HashTag

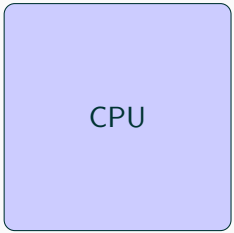
Hash-based Integrity Protection for Tagged Architectures

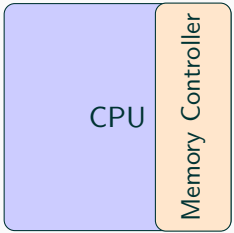
Lukas Lamster Martin Unterguggenberger David Schrammel Stefan Mangard

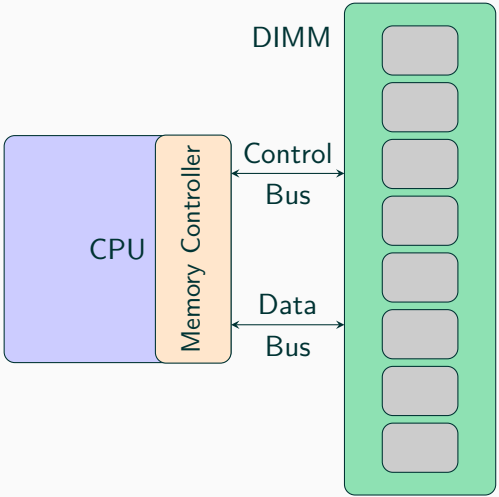
August 10, 2023

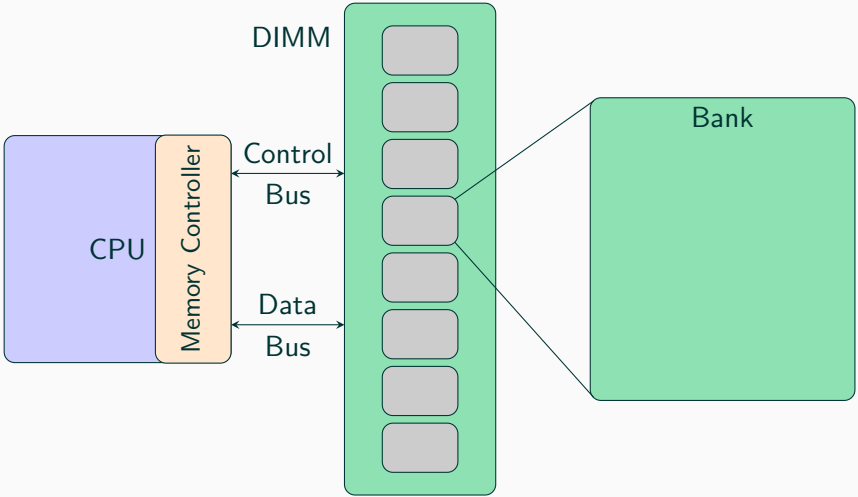
IAIK – Graz University of Technology

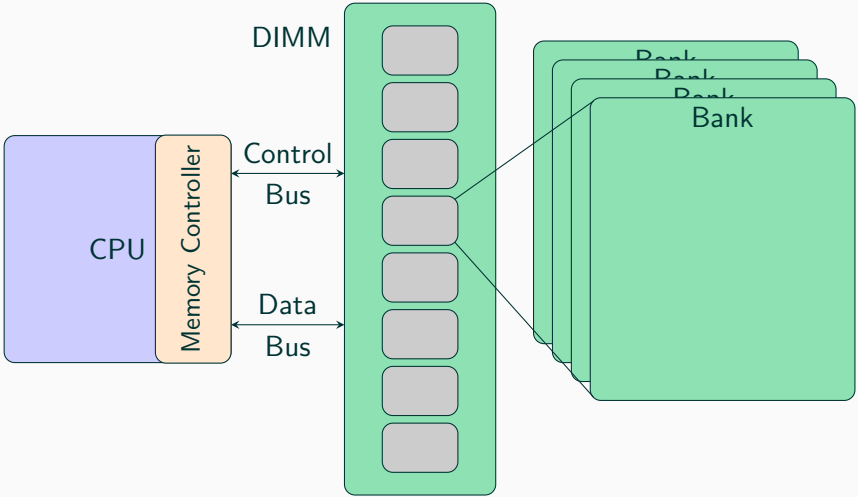
- Memory Tagging
  - Hardware-enforced security
  - Mitigate memory safety issues
  - Introduces performance and memory overhead
- DRAM Integrity Protection
  - Detect and correct errors in data
  - Low memory overhead
  - Widely used in server systems
- **Our Contribution:**
  - We combine integrity protection and memory tagging
  - We perform a case study for existing memory tagging schemes
  - We reduce the performance overhead by an average factor of 20

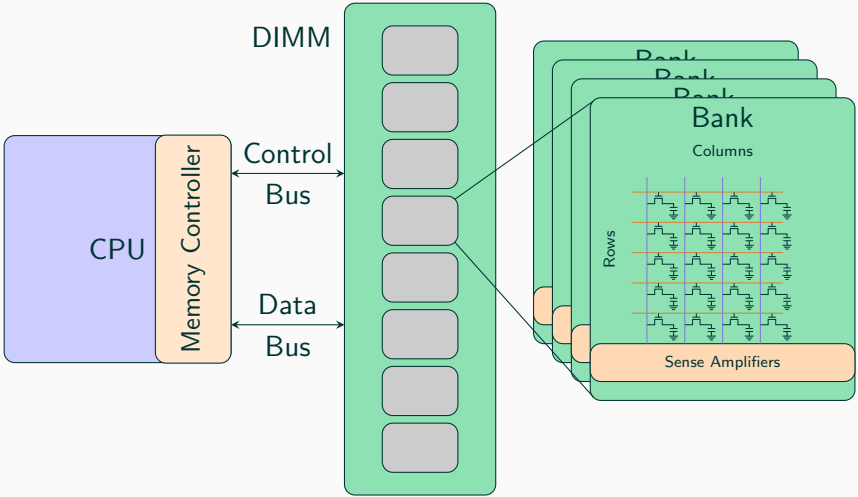

CPU

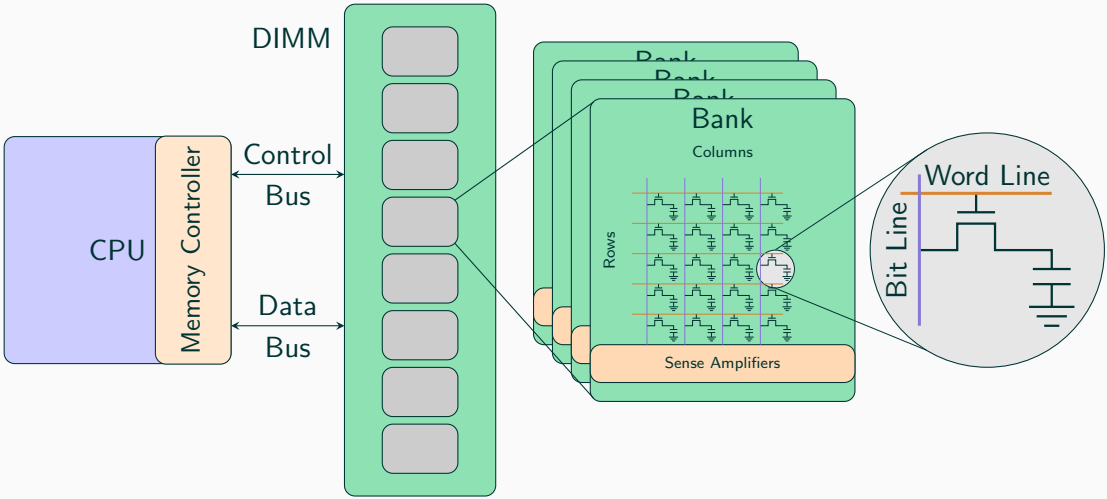

CPU
Memory Controller


DIMM
CPU
Memory Controller
Control
Bus
Data
Bus


DIMM
Bank
CPU
Memory Controller
Control
Bus
Data
Bus


DIMM
CPU
Memory Controller
Control
Bus
Data
Bus
Bank


DIMM
CPU
Memory Controller
Control
Bus
Data
Bus
Bank
Columns
Rows
Sense Amplifiers

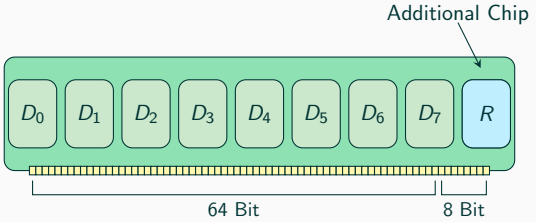

DIMM
CPU
Memory Controller
Control
Bus
Data
Bus
Bank
Columns
Rows
Sense Amplifiers
Word Line
Bit Line

- Cells **leak charge**
- Leakage **not constant**
  Temperature, Radiation, ...
- Larger structures **influence multiple bits**
- **Common** vs. **uncommon** faults

Failure Mode	Bits Affected	Description
F1	1	Single faulty bit
F2	8	Single stuck pin
F3S	up to 56	Multiple stuck pins in a single chip
F3M	up to 56	Multiple stuck pins in multiple chips
F4	up to 64	Broken chip (all pins stuck)
F5S	up to 57	F3S + transient fault
F5M	up to 57	F3M + transient fault

Note: We only consider **naturally occuring** faults

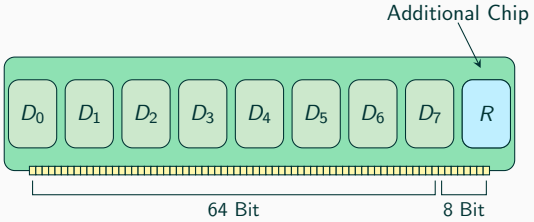
- Use **E**rror **C**orrecting **C**odes (ECC)
- Add **redundancy** in additional chip
- Store **linear checksum** on write
- Verify on load
- Bus width **increases**



$$R = f(D_0, D_1, \cdots, D_7)$$

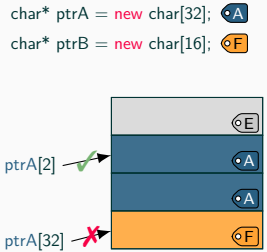
Error Correction

- Use **E**rror **C**orrecting **C**odes (ECC)
- Add **redundancy** in additional chip
- Store **linear checksum** on write
- Verify on load
- Bus width **increases**
- Limited error detection
  - Bounded by hamming distance
- Potential miscorrection
  - In case of large errors

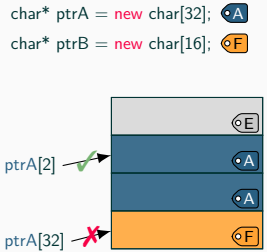


$$R = f(D_0, D_1, \cdots, D_7)$$

- Store **metadata** for each allocation
- Check metadata on access
- Enforce **tagging policies**
- Provide **memory safety**
- Implement **domain isolation**

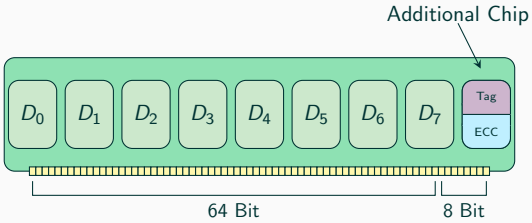


- Store **metadata** for each allocation
- Check metadata on access
- Enforce **tagging policies**
- Provide **memory safety**
- Implement **domain isolation**
- Additional storage overhead
- Increased memory pressure



Can we combine tagging and integrity protection?

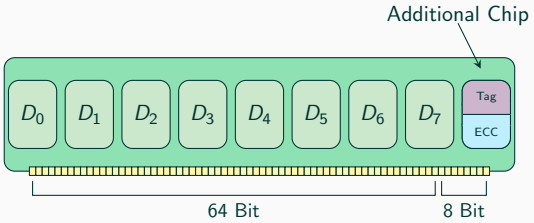
? "Steal" bits from linear code?



Combining Tagging and Integrity Protection

Can we combine tagging and integrity protection?

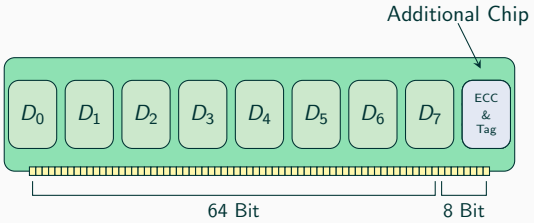
- ? "Steal" bits from linear code?
- 👎 Weakens error detection
- 👎 Weakens error correction



Combining Tagging and Integrity Protection

Can we combine tagging and integrity protection?

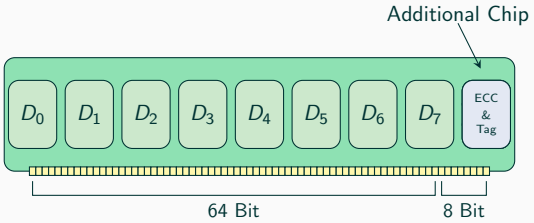
- ? "Steal" bits from linear code?
- 👎 Weakens error detection
- 👎 Weakens error correction
- ? Implicitly encode tags?



Combining Tagging and Integrity Protection

Can we combine tagging and integrity protection?

- ? "Steal" bits from linear code?
- 👎 Weakens error detection
- 👎 Weakens error correction
- ? Implicitly encode tags?
- 👎 Cannot read tags, aliasing possible
- 👎 Tag size is limited

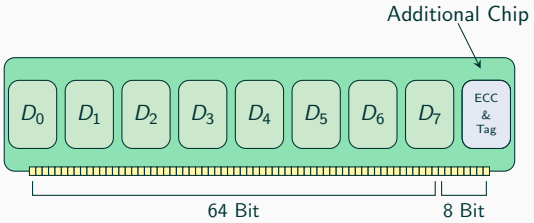


Combining Tagging and Integrity Protection

Can we combine tagging and integrity protection?

- ? "Steal" bits from linear code?
- 👎 Weakens error detection
- 👎 Weakens error correction
- ? Implicitly encode tags?
- 👎 Cannot read tags, aliasing possible
- 👎 Tag size is limited

Takeaways:



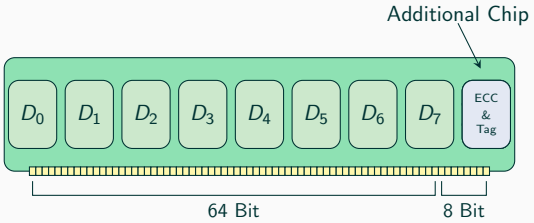
Combining Tagging and Integrity Protection

Can we combine tagging and integrity protection?

- ? "Steal" bits from linear code?
- 👎 Weakens error detection
- 👎 Weakens error correction
- ? Implicitly encode tags?
- 👎 Cannot read tags, aliasing possible
- 👎 Tag size is limited

Takeaways:

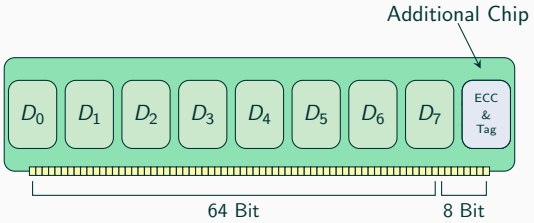
- 💡 **Explicitly** store tag



Combining Tagging and Integrity Protection

Can we combine tagging and integrity protection?

- ? "Steal" bits from linear code?
- 👎 Weakens error detection
- 👎 Weakens error correction
- ? Implicitly encode tags?
- 👎 Cannot read tags, aliasing possible
- 👎 Tag size is limited

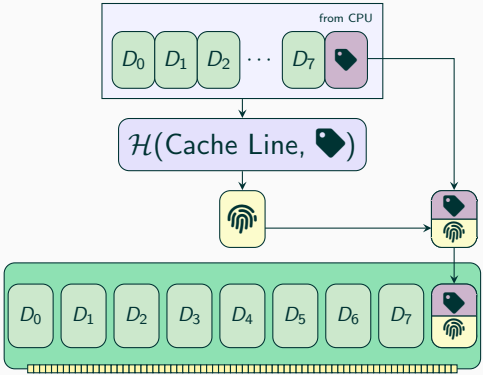


Takeaways:

- 💡 **Explicitly** store tag
- 💡 Use **non-linear** function

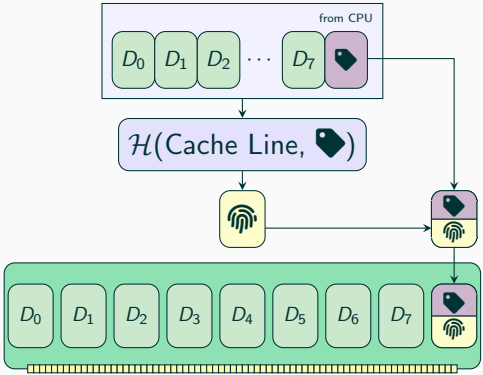
Idea:

- Replace linear checksum with **hash**
  Not necessarily a cryptographic hash
- Compute on **cache line granularity**
- **Truncate output** to accommodate tag



Idea:

- Replace linear checksum with **hash**
  Not necessarily a cryptographic hash
- Compute on **cache line granularity**
- **Truncate output** to accommodate tag
- Tags are readable

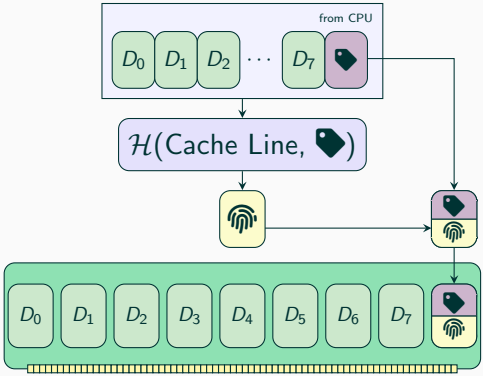


Idea:

- Replace linear checksum with **hash**
  Not necessarily a cryptographic hash
- Compute on **cache line granularity**
- **Truncate output** to accommodate tag

Tags are readable

Errors are detectable



We want to **detect** and **correct** errors
What properties does $\mathcal{H}$ need to have?

We want to **detect** and **correct** errors

What properties does $\mathcal{H}$ need to have?

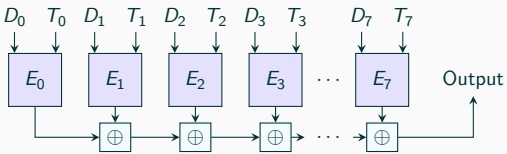
- $P(\mathcal{H}(D) = \mathcal{H}(D'))$ should be low

We want to **detect** and **correct** errors
What properties does $\mathcal{H}$ need to have?

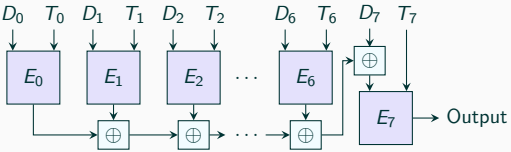
- $P(\mathcal{H}(D) = \mathcal{H}(D'))$ should be low
- $\mathcal{H}$ should be quick to compute

We want to **detect** and **correct** errors
What properties does $\mathcal{H}$ need to have?

- $P(\mathcal{H}(D) = \mathcal{H}(D'))$ should be low
- $\mathcal{H}$ should be quick to compute



Not cryptographically secure!

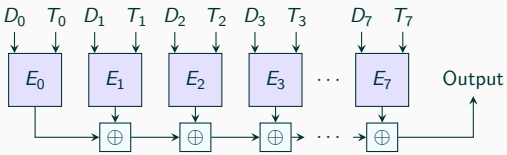


Slightly slower

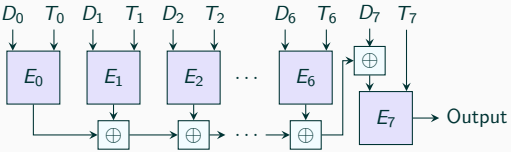
We want to **detect** and **correct** errors

What properties does $\mathcal{H}$ need to have?

- $P(\mathcal{H}(D) = \mathcal{H}(D'))$ should be low
- $\mathcal{H}$ should be quick to compute



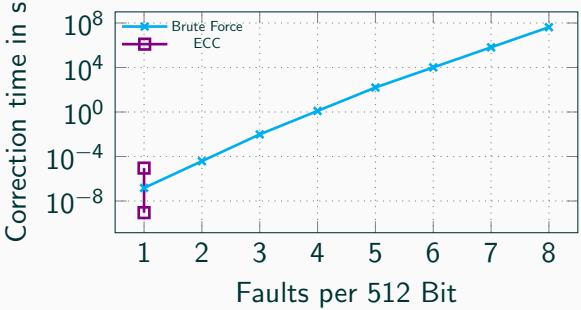
Not cryptographically secure!



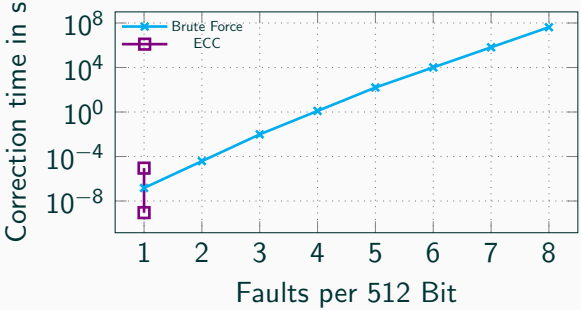
Slightly slower

Note: Any function with strong diffusion is suitable if no cryptographic security is required

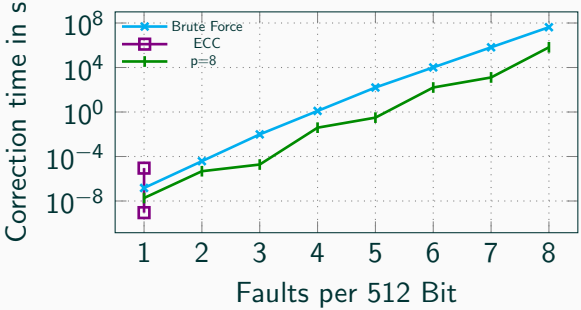
- Iterate over possible errors e
- Compute $\mathcal{H}(D' + e)$
- Matching hash $\rightarrow$ error corrected
- Strong complexity growth



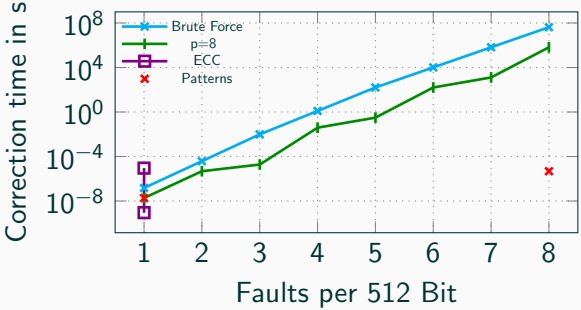
- Iterate over possible errors e
- Compute $\mathcal{H}(D' + e)$
- Matching hash $\rightarrow$ error corrected
- Strong complexity growth
- Decrease complexity?



- Iterate over possible errors e
- Compute $\mathcal{H}(D' + e)$
- Matching hash $\rightarrow$ error corrected
- Strong complexity growth
- Decrease complexity?
  - Add **parity bits**



- Iterate over possible errors e
- Compute $\mathcal{H}(D' + e)$
- Matching hash $\rightarrow$ error corrected
- Strong complexity growth
- Decrease complexity?
  - Add **parity bits**
  - Consider **error patterns**



Case Study: Tagged Memory Architectures

Architecture	Tag Size	Granularity	Bit Distribution			Faulty Bits	Correctable Failure Modes							
			Parity	Hash	Tag	in Hash (d)	F1	F2	F3S(8,4)	F3S(4,4)	F3M	F5S(4,4)	F5S(8,_)	F5M
DIFT [6], HDFI [5] Shakti-t [4], M-Machine [3]	1 bit	8 bytes	8	+ 48	+ 8	4	✓	✓	✓	✓	3	✓	(8,3)	2
SPARC ADI [1]	4 bits	64 bytes	8	+ 52	+ 4	5	✓	✓	✓	✓	3	✓	(8,3)	3
CHERI 128 [9, 8]	1 bit	16 bytes	8	+ 52	+ 4	5	✓	✓	✓	✓	3	✓	(8,3)	3
CHERI 256 [9, 8]	1 bit	32 bytes	8	+ 54	+ 2	5	✓	✓	✓	✓	3	✓	(8,4)	3
ARM MTE [2]	4 bits	16 bytes	8	+ 40	+ 16	4	✓	✓	(8,3)	✓	2	(4,2)	(8,2)	2
lowRISC [7]	4 bits	8 bytes	8	+ 24	+ 32	2	✓	✓	✗	✗	✗	✗	✗	✗
Model A	32 bits	64 bytes	1	+31	+32	3	✓	✓	(8,2)	(4,2)	2	✗	✗	✗
Model B	46 bits	64 bytes	1	+17	+46	1	✓	✓	✗	✗	✗	✗	✗	✗
Model C	51 bits	64 bytes	1	+12	+51	1	✓	✗	✗	✗	✗	✗	✗	✗
No Tagging, SEC-DED	-	-	64	-	-	-	✓	✓	✗	✗	✗	✗	✗	✗
No Tagging, Chipkill	-	-	64	-	-	-	✓	✓	(✓)	✓	✗	✗	✗	✗

Case Study: Tagged Memory Architectures

Architecture	Tag Size	Granularity	Bit Distribution			Faulty Bits	Correctable Failure Modes							
			Parity	Hash	Tag	in Hash (d)	F1	F2	F3S(8,4)	F3S(4,4)	F3M	F5S(4,4)	F5S(8,_)	F5M
DIFT [6], HDFI [5] Shakti-t [4], M-Machine [3]	1 bit	8 bytes	8	+ 48	+ 8	4	✓	✓	✓	✓	3	✓	(8,3)	2
SPARC ADI [1]	4 bits	64 bytes	8	+ 52	+ 4	5	✓	✓	✓	✓	3	✓	(8,3)	3
CHERI 128 [9, 8]	1 bit	16 bytes	8	+ 52	+ 4	5	✓	✓	✓	✓	3	✓	(8,3)	3
CHERI 256 [9, 8]	1 bit	32 bytes	8	+ 54	+ 2	5	✓	✓	✓	✓	3	✓	(8,4)	3
ARM MTE [2]	4 bits	16 bytes	8	+ 40	+ 16	4	✓	✓	(8,3)	✓	2	(4,2)	(8,2)	2
lowRISC [7]	4 bits	8 bytes	8	+ 24	+ 32	2	✓	✓	✗	✗	✗	✗	✗	✗
Model A	32 bits	64 bytes	1	+31	+32	3	✓	✓	(8,2)	(4,2)	2	✗	✗	✗
Model B	46 bits	64 bytes	1	+17	+46	1	✓	✓	✗	✗	✗	✗	✗	✗
Model C	51 bits	64 bytes	1	+12	+51	1	✓	✗	✗	✗	✗	✗	✗	✗
No Tagging, SEC-DED	-	-	64	-	-	-	✓	✓	✗	✗	✗	✗	✗	✗
No Tagging, Chipkill	-	-	64	-	-	-	✓	✓	(✓)	✓	✗	✗	✗	✗

Case Study: Tagged Memory Architectures

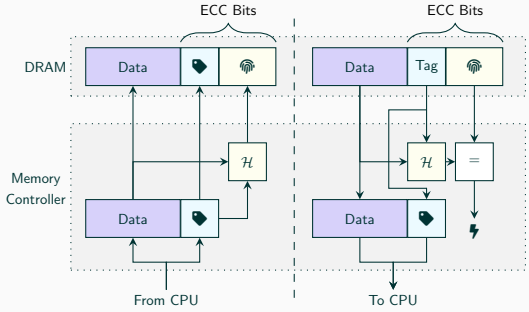
Architecture	Tag Size	Granularity	Bit Distribution			Faulty Bits	Correctable Failure Modes							
			Parity	Hash	Tag	in Hash (d)	F1	F2	F3S(8,4)	F3S(4,4)	F3M	F5S(4,4)	F5S(8,_)	F5M
DIFT [6], HDFI [5] Shakti-t [4], M-Machine [3]	1 bit	8 bytes	8	+ 48	+ 8	4	✓	✓	✓	✓	3	✓	(8,3)	2
SPARC ADI [1]	4 bits	64 bytes	8	+ 52	+ 4	5	✓	✓	✓	✓	3	✓	(8,3)	3
CHERI 128 [9, 8]	1 bit	16 bytes	8	+ 52	+ 4	5	✓	✓	✓	✓	3	✓	(8,3)	3
CHERI 256 [9, 8]	1 bit	32 bytes	8	+ 54	+ 2	5	✓	✓	✓	✓	3	✓	(8,4)	3
ARM MTE [2]	4 bits	16 bytes	8	+ 40	+ 16	4	✓	✓	(8,3)	✓	2	(4,2)	(8,2)	2
lowRISC [7]	4 bits	8 bytes	8	+ 24	+ 32	2	✓	✓	✗	✗	✗	✗	✗	✗
Model A	32 bits	64 bytes	1	+31	+32	3	✓	✓	(8,2)	(4,2)	2	✗	✗	✗
Model B	46 bits	64 bytes	1	+17	+46	1	✓	✓	✗	✗	✗	✗	✗	✗
Model C	51 bits	64 bytes	1	+12	+51	1	✓	✗	✗	✗	✗	✗	✗	✗
No Tagging, SEC-DED	-	-	64	-	-	-	✓	✓	✗	✗	✗	✗	✗	✗
No Tagging, Chipkill	-	-	64	-	-	-	✓	✓	(✓)	✓	✗	✗	✗	✗

A wide range of TMAs can be implemented

Many error patterns are correctable

Combining tagging with hash-based integrity protection is feasible

- Integrity is verified on each read
- Computing the hash takes time
- This impacts the system performance
- But how big is the impact?

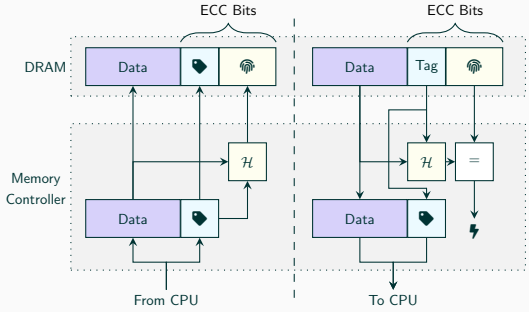


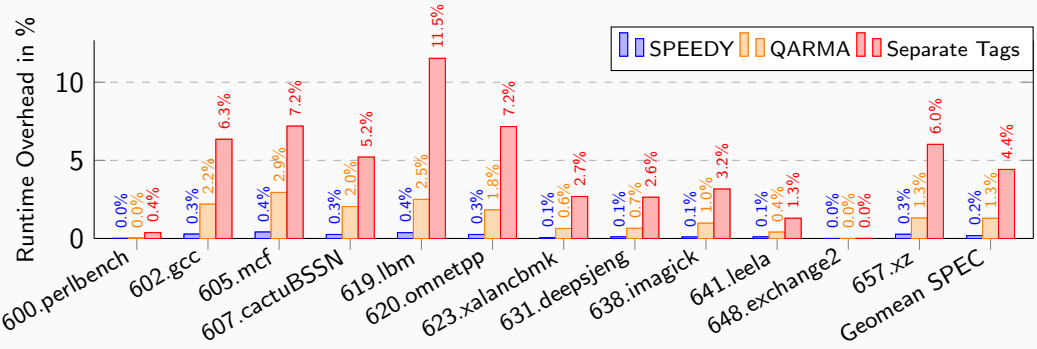
Performance Analysis

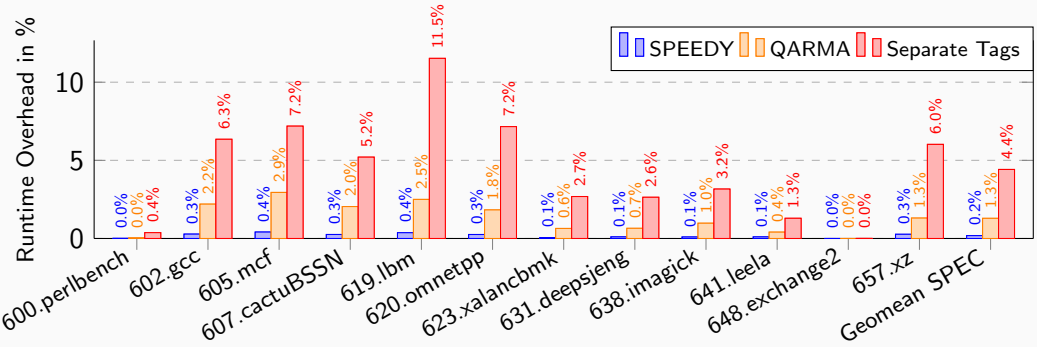
- Integrity is verified on each read
- Computing the hash takes time
- This impacts the system performance
- But how big is the impact?

Model system in gem5

Benchmark and measure overhead




Runtime Overhead in %
0
5
10
SPEEDY
QARMA
Separate Tags
0.0%
0.0%
0.4%
0.3%
2.2%
6.3%
0.4%
2.9%
7.2%
0.3%
2.0%
5.2%
0.4%
2.5%
11.5%
0.3%
1.8%
7.2%
0.1%
0.6%
2.7%
0.1%
0.7%
2.6%
0.1%
1.0%
3.2%
0.1%
0.4%
1.3%
0.0%
0.0%
0.0%
0.3%
1.3%
6.0%
0.2%
1.3%
4.4%
600.perlbench
602.gcc
605.mcf
607.cactuBSSN
619.lbm
620.omnetpp
623.xalancbmk
631.deepsjeng
638.imagick
641.leela
648.exchange2
657.xz
Geomean SPEC



Low overall overhead

Much faster than regular TMA

Hash-based Integrity Protection and Memory Tagging

- Replace ECC with truncated hash
- Co-locate tags in additional chip
- Eliminate storage and tag fetch overheads
- Still offer error detection and correction

» Future Work

- Consider DDR5 on-chip ECC
- Adapt to different granularities and burst sizes
- Consider alternative hash functions

HashTag

Hash-based Integrity Protection for Tagged Architectures

Lukas Lamster Martin Unterguggenberger David Schrammel Stefan Mangard

August 10, 2023

IAIK – Graz University of Technology

References

[1] Aingaran et al. M7: Oracle's Next-Generation Sparc Processor. In: IEEE Micro (2015).

[2] ARM Limited. Memory Tagging Extension: Enhancing memory safety through architecture. 2019. URL: https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/enhancing-memory-safety.

[3] Carter et al. Hardware Support for Fast Capability-based Addressing. In: ASPLOS'94. 1994.

[4] Menon et al. Shakti-T: A RISC-V Processor with Light Weight Security Extensions. In: HASP'17. 2017.

[5] Song et al. HDFI: Hardware-Assisted Data-Flow Isolation. In: S&P'16. 2016.

[6] Suh et al. Secure program execution via dynamic information flow tracking. In: ASPLOS'04. 2004.

[7] lowRISC Team. Tag support in the Rocket core. https://lowrisc.org/docs/minion-v0.4/tag_core/. Accessed: 2022-10-01. 2017. (Visited on 10/2022).

[8] Watson et al. CHERI: A Hybrid Capability-System Architecture for Scalable Software Compartmentalization. In: S&P'15. 2015.

[9] Woodruff et al. The CHERI capability model: Revisiting RISC in an age of risk. In: ISCA'14. 2014.