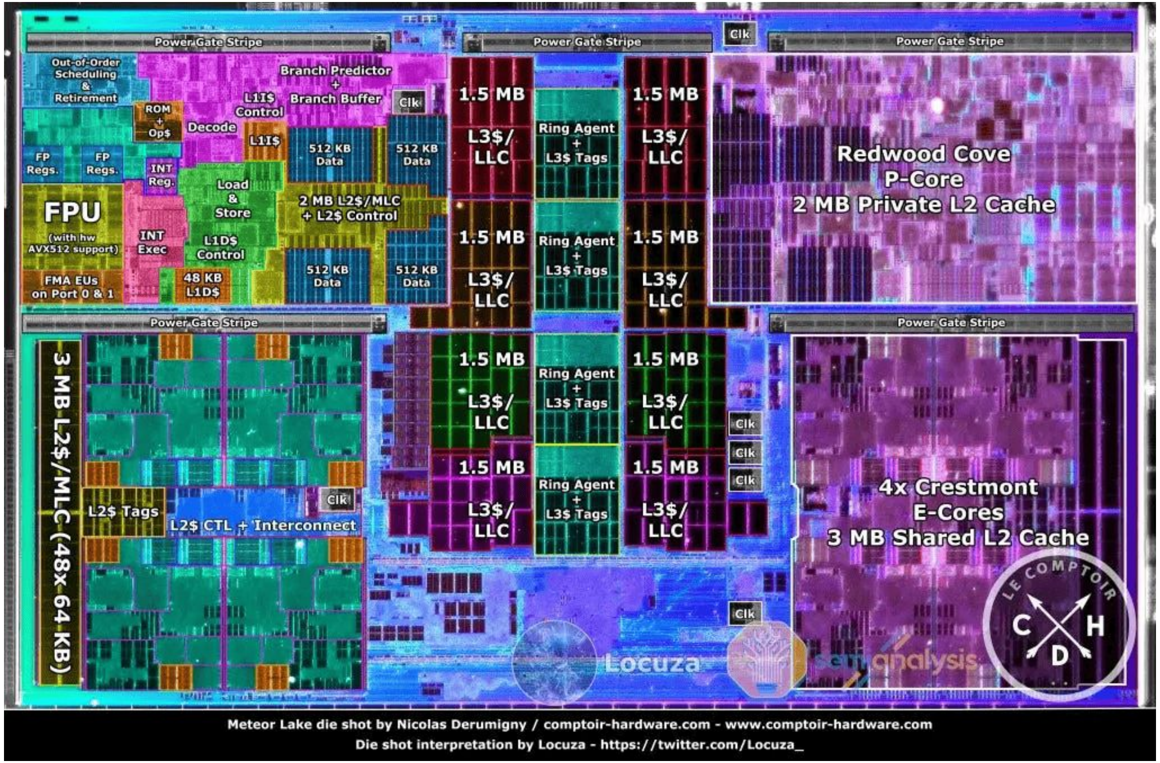
Hardware-software co-designs for microarchitectural security

Summer Research Institute 2025 (SuRI)

EPFL – June 12, 2025

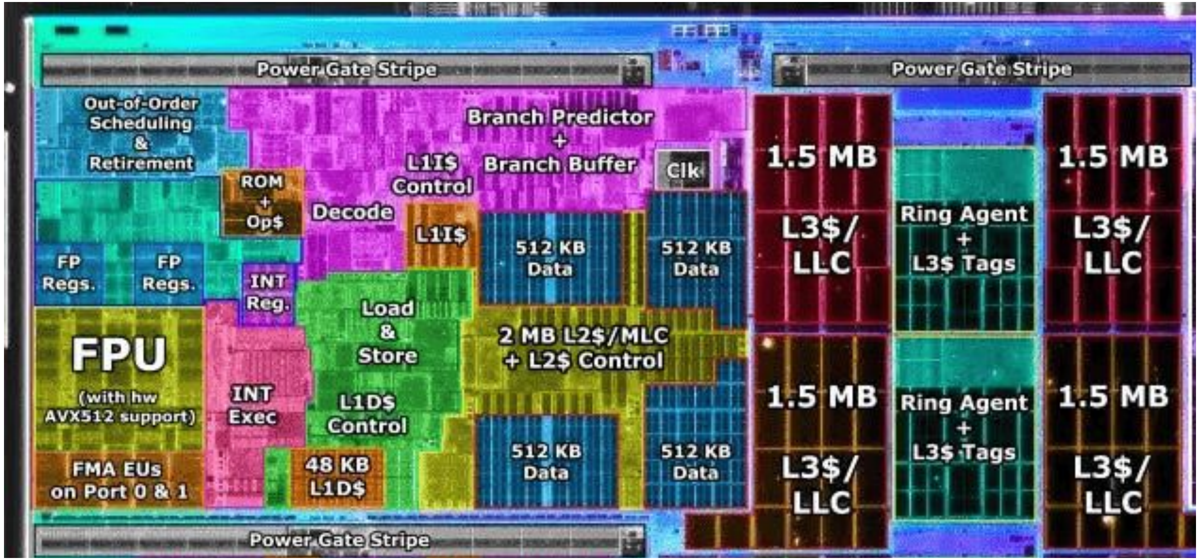
Lesly-Ann Daniel, KU Leuven

Processors are full of optimizations



Intel Meteor Lake – Credit https://semianalysis.com/2022/05/26/meteor-lake-die-shot-and-architecture/

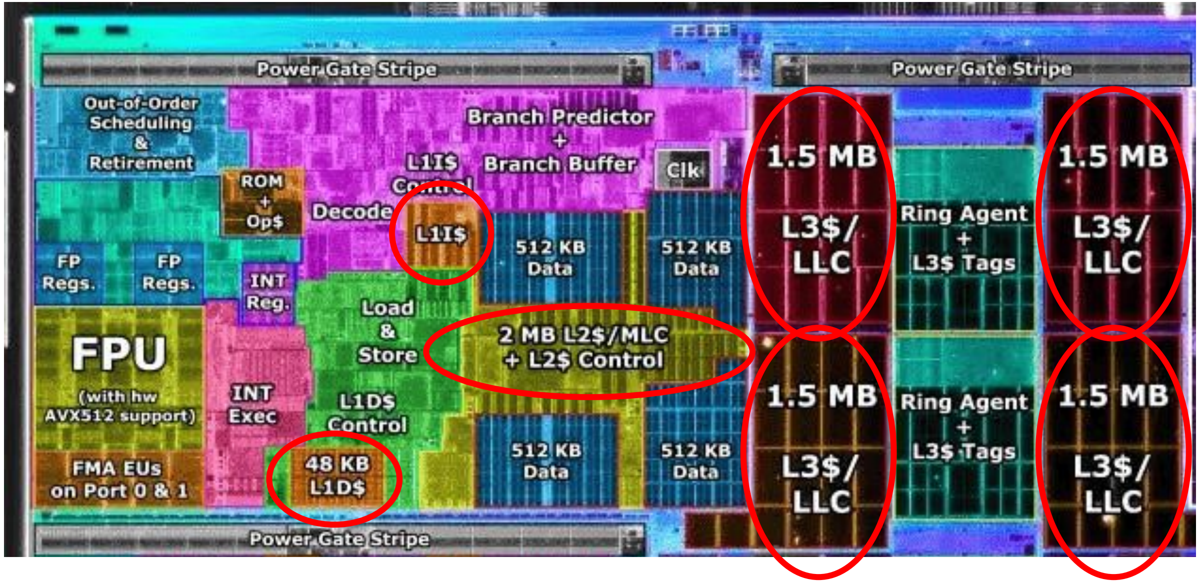
Processors are full of optimizations



Intel Meteor Lake – Credit https://semianalysis.com/2022/05/26/meteor-lake-die-shot-and-architecture/

Processors are full of optimizations

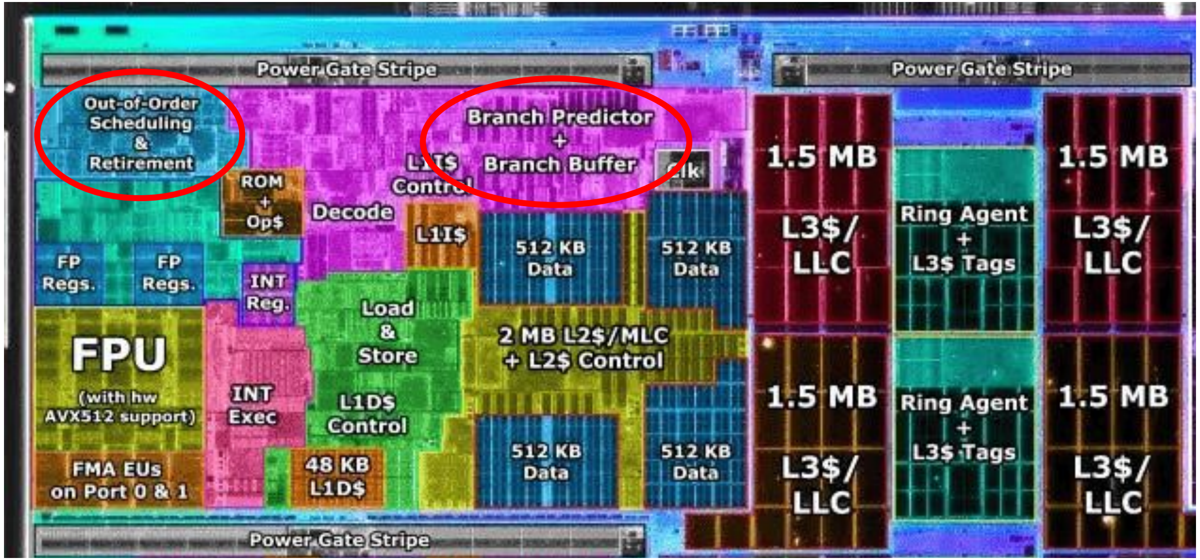
- Caches



Intel Meteor Lake – Credit https://semianalysis.com/2022/05/26/meteor-lake-die-shot-and-architecture/

Processors are full of optimizations

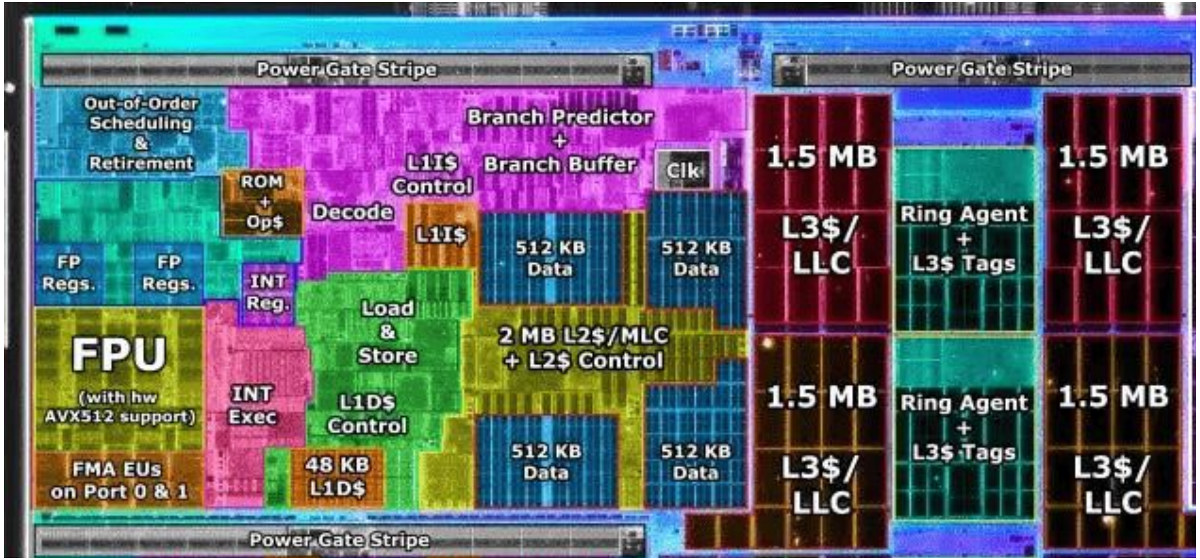
- Caches
- Out-of-order speculative execution



Intel Meteor Lake – Credit https://semianalysis.com/2022/05/26/meteor-lake-die-shot-and-architecture/

Processors are full of optimizations

- Caches
- Out-of-order speculative execution
- And more [1]?



Intel Meteor Lake – Credit https://semianalysis.com/2022/05/26/meteor-lake-die-shot-and-architecture/

[1] Vicarte, Jose Rodrigo Sanchez, et al. "Opening pandora's box: A systematic study of new ways microarchitecture can leak private data." ISCA, 2021

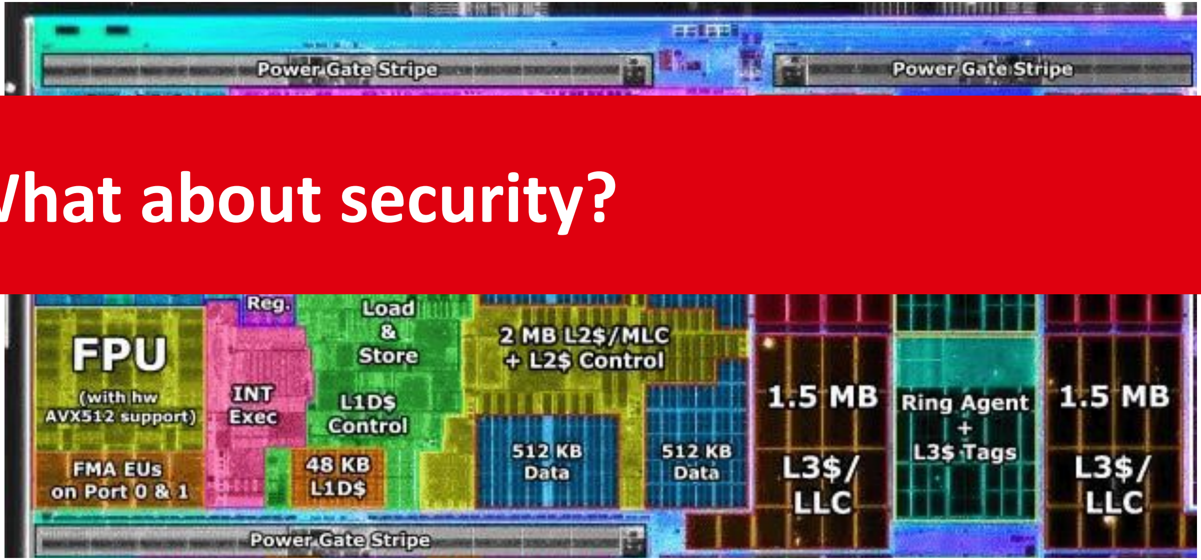
Processors are full of optimizations

- Caches

What about security?

execution

- And more [1]?



Intel Meteor Lake – Credit https://semianalysis.com/2022/05/26/meteor-lake-die-shot-and-architecture/

[1] Vicarte, Jose Rodrigo Sanchez, et al. "Opening pandora's box: A systematic study of new ways microarchitecture can leak private data." ISCA, 2021

… Well security is not good :(

CNN BUSINESS Markets Tech Media Success Video

Major chip flaws affect billions of devices

by Selena Larson @CNNTech

January 4, 2018: 9:44 AM ET

BBC

Home News Sport Business Innovation Culture Arts Travel Earth Audio Video Live

'Foreshadow' attack affects Intel chips

15 August 2018

Share Save

Dave Lee
North America technology reporter

Spectre flaws continue to haunt Intel and AMD as researchers find fresh attack method

The indirect branch predictor barrier is less of a barrier than hoped

Thomas Claburn Fri 18 Oct 2024 // 14:01 UTC

GoFetch

non exhaustive list

Back to the basics

Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems

Paul C. Kocher

Cryptography Research, Inc.
607 Market Street, 5th Floor, San Francisco, CA 94105, USA.
E-mail: paul@cryptography.com.

Abstract. By carefully measuring the amount of time required to perform private key operations, attackers may be able to find fixed Diffie-Hellman exponents, factor RSA keys, and break other cryptosystems. Against a vulnerable system, the attack is computationally inexpensive and often requires only known ciphertext. Actual systems are potentially at risk, including cryptographic tokens, network-based cryptosystems, and other applications where attackers can make reasonably accurate timing measurements. Techniques for preventing the attack for RSA and Diffie-Hellman are presented. Some cryptosystems will need to be revised to protect against the attack, and new protocols and algorithms may need to incorporate measures to prevent timing attacks.

1996

Cache-timing attacks on AES

Daniel J. Bernstein *

Department of Mathematics, Statistics, and Computer Science (M/C 249)
The University of Illinois at Chicago
Chicago, IL 60607–7045
djb@cr.yp.to

Abstract. This paper demonstrates complete AES key recovery from known-plaintext timings of a network server on another computer. This attack should be blamed on the AES design, not on the particular AES library used by the server; it is extremely difficult to write constant-time high-speed AES software for common general-purpose computers. This paper discusses several of the obstacles in detail.

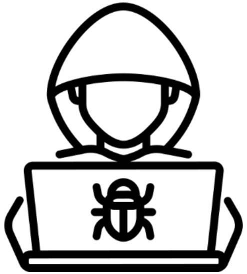
2005

Memory accesses leak

Victim program

```
x = tab[secret]
```

Data cache



Attacker
Shares microarchitecture with victim

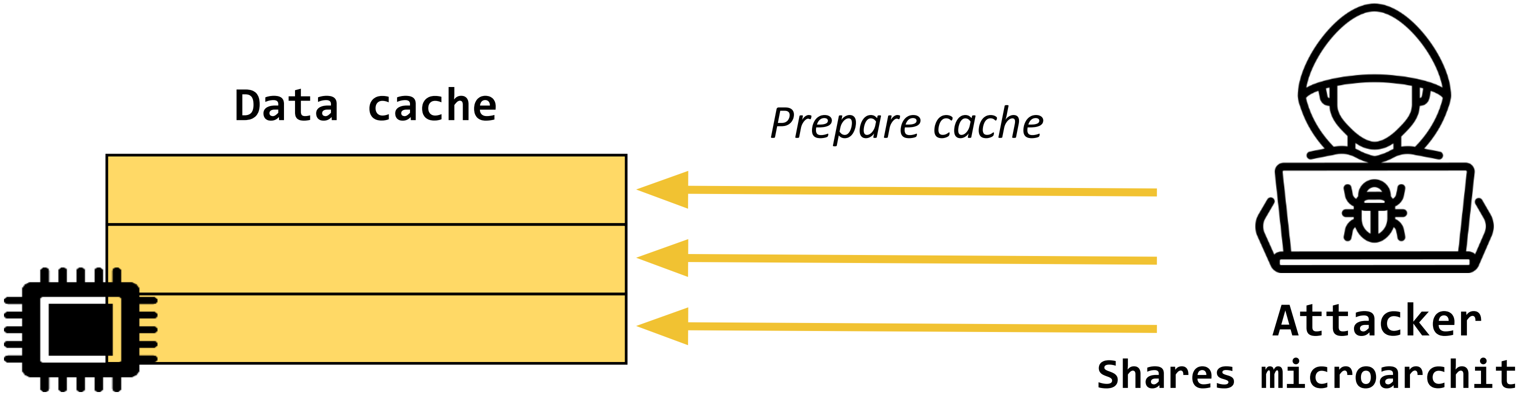
Memory accesses leak

Victim program

```
x = tab[secret]
```

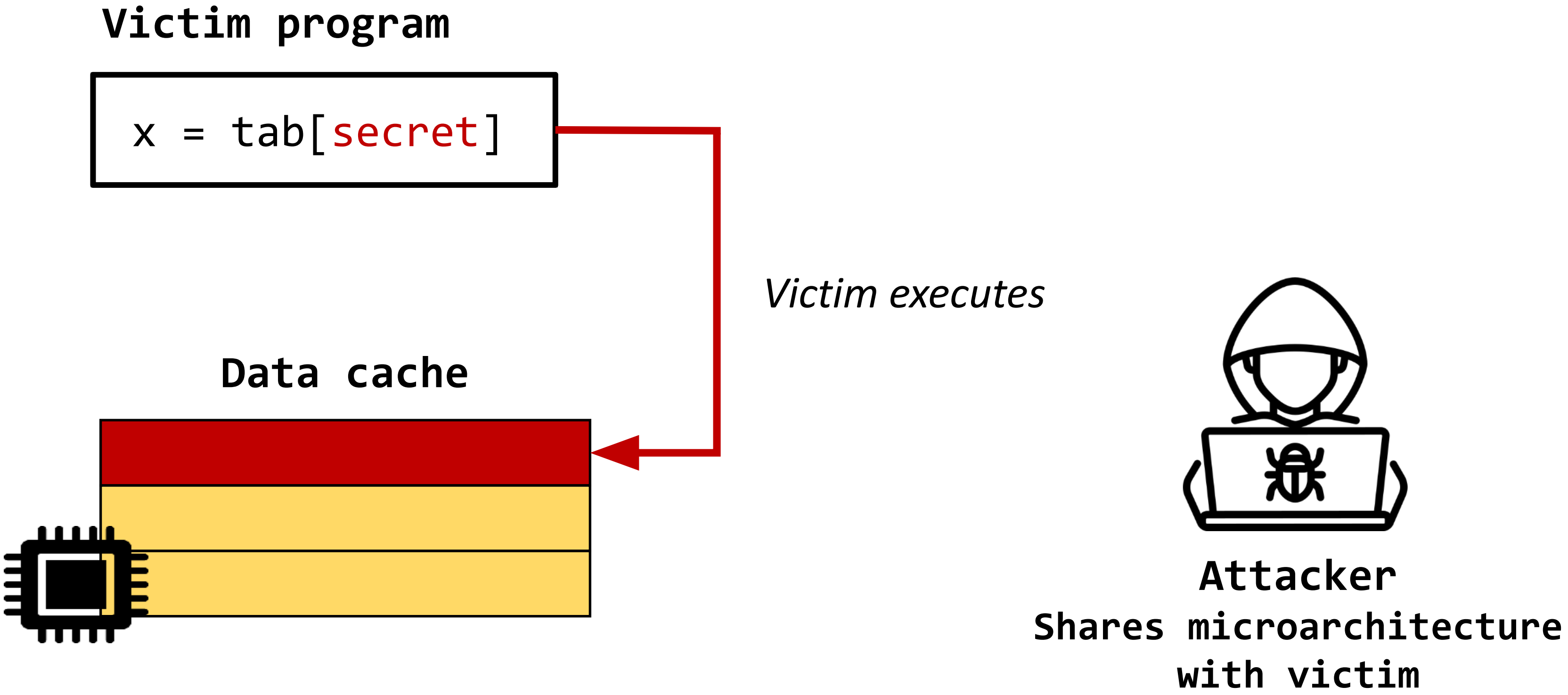
Data cache

Prepare cache



Attacker
Shares microarchitecture with victim

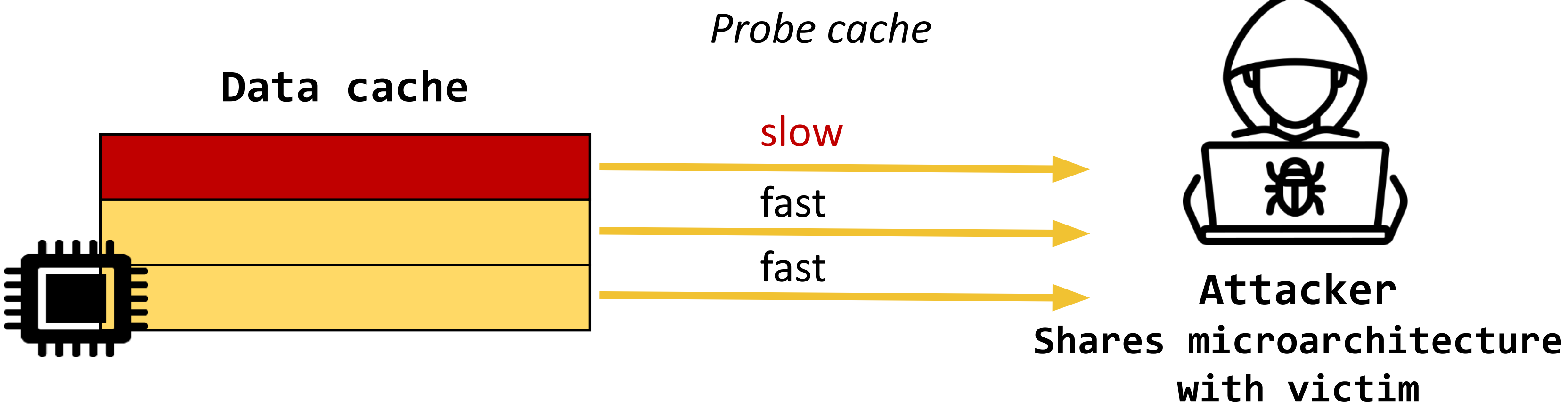
Memory accesses leak



Memory accesses leak

Victim program

```
x = tab[secret]
```

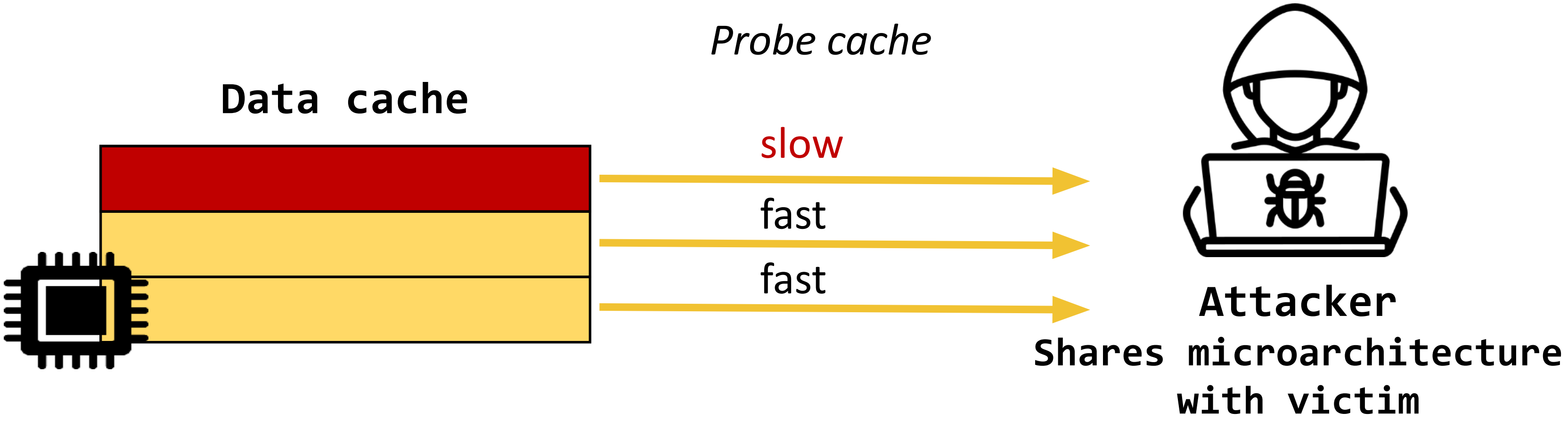


Memory accesses leak

Victim program

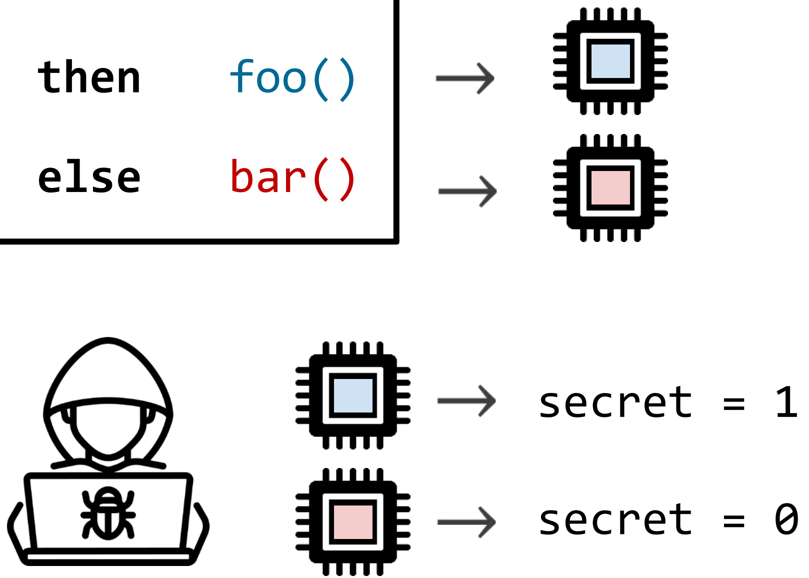
```
x = tab[secret]
```

- caches
- data pre-fetchers
- load/store dependencies
- …



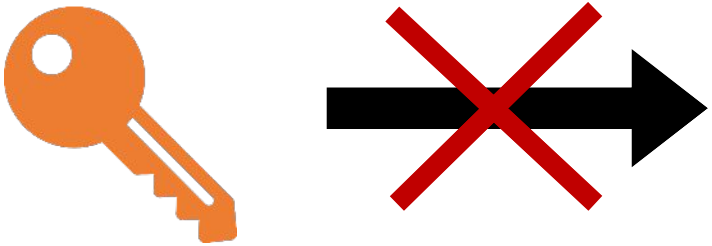
Control-flow leaks

```
if secret
then    foo()
else    bar()
```



- end-to-end timing
- different resource consumption
- branch predictor state
- instruction cache
- instruction prefetcher
- micro-op cache
- …

Solution? Constant-time programming!



Unsafe instructions

- Control-Flow
- Memory accesses
- Variable-time instr.

- Full software countermeasure
- De facto standard for crypto: BearSSL, Libsodium, HACL*, etc.
- Believed to be secure …

… Until it was broken :(

CNN BUSINESS Markets Tech Media Success Video

Major chip flaws affect billions of devices

by Selena Larson @CNNTech

January 4, 2018: 9:44 AM ET

BBC

Home News Sport Business Innovation Culture Arts Travel Earth Audio Video Live

'Foreshadow' attack affects Intel chips

15 August 2018

Share Save

Dave Lee
North America technology reporter

Spectre flaws continue to haunt Intel and AMD as researchers find fresh attack method

The indirect branch predictor barrier is less of a barrier than hoped

Thomas Claburn Fri 18 Oct 2024 // 14:01 UTC

… Until it was broken :(

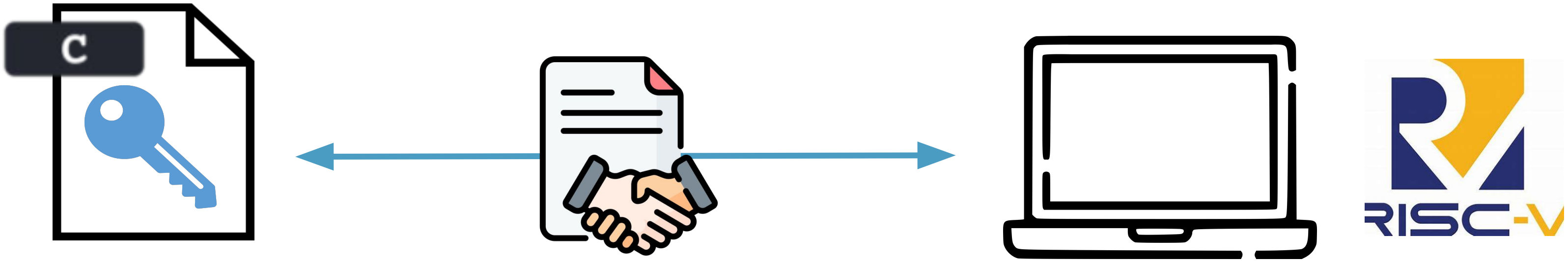
Some attacks stem from performance-critical optimizations!

Should we just disable optimizations?

New research opportunities!

Hardware-software co-design

Investigate more secure and performant defenses against microarchitectural attacks



Proteus: An Extensible RISC-V Core for Hardware Extensions

(RISC-V Summit ’23)

Marton Bognar, Job Noorman, Frank Piessens

A modular textbook processor to study HW extensions

- **In/Out-of order** pipelines
- **Optimizations**: branch predictors, cache, prefetchers, …
- **Configurable**: #exec units, ROB size, …
- **Extensible**: plugin system
- **SpinalHDL** □ verilog □ FPGA / simulator

HW/SW Co-Designs for End-to-End Security

ProSpeCT: Provably Secure Speculation for the Constant-Time Policy

Lesly-Ann Daniel[1], Marton Bognar[1], Job Noorman[1], Sébastien Bardin[2], Tamara Rezk[3] and Frank Piessens[1]

[1]imec-DistriNet, KU L
[2]CEA, List, Unive
[3]INRIA, Université Côte

Abstract

We propose ProSpeCT, a generic formal processor mod providing provably secure speculation for the constant-tin policy. For constant-time programs under a *non-speculati* semantics, ProSpeCT guarantees that speculative and out-c order execution cause no microarchitectural leaks. This gu antee is achieved by tracking secrets in the processor pipelir and ensuring that they do not influence the microarchitectur state during speculative execution. Our formalization cove echanisms, generalizing pri proof covers all known Spect jection (LVI) attacks.

USENIX'23

Libra: Architectural Support For Principled, Secure And Efficient Balanced Execution On High-End Processors

Hans Winderix
frank.piessens@kuleuven.be
DistriNet, KU Leuven
Leuven, Belgium

Marton Bognar
frank.piessens@kuleuven.be
DistriNet, KU Leuven
Leuven, Belgium

Lesly-Ann Daniel
frank.piessens@kuleuven.be
DistriNet, KU Leuven
Leuven, Belgium

Frank Piessens
frank.piessens@kuleuven.be
DistriNet, KU Leuven
Leuven, Belgium

ABSTRACT

Control-flow leakage (CFL) attacks enable an attacker to expose control-flow decisions of a victim program via side-channel obser- vations. *Linearization* (i.e., elimination) of secret-dependent control against these attacks, yet it comes ely, *balancing* secret-dependent verhead, but is notoriously inse-

KEYWORDS

Microarchitectural Side-Channels, Control-Flow Leakage, HW/SW Leakage Contracts, HW/SW Codesign, Secure Compilation, Control-Flow Balancing

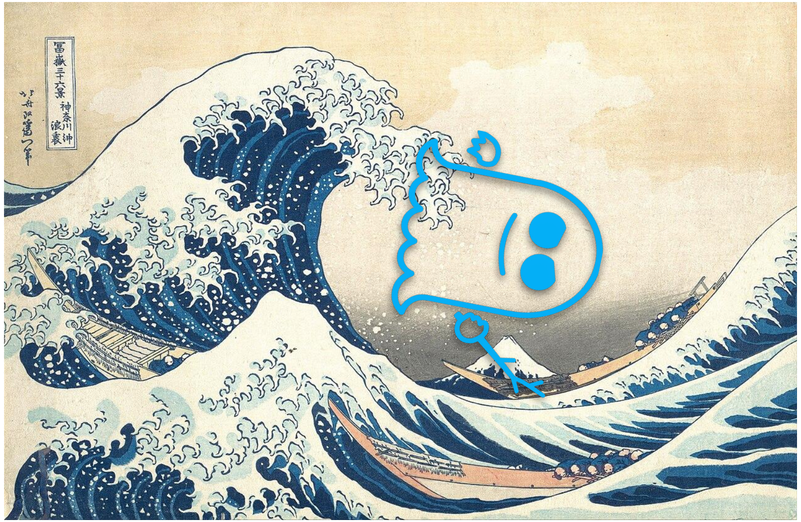
ACM Reference Format:
Hans Winderix, Marton Bognar, Lesly-Ann Daniel, and Frank Piessens. 2018. Libra: Architectural Support For Principled, Secure And Efficient Bal-

CCS'24

ProSpeCT
Provably Secure Speculation for the Constant-Time Policy

Lesly-Ann Daniel, Marton Bognar, Job Noorman, Sébastien Bardin, Tamara Rezk, Frank Piessens

KU Leuven, Inria, CEA



USENIX'23

Constant-time is vulnerable to Spectre

```
char array[len]
char mysecret
if (idx < len)
    x = array[idx]
    load(x)
```

Constant-time is vulnerable to Spectre

```
char array[len]
char mysecret
if (idx < len)
    x = array[idx]
    load(x)
```

Predict condition true

Consider `idx = len`

Constant-time is vulnerable to Spectre

```
char array[len]
char mysecret
if (idx < len)
      x = array[idx]
      load(x)
```

Consider `idx = len`

Predict condition true

`x = mysecret`

Constant-time is vulnerable to Spectre

```
char array[len]
char mysecret
if (idx < len)
    x = array[idx]
    load(x)
```

Consider `idx = len`

Predict condition true

`x = mysecret`

Leak `mysecret` to microarchitecture!

How can I protect my code?

Constant-Time Foundations for the New Spectre Era

Sunjay Cauligi[†] Craig Disselkoen[†] Klaus v. Gleissenthall[†]
Dean Tullsen[†] Deian Stefan[†] Tamara Rezk[⋆] Gilles Barthe[♠♣]
[†]UC San Diego, USA [⋆]INRIA Sophia Antipolis, France
[♠]MPI for Security and Privacy, Germany [♣]IMDEA Software Institute, Spain

Speculative constant-time

- Hard to reason about
- New speculation mechanisms?

How can I protect my code?

Constant-Time Foundations for the New Spectre Era

Sunjay Cauligi[†] Craig Disselkoen[†] Klaus v. Gleissenthall[†]
Dean Tullsen[†] Deian Stefan[†] Tamara Rezk[★] Gilles Barthe[♠♣]
[†]UC San Diego, USA [★]INRIA Sophia Antipolis, France

Need security for CT code!

- Hard to reason about
- New speculation mechanisms?

We need Secure Speculation for Constant-Time!

 Developers should not care about speculations

 Hardware shall not speculatively leak secrets

 But still be efficient and enable speculation

Hardware Secrecy Tracking

Software side

- Label secrets
- Constant-time program

Hardware side

- Track security labels
- Secrets do not speculatively flow to unsafe instructions

ConTExT: A Generic Approach for Mitigating Spectre

Michael Schwarz[1], Moritz Lipp[1], Claudio Canella[1]

[1]Graz University of Technolo

SpectreGuard: An Efficient Data-centric Defense Mechanism against Spectre Attacks

Farzad Farshchi, University of Kansas

Heechul Yun, University of Kansas

Speculative Privacy Tracking (SPT): Leaking Information From Speculative Execution Without Compromising Privacy

Rutvik Choudhary, UIUC, USA

Jiyong Yu, UIUC, USA

Christopher W. Fletcher, UIUC, USA

Adam Morrison, Tel Aviv University, Israel

Illustration with Spectre-v1

```
char array[len]
secret char mysecret
if (idx < len)
    x = array[idx]
    load(x)
```

Consider `idx = len`

Illustration with Spectre-v1

```
char array[len]
secret char mysecret
if (idx < len)
     x = array[idx]
     load(x)
```

Developer marks secrets

Consider `idx = len`

Illustration with Spectre-v1

```
char array[len]
secret char mysecret
if (idx < len)
    x = array[idx]
    load(x)
```

Developer marks secrets

Speculative execution 

Consider `idx = len`

Illustration with Spectre-v1

```
char array[len]
secret char mysecret
if (idx < len)
     x = array[idx]
     load(x)
```

Developer marks secrets

Speculative execution

`x = mysecret:`**`secret`**

Consider `idx = len`

Illustration with Spectre-v1

```
char array[len]
secret char mysecret
if (idx < len)
    x = array[idx]
    load(x)
```

Consider `idx = len`

Developer marks secrets

Speculative execution

`x = mysecret:`**`secret`**

Speculative execution + **`secret`**

=

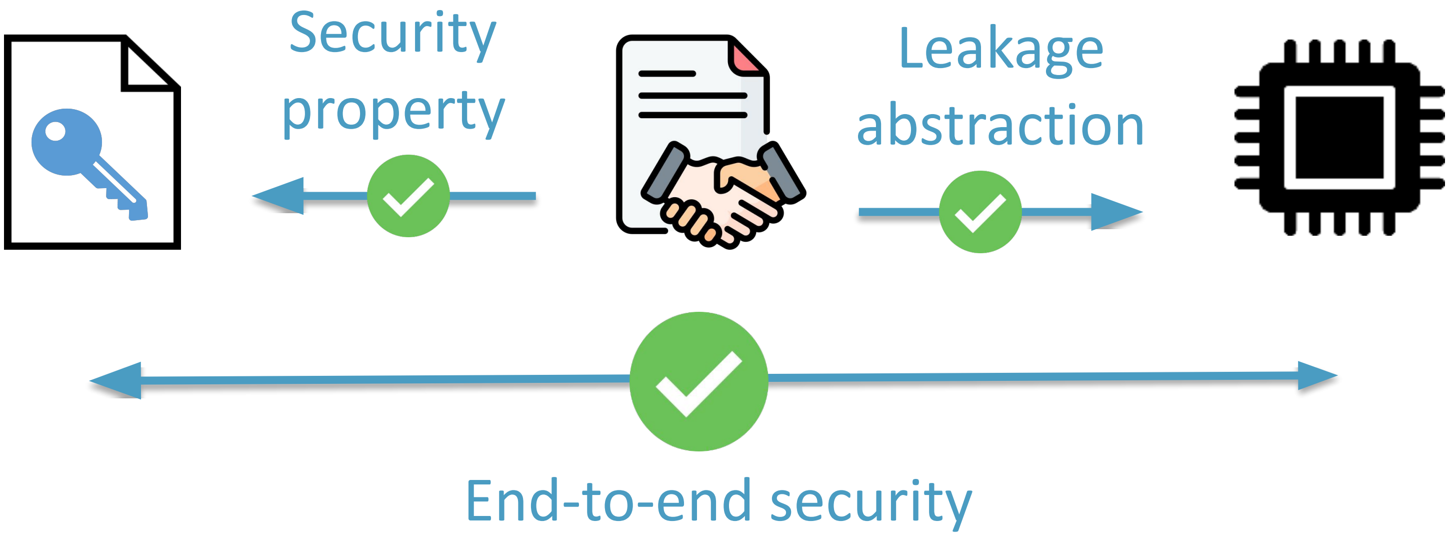
`x` *not* forwarded to `load`

How do I know that my defense works?

How do I know that my defense works?

Hardware-Software Contracts for Secure Speculation

Marco Guarnieri[*], Boris Köpf[†], Jan Reineke[‡], and Pepe Vila[*]
[*]*IMDEA Software Institute* [†]*Microsoft Research* [‡]*Saarland University*



ProSpeCT: Generic formal processor model for HST

Semantics of generic out-of-order speculative processor with HST

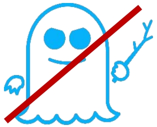
→ Abstract microarchitectural context

→ Functions *update*, *predict*, *next*

All public values are leaked / influence predictions

→ Captures all known variants of Spectre

→ And futuristic mechanisms Load Value Prediction



Security proof

Constant-time programs (ISA semantics)
do not leak secrets (microarchitectural semantics)

Load Prediction: Rollback correct executions?

```
char secret mysecret
x = load mysecret
y = x + 4
```

Load Prediction: Rollback correct executions?

```
char secret mysecret
x = load mysecret
y = x + 4
```

Predict x = 0 

Compute y = 4

Load Prediction: Rollback correct executions?

```
   char secret mysecret
1: x = load mysecret
2: y = x + 4
```

Predict `x` = 0 

Compute `y` = 4

Resolve prediction:

- if `mysecret` = 0: Commit and continue to line 3
- if `mysecret` != 0: Rollback to line 1

That leaks!

Load Prediction: Rollback correct executions?

```
   char secret mysecret
1: x = load mysecret
2: y = x + 4
```

Predict `x` = 0 

Compute `y` = 4

Resolve prediction:

- if `mysecret` = 0: Rollback to line 1
- if `mysecret` != 0: Rollback to line 1

Always rollback when actual value is secret

Implementation on Proteus and Evaluation

Performance overhead [1]

Speculation/Crypto	25/75	50/50	75/25	90/10
Precise (Key)	**0%**	**0%**	**0%**	**0%**
Conservative (All)	10%	25%	36%	45%

No overhead in SW for CT code
when secrets are precisely annotated

Hardware Cost:

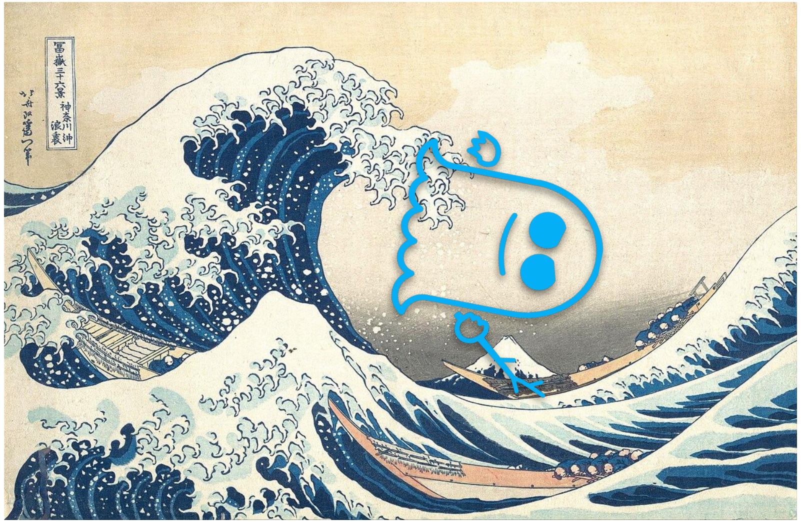
Synthesized on FPGA

- LUTs: +17%
- Registers: +6%
- Critical path: +2%

[1] Jacob Fustos, Farzad Farshchi, and Heechul Yun. “SpectreGuard: An Efficient Data-Centric Defense Mechanism against Spectre Attacks”. In: DAC. 2019

Did we get rid of Spectre?

- Compiler support
  - Partition secret/public
  - Extensive evaluation
- Extension to new optimizations
- Hardware verification
- Lightweight HW defenses?



Libra

Architectural Support for Principled, Secure and Efficient Balanced Execution on High-End Processors

Hans Winderix, Marton Bognar, Lesly-Ann Daniel, Frank Piessens

KU Leuven

CCS'24

Libra

Dream of secure balanced executions?
Let's make it real!

Hans Winderix, Marton Bognar, Lesly-Ann Daniel, Frank Piessens

KU Leuven

CCS'24

State of the art software countermeasures

Vuln. code

```
beq s1 a0 Target
    add a1 a2 a3
    j End
Target:
    add a2 a3 a4
End:
```

Linearization [1]

```
sub t0 s1 a0
setqz t0 t0
addi t0 t0 -1 ;tt mask
not t1 t0     ;ff mask
and t2 a1 t0
add a1 a2 a3
and a1 a1 t1
or  a1 a1 t2
and t2 a2 t1
add a2 a3 a4
and a2 a3 t0
or  a2 a2 t2
```

Balancing

```
beq s1 a0 Target
    add a1 a2 a3
    j End
Target:
    add a2 a3 a4
    j End
End:
```

=

[1] Molnar et al., The program counter security model: Automatic detection and removal of control-flow side channel attacks (ICISC 2005)

Branch balancing, are you kidding me?

"What about branch predictors or instruction caches?"
– Any side-channel expert

"We all know it's insecure on high-end processors!"
– Any reasonable cryptographer

Antoon Purnal
@PurnalToon

Supreme Court votes 6-3 in favor of branching on secrets in cryptographic code

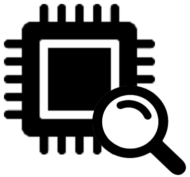
8:01 p.m. · 30 jun. 2022

Branch balancing, are you kidding me?

"But actually why not?"

– Hopeful dreamer

What would it take to balance branches on modern CPUs?



What **microarchitectural features** leak control-flow?

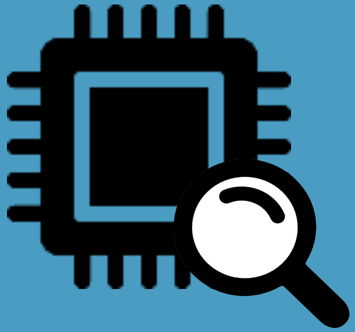
→ **Characterization** of HW sources of **control-flow leakage**

Libra: Architectural support for balanced execution

Can it improve **performance** over linearization?

→ HW **implementation** & evaluation (19.3% less overhead)

Characterization HW sources of control-flow leakage



Literature review

65 attack papers

29 optimizations

Balanceable leakage

Independent of pc

- instruction latency
- data cache
- data TLB
- loads/store buffer dep.
- data dependencies
- …

➔ can be handled in SW ☺️

➔ but not in a principled way ☹️

Unbalanceable leakage

Dependent of pc

- instruction cache
- instruction TLB
- instruction prefetcher
- branch predictors
- µ-op caches
- …

➔ cannot be handled in SW ☹️

Balanceable leakage

Independent of pc

Unbalanceable leakage

Dependent of pc

Disable optims. producing unbalanceable leakage?

Balanceable leakage:

- loads/store buffer dep.
- data dependencies
- …

→ can be handled in SW ☺️

→ but not in a principled way ☹️

Unbalanceable leakage:

- branch predictors
- µ-op caches
- …

→ cannot be handled in SW ☹️

Balanceable leakage

Independent of pc

Unbalanceable leakage

Dependent of pc

Disable optims. producing unbalanceable leakage?

- loads/store buffer dep.
- branch predictors

No! We handle unbalanceable leakage with new HW/SW co-design!

→ can be handled in SW 🙂

→ but not in a principled way ☹

Libra: a new HW/SW co-design for balancing

SW handles balanceable leakage

HW/SW Contract for balanced execution

HW support to address unbalanceable leakage **efficiently**

2-D Leakage contract for balanced executions

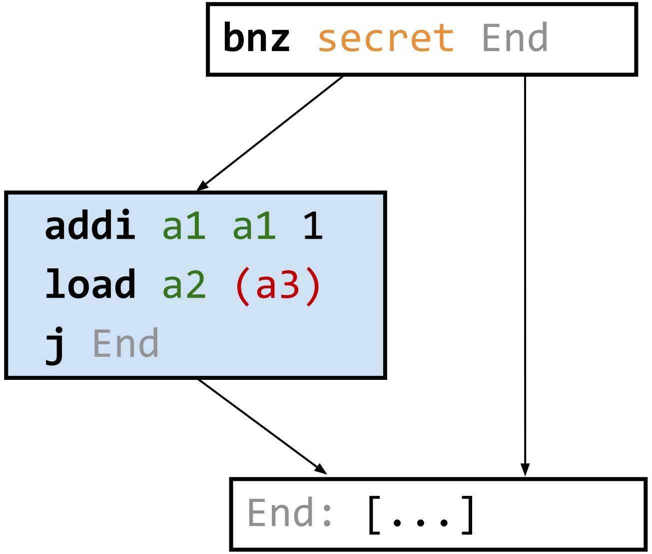
1. Leakage classes

- same observation – `add x1 x1 x2` ~ `sub x1 x1 x2`
- *dummy (no-op)* instruction for each class – `mv x1 x1`

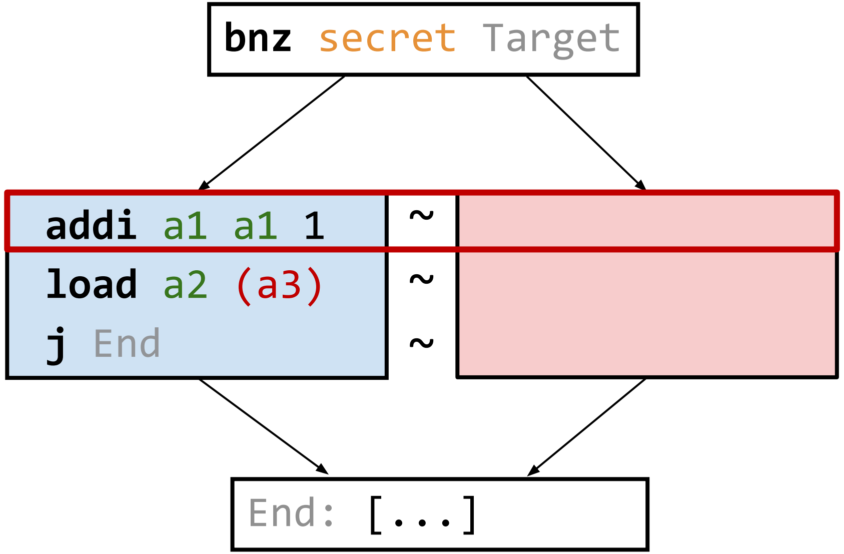
2. Safe/Unsafe instructions

- ***Safe***: timing does not depend on operands – `add x1 x1 x2`
- ***Unsafe***: timing depends on operands – `load x1 (x2)`

.c Software balances secret branches w.r.t. contract

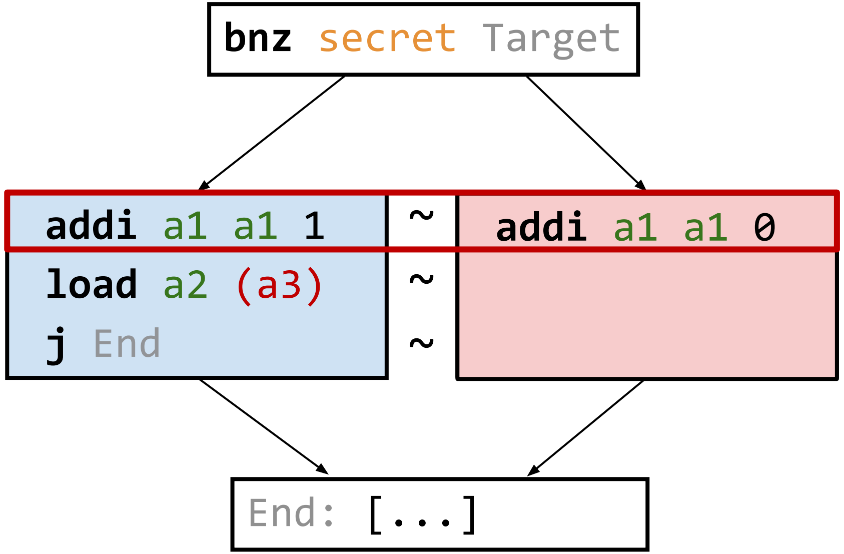


Software balances secret branches w.r.t. contract



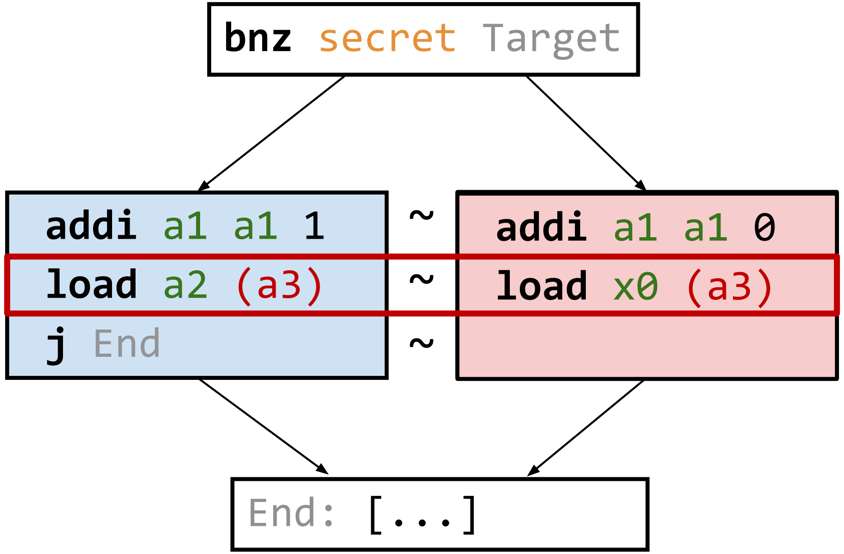
1. Instruction per instruction

.C Software balances secret branches w.r.t. contract



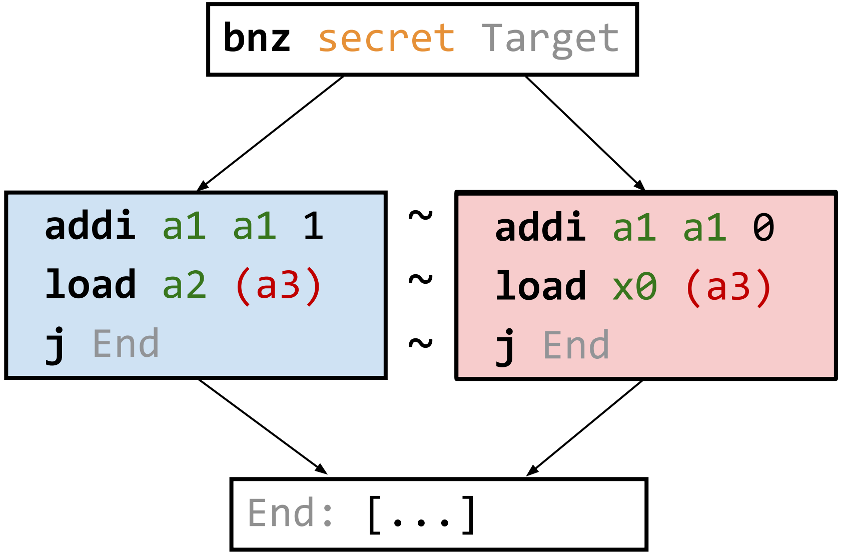
1. Instruction per instruction
2. With dummy instruction in same leakage class

.C Software balances secret branches w.r.t. contract



1. Instruction per instruction
2. With dummy instruction in same leakage class
3. Balance operands of unsafe instructions

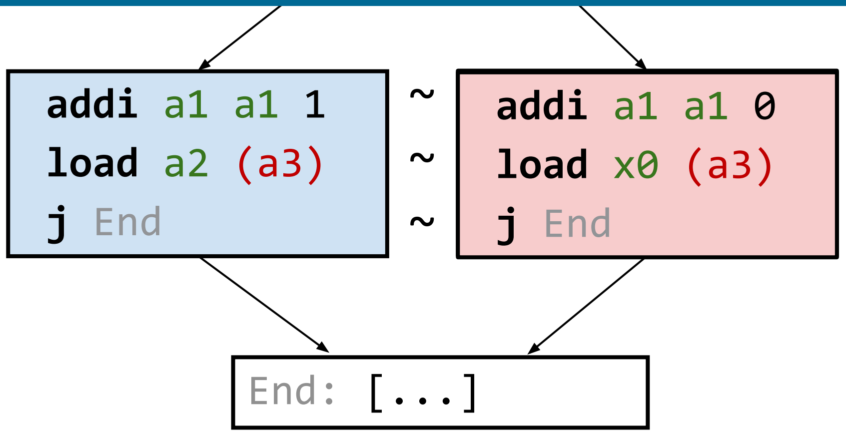
Software balances secret branches w.r.t. contract



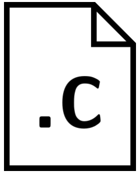
1. Instruction per instruction
2. With dummy instruction in same leakage class
3. Balance operands of unsafe instructions

Software balances secret branches w.r.t. contract

Software secure w.r.t. *balanceable* observervations



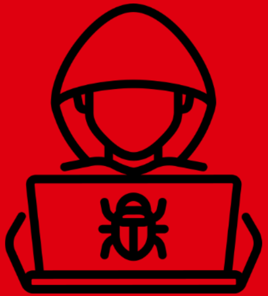
2. With dummy instruction in same leakage class
3. Balance operands of unsafe instructions



Software balances secret branches w.r.t. contract

Software secure w.r.t. *balanceable* observervations

… But still insecure w.r.t. *unbalanceable* observations



I can still see differences
in instruction cache!

Folding transformation

Key Idea: *interleave* secret-dependent branches

```
bnz secret Target
    addi a1 a1 1
    load a2 (a3)
    j End
Target:
    addi a1 a1 0
    load x0 (a3)
    j End
End:
```

```
    add a1 a1 1
    add a1 a1 0
    load a2 (a3)
    load x0 (a3)
    j End
    j End
```

slice

Folding transformation

ISA extension to inform CPU:

- → how to navigate folded region
- → secret region so adapt behavior

```
bnz secret Target
    addi a1 a1 1
    load a2 (a3)
    j End
Target:
    addi a1 a1 0
    load x0 (a3)
    j End
End:
```

```
lo.bnz secret offT:1 offF:0 #bb:2
    add a1 a1 1                  ;pc+2
    add a1 a1 0                  ;pc+2
    load a2 (a3)                 ;pc+2
    load x0 (a3)                 ;pc+2
    lo.beq x0 offT:0 offF:0 #bb:1
    lo.beq x0 offT:0 offF:0 #bb:1
```

.c + Folding transformation

ISA extension to inform CPU:
→ how to navigate folded region
→ secret region so adapt behavior

```
bnz secret Target
```

```
    addi a1 a1 0
    load x0 (a3)
    j End
End:
```

```
lo.bnz secret offT:1 offF:0 #bb:2
```

```
    load x0 (a3)                 ;pc+2
    lo.beq x0 offT:0 offF:0 #bb:1
    lo.beq x0 offT:0 offF:0 #bb:1
```

Important requirement: slice-granular leakage

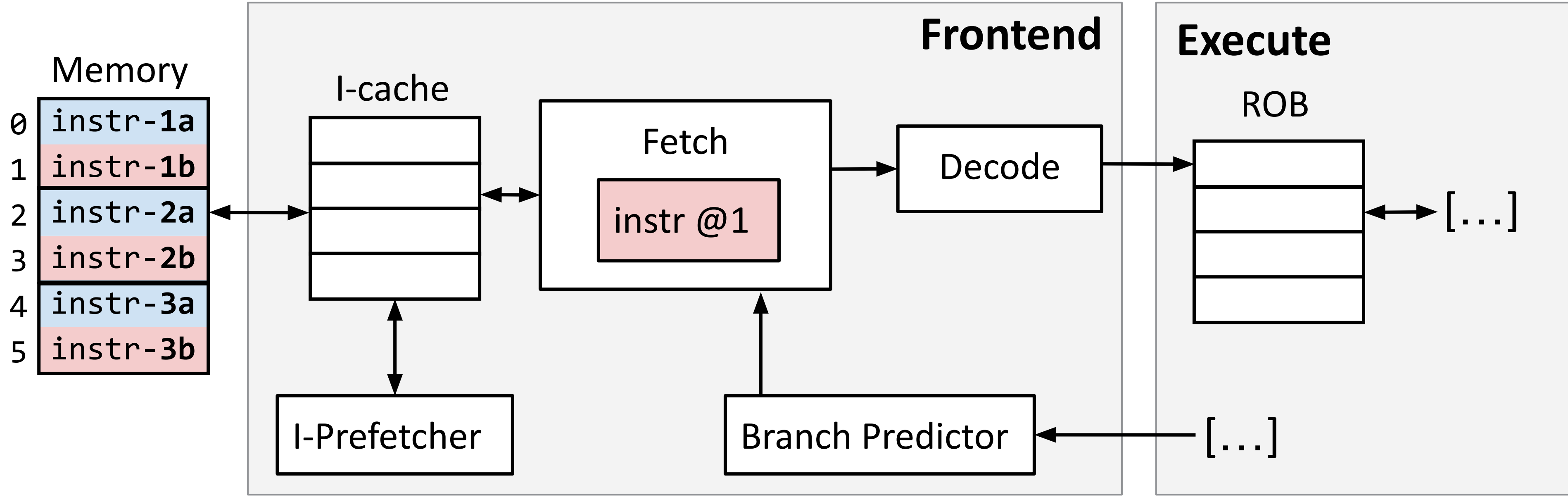
Hardware guarantees slice-granular leakage?

Optimizations producing unbalanceable leakage

- 5 subcategories
  - guidelines to adapt for Libra

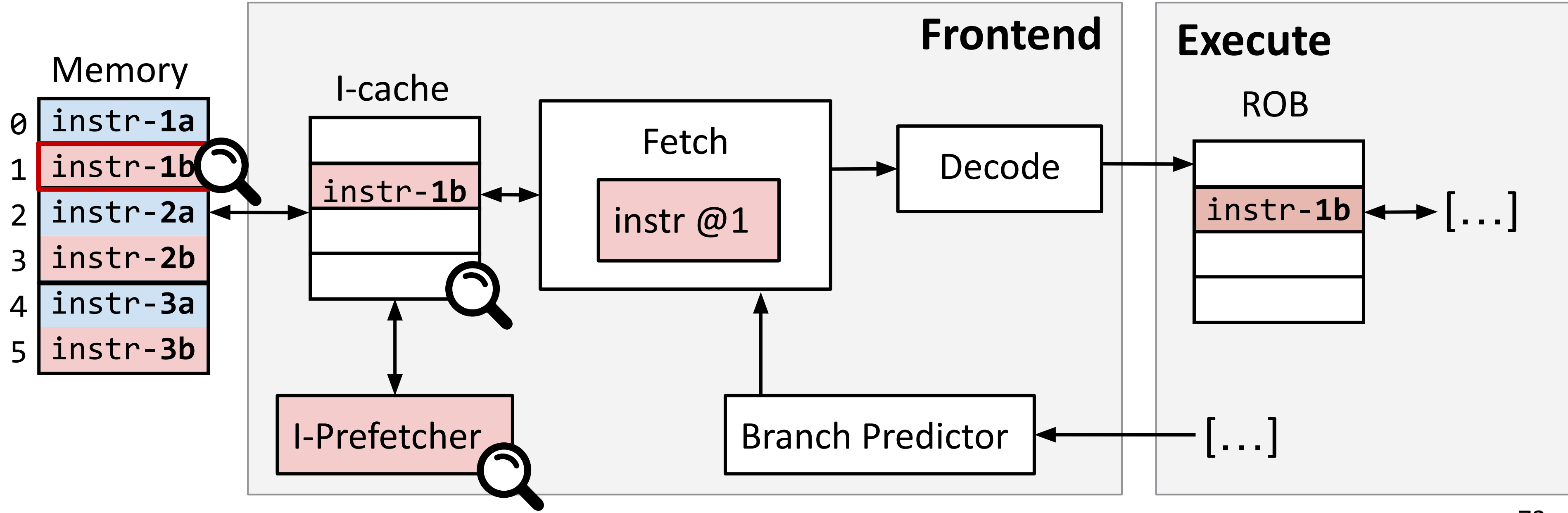
Category: instruction buffering

E.g. I-cache, I-prefetcher, MMU, I-TLB, etc.



Category: instruction buffering

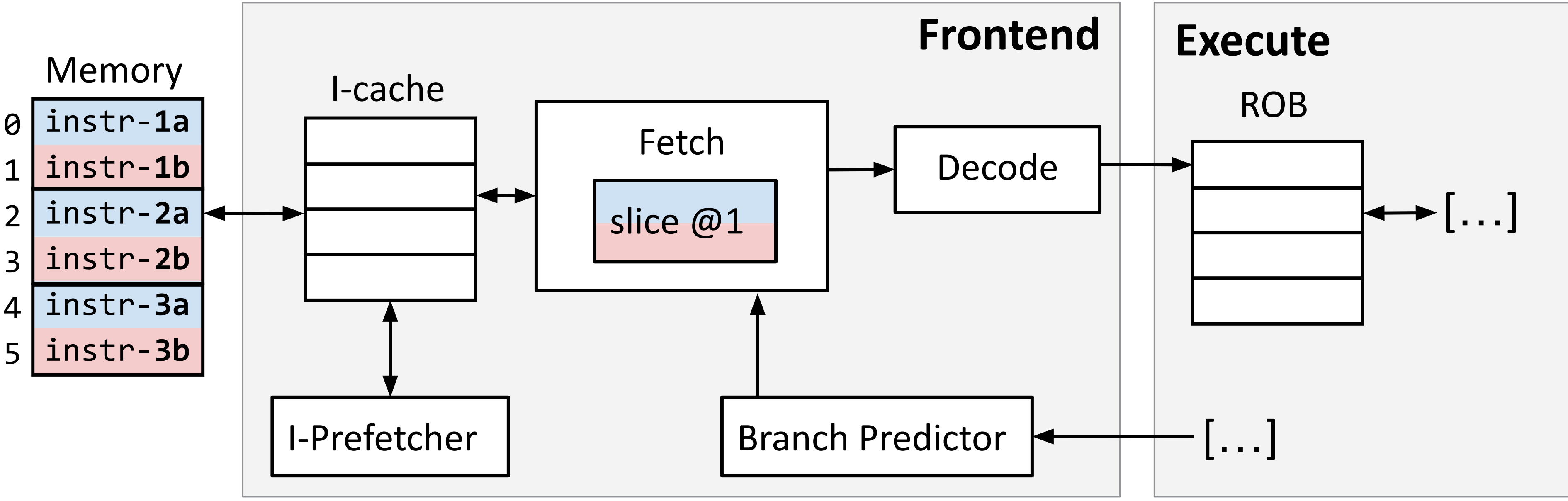
E.g. I-cache, I-prefetcher, MMU, I-TLB, etc.



Category: instruction buffering

E.g. I-cache, I-prefetcher, MMU, I-TLB, etc.

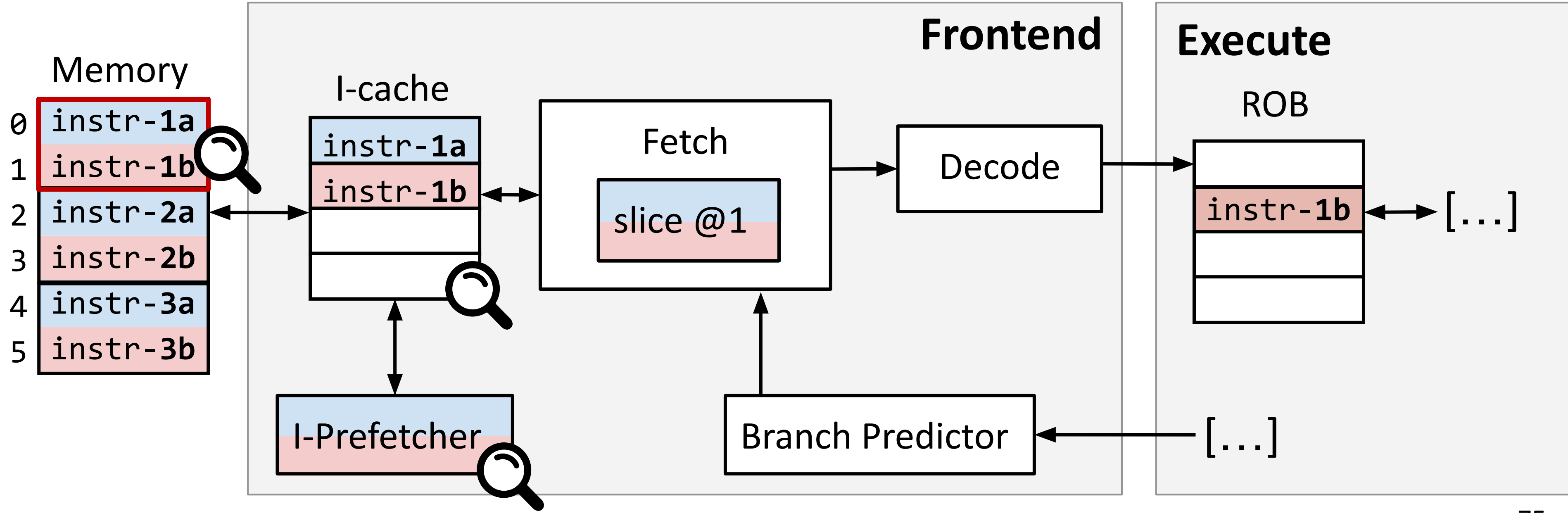
Guideline: slice-granular fetch



Category: instruction buffering

E.g. I-cache, I-prefetcher, MMU, I-TLB, etc.

Guideline: slice-granular fetch



Category: pc-based mappings

E.g. pc-dep prefetcher, branch predictors, etc.

Branch Target Buffer
pc_1 ↦ target_1
pc_2 ↦ target_2
pc_n ↦ target_n

Category: pc-based mappings

E.g. pc-dep prefetcher, branch predictors, etc.

Guideline: slice-based mappings

Branch Target Buffer
pc_1 ↦ target_1
pc_2 ↦ target_2
pc_n ↦ target_n

Branch Target Buffer
slice_pc_1 ↦ slice_target_1
slice_pc_2 ↦ slice_target_2
slice_pc_n ↦ slice_target_n

Evaluation

Q1. Feasibility

Q2. Security

Q3. Performance

Q4. HW cost

Libra implementation on Proteus

Sources of unbalanceable leakage.

- instruction caches
- instruction prefetcher
- branch target predictor

} Libra-aware fetch unit (instruction caches, instruction prefetcher)

→ disable in folded regions (branch target predictor)

Q1. Feasibility ✅

Security evaluation

Benchmark 11 programs [1]

- baseline
- balanced
- linearized
- libra

RTL-level noninterference testing

- Run programs with ≠ secret
- Monitor side-channel signals

Q2. Security ✅

[1] H. Winderix, J. T. Mühlberg, and F. Piessens, "Compiler-assisted hardening of embedded software against interrupt latency side-channel attacks," in EuroS&P, 2021.

Execution time overhead

	Balanced (insecure)	Linearized (secure)	Libra (secure)
Min	+0%	+8%	-2%
Max	+282%	+225%	+227%
Mean	**+42%**	**+56%**	**+45%**

Compared to linearization

-19.3% overhead

Q3. Performance ✅

Hardware Cost (FPGA)

	Base	Libra	Increase
LUT	16.5k	18.4k	**+11%**
Registers	13.6k	14.9k	**+9.5%**
Critical path	37.4ns	37.4ns	**+0%**

Small area increase

No impact on CP

Q4. HW cost ✅

A new era for balancing?

Well, there are still challenges!

- HW verif/synthesis for balancing contracts
- Automatic balancing transformation
- Evaluation on larger benchmarks
- Feasibility with more complex optimizations?

Exploring HW-SW Co-Designs

Let’s take a dive

A common methodology

- Rigorous formalization and security proofs
- Implementations Proteus RISC-V core
- Experimental evaluation

HW/SW co-designs can be effective and efficient solutions against side-channel attacks

Many remaining challenges!

- **New defenses:** new processors optims, emerging applications, platforms, etc.
- **Compiler support:**
  - needed for adoption and better evaluation
  - parametric in leakage contract
- **Hardware verification:** support defenses and scale existing techniques
- **Comparison** of existing defenses on the same baseline

Ecosystem to implement, evaluate, and compare security defenses?