

Computer Architecture

Prefetching

Prof. Onur Mutlu

ETH Zürich

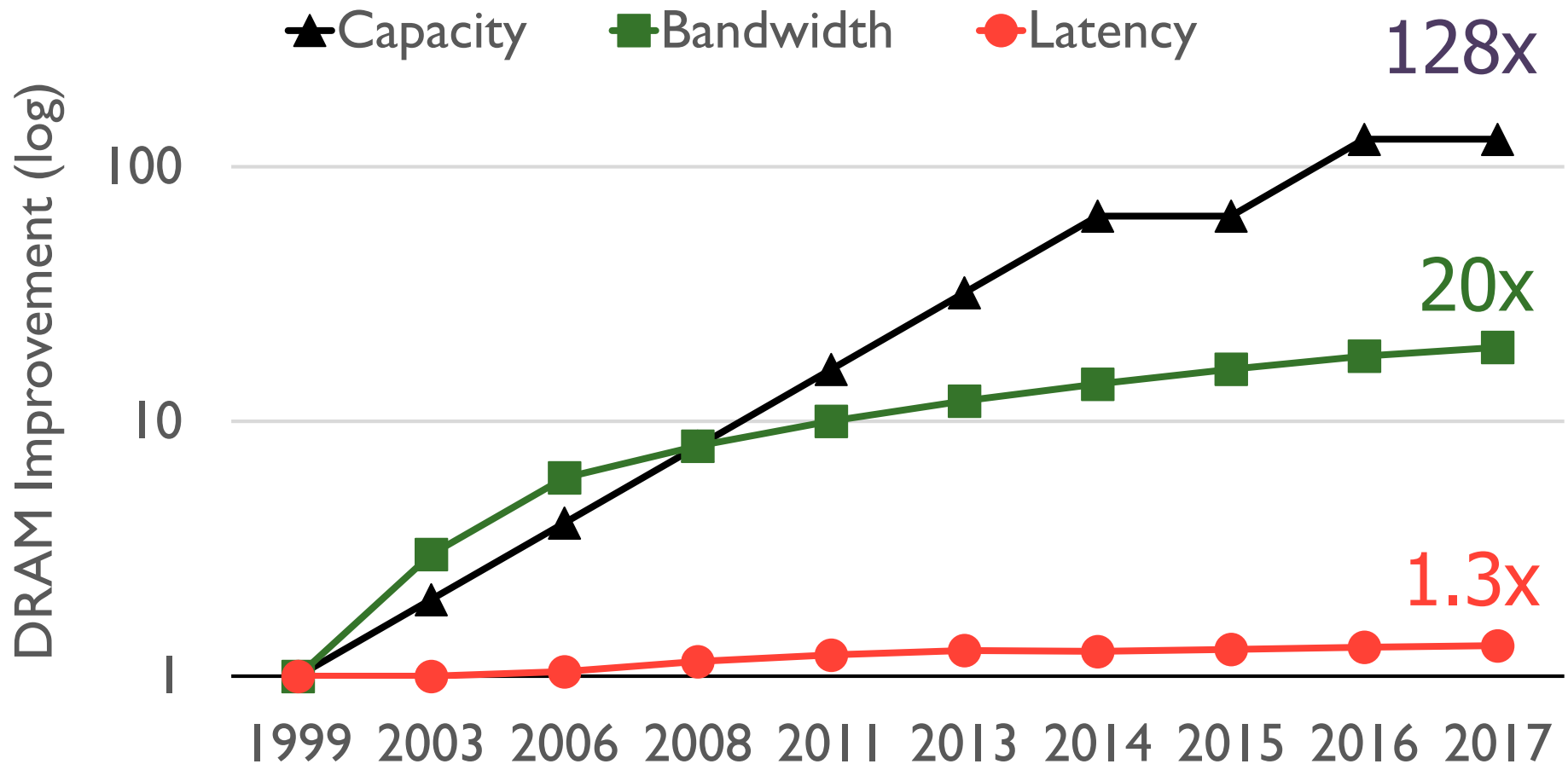
Fall 2024

14 November 2024

Agenda

- Introduction to prefetching
 - What, Why, When, Where, How
- Basic prefetching techniques
 - Software, Hardware, Execution-based
- How to evaluate and control a prefetcher?
 - Metrics & throttling

The (Memory) Latency Problem Latency Lags Behind



Memory latency remains almost constant

Latency Reduction, Hiding, and Tolerance

- Fundamentally reduce latency as much as possible
 - Data-centric approach
- Hide latency seen by the processor
 - Processor-centric approach
 - Caching, Prefetching
- Tolerate (or, amortize) latency seen by the processor
 - Processor-centric approach
 - Multithreading, Out-of-order Execution, Runahead Execution

Conventional Latency Tolerance Techniques

- Caching
 - ❑ Widely used, simple, effective, but inefficient, passive
 - ❑ Not all applications/phases exhibit temporal or spatial locality
- Prefetching
 - ❑ Works well for regular memory access patterns
 - ❑ Prefetching irregular access patterns is difficult, inaccurate, and hardware-intensive
- Multithreading
 - ❑ Works well if there are multiple threads
 - ❑ Improving single thread performance using multithreading hardware is an ongoing research effort
- Out-of-order execution
 - ❑ **Tolerates cache misses that cannot be prefetched**
 - ❑ Requires extensive hardware resources for tolerating long latencies

Prefetching

- Idea: Fetch the data before it is needed by the program (i.e., pre-fetch or pre-load)
- Why?
 - ❑ Memory latency is high. If we can prefetch accurately and early enough, we can reduce/eliminate that latency.
 - ❑ Can eliminate compulsory cache misses
 - ❑ Can it eliminate all cache misses? Capacity, conflict? Coherence?
- Involves predicting which address will be needed in the future
 - ❑ Works if programs have predictable miss address patterns

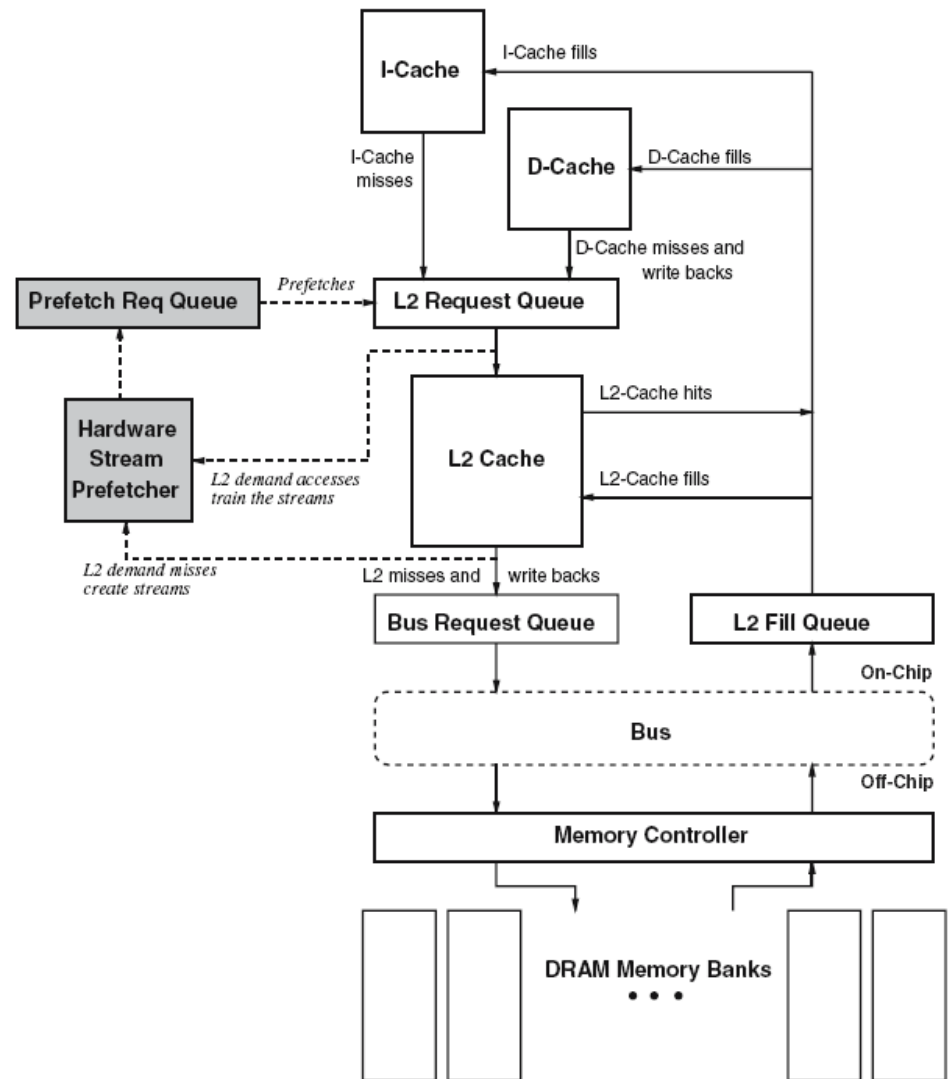
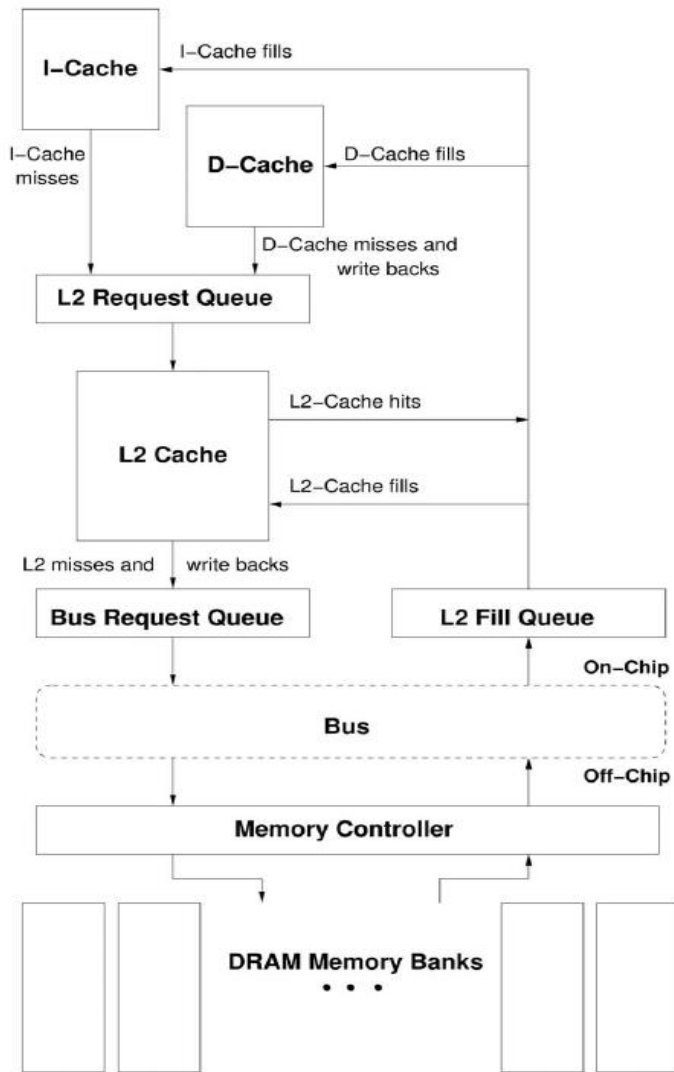
Prefetching and Correctness

- Does a misprediction in prefetching affect correctness?
- No, prefetched data at a “mispredicted” address is simply not used
- There is no need for state recovery
 - In contrast to branch misprediction or value misprediction

Basics

- In modern systems, prefetching is usually done at **cache block granularity**
- Prefetching is a technique that can reduce both
 - ❑ Miss rate
 - ❑ Miss latency
- Prefetching can be done by
 - ❑ Hardware
 - ❑ Compiler
 - ❑ Programmer
 - ❑ System

How a HW Prefetcher Fits in the Memory System



Prefetching: The Four Questions

■ What

- What addresses to prefetch (i.e., address prediction algorithm)

■ When

- When to initiate a prefetch request (early, late, on time)

■ Where

- Where to place the prefetched data (caches, separate buffer)
- Where to place the prefetcher (which level in memory hierarchy)

■ How

- How does the prefetcher operate and who operates it (software, hardware, execution/thread-based, cooperative, hybrid)

Challenge in Prefetching: What

- **What** addresses to prefetch
 - Prefetching useless data wastes resources
 - Memory bandwidth
 - Cache or prefetch buffer space
 - Energy consumption
 - These could all be utilized by demand requests or more accurate prefetch requests
 - **Accurate** prediction of addresses to prefetch is important
 - Prefetch accuracy = used prefetches / sent prefetches
- **How do we know what to prefetch?**
 - Predict based on past access patterns
 - Use the compiler's/programmer's knowledge of data structures
- **Prefetching algorithm** determines what to prefetch

Some Predictable Address Access Patterns?

■ Cache Block Addresses

$A, A+1, A+2, A+3, A+4, \dots$

$B, B+42, B+84, B+126, \dots$

$C, C+2, C+5, C+9, C+11, C+14, C+18, C+20, C+23, C+27, \dots$

$X, Y, T, Q, R, S, X, Y, T, A, B, C, D, E, X, Y, T, F, G, H, X, Y, T, \dots$

$A+1, A+67, A+18, A+7, A+99, Z+1, Z+67, Z+18, Z+7, Z+99, P+1, P+67, P+18, P+7, P+99, \dots$

Challenges in Prefetching: When

- **When** to initiate a prefetch request
 - Prefetching too early
 - Prefetched data might not be used before it is evicted from storage
 - Prefetching too late
 - Might not hide the whole memory latency
- When a data item is prefetched affects the **timeliness** of the prefetcher
- Prefetcher can be made more timely by
 - Making it more **aggressive**: try to stay far ahead of the processor's demand access stream (hardware)
 - Moving the **prefetch instructions earlier in the code** (software)

Challenges in Prefetching: Where (I)

- **Where** to place the prefetched data
 - In **cache**
 - + Simple design, no need for separate buffers
 - Can evict useful demand data → cache pollution
 - In a separate **prefetch buffer**
 - + Demand data protected from prefetches → no cache pollution
 - More complex memory system design
 - Where to place the prefetch buffer
 - When to access the prefetch buffer (parallel vs. serial with cache)
 - When to move the data from the prefetch buffer to cache
 - How to size the prefetch buffer
 - Keeping the prefetch buffer coherent
- Many modern systems place prefetched data into the cache
 - Many Intel, AMD, IBM systems and more ...

Challenges in Prefetching: Where (II)

- Which level of cache to prefetch into?
 - Memory to L4/L3/L2, memory to L1. Advantages/disadvantages?
 - L3 to L2? L2 to L1? (a separate prefetcher between levels)

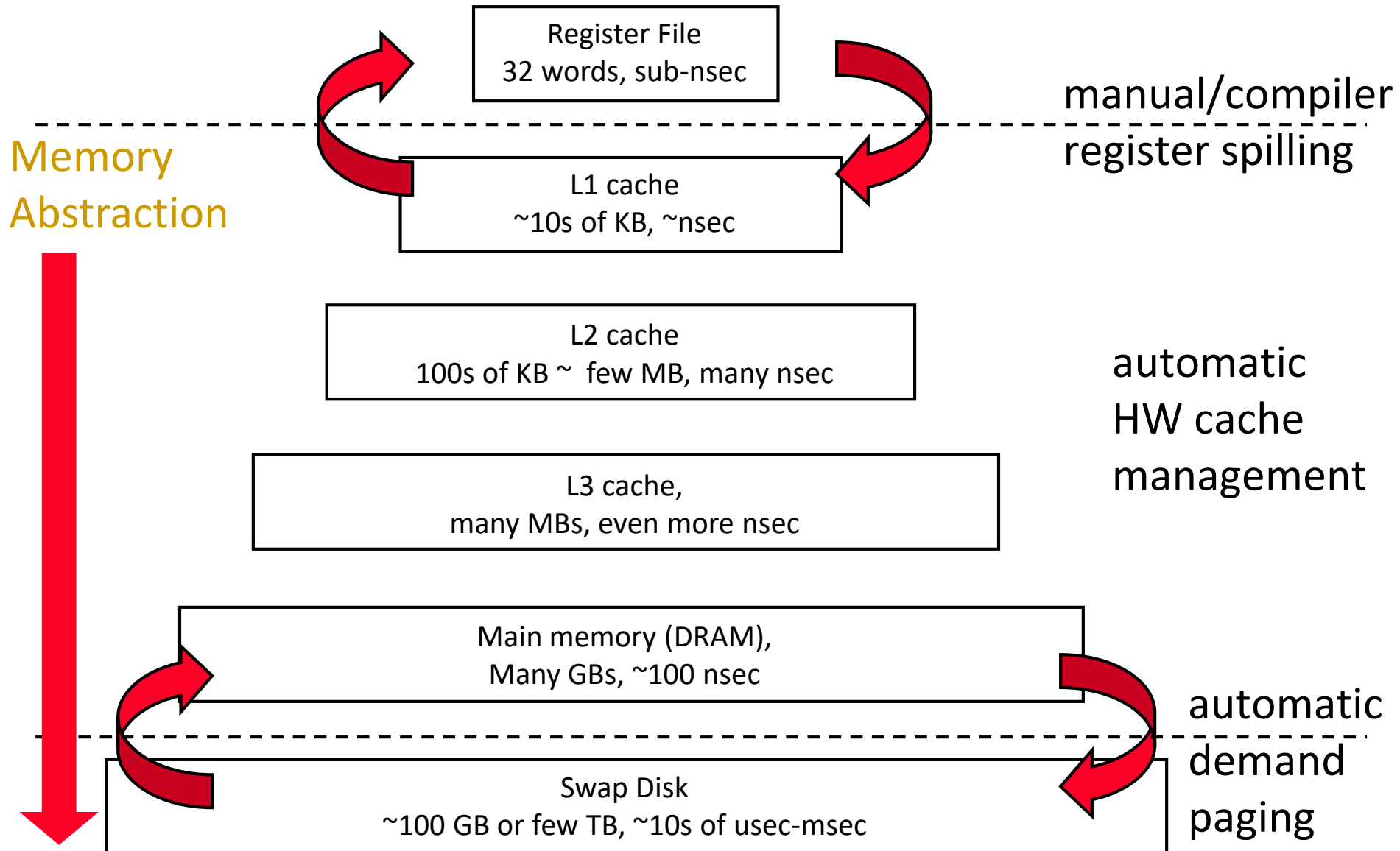
- Where to place the prefetched data in the cache?
 - Do we treat prefetched blocks the same as demand-fetched blocks?
 - Prefetched blocks are not known to be needed
 - With LRU, a demand block is placed into the MRU position

- Do we skew the replacement policy such that it favors the demand-fetched blocks?
 - E.g., place all prefetches into the LRU position in a way?

Challenges in Prefetching: Where (III)

- **Where** to place the hardware prefetcher in the memory hierarchy?
 - ❑ In other words, what access patterns does the prefetcher see?
 - ❑ L1 hits and misses
 - ❑ L1 misses only
 - ❑ L2 misses only
- Seeing a more complete access pattern:
 - + Potentially better **accuracy** and **coverage** in prefetching
 - Prefetcher needs to examine more requests (bandwidth intensive, more ports into the prefetcher?)

A Modern Memory Hierarchy



Challenges in Prefetching: **How**

- **Software** prefetching
 - ❑ ISA provides prefetch instructions
 - ❑ Programmer or compiler inserts prefetch instructions (effort)
 - ❑ Usually works well only for “regular access patterns”
- **Hardware** prefetching
 - ❑ Specialized hardware monitors memory accesses
 - ❑ Memorizes, finds, learns address strides/patterns/correlations
 - ❑ Generates prefetch addresses automatically
- **Execution-based** prefetchers
 - ❑ A “thread” is executed to prefetch data for the main program
 - ❑ Can be generated by either software/programmer or hardware

Challenges in Prefetching: **How**

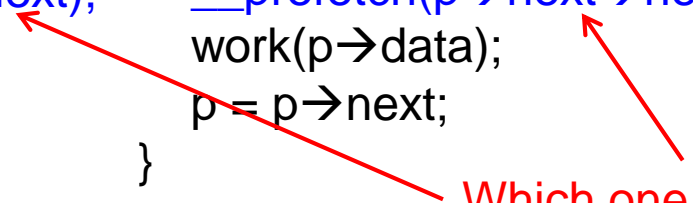
- **Software** prefetching
 - ❑ ISA provides prefetch instructions
 - ❑ Programmer or compiler inserts prefetch instructions into code
 - ❑ Usually works well only for “regular access patterns”
- **Hardware** prefetching
 - ❑ Specialized hardware monitors memory accesses
 - ❑ Memorizes, finds, learns address strides/patterns/correlations
 - ❑ Generates prefetch addresses automatically
- **Execution-based** prefetching
 - ❑ A “thread” is executed to prefetch data for the main program
 - ❑ Can be generated by either software/programmer or hardware

Software Prefetching (I)

- Idea: Compiler/programmer places prefetch instructions into appropriate places in code
- Prefetch instructions prefetch data into caches
- Compiler or programmer can insert such instructions into the program

Software Prefetching (II)

```
for (i=0; i<N; i++) {      while (p) {                  while (p) {
    __prefetch(a[i+8]);      __prefetch(p->next);      __prefetch(p->next->next->next);
    __prefetch(b[i+8]);      work(p->data);                  work(p->data);
    sum += a[i]*b[i];        p = p->next;                  p = p->next;
}                             }                             }
```



Which one is better?

- Can work for very regular array-based access patterns. Issues:
 - Prefetch instructions take up processing/execution bandwidth
 - **How early to prefetch?** Determining this is difficult
 - Prefetch distance depends on hardware implementation (memory latency, cache size, time between loop iterations) → portability?
 - Going too far back in code reduces accuracy (branches in between)
 - Need “special” prefetch instructions in ISA?
 - Alpha load into register 31 treated as prefetch (r31==0)
 - PowerPC *dcbt* (data cache block touch) instruction
 - Not easy to do for pointer-based data structures

Software Prefetching (III)

- Where should a compiler insert prefetches?
 - Prefetch for every load access?
 - Too bandwidth intensive (both memory and execution bandwidth)
 - Profile the code and determine loads that are likely to miss
 - What if profile input set is not representative?
 - How far ahead before the miss should the prefetch be inserted?
 - Profile and determine probability of use for various prefetch distances from the miss
 - What if profile input set is not representative?
 - Usually need to insert a prefetch far in advance to cover 100s of cycles of main memory latency → reduced accuracy

Challenges in Prefetching: **How**

■ Software prefetching

- ❑ ISA provides prefetch instructions
- ❑ Programmer or compiler inserts prefetch instructions into code
- ❑ Usually works well only for “regular access patterns”

■ Hardware prefetching

- ❑ Specialized hardware monitors memory accesses
- ❑ Memorizes, finds, learns address strides/patterns/correlations
- ❑ Generates prefetch addresses automatically

■ Execution-based prefetching

- ❑ A “thread” is executed to prefetch data for the main program
- ❑ Can be generated by either software/programmer or hardware

Hardware Prefetching

- Idea: Specialized hardware observes load/store access patterns and prefetches data based on past access behavior
- Tradeoffs:
 - + Can be tuned to system implementation
 - + Does not waste instruction execution bandwidth
 - More hardware complexity to detect patterns
 - Software can be more efficient in some cases

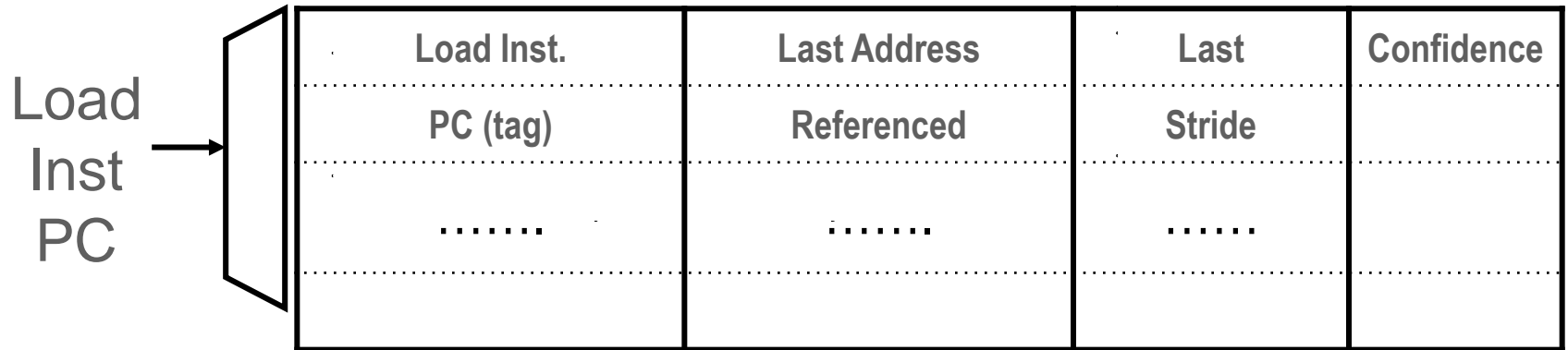
Next-Line Prefetchers

- Simplest form of hardware prefetching: always prefetch next N cache lines after a demand access (or a demand miss)
 - Next-line prefetcher (or next sequential prefetcher)
 - Tradeoffs:
 - + Simple to implement. No need for sophisticated pattern detection
 - + Works well for sequential/streaming access patterns (instructions?)
 - Can waste bandwidth with irregular patterns
 - And, even with regular patterns:
 - What is the prefetch accuracy if access stride = 2 and $N = 1$?
 - What if the program is traversing memory from higher to lower addresses?
 - Also prefetch “previous” N cache lines?

Stride Prefetchers

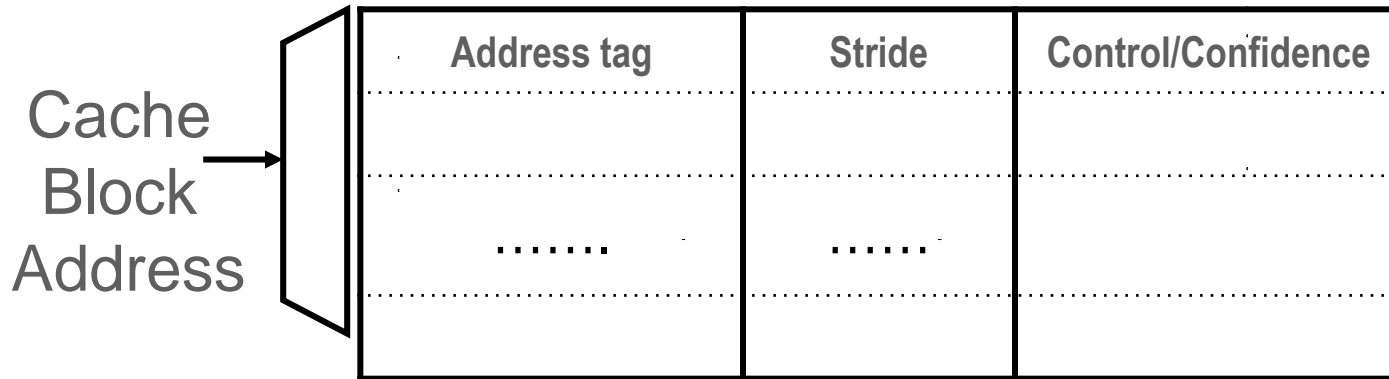
- Consider the following strided memory access pattern:
 - $A, A+N, A+2N, A+3N, A+4N\dots$
 - $\text{Stride} = N$
- Idea: Record the stride between consecutive memory accesses; if stable, use it to predict next M memory accesses
- Two types
 - Stride determined on a per-instruction basis
 - Stride determined on a per-memory-region basis

Instruction Based Stride Prefetching



- Each load/store instruction can lead to a memory access pattern with a different stride
 - Can only detect strides caused by each instruction
- Timeliness of prefetches can be an issue
 - Initiating the prefetch when the load is fetched the next time can be too late
 - Potential solution: Look ahead in the instruction stream

Memory-Region Based Stride Prefetching



- Can detect strided memory access patterns that appear due to multiple instructions
 - $A, A+N, A+2N, A+3N, A+4N \dots$ where each access could be due to a different instruction
- **Stream prefetching (stream buffers)** is a special case of memory-region based stride prefetching where $N = 1$

Tradeoffs in Stream/Stride Prefetching

- Instruction based stride prefetching vs. memory region based stride prefetching
- The latter can exploit strides that occur due to the **interaction of multiple instructions**
- The latter can more easily get **further ahead** of the processor access stream
 - No need for lookahead PC
- The latter is more hardware intensive
 - Usually there are more data addresses to monitor than instructions