Implicit Memory Tagging: No-Overhead Memory Safety Using Alias-Free Tagged ECC

Michael B. Sullivan, Mohamed Tarek Ibn Ziad, Aamer Jaleel, Stephen W. Keckler
NVIDIA | ISCA 2023 Full Talk

What is Memory Safety?

- A program property that guarantees memory objects can only be accessed:
  1. Between their intended bounds
  2. During their lifetime

- Many programming languages (C/C++, CUDA/OpenACC) do **not** ensure memory safety.

What is a Memory Safety Violation?

- A program property that guarantees memory objects can only be accessed:

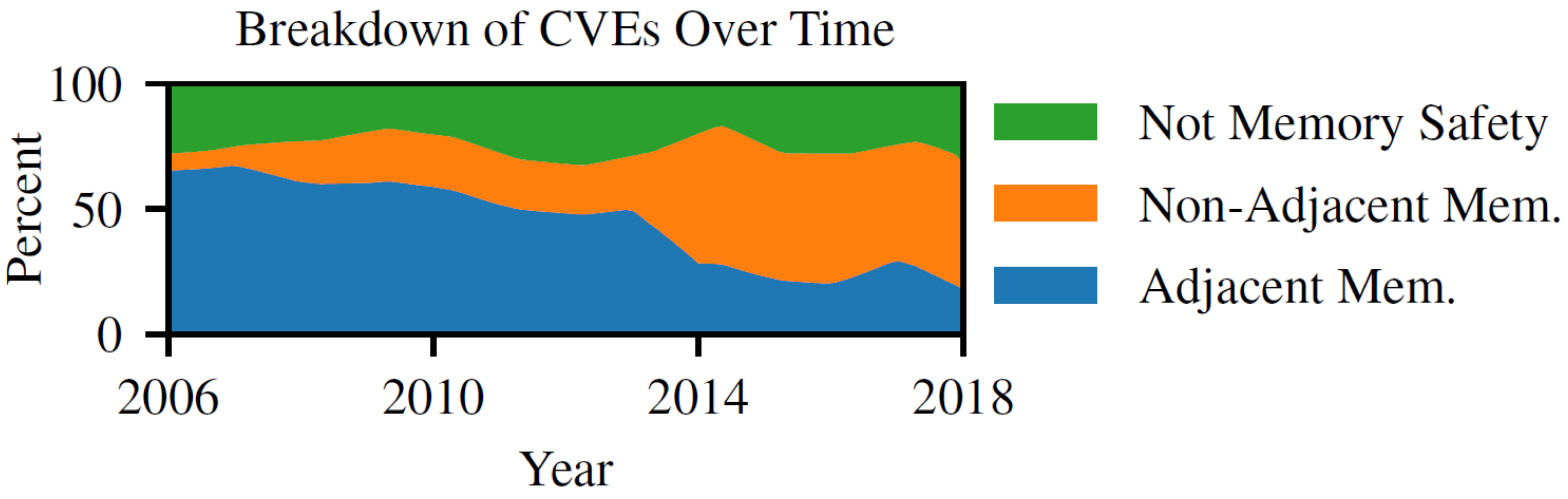
  1. ~~Between their intended bounds~~ Buffer Over-/Under-flow
  2. ~~During their lifetime~~ Use After Free

- Many programming languages (C/C++, CUDA/OpenACC) do **not** ensure memory safety.

- Memory safety violations are both a **correctness** and **security** issue
  - e.g., non-deterministic program output, buffer-overflow attacks

The Importance of Memory Safety

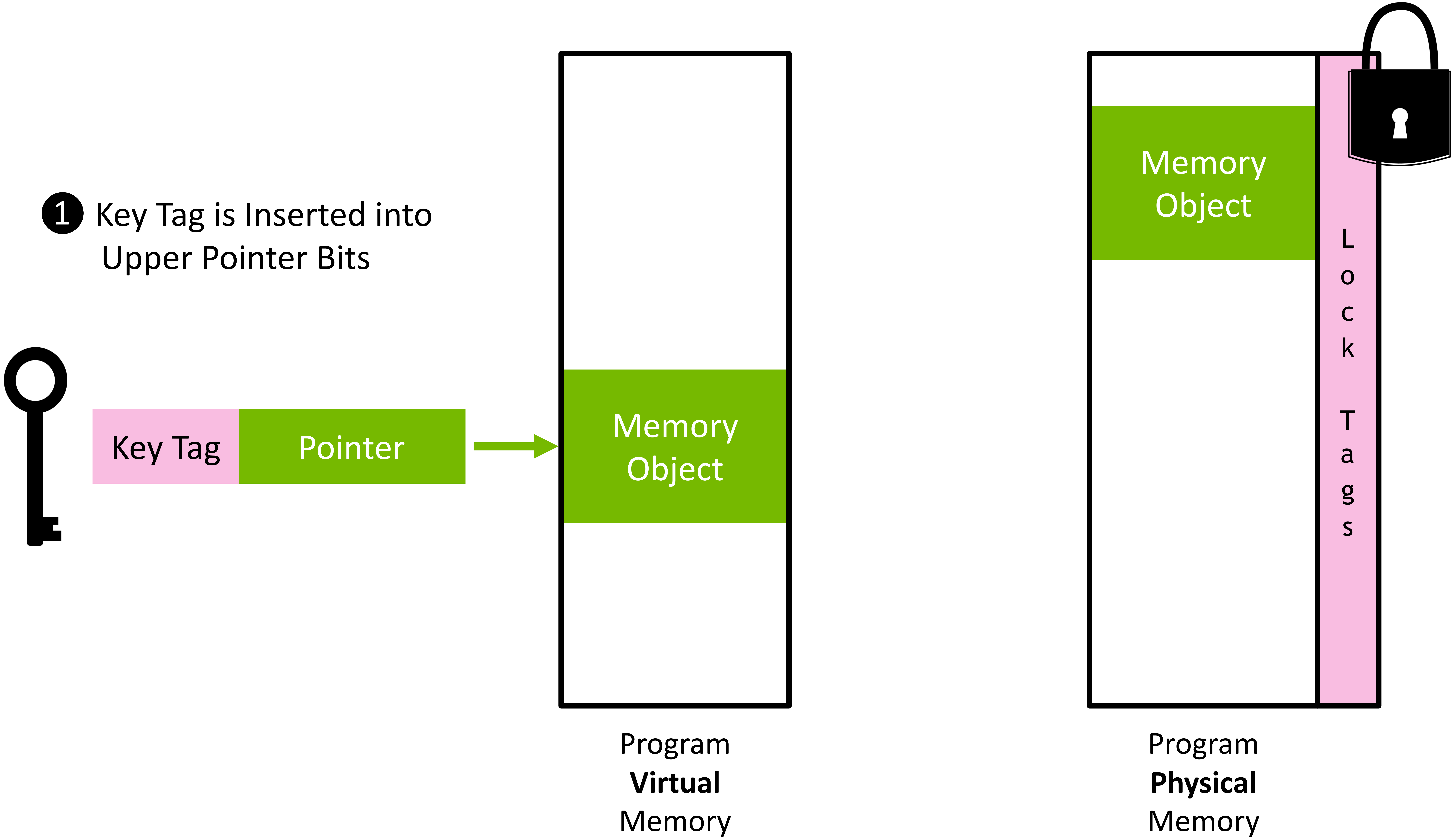
- Memory safety violations are remote attacks
- They are perhaps the most common security exploits
- E.g., from Microsoft’s Common Vulnerabilities and Exposures (CVE) database



~70% Memory Safety

Memory Tagging for Memory Safety

A mostly-hardware scheme to detect memory safety violations



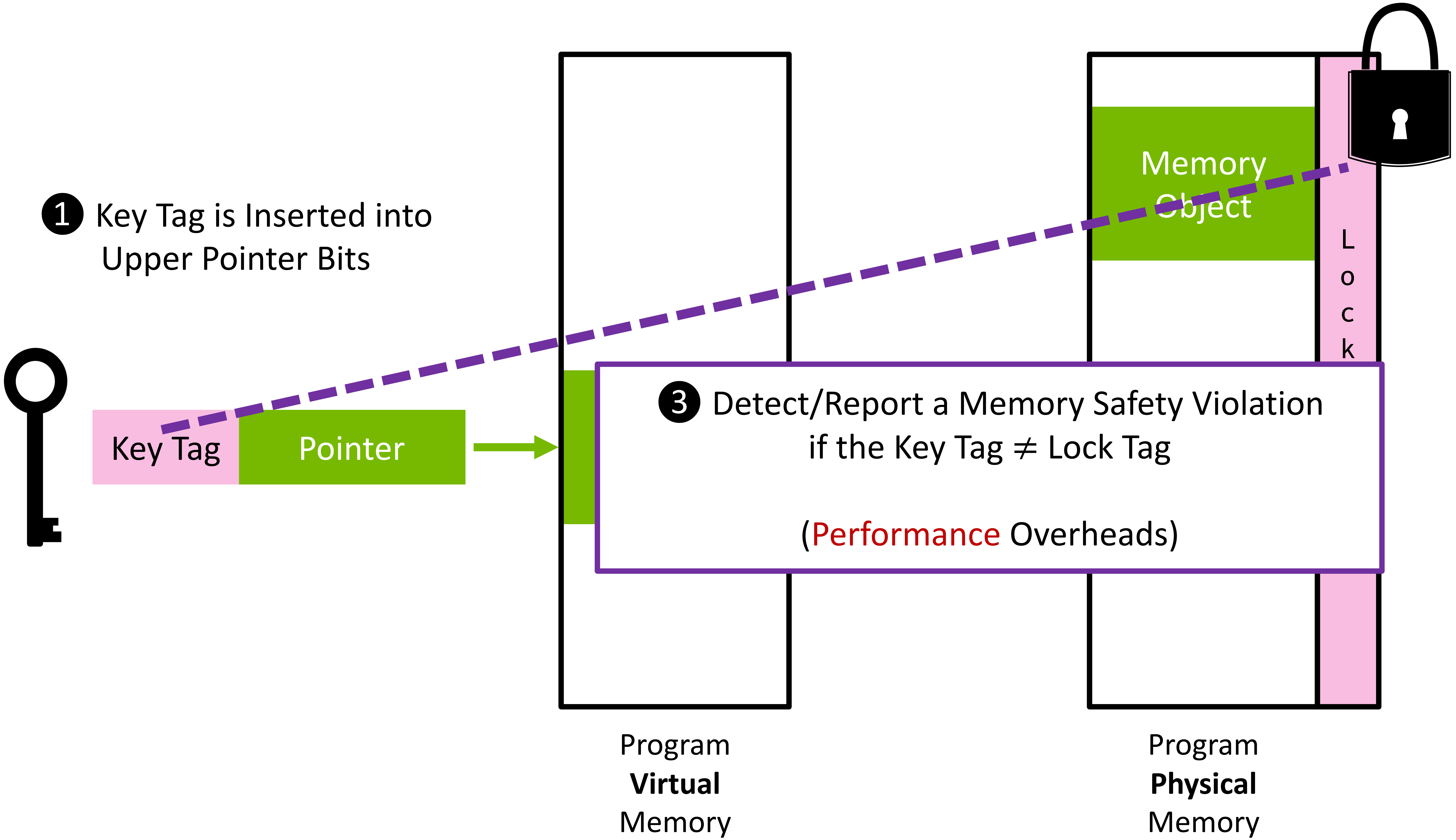
❶ Key Tag is Inserted into Upper Pointer Bits
Key Tag
Pointer
Memory Object
Program **Virtual** Memory
Memory Object
Lock Tags
Program **Physical** Memory

❷ Lock Tag associated with physical memory entries.

(Storage overheads.)

Memory Tagging for Memory Safety

A mostly-hardware scheme to detect memory safety violations



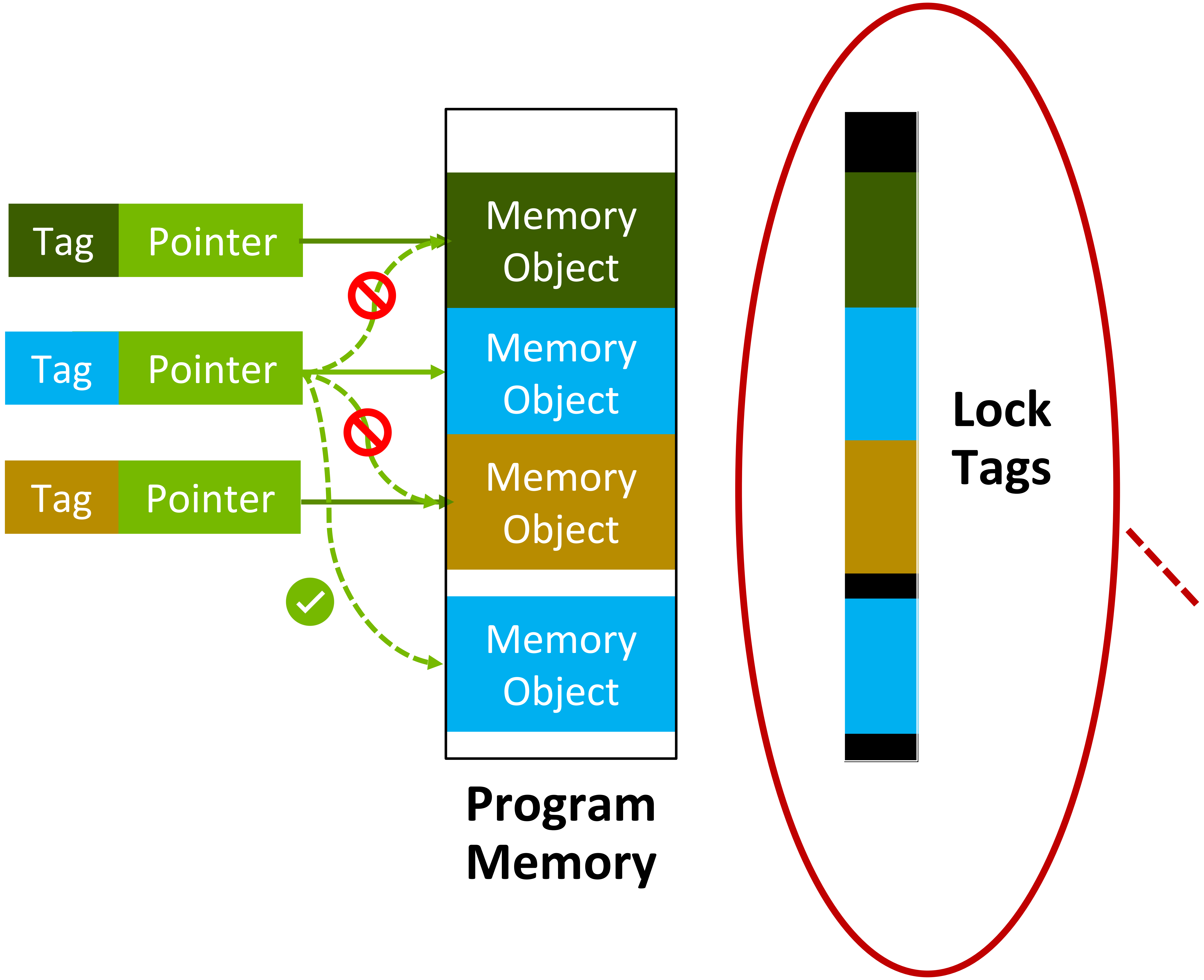
❶ Key Tag is Inserted into Upper Pointer Bits

❷ Lock Tag associated with physical memory entries.

❸ Detect/Report a Memory Safety Violation if the Key Tag ≠ Lock Tag

(Performance Overheads)

The Pros and Cons of Memory Tagging



+ Simple
+ Handles adjacent overflow
+ Handles non-adjacent overflow
+ Popular and Used in Industry (e.g., **SPARC ADI, ARM MTE**)

- Storage and movement of lock tag meta-data

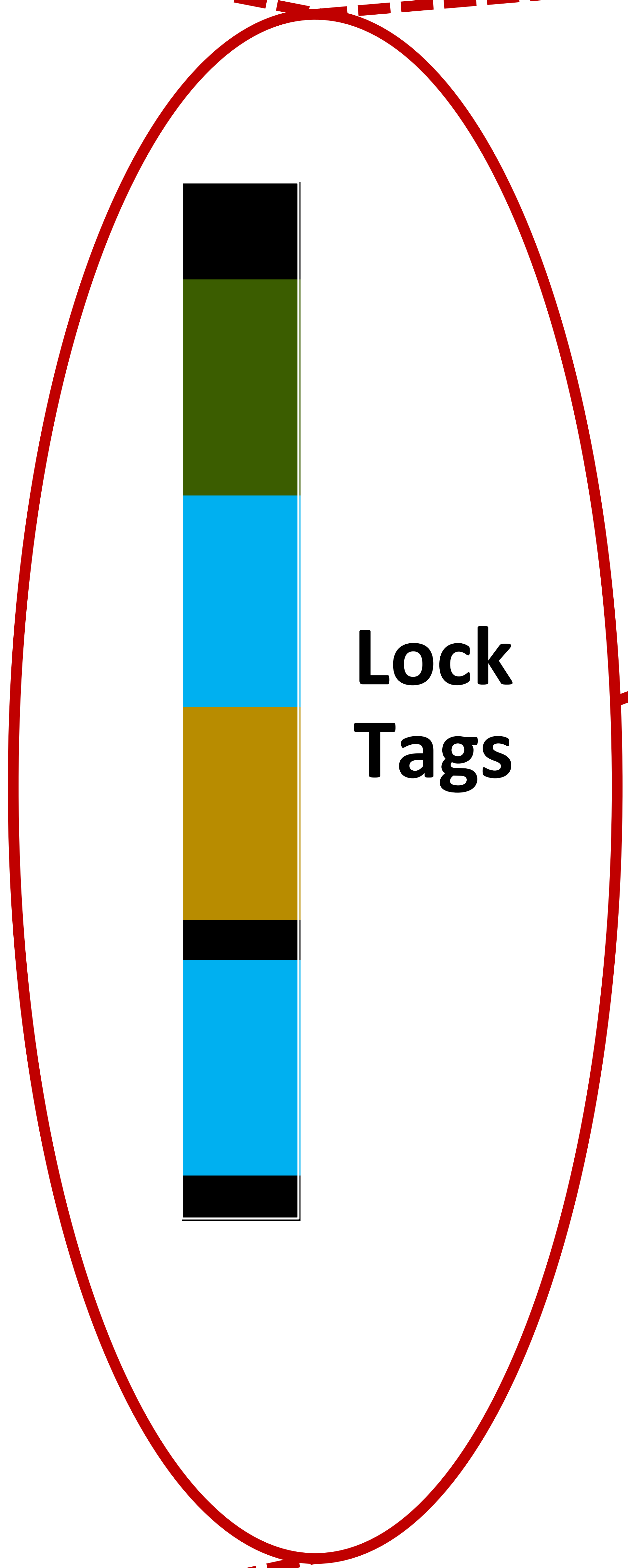
Tradeoff

- Probabilistic security for non-adjacent overflows

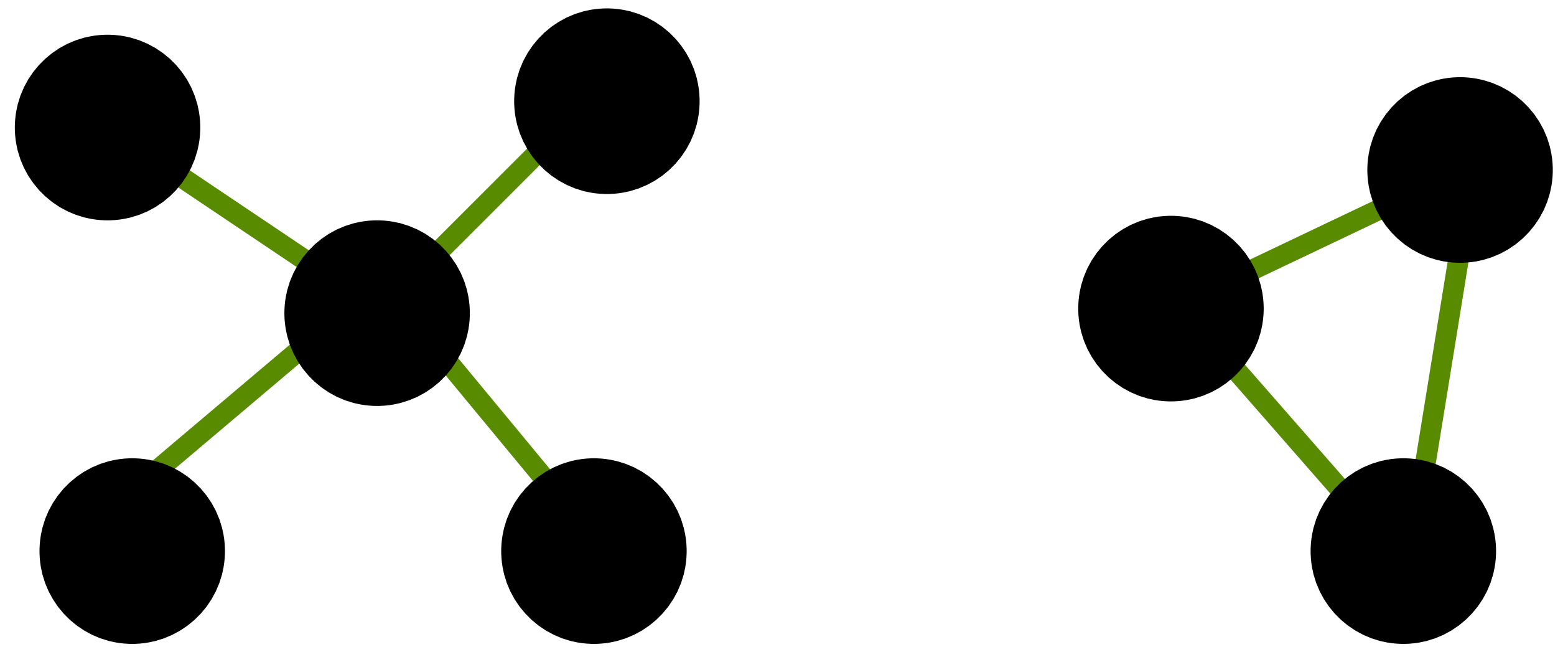
Memory Tagging on GPUs

Storage of lock tag meta-data is especially costly on capacity-limited GPUs!

e.g., An NVIDIA H100 GPU
(80GB HBM3 DRAM)



Movement of lock tag meta-data is especially costly for fine-grained access workloads
(e.g. graph, SPMV)



- Storage and movement of lock tag meta-data

Two Implementation Alternatives

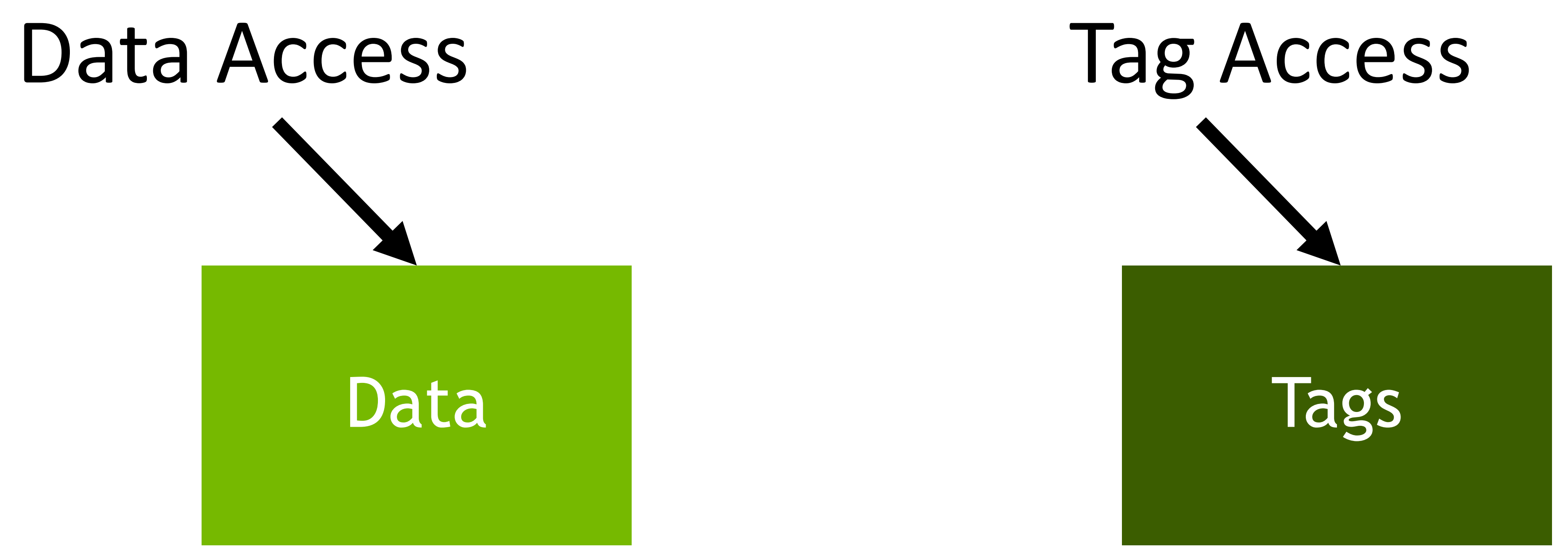
1) Tag Carve-Out, 2) ECC Stealing

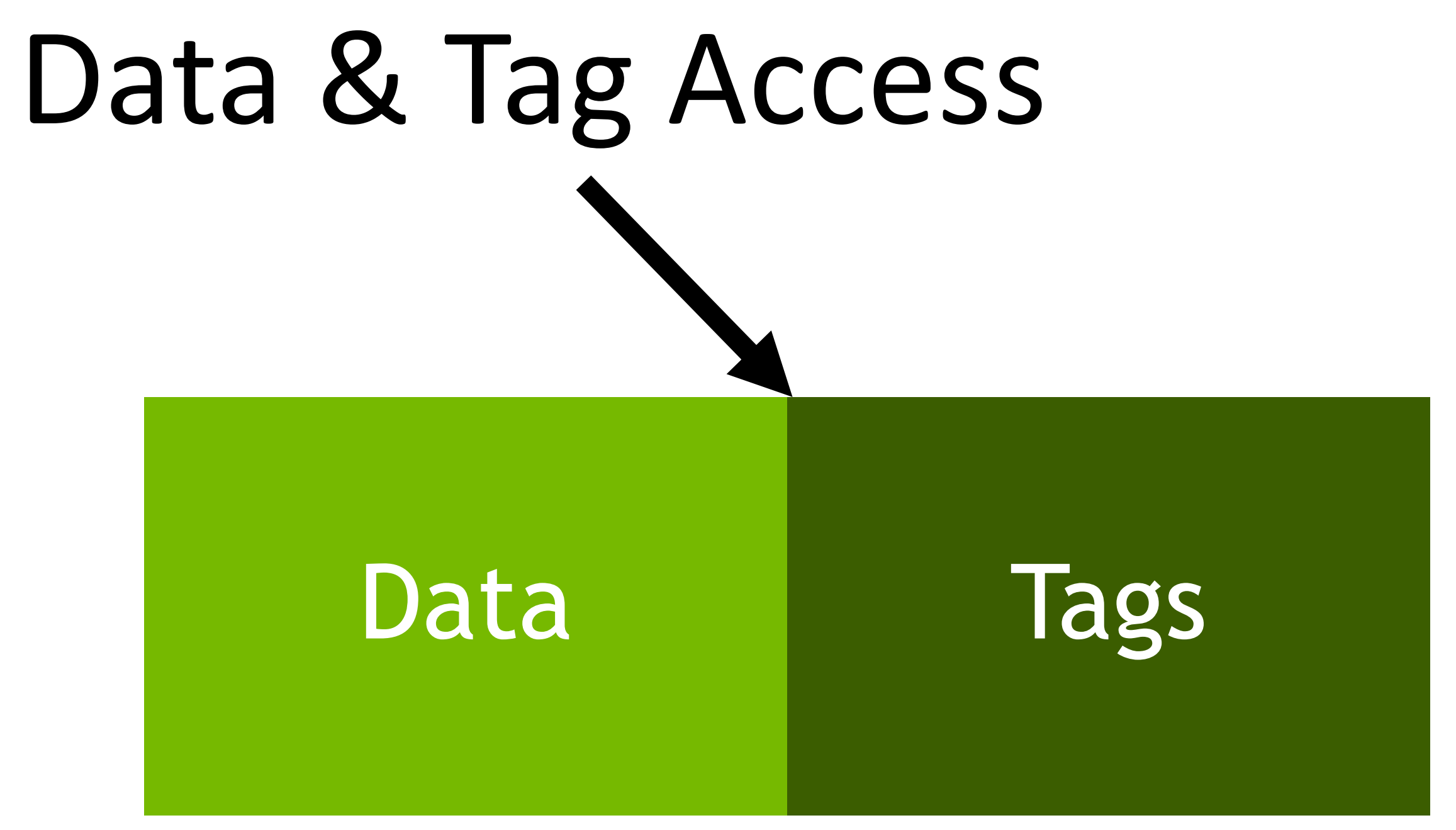
① Tag Carve-Out

- Meta-data in dedicated embedded carve-out
- Tags are cached once on-chip
- aka “Disjoint” tag storage [1]

② ECC Stealing

- Meta-data in dedicated sideband redundancy
- We assume this is taken from ECC redundancy
- aka “Widened” tag storage [1]





[1] Samuel Jero, Nathan Burow, Bryan Ward, Richard Skowyra, Roger Khazan, Howard Shrobe, and Hamed Okhravi. 2022.
TAG: Tagged Architecture Guide.
ACM Comput. Surv. 55, 6, Article 124 (June 2023), 34 pages.

Implementation Alternatives Pros and Cons

Benefits/Drawbacks to 1) Tag Carve-Out, 2) ECC Stealing

① Tag Carve-Out

- Meta-data in dedicated embedded carve-out
- Tags are cached once on-chip

+ Works on any underlying memory

- Storage overheads
- Tag movement overheads

② ECC Stealing

- Meta-data in dedicated sideband redundancy
- We assume this is taken from ECC redundancy

+ No storage overheads (above ECC alone)
+ No perf. overheads (above ECC alone)

- Greatly degraded reliability!

Slowdown of Embedded Tagging

Using cycle-accurate simulation

Memory Bandwidth Bound Programs with Poor Locality!

Slowdowns Across Three Workload Suites
Slowdown Over Untagged (%)
50%
40%
30%
20%
10%
0%
-10%
MLPerf
HPC & Sparse Linear Algebra (HPC+SLA)
Stream
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96
101
106
111
116
121
126
131
136
141
146
151
156
161
166
171
176
181
186
191
Workload #
Tag Carve-Out (Low-Tag-Storage)
Tag Carve-Out (High-Tag-Storage)
Max/Average
Average Slowdown (HMean)
50%
40%
30%
20%
10%
0%
-10%
Max (◆)
MLPerf
HPC+SLA
STREAM

TS=8b
TG=32B

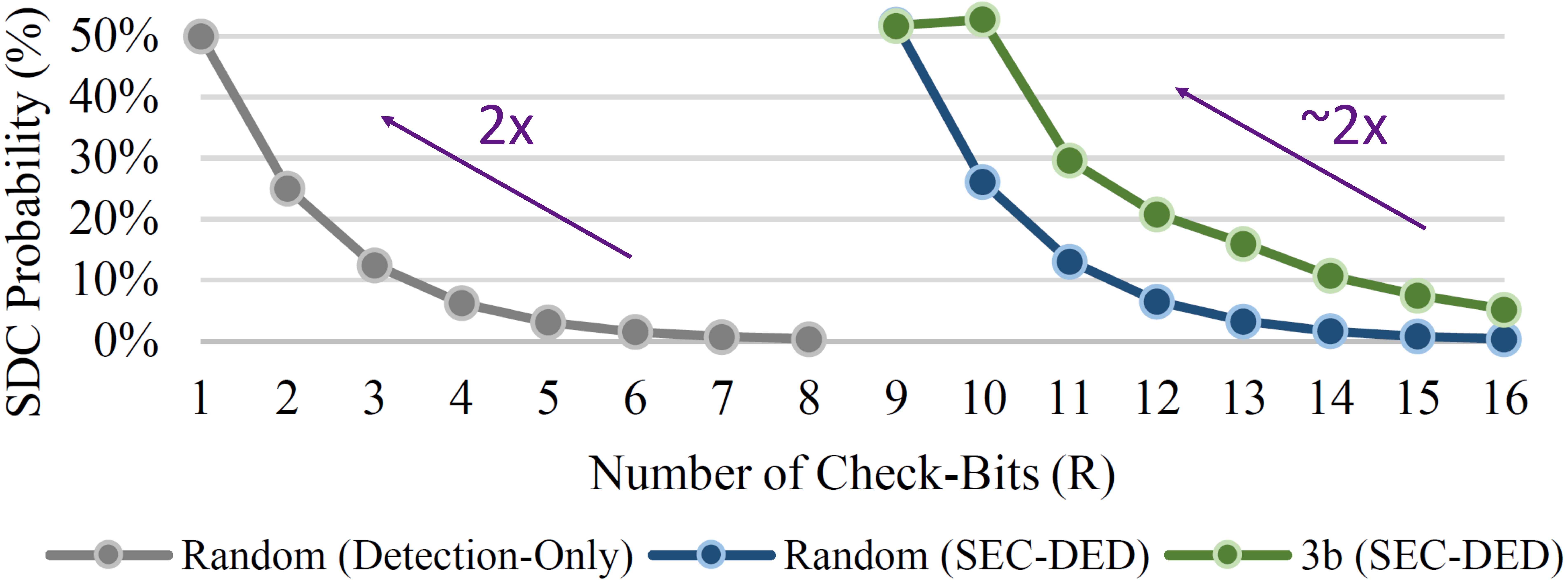
TS=16b
TG=32B

3.25% storage overheads
Up to ~30% slowdown

6.5% storage overheads
Up to ~40% slowdown

Stealing ECC Drastically Reduces Reliability

~2x SDC risk for every bit stolen… ☹



Tag Size vs Security

Larger Tags → Better Security

- Non-adjacent memory security increases with the number of unique valid tags, with detection rate:

$$100\% - \frac{100\%}{\text{Num.Tags}}$$

- Prior memory tagging approaches are limited to TS=4b, for performance and storage reasons. This limits the detection rate to $\leq {}^{15}/_{16}$

- Next, we show that Implicit Memory Tagging allows for **large tags** to be used **without performance, storage, or resilience concerns**, improving probabilistic security by 2 or 3 orders of magnitude.

❷ Implicit Memory Tagging (IMT):

No-Overhead Memory Safety
Using Alias-Free Tagged ECC

Alias-Free Tagged ECC

A general mechanism for tag equivalence checking in ECC

- ECC codes for memories are **shortened**, because of power-of-two sized data blocks.

- E.g., 10b SEC-DED could protect 501 data bits, but GPU memory accesses are 256b…

10b
256b
245b
ECC
Data
Shortening

Alias-Free Tagged ECC

A general mechanism for tag equivalence checking in ECC

- ECC codes for memories are **shortened**, because of power-of-two sized data blocks.
- E.g., 10b SEC-DED could protect 501 data bits, but GPU memory accesses are 256b…
- Alias-Free Tagged ECC uses the unused error correction capabilities for tag checking.
- Requires a minor-yet-careful redesign of the ECC code...

10b
256b
245b
ECC
Data
Tag
Shortening

Alias-Free Tagged ECC Main Take-Aways

100% tag mismatch detection, no false positives, maintains ECC detection and correction

1. **Unambiguous tag mismatch**: 100% of tag mismatches are detected (in the absence of a data error).

2. **Proper TMM attribution**: Tag mismatches are reported as-such.

3. **Preserving Single-Bit Error Correction**: Single-bit correcting ECC still operates as expected.

4. **Maximum Tag Size**: For most codeword sizes, up to a TS=R-1 is supported (R check-bits).

Alias-Free Tagged ECC Main Take-Aways

100% tag mismatch detection, no false positives, maintains ECC detection and correction

1. **Unambiguous tag mismatch**: 100% of tag mismatches are detected (in the absence of a data error).
2. **Proper TMM attribution**: Tag mismatches are reported as-such.
3. **Preserving Single-Bit Error Correction**: Single-bit correcting ECC still operates as expected.
4. **Maximum Tag Size**: For most codeword sizes, up to a TS=R-1 is supported (R check-bits).

See the paper for:

1. The derivation of the maximum possible tag size.
2. A principled method to construct maximum-tag-size alias-free tagged ECC codes.

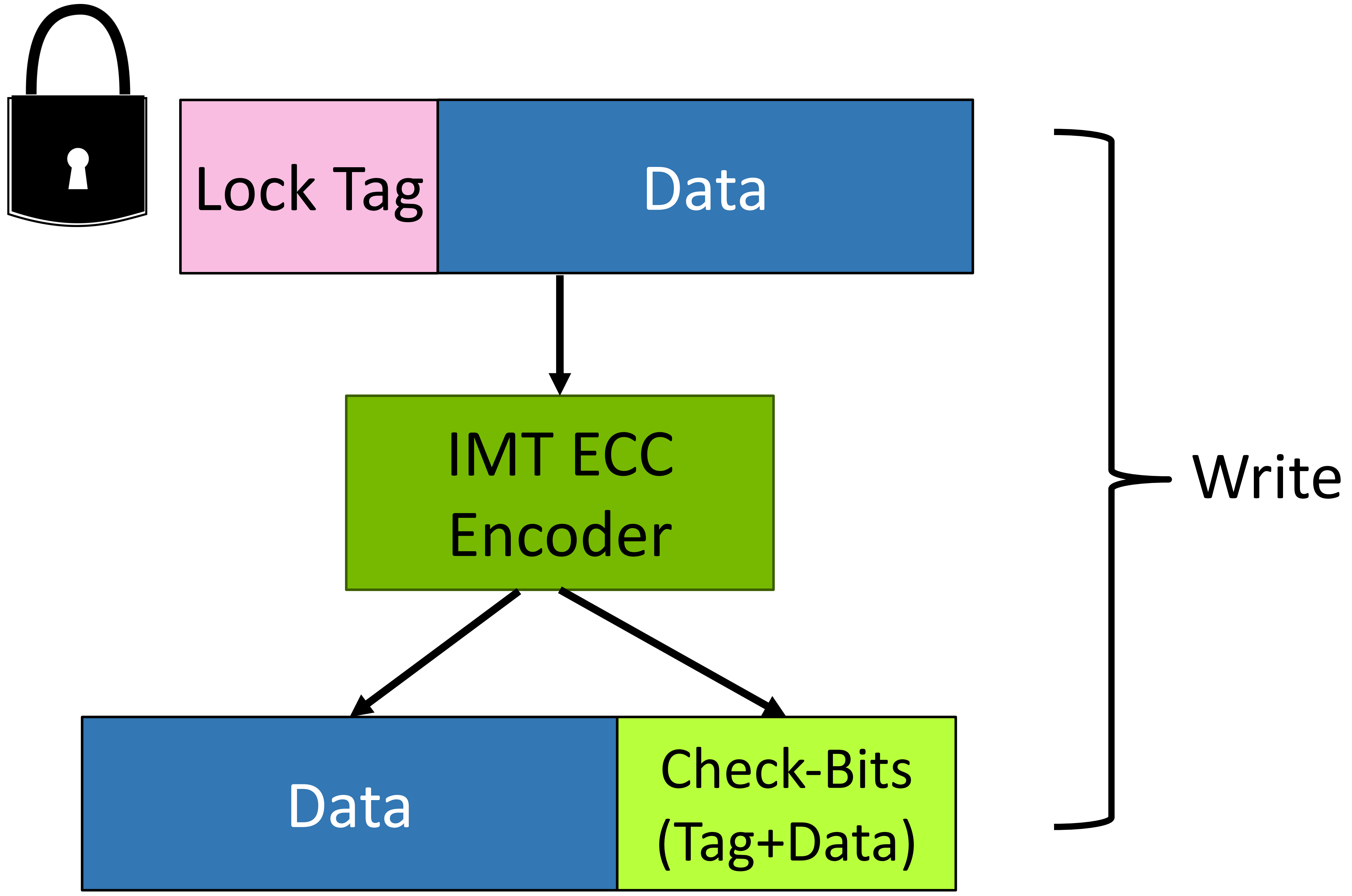
Implicit Memory Tagging: Main Idea Is Simple

Use Alias-Free Tagged ECC for memory tagging to ensure memory safety

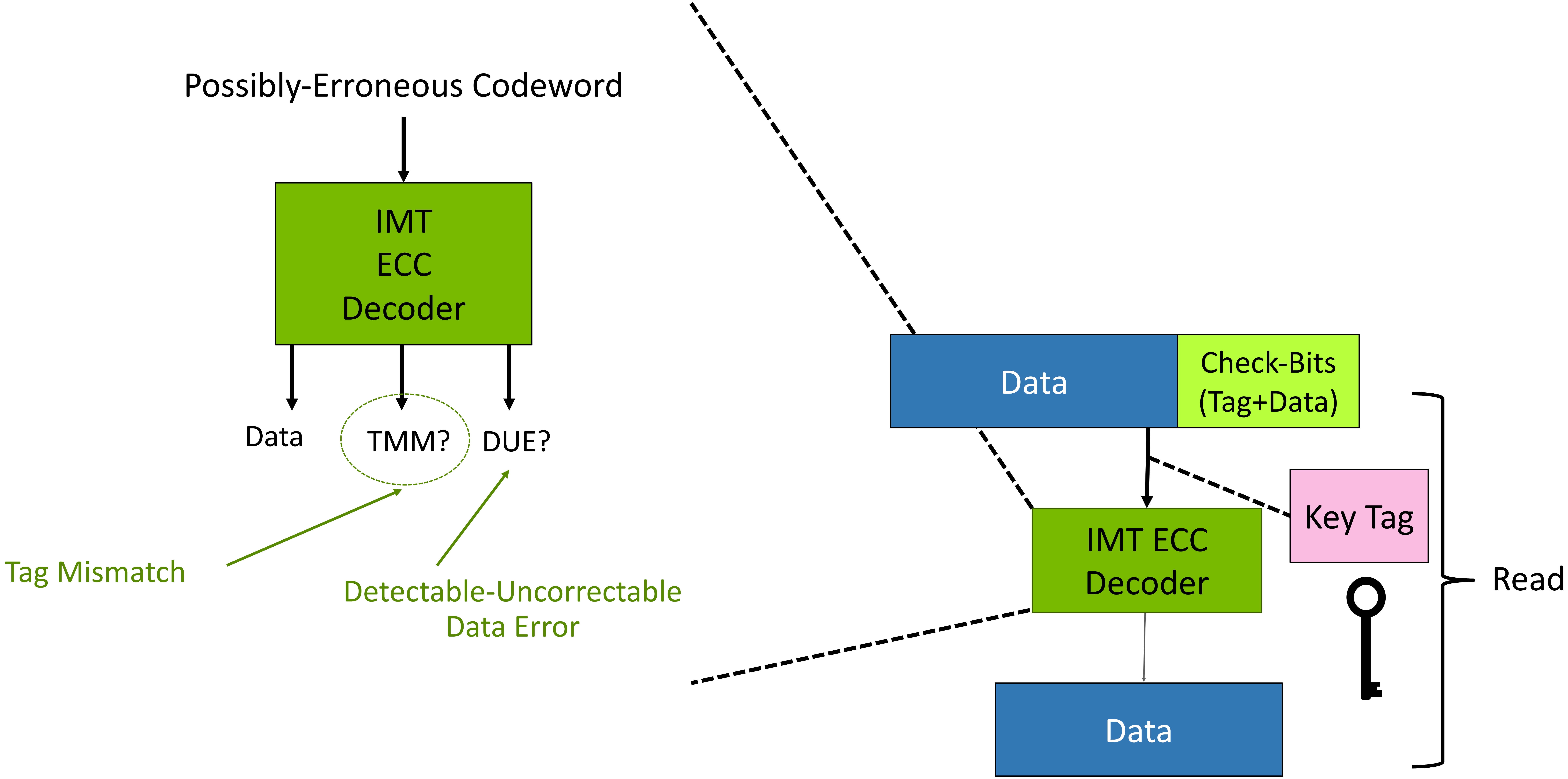
Implicit Memory Tagging: **Alias-Free Tagged ECC Applied to Memory Safety!**

- Overcomes the main limitations of Memory Tagging
- HW: identical to “ECC stealing”, but with modified ECC encoders/decoders
- **~0 storage overheads, ~0 perf. overheads, ~0 resilience regression, high security**

Implicit Memory Tag Checking Example



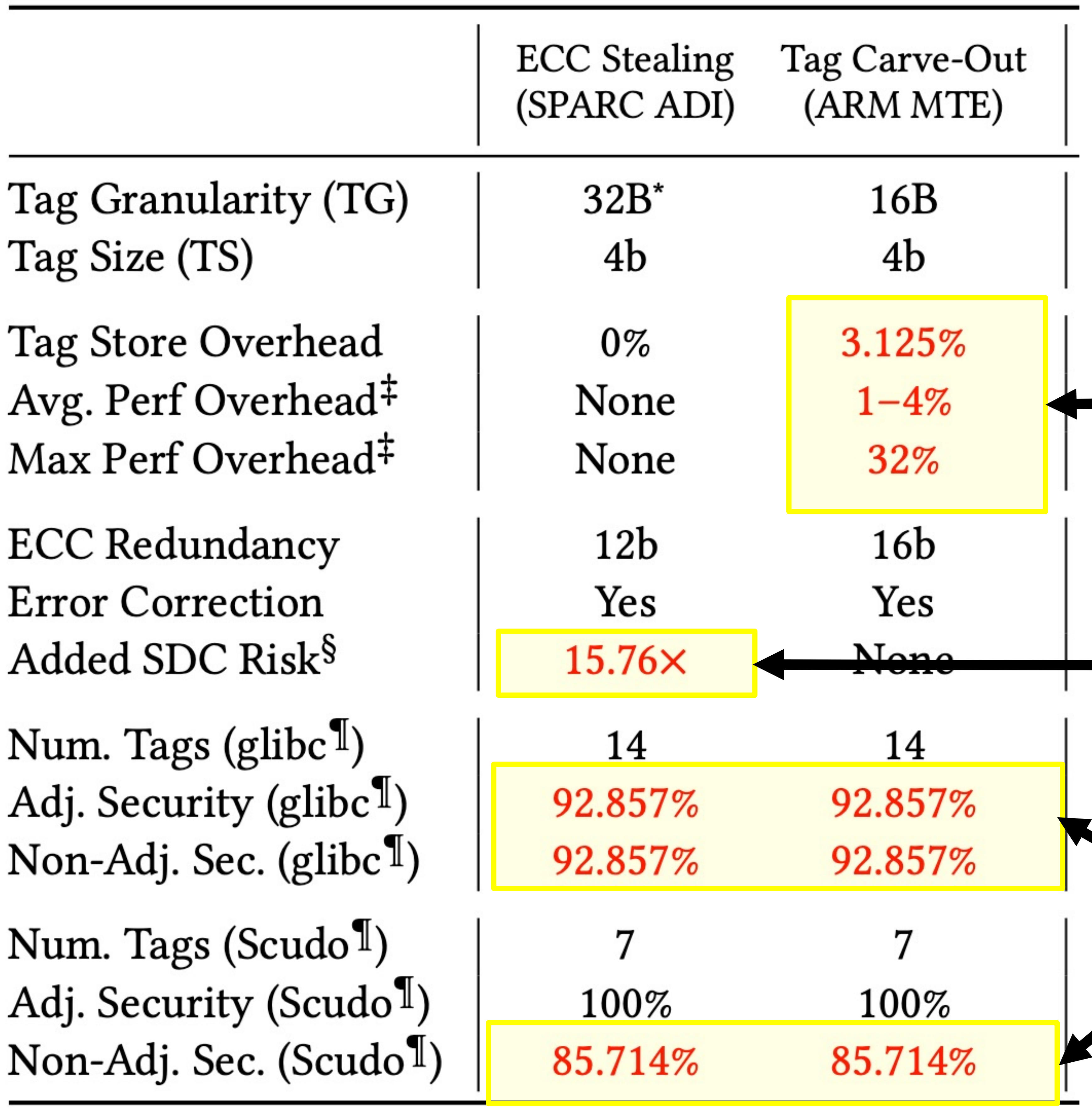
Implicit Memory Tag Checking Example



The Advantages of Implicit Memory Tagging

Superior performance, security, resilience, with no storage overheads

Table 1: A comparison of alternative memory tagging implementations.



	ECC Stealing (SPARC ADI)	Tag Carve-Out (ARM MTE)
Tag Granularity (TG)	32B*	16B
Tag Size (TS)	4b	4b
Tag Store Overhead	0%	3.125%
Avg. Perf Overhead‡	None	1–4%
Max Perf Overhead‡	None	32%
ECC Redundancy	12b	16b
Error Correction	Yes	Yes
Added SDC Risk§	15.76×	None
Num. Tags (glibc¶)	14	14
Adj. Security (glibc¶)	92.857%	92.857%
Non-Adj. Sec. (glibc¶)	92.857%	92.857%
Num. Tags (Scudo¶)	7	7
Adj. Security (Scudo¶)	100%	100%
Non-Adj. Sec. (Scudo¶)	85.714%	85.714%

Baselines

Tag Carve-Out:
Storage & Performance
Overheads

ECC Stealing:
Degraded Reliability

Both:
Weak Probabilistic Security

The Advantages of Implicit Memory tagging

Superior performance, security, resilience, with no storage overheads

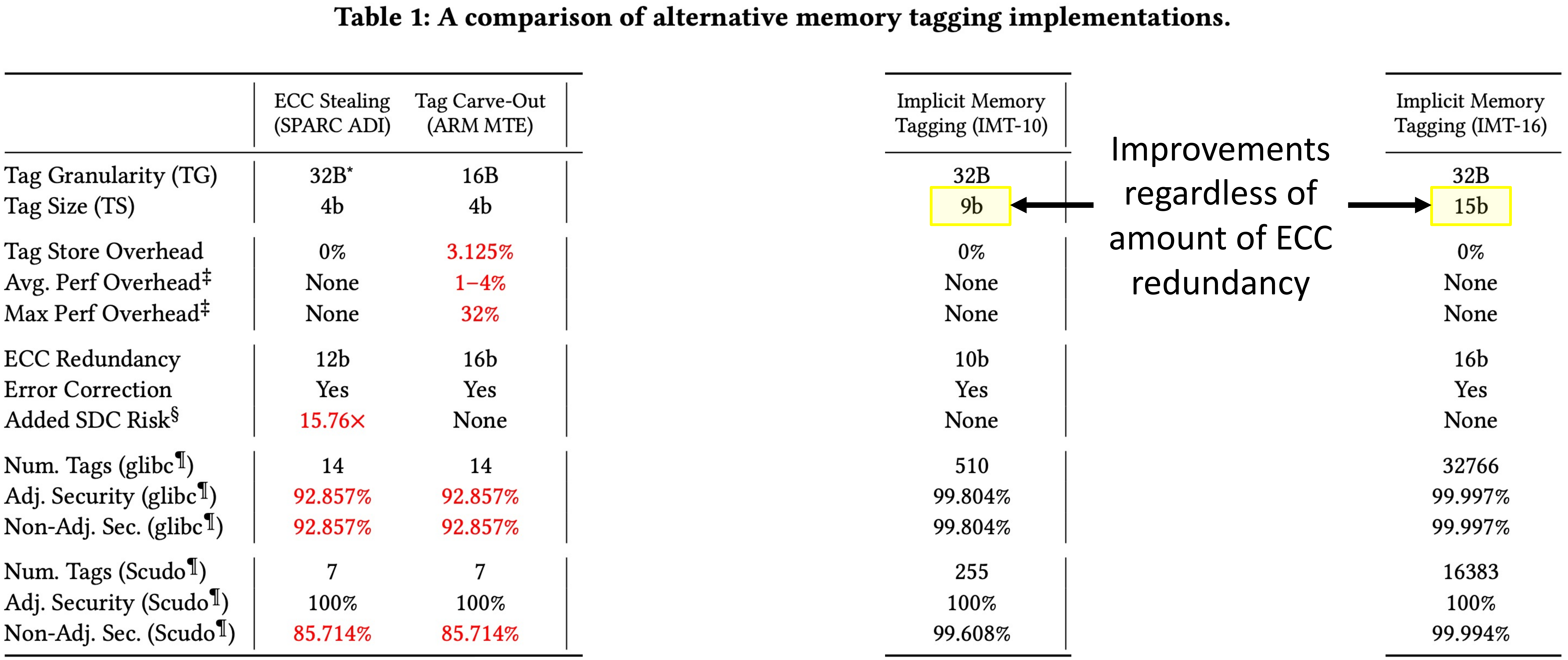


Table 1: A comparison of alternative memory tagging implementations.

	ECC Stealing (SPARC ADI)	Tag Carve-Out (ARM MTE)	Implicit Memory Tagging (IMT-10)	Implicit Memory Tagging (IMT-16)
Tag Granularity (TG)	32B*	16B	32B	32B
Tag Size (TS)	4b	4b	9b	15b
Tag Store Overhead	0%	3.125%	0%	0%
Avg. Perf Overhead‡	None	1–4%	None	None
Max Perf Overhead‡	None	32%	None	None
ECC Redundancy	12b	16b	10b	16b
Error Correction	Yes	Yes	Yes	Yes
Added SDC Risk§	15.76×	None	None	None
Num. Tags (glibc¶)	14	14	510	32766
Adj. Security (glibc¶)	92.857%	92.857%	99.804%	99.997%
Non-Adj. Sec. (glibc¶)	92.857%	92.857%	99.804%	99.997%
Num. Tags (Scudo¶)	7	7	255	16383
Adj. Security (Scudo¶)	100%	100%	100%	100%
Non-Adj. Sec. (Scudo¶)	85.714%	85.714%	99.608%	99.994%

Improvements regardless of amount of ECC redundancy

Baselines

IMT-10: 10b ECC, minimum SEC-DED

IMT-16: 16b ECC, same as GPU DRAM

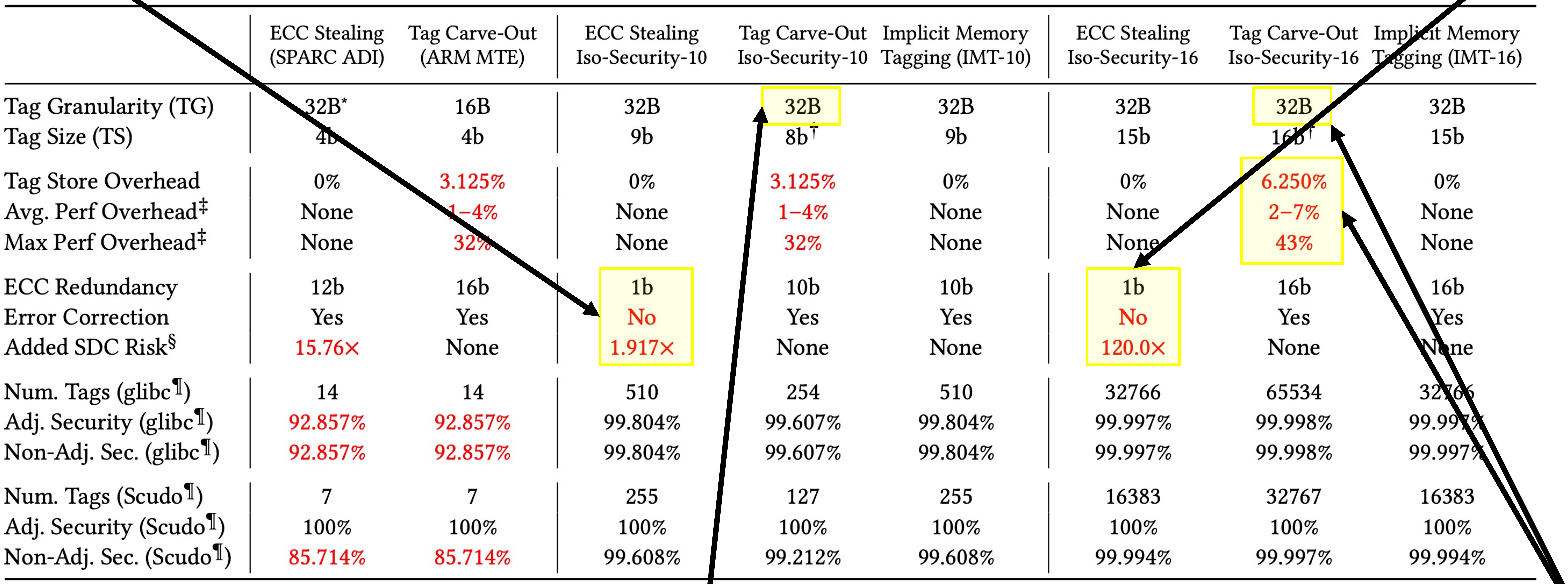
The Advantages of Implicit Memory tagging

Superior performance, security, resilience, with no storage overheads

No Error Correction

No Error Correction and Large SDC Risk

Table 1: A comparison of alternative memory tagging implementations.



	ECC Stealing (SPARC ADI)	Tag Carve-Out (ARM MTE)	ECC Stealing Iso-Security-10	Tag Carve-Out Iso-Security-10	Implicit Memory Tagging (IMT-10)	ECC Stealing Iso-Security-16	Tag Carve-Out Iso-Security-16	Implicit Memory Tagging (IMT-16)
Tag Granularity (TG)	32B*	16B	32B	32B	32B	32B	32B	32B
Tag Size (TS)	4b	4b	9b	8b†	9b	15b	16b†	15b
Tag Store Overhead	0%	3.125%	0%	3.125%	0%	0%	6.250%	0%
Avg. Perf Overhead‡	None	1–4%	None	1–4%	None	None	2–7%	None
Max Perf Overhead‡	None	32%	None	32%	None	None	43%	None
ECC Redundancy	12b	16b	1b	10b	10b	1b	16b	16b
Error Correction	Yes	Yes	No	Yes	Yes	No	Yes	Yes
Added SDC Risk§	15.76×	None	1.917×	None	None	120.0×	None	None
Num. Tags (glibc¶)	14	14	510	254	510	32766	65534	32766
Adj. Security (glibc¶)	92.857%	92.857%	99.804%	99.607%	99.804%	99.997%	99.998%	99.997%
Non-Adj. Sec. (glibc¶)	92.857%	92.857%	99.804%	99.607%	99.804%	99.997%	99.998%	99.997%
Num. Tags (Scudo¶)	7	7	255	127	255	16383	32767	16383
Adj. Security (Scudo¶)	100%	100%	100%	100%	100%	100%	100%	100%
Non-Adj. Sec. (Scudo¶)	85.714%	85.714%	99.608%	99.212%	99.608%	99.994%	99.997%	99.994%

Baselines

Iso-Security (10b ECC, minimum SEC-DED)

Iso-Security (16b ECC)

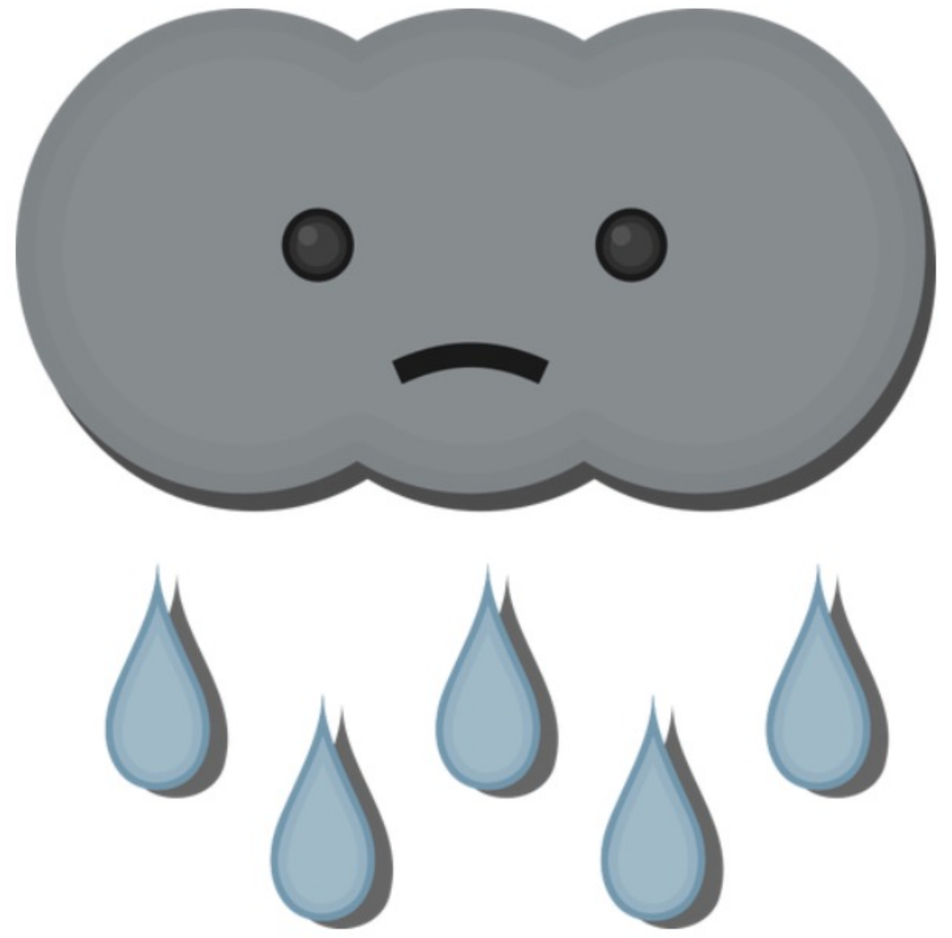
Larger TG and
No Storage/Perf Improvement

Larger TG and
Worse Storage/Perf

Conclusion

Implicit Memory Tagging: No-Overhead Memory Safety Using Alias-Free Tagged ECC

C/C++ on CPU and CUDA/OpenACC on GPU is memory unsafe.



We dove into memory tagging.

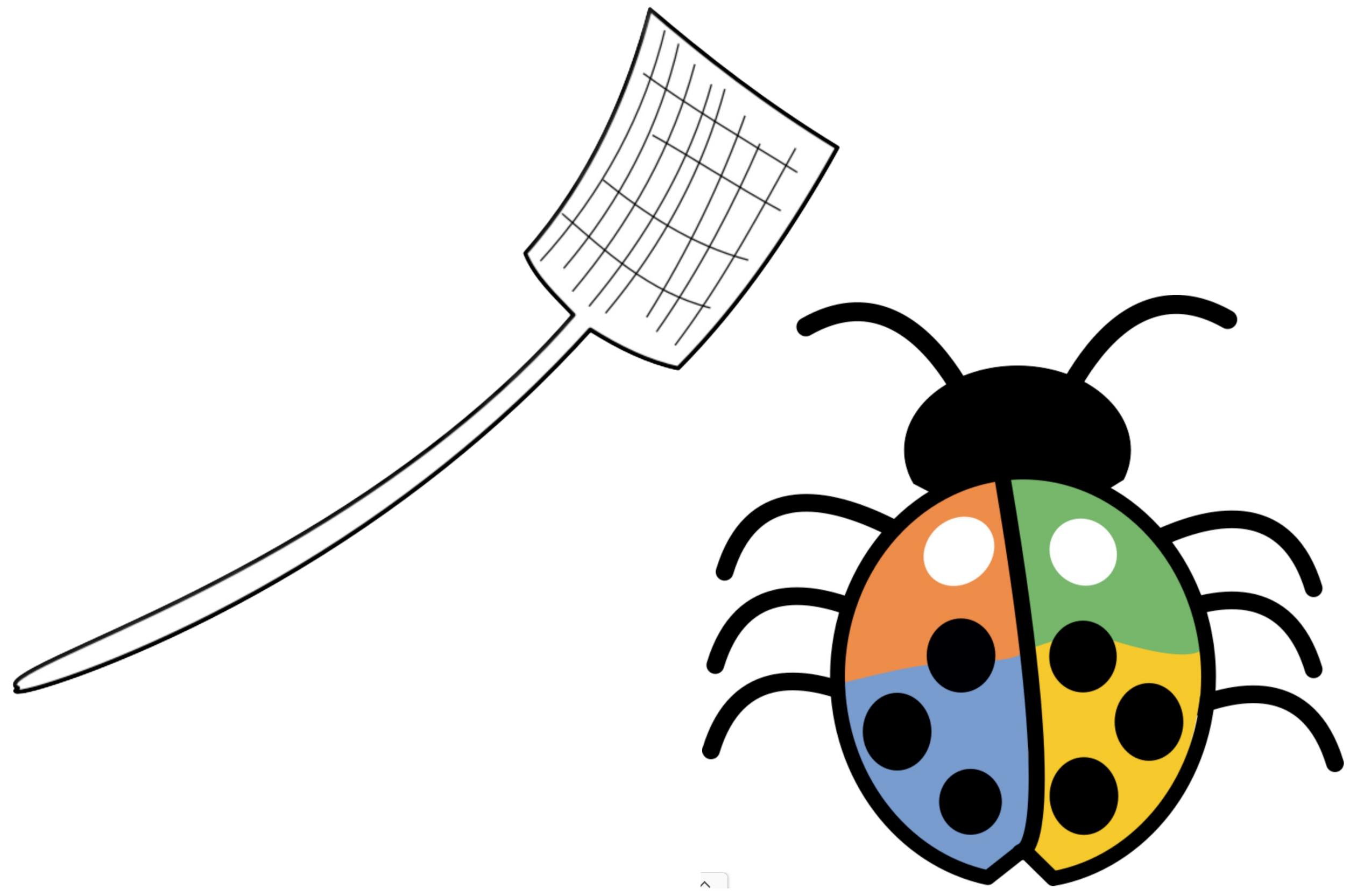
Popular! SPARC ADI (sideband tags) and ARM MTE (embedded tags)
Limited security OR high overheads (storage/performance/reliability)

Alias-Free Tagged ECC: a general mechanism to check tag equivalence in ECC.

Up to a 15 bit tag is possible, using all available upper pointer bits

Implicit Memory Tagging: Alias-Free Tagged ECC applied to memory safety.

Avoids downsides of prior memory tagging approaches.
0 Performance / 0 Resilience / 0 Storage Overheads, Superior Security



Backup

(Optional) Avoiding Mis-Attribution

Domain-specific sanity check. Provides 100% precise attribution.

In Hardware
In GPU Driver
Data Bits
Check Bits
Ptr. Tag
Decoder
Single-Bit Corrected Data
Address
Syndrome
Ptr. Tag
Tag Extraction
Ref. Tag
Ext. Tag
Ptr. Tag
==?
Tag/Data/Both?
Tag Values

$$\text{Diagnosis} = \begin{cases} \text{TMM}: & \text{Ref} \neq \text{Ptr} \quad \text{and} \quad \text{Ref} = \text{Ext} \\ \text{DUE}: & \text{Ref} = \text{Ptr} \quad \text{and} \quad \text{Ref} \neq \text{Ext} \\ \text{BOTH}: & \text{Ref} \neq \text{Ptr} \quad \text{and} \quad \text{Ref} \neq \text{Ext} \end{cases}$$

We maintain the assigned tags in the driver.

The address-to-tag mapping is only queried on a fatal IMT event!

Tag Carve-Out HW Implementation (e.g., ARM MTE)

Embedding tags into a dedicated carve-out…

VA-Tag| Data | Lock-Tag
Data / MTE Tag
Lock Tag
LDST
MIO
L1 CACHE
uTLB
L2 CACHE
DRAM
Memory Safety TagCheck
Lock Tag
HW Memory Safety MetaData Fetch
L2 miss
Lock Tag Carve-Out

Tags are cached densely-packed in LLC…
But sparse workloads will incur ~2X traffic to memory

ECC Stealing HW Implementation (e.g., SPARC ADI)

"Steal" ECC bits at every level of the memory hierarchy…

SM
Registers
ECC
L1/SharedMem
ECC
SM
Registers
ECC
L1/SharedMem
ECC
High-Speed Crossbar
L2 Slice
ECC
DRAM
ECC
L2 Slice
ECC
DRAM
ECC
L2 Slice
ECC
DRAM
ECC
SM
Registers
L1/SharedMem
SM
Registers
L1/SharedMem
ECC
Lock Tag
High-Speed Crossbar
L2 Slice
DRAM
L2 Slice
DRAM
L2 Slice
DRAM

Also, widen all address busses to the full 64b width to carry both the { Key Tag, Pointer Address }

Further Slowdown Analysis

Slowdown is high for 1) bandwidth-bound, 2) high read bloat programs ☹

Slowdown or Read Bloat (%)
70%
60%
50%
40%
30%
20%
10%
0%
-10%
Outlier Bloat (114%)
All Workloads, Sorted by % Slowdown
1024
896
768
640
512
384
256
128
0
DRAM Bandwidth (MB/s)
% Slowdown
% Read Bloat
DRAM B/W

A. Low DRAM Bandwidth
→ Low Slowdown

B. High DRAM Bandwidth
Low Read Bloat
→ Medium Slowdown

C. High DRAM Bandwidth
High Read Bloat
→ High Slowdown

Possible attacks

Denial of Service

E.g., intentionally crash a server program.

Information Leakage

E.g., use a buffer overread to read private data.

Data Corruption

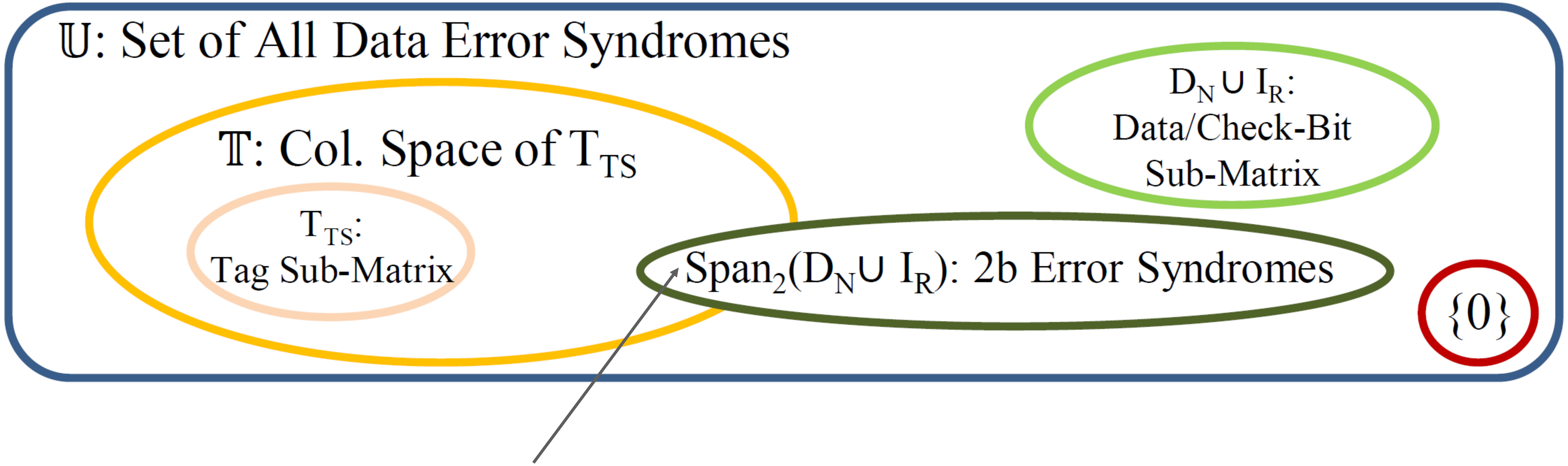
E.g., use a buffer overflow to overwrite critical data.

Arbitrary Code Execution

E.g., overwrite a function pointer or return address to hijack the control flow, OR overwrite existing functions with your own code!

Alias-Free Tagged ECC Visualization

A High-Level Set Intersection View



$\mathbb{U}$: Set of All Data Error Syndromes
$\mathbb{T}$: Col. Space of T_{TS}
T_{TS}:
Tag Sub-Matrix
$D_N \cup I_R$:
Data/Check-Bit
Sub-Matrix
$\text{Span}_2(D_N \cup I_R)$: 2b Error Syndromes
{0}

Some Risk of Mis-Attribution (DUE→TMM), at least in general case.

Not a big deal!

(More on this later…)

NVIDIA®