Week 10

CS110P.07

Suting Chen

Cache

- Introduction to cache （你们一定没翘课吧？）
- How is cache indexed?
- Associativity
- Replacement Policy
- Measure performance

Suting Chen

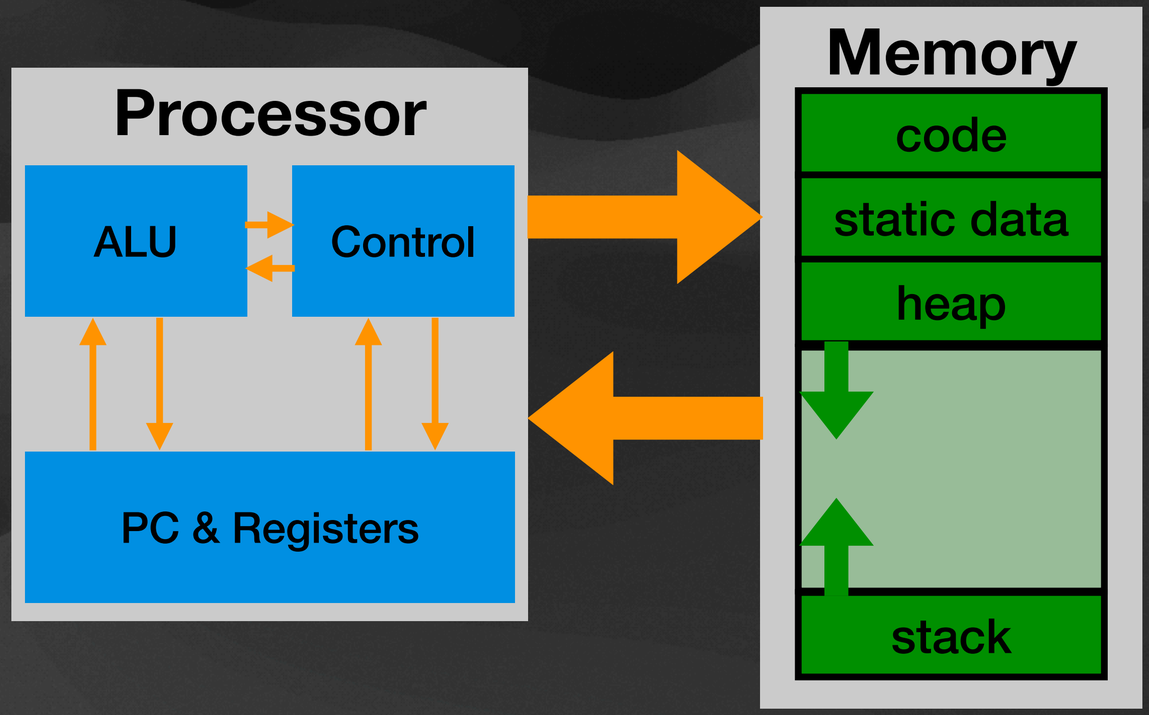
Introduction to cache

- What is cache?
- Why cache?

Suting Chen

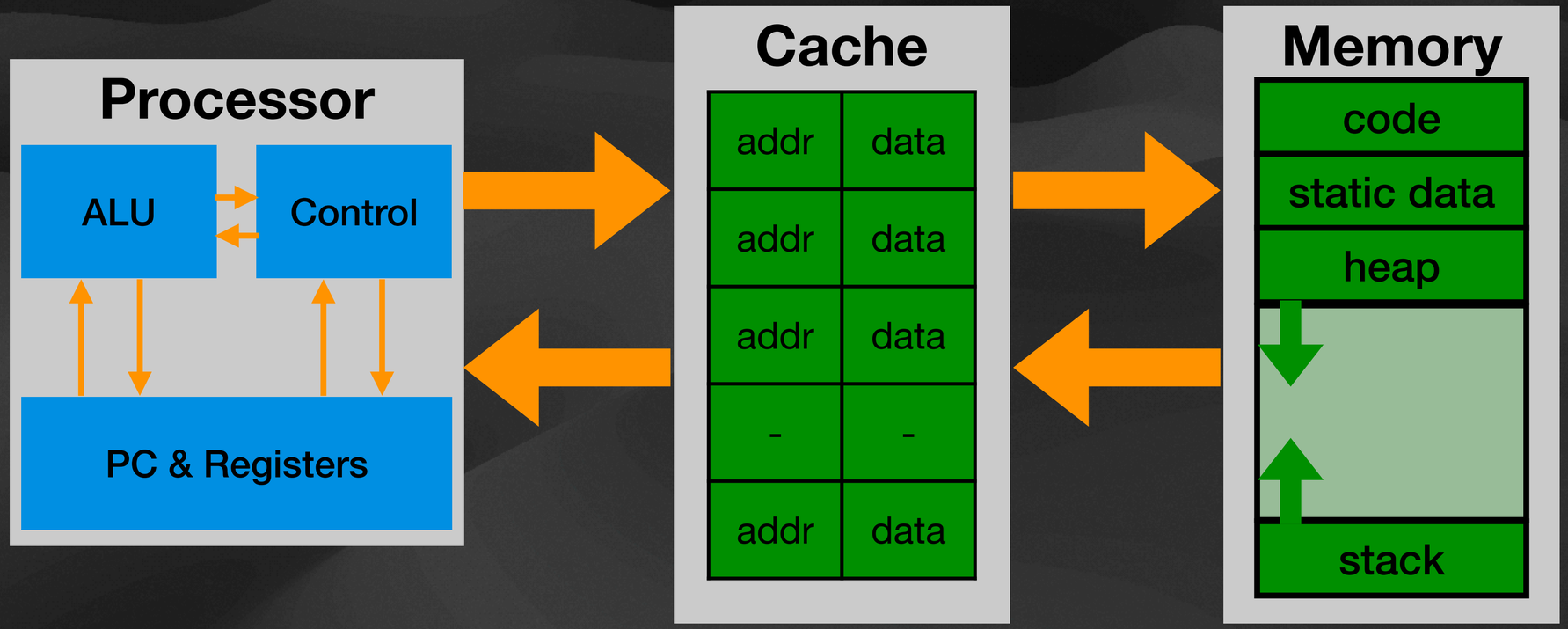
What is cache?

- Pronounce as “$”
- Origin model



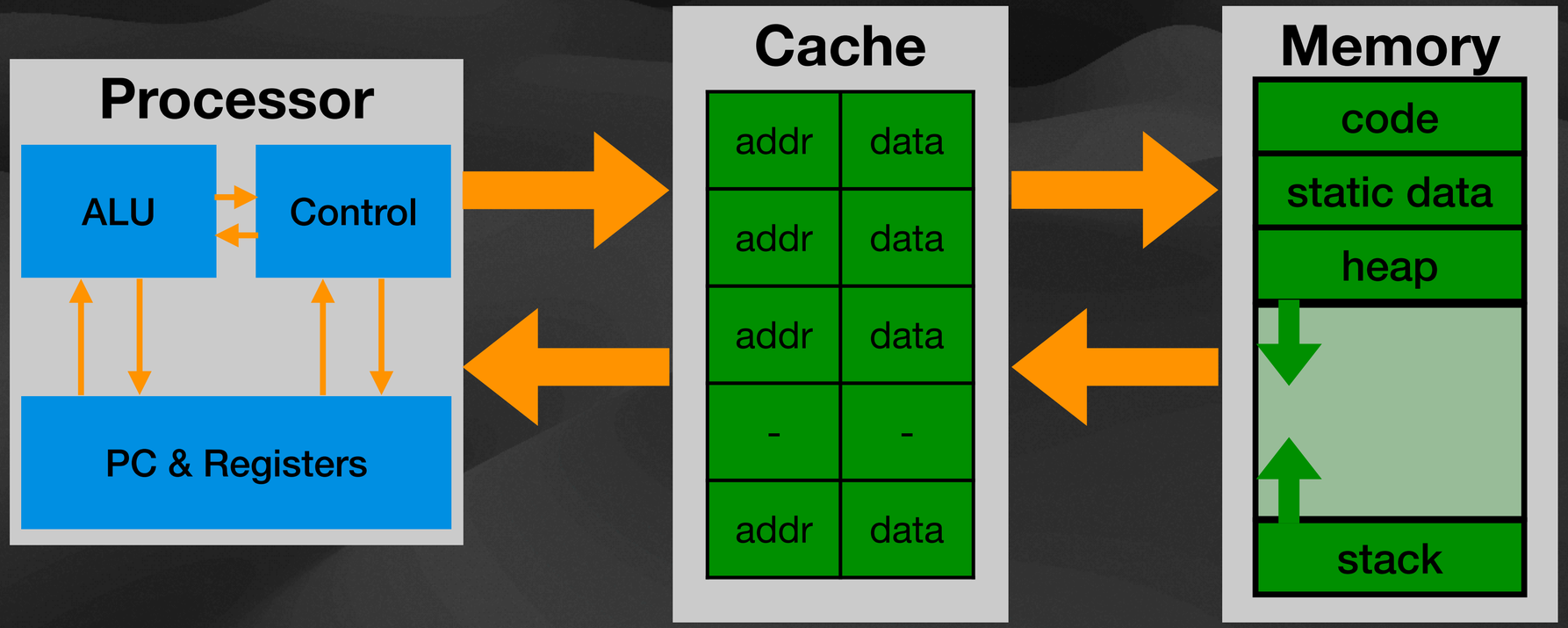
What is cache?

- New model with $ added



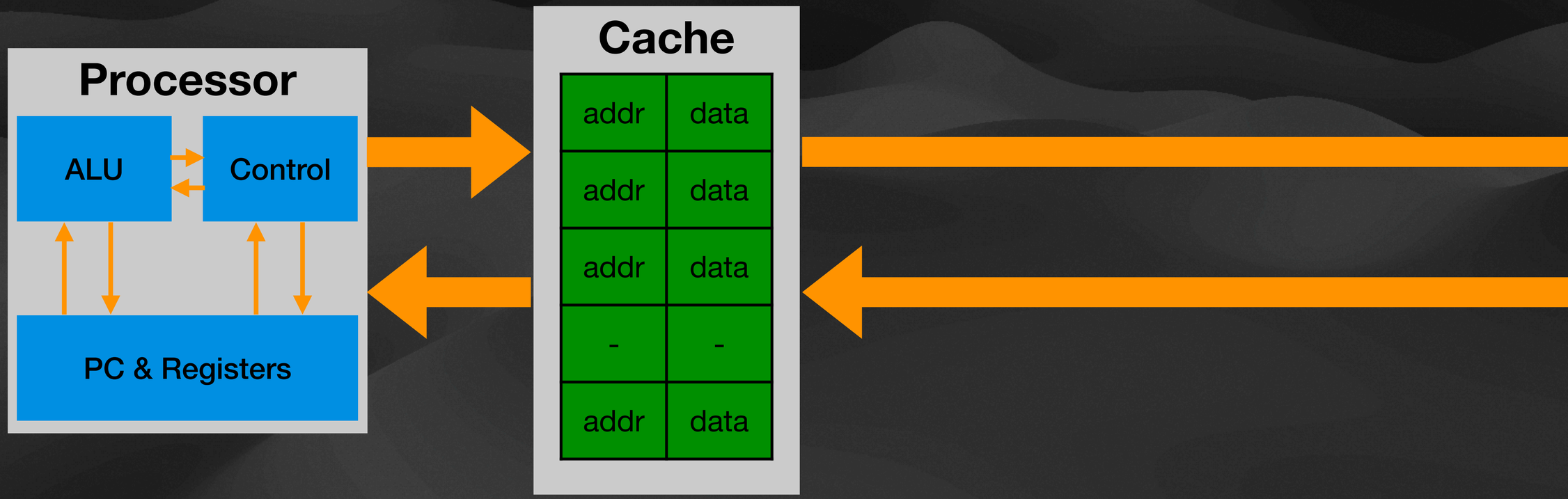
Why cache?

Memory is TOO slow!



Why cache?

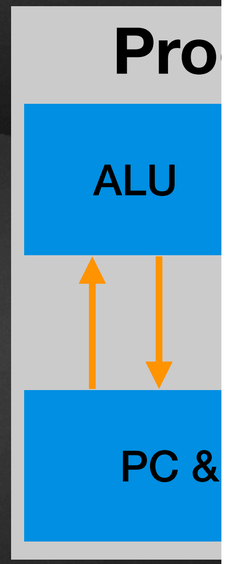
Memory is TOO slow!

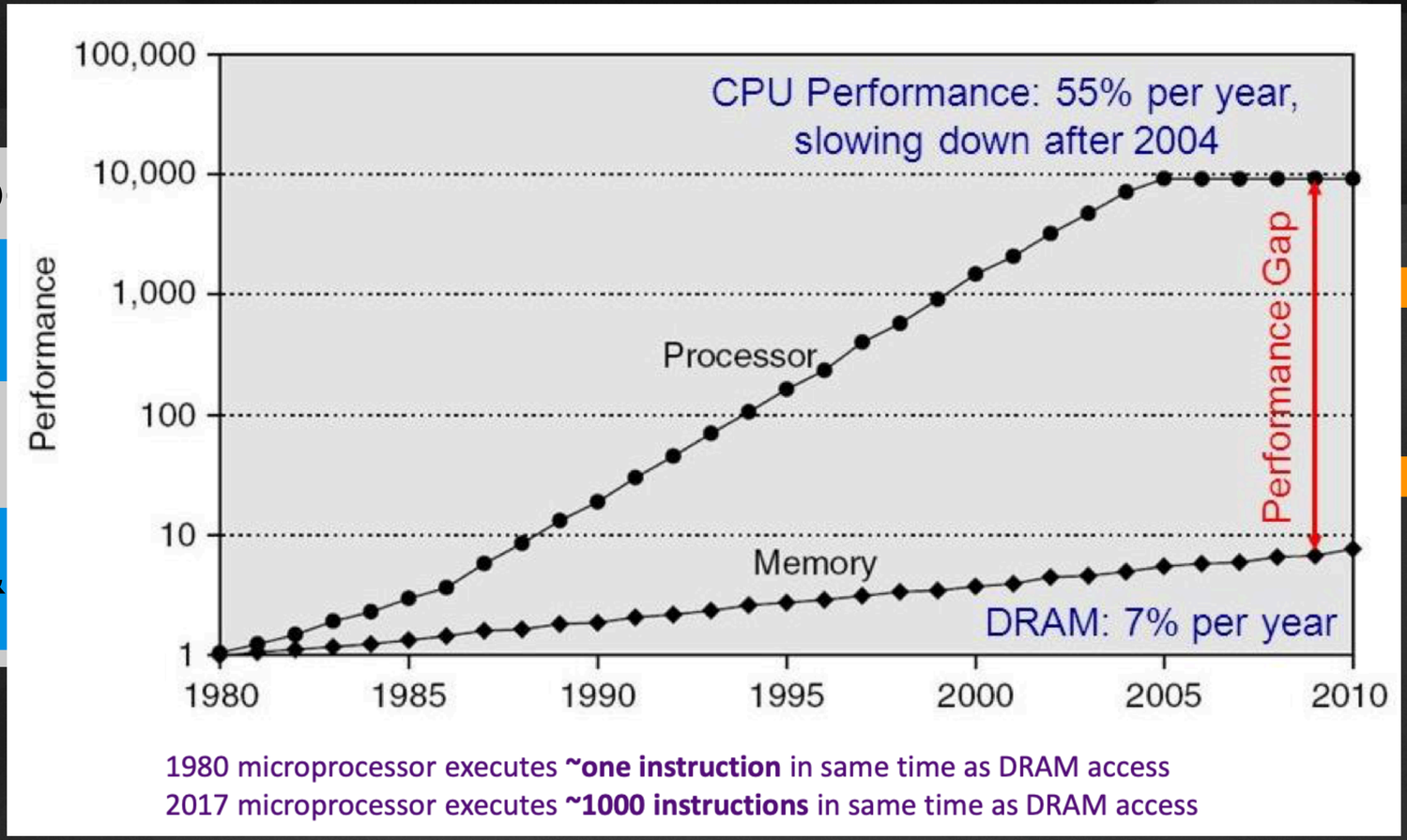


Suting Chen

Why cache?

Memory is TOO slow!

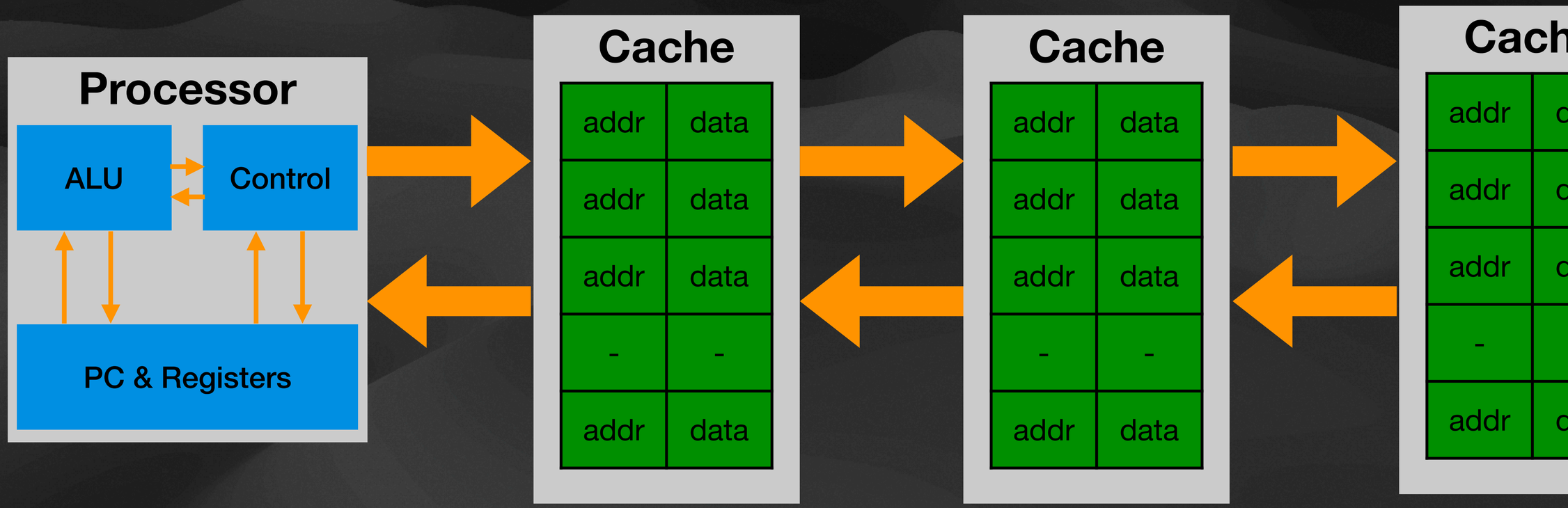




Suting Chen

Why cache?

Sometime one piece don’t meet our needs



Suting Chen

Exercise

Introduction

Suting Chen

How is cache indexed?

Hash table!

- Tag | Index | Offset

How is cache indexed?

Tag | Index | Offset

- In the slides, we consider 16-bit addresses.
  - Tag = 8
  - Index = 3
  - Offset = 5

```
0b1111101101011010
```

```
0b 11111011 010 11010
```

How is cache indexed?

Tag | Index | Offset

```
0b 11111011 010 11010
```

- Introduce “block”
  - 访问一次内存得到的“最少”数据
  - 同时也是cache中实际储存的“data”

- Block_size = 2^{Offset_bit} bytes
- Here, $2^5 = 32$ bytes

How is cache indexed?

Tag | Index | Offset

```
0b 11111011 010 11010
```

- How to get the byte we are asking for in a block?

```
uint8_t block[32];

return block[offset];
```

  - That's why “offset”!

How is cache indexed?

Tag | **Index** | **Offset**

```
0b 11111011 010 11010
```

- The whole cache is partitioned into 2^{index} parts
    1. Find the “allowed” cache block(s)
    2. Compare with “Tag”
    3. Read

Index	Tag	Data
000	01101101	0x91e989af
001	01000010	0xf4958104
010	11111011	0x82a0f265
011	01100101	0xfcf4ceef
100	00010101	0x88b5af65
101	10110101	0x190349ff
110	10011010	0x4358f247
111	00000011	0xf1e6433b

How is cache indexed?

Tag | **Index** | Offset

```
0b 11111011 010 11010
```

- The whole cache is partitioned into 2^{index} parts
  1. Find the “allowed” cache block(s)
  2. Compare with “Tag”
  3. Read

Index	Tag	Data
000	01101101	0x91e989af
001	01000010	0xf4958104
010	11111011	0x82a0f265
011	01100101	0xfcf4ceef
100	00010101	0x88b5af65
101	10110101	0x190349ff
110	10011010	0x4358f247
111	00000011	0xf1e6433b

How is cache indexed?

Tag | **Index** | **Offset**

```
0b 11111011 010 11010
```

- The whole cache is partitioned into 2^{index} parts
    1. Find the “allowed” cache block(s)
    2. Compare with “Tag”
    3. Read

Index	Tag	Data
000	01101101	0x91e989af
001	01000010	0xf4958104
010	11111011	0x82a0f265
011	01100101	0xfcf4ceef
100	00010101	0x88b5af65
101	10110101	0x190349ff
110	10011010	0x4358f247
111	00000011	0xf1e6433b

How is cache indexed?

Tag | **Index** | **Offset**

```
0b 11111011 010 11010
```

- The whole cache is partitioned into 2^{index} parts
    1. Find the “allowed” cache block(s)
    2. Compare with “Tag”
    3. Read

Index	Tag	Data
000	01101101	0x91e989af
001	01000010	0xf4958104
010	11111011	0x82a0f265
011	01100101	0xfcf4ceef
100	00010101	0x88b5af65
101	10110101	0x190349ff
110	10011010	0x4358f247
111	00000011	0xf1e6433b

How is cache indexed?

Tag | **Index** | **Offset**

```
0b 11111011 010 11010
```

- The whole cache is partitioned into 2^{index} parts
    1. Find the “allowed” cache block(s)
    2. Compare with “Tag”
    3. Read
    4. Consider offset

Index	Tag	Data
000	01101101	0x91e989af
001	01000010	0xf4958104
010	11111011	0x82a0f265
011	01100101	0xfcf4ceef
100	00010101	0x88b5af65
101	10110101	0x190349ff
110	10011010	0x4358f247
111	00000011	0xf1e6433b

By the way …

- How many comparators do we need?

Index	Tag	Data
000	01101101	0x91e989af
001	01000010	0xf4958104
010	11111011	0x82a0f265
011	01100101	0xfcf4ceef
100	00010101	0x88b5af65
101	10110101	0x190349ff
110	10011010	0x4358f247
111	00000011	0xf1e6433b

How is cache indexed?

More Tags

1. Find the “allowed” cache block(s)
2. Compare with “Tag”s
3. Read

4. Consider offset

```
0b 11111011 010 11010
```

Index	Tag	Data	Tag	Data
000	01101101	0x91e989af	10100100	0x05977891
001	01000010	0xf4958104	01010000	0x08ecc2c9
010	11111011	0x82a0f265	10011010	0xa39643ef
011	01100101	0xfcf4ceef	11100100	0xe6a129e3
100	00010101	0x88b5af65	10110000	0x4266d1c9
101	10110101	0x190349ff	10001111	0xdd1daeef
110	10011010	0x4358f247	00100111	0x52a16ae8
111	00000011	0xf1e6433b	10011000	0xaac20e31

Associativity

- 2-way Set Associative

Index	Tag	Data	Tag	Data
000	01101101	0x91e989af	10100100	0x05977891
001	01000010	0xf4958104	01010000	0x08ecc2c9
010	11111011	0x82a0f265	10011010	0xa39643ef
011	01100101	0xfcf4ceef	11100100	0xe6a129e3
100	00010101	0x88b5af65	10110000	0x4266d1c9
101	10110101	0x190349ff	10001111	0xdd1daeef
110	10011010	0x4358f247	00100111	0x52a16ae8
111	00000011	0xf1e6433b	10011000	0xaac20e31

Associativity

Index	Tag	Data	Tag	Data	Tag	Data	Tag	Data
000	01101101	0x91e989af	10100100	0x05977891	01010101	0x88b5af65	01000010	0xdd1daeef
001	01000010	0xf4958104	01010000	0x08ecc2c9	01110101	0x190349ff	11111011	0x52a16ae8
010	11111011	0x82a0f265	10011010	0xa39643ef	01111111	0x4358f247	01100101	0xaac20e31
011	01100101	0xfcf4ceef	11100100	0xe6a129e3	01000001	0xf1e6433b	00010101	0xfcf4ceef
100	00010101	0x88b5af65	10110000	0x4266d1c9	00010110	0x88b5af65	10110101	0xf4958104
101	10110101	0x190349ff	10001111	0xdd1daeef	00101001	0x190349ff	10011010	0x82a0f265
110	10011010	0x4358f247	00100111	0x52a16ae8	10001111	0x4358f247	00000011	0xfcf4ceef
111	00000011	0xf1e6433b	10011000	0xaac20e31	11000000	0xf1e6433b	00111000	0x88b5af65

Suting Chen

Associativity

Direct mapped & Fully associative

Index	Tag	Data
000	01101101	0x91e989af
001	01000010	0xf4958104
010	11111011	0x82a0f265
011	01100101	0xfcf4ceef
100	00010101	0x88b5af65
101	10110101	0x190349ff
110	10011010	0x4358f247
111	00000011	0xf1e6433b

Tag	Data
01101101	0x91e989af
01000010	0xf4958104
11111011	0x82a0f265
01100101	0xfcf4ceef
00010101	0x88b5af65
10110101	0x190349ff
10011010	0x4358f247
00000011	0xf1e6433b

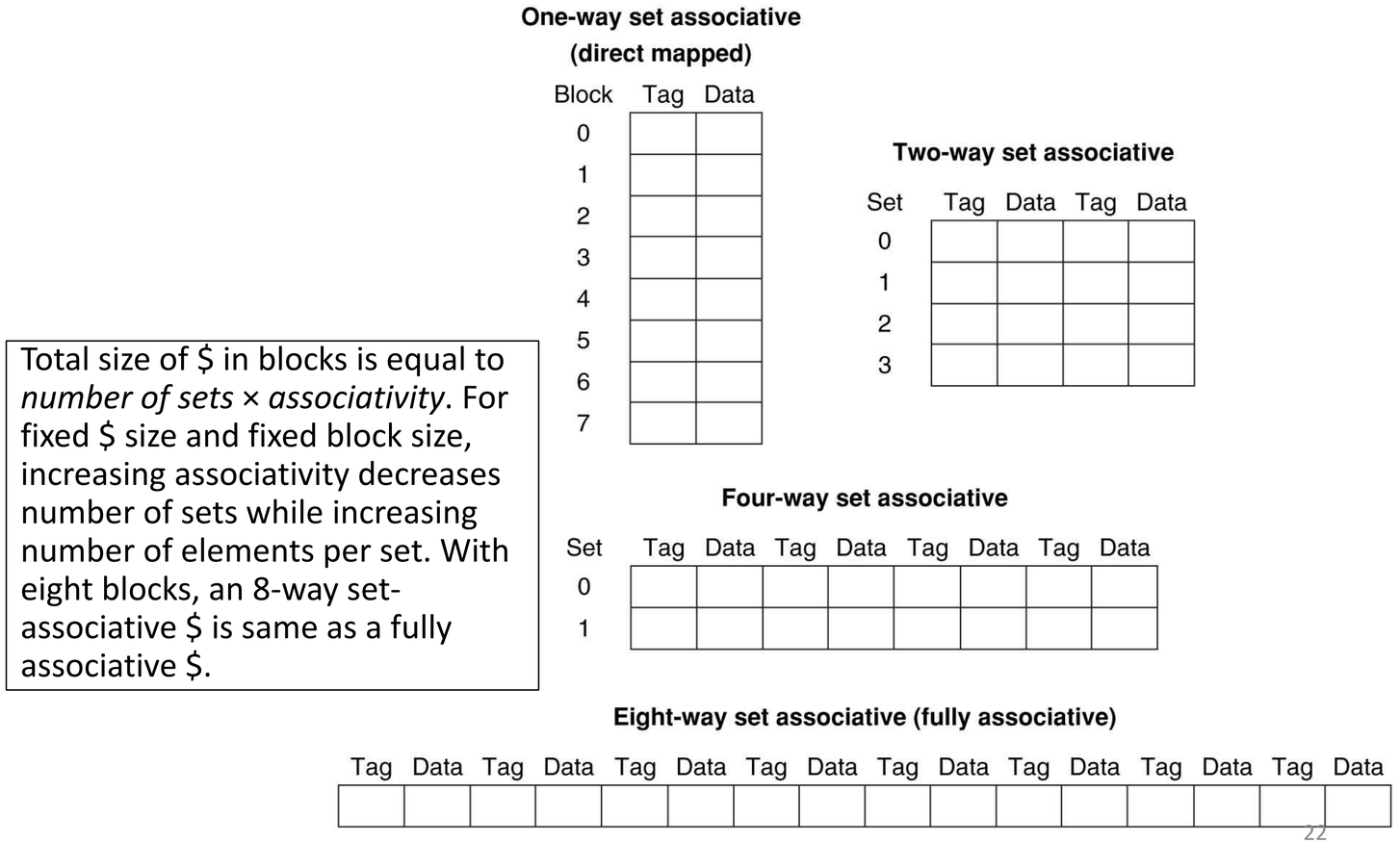
Associativity

Cache Names for Each Organization

- "Fully Associative": Line can go anywhere
  - First design in lecture
  - Note: No Index field, but 1 comparator/ line
- "Direct Mapped": Line goes one place
  - Note: Only 1 comparator
  - Number of sets = number blocks
- "N-way Set Associative": N places for a line
  - Number of sets = number of lines/ N
  - N comparators
  - ***Fully Associative: N = number of lines***
  - ***Direct Mapped: N = 1***

Associativity

Different Organizations of an Eight-Block Cache



Total size of $ in blocks is equal to *number of sets* × *associativity*. For fixed $ size and fixed block size, increasing associativity decreases number of sets while increasing number of elements per set. With eight blocks, an 8-way set-associative $ is same as a fully associative $.

Exercise

Indexing cache

Suting Chen

Replacement Policy

What is replacement

1. Find the “allowed” cache block(s)
2. Compare with “Tag”
3. Read

```
0b 10111011 010 11010
```

Index	Tag	Data
000	01101101	0x91e989af
001	01000010	0xf4958104
010	11111011	0x82a0f265
011	01100101	0xfcf4ceef
100	00010101	0x88b5af65
101	10110101	0x190349ff
110	10011010	0x4358f247
111	00000011	0xf1e6433b

Suting Chen

Replacement Policy

What is replacement

1. Find the “allowed” cache block(s)
2. Compare with “Tag”
3. Read

```
0b 10111011 010 11010
```

Index	Tag	Data
000	01101101	0x91e989af
001	01000010	0xf4958104
010	11111011	0x82a0f265
011	01100101	0xfcf4ceef
100	00010101	0x88b5af65
101	10110101	0x190349ff
110	10011010	0x4358f247
111	00000011	0xf1e6433b

Suting Chen

Replacement Policy

What is replacement

1. Find the “allowed” cache block(s)
2. Compare with “Tag”
3. Read

4. Replace

```
0b 10111011 010 11010
```

Index	Tag	Data
000	01101101	0x91e989af
001	01000010	0xf4958104
010	10111011	0x8365b3c1
011	01100101	0xfcf4ceef
100	00010101	0x88b5af65
101	10110101	0x190349ff
110	10011010	0x4358f247
111	00000011	0xf1e6433b

Suting Chen

Replacement Policy

What if we are not direct-mapped?

```
0b 10111011 010 11010
```

1. Find the “allowed” cache block(s)
2. Compare with “Tag”
3. Read

4. Replace Which?

Index	Tag	Data	Tag	Data
000	01101101	0x91e989af	10100100	0x05977891
001	01000010	0xf4958104	01010000	0x08ecc2c9
010	11111011	0x82a0f265	10011010	0xa39643ef
011	01100101	0xfcf4ceef	11100100	0xe6a129e3
100	00010101	0x88b5af65	10110000	0x4266d1c9
101	10110101	0x190349ff	10001111	0xdd1daeef
110	10011010	0x4358f247	00100111	0x52a16ae8
111	00000011	0xf1e6433b	10011000	0xaac20e31

Replacement Policy

Popular policies

- Random
- LRU
- FIFO

Suting Chen

Replacement Policy — Random

```
0b 10111011 010 11010
```

- Simply randomly pick one to kick

Index	Tag	Data	Tag	Data
000	01101101	0x91e989af	10100100	0x05977891
001	01000010	0xf4958104	01010000	0x08ecc2c9
010	11111011	0x82a0f265	10011010	0xa39643ef
011	01100101	0xfcf4ceef	11100100	0xe6a129e3
100	00010101	0x88b5af65	10110000	0x4266d1c9
101	10110101	0x190349ff	10001111	0xdd1daeef
110	10011010	0x4358f247	00100111	0x52a16ae8
111	00000011	0xf1e6433b	10011000	0xaac20e31

Replacement Policy — Random

- Simply randomly pick one to kick
- Only if there are no empty space!

Replacement Policy — LRU

Least Recently Used

- Time records the last time used

`0b 10111011 010 11010`

Index	Tag	Time	Data	Tag	Time	Data
000	01101101	34	0x91e989af	10100100	45	0x05977891
001	01000010	28	0xf4958104	01010000	2	0x08ecc2c9
010	11111011	32	0x82a0f265	10011010	41	0xa39643ef
011	01100101	11	0xfcf4ceef	11100100	22	0xe6a129e3
100	00010101	49	0x88b5af65	10110000	33	0x4266d1c9
101	10110101	49	0x190349ff	10001111	34	0xdd1daeef
110	10011010	49	0x4358f247	00100111	36	0x52a16ae8
111	00000011	32	0xf1e6433b	10011000	18	0xaac20e31

Replacement Policy — LRU

Least Recently Used

- Time records the last time used

- Compare and evict

`0b 10111011 010 11010`

Index	Tag	Time	Data	Tag	Time	Data
000	01101101	34	0x91e989af	10100100	45	0x05977891
001	01000010	28	0xf4958104	01010000	2	0x08ecc2c9
010	11111011	32	0x82a0f265	10011010	41	0xa39643ef
011	01100101	11	0xfcf4ceef	11100100	22	0xe6a129e3
100	00010101	49	0x88b5af65	10110000	33	0x4266d1c9
101	10110101	49	0x190349ff	10001111	34	0xdd1daeef
110	10011010	49	0x4358f247	00100111	36	0x52a16ae8
111	00000011	32	0xf1e6433b	10011000	18	0xaac20e31

Replacement Policy — LRU

Least Recently Used

- Time records the last time used

- Compare and evict
- Remember to update Time!

0b 10111011 010 11010

Index	Tag	Time	Data	Tag	Time	Data
000	01101101	34	0x91e989af	10100100	45	0x05977891
001	01000010	28	0xf4958104	01010000	2	0x08ecc2c9
010	10111011	50	0x3250af22	10011010	41	0xa39643ef
011	01100101	11	0xfcf4ceef	11100100	22	0xe6a129e3
100	00010101	49	0x88b5af65	10110000	33	0x4266d1c9
101	10110101	49	0x190349ff	10001111	34	0xdd1daeef
110	10011010	49	0x4358f247	00100111	36	0x52a16ae8
111	00000011	32	0xf1e6433b	10011000	18	0xaac20e31

Suting Chen

Exercise

Replacement policy

Suting Chen

Replacement policy

LRU

```
0: 0b 1100 01 10
1: 0b 0110 01 01
2: 0b 1110 11 00
3: 0b 1100 01 10
4: 0b 0110 10 10
5: 0b 1111 01 11
```

表格 1

Index	Visit	0	1	2	3	4	5
00	slot0						
	slot1						
01	slot0	1100					
	slot1						
10	slot0						
	slot1						
11	slot0						
	slot1						

Suting Chen

Replacement policy

LRU

```
0: 0b 1100 01 10
1: 0b 0110 01 01
2: 0b 1110 11 00
3: 0b 1100 01 10
4: 0b 0110 10 10
5: 0b 1111 01 11
```

表格 1

Index	Visit	0	1	2	3	4	5
00	slot0						
	slot1						
01	slot0	1100					
	slot1		0110				
10	slot0						
	slot1						
11	slot0						
	slot1						

Suting Chen

Replacement policy

LRU

```
0: 0b 1100 01 10
1: 0b 0110 01 01
2: 0b 1110 11 00
3: 0b 1100 01 10
4: 0b 0110 10 10
5: 0b 1111 01 11
```

表格 1

Index	Visit	0	1	2	3	4	5
00	slot0						
	slot1						
01	slot0	1100					
	slot1		0110				
10	slot0						
	slot1						
11	slot0			1110			
	slot1						

Suting Chen

Replacement policy

LRU

```
0: 0b 1100 01 10
1: 0b 0110 01 01
2: 0b 1110 11 00
3: 0b 1100 01 10
4: 0b 0110 10 10
5: 0b 1111 01 11
```

表格 1

Index	Visit	0	1	2	3	4	5
00	slot0						
	slot1						
01	slot0	1100			1100		
	slot1		0110				
10	slot0						
	slot1						
11	slot0			1110			
	slot1						

Suting Chen

Replacement policy

LRU

```
0: 0b 1100 01 10
1: 0b 0110 01 01
2: 0b 1110 11 00
3: 0b 1100 01 10
4: 0b 0110 10 10
5: 0b 1111 01 11
```

表格 1

Index	Visit	0	1	2	3	4	5
00	slot0						
	slot1						
01	slot0	1100			1100		
	slot1		0110				
10	slot0					0110	
	slot1						
11	slot0			1110			
	slot1						

Suting Chen

Replacement policy

LRU

```
0: 0b 1100 01 10
1: 0b 0110 01 01
2: 0b 1110 11 00
3: 0b 1100 01 10
4: 0b 0110 10 10
5: 0b 1111 01 11
```

表格 1

Index	Visit	0	1	2	3	4	5
00	slot0						
	slot1						
01	slot0	1100			1100		
	slot1		0110				1111
10	slot0					0110	
	slot1						
11	slot0			1110			
	slot1						

Suting Chen

Write Policy

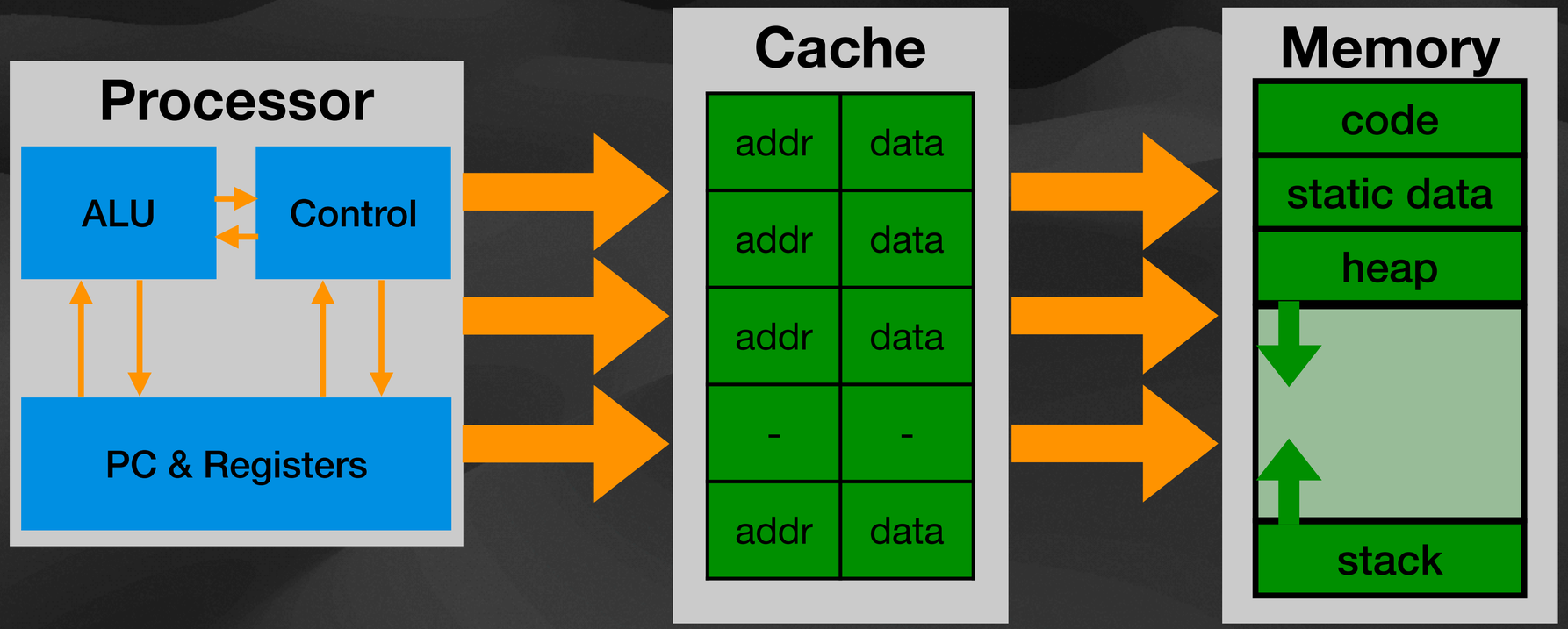
- Write Through
- Write Back

- Write Allocate
- No Write Allocate

Suting Chen

Write Through

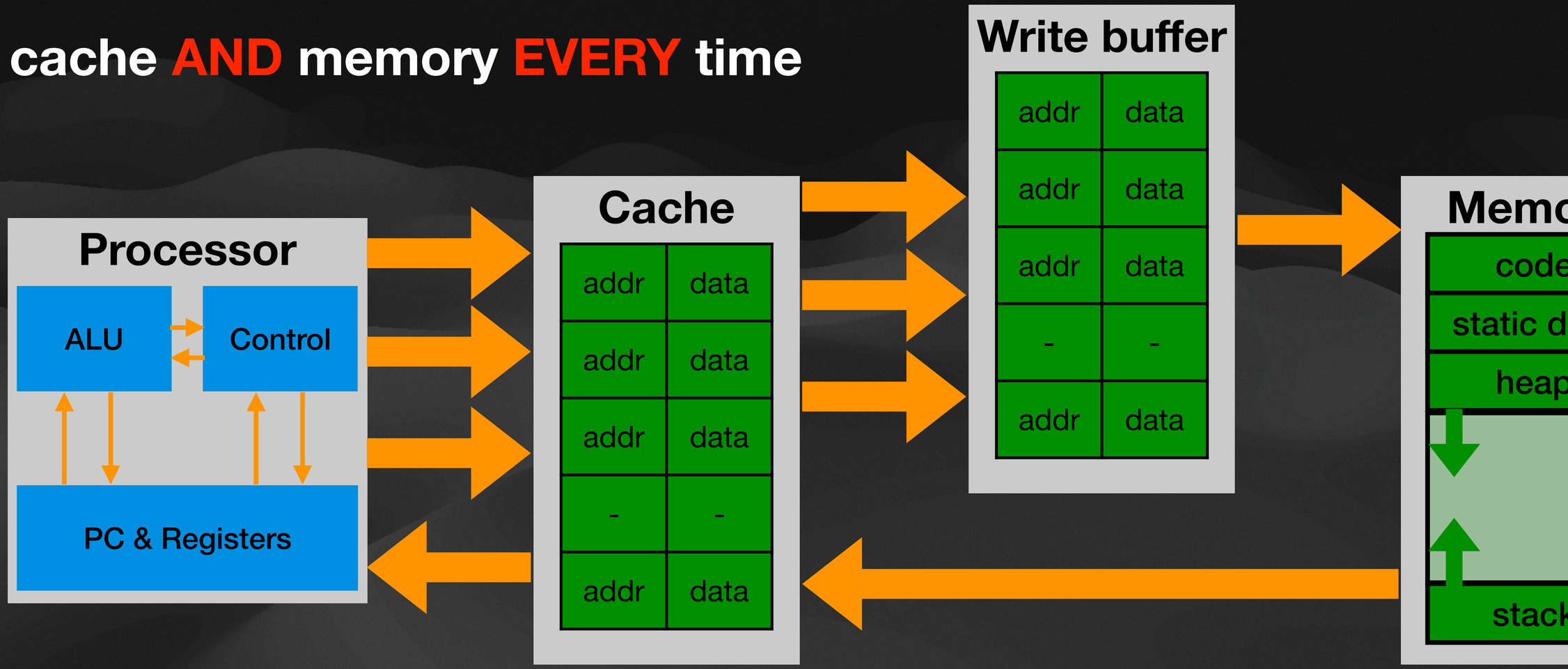
Write cache AND memory EVERY time



Suting Chen

Write Through

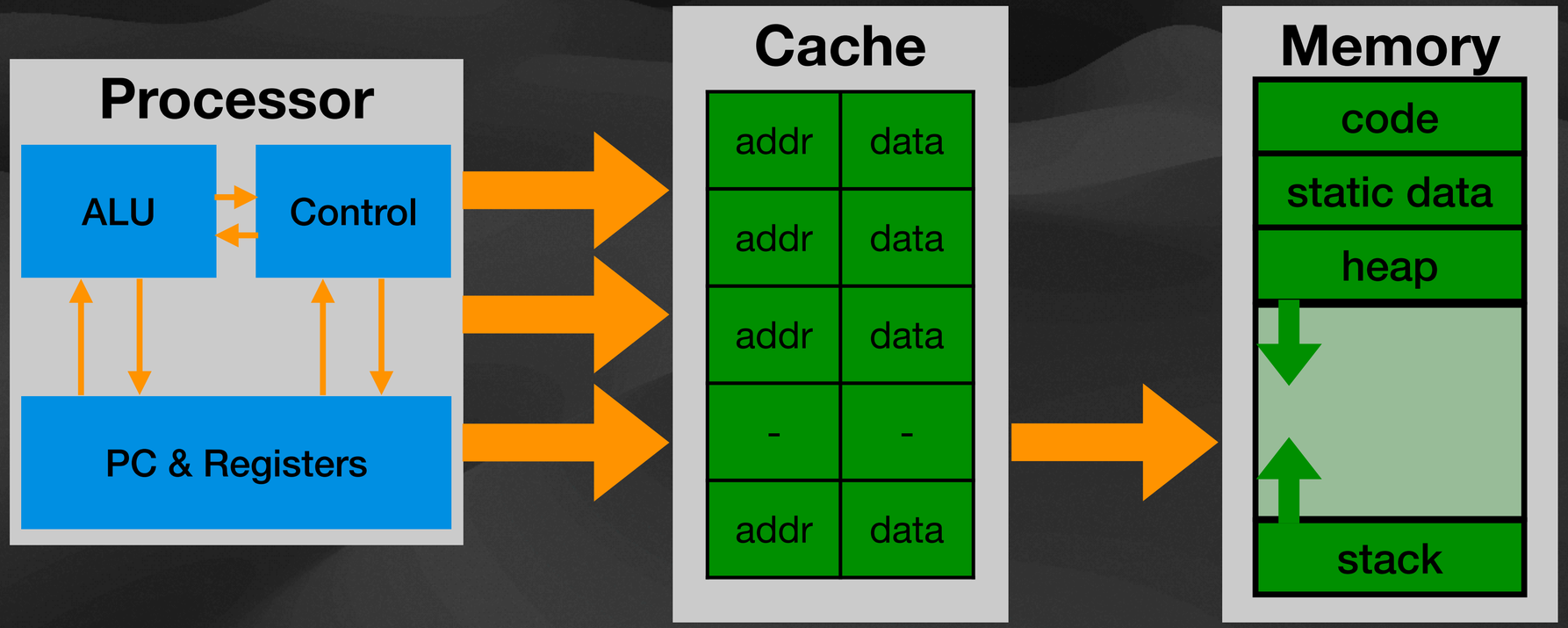
Write cache AND memory EVERY time



Suting Chen

Write Back

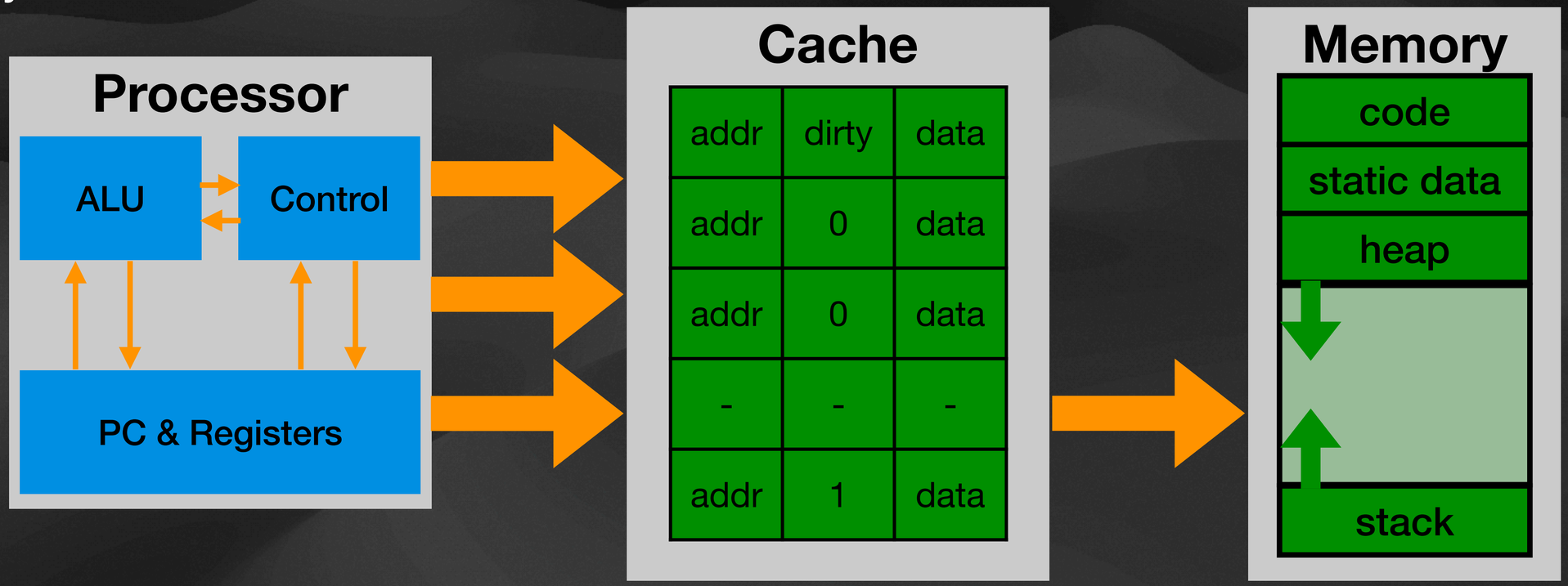
Write memory ONLY when evicted



Write Back

Write memory ONLY when evicted and modified

- “Dirty bit” set to 1 if modified



Write Allocate & No Write Allocate

- Write Allocate
    - If writing to memory not in the cache fetch it first
- No Write Allocate
    - If writing to memory not in the cache, no fetch

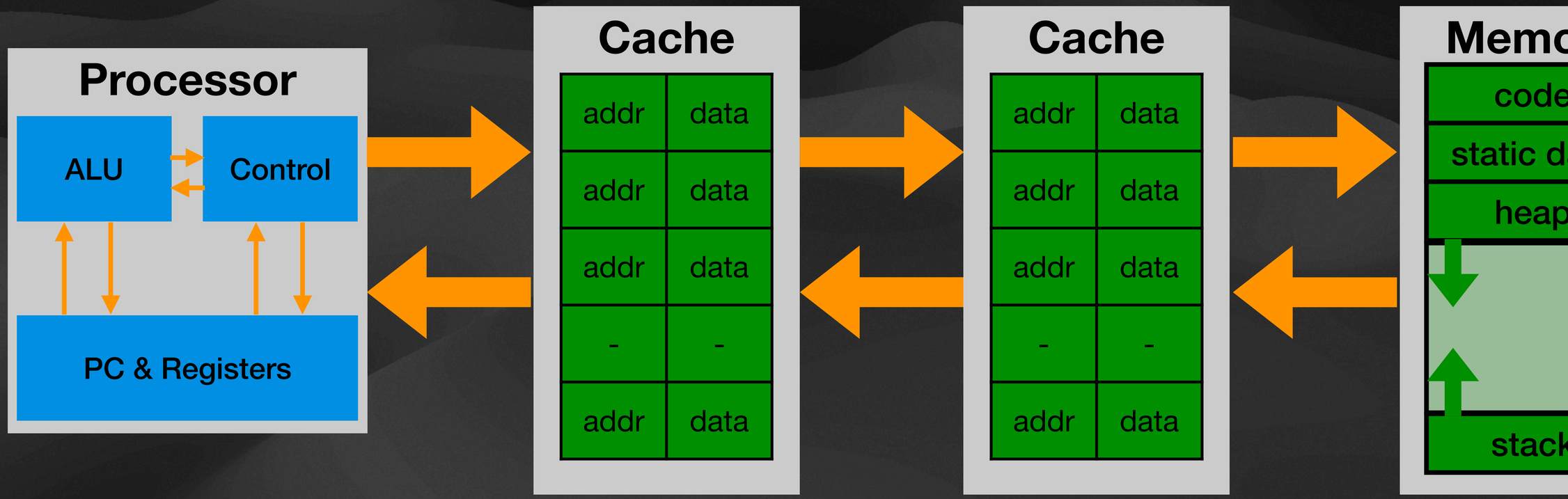
Suting Chen

Measure performance

- Hit rate / Miss rate
- Hit time / Miss penalty
- Average Memory Access Time (AMAT)

- 3C

Hit rate / Miss rate



Suting Chen

Hit rate / Miss rate

- Local hit rate / Local miss rate
- L2 cache miss rate = $\frac{\#\ of\ L2\ miss}{\#\ of\ L1\ miss}$
- Global miss rate =

$$\frac{\#\ L2\ miss}{\#\ access} = \frac{\#\ L2\ miss}{\#\ L1\ miss} \times \frac{\#\ L1\ miss}{\#\ access} = \text{L2 local} \times \text{L1 local}$$

Hit time / Miss penalty / AMAT

- “Miss time” = Hit time + Miss penalty

- AMAT = Hit time + Miss rate × Miss penalty
- AMAT = L1 Hit time +

  L1 local miss rate × (L2 Hit time + L2 local miss rate × Miss penalty)

Compulsory / Capacity / Conflict miss

- Compulsory: Cold start
- Capacity: Cache too small
- Conflict: associativity
- Coherence: shared data

Index	Tag	Data	Tag	Data
000	01101101	0x91e989af	10100100	0x05977891
001	01000010	0xf4958104	01010000	0x08ecc2c9
010	11111011	0x82a0f265	10011010	0xa39643ef
011	01100101	0xfcf4ceef	11100100	0xe6a129e3
100	00010101	0x88b5af65	10110000	0x4266d1c9
[illegible]	[illegible]	[illegible]	[illegible]0001111	0xdd1daeef
110	10011010	0x4358f247	00100111	0x52a16ae8
111	00000011	0xf1e6433b	10011000	0xaac20e31

Don’t waste too much time considering this!

Exercise

Something else

Suting Chen

Cachegrind

```
> valgrind --tool=cachegrind ./main
```