HACK IN BO®
Winter 2018 Edition

Take a Walk on the Wild Side(-Channel)

Enrico Perla

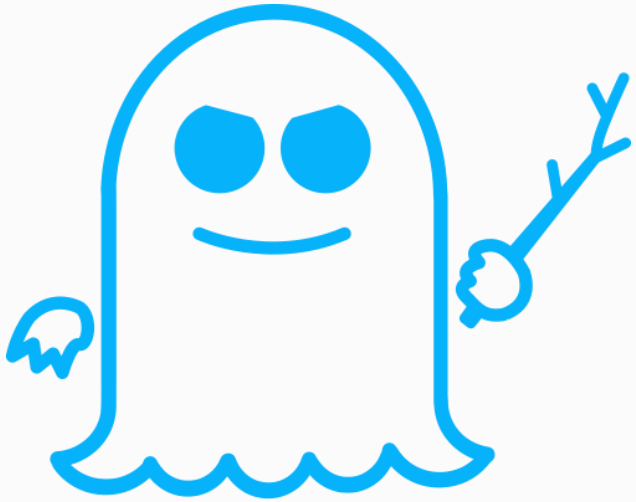
DISCLAIMER

This presentation is my own work and does not necessarily reflect the views of my previous or current employer.

This presentation lives on the shoulder of giants: Anders Fogh, Matt Miller and Christopher Ertl and all the other real deal researchers (Jann Horn, Daniel Gruss and the rest of the Graz University team and many others I can’t cite for space reasons).

SO, WHAT HAPPENED

- Spectre and Meltdown hit the news



SPECTRE

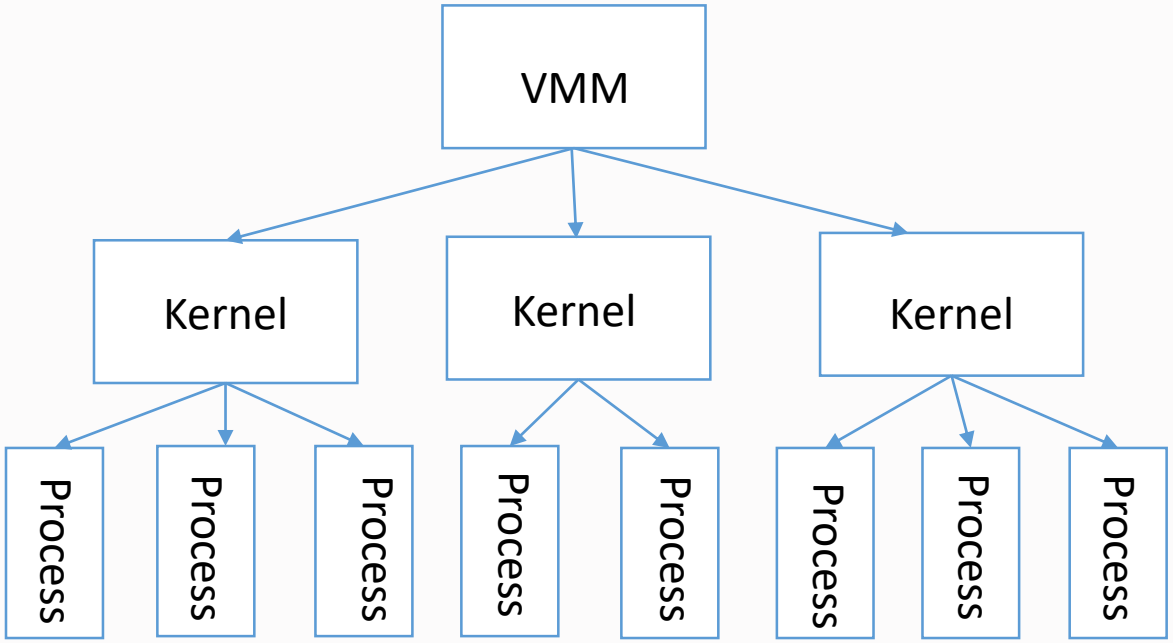
MELTDOWN

WHAT ARE WE DEALING WITH

- A new class of hardware vulnerabilities
- Information potentially leaking across privilege/isolation boundaries
  - A lower privilege entity may steal information from higher privilege entities
- Principles affect many modern CPUs
- Patching is not always straightforward

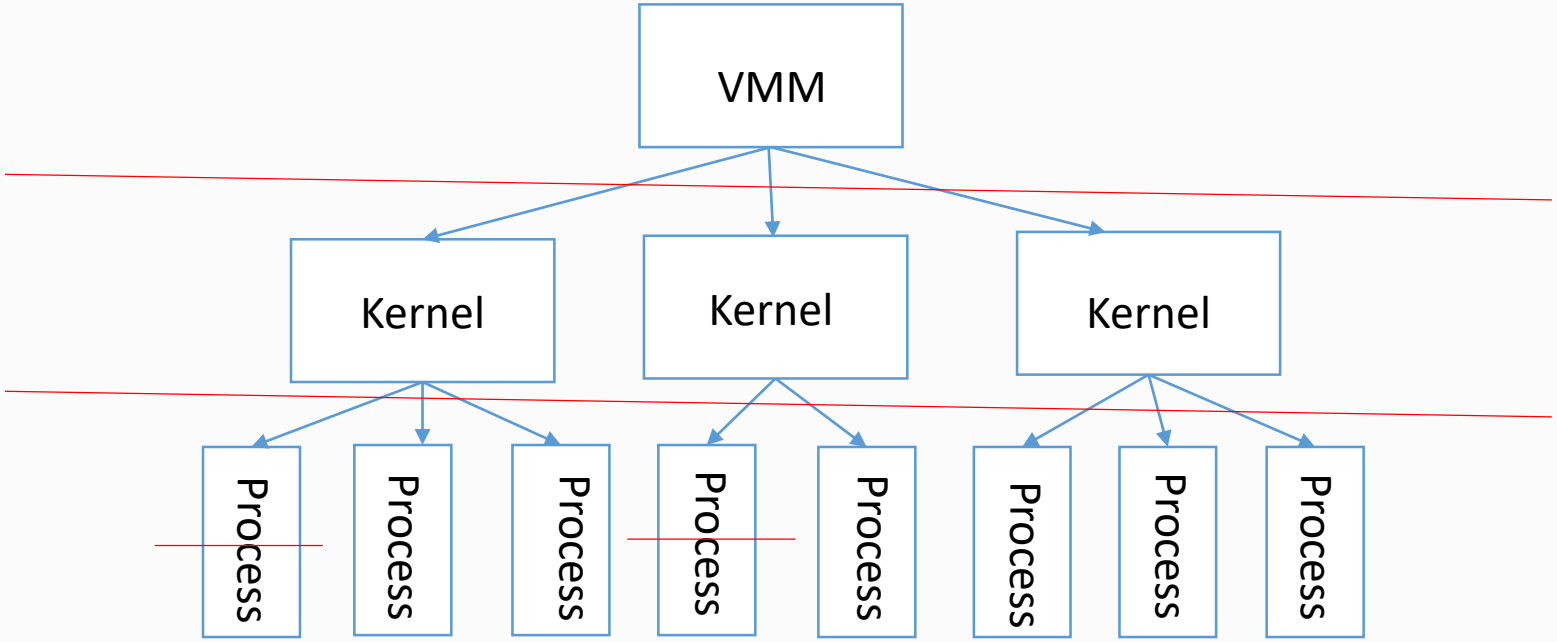
ISOLATION

- Hardware is not infinite, resources need to be shared
- Sharing and orchestration done by a higher privileged entity to avoid interferences



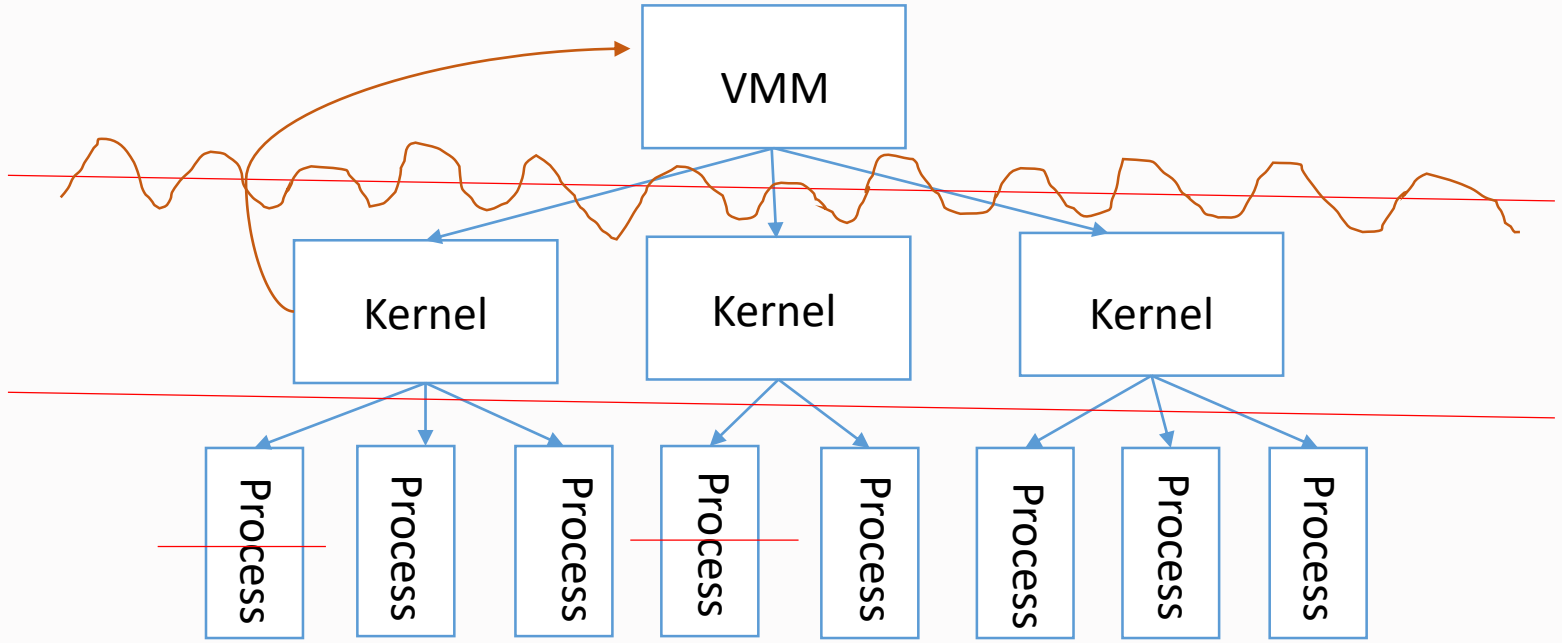
ISOLATION

- Hardware and software build on assumptions and define interfaces across privilege levels
- Data not exposed by these interfaces is not accessible by lower privilege levels



HOW TO BREAK ISOLATION

- Challenge and Bypass interface checks/restrictions and assumptions
- Issues hide in complexity, performance optimizations, usability shortcuts and legacy/retro compatibility



PRIVILEGE ESCALATION

- End tail of golden era of memory corruption attacks
- More and more attempts at formalizing the behaviour and designing better defences
  - Thomas Dullien – *Weird Machines, Exploitability and Provable Unexploitability*
    https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8226852&tag=1
- Attackers move down the stack as easier paths get closed
  - Improvements in userland defences -> kernel exploitation
- Increasing interest into challenging hardware assumptions

PROGRAM EXECUTION

A program is a set of instructions with a well defined/expected flow

MODEL: instructions execute sequentially, one after the other

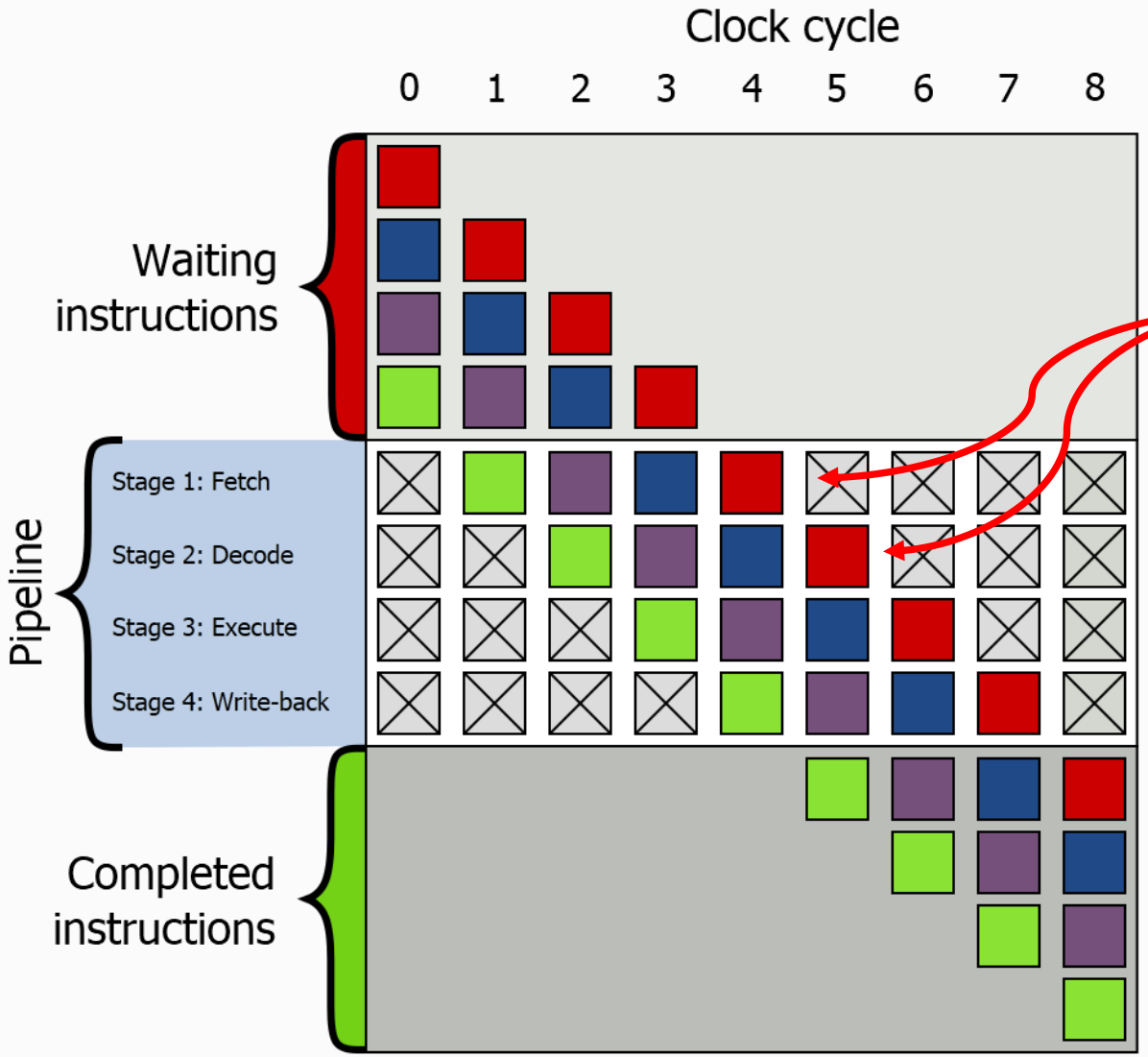
PROGRAM EXECUTION

A program is a set of instructions with a well defined/expected flow

MODEL: instructions execute sequentially, one after the other

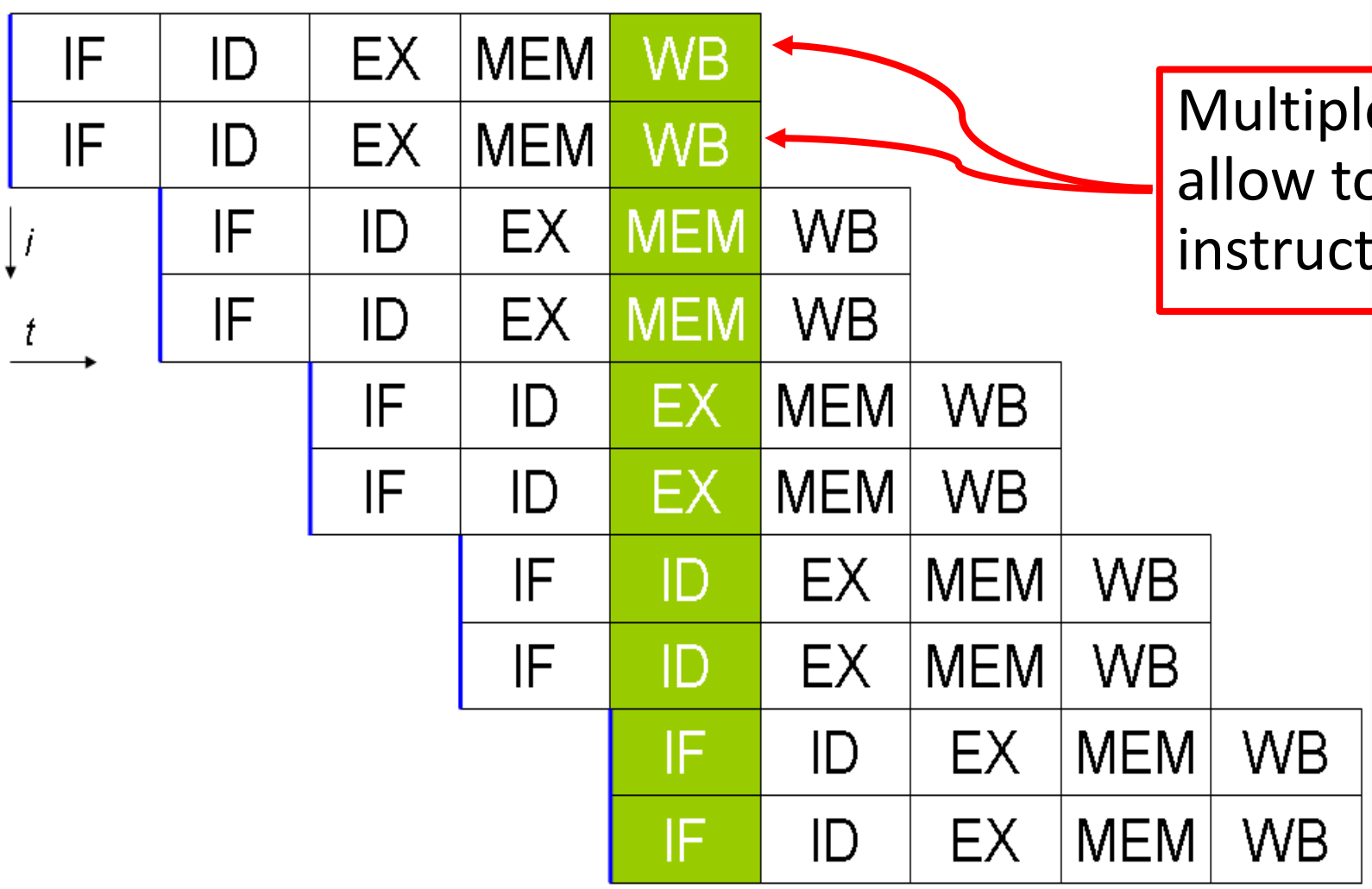
REALITY: this model of execution would be too slow. Modern CPUs use parallelism and speculation to improve performance

PIPELINE



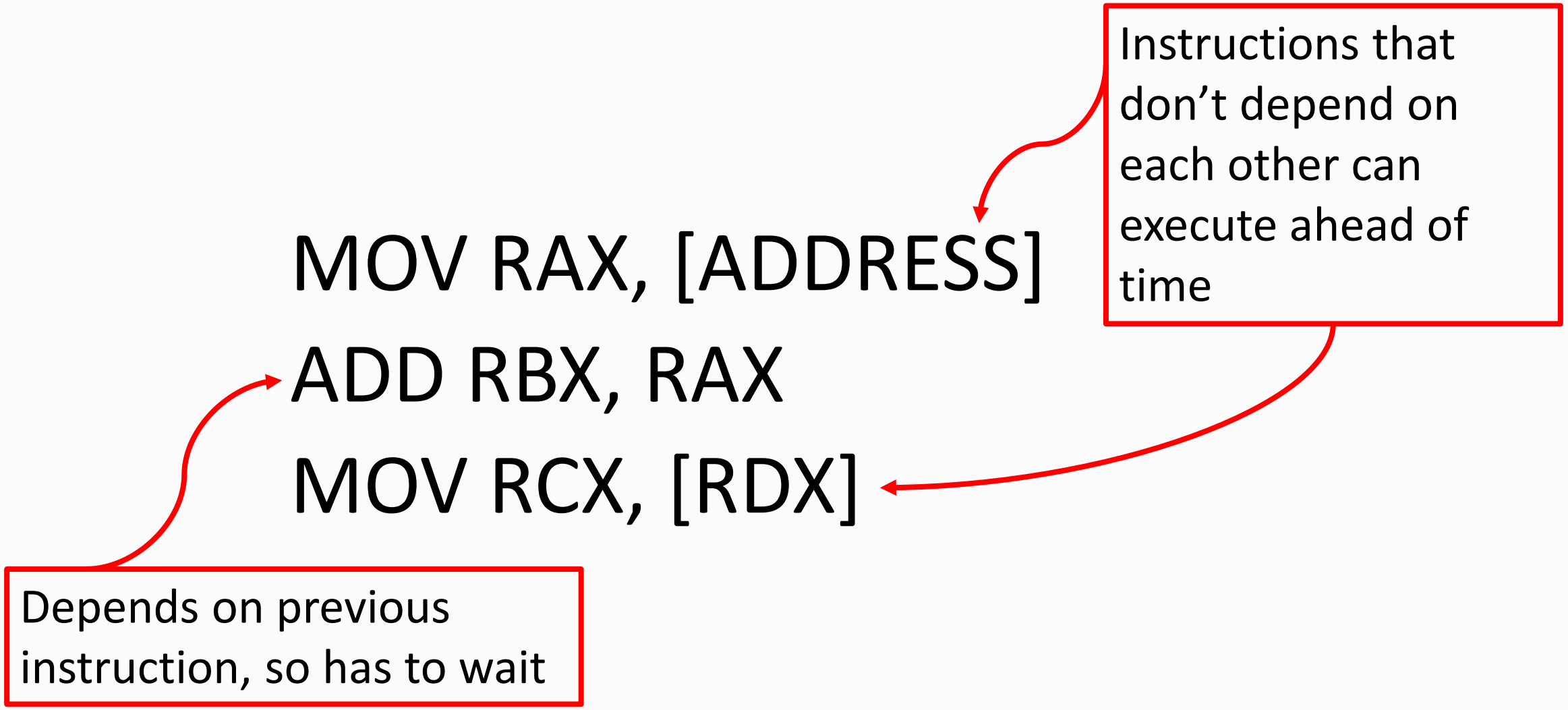
Instructions are broken down into smaller steps that are executed independently

SUPERSCALAR

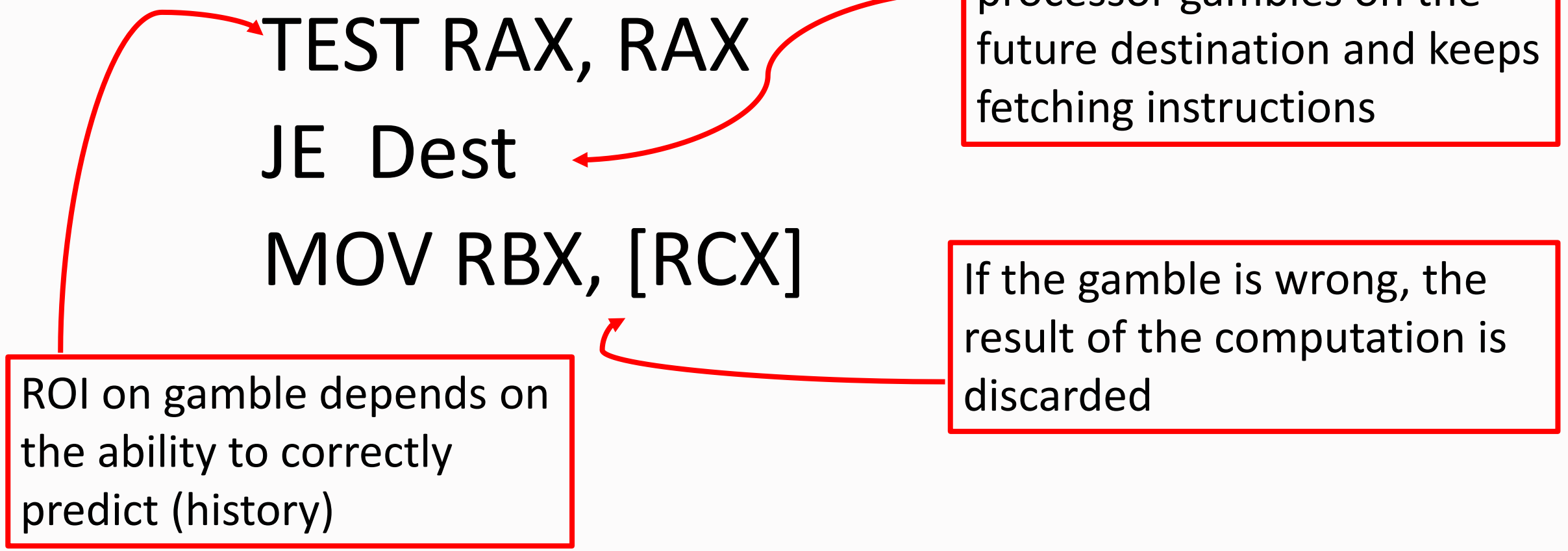


Multiple execution units allow to execute > 1 instruction per cycle

OUT OF ORDER EXECUTION



SPECULATIVE EXECUTION



SPECULATIVE EXECUTION

MODEL: whenever the processor guesses wrong, the discarded results do not leave visible traces and execution proceeds through the right path.

SPECULATIVE EXECUTION

MODEL: whenever the processor guesses wrong, the discarded results do not leave visible traces and execution proceeds through the right path.

REALITY: the thrown away execution leaves side effects. These side effects can be measured to extract information.

SPECULATIVE EXECUTION

MODEL: whenever the processor guesses wrong, the discarded results do not leave visible traces and execution proceeds through the right path.

REALITY: the thrown away execution leaves side effects. These side effects can be measured to extract information.

ATTACK: a lower privileged entity may extract this information to leak data from a more privileged entity.

SIDE EFFECTS

- Access to main memory is slow
- Programs tend to access the same (or adjacent) memory locations multiple times
- CPU have a set of caches where recently accessed memory is stored
- Cache traffic is not discarded after a mispredicted speculation path
- Caches are shared across different privilege levels
- Different time of access leaks information on whether a given memory line is in cache or not

SIDE EFFECTS

- Access to main memory is slow
- Programs tend to access the same (or adjacent) memory locations multiple times
- CPU have a set of caches where recently accessed memory is stored
- Cache traffic is not discarded after a mispredicted speculation path
- Caches are shared across different privilege levels
- Different time of access leaks information on whether a given memory line is in cache or not

SIDE-CHANNEL ATTACKS

HACK IN BO® Winter 2018 Edition

MODEL: 1.A lower privileged entity cannot reliably control speculation paths.
2.Extractable information is not valuable enough (address vs content).

SIDE-CHANNEL ATTACKS

HACK IN BO® Winter 2018 Edition

MODEL: 1.A lower privileged entity cannot reliably control speculation paths. 2.Extractable information is not valuable enough (address vs content).

REALITY: the prediction algorithm can be trained. Speculation choices become predictable. Certain code patterns leak content information.

SIDE-CHANNEL ATTACKS

MODEL: 1.A lower privileged entity cannot reliably control speculation paths. 2.Extractable information is not valuable enough (address vs content).

REALITY: the prediction algorithm can be trained. Speculation choices become predictable. Certain code patterns leak content information.

ATTACK: *Spectre V1*: an attacker may force a mispredicted branch with controlled input, leading to a speculative out-of-bounds load whose content is used as input for a subsequent load. The second load leaks the first load content. Attackers can find these sequences in higher privileged code or, in certain circumstances, create them (JIT).

SPECTRE V1

Speculate with index bigger than max_index

```
if (controlled_index < max_index) {
        value1 = index_array[controlled_index];
        value2 = data_array[value1 * 0x40];
}
```

Second memory dereference populates cache line that leaks value1 when data_array address is known

SIDE CHANNEL ATTACKS

HACK IN BO® Winter 2018 Edition

MODEL: lower privileged entities cannot influence the destination of predicted speculation.

SIDE CHANNEL ATTACKS

MODEL: lower privileged entities cannot influence the destination of predicted speculation.

REALITY: space matters. Prediction tables don't contain the whole source address and therefore aliasing from lower privileged entities may be possible.

SIDE CHANNEL ATTACKS

MODEL: lower privileged entities cannot influence the destination of predicted speculation.

REALITY: space matters. Prediction tables don't contain the whole source address and therefore aliasing from lower privileged entities may be possible.

ATTACK: *Spectre V2*: speculative ROP. Indirect branches can potentially be made to mispredict the target and jump to interesting gadgets.

SPECTRE V2

Attacker trains the indirect branch to point to some different location.

```
(*function_ptr)(par1, …);
```

New target contains a code sequence similar to V1

Number of attackable places increases significantly. Attacker may also control parameters.

SIDE CHANNEL ATTACKS

HACK IN BO® Winter 2018 Edition

MODEL: speculation stops on a privilege boundary (violation).

SIDE CHANNEL ATTACKS

MODEL: speculation stops on a privilege boundary (violation).

REALITY: exceptions are deferred to instruction retirement, so speculative paths may access data that would be otherwise not accessible.

SIDE CHANNEL ATTACKS

MODEL: speculation stops on a privilege boundary (violation).

REALITY: exceptions are deferred to instruction retirement, so speculative paths may access data that would be otherwise not accessible.

ATTACK: *Meltdown*: attacker can construct code that would normally trap in order to access memory beyond an exception boundary. This allows to leak data from kernel to user space. Exfiltration is done through similar constructs as V1.

Meltdown

Access to kernel_address traps.
Stash into a speculation path or a transaction for repeated use.

```
value1 = *kernel_address;
value2 = userland_array[value1 * 0x40];
```

CONDITIONS FOR A SUCCESSFUL ATTACK

- Have the CPU enter a speculation path
- Have the CPU stay in the speculation path long enough
- Have the speculation path leave side effects
- Do not interfere with the side effects
- Have a way to measure the side effects

THE SINGLE BEST SLIDE I'VE EVER SEEN

HACK IN BO®

1. Speculation primitive	Example

Conditional branch misprediction	`if (n < *p) {`
	`  // can speculate when n >= *p`
	`}`

Indirect branch misprediction	`// can speculate wrong branch target` `(*FuncPtr)();`

Exception delivery	`// may do permission check at`
	`// retirement`
	`value = *p;`

2. Windowing gadget	Example
Non-cached load	`// *p not present in cache` `value = *p;`
Dependency chain of loads	`value = ********p;`
Dependency chain of ALU operations	`value += 10;` `value += 10;` `value += 10;`
…	

3. Disclosure gadget	Example
One level of memory indirection, out-of-bounds	`if (x < y)` `  return buf[x];`

Two levels of memory indirection, out-of-bounds	`if (x < y) {`
	`  n = buf[x];`
	`  return buf2[n];`
	`}`

Three levels of memory indirection, out-of-bounds	`if (x < y) {`
	`  char *p = buf[n];`
	`  char b = *p;`
	`  return buf2[b];`
	`}`

X 7.65 inch

4. Disclosure primitive	Example
FLUSH+RELOAD	Priming phase: flush candidate cache lines Trigger phase: cache line is loaded based off secret Observing phase: load candidate cache lines, fastest access may be signal
EVICT+TIME	Priming phase: evict congruent cache line Trigger phase: cache line is loaded based off secret Observing phase: measure time of operation, slowest operation may be signal
PRIME+PROBE	Priming phase: load candidate cache lines Triggering phase: cache set is evicted based off secret Observing phase: load candidate cache lines, slowest access may be signal

DEFENCE

Increasing level of complexity, increasing level of effectiveness

FIXING THE INDIVIDUAL ISSUE

- A design issue, not strictly a bug
  - Naturally a class, see next slide ;-)
- Affects all major CPUs
  - Sharing and performance optimizations are fundamental points on the evolution scale of CPUs
- Reinforced take-aways:
  - sharing of resources should be done with side-channel attacks in mind
  - likely need more barriers at privilege boundaries

PREVENTING THE CLASS

- Eradicate/Reduce the Speculation Primitive
  - Do not have the CPU enter a dangerous speculation path
  - Code/compiler fixing the paths
  - Explicit serialization
  - LFENCE, MEMBAR, etc.
- Implicit serialization
  - CMOV
- Speculation safe branches
  - FAR JMP, retpoline, etc.
- Manage indirect branch prediction
  - IBRS, IBPB, STIBP, etc.
- Disable prediction
  - HW_BTI

KILLING THE EXPLOITATION TECHNIQUE

- Make privileged data less accessible
  - Similar concept to arbitrary read/write defences
  - Separated kernel/page tables (KPTI/KVA Shadow/etc.)
  - 1:1 physical mapping much less popular at parties
- Reduce disclosure precision
  - Works when the attacker has less direct access to the hardware primitives (e.g. browsers/Javascript)
- Reduce sharing of physical pages across guests

IS THE SKY FALLING?

- Seriously, no.
  - Still a read primitive, no corruption.
- Very interesting class, expect variations and evolutions in the next years
- Some contexts more sensitive than others
  - Cloud environments vs single user machines
- Likely to shape the way we think about hardware and software
  - Process already in motion on the isolation front (e.g. memory tagging)

QUESTIONS?

HACK IN BO®
Winter 2018 Edition