Morphable Counters: Enabling Compact Integrity Trees for Low-Overhead Secure Memories

MICRO-2018

Gururaj Saileshwar[1]

Prashant Nair[1] Prakash Ramrakhyani[2] Wendy Elsasser[2]

Jose Joao[2] Moinuddin Qureshi[1]

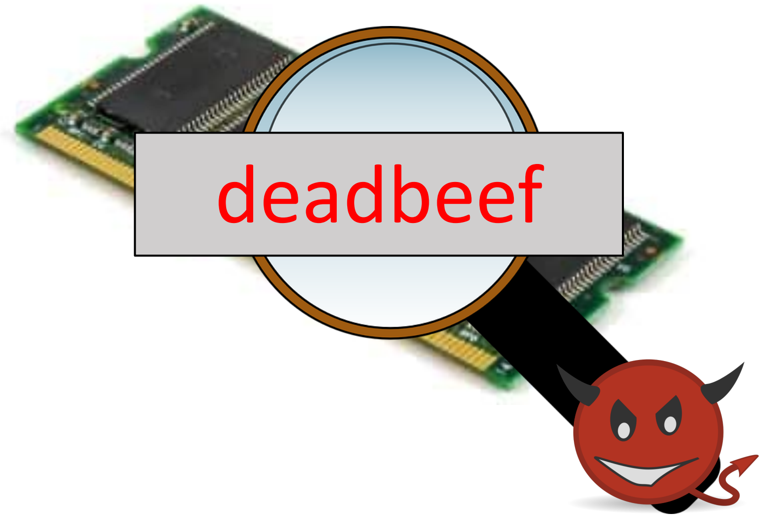
[1] Georgia Tech

[2] arm Research

Securing Main-Memory against Physical Attacks

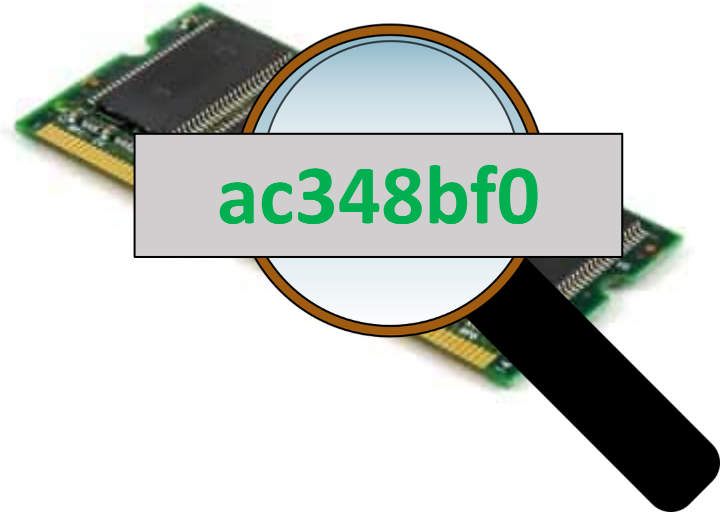
Securing Main-Memory against Physical Attacks

Unauthorized Reads



Securing Main-Memory against Physical Attacks

Unauthorized Reads

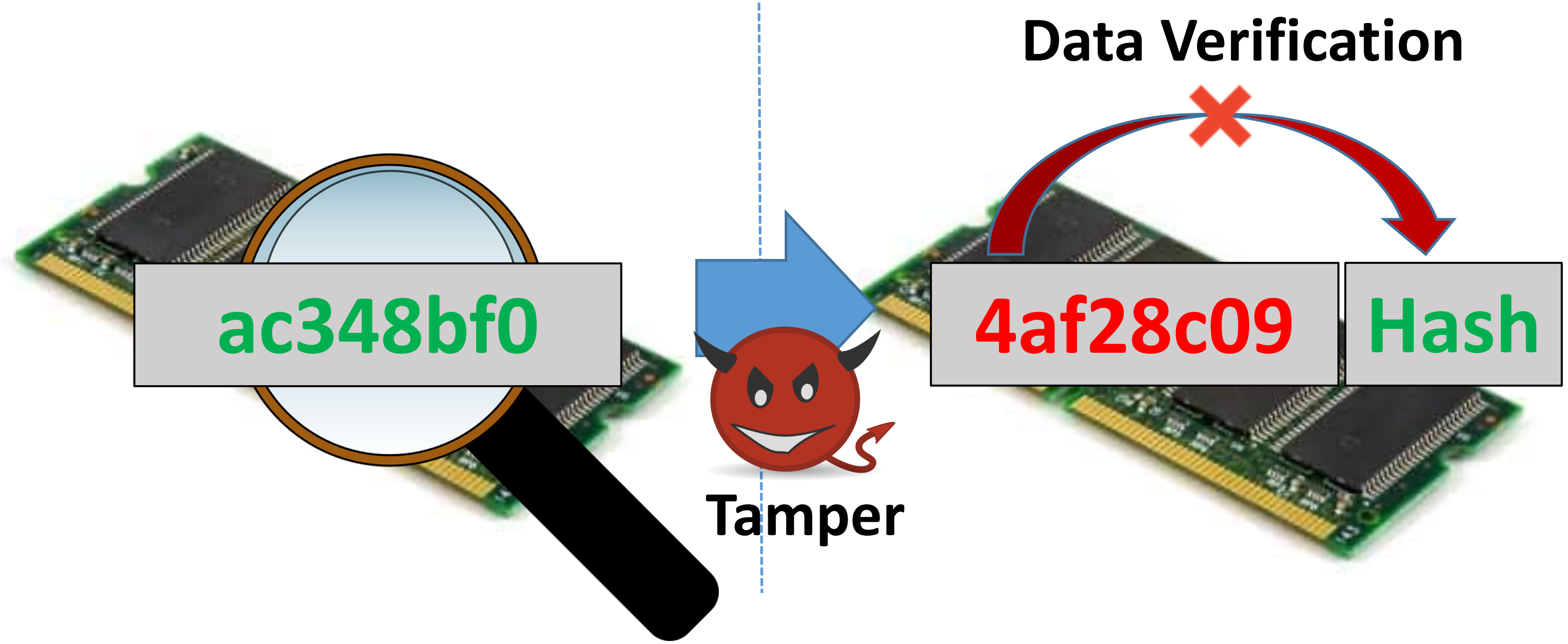


Data Encryption

Securing Main-Memory against Physical Attacks

Unauthorized Reads

Unauthorized Writes

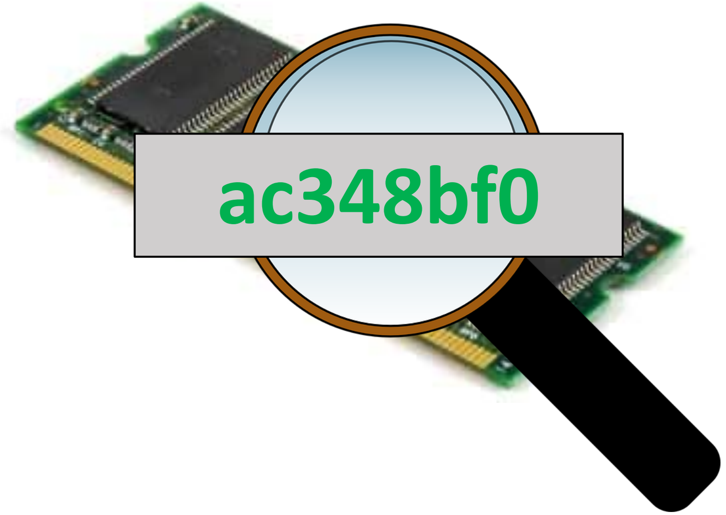


Data Encryption

Cryptographic Signatures

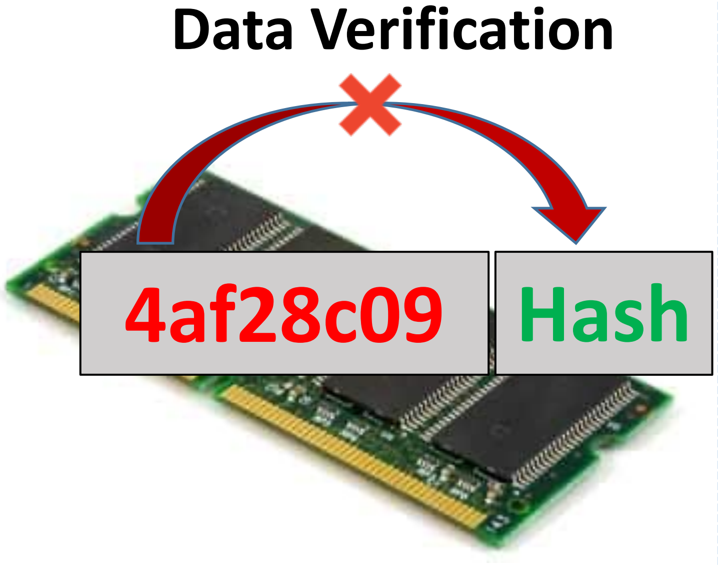
Securing Main-Memory against Physical Attacks

Unauthorized Reads



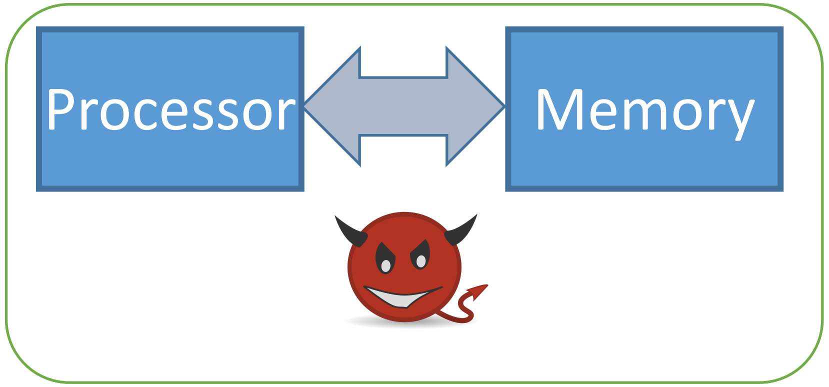
Data Encryption

Unauthorized Writes



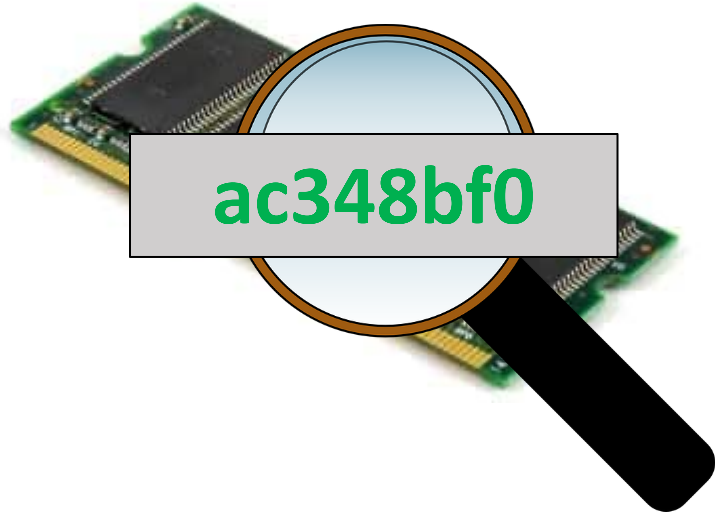
Cryptographic Signatures

Replay Attack



Securing Main-Memory against Physical Attacks

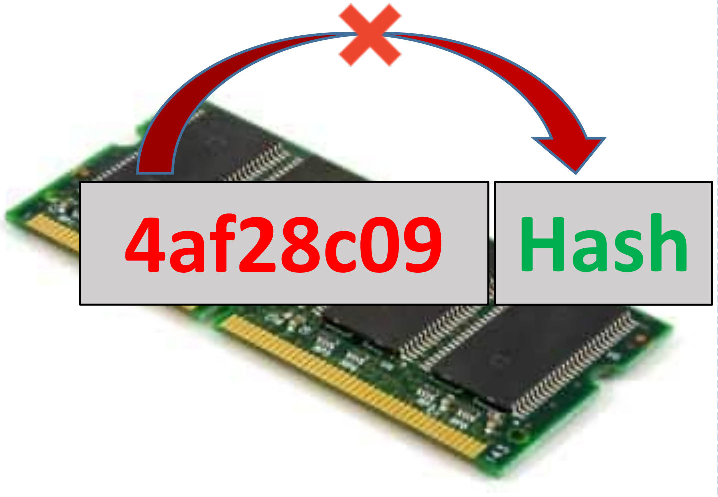
Unauthorized Reads



Data Encryption

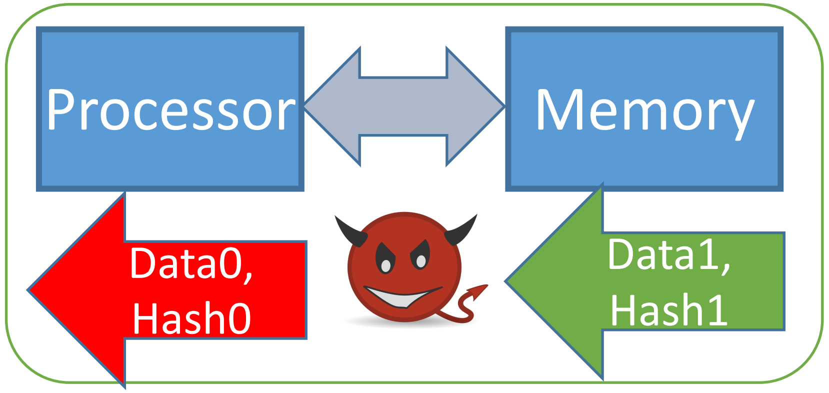
Unauthorized Writes

Data Verification



Cryptographic Signatures

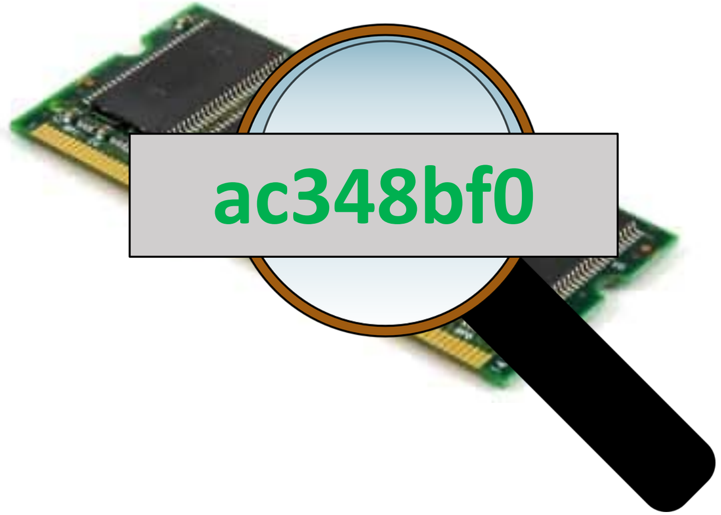
Replay Attack



Can force re-use of keys, repeat sensitive transactions etc.

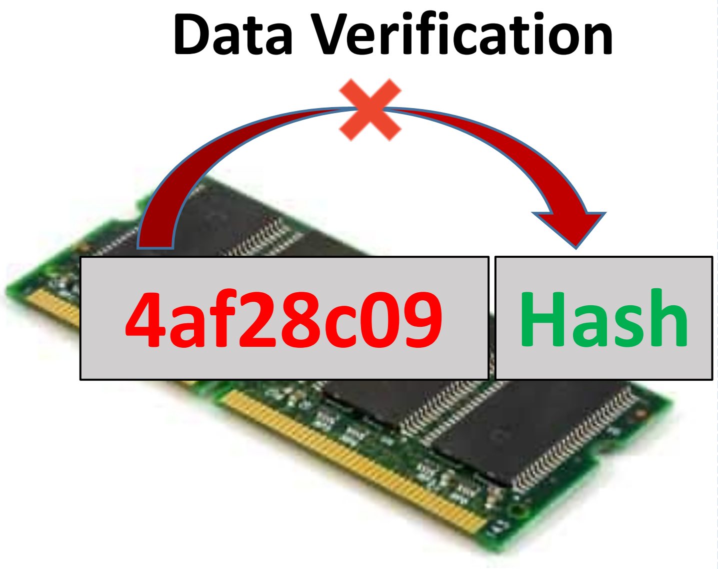
Securing Main-Memory against Physical Attacks

Unauthorized Reads



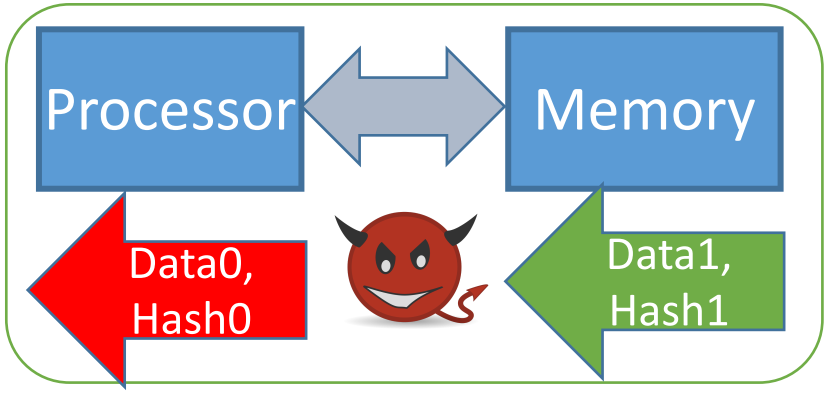
Data Encryption

Unauthorized Writes



Cryptographic Signatures

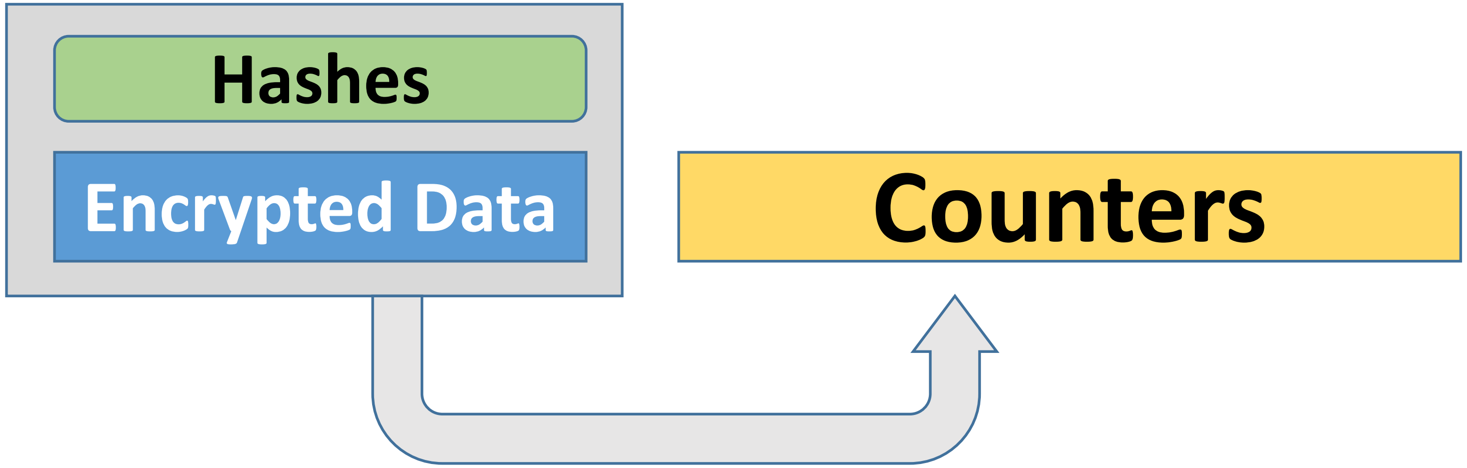
Replay Attack



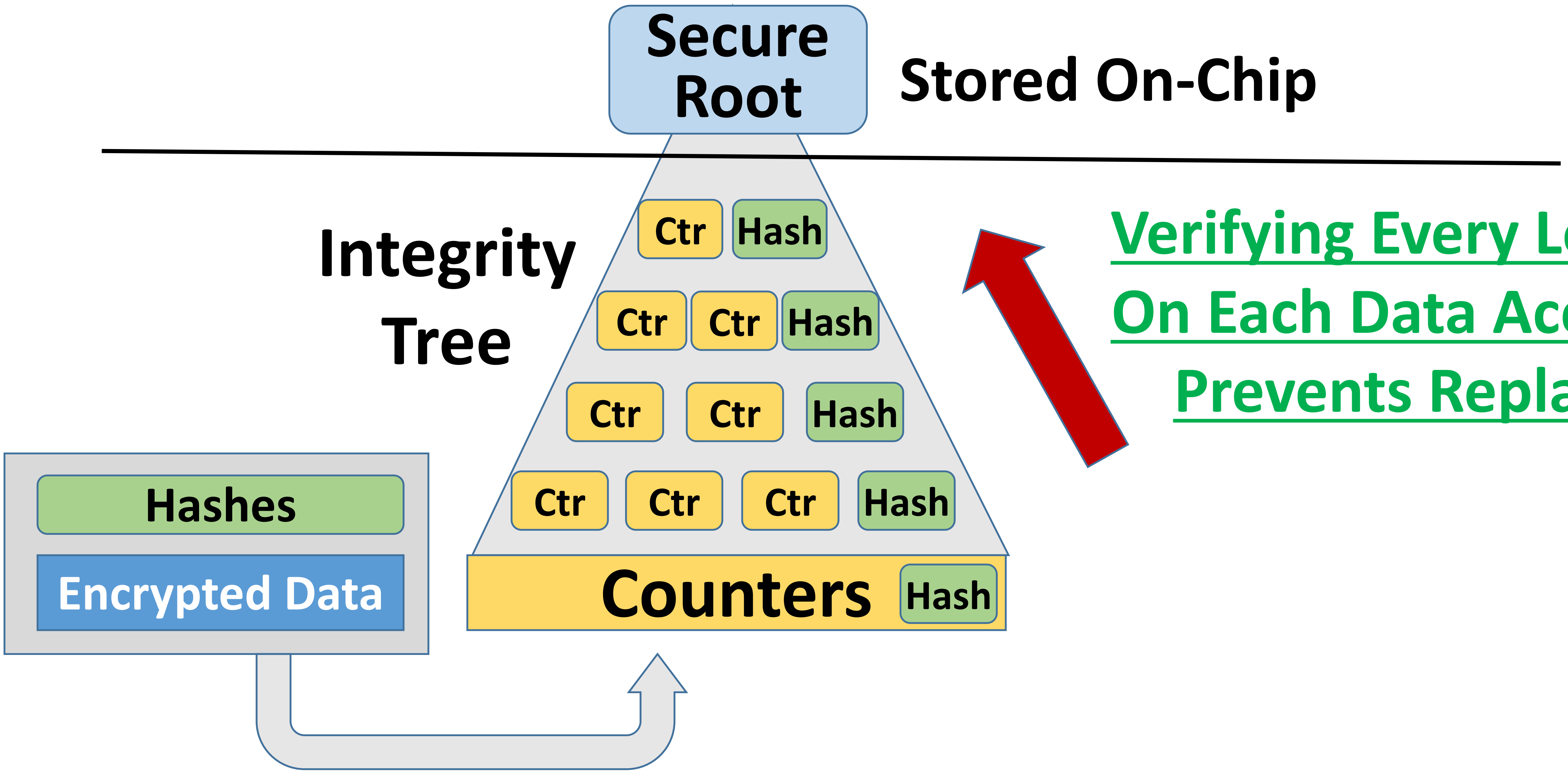
Can force re-use of keys, repeat sensitive transactions etc.

Replay attack protection - focus of this work

Replay Attack Protection with Integrity-Trees

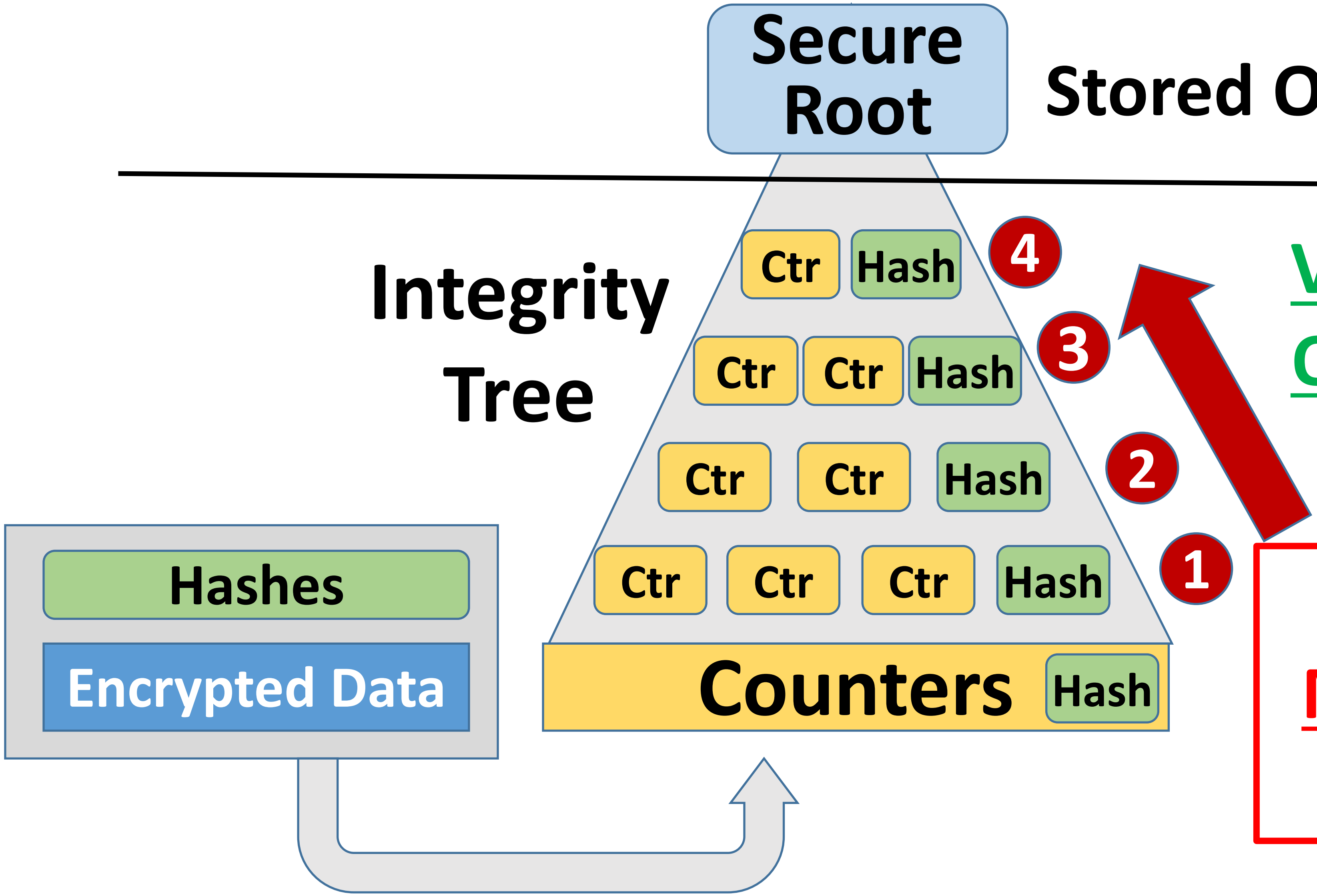


Replay Attack Protection with Integrity-Trees



Verifying Every Level On Each Data Access Prevents Replay

Replay Attack Protection with Integrity-Trees



Secure Root

Stored On-Chip

Integrity Tree

Ctr Hash 4

Ctr Ctr Hash 3

Ctr Ctr Hash 2

Ctr Ctr Ctr Hash 1

Counters Hash

Hashes

Encrypted Data

Verifying Every Level On Each Data Access Prevents Replay

Causes Extra Memory Accesses & Slowdown

This Talk: Designing Compact Integrity Trees

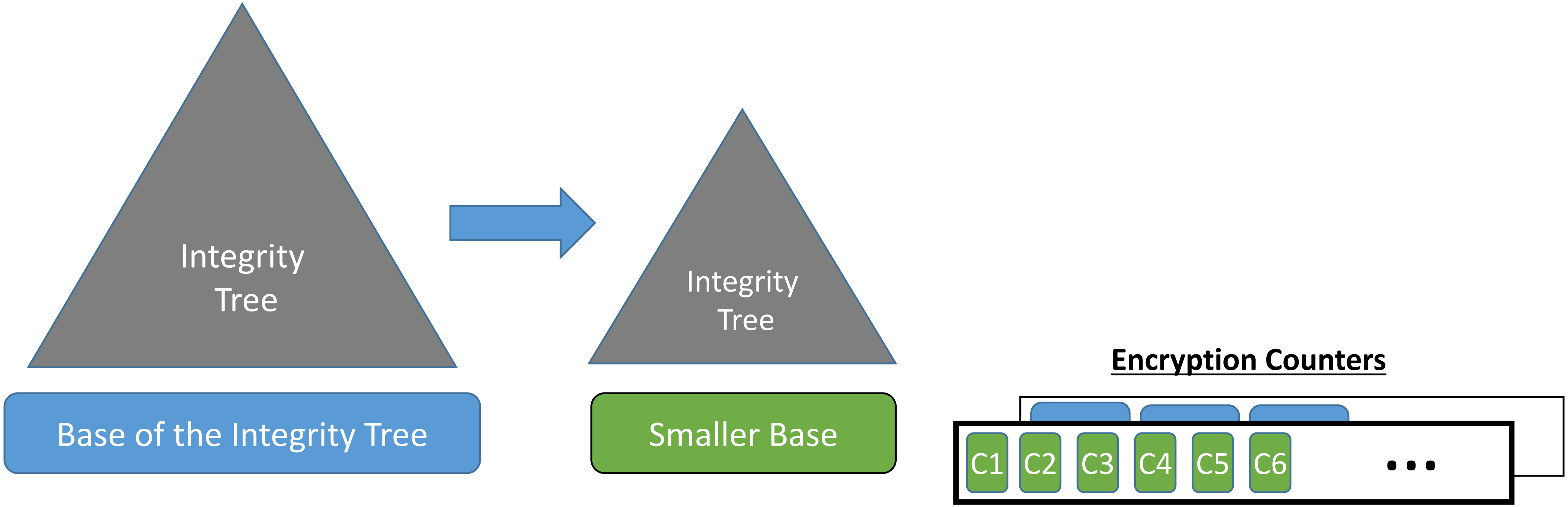
Integrity Tree

Base of the Integrity Tree

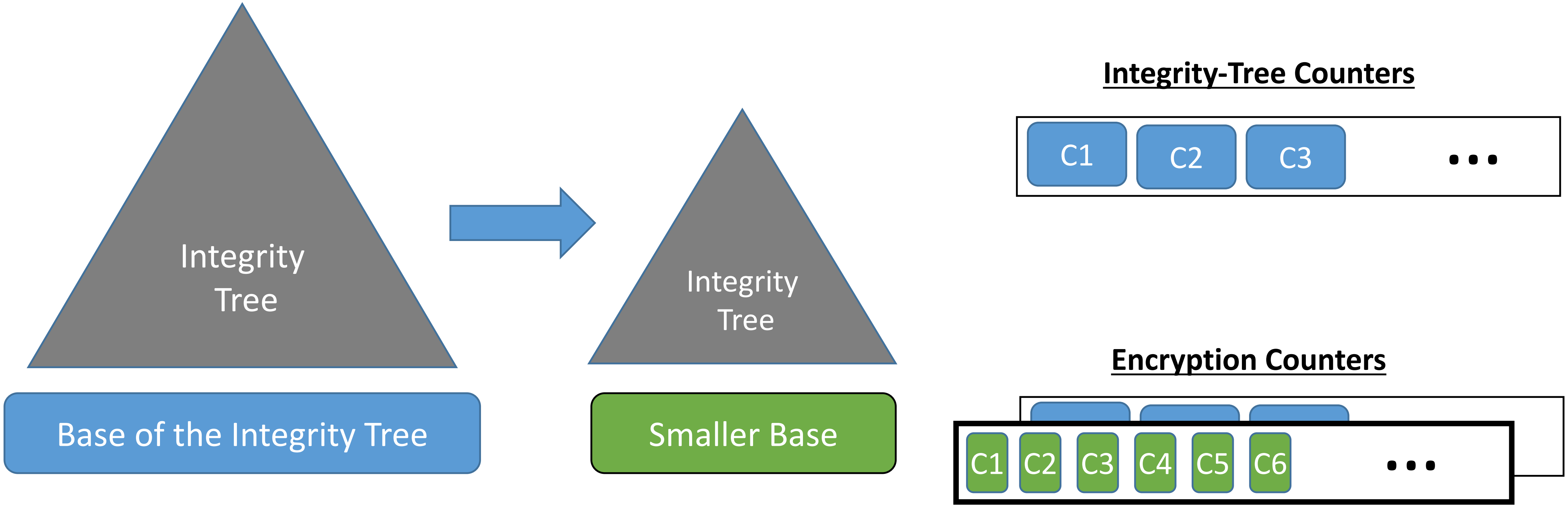
Encryption Counters

C1 C2 C3 …

This Talk: Designing Compact Integrity Trees



This Talk: Designing Compact Integrity Trees



Integrity Tree

Base of the Integrity Tree

Integrity Tree

Smaller Base

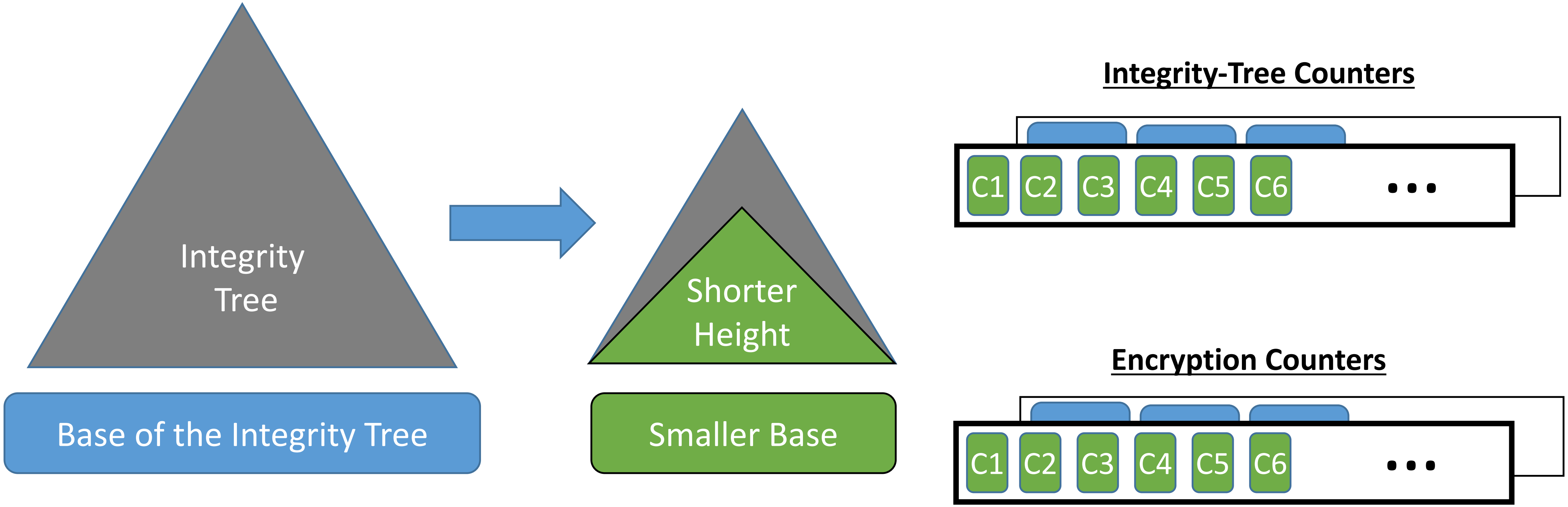
Integrity-Tree Counters

C1 C2 C3 …

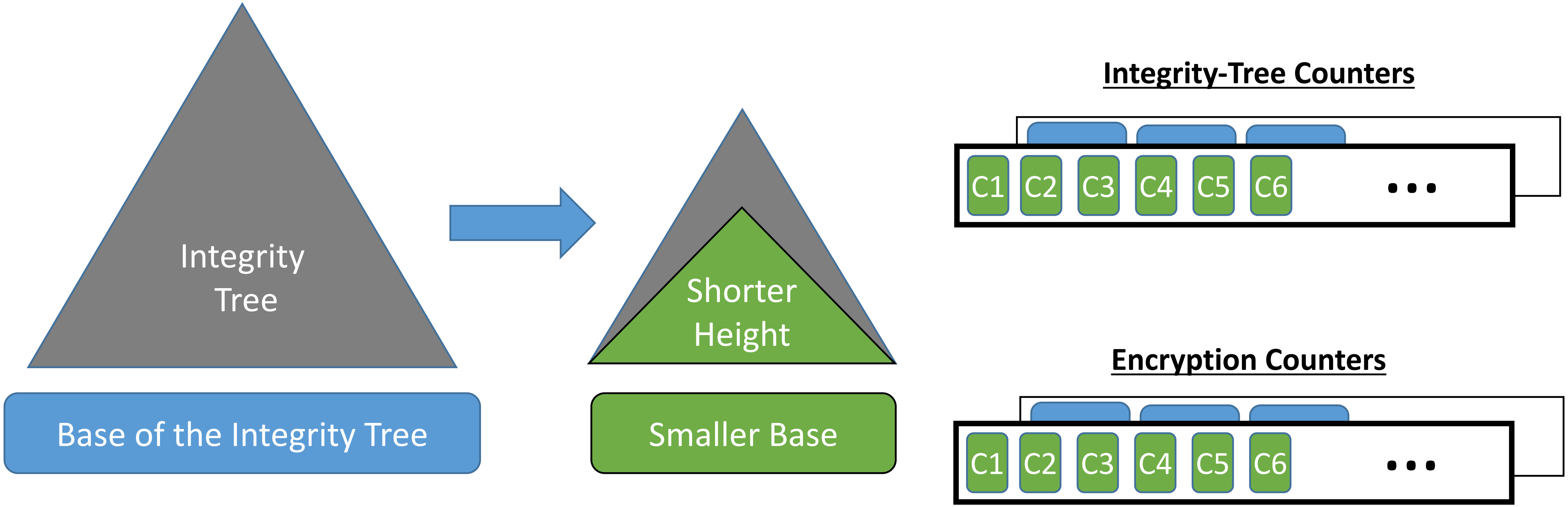
Encryption Counters

C1 C2 C3 C4 C5 C6 …

This Talk: Designing Compact Integrity Trees

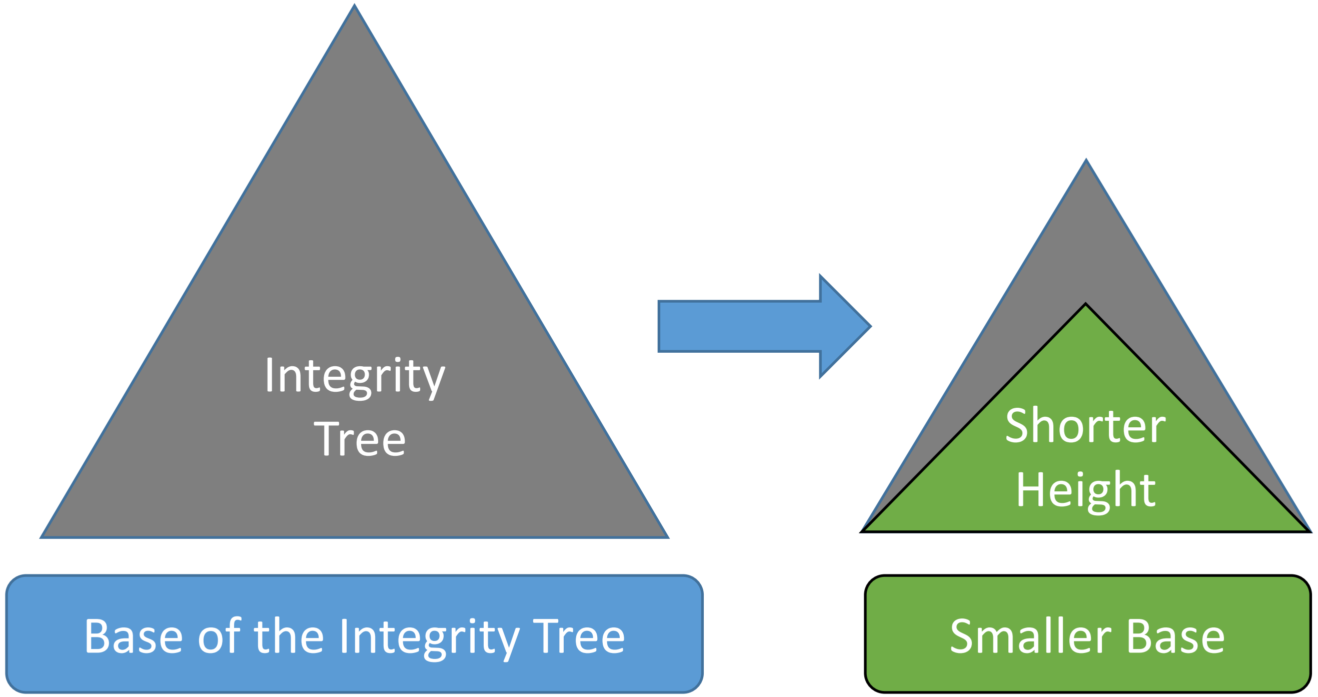


This Talk: Designing Compact Integrity Trees



Goal: Pack more Counters per Cacheline for Low-Overhead Integrity Trees

This Talk: Designing Compact Integrity Trees



Benefits of Our Design

Tree-Size

- **8.5x smaller** vs VAULT[1]
- **4x smaller** vs Baseline

Speedup

- **13.5%** vs VAULT[1]
- **6.3%** vs Baseline

[1] VAULT - Taassori et al., *ASPLOS, 2018*.

Goal: Pack more Counters per Cacheline for Low-Overhead Integrity Trees

Agenda

- Introduction

- **Background and Motivation**

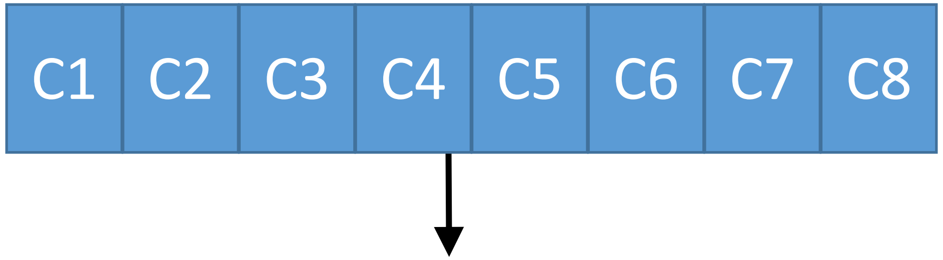
- Design

- Results

Split-Counters - More Counters/Cacheline

Naïve Counters

512 bit cache line

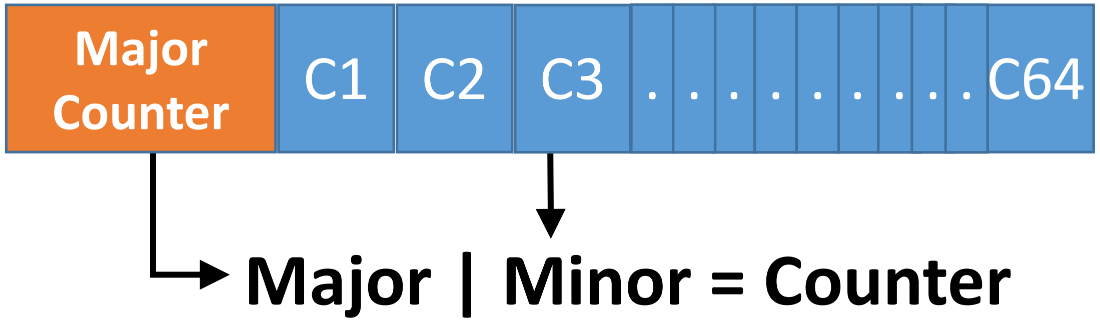


C1 C2 C3 C4 C5 C6 C7 C8

8 counters x 64b

Split-Counters - More Counters/Cacheline

Split Counters[1] *(Share Significant Bits)*

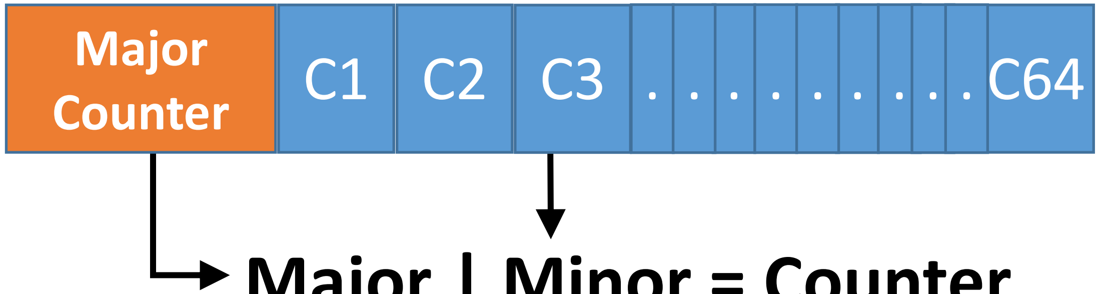


[1] Yan et al., *ISCA, 2006*

Split-Counters - More Counters/Cacheline

Split Counters*[1] *(Share Significant Bits)

64 x 7-bit minor counters



Major | Minor = Counter

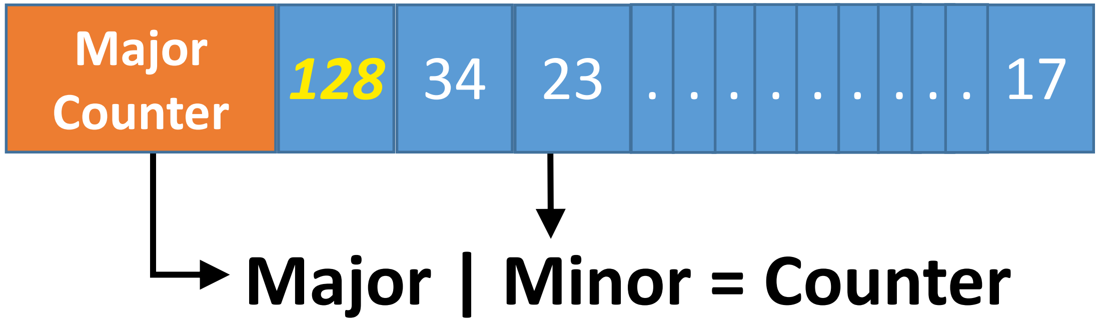
✓ **64-ary Design**

[1] Yan et al., *ISCA, 2006*

Split-Counters - More Counters/Cacheline

Split Counters[1] *(Share Significant Bits)*

64 x 7-bit minor counters



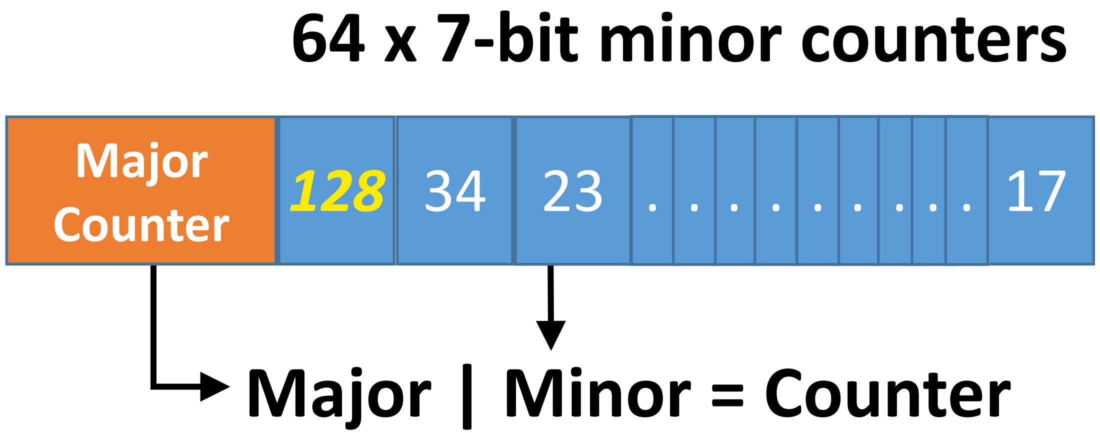
7-bit Minor Counters Can Overflow

✓ **64-ary Design**

[1] Yan et al., *ISCA, 2006*

Split-Counters - More Counters/Cacheline

Split Counters[1] *(Share Significant Bits)*



✓ **64-ary Design**

7-bit Minor Counters Can Overflow

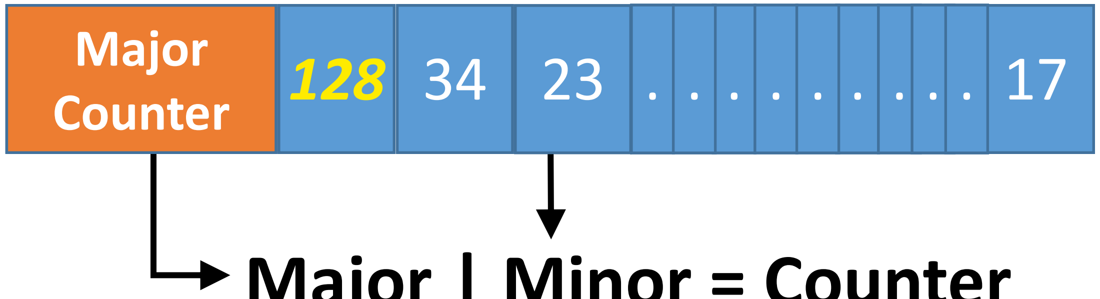
Increment shared Major counter → ***Changes values of ALL counters!***

[1] Yan et al., *ISCA, 2006*

Split-Counters - More Counters/Cacheline

Split Counters*[1] *(Share Significant Bits)

64 x 7-bit minor counters



Major | Minor = Counter

✓ **64-ary Design**

7-bit Minor Counters Can Overflow

Increment shared Major counter → ***Changes values of ALL counters!***

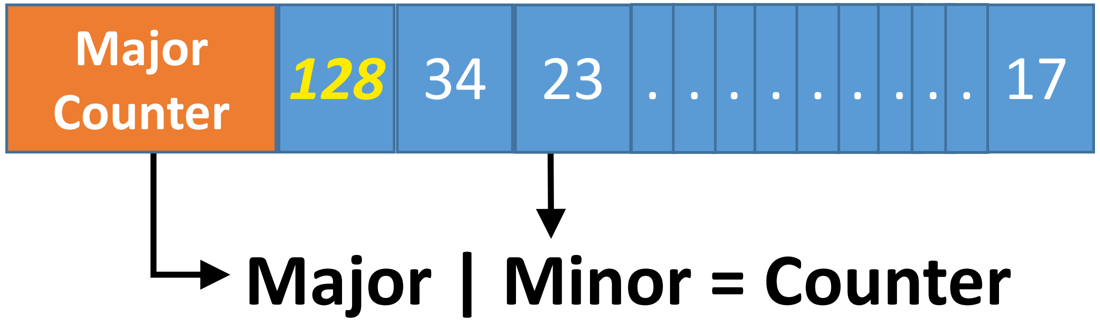
✗ **1. Re-encrypt 64 Data Lines (128 reads/writes)**

[1] Yan et al., *ISCA, 2006*

Split-Counters - More Counters/Cacheline

Split Counters*[1] *(Share Significant Bits)

64 x 7-bit minor counters



Major Counter

128 34 23 17

Major | Minor = Counter

✓ **64-ary Design**

7-bit Minor Counters Can Overflow

Increment shared Major counter → ***Changes values of ALL counters!***

✗

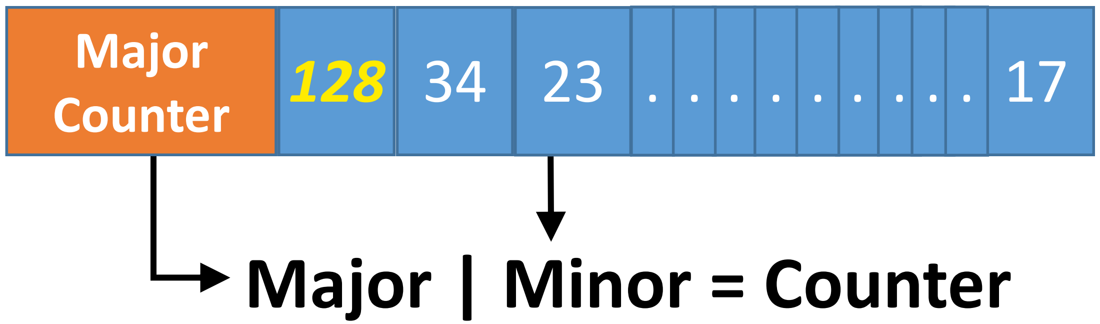
1. **Re-encrypt 64 Data Lines (128 reads/writes)**
2. **Update 64 Hashes (128 reads/writes)**

[1] Yan et al., *ISCA, 2006*

Split-Counters - More Counters/Cacheline

Split Counters*[1] *(Share Significant Bits)

64 x 7-bit minor counters



✓ **64-ary Design**

7-bit Minor Counters Can Overflow

Increment shared Major counter → ***Changes values of ALL counters!***

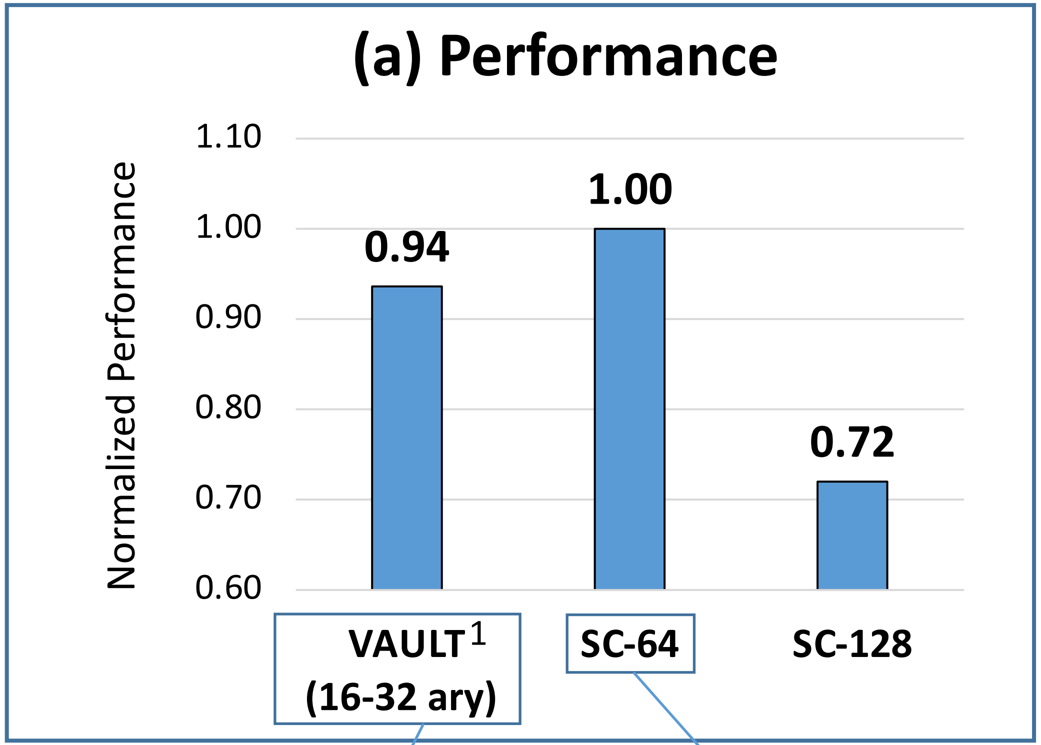
✗

1. **Re-encrypt 64 Data Lines (128 reads/writes)**
2. **Update 64 Hashes (128 reads/writes)**

Trade-off: Packing more Counters vs Overflow Updates !

[1] Yan et al., *ISCA, 2006*

Impact of Packing More Counters

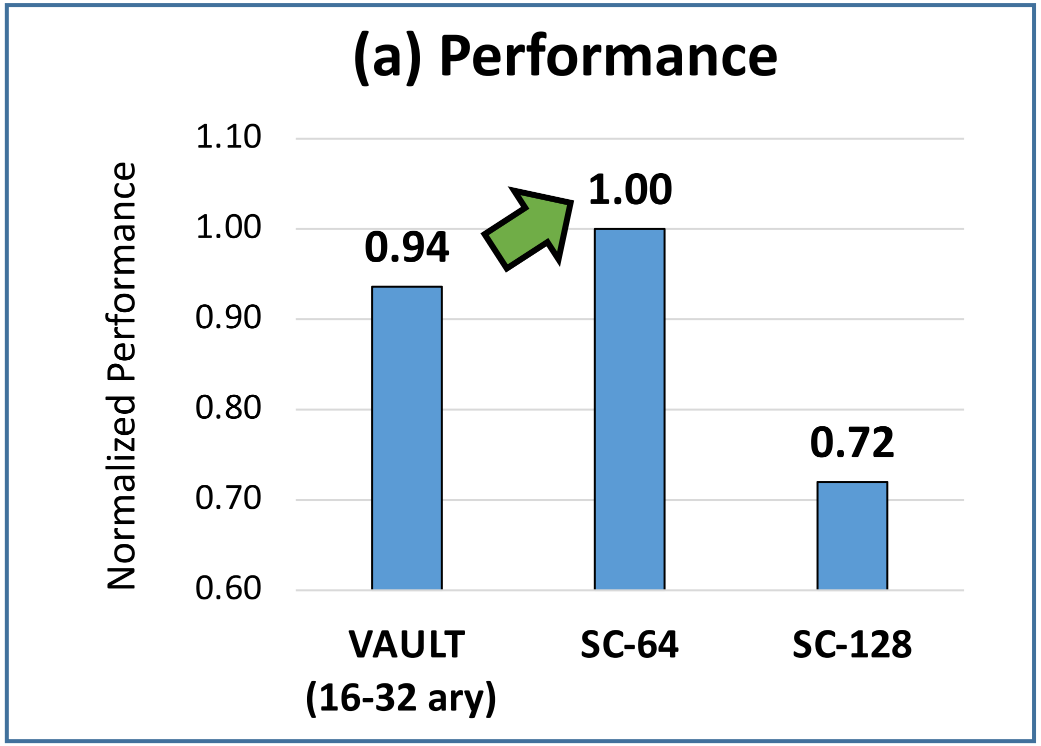


State of the art

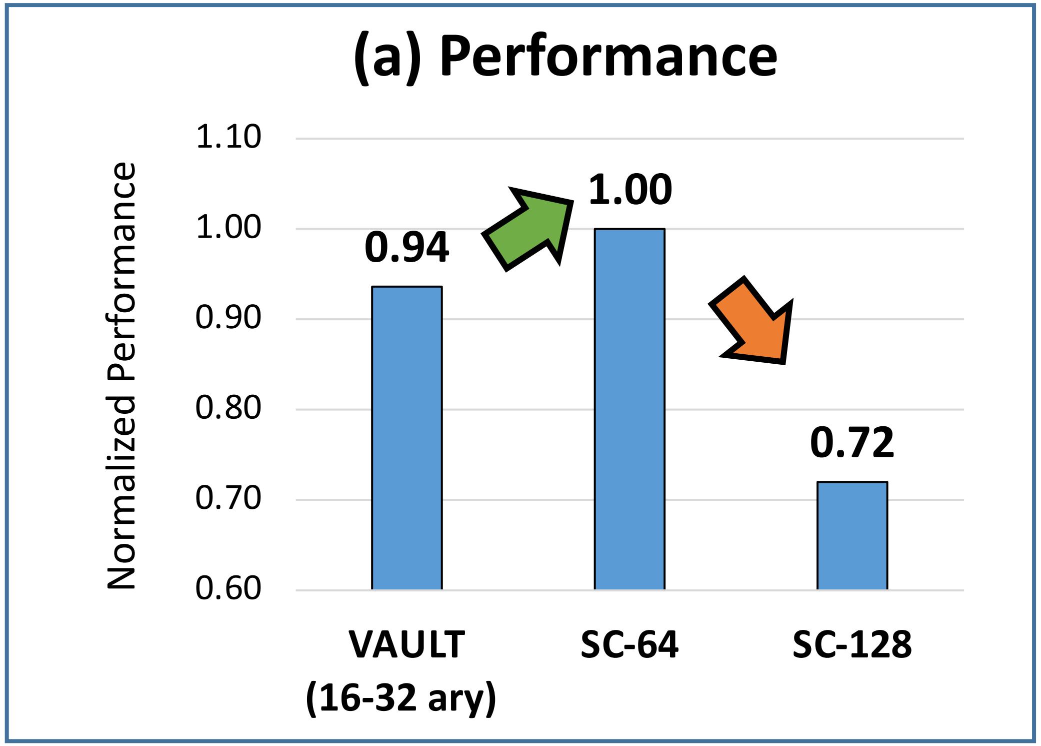
Baseline

[1] VAULT - Taassori et al., *ASPLOS, 2018*.

Impact of Packing More Counters

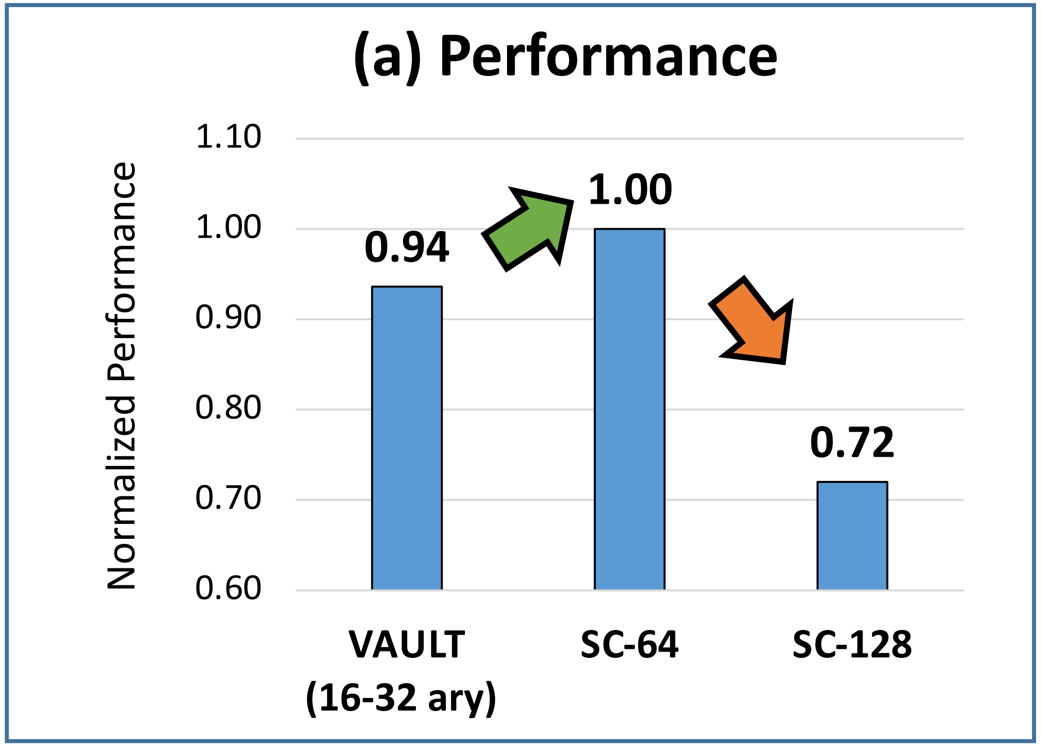


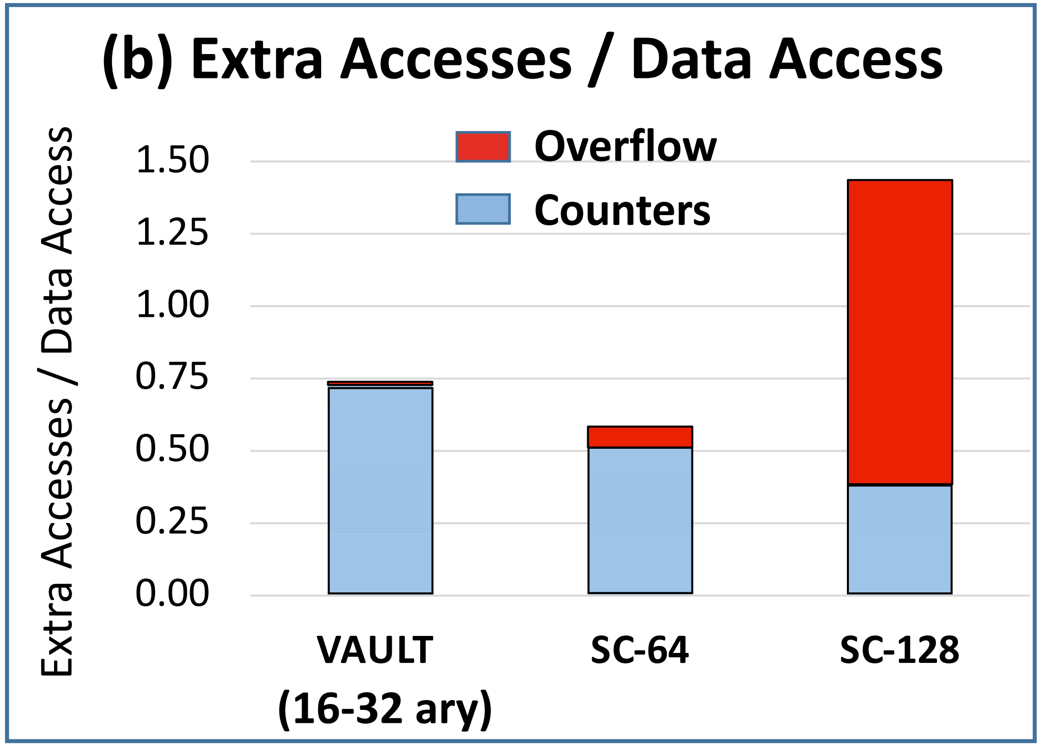
Impact of Packing More Counters



Performance Increases, then Decreases!

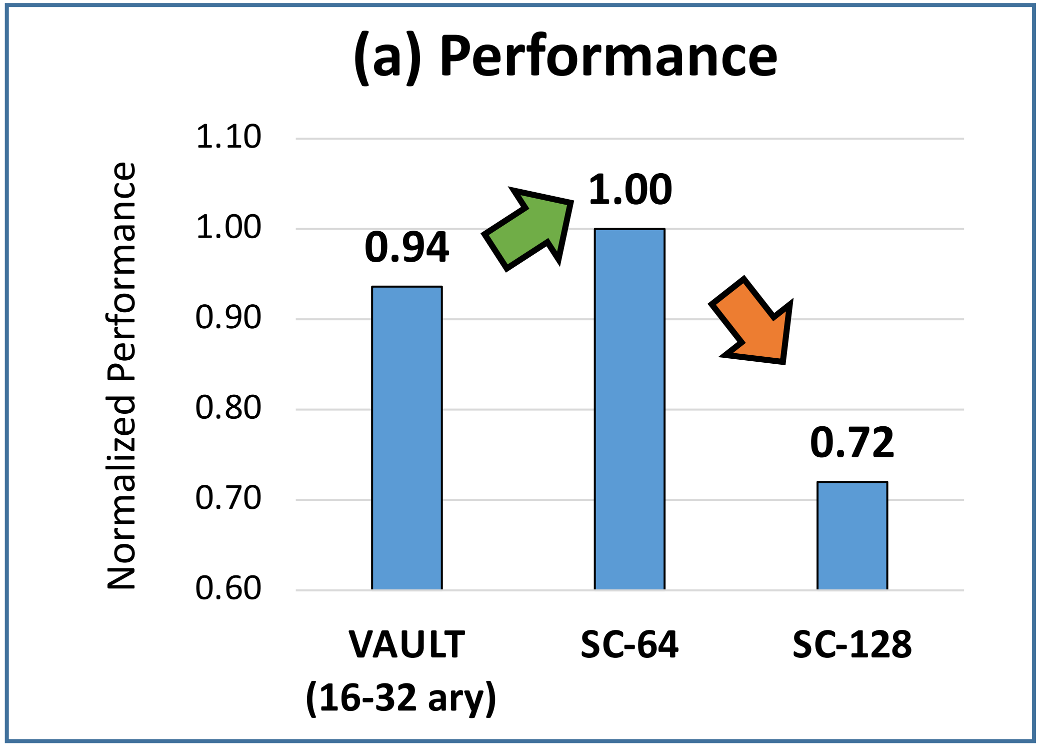
Impact of Packing More Counters

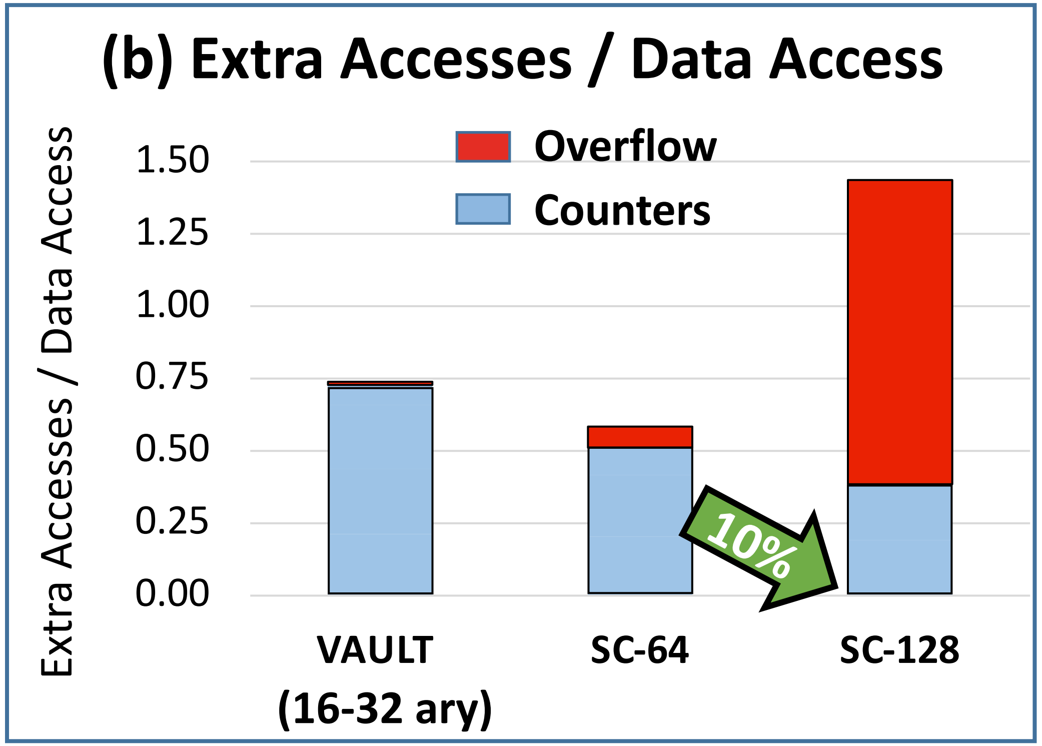




Performance Increases, then Decreases!

Impact of Packing More Counters

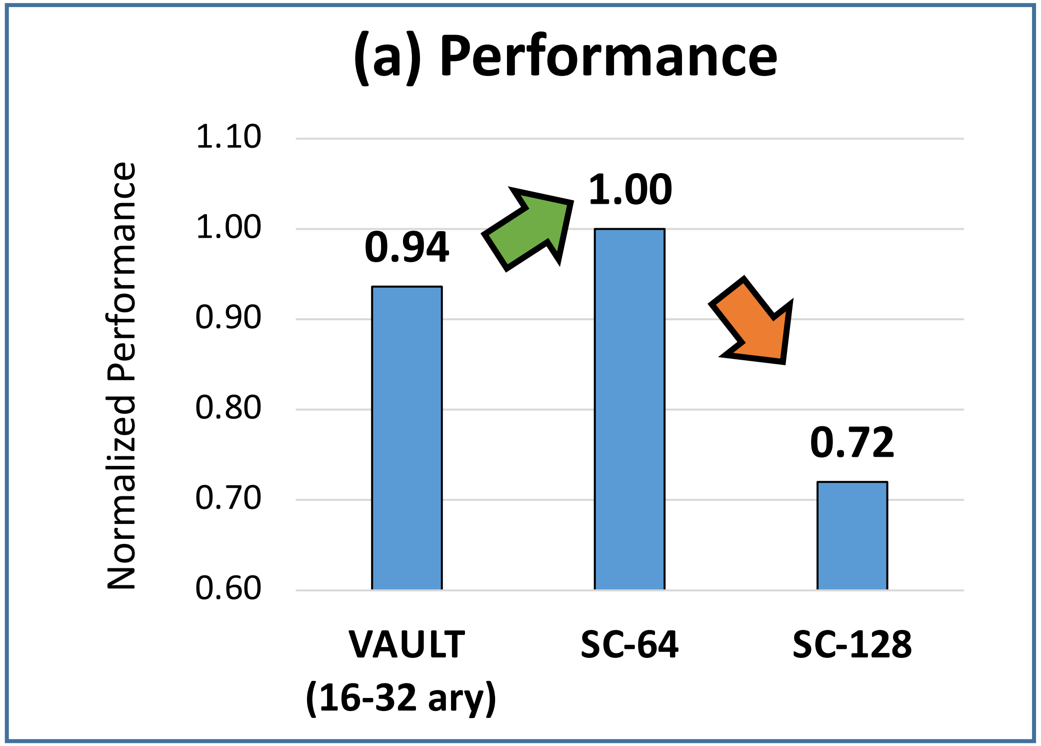


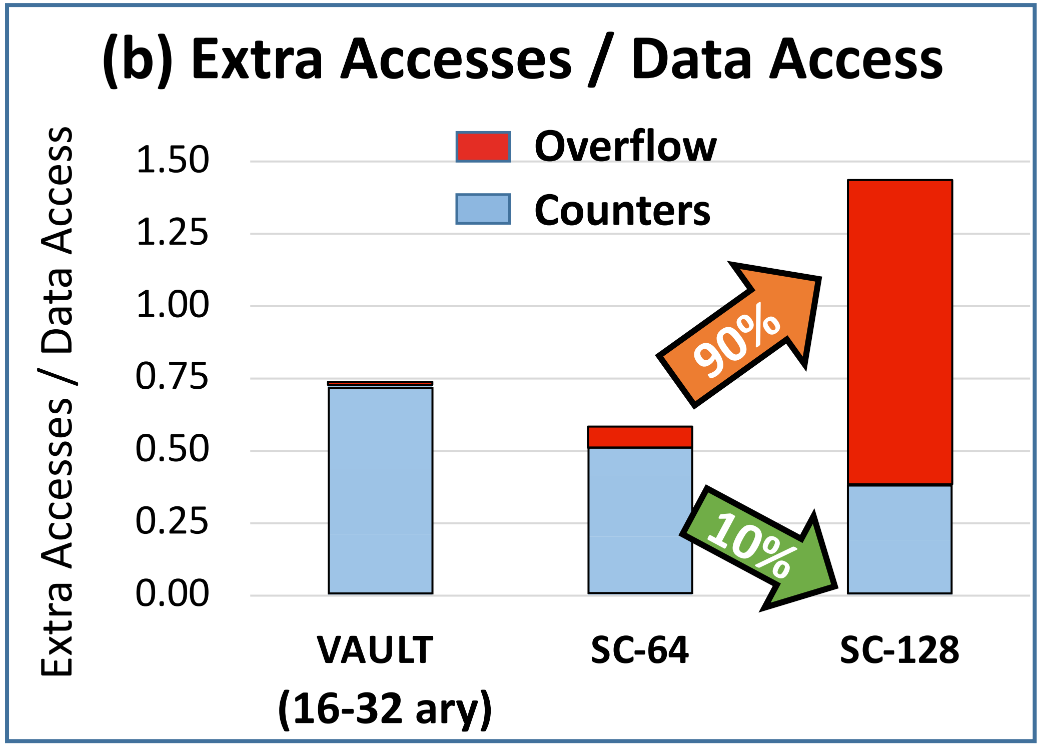


Performance Increases, then Decreases!

Benefits (counter accesses),

Impact of Packing More Counters

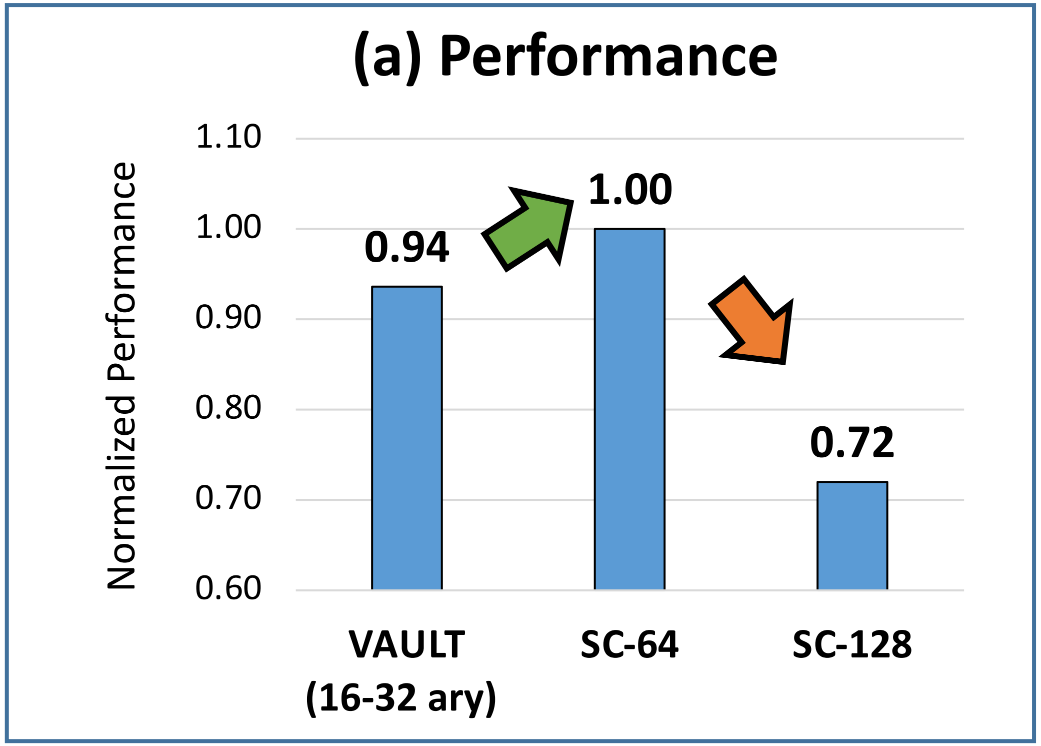


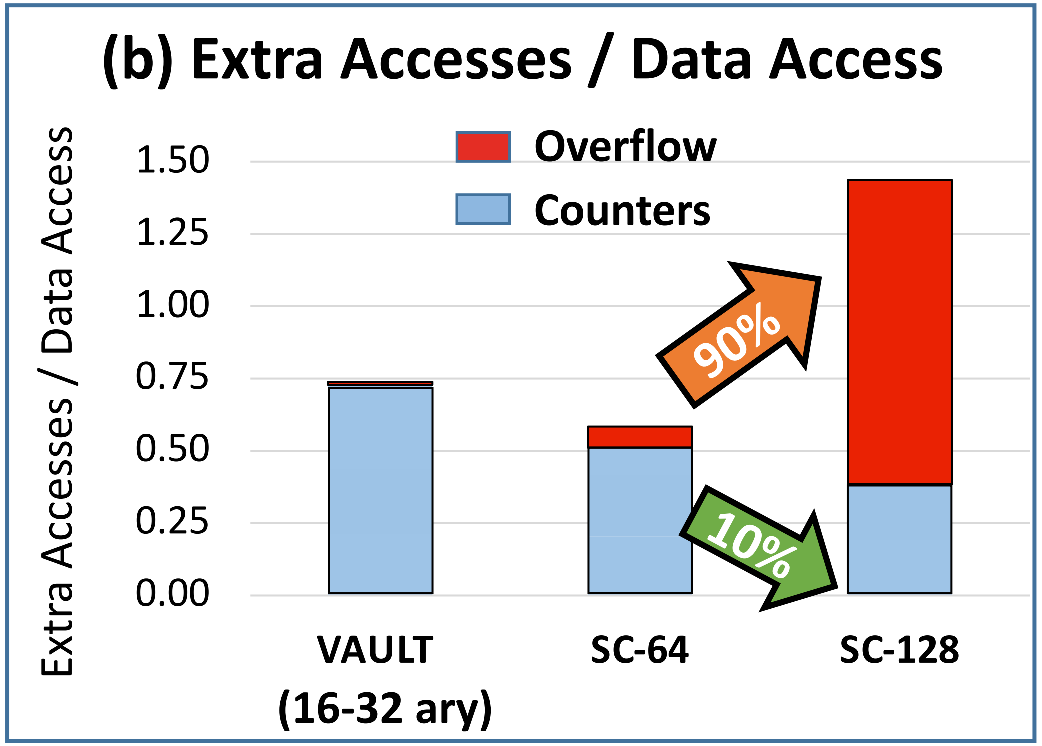


Performance Increases, then Decreases!

Benefits (counter accesses), outweighed by costs (overflows)

Impact of Packing More Counters





Performance Increases, then Decreases!

Benefits (counter accesses), outweighed by costs (overflows)

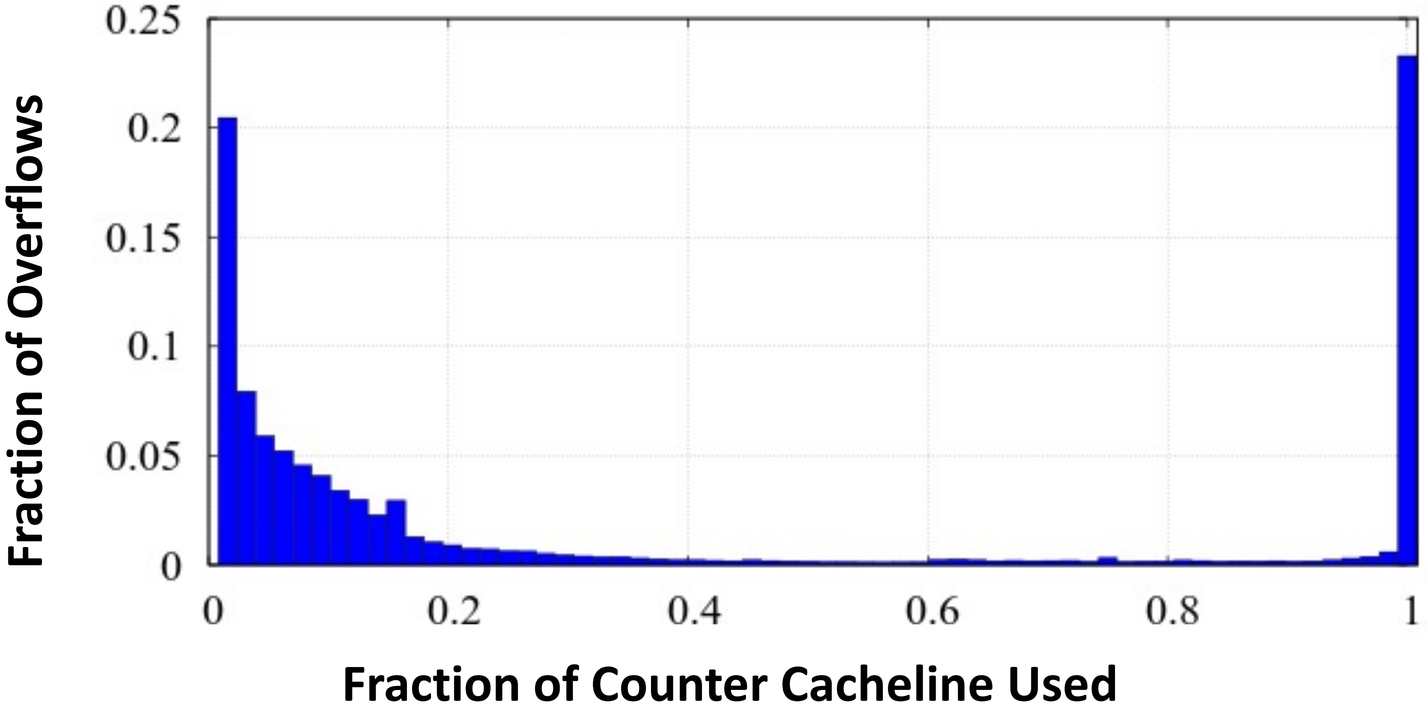
Goal: Pack more counters/cacheline, but fewer overflows !

Agenda

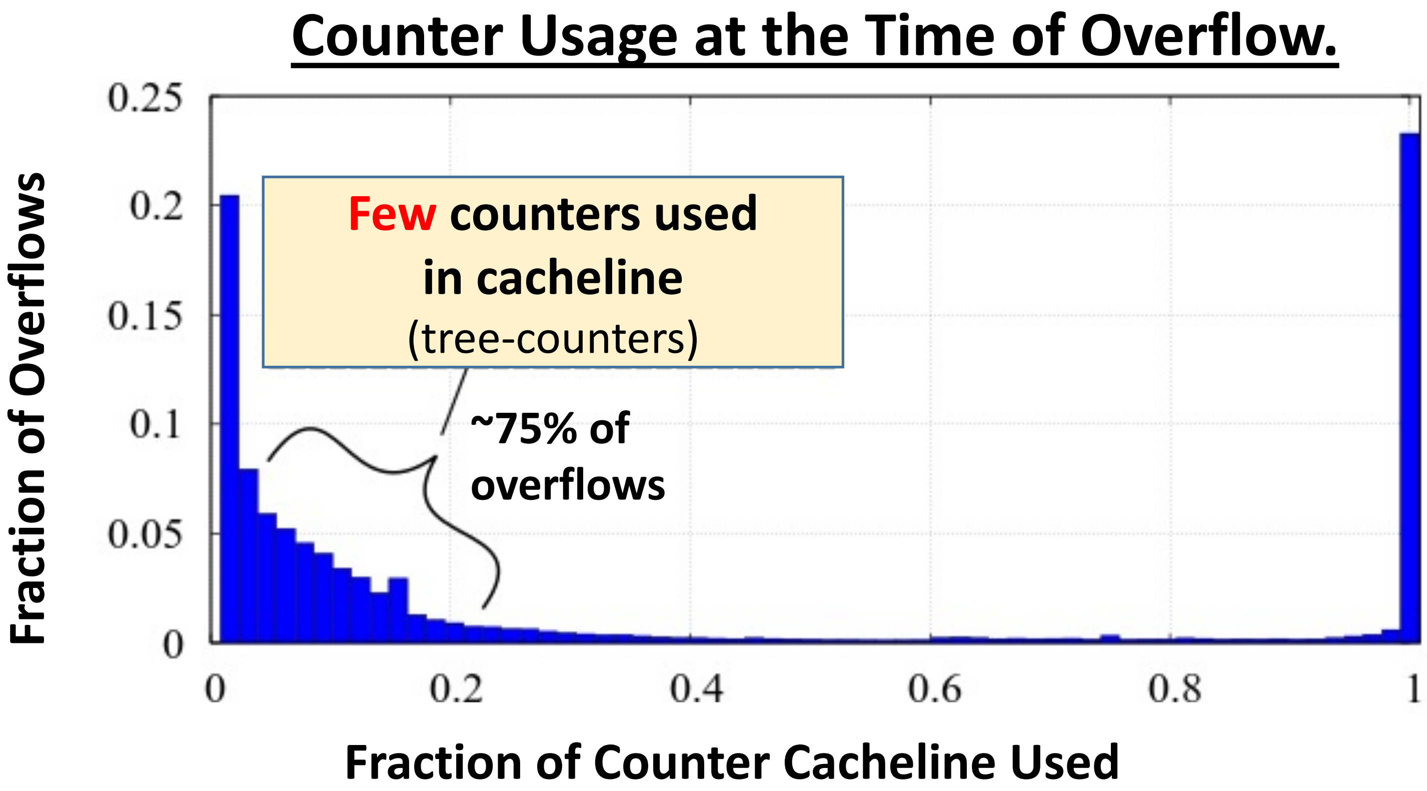
- Introduction
- Background and Motivation
- **Design**
- Results

Analysis of Counter Overflows

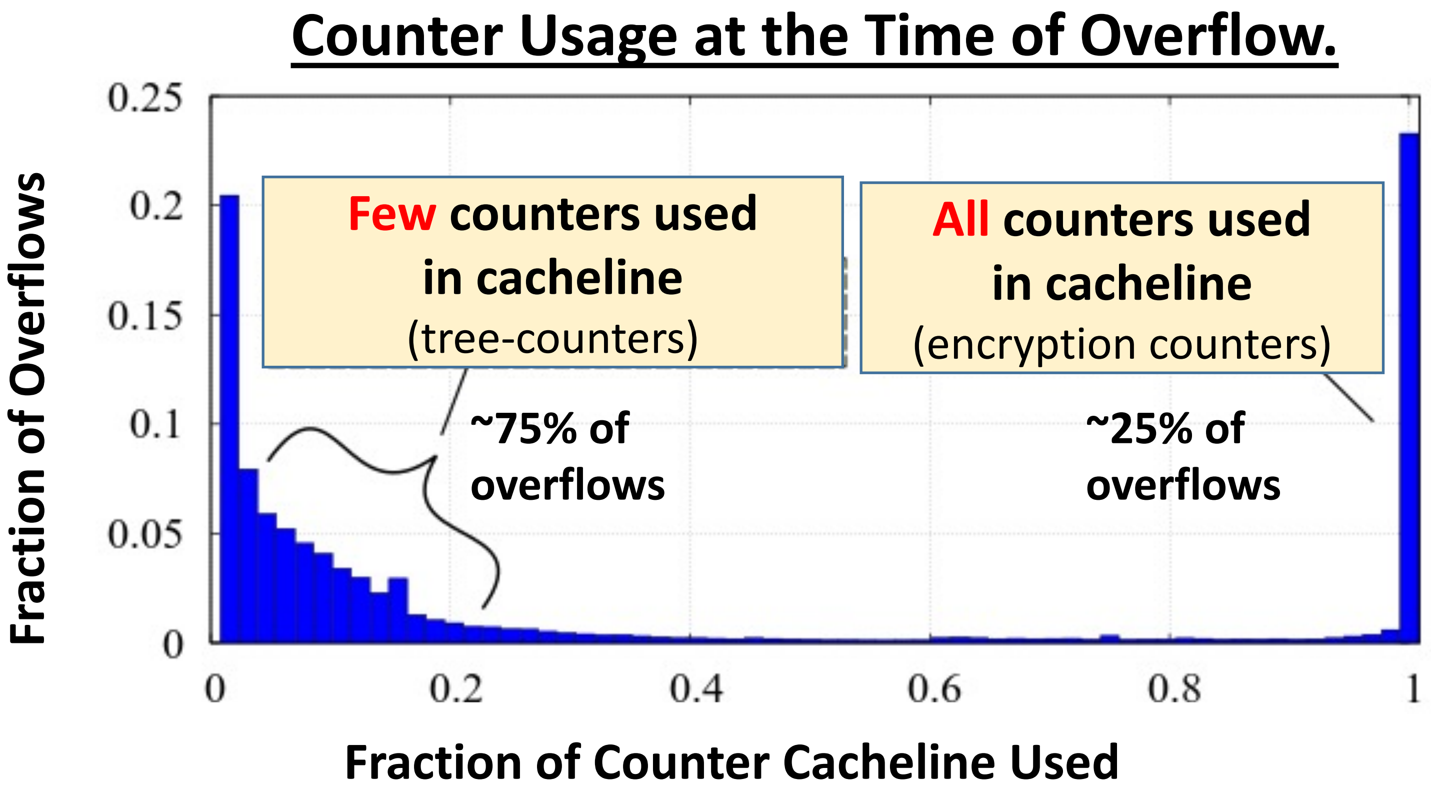
Counter Usage at the Time of Overflow.



Analysis of Counter Overflows

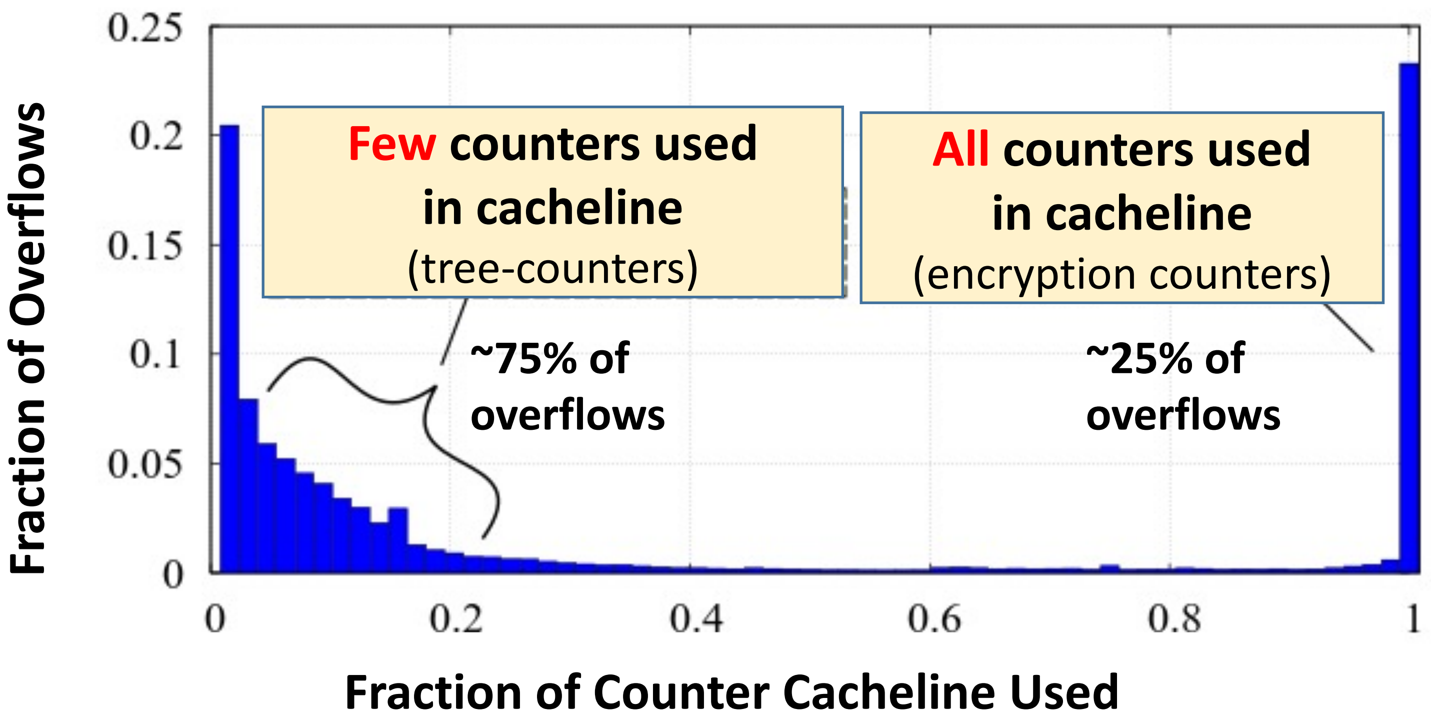


Analysis of Counter Overflows



Analysis of Counter Overflows

Counter Usage at the Time of Overflow.



Few counters used in cacheline (tree-counters)

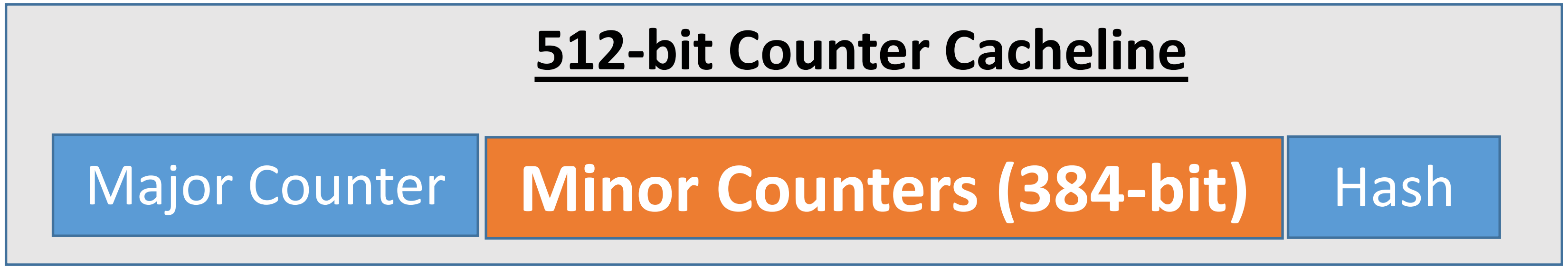
~75% of overflows

All counters used in cacheline (encryption counters)

~25% of overflows

Insight: Bimodal pattern in overflows → *Morphable Counters* with customized formats to reduce overflows

Few Counters Used: Compress Zero Counters



512-bit Counter Cacheline

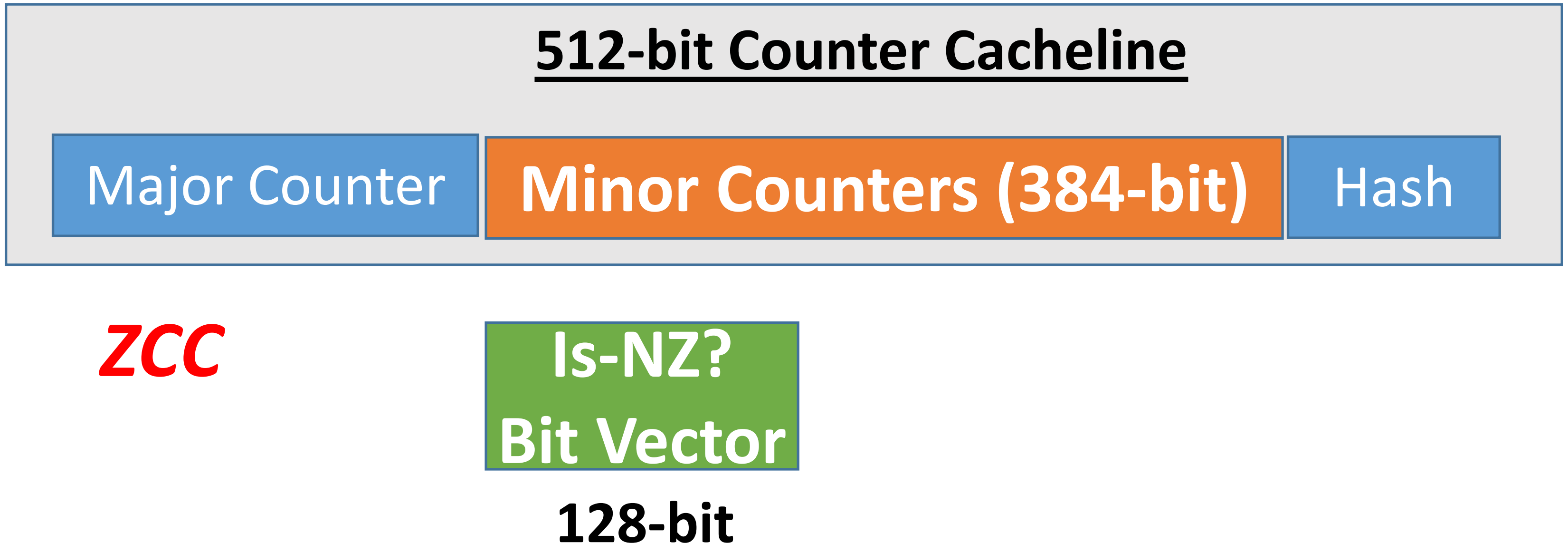
Major Counter

Minor Counters (384-bit)

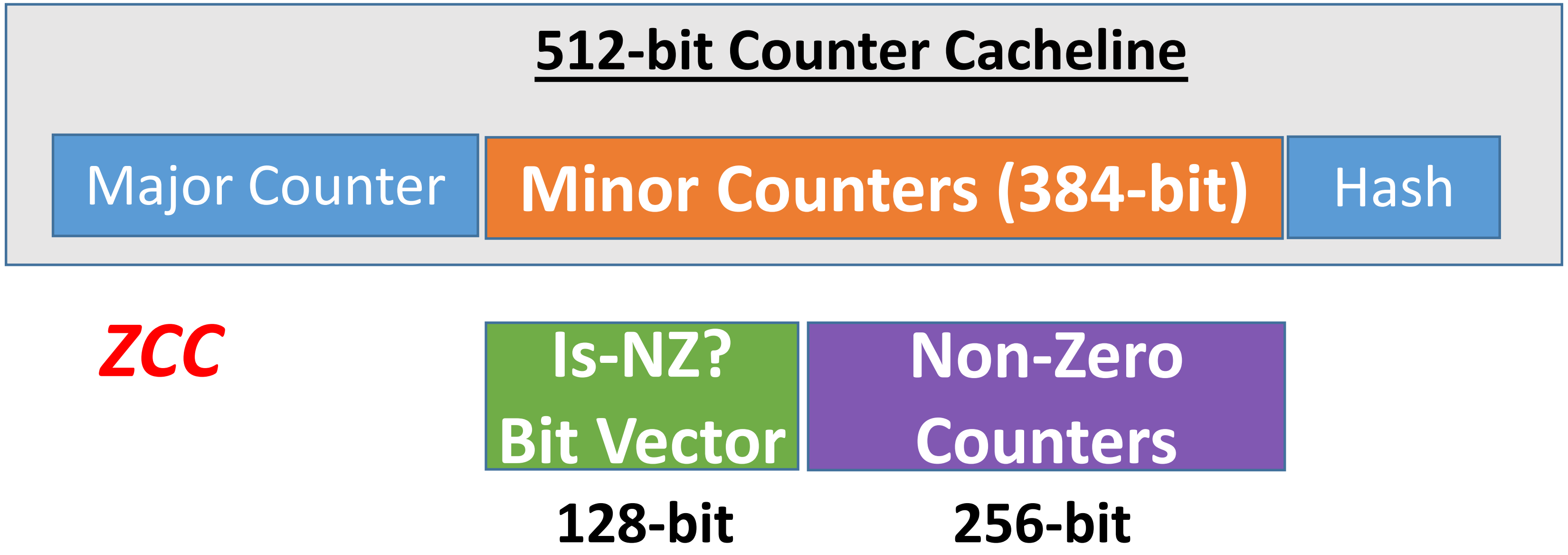
Hash

Insight: When few counters non-zero, allocate bits only to them

Few Counters Used: Compress Zero Counters



Few Counters Used: Compress Zero Counters



512-bit Counter Cacheline

Major Counter

Minor Counters (384-bit)

Hash

ZCC

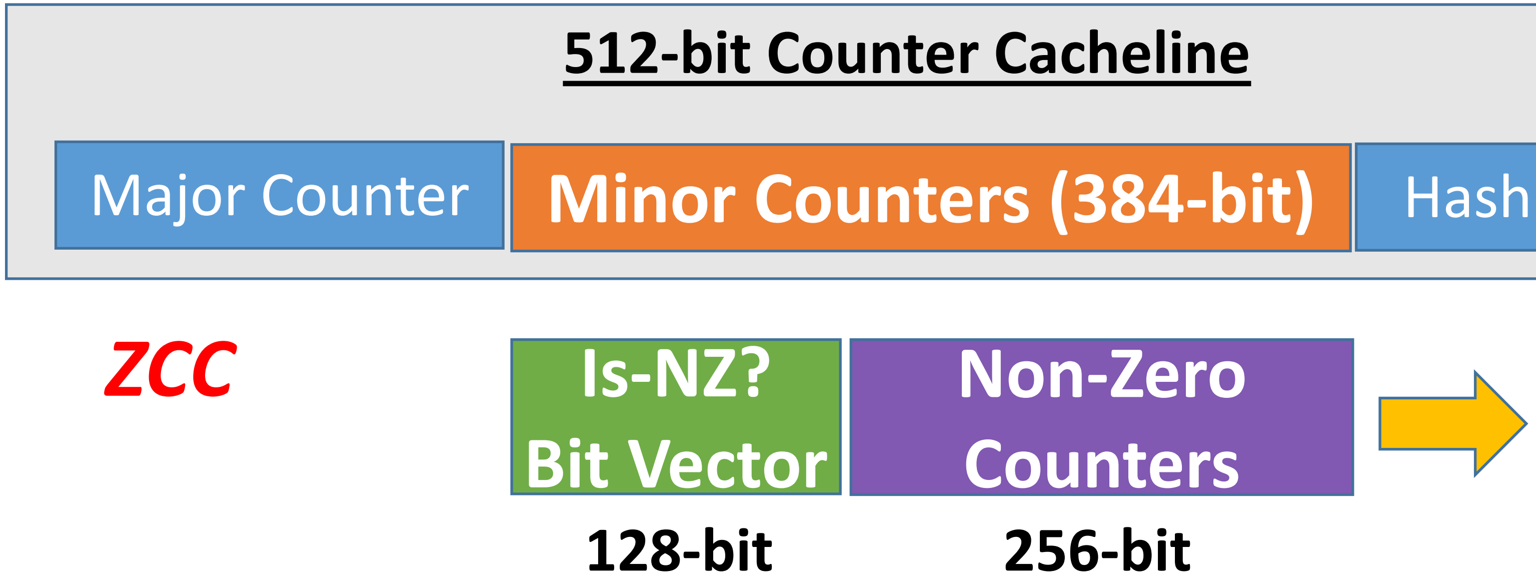
Is-NZ? Bit Vector

128-bit

Non-Zero Counters

256-bit

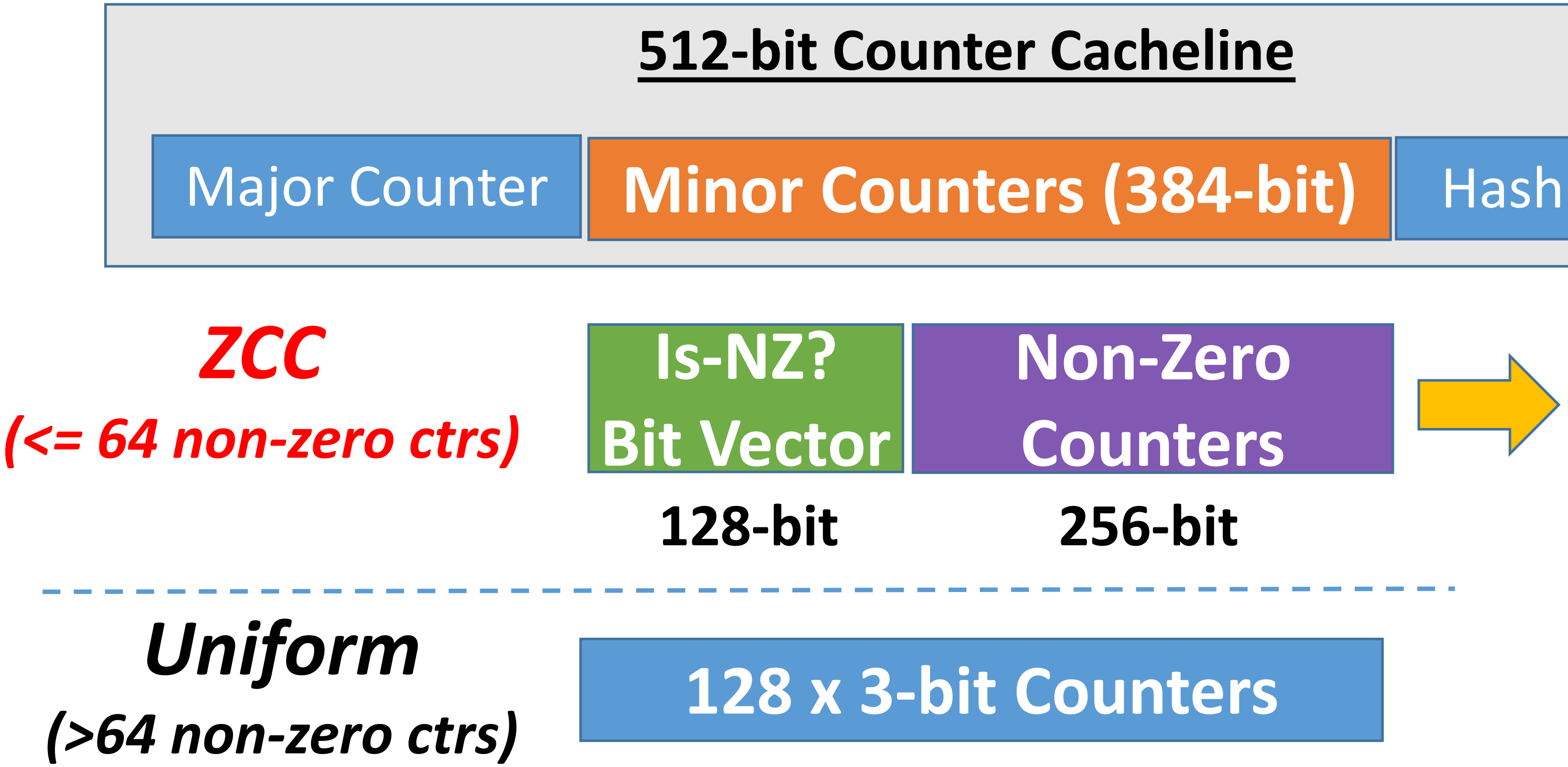
Few Counters Used: Compress Zero Counters



512-bit Counter Cacheline

Non-Zero Counters	Counter Size
<=16 ctrs	16-bits/ctr
<=32 ctrs	8-bits/ctr

Few Counters Used: Compress Zero Counters



512-bit Counter Cacheline

Major Counter | Minor Counters (384-bit) | Hash

ZCC

(<= 64 non-zero ctrs)

Is-NZ? Bit Vector | Non-Zero Counters

128-bit | 256-bit

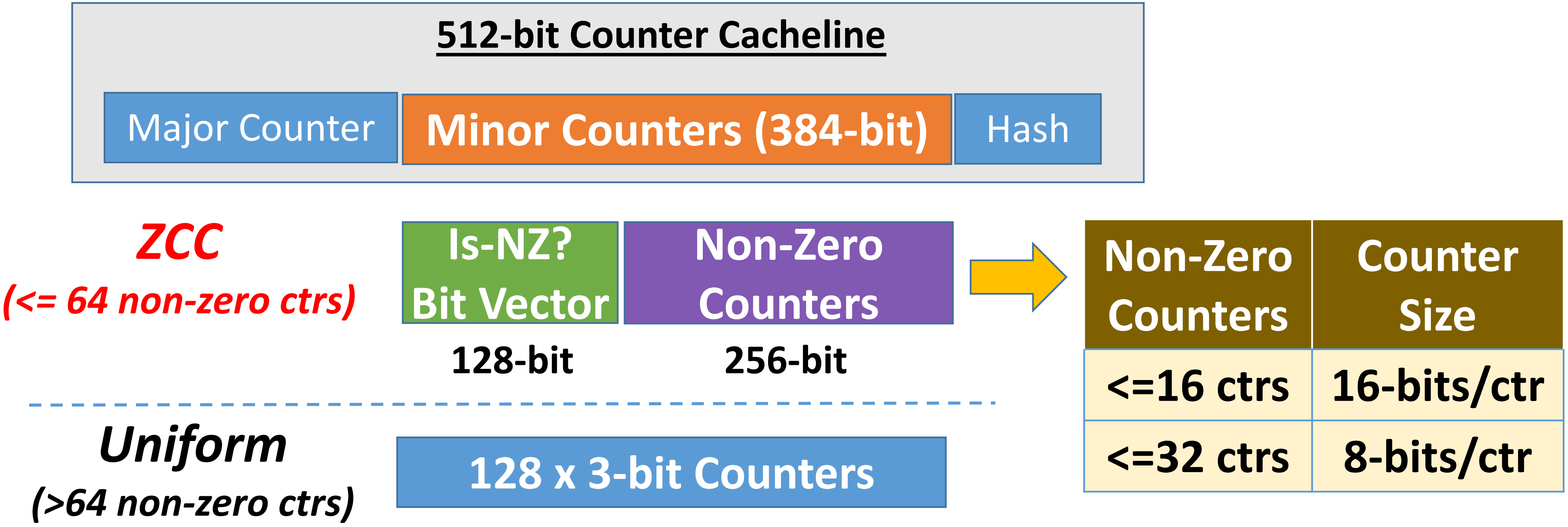
Uniform

(>64 non-zero ctrs)

128 x 3-bit Counters

Non-Zero Counters	Counter Size
<=16 ctrs	16-bits/ctr
<=32 ctrs	8-bits/ctr

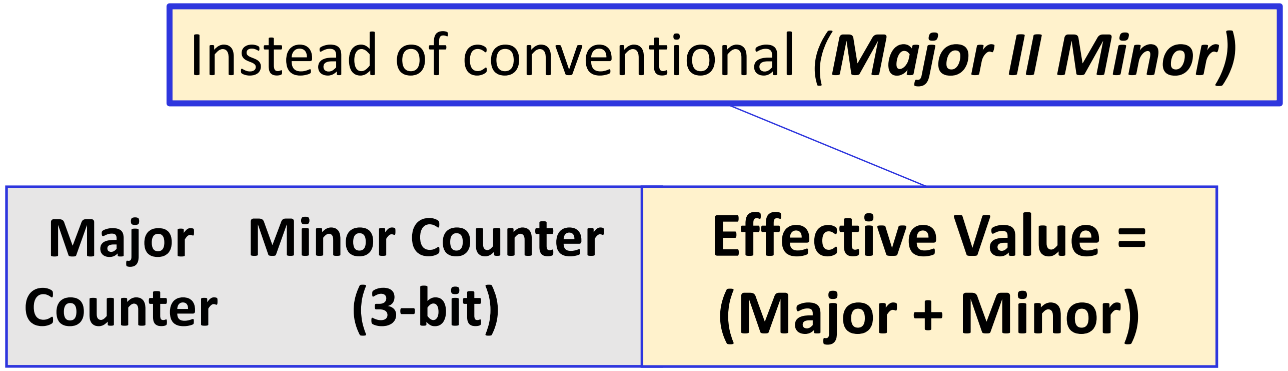
Few Counters Used: Compress Zero Counters



Non-Zero Counters	Counter Size
<=16 ctrs	16-bits/ctr
<=32 ctrs	8-bits/ctr

ZCC provides large overflow-tolerant counters, when less than 25% counters are used out of 128

Avoiding Overflows When All Counters Used



Instead of conventional *(**Major II Minor**)*

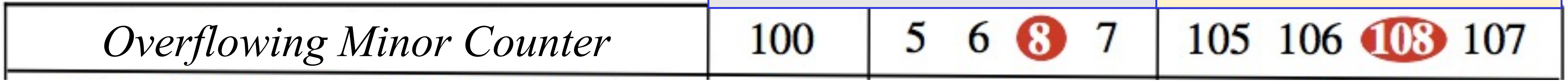
Major Counter

Minor Counter (3-bit)

Effective Value = (Major + Minor)

Avoiding Overflows When All Counters Used

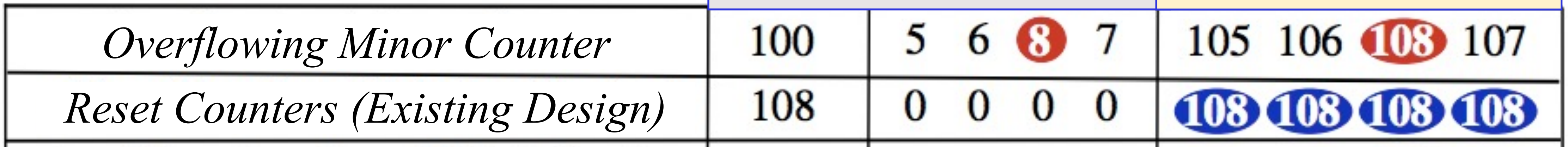
Instead of conventional ***(Major II Minor)***



	Major Counter	Minor Counter (3-bit)	Effective Value = (Major + Minor)
Overflowing Minor Counter	100	5 6 8 7	105 106 108 107

Avoiding Overflows When All Counters Used

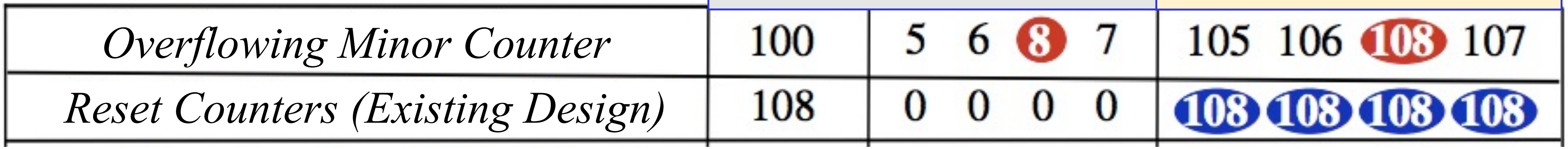
Instead of conventional *(**Major II Minor**)*



	Major Counter	Minor Counter (3-bit)				Effective Value = (Major + Minor)			
Overflowing Minor Counter	100	5	6	8	7	105	106	108	107
Reset Counters (Existing Design)	108	0	0	0	0	108	108	108	108

Avoiding Overflows When All Counters Used

Instead of conventional *(**Major II Minor**)*

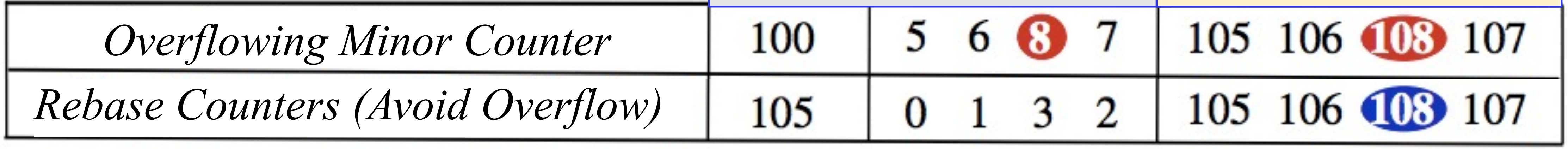


	Major Counter	Minor Counter (3-bit)	Effective Value = (Major + Minor)
Overflowing Minor Counter	100	5 6 8 7	105 106 108 107
Reset Counters (Existing Design)	108	0 0 0 0	108 108 108 108

Counters changed
– re-encryption needed

Avoiding Overflows When All Counters Used

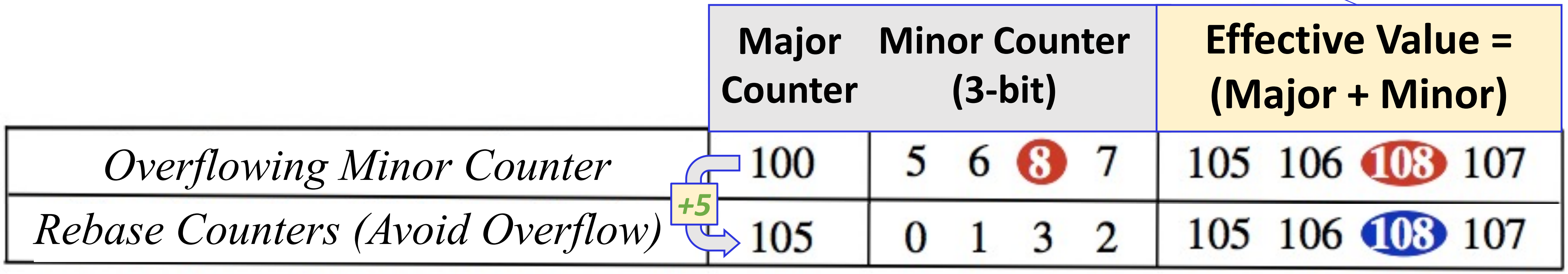
Instead of conventional *(**Major II Minor**)*



	Major Counter	**Minor Counter (3-bit)**	**Effective Value = (Major + Minor)**
Overflowing Minor Counter	100	5 6 8 7	105 106 108 107
Rebase Counters (Avoid Overflow)	105	0 1 3 2	105 106 108 107

Avoiding Overflows When All Counters Used

Instead of conventional *(**Major II Minor**)*



	Major Counter	Minor Counter (3-bit)	Effective Value = (Major + Minor)
Overflowing Minor Counter	100	5 6 8 7	105 106 108 107
Rebase Counters (Avoid Overflow)	105	0 1 3 2	105 106 108 107

Add smallest minor counter

Avoiding Overflows When All Counters Used

Instead of conventional *(**Major II Minor)***

	Major Counter	Minor Counter (3-bit)				Effective Value = (Major + Minor)			
Overflowing Minor Counter	100	5	6	8	7	105	106	108	107
Rebase Counters (Avoid Overflow)	105	0	1	3	2	105	106	108	107

+5

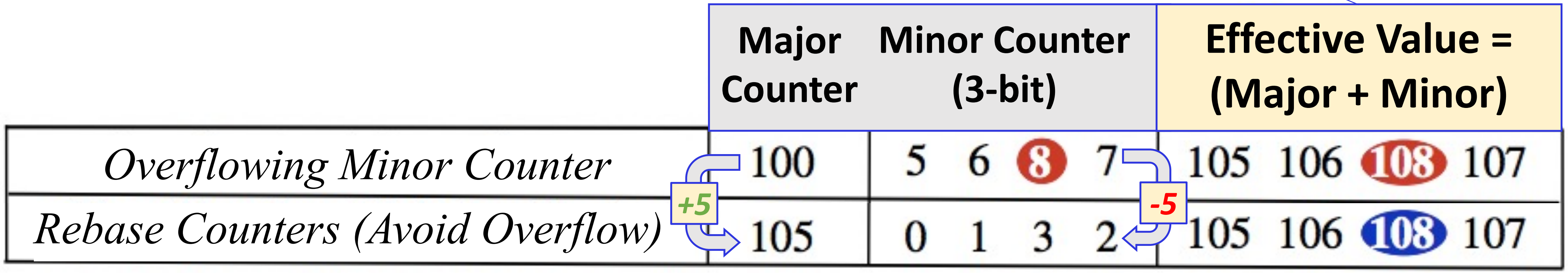
-5

Add smallest minor counter

Subtract that value from all

Avoiding Overflows When All Counters Used

Instead of conventional *(**Major II Minor**)*



	Major Counter	Minor Counter (3-bit)	Effective Value = (Major + Minor)
Overflowing Minor Counter	100	5 6 8 7	105 106 108 107
Rebase Counters (Avoid Overflow)	105	0 1 3 2	105 106 108 107

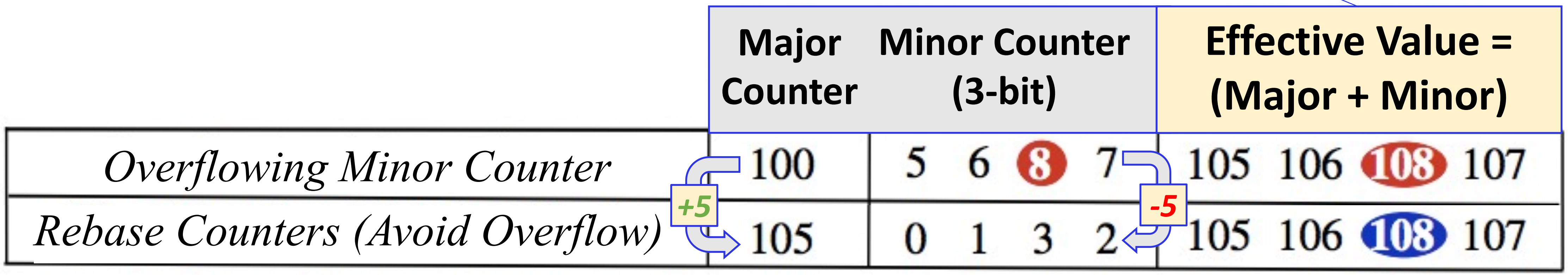
Add smallest minor counter

Subtract that value from all

No change;
No re-encryption;

Avoiding Overflows When All Counters Used

Instead of conventional ***(Major II Minor)***



	Major Counter	Minor Counter (3-bit)	Effective Value = (Major + Minor)
Overflowing Minor Counter	100	5 6 8 7	105 106 108 107
Rebase Counters (Avoid Overflow)	105	0 1 3 2	105 106 108 107

Add smallest minor counter

Subtract that value from all

No change;

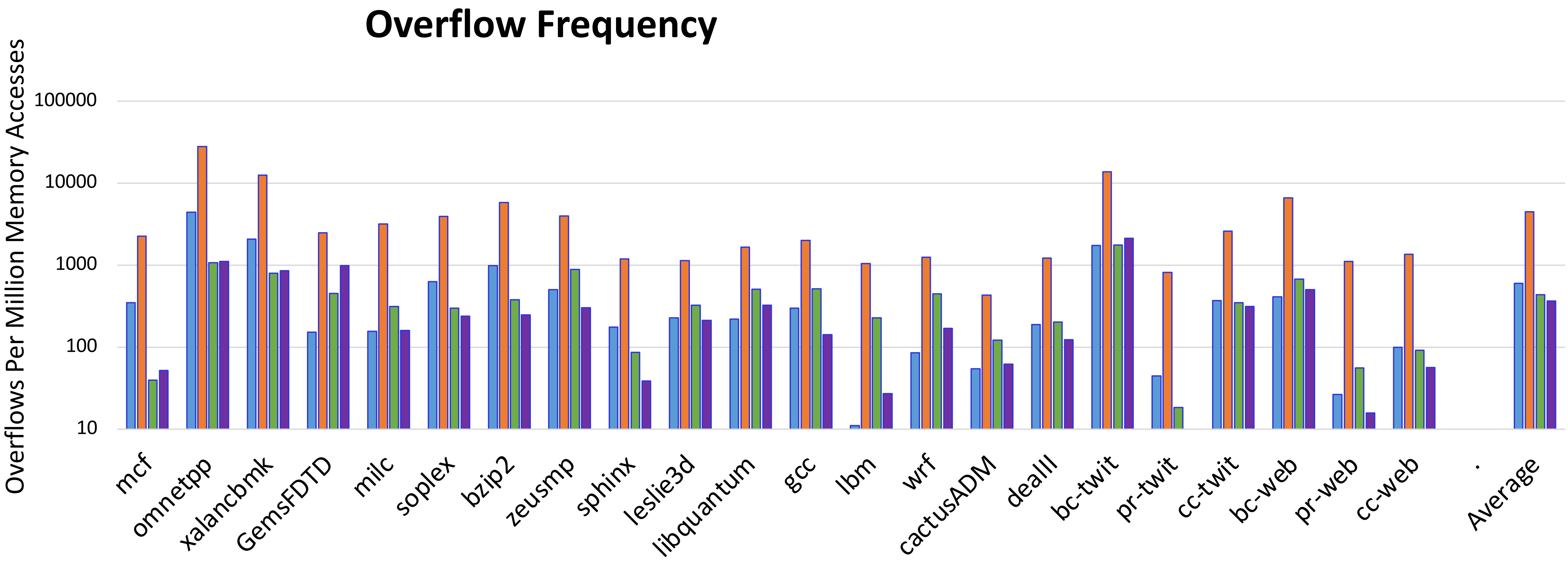
No re-encryption;

Rebasing avoids counter overflow and overheads, when all counters used

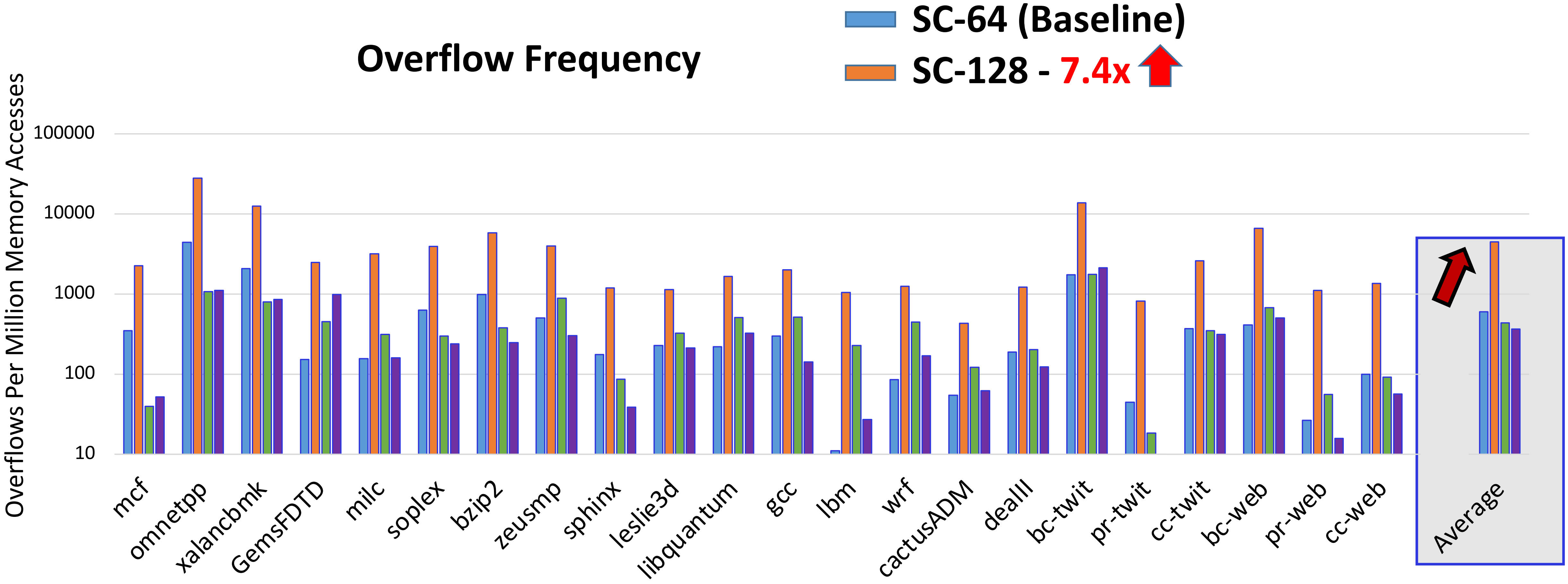
Agenda

- Introduction
- Background and Motivation
- Design
- **Results**

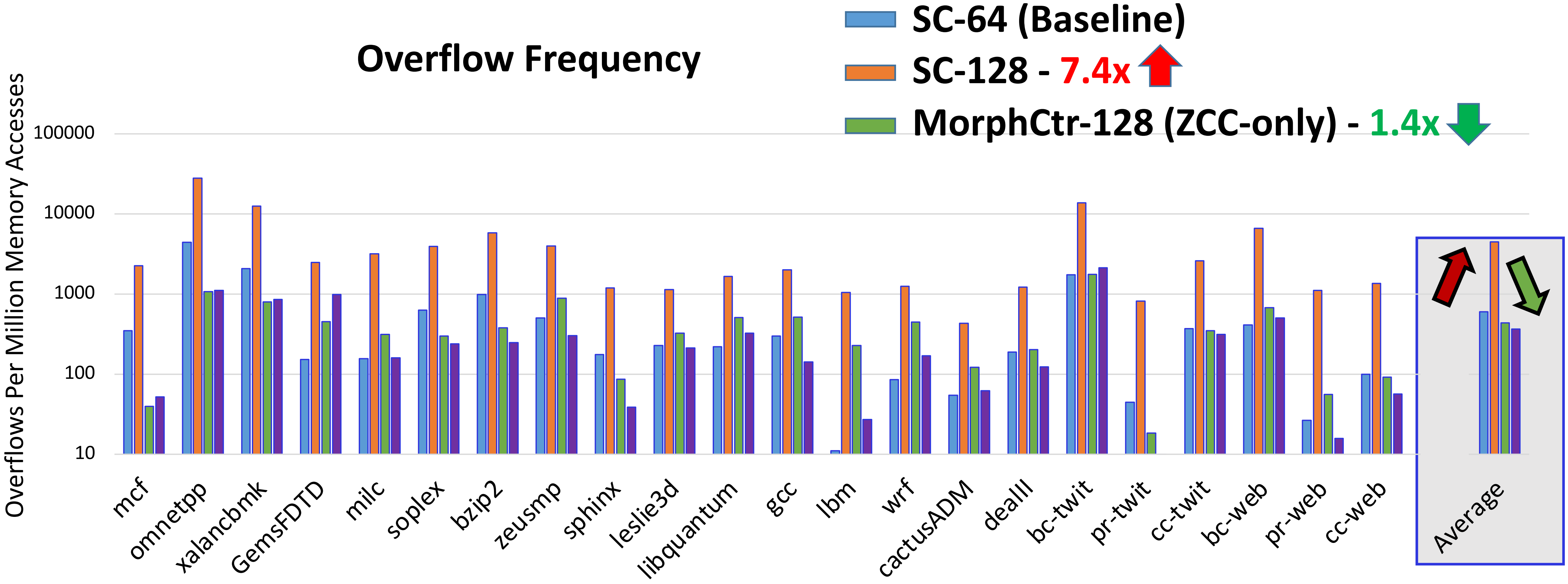
Reduction in Overflows



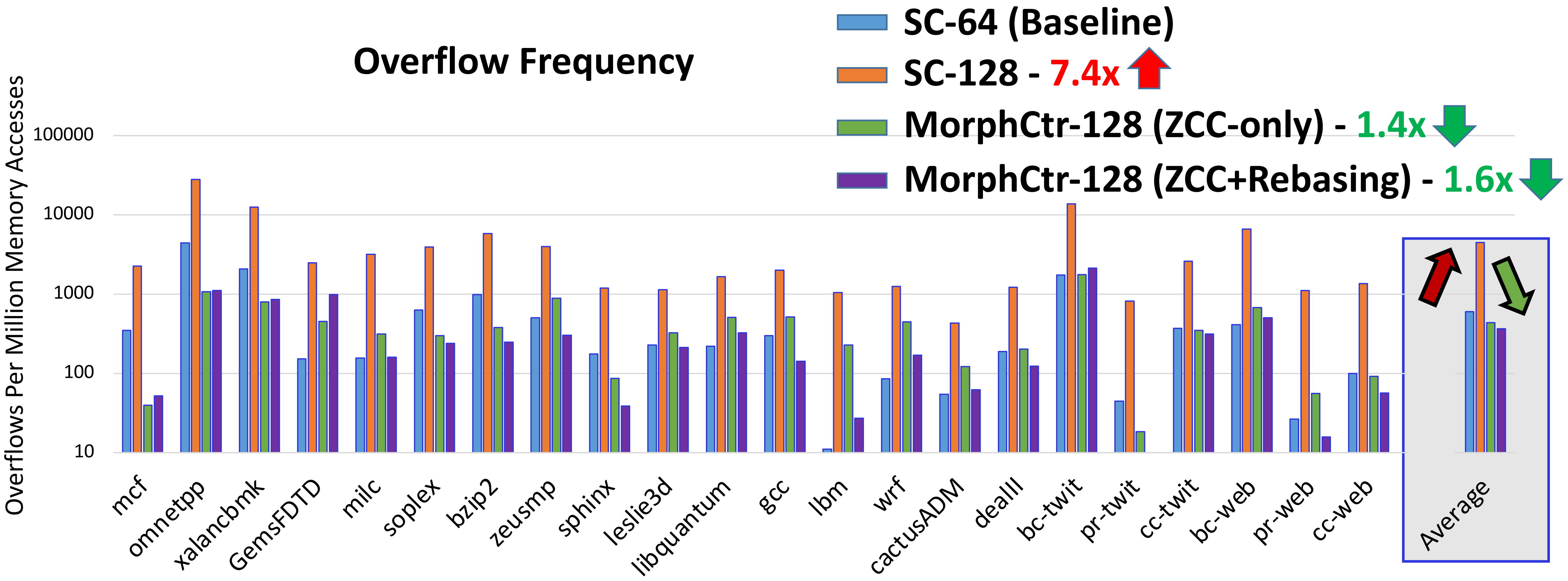
Reduction in Overflows



Reduction in Overflows



Reduction in Overflows



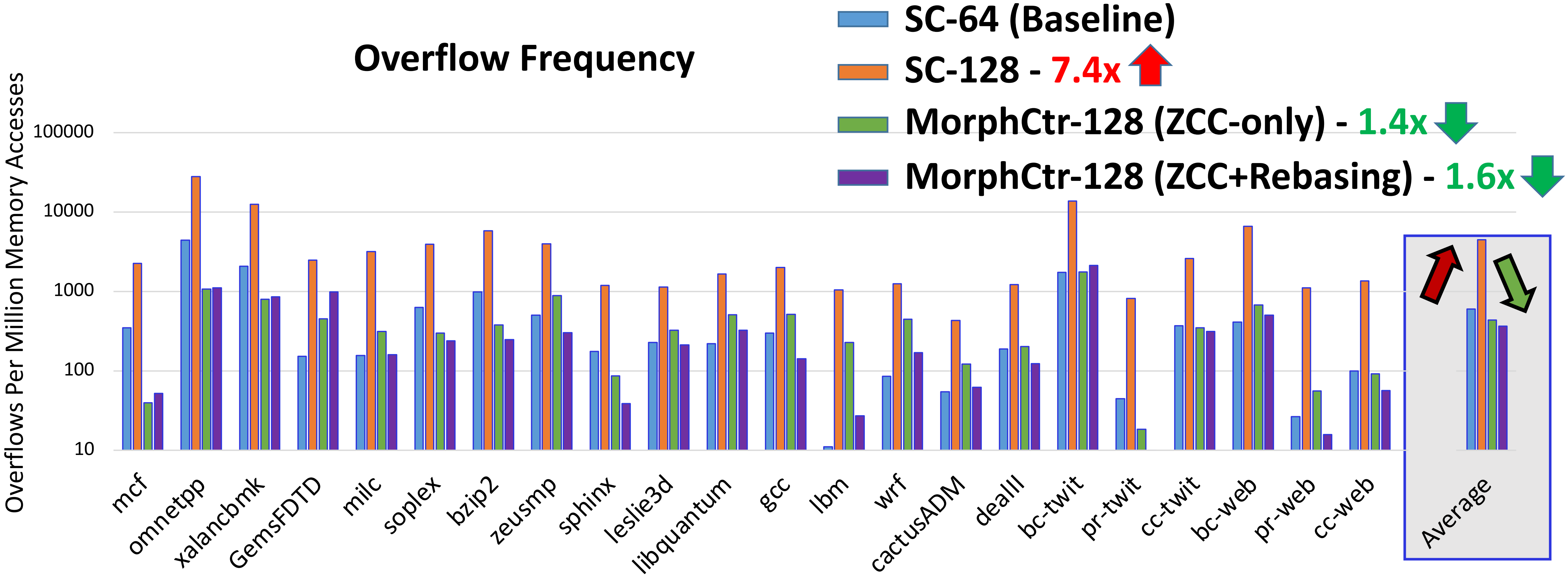
🟦 SC-64 (Baseline)

🟧 SC-128 - 7.4x ⬆

🟩 MorphCtr-128 (ZCC-only) - 1.4x ⬇

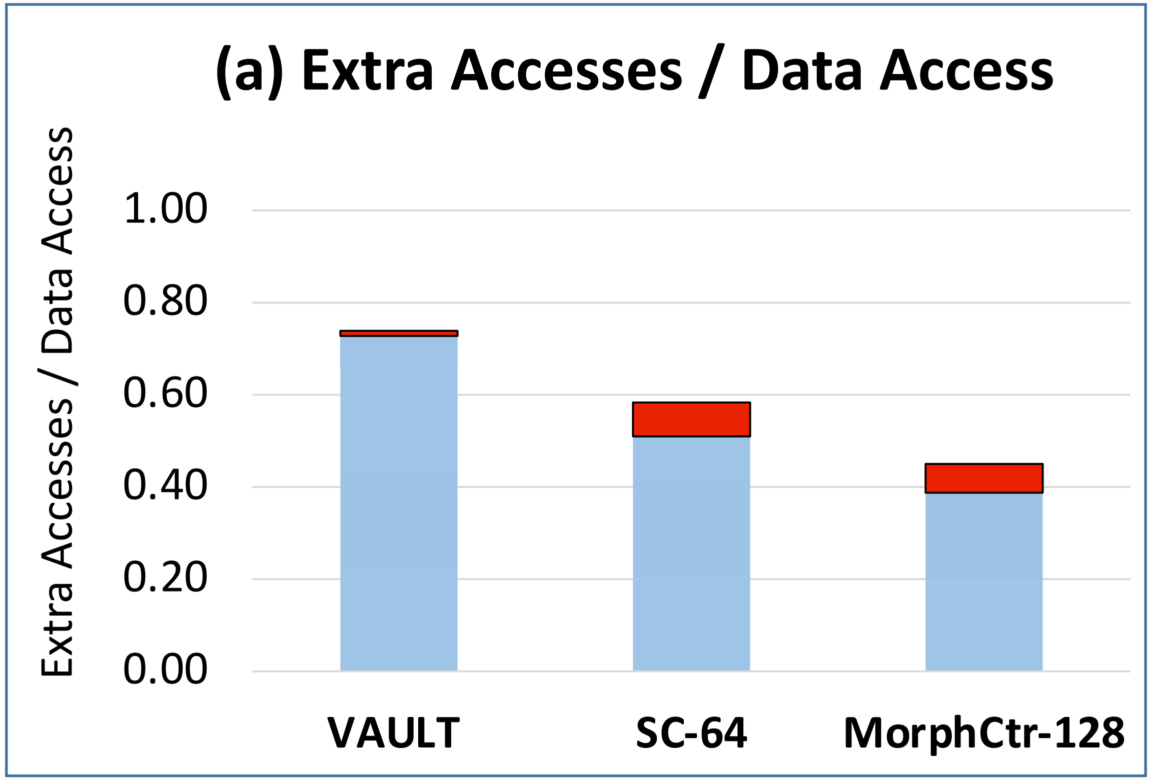
🟪 MorphCtr-128 (ZCC+Rebasing) - 1.6x ⬇

Reduction in Overflows



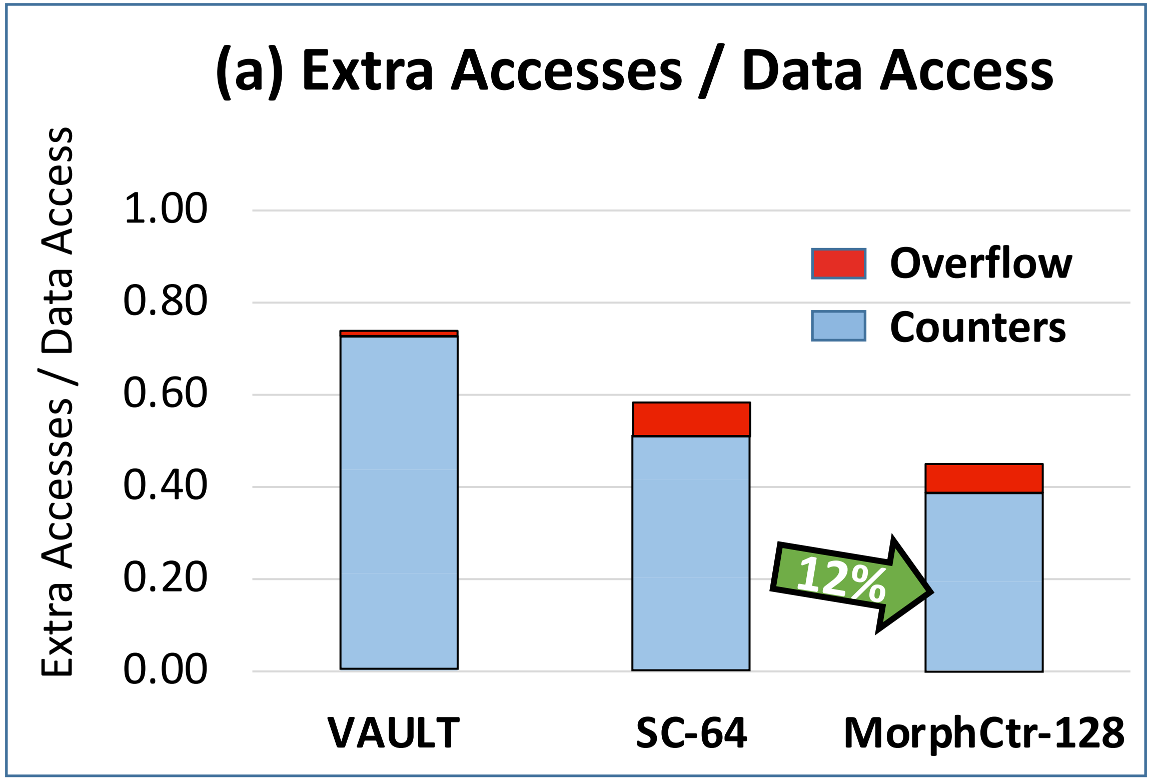
MorphCtr-128 packs 2x Counters / Cacheline, Still, 1.6x Fewer Overflows vs SC-64

Performance Benefits



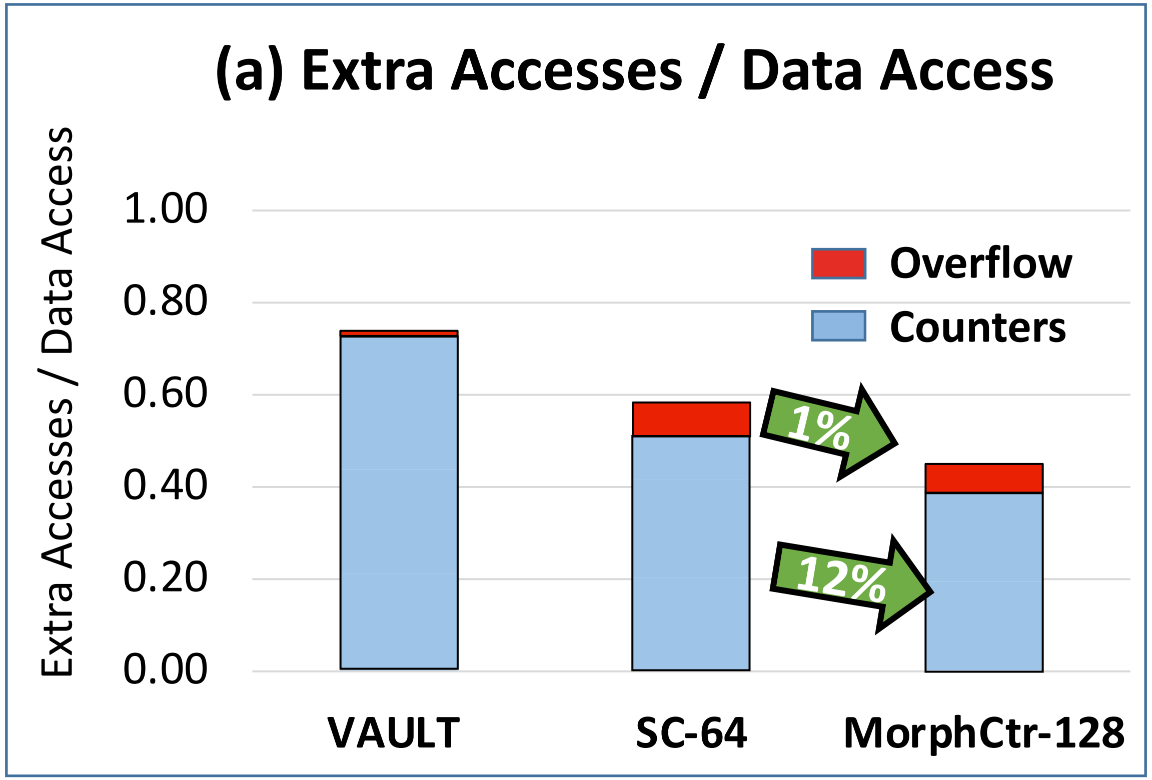
Note: 4 Cores, 8MB LLC, 16GB Secure Memory, 128KB Dedicated Counter Cache

Performance Benefits



Note: 4 Cores, 8MB LLC, 16GB Secure Memory, 128KB Dedicated Counter Cache

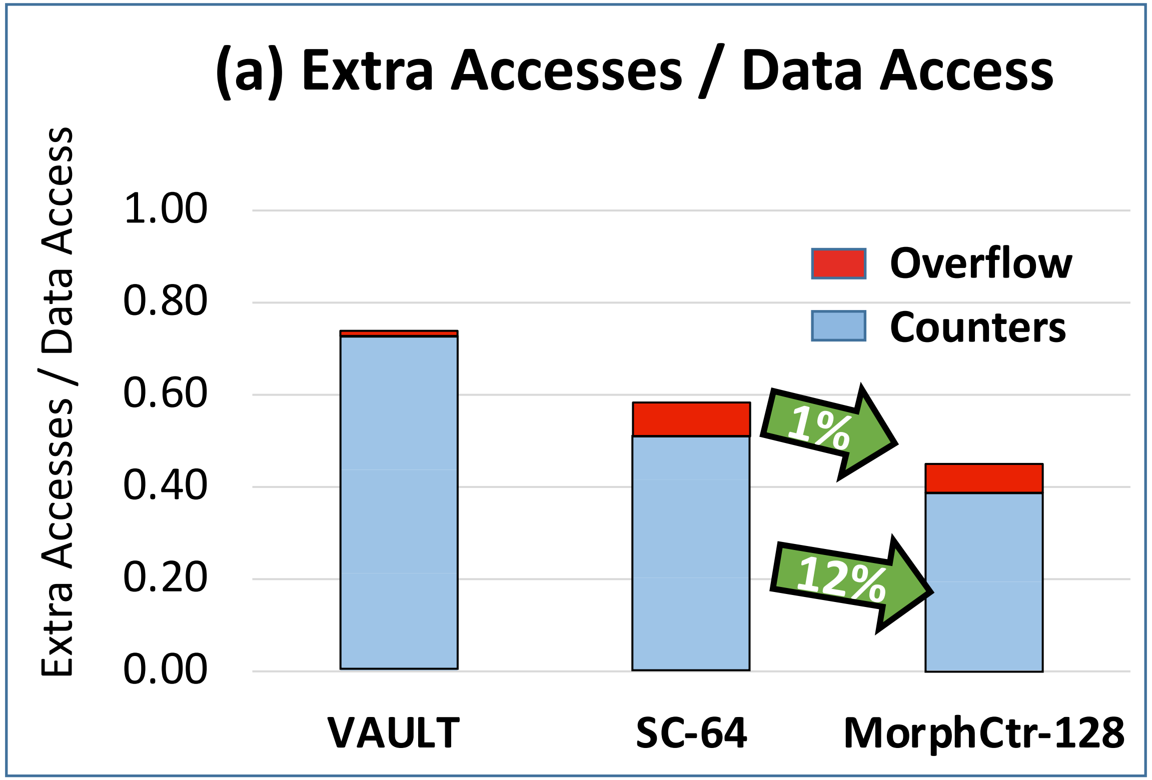
Performance Benefits

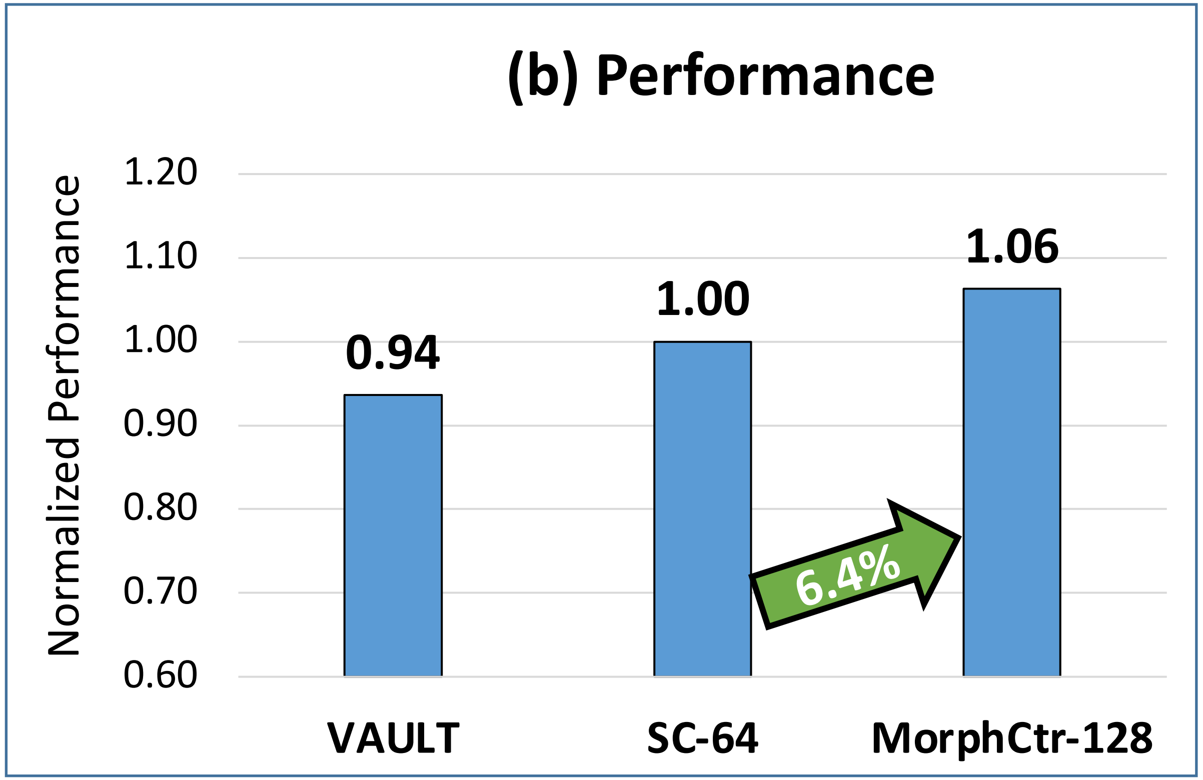


MorphCtr-128 reduces counter accesses, without a bloat in overflow updates

Note: 4 Cores, 8MB LLC, 16GB Secure Memory, 128KB Dedicated Counter Cache

Performance Benefits



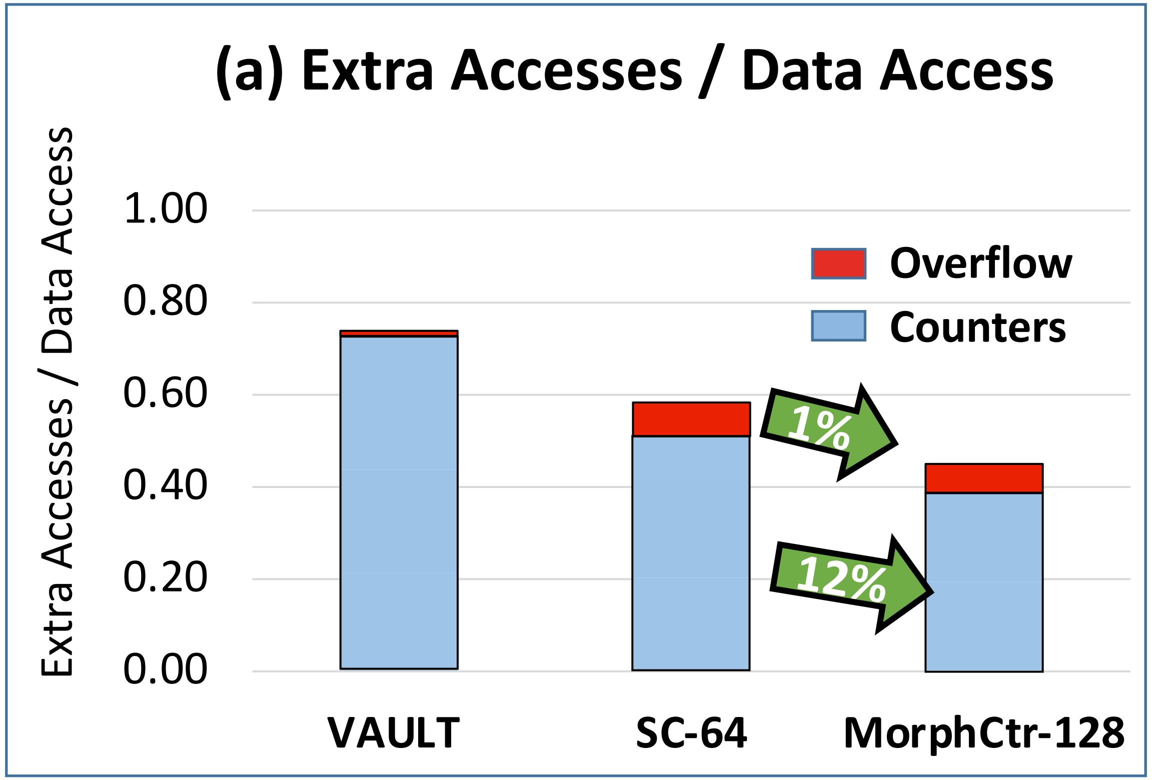


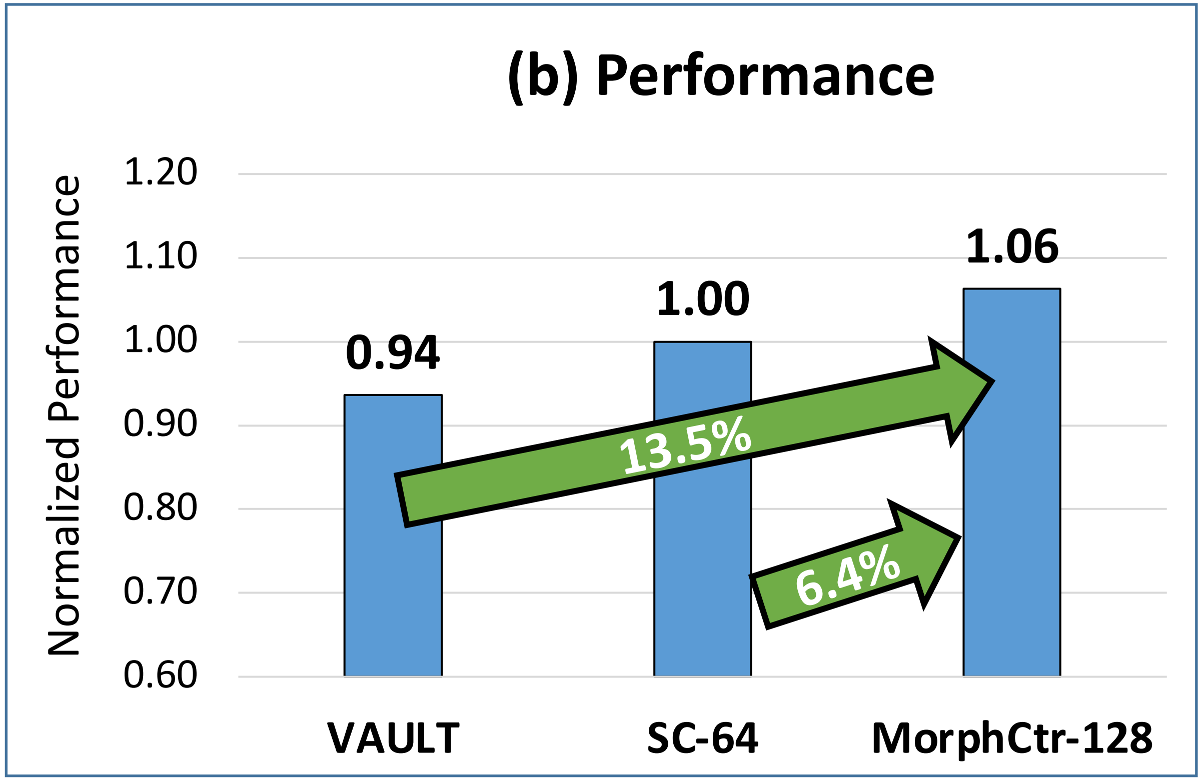
MorphCtr-128 reduces counter accesses, without a bloat in overflow updates

6.4% speedup vs Baseline, 13.5% vs state-of-the-art

Note: 4 Cores, 8MB LLC, 16GB Secure Memory, 128KB Dedicated Counter Cache

Performance Benefits





MorphCtr-128 reduces counter accesses, without a bloat in overflow updates

6.4% speedup vs Baseline, 13.5% vs state-of-the-art

Note: 4 Cores, 8MB LLC, 16GB Secure Memory, 128KB Dedicated Counter Cache

Storage Benefits

Configuration	Encryption Counter Storage
VAULT	1.6%
SC-64	1.6%
MorphCtr-128	0.8%

Storage Benefits

Configuration	Encryption Counter Storage
VAULT	1.6%
SC-64	1.6%
MorphCtr-128	0.8%

Encryption counter storage reduced by 2X,

Storage Benefits

Configuration	Encryption Counter Storage	Integrity-Tree	
		Storage	Levels Accessed (From Memory)
VAULT	1.6%	0.050%	4
SC-64	1.6%	0.025%	3
MorphCtr-128	0.8%	0.006%	2

Encryption counter storage reduced by 2X,
Integrity-tree size reduced by 4x vs Baseline, 8.5X vs VAULT

Conclusion

- ***Morphable Counters*** ➔ reducing overheads of secure memory

Conclusion

- ***Morphable Counters*** **➔** reducing overheads of secure memory
- **2x Compact & 1.6x Less Overflow-Rate** vs Split Counters

Conclusion

- ***Morphable Counters*** ➔ reducing overheads of secure memory
- **2x Compact & 1.6x Less Overflow-Rate** vs Split Counters
- **For Practically Free** ➔ Only Encoding Change, No Loss in Security

Conclusion

- ***Morphable Counters*** **➔** reducing overheads of secure memory
- **2x Compact & 1.6x Less Overflow-Rate** vs Split Counters
- **For Practically Free ➔** Only Encoding Change, No Loss in Security
- **Closed 1/3rd gap between State-of-the-Art & Non-Secure**
  - 13.5% Speedup with 8.5x Smaller Integrity-Tree than VAULT

Conclusion

- ***Morphable Counters*** **➔** reducing overheads of secure memory
- **2x Compact & 1.6x Less Overflow-Rate** vs Split Counters
- **For Practically Free ➔** Only Encoding Change, No Loss in Security
- **Closed 1/3rd gap between State-of-the-Art & Non-Secure**
  - 13.5% Speedup with 8.5x Smaller Integrity-Tree than VAULT

Thank You! ***Questions?***