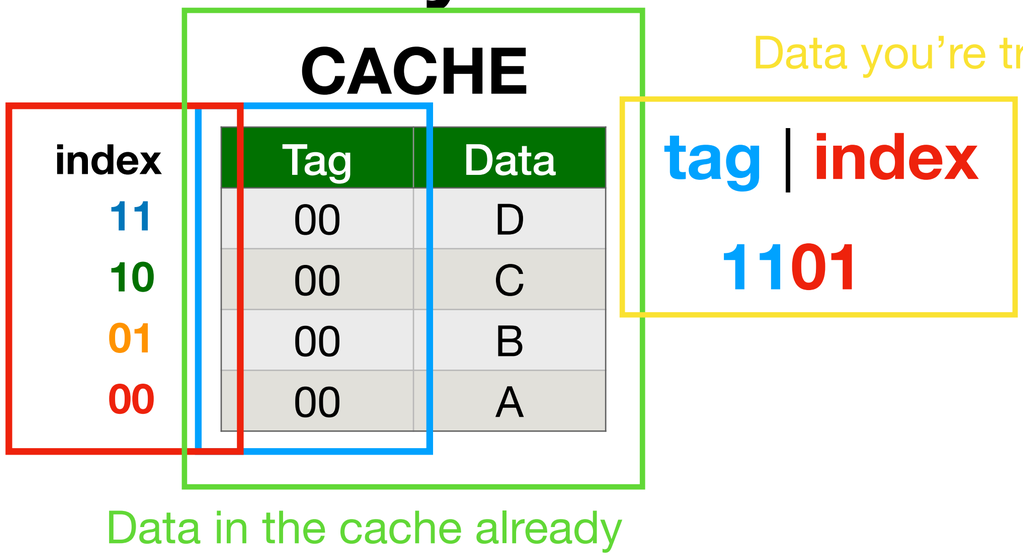
Direct mapped:

Each memory address has **exactly one possible location in the cache** where it can go—this makes lookups **very fast**

It's indexed with LSB: = least significant bit

Good **spatial** locality

4-Byte Directed Mapped Cache



Tag: minimalist label/address

```
address = tag + index
```

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

4-Byte Directed Mapped Cache

CACHE

index	V	Tag	Data
11	0	00	D
10	0	00	C
01	0	00	B
00	0	00	A

tag | index

1101

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

One last tweak: **valid bit**

The valid bit is a **single bit** in each cache line that indicates whether the data stored in that line is valid or not

Valid Bit = 1: The data in the cache line is **valid** and can be used

Valid Bit = 0: The data in the cache line is **invalid**, meaning it doesn't contain useful information and **cannot produce a cache hit**

the cache line is essentially considered **empty**

The access algorithm

CACHE

index	V	Tag	Data
11	0	00	D
10	0	00	C
01	0	00	B
00	0	00	A

Data in the cache already

Data you're trying to load in the register

tag | index

1101

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

1. Split the address between **tag** `t` and **index** `i`
2. Check the entry `i`
3. Is it valid? If no, cache miss!
4. If the bit is valid, then is it the tag `t`? If no, cache miss!
5. Otherwise, cache hit!

The access algorithm

CACHE

index	V	Tag	Data
11	0	00	D
10	0	00	C
01	0	00	B
00	0	00	A

Data in the cache already

Data you're trying to load in the register

tag | index

1101

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

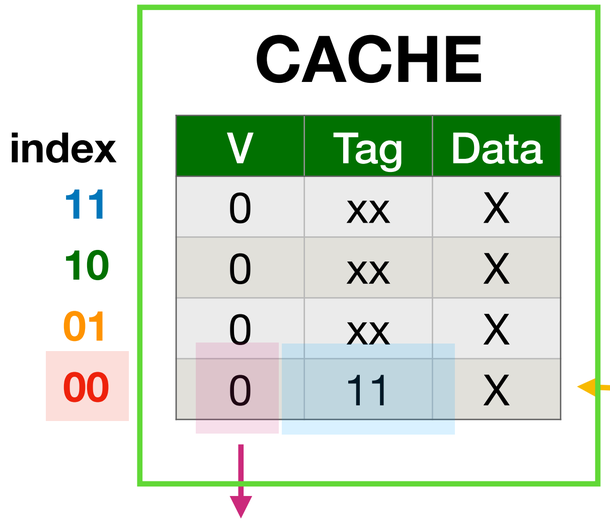
On cache miss, **fill** the cache:

1. Get data **d** from the main memory
2. Is entry valid? If so, **evict**
3. Then, set valid bit = 1, tag = **t**, data = **d** in entry **i**

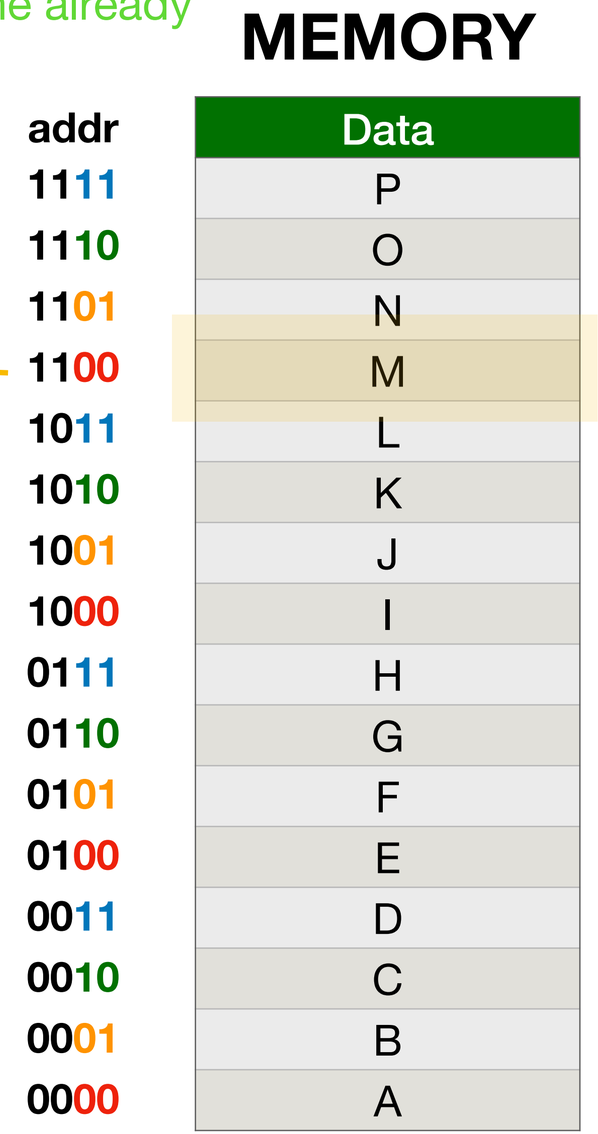
Ex: 4-Byte DM Cache

Data to load in the register

tag | index
1100



Data in the cache already




```
   t   i
load 1100 ⟶ r1
```

1. Check the **index**
2. Check the **valid bit**
3. Check the **tag**
4. Get data **d** from the main memory

Ex: 4-Byte DM Cache

Data to load in the register

tag | index
1100

Data in the cache already

CACHE

index	V	Tag	Data
11	0	xx	X
10	0	xx	X
01	0	xx	X
00	1	11	M

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

`load 1100 ⟶ r1` Cache miss! **M** from `1100` is now in the cache

Ex: 4-Byte DM Cache

Data to load in the register

tag | index
1100

Data in the cache already

CACHE

index	V	Tag	Data
11	0	xx	X
10	0	xx	X
01	0	xx	X
00	1	11	M

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

`load 1100 ⟶ r1` Cache miss! **M** from 1100 is now in the cache

…

`load 1100 ⟶ r1` Cache hit! **M** from 1100 is in the cache

1. Check the **index**
2. Check the **valid bit**
3. Check the **tag**

Ex: 4-Byte DM Cache

Data to load in the register

tag | index
1100

Cache **miss** or **hit**?

```
load 1100
load 1101
load 0100
load 1100
```

CACHE

index	V	Tag	Data
11	0	11	X
10	0	11	X
01	0	11	X
00	0	11	X

Data in the cache already

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

Ex: 4-Byte DM Cache

Data to load in the register

tag | index
1100

Data in the cache already

CACHE

index	V	Tag	Data
11	0	11	X
10	0	11	X
01	0	11	X
00	1	11	M

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

Cache **miss** or **hit**?

```
load 1100
load 1101
load 0100
load 1100
```

Cache miss! **M** from 1100 is now in the cache

Reducing cache misses by increasing block size

Increasing Block Size

Offset

XXXX

CACHE

index	V	Tag	Data
11	0	x	G | H
10	0	x	E | F
01	0	x	C | D
00	0	x	A | B

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

Block size: from 1 byte to 2 bytes

Ex: 8-Byte DM Cache

tag | index | offset

XXXX

CACHE

index	V	Tag	Data
11	0	x	X | X
10	0	x	X | X
01	0	x	X | X
00	0	x	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

```
load 1100
load 1101
load 0100
load 1100
```

1. Check the **index** in the $ (cache)
2. Check the **tag**
3. Check the **valid bit**

Ex: 8-Byte DM Cache

tag | index | offset

XXXX

CACHE

index	V	Tag	Data
11	0	x	X | X
10	1	1	M | N
01	0	x	X | X
00	0	x	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

```
load 1100
load 1101
load 0100
load 1100
```

Cache miss! **M** from 1100 is now in the cache together with **N** from 1101

Ex: 8-Byte DM Cache

tag | index | offset

XXXX

CACHE

index	V	Tag	Data
11	0	x	X | X
10	1	1	M | N
01	0	x	X | X
00	0	x	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

```
load 1100
load 1101
load 0100
load 1100
```

Cache miss! **M** from 1100 is now in the cache together with **N** from 1101

Ex: 8-Byte DM Cache

tag | index | offset

XXXX

CACHE

index	V	Tag	Data
11	0	x	X | X
10	1	1	M | N
01	0	x	X | X
00	0	x	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

load 1100 Cache miss! **M** from 1100 is now in the cache together with **N** from 1101

load 1101 Cache hit! **N** from 1101 is already in the cache! I moved it together with **M**

load 0100

load 1100

Ex: 8-Byte DM Cache

tag | index | offset

XXXX

CACHE

index	V	Tag	Data
11	0	x	X | X
10	1	0	E | F
01	0	x	X | X
00	0	x	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

`load 1100` Cache miss! **M** from 1100 is now in the cache together with **N** from 1101

`load 1101` Cache hit! **N** from 1101 is already in the cache! I moved it together with **M**

`load 0100`

`load 1100`

Ex: 8-Byte DM Cache

tag | index | offset

XXXX

CACHE

index	V	Tag	Data
11	0	x	X | X
10	1	0	E | F
01	0	x	X | X
00	0	x	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

`load 1100` Cache miss! **M** from 1100 is now in the cache together with **N** from 1101

`load 1101` Cache hit! **N** from 1101 is already in the cache! I moved it together with **M**

`load 0100` Cache miss! I evicted **M** and **N. E** from 0100 is now in the cache with **F** from 0101

`load 1100`

Ex: 8-Byte DM Cache

tag | index | offset

XXXX

CACHE

index	V	Tag	Data
11	0	x	X | X
10	1	1	M | N
01	0	x	X | X
00	0	x	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

load 1100 Cache miss! **M** from 1100 is now in the cache together with **N** from 1101

load 1101 Cache hit! **N** from 1101 is already in the cache! I moved it together with **M**

load 0100 Cache miss! I evicted **M** and **N. E** from 0100 is now in the cache with **F** from 0101

load 1100

Ex: 8-Byte DM Cache

tag | index | offset

XXXX

CACHE

index	V	Tag	Data
11	0	x	X | X
10	1	1	M | N
01	0	x	X | X
00	0	x	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

load 1100 Cache miss! **M** from 1100 is now in the cache together with **N** from 1101

load 1101 Cache hit! **N** from 1101 is already in the cache! I moved it together with **M**

load 0100 Cache miss! I evicted **M** and **N. E** from 0100 is now in the cache with **F** from 0101

load 1100 Cache miss! I evicted **E** and **F. M** from 1100 is now in the cache with **N** from 1101

Ex: 8-Byte DM Cache

tag | index | offset

XXXX

CACHE

index	V	Tag	Data
11	0	x	X | X
10	1	1	M | N
01	0	x	X | X
00	0	x	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

`load 1100` Cache miss! **M** from 1100 is now in the cache together with **N** from 1101

`load 1101` Cache hit! **N** from 1101 is already in the cache! I moved it together with **M**

`load 0100` Cache miss! I evicted **M** and **N. E** from 0100 is now in the cache with **F** from 0101

`load 1100` Cache miss! I evicted **E** and **F. M** from 1100 is now in the cache with **N** from 1101

So far, we looked at read misses, now on to store misses

Store Miss

A **store miss** happens when the CPU tries to write data to an address that is not currently in the cache

- The behavior on a store miss depends on the **cache's write policy**

On a store miss, do we fill the block?

- Yes, **write-allocate**

On a store miss, do we immediately write to main memory or just write to the cache?

- The former: **write-through**
- The latter: **write-back**

Write-Back

Cache can get out of synch with the main memory

- I need to add a **dirty bit**

Dirty Bit = 0 (clean) ⟶ in sync with the memory

Dirty Bit = 1 (dirty) ⟶ possibly out of sync with the memory

On a store miss, do we immediately write to main memory or just write to the cache?

- The former: **write-through**
- The latter: **write-back**

Write-Back

Cache can get out of synch with the main memory

- I need to add a **dirty bit**

Dirty Bit = 0 (clean) ⟶ in sync with the memory

Dirty Bit = 1 (dirty) ⟶ possibly out of sync with the memory

CACHE

index	V	D	Tag	Data
11	0	0	x	X | X
10	1	1	0	A | B
01	0	0	x	X | X
00	0	0	x	X | X

The access algorithm

Cache can get out of synch with the main memory

- I need to add a **dirty bit**

Dirty Bit = 0 (clean) ⟶ in sync with the memory

Dirty Bit = 1 (dirty) ⟶ possibly out of sync with the memory

1. Split the address between **tag** `t` and **index** `i`
2. Check the entry `i`
3. Is it valid? If no, cache miss!
4. Is it the tag `t`? If no, cache miss!
5. Otherwise, cache hit!
6. If **store**, set **dirty = 1**

On miss, **fill** the cache:

1. Get data **d** from the main memory
2. Is entry valid? If so, **evict**
3. If **dirty = 1**, then **write-back**
4. Then, set valid bit = 1, tag = `t`, data = **d** in entry `i`, **dirty = 0**

Ex: 8-Byte DM Cache

tag | index | offset

XXXX

CACHE

index	V	D	Tag	Data
11	0	0	x	X | X
10	0	0	x	X | X
01	0	0	x	X | X
00	0	0	x	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

```
load  1100
store 1101
load  0100
load  1100
```

1. Split the address between **tag** `t` and **index** `i`
2. Check the entry `i`
3. Is it valid? If no, cache miss!

Ex: 8-Byte DM Cache

tag | index | offset

XXXX

CACHE

index	V	D	Tag	Data
11	0	0	x	X | X
10	0	0	x	X | X
01	0	0	x	X | X
00	0	0	x	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

```
load  1100
store 1101
load  0100
load  1100
```

On miss, **fill** the cache:

1. Get data **d** from the main memory
2. Is entry valid? If so, **evict**
3. If **dirty = 1**, then **write-back**

Ex: 8-Byte DM Cache

tag | index | offset

XXXX

CACHE

index	V	D	Tag	Data
11	0	0	x	X | X
10	1	0	1	M | N
01	0	0	x	X | X
00	0	0	x	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

```
load  1100
store 1101
load  0100
load  1100
```

On miss, **fill** the cache:

1. Get data **d** from the main memory
2. Is entry valid? If so, **evict**
3. If **dirty = 1**, then **write-back**
4. Then, set valid bit = 1, tag = **t**, data = **d** in entry **i**, **dirty = 0**

Ex: 8-Byte DM Cache

tag | index | offset

XXXX

CACHE

index	V	D	Tag	Data
11	0	0	x	X | X
10	1	0	1	M | N
01	0	0	x	X | X
00	0	0	x	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

Let us assume `store` wants to put **Q** in 1101

```
load   1100
store  1101
load   0100
load   1100
```

1. Split the address between **tag** `t` and **index** `i`
2. Check the entry `i`
3. Is it valid? Yes, cache hit!

Ex: 8-Byte DM Cache

tag | index | offset

XXXX

CACHE

index	V	D	Tag	Data
11	0	0	x	X | X
10	1	1	1	M | Q
01	0	0	x	X | X
00	0	0	x	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

Let us assume `store` wants to put **Q** in 1101

```
load   1100
store  1101
load   0100
load   1100
```

1. Split the address between **tag** `t` and **index** `i`
2. Check the entry `i`
3. Is it valid? Yes, cache hit!
4. If **store**, set **dirty = 1**

Ex: 8-Byte DM Cache

tag | index | offset

XXXX

CACHE

index	V	D	Tag	Data
11	0	0	x	X | X
10	1	1	1	M | Q
01	0	0	x	X | X
00	0	0	x	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

```
load  1100
store 1101
load  0100
load  1100
```

1. Split the address between **tag** `t` and **index** `i`
2. Check the entry `i`
3. Is it valid? If not, cache miss!

Ex: 8-Byte DM Cache

tag | index | offset

XXXX

CACHE

index	V	D	Tag	Data
11	0	0	x	X | X
10	1	1	1	M | Q
01	0	0	x	X | X
00	0	0	x	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

```
load  1100
store 1101
load  0100
load  1100
```

On miss, **fill** the cache:

1. Get data **d** from the main memory
2. Is entry valid? If so, **evict**

Ex: 8-Byte DM Cache

tag | index | offset

XXXX

CACHE

index	V	D	Tag	Data
11	0	0	x	X | X
10 →	1	1	1	M | Q
01	0	0	x	X | X
00	0	0	x	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	Q
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100 →	E
0011	D
0010	C
0001	B
0000	A

```
load  1100
store 1101
load  0100
load  1100
```

On miss, **fill** the cache:

1. Get data **d** from the main memory
2. Is entry valid? If so, **evict**
3. If **dirty = 1**, then **write-back**

Ex: 8-Byte DM Cache

tag | index | offset

XXXX

CACHE

index	V	D	Tag	Data
11	0	0	x	X | X
10	1	0	0	E | F
01	0	0	x	X | X
00	0	0	x	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	Q
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

```
load  1100
store 1101
load  0100
load  1100
```

On miss, **fill** the cache:

1. Get data **d** from the main memory
2. Is entry valid? If so, **evict**
3. If **dirty = 1**, then **write-back**
4. Then, set valid bit = 1, tag = **t**, data = **d** in entry **i**, **dirty = 0**

Ex: 8-Byte DM Cache

tag | index | offset

XXXX

CACHE

index	V	D	Tag	Data
11	0	0	x	X | X
10	1	0	0	E | F
01	0	0	x	X | X
00	0	0	x	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	Q
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

```
load   1100
store  1101
load   0100
load   1100
```

1. Split the address between **tag** `t` and **index** `i`
2. Check the entry `i`
3. Is it valid? Yes, check tag
4. Is the tag `t`? If not, cache miss

Ex: 8-Byte DM Cache

tag | index | offset

XXXX

CACHE

index	V	D	Tag	Data
11	0	0	x	X | X
10	1	0	0	E | F
01	0	0	x	X | X
00	0	0	x	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	Q
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

```
load  1100
store 1101
load  0100
load  1100
```

On miss, **fill** the cache:

1. Get data **d** from the main memory
2. Is entry valid? If so, **evict**
3. If **dirty = 1**, then **write-back**

Ex: 8-Byte DM Cache

tag | index | offset

XXXX

CACHE

index	V	D	Tag	Data
11	0	0	x	X | X
10	1	0	1	M | Q
01	0	0	x	X | X
00	0	0	x	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	Q
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

```
load  1100
store 1101
load  0100
load  1100
```

On miss, **fill** the cache:

1. Get data **d** from the main memory
2. Is entry valid? If so, **evict**
3. If **dirty = 1**, then **write-back**
4. Then, set valid bit = 1, tag = **t**, data = **d** in entry **i**, **dirty = 0**

On to fully associative cache design

Fully Associative Cache

CACHE

V	Tag	Data
0	xxx	X | X
1	110	M | N
0	xxx	X | X
0	xxx	X | X

tag | offset

xxxx

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

Fully associative design:

The index is **no longer relevant** at all; **every cache entry could hold any address.**

The address is split in **tag** + **byte offset** (no index)

The access algorithm

CACHE

V	Tag	Data
0	xxx	X | X
1	110	M | N
0	xxx	X | X
0	xxx	X | X

tag | **offset**

xxxx

1. Check every every cache line's tag **in parallel**
2. If not valid OR not tag **t**, try other indeces
3. If no match (cache miss), **evict** some line and load new block

~~1~~ Split the address between **tag t** and **index i**

~~2~~ Check the entry **i**

3. Is it valid? If no, ~~cache~~ miss!
4. Is it the tag **t**? If no, ~~cache~~ miss!
4. Otherwise, cache hit!

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

Replacement Policy

In an associative cache, you need to **pick which block to evict**:

- Least Recentely Used (LRU)
- Not Most Recently Used (NMRU)

```
load 1100
load 1101
load 0100
load 1100
```

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

Ex: 8-Byte DM Cache

Replacement Policy:

- Least Recentely Used (LRU)

tag | **offset**

xxxx

CACHE

V	Tag	Data
0	xxx	X | X
0	xxx	X | X
0	xxx	X | X
0	xxx	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

1. Check every every cache line's tag **in parallel**
2. If no valid bit or no tag **t**, then evict and load new line

```
load 1100
load 1101
load 0100
load 1100
```

Ex: 8-Byte DM Cache

Replacement Policy:

- Least Recentely Used (LRU)

tag | offset

xxxx

CACHE

V	Tag	Data
1	110	M | N
0	xxx	X | X
0	xxx	X | X
0	xxx	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

1. Check every every cache line's tag **in parallel**
2. If no valid bit or no tag `t`, then evict and load new line

```
load 1100
load 1101
load 0100
load 1100
```

Cache miss! **M** from `1100` is now in the cache together with **N** from `1101`

Ex: 8-Byte DM Cache

Replacement Policy:

- Least Recentely Used (LRU)

tag | offset

xxxx

CACHE

V	Tag	Data
1	110	M | N
0	xxx	X | X
0	xxx	X | X
0	xxx	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

1. Check every every cache line's tag **in parallel**
2. If no valid bit or no tag **t**, then evict and load new line

load 1100 Cache miss! **M** from 1100 is now in the cache together with **N** from 1101

load 1101

load 0100

load 1100

Ex: 8-Byte DM Cache

Replacement Policy:

- Least Recentely Used (LRU)

tag | **offset**

xxxx

CACHE

V	Tag	Data
1	110	M | N
0	xxx	X | X
0	xxx	X | X
0	xxx	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

1. Check every every cache line's tag **in parallel**
2. If no valid bit or no tag **t**, then evict and load new line

load 1100 Cache miss! **M** from 1100 is now in the cache together with **N** from 1101

load 1101 Cache hit! **N** from 1101 is already in the cache! I moved it together with **M**

load 0100

load 1100

Ex: 8-Byte DM Cache

Replacement Policy:

- Least Recentely Used (LRU)

tag | **offset**

xxxx

CACHE

V	Tag	Data
1	110	M | N
0	xxx	X | X
0	xxx	X | X
0	xxx	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

1. Check every every cache line's tag **in parallel**
2. If no valid bit or no tag **t**, then evict and load new line

load 1100 Cache miss! **M** from 1100 is now in the cache together with **N** from 1101

load 1101 Cache hit! **N** from 1101 is already in the cache! I moved it together with **M**

load 0100

load 1100

Ex: 8-Byte DM Cache

Replacement Policy:

- Least Recentely Used (LRU)

tag | **offset**

xxxx

CACHE

V	Tag	Data
1	110	M | N
1	010	E | F
0	xxx	X | X
0	xxx	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

1. Check every every cache line's tag **in parallel**
2. If no valid bit or no tag **t**, then evict and load new line

load 1100 Cache miss! **M** from 1100 is now in the cache together with **N** from 1101

load 1101 Cache hit! **N** from 1101 is already in the cache! I moved it together with **M**

load 0100 Cache miss! **E** from 0100 is now in the cache together with **F** from 0101

load 1100

Ex: 8-Byte DM Cache

Replacement Policy:

- Least Recentely Used (LRU)

tag | **offset**

xxxx

CACHE

V	Tag	Data
1	110	M | N
1	010	E | F
0	xxx	X | X
0	xxx	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

1. Check every every cache line's tag **in parallel**
2. If no valid bit or no tag **t**, then evict and load new line

load 1100 Cache miss! **M** from 1100 is now in the cache together with **N** from 1101

load 1101 Cache hit! **N** from 1101 is already in the cache! I moved it together with **M**

load 0100 Cache miss! **E** from 0100 is now in the cache together with **F** from 0101

load 1100

Ex: 8-Byte DM Cache

Replacement Policy:

- Least Recentely Used (LRU)

tag | **offset**

xxxx

CACHE

V	Tag	Data
1	110	M | N
1	010	E | F
0	xxx	X | X
0	xxx	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

1. Check every every cache line's tag **in parallel**
2. If no valid bit or no tag **t**, then evict and load new line

load 1100 Cache miss! **M** from 1100 is now in the cache together with **N** from 1101

load 1101 Cache hit! **N** from 1101 is already in the cache! I moved it together with **M**

load 0100 Cache miss! **E** from 0100 is now in the cache together with **F** from 0101

load 1100 Cache hit! **M** from 1101 is **still** in the cache!

On to set associative cache design (middleground)

Set Associative Cache

CACHE

WAY 0

index or set	V	Tag	Data
0	0	xx	X | X
1	0	xx	X | X

WAY 1

V	Tag	Data
0	xx	X | X
0	xx	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

Set associative design:

Divide the storage into sets of size 2^k
The cache has 2^k ways:

There are: $2^n / 2^k = 2^{n-k}$ sets

tag | index | offset

XXXX

Each set has one index; do the “associative” thing within each set

Other designs are special cases:

- Direct mapped: k = 0
- Fully associative: k = n

The access algorithm

CACHE

WAY 0

index or set	V	Tag	Data
0	0	xx	X | X
1	0	xx	X | X

WAY 1

V	Tag	Data
0	xx	X | X
0	xx	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

1. Check every cache line **in the set in parallel**

~~1~~ Split the address between **tag** `t` and **index** `i`

~~2~~ Check the entry `i`

3. Is it valid? If no, ~~cache~~ miss!
4. Is it the tag `t`? If no, ~~cache~~ miss!
4. Otherwise, cache hit!

`tag` | `index` | offset

XXXX

2. If not valid OR not tag `t`, try other indeces **in the set**
3. If no match (cache miss), **evict** some line **in the set** and load new block

tag | index | offset

XXXX

Set Associative Cache

CACHE

WAY 0

index or set	V	Tag	Data
0	0	xx	X | X
1	0	xx	X | X

WAY 1

V	Tag	Data
0	xx	X | X
0	xx	X | X

1. Check every cache line **in the set in parallel**
2. If no valid bit or no tag **t**, then evict **from the set** and load new line

```
load 1100
load 1101
load 0100
load 1100
```

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

tag | index | offset

XXXX

Set Associative Cache

CACHE

WAY 0

index or set	V	Tag	Data
0	1	11	M | N
1	0	xx	X | X

WAY 1

V	Tag	Data
0	xx	X | X
0	xx	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

1. Check every cache line **in the set in parallel**
2. If no valid bit or no tag `t`, then evict **from the set** and load new line

```
load 1100
load 1101
load 0100
load 1100
```

Cache miss! **M** from `1100` is now in the cache together with **N** from `1101`

tag | index | offset

XXXX

Set Associative Cache

CACHE

WAY 0

index or set	V	Tag	Data
0	1	11	M | N
1	0	xx	X | X

WAY 1

V	Tag	Data
0	xx	X | X
0	xx	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

1. Check every cache line **in the set in parallel**
2. If no valid bit or no tag `t`, then evict **from the set** and load new line

```
load 1100
```
Cache miss! **M** from `1100` is now in the cache together with **N** from `1101`

```
load 1101
```
Cache hit! **N** from `1101` is already in the cache! I moved it together with **M**

```
load 0100
load 1100
```

tag | index | offset

XXXX

Set Associative Cache

CACHE

WAY 0

index or set	V	Tag	Data
0	1	11	M | N
1	0	xx	X | X

WAY 1

V	Tag	Data
0	xx	X | X
0	xx	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

1. Check every cache line **in the set in parallel**
2. If no valid bit or no tag `t`, then evict **from the set** and load new line

`load 1100` Cache miss! **M** from `1100` is now in the cache together with **N** from `1101`

`load 1101` Cache hit! **N** from `1101` is already in the cache! I moved it together with **M**

`load 0100`

`load 1100`

tag | index | offset

XXXX

Set Associative Cache

CACHE

WAY 0

index or set	V	Tag	Data
0	1	11	M | N
1	0	xx	X | X

WAY 1

V	Tag	Data
1	01	E | F
0	xx	X | X

1. Check every cache line **in the set in parallel**
2. If no valid bit or no tag `t`, then evict **from the set** and load new line

`load 1100` Cache miss! **M** from `1100` is now in the cache together with **N** from `1101`

`load 1101` Cache hit! **N** from `1101` is already in the cache! I moved it together with **M**

`load 0100` Cache miss! **E** from `0100` is now in the cache together with **F** from `0101`

`load 1100`

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

tag | index | offset

XXXX

Set Associative Cache

CACHE

WAY 0

index or set	V	Tag	Data
0	1	11	M | N
1	0	xx	X | X

WAY 1

V	Tag	Data
1	01	E | F
0	xx	X | X

1. Check every cache line **in the set in parallel**
2. If no valid bit or no tag `t`, then evict **from the set** and load new line

`load 1100` Cache miss! **M** from `1100` is now in the cache together with **N** from `1101`

`load 1101` Cache hit! **N** from `1101` is already in the cache! I moved it together with **M**

`load 0100` Cache miss! **E** from `0100` is now in the cache together with **F** from `0101`

`load 1100` Cache hit! **M** from `1101` is **still** in the cache!

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

tag | index | offset

XXXX

Set Associative Cache

CACHE

WAY 0

index or set	V	Tag	Data
0	1	11	M | N
1	0	xx	X | X

WAY 1

V	Tag	Data
1	01	E | F
0	xx	X | X

1. Check every cache line **in the set in parallel**
2. If no valid bit or no tag **t**, then evict **from the set** and load new line

load 1100 Cache miss! **M** from 1100 is now in the cache together with **N** from 1101

load 1101 Cache hit! **N** from 1101 is already in the cache! I moved it together with **M**

load 0100 Cache miss! **E** from 0100 is now in the cache together with **F** from 0101

load 1100 Cache hit! **M** from 1101 is **still** in the cache!

load 1010

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

tag | index | offset

XXXX

Set Associative Cache

CACHE

WAY 0

index or set	V	Tag	Data
0	1	11	M | N
1	1	10	K | L

WAY 1

V	Tag	Data
1	01	E | F
0	xx	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

1. Check every cache line **in the set in parallel**
2. If no valid bit or no tag `t`, then evict **from the set** and load new line

`load 1100` Cache miss! **M** from `1100` is now in the cache together with **N** from `1101`

`load 1101` Cache hit! **N** from `1101` is already in the cache! I moved it together with **M**

`load 0100` Cache miss! **E** from `0100` is now in the cache together with **F** from `0101`

`load 1100` Cache hit! **M** from `1101` is **still** in the cache!

`load 1010` Cache miss! **K** from `1010` is now in the cache together with **L** from `1011`

tag | index | offset

XXXX

Set Associative Cache

CACHE

WAY 0

index or set	V	Tag	Data
0	1	11	M | N
1	0	xx	X | X

WAY 1

V	Tag	Data
1	01	E | F
0	xx	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

1. Check every cache line **in the set in parallel**
2. If no valid bit or no tag `t`, then evict **from the set** and load new line

`load 1100` Cache miss! **M** from `1100` is now in the cache together with **N** from `1101`

`load 1101` Cache hit! **N** from `1101` is already in the cache! I moved it together with **M**

`load 0100` Cache miss! **E** from `0100` is now in the cache together with **F** from `0101`

`load 1100` Cache hit! **M** from `1101` is **still** in the cache!

`load 1000`

tag | index | offset

XXXX

Set Associative Cache

CACHE

WAY 0

index or set	V	Tag	Data
0	1	10	I | J
1	0	xx	X | X

WAY 1

V	Tag	Data
1	01	E | F
0	xx	X | X

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

1. Check every cache line **in the set in parallel**
2. If no valid bit or no tag `t`, then evict **from the set** and load new line

`load 1100` Cache miss! **M** from `1100` is now in the cache together with **N** from `1101`

`load 1101` Cache hit! **N** from `1101` is already in the cache! I moved it together with **M**

`load 0100` Cache miss! **E** from `0100` is now in the cache together with **F** from `0101`

`load 1100` Cache hit! **M** from `1101` is **still** in the cache!

`load 1000` Cache miss! I need to **evict** one line (LRU); then load **I** from `1000` in the cache together with **J** from `1001`

Cache performance

Cache performance

The average access time t_{avg}:

$t_{avg} = t_{hit} + \%_{miss} * t_{miss}$

$t_{avg} = 4 + 5\% * 100$

$t_{avg} = 9$ cycles

The average access time t_{avg}:

$t_{avg} = t_{hit} + \%_{miss} * t_{miss}$

$t_{avg} = 1\text{ ns} + 5\% * 50\text{ ns}$

$t_{avg} = 3.5$ ns

Three types of cache misses (3 Cs):

- **Cold** or **Compulsory:** first access ever to a block
- **Capacity:** the cache is too small
- **Conflict:** mapping collision (esp. direct mapped), the associativity is too low