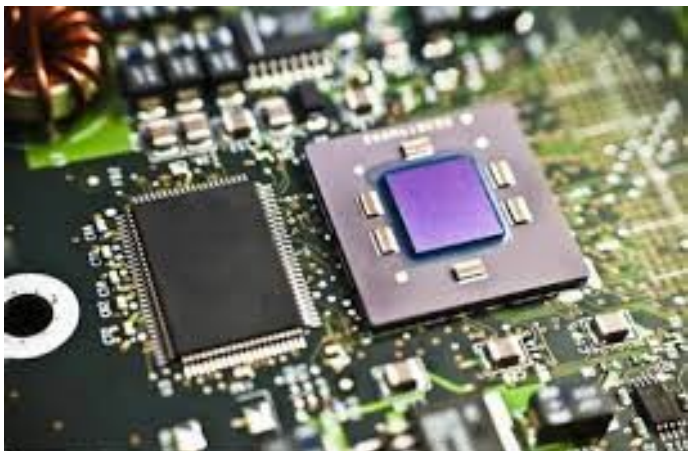
Microarchitectural Attack

Transient State Attacks



Dr Milan Patnaik
Indian Institute of Technology Madras, India
Rashtriya Raksha University, India

@ Thanks to Prof Chester Rebiero, IIT Madras for some of his slides

Outline

- Cache Timing Attacks.
  - Cache Covert Channel.
  - Flush + Reload Attack
- Cache Collision Attacks.
  - Prime + Probe Attack
  - Time Driven Attacks
- Transient Micro-architectural Attacks.
  - Meltdown
  - Spectre

Security

- Cryptography
- Passwords
- Information Flow Policies
- Privileged Rings
- ASLR
- Virtual Machines and confinement
- Javascript and HTML5
  (due to restricted access to system resouces)
- Enclaves (SGX and Trustzone)

Security

- Cryptography
- Passwords
- Information Flow Policies
- Privileged Rings
- ASLR
- Virtual Machines and confinement
- Javascript and HTML5
  (due to restricted access to system resouces)
- Enclaves (SGX and Trustzone)

Cache timing attack

Branch prediction attack

Speculation Attacks

Row hammer

Fault Injection Attacks

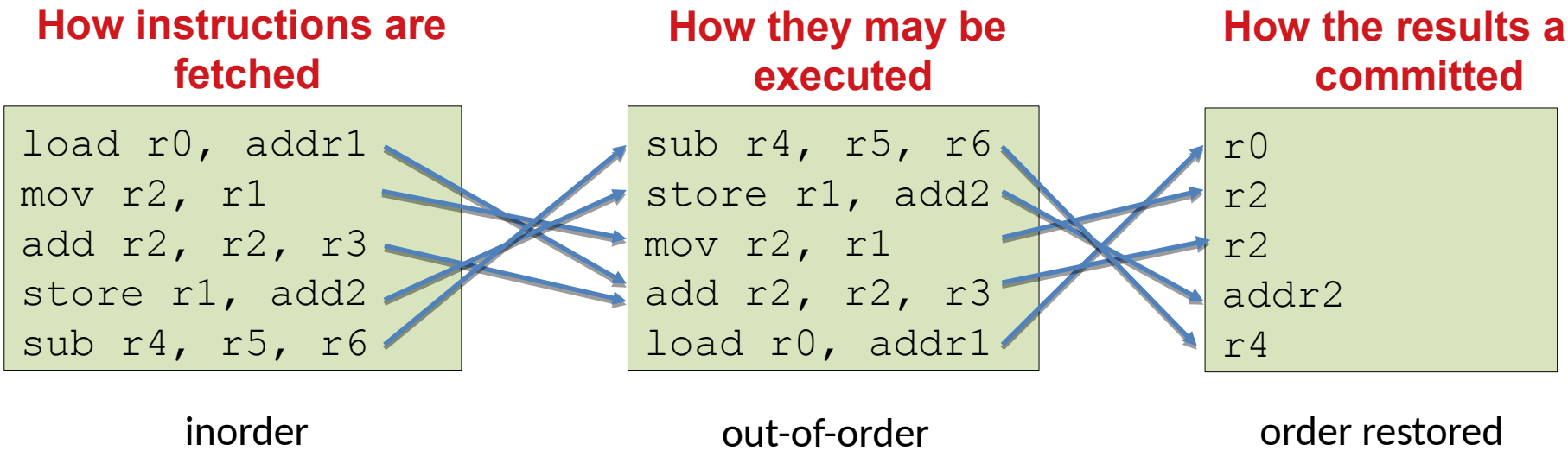
cold boot attacks

DRAM Row buffer (DRAMA)

..... and many more

Instruction Level Parallelism

Out of Order Execution



Out of the processor core, execution looks in-order
Inside the processor core, execution is done out-of-order

Speculative Execution : Case 1

```
  cmp r0, r1
  jnz label
  load r0, addr1
  mov r2, r1
  add r2, r2, r3
  store r1, add2
  sub r4, r5, r6
   :
   :
   :
label:
  more
instructions
```

How instructions are fetched

```
  cmp r0, r1
  jnz label
  load r0, addr1
  mov r2, r1
  add r2, r2, r3
  store r1, add2
  sub r4, r5, r6
   :
   :
   :
label:
  more instructions
```

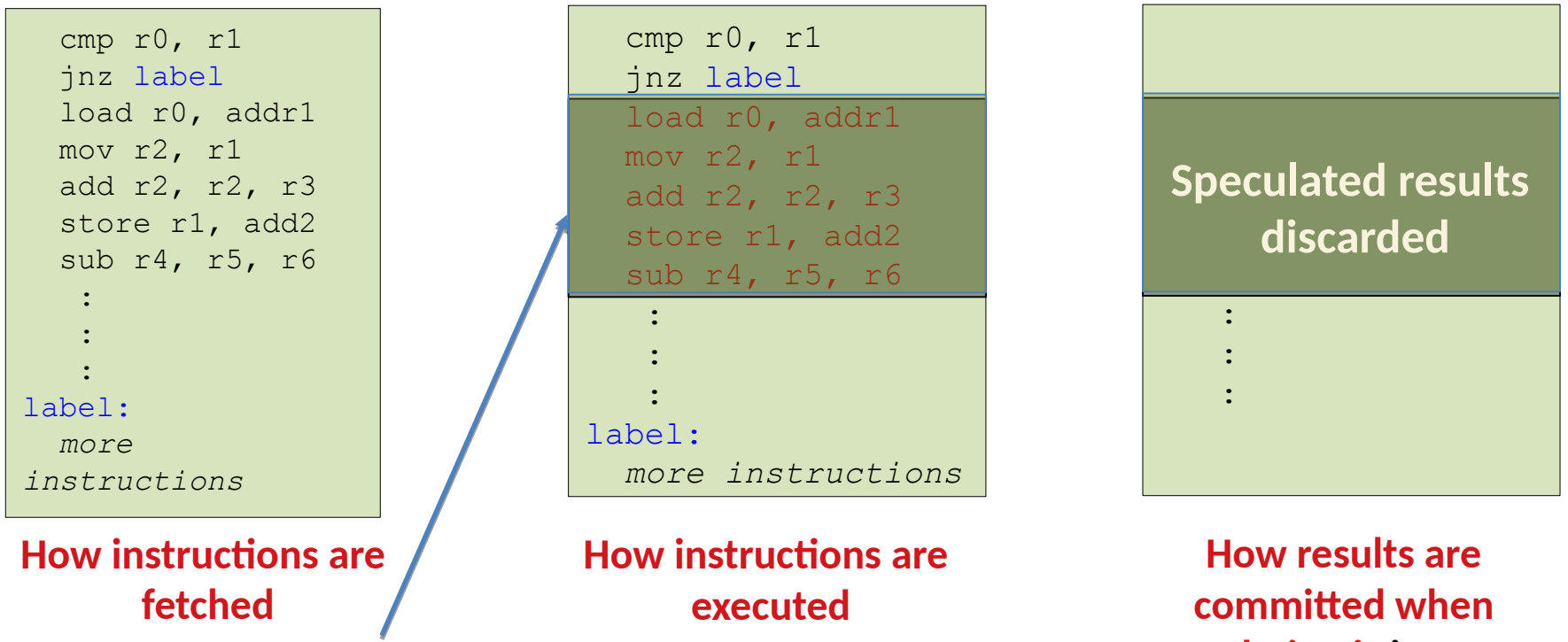
How instructions are executed

```
  r0
  r2
  r2
  add2
  r4
   :
   :
   :
```

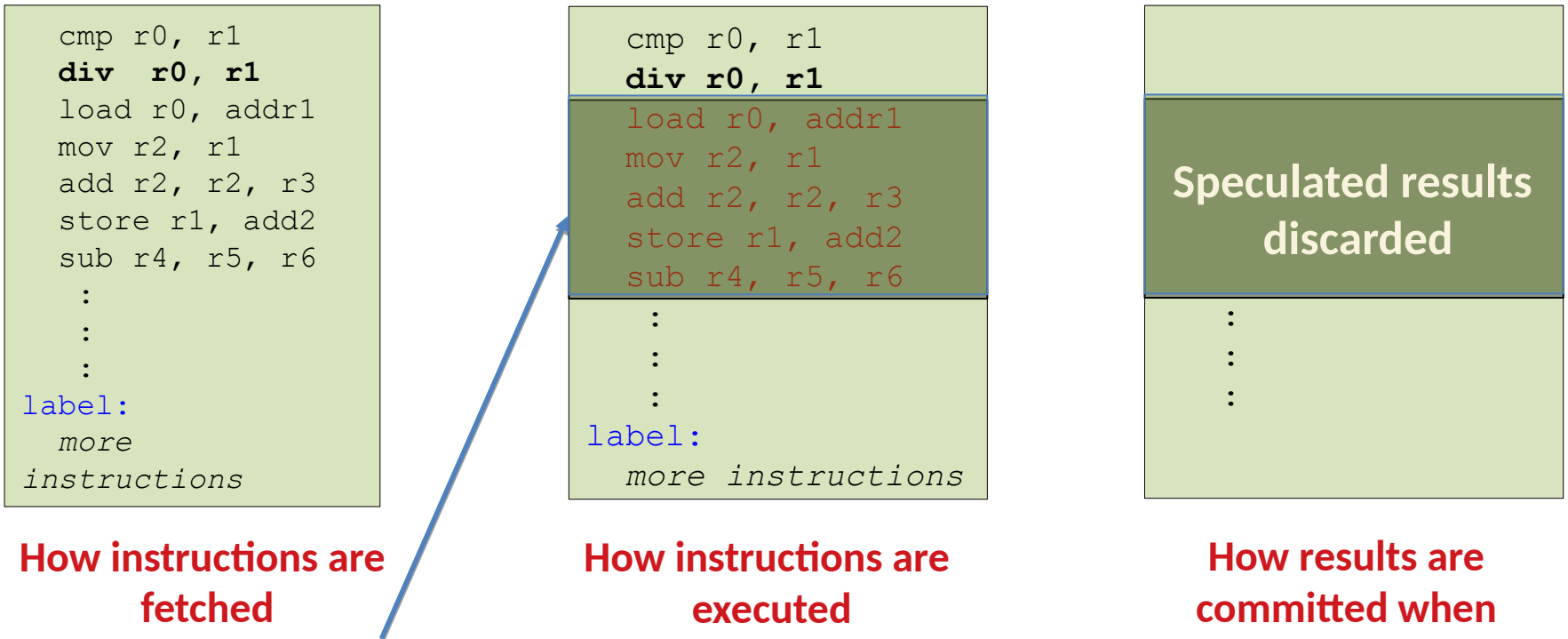
How results are committed when speculation is **correct**

Speculative execution
(transient instructions)

Speculative Execution : Case 1



Speculative Execution : Case 2



ILP Processors in Modern Processors

Common name	Issue structure	Hazard detection	Scheduling	Distinguishing characteristic	Examples
Superscalar (static)	Dynamic	Hardware	Static	In-order execution	Mostly in the embedded space: MIPS and ARM, including the ARM Cortex-A8
Superscalar (dynamic)	Dynamic	Hardware	Dynamic	Some out-of-order execution, but no speculation	None at the present
Superscalar (speculative)	Dynamic	Hardware	Dynamic with speculation	Out-of-order execution with speculation	Intel Core i3, i5, i7; AMD Phenom; IBM Power 7
VLIW/LIW	Static	Primarily software	Static	All hazards determined and indicated by compiler (often implicitly)	Most examples are in signal processing, such as the TI C6x
EPIC	Primarily static	Primarily software	Mostly static	All hazards determined and indicated explicitly by the compiler	Itanium

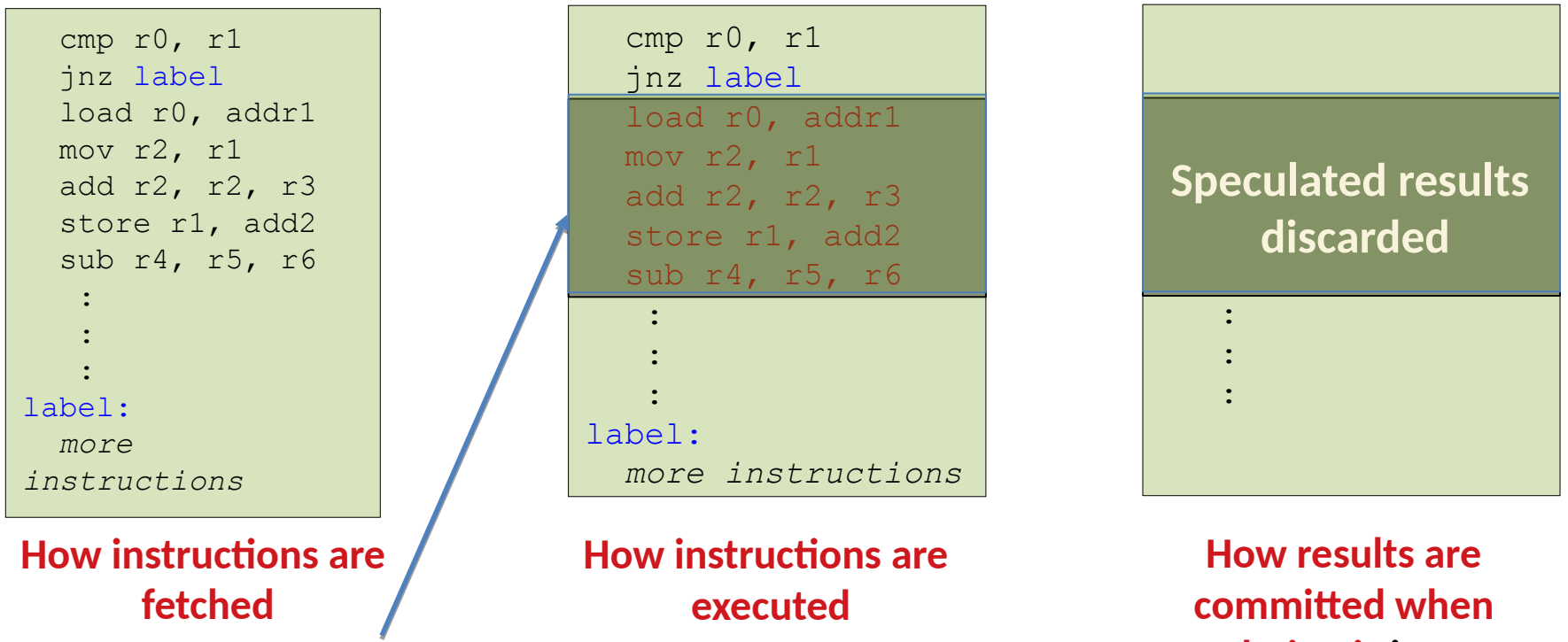
Speculation Attacks

Meltdown and Spectre

Speculation Attacks : Meltdown

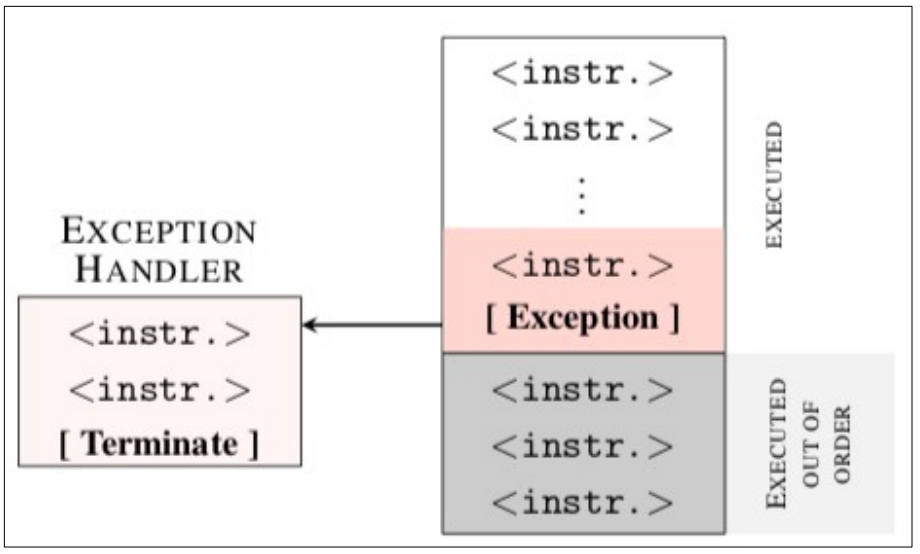
Yuval Yarom’s talk on Meltdown and Spectre at the Cyber security research bootcamp 2018

Speculative Execution : Case 1



Speculative Execution And Micro-architectural State

```
1 raise_exception();
2 // the line below is never reached
3 access(probe_array[data * 4096]);
```

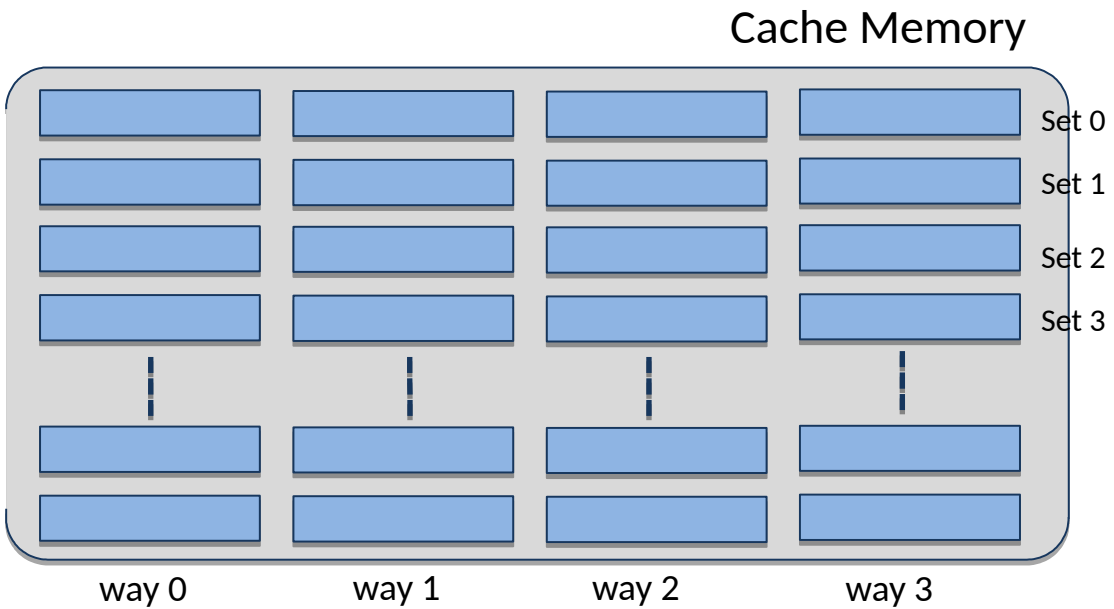


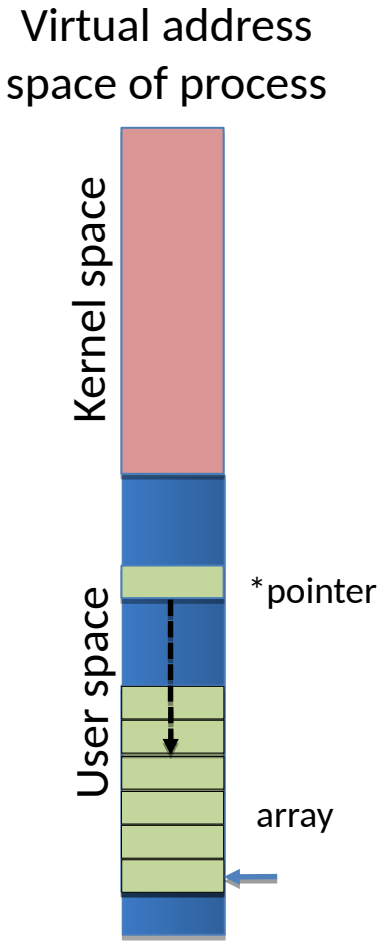
Even though line 3 is not reached, the micro-architectural state is modified due to Line 3.

Meltdown Concept

Normal Circumstances

```
i = *pointer
y = array[i * 256]
```

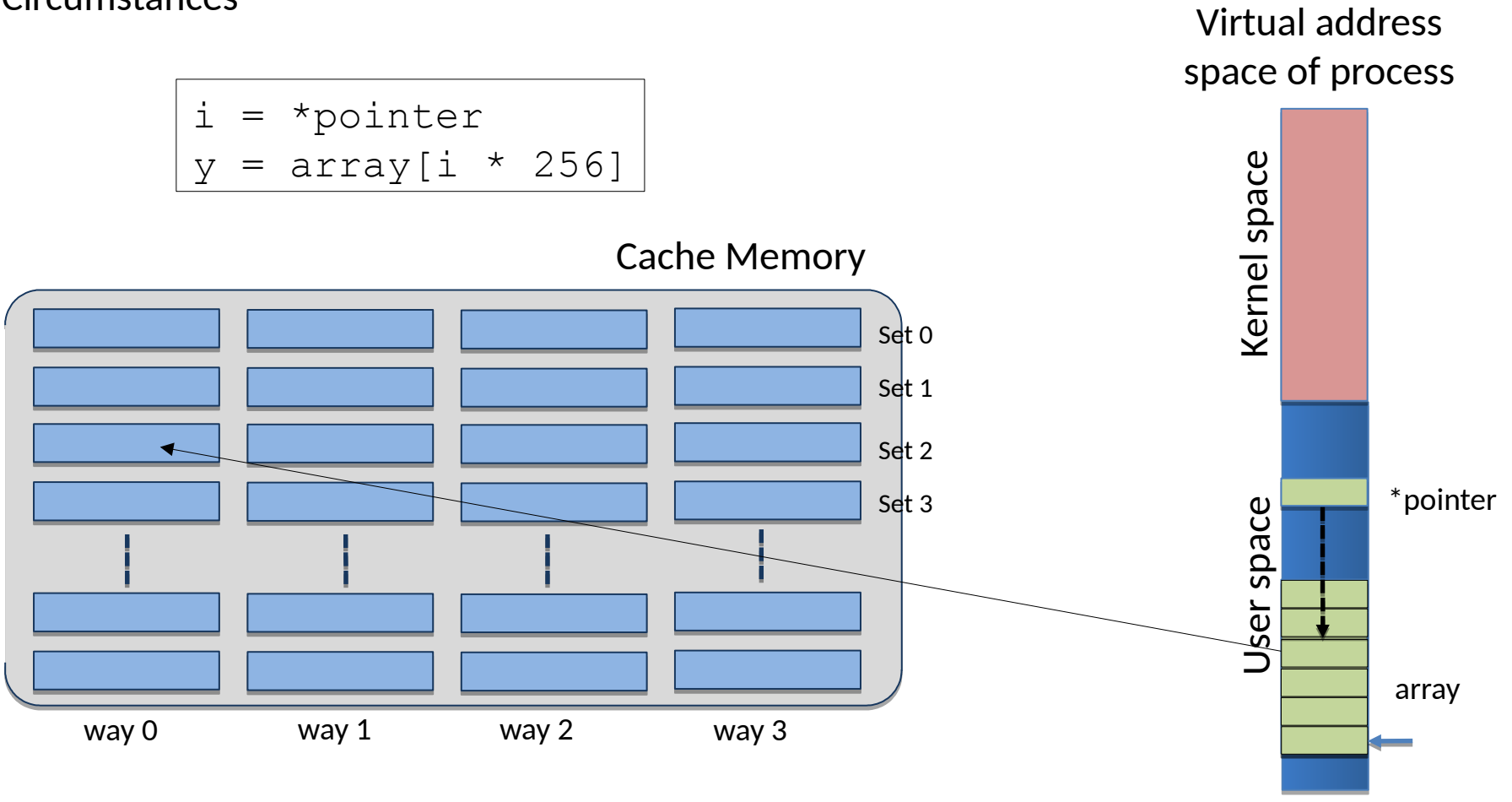




Meltdown Concept

Normal Circumstances

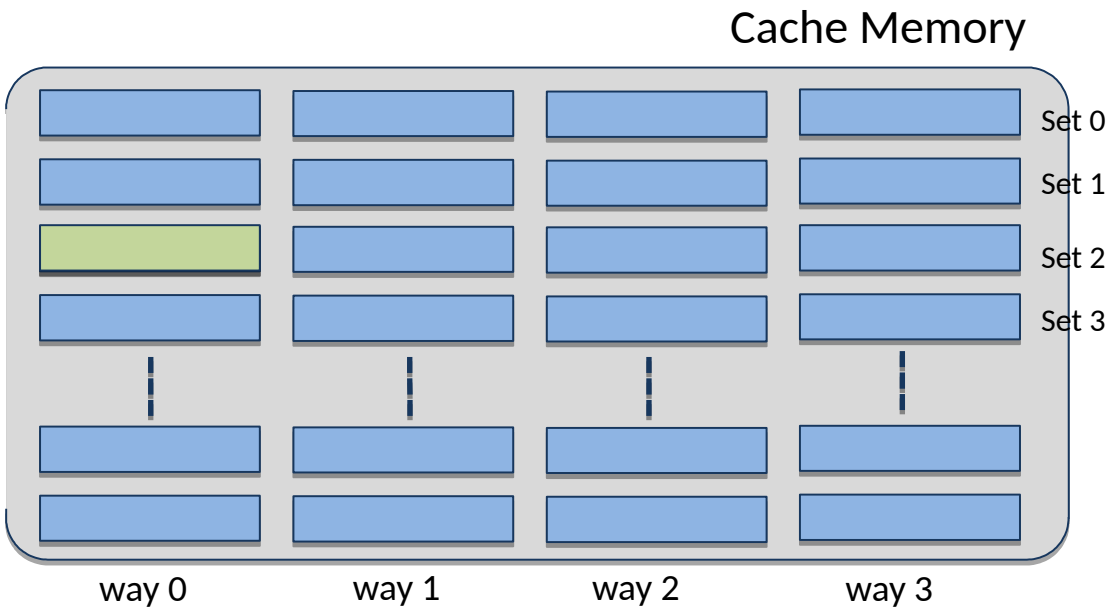
```
i = *pointer
y = array[i * 256]
```

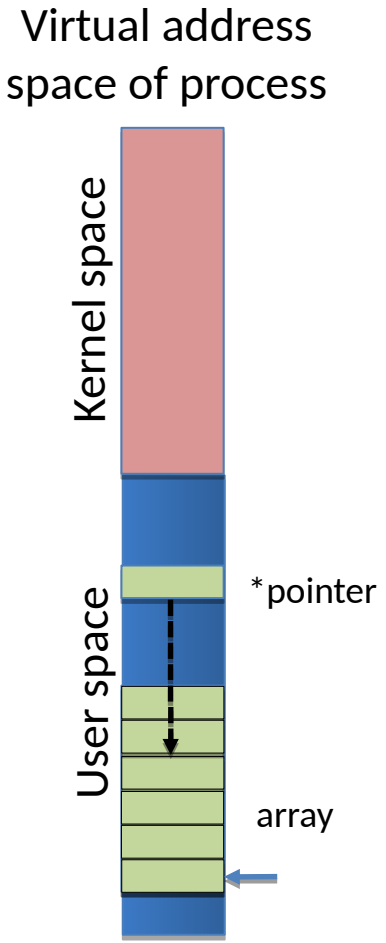


Meltdown Concept

Normal Circumstances

```
i = *pointer
y = array[i * 256]
```

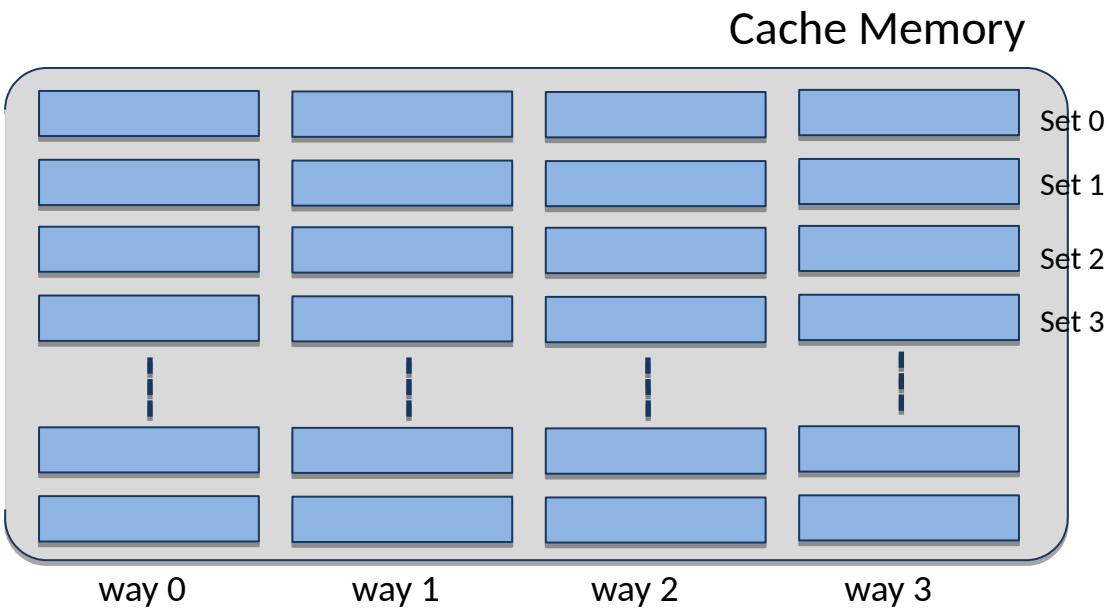


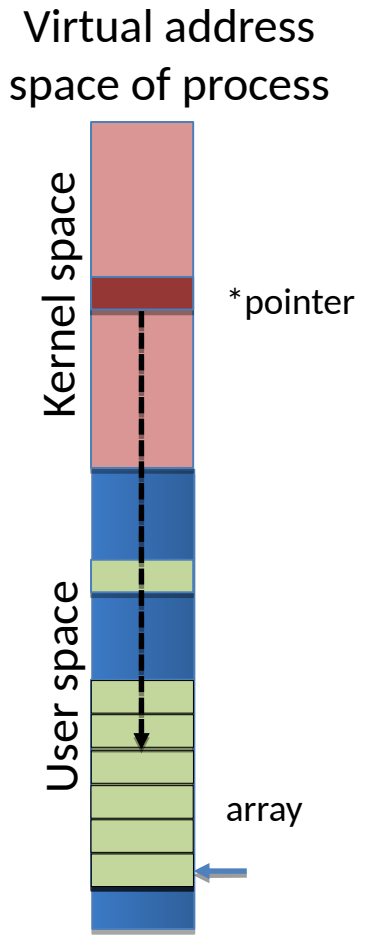


Meltdown Concept

Not normal Circumstances

```
i = *pointer
y = array[i * 256]
```

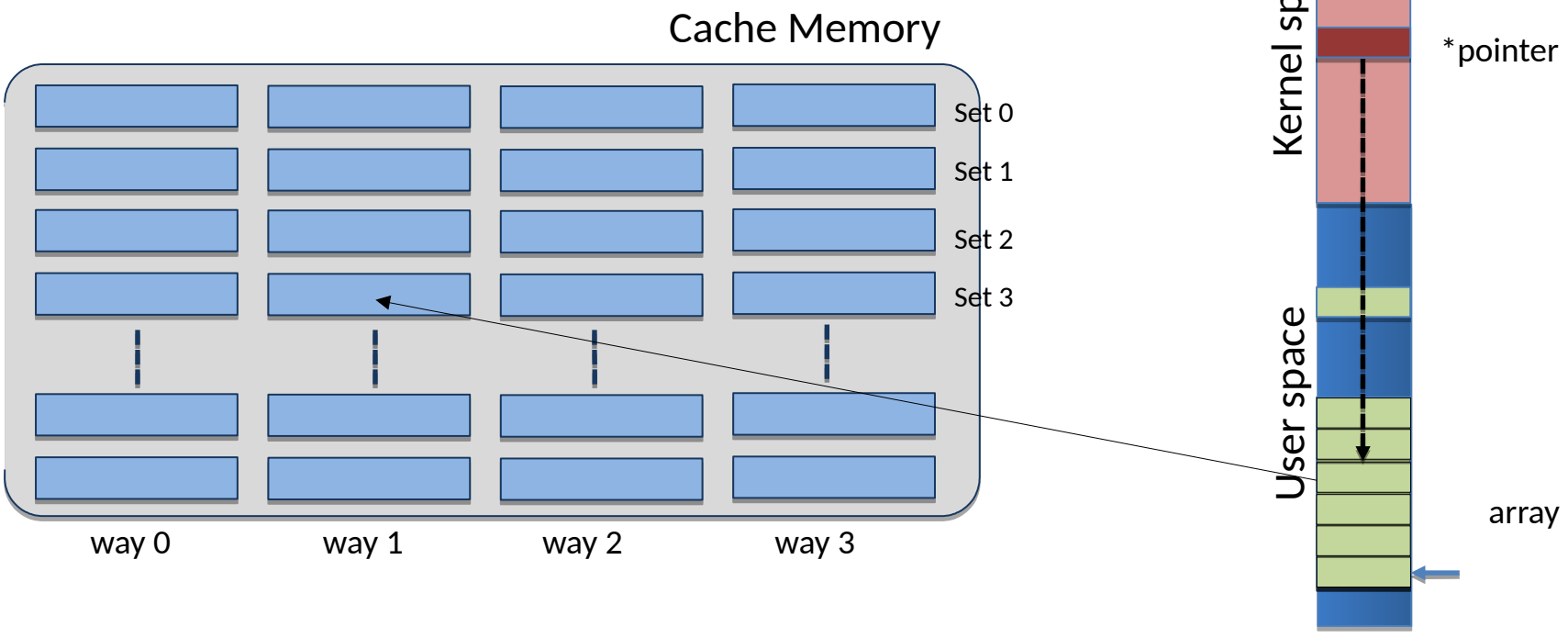




Meltdown Concept

Not normal Circumstances

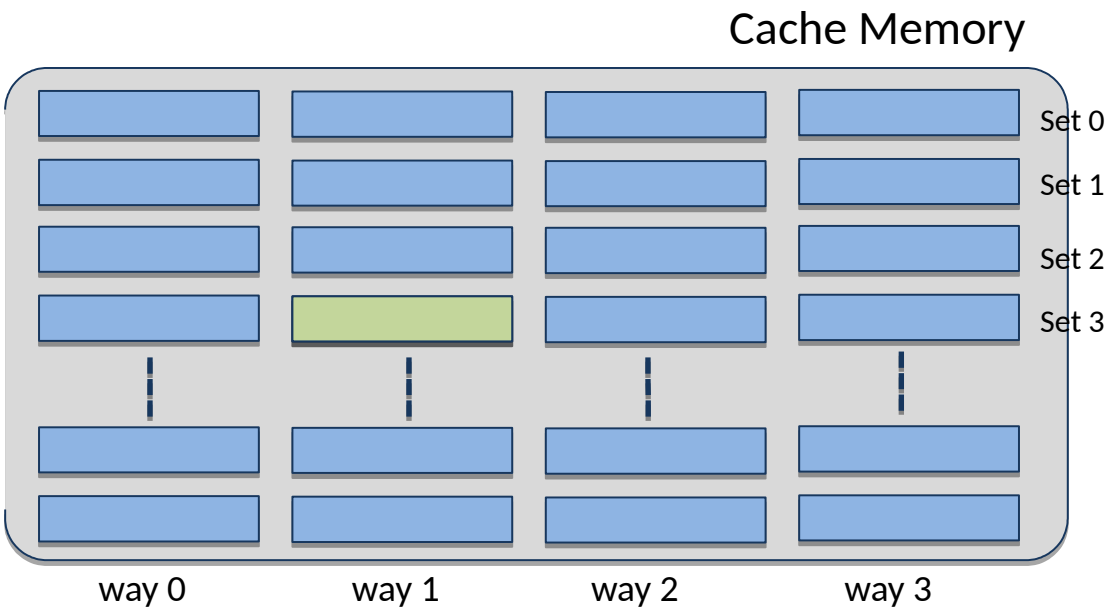
```
i = *pointer
y = array[i * 256]
```

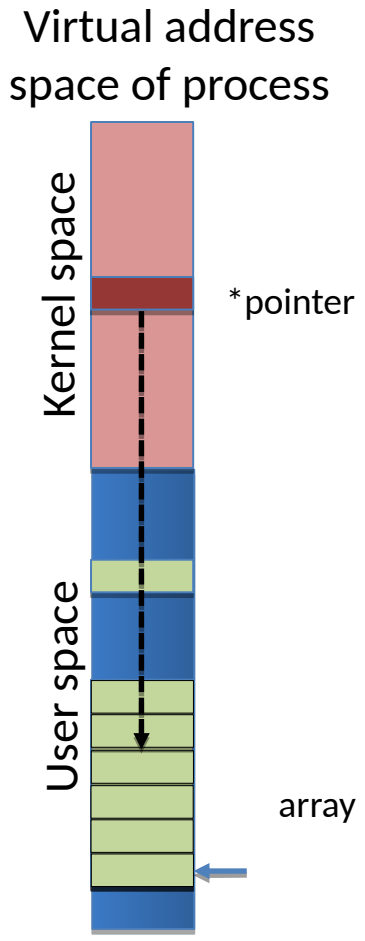


Meltdown Concept

Not normal Circumstances

```
i = *pointer
y = array[i * 256]
```

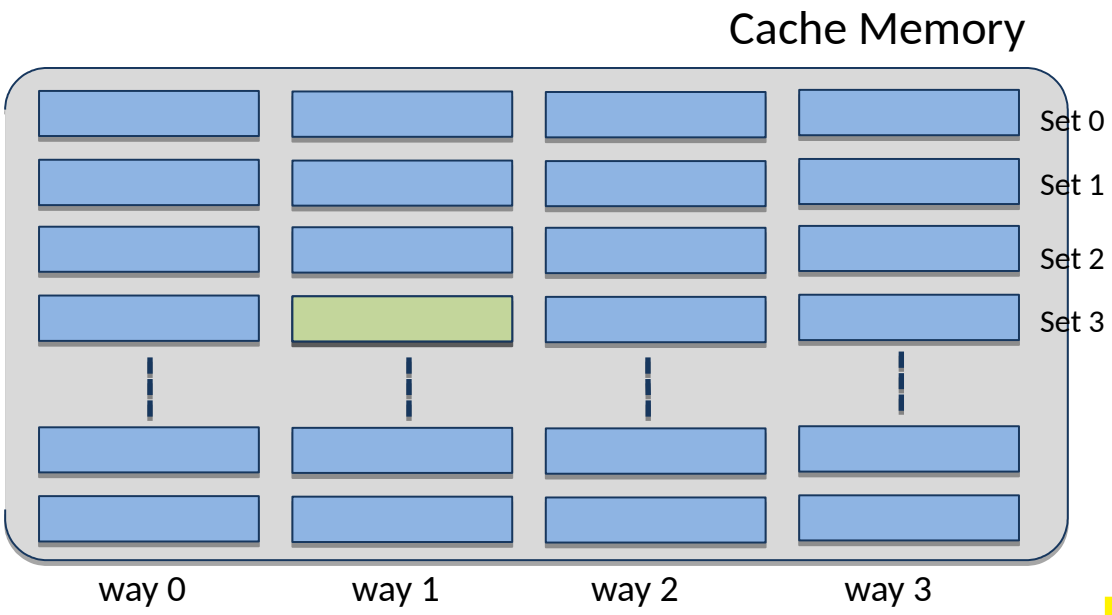




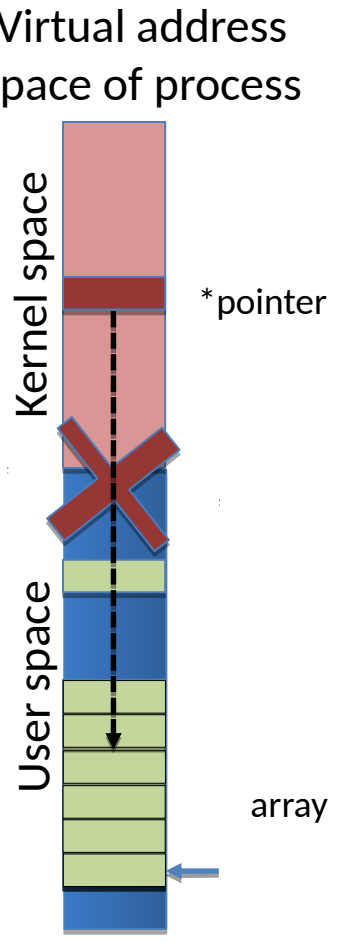
Meltdown Concept

Not normal Circumstances

```
i = *pointer
y = array[i * 256]
```



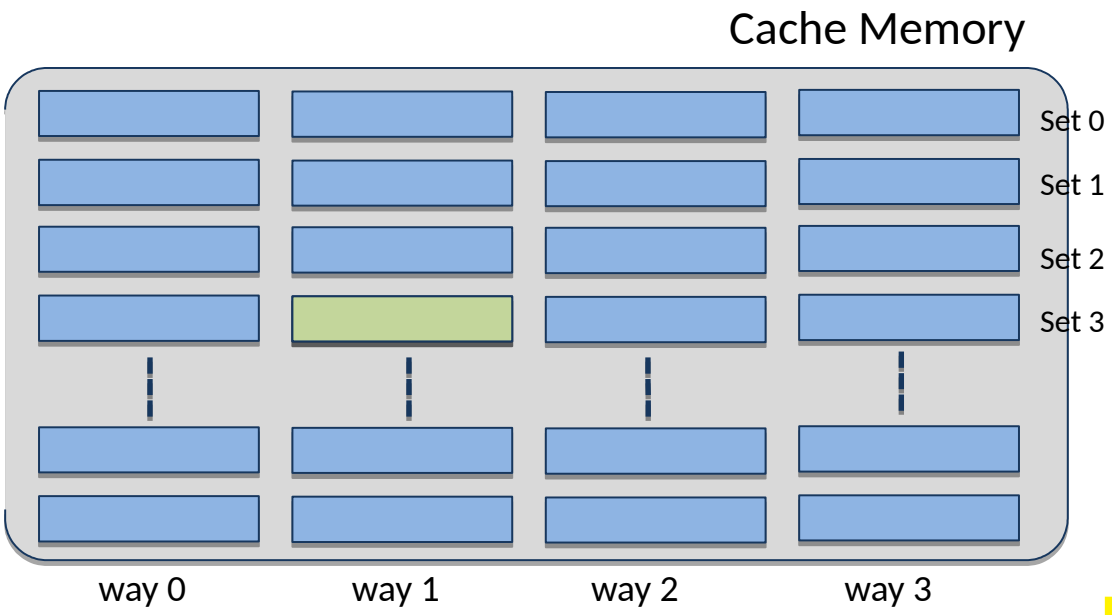
cache miss



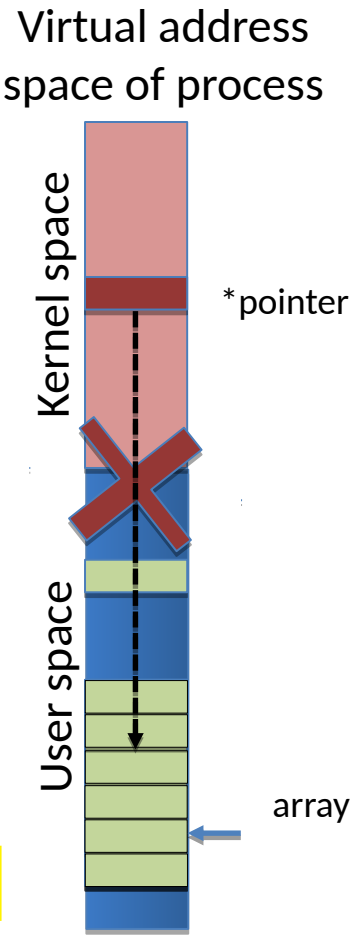
Meltdown Concept

Not normal Circumstances

```
i = *pointer
y = array[i * 256]
```



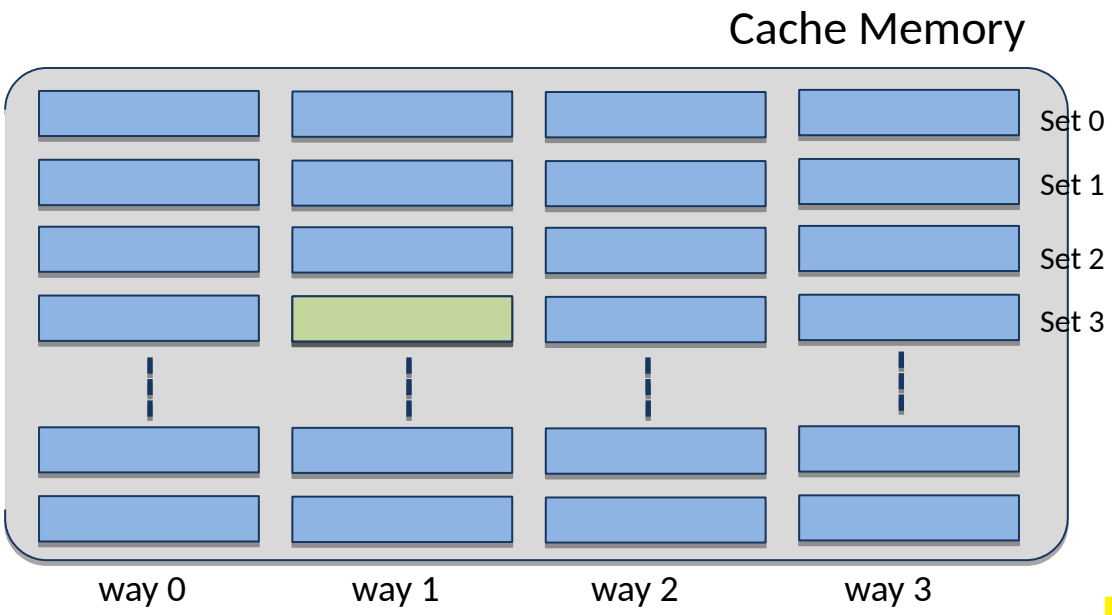
cache miss



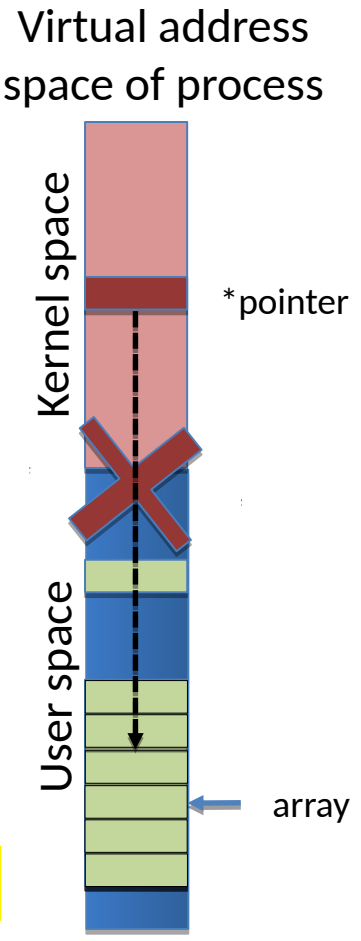
Meltdown Concept

Not normal Circumstances

```
i = *pointer
y = array[i * 256]
```



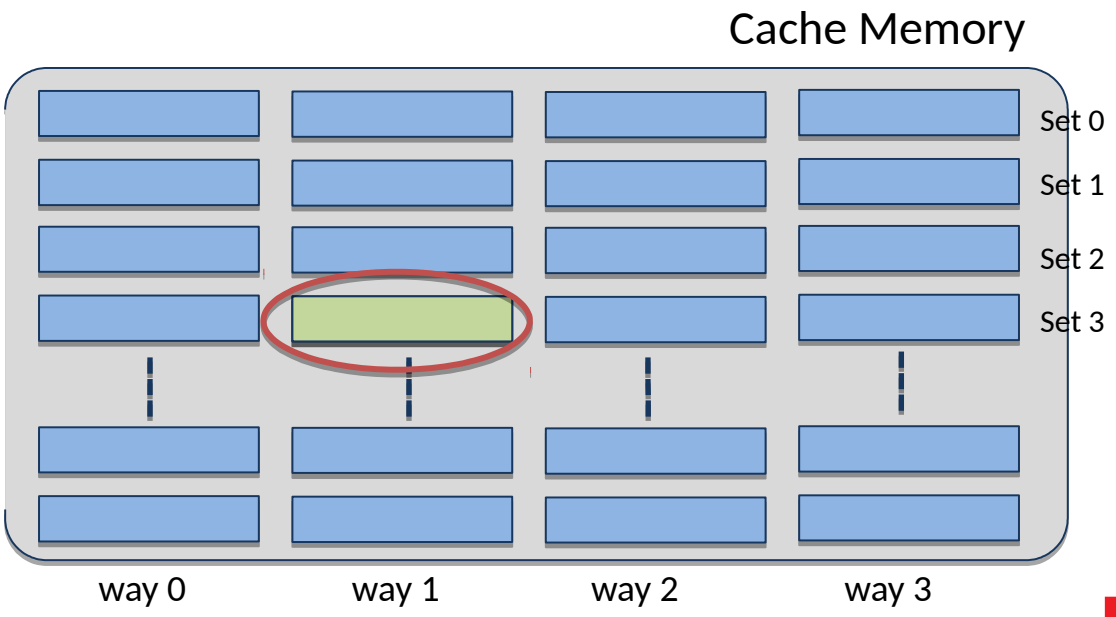
cache miss



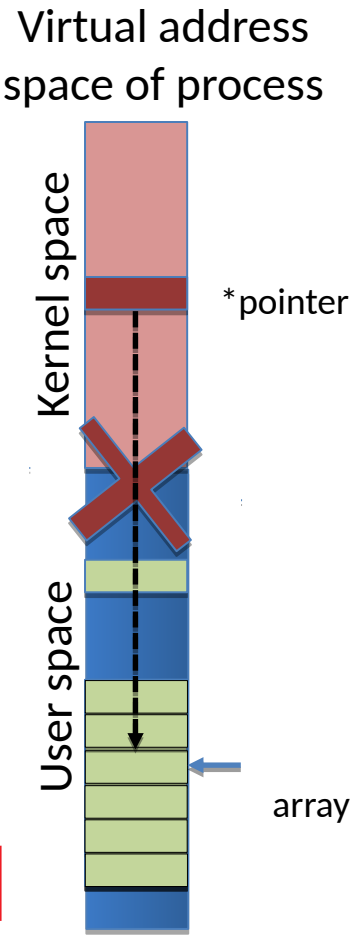
Meltdown Concept

Not normal Circumstances

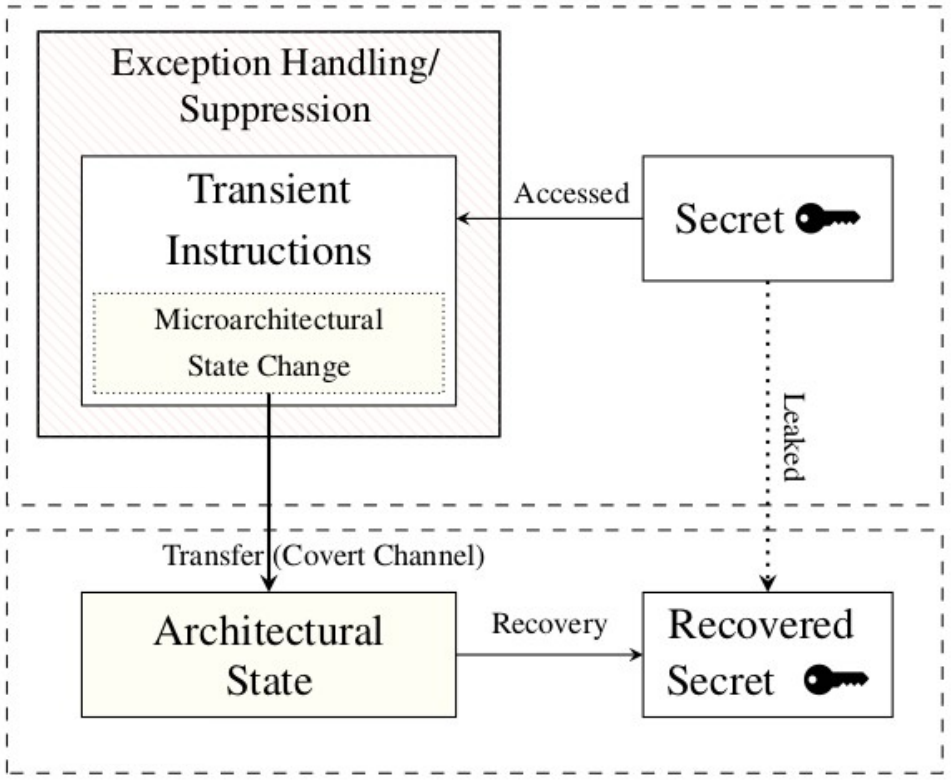
```
i = *pointer
y = array[i * 256]
```



cache hit



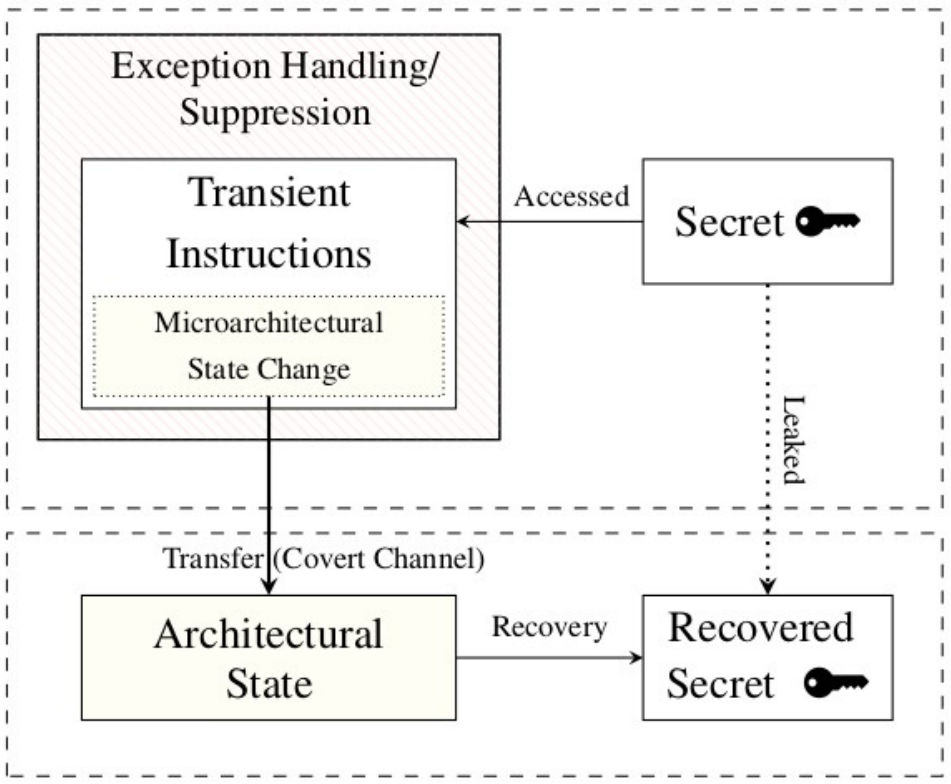
Meltdown : The Attack



Credits : Moritz Lipp et al

* **Executing Transient Instructions**
  - Exception Handling
  - Exception Supression
* **Building a Covert Channel**

Meltdown : The Attack



Credits : Moritz Lipp et al

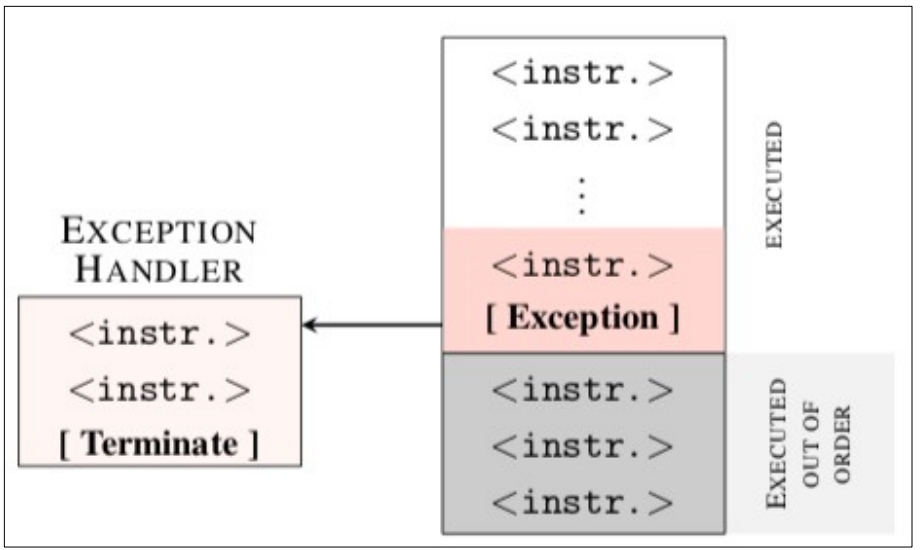
Step 1
Content of attacker-chosen memory location loaded into register.

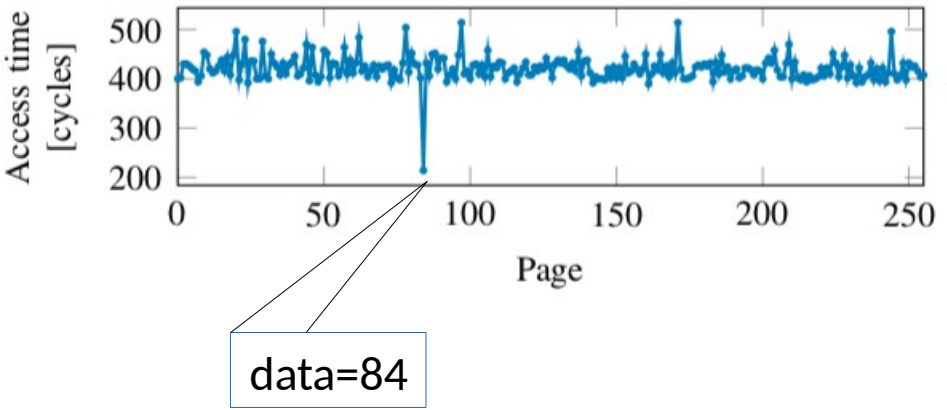
Step 2
Transient instruction accesses cache line based on secret content of register.

Step 3
Attacker uses Flush+Reload to determine accessed cache line and secret stored at chosen memory location.

Speculative Execution And Micro-architectural State

```
1 raise_exception();
2 // the line below is never reached
3 access(probe_array[data * 4096]);
```

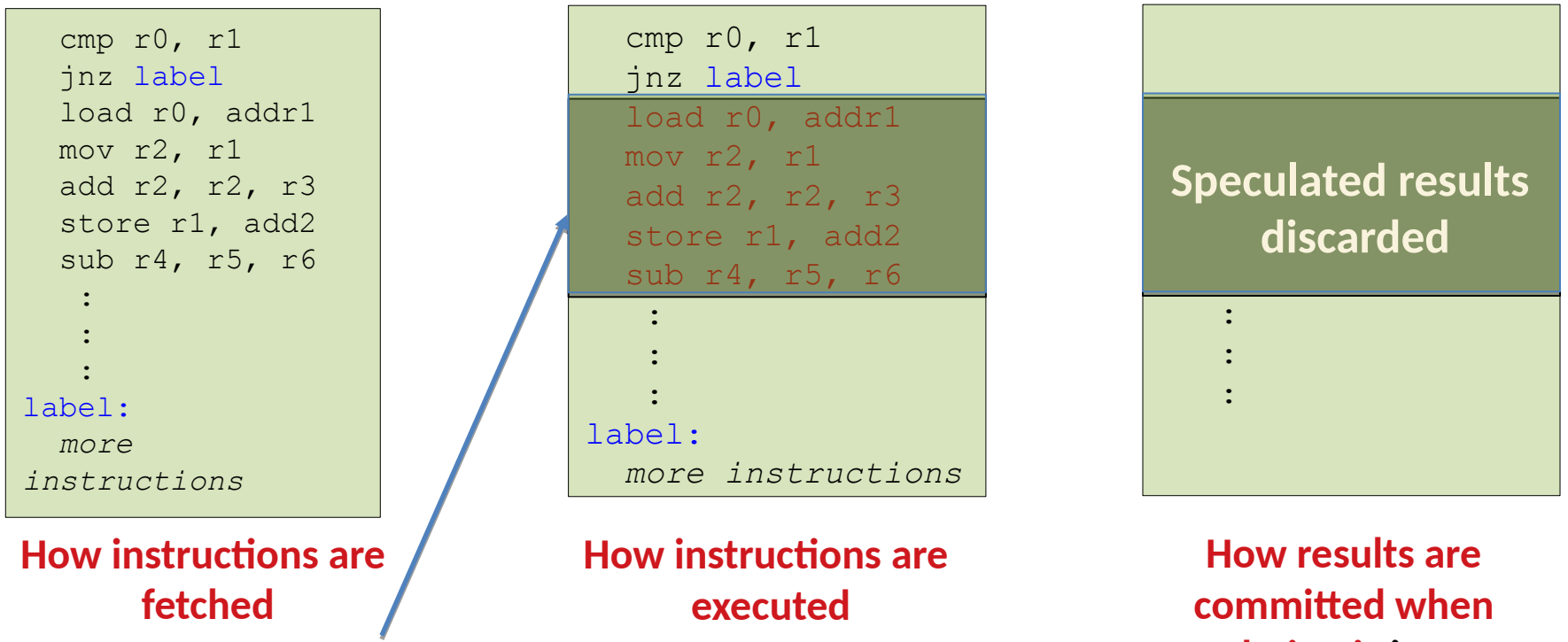




Speculation Attacks : Spectre

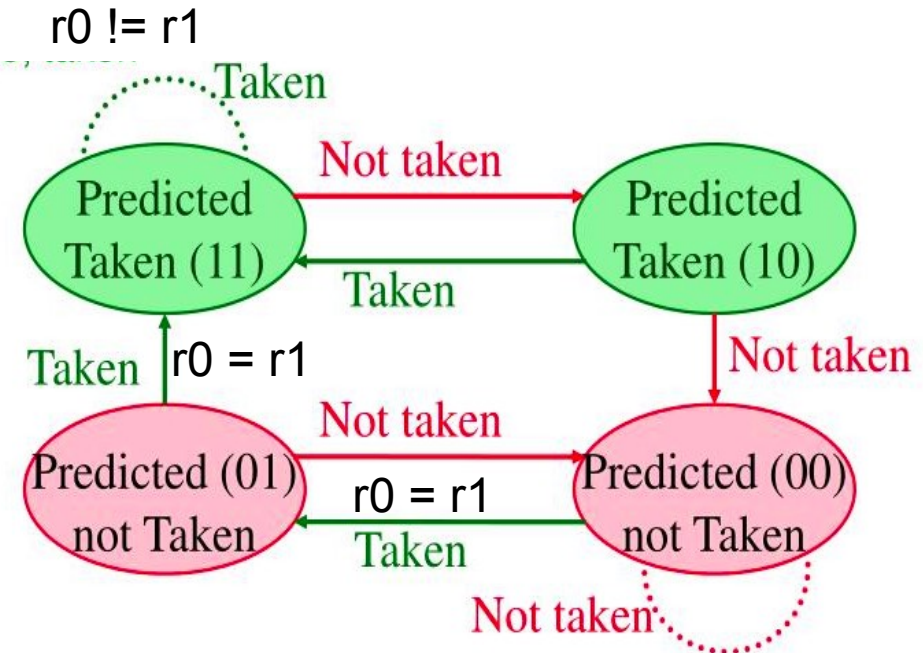
Yuval Yarom’s talk on Meltdown and Spectre at the Cyber security research bootcamp 2018

Speculative Execution : Case 1

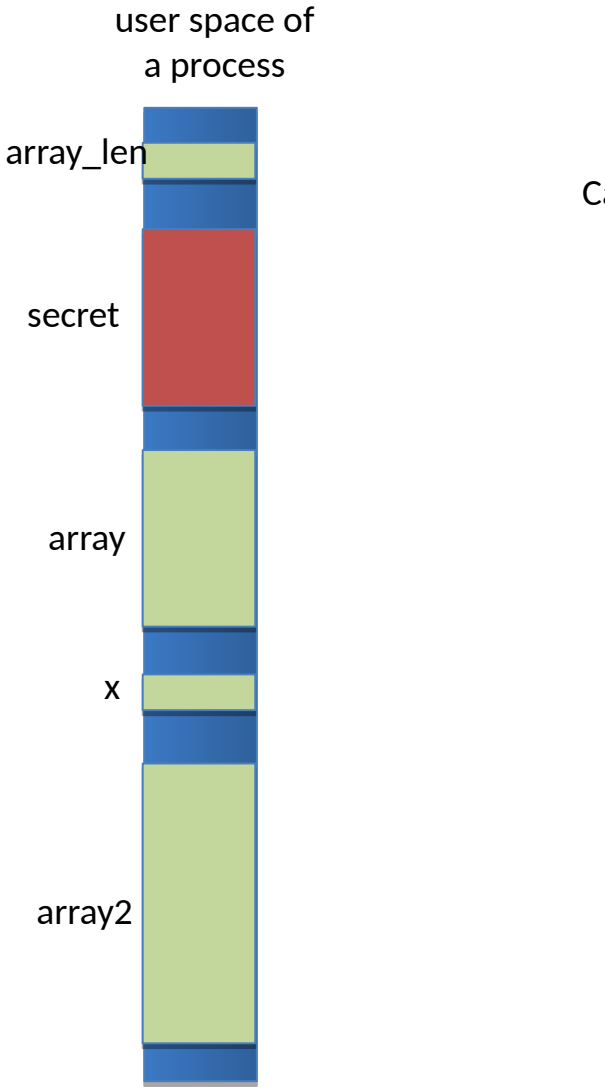


Branch Prediction

```
   cmp r0, r1
   jnz label
   load r0, addr1
   mov r2, r1
   add r2, r2, r3
   store r1, add2
   sub r4, r5, r6
    :
    :
    :
label:
   more instructions
```

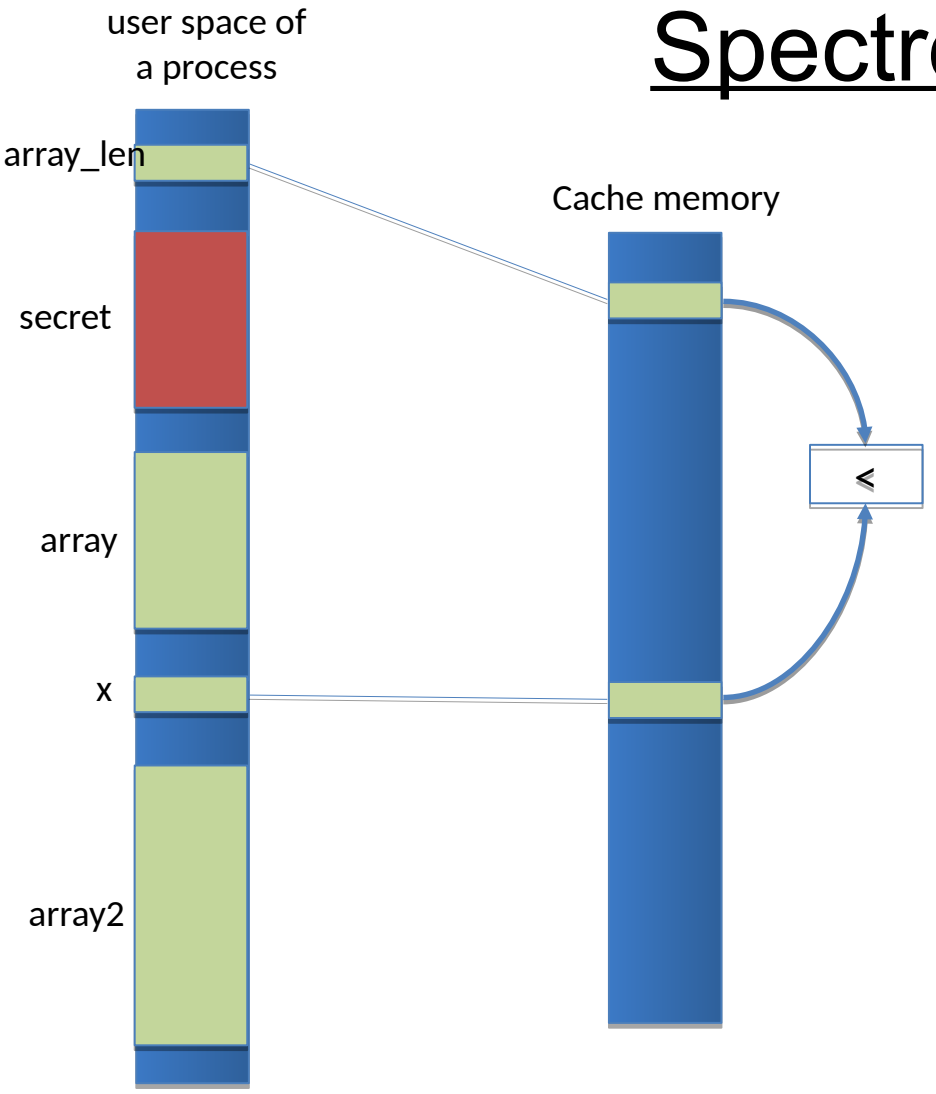


Spectre (Variant 1)



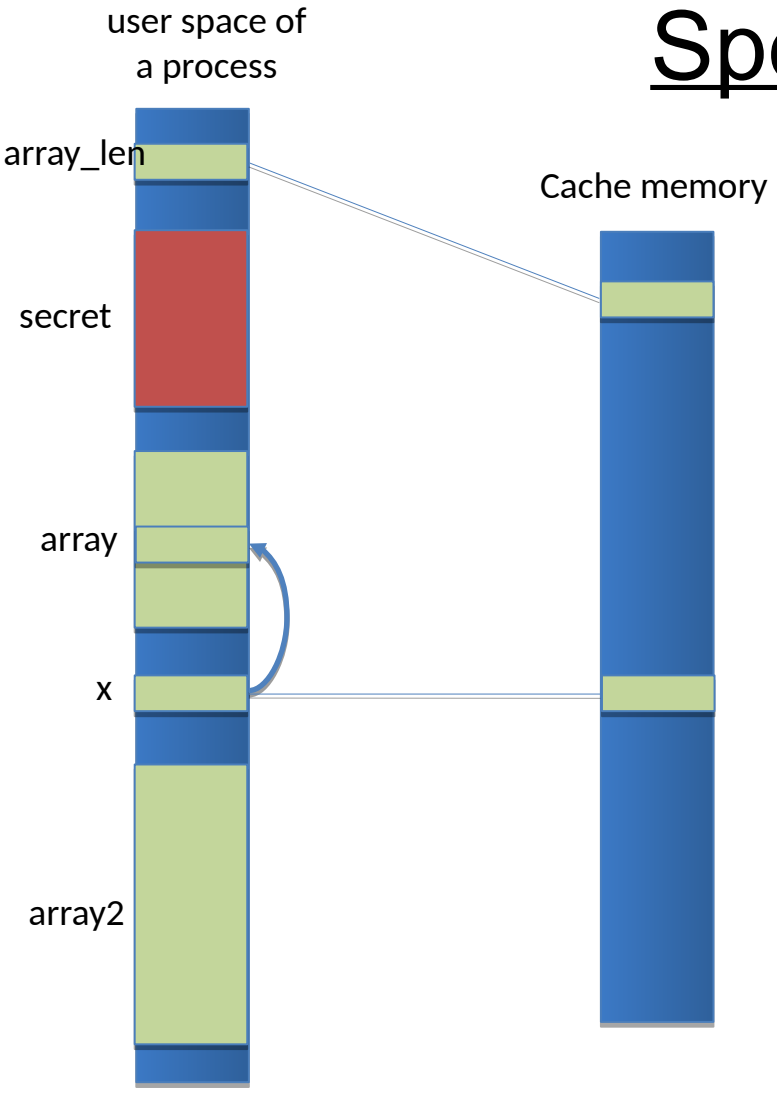

```
if (x < array_len){
    i = array[x];
    y = array2[i * 256];
}
```

Spectre (Variant 1)




```
if (x < array_len){
     i = array[x];
     y = array2[i * 256];
}
```

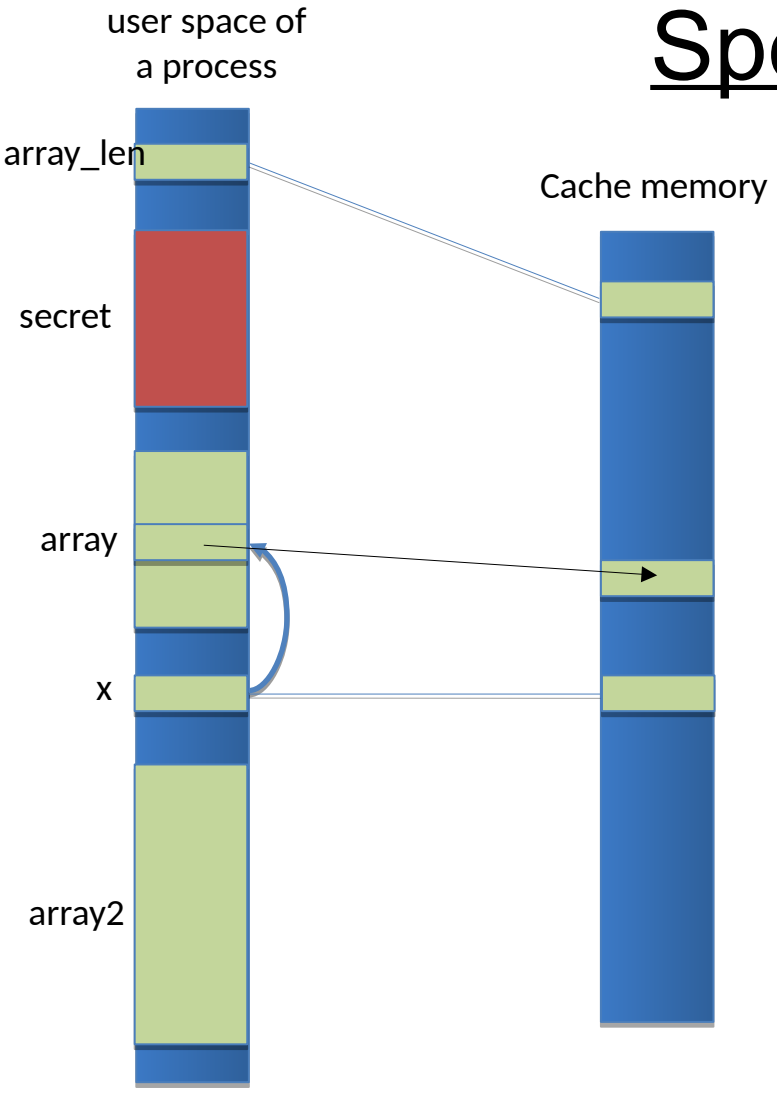
Spectre (Variant 1)



Normal Behavior

```
if (x < array_len){
    i = array[x];
    y = array2[i * 256];
}
```

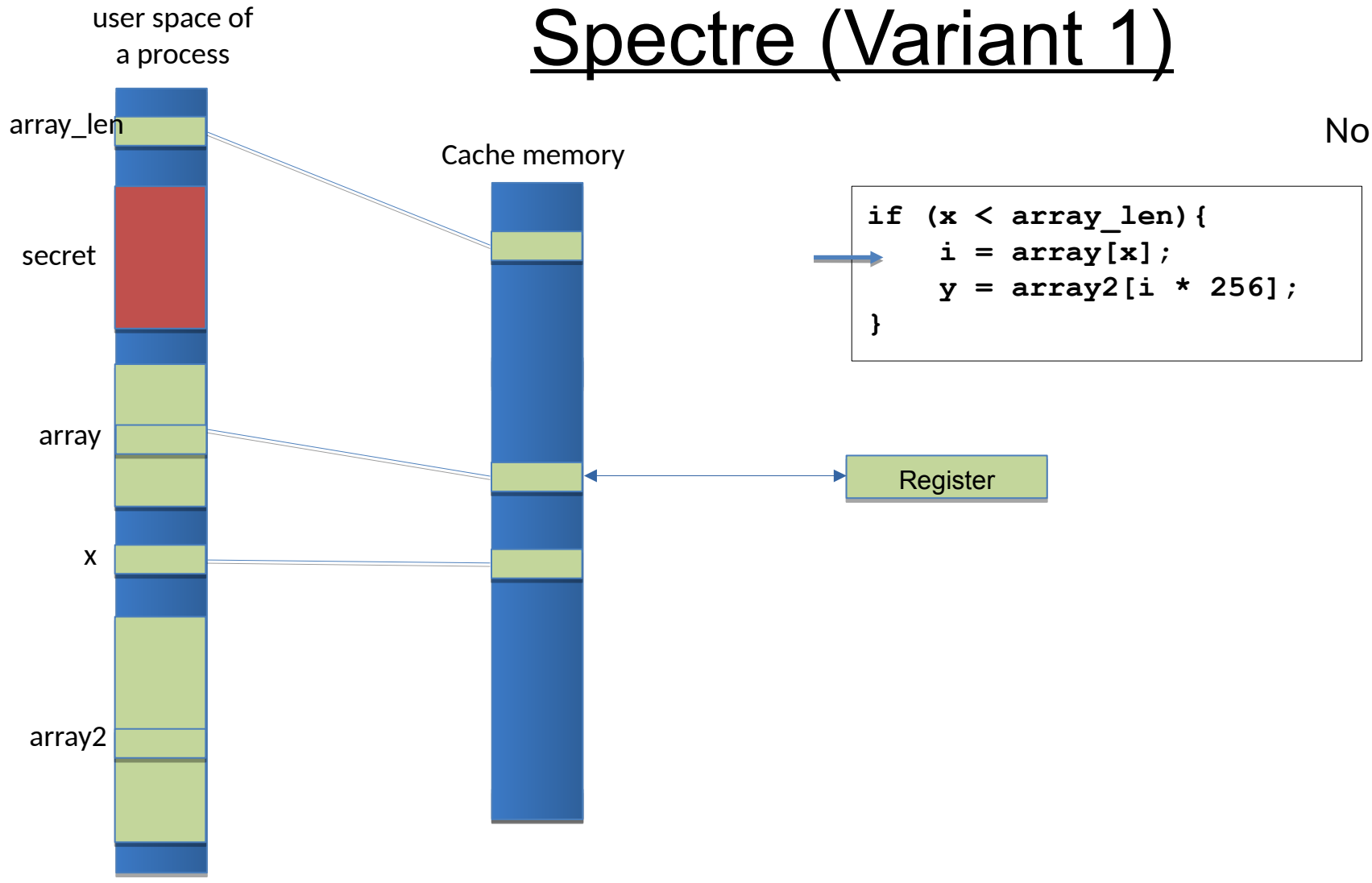
Spectre (Variant 1)



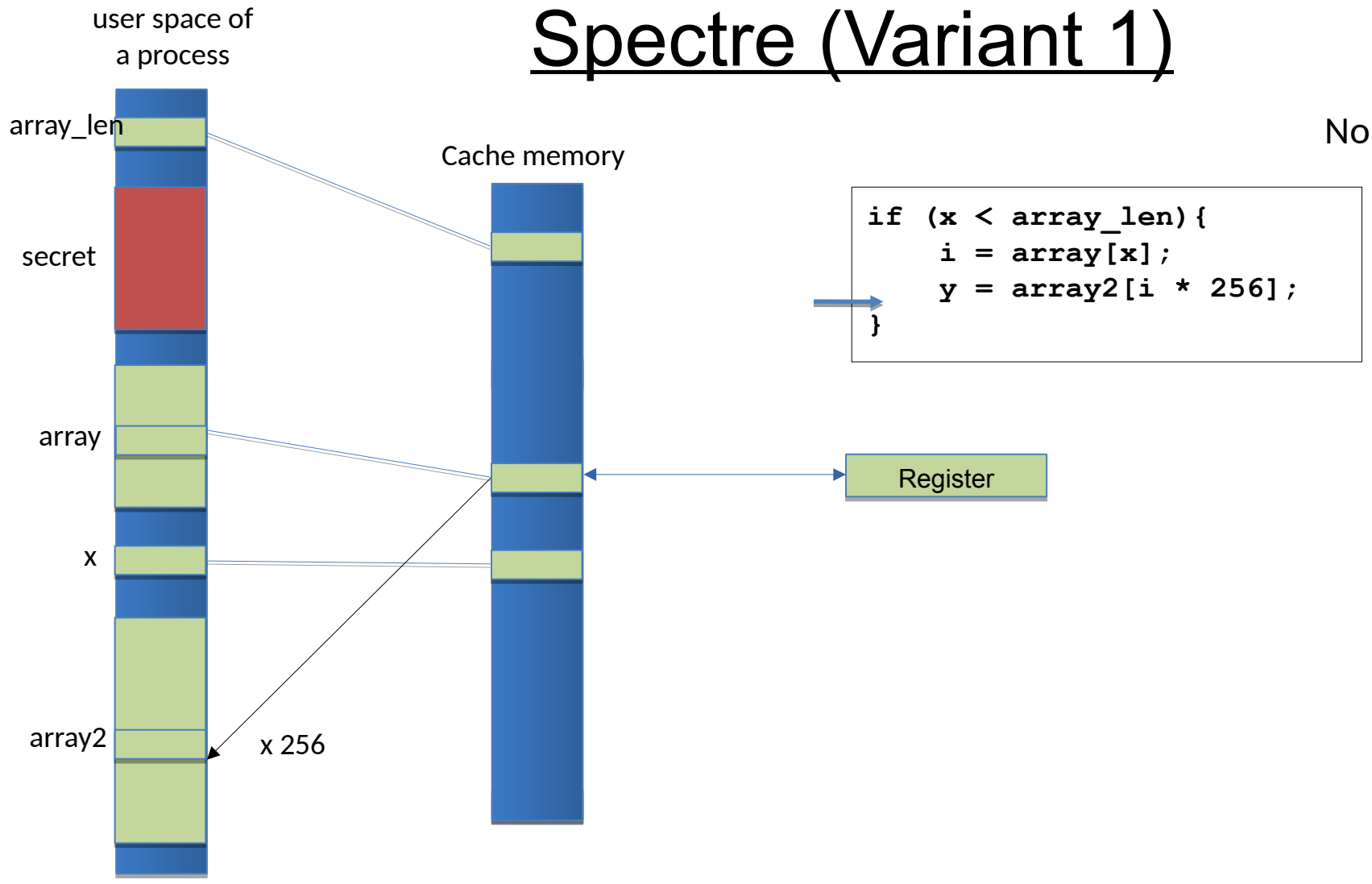
Normal Behavior

```
if (x < array_len){
    i = array[x];
    y = array2[i * 256];
}
```

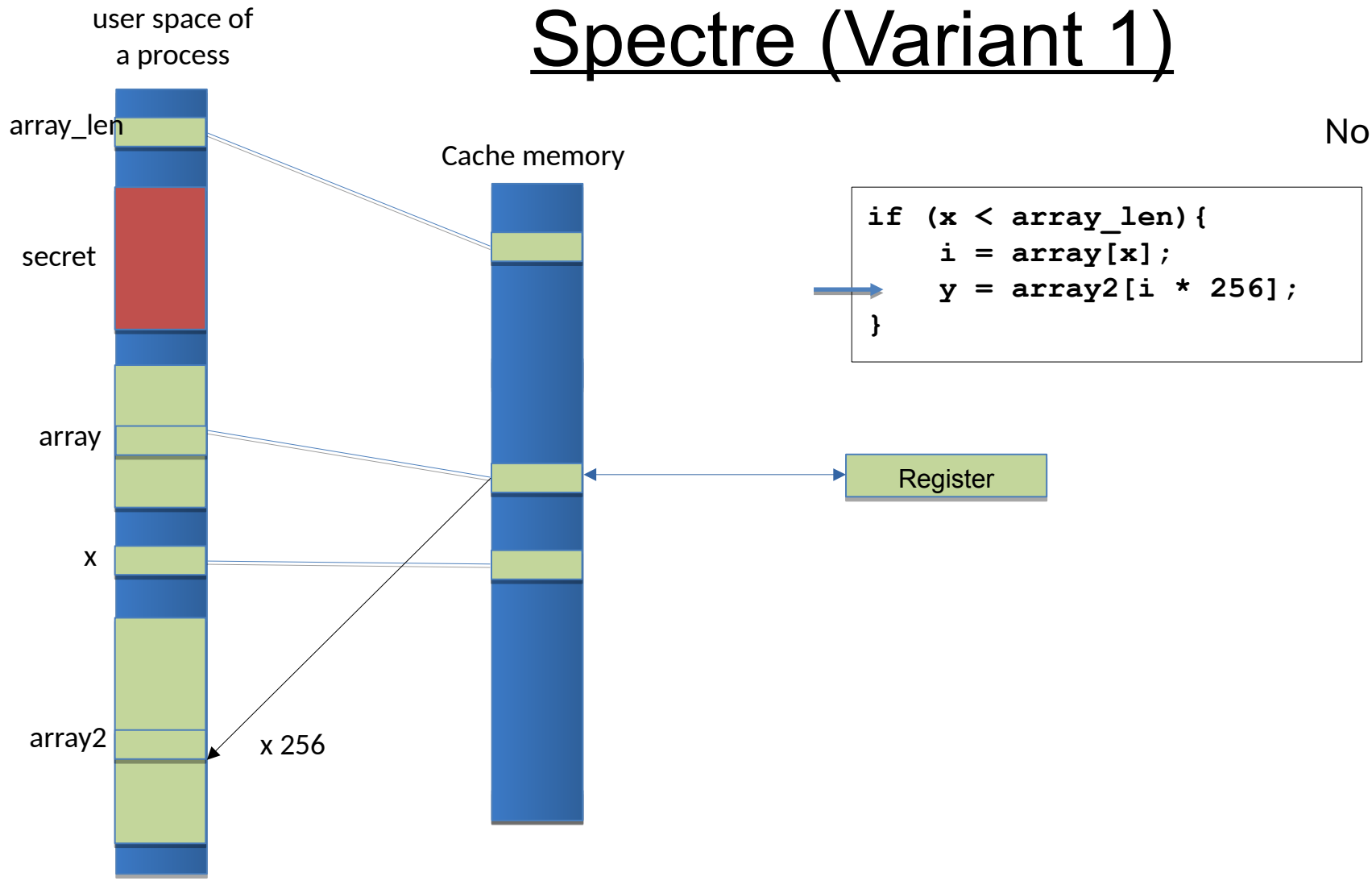
Spectre (Variant 1)



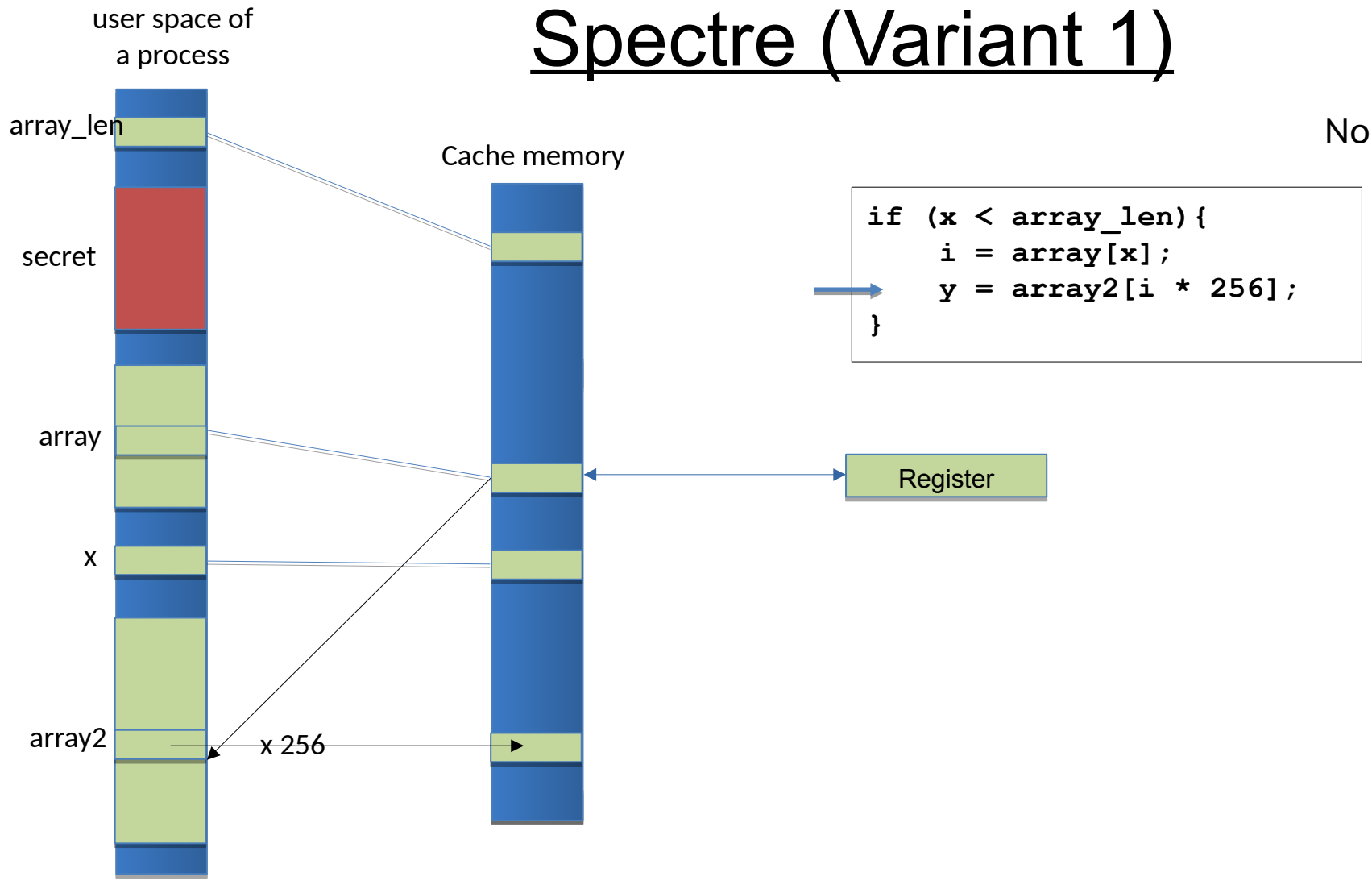
Spectre (Variant 1)



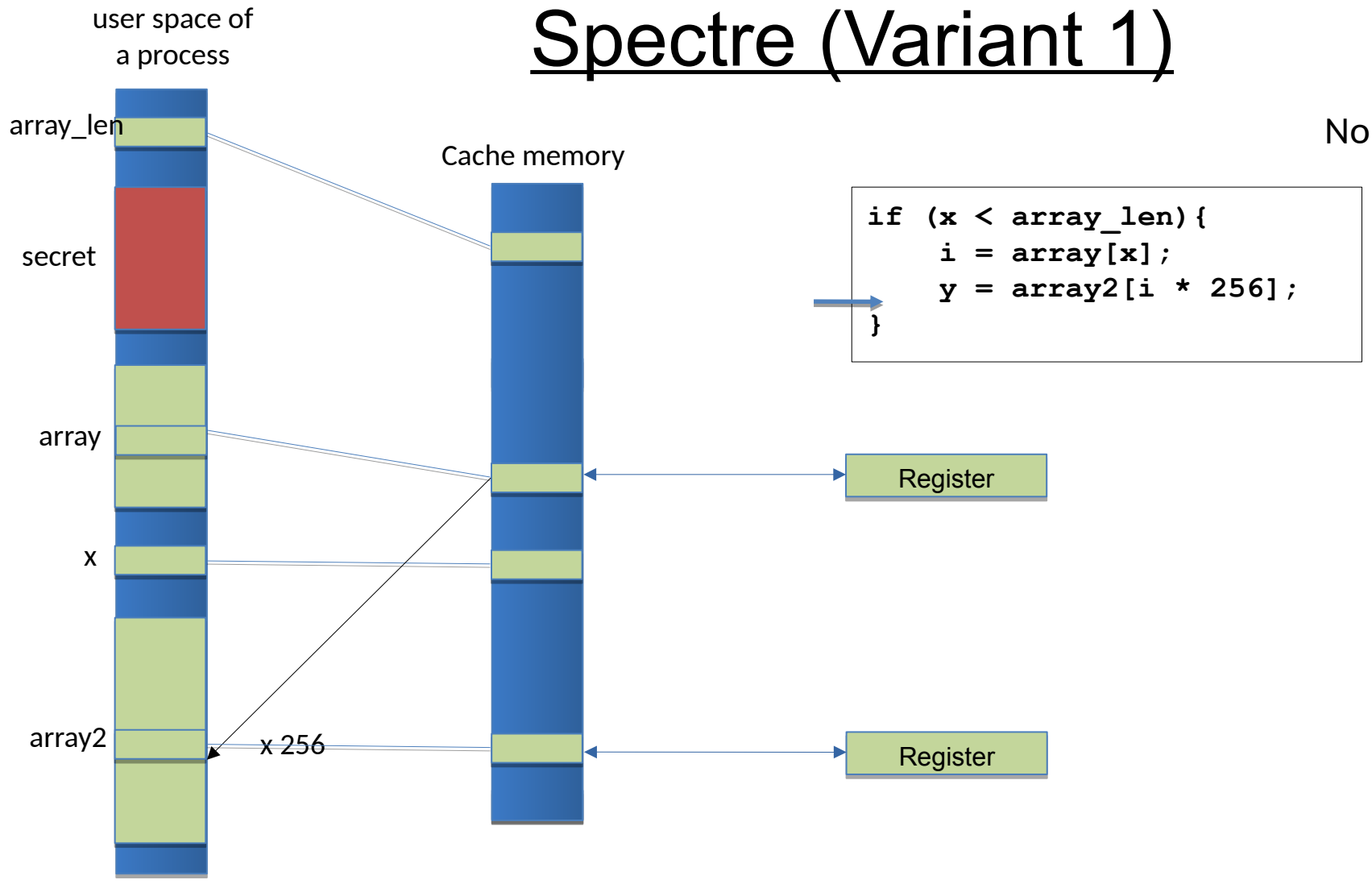
Spectre (Variant 1)



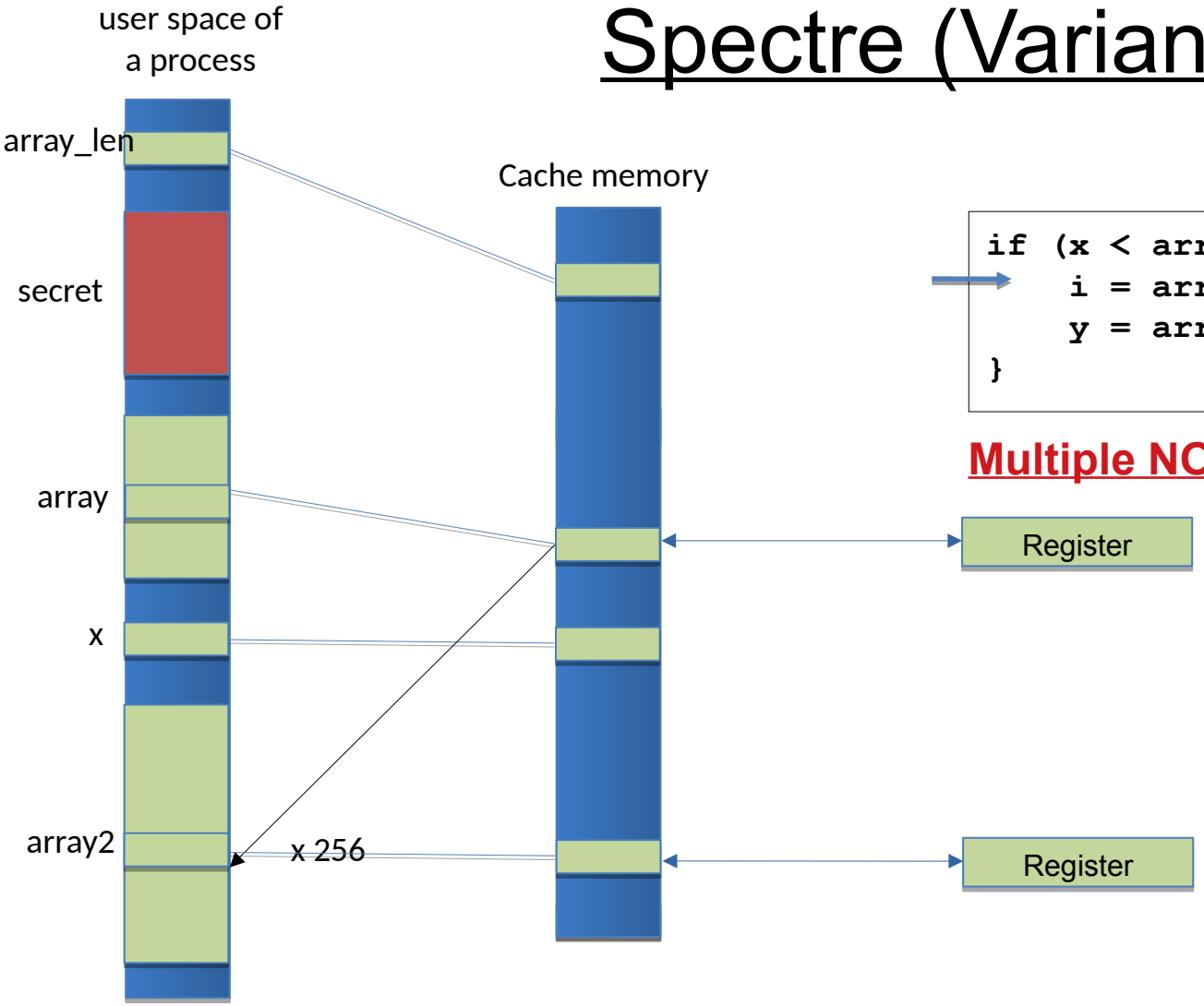
Spectre (Variant 1)



Spectre (Variant 1)



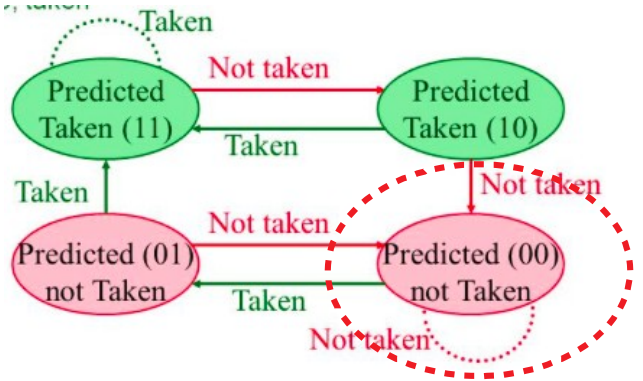
Spectre (Variant 1)



Normal Behavior

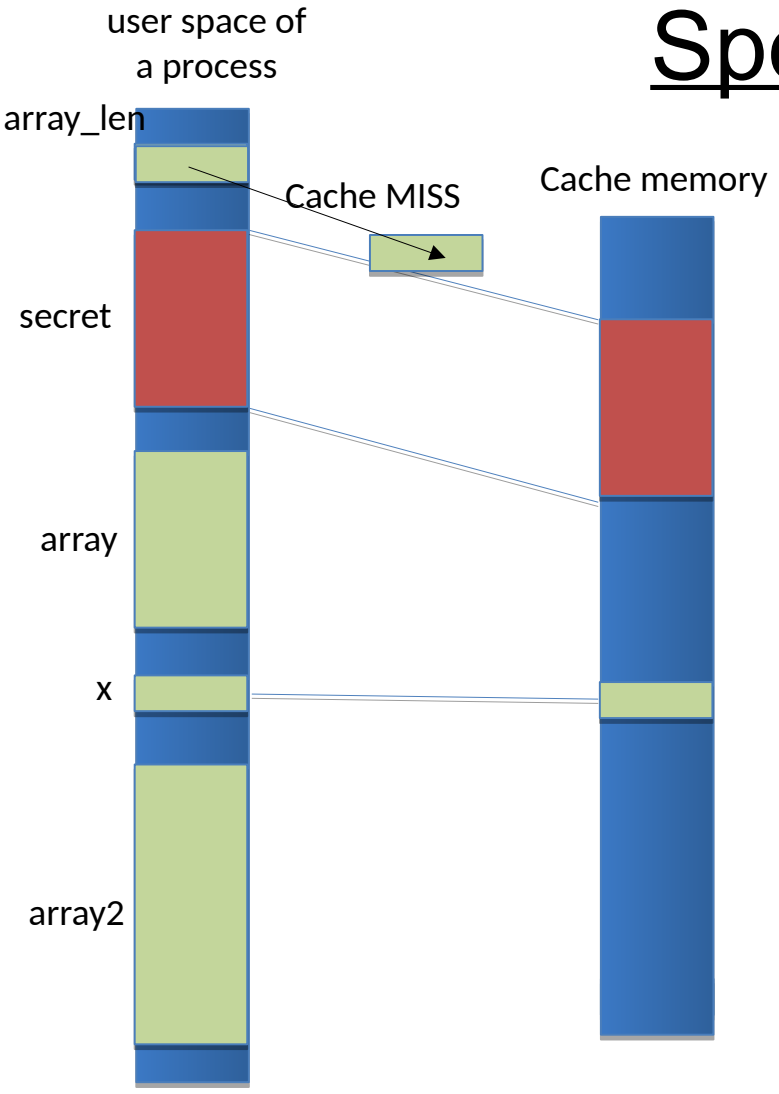
```
if (x < array_len){
    i = array[x];
    y = array2[i * 256];
}
```

Multiple NOT TAKEN Loops



Branch NOT TAKEN = TRUE if Condition

Spectre (Variant 1)

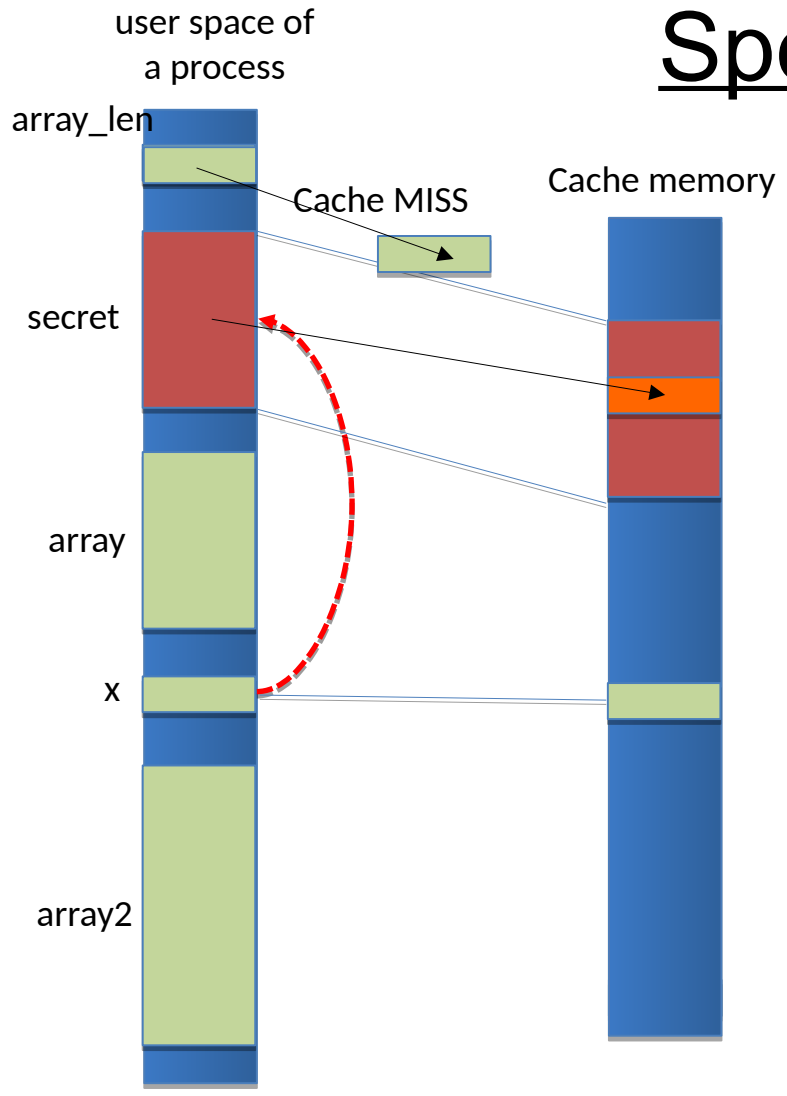


Under Attack

```
if (x < array_len){
     i = array[x];
     y = array2[i * 256];
}
```

- x > array_len
- array_len not in cache
- secret in cache memory

Spectre (Variant 1)

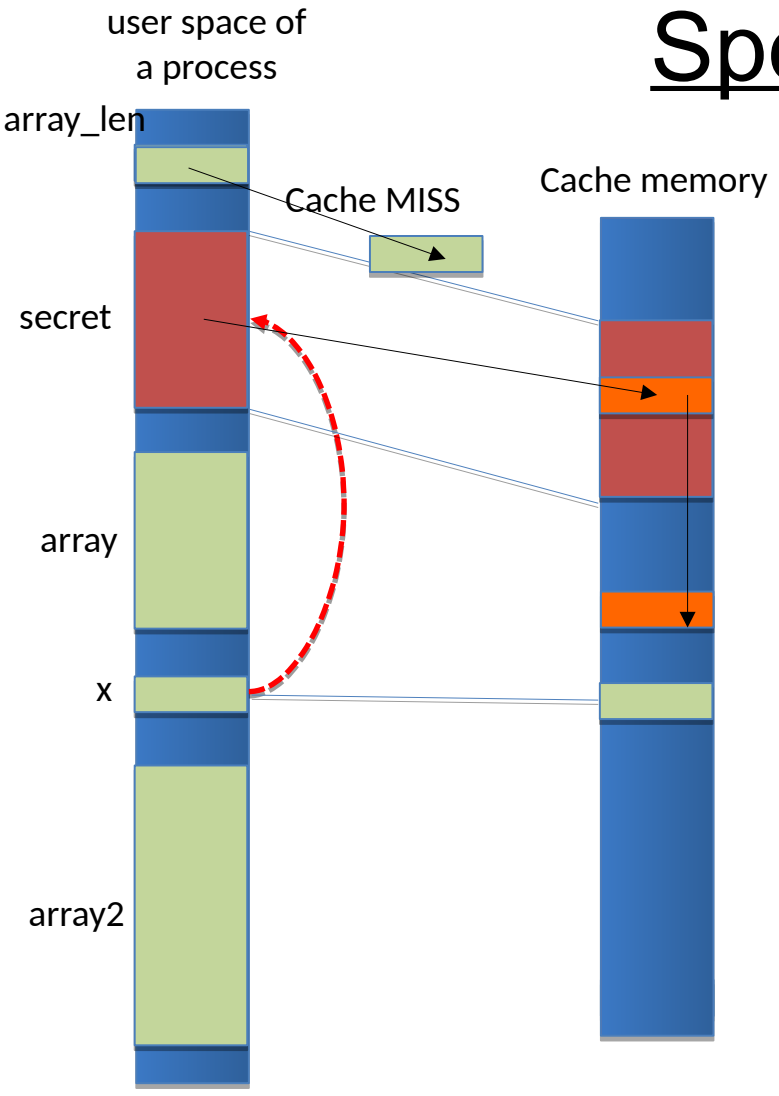


Under Attack

```
if (x < array_len){
    i = array[x];
    y = array2[i * 256];
}
```

- x > array_len
- array_len not in cache
- secret in cache memory

Spectre (Variant 1)

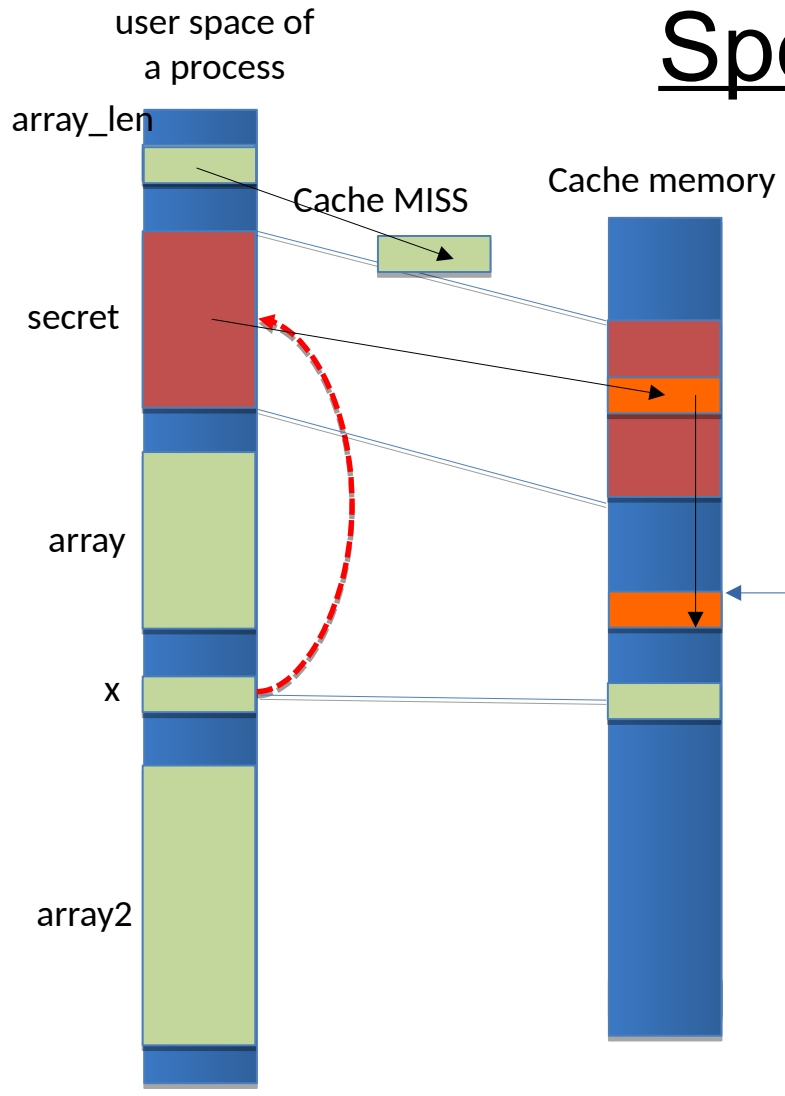


Under Attack

```
if (x < array_len){
    i = array[x];
    y = array2[i * 256];
}
```

- x > array_len
- array_len not in cache
- secret in cache memory

Spectre (Variant 1)

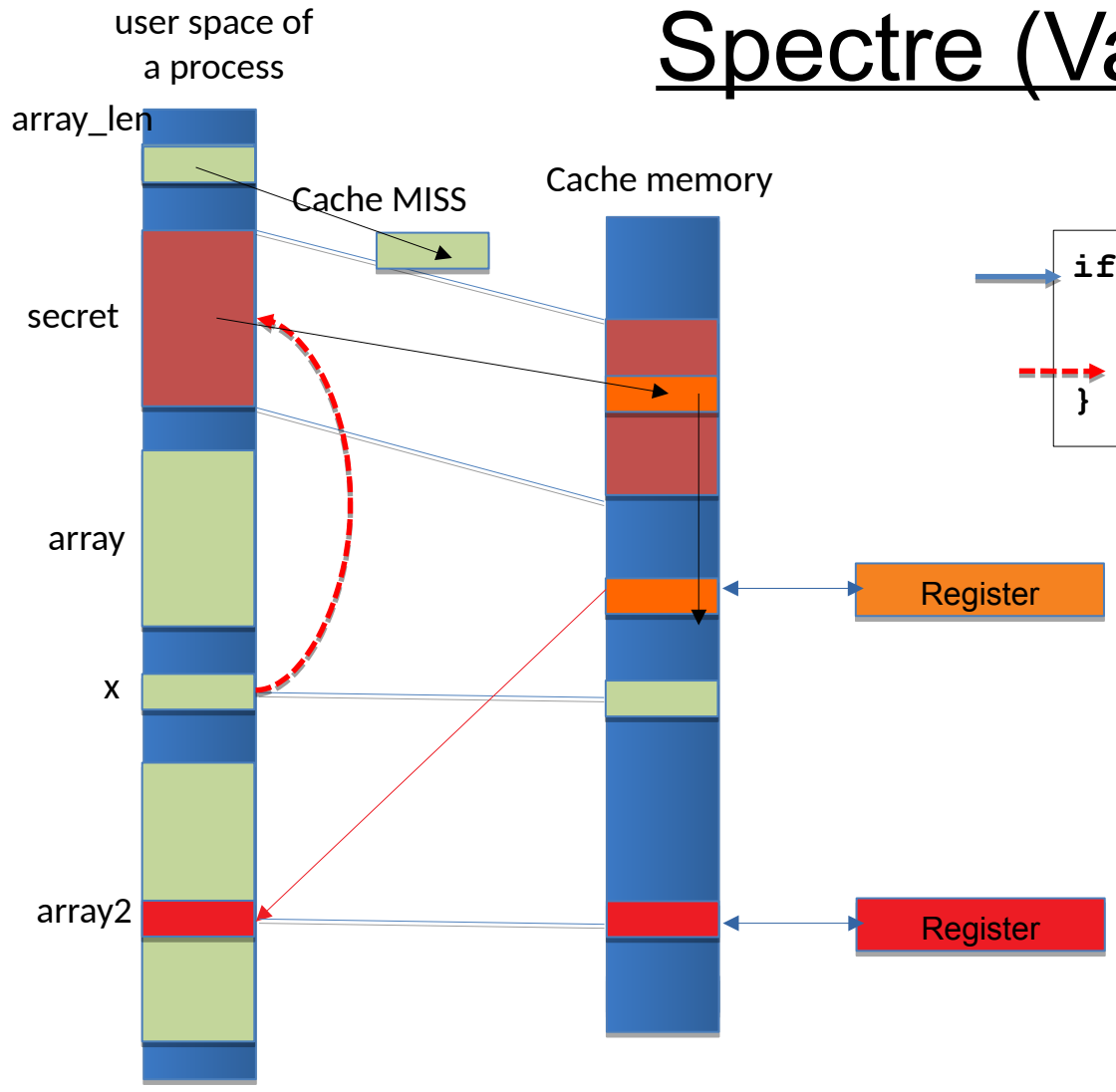


Under Attack

```
if (x < array_len){
    i = array[x];
    y = array2[i * 256];
}
```

- x > array_len
- array_len not in cache
- secret in cache memory

Spectre (Variant 1)

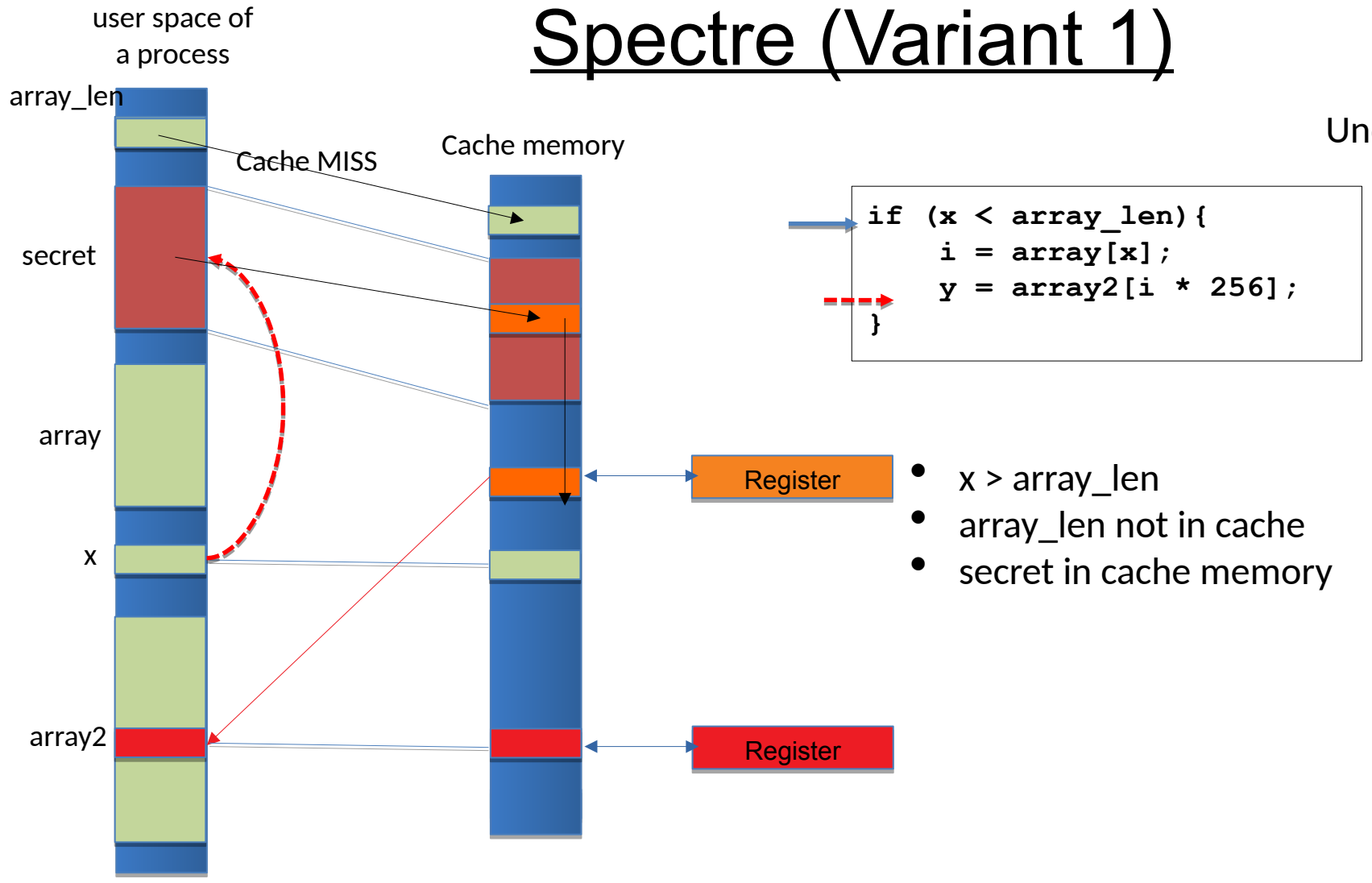


Under Attack

```
if (x < array_len){
    i = array[x];
    y = array2[i * 256];
}
```

- x > array_len
- array_len not in cache
- secret in cache memory

Spectre (Variant 1)

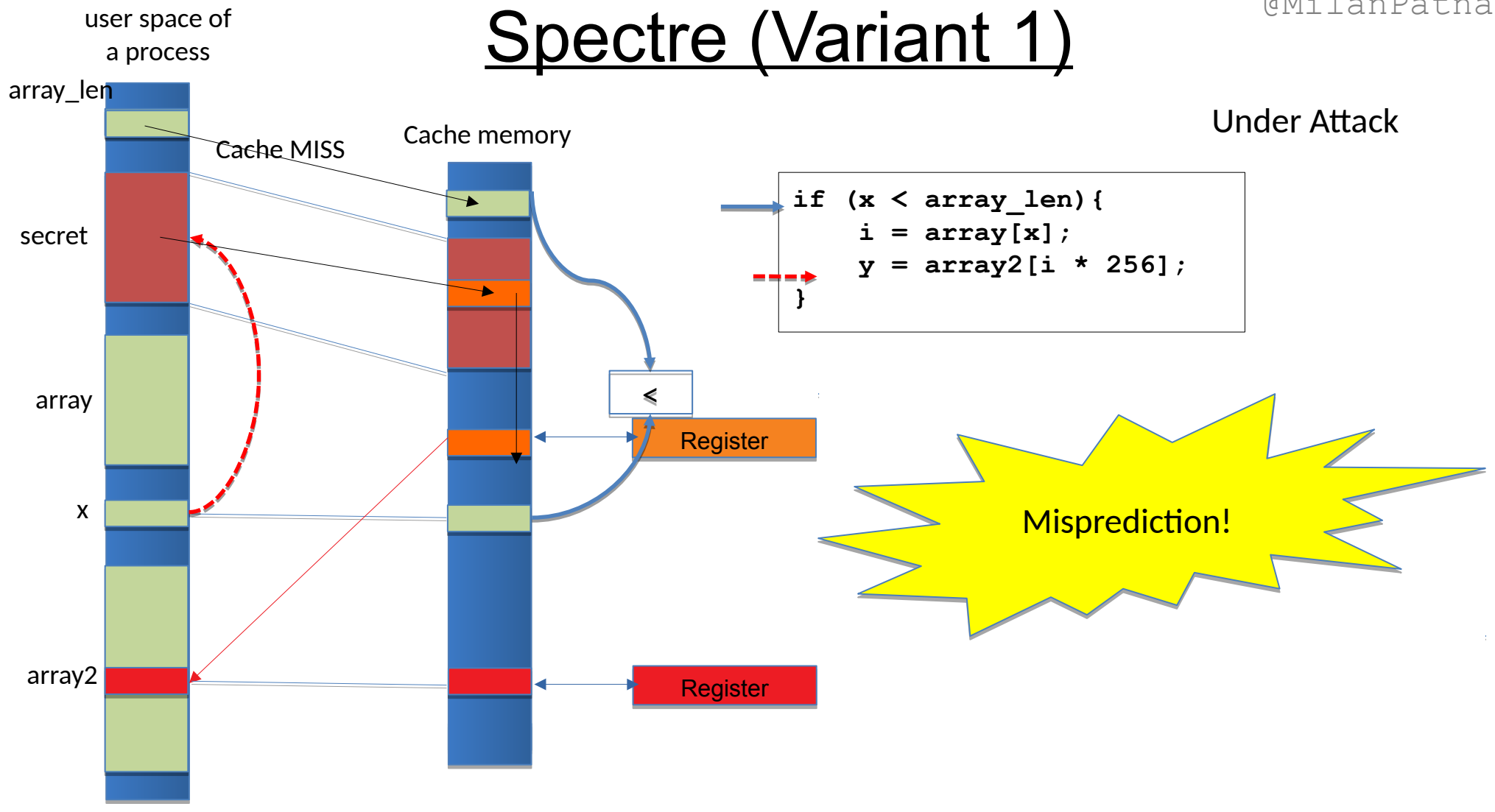


Under Attack

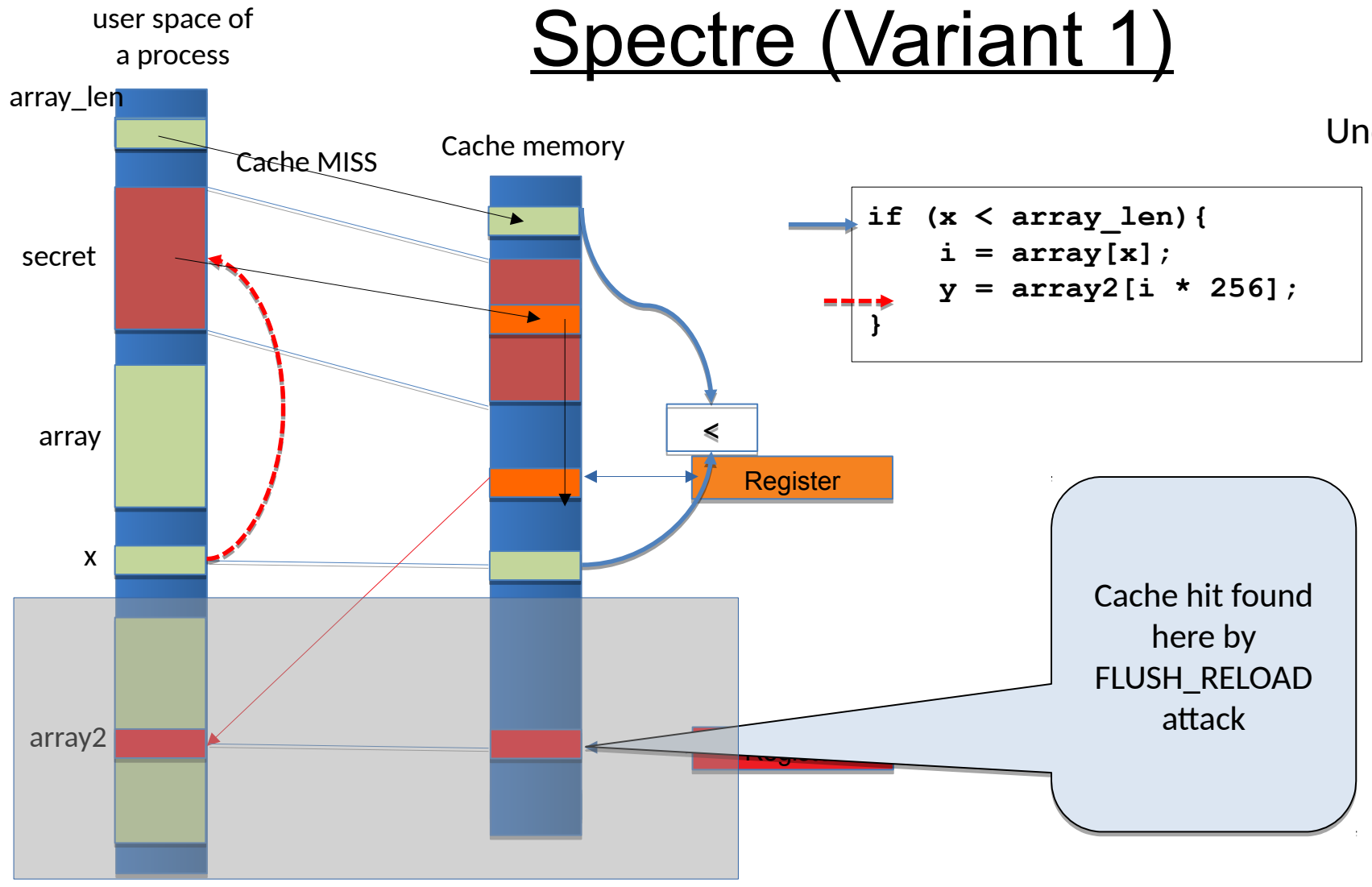
```
if (x < array_len){
    i = array[x];
    y = array2[i * 256];
}
```

- x > array_len
- array_len not in cache
- secret in cache memory

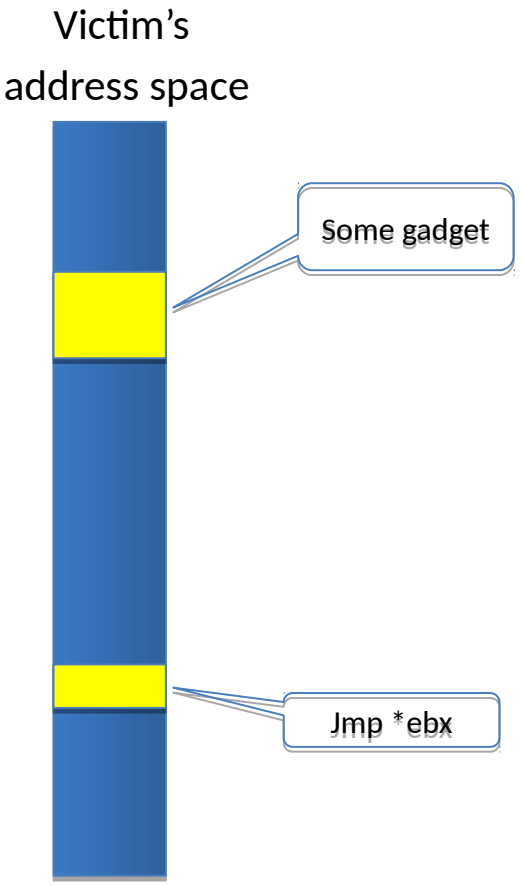
Spectre (Variant 1)



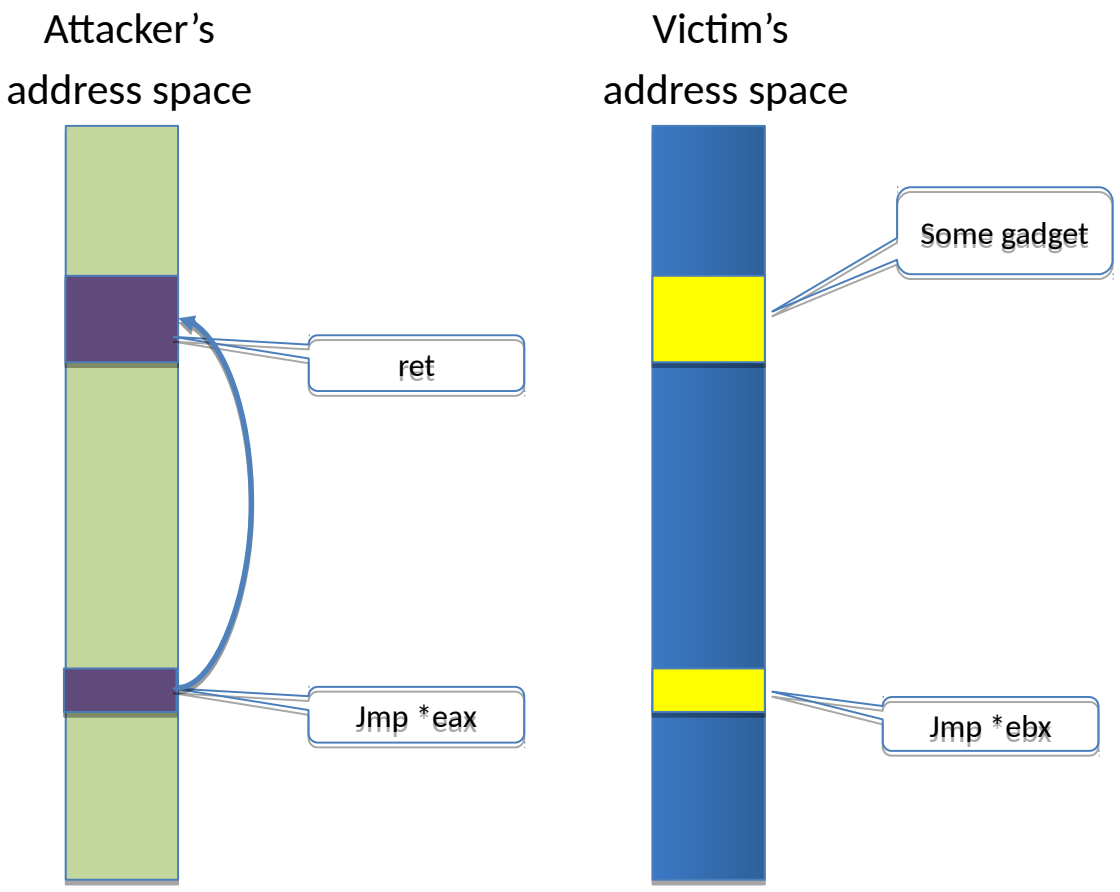
Spectre (Variant 1)



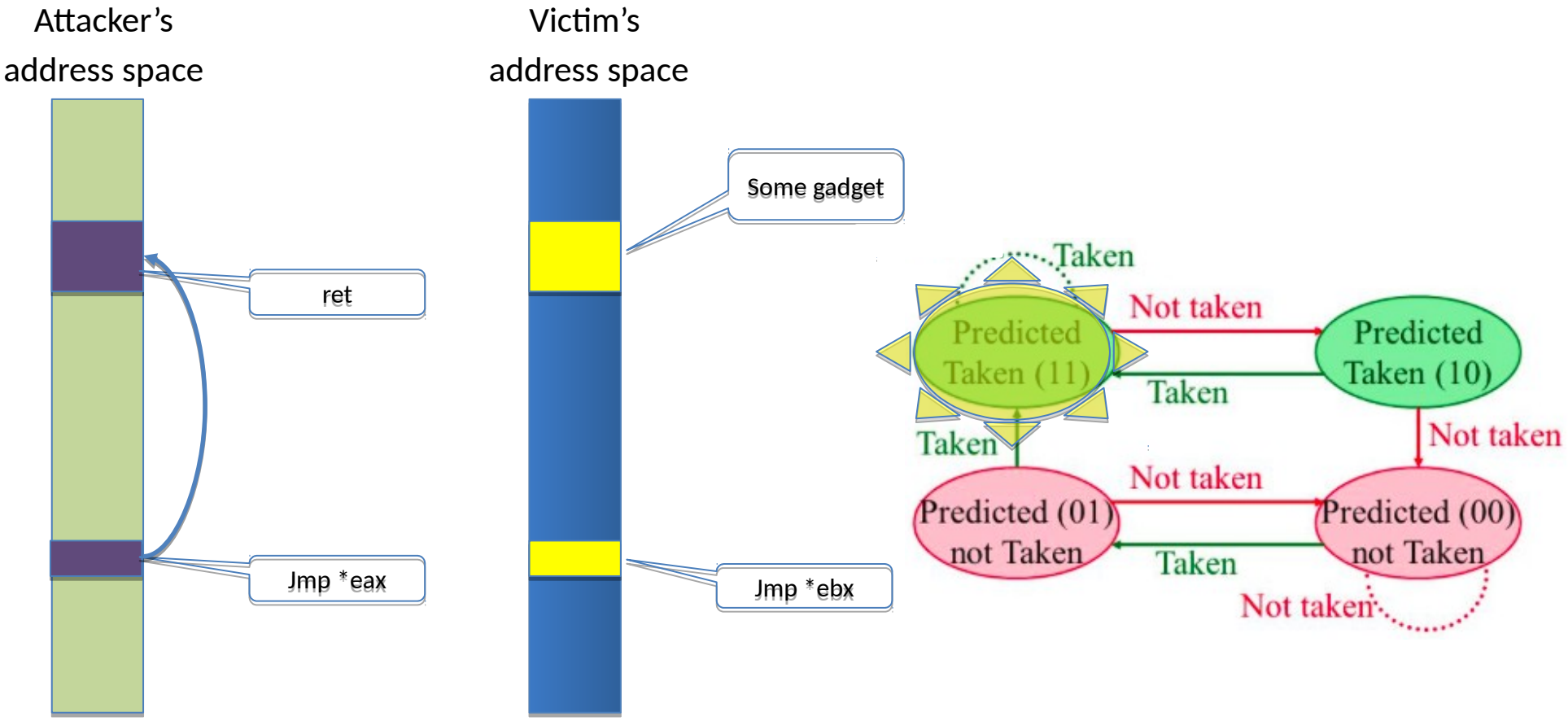
Spectre (Variant 2)



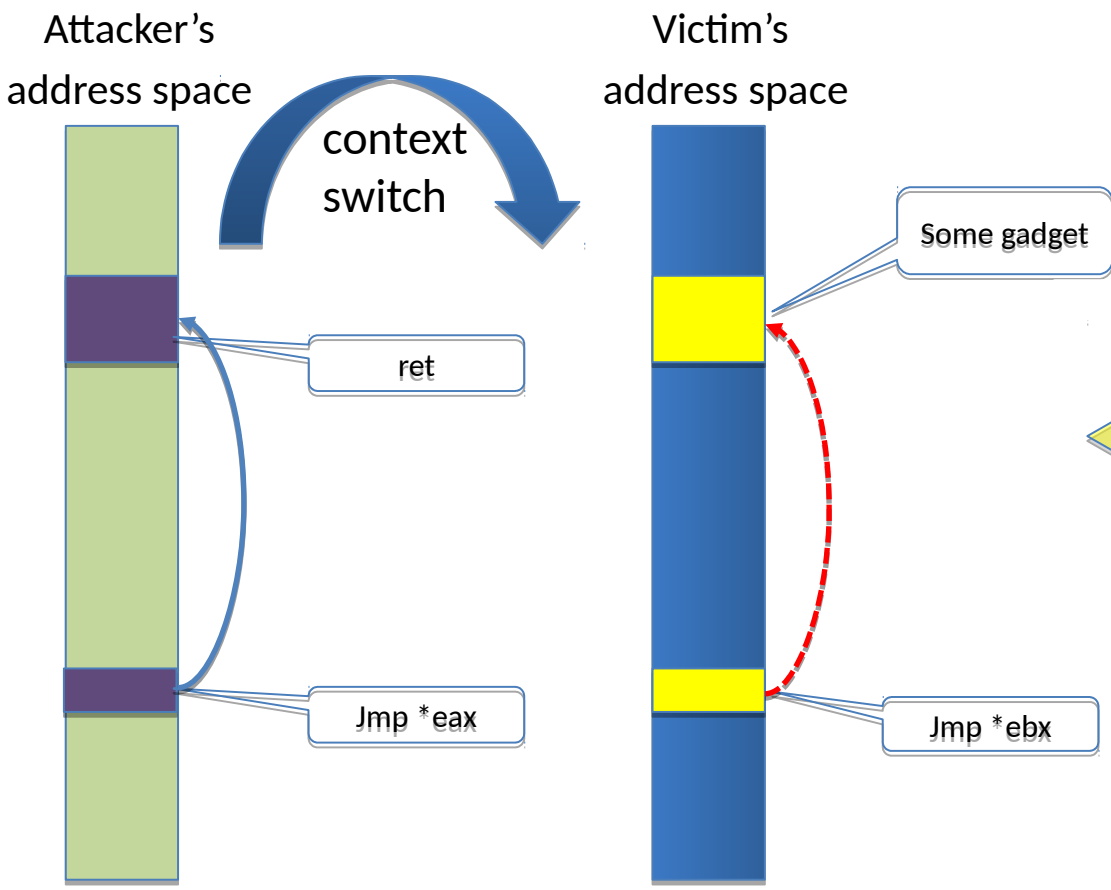
Spectre (Variant 2)

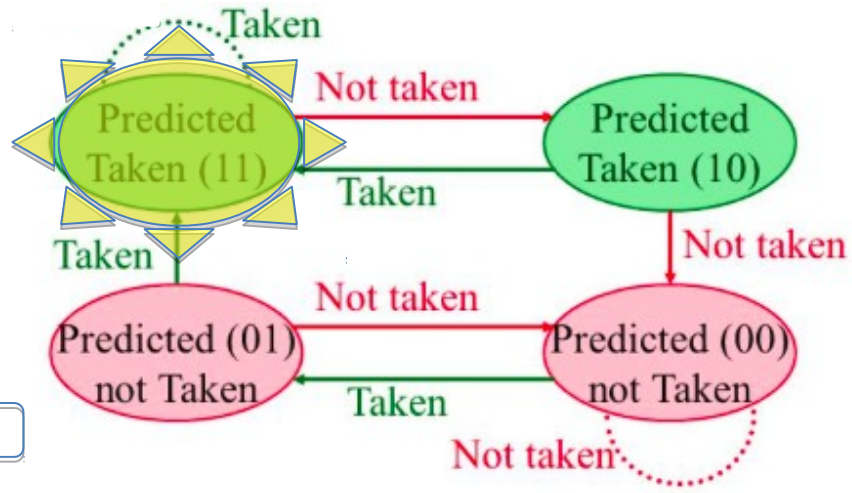


Spectre (Variant 2)



Spectre (Variant 2)





Questions

Cache Attacks