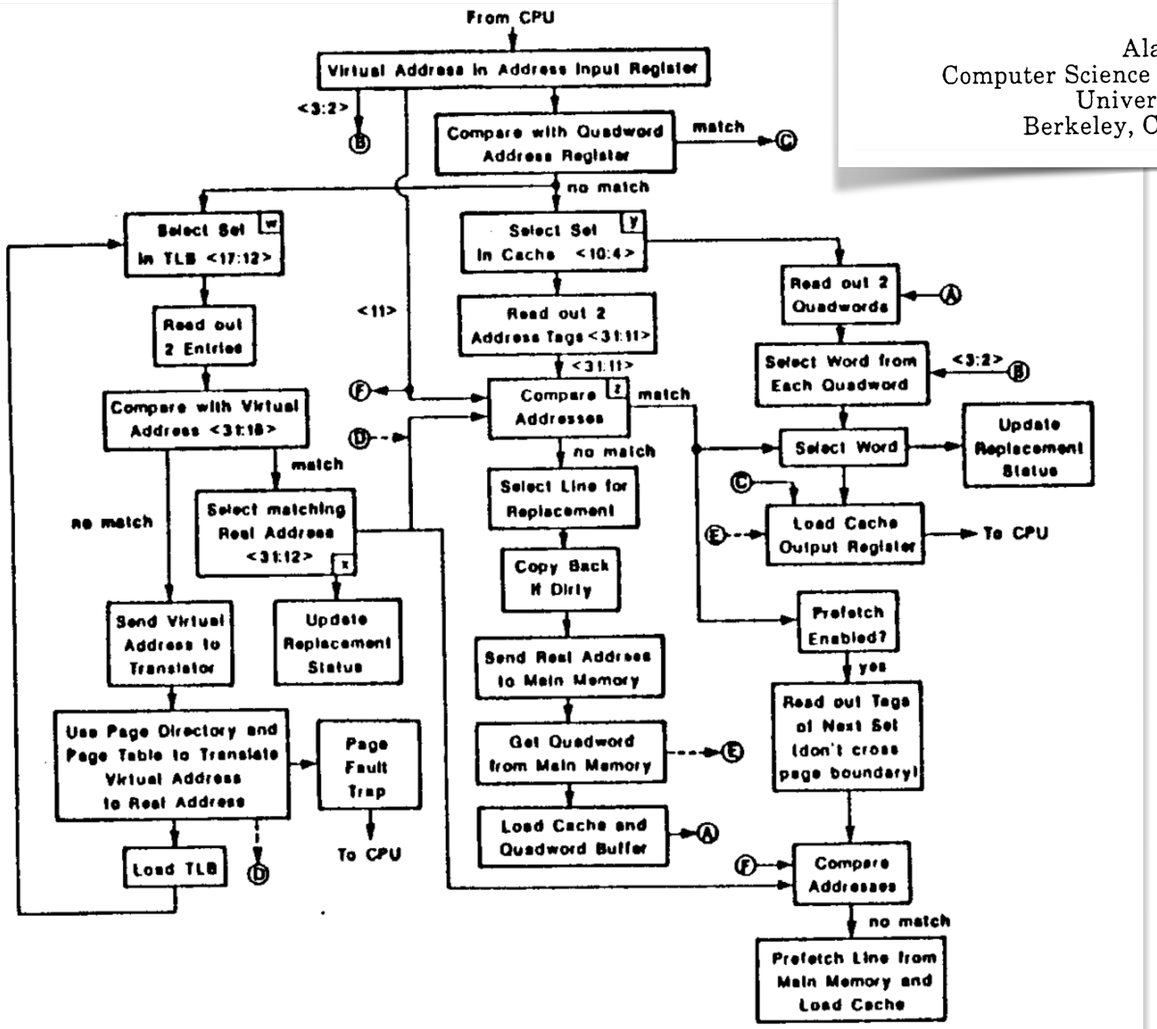
Mitigating Cache Misses

HW2 (part 1) due tonight!
Colloquium in Rose Hill Theater after class

Design of CPU Cache Memories[†]

Alan Jay Smith
Computer Science Division, EECS Department
University of California
Berkeley, California 94720, USA



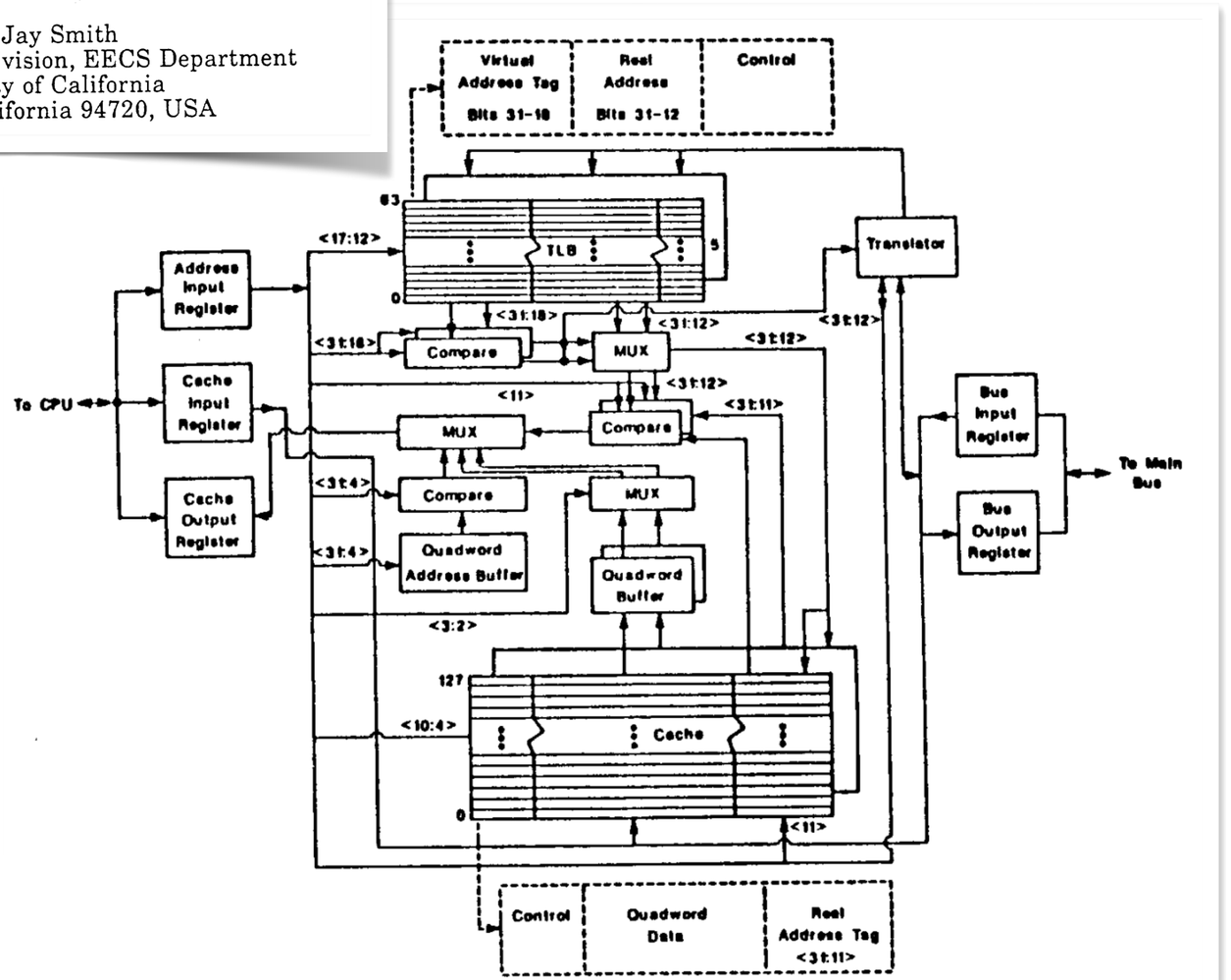


Image credit: https://www2.eecs.berkeley.edu/Pubs/TechRpts/1987/CSD-87-357.pdf

Outline

- Understanding cache misses
- Mitigating misses with associativity
- The promise and pitfalls of prefetching

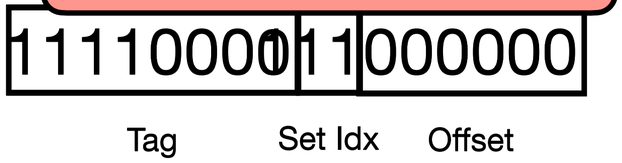
Understanding Cache Misses

- *Compulsory* misses occur on the first access to a block, so the data cannot yet be in the cache ➡️ these misses are (somewhat) unavoidable!
- *Capacity* misses occur if the cache cannot contain all of the blocked needed during the execution ➡️ these misses are (totally) unavoidable!
- *Conflict* misses are a subset of capacity misses in which a set within the cache cannot contain all of the blocks needed during the execution that can be mapped to that set ➡️ these misses are avoidable by cache construction!
- *Coherence* and *prefetching* misses are… we will come back to these next week!

Mitigating Conflict Misses

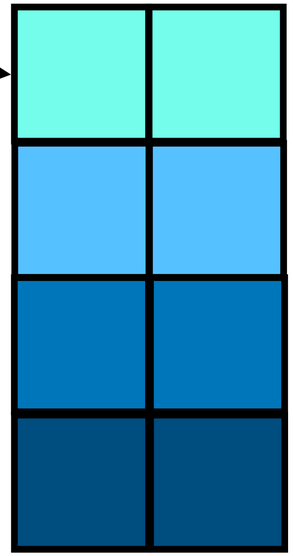
64 bytes per block

What happens on `0xf2c0`?



`0xf0c0`: Compulsory Miss

`0xf1c0`: Compulsory Miss



Set Idx 0

Set Idx 1

Set Idx 2

Set Idx 3

2-way set associative cache

Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers

Norman P. Jouppi
Digital Equipment Corporation Western Research Lab
100 Hamilton Ave., Palo Alto, CA 94301

3. Reducing Conflict Misses: Miss Caching and Victim Caching

Misses in caches can be classified into four categories: conflict, compulsory, capacity [7], and coherence. Conflict misses are misses that would not occur if the cache was fully-associative and had LRU replacement. Compulsory misses are misses required in any cache organization because they are the first references to an instruction or piece of data. Capacity misses occur when the cache size is not sufficient to hold data between references. Coherence misses are misses

Takeaway: we can avoid conflict misses by constructing our cache with a higher associativity!

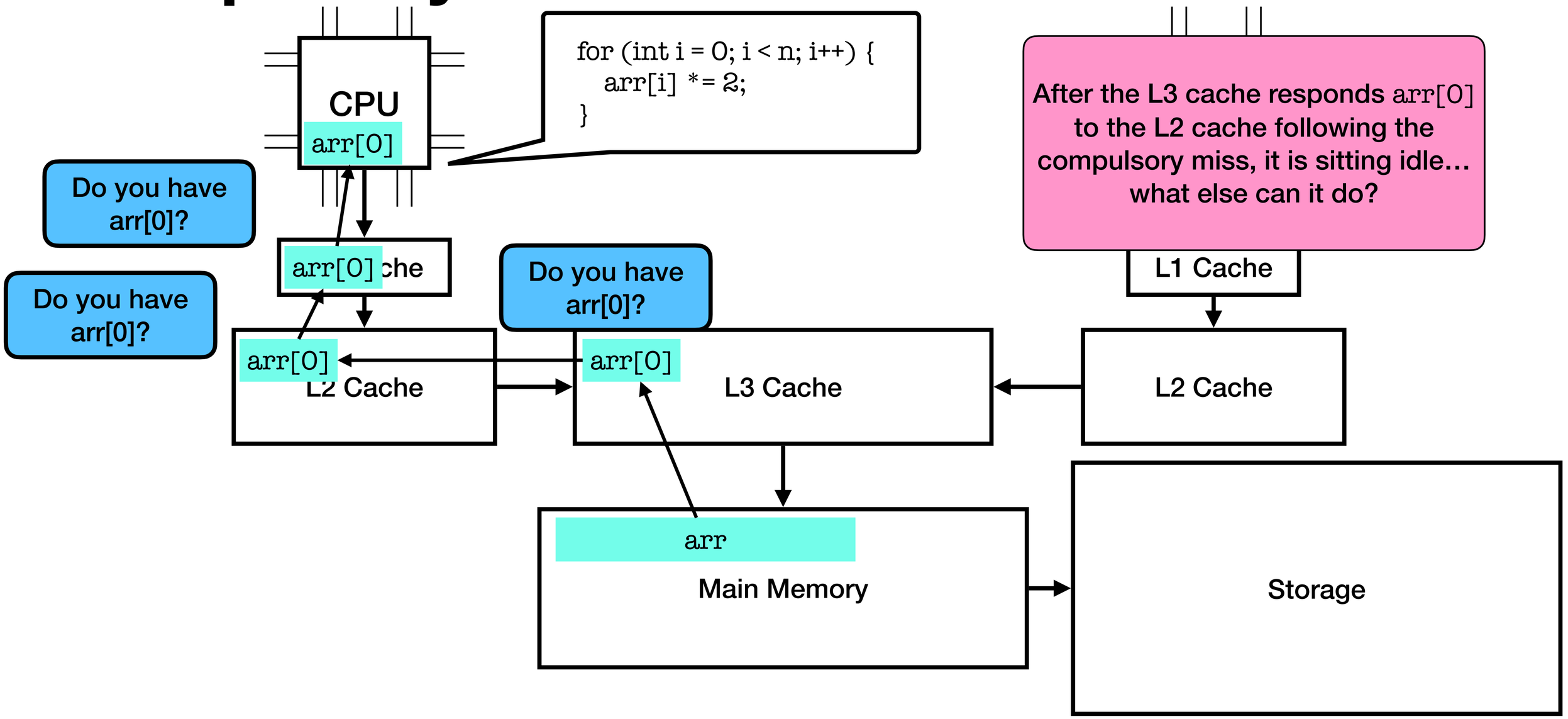
Chat with your neighbor(s)!

If we can mitigate conflict misses by increasing associativity, why aren't all caches fully associative? Is it more likely that an instruction cache or L3 cache is fully associative?

More "work" required to perform a lookup in a cache with a higher associativity

In caches closer to the processor state, we want lookups to be fast so these caches typically have lower associativity

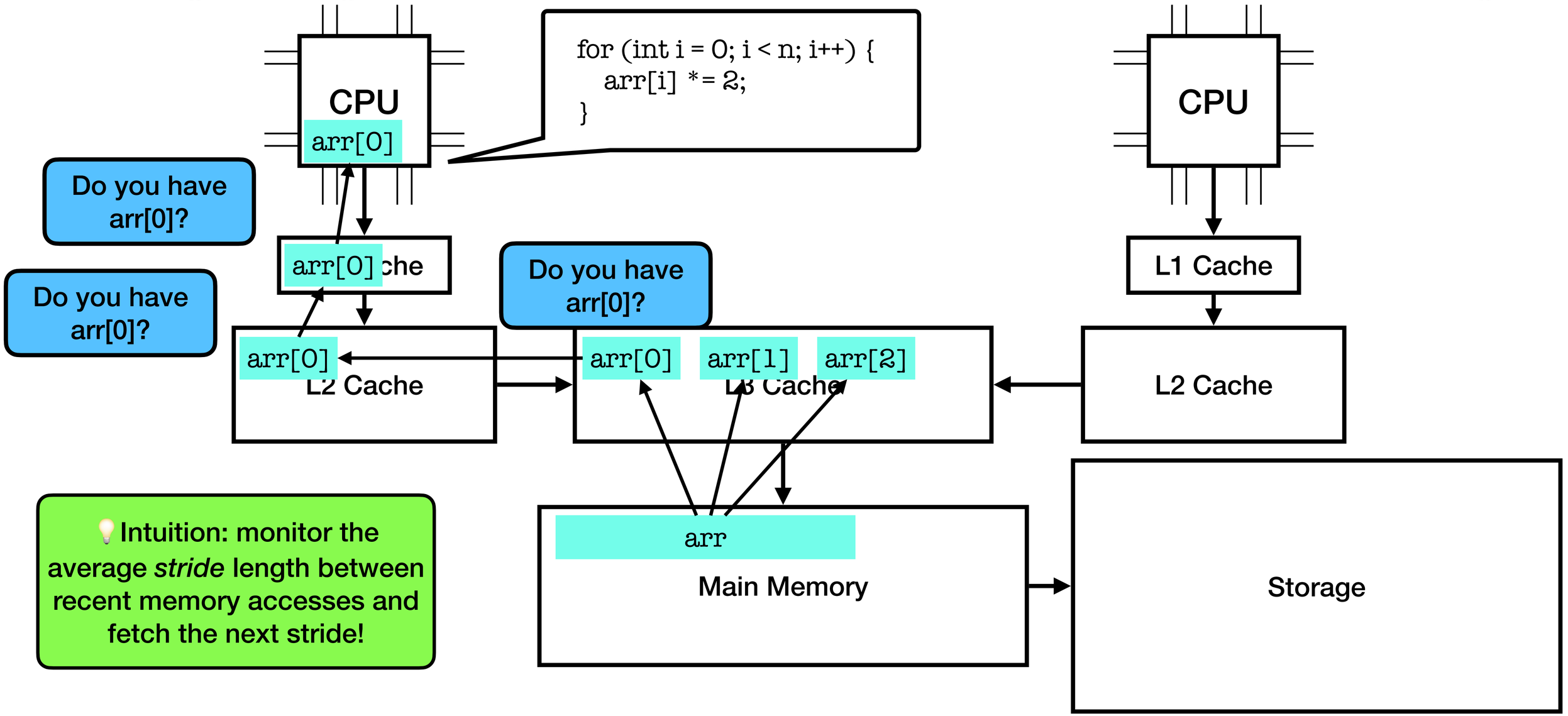
Compulsory Misses



Hardware Prefetching

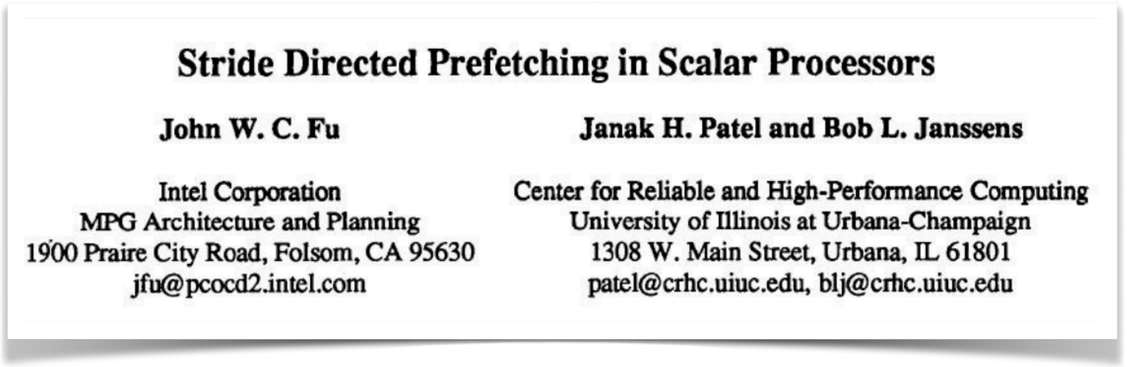
- When a cache is not performing a read or write, it spends cycles idle… this is inefficient!
- Instead, a cache controller can try to *predict* the next cache operations given the recent access pattern of addresses ➡️ the action of filling the cache with data that hasn't been accessed before is called *prefetching*
- When designing a prefetcher, a cache designer needs to determine: (a) Which address do we want to prefetch? (b) When are prefetches initiated? (c) Where should prefetched data be placed?
- We don't want our prefetching work to supersede the true processor requests!

Mitigating Compulsory Misses with Prefetching



Chat with your neighbor(s)!

If cache designs rely on applications exhibiting spatial locality, large cache block sizes eliminate compulsory misses within a block. Pitch an argument in favor of hardware prefetching over large cache blocks and vice versa!



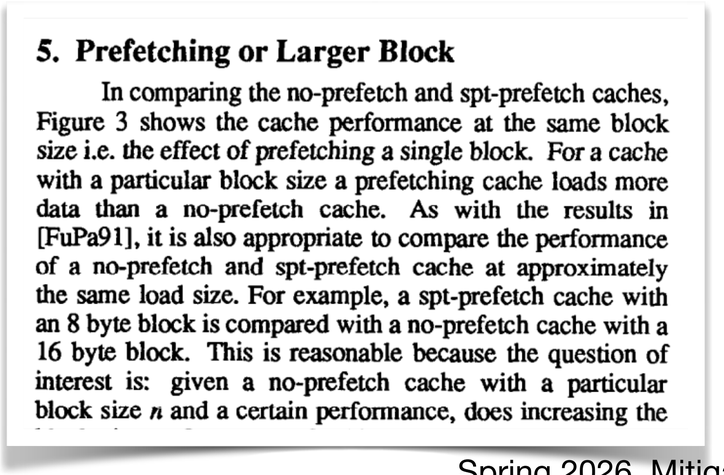
Stride Directed Prefetching in Scalar Processors

John W. C. Fu

Intel Corporation
MPG Architecture and Planning
1900 Praire City Road, Folsom, CA 95630
jfu@pcocd2.intel.com

Janak H. Patel and Bob L. Janssens

Center for Reliable and High-Performance Computing
University of Illinois at Urbana-Champaign
1308 W. Main Street, Urbana, IL 61801
patel@crhc.uiuc.edu, blj@crhc.uiuc.edu

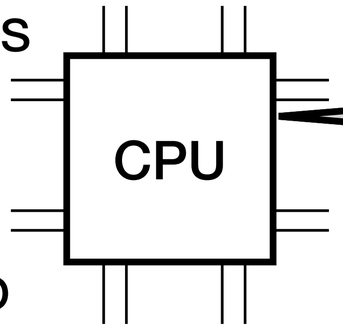


5. Prefetching or Larger Block

In comparing the no-prefetch and spt-prefetch caches, Figure 3 shows the cache performance at the same block size i.e. the effect of prefetching a single block. For a cache with a particular block size a prefetching cache loads more data than a no-prefetch cache. As with the results in [FuPa91], it is also appropriate to compare the performance of a no-prefetch and spt-prefetch cache at approximately the same load size. For example, a spt-prefetch cache with an 8 byte block is compared with a no-prefetch cache with a 16 byte block. This is reasonable because the question of interest is: given a no-prefetch cache with a particular block size *n* and a certain performance, does increasing the

Software Prefetching… the best of both?

- For irregular access patterns (e.g., applications that rely on heap addresses), hardware prefetching is largely ineffective…
- Fortunately, we know many of the addresses we are going to need by examining the binary at runtime!
- Developers can explicitly tell the program to asynchronously prefetch data to avoid misses ➡️ looks like memory operations with several “soon to execute” addresses to load




```
while (true) {
        __builtin_prefetch(node->next);
        __builtin_prefetch(node->prev);
        __builtin_prefetch(0xC5181CA);
       // super slow operation!
       if (x) {
              node = node->next;
       } else if (y) {
              node = node->prev;
       }
              node = 0xC5181CA;
       }
}
```

Requires new instructions in the instruction set for your processor to interpret!

Takeaways

- By understanding the types of cache misses that can occur, we can develop strategies to mitigate them as frequently as possible in the construction of our cache logic
- Strategies like higher associativity and larger cache blocks can help, but also suffer trade-offs in complexity/energy use and fragmentation
- Prefetching allows caches to predict what to do next when they would otherwise be idle, but if done poorly can pollute the cache with unnecessary data