An Analytical Approach to Memory System Design

NATHAN BECKMANN

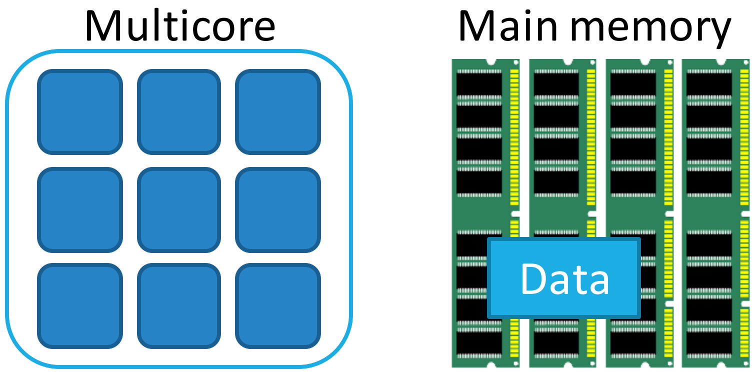
CSAIL MIT

PHD DEFENSE

17 AUGUST 2015

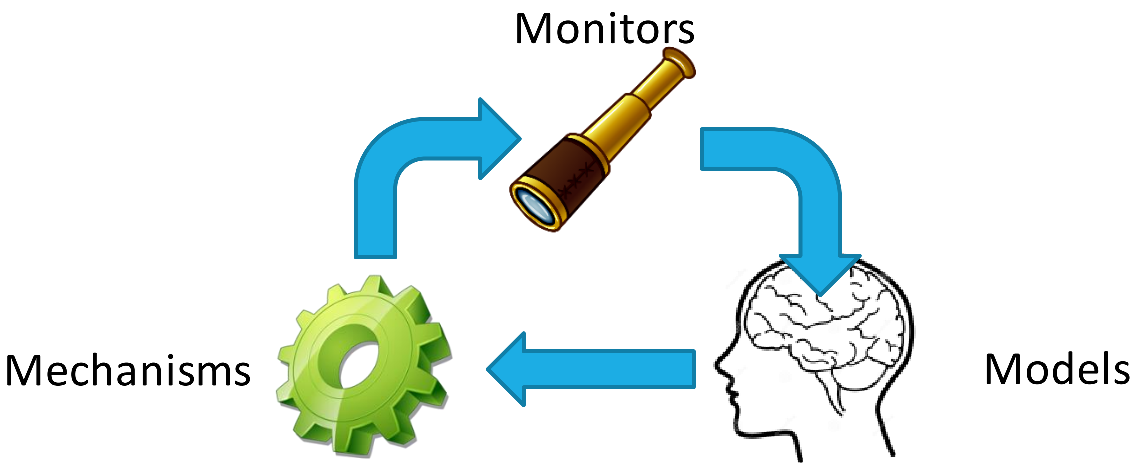
Executive Summary

Data movement is a growing problem in multicores
- DRAM 1000× energy of FP multiply-add
- Consumes ≈50% of energy and caches take >50% area
- Bigger multicores ➔ More memory requests & greater distances



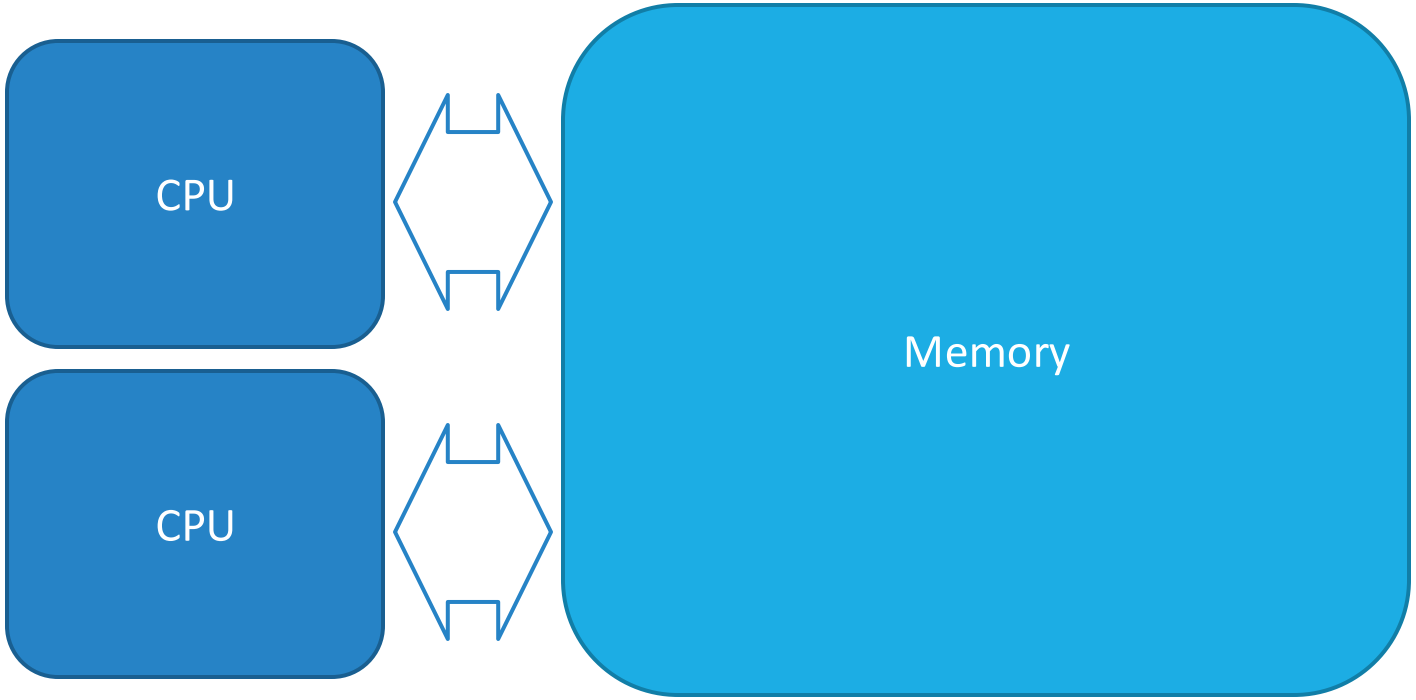
Traditional heuristics do not scale
- Multicores are diverse ➔ "common case" is far from optimal

An *analytical design* gives robust, high performance



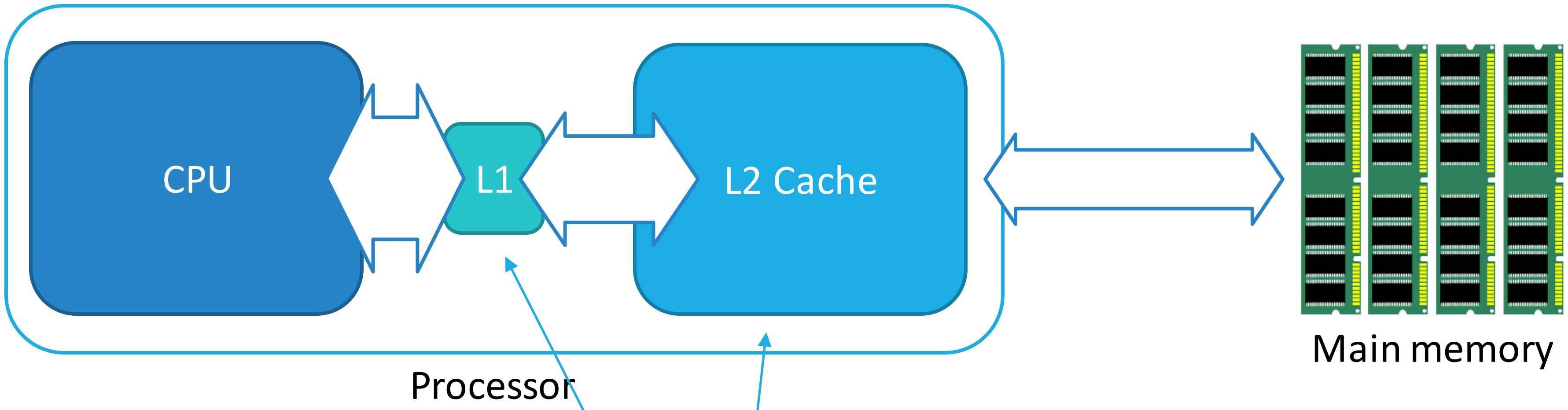
Two in-depth applications of this blueprint
- Virtual cache hierarchies
- Cache replacement

Goal: Large, Fast, Low-Energy Memory



Reality: Cache Hierarchy

Caches take >50% chip area

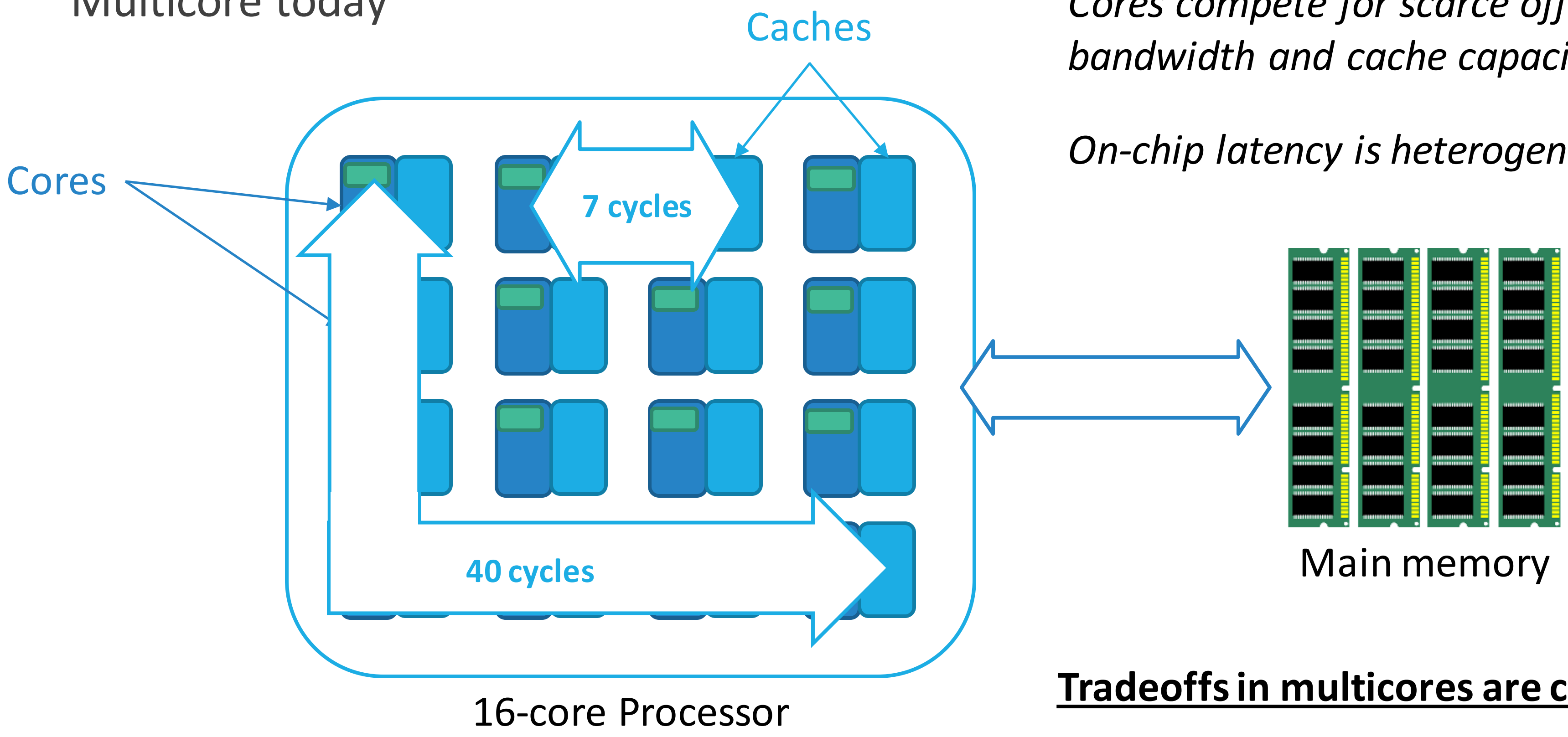


Sun UltraSPARC circa 1998: 16KB L1, 4MB L2

➔ *nearly all apps benefit from hierarchy & if not penalty is small*

Data Movement Is A Growing Problem

Multicore today



Cores compete for scarce off-chip memory bandwidth and cache capacity

On-chip latency is heterogeneous and growing

<u>Tradeoffs in multicores are complicated!</u>

Traditional Heuristics Are Insufficient

Architects try to "make common case fast"

- But programs vary greatly in their memory behavior
  - Access rate
  - Working set size
  - Reference locality
- What is the "common case"?

Design to particular benchmarks:

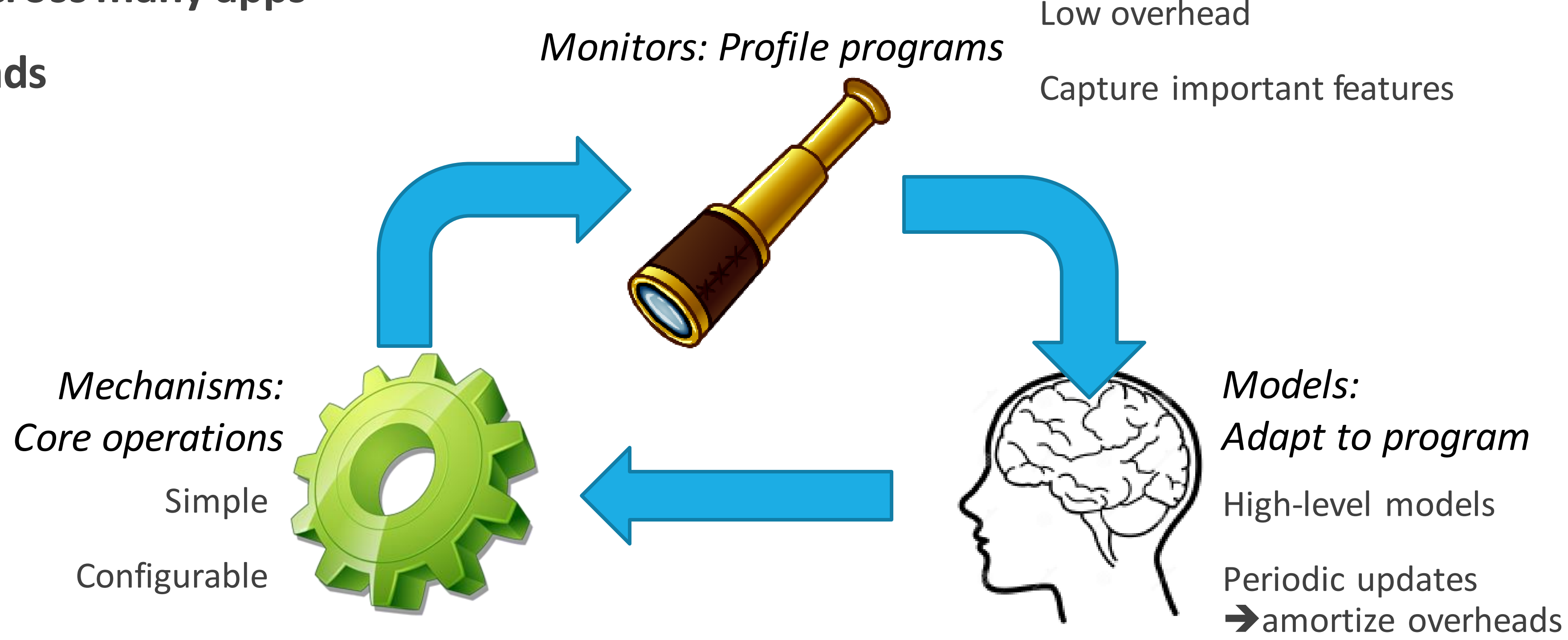
- Observe behavior for apps that perform poorly
- Find techniques that improve performance
- ➔ *State-of-the-art techniques do not perform well across all benchmarks*

Our Analytical Approach

+ Performance of app-specific hardware design

+ Generality across many apps

+ Low overheads



Thesis Contributions

This talk!

Virtual cache hierarchies

- Place data across banks to reduce data movement *[Jigsaw, PACT'13]*
- Schedule threads to reduce contention for banks *[CDCS, HPCA'15]*
- Exploit memory heterogeneity to build virtual hierarchies *[Jenga, Under submission]*

Cache replacement

- Provable convex cache performance *[Talus, HPCA'15]*
- Replacement by economic value added *[EVA, Under submission]*

Cache modeling

- An accurate model for high-performance replacement policies *[Under submission]*
- Explicit, closed-form solutions of hit rate using differential equations *[In preparation]*

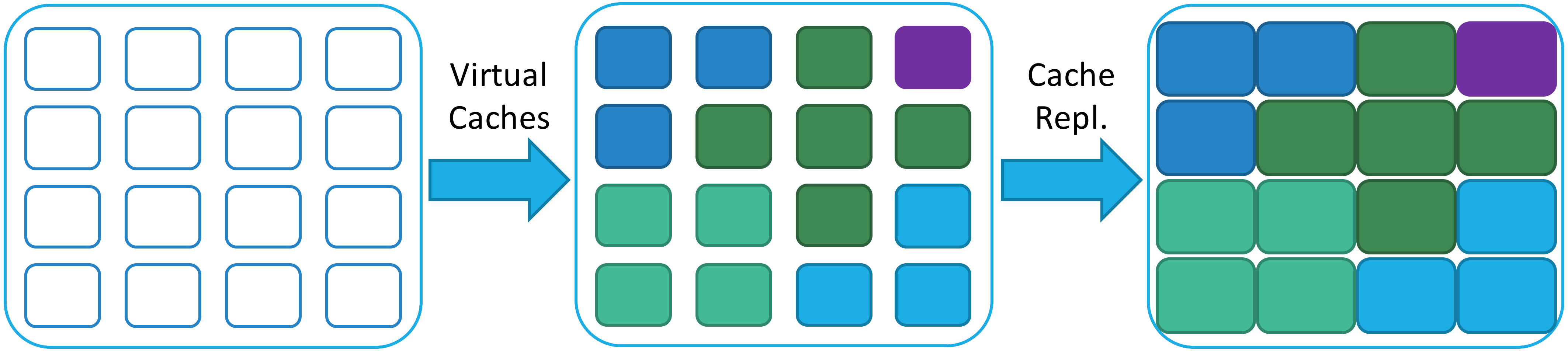
An Analytical Memory Design

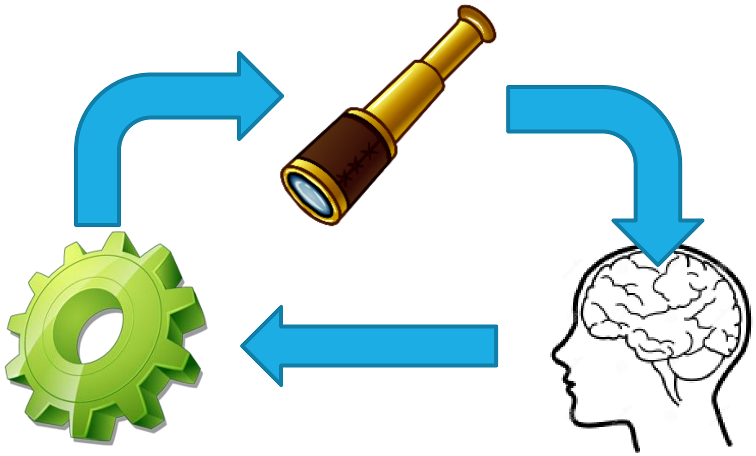
Virtual caches place data in banks to fit working set near where it is used

- Reduce data movement energy by >40%

Analytical cache replacement makes better use of cache space

- Increase effective capacity by 10% over state-of-the-art





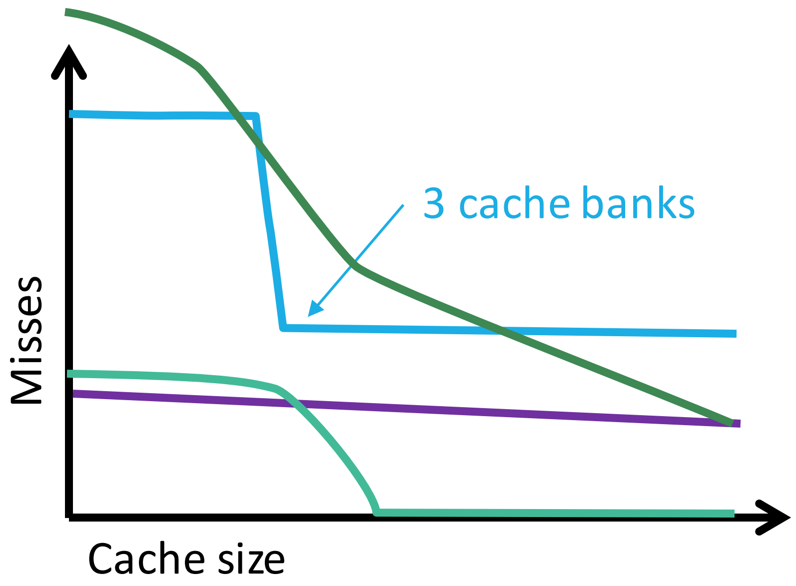
Our analytical approach is a blueprint for future robust, scalable memory systems

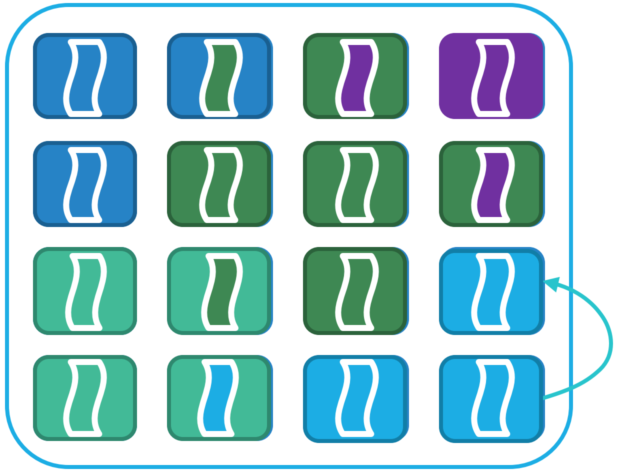
Virtual Cache Hierarchies

Virtual Caches: Jigsaw

Key idea: Schedule data across cache banks

- Control both capacity and placement
- ➔ Minimize cache misses & access latency





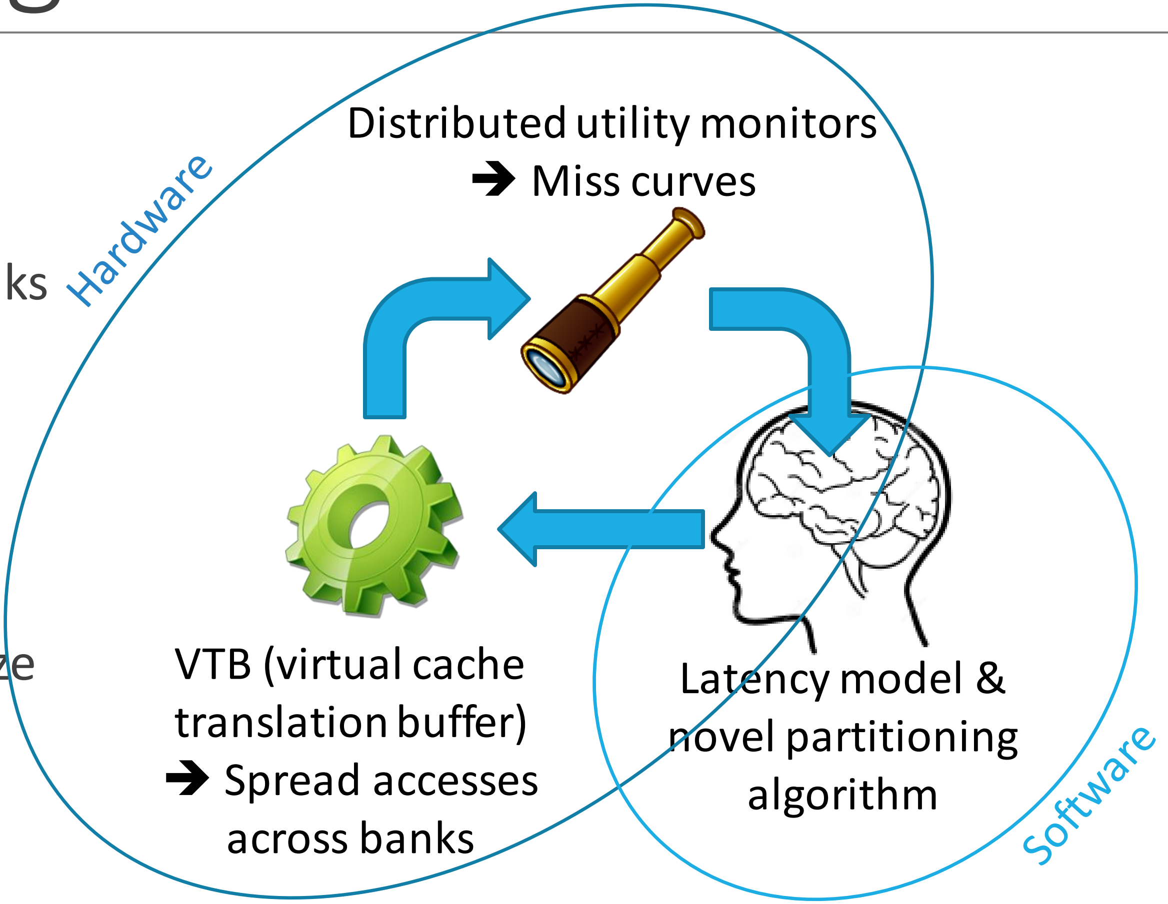
Virtual Caches: Jigsaw

Single-level virtual caches (VCs)

VCs combine partitions of physical banks

Map data to VCs

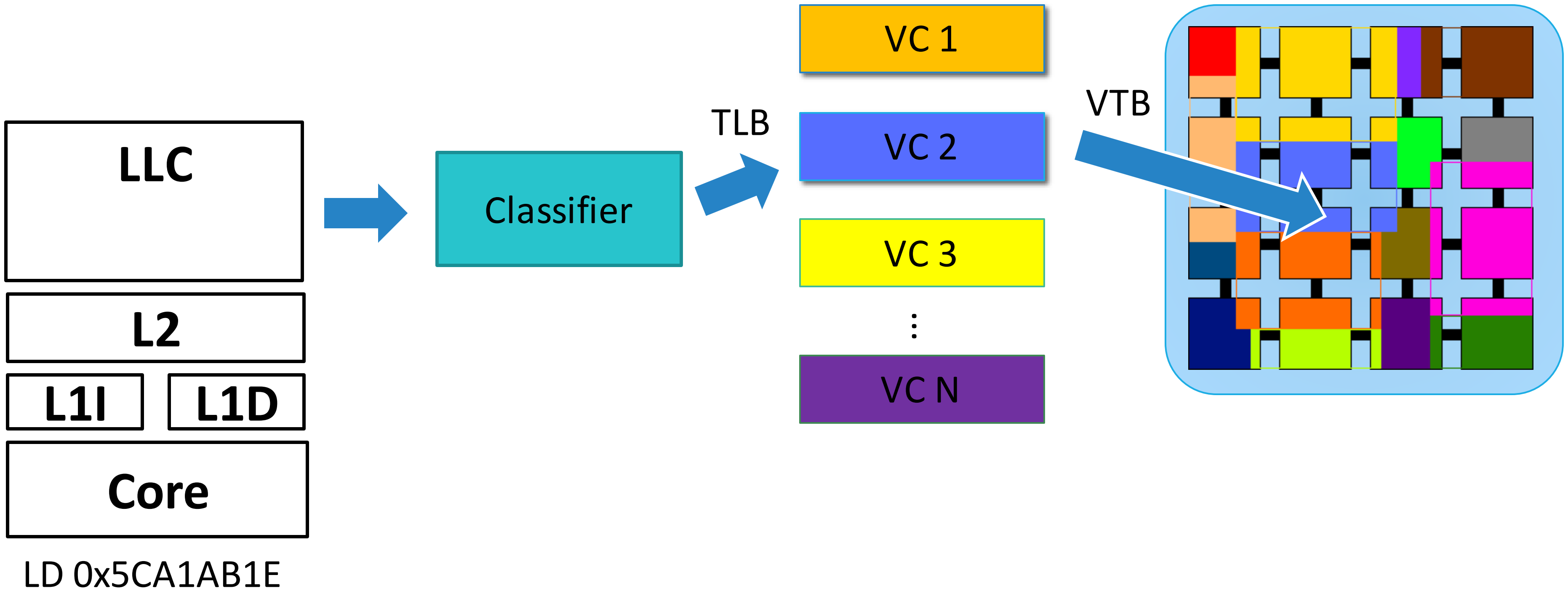
Periodically reconfigure VCs to minimize data movement



Operation: Access

Data does not move ➔ **single-lookup**

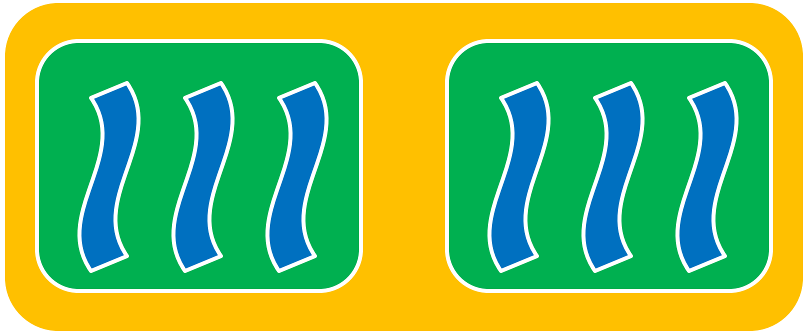
Data ➔ virtual caches, so **no LLC coherence required**



Data Classification

Jigsaw classifies data based on access pattern

- Thread, Process, and Global



- 6 thread VCs
- 2 process VCs
- 1 global VC

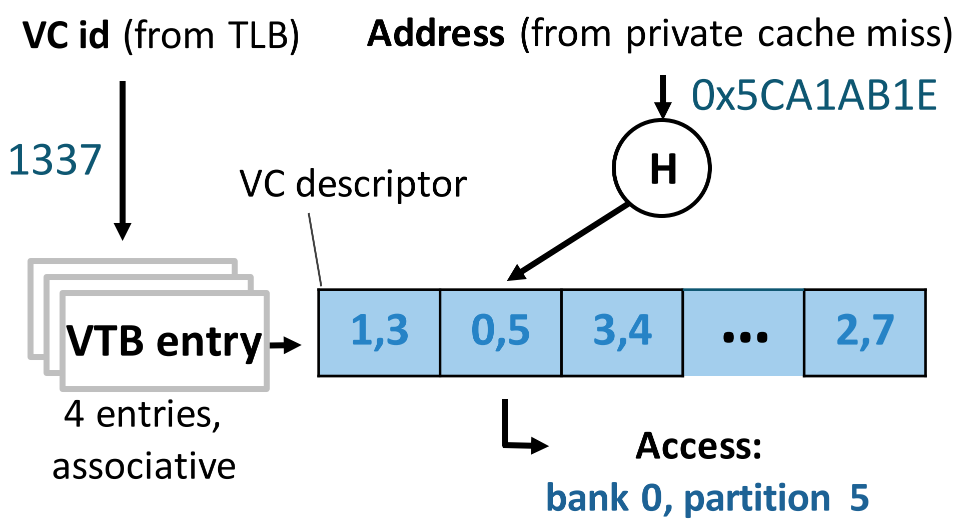
Data lazily re-classified on TLB miss

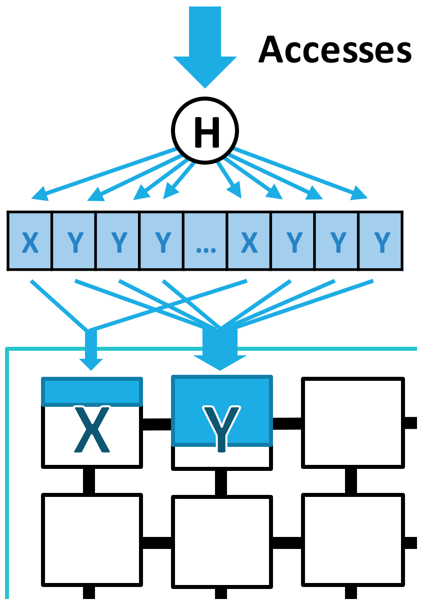
- Negligible overhead

Virtual Cache Translation Buffer

The VTB gives the *unique location* of an address in the LLC

Configurable map: {Address, VC} ➔ {Bank, partition}



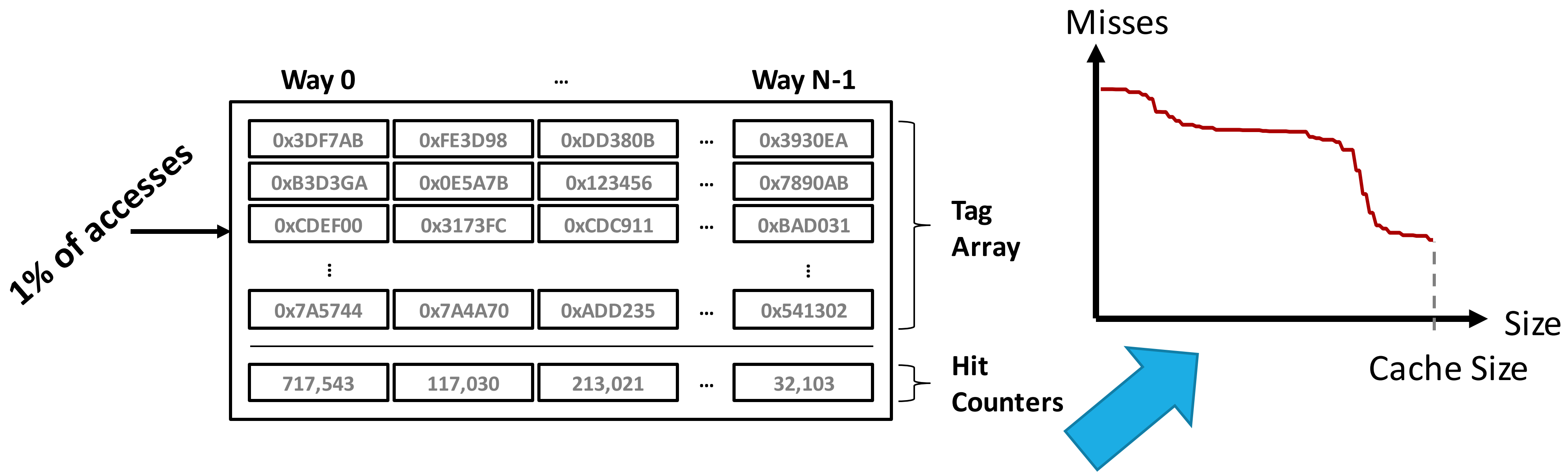


Monitoring

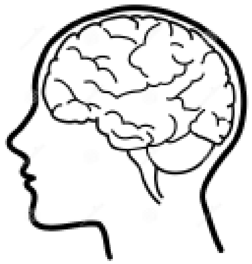
Software requires miss curves for each VC

Jigsaw adds *geometric monitors* (GMONs) distributed across tiles

Geometric monitors monitor very large caches at low overhead--$O(\log size)$; see thesis

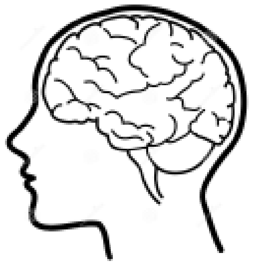


Configuration



1. Total memory latency = cache access latency + cache misses
2. Size virtual caches to minimize latency
3. Place virtual caches
   - Solve each independently for simplicity starting from optimistic assumptions

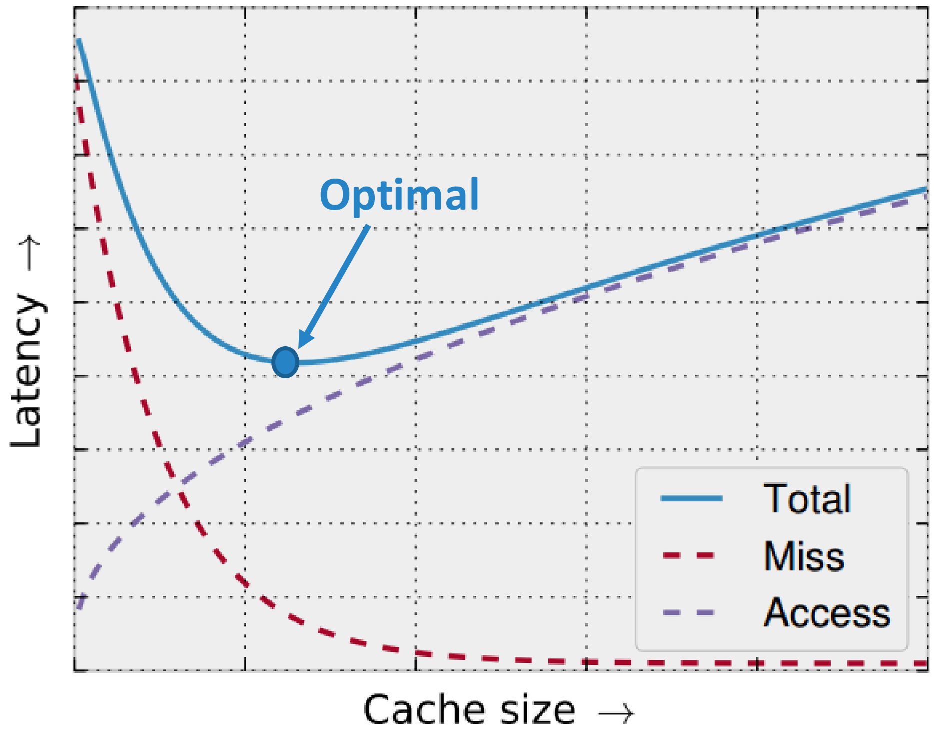
Modeling Virtual Cache Latency



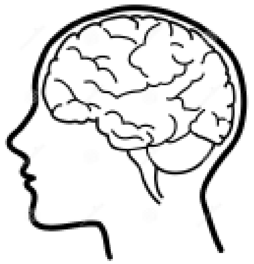
LLC miss latency decrease with larger size

Cache access latency increases with larger size

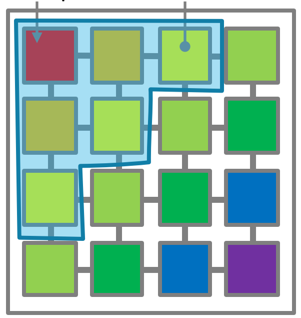
Optimal allocation balances these

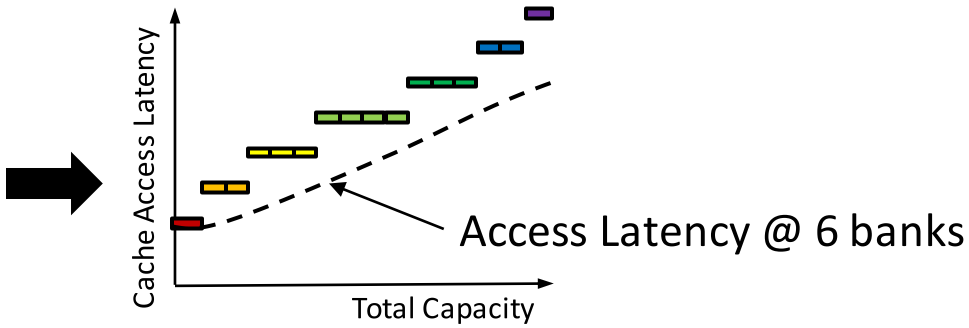


Modeling Access Latency



Construct *access latency curve* using network distance

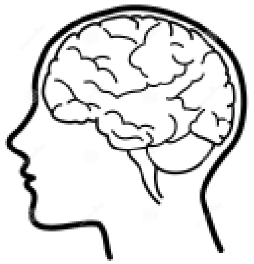




The average value of this curve gives the access latency

- E.g., hierarchy with a VC of 6 banks

Configuration: Sizing



Partitioning problem: Divide cache capacity *S* among *P* partitions to minimize objective

- Given curves $\{f_p\}$, choose sizes $\{s_p\}$ such that $0 \le s_p$ & $\sum_p s_p \le S$ to minimize $\sum_p f_p(s_p)$
- Traditional partitioning minimizes misses
- Jigsaw minimizes total latency (including on-chip latency)

NP-complete in general

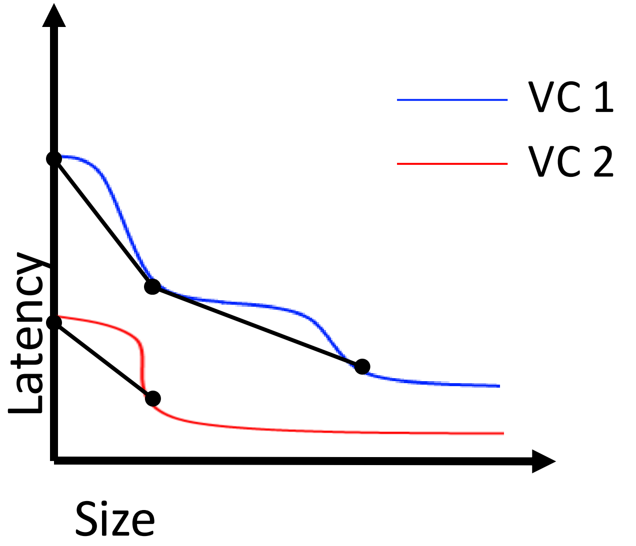
Prior approaches:

- ~~Hill climbing is fast, but gets stuck in local optima~~
- Lookahead *[UCP, MICRO'06]* produces good outcomes, but scales quadratically

Can we scale Lookahead?

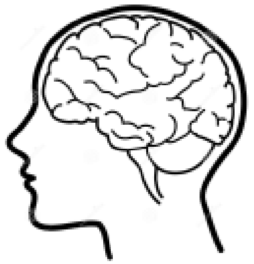
Configuration: Peekahead

Lookahead scans miss curves to find allocations that maximize hits / capacity



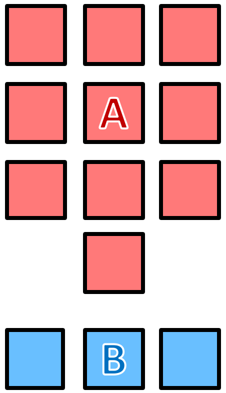
Observation: Lookahead only ever allocates along *convex hull* of the objective curve

Convex hulls can be found in linear time
- Some details and corner cases; see thesis



Configuration: Placement

<u>Allocations</u>

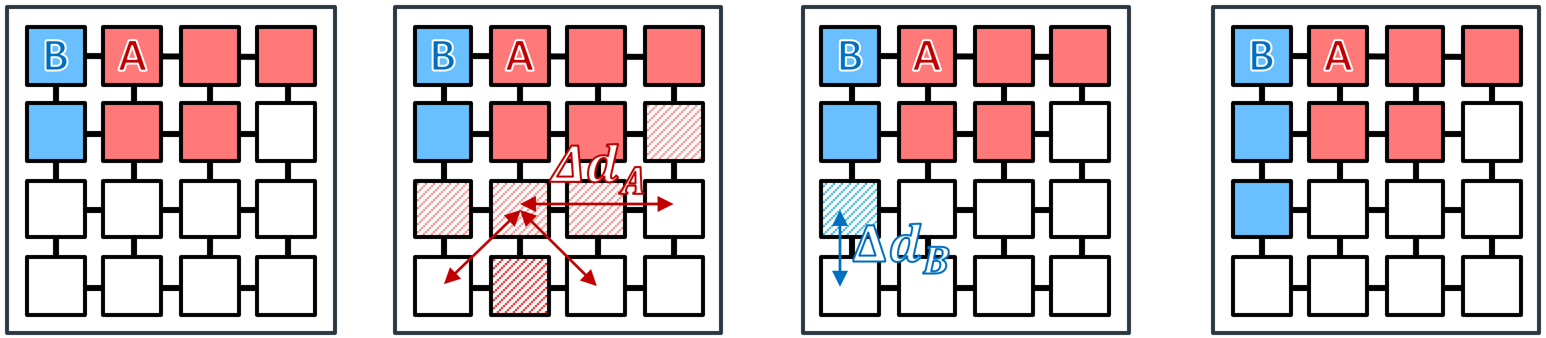


Intensity $=$ Accs/Size

100 accs to A per K instr $\Rightarrow$
$I_A = 100$ accs/10 banks
$= 10$ accs/bank

75 accs to B per K instr $\Rightarrow$
$I_B = 75$ accs/3 banks
$= 25$ accs/bank

Start $\Rightarrow$ Decide $\Rightarrow$ Place



A's harm:
$\Delta d_A = 2$ hops
$\Delta L_A = I_A \cdot \Delta d_A = 20$

B's harm:
$\Delta d_B = 1$ hop
$\Delta L_B = I_B \cdot \Delta d_B = 25$

Place B
$(\Delta L_B > \Delta L_A)$

Virtual Cache Overheads

Cache partitioning adds 8KB / bank

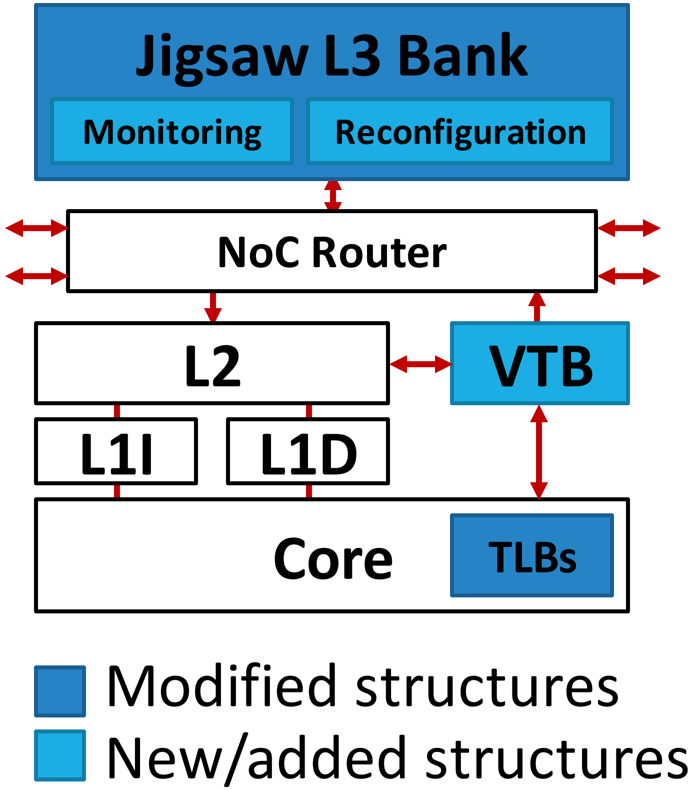
VTBs add 1KB / core

Monitors add 8KB / core

Total: 17KB / tile ➔ 3% area overhead over caches

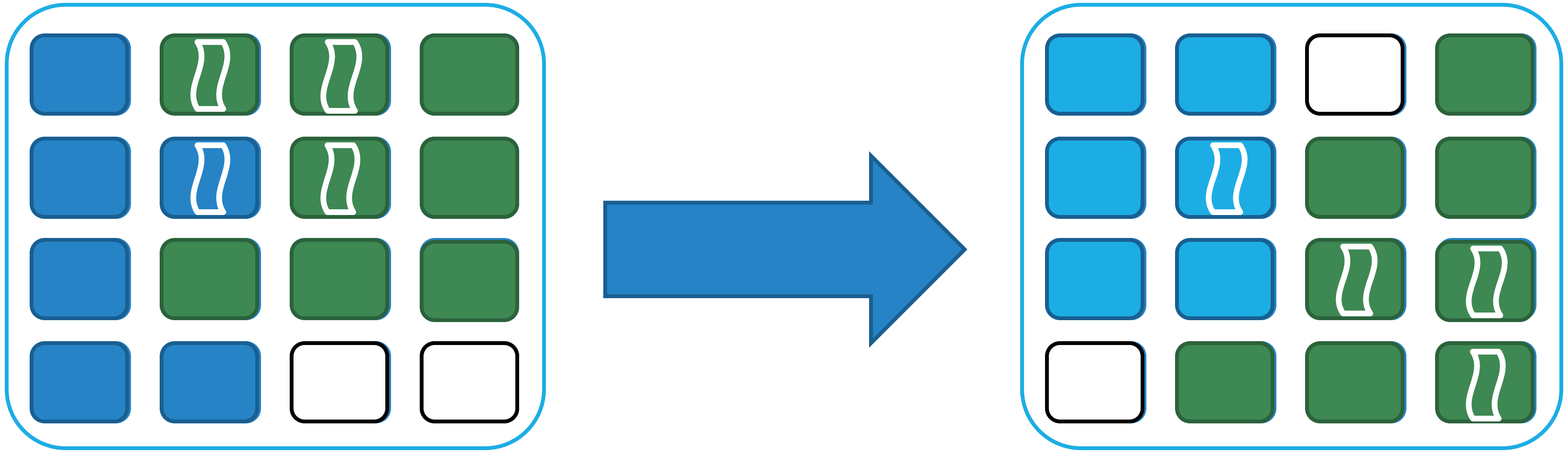
Negligible energy overheads

OS runtime takes ≈0.2% of system cycles



Contention-Aware Thread Scheduling

Virtual caches introduce *capacity contention* between threads



Poor thread placement ➔ unnecessary data movement

Schedule threads to cluster around shared VCs, separate private VCs

Evaluation of Virtual Caches

Execution-driven simulation using zsim *[Zsim, ISCA'13]*

Workloads:

- 64-core, random mixes of SPECCPU2006
- See thesis for other system sizes, multithreaded programs, etc

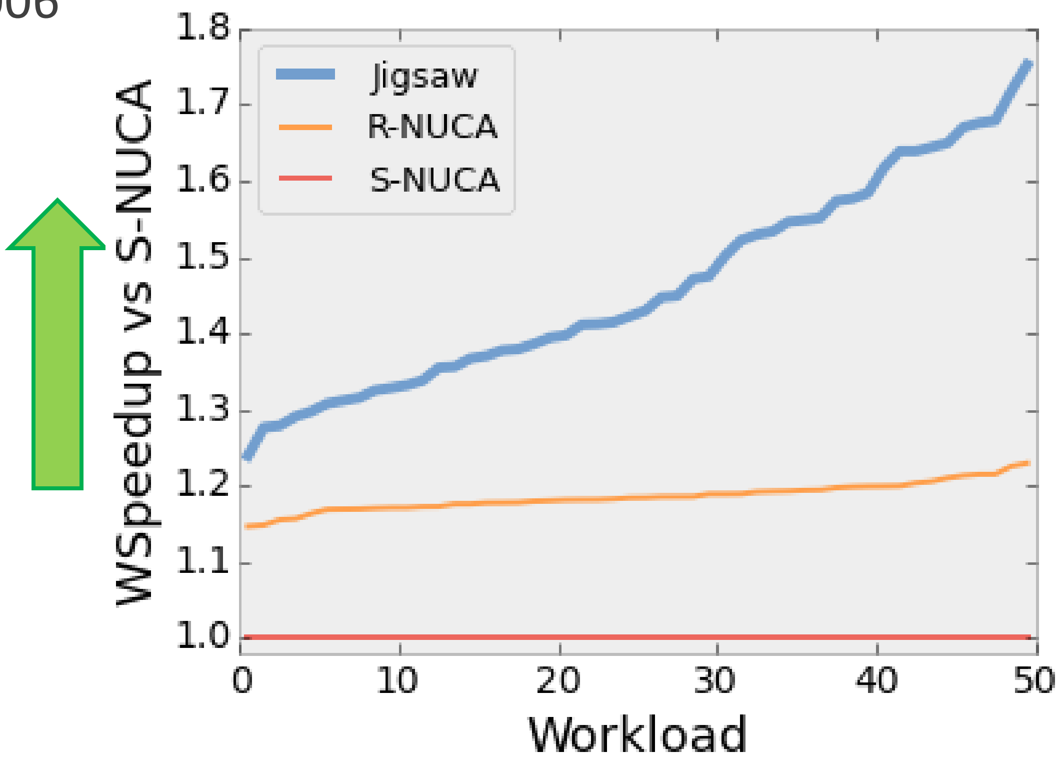
Cache organizations

- "S-NUCA" – conventional shared cache with lines spread across banks (baseline)
- "R-NUCA" – similar classification as Jigsaw but fixed placement heuristics
- Jigsaw

Evaluation: Performance

64-core multiprogrammed mixes of SPECCPU2006

Weighted speedup vs. S-NUCA



Jigsaw achieves best performance

- Up to 75% improved w. speedup
- Gmean +46% w. speedup
  - vs. up to 23% / gmean 19% for R-NUCA

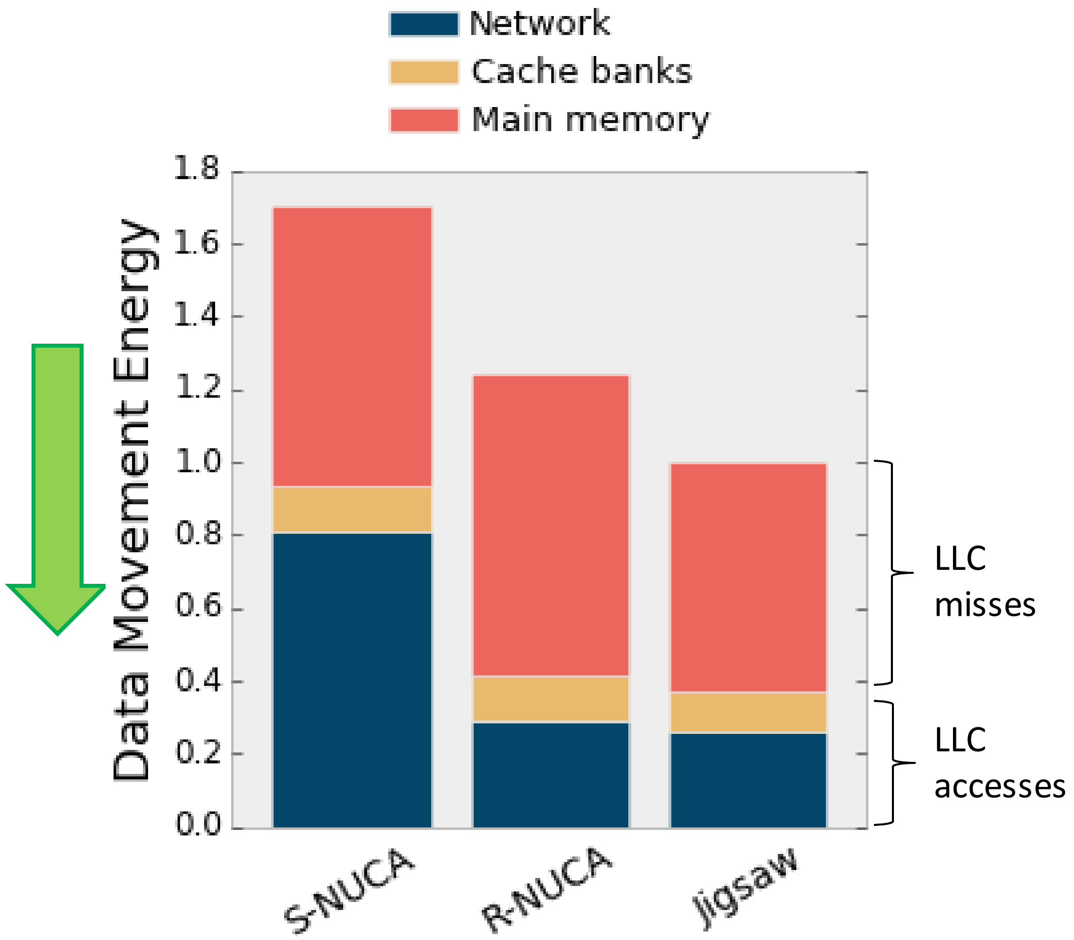
Evaluation: Energy Breakdown

64-core multiprogrammed mixes of SPECCPU2006

Breakdown data movement energy
- Normalized to Jigsaw

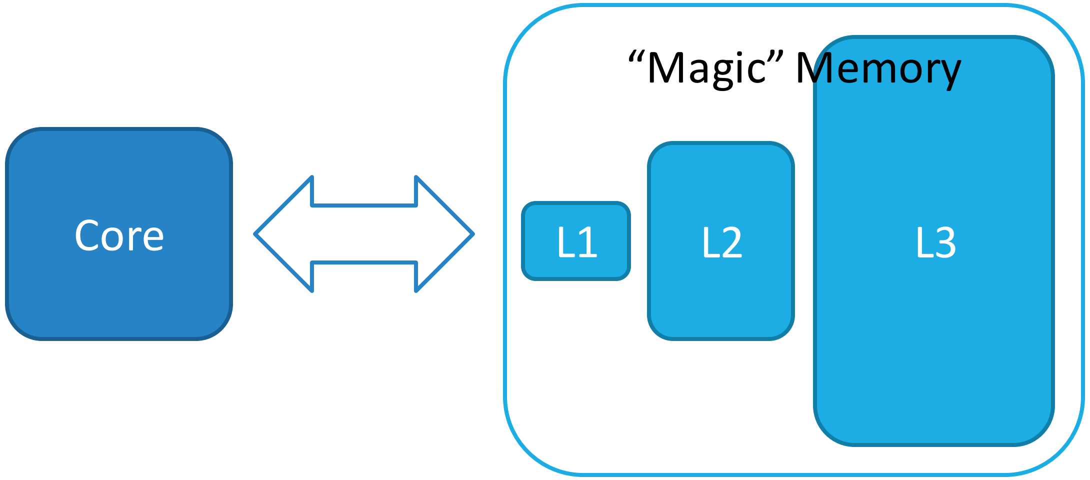
R-NUCA reduces network distance but adds LLC misses
- Placement heuristics limit capacity

Jigsaw reduces data movement from both on-chip network and LLC misses
- Saves 70% vs. S-NUCA
- Saves 20% vs. R-NUCA



Virtual Cache Hierarchies: Jenga

Hierarchies provide the illusion of a single large & fast memory



Virtual caches handle this!

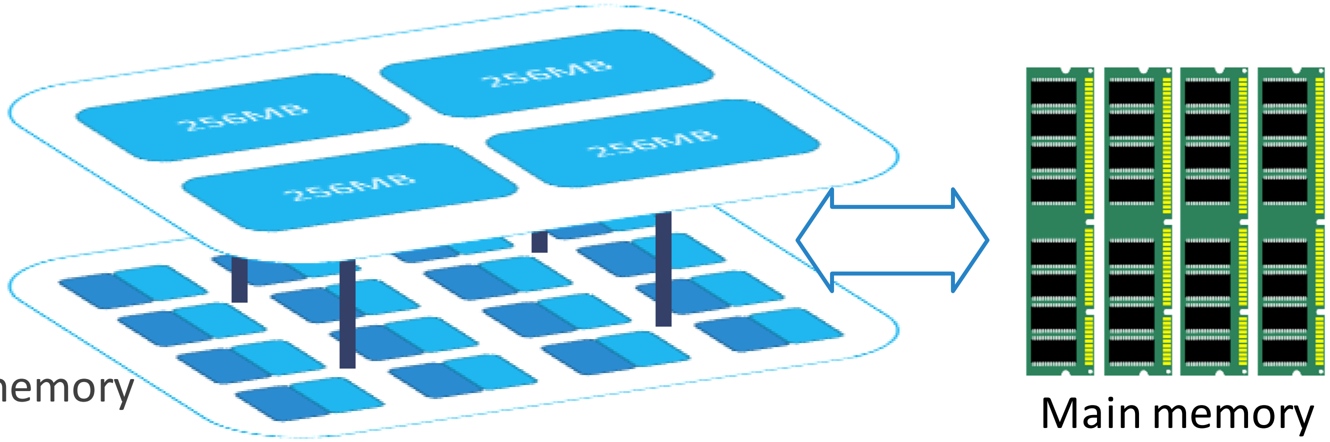
Hierarchy is useful for two reasons:

1. ~~Adapt across diverse apps~~
   - ~~Working [illegible]allest level~~
   - ~~Effi[illegible] with big differen[illegible] across levels~~
2. Adapt to diversity *within one app*
   - E.g., different data structures
   - Single-level VCs insufficient

Heterogeneous Memories

New memory technologies give 100s MB capacity nearby

- 3D-stacked DRAM
- eDRAM w/ interposers
- PCM, memristors, etc



Deepen the cache hierarchy?

- Significant energy & bandwidth vs. main memory

…But comparable latency to main memory

- ➔ Large penalty for apps that don't need it!

How do we design memory systems to harness these heterogeneous memories?

Virtual Cache Hierarchies

Expose heterogeneity to software

Build *virtual cache hierarchies (VHs)* out of heterogeneous cache banks

- Multi-level hierarchies only when beneficial
- Use caches best suited to access pattern
  - Small working sets ➔ local on-chip bank
  - Large working sets ➔ stacked DRAM vault

VHs simply do not use stacked DRAM when not beneficial

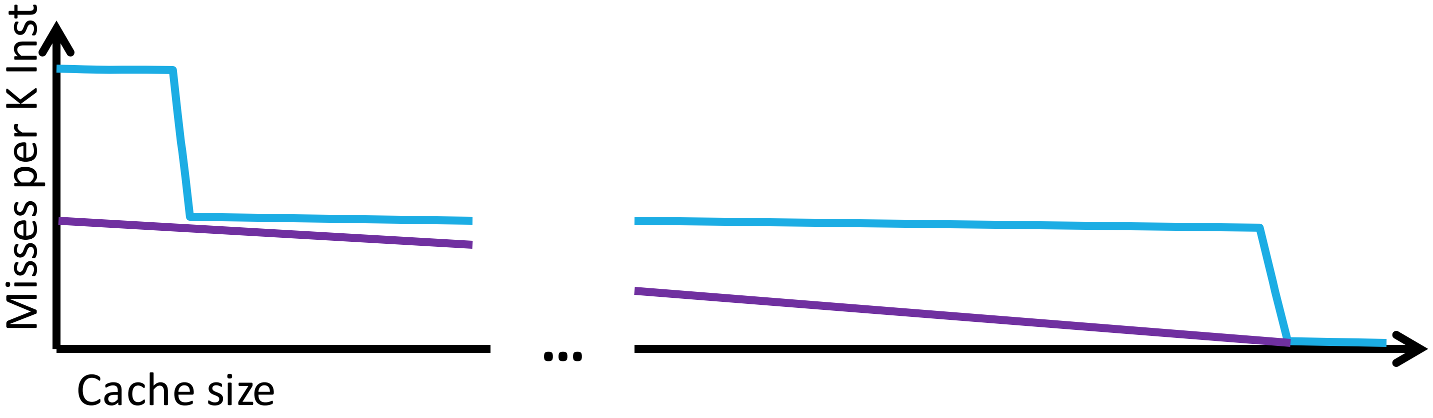
- Reduces bandwidth and energy *[BEAR, ISCA'15]*

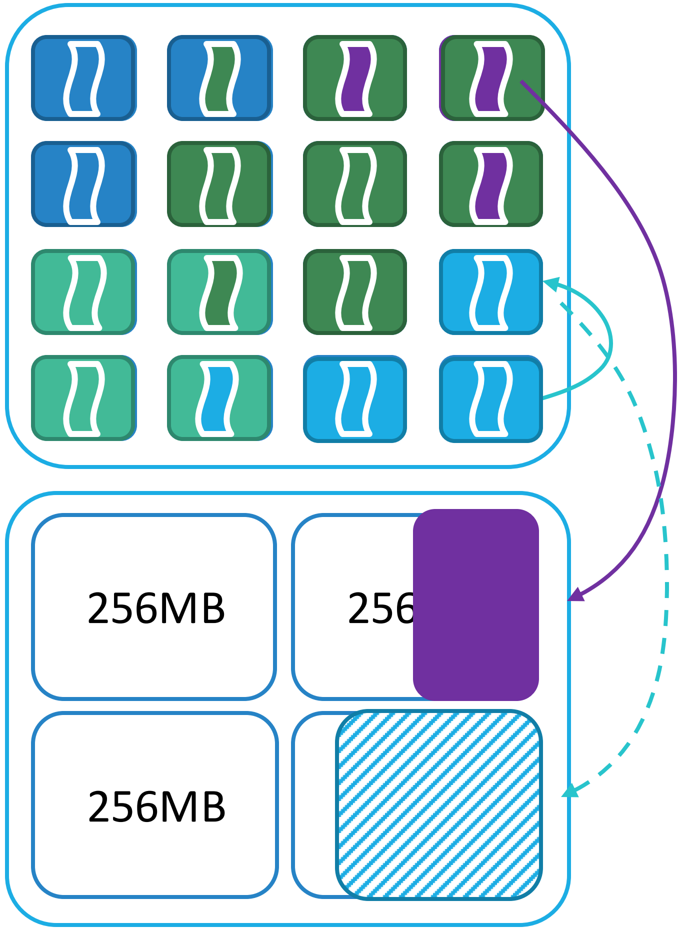
Virtual Cache Hierarchies

Chain multiple VCs to make virtual cache hierarchies

Give applications the *hierarchy they want*

- Use hierarchy only when it is beneficial
- ➔ Efficient integration of new memories (e.g., stacked DRAM)

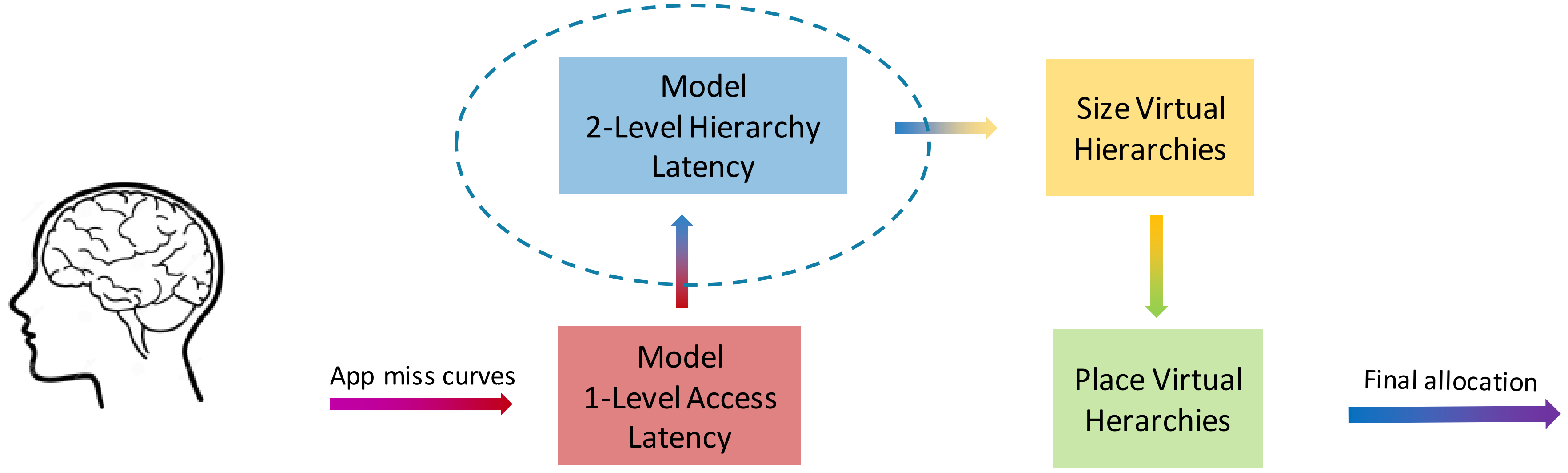




Jenga Operation

Jenga adds another level to the VTB and doesn't change monitors

Model access latency of a two-level cache hierarchy

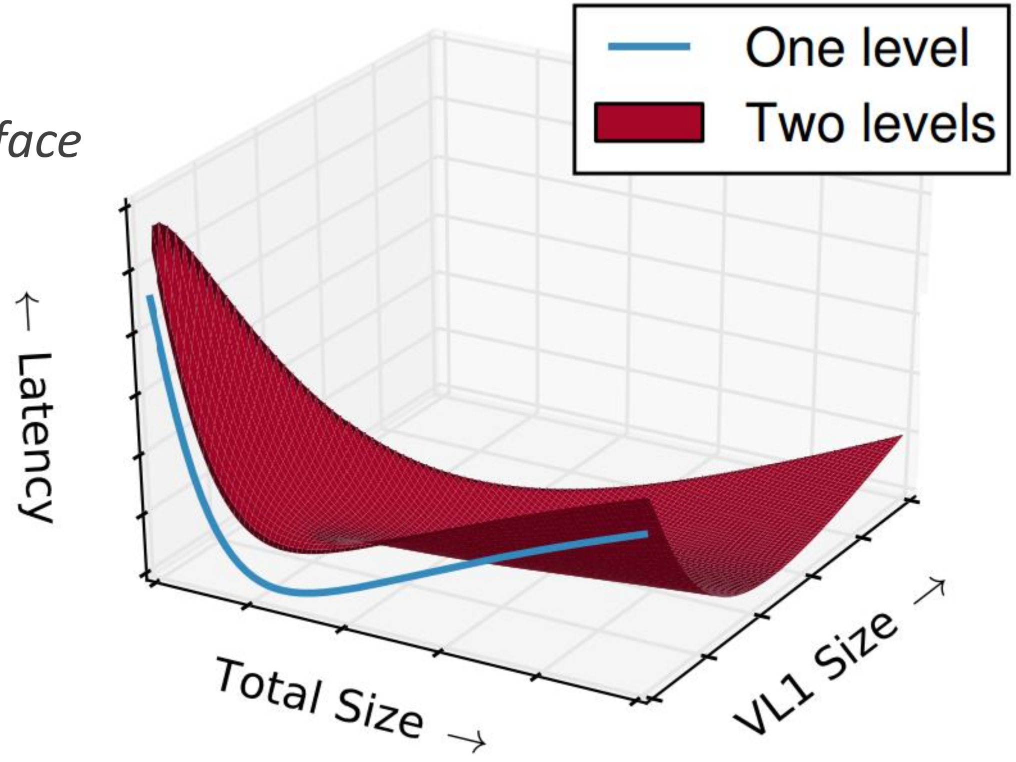


Modeling Hierarchy Latency

$$\text{Latency} = \text{Accesses} \times \text{L1 Latency} + \text{L1 Misses} \times \text{L2 Latency} + \text{L2 Misses} \times \text{Memory Latency}$$

Two-level virtual hierarchies give *latency surface*

- Total size
- VL1 size



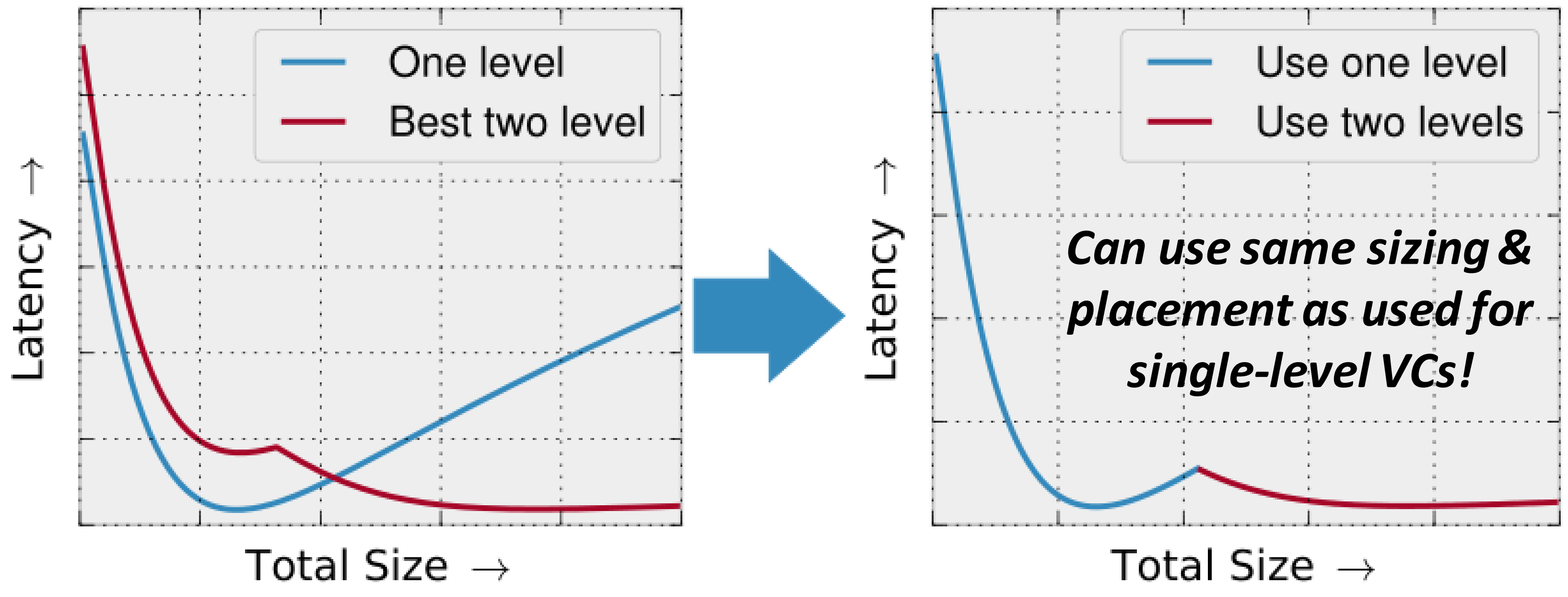
Complex tradeoffs

- VL1 size influences VL2 access latency
- VL2 size influences VL1 miss penalty
- Etc

Modeling Hierarchy Latency

Jenga selects the hierarchy that performs best at every size

- One vs. two levels
- VL1 size



Evaluation

36-tile multicore with 18MB SRAM cache and 1GB stacked DRAM

20 random mixes of SPECCPU2006

Cache organizations

- S-NUCA: “LRU” baseline, no stacked DRAM
- Jigsaw: No stacked DRAM
- Alloy: Stacked DRAM L4
  - Issues parallel, speculative memory accesses
  - *Spends energy to improve performance*
- JigAlloy: Jigsaw L3 + Alloy L4
- Jenga

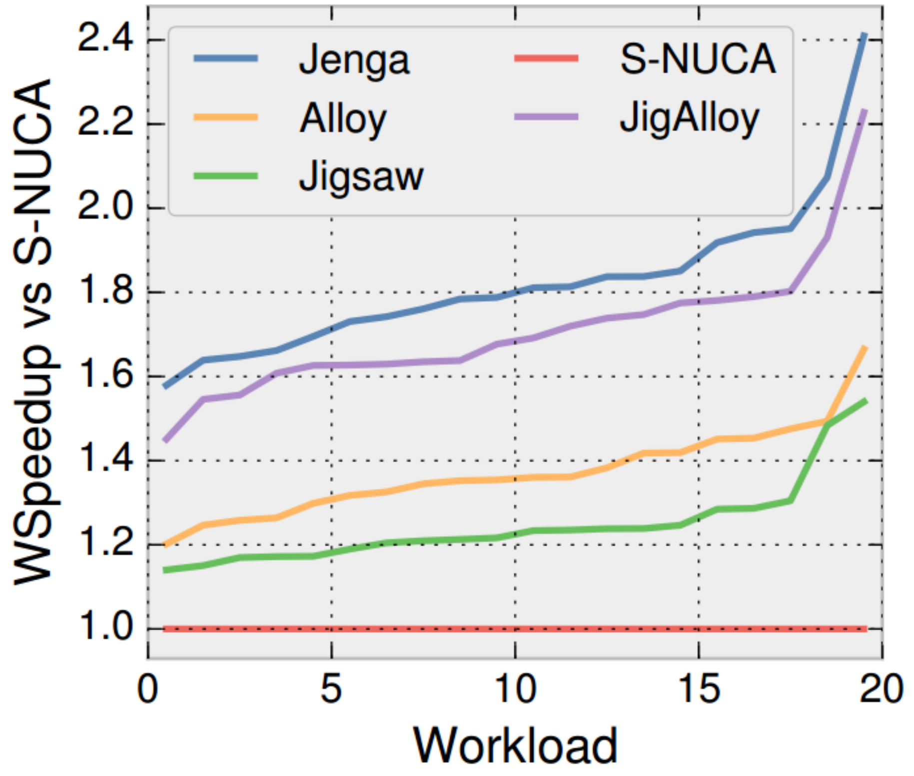
Evaluation: Performance

Jenga improves weighted speedup…

vs. S-NUCA by up to 2.2X/gmean 82%

vs. JigAlloy by up to 13%/gmean 7%

- Up to 24% for individual apps



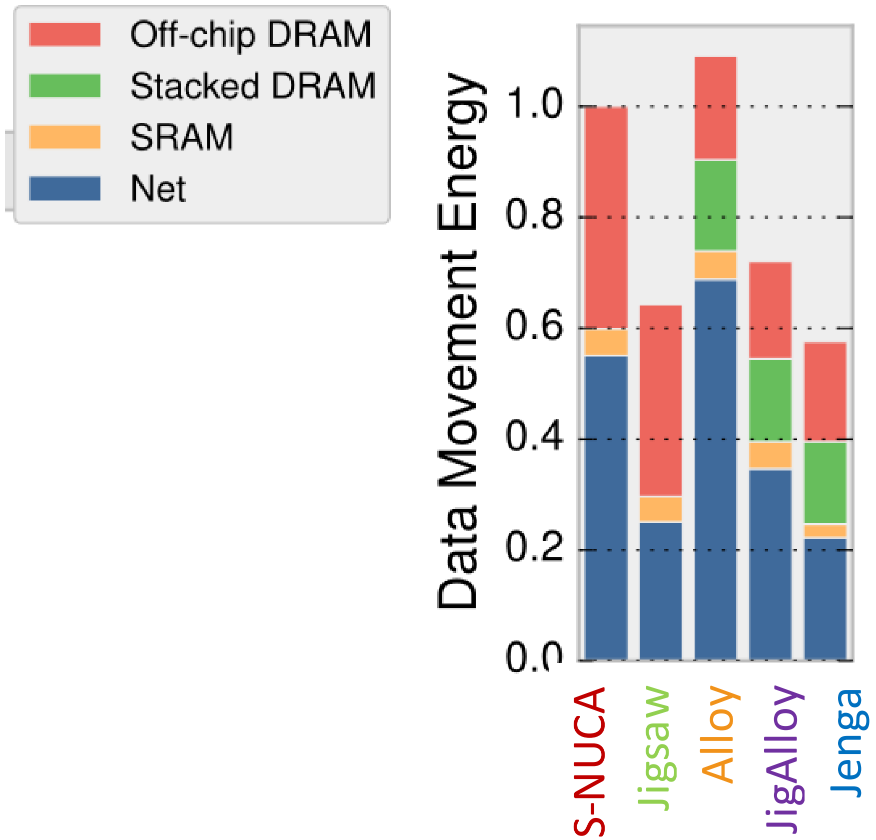
Evaluation: Energy

JigAlloy spends energy to improve performance
- Adds 12% energy vs Jigsaw

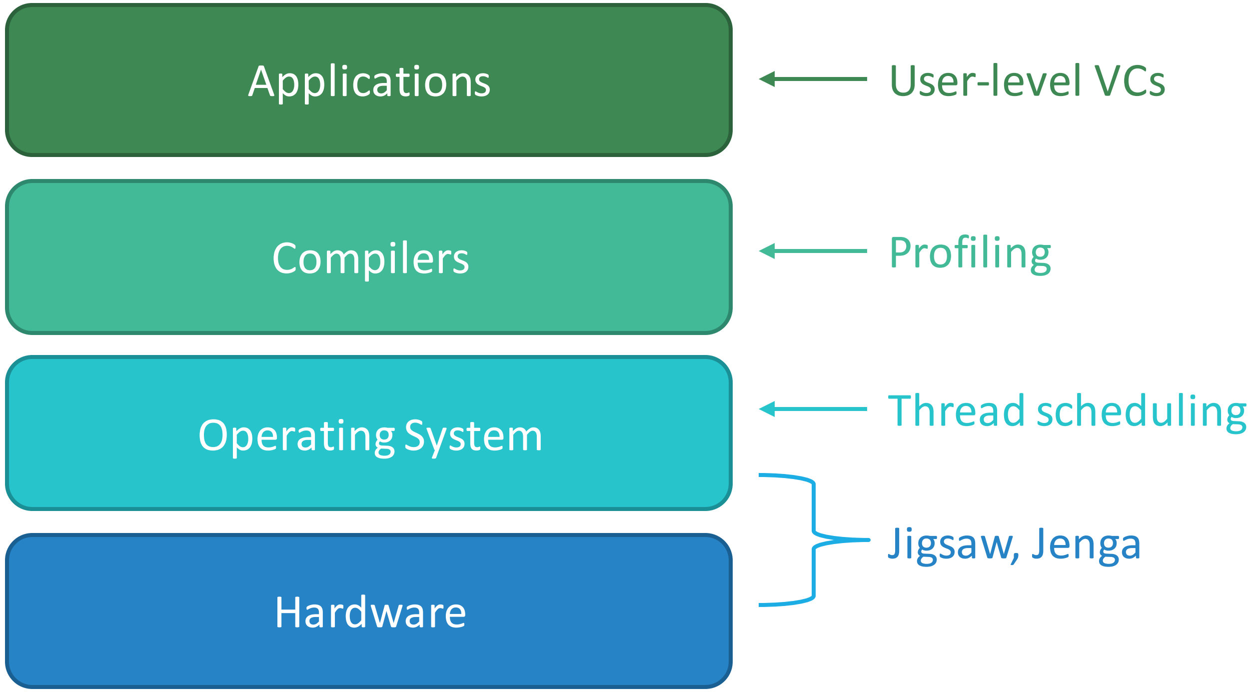
Jenga reduces data movement energy…
- vs. S-NUCA by 43%
- vs. JigAlloy by 20%
- vs. Jigsaw by 11%

Jenga sidesteps the energy-performance tradeoff!

Overall, Jenga improves energy-delay product…
- vs. S-NUCA by up to 3.6X/gmean 2.6X
- vs. JigAlloy by up to 24%/gmean 15%



Other Work:
Virtual Caches Up The System Stack



Virtual Hierarchy Summary

Adapt cache resources to suit applications

- Enough space to fit working set
- At minimum distance

Improve performance *and* save energy

Cache performance scales independent of system size

- Implications for system architecture and algorithms

Robust framework to manage heterogeneous memories

Analytical Cache Replacement

High-Performance Cache Replacement

Optimal policy (Belady's MIN) is impractical

Empirical policies:
- Traditional: LRU, LFU, random
- Statistical cost function *[IGRD, ICS'04]*
- Bypass streaming accesses *[DIP, ISCA'07]*
- Predict likelihood of reuse *[SDBP, MICRO'10]*
- Predict time until reference *[RRIP, ISCA'10]* *[SHIP, MICRO'11]*
- Protect lines from eviction *[PDP, MICRO'12]*
- Use data mining to find best policy *[GIPR, MICRO'13]*
- Etc

Perform poorly on some apps
➔ Not making best use of information

Analytical policies:
- Independent reference model *[Aho, J. ACM'71]*

Assumes static behavior
➔ Not a good model of LLC accesses

We use a simple reference model that captures dynamism and solve for its optimal policy: EVA.

EVA is practical and outperforms empirical policies.

Background: Independent Reference Model

Analytical policies use a simplified *memory reference model* to derive optimal policy

Prior work uses *independent reference model* *[Aho, JACM'71]*

- Candidates have static, non-uniform reference probabilities
- E.g., a 4-way cache

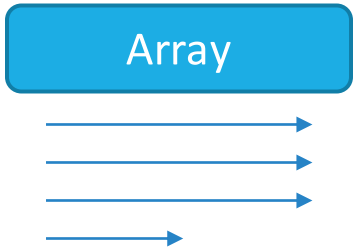
0x1000	0x1234	0xBEEF	0x1337
0.1	0.2	0.05	0.0001

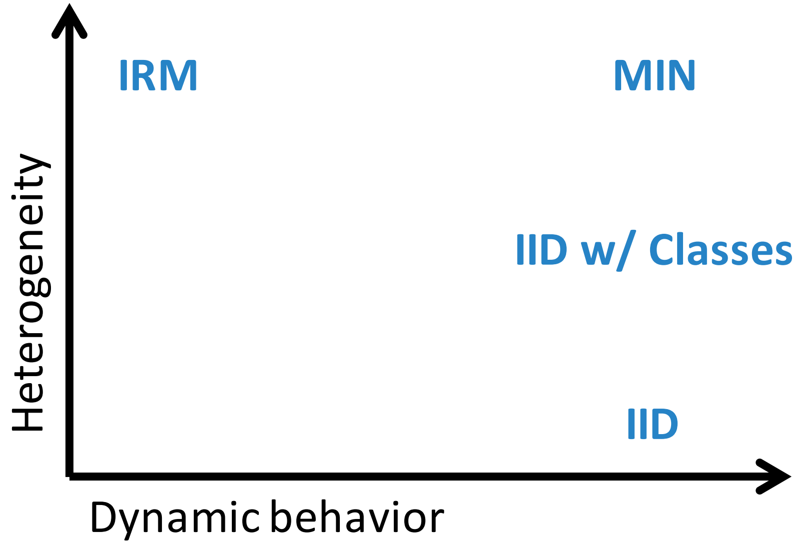
Optimal policy: *evict candidate with lowest reference probability*

Background: Independent Reference Model

IRM is a poor model of LLC references
- Focuses on heterogeneity
- At the expense of dynamic behavior

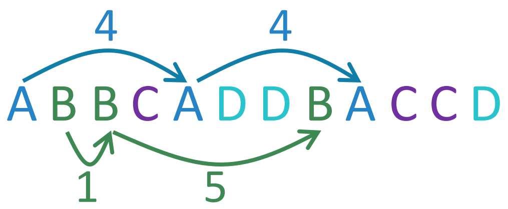
Relatively few threads access LLC
- LLC candidates tend to behave similarly
- Dynamic behavior is paramount





Iid Reuse Distance Model

Reuse distance is the number of accesses between references to same address



Reuse distances are independently and identically distributed according to the *reuse distance distribution*, $\mathrm{P}(D = d)$.

What is the right replacement policy?

What’s the Right Approach?

Goal: Maximize hit rate

Constraint: Limited cache space

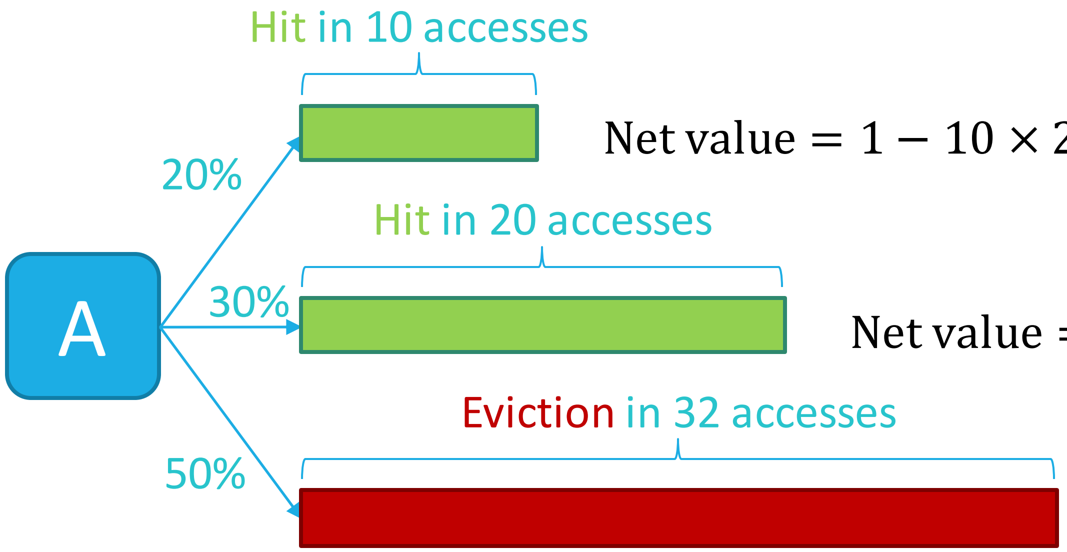
The replacement policy must balance the probability a candidate will hit (reward) against the cache space it takes away from other candidates (opportunity cost)

But how do we tradeoff between these incommensurable objectives?

Replacement By Economic Value Added

Key idea: Time spent in the cache costs *forgone hits*

$\text{Cache hit rate} = 40\%$
$\text{Cache size} = 16 \text{ lines}$
$\text{Line hit rate} = (40\%)/16 = 2.5\%$



$\text{Net value} = 1 - 10 \times 2.5\% = 0.75$

$\text{Net value} = 1 - 20 \times 2.5\% = 0.5$

$\text{Net value} = 0 - 32 \times 2.5\% = -0.8$

$$\boldsymbol{EVA = 20\% \times 0.75 + 30\% \times 0.5 + 50\% \times -0.8 = -0.1}$$

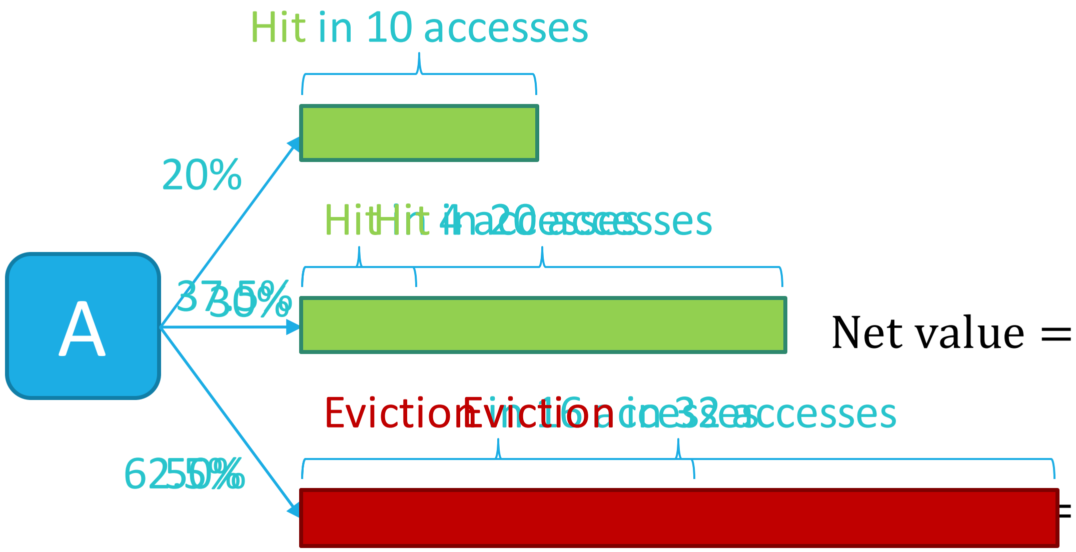
➔ A tends to lower the cache's hit rate

Replacement By Economic Value Added

Key idea: Time spent in the cache costs *forgone hits*

$\text{Cache hit rate} = 40\%$
$\text{Cache size} = 16 \text{ lines}$
$\text{Line hit rate} = (40\%)/16 = 2.5\%$

…16 accesses later



$\text{Net value} = 1 - 4 \times 2.5\% = 0.9$

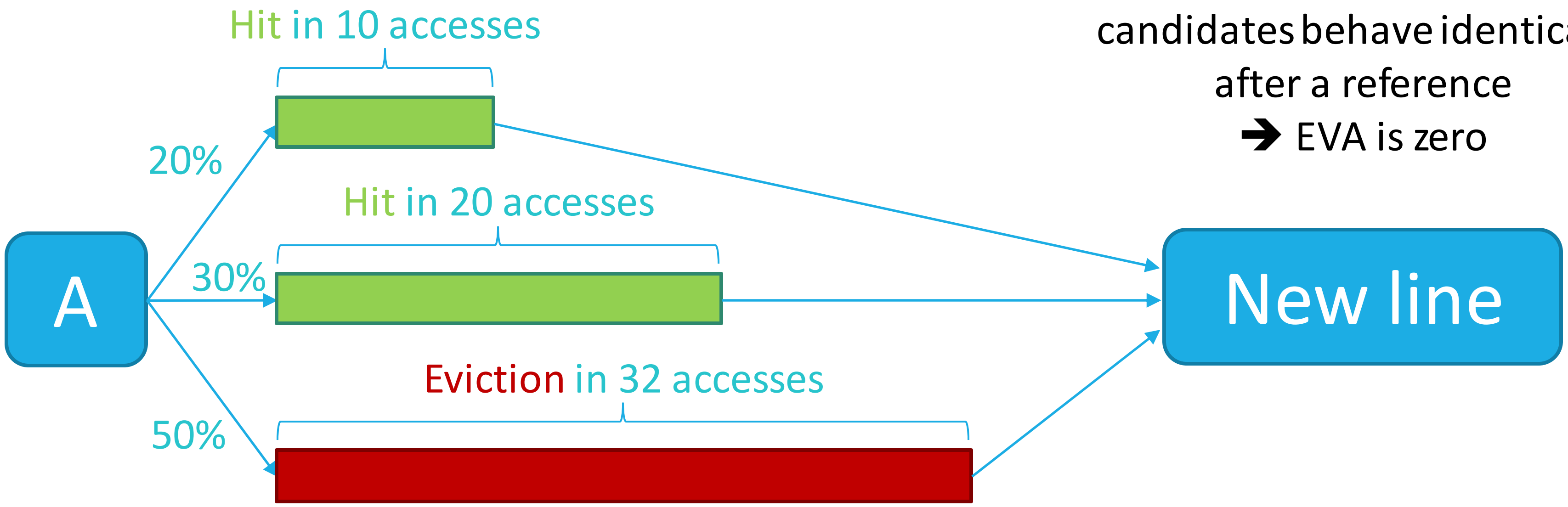
$= 0 - 16 \times 2.5\% = -0.4$

$$\boldsymbol{EVA = 37.5\% \times 0.9 + 62.5\% \times -0.4 = 0.0375}$$

➔ A now tends to increase hit rate

Replacement By Economic Value Added

What about the future?

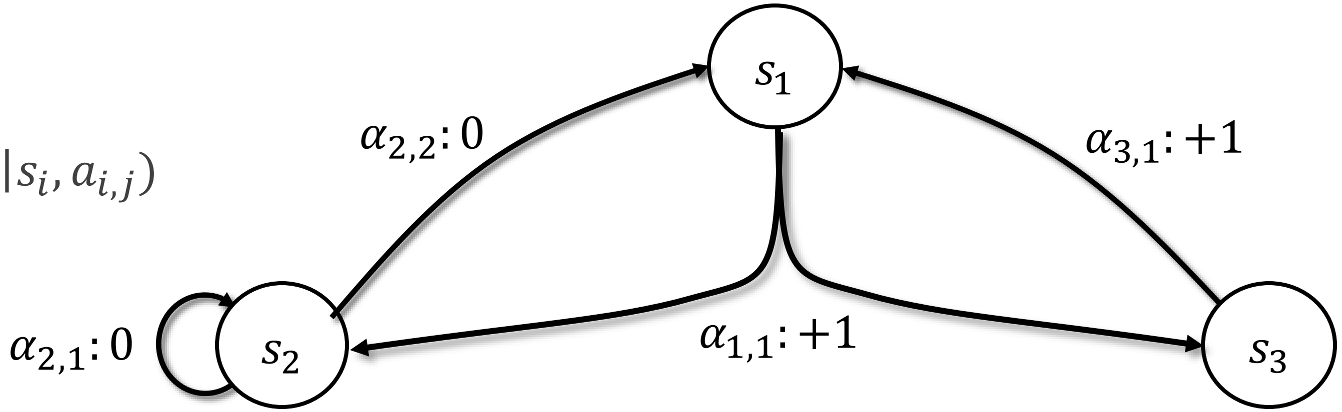


In the iid reuse distance model, candidates behave identically after a reference
➔ EVA is zero

Cache Replacement As An MDP

Markov decision processes extend Markov chains with decision making

- States s_i
- Actions $\alpha_{i,j}$
- Rewards $r(s_i, a_{i,j})$
- Transition probabilities $\mathrm{P}(s_k | s_i, a_{i,j})$



MDP theory lets one find the optimal policy to maximize some metric, e.g. total reward

Cache Replacement As An MDP

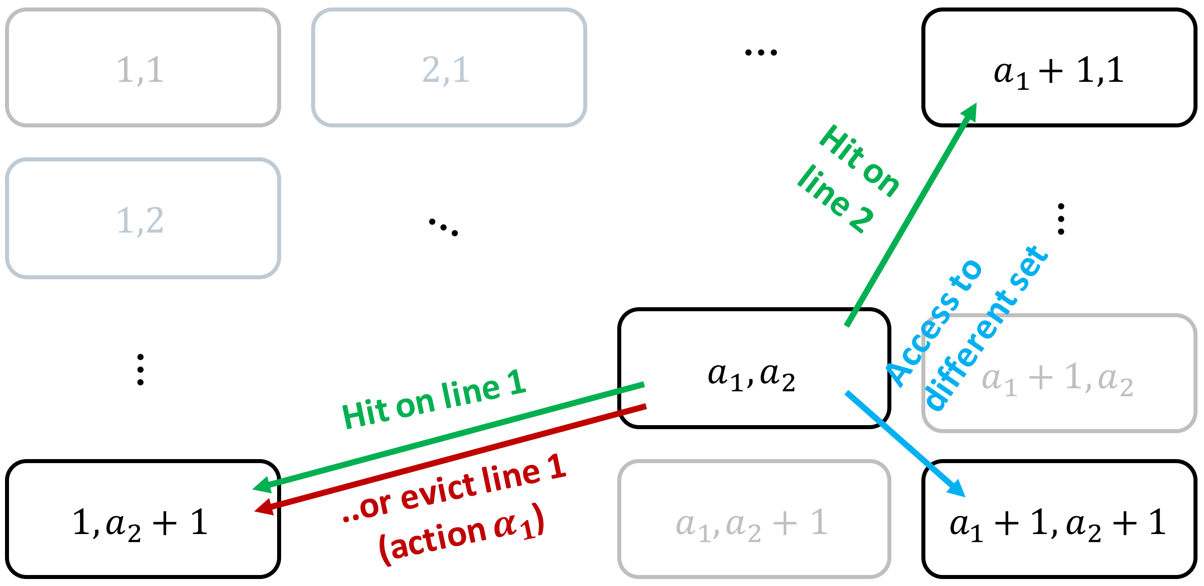
States: each candidate's age

Actions: which candidate to evict

Rewards: +1 for hit

Transition probabilities: iid model

Objective: maximize average reward
- I.e., maximize hit rate



Cache Replacement As An MDP

MDP theory gives the optimal policy

Define bias as the *expected total reward minus the average reward*

- This is EVA: hits minus forgone hits

The optimal policy maximizes the *expected future bias*

- ➔ Evicting candidate with lowest EVA is optimal

Recovering Some Heterogeneity

Some programs have clearly distinct *classes* of accesses

Iid memory reference model w/ classification
- Different reuse distance distribution per class, $\mathrm{P}(D_C = d)$
- Within each class, reuse distances are identically distributed
- All reuse distances are independent

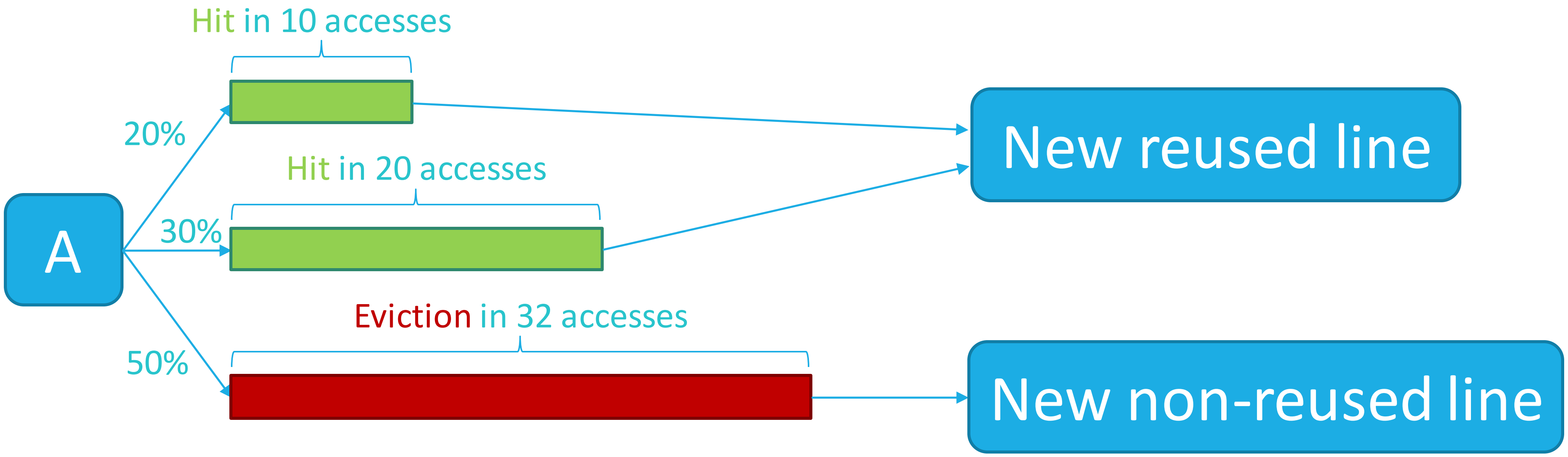
Reuse vs. non-reused classification

[RRIP, ISCA'10][DCS,HPCA'12]
- Distinguishes working set that fits in cache
- Simple but effective

Replacement By Economic Value Added

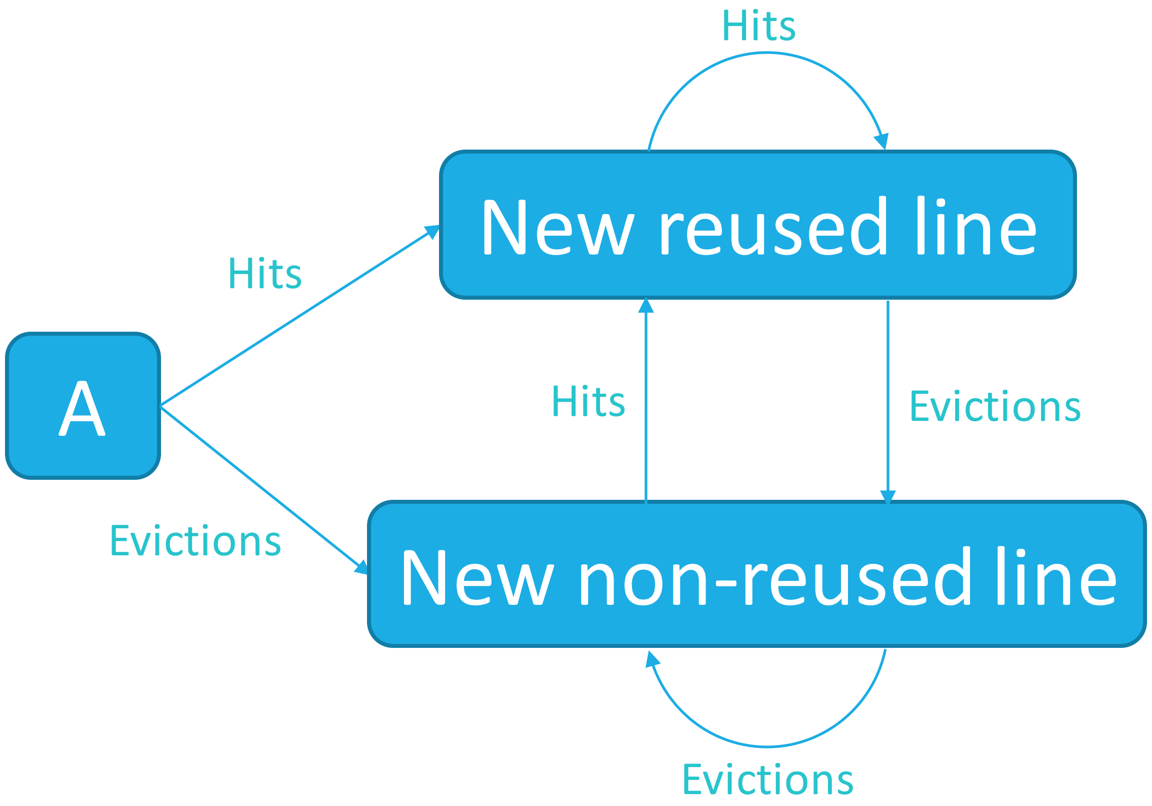
What about the future?

With classification, candidates do **not** behave identically after a reference



Replacement By Economic Value Added

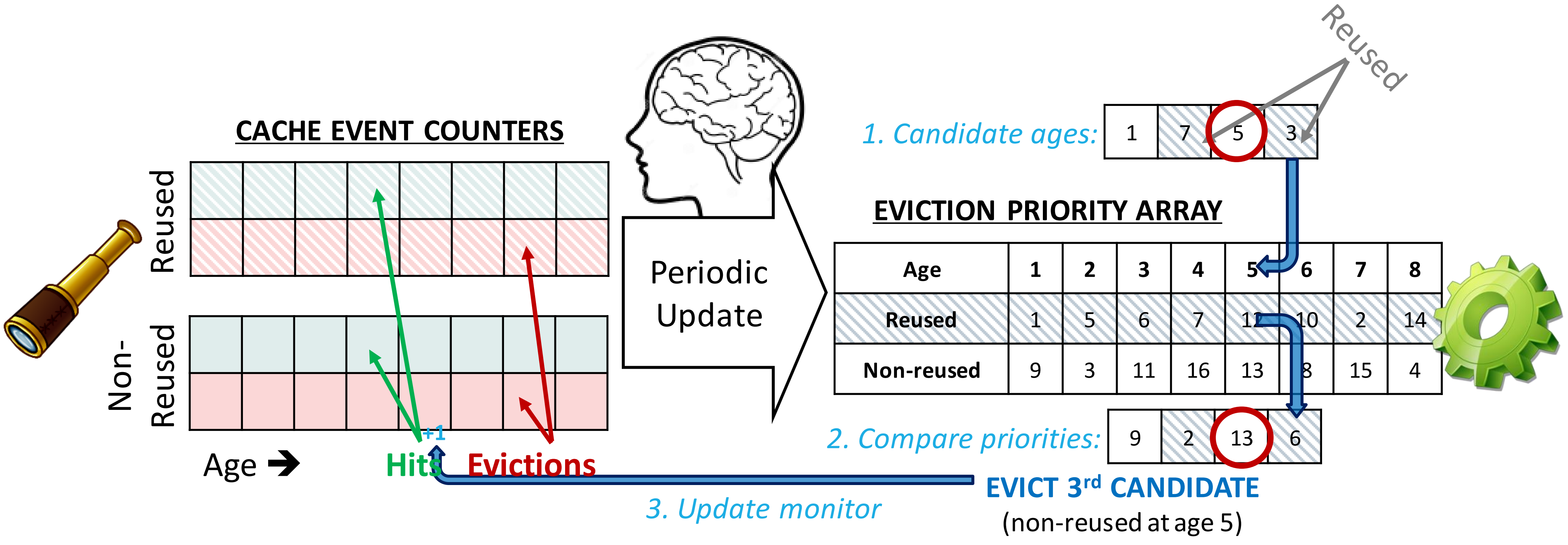
What about the future?



With classification, candidates do **not** behave identically after a reference

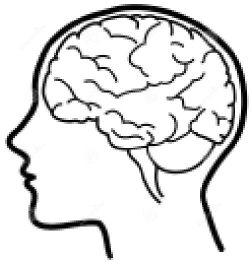
Can account for EVA in future lifetimes by adding a single constant term (see thesis)

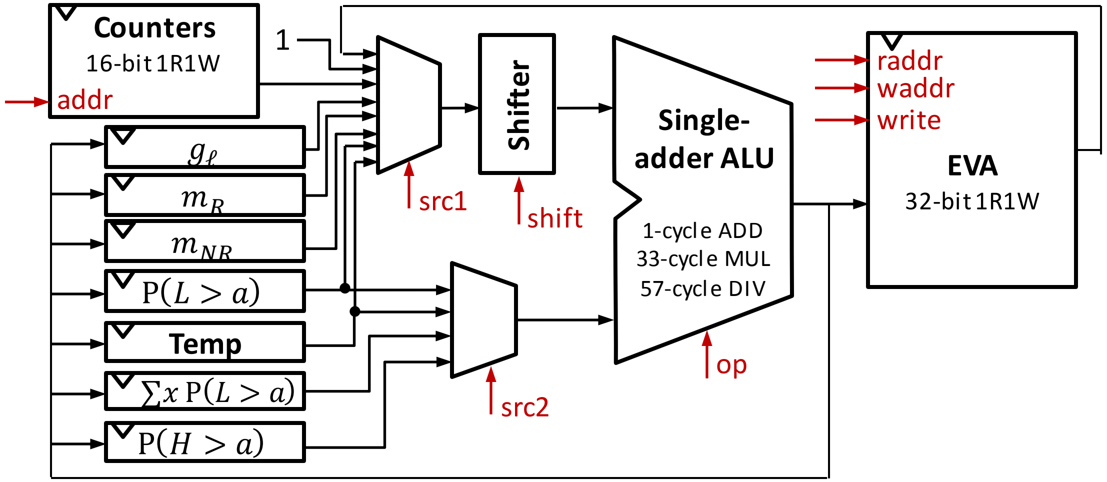
Implementation



Implementation – Updates

Small circuit computes EVA





Sorting FSM computes eviction priorities (not shown)

Implementation – Overheads

Synthesized in 65nm commercial manufacturing process

SRAM overheads from CACTI

	Area		Energy	
	mm^2	vs. 1MB	nJ / LLC miss	vs. 1MB
Ranking	0.010	0.05%	0.014	0.6%
Counters	0.025	0.14%	0.010	0.4%
Updates	0.052	0.30%	380 / 128K	0.1%
Total	–	**0.5%**	–	**1.1%**

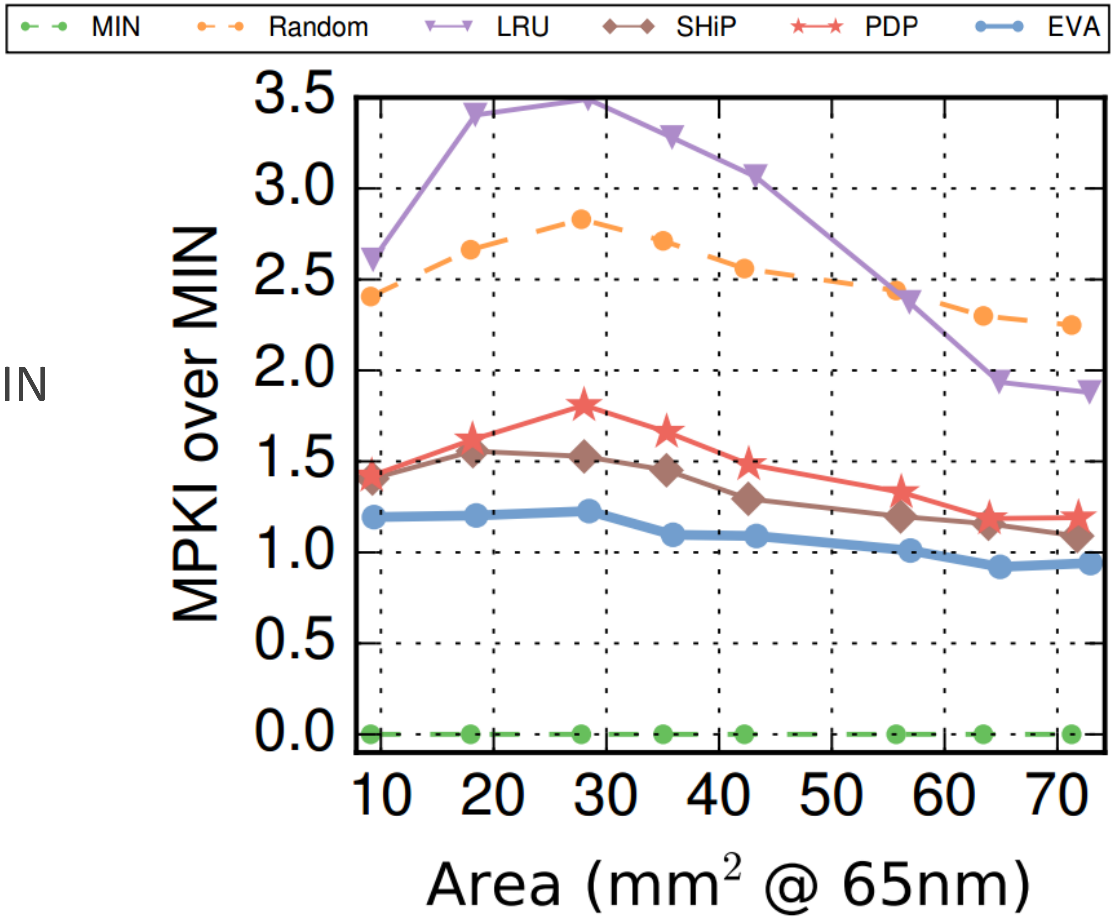
Evaluation – Cache Performance

EVA greatly reduces misses vs. prior policies

- Avg. MPKI over SPECCPU2006
- LLC sizes 1MB to 8MB

EVA closes 57% of gap between random and MIN

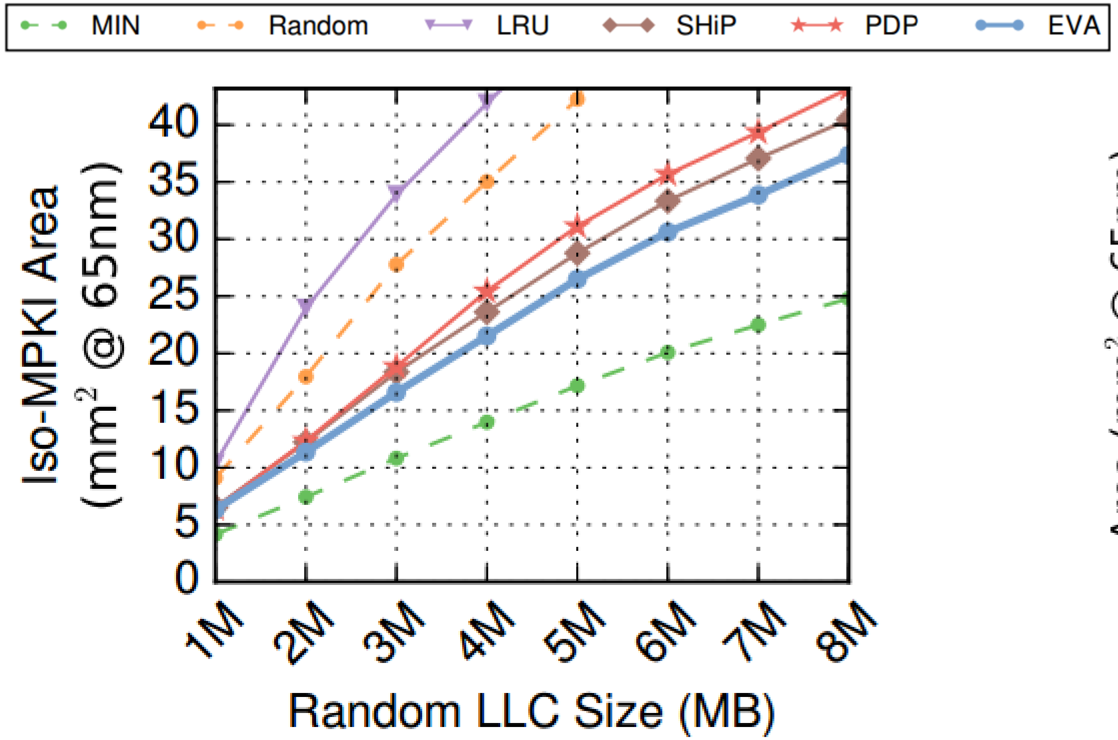
- vs. 41%-45% for prior policies

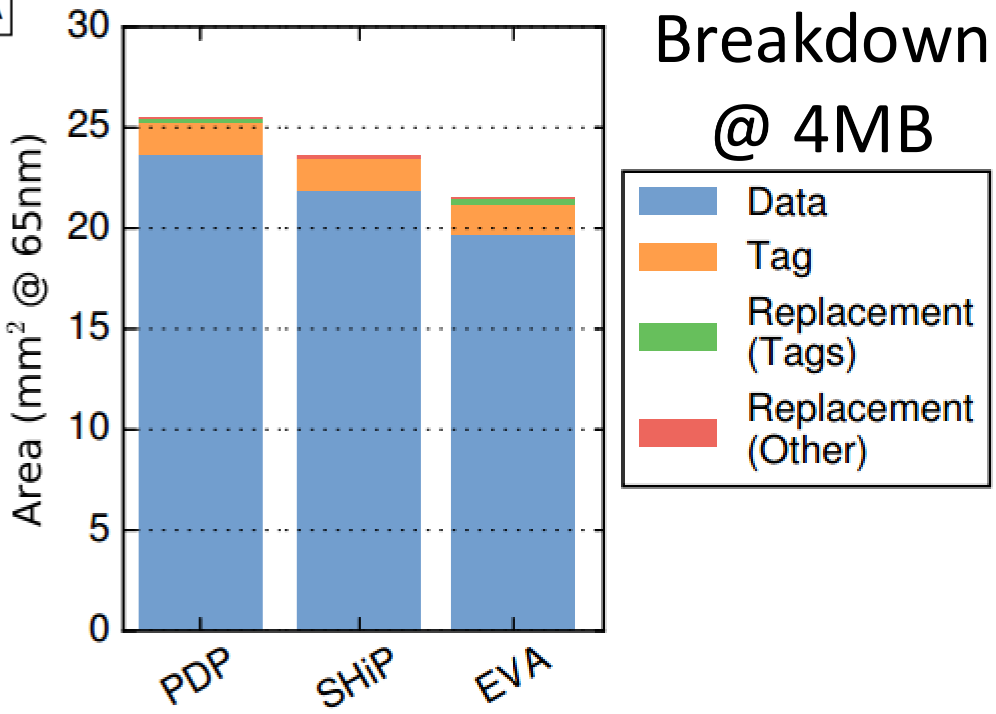


Evaluation – Cache Area

Because EVA improves performance, it requires less space to match performance

- EVA saves 9% area vs SHiP





Other Work

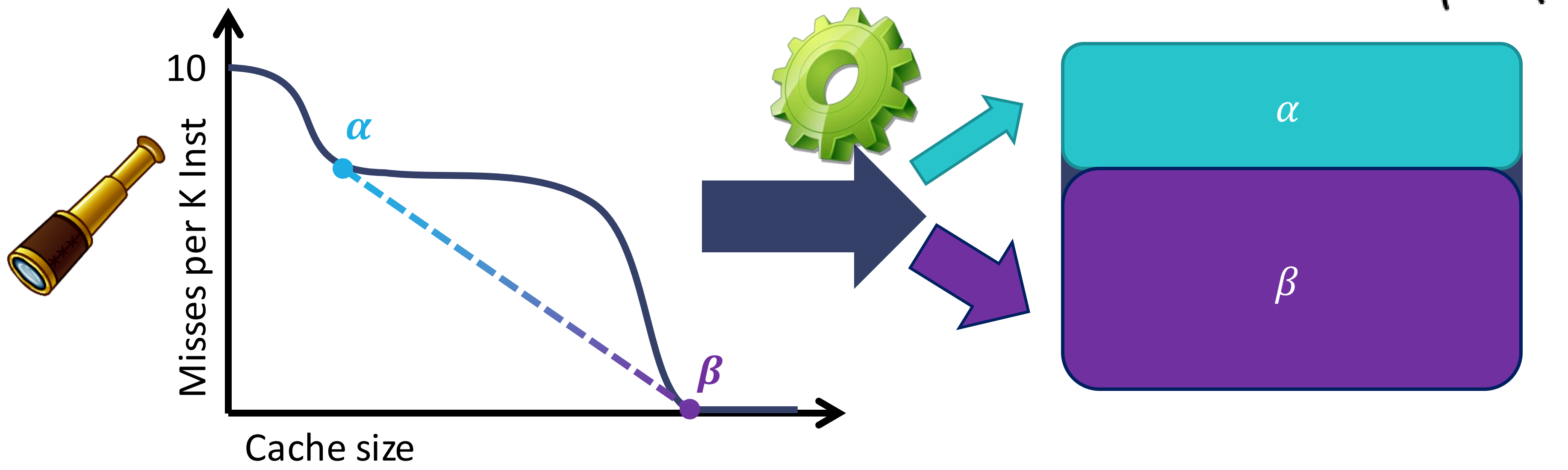
Provable Convex Cache Performance

Partitioning and high-performance replacement should be complementary

- Partitioning requires miss curves – easy for LRU, hard otherwise

We use partitioning to fix LRU's problems without sacrificing its benefits

- Specifically, we avoid performance cliffs and guarantee *convex miss curves*
- Prove it works using simple model of miss curve scaling



A Cache Model for Modern Processors

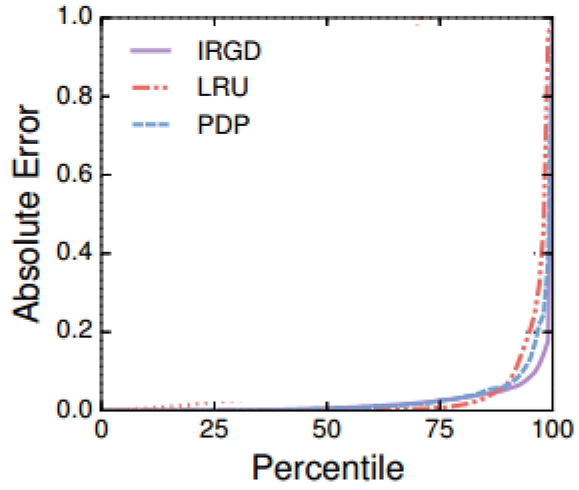
Accurately model performance of arbitrary replacement policies

- Use iid memory reference model
- Model replacement policies as *ranking functions*: $R(a) = \text{eviction priority}$

Simple set of equations for probability distribution of age, hits, and evictions

- Fixed-point iteration converges rapidly

Mean error of ~3% across policies, benchmarks & LLC sizes



Cache Calculus

Explicit, closed-form solutions of cache performance

Relax discrete cache model into system of ordinary differential equations

- E.g., for random replacement:

$$H'' = \frac{D''}{D'} H' - \frac{D'}{1-D} E'$$
$$E'' = -\frac{m}{S}(H' + E')$$

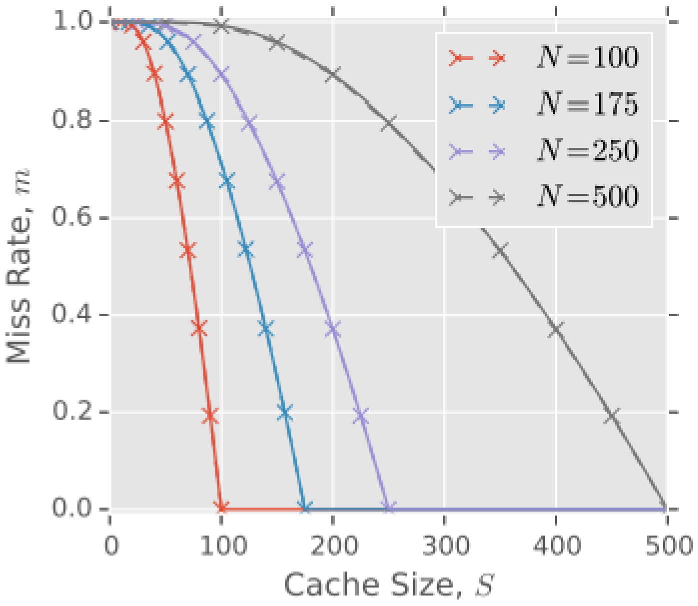
Can use numerical analysis to solve for arbitrary access patterns

Can solve explicitly miss rate m on particular access patterns

- E.g., scanning: $m = 1 - \text{ProductLog}\left(-\omega \mathrm{e}^{-\omega}\right)/\omega$ where $\omega = \mathrm{N}/S$
- E.g., stack: $m = 1 - \frac{1}{2\omega} - \frac{1}{4\omega^2} - \frac{1}{4\omega^3} + O\left(\frac{1}{\omega^4}\right)$

Array size   Cache size

Extremely good match with simulation!



Acknowledgements – Research

Anant Agarwal
2008-2012

Frans Kaashoek

Nickolai Zeldovich

2012-2013

Daniel Sanchez
2013-

Acknowledgements – Research

Acknowledgements – Education










Conclusion

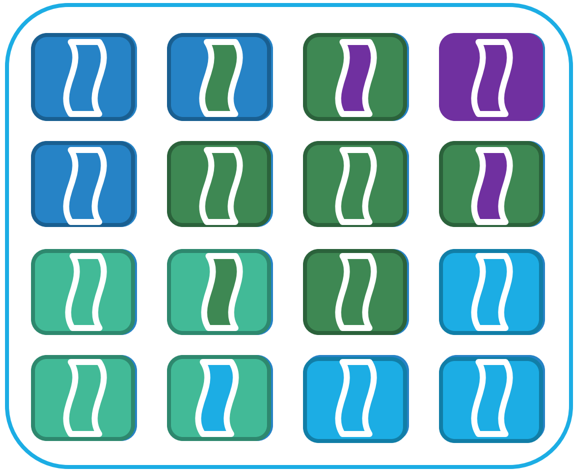
Data movement is a growing problem in current systems

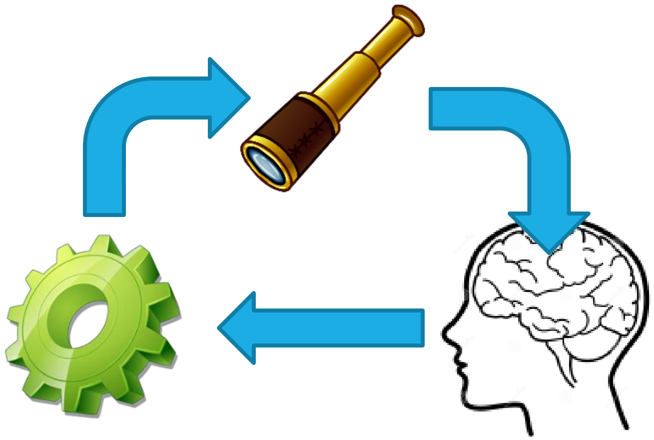
“Common case”, heuristic design is insufficient

Analytical memory systems achieve robust, high performance

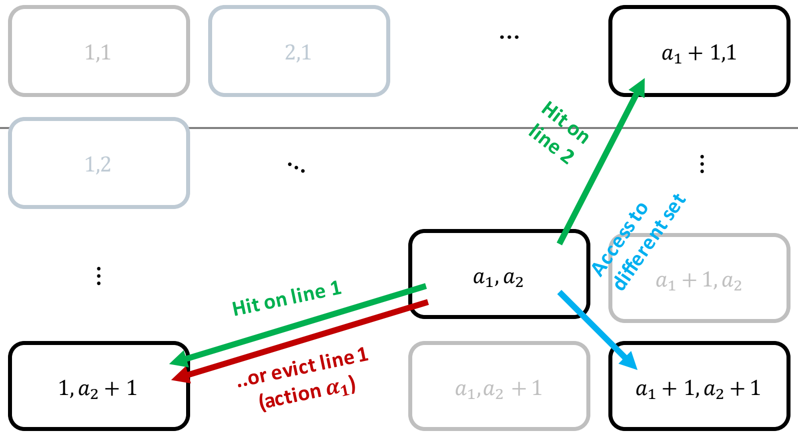
- Virtual cache hierarchies
- Analytical cache replacement

Mechanisms, monitoring, and models give a blueprint for future memory systems





Questions?


1,1
2,1
…
$a_1+1,1$
Hit on line 2
1,2
⋱
⋮
⋮
a_1, a_2
Access to different set
a_1+1, a_2
Hit on line 1
..or evict line 1 (action α_1)
$1, a_2+1$
a_1, a_2+1
a_1+1, a_2+1