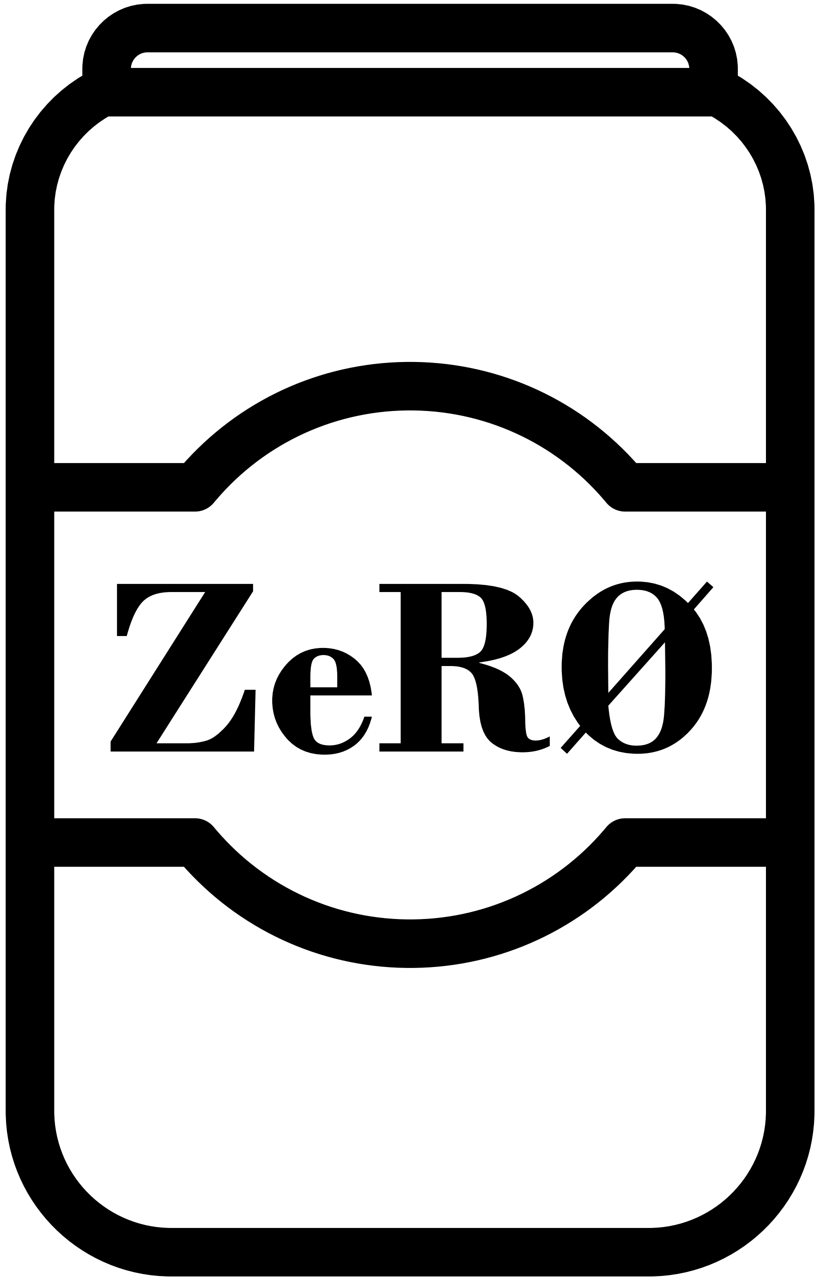


Zero-Overhead Resilient Operation Under Pointer Integrity Attacks

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

CS@CU Computer Science

Columbia University

06/16/2021

Security?

Security!

Inefficient security inconveniences the user

Most end users want security,
but do not want the inconvenience of having it.

Inefficient security inconveniences the user

Slow Performance
User want a snappy experience and security tends to detract from it.

Inefficient security inconveniences the user

Slow Performance
User want a snappy experience and security tends to detract from it.

Energy Drain
Inefficient protections drain precious resources such as battery.

Inefficient security inconveniences the user

Slow Performance
User want a snappy experience and security tends to detract from it.

Energy Drain
Inefficient protections drain precious resources such as battery.

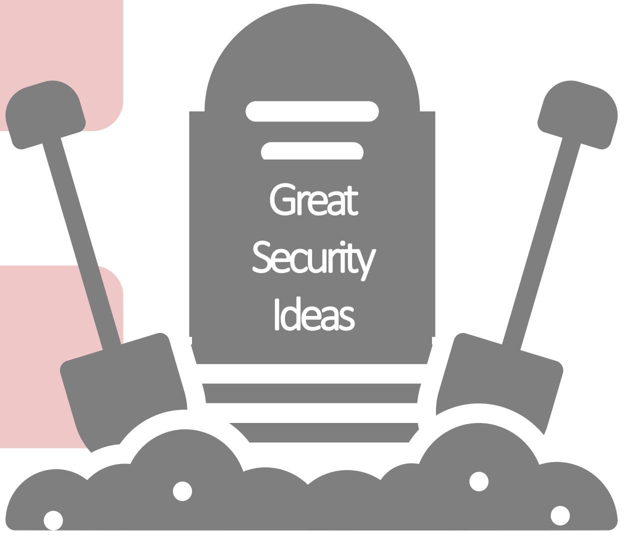
System Stability
Users can’t be bothered with updates and patches.

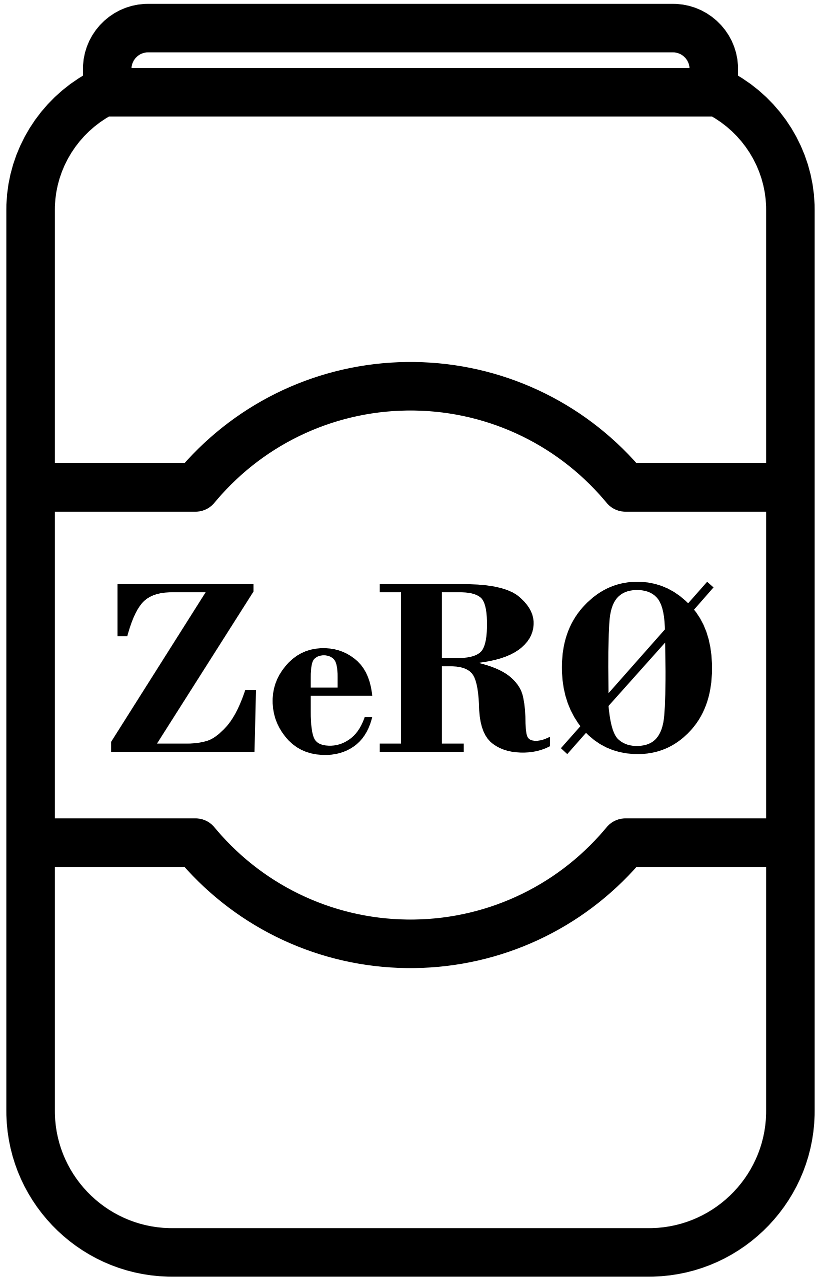
Inefficient security inconveniences the user

Slow Performance
User want a snappy experience and security tends to detract from it.

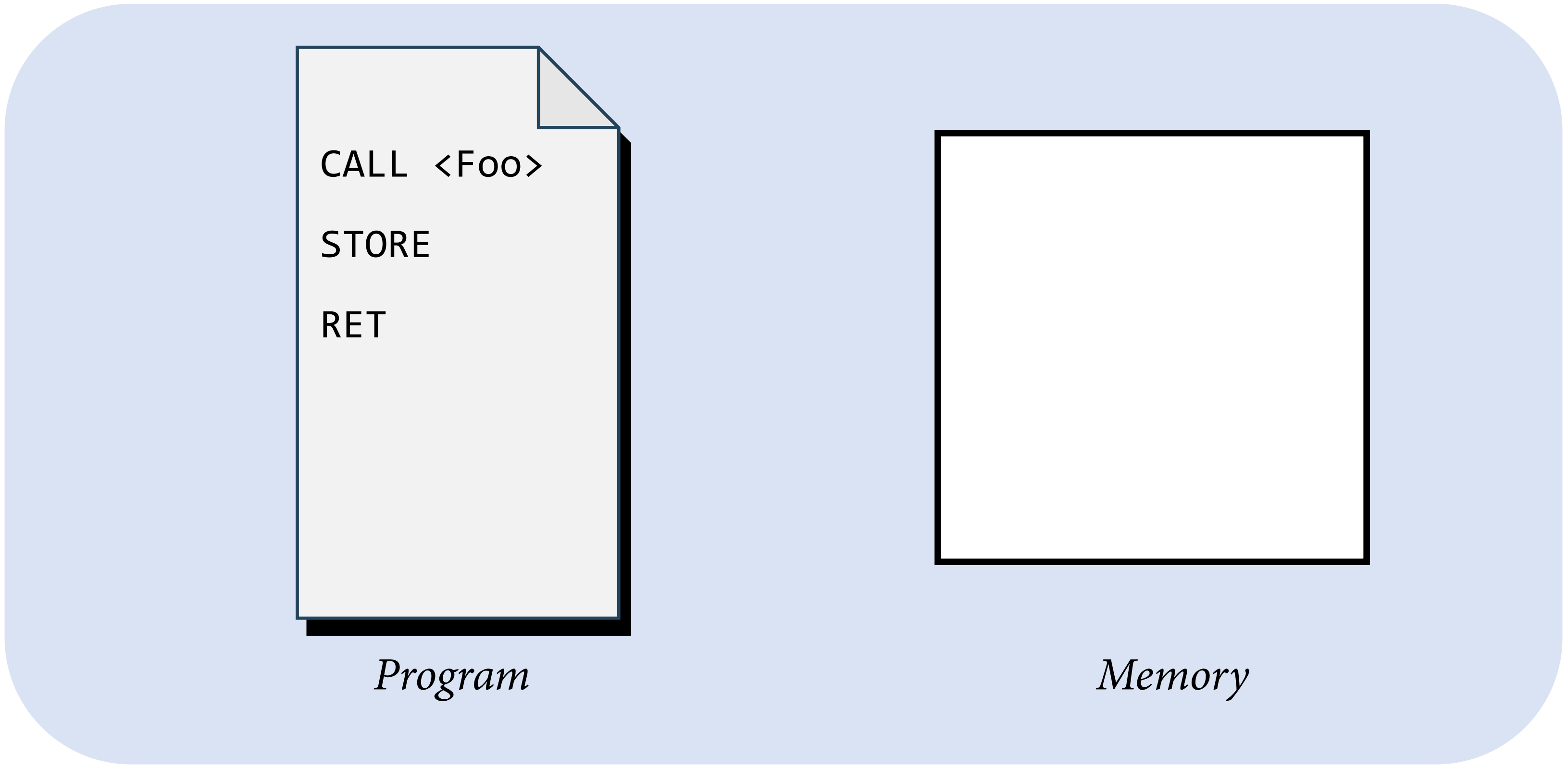
Energy Drain
Inefficient protections drain precious resources such as battery.

System Stability
Users can't be bothered with updates and patches.



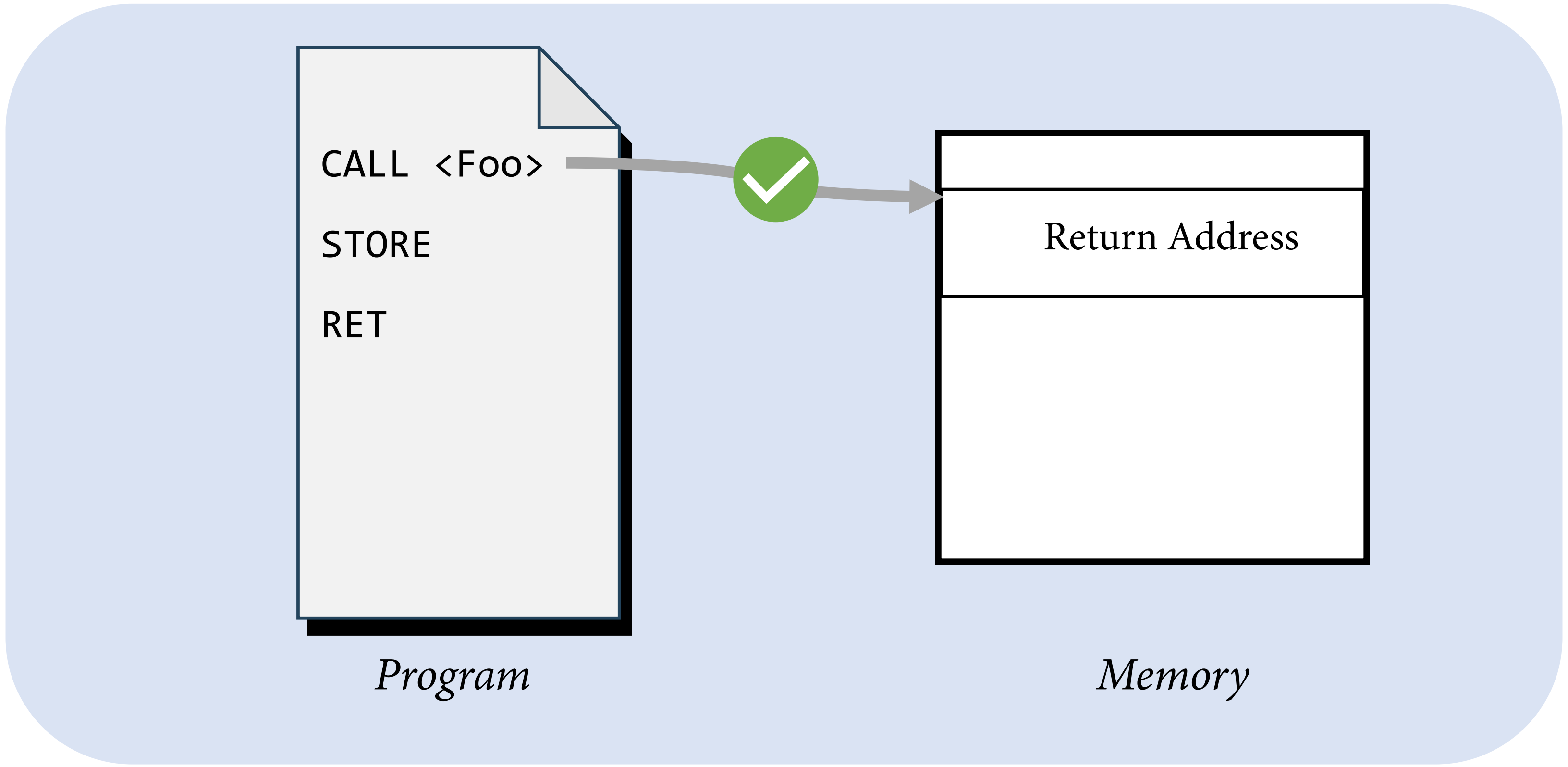

ZeRØ

Return Address Protection

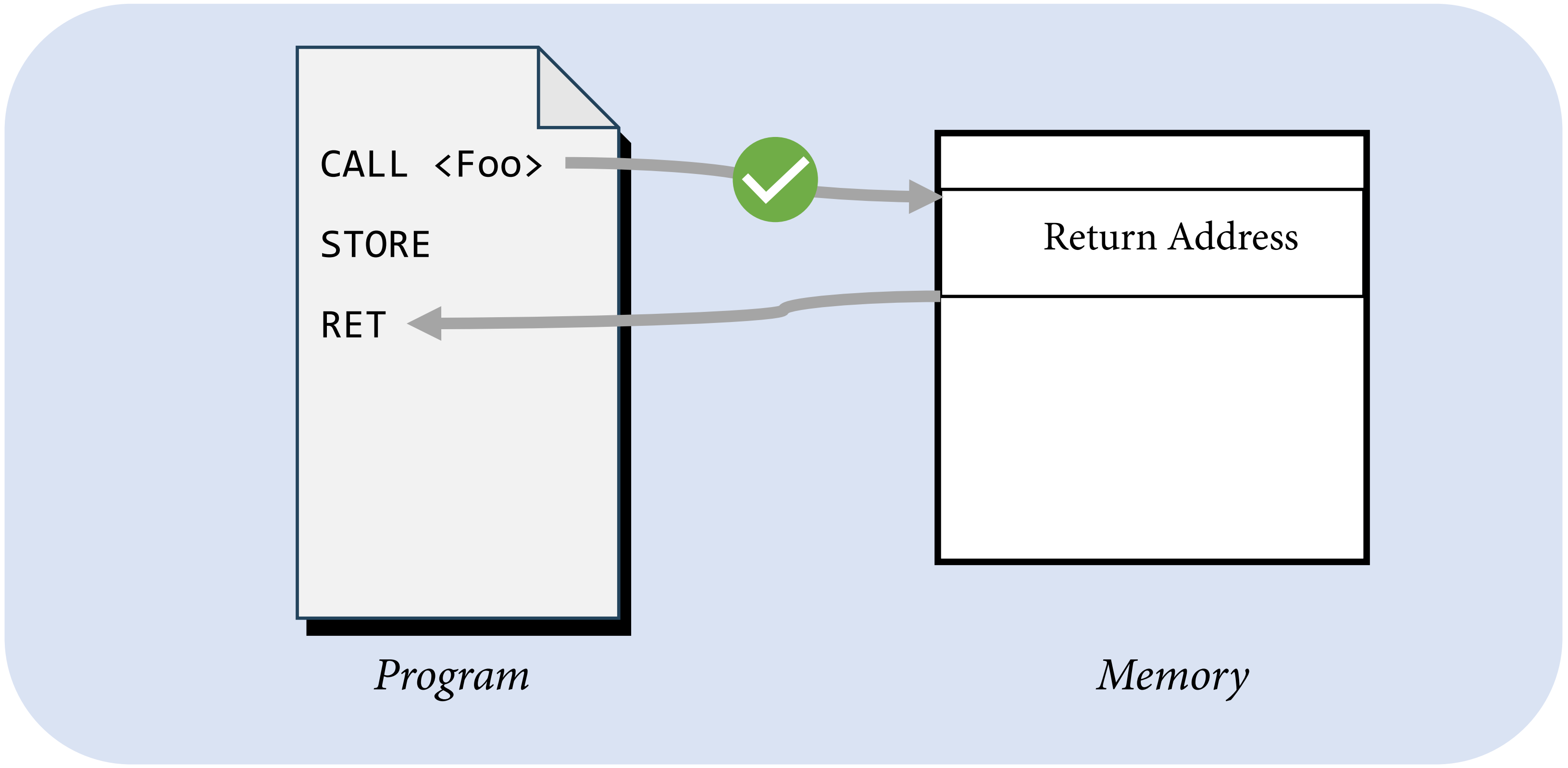


```
CALL <Foo>
STORE
RET
```

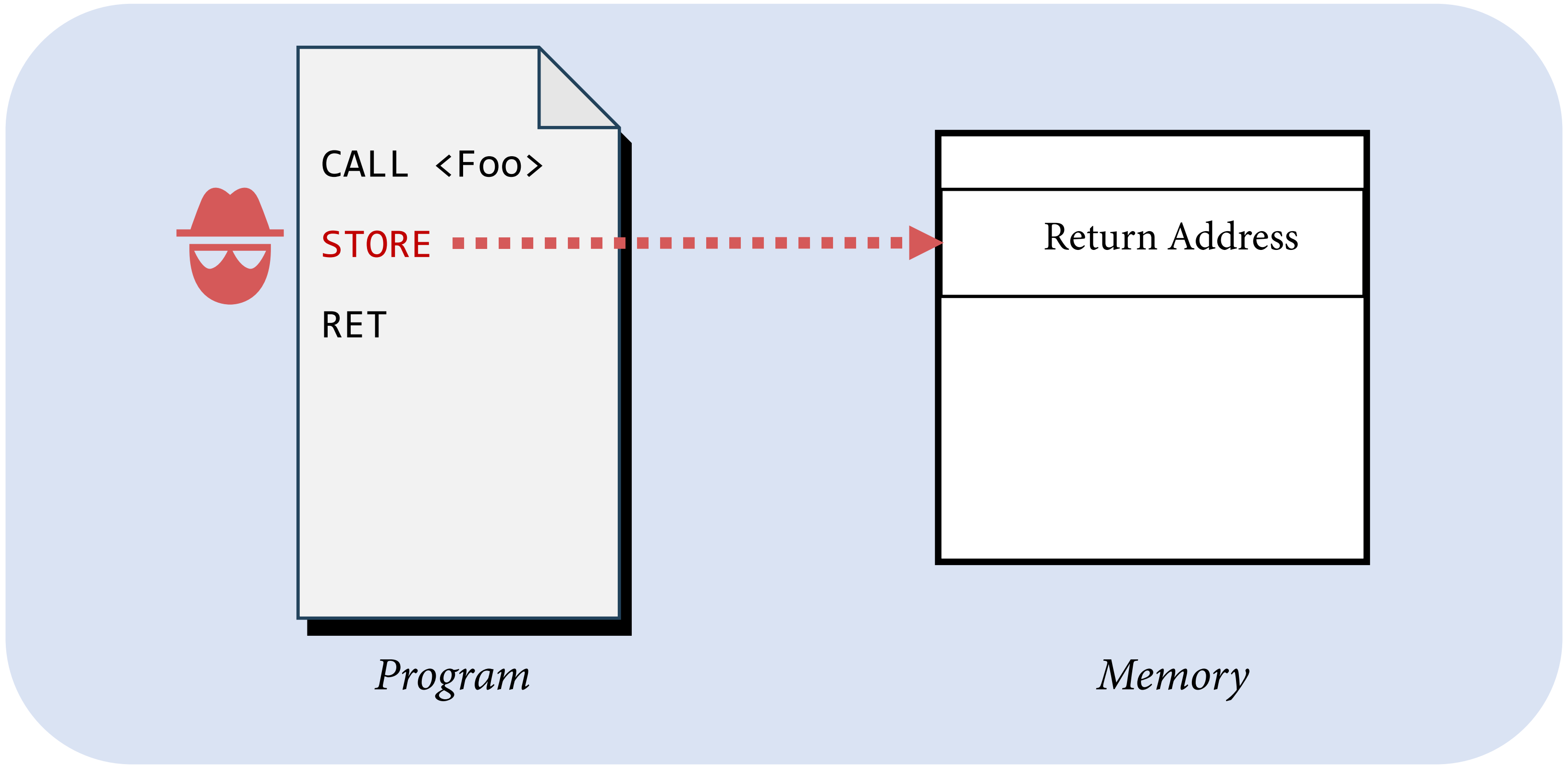
Return Address Protection



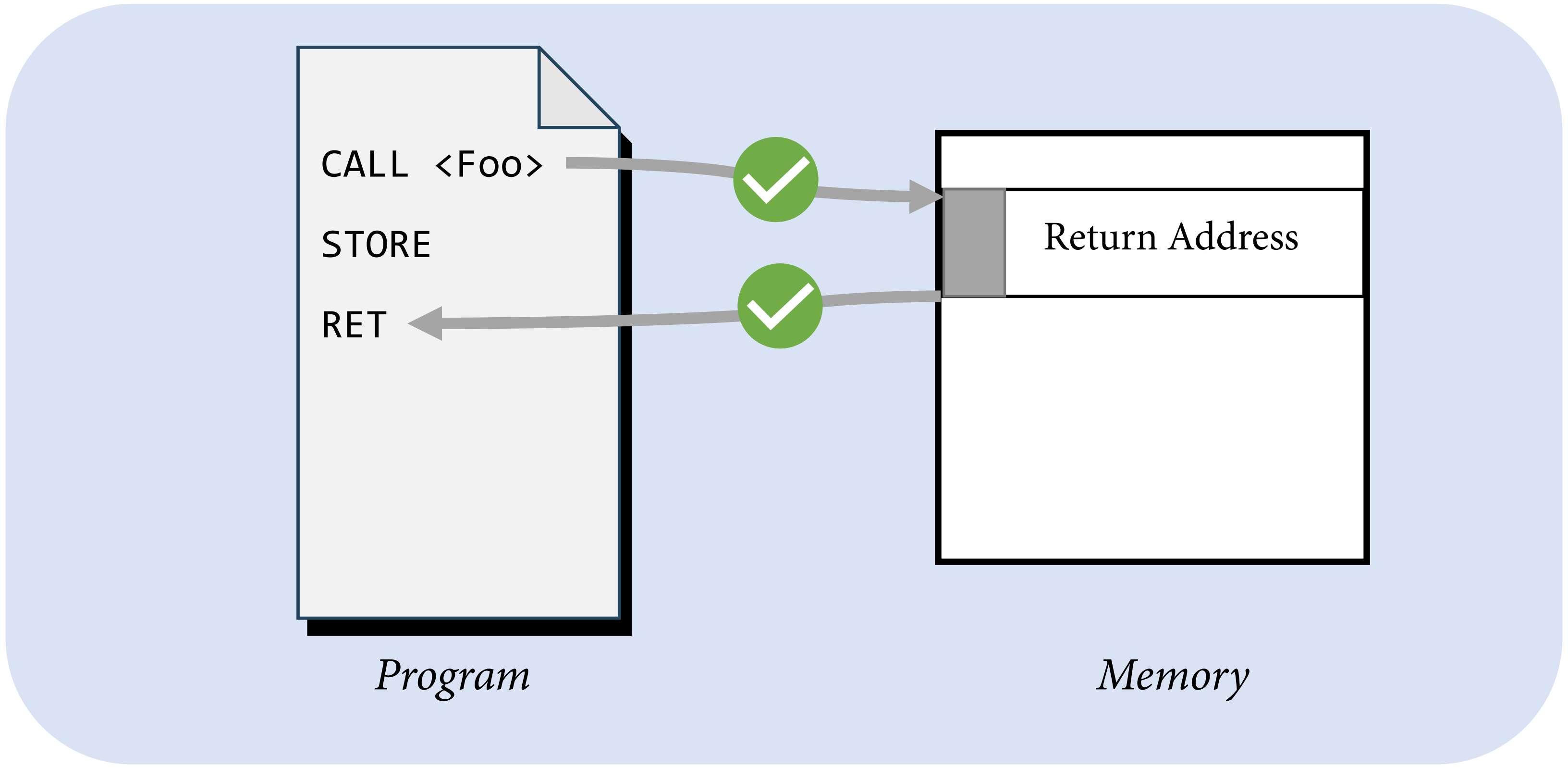
Return Address Protection



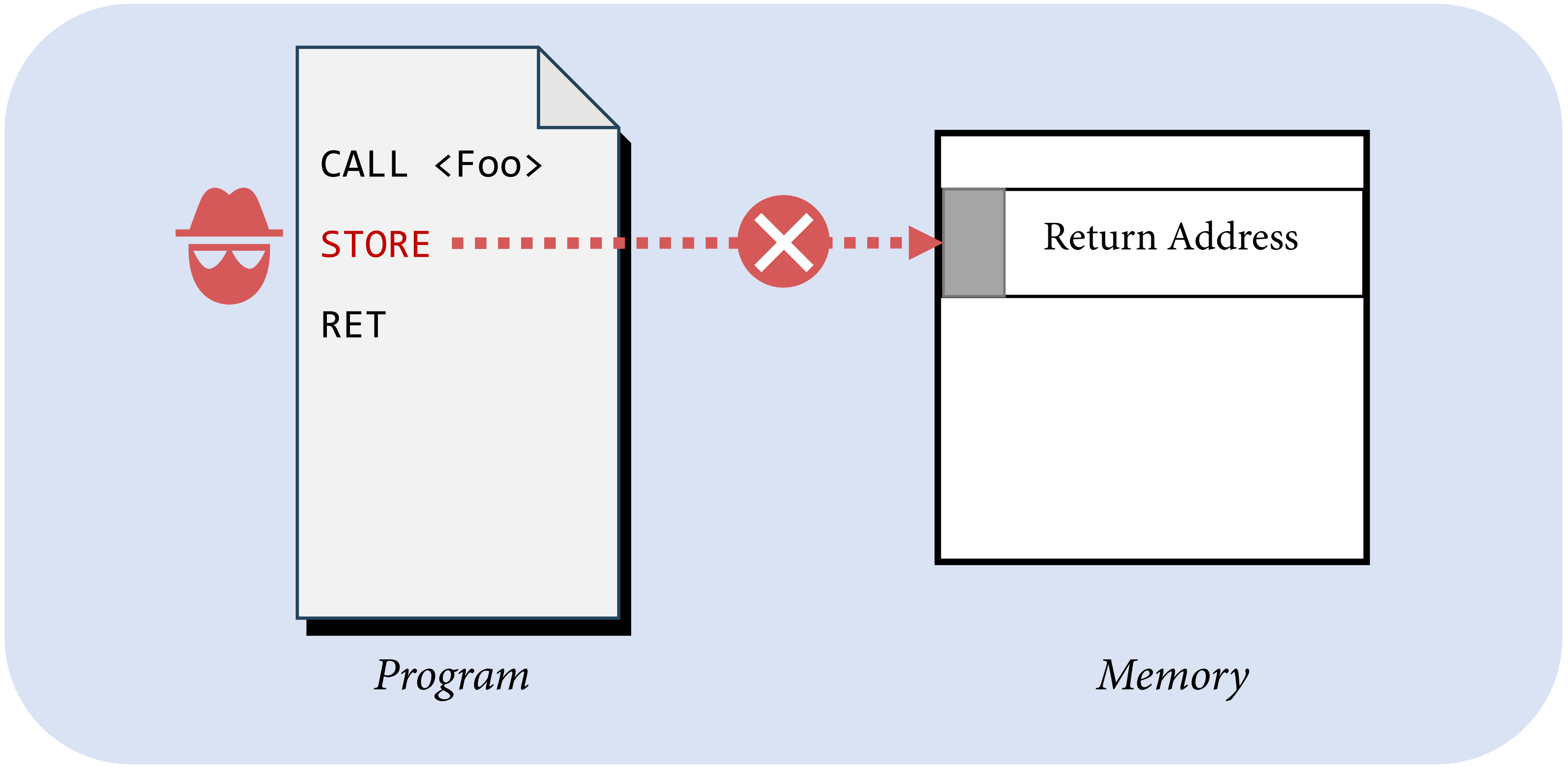
Return Address Protection



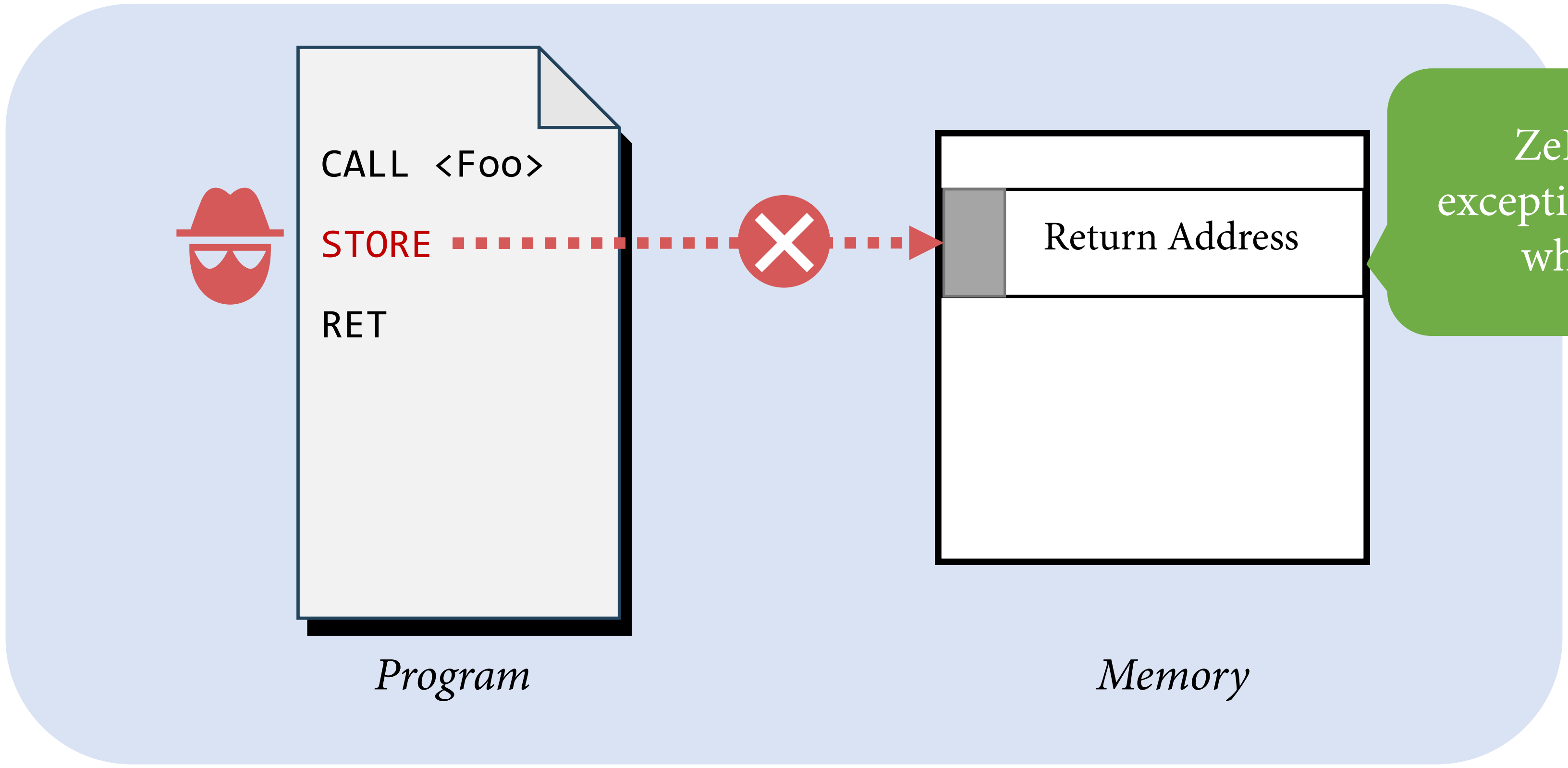
Return Address Protection



Return Address Protection

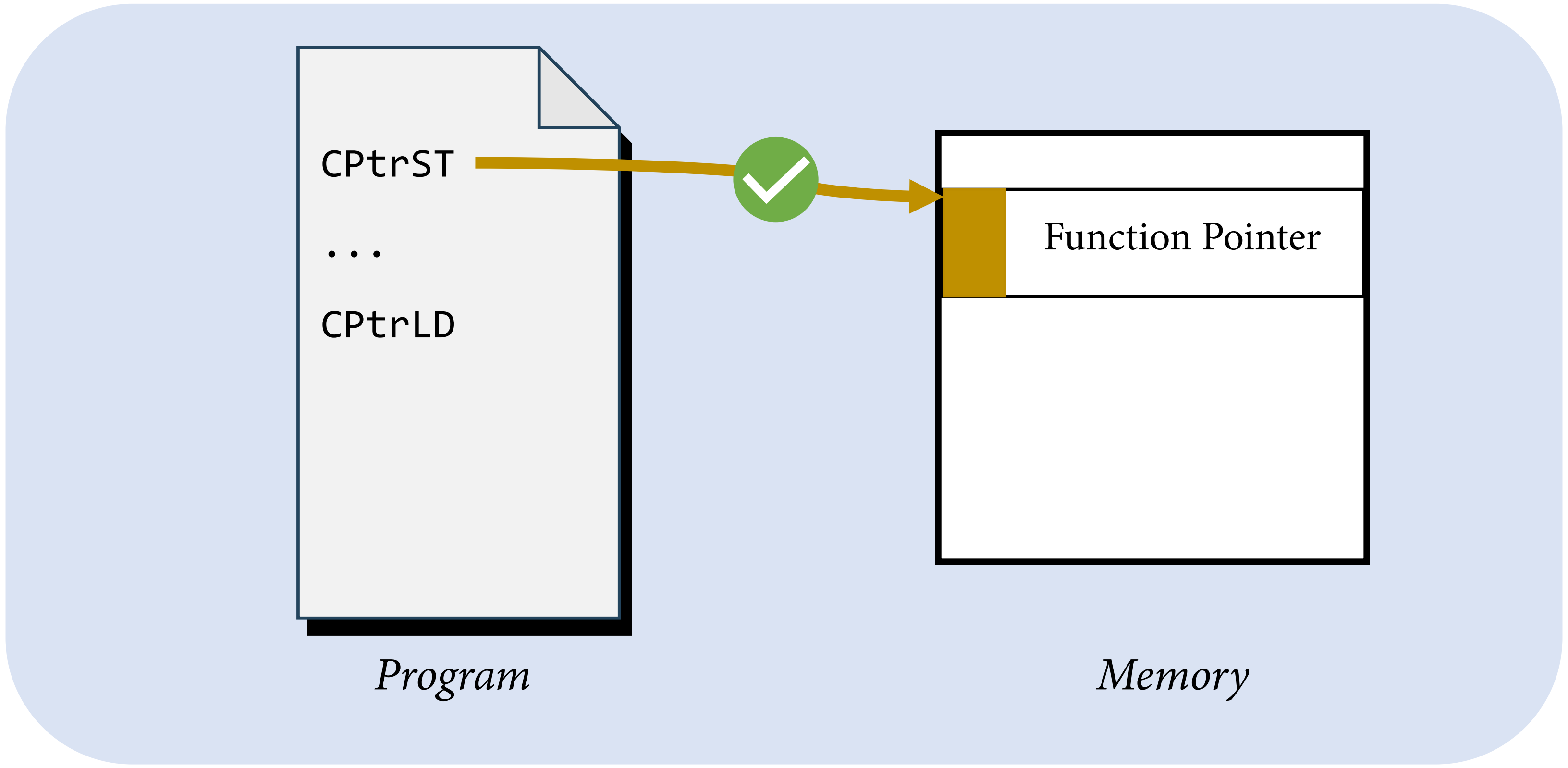


Return Address Protection

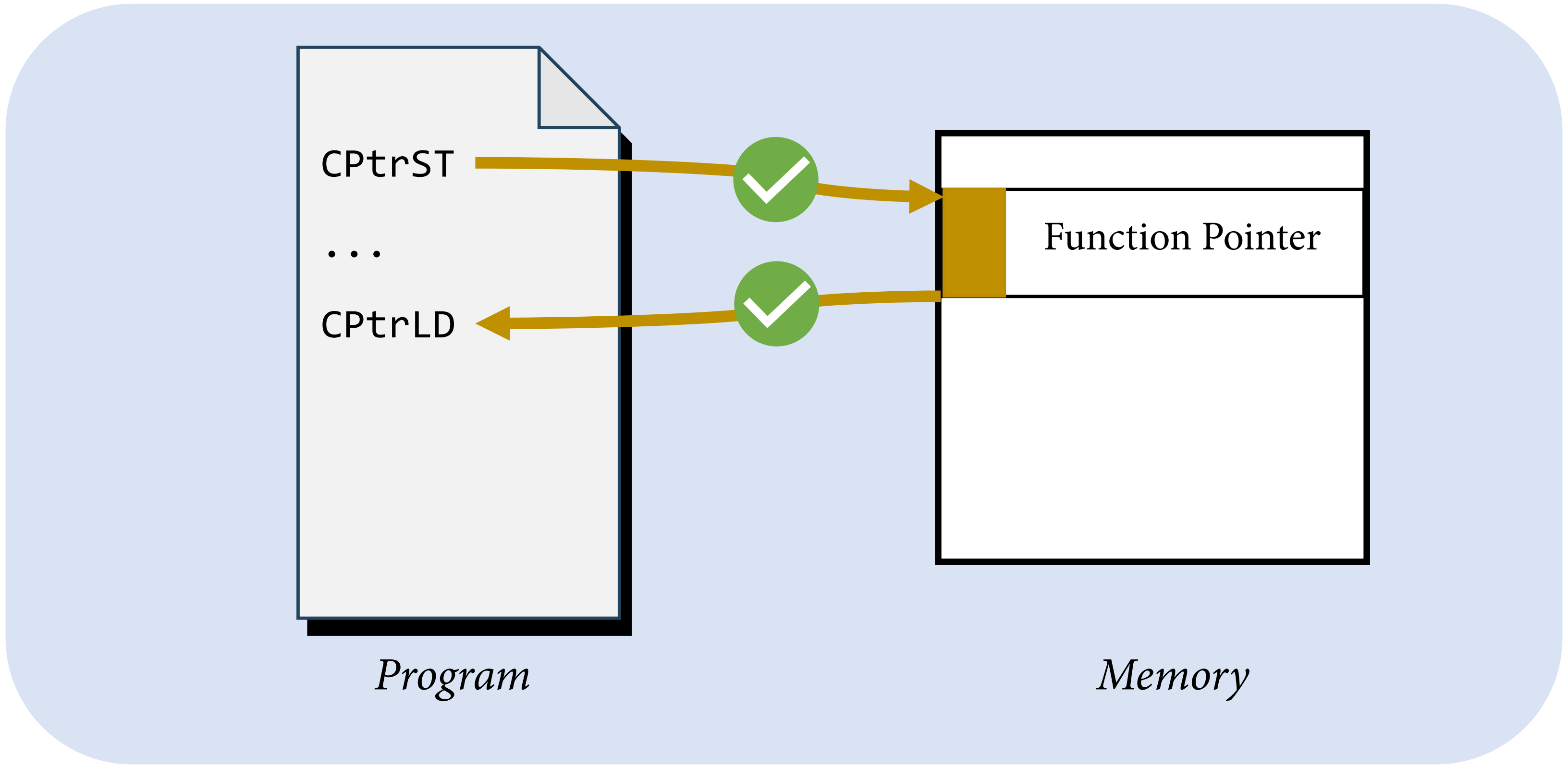


ZeRØ uses advisory exceptions to avoid crashing when under attack.

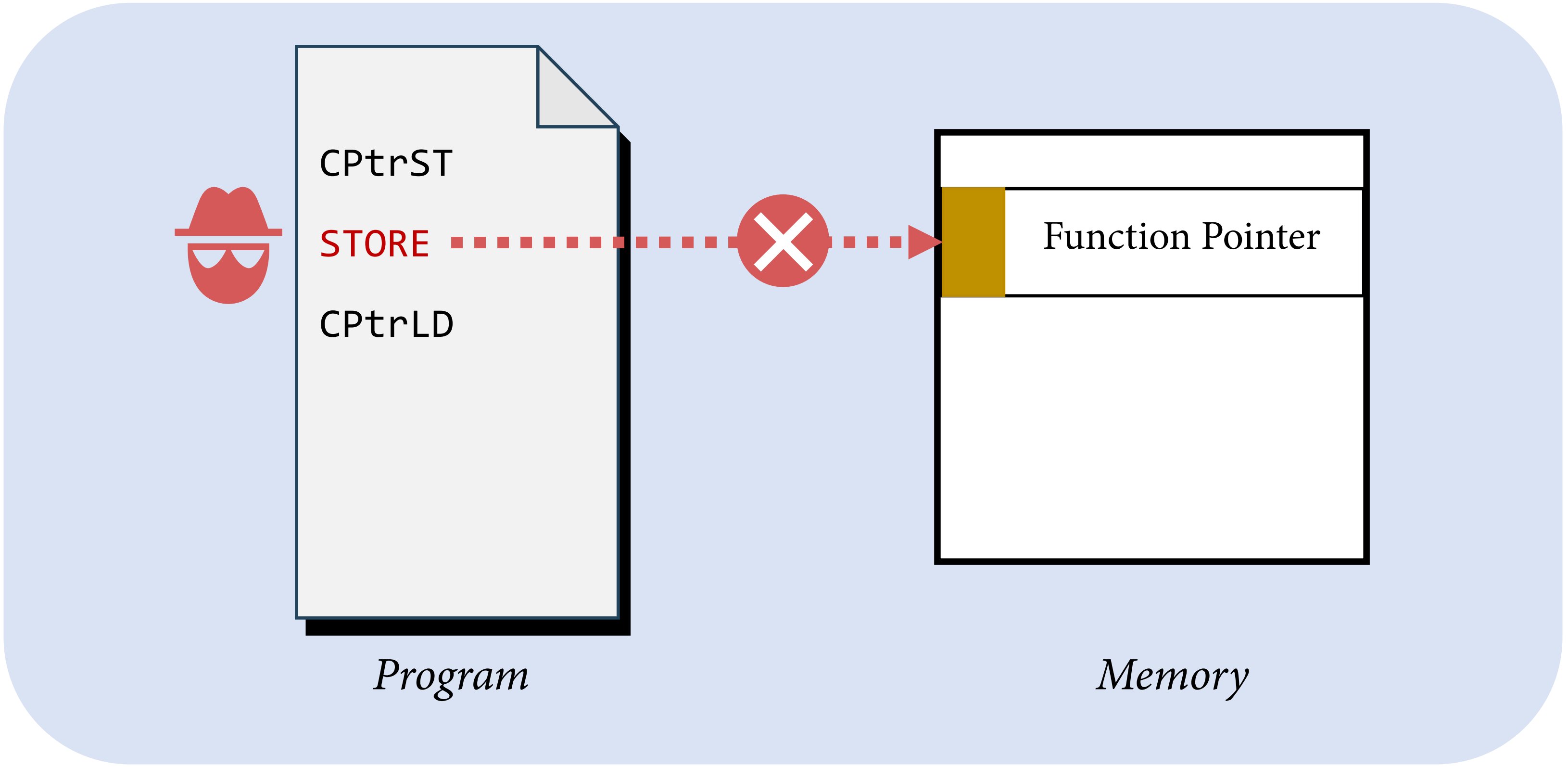
Code Pointer Integrity



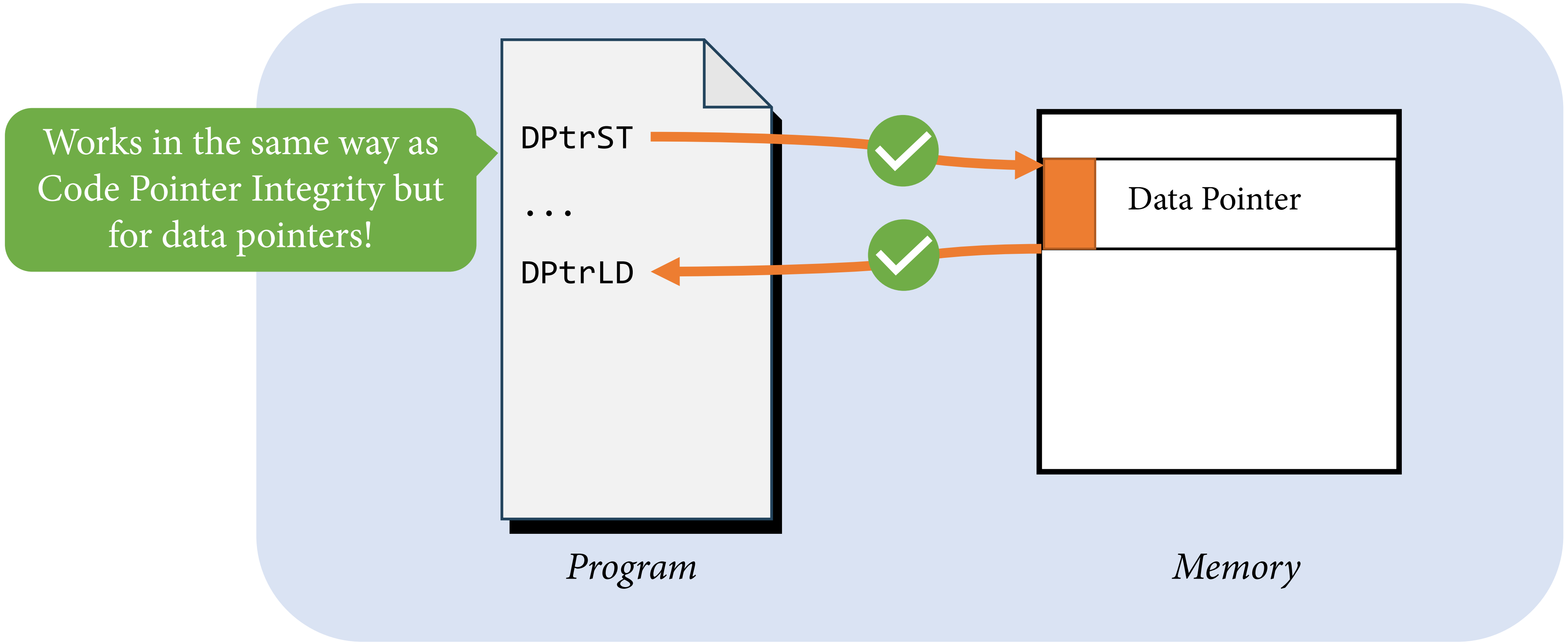
Code Pointer Integrity

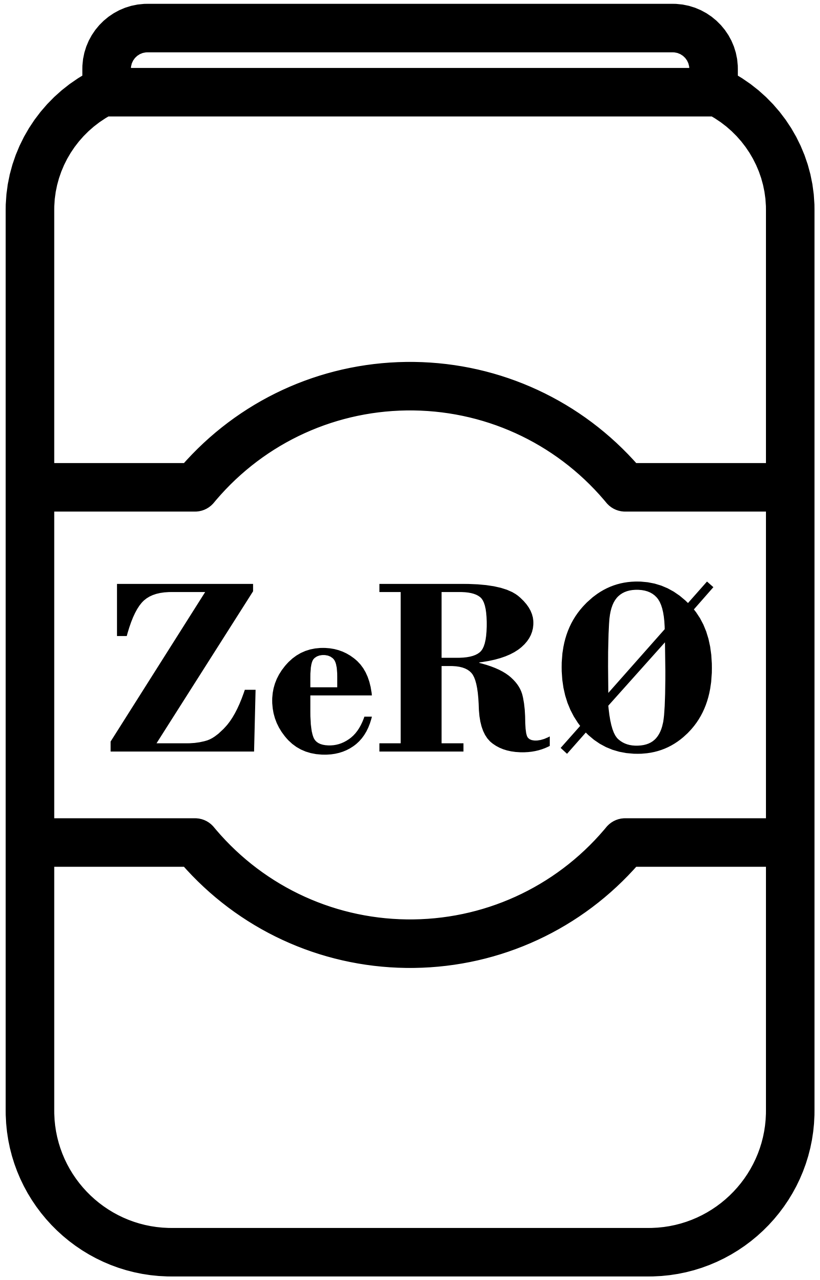


Code Pointer Integrity



Data Pointer Integrity


Works in the same way as Code Pointer Integrity but for data pointers!
DPtrST
...
DPtrLD
Data Pointer
Program
Memory



ISA Extensions

ZeRØ ISA Extensions

1 **Return Address Integrity** → None. Relies on Call Return semantics.

ZeRØ ISA Extensions

1. **Return Address Integrity** → None. Relies on Call Return semantics.
2. **Code Pointer Integrity** → `CPtrST/CPtrLD Address, Value`
3. **Data Pointer Integrity** → `DPtrST/DPtrLD Address, Value`

ZeRØ ISA Extensions

1. **Return Address Integrity** → None. Relies on Call Return semantics.
2. **Code Pointer Integrity** → `CPtrST/CPtrLD Address, Value`
3. **Data Pointer Integrity** → `DPtrST/DPtrLD Address, Value`

Same layout as regular load & stores!

ZeRØ ISA Extensions

1 **Return Address Integrity** → None. Relies on Call Return semantics.

2 **Code Pointer Integrity** → `CPtrST/CPtrLD Address, Value`

3 **Data Pointer Integrity** → `DPtrST/DPtrLD Address, Value`

`ClearMeta Address, Mask`

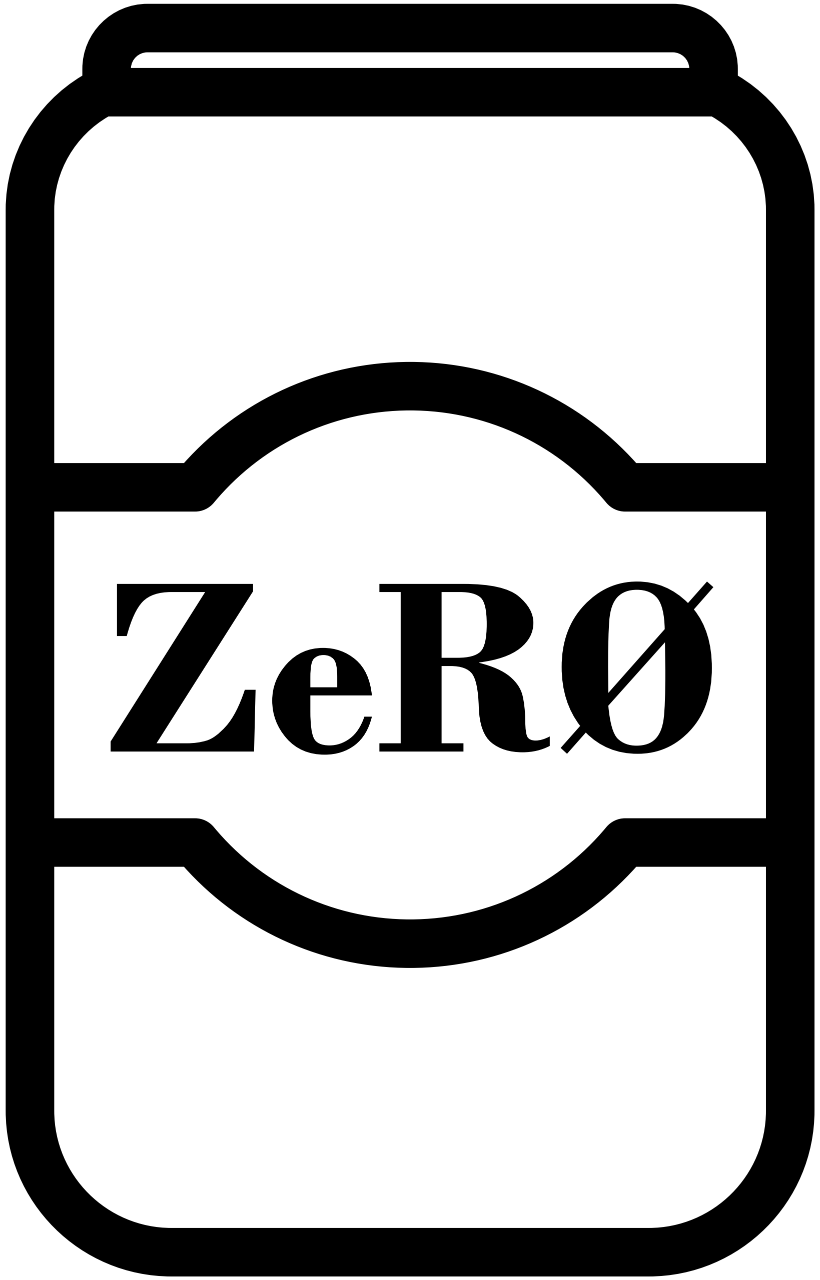
ZeRØ ISA Extensions

1. **Return Address Integrity** → None. Relies on Call Return semantics.
2. **Code Pointer Integrity** → `CPtrST/CPtrLD Address, Value`
3. **Data Pointer Integrity** → `DPtrST/DPtrLD Address, Value`

`ClearMeta Address, Mask`

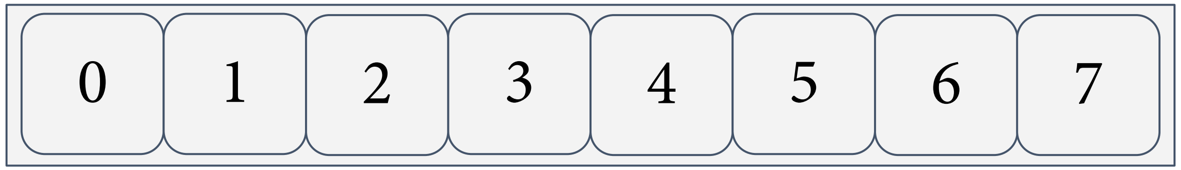
Invoked on `free` or `delete`.

How can we keep track of ZeRØ bits?



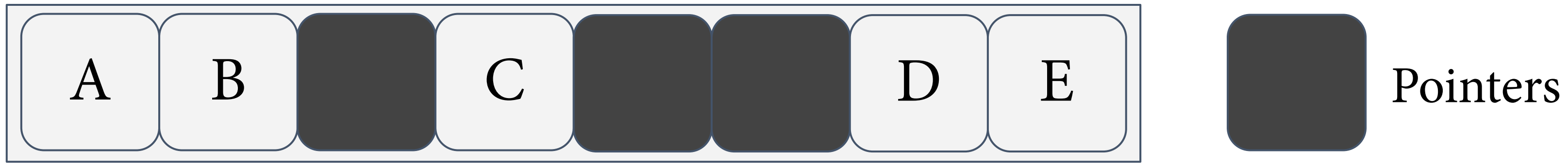
Cache Line Formats

Cache Line Formats



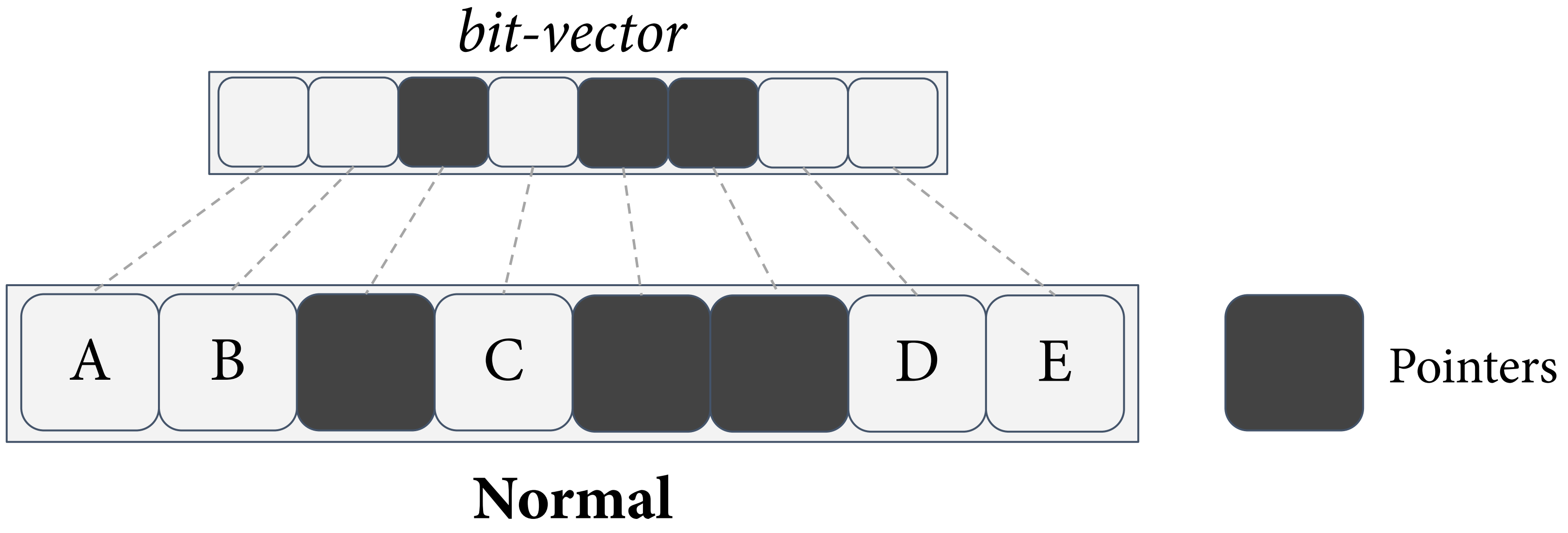
Normal

Cache Line Formats



Normal

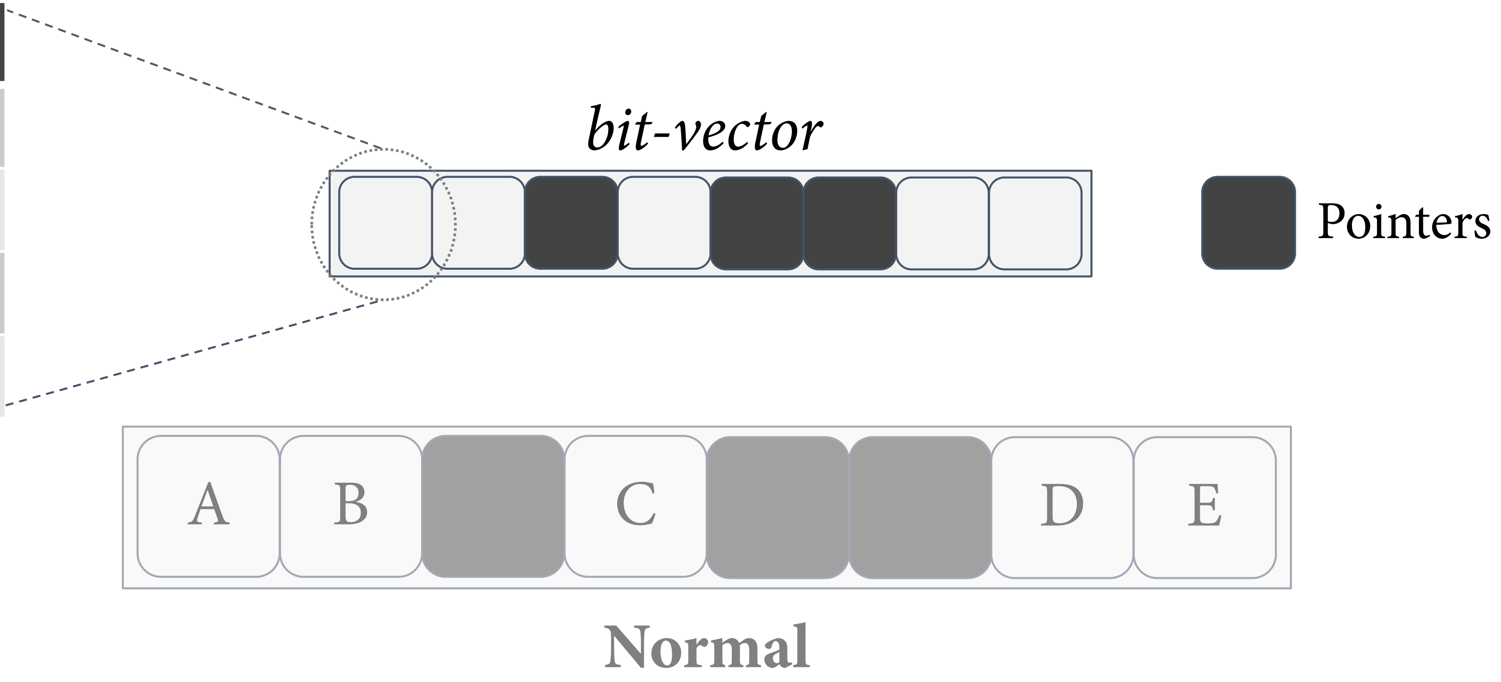
Cache Line Formats



Cache Line Formats

Format Encoding Table

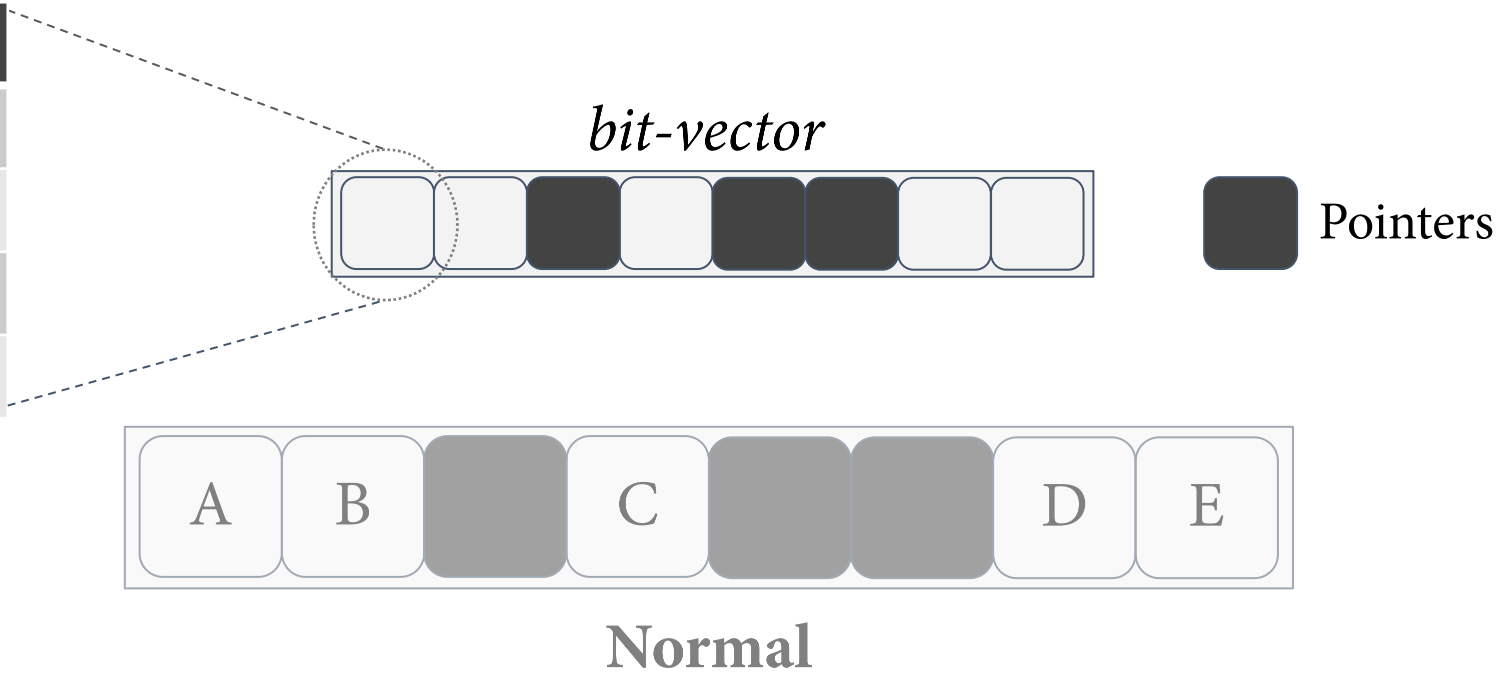
Type	Bits
Return address	01



Cache Line Formats

Format Encoding Table

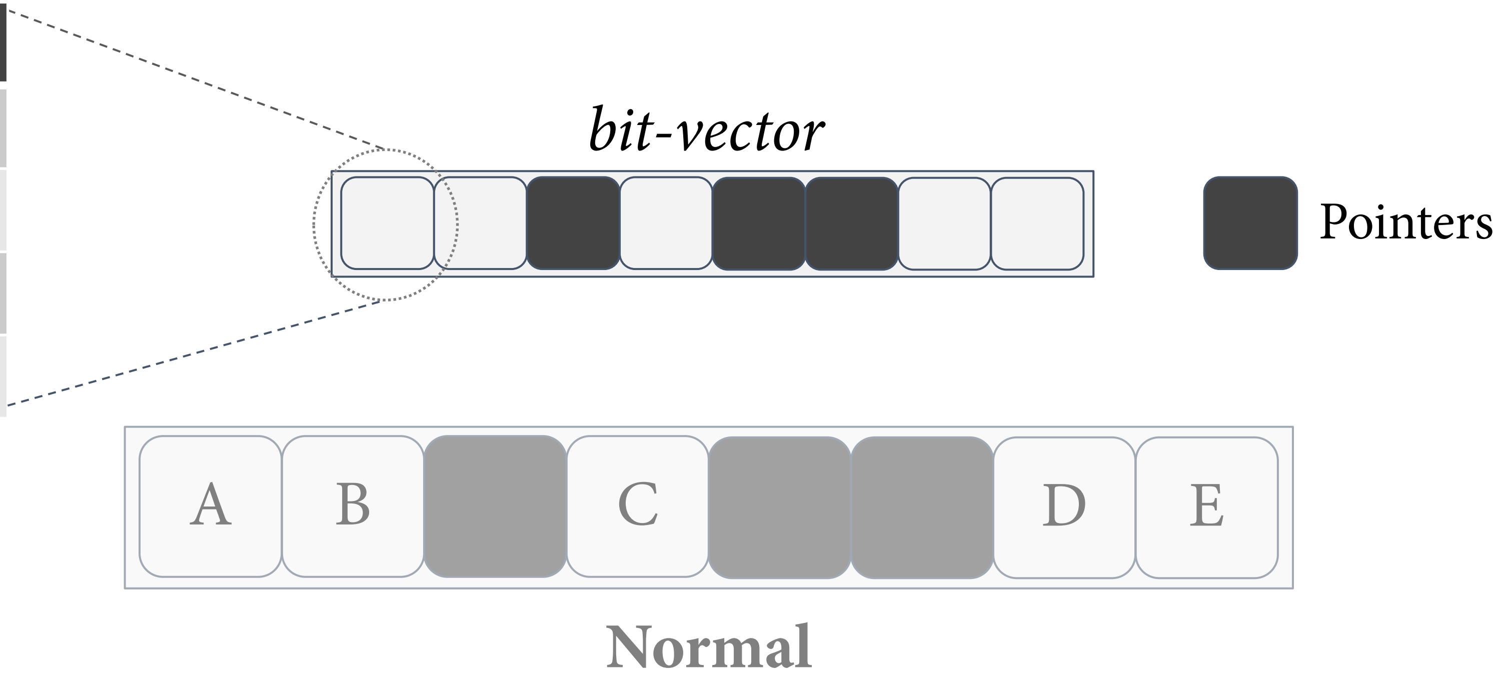
Type	Bits
Return address	01
Function pointer	10



Cache Line Formats

Format Encoding Table

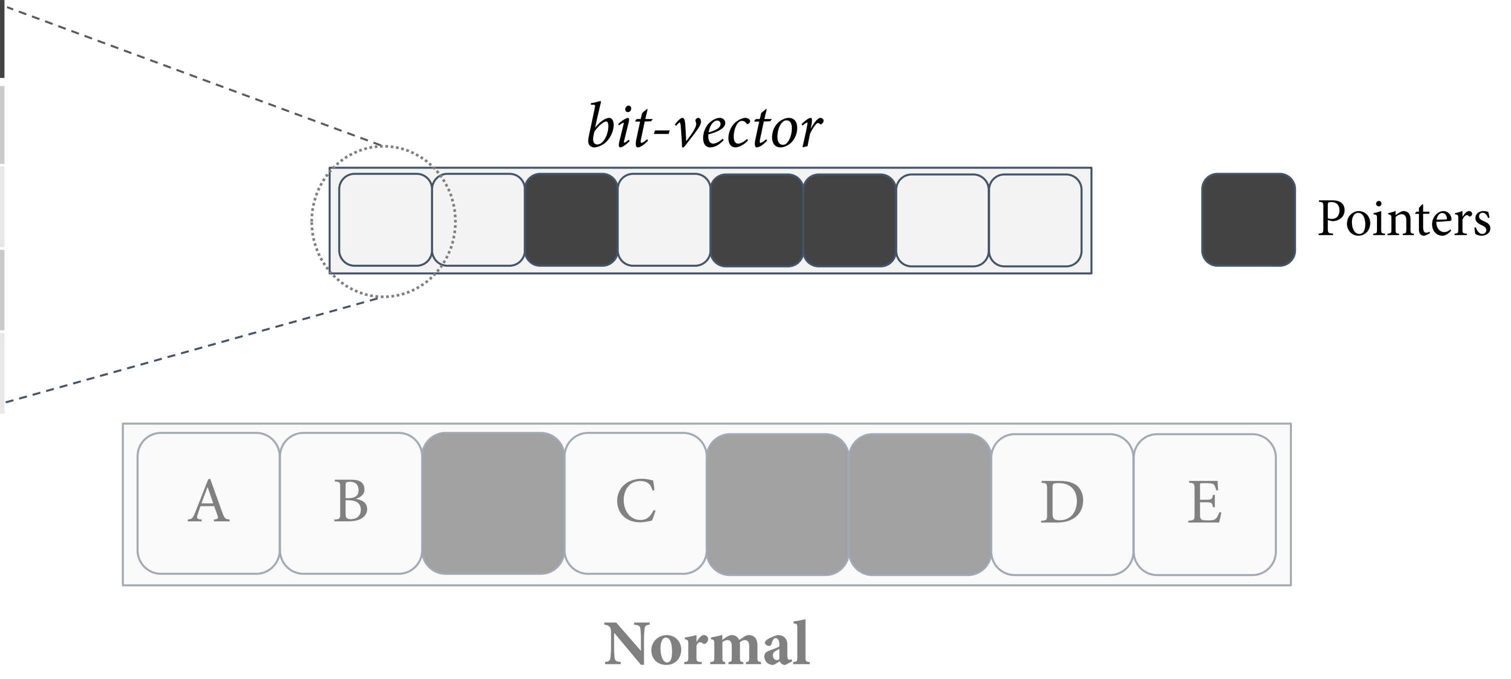
Type	Bits
Return address	01
Function pointer	10
Data pointer	11



Cache Line Formats

Format Encoding Table

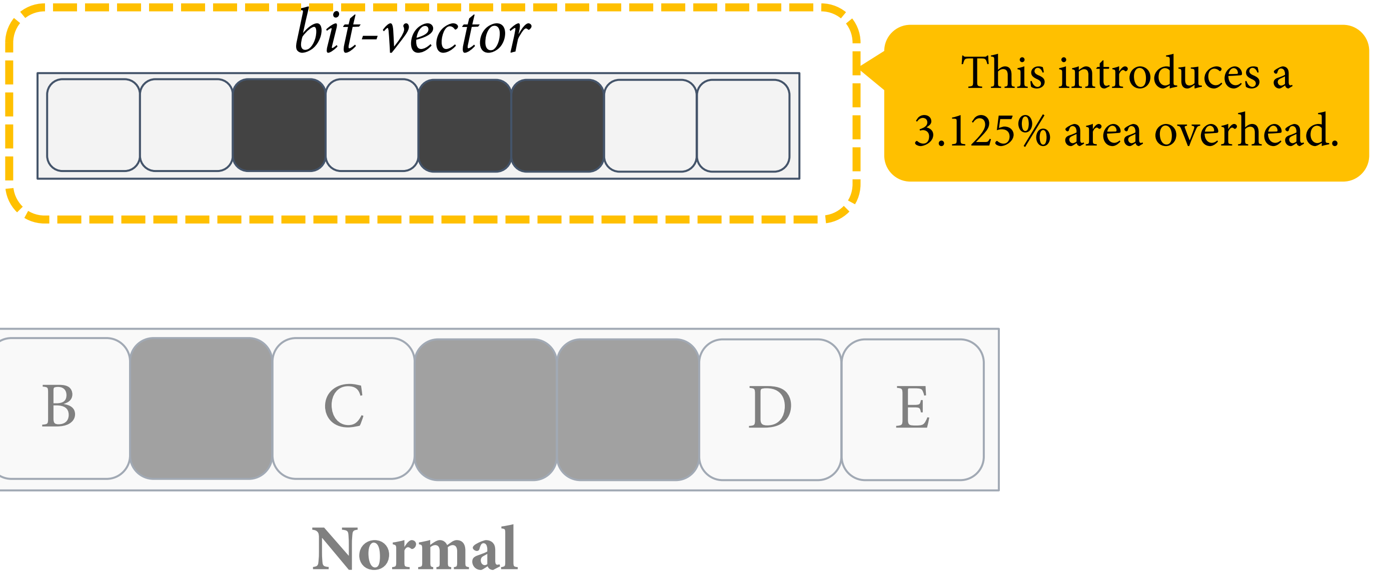
Type	Bits
Regular data	00
Return address	01
Function pointer	10
Data pointer	11



Cache Line Formats

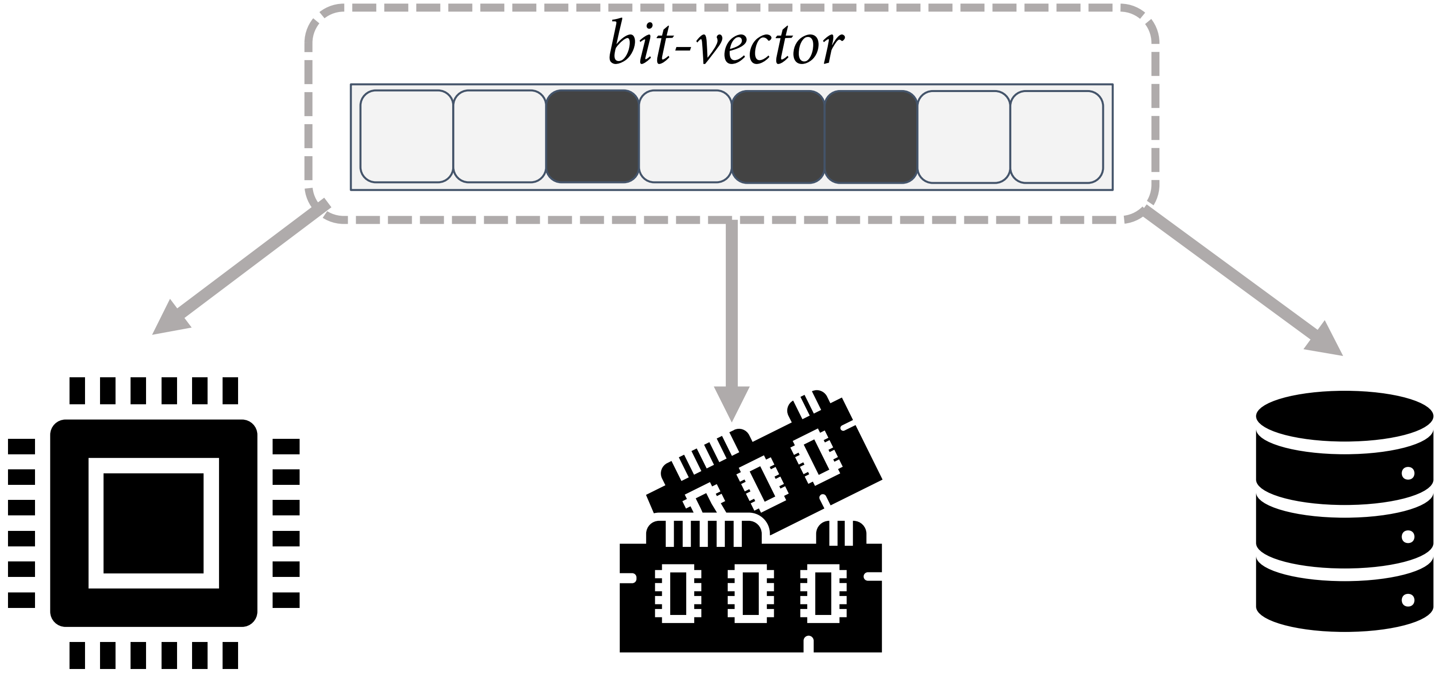
Format Encoding Table

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



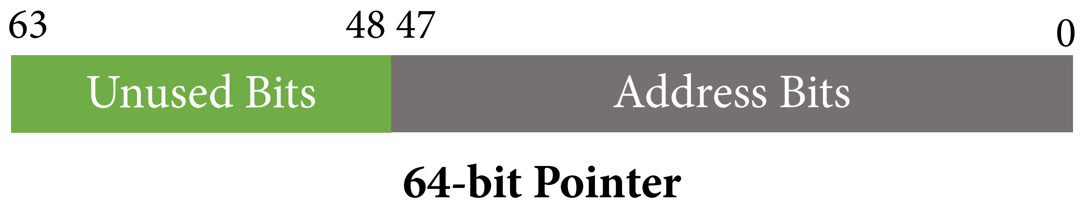
Cache Line Formats

Using a bit-vector throughout the memory hierarchy is **inefficient!**



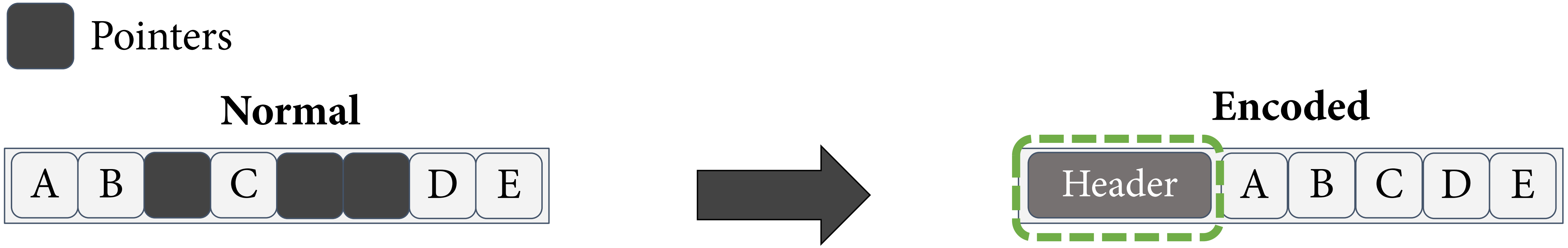
Cache Line Formats

In ZeRØ, we encode metadata **within** unused pointer bits.



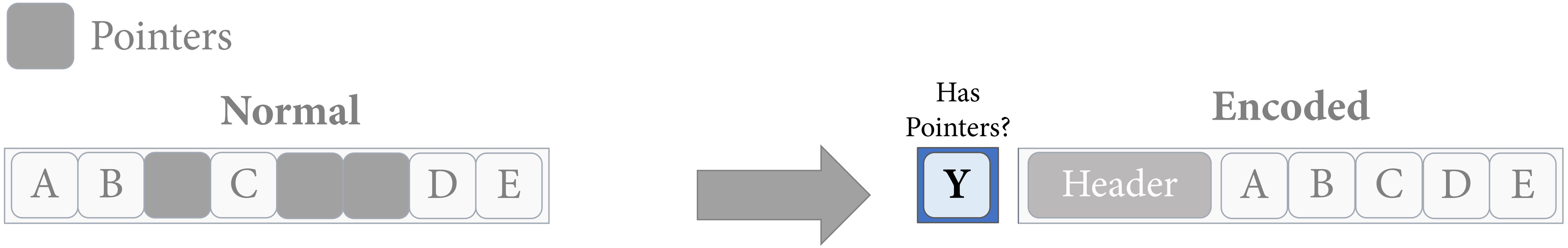
Cache Line Formats

In ZeRØ, we encode metadata **within** unused pointer bits.



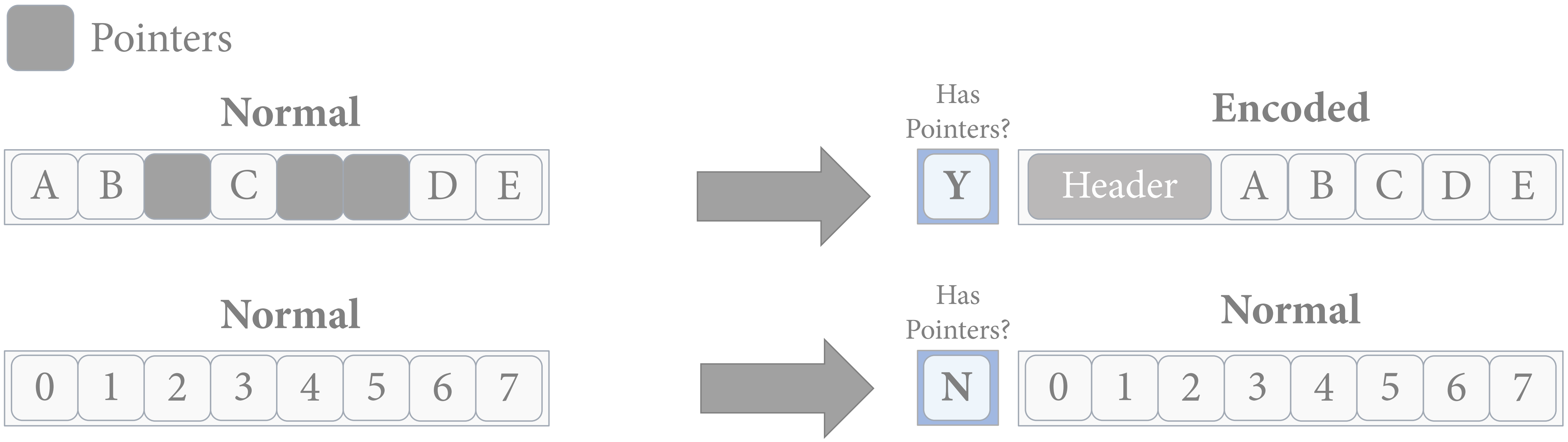
Cache Line Formats

In ZeRØ, we encode metadata **within** unused pointer bits.



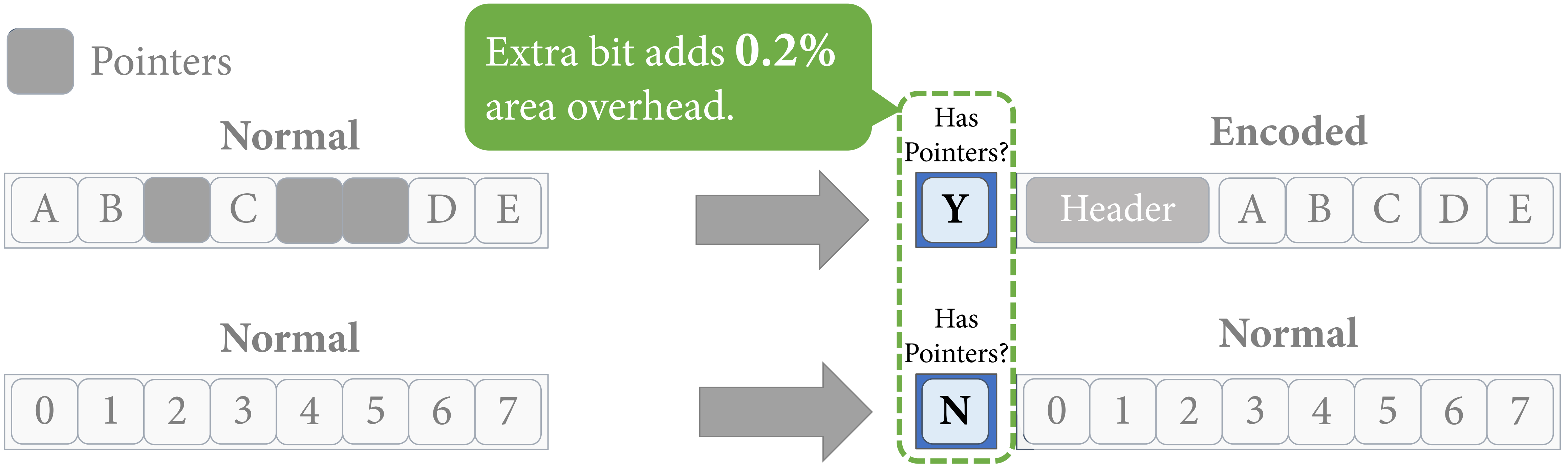
Cache Line Formats

In ZeRØ, we encode metadata **within** unused pointer bits.



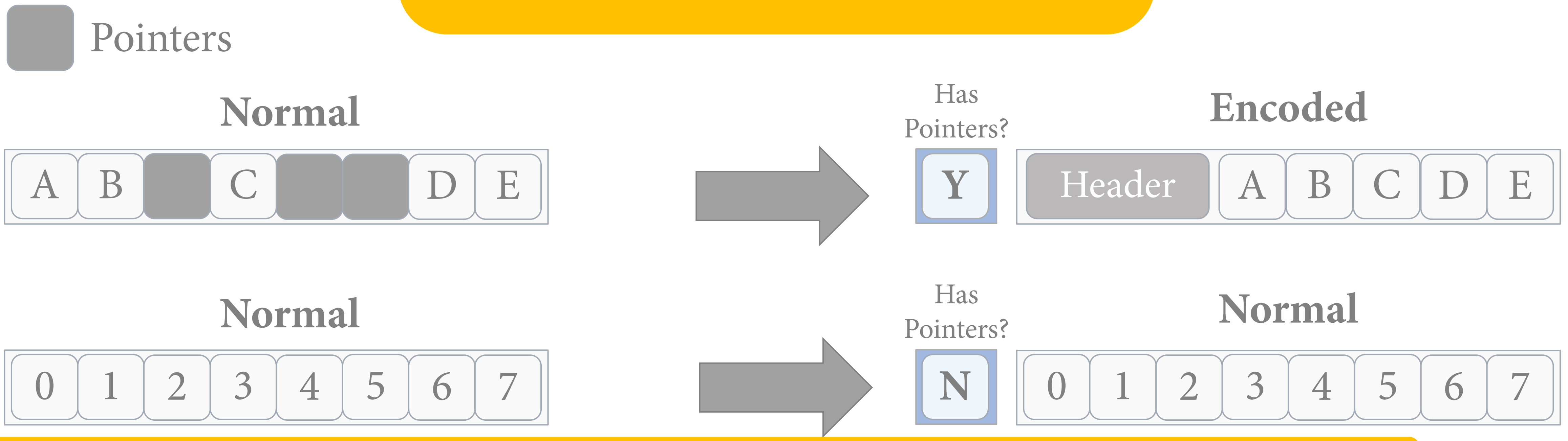
Cache Line Formats

In ZeRØ, we encode metadata **within** unused pointer bits.

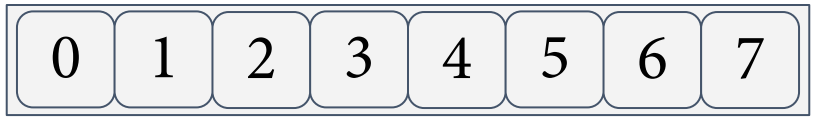


Cache Line Formats

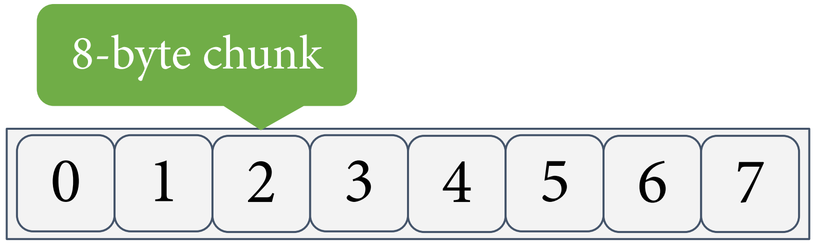
A novel variant of Californs



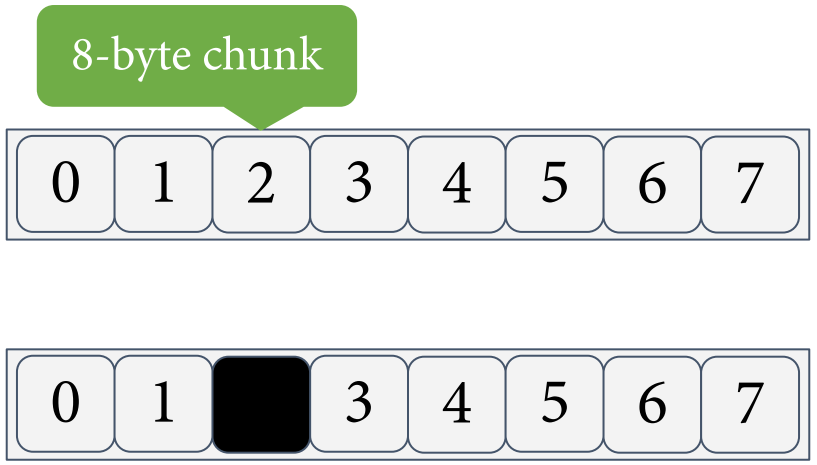
Cache Line Formats



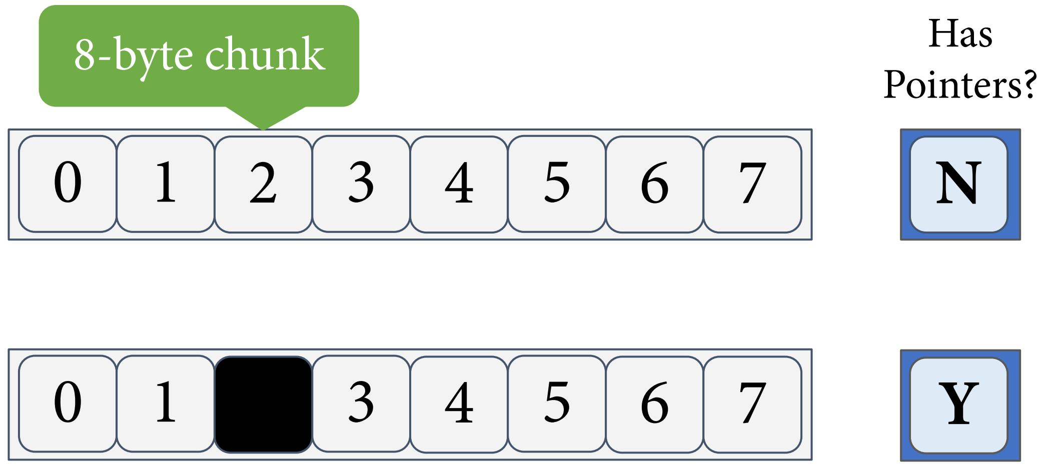
Cache Line Formats



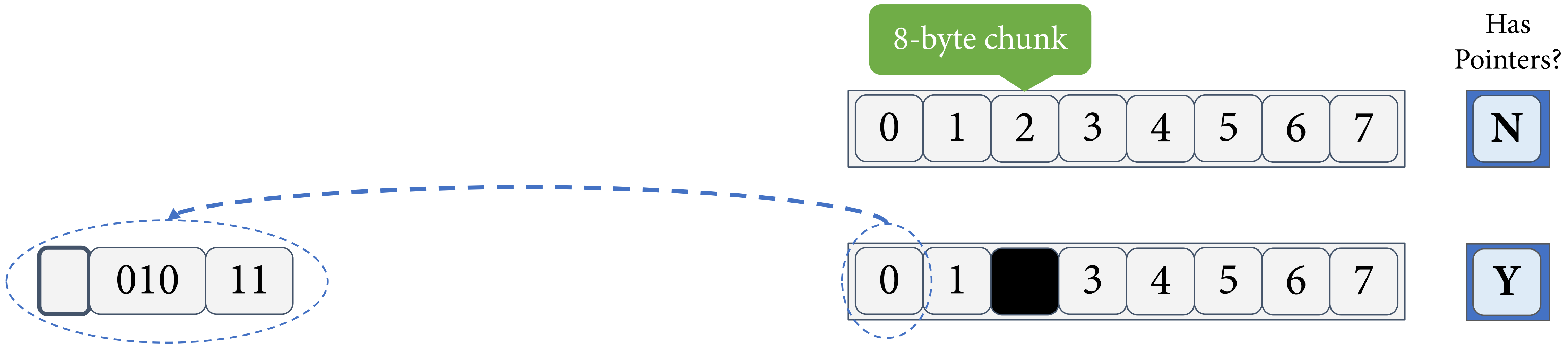
Cache Line Formats



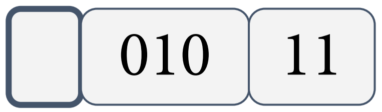
Cache Line Formats

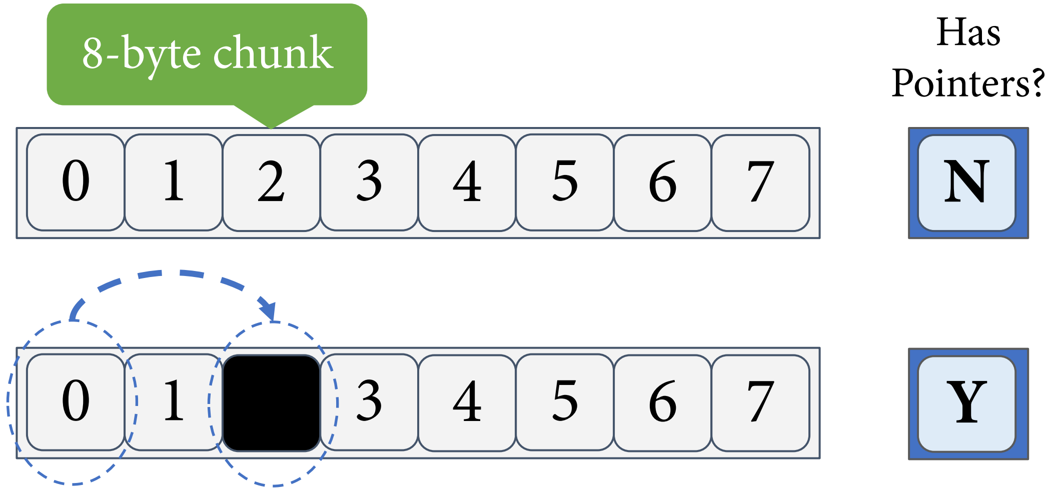


Cache Line Formats

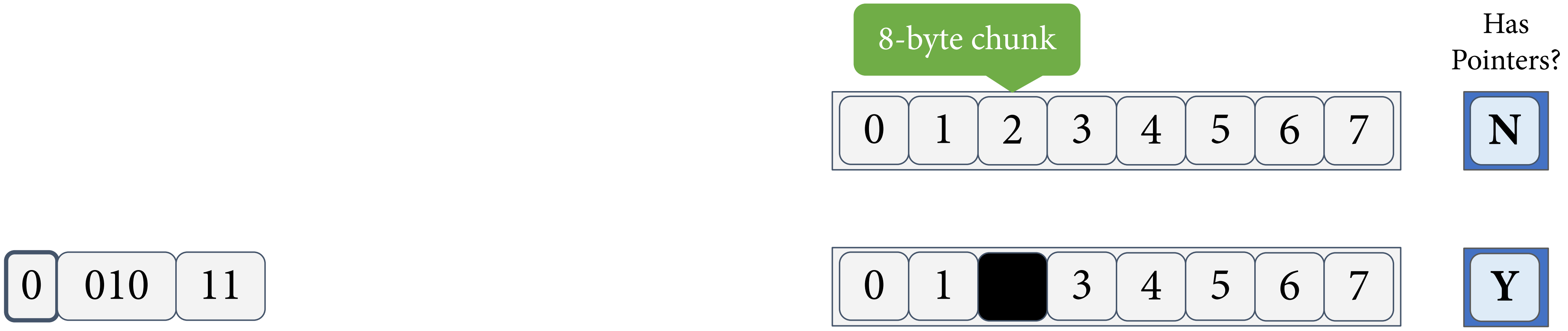


Cache Line Formats

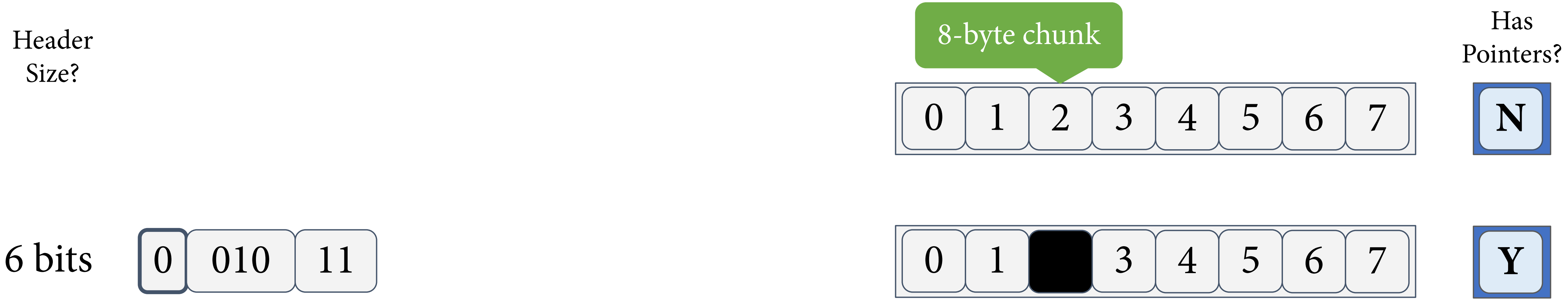




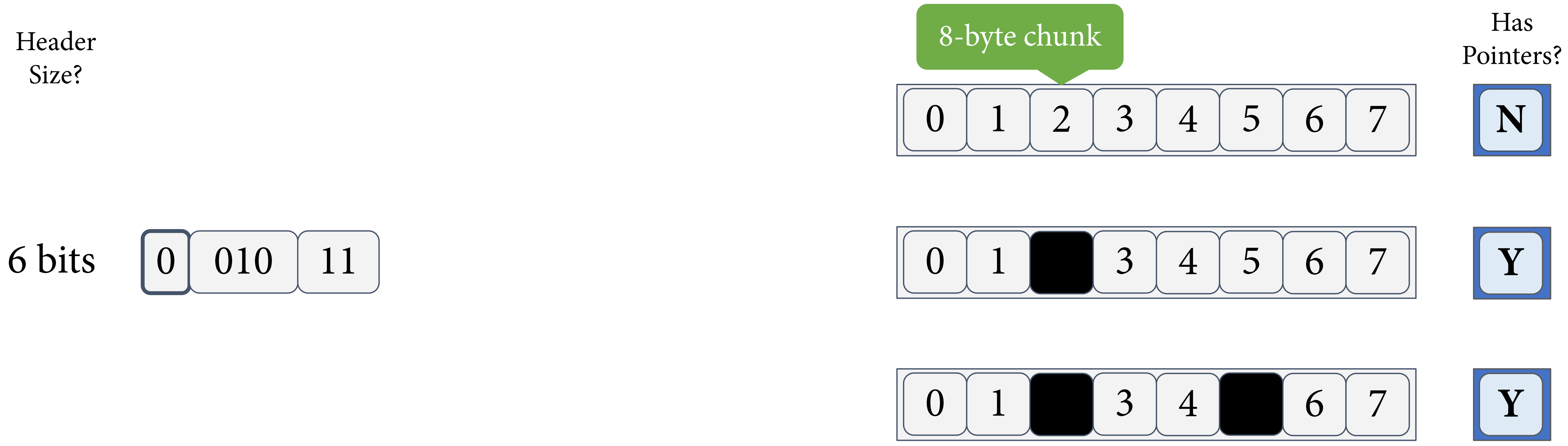
Cache Line Formats



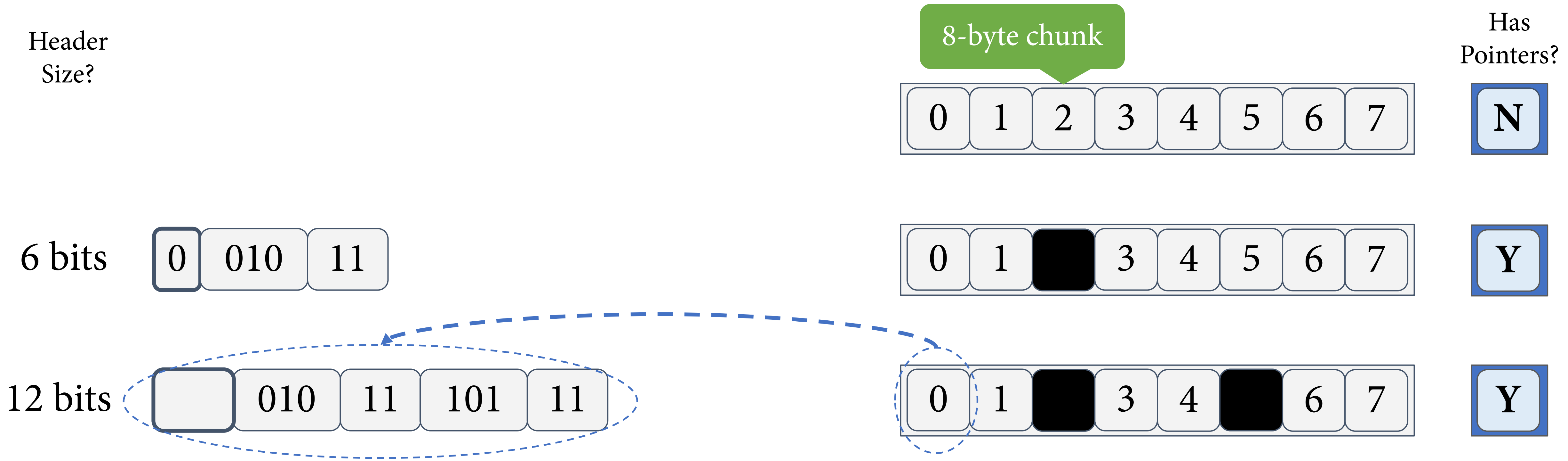
Cache Line Formats



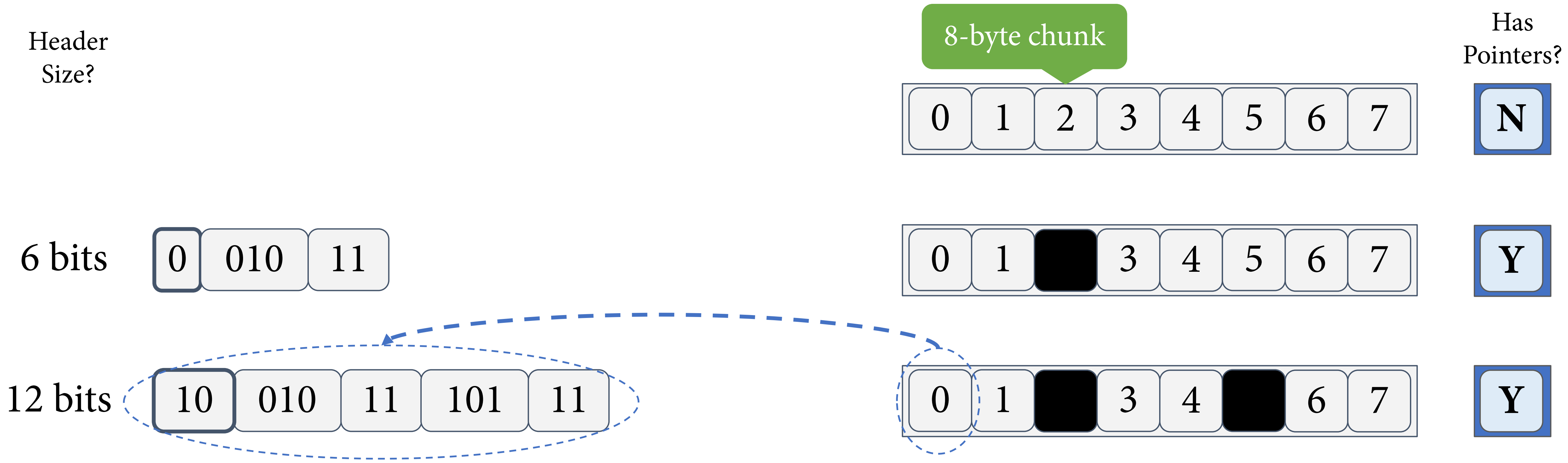
Cache Line Formats



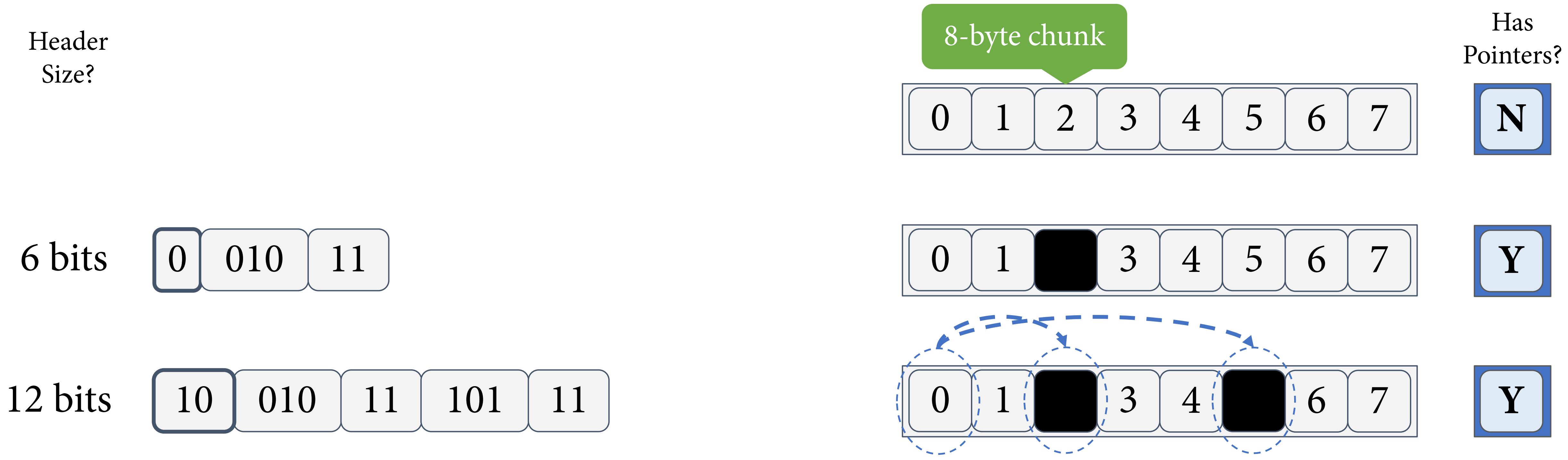
Cache Line Formats



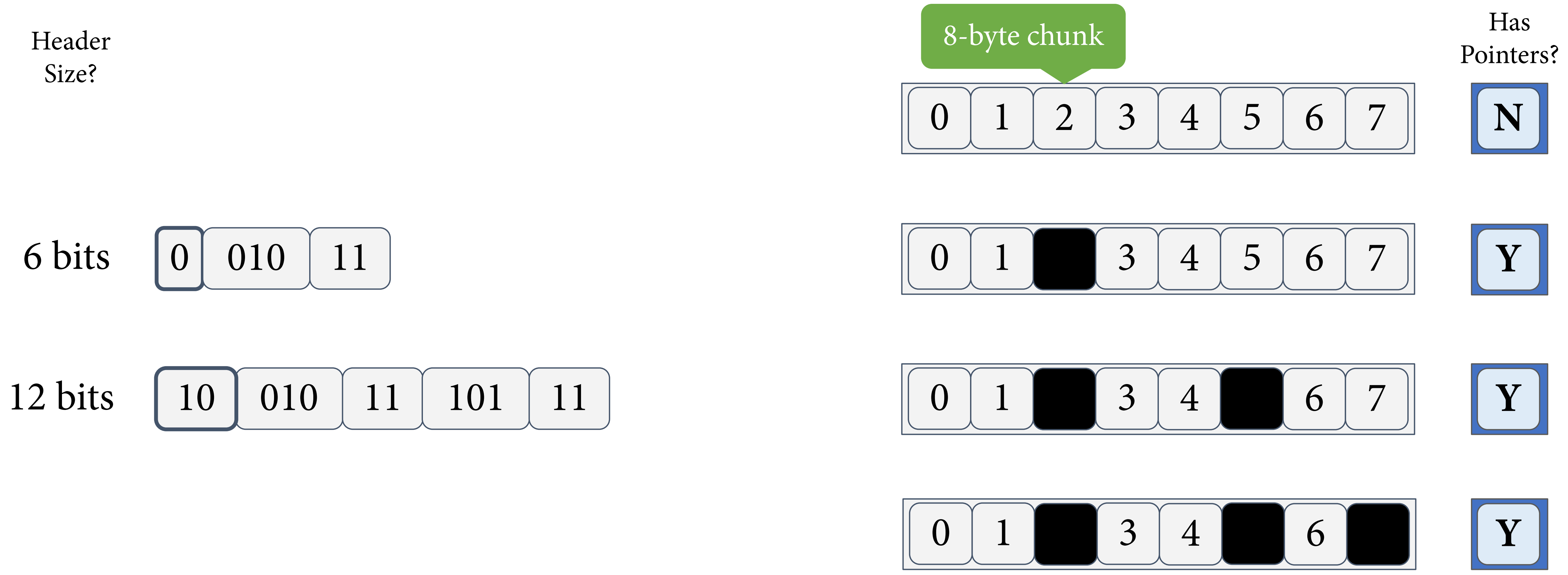
Cache Line Formats



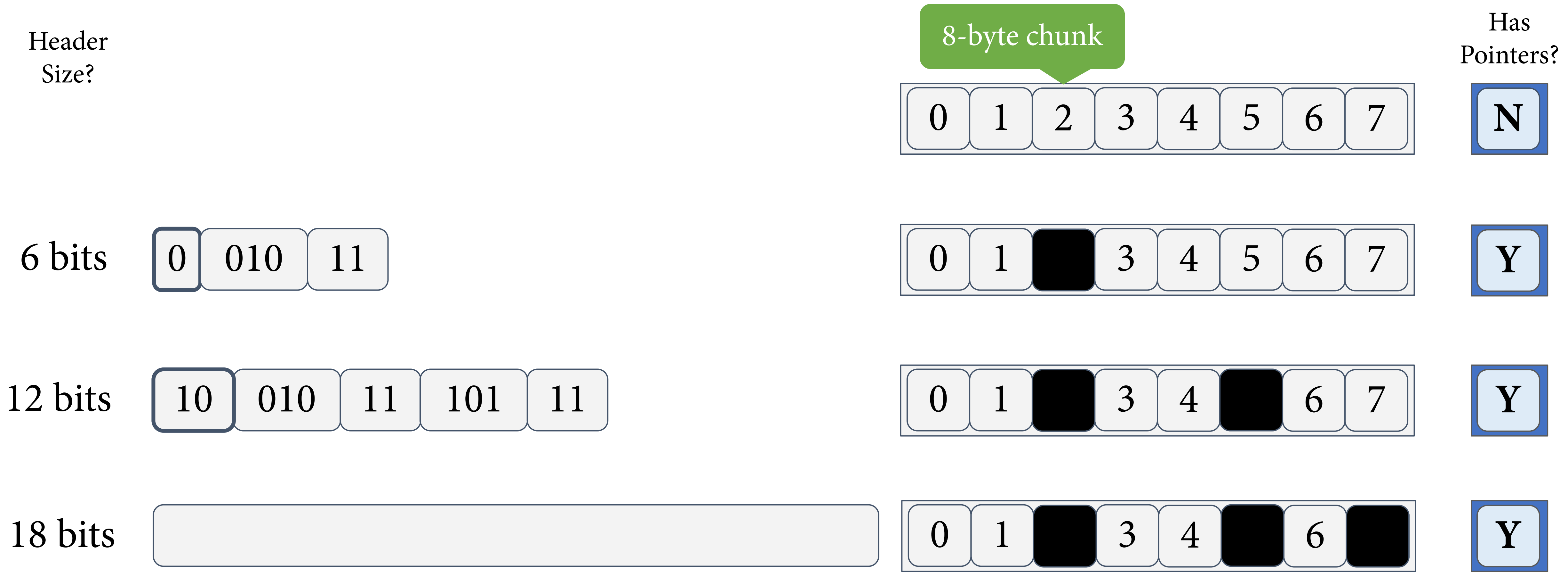
Cache Line Formats



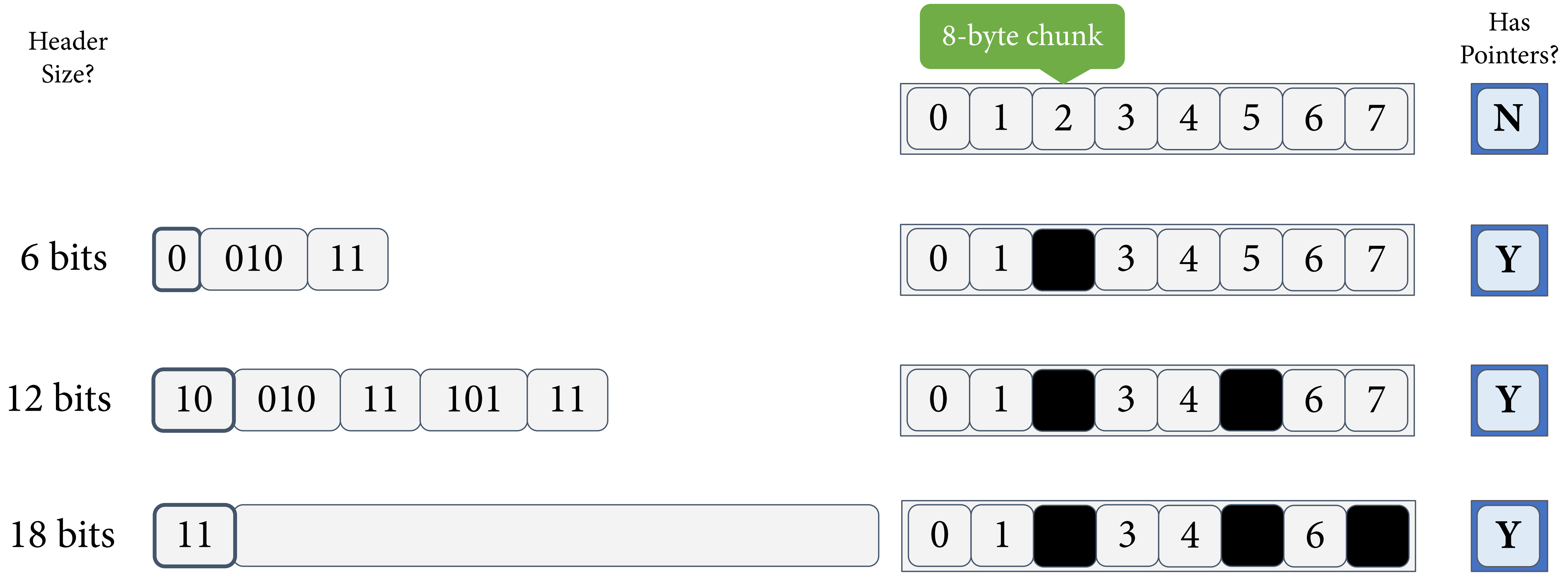
Cache Line Formats



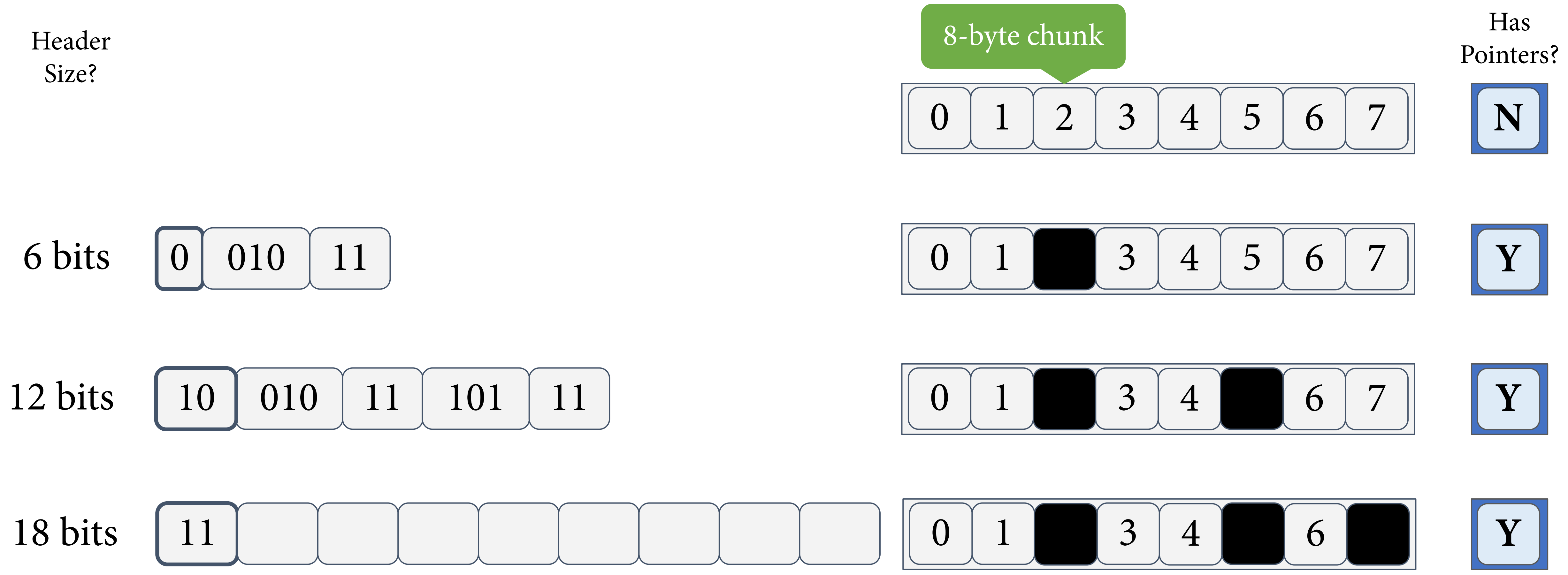
Cache Line Formats



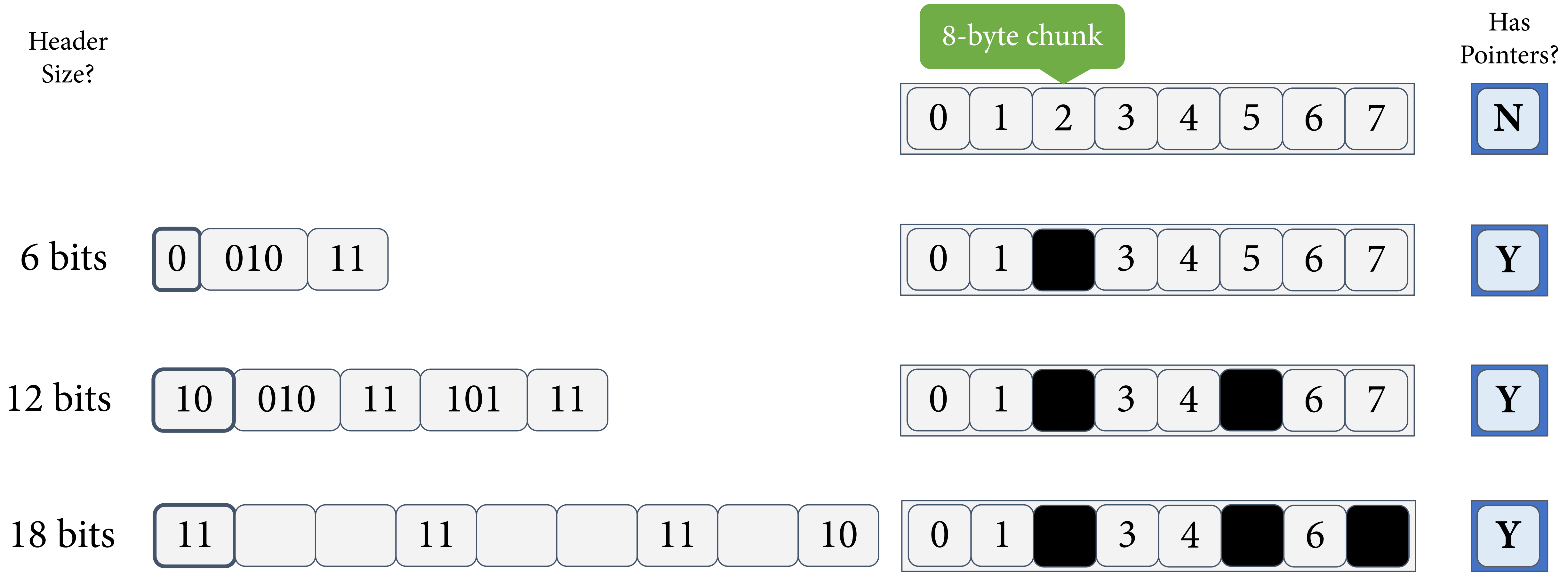
Cache Line Formats



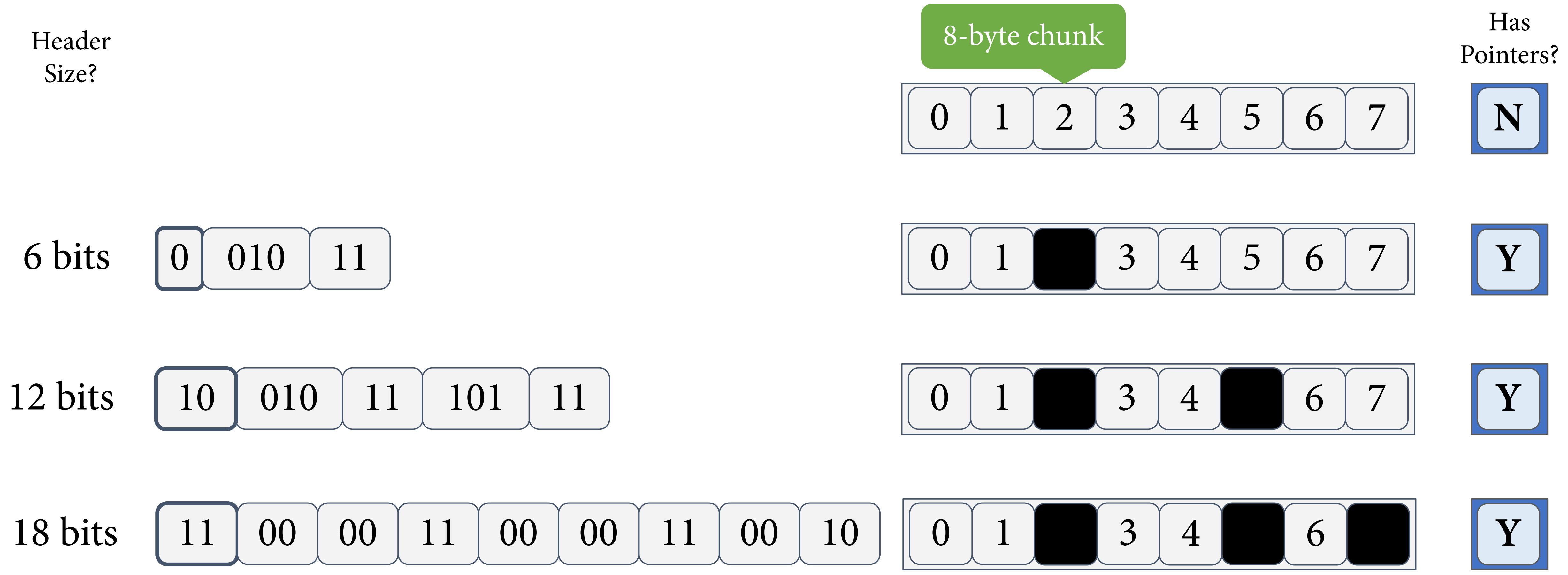
Cache Line Formats



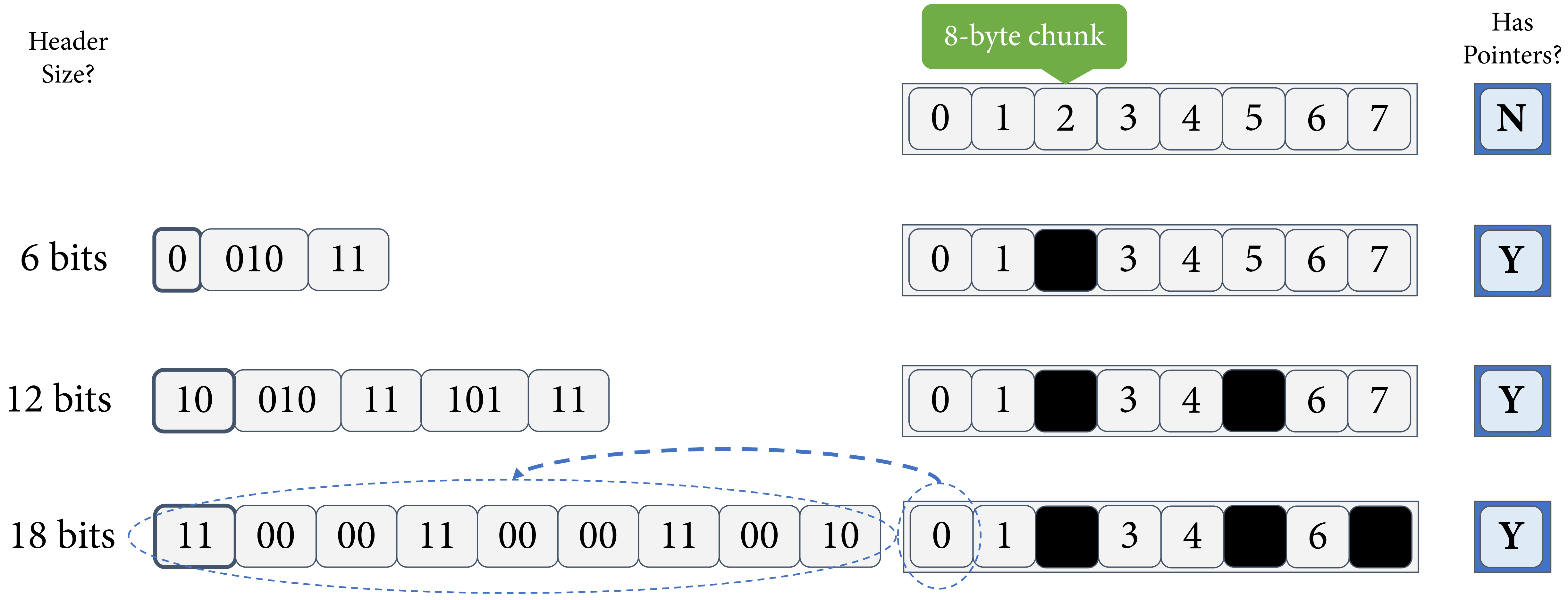
Cache Line Formats



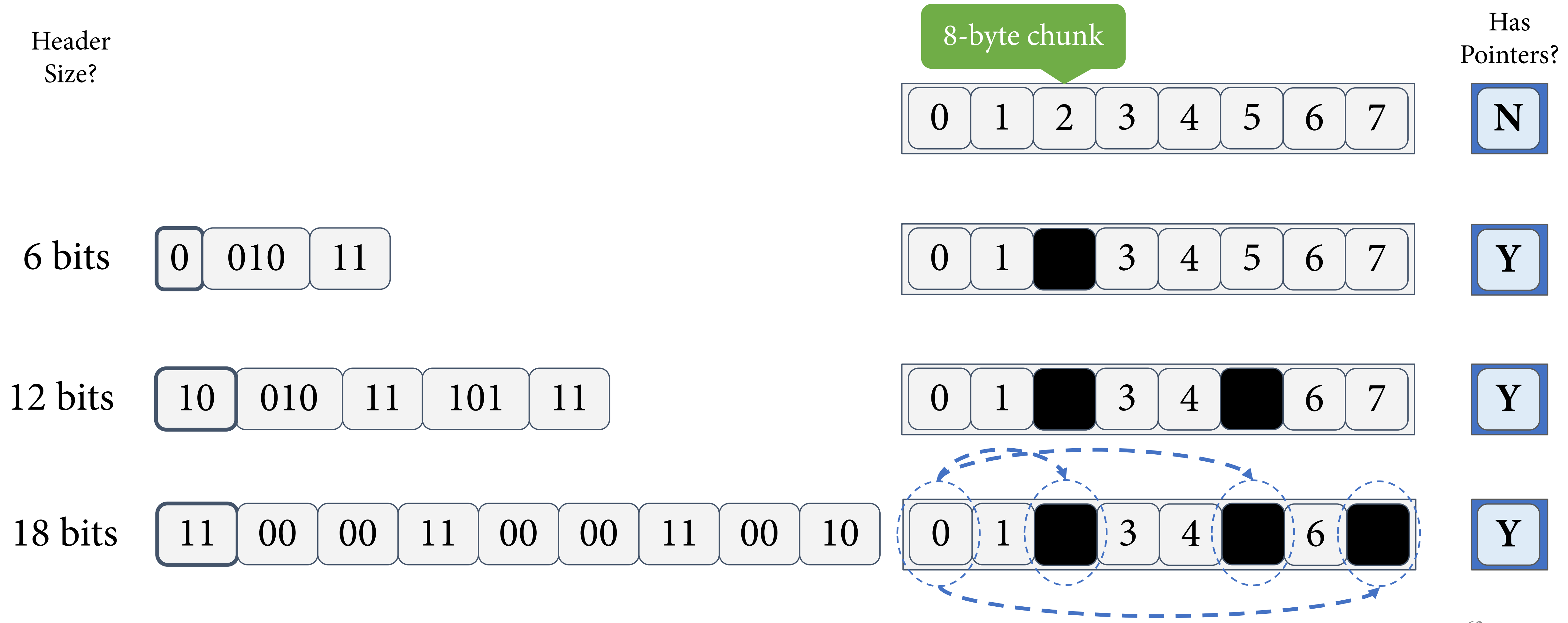
Cache Line Formats

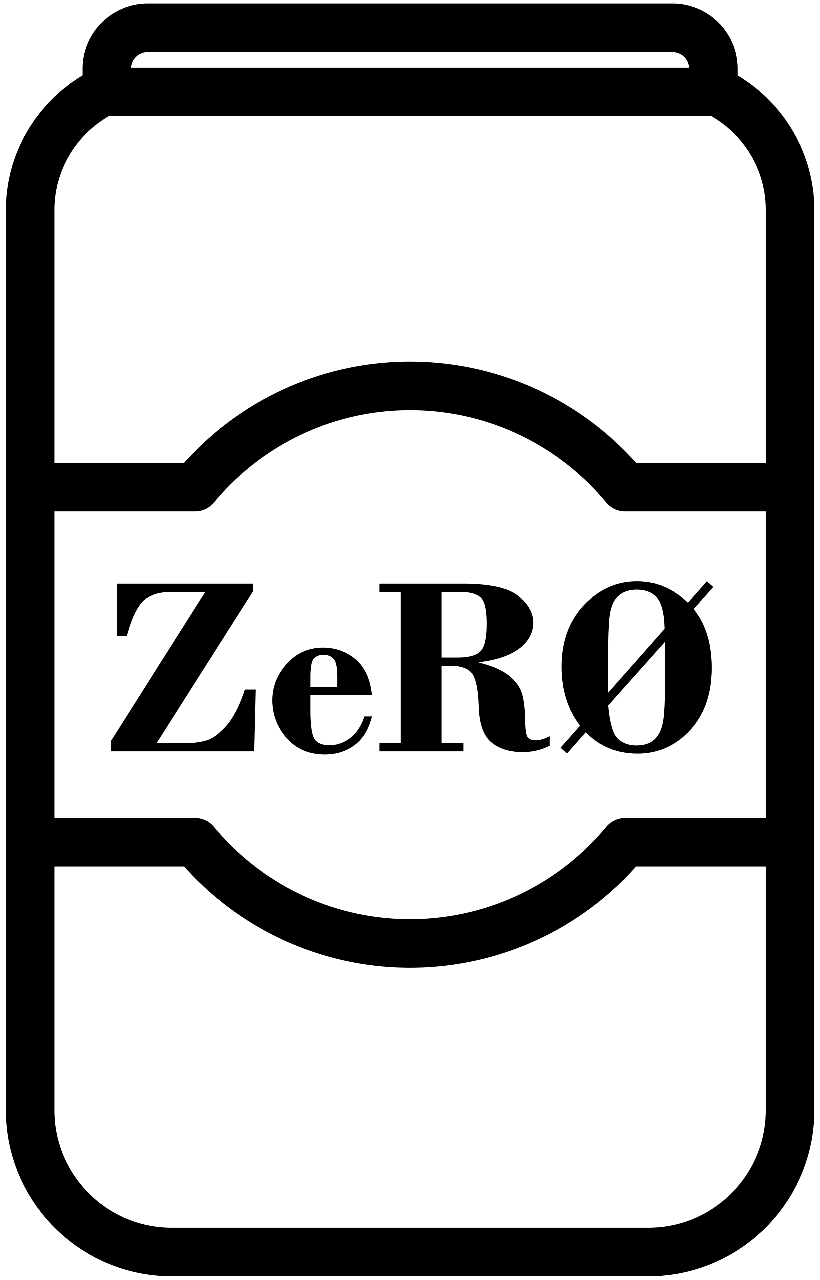


Cache Line Formats



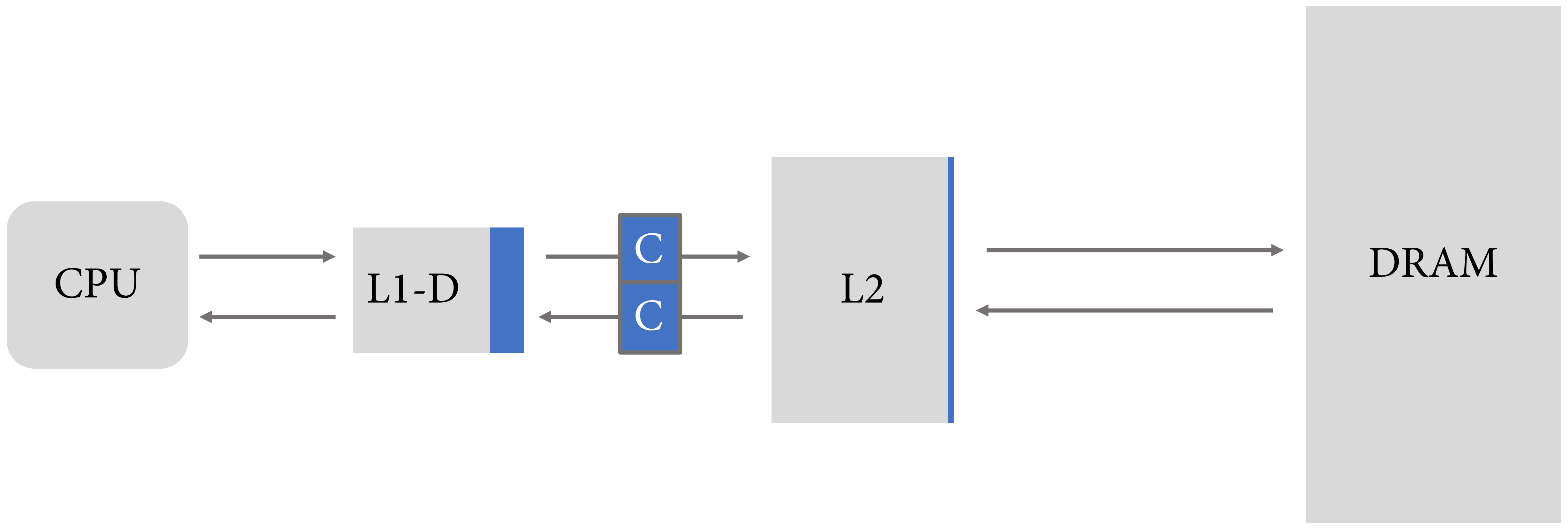
Cache Line Formats



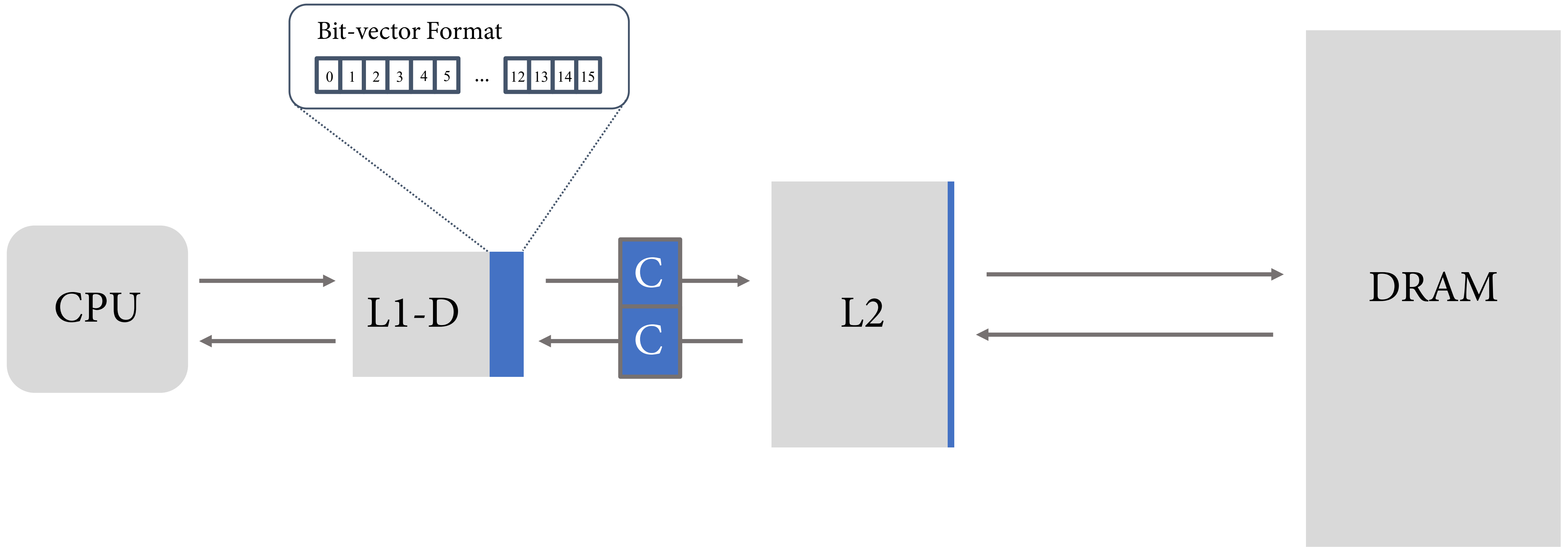


Microarchitectural Overview

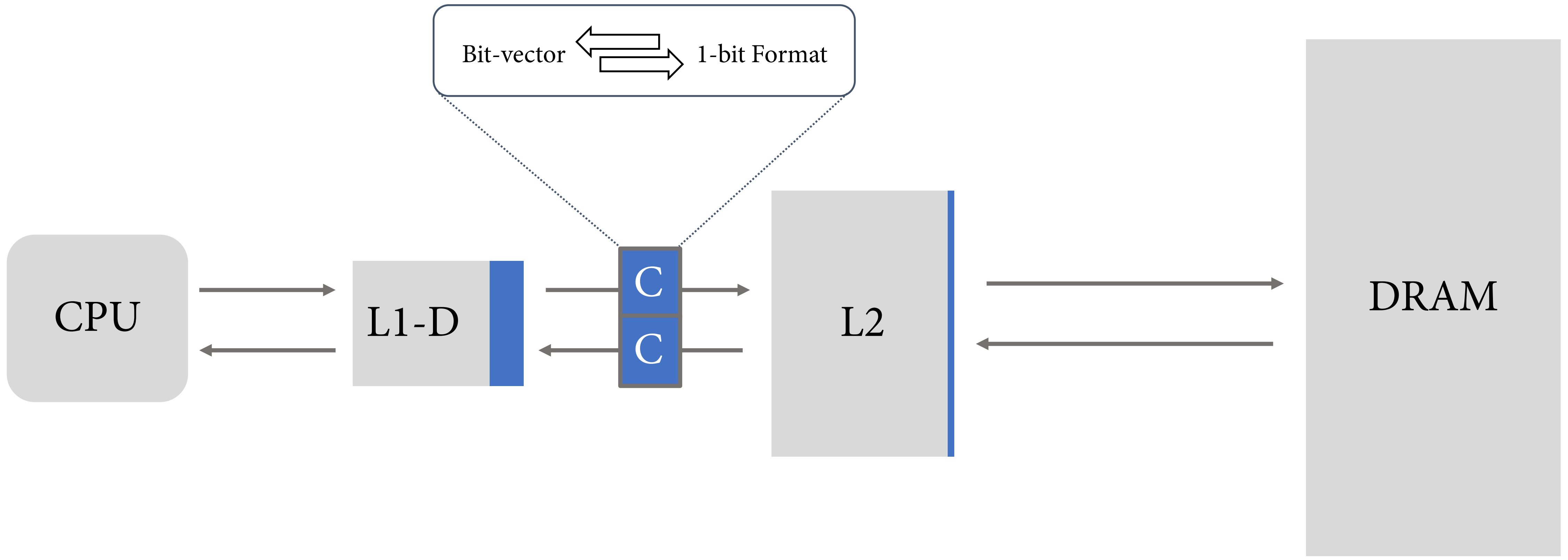
Microarchitectural Overview



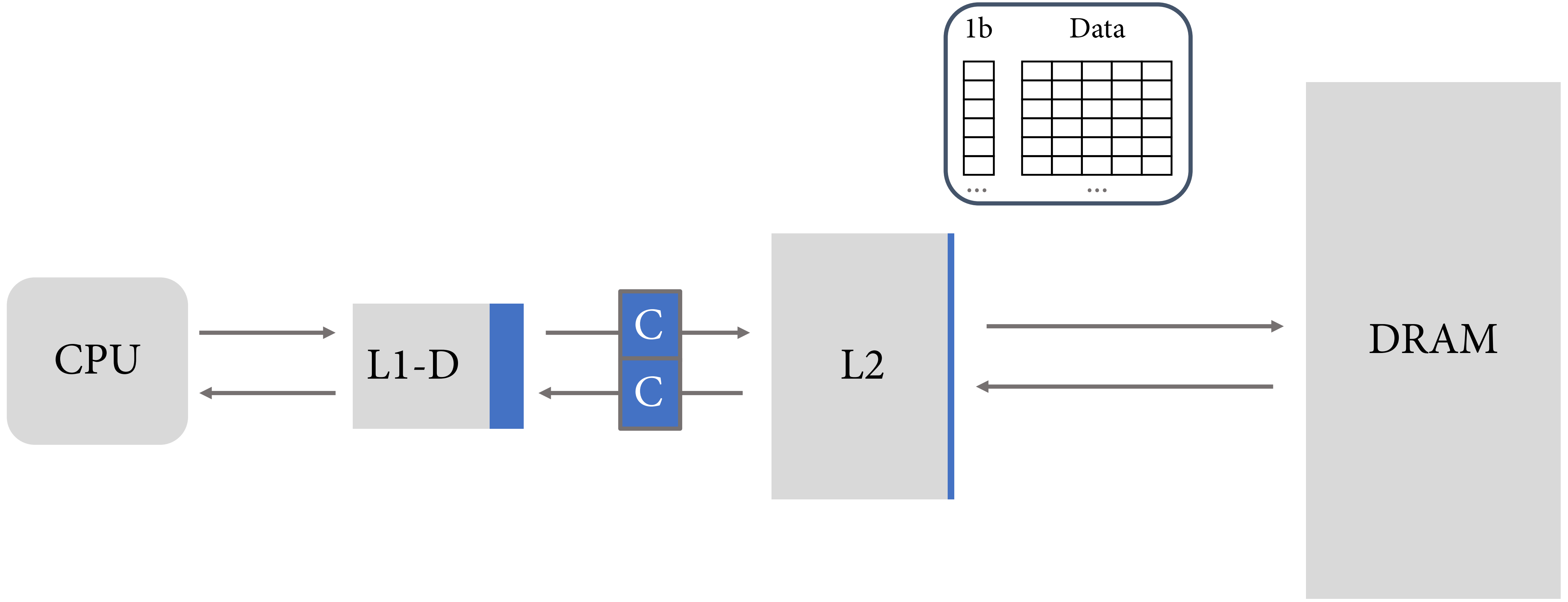
Microarchitectural Overview



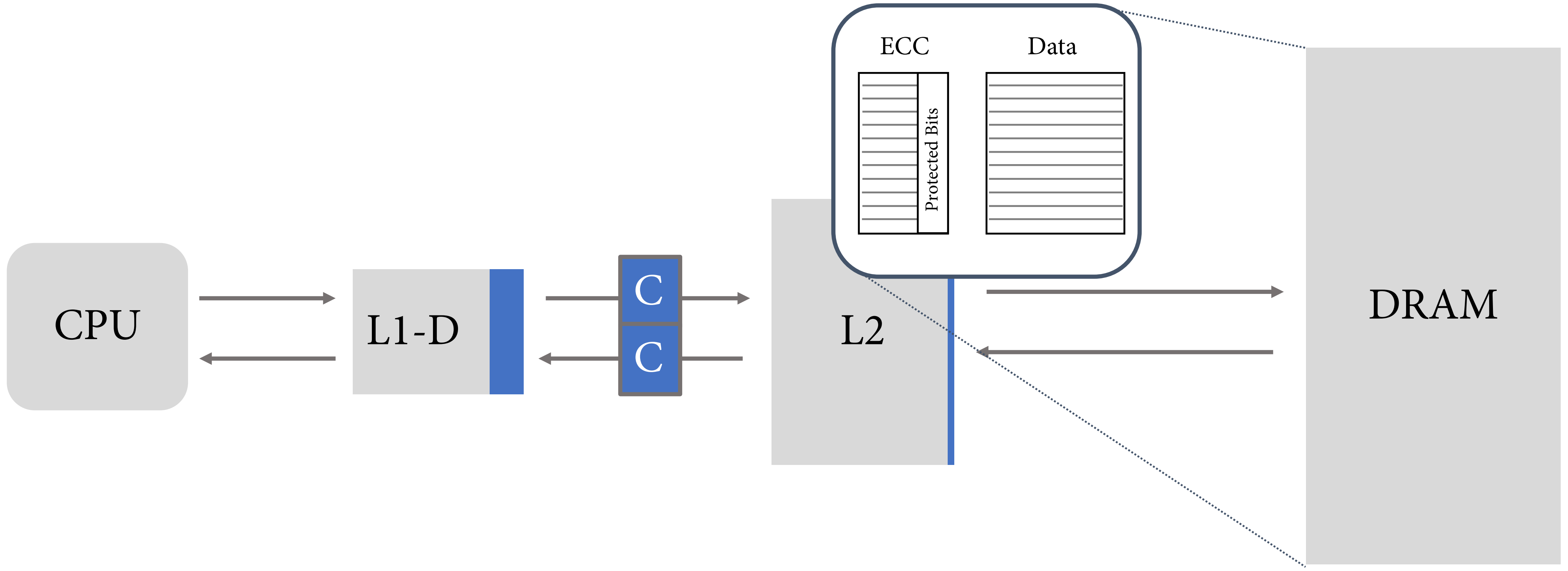
Microarchitectural Overview



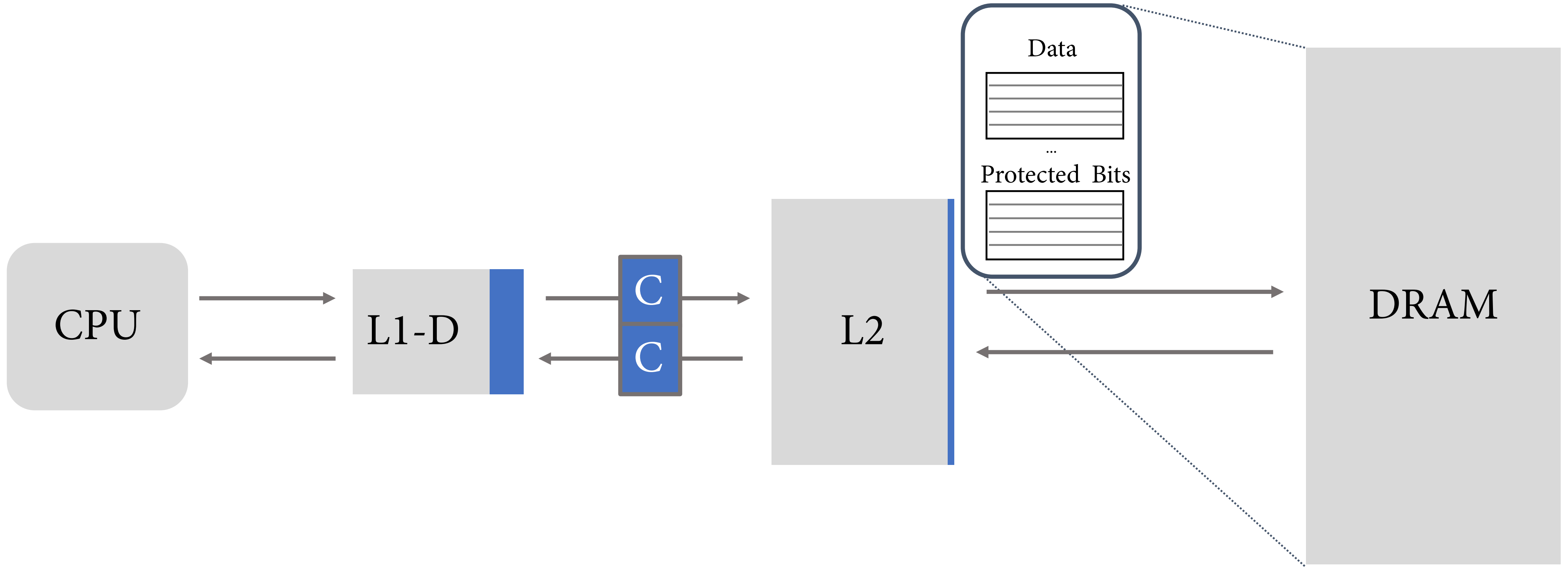
Microarchitectural Overview

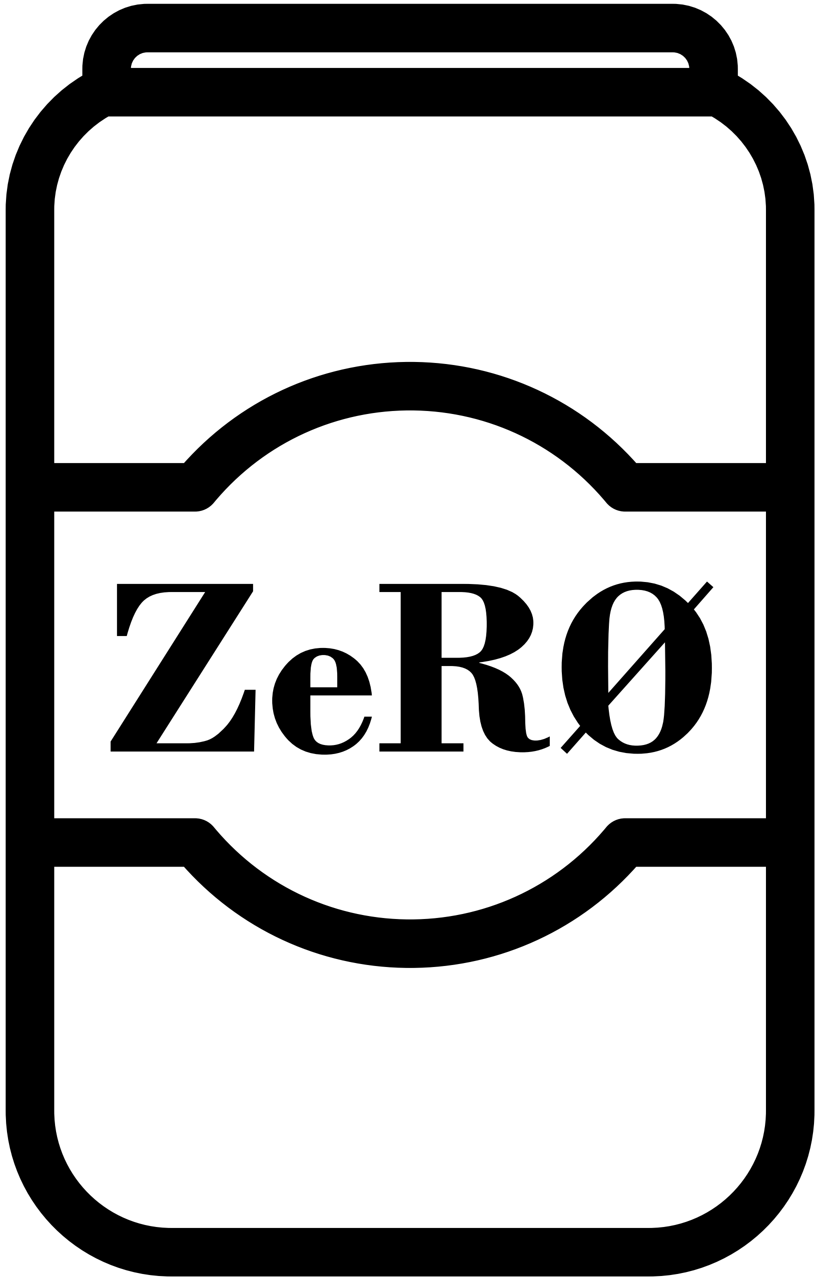


Microarchitectural Overview



Microarchitectural Overview





Performance

ZeRØ Performance Overheads

Hardware Modifications

ZeRØ Performance Overheads

Hardware Modifications
Our hardware measurements show minimal latency/area/power overheads.

ZeRØ Performance Overheads

Hardware Modifications
Our hardware measurements show minimal latency/area/power overheads.

00010010
101001101
00010010
111001001
00010010

Software Modifications

ZeRØ Performance Overheads

Hardware Modifications
Our hardware measurements show minimal latency/area/power overheads.

Software Modifications

- Our special load/stores do not change the binary size.

ZeRØ Performance Overheads

Hardware Modifications
Our hardware measurements show minimal latency/area/power overheads.

00010010
101001101
00010010
111001001
00010010

Software Modifications

- Our special load/stores do not change the binary size.
- The `ClearMeta` instructions are only called on memory deallocation.

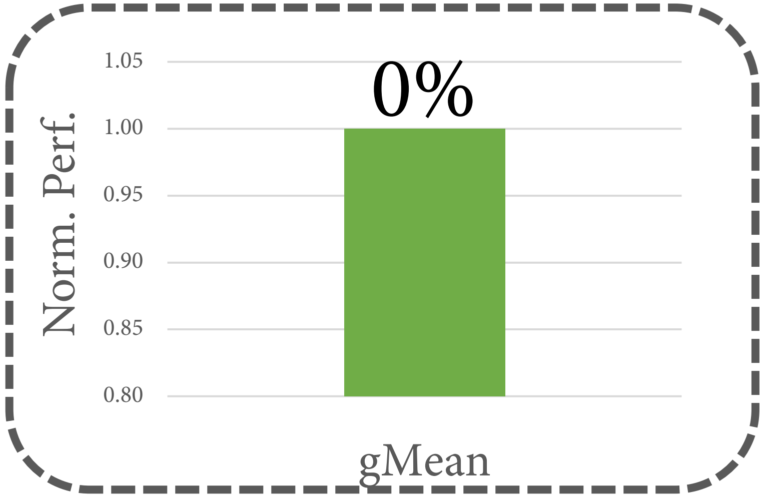
Performance Results (x86_64)

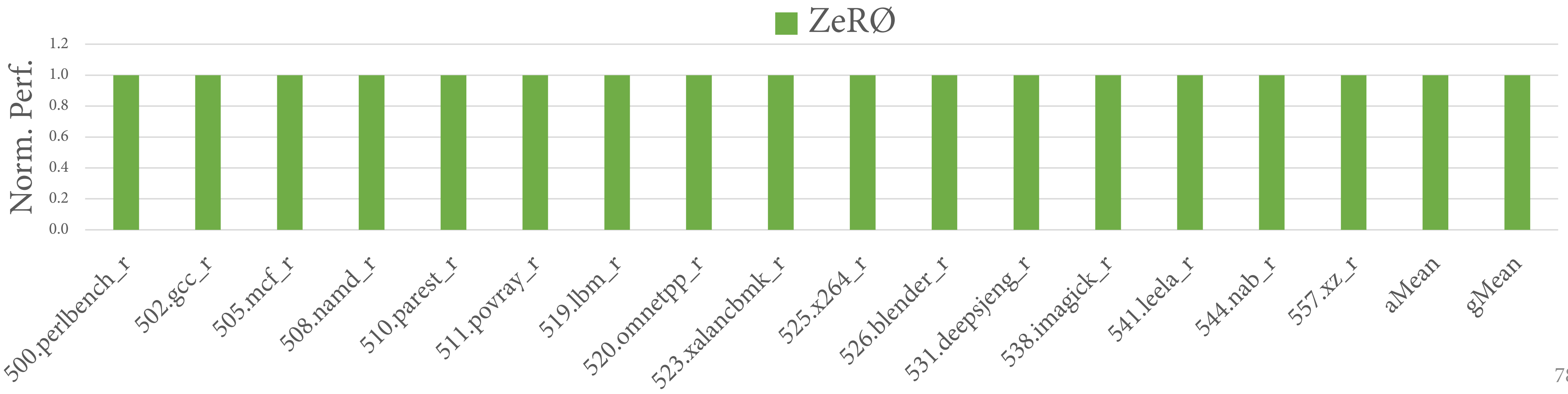
Experimental Setup

We use emulate ZeRØ on x86_64 by modifying LLVM to emit new instructions.

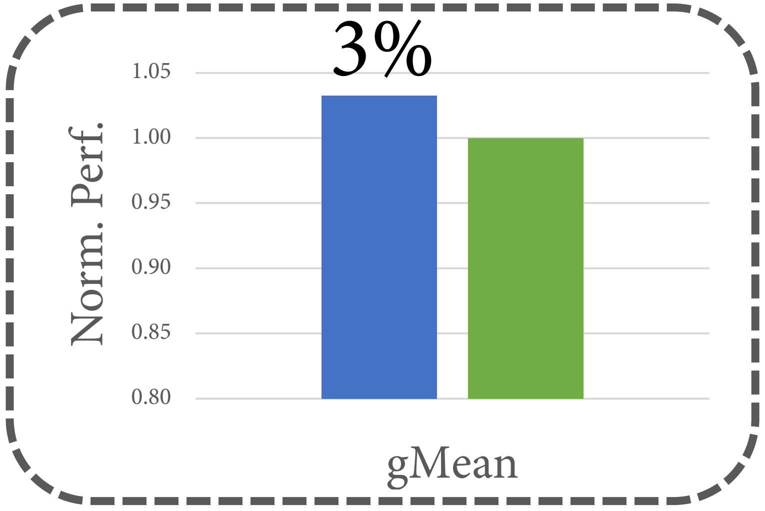
- `ClearMeta` is emulated using dummy stores.

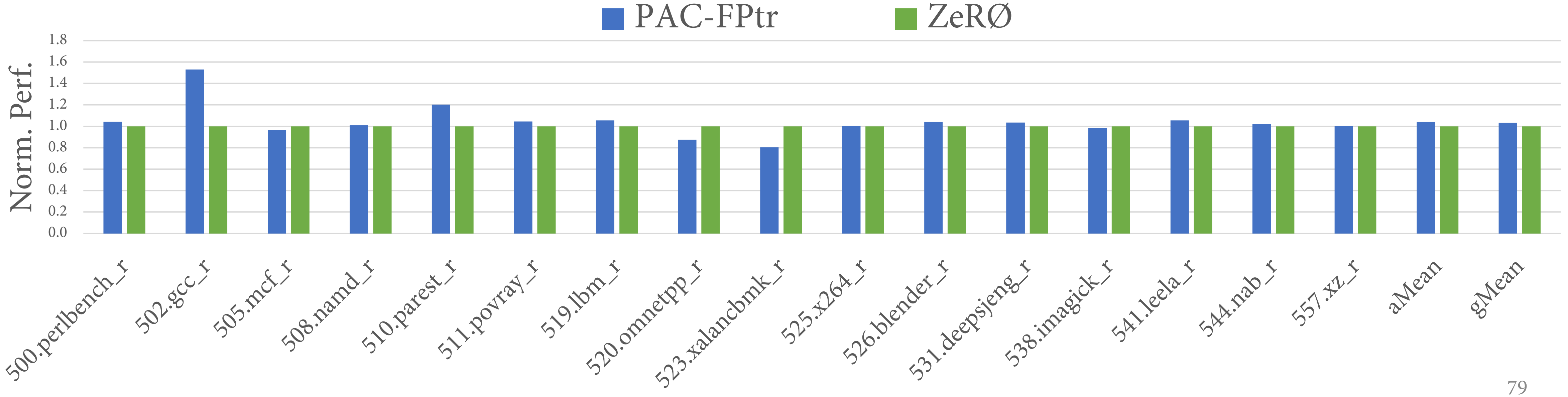
Performance Results (x86_64)



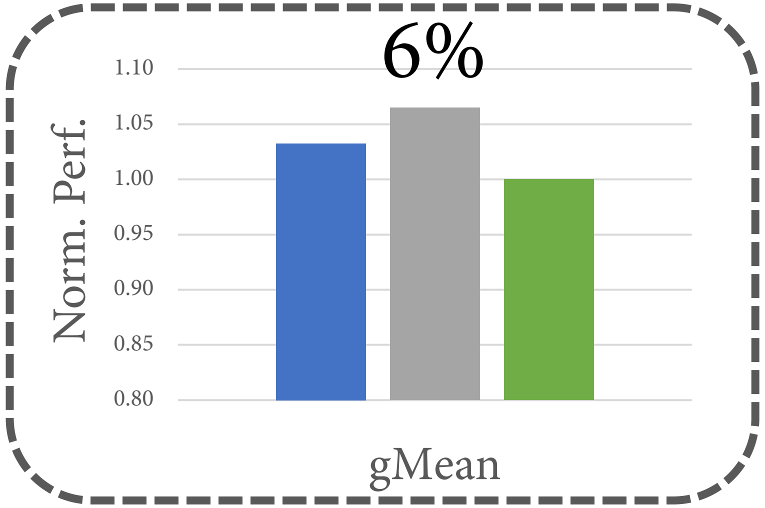


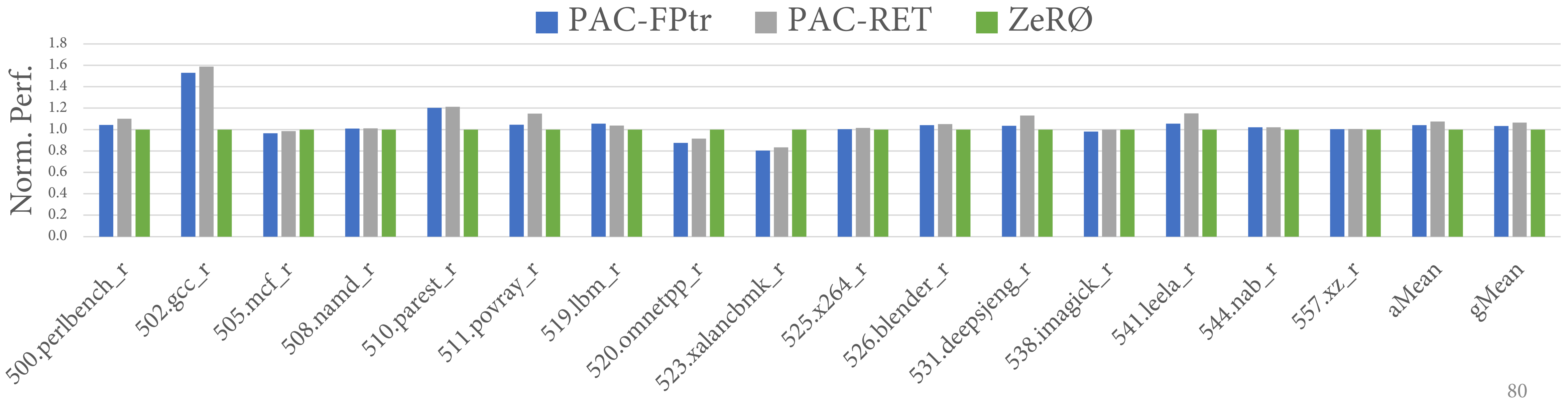
Performance Results (x86_64)



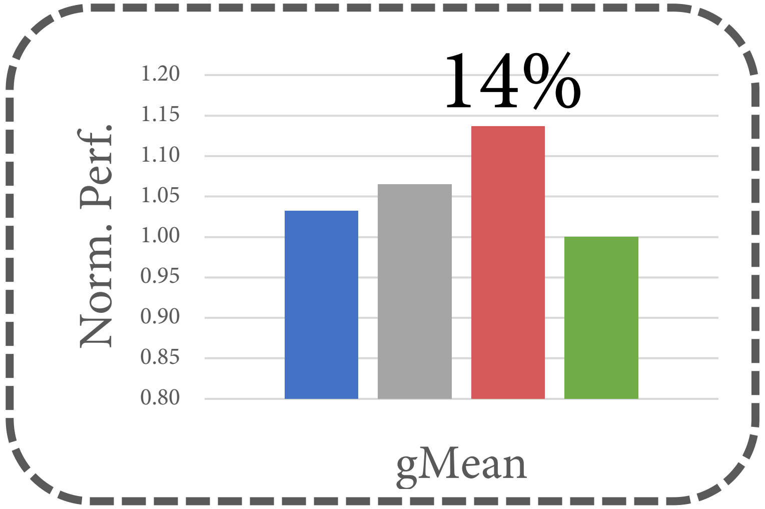


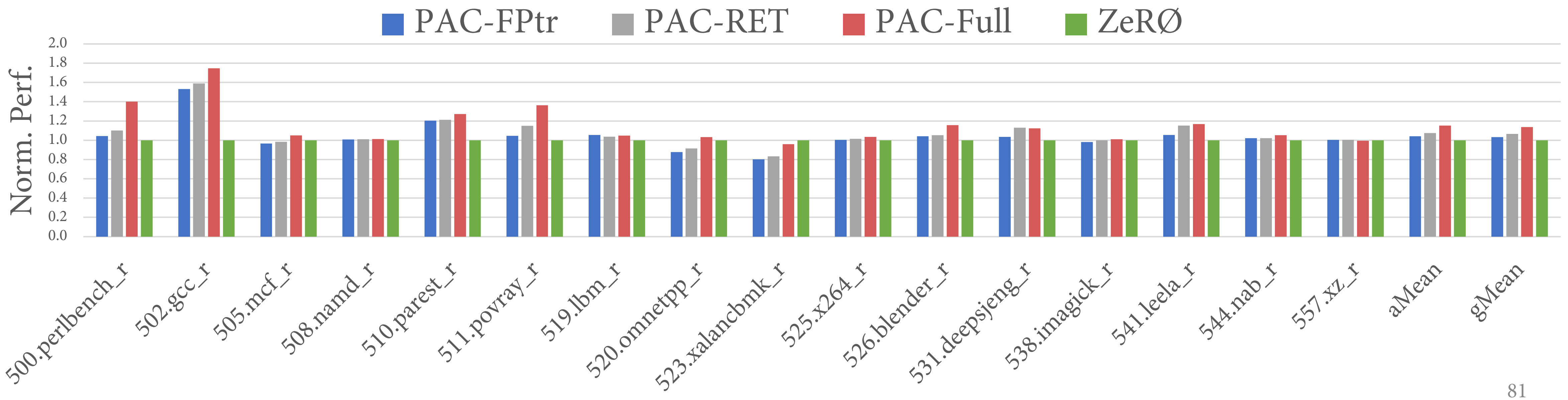
Performance Results (x86_64)



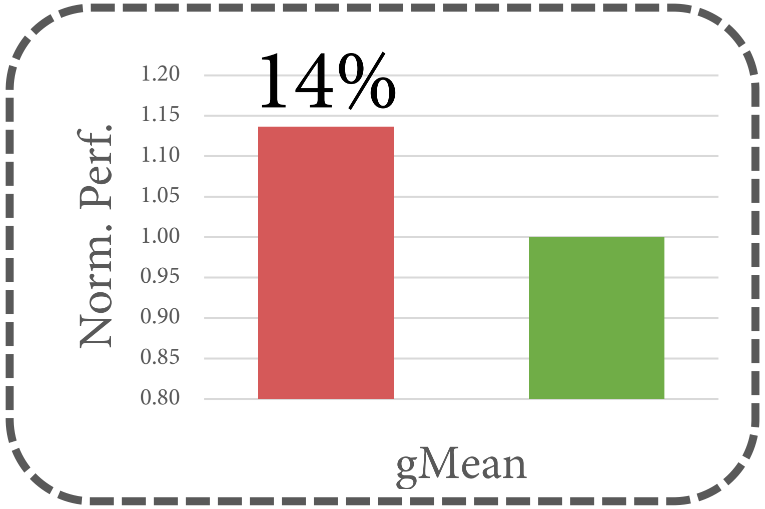


Performance Results (x86_64)





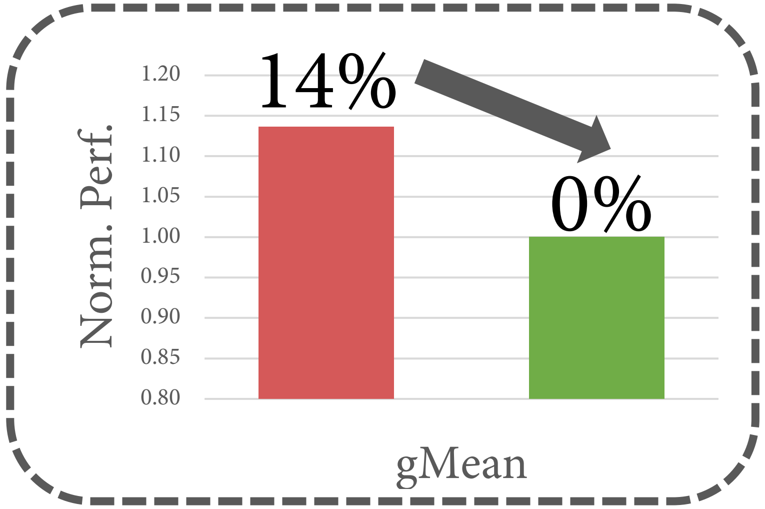
Performance Results (x86_64)



PAC's overheads are attributed to the extra QARMA encryption invocations upon pointer:

- loads/stores
- usages

Performance Results (x86_64)



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

ZeRØ does not compromise on security

No Pointer Manipulation
Protects against all known pointer manipulation attacks (e.g. ROP, JOP/COP, COOP, DOP).

Handling Security Violations

Advisory Exceptions

- Skip faulty instructions.
- Do NOT crash the running process.

Handling Security Violations

Advisory Exceptions

- Skip faulty instructions.
- Do NOT crash the running process.

Permit List

- Initialized during program startup

Handling Security Violations

Advisory Exceptions

- Skip faulty instructions.
- Do NOT crash the running process.

Permit List

- Initialized during program startup
- Avoid false alarms for non-type aware functions (e.g., `memcpy` and `memmove`)

Handling Third Party Code

We can pick from the following options:

Handling Third Party Code

We can pick from the following options:

1 **Compile with ZeRØ**
Compile third party code with ZeRØ support.

Handling Third Party Code

We can pick from the following options:

1 **Compile with ZeRØ**
Compile third party code with ZeRØ support.

2 **Add to Permit List**
Add to a permit list during program initialization.

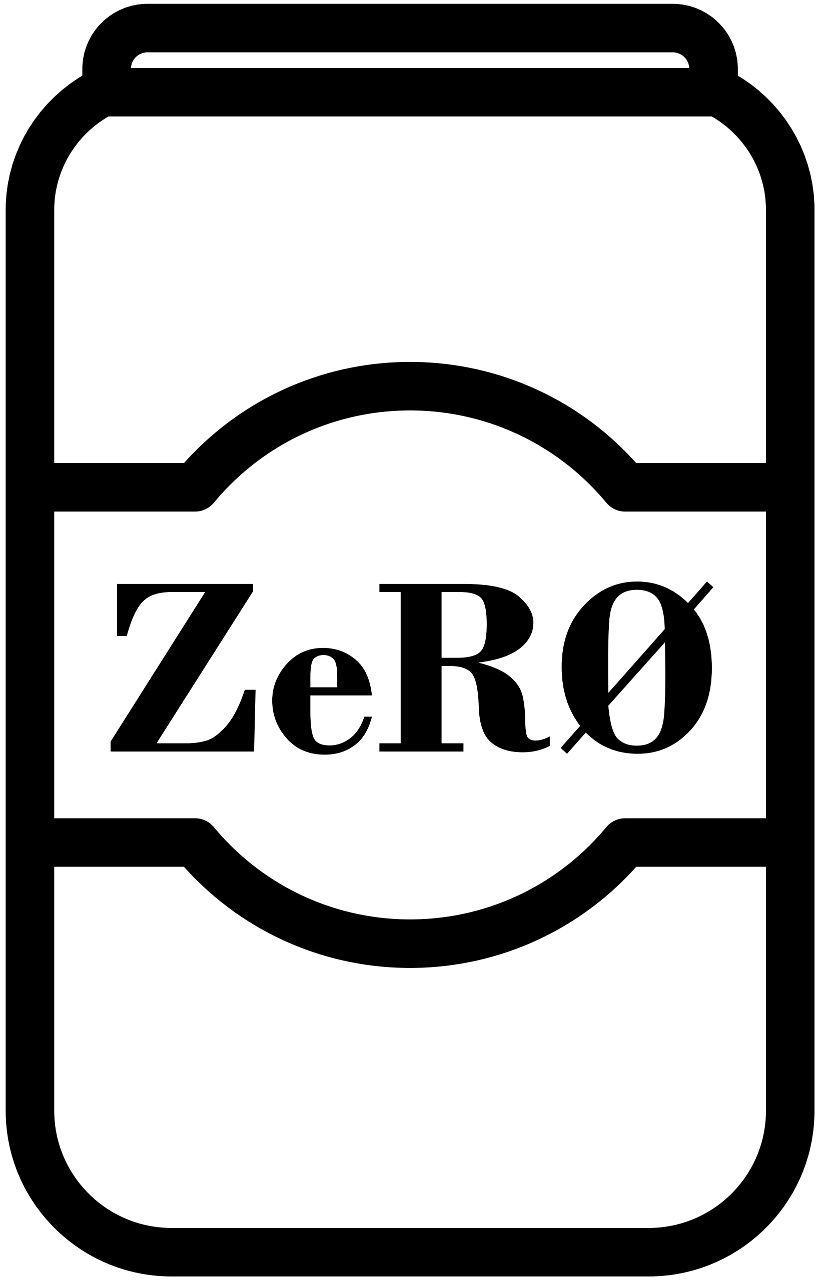
Handling Third Party Code

We can pick from the following options:

1. **Compile with ZeRØ**
   Compile third party code with ZeRØ support.

2. **Add to Permit List**
   Add to a permit list during program initialization.

3. **Invoke `ClearMeta`**
   `ClearMeta` is inserted before passing pointers to external libraries.



Limitations

Limitations

Non-pointer Data Corruption
These attacks require a full memory safety solution.

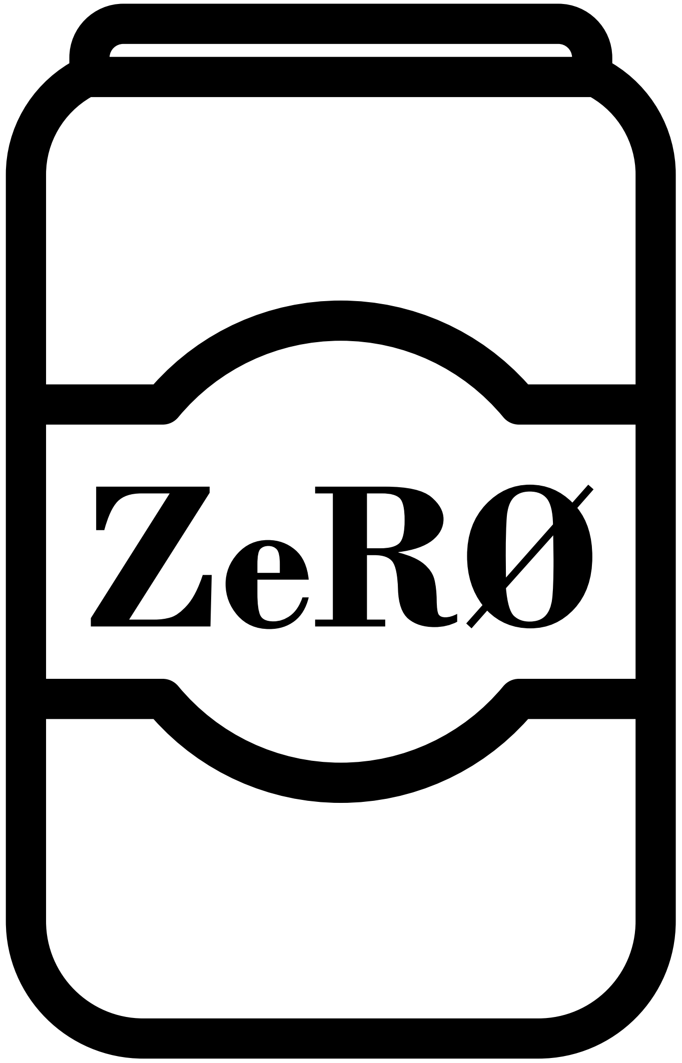
Limitations

Non-pointer Data Corruption
These attacks require a full memory safety solution.

Full Memory Safety
No-FAT is well suited for cloud/server and end-user deployments.

Checkout our paper & talk!
https://isca21.arroyo.me

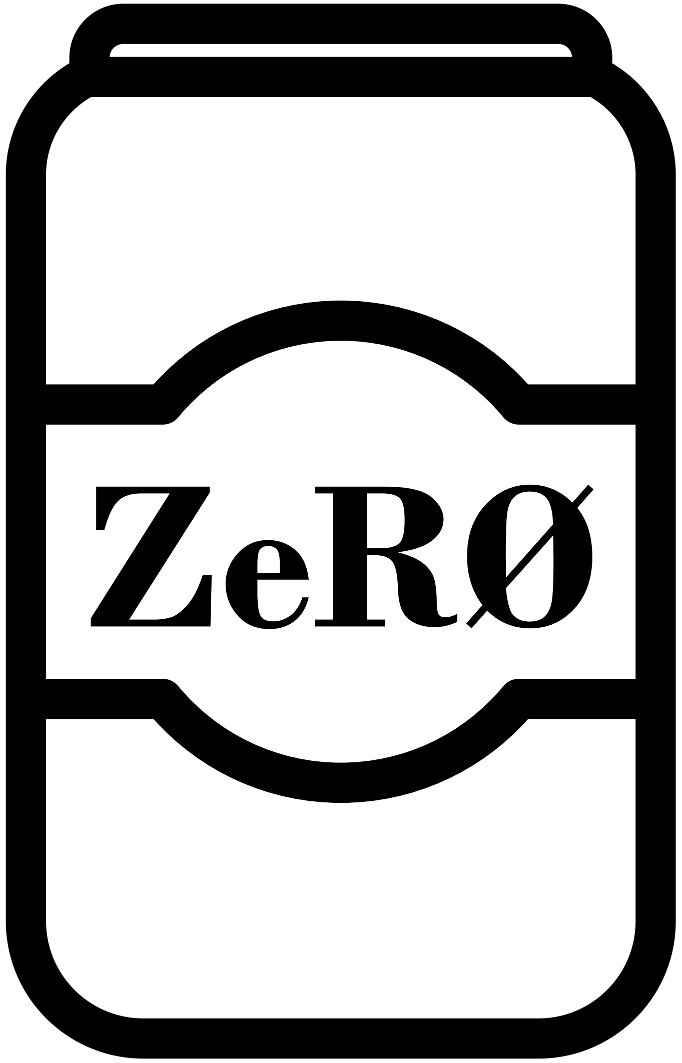
An efficient pointer integrity mechanism



An ideal candidate for end-user deployment.

- ✓ **Easy to Implement**
- ✓ **No Runtime Overheads**
- ✓ **Offers Robust Security**

An efficient pointer integrity mechanism



An ideal candidate for end-user deployment.

- ✓ **Easy to Implement**
- ✓ **No Runtime Overheads**
- ✓ **Offers Robust Security**

A drop-in replacement for ARM PAC