ZeRØ: Zero-Overhead Resilient Operation Under Pointer Integrity Attacks

Mohamed Tarek, Miguel Arroyo, Evgeny Manzhosov, and Simha Sethumadhavan

Columbia University

04/15/2021

COLUMBIA | ENGINEERING

The Fu Foundation School of Engineering and Applied Science

About Me

Mohamed Tarek

4th year PhD Candidate

@M_TarekIbnZiad

https://cs.columbia.edu/~mtarek

Memory Safety is a serious problem!

Computing Sep 6

Apple says China's Uighur Muslims were targeted in the recent iPhone hacking campaign

The tech giant gave a rare statement that bristled at Google's analysis of the novel hacking operation.

Memory Safety is a serious problem!

Computing Sep 6

Apple says China's Uighur Muslims were targeted in the recent iPhone hacking campaign

The tech giant gave a rare statement that bristled at Google's analysis of the novel hacking operation.

EDITOR'S PICK | 42,742 views | Nov 21, 2018, 07:00am

Exclusive: Saudi Dissidents Hit With Stealth iPhone Spyware Before Khashoggi's Murder

Memory Safety is a serious problem!

Computing Sep 6

Apple says China's Uighur Muslims were targeted in the recent iPhone hacking campaign

The tech giant gave a rare statement that bristled at Google's analysis of the novel hacking operation.

The New York Times

WhatsApp Rushes to Fix Security Flaw Exposed in Hacking of Lawyer's Phone

EDITOR'S PICK | 42,742 views | Nov 21, 2018, 07:00am

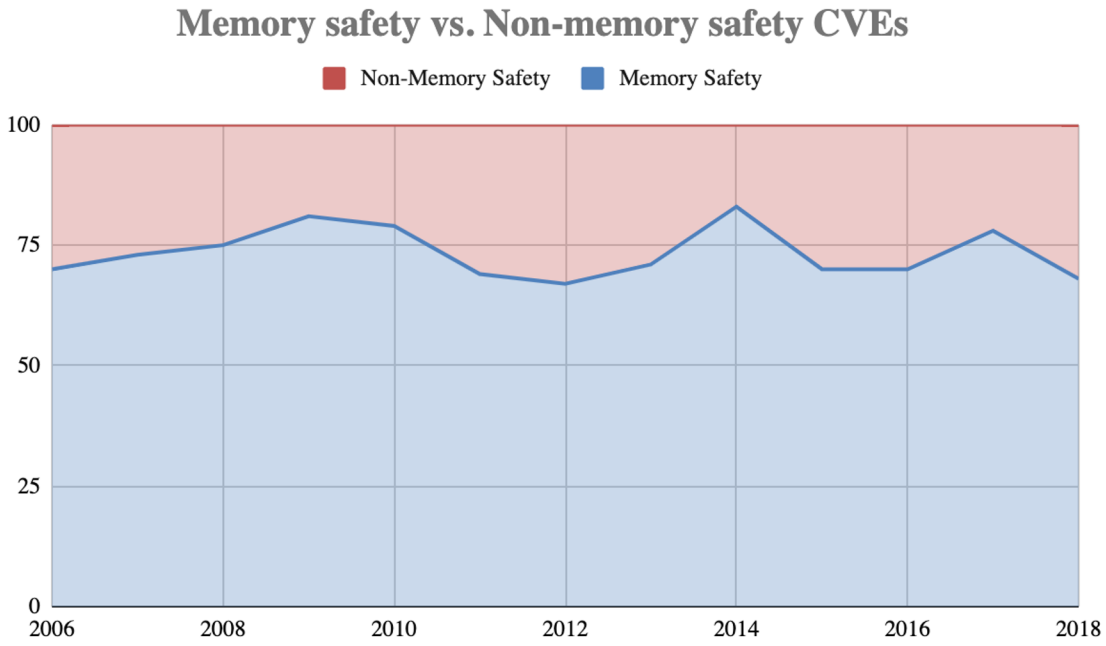
Exclusive: Saudi Dissidents Hit With Stealth iPhone Spyware Before Khashoggi's Murder

It’s easy to make mistakes

It’s easy to make mistakes

SEGFAULT!

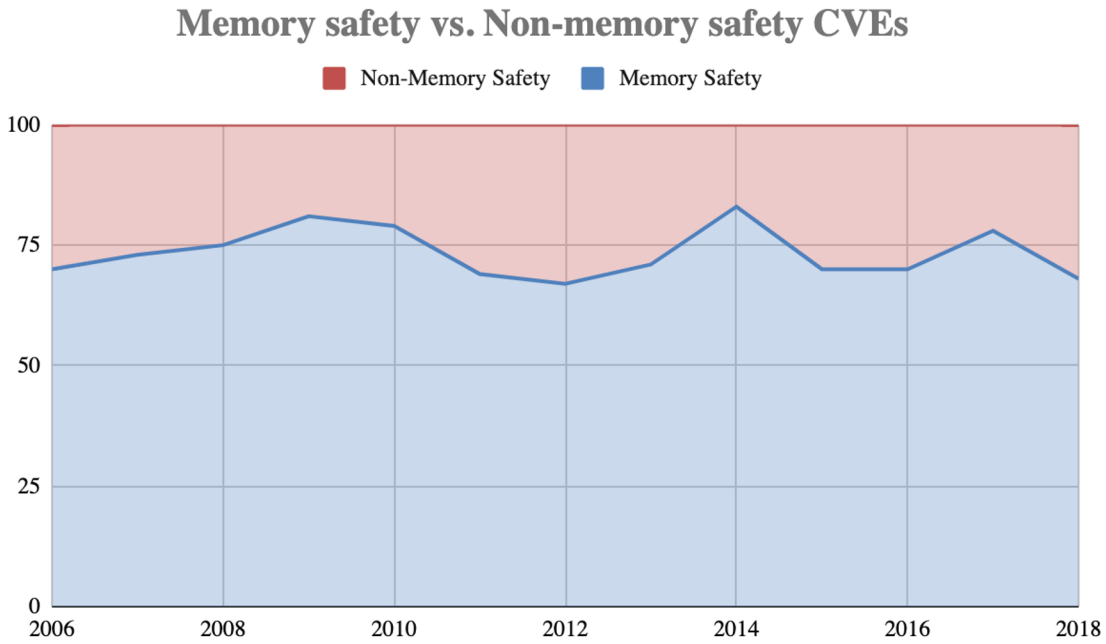
Prevalence of Memory Safety Vulns



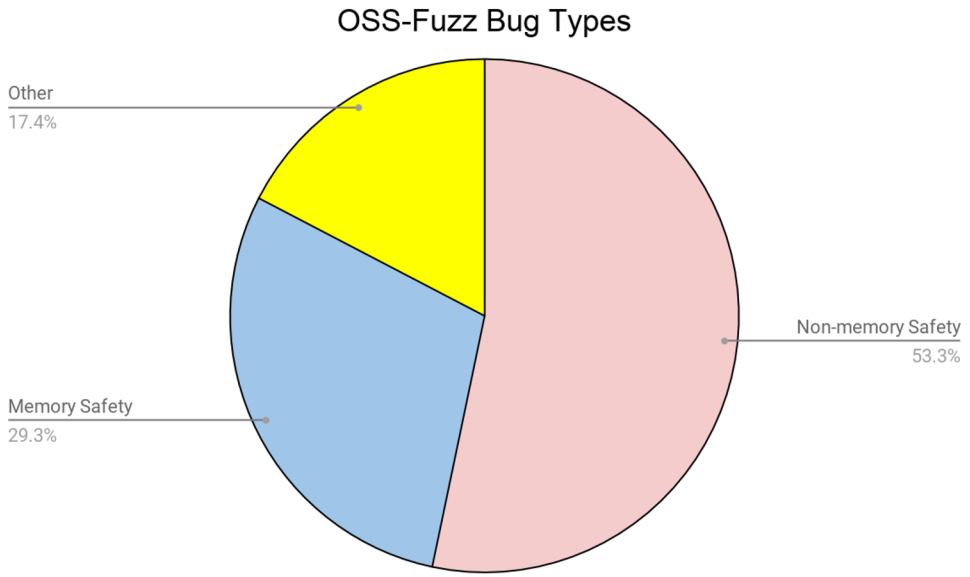
Microsoft Product CVEs

Source: *Matt Miller, Microsoft Security Response Center (MSRC) - BlueHat 2019*

Prevalence of Memory Safety Vulns



Microsoft Product CVEs



Google OSS-Fuzz bugs from 2016-2018.

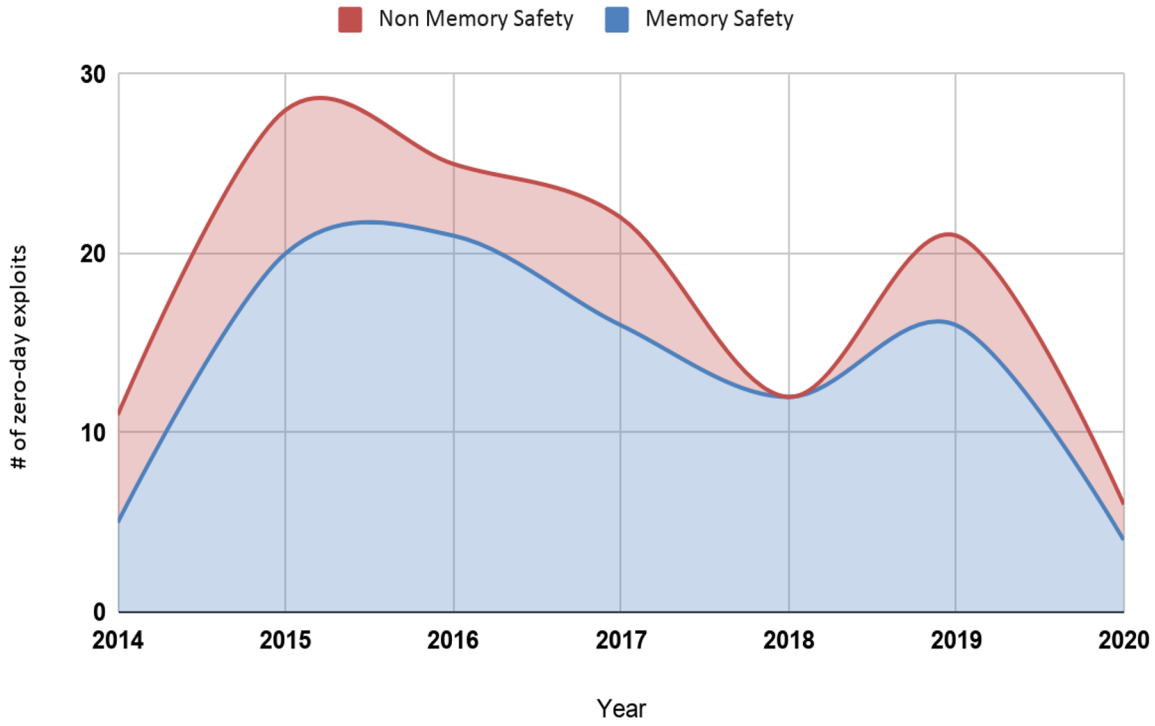
Source: *Matt Miller, Microsoft Security Response Center (MSRC) - BlueHat 2019*

Source: *https://security.googleblog.com/2018/11/a-new-chapter-for-oss-fuzz.html*

ATTACKERS

MEMORY SAFETY

Attackers Prefer Memory Safety Vulns



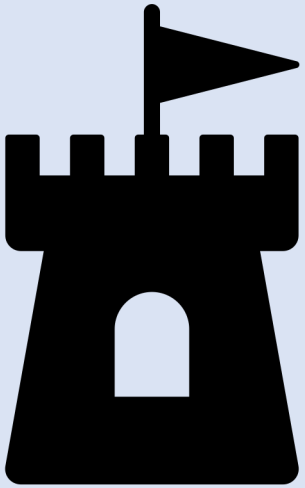
Zero-day “in the wild” exploits
from 2014-2020

Source: *Google Project Zero, 0day "In the Wild" spreadsheet. Last updated: April 2020*

How To Fix Memory (Un)Safety?

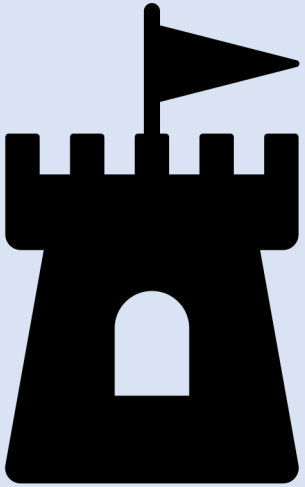
How To Fix Memory (Un)Safety?

Memory Safe Languages



How To Fix Memory (Un)Safety?

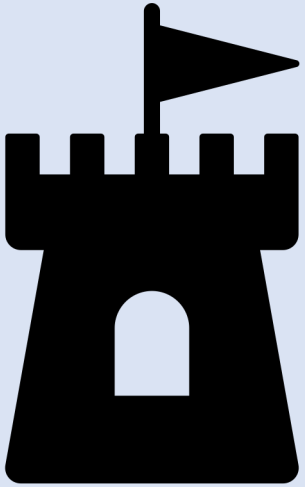
Memory Safe Languages



- Performance?
- Legacy Code?

How To Fix Memory (Un)Safety?

Memory Safe Languages

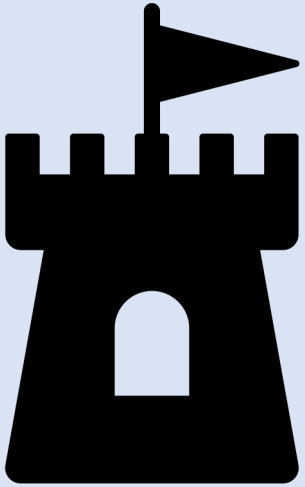


- Performance?
- Legacy Code?

Pre-deployment Testing

How To Fix Memory (Un)Safety?

Memory Safe Languages



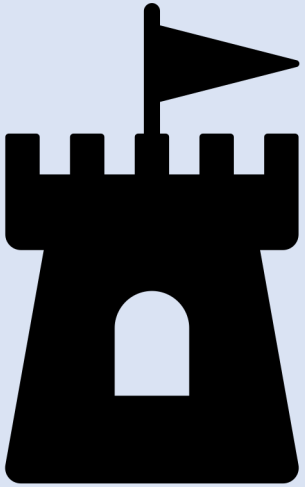
- Performance?
- Legacy Code?

Pre-deployment Testing

- Time?
- Scalability?

How To Fix Memory (Un)Safety?

Memory Safe Languages



- Performance?
- Legacy Code?

Pre-deployment Testing

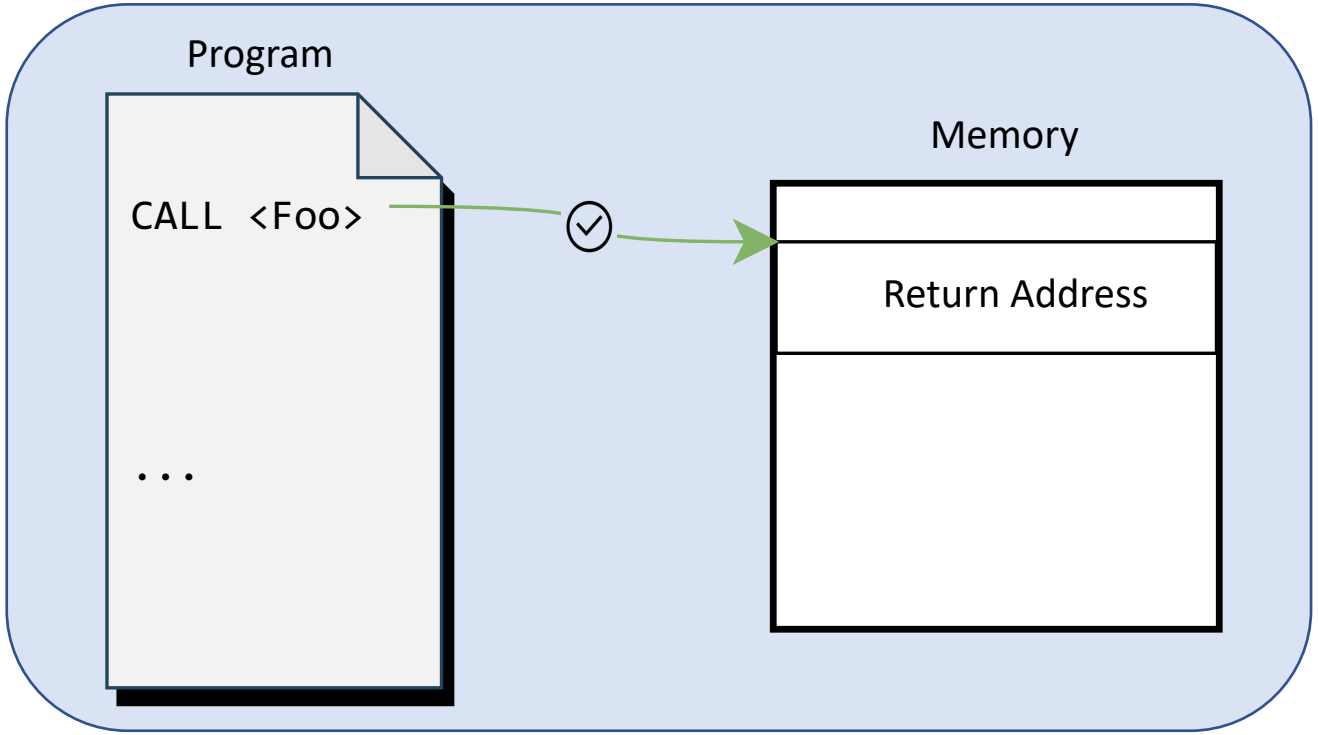
- Time?
- Scalability?

Post-deployment Mitigations


ZeRØ

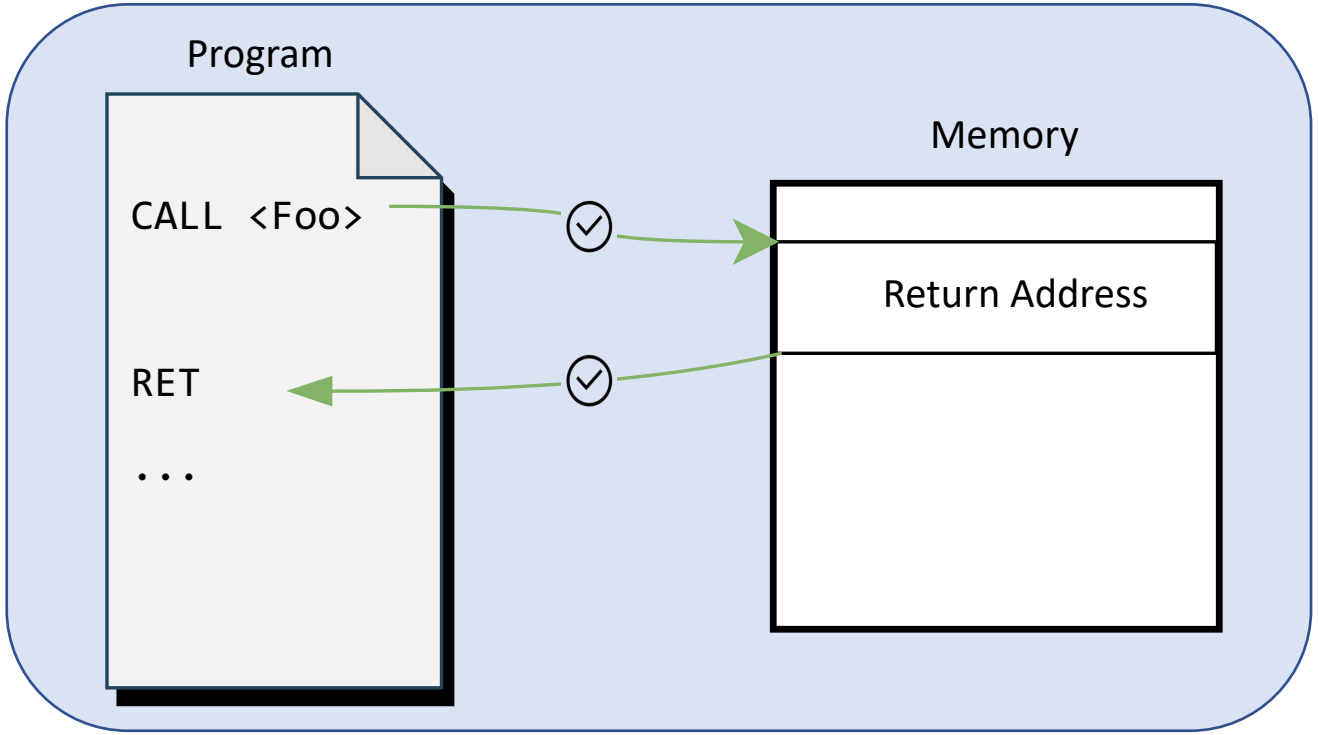
Overview

ZeRØ: Overview



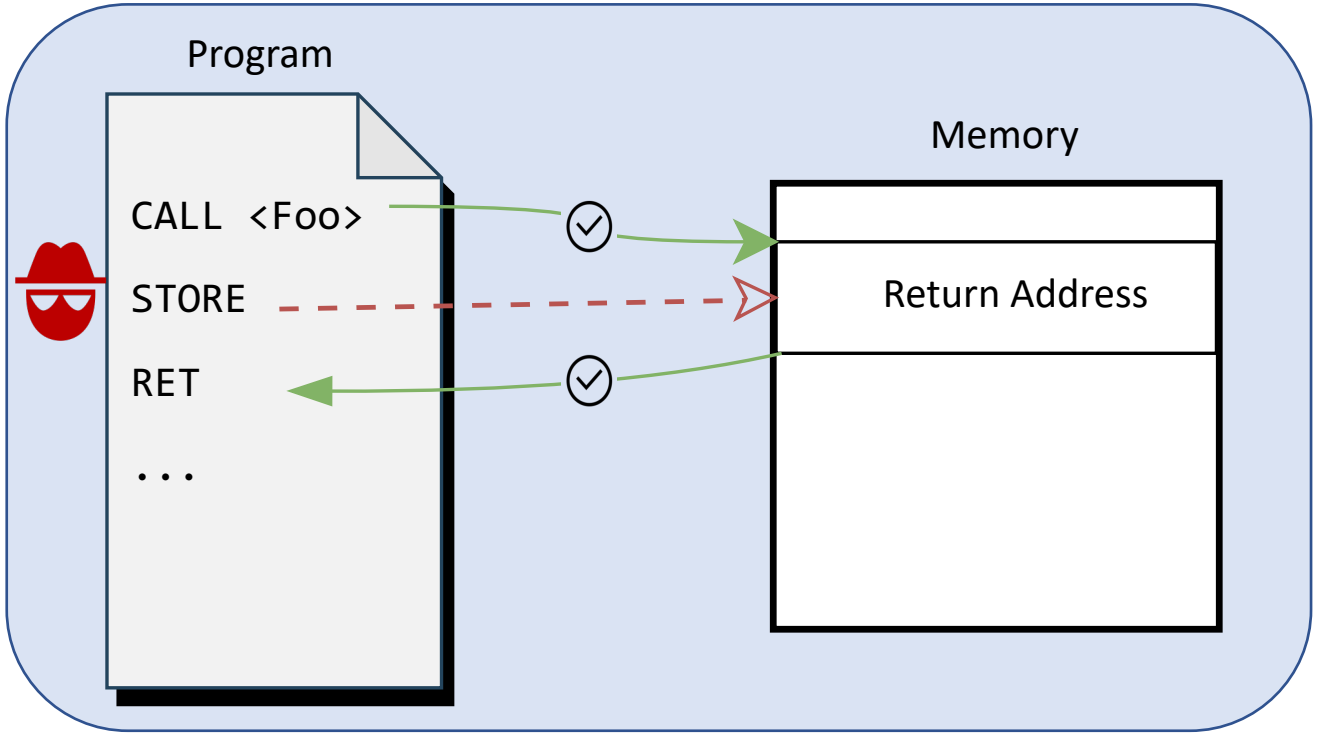
Return Address Protection

ZeRØ: Overview



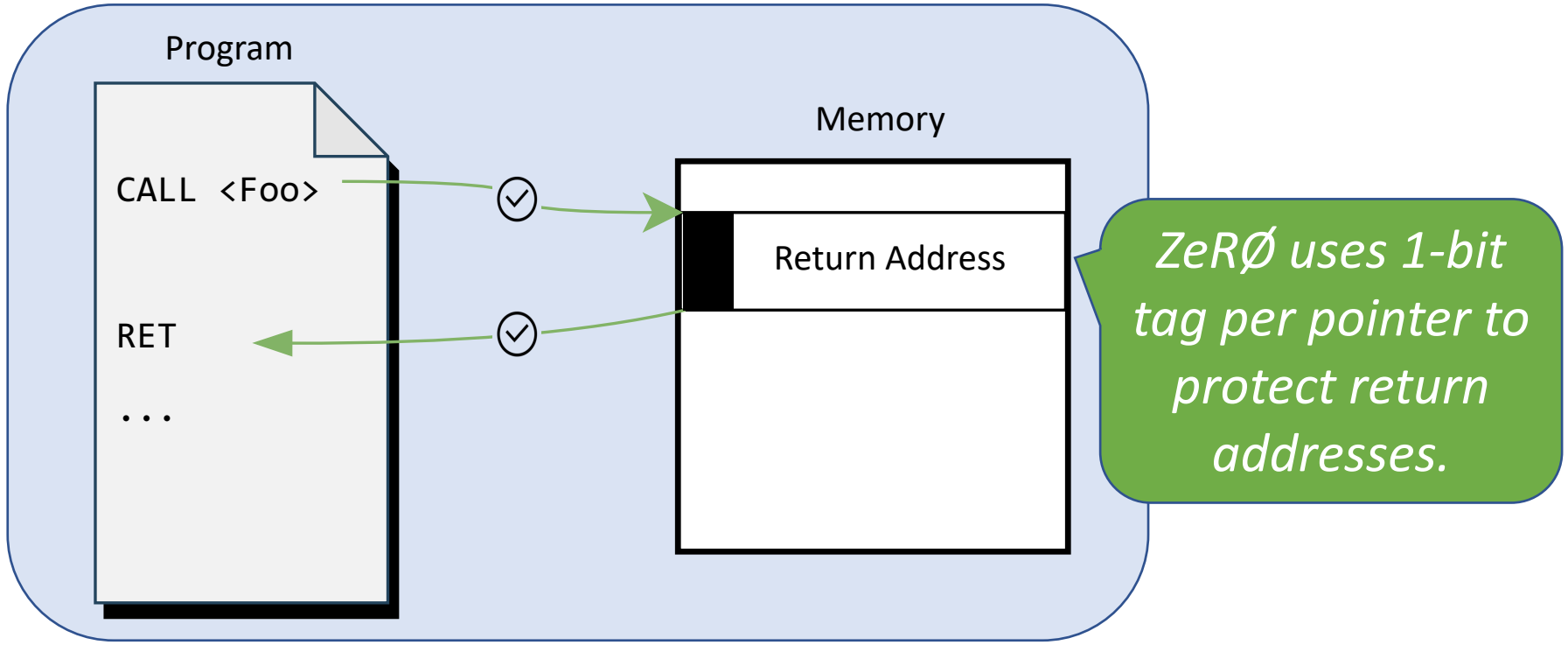
Return Address Protection

ZeRØ: Overview



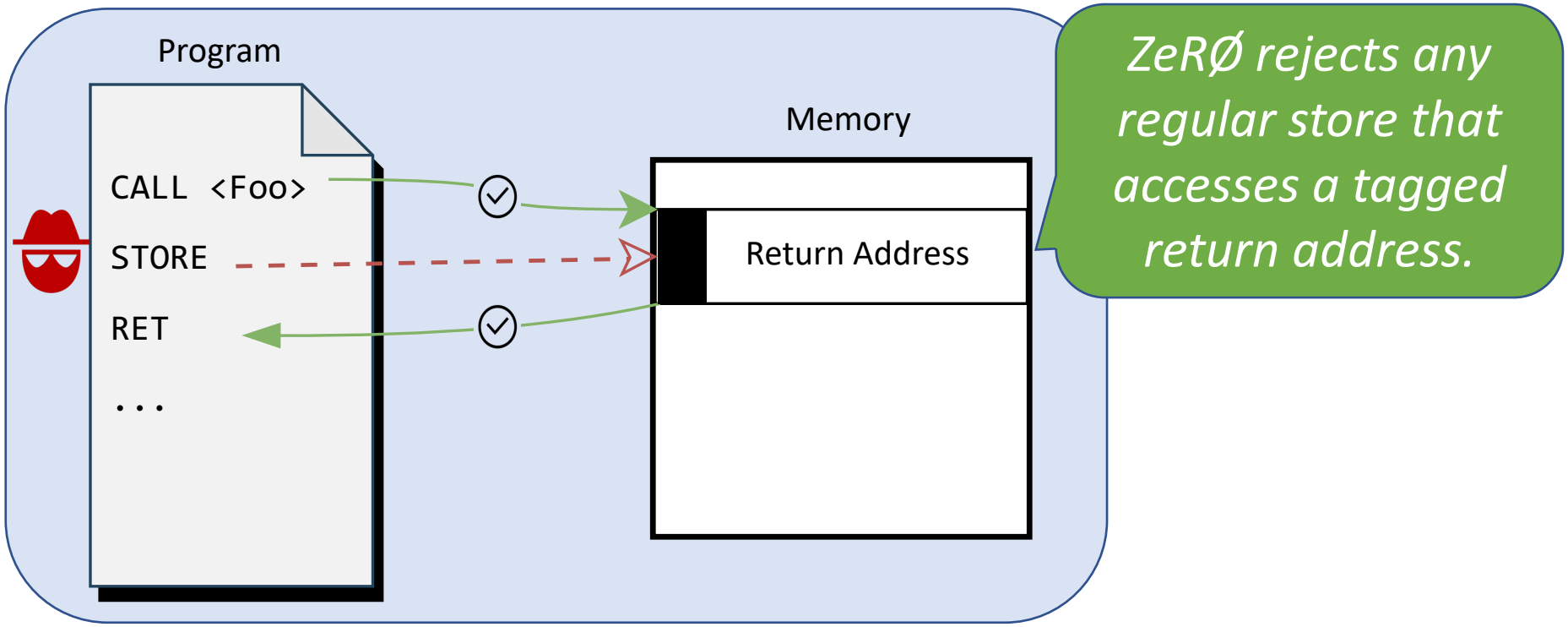
Return Address Protection

ZeRØ: Overview



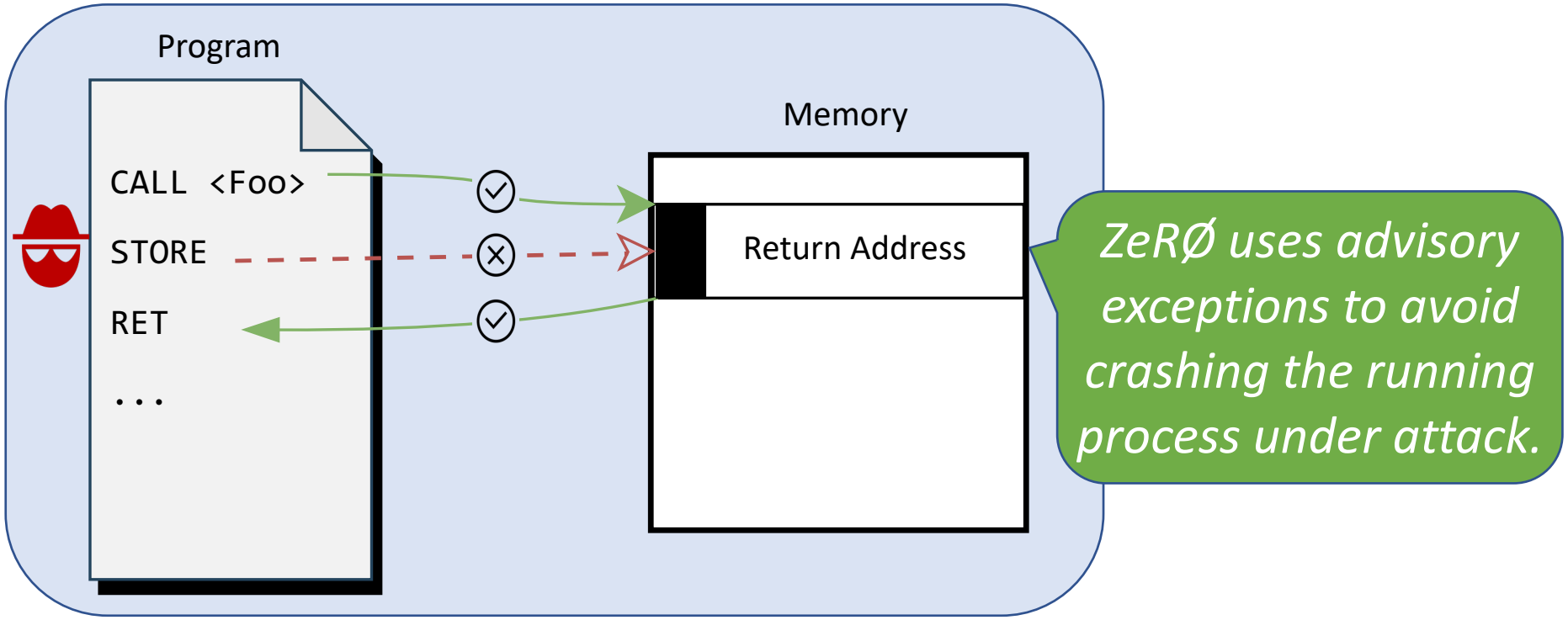
Return Address Protection

ZeRØ: Overview



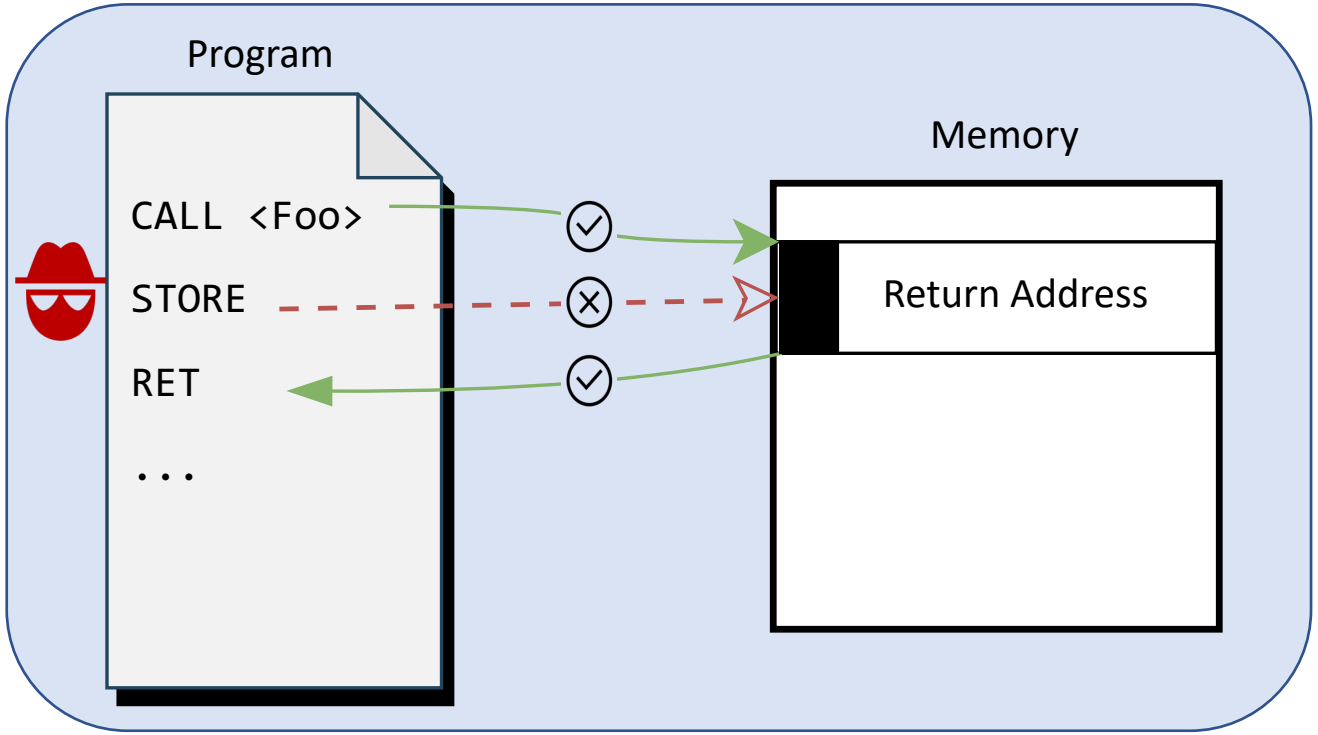
Return Address Protection

ZeRØ: Overview



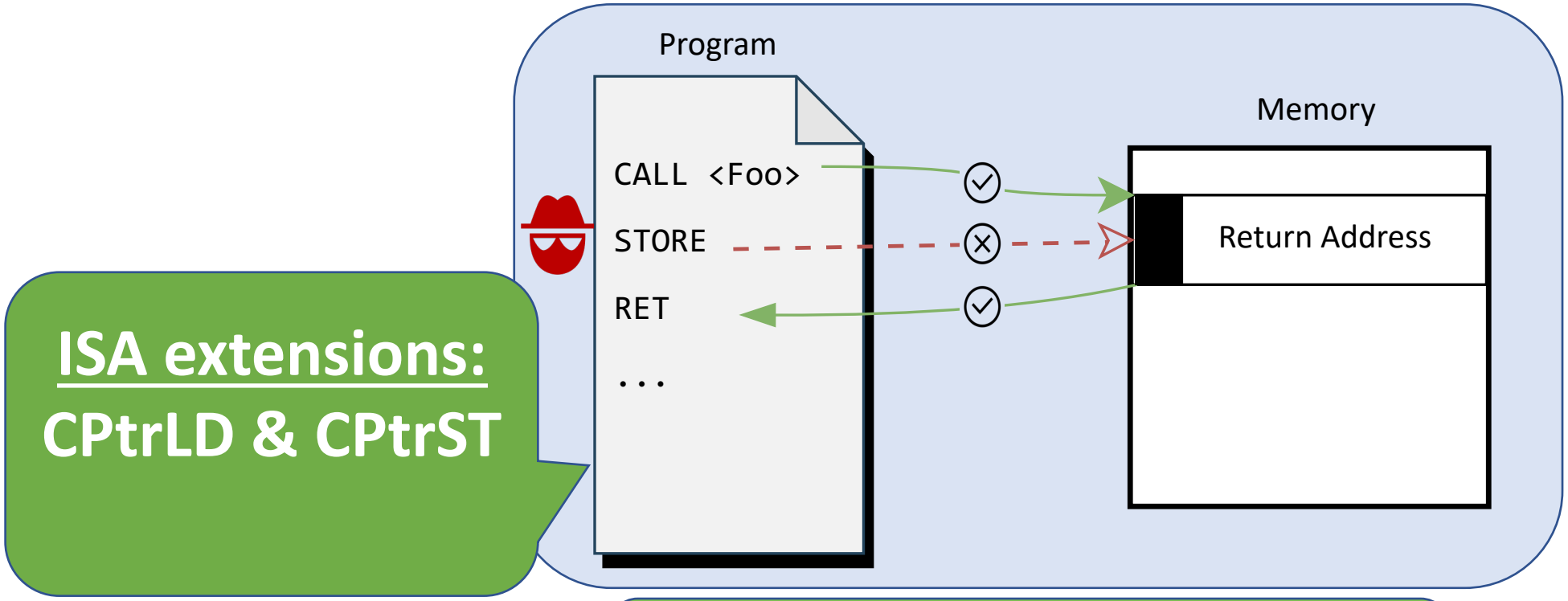
Return Address Protection

ZeRØ: Overview



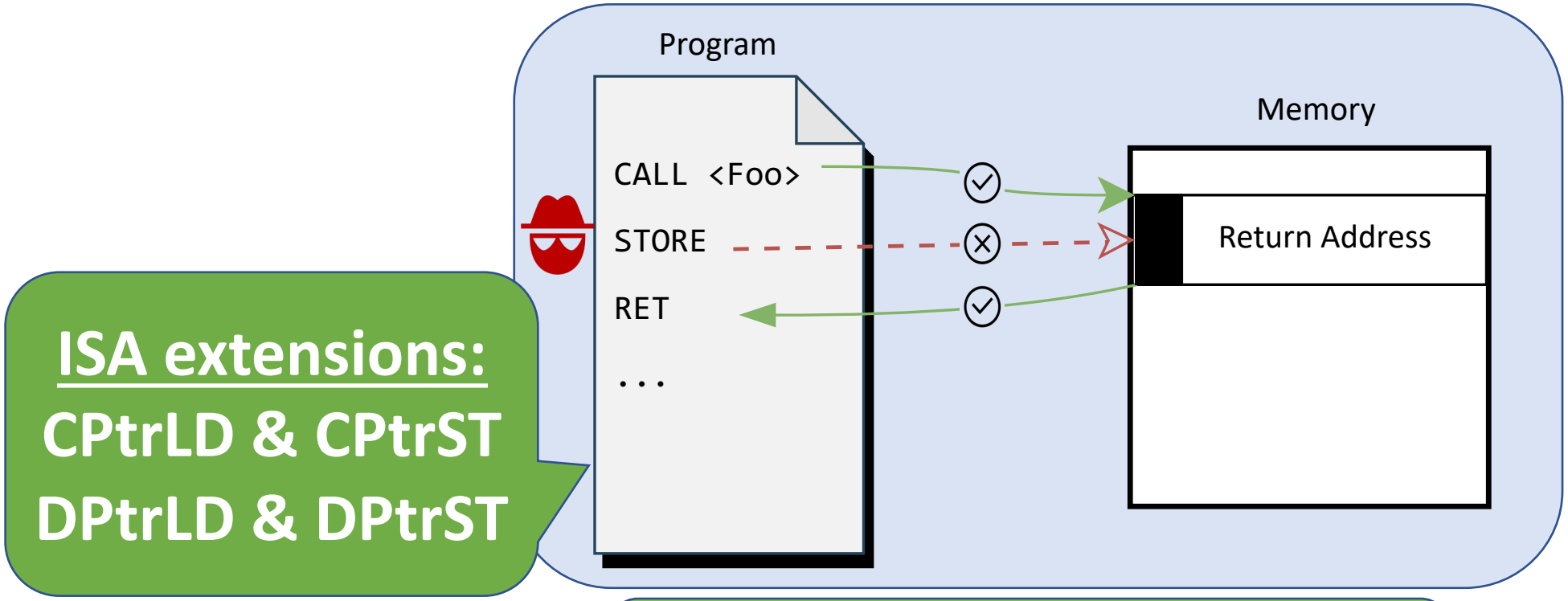
ZeRØ protects return addresses, code pointers, and data pointers.

ZeRØ: Overview



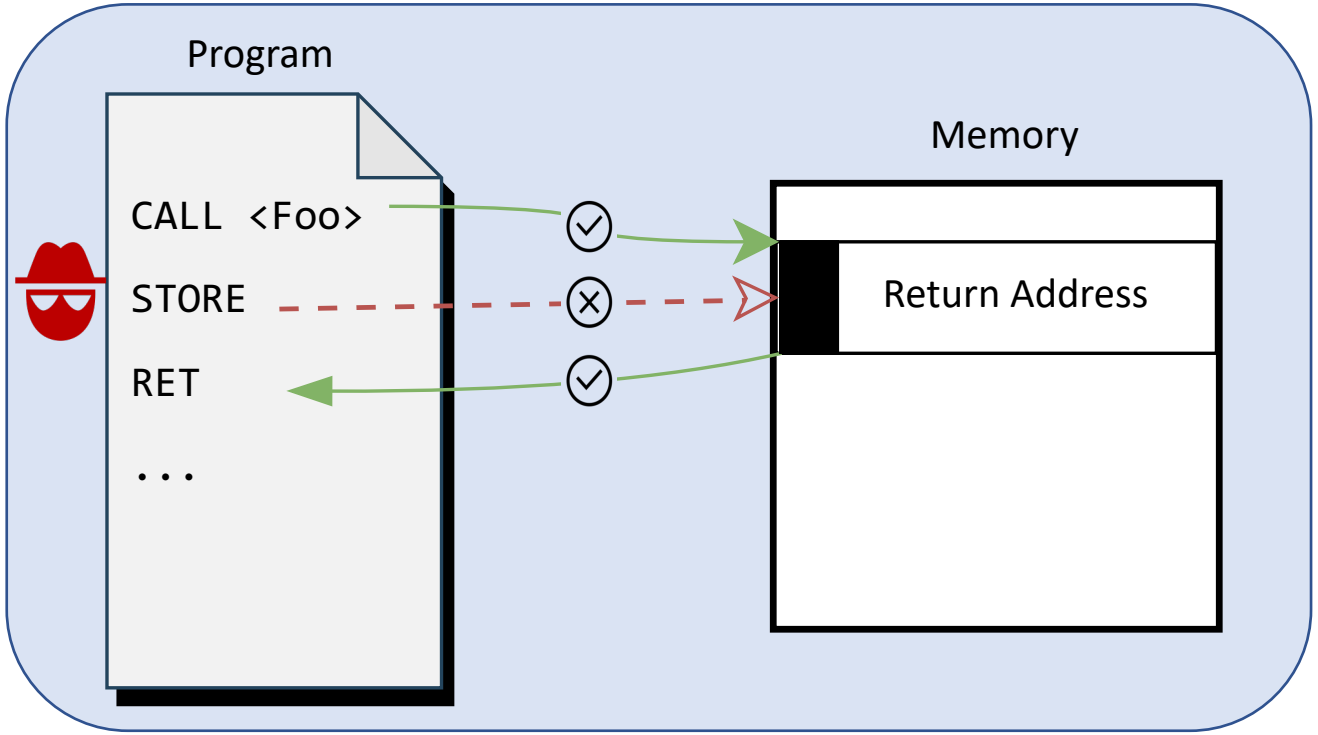
ZeRØ protects return addresses, code pointers, and data pointers.

ZeRØ: Overview



ZeRØ protects return addresses, code pointers, and data pointers.

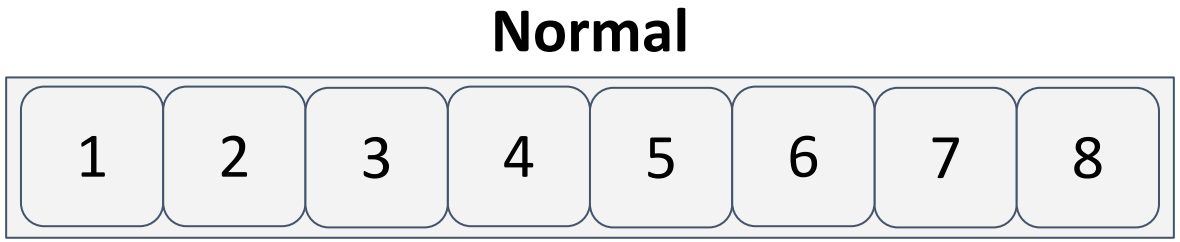
ZeRØ: Overview



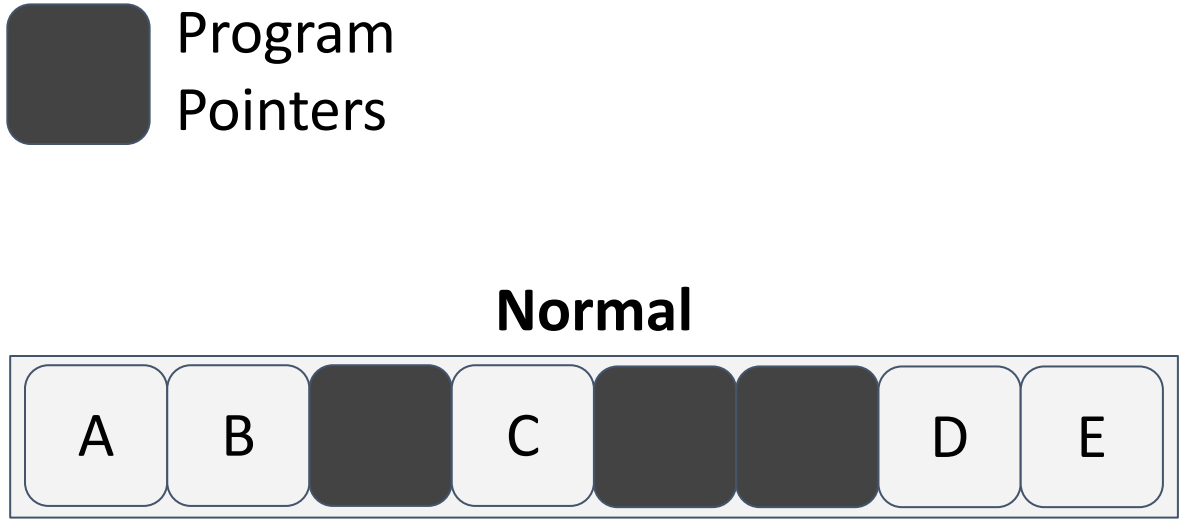
How can ZeRØ efficiently identify if a memory word is a return address, code pointer, data pointer, or regular data?

Cache Line Formats

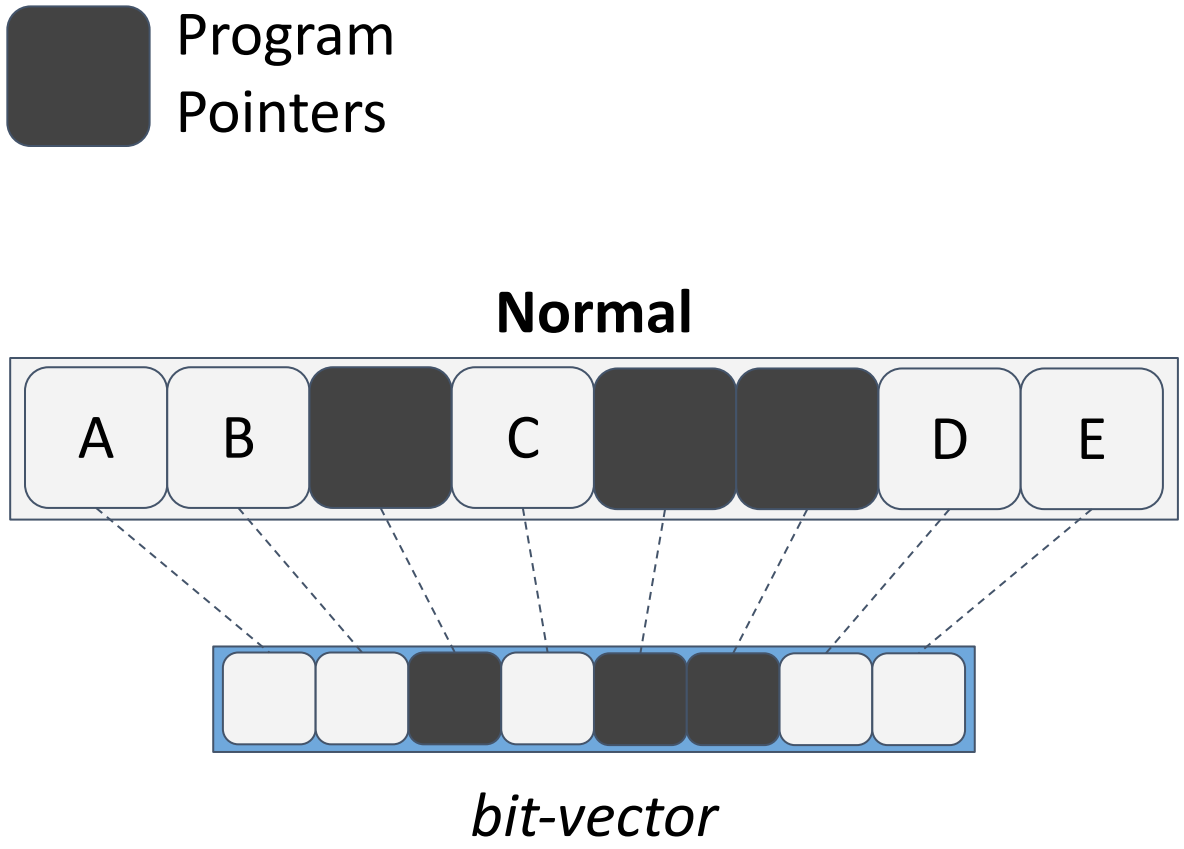
ZeRØ: Cache Line Formats



ZeRØ: Cache Line Formats

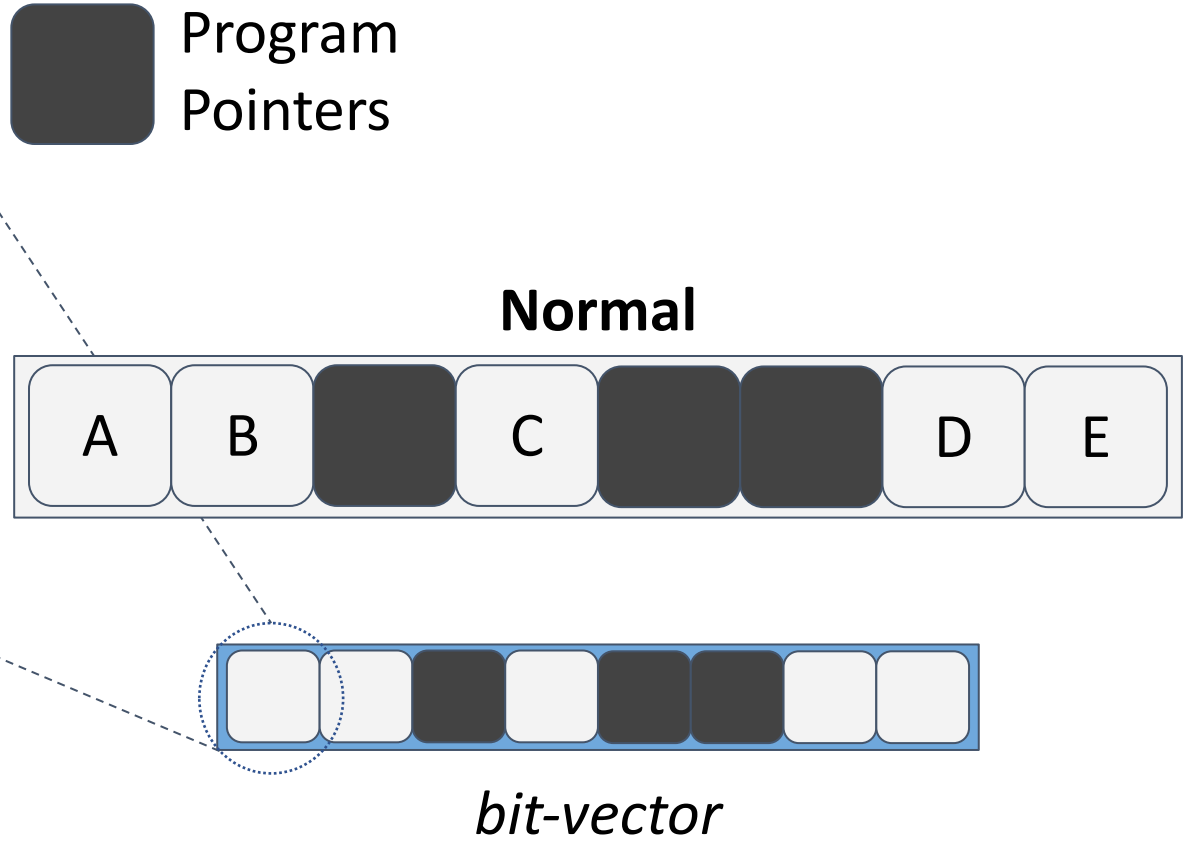


ZeRØ: Cache Line Formats



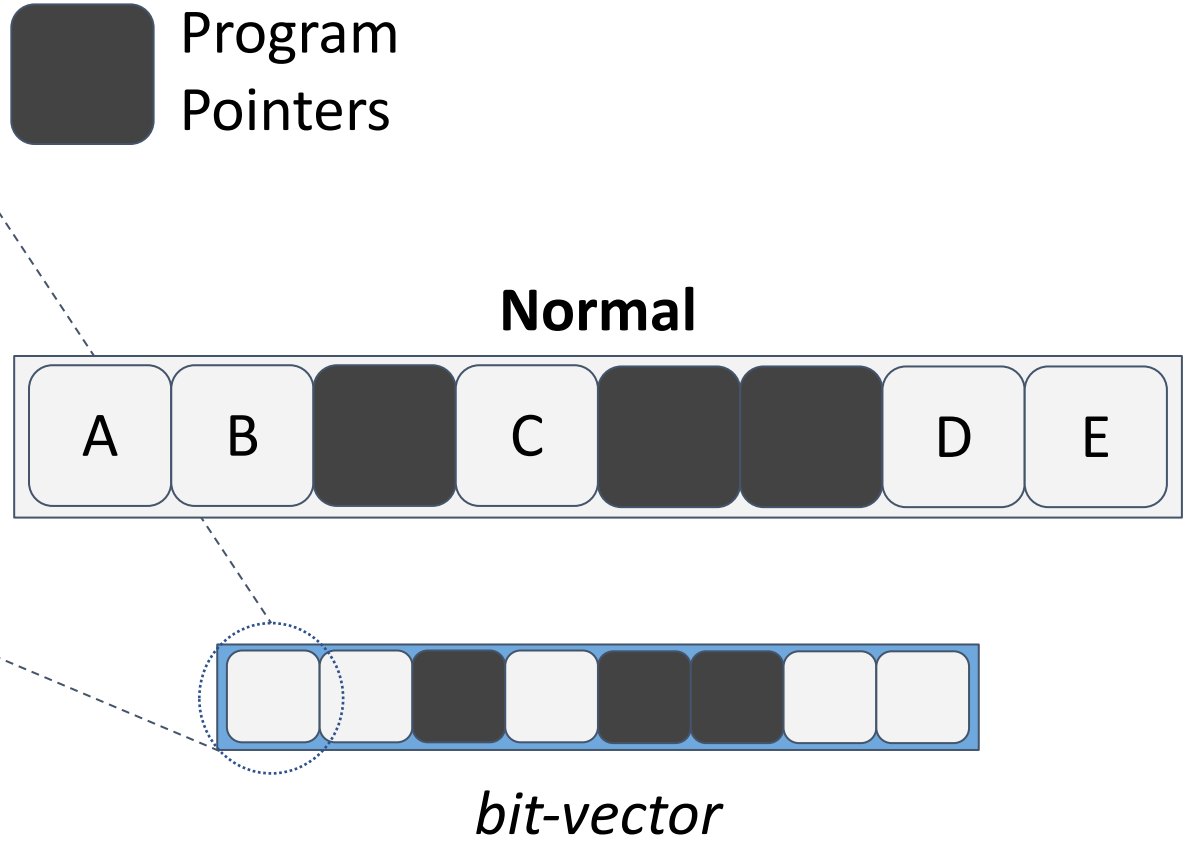
ZeRØ: Cache Line Formats

Type	Bits
Return address	01



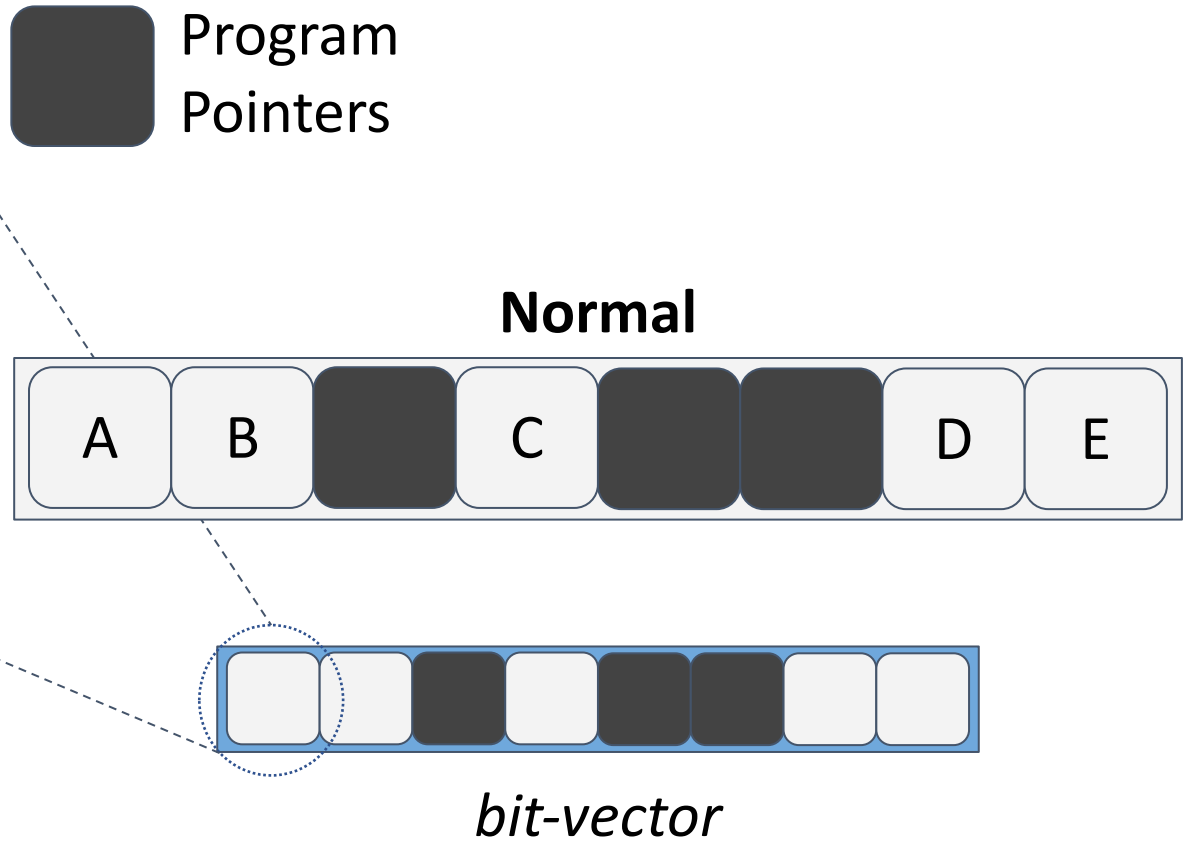
ZeRØ: Cache Line Formats

Type	Bits
Return address	01
Function pointer	10



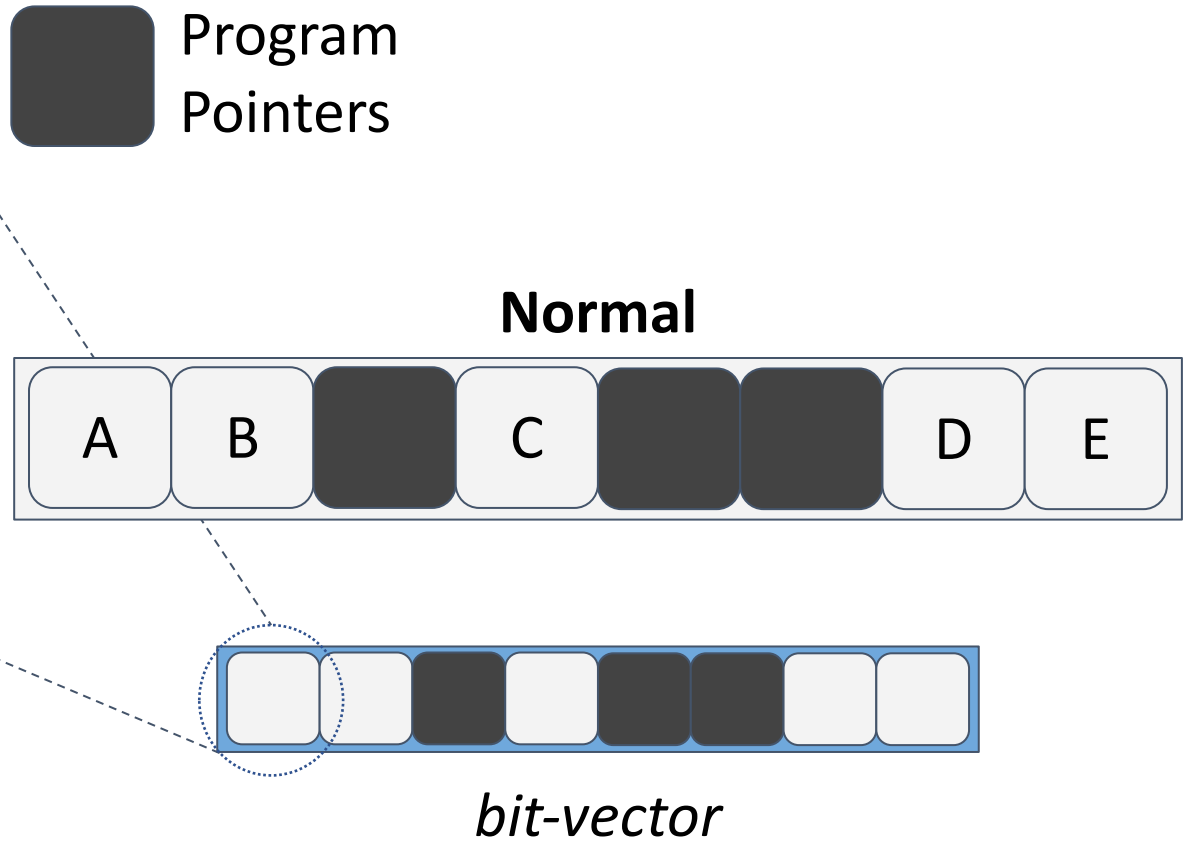
ZeRØ: Cache Line Formats

Type	Bits
Return address	01
Function pointer	10
Data pointer	11

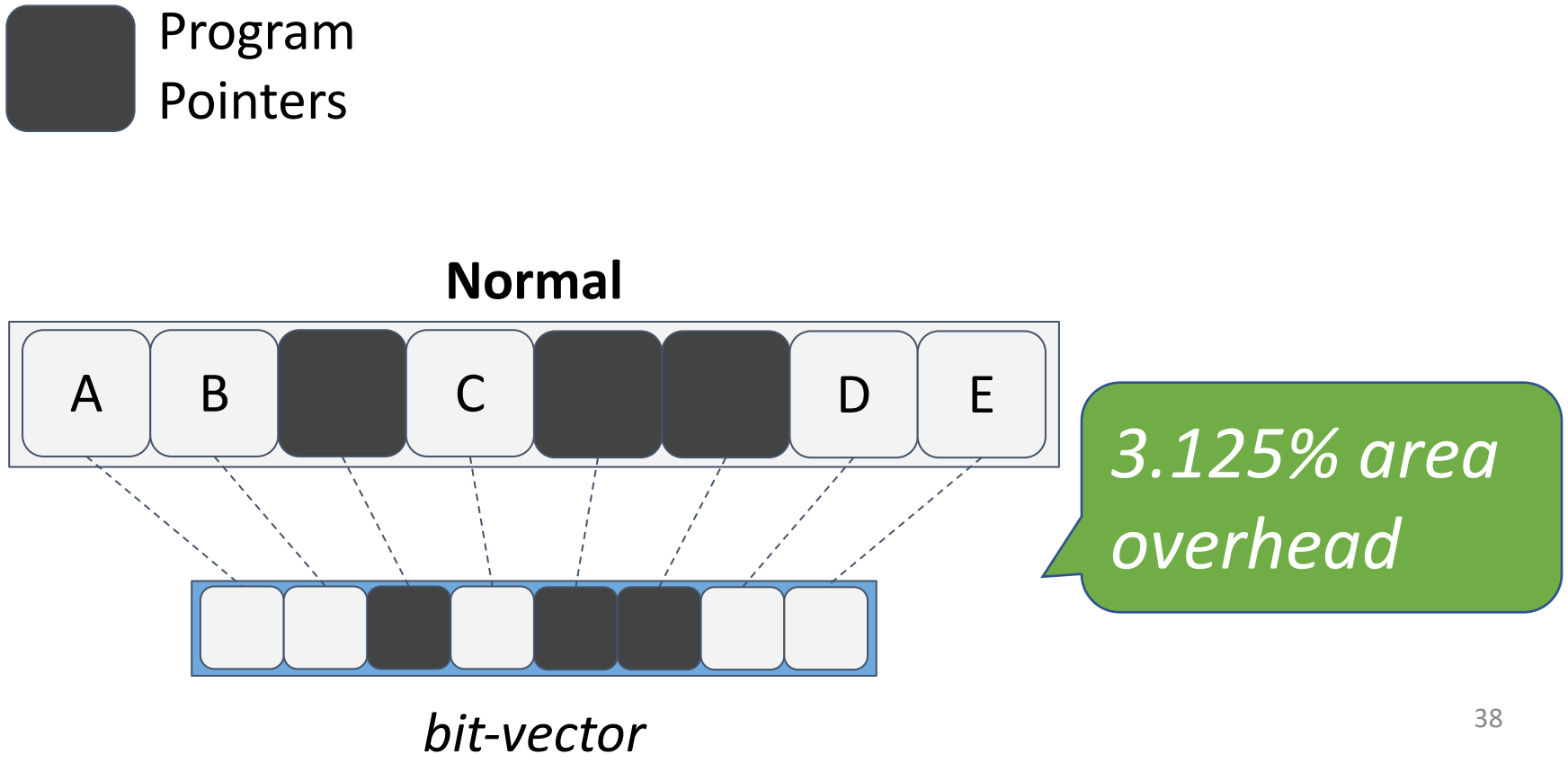


ZeRØ: Cache Line Formats

Type	Bits
Regular data	00
Return address	01
Function pointer	10
Data pointer	11



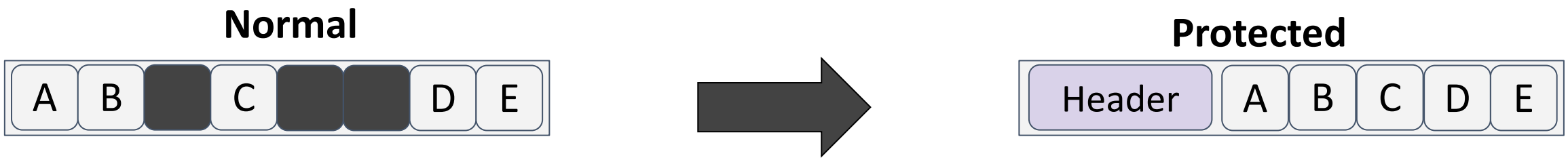
ZeRØ: Cache Line Formats



ZeRØ: Cache Line Formats

Our Metadata: Encoded within unused pointer bits.

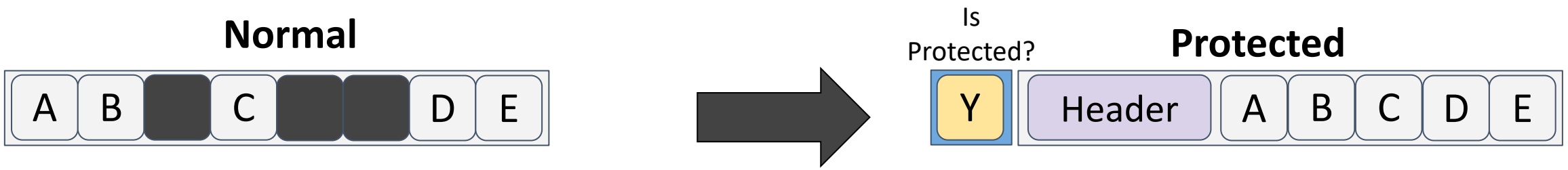
Program Pointers



ZeRØ: Cache Line Formats

Our Metadata: Encoded within unused pointer bits.

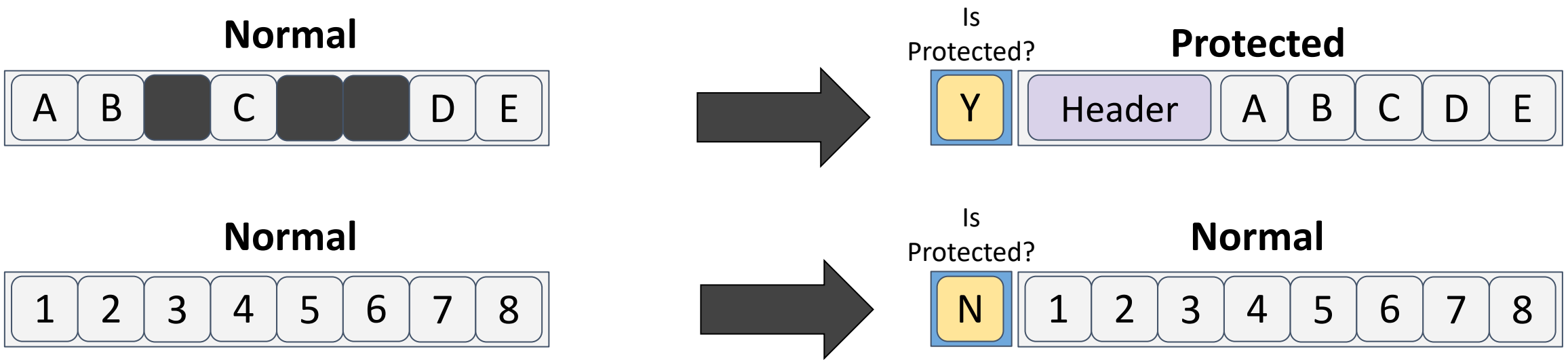
Program Pointers



ZeRØ: Cache Line Formats

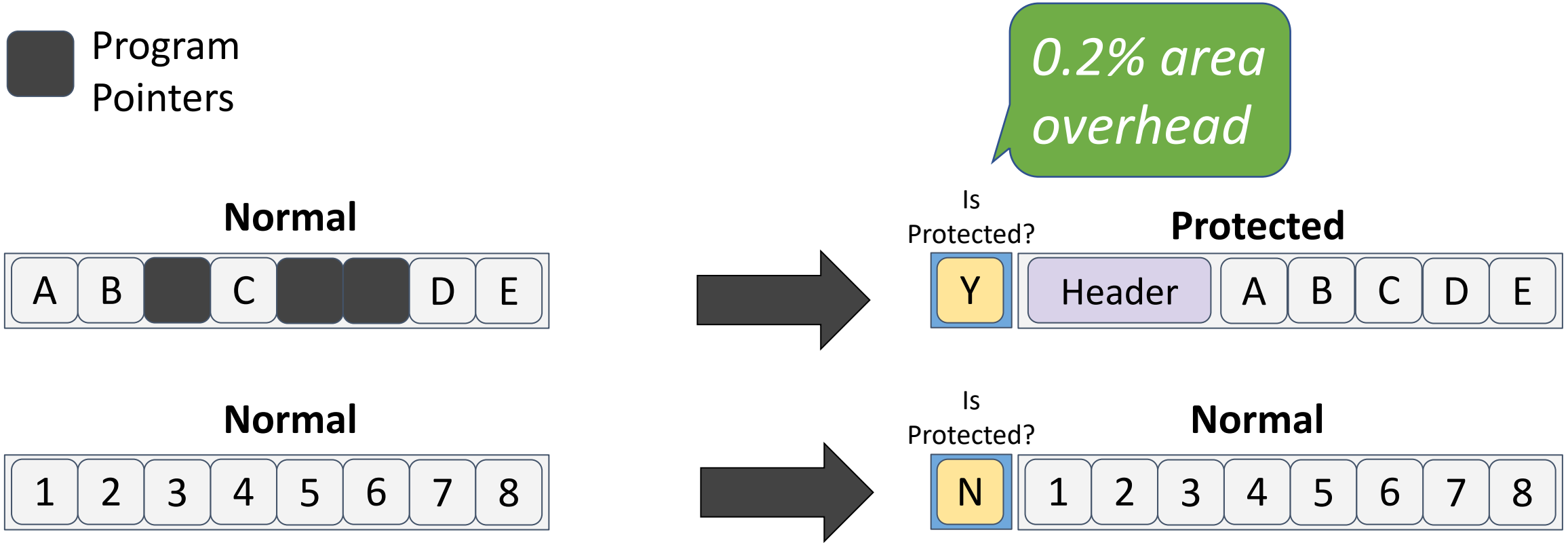
Our Metadata: Encoded within unused pointer bits.

Program Pointers



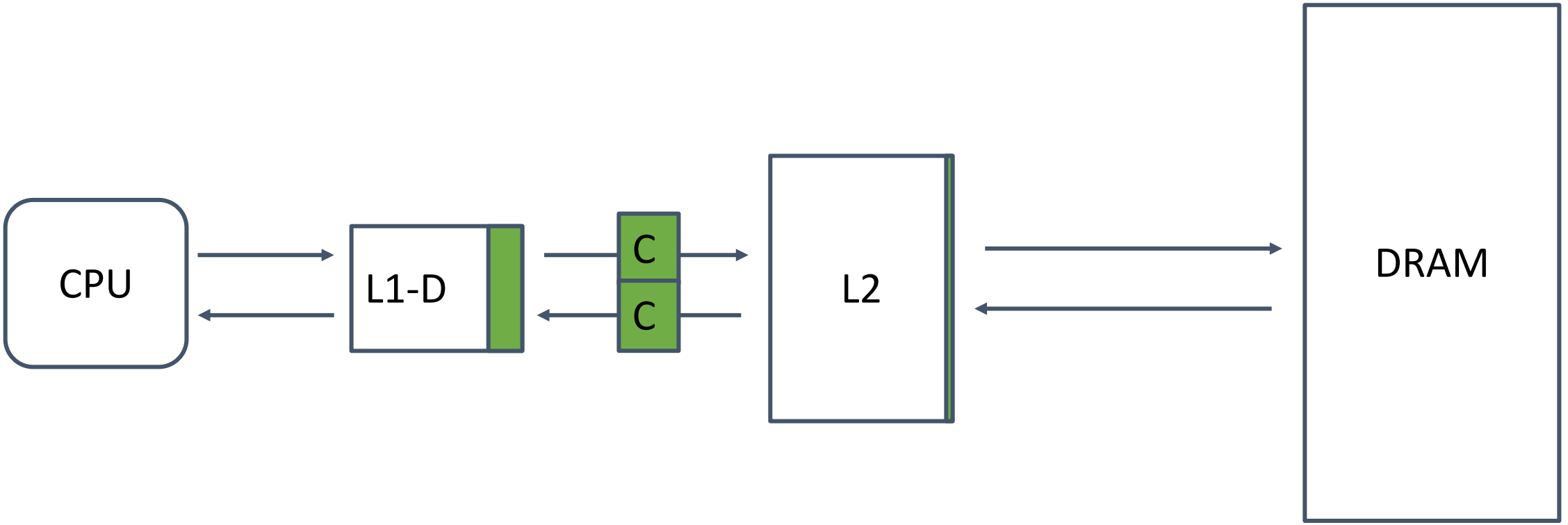
ZeRØ: Cache Line Formats

Our Metadata: Encoded within unused pointer bits.

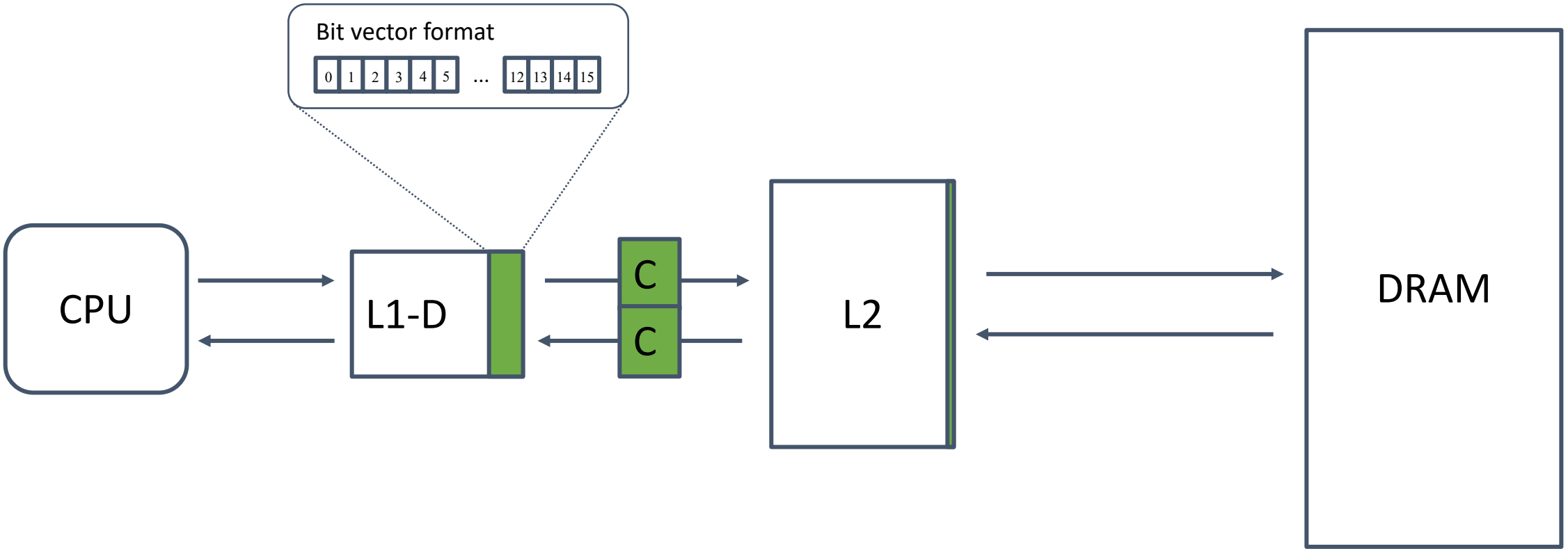


Microarchitectural Overview

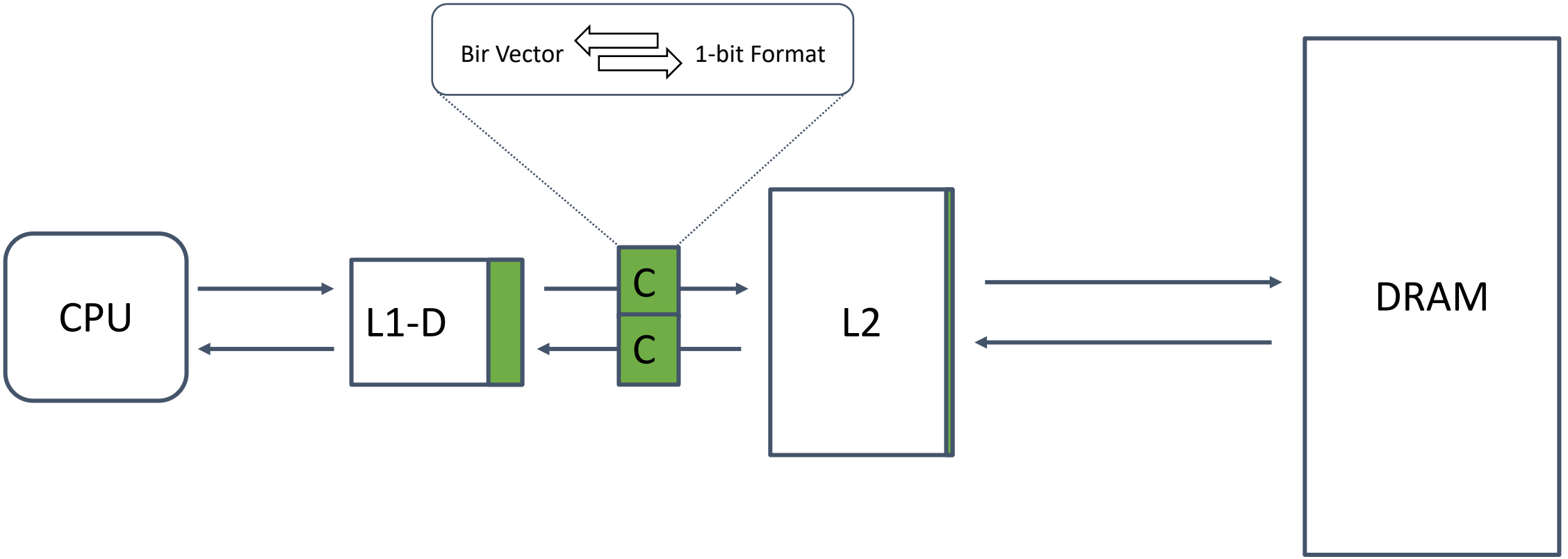
ZeRØ: Microarchitectural Overview



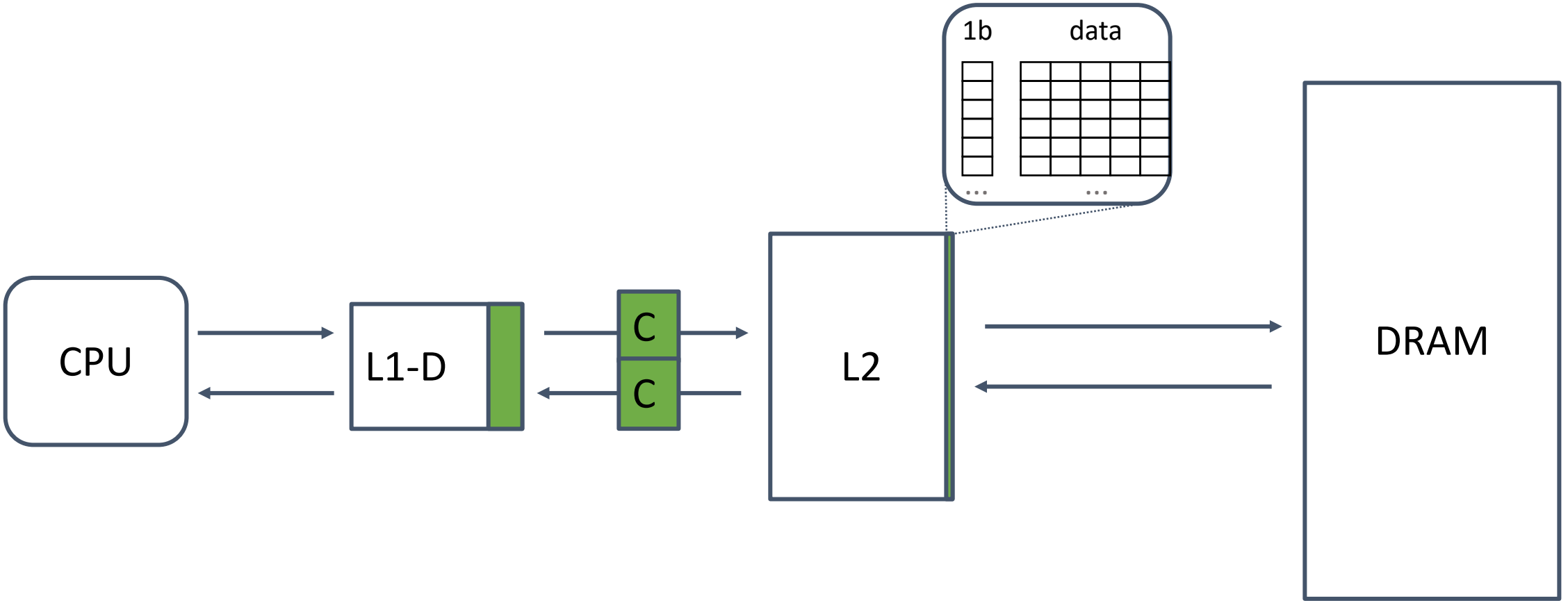
ZeRØ: Microarchitectural Overview



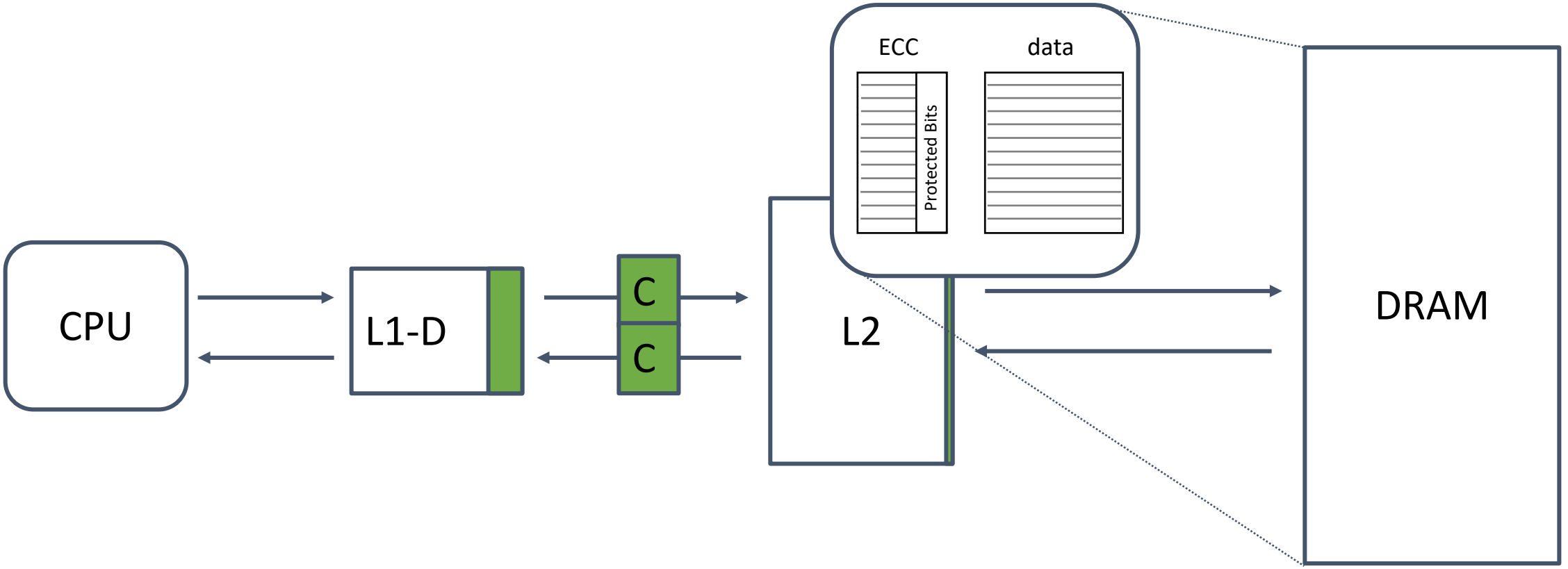
ZeRØ: Microarchitectural Overview



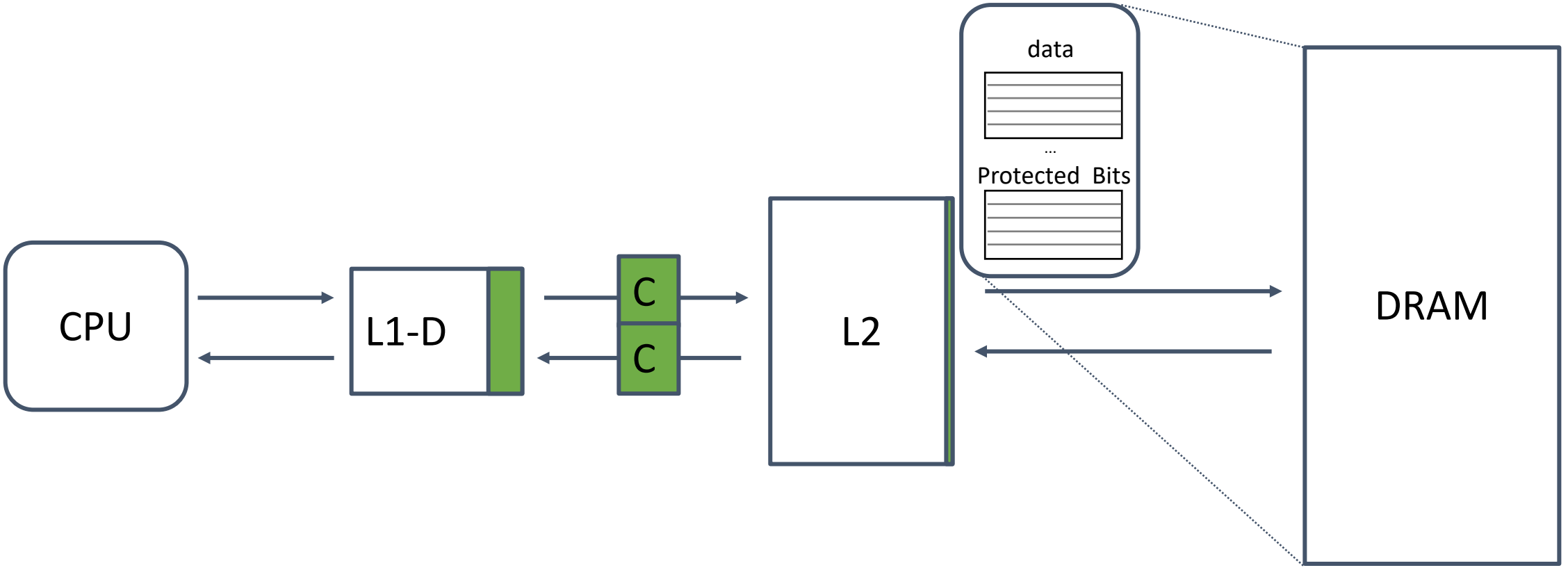
ZeRØ: Microarchitectural Overview



ZeRØ: Microarchitectural Overview



ZeRØ: Microarchitectural Overview



ISA Extensions

ZeRØ: ISA Extensions

```
CPtrST/CPtrLD Address, Value
```

```
DPtrST/DPtrLD Address, Value
```

ZeRØ: ISA Extensions

`CPtrST/CPtrLD Address, Value`

`DPtrST/DPtrLD Address, Value`

Same Layout as regular Loads/Stores

ZeRØ: ISA Extensions

```
CPtrST/CPtrLD Address, Value
```

```
DPtrST/DPtrLD Address, Value
```

```
ClearMeta Address, Mask
```

Only invoked upon free() or delete()

Performance

ZeRØ: Performance Overheads

- Hardware Overheads.

- Software Overheads.

ZeRØ: Performance Overheads

- Hardware Overheads.
  - Our hardware measurements show that ZeRØ has minimal latency/area/power overheads.

- Software Overheads.

ZeRØ: Performance Overheads

- Hardware Overheads.
  - Our hardware measurements show that ZeRØ has minimal latency/area/power overheads.

- Software Overheads.
  - Our special loads/stores do not change binary size.

ZeRØ: Performance Overheads

- Hardware Overheads.
  - Our hardware measurements show that ZeRØ has minimal latency/area/power overheads.

- Software Overheads.
  - Our special loads/stores do not change binary size.
  - The ClearMeta instructions are only called upon memory deallocation.

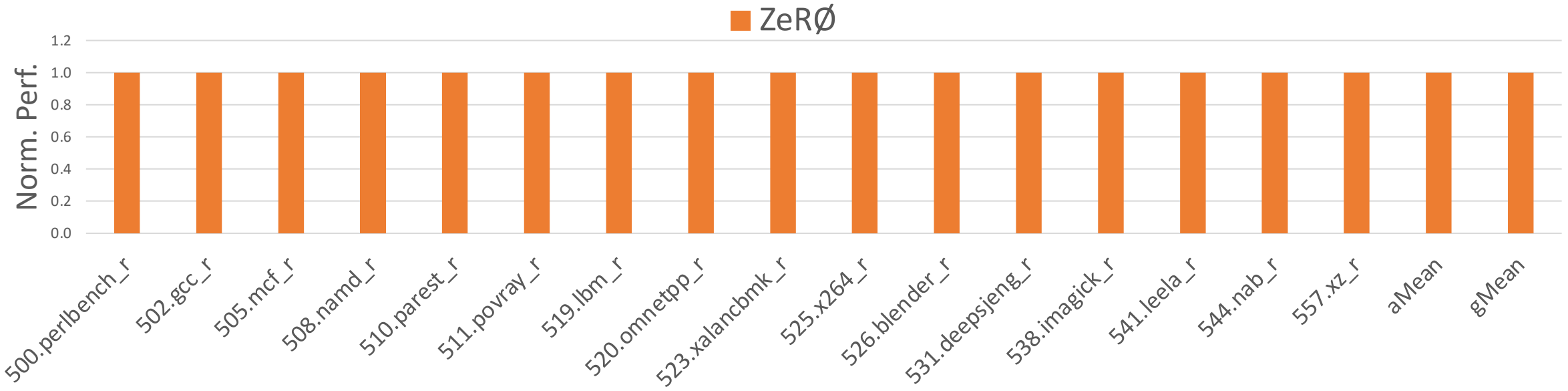
ZeRØ: Performance Overheads

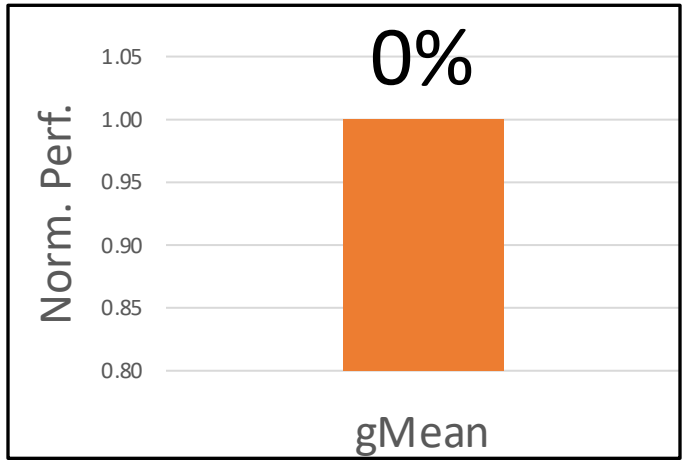
- Hardware Overheads.
  - Our hardware measurements show that ZeRØ has minimal latency/area/power overheads.

- Software Overheads.
  - Our special loads/stores do not change binary size.
  - The ClearMeta instructions are only called upon memory deallocation.

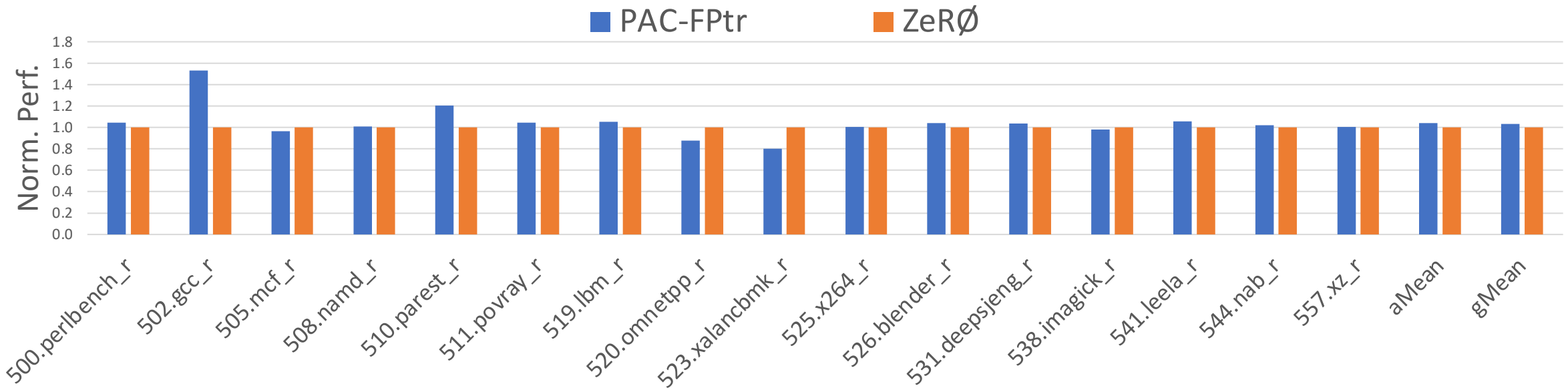
The ClearMeta instructions are emulated on x86_64 using dummy stores

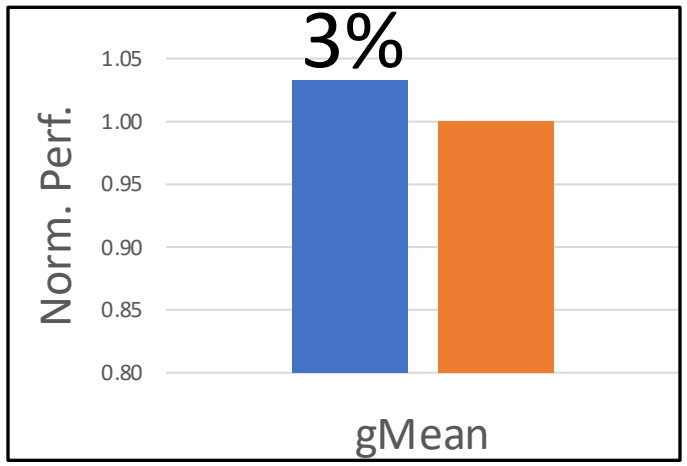
ZeRØ: Performance Results on x86_64



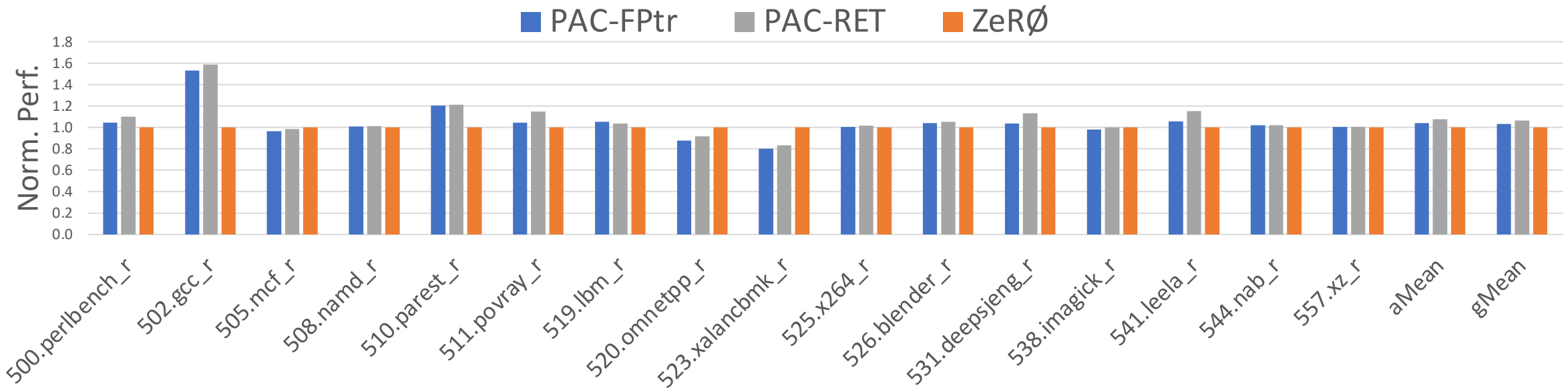


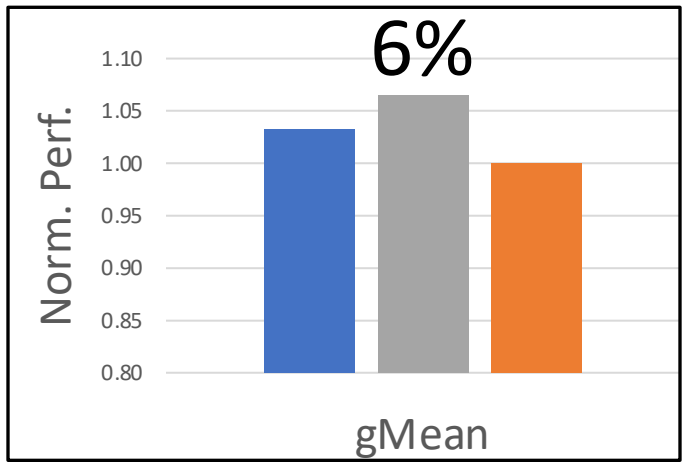
ZeRØ: Performance Results on x86_64



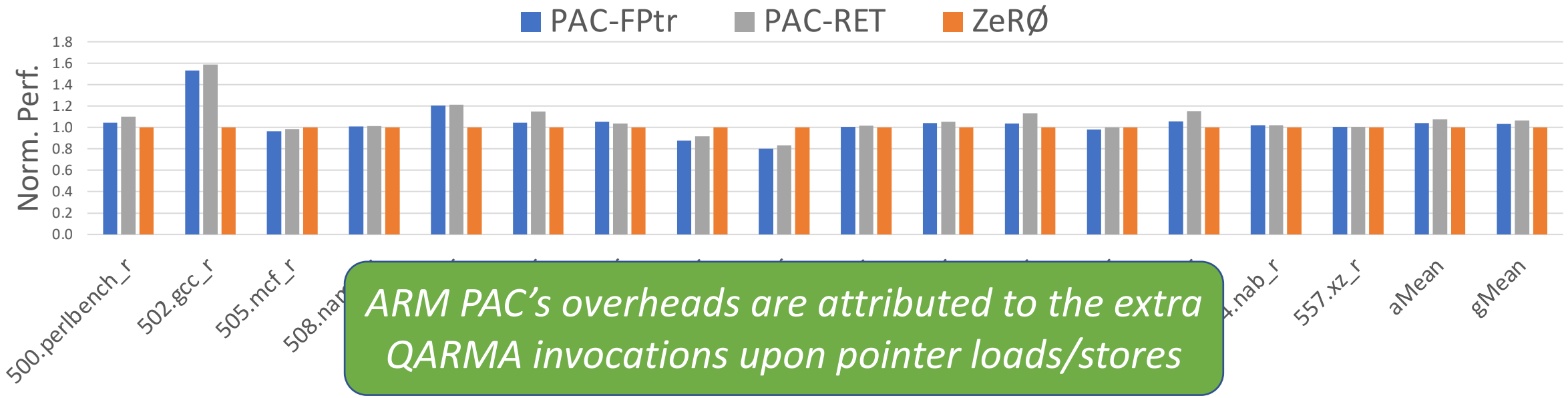


ZeRØ: Performance Results on x86_64

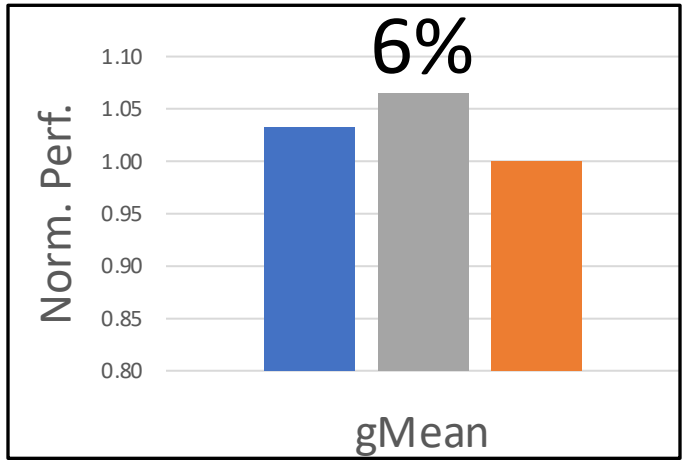




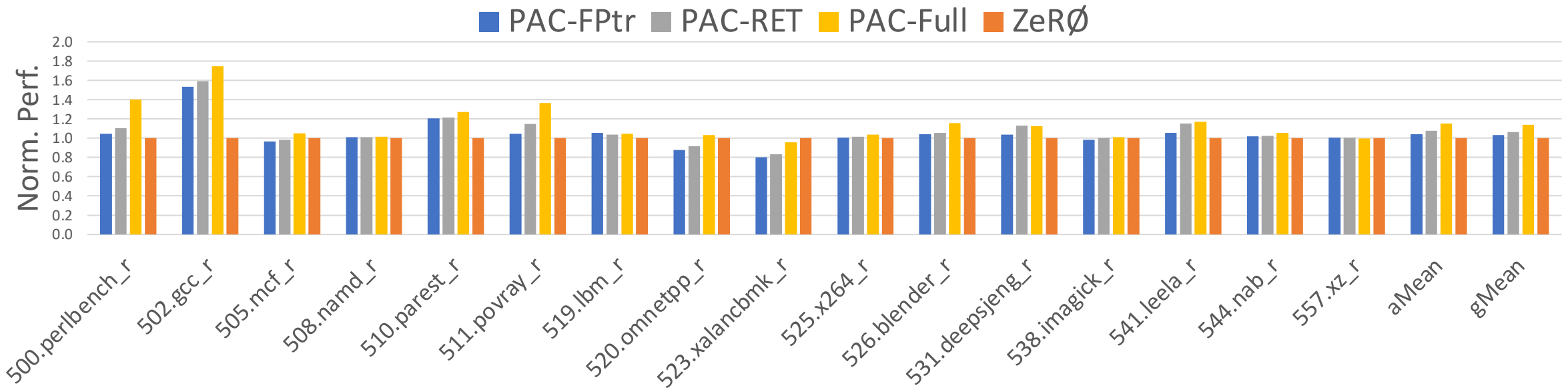
ZeRØ: Performance Results on x86_64

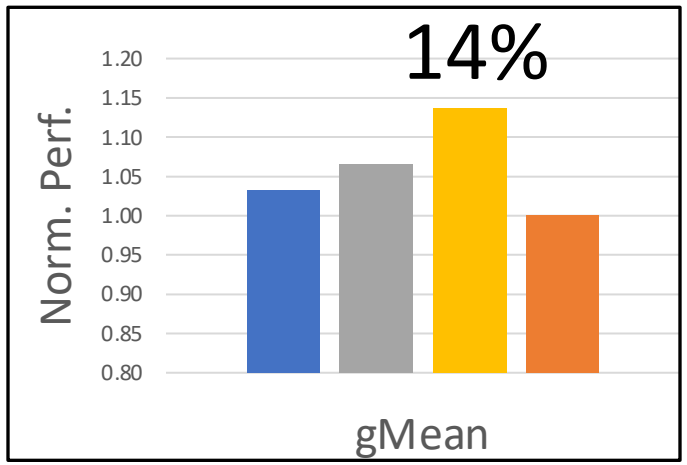


ARM PAC's overheads are attributed to the extra QARMA invocations upon pointer loads/stores

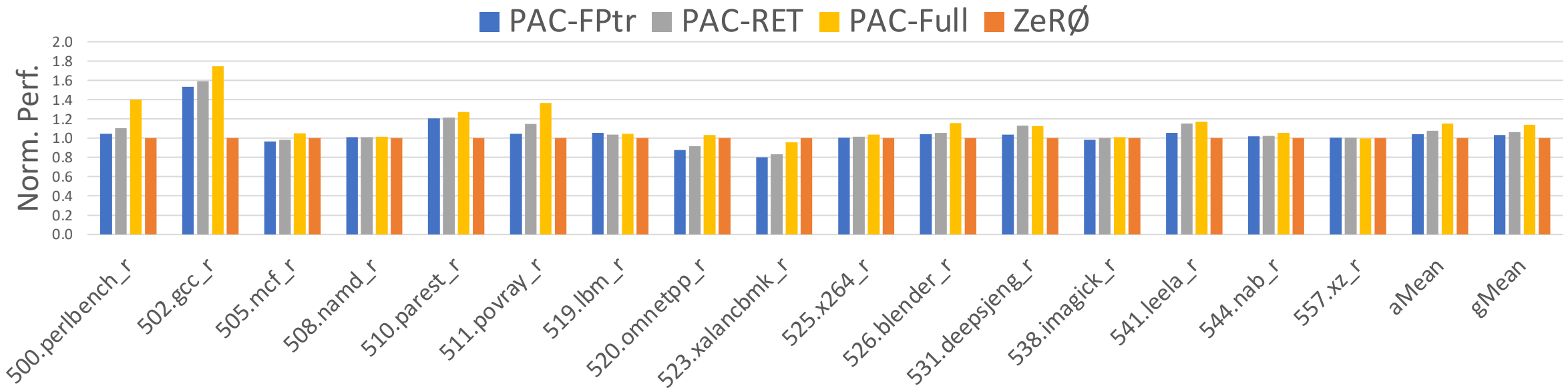


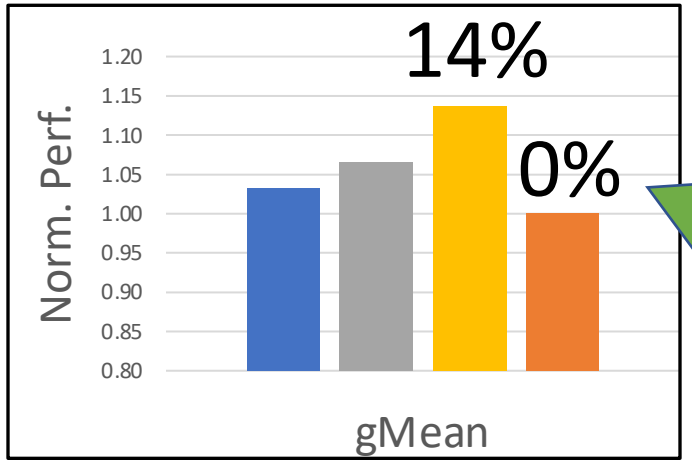
ZeRØ: Performance Results on x86_64





ZeRØ: Performance Results on x86_64





ZeRØ reduces the average runtime overheads of pointer integrity from 14% to 0%

Limitations

ZeRØ: Limitations

- Non-pointer data corruption attacks.
  - Require a full memory safety solution.

ZeRØ: Limitations

- Non-pointer data corruption attacks.
  - Require a full memory safety solution.

- Third-party code.
  - Clear the metadata bits before passing pointers to shared libraries.

Conclusion

- ZeRØ provides an efficient pointer integrity mechanism:
  - Is **easy to implement**.
  - Has **no runtime overheads**.
  - Offers **robust security**.

- ZeRØ can be applied to a wide variety of systems:
  - Ranging from servers to mobile devices.