A Formal Model of Cache Speculation Side-Channels

Catalin Marinas
TLA$^+$ Community Event 2020

© 2020 Arm Limited (or its affiliates)

Introduction

Why a formal model of cache speculation side-channels:

- A (simpler) way to reason about the behaviour of a system and its security properties
- Allows exploring similar vulnerabilities at an abstract level in a unified way
  - Same specification can cover Spectre variants and Meltdown

Aim for the formal model:

- Does the CPU specification guarantee the security properties under speculative execution for *any* code sequence?

Not covered:

- Is a *specific* code sequence vulnerable under the CPU specification?

© 2020 Arm Limited (or its affiliates)

Spectre v1 Example

- Bypassing software checking of untrusted values (64-bit ARM assembly)

```
        LDR     X1, [X2]                // X2 is a pointer to kernel_array->length
        CMP     X0, X1                  // X0 holds untrusted_offset_from_user
        BGE     out_of_range
        LDRB    W4, [X5, X0]            // X5 holds kernel_array->data
        AND     X4, X4, #1              // arithmetic ops on potentially private data
        LSL     X4, X4, #8
        ADD     X4, X4, #0x200
        LDRB    X7, [X8, X4]            // X8 holds user_array->data, X4 secret-derived
out_of_range
```

© 2020 Arm Limited (or its affiliates)

Abstract Machine – State

- Constants (sets) and operation tables

$$
\begin{aligned}
LADDRS &\triangleq \{l1, l2, \dots\} \\
HADDRS &\triangleq \{h1, h2, \dots\} \\
DATA &\triangleq \{d1, d2, \dots\} \\
REGS &\triangleq \{r1, r2, \dots\}
\end{aligned}
$$

$$
\begin{aligned}
ADDRS &\triangleq LADDRS \cup HADDRS \\
VALUES &\triangleq ADDRS \cup DATA \cup \{\text{“}zero\text{”}\}
\end{aligned}
$$

$$OPTABLES \triangleq [VALUES \times VALUES \rightarrow VALUES]$$

- CPU security/privilege mode

$$\boldsymbol{mode} \in \{\text{“}low\text{”}, \text{“}high\text{”}\}$$

- CPU registers (array/function)

$$\boldsymbol{regs} \in [REGS \rightarrow VALUES]$$

Register	r1	r2	r3	r4	…
Value	d2	d1	l1	h1	…

- Memory (array/function)

$$
\begin{aligned}
\boldsymbol{mem} &\in [ADDRS \rightarrow VALUES] \\
\boldsymbol{cached} &\in [ADDRS \rightarrow BOOLEAN]
\end{aligned}
$$

Address	l1	l2	…	h1	h2	…
Value	d2	l1	…	d3	h1	…
Cached	T	F	…	F	T	…

© 2020 Arm Limited (or its affiliates)

Abstract Machine – Actions (state transitions)

- Sets of *valid* instructions (*tuples* of mnemonic and arguments)

$$
\begin{aligned}
Havoc &\triangleq \{\langle \text{"}\boldsymbol{SET}\text{"}, r, v\rangle : r \in REGS, v \in VALUES\} \\
Move &\triangleq \{\langle \text{"}\boldsymbol{MOV}\text{"}, rt, rm\rangle : rt, rm \in REGS\} \\
Load &\triangleq \{\langle \text{"}\boldsymbol{LDR}\text{"}, rt, rm\rangle : rt, rm \in REGS \land AccessOK(regs[rm])\} \\
Store &\triangleq \{\langle \text{"}\boldsymbol{STR}\text{"}, rt, rm\rangle : rt, rm \in REGS \land AccessOK(regs[rm])\} \\
Op &\triangleq \{\langle \text{"}\boldsymbol{OP}\text{"}, rt, rm, rn, op\rangle : rt, rm, rn \in REGS, op \in OPTABLES\} \\
Exception &\triangleq IF\ mode = \text{"low"}\ THEN\ \{\langle \text{"}\boldsymbol{HCALL}\text{"}\rangle\}\ ELSE\ \{\langle \text{"}\boldsymbol{LRET}\text{"}\rangle\}
\end{aligned}
$$

- The set of all possible *valid* instructions

$$Instructions \triangleq Havoc \cup Move \cup Load \cup Store \cup Op \cup Exception$$

- Instruction dispatch/execution: $\boldsymbol{Execute}(instr)$

© 2020 Arm Limited (or its affiliates)

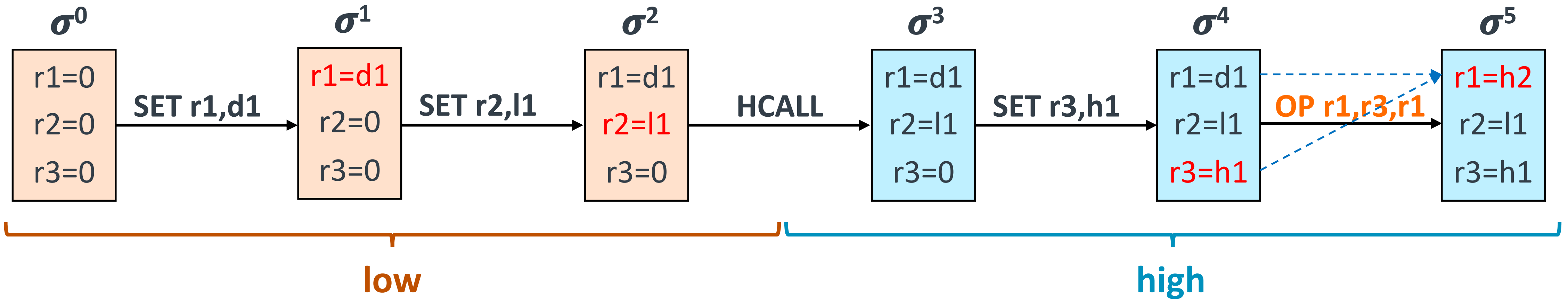
Program Execution as Succession of States

- Initial *SimpleCPU* state

$$\begin{aligned} \mathbf{Init} \triangleq\ & \land\ mode = \text{“}low\text{”} \\ & \land\ regs = [r \in REGS \mapsto \text{”}zero\text{”}] \\ & \land\ mem \in [ADDRS \rightarrow VALUES] \\ & \land\ cached = [a \in ADDRS \mapsto FALSE] \end{aligned}$$

- Next *SimpleCPU* step

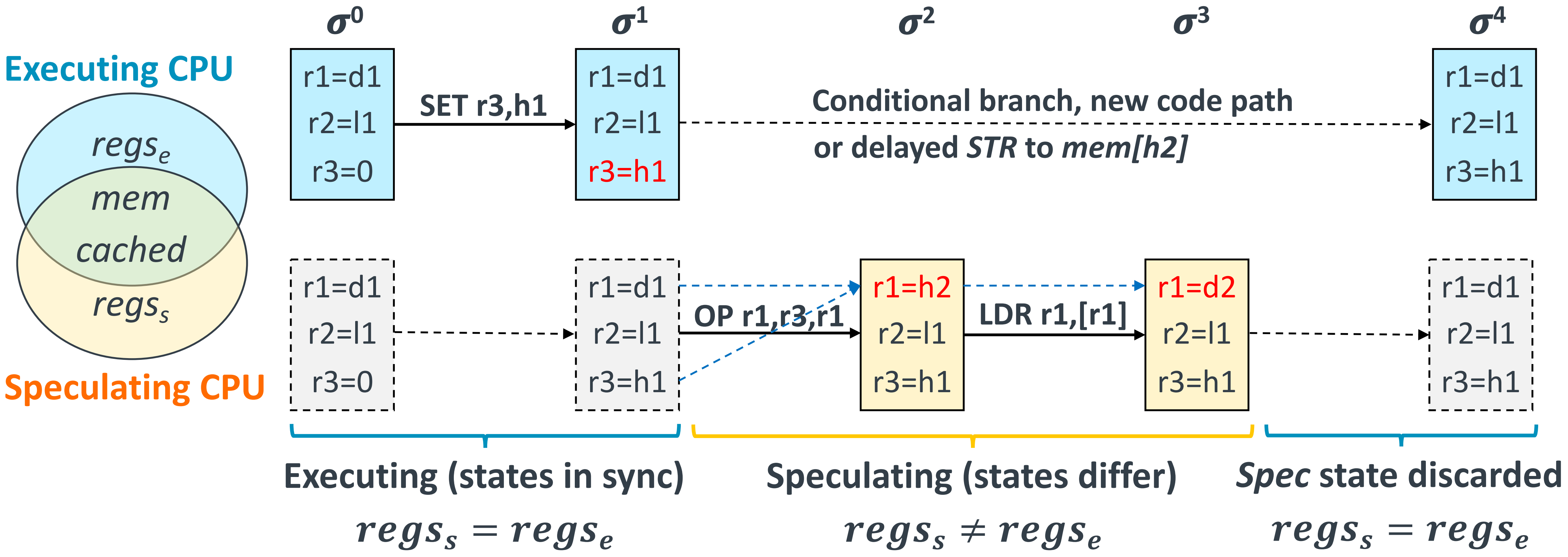
$$\mathbf{Next} \triangleq \exists instr \in Instructions : \mathbf{Execute}(instr)$$



© 2020 Arm Limited (or its affiliates)

Abstract Model of Speculative Execution

- Speculative execution modelled as a *"shadow" abstract machine* (CPU) sharing the ***mem*** and ***cached*** states with the *executing machine* but with a separate ***regs*** state
  - *Speculating machine* supports a **subset** of *Instructions*, e.g. $Load \cup Op$



© 2020 Arm Limited (or its affiliates)

TLA$^+$ Specification of Speculative Execution

- Speculating machine has its own registers

$$ExecCPU \triangleq \textbf{\textit{INSTANCE}}\ SimpleCPU$$
$$SpecCPU \triangleq \textbf{\textit{INSTANCE}}\ SimpleCPU\ \textbf{\textit{WITH}}\ regs \leftarrow specregs$$

- Speculating registers state discarded on (committed) instruction execution

$$Exec(instr) \triangleq ExecCPU!Execute(instr) \land specregs' = regs'$$
$$Spec(instr) \triangleq SpecCPU!Execute(instr) \land \textbf{\textit{UNCHANGED}}\ \langle regs, mem \rangle$$

- Only certain instructions are available under speculation

$$ExecInstr \triangleq Havoc \cup Move \cup Load \cup Store \cup Op \cup Exception$$
$$SpecInstr \triangleq Load \cup Op$$

- Either execution or speculation

$$\begin{aligned} \textbf{\textit{Next}} \triangleq\ & \lor \exists instr \in ExecInstr : Exec(instr) \\ & \lor \exists instr \in SpecInstr : Spec(instr) \end{aligned}$$

© 2020 Arm Limited (or its affiliates)

Roles, States and Observations

Victim: high security mode (e.g. OS kernel)

- High *mode* state consists of CPU registers and high security memory

$$HighState \triangleq \langle regs, [addr \in HADDRS \mapsto mem[addr]]\rangle$$

- High *mode* input and output consist of CPU registers and low security memory

$$Input \triangleq \langle regs, [addr \in LADDRS \mapsto mem[addr]]\rangle$$

$$Output \triangleq \langle regs, [addr \in LADDRS \mapsto mem[addr]]\rangle$$

© 2020 Arm Limited (or its affiliates)

Roles, States and Observations

Attacker: low security mode (e.g. user application)

- Low *mode* state consists of CPU registers and user memory:

$$LowState \triangleq \langle regs, [addr \in LADDRS \mapsto mem[addr]]\rangle$$

- Without *side-channels*, the attacker can only observe the values in low memory*:*

$$LowObs \triangleq [addr \in LADDRS \mapsto \langle mem[addr]\rangle]$$

- With *cache side-channels*, the attacker can additionally observe the *cached* state of a memory location (e.g. by measuring the access time)

$$LowObs \triangleq [addr \in LADDRS \mapsto \langle mem[addr], cached[addr]\rangle]$$

© 2020 Arm Limited (or its affiliates)

Security Properties – Confidentiality

- The *attacker*'s observations (*LowObs*) is a deterministic function of the initial *LowState*, the *victim*'s *Output* (same as *LowState*) and its own actions
  - The *attacker* cannot observe anything other than what the *victim* explicitly allows in its *output*
- Formal model: any two *LowState-identical* behaviours of a system *P*, with the same initial *LowObs*, have identical *LowObs* observations

$$
\begin{aligned}
&\forall\, \sigma_1, \sigma_2 \vDash P : \\
&\qquad (\boldsymbol{LowObs}(\sigma_1^0) = \boldsymbol{LowObs}(\sigma_2^0) \land \\
&\qquad\ \forall i \in Nat : LowState(\sigma_1^i) = LowState(\sigma_2^i)) \Rightarrow \\
&\quad \forall i \in Nat : \boldsymbol{LowObs}(\sigma_1^i) = \boldsymbol{LowObs}(\sigma_2^i)
\end{aligned}
$$

(not valid TLA$^+$ syntax)

© 2020 Arm Limited (or its affiliates)

Security Properties – Integrity

- The *victim*'s state is a deterministic function of the initial *HighState*, *victim*'s *Input* and its own actions
  - *Victim*'s execution (sequence of states) is not affected by the *attacker* beyond the *Input* provided
- Formal model: any two behaviours of a system *P*, with the same initial *HighState* and same *Input*, have identical *HighState* and *Output*

$$
\begin{aligned}
&\forall\ \sigma_1, \sigma_2 \vDash P : \\
&\quad (\boldsymbol{HighState}(\sigma_1^0) = \boldsymbol{HighState}(\sigma_2^0) \land \\
&\quad \forall i \in Nat : Input(\sigma_1^i) = Input(\sigma_2^i)) \Rightarrow \\
&\quad \forall i \in Nat : (\boldsymbol{HighState}(\sigma_1^i) = \boldsymbol{HighState}(\sigma_2^i) \land \\
&\qquad \boldsymbol{Output}(\sigma_1^i) = \boldsymbol{Output}(\sigma_2^i))
\end{aligned}
$$

(not valid TLA$^+$ syntax)

© 2020 Arm Limited (or its affiliates)

Confidentiality in TLA$^+$

- Hyperproperties not supported directly, creating a new specification for two behaviours

$$
\begin{aligned}
\mathbf{Init} \triangleq\ & \land\ CPU1!Init \land CPU2!Init \\
& \land\ CPU1!LowState = CPU2!LowState \\
& \land\ CPU1!LowObs = CPU2!LowObs
\end{aligned}
$$

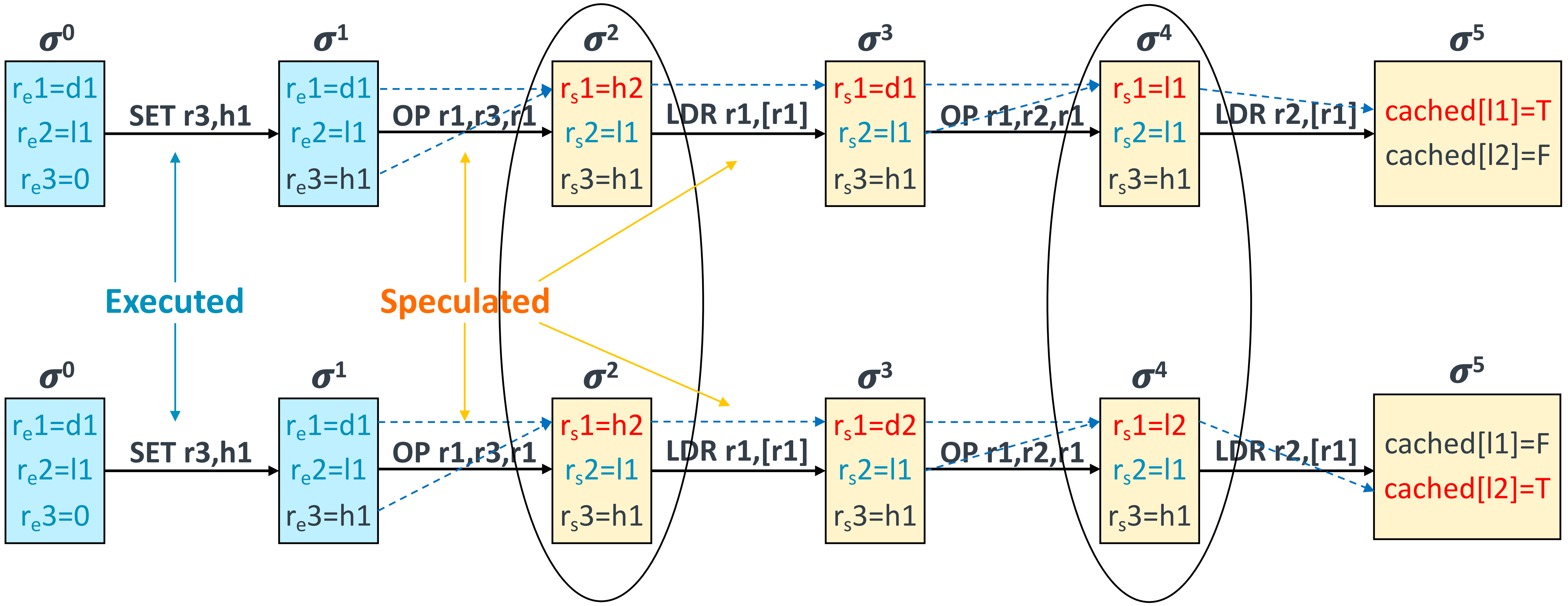
$$
\begin{aligned}
\mathbf{Next} \triangleq\ & \lor\ \exists instr \in ExecInstr : (CPU1!Exec(instr) \land CPU2!Exec(instr) \land \\
& \qquad\qquad regs1' = regs2') \\
& \lor\ \exists instr \in SpecInstr : CPU1!Spec(instr) \land CPU2!Spec(instr)
\end{aligned}
$$

$$\mathbf{Spec} = Init \land \square[Next]_{vars}$$

$$\mathbf{THEOREM}\ Spec \Rightarrow \square(CPU1!LowObs = CPU2!LowObs)$$

© 2020 Arm Limited (or its affiliates)

Confidentiality under Speculative Execution (high *mode*)



LowObs identical

h2 contains secret data

Different values at *h2* leads to *different* low addresses

LowObs different

© 2020 Arm Limited (or its affiliates)

Security Vulnerabilities in the Abstract Machine

- **Spectre v1**: *confidentiality property* violated by the *speculating machine (SpecCPU)*
  - Software workarounds aim to make *speculative execution* deterministic of only non-confidential state
- **Spectre v2**: *integrity* of the victim's *speculating machine* affected through branch predictor training by the attacker
  - Software workarounds to make the *speculative execution trace* deterministic of the high state and high *mode* input only
- **Meltdown (v3)**: address space isolation not guaranteed by the *speculating machine* (*AccessOK* in a low *mode speculated Load* instruction is *TRUE* for high addresses)
  - Software workaround to enforce *AccessOK* by other means like KPTI (kernel page table isolation)
- **Spectre v4** (Speculative Store Bypass): similar to Spectre v1 where the *speculating machine* can read older instances of *mem* (prior to *executed Store* instructions)
  - SSBD as hardware workaround, fine-grained SSBB as software workaround

© 2020 Arm Limited (or its affiliates)

Effects of (ARM) Barriers on the Abstract Machine

- **CSDB** (Consumption of Speculative Data Barrier): disallows the ***Havoc*** set of instructions (or part of – data value prediction) in the *speculating machine*
- **SSBB** (Speculative Store Bypass Barrier): prevents ***Load*** instructions in the *speculating machine* from reading older instances of ***mem*** prior to an *executed* ***Store*** instruction
- **SB** (Speculation Barrier): prevents subsequent instructions in the *speculating machine* from changing the output of an *observation function* (e.g. *LowObs*)

© 2020 Arm Limited (or its affiliates)

Notable References

- Arm Limited. *Vulnerability of Speculative Processors to Cache Timing Side-Channel Mechanism* whitepaper
- Lamport. *Specifying Systems*
- Subramanyan et al. *A Formal Foundation for Secure Remote Execution of Enclaves*
- Guarnieri et al. *SPECTECTOR: Principled Detection of Speculative Information Flows*
- Several other papers on non-interference and observational determinism
- Catalin Marinas. TLA$^+$ model of the cache speculation side-channels
  http://procode.org/cachespec/

© 2020 Arm Limited (or its affiliates)

arm

Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
شكرًا
ধন্যবাদ
תודה

© 2020 Arm Limited (or its affiliates)