Architectural Support for Differentiated Memory in Datacenter Servers

Chun Deng and Anna Eaton

DAM workshop Jan. 2025

Differentiated Access Memories in Datacenter Servers

- How do we do memory allocation among differentiated access memories in datacenter servers?

- Unlike in ML accelerators, it is infeasible to do compile time optimizations
- Datacenter workloads are highly diverse, and written in traditional languages

- Given some page, which memory does it want to be in?

Our Approach

- MMU on processor tracks accesses to each page/huge page
- Use this stream of accesses to estimate “fit” to each available memory type
- Software uses estimates to decide where to place and when to move pages

- Key insight: MMU can inexpensively estimate fit in constant time by modeling it as compression of the access sequence
  - Good compression = good fit
  - Each memory type has a compressor trained on “good” access sequences

- Rest of this talk: technical challenges and implementation details

Outline

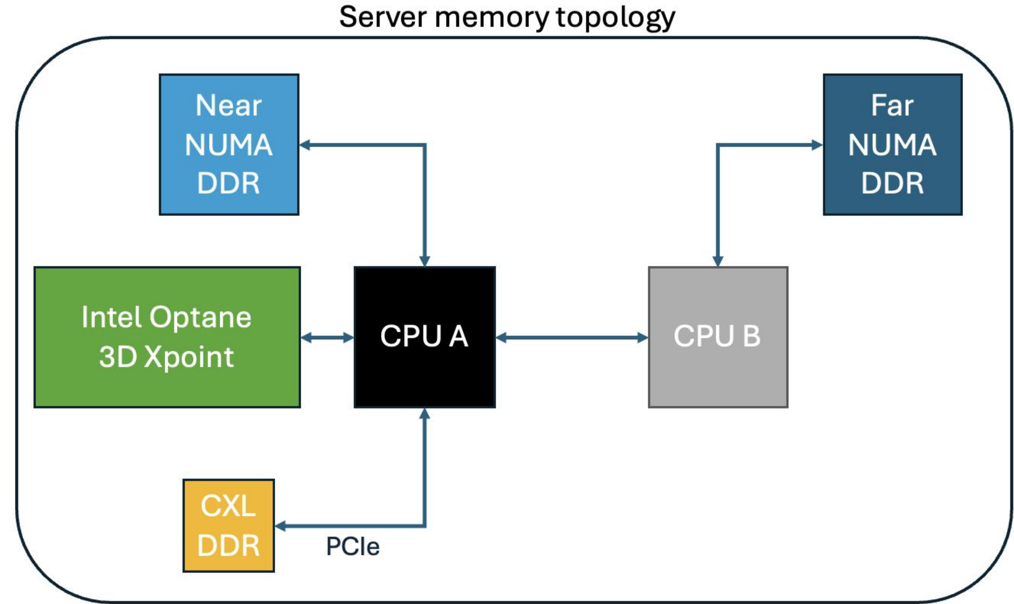
- Introduction
- Problem 1: which memory is best for an access sequence?
- Problem 2: classifying running applications at memory speeds
- Problem 3: deciding where to place and when to move pages
- Discussion and future work

Outline

- Introduction
- Problem 1: which memory is best for an access sequence?
- Problem 2: classifying running applications at memory speeds
- Problem 3: deciding where to place and when to move pages
- Discussion and future work

Differentiated Memory

- Don't assume a strict hierarchy of memories
- Leverage differences in memory for performance gains



	Read Latency (ns)	Write Latency (ns)	Bandwidth (Gb/s)	Size (GB)
Near NUMA DDR	60–80	80–100	20–25	512
Far NUMA DDR	100–200	120–250	20–30	512
Optane 3D Xpoint	300–500	10000–15000	10–20	960
CXL DDR	300–500	300–700	8–15	256

Cost model

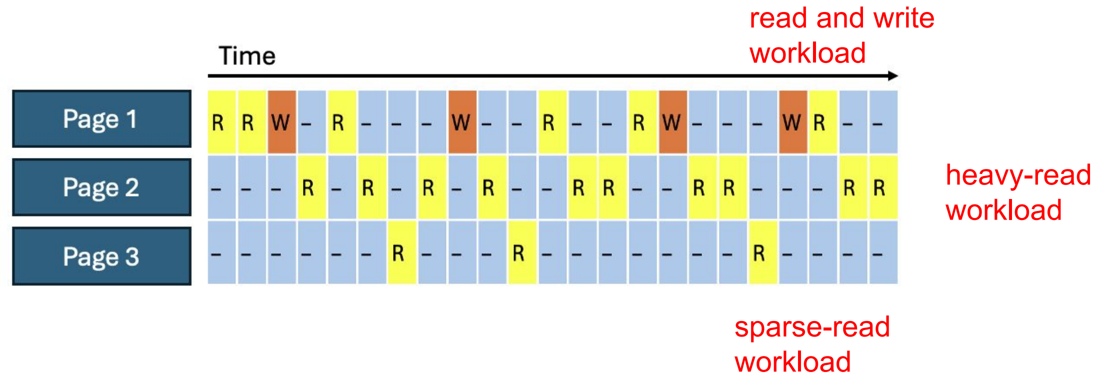
- Assign some relative cost to operations in different memories

- Non-volatile memories don't have this standby/refresh cost

	Read	Write	Standby including refresh	Standby excluding refresh	Write Endurance	Retention
SRAM	1	1	10	10	infinite	N/A
DRAM	50	50	12	5	infinite	64 ms
Si Gain Cell	2	2	20	5	infinite	10 us
Oxide Gain Cell	10	10	6	5	infinite	10 s
RRAM	20	1000	0	0	10^5	10 y
MRAM	10	1000	0	0	10^{10}	10 y
FeRAM	100	100	0	0	10^{15}	10 y
FeFET	10	100	0	0	10^5	10 y

Page access traces and cost model

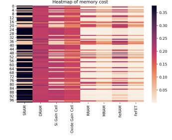
- Each page is accessed with reads and writes
- Maintain a sparse record of each page's history
- Applying a cost model using the characteristics of each memory can show which memories each page would prefer



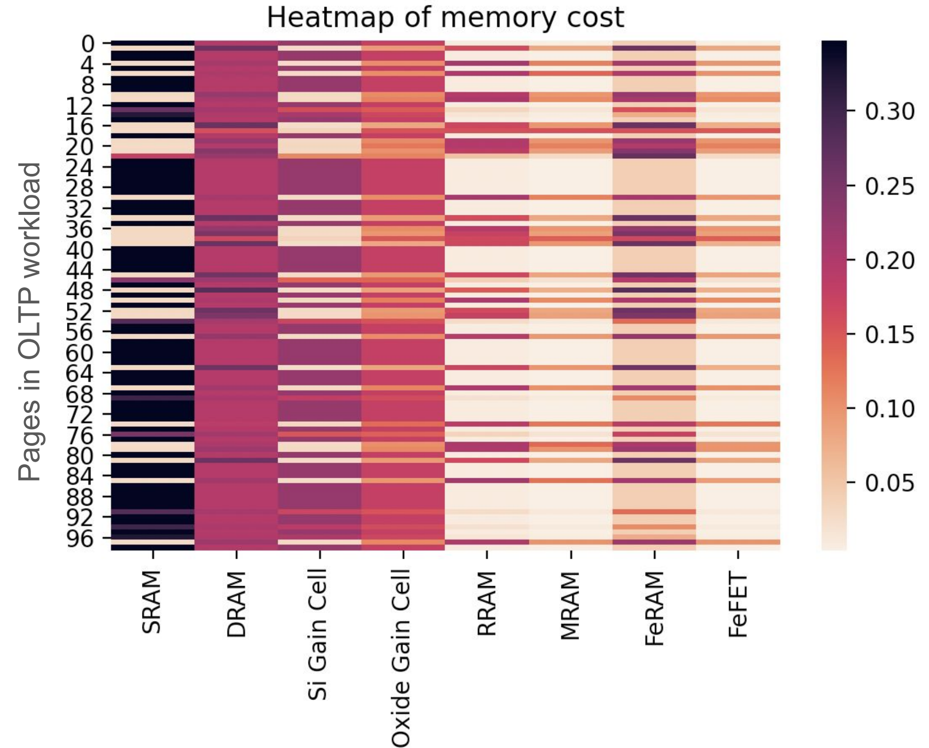
Methodology: Looking at real memory traces

- Capture memory accesses on OLTP (online transaction processing) and OLAP (online analytical processing) workloads, using a synthetic database benchmark on an Intel x86_64 machine
- Split memory accesses by page
- Simulate the memory cost function for each type of memory for each page
- Choose the winning memories for each page

Results: Looking at real memory traces

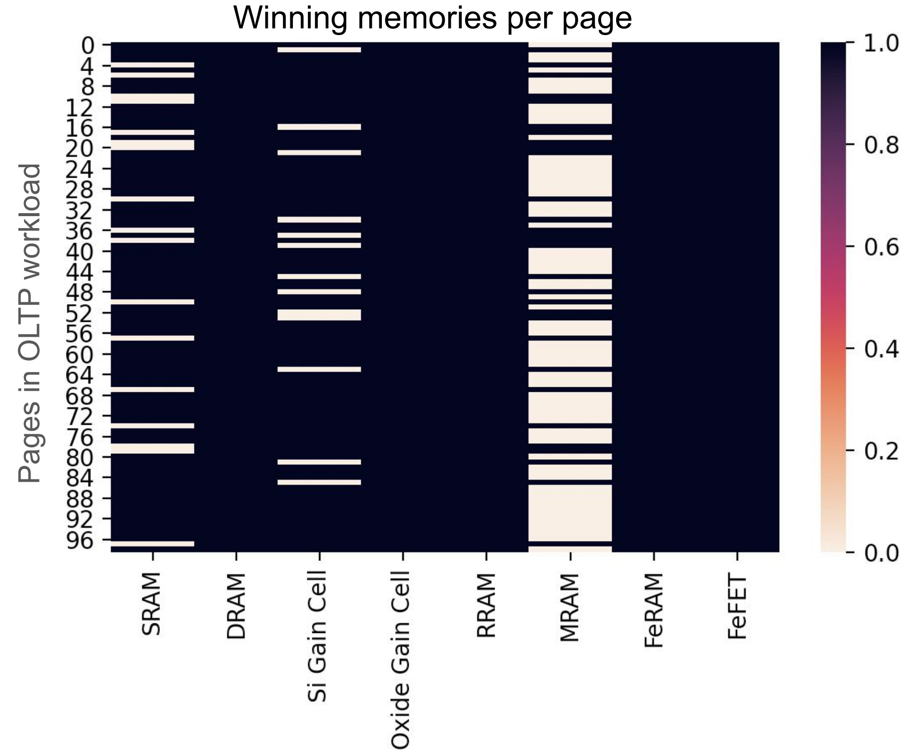


- High variation in memory preferences



Results: Looking at real memory traces

- 100 pages in OLTP workload preferred mostly SRAM, Si gain cell, or MRAM
- Evident diversity in preferred memories



Outline

- Introduction
- Problem 1: which memory is best for an access sequence?
- Problem 2: classifying running applications at memory speeds
- Problem 3: deciding where to place and when to move pages
- Discussion and future work

Profile memory access as a distribution

- We can represent a memory access over some time window of size k to be a random variable X ϵ $[0, 1, 2]^k$, where 0 represents no access, 1 for read, and 2 for write.
- The distribution of X is most likely non-stationary - the distribution of memory access pattern changes across time windows.

Dealing with shifting distributions

Problem: The distribution of X is most likely non-stationary. We want to be reactive without thrashing.

Solution:

- Case 1: If the frequency of distribution shift is much more frequent than the frequency permitted by cost of data movement, we should not be moving at all.
- Case 2:If the frequency is much less frequent, then we are locally stationary.
- If we detect thrashing, we can shift to Case 1.

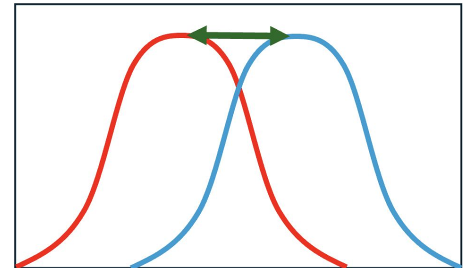
Different Memories prefers different distributions

Naturally formulates into a problem about distribution matching:

$$\operatorname{argmin} \mathrm{D}_{KL}(X^*, \Theta)$$

- X^* - realized distribution of reads and writes
- Θ - set of distributions that memories prefer - we extract this from access pattern traces.
- D_{KL} - Kullback-Leibler Divergence (this measures how “close” two distributions match each other).

Problem: Minimize state and computation required.



Connection to Compression

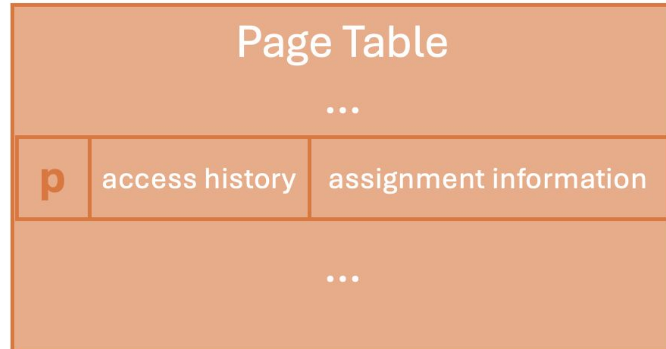
The expected code length of a universal lossless compressor is

$$\mathrm{E}[\frac{1}{Q(X)}] = \mathrm{H(P)} \text{ - } \mathrm{D}_{KL}(P, Q)$$

- We can use expected code length of compression as a proxy measure for $\boldsymbol{D_{KL}}$.
- In other words, we can train a lz78 dictionary, and use the compressibility of the actual access sequence, to measure how close the access sequence is to distributions.
- Can be done in hardware efficiently as a prefix tree lookup. The runtime scales with the depth of the prefix tree, which we can set to be a small constant.

Storing State in Page Table Entries

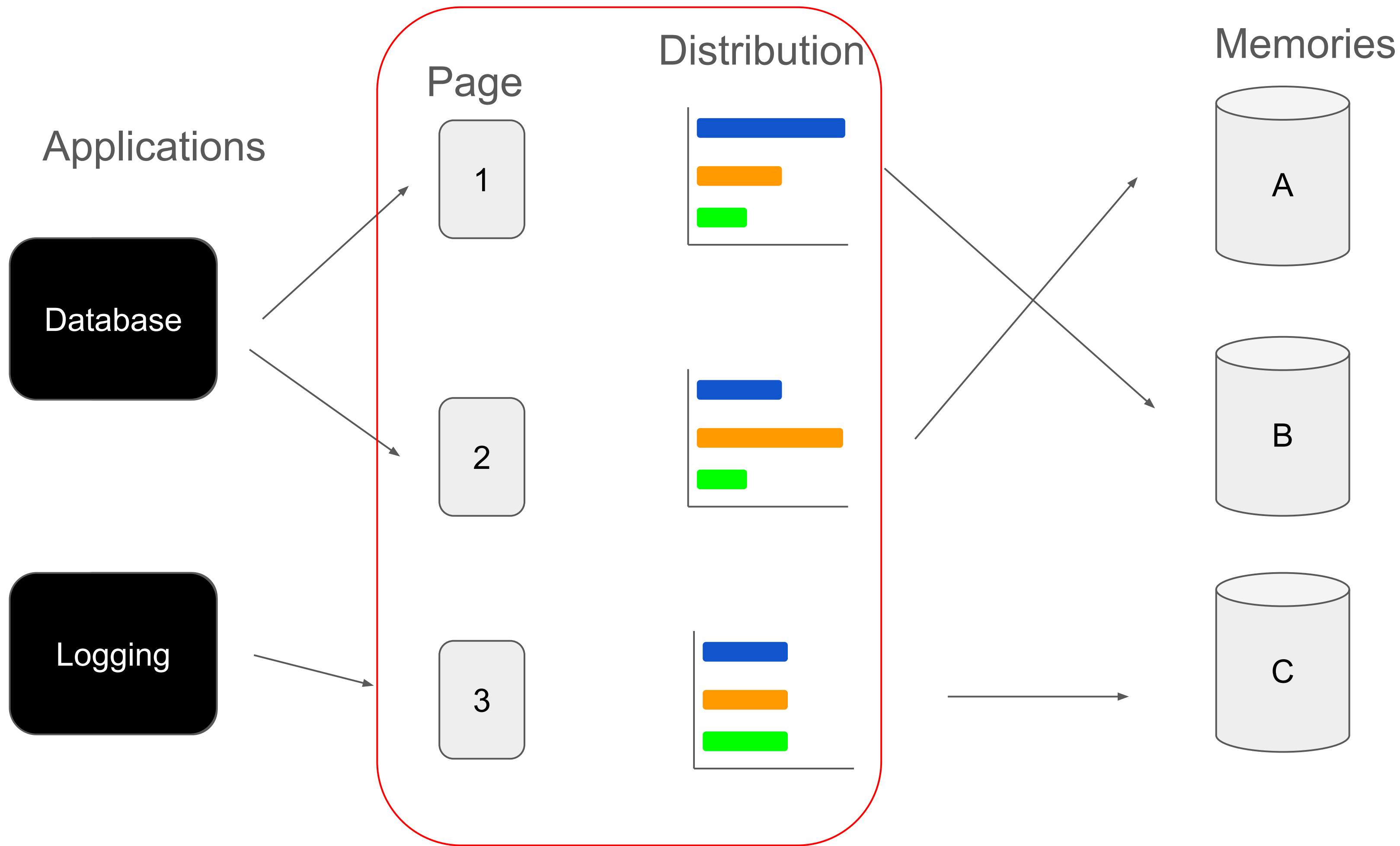
- For each compressor, we need to maintain the number of output characters for a time window, and the level in the prefix tree (6 + 2 = 8 bits, per class)
- Assignment to which memory, and which tier (4 + 4 = 8 bits)
- Under this design, we can support 7 classes of memories while doubling the PTE size.



Characterizing page distribution as “Mixtures”

- We can continuously characterize a page’s distributions as a mixture of distributions from the set of distributions, Θ, over a time window.

- We can induce a cost function that is defined over a linear combination of these mixtures
    - This, during runtime, is implemented as a lookup table.

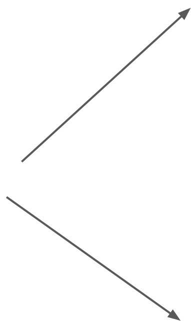

Applications
Database
Logging
Page
1
2
3
Distribution
Memories
A
B
C

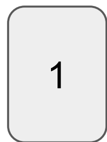
Outline

- Introduction
- Problem 1: which memory is best for an access sequence?
- Problem 2: classifying running applications at memory speeds
- Problem 3: deciding where to place and when to move pages
- Discussion and future work

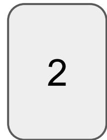
Applications
Page
Distribution
Memories

Database



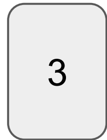


1

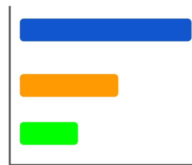


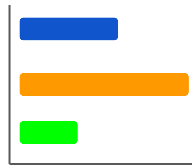
2

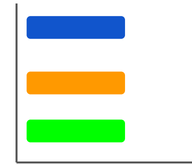
Logging

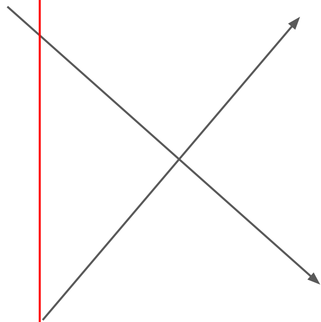


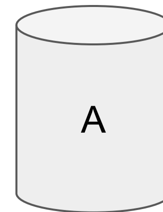
3



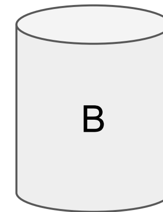




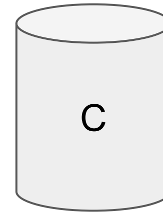




A



B



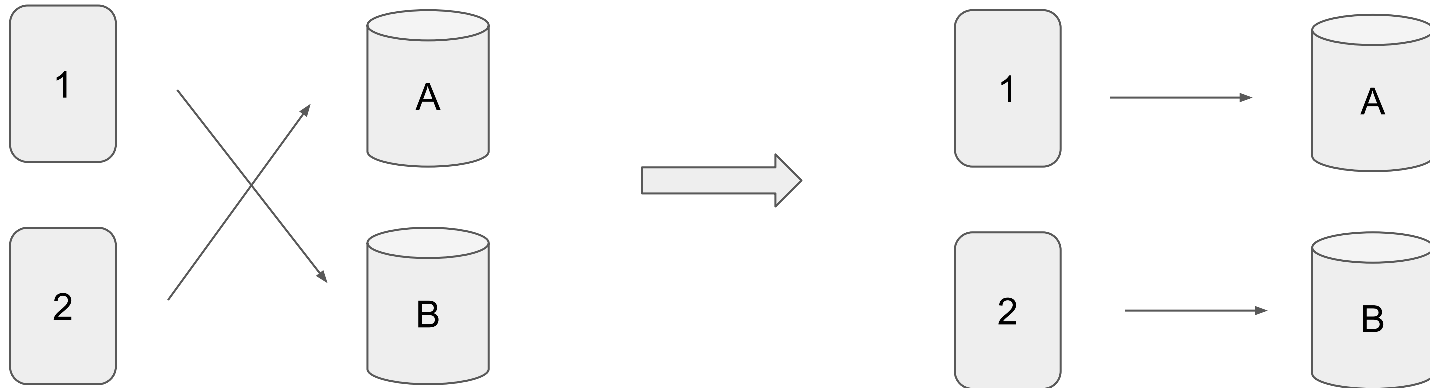
C

Step 2: Cost-aware page memory matching

- Memories have finite capacity
- This problem reduces down to maximum weighted bipartite matching
  - Solved with Edmonds-Karp (augmenting paths approximation), and Hungarian Algorithm O(N^3)
  - Too slow for runtime.

The swap operator

- In summary, “Who should I swap with such that I improve my matching set”?
- At the end of these swaps, we would arrive at a locally optimal solution for assignments.
- This operator can be done in constant time, in hardware, with bucket sort.



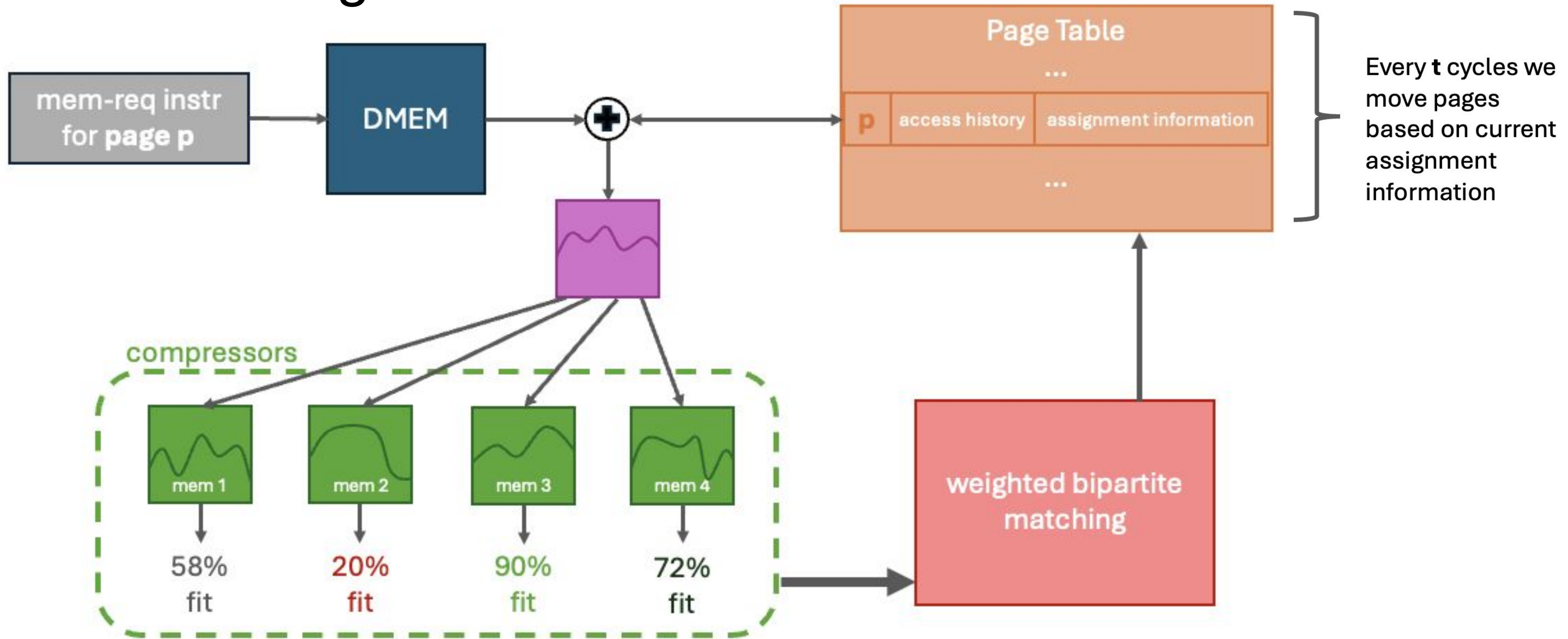
Software and OS Control

- While implementing the swap operator in hardware is strictly faster, this can be done in software as well.
- Memory movement can be communicated to a DSA engine, and this can be governed by OS.
- This swap operator is implemented as maintaining a bucket sort of each memory on how much the item prefers other buckets.
    - For 4 memories, we maintain 16 buckets
    - To check for swap candidates, we look towards the other memory and pop the top bucket (the page whose most willing to move), and check if that improves our matching

Local and Global Optimality

- We can achieve locally optimal placement by using the local swap operator.
- This local optimal is also globally optimal if the weights satisfy the property that an augmenting path is always decomposable to pairwise swaps.
- This is achieved by setting the domination property on the weights, or by forcing any augmenting path must be of length 1.
- Practically, locally optimal is often sufficient.

Hardware diagram



We are currently implementing both the compressor classification and the score-based matching on a RISC-V core simulated in Gem5

Outline

- Introduction
- Problem 1: which memory is best for an access sequence?
- Problem 2: classifying running applications at memory speeds
- Problem 3: deciding where to place and when to move pages
- Discussion and future work

Open Problems

1. Learning the set of distributions to maximally separate the access patterns
2. Using access information to design new classes of memories.
3. Where should we store state? Is page table entries the correct granularity and abstraction?

Open Problems

1. Learning the set of distributions to maximally separate the access patterns
2. Using access information to design new classes of memories.
3. Where should we store state? Is page table entries the correct granularity and abstraction?

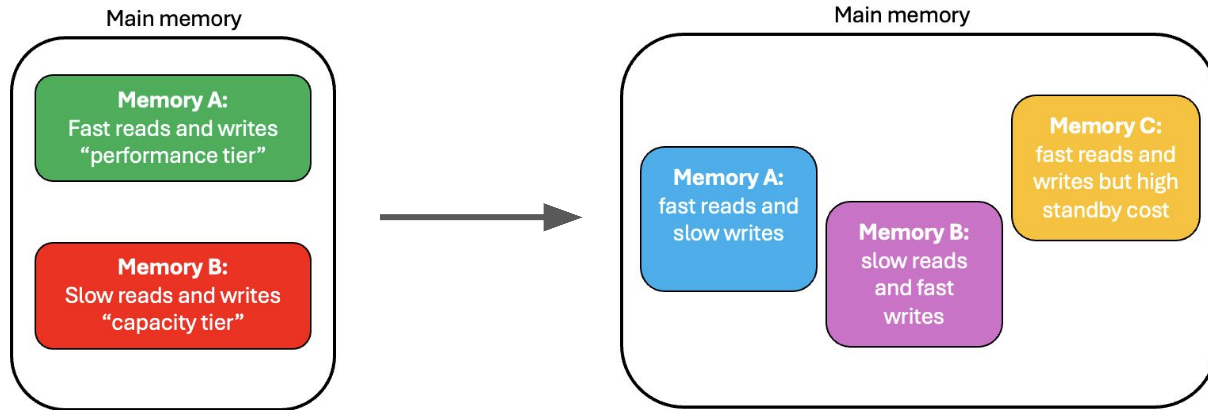
Open Problems

1. Learning the set of distributions to maximally separate the access patterns
2. Using access information to design new classes of memories.
3. Where should we store state? Is page table entries the correct granularity and abstraction?

Questions?

Differentiated Memory

- Leverage differences in memory for performance gains and cost reductions
- Given some page, which memory does it want to be in?
- Don't assume a strict hierarchy of tiers



Open Problems

1. Learning the set of distributions to maximally separate the access patterns
2. Using access information to design new classes of memories.
3. Can compiler-based optimizations help? Is spatial granular information actually useful for making better decisions?
4. Where should we store state? Is page table entries the correct granularity and abstraction?
5. The setting of window sizes is fairly important. Are there better approaches to deal with moving distribution and t+1 predictions?

Problem formulation

- We have a set of pages that need to be allocated to physical memory, each of which are accessed in some pattern of reads, writes, and standbys
- We have a set of memories each with some cost and latency for reads, writes, and standbys, as well as a fixed capacity
- **How do we dynamically allocate the pages to the memories with limited capacities**, online, given constraints on metadata storage, computation overhead, and data movement?

TODO: hardware implementation?

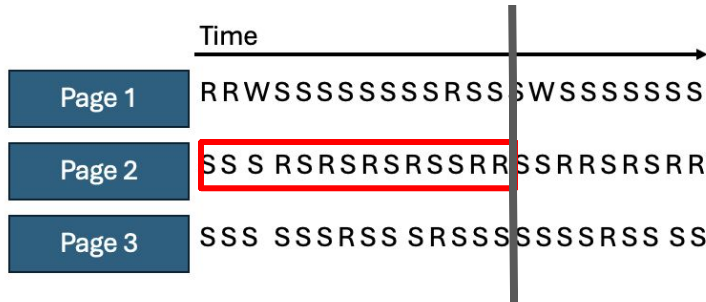
Add a graphic that shows the pages coming in, getting matched, and getting moved to a new location

Learning compressor distributions to represent memories

-
- Synthetic traces?

Implementation questions

1. Given some stored information the preceding access pattern, how do we decide where a page should be?
2. Once pages have preferences, how do we decide which page goes where and manage capacity constraints?



3. Can we do this in a lightweight way in hardware?

Differentiated access memory

- Memory wall leading to diversification
- Choice between a selection of SRAM, DRAM, gain cells, RRAM, MRAM, FeRAM and FeFETs
- Results in research questions across the stack

Allocation problem: the choice of which memory for what