black hat®
USA 2024

AUGUST 7-8, 2024
BRIEFINGS

Bypassing ARM's Memory Tagging Extension with a Side-Channel Attack

Speaker: Juhee Kim

Whoami

Juhee Kim

Ph.D Student at CompSec Lab,
Seoul National University
kimjuhi96@snu.ac.kr

Focuses on

- Software and Systems security
- Bug finding, Attack mitigation
- Linux kernel, Web browser, GPU/ML systems

Contributors

Jinbum Park

- Security researcher at Samsung Research
- System security, Confidential Computing
- Published in USENIX Security and ASPLOS

Sihyeon Roh

- Ph.D Student at CompSec Lab
- Hardware side-channels

Jaeyoung Chung

- Ph.D Student at CompSec Lab
- System Security
- CTF player

Youngjoo Lee

- Ph.D Student at CompSec Lab
- Fuzzing, Browser security, Bug gounty
- CTF player

Samsung Research

Georgia Institute of Technology

Taesoo Kim

- Vice president of Samsung Research
  Professor of Georgia Tech
- Won several best paper awards from USENIX Security, EuroSys

Byoungyoung Lee

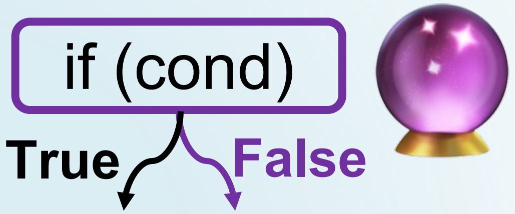
- Professor of Seoul National University
- Leads CompSec Lab
- System security, Confidential computing
- Previous CTF player
- Spoken at Black Hat

Roadmap

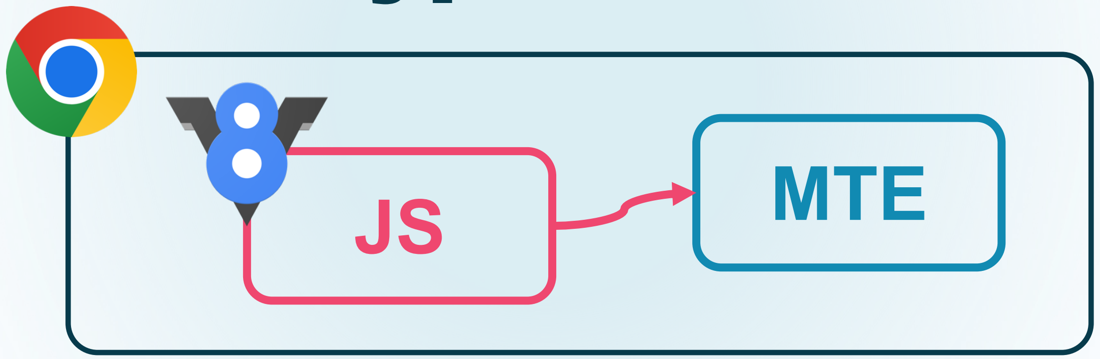
ARM Memory Tagging Extension

Cache Side-Channel

Speculative Execution



Real-world MTE Bypass Attack



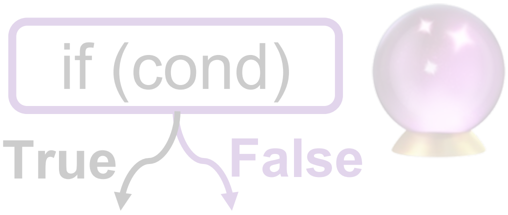
MTE Tag Leakage Side-Channel

Roadmap

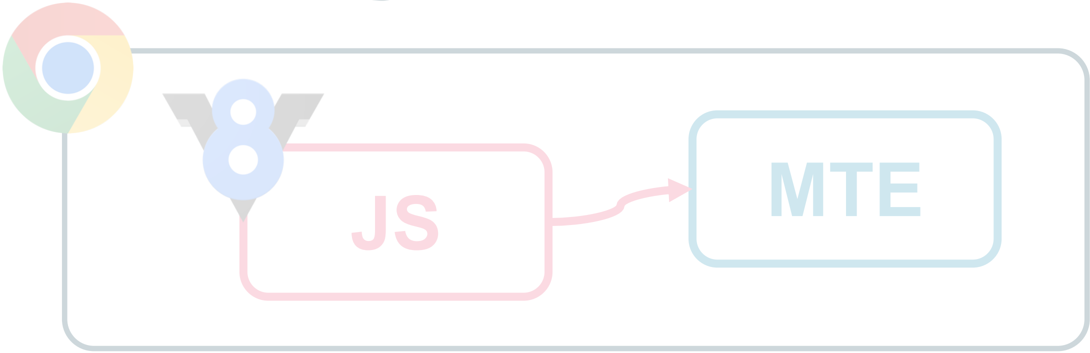
ARM Memory Tagging Extension

Cache Side-Channel

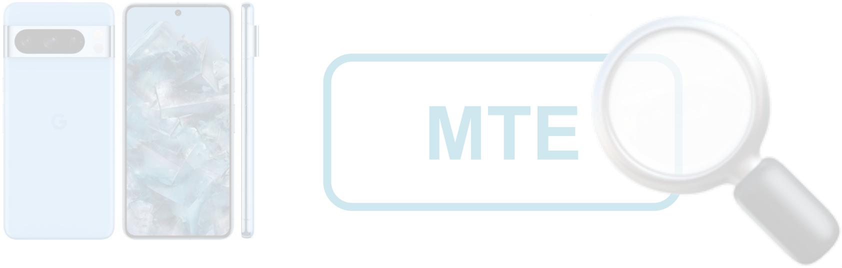
Speculative Execution



Real-world MTE Bypass Attack



MTE Tag Leakage Side-Channel



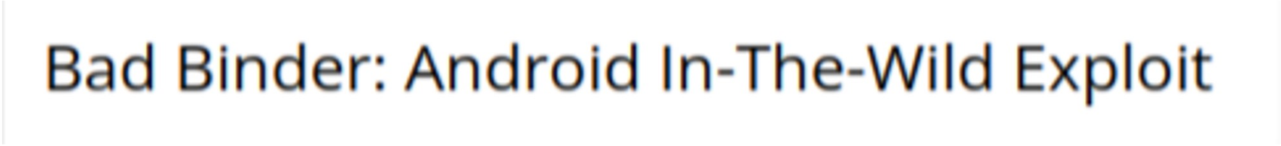
Memory corruption attacks

have been the most pervasive and dangerous security threats

Heartbleed (2014)

OpenSSL information leak

Bad Binder (2019)



Bad Binder: Android In-The-Wild Exploit

reggreSSHion (2024)

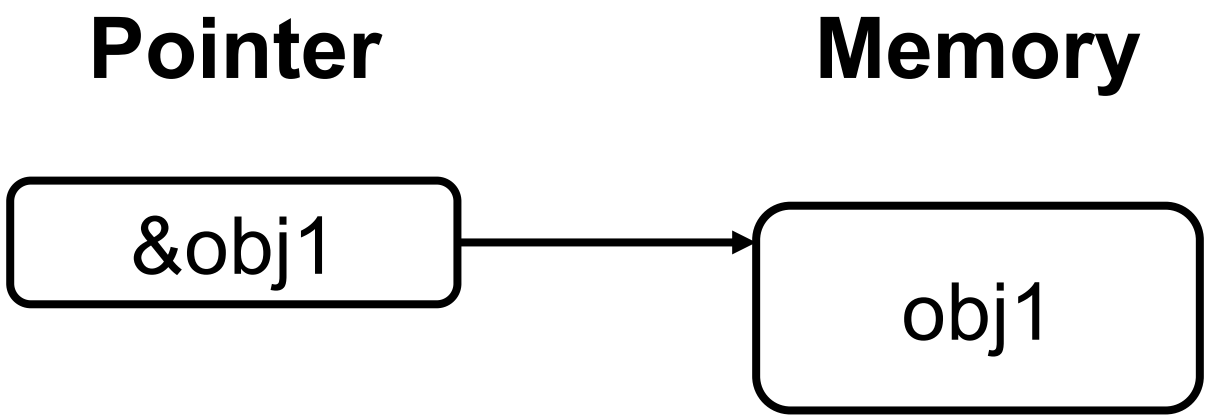
regreSSHion: Remote Unauthenticated Code Execution Vulnerability in OpenSSH server

BLASTPASS (2023)

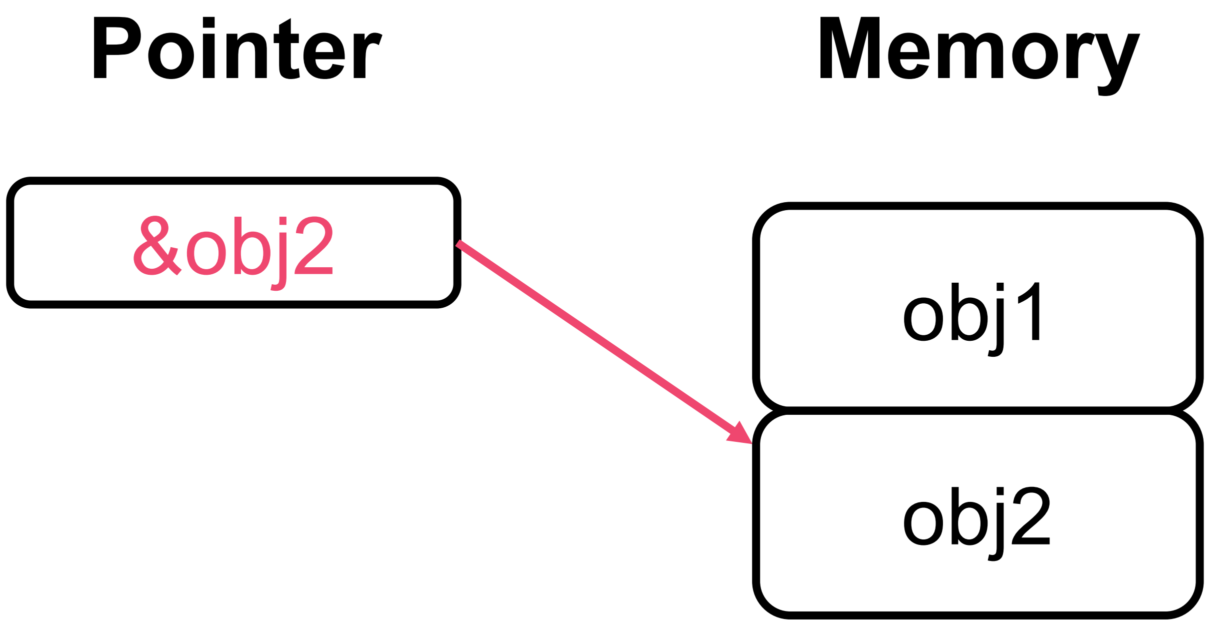
NSO Group iPhone Zero-Click, Zero-Day Exploit Captured in the Wild

What is Memory Corruption?

Valid Access

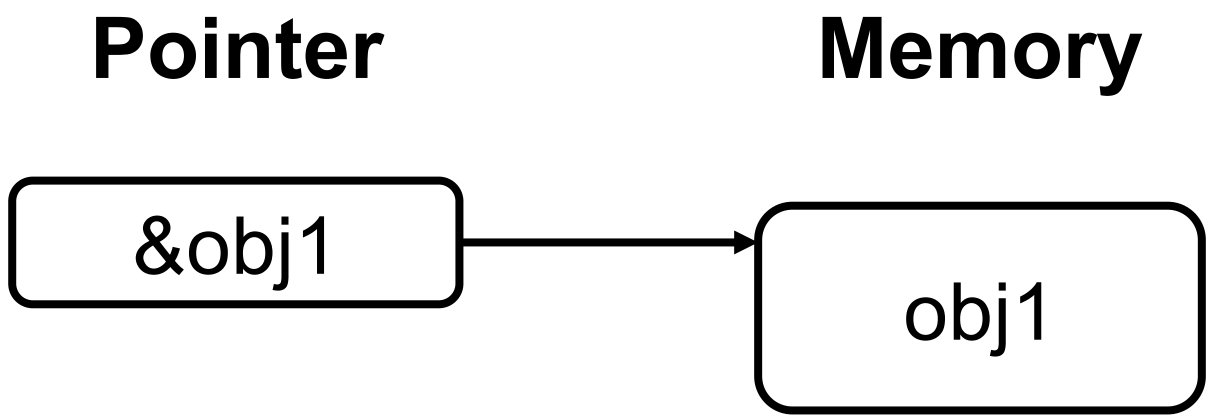


Invalid Access (Out-of-bounds)



What is Memory Corruption?

Valid Access



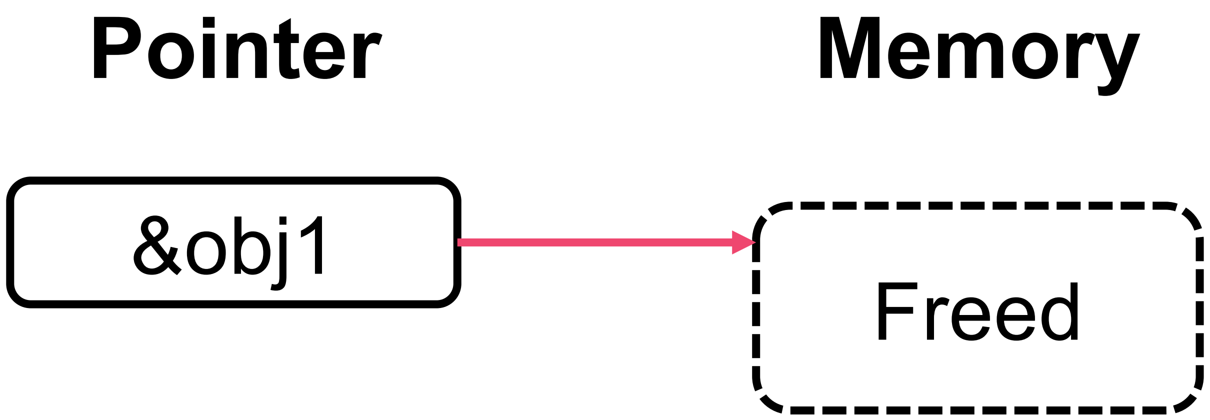
Pointer

Memory

&obj1

obj1

Invalid Access (Use-after-free)



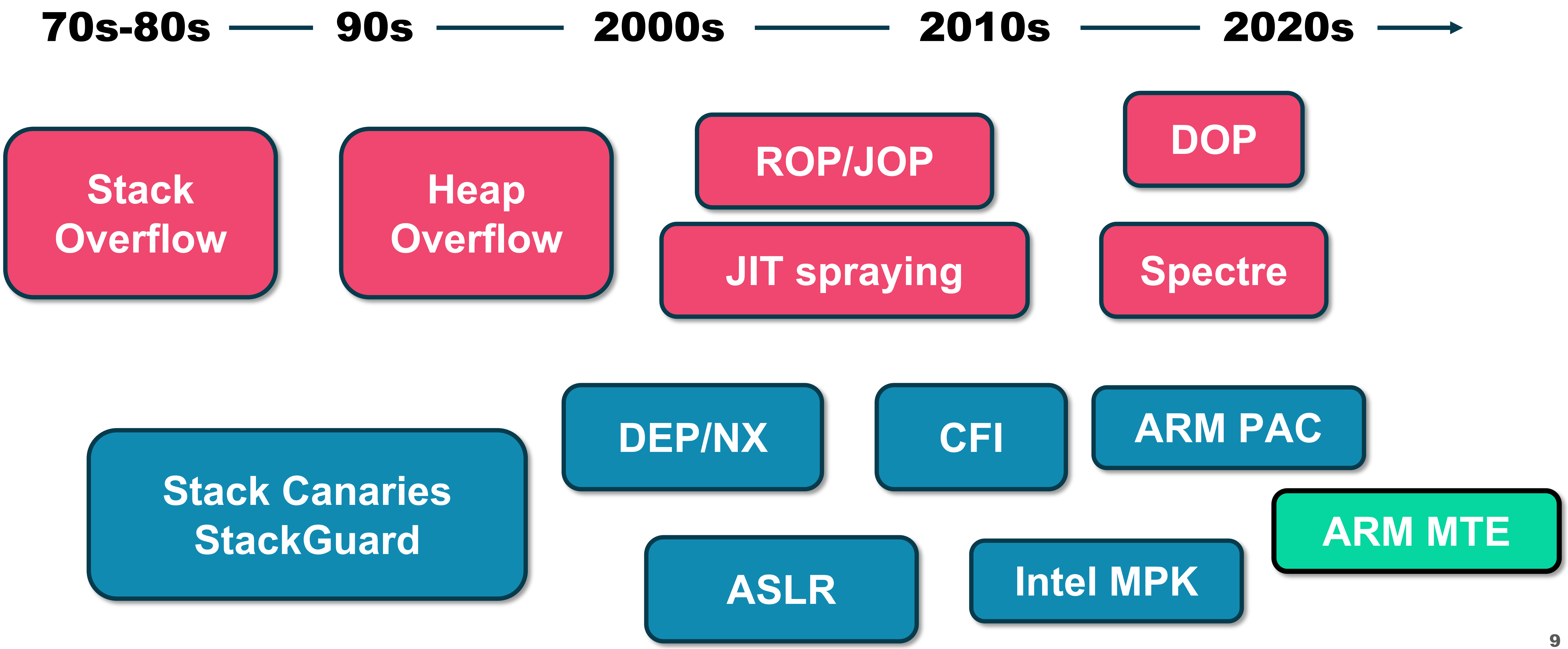
Pointer

Memory

&obj1

Freed

Attack and Defense Techniques



70s-80s → 90s → 2000s → 2010s → 2020s →

Stack Overflow

Heap Overflow

ROP/JOP

JIT spraying

DOP

Spectre

Stack Canaries StackGuard

DEP/NX

ASLR

CFI

Intel MPK

ARM PAC

ARM MTE

Google Pixel 8 / 8 pro — First MTE hardware released in Sep. 2023

*“**MTE** being one **key** feature that is delivering **secure mobile experiences**”*
- Arm (Feb 2023)

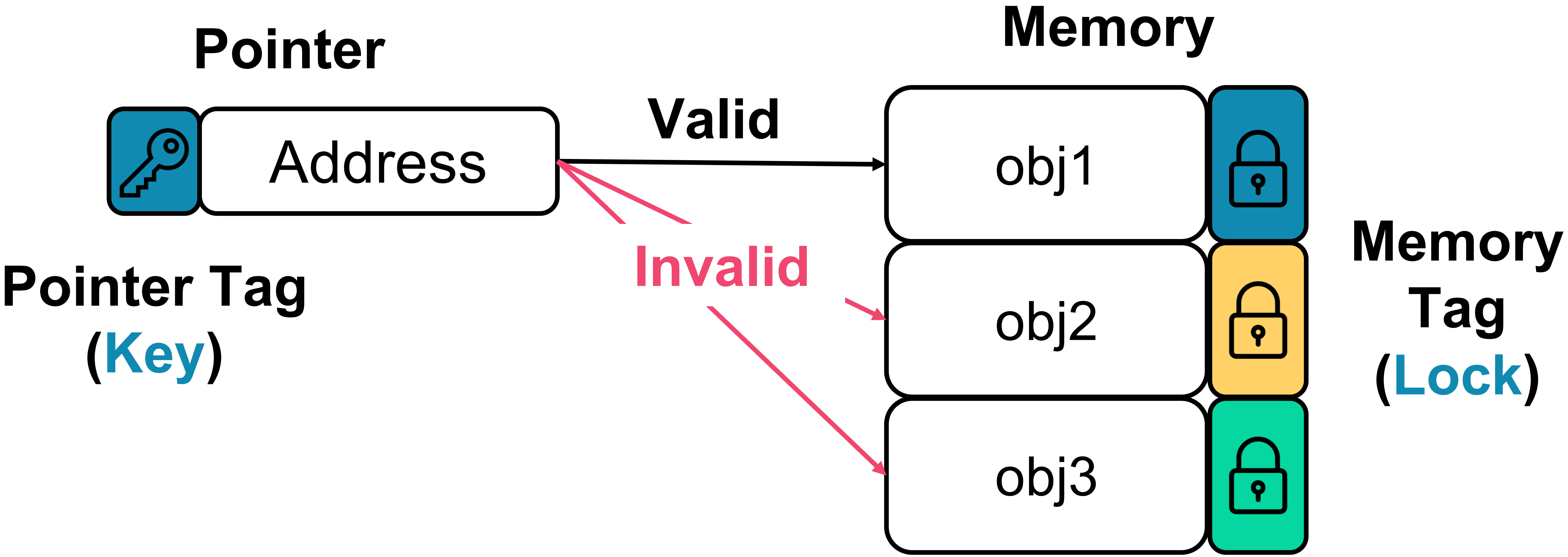
*“**MTE** is still by far the most promising path forward for improving C/C++ software security”*
- Google Project Zero (Aug 2023)

*“**Memory tagging** has the potential to provide good value both for **discovering vulnerabilities** and as **a mitigation for vulnerabilities**”*
- Microsoft (Mar 2020)

Why is MTE so Special?

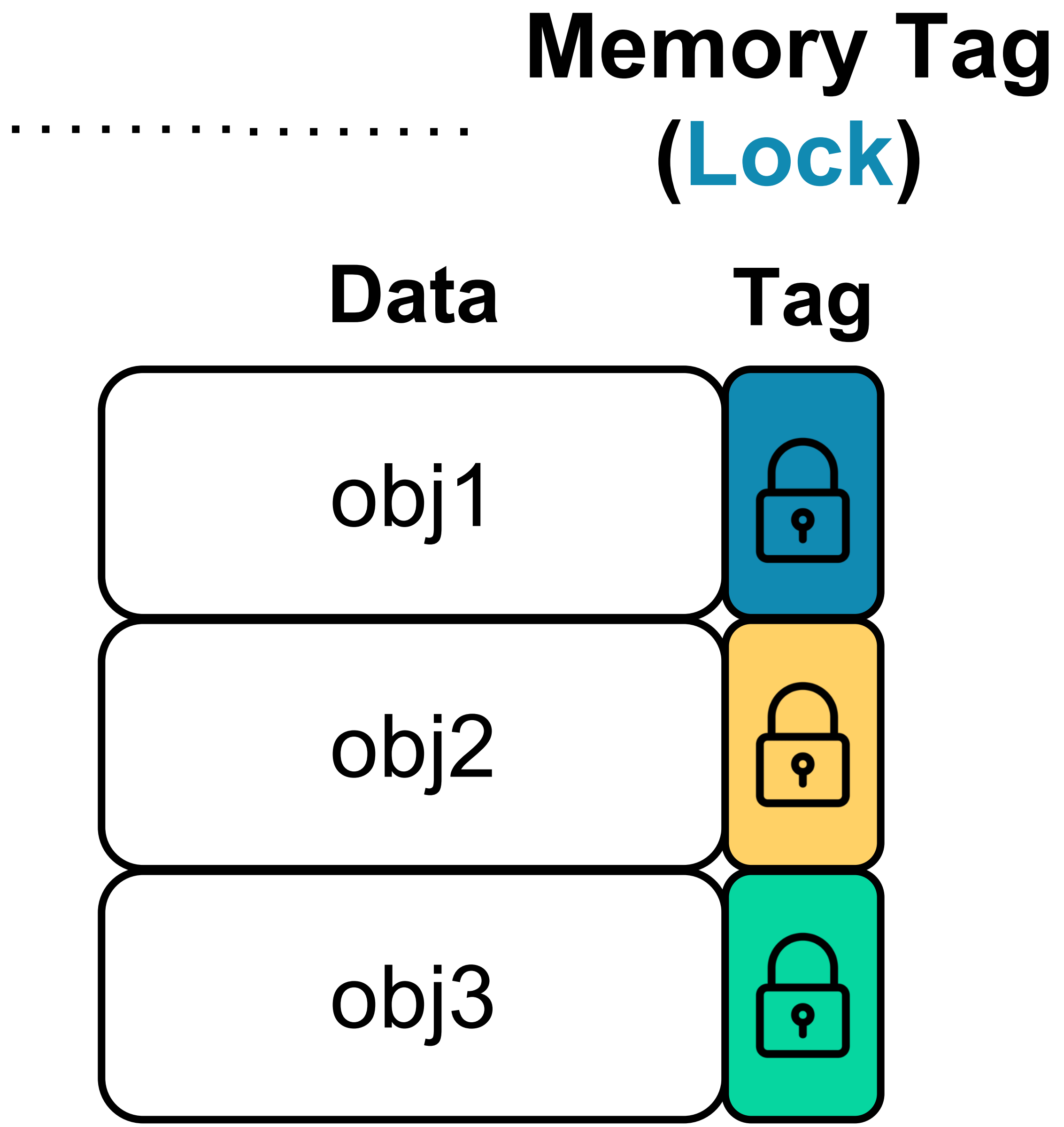
Hardware-based
Memory Corruption Detection
Fast and Compatible

ARM Memory Tagging Extensions

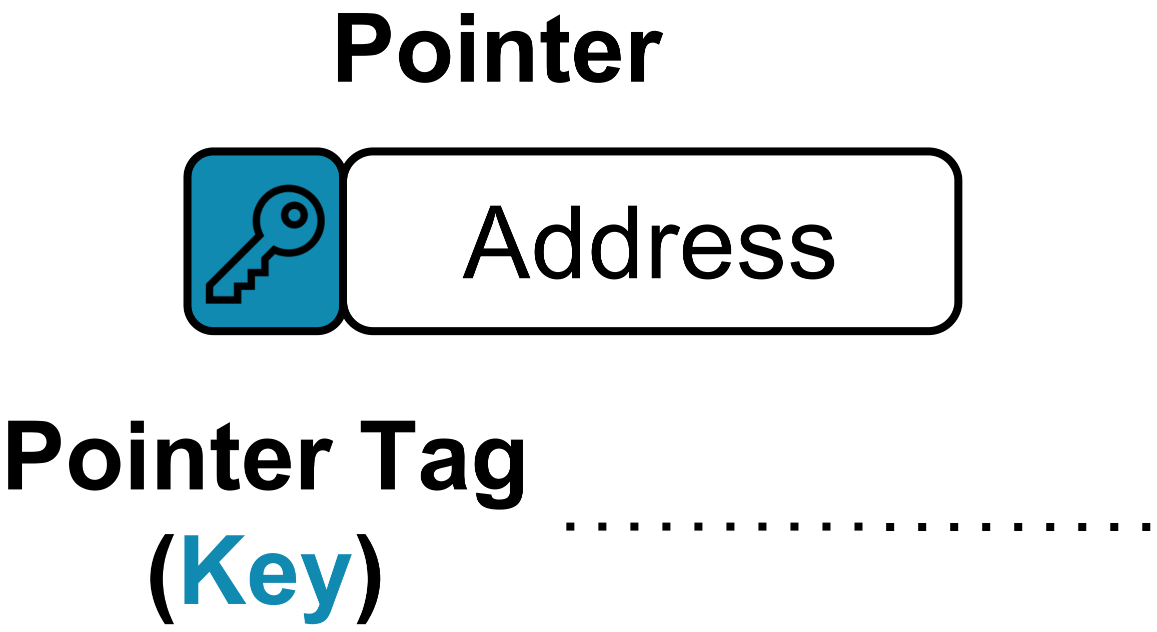


(1) Memory Tag

Dedicated memory region stores a 4-bit tag per 16-byte data



(2) Pointer Tag



Pointer

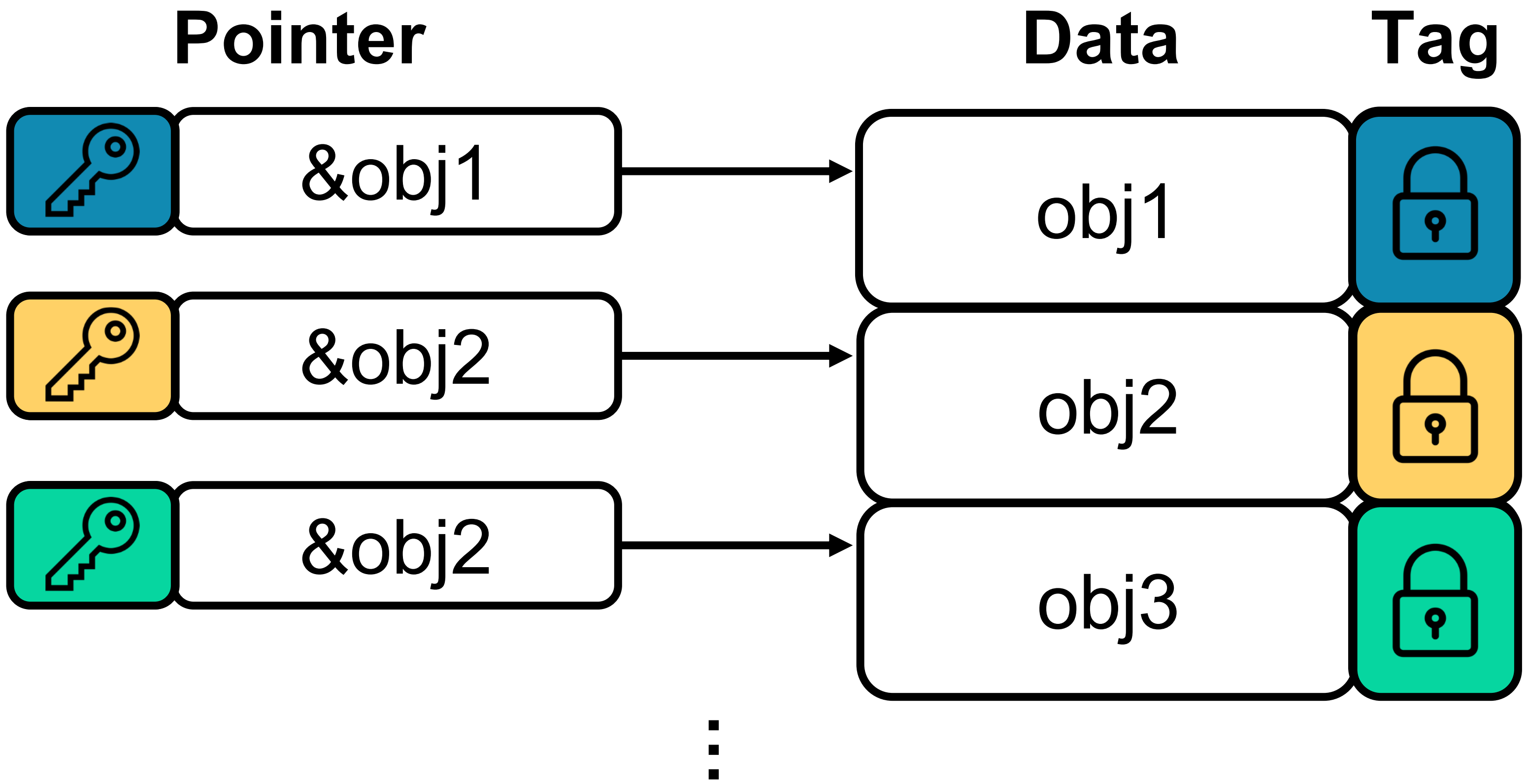
Address

Pointer Tag
(Key)

A pointer stores a 4-bit tag in its unused space

(3) Tag Allocation

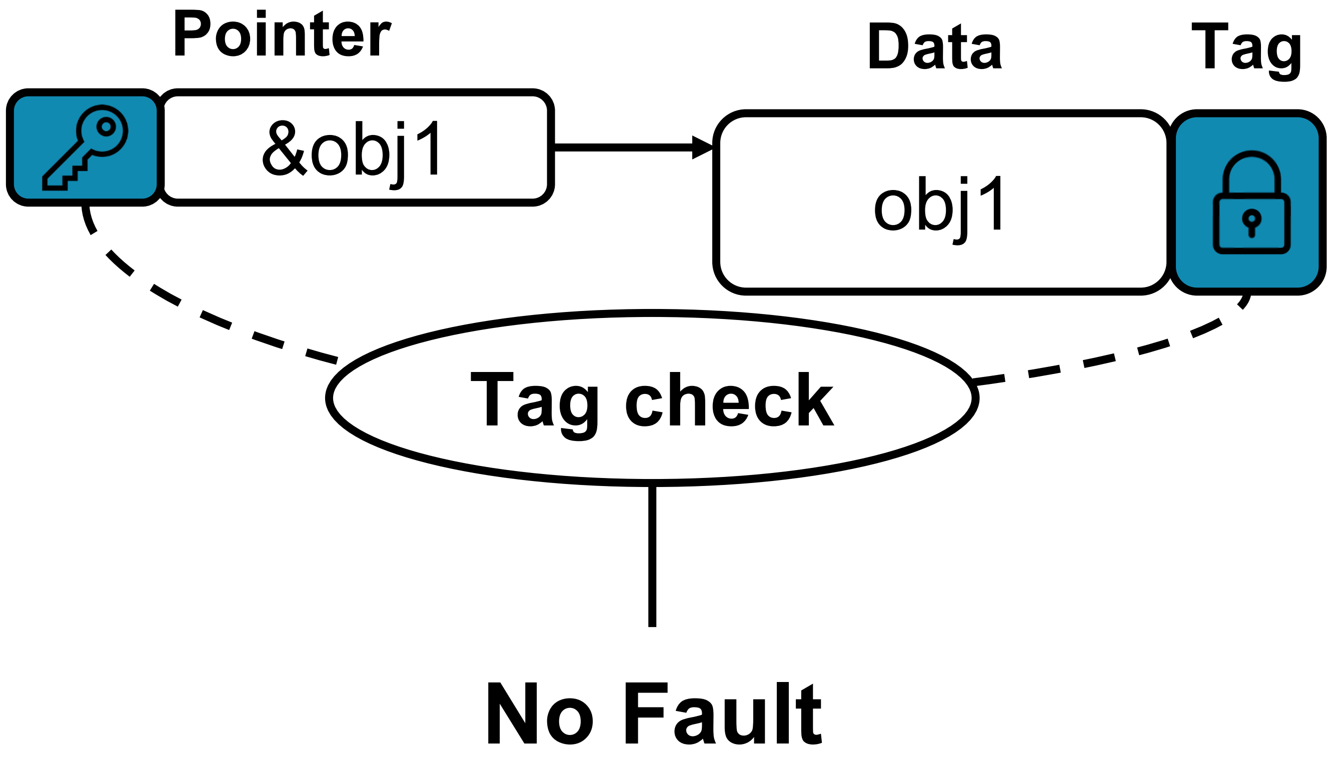
New instructions to create a random tag and load/store memory tags



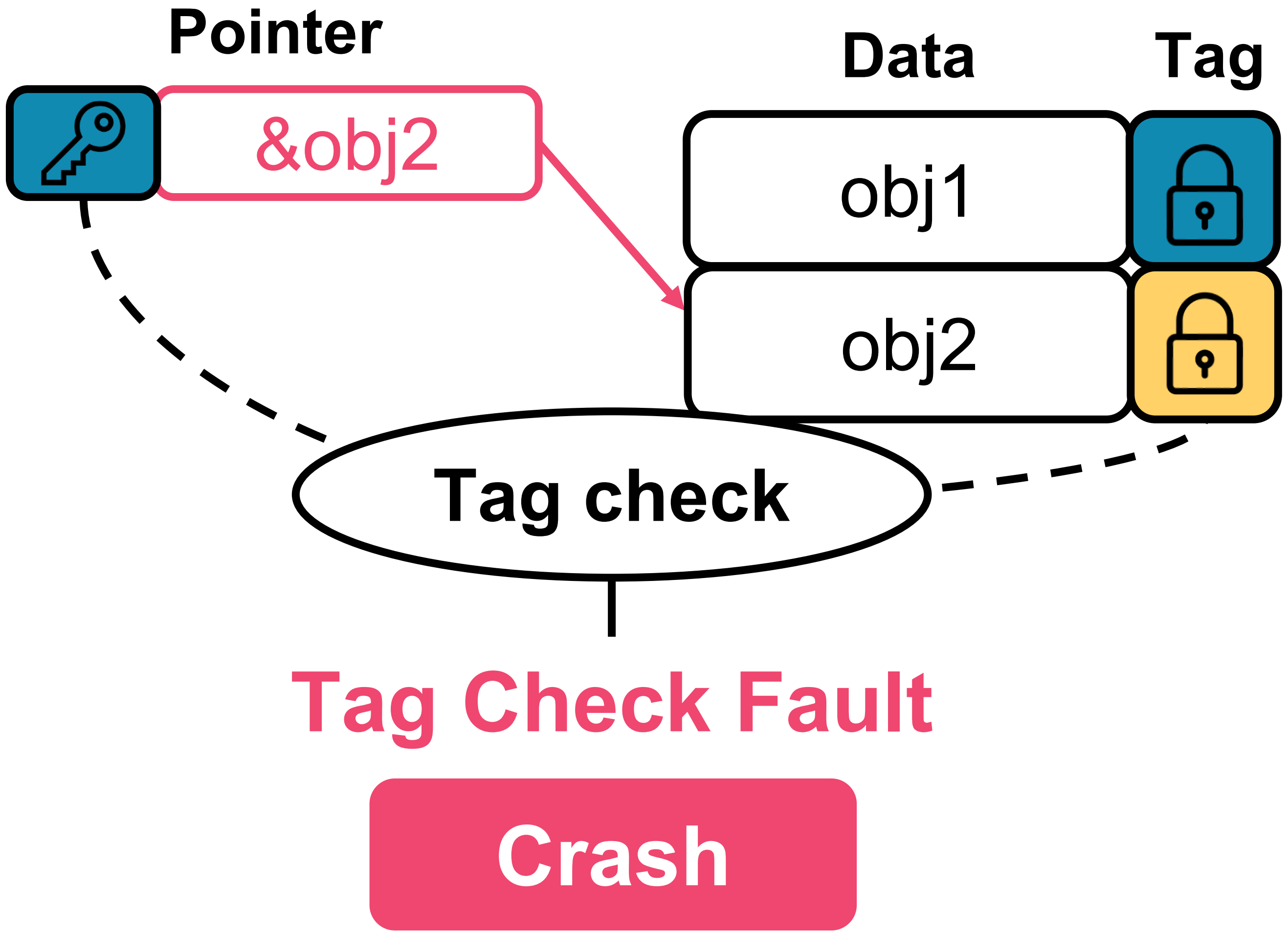
(4) Tag Check

Transparently done by hardware

Valid memory access



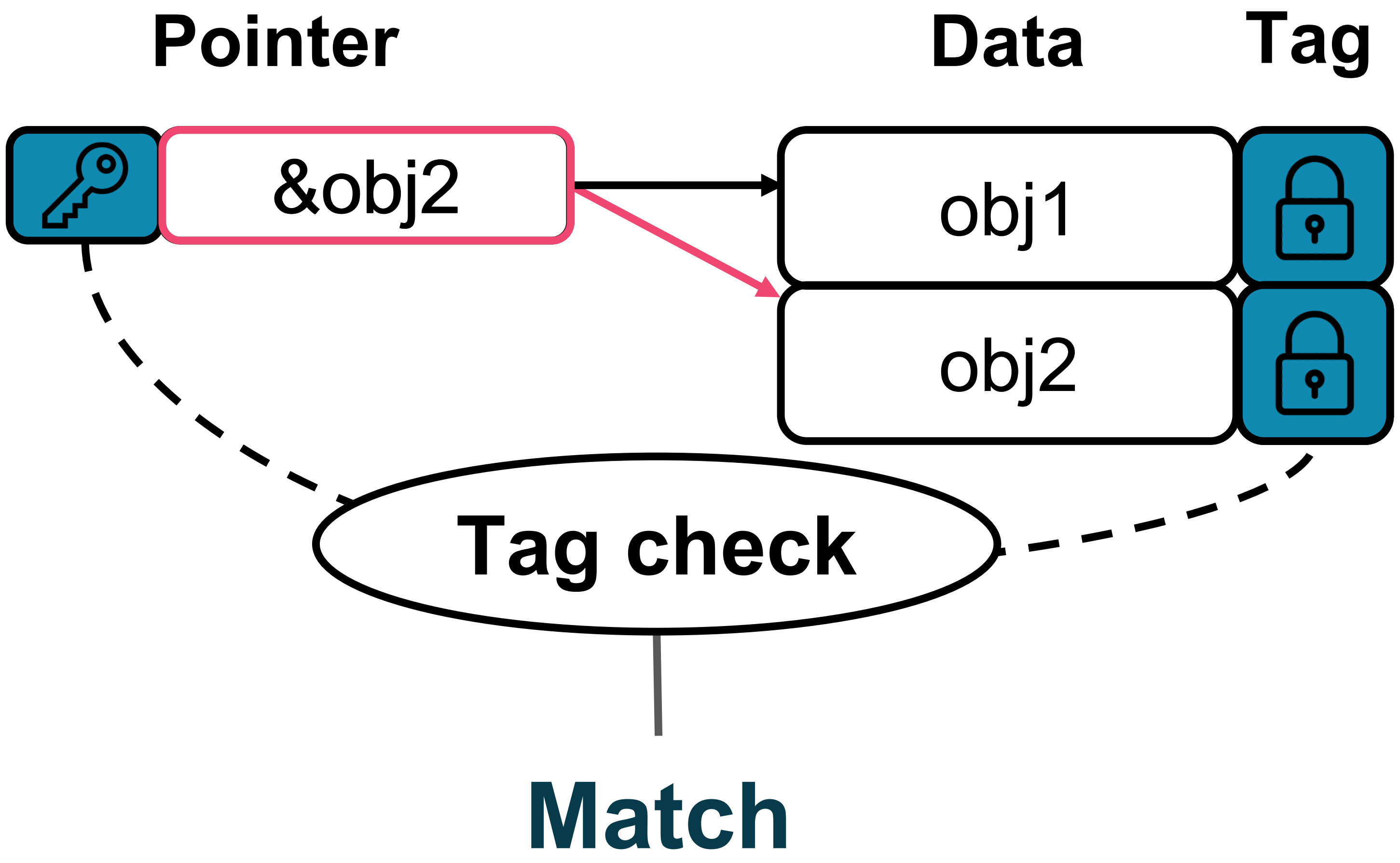
Invalid memory access



How to Bypass MTE?

(1) Tag Collision (16 possible tags)

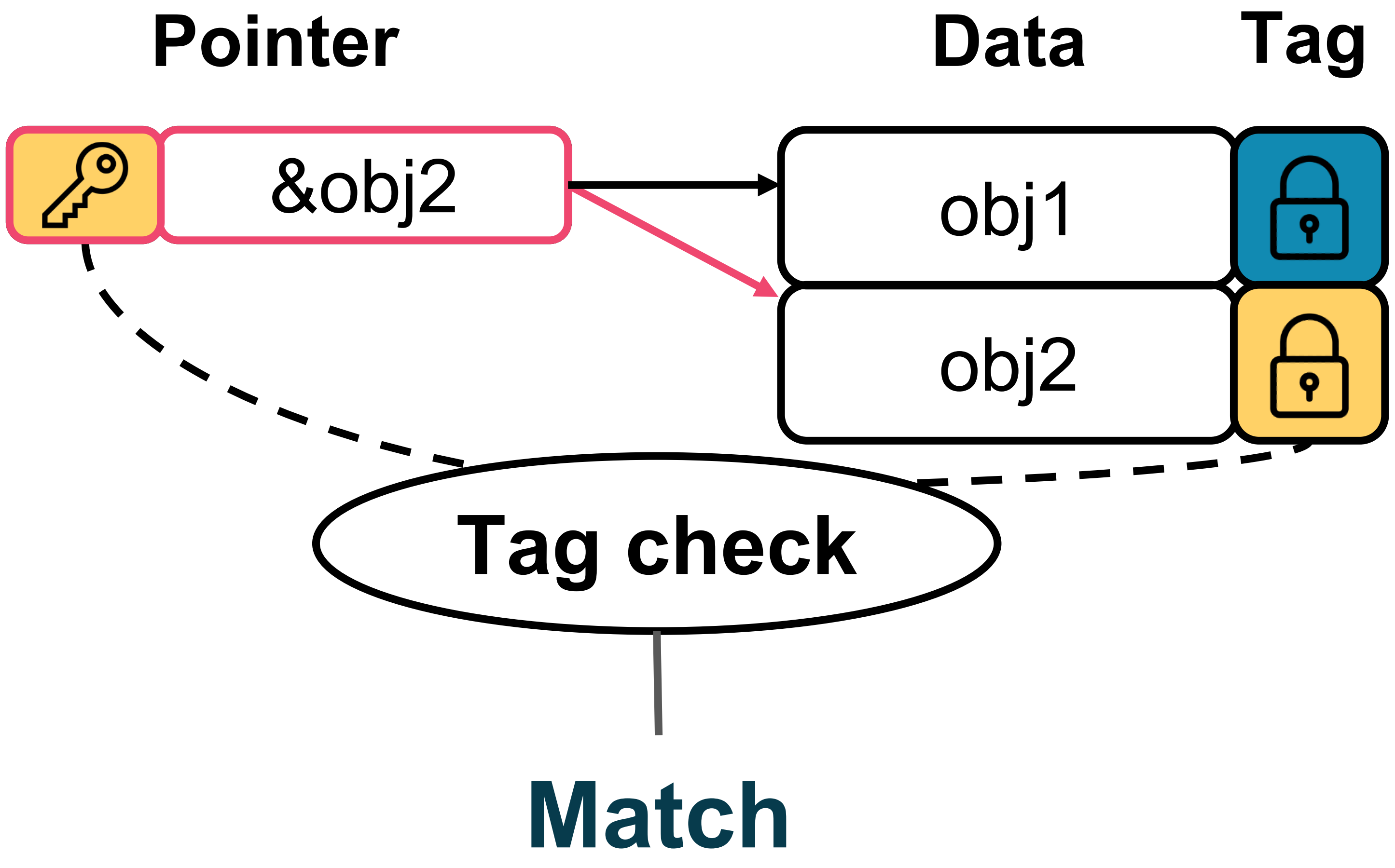
Wait until the pointer tag matches the target memory tag



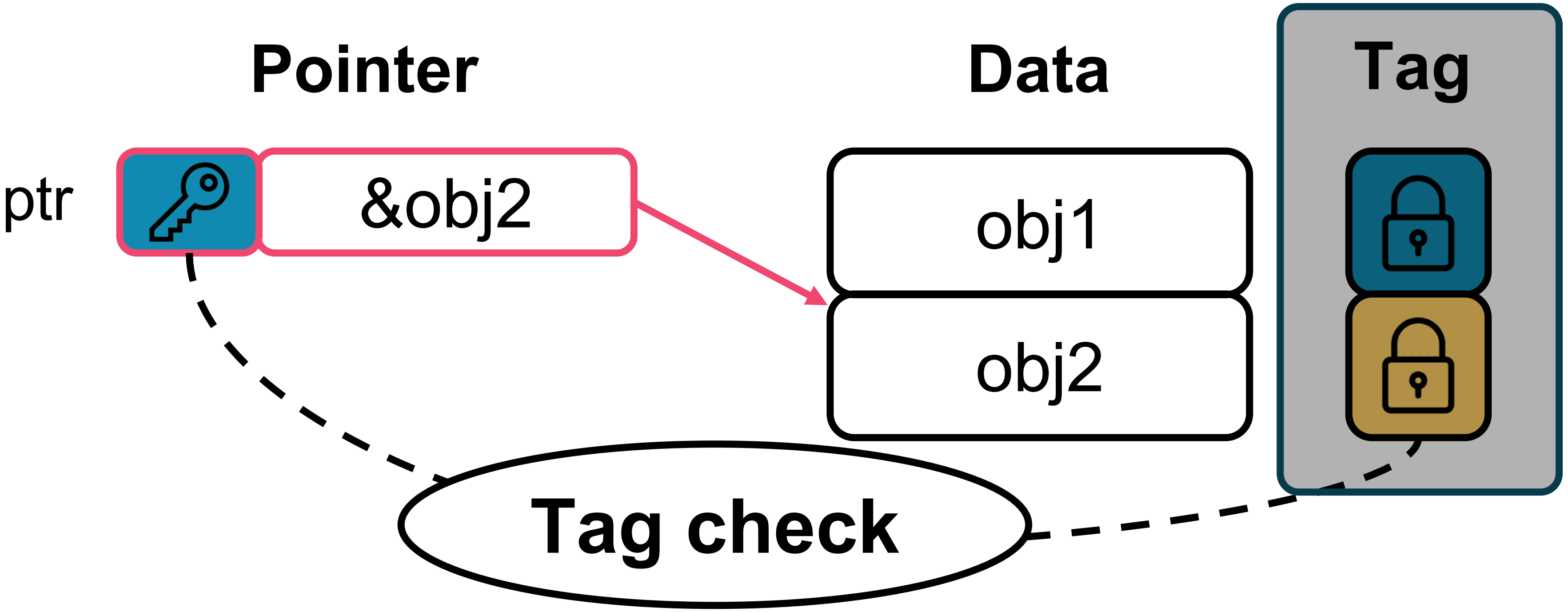
How to Bypass MTE?

(2) Pointer Tag Corruption

Corrupt the pointer tag to the target memory tag



Challenge: Random Tags



Match ⇒ ***Attack Succeeds 1/16 (6%)***

Mismatch

Crash ⇒ ***Attack Fails 15/16 (94%)***

MTE Bypass Requirement

A Reliable way to leak MTE tag of any address

Approach

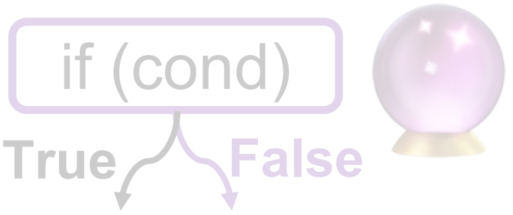
- Leak tag check result from Cache Side-channel
- Exploit Speculative Execution to avoid crash

Roadmap

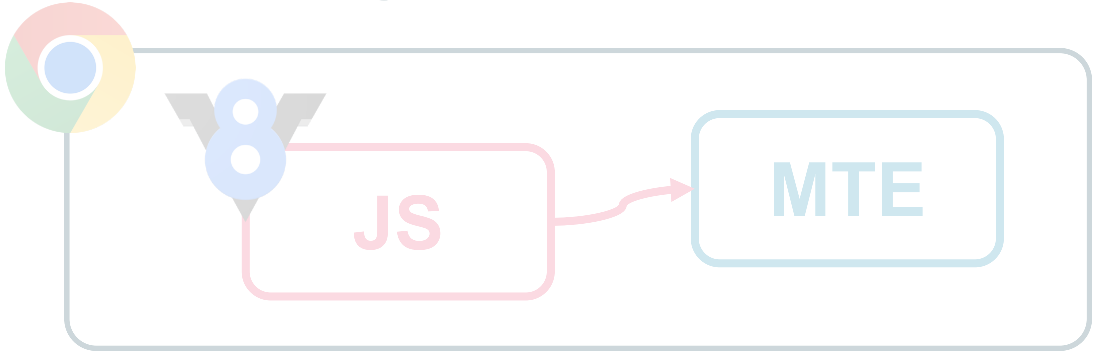
ARM Memory Tagging Extension

Cache Side-Channel

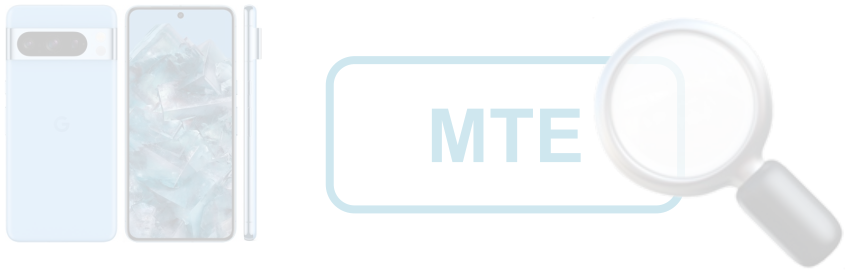
Speculative Execution



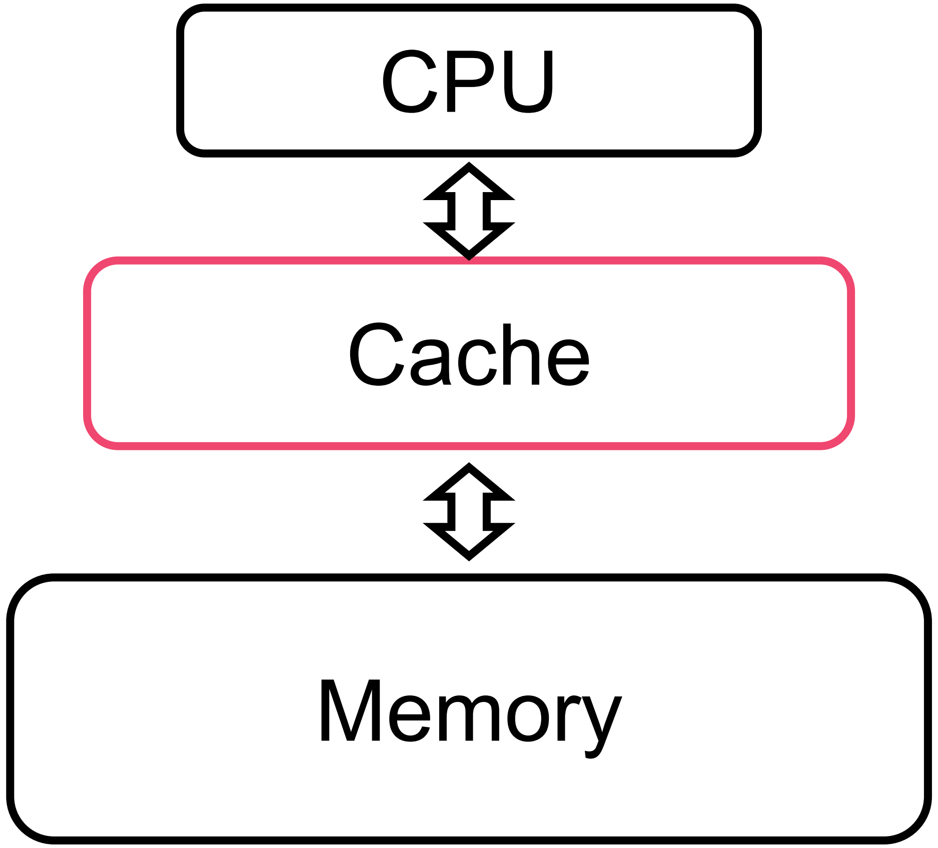
Real-world MTE Bypass Attack



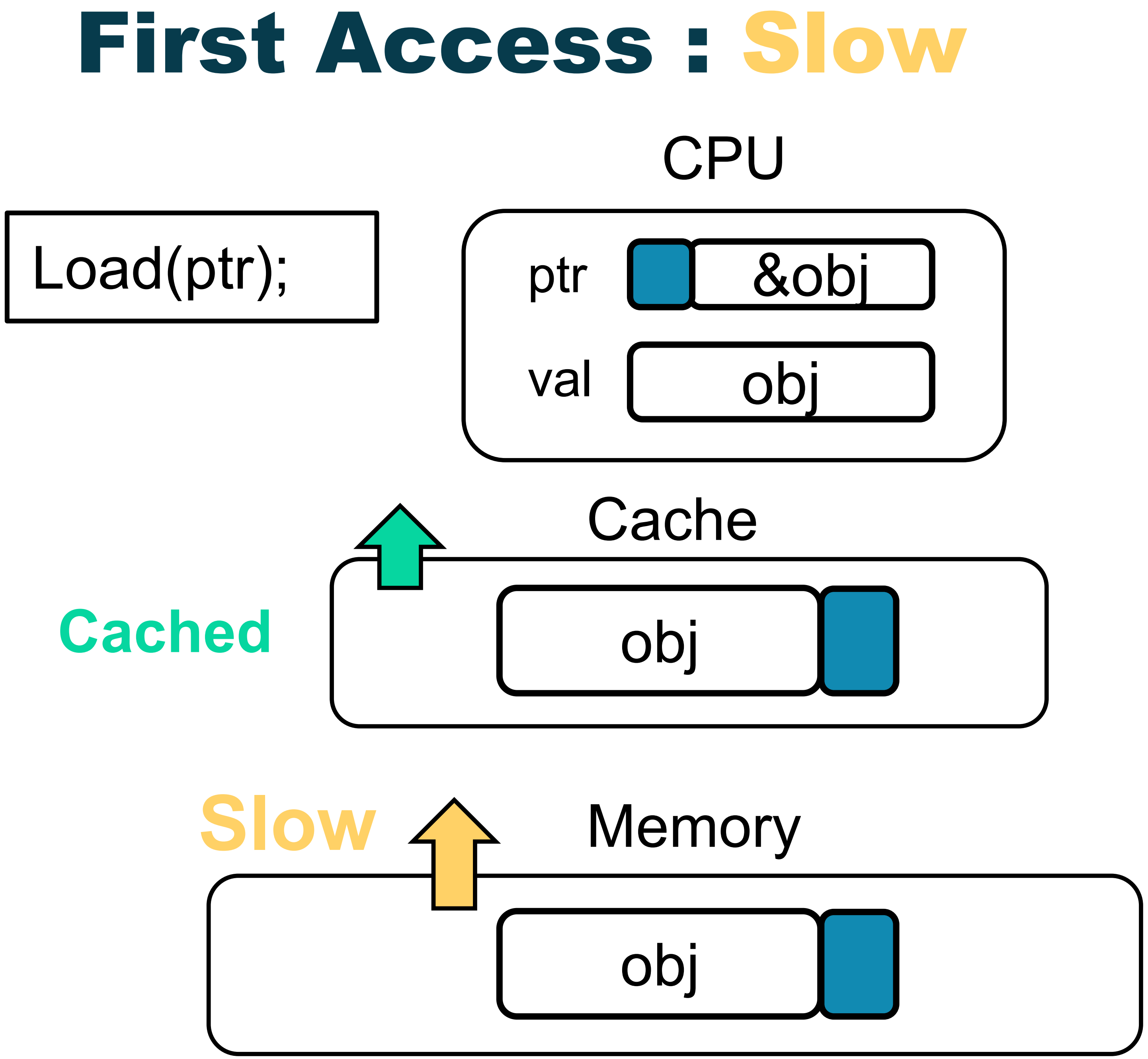
MTE Tag Leakage Side-Channel

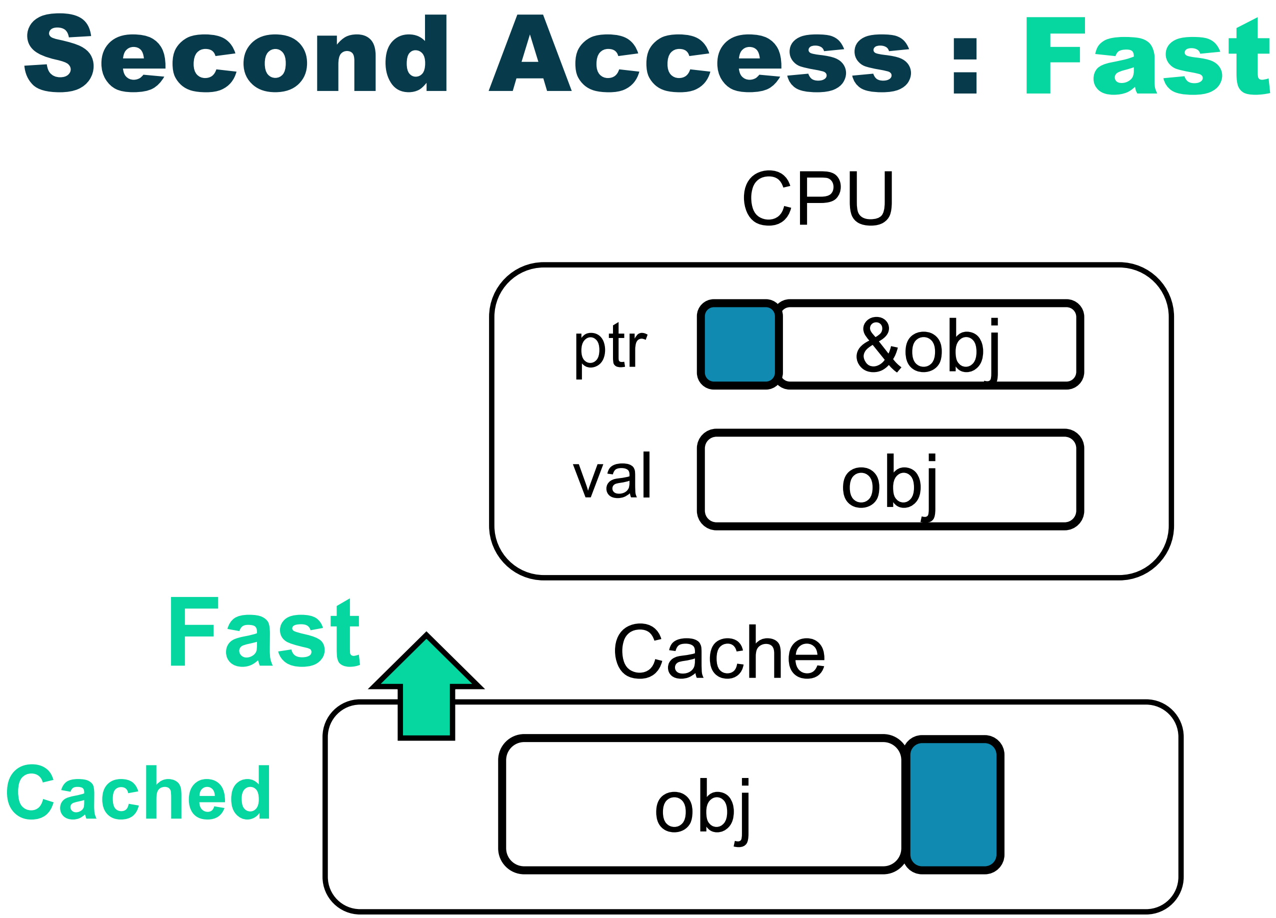


What is Cache?



What is Cache?



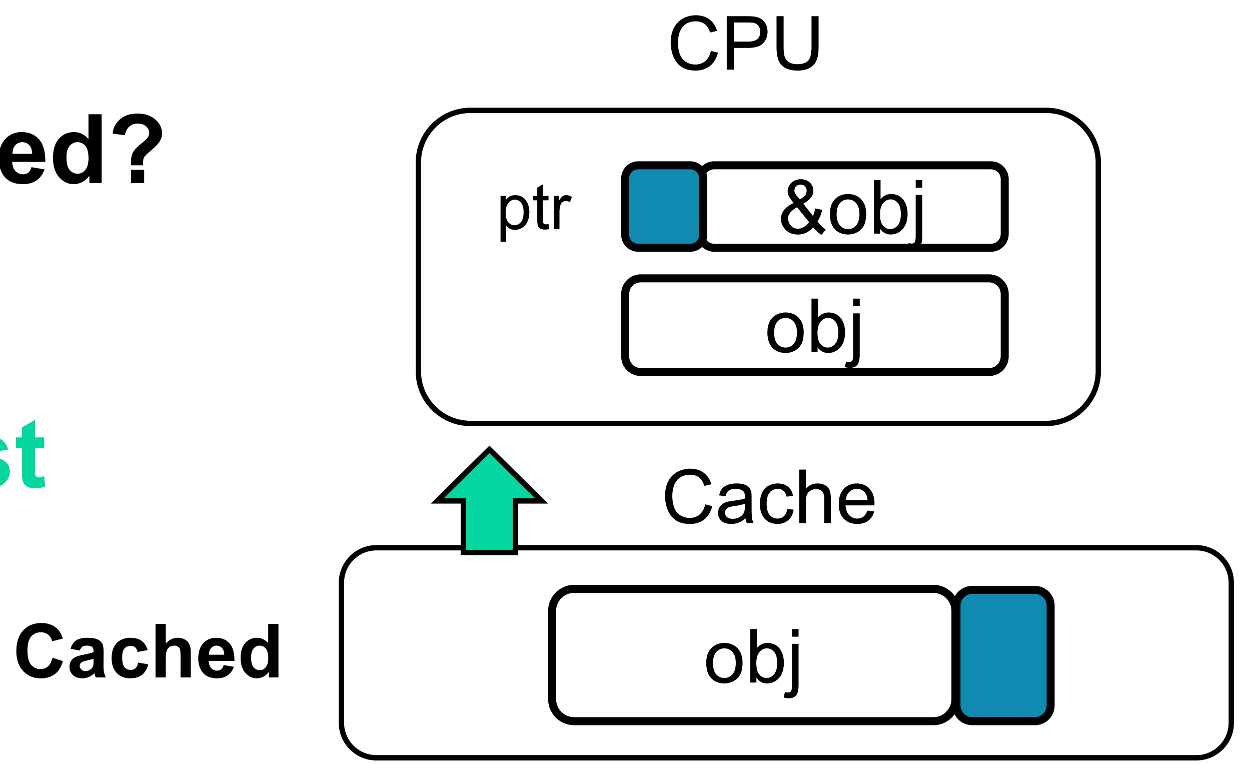


Cache Side-Channel

Q. Has obj been accessed?

```
Load(ptr);
```

Fast



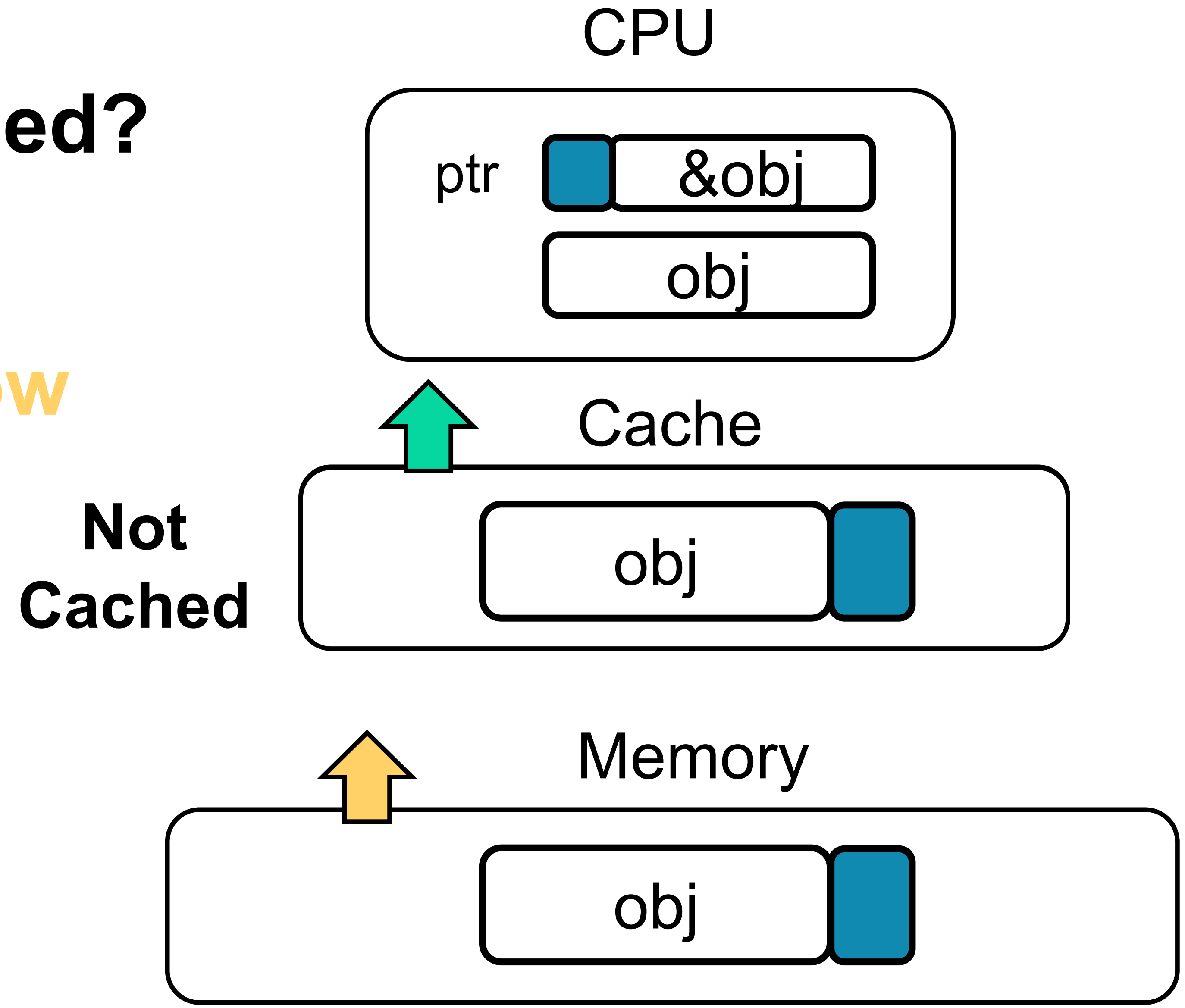
A. ptr has been accessed!

What is Cache Side-Channel?

Q. Has obj been accessed?

```
Load(ptr);
```

⌛ Slow



A. ptr has NOT been accessed!

Exploit cache side-channel

→ Leak whether an address is accessed

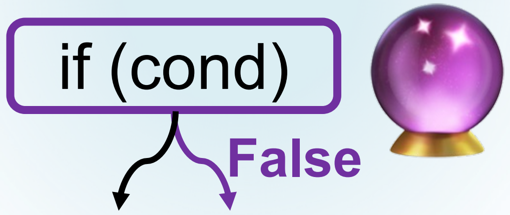
Roadmap

ARM Memory Tagging Extension

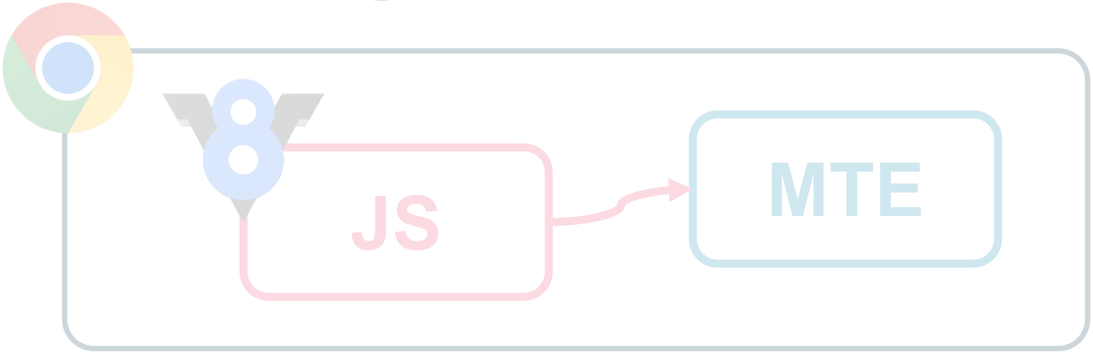
Cache Side-Channel

Cache ⌛

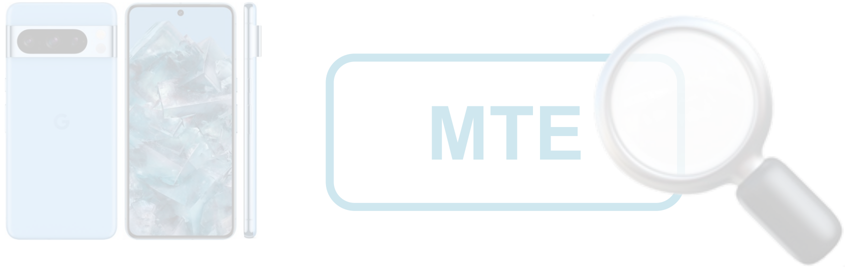
Speculative Execution



Real-world MTE Bypass Attack

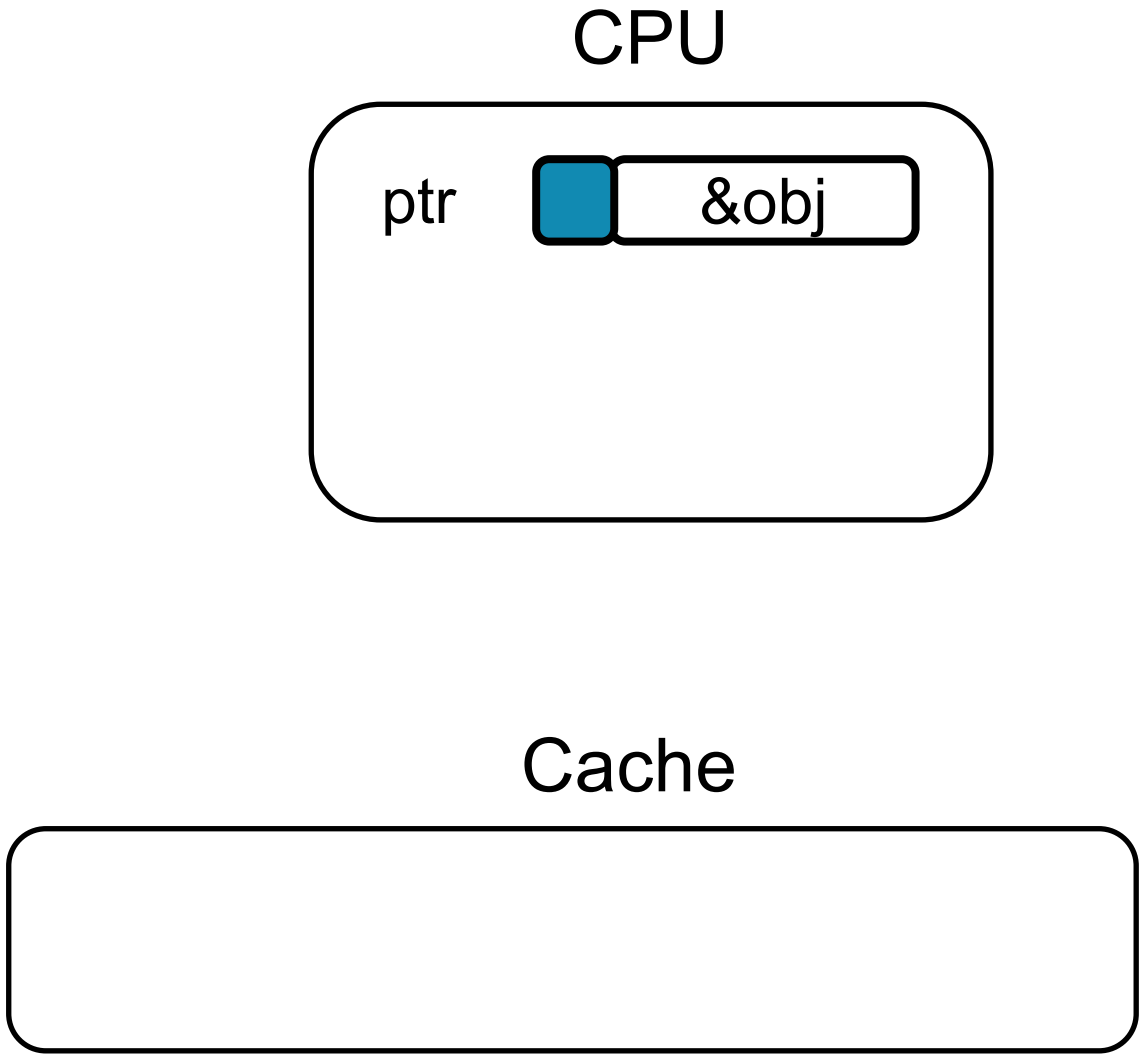


MTE Tag Leakage Side-Channel



What is Speculative Execution?

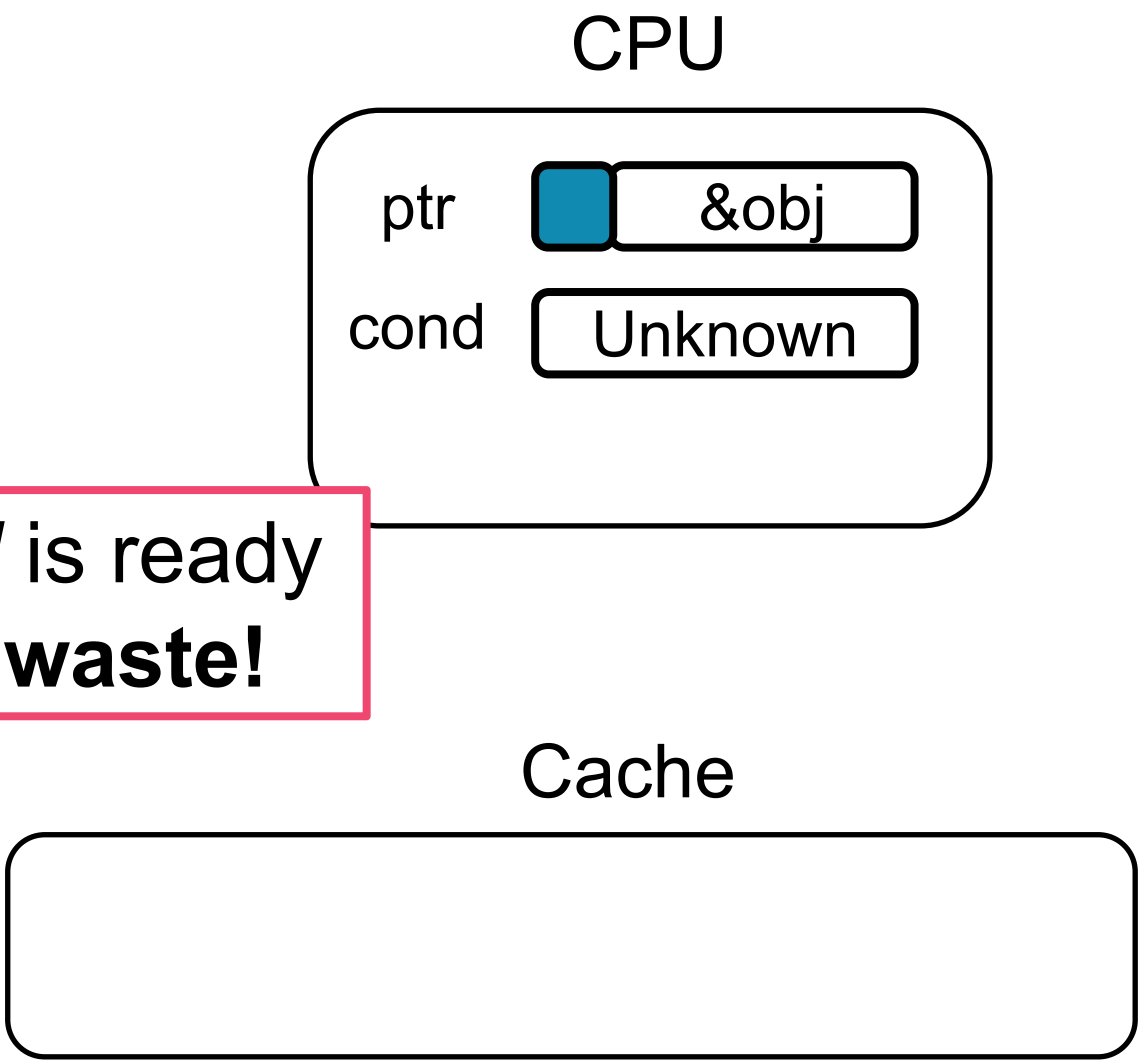
```
…
if (cond) {        ←

  v = Load(ptr);
  …
}
```



What is Speculative Execution?

```
…
if (cond) {  ←

  v = Load(ptr);
  …
}
```

Waiting until *cond* is ready
Huge resource waste!



CPU

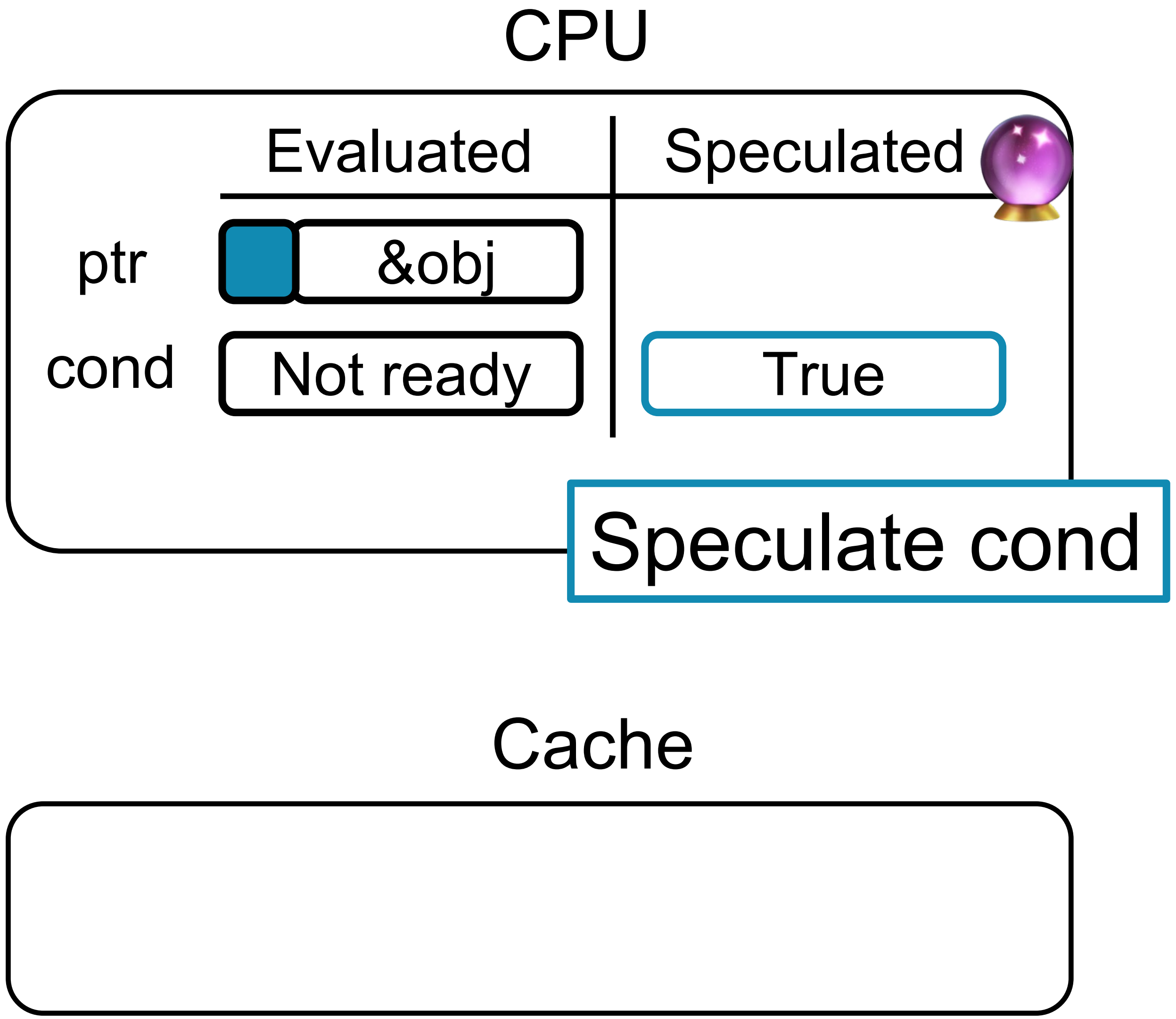
ptr &obj

cond Unknown

Cache

What is Speculative Execution?

```
…
if (cond) {        ←

  v = Load(ptr);
  …
}
```



CPU

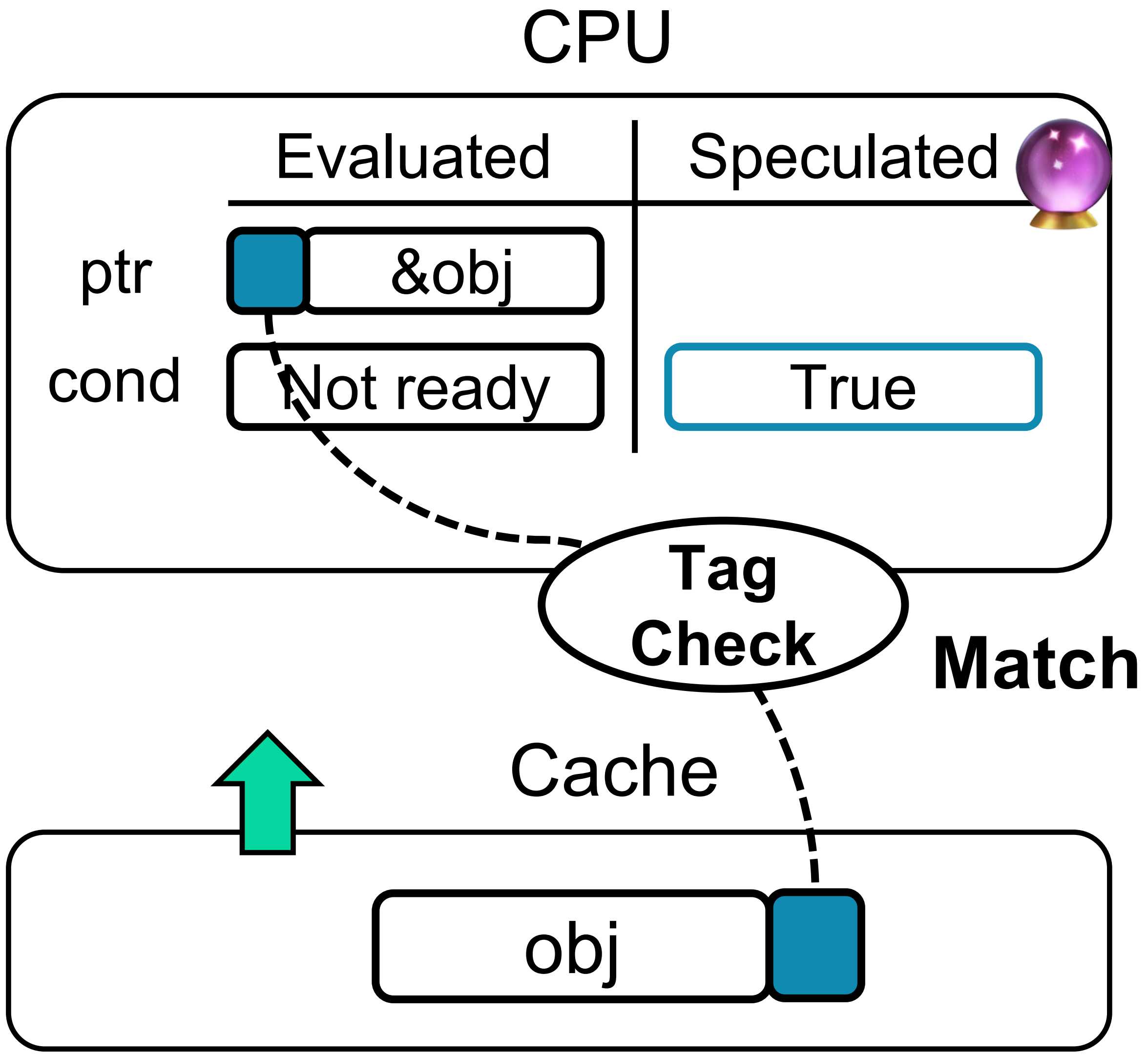
	Evaluated	Speculated
ptr	🟦 &obj	
cond	Not ready	True

Speculate cond

Cache

What is Speculative Execution?

```
…
if (cond) {

  v = Load(ptr); ←
  …
}
```



CPU

	Evaluated	Speculated
ptr	&obj	
cond	Not ready	True

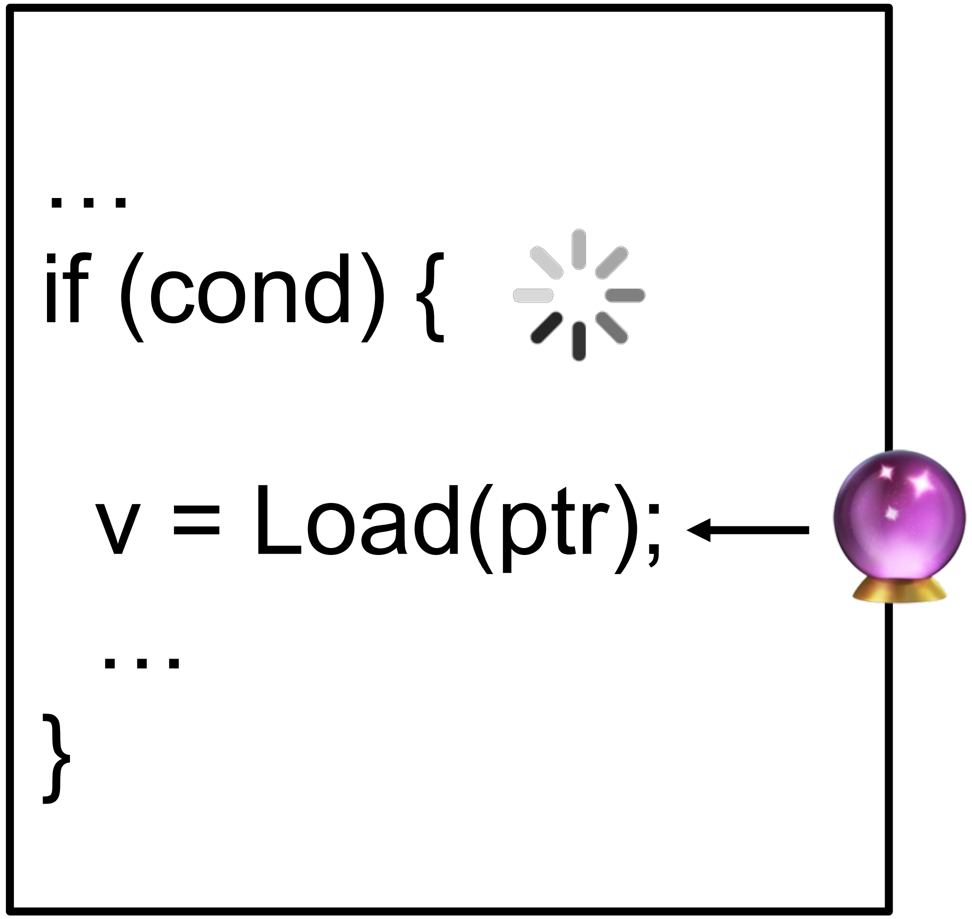
Tag Check

Match

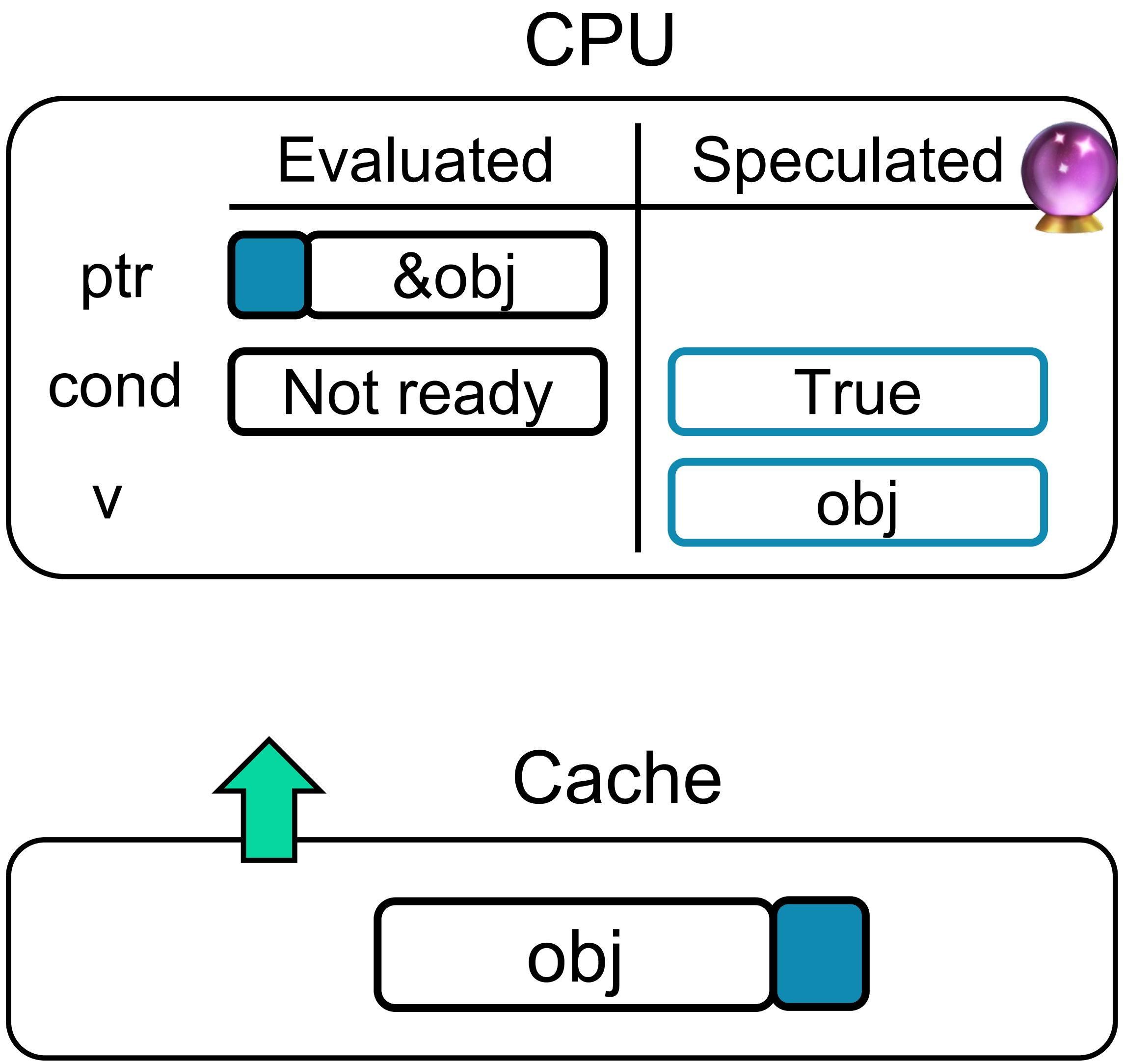
Cache

obj

What is Speculative Execution?



```
…
if (cond) {

  v = Load(ptr);
  …
}
```



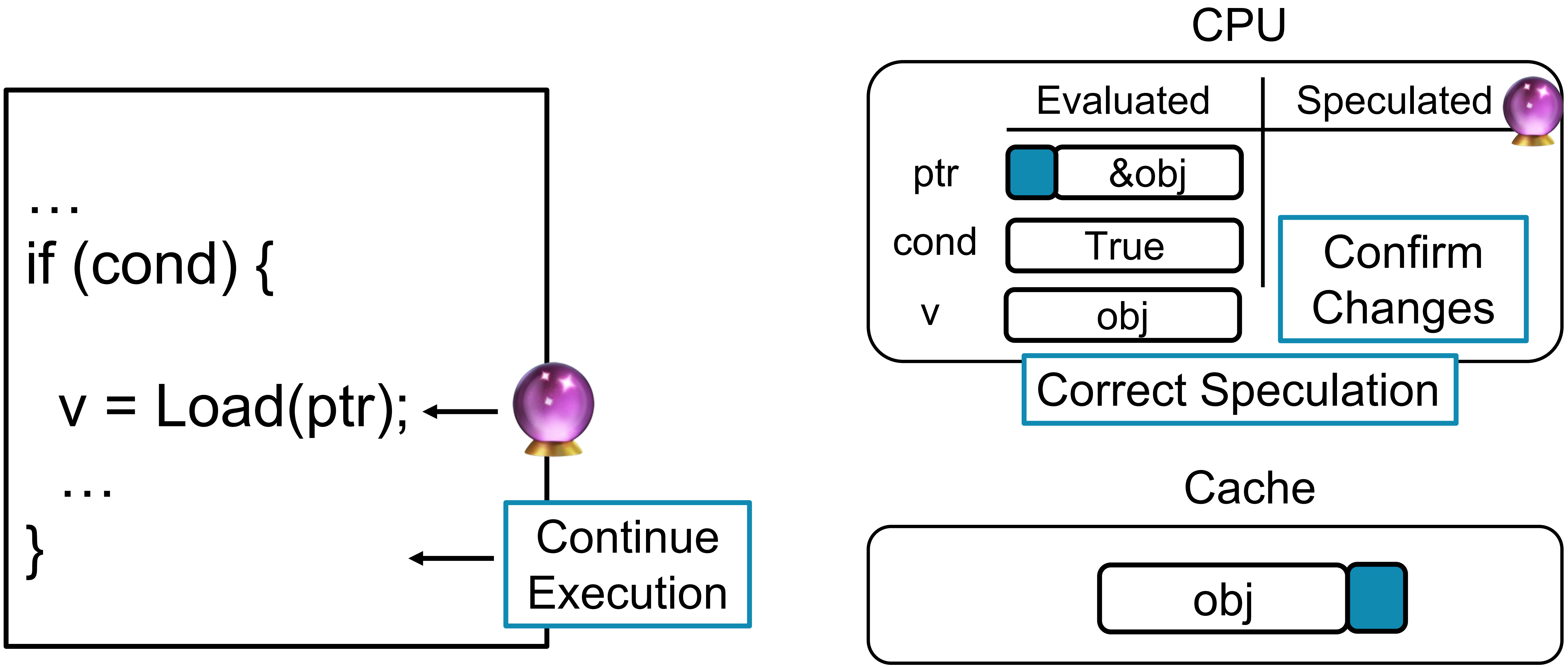
CPU

	Evaluated	Speculated
ptr	&obj	
cond	Not ready	True
v		obj

Cache

obj

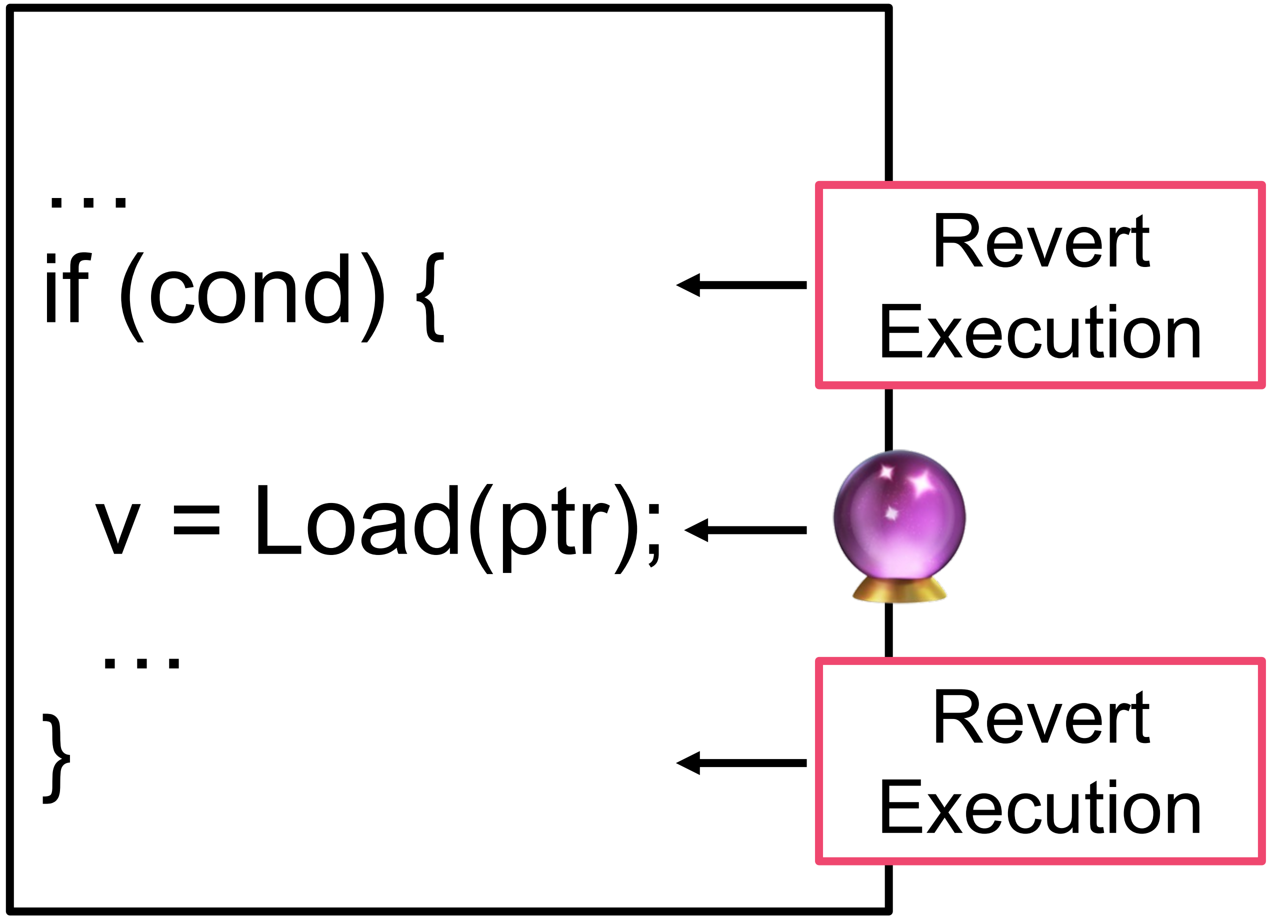
What is Speculative Execution?



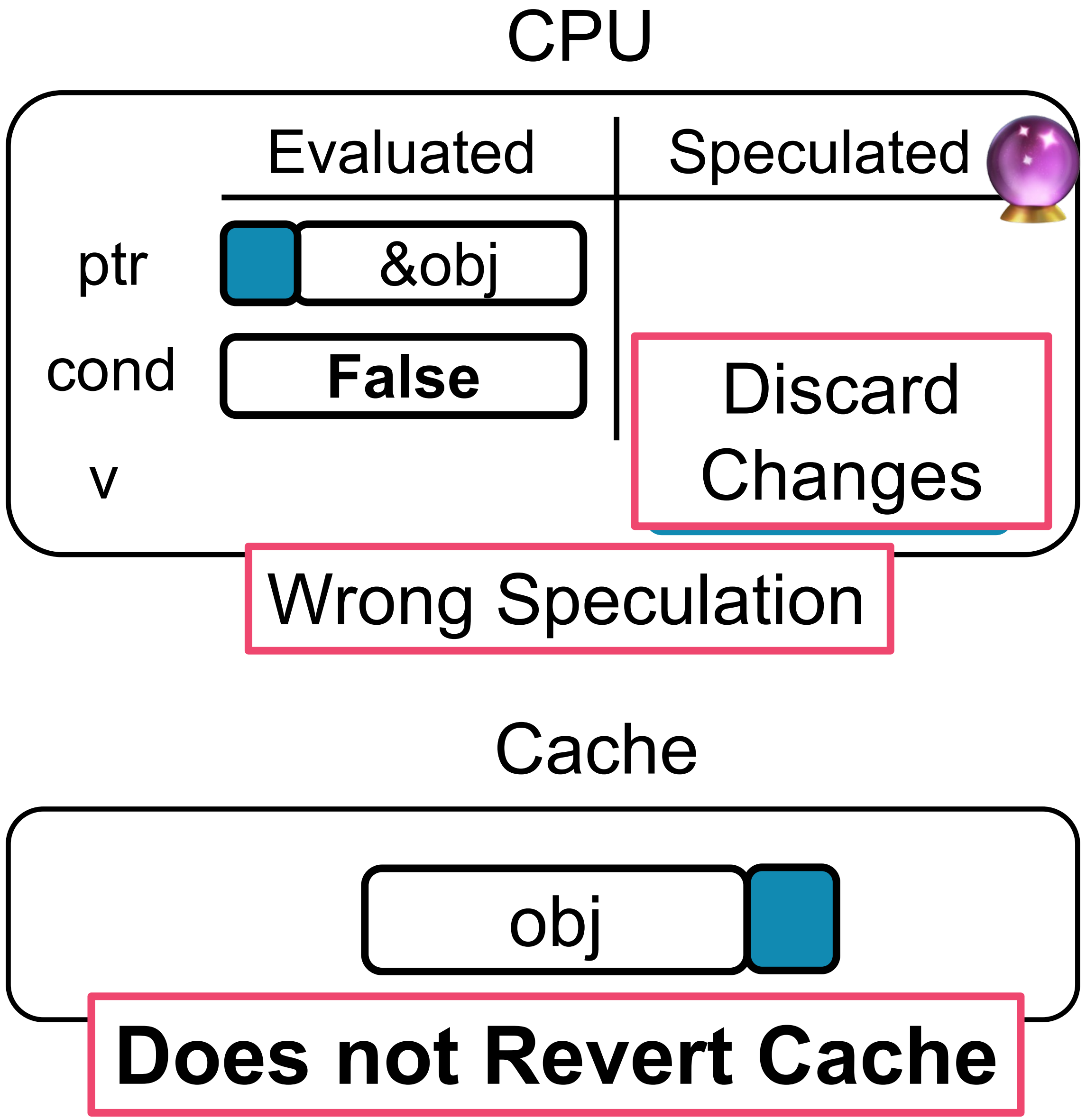
```
…
if (cond) {

  v = Load(ptr);
  …
}
```

	Evaluated	Speculated
ptr	&obj	
cond	True	
v	obj	

What is Speculative Execution?



```
…
if (cond) {

  v = Load(ptr);
  …
}
```



CPU

	Evaluated	Speculated
ptr	■ &obj	
cond	**False**	
v		

Discard Changes

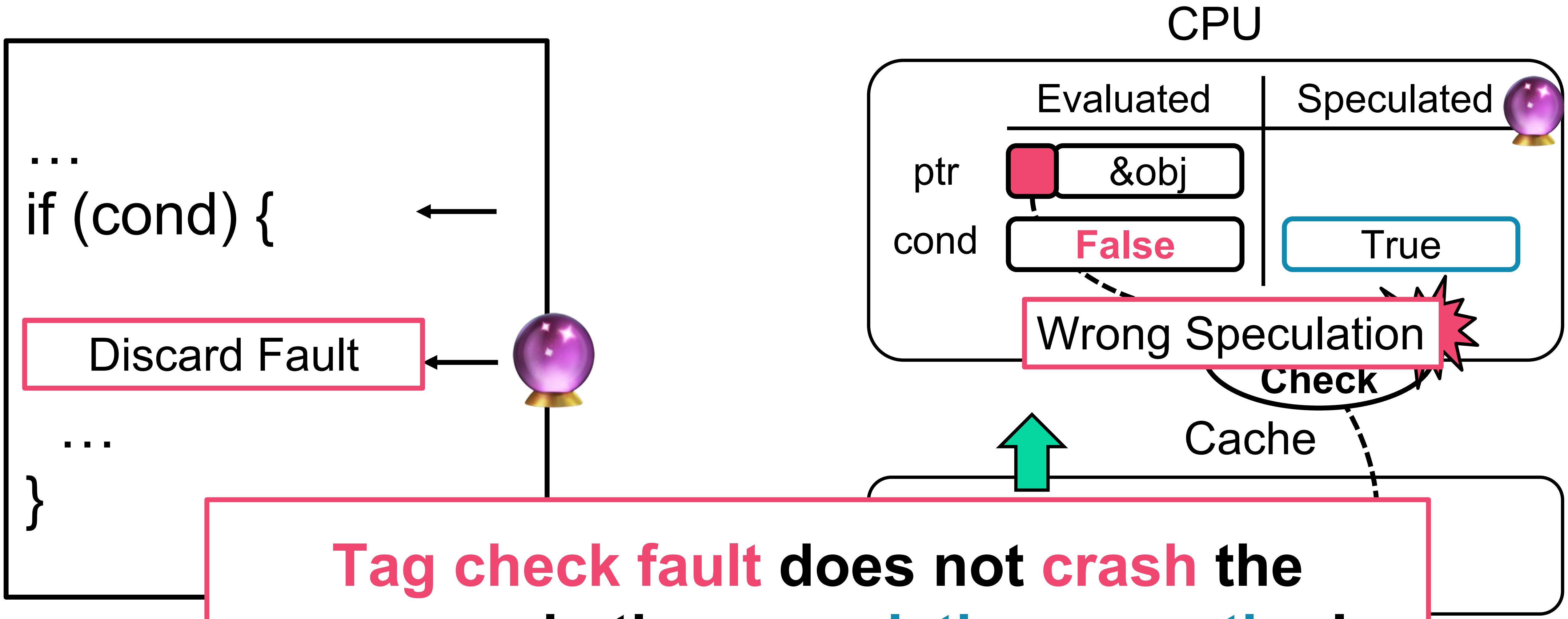
Wrong Speculation

Cache

obj ■

Does not Revert Cache

Tag check fault on Speculative Execution?



Tag check fault does not crash the program in the speculative execution!

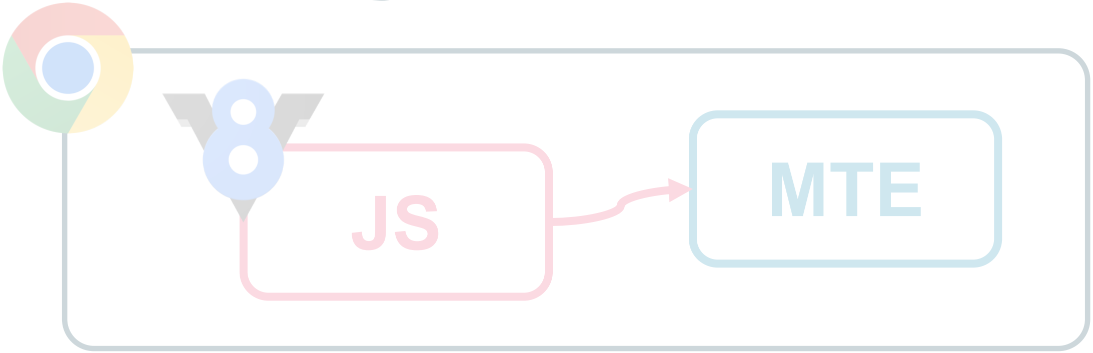
Exploit cache side-channel
→ Leak whether an address is accessed

Exploit speculative execution
→ Avoid crash on tag check fault

Roadmap

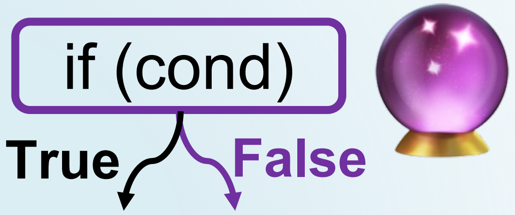
ARM Memory Tagging Extension

Real-world MTE Bypass Attack



Cache Side-Channel

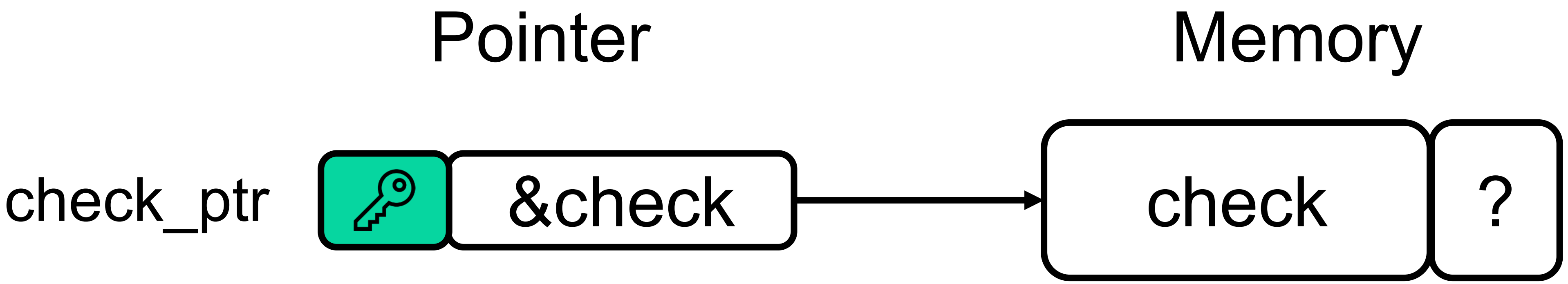
Speculative Execution



MTE Tag Leakage Side-Channel

MTE Side-channel attack

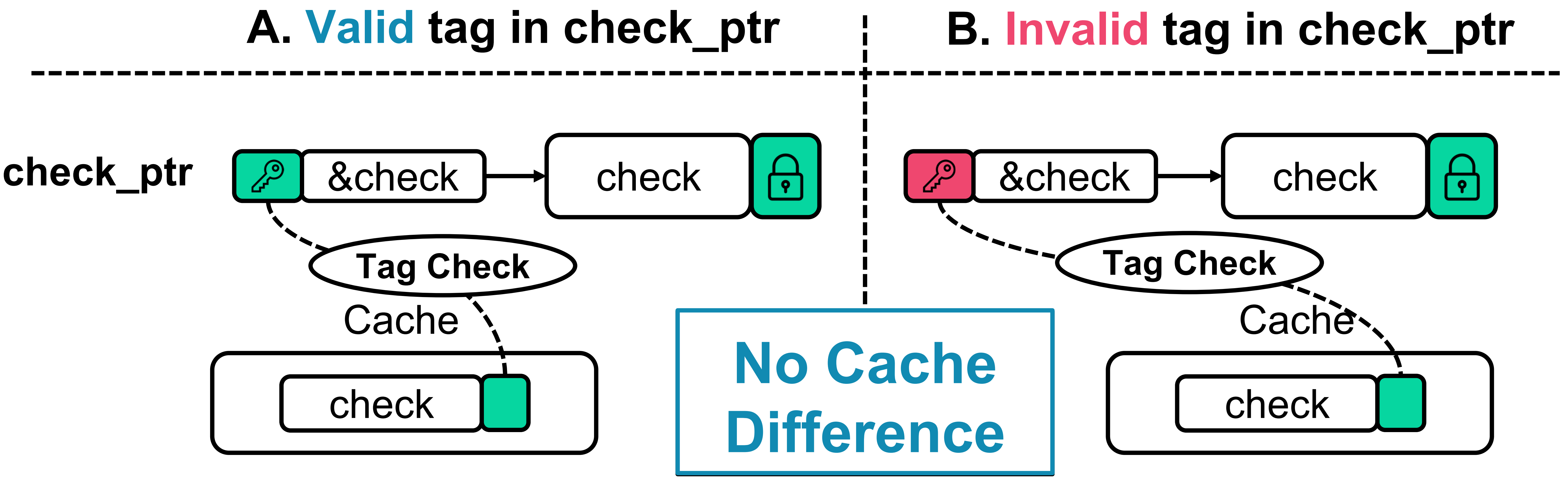
Goal: Leak the memory tag given a pointer



MTE Side-channel attack

Two test cases:

Access(check_ptr);

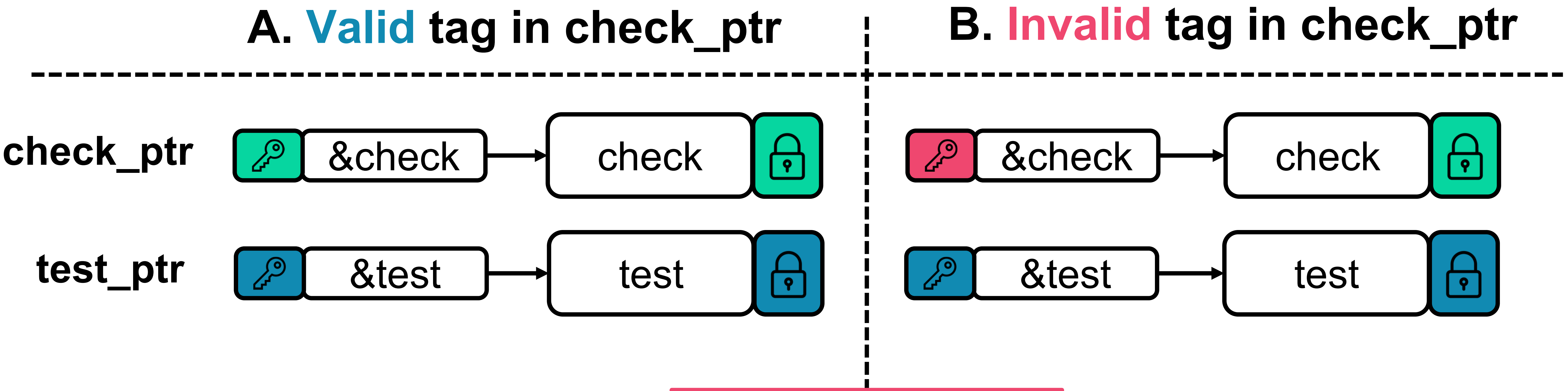


No Cache Difference

MTE Side-channel attack

Two test cases:

```
Access(check_ptr); Access(test_ptr);
```

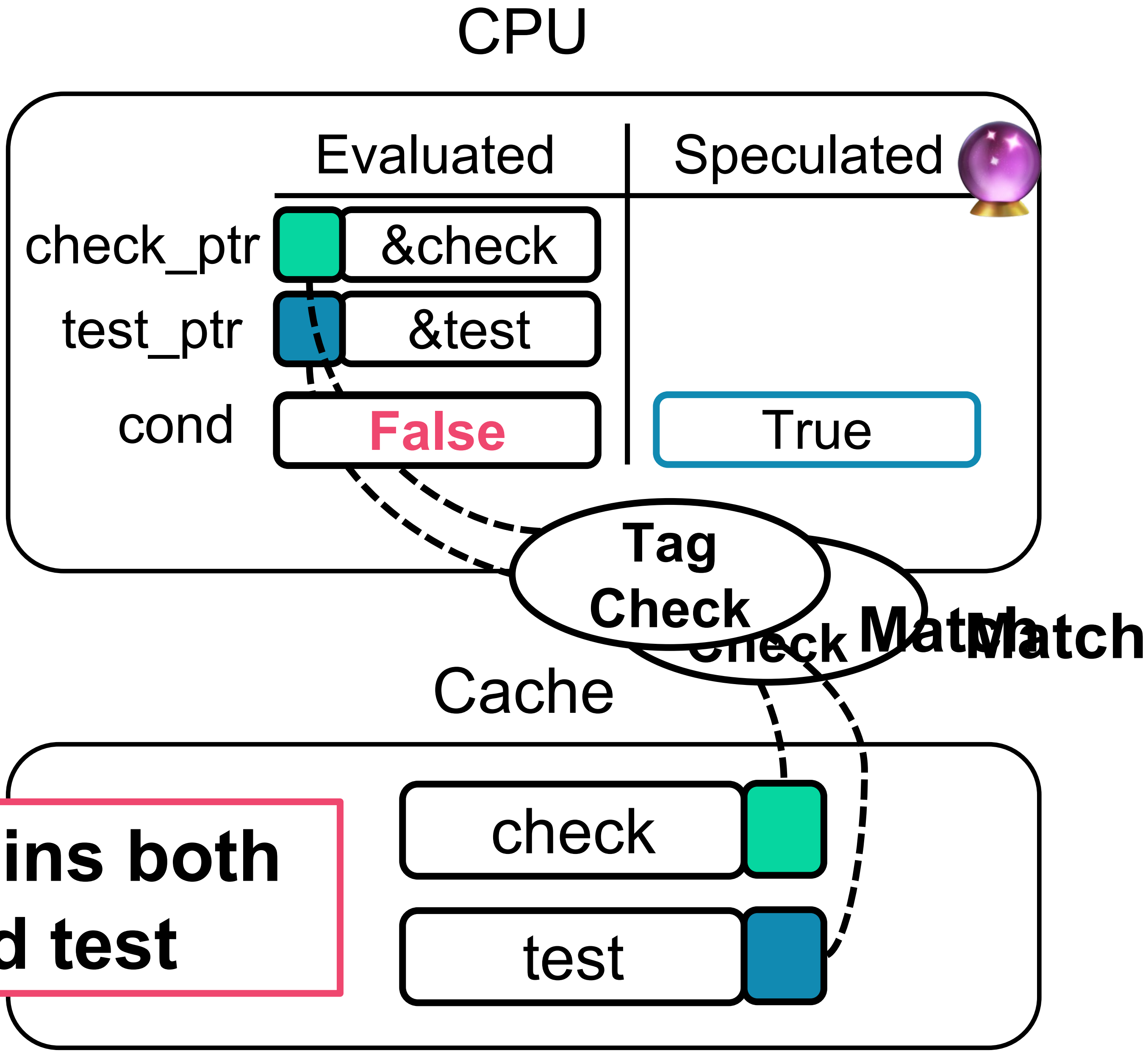


Cache Difference?

A. Valid tag in check_ptr

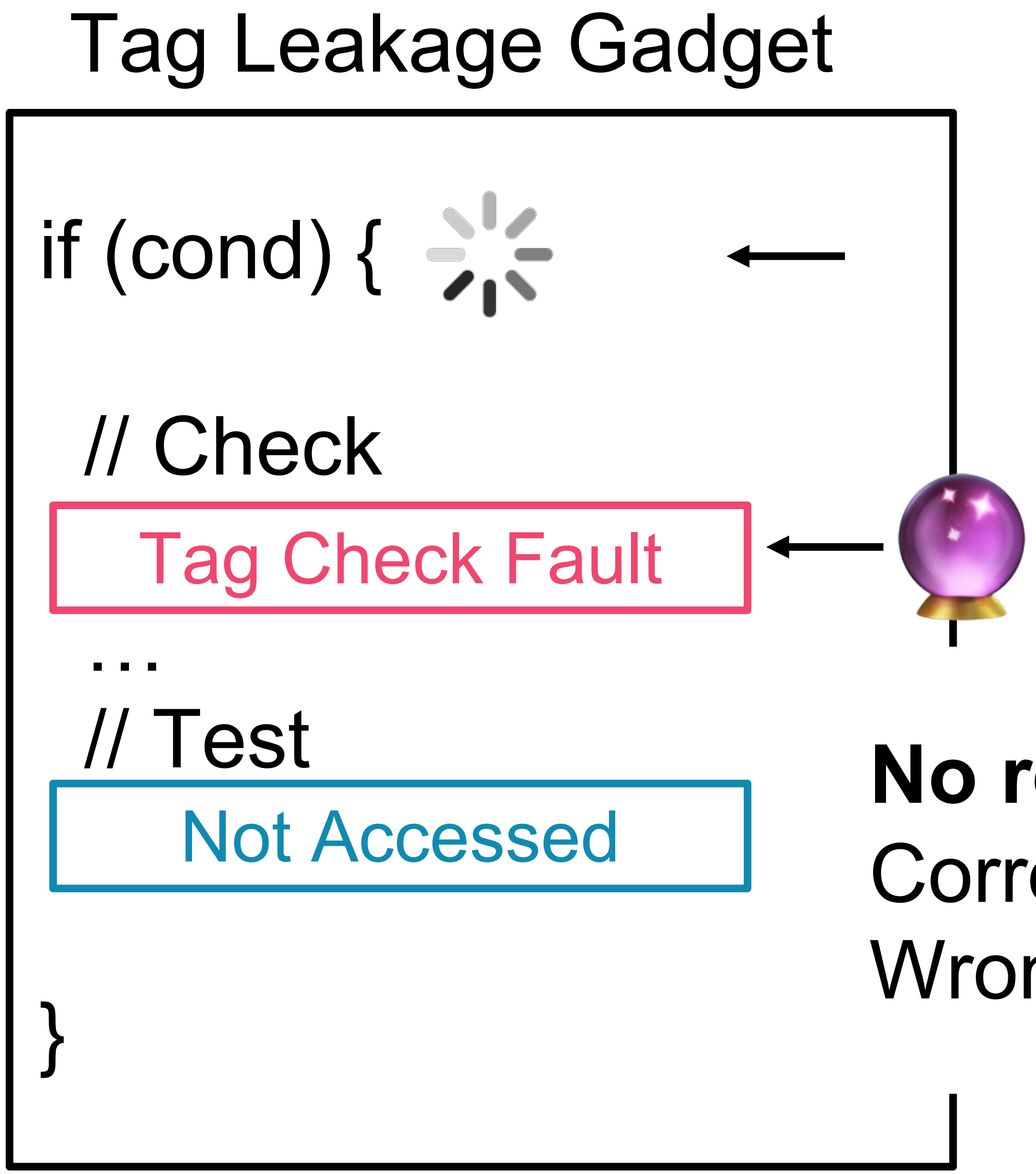
Tag Leakage Gadget

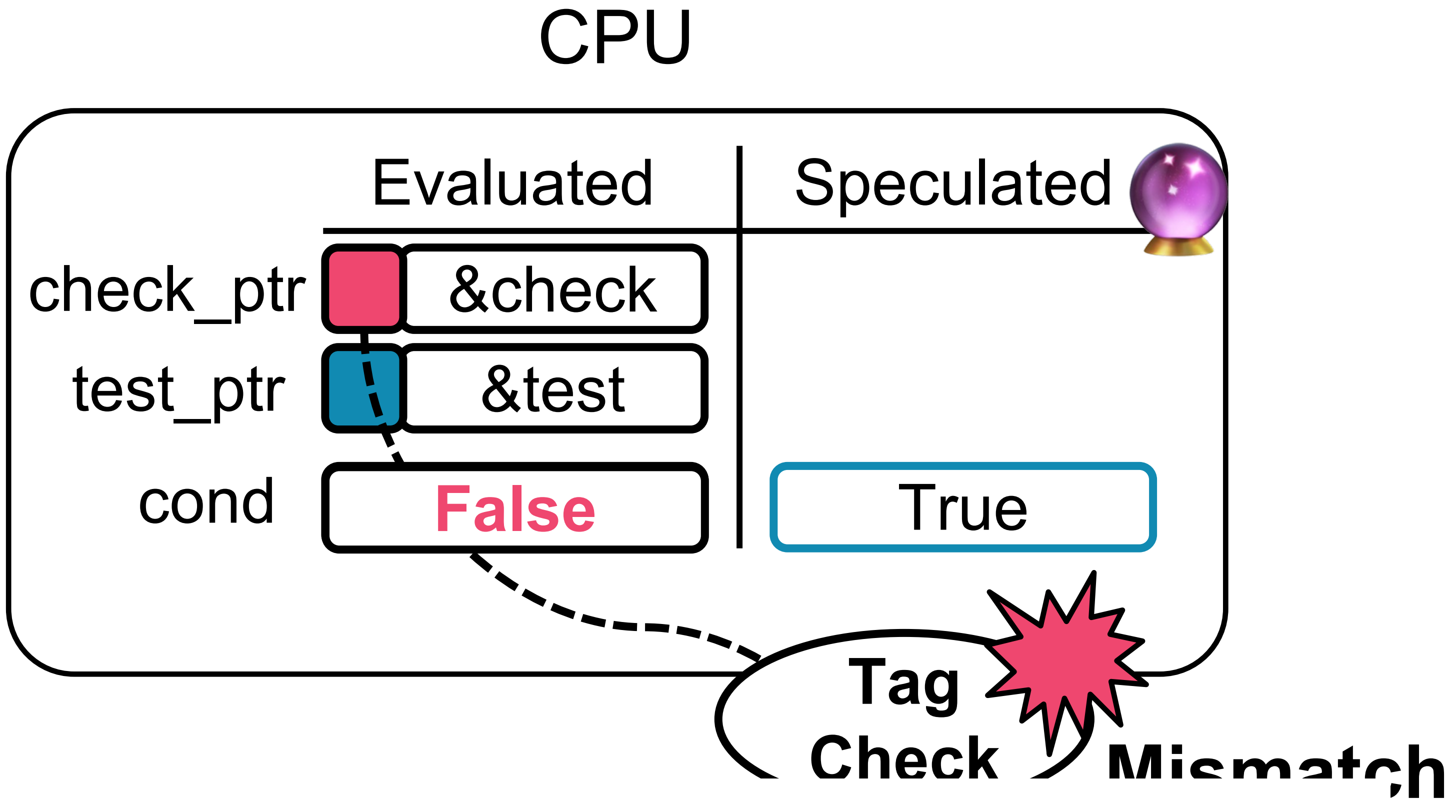
```
if (cond) {

 // Check
 Access(check_ptr);
 …
 // Test
 Access(test_ptr);

}
```



Cache contains both check and test

B. Invalid tag in check_ptr





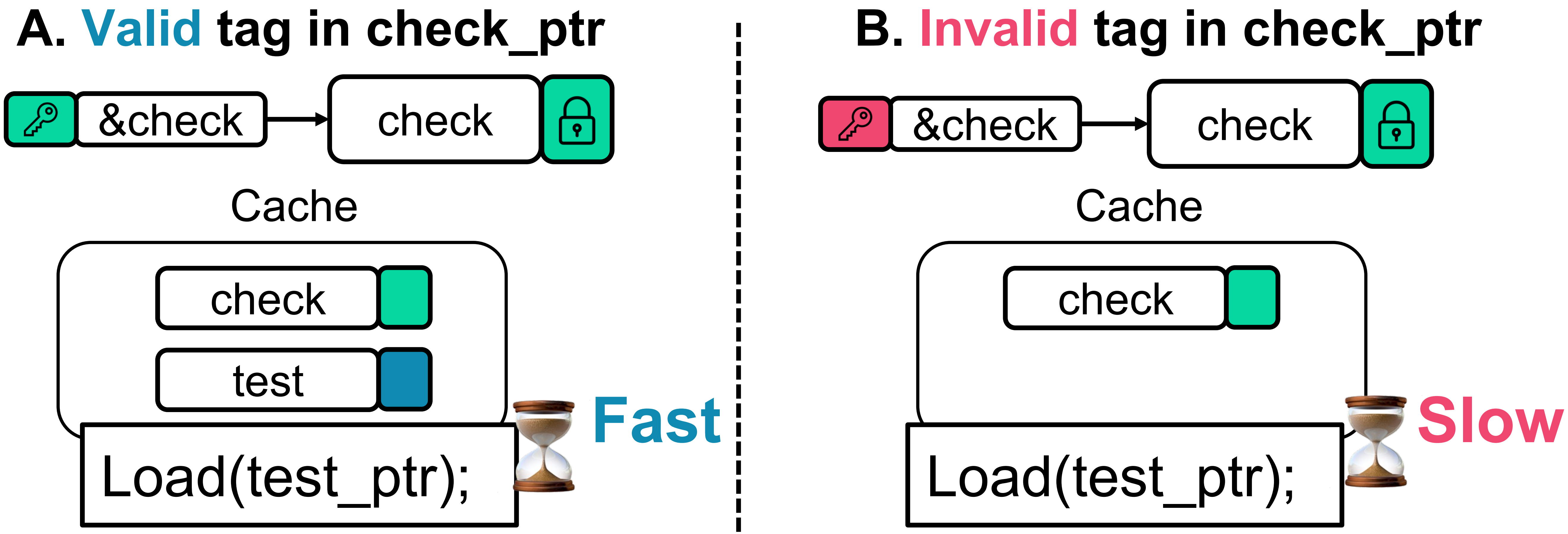
No reason to continue speculative execution

Correct spec→ (synchronous) tag check fault

Wrong spec → Revert execution

check, not test

Leak by Cache Side-Channel



Leak whether the tag is Valid/Invalid by test_ptr access latency!

Do new MTE chips contain the tag leakage side-channels?

PACMAN – ISCA 2022, DEF CON 30

- Discovered a Pointer Authentication Code (PAC) side-channel

MTE as Tested – Google Project Zero, POC 2023

- Attempted to find a MTE tag side-channel → **Failed**

Our work

- **Found 2 Tag Leakage Gadgets + Susepected Root Causes**
- Gadget poc: **https://github.com/compsec-snu/tiktag**
- Detailed analysis in our paper: **https://arxiv.org/abs/2406.08719**

StickyTags – VUSec, IEEE S&P 2024

- Orthogonally found one of our tag leakage gadets

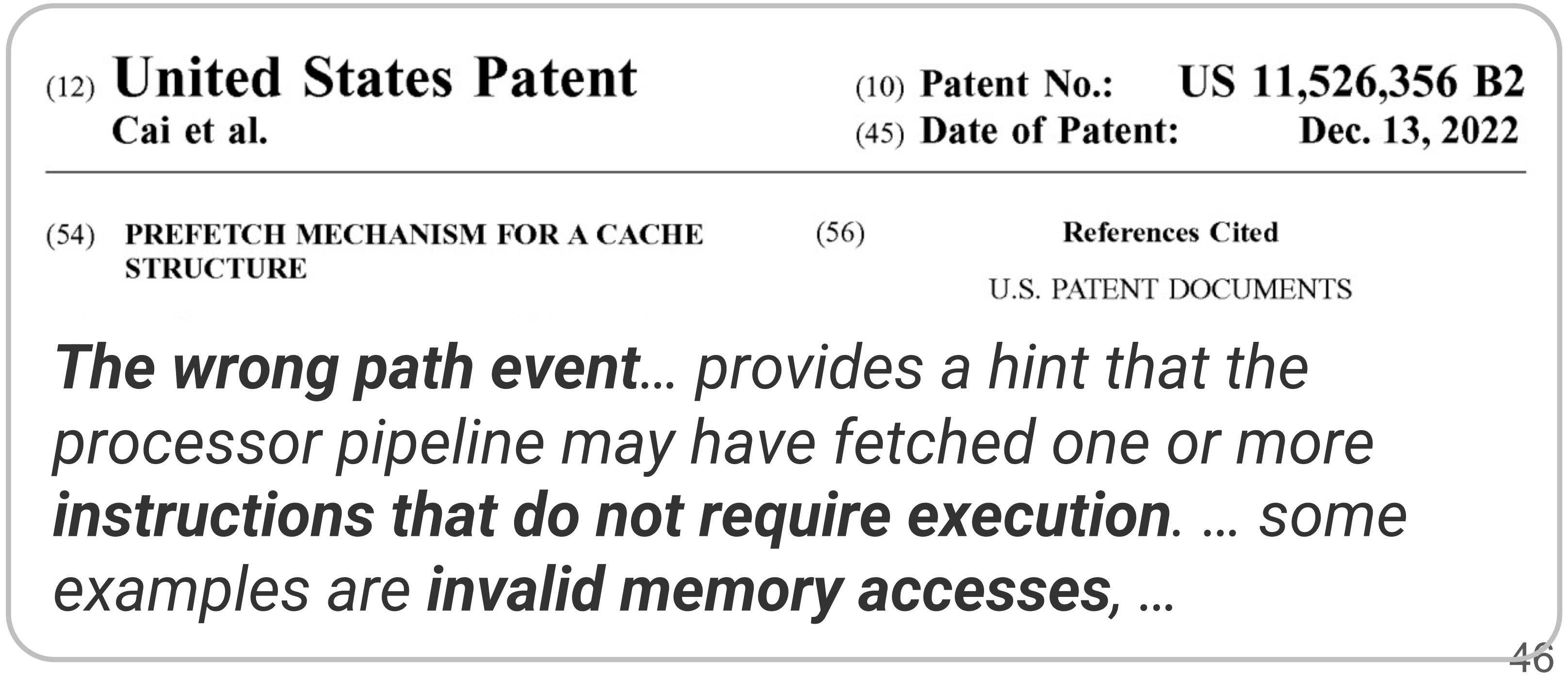
```
if (cond) {
  // Check
                k_ptr;
 Access(test_ptr);
}
```

Gadget 1: Multiple Loads

```
if (cond) {
  …
  // Check: 2+ load
  *check_ptr;    Tag Check Fault
  *check_ptr;    Tag Check Fault
  …
  // Test: load/store
  *test_ptr;     No Access
}
```

Suspected root cause

- On ***multiple faults,*** the CPU ***re-speculates that the speculation was wrong***
  => **stop/reduce speculations** in branch speculation and memory prefetcher



(12) **United States Patent**
Cai et al.

(10) **Patent No.:** **US 11,526,356 B2**
(45) **Date of Patent:** **Dec. 13, 2022**

(54) **PREFETCH MECHANISM FOR A CACHE STRUCTURE**

(56) **References Cited**

U.S. PATENT DOCUMENTS

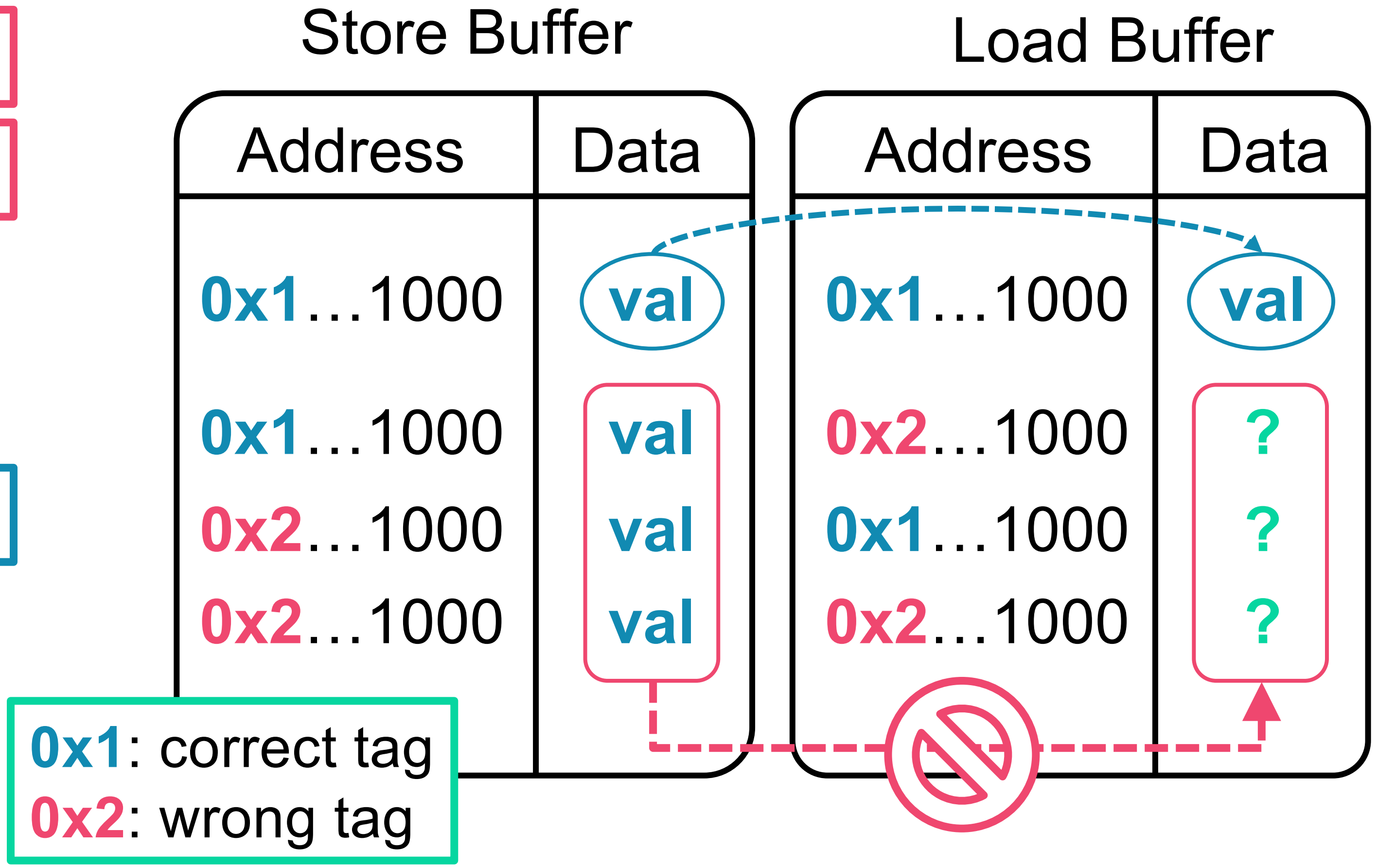
The wrong path event… *provides a hint that the processor pipeline may have fetched one or more* ***instructions that do not require execution****. … some examples are* ***invalid memory accesses****, …*

Gadget 2: Store-to-Load Forwarding

```
if (cond) {
  // Check: store-to-load
  *check_ptr = val;       Tag Check Fault
  val = *check_ptr;       Tag Check Fault

  // Test: dependent load/store
  *(test_ptr+val);        No Access
}
```

Suspected root cause

- On **tag check fault**, the CPU blocks ***store-to-load forwarding***



Store Buffer

Address	Data
0x1…1000	val
0x1…1000	val
0x2…1000	val
0x2…1000	val

Load Buffer

Address	Data
0x1…1000	val
0x2…1000	?
0x1…1000	?
0x2…1000	?

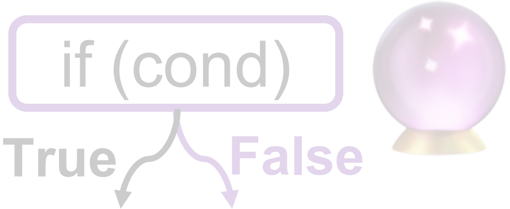
0x1: correct tag
0x2: wrong tag

Roadmap

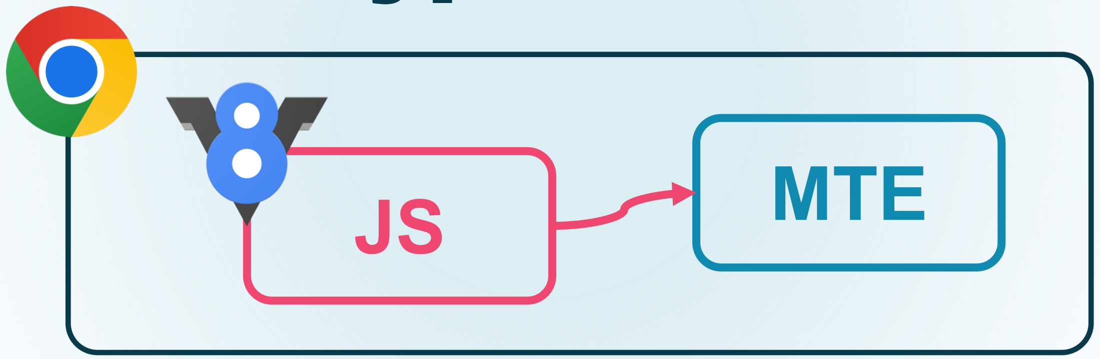
ARM Memory Tagging Extension

Cache Side-Channel

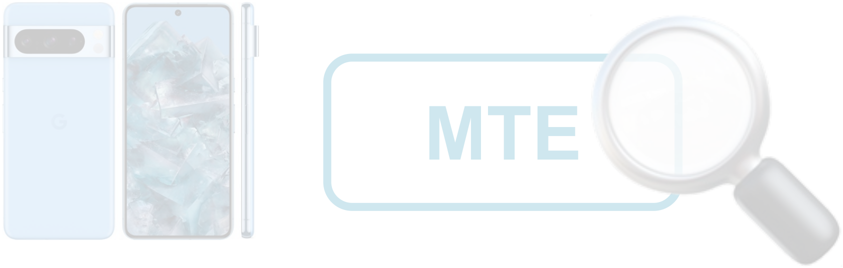
Speculative Execution



Real-world MTE Bypass Attack



MTE Tag Leakage Side-Channel



Real-world MTE-Enabled Software

- MTE became recently available
- Software systems that provide (optional) MTE support

android

Google Chrome

Linux Kernel

Secure OSes

GrapheneOS

Unikraft

OPTEE

- More software systems are likely to adopt MTE in the near future

Real-world Gadgets & Attacks

When MTE is enabled

1. Google Chrome V8 Engine

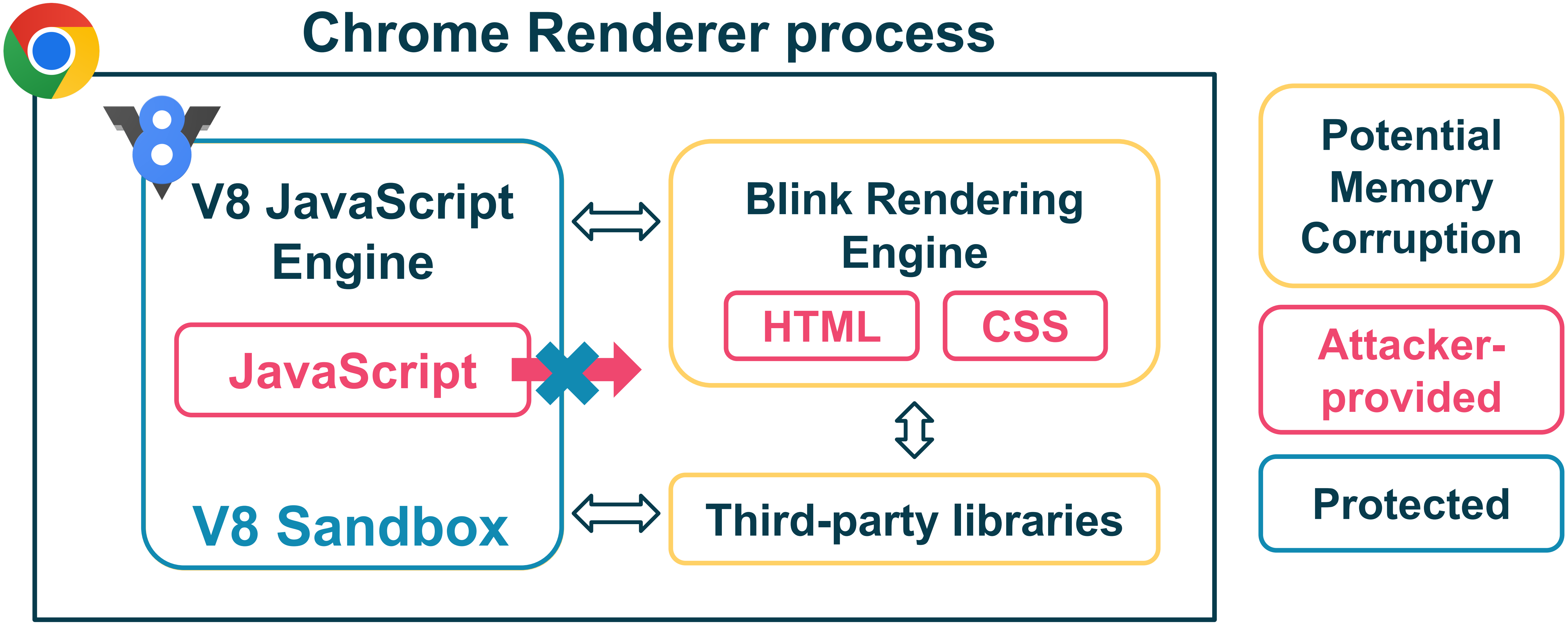
Constructed exploitable Gadget 2 from JavaScript
→ Leak MTE tag of the renderer memory

2. Linux kernel

Found potential Gadget 1 in snd_timer()
→ Leak MTE tag of the kernel memory from user space

Refer to our paper for the details: https://arxiv.org/abs/2406.08719

Google Chrome Threat Model



Chrome Renderer process

V8 JavaScript Engine

JavaScript

V8 Sandbox

Blink Rendering Engine

HTML

CSS

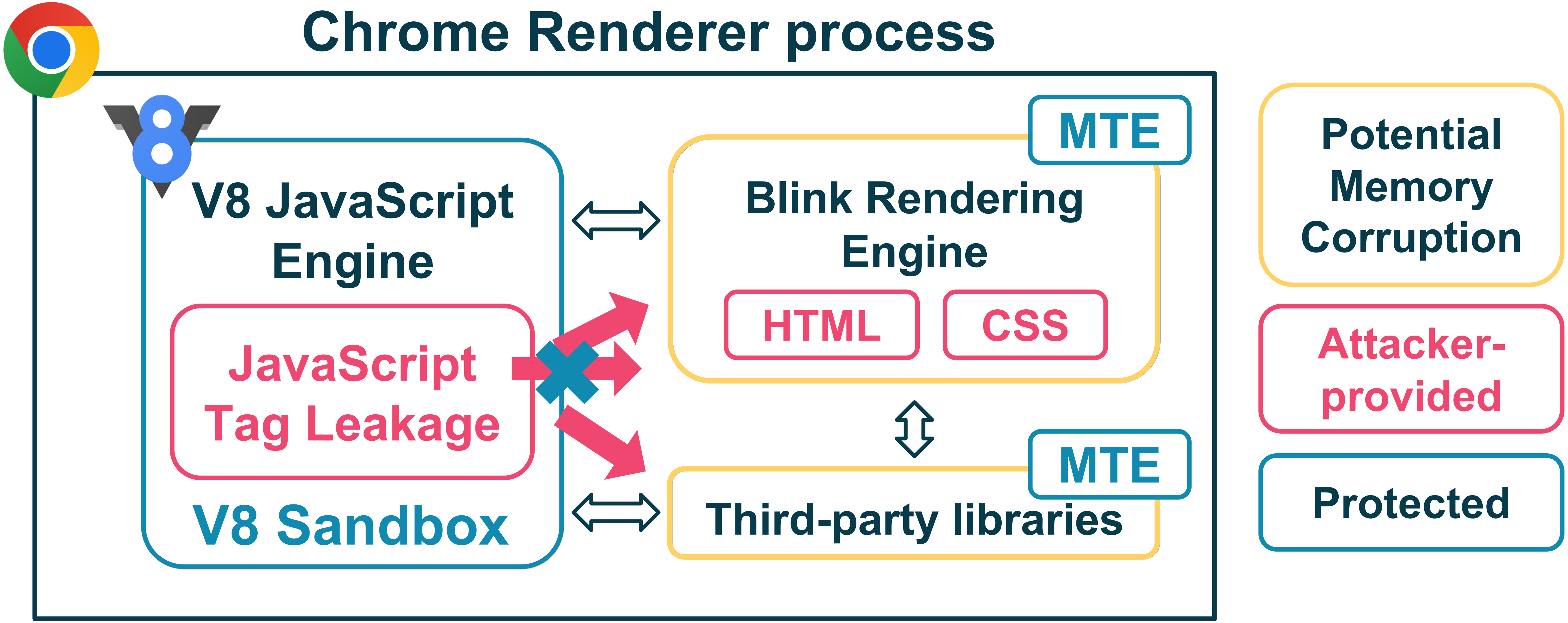
Third-party libraries

Potential Memory Corruption

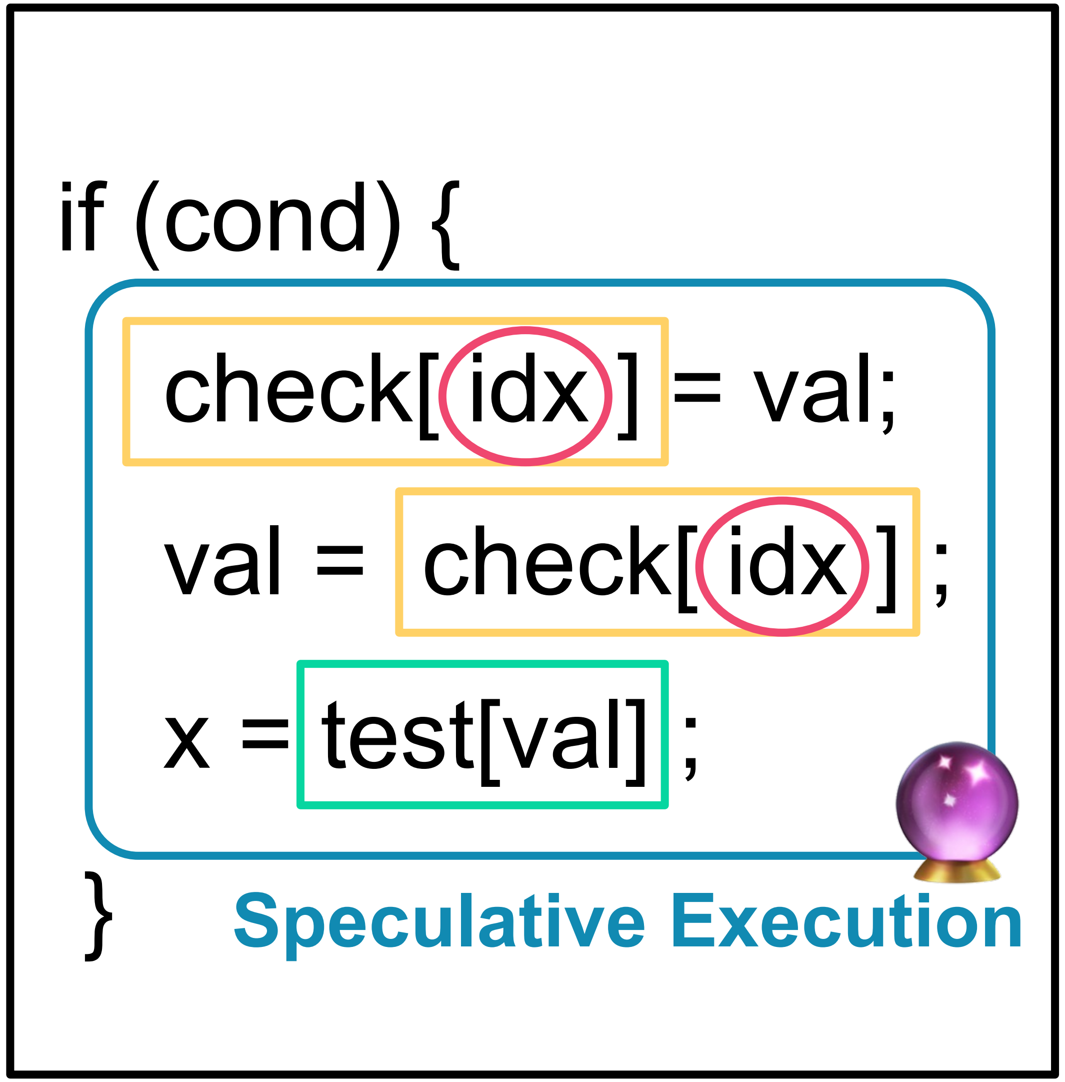
Attacker-provided

Protected

Google Chrome Threat Model



Gadget 2 from JavaScript



```
if (cond) {
  check[idx] = val;
  val = check[idx];
  x = test[val];
}
```

idx: out-of-bounds index (64-bit)

check[idx] : check_ptr

test[val] : test_ptr

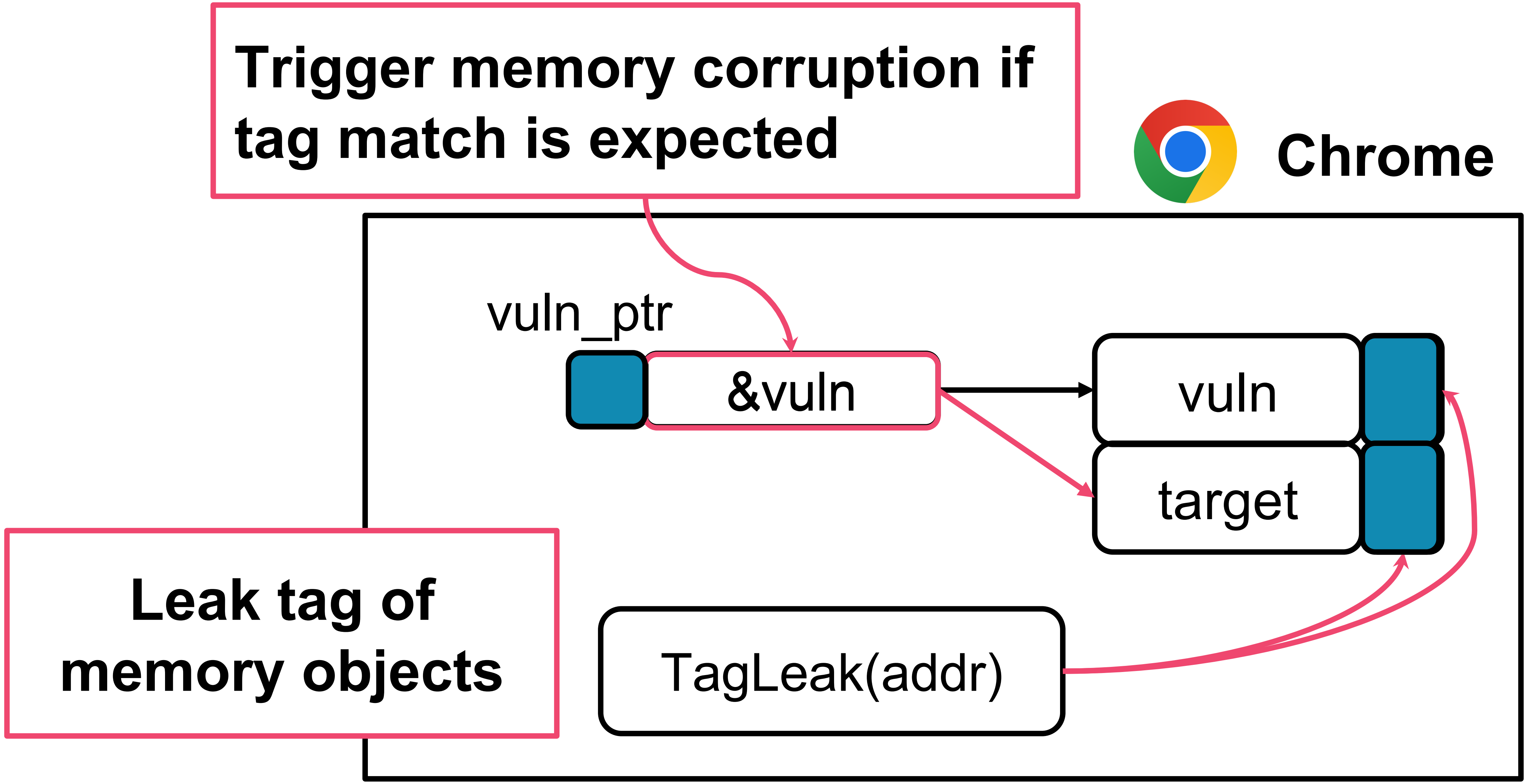
Gadget 2 in Google V8 (JavaScript)

```
TagLeak(target) {
  for (let tag=0; tag < 16; ++tag) {   ← Iterate all tag values
    idx = AddrToIdx(tag, target);      ← out-of-bounds index
      if (cond) {
        check[ idx ] = val;
        val = check[ idx ];             Tag Leakage
        x = test[val];                  Gadget
      }
    time[tag] = Measure( test[val] );
  }
  return time.indexOf(min(time));
}
```

check[idx] Valid tag	Invalid tag
No fault	Tag Check Fault
No fault	Tag Check Fault
Access	No Access
Fast	**Slow**

Tag Leaked!

Chrome MTE Bypass Attack

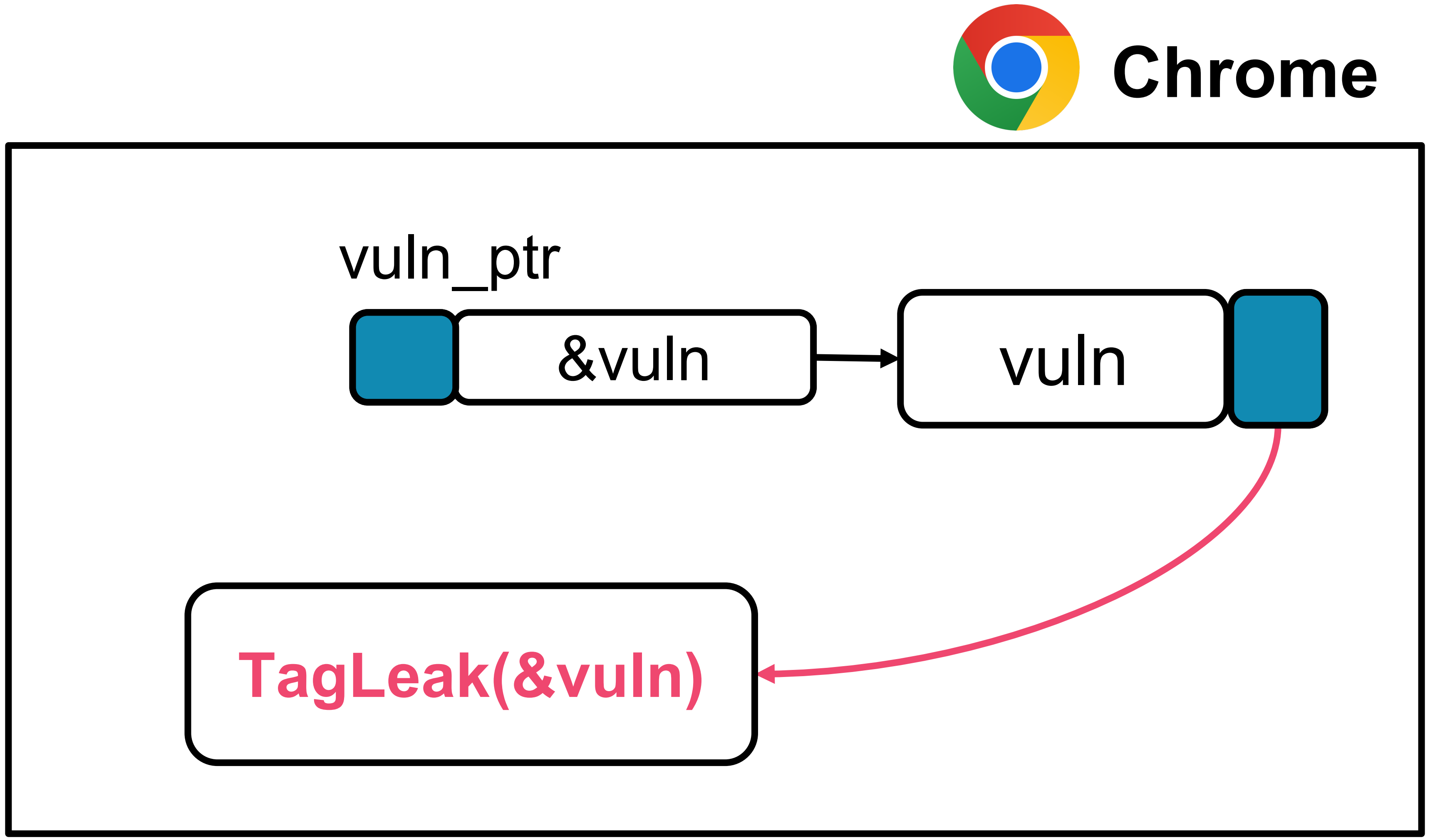


Trigger memory corruption if tag match is expected

Leak tag of memory objects

1. Leak MTE Tag of vulnerable object

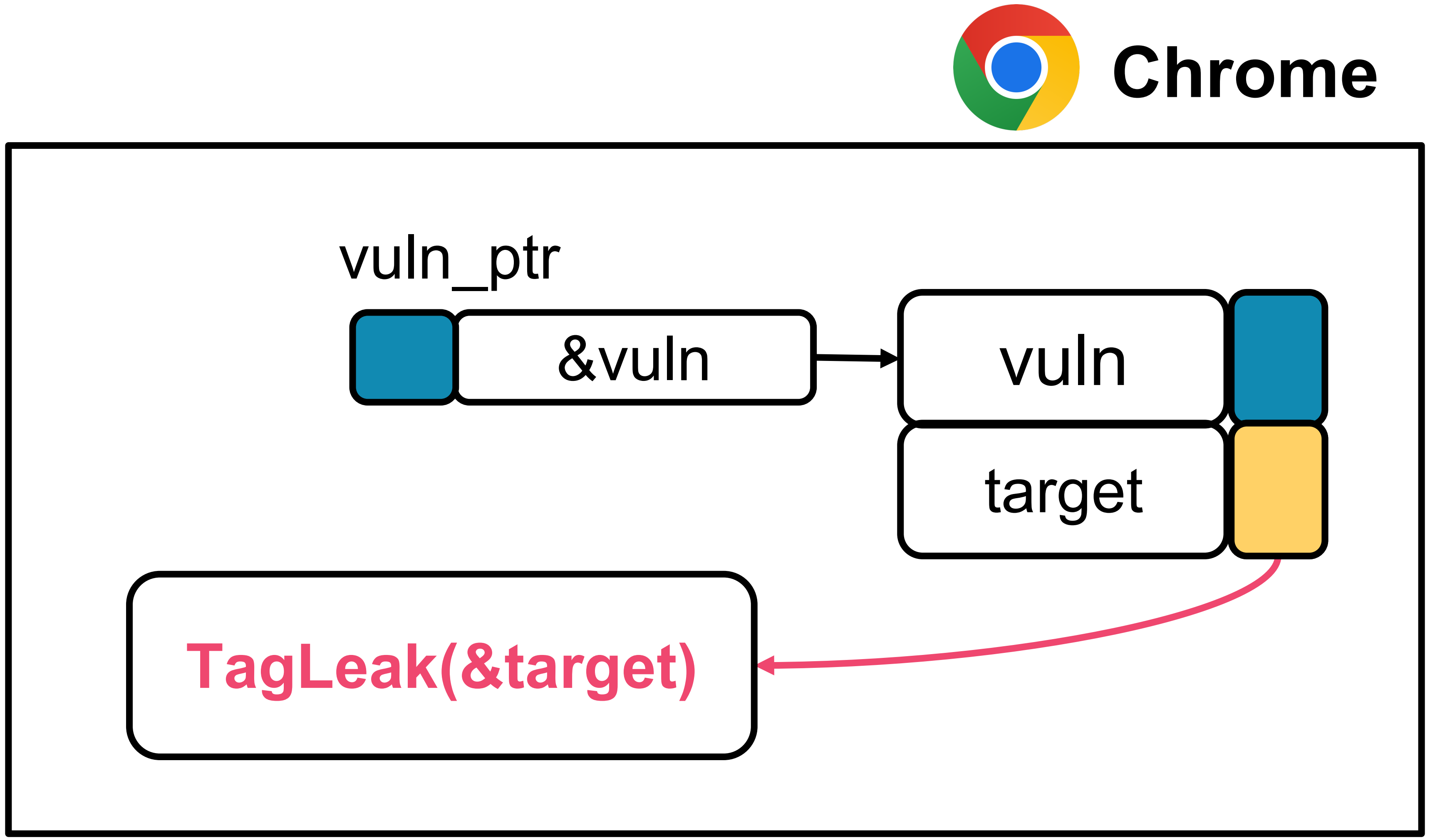
vuln.tag = 🟦



2. Leak MTE Tag of target object

vuln.tag = 🟦

target.tag = 🟨

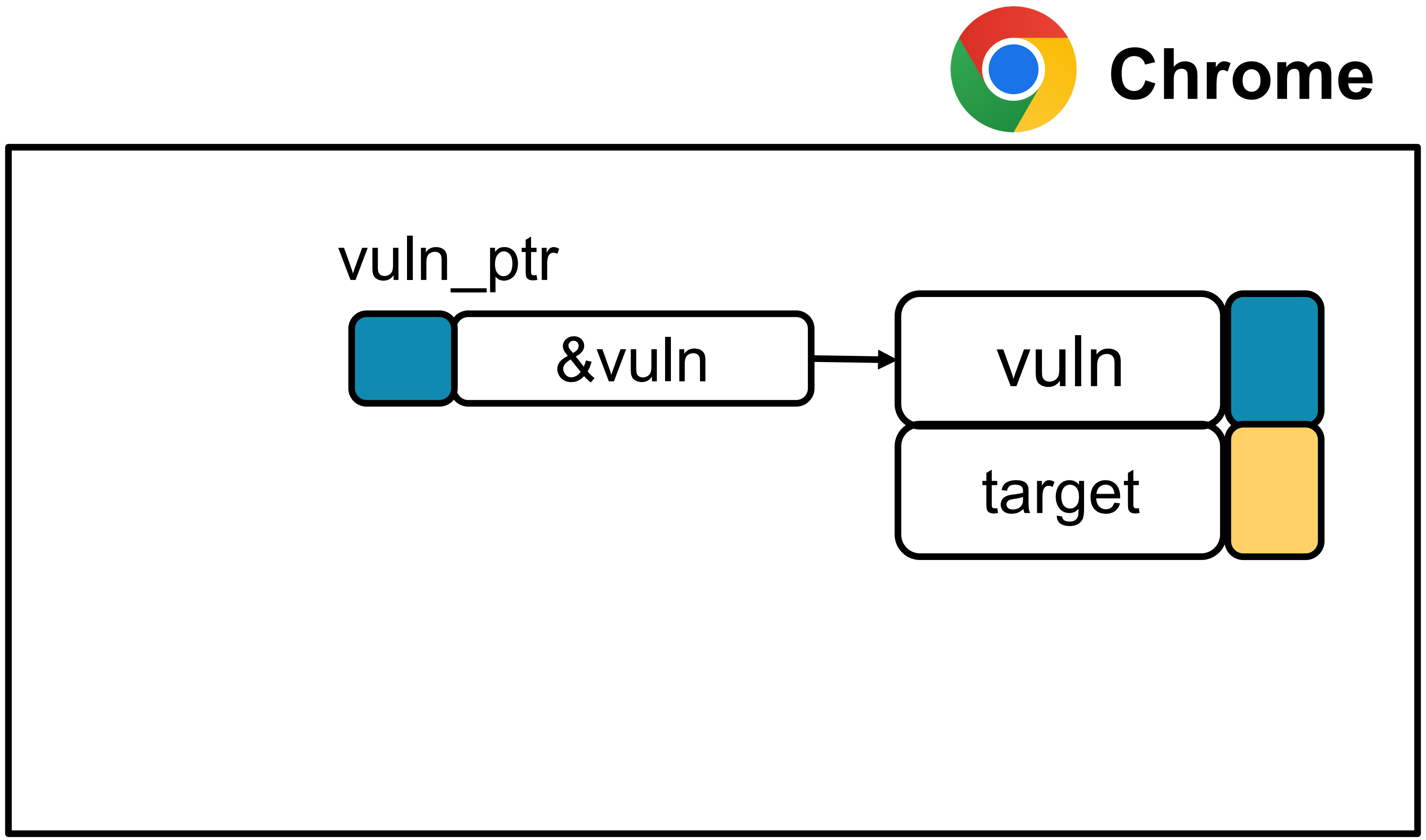


3. Reallocate target on tag mismatch

vuln.tag = 🟦

target.tag = 🟨

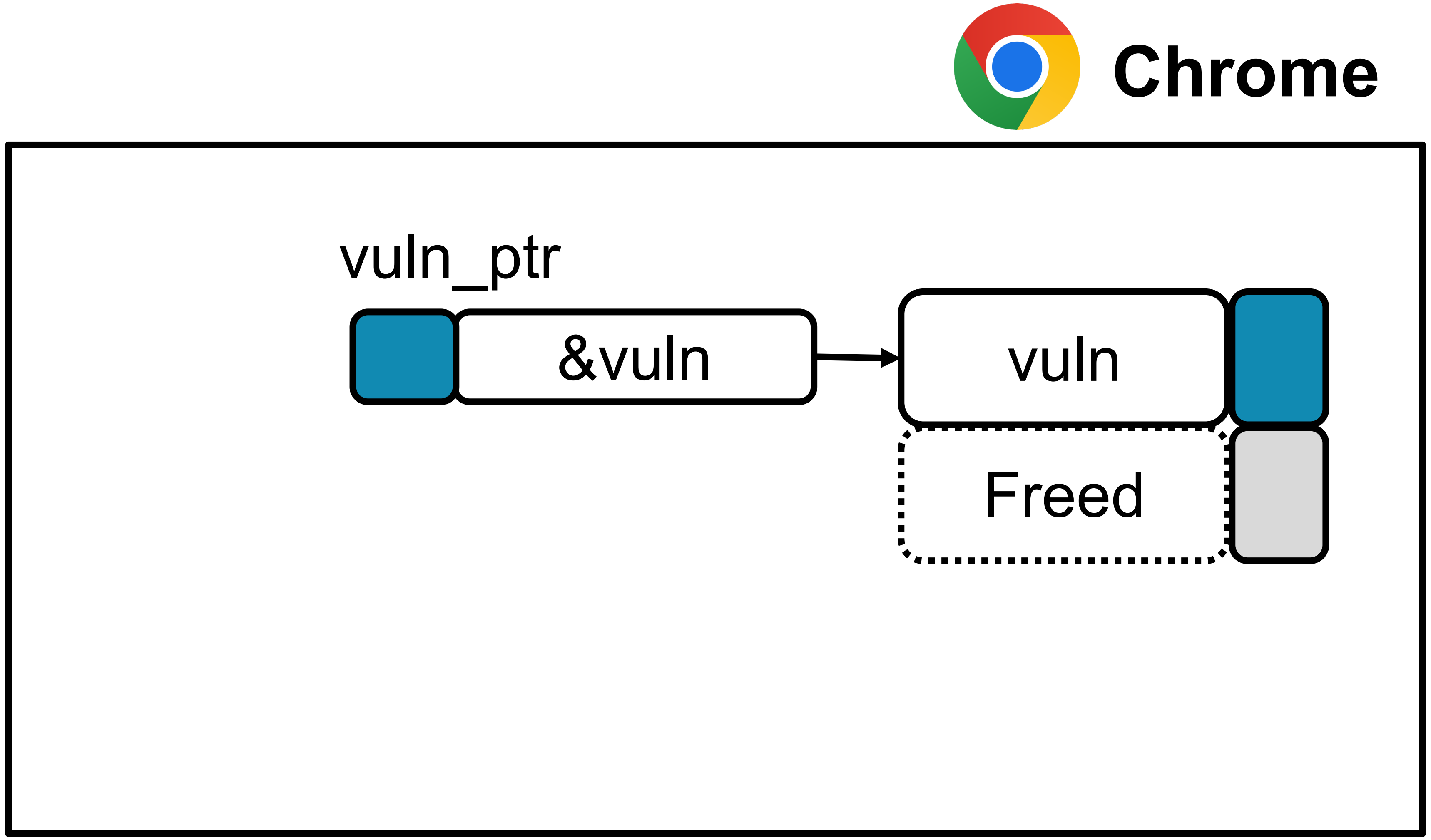
vuln.tag != target.tag



3. Reallocate target on tag mismatch

vuln.tag = 🟦

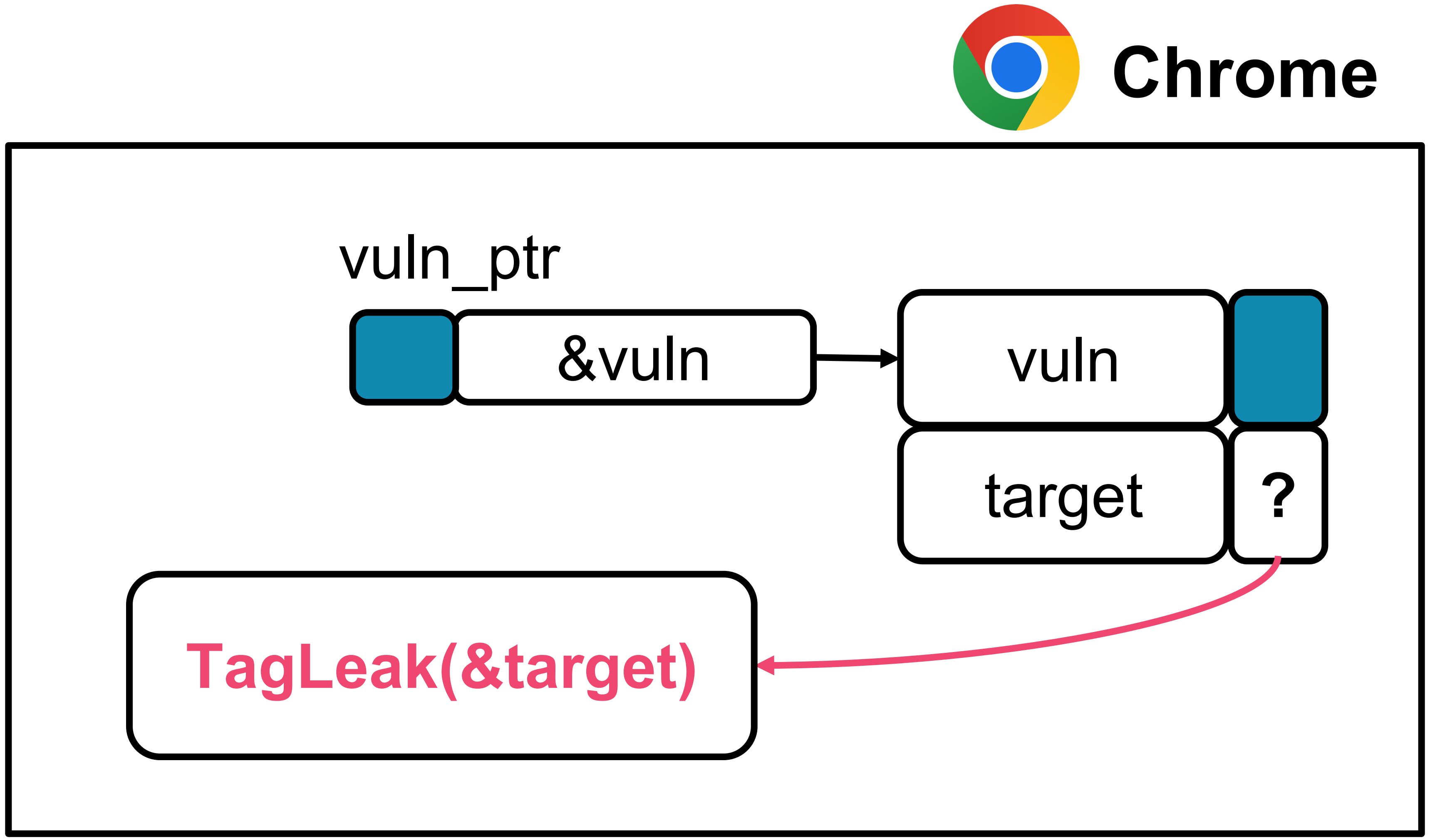
Free(target);



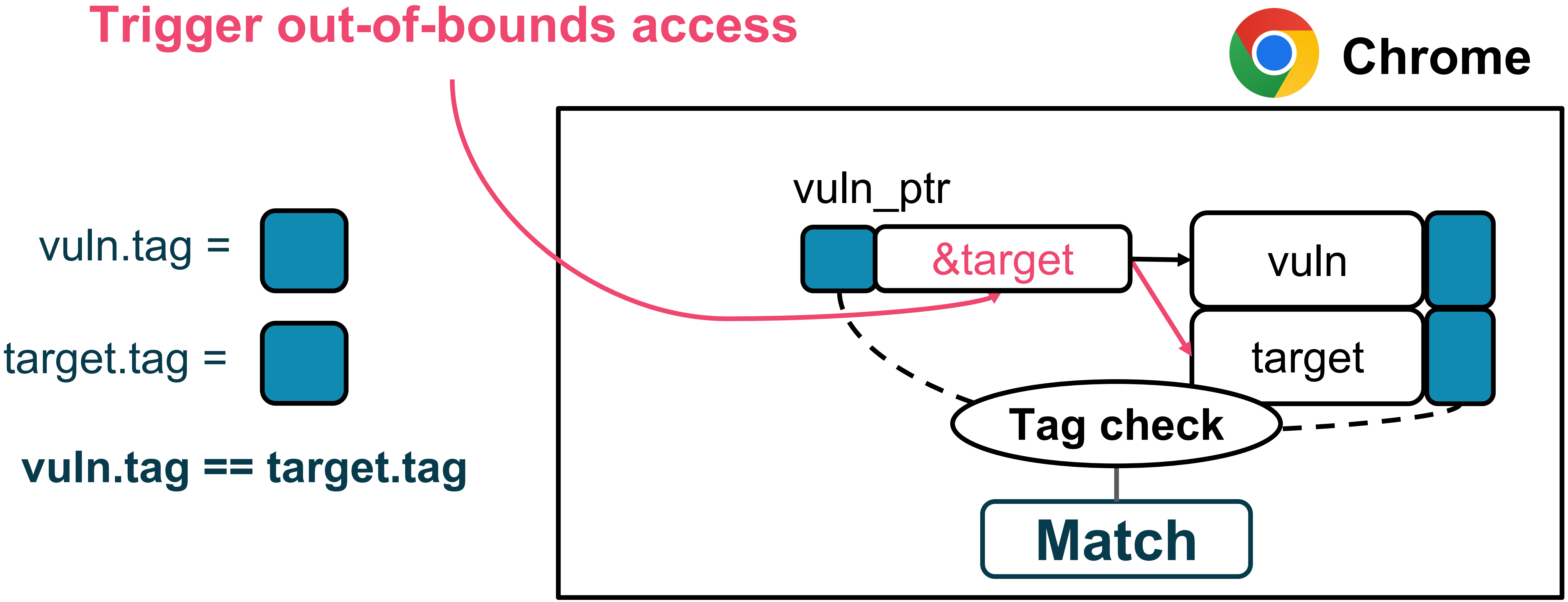
3. Reallocate target on tag mismatch

vuln.tag = 🟦

```
Free(target);
Alloc(target);
```

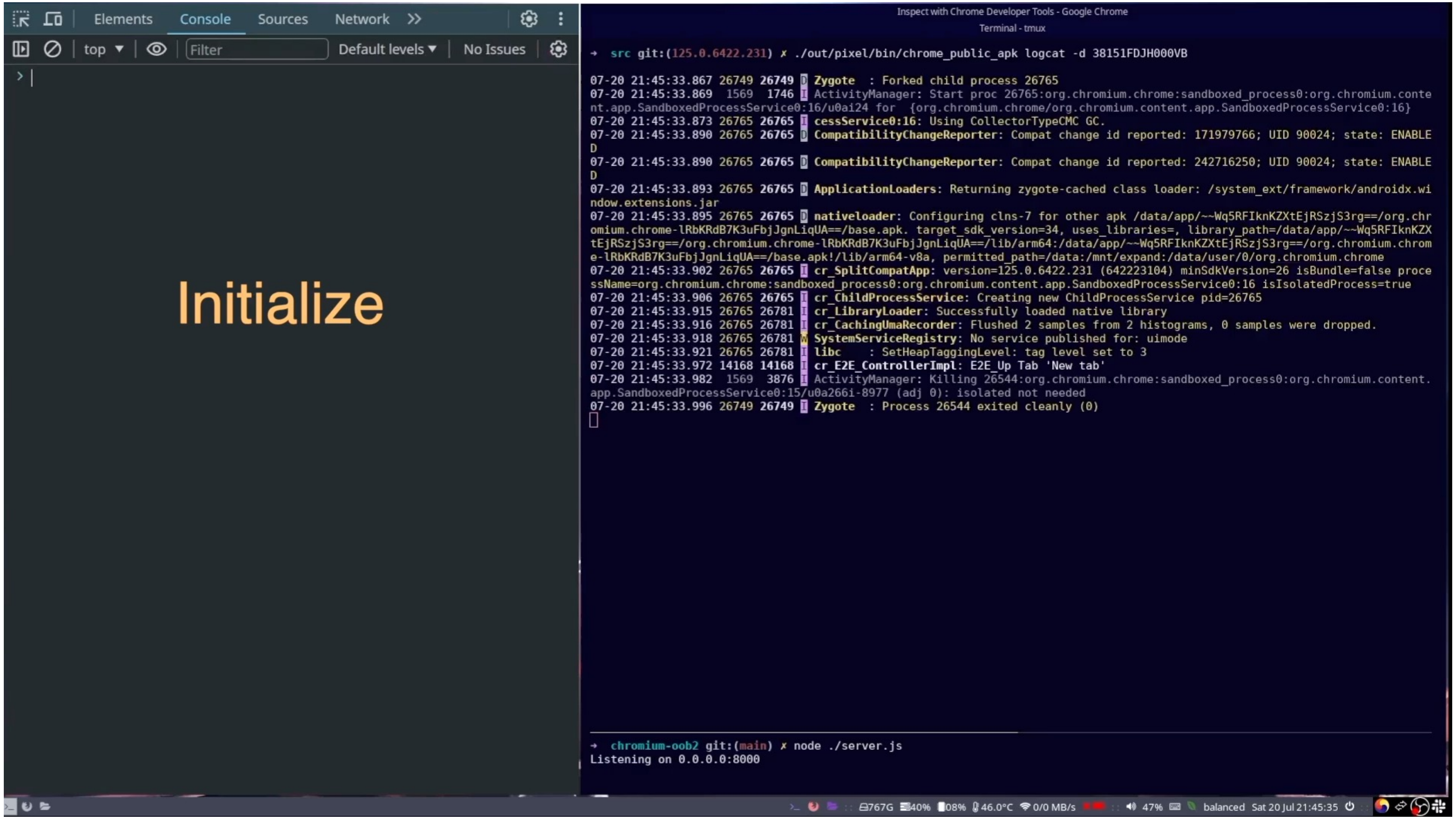


4. Trigger vulnerability on tag match



CVE-2023-5217 Chrome libvpx heap overlfow

Original Memory Corruption → Attack Fail

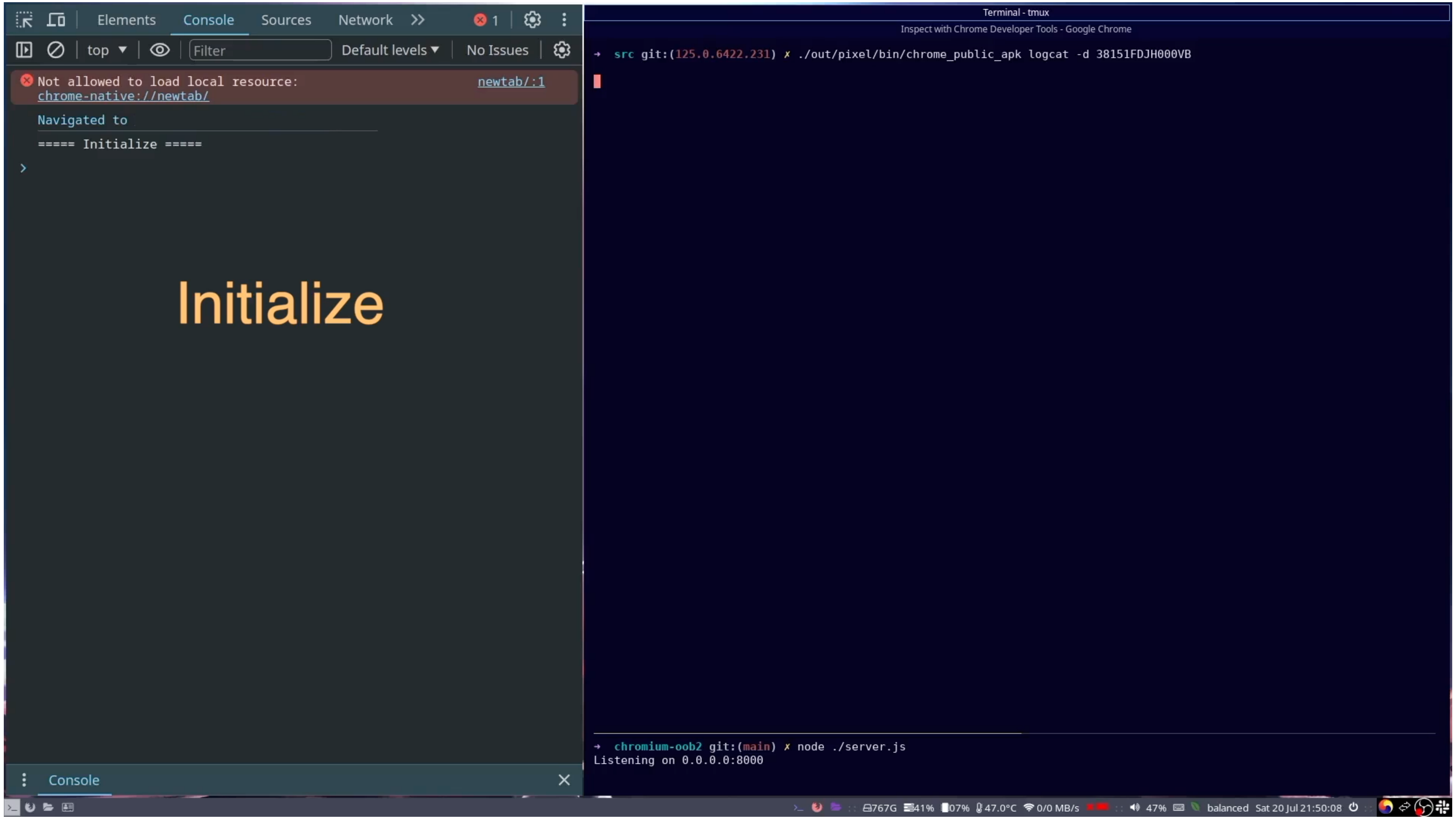


```
→ src git:(125.0.6422.231) ✗ ./out/pixel/bin/chrome_public_apk logcat -d 38151FDJH000VB

07-20 21:45:33.867 26749 26749 D Zygote  : Forked child process 26765
07-20 21:45:33.869  1569  1746 I ActivityManager: Start proc 26765:org.chromium.chrome:sandboxed_process0:org.chromium.conte
nt.app.SandboxedProcessService0:16/u0ai24 for  {org.chromium.chrome/org.chromium.content.app.SandboxedProcessService0:16}
07-20 21:45:33.873 26765 26765 I cessService0:16: Using CollectorTypeCMC GC.
07-20 21:45:33.890 26765 26765 D CompatibilityChangeReporter: Compat change id reported: 171979766; UID 90024; state: ENABLE
D
07-20 21:45:33.890 26765 26765 D CompatibilityChangeReporter: Compat change id reported: 242716250; UID 90024; state: ENABLE
D
07-20 21:45:33.893 26765 26765 D ApplicationLoaders: Returning zygote-cached class loader: /system_ext/framework/androidx.wi
ndow.extensions.jar
07-20 21:45:33.895 26765 26765 D nativeloader: Configuring clns-7 for other apk /data/app/~~Wq5RFIknKZXtEjRSzjS3rg==/org.chr
omium.chrome-lRbKRdB7K3uFbjJgnLiqUA==/base.apk. target_sdk_version=34, uses_libraries=, library_path=/data/app/~~Wq5RFIknKZX
tEjRSzjS3rg==/org.chromium.chrome-lRbKRdB7K3uFbjJgnLiqUA==/lib/arm64:/data/app/~~Wq5RFIknKZXtEjRSzjS3rg==/org.chromium.chrom
e-lRbKRdB7K3uFbjJgnLiqUA==/base.apk!/lib/arm64-v8a, permitted_path=/data:/mnt/expand:/data/user/0/org.chromium.chrome
07-20 21:45:33.902 26765 26765 I cr_SplitCompatApp: version=125.0.6422.231 (642223104) minSdkVersion=26 isBundle=false proce
ssName=org.chromium.chrome:sandboxed_process0:org.chromium.content.app.SandboxedProcessService0:16 isIsolatedProcess=true
07-20 21:45:33.906 26765 26765 I cr_ChildProcessService: Creating new ChildProcessService pid=26765
07-20 21:45:33.915 26765 26781 I cr_LibraryLoader: Successfully loaded native library
07-20 21:45:33.916 26765 26781 I cr_CachingUmaRecorder: Flushed 2 samples from 2 histograms, 0 samples were dropped.
07-20 21:45:33.918 26765 26781 W SystemServiceRegistry: No service published for: uimode
07-20 21:45:33.921 26765 26781 I libc    : SetHeapTaggingLevel: tag level set to 3
07-20 21:45:33.972 14168 14168 I cr_E2E_ControllerImpl: E2E_Up Tab 'New tab'
07-20 21:45:33.982  1569  3876 I ActivityManager: Killing 26544:org.chromium.chrome:sandboxed_process0:org.chromium.content.
app.SandboxedProcessService0:15/u0a266i-8977 (adj 0): isolated not needed
07-20 21:45:33.996 26749 26749 I Zygote  : Process 26544 exited cleanly (0)
```

```
→ chromium-oob2 git:(main) ✗ node ./server.js
Listening on 0.0.0.0:8000
```

CVE-2023-5217 Chrome libvpx heap overlfow

With MTE Tag Leakage → Attack Success



Elements
Console
Sources
Network
top
Filter
Default levels
No Issues

```
Not allowed to load local resource:                newtab/:1
chrome-native://newtab/
Navigated to
===== Initialize =====
>
```

Initialize
Console
Terminal - tmux
Inspect with Chrome Developer Tools - Google Chrome

```
→ src git:(125.0.6422.231) ✗ ./out/pixel/bin/chrome_public_apk logcat -d 38151FDJH000VB
```

```
→ chromium-oob2 git:(main) ✗ node ./server.js
Listening on 0.0.0.0:8000
```

767G 41% 07% 47.0°C 0/0 MB/s 47% balanced Sat 20 Jul 21:50:08

Vendor Responses

ARM

- Acknowledged the MTE tag side-channel in multiple ARM cores
- MTE Tags are not a secret
  → Tag leakage is not a security vulnerability
- Expected the cost of the hardware fix to be low
  and recommended the fix.

ARM MTE Security Updates:
https://developer.arm.com/Arm%20Security%20Center/Arm%20Memory%20Tagging%20Extension

Vendor Responses

Google Android Security Team

- MTE tag leakage are **hardware flaw** of Pixel 8 & Pixel 8 pro

- **Still, MTE is a strong mitigation against limited-shot exploits:**
  - Minimal attack surface (e.g., Messaging app)
  - Physically remote attack (e.g., Bluetooth, NFC, Wi-Fi, …)
  - Process isolation, IPC attack (e.g., Android, Chrome browser)

Vendor Responses

Google Chrome V8 Security Team

- **data confidentiality** (including MTE tag’s confidentiality) is out of scope of the V8 Sandbox

- Currently doesn’t plan to adopt MTE on renderer due to **potential side-channel issues**

Takeaway

- **ARM MTE** is a promising security feature to defend against **memory corruption attacks**

- However, current MTE hardware contains **tag leakage side-channel issues**

- MTE-based security can be improved by **software and hardware enhancement** in the future

Questions?