

# Cache controllers; metrics

My CPU when the L1 cache  
misses



This little maneuver is gonna cost us 3 nanoseconds

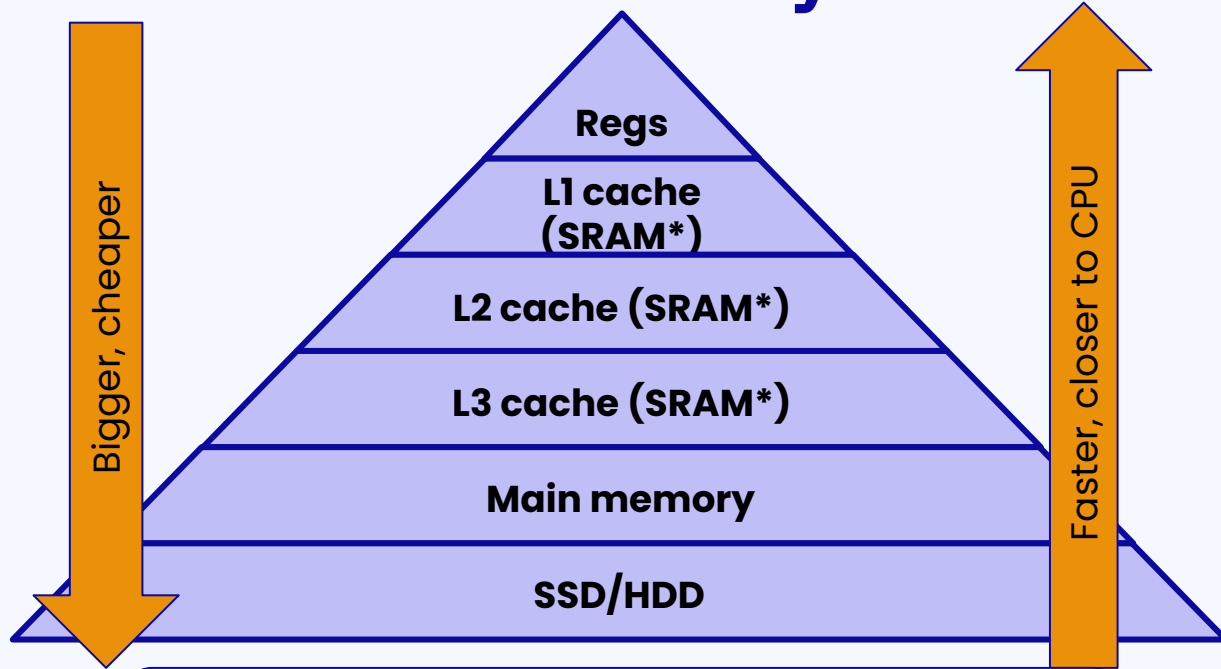


Async (📍 Paris Arc 🇫🇷)

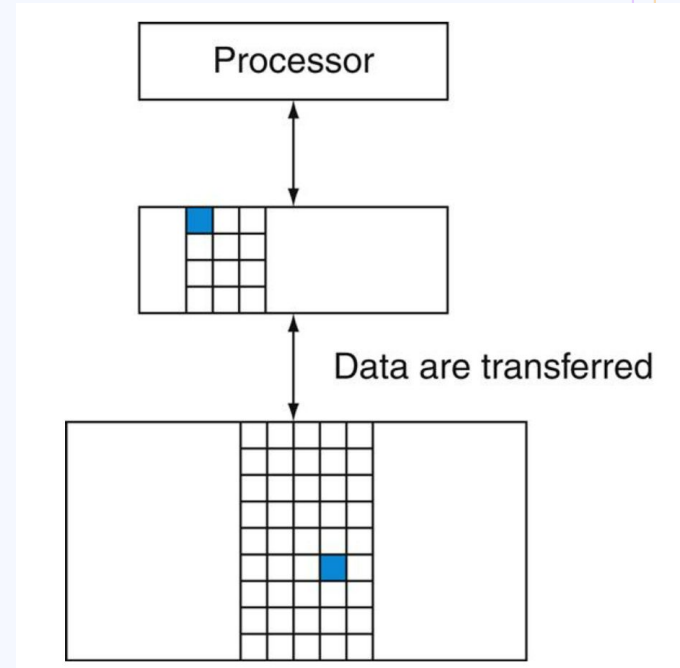
@0xAsync

No mom it's not a "messy pile of clothes on my chair" it's an L1 cache for fast random access to my frequently used clothes in  $O(1)$  time. It needs to be big to avoid expensive cache misses (looking in my closet). I NEED to be minimizing latency, this is important to me. Please.

# Mem. hierarchy review



Each level stores (**caches**) a subset of the one below it, for faster access of specific data



P&H 5.2



What questions do we need to ask when designing a cache?

- How do we decide what goes where in a cache?
  - What control information do we need to keep around in order to implement our cache?
    - How do we decide what data to evict?
- What is our scheme of maintaining consistent data?
  - How do we build an efficient memory hierarchy?  
(What is “efficient”)

# The code

Assume address 0x20000004 is in register a0

```
lw t0, 0(a0)
```

```
lw t1, 4(a0)
```

```
lhu t2, 2(a0)
```

```
lw t0, 1024(a0)
```

```
addi t1, t1, 1
```

```
sw t1, 4(a0)
```

```
sw t1, 8(a0)
```

# Terminology

**Block:** minimum unit of information that can be present/not present in a cache

**Valid bit:** indicates whether data has been pulled in to that block of the cache

**Tag:** the upper bits of an address, used to uniquely identify which data is in the cache

# How do we decide what goes where?

For now: assume 1 word (4 byte) block size

Assume one cache that holds 256 ( $2^8$ ) blocks (1KB)

Assume address 0x20000004 is in register a0

```
lw t0, 0(a0)
```

```
lw t1, 4(a0)
```

Block address – the address of the block of memory we care about  
Why do we bring things in to cache before loading them into a register?  
Where in the cache do these blocks go?

# What control information do we need?

Assume address 0x20000004 is in register a0

```
lw t0, 0(a0)
```

```
lw t1, 4(a0)
```

```
lhu t2, 2(a0)
```

How do we tell that the block with the data we need already exists in the cache?

# How do we decide what data to evict?

Assume address 0x20000004 is in register a0

```
lw t0, 0(a0)
```

```
lw t1, 4(a0)
```

```
lhu t2, 2(a0)
```

```
lw t0, 1024(a0)
```

Address maps to a cache block already in use:  
how do we know that it's already in use?



## What is our scheme of maintaining consistent data?

Assume address 0x20000004 is in register a0

```
lw t0, 0(a0)
```

```
lw t1, 4(a0)
```

```
lhu t2, 2(a0)
```

```
lw t0, 1024(a0)
```

```
addi t1, t1, 1
```

```
sw t1, 4(a0)
```

```
sw t1, 8(a0)
```

What happens when data that exists in the cache is modified?

What happens when data that does not exist in the cache is modified?

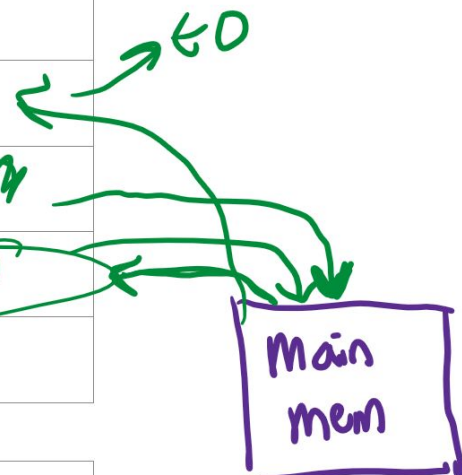
0x20000004 = 0b0010 0000 ..... 0000 0100

35 bits: block addr  
 22 bits: tag  
 8 bits: cache block  
 byte offset

Assume address 0x20000004 is in register a0

```
lw t0, 0(a0)
lw t1, 4(a0)
lhu t2, 2(a0)
lw t0, 1024(a0)
addi t1, t1, 1
sw t1, 4(a0)
sw t1, 8(a0)
```

| \$ block | V | tag (22 bits) | data (32 bits) |
|----------|---|---------------|----------------|
| 0        | 0 |               |                |
| 1        | 1 | 0010...01     |                |
| → 2      | 1 | 0010...0      | MEM            |
| 3        | 1 | 0010...0      | MEM            |
| 4        | 0 |               |                |
| ⋮        |   |               |                |
| 254      | 0 |               |                |
| 255      | 0 |               |                |



# Write through vs write back

Write through: every time data is changed in cache, change is done to lower level in hierarchy

Pro: Data are kept consistent (and don't have to write on evict)

Con: Seems slower

Write back: changes to lower level in hierarchy are only done when data is evicted from cache

Pro: Fewer writes to main memory

Con: Consistency/control issues