# Context-Base Computational Value Prediction with Value Compression

Yasuo Ishii

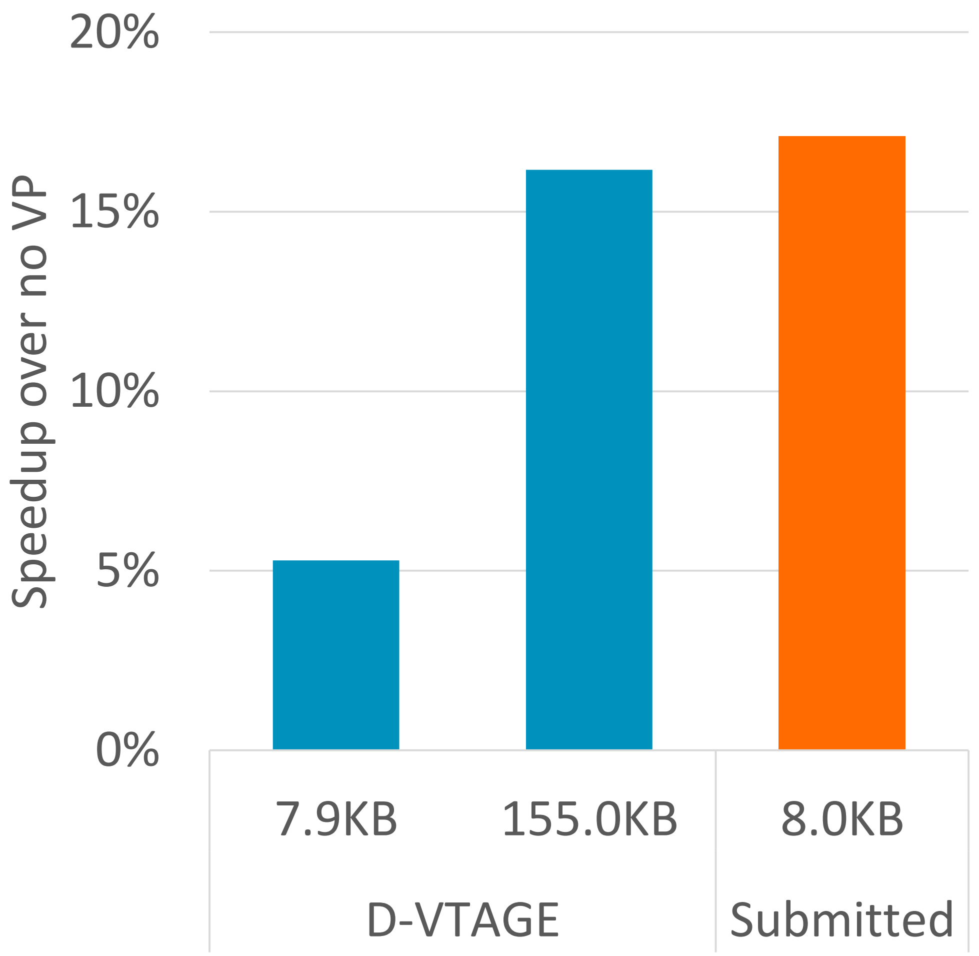
June 3 2018

## Submitted Track: 8KB

arm

## Motivation and Achievement

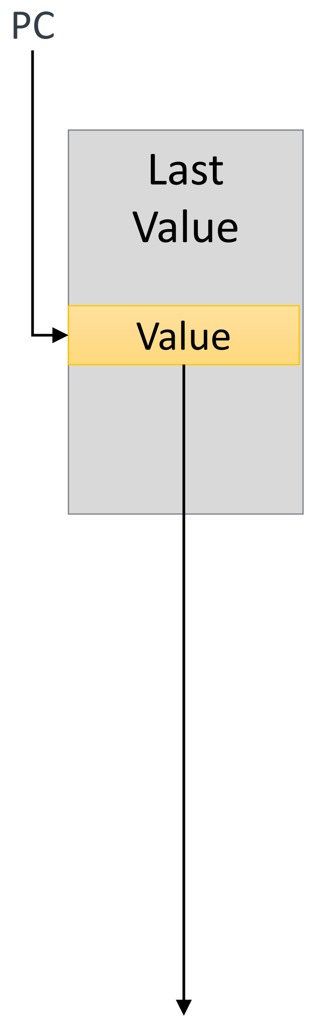
- Known modern value predictors (e.g., D-VTAGE) need large storage to realize noticeable speedup.
- This study proposes 3 techniques to realize comparable speedup with limited storage.
  - Context-base Computational Value Prediction
  - Value Compression Cache with Lazy Allocation
  - Accuracy / Coverage Control
- Proposed predictor realize large 155KB D-VTAGE equivalent performance with 8KB budget.
  - 17.1% speedup for 135 CVP-1 traces



arm

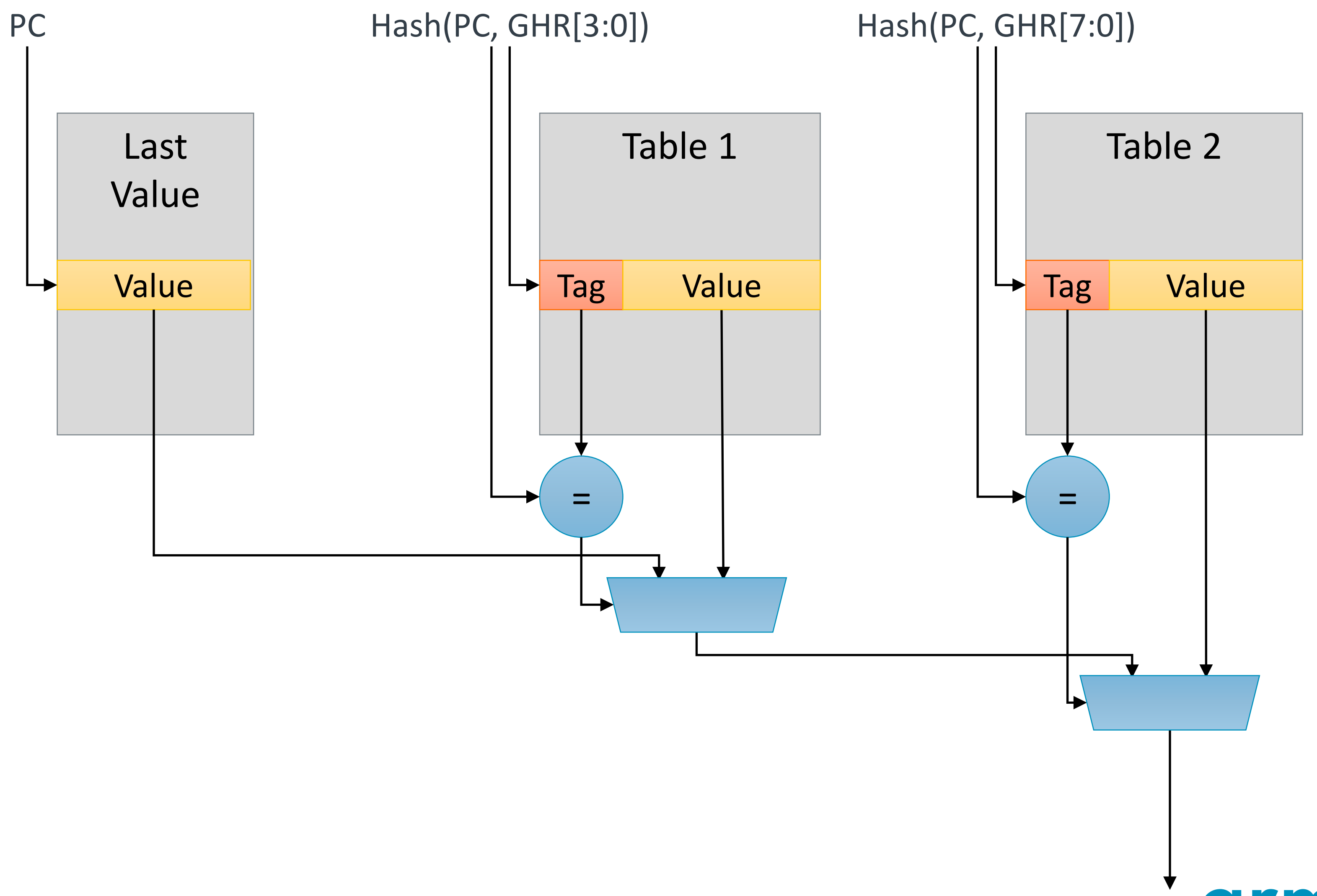
## Context-Base Computational Value Prediction

## Prior Work 1: Last Value Predictor [Lipasti+ 1996]

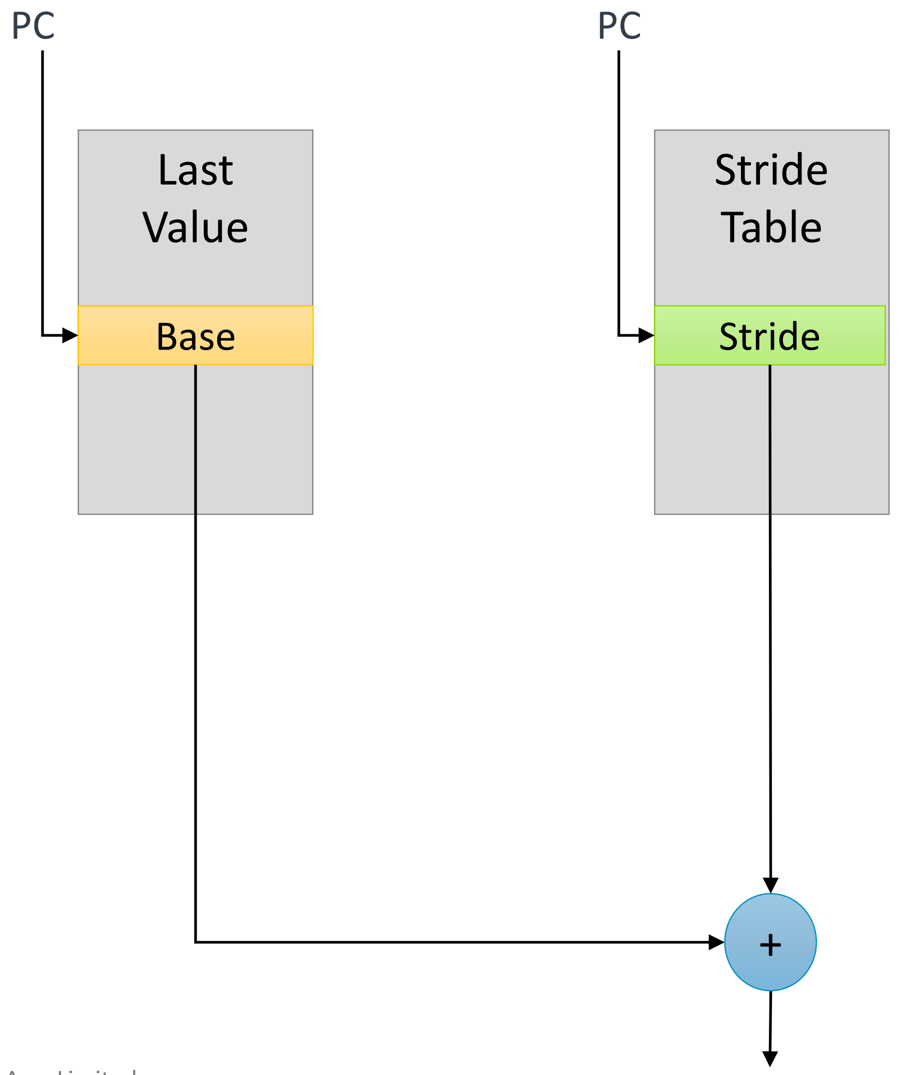


Each entry can be tagged by PC

# Prior Work 2: Value TAGE Predictor (VTAGE) [Perais+ 2014]

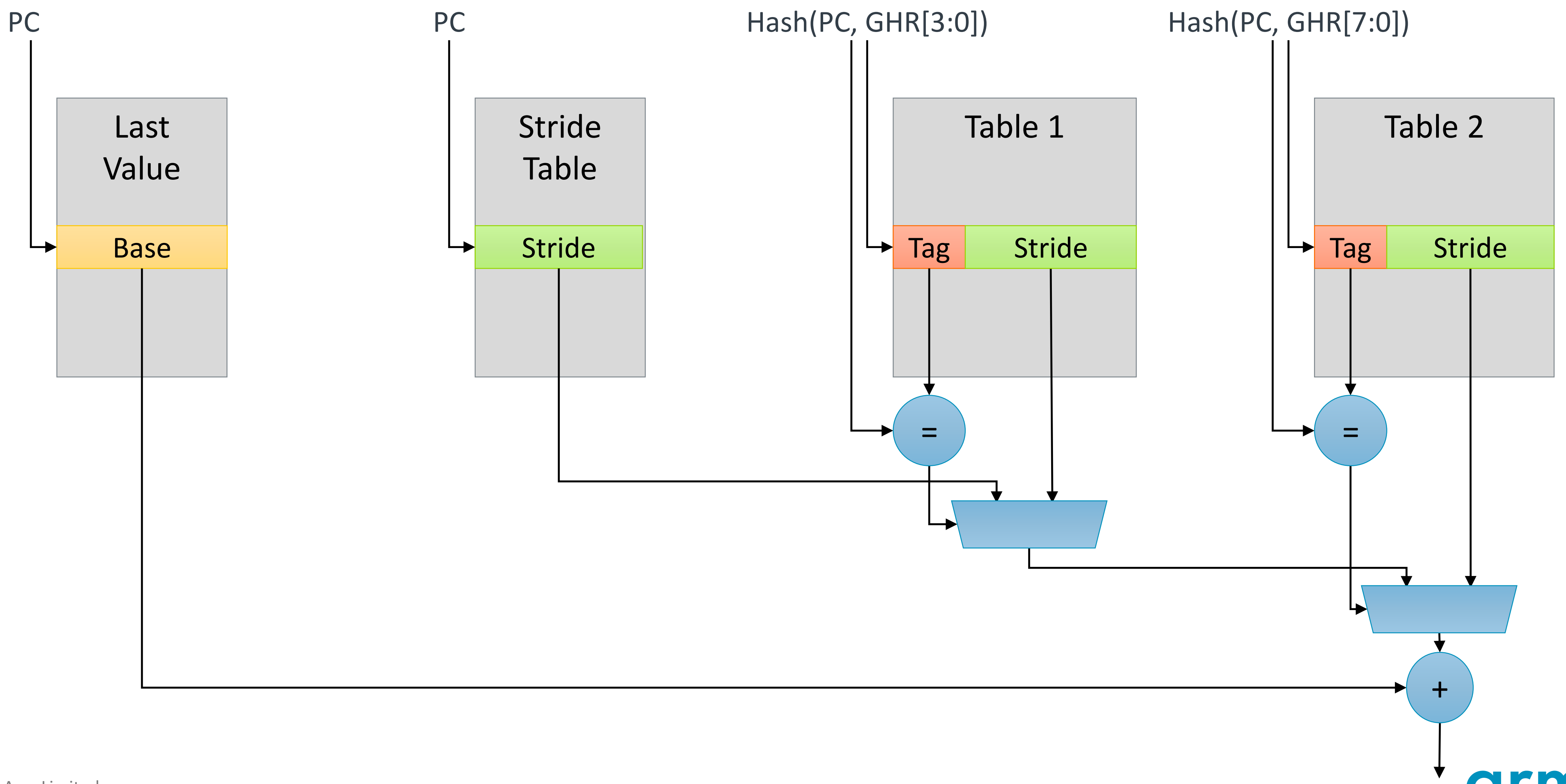


## Prior Work 3: Stride Value Predictor [Gabbay+ 1996]



Each entry can be tagged by PC

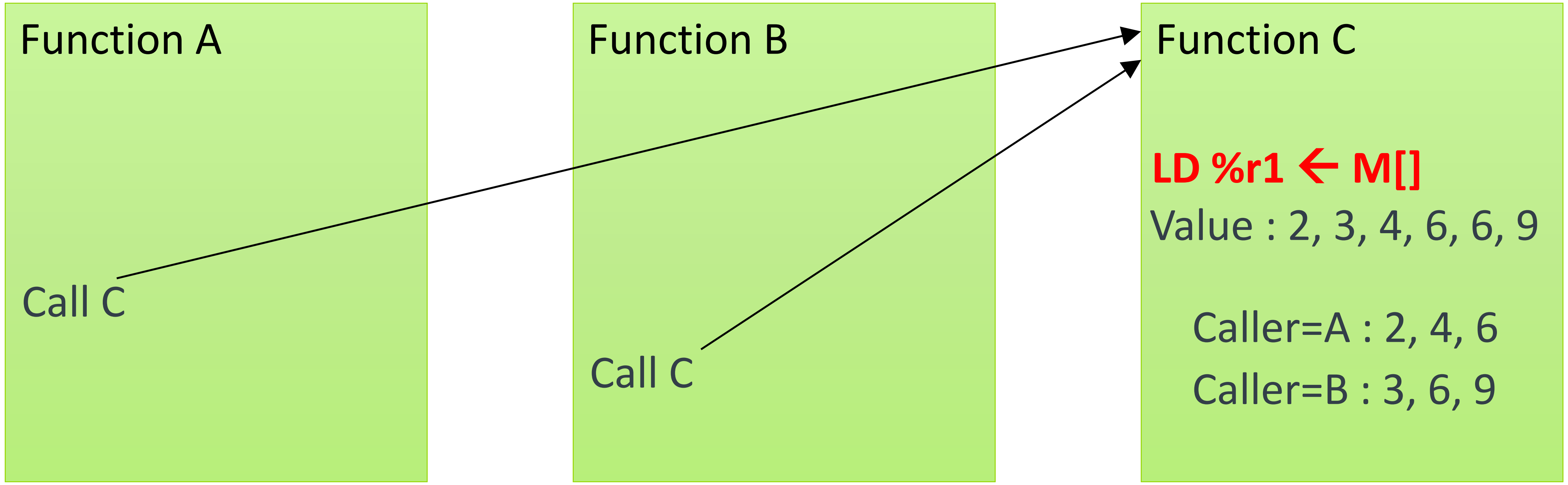
## Prior Work 4: Differential VTAGE (D-VTAGE) [Perais+ 2015]



## Prior Work: Prediction Algorithm Summary

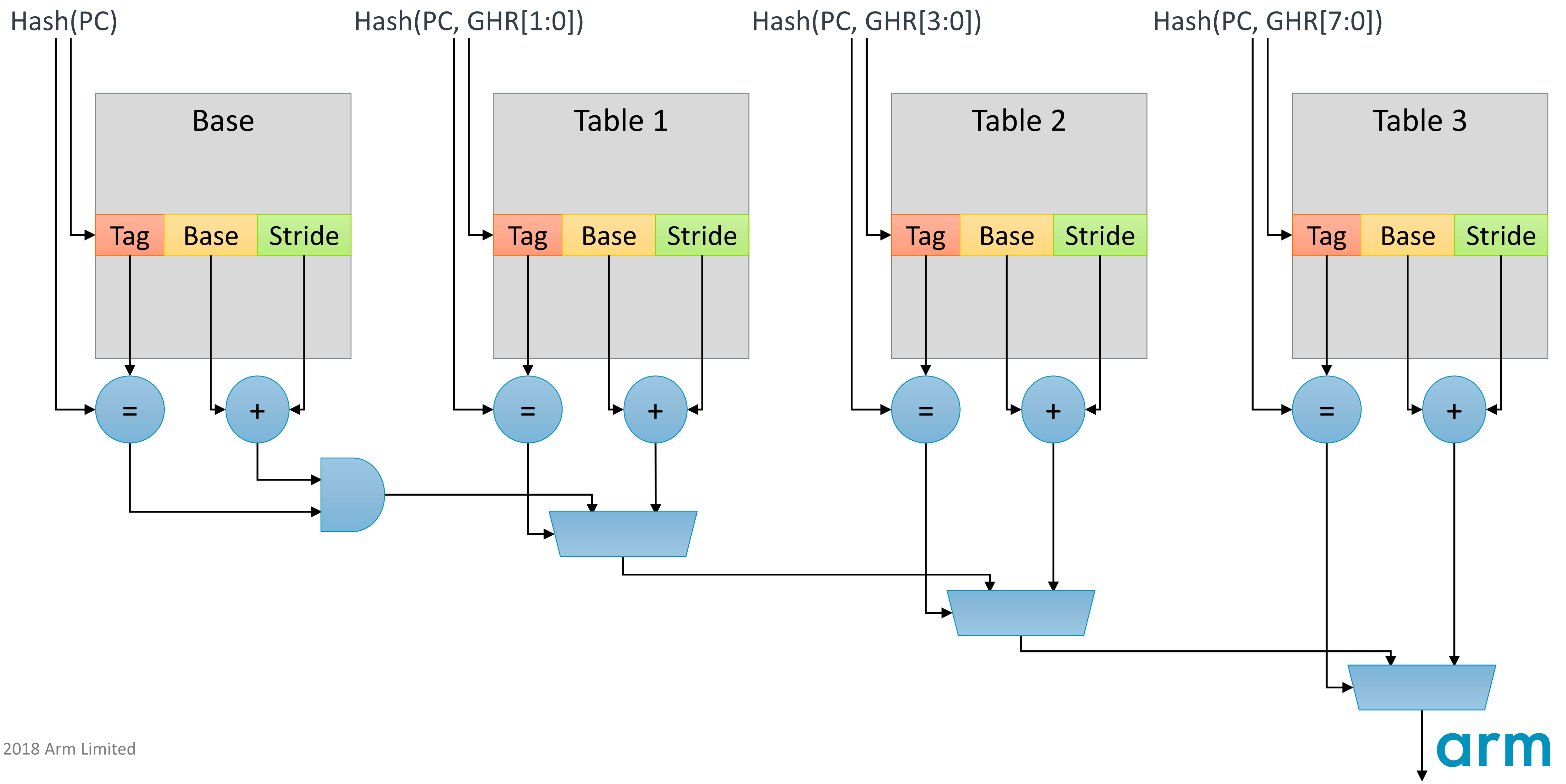
| | **Last Value Prediction** | **Delta Prediction** |
|---|---|---|
| Last Value | PC-Localized | NONE |
| Stride Value | PC-Localized | PC-Localized |
| VTAGE | **Multi-level History Correlated** | NONE |
| D-VTAGE | PC-Localized | **Multi-level History Correlated** |

# Motivation of New Value Prediction Algorithm



To predict this load instruction in function C, both last value and stride need to be correlated with context information.

# Context-base Computational VTAGE (CBC-VTAGE)



# Prediction Algorithm Comparison

| | **Last Value Prediction** | **Delta Prediction** |
|---|---|---|
| Last Value | PC-Localized | NONE |
| Stride Value | PC-Localized | PC-Localized |
| Per-path Value<br>Per-path Stride | 1-level<br>History Correlated | 1-level<br>History Correlated |
| VTAGE | **Multi-level<br>History Correlated** | NONE |
| D-VTAGE | PC-Localized | **Multi-level<br>History Correlated** |
| **CBC-VTAGE** | **Multi-level<br>History Correlated** | **Multi-level<br>History Correlated** |

## Other Optimizations

- Update policy
  - At update time, ALL tag hit entries with non-zero stride will be updated.
  - This is mandatory update policy change for CBC-VTAGE
- Bank-interleave
  - Based on PC value, each tagged component can be correlated with different branch history length
  - Some branch predictors (e.g., ISL-TAGE) used bank interleave to improve storage efficiency [Seznec 2011]
- No Tagless Components
  - All components including base components are (partially) tagged.
  - Some existing predictors correlate branch history length=0 to tagged components to enhance tagless component capability [Ishii+ 2011], [Ishii 2014], [Seznec 2011].

## Value Compression Cache

arm

## Value Locality and Value Compression

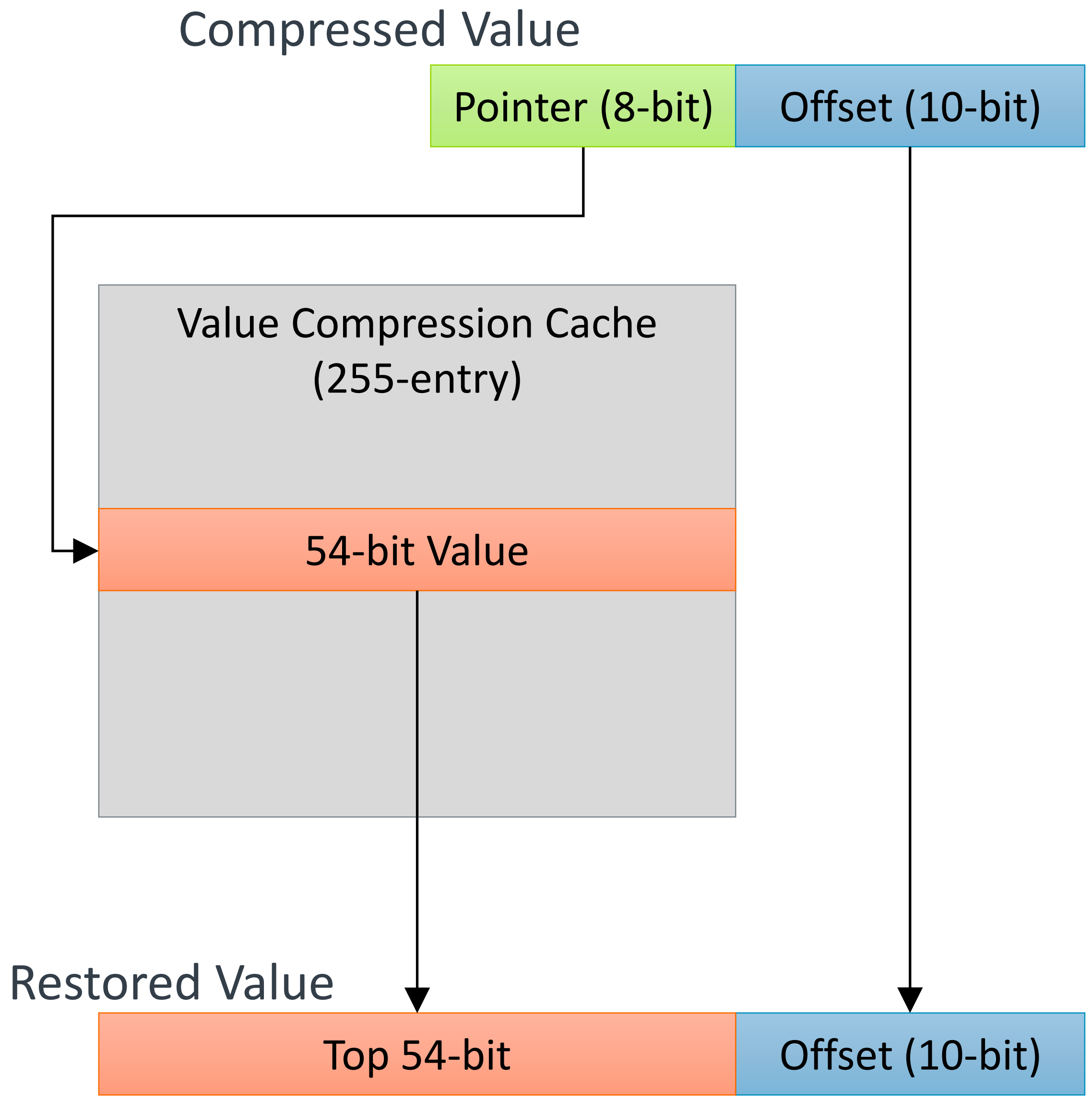
- CBC-VTAGE entry is wider than other predictors

- Key Idea: utilize spatial locality to compress the value
  - Integer value tends to be close to 0 (e.g., -1. 0, 1, 2, …)
  - Address value tends to point small subset of huge memory address space

- Known Approach: Region Cache [Seznec 2011]
  - This was originally proposed for indirect predictor in Championship Branch Prediction  in 2011.
  - Replace top n-bit by pointer to the buffer
  - The buffer hold up to $2^n$-entry full associative buffer

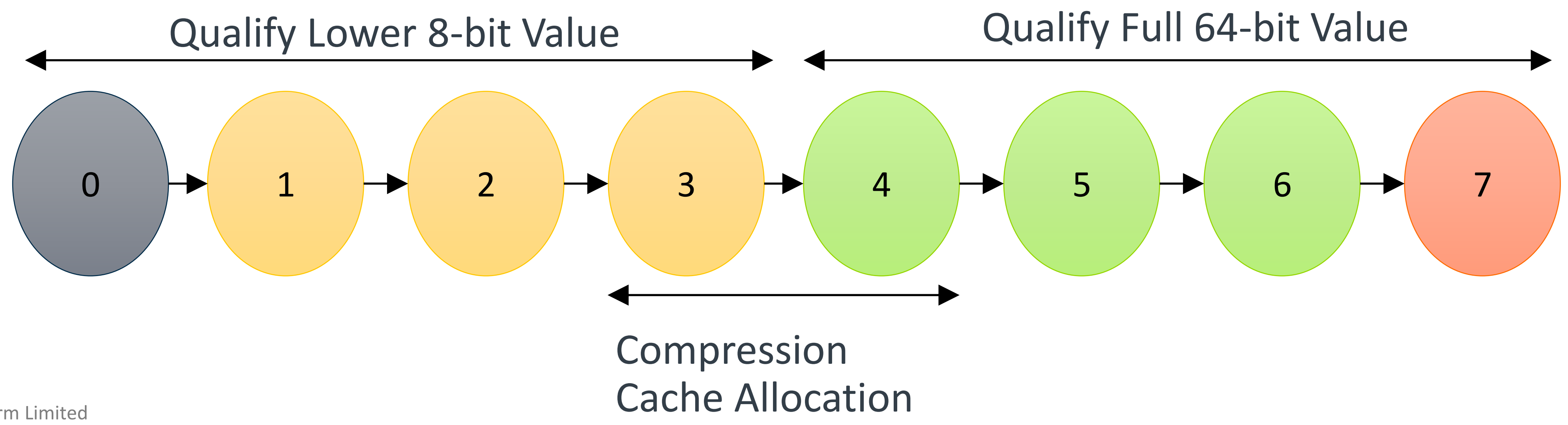
# Value Compression Cache

- To exploit spatial locality in register values, we introduced value compression cache
- Instead of higher 54-bit value, each entry tracks 8-bit pointer and lower 10-bit offset.
- 255-entry full associative buffer
  - 2/4-way skewed associative is good enough for real HW
  - 256th entry is reserved for 0.
- Full 64-bit value is restored during lookup



## Lazy Allocation to Exclude Useless Entries

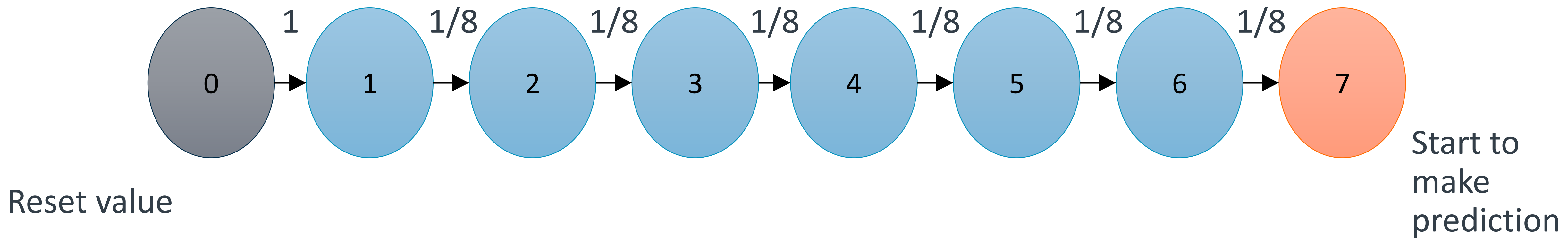
- Non-negligible number of prediction entries are evicted before they started to make prediction (before confidence counter got saturated).
  - If the counter doesn't make prediction, it just wastes the allocated resource.
- To mitigate this issue, we allocate compression cache entry only when confidence reach a certain threshold.



## Coverage / Accuracy Control

## Prior Work : Forward Probabilistic Counter [Riley+ 2006]

- Prediction entry employs confidence counter to measure the training progress
  - To improve performance by value prediction, the prediction accuracy must be high (~99%) for CVP-1 environment because misprediction penalty is significant.
  - To realize very high prediction accuracy, confidence counter should be very wide.
- To mimic wide confidence counter, forward probabilistic counter is utilized.
  - In this study, we utilize 3-bit forward probabilistic counter as the default confidence estimator
  - If each transition can happen 1/8, 3-bit counter can almost emulate 6-bit counter.



## Optimization 1: Static Forward Probability Optimization

- Some existing papers [Sheikh+ 2017] indicated that applying more resource for 'high-cost' instructions may improve value predictor performance.
- In this study, we change the forward probability based on instruction type.

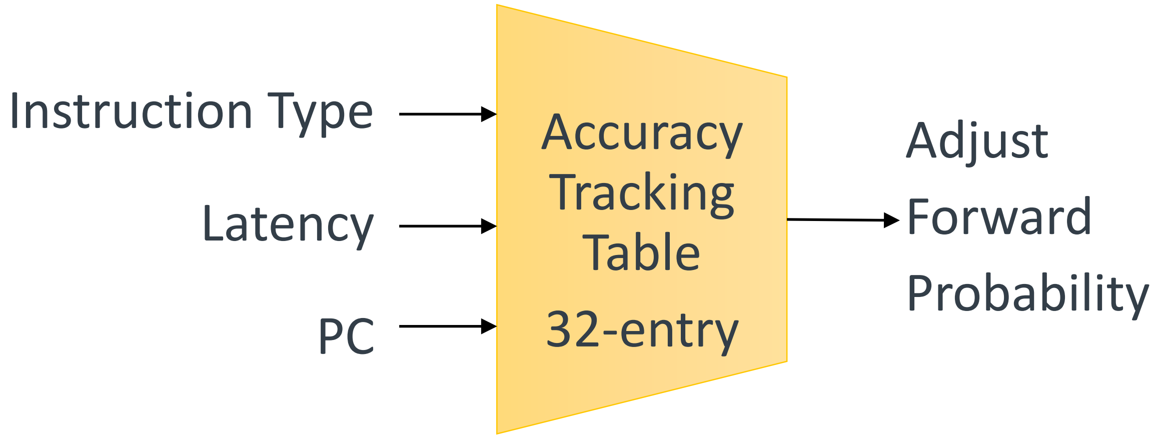
| Instruction Type | Latency (cycle) | Forward Probability (Stride = 0) | Forward Probability (Stride != 0) |
|---|---|---|---|
| ALU operation | 1 | 3.5% (=9/256) | 10.2% (=26/256) |
| FP operation | 4 | 28.1% (=72/256) | 81.2% (=208/256) |
| Load (L1 hit) | 3 | 45.7% (=117/256) | 100% |
| Load (L1 miss) | >12 | 59.8% (=153/256) | 100% |
| Load (L2 miss) | >60 | 66.8% (=171/256) | 100% |

## Optimization 2: Adaptive Forward Progress Counter

- Even though static optimization works, the best forward probability is different for each workload. To adjust accuracy / coverage balance, direct map 32-entry accuracy tracking table is employed.

- This table reduces forward probability if prediction accuracy is lower than threshold.

- Accuracy for each instruction type measured by forward probabilistic counter.



| Instruction Type | Target Accuracy |
|---|---|
| Load | 98.8% (=253/256) |
| Other | 97.3% (=249/256) |

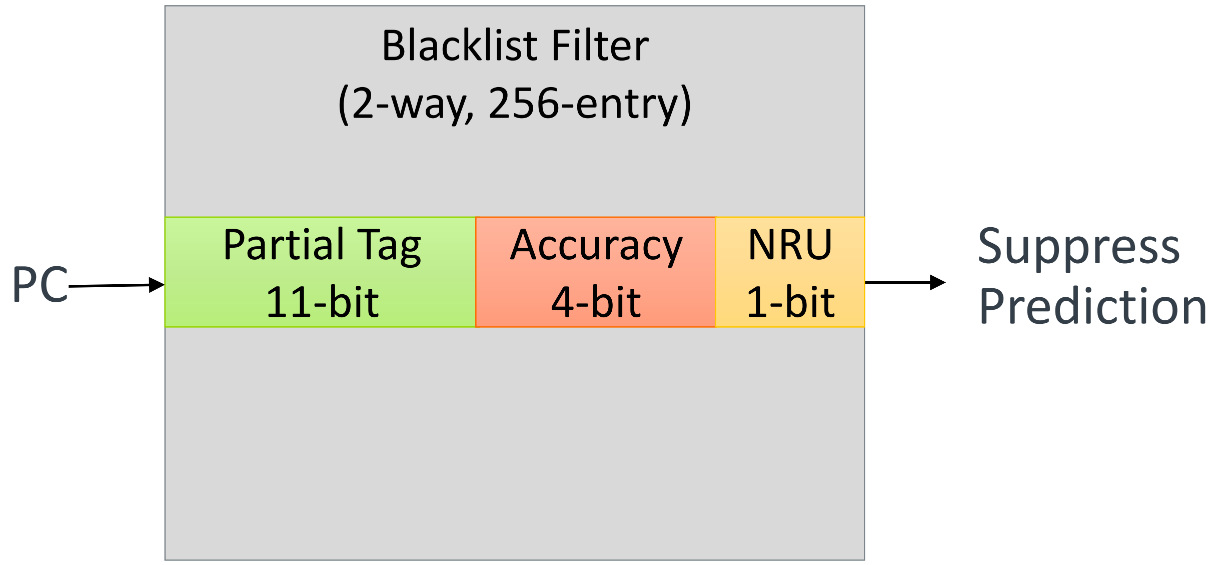
NOTE: Even if target accuracy of Load instruction is higher than the others, overall forward probability of load instructions are higher than the others since default forward probability is much higher for load instructions.

## Optimization 3: Blacklist Filtering

- Even while predictor can realize high accuracy, few instructions can still make mispredictions.

- Blacklist filter is employed to exclude these outliers from making prediction.

- If accuracy of value prediction from given PC is lower than threshold, the value predictor stops to make prediction.



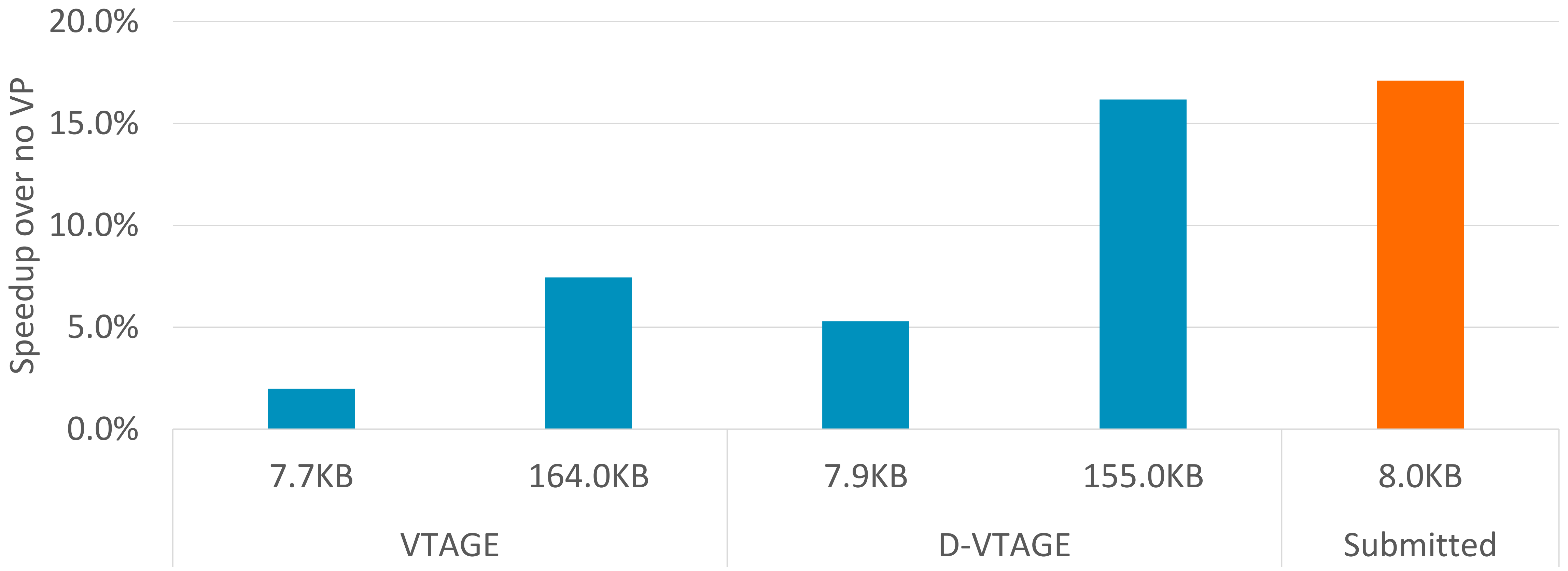
| Instruction Type | Threshold Accuracy |
|---|---|
| Load | 95.7% (=245/256) |
| Other | 96.1% (=246/256) |

## Evaluation

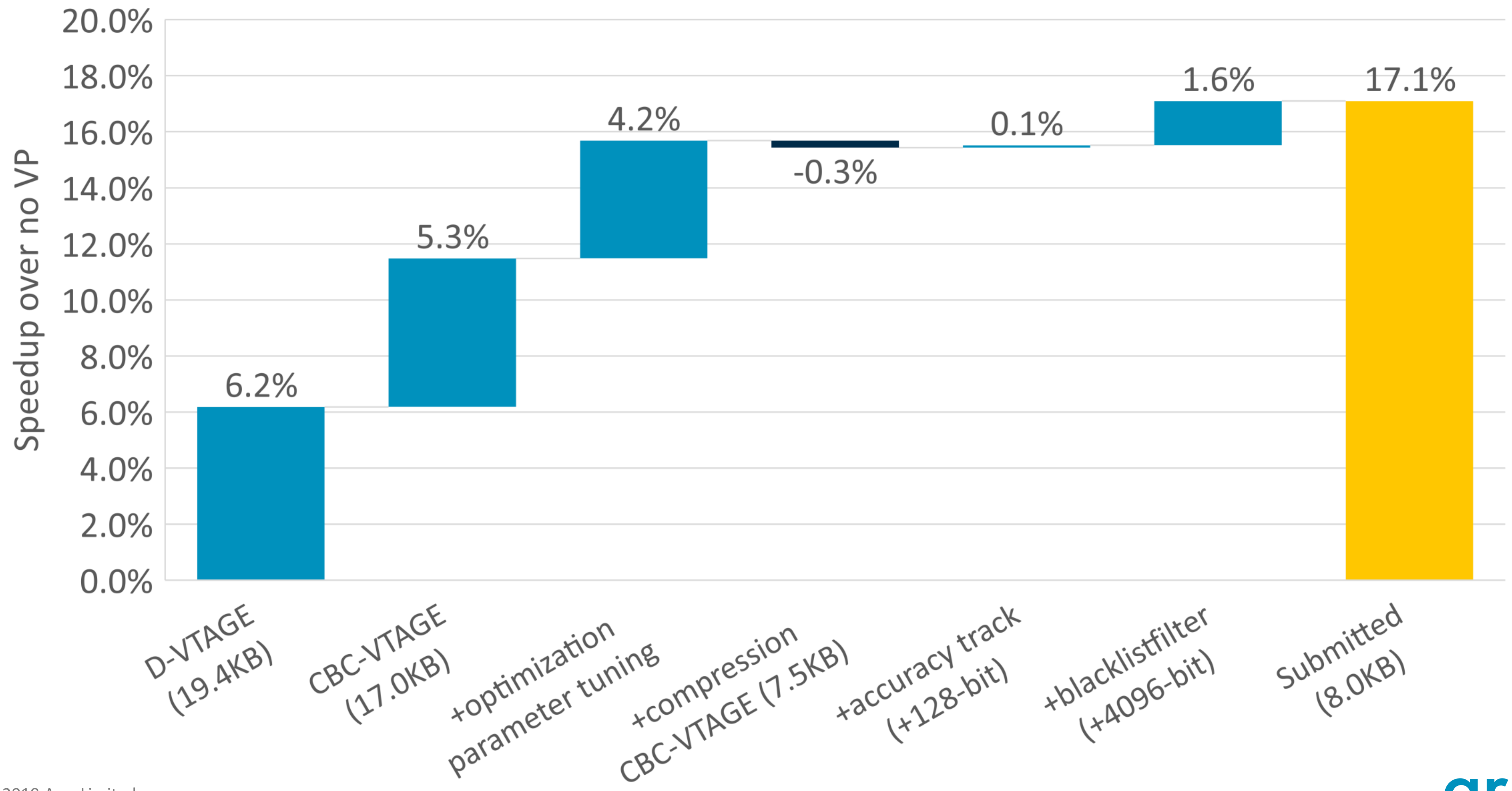
## Budget Counting (total : 65403-bit)

| Resource | Configuration | Bit Count |
|---|---|---|
| CBC-VTAGE | 8-tables, 128-entry per table<br>13-bit tag, 3-bit confidence,<br>2-bit usefulness, 28-bit value | 47104-bit |
| Value Compression Cache | 255-entry, full associative<br>54-bit value, 1-bit NRU for CLOCK | 14025-bit |
| Accuracy Tracking Table | 32-entry, direct-map<br>4-bit forward probabilistic counters | 128-bit |
| Blacklist Filter | 256-entry, 2-way skewed associative<br>11-bit tag, 4-bit counter, 1-bit NRU | 4096-bit |
| Miscellaneous | Random seed (32-bit)<br>Global Tick counter (10-bit)<br>CLOCK pointer (9-bit) | 51-bit |

## Speedup Over No Value Prediction



## Speedup Breakdown



## Summary

## Summary

- Context-base Computational Value Prediction by TAGE inspired predictor (CBC-VTAGE)
  - captures computational depending on context patterns efficiently
  - it can outperform similar size D-VTAGE predictor by 5.3%.

- Value Compression Cache to exploit value locality
  - Lazy allocation helps to exclude useless values from compression storage
  - Compressed the storage size by ~55% with 0.3% slowdown

- Accuracy / Coverage Control
  - Static forward probability optimization is very important to maximize the performance (4.2% performance)
  - Accuracy tracking table / blacklist filtering can push additional 1.7% performance with reasonable HW cost