University of Patras

HMTY

Instruction-based Reuse Distance Prediction Replacement Policy

Pavlos Petoumenos, Georgios Keramidas, and Stefanos Kaxiras

Contributions

- Instruction-based Reuse Distance Prediction Replacement Policy
- Simple and clear implementation
- Efficient reuse distance sampling

Motivation

- LRU: Inefficient for LLCs
  - Much worse than the theoretical optimal
- OPT: Needs future reuse behavior

- Can we approximate it?
  - Yes we can!
  - Strong correlation between instructions (PC) and moment of next access

Overall Methodology

- Observe
- Remember
- Manage

Overall Methodology

- Observe
  - Track cache lines and find their reuse behavior
- Remember
- Manage

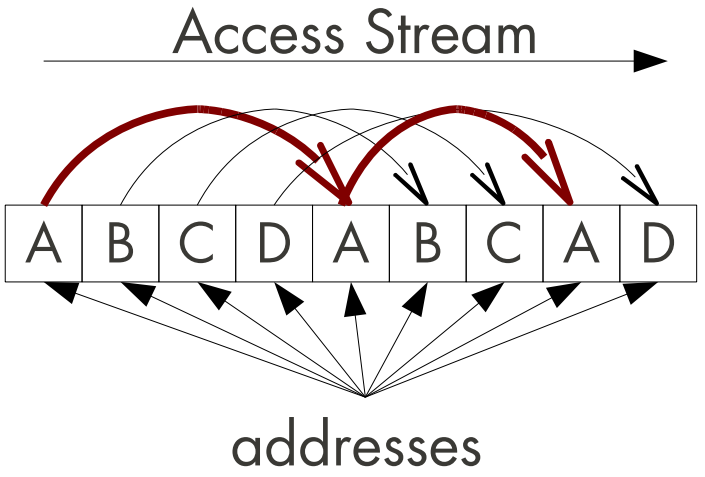
Overall Methodology

- Observe
  - Track cache lines and find their reuse behavior
- Remember
  - Filter and store instructions and the reuse behavior associated with them
- Manage

Overall Methodology

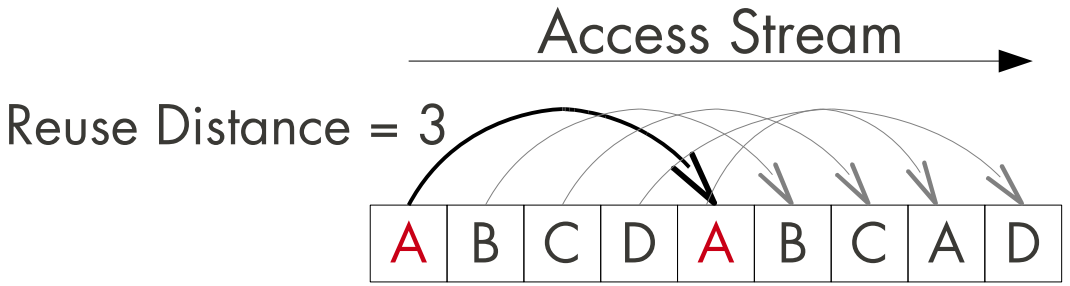
- Observe
  - Track cache lines and find their reuse behavior
- Remember
  - Filter and store instructions and the reuse behavior associated with them
- Manage
  - Implement an optimal-like algorithm using this info

Observe: Quantify Reuse Behavior



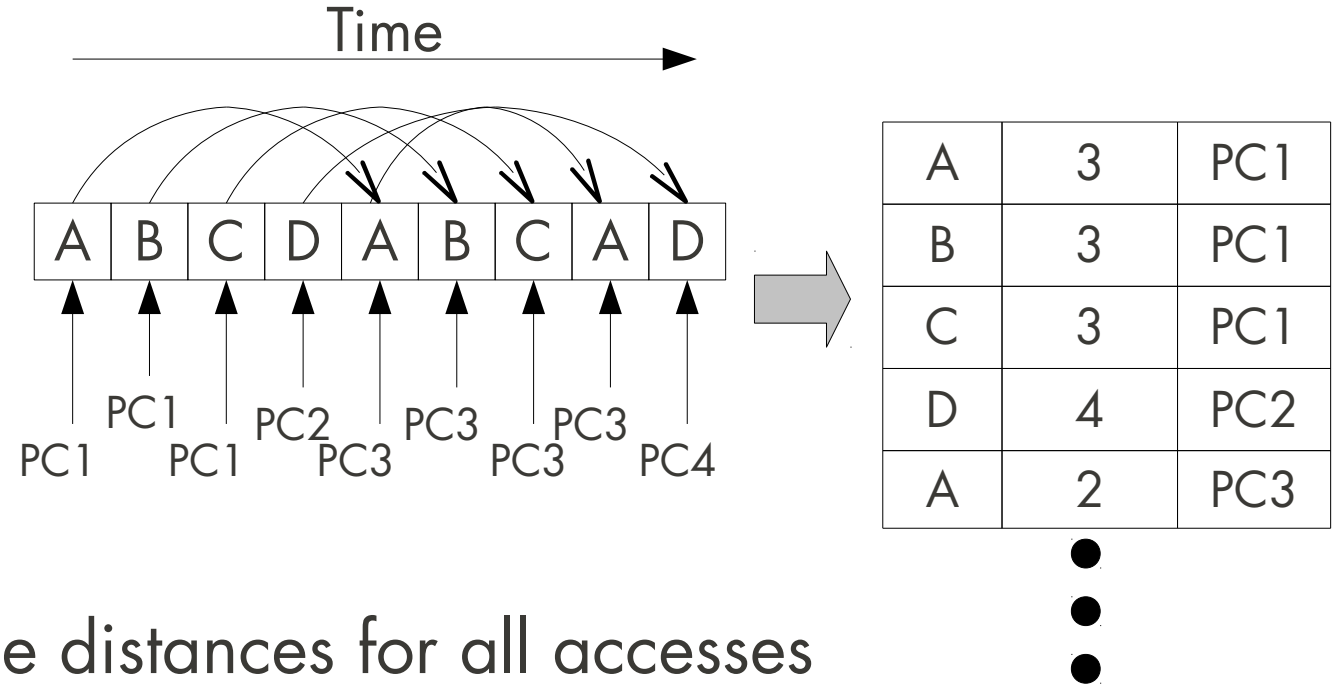
- Some lines are reused faster than others
  - How can we quantify this?

Observe: Quantify Reuse Behavior



- Reuse Distance
  - The # of intervening LLC accesses between two accesses to the same cache line

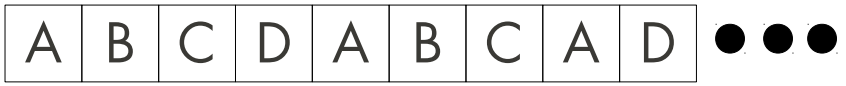
Observe



- Ideally:
  - Produce reuse distances for all accesses
  - Associate them with the instructions which caused them
- Infeasible:
  - Required storage → an order of magnitude less than memory footprint of the program

Observe: RDSampler

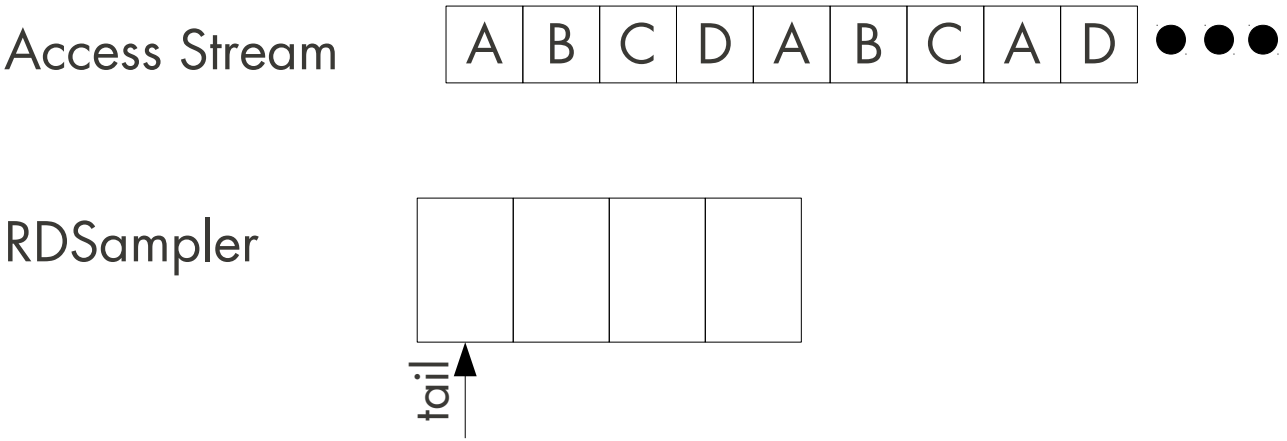
Access Stream



RDSampler

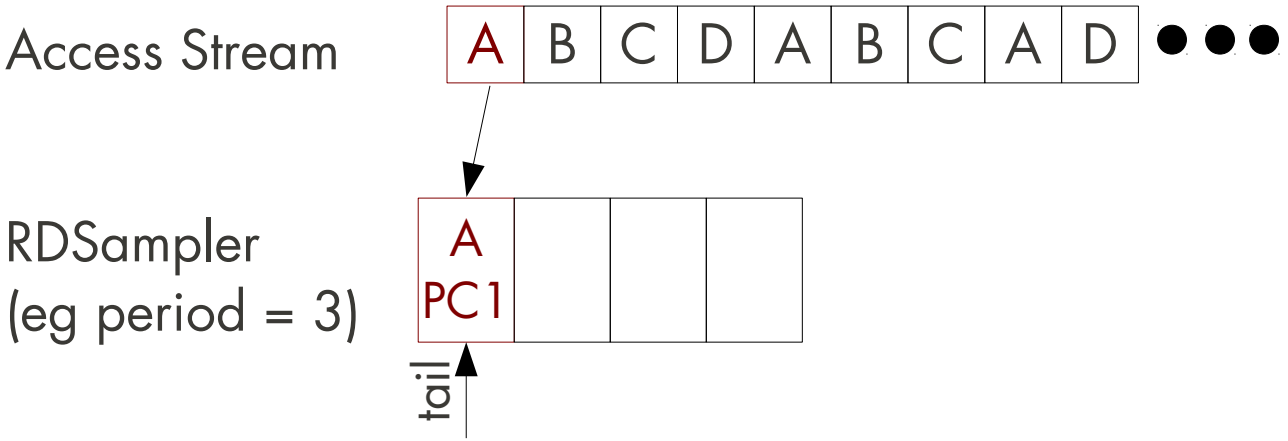
- Reuse-Distance Sampler (RDSampler)

Observe: RDSampler



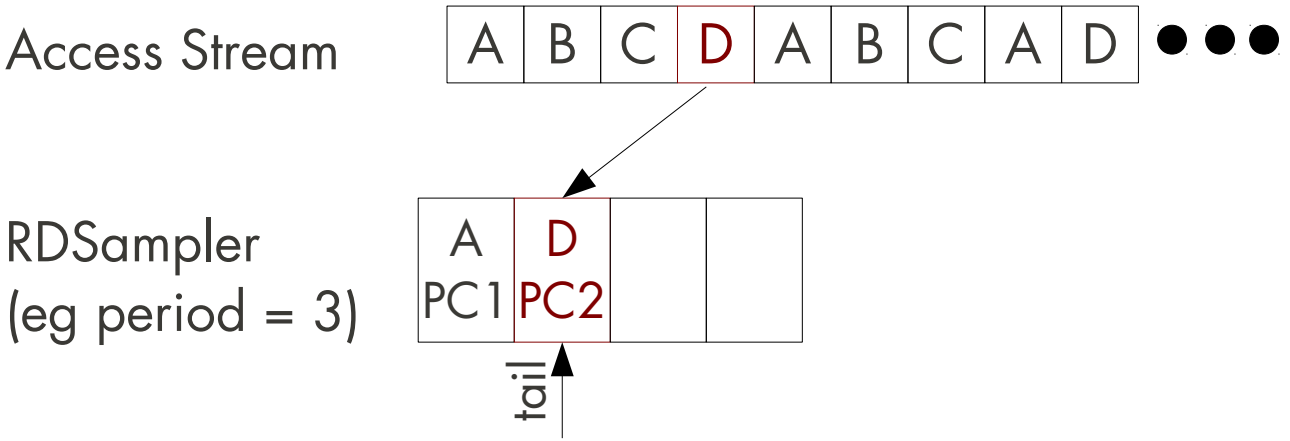
- Reuse-Distance Sampler (RDSampler)
  - Small circular FIFO Buffer

Observe: RDSampler



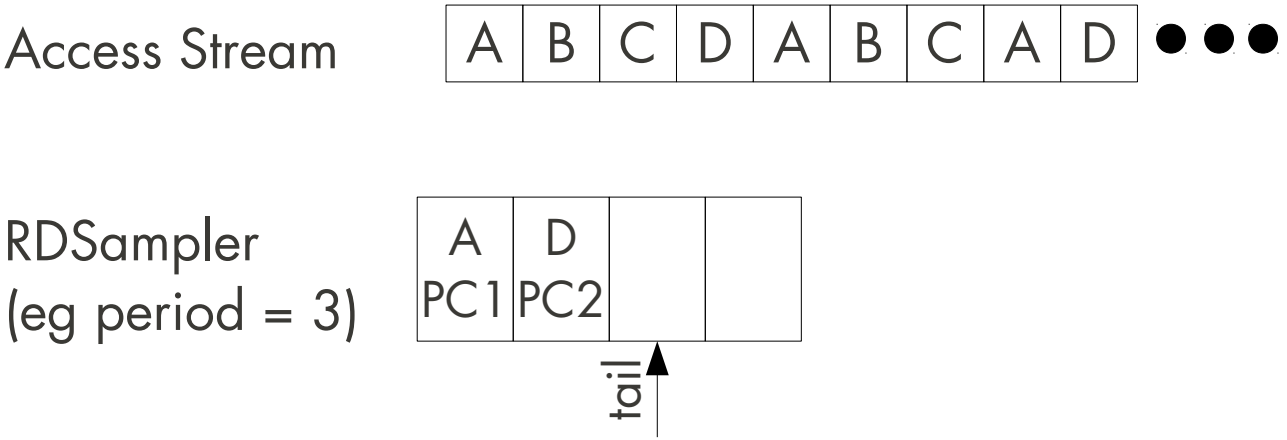
- Reuse-Distance Sampler (RDSampler)
  - Small circular FIFO Buffer
  - Periodically samples the access stream
    - Sample → accessed address, instruction which caused the access

Observe: RDSampler



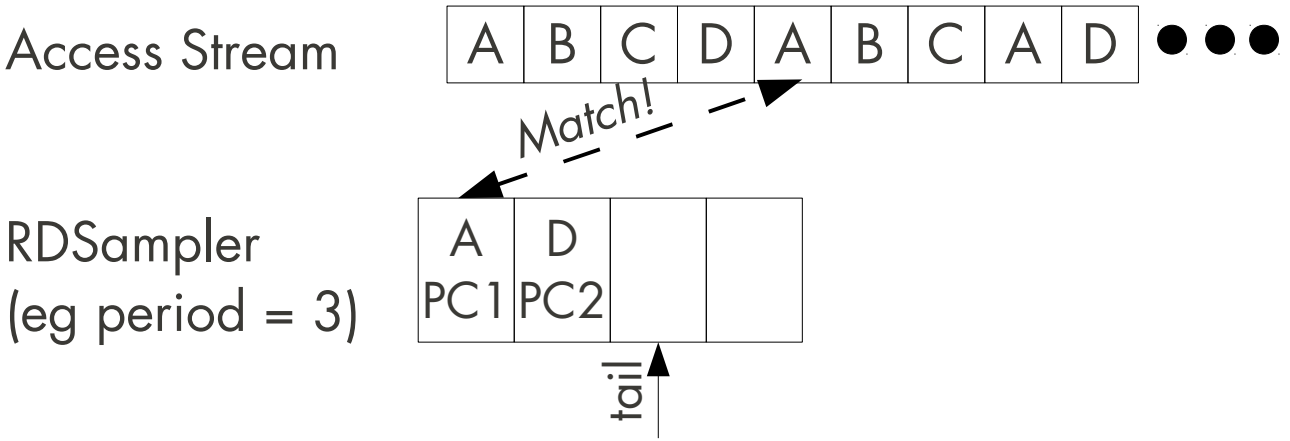
- Reuse-Distance Sampler (RDSampler)
  - Small circular FIFO Buffer
  - Periodically samples the access stream
    - Sample → accessed address, instruction which caused the access

Observe: RDSampler



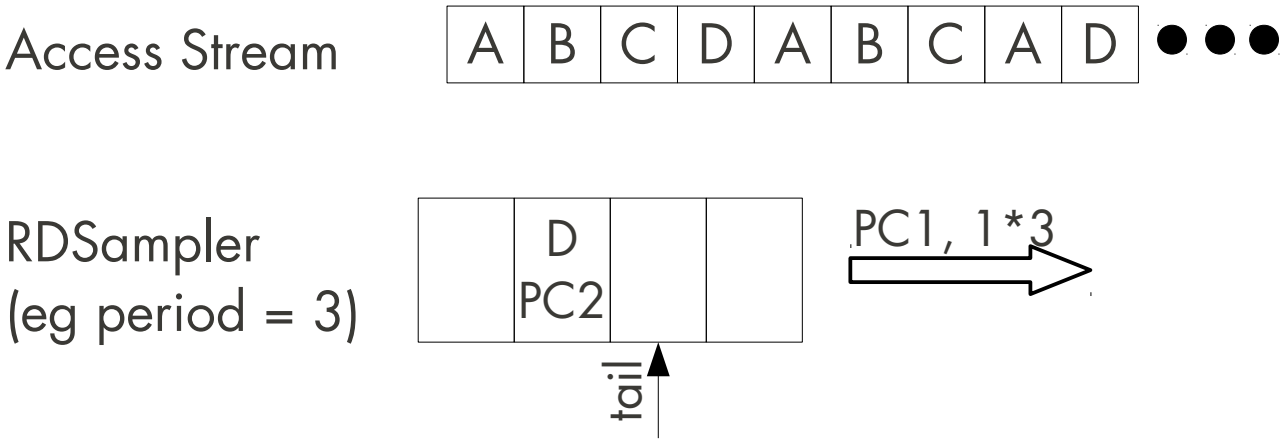
- Reuse-Distance Sampler (RDSampler)
  - On each access, we search the RDSampler for a matching address

Observe: RDSampler



- Reuse-Distance Sampler (RDSampler)
  - On each access, we search the RDSampler for a matching address
    - Match →

Observe: RDSampler

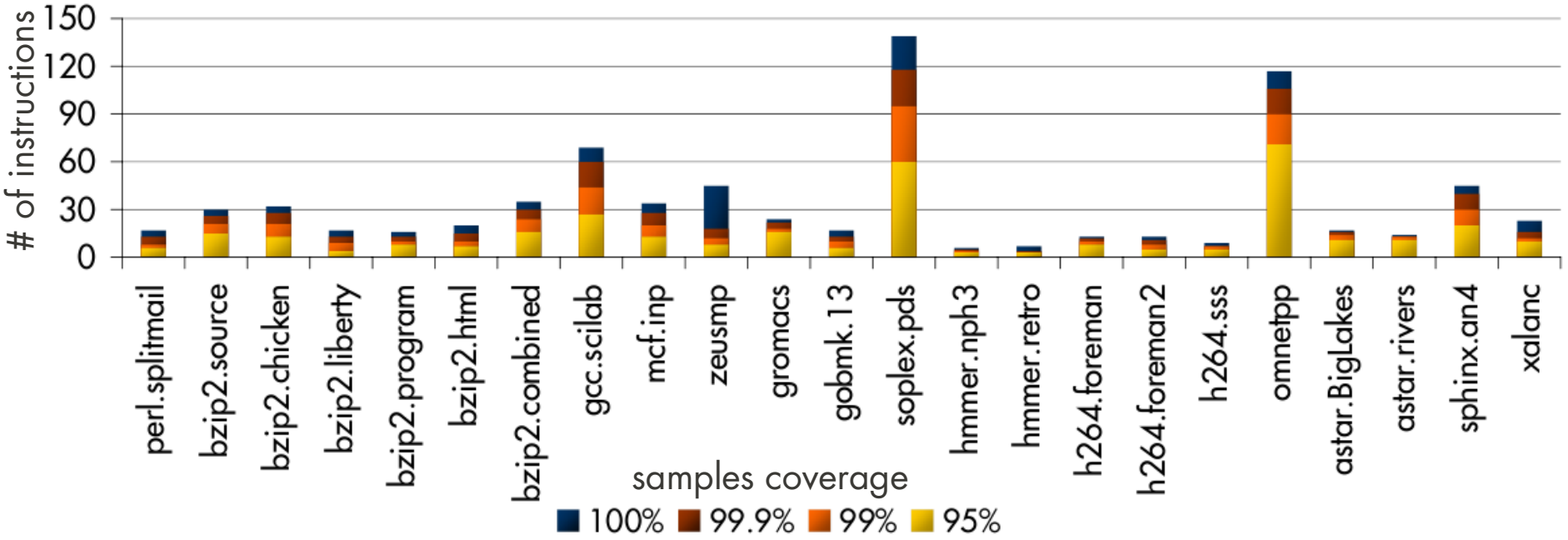


- Reuse-Distance Sampler (RDSampler)
  - On each access, we search the RDSampler for a matching address
    - Match → Produce an instruction-reuse distance pair
    - Reuse Distance = FIFO Position * Sampling Period

Remember

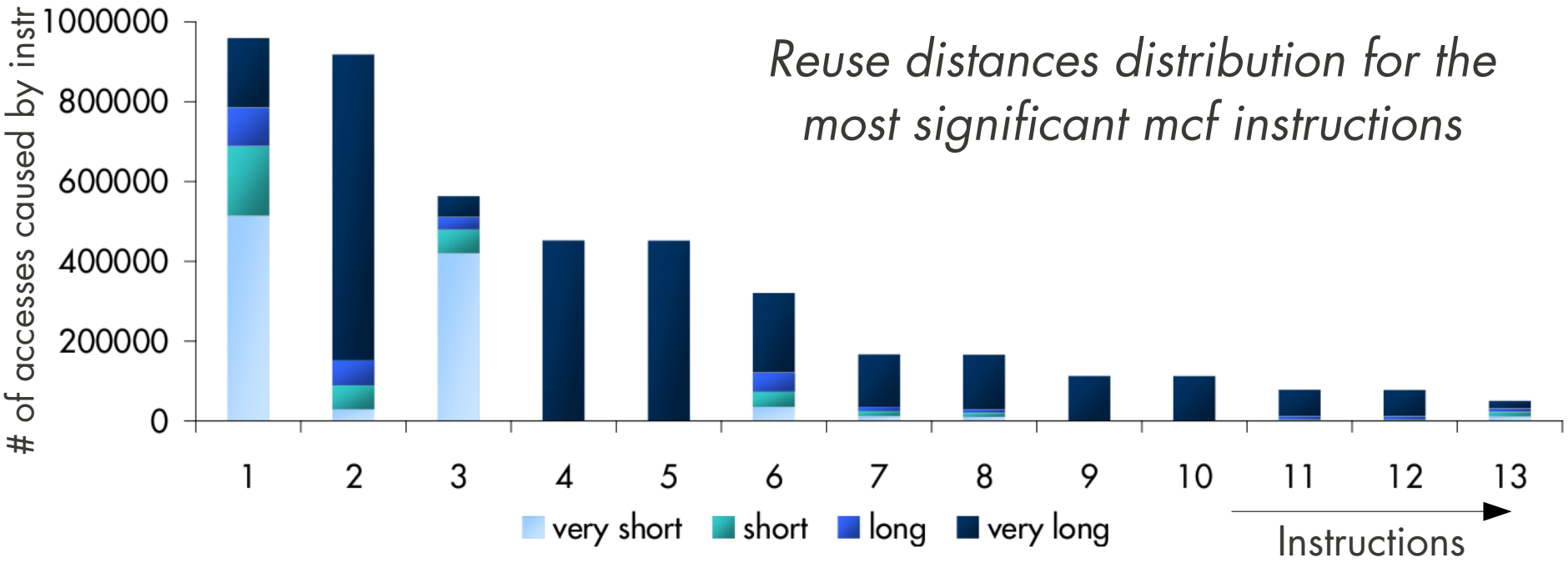
- We don't use this info immediately
  - We need some storage
- We need to predict *the* most likely reuse distance for a given instruction
  - We have to filter the reuse distance information
- How easy is that?

Remember: Storage



- # Instructions sampled by RDSampler
  - Only a few benchmarks have more than 32
  - Most of the accesses are caused by very few instrs

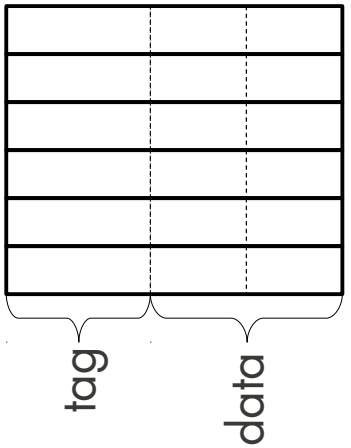
Remember: Filtering



Reuse distances distribution for the most significant mcf instructions

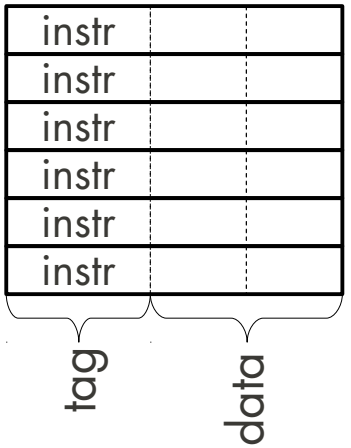
- mcf: not regular access patterns
  - But still: reuse distances correlate well with instrs
  - We only have to remove the noise

Remember: Instruction-based Reuse Distance Predictor (IbRDPredictor)



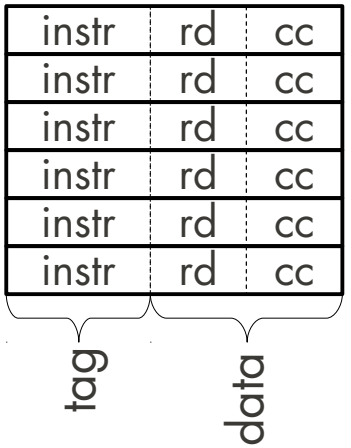
- IbRDPredictor:
  - A small cache-like structure

Remember: Instruction-based Reuse Distance Predictor (IbRDPredictor)



- IbRDPredictor:
  - A small cache-like structure
  - Addressed by the instruction

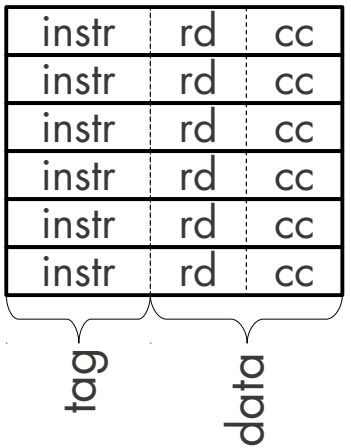
Remember: Instruction-based Reuse Distance Predictor (IbRDPredictor)



- IbRDPredictor:
  - A small cache-like structure
  - Addressed by the instruction
  - Holds the reuse distance of the instruction and a saturating confidence counter

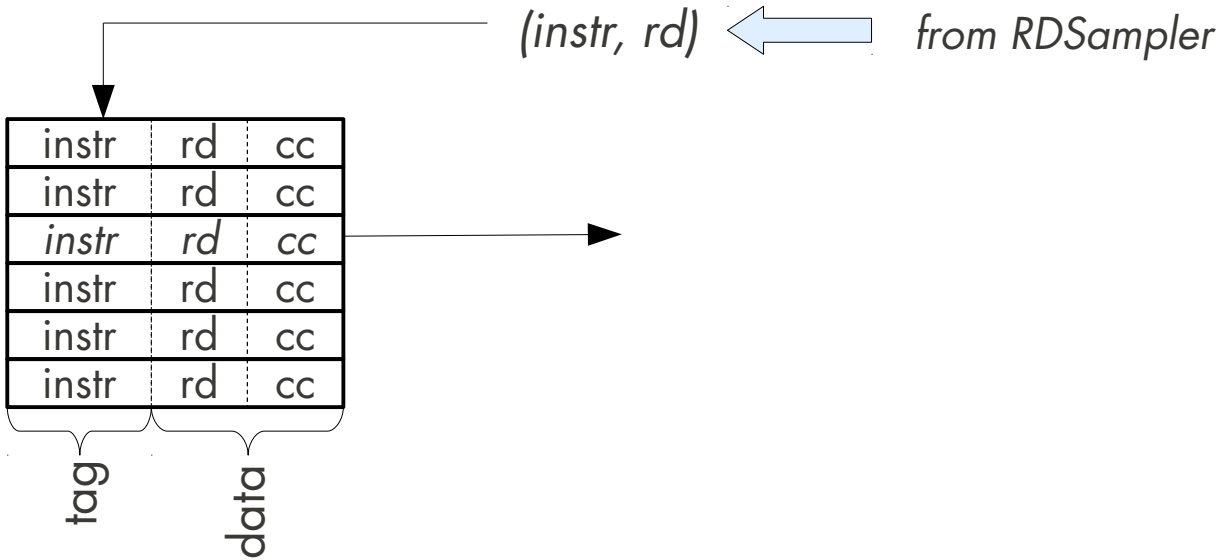
Remember: IbRDPredictor Update

(instr, rd) ⟵ *from RDSampler*



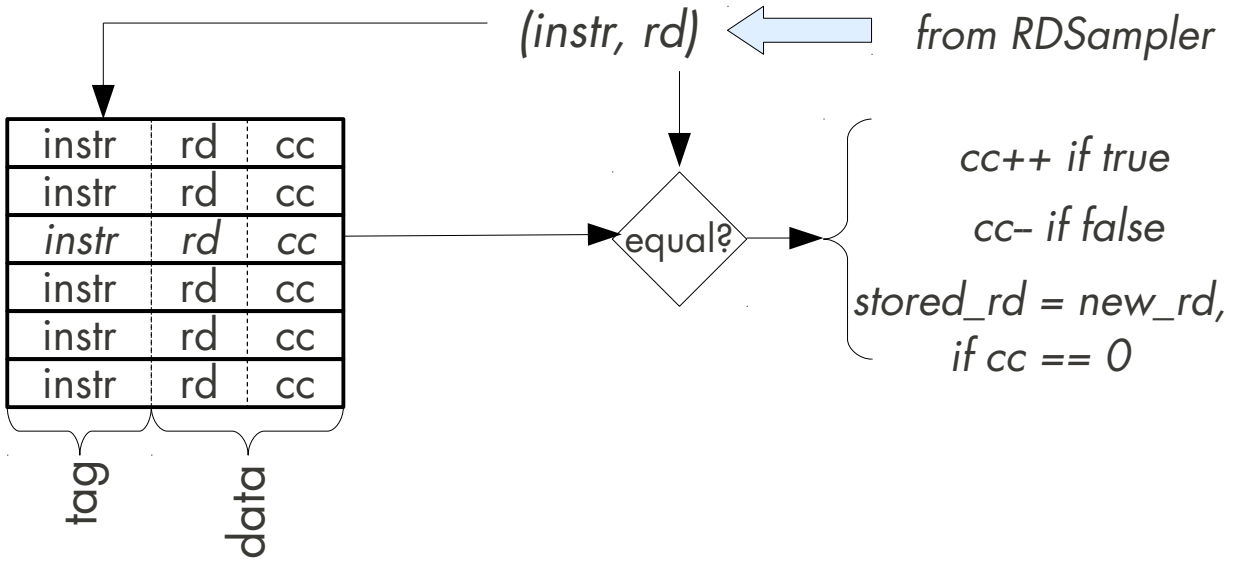
- Two functions
- Update: the sampler provides an instr-rd pair

Remember: IbRDPredictor Update



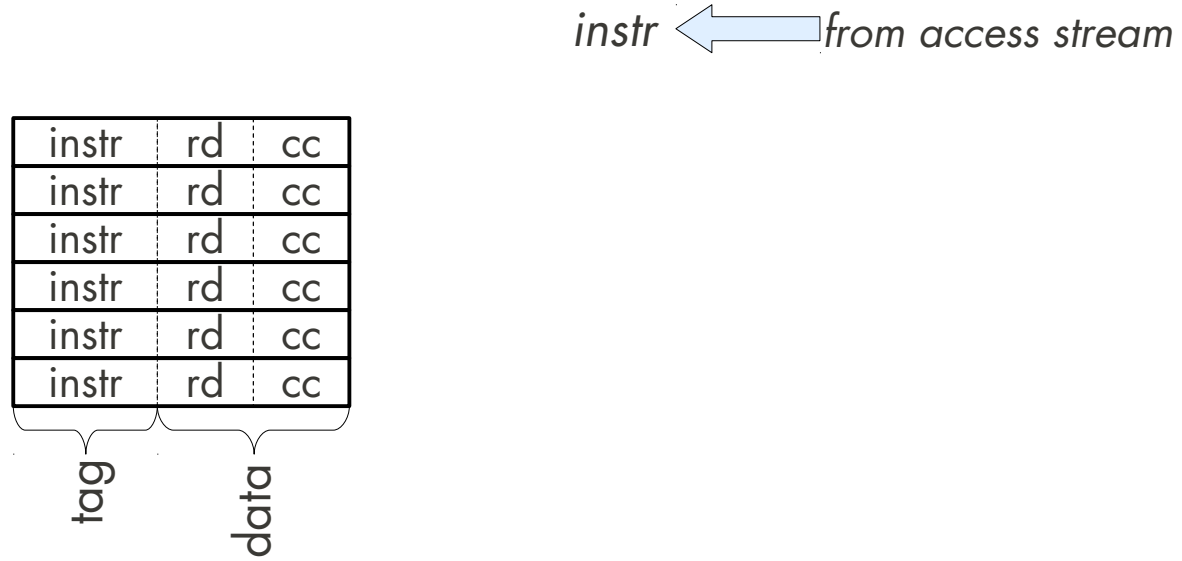
- Two functions
- Update: the sampler provides an instr-rd pair
  - Find the entry for the given instruction

Remember: IbRDPredictor Update



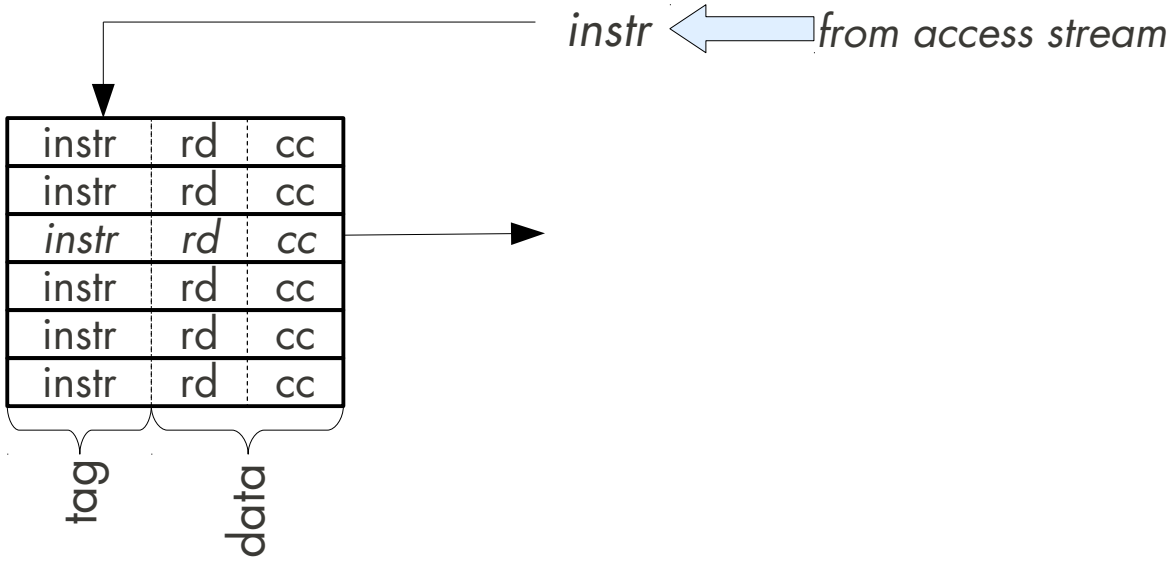
- Two functions
- Update: the sampler provides an instr-rd pair
  - Find the entry for the given instruction
  - If new_rd = stored_rd, increase the confidence
  - If not, decrease it
  - Confidence = 0 → change the stored reuse distance

Remember: IbRDPredictor Lookup



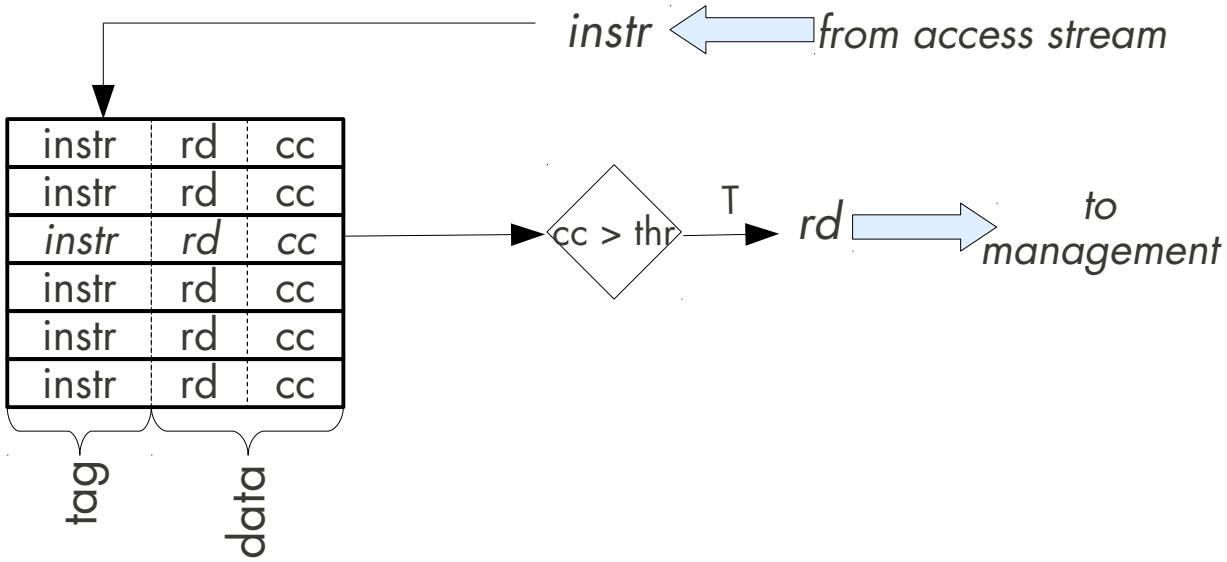
- Two functions
- Lookup: an instruction causes a LLC access & we want to predict its reuse behavior

Remember: IbRDPredictor Lookup



- Two functions
- Lookup: an instruction causes a LLC access & we want to predict its reuse behavior
  - Find the entry for the given instruction

Remember: IbRDPredictor Lookup



- Two functions
- Lookup: an instruction causes a LLC access & we want to predict its reuse behavior
  - Find the entry for the given instruction
  - If entry found & confidence above threshold, return stored reuse distance

Manage

- We are able to predict reuse behaviors
- What can we do with this ability?

Manage – IbRDP Replacement Policy

- Step 1: Approximate Belady's OPT algorithm
  - OPT: replace the line used farthest in the future
  - Stored in each line: reuse distance prediction and time of last access
  - Upon a miss, we evict the line predicted to be used farthest in the future

Manage – IbRDP Replacement Policy

- Step 1: Approximate Belady's OPT algorithm
  - OPT: replace the line used farthest in the future
  - Stored in each line: reuse distance prediction and time of last access
  - Upon a miss, we evict the line predicted to be used farthest in the future

- But:
  - What about the lines with no prediction?
  - What about the lines with wrong prediction?

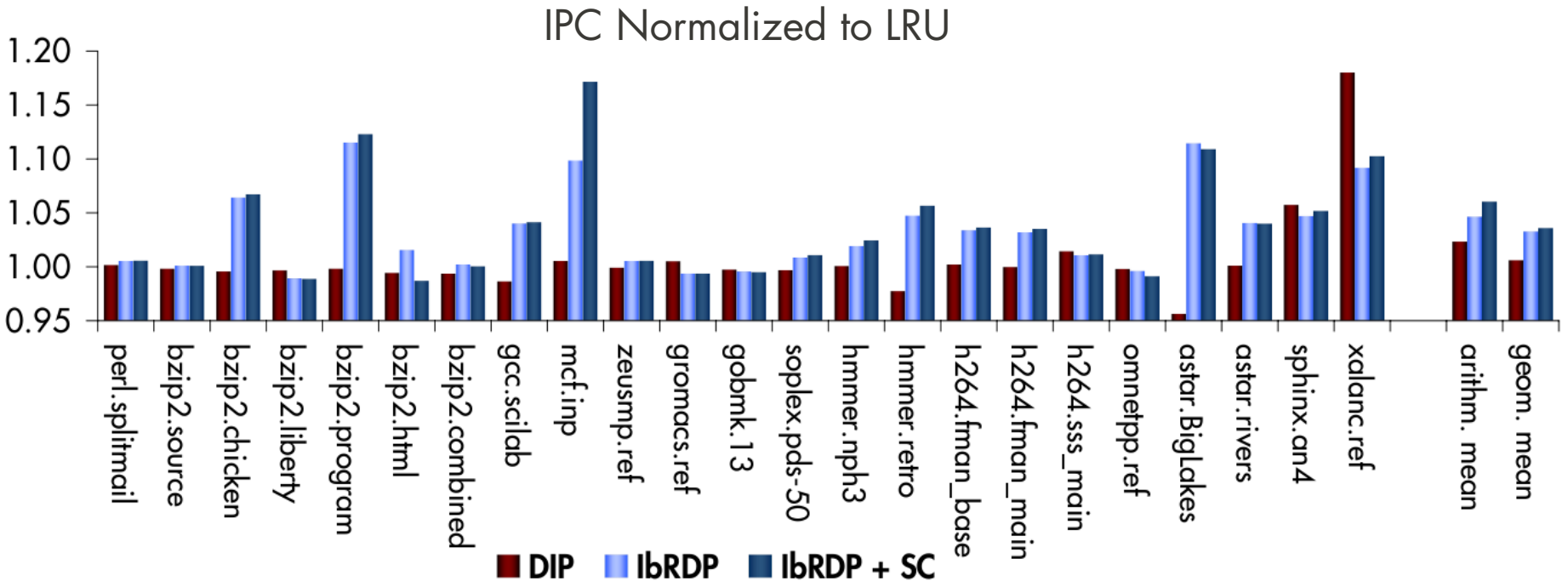
Manage – IbRDP Replacement Policy

- Step 2: Take LRU into account
  - Plan B for lines with no prediction / failsafe for wrong predictions
  - Based on time of last access, choose the LRU line
  - *Not exactly LRU, but close*
- Replace "pseudo-OPT" or "pseudo-LRU"?
  - Replace the line whose last (for pseudo-LRU) or next (for pseudo-OPT) reuse is farthest from the present time

Manage – IbRDP Replacement Policy

- Step 3: Cache Bypassing
  - If the newly fetched line will be reused too far in the future, don't put it in the cache
- Step 1 + Step 2
  - IbRDP Replacement Policy (IbRDP)
- Step 1 + Step 2 + Step 3
  - IbRDP Replacement Policy with Selective Caching (IbRDP+SC)

Evaluation – IPC Improvement



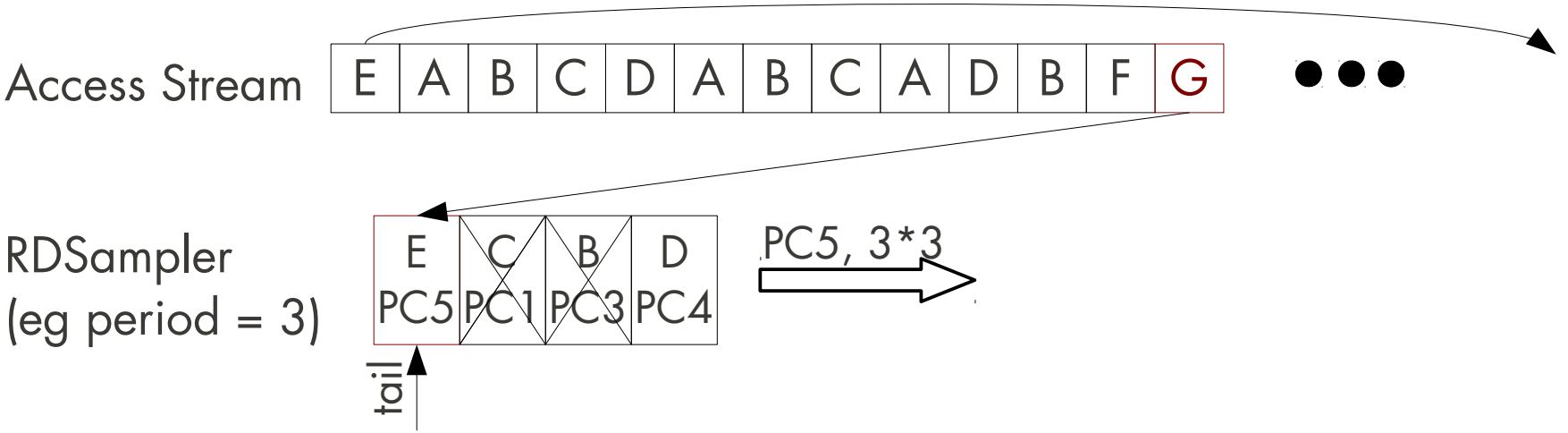
- IbRDP
  - best→+11.5%, worst→-1.1%, avg→+4.7%
- IbRDP+SC
  - best→+16.6%, worst→-1.4%, avg→+6.1%

Conclusions

- Instruction-based Reuse Distance Prediction
  - Offers high predictability of reuse behavior
  - Reasonable hardware cost
- IbRDP Replacement Policy
  - Achieves good speedups
  - Never hurts performance much

Thank You!

Observe: RDSampler

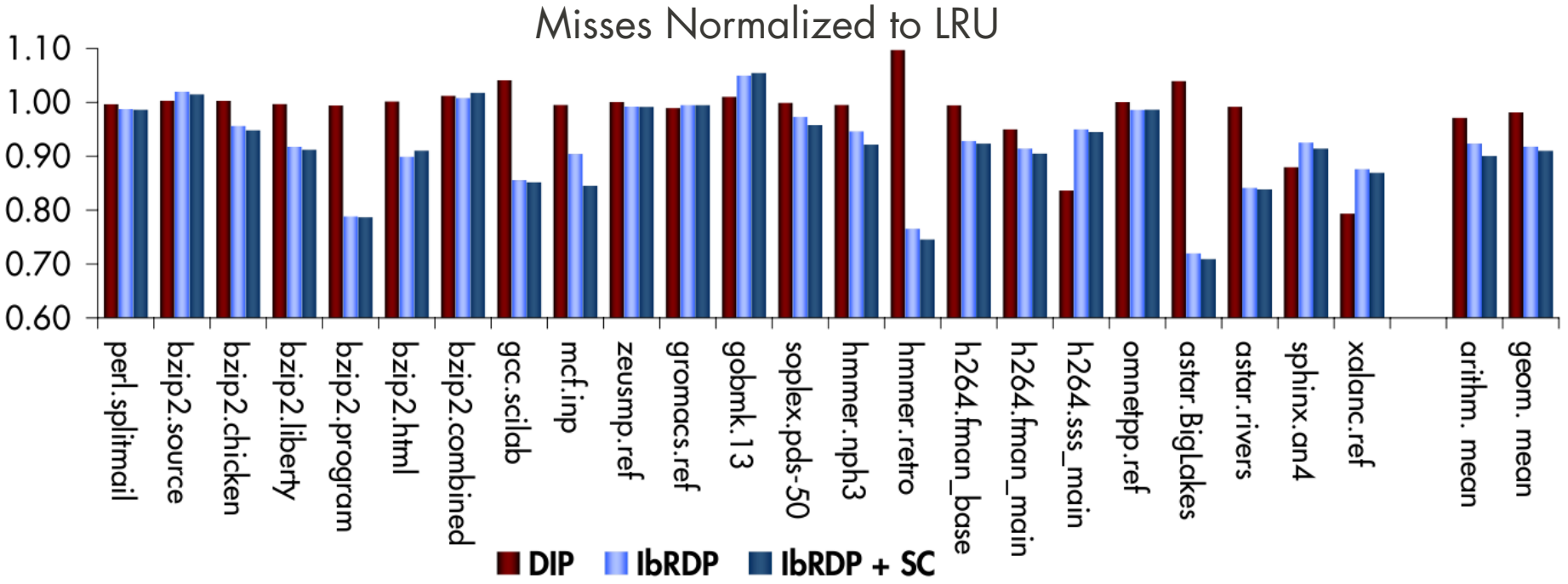


- Reuse-Distance Sampler (RDSampler)
  - When we take a new sample, we replace the old data
  - If still valid, treated as a match
  - Reuse distance = RDSampler's max reuse distance

Practical Implementation

- Available Storage: 129 Kbits
  - To keep full information, we would need more storage
  - On the other hand, we don't really need full info
- Design choices:
  - # of bits for instructions and data addresses
  - Predictor size
  - Quanta for reuse distances & timestamps
  - # of bits for reuse distances and timestamps
  - Sampler size and sampling period
- Storage: 120.4 Kbits

Evaluation – Misses



- IbRDP: best→-28%, worst→+5%, avg→-7.7%
- IbRDP+SC: best→-29%, worst→+5.5%, avg→-10%
- Relative to DIP:
  - Almost always better or equal (exceptions: xalancbmk, h264.sss, gobmk, sphinx)
  - No misses increase as bad as DIP's
  - On Average: -5.7% (IbRDP), -8% (IbRDP+SC)