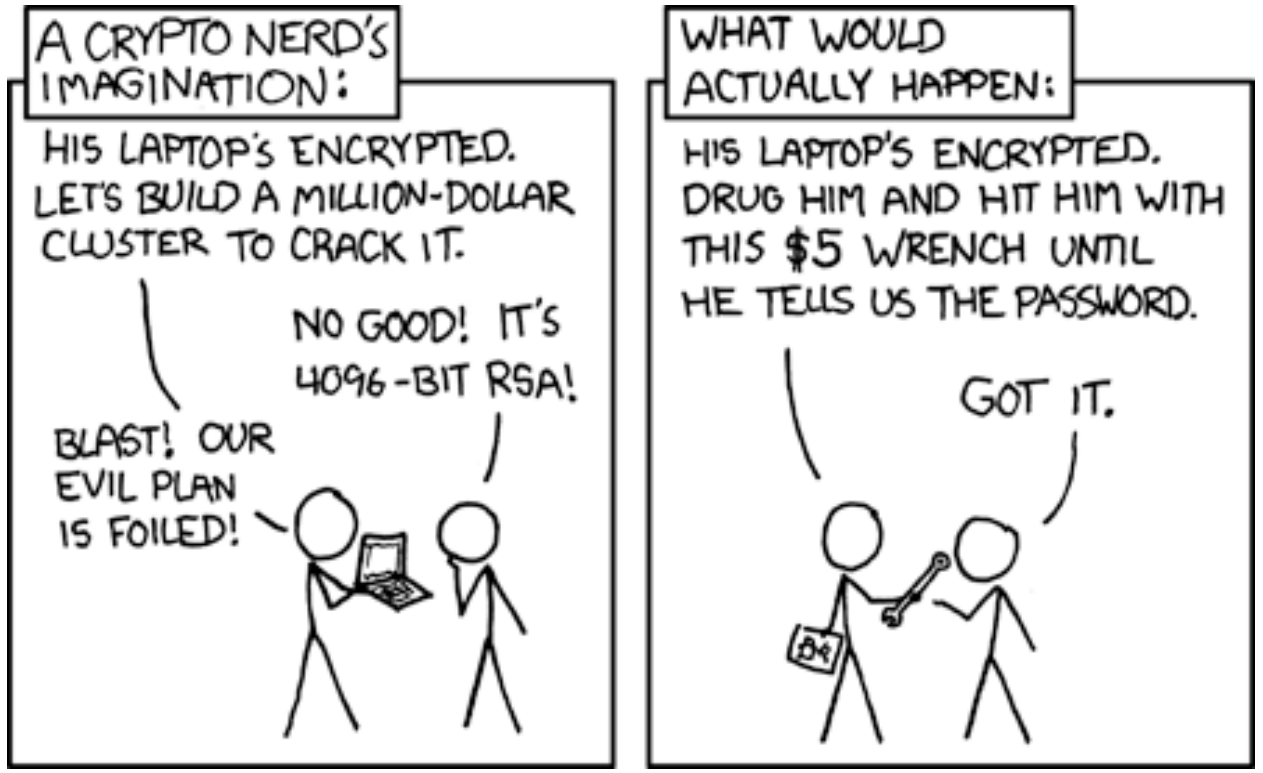
Hardware Security

Not That Hard, Not That Secure

A long-winded journey led by Rutvik Choudhary



⚠ *A system is only secure as it's weakest link!*

What does it mean to be “secure” anyways?

A lot of encryption relies on the “hardness” of certain math problems

Factorizing integers is hard

Computing logarithms is hard 

Finding the shortest non-zero vector in a lattice is hard 

Some encryption relies on the fact that you can’t do any better than brute force guessing 

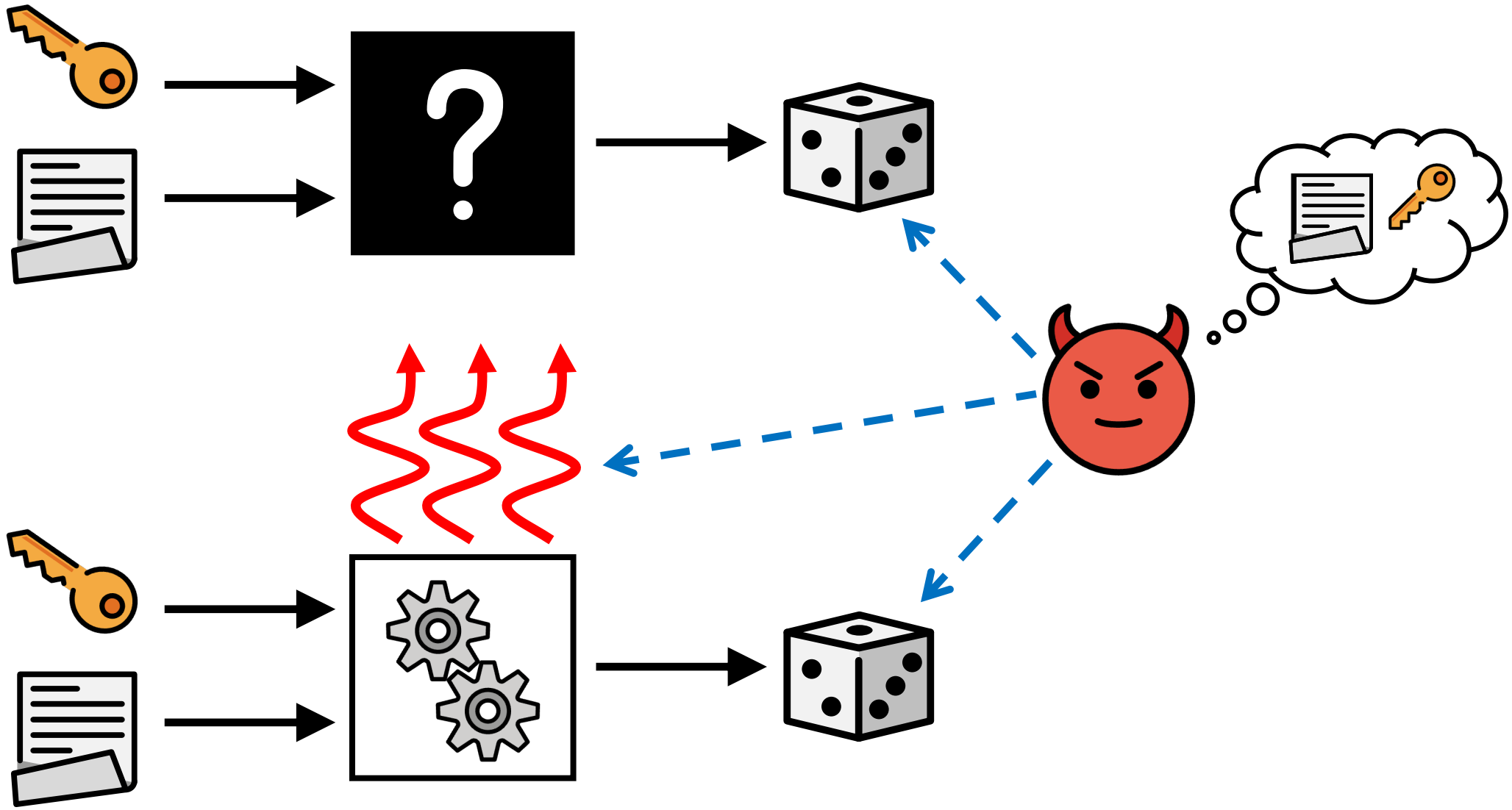
A lot of practical security mechanism try to just block you from doing anything you’re not supposed to be doing

What does it mean to be “secure” anyways?

But such guarantees only exist in the mind…

…the real world is not so ideal!

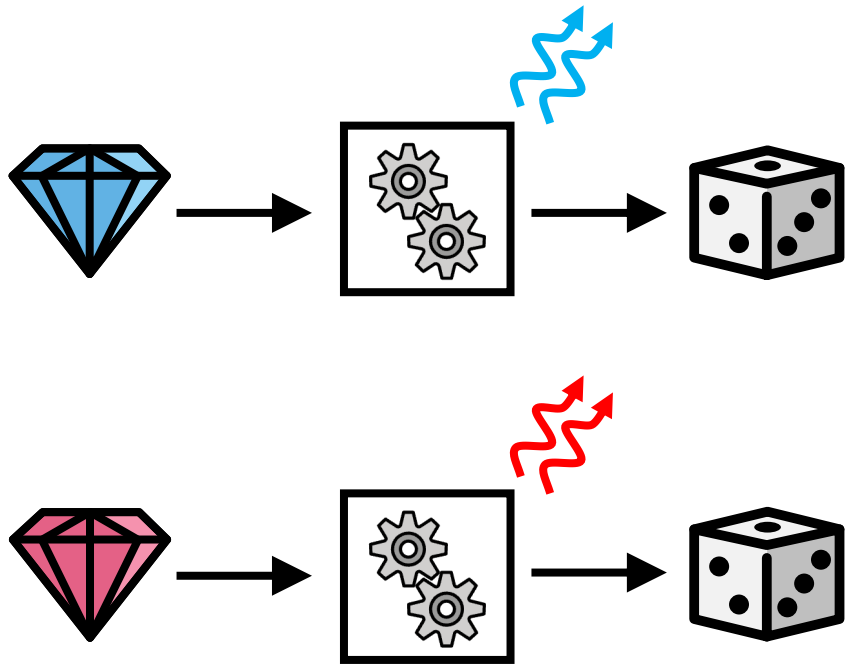
"Nobody's Perfect" — Hannah Montanna

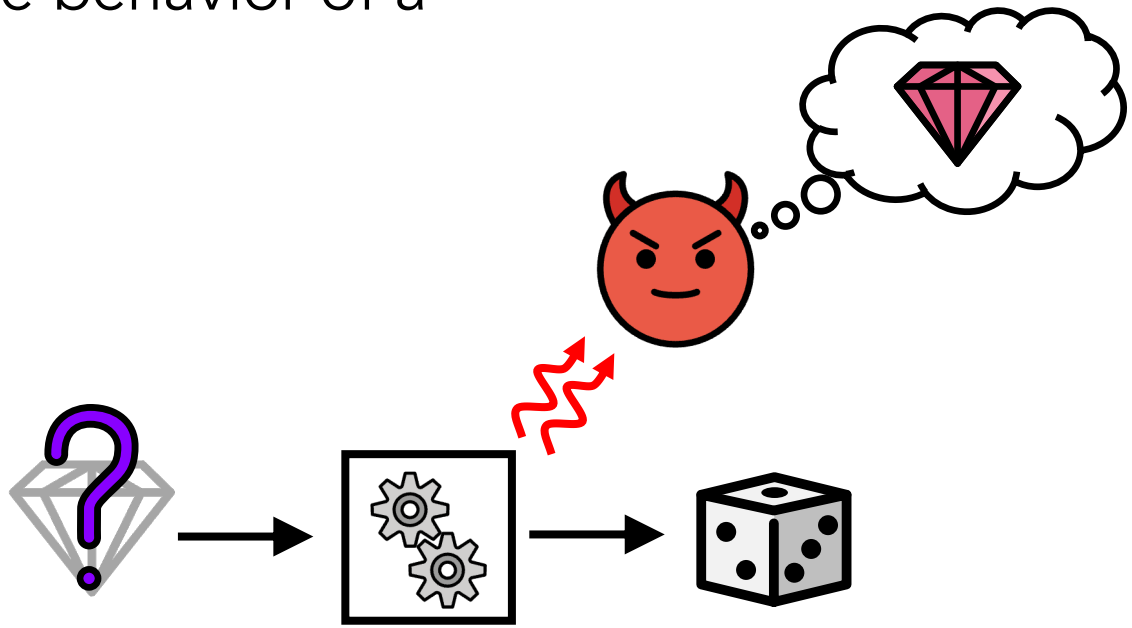


Secure systems' worst gossip: The SIDE CHANNEL

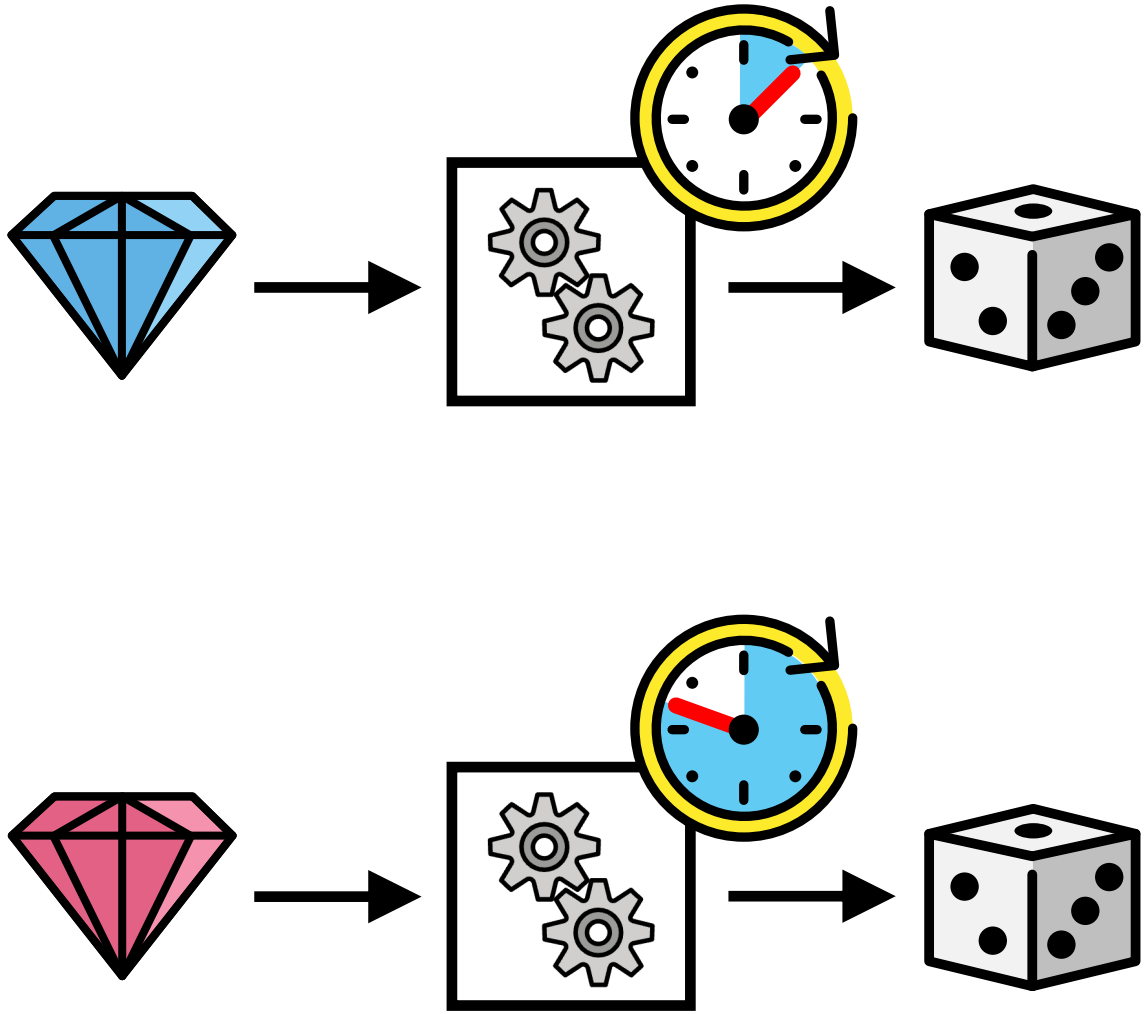
A side channel occurs when an adversary can learn something they theoretically shouldn't have because the implementation is broken

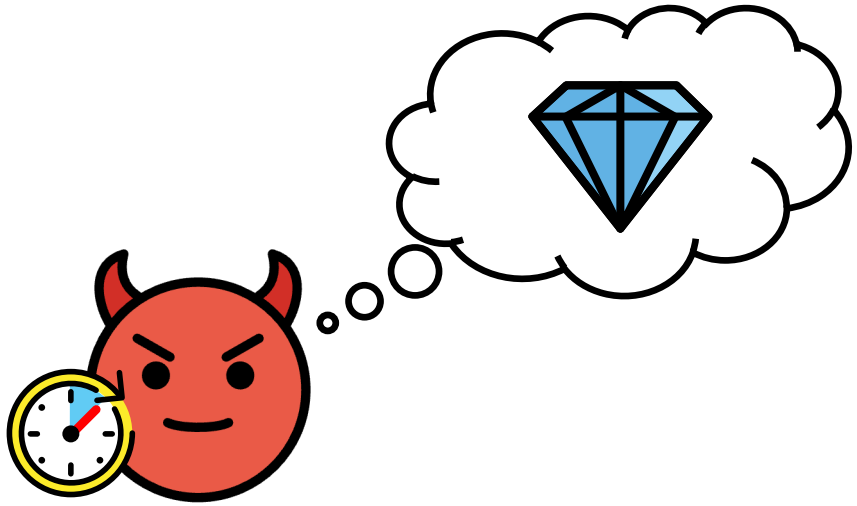
In a nutshell, this happens whenever the behavior of a program varies due to the input

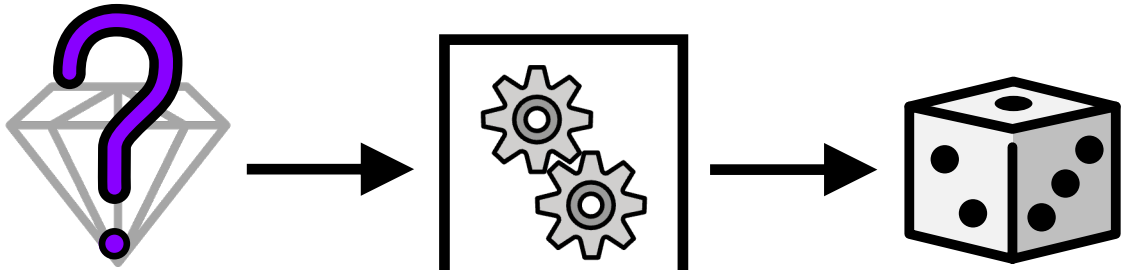




Timing is everything!





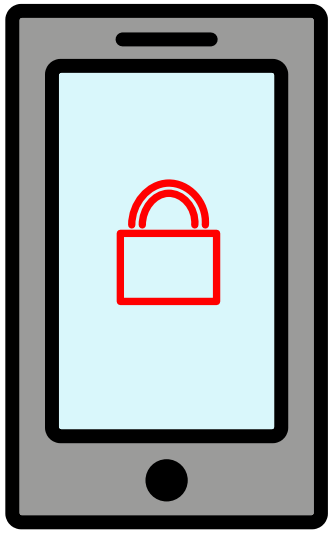


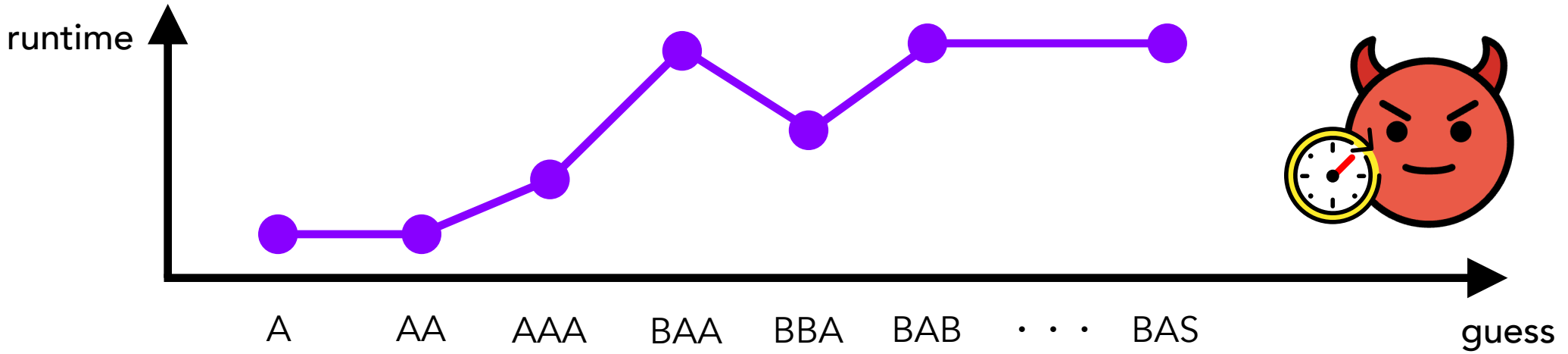
Simple Example: A *terrible* password checker

```
input_uname = input("username")
input_pwd = input("password")
real_pwd = DATABASE[input_uname]
if (len(input_pwd) != len(real_pwd)) {
  exit("ERROR: wrong password!")
}
for (i = 0; i < len(real_pwd); i++) {
  if (input_pwd[i] != real_pwd[i]) {
    exit("ERROR: wrong password!")
  }
}
```

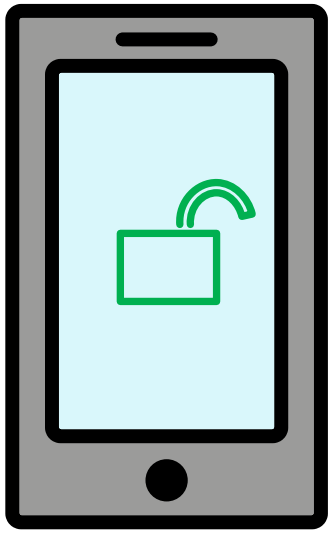
Early termination means the amount of work the program does is a function of how many characters match!

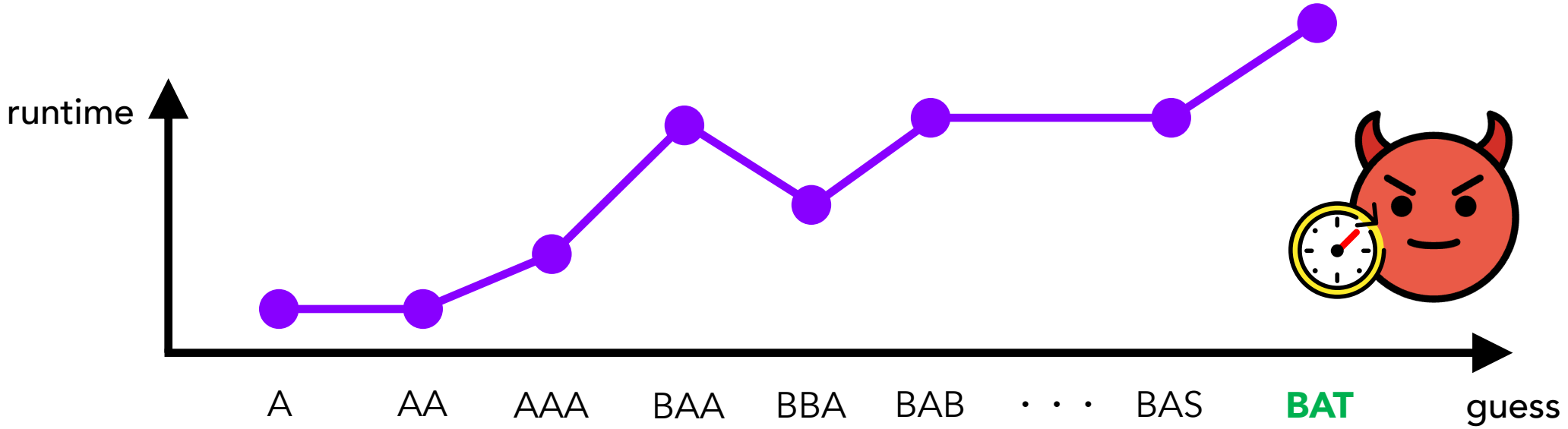
Breaking Bad (Code)





Breaking Bad (Code)





"Time" to pay attention!

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 t
form private key operations, attackers may be able to find fixed
Hellman exponents, factor RSA keys, and break other cryptosy
Against a vulnerable system, the attack is computationally inex
and often requires only known ciphertext. Actual systems are po

1996

Remote Timing Attacks are Practical

David Brumley
Stanford University
dbrumley@cs.stanford.edu

Dan Boneh
Stanford University
dabo@cs.stanford.edu

Abstract

Timing attacks are usually used to attack weak computing devices such as smartcards. We show that timing attacks apply to general software systems. Specifically, we devise a timing attack against OpenSSL. Our experiments show that we can extract private keys from an OpenSSL-based web server running on a machine in the local network. Our results demonstrate that timing attacks against network servers are practical and therefore security systems should defend against them.

The attacking machine and the server were in different buildings with three routers and multiple switches between them. With this setup we were able to extract the SSL private key from common SSL applications such as a web server (Apache+mod_SSL) and a SSL-tunnel.

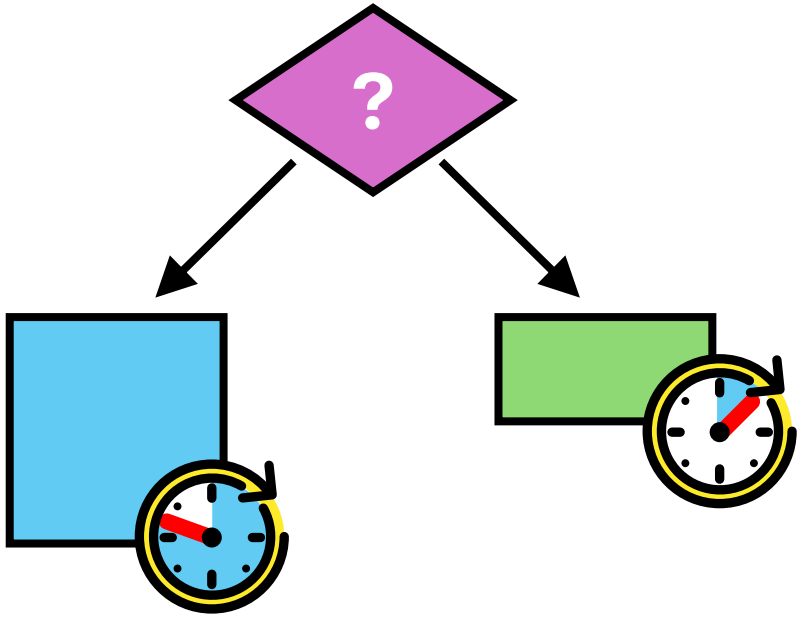
Interprocess. We successfully mounted the attack between two processes running on the same machine. A hosting center that hosts two domains on the same machine might give management access to the admins of each domain. Since both domain are hosted on the same machine, one admin could use

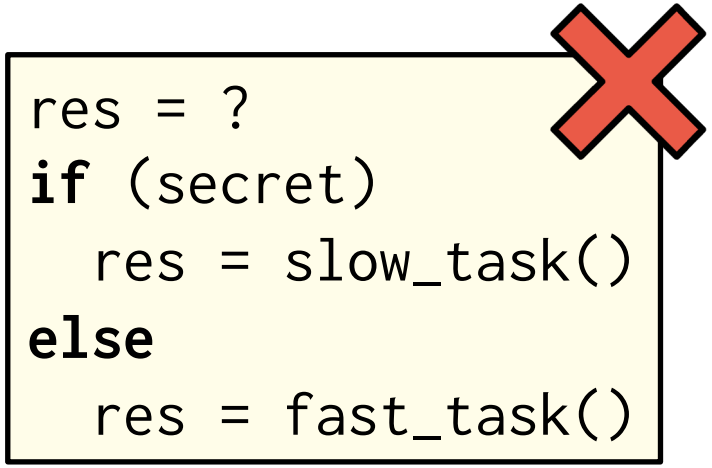
2003

Constant-time code to the resuce!

GOAL: Make the timing of the program *independent* of the inputs of the program

Just don't branch on secret values!

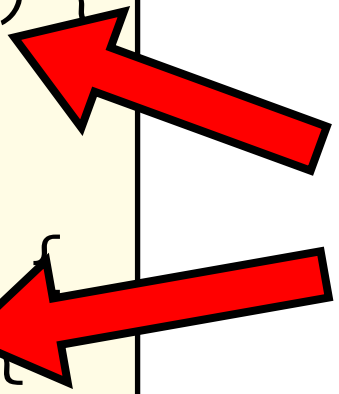




```
res = ?
if (secret)
  res = slow_task()
else
  res = fast_task()
```

RECALL: The *terrible* password checker

```
input_uname = input("username")
input_pwd = input("password")
real_pwd = DATABASE[input_uname]
if (len(input_pwd) != len(real_pwd)) {
  exit("ERROR: wrong password!")
}
for (i = 0; i < len(real_pwd); i++) {
  if (input_pwd[i] != real_pwd[i]) {
    exit("ERROR: wrong password!")
  }
}
```

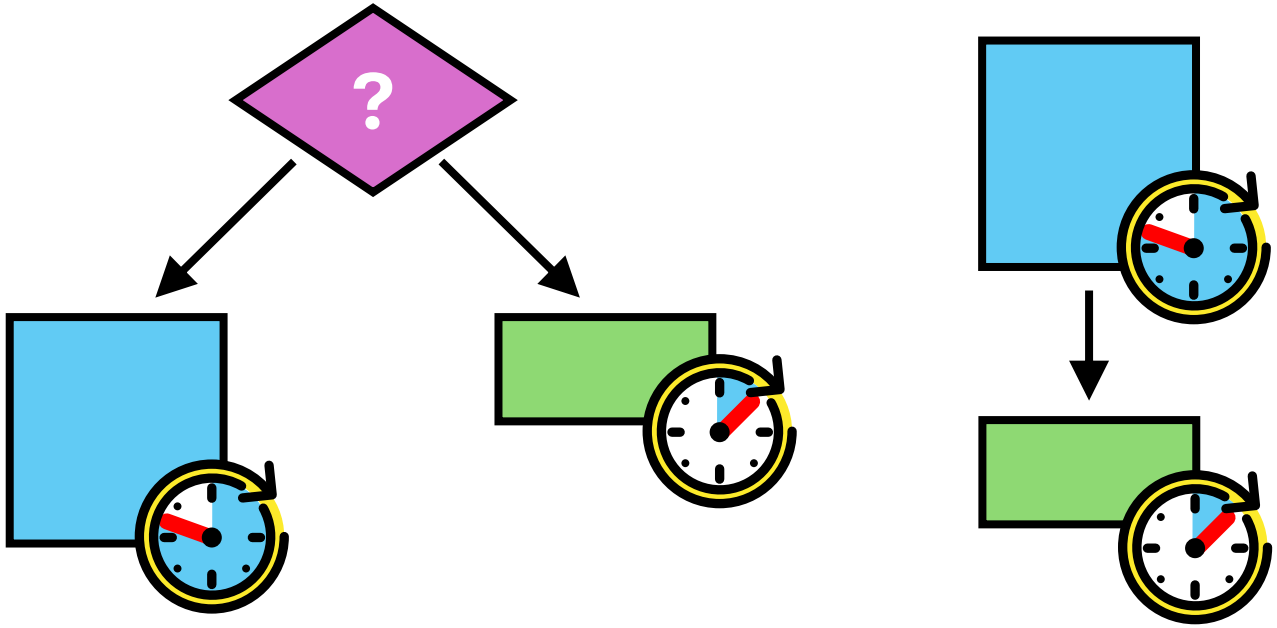


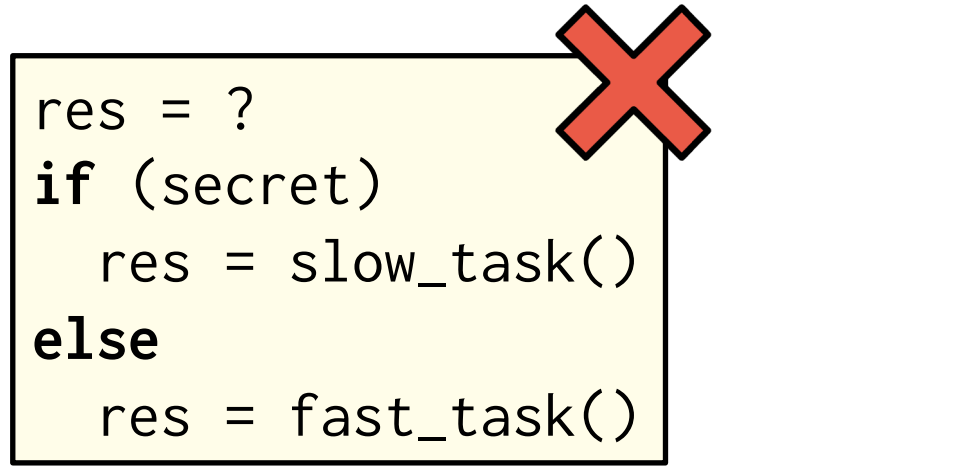
Secret-dependent branches!

Constant-time code to the resuce!

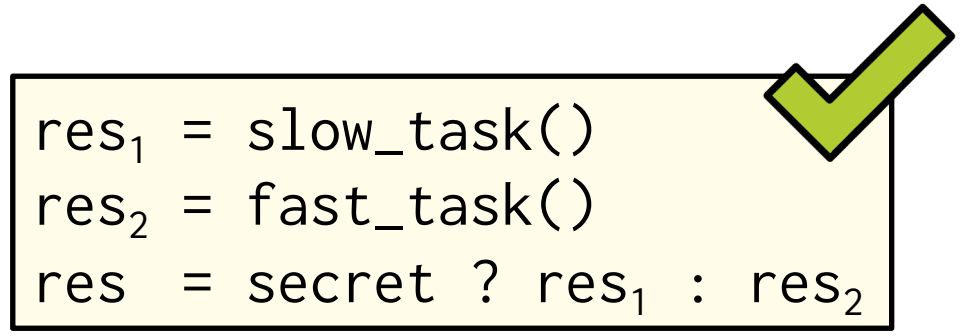
GOAL: Make the timing of the program *independent* of the inputs of the program

Just don't branch on secret values!





```
res = ?
if (secret)
  res = slow_task()
else
  res = fast_task()
```



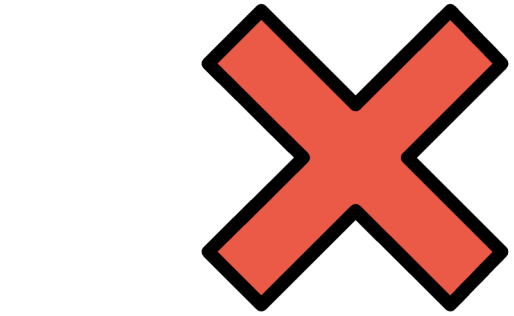
```
res1 = slow_task()
res2 = fast_task()
res  = secret ? res1 : res2
```

A *better* password checker!

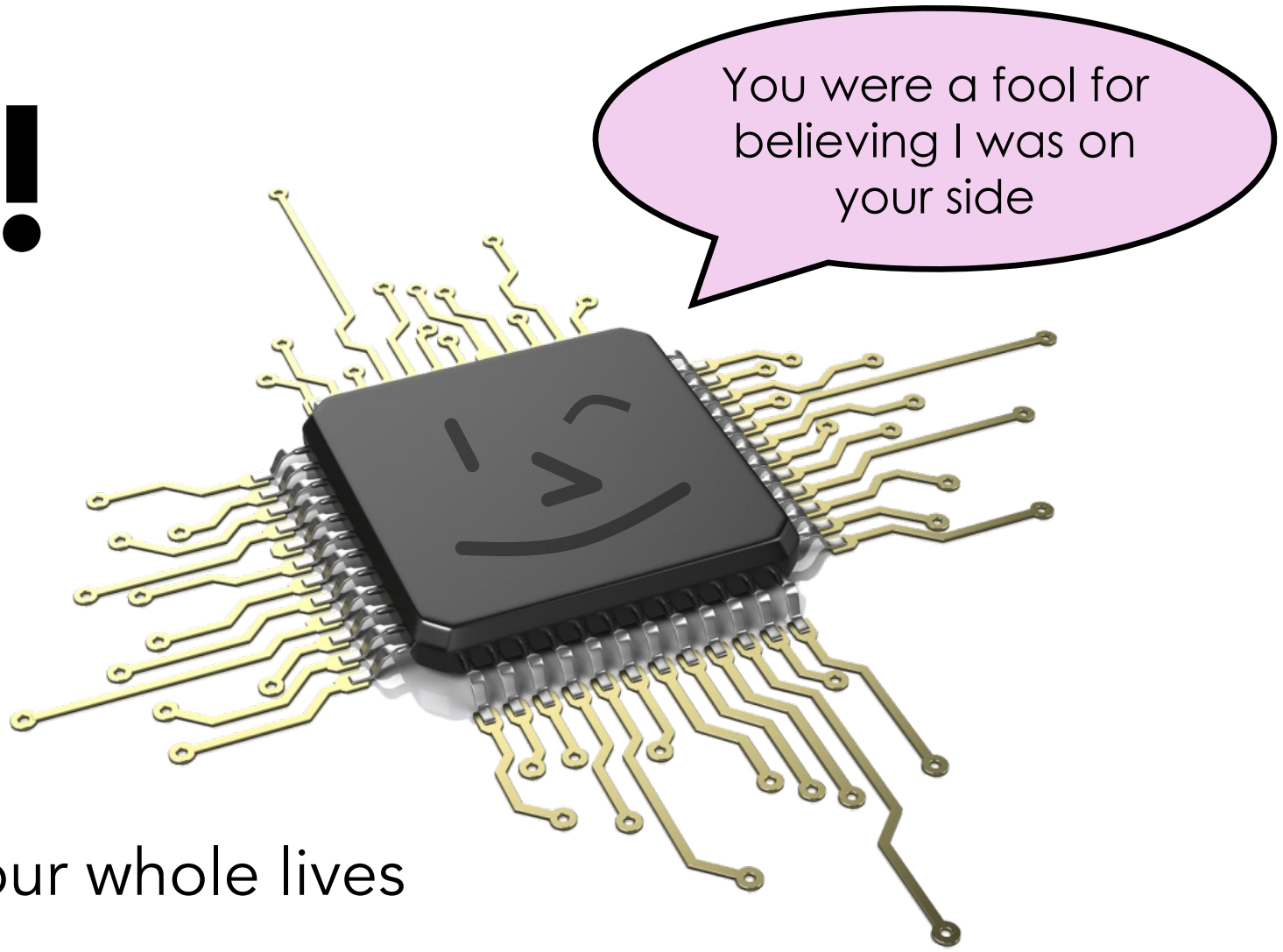
```
input_uname = input("username")
input_pwd = input("password")
real_pwd = DATABASE[input_uname]
result = (len(input_pwd) ^ len(real_pwd)) << 8
for (i = 0; i < min{len(input_pwd), len(real_pwd)}; i++) {
  result |= input_pwd[i] ^ real_pwd[i]
}
if (result != 0) {
  exit("ERROR: wrong password!")
}
```

Please do not actually check your passwords this way

And so we're done right…?



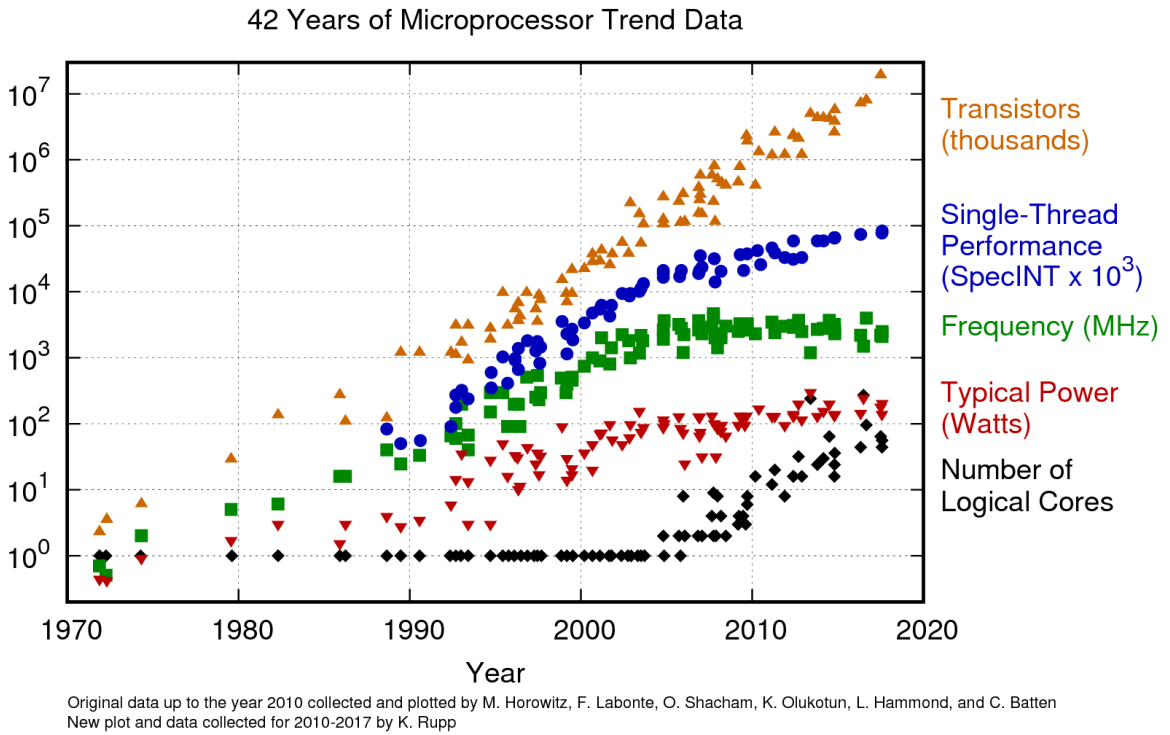
NO!



This loser is about to ruin our whole lives

You can’t trust anyone, not even your CPU…

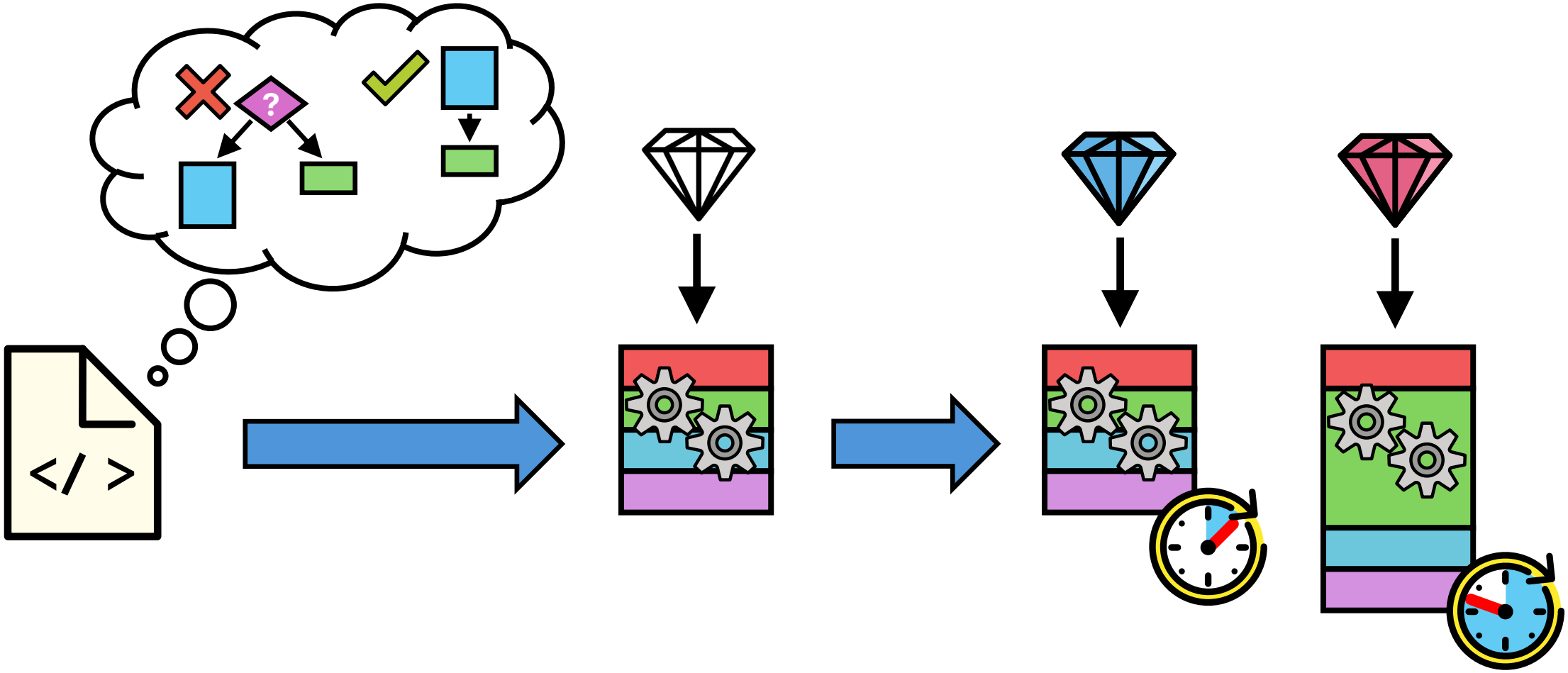
The reality is you can have code that *looks* side-channel free, but when run on actual hardware, you inevitably end up with data-dependent obsevations



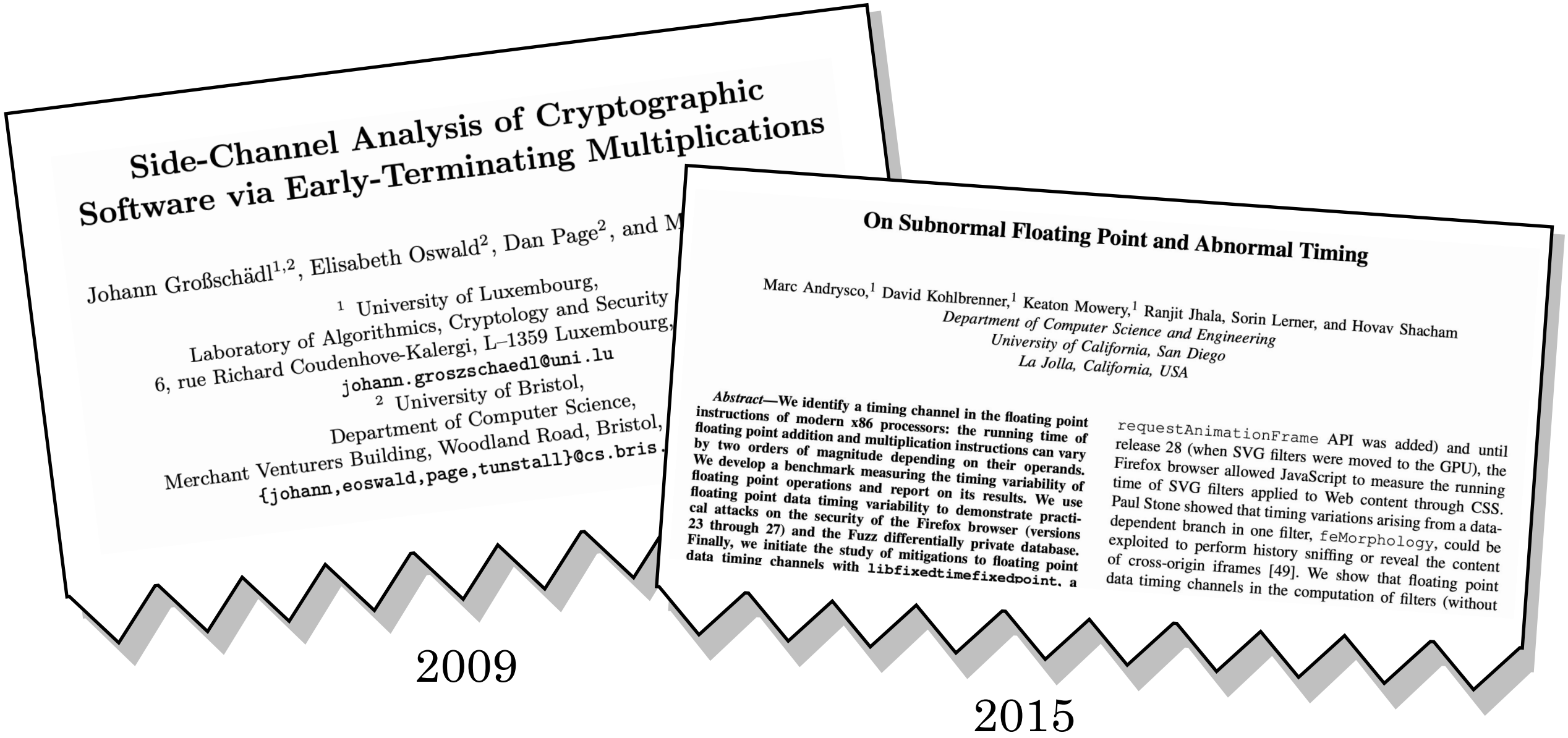
And it’s really our fault, we want computers to run faster and faster; they’re going to take shortcuts, and those shortcuts will end up data-dependent

How can CPUs break secure code?

1 Some instructions can take different times depending on their inputs



Leaky instructions (far and few in between)



Side-Channel Analysis of Cryptographic Software via Early-Terminating Multiplications

Johann Großschädl[1,2], Elisabeth Oswald[2], Dan Page[2], and M

[1] University of Luxembourg,
Laboratory of Algorithmics, Cryptology and Security
6, rue Richard Coudenhove-Kalergi, L–1359 Luxembourg,
johann.groszschaedl@uni.lu
[2] University of Bristol,
Department of Computer Science,
Merchant Venturers Building, Woodland Road, Bristol,
{johann,eoswald,page,tunstall}@cs.bris.

2009

On Subnormal Floating Point and Abnormal Timing

Marc Andrysco,[1] David Kohlbrenner,[1] Keaton Mowery,[1] Ranjit Jhala, Sorin Lerner, and Hovav Shacham
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, California, USA

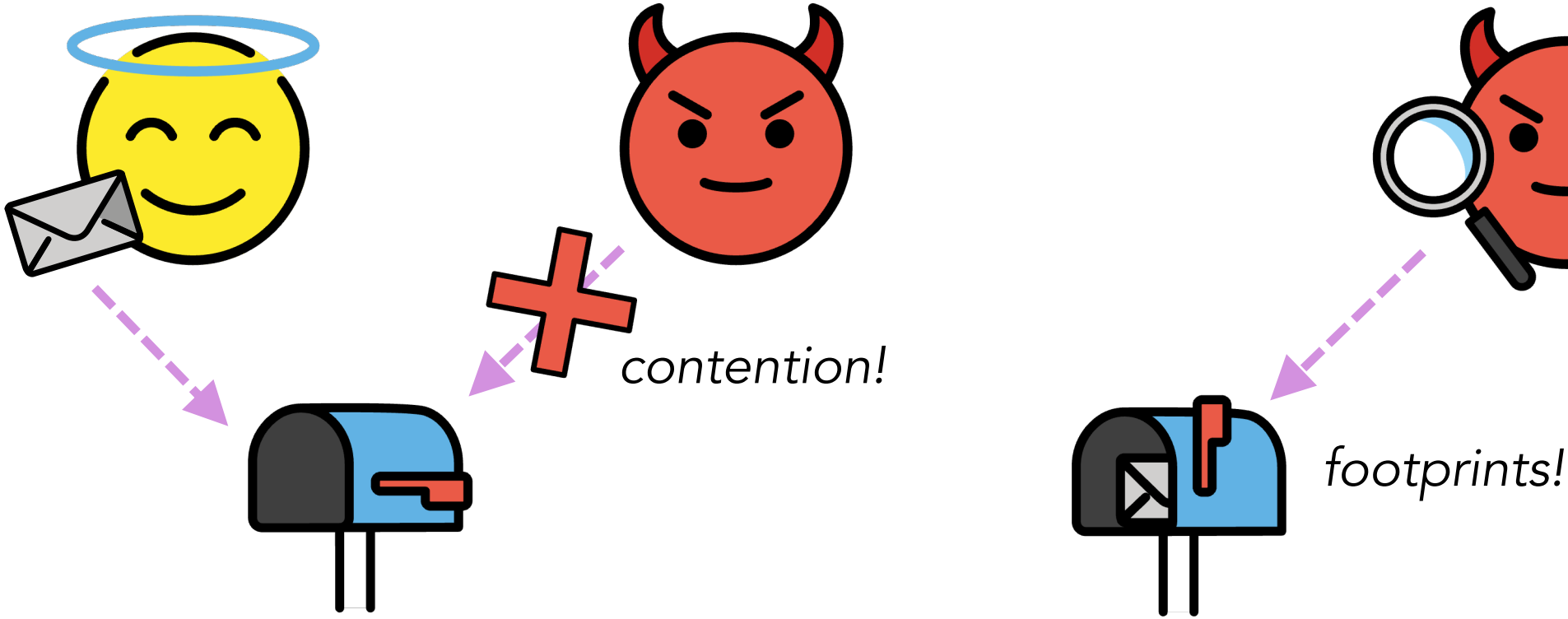
***Abstract*—We identify a timing channel in the floating point instructions of modern x86 processors: the running time of floating point addition and multiplication instructions can vary by two orders of magnitude depending on their operands. We develop a benchmark measuring the timing variability of floating point operations and report on its results. We use floating point data timing variability to demonstrate practical attacks on the security of the Firefox browser (versions 23 through 27) and the Fuzz differentially private database. Finally, we initiate the study of mitigations to floating point data timing channels with `libfixedtimefixedpoint`, a**

`requestAnimationFrame` API was added) and until release 28 (when SVG filters were moved to the GPU), the Firefox browser allowed JavaScript to measure the running time of SVG filters applied to Web content through CSS. Paul Stone showed that timing variations arising from a data-dependent branch in one filter, `feMorphology`, could be exploited to perform history sniffing or reveal the content of cross-origin iframes [49]. We show that floating point data timing channels in the computation of filters (without

2015

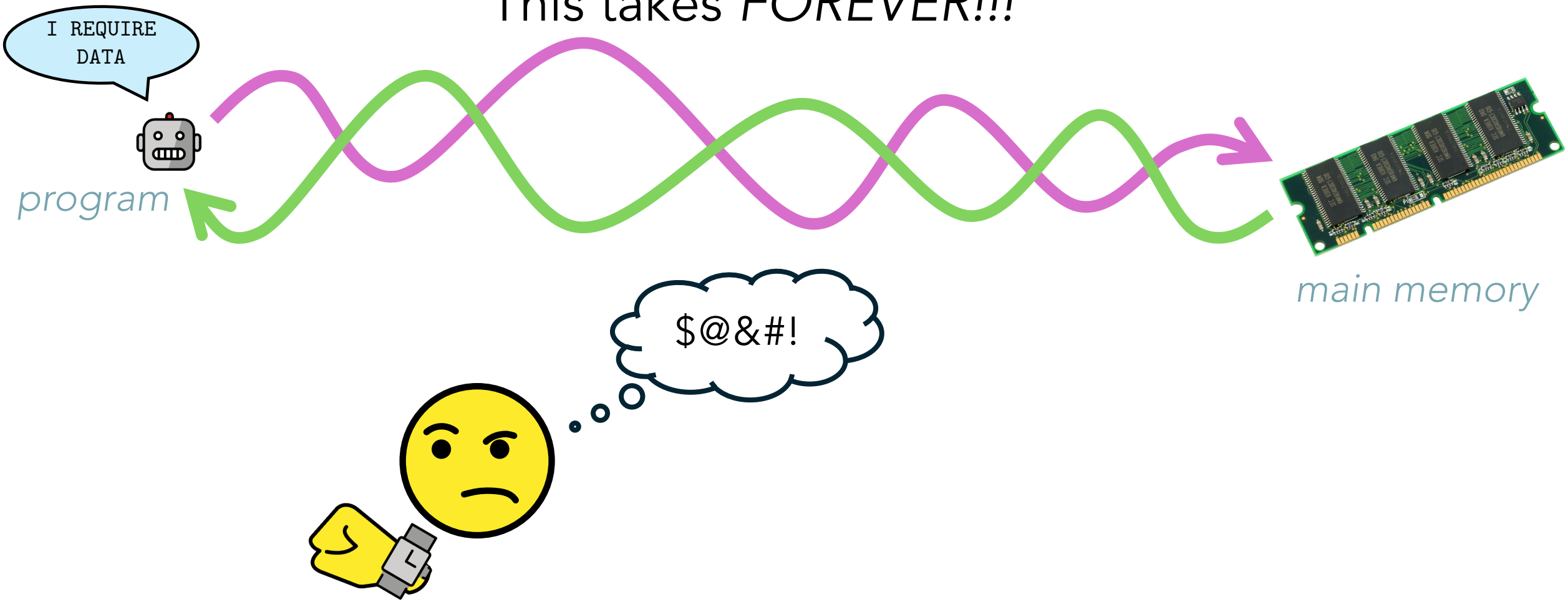
How can CPUs break secure code?

2 SHARED RESOURCES allow an attacker to snoop on the activity of a victim



Caches: Paving the road to hell with good intentions

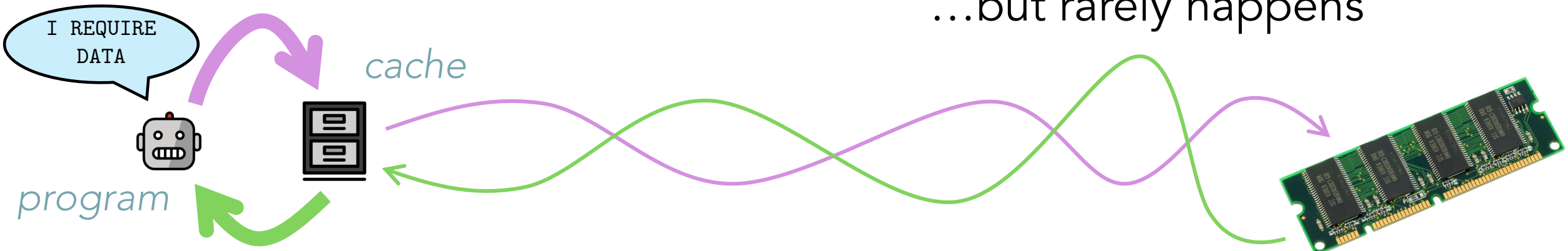
This takes *FOREVER!!!*



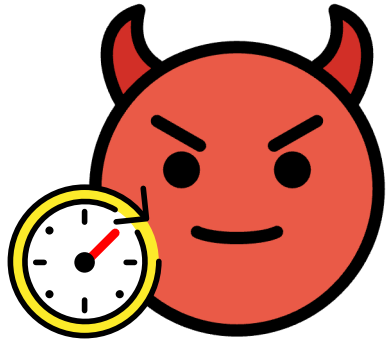
Caches: Paving the road to hell with good intentions

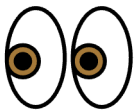
This still takes *FOREVER*…

…but rarely happens



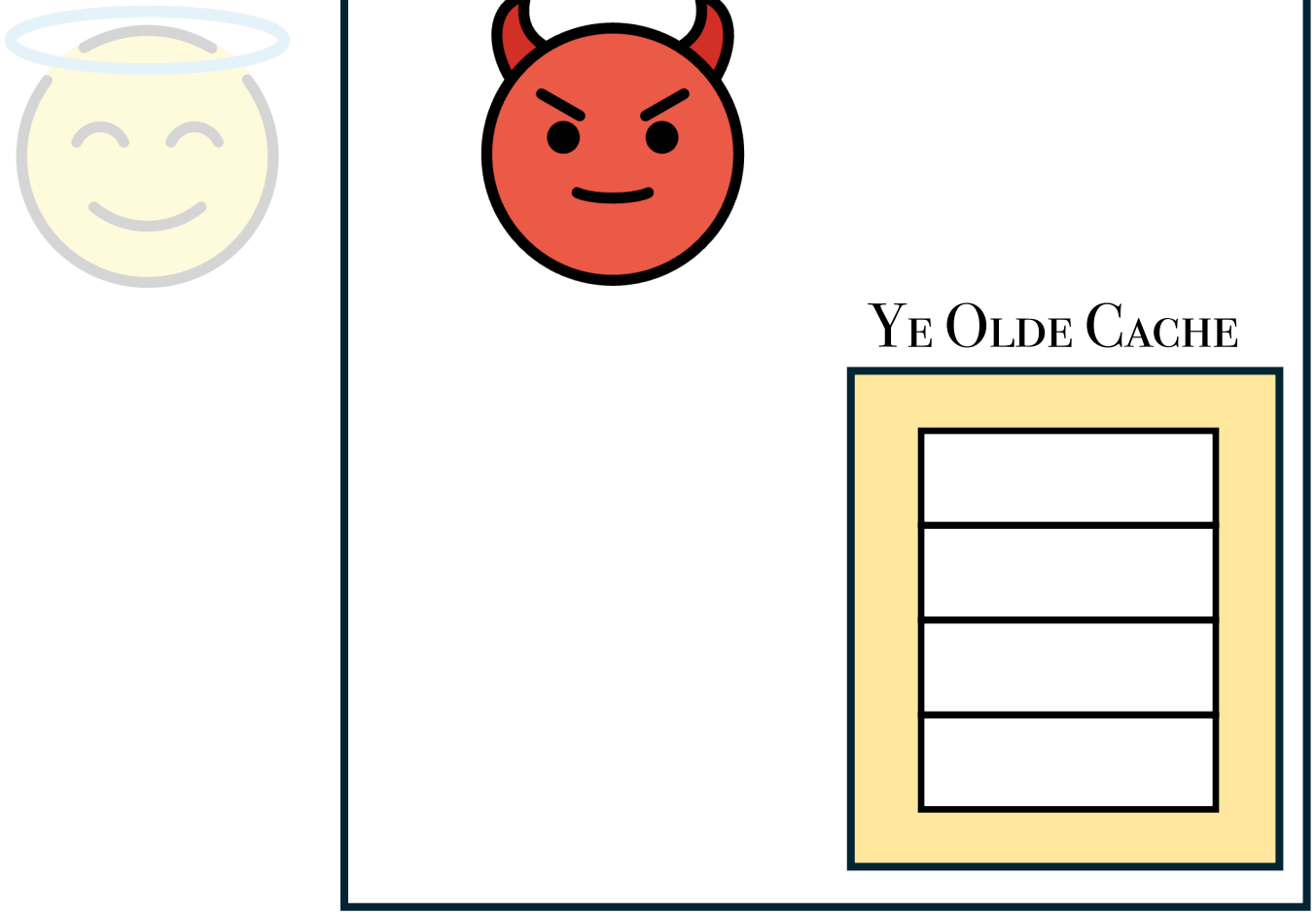
Super quick!



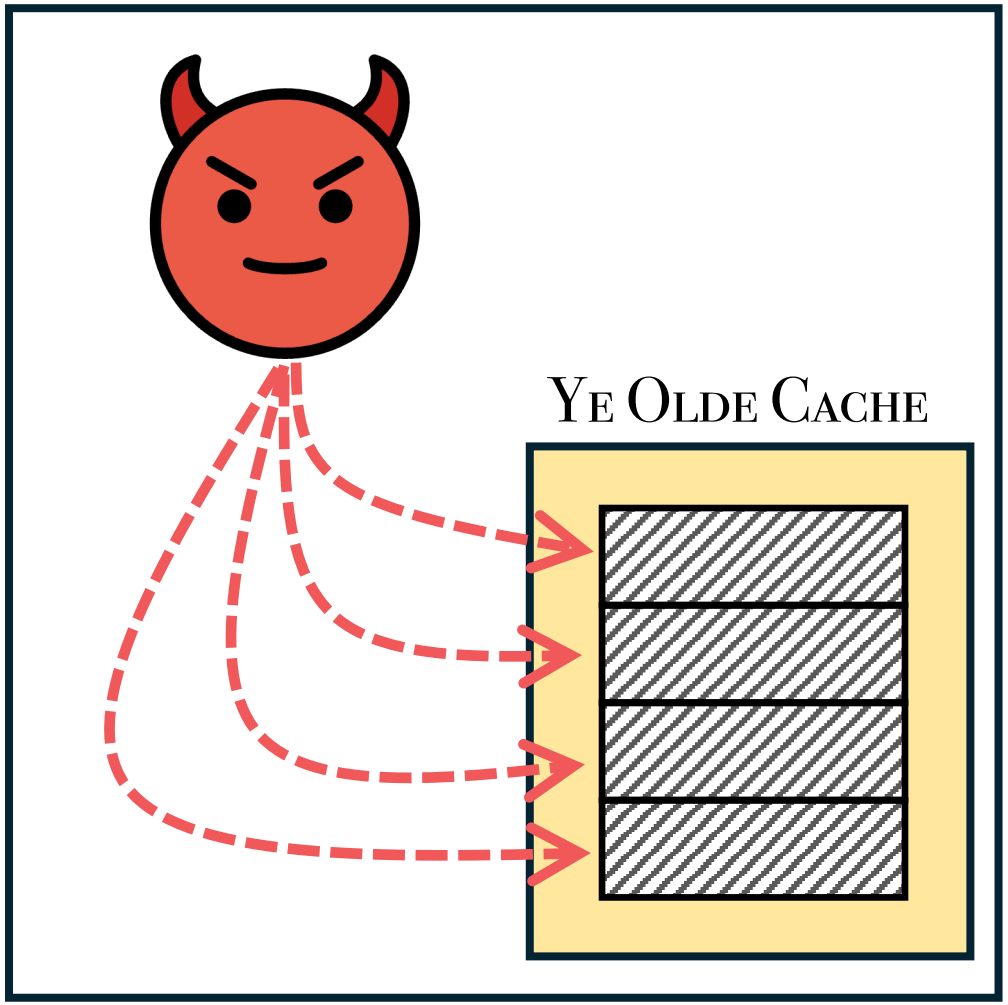


See the problem??

Cache Attacks 101: Prime + Probe

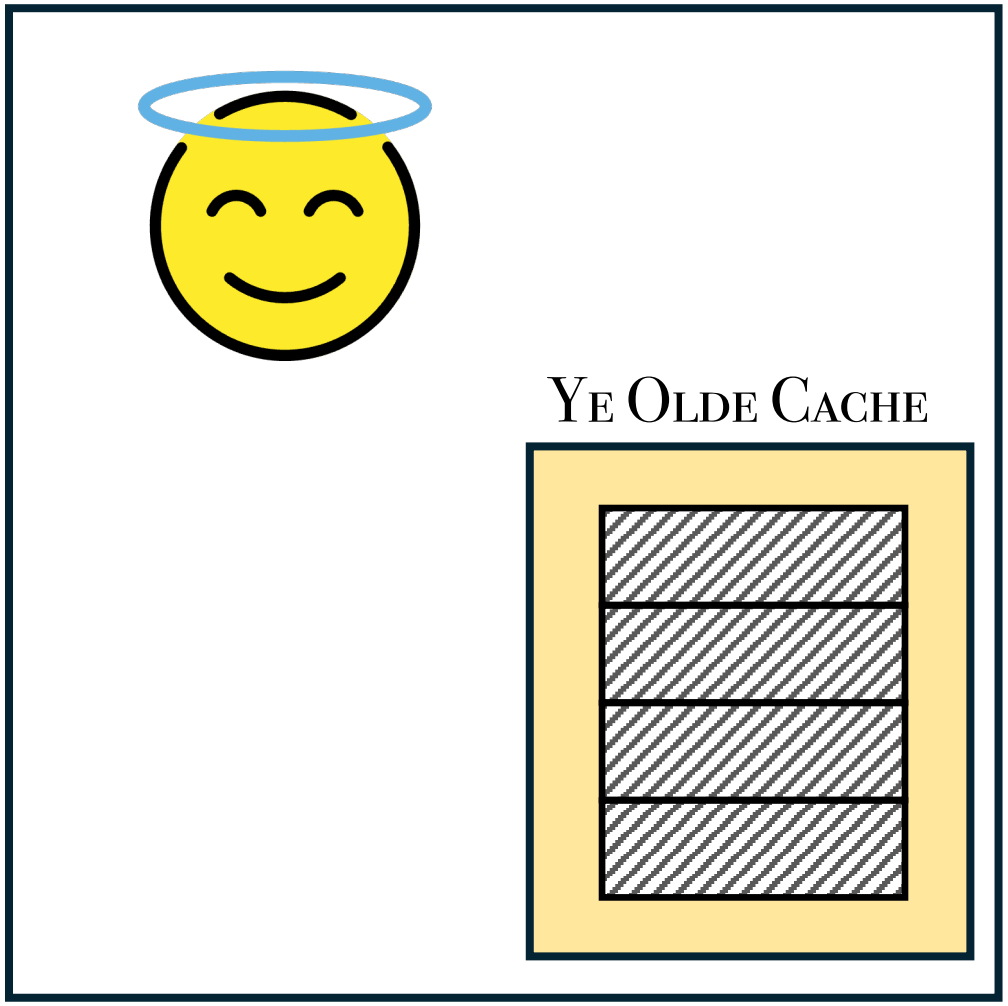


Cache Attacks 101: Prime + Probe



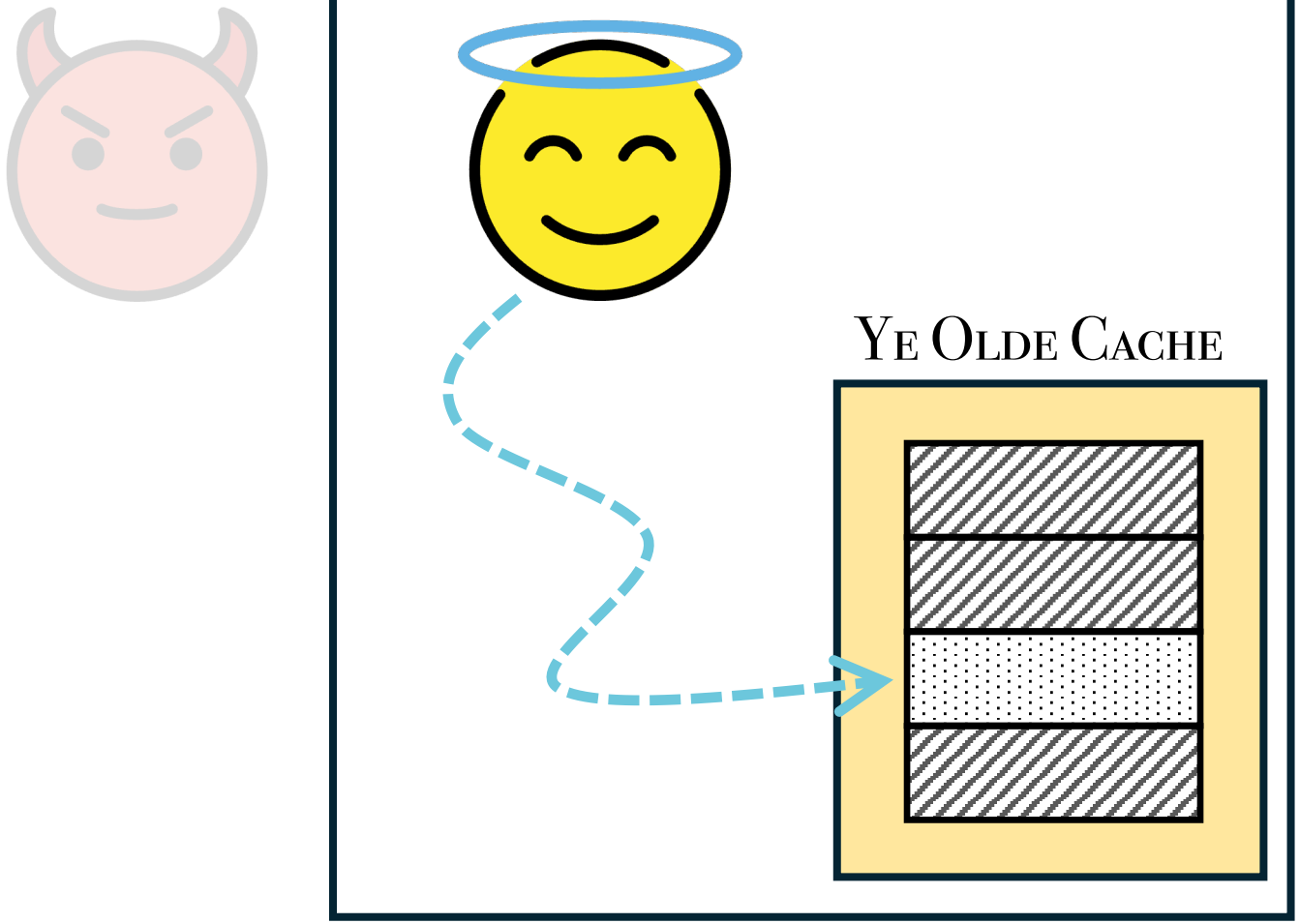
1. The attacker *primes* the cache by filling it up

Cache Attacks 101: Prime + Probe



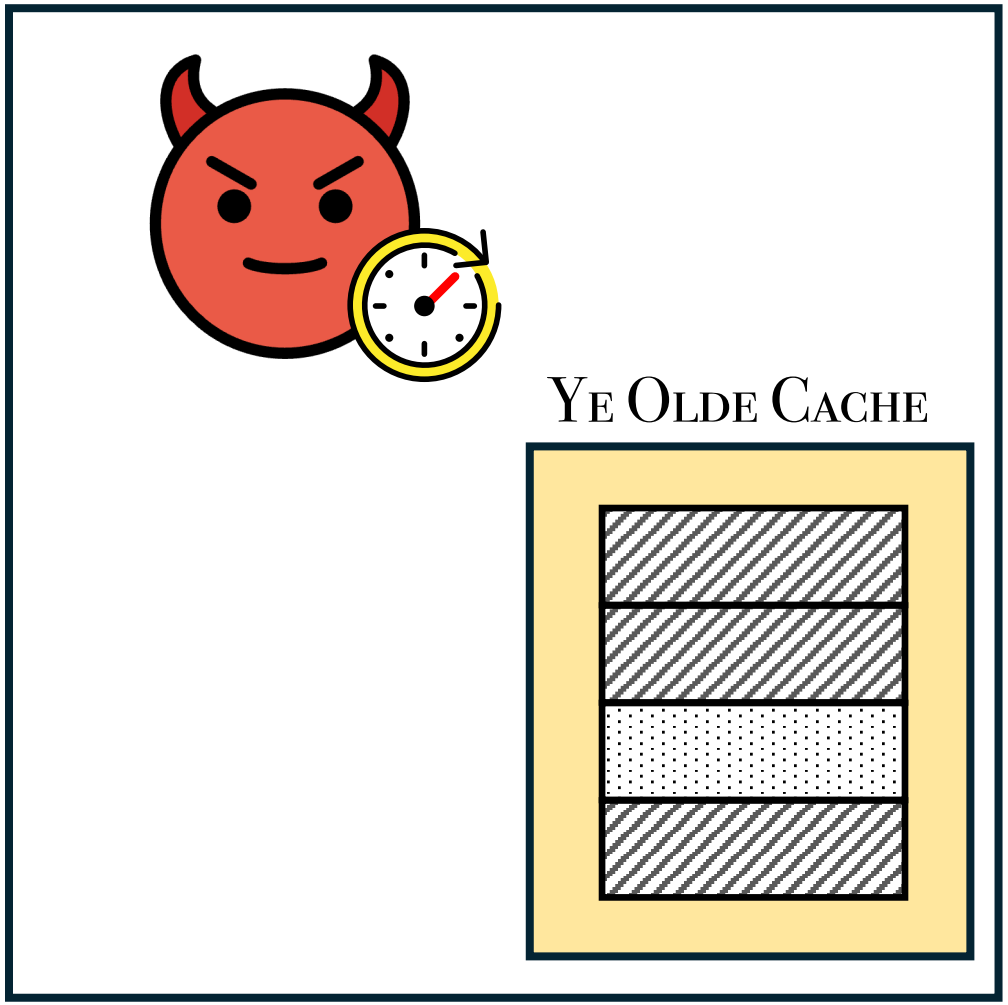
1. The attacker *primes* the cache by filling it up
2. **Context switch!** The victim comes in (the poor sap)

Cache Attacks 101: Prime + Probe



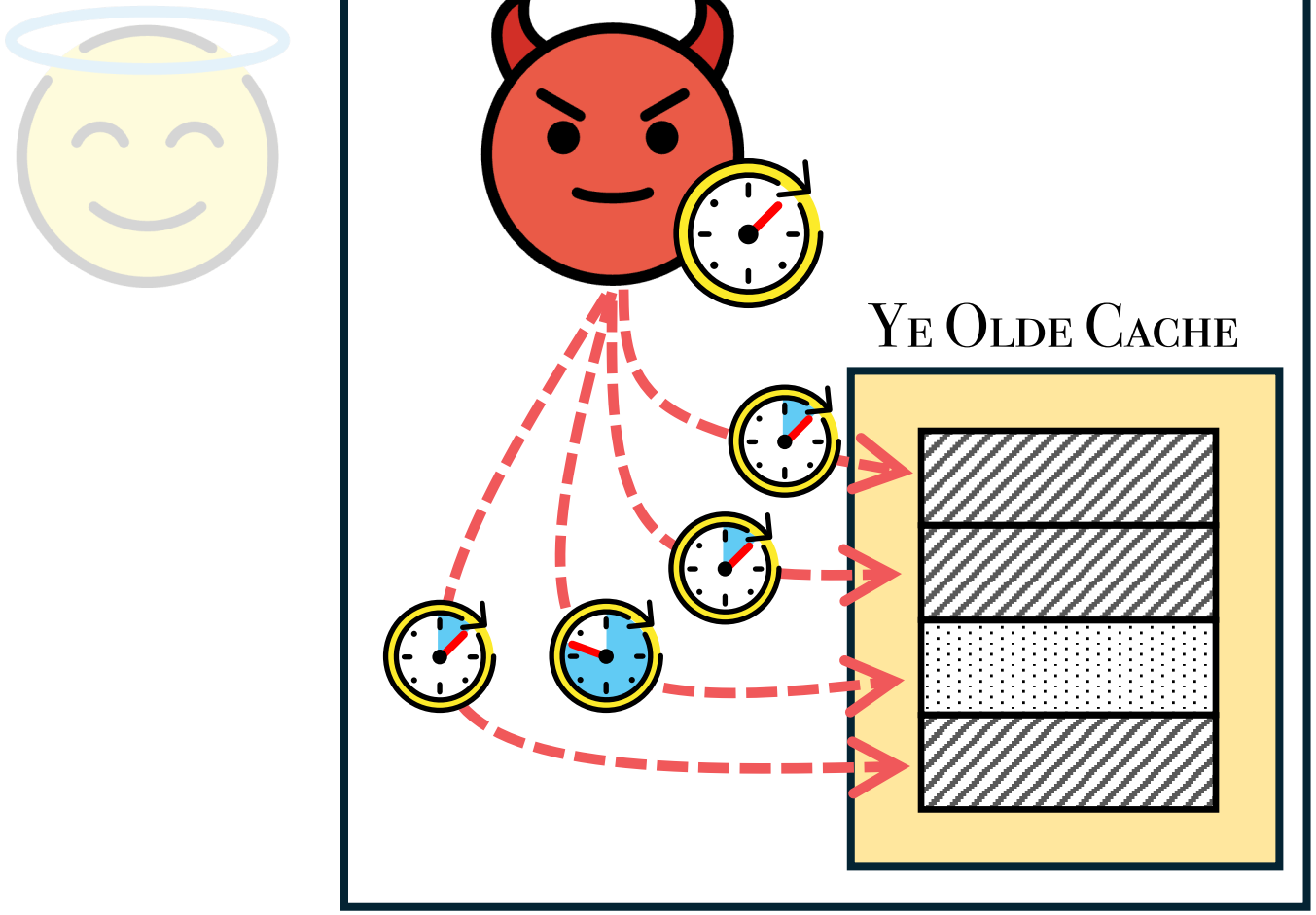
1. The attacker *primes* the cache by filling it up
2. **Context switch!** The victim comes in (the poor sap)
3. The victim accesses some cache line

Cache Attacks 101: Prime + Probe



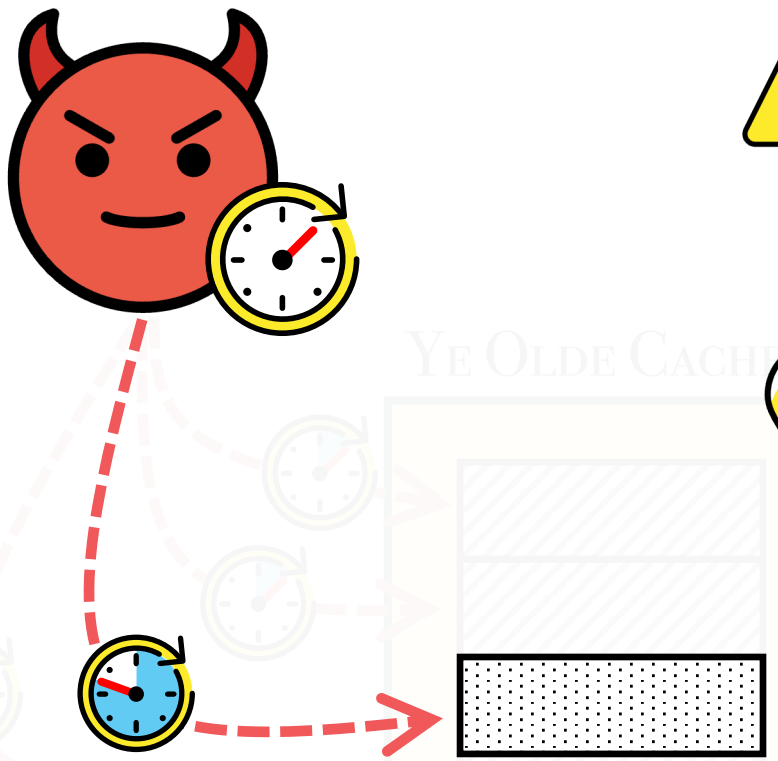
1. The attacker *primes* the cache by filling it up
2. **Context switch!** The victim comes in (the poor sap)
3. The victim accesses some cache line
4. **Context switch!** The attacker is back, but with a timer now

Cache Attacks 101: Prime + Probe



1. The attacker *primes* the cache by filling it up
2. **Context switch!** The victim comes in (the poor sap)
3. The victim accesses some cache line
4. **Context switch!** The attacker is back, but with a timer now
5. The attacker *probes* the cache by accessing each line

Cache Attacks 101: Prime + Probe



Note that the attacker can't see the victim's data inside the cache line, only the *location* of this line

However, if the location of this cache line depends on secret data, we can learn that secret data!

They call it a “cache” because security “pays”…

Cache Attacks and Countermeasures: the Case of AES

2005-08-14

Dag Arne Osvik[1], Adi Shamir[2] and Eran Tromer[2]

[1] dag.arne@osvik.no

[2] Department of Computer Science and Applied Mathemat
Weizmann Institute of Science, Rehovot 76100, Israel
{adi.shamir, eran.tromer}@weizmann.ac.il

Abstract. We describe several software side-channel attacks based on inter
the state of the CPU's memory cache. This leakage reveals memory access pa
for cryptanalysis of cryptographic primitives that employ data-dependent t
allow an unprivileged process to attack other processes running in paral
despite partitioning methods such as memory protection, sandboxing a
our methods require only the ability to trigger services that perform en
unknown key, such as encrypted disk partitions or secure network links

2005

Cache Games – Bringing Access-Based Cache Attacks on AES to Practice

Endre Bangerter
Bern University of Applied Sciences
endre.bangerter@bfh.ch

David Gullasch
Bern University of Applied Sciences, Dreamlab Technologies
david.gullasch@bfh.ch

Stephan Krenn
Bern University of Applied Sciences, University of Fribourg
stephan.krenn@bfh.ch

***Abstract*—Side channel attacks on cryptographic systems are attacks exploiting information gained from physical implementations rather than utilizing theoretical weaknesses of a scheme. In particular, during the last years, major achievements were made for the class of access-driven cache-attacks. The source of information leakage for such attacks are the locations of memory accesses performed by a victim process.**

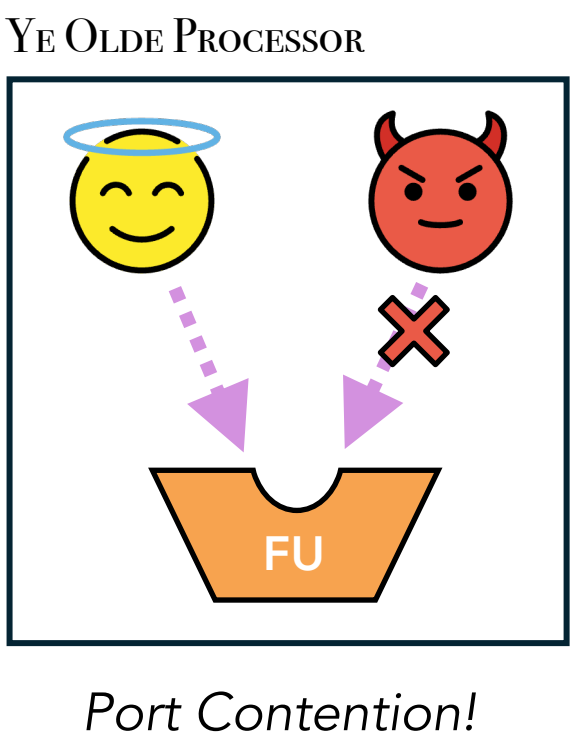
In this paper we analyze the case of AES and present an attack which is capable of recovering the full secret key in almost realtime for AES-128, requiring only a very limited number of observed encryptions. Unlike most other attacks, ours neither needs to know the ciphertext, nor does it need to know any information about the plaintext (such as its distribution, etc.).

attacks based on cache access mechanisms of microprocessors represented a vivid area of research in the last few years, e.g., [1]–[15]. These cache based side-channel attacks (or *cache attacks* for short) split into three types: *time-driven*, *trace-driven*, and *access-driven* attacks.

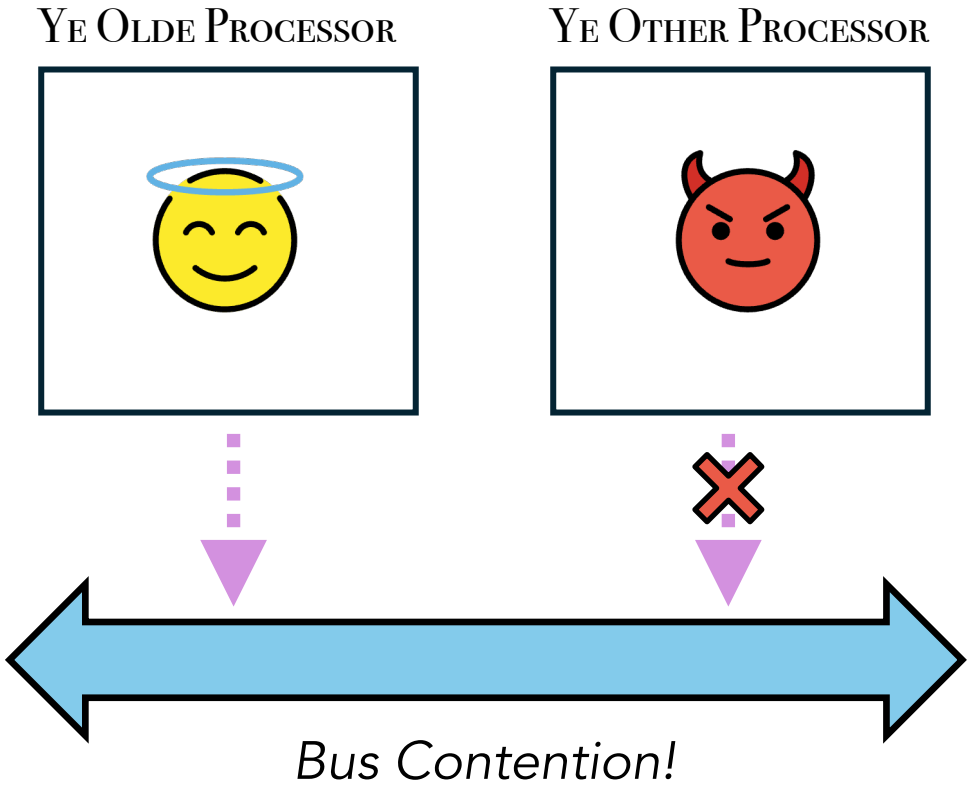
In time-driven attacks an adversary is able to observe the overall time needed to perform certain computations, such as whole encryptions [8]–[11]. From these timings he can make inferences about the overall number of cache hits and misses during an encryption. On the other hand, in trace-driven attacks, an adversary is able to obtain a profile of the cache

2011

Sharing is NOT caring! Contention based side channels



Port Contention!



Bus Contention!

Can we do something about it?

```
if (secret) {
  // do work
}
```

Thou shalt not branch on secret values

```
x = array[secret]
```

Thou shalt not access memory using secret values

```
y = 1.f / secret
```

Thou shalt not pass secret values to data-dependent timing ops

And we never had problems again!

Sidenote: Sometimes your friends are your opps…

"Alright whatever the general principle still holds…"

Breaking Bad: How Compilers Break Constant-Time Implementations

Moritz Schneider
ETH Zurich

Daniele Lain
ETH Zurich

Ivan Puddu
ETH Zurich

Nicolas Dutly
ETH Zurich

Srdjan Čapkun
ETH Zurich

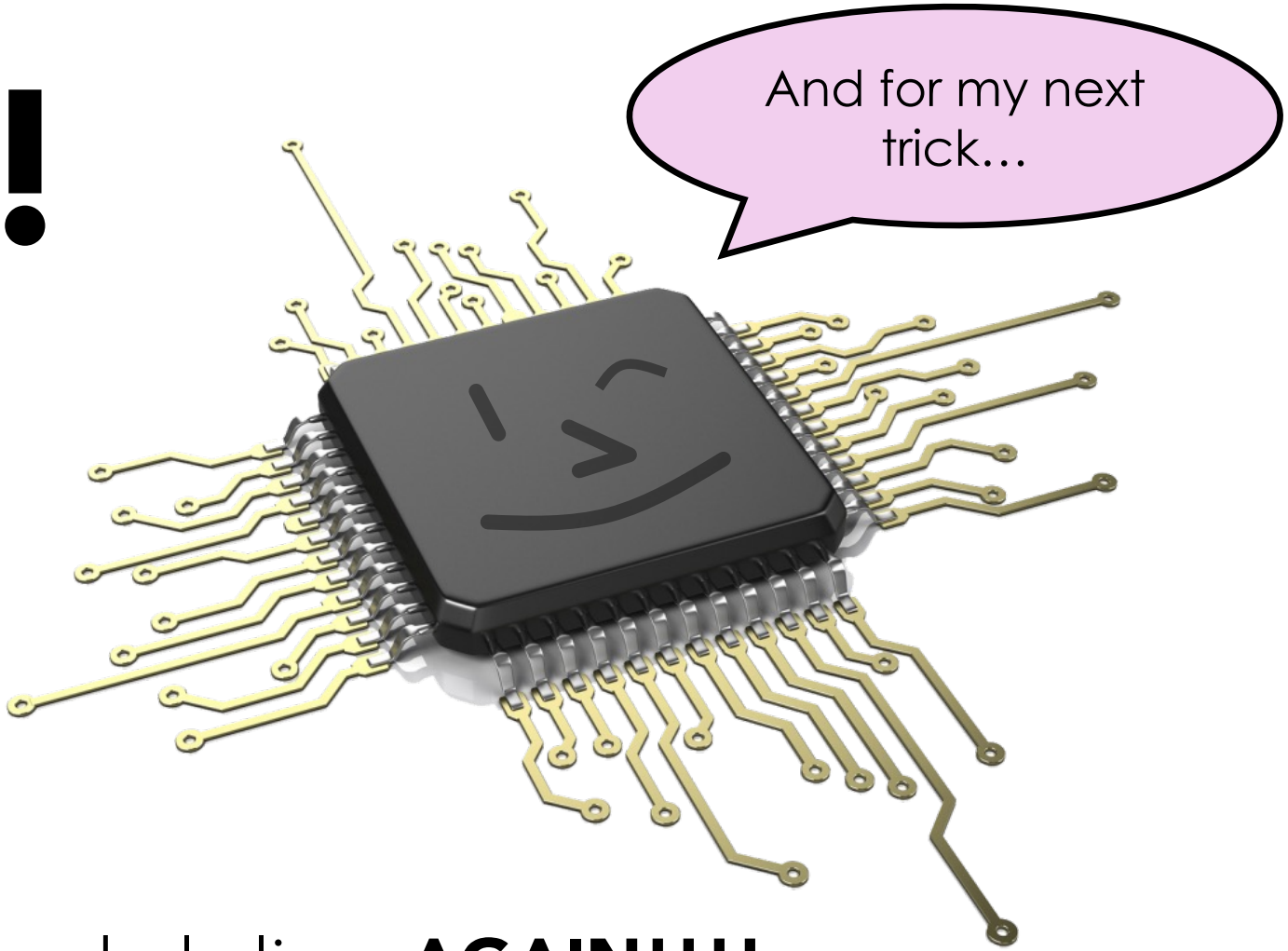
***Abstract*—The implementations of most hardened cryptographic libraries use defensive programming techniques for side-channel resistance. These techniques are usually specified as guidelines to developers on specific code patterns to use or avoid. Examples include performing arithmetic operations to choose between two variables instead of executing a secret-dependent branch. However, such techniques are only meaningful if they persist across compilation. In this paper, we investigate how optimizations used by modern compilers break the protections introduced by defensive programming techniques. Specifically, how compilers break high-level constant-time implementations used to mitigate timing side-channel attacks. We run a large-scale experiment to see if such compiler-induced issues manifest in state-of-the-art cryptographic libraries. We develop a tool that can**

be deployed everywhere, leaving less vetted architectures as second-class citizens in terms of security and hence potentially more susceptible to attacks. A solution to this problem is to compile the portable source code of security-critical libraries with special compilers that automatically remove side channels [12], [38]. However, these compilers suffer from a set of shortcomings: support for processor architectures is poor, they might require expert knowledge (e.g., to annotate the code), and they struggle to provide support for modern features of the processor or the employed source code languages. As a consequence, this approach is rarely used in practice, e.g., the binaries of security-critical libraries

2024

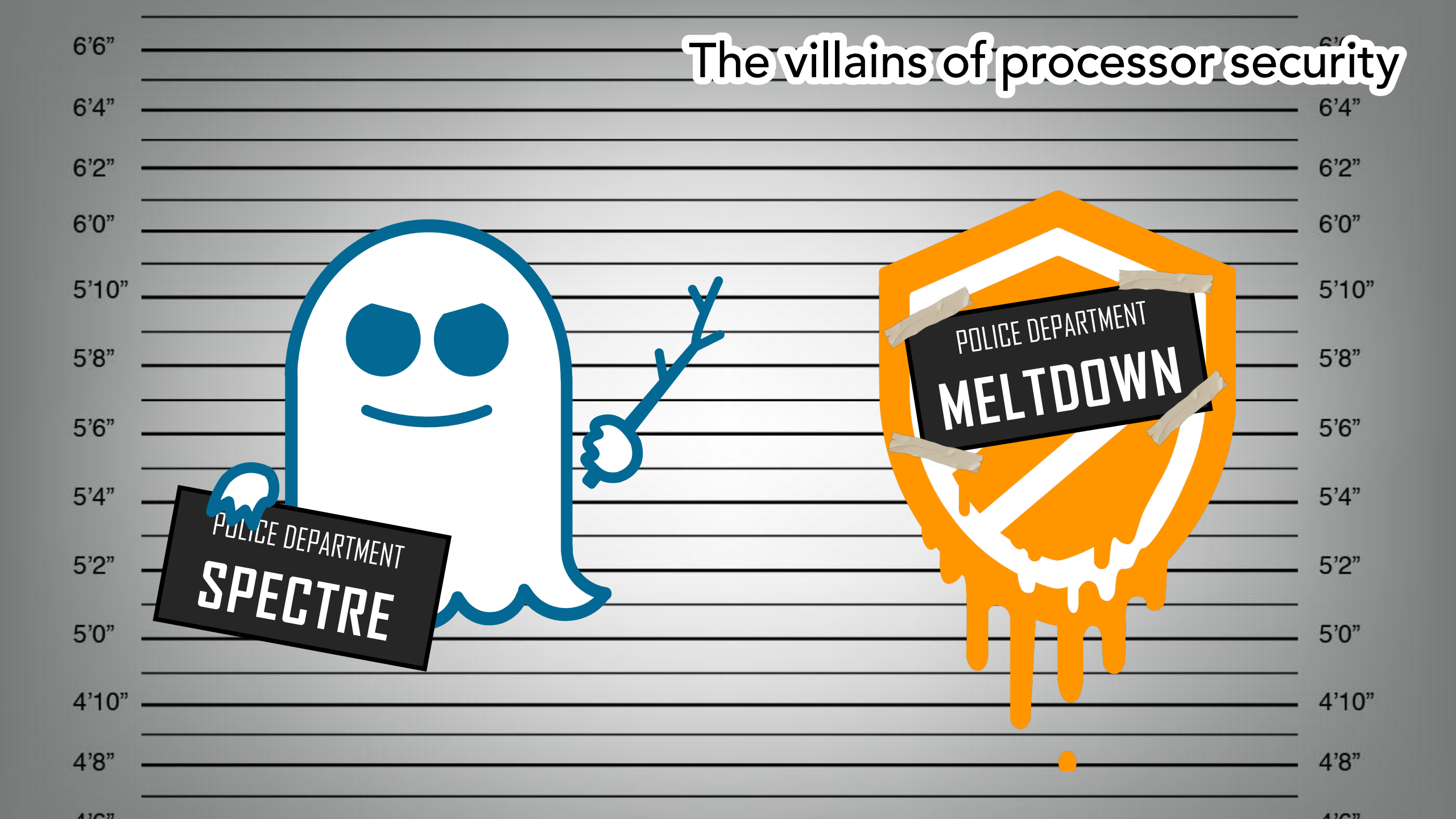
Ok but *fundamentally* we're done right…?

NO!



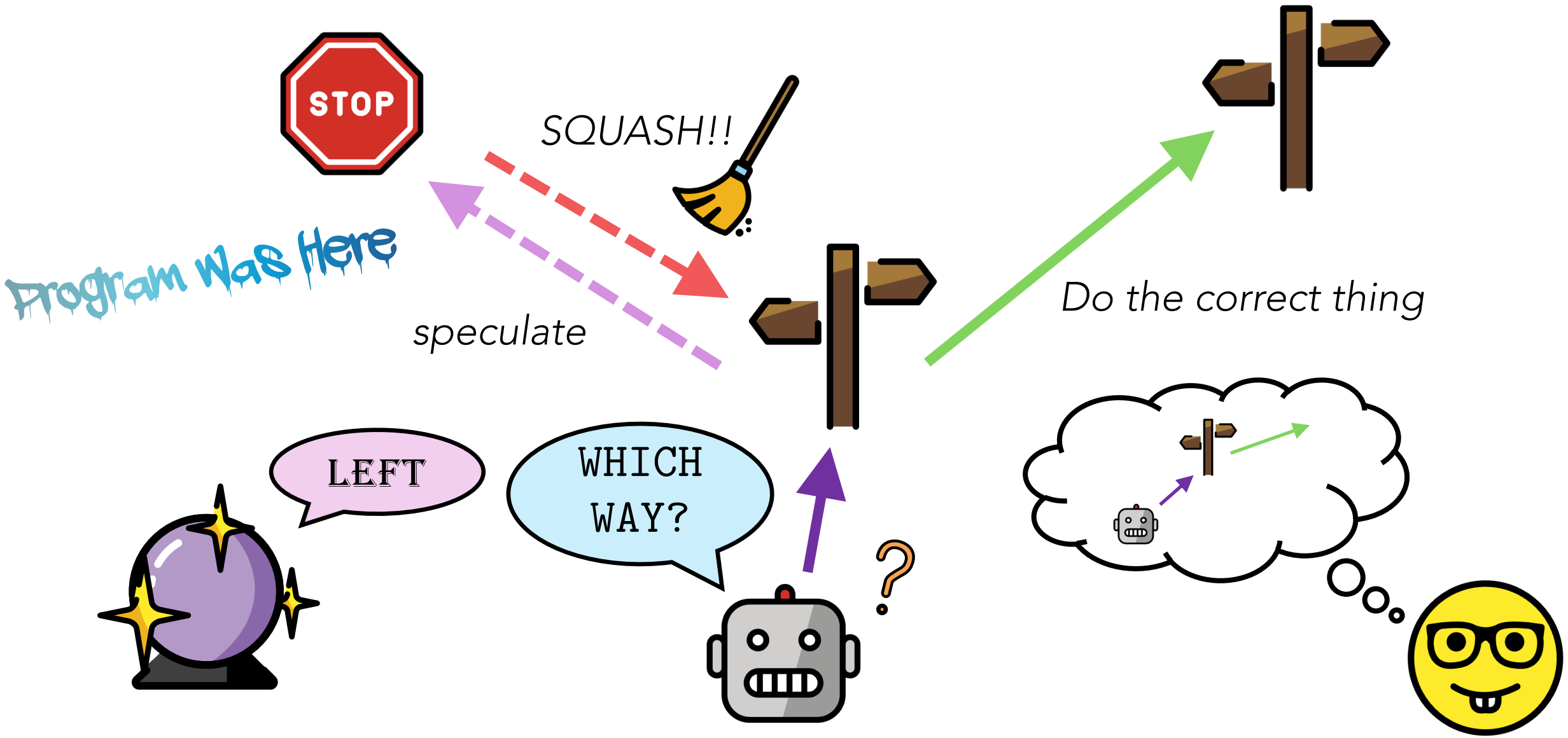
This loser is about to ruin our whole lives **AGAIN!!!!**

The villains of processor security



The villains of processor security

Speculation, AKA the art of just guessing



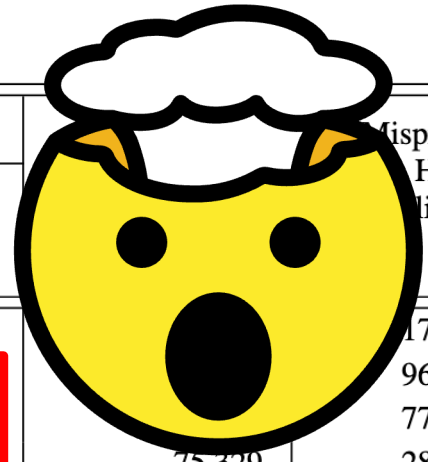
You're sooo predictable (but that's a good thing)

SPECint2017 Benchmark	Avg # Phases	# Static Branches		Avg. Acc.	Avg. Acc. excl. H2Ps	# App. Inputs	H2P Appearance Across Inputs		# Static H2P Branches		Avg. Dyn. Execs per H2P per Slice	% Mispreds due to H2Ps per Slice
		Total	Median per Slice				Total	3+ Inputs	Avg per Input	Avg per Slice		
600.perlbench_s	6.5	13,865	1,863	0.987	0.989	4	62	16	21.5	1	93,815	17.3%
605.mcf_s	11.4	1,755	99	0.921	0.998	8	29	20	19.0	10	249,195	96.9%
620.omnetpp_s	11.8	7,099	823	0.975	0.994	5	46	28	28.0	8	74,630	77.6%
623.xalancbmk_s	7.5	8,563	3,103	0.997	0.998	4	28	8	14.5	6	75,329	28.6%
625.x264_s	13.9	4,892	1,068	0.946	0.975	14	23	7	6.0	1	65,593	54.2%
631.deepsjeng_s	9.4	3,162	856	0.946	0.963	12	68	49	40.0	13	44,412	31.2%
641.leela_s	8.8	3,623	582	0.880	0.960	10	77	68	56.5	34	35,614	66.4%
648.exchange2_s	8.4	3,765	1,330	0.986	0.992	5	38	19	20.0	7	142,320	44.7%
657.xz_s	7.6	2,373	211	0.897	0.980	5	163	50	63.0	10	75,759	80.5%
MEAN	9.5	5,455	1,104	0.952	0.984	7	59	29	30.0	10	95,185	55.3%

TABLE I: Summary statistics of our SPECint 2017 data set, which includes an expanded collection of inputs for each benchmark. Metrics are averaged over 10B-instruction traces from each input. Accuracy and H2P statistics are reported for TAGE-SC-L 8KB.

Table from "Branch Prediction Is Not A Solved Problem: Measurements, Opportunities, and Future Directions" by Lin and Tarsa

You're sooo predictable (but that's a good thing)

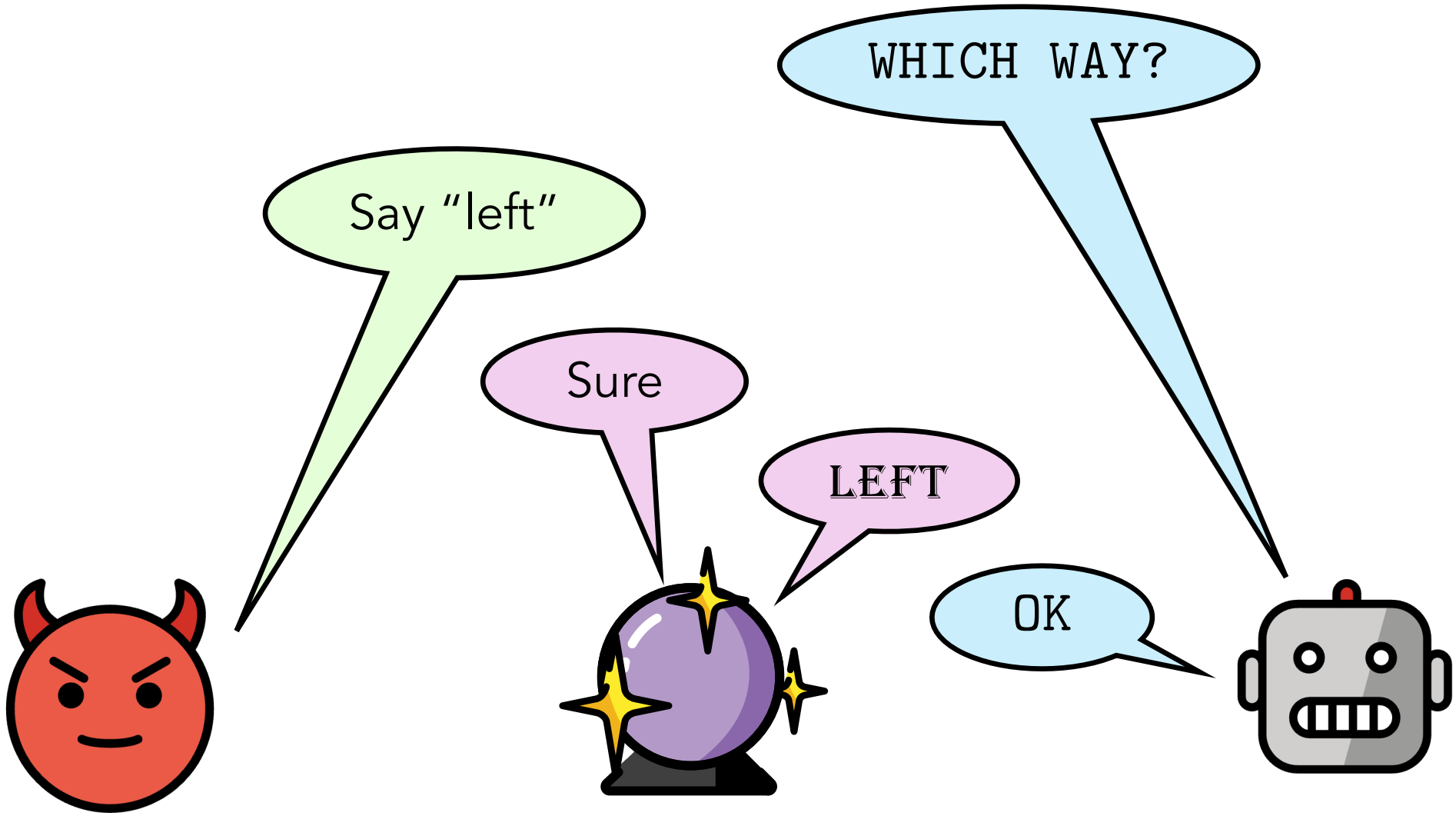


SPECint2017 Benchmark	Avg # Phases	# Static Branches		Avg. Acc.	Avg. Acc. excl. H2Ps	H2P				H2P		Mispreds
		Total	Median per Slice							Avg per Slice		H2Ps
600.perlbench_s	6.5	13,865	1,863	0.987	0.989	4	62	16	21.5	1		17.3%
605.mcf_s	11.4	1,755	99	0.921	0.9[illegible]							96.9%
620.omnetpp_s	11.8	7,099	823	0.973	0.994							77.6%
623.xalancbmk_s	7.5	8,563	3,103	0.997	0.998						75,329	28.6%
625.x264_s	13.9	4,892	1,068	0.946	0.975						65,593	54.2%
631.deepsjeng_s	9.4	3,162	856	0.946	0.963						44,412	31.2%
641.leela_s	8.8	3,623	582	0.880	0.960						35,614	66.4%
648.exchange2_s	8.4	3,765	1,330	0.986	0.992						142,320	44.7%
657.xz_s	7.6	2,373	211	0.897	0.980	5	163	50	63.0	10	75,759	80.5%
MEAN	9.5	5,455	1,104	0.952	0.984	7	59	29	30.0	10	95,185	55.3%

TABLE I: Summary statistics of our SPECint 2017 data set, which includes an expanded collection of inputs for each benchmark. Metrics are averaged over 10B-instruction traces from each input. Accuracy and H2P statistics are reported for TAGE-SC-L 8KB.

Table from "Branch Prediction Is Not A Solved Problem: Measurements, Opportunities, and Future Directions" by Lin and Tarsa

Unfortunately, predictors are pushovers

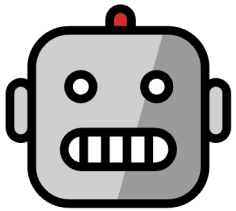


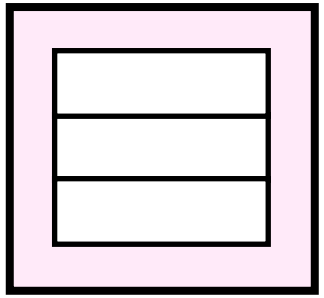
⚠ *This is grossly simplified but it doesn’t matter*

Bypassing the bounds check

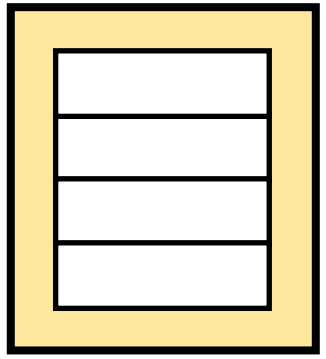
```
x = ...
if (x < A.size) {
  y = A[x]
  z = B[y]
}
```

SPECTRE GADGET

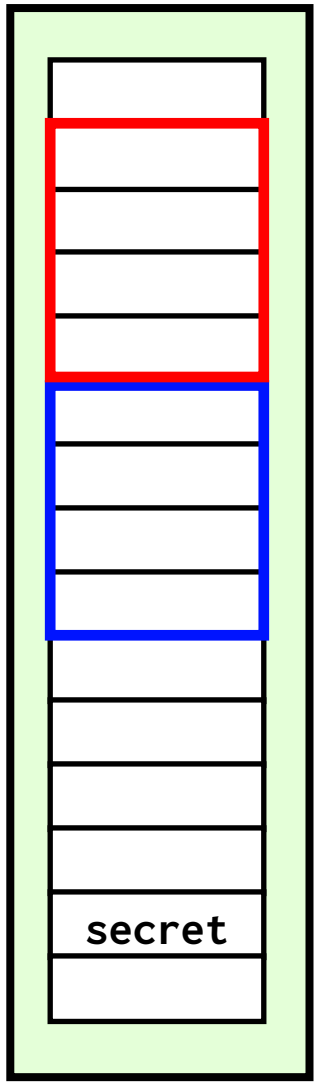




REGS

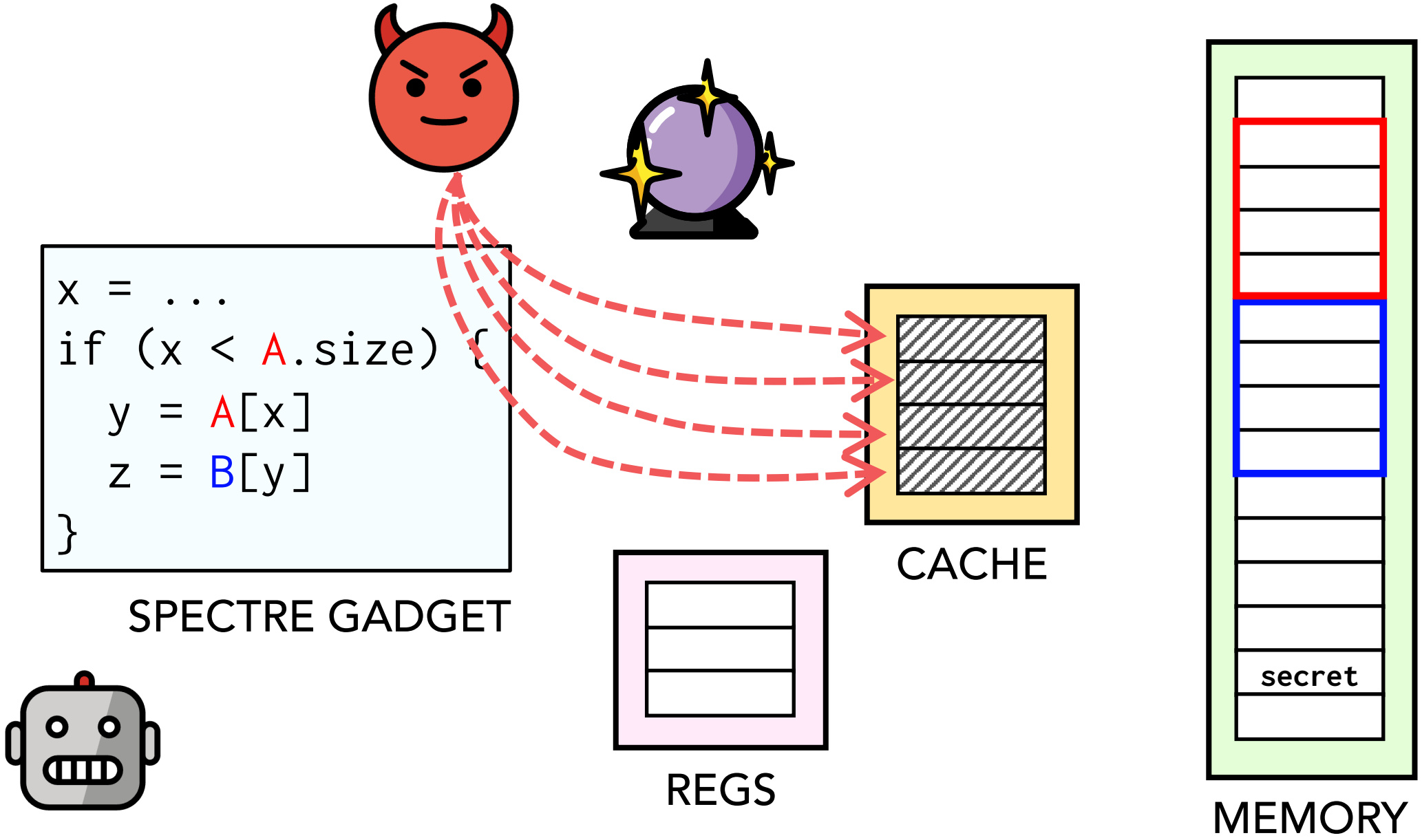


CACHE

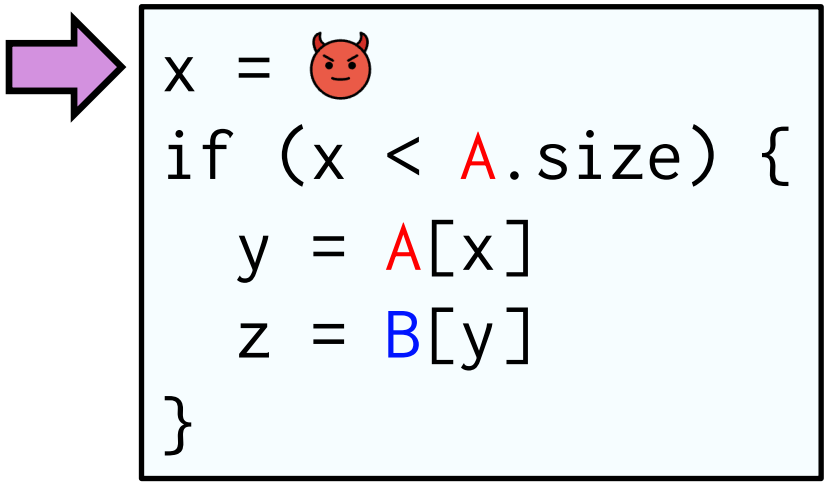


MEMORY

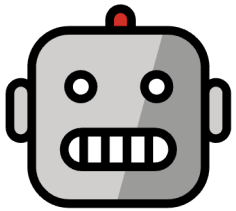
Bypassing the bounds check

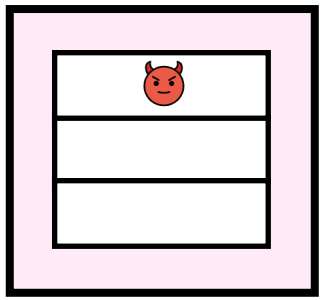


Bypassing the bounds check

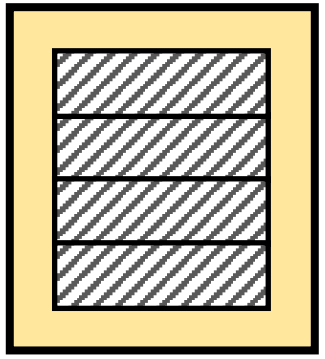


SPECTRE GADGET

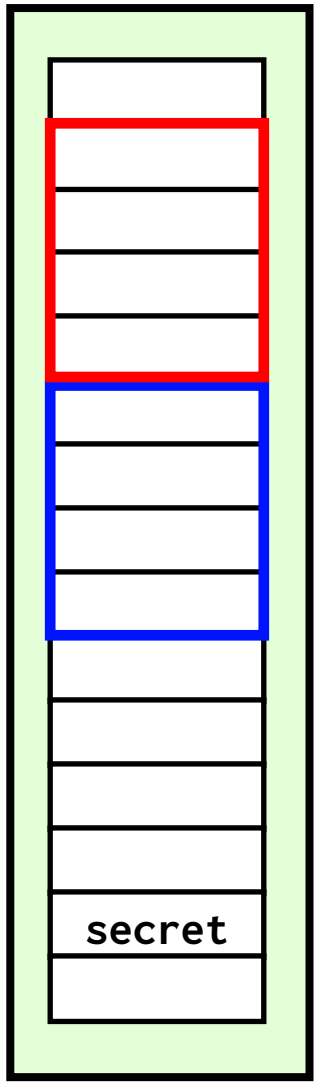




REGS

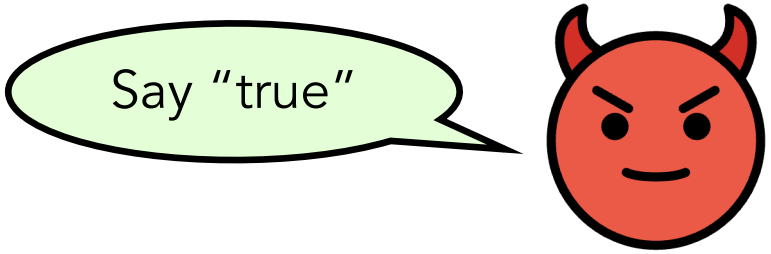


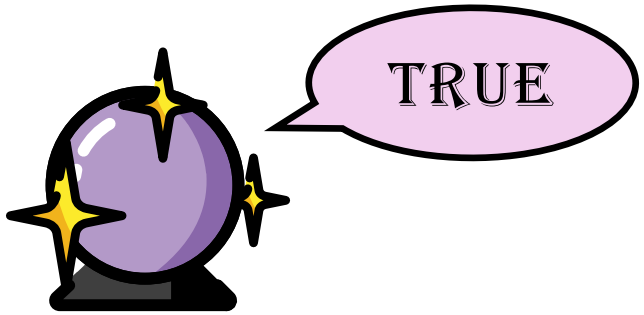
CACHE



MEMORY

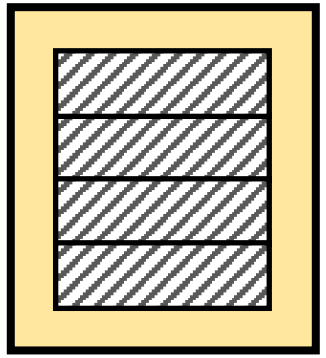
Bypassing the bounds check



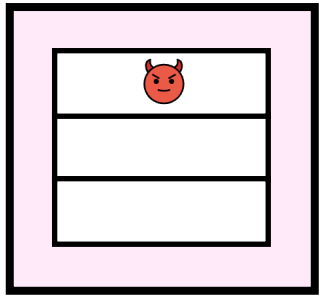



```
x = 😈
if (x < A.size) {
   y = A[x]
   z = B[y]
}
```

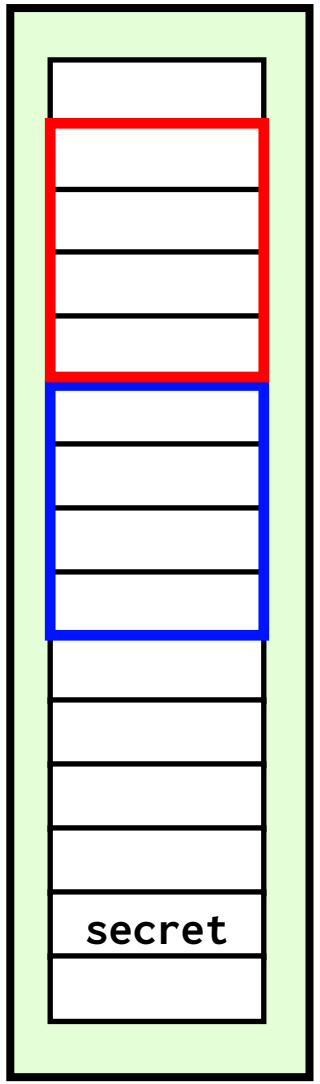
SPECTRE GADGET



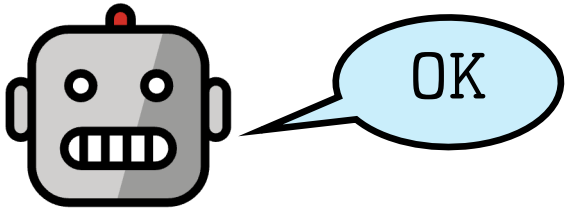
CACHE



REGS



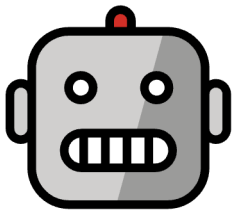
MEMORY

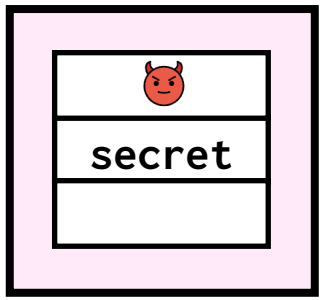


Bypassing the bounds check

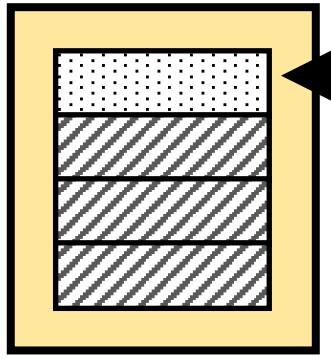
```
x = 
if (x < A.size) {
   y = A[x]
   z = B[y]
}
```

SPECTRE GADGET

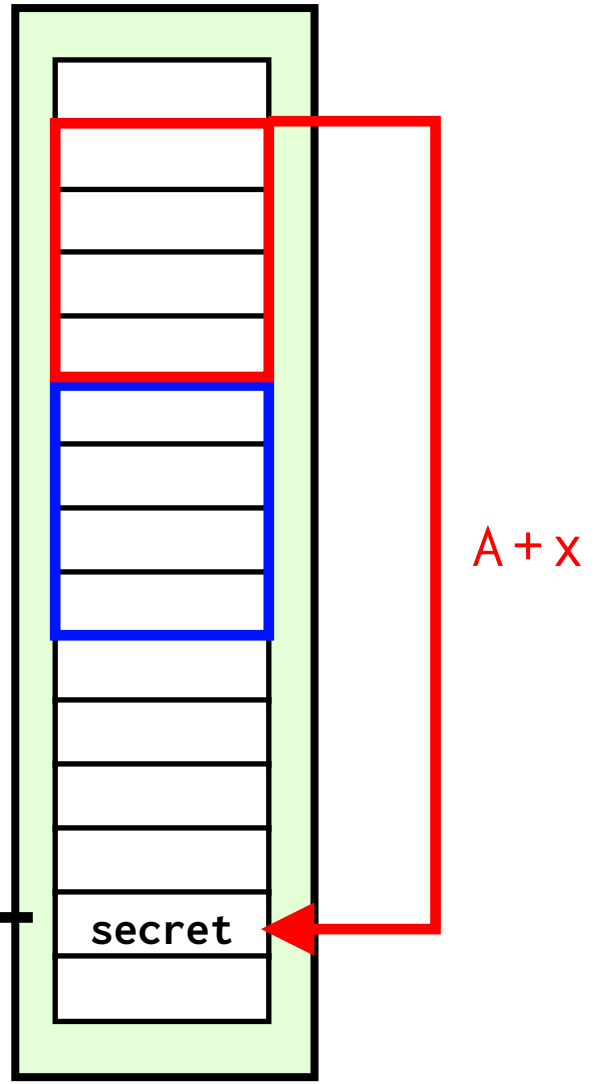




REGS



CACHE



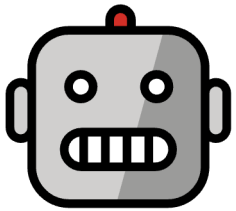
A+x

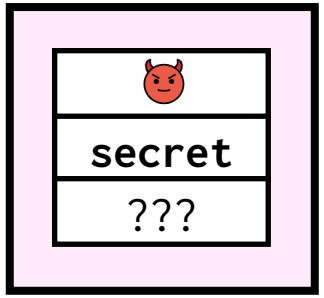
MEMORY

Bypassing the bounds check

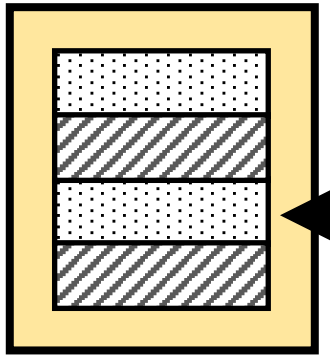
```
x = 😈
if (x < A.size) {
   y = A[x]
   z = B[y]
}
```

SPECTRE GADGET

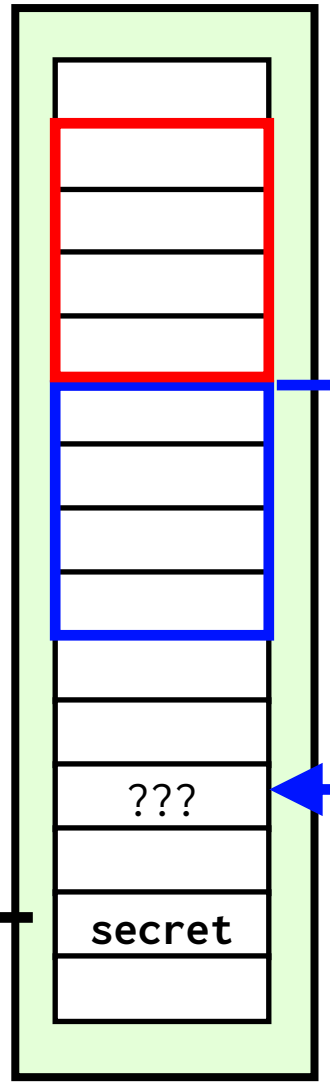




REGS



CACHE



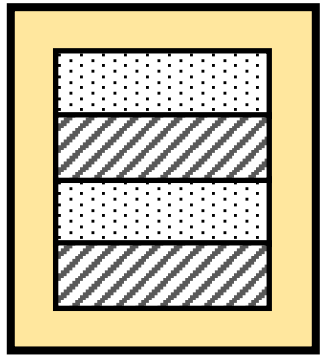
B + secret

MEMORY

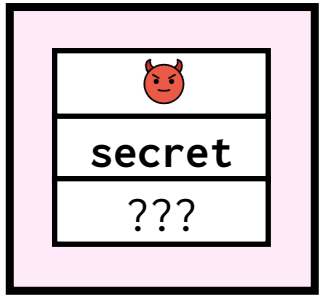
Bypassing the bounds check

```
x = 😈
if (x < A.size) {
   y = A[x]
   z = B[y]
}
```

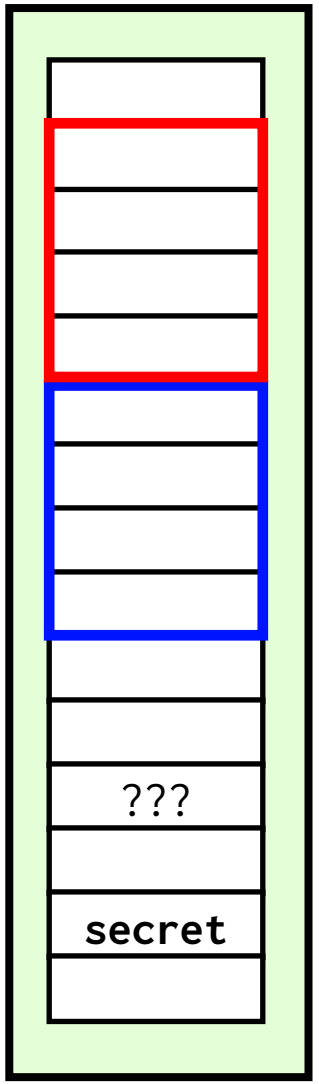
SPECTRE GADGET



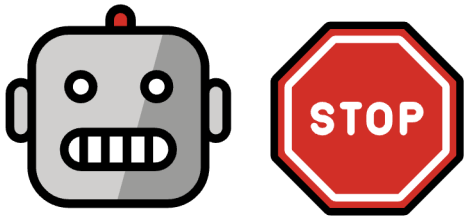
CACHE



REGS



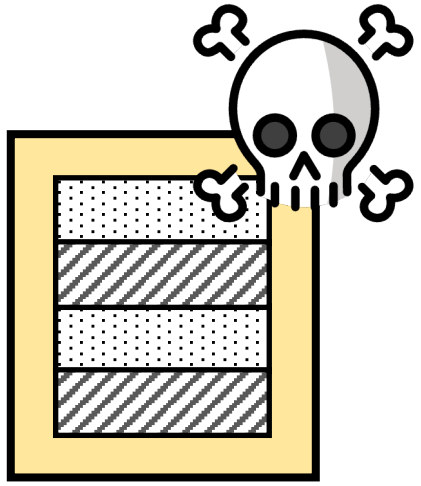
MEMORY



Bypassing the bounds check

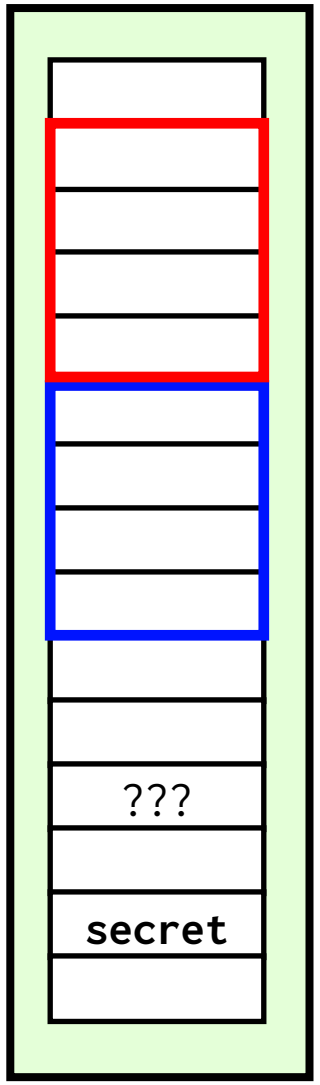
```
x = 😈
if (x < A.size) {
   y = A[x]
   z = B[y]
}
```

SPECTRE GADGET

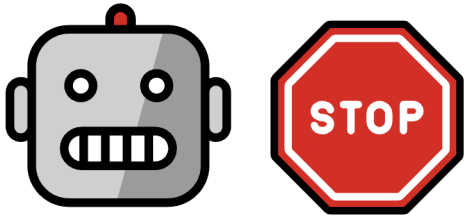


CACHE

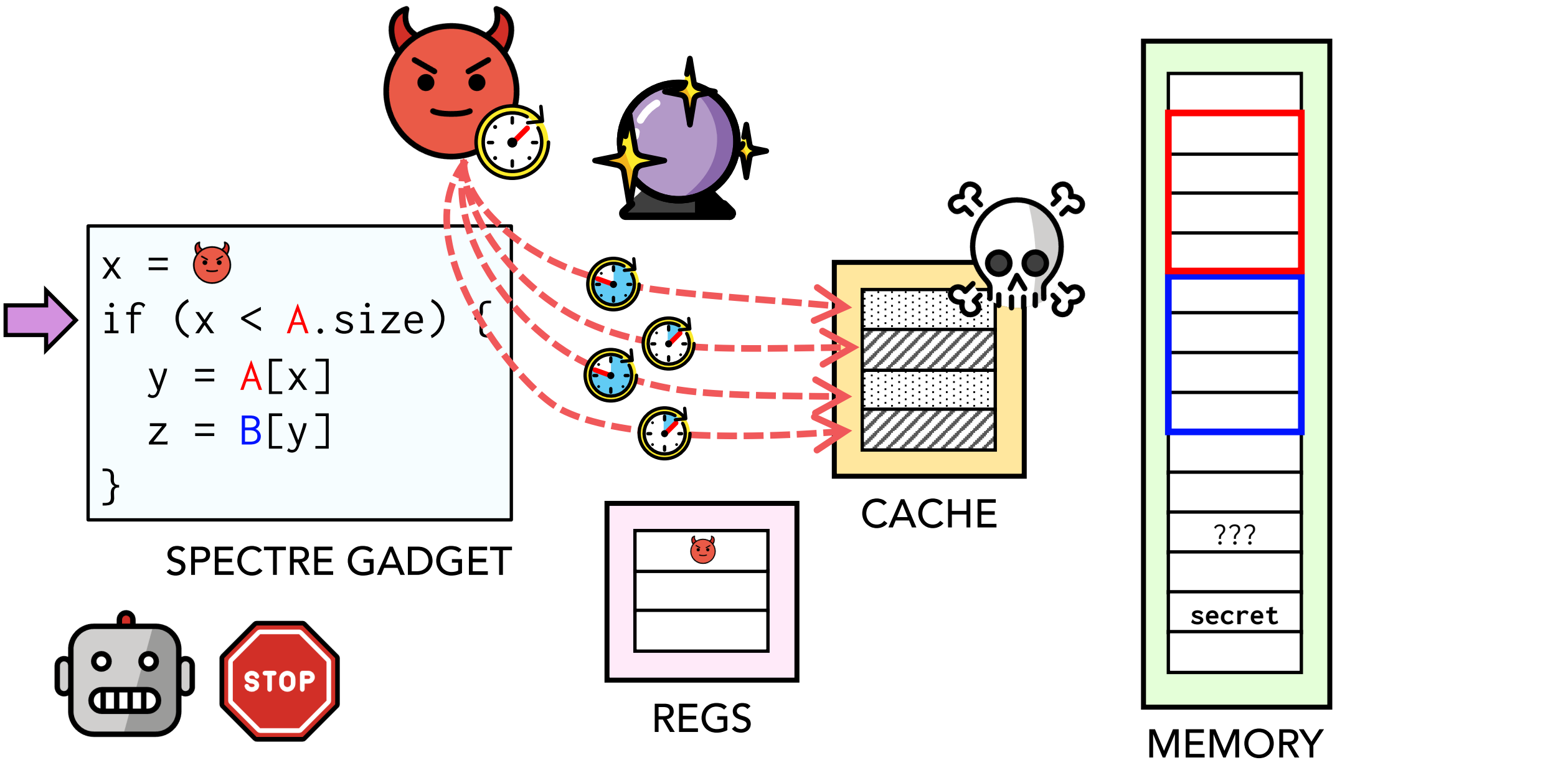
REGS



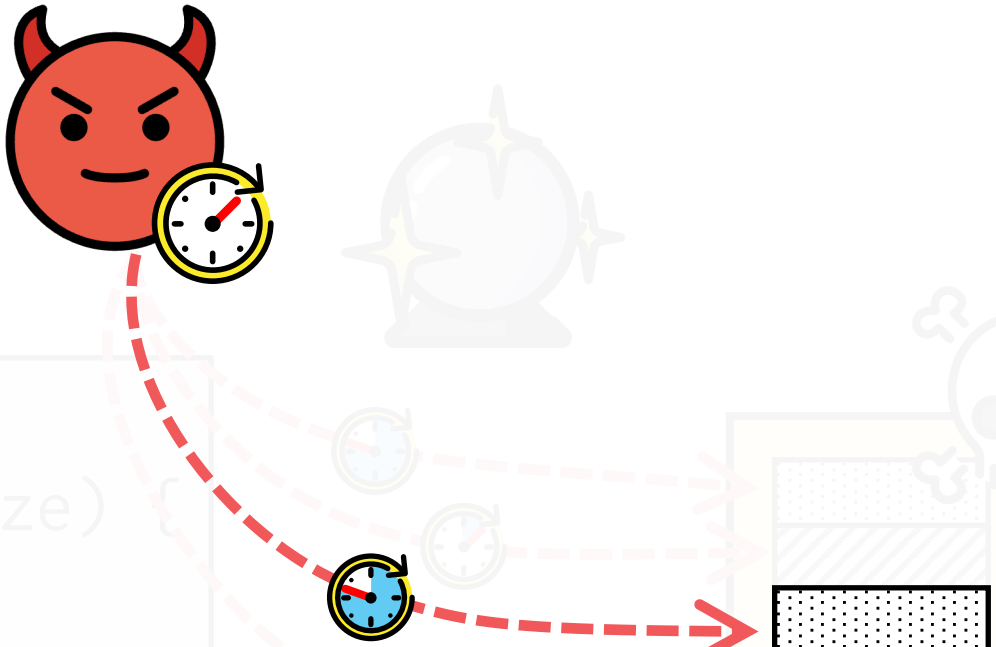
MEMORY



Bypassing the bounds check



Bypassing the bounds check



The location of this cache line depended on the value of `B+secret`... it was a function of a secret!

Mama mia!

NetSpectre: Read Arbitrary Memory over Network

Michael Schwarz
Graz University of Technology

Moritz Lipp
Graz University of Technology

SMoTherSpectre: Exploiting Speculative Execution through Port Contention

Atri Bhattacharyya *
EPFL

Alessandro Sorniotti †
IBM Research – Zurich

Spook.js: Attacking Chrome Strict Site Isolation via Speculative Execution

Ayush Agarwal*

iLeakage: Browser-based Timerless Speculative Execution Attacks on Apple Devices

Stephan van Schaik
University of Michigan
Ann Arbor, USA

Daniel Genkin
Georgia Tech
Atlanta, USA
genkin@gatech.edu

Yuval Yarom*
Ruhr University Bochum
Bochum, Germany
yuval.yarom@rub.de

Proceedings of the 2023 ACM SIGSAC Conference on Computer (CCS '23), November 26–30, 2023, Copenhagen, https://doi.org/10.1145/

Speculative Buffer Overflows: Attacks and Defenses

Vladimir Kiriansky
vlk@csail.mit.edu

Carl Waldspurger
carl@waldspurger.org

Abstract

Practical attacks that exploit speculative execution ca
confidential information via microarchitectural side
nels. The recently-demonstrated Spectre attacks lev
speculative loads which circumvent access checks to
memory-resident secrets, transmitting them to an atta
using cache timing or other covert communication char

We introduce SPECTRE1.1, a new Spectre-v1 variant
leverages speculative *stores* to create *speculative buffer*
flows. Much like classic buffer overflows, speculative ou
bounds stores can modify data and code pointers. Data-va
attacks can bypass some Spectre-v1 mitigations, either
rectly or by redirecting control flow. Control-flow attac
enable arbitrary speculative code execution, which can
pass fence instructions and all other software mitigatio

PACMAN: Attacking ARM Pointer Authentication with Speculative Execution

Joseph Ravichandran*
MIT CSAIL
Cambridge, MA, USA
jravi@mit.edu

Weon Taek Na*
MIT CSAIL
Cambridge, MA, USA
weontaek@mit.edu

Jay Lang

Mengjia Yan
MIT CSAIL
Cambridge, MA, USA
mengjiay@mit.edu

1 INTRODUCTION

Modern systems are becoming increasingly complex, exposing a large attack surface with vulnerabilities in both software and hardware. In the software layer, memory corruption vulnerabilities [16, 56, 59, 61] (such as buffer overflows) can be exploited by attackers to alter the behavior or take full control of a victim program. In the hardware layer, micro-architectural side channel vulnerabilities [18, 25] (such as Spectre [37] and Meltdown [43]) can be exploited to leak arbitrary data within the victim program's address space. Today, it is common for security researchers to ex-

Spectre Returns! Speculation Attacks using the Return Stack Buffer

Esmaeil Mohammadian Koruyeh, Khaled N. Khasawneh,
Chengyu Song and Nael Abu-Ghazaleh
Computer Science and Engineering Department
University of California, Riverside
naelag@ucr. edu

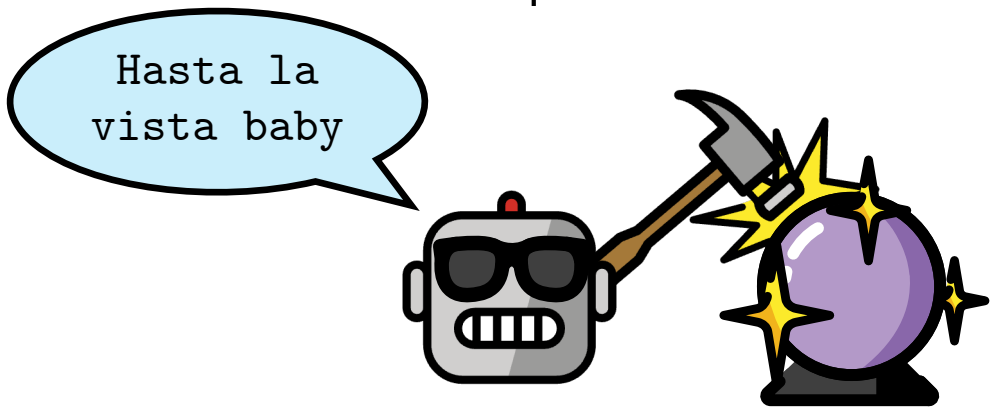
Abstract

The recent Spectre attacks exploit speculative execution, a pervasively used feature of modern microprocessors, to

not take effect until the instruction is committed. The recent Spectre attack [23, 13, 31] has shown that this behavior can be exploited to expose information that is otherwise inaccessible. In the two variants of Spectre attacks, attackers either mistrain the branch predictor unit

Alright, we can fix this the strawman way…

If speculation causes security vulnerabilities, why not just disable it?



Unfortunately branch prediction is one of the fundamental pillars of modern processors

"We do not feel justified in giving up what has come to the industry like a gift from heaven on the possibility that a hazard may be involved in it"

— Standard Oil president Frank Howard, 1925

… defending leaded gasoline

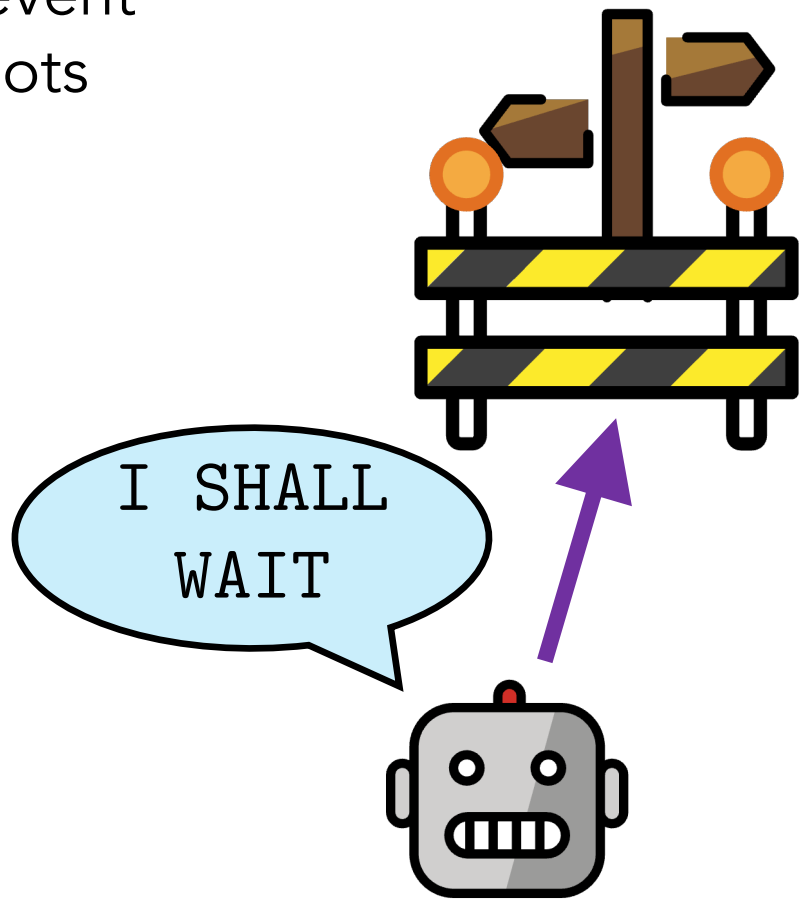
Goals for our cake:

1. Have it
2. Eat it

… or the dumb software way…

Second easiest thing we can do is prevent speculation in just the problematic spots

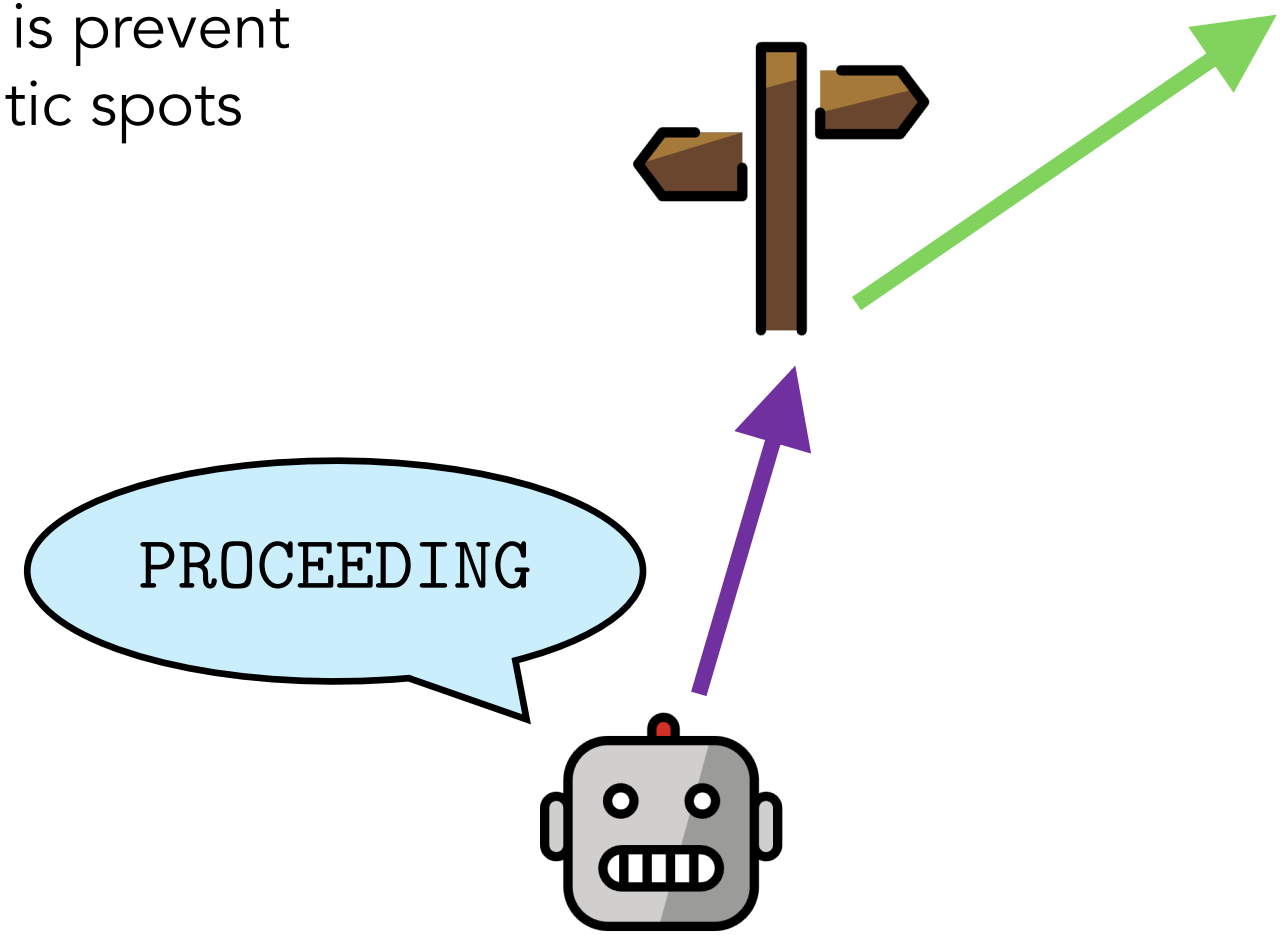
```
x = ...
if (cond) {
  lfence
  y = load(x)
  z = load(y)
}
```



… or the dumb software way…

Second easiest thing we can do is prevent speculation in just the problematic spots

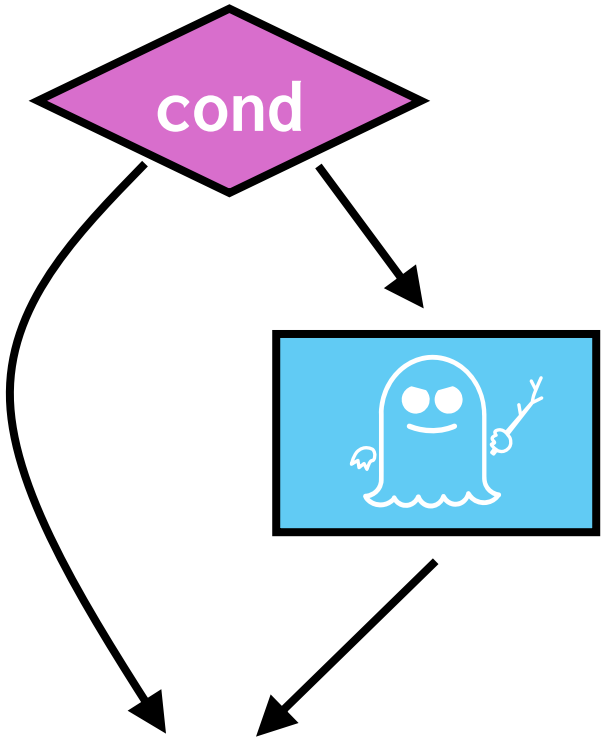
```
x = ...
if (cond) {
  lfence
  y = load(x)
  z = load(y)
}
```

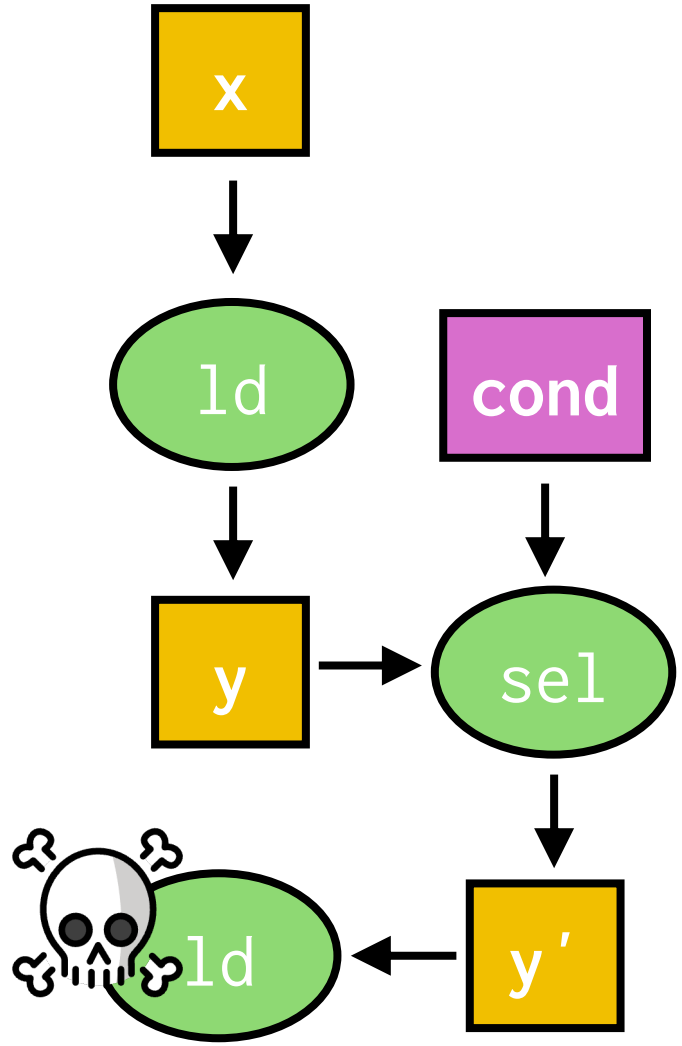


… or the clever software way…

Speculative load hardening (SLH) ties the loaded value to the branch predicate via a *data-dependency*, which can't be speculated past!

```
x = ...
if (cond) {
  y = load(x)
  y' = cond ? y : 0
  z = load(y')
}
```

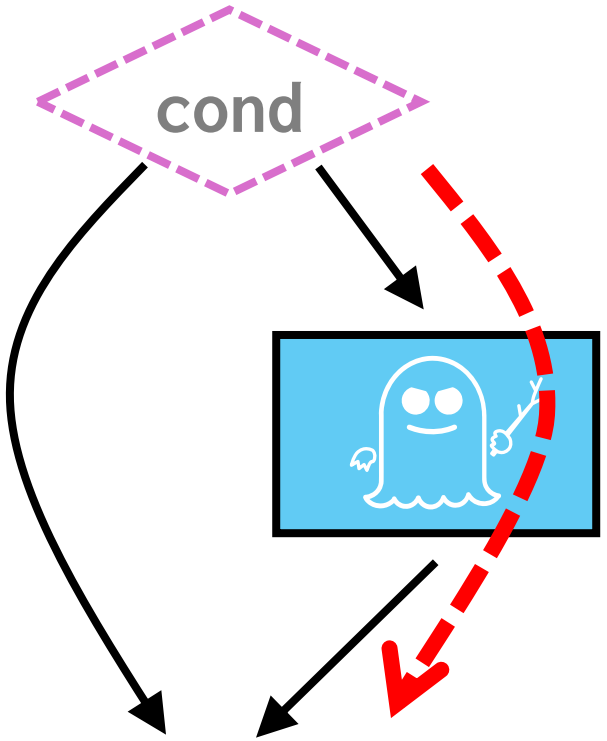


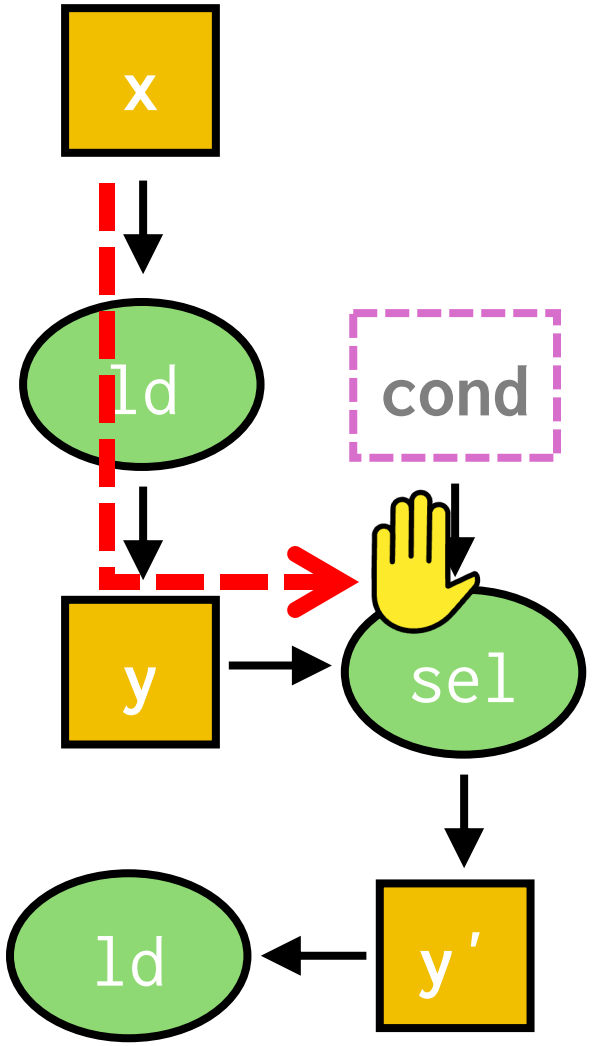


… or the clever software way…

Speculative load hardening (SLH) ties the loaded value to the branch predicate via a *data-dependency*, which can't be speculated past!

```
x = ...
if (cond) {
  y = load(x)
  y' = cond ? y : 0
  z = load(y')
}
```





… or the hard(ware) way!

Speculative Taint Tracking (STT): A Comprehensive Protection for Speculatively Accessed Data

Jiyong Yu
University of Illinois at Urbana-Champaign
jiyongy2@illinois.edu

Mengjia Yan

em Khyzha
viv University
a@mail.tau.ac.il

MuonTrap: Preventing Cross-Domain Spectre-Like Attacks by Capturing Speculative State

Sam Ainsworth
University of Cambridge
Cambridge, UK
sam.ainsworth@cl.cam.ac.uk

Half&Half: Demystifying Intel's Directional for Fast, Secure Partitioned Ex

Hosein Yavarzadeh*, Mohammadkazem Taram†, Shravan Narayan*

*University of California San Diego, †Purdue University,

DAWG: A Defense Against Cache Timing Attacks in Speculative Execution Processors*

Vladimir Kiriansky†, Ilia Lebedev†, Saman Amarasinghe†, Srinivas Devadas†, Joel Emer‡
†*MIT CSAIL,* ‡*NVIDIA / MIT CSAIL*
{vlk, ilebedev, saman, devadas, emer}@csail.mit.edu

***Abstract*—Software side channel attacks have become a serious concern with the recent rash of attacks on speculative processor architectures. Most attacks that have been demonstrated exploit the cache tag state as their exfiltration chan**

victim's protection domain

NDA: Preventing Speculative Execution Attacks at Their Source

Ofir Weisse
University of Michigan

Ian Neal
University of Michigan

Kevin Loughlin
University of Michigan

Thomas F. Wenisch
University of Michigan

ABSTRACT

Speculative execution attacks like Meltdown and Sp
accessing secret data in wrong-path execution. Secrets
mitted and recovered by the attacker via a covert char
mitigations either require code modifications, address
exploit techniques, or block only the cache covert cha
than battling exploit techniques and covert channels
we seek to close off speculative execution attacks at t
Our key observation is that these attacks require a chai
dent wrong-path instructions to access and transmit secre
propose *NDA*, a technique to restrict speculative data pr
NDA breaks the attacks' wrong-path dependence chains
allowing speculation and dynamic scheduling. We describe
space of *NDA* variants that differ in the constraints they
they prevent. *NDA* preserves much of the performance ad
dynamic scheduling and the classes of speculative execution
of out-of-order execution: on SPEC CPU 2017, *NDA* variant
68-96% of the performance gap between in-order and unconst

Doppelganger Loads: A Safe, Complexity-Effective Optimization for Secure Speculation Schemes

Amund Bergland Kvalsvik, Pavlos Aimoniotis†, Stefanos Kaxiras†, and Magnus Själander
Kvalsvik@ntnu.no|Pavlos.Aimoniotis@it.uu.se|Stefanos.Kaxiras@it.uu.se|Magnus.Sjalander@ntnu.no
Norwegian University of Science and Technology, Trondheim, Norway
†Uppsala University, Uppsala, Sweden

hannel attacks have forced computer architects
ative execution. Effectively preventing microar-
from leaking sensitive information will be a key
ture processor design.
limitation of many secure speculation schemes is
e available memory parallelism, as unsafe loads
e particular scheme) are blocked, as they might
information. Our contribution is to show that it
cover some of this lost memory parallelism, by
the addresses of these loads in a threat-model
, i.e., without worsening the security guarantees
g secure scheme. To demonstrate the generality
we apply it to three different secure speculation
peculative Data Access (NDA), Speculative Taint
and Delay-on-Miss (DoM).

ACM Reference Format:
Amund Bergland Kvalsvik, Pavlos Aimoniotis†, Stefanos Kaxiras†, and Magnus Själander. 2023. Doppelganger Loads: A Safe, Complexity-Effective Optimization for Secure Speculation Schemes. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA '23), June 17–21, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3579371.3589088

1 INTRODUCTION

With the disclosure of Spectre [26] many architectures, previously considered secure, were shown to be unsafe because speculative execution in large part disregarded security vulnerabilities. Since their original introduction, many variations of the attacks have been developed [2, 8, 10, 12, 35, 42, 51], and early mitigation strategies [3, 4, 6, 7, 19, 22, 24, 25, 27, 29, 30, 38–41, 47, 49, 52–56] strive to comprehensively eliminate speculative side-channel attacks at

GhostMinion: A Strictness-Ordered Cache System for Spectre Mitigation

Sam Ainsworth
sam.ainsworth@ed.ac.uk
University of Edinburgh
Edinburgh, UK

ABSTRACT

Out-of-order speculation, a technique ubiquitous since the early 1990s, remains a fundamental security flaw. Via attacks such as Spectre and Meltdown, an attacker can trick a victim, in an otherwise entirely correct program, into leaking its secrets through the effects of misspeculated execution, in a way that is entirely invisible to the programmer's model. This has serious implications for application sandboxing and inter-process communication.

Designing efficient mitigations that preserve the performance of out-of-order execution has been a challenge. The speculation-hiding techniques in the literature have been shown to not close such channels comprehensively, allowing adversaries to redesign attacks. Strong, precise guarantees are necessary, but mitigations must achieve high performance to be adopted. We present Strict-

functionally correct programs can be compromised through the leakage of secrets via incorrectly predicted program behaviour. Still, the side channels that cause this leakage, for example caches, are desirable in many cases; the speed of modern microarchitectures is only possible because of structures that leak soft state. Out-of-order speculation, which allows attackers to execute complex speculative code, is mandatory for high instruction-level parallelism.

Many different mitigation techniques, with differing threat models and security properties, have been proposed. Often the tradeoffs are unclear, as the guarantees necessary, and provided, are typically opaque. This has resulted in subtle bugs in many proposed countermeasures. Since precise models of the behaviour guaranteed by mitigations are typically unavailable, sophisticated attacks to circumvent restrictions, such as SpectreRewind [10] and Speculative Interference [5] have appeared to fill in the gaps. We now know that

Well… at least there's plenty of grant money in the future

Opening Pandora's Box: A Systematic Study of New Ways Microarchitecture Can Leak Private Data

Jose Rodrigo Sanchez Vicarte, Pradyumna Shome, Nandeeka Nayak, Caroline Trippel†, Adam Morrison‡, David Kohlbrenner§, Christopher W. Fletcher

University of Illinois at Urbana-Champaign, †Stanford University, ‡Tel Aviv University, §University of Washington
{josers2, pshome2, ndnayak2, cwfletch}@illinois.edu, trippel@stanford.edu, mad@cs.tau.ac.il, dkohlbre@cs.washington.edu

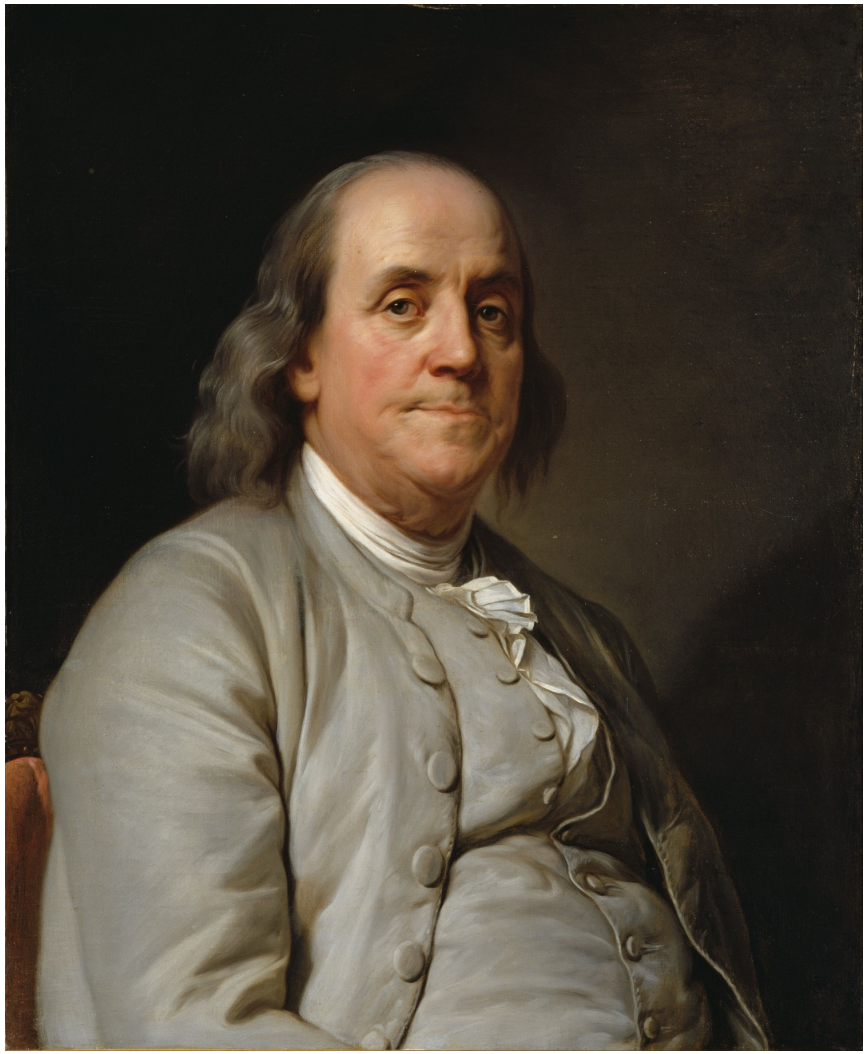
***Abstract*—Microarchitectural attacks have plunged Computer Architecture into a security crisis. Yet, as the slowing of Moore's law justifies the use of ever more exotic microarchitecture, it is likely we have only seen the tip of the iceberg.**

To better anticipate this security crisis, this paper performs a systematic security-centric analysis of the Computer Architecture literature. Our rationale is that when implementing current and future processors, microarchitects will (quite reasonably) look to previously-proposed ideas. Our study uncovers seven classes of microarchitectural optimization with novel security implications, proposes a conceptual framework through which to study them and demonstrates several proofs-of-concept to show their efficacy. The optimizations we study range from those that leak as much privacy as Spectre/Meltdown (but without exploiting speculative execution) to those that otherwise undermine security-critical

"value prediction has regained interest in an era where Moore's Law is fading and Dennard Scaling is gone" [7].

Motivating example: data memory-dependent prefetchers leak as much privacy as Spectre/Meltdown. As a motivating example, consider prefetching. While the community is well-aware of the security implications of "traditional" stride and software prefetching [8,9], this is only the tip of the iceberg in the prefetcher literature. In particular, there is significant literature on *data memory-dependent prefetchers*, i.e., prefetchers that take into account the *contents of data memory directly* [10–16]. Such prefetchers are effective in cases where stride prefetchers fail, e.g., in applications domi-

2021



“Those who would give up essential liberty to purchase a little temporary safety deserve neither liberty nor safety.”

"Those who would give up security *essential liberty to*

purchase a little temporary performance *safety deserve neither*

security performance *liberty nor safety."*