You <u>must</u> be logged in PollEv to get credit

Participation

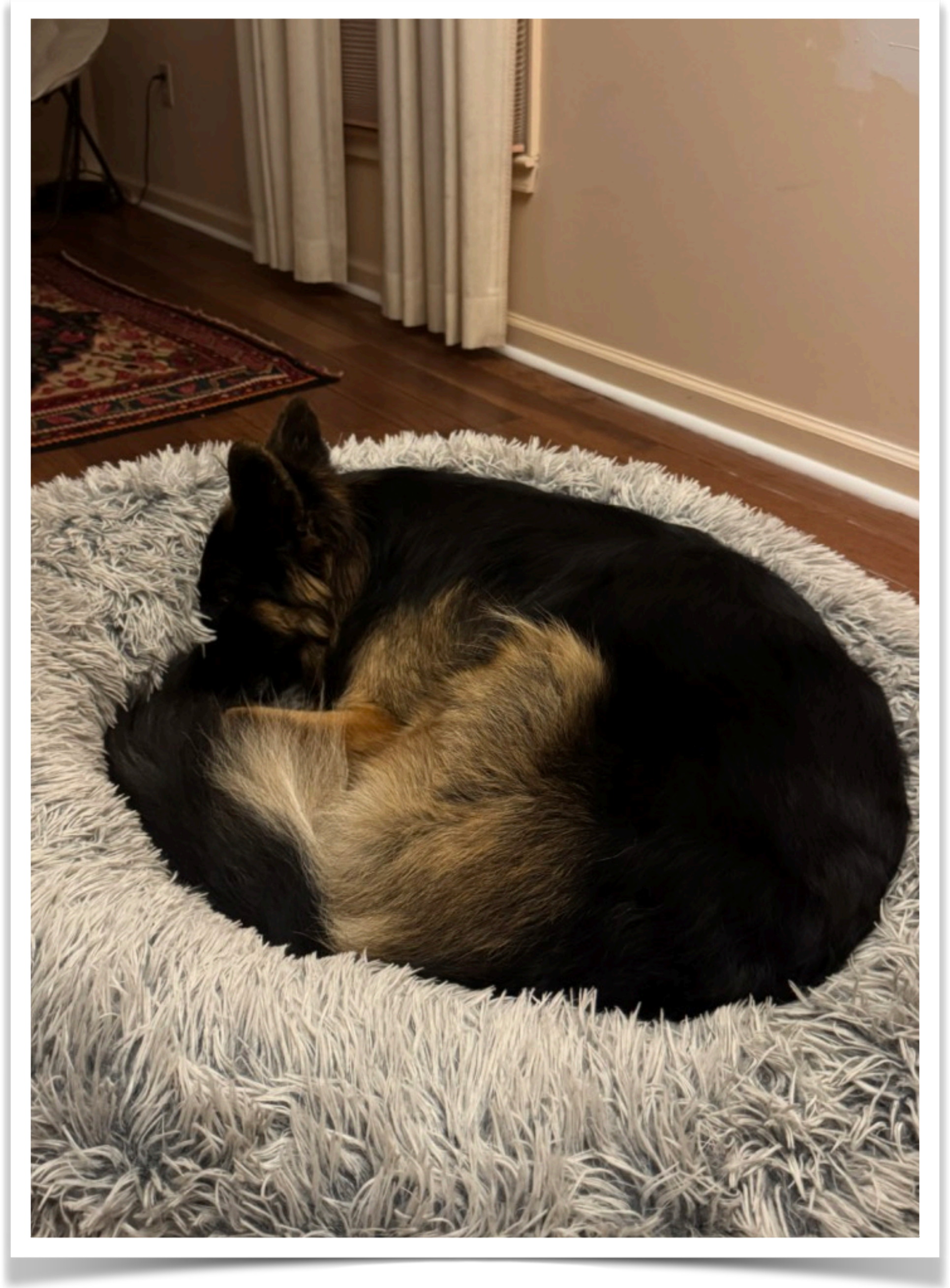
The “participation” segment of your grade has three main components:

- 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.

BY NC SA

Cornell Bowers CIS
Computer Science



CS3410: Computer Systems and Organization

LEC17: Caches (Vol. III)

Professor Giulia Guidi
Monday, October 27, 2025

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

BY NC SA

Plan for Today

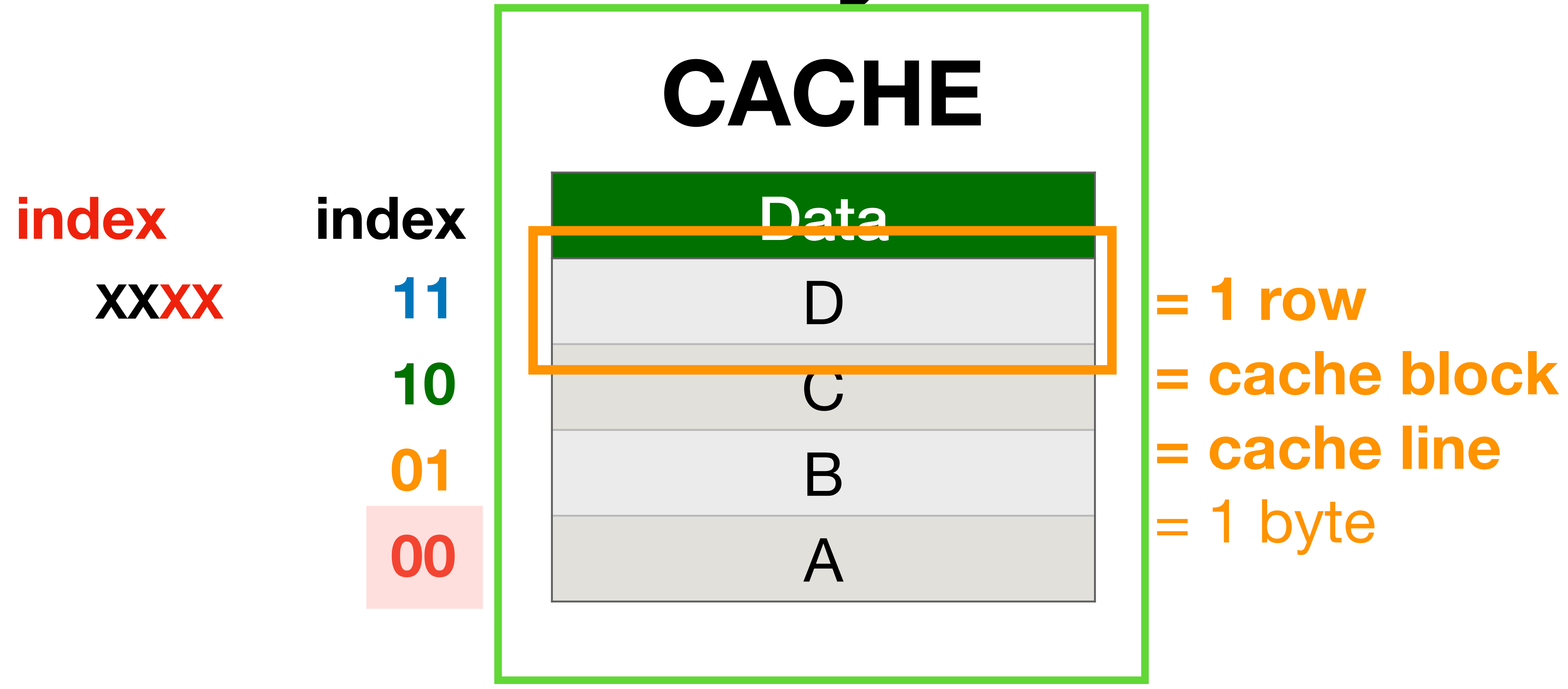
- Review of direct mapped cache
- Cache design: fully associative and set associative

Updated assignment schedule (see Ed and course website)

A9 now due Nov 5

Direct mapped cache design

4-Byte Directed Mapped Cache



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

= 1 byte

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

BY NC SA

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!

CC BY NC SA

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**

CC BY NC SA

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	0	11	X

It's a **cache miss**!

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

```
  t i
load 1100
```

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

cc BY NC SA

Ex: 4-Byte DM Cache

Data to load in the register

tag | index

1100

CACHE

Data in the cache already

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
load 1101
load 0100
load 1100
```

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

cc BY NC SA

Ex: 4-Byte DM Cache

Data to load in the register

tag | index

xxxx

CACHE

Data in the cache already

index	V	Tag	Data
11	0	xx	X
10	0	xx	X
01	1	11	N
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` Cache miss! **M** from `1100` is now in the cache

`load 1101` Cache miss! **N** from `1101` is now in the cache

`load 0100` Cache miss! Evict **M** and fill that line with **E** from `0100`

`load 1100` Cache miss! Evict **E** and fill that line with **M** from `1100`

CC BY NC SA

Reducing cache misses by increasing block size

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**

CC BY NC SA

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

CC BY NC SA

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`

CC BY NC SA

On to store misses

The write-back 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**

If write-through the algorithm for directed mapped cache is the same as slides 6-7

CC BY NC SA

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
```

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

cc BY NC SA

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
```

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**

CC BY NC SA

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
```

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**

CC BY NC SA

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
```

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

CC BY NC SA

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
```

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**

cc BY NC SA

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
```

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

CC BY NC SA

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
```

On miss, **fill** the cache:

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

CC BY NC SA

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
```

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**

cc BY NC SA

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
```

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**

CC BY NC SA

Core Ideas and Challenges

One line per address. One chance.

A **direct mapped cache** is like an assigned seat on the plane:

- If there are empty seats, you **must** still sit in your assigned one

Good things:

- It's energy efficient
- The hardware is simple
- The lookup is super fast

CC BY NC SA

Core Ideas and Challenges

One line per address. One chance.

A **direct mapped cache** is like an assigned seat on the plane:

- If there are empty seats, you **must** still sit in your assigned one

Bad things:

- Conflict misses: if two **hot** addresses map to the **exact same** line → **thrash city!**
  - Cache **thrashing** is a thing:
    - You access A → evict B → then, access B → evict A → repeat until sanity is lost
- It can lead to trashing even with **good locality**

cc BY NC SA

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:

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

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

A **fully associative cache** is like open seating on Southwest Airlines:

- Sit wherever you want

cc BY NC SA

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**

~~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!

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

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

CC BY NC SA

Replacement Policy

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

- Least Recently 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

cc BY NC SA

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

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
```

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

cc BY NC SA

Ex: 8-Byte DM Cache

Replacement Policy:

- Least Recently Used (LRU)

tag | offset

XXXX

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`

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

CC BY NC SA

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`

cc BY NC SA

Ex: 8-Byte DM Cache

Replacement Policy:

- Least Recentely Used (LRU)

tag | offset

XXXX

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`

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

CC BY NC SA

Ex: 8-Byte DM Cache

Replacement Policy:

- Least Recently 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`

cc BY NC SA

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`

CC BY NC SA

Ex: 8-Byte DM Cache

Replacement Policy:

- Least Recently 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`

cc BY NC SA

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!

CC BY NC SA

The cost of associativity:

In direct mapped, you look at **one entry**. In fully associative you look at **every entry** but fewer evictions!

On to set associative cache design (middleground)

BY NC SA

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

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

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

tag | index | 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

CC BY NC SA

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**

- ~~Split the address between **tag** `t` and **index** `i`~~
- ~~Check the entry `i`~~

tag | index | offset

XXXX

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

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

CC BY NC SA

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

cc BY NC SA

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

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
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

cc BY NC SA

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

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`

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

cc BY NC SA

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

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`

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

cc BY NC SA

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			WAY 1		
index or set	V	Tag	Data	V	Tag	Data
0	1	11	M | N	1	01	E | F
1	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` 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!

CC BY NC SA

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

cc BY NC SA

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

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`

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

cc BY NC SA

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`

CC BY NC SA

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`

cc BY NC SA

Cache performance

Cache performance

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

BY NC SA

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}:

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

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

CC BY NC SA