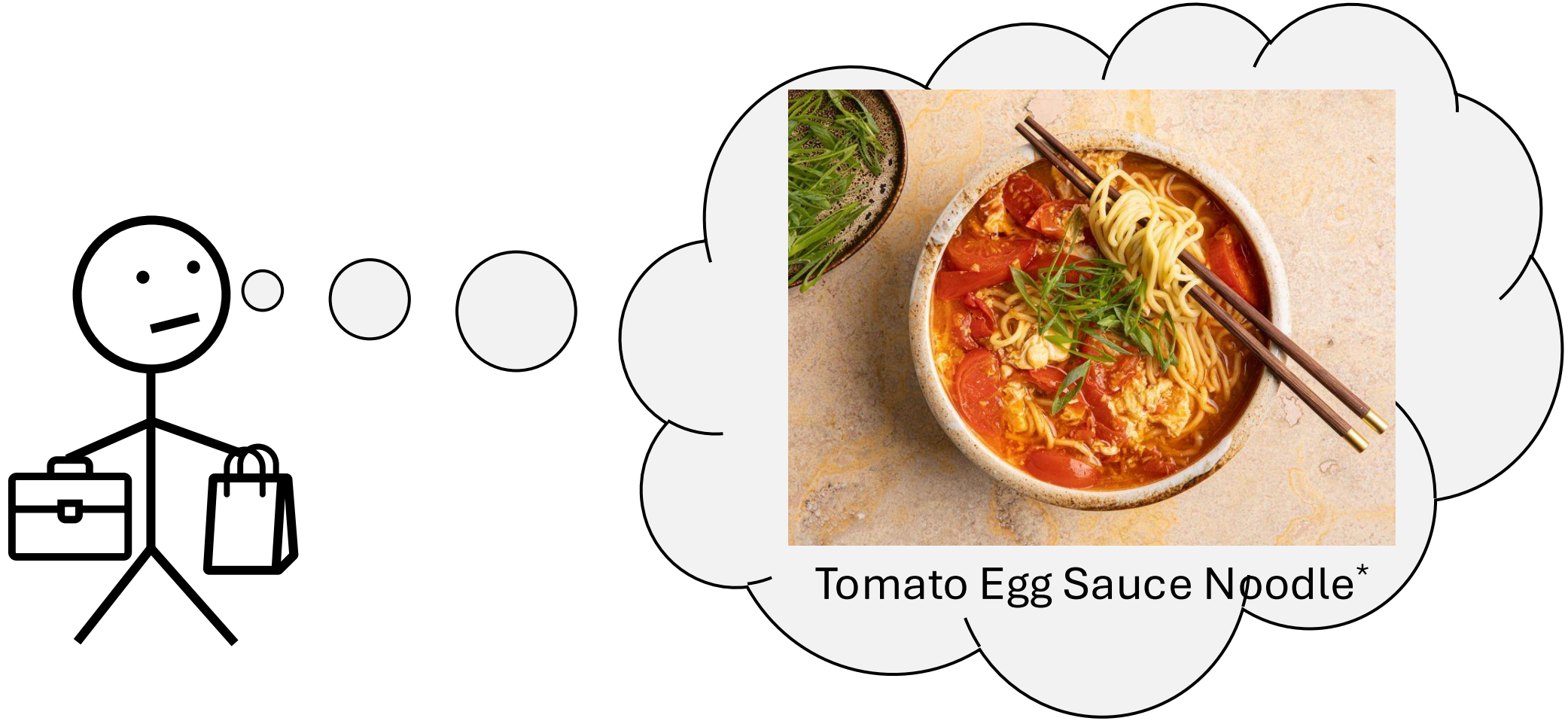
ECE 382N-Sec (FA26):

L4: Transient-Execution Attacks

Neil Zhao

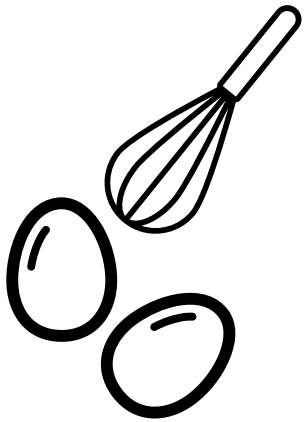
neil.zhao@utexas.edu

Let’s Talk about Cooking Noodles

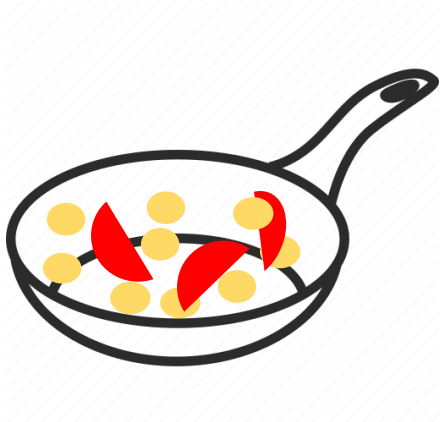


*https://www.kitchenstories.com/en/recipes/tomato-and-egg-noodle-soup

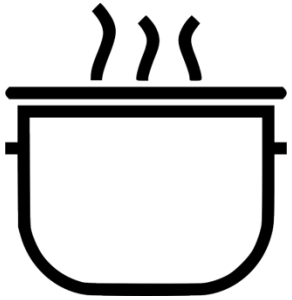
Let’s Talk about Cooking Noodles



Beat two eggs
(2 min)



Make scrambled eggs and mix in canned tomato
(6 min)



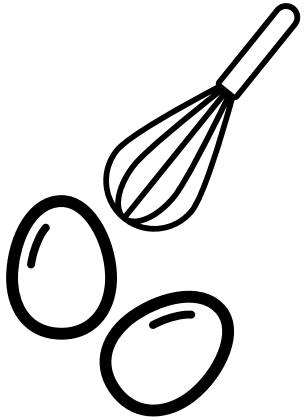
Boil noodles
(6 min)

Mix everything and enjoy ☺!

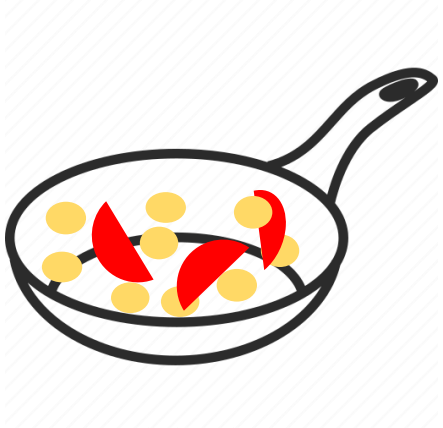
Total preparation time: 14 minutes

Can we be faster?

Let's Talk about Cooking Noodles



Beat two eggs
(2 min)



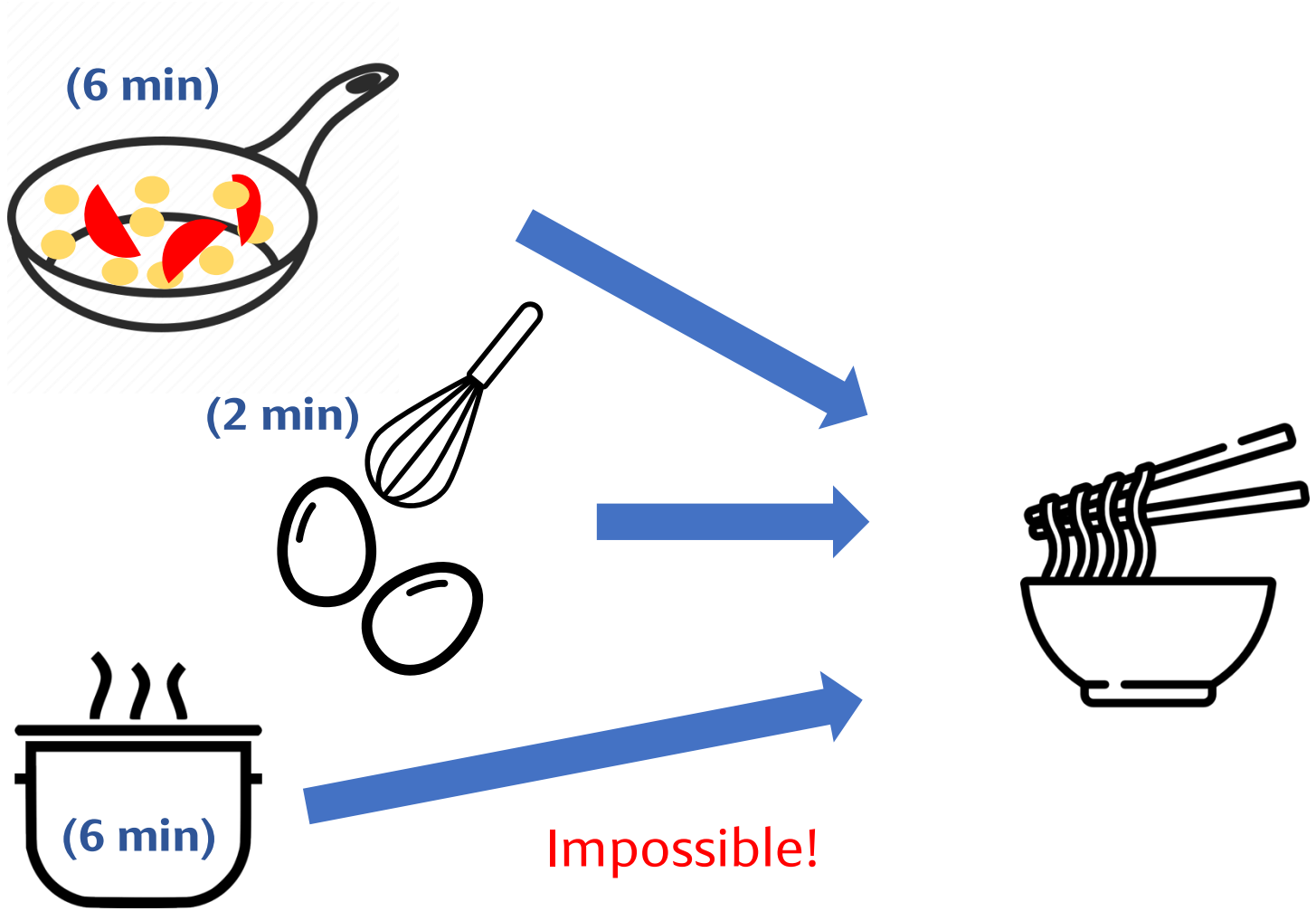
Make scrambled eggs and
mix in canned tomato
(6 min)

Can we be even faster?

Boil noodles
(6 min)

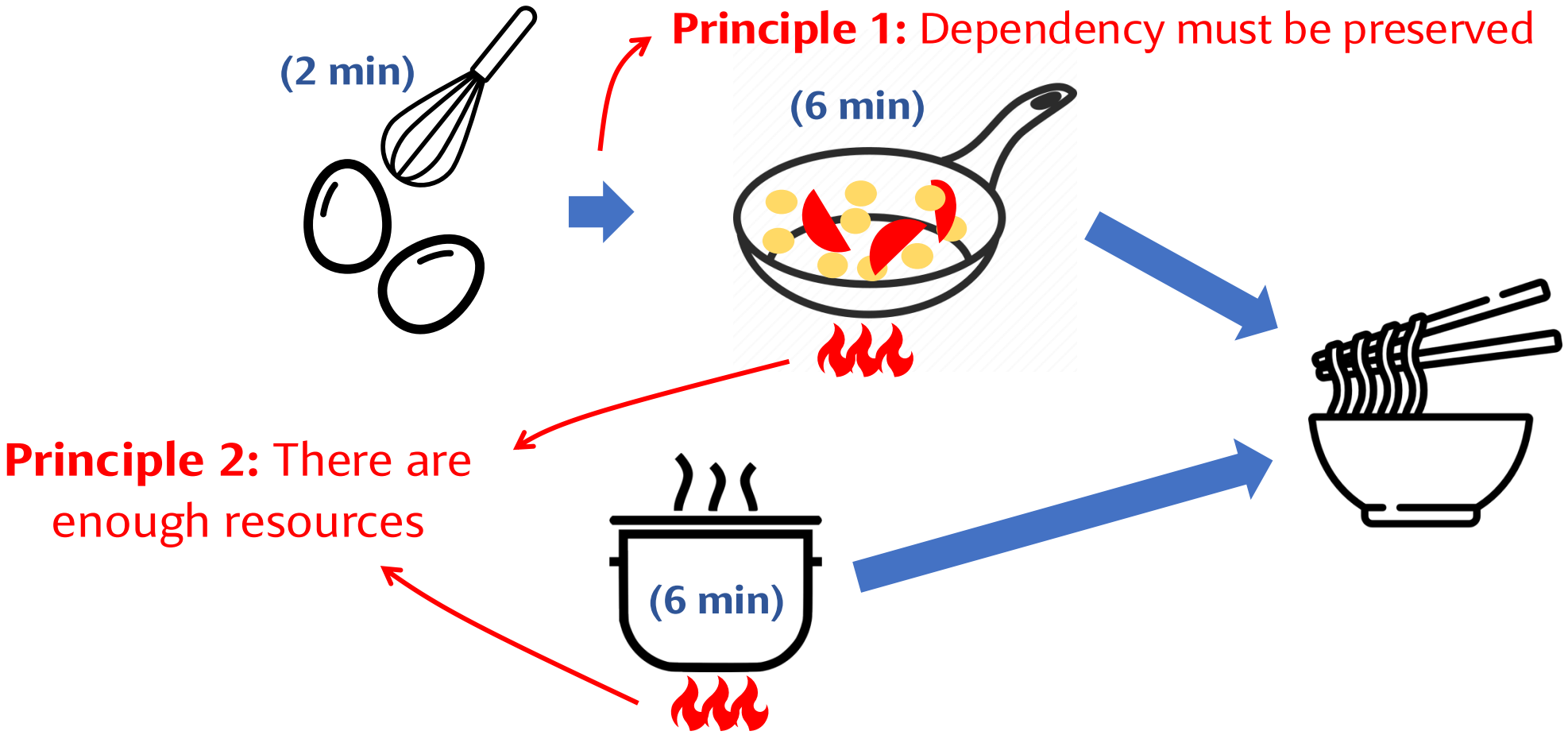
Total preparation time: 8 minutes

Let's Talk about Cooking Noodles



Total preparation time: 6 minutes?

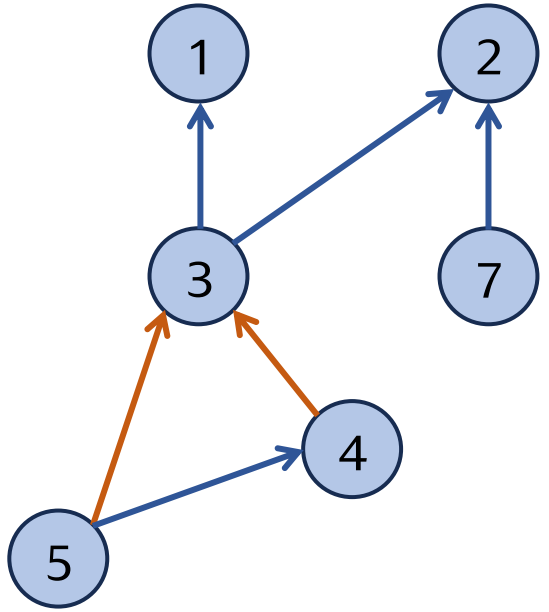
Let's Talk about Cooking Noodles



Out-of-Order Execution & Speculation

```
1.  a = 10;
2.  d = 2;
3.  if (a < d) {
4.    b = 4;
5.    c = b * 2;
6.  }
7.  e = d / 2;
```

Program Dependency Graph (PDG)



Data dependence

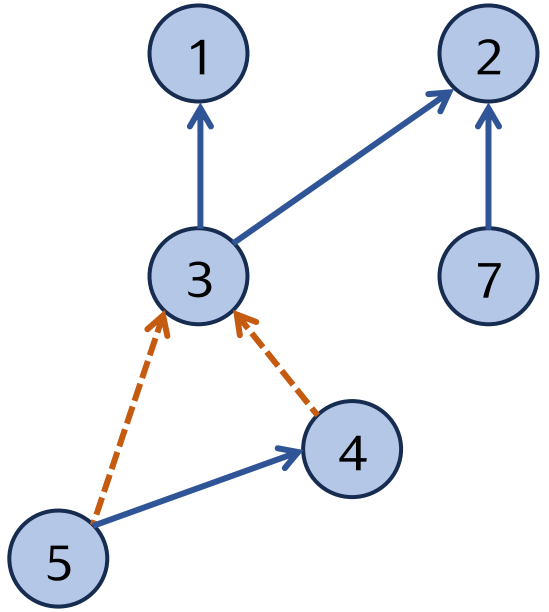
Ctrl. dependence

👎 Parallelism is constrained by the web of dependencies

Out-of-Order Execution & Speculation

```
1.  a = 10;
2.  d = 2;
3.  if (a < d) {
4.    b = 4;
5.    c = b * 2;
6.  }
7.  e = d / 2;
```

Program Dependency Graph (PDG)



Data dependence

Ctrl. dependence

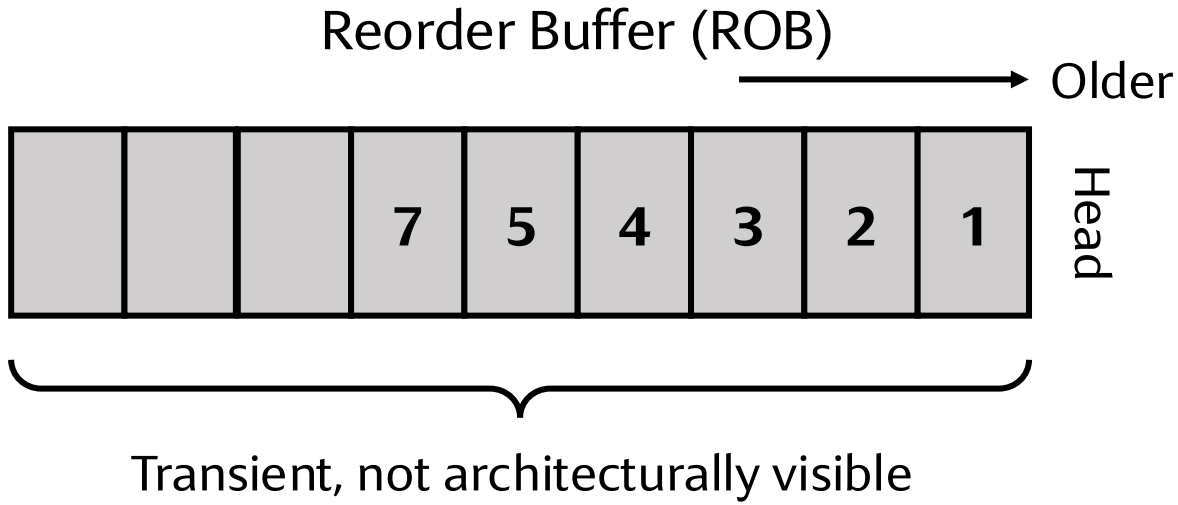
Ctrl. dependence "removed" via prediction

👍 Prediction "removes" dependencies and unlocks parallelism

Out-of-Order Execution & Speculation

Predict taken

```
1.  a = 10;
2.  d = 2;
3.  if (a < d) {
4.    b = 4;
5.    c = b * 2;
6.  }
7.  e = d / 2;
```

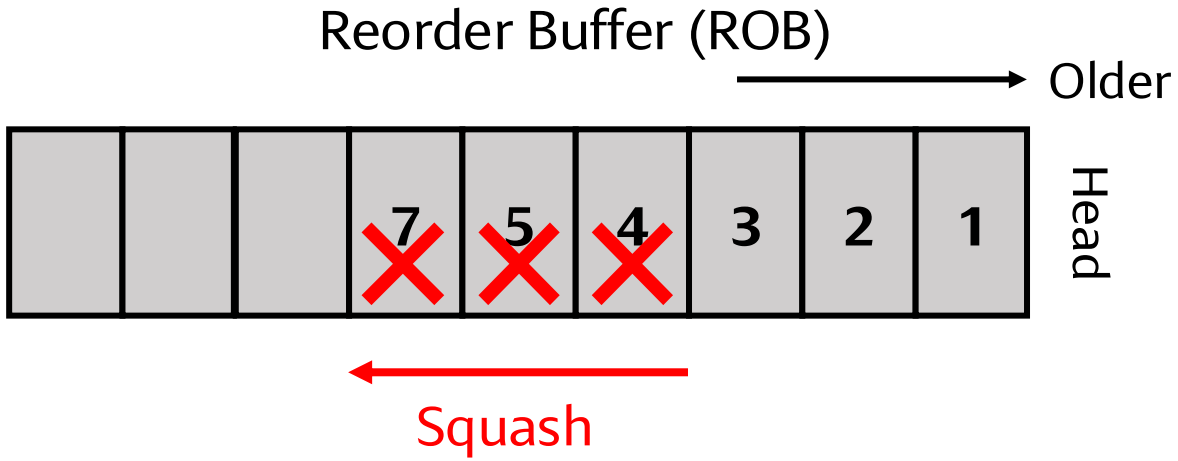


Out-of-Order Execution & Speculation

Predict taken

Misprediction!

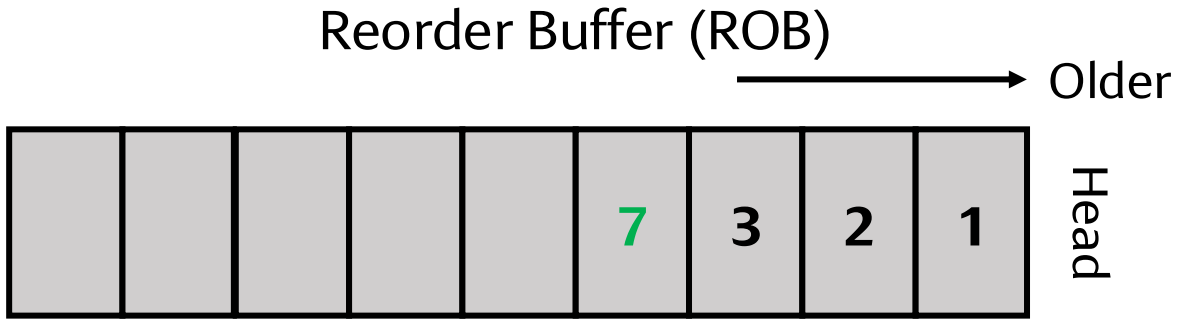
```
1.  a = 10;
2.  d = 2;
3.  if (a < d) {
4.    b = 4;
5.    c = b * 2;
6.  }
7.  e = d / 2;
```



Out-of-Order Execution & Speculation

Steer towards not taken

```
1.  a = 10;
2.  d = 2;
3.  if (a < d) {
4.    b = 4;
5.    c = b * 2;
6.  }
7.  e = d / 2;
```



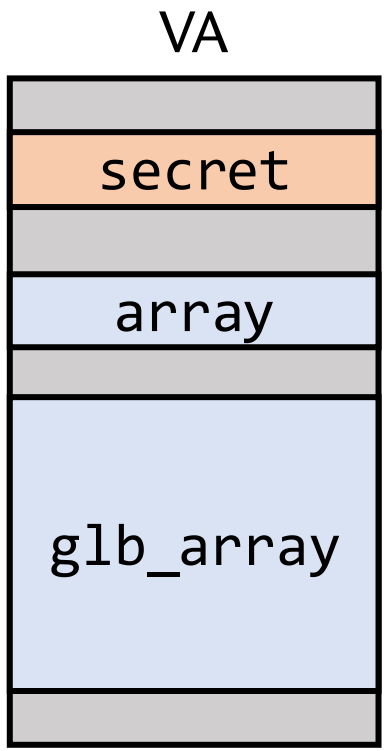
Mistakes made during the mis-speculation are never architecturally visible

Spectre: The “Worst-Ever” CPU Vulnerability?

A textbook Spectre v1 attack example

```
1  void vulnerable_code(size_t idx) {
2    if (idx < array_size) {
3      u8 data = array[idx];
4      u8 junk = glb_array[data * 4096];
5    }
6  }
```

Setup: Attacker can invoke the vulnerable code with any index and access the public glb_array, but cannot directly read the secret

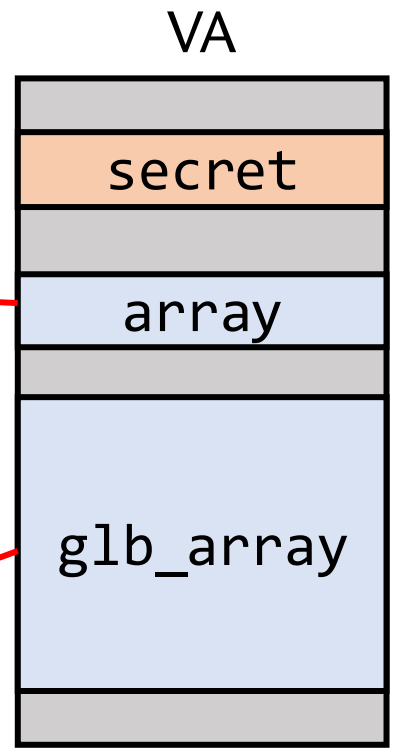


Spectre: The “Worst-Ever” CPU Vulnerability?

A textbook Spectre v1 attack example

```
1  void vulnerable_code(size_t idx) {
2    if (idx < array_size) {
3      u8 data = array[idx];
4      u8 junk = glb_array[data * 4096];
5    }
6  }
```

Taken



Attacker invocation 1: `vulnerable_code(0);`

Attacker invocation 2: `vulnerable_code(0);`

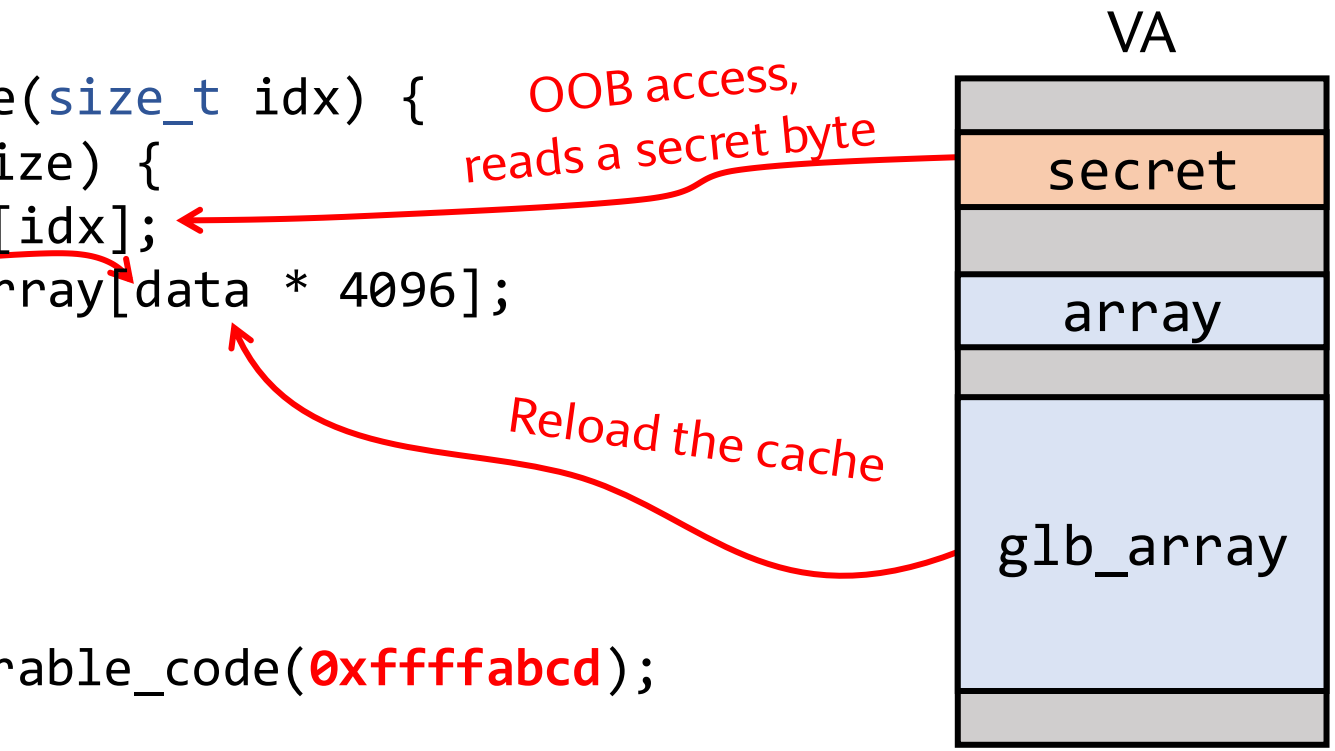
…

Spectre: The “Worst-Ever” CPU Vulnerability?

A textbook Spectre v1 attack example

Predict taken

```
1  void vulnerable_code(size_t idx) {
2    if (idx < array_size) {
3      u8 data = array[idx];
4      u8 junk = glb_array[data * 4096];
5    }
6  }
```



Attacker invocation 8: `vulnerable_code(0xffffabcd);`

A More Complete PoC

A textbook Spectre v1 attack example

Vulnerable code

```
1  void vulnerable_code(size_t idx) {
1    if (idx < array_size) {
2      u8 data = array[idx];
3      u8 junk = glb_array[data * 4096];
4    }
5  }
```

Attacker

```
for (size_t i = 0; i < 10; i++) {
  vulnerable_code(0); // mistraining
}

flush(glb_array); // prepare F+R
flush(array_size);
```

Attacker needs to flush “array_size” to enlarge the speculation window so that they can speculatively read the secret before the misprediction is corrected

A More Complete PoC

A textbook Spectre v1 attack example

Vulnerable code

```
1  void vulnerable_code(size_t idx) {
1    if (idx < array_size) {
2      u8 data = array[idx];
3      u8 junk = glb_array[data * 4096];
4    }
5  }
```

Attacker

```
for (size_t i = 0; i < 10; i++) {
  vulnerable_code(0); // mistraining
}

flush(glb_array); // prepare F+R
flush(array_size);

vulnerable_code(malicious_idx);

reload_and_decode(glb_array);
```

A More Complete PoC

A textbook Spectre v1 attack example

Vulnerable code

```
1  void vulnerable_code(size_t idx) {
1    if (idx < array_size) {
2      u8 data = array[idx];
3      u8 junk = glb_array[data * 4096];
4    }
5  }
```

Attacker

```
for (size_t i = 0; i < 10; i++) {
  vulnerable_code(0); // mistraining
}

flush(glb_array); // prepare F+R
flush(array_size);

vulnerable_code(malicious_idx);
```

FAQ

Q: What if the attacker cannot access “glb_array” for Flush+Reload?

A: Use other covert channels, such as Prime+Probe

A More Complete PoC

A textbook Spectre v1 attack example

Vulnerable code

```
1  void vulnerable_code(size_t idx) {
1    if (idx < array_size) {
2      u8 data = array[idx];
3      u8 junk = glb_array[data * 4096];
4    }
5  }
```

Attacker

```
for (size_t i = 0; i < 10; i++) {
  vulnerable_code(0); // mistraining
}

flush(glb_array); // prepare F+R
flush(array_size);

vulnerable_code(malicious_idx);
```

FAQ

Q: What if the attacker cannot flush “array_size”? Can we flush the index?

A: Like Evict+Reload, use an eviction set. And no, we need the index cached for the OOB access

A More Complete PoC

A textbook Spectre v1 attack example

Vulnerable code

```
1  void vulnerable_code(size_t idx) {
1    if (idx < array_size) {
2      u8 data = array[idx];
3      u8 junk = glb_array[data * 4096];
4    }
5  }
```

Attacker

```
for (size_t i = 0; i < 10; i++) {
  vulnerable_code(0); // mistraining
}

flush(glb_array); // prepare F+R
flush(array_size);

vulnerable_code(malicious_idx);
```

FAQ

Q: Do I need to use an eviction set to evict “array_size”? Why not traverse a large-enough array?

A: Traversing a large array would evict both the “array_size” and the secret string

Spectre = Misprediction + Microarchitectural Side/Covert Channel

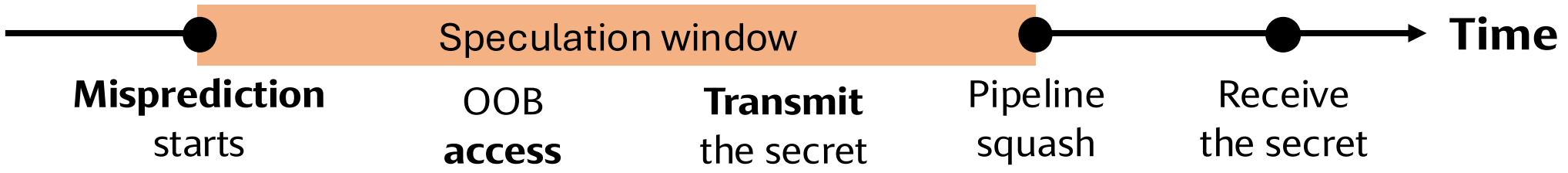
Using Flush+Reload

```
void vulnerable_code(size_t idx) {
  if (idx < array_size) { // mispredict
    u8 data = array[idx]; // access
    u8 junk = glb_array[data*4096]; // transmit
  }
}
```

Using SMT port contention[1]

```
void vulnerable_code(size_t idx) {
  if (idx < array_size) { // mispredict
    u8 data = array[idx]; // access
    if (data) div(); // transmit via
    else mul();      // port contention
  }
}
```

Spectre gadget/Universal Read Gadget
(Leak arbitrary byte from the same address space)



[1]Bhattacharyya et al. "SMoTherSpectre: Exploiting Speculative Execution through Port Contention"

Spectre Software Mitigation: LFENCE


```
void vulnerable_code(size_t idx) {
  if (idx < array_size) {
    LFENCE; // block spec. access
    u8 data = array[idx]; // Access
    LFENCE; // block transmission
    u8 junk = glb_array[data * 4096]; // Transmit
  }
}
```

Stops all instructions, high performance overhead

"The LFENCE instruction and other serializing instructions ensure that **no later instruction will execute**, even speculatively, until all prior instructions have completed locally." from Intel's Spectre mitigation guidance[1]

[1]https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/bounds-check-bypass.html

Spectre Software Mitigation: Array Index Nospec


```
void vulnerable_code(size_t idx) {
  if (idx < array_size) {
    _idx = array_index_nospec(idx, array_size);
    u8 data = array[idx]; // Access

    u8 junk = glb_array[data * 4096]; // Transmit
  }
}
```

A branchless helper MACRO that clamps the index within the range of [0, array_size)

Manual effort!

Why Do We Have This Spectre Gadget

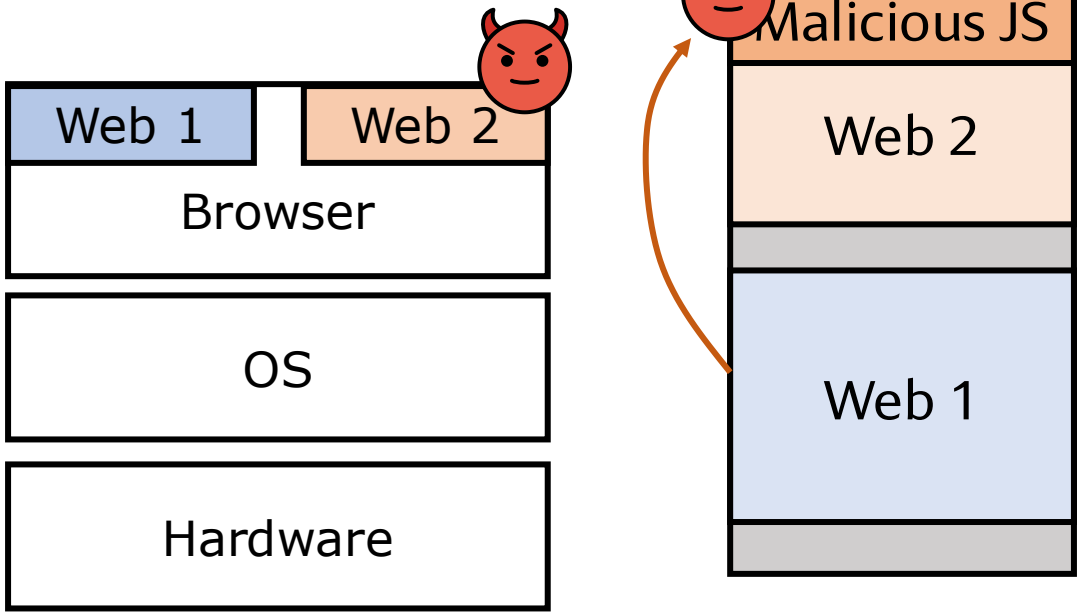
This does not look realistic?

```
void vulnerable_code(size_t idx) {
  if (idx < array_size) {
    u8 data = array[idx];
    u8 junk = glb_array[data * 4096];
  }
}
```

Why Do We Have Spectre Gadgets?

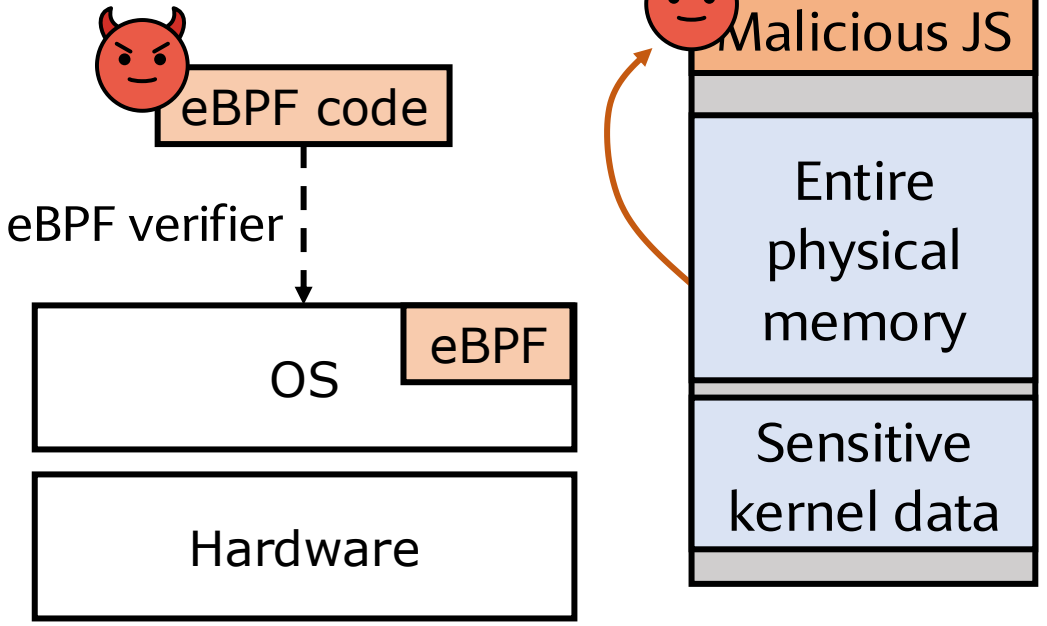
The attacker crafted it

Spectre in Browser



Defense: Cross-Site Isolation (Run different sites inside different browser processes)

Spectre in Linux Kernel



Defense: Accept eBPF code from only root users

Why Do We Have Spectre Gadgets?

The victim happens to have it

```
void vulnerable_code(size_t idx) {
  if (idx < array_size) {
    u8 data = array[idx];
    u8 junk = glb_array[data * 4096];
  }
}
```

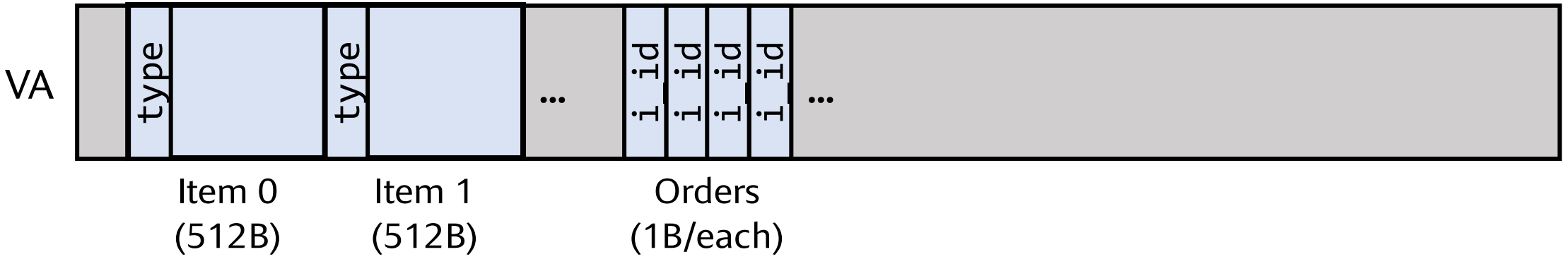
It's common for benign programs to have "tandem" array accesses or pointer chasing

A Toy Sales Management System

```
typedef struct {
    uint8_t type;
    char name[64];
    char desc[447];
} Item; // 512B
```

```
typedef struct {
    uint8_t item_id;
} Order; // 1B
```

```
typedef struct {
    Item *items;
    Order *orders;
    char memo[512];
    size_t num_orders, num_items;
    char manager[256];
} SalesRecords;
```



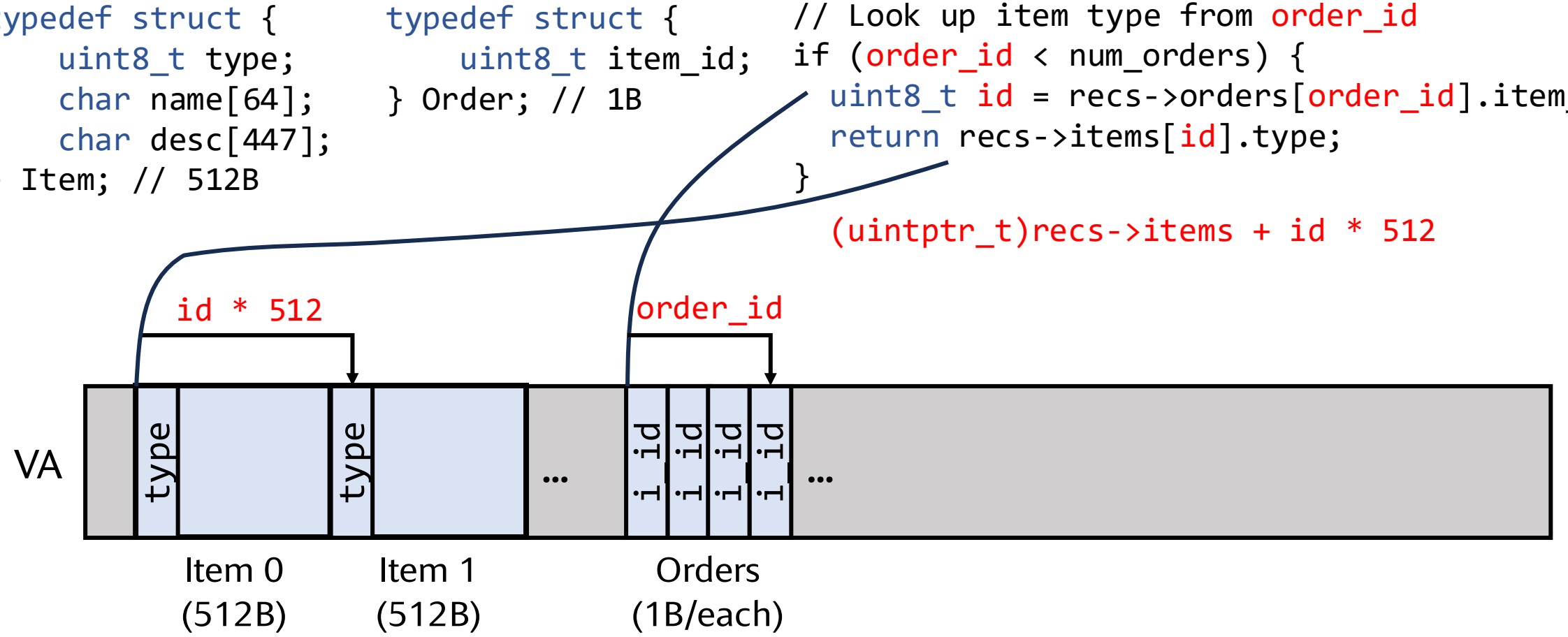
A Toy Sales Management System

```
typedef struct {
    uint8_t type;
    char name[64];
    char desc[447];
} Item; // 512B
```

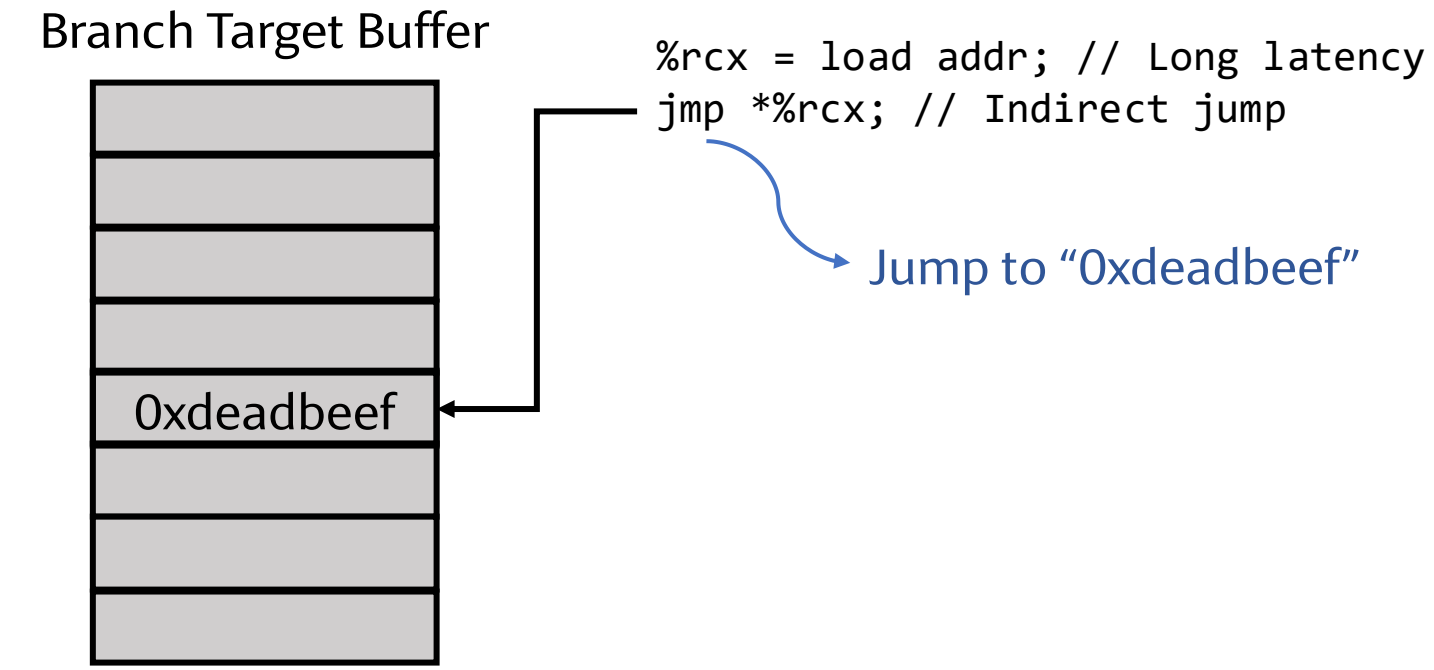
```
typedef struct {
    uint8_t item_id;
} Order; // 1B
```

```
// Look up item type from order_id
if (order_id < num_orders) {
  uint8_t id = recs->orders[order_id].item_id;
  return recs->items[id].type;
}
```

(uintptr_t)recs->items + id * 512

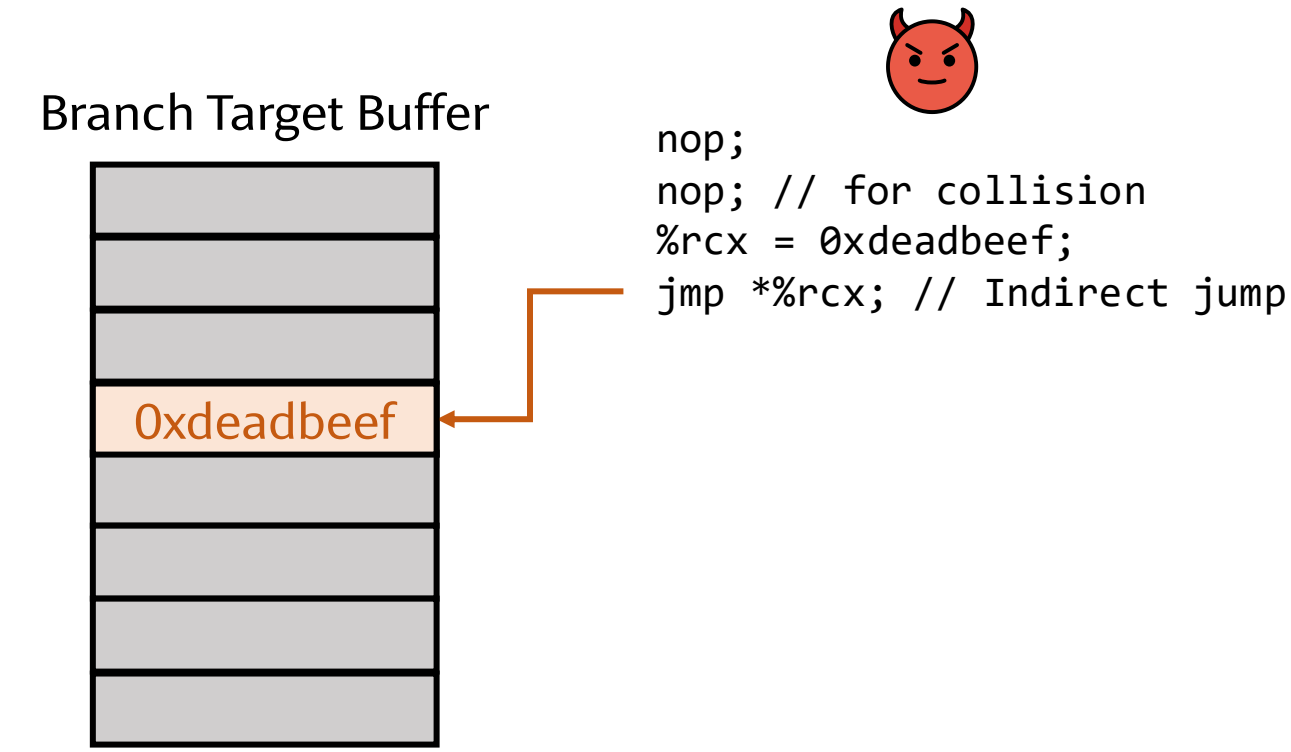


Spectre v2: Branch Target Inject



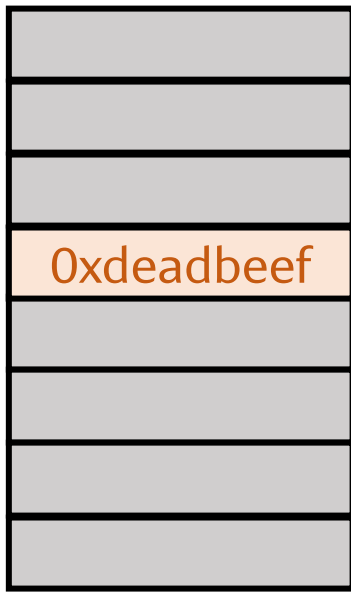
Predicts the destination of a jmp, indexed by PC

Spectre v2: Branch Target Inject



Predicts the destination of a jmp, indexed by PC

Spectre v2: Branch Target Inject



Predicts the destination of a jmp, indexed by PC

```
nop;
nop; // for collision
%rcx = 0xdeadbeef;
jmp *%rcx; // Indirect jump
```

```
if (idx < array_size) {
  u64 %rax = array[idx]; // Access
}
jmp *%rcx; // Indirect jump
```

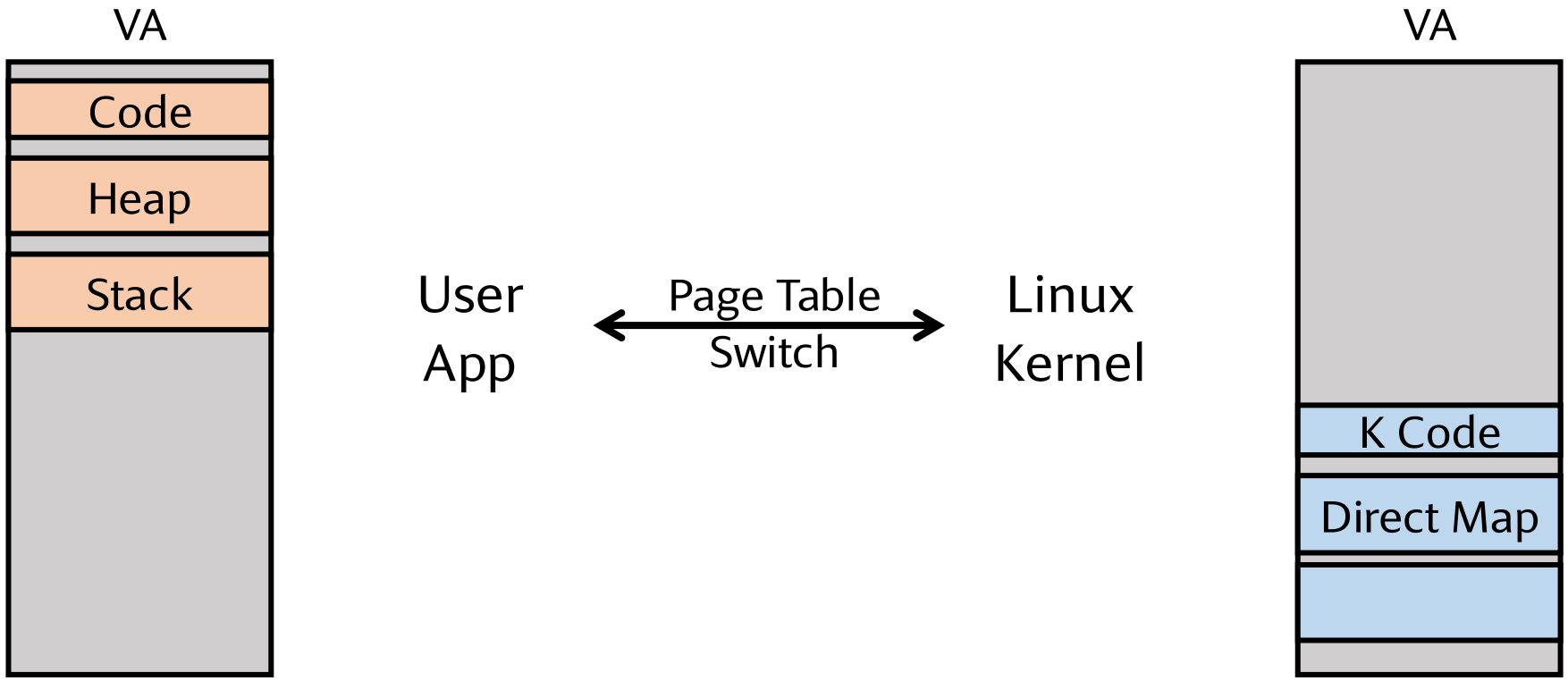
```
...
%rax = 10;
%rdx = array_base_ptr;
%rax <<= 3;
load %rdx + %rax; // Transmit
...
```

Misuse existing code out-of-context to craft a Spectre gadget

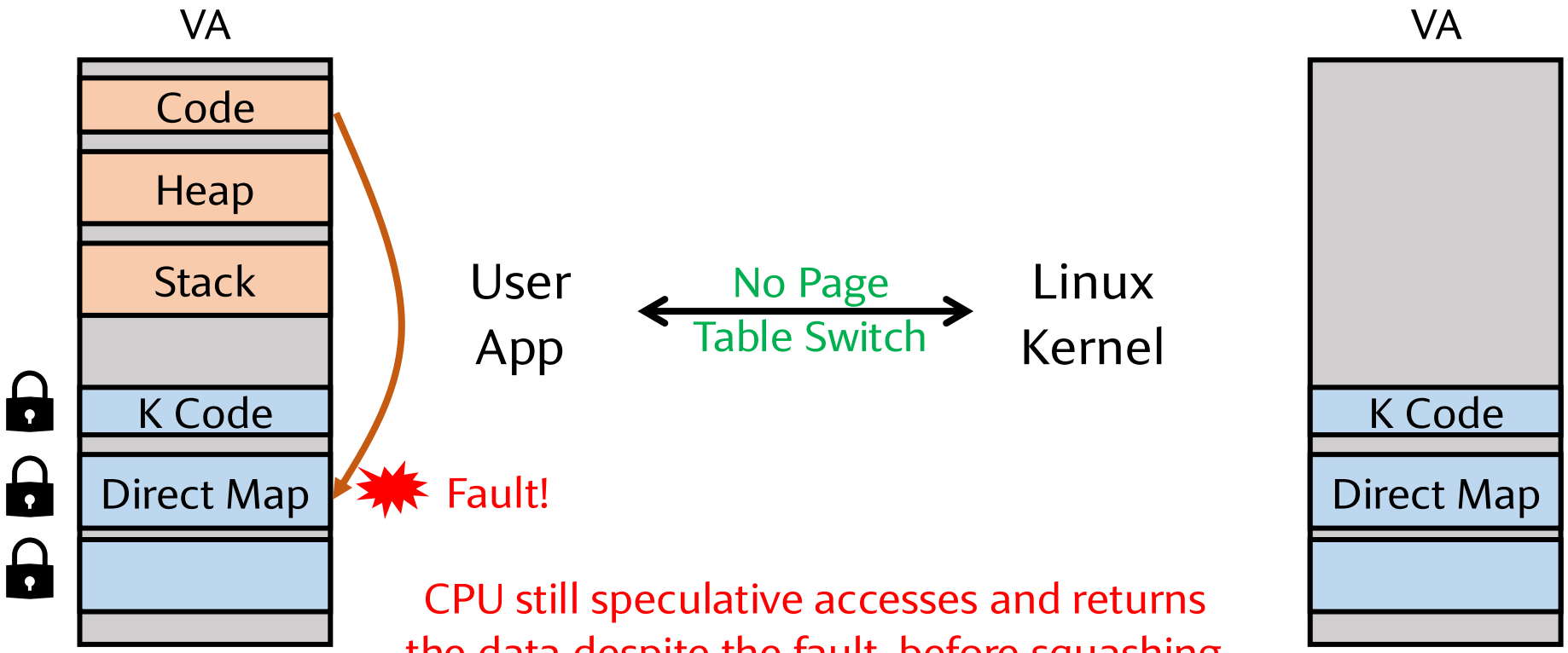
Is Spectre a Bug?

No, it's a feature. We did not realize that clever performance-enhancing techniques can have unintended security implications

Meltdown Attack: This One Actually is a Bug



Meltdown Attack: This One Actually is a Bug



CPU still speculative accesses and returns the data despite the fault, before squashing