ECSE 425 Lecture 20: Cache Basics

H&P Appendix C

Last Time

- Introduction to Memory Hierarchy
- What are caches?
- Why cache?
- Four Questions
  - Q1: Block placement

Today

- Two Questions:
  - Q1: Block placement
  - Q2: Block identification

Four Memory Hierarchy Questions

Main memory is divided into blocks each consisting of several data elements (*e.g.* bytes)

1. Block placement
   – Where can a block be placed in the upper level?
2. Block identification
   – How is a block found if it is in the upper level?
3. Block replacement
   – Which block should be replaced on a miss?
4. Write strategy
   – What happens on a write?

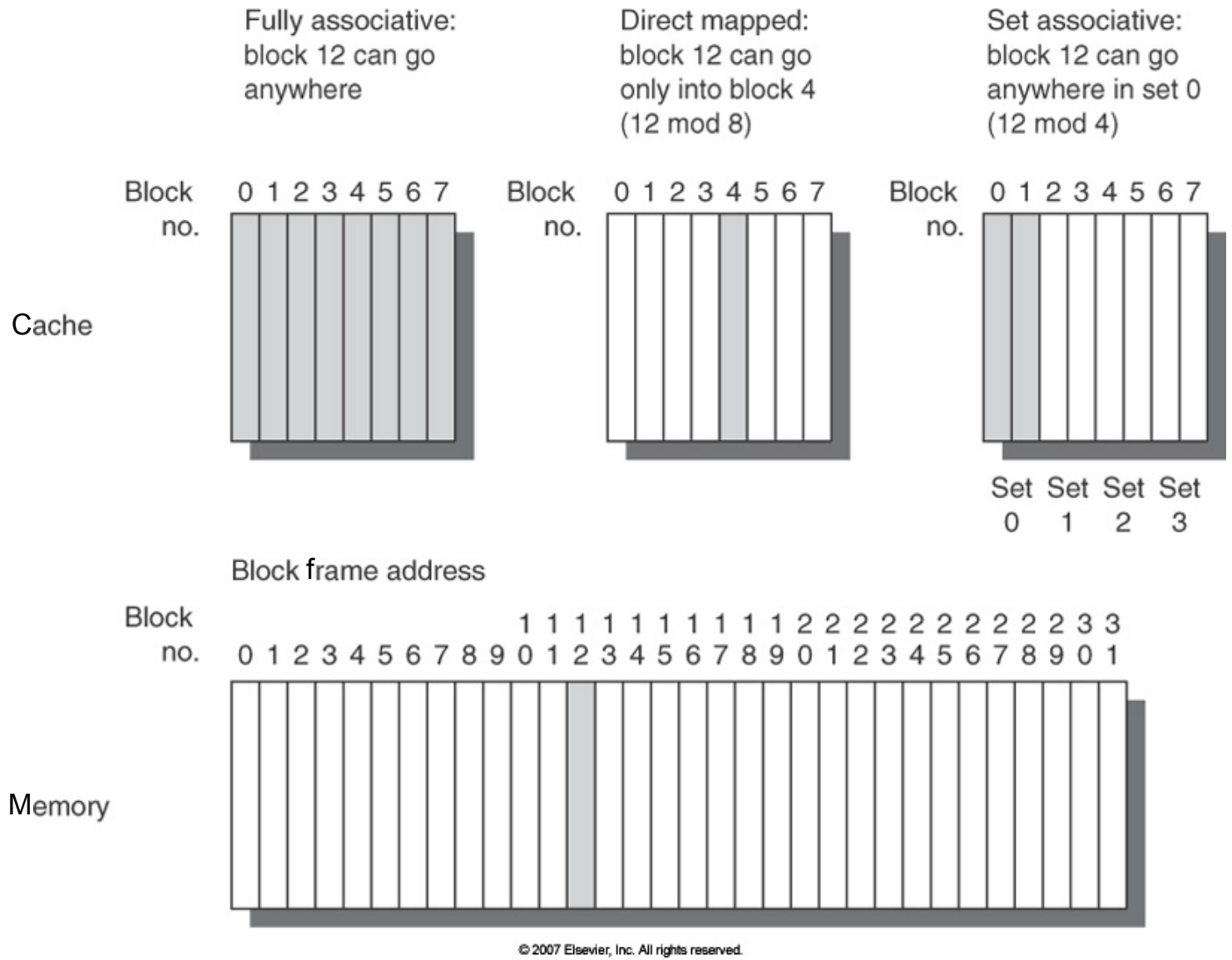
Q1: Block Placement

- Fully associative
  - A block can appear anywhere in the cache
  - Slow and complex, but the best hit rate
- Direct mapped cache
  - Each block can only appear in one place in the cache
    - Block address mod # blocks in cache
  - Fast and simple, but more misses
- Set associative caches strike a balance

Set Associative Caches

- A block can be placed in a *set* of places in the cache (called *ways*)
- First, map a block onto a set
  - Block address mod # sets in cache
- The block can be placed anywhere in that set
  - *n* blocks in a set => *n*-way set associative

Block Placement



Generalized Associativity

- In general, *m* blocks in cache, *n* blocks in a set, *s* sets in cache
  - *m* = *s* * *n*
- *n*-way set associative => *n* > 1 and *s* > 1
  - fully associative => *n* = *m* and *s* = 1 (*m*-way s.a.)
  - direct mapped => *n* = 1 and *s* = *m* (1-way s.a.)
- Mapping: set number is called “index”
- index = block # mod *s*

Real Caches

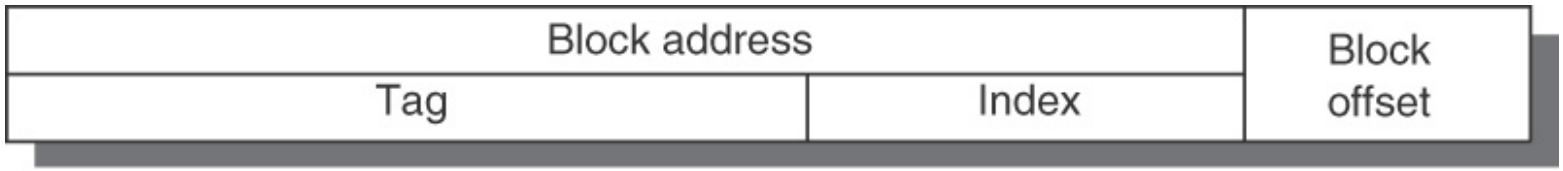
- Most processor caches today are
  - Direct mapped, or *n*-way set associative, $n \leq 8$
- Full-associativity reserved for small, specialized memories
  - E.g., Translation Look-aside Buffer (TLB)

Q2: Block Identification

- Caches hold a subset of blocks in main memory
  - Many blocks map to one or a few entries in cache
  - How is a block found if it is in the cache?
- Divide the memory address into fields
- Block address (higher-order bits)
  - Indicates the block number in memory we will access
- Block offset (lower-order bits)
  - Indicates the data within a block we want to retrieve

Address Fields

- Index field: select the *set* (or entry in the cache)
  - Cache with 2^n sets requires an n-bit index
- Tag field: compared for a hit
  - Many blocks map to the same entries
  - If the tag matches, the entry contains the right data
- Block offset: select the word
  - Typical cache blocks hold 64 bytes—which 4 (or 8) does the CPU need?

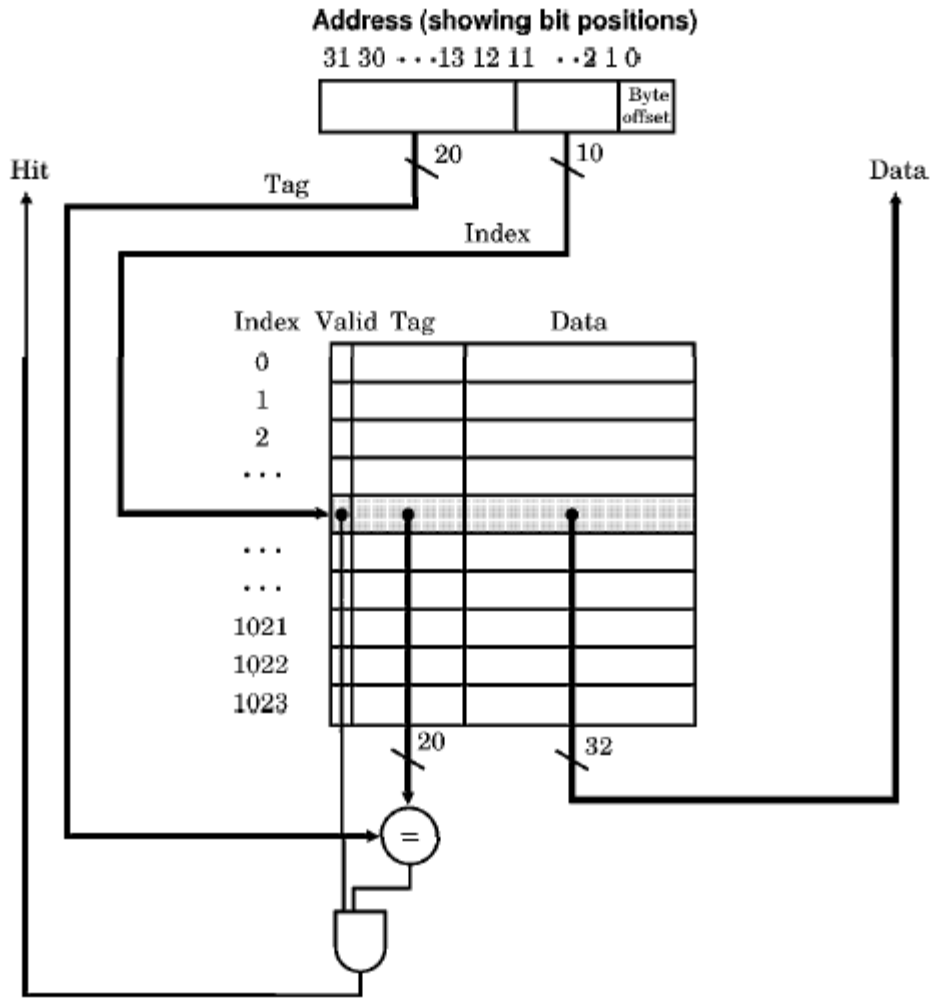


Block Identification by Cache Type

- Searching for a matching block
  - Use index to find the correct set
  - Use tag to determine if correct data is present
- Direct mapped—1 comparison
  - Index directly provides set in cache
- *n*-way set associative ($n > 1$)—*n* comparisons
  - Index identifies the set, *n* tags to compare
  - As *n* increases, index shrinks and tag grows
- Fully associative
  - No index, only tag; compare all tags in the cache
- Parallel tag comparison minimizes hit time

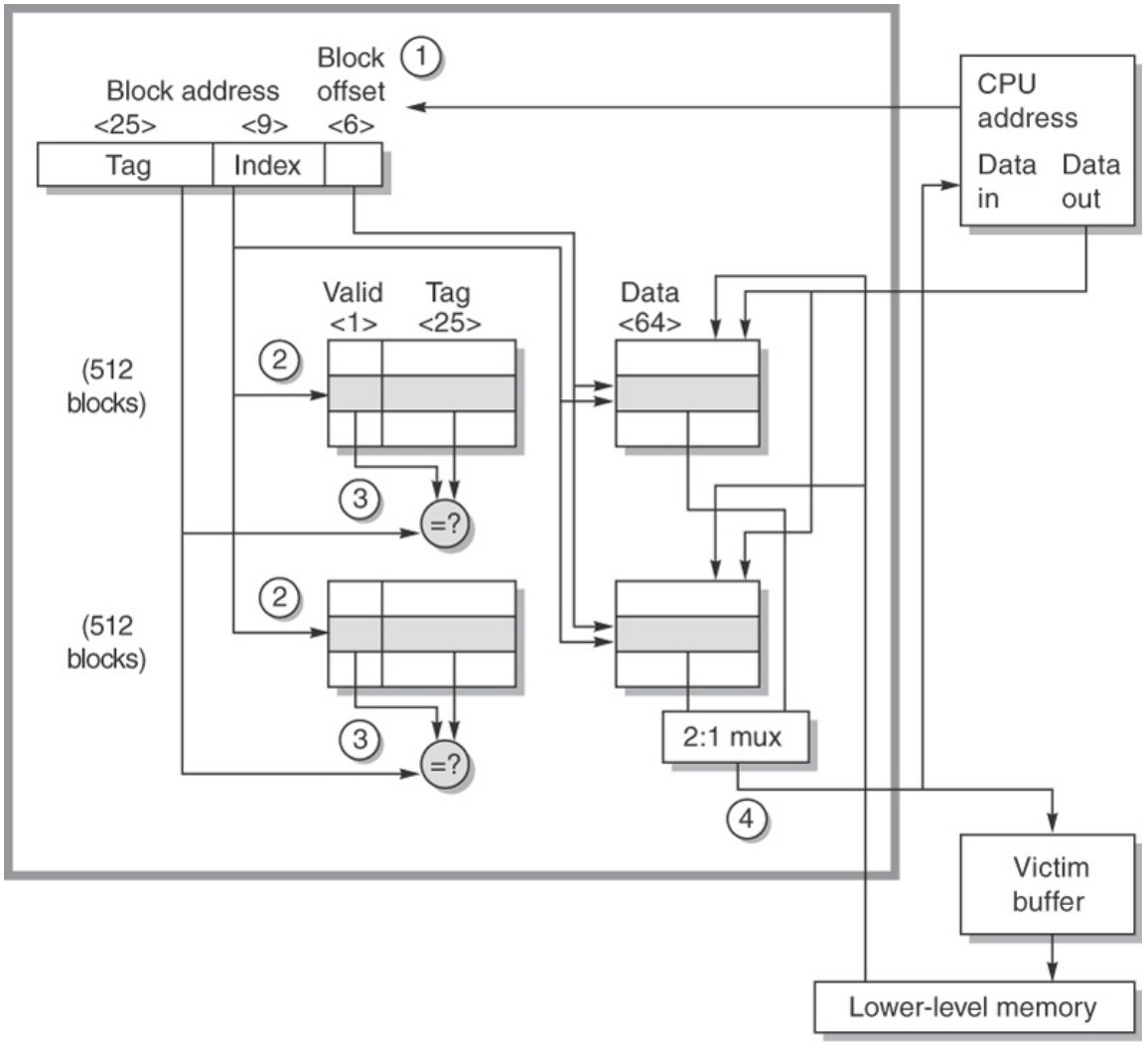
Direct Mapped Cache

- Example
  - 20-bit tag
  - 10-bit index
  - 2-bit offset
- What's the total data storage?
- What's the total required storage (valid, tag and data bits)?



Opteron's 2-way Associative Cache

- 25-bit tag
- 9-bit index
  - 512 sets
- 6-bit offset
  - 64-byte blocks
- 2-way associative
- What's the total data storage?
- What's the total required storage?



Cache Design Example

- Draw the block diagram for a cache
  - 32 KB cache
  - 4-way set associative
  - 32 B blocks
- How many bits are required for each of the address fields?
- What is the total required storage?

Summary

- Block placement
- Block identification

Next Time

- Last two questions
  - And the end of Appendix C.1
- Cache performance
  - Read Appendix C.2!