Defending against transient execution attacks

Frank Piessens

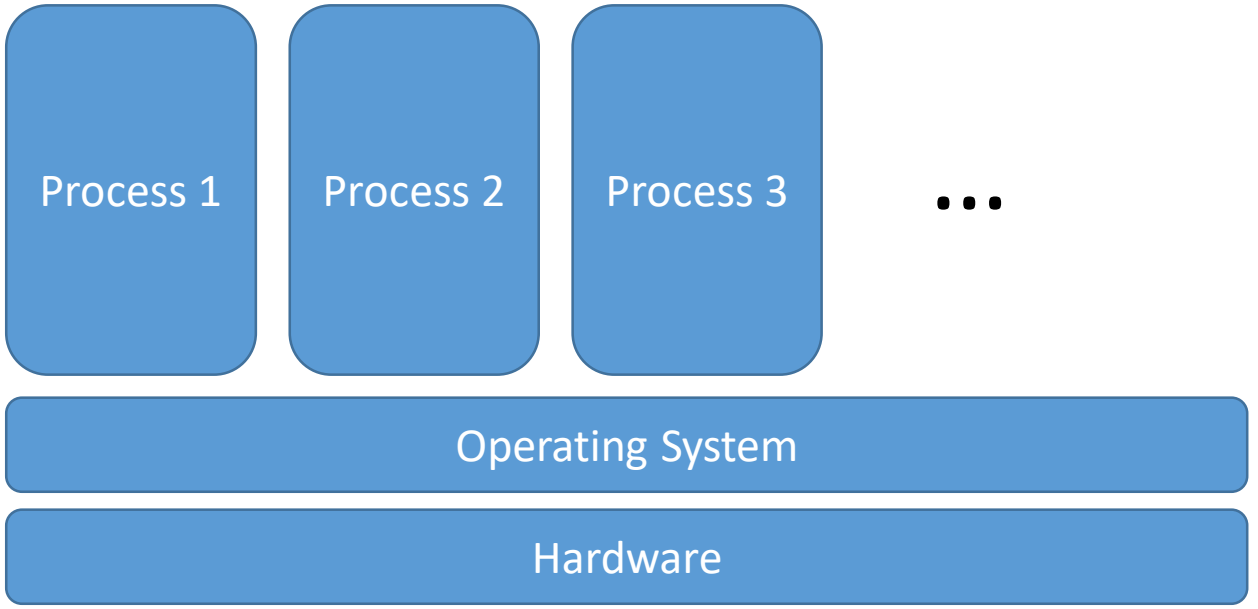
DIMVA

June 20, 2019

Overview

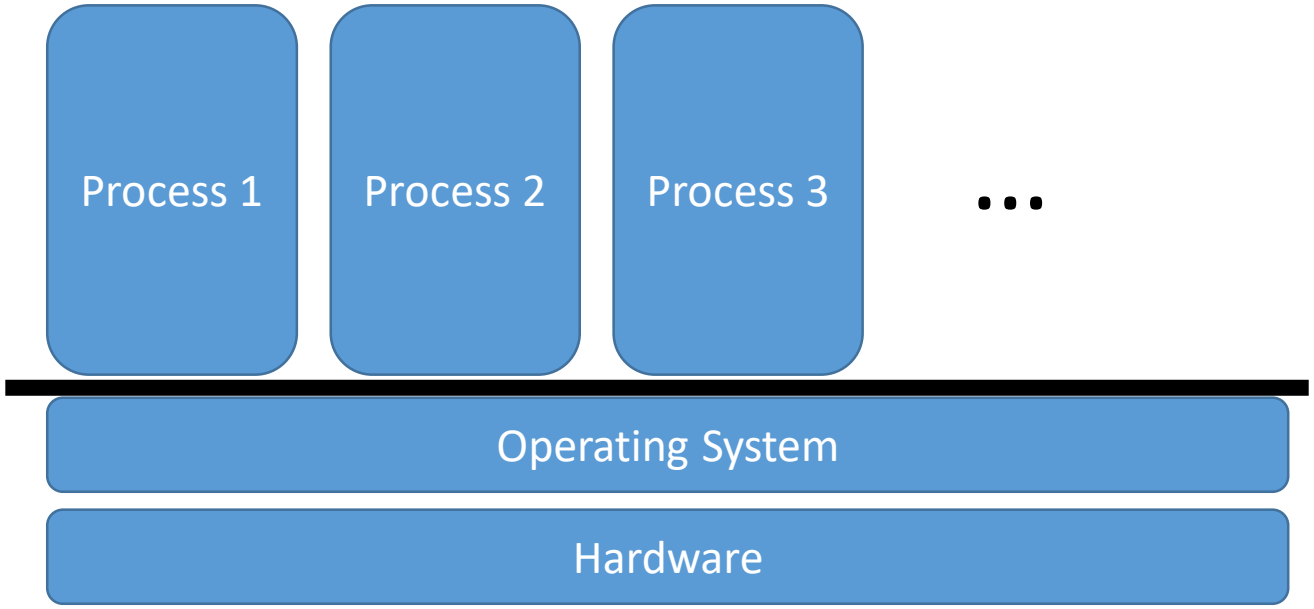
- Introduction: protection mechanisms
- Micro-architectural side-channel attacks by example
  - Side-channel attacks
  - Transient execution attacks
- A systematization of transient execution attacks
- Towards defenses
- Conclusions

Classic hierarchical OS protection



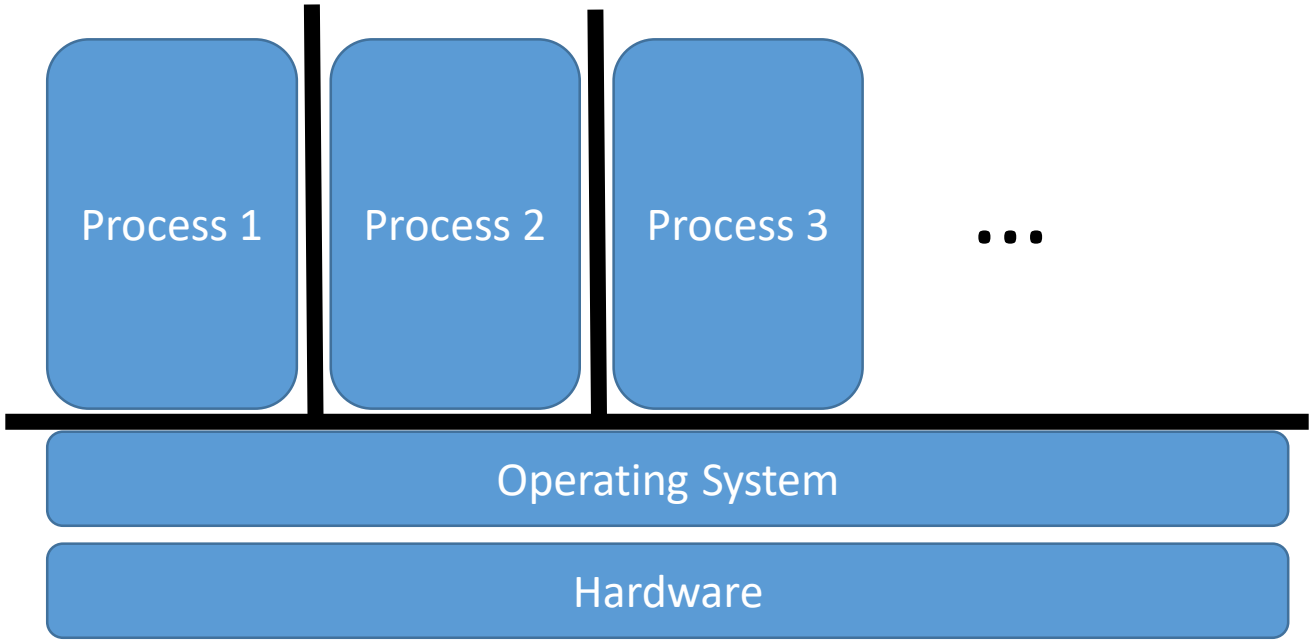
Protecting the kernel: privilege levels

OS is protected from applications by privilege level



Protecting processes: virtual memory

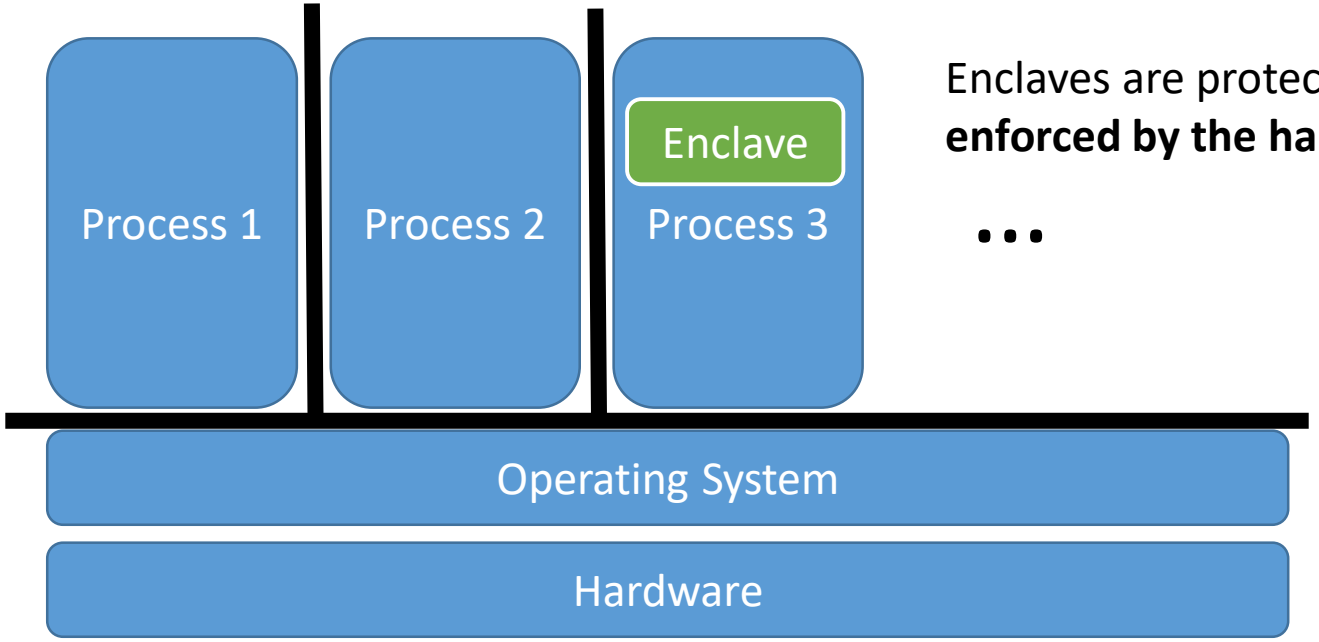
Processes are protected from each other through memory isolation



OS is protected from applications by privilege level

Protecting critical software: enclaves

Processes are protected from each other through memory isolation



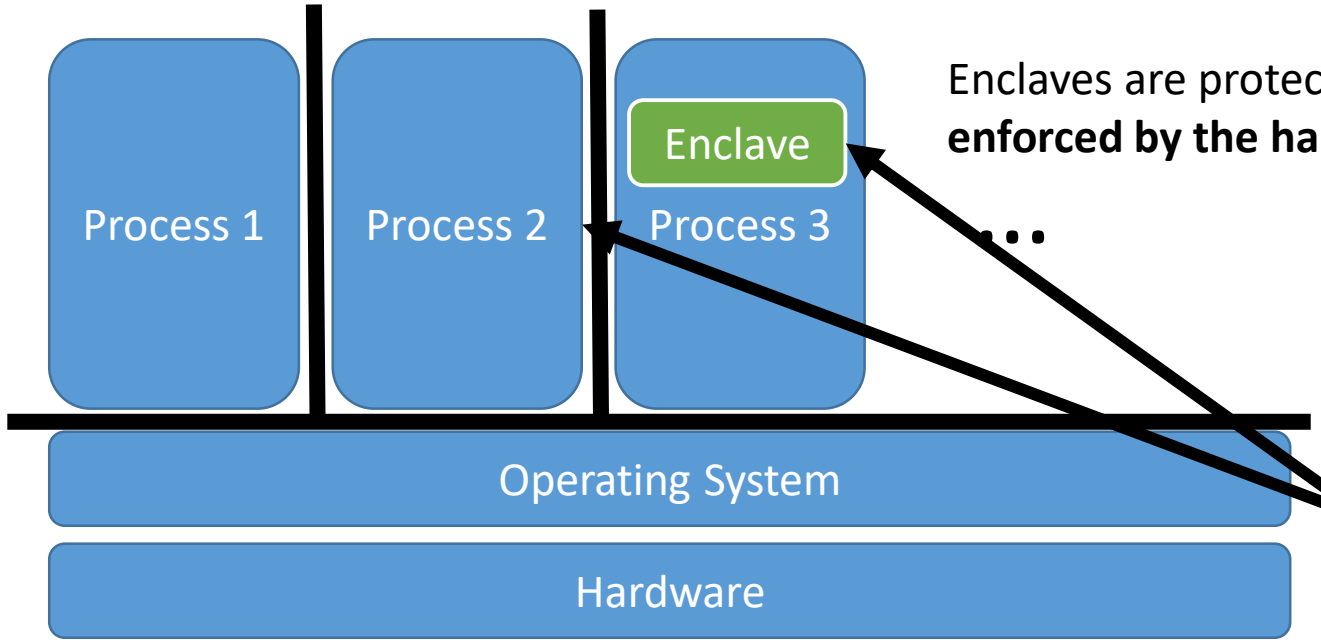
Enclaves are protected by memory isolation **enforced by the hardware only**

...

OS is protected from applications by privilege level

Fine-grained protection: software

Processes are protected from each other through memory isolation



Enclaves are protected by memory isolation **enforced by the hardware only**

…

OS is protected from applications by privilege level

Software can additionally enforce more fine-grained protection:

- Safe languages
- Safe VM's
- WebAssembly
- …

and can implement security relevant checks such as bounds checks

Overview

- Introduction: protection mechanisms
- Micro-architectural side-channel attacks by example
  - Side-channel attacks
  - Transient execution attacks
- A systematization of transient execution attacks
- Towards defenses
- Conclusions

Introduction



- In 2018, micro-architectural attacks have come of age:
  - Meltdown breaks user/kernel isolation
  - Spectre breaks several isolation including process boundaries and software defined boundaries
  - Foreshadow breaks SGX enclave isolation
- Hardware and system software vendors are scrambling to address these attacks
- Regularly waves of new issues
  - E.g. MDS attacks in May 2019

References:

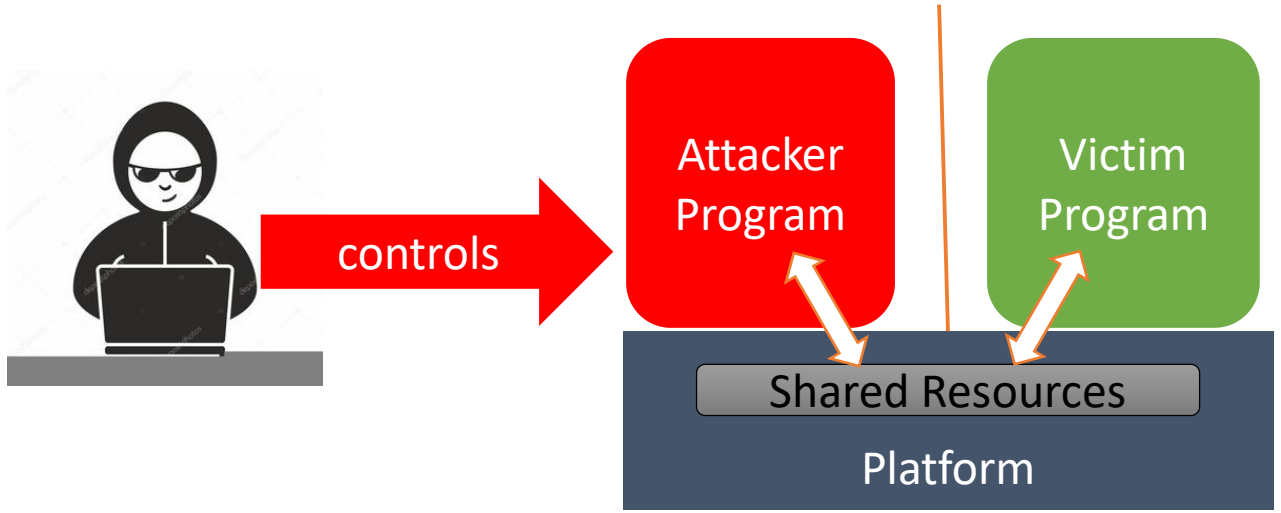
Paul Kocher et al. *Spectre Attacks: Exploiting Speculative Execution*, IEEE S&P 2019

Moritz Lipp et al. *Meltdown: Reading Kernel Memory from User Space*, USENIX Security Symposium 2018

Jo Van Bulck et al. *Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution*, USENIX Security Symposium 2018

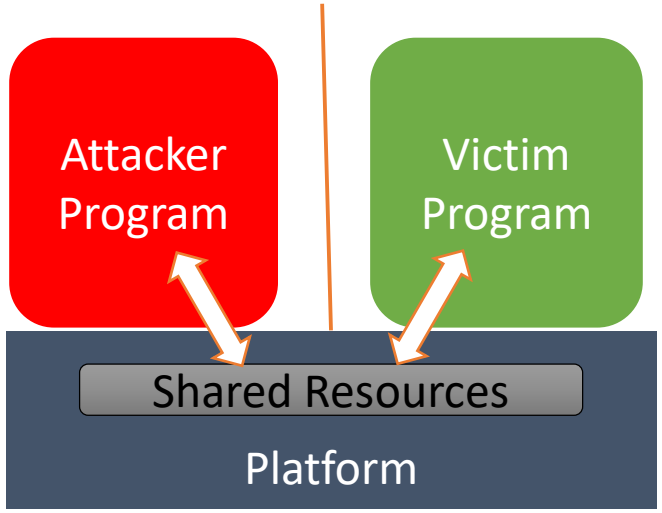
Attacker model: Shared platform attacker

- The attacker can run code on the same platform where victim code is running.
- The objective of the attacker is to learn more about the victim than what one can learn through intended communication interfaces.



Micro-architectural attacks

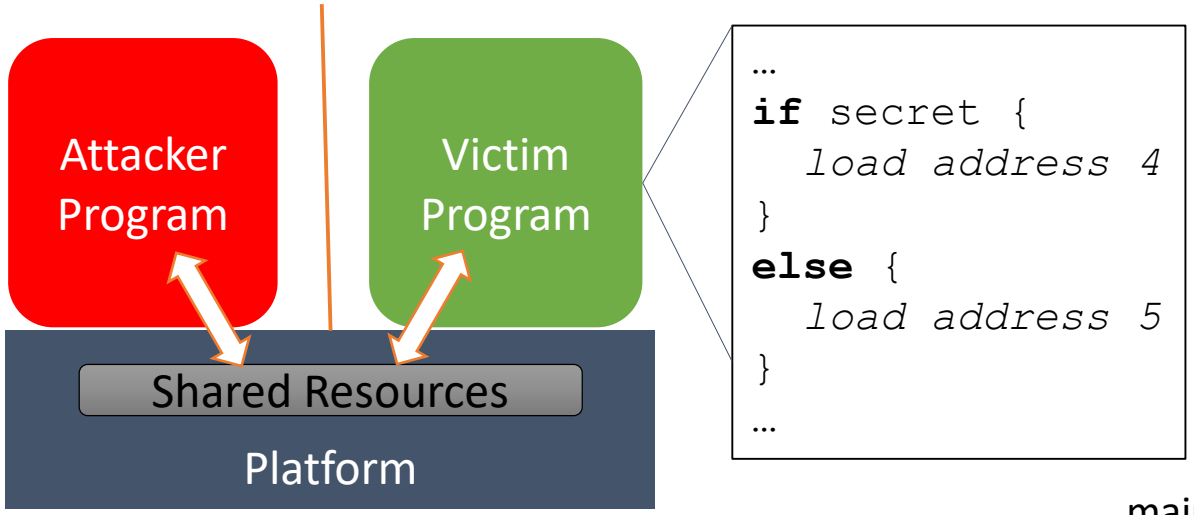
- The attacker uses shared *micro-architectural* resources
  - Architectural state: state as observed by software (memory, registers, …)
  - Micro-architectural state: additional state in the processor implementation, usually for performance (caches, branch predictors, various CPU buffers, …)



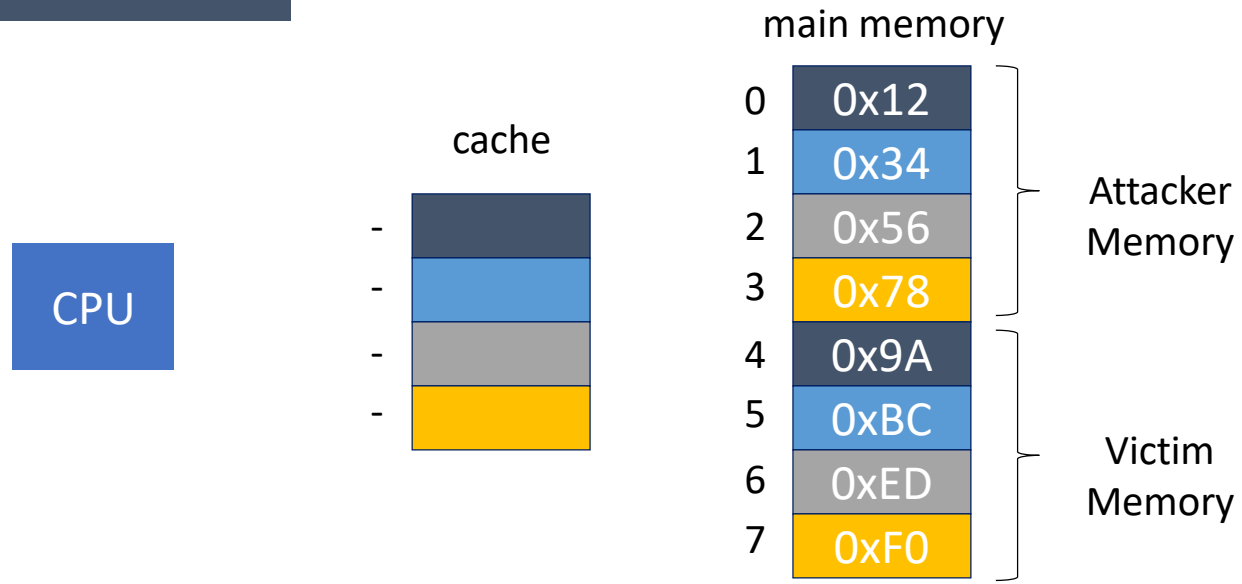
Overview

- Introduction: protection mechanisms
- Micro-architectural side-channel attacks by example
  - Side-channel attacks
  - Transient execution attacks
- A systematization of transient execution attacks
- Towards defenses
- Conclusions

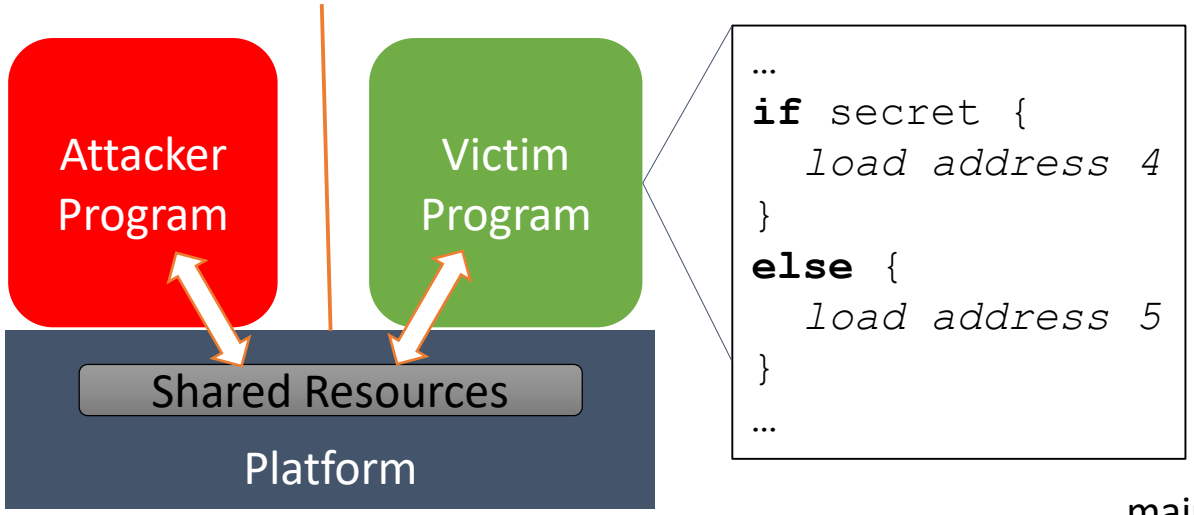
Side-channels: a simple example of a cache-attack



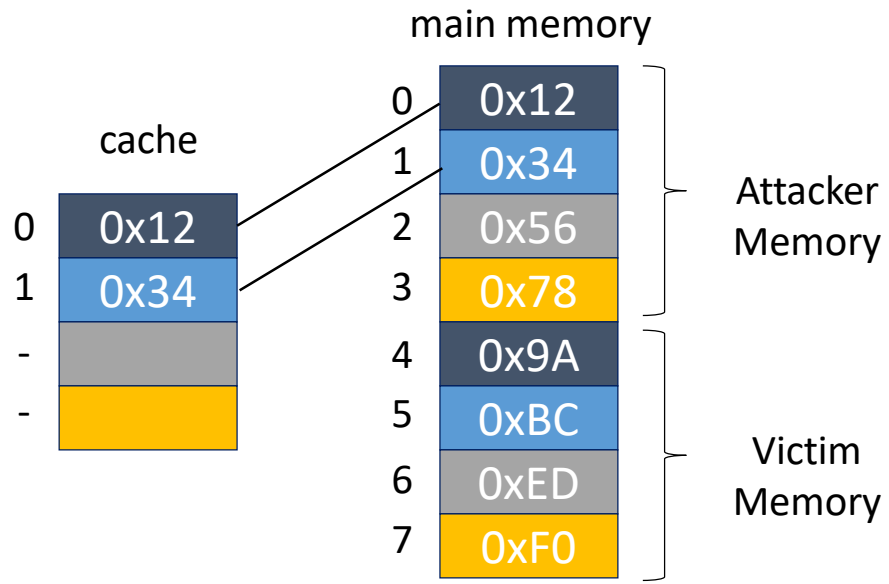
- The shared resources between attacker and victim program include a direct-mapped cache



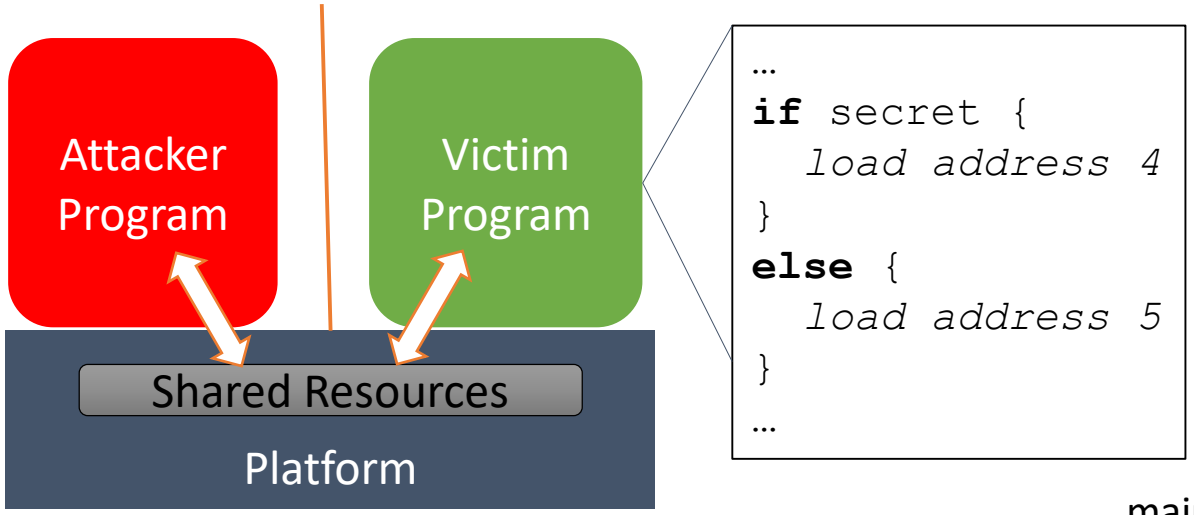
Side-channels: a simple example of a cache-attack



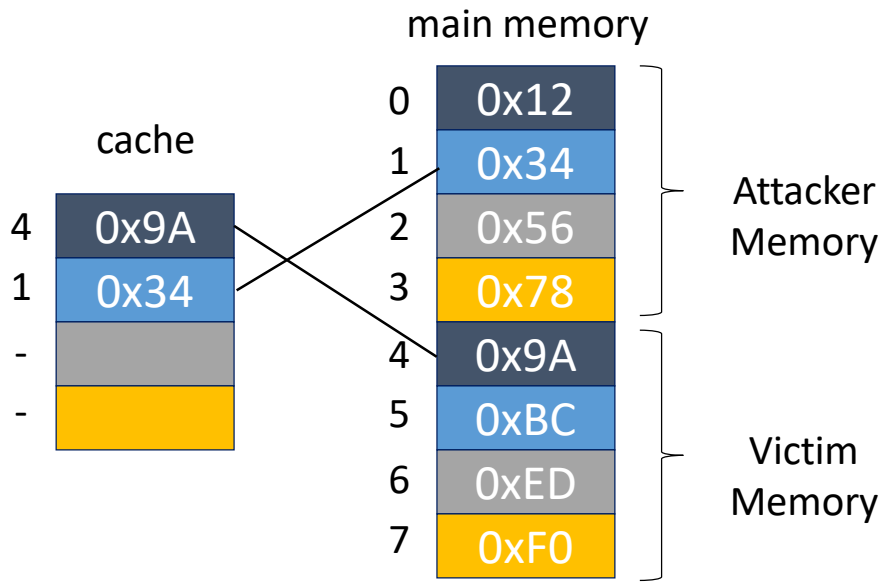
- The shared resources between attacker and victim program include a direct-mapped cache
  - First the attacker program runs and occupies the first two cache lines



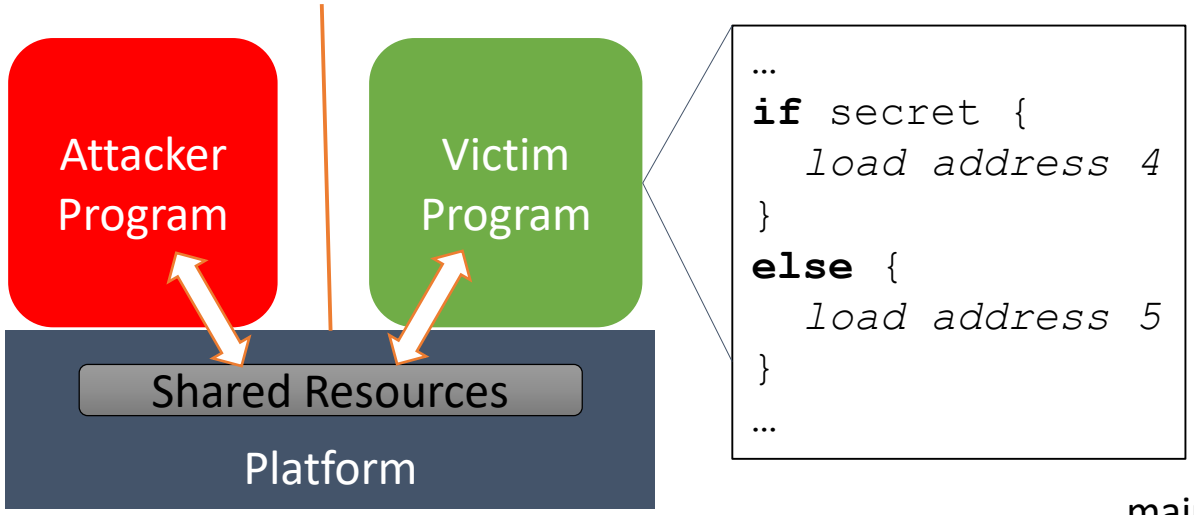
Side-channels: a simple example of a cache-attack



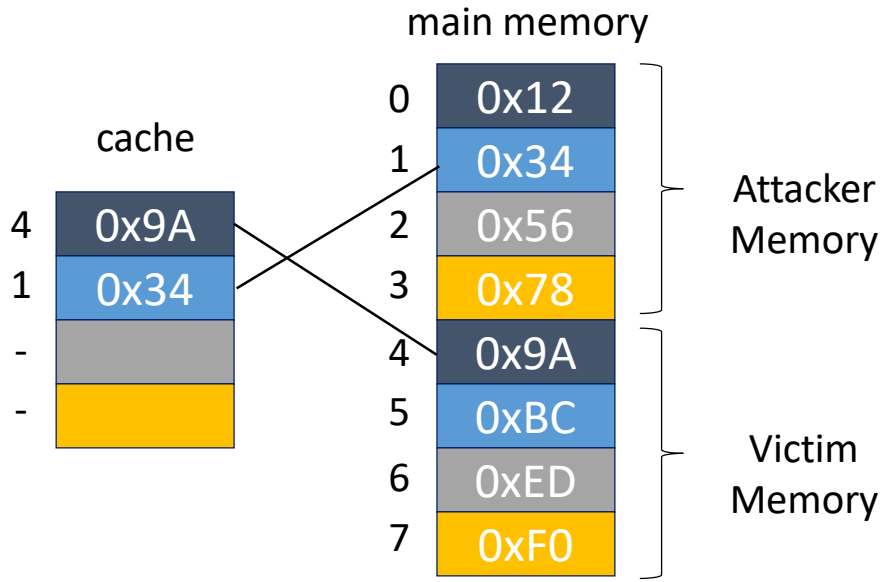
- The shared resources between attacker and victim program include a direct-mapped cache
  - First the attacker program runs and occupies the first two cache lines
  - Next the victim program runs and performs **secret-dependent** memory accesses



Side-channels: a simple example of a cache-attack



- The shared resources between attacker and victim program include a direct-mapped cache
  - First the attacker program runs and occupies the first two cache lines
  - Next the victim program runs and performs **secret-dependent** memory accesses
  - Finally, attacker measures duration of an access to address 0



Cache attacks

- Cache-based side-channel attacks have been understood for quite a while
- Countermeasures exist:
  - At the hardware level, e.g. cache partitioning
  - At the software level, e.g. the crypto constant time model

Qian Ge, Yuval Yarom, David Cock, Gernot Heiser: A survey of microarchitectural timing attacks and countermeasures on contemporary hardware. J. Cryptographic Engineering (2018)

Overview

- Introduction: protection mechanisms
- Micro-architectural side-channel attacks by example
  - Side-channel attacks
  - Transient execution attacks
- A systematization of transient execution attacks
- Towards defenses
- Conclusions

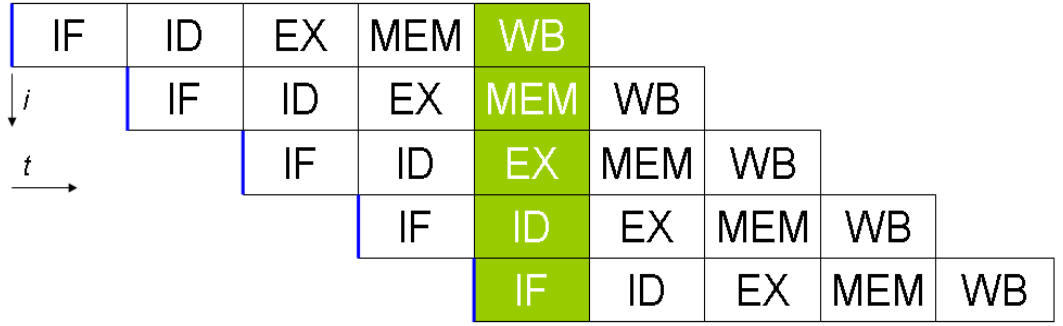
Transient execution attacks

- Transient execution attacks amplify the impact of existing side-channels
- The key observations are:
  - Processors execute instructions concurrently, speculatively and out-of-order
    - Completed instructions **commit** in program order
    - No architectural effects are visible until instruction is committed
    - **Transient** execution is any execution that never gets committed
  - Transiently executed instructions *also impact the micro-architectural state*
    - Hence: can **send** on a micro-architectural side-channel
  - Transient execution is less secure
    - An attacker *can influence what instructions a victim executes transiently*
      - The attacker now **controls the sending side of the side-channel too**!
    - Transient execution is *less confined by security checks*
      - The attacker can use transient execution to **access data that is inaccessible architecturally**

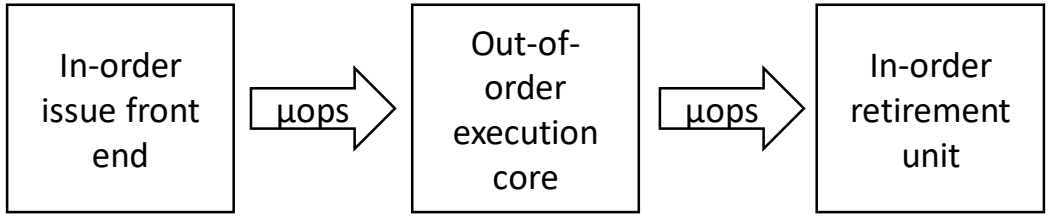
Transient execution

- All major processors support instruction-level parallelism
  - Processor implementations are pipelined
  - To keep the hardware busy, instructions are executed *out-of-order* and *speculatively*
- Transient execution = execution that *never gets committed*
  - No visible *architectural* effects
  - But there are persistent *micro-architectural* effects

A simple 5-stage pipeline:



Out-of-order execution:



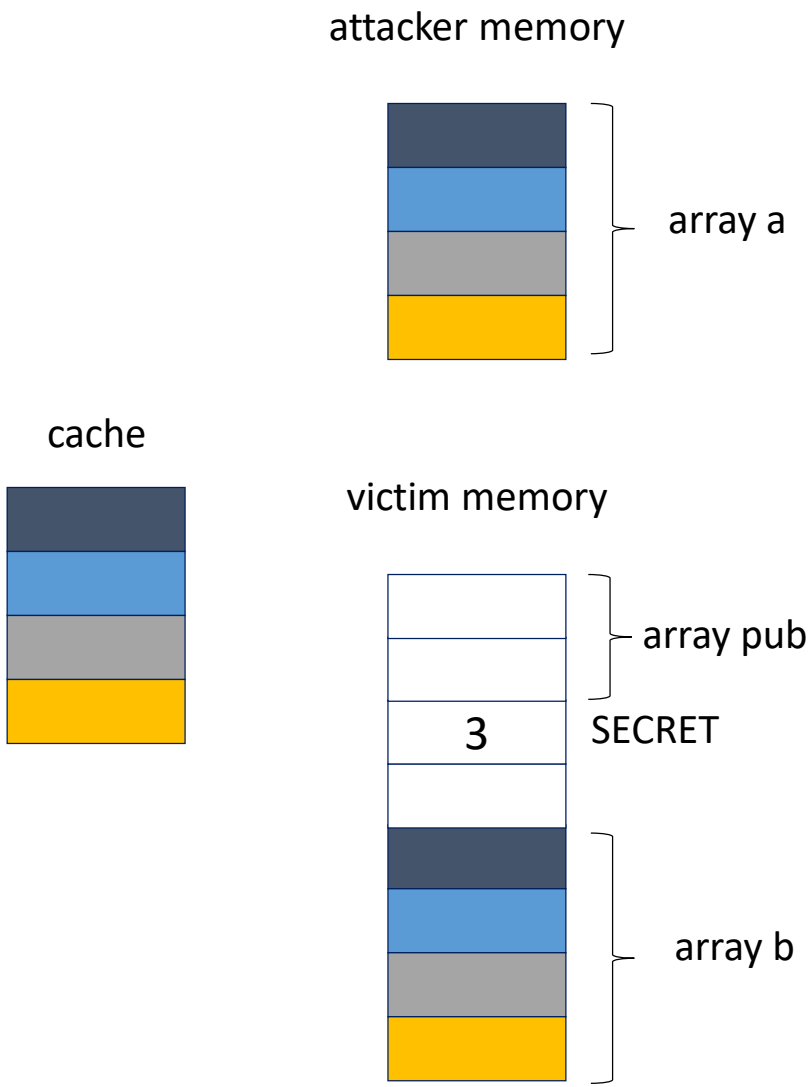
A simple example of a transient execution attack

attacker code

```
// train the branch predictor
process(0); process(0); …
// prime the cache
for (j=0; j<4; j++) z = a[j];
// attack!
process(size);
// measure access time to a[j] for all j
// slowest j is the SECRET
```

victim code

```
void process(int i) {
  int y;
  if (i < size) y = b[pub[i]];
}
```



A simple example of a transient execution attack

attacker code

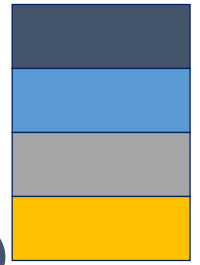
```
// train the branch predictor
process(0); process(0); …
// prime the cache
for (j=0; j<4; j++) z = a[j];
// attack!
process(size);
// measure access time to a[j] for all j
// slowest j is the SECRET
```

victim code

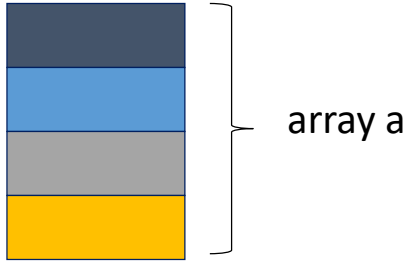
```
void process(int i) {
  int y;
  if (i < size) y = b[pub[i]];
}
```

Branch predictor learns that usually then branch is taken

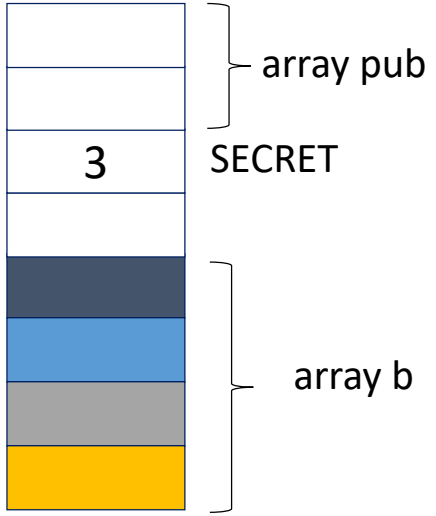
cache



attacker memory



victim memory



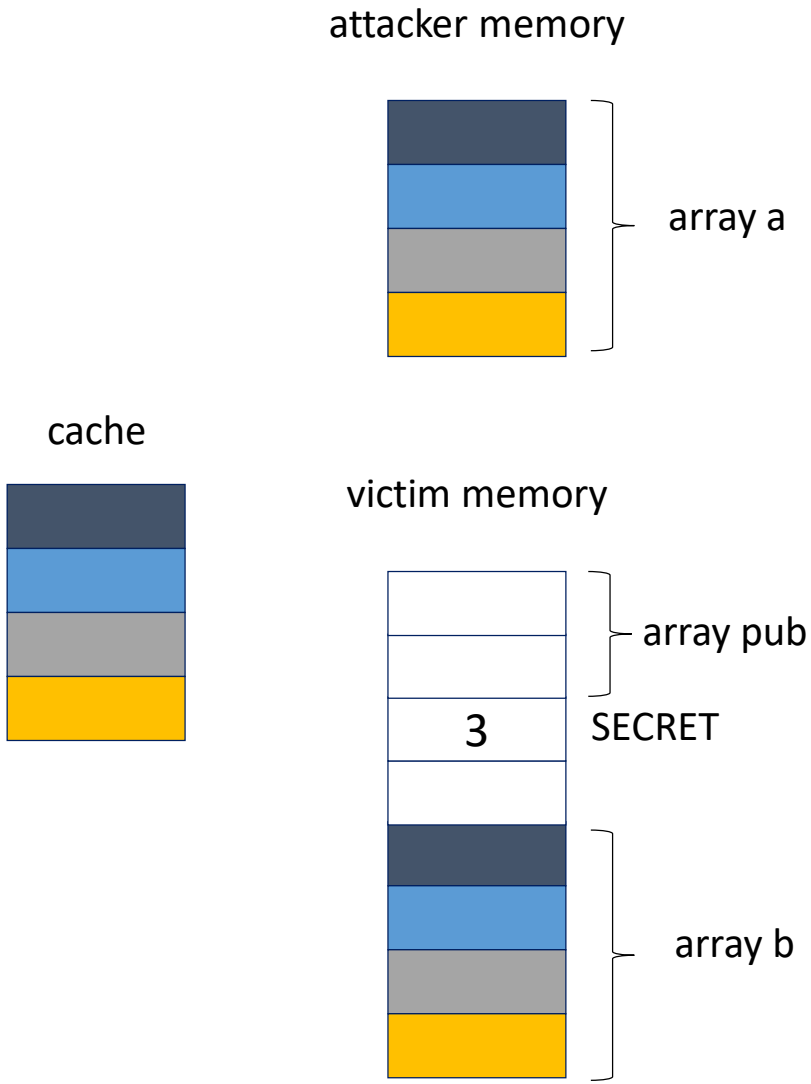
A simple example of a transient execution attack

attacker code

```
// train the branch predictor
process(0); process(0); …
// prime the cache
for (j=0; j<4; j++) z = a[j];
// attack!
process(size);
// measure access time to a[j] for all j
// slowest j is the SECRET
```

victim code

```
void process(int i) {
  int y;
  if (i < size) y = b[pub[i]];
}
```



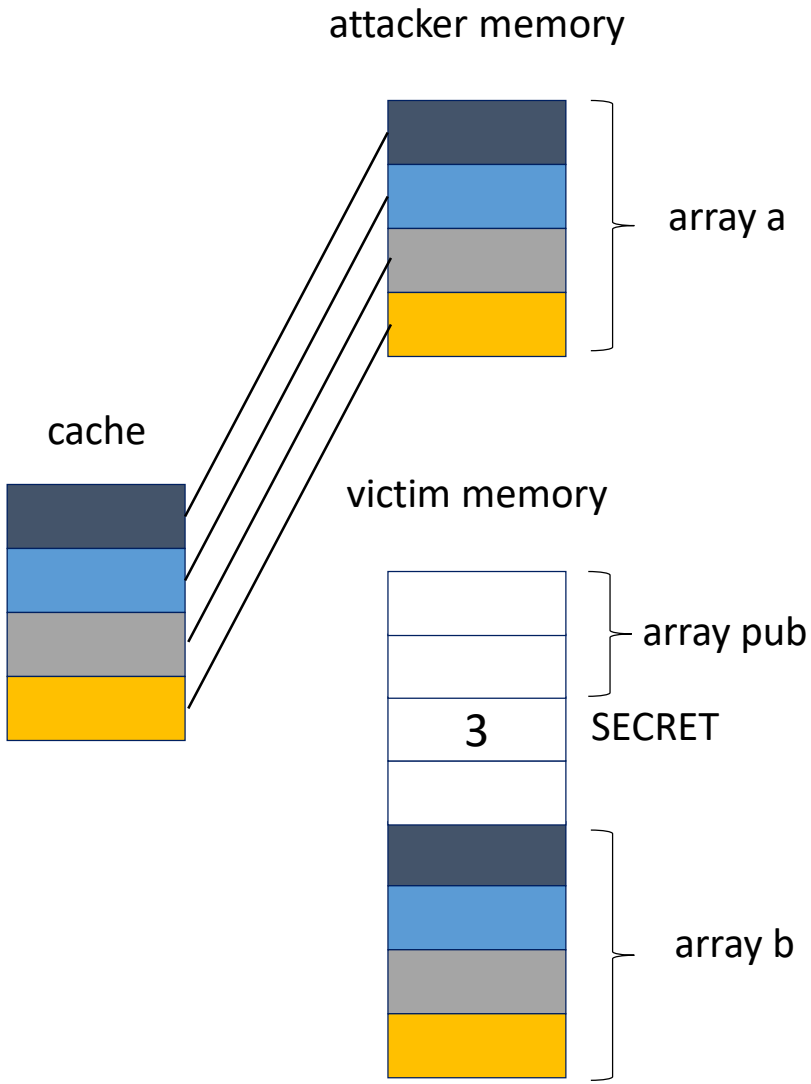
A simple example of a transient execution attack

attacker code

```
// train the branch predictor
process(0); process(0); …
// prime the cache
for (j=0; j<4; j++) z = a[j];
// attack!
process(size);
// measure access time to a[j] for all j
// slowest j is the SECRET
```

victim code

```
void process(int i) {
  int y;
  if (i < size) y = b[pub[i]];
}
```



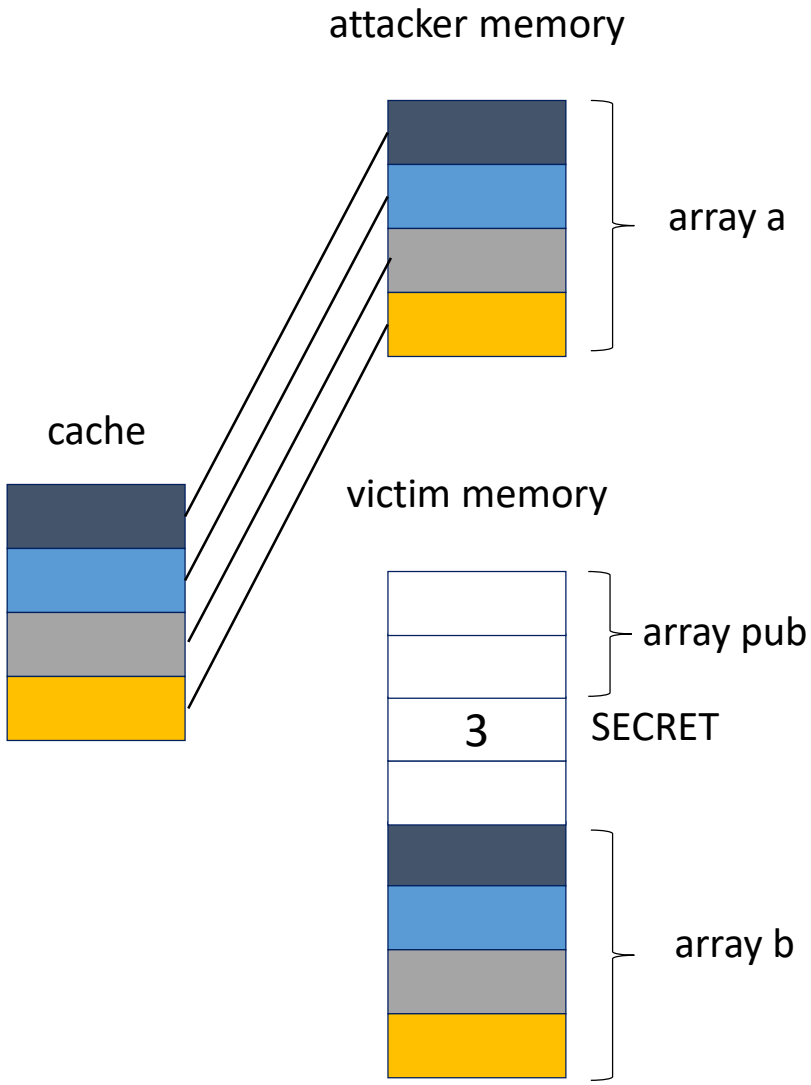
A simple example of a transient execution attack

attacker code

```
// train the branch predictor
process(0); process(0); …
// prime the cache
for (j=0; j<4; j++) z = a[j];
// attack!
process(size);
// measure access time to a[j] for all j
// slowest j is the SECRET
```

victim code

```
void process(int i) {
  int y;
  if (i < size) y = b[pub[i]];
}
```



A simple example of a transient execution attack

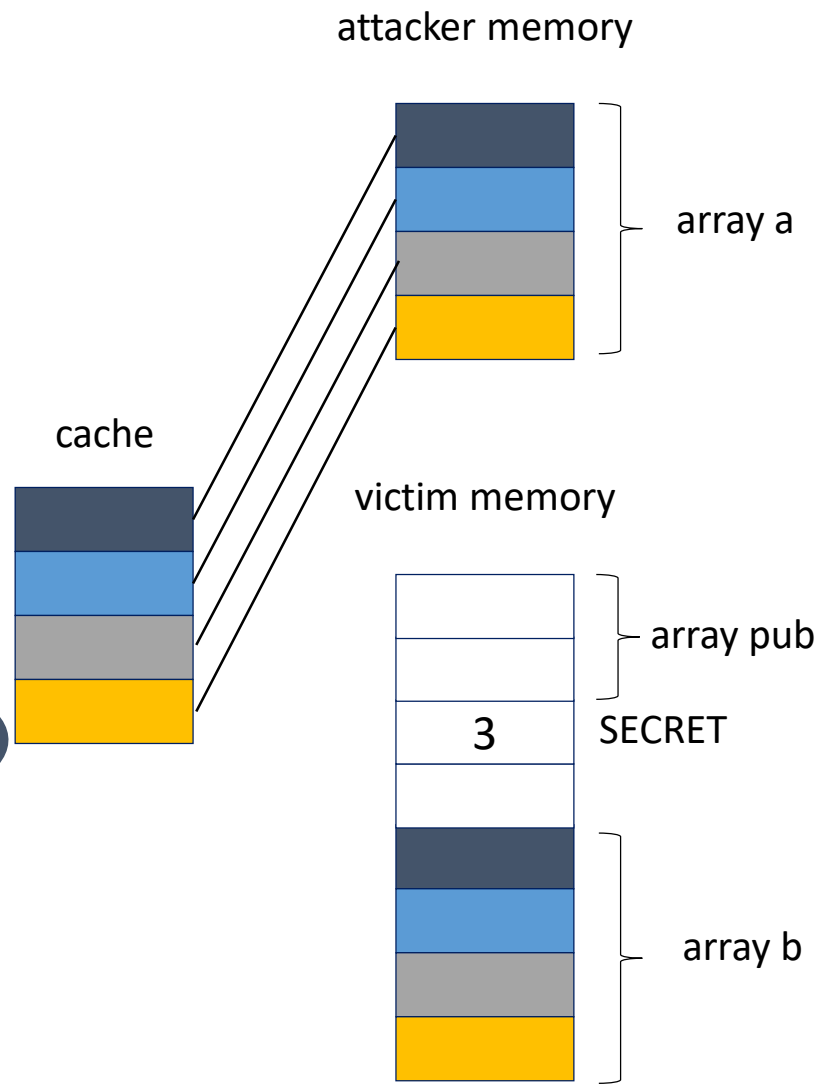
attacker code

```
// train the branch predictor
process(0); process(0); …
// prime the cache
for (j=0; j<4; j++) z = a[j];
// attack!
process(size);
// measure access time to a[j] for all j
// slowest j is the SECRET
```

victim code

```
void process(int i) {
  int y;
  if (i < size) y = b[pub[i]];
}
```

CPU speculatively executes the then branch



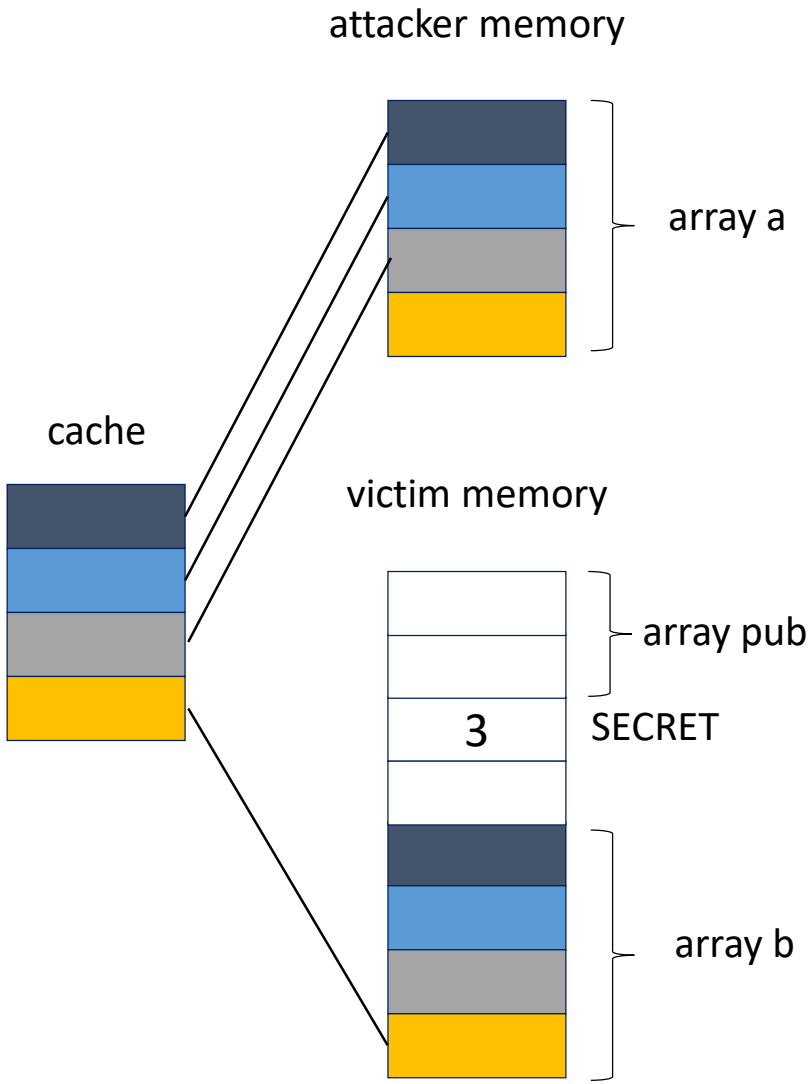
A simple example of a transient execution attack

attacker code

```
// train the branch predictor
process(0); process(0); …
// prime the cache
for (j=0; j<4; j++) z = a[j];
// attack!
process(size);
// measure access time to a[j] for all j
// slowest j is the SECRET
```

victim code

```
void process(int i) {
  int y;
  if (i < size) y = b[pub[i]];
}
```



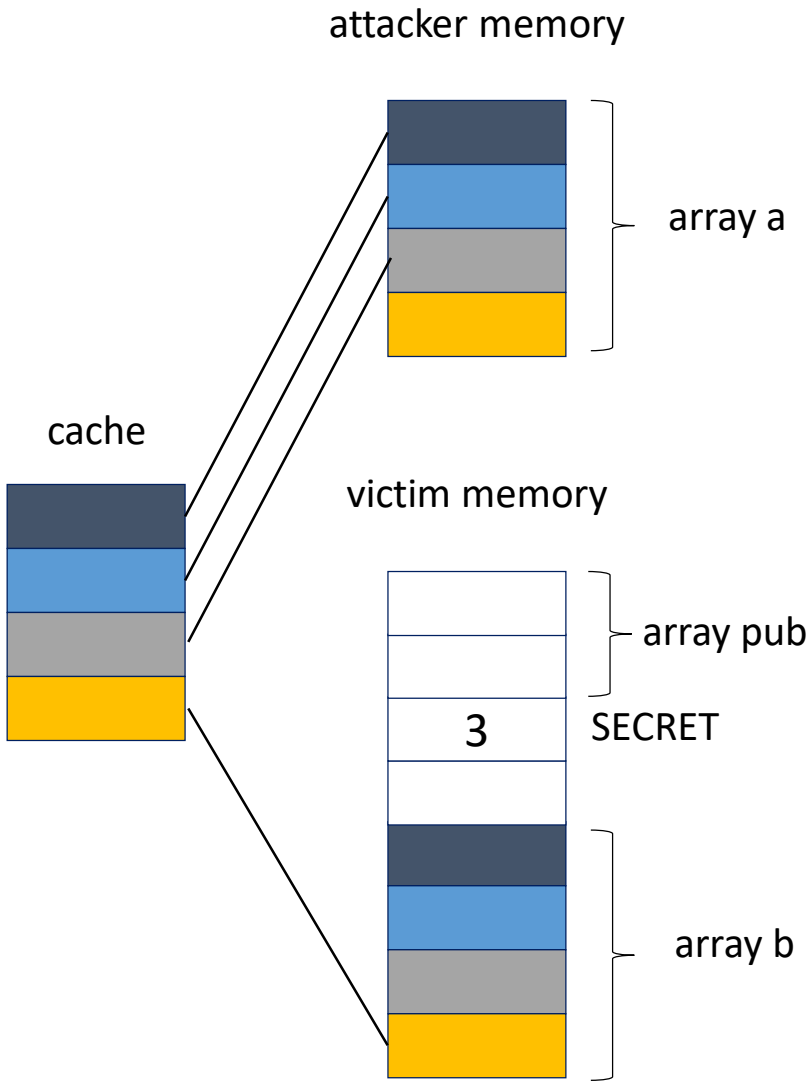
A simple example of a transient execution attack

attacker code

```
// train the branch predictor
process(0); process(0); …
// prime the cache
for (j=0; j<4; j++) z = a[j];
// attack!
process(size);
// measure access time to a[j] for all j
// slowest j is the SECRET
```

victim code

```
void process(int i) {
  int y;
  if (i < size) y = b[pub[i]];
}
```



Overview

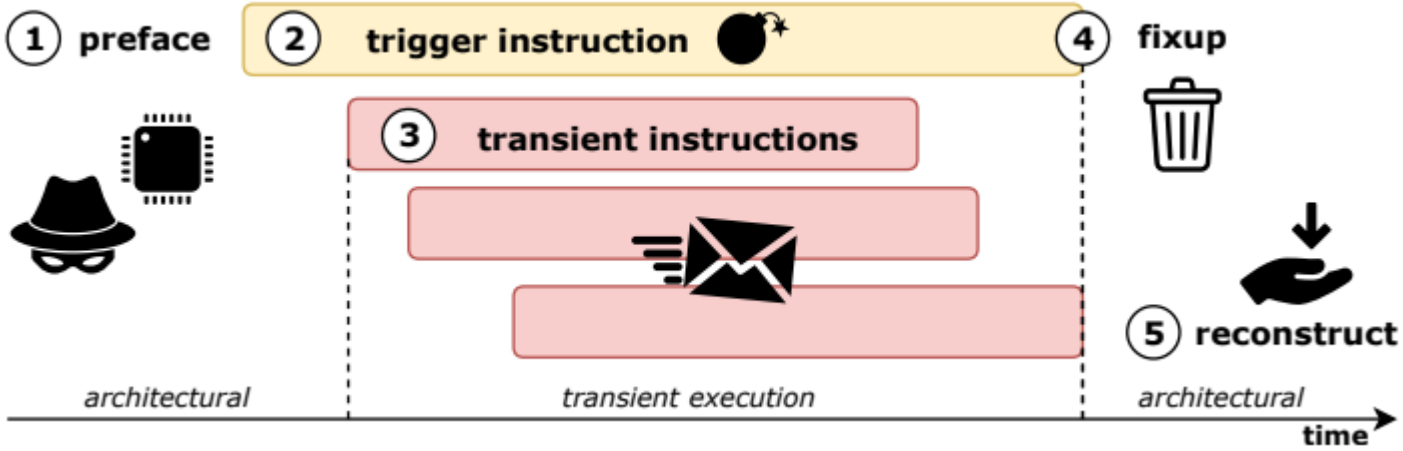
- Introduction: protection mechanisms
- Micro-architectural side-channel attacks by example
  - Side-channel attacks
  - Transient execution attacks
- A systematization of transient execution attacks
- Towards defenses
- Conclusions

Transient execution attacks

- The example we discussed is a simplified Spectre Variant 1 attack
  - Many other variants exist
- Note the **devastating** nature of this kind of attack
  - on any kind of software-enforced confidentiality
  - on any kind of hardware-enforced confidentiality where hardware resources are shared over protection boundaries
- It is important to build up a systematic understanding of these attacks and possible defenses
  - Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, Daniel Gruss, *A systematic evaluation of transient execution attacks and defenses*, Usenix Security 2019

General transient execution attack structure

1. Prime the micro-architectural state
2. Trigger transient execution (misprediction or fault)
3. Send on the covert channel
4. CPU flushes architectural effects of transient execution
5. Read from the covert channel



Source: Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, Daniel Gruss, *A systematic evaluation of transient execution attacks and defenses*, Usenix Security 2019

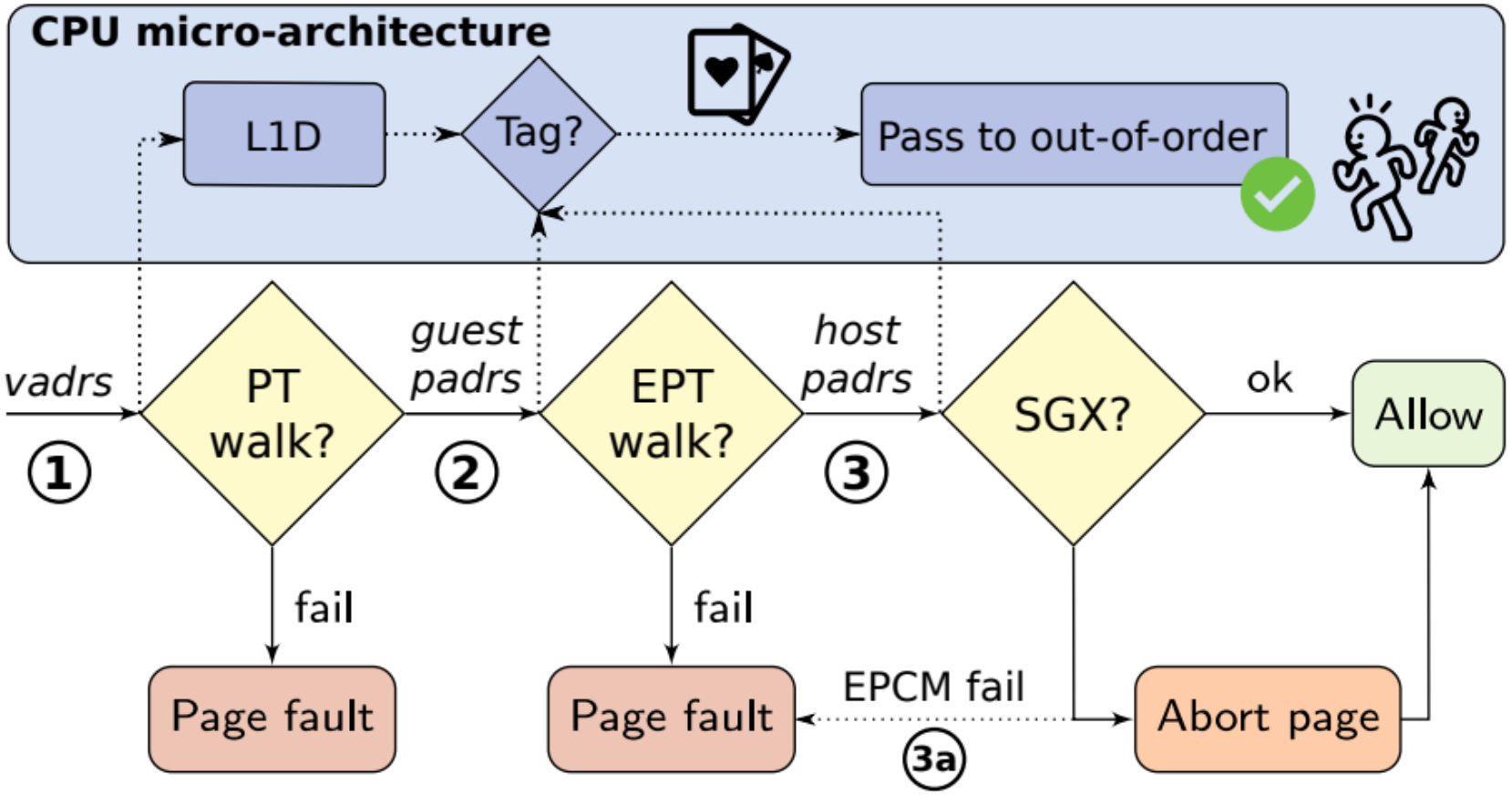
An example of a fault-based attack

- We know by now that values used in transient execution can be sent to the architectural level using a side-channel
- Hence, if we can make transient execution to work on values that are architecturally not accessible, we can exfiltrate these values
- A common way to do this is to execute a faulting load
  - Meltdown used this trick to read kernel memory from user space
  - Foreshadow / Foreshadow-NG use this to read from the L1 cache
  - The most recent wave of attacks use this to read from small buffers within the CPU (store buffer, line fill buffer)
- Let’s look in more detail at Foreshadow / Foreshadow-NG

What happens on a faulting load?

```
movb (vadrs), %al
```

<instructions to send out value in %al on a side-channel>

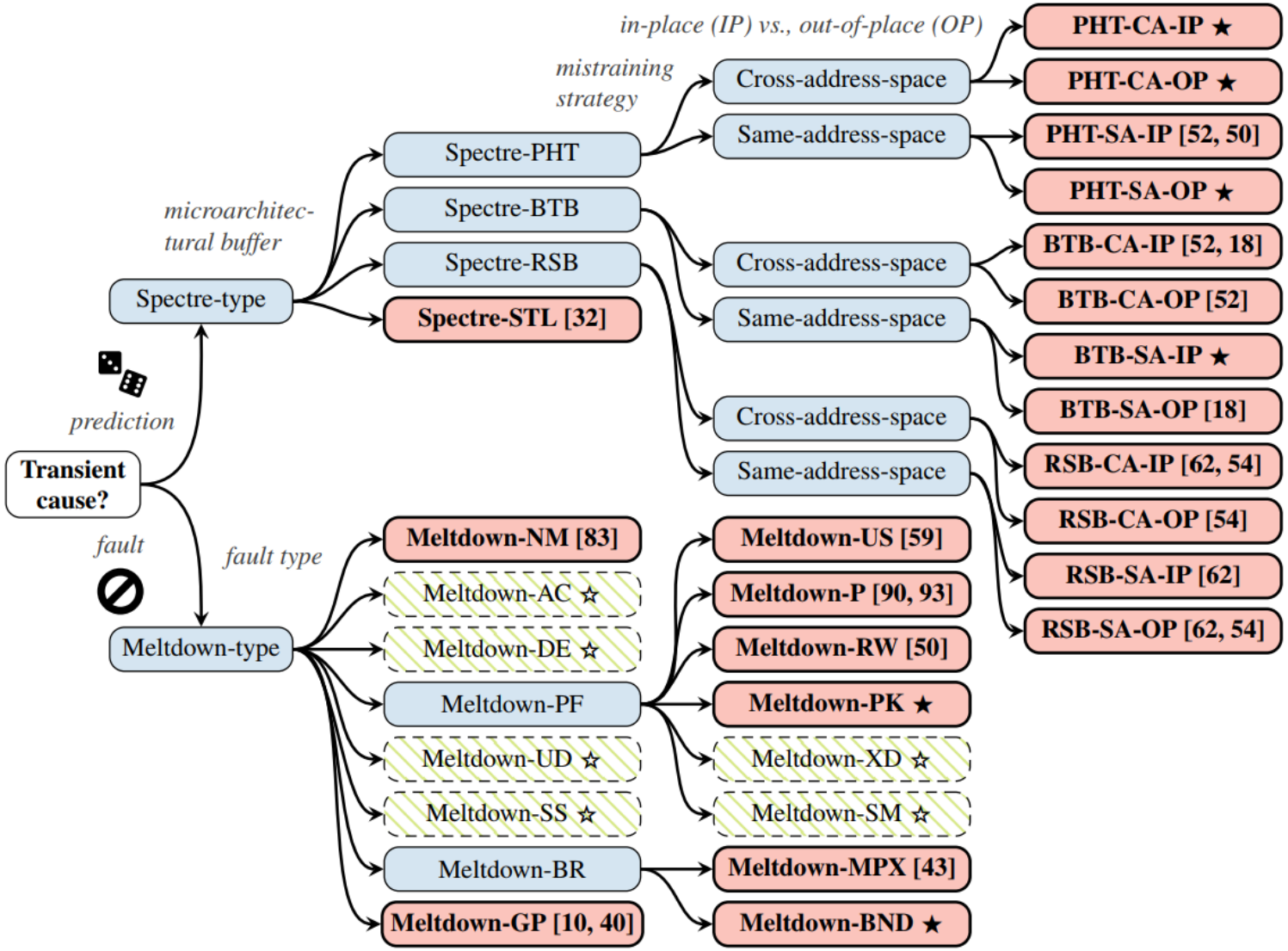


Sources:

Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, Raoul Strackx, *Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution*, Usenix Security 2018

Ofir Weisse, Jo Van Bulck, Marina Minkin, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Raoul Strackx, Thomas F Wenisch, Yuval Yarom, *Foreshadow-NG: Breaking the virtual memory abstraction with transient out-of-order execution,* Technical report

Classification of transient execution attacks



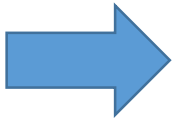
Source: Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, Daniel Gruss, *A systematic evaluation of transient execution attacks and defenses*, Usenix Security 2019

Overview

- Introduction: protection mechanisms
- Micro-architectural side-channel attacks by example
  - Side-channel attacks
  - Transient execution attacks
- A systematization of transient execution attacks
- Towards defenses
- Conclusions

Spectre defenses: simple example

```
void process(int i) {
  int y;
  if (i < size) y = b[pub[i]];
}
```



```
void process(int i) {
  int y;
  if (i < size) y = b[pub[i % size ]];
}
```

Spectre defenses

Attack		InvisiSpec [94]	SafeSpec [47]	DAWG [49]	RSB Stuffing [42]	Retpoline [88]	Poison Value [74]	Index Masking [74]	Site Isolation [86]	SLH [16, 22]	YSNB [68]	IBRS [3, 43]	STIPB [3, 43]	IBPB [3, 43]	Serialization [4, 40]	Taint Tracking [52]	Timer Reduction [52]	Sloth [50]	SSBD/SSBB [2, 43, 6]
Intel	Spectre-PHT	□	□	□	◇	◇	●	◐	◐	●	○	◇	◇	◇	◐	■	◐	◧	◇
	Spectre-BTB	□	□	□	◇	●	◇	◇	◐	◇	◇	●	◐	◐	◇	■	◐	◇	◇
	Spectre-RSB	□	□	□	◐	◇	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	◇	◇
	Spectre-STL	□	□	□	◇	◇	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	■	●
ARM	Spectre-PHT	□	□	□	◇	◇	●	◐	◐	●	○	◇	◇	◇	◐	■	◐	◧	◇
	Spectre-BTB	□	□	□	◇	●	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	◇	◇
	Spectre-RSB	□	□	□	◐	◇	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	◇	◇
	Spectre-STL	□	□	□	◇	◇	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	■	●
AMD	Spectre-PHT	□	□	□	◇	◇	●	◐	◐	●	○	◇	◇	◇	◐	■	◐	◧	◇
	Spectre-BTB	□	□	□	◇	●	◇	◇	◐	◇	◇	■	◧	◧	◇	■	◐	◇	◇
	Spectre-RSB	□	□	□	◐	◇	◇	◇	◐	◇	◇	◇	◇	◧	◇	■	◐	◇	◇
	Spectre-STL	□	□	□	◇	◇	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	■	●

Symbols show if an attack is mitigated (●), partially mitigated (◐), not mitigated (○), theoretically mitigated (■), theoretically impeded (◧), not theoretically impeded (□), or out of scope (◇).

Source: Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, Daniel Gruss, *A systematic evaluation of transient execution attacks and defenses*, Usenix Security 2019

Accepting defeat?

- It is not the first time that performance optimizations radically impact the hardware/software interface contract
  - E.g. Weak memory consistency models replaced the simple notion of sequentially consistent interleaving for multi-threaded code
  - The performance cost of the simple contract is just too high
- Will something similar happen for transient execution attacks?
  - The ISA specification could indicate what leaks can happen, and it will be up to software to deal with this

Overview

- System model
- Micro-architectural side-channel attacks by example
  - Side-channel attacks
  - Exploiting speculative execution
- A systematization of transient execution attacks
- Towards defenses
- Conclusions

Conclusions

- Micro-architectural attacks, and in particular transient execution attacks are a fundamentally new class of attacks:
  - That break all major security mechanisms
  - That are not easy to defend against
- Short-term defenses include:
  - Hardware patches, e.g. L1TF microcode updates
  - OS patches, e.g. KPTI
  - Compiler patches, e.g. speculative load hardening
- Long-term defenses are the subject of current research
  - But fundamental new ideas seem to be required