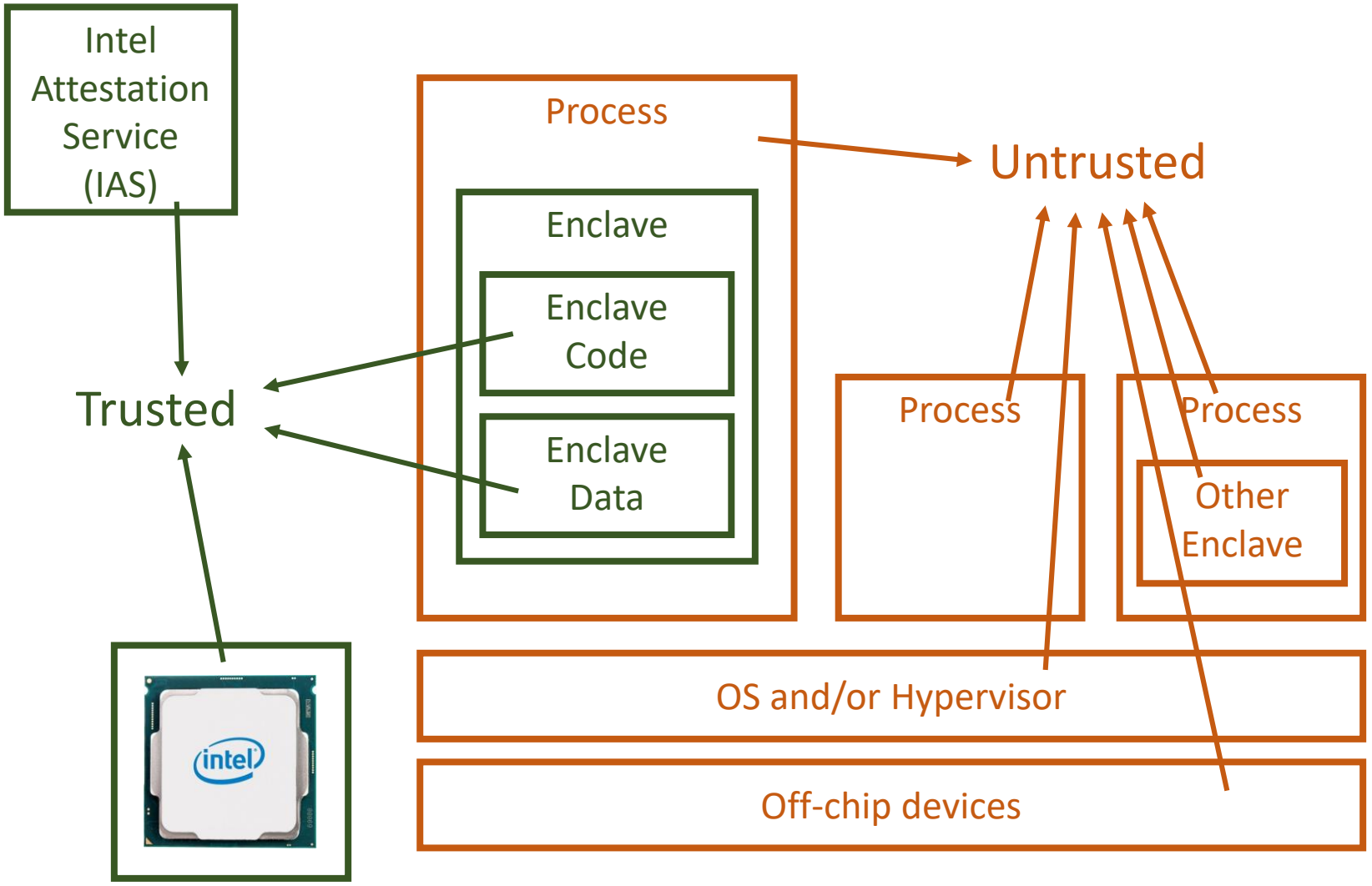
Hardware Enclave Attacks

CS261

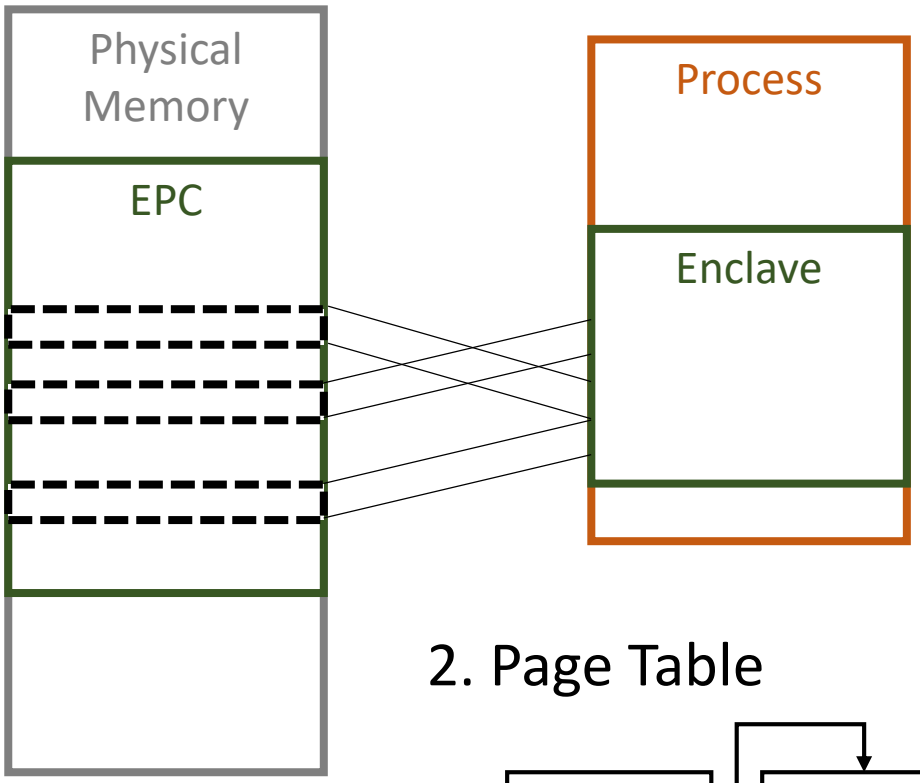
Threat Model of Hardware Enclaves



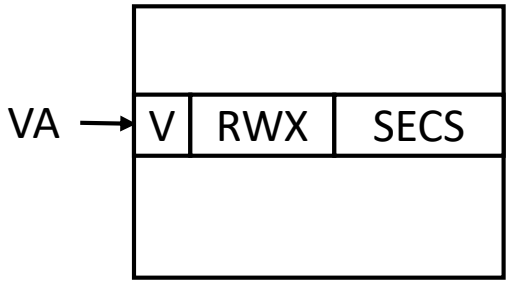
Attacks on Hardware Enclaves

- Attacks on Intel services:
  - Traditional server-based attacks (not interesting)
- Attacks on enclave code:
  - Exploiting software vulnerabilities
  - Interesting API-based attacks: Iago attacks (ASPLOS'13)
- Attacks on Intel CPUs:
  - Cache timing side channels, Spectre / Meltdown (Foreshadow)
  - Controlled-channel attacks

Enclave Page Permissions

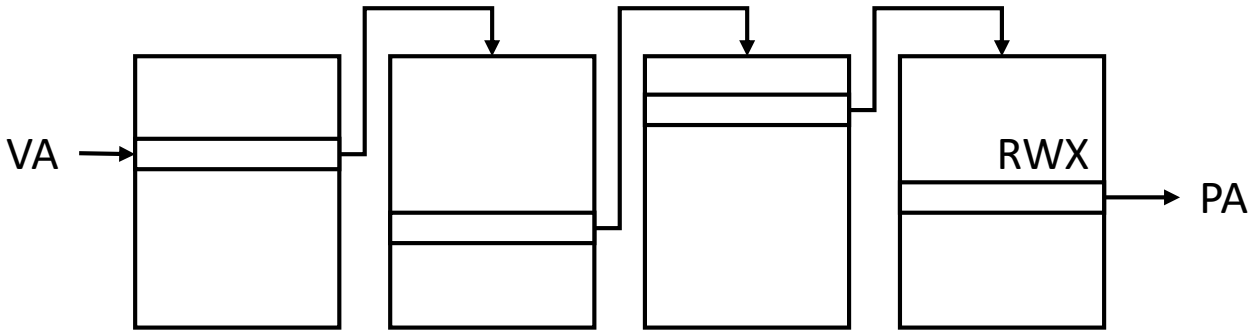


1. EPCM

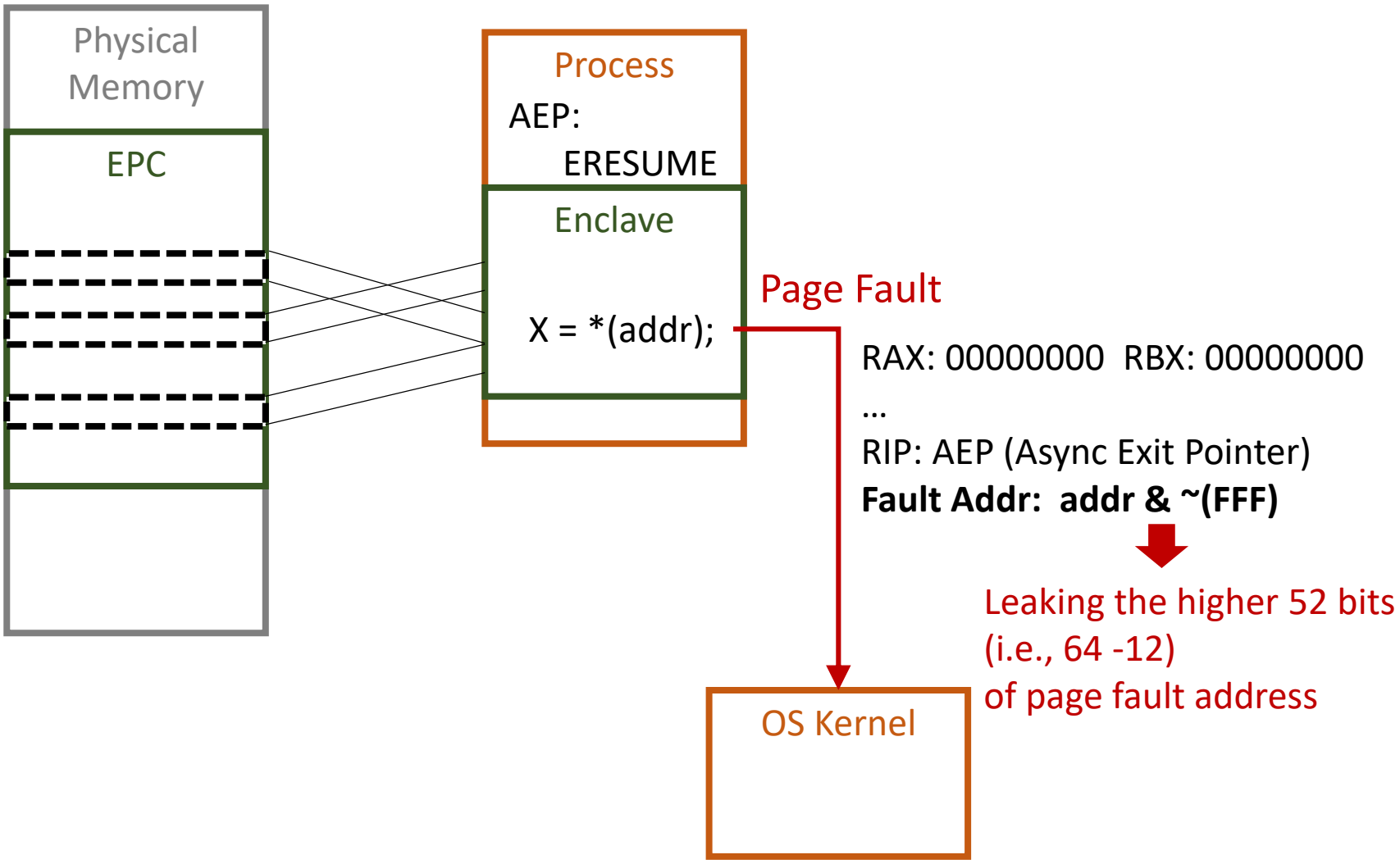


Enclave Page Permission
= EPCM[RWX] AND PT[RWX]

2. Page Table



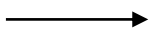
Page Faults in Enclaves

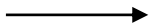


Target Code

- Input-dependent branches

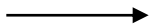
```
if (secret & 0x1)    process_one();
else  process_zero();
```


Page A


Page B

- Input-dependent data access

```
data_array[secret << 12] = 1;
```

secret = 0  Page X

secret = 1  Page X + 1

secret = 2  Page X + 2

Distinguishing Same Page Addresses

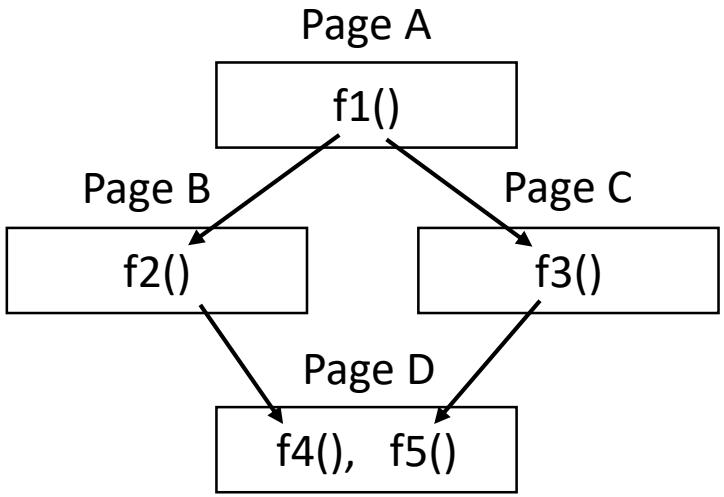
```
f1() {
    …
    f2();
    …
    f3();
    …
}
```

```
f2() {
    …
    f4();
    …
}
```

```
f3() {
    …
    f5();
    …
}
```

```
f4() {
    …
}
```

```
f5() {
    …
}
```



Distinguishing Same Page Addresses

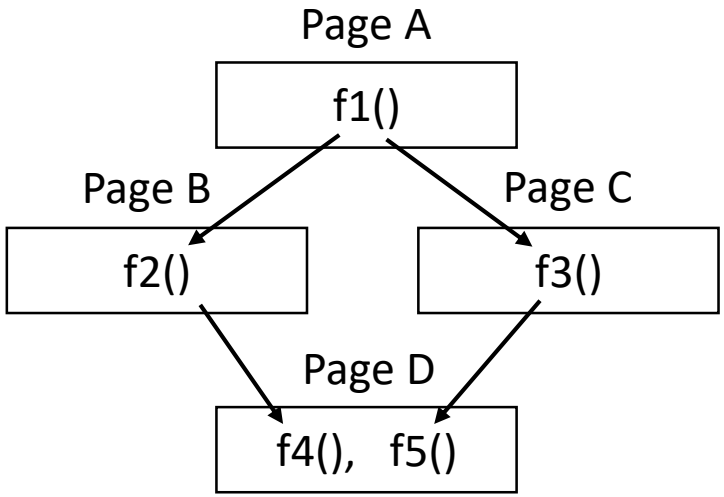
```
f1() {
   …
   f2();
   …
   f3();
   …
}
```

```
f2() {
   …
   f4();
   …
}
```

```
f3() {
   …
   f5();
   …
}
```

```
f4() {
   …
}
```

```
f5() {
   …
}
```



Page addresses:

A

Distinguishing Same Page Addresses

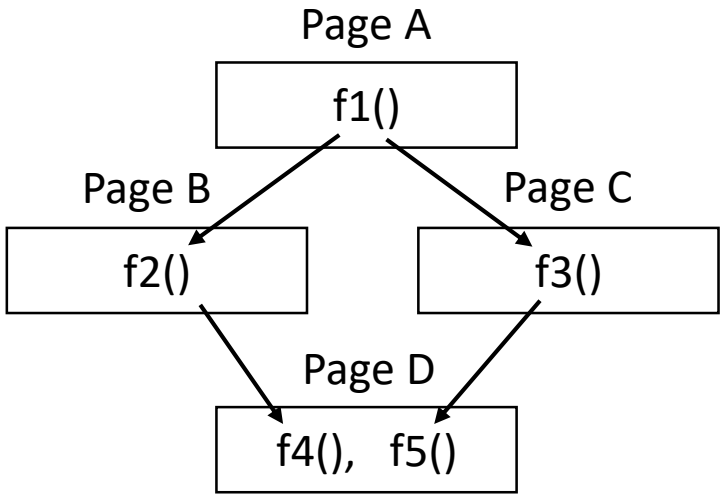
```
f1() {
   …
   f2();
   …
   f3();
   …
}
```

```
f2() {
   …
   f4();
   …
}
```

```
f3() {
   …
   f5();
   …
}
```

```
f4() {
   …
}
```

```
f5() {
   …
}
```



Page addresses:

A B

Distinguishing Same Page Addresses

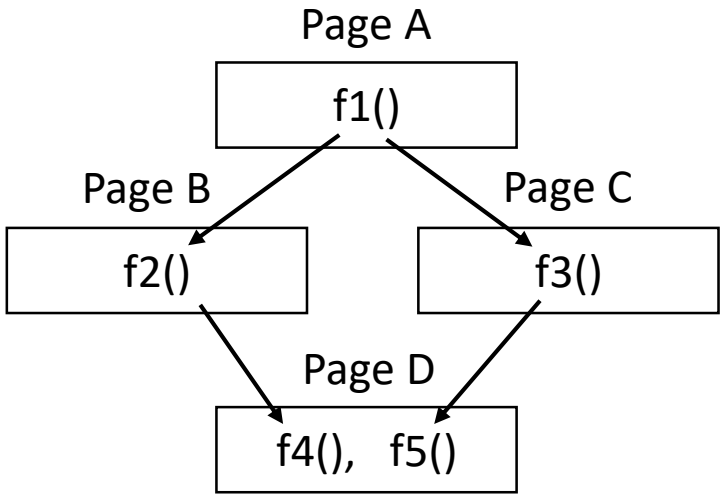
```
f1() {
   …
   f2();
   …
   f3();
   …
}
```

```
f2() {
   …
   f4();
   …
}
```

```
f3() {
   …
   f5();
   …
}
```

```
f4() {
   … ←
}
```

```
f5() {
   …
}
```



Page addresses:

A B D

Distinguishing Same Page Addresses

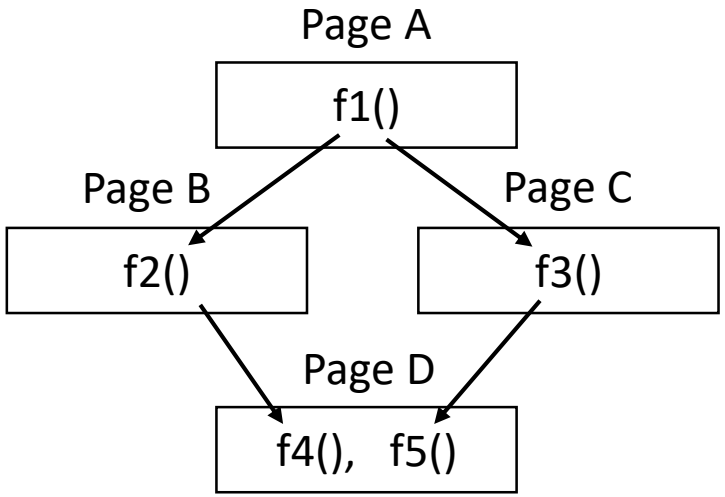
```
f1() {
   …
   f2();
   …
   f3();
   …
}
```

```
f2() {
   …
   f4();
   …
}
```

```
f3() {
   …
   f5();
   …
}
```

```
f4() {
   …
}
```

```
f5() {
   …
}
```



Page addresses:

A B D B A

Distinguishing Same Page Addresses

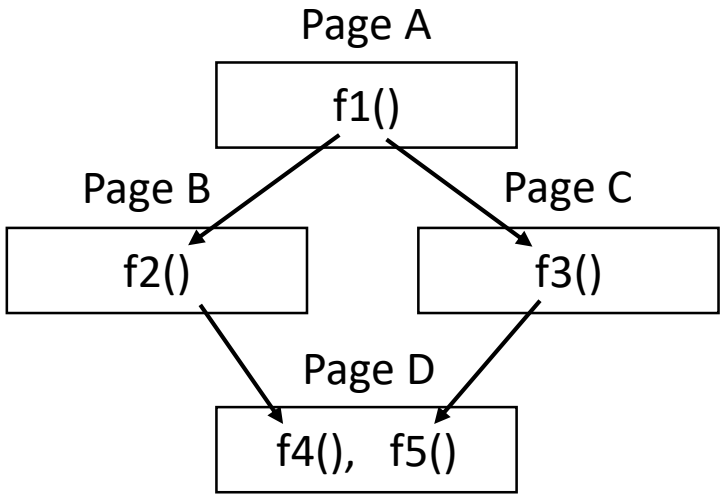
```
f1() {
   …
   f2();
   …
   f3();
   …
}
```

```
f2() {
   …
   f4();
   …
}
```

```
f3() {
   …
   f5();
   …
}
```

```
f4() {
   …
}
```

```
f5() {
   …
}
```



Page addresses:

A B D B A C

Distinguishing Same Page Addresses

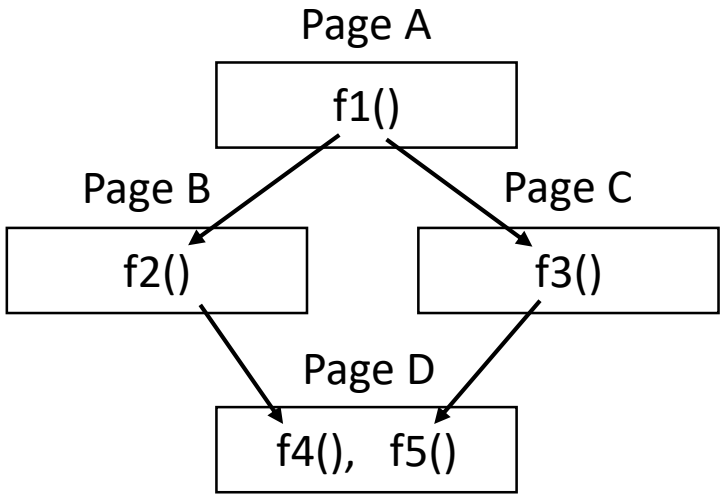
```
f1() {
    …
    f2();
    …
    f3();
    …
}
```

```
f2() {
    …
    f4();
    …
}
```

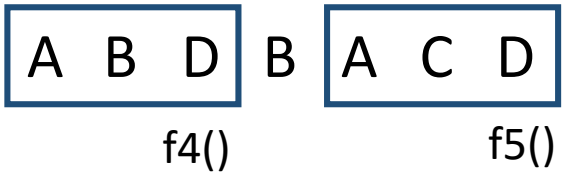
```
f3() {
    …
    f5();
    …
}
```

```
f4() {
    …
}
```

```
f5() {
    …
}
```

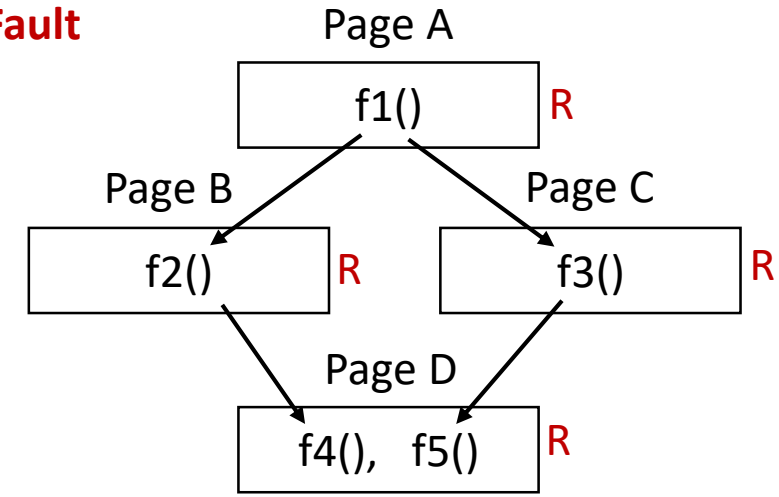


Page addresses:



Update the Page Table

```
f1() {
   …
   f2();
   …
   f3();
   …
}
```




```
f2() {
   …
   f4();
   …
}

f4() {
   …
}
```

```
f3() {
   …
   f5();
   …
}

f5() {
   …
}
```

Page addresses:

A

Update the Page Table

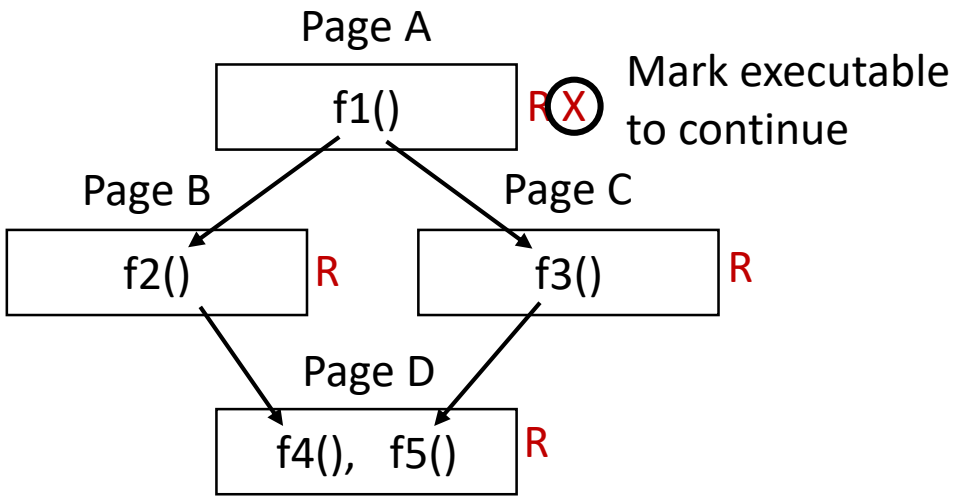
```
f1() {
   …
   f2();
   …
   f3();
   …
}
```

```
f2() {
   …
   f4();
   …
}
```

```
f3() {
   …
   f5();
   …
}
```

```
f4() {
   …
}
```

```
f5() {
   …
}
```



Page addresses:

A B

Update the Page Table

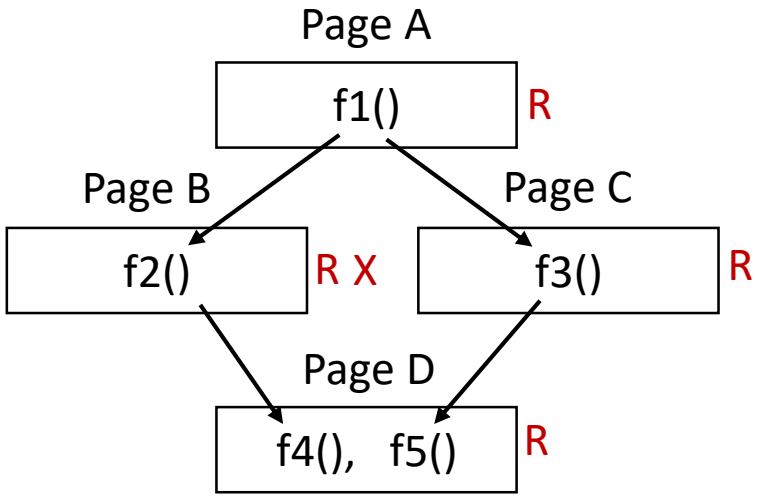
```
f1() {
   …
   f2();
   …
   f3();
   …
}
```

```
f2() {
   …
   f4();
   …
}
```

```
f3() {
   …
   f5();
   …
}
```

```
f4() {
   …
}
```

```
f5() {
   …
}
```



Page addresses:

A B D

Example: Hunspell Checker

- Phase 1: inserts dictionary into hash buckets
- Phase 2: looks up words from a secret document

Hunspell Insertion

Word	Pages
word1	A, D
word2	B, D
word3	A, E
word4	B, D, F

- Hash::add_word(std::string word) {

```
    struct hentry *hp = malloc(…);
    int i = hash(word);
    struct hentry *dp = tableptr[i];
    while (dp->next != NULL) {
        dp = dp->next;
    }
    strcpy(hp->word, word);
    dp->next = hp;
}
```

→ Page(tableptr[i])

→ Page(node 1)

Page(node 2)

…

→ Page(new node)

Hunspell Lookup

Word	Pages
word1	A, D
word2	B, D
word3	A, E
word4	B, D, F

- Hash::lookup(std::string word) {
  int i = hash(word);
  struct hentry *dp = tableptr[i];
  while (dp != NULL) {
  if (!strcmp(hp->word, word))
  return dp;
  dp = dp->next;
  }
  }

Page(tableptr[i])

Page(node 1)

Page(node 2)

…

Match with the oracle

Side Channels vs Controlled Channels

	Cache Side Channels	**Controlled Channels**
Granularity	Cachelines (64-byte)	Pages (4KB)
Noisiness	Highly noisy	Noiseless and Lossless
Synchronization	Two-phase synchronization (e.g., PRIME+PROBE, FLUSH+RELOAD)	No synchronization with the victim
Scope	Common to most platforms	Specific to enclaves
Privileges	Non-root	Need root privileges

Mitigation

- ASLR (Address Space Layout Randomization)?
  - Not working ➔ Can detect entry points and “start-up” patterns
- Self-paging
  - Some architecture (e.g., RISC-V) suggests self-paging in enclaves
  - The OS never gets any page faults
- Detecting attacks
  - Execution time, page fault count, etc
- Forbidding page faults from enclave code ➔ T-SGX

T-SGX (NDSS'17)

- Intel TSX (Transactional Synchronization Extensions)
  - Any fault ➔ abort handler

```
unsigned status;

// Begin a transaction
if ((status = _xbegin()) == _XBEGIN_STARTED) {
   // Run any code
   _xend();
} else {
   // Abort
}
```

- Can forbid all page faults in enclaves (i.e., no paging)

Other Enclave Attacks

- Page table access/dirty bits (USENIX‘17)
  - Recently read ➔ access bit; Recently written ➔ dirty bit
  - Can be observed without page faults
- Branch Predictor States (USENIX’17)
  - Enclave and non-enclave code shares branch predictor states
  - Can observe which branches are taken
- Addresses on memory bus (CCS’13)
  - Every memory command (read / write) is visible on bus
  - Can observe with a DIMM interposer

Questions?

Hardware Enclave Attacks