

# Lecture 16: Cache policies

Philipp Koehn

October 12, 2022

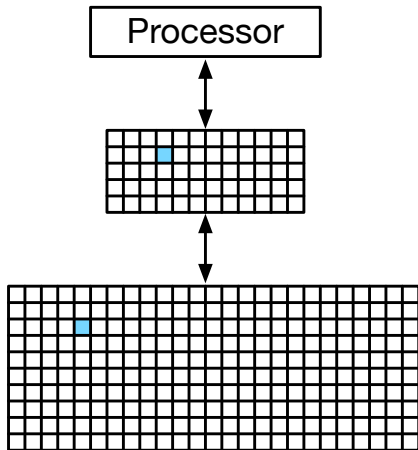
601.229 Computer Systems Fundamentals



# Memory Tradeoff

- ▶ Fastest memory is on same chip as CPU  
... but it is not very big (say, 32 KB in L1 cache)
- ▶ Slowest memory is DRAM on different chips  
... but can be very large (say, 256GB in compute server)
- ▶ Goal: illusion that large memory is fast
- ▶ Idea: use small memory as cache for large memory
- ▶ Note: in reality there are additional levels of cache (L1, L2, L3)

# Simplified View



Smaller memory mirrors some of the large memory content

# Cache organization

## Previously: Direct Mapping

- ▶ Each memory block is mapped to a specific slot in cache
- ⇒ Use part of the address as index to cache

|                |                |           |
|----------------|----------------|-----------|
| 0010 0011 1101 | 1100 0001 0011 | 1010 1111 |
| Tag            | Index          | Offset    |

- ▶ Since multiple memory blocks are mapped to same slot  
→ contention, newly loaded blocks discard old ones

# Concerns

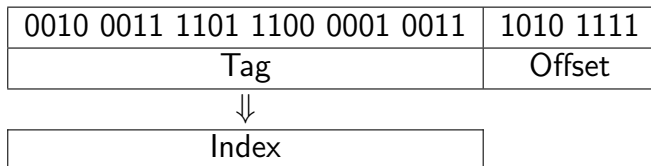
- ▶ Is this the best we got?
- ▶ Some benefits from locality:  
neighboring memory blocks placed in different cache slots
- ▶ But: we may have to pre-empt useful cached blocks
- ▶ We do not even know which ones are still useful

# Now: Associative Cache

- ▶ Place block anywhere in cache
- ⇒ Block tag now full block address in main memory
- ▶ Previously: 32-bit memory address gets mapped to

|                |                |           |
|----------------|----------------|-----------|
| 0010 0011 1101 | 1100 0001 0011 | 1010 1111 |
| Tag            | Index          | Offset    |

- ▶ Now



# Cache Organization

- ▶ Cache sizes
  - ▶ block size: 256 bytes (8 bit offset)
  - ▶ cache size: 1MB (4096 slots)

|      | <b>Tag<br/>(24 bits)</b> | <b>Valid<br/>(1 bit)</b> | <b>Data<br/>256 bytes</b> |
|------|--------------------------|--------------------------|---------------------------|
| 0    |                          |                          |                           |
| 1    |                          |                          |                           |
| ...  |                          |                          |                           |
| 4095 |                          |                          |                           |



# Cache Organization

- ▶ Cache sizes
  - ▶ block size: 256 bytes (8 bit offset)
  - ▶ cache size: 1MB (4096 slots)

|      | <b>Tag<br/>(24 bits)</b> | <b>Valid<br/>(1 bit)</b> | <b>Data<br/>256 bytes</b> |
|------|--------------------------|--------------------------|---------------------------|
| 0    |                          |                          |                           |
| 1    | \$d0f012                 | 1                        | 93 f4 8d 19 ....          |
| ...  |                          |                          |                           |
| 4095 |                          |                          |                           |

- ▶ Read memory value for address \$d0f01234
  - ▶ cache miss → load into cache
  - ▶ data block: \$d0f01200-\$d0f012ff
  - ▶ tag: \$d0f012
  - ▶ placed somewhere (say, index 1)

# Trade-Off

- ▶ Direct mapping (slot determined from address)
    - ▶ disadvantage: two useful blocks contend for same slot
      - many cache misses
  - ▶ Associative (lookup table for slot)
    - ▶ disadvantage: finding block in cache expensive
      - slow, power-hungry
- ⇒ Looking for a compromise

# Set-Associative Cache

- ▶ Mix of direct and associative mapping
- ▶ From direct mapping:  
use part of the address to determine a subset of cache

|                   |              |           |
|-------------------|--------------|-----------|
| 0010 0011 1101 11 | 00 0001 0011 | 1010 1111 |
| Tag               | Index        | Offset    |

- ▶ Associative mapping:  
more than one slot for each indexed part of cache

# Cache Organization

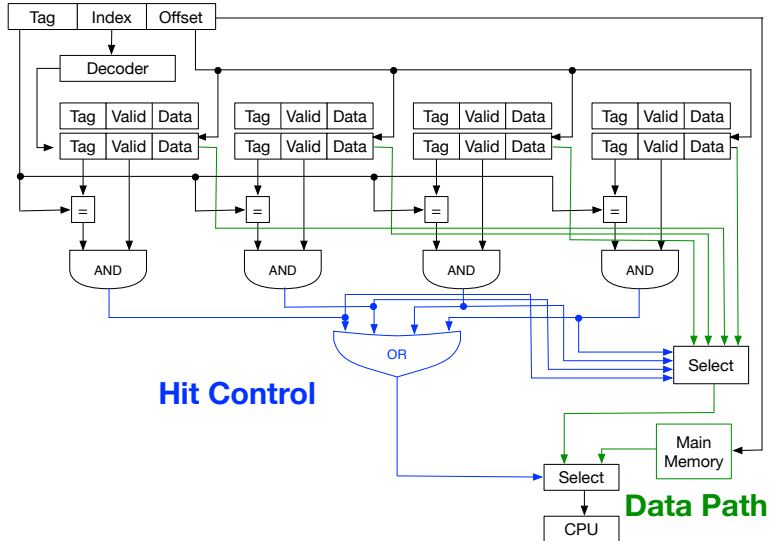
- ▶ Cache sizes
  - ▶ block size: 256 bytes (8 bit offset)
  - ▶ cache size: 1MB (1024 sets of 4 slots)

| <b>Index</b> | <b>Tag<br/>(14 bits)</b> | <b>Valid<br/>(1 bit)</b> | <b>Data<br/>256 bytes</b> |
|--------------|--------------------------|--------------------------|---------------------------|
| 0            |                          |                          |                           |
|              |                          |                          |                           |
|              |                          |                          |                           |
|              |                          |                          |                           |
| 1            |                          |                          |                           |
|              |                          |                          |                           |
| ...          | ...                      | ...                      | ...                       |

# Clicker quiz!

Clicker quiz omitted from public slides

# Cache Read Control (4-Way Set Associative)



# Caching Strategies

- ▶ Read in blocks as needed
- ▶ If cache full, discard blocks based on
  - ▶ randomly

# Caching Strategies

- ▶ Read in blocks as needed
- ▶ If cache full, discard blocks based on
  - ▶ randomly
  - ▶ number of times accessed



# Caching Strategies

- ▶ Read in blocks as needed
- ▶ If cache full, discard blocks based on
  - ▶ randomly
  - ▶ number of times accessed
  - ▶ least recently used

# Caching Strategies

- ▶ Read in blocks as needed
- ▶ If cache full, discard blocks based on
  - ▶ randomly
  - ▶ number of times accessed
  - ▶ least recently used
  - ▶ first in, first out

First in, first out

# First In, First Out (FIFO)

- ▶ Consider order in which cache blocks loaded
  - ▶ Oldest block gets discarded first
- ⇒ Need to keep a record of when blocks were loaded

# Timestamp

- ▶ Each record requires additional timestamp

| Index | Tag<br>(14 bits) | Valid<br>(1 bit) | Timestamp | Data<br>256 bytes |
|-------|------------------|------------------|-----------|-------------------|
| 0     |                  |                  |           |                   |
|       |                  |                  |           |                   |
|       |                  |                  |           |                   |
|       |                  |                  |           |                   |
| 1     |                  |                  |           |                   |
|       |                  |                  |           |                   |
|       |                  |                  |           |                   |
| ...   | ...              | ...              | ...       | ...               |

- ▶ Store actual time?
  - ▶ time can be easily set when slot filled
  - ▶ but: finding oldest slot requires loop with min calculation

# Maintain Order

- ▶ Actual access time not needed, but ordering of cache
- ▶ For instance, for 4-way associative array
  - ▶ 0 = newest block
  - ▶ 3 = oldest block
- ▶ When new slot needed
  - ▶ find slot with timestamp value 3
  - ▶ use slot for new memory block
  - ▶ increase all timestamp counters by 1

# Example

► Initial

| Index | Tag<br>(14 bits) | Valid<br>(1 bit) | Order | Data<br>256 bytes |
|-------|------------------|------------------|-------|-------------------|
| 0     |                  | 0                |       |                   |
|       |                  | 0                |       |                   |
|       |                  | 0                |       |                   |
|       |                  | 0                |       |                   |

# Example

- ▶ First block

| <b>Index</b> | <b>Tag<br/>(14 bits)</b> | <b>Valid<br/>(1 bit)</b> | <b>Order</b> | <b>Data<br/>256 bytes</b> |
|--------------|--------------------------|--------------------------|--------------|---------------------------|
| 0            | 3e12                     | 0                        | 11           | 4f 4e 53 ff 00 01 .....   |
|              |                          | 0                        | 10           |                           |
|              |                          | 0                        | 01           |                           |
|              |                          | 0                        | 00           |                           |

- ▶ All valid bits are 0
- ▶ Each slot has unique order value



# Example

- ▶ Second block

| Index | Tag<br>(14 bits) | Valid<br>(1 bit) | Order | Data<br>256 bytes       |
|-------|------------------|------------------|-------|-------------------------|
| 0     | 3e12             | 1                | 01    | 4f 4e 53 ff 00 01 ..... |
|       | 0ff0             | 1                | 00    | 00 01 f0 01 02 63 ..... |
|       |                  | 0                | 11    |                         |
|       |                  | 0                | 10    |                         |

- ▶ Load data
- ▶ Set valid bit
- ▶ Increase order counters

# Example

- ▶ Third block

| Index | Tag<br>(14 bits) | Valid<br>(1 bit) | Order | Data<br>256 bytes       |
|-------|------------------|------------------|-------|-------------------------|
| 0     | 3e12             | 1                | 10    | 4f 4e 53 ff 00 01 ..... |
|       | 0ff0             | 1                | 01    | 00 01 f0 01 02 63 ..... |
|       | 6043             | 1                | 00    | f0 f0 f0 34 12 60 ..... |
|       |                  | 0                | 11    |                         |

- ▶ Load data
- ▶ Set valid bit
- ▶ Increase order counters

# Example

- ▶ Fourth block

| Index | Tag<br>(14 bits) | Valid<br>(1 bit) | Order | Data<br>256 bytes       |
|-------|------------------|------------------|-------|-------------------------|
| 0     | 3e12             | 1                | 11    | 4f 4e 53 ff 00 01 ..... |
|       | 0ff0             | 1                | 10    | 00 01 f0 01 02 63 ..... |
|       | 2043             | 1                | 01    | f0 f0 f0 34 12 60 ..... |
|       | 37ab             | 1                | 00    | 4a 42 43 52 4a 4a ..... |

- ▶ Load data
- ▶ Set valid bit
- ▶ Increase order counters

# Example

- ▶ Fifth block

| Index | Tag<br>(14 bits) | Valid<br>(1 bit) | Order | Data<br>256 bytes       |
|-------|------------------|------------------|-------|-------------------------|
| 0     | 0561             | 1                | 00    | 9a 8b 7d 3d 4a 44 ..... |
|       | 0ff0             | 1                | 11    | 00 01 f0 01 02 63 ..... |
|       | 2043             | 1                | 10    | f0 f0 f0 34 12 60 ..... |
|       | 37ab             | 1                | 01    | 4a 42 43 52 4a 4a ..... |

- ▶ Discard oldest block
- ▶ Load new data
- ▶ Increase order counters

Least recently used

# Least Recently Used (LRU)

- ▶ Base decision on last-used time, not load time
  - ▶ Keeps frequently used blocks longer in cache
  - ▶ Also need to maintain order
- ⇒ Update with every access (not just miss)

# Example

| Slot 0 |       | Slot 1 |       | Slot 2 |       | Slot 3 |       |
|--------|-------|--------|-------|--------|-------|--------|-------|
| Access | Order | Access | Order | Access | Order | Access | Order |
|        | 01    |        | 11    |        | 10    |        | 00    |

# Example

| Slot 0 |       | Slot 1 |       | Slot 2 |       | Slot 3 |       |
|--------|-------|--------|-------|--------|-------|--------|-------|
| Access | Order | Access | Order | Access | Order | Access | Order |
|        | 01    |        | 11    |        | 10    |        | 00    |
|        | 01    |        | 11    |        | 10    | Hit    | 00    |



# Example

| Slot 0 |       | Slot 1 |       | Slot 2 |       | Slot 3 |       |
|--------|-------|--------|-------|--------|-------|--------|-------|
| Access | Order | Access | Order | Access | Order | Access | Order |
|        | 01    |        | 11    |        | 10    |        | 00    |
|        | 01    |        | 11    |        | 10    | Hit    | 00    |
|        | 10    | Hit    | 00    |        | 11    |        | 01    |

# Example

| Slot 0 |       | Slot 1 |       | Slot 2 |       | Slot 3 |       |
|--------|-------|--------|-------|--------|-------|--------|-------|
| Access | Order | Access | Order | Access | Order | Access | Order |
|        | 01    |        | 11    |        | 10    |        | 00    |
|        | 01    |        | 11    |        | 10    | Hit    | 00    |
|        | 10    | Hit    | 00    |        | 11    |        | 01    |
| Hit    | 00    |        | 01    |        | 11    |        | 10    |

# Example

| Slot 0 |       | Slot 1 |       | Slot 2 |       | Slot 3 |       |
|--------|-------|--------|-------|--------|-------|--------|-------|
| Access | Order | Access | Order | Access | Order | Access | Order |
|        | 01    |        | 11    |        | 10    |        | 00    |
|        | 01    |        | 11    |        | 10    | Hit    | 00    |
|        | 10    | Hit    | 00    |        | 11    |        | 01    |
| Hit    | 00    |        | 01    |        | 11    |        | 10    |
|        | 01    |        | 10    | Miss   | 00    |        | 11    |

# Example

| Slot 0 |       | Slot 1 |       | Slot 2 |       | Slot 3 |       |
|--------|-------|--------|-------|--------|-------|--------|-------|
| Access | Order | Access | Order | Access | Order | Access | Order |
|        | 01    |        | 11    |        | 10    |        | 00    |
|        | 01    |        | 11    |        | 10    | Hit    | 00    |
|        | 10    | Hit    | 00    |        | 11    |        | 01    |
| Hit    | 00    |        | 01    |        | 11    |        | 10    |
|        | 01    |        | 10    | Miss   | 00    |        | 11    |

- ▶ Miss: set to 0 and increase other counters
- ▶ Hit least recently used: set to 0 and increase other counters
- ▶ Hit most recently used: no change
- ▶ Hit others: increase some counters

# Quite Complicated

- ▶ First look up order of accessed block
- ▶ Compare each other block's order to that value
- ▶ Increasingly costly with higher associativity
- ▶ Note: this has to be done every time memory is **accessed**  
(not just during cache misses)

# Aproximation: Bit Shifting

- ▶ Keep an  $(n-1)$ -bit map for an  $n$ -way associative set
- ▶ Each time a block in a set is accessed
  - ▶ shift all bits to the right
  - ▶ set the highest bit of the accessed block
- ▶ Slot with value 0 is candidate for removal

# Example

| Slot 0 |       | Slot 1 |       | Slot 2 |       | Slot 3 |       |
|--------|-------|--------|-------|--------|-------|--------|-------|
| Access | Order | Access | Order | Access | Order | Access | Order |
|        | 010   |        | 000   |        | 001   |        | 100   |

# Example

| Slot 0 |       | Slot 1 |       | Slot 2 |       | Slot 3 |       |
|--------|-------|--------|-------|--------|-------|--------|-------|
| Access | Order | Access | Order | Access | Order | Access | Order |
|        | 010   |        | 000   |        | 001   |        | 100   |
|        | 001   | Hit    | 100   |        | 000   |        | 010   |



# Example

| Slot 0 |       | Slot 1 |       | Slot 2 |       | Slot 3 |       |
|--------|-------|--------|-------|--------|-------|--------|-------|
| Access | Order | Access | Order | Access | Order | Access | Order |
|        | 010   |        | 000   |        | 001   |        | 100   |
|        | 001   | Hit    | 100   |        | 000   |        | 010   |
|        | 000   |        | 010   | Miss   | 100   |        | 001   |

# Example

| Slot 0 |       | Slot 1 |       | Slot 2 |       | Slot 3 |       |
|--------|-------|--------|-------|--------|-------|--------|-------|
| Access | Order | Access | Order | Access | Order | Access | Order |
|        | 010   |        | 000   |        | 001   |        | 100   |
|        | 001   | Hit    | 100   |        | 000   |        | 010   |
|        | 000   |        | 010   | Miss   | 100   |        | 001   |
|        | 000   | Hit    | 101   |        | 010   |        | 000   |

# Example

| Slot 0 |       | Slot 1 |       | Slot 2 |       | Slot 3 |       |
|--------|-------|--------|-------|--------|-------|--------|-------|
| Access | Order | Access | Order | Access | Order | Access | Order |
|        | 010   |        | 000   |        | 001   |        | 100   |
|        | 001   | Hit    | 100   |        | 000   |        | 010   |
|        | 000   |        | 010   | Miss   | 100   |        | 001   |
|        | 000   | Hit    | 101   |        | 010   |        | 000   |
|        | 000   | Hit    | 110   |        | 001   |        | 000   |

# Example

| Slot 0 |       | Slot 1 |       | Slot 2 |       | Slot 3 |       |
|--------|-------|--------|-------|--------|-------|--------|-------|
| Access | Order | Access | Order | Access | Order | Access | Order |
|        | 010   |        | 000   |        | 001   |        | 100   |
|        | 001   | Hit    | 100   |        | 000   |        | 010   |
|        | 000   |        | 010   | Miss   | 100   |        | 001   |
|        | 000   | Hit    | 101   |        | 010   |        | 000   |
|        | 000   | Hit    | 110   |        | 001   |        | 000   |
| Miss   | 100   |        | 011   |        | 000   |        | 000   |

# Example

| Slot 0 |       | Slot 1 |       | Slot 2 |       | Slot 3 |       |
|--------|-------|--------|-------|--------|-------|--------|-------|
| Access | Order | Access | Order | Access | Order | Access | Order |
|        | 010   |        | 000   |        | 001   |        | 100   |
|        | 001   | Hit    | 100   |        | 000   |        | 010   |
|        | 000   |        | 010   | Miss   | 100   |        | 001   |
|        | 000   | Hit    | 101   |        | 010   |        | 000   |
|        | 000   | Hit    | 110   |        | 001   |        | 000   |
| Miss   | 100   |        | 011   |        | 000   |        | 000   |

- ▶ There may be multiple blocks with order pattern 000  
→ pick one randomly
- ▶ Maybe do not change, if most recently used block is used again