

CSE 502:

Computer Architecture

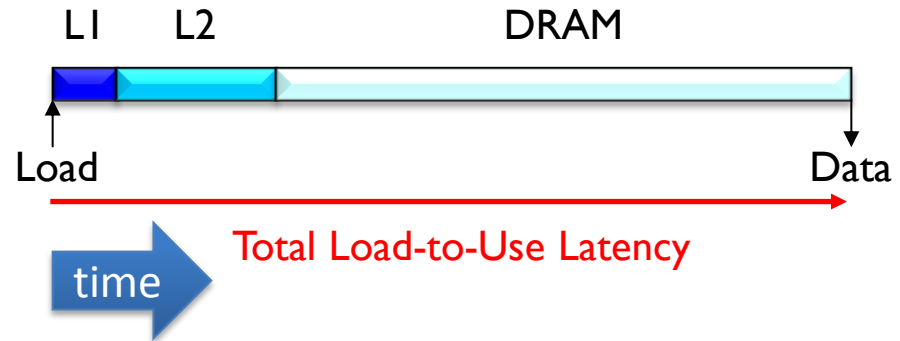
Prefetching

Prefetching (1/3)

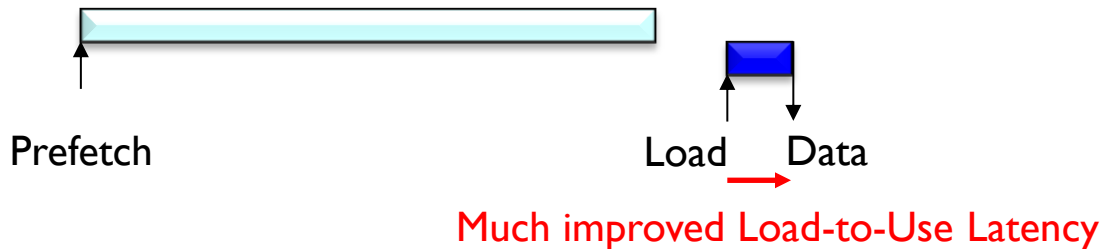
- Fetch block ahead of demand
- Target compulsory, capacity, (& coherence) misses
 - Why not conflict?
- Big challenges:
 - Knowing “what” to fetch
 - Fetching useless blocks wastes resources
 - Knowing “when” to fetch
 - Too early → clutters storage (or gets thrown out before use)
 - Fetching too late → defeats purpose of “pre”-fetching

Prefetching (2/3)

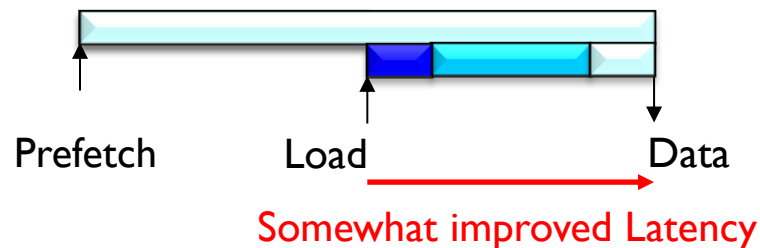
- Without prefetching:



- With prefetching:



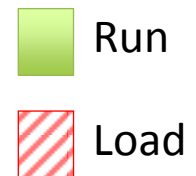
- Or:



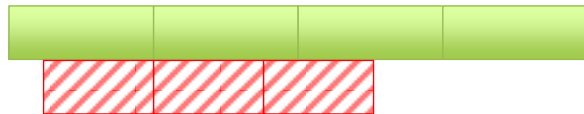
Prefetching must be accurate and timely

Prefetching (3/3)

- Without prefetching:



- With prefetching:



Prefetching removes loads from critical path

Common “Types” of Prefetching

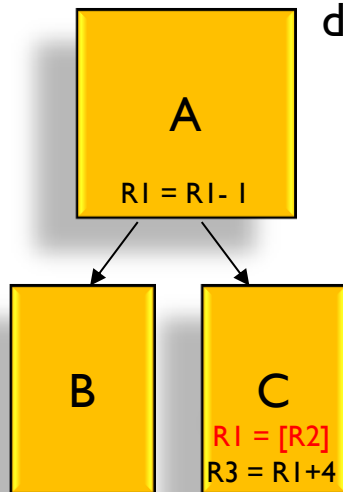
- Software
- Next-Line, Adjacent-Line
- Next-N-Line
- Stream Buffers
- Stride
- “Localized” (e.g., PC-based)
- Pointer
- Correlation

Software Prefetching (1/4)

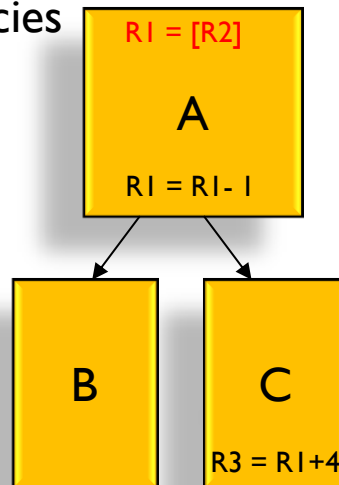
- Compiler/programmer places prefetch instructions
- Put prefetched value into...
 - Register (binding, also called “hoisting”)
 - May prevent instructions from committing
 - Cache (non-binding)
 - Requires ISA support
 - May get evicted from cache before demand

Software Prefetching (2/4)

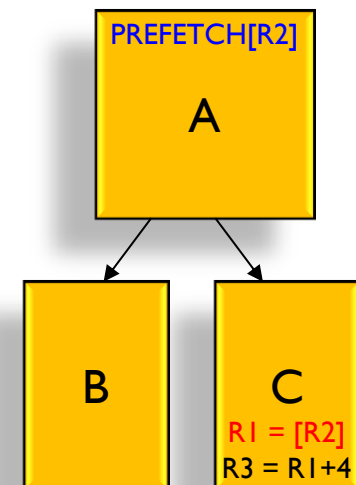
Hoisting must
be aware of
dependencies



(Cache misses in red)



Hopefully the load miss
is serviced by the time
we get to the consumer



Using a prefetch instruction
can avoid problems with
data dependencies

Software Prefetching (3/4)

```
for (I = 1; I < rows; I++)  
{  
    for (J = 1; J < columns; J++)  
    {  
        prefetch(&x[I+1,J]);  
        sum = sum + x[I,J];  
    }  
}
```


Software Prefetching (4/4)

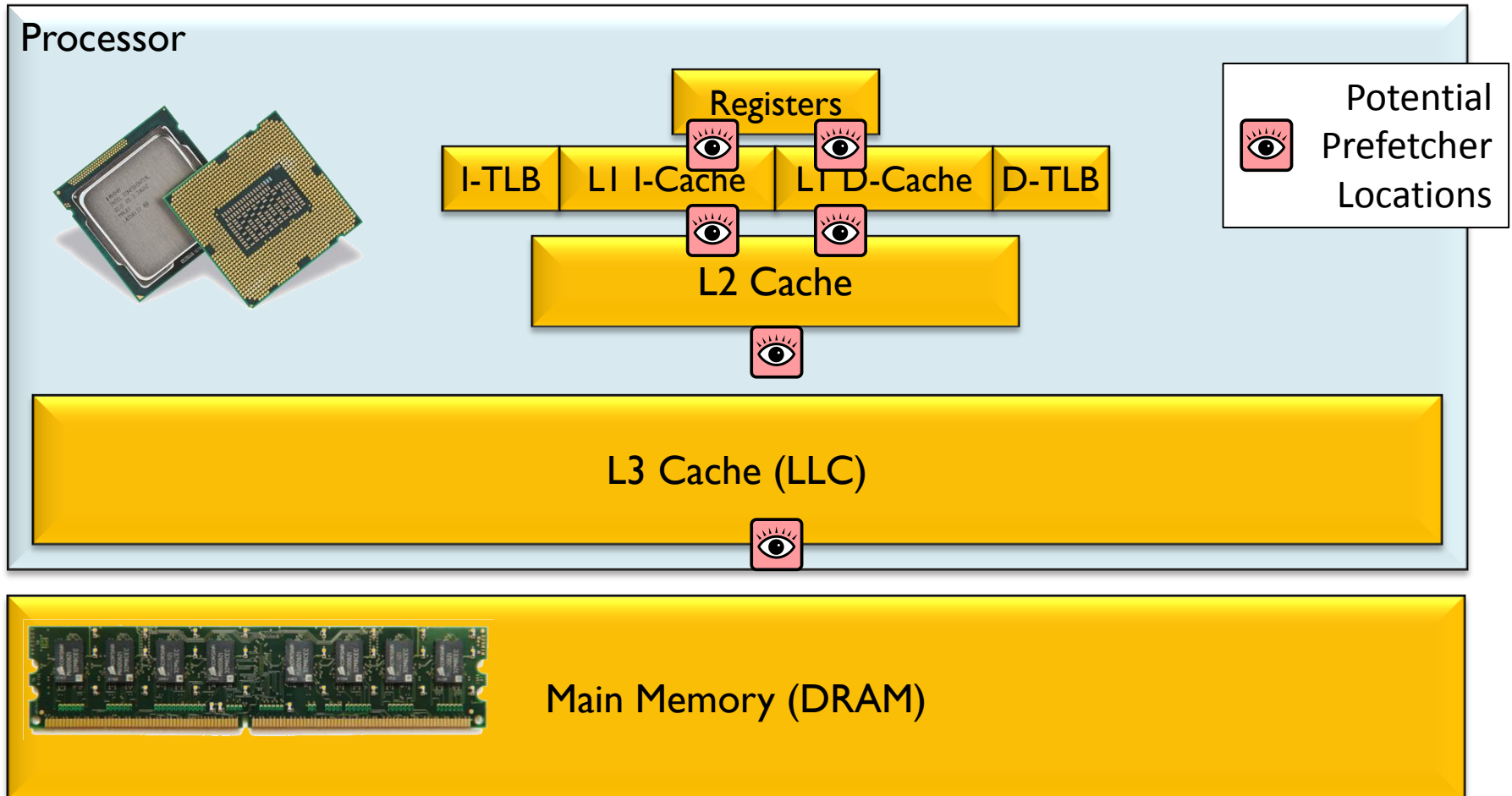
- Pros:
 - Gives programmer control and flexibility
 - Allows time for complex (compiler) analysis
 - No (major) hardware modifications needed
- Cons:
 - Hard to perform timely prefetches
 - At IPC=2 and 100-cycle memory \rightarrow move load 200 inst. earlier
 - Might not even have 200 inst. in current function
 - Prefetching earlier and more often leads to low accuracy
 - Program may go down a different path
 - Prefetch instructions increase code footprint
 - May cause more I\$ misses, code alignment issues

Hardware Prefetching (1/3)

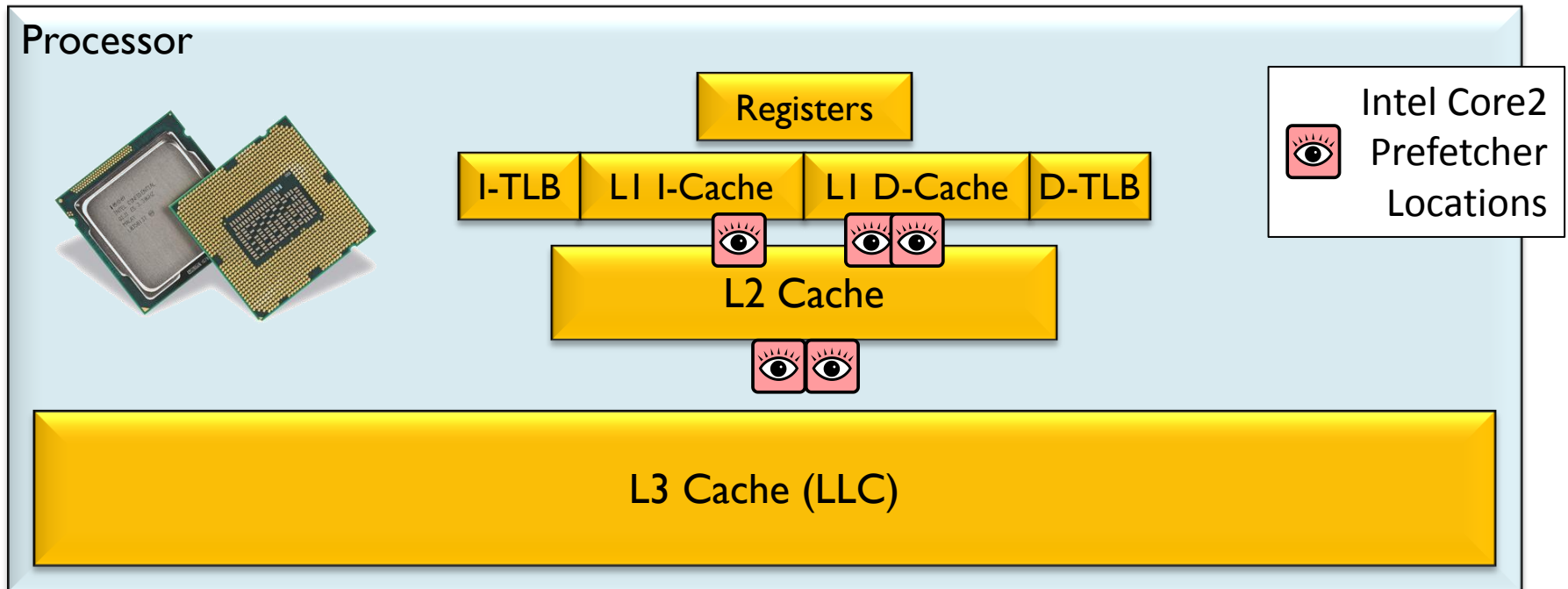
- Hardware monitors memory accesses
 - Looks for common patterns
- Guessed addresses are placed into *prefetch queue*
 - Queue is checked when no demand accesses waiting
- Prefetchers look like READ requests to the hierarchy
 - Although may get special “prefetched” flag in the state bits
- Prefetchers trade bandwidth for latency
 - Extra bandwidth used **only** when guessing incorrectly
 - Latency reduced **only** when guessing correctly

No need to change software

Hardware Prefetching (2/3)



Hardware Prefetching (3/3)



- Real CPUs have multiple prefetchers
 - Usually closer to the core (easier to detect patterns)
 - Prefetching at LLC is hard (cache is banked and hashed)

Next-Line (or Adjacent-Line) Prefetching

- On request for line X , prefetch $X+1$ (or $X \oplus 1$)
 - Assumes spatial locality
 - Often a good assumption
 - Should stop at physical (OS) page boundaries
- Can often be done efficiently
 - Adjacent-line is convenient when next-level block is bigger
 - Prefetch from DRAM can use bursts and row-buffer hits
- Works for I\$ and D\$
 - Instructions execute sequentially
 - Large data structures often span multiple blocks

Simple, but usually not timely

Next-N-Line Prefetching

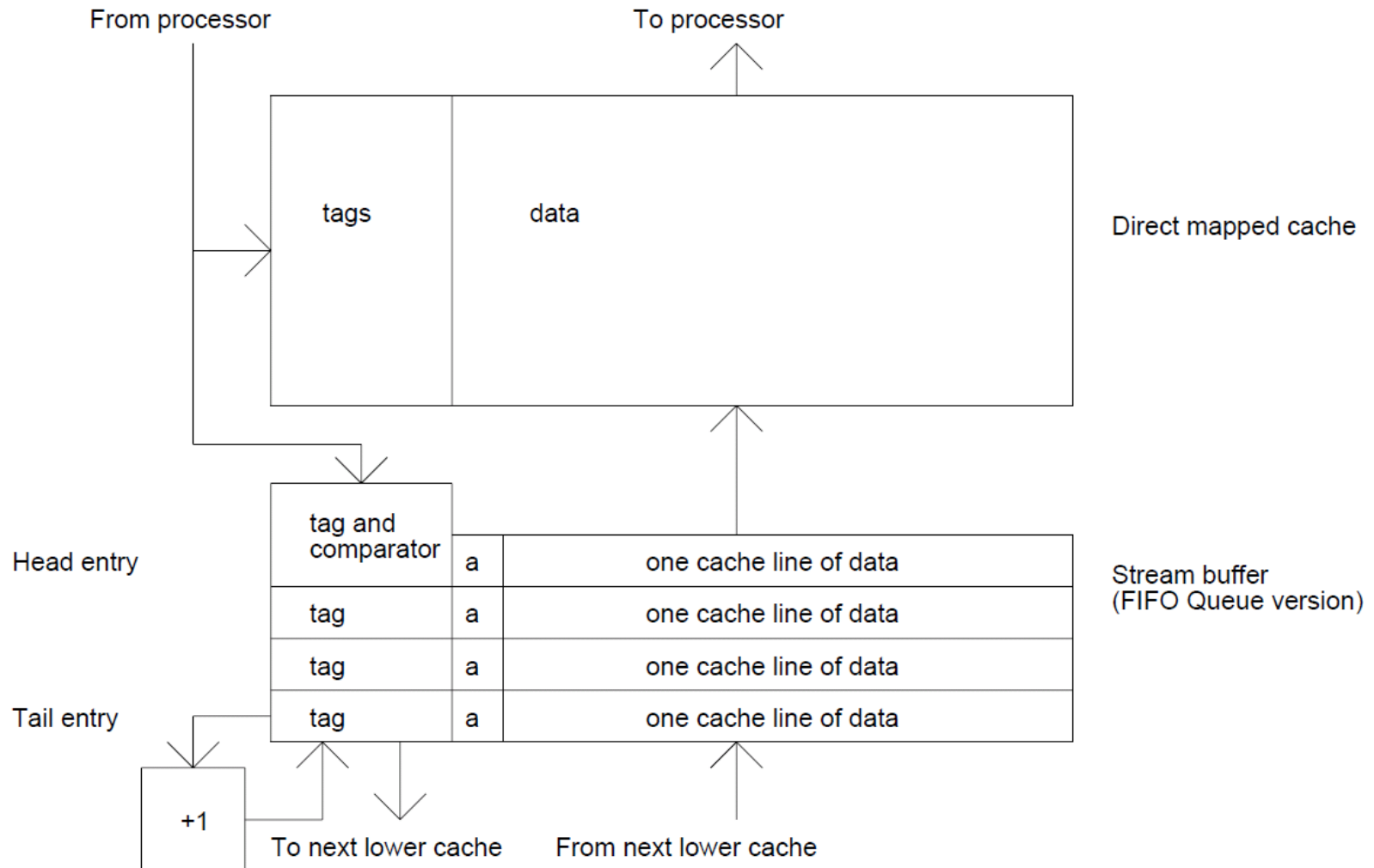
- On request for line X , prefetch $X+1, X+2, \dots, X+N$
 - N is called “prefetch depth” or “prefetch degree”
- Must carefully tune depth N . Large N is ...
 - More likely to be useful (correct and timely)
 - More aggressive \rightarrow more likely to make a mistake
 - Might evict something useful
 - More expensive \rightarrow need storage for prefetched lines
 - Might delay useful request on interconnect or port

Still simple, but more timely than Next-Line

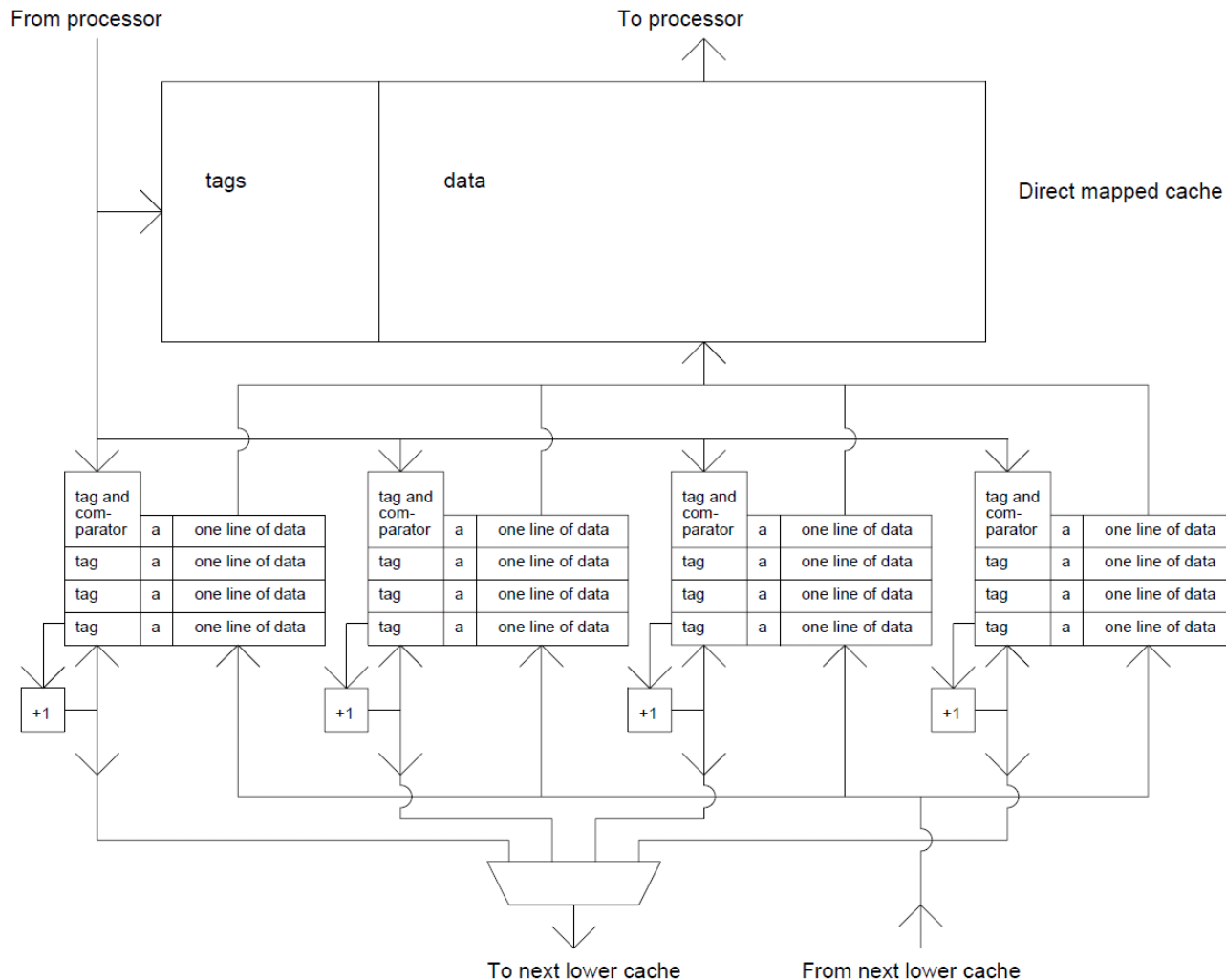
Stream Buffers (1/3)

- What if we have multiple inter-twined streams?
 - A, B, A+1, B+1, A+2, B+2, ...
- Can use multiple stream buffers to track streams
 - Keep next-N available in buffer
 - On request for line X, shift buffer and fetch X+N+1 into it
- Can extend to “quasi-sequential” stream buffer
 - On request Y in [X...X+N], advance by Y-X+1
 - Allows buffer to work when items are skipped
 - Requires expensive (associative) comparison

Stream Buffers (2/3)

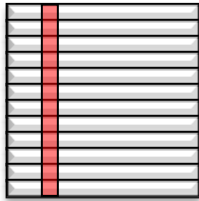


Stream Buffers (3/3)



Can support multiple streams in parallel

Stride Prefetching (1/2)



Column in matrix

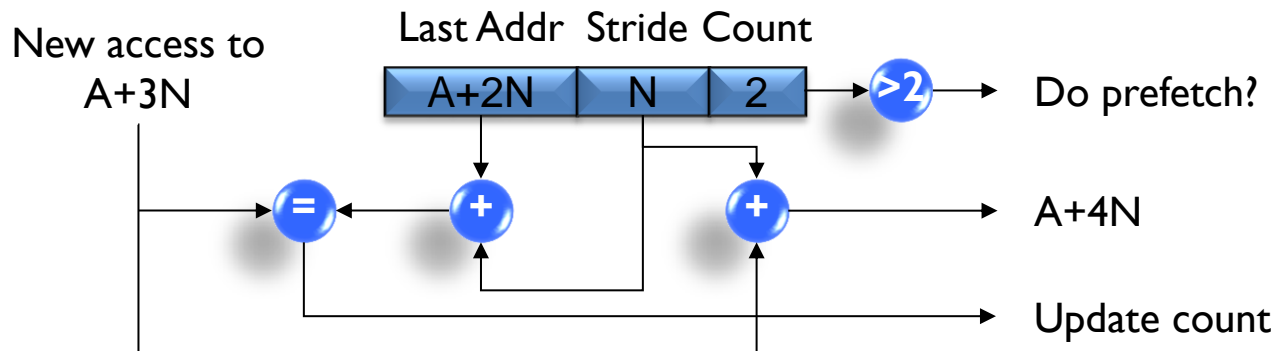


Elements in array of `structs`

- Access patterns often follow a stride
 - Accessing column of elements in a matrix
 - Accessing elements in array of `structs`
- Detect stride S , prefetch depth N
 - Prefetch $X+1 \cdot S$, $X+2 \cdot S$, ..., $X+N \cdot S$

Stride Prefetching (2/2)

- Must carefully select depth N
 - Same constraints as Next-N-Line prefetcher
- How to determine if $A[i] \rightarrow A[i+1]$ or $X \rightarrow Y$?
 - Wait until $A[i+2]$ (or more)
 - Can vary prefetch depth based on confidence
 - More consecutive strided accesses \rightarrow higher confidence

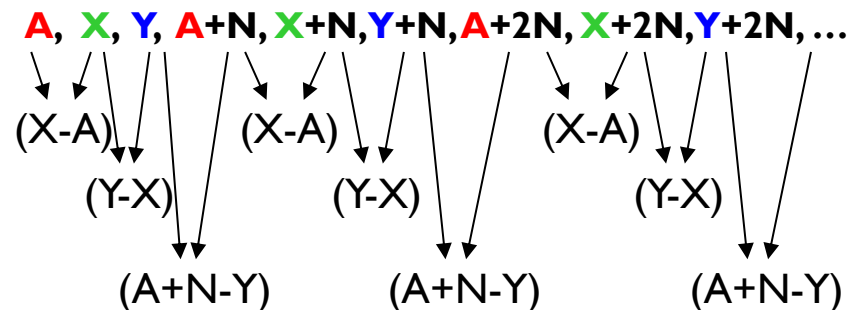


“Localized” Stride Prefetchers (1/2)

- What if multiple strides are interleaved?
 - No clearly-discernible stride
 - Could do multiple strides like stream buffers
 - Expensive (must detect/compare many strides on each access)
 - Accesses to structures usually localized to an instruction



Miss pattern looks like:



Load R1 = [R2]

Load R3 = [R4]

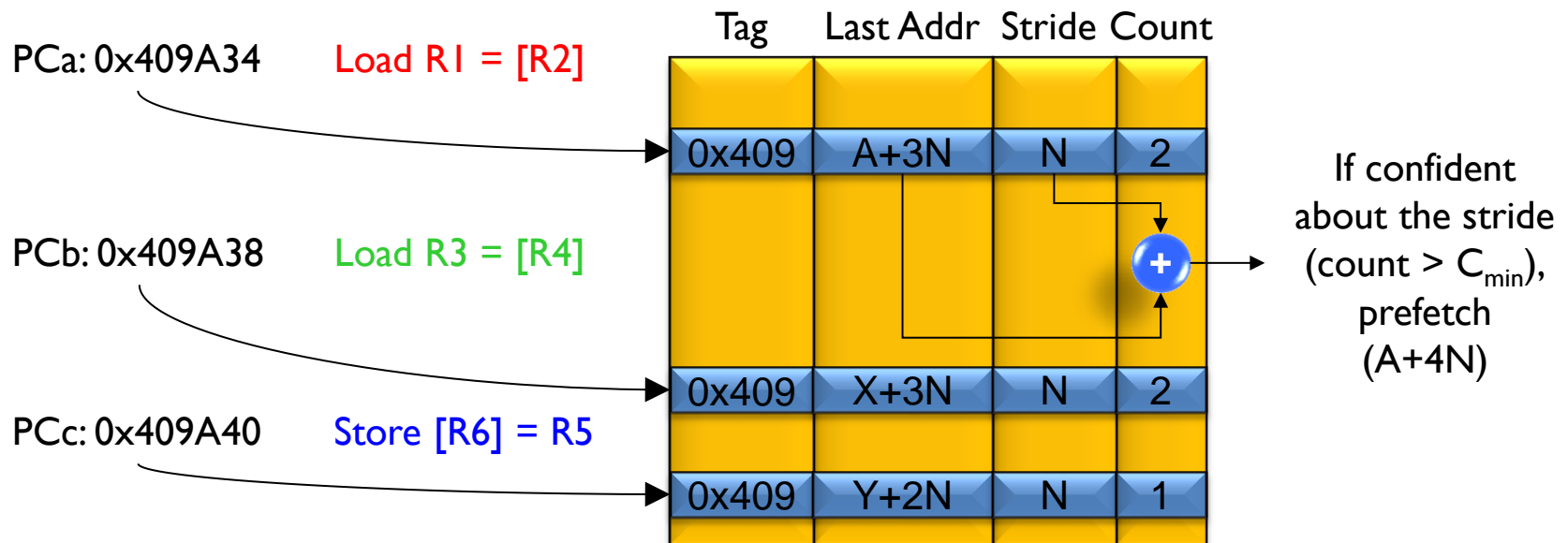
Add R5, R1, R3

Store [R6] = R5

Use an array of strides, indexed by PC

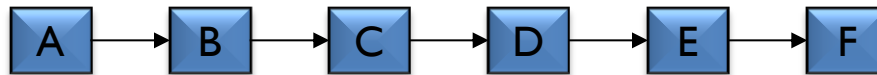
“Localized” Stride Prefetchers (2/2)

- Store PC, last address, last stride, and count in RPT
- On access, check RPT (Reference Prediction Table)
 - Same stride? \rightarrow count++ if yes, count-- or count=0 if no
 - If count is high, prefetch (last address + stride*N)

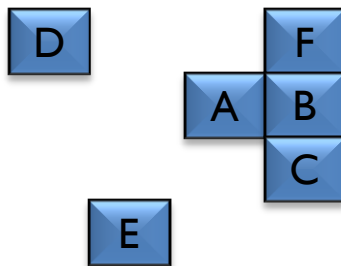


Other Patterns

- Sometimes accesses are regular, but no strides
 - Linked data structures (e.g., lists or trees)



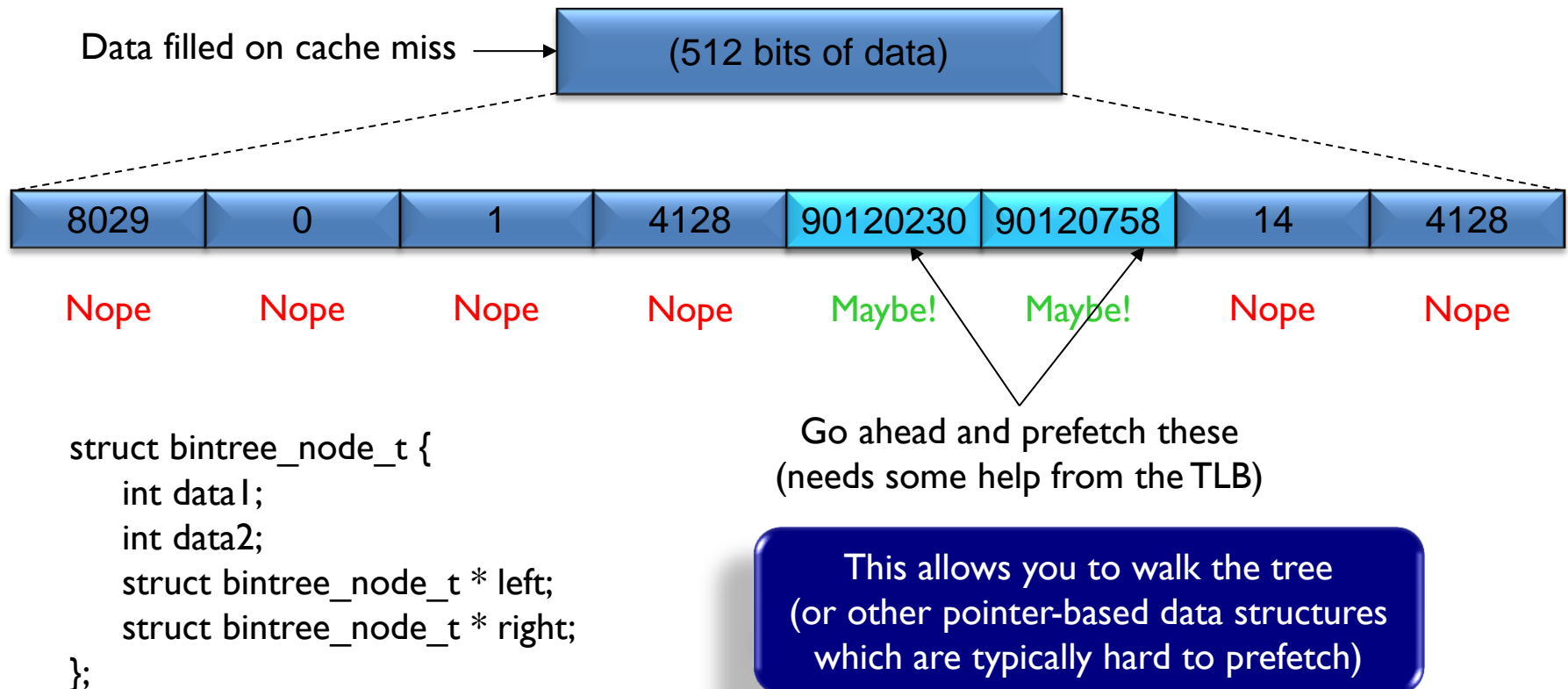
Linked-list traversal



Actual memory
layout

(no chance to detect a stride)

Pointer Prefetching (1/2)

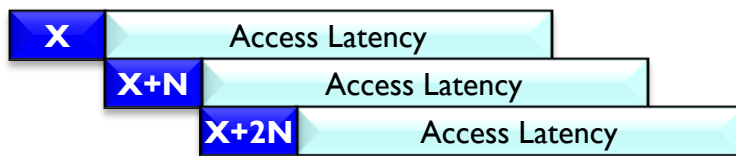


Pointers usually “look different”

Pointer Prefetching (2/2)

- Relatively cheap to implement
 - Don't need extra hardware to store patterns
- Limited lookahead makes timely prefetches hard
 - Can't get next pointer until fetched data block

Stride Prefetcher:



Pointer Prefetcher:



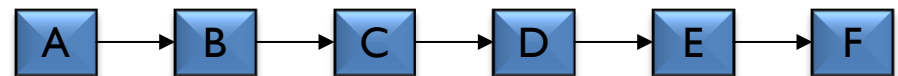
Pair-wise Temporal Correlation (1/2)

- Accesses exhibit temporal correlation
 - If E followed D in the past → if we see D, prefetch E

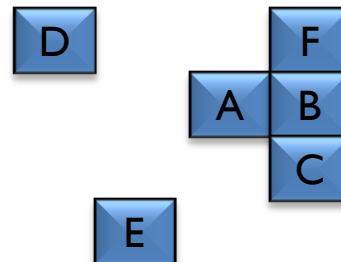
Correlation Table

D	D	E	10
F	F	?	00
A	A	B	11
B	B	C	11
C	C	D	11
E	E	F	01

Linked-list traversal



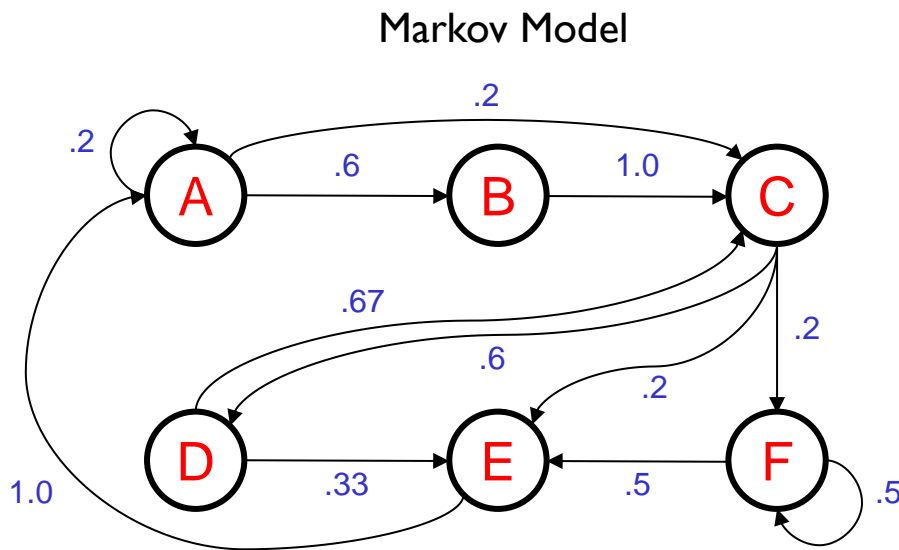
Actual memory layout



Can use recursively to get more lookahead 😊

Pair-wise Temporal Correlation (2/2)

- Many patterns more complex than linked lists
 - Can be represented by a Markov Model
 - Required tracking **multiple** potential successors
- Number of candidates is called breadth



Correlation Table

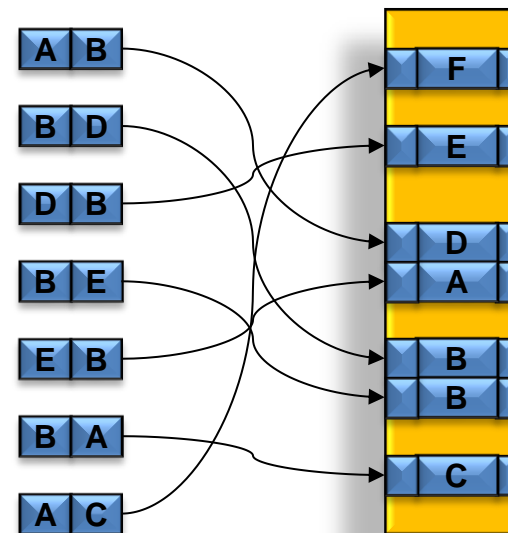
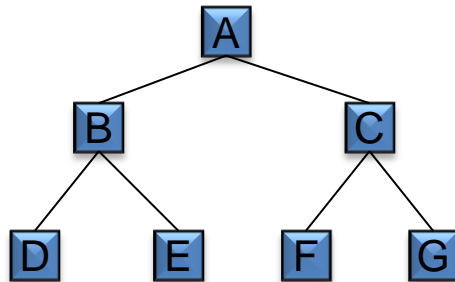
D	D	C	11	E	01
F	F	E	11	?	00
A	A	B	11	C	01
B	B	C	11	?	00
C	C	D	11	F	10
E	E	A	11	?	00

Recursive breadth & depth grows exponentially ☹️

Increasing Correlation History Length

- Longer history enables more complex patterns
 - Use history hash for lookup
 - Increases training time

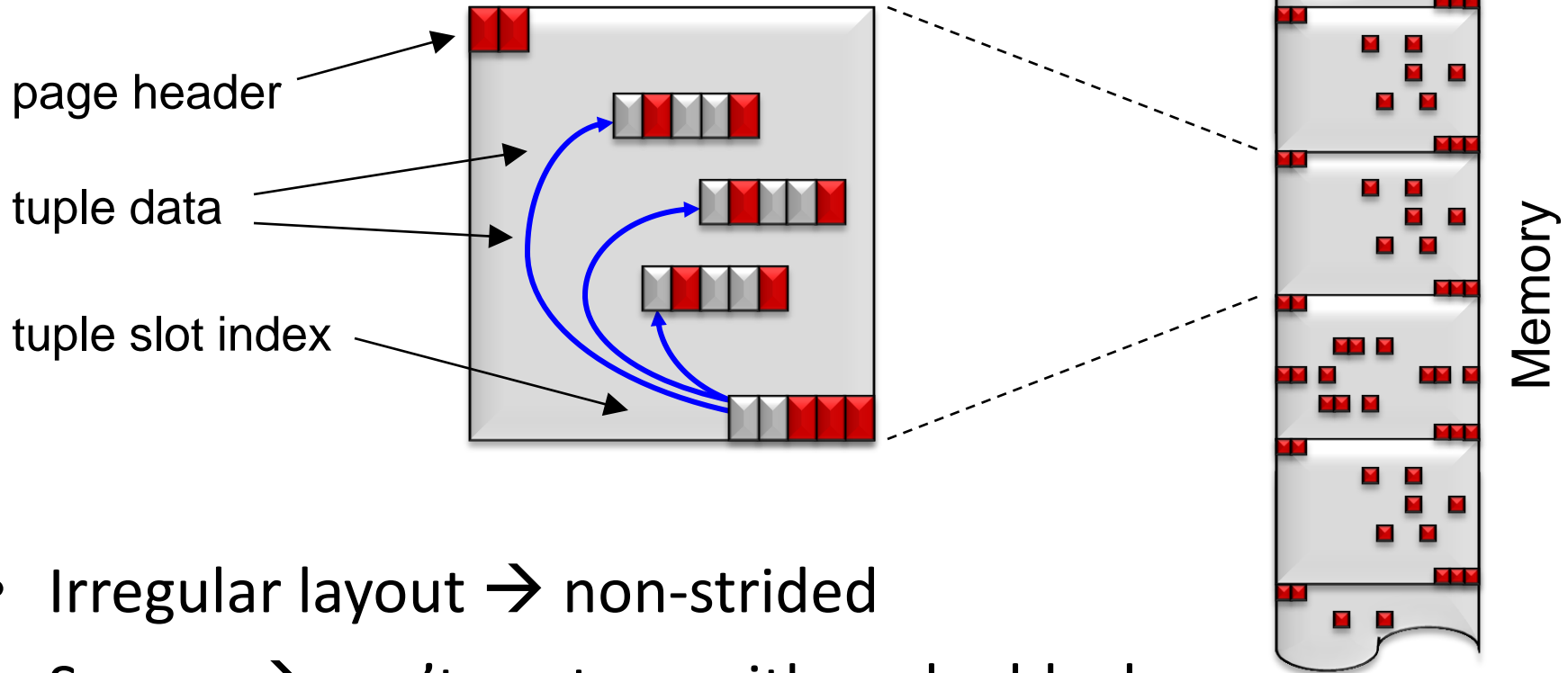
DFS traversal: ABDBEBACFCGCA



Much better accuracy 😊, exponential storage cost ☹️

Spatial Correlation (1/2)

Database Page in Memory (8kB)

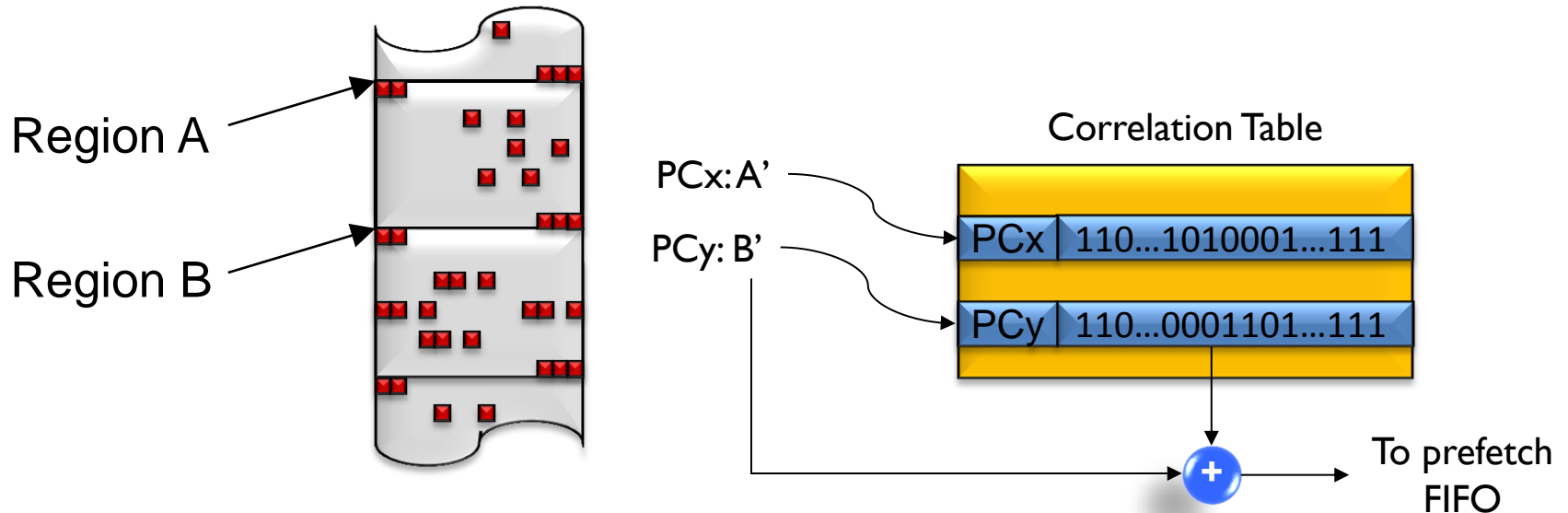


- Irregular layout → non-strided
- Sparse → can't capture with cache blocks
- But, repetitive → predict to improve MLP

Large-scale *repetitive* spatial access patterns

Spatial Correlation (2/2)

- Logically divide memory into regions
- Identify region by base address
- Store spatial pattern (bit vector) in correlation table



Evaluating Prefetchers

- Compare against larger caches
 - Complex prefetcher vs. simple prefetcher with larger cache
- Primary metrics
 - Coverage: prefetched hits / base misses
 - Accuracy: prefetched hits / total prefetches
 - Timeliness: latency of prefetched blocks / hit latency
- Secondary metrics
 - Pollution: misses / (prefetched hits + base misses)
 - Bandwidth: total prefetches + misses / base misses
 - Power, Energy, Area...

Hardware Prefetcher Design Space

- What to prefetch?
 - Predictors regular patterns (x , $x+8$, $x+16$, ...)
 - Predicted correlated patterns ($A...B \rightarrow C$, $B..C \rightarrow J$, $A..C \rightarrow K$, ...)
- When to prefetch?
 - On every reference \rightarrow lots of lookup/prefetcher overhead
 - On every miss \rightarrow patterns filtered by caches
 - On prefetched-data hits (positive feedback)
- Where to put prefetched data?
 - Prefetch buffers
 - Caches

What's Inside Today's Chips

- Data L1
 - PC-localized stride predictors
 - Short-stride predictors within block → prefetch next block
- Instruction L1
 - Predict future PC → prefetch
- L2
 - Stream buffers
 - Adjacent-line prefetch