nedeljainformatike v6.0

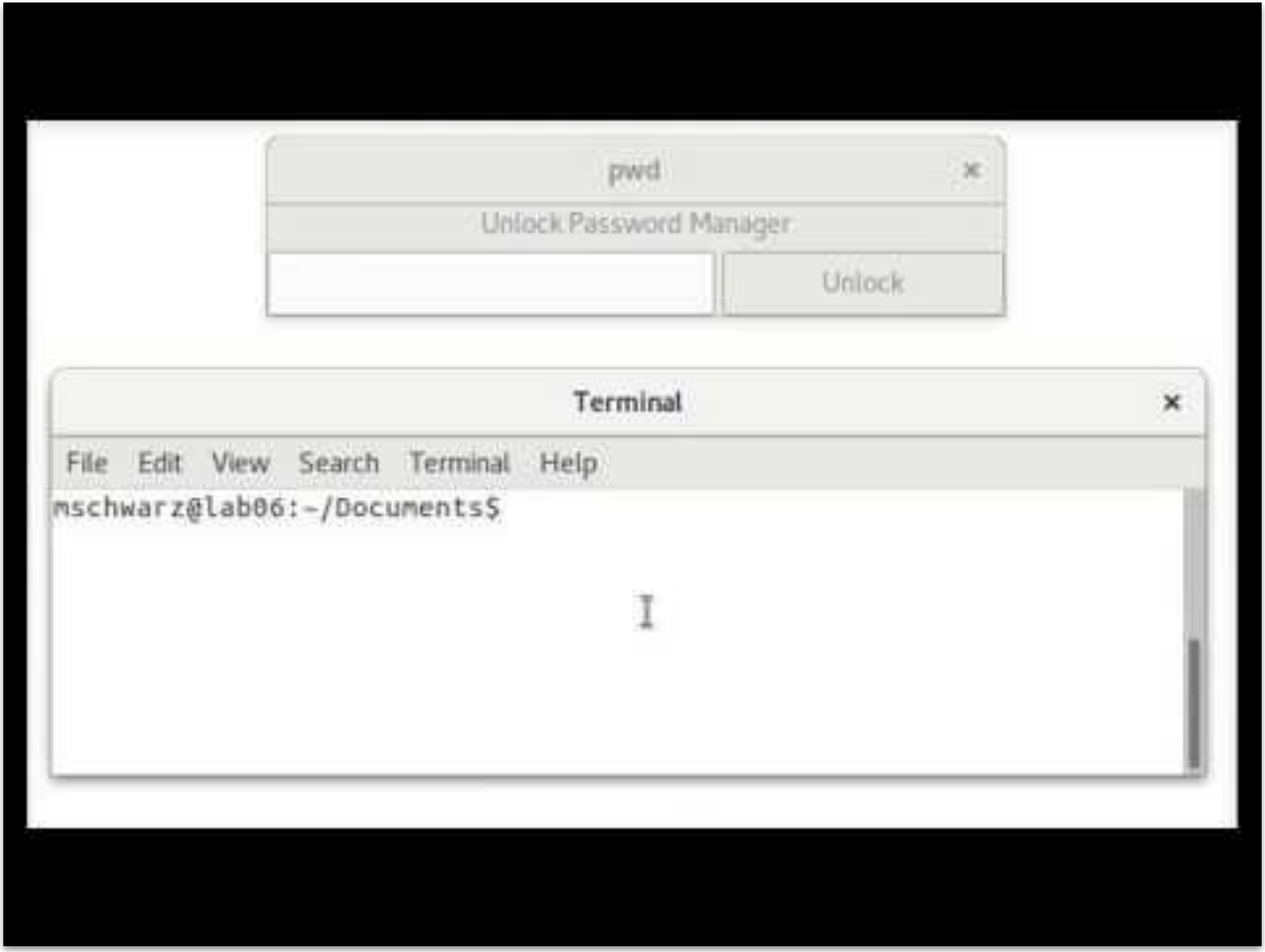
Meltdown: napadi na nivou mikroarhitekture

Marina Ivanović

Matematička gimnazija

23. 04. 2021.

Meltdown na delu



Pregled

- Ranjivost modernih procesora, koja dovodi do potpunog pristupa memoriji, koja bi inače trebalo da bude zaštićena.
- Efektivno, svi Intelovi procesori od 1995. su zahvaćeni!
- Nekoliko ljudi iz različitih institucija je otkrilo Meltdown u isto vreme:
  - Google Project Zero
  - Cyberus Technology
  - Graz University of Technology
- Rad je objavljen 2018. godine

Agenda

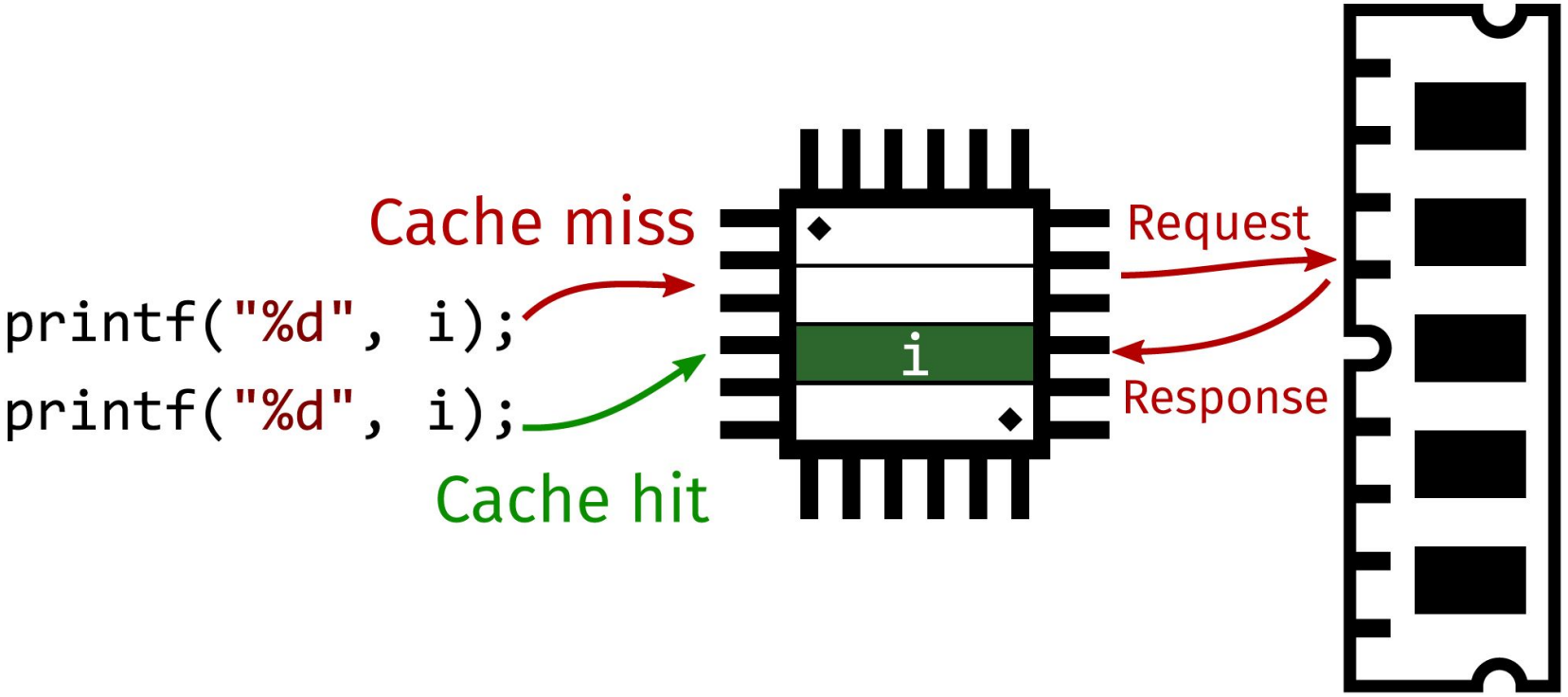
1. „Cache side-channel” napadi
2. „Out-of-order” izvršavanje
3. Meltdown

Keš memorija

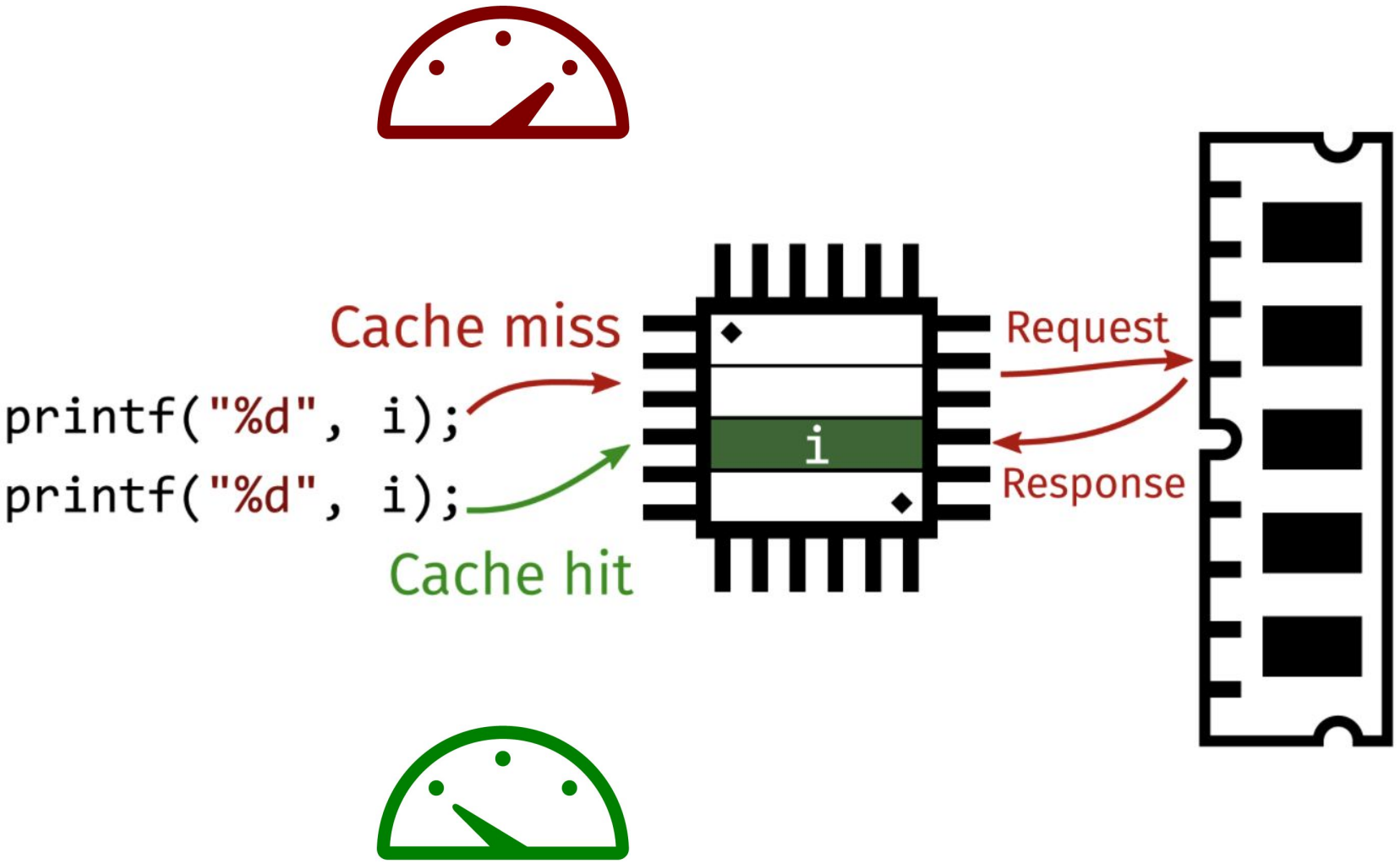
- Memorija velike brzine
- Čuva podatke kojima je “nedavno pristupano”

  → u slučaju da procesor želi ponovo da im pristupi, to može da uradi brzo

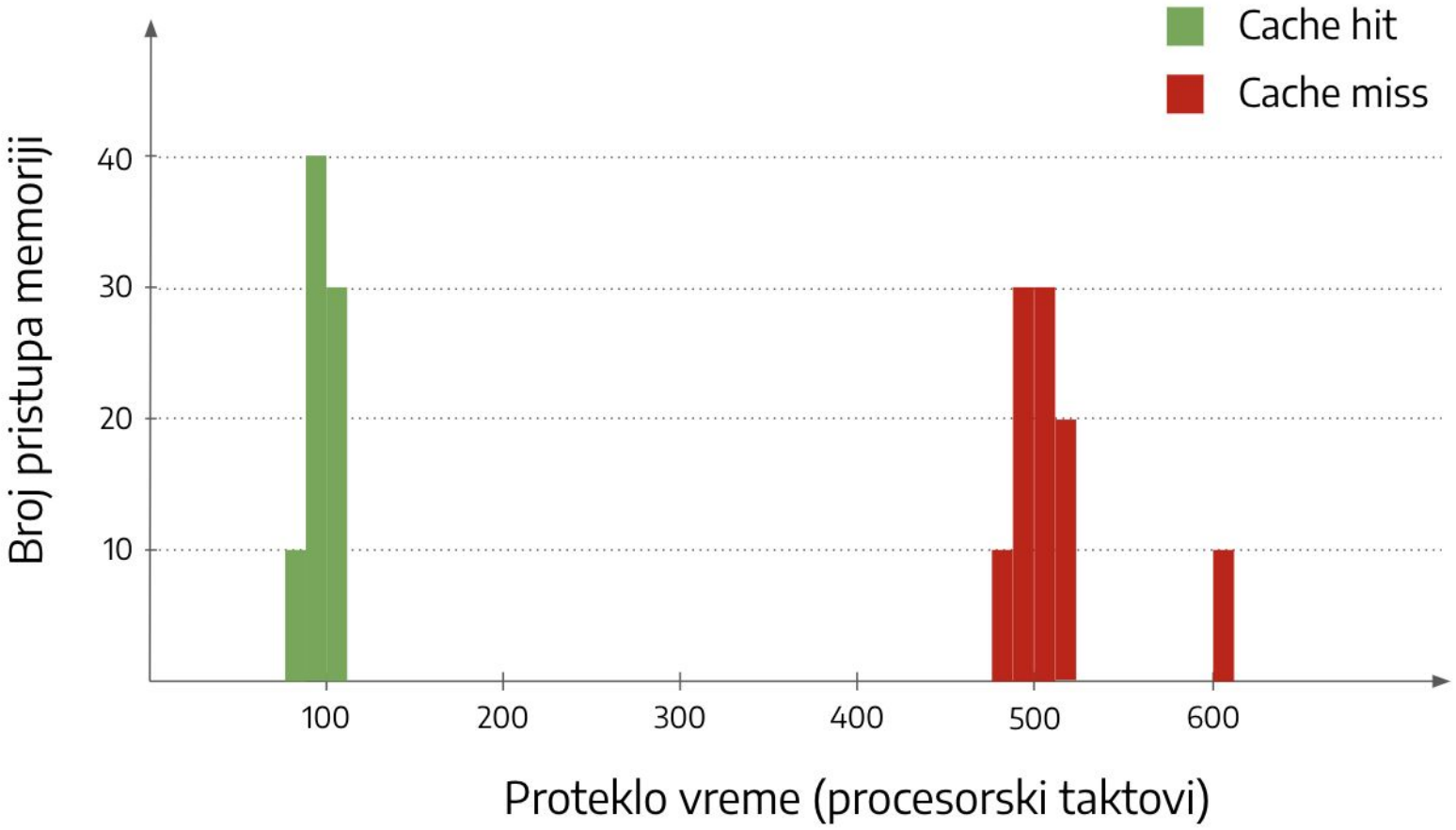
Keš na primeru



Kako iskoristiti keš za napad?



Vreme pristupa memoriji



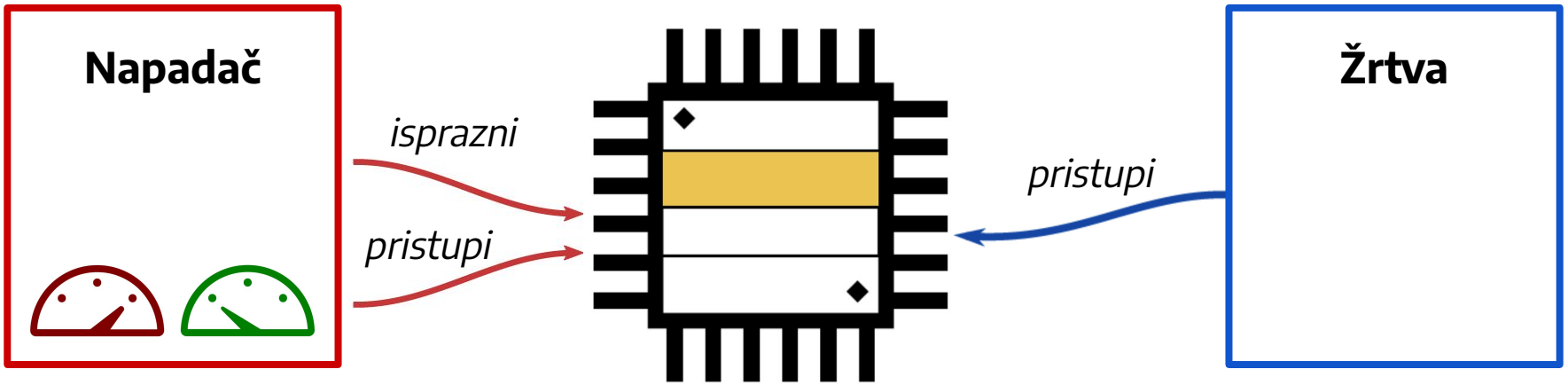
1. pitalica

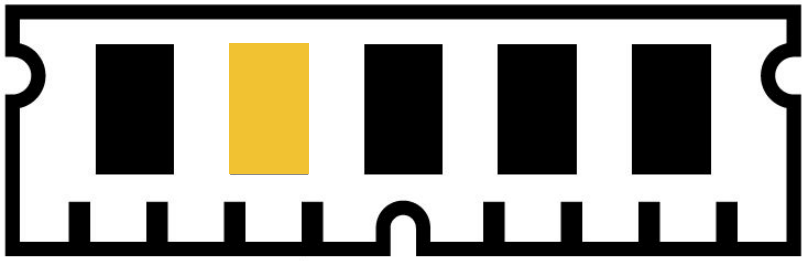
Realizujete sledeći eksperiment: probate da pristupite određenoj memorijskoj lokaciji i usput merite broj procesorskih taktova koji vam je za to potreban. Ako su vam na raspolaganju merenja sa prethodnog slajda, i izmerili ste 90 taktova, koji zaključak biste doneli na osnovu ovog merenja?

Ponuđeni odgovori:

a) Nedavno je bilo pristupljeno ovoj memorijskog lokaciji.
b) Nedavno nije bilo pristupljeno ovoj memorijskog lokaciji.

Napad: „isprazni pa ponovo popuni”



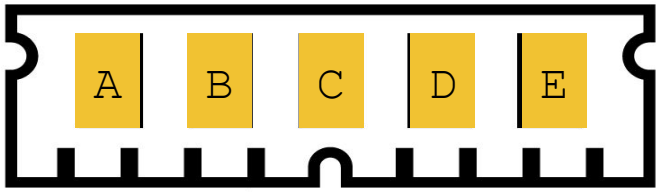


Zajednička memorija

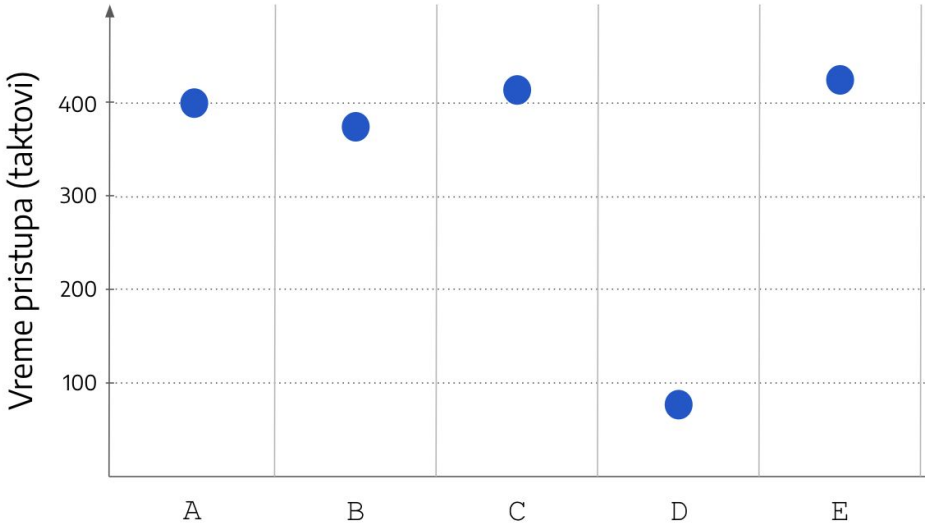
na engleskom: Flush + Reload

2. pitalica

Zajednička memorija izgleda ovako:

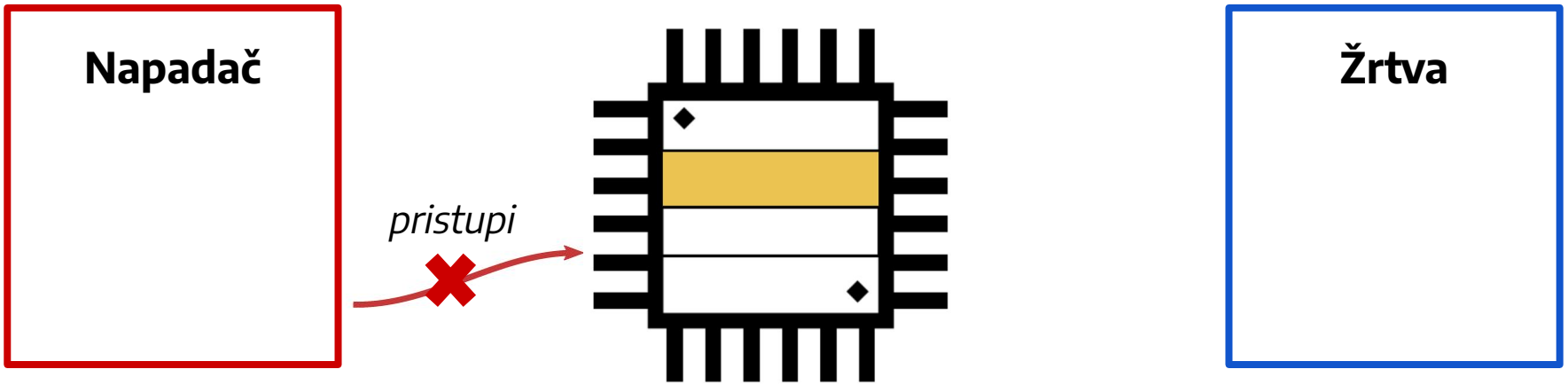


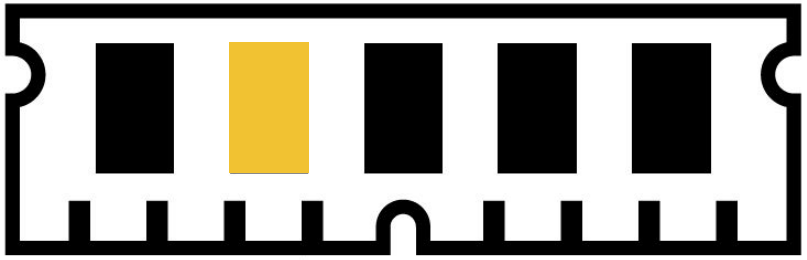
Napadač je izvršio „isprazni pa ponovo popuni” napad i izmerio vreme pristupa ovih 5 lokacija u memoriji. Rezultati izgledaju ovako:



Šta zaključujete, kom slovu je žrtva najverovatnije pristupila u međuvremenu?

Zabranjen pristup memoriji





Zajednička memorija

→ i napadač i žrtva moraju imati pristup zajedničkoj memoriji

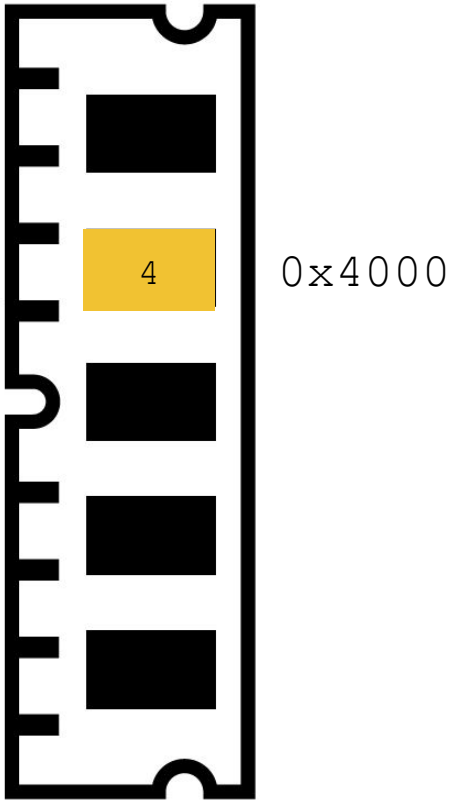
Agenda

1. „Cache side-channel” napadi
2. „Out-of-order” izvršavanje
3. Meltdown

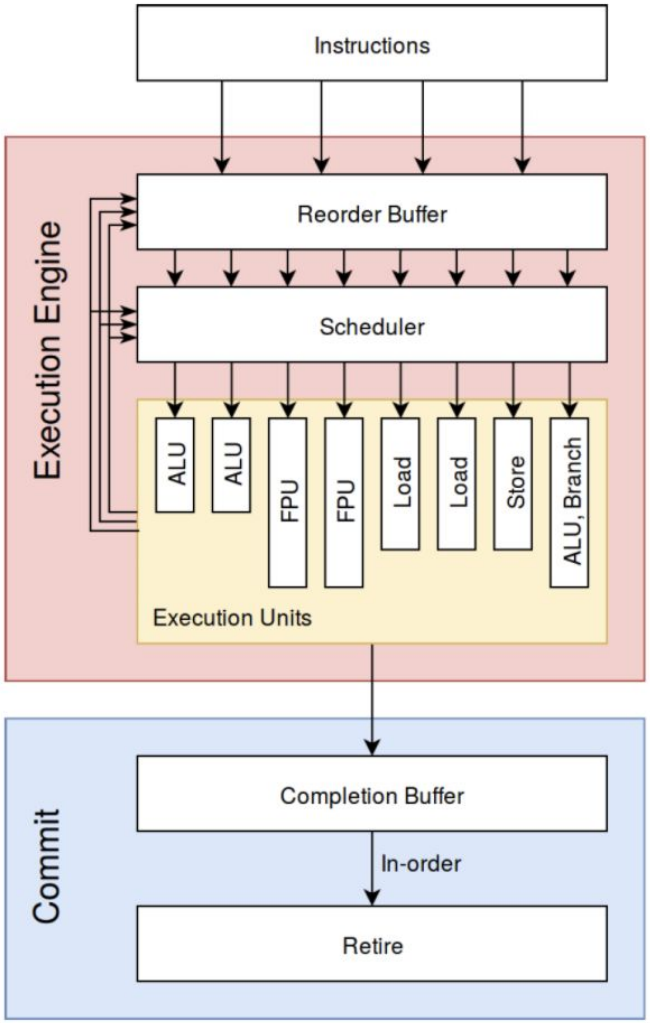
Kako se ovo izvršava?

```
1: mov eax, [0x4000]
2: add eax, 2
3: add ebx, 10
4: mul ecx, 4
5: add edx, edx
```

```
1: eax = 4 (mem)
2: eax += 2
3: ebx += 10
4: ecx *= 4
5: edx += edx
```

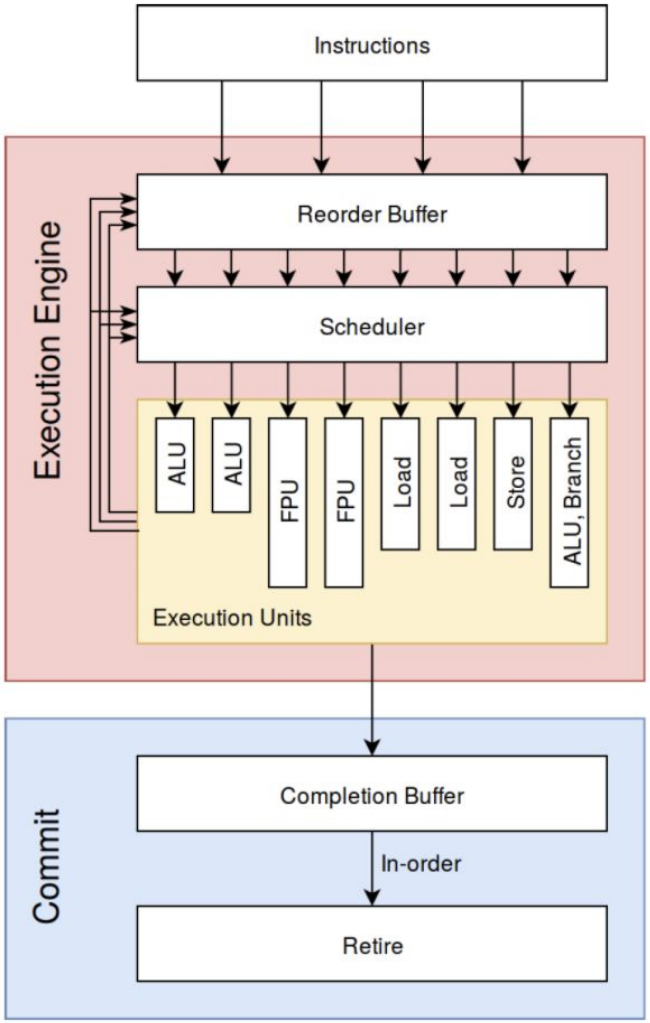


OoO izvršavanje - primer




```
1: mov eax, [0x4000]

2: add eax, 2

3: add ebx, 10

4: mul ecx, 4

5: add edx, edx
```

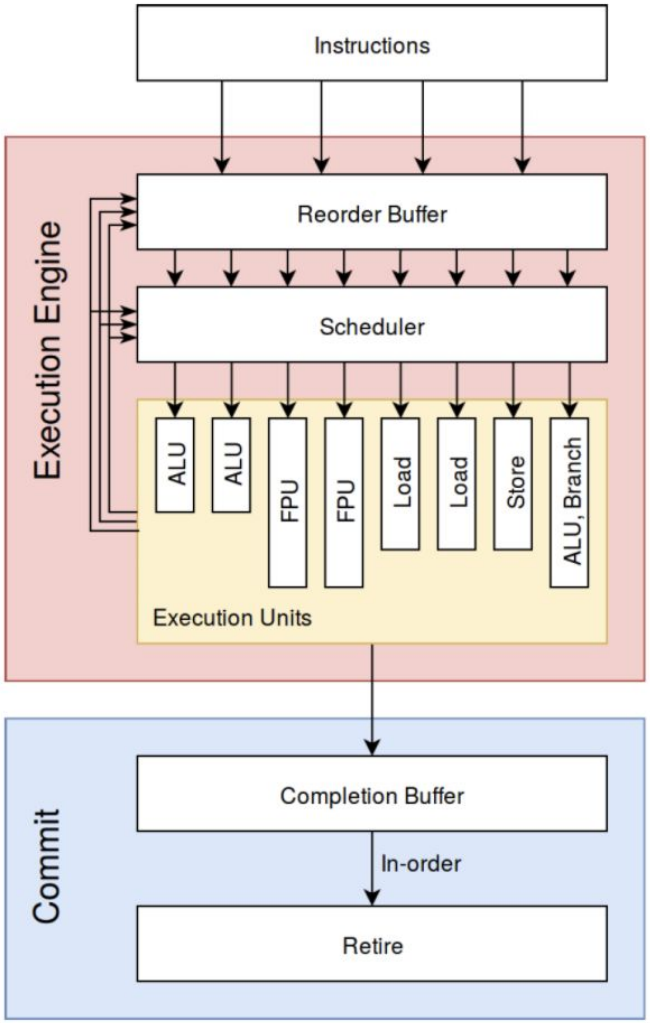
OoO izvršavanje - primer



1 2 3 4

```
1: mov eax, [0x4000]

2: add eax, 2

3: add ebx, 10

4: mul ecx, 4

5: add edx, edx
```

OoO izvršavanje - primer

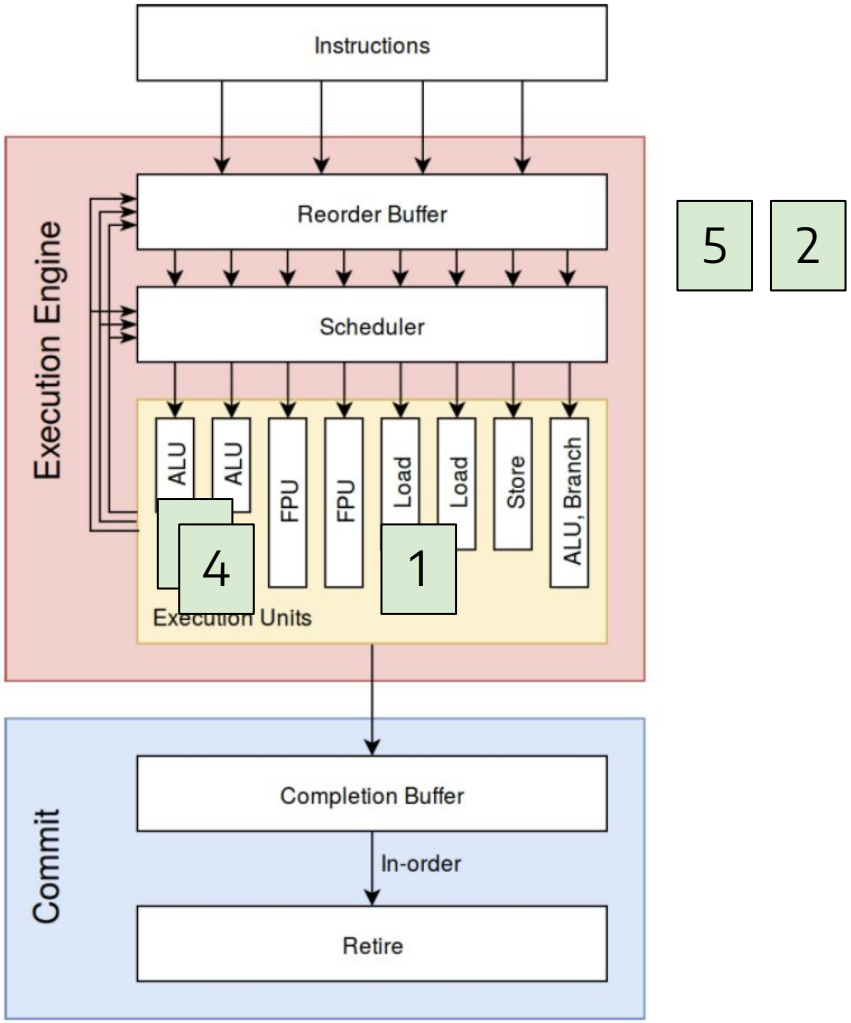


5

1 2 3 4

```
1: mov eax, [0x4000]

2: add eax, 2

3: add ebx, 10

4: mul ecx, 4

5: add edx, edx
```

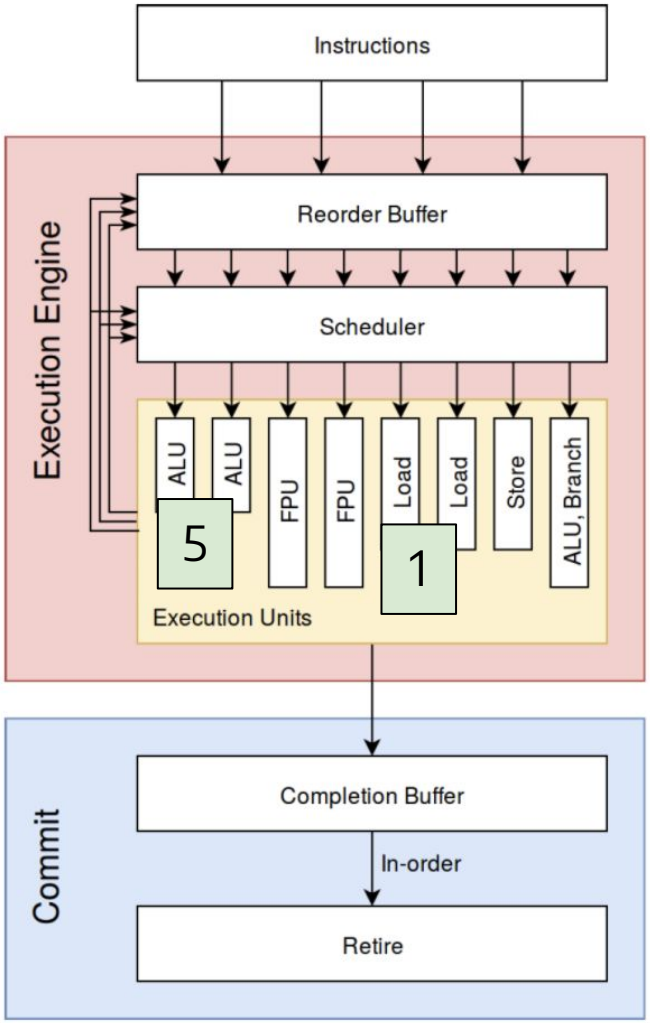
OoO izvršavanje - primer



5 2

```
1: mov eax, [0x4000]

2: add eax, 2

3: add ebx, 10

4: mul ecx, 4

5: add edx, edx
```

OoO izvršavanje - primer



2

5

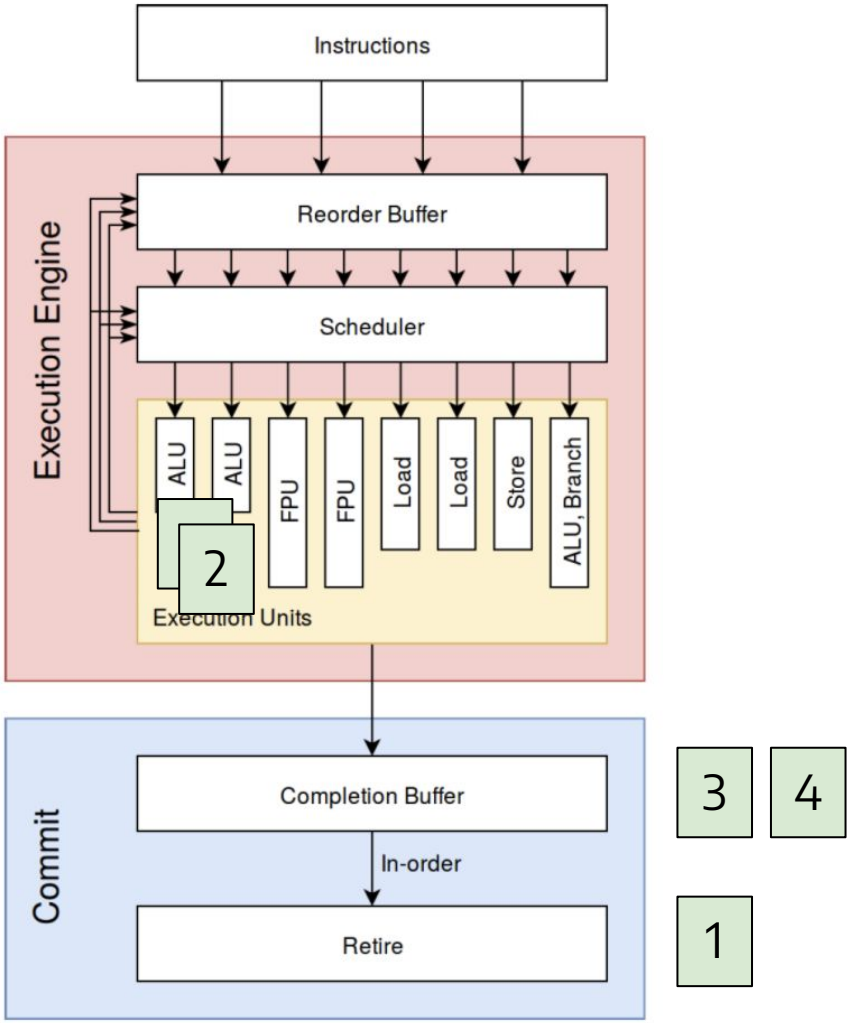
1

3

4

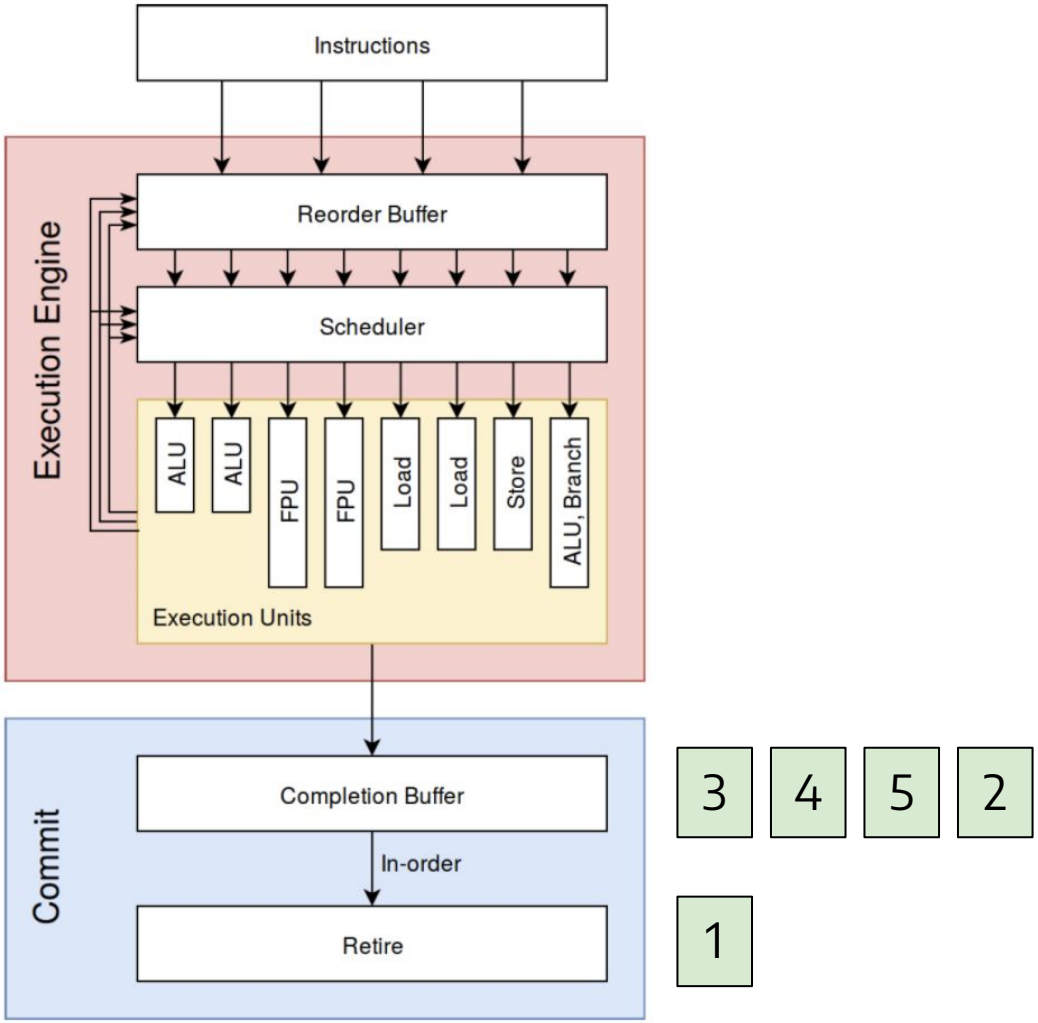
```
1: mov eax, [0x4000]

2: add eax, 2

3: add ebx, 10

4: mul ecx, 4

5: add edx, edx
```

OoO izvršavanje - primer




```
1: mov eax, [0x4000]

2: add eax, 2

3: add ebx, 10

4: mul ecx, 4

5: add edx, edx
```

OoO izvršavanje - primer

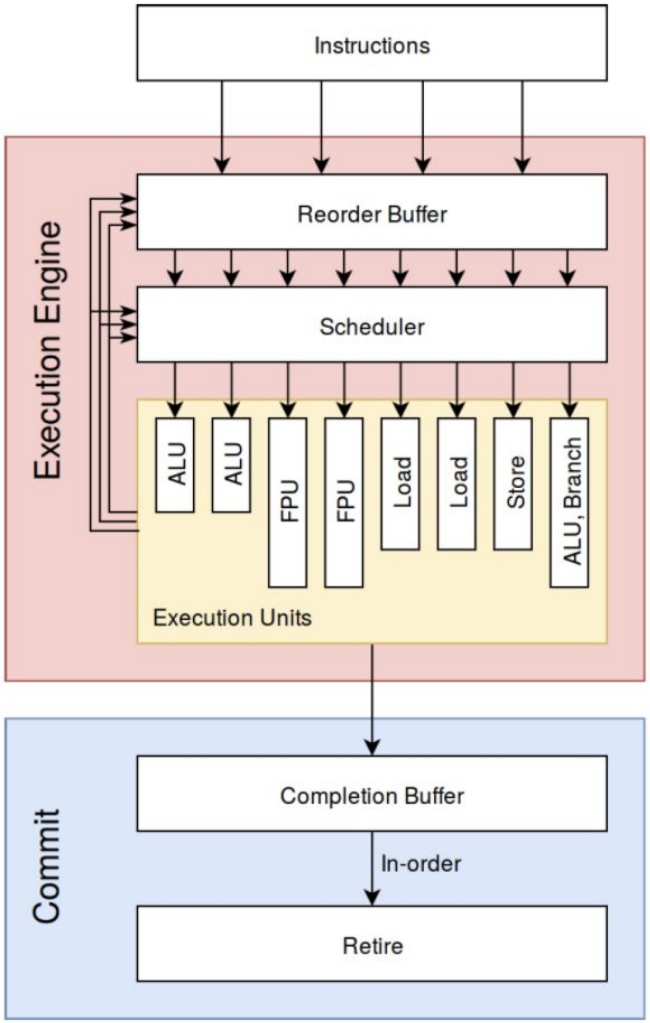


3 4 5 2

1

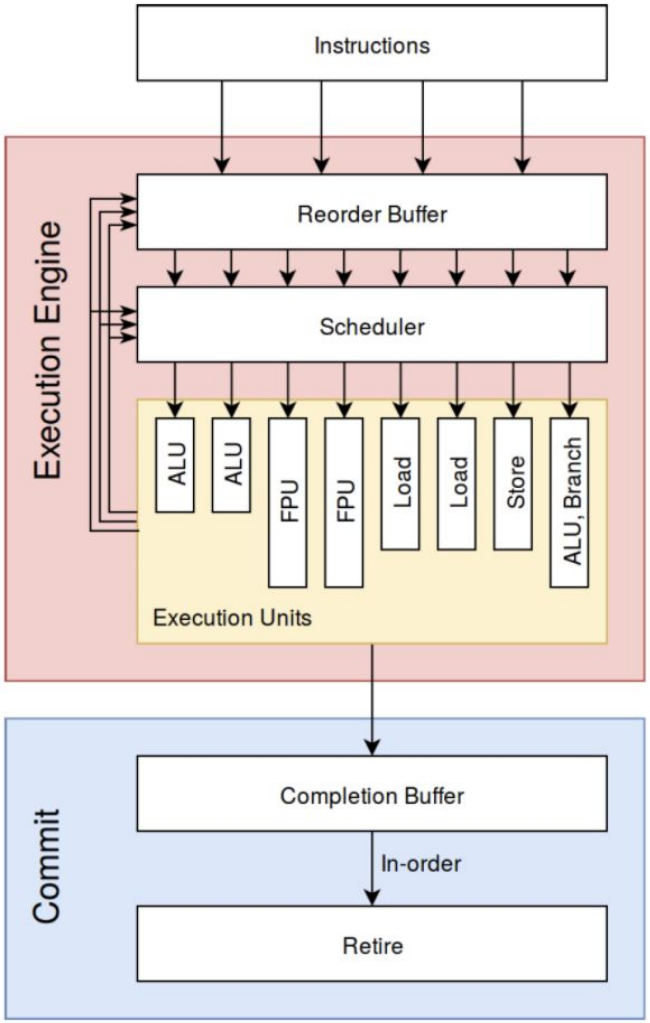
```
1: mov eax, [0x4000]

2: add eax, 2

3: add ebx, 10

4: mul ecx, 4

5: add edx, edx
```

OoO izvršavanje - primer




```
1: mov eax, [0x4000]

2: add eax, 2

3: add ebx, 10

4: mul ecx, 4

5: add edx, edx
```

3. pitalica



1 2

```
1: mov eax, [0x4000]

2: mov ebx, [0x4000 + eax]
```

Šta mislite, da li će vrednost sa lokacije `0x4004` da se učita *pre* nego što se instrukcija `1` potpuno završi (uključujući i deo `retire`)?

**high-level ideja, implementacija je malo drugačija*

Zabranjen pristup memoriji

`retire`
- proveri da li je pristup memoriji dozvoljen
- ako jeste, sačuvaj vrednost u registar
- ako nije, baci izuzetak (*throw exception*)

→ provera da li je pristup memoriji zabranjen se vrši tek u `retire` delu

→ u međuvremenu, pre nego što se baci izuzetak, druge instrukcije mogu da se izvrše (zbog OoO izvršavanja)

Agenda

1. „Cache side-channel” napadi
2. „Out-of-order” izvršavanje
3. Meltdown

Meltdown: potpuno topljenje sigurnosnih barijera

Korak 1: pristup zabranjenoj memoriji

```
1: mov eax, [zabranjena_mem_lokacija]

2: neka instrukcija
```

nakon ove instrukcije će biti bačen izuzetak

→ moramo da ga „uhvatimo”, da bi naš program nastavio nesmetano da radi

zbog OoO izvršavanja, ova instrukcija će se izvršiti pre nego što bude bačen izuzetak

→ možemo da je iskoristimo da pročitamo šta je u memoriji kojoj je pristup zabranjen

Korak 2: korišćenje te vrednosti kao indeks niza

```
1: mov eax, [zabranjena_mem_lokacija]

2: mov ebx, [naš_niz + eax]
```

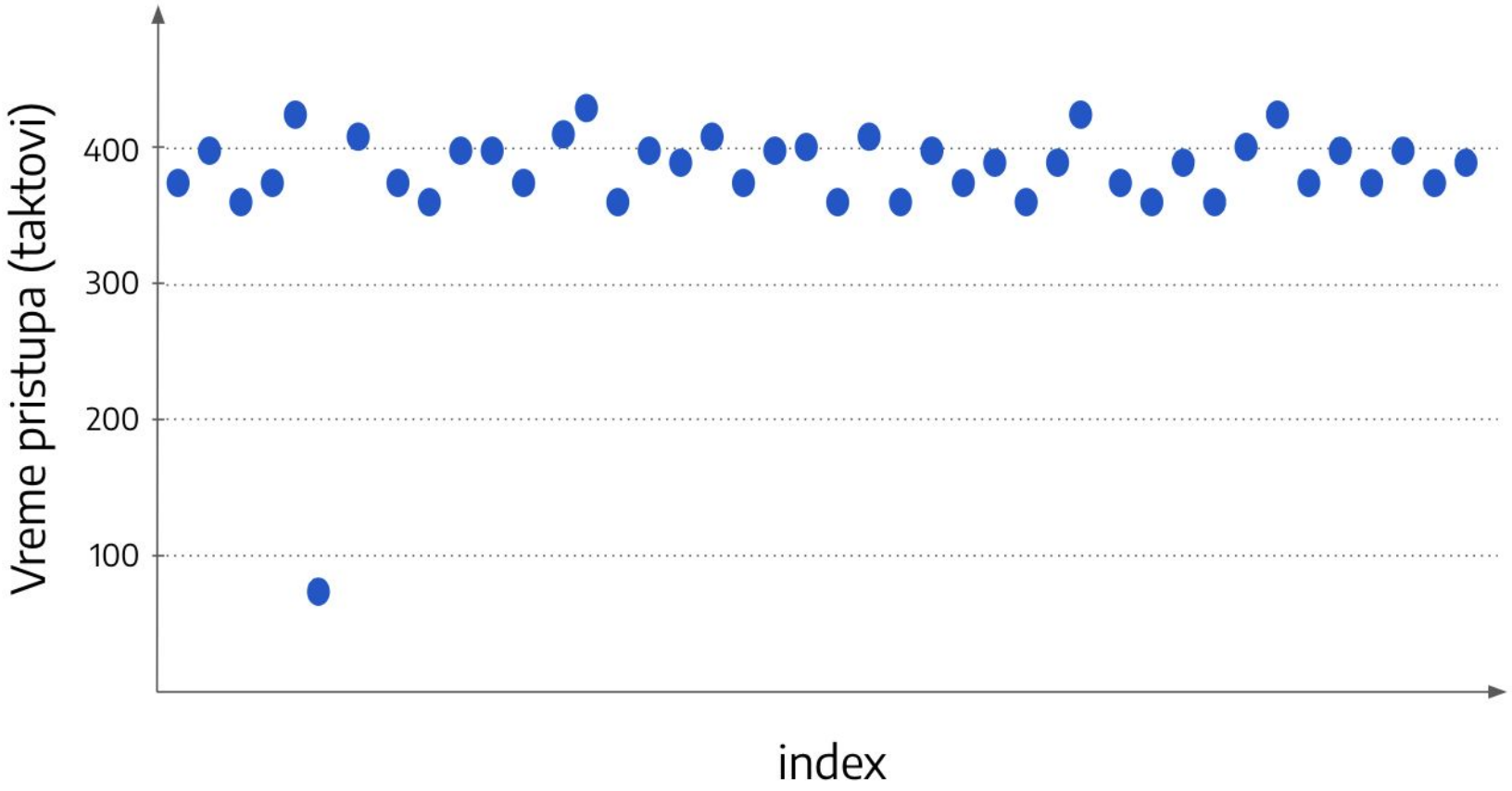
→ nakon što bude bačen izuzetak, vrednost `5` neće biti u `ebx`

→ ALI, 5. element našeg niza je keširan!

```
naš_niz        0              zabranjena_mem_lokacija        5
               1
               2
               3
               4
               5   → ebx
               6
               ...
```

**high-level ideja, implementacija je malo drugačija*

Korak 3: koji član niza je keširan?



*high-level ideja, implementacija je malo drugačija

Meltdown u 3 ipo koraka

1. Pristupi vrednosti u memoriji, kojoj je pristup zabranjen
2. Iskoristi tu vrednost da modifikuješ keš, na osnovu memorijske adrese

2.1. Poželjno, zaobići izuzetak, koji ćeš dobiti zbog neovlašćenog pristupa memoriji

3. Iskoristi cache side-channel za čitanje vrednosti koji inače ne bi trebalo

Ponovi ova 3 ipo koraka, sve dok ne pročitaš svaku vrednost u memoriji koju želiš

Meltdown: potpuno topljenje sigurnosnih barijera

Am I affected by the vulnerability?
- Most certainly, yes.

Can I detect if someone has exploited Meltdown against me?
- Probably not. [...] does not leave any traces [...].

What can be leaked?
- [...] Exploit can read the memory content of your computer. This may include passwords and sensitive data stored on the system.

Which systems are affected by Meltdown?
- Desktop, Laptop, and Cloud computers may be affected by Meltdown. [...] Every Intel processor which implements out-of-order execution [...], which is effectively every processor since 1995 [...].

preuzeto sa sajta meltdownattack.com

Agenda

1. „Cache side-channel” napadi
2. „Out-of-order” izvršavanje
3. Meltdown
4. I šta sad?

...ali ovo nije sve, imamo i Spectre

4. pitalica

Uzmite u obzir sledeći pseudo-kod.

```
array_size = 3
array = {0, 1, 2}  3, 4, 5, 6, 7, … ?

if (index < array_size)
    tmp = array[index]
```

Ako je vrednost `index` veća od vrednosti `array_size` (npr. `7`), da li će se izvršiti deo koda pod `if`-om?

Rešenje i posledice

- Rešenje postoji!
  - Bolja izolacija delova memorije
  - Procesor koji ne pogađa šta će se desiti sa grananjem
- ... ali i posledice
  - Performanse procesora se smanjuju i do 30%

Velika lekcija: napadi na nivou mikroarhitekture

- Arhitektura: koja je logika iza rada naših računara?

```
if (index < array_size)
    tmp = array[index]
```

ovo se sigurno neće izvršiti

- Mikroarhitektura: kako naši računari rade *zapravo*?

```
if (index < array_size)
    tmp = array[index]
```

... ili možda ipak hoće?

- Obično, kada razgovaramo o sigurnosti, razgovaramo o arhitekturi. Međutim, Meltdown i Spectre su nam pokazali da je mikroarhitektura jednako bitna i jednako opasna.

Hvala na pažnji!

Pitanja?

Reference

- Meltdown: Reading Kernel Memory from User Space
- Meltdown and Spectre
- Predavanja prof. Srđana Čapkuna sa ETH