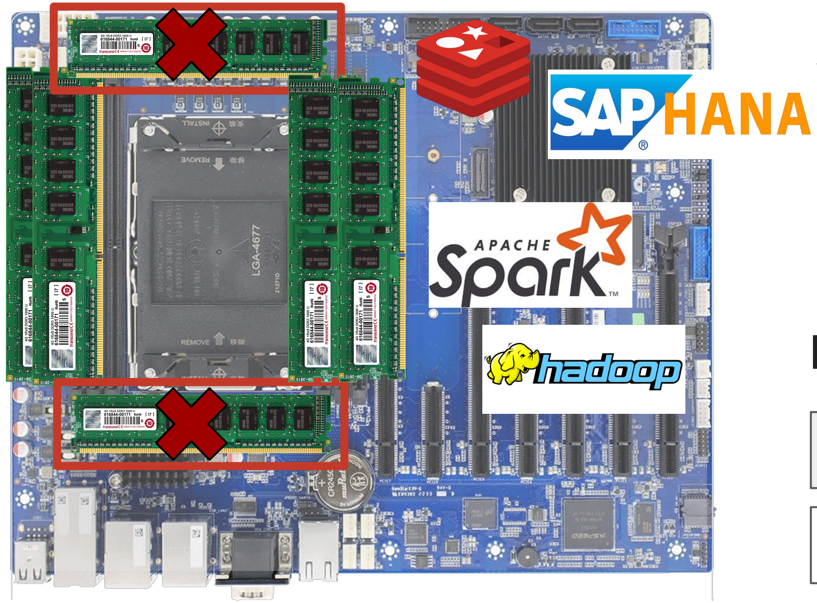
Tolerate It if You Cannot Reduce It: Handling Latency in Tiered Memory

Musa Unal, Vishal Gupta, Yueyang Pan, Yujie Ren, Sanidhya Kashyap

EPFL

Tiered memory



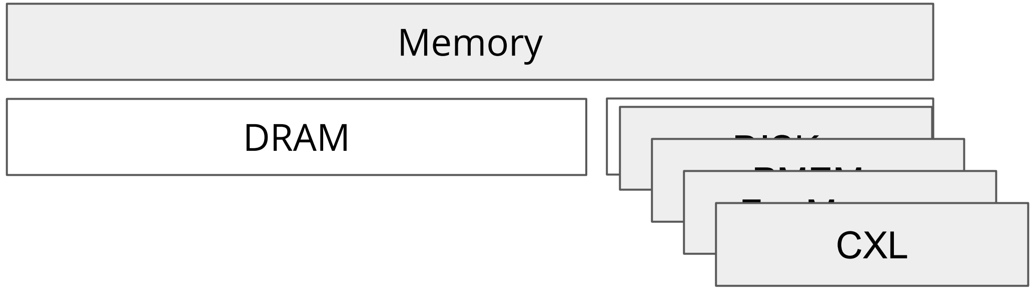
Why ?

Applications demand for memory.

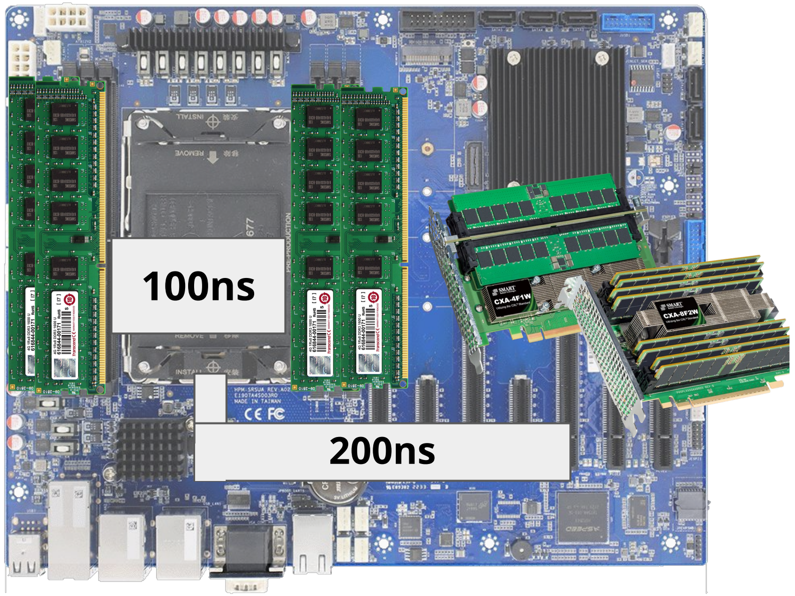
We want to expand main memory.

We can not add more DRAMs.

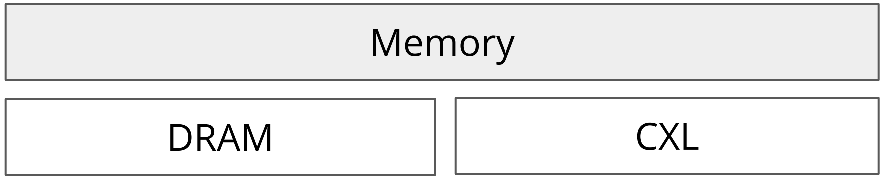
How ?



CXL expands main memory



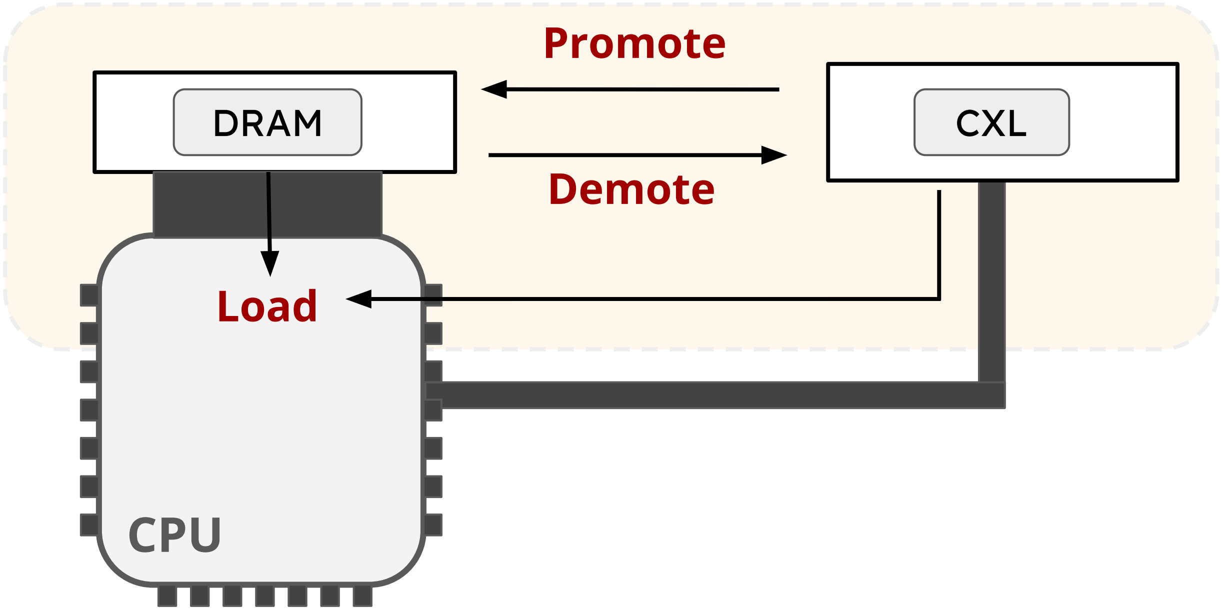
CXL Attached Memory



- Protocol for fast memory access over PCIe
- Latency is 2x higher (or higher with different topologies)

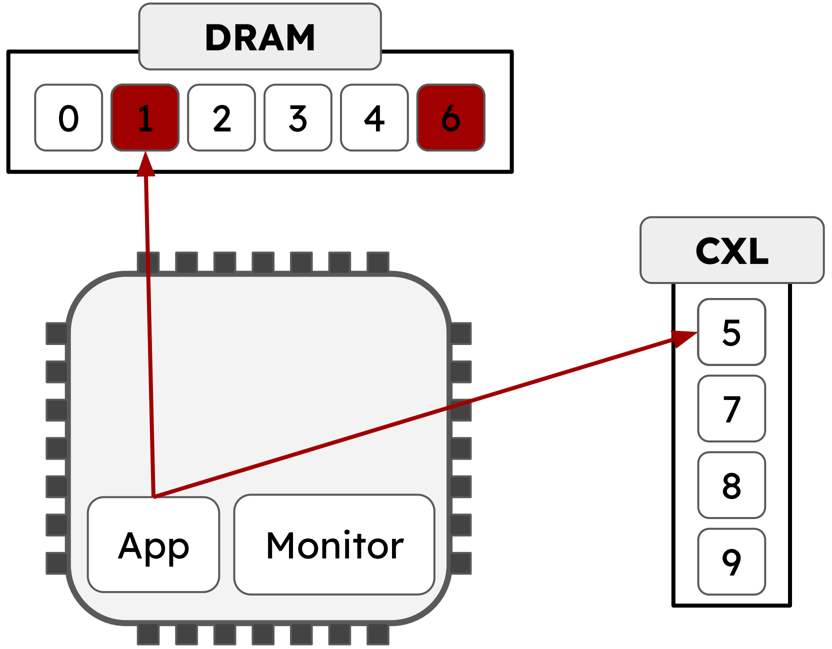
How can we handle the latency in tiered memory?

Conventional approach to handle latency

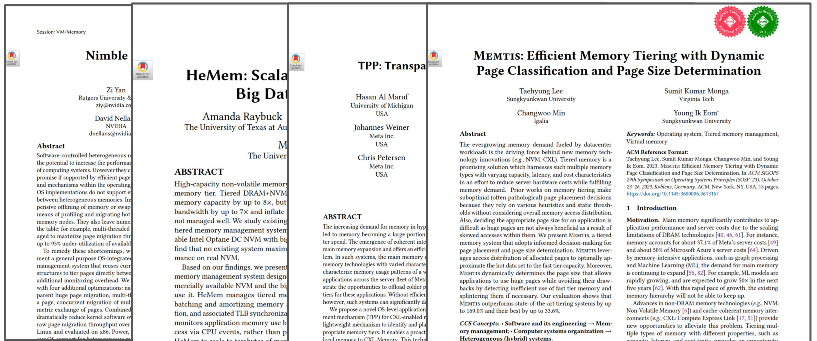


Reduce

Page migration can reduce the latency



1. Tiering systems monitor pages hotness information.
2. Promote hot pages
   Demote cold pages
3. Increase **locality**, and **reduce** the latency


Nimble

HeMem: Scala
Big Dat

TPP: Transpa

MEMTIS: Efficient Memory Tiering with Dynamic Page Classification and Page Size Determination

Taehyung Lee, Sungkyunkwan University
Sumit Kumar Monga, Virginia Tech
Changwoo Min, Igalia
Young Ik Eom*, Sungkyunkwan University

Abstract

The evergrowing memory demand fueled by datacenter workloads is the driving force behind new memory technology innovations (*e.g.*, NVM, CXL). Tiered memory is a promising solution which harnesses such multiple memory types with varying capacity, latency, and cost characteristics in an effort to reduce server hardware costs while fulfilling memory demand. Prior works on memory tiering make suboptimal (often pathological) page placement decisions because they rely on various heuristics and static thresholds without considering overall memory access distribution. Also, deciding the appropriate page size for an application is difficult as huge pages are not always beneficial as a result of skewed accesses within them. We present MEMTIS, a tiered memory system that adopts informed decision-making for page placement and page size determination. MEMTIS leverages access distribution of allocated pages to optimally approximate the hot data set to the fast tier capacity. Moreover, MEMTIS dynamically determines the page size that allows applications to use huge pages while avoiding their drawbacks by detecting inefficient use of fast tier memory and splintering them if necessary. Our evaluation shows that MEMTIS outperforms state-of-the-art tiering systems by up to 169.0% and their best by up to 33.6%.

CCS Concepts: • **Software and its engineering → Memory management**; • **Computer systems organization → Heterogeneous (hybrid) systems.**

Keywords: Operating system, Tiered memory management, Virtual memory

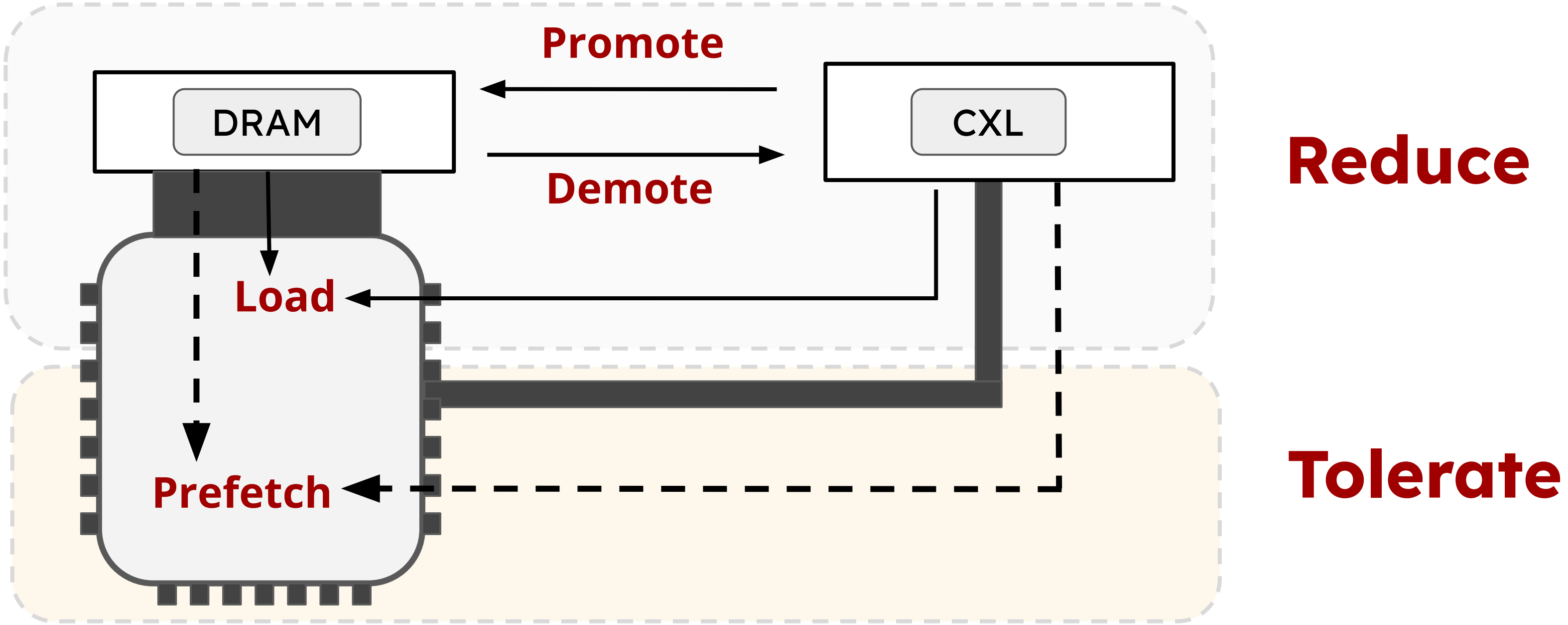
ACM Reference Format:
Taehyung Lee, Sumit Kumar Monga, Changwoo Min, and Young Ik Eom. 2023. MEMTIS: Efficient Memory Tiering with Dynamic Page Classification and Page Size Determination. In *ACM SIGOPS 29th Symposium on Operating Systems Principles (SOSP '23), October 23–26, 2023, Koblenz, Germany*. ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3600006.3613167

1 Introduction

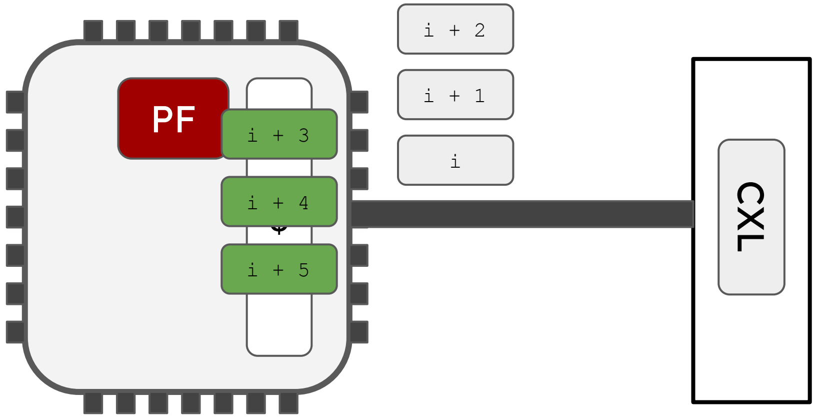
Motivation. Main memory significantly contributes to application performance and server costs due to the scaling limitations of DRAM technologies [40, 46, 61]. For instance, memory accounts for about 37.1% of Meta's server costs [49] and about 50% of Microsoft Azure's server costs [64]. Driven by memory-intensive applications, such as graph processing and Machine Learning (ML), the demand for main memory is continuing to expand [55, 82]. For example, ML models are rapidly growing, and are expected to grow 50× in the next five years [62]. With this rapid pace of growth, the existing memory hierarchy will not be able to keep up.

Advances in non-DRAM memory technologies (*e.g.*, NVM: Non-Volatile Memory [6]) and cache-coherent memory interconnects (*e.g.*, CXL: Compute Express Link [17, 51]) provide new opportunities to alleviate this problem. Tiering multiple types of memory with different properties, such as

Missing aspect to handle latency



Hardware prefetchers can tolerate latency

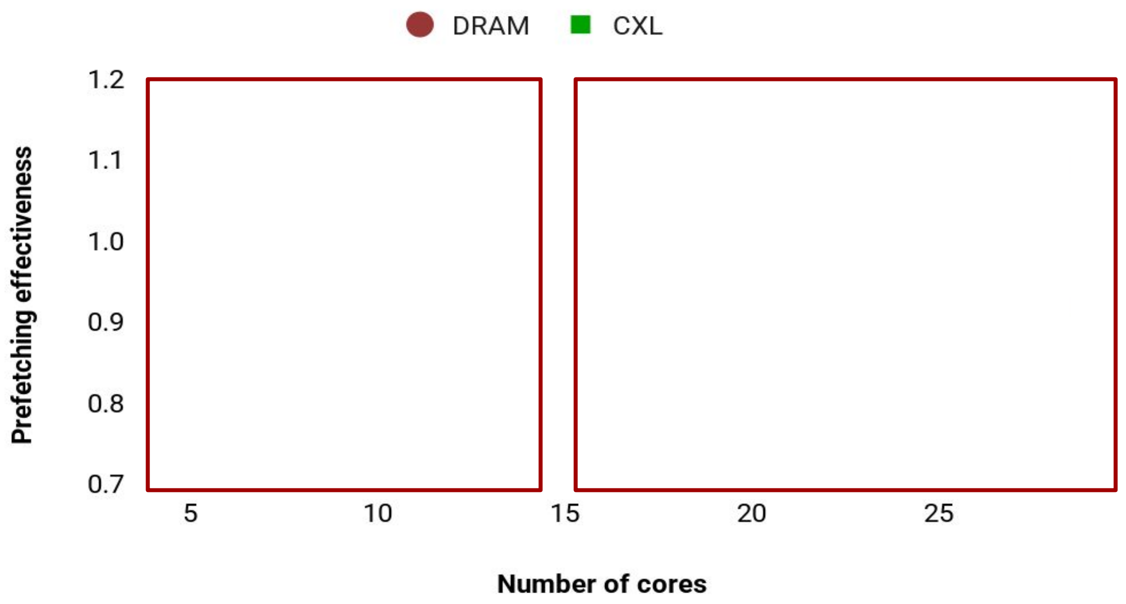


Sequential Pointer Chase

Stride Size	**CXL Lat / DRAM Lat**
4096	~2.01
128	~1.79
64	~1.55

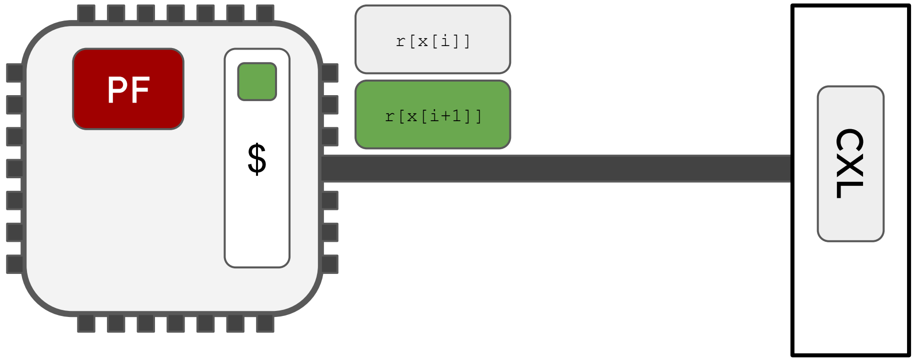
Hardware prefetchers can help to hide CXL's latency.

Hardware prefetchers can be harmful



Prefetching can degrade the performance under high loads.

Software prefetching can help as well



```
for i in range(1M):
  e = records[x[i]]
  work(e)
```

```
for i in range(1M):
  e = records[x[i]]
  prefetch(records[x[i+1]])
  work(e)
```

```
for i in range(1M):
  e = records[x[i]]
  prefetch(records[x[i+k]])
  work(e)
```

Prefetch distance

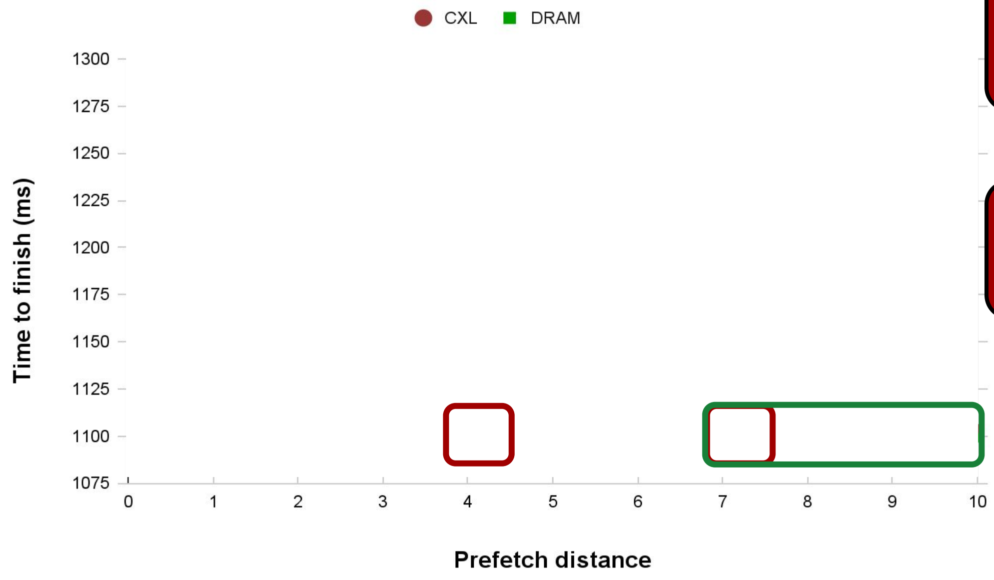
Timeliness is important!

Early prefetches can be evicted.

Late prefetches are not helpful.

Prefetch distance matters

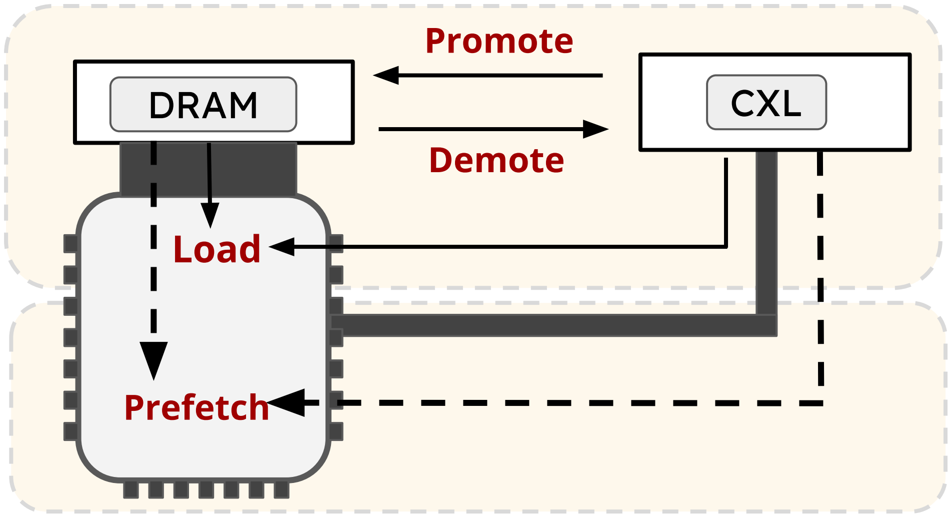
Scan microbenchmark



Prefetch distance should differ across tiers

Prefetching can effectively **hide** the latency

Linden: Combining reduce and tolerate



Reduce → Hotness

➕

Tolerate → Prefetchability

Prefetchability as a tolerate metric

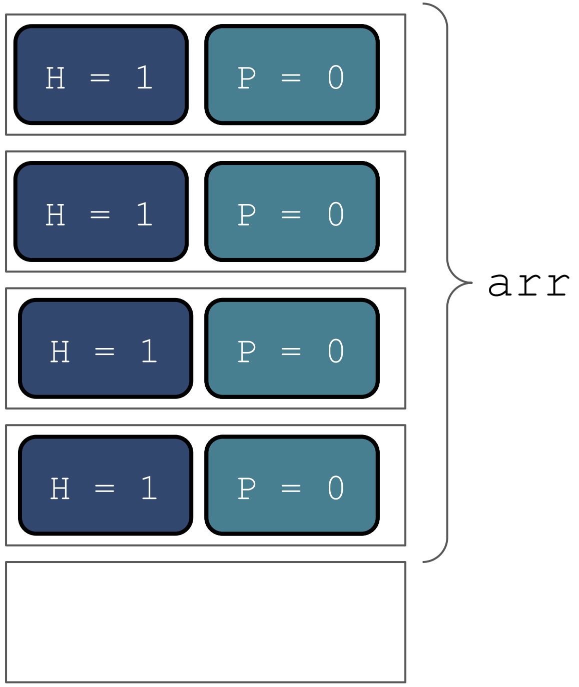
Can we identify memory regions that will get benefit from prefetching ?

Prefetchability:
Percentage of the loads to particular region that can be hidden by prefetching.

$$\frac{\textit{Number of loads hidden with prefetching}}{\textit{Issued loads}}$$

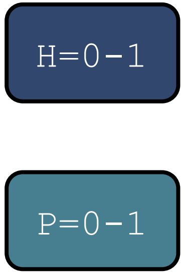
Thread	**Region**	**Access Pattern**	**Prefetchability Ratio**
T_1	arr	Sequential	1
T_2	arr_2	Random	0

Combining hotness and prefetchability



Memory Pages

```
// rand access
arr = [...]
for j in range(10):
      for i in range(1000):
            sum += arr[rand]
```



Hotness

Prefetchability

Linden’s objectives and workflow

Prefetching can hide CXL’s latency.

Under high contentions prefetching can be harmful

Prefetch distances matter in different tiers

Compiler

Identify prefetchable access patterns

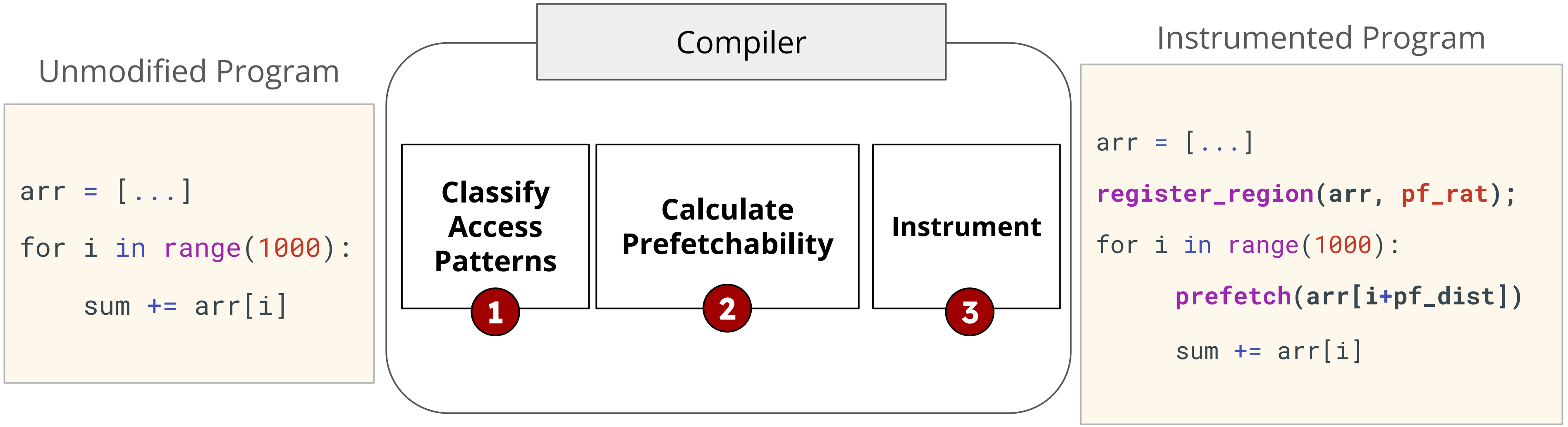
Insert software prefetch hints.

Runtime

Observe dynamic behaviours

Adapts to the current state

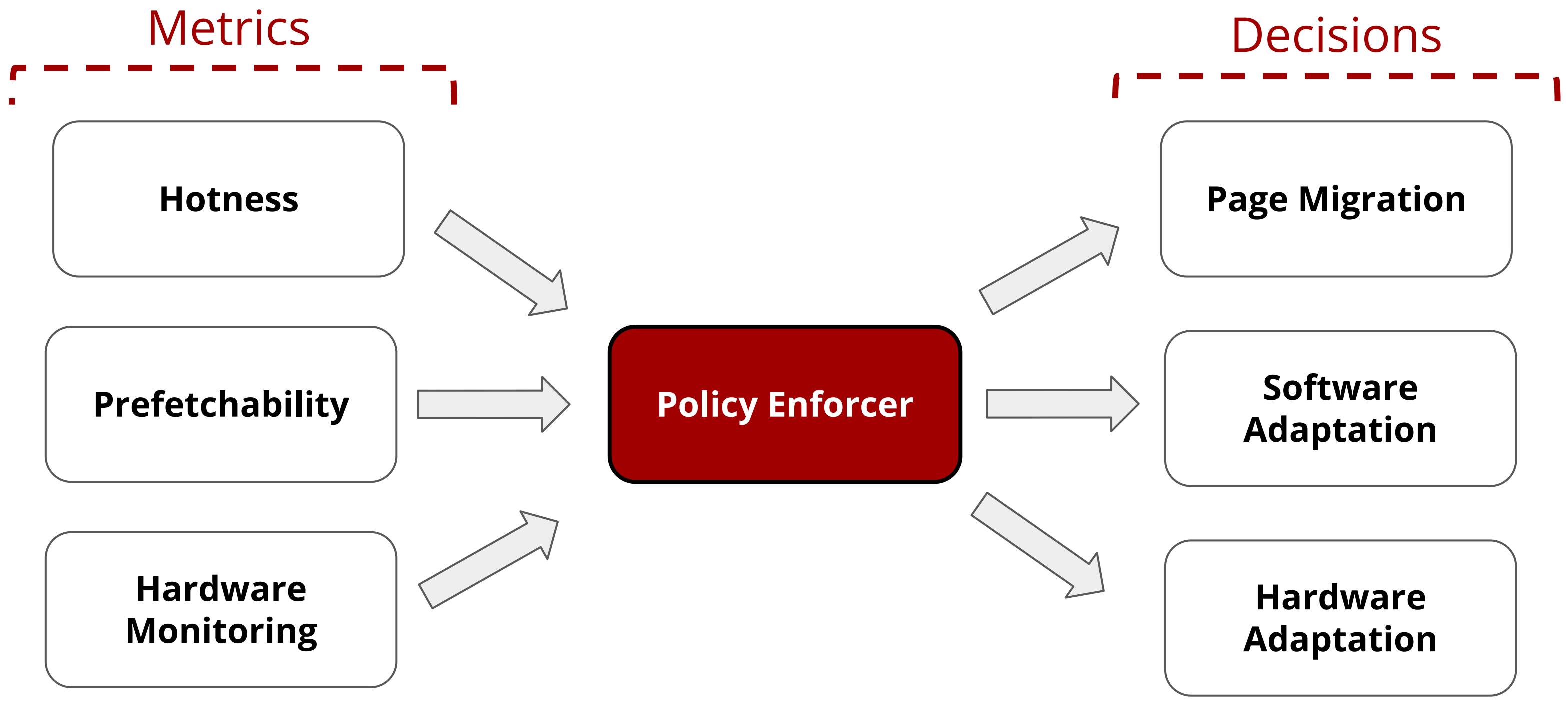
Linden compiler



```
arr = [...]
for i in range(1000):
    sum += arr[i]
```

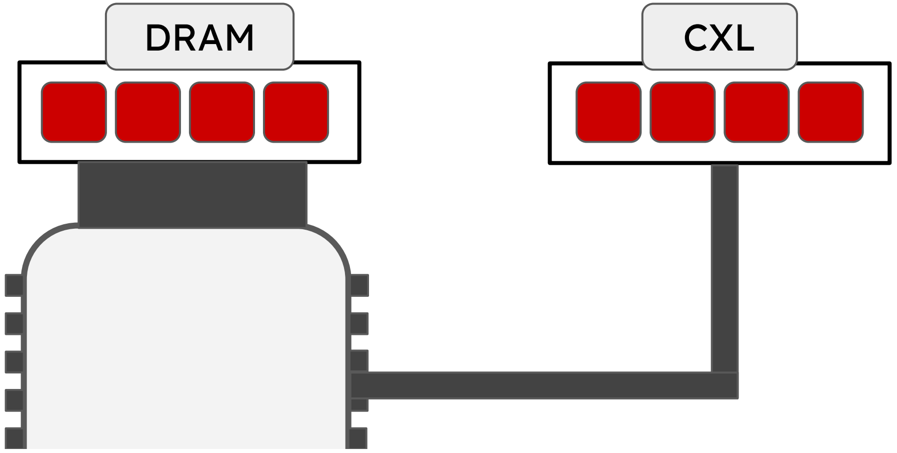
```
arr = [...]
register_region(arr, pf_rat);
for i in range(1000):
    prefetch(arr[i+pf_dist])
    sum += arr[i]
```

Linden’s runtime responsibilities



Page migration

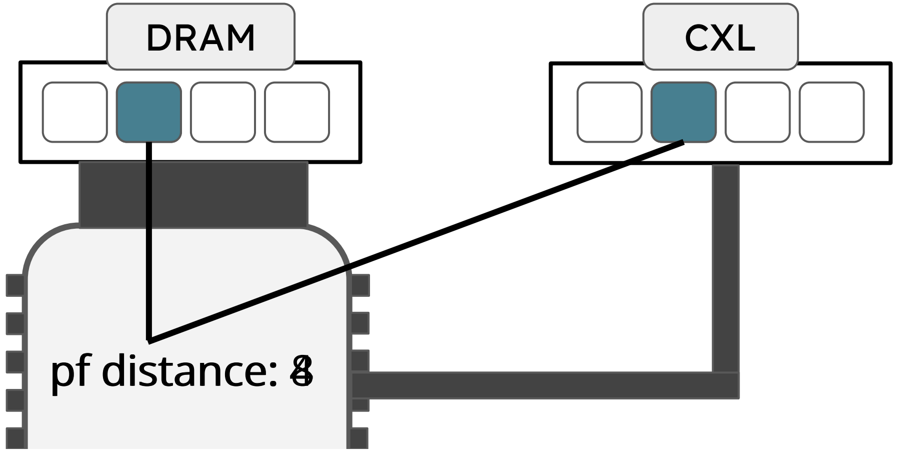
What happens if all regions are "hot" ?



- *Hotness + Prefetchability*
- Find memory regions that can **tolerate** the latency.
- Then migrate these pages to the CXL.

Software adaptation

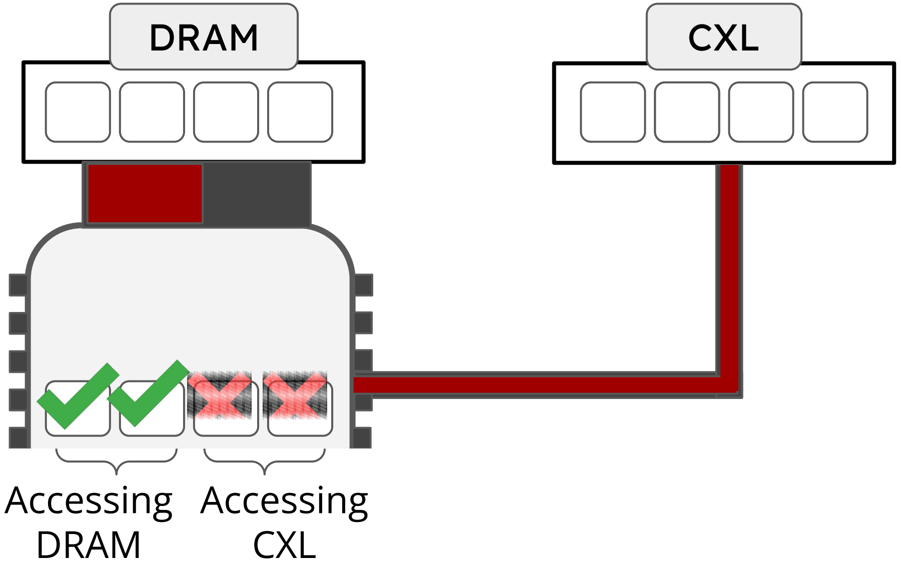
What if a region with prefetch migrated?



- Prefetch distance should be updated!
- Dynamically adjust the prefetch distances with considering the tier information.

Hardware adaptation

What happens under high contention.



- Disable hardware prefetching!
- Check which threads are accessing the CXL memory.

Linden: Tolerate It if You Cannot Reduce It

- Two techniques for tiered memory management:
  - Reduce understands the hotness.
  - Tolerate understands the prefetchability.

- Why toleration matters ?
  - Can help to hide CXL’s latency
  - Careful monitoring is essential
  - Hardware and software adaptations needed

Thanks