CS-773 Paper Presentation

Improving the Utilization of Micro-operation Caches in x86 Processors

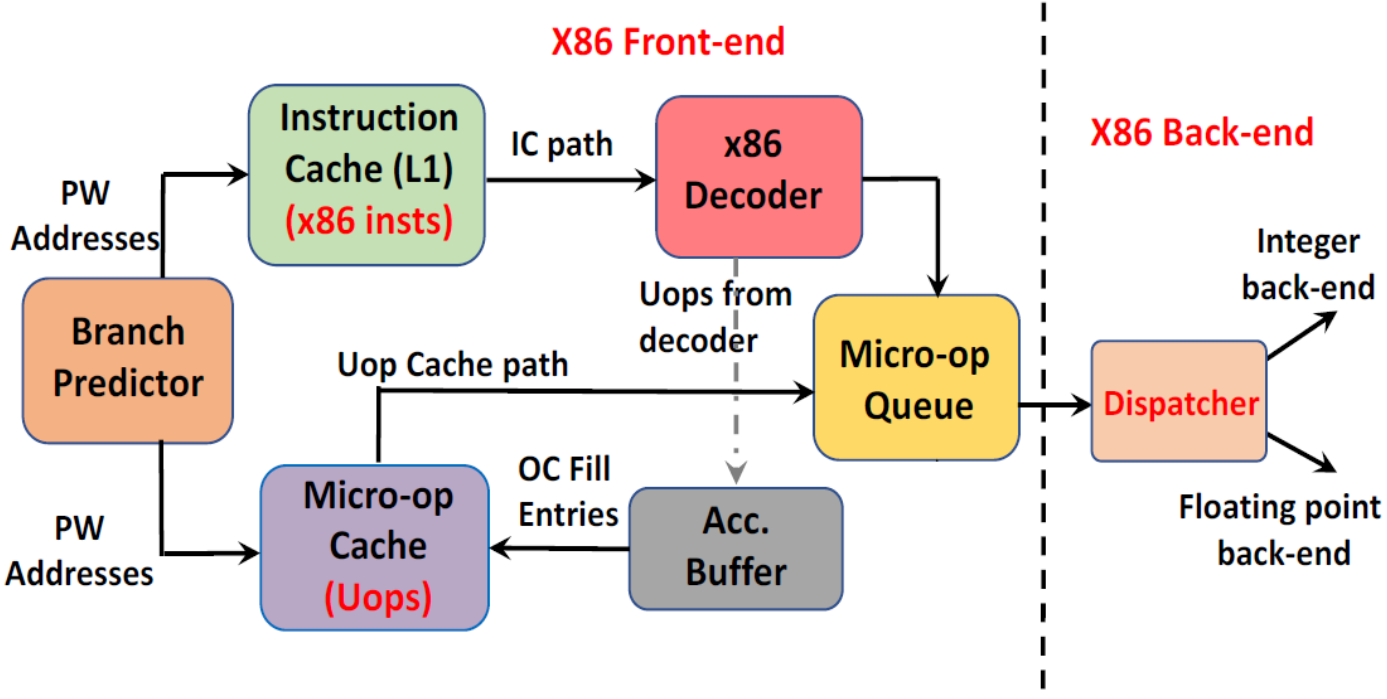
Sumon Nath
Hyperthreads(#6)
sumon@cse.iitb.ac.in

Some of the figures are adapted and modified from the paper

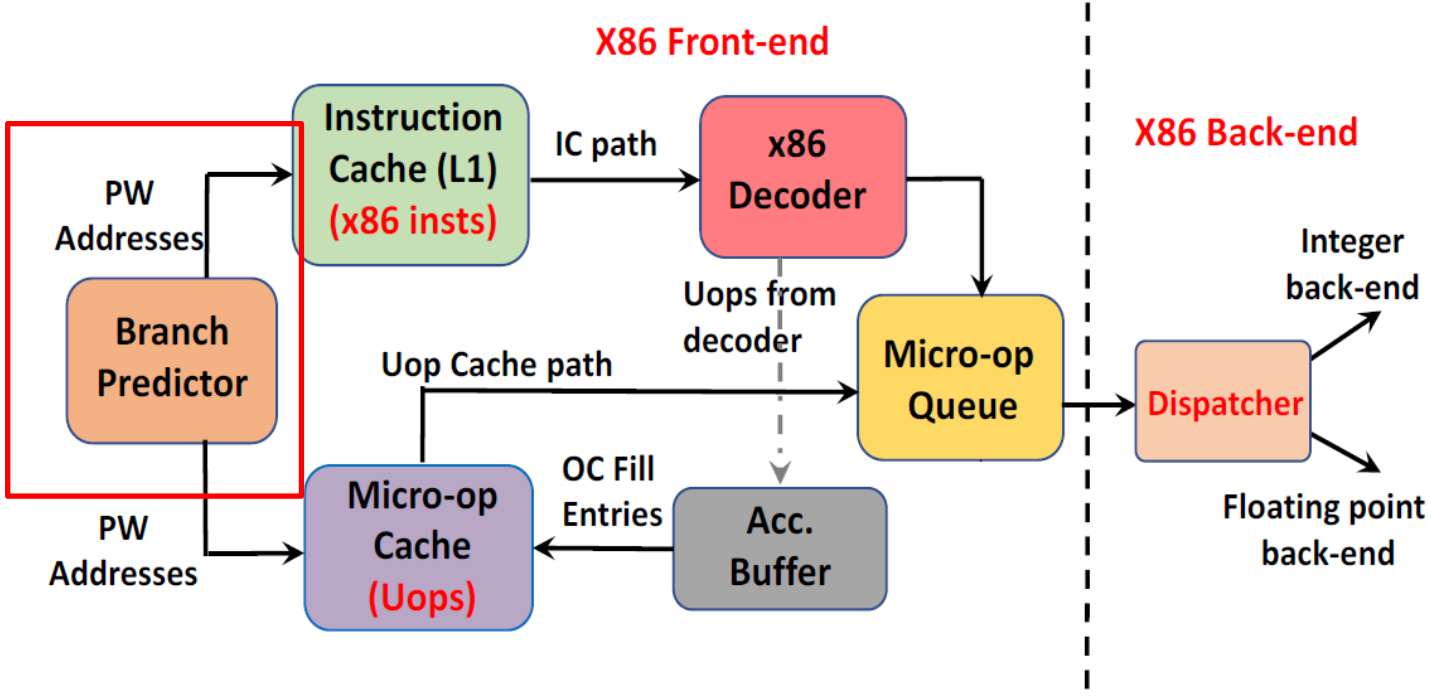
Coming up

- Background - Micro op cache
- Impact of Micro op cache
- Motivation - Fragmentation
- Solutions proposed
- Conclusion

Background: CPU frontend

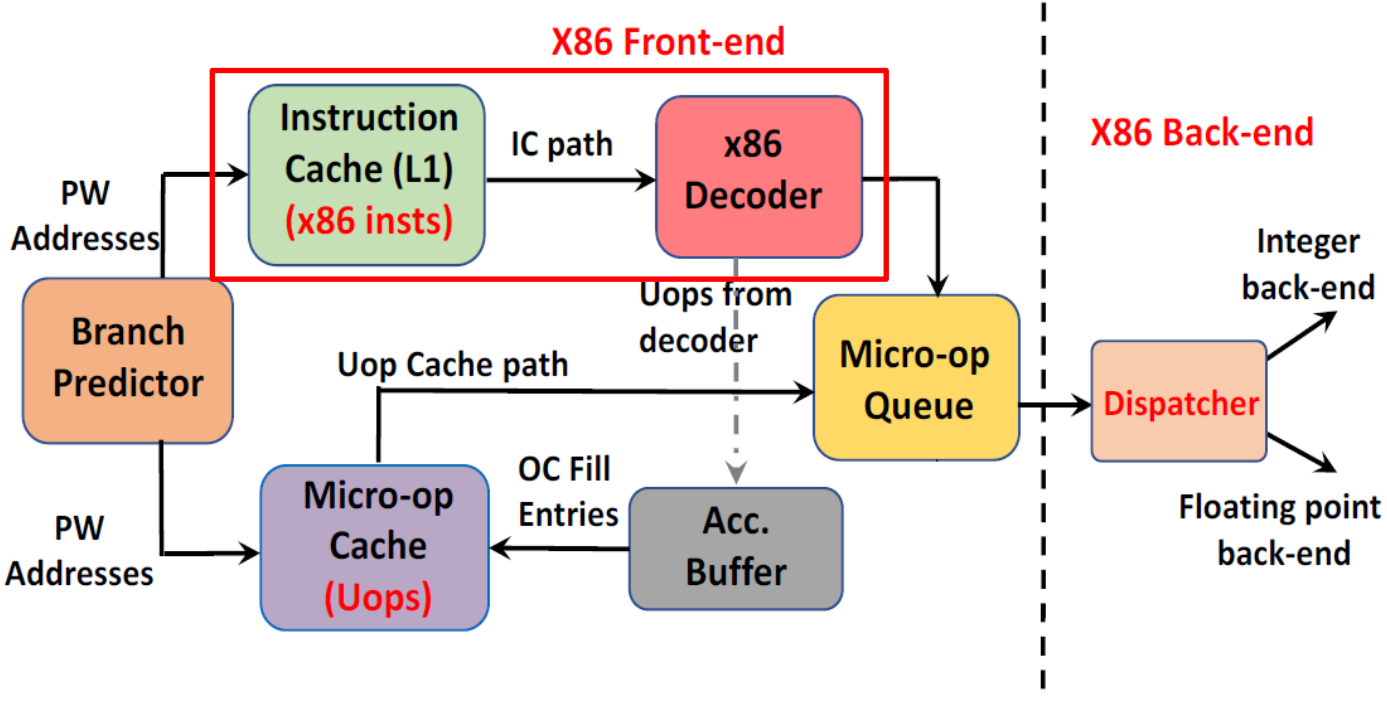


Background: CPU frontend



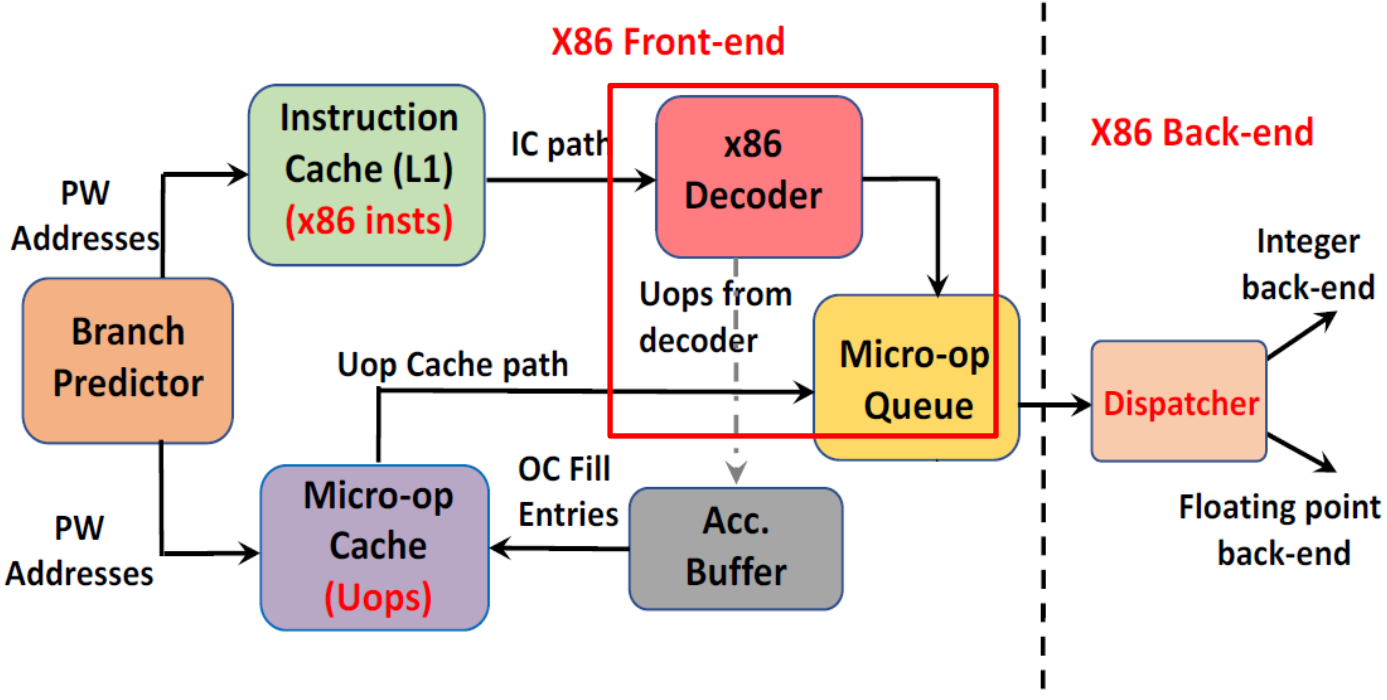
Range of address ~ basic block

Background: CPU frontend



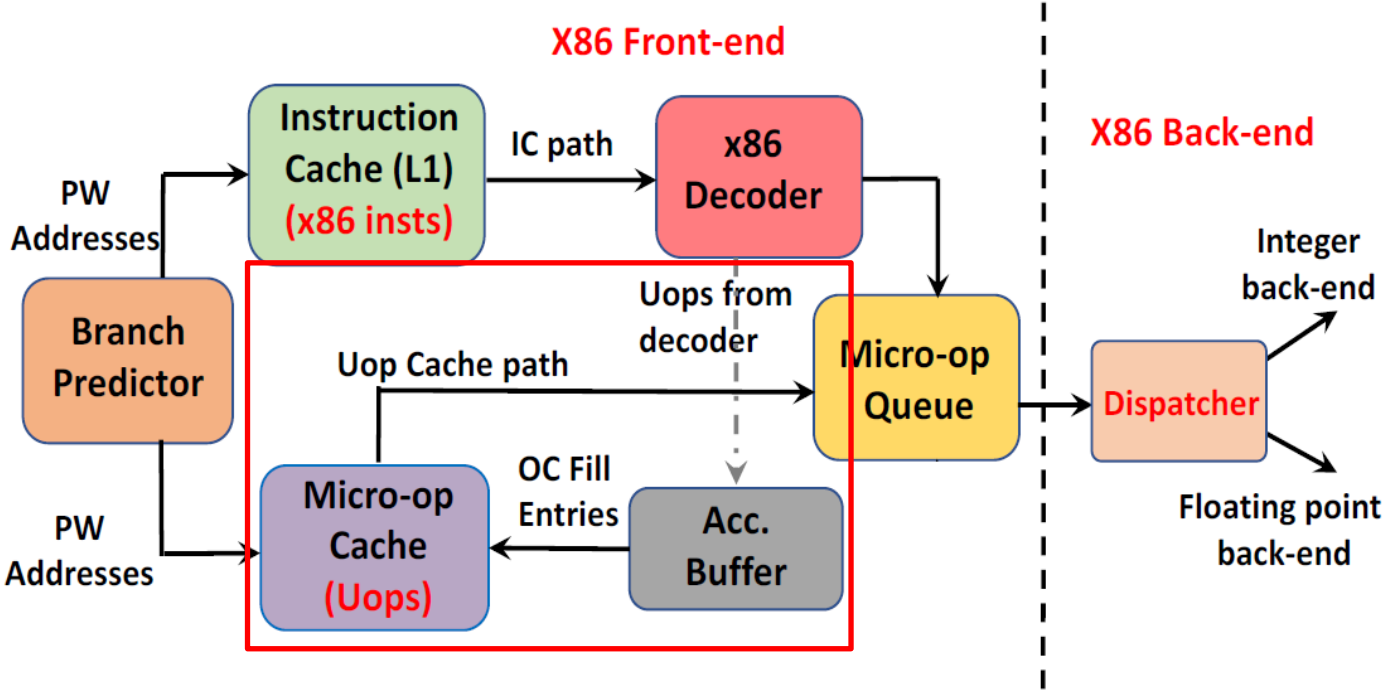
Variable length ISA -> Power hungry decoder

Background: CPU frontend



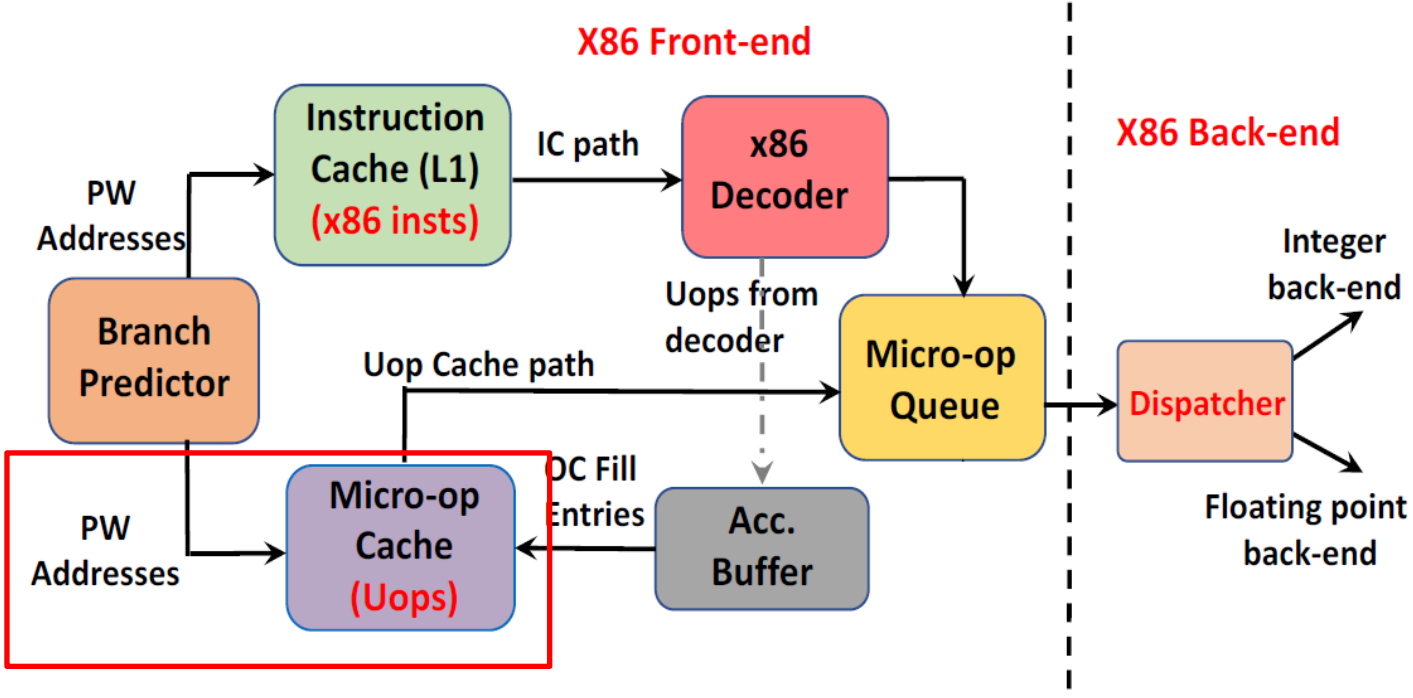
Fixed length micro-op(uop) -> simpler execution logic

Background: CPU frontend



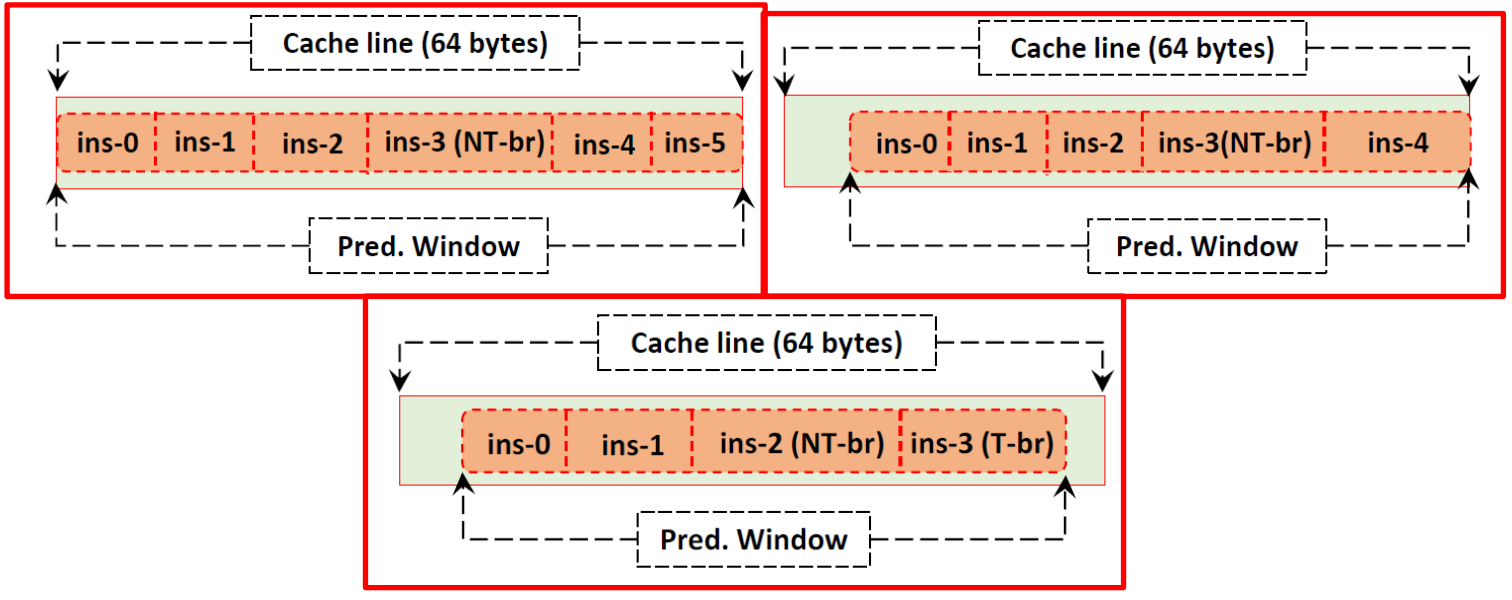
Uops cached into uop cache

Background: CPU frontend



Hit uop cache -> bypass fetch & decode

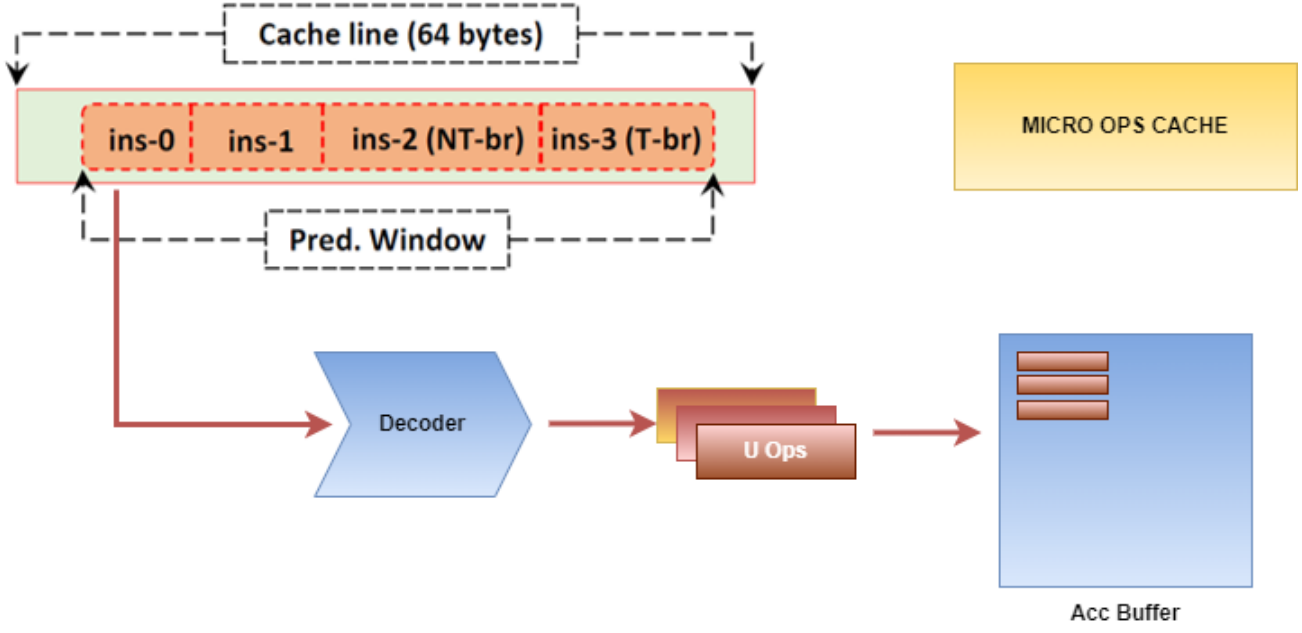
Prediction window(PW)



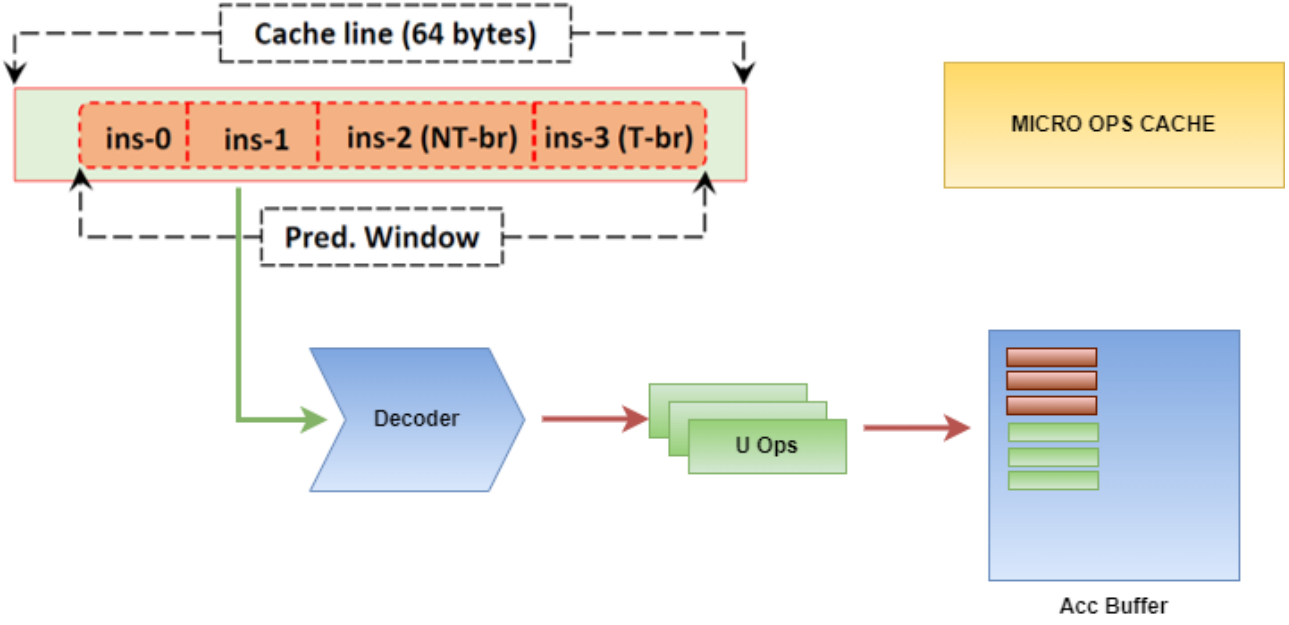
- PW : can **start and end anywhere** in a cache line
- Termination conditions

  (1) **I-cache line boundary** (2) **Predicted taken branch**

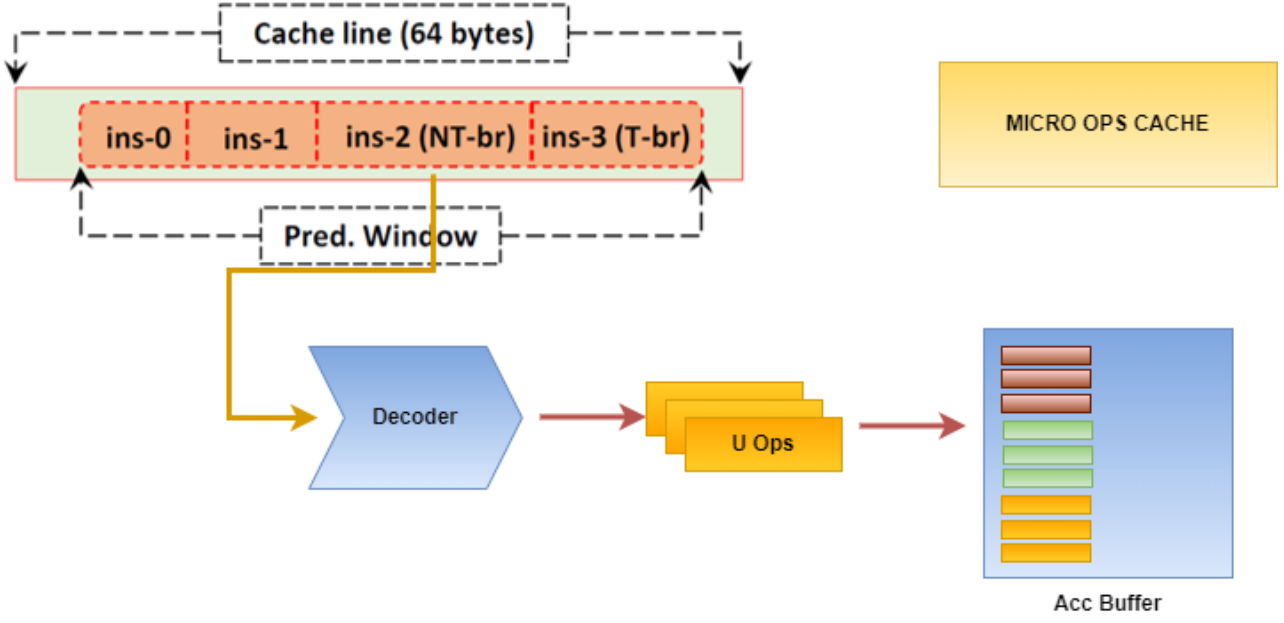
Uop cache entry



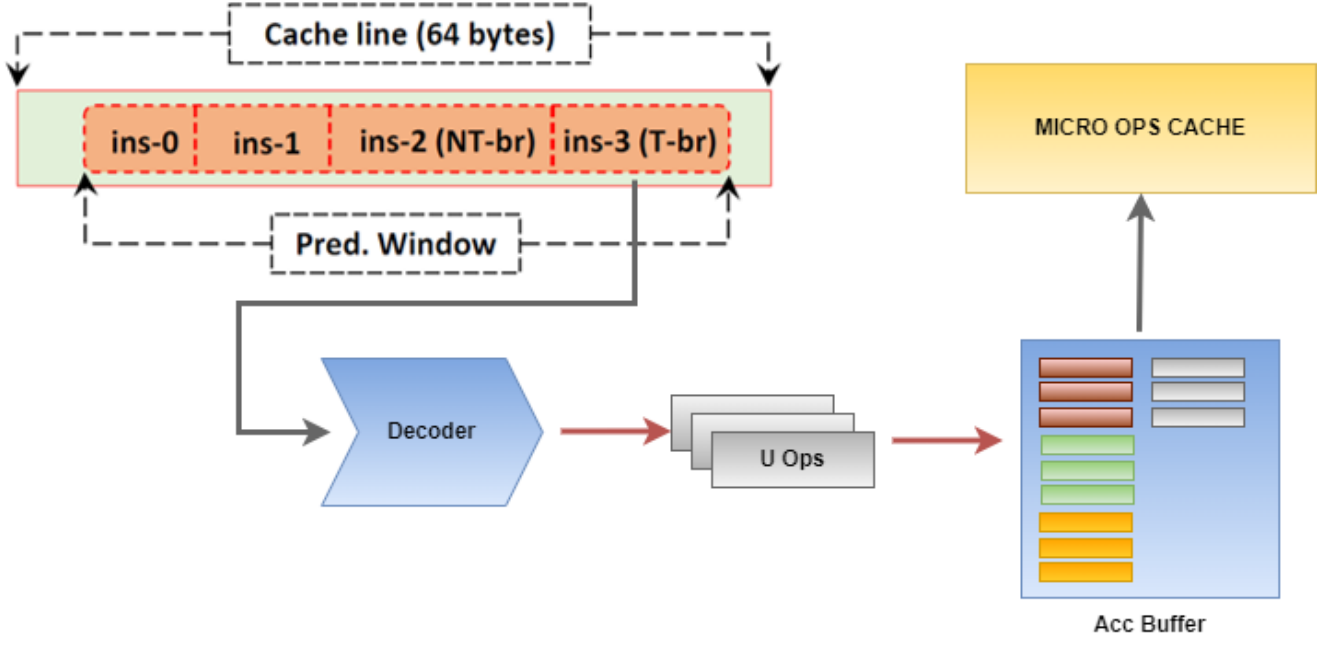
Uop cache entry



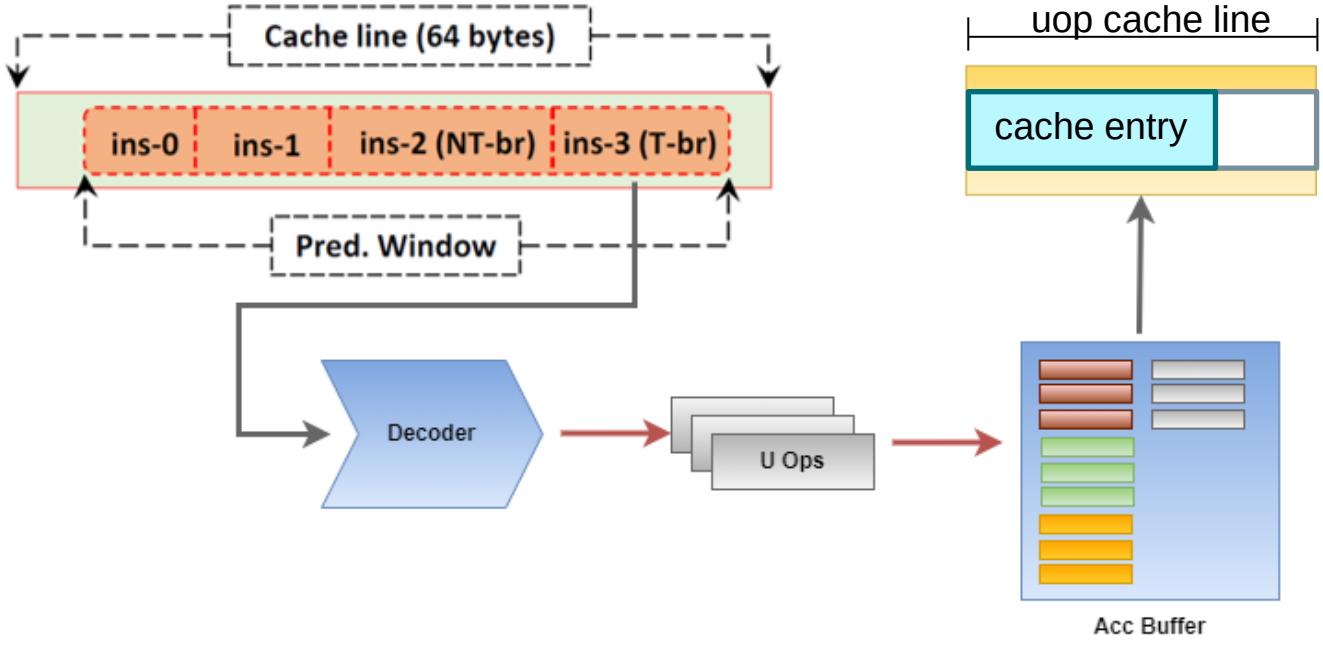
Uop cache entry



Uop cache entry

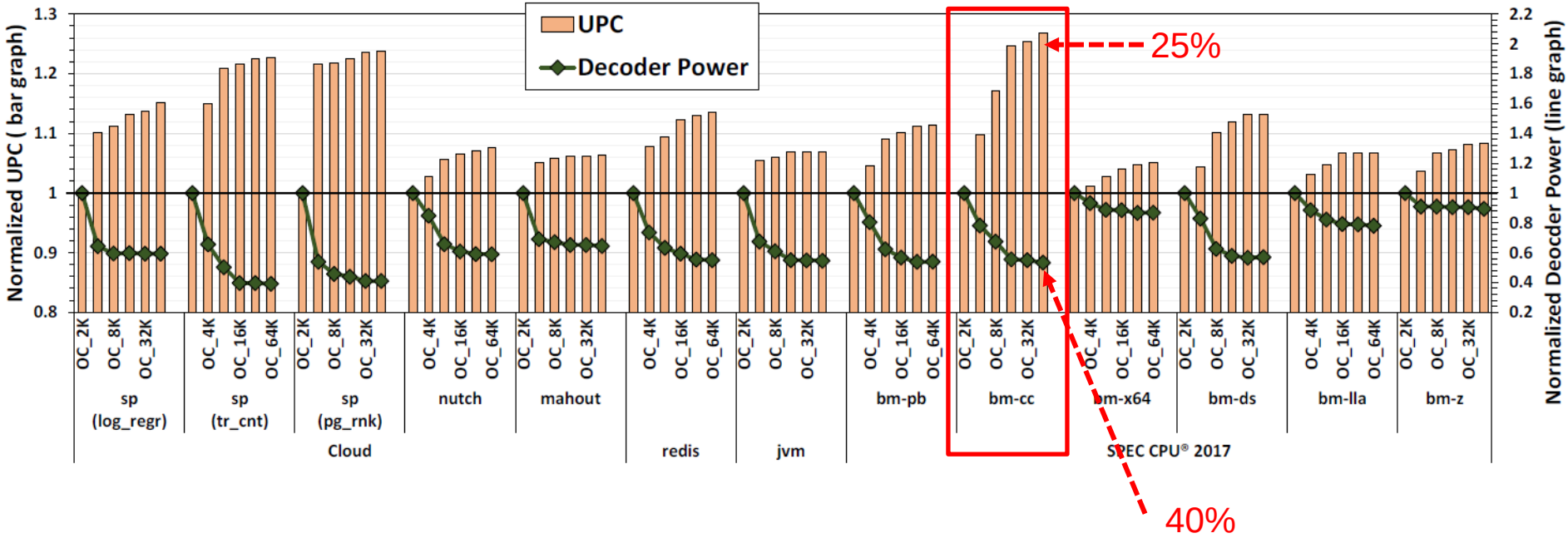


Uop cache entry

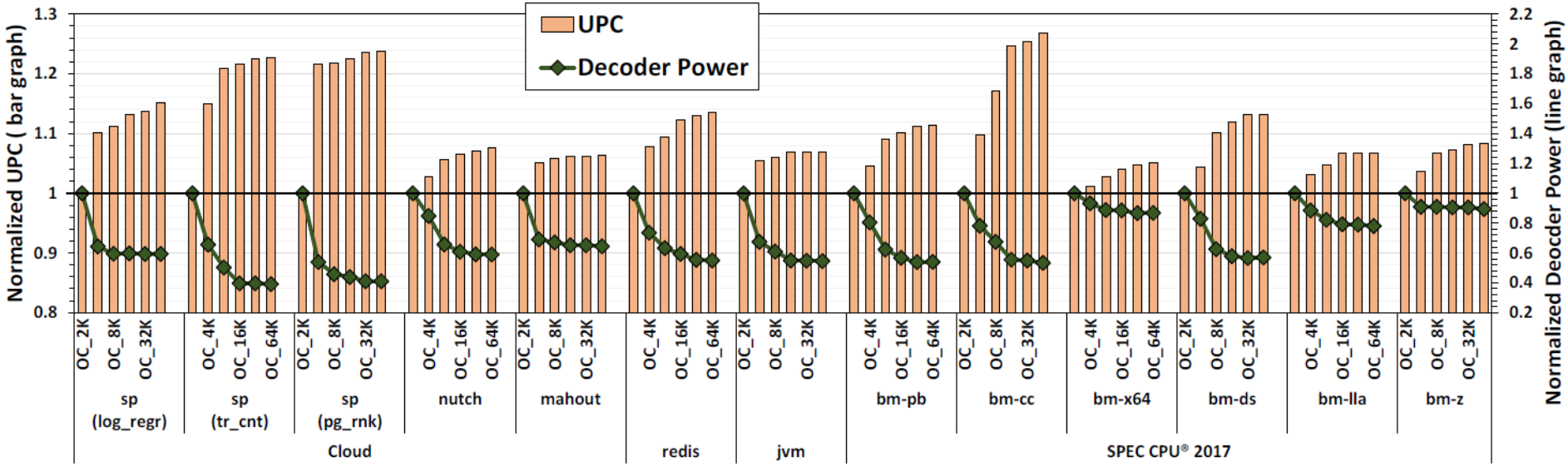


- PW termination condition -> uop cache entry written to uop cache line
- Uop cache entry: **set of uops**
- Takeaway: uop cache entry **may not occupy** entire uop cache line

Impact of Uop cache



Impact of Uop cache

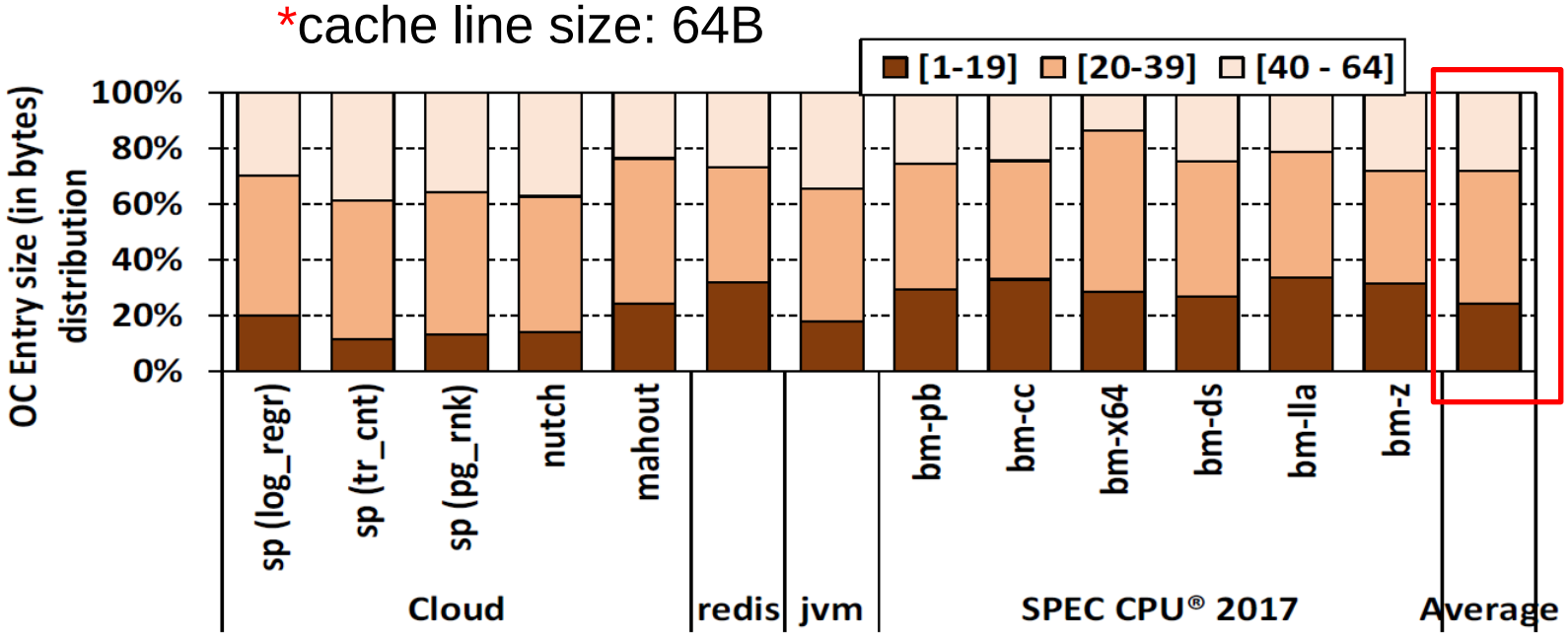


- Average UPC improvement: **11.2 %** (**64K** uop cache)
- Average Reduction Decoder power consumption: **39.2 %**

What next?

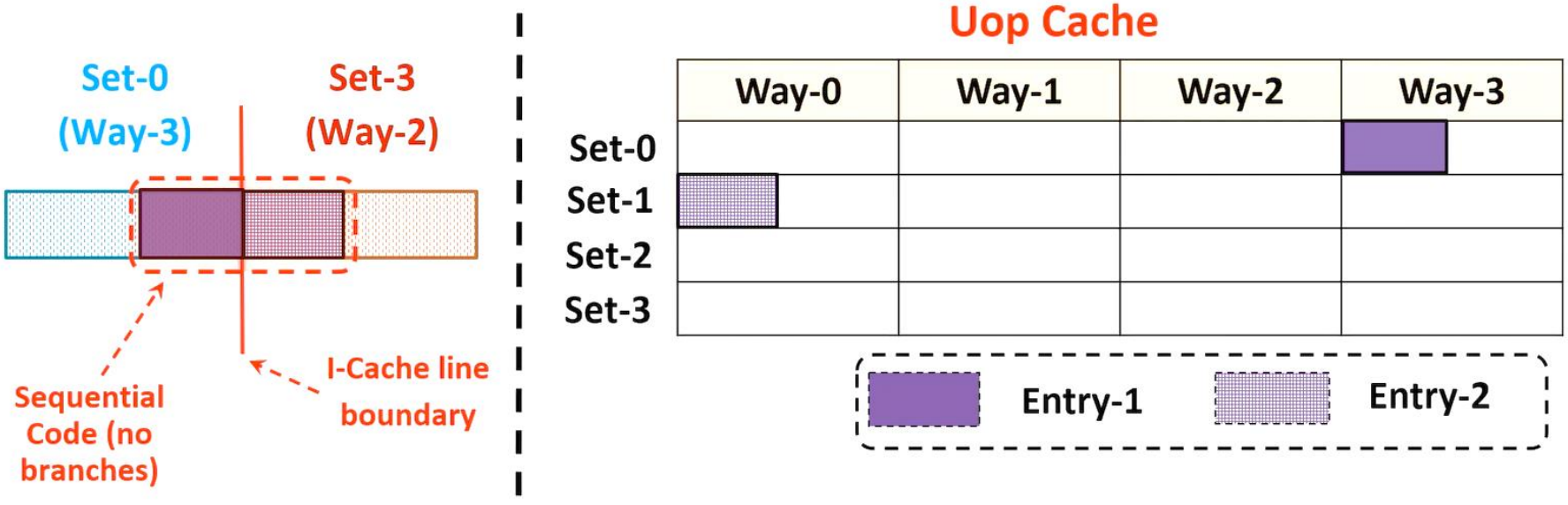
- Throw money: Increase Uop cache size
- Optimize the current Uop cache design

Motivation: Fragmentation



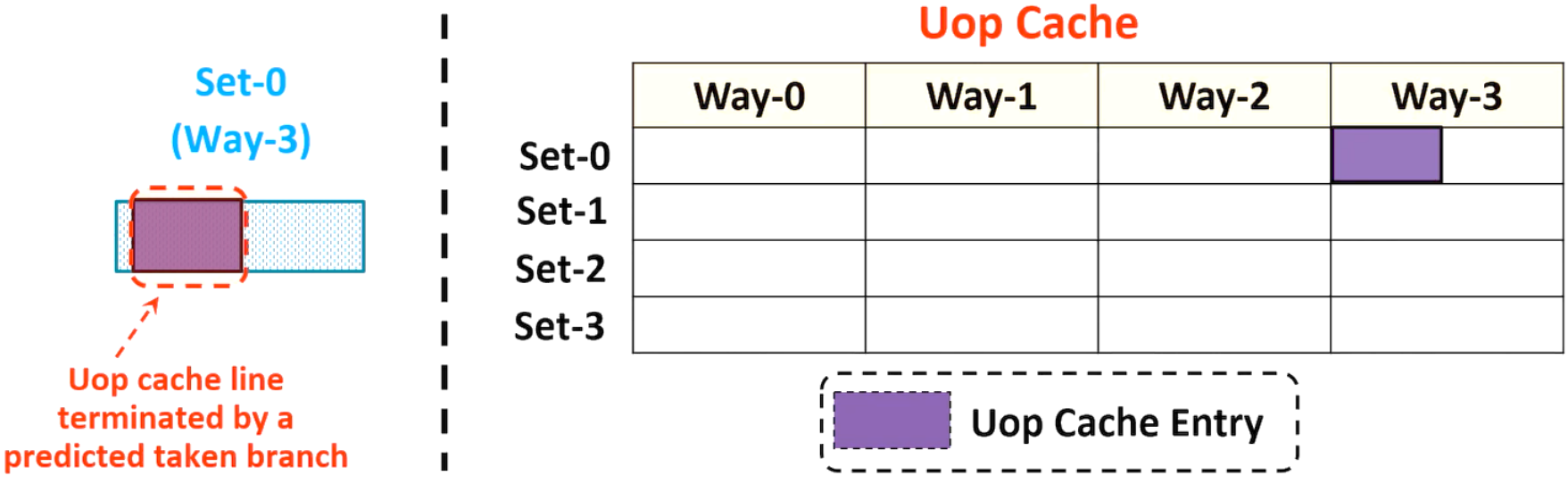
- 72 % of Uop cache lines are **highly fragmented**
- Main reason: **termination conditions**

Fragmentation source: I-cache line boundary



- Termination condition leads to **smaller uop cache entries**
- **Low uop cache utilization**

Fragmentation source: Predicted taken branch



- Termination condition leads to **smaller uop cache entries**
- **Low uop cache utilization**

Fragmentation source: other conditions

Other terminating conditions:

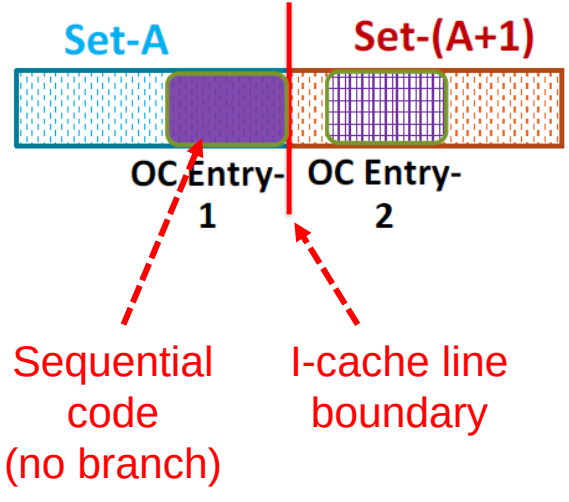
- Max. no. of **immediate/displacement** values per cache line
- Max. no. of **microcoded instructions** per cache line

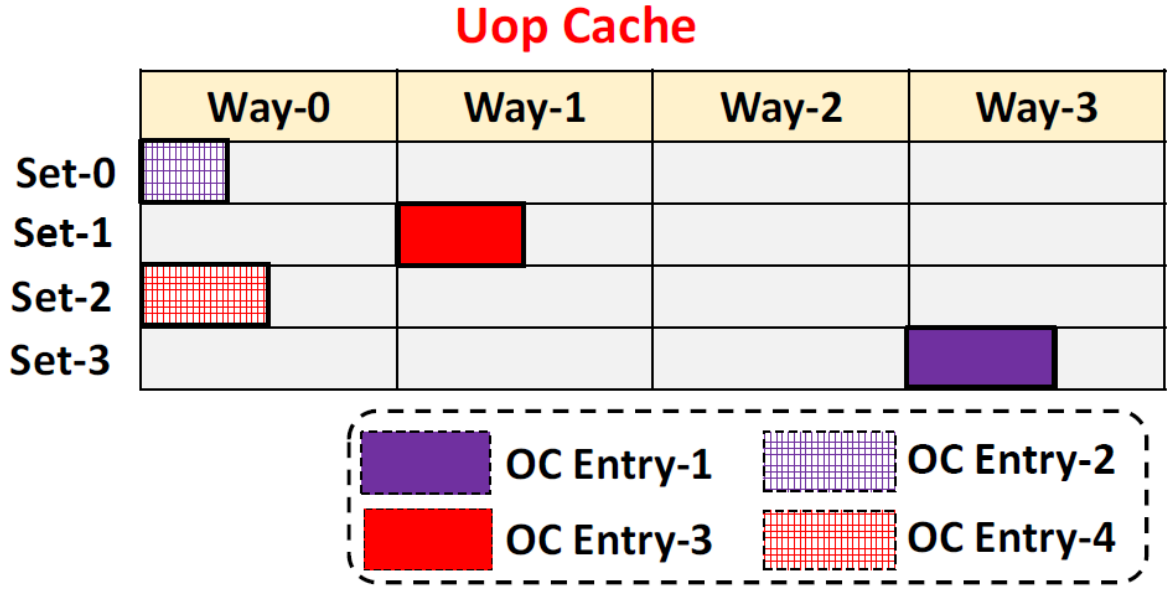
Solutions proposed

Two solutions to reduce fragmentation:

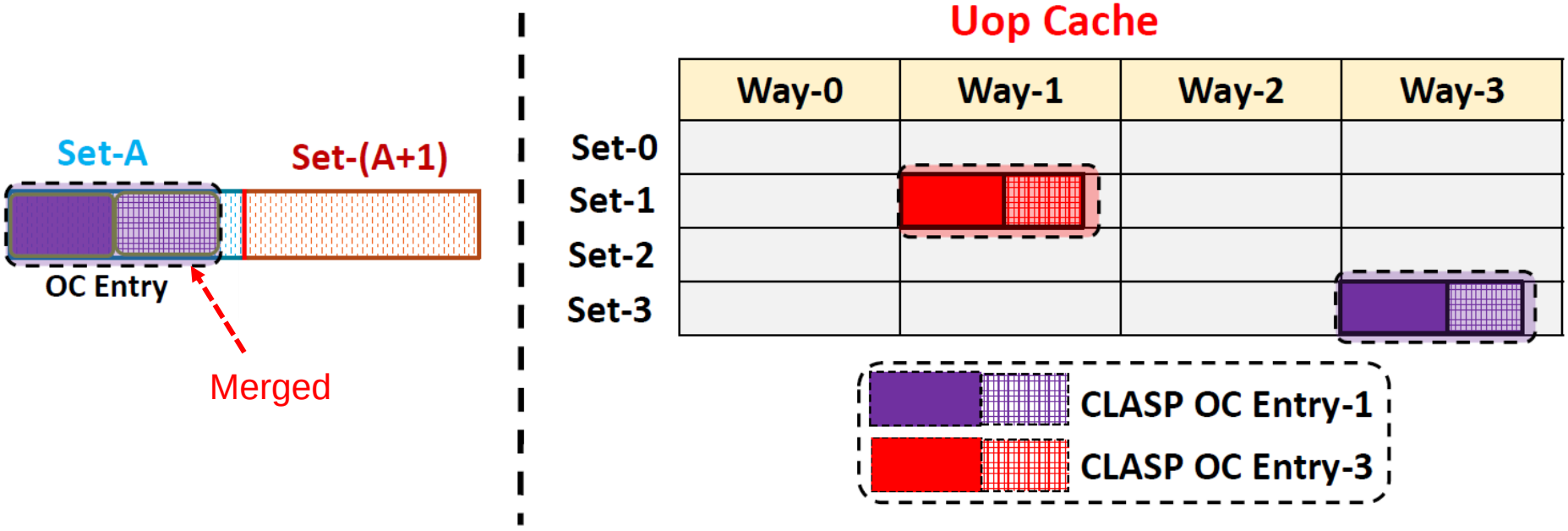
- CLASP
- Compaction
  - RAC
  - PWAC
  - Forced-PWAC

Cache line boundary agnostic Uop (CLASP)



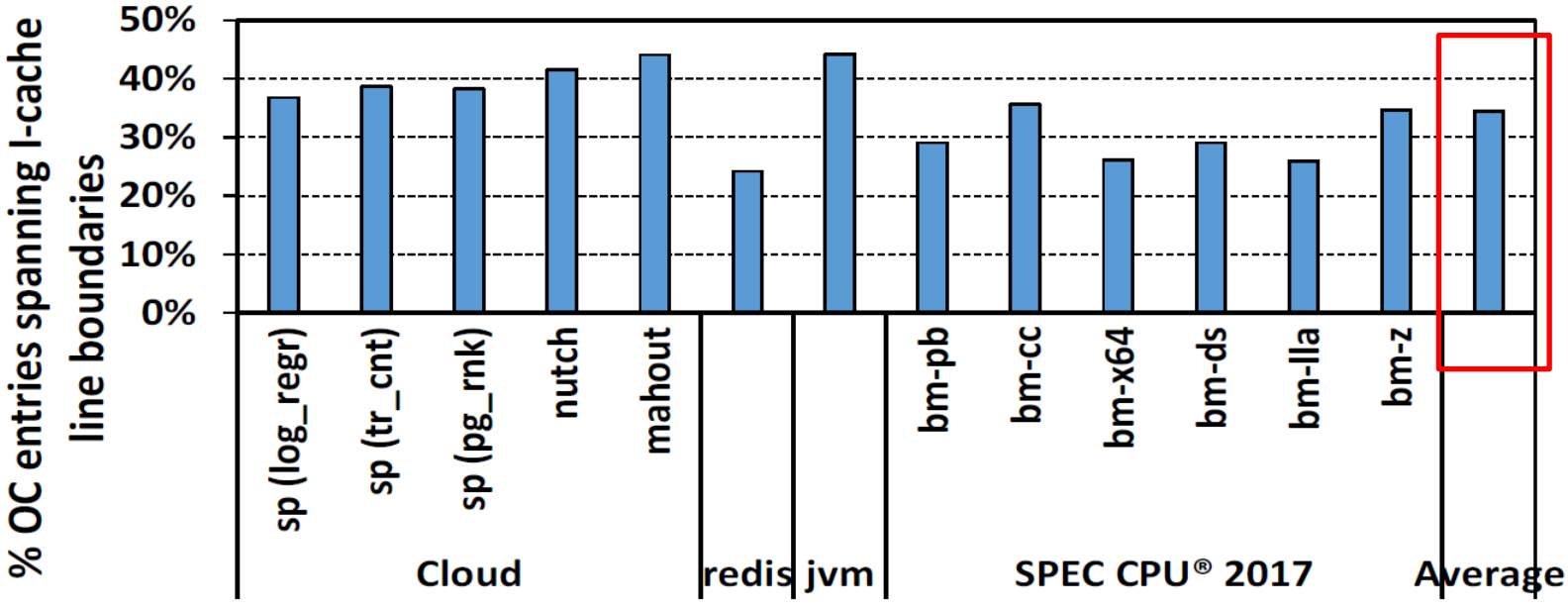


Cache line boundary agnostic Uop (CLASP)



- Relaxes **I-cache line boundary** termination condition
- Merges uop cache entries from **sequential code**
- **Doubles dispatch bandwidth**

Cache line boundary agnostic Uop (CLASP)

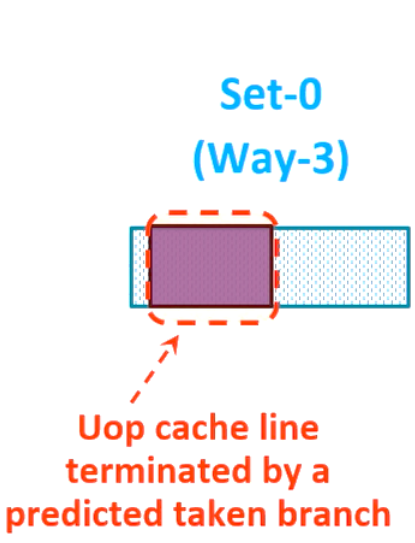


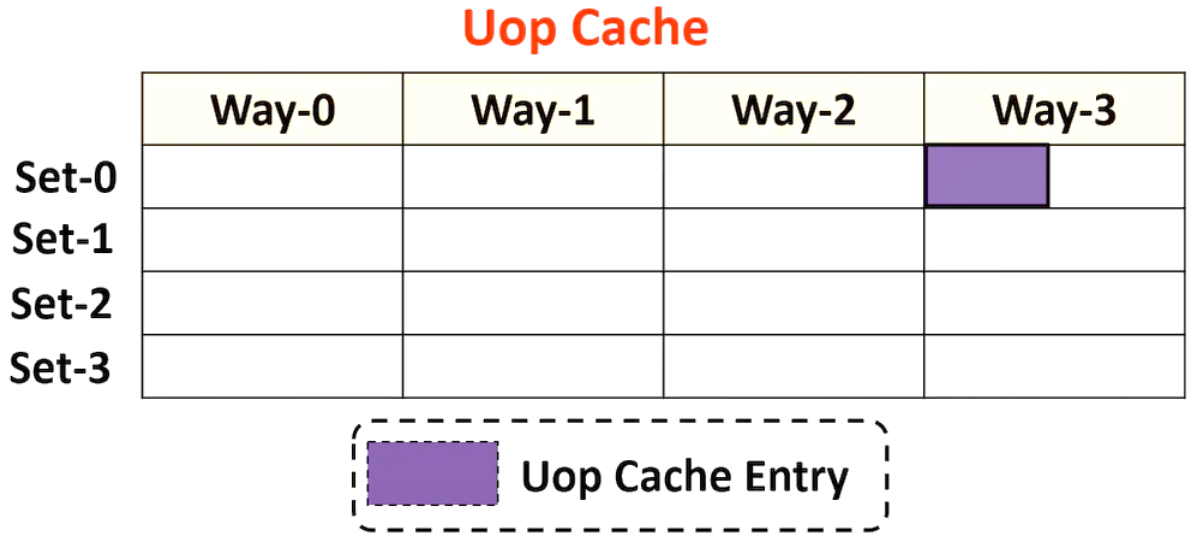
On average **35%** of Uop cache entries are merged by **CLASP**

Pause

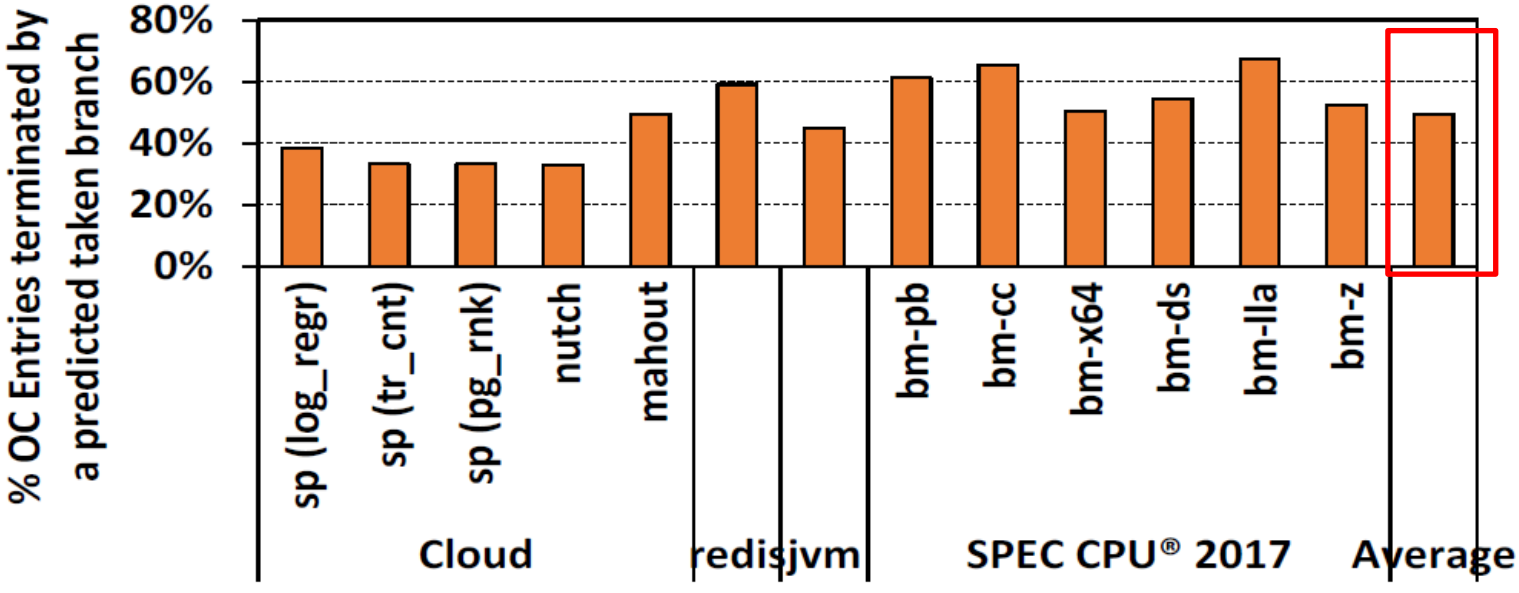
Any questions?

Fragmentation source: Predicted taken branch



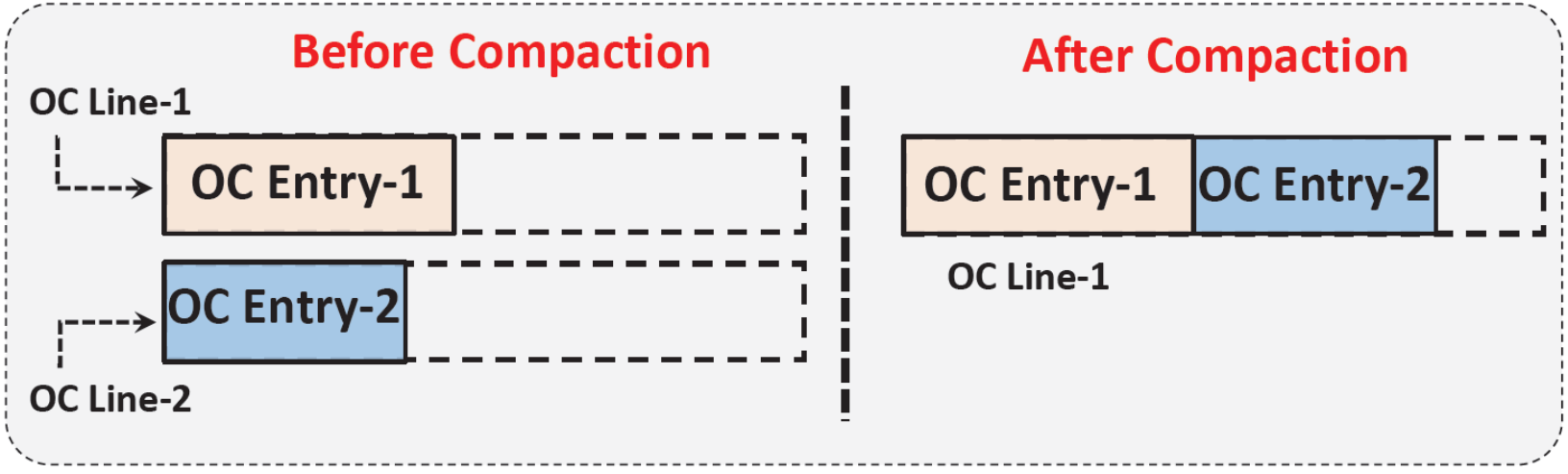


Fragmentation source: Predicted taken branch



On average **49%** of Uop cache entries terminated by **predicted taken branches**

Compaction

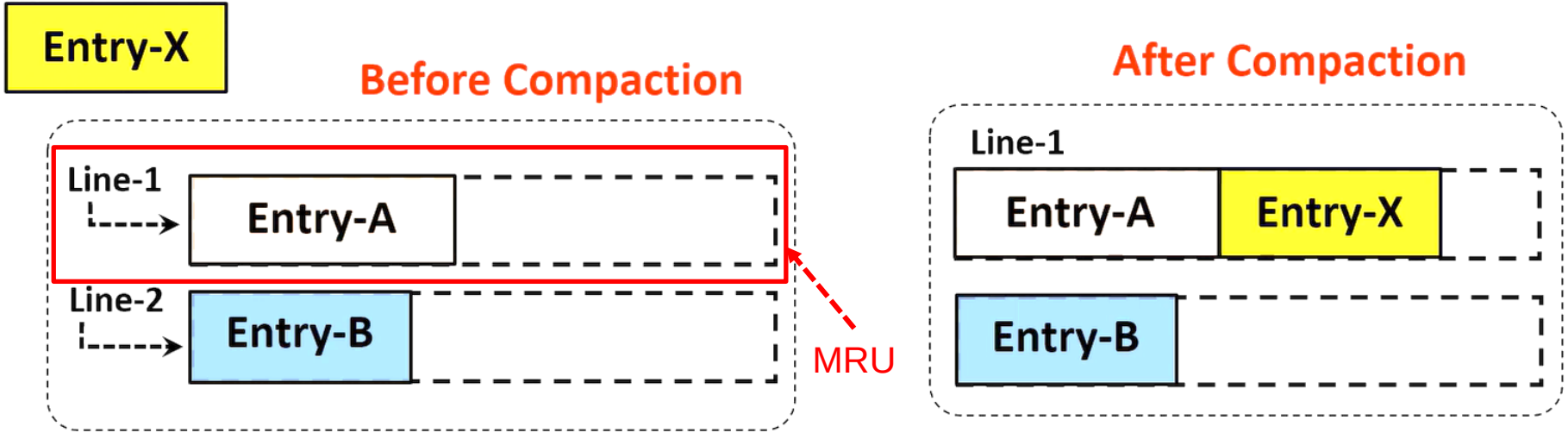


- Target termination conditions **other** than **I-cache line boundary**
- Compacts Uop cache entries in a **single cache line**
- Entries are **not** merged together to a **single entry unlike CLASP**
- **Dispatch bandwidth** remains same unlike **CLASP**

Compaction: Challenge

- Which Uop cache entries to compact together?

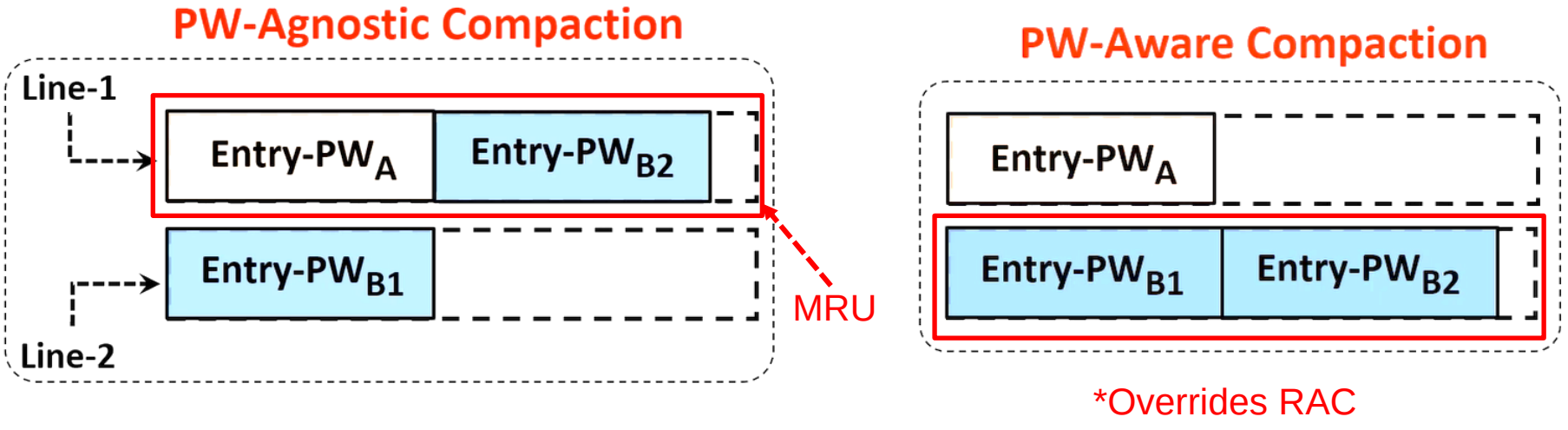
Replacement aware compaction(RAC)



Note: compaction at the time of **Uop cache fill**

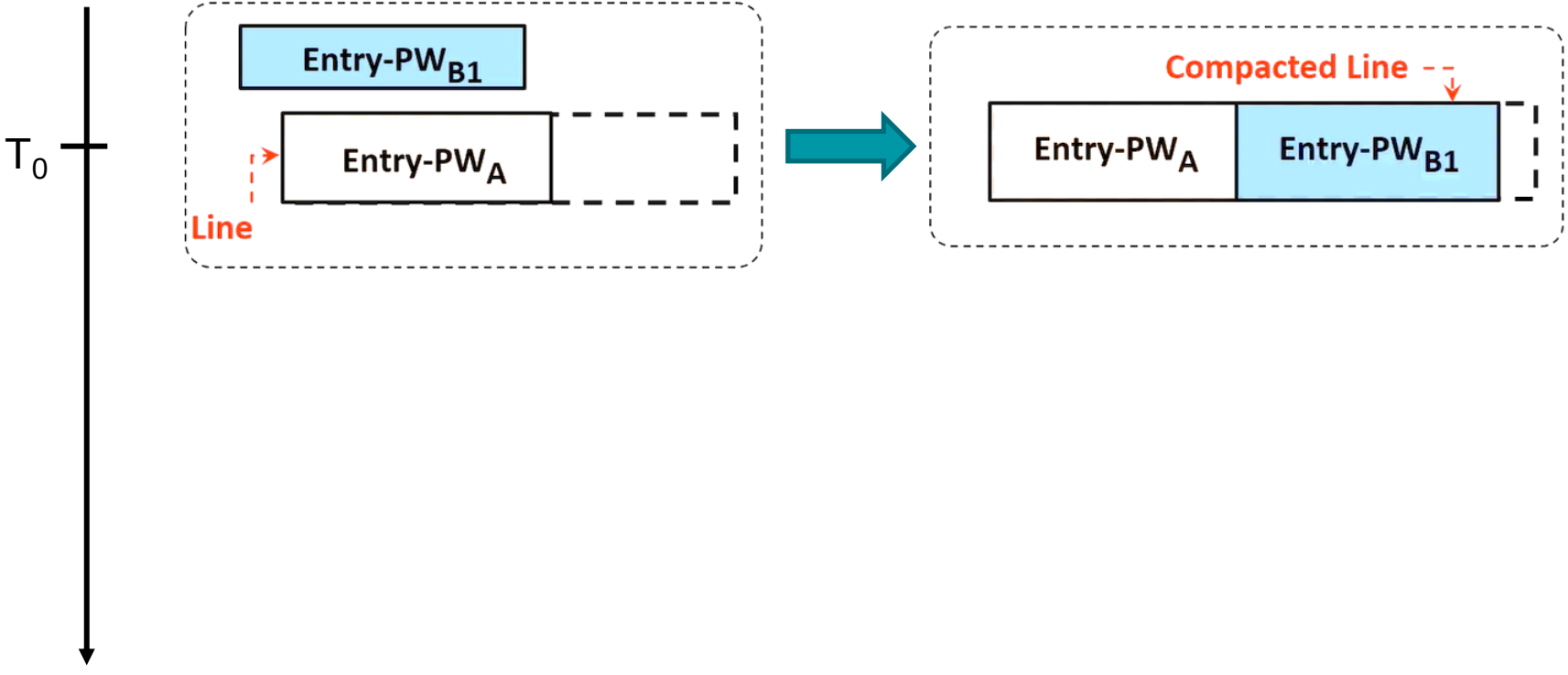
- Compacts new **Uop cache entry** with the **MRU line** in cache set
- Ensures Uop cache entries are **temporally correlated**

PW aware compaction (PWAC)

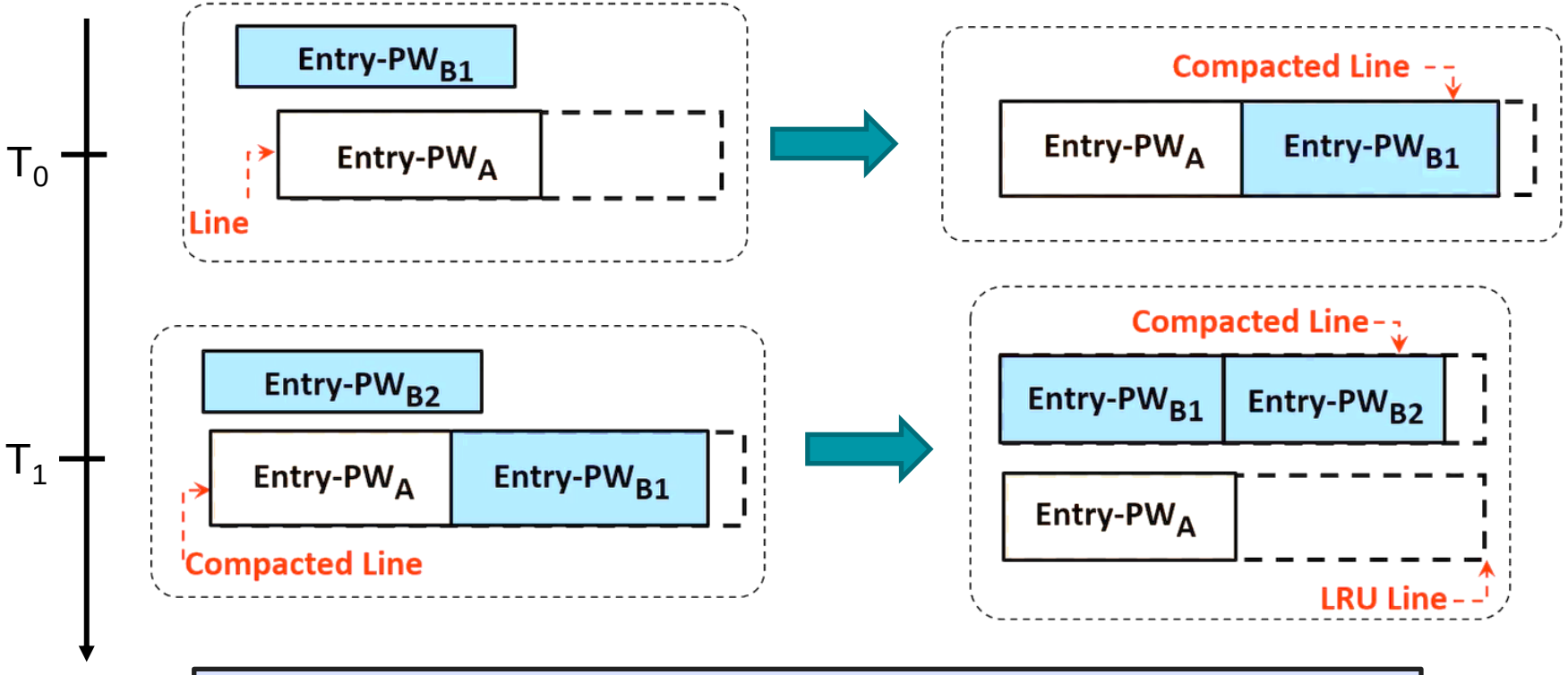


Attempts to compact new Uop cache entry with entries from **same Prediction window**

Forced-PWAC

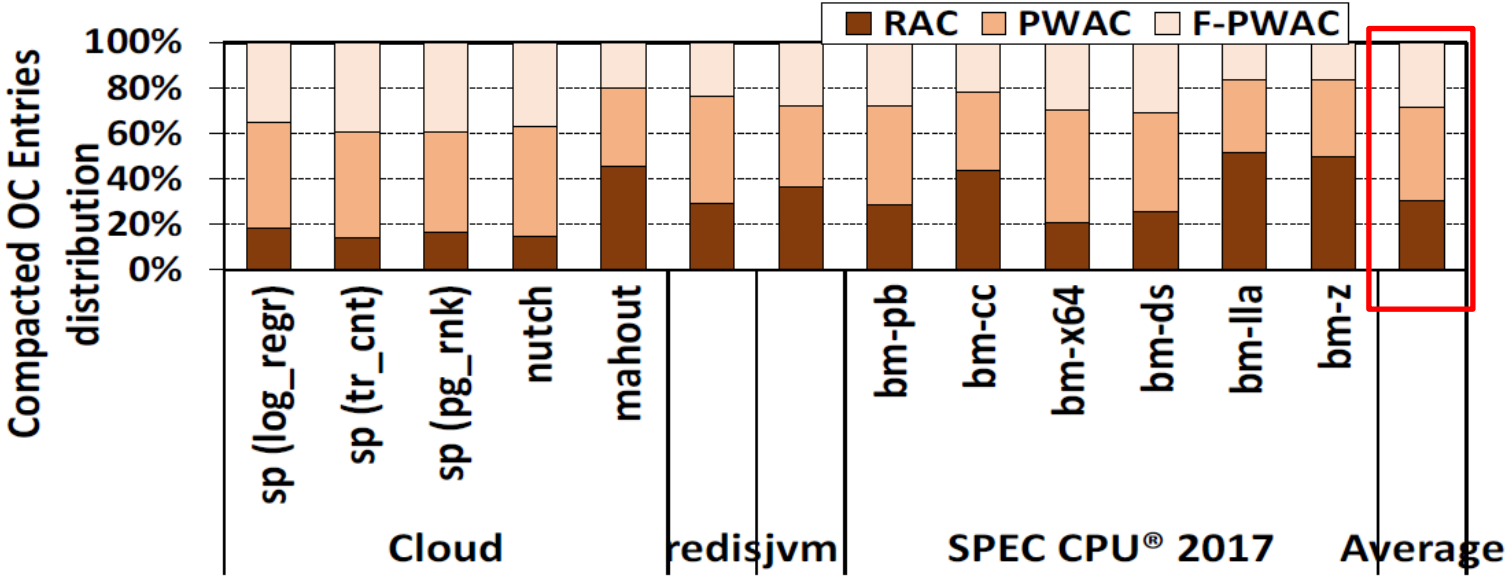

T0
Entry-PWB1
Entry-PWA
Line
Compacted Line
Entry-PWA
Entry-PWB1

Forced-PWAC



Forces compaction of entries from same PW

Compaction technique distribution



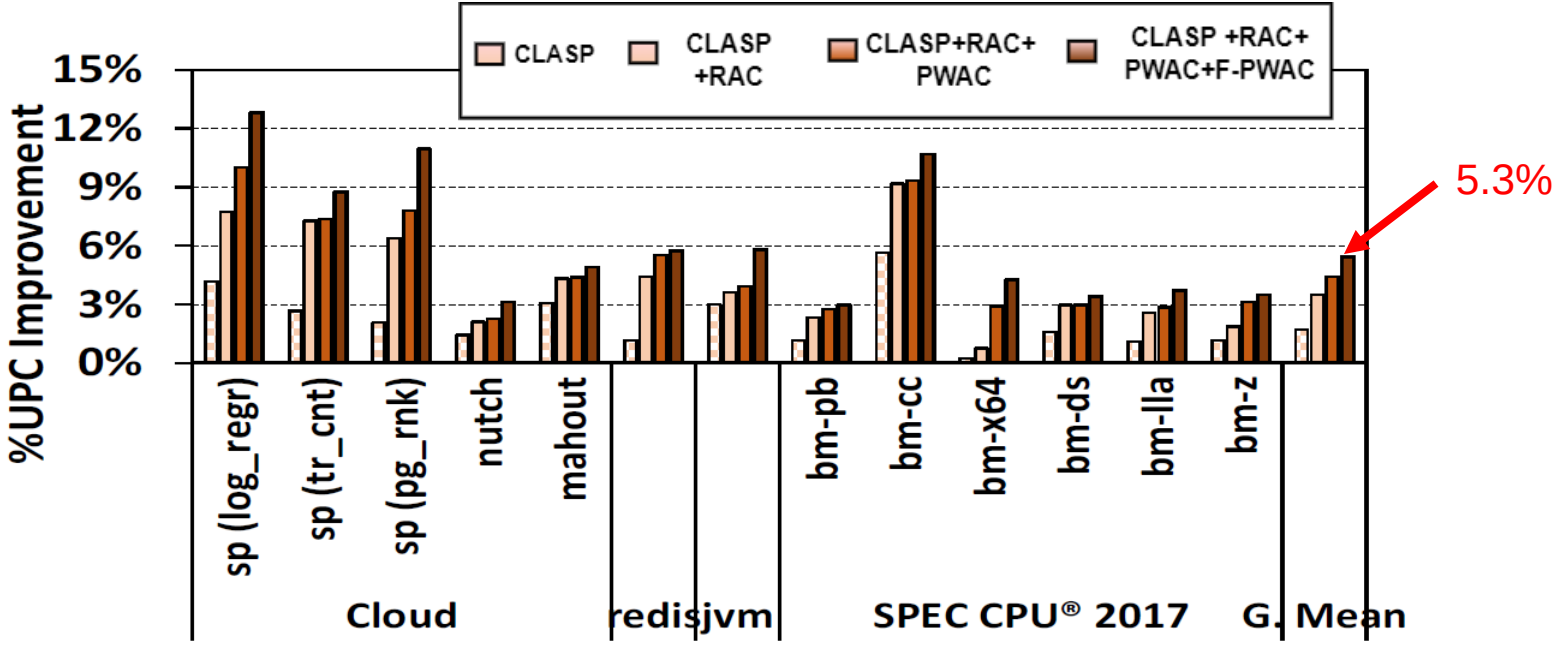
- Note: **all 3** compaction techniques work simultaneously
- Usage priority: **F-PWAC > PWAC > RAC**
- **Almost equal** distribution

Evaluation setup

Core	3 GHz, x86 CISC-based ISA Dispatch Width: 6 instructions Retire Width: 8 instructions Issue Queue: 160 entries Decoder Latency: 3 cycles Decoder Bandwidth: 4 insts/cycle
Uop Cache	32-sets, 8-way associative True LRU replacement policy Bandwidth: 8 uops/cycle Uop Size: 56-bits; ROB: 256 Uop Queue Size: 120 uops Max uops per uop cache entry: 8 Imm/disp operand size: 32 bits Max imm/disp per OC entry: 4 Max U-coded insts per OC entry: 4
Branch Predictor	Tage Branch Predictor 2 branches per BTB entry; 2-level BTBs
L1-I	32KB, 8-way associative, 64Bytes cache line; True LRU Replacement branch prediction directed pretcher Bandwidth: 32 Bytes/cycle
L1-D	32KB, 4-way associative, 64Bytes; True LRU replacement policy
L2 Cache	512KB (private), 8-way associative, 64Bytes cache line, unified I/D cache
L3 Cache	2MB (shared), 16-way associative, 64Bytes cache line, RRIP repl. policy
Off-chip DRAM	2400 MHz

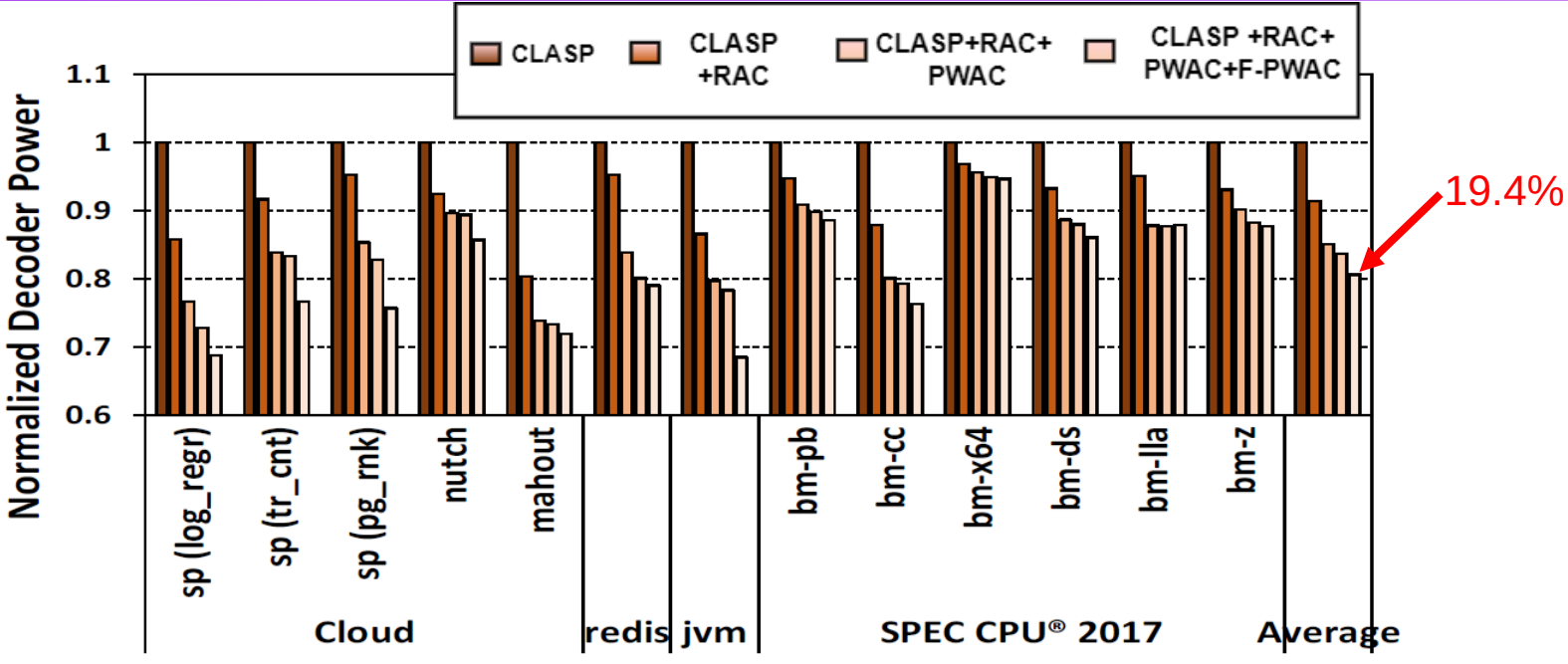
Baseline Uop cache fits 2K uops

Performance improvement



Performance improvement with CLASP + compaction: **5.3 %**

Power reduction



Decoder power reduction with CLASP + compaction: **19.4 %**

Conclusion

- Uop cache is **highly fragmented** due to terminating conditions.
- **CLASP** reduces fragmentation by relaxing **I-cache line boundary** termination condition
- **Compaction** reduces fragmentation by **joining** possibly unrelated uop cache entries.