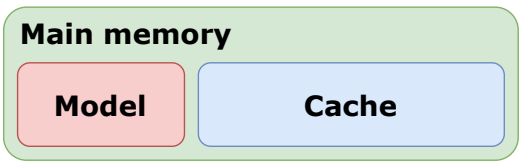
# Kosmo: Efficient Online Miss Ratio Curve Generation for Eviction Policy Evaluation

Kia Shakiba, Sari Sultan, and Michael Stumm

University of Toronto

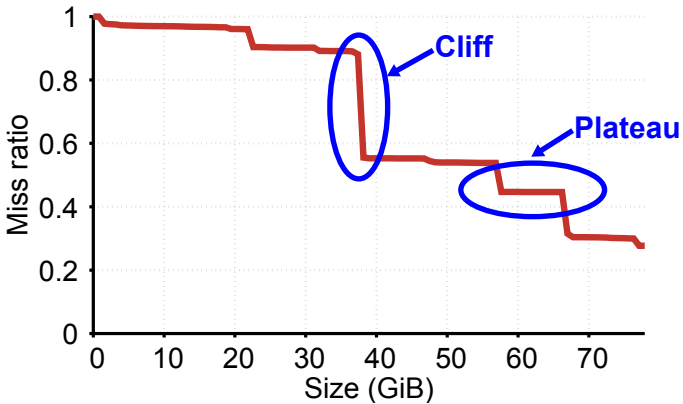
# Motivation

- In-memory caches are key for high performance
- Important to model these caches to properly configure size
  - Too small $\rightarrow$ Poor performance
  - Too large $\rightarrow$ Wasted resources / High cost
- Modelling done online to dynamically adjust cache



# Miss Ratio Curves (MRCs)

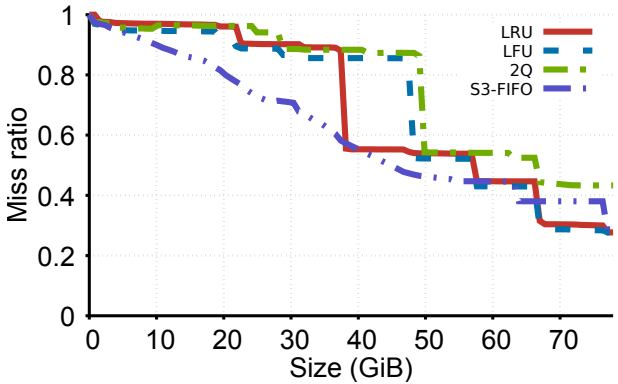
Only known tool to show trade-off between cache size and miss ratio



# MRCs are eviction policy-specific

Many policies exist

- LRU
- LFU
- FIFO
- 2Q
- S3-FIFO
- etc.

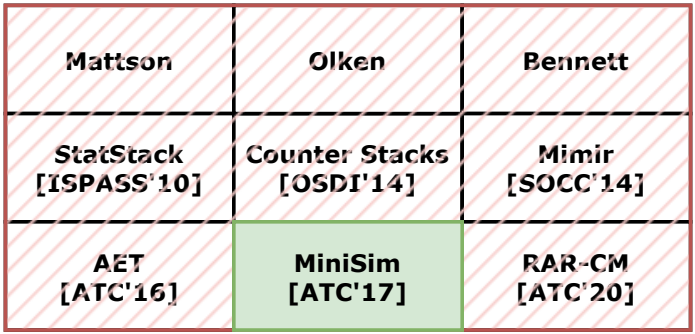



But which do we use?

# Many MRC generation algorithms exist

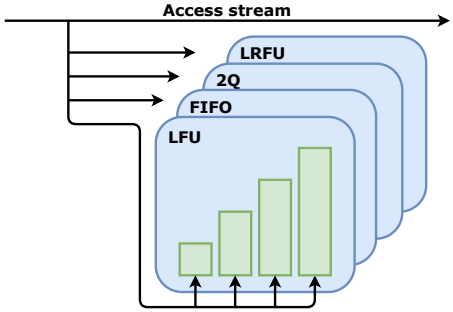
| Mattson | Olken | Bennett |
|---|---|---|
| StatStack [ISPASS'10] | Counter Stacks [OSDI'14] | Mimir [SOCC'14] |
| AET [ATC'16] | MiniSim [ATC'17] | RAR-CM [ATC'20] |

# All but MiniSim only work for LRU



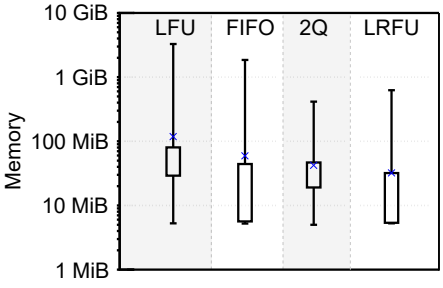
# Simulations and Miniature Simulations (MiniSim)

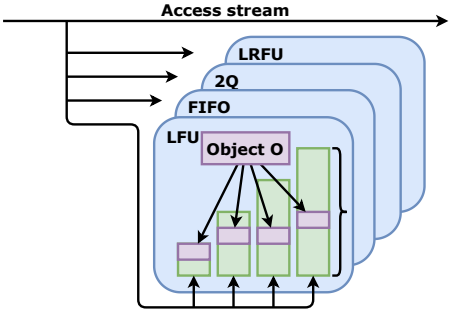
- Many individual simulations (e.g., 100)
- Final miss ratios are used to construct MRCs
- Separate series of sims per eviction policy



# Limitations of MiniSim

- High memory usage
- A priori specification of cache sizes to simulate



# Kosmo - Key ideas

| | Object allocation in caches | Simulated cache lifetime |
|---|---|---|
| MiniSim | Each has own copy | Maintained throughout |
| Kosmo | All share one copy | Reconstructed dynamically |

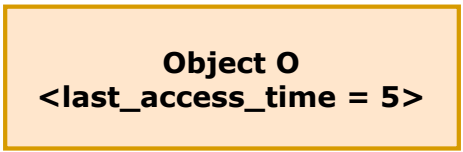
# Kosmo - How do we reconstruct a cache's stack?

For a cache of size $S$, for each object, Kosmo determines:

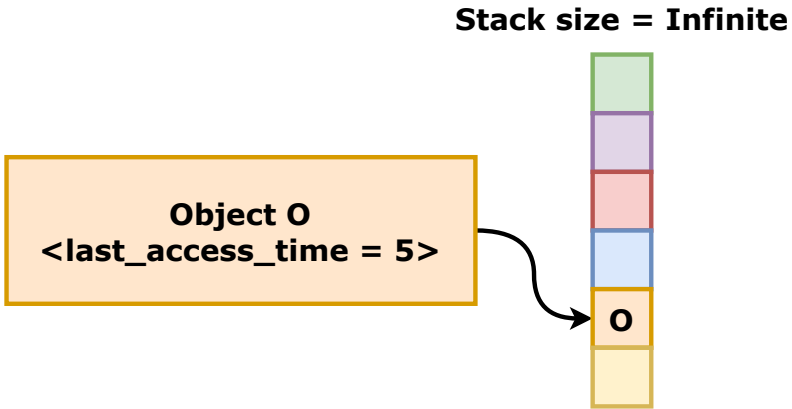
1. If it exists in the cache
2. Its position in the cache's stack

# Kosmo – LRU as an example

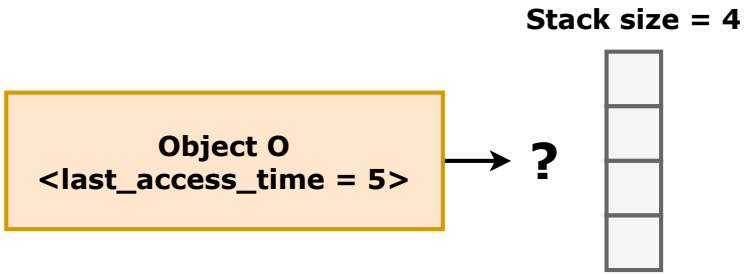
How do we determine an object's position in an LRU stack?



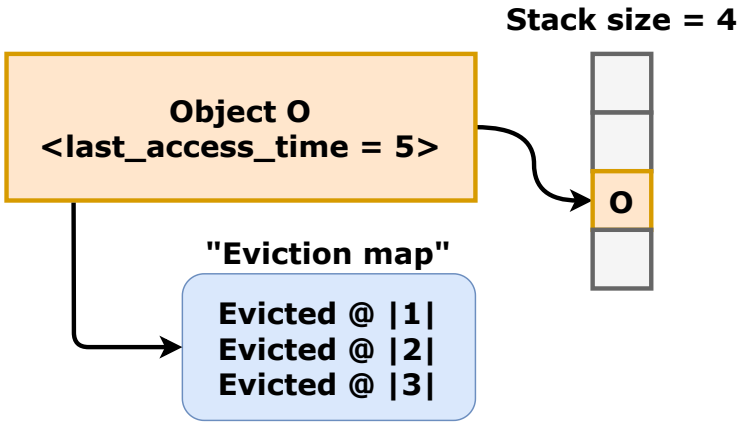
# Kosmo – LRU as an example



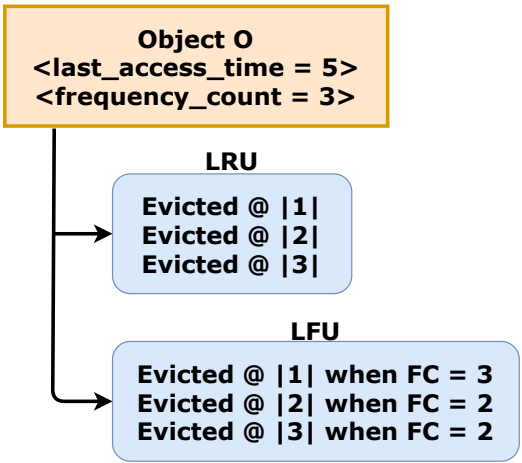
# Kosmo – LRU as an example



# Kosmo – LRU as an example



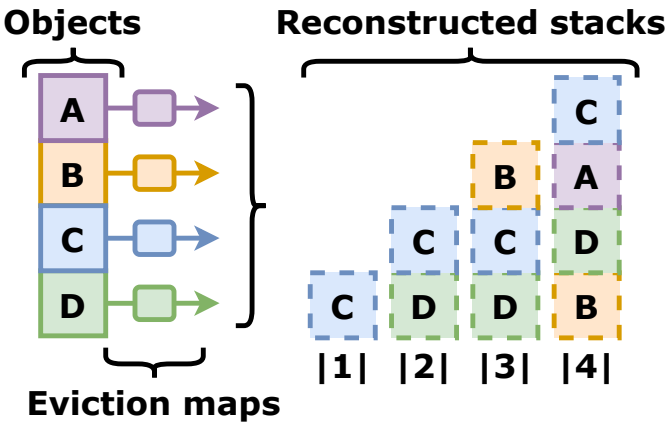
# Kosmo – Extending to LFU



# Kosmo – Reconstructing cache stacks

Using eviction maps, we can reconstruct a cache of any size.

*No a priori specification of cache sizes needed!*



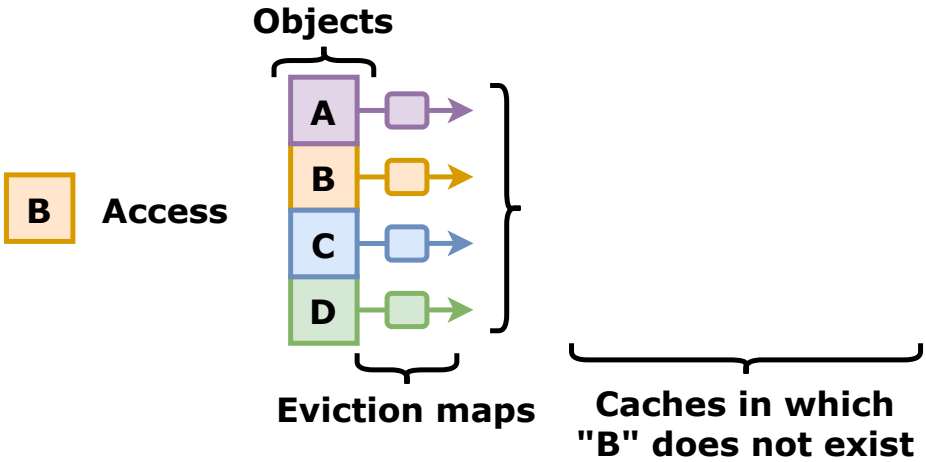
# Kosmo – Keeping eviction maps up-to-date

On each access, evictions may occur in some caches

- Eviction maps must be remain up-to-date!

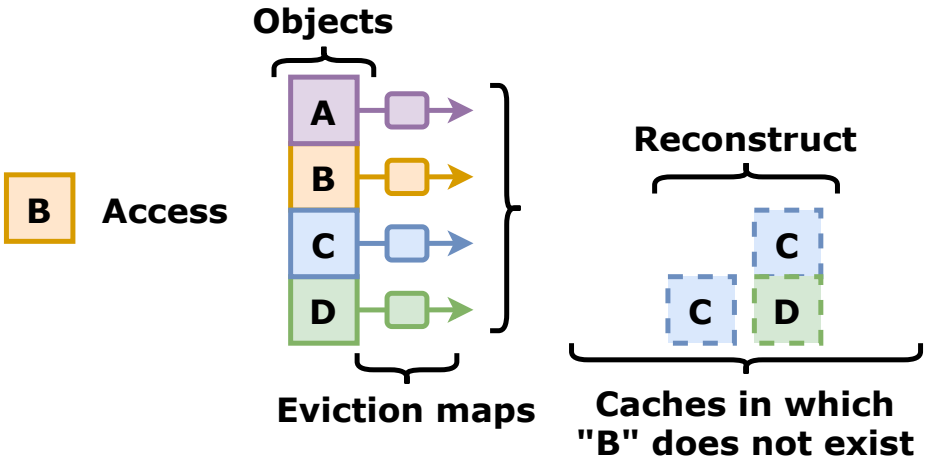
# Kosmo – Step 1

All caches in which “B” does not exist will have an eviction



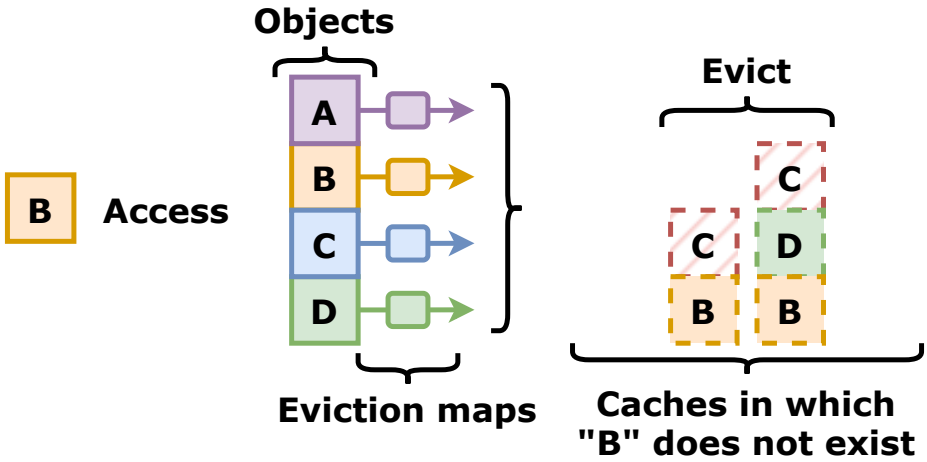
# Kosmo – Step 2

Reconstruct only these caches



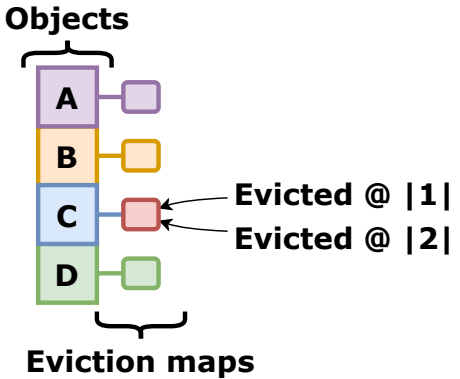
# Kosmo – Step 3

Determine which objects are evicted from these caches



# Kosmo – Step 3

Update these objects' eviction maps

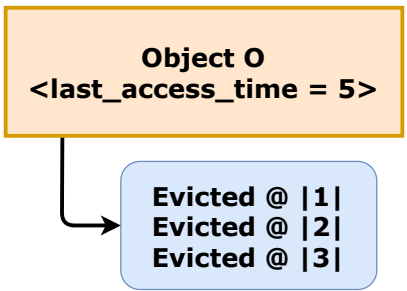


# Kosmo – Many eviction records!

Problem: Storing a record of each eviction for each object is costly

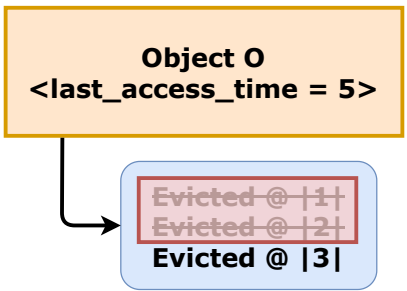
# Kosmo – Eviction record pruning

Eviction maps may contain redundant entries

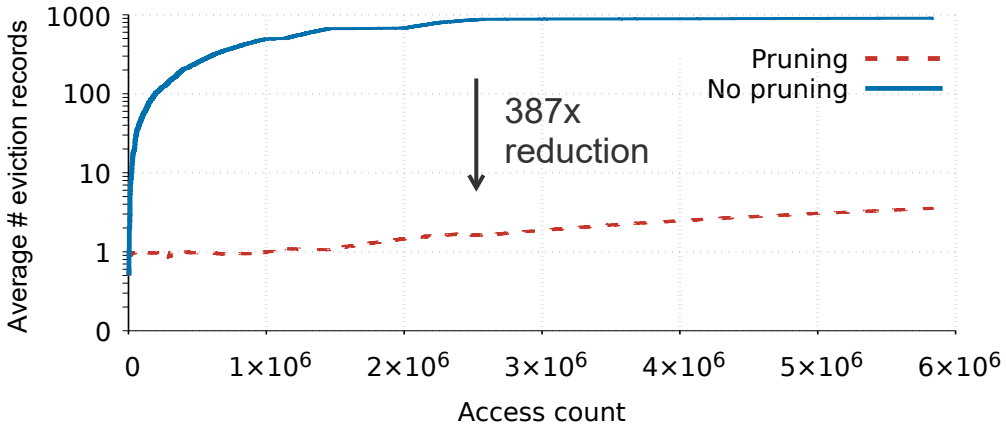


# Kosmo – Eviction record pruning

Reduce size of eviction maps by pruning redundant entries



# Kosmo – Eviction record pruning

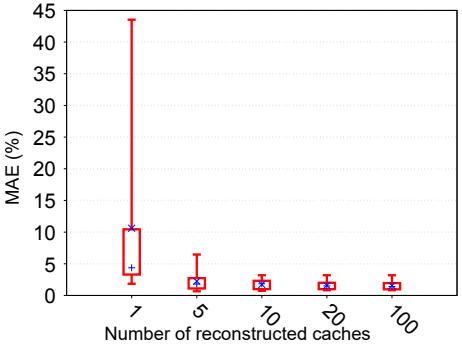


# Kosmo – Many reconstructed caches!

Problem: Many cache stacks are reconstructed on each access

# Kosmo – Reduced number of reconstructed caches

- Reduce number of reconstructed cache stacks to a configurable value
- Performance versus accuracy trade-off



# Kosmo – Many objects!

Problem: The number of objects can grow quite large

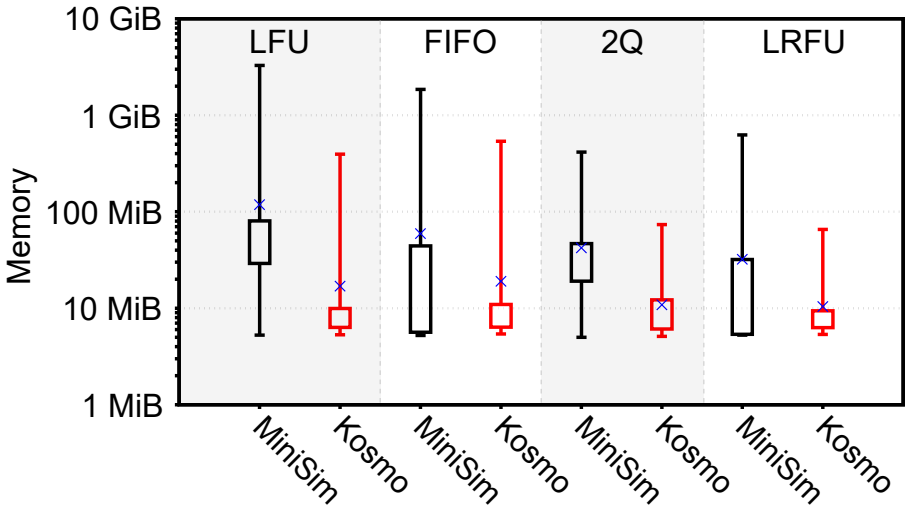
# Kosmo – SHARDS

- Use SHARDS sampling
  - Reduced number of stored objects to a small constant
  - Reduced overhead by a factor of over 1,000

# Kosmo – Results

How does Kosmo compare to MiniSim?

# Memory usage



# Conclusion

- Modelling non-LRU policies requires Kosmo or MiniSim
- The overhead of MRC generation is important
  - Kosmo has a significantly lower memory footprint than MiniSim
- Kosmo removes need for a priori specification of parameters