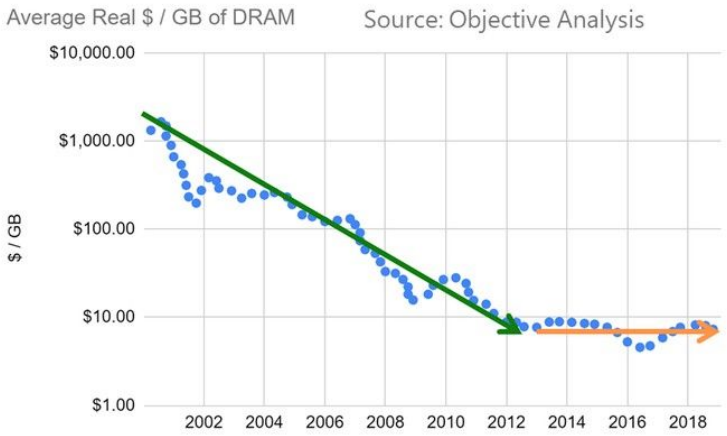
Hardware Support for Server Memory Placement

DRAM Costs Too Much in Datacenters

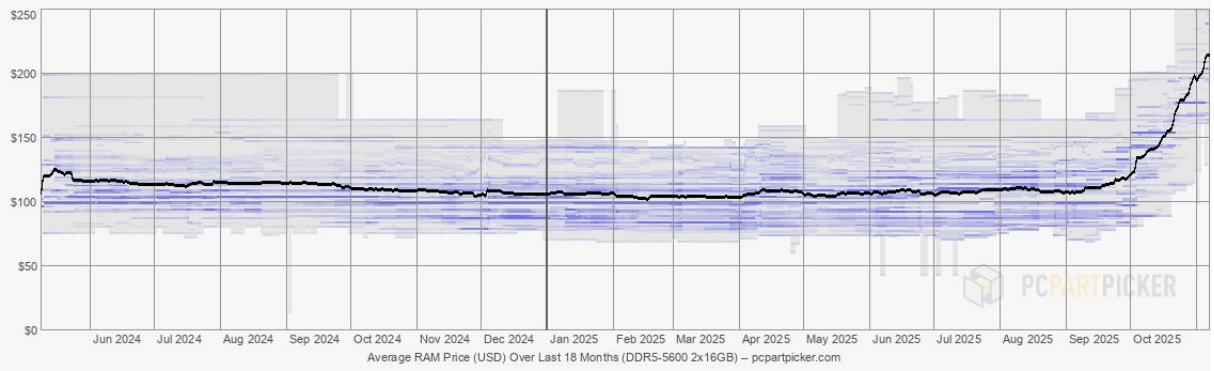
2x Intel Sapphire Rapids Server	
Component	**AI Server**
CPU	$ 1,850
8 GPU + 4 NVSwitch Baseboard	$ -
Memory	$ 3,930
Storage	$ 1,536
SmartNIC	$ 654
Chassis (Case, backplanes, cabling)	$ 395
Motherboard	$ 350
Cooling (Heatsinks+fans)	$ 275
Power Supply	$ 300
Assembly and Test	$ 495
Markup	$ 689
Total Cost	$ 10,474
DRAM BOM %	37.5%
NAND BOM %	14.7%
Memory BOM %	52.2%



- DRAM costs dominate servers
  - Microsoft reports 50% of server cost is memory (Source: TPP)

Source: https://semianalysis.com/2023/05/29/ai-server-cost-analysis-memory-is/

Recent Events: DRAM Pricing Surge



DRAM pricing can fluctuate quite significantly…

Memory Technologies: CXL and 3D Crosspoint

- The need to reduce costs has pushed research into like memory technologies like 3D Crosspoint

- CXL can attach memory extenders with pre-existing, older memories
  - Controller latency and routing are ongoing areas of research

- CXL can also be used to attach new memories
  - Non-volatile RAMs: MRAM, FRAM, RRAM, FeRAM
  - Flash: ultra-fast NAND
  - Rich, complex set of tradeoffs

Source: https://semianalysis.com/2023/05/29/ai-server-cost-analysis-memory-is/

Previous Work Has Shown Placement is Complex

Insights:

- Bandwidth pressure changes the latency sensitivity of pages (Colloid)
- Heavily-accessed memory can perform well in CXL (Beyond Hotness)
- Fixed rule-based counters don't work well (Memtis)
- Non-volatile technologies require separate read/write policies (HeMem)

Previous Work:

- HeMem (SOSP 21')
- Memtis (SOSP 23')
- TPP (ASPLOS 23')
- Colloid (SOSP 24')
- Nomad (OSDI 24')
- Beyond Hotness (OSDI 25')

Our Core Question

How can a system best use a diverse set of memories with complex tradeoffs?

… without much overhead.

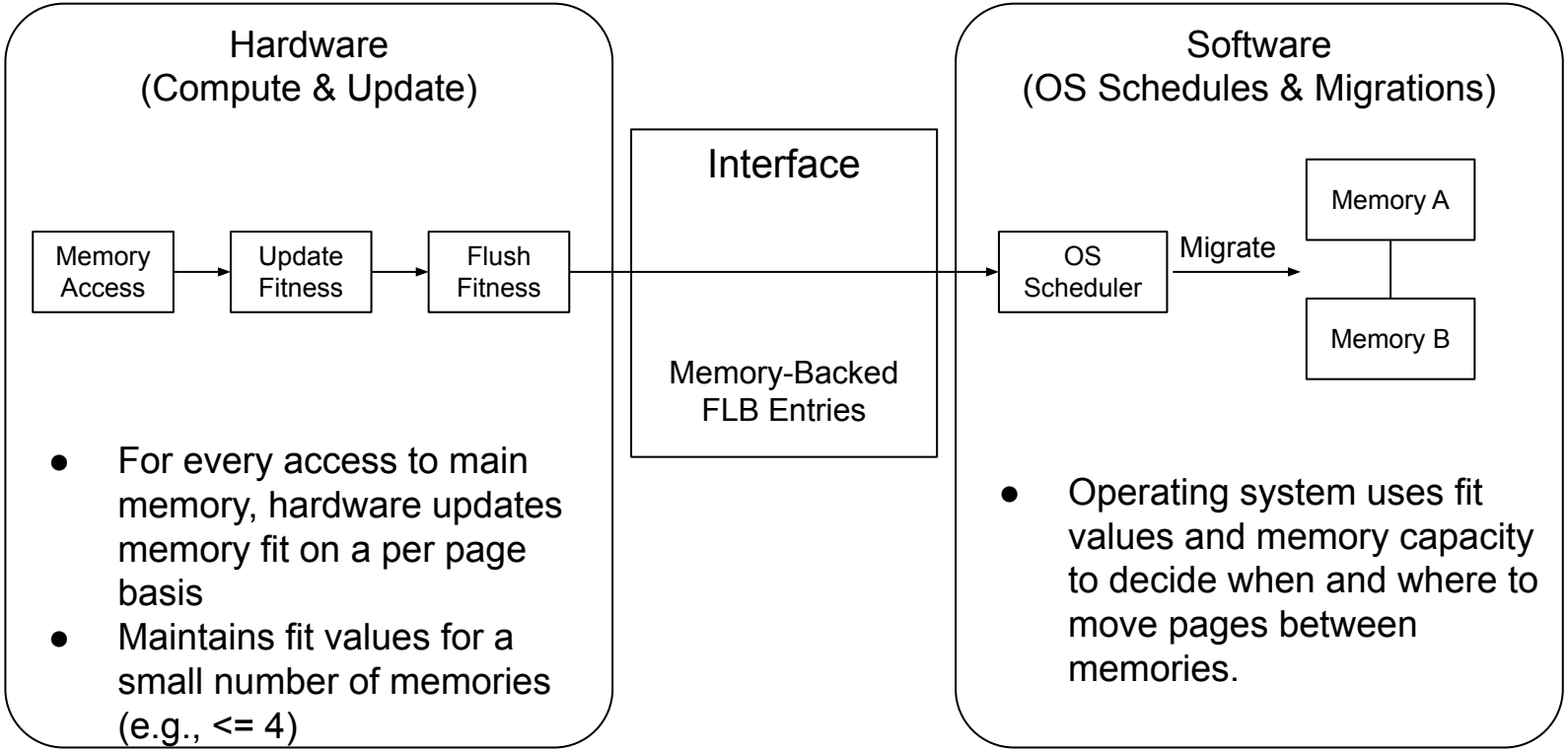
Deciding Page Placement with Memory Fit

1. "Memory fit" quantifies how well the access pattern to a page matches a memory's tradeoffs
   a. Can compare if page 1 or page 2 is better for a memory type
   b. Can compare if memory type A or type B is better for a page

2. We compute memory fit with Soft Context Tree Weighting, a novel algorithm based on information theory
   a. Estimates how far an access pattern is from "good" patterns for a memory type
   b. Build the estimator offline from "good" patterns, online estimates are cheap to compute in hardware

3. Operating system uses memory fit to decide when and how to move pages between memories, rather than "hotness" access counts or detailed performance counter data gathering

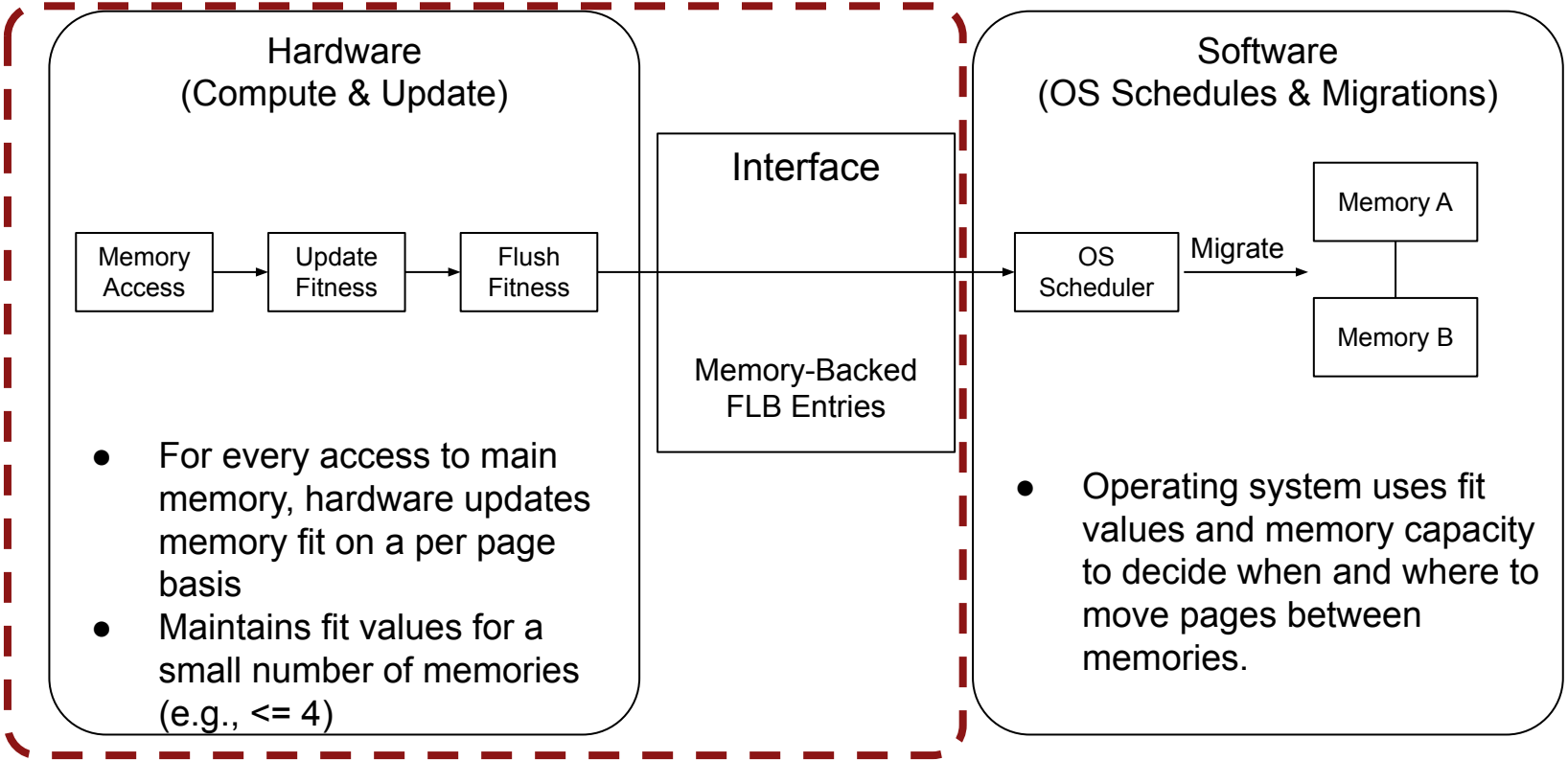
Hardware/Software Interface



- For every access to main memory, hardware updates memory fit on a per page basis
- Maintains fit values for a small number of memories (e.g., <= 4)

- Operating system uses fit values and memory capacity to decide when and where to move pages between memories.

Hardware/Software Interface

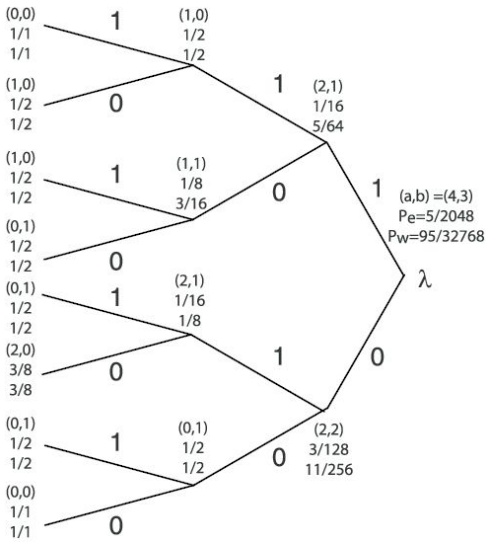


- For every access to main memory, hardware updates memory fit on a per page basis
- Maintains fit values for a small number of memories (e.g., <= 4)

- Operating system uses fit values and memory capacity to decide when and where to move pages between memories.

Model and Theory: Soft Context Tree

- Run-length access patterns for sparsity
  - Format = ([read, write], float), where float represents a normalized delay for the operator.
- Given a dataset of traces, evaluate the “advantage” of a trace for each memory
  - Construct the trees such that the cross-entropy scores reflect this advantage value
- Operationally, these trees are counting events with learned scores.

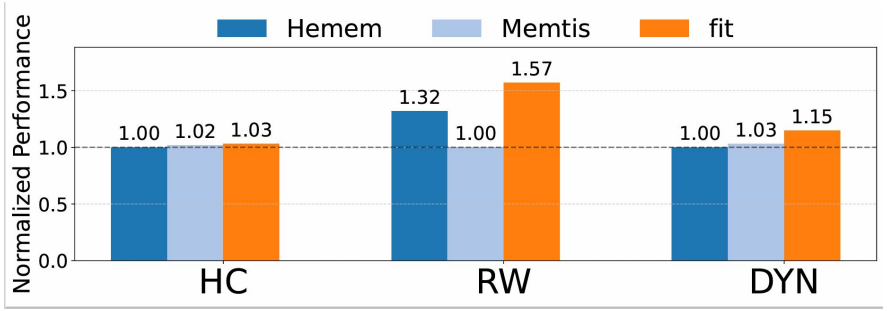


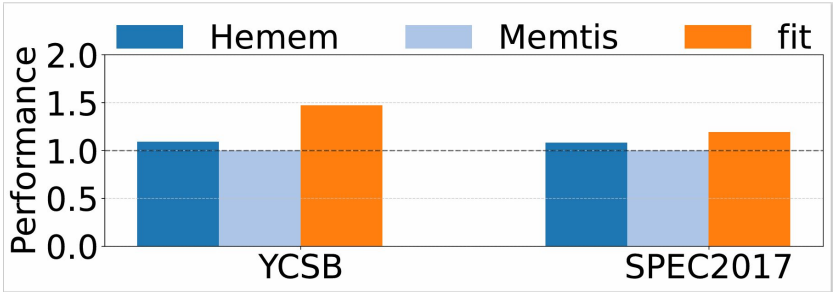
… Many more details, happy to go over them.

Simulation

- Heterogeneous Memory Simulator (DRAMSim3 backend) to simulate the performance of the model over application traces.
  - NVM & DRAM
  - Synthetic(Hot-cold, Read-Write, Dynamic)
  - SPEC2017 & YCSB mixtures

- Motivation for implementing this in hardware.

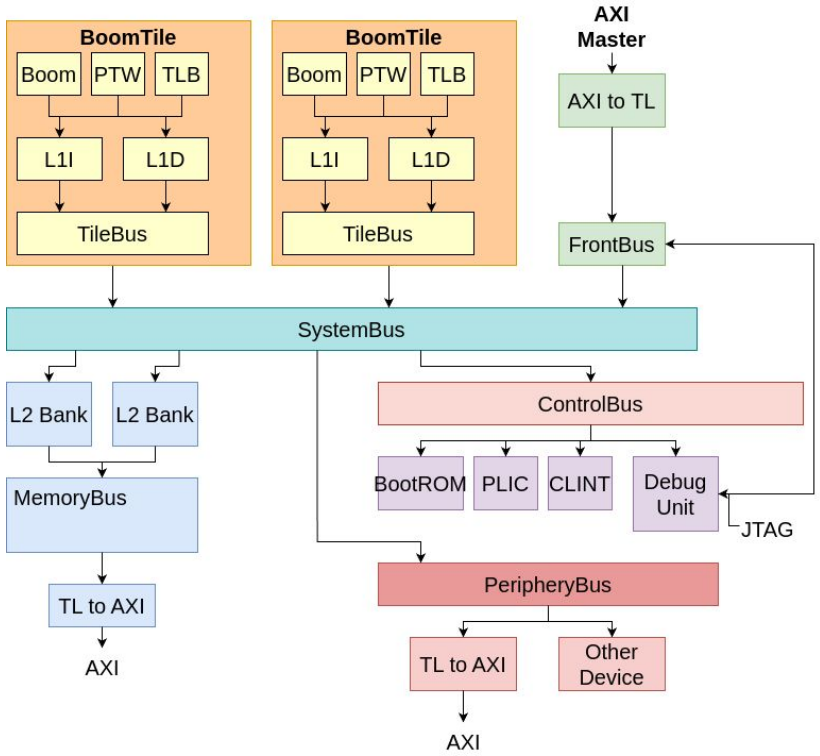




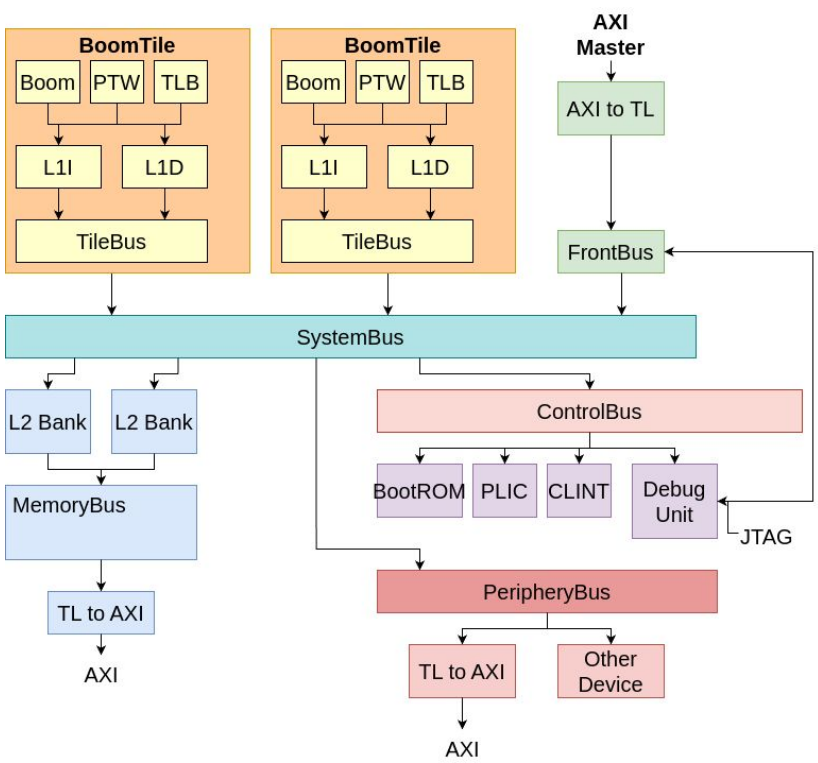
Architecture Design: Model Inference

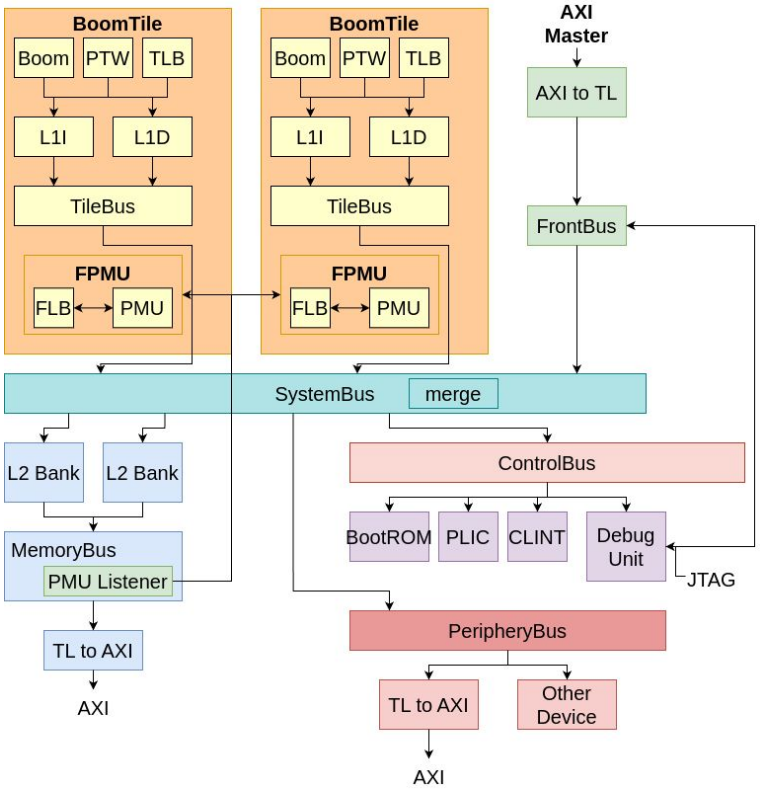
- Experiment Platform: Berkeley BOOM (v4) chip.
  - Out of order, superscalar, speculative.

- We build the inference engine of our model as a PMU (performance monitoring unit)
  - One per core



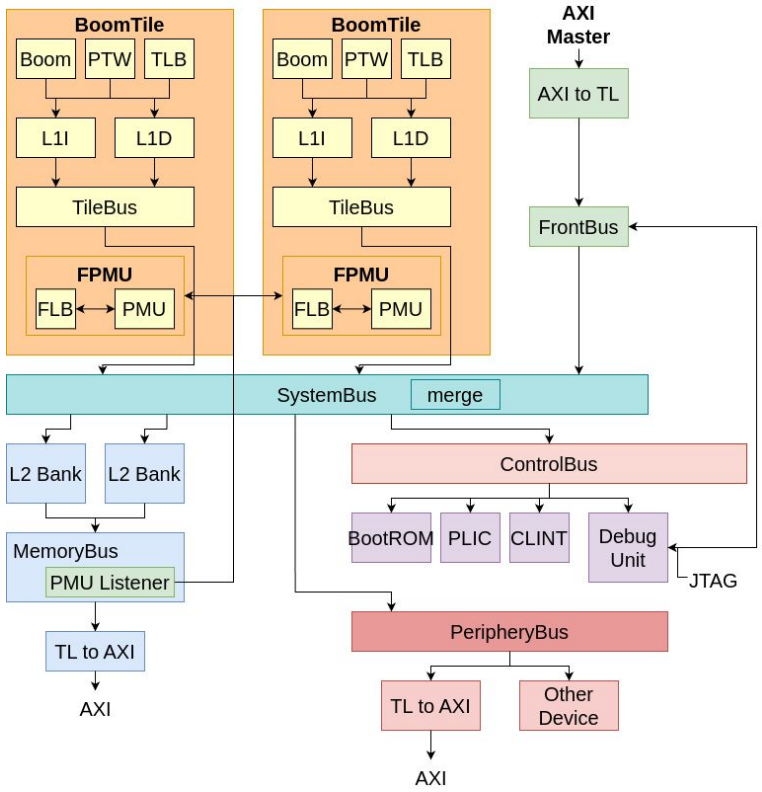
Architecture Overview





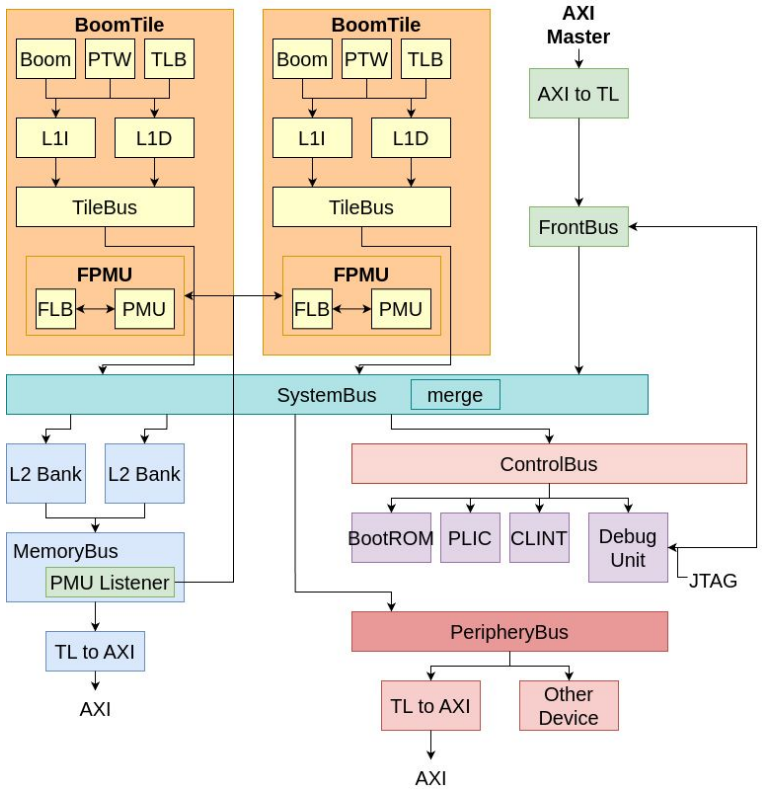
Example: Memory Access

1. Address Translation:
   a. Check the TLB
      i. Miss? Walk the Page tables.
      ii. After walking the page tables, initialize a fetch for an flb entry
2. Memory Access
   a. Walk the cache hierarchy
      i. Miss? Goes out to MemoryBus, and triggers FPMU.



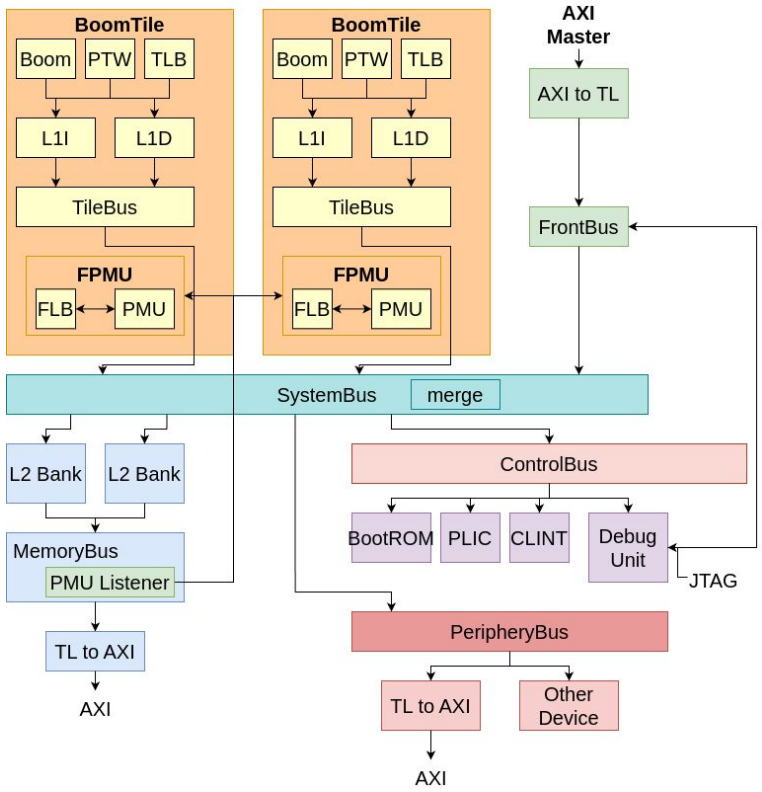
HW-SW Touchpoints

- Each entry in the FLB is indexed by the physical frame number.
  - backed by memory (readable to the OS)
- We additionally create a circular buffer to notify the OS of important score updates
- OS takes over in terms of memory scheduling policies and managing migrations.



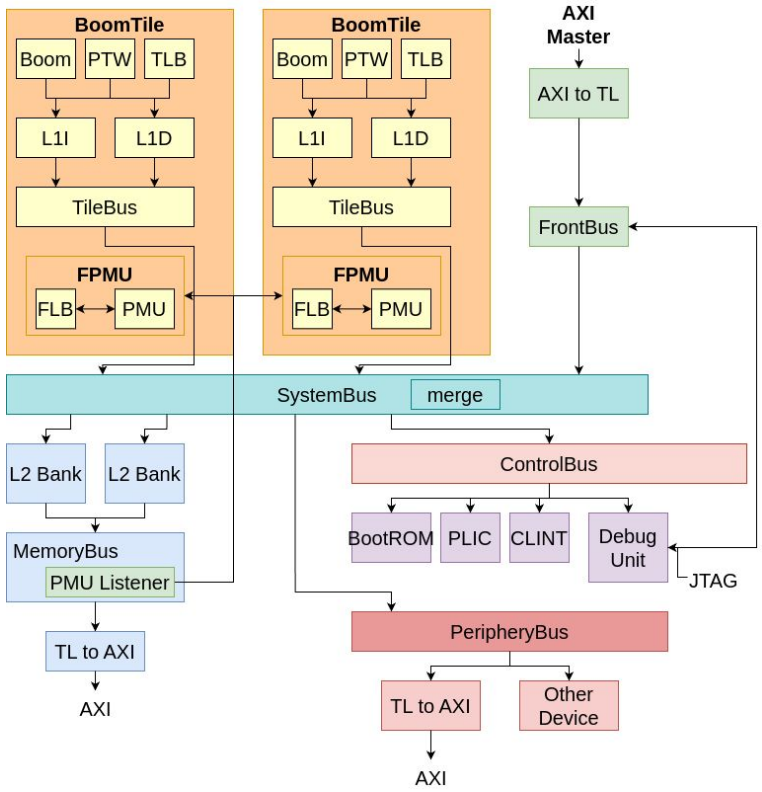
Important Questions:

- How should we manage scores consistently across multiple cores?
- How should we handle the latency in retrieving and updating fitness scores?
- How should we manage the fitness cache (FLB) such that it does not introduce any delays for the core?
- How should we minimize the footprint of this performance monitoring unit?



Consistency:

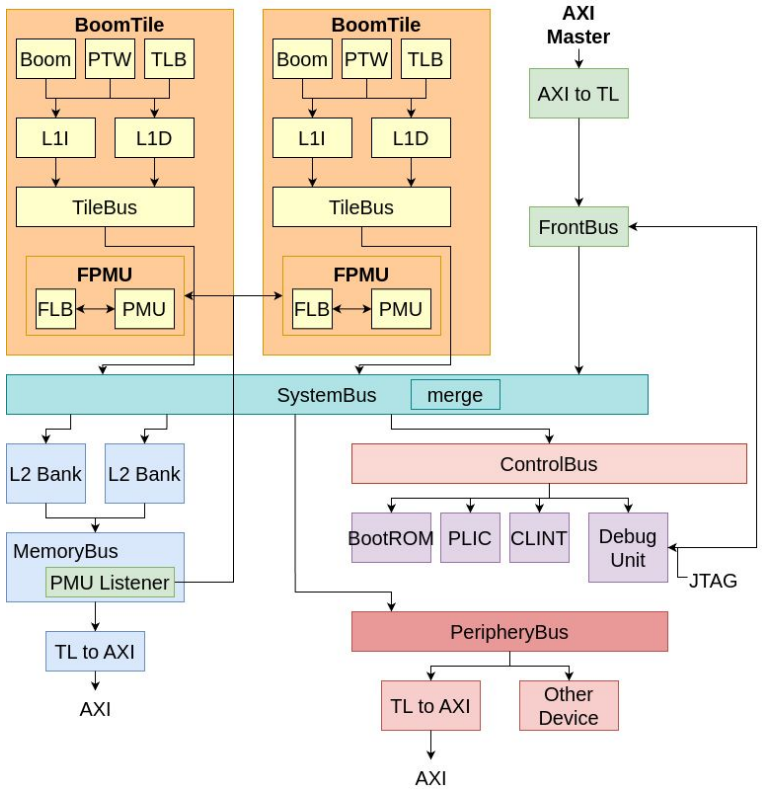
- FLB entries are PFN-indexed
  - Multiple cores could generate an update to the same physical page.
- We guarantee consistency by employing a merging circuit during writeback on the system bus.
  - This circuit could either do exception handling, race the writeback, or merge the scores.



Latency:

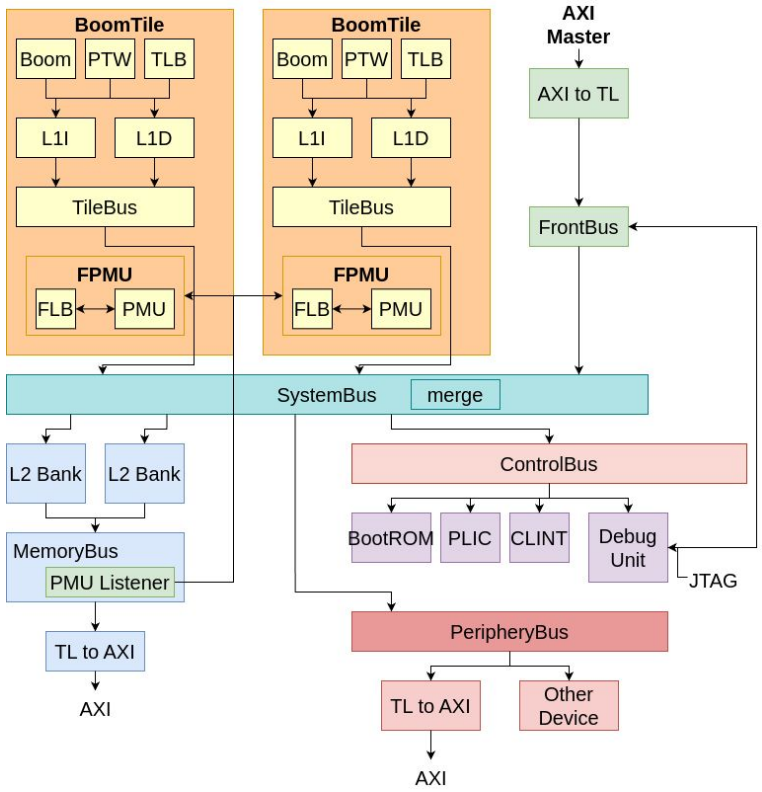
- FLB is backed in memory, writes directly to the LLC, and is written back on eviction/significant fit value change.

- Significant updates are flushed apriori to system reads, which prevents stale fitness scores from being used by OS policies.



Non-Blocking:

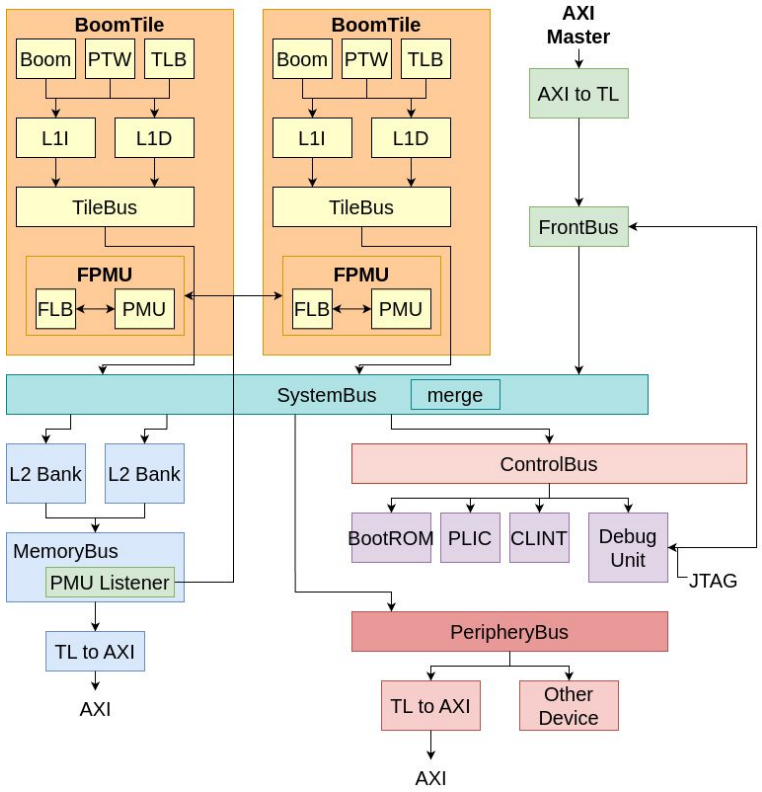
- In the scenario where there are many tlb misses, the PMU's loading buffer can get overwhelmed.
- We introduce a fast-path and slow-path in the FLB retrieval mechanism:
  - Fast-path: Initial TLB miss fetches the FLB entry.
  - Slow-path: If memory access misses the FLB, initiate the fetch, ignore the current request and repair.



Minimum Footprint:

- The PMU unit is heavily pipelined, and the computation is incredibly cheap
  - Far quicker than a typical memory fill

- Optimization target: FLB size and PMU listener signaling.



Thanks you!

Questions?