BranchSpec: Information Leakage Attacks Exploiting Speculative Branch Instruction Executions

Md Hafizul Islam Chowdhuryy

University of Central Florida
Orlando, FL

Hang Liu

Stevens Institute of Technology
Hoboken, NJ

Fan Yao

University of Central Florida
Orlando, FL

38th IEEE International Conference on Computer Design
October 18 - 21, 2020

Background

- ❖ Security issues of speculation are raising critical concerns.
- ❖ Microarchitectural state changes remain beyond speculation.
- ❖ Unintended data could be exfiltrated via side channels.
  - E.g., Spectre and Meltdown.
  - Demonstrated using Cache, TLB and function units.

Motivation

- ❖ Branch predictor unit (BPU) is one of the most critical components
- ❖ BPU is used to trigger mis-speculation in transient execution attacks
- ❖ BPU can transfer secret in non-speculative domain (e.g., BranchScope[1])

1. BranchScope: A New Side-Channel Attack on Directional Branch Predictor, ASPLOS’2018

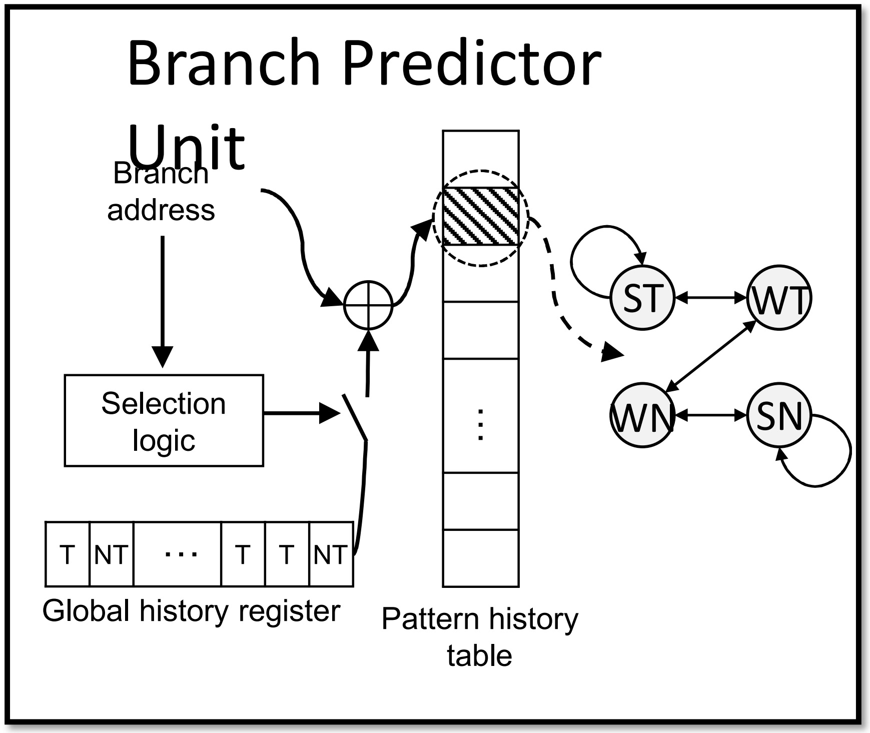
Motivation

- ❖ Branch predictor unit (BPU) is one of the most critical components
- ❖ BPU is used to trigger mis-speculation in transient execution attacks
- ❖ BPU can transfer secret in non-speculative domain (e.g., BranchScope[1])

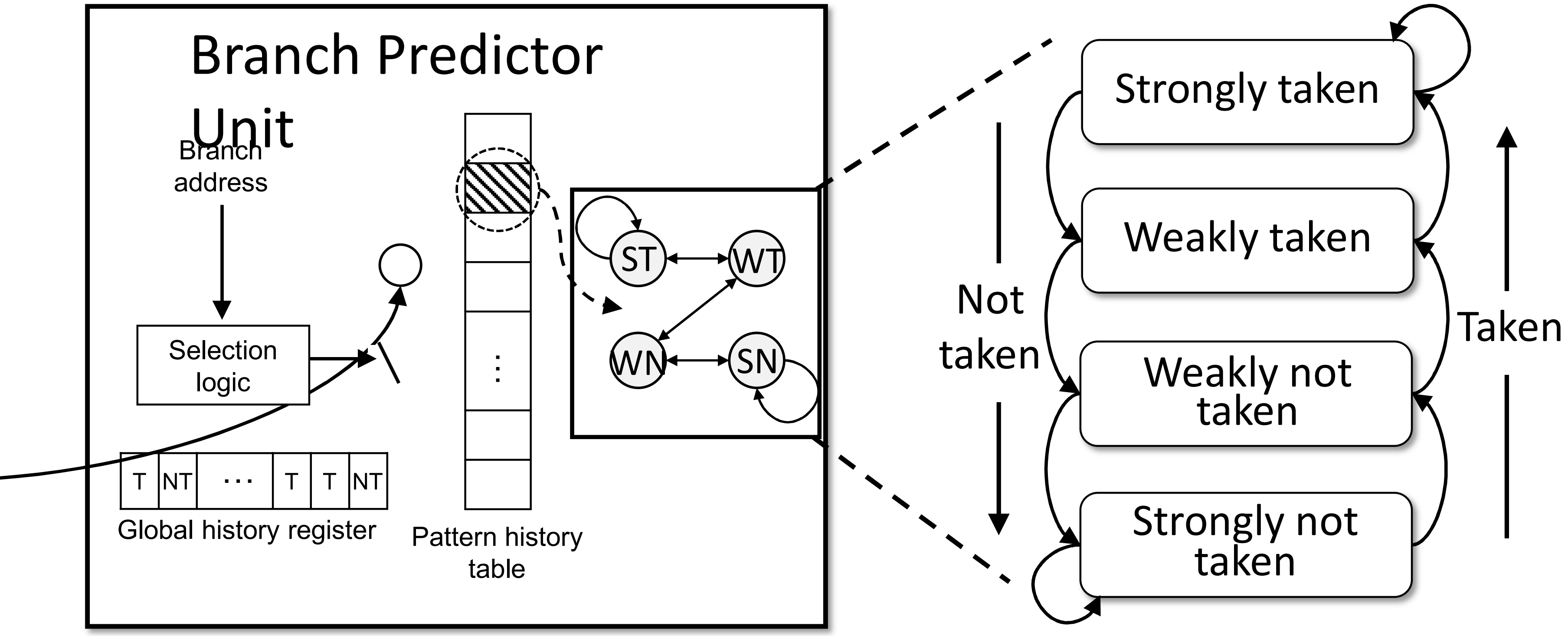
- ❖ Can we use branch predictor as **transmitting medium** in transient execution domain?

1. BranchScope: A New Side-Channel Attack on Directional Branch Predictor, ASPLOS'2018

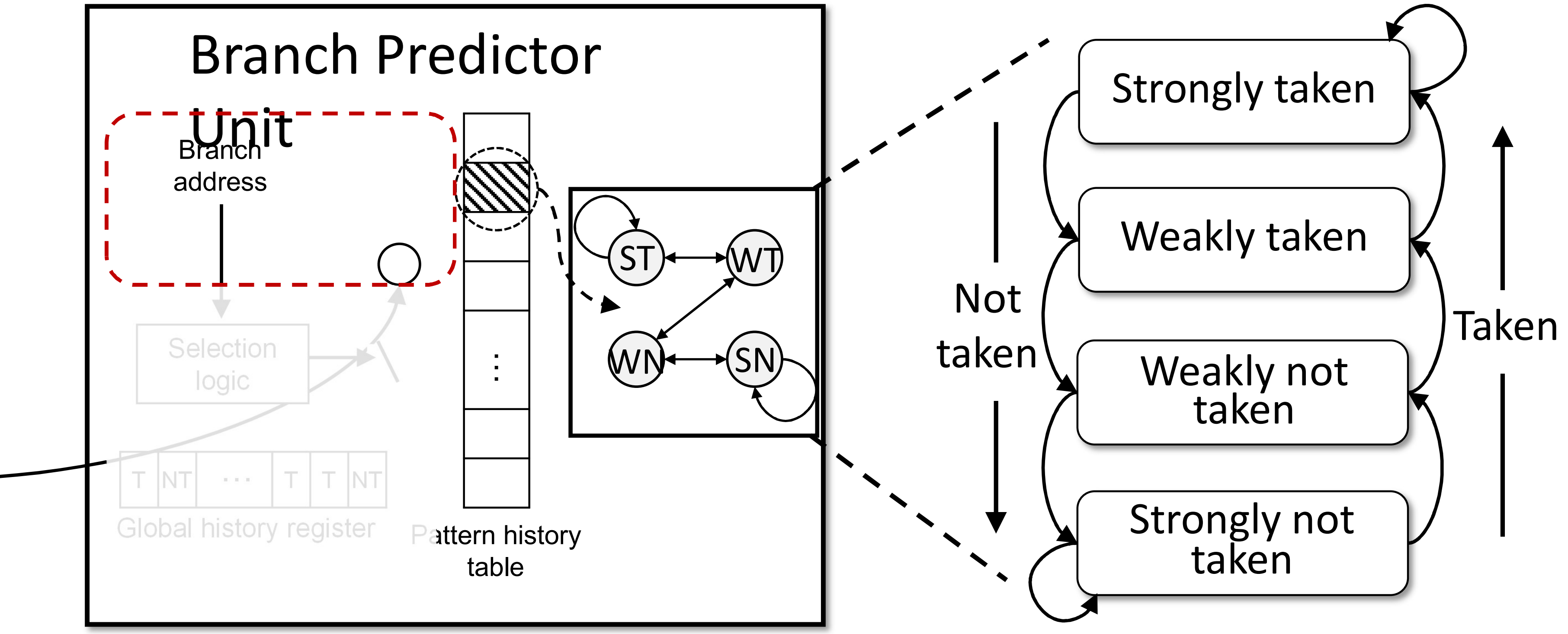
Modern Branch Predictor Architecture



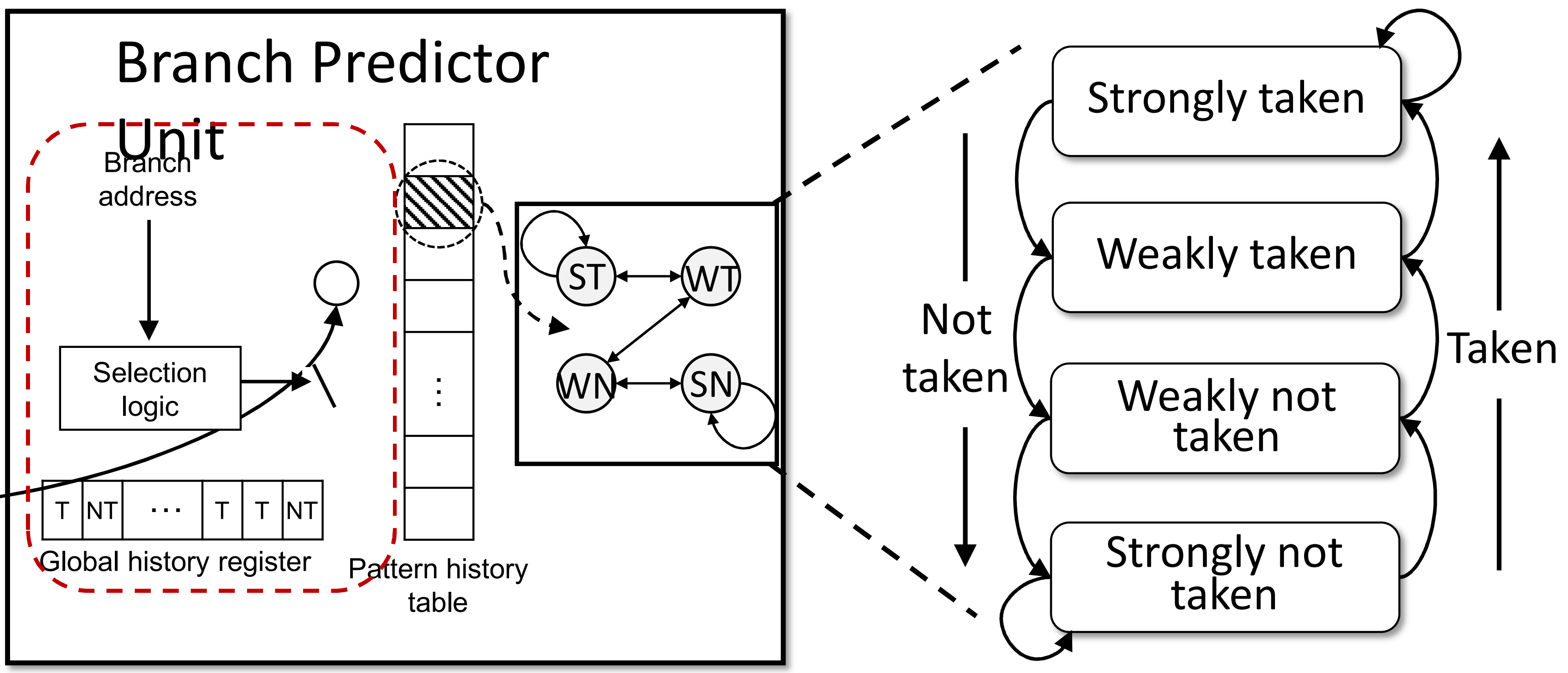
Modern Branch Predictor Architecture



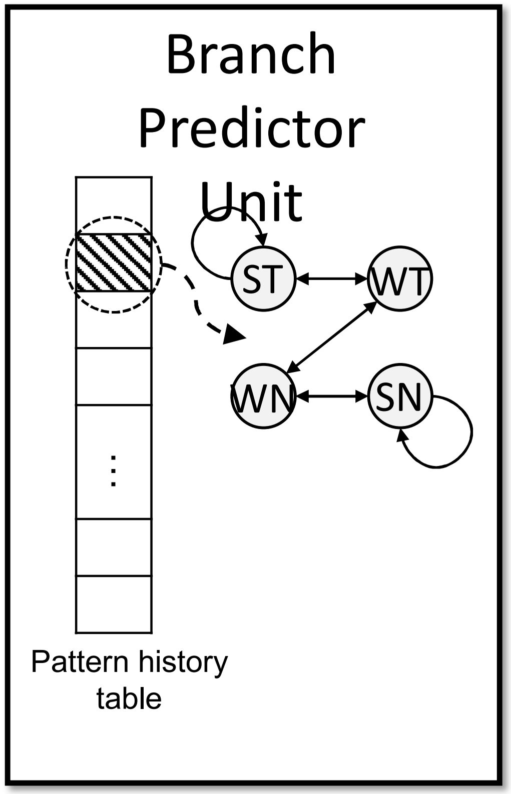
Modern Branch Predictor Architecture



Modern Branch Predictor Architecture

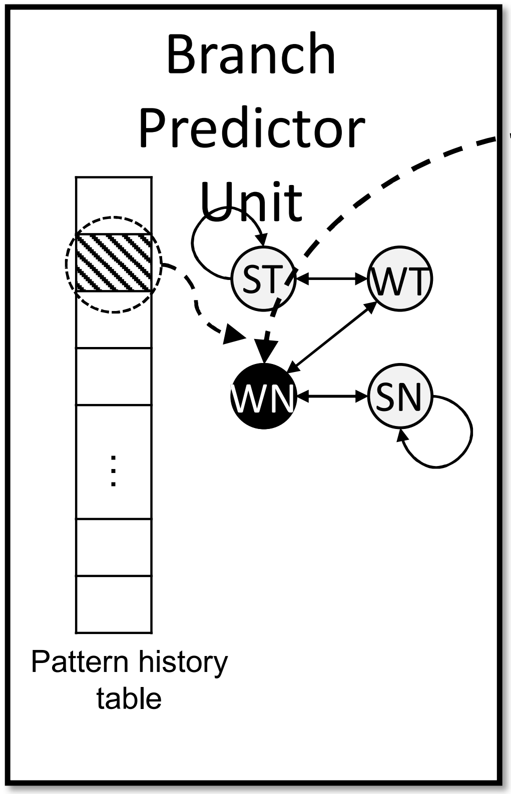


Do PHT Changes Remain After Speculation?



```
  bool control;
① if (i < bound)
  ② if (control)
      <some_operations>;
```

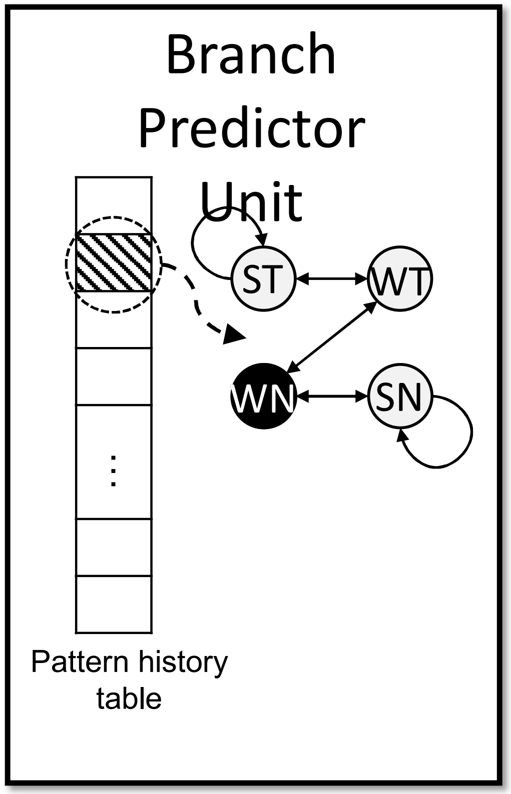
Do PHT Changes Remain After Speculation?



1. *Initialize {not-taken}* the PHT state

```
  bool control;
① if (i < bound)
  ② if (control)
       <some_operations>;
```

Do PHT Changes Remain After Speculation?

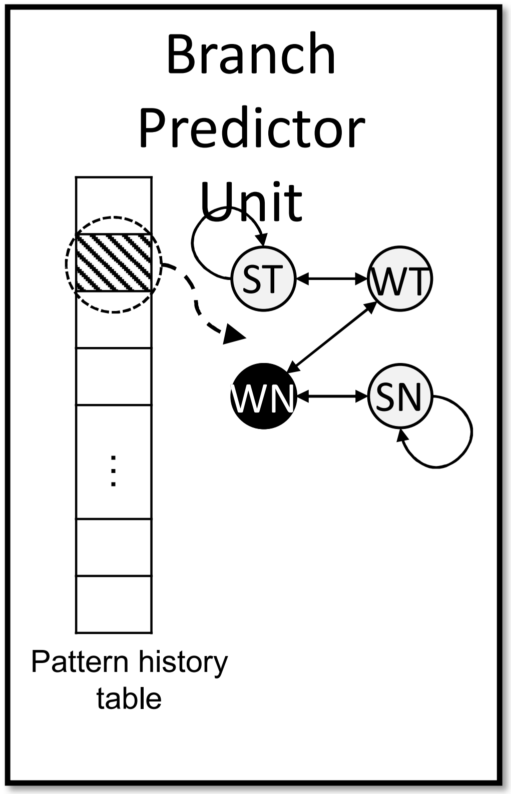


1. ***Initialize {not-taken}*** the PHT state

2. Trigger ***speculation***

```
  bool control;
① if (i < bound)
  ② if (control)
      <some_operations>;
```

Do PHT Changes Remain After Speculation?



1. ***Initialize {not-taken}*** the PHT state

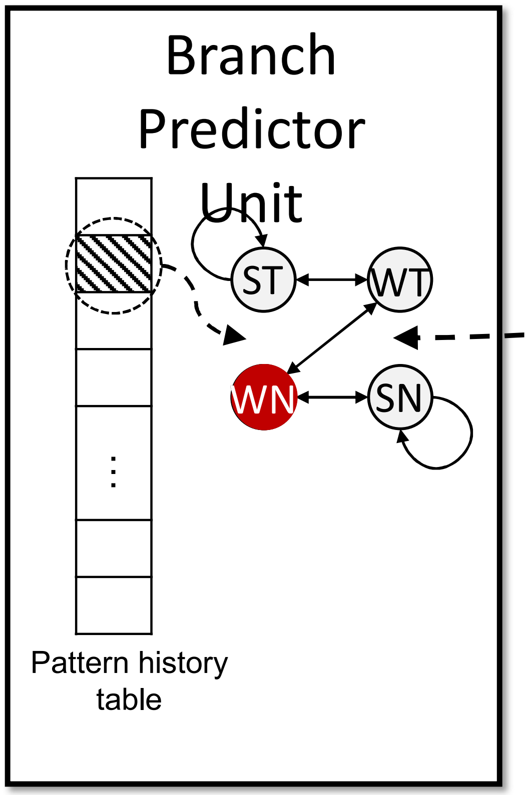
2. Trigger ***speculation***

```
   bool control;
① if (i < bound)
   ② if (control)
        <some_operat
```

Start of ***Speculation***

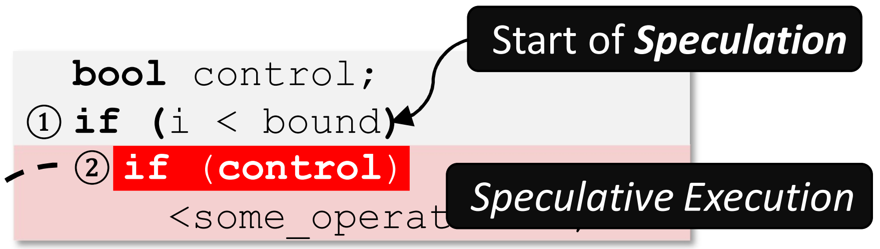
Speculative Execution

Do PHT Changes Remain After Speculation?



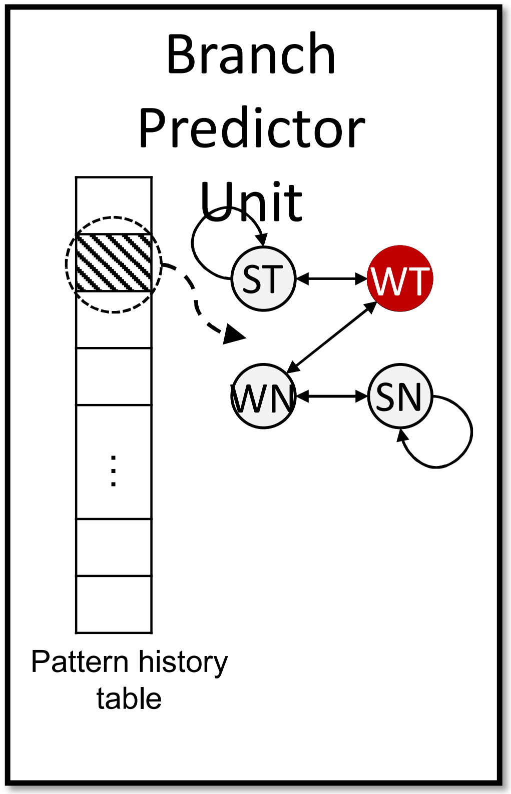
1. *Initialize {not-taken}* the PHT state

2. Trigger ***speculation***



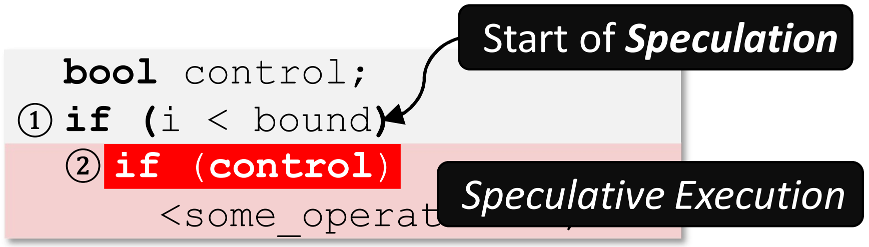
```
   bool control;
① if (i < bound)
  ② if (control)
       <some_operat
```

Do PHT Changes Remain After Speculation?



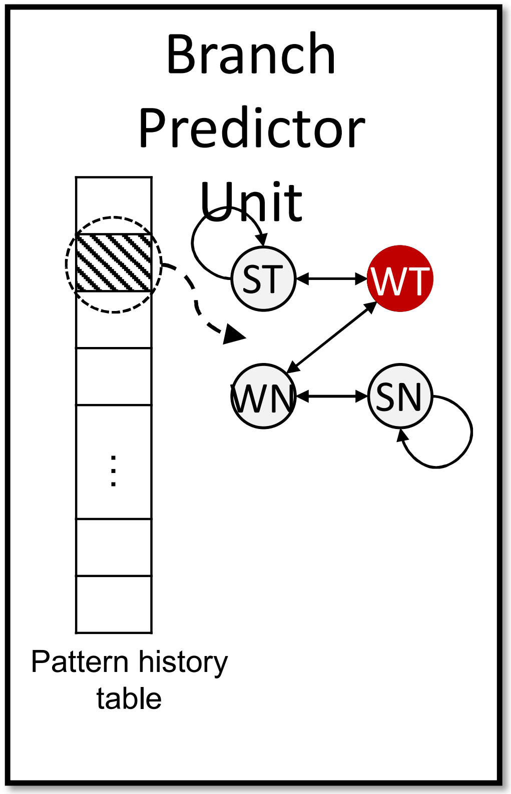
1. *Initialize {not-taken}* the PHT state

2. Trigger ***speculation***



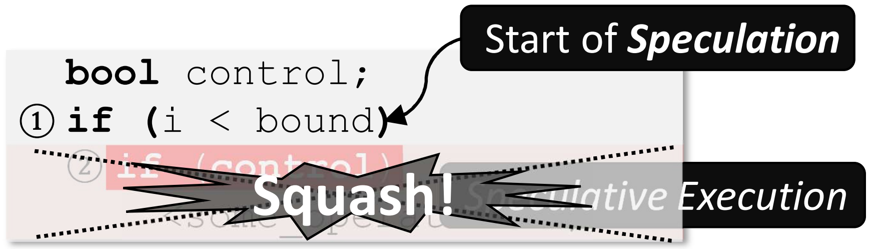
```
  bool control;
① if (i < bound)
  ② if (control)
      <some_operat
```

Do PHT Changes Remain After Speculation?

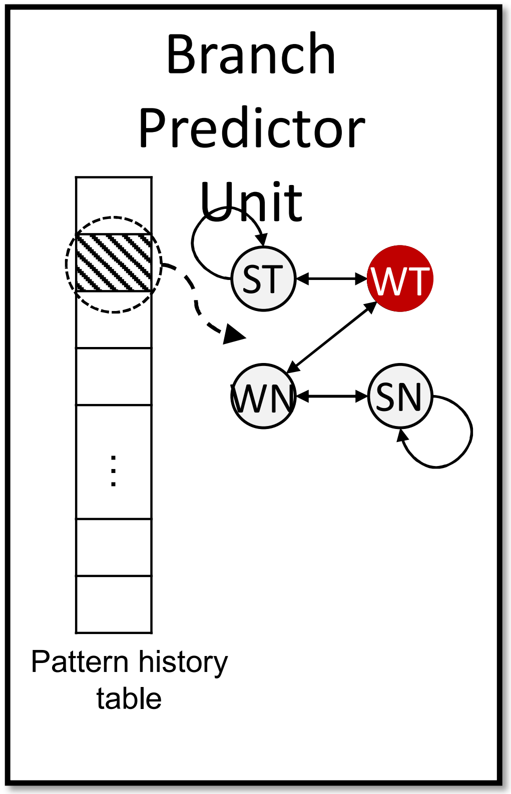


1. ***Initialize {not-taken}*** the PHT state

2. Trigger ***speculation***


```
  bool control;
① if (i < bound)
② if (control)
    <some_operat
```

Do PHT Changes Remain After Speculation?



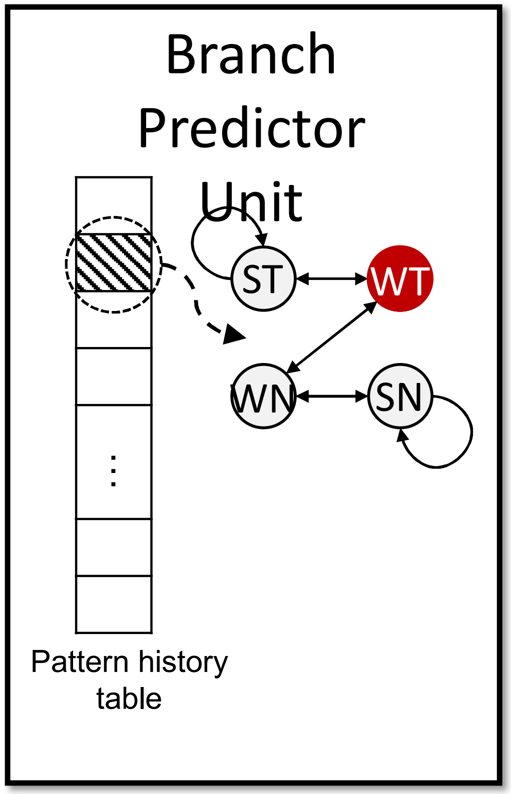
1. ***Initialize {not-taken}*** the PHT state

2. Trigger ***speculation***

3. Measure execution time ***{taken}***

```
  bool control;
① if (i < bound)
  ② if (control)
       <some_operations>;
```

Do PHT Changes Remain After Speculation?



1. ***Initialize {not-taken}*** the PHT state

2. Trigger ***speculation***

3. Measure execution time ***{taken}***

```
   bool control;
①  if (i < bound)
    ② if (control)
        <some_operations>;
```

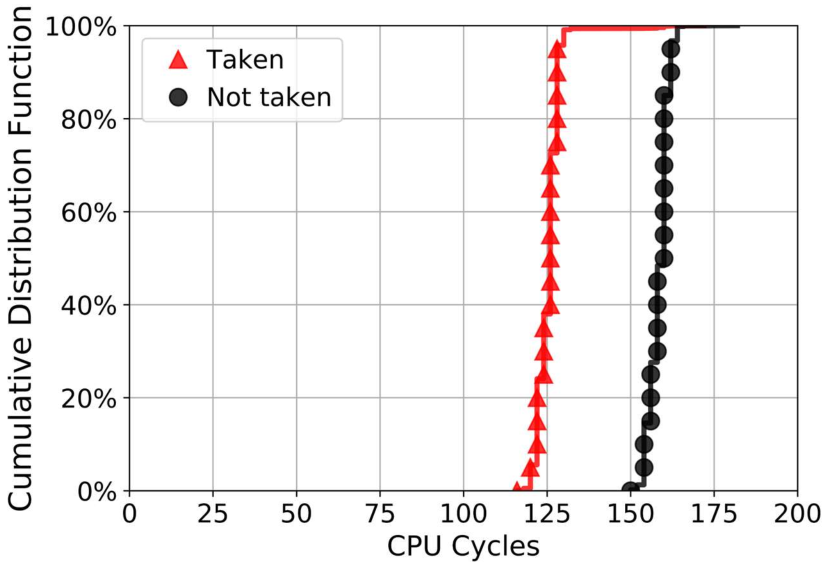
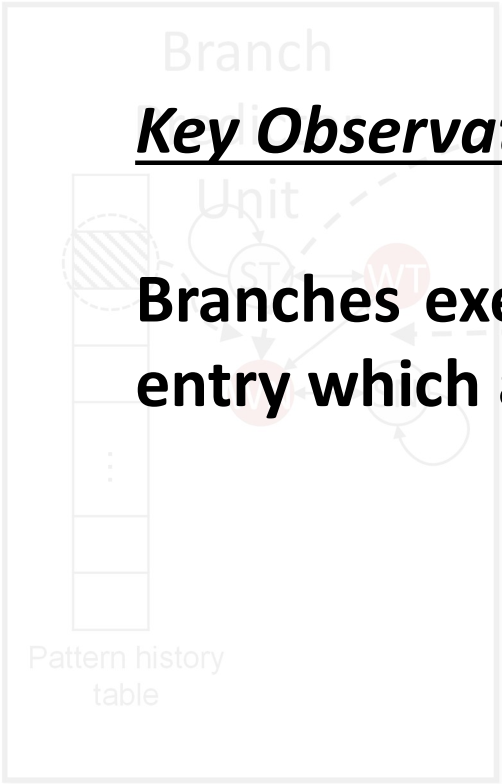


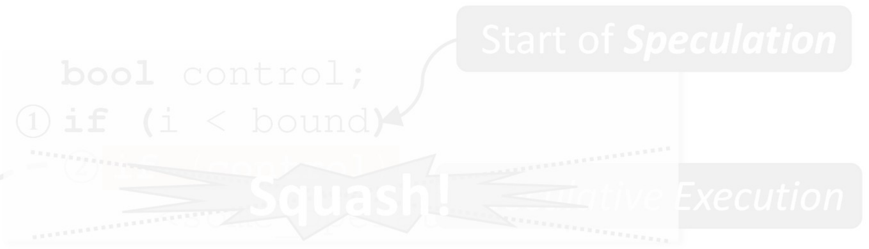
Figure 1: Execution time of branch ② in step 3 for different outcome of the branch in step 2.

Do PHT Changes Remain After Speculation?



1. ***Initialize {not-taken}*** the PHT state

Key Observation:



```
bool control;
① if (i < bound)
```

Branches executed in the speculative path change PHT entry which are not restored in case of mis-speculation.

3. Measure execution time ***{taken}***

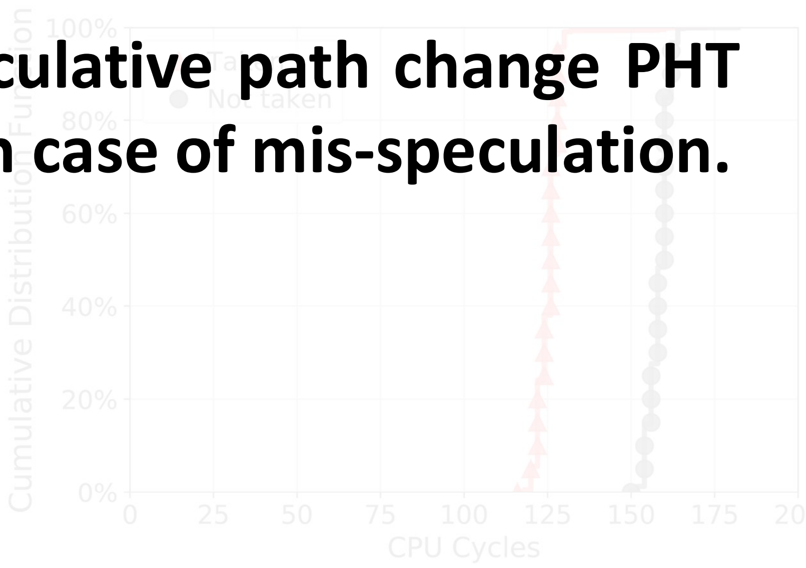


Figure 1: Execution time of branch ② in step 3 for different outcome of the branch in step 2

BranchSpec: Side Channel Attack

Victim

```
// Parent branch
if (x < bound)
  ....
  ....
  // Victim branch, b_V
  if (array1[x])
    <some_operations>;
```

BranchSpec: Side Channel Attack

Victim

```
// Parent branch
if (x < bound)
  ....
  ....
  // Victim branch, b_V
  if (array1[x])
    <some_operations>;
```

BranchSpec: Side Channel Attack

Step 1: Preset PHT entry (PHT_v) of victim branch (b_v)

- Attacker uses a **congruent branch** of b_v (i.e., b_a)
- Executes b_a twice with *taken* outcome

Victim

```
// Parent branch
if (x < bound)
  ....
  ....
  // Victim branch, b_v
  if (array1[x])
    <some_operations>;
```

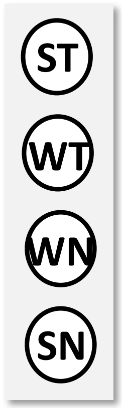
BranchSpec: Side Channel Attack

Step 1: Preset PHT entry (PHT$_v$) of victim branch (b$_v$)

- Attacker uses a **congruent branch** of b$_v$ (i.e., b$_a$)
- Executes b$_a$ twice with *taken* outcome

Victim

```
// Parent branch
if (x < bound)
  ....
  ....
  // Victim branch, bv
  if (array1[x])
    <some_operations>;
```



Initial state (PHT$_v$)

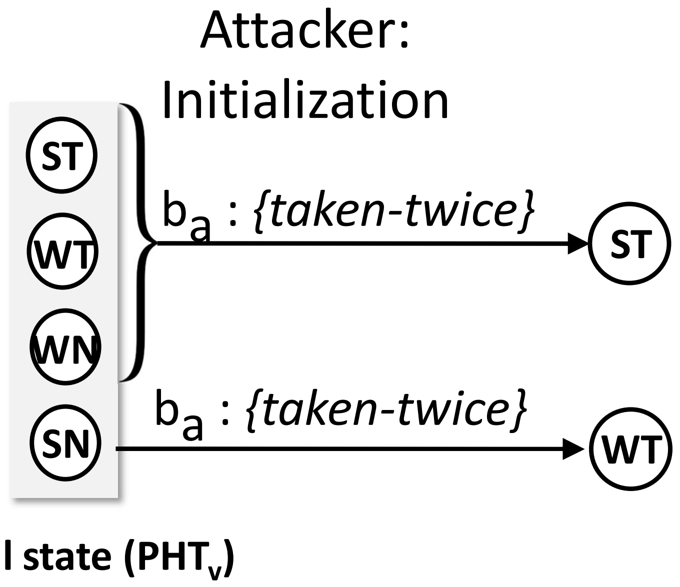
BranchSpec: Side Channel Attack

Step 1: Preset PHT entry (PHT_V) of victim branch (b_V)

- Attacker uses a **congruent branch** of b_V (i.e., b_a)
- Executes b_a twice with *taken* outcome

Victim

```
// Parent branch
if (x < bound)
  ....
  ....
  // Victim branch, bv
  if (array1[x])
    <some_operations>;
```



BranchSpec: Side Channel Attack

Step 2: Victim executes b_V **speculatively**

Victim

```
// Parent branch
if (x < bound)
  ....
  ....
  // Victim branch, bV
  if (array1[x])
    <some_operations>;
```

BranchSpec: Side Channel Attack

Step 2: Victim executes b_V **speculatively**

- Attacker can trigger mis-speculation of parent branch using congruent branch

Victim

```
// Parent branch
if (x < bound)
  ....
  ....
  // Victim branch, b_V
  if (array1[x])
    <some_operations>;
```

BranchSpec: Side Channel Attack

Step 2: Victim executes b_v **speculatively**

- Attacker can trigger mis-speculation of parent branch using congruent branch
- PHT entry of victim branch (PHT_v) is updated based on b_v outcome

Victim

```
// Parent branch
if (x < bound)
   ....
   ....
   // Victim branch, bv
   if (array1[x])
     <some_operations>;
```

BranchSpec: Side Channel Attack

Step 2: Victim executes b_V **speculatively**

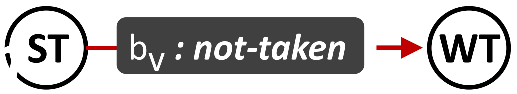
- Attacker can trigger mis-speculation of parent branch using congruent branch
- PHT entry of victim branch (PHT_V) is updated based on b_V outcome

Victim

```
// Parent branch
if (x < bound)
   ....
   ....
   // Victim branch, bV
   if (array1[x])
     <some_operations>;
```

Victim b_V
Speculative execution

b_V resolved as ***Not take***



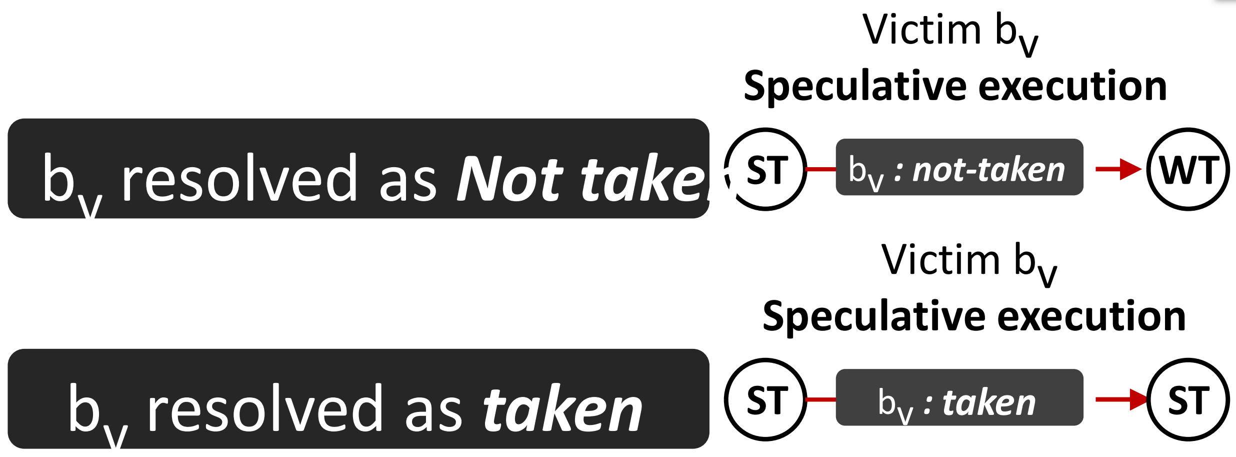
BranchSpec: Side Channel Attack

Step 2: Victim executes b_v **speculatively**

- Attacker can trigger mis-speculation of parent branch using congruent branch
- PHT entry of victim branch (PHT_v) is updated based on b_v outcome

Victim

```
// Parent branch
if (x < bound)
  ....
  ....
  // Victim branch, bv
  if (array1[x])
    <some_operations>;
```



BranchSpec: Side Channel Attack

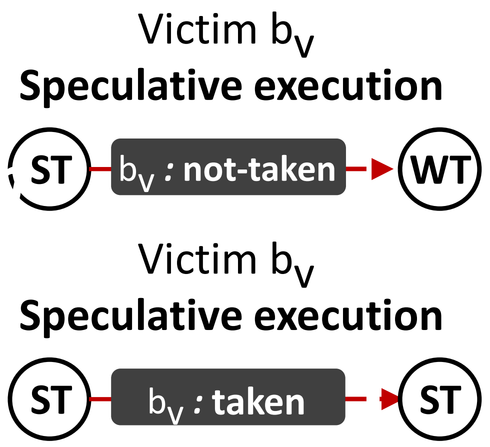
Step 3: Attacker probes PHT_v to infer b_v outcome

Victim

```
// Parent branch, bv0
if (x < bound)
   ....
   ....
   // Victim branch, bv
   if (array1[x])
     <some_operations>;
```

b_v resolved as ***Not take***

b_v resolved as ***taken***



BranchSpec: Side Channel Attack

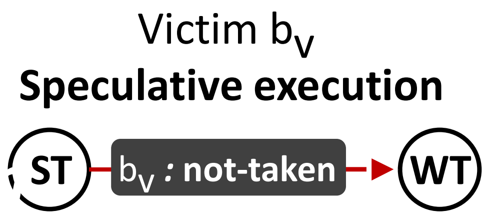
Step 3: Attacker probes PHT_V to infer b_V outcome

- Execute b_a twice with ***not taken*** outcome

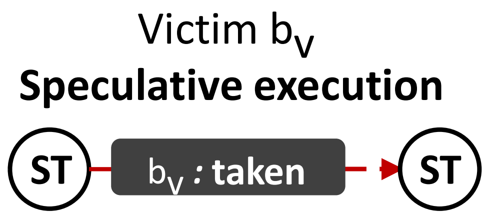
Victim

```
// Parent branch, b_v0
if (x < bound)
    ....
    ....
    // Victim branch, b_v
    if (array1[x])
      <some_operations>;
```

b_V resolved as ***Not take***



b_V resolved as ***taken***



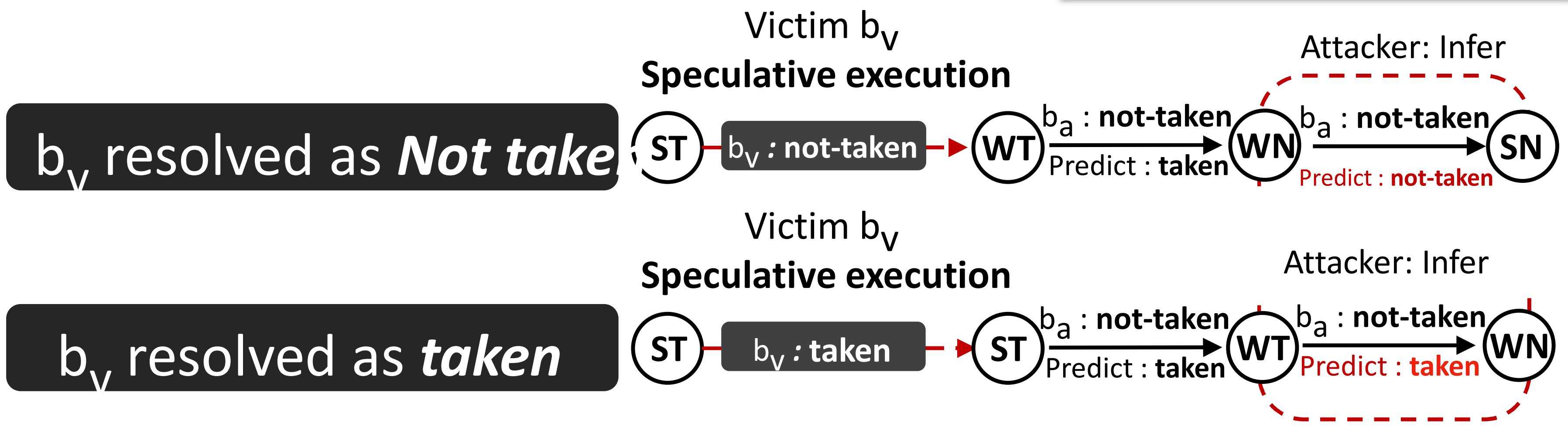
BranchSpec: Side Channel Attack

Victim

```
// Parent branch, b_v0
if (x < bound)
  ....
  ....
  // Victim branch, b_v
  if (array1[x])
    <some_operations>;
```

Step 3: Attacker probes PHT_v to infer b_v outcome

- Execute b_a twice with ***not taken*** outcome



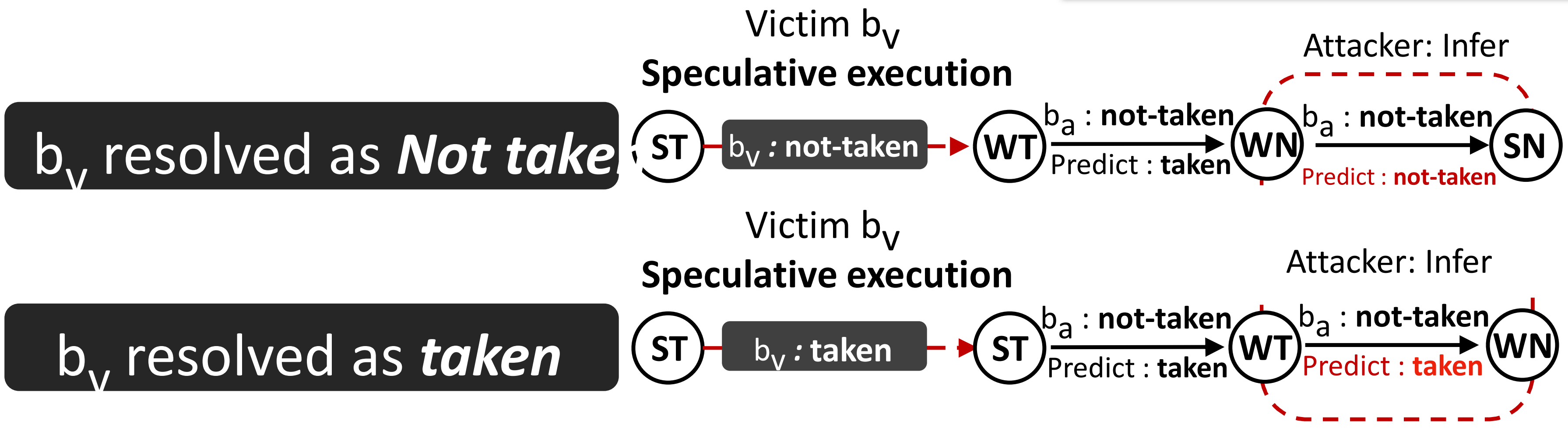
BranchSpec: Side Channel Attack

Step 3: Attacker probes PHT_v to infer b_v outcome

- Execute b_a twice with ***not taken*** outcome
- Measure execution time of **second** execution

Victim

```
// Parent branch, b_v0
if (x < bound)
    ....
    ....
    // Victim branch, b_v
    if (array1[x])
      <some_operations>;
```



BranchSpec: Side Channel Attack

Victim

```
// Parent branch, b_v0
....
// Victim branch, b_v
if (array1[x])
  <some_operations>;
```

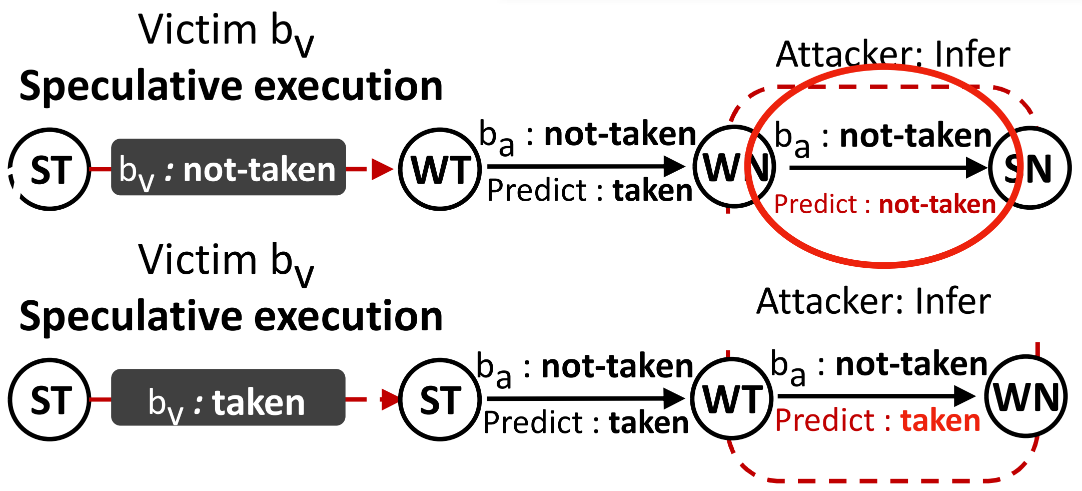
Step 3: Att

- Execute b_a twice with ***not taken*** outcome
- Measure execution time of **second** execution

Correct prediction of b_a -> Shorter execution time

b_v resolved as ***Not take***

b_v resolved as ***taken***



BranchSpec: Side Channel Attack

Victim

```
// Parent branch, b_v0
    ....
// ...anch, b_v
    <some_operations>;
```

Step 3: Att

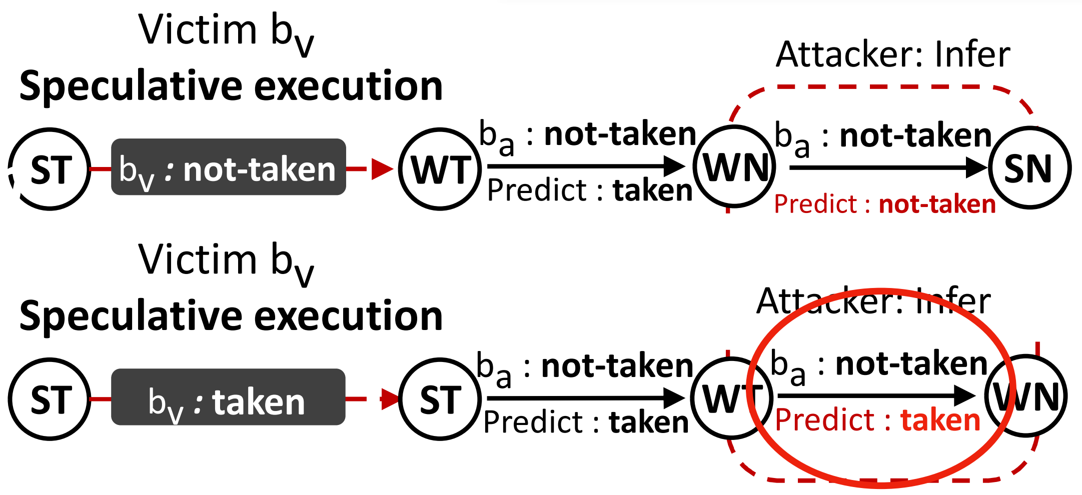
- Execute b_a twice with *not taken* outcome
- Measu

Correct prediction of b_a -> Shorter execution time

Mis-prediction of b_a -> Longer execution

b_v resolved as ***Not taken***

b_v resolved as ***taken***



Results and Characteristics of BranchSpec

- First work to show information leakage via branch predictor in transient execution attacks
  - Implemented on processors **with and w/o SMT**
  - Bit error rate is less than **4%**
  - Potentially targeted applications: **Crypto algorithms, image processing** and **ML programs**

- Enables even stronger attack capabilities
  - Completely uses BPU for end-to-end attack
  - Utilizes more common code patterns than Spectre V1

Spectre V1 Gadgets

```
if (x < array1_size)
  y = array2(array1[x] * 4096);
```

BranchSpec Gadgets

```
if (x < bound)
  if (array1[x])    // b_V
    <some_operations>;
```

```
if (x < bound)
  for (i = 0; i < bound; i++)
    if (array1[x + i]) // b_V
      <some_operations>;
```

```
for (i = x; i < bound; i++)
  if (array1[i]) // b_V
    <some_operations>;
```

BranchSpec: Covert Channel Attack

- Covert channel using BranchSpec
  - With optimizations, 131 Kbps transmission rate within 3.7% error rate

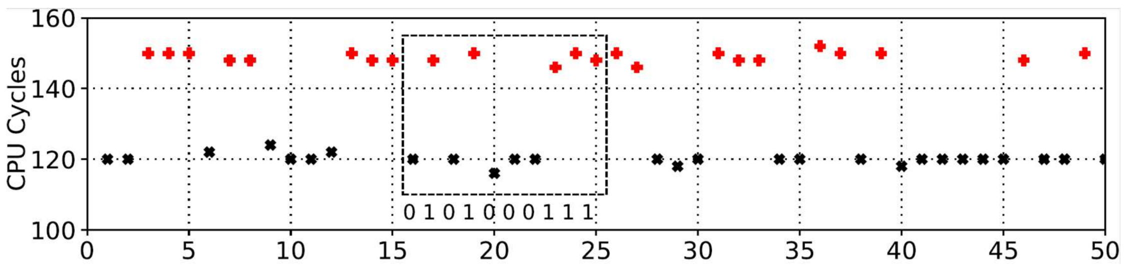


Figure 3: Latency traces for a 50-bit transmission by Spy corresponding to the covert channel in Figure 2.

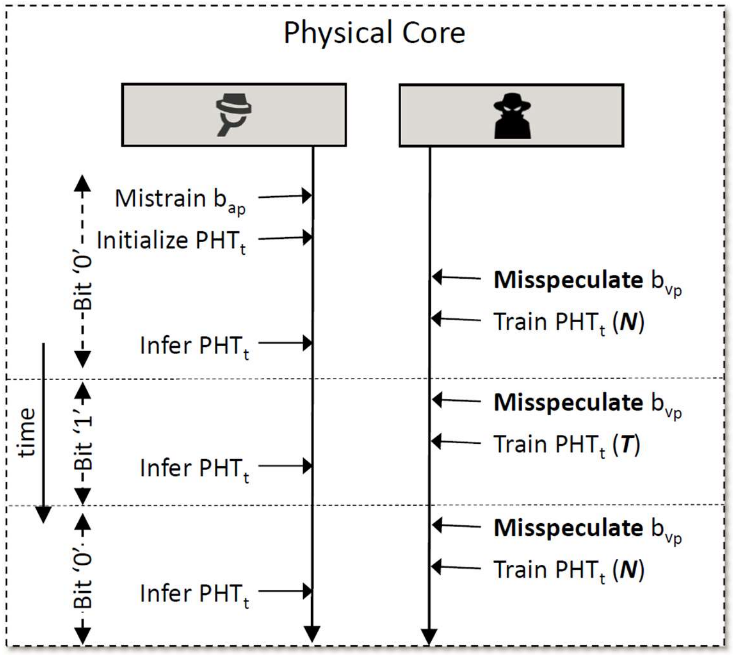


Figure 2: Illustration of BranchSpec covert channel protocol.

Potential Mitigations

- Existing system level defenses are ineffective
  - E.g., Retpoline, IBRS and others
- Potential architecture level mitigations
  - Restoring states for transient branches
  - Delaying PHT update
  - Enabling invisible PHT entry update

Conclusion

- Branches executed in speculation change PHT states, which are not restored after transient execution finishes.
- The vulnerability allows BPU to be used as *transmitting medium* in transient execution attacks.
- We demonstrate new forms of side and covert channels exploiting the discovered threat.
- We discuss potential mitigations to secure branch executions in speculative domain.

Thanks! Questions?

Md Hafizul Islam Chowdhuryy
Email: reyad@knights.ucf.edu

Source code available: https://github.com/fanyao/branchspec