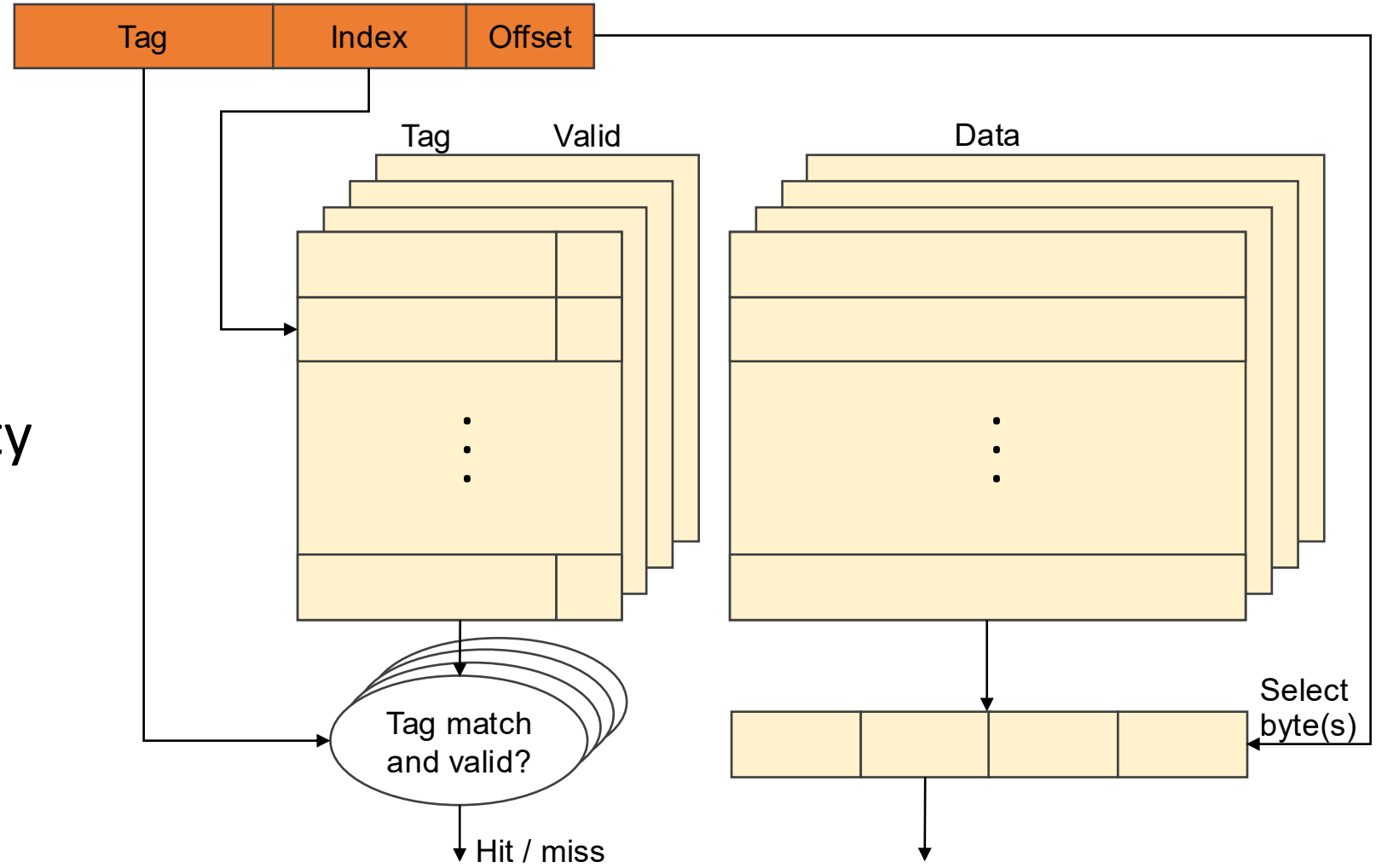
Prefetching

Advanced Topics in Computer Architecture

Timothy Jones

Caching

- We’re all familiar with caching
- Caches store data close to the core
- Caches take advantage of locality
  - Spatial locality
  - Temporal locality



Cache performance

- Cache hit and miss rates give an indication of cache performance
  - But they fail to capture the impact of the cache on the overall system
- From the CPU's point of view, we want to reduce the average memory access time (AMAT)
  - The average time it takes to load data, including the time to access the cache and the time taken to service a miss
  - Including a cache should reduce AMAT, otherwise it is doing more harm than good!

AMAT = Cache hit time + Cache miss rate * Cache miss penalty

Improving cache performance

AMAT = Cache hit time + Cache miss rate * Cache miss penalty

- The cache hit time depends on size, associativity, pipelining
- The cache miss penalty depends on where else the data is
- We're going to focus on improving the cache miss rate
  - We can classify cache misses into four categories
  - Compulsory, capacity, conflict & coherence
- Prefetching primarily reduces compulsory and capacity misses

Prefetching

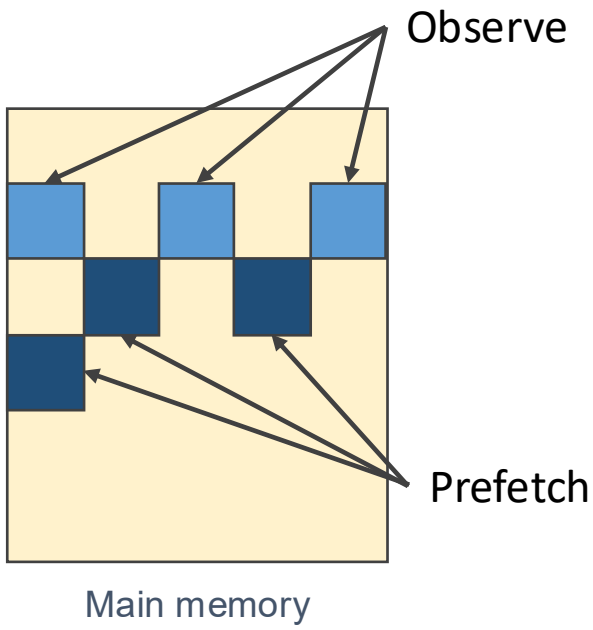
- This is a technique to bring data into the cache before it is needed
- The idea is to make a prediction about what data the program will use in the near future
  - Then load that data into the cache so that it arrives *before* required
- Prefetching can be performed in hardware or software
  - Processors often provide special instructions to do this in software
- We're going to look at a variety of hardware, software and thread-based techniques

Metrics for prefetcher performance

- Accuracy: fraction of prefetches that are used before being evicted
  - Useless prefetches waste memory bandwidth and can evict useful data (cache pollution)
- Coverage: fraction of the original misses that prefetching removes
- Timeliness: whether prefetched data arrives at the right time
  - Too late and the miss latency is only partly hidden
  - Too early and the data may be evicted before it is used
- These trade off against each other
  - Prefetching more aggressively usually raises coverage but lowers accuracy

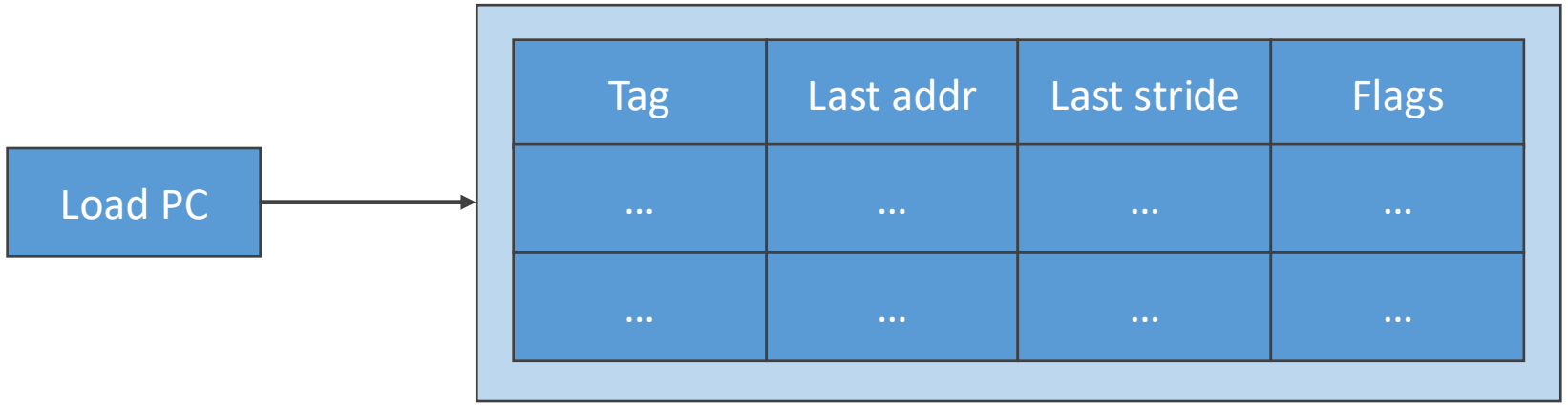
Simple prefetchers

- Next-line is a simple prefetcher
  - Does what it says on the tin!
- Stride prefetchers are also relatively simple
- The prefetcher identifies simple patterns in the accesses made
  - E.g. 0x1000, 0x1100, 0x1200
- It learns this stride and prefetches based on it



Stride prefetcher

- Stride prefetchers are great for accessing dense matrices / arrays
  - They usually provide little help for pointers
- Although simple, there are complexities
  - For example, how to distinguish between multiple interleaved streams
  - What prefetch degree (# blocks) or depth (distance ahead) to use

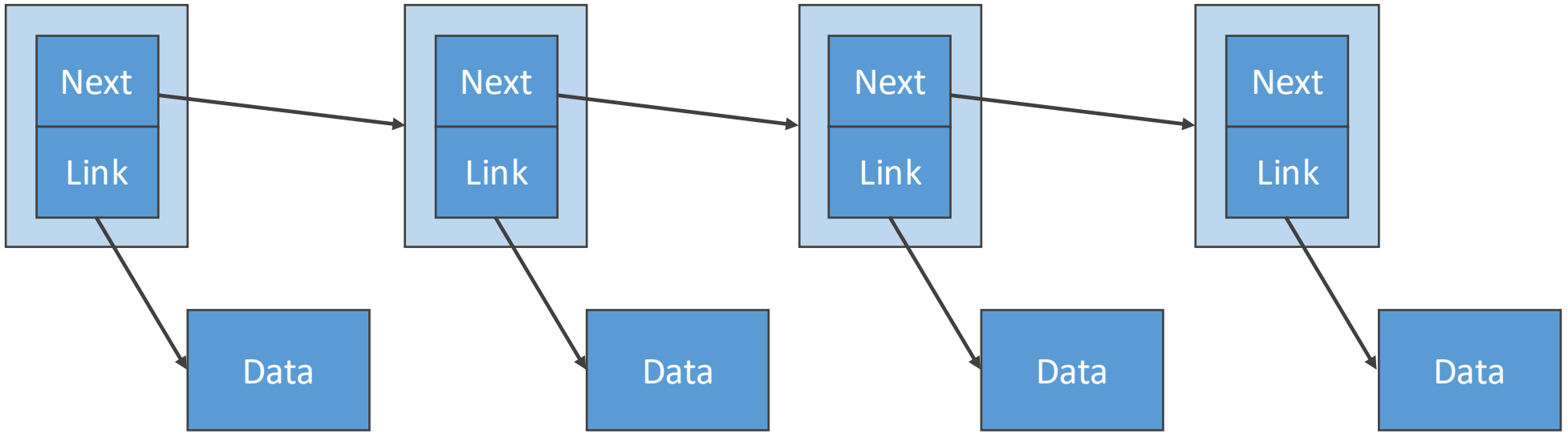


More complex prefetching

- Stride prefetchers are effective for a lot of workloads
  - Think array traversals
- But they can’t pick up more complex patterns
- In particular, two types of access pattern are problematic
  - Those based on pointer chasing
  - Those that are dependent on the value of the data
- More complex prefetchers are required for these

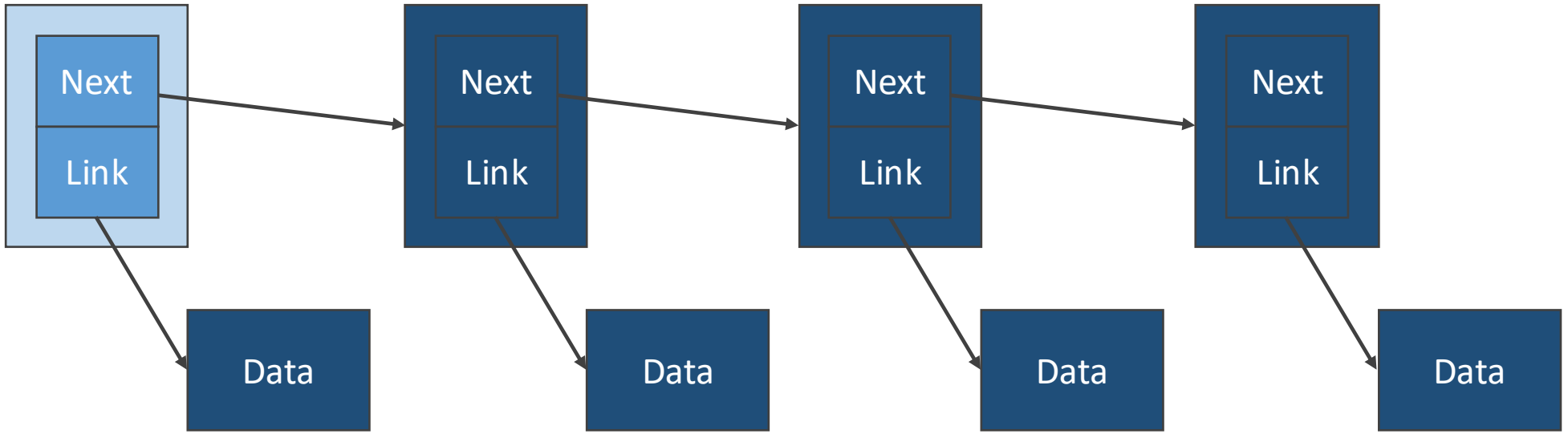
Correlation prefetching

- Irregular access patterns are not picked up well by stride prefetchers
- Correlation prefetching means correlating pairs / groups of addresses
  - When you see the first address, you can prefetch the next address



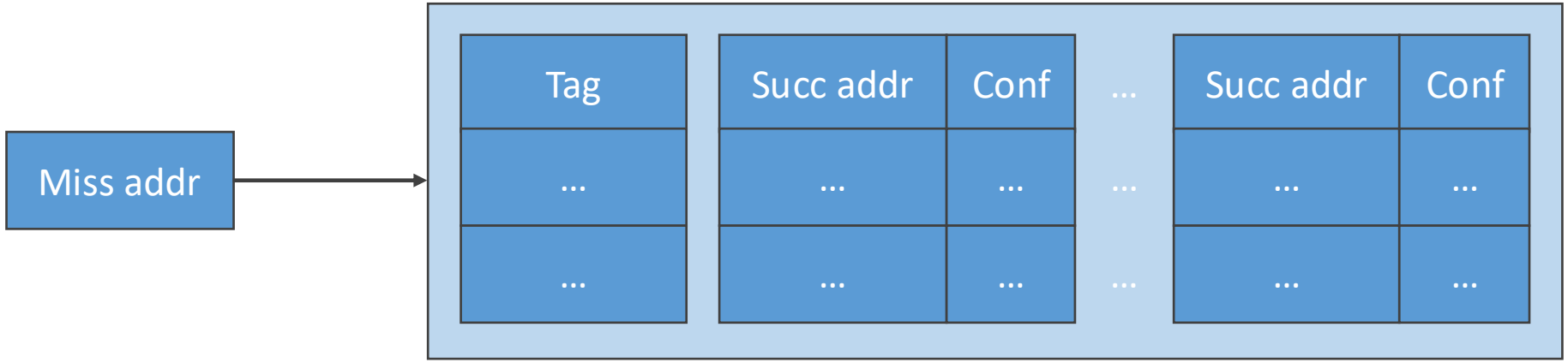
Correlation prefetching

- Irregular access patterns are not picked up well by stride prefetchers
- Correlation prefetching means correlating pairs / groups of addresses
  - When you see the first address, you can prefetch the next address



Correlation prefetching

- These work by exploiting temporal correlation
  - Essentially a hardware lookup table mapping one address to another
- They have their downsides too though
  - Must have seen the accesses before and require enormous state
- The Markov prefetcher is the simplest design

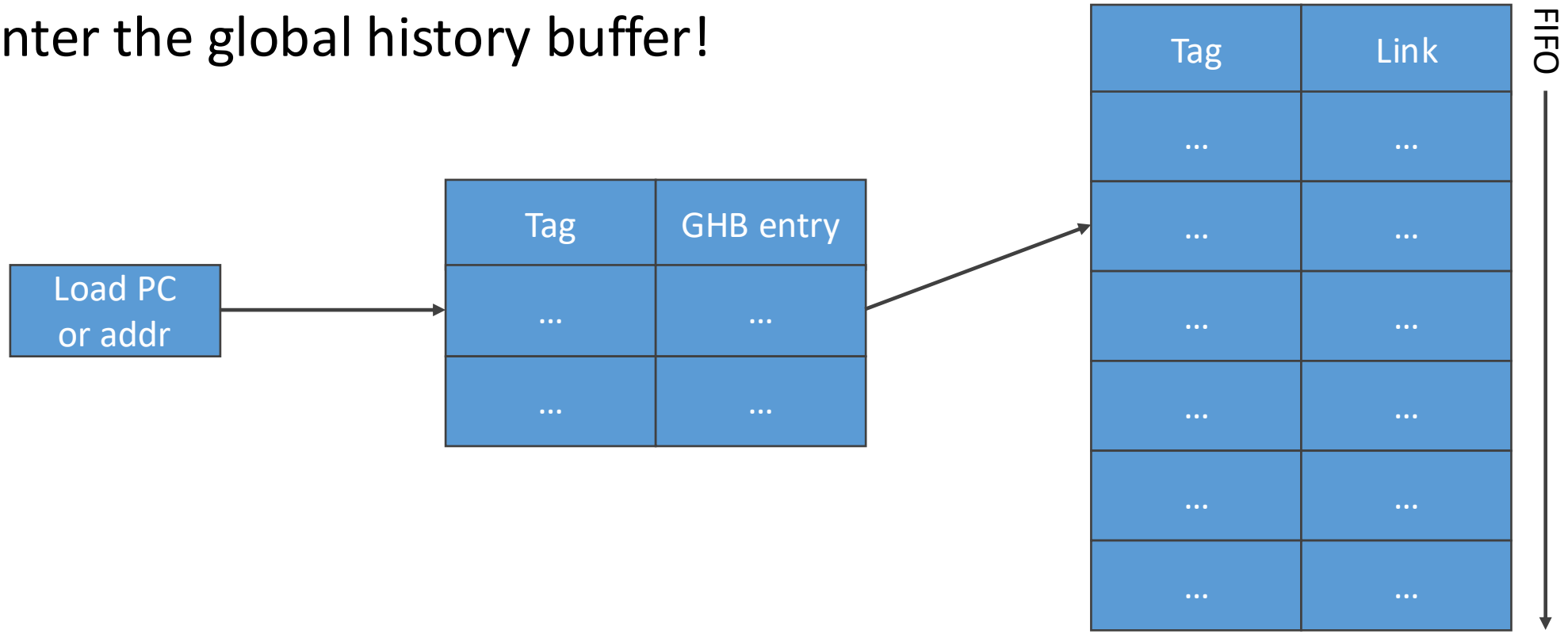


Correlation prefetching

- Markov prefetcher prefetches all successor addresses
  - LRU to manage the table entries
  - Limited depth and coverage
- Depth can be addressed by associating a stream with each load PC
  - This has its own downsides too, of course
- Depth can also be addressed by dead-block prediction
  - Based on code correlation or time keeping
- Coverage can be addressed by increasing storage
  - Perhaps by moving the correlation table into main memory

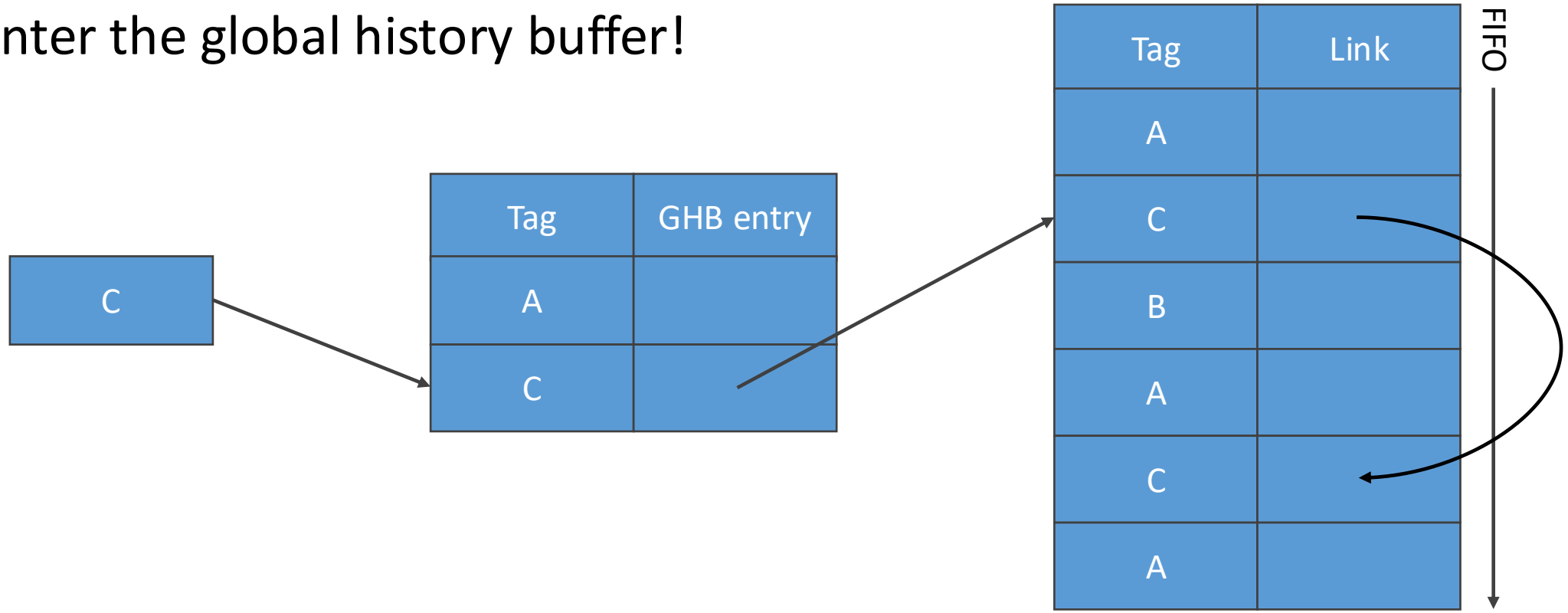
Correlation prefetching

- The Markov predictor structure limits it to a fixed number of successors
- Enter the global history buffer!



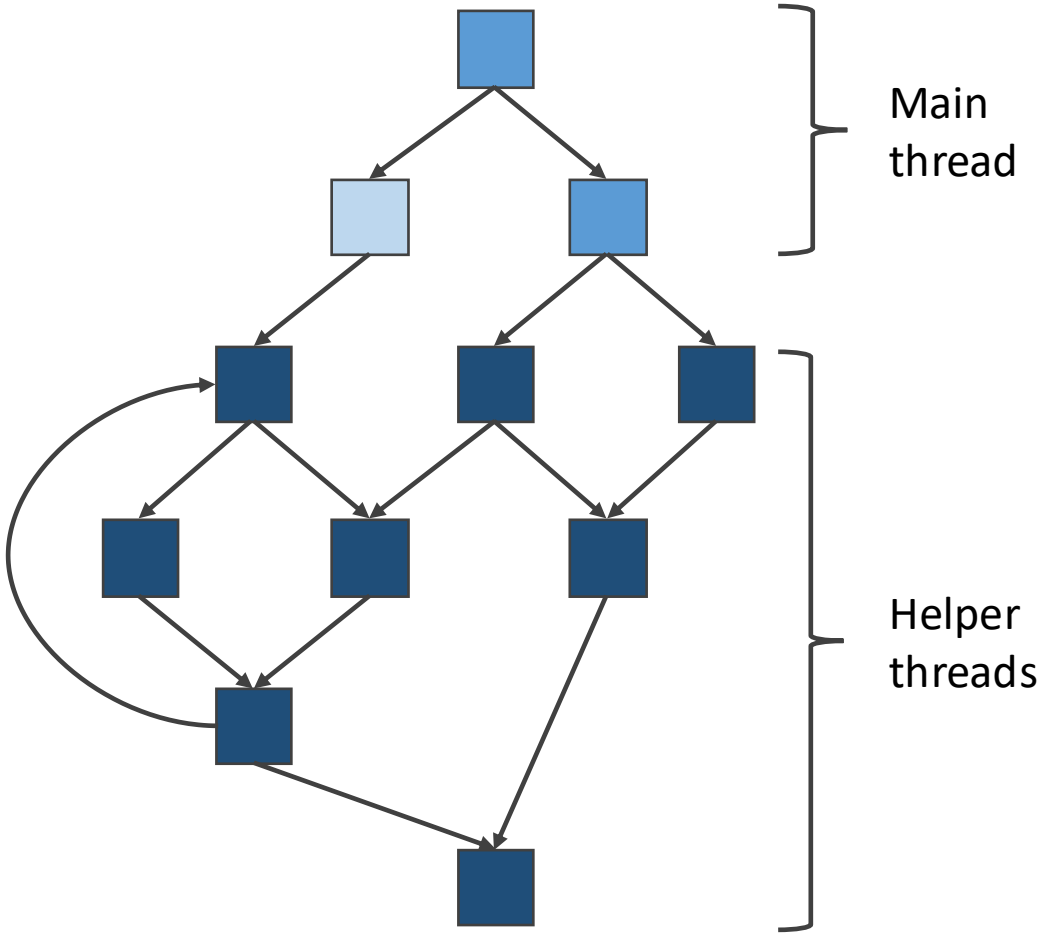
Correlation prefetching

- The Markov predictor structure limits it to a fixed number of successors
- Enter the global history buffer!



Helper-thread-based prefetching

- Instead of dedicated hardware, we could use spare execution resources
  - E.g. a different SMT context
- Using a helper thread we can explore the future control-flow graph
- Executes a cut-down version of the main thread's code



Software prefetching

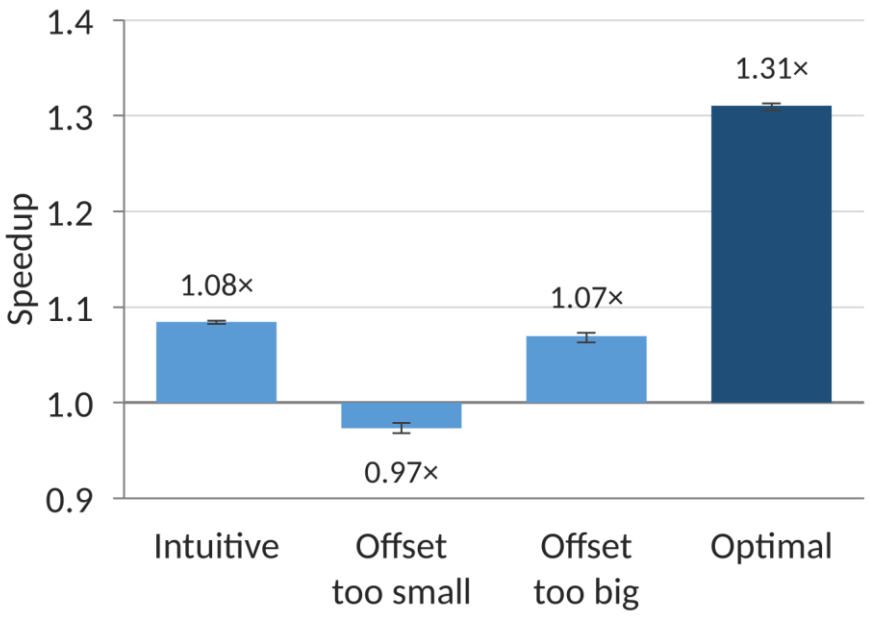
- Sometimes the programmer is better placed to help
  - Should know what's accessed next
- The architecture provides prefetching hint primitives
- When seen, the processor decides whether to prefetch
- This is the value-dependent pattern from earlier

```
for (i=0; i<NUM; i++) {
  A[B[i]]++;
}
```



```
for (i=0; i<NUM; i++) {
  SWPF(B[i + offset*2]);
  SWPF(A[B[i + offset]]);
  A[B[i]]++;
}
```

Software prefetching




```
for (i=0; i<NUM; i++) {
  A[B[i]]++;
}
```



```
for (i=0; i<NUM; i++) {
  SWPF(B[i + offset*2]);
  SWPF(A[B[i + offset]]);
  A[B[i]]++;
}
```

Summary

Approach	Predicts from	Good for	Costs and limits
Next-line	The next block in memory	Sequential access	Trivial hardware but wasted bandwidth when access isn't sequential
Stride	A constant stride per load PC	Dense arrays and regular loops	Small table that's little help for pointers or irregular access
Correlation (Markov, GHB)	Recorded history of miss addresses	Irregular patterns that repeat, e.g. pointer chasing	Large state and it must have seen the pattern before
Helper thread	Running a cut-down copy of the code ahead	Complex, hard-to-predict access	Needs a spare thread context and this helper thread must stay ahead of main
Software	Programmer or compiler knowledge	Value-dependent access, like A[B[i]]	Extra instructions and the prefetch distance might be hard to tune