# Practical Byte-Granular Memory Blacklisting using Califorms

Hiroshi Sasaki, Miguel A. Arroyo, Mohamed Tarek Ibn Ziad, Koustubha Bhat, Kanad Sinha, Simha Sethumadhavan

Columbia University
IN THE CITY OF NEW YORK

# Californs

Hiroshi Sasaki, Miguel A. Arroyo, Mohamed Tarek Ibn Ziad,
Koustubha Bhat, Kanad Sinha, Simha Sethumadhavan

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

# 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.

**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***

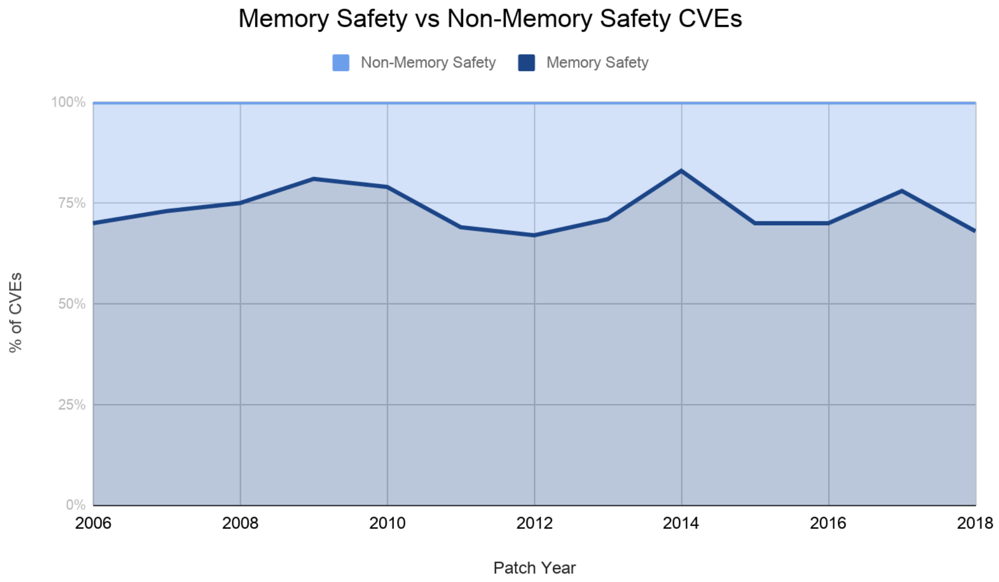
**Exclusive: Saudi Dissidents Hit With Stealth iPhone Spyware Before Khashoggi's Murder**

# IT'S EASY TO MAKE MISTAKES

# IT’S EASY TO MAKE MISTAKES

SEGFAULT!

# PREVALENCE OF MEMORY SAFETY VULNS

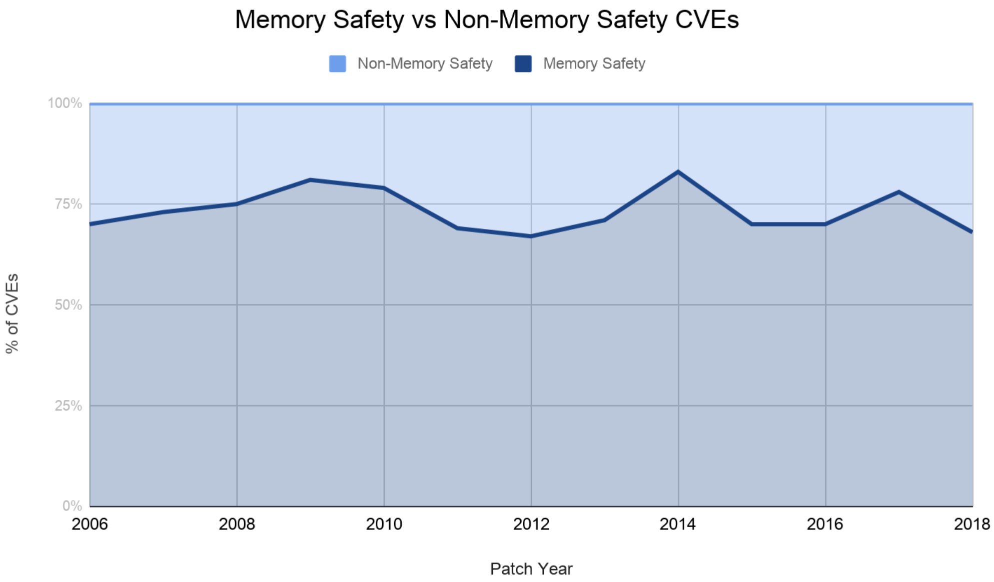



Microsoft Product CVEs

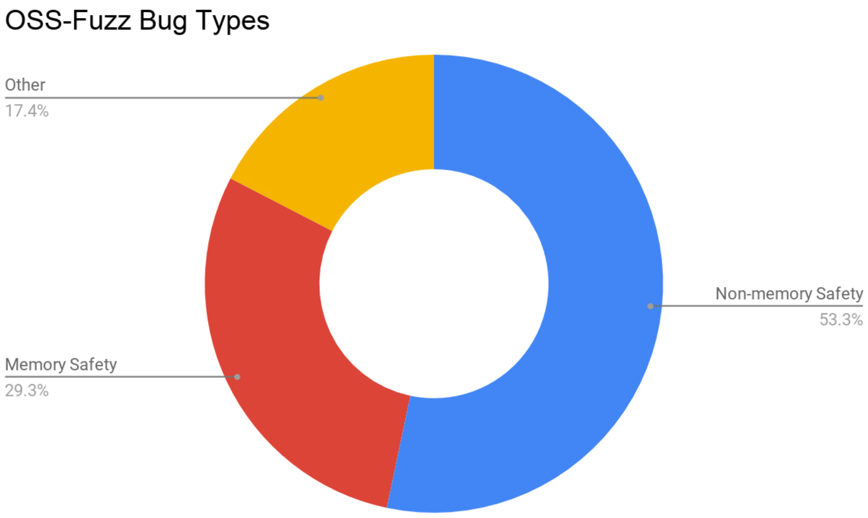


***Source:*** *Matt Miller, Microsoft Security Response Center (MSRC) - BlueHat 2019*

# PREVALENCE OF MEMORY SAFETY VULNS




Microsoft Product CVEs




Google OSS-Fuzz bugs from 2016-2018.



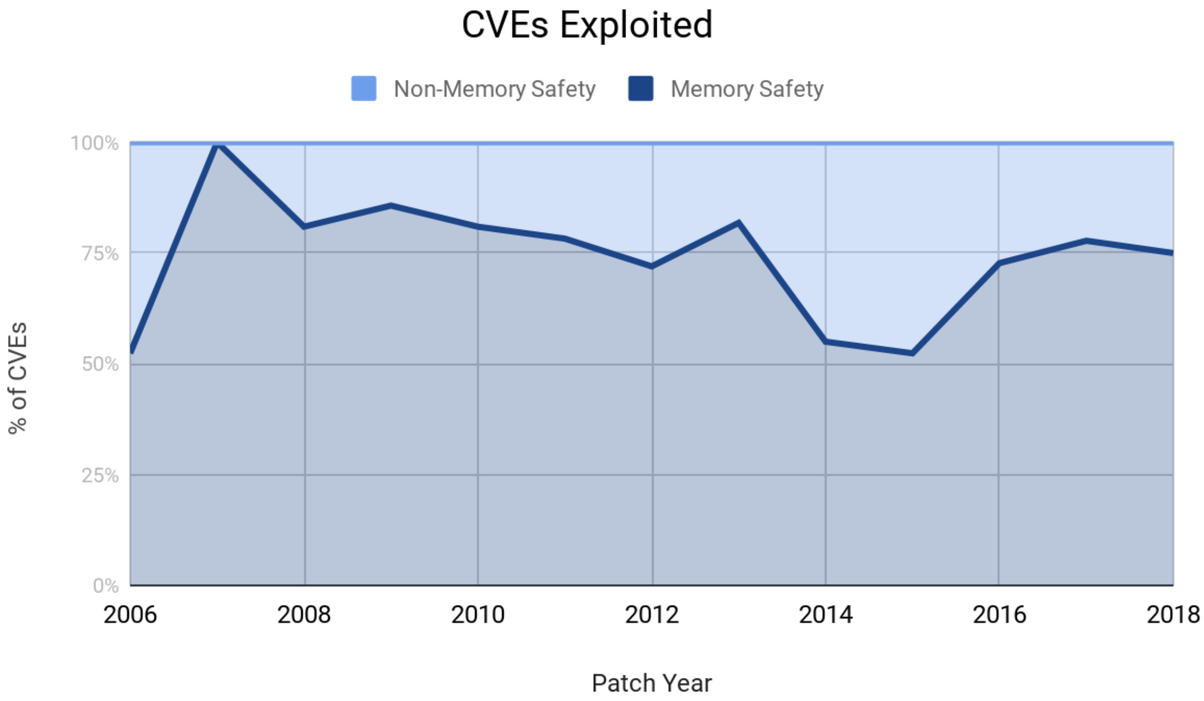
**Source:** *Matt Miller, Microsoft Security Response Center (MSRC) - BlueHat 2019*

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

# ATTACKERS

# MEMORY SAFETY

# ATTACKERS PREFER MEMORY SAFETY VULNS




Microsoft Product Exploits

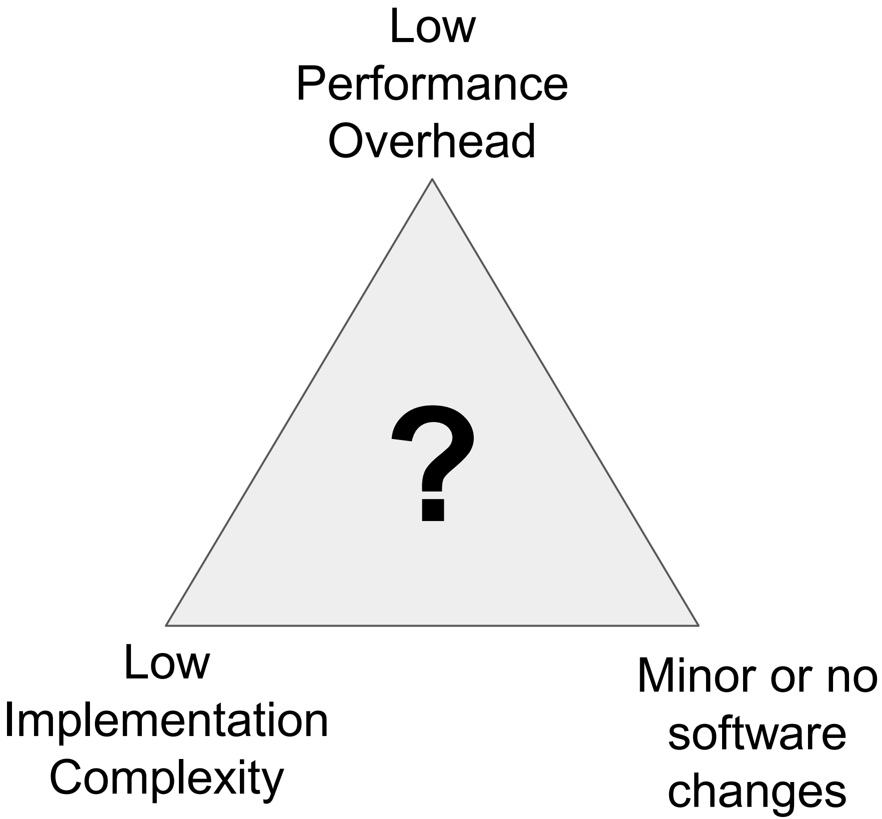


**Source:** *Matt Miller, Microsoft Security Response Center (MSRC) - BlueHat 2019*

RESEARCHERS TOO

MEMORY SAFETY

# CURRENT SOLUTIONS AREN'T PRACTICAL

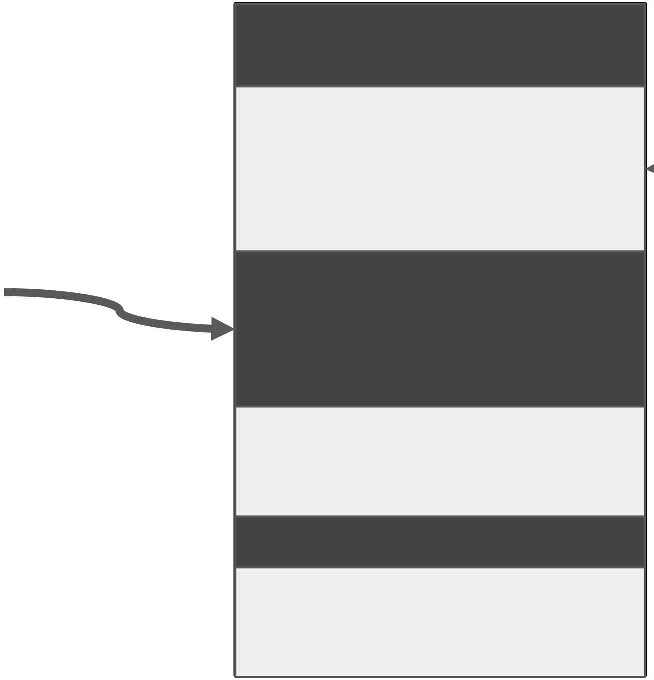


# CALIFORMS

- Low Overhead
- Robust Security
- Legacy Software Compatibility
- Easy to Implement
- 32-bit Compatible

# MEMORY BLACKLISTING

*A tripwire is a **blacklisted** location.*

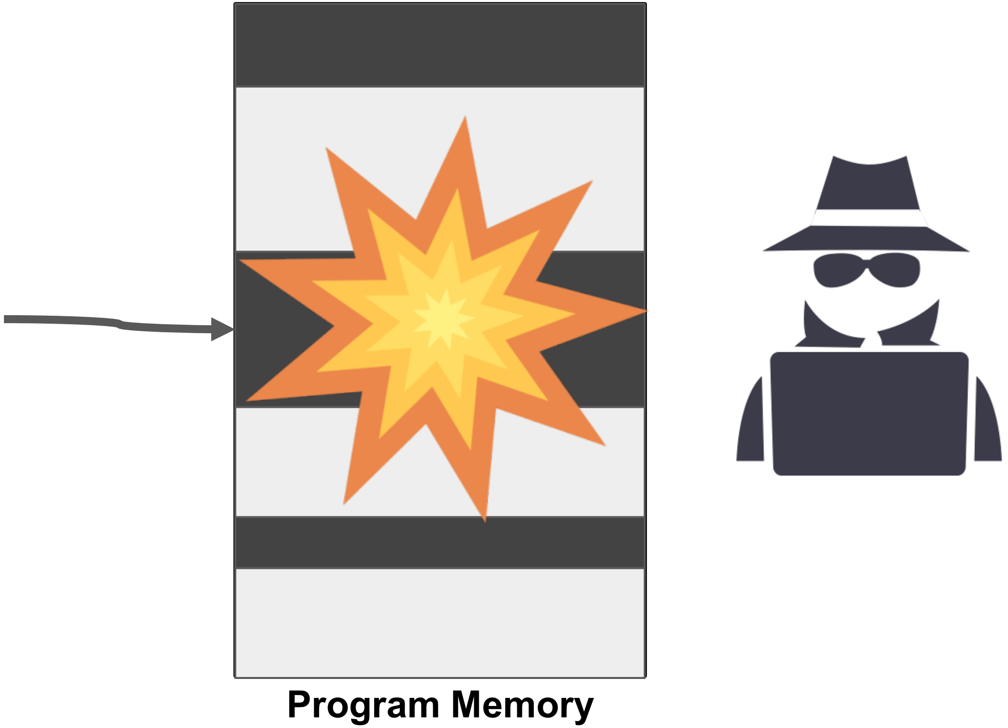


*This is program data.*

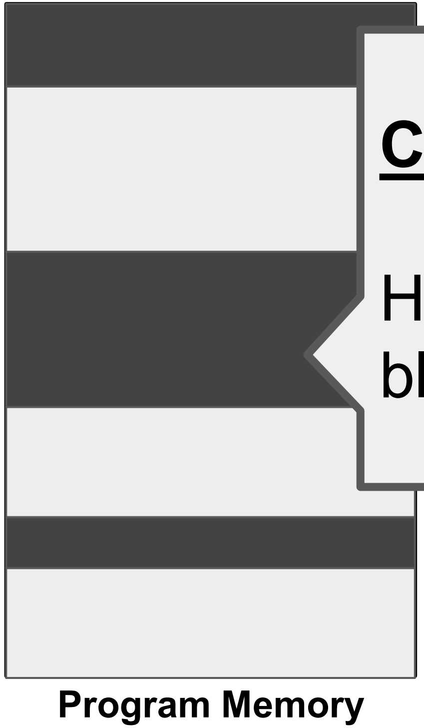
**Program Memory**

# MEMORY BLACKLISTING

*Accesses to this region trigger an **exception!***



# MEMORY BLACKLISTING

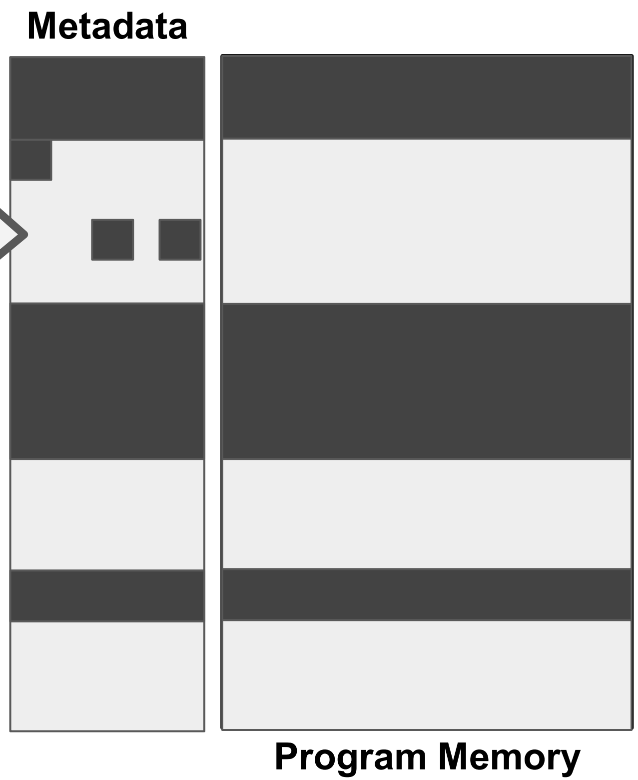



**<u>Challenge</u>**

How to efficiently **identify** blacklisted locations?

# MEMORY BLACKLISTING

**Simple**
1-bit per byte



# MEMORY BLACKLISTING

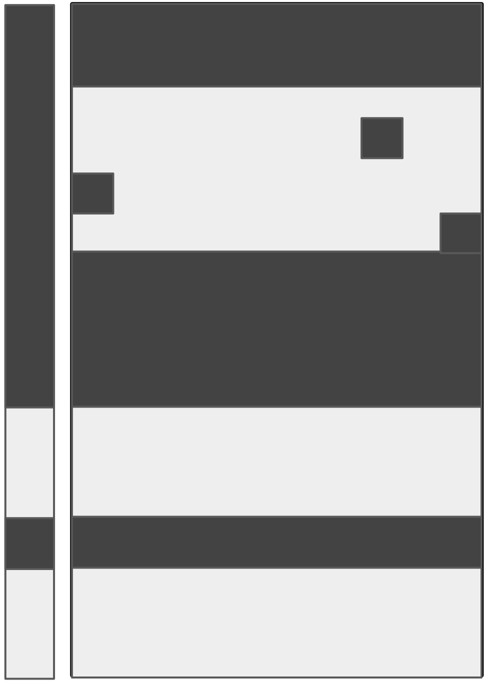
**Simple**
1-bit per byte


**12.5% memory** overhead!

**~200% runtime** overhead!

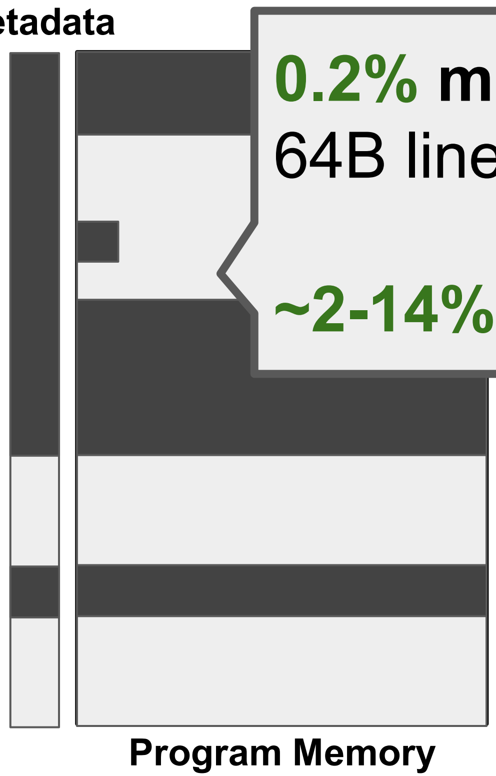
# MEMORY BLACKLISTING

**Califorms**
1-bit per cache line



# MEMORY BLACKLISTING

**Califorms**
1-bit per cache line




**0.2% memory** overhead for 64B line!

**~2-14% runtime** overhead!

# OUR TALK

- Califorms
- Benefits
  - Performance, Security
- Related work
  - State-of-the-art Memory Safety Mitigations
- Conclusion

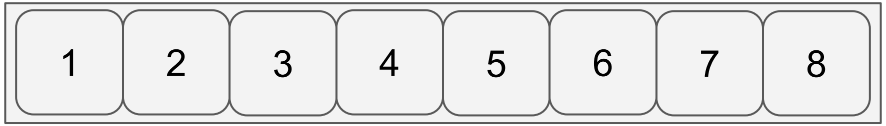
# CALIFORMS

## MEMORY BLACKLISTING

# CALIFORMS: CACHE LINE FORMATS

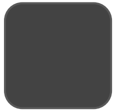
**Our Metadata:** Encoded within unused data.

**Normal**



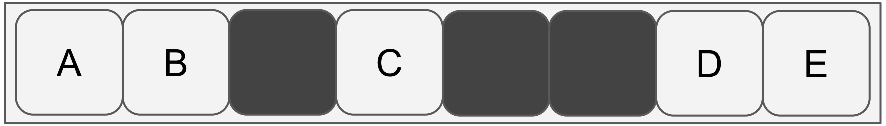
# CALIFORMS: CACHE LINE FORMATS

**Our Metadata:** Encoded within unused data.



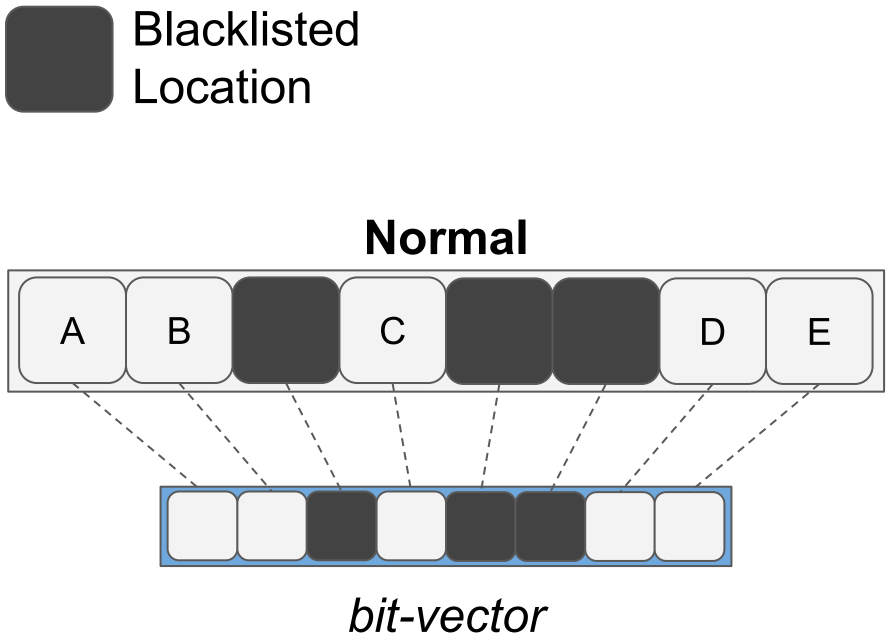
Blacklisted
Location

**Normal**



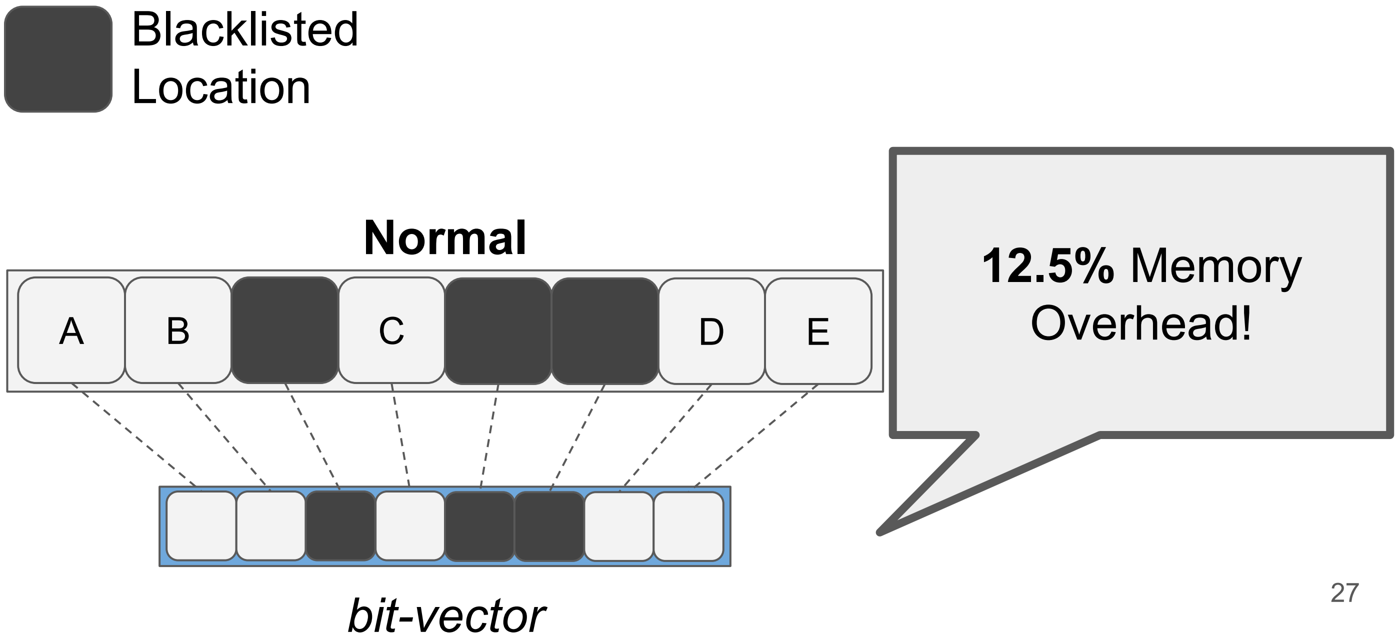
# CALIFORMS: CACHE LINE FORMATS

**Our Metadata:** Encoded within unused data.



# CALIFORMS: CACHE LINE FORMATS

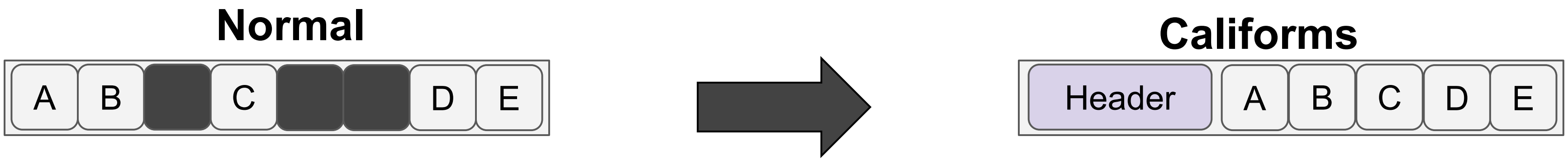
**Our Metadata:** Encoded within unused data.



# CALIFORMS: CACHE LINE FORMATS

**Our Metadata:** Encoded within unused data.

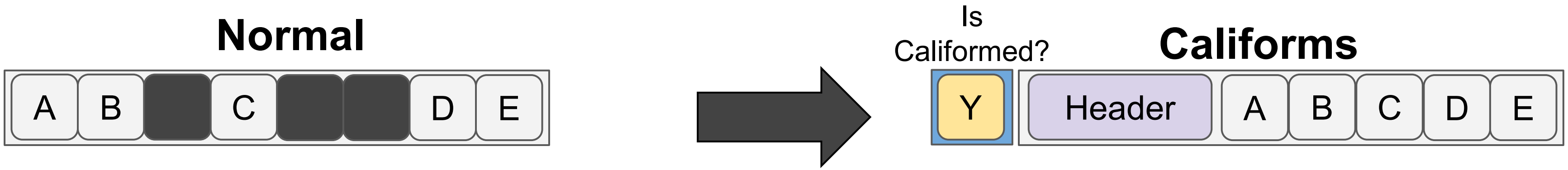
Blacklisted Location



# CALIFORMS: CACHE LINE FORMATS

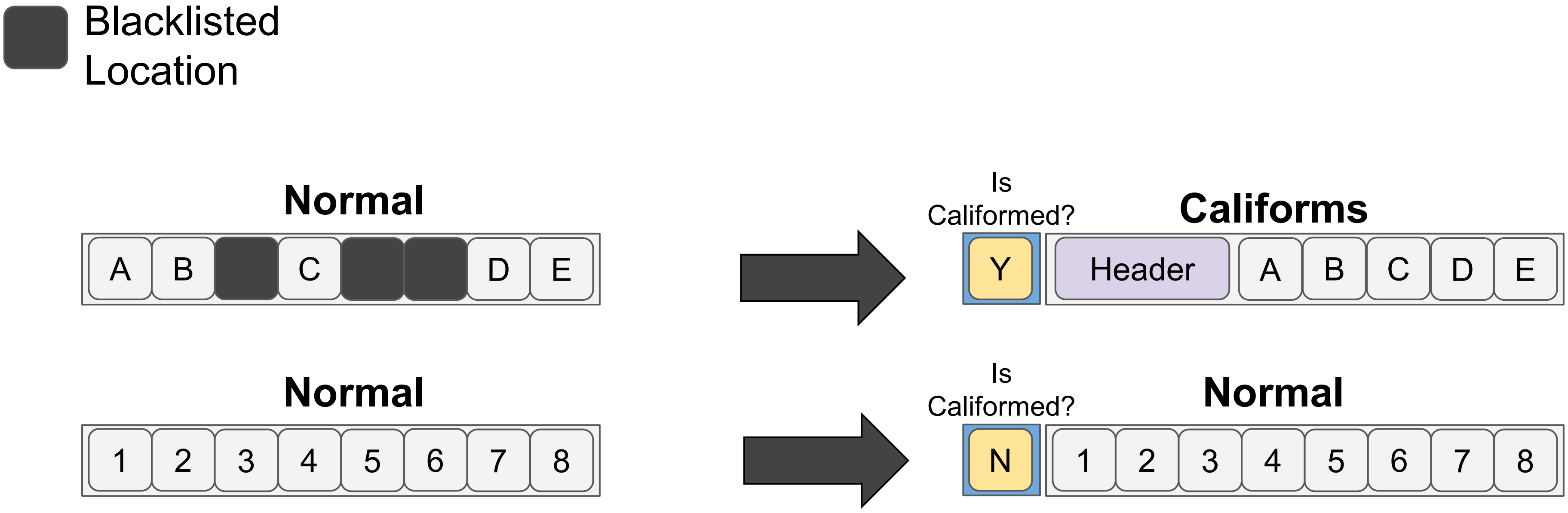
**Our Metadata:** Encoded within unused data.

Blacklisted Location



# CALIFORMS: CACHE LINE FORMATS

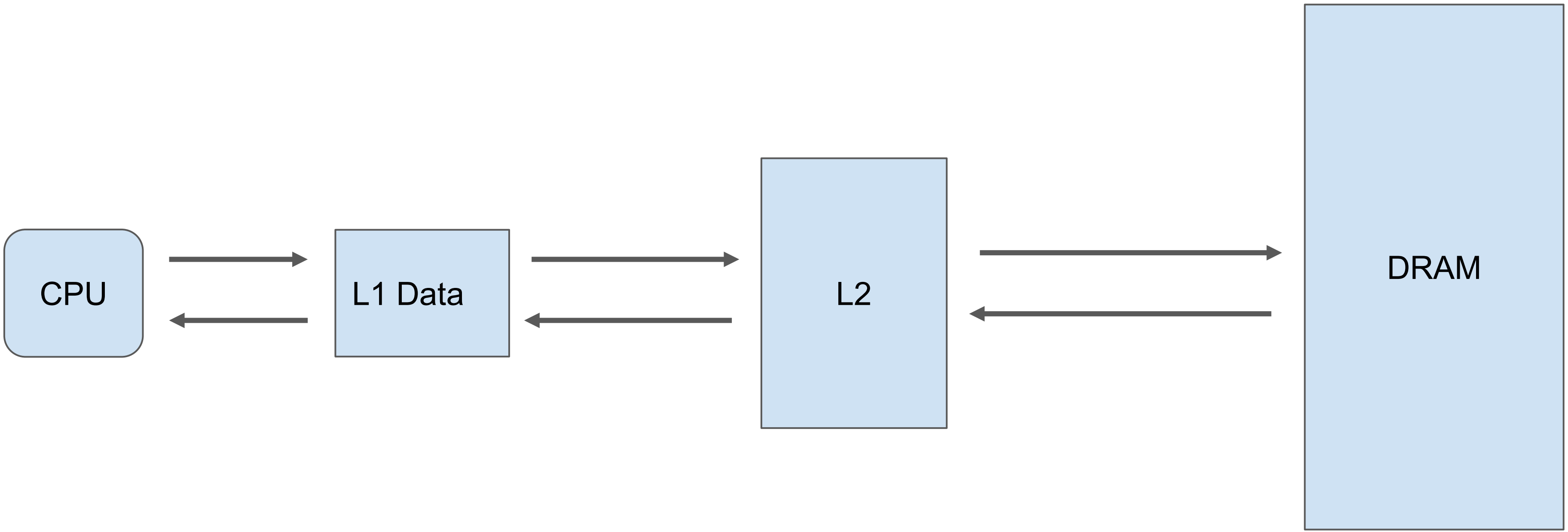
**Our Metadata:** Encoded within unused data.



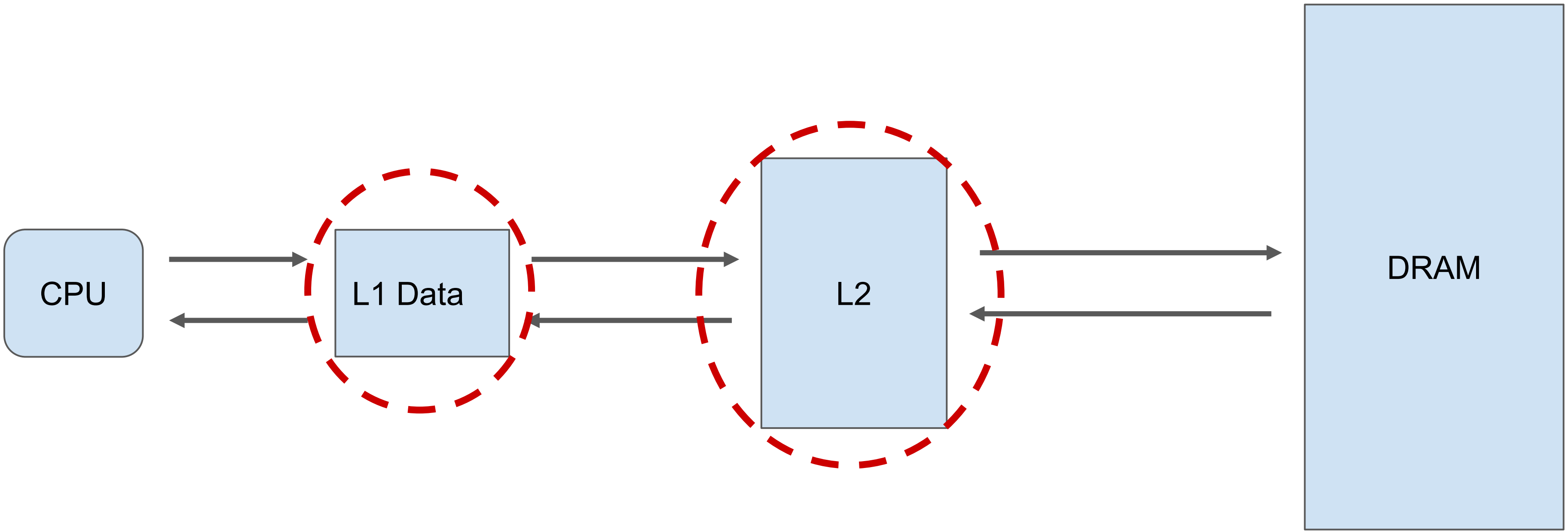
# CALIFORMS: FULL SYSTEM

- **Microarchitecture**
- **Architecture Support**
- **Software**

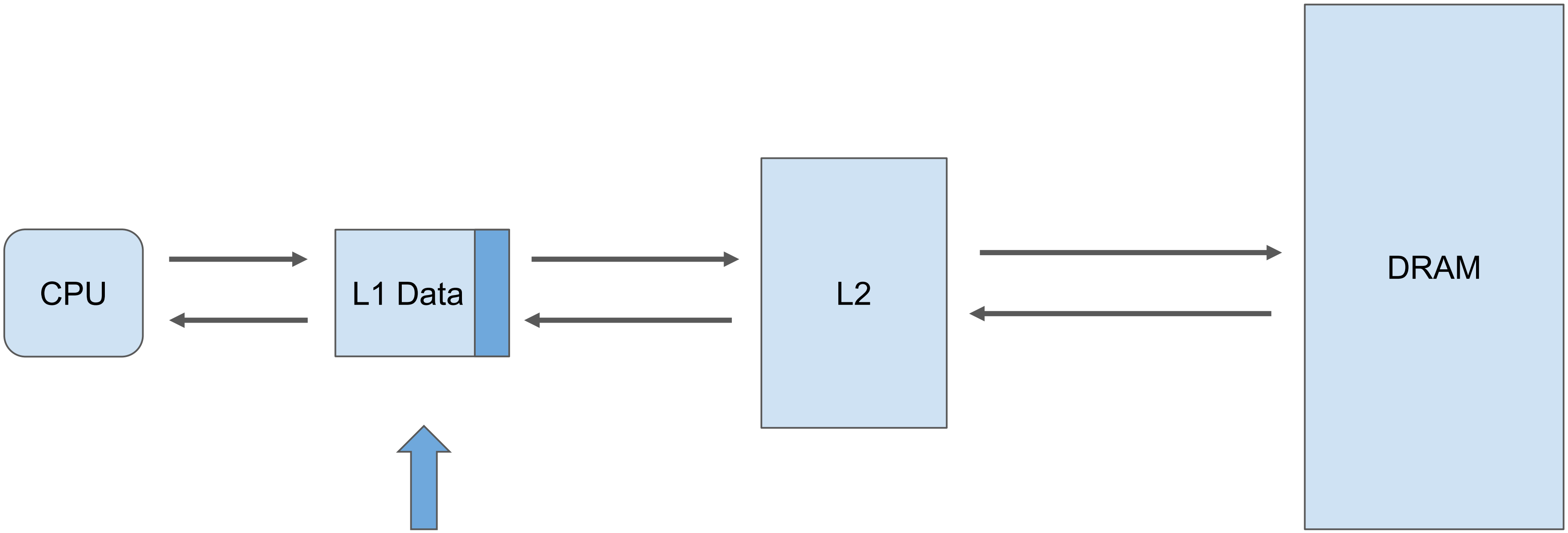
# CALIFORMS: MICROARCHITECTURE



# CALIFORMS: MICROARCHITECTURE



# CALIFORMS: MICROARCHITECTURE

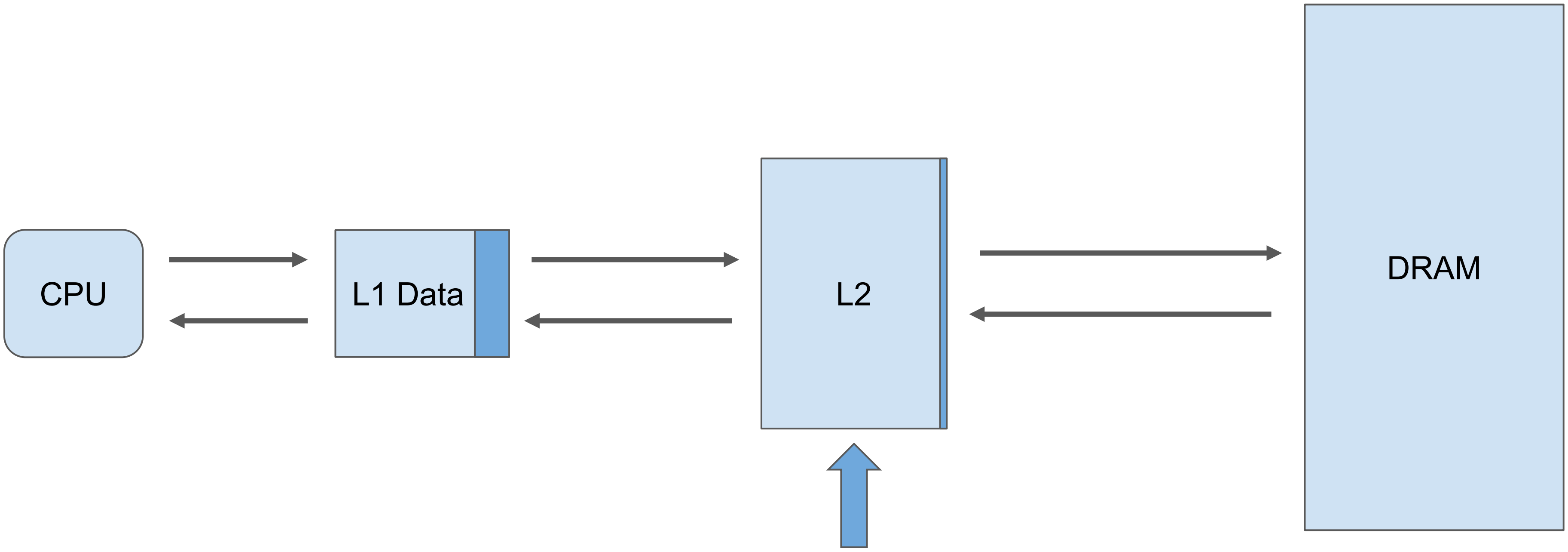



*BitVector Califorms*

No Latency Overhead

12.5% memory overhead

# CALIFORMS: MICROARCHITECTURE

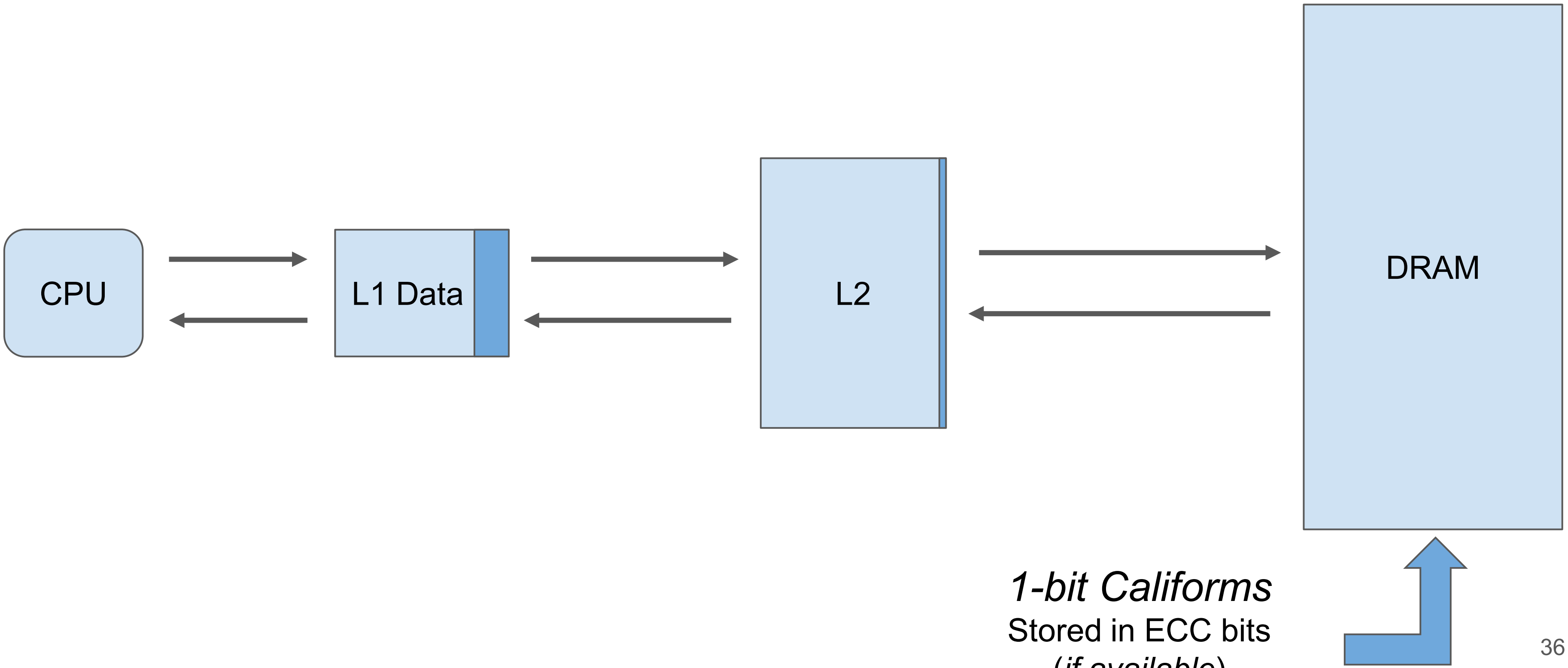



*1-bit Califorms*

1 Cycle *or can be hidden*

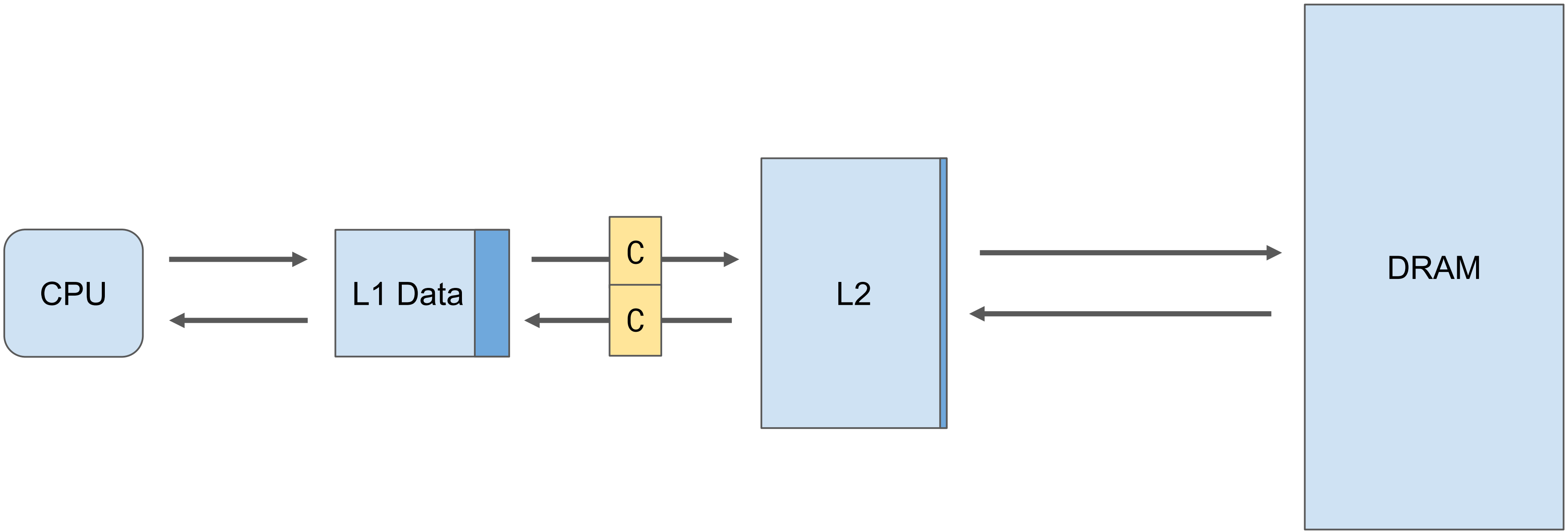
0.2% memory overhead

# CALIFORMS: MICROARCHITECTURE

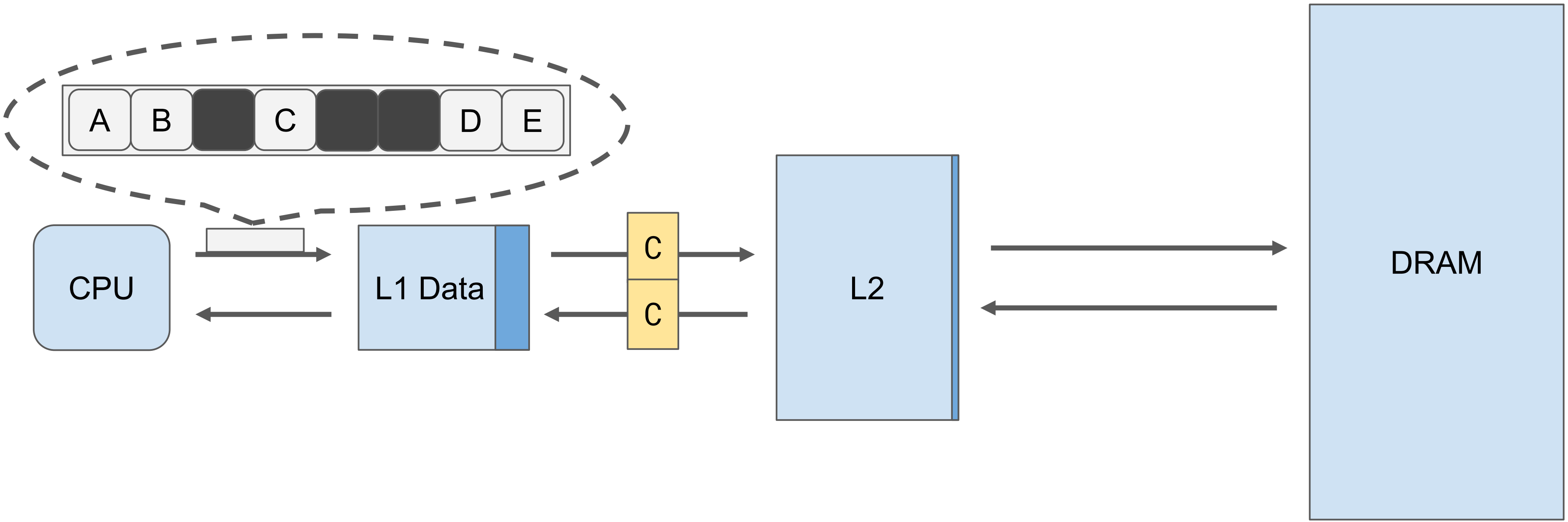



*1-bit Califorms*
Stored in ECC bits
(*if available*)

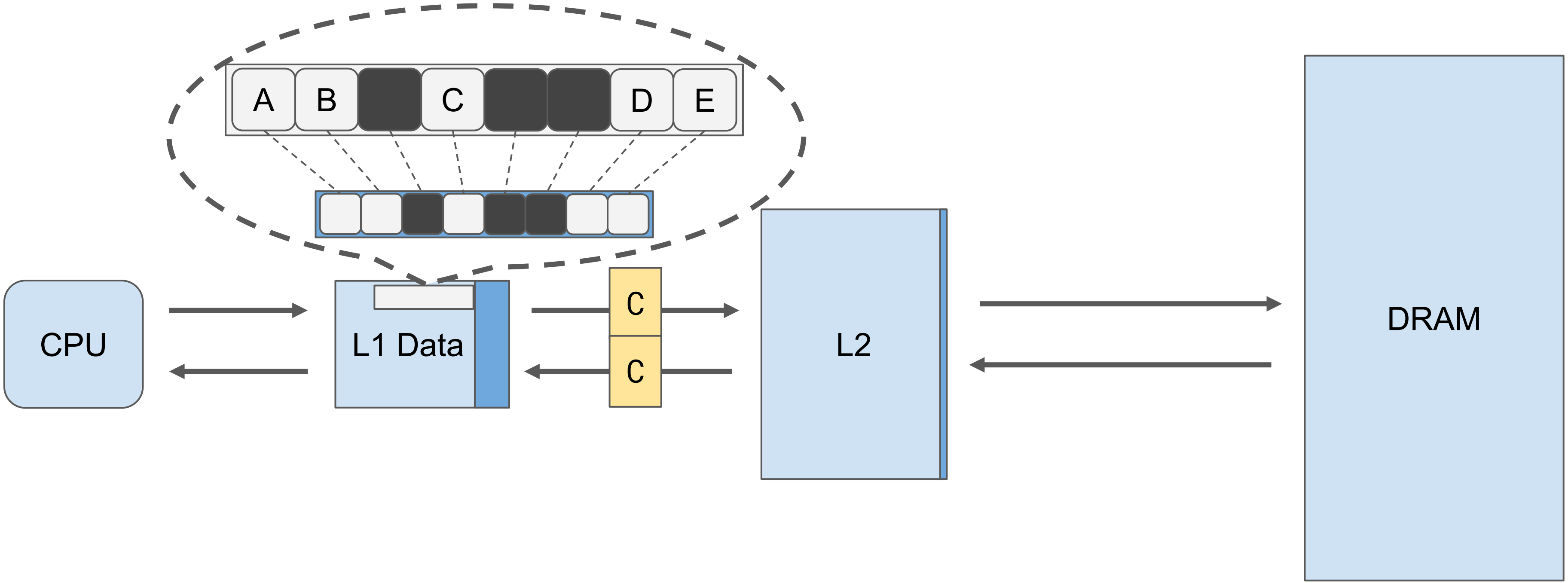
# CALIFORMS: MICROARCHITECTURE



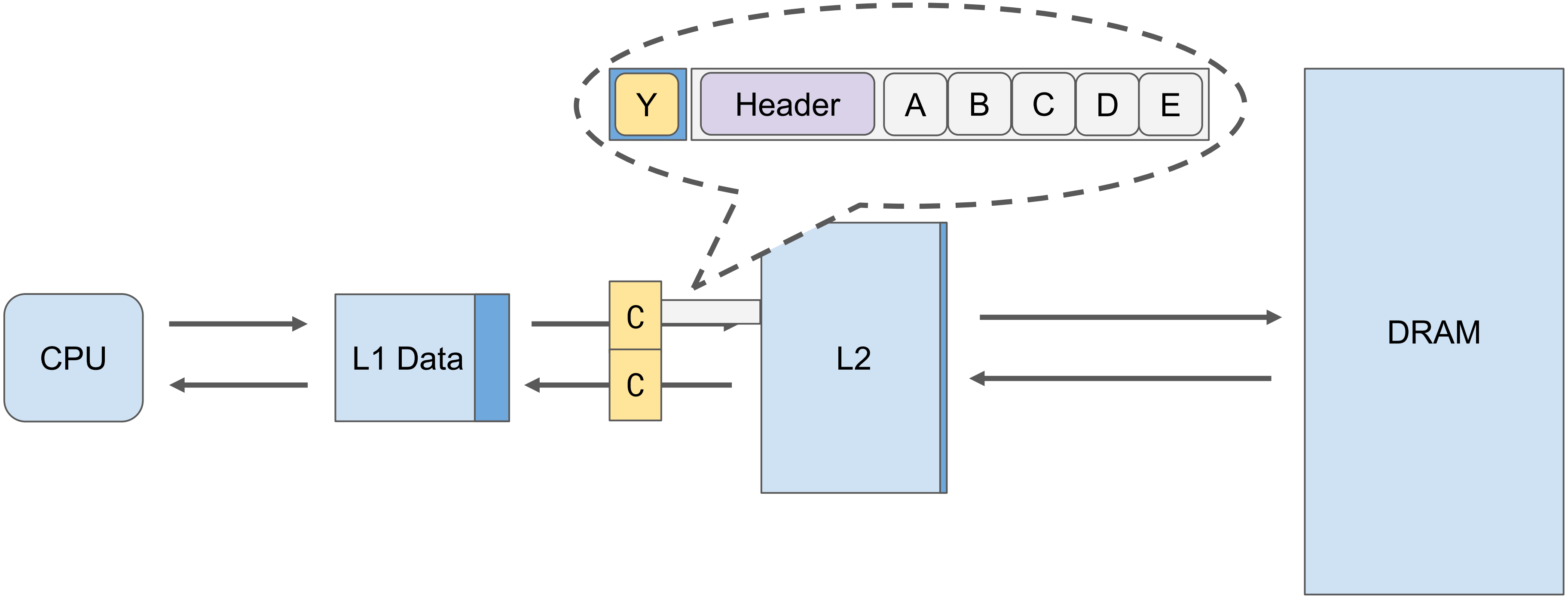
# CALIFORMS: MICROARCHITECTURE



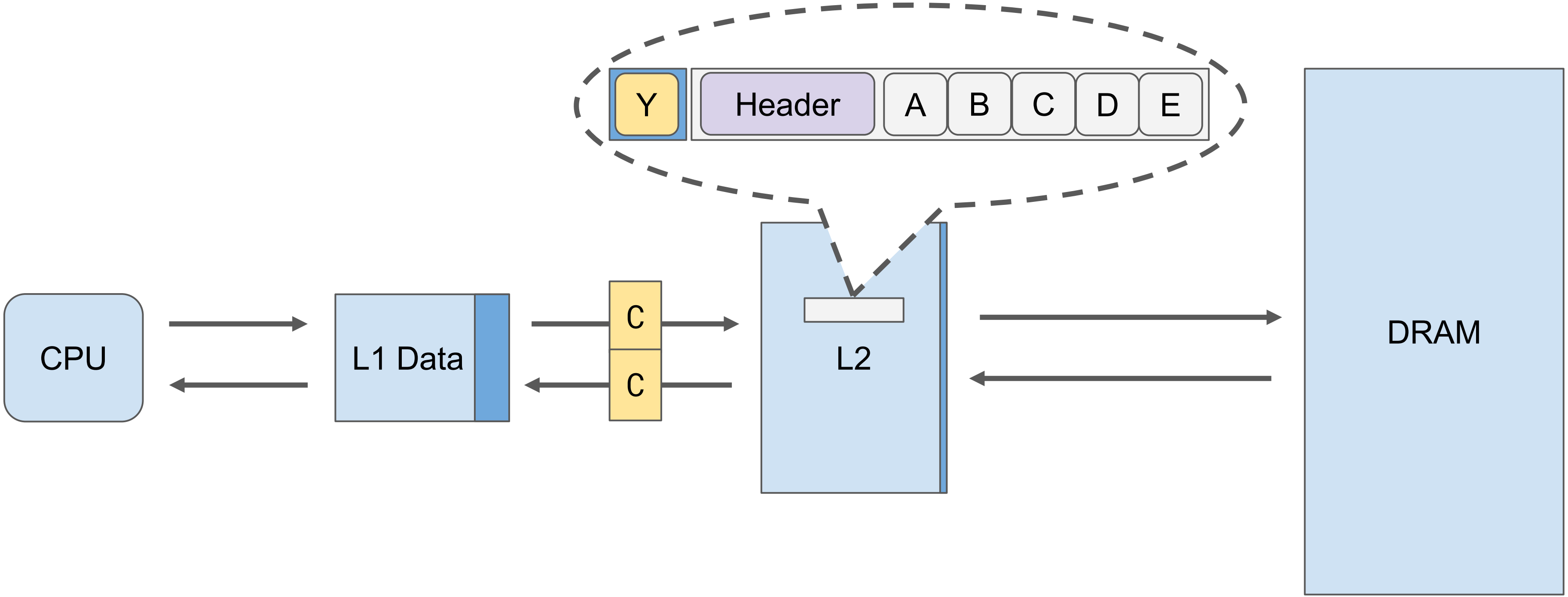
# CALIFORMS: MICROARCHITECTURE



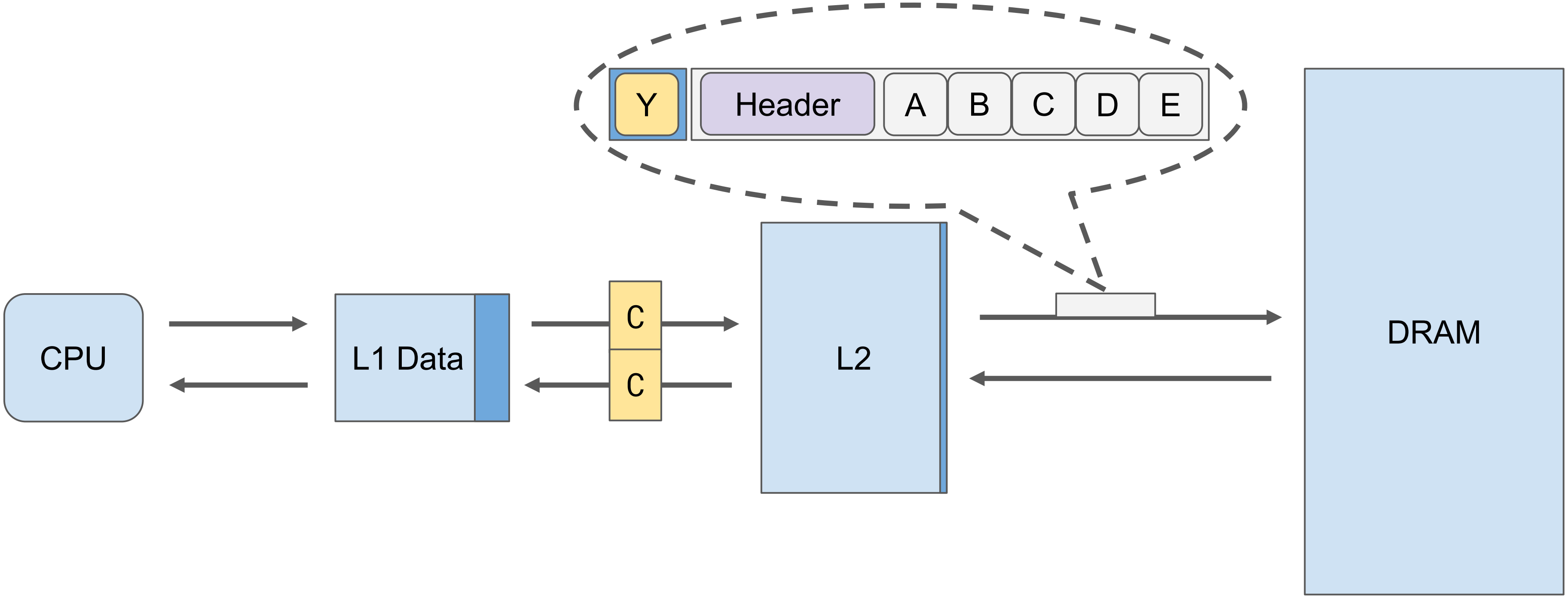
# CALIFORMS: MICROARCHITECTURE



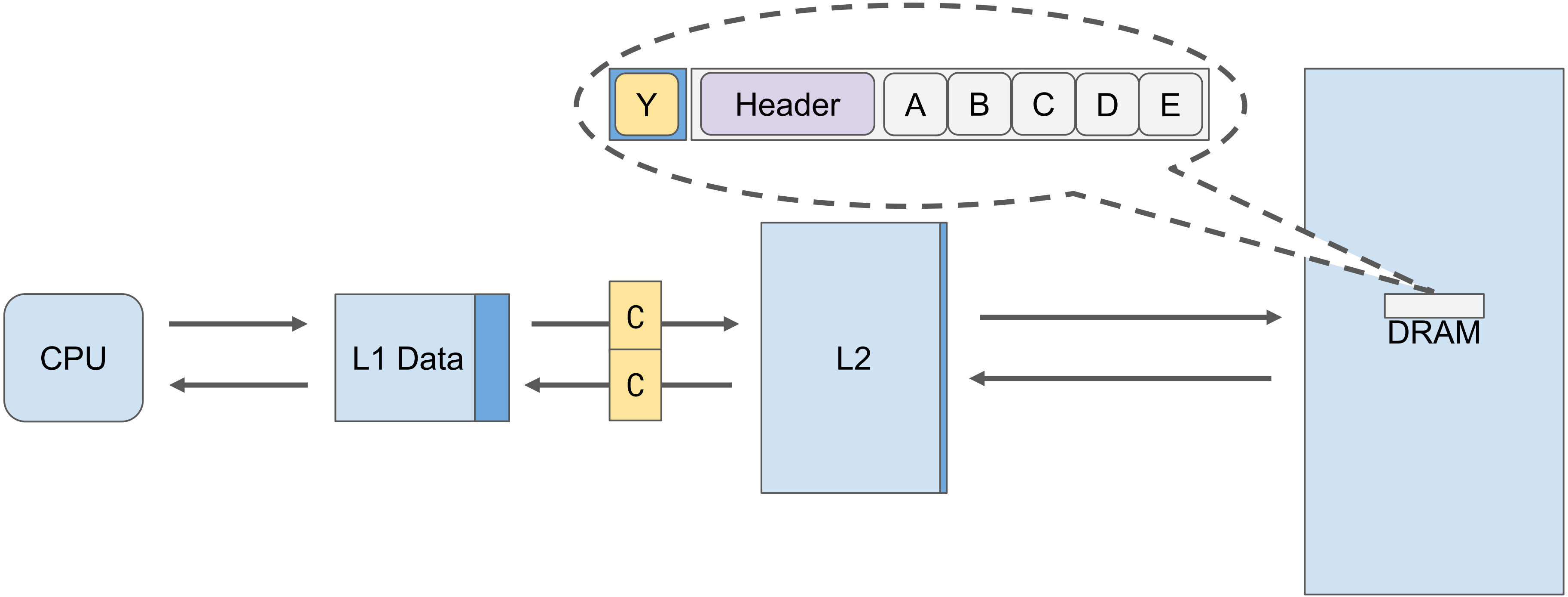
# CALIFORMS: MICROARCHITECTURE



# CALIFORMS: MICROARCHITECTURE



# CALIFORMS: MICROARCHITECTURE



# CALIFORMS: FULL SYSTEM

- **Microarchitecture**
  - Cache controller.
  - L1/L2 Califorms converters.
- **Architecture Support**
- **Software**

# CALIFORMS: FULL SYSTEM

- **Microarchitecture**
  - Cache controller.
  - L1/L2 Califorms converters.
- **Architecture Support**
  - A new **Blacklisting** instruction.
- **Software**
  - Compiler, memory allocator and OS extensions.

# CALIFORMS: FULL SYSTEM

For more details, please refer to our paper.

- **Microarchitecture**
  - Cache controller.
  - L1/L2 Califorms converters.
- **Architecture Support**
  - A new **Blacklisting** instruction.
- **Software**
  - Compiler, memory allocator and OS extensions.

# CALIFORMS: SUMMARY

- **Has no false positives**
  - Precise storage (0-64 blacklisted locations per cache line).

# CALIFORMS: SUMMARY

- **Has no false positives**
  - Precise storage (0-64 blacklisted locations per cache line).
- **Supports existing performance optimizations**
  - Critical word first.

# CALIFORMS: SUMMARY

- **Has no false positives**
  - Precise storage (0-64 blacklisted locations per cache line).
- **Supports existing performance optimizations**
  - Critical word first.
- **Integrates into existing microarchitectures**
  - Does NOT disturb coherency.

# CALIFORMS

## PERFORMANCE

# CALIFORMS: PERFORMANCE OVERHEADS

- **Hardware Overheads**

- **Blacklisting Overheads**

# CALIFORMS: PERFORMANCE OVERHEADS

- **Hardware Overheads**

- **Blacklisting Overheads**

# CALIFORMS: PERFORMANCE OVERHEADS

- **Hardware Overheads**
- **Blacklisting Overheads**

# CALIFORMS: INSERTION POLICIES

```
struct A_opportunistic
{
  char c;
  char tripwire[3];
  int i;
  char buf[64];
  void (*fp)();
}
```

**(1) Opportunistic**

***Tripwire Insertion Policies***

# CALIFORMS: INSERTION POLICIES

```
struct A_opportunistic
{
  char c;
  char tripwire[3];
  int i;
  char buf[64];
  void (*fp)();
}
```

**(1) Opportunistic**

```
struct A_full {
  char tripwire[2];
  char c;
  char tripwire[1];
  int i;
  char tripwire[3];
  char buf[64];
  char tripwire[2];
  void (*fp)();
  char tripwire[1];
}
```

**(2) Full**

***Tripwire Insertion Policies***

# CALIFORMS: INSERTION POLICIES

```
struct A_opportunistic
{
  char c;
  char tripwire[3];
  int i;
  char buf[64];
  void (*fp)();
}
```

**(1) Opportunistic**

```
struct A_full {
  char tripwire[2];
  char c;
  char tripwire[1];
  int i;
  char tripwire[3];
  char buf[64];
  char tripwire[2];
  void (*fp)();
  char tripwire[1];
}
```

**(2) Full**

```
struct A_intelligent {
  char c;
  int i;
  char tripwire[3];
  char buf[64];
  char tripwire[2];
  void (*fp)();
  char tripwire[3];
}
```

**(3) Intelligent**

***Tripwire Insertion Policies***

# CALIFORMS

## EVALUATION METHODOLOGY

# CALIFORMS: EVALUATION METHODOLOGY

- **Emulating the Blacklisting instruction**
  - Inserting dummy stores to blacklisted bytes.

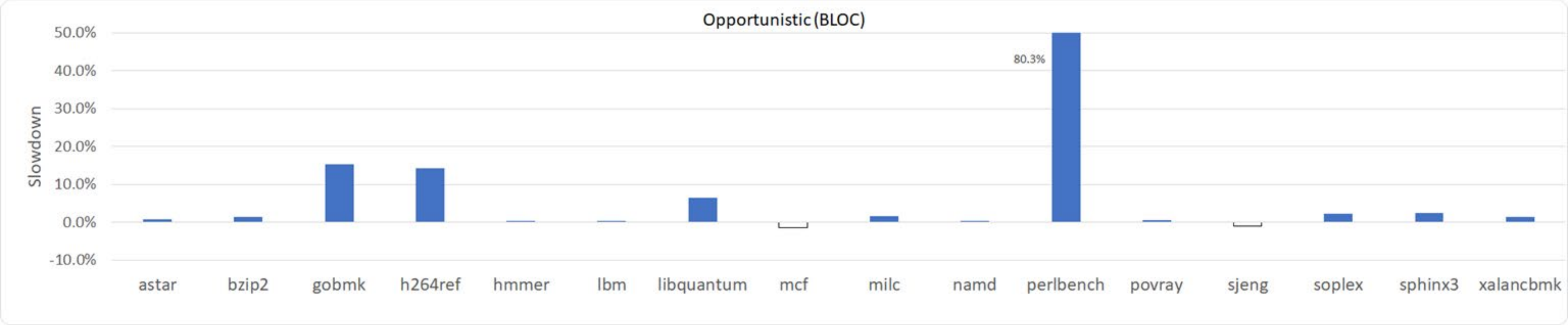
# CALIFORMS: EVALUATION METHODOLOGY

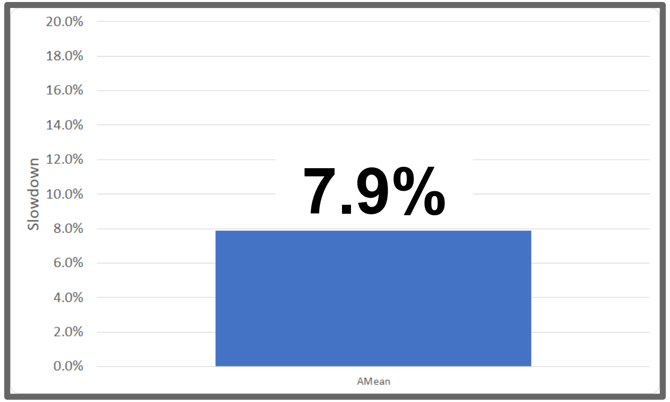
- **Emulating the Blacklisting instruction**
  - Inserting dummy stores to blacklisted bytes.
- **NO simulations**
  - Taking results from a real Skylake-based machine.

# CALIFORMS: EVALUATION METHODOLOGY

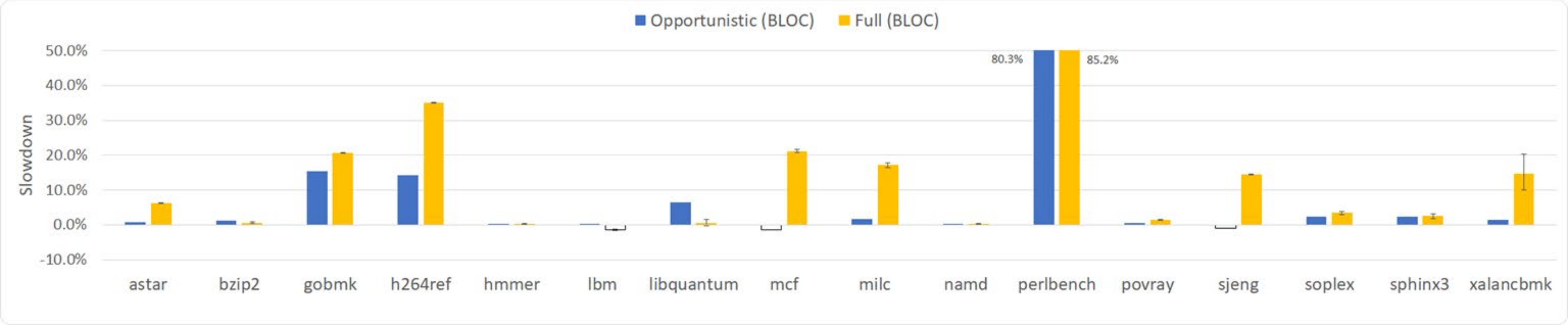
- **Emulating the Blacklisting instruction**
  - Inserting dummy stores to blacklisted bytes.
- **NO simulations**
  - Taking results from a real Skylake-based machine.
- **Using SPEC2006 benchmarks with reference inputs**
  - Running experiments to completion.

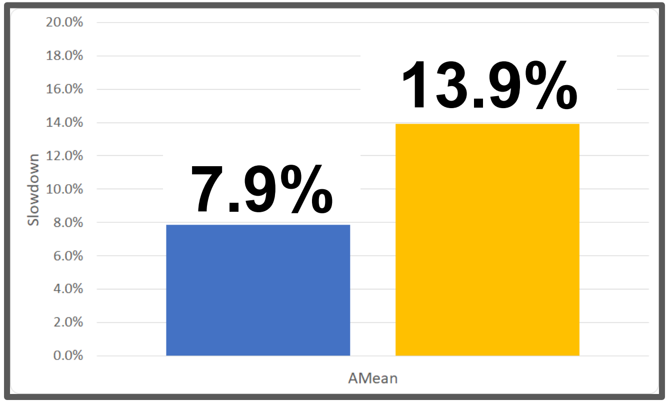
# CALIFORMS: POLICIES OVERHEADS



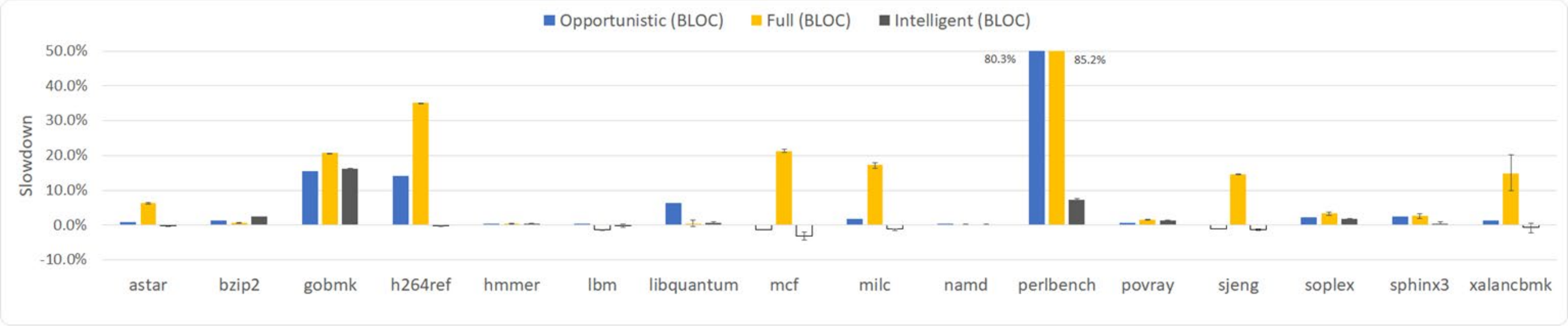



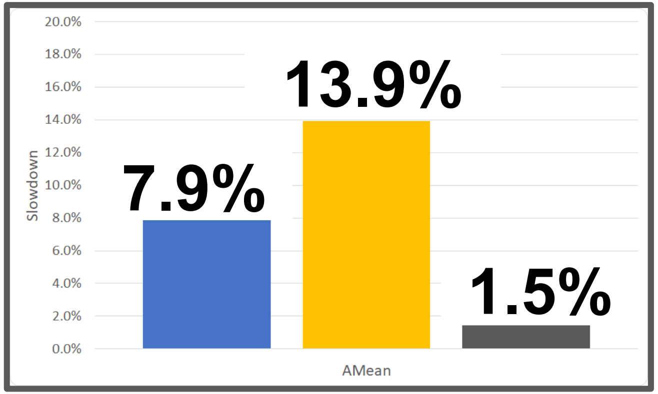
# CALIFORMS: POLICIES OVERHEADS



# CALIFORMS: POLICIES OVERHEADS



# CALIFORMS: POLICIES OVERHEADS

```
struct A_opportunistic
{
  char c;
  char tripwire[3];
  int i;
  char buf[64];
  void (*fp)();
}
```

**(1) Opportunistic**

Provides the best **performance-security tradeoff.**

**(2) Full**

```
struct A_intelligent {
  char c;
  int i;
  char tripwire[3];
  char buf[64];
  char tripwire[2];
  void (*fp)();
  char tripwire[3];
}
```

**(3) Intelligent**

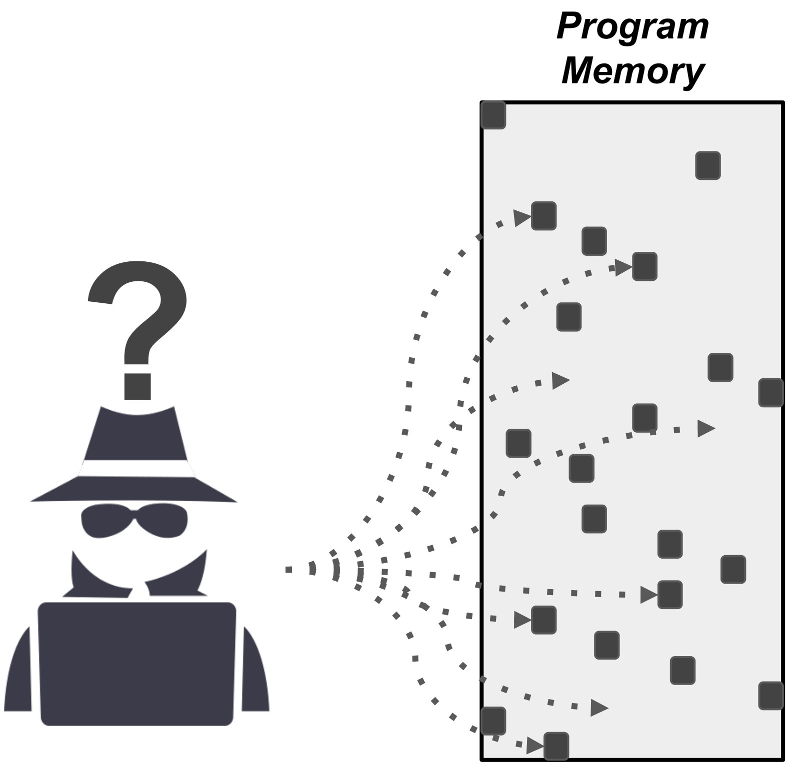
***Tripwire Insertion Policies***

# CALIFORMS

## SECURITY BENEFITS

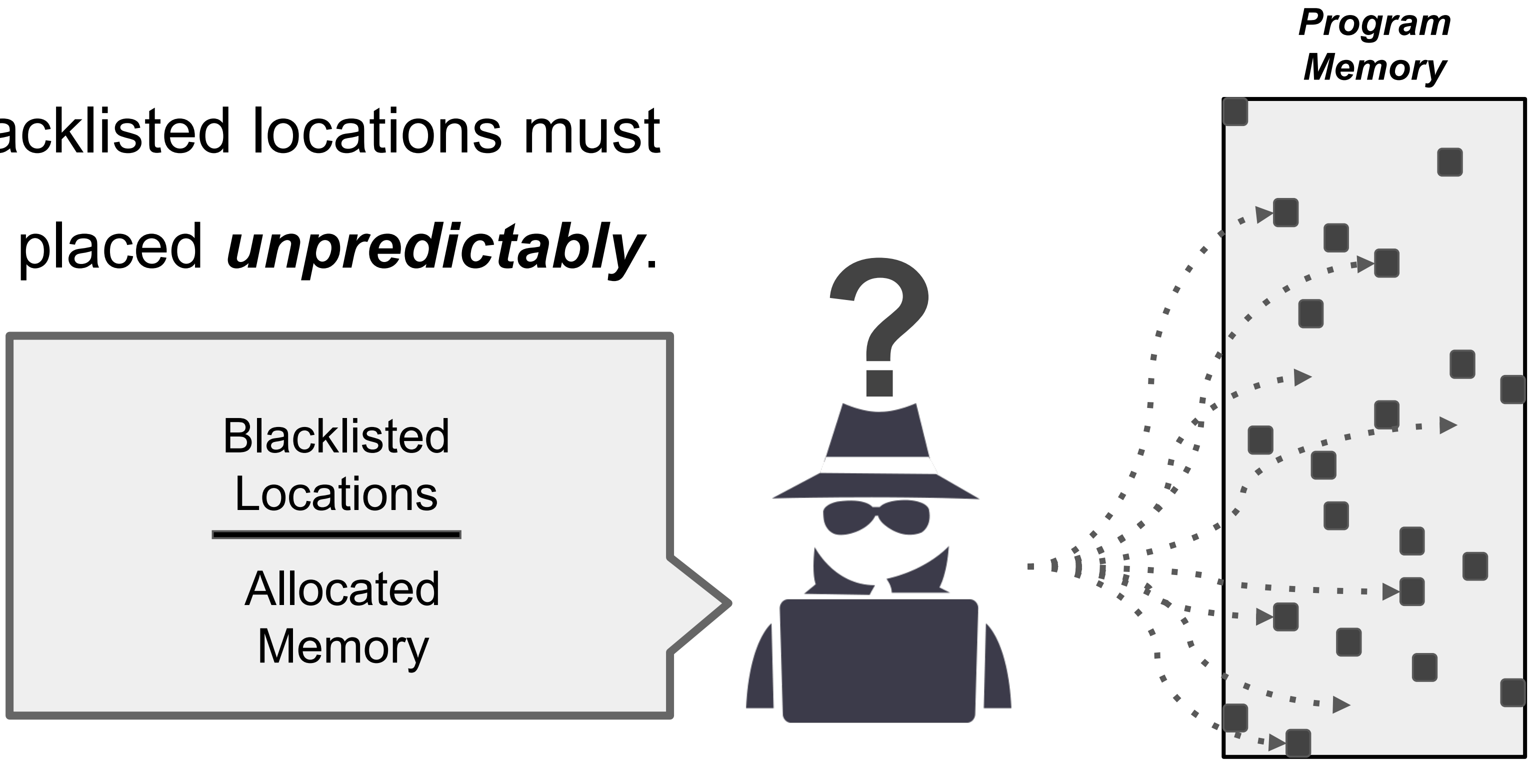
# CALIFORMS: SECURITY BENEFITS

- Blacklisted locations must be placed ***unpredictably***.



# CALIFORMS: SECURITY BENEFITS

- Blacklisted locations must be placed ***unpredictably***.



# CALIFORMS: SECURITY BENEFITS

This is the **best case** for me.
Only one object!

We insert **up to 7** tripwires between fields

| | |
|---|---|
| 0 | `int pin;` |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |

# CALIFORMS: SECURITY BENEFITS

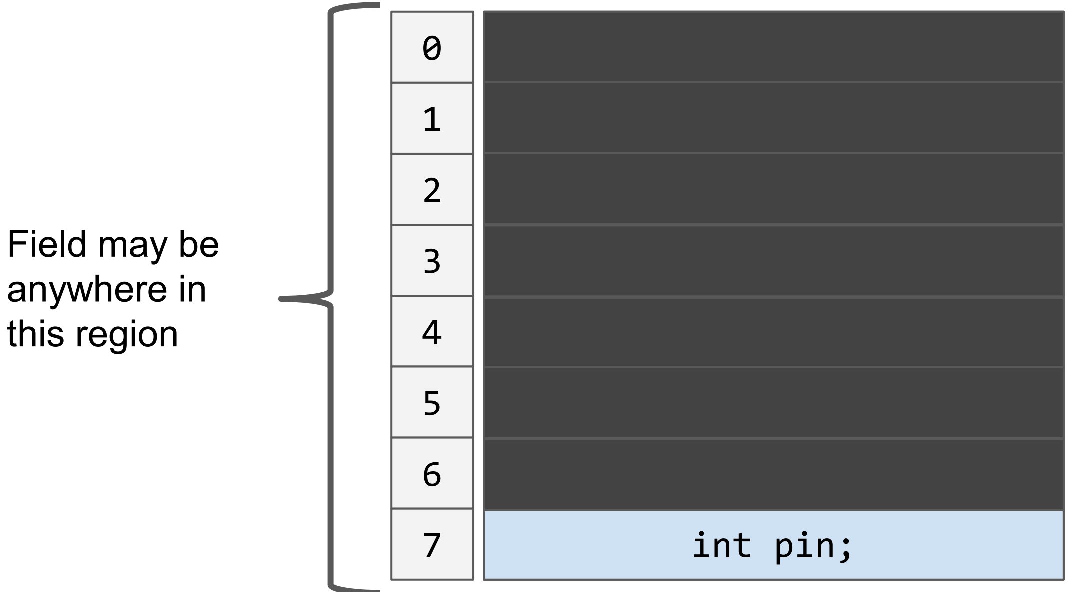
Let me **guess** where the field is.

Field may be anywhere in this region

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | `int pin;` |
| 5 | |
| 6 | |
| 7 | |

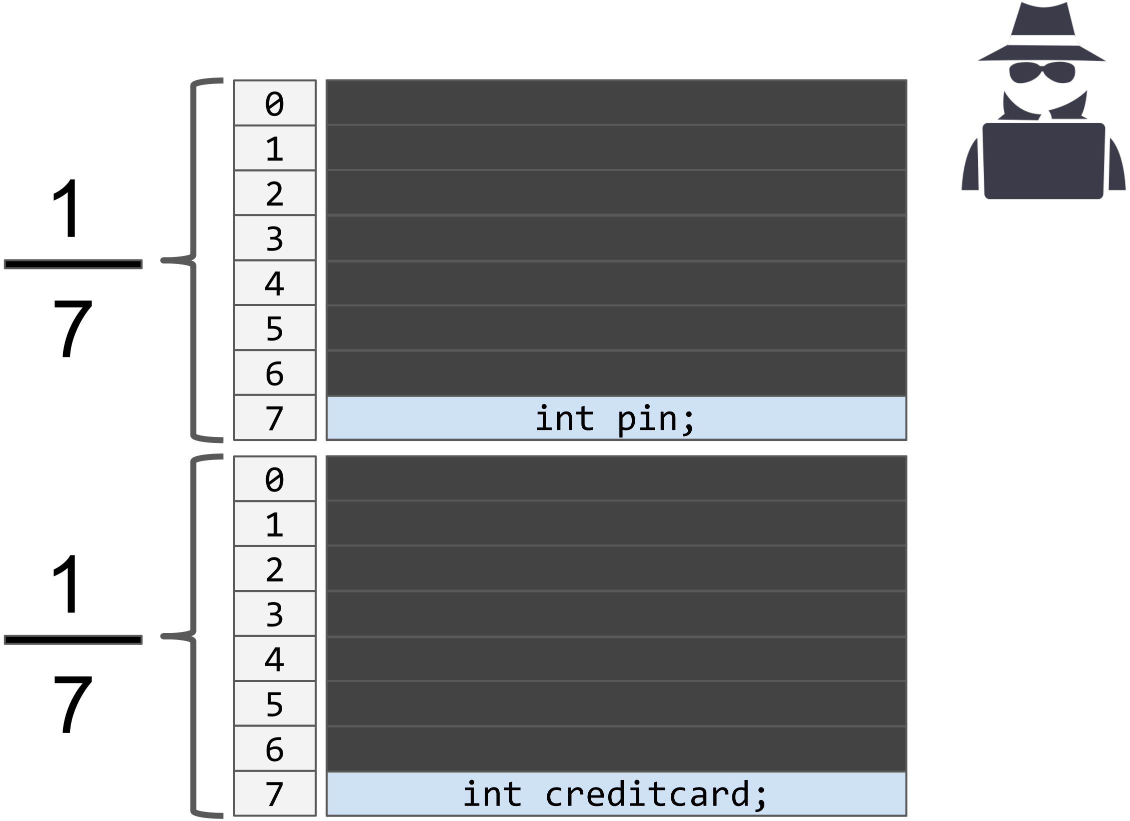
# CALIFORMS: SECURITY BENEFITS

Let me **guess** where the field is.



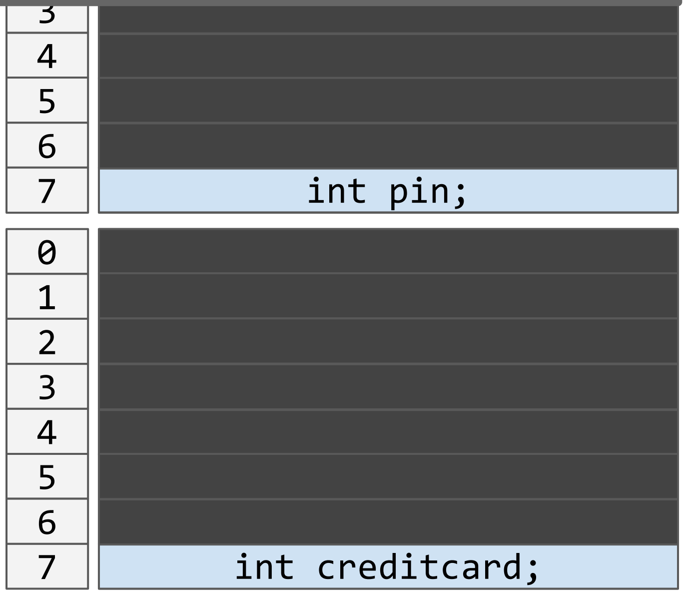

$$\frac{1}{7}$$

# CALIFORMS: SECURITY BENEFITS



# CALIFORMS: SECURITY BENEFITS

The more I need to disclose the **harder** it is!




$$\frac{1}{7^n}$$

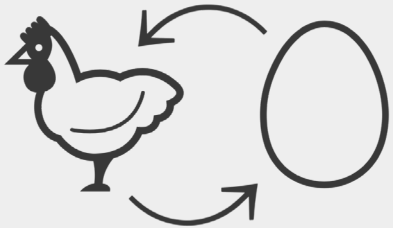
where $n$ is the number of fields to be disclosed.

# CALIFORMS: SECURITY BENEFITS

Can I **disable** blacklisted memory?

# CALIFORMS: SECURITY BENEFITS

Can I **disable** blacklisted memory?

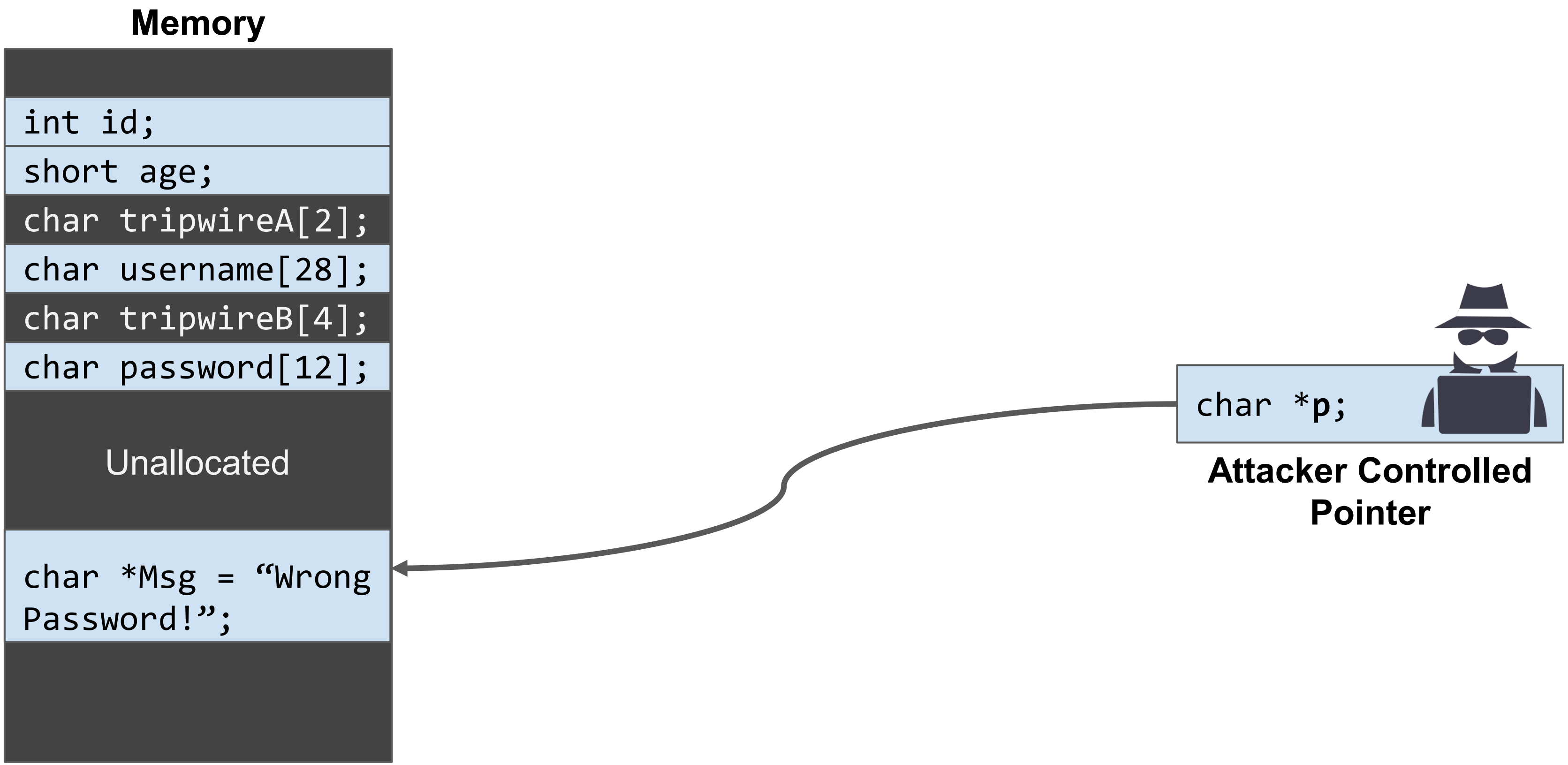


They would first need to bypass Califorms.

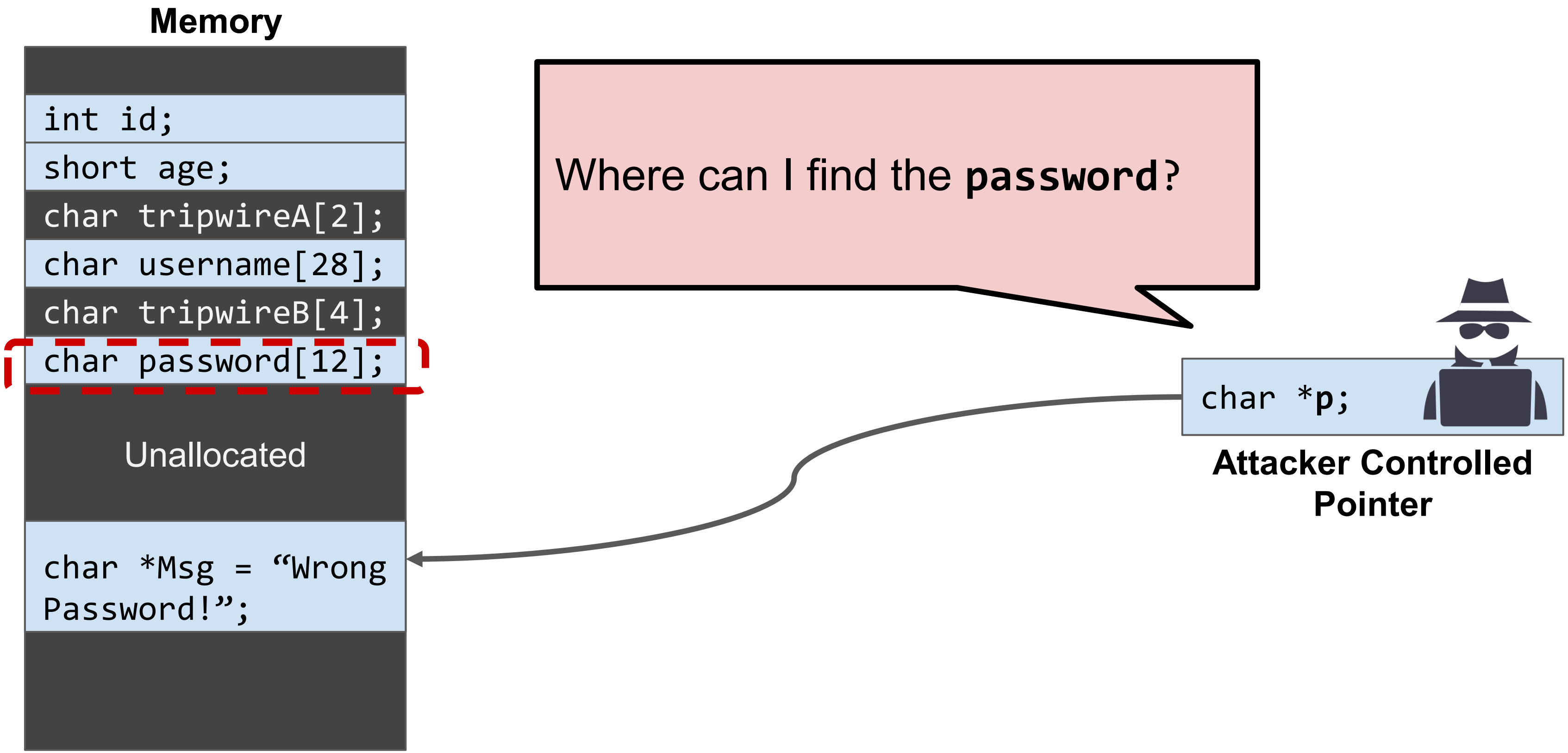
# MEMORY SCANNING ATTACK

**WITH** CALIFORMS

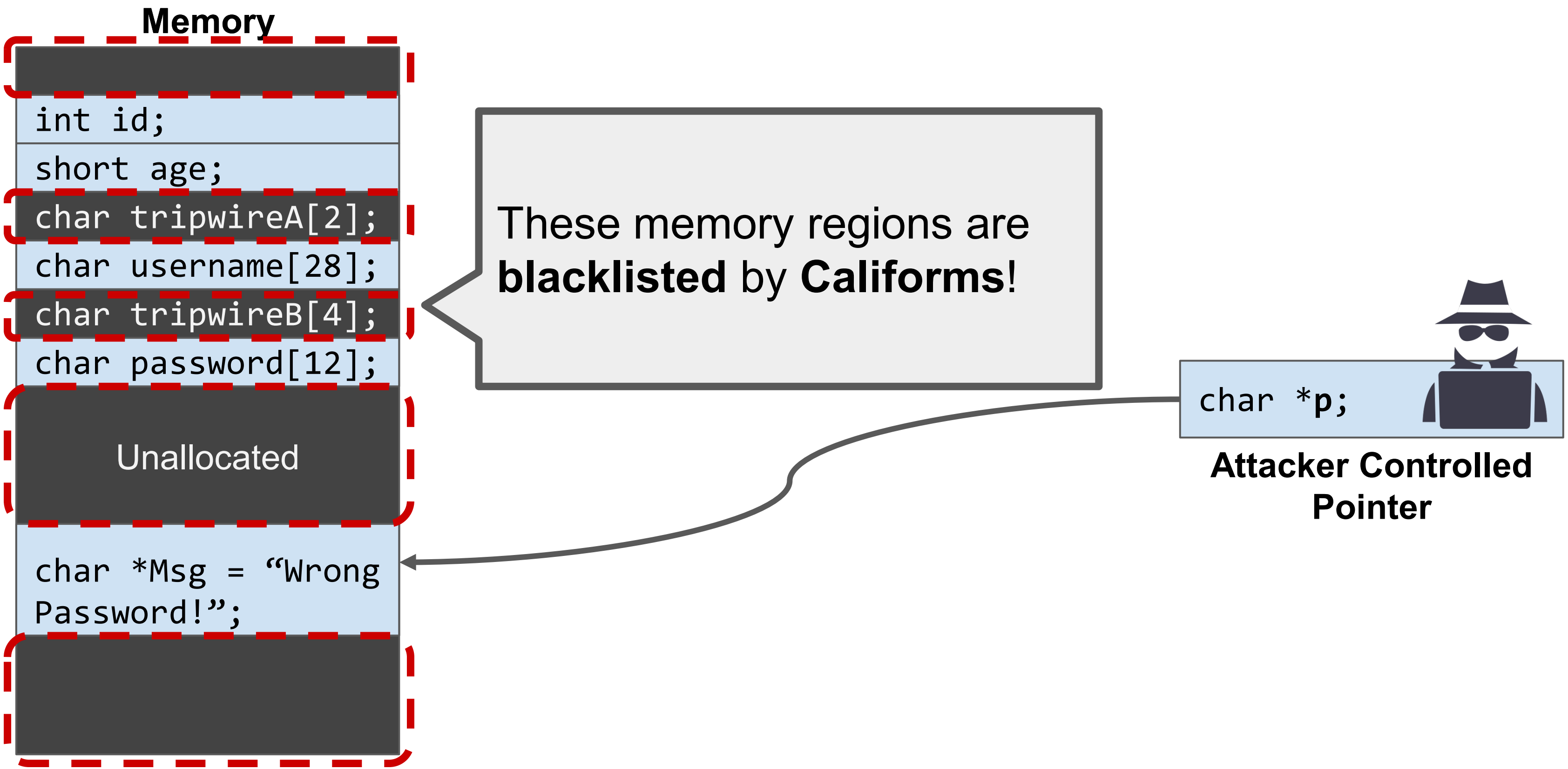
# EXAMPLE: MEMORY SCANNING ATTACK



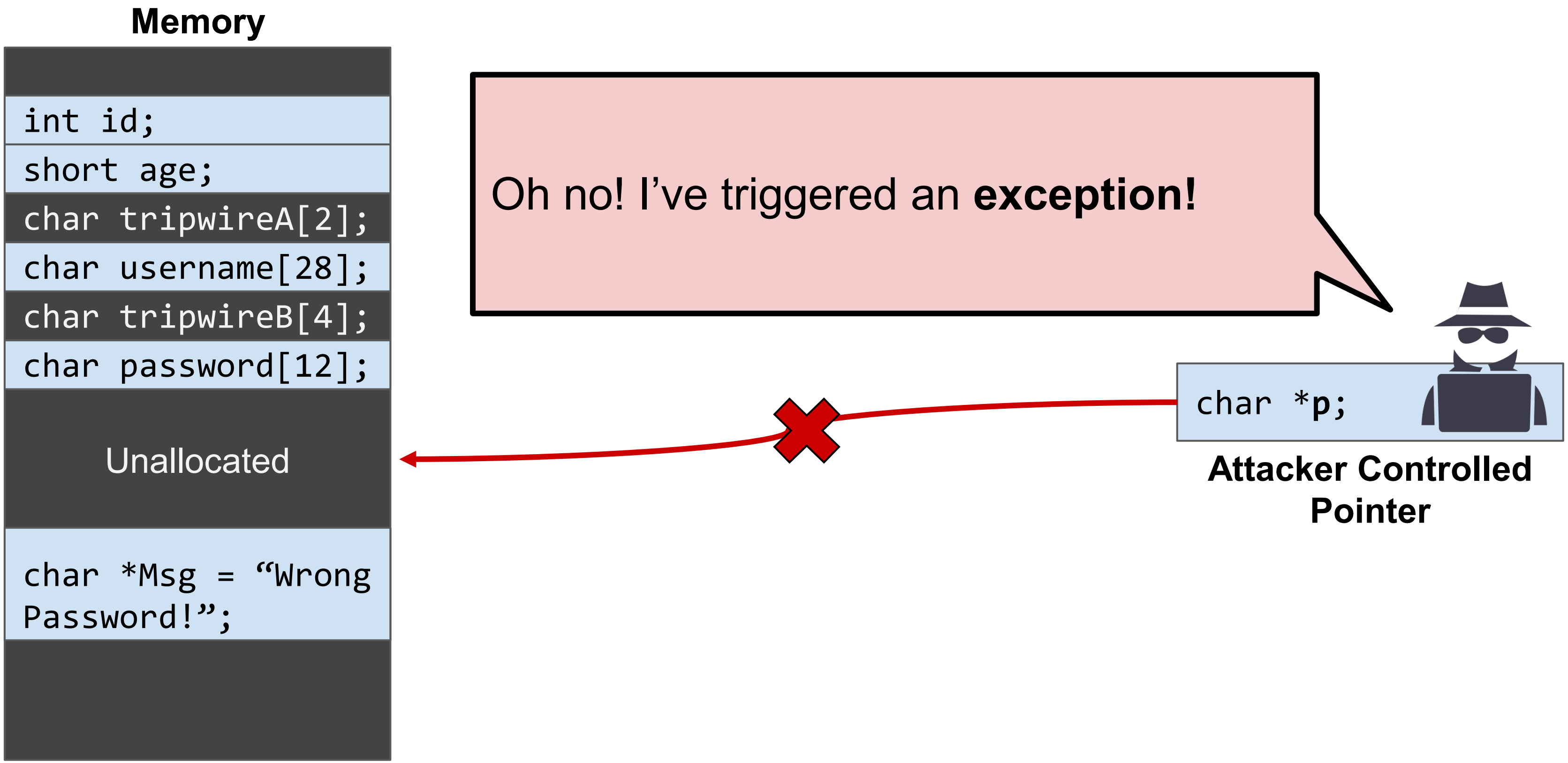
# EXAMPLE: MEMORY SCANNING ATTACK



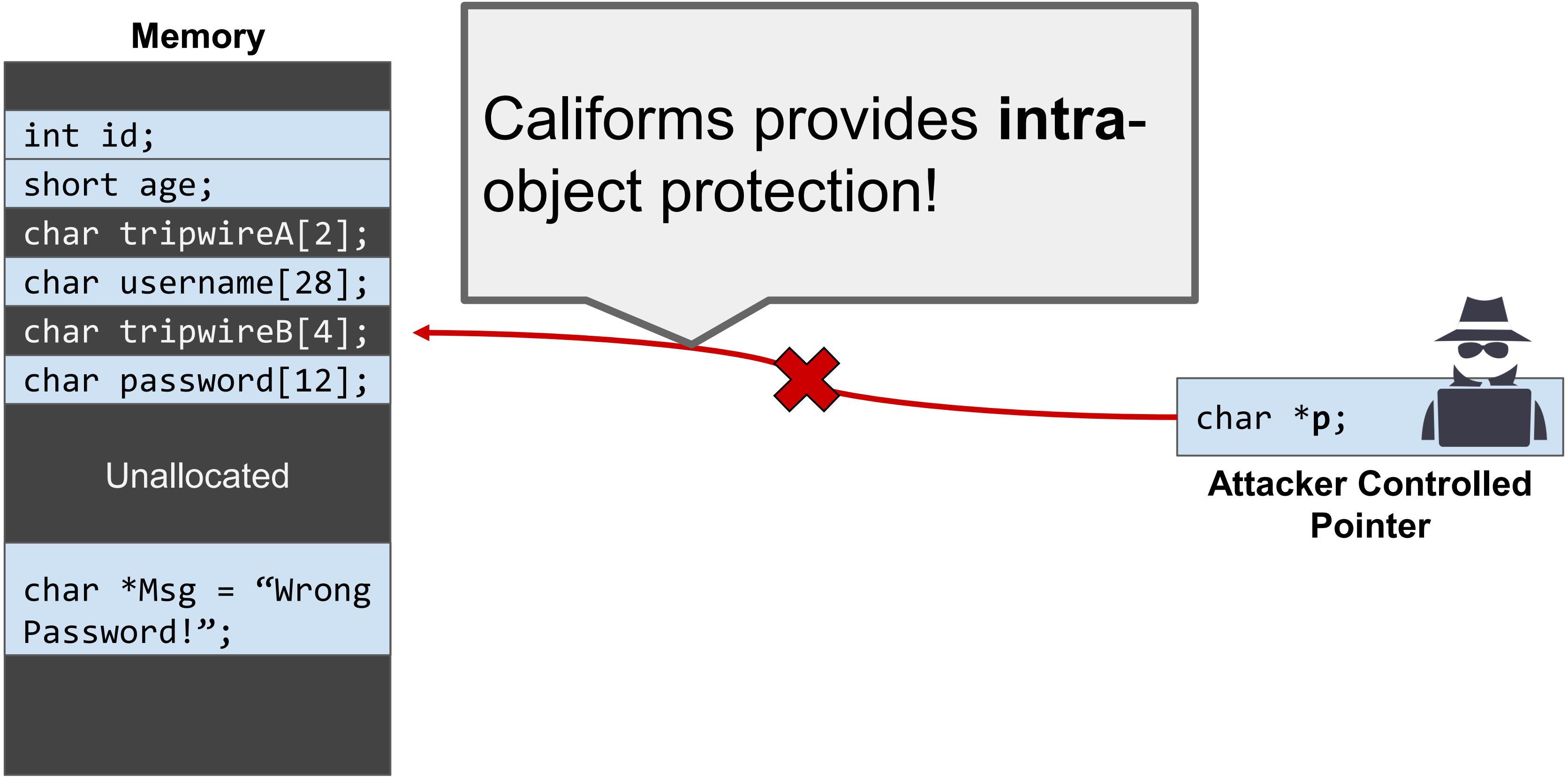
# EXAMPLE: MEMORY SCANNING ATTACK



# EXAMPLE: MEMORY SCANNING ATTACK



# EXAMPLE: MEMORY SCANNING ATTACK



# RELATED WORK

# RELATED WORK

| Technique | Program Memory Footprint | Performance Overhead |
| --- | --- | --- |
| Base & Bound | $\propto$ # of pointers | $\propto$ # of pointer dereferences |

# RELATED WORK

| Technique | Program Memory Footprint | Performance Overhead |
|---|---|---|
| Base & Bound | $\propto$ # of pointers | $\propto$ # of pointer dereferences |
| FAT Pointers | $\propto$ # of pointers and physical mem. | $\propto$ # of pointer operations |

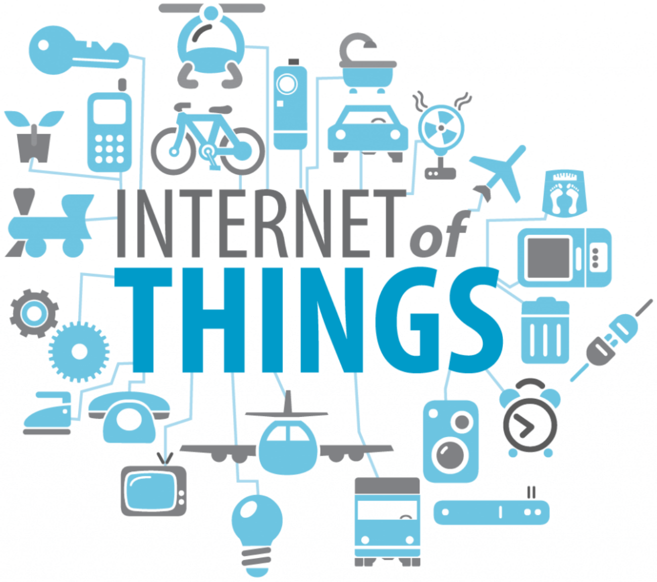
# RELATED WORK

| Technique | Program Memory Footprint | Performance Overhead |
|---|---|---|
| Base & Bound | $\propto$ # of pointers | $\propto$ # of pointer dereferences |
| FAT Pointers | $\propto$ # of pointers and physical mem. | $\propto$ # of pointer operations |
| Califorms | $\propto$ # blacklisted locations | $\propto$ # of blacklisting instructions |

# CONCLUSION

- Californs can be applied to **non 64-bit systems** (e.g IoT, CPS, etc).

- Califorms' blacklisting is an efficient solution to memory safety:
  - Is **easy to implement**.
  - Has **low overheads**.
  - Offers **robust security**.



# CONCLUSION

- Califorms can be applied to **non 64-bit systems** (e.g IoT, CPS, etc).

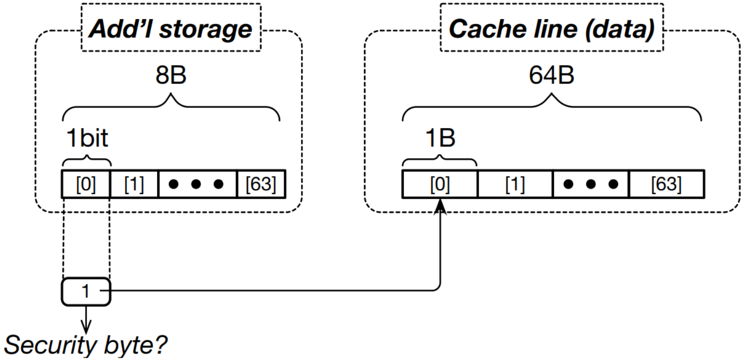
- Califorms' blacklisting is an efficient solution to memory safety:
  - Is **easy to implement.**
  - Has **low overheads**.
  - Offers **robust security**.

QUESTIONS?

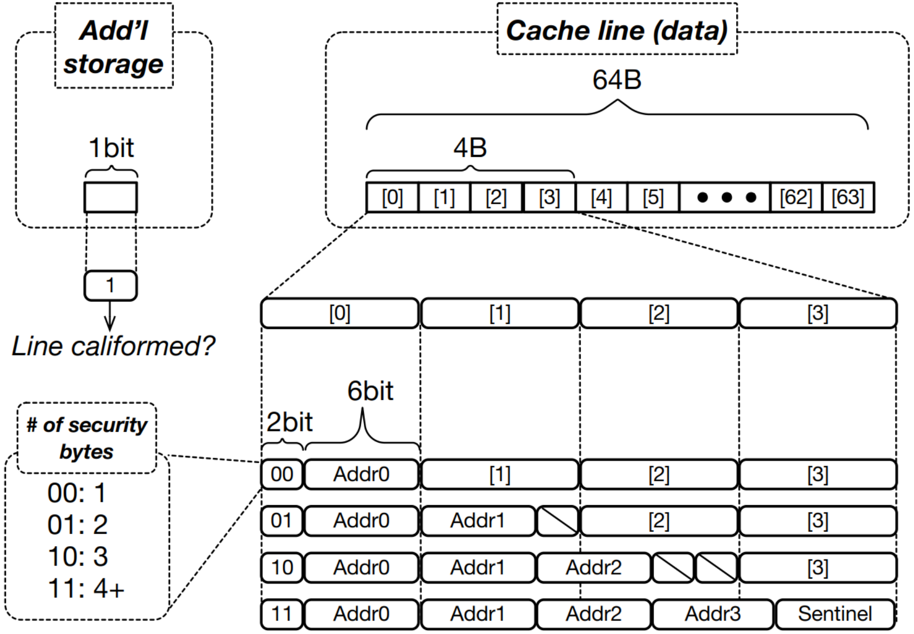
Stop by during the poster session to chat!
2:50-4:00pm

# BACKUPS

# CALIFORMS: ENCODING SCHEMES

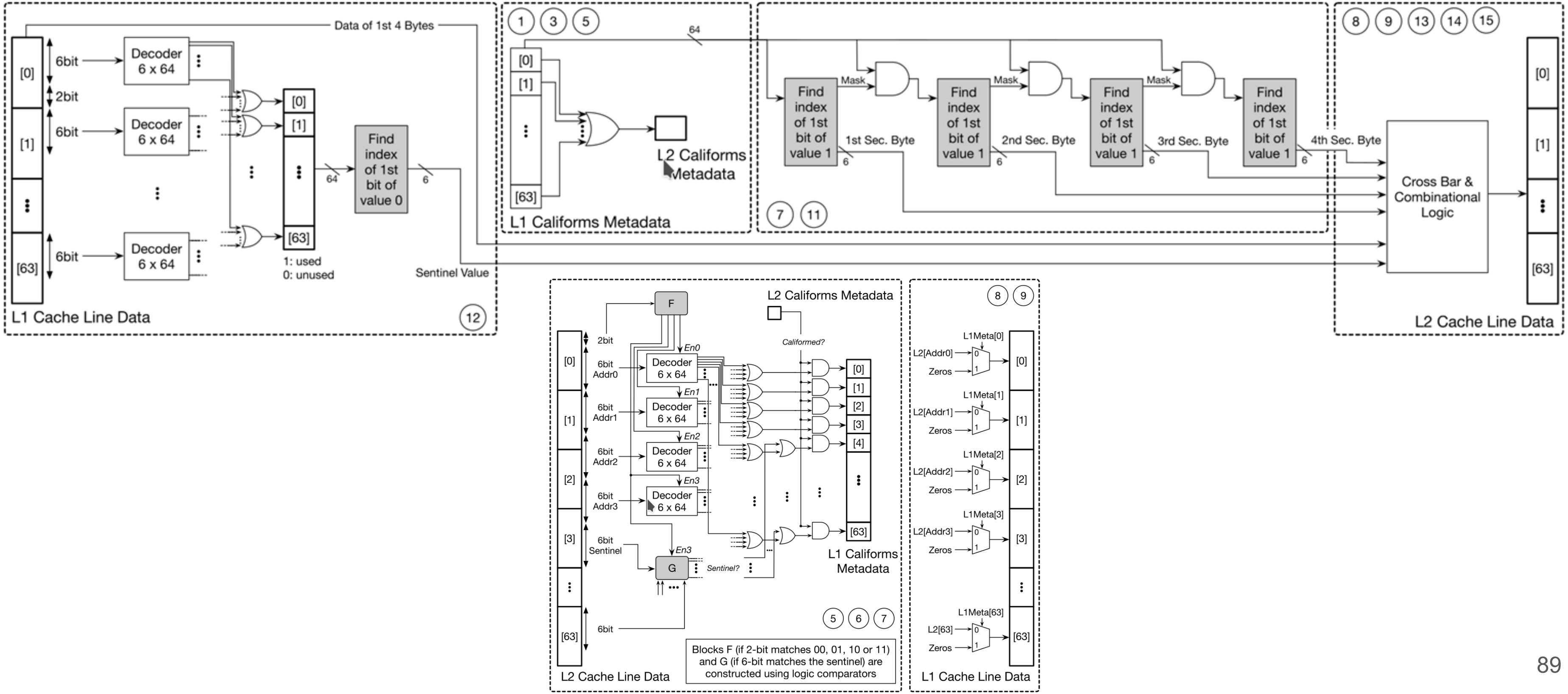



**califorms-bitvector**

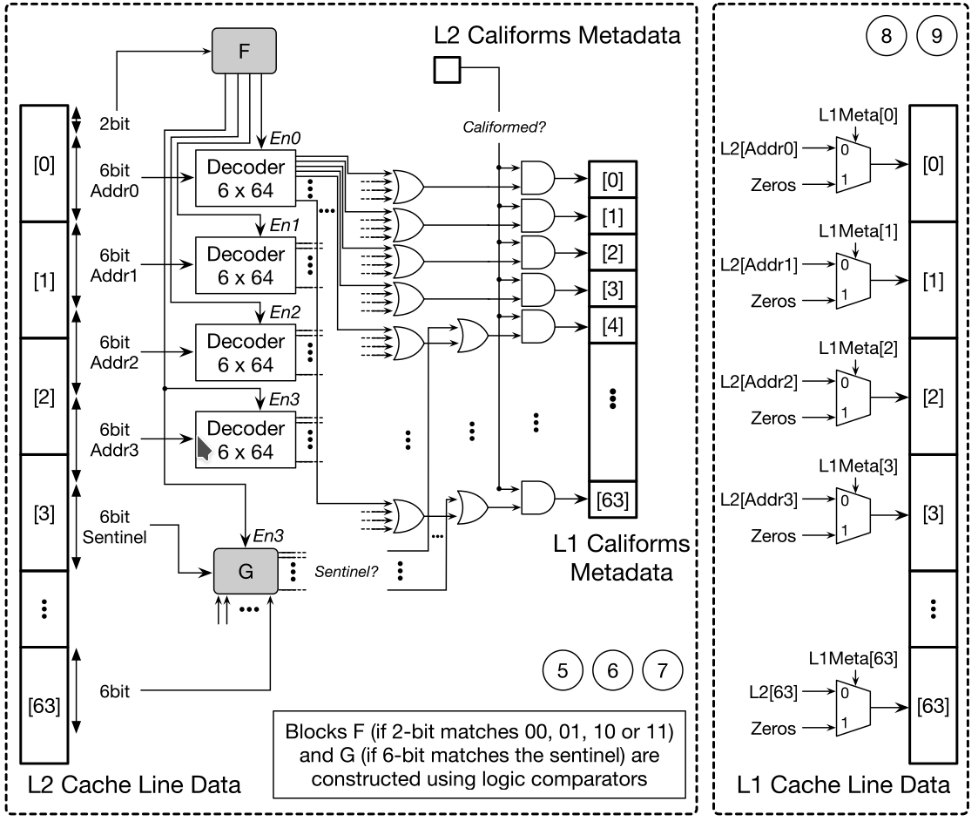



**califorms-sentinel**

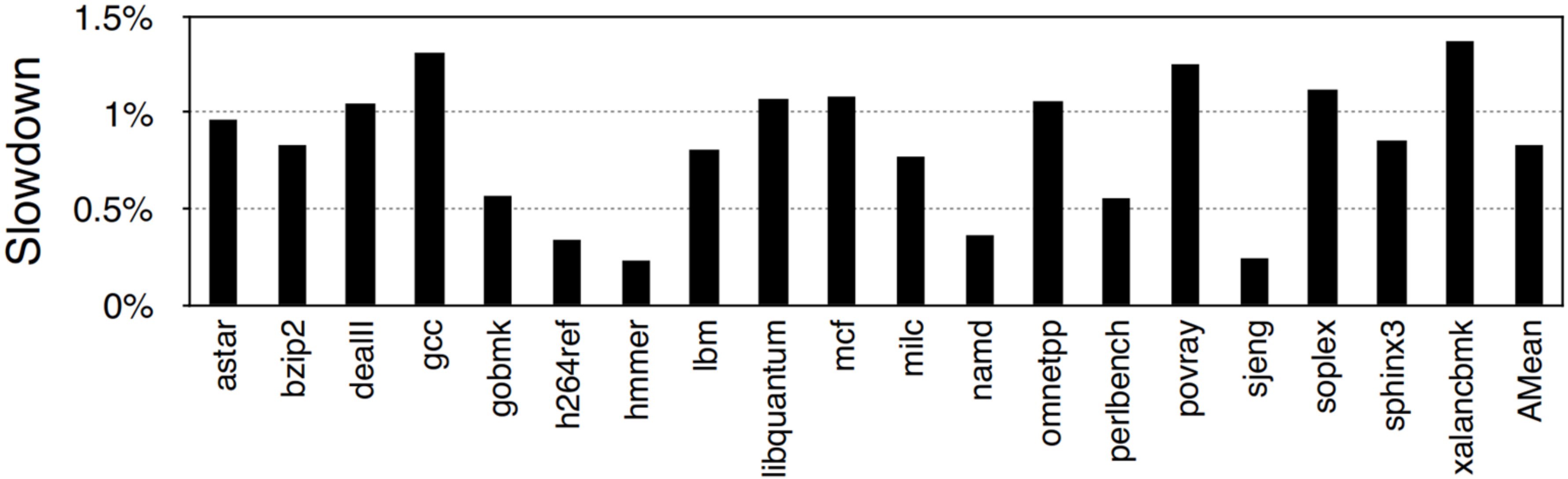
# CALIFORMS: HARDWARE DIAGRAM



# CALIFORMS: HARDWARE DIAGRAM



# CALIFORMS: CONSERVATIVE ANALYSIS



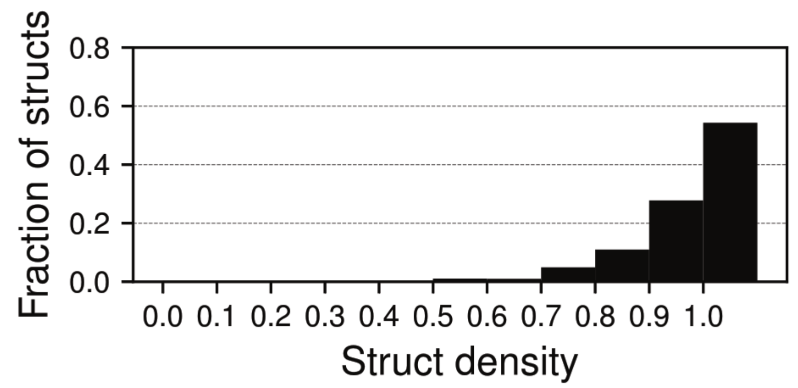

*Slowdown with additional one-cycle access latency for both L2 and L3 caches.*

# CALIFORMS: HARDWARE PERFORMANCE

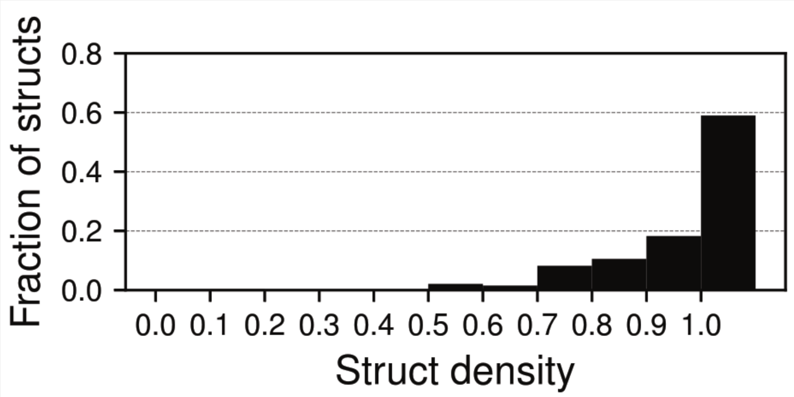
| **L1 Califorms** | **Area (GE)** | **Delay (*ns*)** | **Power (*mW*)** |
|---|---|---|---|
| L1 Overheads | [+18.69%] 412,263.87 | [+1.85%] 1.65 | [+2.12%] 16.17 |
| Fill Module | 8,957.16 | 1.43 | 0.18 |
| Spill Module | 34,561.80 | 5.50 | 0.52 |

# CALIFORMS: OPPORTUNISTIC POLICY

**Normally Occurring Dead Bytes**




*SPEC CPU2006 C and C++ Benchmarks*




*V8 JavaScript Engine*

**Struct density** = $\sum_i^{\#fields}(\texttt{sizeof}(\texttt{field}_i))/\texttt{sizeof}(\texttt{struct})$