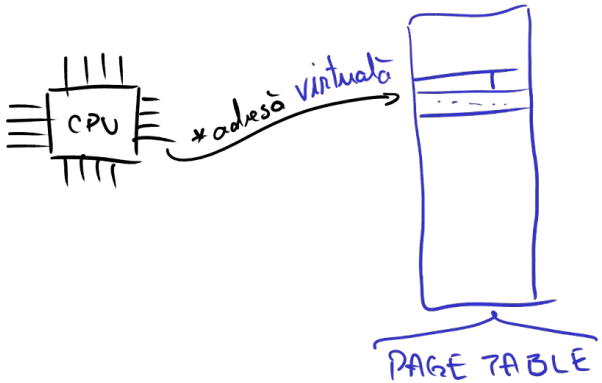
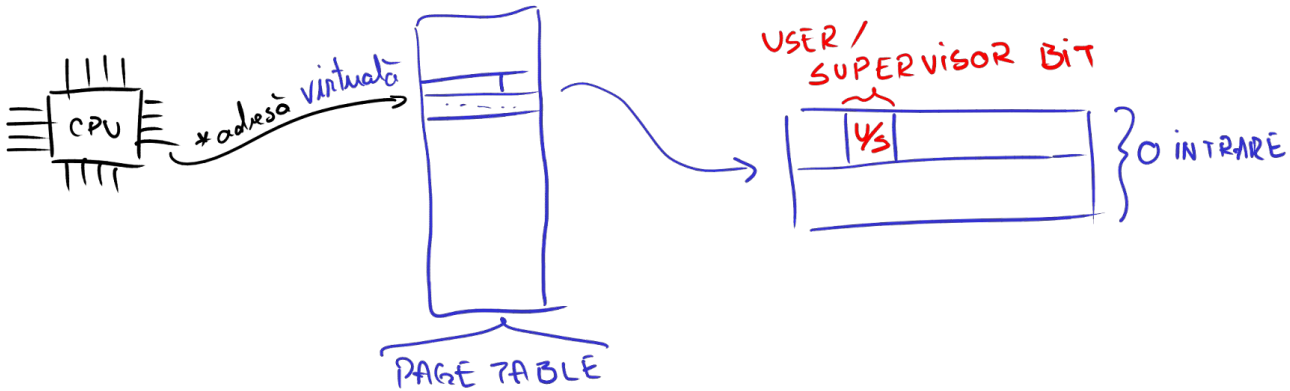
Atacuri Speculative

Meltdown & Spectre

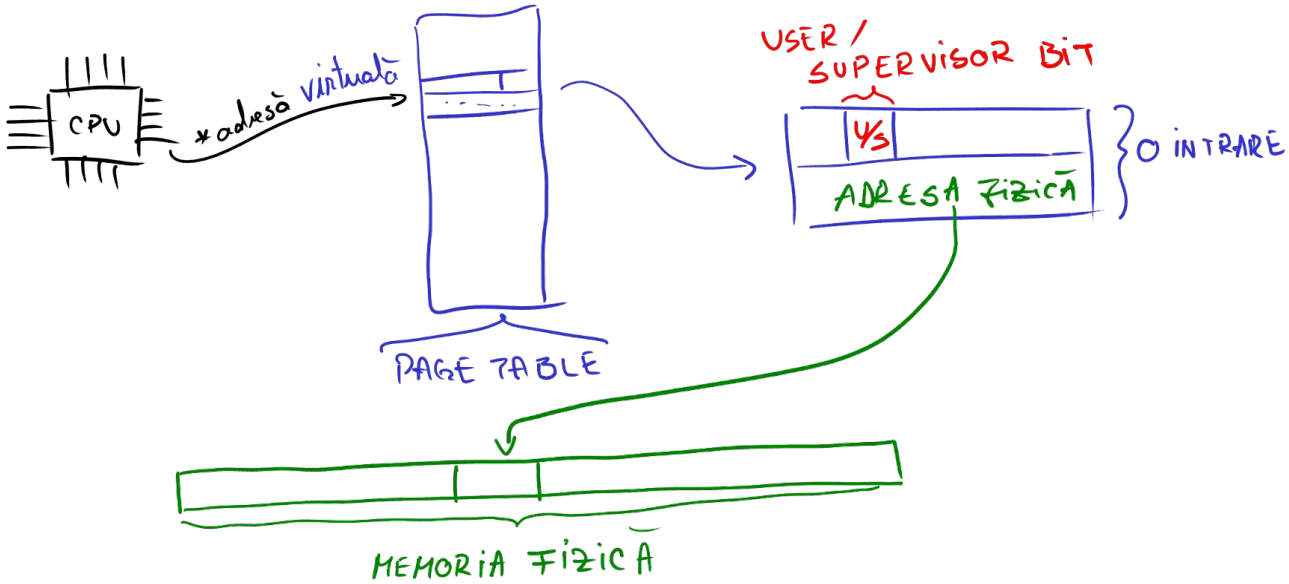
Memoria Virtuala



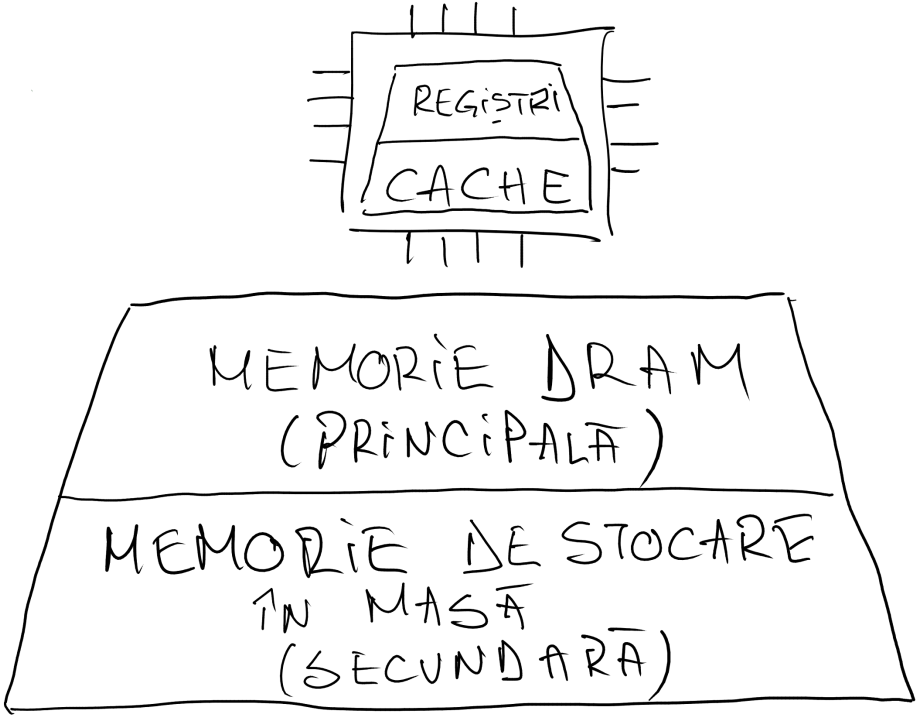
Memoria Virtuala



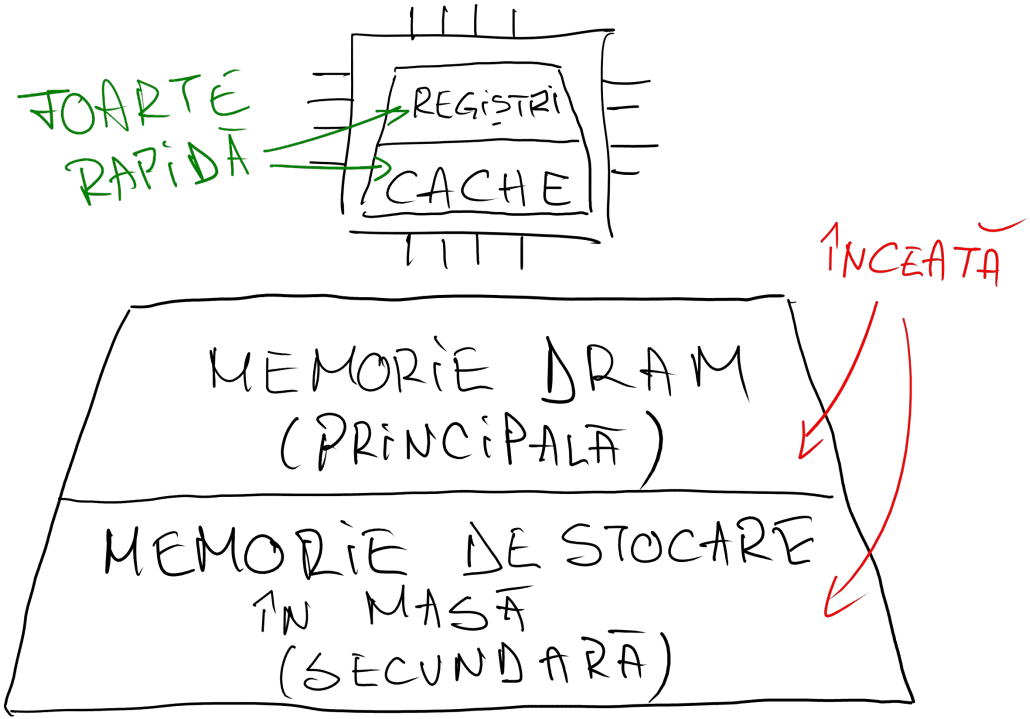
Memoria Virtuala



Ierarhia memoriei în sistem



Ierarhia memoriei în sistem

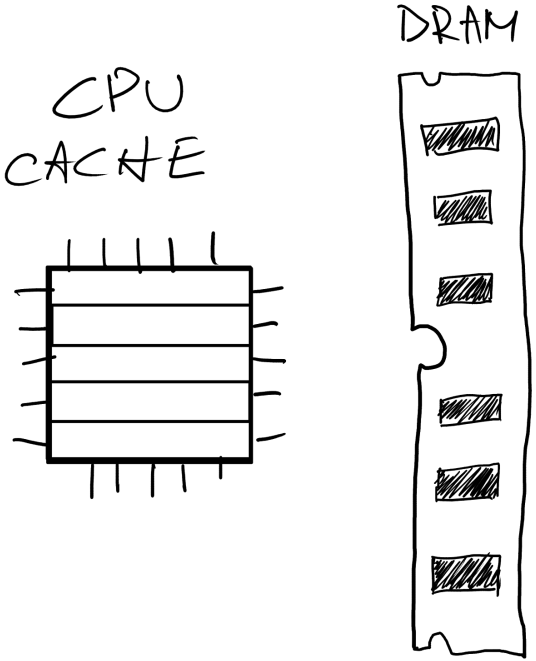


Side-channel Attacks

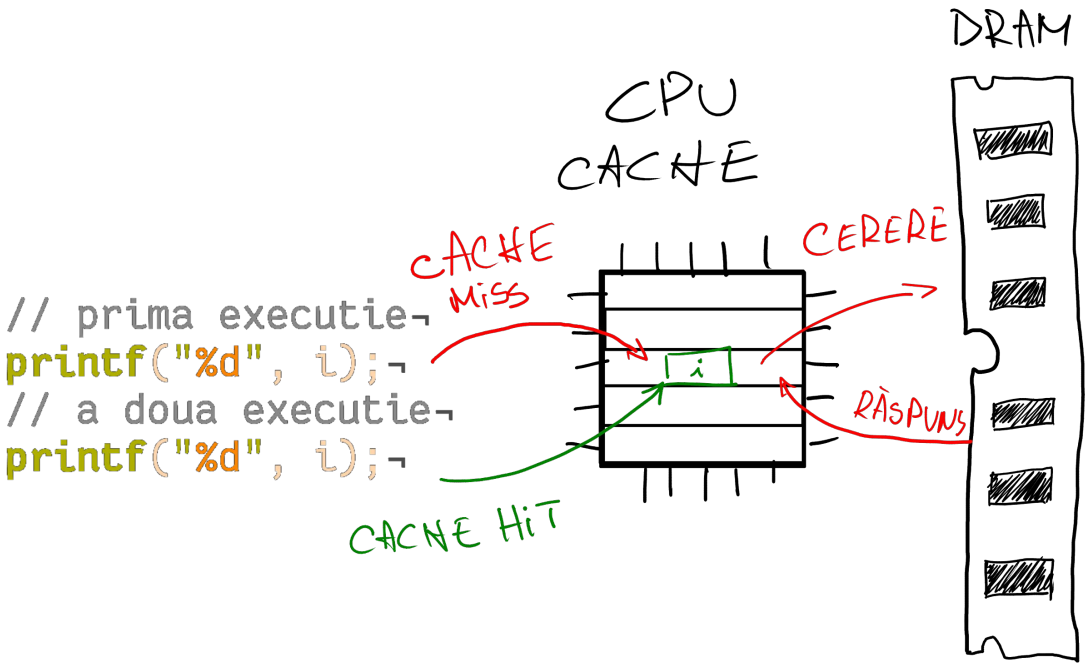
- Execuția corectă a unui program NU implică și că acesta s-a executat în siguranță
- Pot apărea scurgeri de informație din cauza arhitecturii la nivel hardware
- Nu sunt necesare niciun fel de greșeli de implementare la nivel software
- Pentru extragerea de informație, pot fi exploatate efecte secundare neintenționate din:
  - Consumul energetic ⚡
  - Timpul de execuție ⌛
  - Memoria Cache a CPU

Memoria Cache

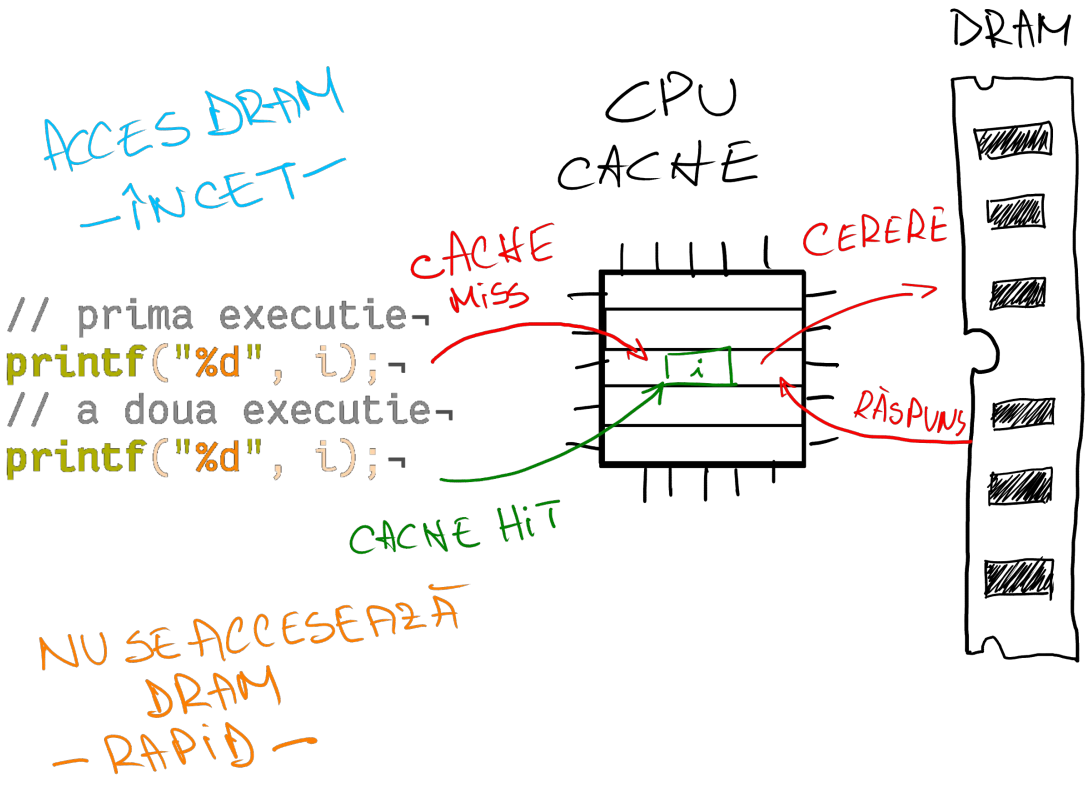
```
// prima executie¬
printf("%d", i);¬
// a doua executie¬
printf("%d", i);¬
```



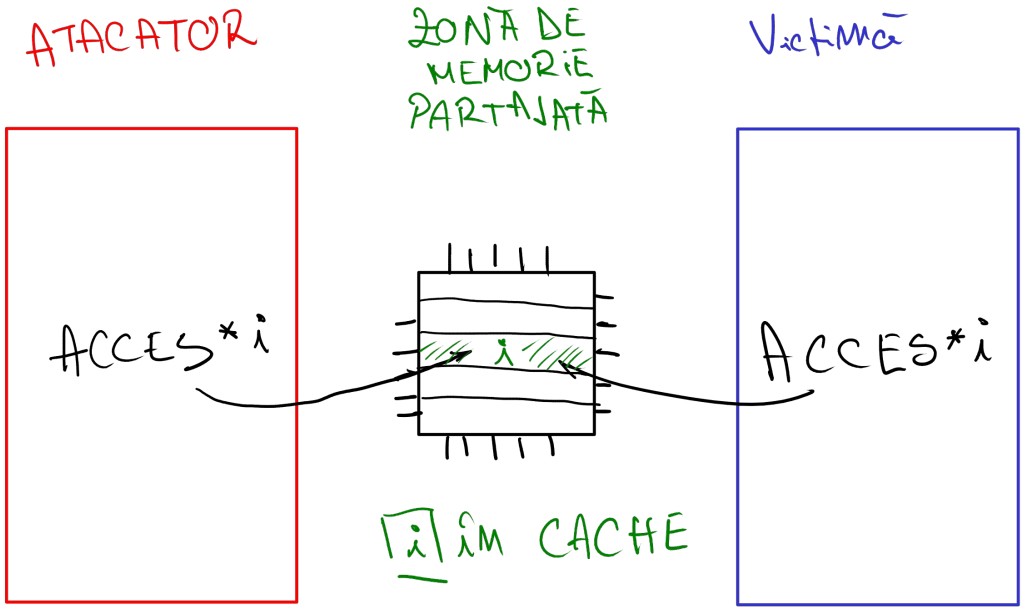
Memoria Cache



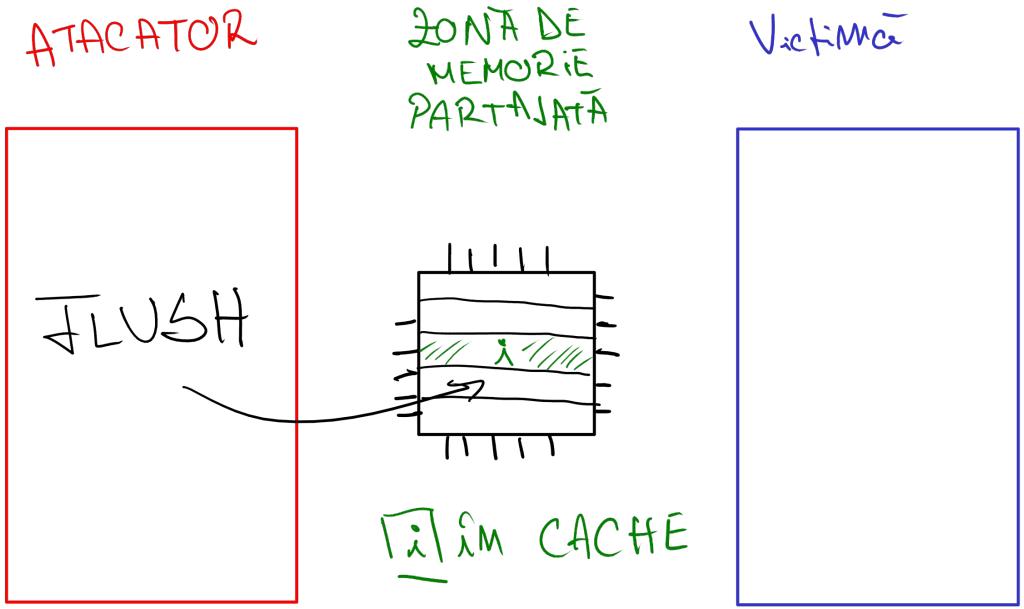
Memoria Cache



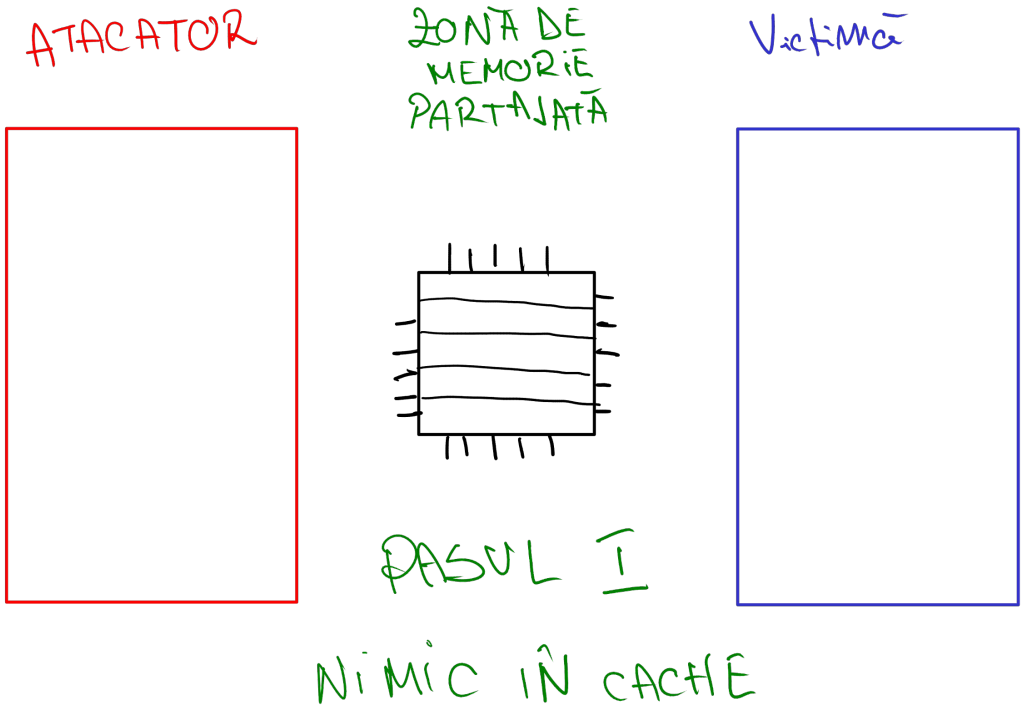
Flush and Reload



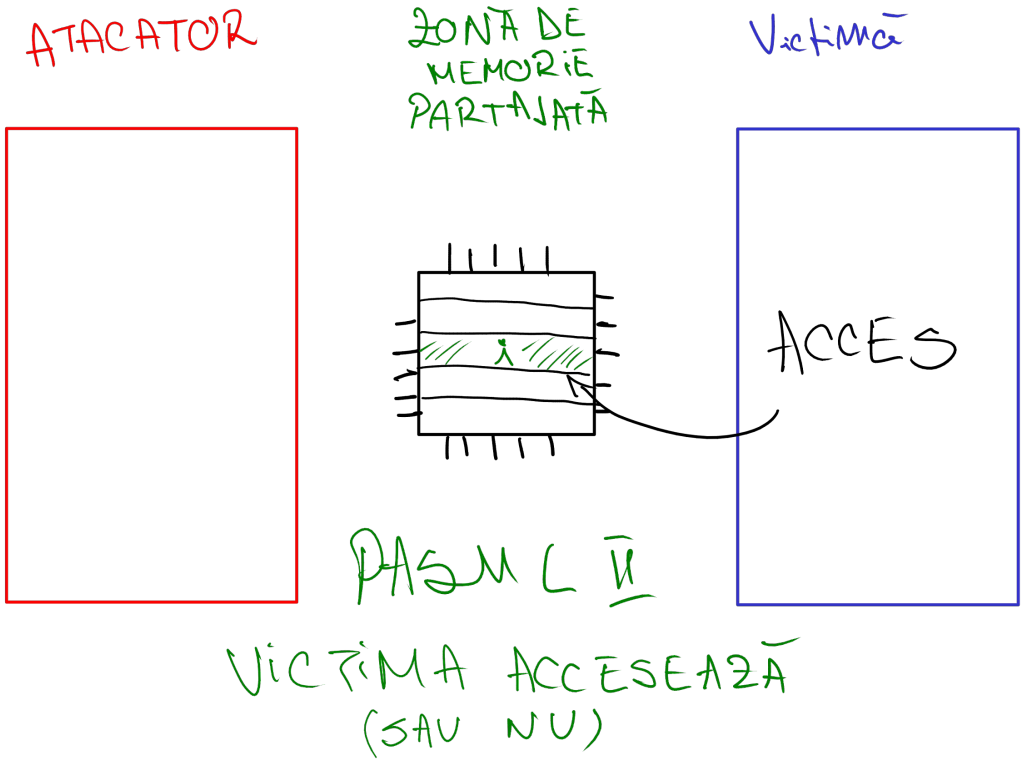
Flush and Reload



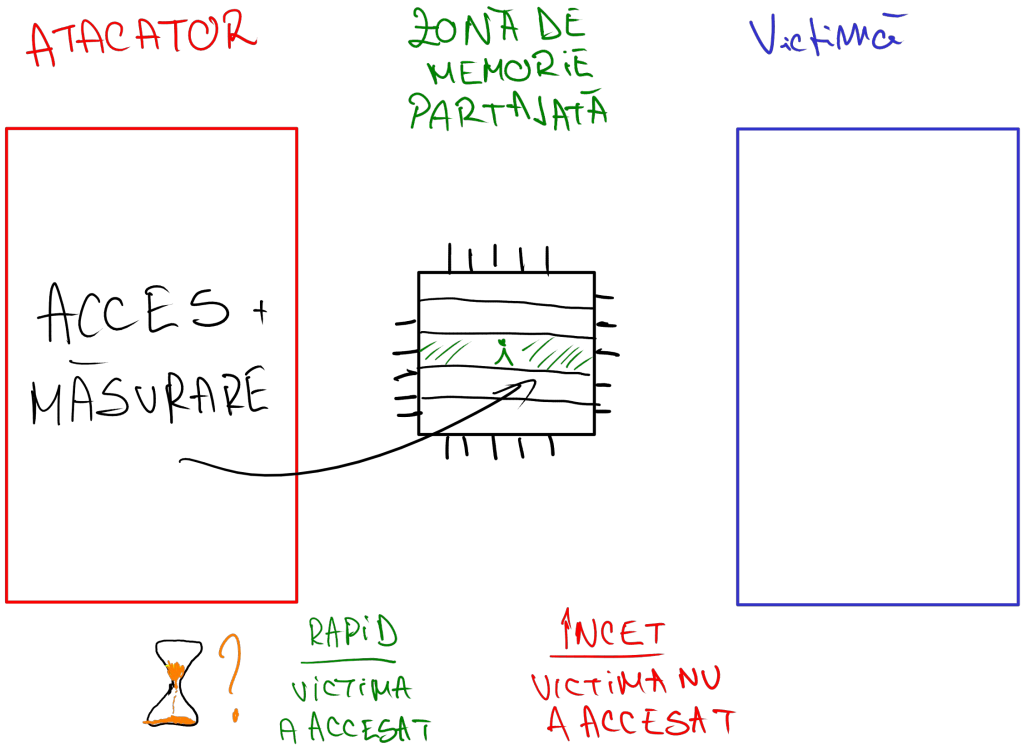
Flush and Reload



Flush and Reload

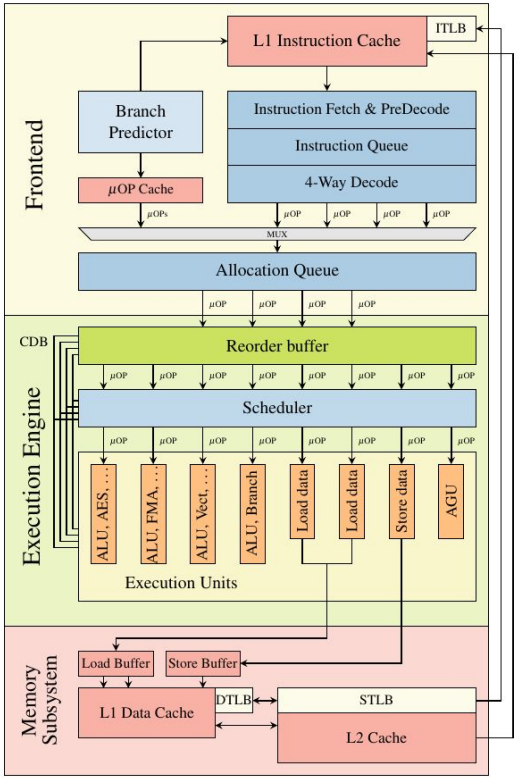


Flush and Reload



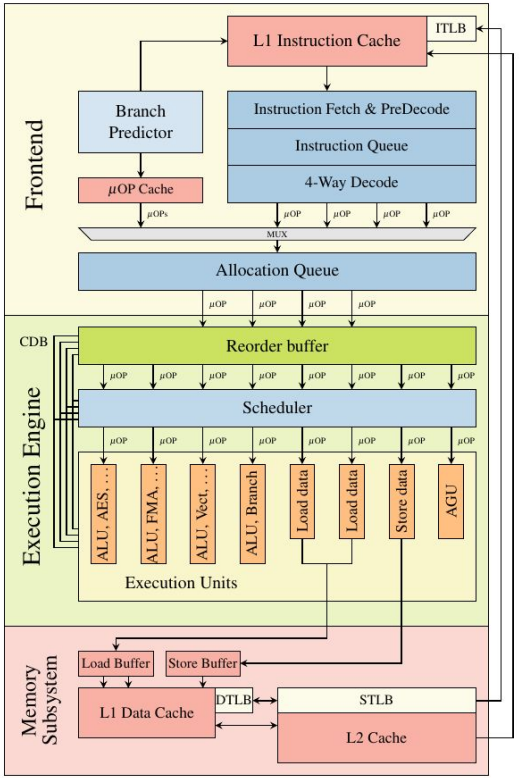
Execuția Out-of-Order și Speculativă

- Instrucțiunile sunt preluate și decodate în microinstrucțiuni în frontend
- Trimise pentru execuție către motorul de execuție (backend)
- Reordonare, redirecționare către nucleele de execuție



Execuția Out-of-Order și Speculativă

- Execuție în paralel, out-of-order
- Rezultatele se salvează dacă toate dependințele au fost executate cu succes
- În caz contrar instrucțiunile executate în avans (**Speculativ**) sunt ignorate și se revine la o stare anterioară



Meltdown

- Se poate citi orice zonă din memorie prin intermediul execuției speculative
- Race condition între verificarea drepturilor de acces și accesarea memoriei.
- Posibil deoarece întreaga memorie kernel se încarcă în spațiul virtual al unui proces (pre KAISER)
- Mitigare software: KAISER. Presupune o separare mai strictă a zonei kernel de userland
- Principala arhitectura afectată: Intel

Meltdown

```
// acces ilegal -> segmentation fault
char data = *(char*) 0xffffcbaf0000c000;

// executat speculativ
probe[data * 4096] ^= 1;
```

Meltdown

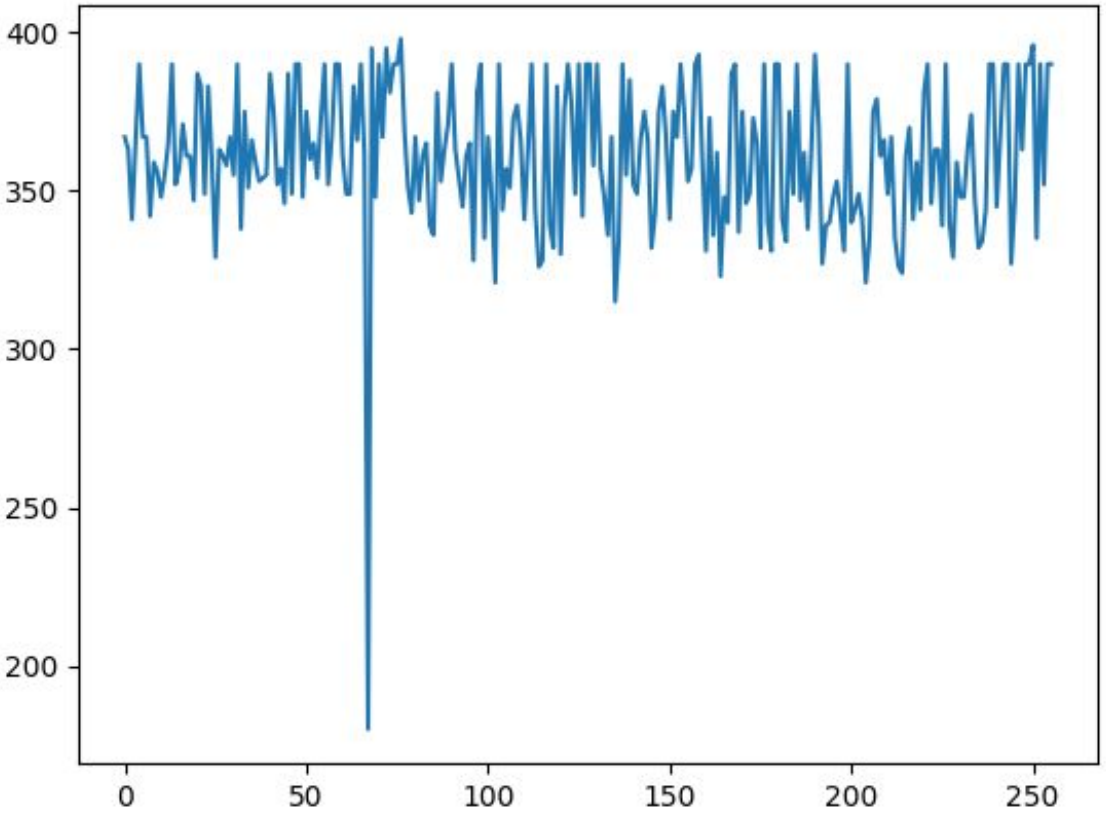
```
// acces ilegal -> segmentation fault
char data = *(char*) 0xffffcbaf0000c000;

// executat speculativ
probe[data * 4096] ^= 1;
```

- Se tratează excepțiile apărute pentru a preveni crash-ul
- Flush and Reload
- Index-ul cache-hit-ului corespunde cu valoarea transmisă
- Se pot transmite astfel 8 bytes o dată

Meltdown

Să considerăm secretul “*Cheia secreta din kernel*”

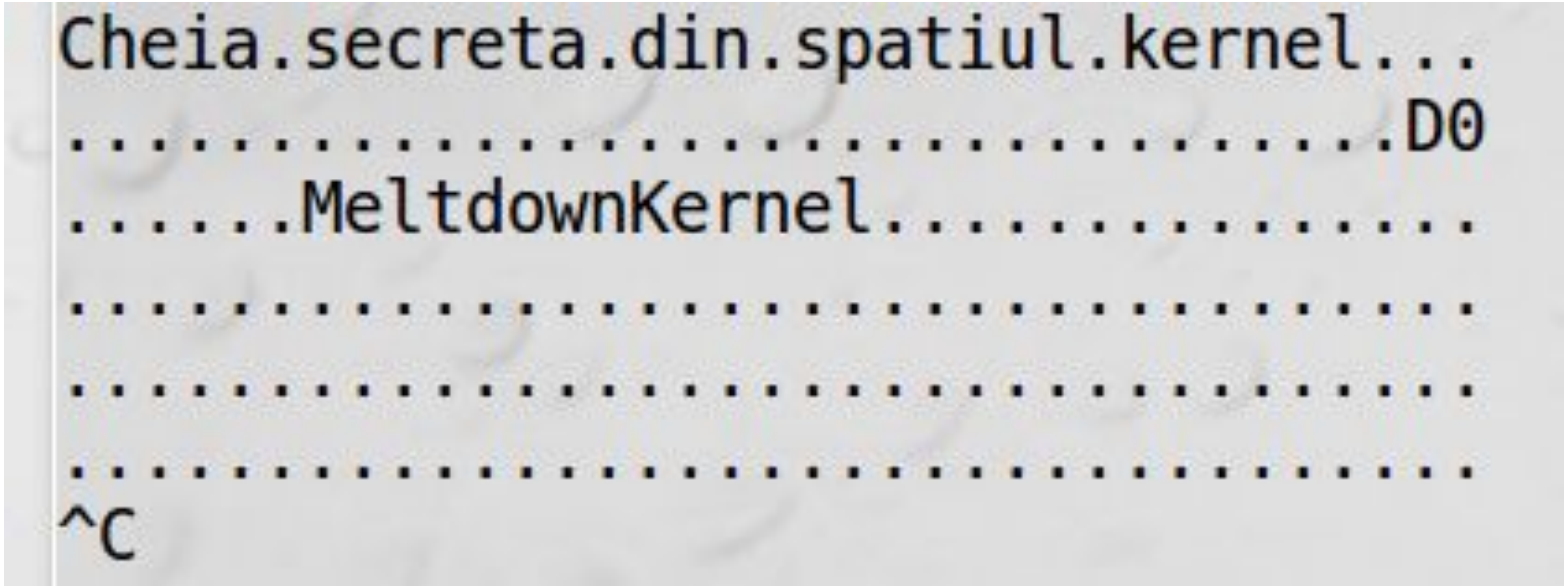
Meltdown



Cache-hit la index-ul 67 corespunzător literei ‘C’

Meltdown Dump

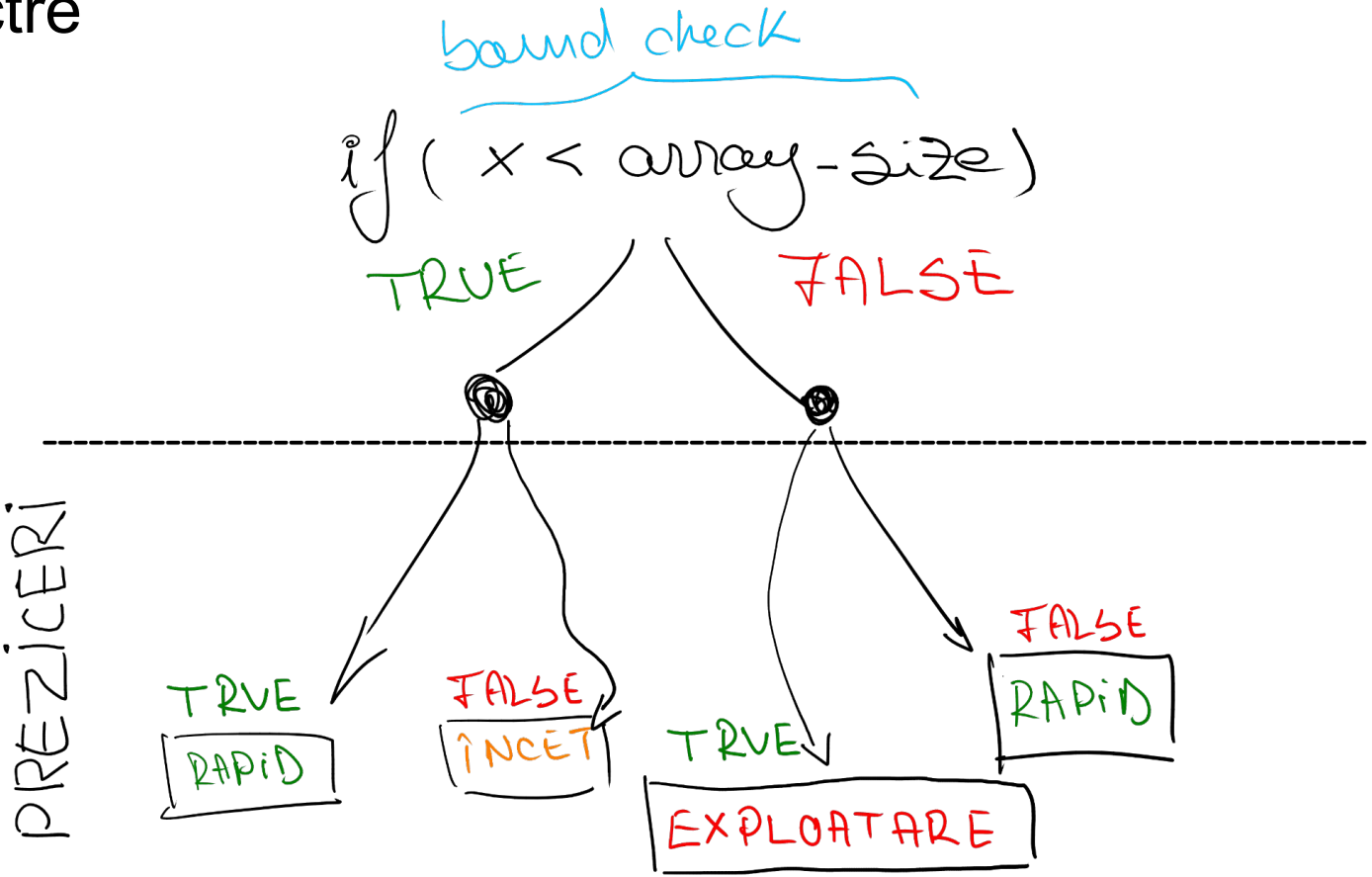
- Se repetă pentru toate adresele de interes



Spectre

- Clasa largă de atacuri asemănătoare cu Meltdown
- Afectează majoritatea arhitecturilor folosite (Intel, AMD, ARM etc.)
- Foarte greu de mitigat (variantă nouă documentată în 2022)
- Permite citirea arbitrară a unor zone de memorie la care victima are acces prin intermediul unei zone de memorie partajată
- Exploatează
  - Branch Predictor-ul (variantă 1 - Bound Check Bypass)
    - Executarea speculativă a unor instrucțiuni specifice din ramuri prezise de branch predictor
  - Branch Target Buffer (variantă 2 - Branch Target Injection)

Spectre


bound check
if (x < array-size)
TRUE
FALSE
PREZICERI
TRUE
RAPID
FALSE
ÎNCET
TRUE
EXPLOATARE
FALSE
RAPID

Spectre

```
static char *secret = "Secret Spectre";
```

Spectre

```
static char *secret = "Secret Spectre";

int array_size = 10, junk;
uint8_t array[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

uint8_t probe[256 * 4096];
```

Spectre

```
static char *secret = "Secret Spectre";


int array_size = 10, junk;
uint8_t array[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};


uint8_t probe[256 * 4096];


if (x < array_size) {
  probe[array[x] * 4096] ^= 1;
} else {
  printf("ramura else");
}
```

DEMO