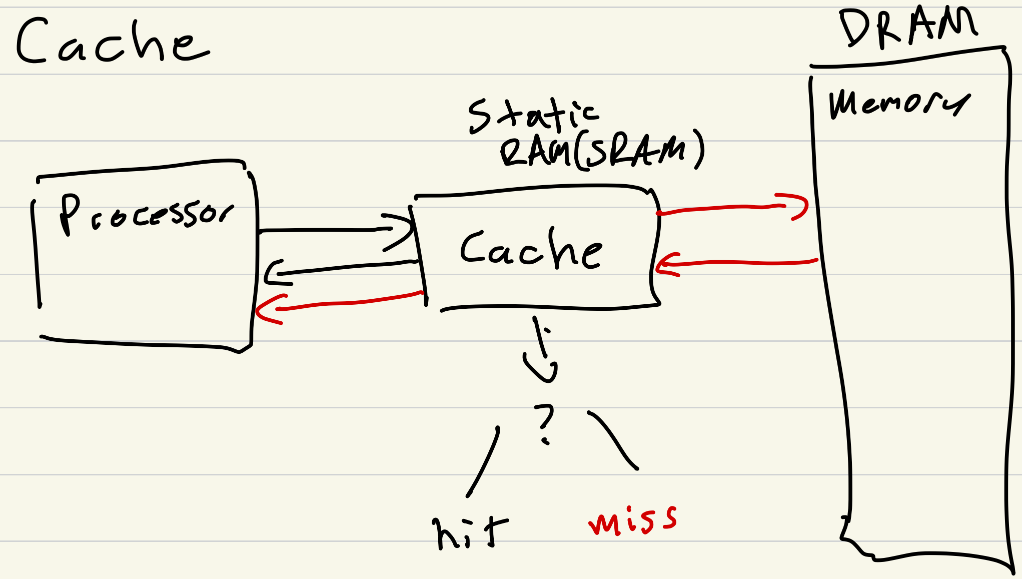
CS631-01 Cache Simulation

Analysis

~~AUIPC~~

Cache



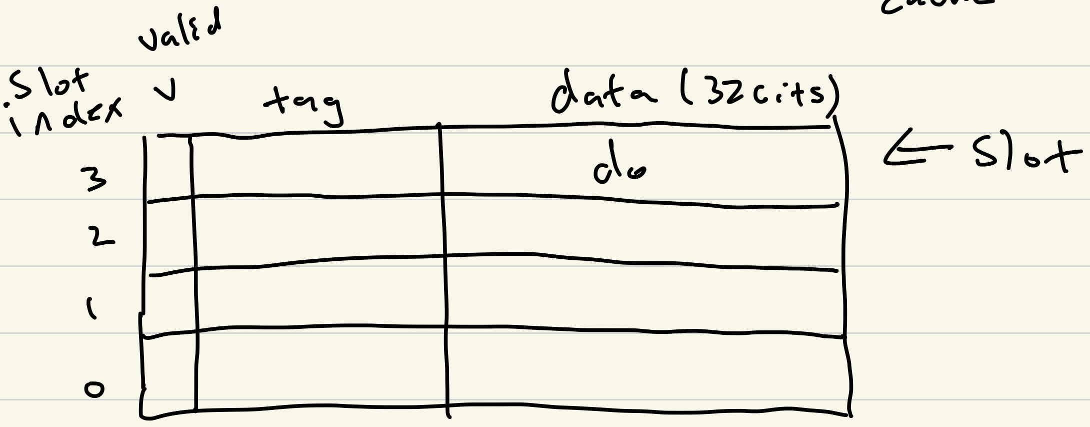
# memory requests

$$\text{hit rate} = \frac{\#\text{ hits}}{\#\text{ reqs}}$$

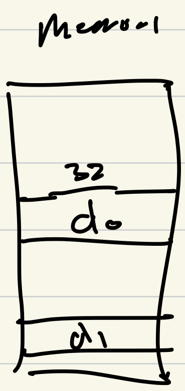
$$\text{miss rate} = \frac{\#\text{ misses}}{\#\text{ reqs}}$$

Direct Mapped

4 word cache



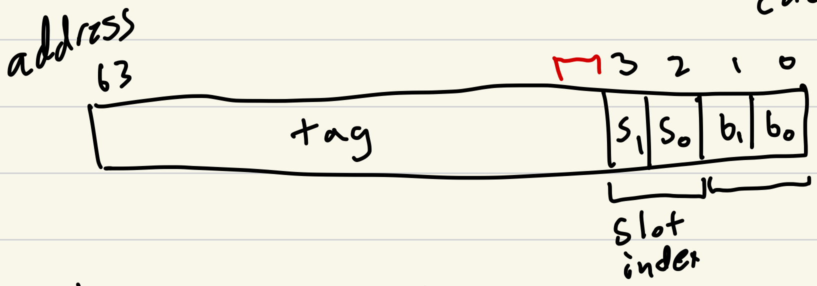
addr assume addr is word aligned



addr_word = addr / 4
byte

slot_index = addr_word % 4

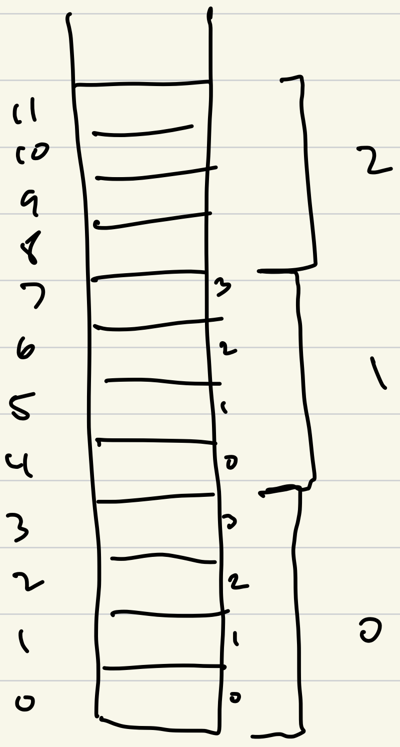
N
# of slot in cache



slot index = (addr >> 2) & 0b11

tag = addr >> 4

Memory



byte addr

word addr

Direct Mapped Pseudo Code

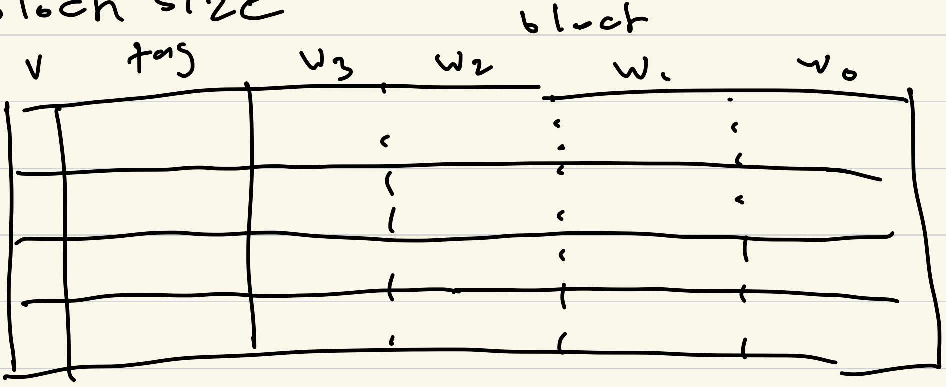
```
tag = addr >> 4;
index_mask = 0b11
slot_index = (addr >> 2) & index_mask
 slot = cache[slot_index];
if (slot.valid == 1 && slot.tag == tag) {
     // hit
      return slot.data
} else {
     // miss
      slot.data = *((uint32_t*) addr)
      slot.tag = tag
     slot.valid = 1
}
```

Principles of locality

temporal

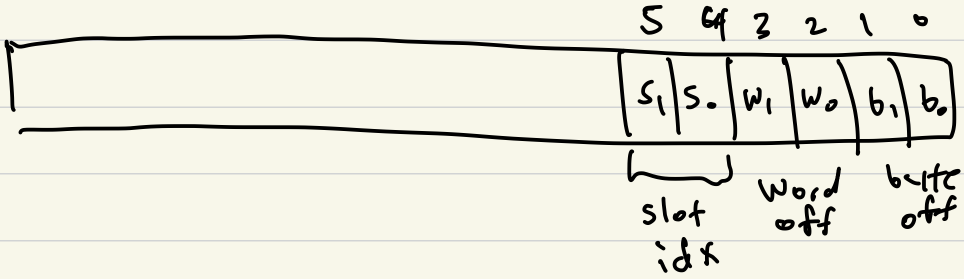
spatial

Block size



addr

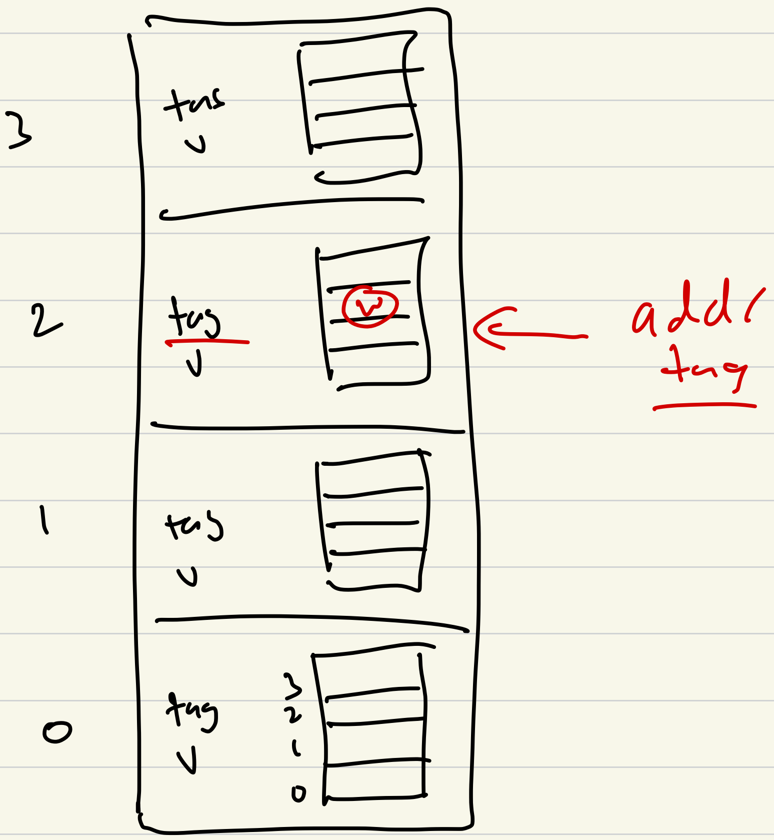
addr_word = addr / 4



slot_idx = addr_word % ~~4~~ 16

$\text{slot\_idx} = \text{addr} >> \underset{\text{bits}}{\underline{4}}$

slots array



hit

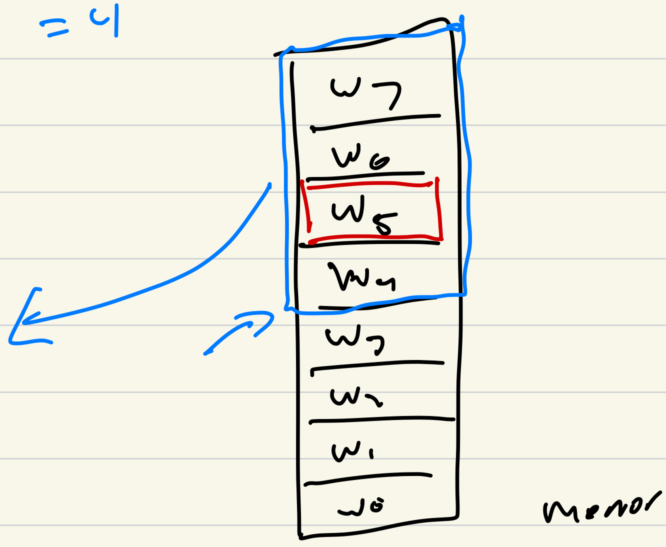
data = slot.block[o];

x

block size = 4

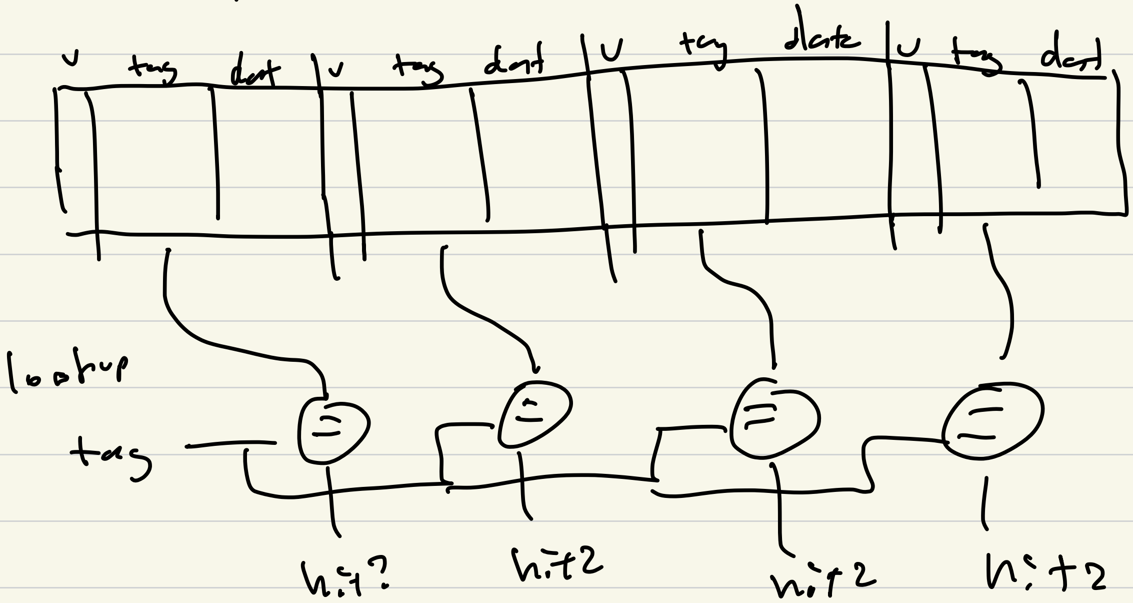
miss

addr

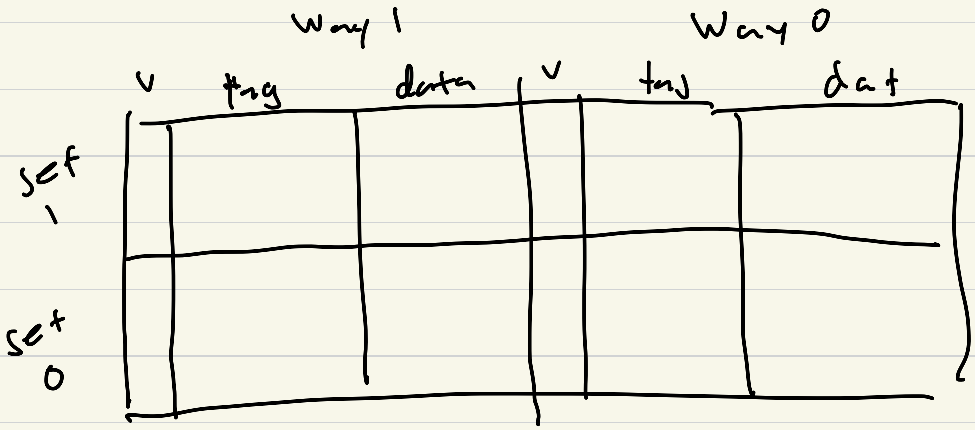


read entire block into cache slot

Fully Associative Cache

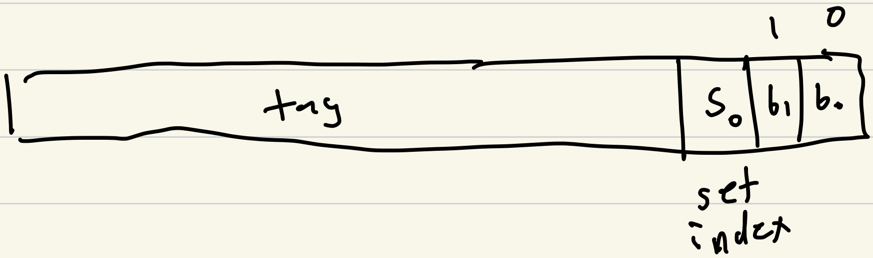


Set Associative Cache



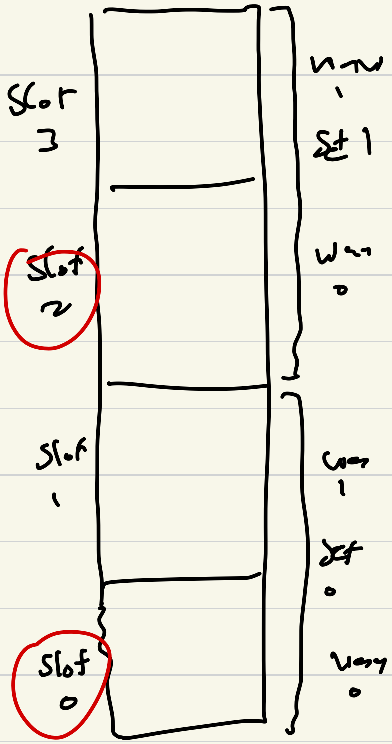
2-way
n-way set associative cache

addr



SA Pseudo Code Lookup

```
num_ref += 1;
 num_ways = 2;
 addr_tag = addr >> 3
 set_index = (addr >> 2) & 0b1
 set_base = set_index * 2
for (i=0; i<2; i++) {
    slot = cache[set_base + i]
    if (slot.valid && slot.tag == tag)
      // hit
      slot.timestamp = num_refs;
      return slot.data
}
// miss
 slot = find_lru_in_set(cache, set_base)
 slot.data = ~~*~~((uint32_t ~~*~~) addr)
 slot.tag = tag
 slot.timestamp = num_refs
 return slot.data
```



Slot 3
way 1
Set 1
Slot 2
way 0
Slot 1
way 1
Set 0
Slot 0
way 0