Breaking and Fixing Speculative Load Hardening

Zhiyuan Zhang
University of Adelaide
Supervisors: Yuval Yarom and Chitchanok Chuengsatiansup

Outline

Zhiyuan Zhang, Gilles Barthe, Chitchanok Chuengsatiansup, Peter Schwabe, Yuval Yarom, “Breaking and Fixing Speculative Load Hardening”, eprint https://eprint.iacr.org/2022/715.

Outline

Zhiyuan Zhang, Gilles Barthe, Chitchanok Chuengsatiansup, Peter Schwabe, Yuval Yarom, “Breaking and Fixing Speculative Load Hardening”, eprint https://eprint.iacr.org/2022/715.

Attack

Mitigation

Semantic Proof

Outline

Zhiyuan Zhang, Gilles Barthe, Chitchanok Chuengsatiansup, Peter Schwabe, Yuval Yarom, “Breaking and Fixing Speculative Load Hardening”, eprint https://eprint.iacr.org/2022/715.

Attack

Mitigation

Semantic Proof

Outline

Zhiyuan Zhang, Gilles Barthe, Chitchanok Chuengsatiansup, Peter Schwabe, Yuval Yarom, “Breaking and Fixing Speculative Load Hardening”, eprint https://eprint.iacr.org/2022/715.

- Speculative Execution
- Spectre Attack and Speculative Load Hardening (SLH)
- Break SLH via control flow transfer and variant-time executions
- Fix SLH → Ultimate SLH and performance evaluation

Speculative Execution

```
if (index < arrayLen) {
  x = array[index];
  y = array2[x * 4096];
}
```

Speculative Execution

```
if (index < arrayLen) {
  x = array[index];
  y = array2[x * 4096];
}
```

Speculative Execution

```
if (index < arrayLen) {
  x = array[index];
  y = array2[x * 4096];
}
```

Resolve condition takes time

- Compute condition
- Fetch value from main memory

Speculative Execution

```
if (index < arrayLen) {
  x = array[index];
  y = array2[x * 4096];
}
```

Resolve condition takes time

- Compute condition
- Fetch value from main memory

Let’s predict the branch condition

Speculative Execution

```
if (index < arrayLen) {
   x = array[index];
   y = array2[x * 4096];
}
```

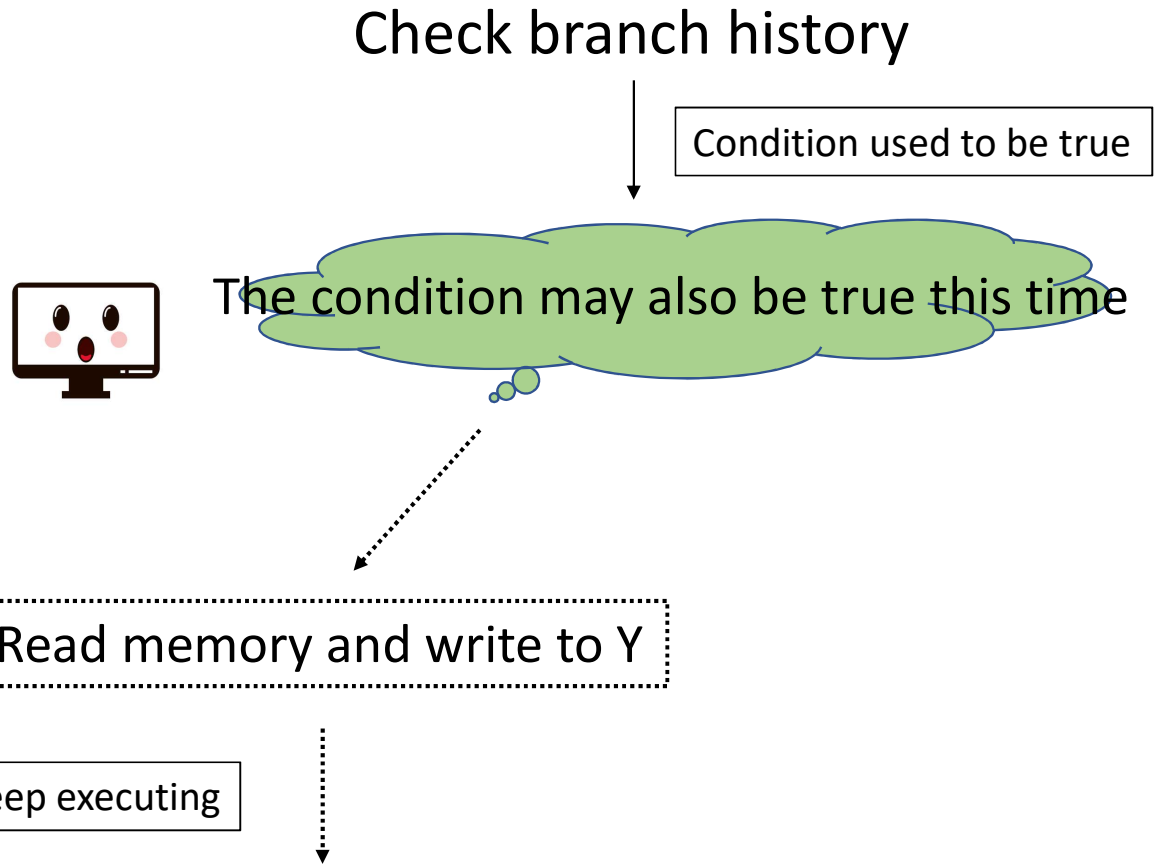
Check branch history

Condition used to be true

The condition may also be true this time

Speculative Execution

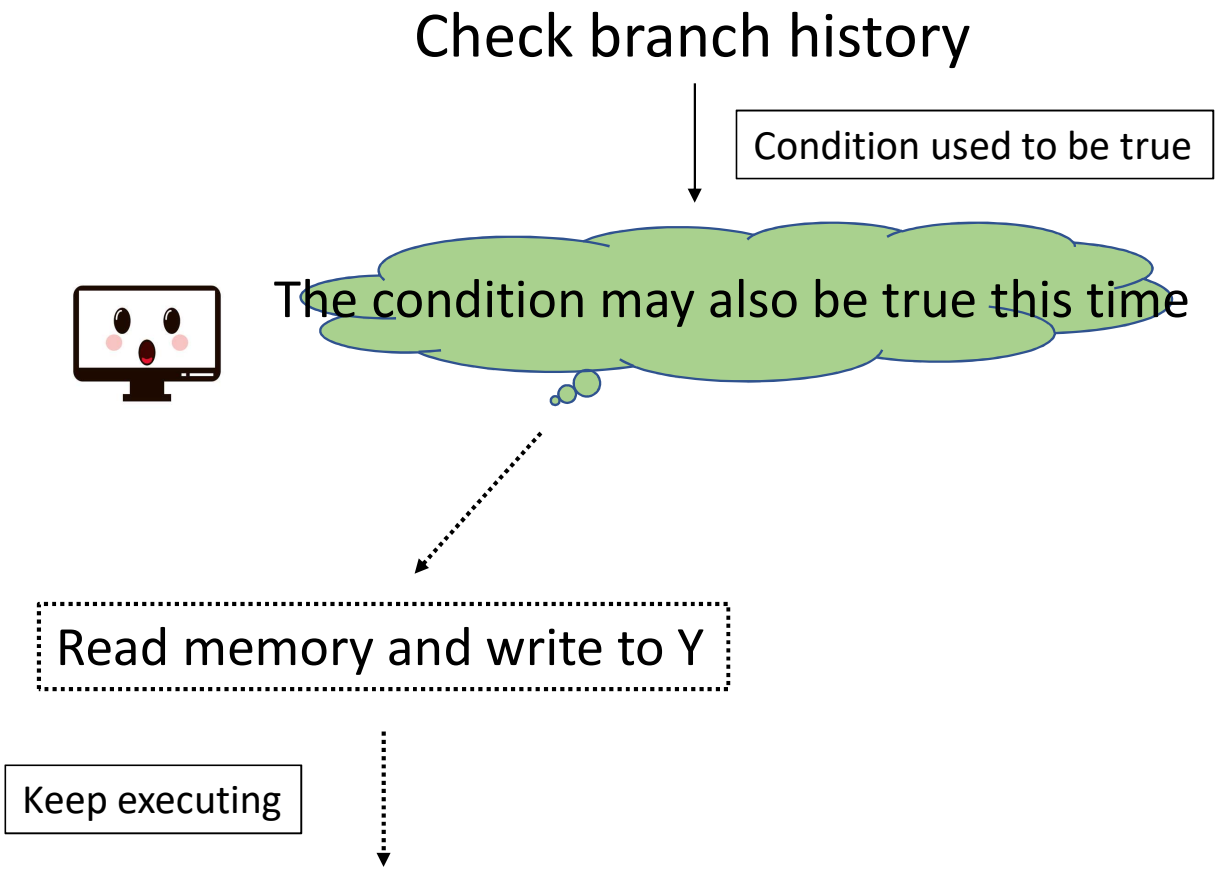
```
if (index < arrayLen) {
   x = array[index];
   y = array2[x * 4096];
}
```



Speculative Execution

```
if (index < arrayLen) {
   x = array[index];
   y = array2[x * 4096];
}
```

After a while, the condition is resolved to be true



Speculative Execution

```
if (index < arrayLen) {
  x = array[index];
  y = array2[x * 4096];
}
```

After a while, the condition is resolved to be true

Let's retire these executions

Check branch history

Condition used to be true

The condition may also be true this time

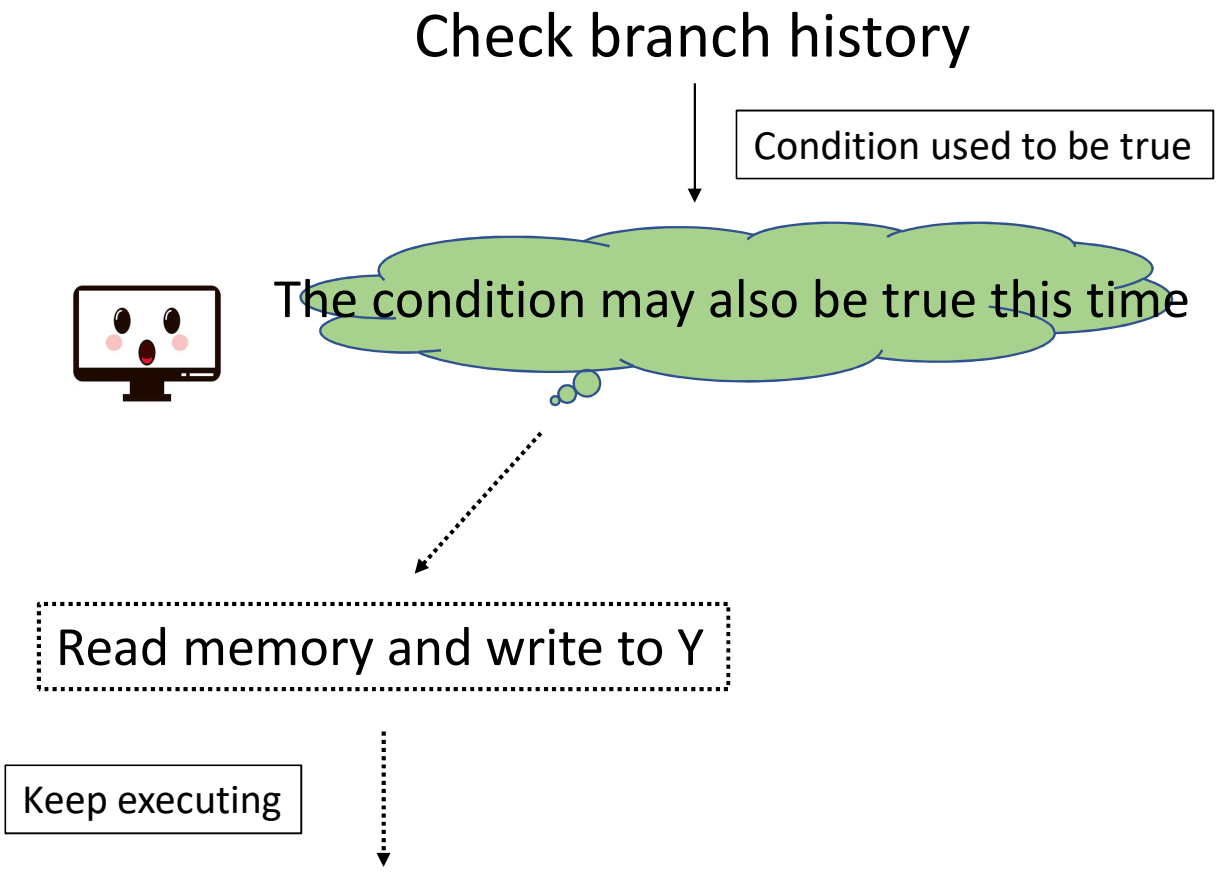
Read memory and write to Y

Keep executing

Speculative Execution

```
if (index < arrayLen) {
  x = array[index];
  y = array2[x * 4096];
}
```

After a while, the condition is resolved to be false



Speculative Execution

```
if (index < arrayLen) {
  x = array[index];
  y = array2[x * 4096];
}
```

After a while, the condition is resolved to be false

Let's squash these executions, start executing on the correct path

Check branch history

Condition used to be true

The condition may also be true this time

Exit

Speculative Execution

```
if (index < arrayLen) {
  x = array[index];
  y = array2[x * 4096];
}
```

After a while, the condition is resolved to be false

Let's squash these executions, start executing on the correct path

Check branch history

Condition used to be true

The condition may also be true this time

Exit

Don't want misprediction next time, update branch history

Spectre Attack

```
if (index < arrayLen) {
  x = array[index];
  y = array2[x * 4096];
}
```

Spectre Attack

```
if (index < arrayLen) {
  x = array[index];
  y = array2[x * 4096];
}
```

I want to leak entire virtual space memory.
It would be good if I can execute the code with an out-of-bound index

Spectre Attack

```
if (index < arrayLen) {
  x = array[index];
  y = array2[x * 4096];
}
```

I want to leak entire virtual space memory.
It would be good if I can execute the code with an out-of-bound index

The speculative execution is based on branch history.
I think I could poison the branch history…

Spectre Attack

```
if (index < arrayLen) {
  x = array[index];
  y = array2[x * 4096];
}
```

Keep feeding in-bound index

Spectre Attack

```
if (index < arrayLen) {
  x = array[index];
  y = array2[x * 4096];
}
```

Keep feeding in-bound index

Flush arrayLen

Spectre Attack

```
if (index < arrayLen) {
  x = array[index];
  y = array2[x * 4096];
}
```

Keep feeding in-bound index

Flush arrayLen

Flush array2

Spectre Attack

```
if (index < arrayLen) {
  x = array[index];
  y = array2[x * 4096];
}
```

Keep feeding in-bound index

Flush arrayLen

Flush array2

Feed an out-of-bound index

Spectre Attack

```
if (index < arrayLen) {
  x = array[index];
  y = array2[x * 4096];
}
```

Keep feeding in-bound index

Flush arrayLen

Flush array2

Feed an out-of-bound index

According to branch history, I predict index is in-bound

Spectre Attack

```
if (index < arrayLen) {
  x = array[index];
  y = array2[x * 4096];
}
```

Keep feeding in-bound index

Flush arrayLen

Flush array2

Feed an out-of-bound index

According to branch history, I predict index is in-bound

Read array[index] and access array2

Spectre Attack

```
if (index < arrayLen) {
  x = array[index];
  y = array2[x * 4096];
}
```

It's a misprediction. Not a big deal.
I rollback all executions.

Keep feeding in-bound index

Flush arrayLen

Flush array2

Feed an out-of-bound index

According to branch history, I predict index is in-bound

Read array[index] and access array2

Spectre Attack

```
if (index < arrayLen) {
  x = array[index];
  y = array2[x * 4096];
}
```

It's a misprediction. Not a big deal.
I rollback all executions.

Keep feeding in-bound index

Flush arrayLen

Flush array2

Feed an out-of-bound index

According to branch history, I predict index is in-bound

Read array[index] and access array2

The cache status has been changed

Spectre Attack

```
if (index < arrayLen) {
  x = array[index];
  y = array2[x * 4096];
}
```

It's a misprediction. Not a big deal.
I rollback all executions.

You leak architecture changes

- Cache status
- Execution port contention
- Power consumptions
- ……

Keep feeding in-bound index

Flush arrayLen

Flush array2

Feed an out-of-bound index

According to branch history, I predict index is in-bound

Read array[index] and access array2

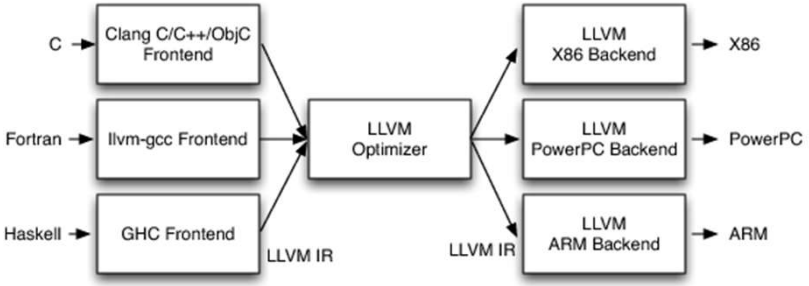
The cache status has been changed

Speculative Load Hardening (SLH)

Implemented in LLVM

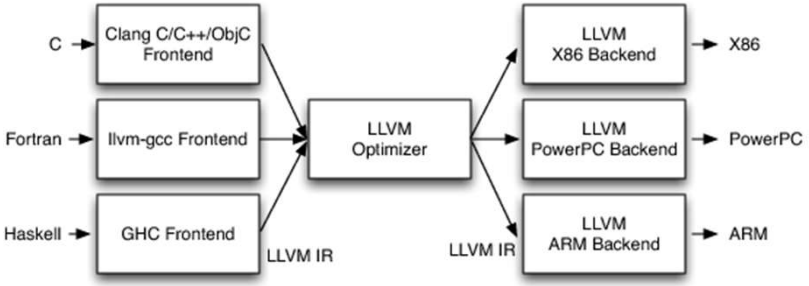
Speculative Load Hardening (SLH)

Implemented in LLVM



Speculative Load Hardening (SLH)

Implemented in LLVM



Track speculative state

Poison loaded value / address

Speculative Load Hardening (SLH)


```
if (index < arrayLen) {
  x = array[index];
  y = array2[x * 4096];
}
```

Track speculative state

Poison loaded value / address

Speculative Load Hardening (SLH)


```
mask = 0;
if (index < arrayLen) {
  x = array[index];
  y = array2[x * 4096];
}
```

Track speculative state

Poison loaded value / address

Speculative Load Hardening (SLH)


```
mask = 0;
if (index < arrayLen) {
  mask = index < arrayLen ? mask : -1;
  x = array[index];
  y = array2[x * 4096];
}
```

Track speculative state

Poison loaded value / address

Speculative Load Hardening (SLH)


```
mask = 0;
if (index < arrayLen) {
  mask = index < arrayLen ? mask : -1;
  x = array[index];
  y = array2[x * 4096];
}
```

Track speculative state

Conditional update cannot be speculated

Poison loaded value / address

Speculative Load Hardening (SLH)


```
mask = 0;
if (index < arrayLen) {
  mask = index < arrayLen ? mask : -1;
  x = array[index] | mask;
  y = array2[x * 4096];
}
```

Track speculative state

Conditional update cannot be speculated

Poison loaded value / address

Speculative Load Hardening (SLH)


```
mask = 0;
if (index < arrayLen) {
  mask = index < arrayLen ? mask : -1;
  x = array[index] | mask;
  y = array2[x * 4096];
}
```

Track speculative state

Poison loaded value / address

Conditional update cannot be speculated

Poison the value with -1 to avoid race condition between flushing the pipeline and loading memory

Limitation of SLH

- SLH hardens memory reading

Limitation of SLH

- SLH only hardens memory reading

Limitation of SLH

- SLH only hardens memory reading
- Leaks could from
  - Control flow transfer
  - Speculative store
  - Limited execution resources
    - Execution Ports
    - Reservation Station
    - ……

Limitation of SLH

- SLH only hardens memory reading
- Leaks could from
  - Control flow transfer
  - Speculative store
  - Limited execution resources
    - Execution Ports
    - Reservation Station
    - ……

Attacker Model

- We have interest in crypto code

Attacker Model

- We have interest in crypto code
- Crypto code in constant-time
  - No secret-relevant memory access
  - No control flow transfer based on secret
  - No variant-time executions

Attacker Model

- We have interest in crypto code
- Crypto code in constant-time
  - No secret-relevant memory access
  - No control flow transfer based on secret
  - No variant-time executions
- However, constant-time computing is not efficient

Attacker Model

- We have interest in crypto code
- Crypto code in constant time
  - No secret-relevant memory access
  - No control flow transfer based on secret
  - No variant-time executions
- However, constant-time computing is not efficient
- Only use constant-time computing on secret value

Attacker Model

- We have interest in crypto code
- Crypto code in constant time
  - No secret-relevant memory access
  - No control flow transfer based on secret
  - No variant-time executions
- However, constant-time computing is not efficient
- Only use constant-time computing on secret value

```
victim(int value, int isPublic) {
  if (isPublic) {
    //Leaky code
  }
}
```

Leak via control flow transfer

Leak via control flow transfer

```
  // Boundary Check
  if (isPublic) {
    if (value == 0) {
      a2 = a1 | a2;
      a3 = a2 | a3;
      ...
    } else {
      a1 = crc32(a1, a1);
      a2 = crc32(a2, a2);
      ...
    }
  }
}
```

Leak via control flow transfer

```
// Boundary Check
if (isPublic) {
  if (value == 0) {
    a2 = a1 | a2;
    a3 = a2 | a3;
    ...
  } else {
    a1 = crc32(a1, a1);
    a2 = crc32(a2, a2);
    ...
  }
}
}
```

Flush the outer branch

- Brings > 150 cycles speculation window

Leak via control flow transfer

```
// Boundary Check
if (isPublic) {
  if (value == 0) {
    a2 = a1 | a2;
    a3 = a2 | a3;
    ...
  } else {
    a1 = crc32(a1, a1);
    a2 = crc32(a2, a2);
    ...
  }
}
}
```

Flush the outer branch

- Brings > 150 cycles speculation window

The inner branch is resolved much faster

Leak via control flow transfer

```
// Boundary Check
if (isPublic) {
  if (value == 0) {
    a2 = a1 | a2;
    a3 = a2 | a3;
    ...
  } else {
    a1 = crc32(a1, a1);
    a2 = crc32(a2, a2);
    ...
  }
}
}
```

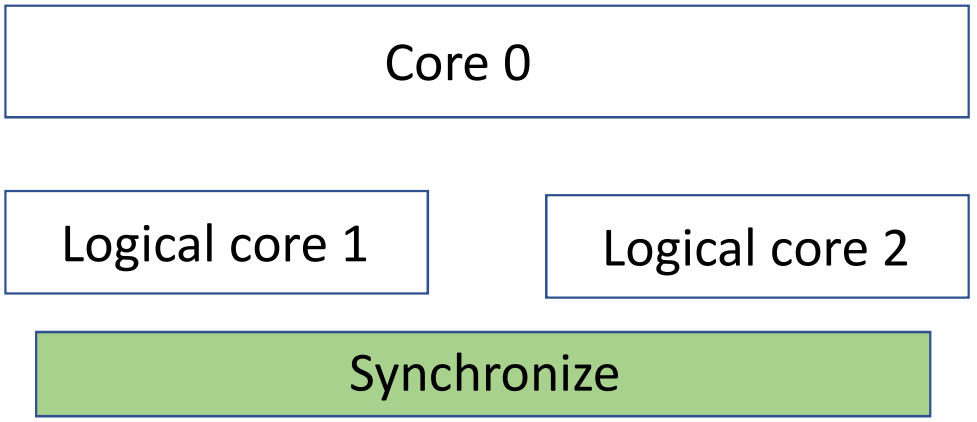
Flush the outer branch
- Brings > 150 cycles speculation window

The inner branch is resolved much faster

Instructions are sent to various execution ports
I measure which port is being used.

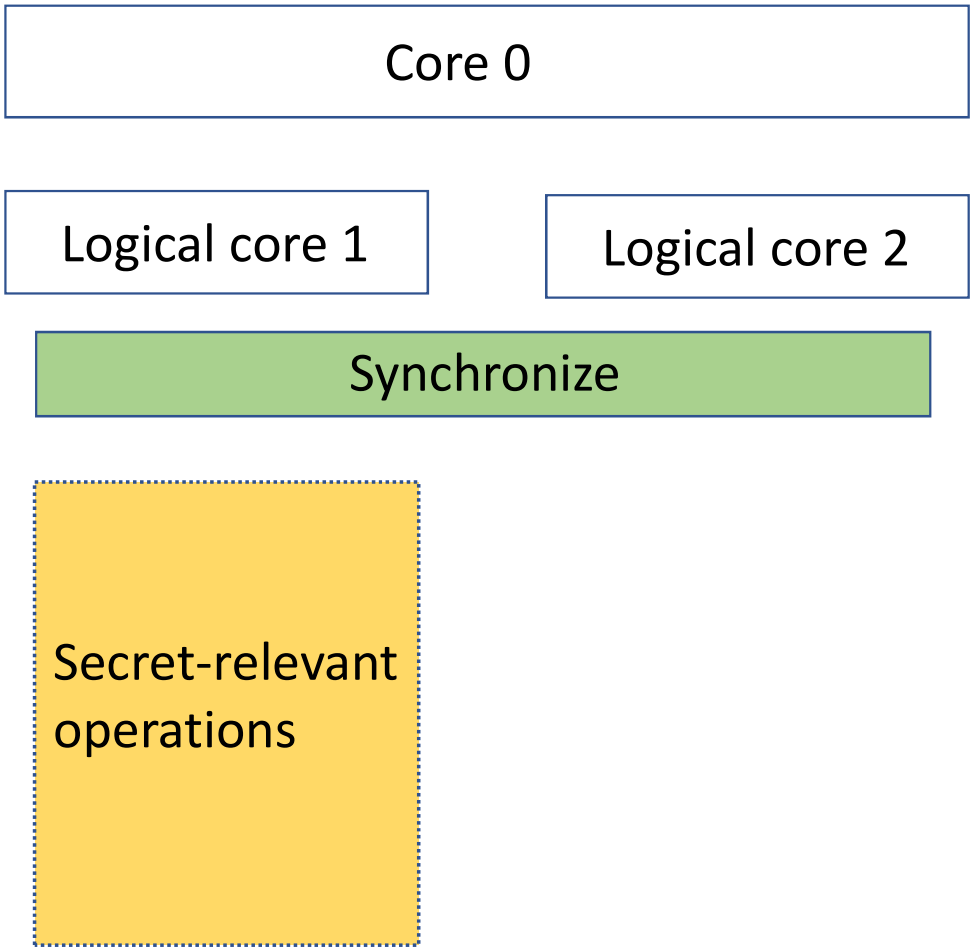
Port Contention

```
// Boundary Check
if (isPublic) {
  if (value == 0) {
    a2 = a1 | a2;
    a3 = a2 | a3;
    ...
  } else {
    a1 = crc32(a1, a1);
    a2 = crc32(a2, a2);
    ...
  }
}
```



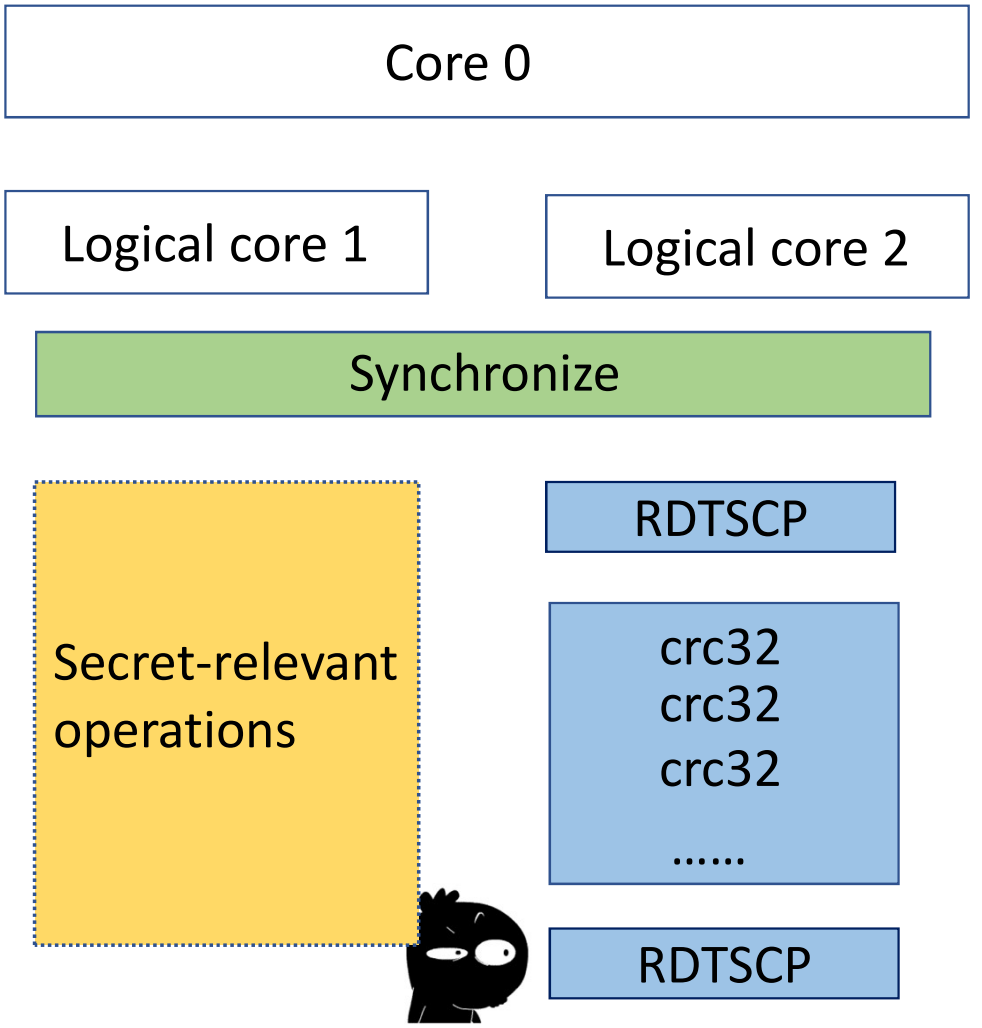
Port Contention

```
// Boundary Check
if (isPublic) {
  if (value == 0) {
    a2 = a1 | a2;
    a3 = a2 | a3;
    ...
  } else {
    a1 = crc32(a1, a1);
    a2 = crc32(a2, a2);
    ...
  }
}
```



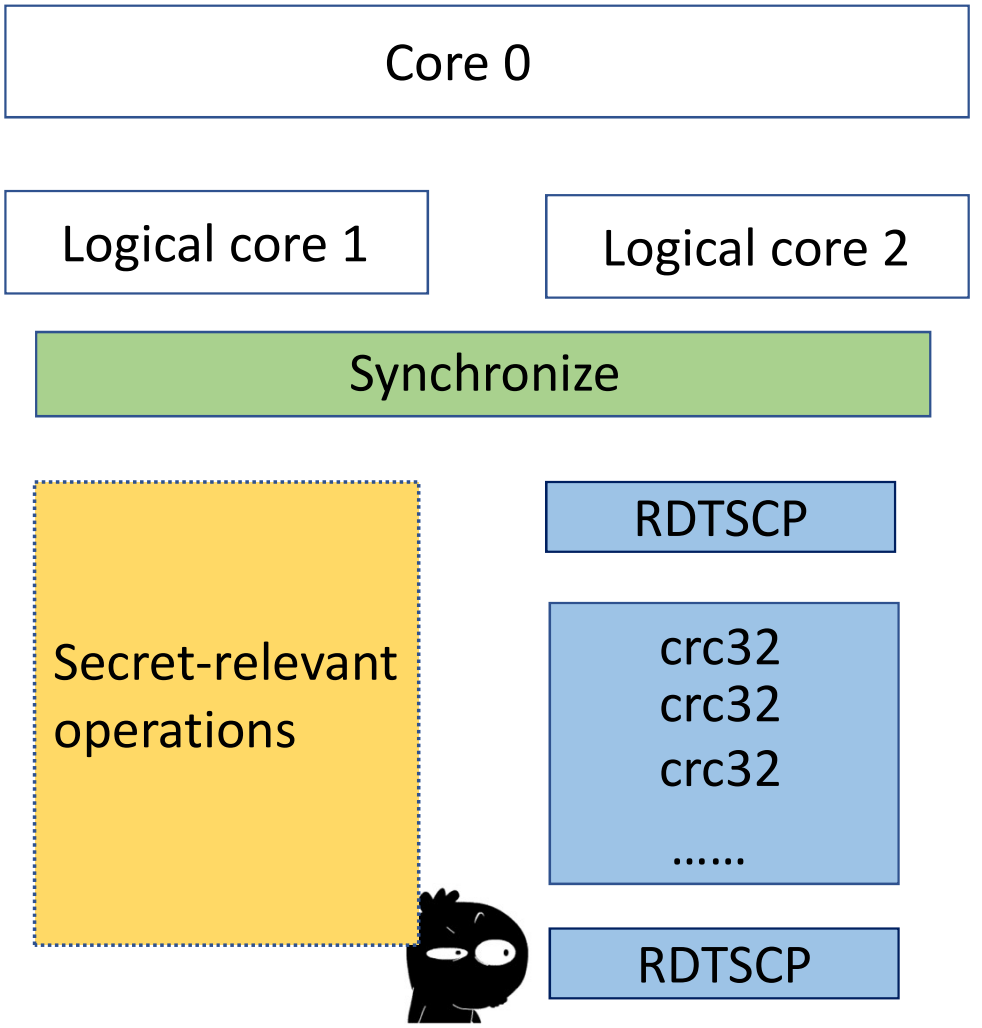
Port Contention

```
// Boundary Check
if (isPublic) {
  if (value == 0) {
    a2 = a1 | a2;
    a3 = a2 | a3;
    ...
  } else {
    a1 = crc32(a1, a1);
    a2 = crc32(a2, a2);
    ...
  }
}
```



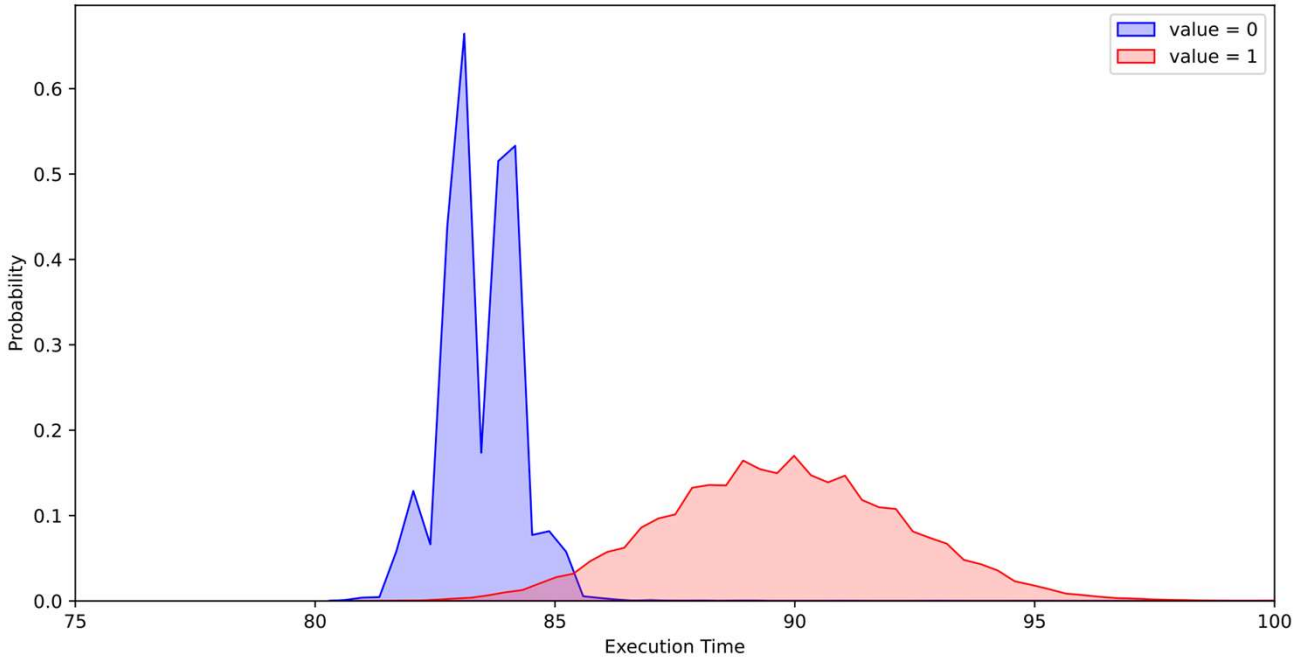
Port Contention

```
// Boundary Check
if (isPublic) {
  if (value == 0) {
    a2 = a1 | a2;
    a3 = a2 | a3;
    ...
  } else {
    a1 = crc32(a1, a1);
    a2 = crc32(a2, a2);
    ...
  }
}
```



Port Contention

```
// Boundary Check
if (isPublic) {
  if (value == 0) {
    a2 = a1 | a2;
    a3 = a2 | a3;
    ...
  } else {
    a1 = crc32(a1, a1);
    a2 = crc32(a2, a2);
    ...
  }
}
```



Fix SLH

```
// Boundary Check
if (isPublic) {
  if (value == 0) {
    a2 = a1 | a2;
    a3 = a2 | a3;
    ...
  } else {
    a1 = crc32(a1, a1);
    a2 = crc32(a2, a2);
    ...
  }
}
```

Fix SLH

```
// Boundary Check
mask = 0;
if (isPublic) {
  mask = isPublic ? mask : -1;
  if (value == 0) {
    mask = value == 0 ? mask : -1;
    a2 = a1 | a2;
    a3 = a2 | a3;
    ...
  } else {
    a1 = crc32(a1, a1);
    a2 = crc32(a2, a2);
    ...
  }
}
```

Compiled by SLH

Fix SLH

```
// Boundary Check
mask = 0;
isPublic |= mask;
if (isPublic) {
  mask = isPublic ? mask : -1;
  value |= mask;
  if (value == 0) {
    mask = value == 0 ? mask : -1;
    a2 = a1 | a2;
    a3 = a2 | a3;
    ...
  } else {
    a1 = crc32(a1, a1);
    a2 = crc32(a2, a2);
    ...
  }
}
```

Compiled by USLH

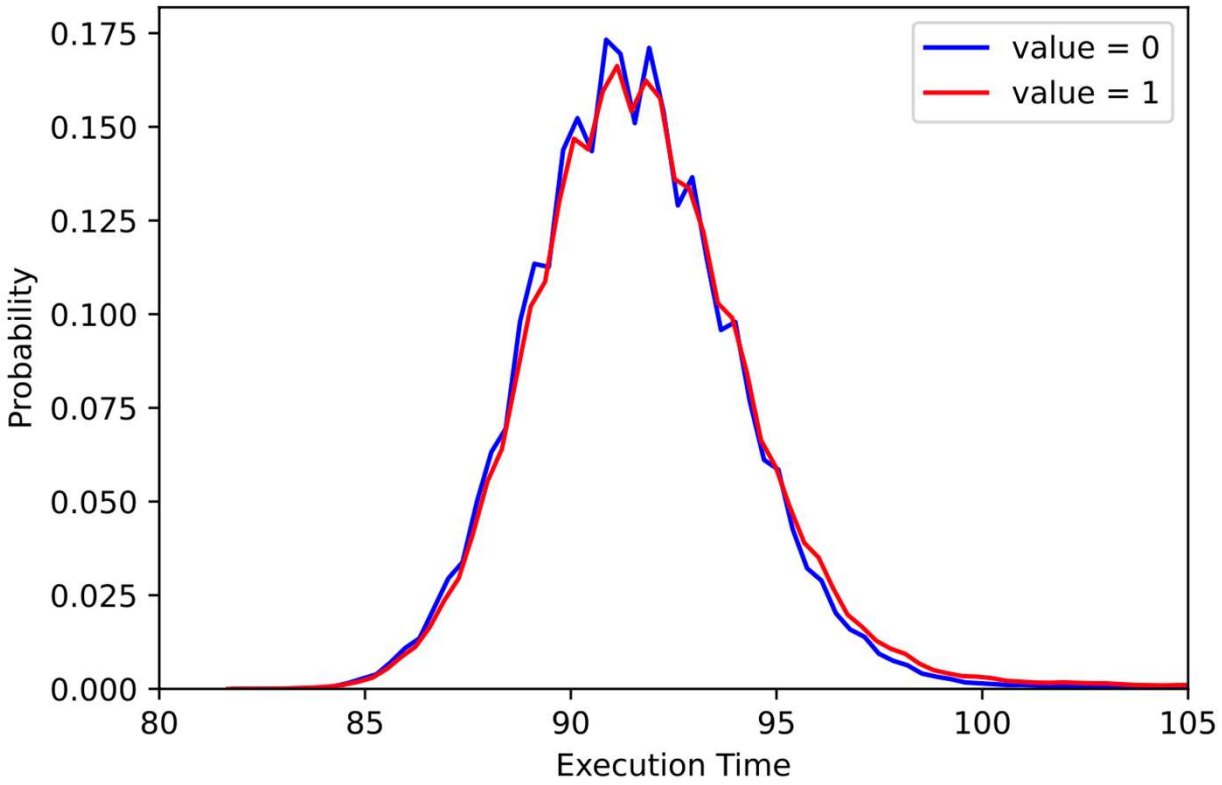
Fix SLH

```
// Boundary Check
mask = 0;
isPublic |= mask;
if (isPublic) {
  mask = isPublic ? mask : -1;
  value |= mask;
  if (value == 0) {
    mask = value == 0 ? mask : -1;
    a2 = a1 | a2;
    a3 = a2 | a3;
    ...
  } else {
    a1 = crc32(a1, a1);
    a2 = crc32(a2, a2);
    ...
  }
}
```

Compiled by USLH

Mitigation Result

```
// Boundary Check
mask = 0;
isPublic |= mask;
if (isPublic) {
  mask = isPublic ? mask : -1;
  value |= mask;
  if (value == 0) {
    mask = value == 0 ? mask : -1;
    a2 = a1 | a2;
    a3 = a2 | a3;
    ...
  } else {
    a1 = crc32(a1, a1);
    a2 = crc32(a2, a2);
    ...
  }
}
```



Variant-time Instructions

Variant-time Instructions

- DIV Instruction

Variant-time Instructions

- DIV Instruction

- REPEAT instruction (REP MOV RAX, RBX)
  - The number of iteration depends on ECX

Variant-time Instructions

- DIV Instruction
- REPEAT instruction (REP MOV RAX, RBX)
  - The number of iteration depends on ECX
- Floating point instructions

Variant-time Instructions

- DIV Instruction
- REPEAT instruction (REP MOV RAX, RBX)
  - The number of iteration depends on ECX
- Floating point instructions

Attacking variant-time instructions

```
value = sqrtsd(value);
value = mulsd(value, value);
```

Attacking variant-time instructions

```
value = sqrtsd(value);
value = mulsd(value, value);
```

Attacking variant-time instructions

```
value = sqrtsd(value);
value = mulsd(value, value);
```

On i7-10710U:

Executing a pair of SQRTSD and MULSD:

- 65536: 5 cycles
- 2.34e-308: 7 cycles

Attacking variant-time instructions

```
value = sqrtsd(value);
value = mulsd(value, value);
```

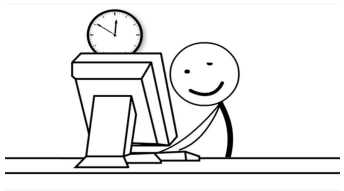
On i7-10710U:

Executing a pair of SQRTSD and MULSD:

- 65536: 5 cycles
- 2.34e-308: 7 cycles

Do constant-time computing

Variant-time Instructions



Okay, constant-time computing is slow. I use non-constant-time computing for public data

Variant-time Instructions

```
// Boundary Check
if (isPublic) {
  value = sqrtsd(value);
  value = mulsd(value, value);
}
```

Okay, constant-time computing is slow. I use non-constant-time computing for public data

Variant-time Instructions

```
// Boundary Check
if (isPublic) {
  value = sqrtsd(value);
  value = mulsd(value, value);
}
```

Okay, constant-time computing is slow. I use non-constant-time computing for public data

The code can be speculatively executed. Measure the execution time.

Variant-time Instructions

```
// Boundary Check
if (isPublic) {
  value = sqrtsd(value);
  value = mulsd(value, value);
}
```

I wait until the branch is resolved

Okay, constant-time computing is slow. I use non-constant-time computing for public data

The code can be speculatively executed. Measure the execution time.

Variant-time Instructions

```
// Boundary Check
if (isPublic) {
  value = sqrtsd(value);
  value = mulsd(value, value);
}
```

I wait until the branch is resolved

Okay, constant-time computing is slow. I use non-constant-time computing for public data

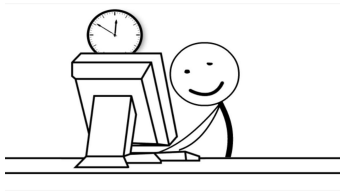
The code can be speculatively executed. Measure the execution time.

Constant-time under speculative execution. No spectre attacks!

Variant-time Instructions

```
// Boundary Check
if (isPublic) {
  value = sqrtsd(value);
  value = mulsd(value, value);
}
```

I wait until the branch is resolved



Okay, constant-time computing is slow. I use non-constant-time computing for public data

The code can be speculatively executed. Measure the execution time.

Constant-time under speculative execution. No spectre attacks!

I doubt it.

Review pipeline stages

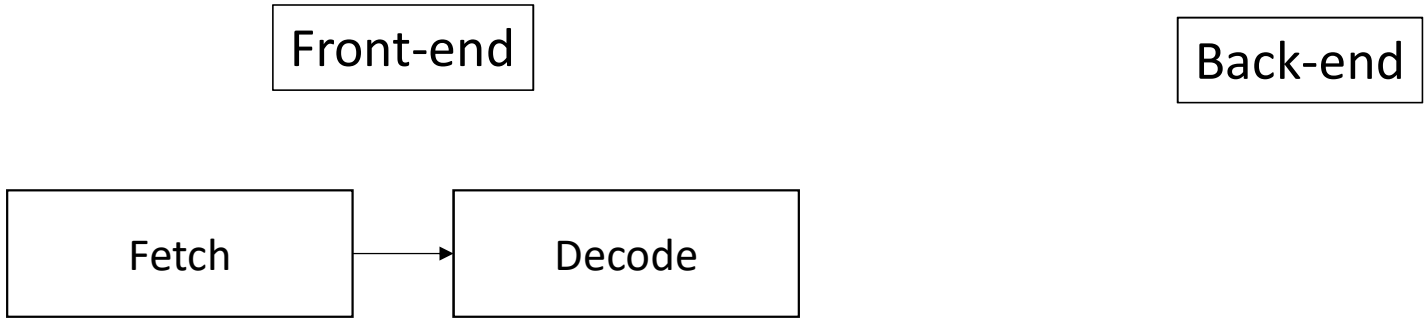
How is an instruction handled by the processer

Front-end

Back-end

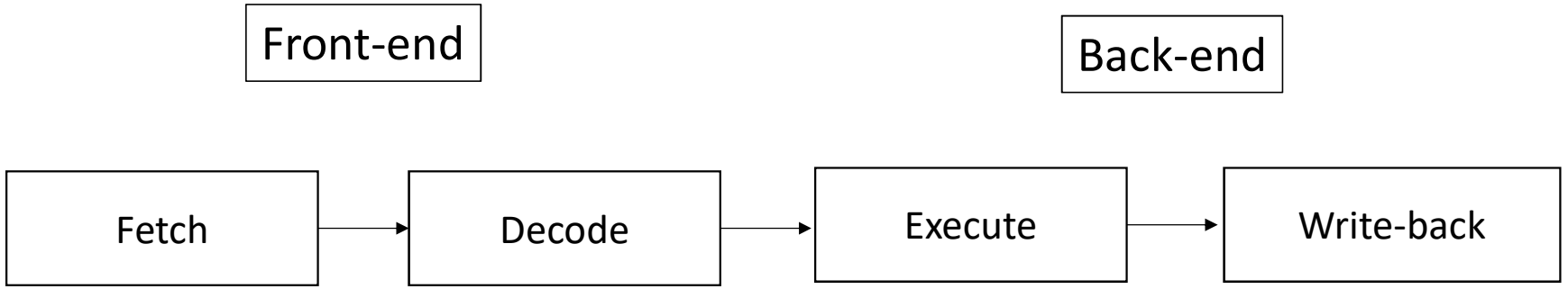
Review pipeline stages

How is an instruction handled by the processer

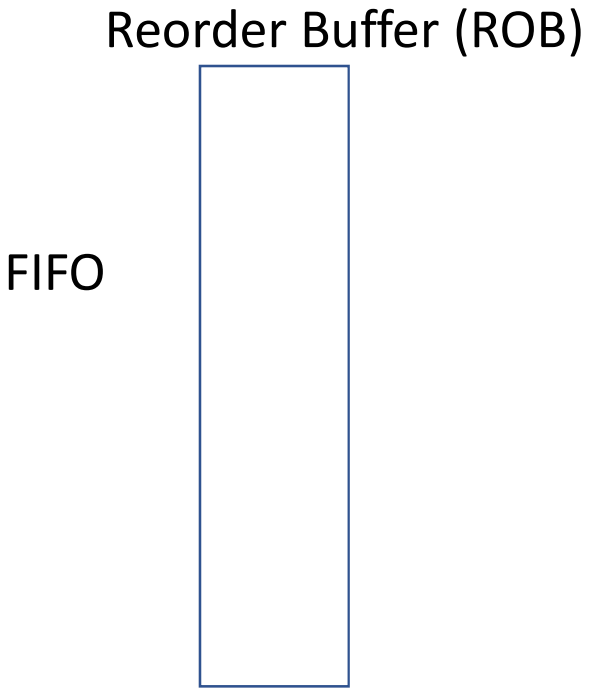


Review pipeline stages

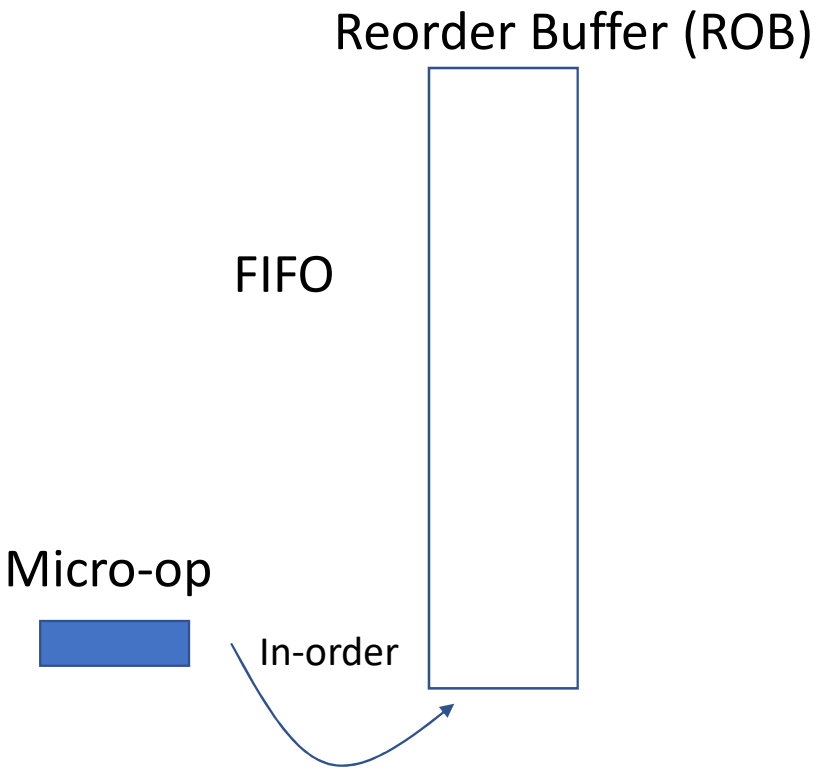
How is an instruction handled by the processer



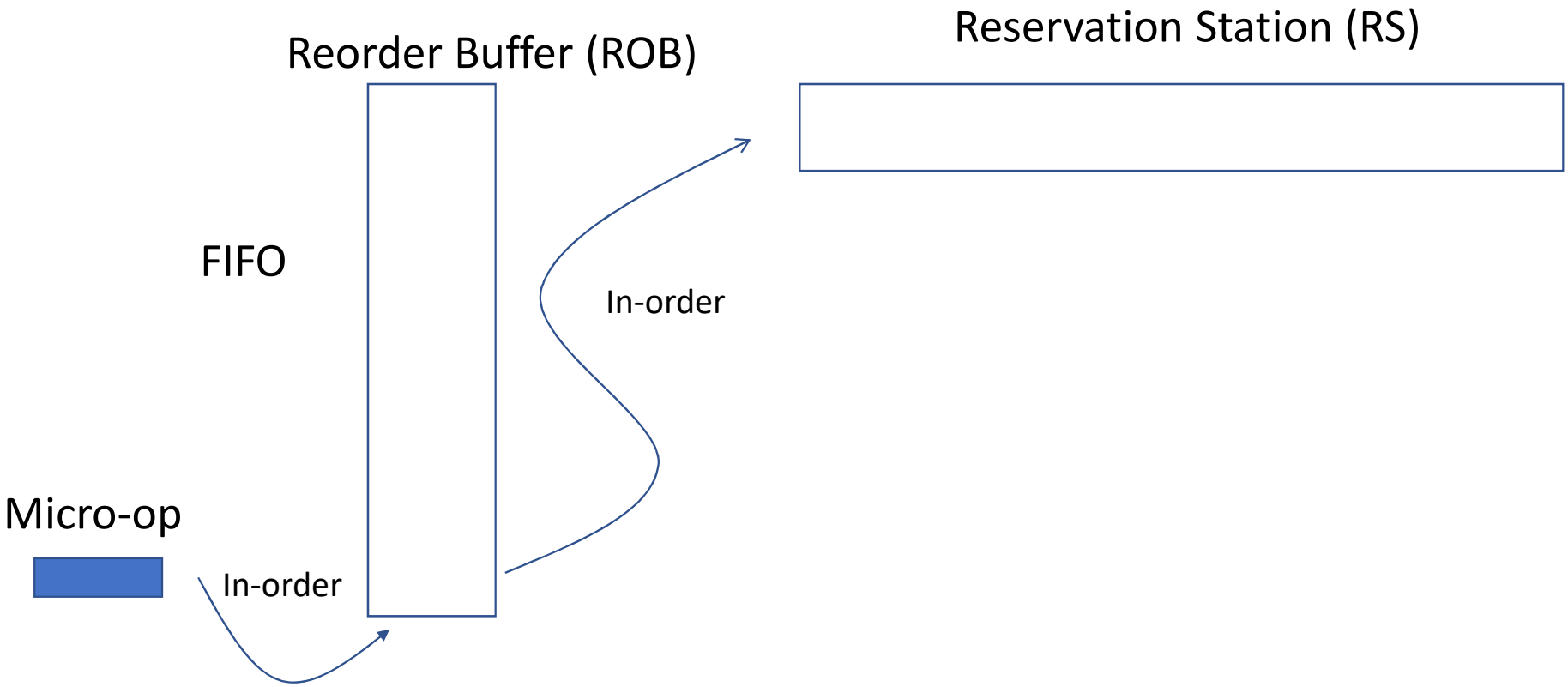
Execution Engine



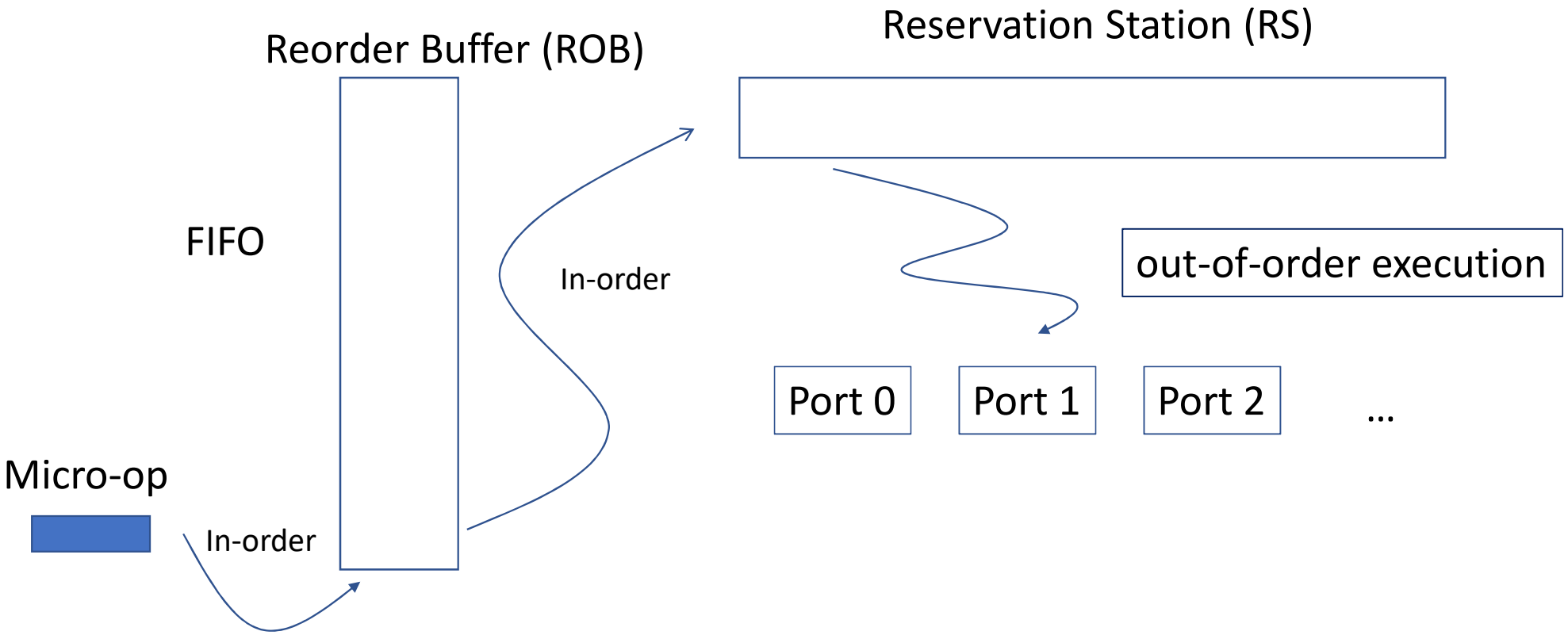
Execution Engine



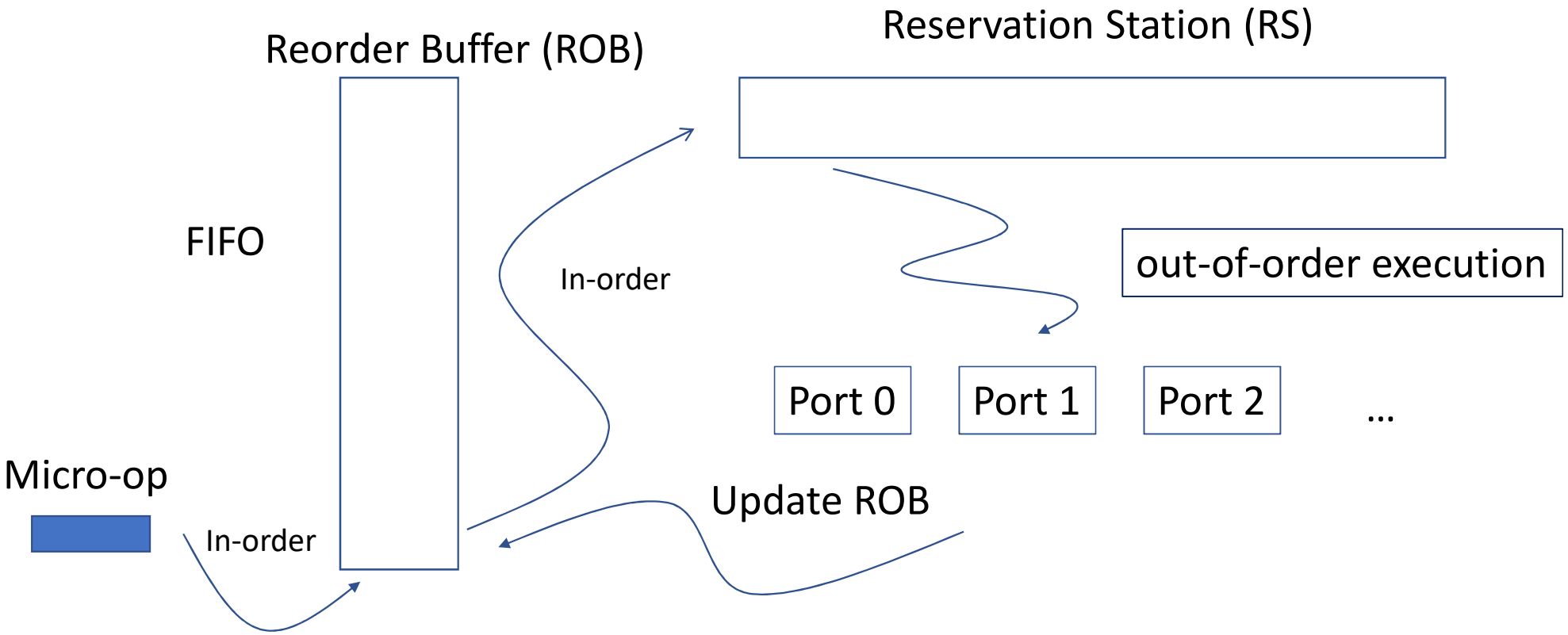
Execution Engine



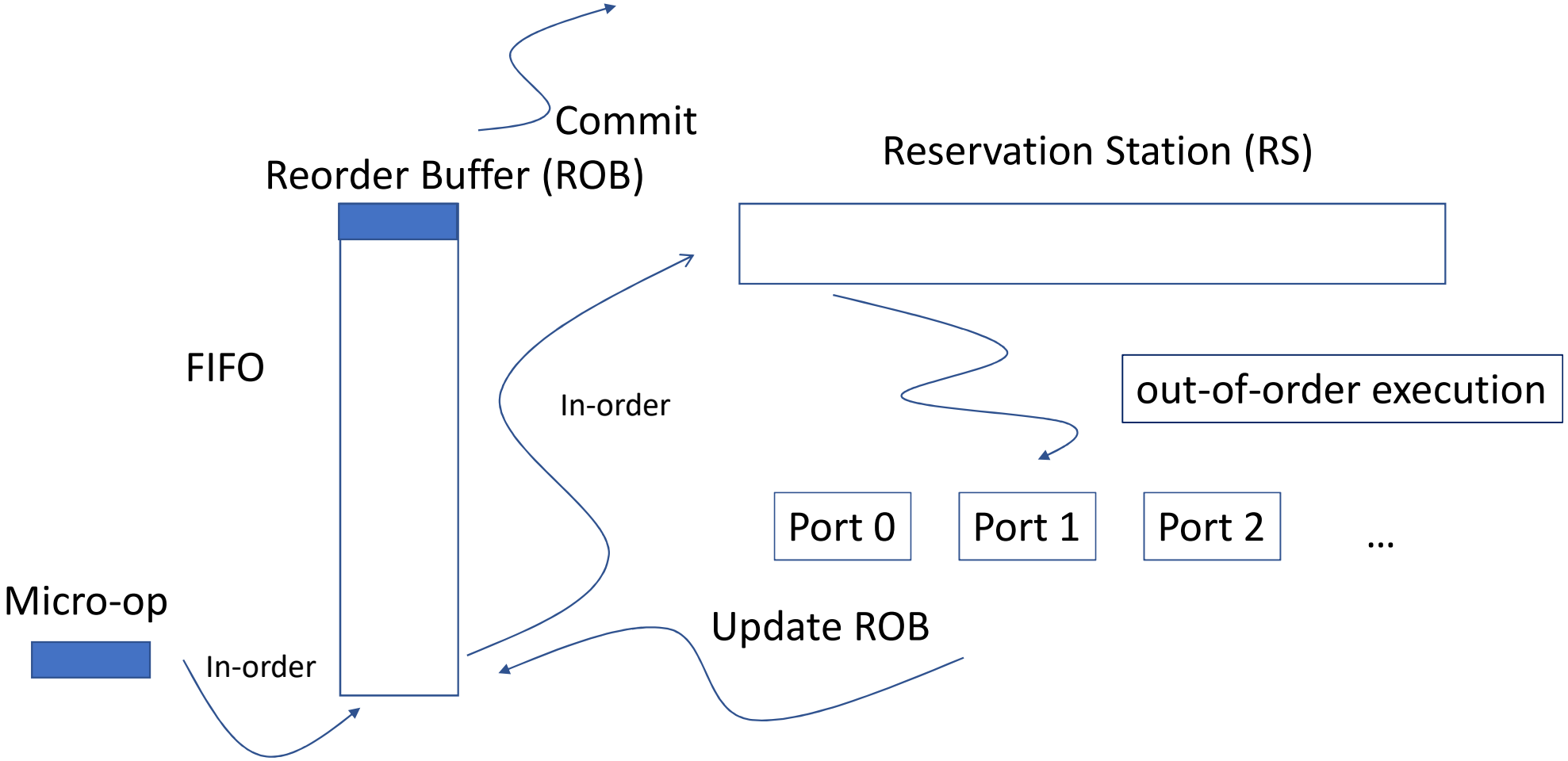
Execution Engine



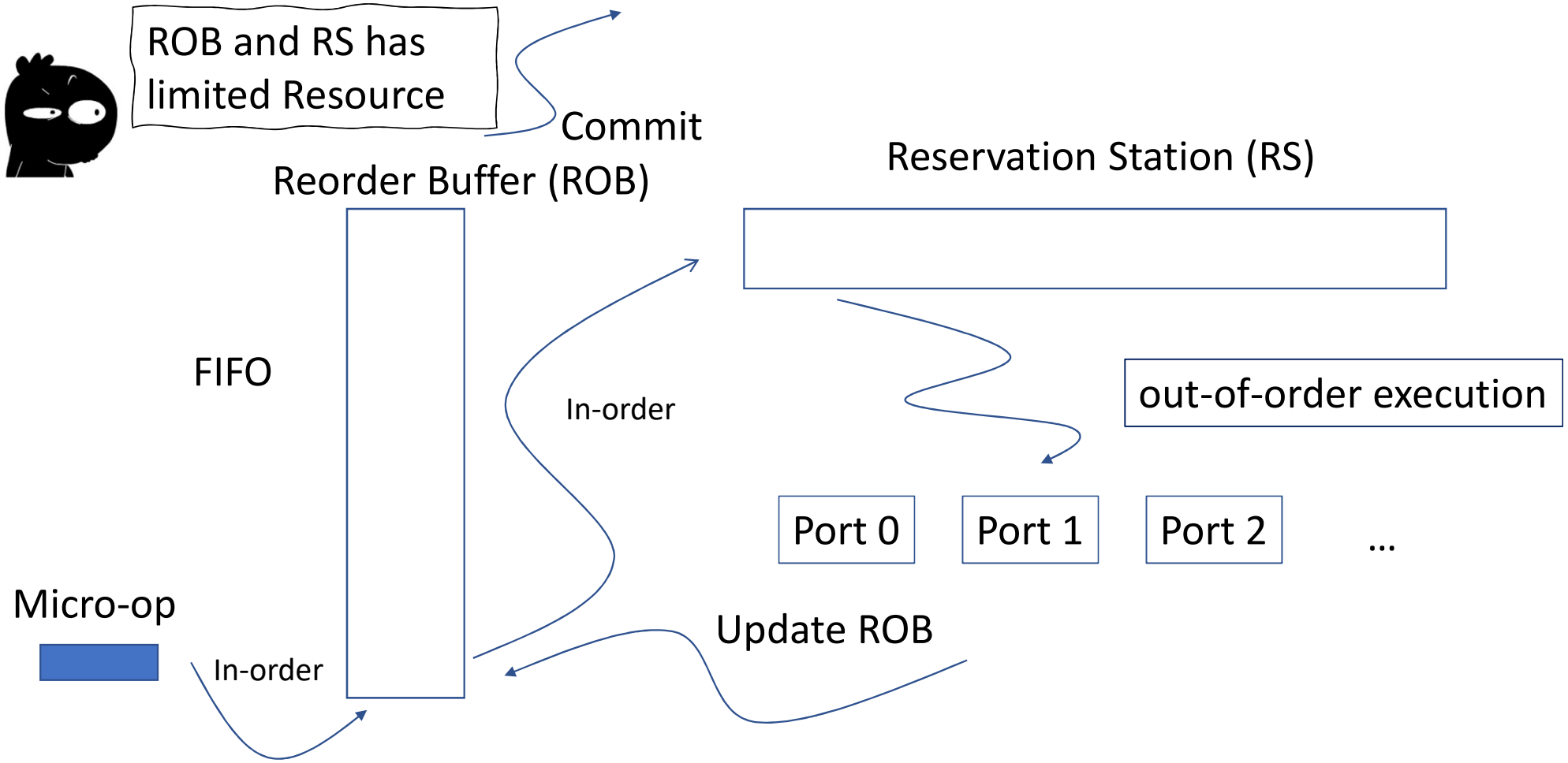
Execution Engine



Execution Engine

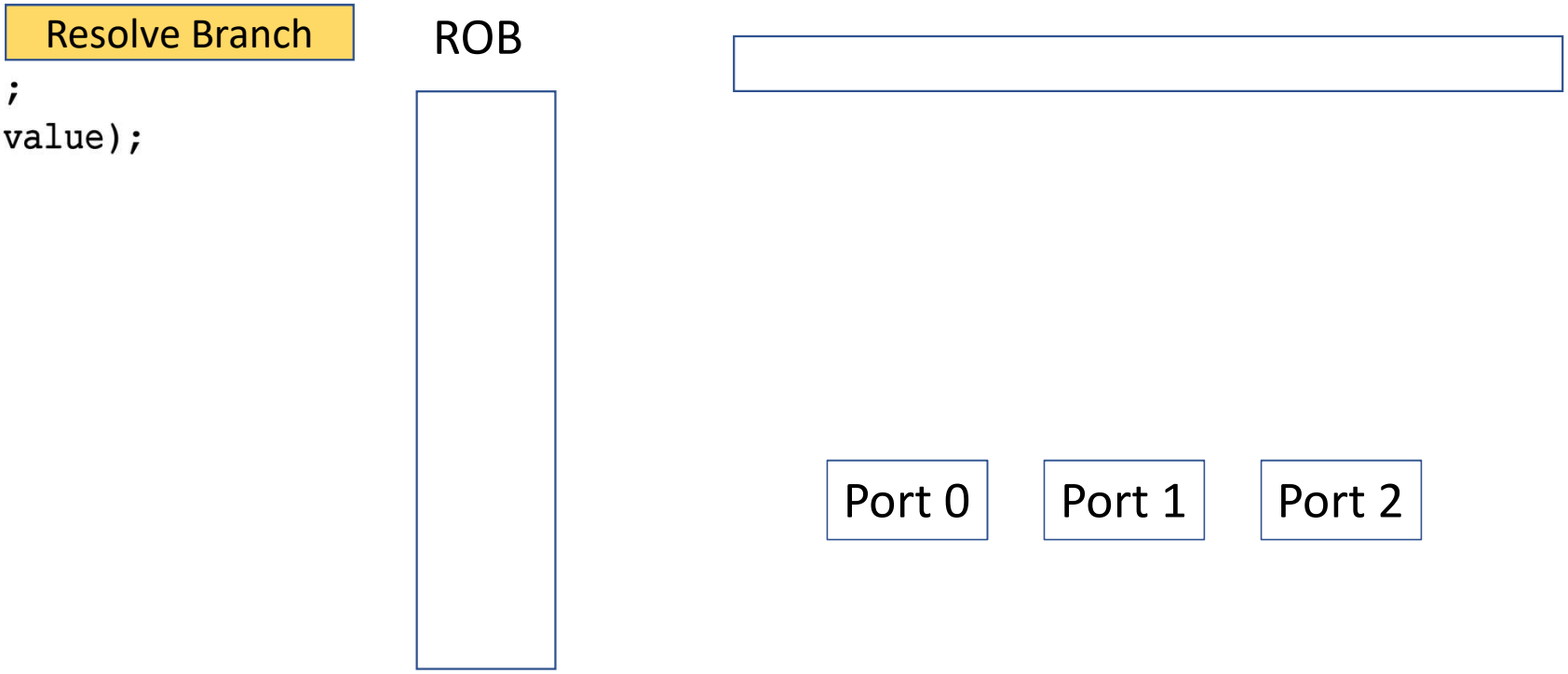


Execution Engine



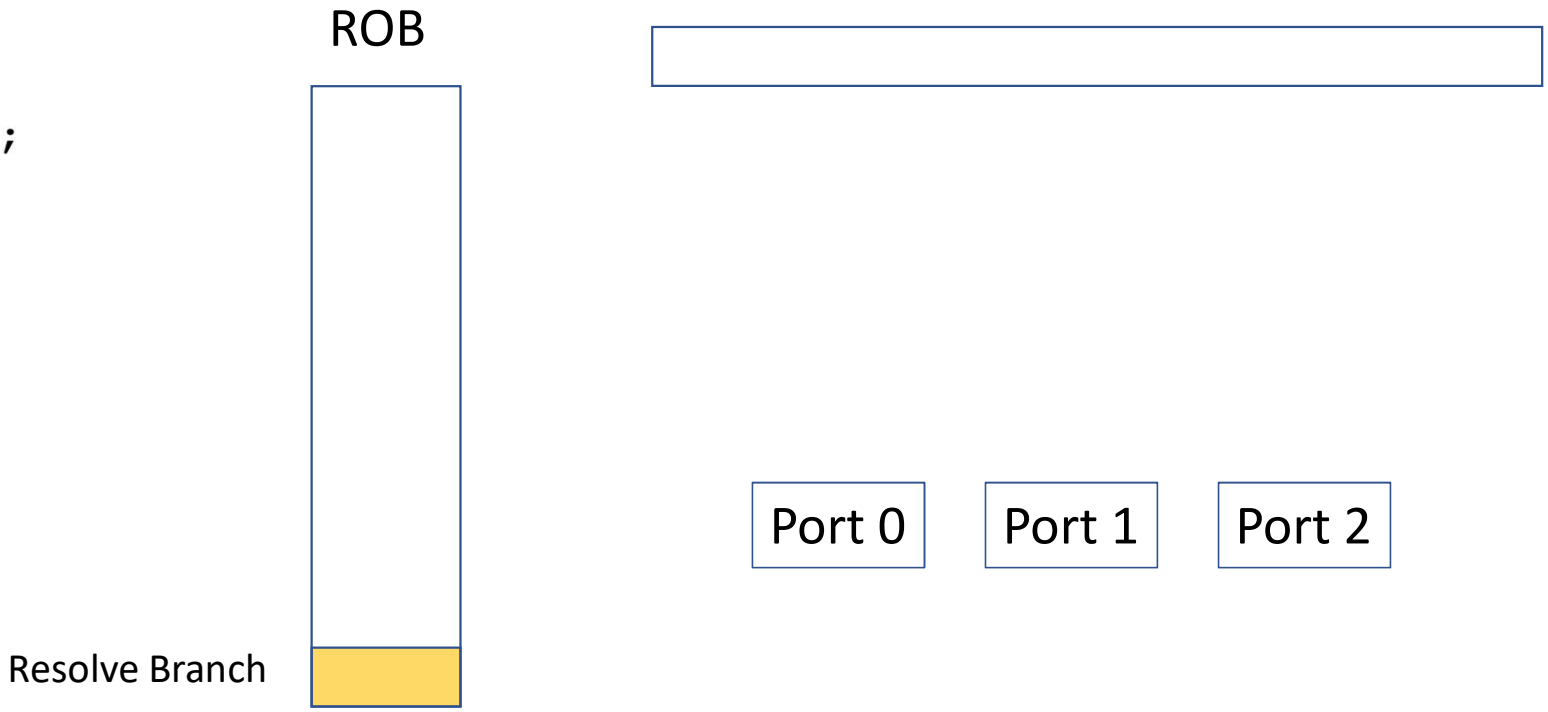
Variant-time execution under speculation

```
// Boundary Check
if (isPublic) {
  value = sqrtsd(value);
  value = mulsd(value, value);
}
```



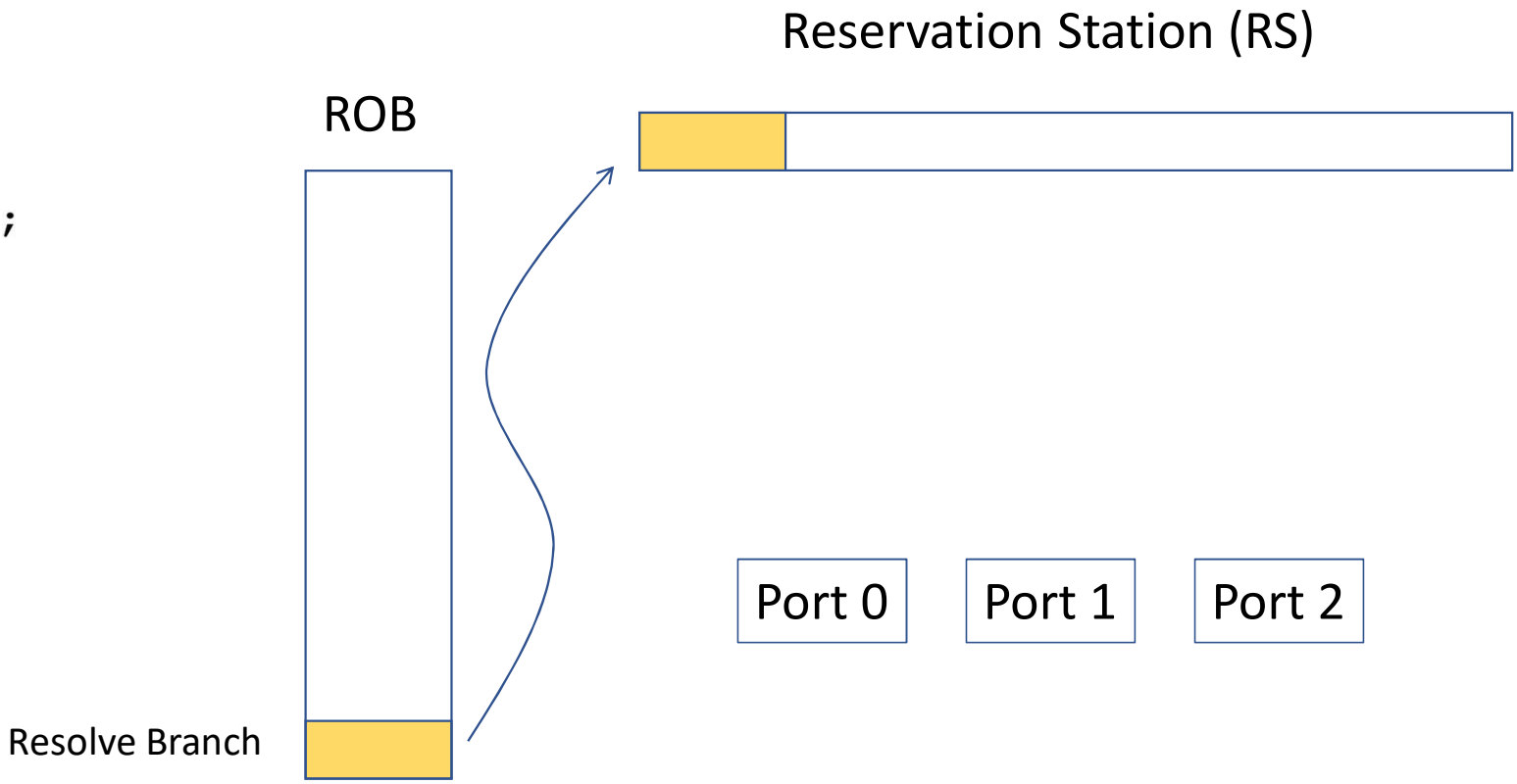
Variant-time execution under speculation

```
// Boundary Check
if (isPublic) {
  value = sqrtsd(value);
  value = mulsd(value, value);
}
```



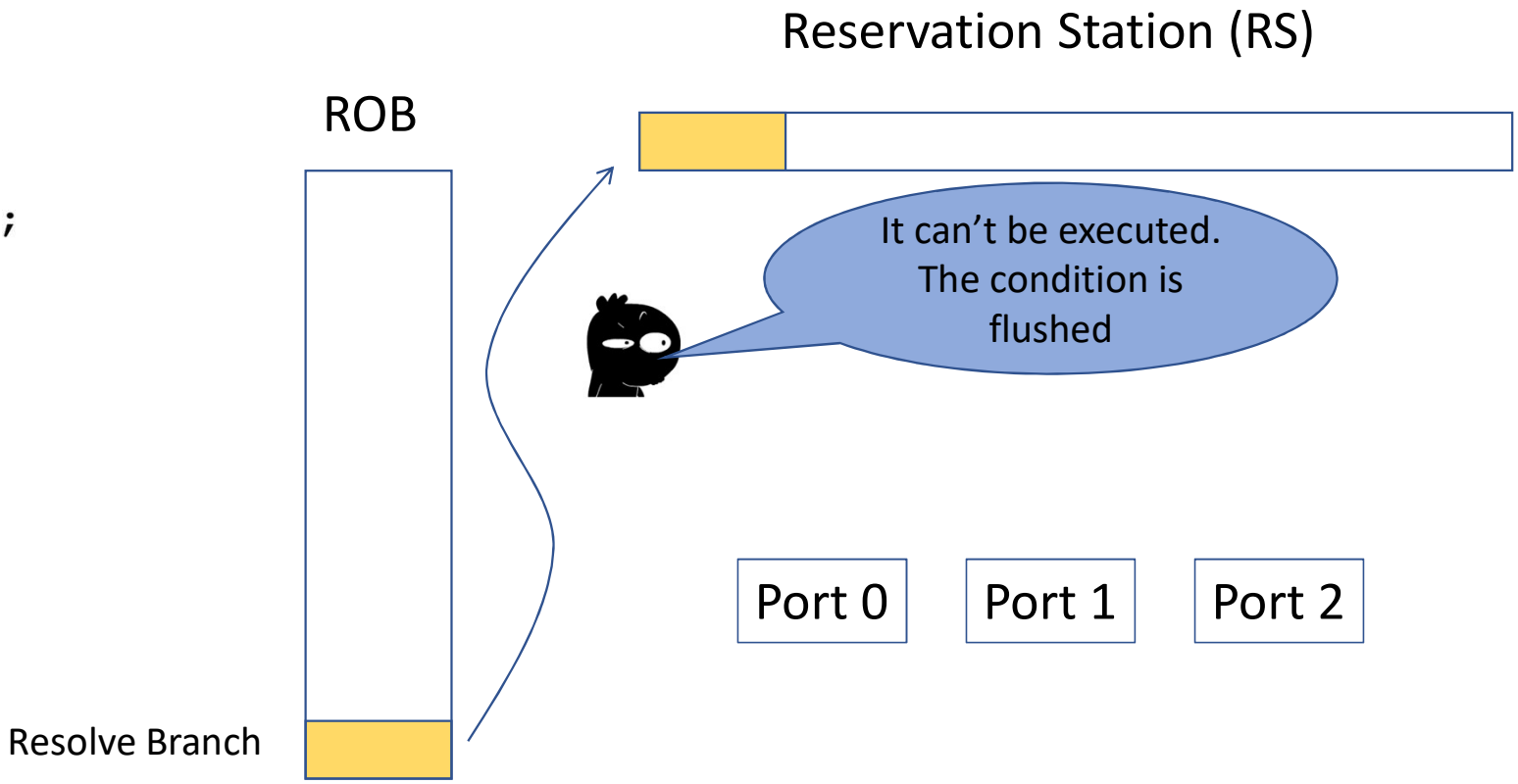
Variant-time execution under speculation

```
// Boundary Check
if (isPublic) {
  value = sqrtsd(value);
  value = mulsd(value, value);
}
```



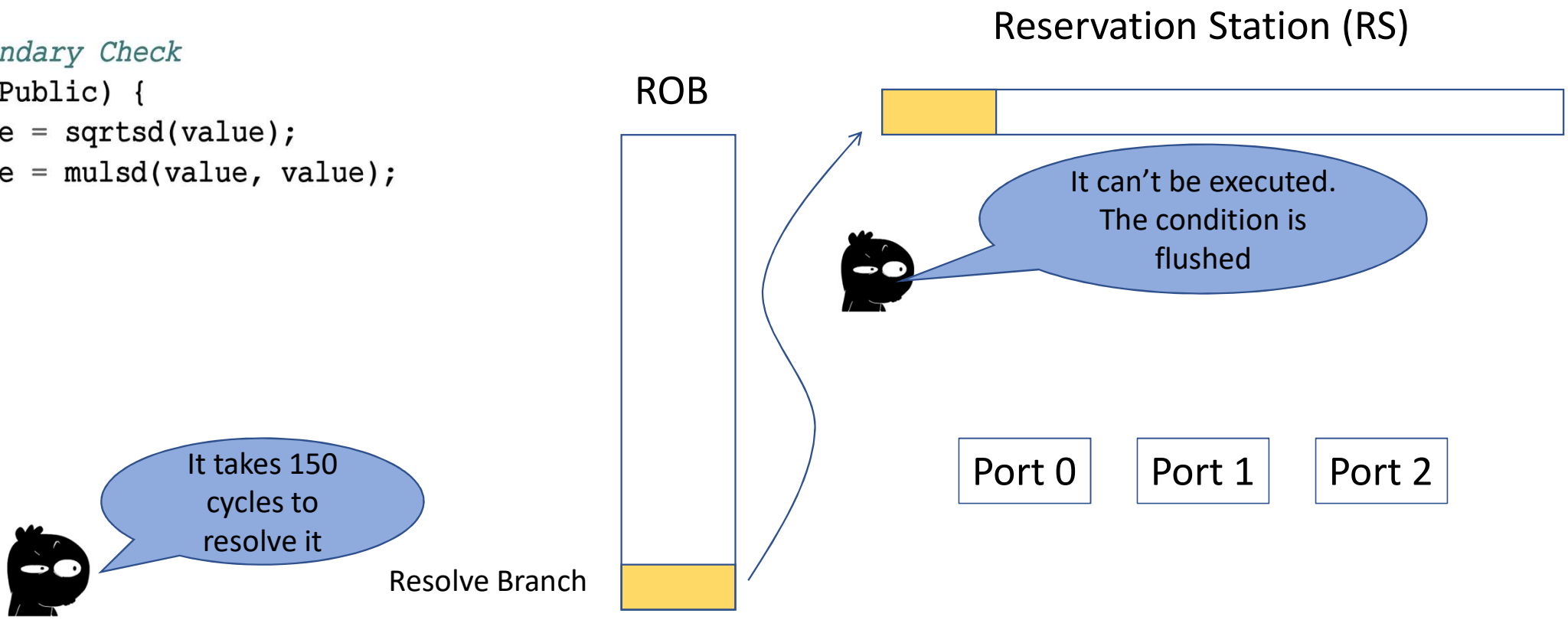
Variant-time execution under speculation

```
// Boundary Check
if (isPublic) {
  value = sqrtsd(value);
  value = mulsd(value, value);
}
```



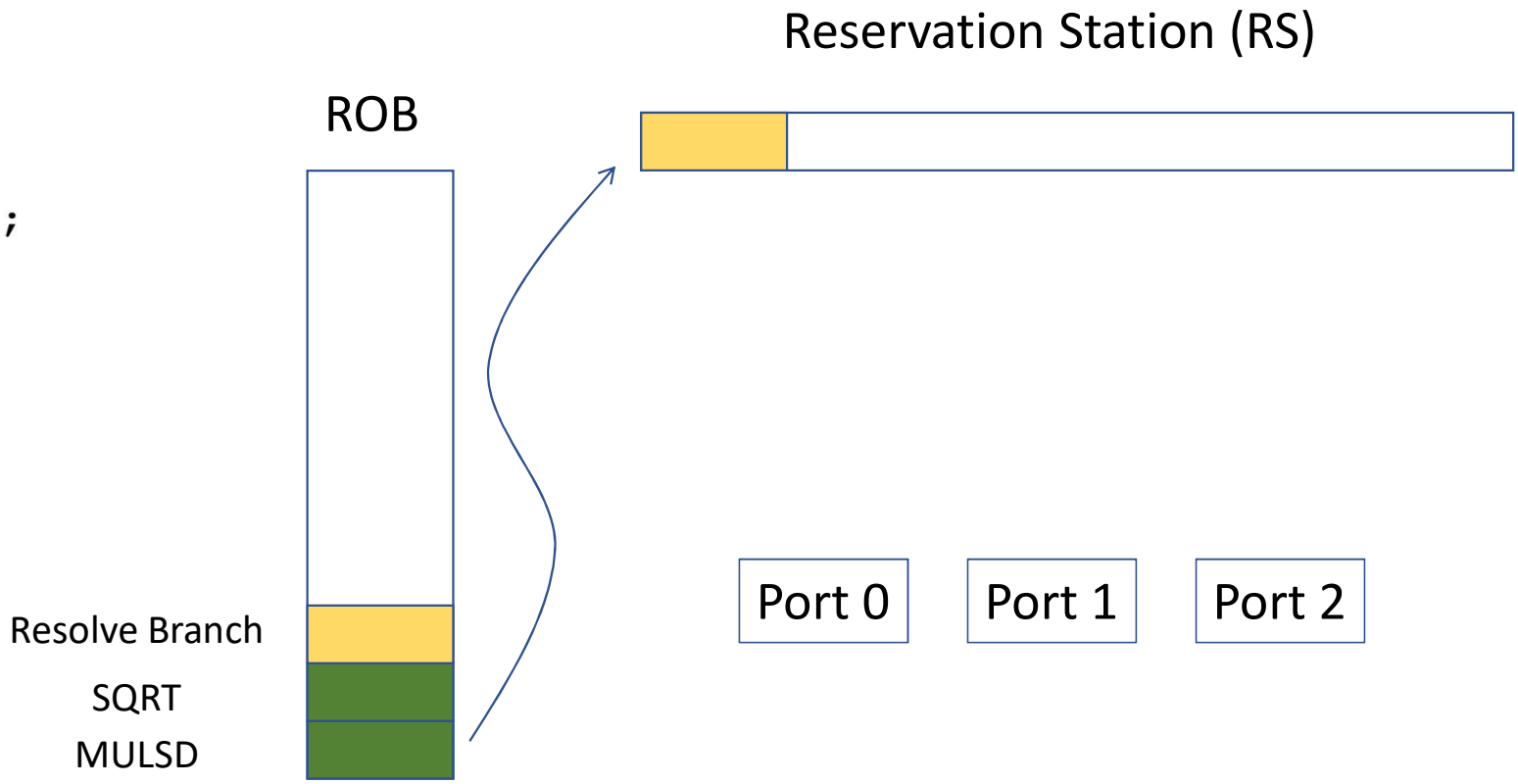
Variant-time execution under speculation

```
// Boundary Check
if (isPublic) {
  value = sqrtsd(value);
  value = mulsd(value, value);
}
```



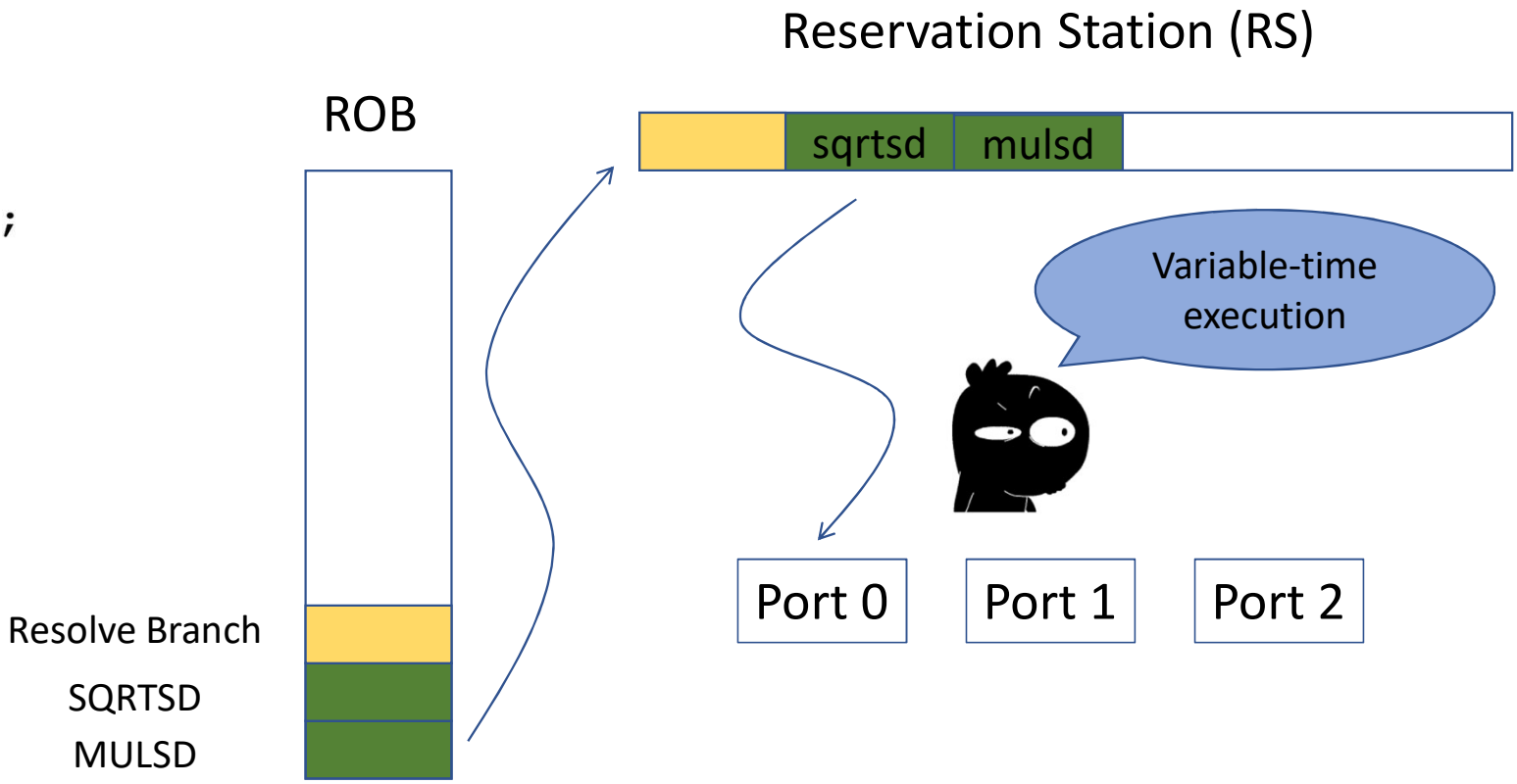
Variant-time execution under speculation

```
// Boundary Check
if (isPublic) {
  value = sqrtsd(value);
  value = mulsd(value, value);
}
```



Variant-time execution under speculation

```
// Boundary Check
if (isPublic) {
  value = sqrtsd(value);
  value = mulsd(value, value);
}
```



Variant-time execution under speculation

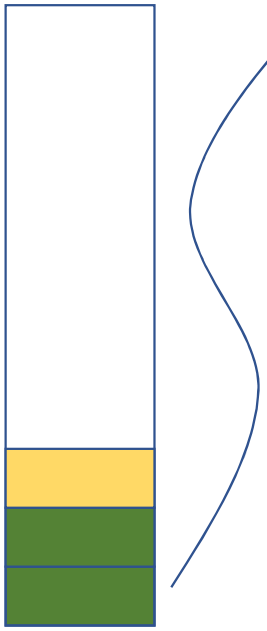
```
// Boundary Check
if (isPublic) {
  value = sqrtsd(value);
  value = mulsd(value, value);
}
```

What if we have more variable-time instructions?

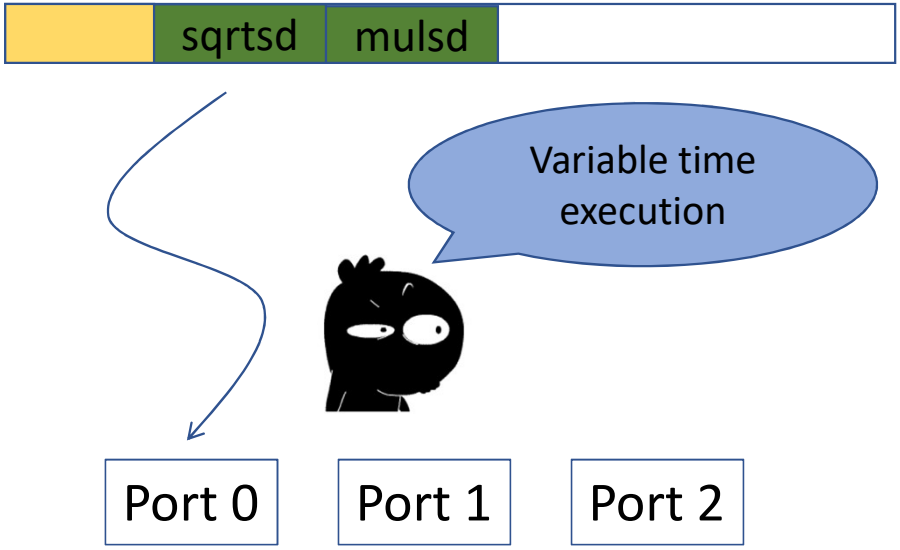
Resolve Branch

SQRTSD

MULSD



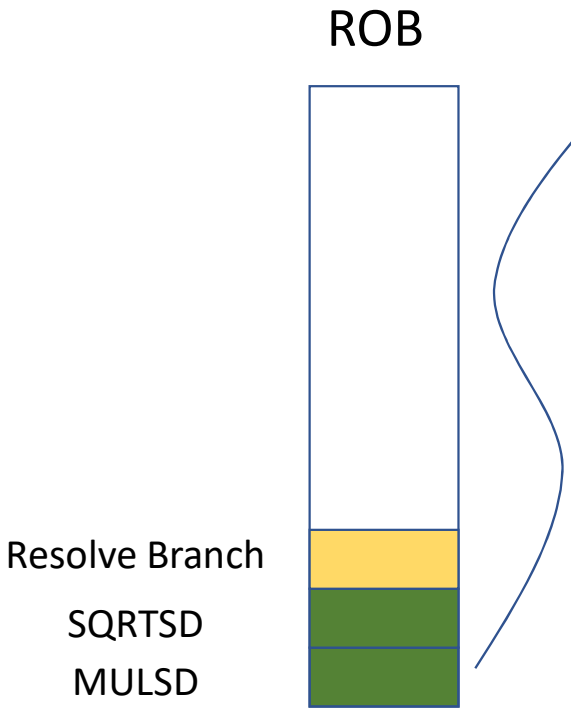
Reservation Station (RS)

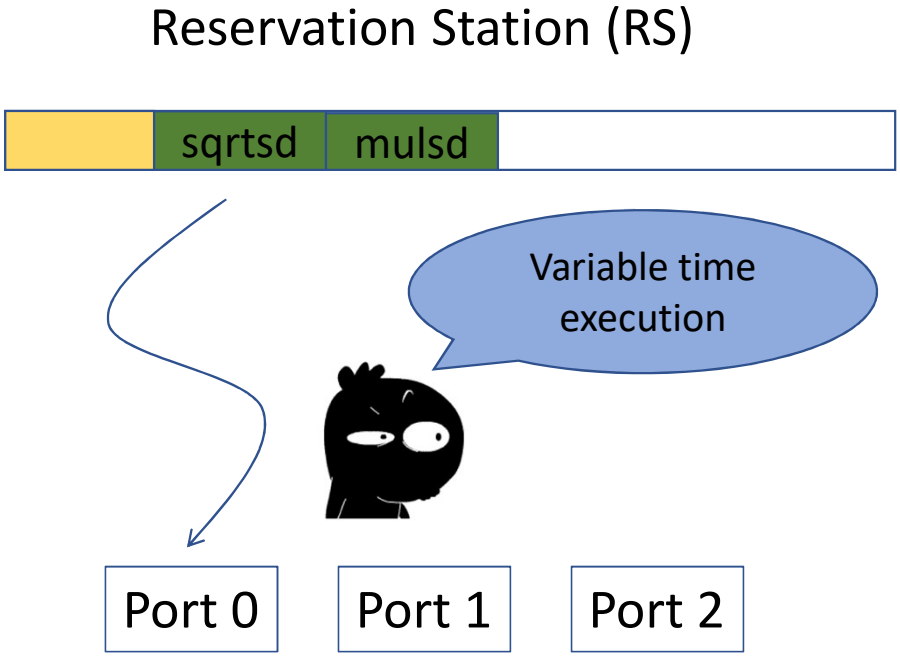


Spectre Attack on variant-time executions

```
// Boundary Check
if (isPublic) {
  value = sqrtsd(value);
  value = mulsd(value, value);
  value = sqrtsd(value);
  value = mulsd(value, value);
  ...
}
```

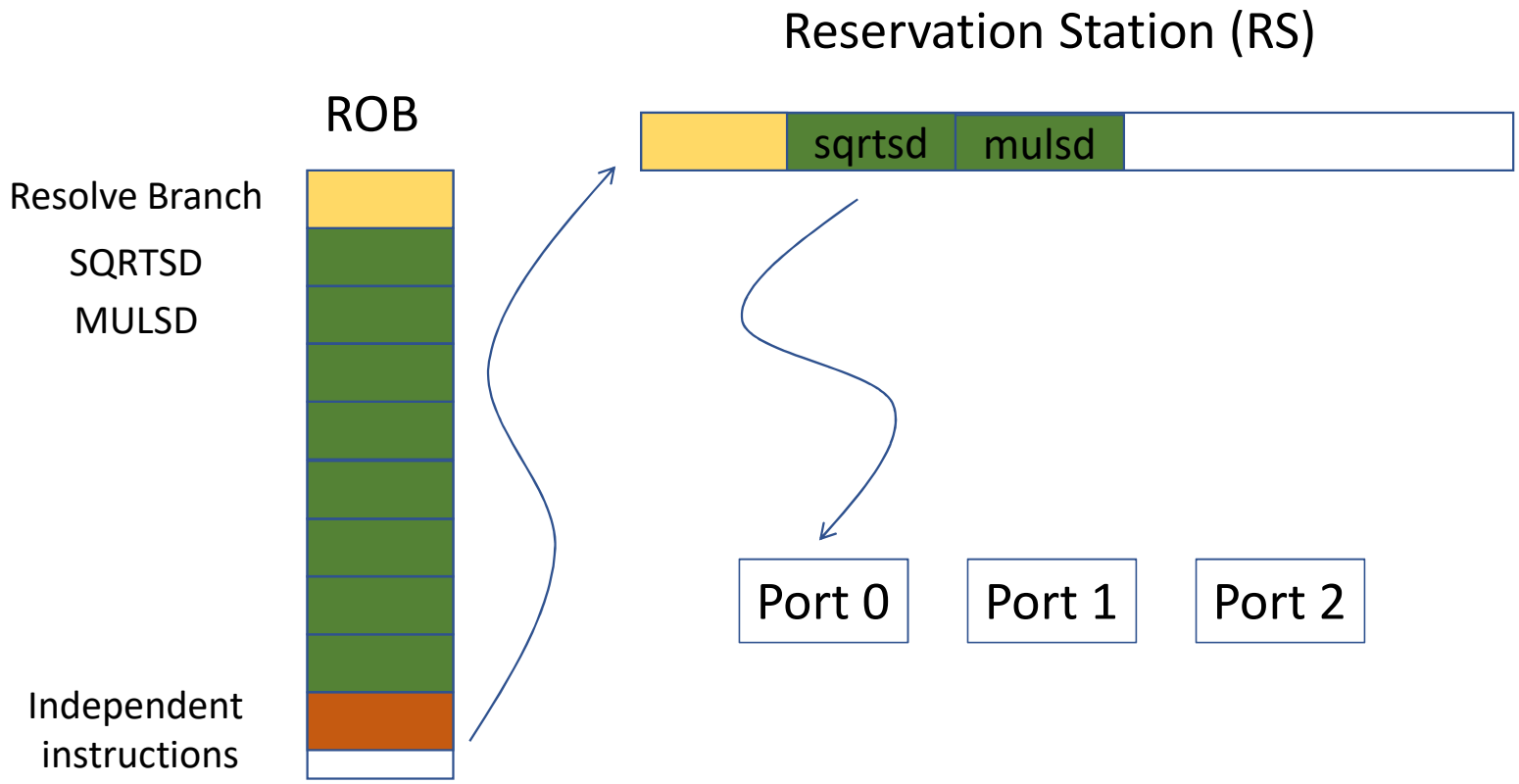
What if we have more variable time instructions?





Spectre Attack on variant-time executions

```
// Boundary Check
if (isPublic) {
  value = sqrtsd(value);
  value = mulsd(value, value);
  value = sqrtsd(value);
  value = mulsd(value, value);
  ...
}
```



Spectre Attack on variant-time executions

```
// Boundary Check
if (isPublic) {
  value = sqrtsd(value);
  value = mulsd(value, value);
  value = sqrtsd(value);
  value = mulsd(value, value);
  ...
}
```

Instructions in ROB is always fixed

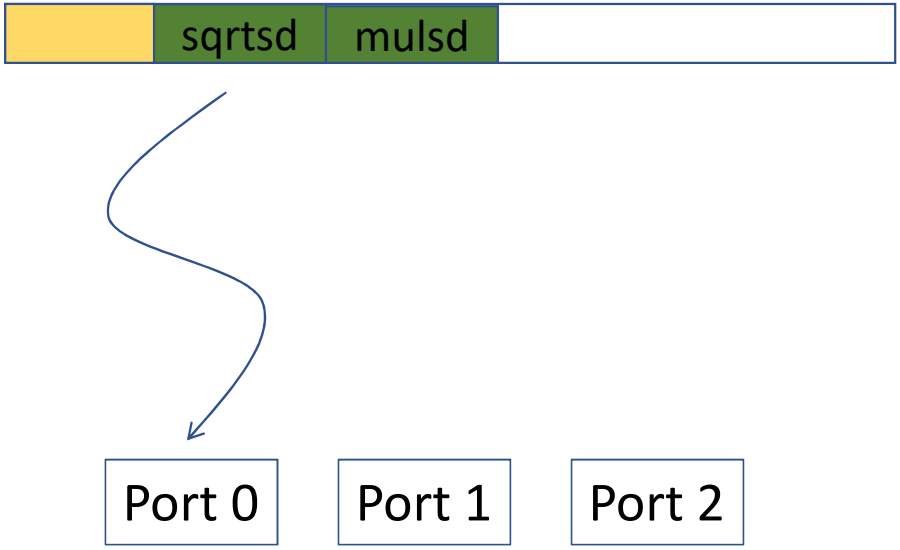
Resolve Branch

SQRTSD

MULSD

Independent instructions

Reservation Station (RS)



Spectre Attack on variant-time executions

```
// Boundary Check
if (isPublic) {
  value = sqrtsd(value);
  value = mulsd(value, value);
  value = sqrtsd(value);
  value = mulsd(value, value);
  ...
}
```

Instructions in ROB is always fixed

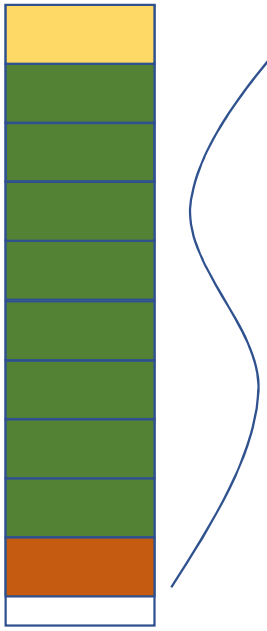
ROB

Resolve Branch

SQRTSD

MULSD

Independent instructions



Reservation Station (RS)

RS will be filled up

Port 0 Port 1 Port 2

Spectre Attack on variant-time executions

```
// Boundary Check
if (isPublic) {
  value = sqrtsd(value);
  value = mulsd(value, value);
  value = sqrtsd(value);
  value = mulsd(value, value);
  ...
}
```

Instructions in ROB is always fixed

Independent instruction may be blocked

Resolve Branch

SQRTSD

MULSD

ROB

Independent instructions

Reservation Station (RS)

RS will be filled up

Port 0

Port 1

Port 2

Spectre Attack on variant-time executions

```
// Boundary Check
if (isPublic) {
  value = sqrtsd(value);
  value = mulsd(value, value);
  value = sqrtsd(value);
  value = mulsd(value, value);
  ...
  independent_memory_access(adrs);
}
```

Instructions in ROB is always fixed

Independent instruction may be blocked

ROB

Resolve Branch

SQRTSD

MULSD

Independent instructions

Reservation Station (RS)

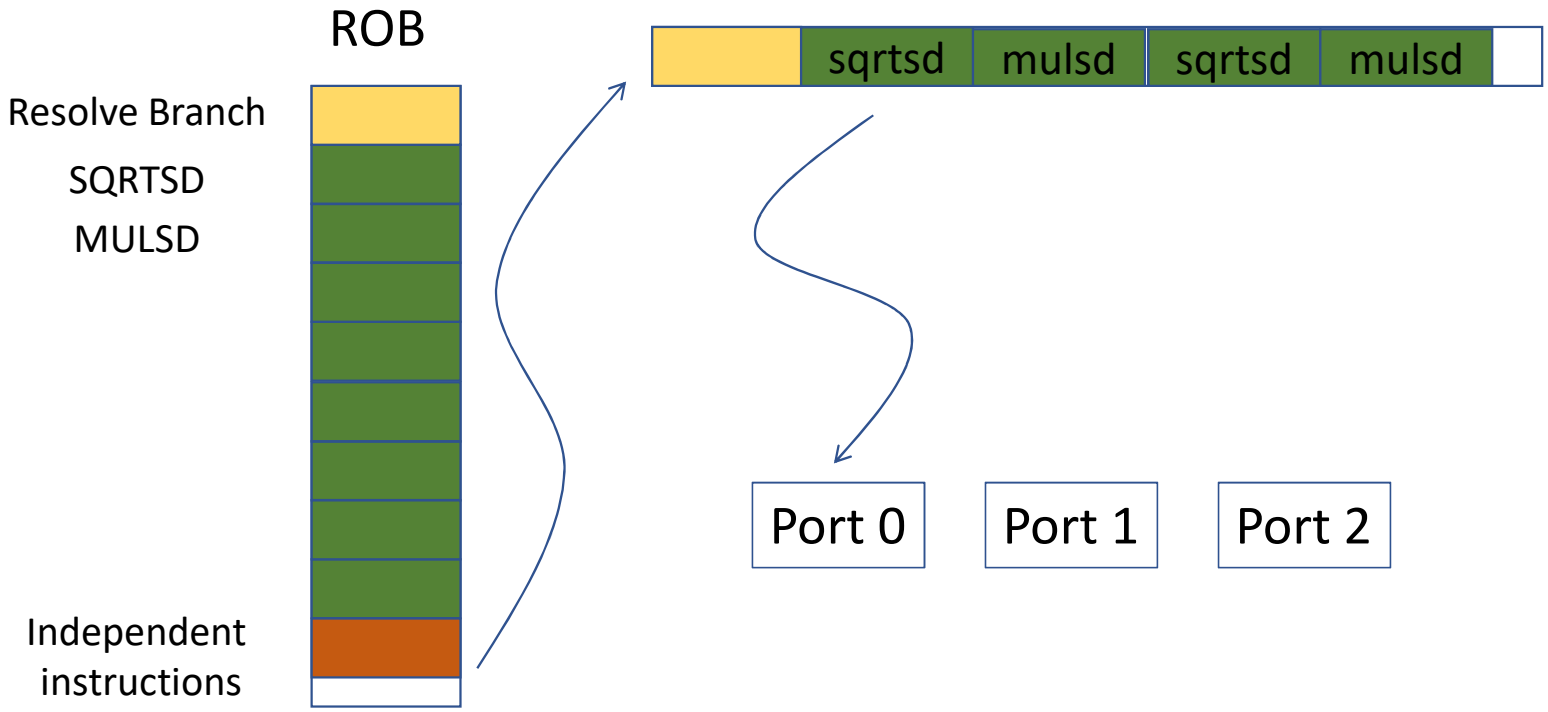
RS will be filled up

Port 0 Port 1 Port 2

Spectre Attack on variant-time executions

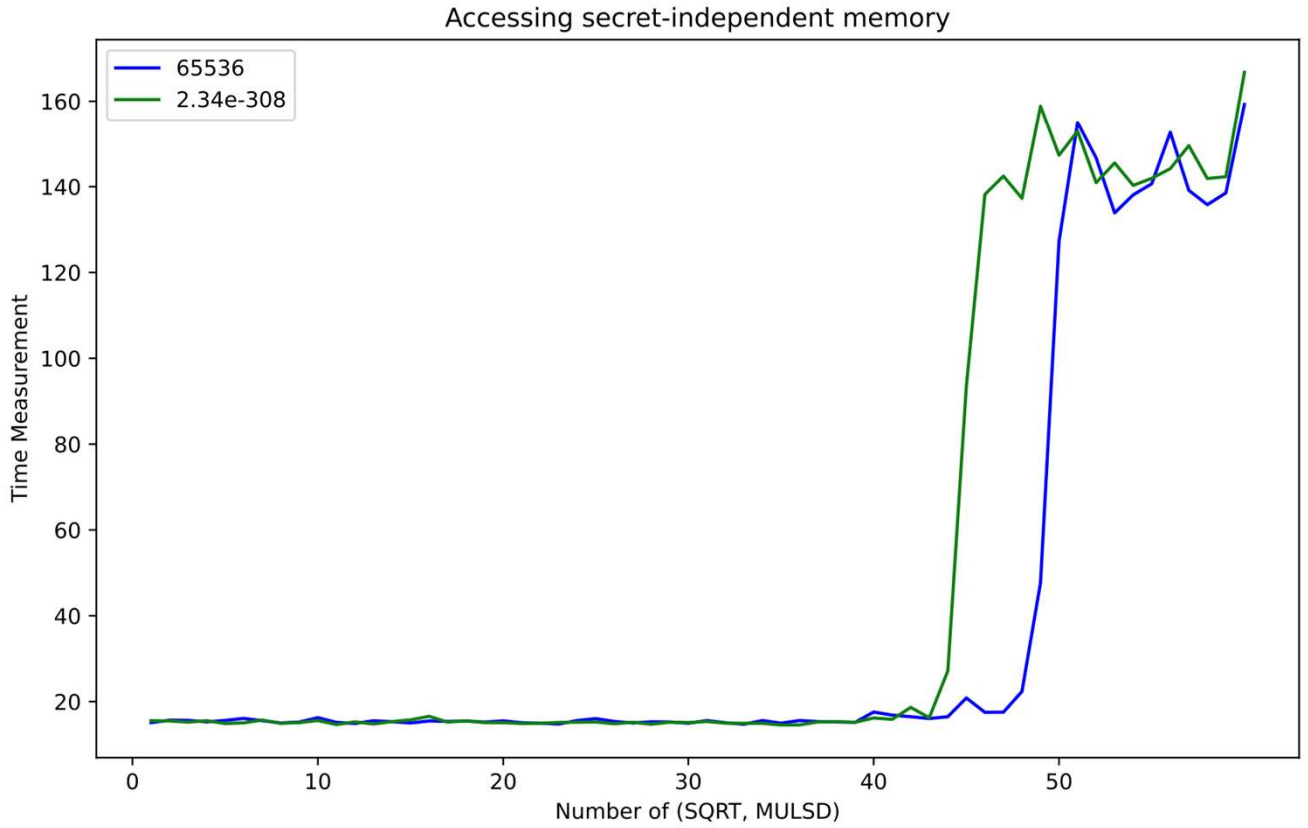
```
// Boundary Check
if (isPublic) {
  value = sqrtsd(value);
  value = mulsd(value, value);
  value = sqrtsd(value);
  value = mulsd(value, value);
  ...
  independent_memory_access(adrs);
}
```

the load may not happen, if RS is not freed before the resolve of branch



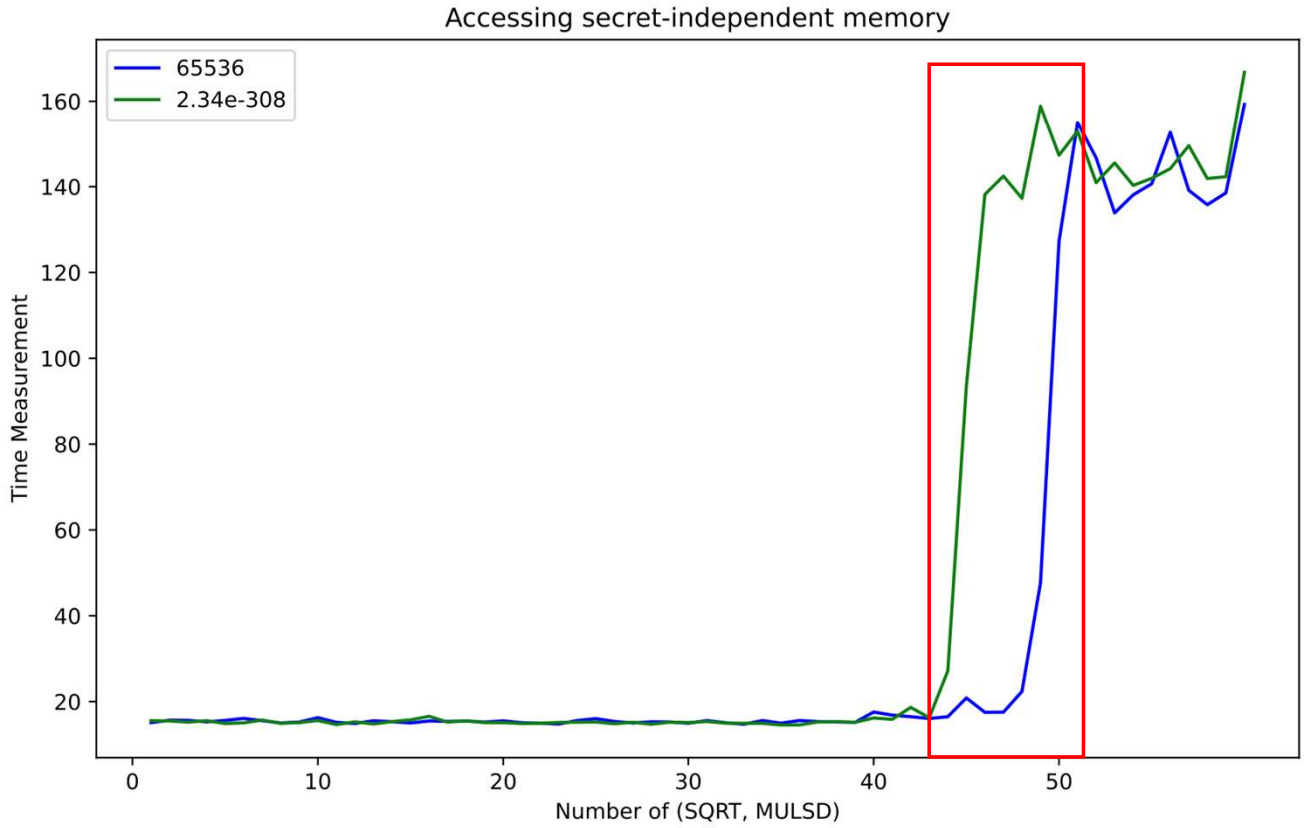
Limitation of resources

```
victim(double value, int isPublic) {
  // Branch training
  for (volatile int i = 0; i < 200; i++);

  // Boundary Check
  if (isPublic) {
    value = sqrtsd(value);
    value = mulsd(value, value);
    ...
    value = sqrtsd(value);
    value = mulsd(value, value);
    memory_access(adrs);
  }
}
```



Limitation of resources

```
victim(double value, int isPublic) {
  // Branch training
  for (volatile int i = 0; i < 200; i++);

  // Boundary Check
  if (isPublic) {
    value = sqrtsd(value);
    value = mulsd(value, value);
    ...
    value = sqrtsd(value);
    value = mulsd(value, value);
    memory_access(adrs);
  }
}
```



Fixing SLH

```
  // Boundary Check
  if (isPublic) {
    value = sqrtsd(value);
    value = mulsd(value, value);
    ...
    value = sqrtsd(value);
    value = mulsd(value, value);
    memory_access(adrs);
  }
}
```

Fixing SLH

```
mask = 0;
// Boundary Check
if (isPublic) {
  mask = isPublic ? mask : -1;
  value = sqrtsd(value);
  value = mulsd(value, value);
  ...
  value = sqrtsd(value);
  value = mulsd(value, value);
  memory_access(adrs);
}
```

Compiled by SLH

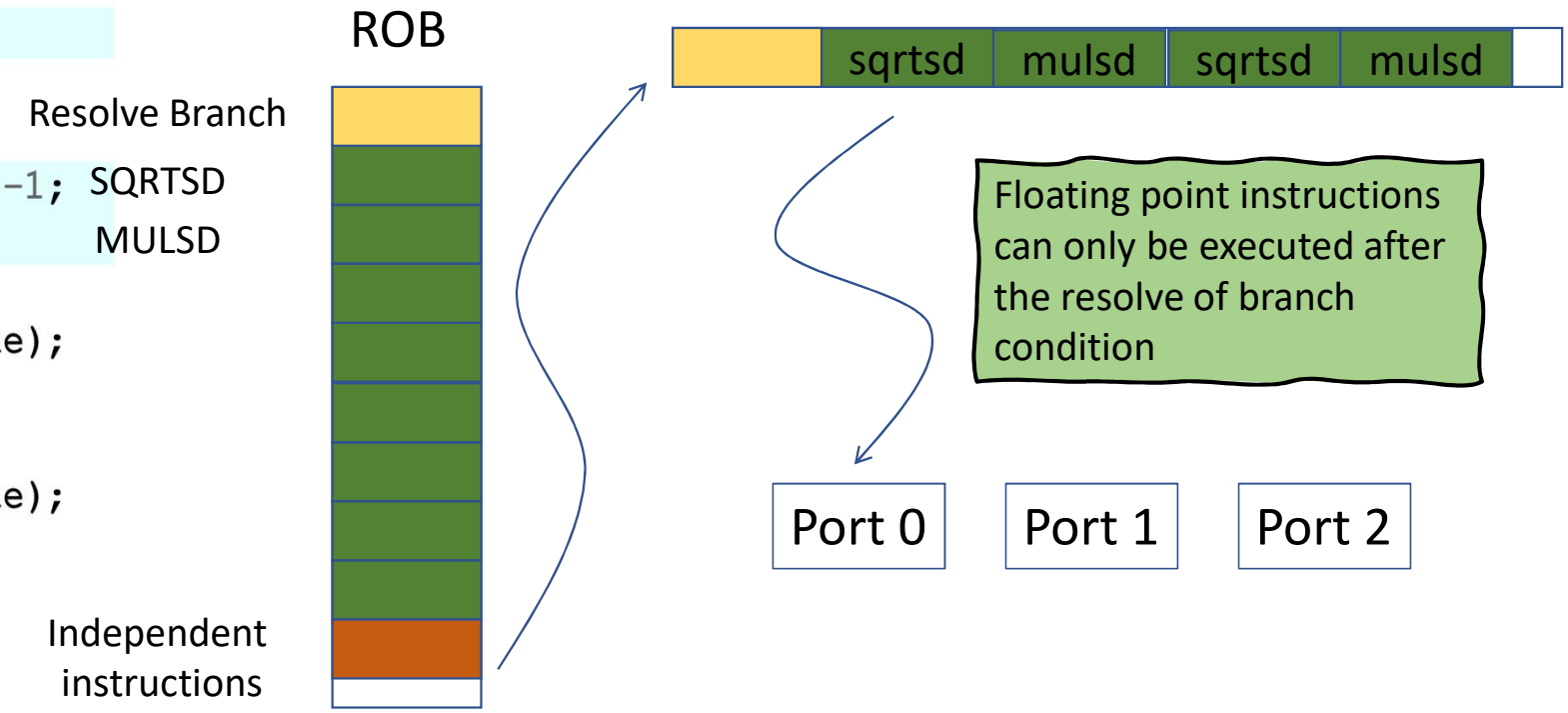
Fixing SLH

```
mask = 0;
// Boundary Check
if (isPublic) {
  mask = isPublic ? mask : -1;
  value |= mask;
  value = sqrtsd(value);
  value = mulsd(value, value);
  ...
  value = sqrtsd(value);
  value = mulsd(value, value);
  memory_access(adrs);
}
```

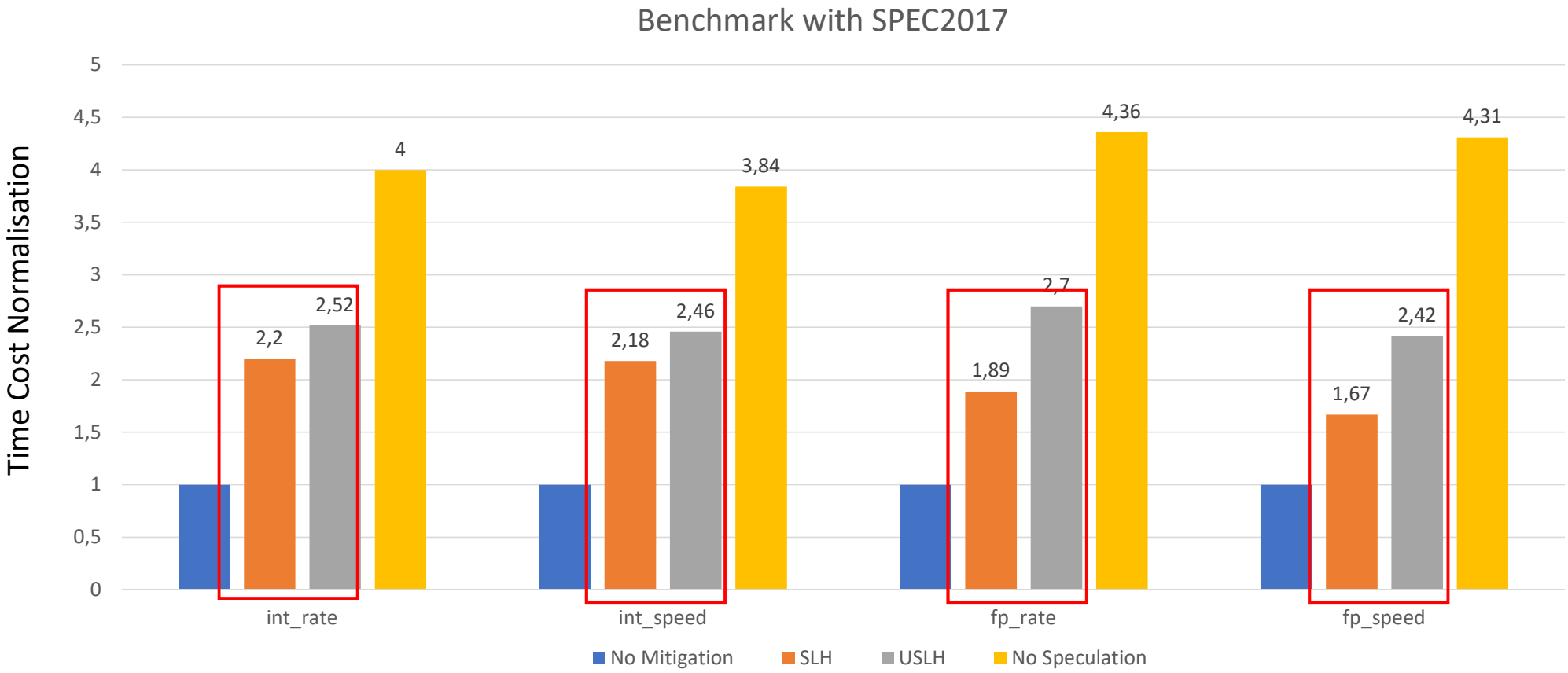
Compiled by USLH

Fixing SLH

```
mask = 0;
// Boundary Check
if (isPublic) {
  mask = isPublic ? mask : -1;
  value |= mask;
  value = sqrtsd(value);
  value = mulsd(value, value);
  ...
  value = sqrtsd(value);
  value = mulsd(value, value);
  memory_access(adrs);
}
```



Performance



Summary

- SLH is good, but it is not perfect
- Variable-time executions also leak secret in speculative execution
- Accessing a secret-independent memory may also leak information
- USLH costs more, but it is still better than disabling speculative execution