BeKnight: Guarding against Information Leakage in Speculatively Updated Branch Predictors

Md Hafizul Islam Chowdhuryy
University of Central Florida

Zhenkai Zhang
Clemson University

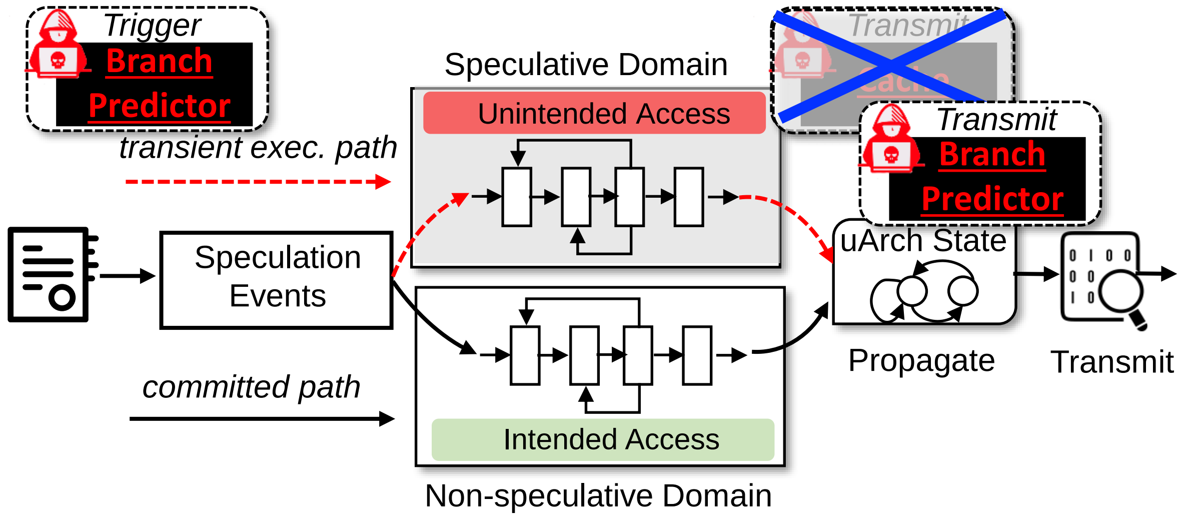
Fan Yao
University of Central Florida

Presenter: Md Hafizul Islam Chowdhuryy
Computer Architecture and Systems Research Lab, University of Central Florida

42nd IEEE/ACM International Conference on Computer-Aided Design (**ICCAD** 2023)
October 29th – November 2nd, 2023

Background

- Speculative execution attacks are extremely dangerous.
  - Break program semantics and leak arbitrary memory locations.
  - Can bypass traditional security measures and detections.



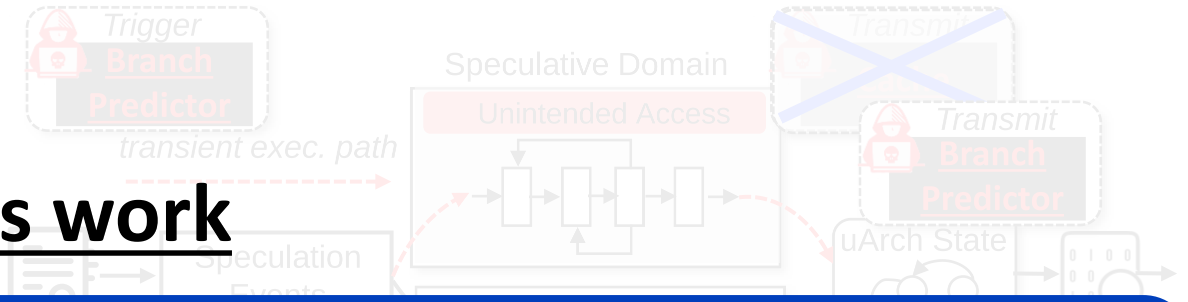
- Branch predictors (BPU) are traditionally used to ***trigger*** speculation.
  - Secret transmission generally through other side channels (i.e., **cache**).
- **BranchSpectre**[TC'21] demonstrate new variant of Spectre attack.
  - Uses BPU for *both* **trigger** and **transmission** in speculative execution attacks.

Only BPU is exploited → Can manifest when all other uArch components are not exploitable.

Utilizes simpler code patterns → Higher exploitability compared to Spectre in real systems.

Background

- Speculative execution attacks are extremely dangerous.
  - Break program semantics and leak arbitrary memory locations.



This work

BeKnight **- Efficient architectural mechanism for *secure* branch predictor in speculation.**

(*retain the performance advantage of speculative branch predictor updates)

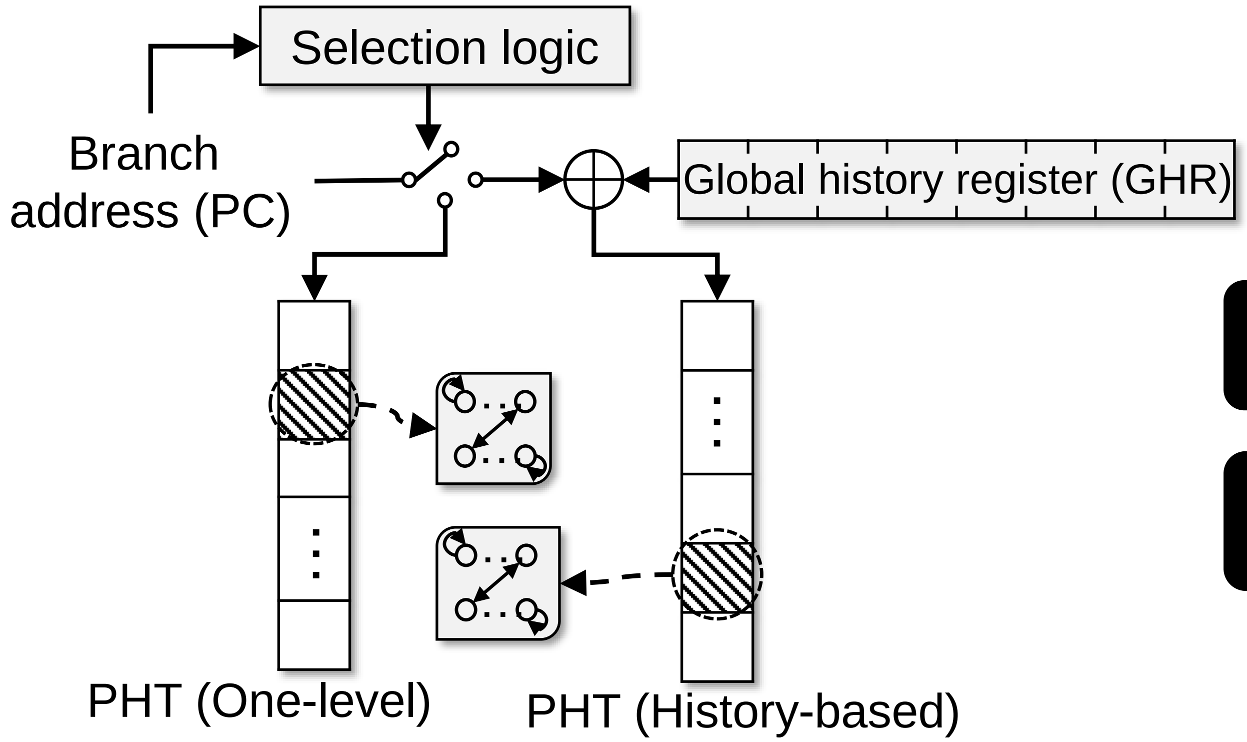
  - Secret transmission generally through other side channels (i.e., **cache**).
- **BranchSpectre**[TC'21] demonstrate new variant of Spectre attack.
  - Uses BPU for *both* **trigger** and **transmission** in speculative execution attacks.

Only BPU is exploited → Can manifest when all other uArch components are not exploitable.

Utilizes simpler code patterns → Higher exploitability compared to Spectre in real systems.

Background: Modern Branch Predictor Unit

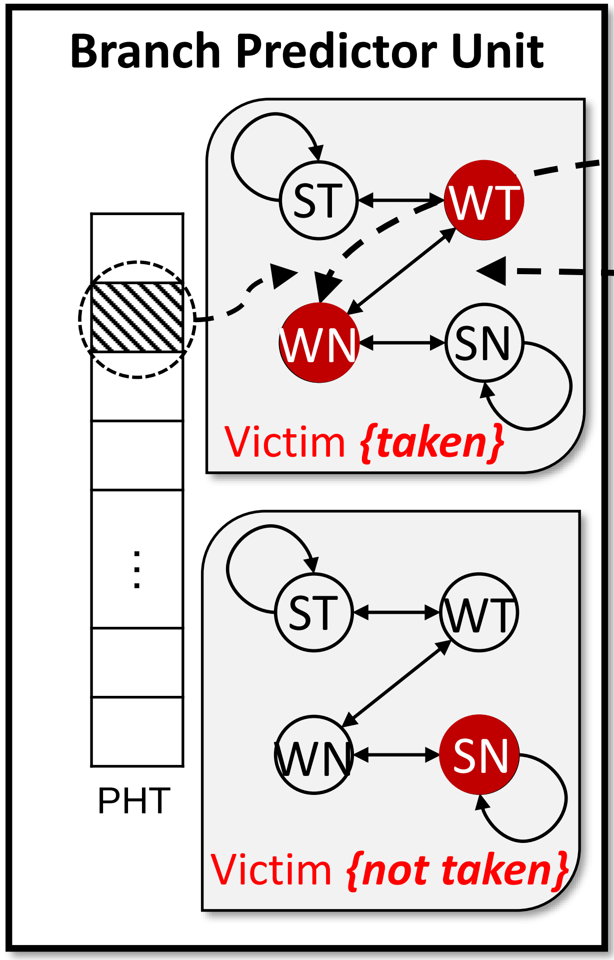
- Modern processors utilize *Hybrid branch predictor* for branch direction prediction.
- Utilize either **branch address**, or both **branch address and history** to index PHT.



Pattern History Table (PHT) is a table of small counters (typically 2-bit/3-bit) each.

Total number of entries in PHT is very large (i.e., 16K entries in recent Intel Processors).

Background: BranchSpectre Attack



1. *Initialize* the PHT entry to known state

2. Victim ***speculation***

3. *Infer* new PHT state by measuring execution time

```
secret[] = {0,...,
data[size];
① if (idx < size)
```

Start of ***Speculation***

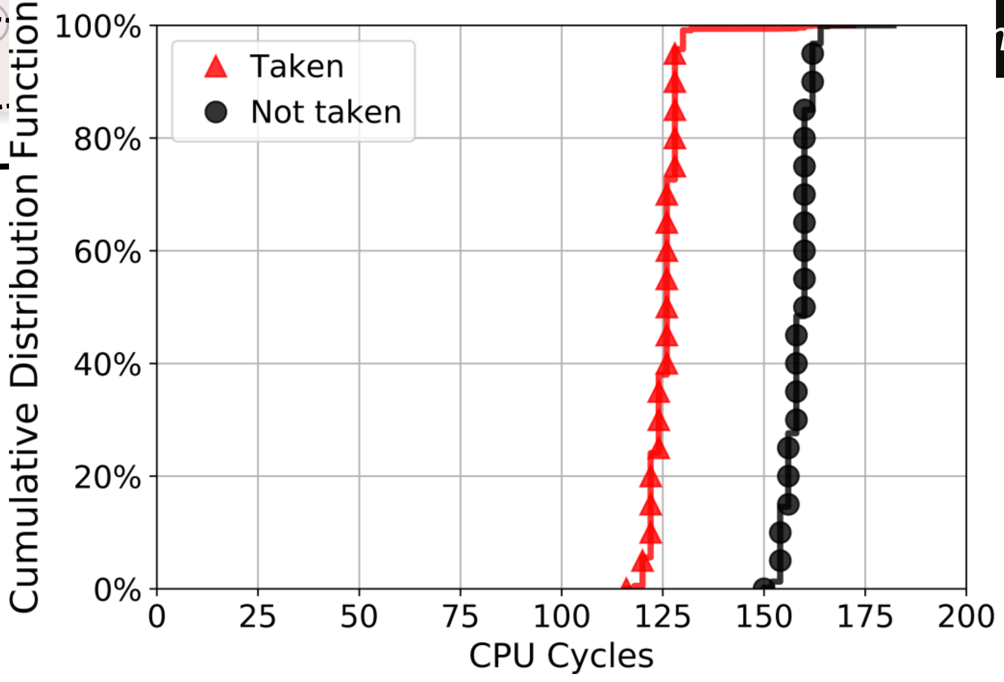


Figure: Execution time of attacker branch {*actual direction* → *taken*} corresponding to different victim branch execution direction.

Threat Model

- *Active attacker* who can manipulate and infer predictor state.
- Attacker can achieve core co-location with victim.
  - Either in round-robin settings or simultaneously in SMT settings.
- Operating System is trusted.

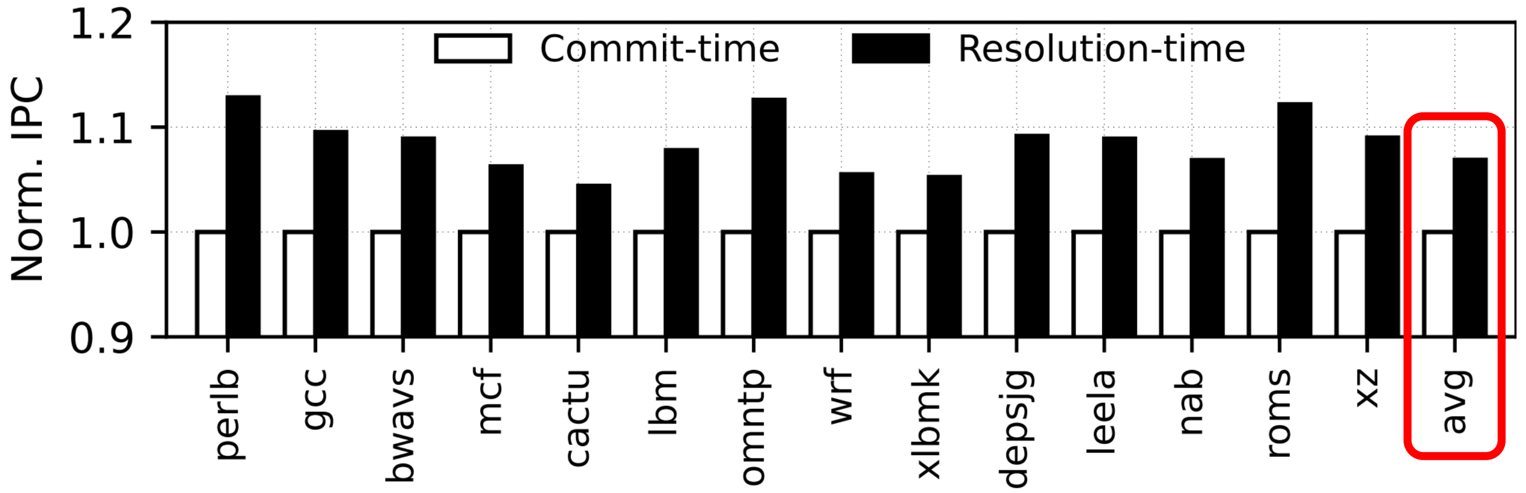
- Same-thread attacks are out of scope.
- Side channel leakage from other hardware components are out of scope.

Design Challenges

Naïve designs:

1. Prevent speculative update of PHT.

Speculative update of PHT offers non-trivial performance improvements!



8.5% Speedup on average if PHT is updated speculatively

Figure: Performance comparison between commit-time update (non-speculative) and resolution-time update (speculative) of PHT .

Design Challenges

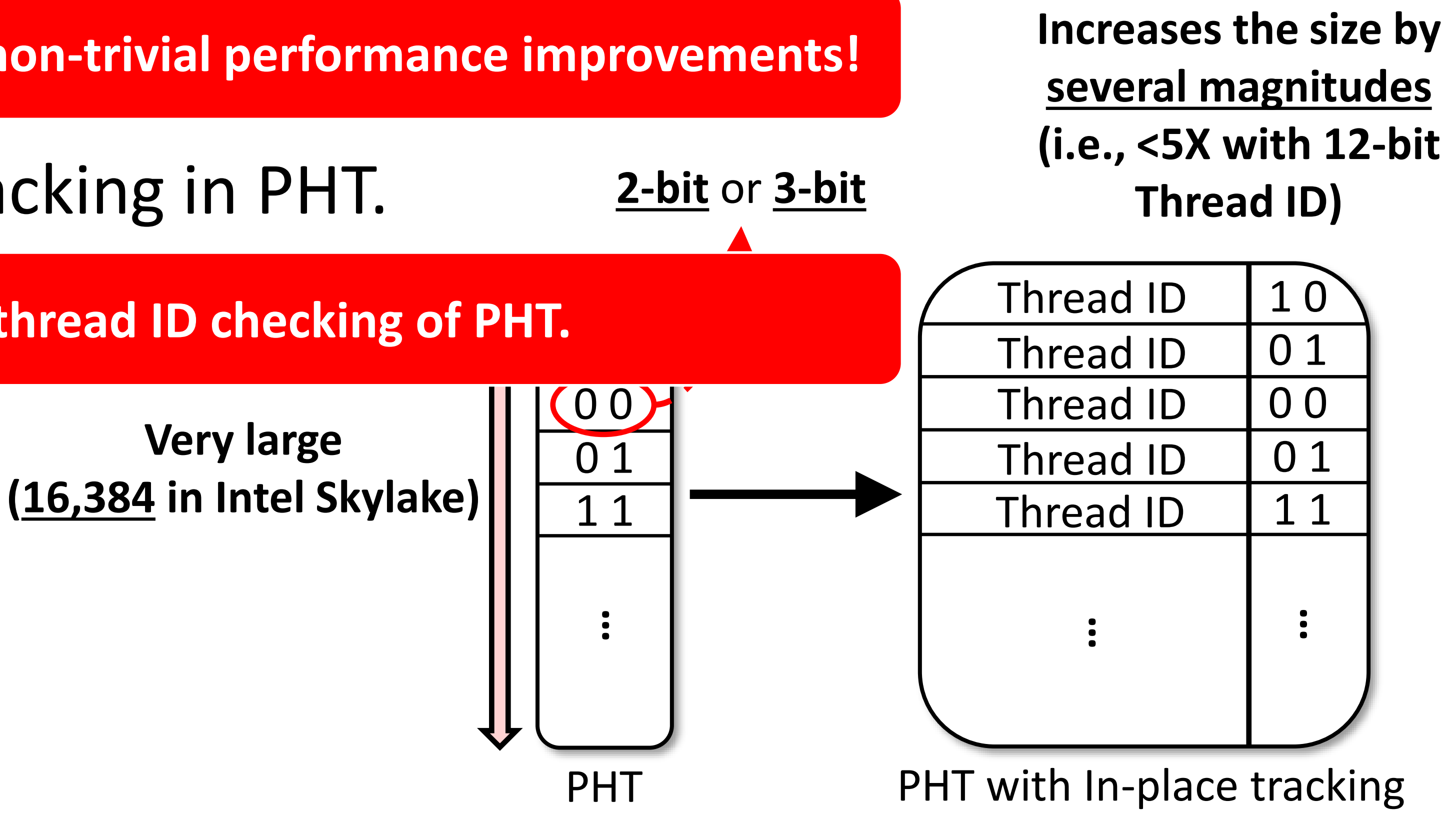
Naïve designs:

1. Prevent speculative update of PHT.

Speculative update of PHT offers non-trivial performance improvements!

2. In-place ownership tracking in PHT.

Prohibitive overheads of in-place thread ID checking of PHT.


Increases the size by several magnitudes (i.e., <5X with 12-bit Thread ID)
2-bit or 3-bit
Very large (16,384 in Intel Skylake)
0 0
0 1
1 1
PHT
Thread ID 1 0
Thread ID 0 1
Thread ID 0 0
Thread ID 0 1
Thread ID 1 1
PHT with In-place tracking

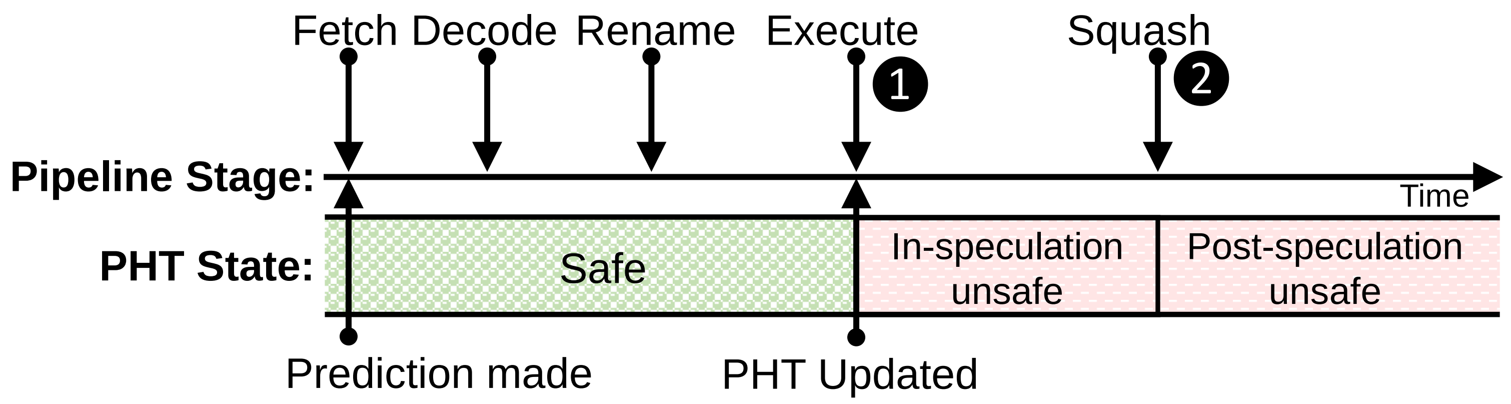
BeKnight Design Principle

- **Security:** Unsafe PHT entries that belong to one thread context *cannot be used to make predictions* on a different context using the non-architectural states.
  1. Changes to PHT by transient instructions.
  2. Changes to PHT by instructions not committed yet.
- **Performance:** Same domain prediction using non-architectural states must be *allowed* to retain the performance gain.
- **Cost:** Must *not require* to track ownership of all PHT entries.

Characterization of PHT Safe/Unsafe State

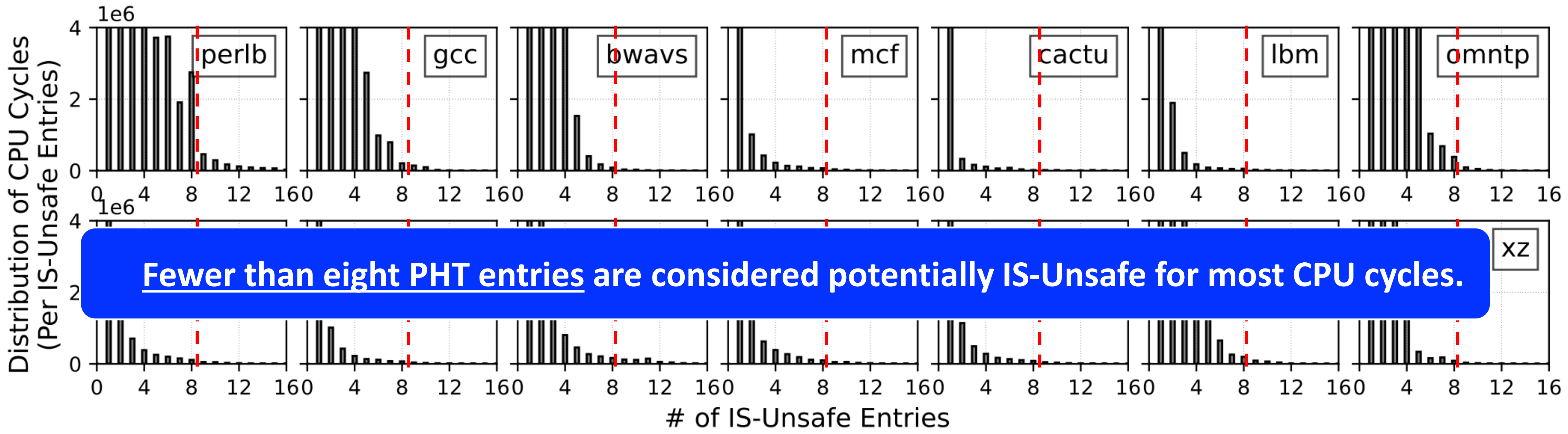
PHT entries are categorized into two classes:

- ***Safe entries:*** Entries updated only by committed instructions.
- ***Unsafe entries:*** Entries updated by *at least* one squashed instructions.
  1. ***In-Speculation Unsafe (IS-Unsafe):*** Updated by in-flight resolved instructions (not yet squashed).
  2. ***Post-Speculation Unsafe (PS-Unsafe):*** Updated by instructions that are already squashed.



Runtime Distribution of *IS-Unsafe* PHT Entries

- At resolution time, the processor can not determine if a branch instruction will be committed.
- Strictly identifying IS-Unsafe entries are not possible.
- We consider all in-flight resolved branch instructions as ***potentially IS-Unsafe***.

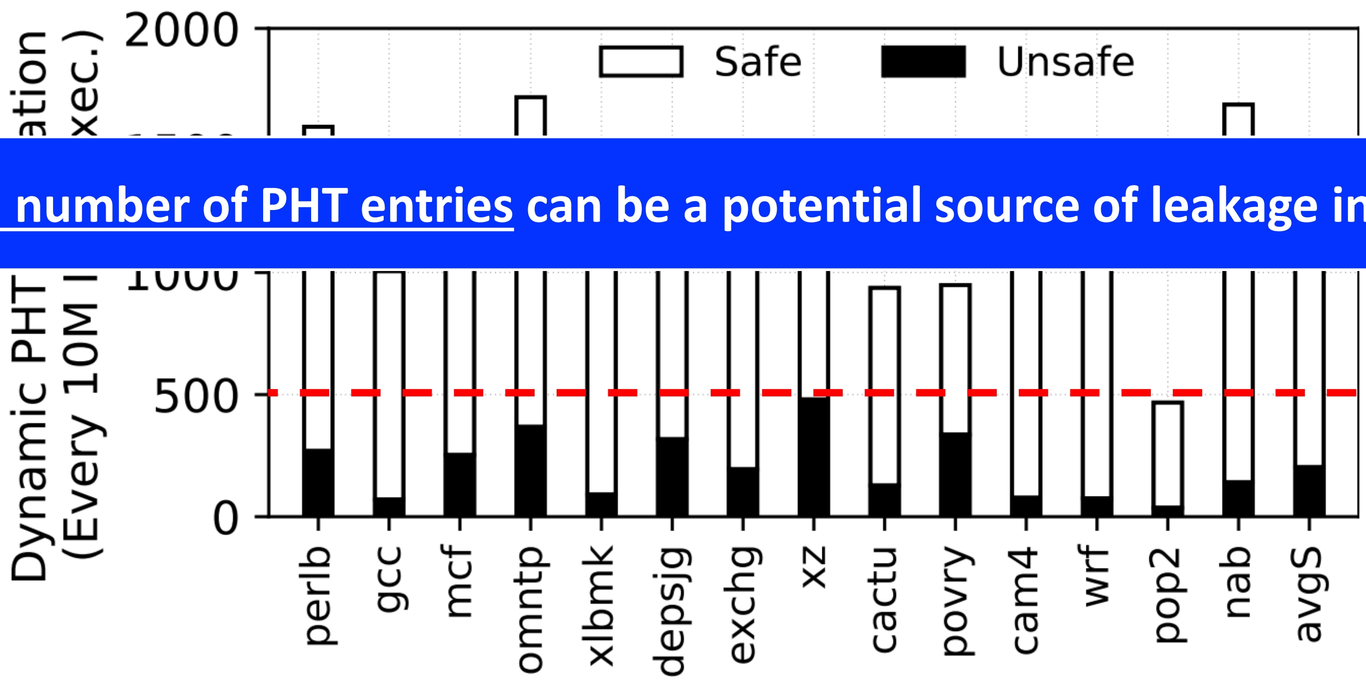


Fewer than eight PHT entries are considered potentially IS-Unsafe for most CPU cycles.

Figure: Runtime distribution of *potentially IS-Unsafe* entries during every CPU cycle of simulation (over 100M cycles).

Runtime Distribution of *PS-Unsafe* PHT Entries

- Record PHT entry update traces for continuous execution of 100 program segments.
  - Each segments with 10M instructions.

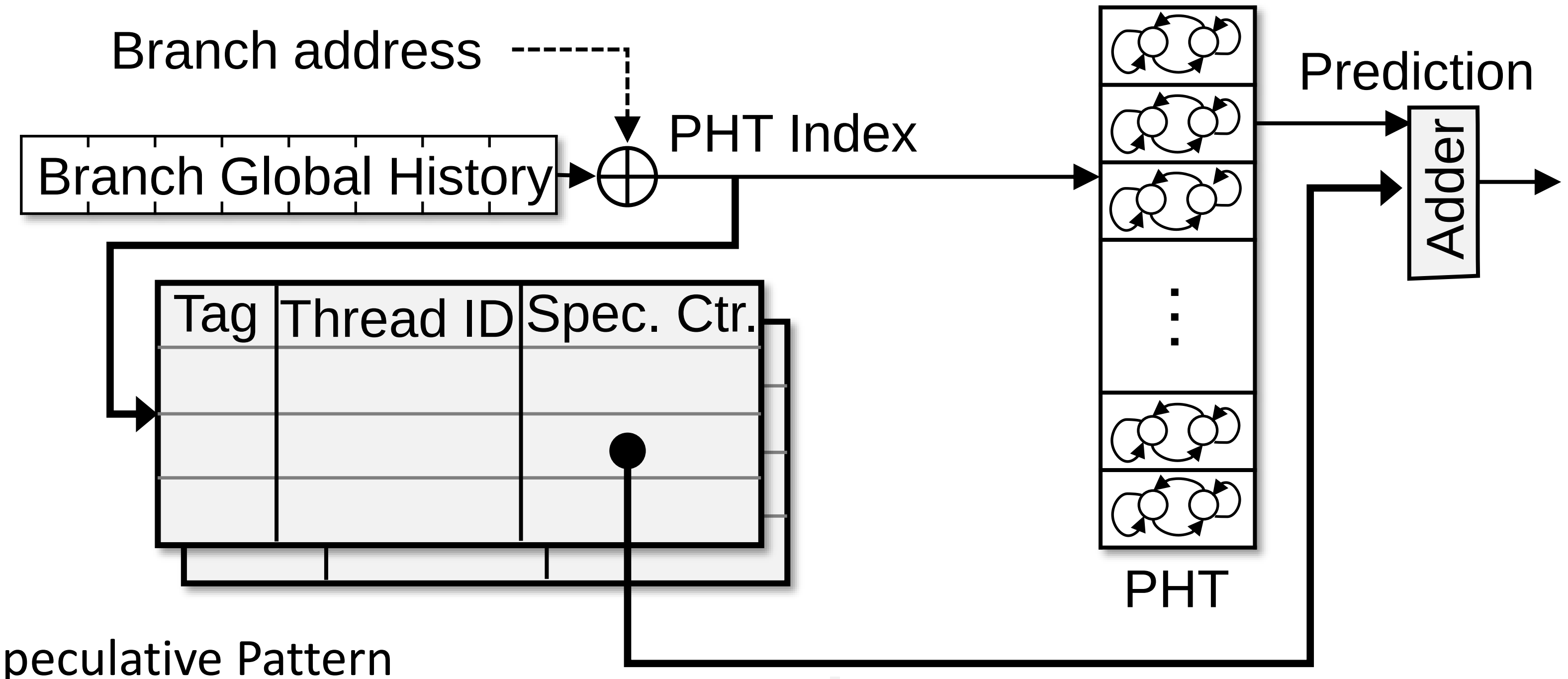


Very small number of PHT entries can be a potential source of leakage in speculative domain.

Very few PHT entries are PS-Unsafe (<202 on average out of 16,000 PHT entries).

Figure: Post-speculation safe/unsafe PHT entries in a history-based predictor.

BeKnight Framework: Overview



Speculative Pattern Lookaside Buffer (**SPLB**)

Integrate a small-size buffer to track and audit the usage of unsafe PHT entries.

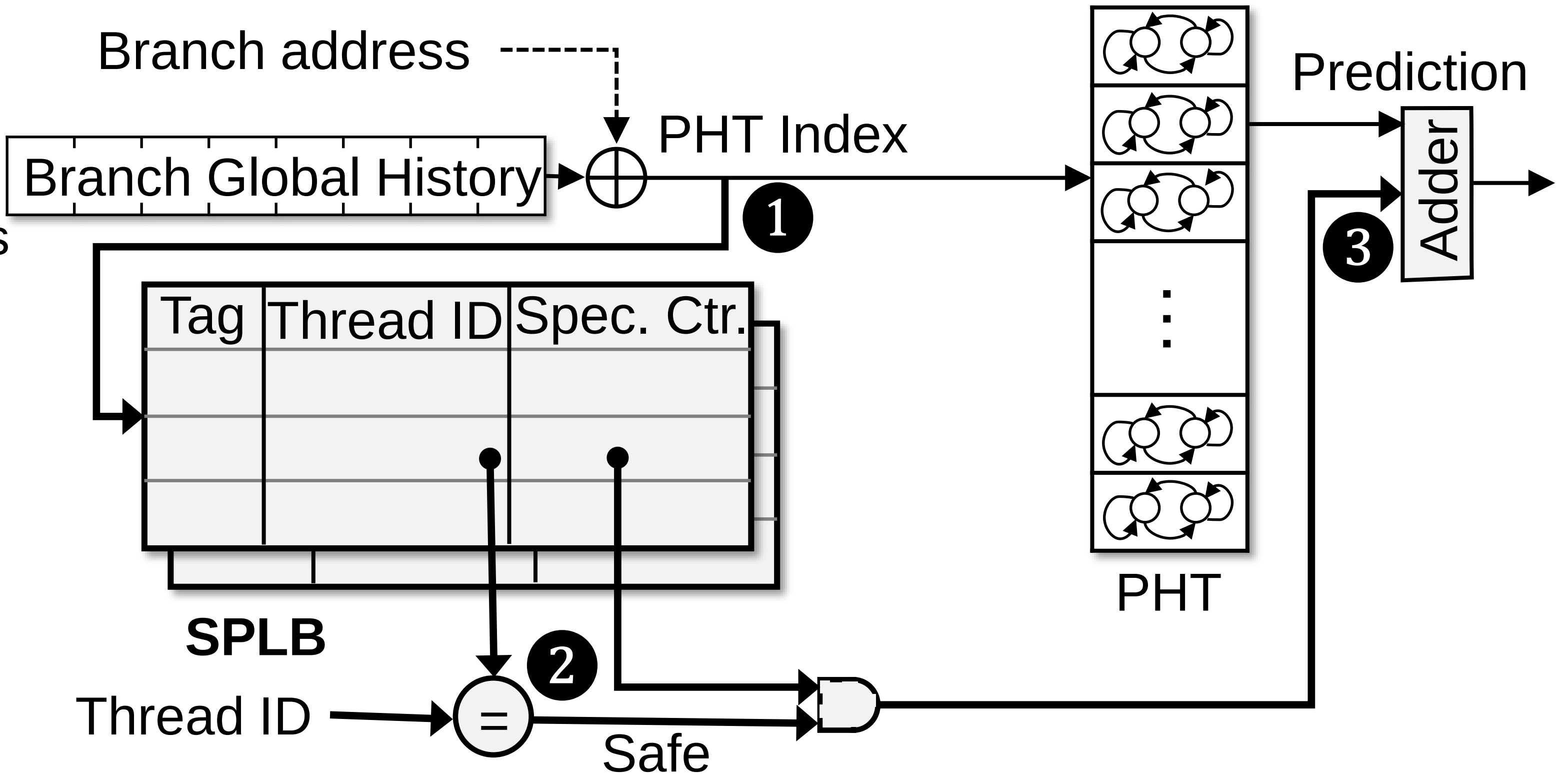
BeKnight Details: Fetch Stage

❶ SPLB Lookup is performed simultaneously with PHT lookup.

❷ Lookup can return one of three states:

1. *Buffer hit*: Speculative updates in SPLB is <u>safe</u>.
2. *Domain conflict*: Speculative updates in SPLB is <u>not safe</u>.
3. *Buffer miss*: No speculative update in SPLB.

❸ For *buffer hit*, SPLB is used along with PHT to provide prediction. For other cases, only PHT is used to provide prediction.

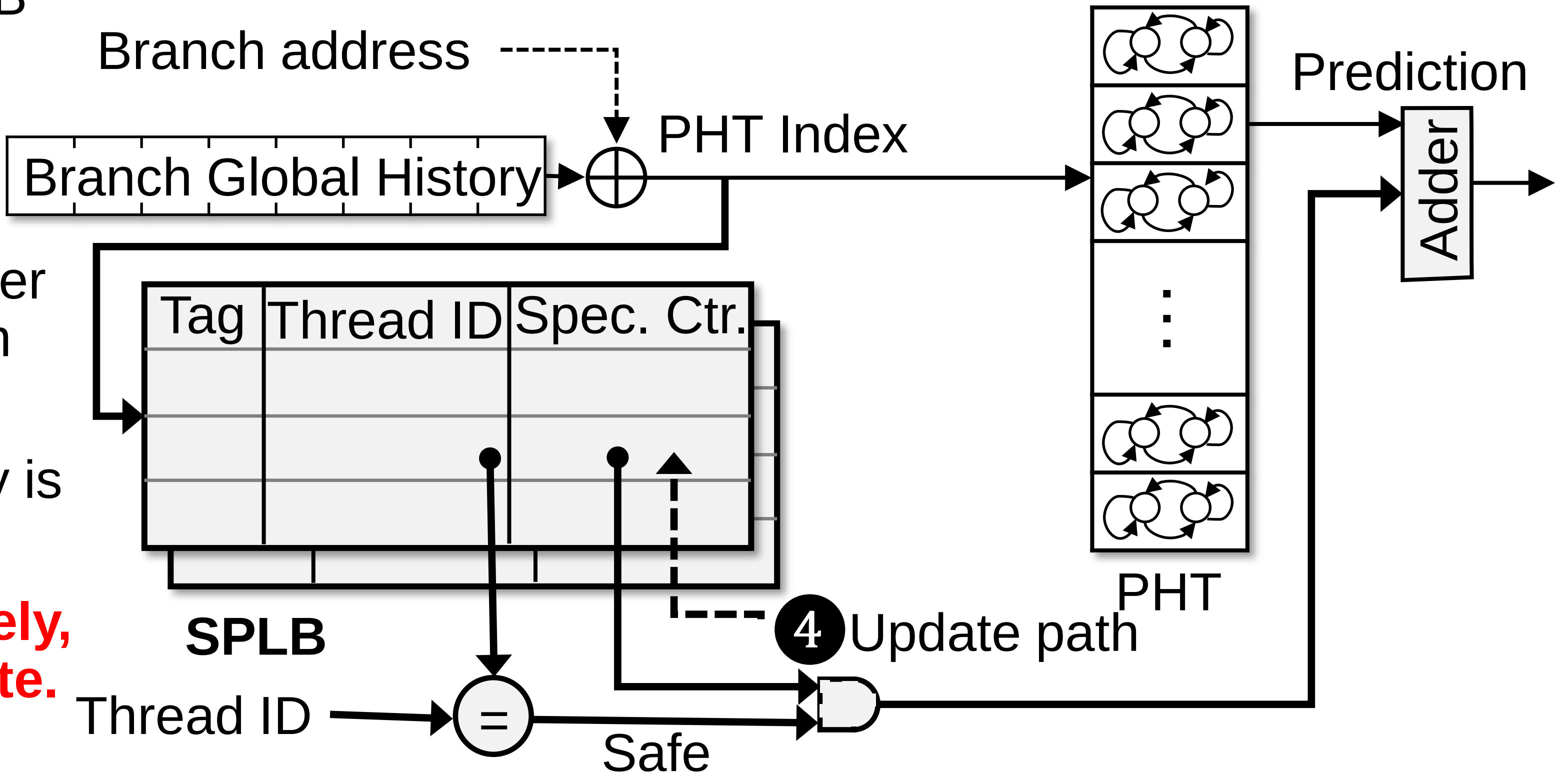


BeKnight Details: Execute Stage

❹ After branch is resolved, SPLB is updated based on resolution.

1. *Buffer hit:* SPLB counter is updated.
2. *Domain conflict:* SPLB counter is discarded (reset to 0), then updated.
3. *Buffer miss:* New SPLB entry is created.

PHT is not updated speculatively, maintains the architectural state.



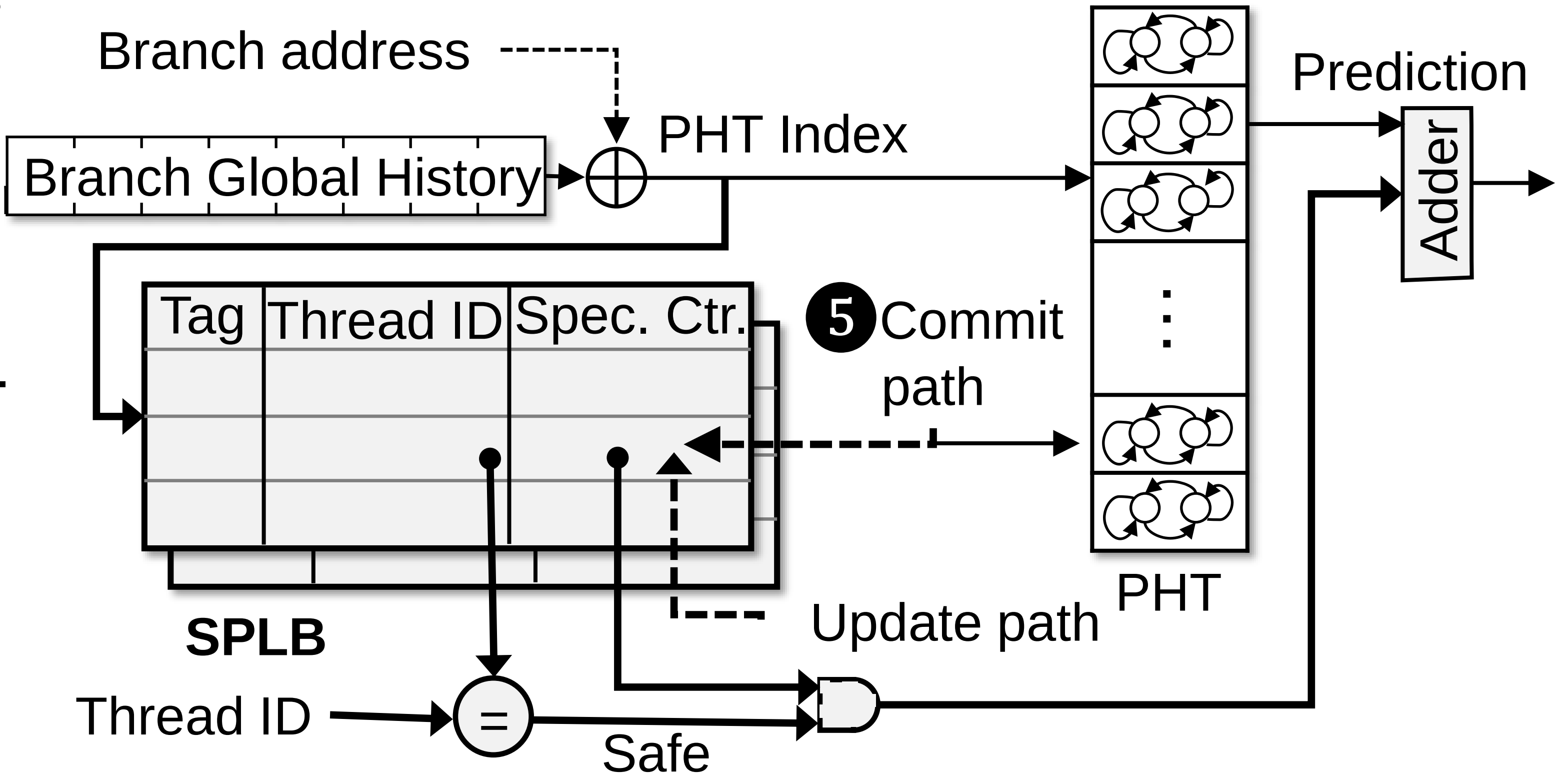
BeKnight Details: Commit Stage

❺ At commit stage, PHT state is updated to maintain correct architectural correct.

1. *Buffer hit:* Update the counte in the opposite direction of branch direction to maintain only the speculative updates.
2. *Domain conflict:* No change.
3. *Buffer miss:* No change.

Only committed instructions update the PHT.

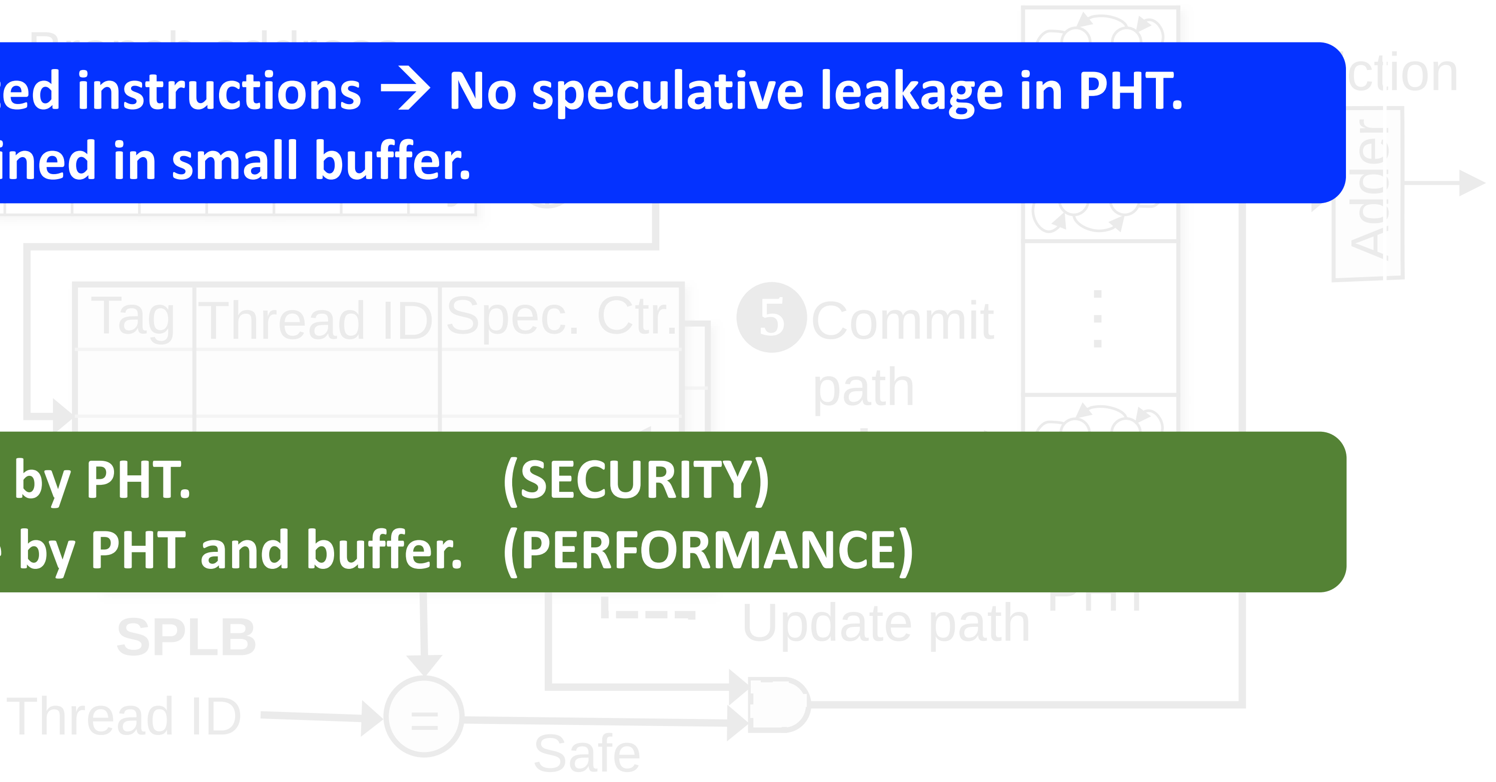
Updates from squashed instructions only exist in SPLB.



BeKnight Details

5 At commit stage, PHT state is

in the opposite direction of branch direction to maintain only the speculative updates.



- **PHT is only updated by committed instructions → No speculative leakage in PHT.**
- **Speculative updates are maintained in small buffer.**

- **Cross-domain predictions made by PHT. (SECURITY)**
- **Same-domain predictions made by PHT and buffer. (PERFORMANCE)**

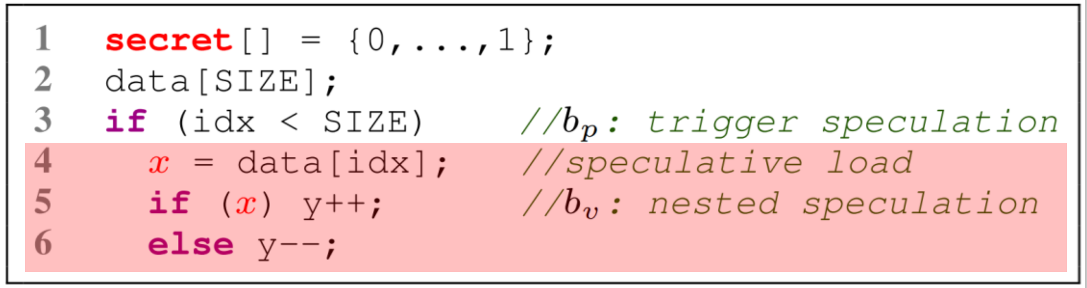
Only committed instructions update the PHT.

Updates from squashed instructions only exist in SPLB.

Evaluation Methodology

- Gem5 full-system simulation (Out-of-order X86 system).
  - **OS:** Ubuntu 18.04
  - **Kernel:** Linux 4.19.83
- **Branch predictor:** Hybrid branch predictor, 16K 3-bit saturating PHT, 64-bit GHR (Modeled similar to recent Intel processors).
- **Benchmarks:** 14 single-, 10 multi-program SPEC2017 workloads.
- **Schemes:**
  - Commit-time update (**CT**) → ***Secure***
  - Resolution-time update without restore (**RT-NR**) → ***Unsecure, but performant***
  - Our scheme (**BeKnight**)

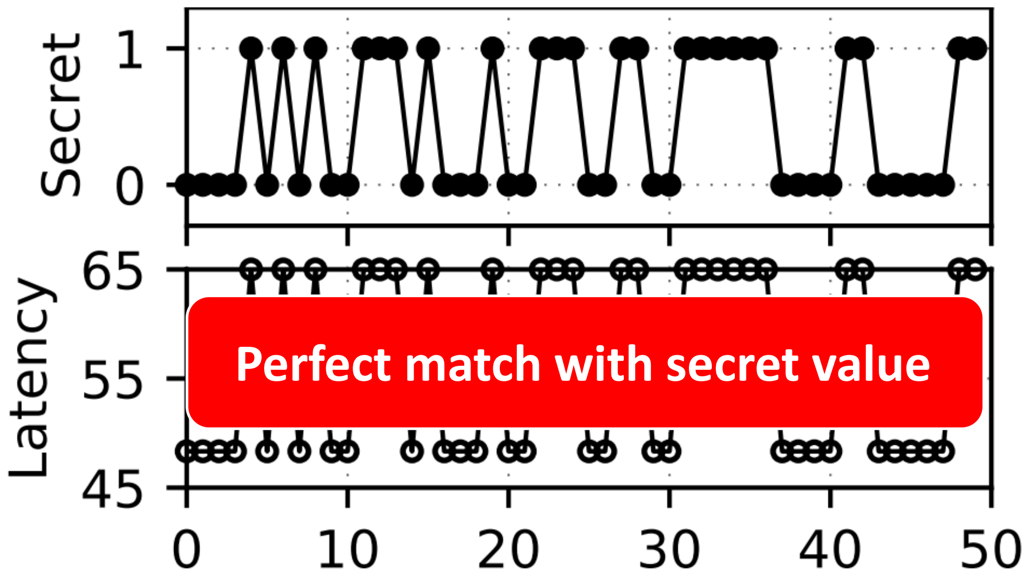
Evaluation: BeKnight Security



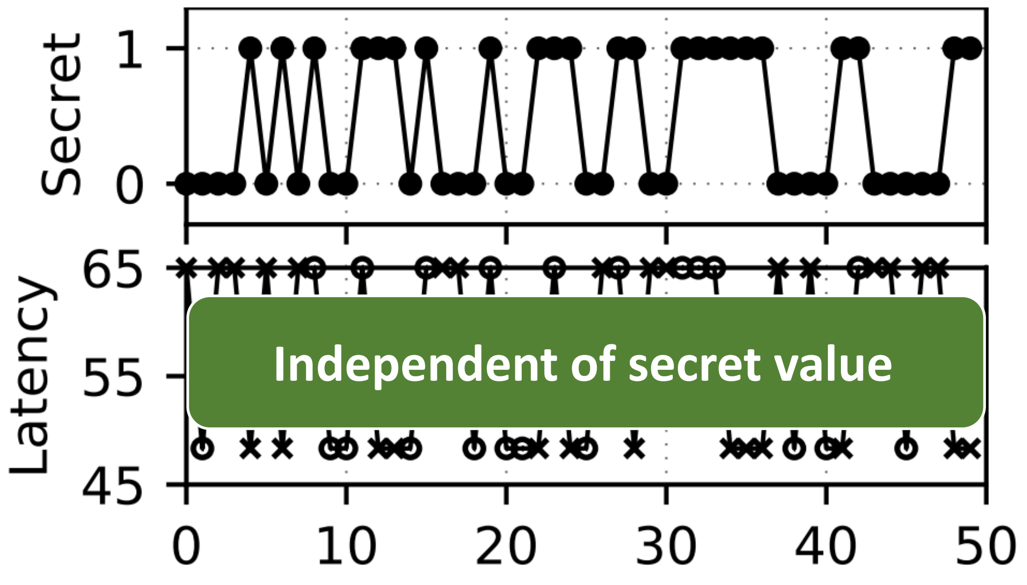
```
secret[] = {0,...,1};
data[SIZE];
if (idx < SIZE)       //b_p: trigger speculation
  x = data[idx];      //speculative load
  if (x) y++;         //b_v: nested speculation
  else y--;
```

- Branch $\mathbf{b_v}$ is executed under speculation of $\mathbf{b_p}$.
- Actual direction of $\mathbf{b_v}$ is dependent on **secret**. (data[idx] points to secret under speculation).
- Attacker observes execution latency of congruent branch: $\mathbf{b_a}$ to infer $\mathbf{b_v}$ direction.

Execution Latency of $\mathbf{b_a}$ corresponding to *actual direction* of $\mathbf{b_v}$.



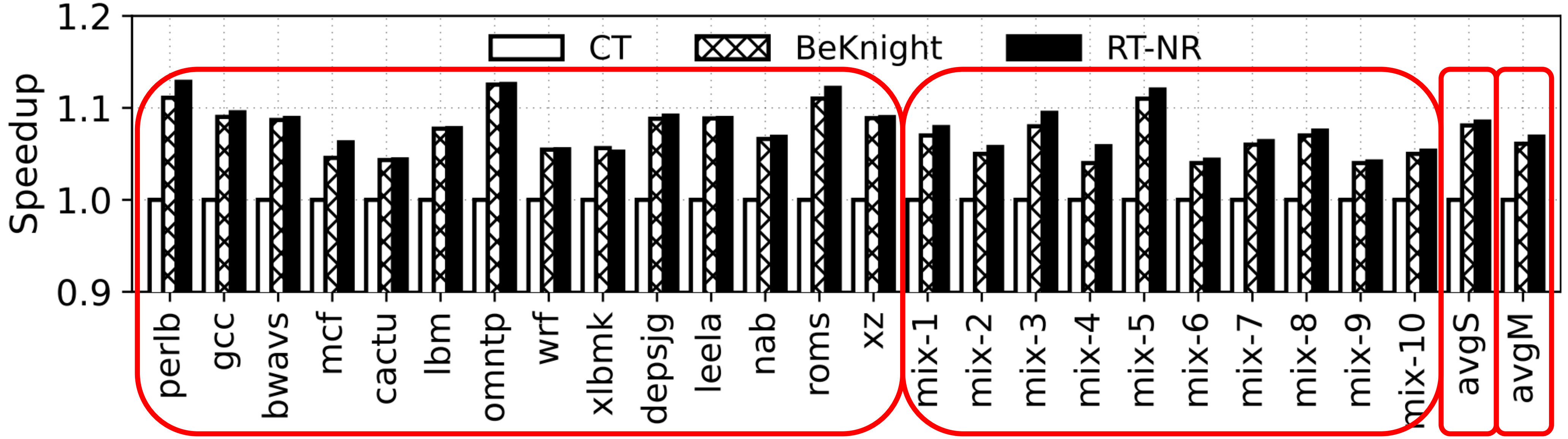
RT-NR



BeKnight

Evaluation: BeKnight Performance

Comparison of system performance (normalized to **CT**)



Non-SMT Configuration

BeKnight performance compared to **CT**:

- **8.1% speedup in single-program workloads.**
- **6.1% speedup in multi-program workloads.**

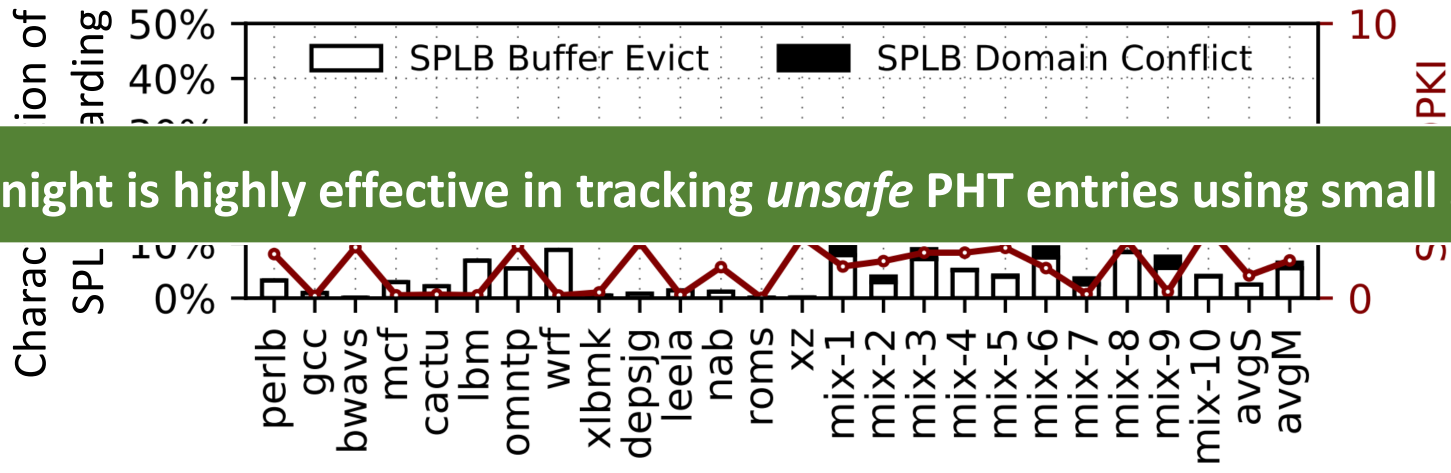
Minimal performance drop over RT-NR (0.3% and 0.8% in single- and multi-program workloads.

Evaluation: BeKnight Performance

BeKnight can influence system performance (compared to RT-NR) in two cases:

- SPLB **eviction** for an unsafe entry because of limited capacity.
- SPLB **domain conflict** when an unsafe entry in SPLB is accessed by another domain.

Breakdown of BeKnight SPLB discard per squash of resolved branches



BeKnight is highly effective in tracking *unsafe* PHT entries using small SPLB.

Low SPLB discard rate: 2.6% in single-program and 6.1% multi-program workloads.
No *Domain Conflict* in single-program workloads, only 1% in multi-program workloads.

Evaluations: Hardware Overhead

- Branch predictor storage overhead: **432 Bytes** for **SPLB**

- BeKnight on-chip logic:
  - Implemented using Verilog
  - Synthesized with Synopsis DC with 22nm

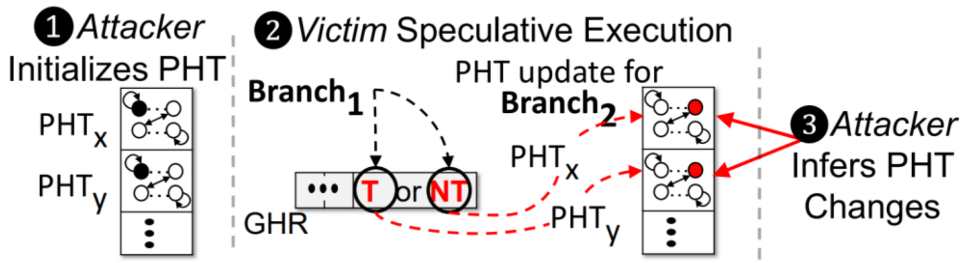
Size	**Latency** (ns)	**Energy** (nJ)	**Area** (mm^2)
64	0.067	0.0002	0.0003
128	0.071	0.0002	0.0004
256	0.085	0.0003	0.0006
512	0.087	0.0004	0.0013

More on Paper

- **Motivation and Problem analysis for Speculatively-updated predictors.**
- **Additional details on BeKnight design:**
  - Complete interaction of BeKnight logic with frontend pipeline.
  - SPLB structure and update procedure.
- **BeKnight result analysis:**
  - SMT performance evaluation.
- **Additional security analysis of BeKnight scheme.**
- **And more…**

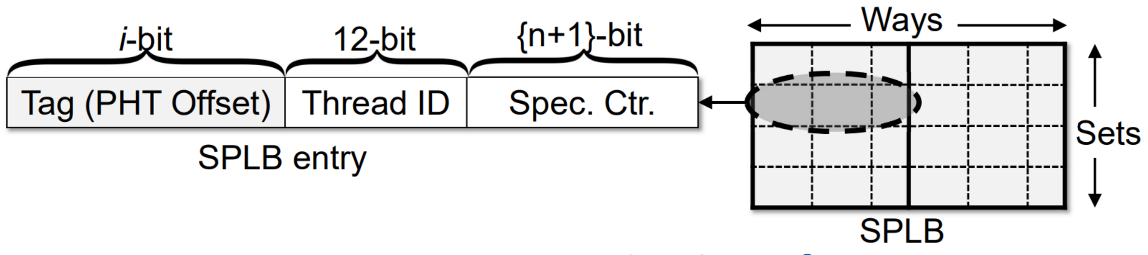
	Non-SMT		SMT	
	Direction	Execution	Direction	Execution
PHT Leakage	✓	✓	✓	✓
GHR Leakage	✓	✓	✓	✓

Security Evaluation of BeKnight



1 Attacker Initializes PHT
PHTx
PHTy
2 Victim Speculative Execution
Branch1
PHT update for Branch2
GHR
T or NT
PHTx
PHTy
3 Attacker Infers PHT Changes

Leakage of GHR



Organization of SPLB

Conclusion

- Secure speculatively-updated branch predictor design.
- Retain the performance benefit of *early* predictor update.
  - Ensure zero leakage from speculative domain.
- Track speculative predictor updates in a very small SPLB.
  - Architecturally correct states are always reflected in PHT.
- Same-domain accesses utilize both SPLB and PHT, cross-domain accesses only utilize PHT.
- Performs almost same as the insecure high-performance baseline.

Thanks!

Md Hafizul Islam Chowdhuryy
CASR Lab (https://casr.ece.ucf.edu)
Email: reyad@knights.ucf.edu