Complicating factors in caching (tradeoffs, coherence)

Which of these programs will (generally) perform better for large values of M and N?

```
for (int i = 0;  i < M; i++) {
    for (int j = 0; j < N; j++) {
        x[i][j] *= 2;
    }
}
```

```
for (int j = 0;  j < N; j++) {
    for (int i = 0; i < M; i++) {
        x[i][j] *= 2;
    }
}
```

What options do we have when designing a memory hierarchy?

# of levels in hierarchy

block size

cache size

data, instruction, or unified

associativity

replacement policy

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

how to find a block

? ? ?

Pick a design space and evaluate how it impacts:

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

block size

associativity/ finding a block

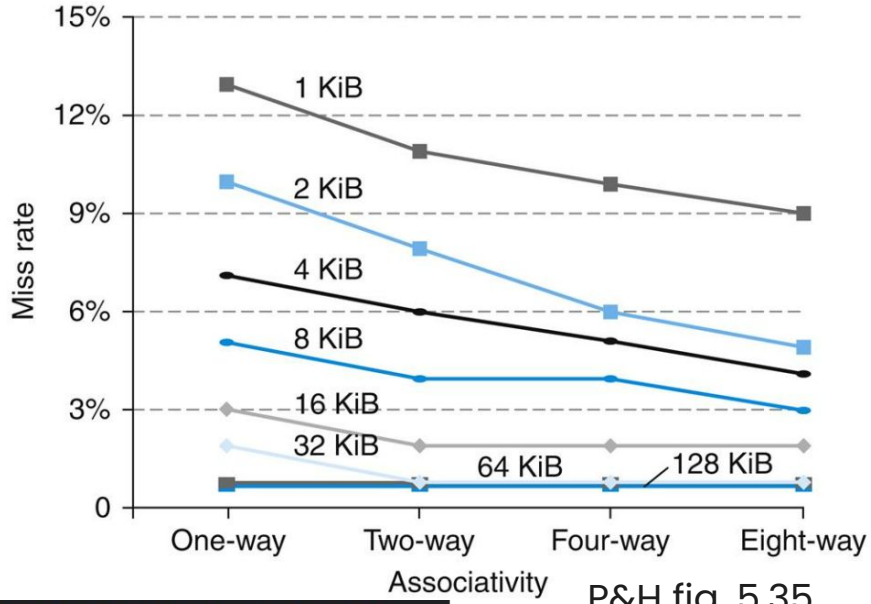
cache size

write through vs. back

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

Advanced optimization: prefetching

Instead of fetching one block on a miss, fetch two (required + next)

    Store next block in a buffer (**why?**)

Used often with I-cache (**why?**)

Possible to have compiler support with special prefetch instructions

Downsides?

Yet more options

Way prediction (hw3)

Optimize data transfer

- Parallelize cache access (banked caches)
- Pipeline the cache
- Critical word first/early restart

Non-blocking cache (on out-of-order processors)

Design tradeoffs

What did we think about when designing a CPU?

In memory hierarchy, we see *performance tradeoffs*

- Sometimes the answer is to compromise
- Sometimes the answer is to innovate

Throwing hardware at the problem has limits and costs

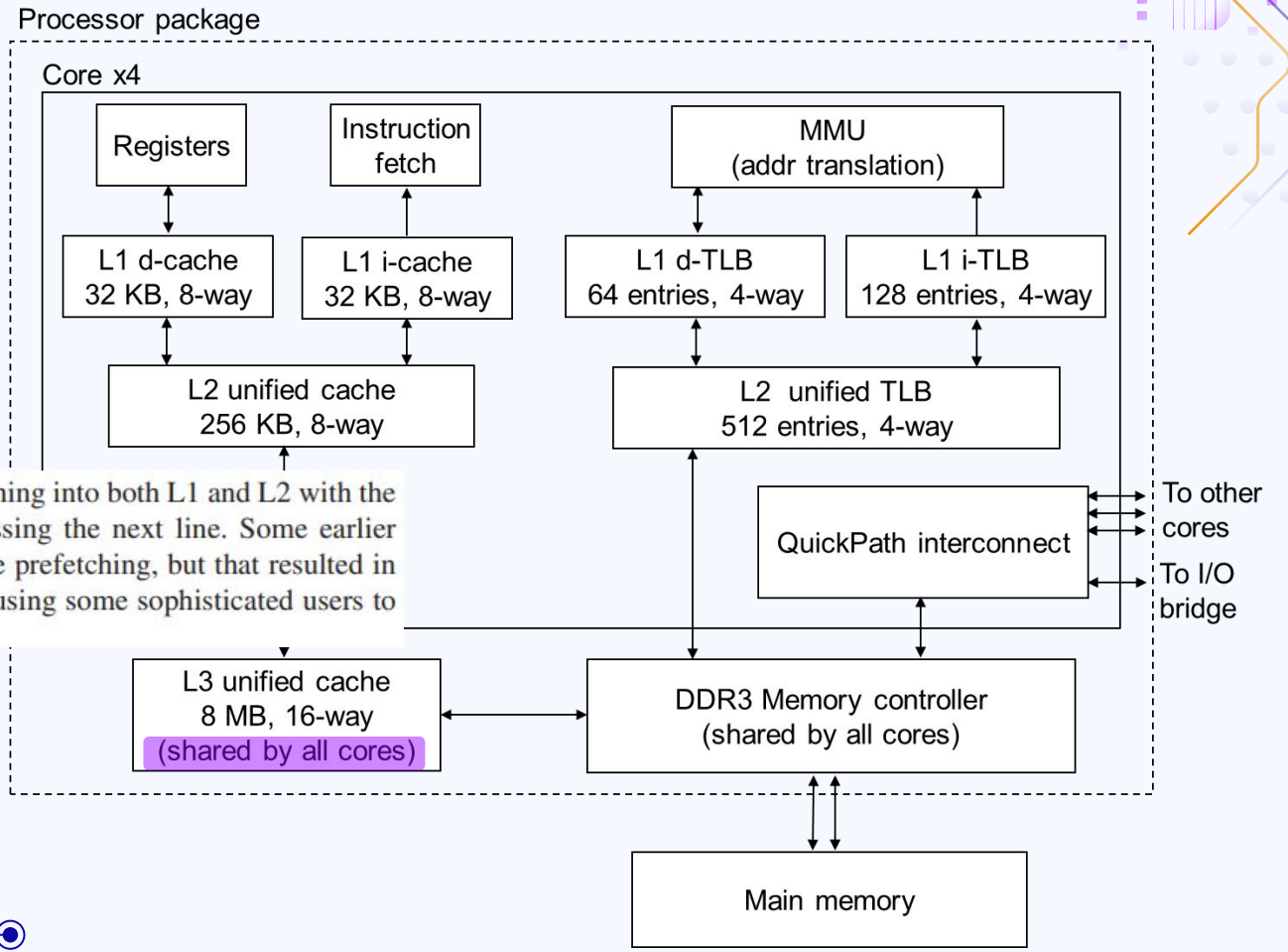
Real world complicates things a *lot*

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

Don’t forget the role of software/compiler on performance

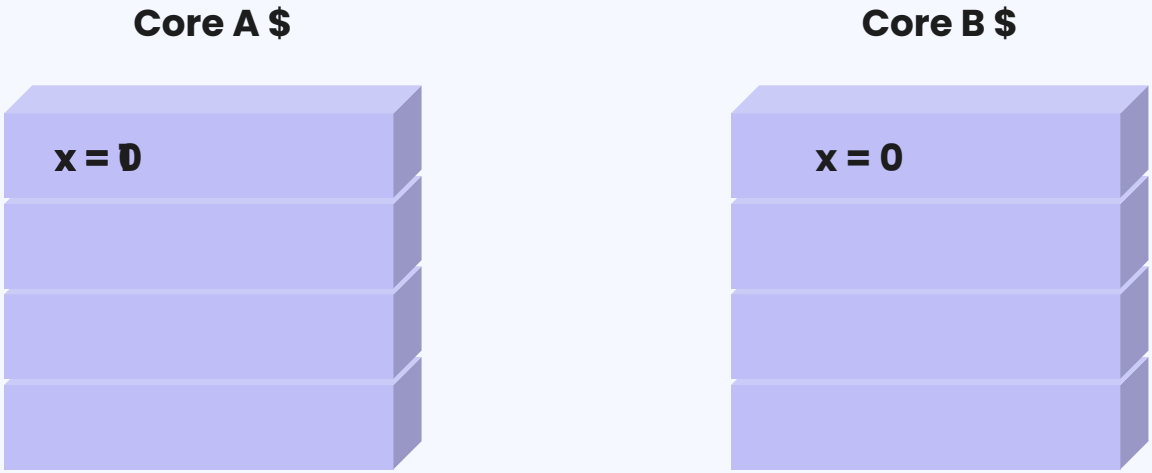
Intel i7

Source
(Bryant & O'Hallaron)



The Intel Core i7 supports hardware prefetching into both L1 and L2 with the most common case of prefetching being accessing the next line. Some earlier Intel processors used more aggressive hardware prefetching, but that resulted in reduced performance for some applications, causing some sophisticated users to turn off the capability.

Cache coherence problem



Definitions

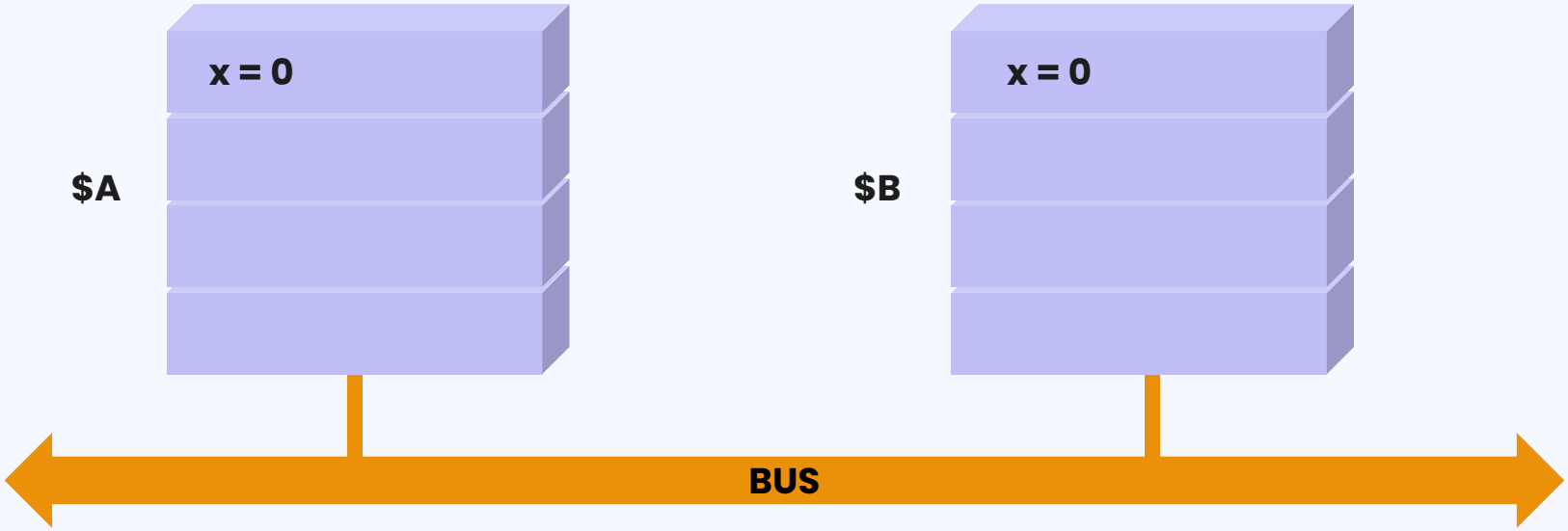
Intuitively: want any read of an item to return most recently written value to item

Coherence – what values can be returned by a read?

1. On a uniprocessor: reads after writes return written value
2. On a multiprocessor: reads by B after writes by A return written value when given sufficient time
3. Two writes to same location by one processor are seen in the same order by all processors

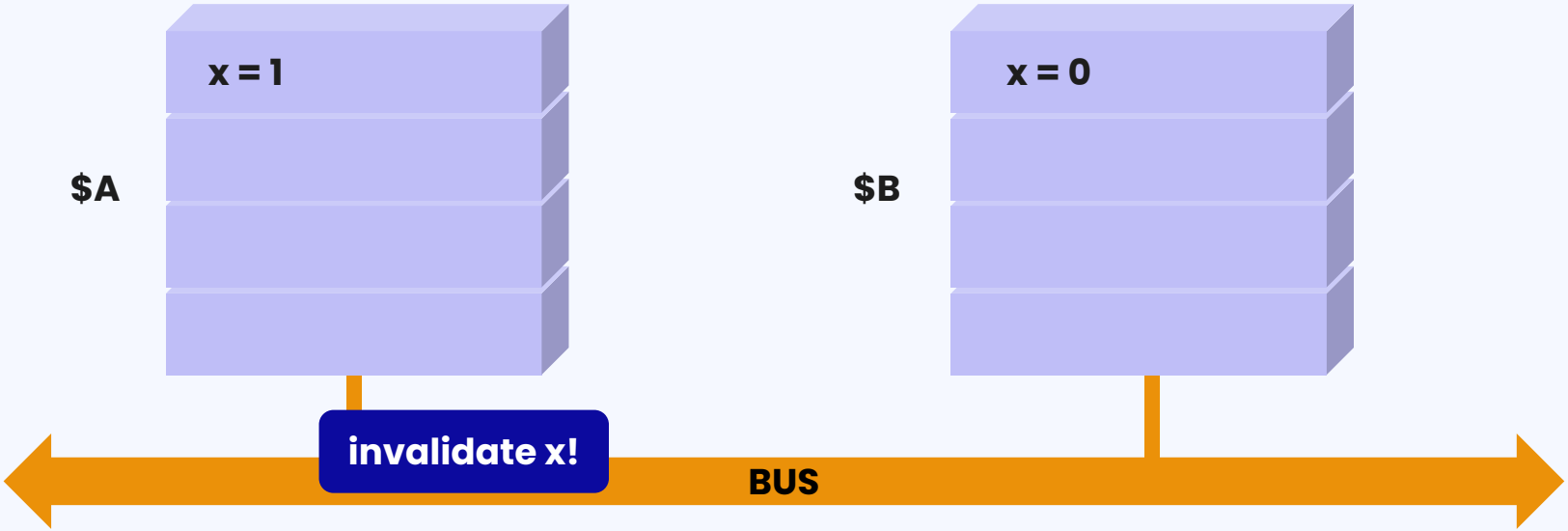
Snooping🔍

1. A reads x
2. B reads x



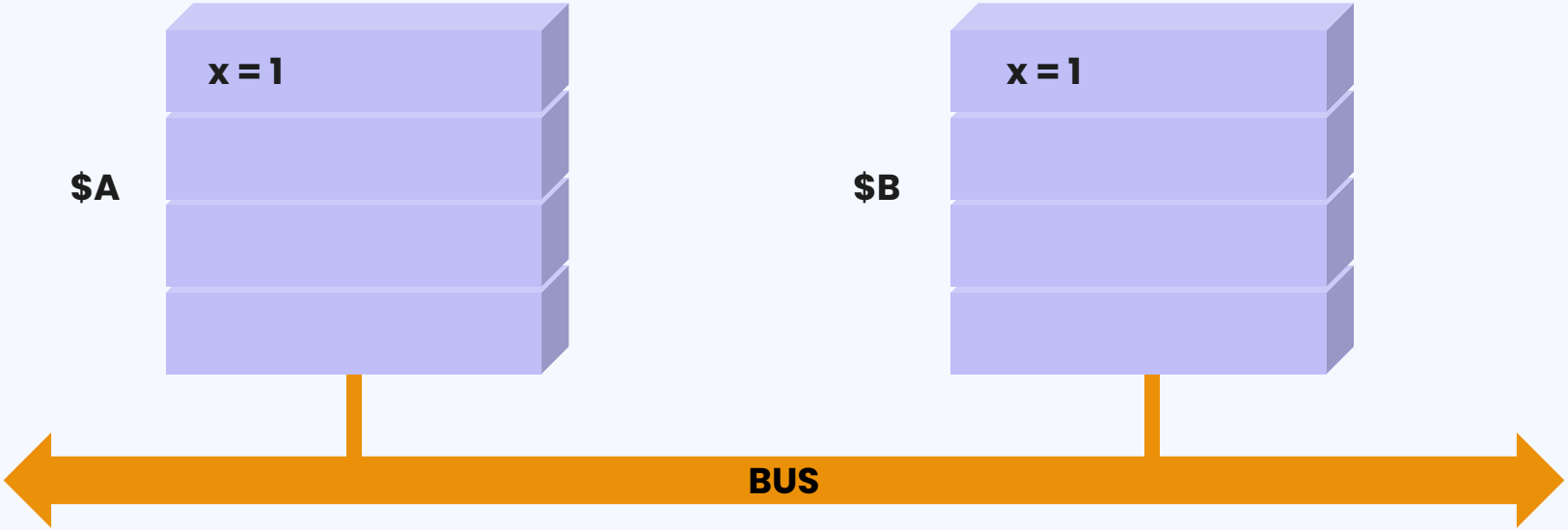
3. A writes x

Snooping🔍



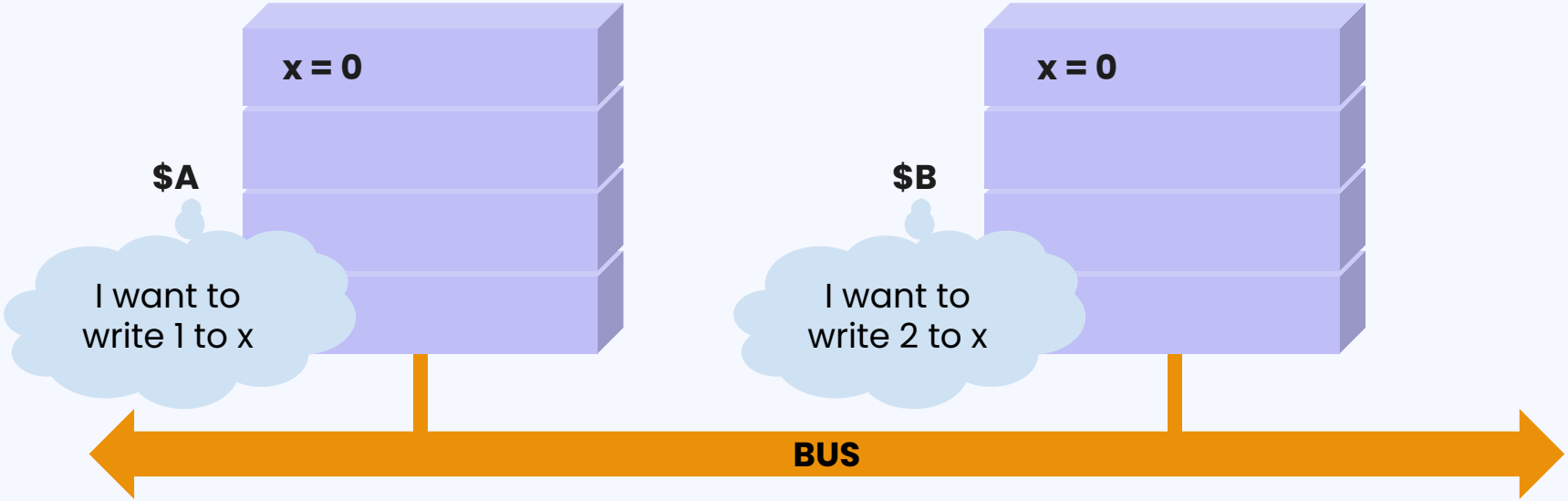
4. B reads x

Snooping🔎



Cache needs to keep state for each block based on bus messages (common protocol: MSI)

What should happen here?



What effects does block size have on coherence protocols?