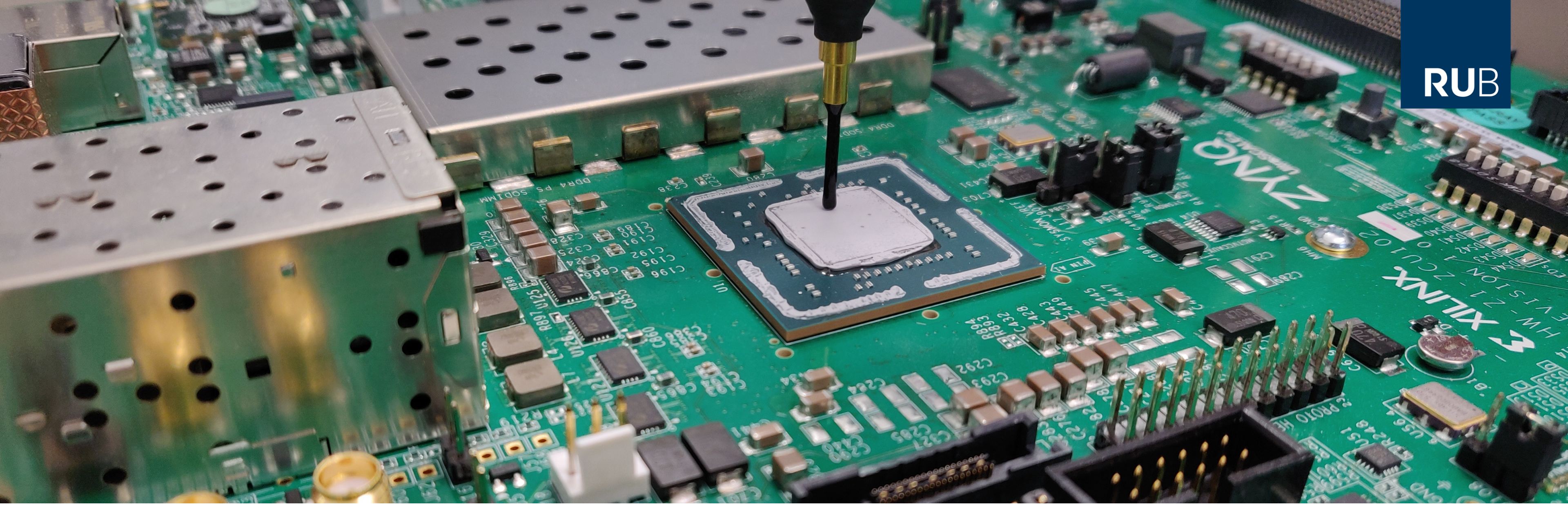


RUHR-UNIVERSITÄT BOCHUM

Risky Translations: Securing TLBs against Timing Side Channels

Florian Stolz, Jan Philipp Thoma, Pascal Sasdrich, Tim Güneysu

florian.stolz@rub.de

CHES 2023

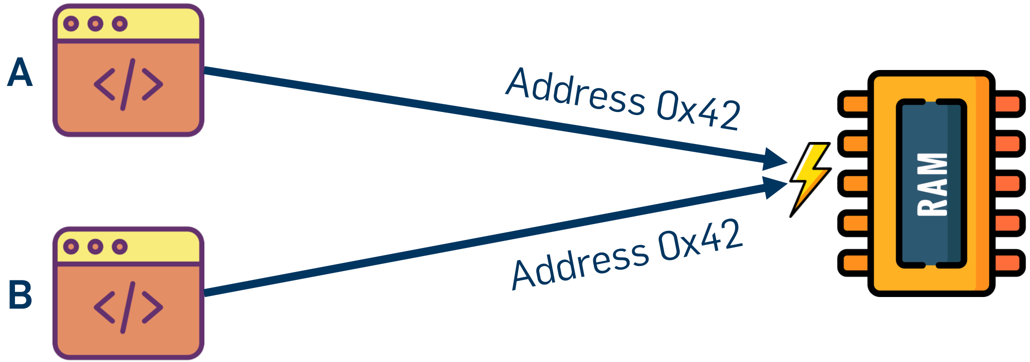
September 13, 2023

Chair for Security Engineering
Faculty of Computer Science
Ruhr University Bochum

Motivation

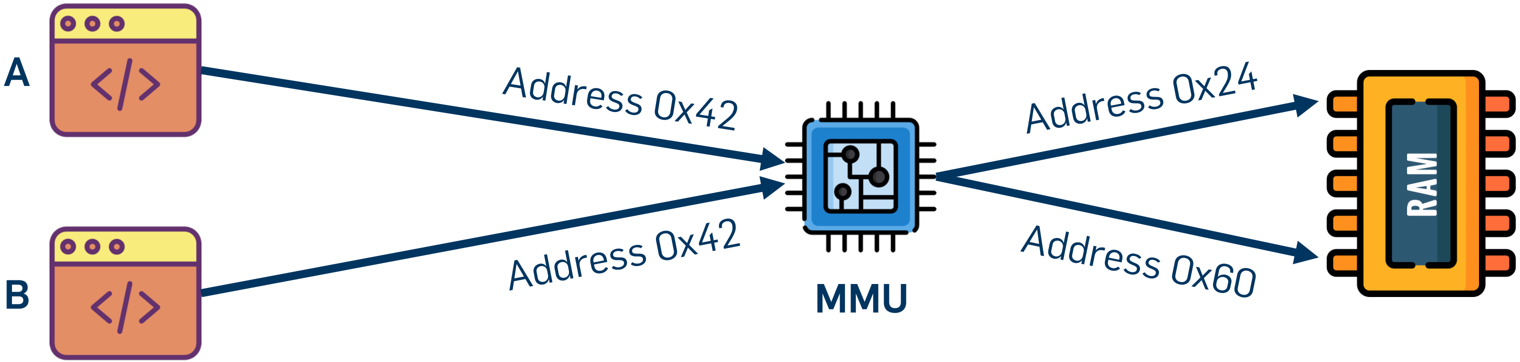
Virtual Memory

- Modern CPUs run multiple applications at once
  - All programs have different RAM requirements
  - All programs should run in isolation



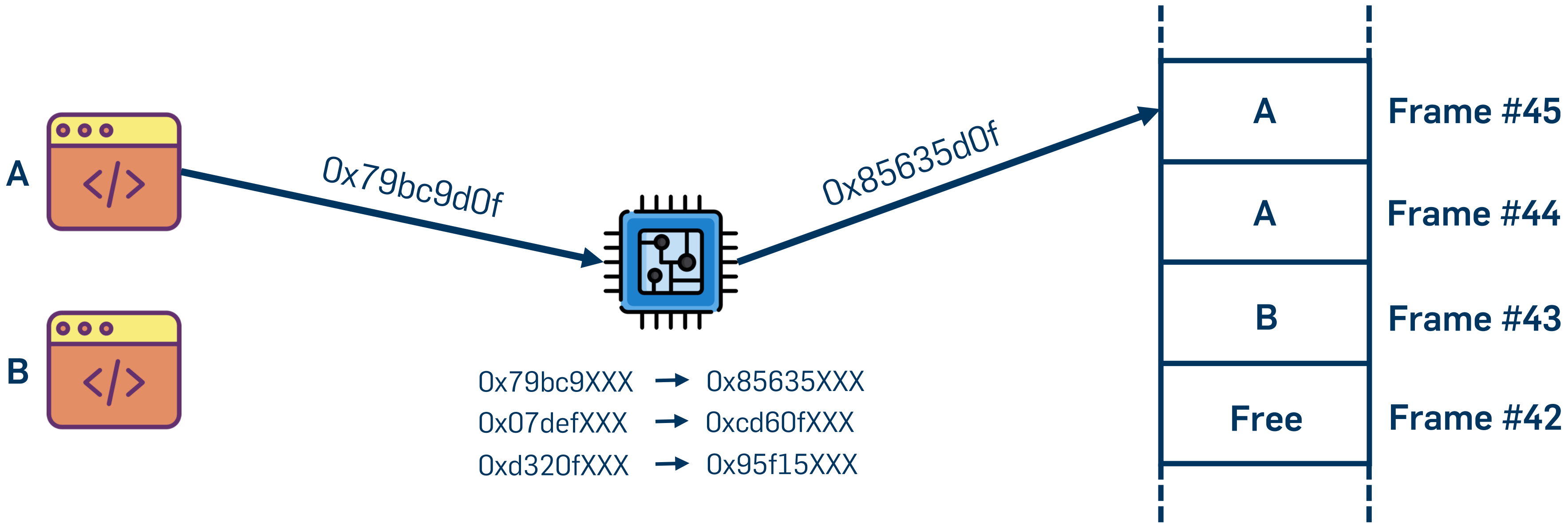
Virtual Memory

- Memory Management Unit abstracts memory
  - Every program gets its own virtual address space
  - Virtual address are mapped arbitrarly to physical addresses



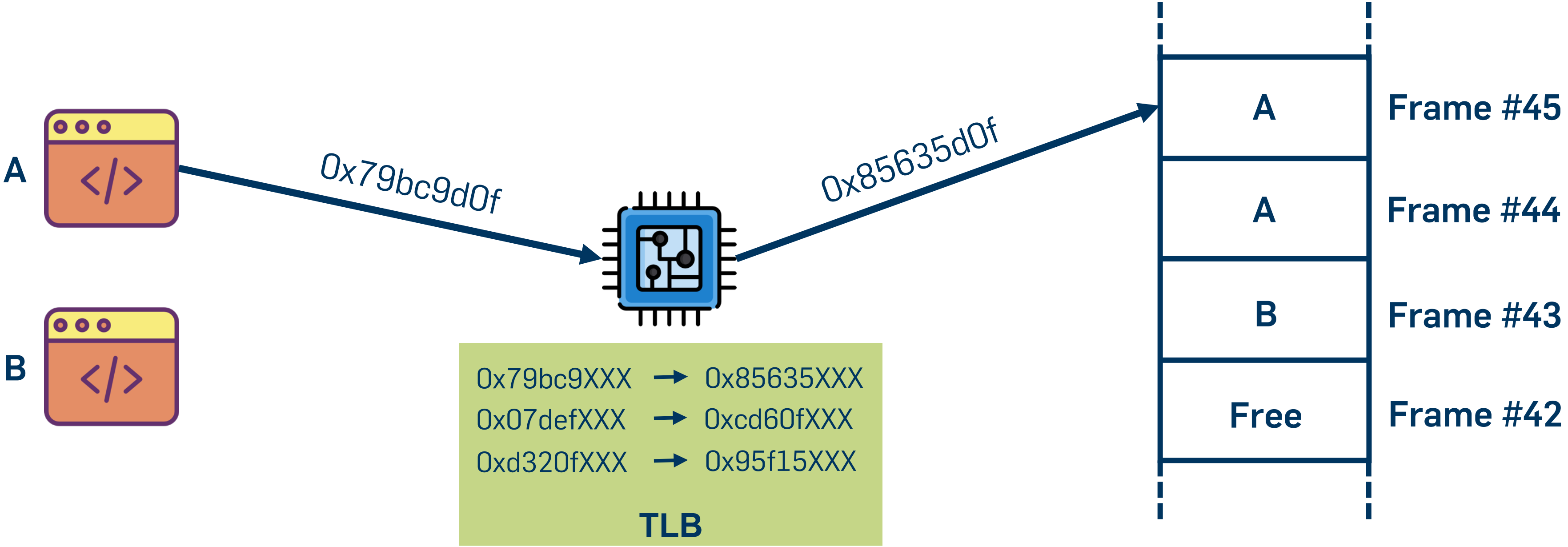
Virtual Memory

- Paging is the defacto memory management standard



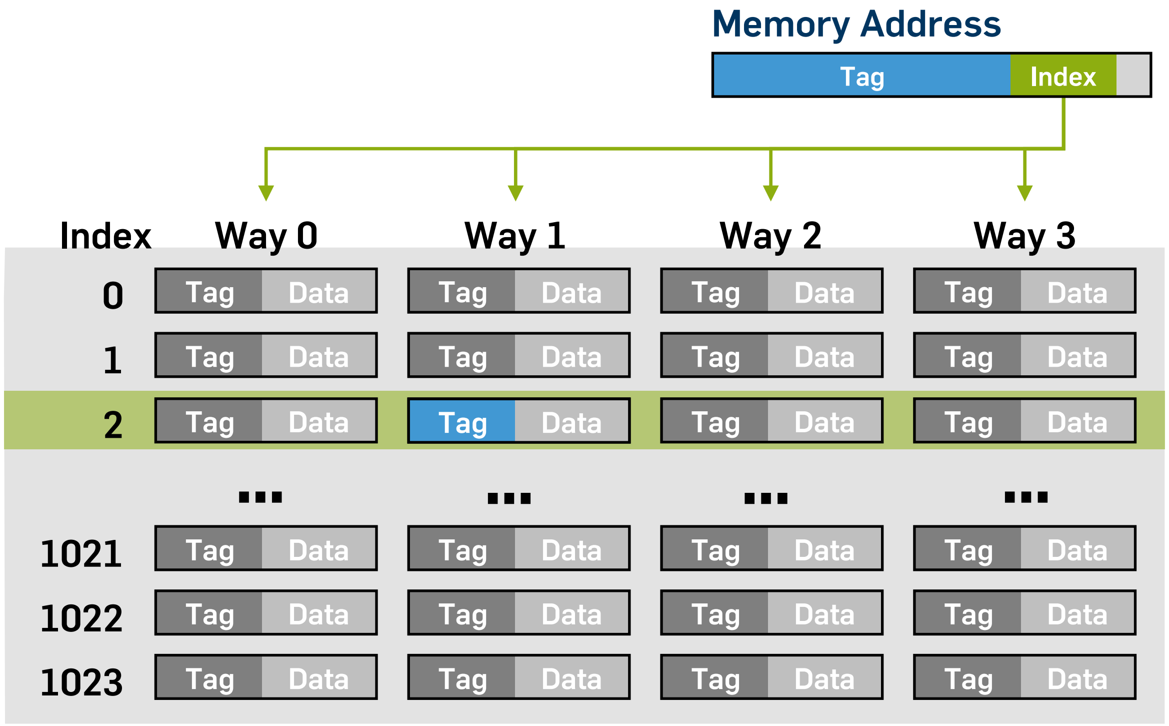
Virtual Memory

- Paging is the defacto memory management standard
  - A *Translation Lookaside Buffer* speeds up memory translations



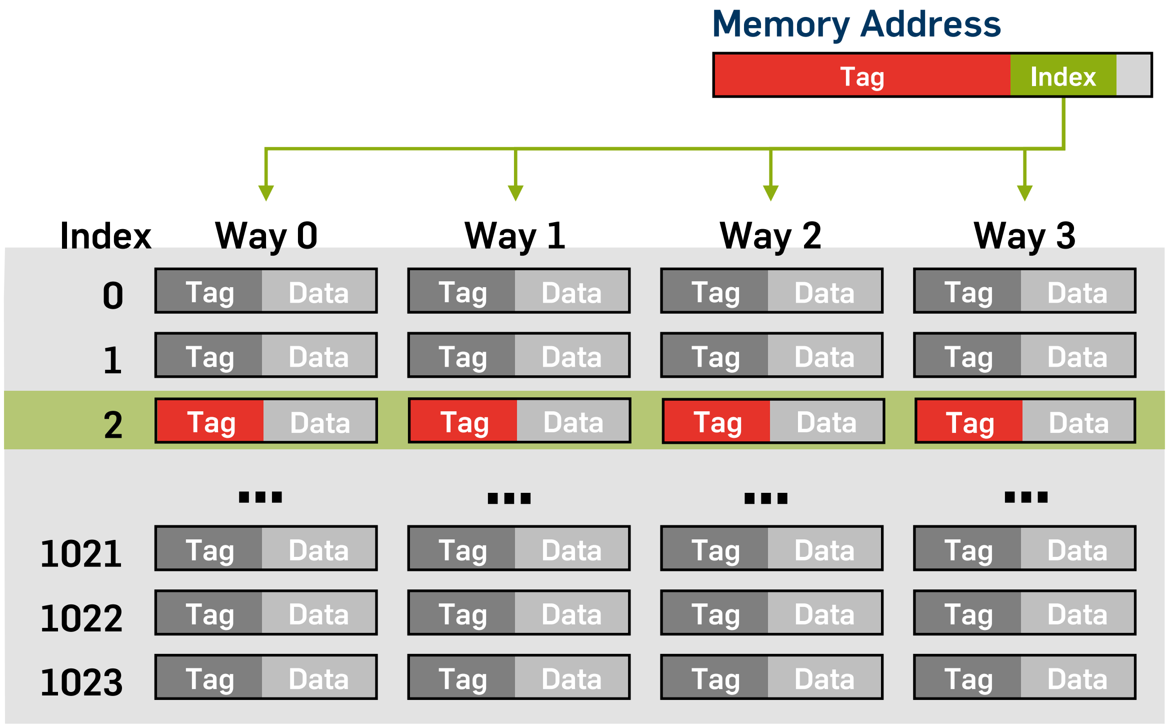
Caches

- Caches are highly efficient storages
  - ➢ Special organization for fast lookups (set-associativity)



Prime + Probe Attack

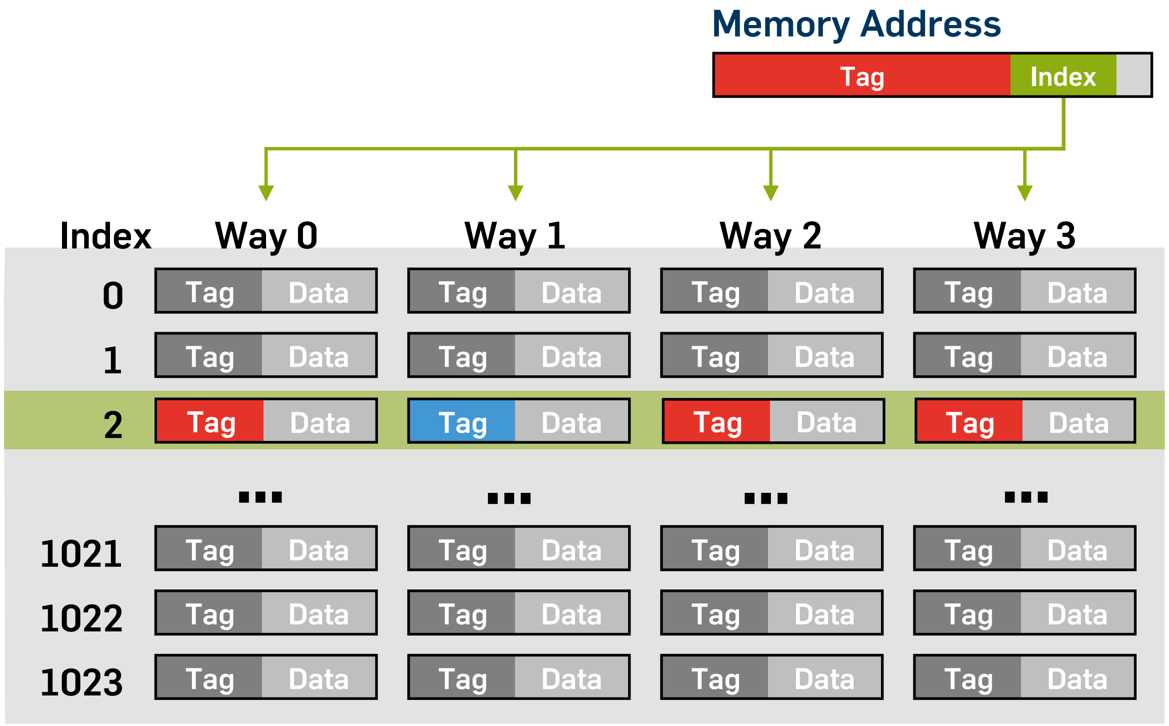
- Some caches are shared and can thus lead to information leakage
  1. Fill a cache set with controlled addresses



Prime + Probe Attack

- Some caches are shared and can thus lead to information leakage

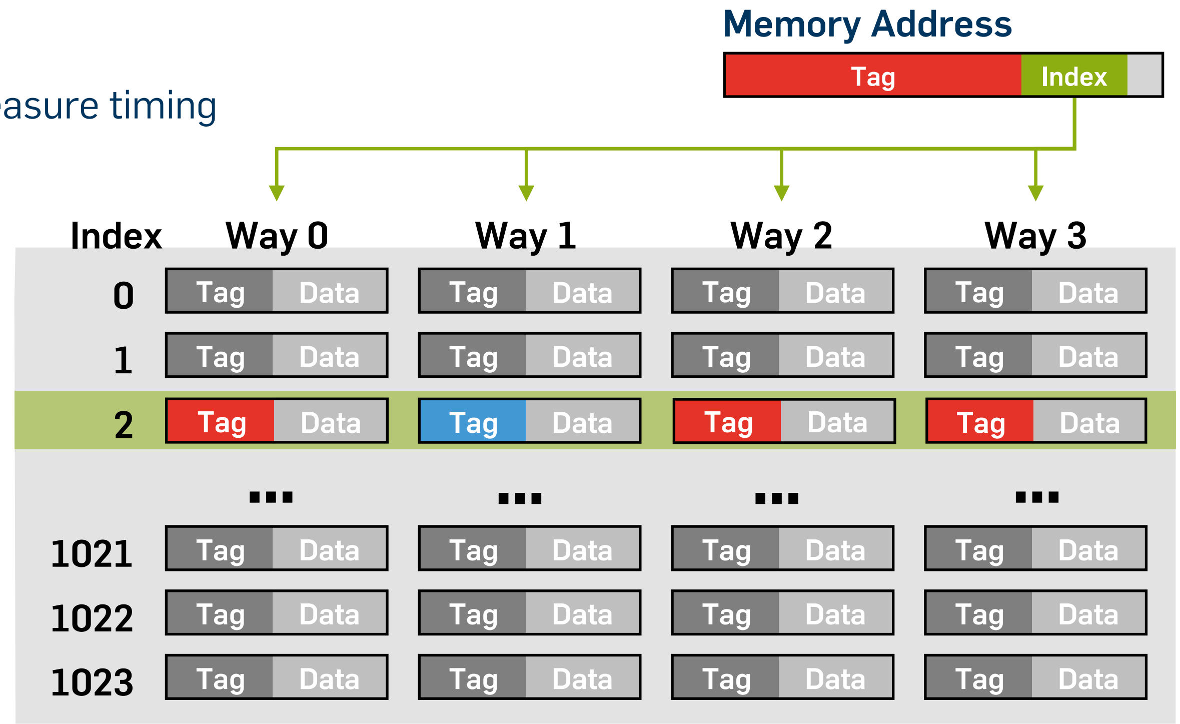
1. Fill a cache set with controlled addresses
2. Victim performs access



Prime + Probe Attack

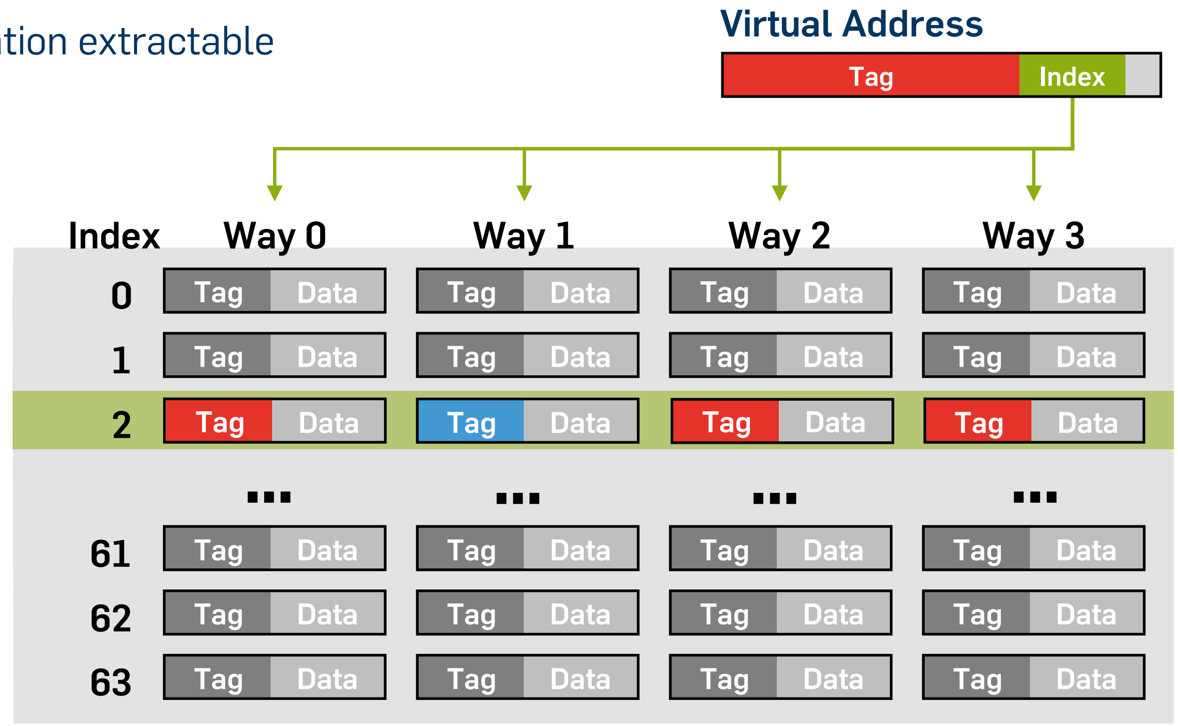
- Some caches are shared and can thus lead to information leakage

1. Fill a cache set with controlled addresses
2. Victim performs access
3. Reaccess addresses and measure timing



Prime + Probe Attack

- TLBs are implemented as set-associative caches
  - Gras *et al.* demonstrated Prime + Probe on TLBs
  - Only coarse-grained information extractable



How to protect TLBs?

Reusing cache attack countermeasures

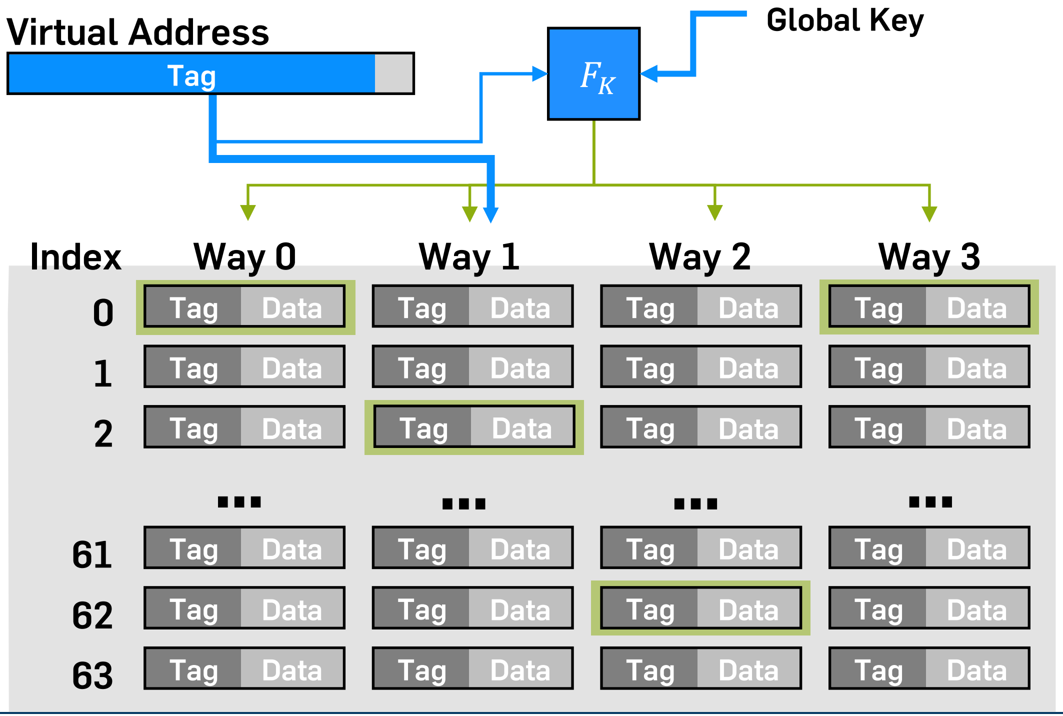
- Deng et al. investigated two countermeasures for TLBs:
  - Static Partitioning
  - Random Fill
- May not always provide best performance/security trade-off

TLBCoat Overview

- TLBCoat employs index randomization with TLB tailored adapations
- Combines strengths of previous approaches:
  - Separation of processes
  - Randomness
- Irons out weaknesses:
  - Allow mutually untrusted secure applications
  - Minimal OS and software modifications

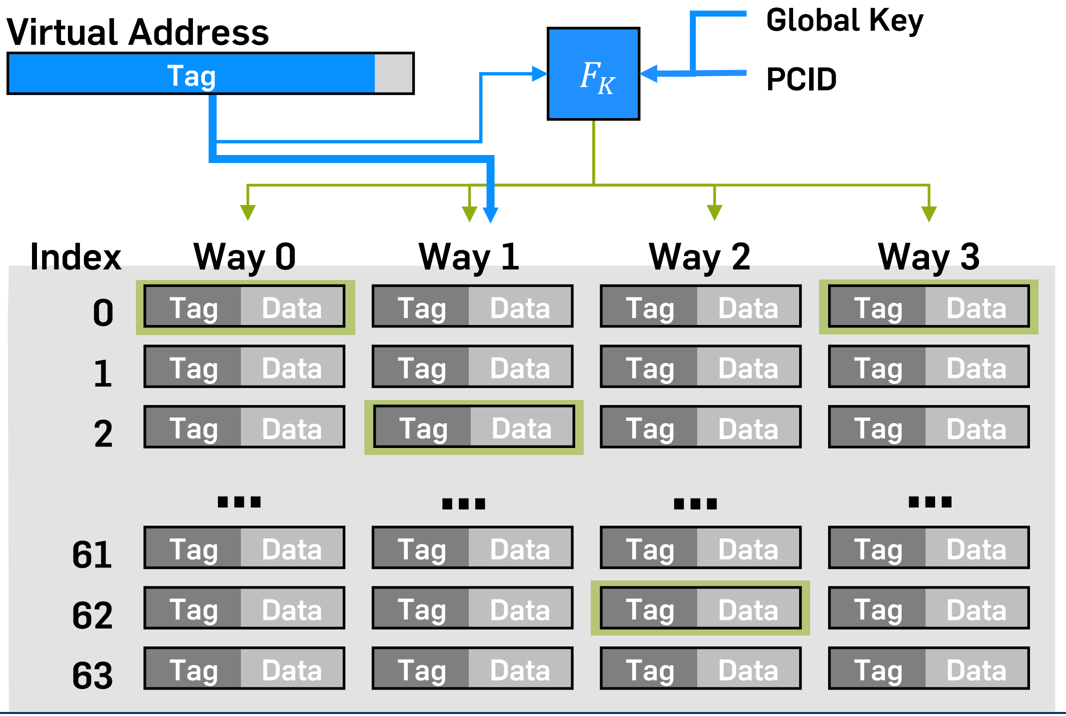
TLBCoat Overview

- Virtual Address is now processed by a randomization function
  - 3-Round PRINCE
- Prevents Prime + Probe
- But now vulnerable to Prime + Prune + Probe



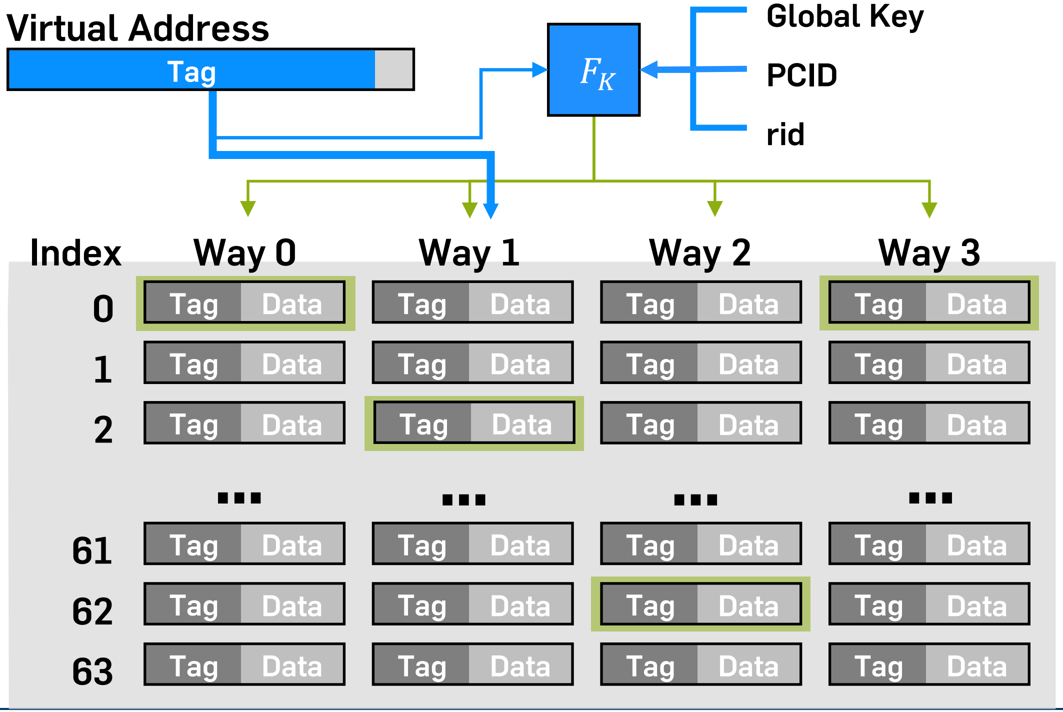
TLBCoat Overview

- Isolate Attacker from victim domain
- Still vulnerable to
  Prime + Prune + Probe



TLBCoat Overview

- Prevent profiling via rerandomization
- When should we rerandomize?

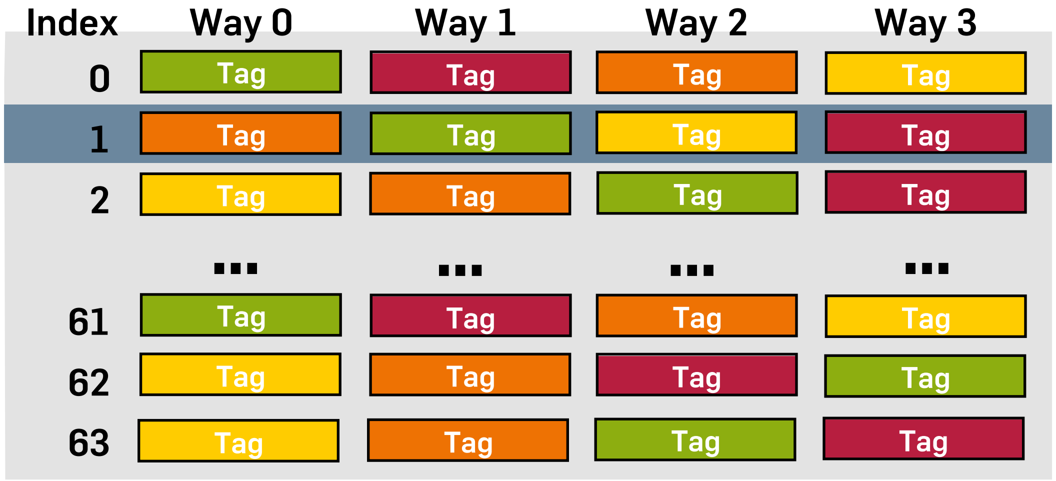


Rerandomization

- Pages represent a chunk of continuous data
  - Conclusion: Access to a page will likely lead to more accesses to the *same* page
- Hypothesis confirmed by running PARSEC benchmarks:
  - Median Miss-to-Hit Ratio: 0.22%
- We add a Miss counter to each process → once it reaches 0 we change the rid
  - Miss counter initialization values depends on the replacement policy and TLB size
  - A miss counter equal to the TLB size has the best performance/security trade-off

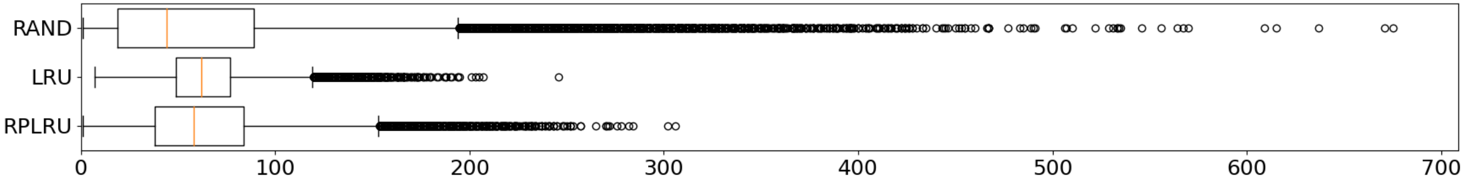
Replacement Policy

- Many works propose index randomization and the usage of *Least Recently Used* at the same time
  - ➢ LRU is not easily implementable in randomized caches
  - ➢ So what to choose instead?



Replacement Policy

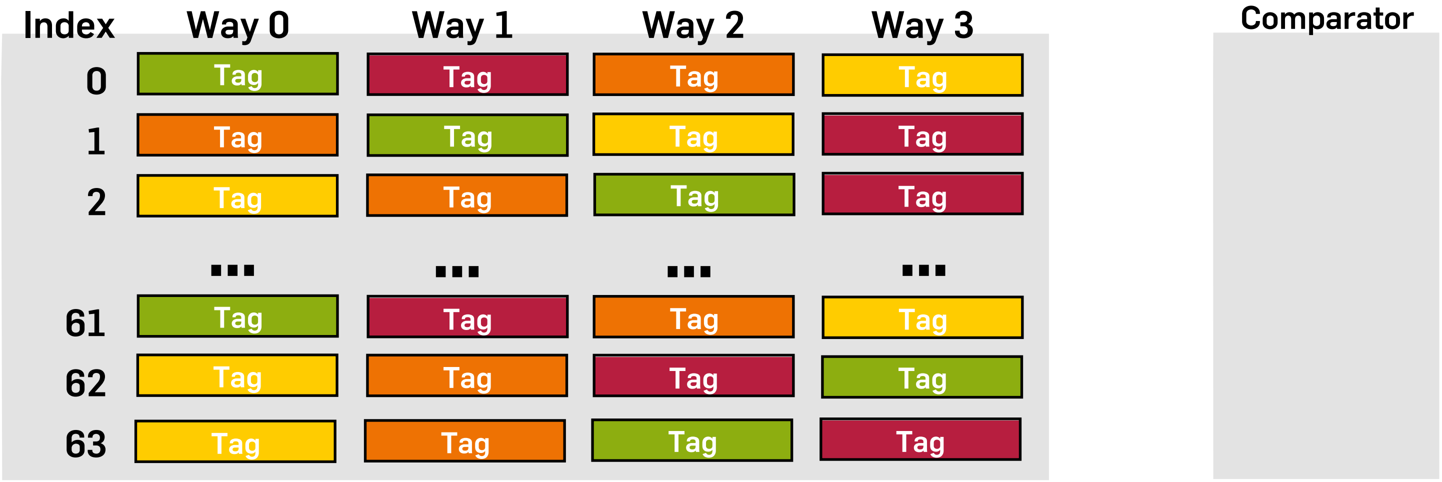
- Simulation: How many tries does attacker require to cause victim eviction?



- Random replacement actually makes it easier to remove addresses
- Consequences:
  - Miss counter must have a low value → more rerandomizations → performance hit

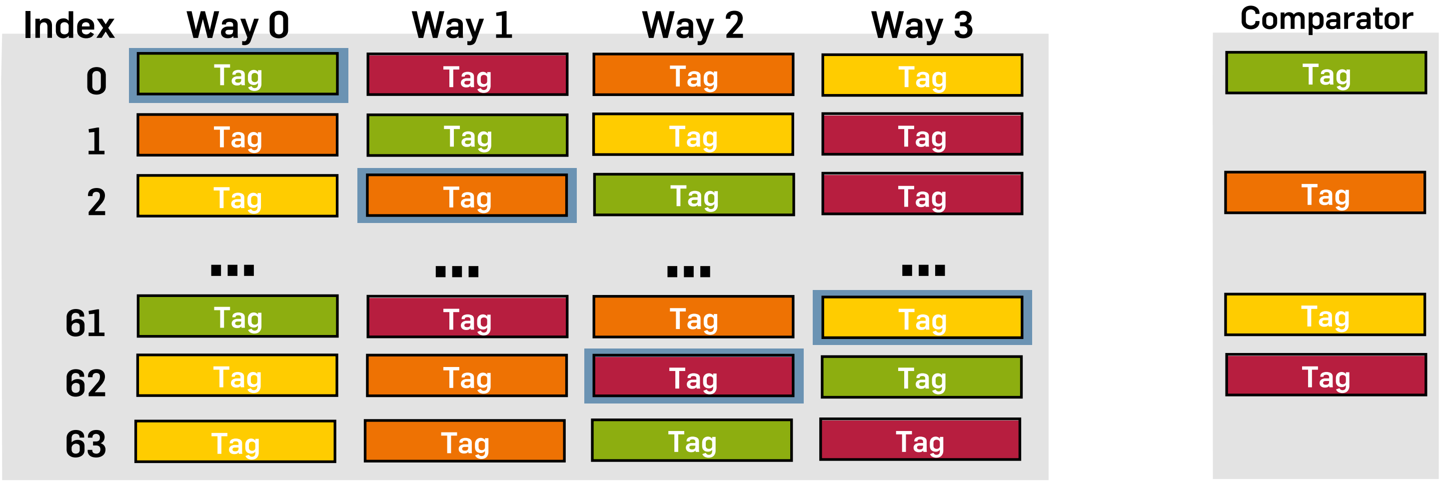
Replacement Policy

- We designed *Random Pseudo Least Recently Used* for TLBCoat



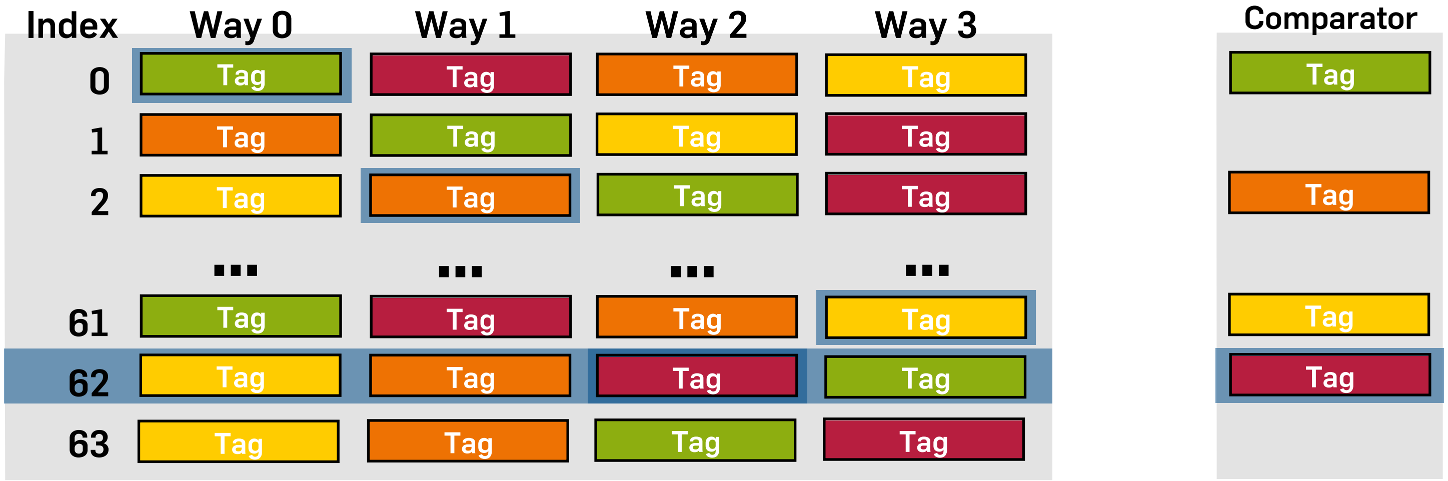
Replacement Policy

- We designed *Random Pseudo Least Recently Used* for TLBCoat



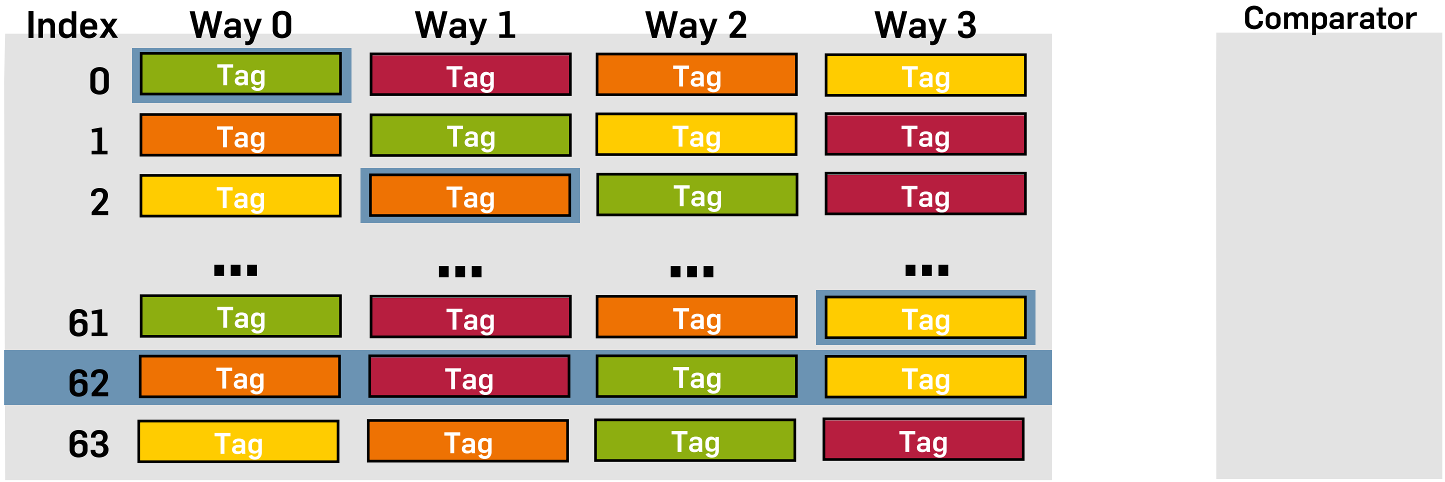
Replacement Policy

- We designed *Random Pseudo Least Recently Used* for TLBCoat



Replacement Policy

- We designed *Random Pseudo Least Recently Used* for TLBCoat



Evaluation

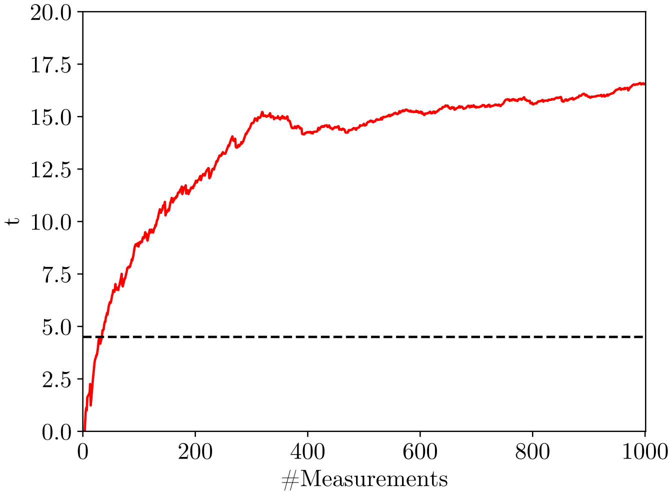
Evaluation Setup

- TLBCoat implementation in Gem5:
  - Full Linux environment on a simulated HiFive Unleashed board
- Standalone cache simulator:
  - Noise-Free functional simulation
  - Easy access to important internal data

Security Evaluation

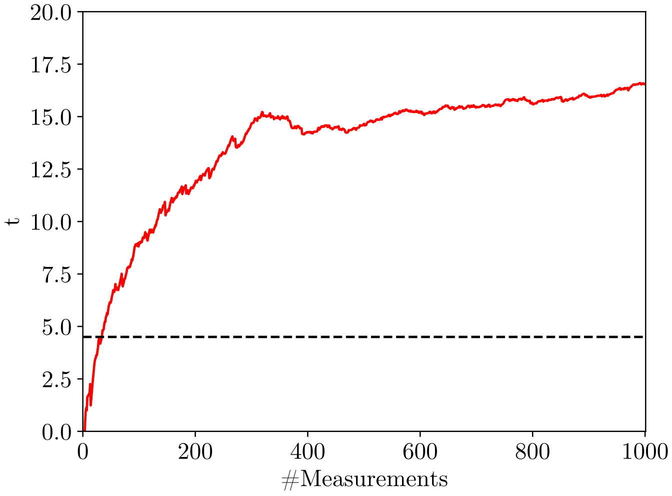
RUB

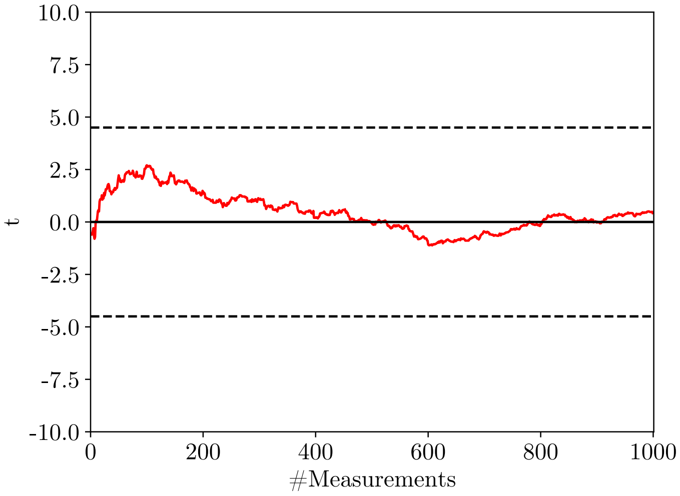
- fixed vs. fixed t-test on a standard set-associative TLB and TLBCoat
  - Victim either performs an access within an eviction set or not
  - Standard TLB: Strong leakage, on average up to 3 cycles delay if an access happened



Security Evaluation

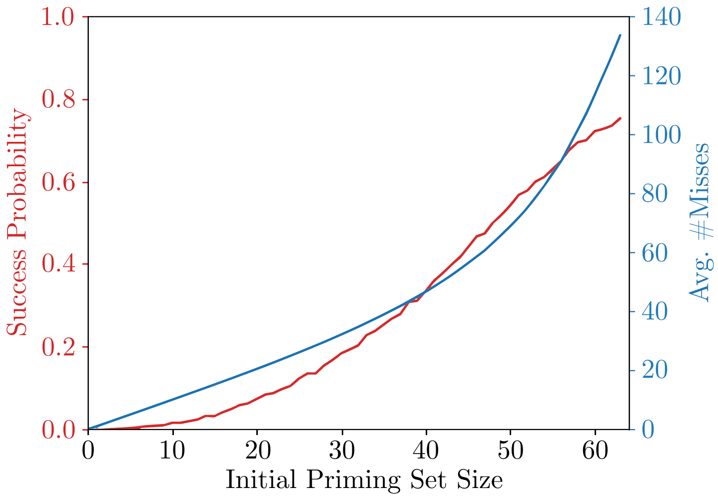
- fixed vs. fixed t-test on a standard set-associative TLB and TLBCoat
  - Victim either performs an access within an eviction set or not
  - Standard TLB: Strong leakage, on average up to 3 cycles delay if an access happened
  - TLBCoat: No significant leakage, placement of victim unpredictable





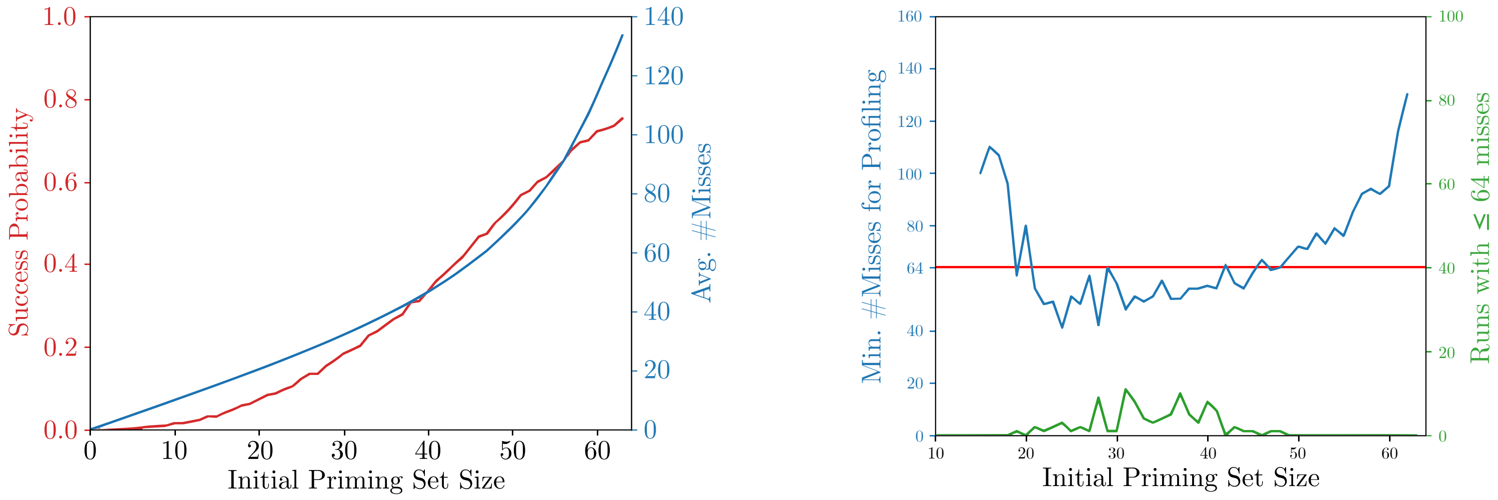
Security Evaluation

- How difficult is eviction set creation for Prime + Prune + Probe?
  - Scenario: Pick a random range of addresses, no rerandomization
  - Larger priming set size → Better chance to evict victim, but more misses



Security Evaluation

- How difficult is eviction set creation for Prime + Prune + Probe?
  - Scenario: Pick a random range of addresses, rerandomization enabled
  - Best results for size 24 → After 100,000 attempts less than 20 were successful
  - Remember: These numbers are the best-case scenario!



Performance Evaluation

- Miss-to-Hit ratio for a low-noise system using PARSEC:

Benchmark	Standard TLB	TLBCoat RPLRU	TLBCoat LRU
Blackscholes	10.99%	0.02%	0.02%
Canneal	7.20%	9.20%	8.73%
Dedup	0.05%	0.04%	0.04%
Fluidanimate	0.41%	0.45%	0.44%
Freqmine	0.17%	0.23%	0.21%
Streamcluster	0.22%	0.61%	0.21%
Swaptions	<0.01%	<0.01%	<0.01%

Performance Evaluation

- Hardware overhead (15nm @ Silvaco's Open-Cell):

Module	Area (GE)
Randomization	2253.76
Ways	47912.05
RPLRU Units	2936.00
Comparator	501.50
Other	1209.98

- Total Delay of TLBCoat: 0.143 ns

Conclusion

- TLBCoat employs randomization to protect against state-of-the-art attacks
- Only small OS modifications required → Miss count and rid saved during context switch
- Hardware modifications do not significantly impact the area or critical path

Thank you!

Any Questions?