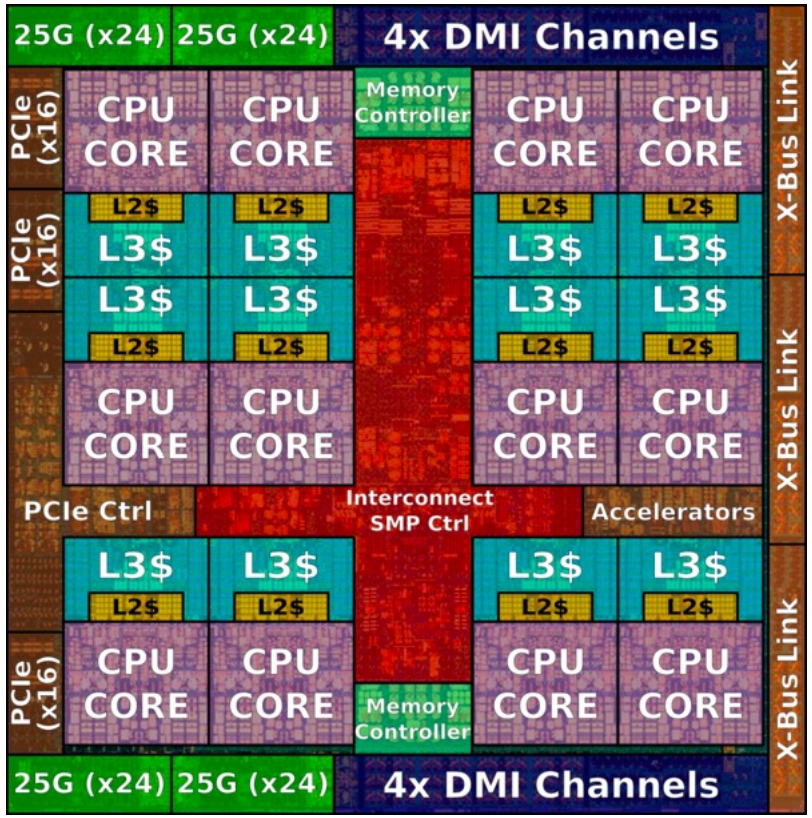
Touché : Towards Ideal and Efficient Cache Compression By Mitigating Tag Area Overheads

Seokin Hong, Bulent Abali, Alper Buyuktosunoglu, Michael B. Healy, Prashant J. Nair

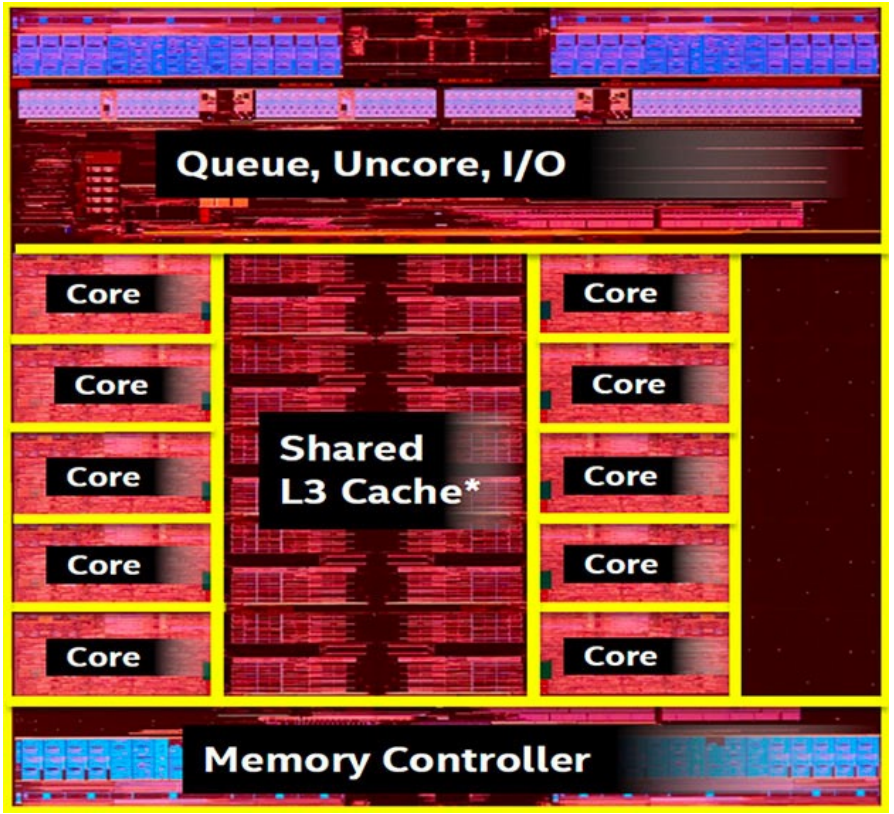
KNU IBM Research UBC

Last-Level Cache: Why Size Matters?

Last-Level Cache (LLC) is shared, occupies significant chip area



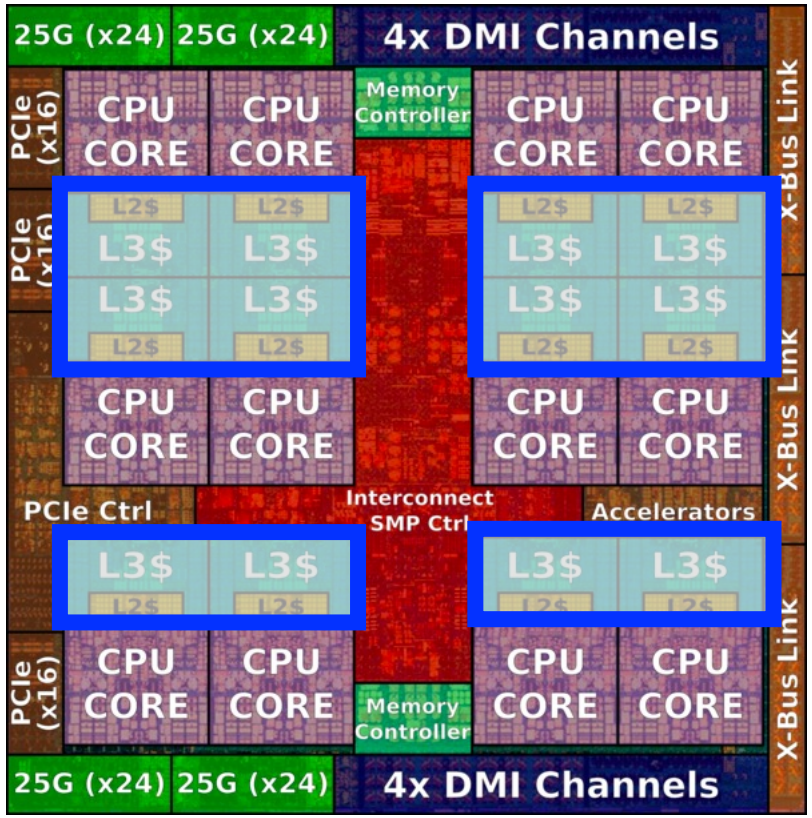
IBM Power 9



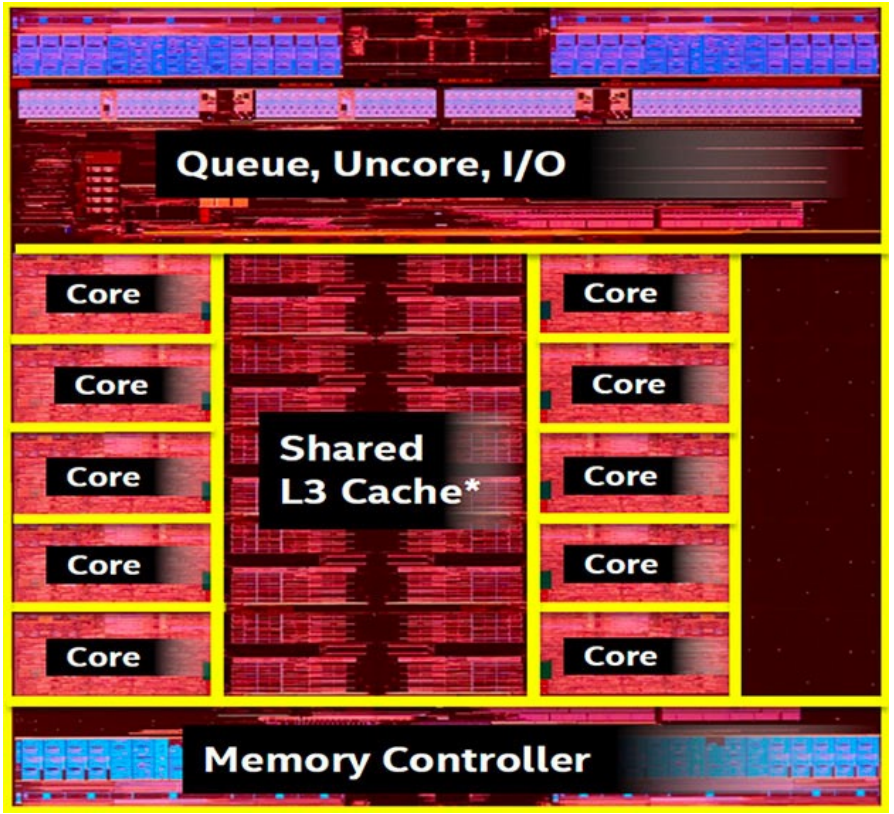
Intel i7

Last-Level Cache: Why Size Matters?

Last-Level Cache (LLC) is shared, occupies significant chip area



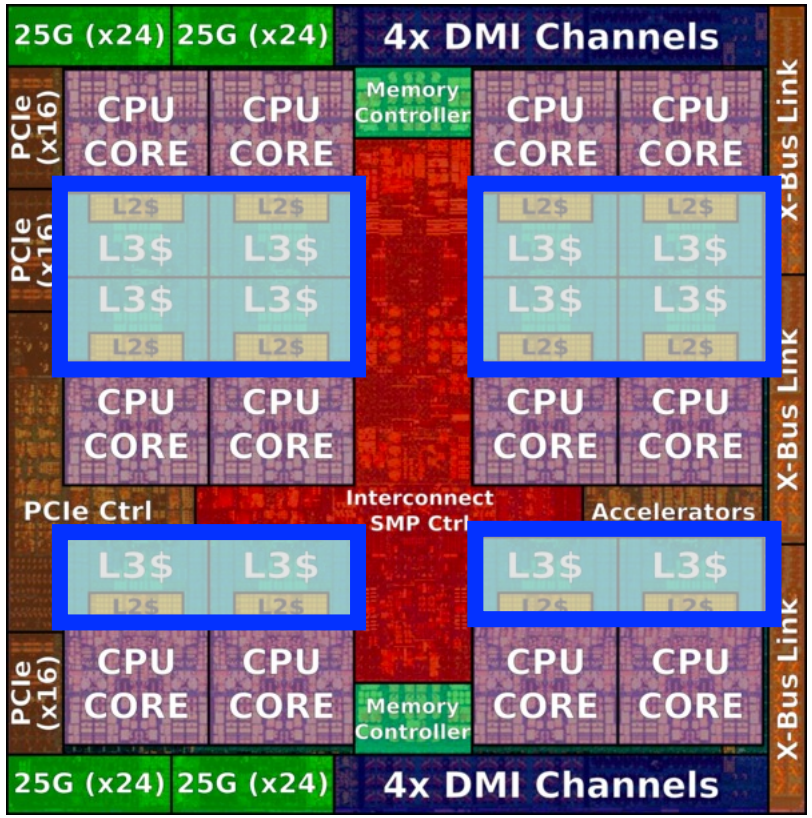
IBM Power 9



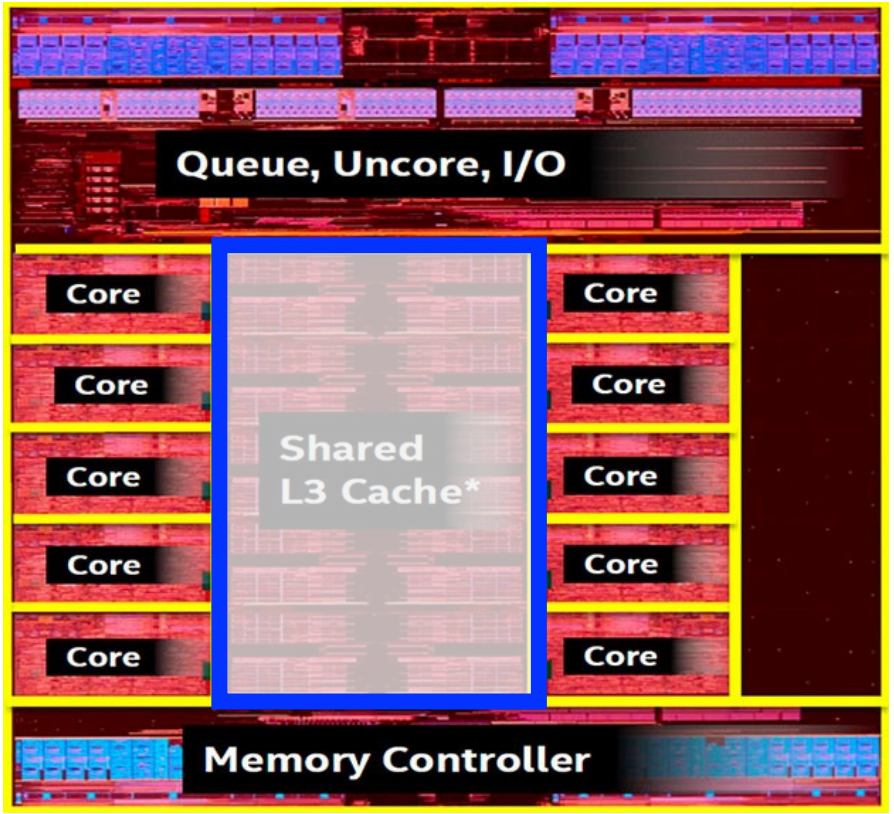
Intel i7

Last-Level Cache: Why Size Matters?

Last-Level Cache (LLC) is shared, occupies significant chip area



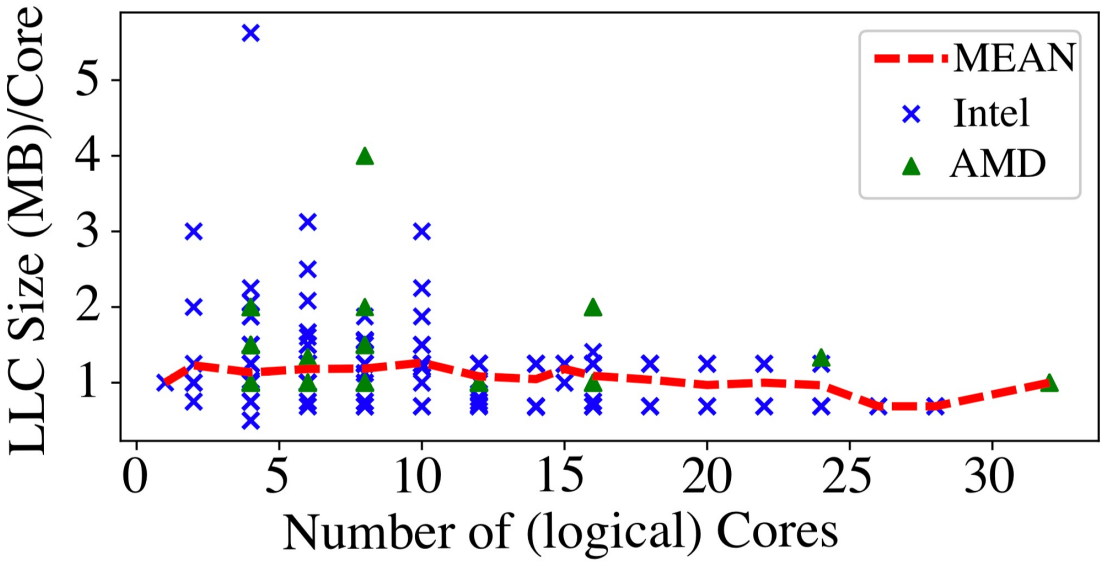
IBM Power 9



Intel i7

Last-Level Cache: Why Size Matters?

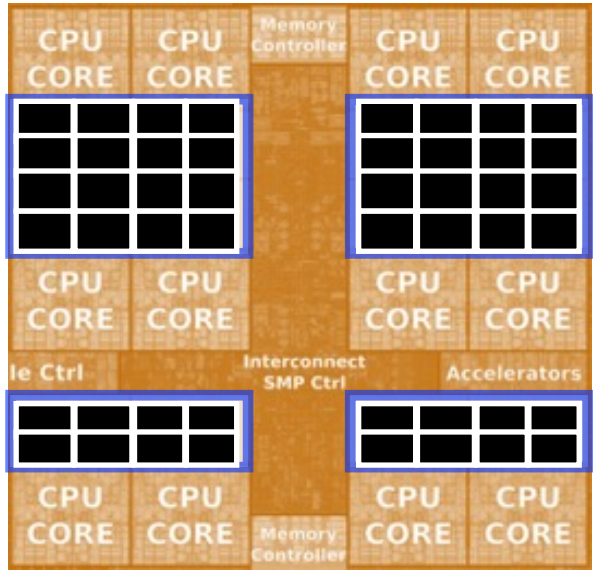
Last-Level Cache (LLC) is shared, occupies significant chip area



LLC Capacity/Core ➡ Stagnant

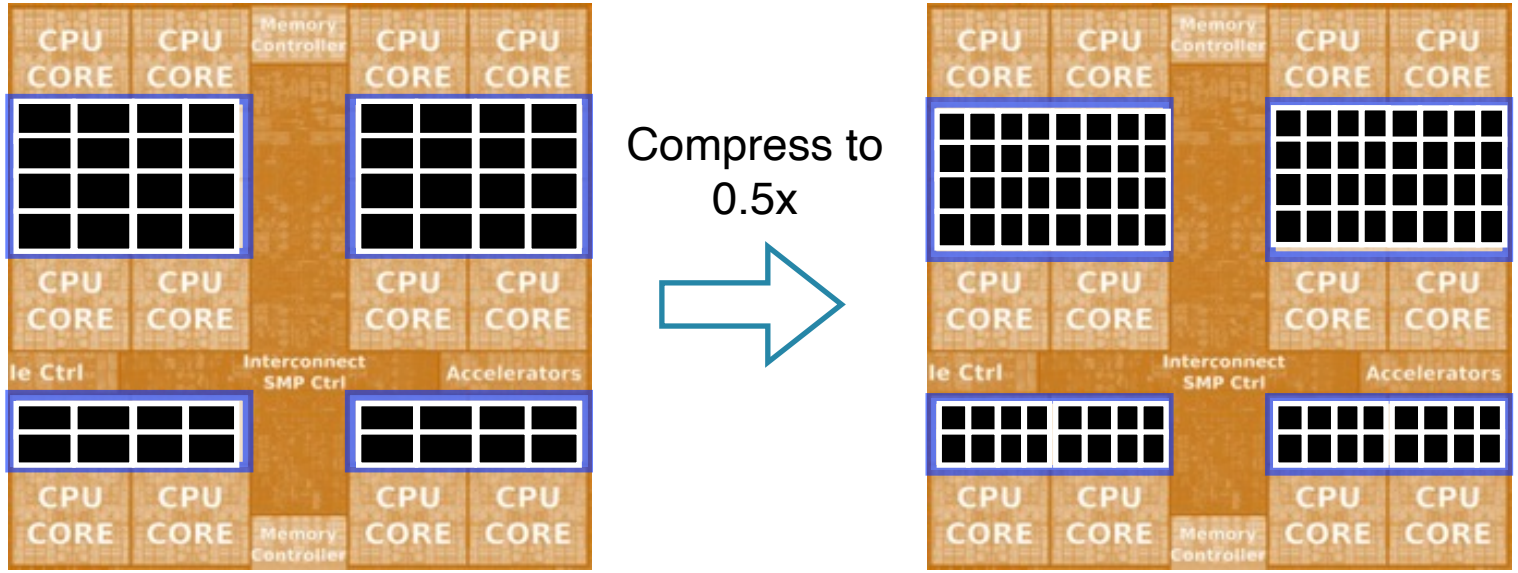
Bigger Applications

Cache Compression Can Save The Day!



With data compression, we can improve the effective LLC Capacity

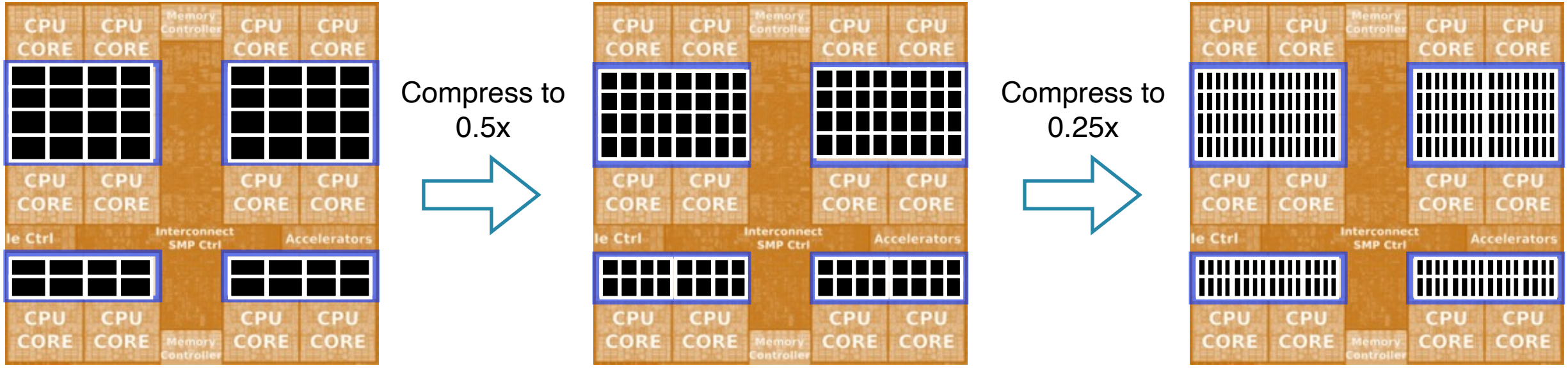
Cache Compression Can Save The Day!



2x effective capacity

With data compression, we can improve the effective LLC Capacity

Cache Compression Can Save The Day!



2x effective capacity

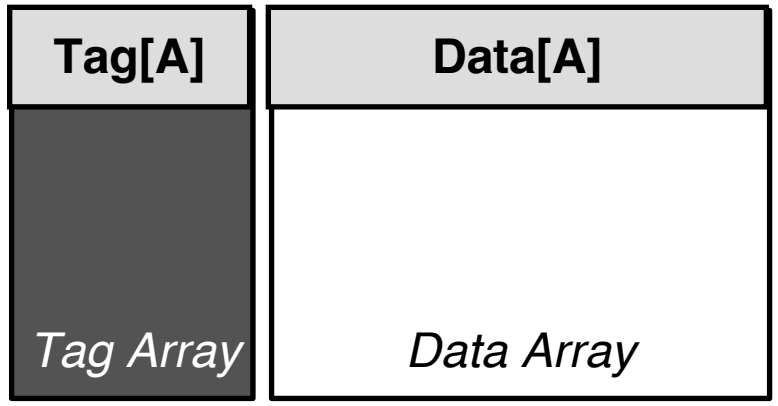
4x effective capacity

With data compression, we can improve the effective LLC Capacity

Cache Compression Can Save The Day!

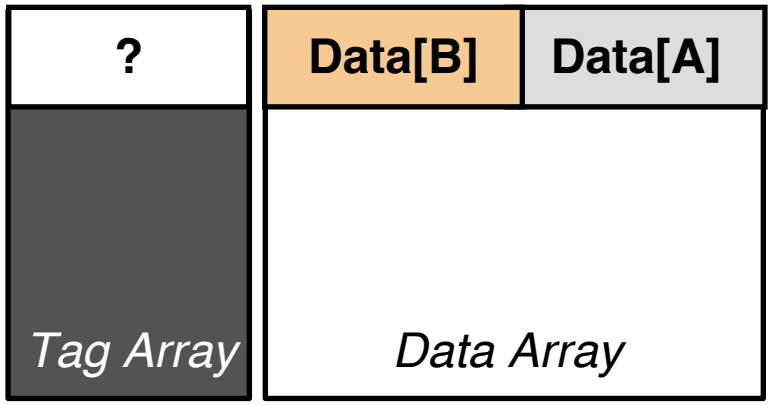
BUT, there is a catch!

Cache Compression: Tradeoffs



Uncompressed Cache

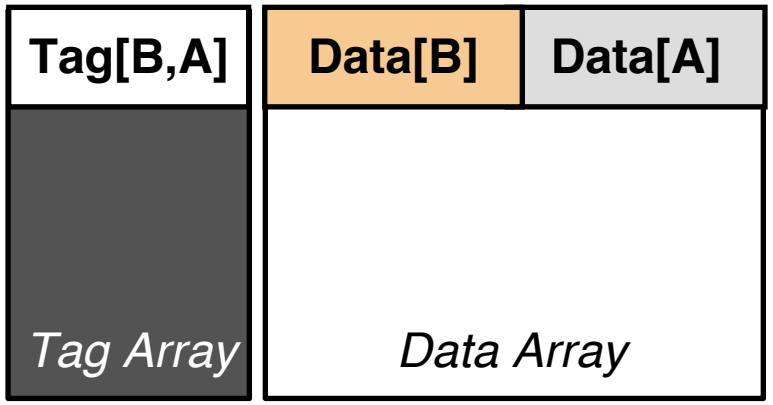
Cache Compression: Tradeoffs



Compressed Cache

Cache Compression: Tradeoffs

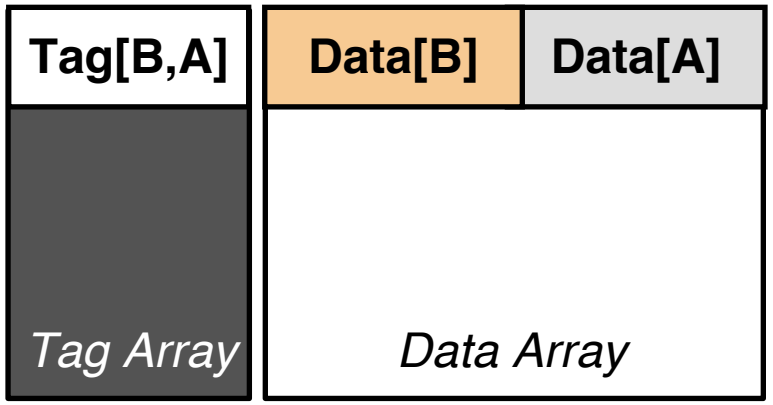
IDEAL COMPRESSION



No Additional Area Overheads

Cache Compression: Tradeoffs

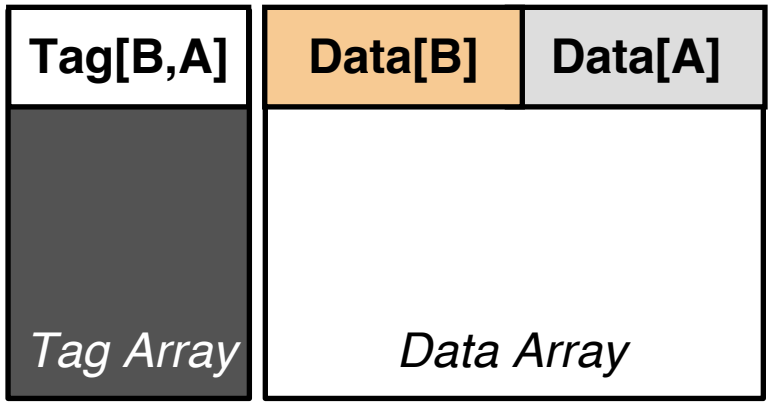
IDEAL COMPRESSION



No Additional Area Overheads

Cache Compression: Tradeoffs

IDEAL COMPRESSION

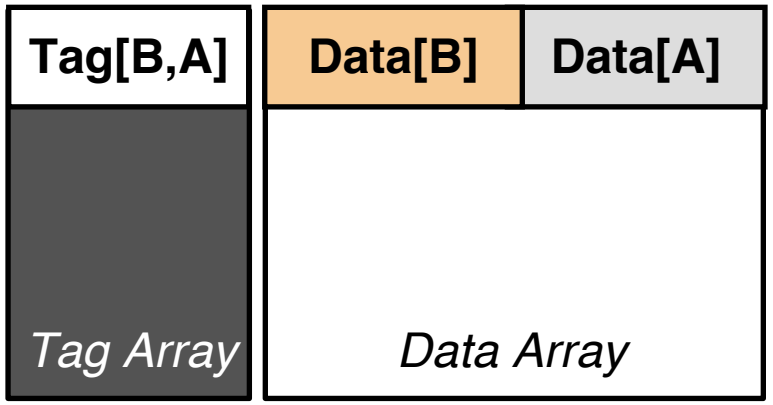


No Additional Area Overheads

PRACTICAL COMPRESSION

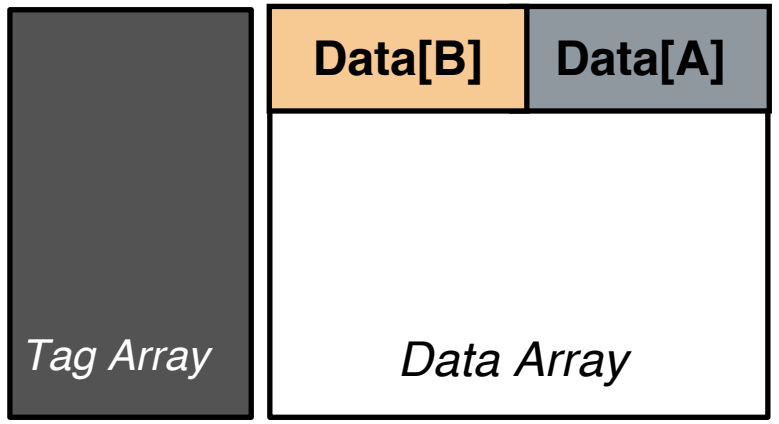
Cache Compression: Tradeoffs

IDEAL COMPRESSION



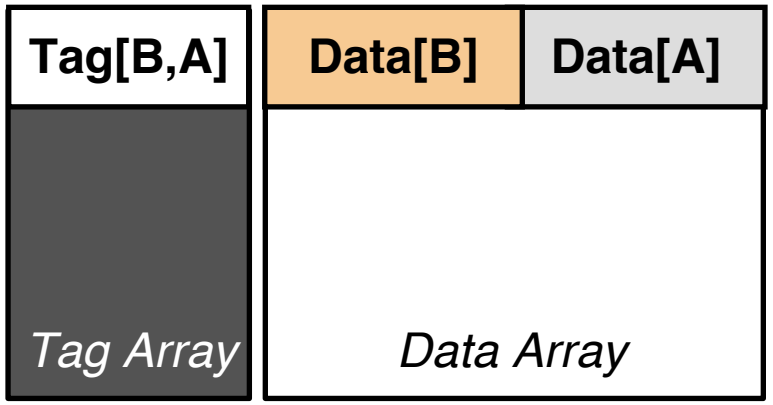
No Additional Area Overheads

PRACTICAL COMPRESSION



Cache Compression: Tradeoffs

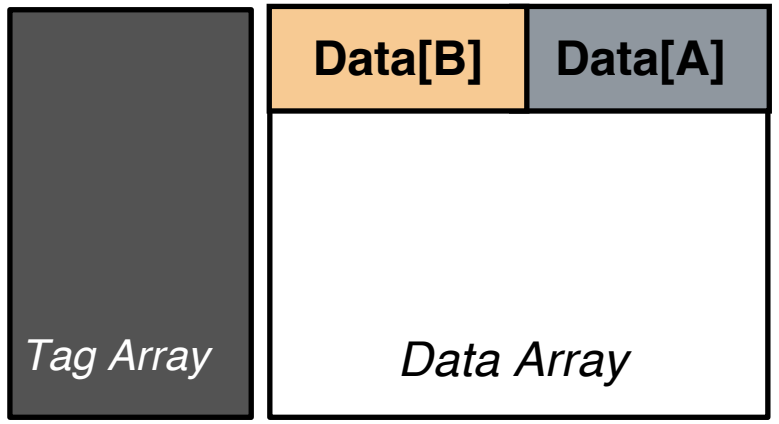
IDEAL COMPRESSION



No Additional Area Overheads

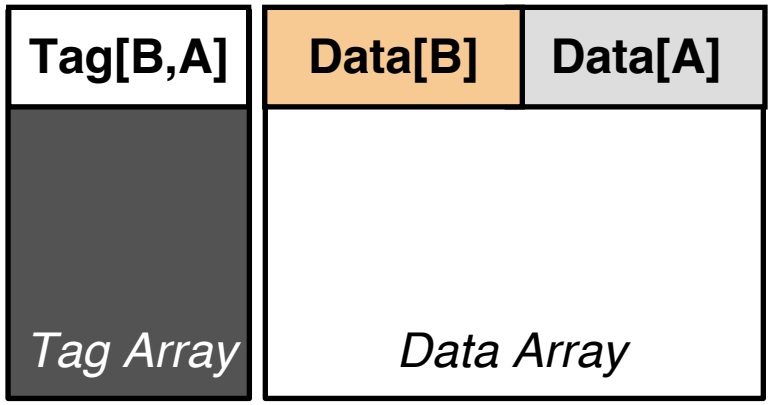
PRACTICAL COMPRESSION

Adding additional tag array



Cache Compression: Tradeoffs

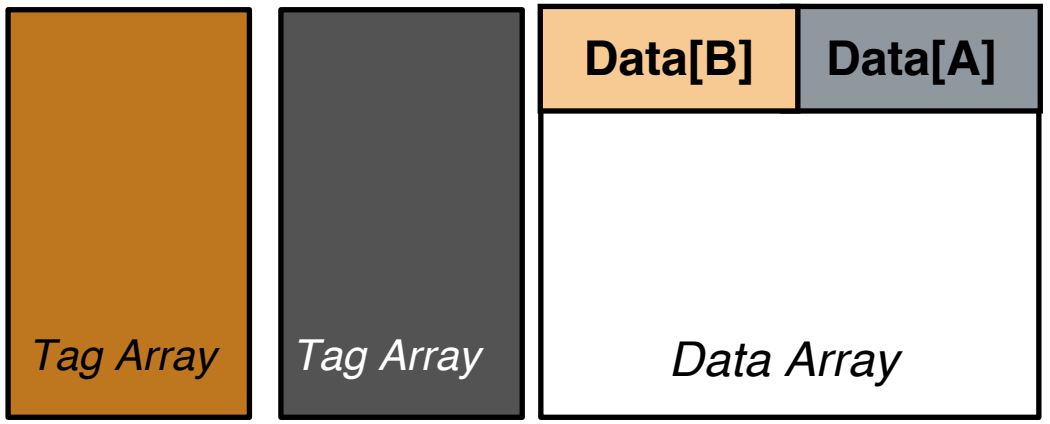
IDEAL COMPRESSION



No Additional Area Overheads

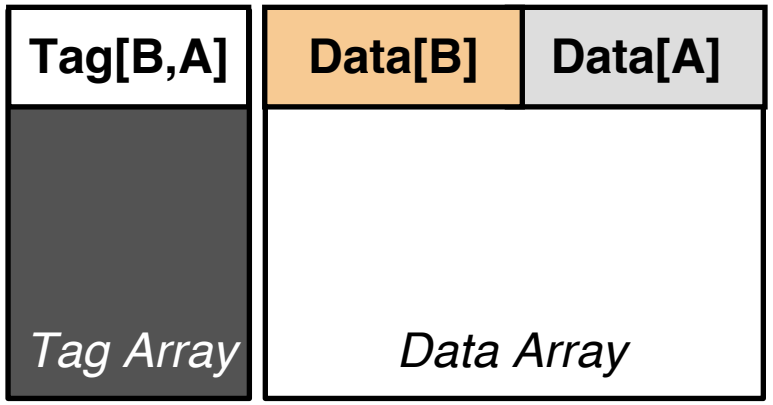
PRACTICAL COMPRESSION

Adding additional tag array



Cache Compression: Tradeoffs

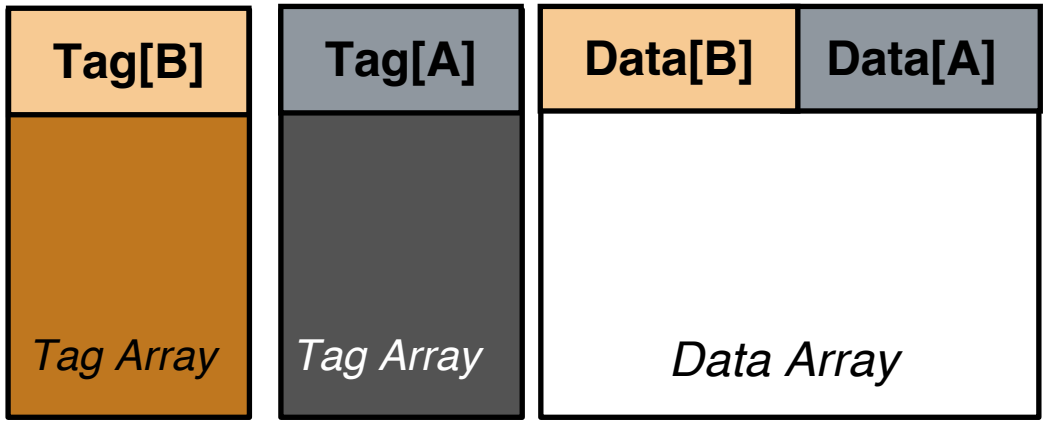
IDEAL COMPRESSION



No Additional Area Overheads

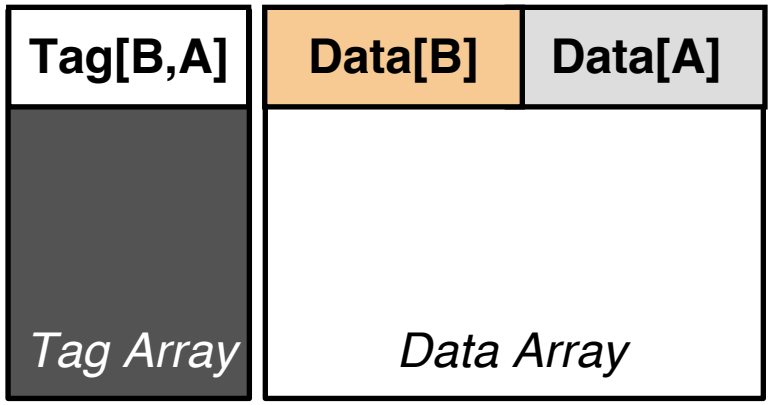
PRACTICAL COMPRESSION

Adding additional tag array



Cache Compression: Tradeoffs

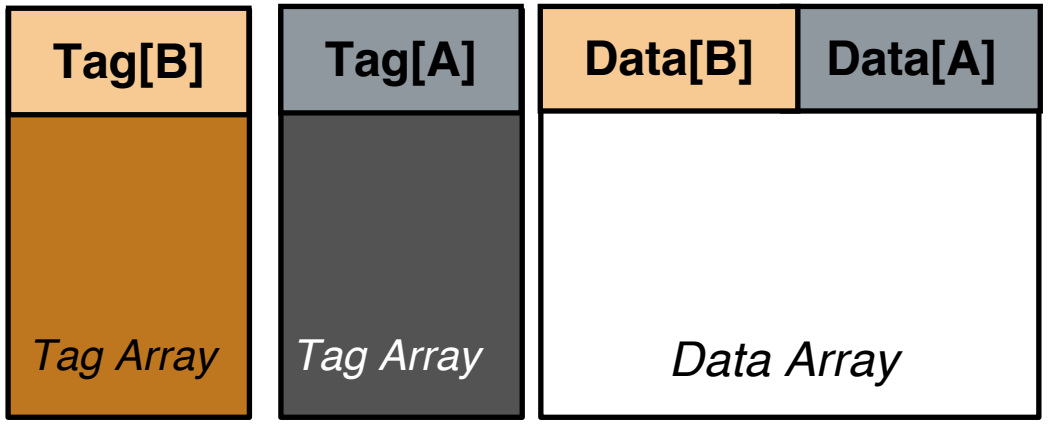
IDEAL COMPRESSION



No Additional Area Overheads

PRACTICAL COMPRESSION

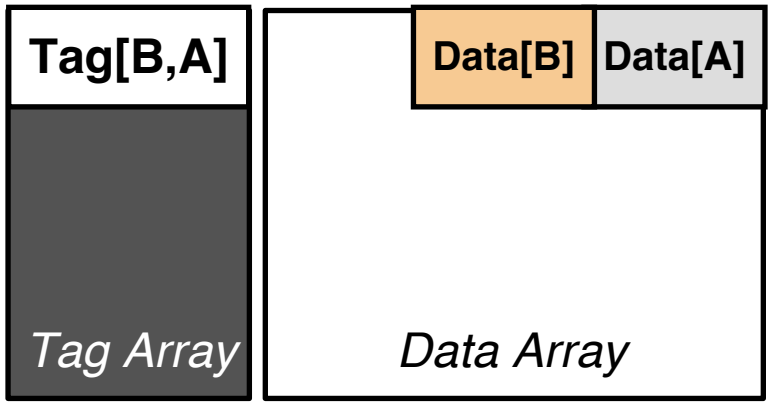
Adding additional tag array



2x Tag Area

Cache Compression: Tradeoffs

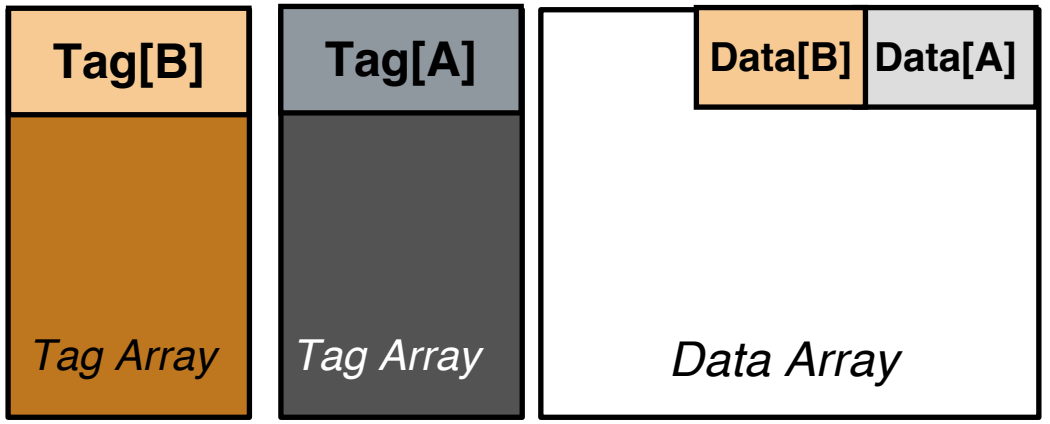
IDEAL COMPRESSION



No Additional Area Overheads

PRACTICAL COMPRESSION

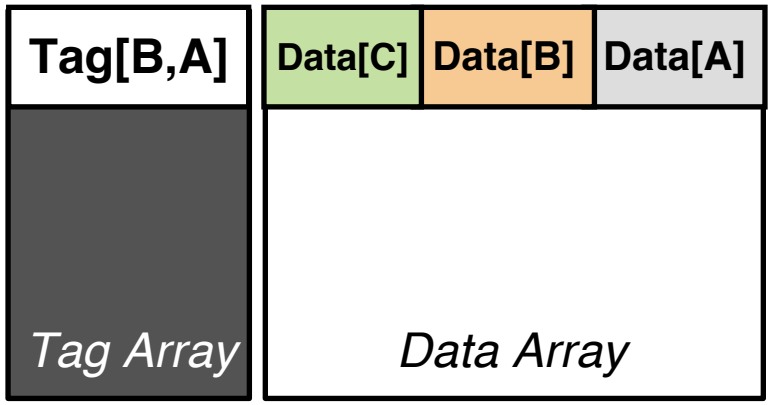
Adding additional tag array



2x Tag Area

Cache Compression: Tradeoffs

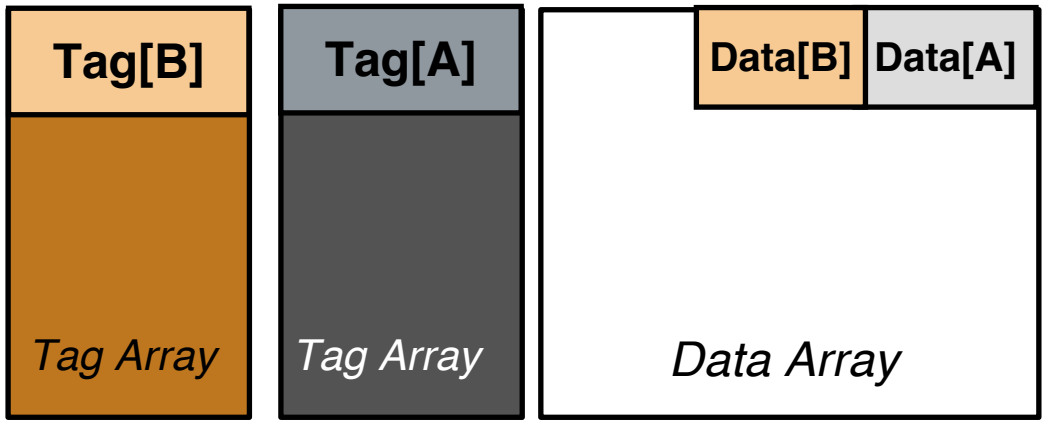
IDEAL COMPRESSION



No Additional Area Overheads

PRACTICAL COMPRESSION

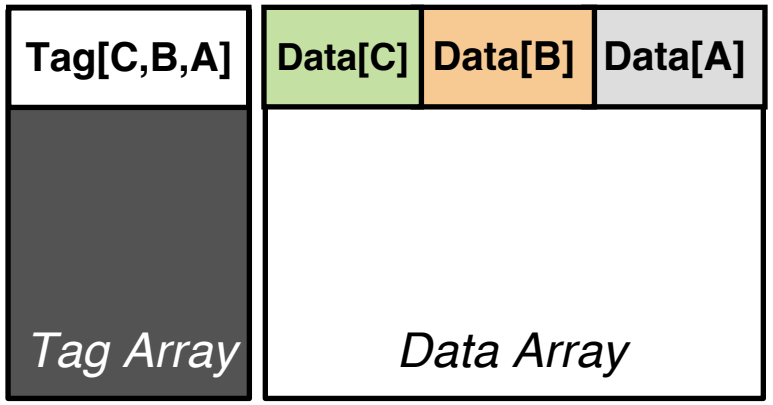
Adding additional tag array



2x Tag Area

Cache Compression: Tradeoffs

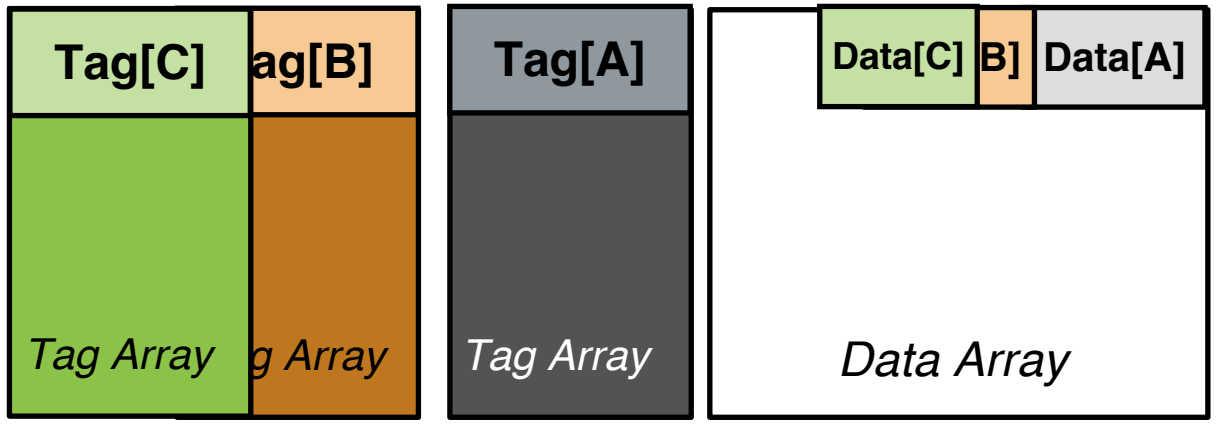
IDEAL COMPRESSION



No Additional Area Overheads

PRACTICAL COMPRESSION

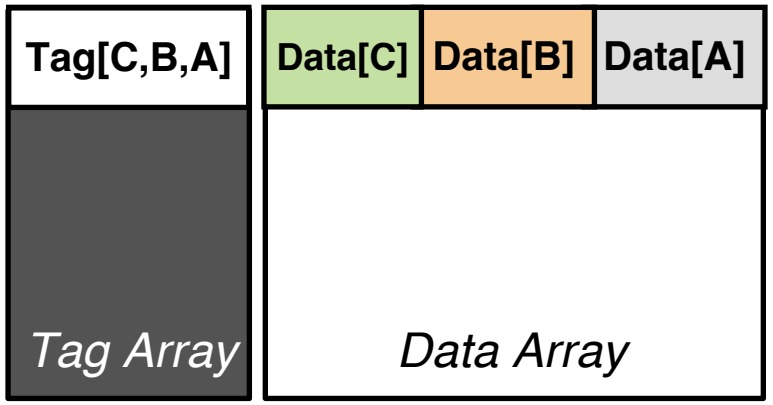
Adding additional tag array



2x Tag Area

Cache Compression: Tradeoffs

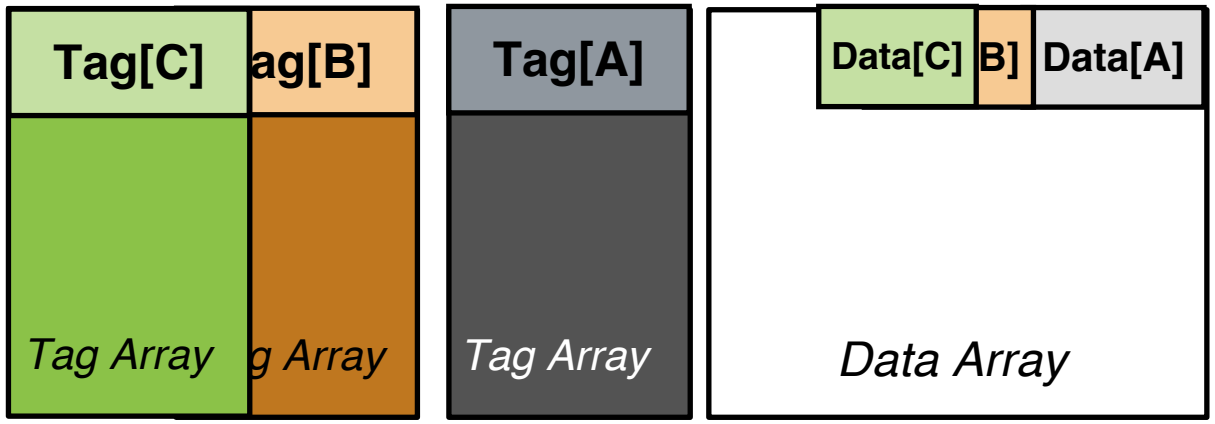
IDEAL COMPRESSION



No Additional Area Overheads

PRACTICAL COMPRESSION

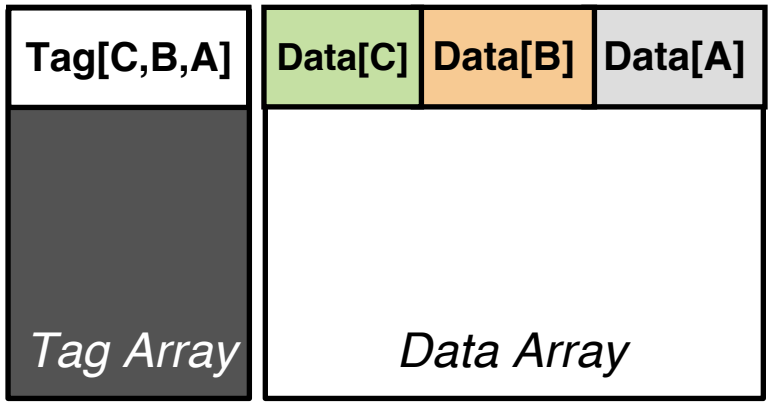
Adding additional tag array



3x Tag Area

Cache Compression: Tradeoffs

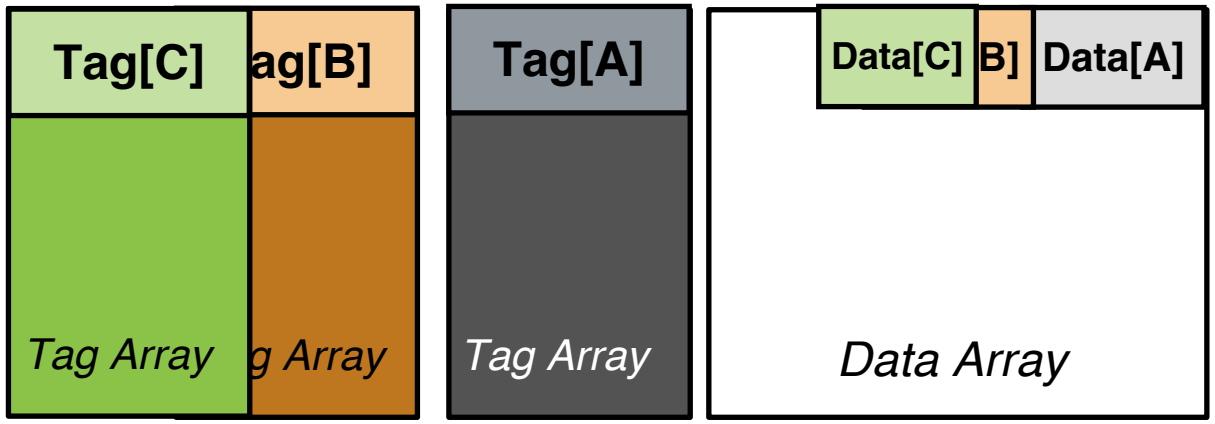
IDEAL COMPRESSION



No Additional Area Overheads

PRACTICAL COMPRESSION

Adding additional tag array

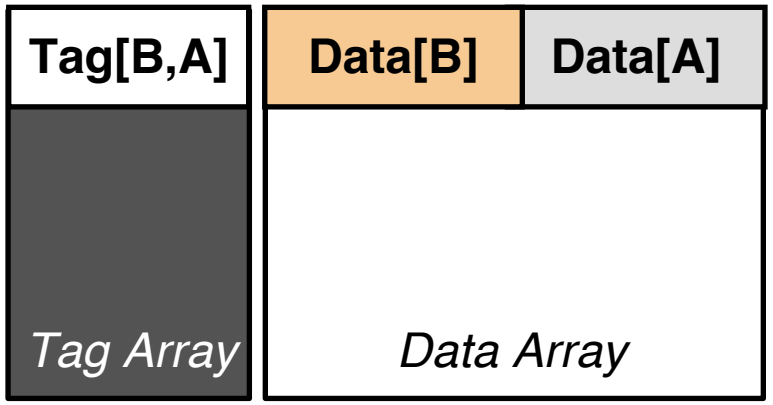


3x Tag Area

Difficult to decide the number of tag arrays at design time!

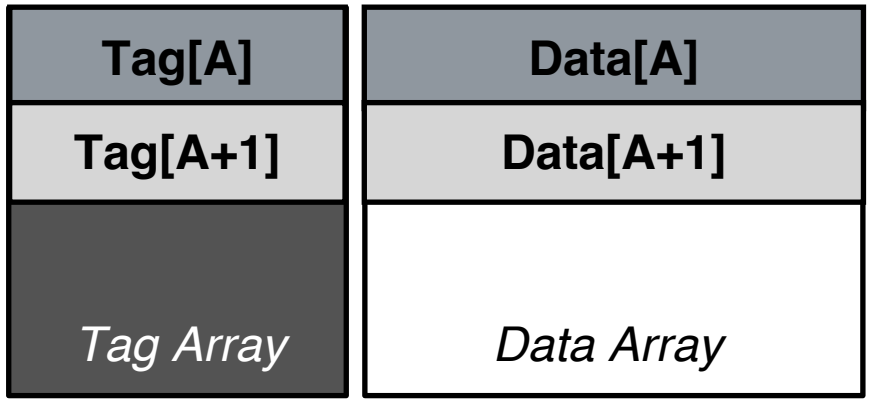
Cache Compression: Tradeoffs

IDEAL COMPRESSION



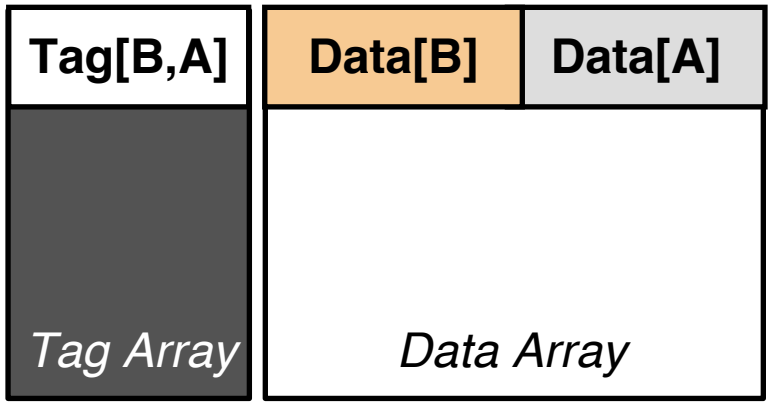
No Additional Area Overheads

PRACTICAL COMPRESSION



Cache Compression: Tradeoffs

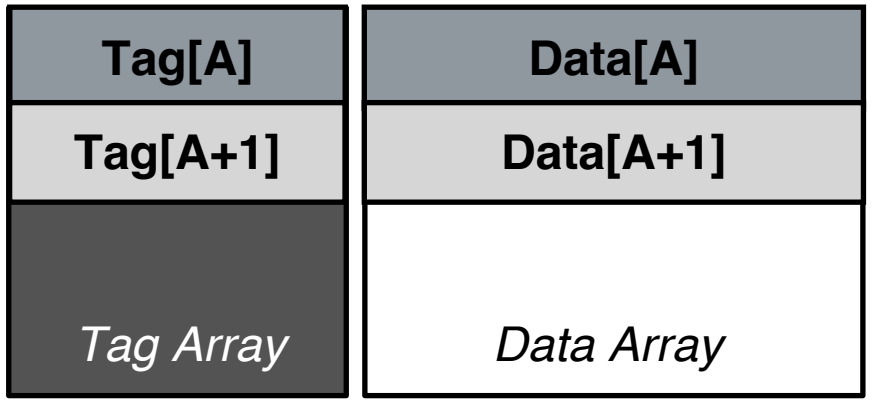
IDEAL COMPRESSION



No Additional Area Overheads

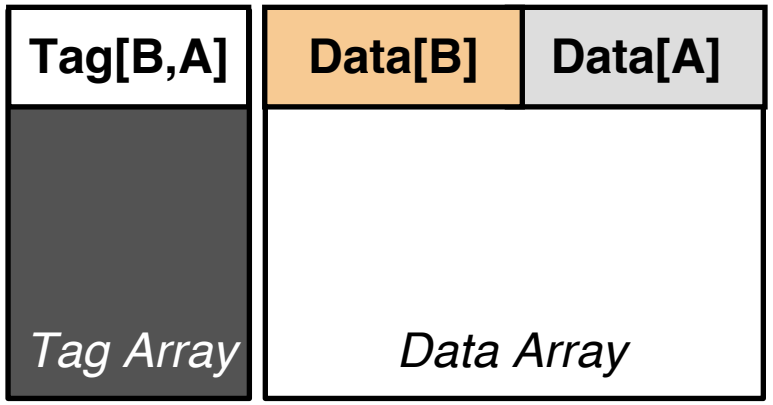
PRACTICAL COMPRESSION

Storing only adjacent blocks together



Cache Compression: Tradeoffs

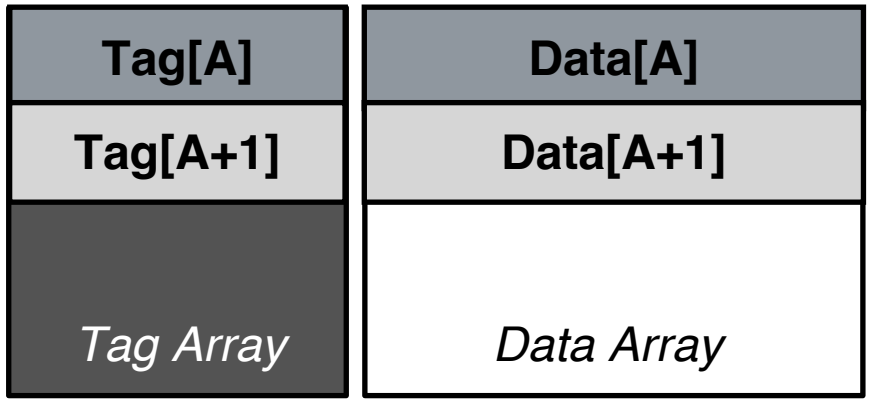
IDEAL COMPRESSION



No Additional Area Overheads

PRACTICAL COMPRESSION

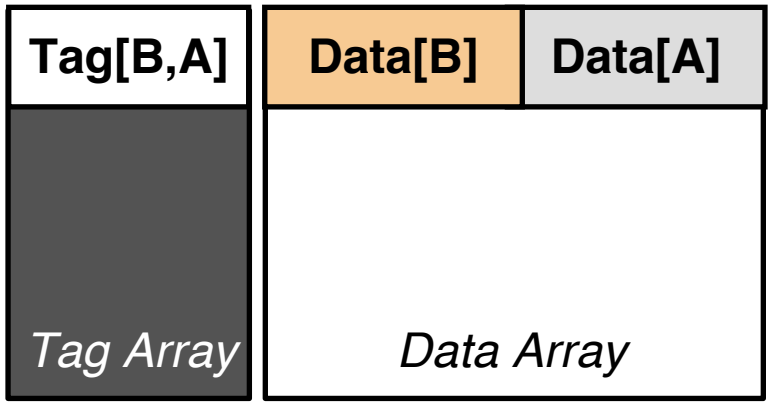
Storing only adjacent blocks together



Tag[A] ≈ Tag[A+1]

Cache Compression: Tradeoffs

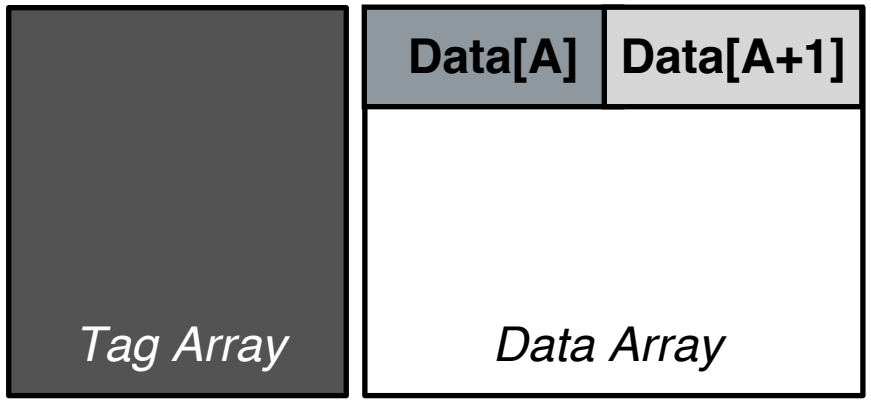
IDEAL COMPRESSION



No Additional Area Overheads

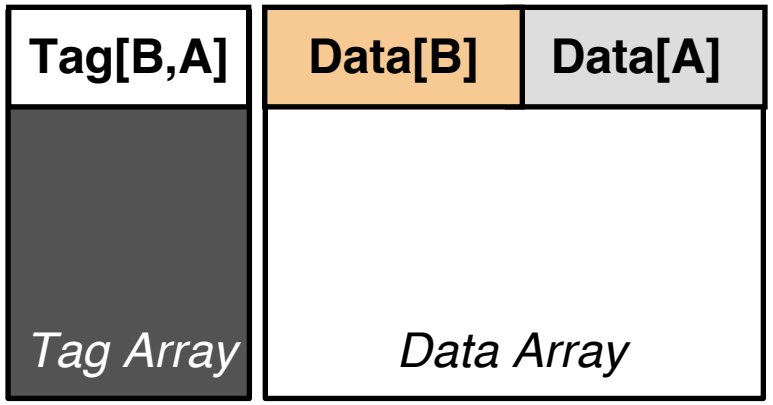
PRACTICAL COMPRESSION

Storing only adjacent blocks together



Cache Compression: Tradeoffs

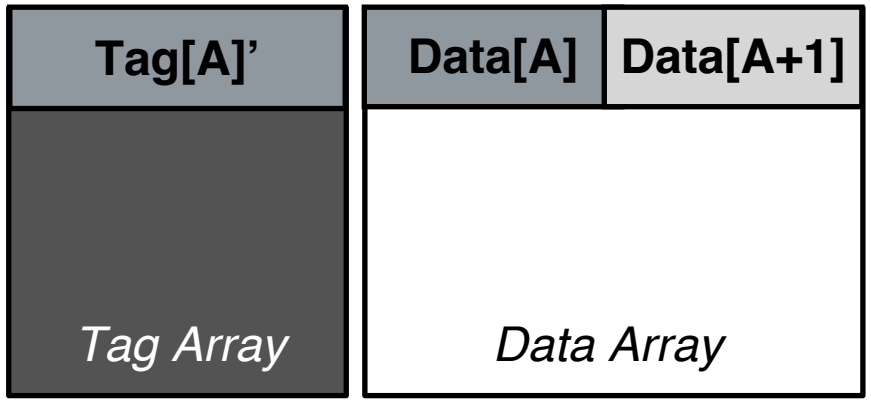
IDEAL COMPRESSION



No Additional Area Overheads

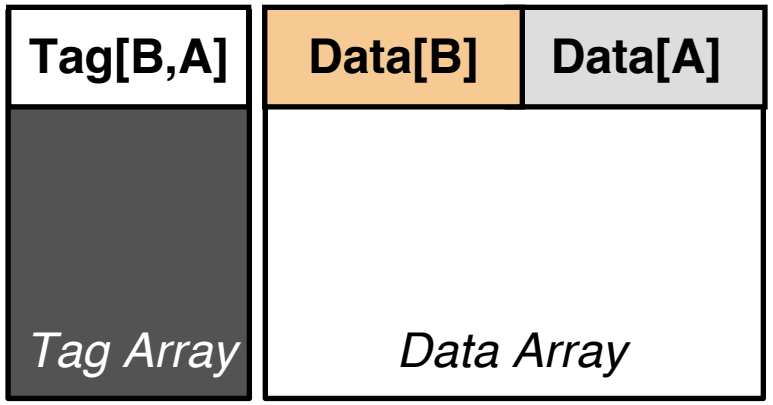
PRACTICAL COMPRESSION

Storing only adjacent blocks together



Cache Compression: Tradeoffs

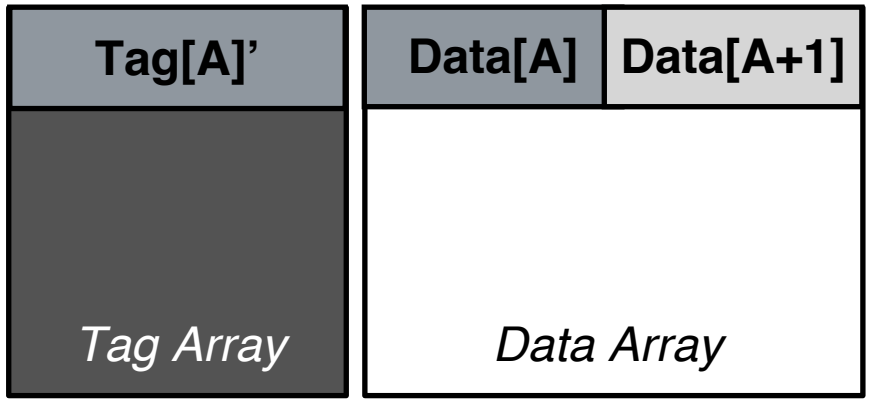
IDEAL COMPRESSION



No Additional Area Overheads

PRACTICAL COMPRESSION

Storing only adjacent blocks together

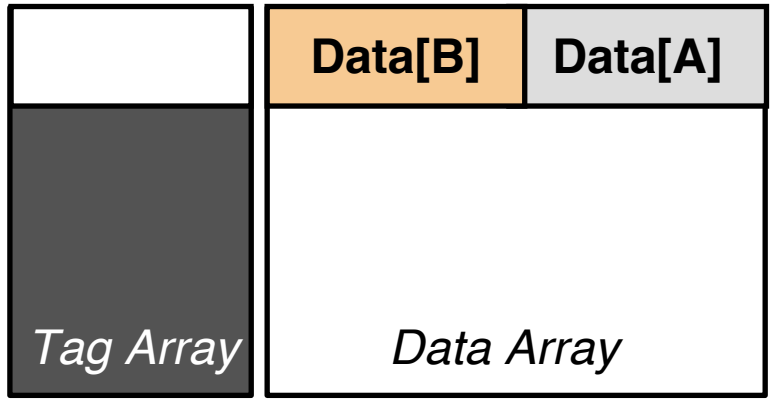


Restriction in Block placement!

Cache Compression: Our Work

Touché :

PRACTICAL TECHNIQUE FOR NEAR-IDEAL COMPRESSION



PRIOR WORK

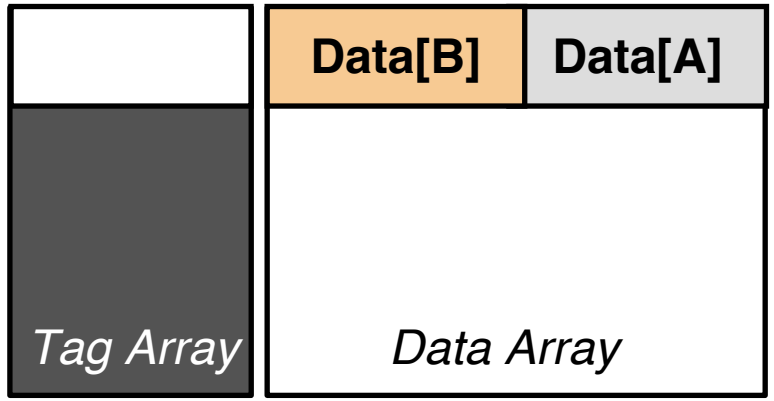
2x Tag Area

Restriction in Block placement

Cache Compression: Our Work

Touché :

PRACTICAL TECHNIQUE FOR NEAR-IDEAL COMPRESSION



No Additional Area Overheads!

PRIOR WORK

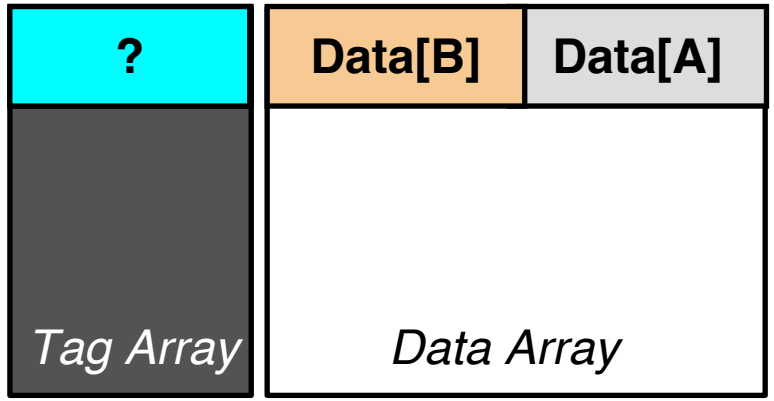
2x Tag Area

Restriction in Block placement

Cache Compression: Our Work

Touché :

PRACTICAL TECHNIQUE FOR NEAR-IDEAL COMPRESSION



No Additional Area Overheads!

PRIOR WORK

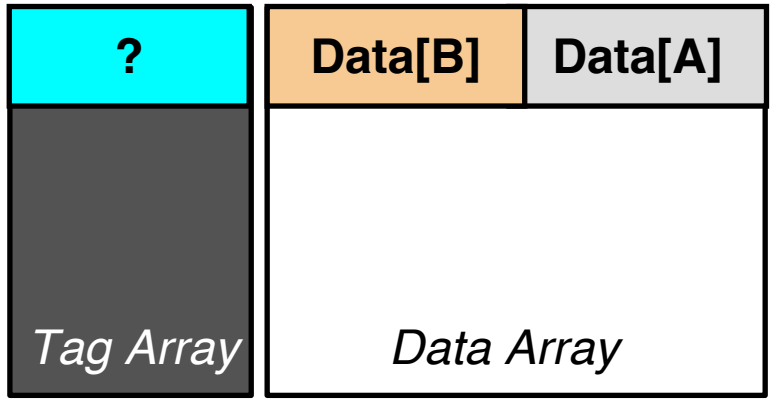
2x Tag Area

Restriction in Block placement

Cache Compression: Our Work

Touché :

PRACTICAL TECHNIQUE FOR NEAR-IDEAL COMPRESSION



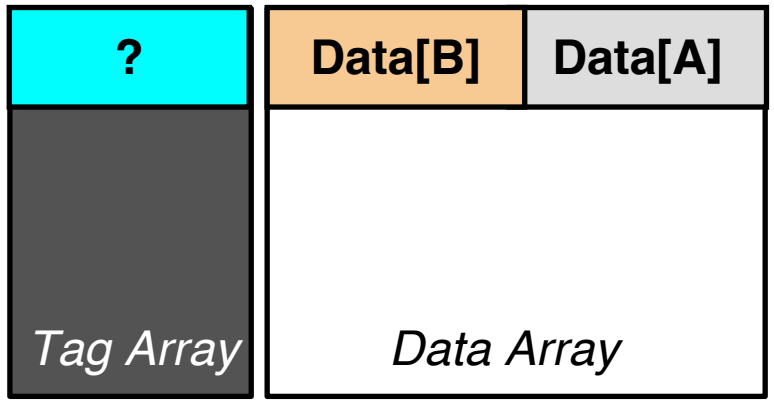
No Additional Area Overheads!

Three Key Mechanisms

Cache Compression: Our Work

Touché :

PRACTICAL TECHNIQUE FOR NEAR-IDEAL COMPRESSION



No Additional Area Overheads!

Three Key Mechanisms

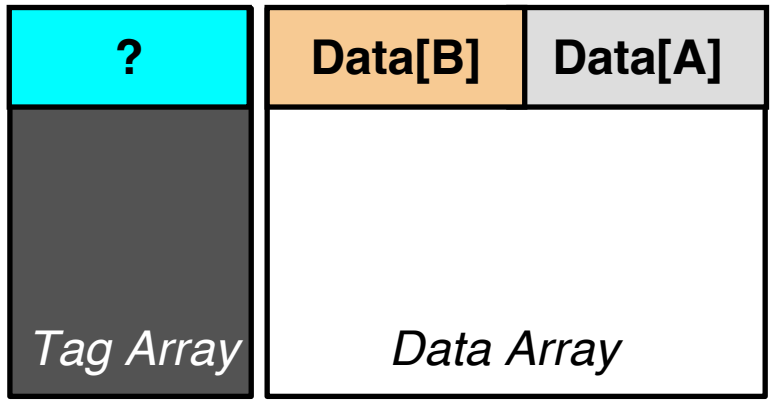
Signature (SIGN) Engine

+

Cache Compression: Our Work

Touché :

PRACTICAL TECHNIQUE FOR NEAR-IDEAL COMPRESSION



No Additional Area Overheads!

Three Key Mechanisms

Signature (SIGN) Engine

+

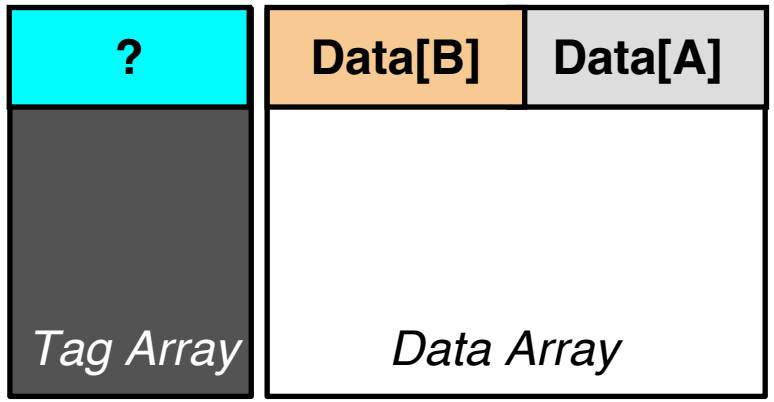
Tag Appended Data (TADA)

+

Cache Compression: Our Work

Touché :

PRACTICAL TECHNIQUE FOR NEAR-IDEAL COMPRESSION



No Additional Area Overheads!

Three Key Mechanisms

Signature (SIGN) Engine

+

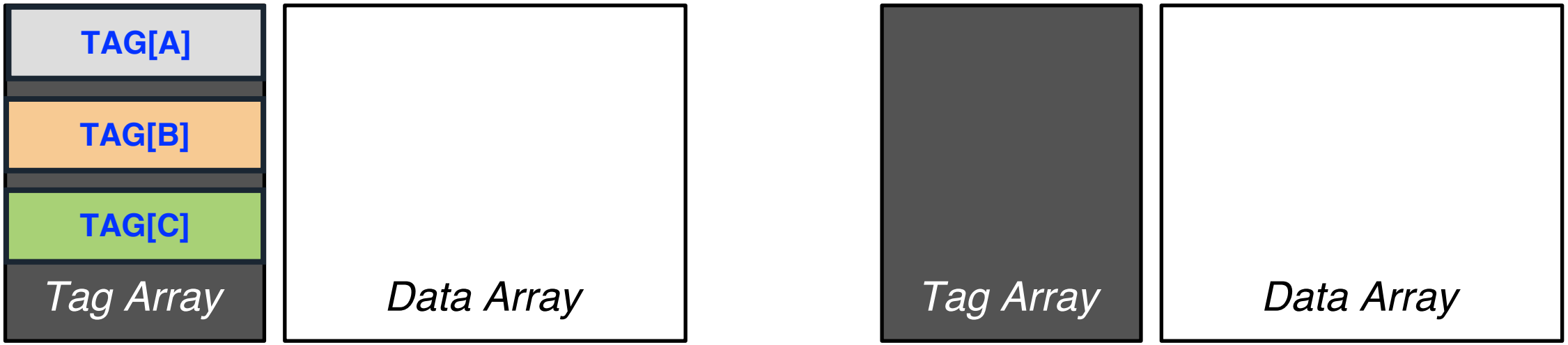
Tag Appended Data (TADA)

+

Superblock Marker (SMARK)

Signature (SIGN) Engine

Key Idea



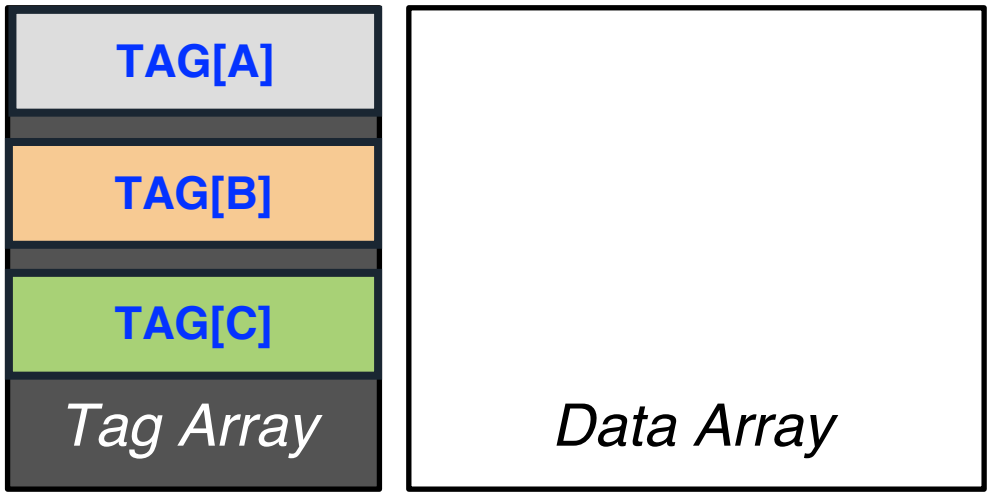
Uncompressed Cache

Touché

Signature (SIGN) Engine

Key Idea

✓ Using **shortened signatures**



Uncompressed Cache

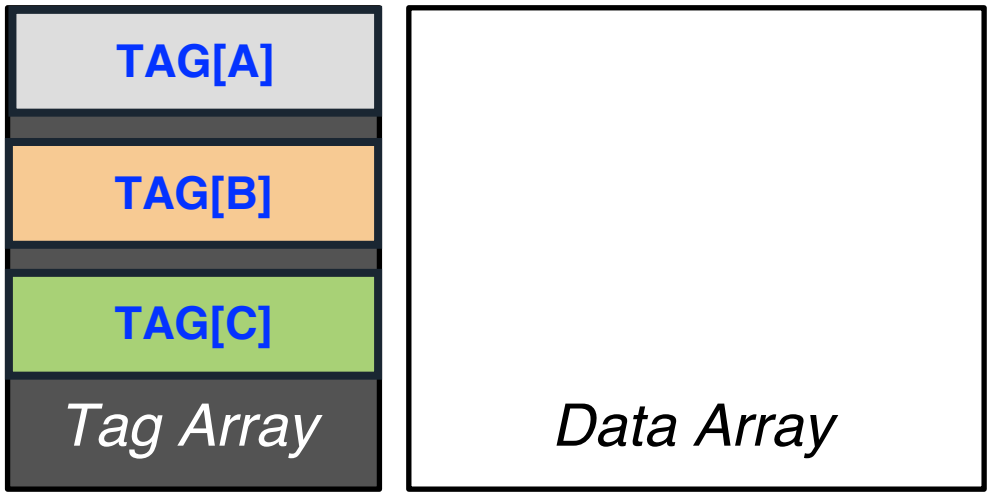
Touché

Signature (SIGN) Engine

Key Idea

✓ Using **shortened signatures**

✓ Store the shortened signatures in a tag entry



Uncompressed Cache

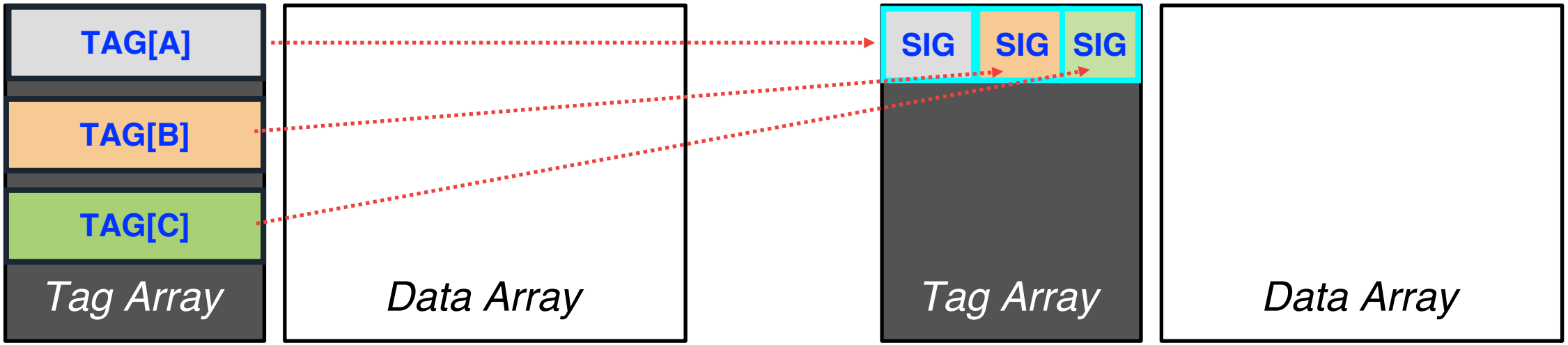
Touché

Signature (SIGN) Engine

Key Idea

✓ Using **shortened signatures**

✓ Store the shortened signatures in a tag entry



Uncompressed Cache

Touché

Signature (SIGN) Engine

How to generate Signature?

TAG[A]

Touché

Signature (SIGN) Engine

How to generate Signature?

✓ Generates shortened signatures (9 bits) from the full tag addresses (29 bits)

Touché

Signature (SIGN) Engine

How to generate Signature?

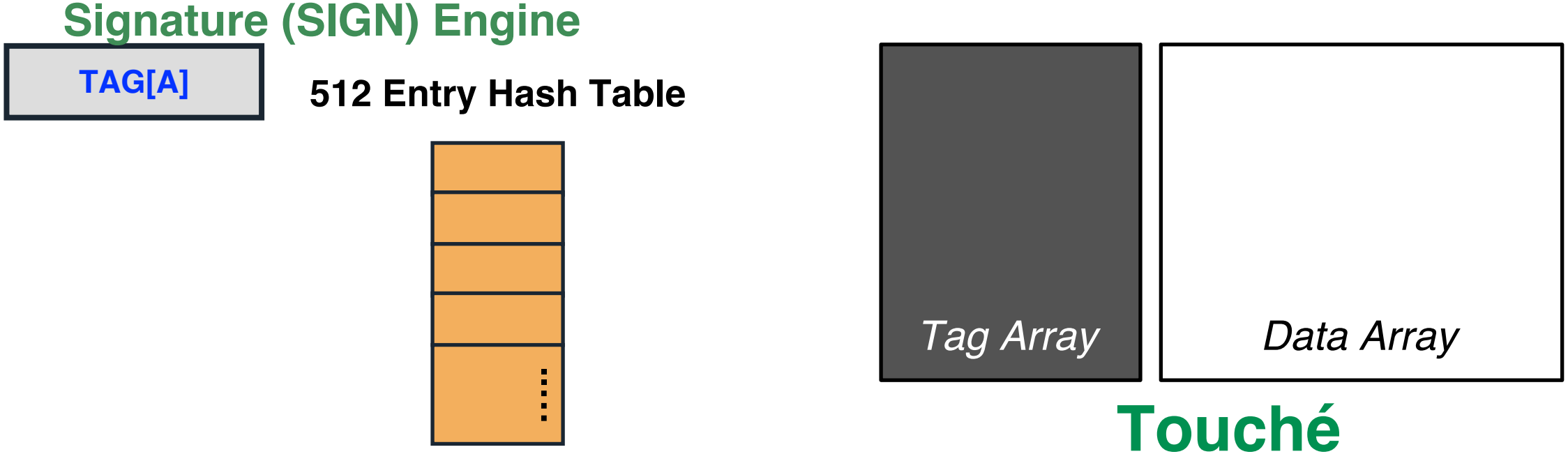
✓ Generates shortened signatures (9 bits) from the full tag addresses (29 bits)

Touché

Signature (SIGN) Engine

How to generate Signature?

✓ Generates shortened signatures (9 bits) from the full tag addresses (29 bits)

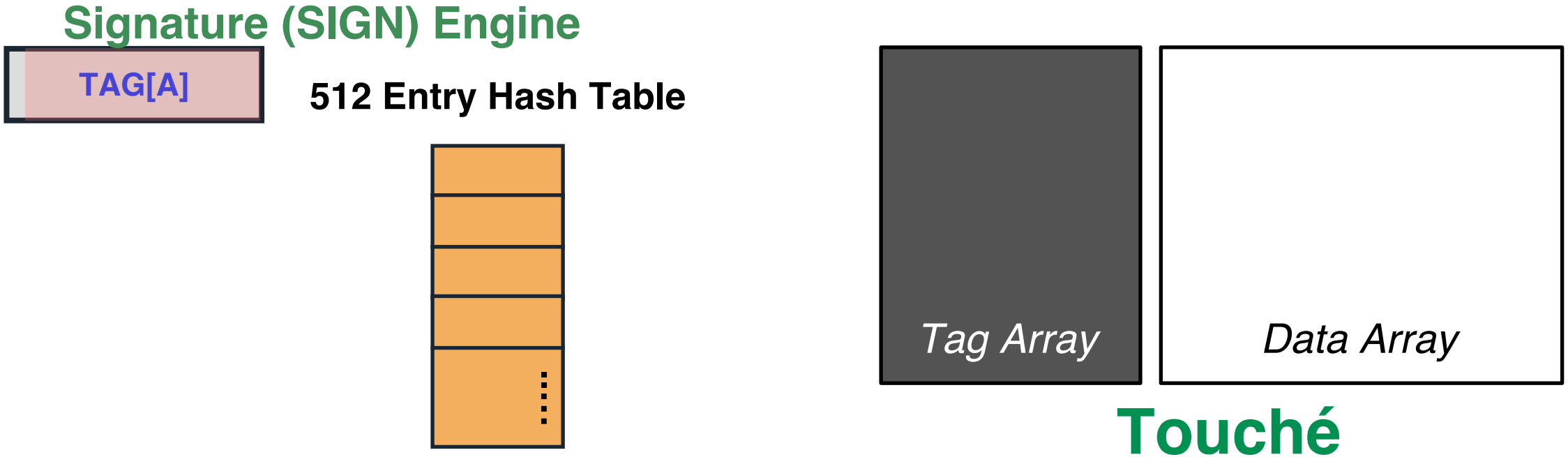


Touché

Signature (SIGN) Engine

How to generate Signature?

✓ Generates shortened signatures (9 bits) from the full tag addresses (29 bits)

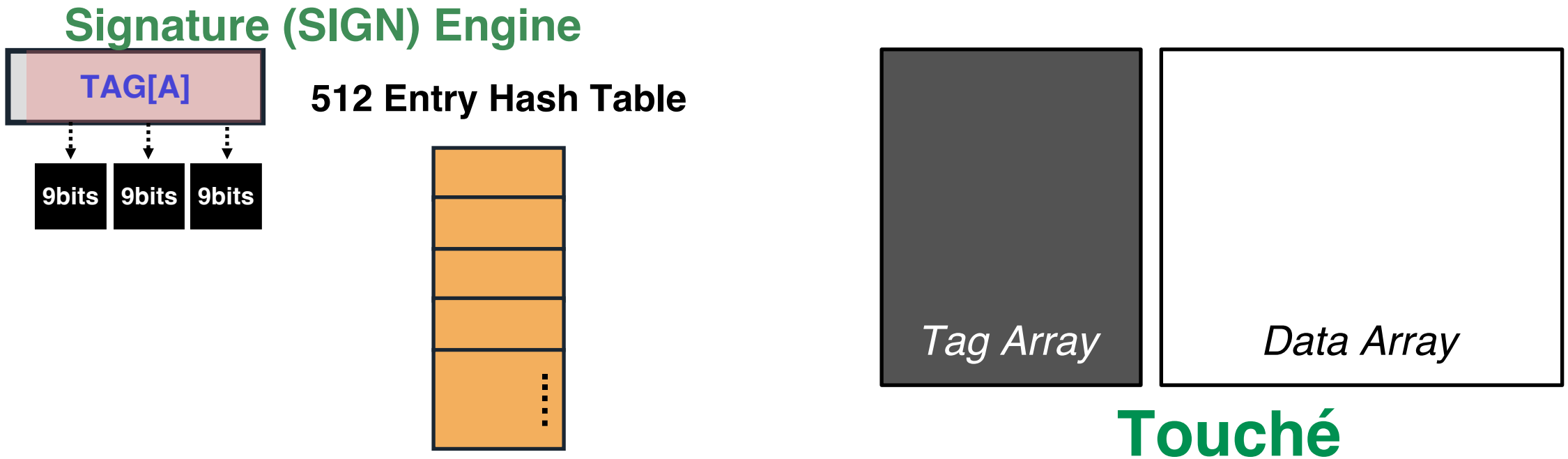


Touché

Signature (SIGN) Engine

How to generate Signature?

✓ Generates shortened signatures (9 bits) from the full tag addresses (29 bits)

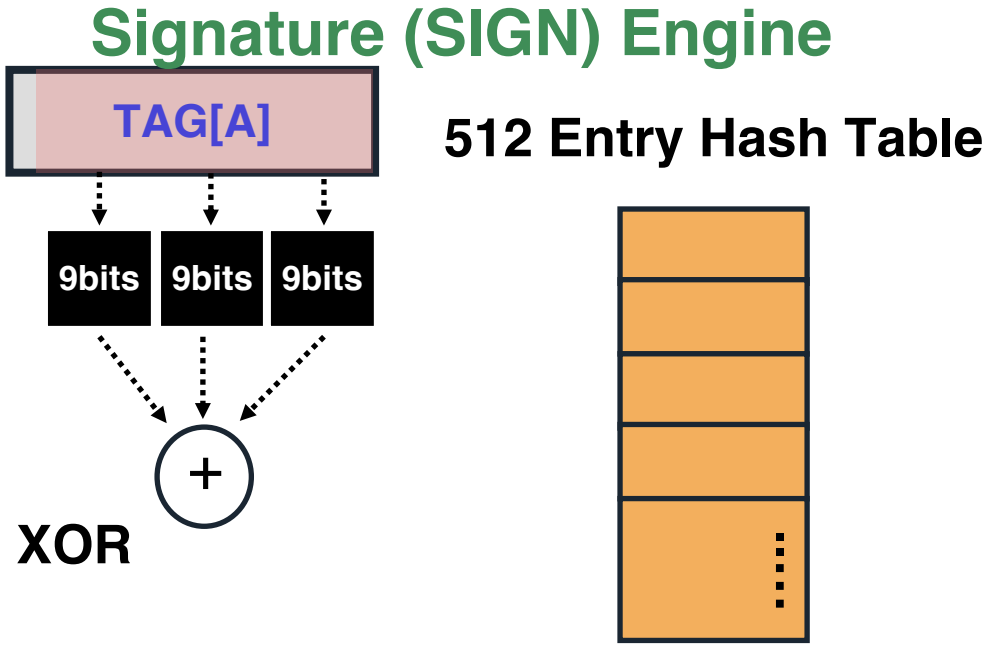


Touché

Signature (SIGN) Engine

How to generate Signature?

✓ Generates shortened signatures (9 bits) from the full tag addresses (29 bits)

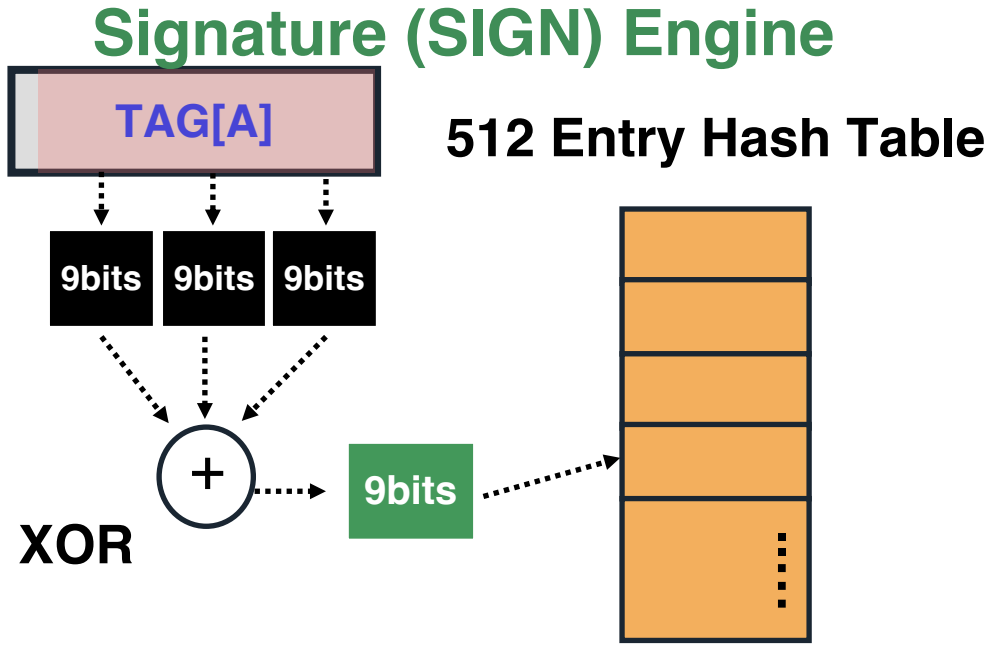


Touché

Signature (SIGN) Engine

How to generate Signature?

✓ Generates shortened signatures (9 bits) from the full tag addresses (29 bits)

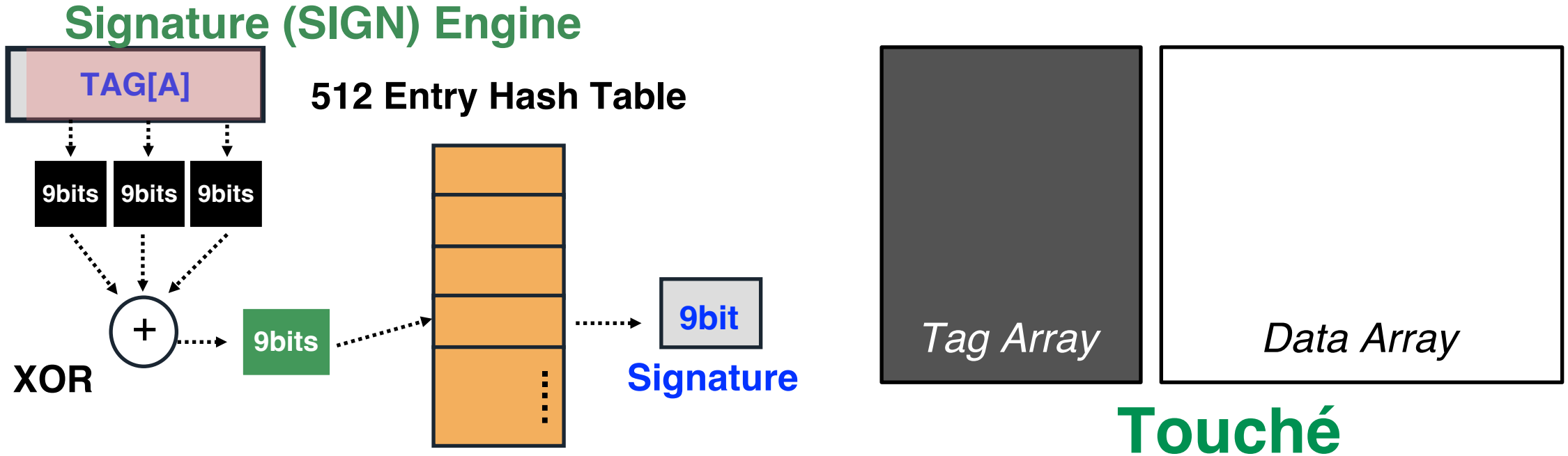


Touché

Signature (SIGN) Engine

How to generate Signature?

✓ Generates shortened signatures (9 bits) from the full tag addresses (29 bits)

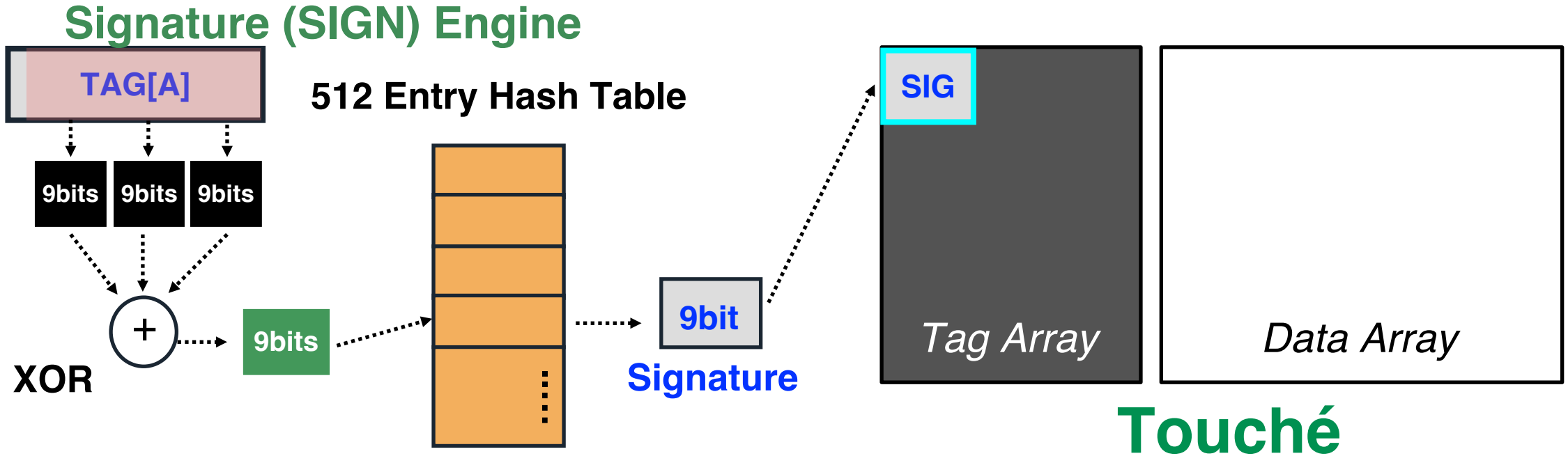


Touché

Signature (SIGN) Engine

How to generate Signature?

✓ Generates shortened signatures (9 bits) from the full tag addresses (29 bits)



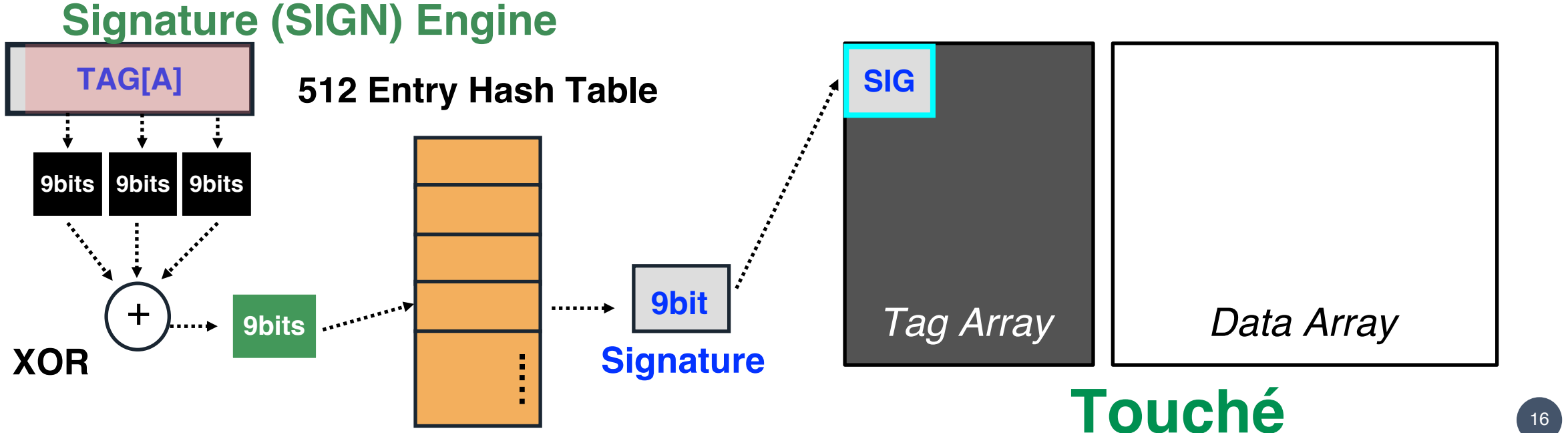
Touché

Signature (SIGN) Engine

How to generate Signature?

✓ Generates shortened signatures (9 bits) from the full tag addresses (29 bits)

✓ Only requires one XOR operation and a hash table lookup



Touché

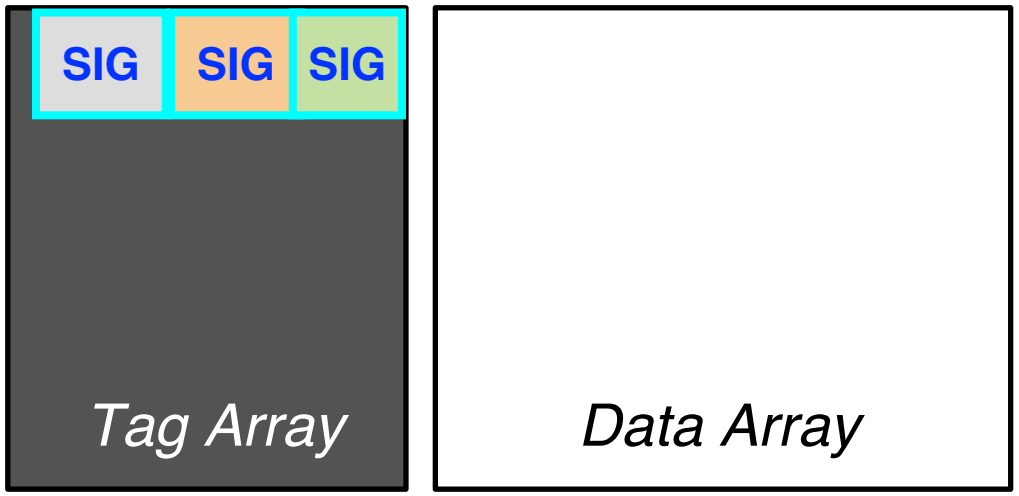
Signature (SIGN) Engine

Key Idea

✓ Using Shortened Signatures

✓ Store the shortened signatures in a tag entry

✓ **Identifying compressed lines** by **using the valid and dirty bits**



Touché

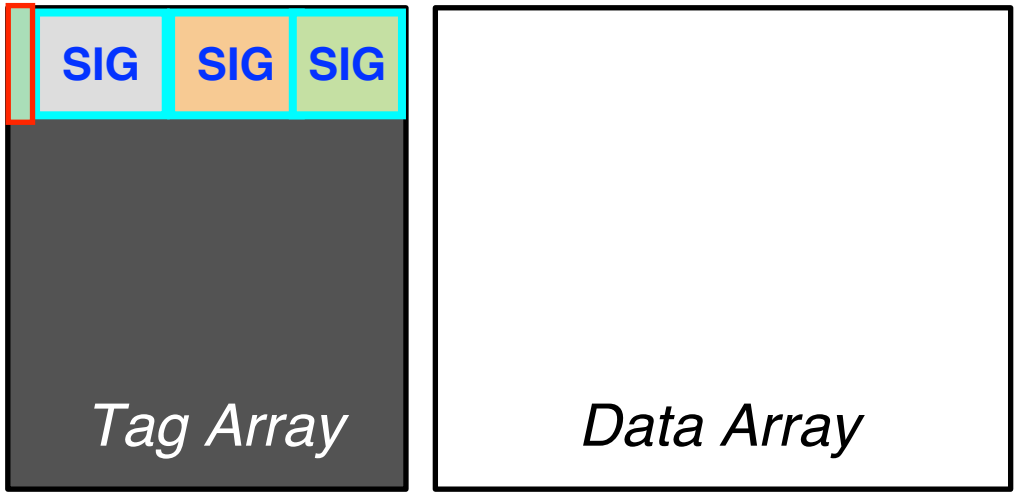
Signature (SIGN) Engine

Key Idea

✓ Using Shortened Signatures

✓ Store the shortened signatures in a tag entry

✓ **Identifying compressed lines** by **using the valid and dirty bits**



Touché

Signature (SIGN) Engine

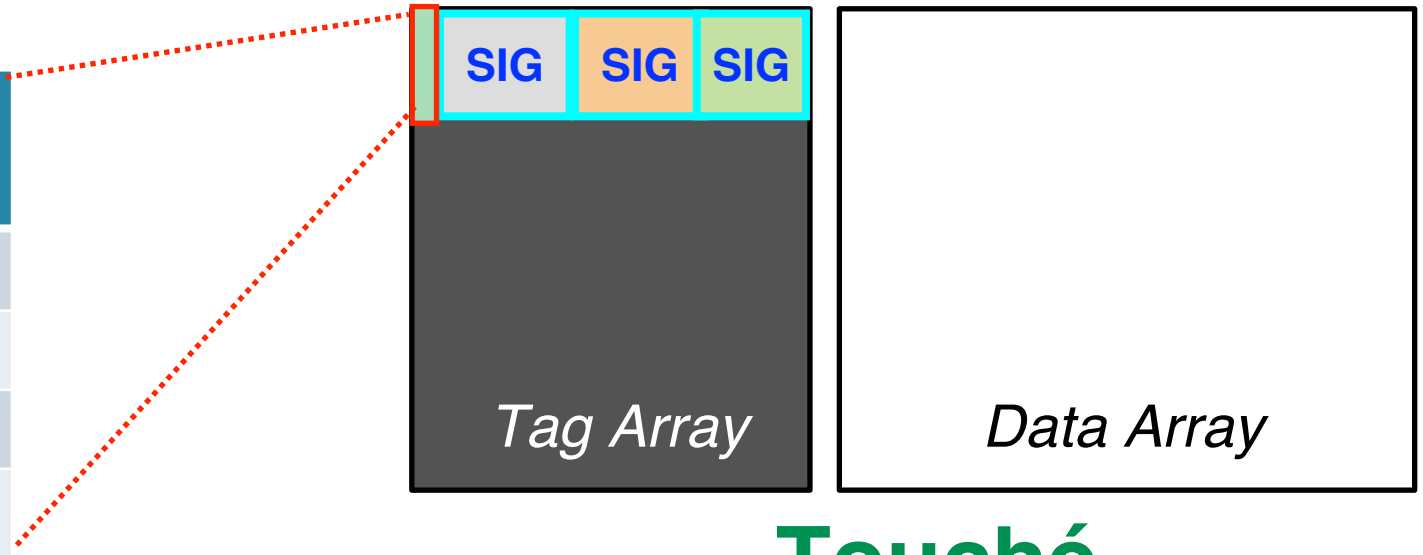
Key Idea

✓ Using Shortened Signatures

✓ Store the shortened signatures in a tag entry

✓ **Identifying compressed lines** by **using the valid and dirty bits**

Valid bit	Dirty bit	Cacheline Status
0	0	Invalid
1	0	Valid and Clean
1	1	Valid and Dirty
0	**1**	**Unused**



Touché

Signature (SIGN) Engine

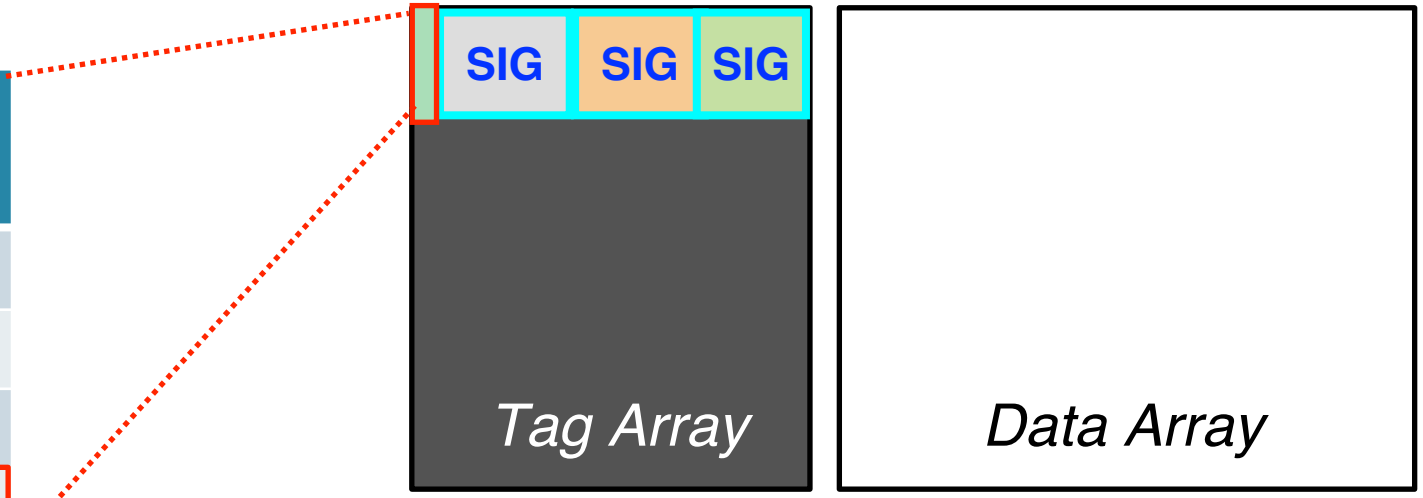
Key Idea

✓ Using Shortened Signatures

✓ Store the shortened signatures in a tag entry

✓ **Identifying compressed lines** by **using the valid and dirty bits**

Valid bit	Dirty bit	Cacheline Status
0	0	Invalid
1	0	Valid and Clean
1	1	Valid and Dirty
0	**1**	**Unused**



Touché

Signature (SIGN) Engine

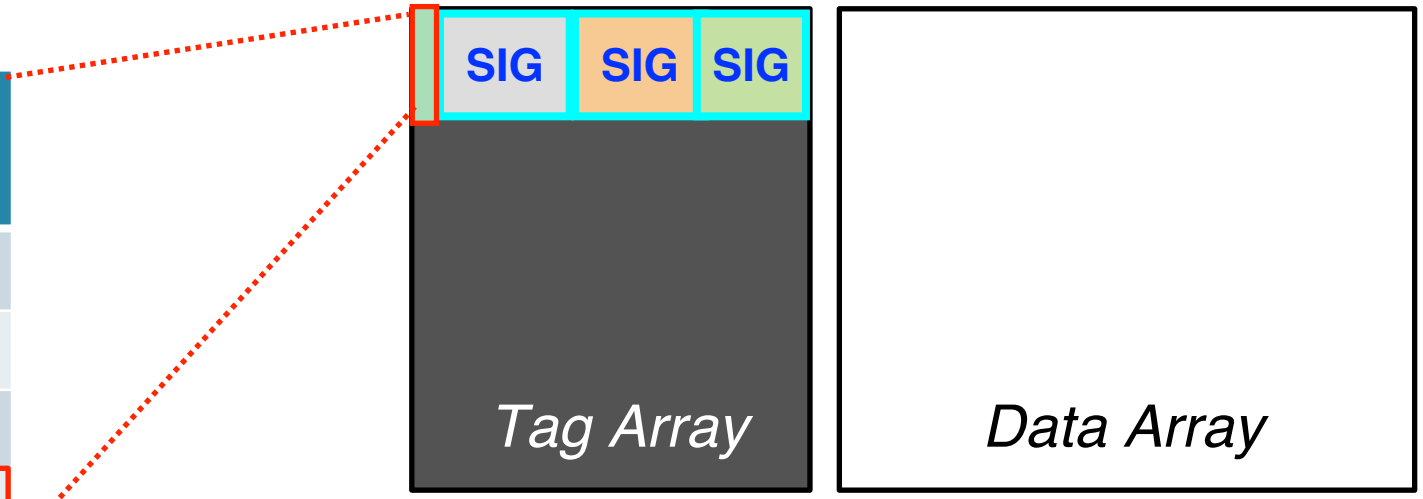
Key Idea

✓ Using Shortened Signatures

✓ Store the shortened signatures in a tag entry

✓ **Identifying compressed lines** by **using the valid and dirty bits**

Valid bit	Dirty bit	Cacheline Status
0	0	Invalid
1	0	Valid and Clean
1	1	Valid and Dirty
0	**1**	**Compressed**



Touché

Signature (SIGN) Engine

An Issue

Signature (SIGN) Engine

An Issue

✓ Signatures are shorter than full tags

Signature (SIGN) Engine

An Issue

✓ Signatures are shorter than full tags
→ Several tags can map into the same signature

Signature (SIGN) Engine

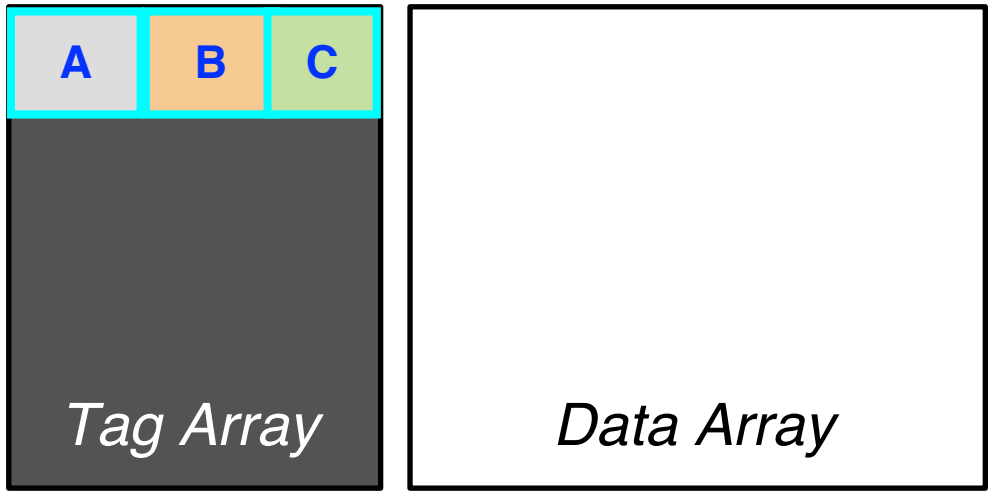
An Issue

✓ Signatures are shorter than full tags
→ Several tags can map into the same signature → **Signature Collision**

Signature (SIGN) Engine

An Issue

✓ Signatures are shorter than full tags
→ Several tags can map into the same signature → **Signature Collision**

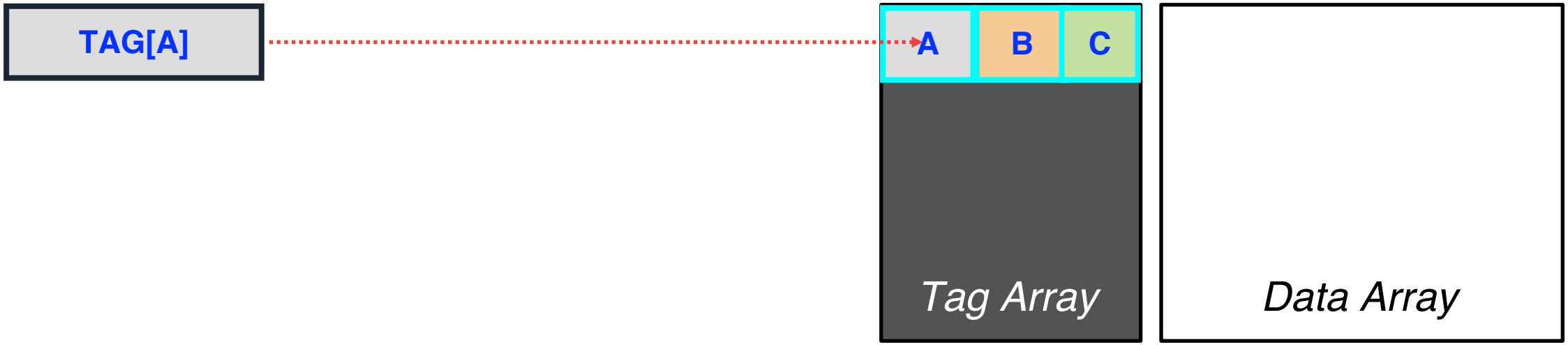


Touché

Signature (SIGN) Engine

An Issue

✓ Signatures are shorter than full tags
→ Several tags can map into the same signature → **Signature Collision**

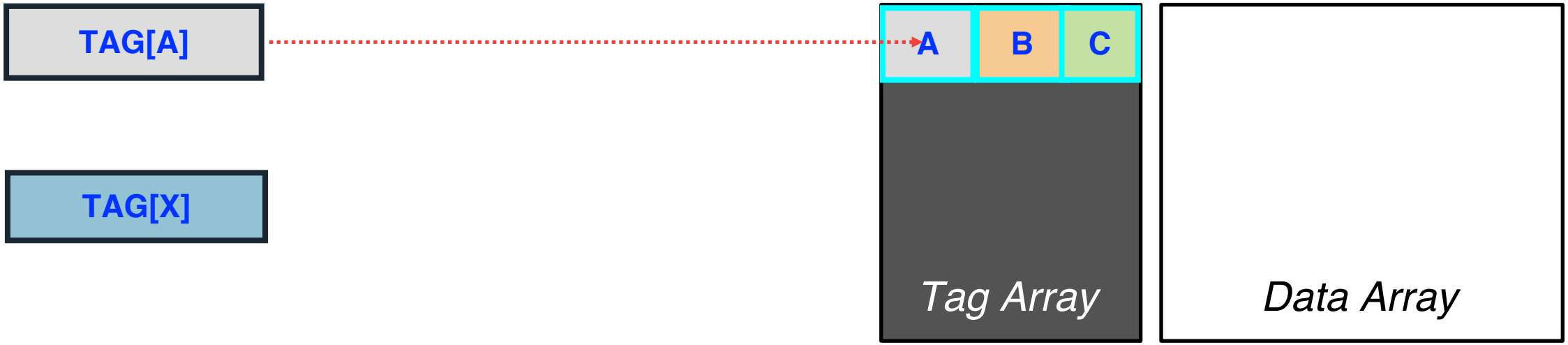


Touché

Signature (SIGN) Engine

An Issue

✓ Signatures are shorter than full tags
→ Several tags can map into the same signature → **Signature Collision**

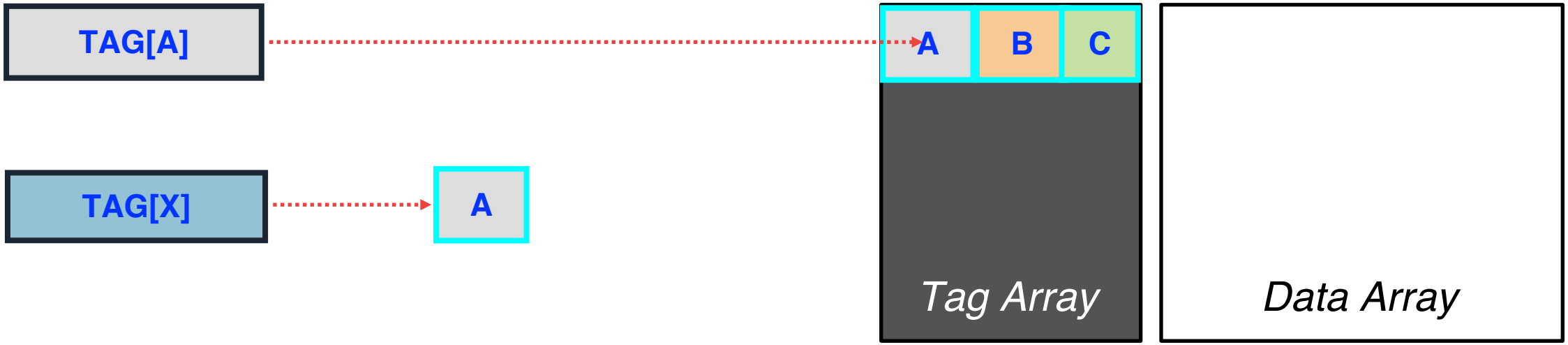


Touché

Signature (SIGN) Engine

An Issue

✓ Signatures are shorter than full tags
→ Several tags can map into the same signature → **Signature Collision**

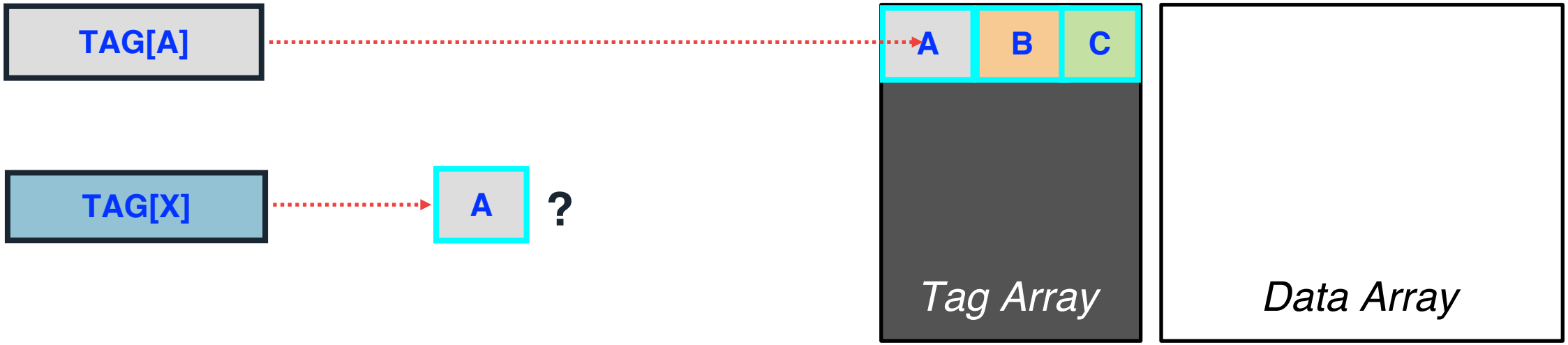


Touché

Signature (SIGN) Engine

An Issue

✓ Signatures are shorter than full tags
→ Several tags can map into the same signature → **Signature Collision**

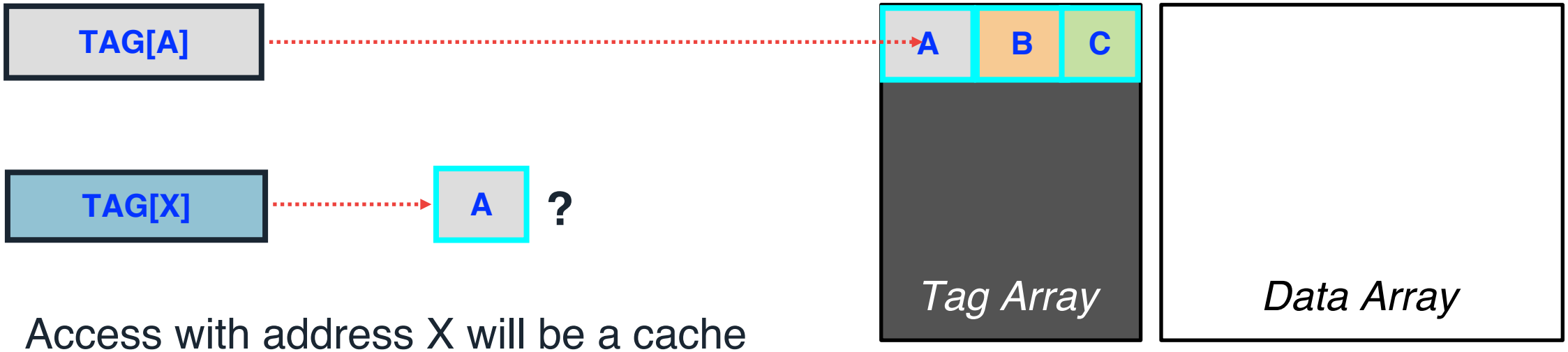


Touché

Signature (SIGN) Engine

An Issue

✓ Signatures are shorter than full tags
→ Several tags can map into the same signature → **Signature Collision**



Access with address X will be a cache

hit, but it should be a cache miss!!

Touché

Signature (SIGN) Engine

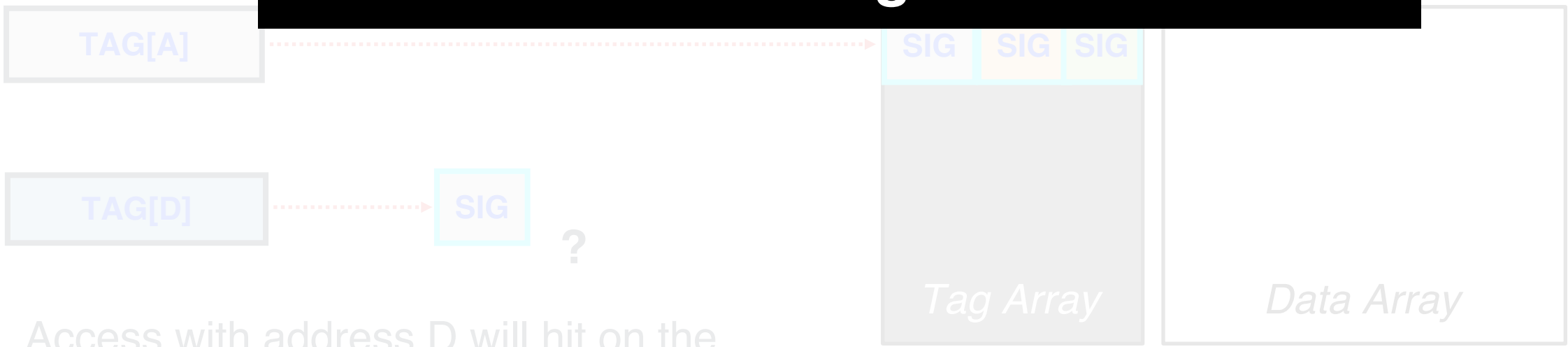
An Issue with SIGN

✓ Signatures are shorter than full tags

Sever

How to resolve the Signature Collision?

Collision



Access with address D will hit on the

cache, but it should be miss!!

Touché

Signature (SIGN) Engine

How to resolve the Signature Collision?

→ Tag Appended Data (TADA)

Tag Appended Data (TADA)

Key Idea

Tag Appended Data (TADA)

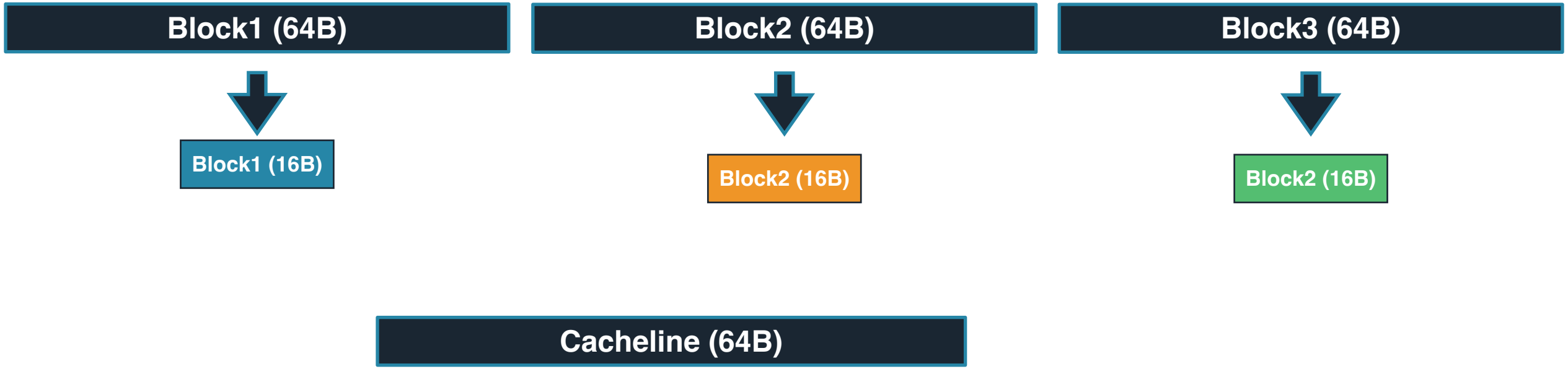
Key Idea

✓ Appending full tag addresses to the end of cacheline

Tag Appended Data (TADA)

Key Idea

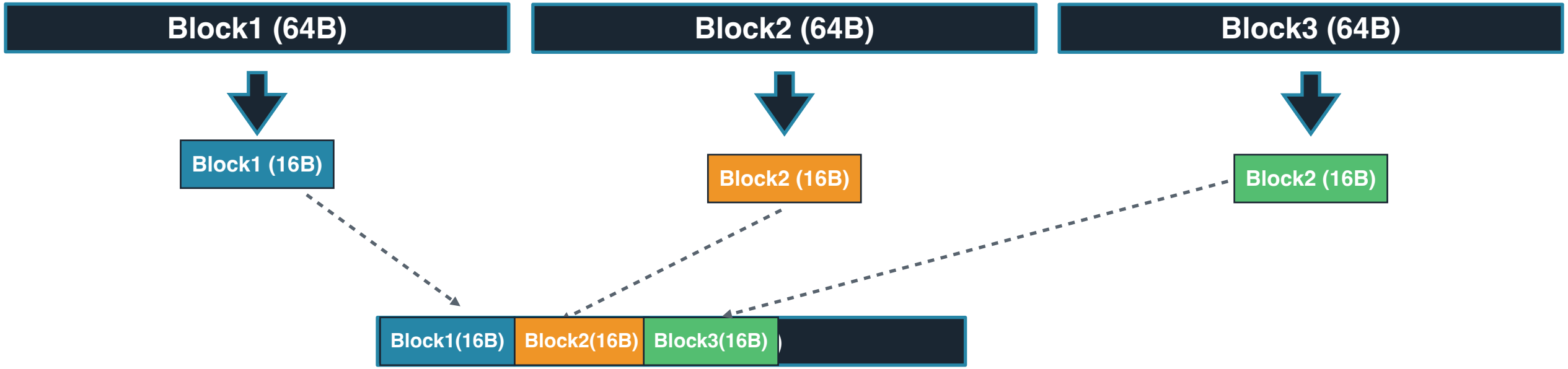
✓ Appending full tag addresses to the end of cacheline



Tag Appended Data (TADA)

Key Idea

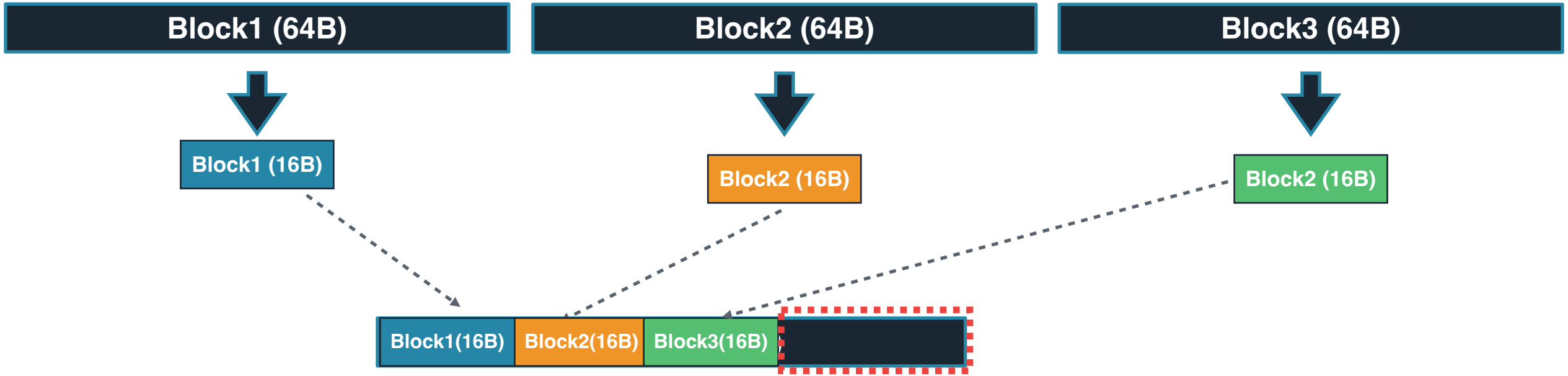
✓ Appending full tag addresses to the end of cacheline



Tag Appended Data (TADA)

Key Idea

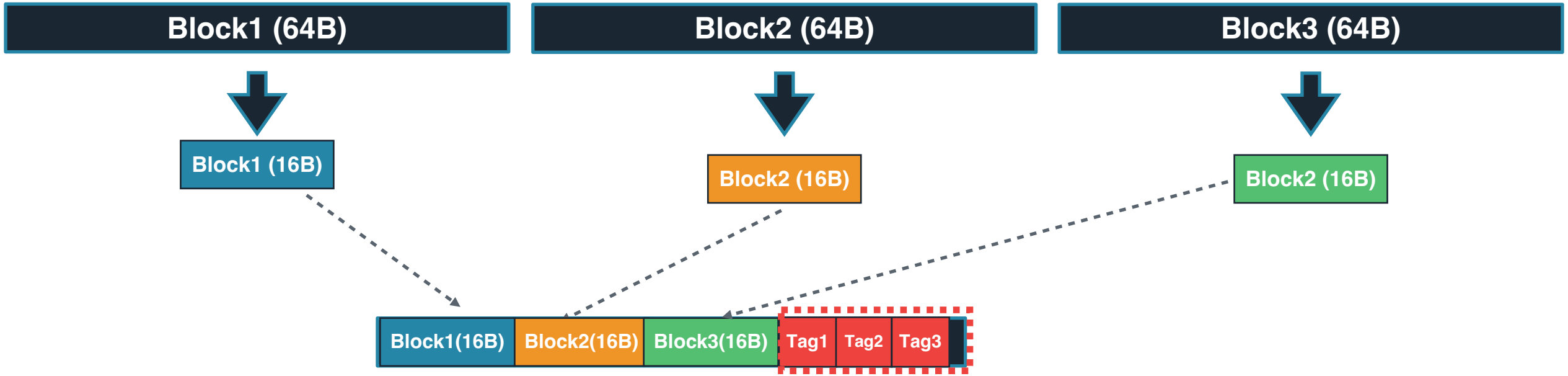
✓ Appending full tag addresses to the end of cacheline



Tag Appended Data (TADA)

Key Idea

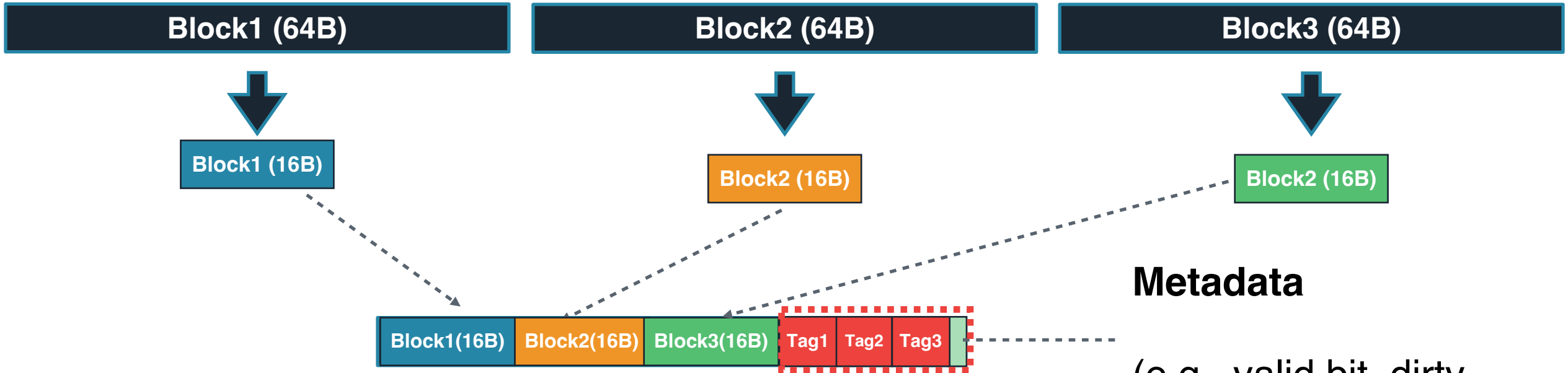
✓ Appending full tag addresses to the end of cacheline



Tag Appended Data (TADA)

Key Idea

✓ Appending full tag addresses to the end of cacheline



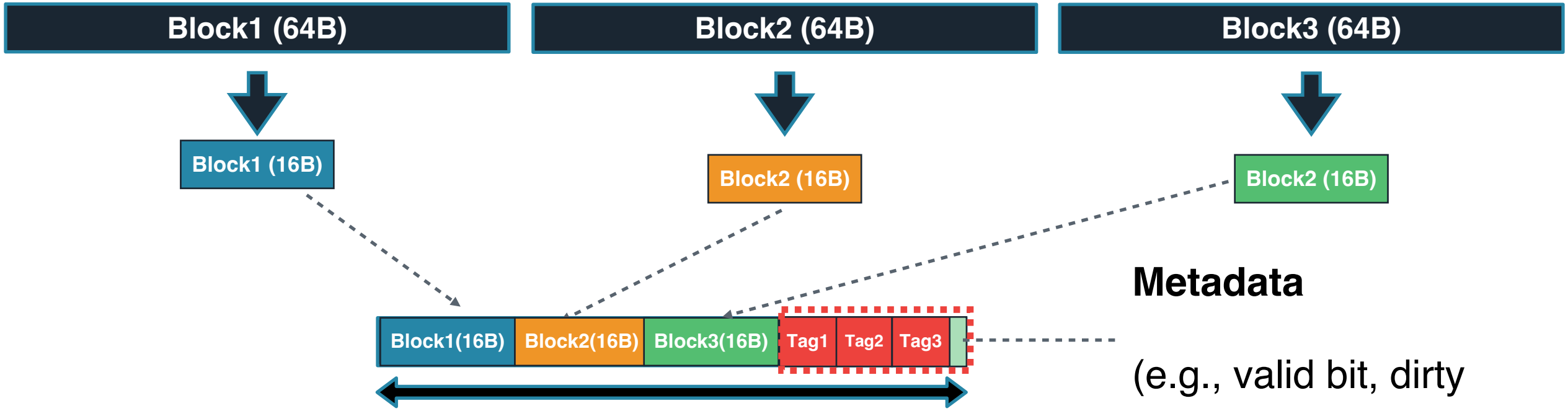
Metadata

(e.g., valid bit, dirty bit, compressibility)

Tag Appended Data (TADA)

Key Idea

✓ Appending full tag addresses to the end of cacheline

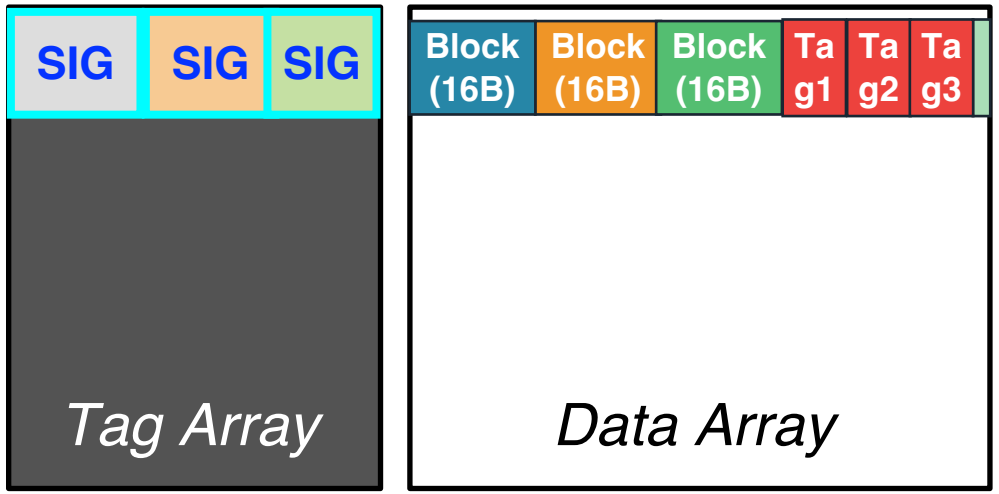


Metadata

(e.g., valid bit, dirty bit, compressibility)

SIGN + TADA

“SIGN+TADA” enable to store **three compressed blocks** in a

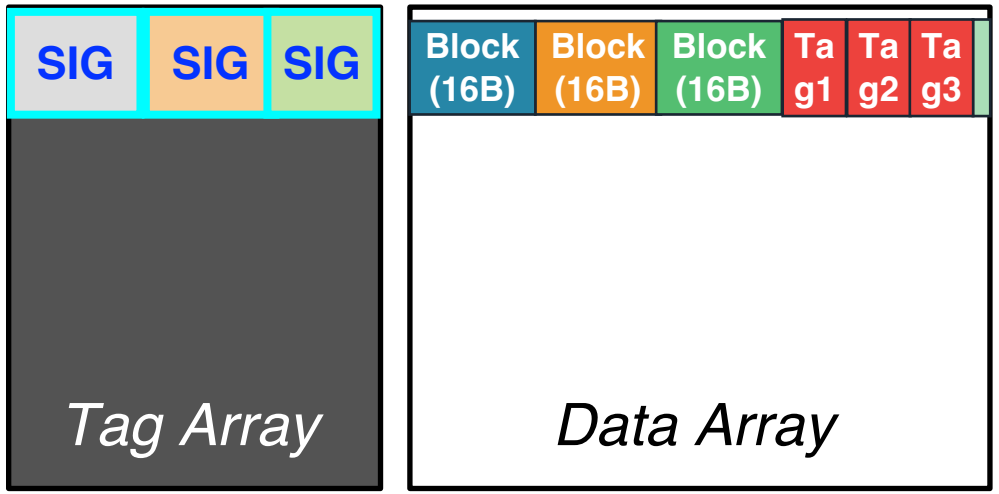


Touché

SIGN + TADA

“SIGN+TADA” enable to store **three compressed blocks** in a

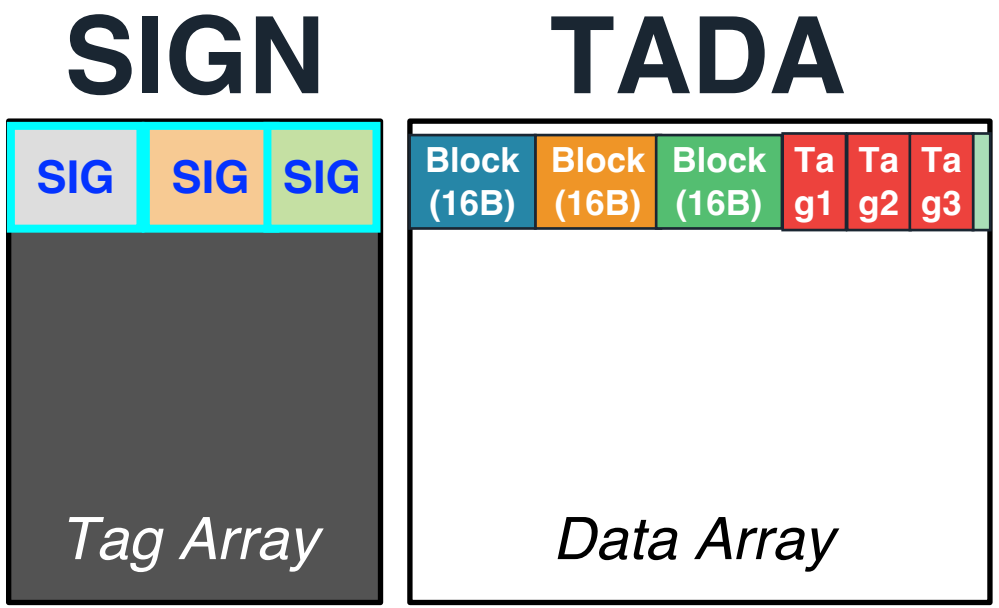
SIGN



Touché

SIGN + TADA

“SIGN+TADA” enable to store **three compressed blocks** in a



Touché

Superblock Marker (SMARK)

To improve the effective cache capacity **further**,

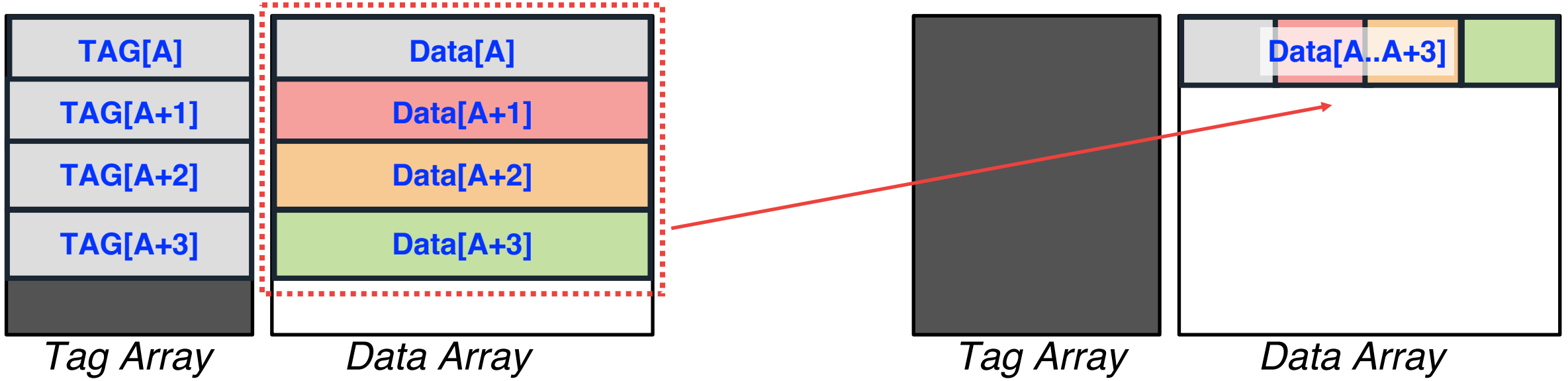
Superblock Marker (SMARK)

To improve the effective cache capacity **further**,

Uncompressed Cache

Superblock Marker (SMARK)

To improve the effective cache capacity **further**,

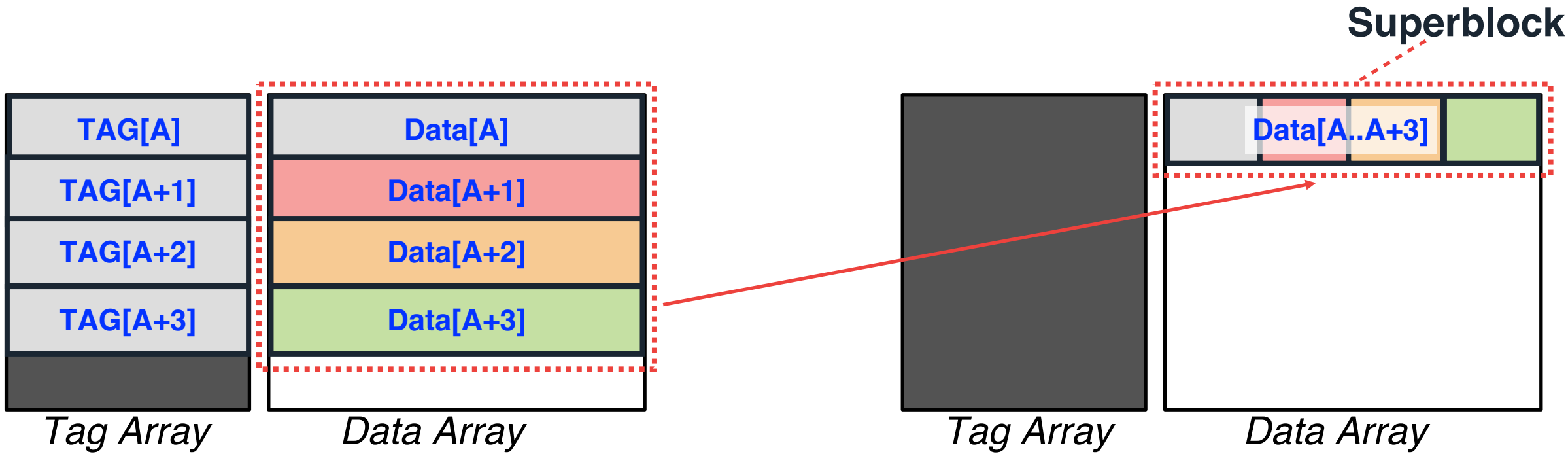


Uncompressed Cache

Touché

Superblock Marker (SMARK)

To improve the effective cache capacity **further**,



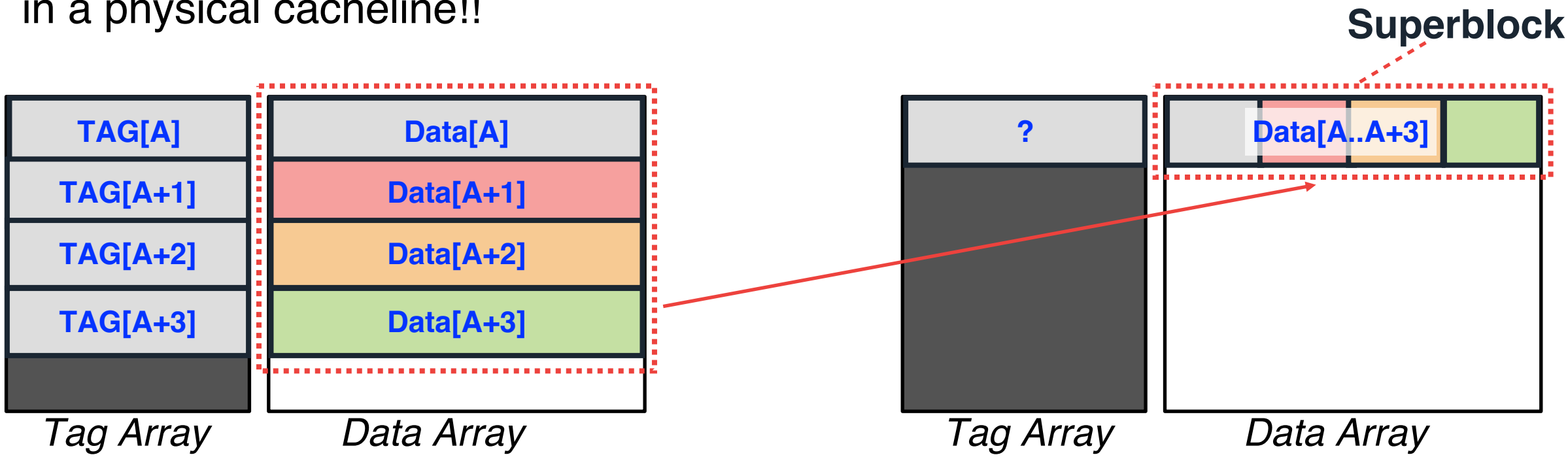
Uncompressed Cache

Touché

Superblock Marker (SMARK)

To improve the effective cache capacity **further**,

SMARK enables to store **four neighboring compressed blocks** (Superblock) in a physical cacheline!!

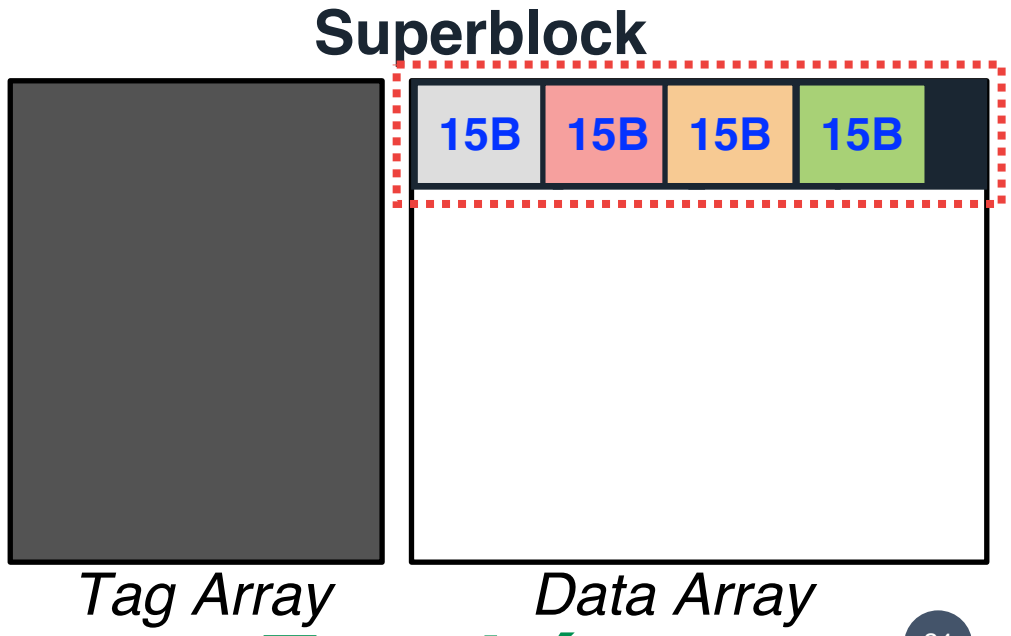


Uncompressed Cache

Touché

Superblock Marker (SMARK)

Key Idea

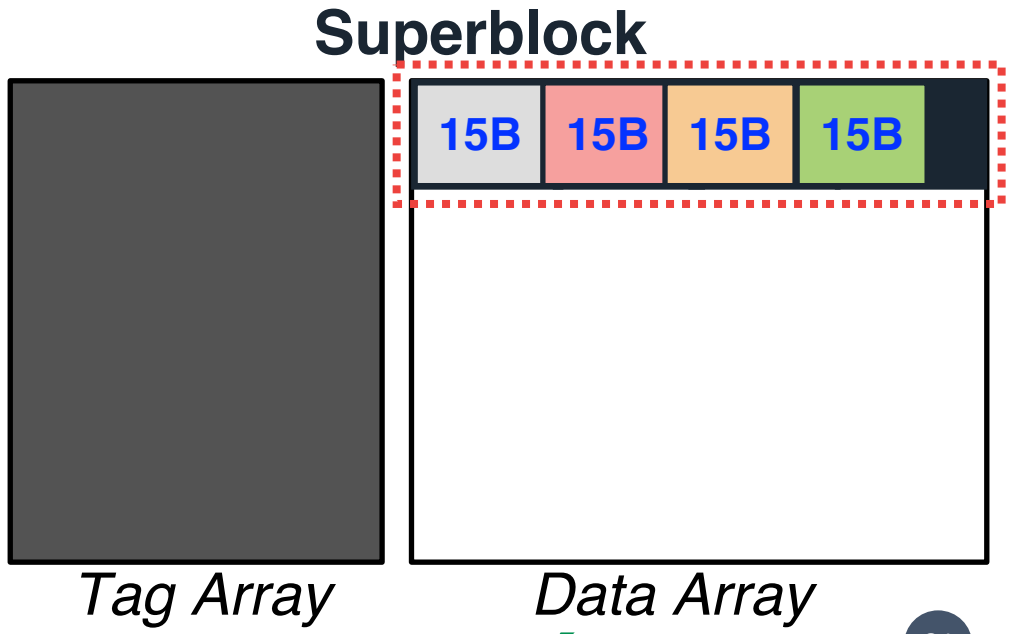


Touché

Superblock Marker (SMARK)

Key Idea

✓ To indicate the superblock, stores a random 16-bit marker (SMARK) in a tag entry

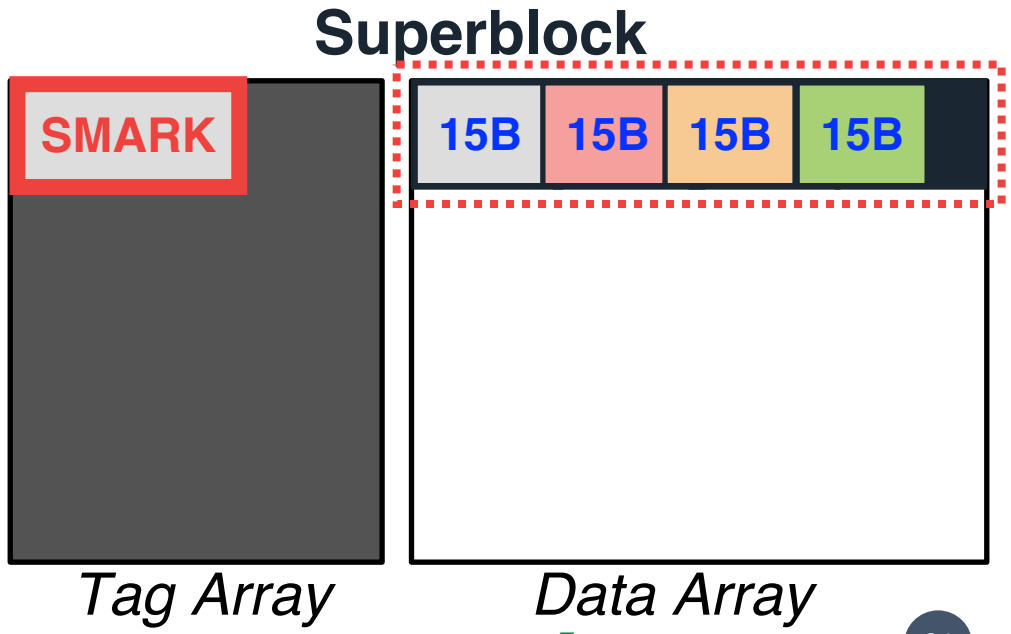


Touché

Superblock Marker (SMARK)

Key Idea

✓ To indicate the superblock, stores a random 16-bit marker (SMARK) in a tag entry



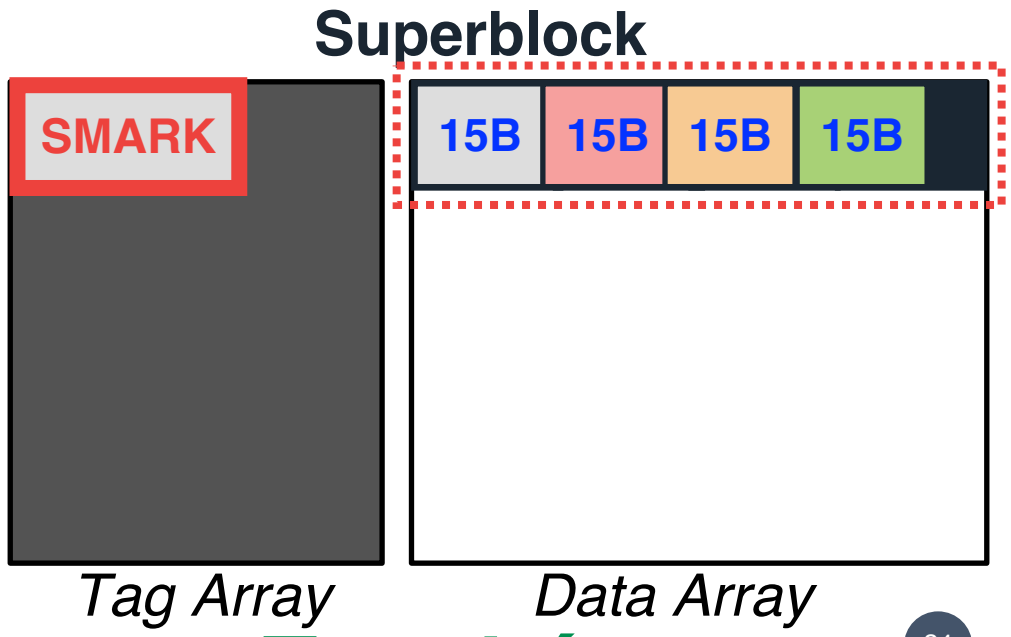
Touché

Superblock Marker (SMARK)

Key Idea

✓ To indicate the superblock, stores a random 16-bit marker (SMARK) in a tag entry

✓ Appends the signature to the marker



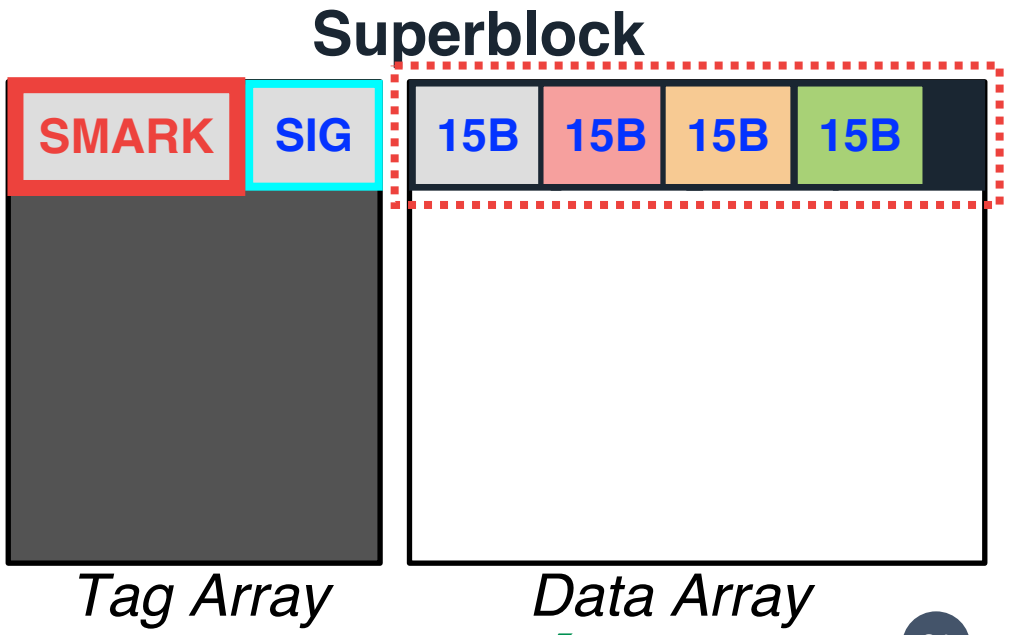
Touché

Superblock Marker (SMARK)

Key Idea

✓ To indicate the superblock, stores a random 16-bit marker (SMARK) in a tag entry

✓ Appends the signature to the marker



Touché

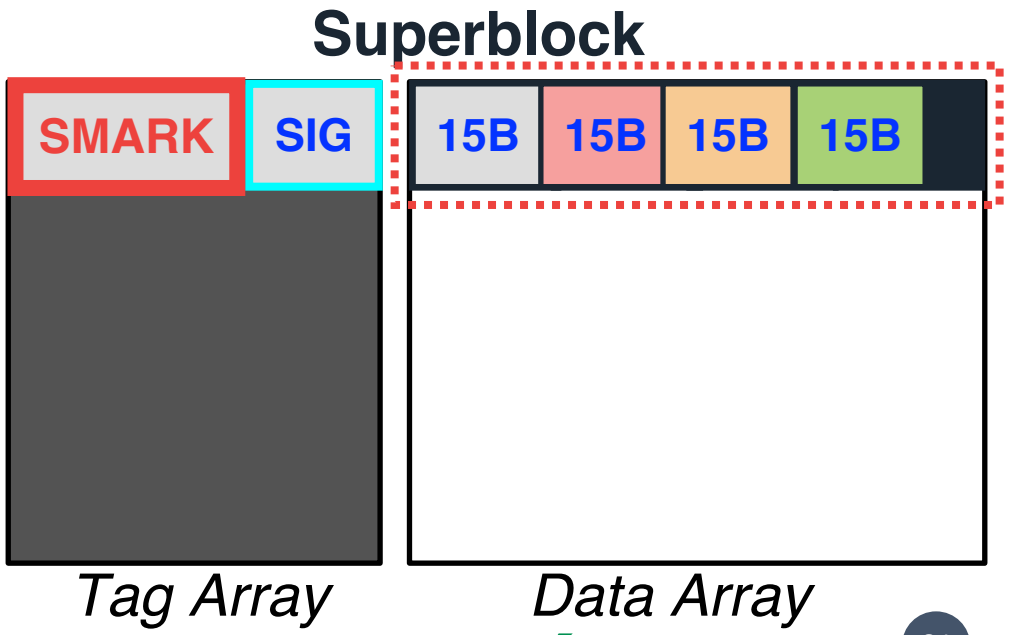
Superblock Marker (SMARK)

Key Idea

✓ To indicate the superblock, stores a random 16-bit marker (SMARK) in a tag entry

✓ Appends the signature to the marker

✓ The full tag is stored at the end of the cacheline



Touché

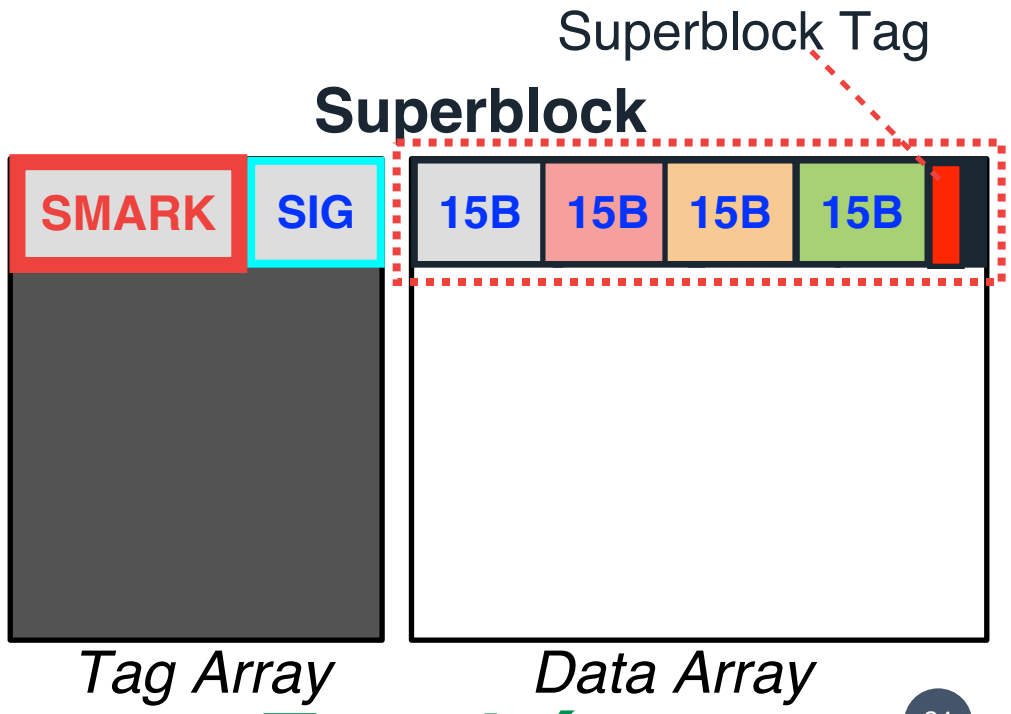
Superblock Marker (SMARK)

Key Idea

✓ To indicate the superblock, stores a random 16-bit marker (SMARK) in a tag entry

✓ Appends the signature to the marker

✓ The full tag is stored at the end of the cacheline

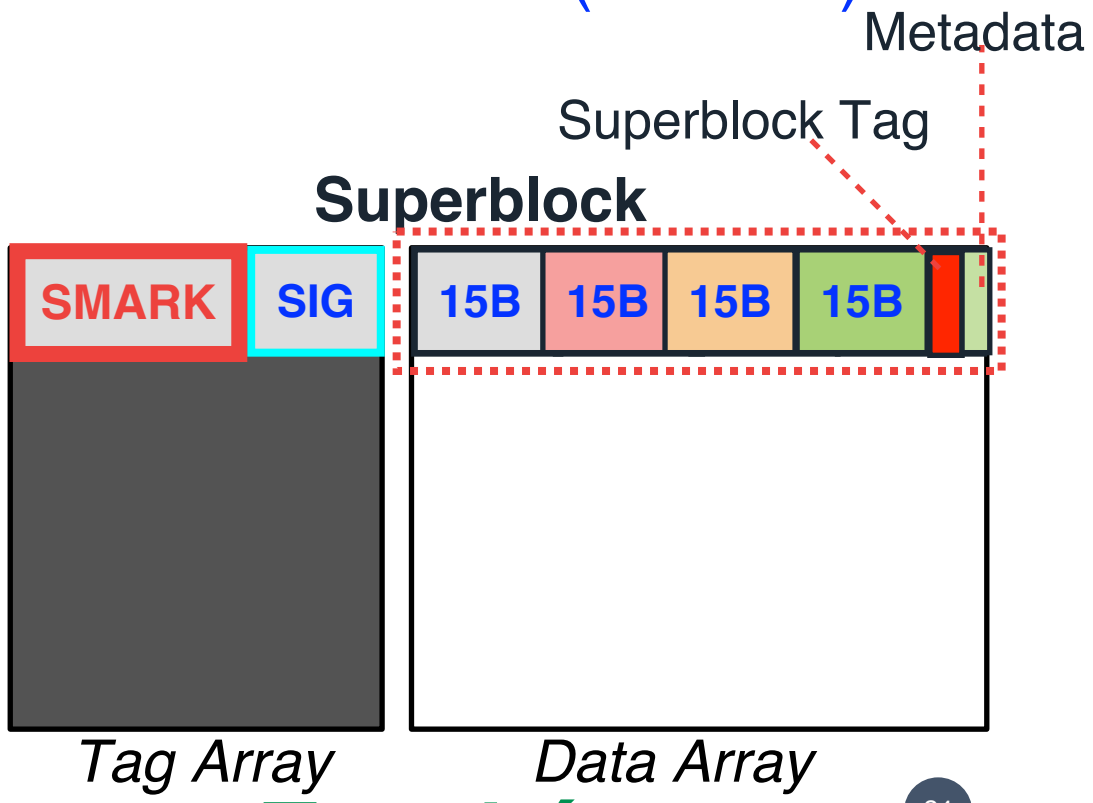


Touché

Superblock Marker (SMARK)

Key Idea

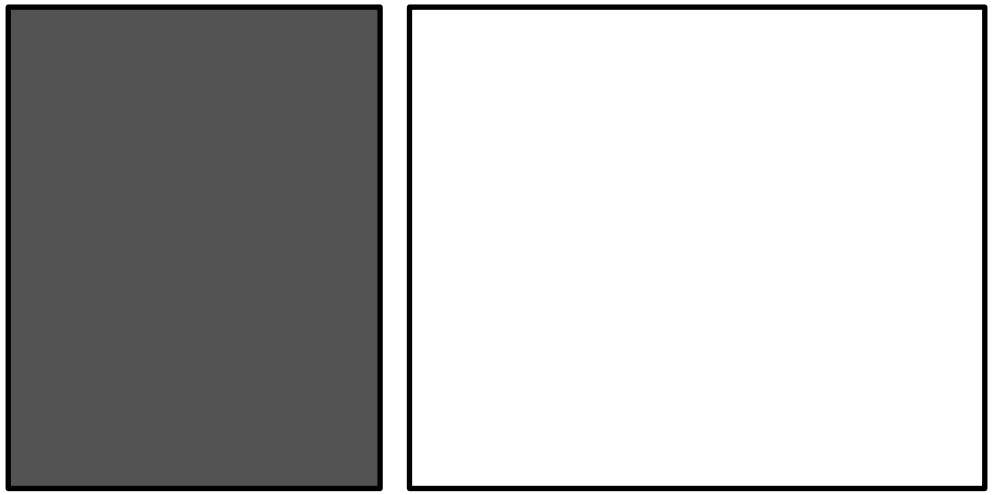
- ✓ To indicate the superblock, stores a random 16-bit marker (SMARK) in a tag entry
- ✓ Appends the signature to the marker
- ✓ The full tag is stored at the end of the cacheline



Touché

Touché = SIGN + TADA + SMARK

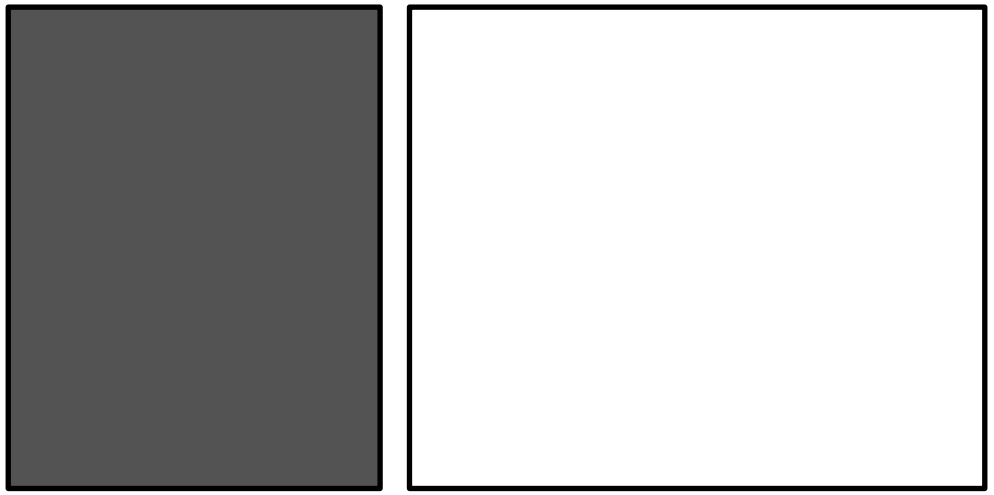
Using **“SIGN+TADA+SMARK”,** Touché enables to store



Touché

Touché = SIGN + TADA + SMARK

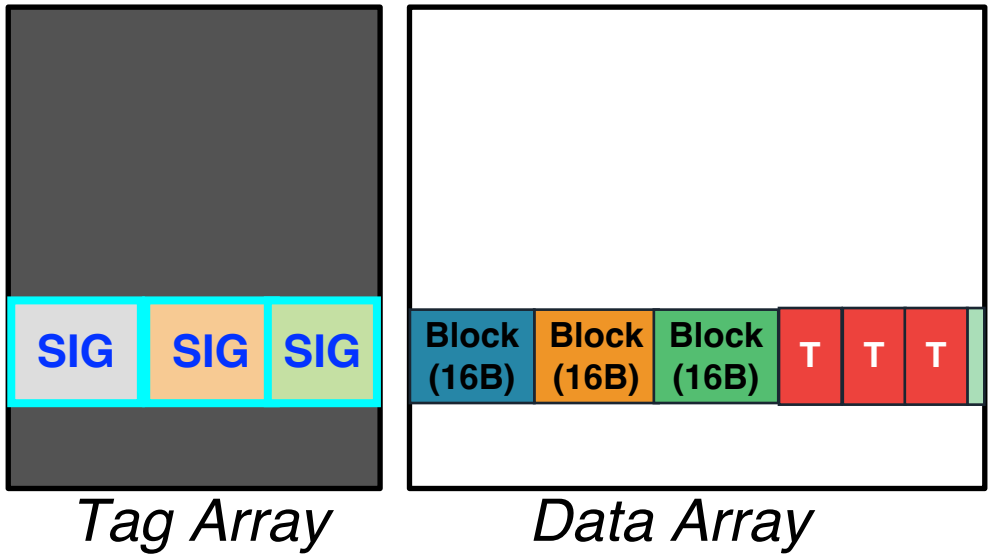
Using **“SIGN+TADA+SMARK”,** Touché enables to store
3 compressed blocks from <u>arbitrary addresses</u> or



Touché

Touché = SIGN + TADA + SMARK

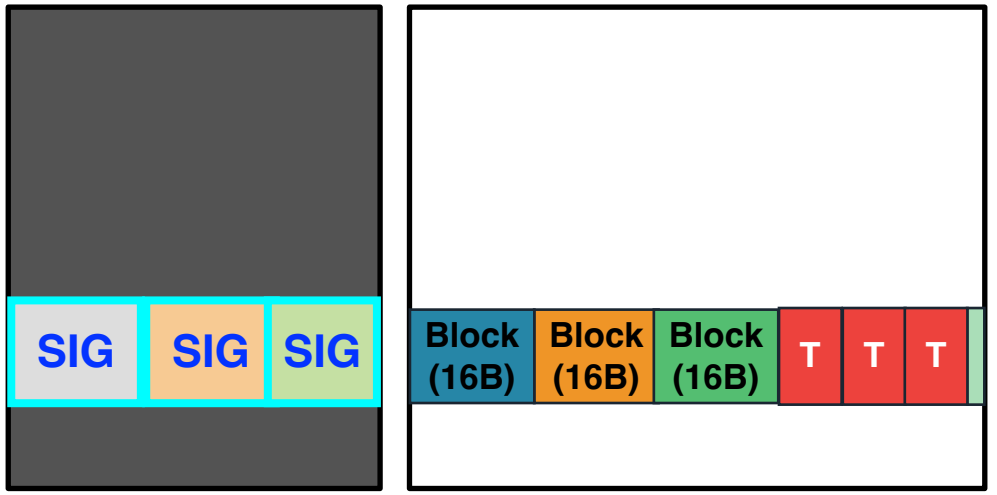
Using **“SIGN+TADA+SMARK”,** Touché enables to store
3 compressed blocks from <u>arbitrary addresses</u> or



Touché

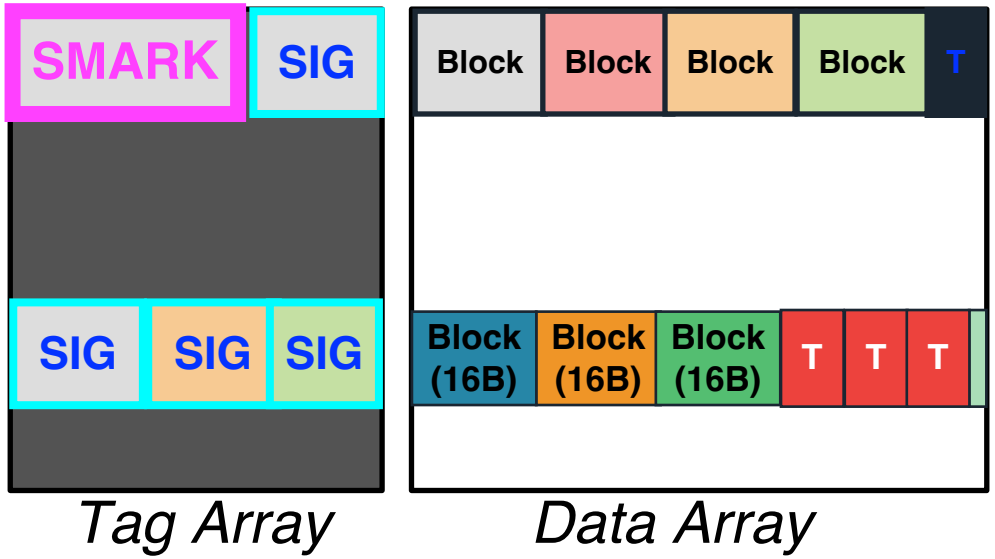
Touché = SIGN + TADA + SMARK

Using **“SIGN+TADA+SMARK”,** Touché enables to store
3 compressed blocks from arbitrary addresses or
4 compressed blocks from contiguous addresses



Touché = SIGN + TADA + SMARK

Using “**SIGN+TADA+SMARK**”, Touché enables to store
3 compressed blocks from arbitrary addresses or
4 compressed blocks from contiguous addresses

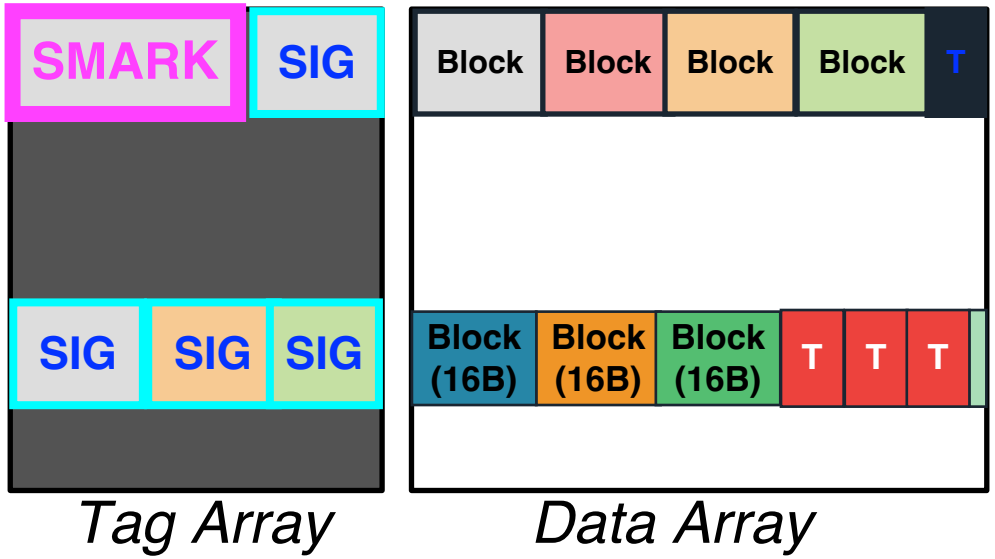


Touché

Touché = SIGN + TADA + SMARK

Using “**SIGN+TADA+SMARK**”, Touché enables to store
3 compressed blocks from arbitrary addresses or
4 compressed blocks from contiguous addresses

without any tag overheads!



Touché

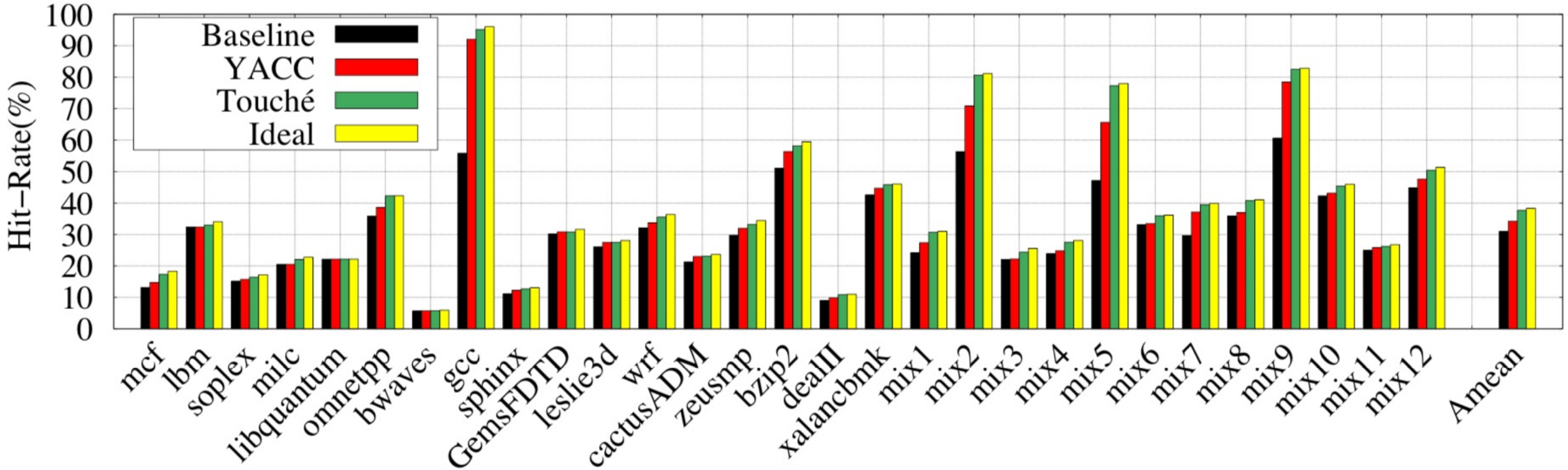
Experimental Methodology

✓ **Trace-based in-house cache simulator based on the USIMM**
- Detailed cache hierarchy
- Processor core models

✓ **Compression algorithms**
- BDI [G. Pekhimenko et al]
- FPC [Alaa R Alameldeen et al]

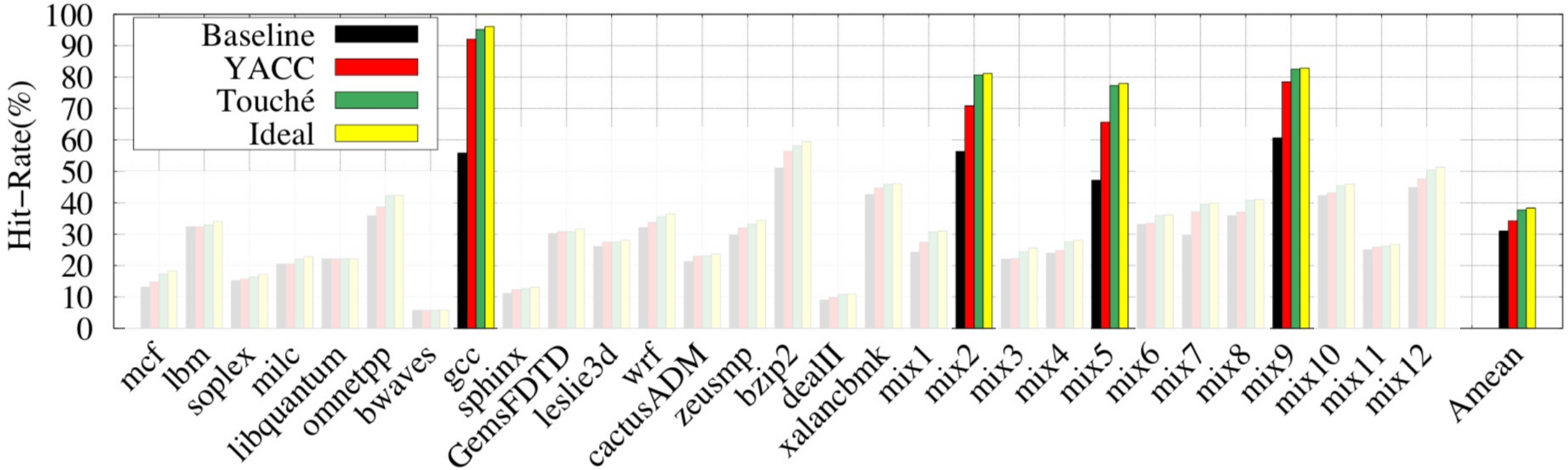
Number of Core **Processor Clock Frequency** **Issue Width**	4 3.2 GHz 8
L1 Cache (Private) **L2 Cache (Private)**	32KB, 8-way, 64B, 4 cycles 256KB, 8-way, 64B, 12 cycles
Last-level Cache **LLC Tag Access** **LLC Data Access**	4MB, 8-way, 64B 5 cycles 30 cycles
Memory Bus Frequency **Memory Channels** **Rank per Channel** **Bank Groups** **Banks per Bank Group** **Rows per Bank** **Columns per Row**	1600MHz (DDR 3200MHz) 2 1 4 4 64K 128
DRAM Access Timings: RCD-RP-CAS **DRAM Refresh Timing: RFC**	22-22-22 420ns

Effect on Last-level Cache Hit-Rate



Effect on Last-level Cache Hit-Rate

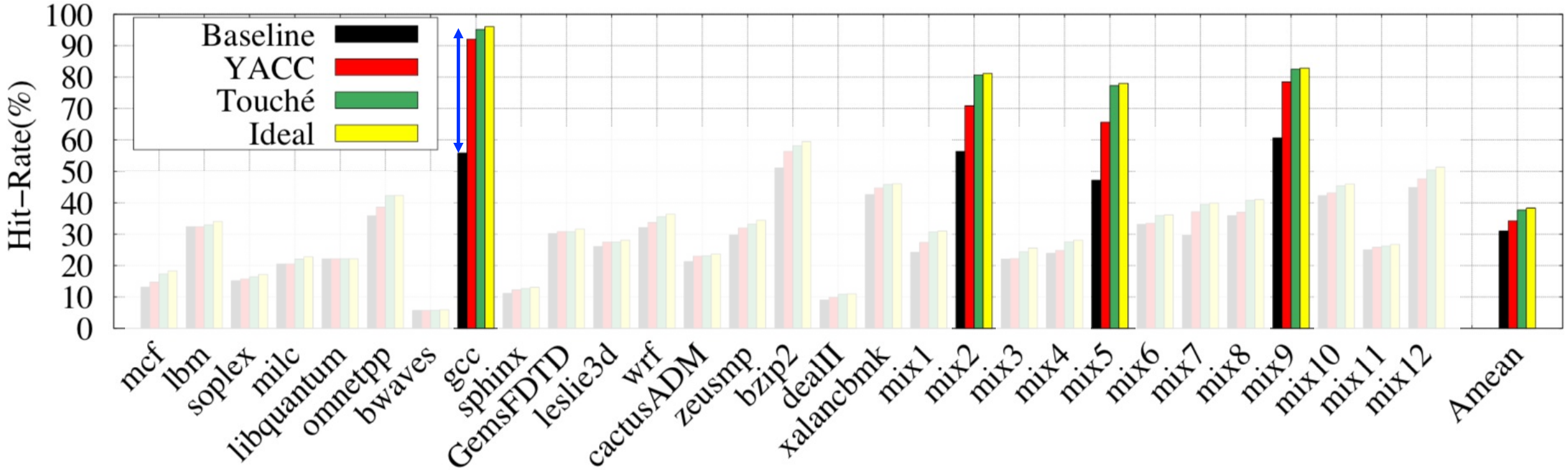
Touché achieves dramatic increase in the LLC Hit-Rate for the workloads extremely sensitive to the LLC capacity



Effect on Last-level Cache Hit-Rate

Touché achieves dramatic increase in the LLC Hit-Rate for the workloads extremely sensitive to the LLC capacity

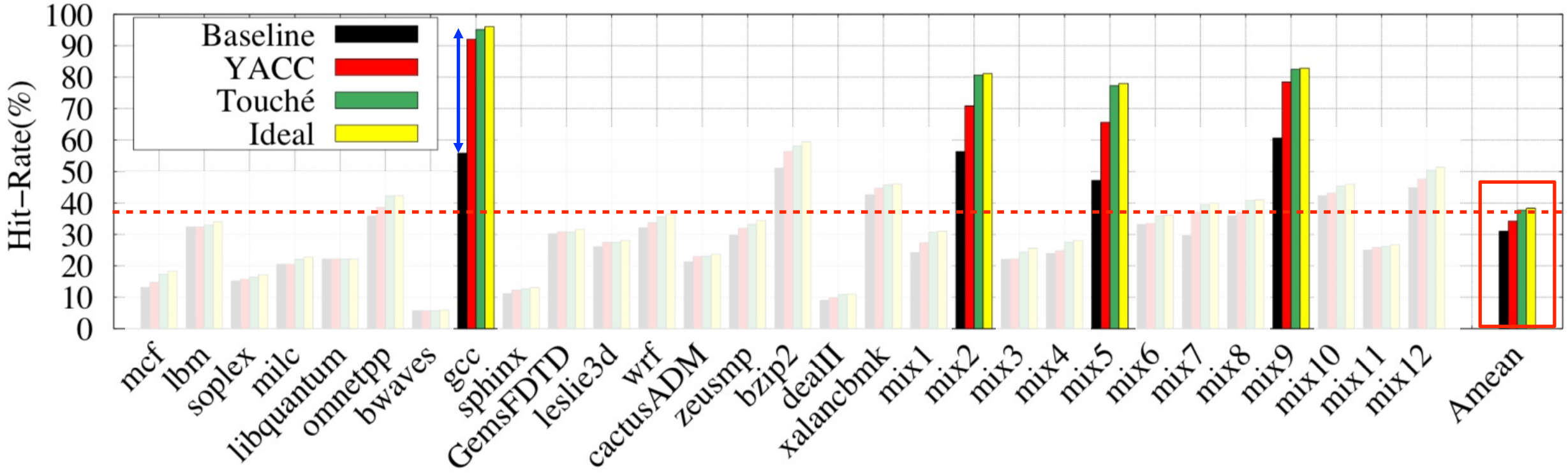
- The hit-rate of gcc increases from 55% to 95%!



Effect on Last-level Cache Hit-Rate

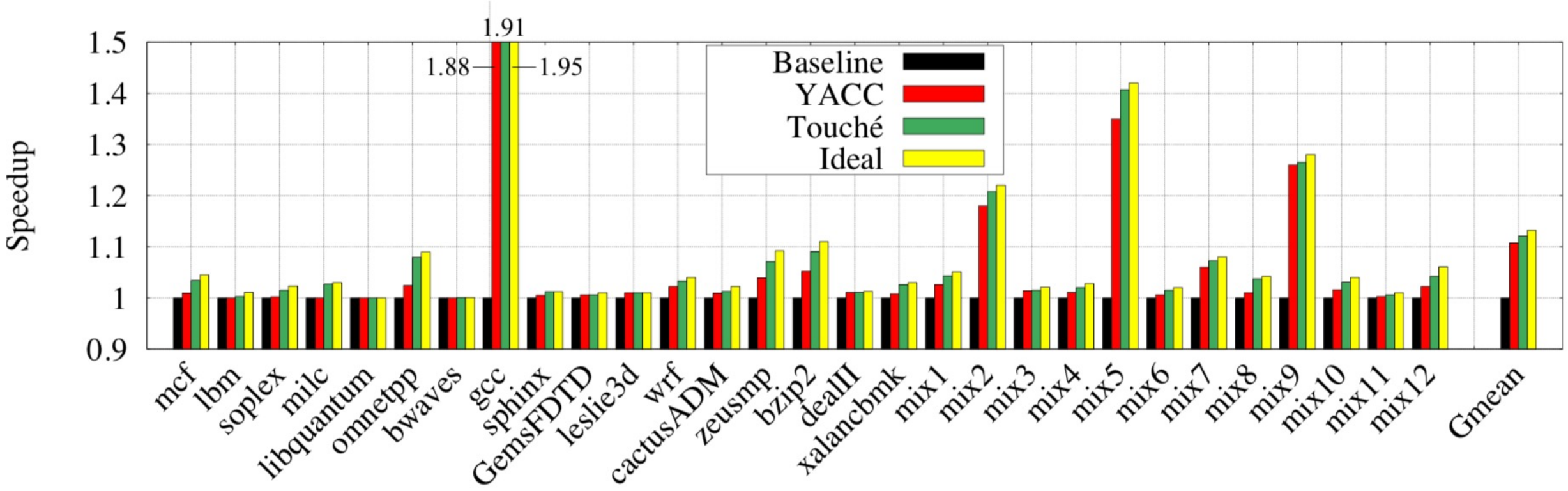
Touché achieves dramatic increase in the LLC Hit-Rate for the workloads extremely sensitive to the LLC capacity

- The hit-rate of gcc increases from 55% to 95%!



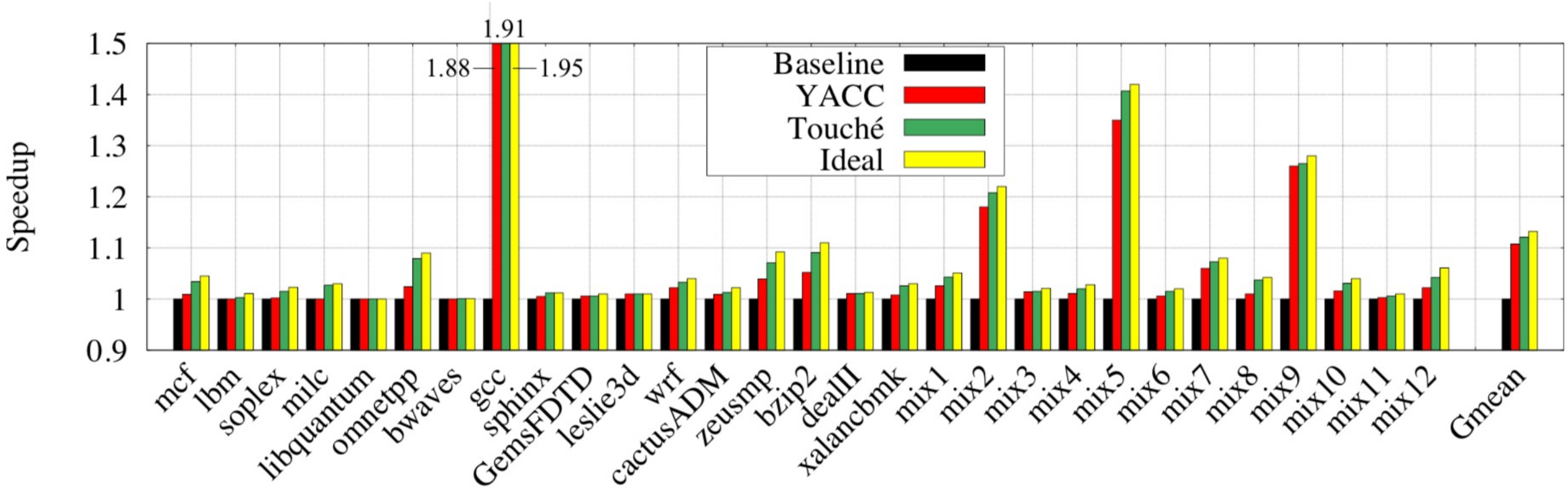
On average, Touché increases the hit rate by 6% (Ideal – 7%)

Performance Impact



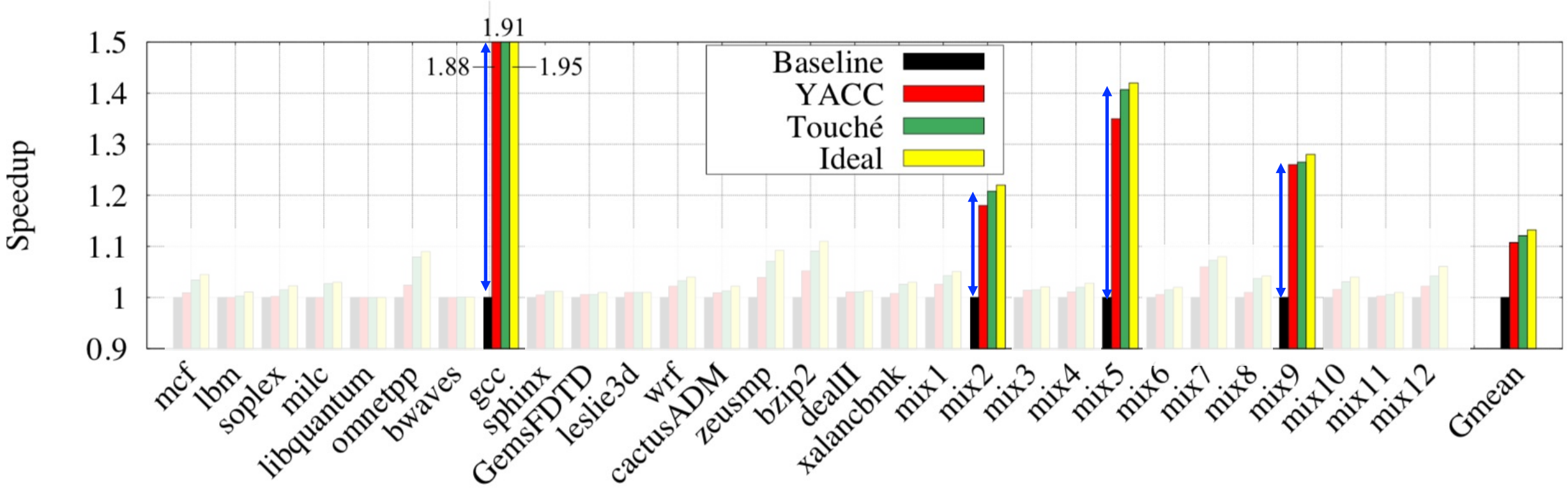
Performance Impact

Touché improves the performance significantly for workloads that are sensitive to the LLC capacity!



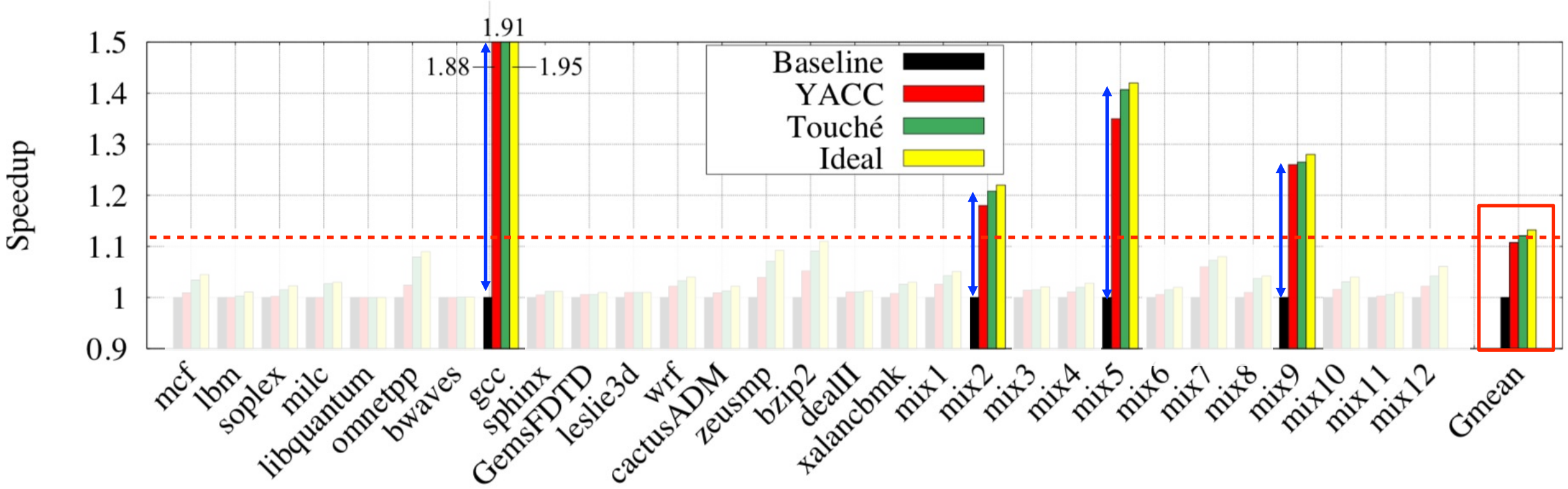
Performance Impact

Touché improves the performance significantly for workloads that are sensitive to the LLC capacity!



Performance Impact

Touché improves the performance significantly for workloads that are sensitive to the LLC capacity!



On average, Touché achieves a speedup of 12% (Ideal : 13%)

Summary

- Last-Level Cache (LLC) capacity/core has stagnated over the past decade.

Summary

- Last-Level Cache (LLC) capacity/core has stagnated over the past decade.
- Data compression is a promising technique to increase the effective capacity of LLC.

Summary

- Last-Level Cache (LLC) capacity/core has stagnated over the past decade.
- Data compression is a promising technique to increase the effective capacity of LLC.
- However, compressed blocks require additional tag entries.

Summary

- Last-Level Cache (LLC) capacity/core has stagnated over the past decade.
- Data compression is a promising technique to increase the effective capacity of LLC.
- However, compressed blocks require additional tag entries.
- **Touché** is a framework that enables LLC compression without any area overheads in the tag or data arrays.

Summary

- Last-Level Cache (LLC) capacity/core has stagnated over the past decade.
- Data compression is a promising technique to increase the effective capacity of LLC.
- However, compressed blocks require additional tag entries.
- **Touché** is a framework that enables LLC compression without any area overheads in the tag or data arrays.
- **Touché** is completely hardware based and achieves a near-ideal speedup of 12% without any area overheads.

Thanks!

"Eureka! Another breakthrough in compression technology!"