Cache tradeoffs and metrics

What options do we have when designing a memory hierarchy?

# of levels in hierarchy

block size

cache size

data, instruction, or unified

private or shared cache

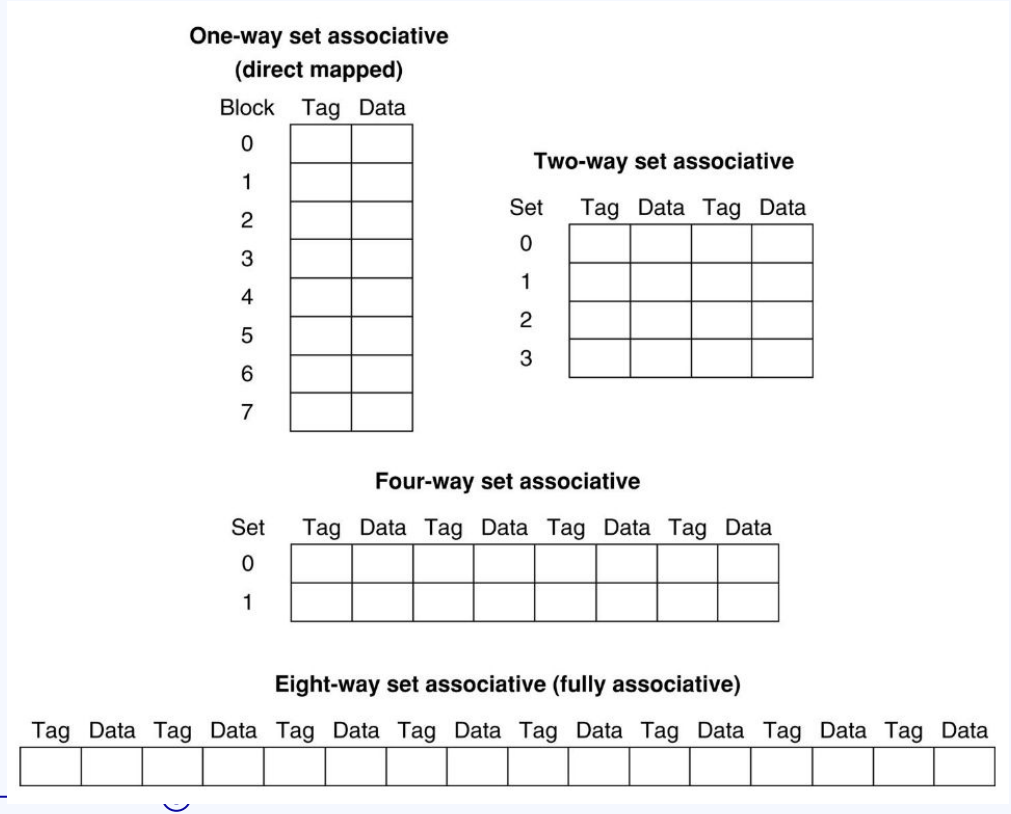
associativity

replacement policy

behavior on write (through/back; allocate; buffers)

how to find a block

Review of associativity



P&H fig. 5.15

Pick a design space and evaluate how it impacts:

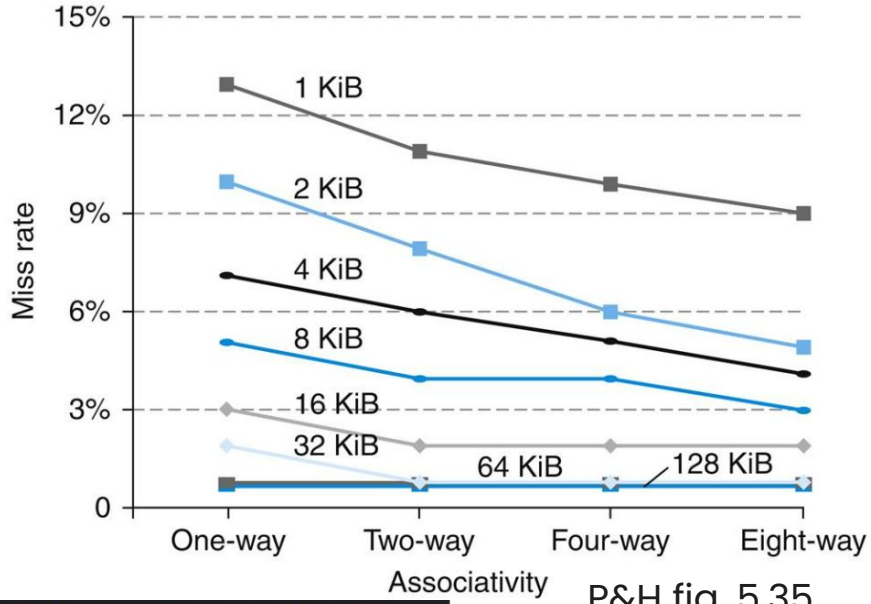
- Miss rate
- Miss penalty
- Hit time
- Other potential consequences?

block size	associativity/ finding a block	cache size	write through vs. back
Decreases miss rate Increases miss penalty Affects whole hierarchy	Decreases miss rate Increases hit time Increases $$/area/energy	Decreases miss rate Increases hit time Increases $$/area/energy	Complicates miss penalty Through: increases hit time Complexity of write buffer

Associativity + performance

Direct-mapped caches aren't really used anymore (gains from even a little bit of associativity are high)

Fully associative caches are costly to implement at large sizes (why fully assoc. TLBs are tiny)

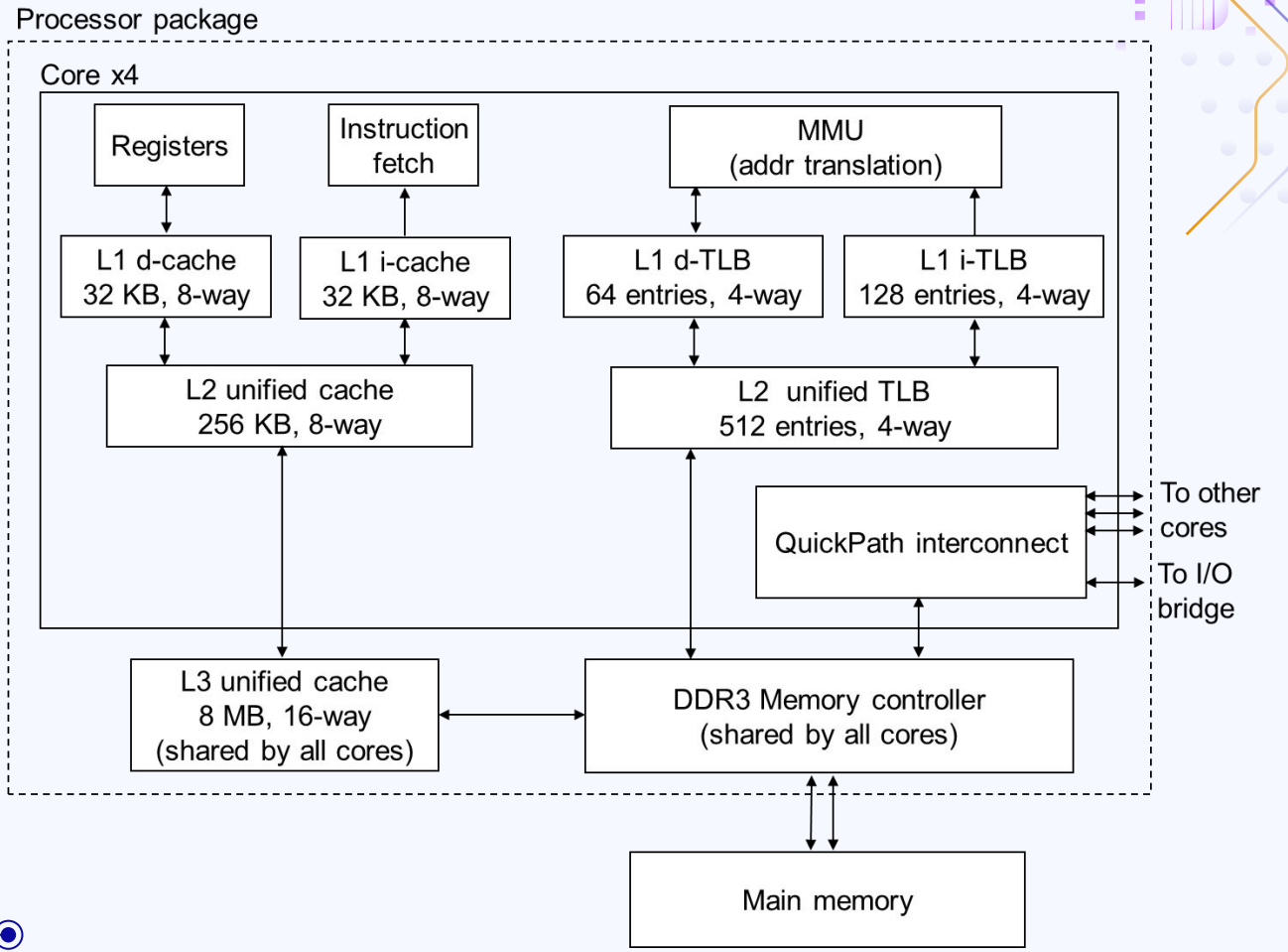


P&H fig. 5.35

Increasing the associativity of the cache reduces the probability of thrashing. The ideal case is a fully associative cache, where any main memory location can map anywhere within the cache. However, building such a cache is impractical for anything other than very small caches, for example, those associated with MMU TLBs. In practice, performance improvements are minimal for above 8-way, with 16-way associativity being more useful for larger L2 caches.

source

Intel i7



Source
(Bryant & O'Hallaron)

Thinking back to everything we've learned so far (CPUs, memory): how has "throwing hardware" at the design helped us?

Design tradeoffs

We designed single-stage CPU for correctness

We designed pipelined CPUs for performance (w/ some complexity tradeoffs)

With memory hierarchy, we encountered the space of *performance tradeoffs*

- Sometimes the answer is to compromise (multiple cache levels; n-way associativity)
- Sometimes the answer is to innovate (TLBs, write buffers, VIPT)

Throwing hardware at the problem has limits and costs ($$, energy, area)

Using advanced tools like gem5 helps us navigate tradeoffs (with a giant caveat!)