Dynamically Finding Minimal Eviction Sets Can Be Quicker Than You Think for Side-Channel Attacks against the LLC

Wei Song

State Key Laboratory of Information Security
Institute of Information Engineering, CAS, Beijing, China

Peng Liu

The Pennsylvania State University, University Park, USA

2019-09-25

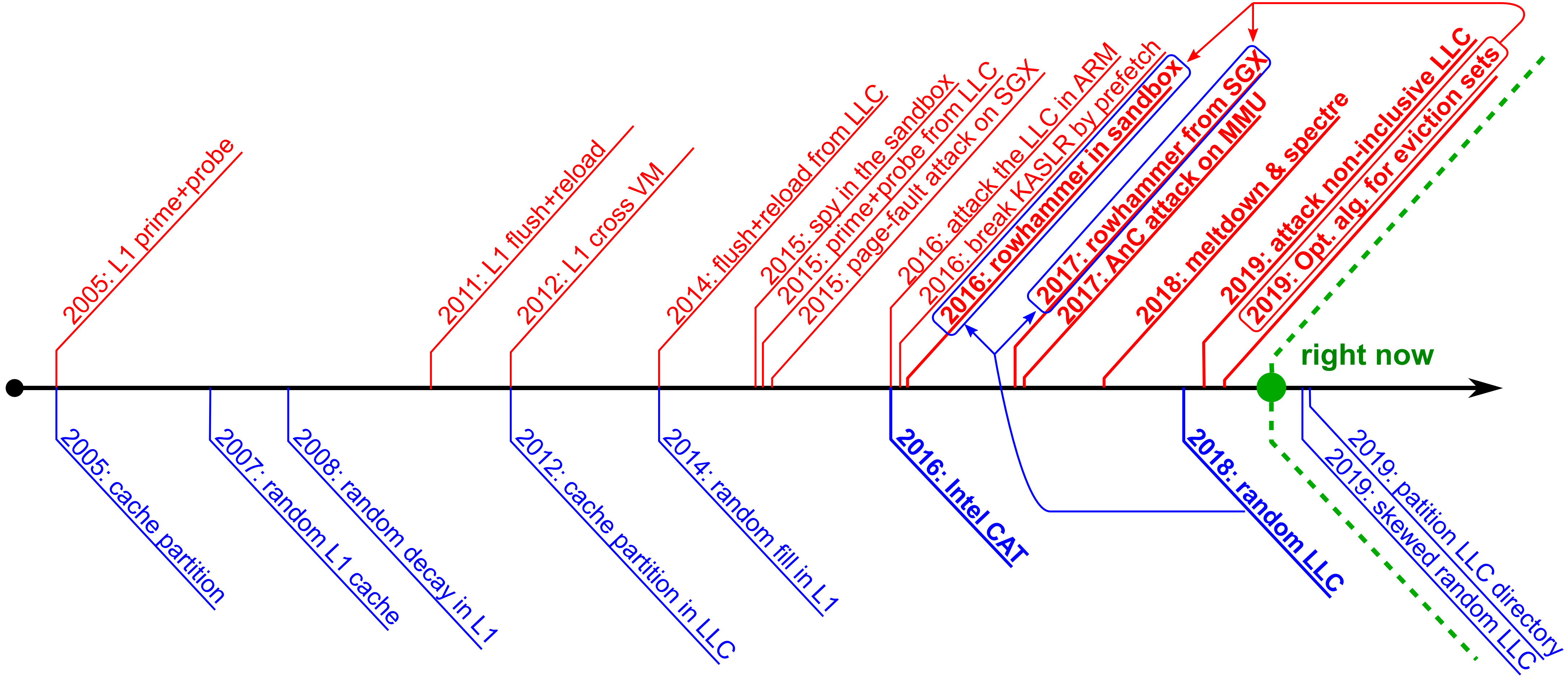
SKLOIS 信息安全国家重点实验室

中国科学院信息工程研究所 INSTITUTE OF INFORMATION ENGINEERING,CAS

PennState

Overview

- The development of cache-side channel attacks and defenses.



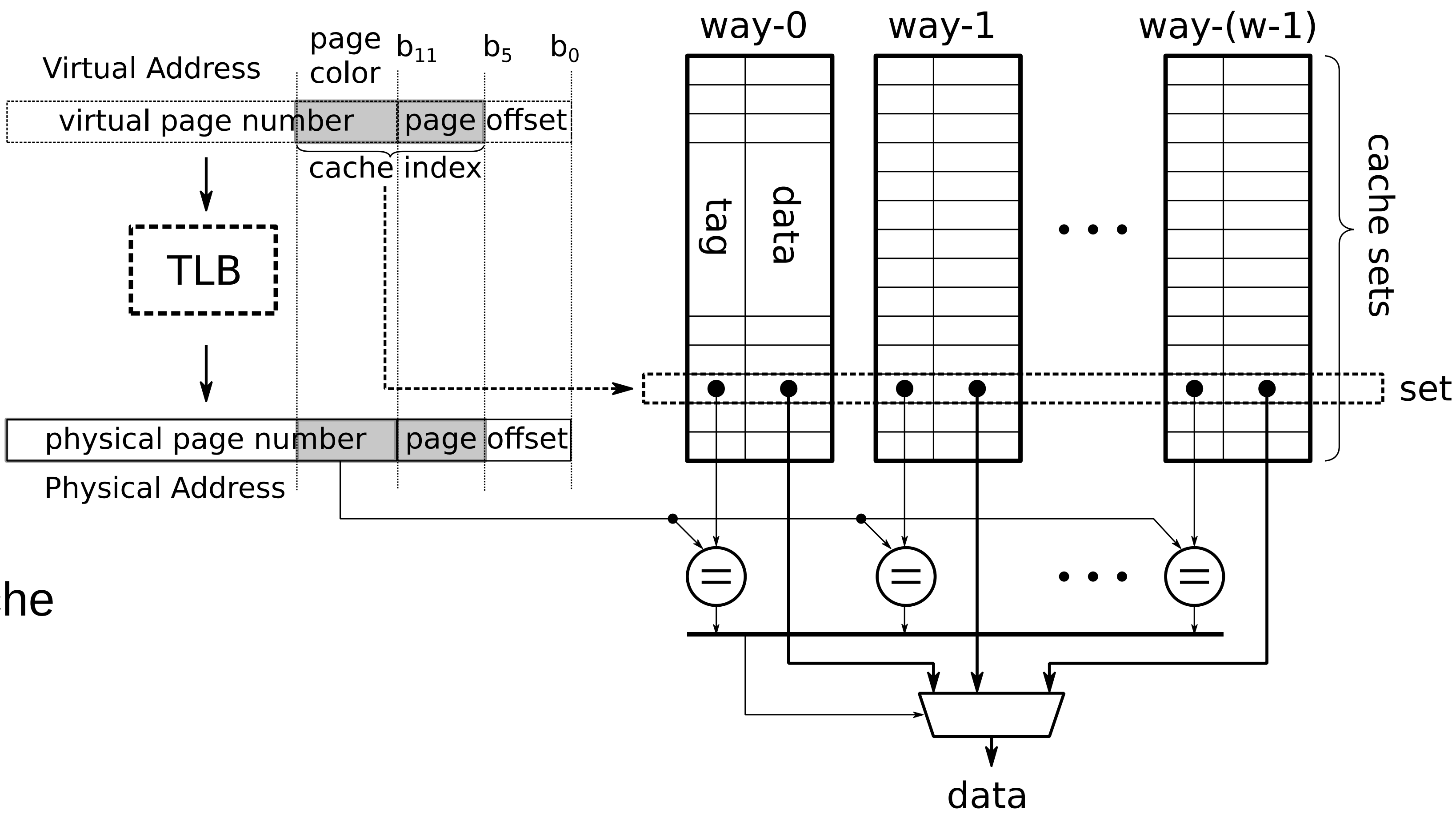
Our Motivations

- Dynamically randomized LLC
  - [Qureshi2018] CEASER: Mitigating conflict based cache attacks via encrypted-address and remapping. (Micro'18)
  - Randomized LLC → **Dynamically finding eviction sets**
    → **Uncontrollable set conflicts**
  - Dynamic remapping → **Limit attacks in short period**
- Optimized eviction set search algorithm
  - [Vila2019] Theory and practice of finding eviction sets. (S&P'19)
  - Prune eviction sets in groups → **Reduce time from $O(n^2)$ to $O(w^2 n)$**

Fast enough?

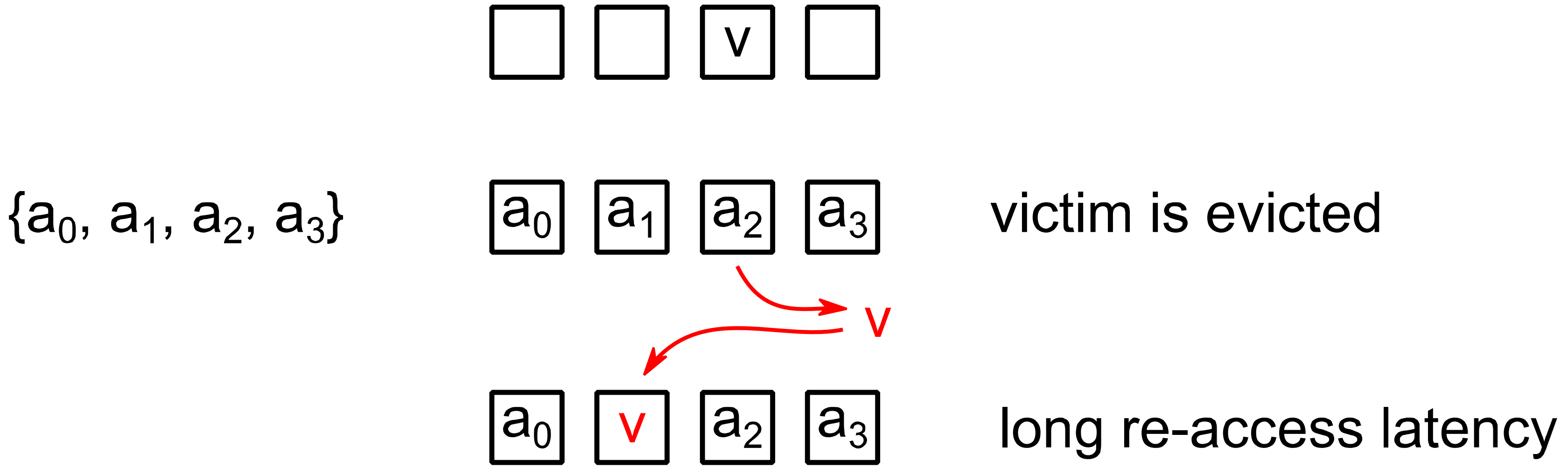
- **Our questions:**
  - **In theory, how fast can an adversary find a minimal eviction set?**
  - **In practice, how fast can a minimal eviction set be found on modern processors?**

Preliminary: Caches and Virtual Memory



- set associative cache
- MMU, TLB
- VIPT, PIPT

Prime+Probe

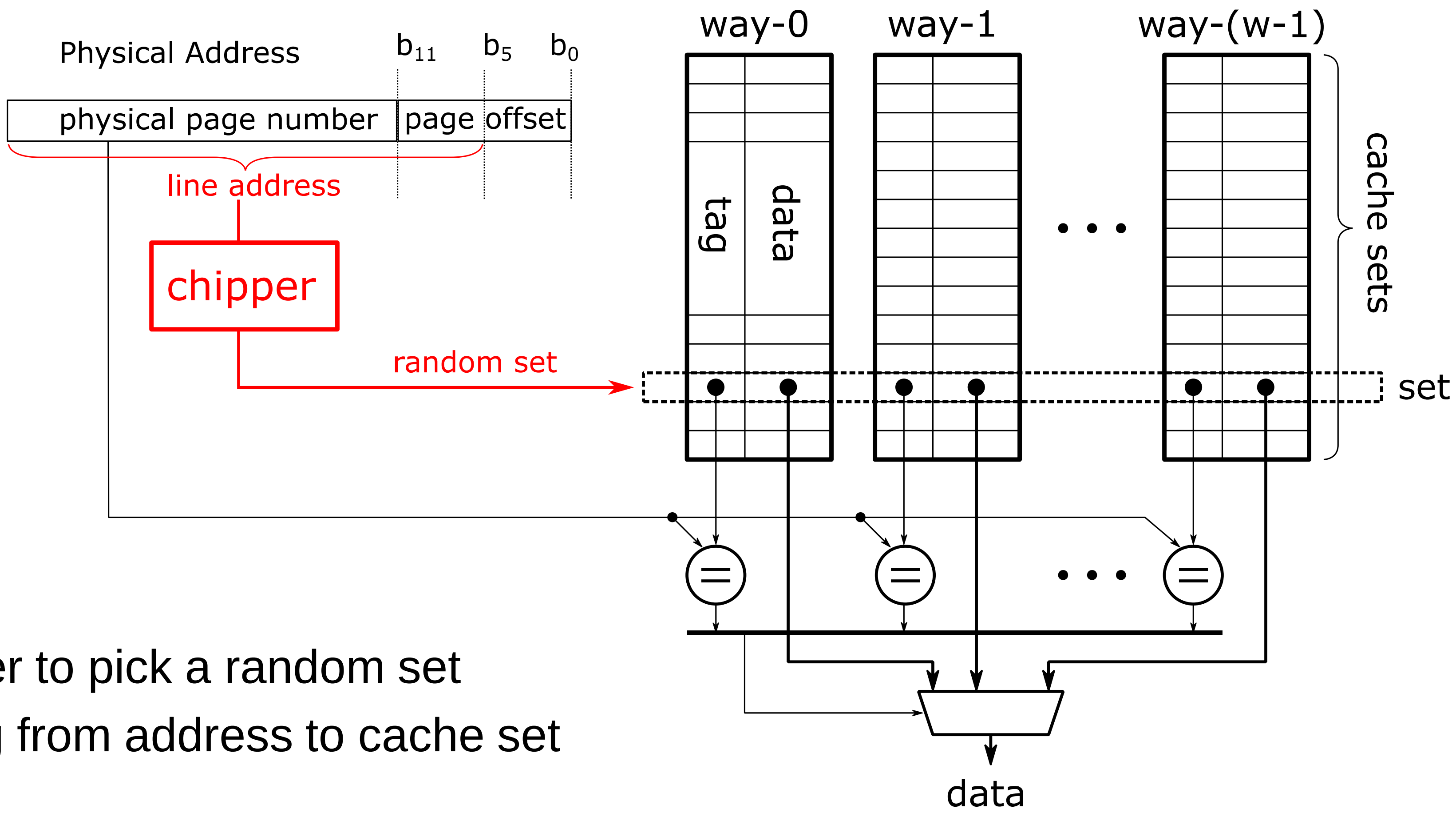


- Victim accesses v.
- Attacker primes the set with an eviction set $\{a_0, a_1, a_2, a_3\}$, force the eviction of v.
- Victim re-accesses v incurs a long delay.

$\{a_0, a_1, a_2, a_3\}$ and v are mapped to the same set (congruent)

Usually eviction sets are computed rather than found.

Randomized LLC (CEASER)



- Use a block chipper to pick a random set
- Break the mapping from address to cache set

Finding an Minimal Eviction Set

- A minimal eviction set
  - An eviction set with **the smallest number (w)** of **congruent cache blocks**.
  - Congruent cache blocks: cache blocks mapped to the same cache set.

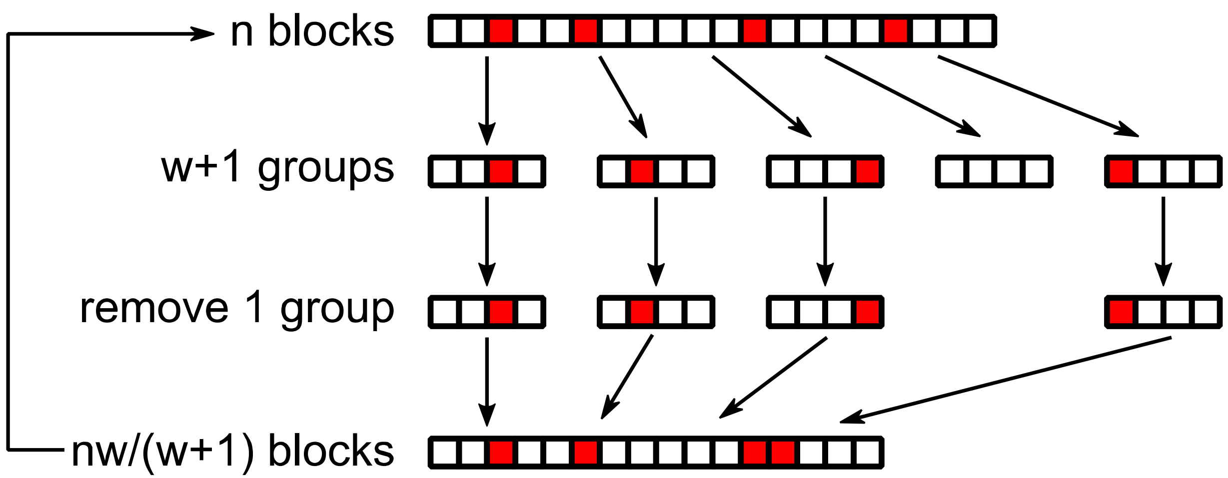
- Assumption
  - Current Intel processors: VPN to PPN mapping is unknown, PPN considered random.
  - CEASER: cache set is considered random.

- Solution
  - Find **a big eviction set (candidate set)** with a large number (n) of random cache blocks.
    - When n is large enough, we can evict any cache block in the shared LLC [Hund2013].
  - Prune the large set into **a minimal one**.

Prune an Eviction Set (the optimized way)

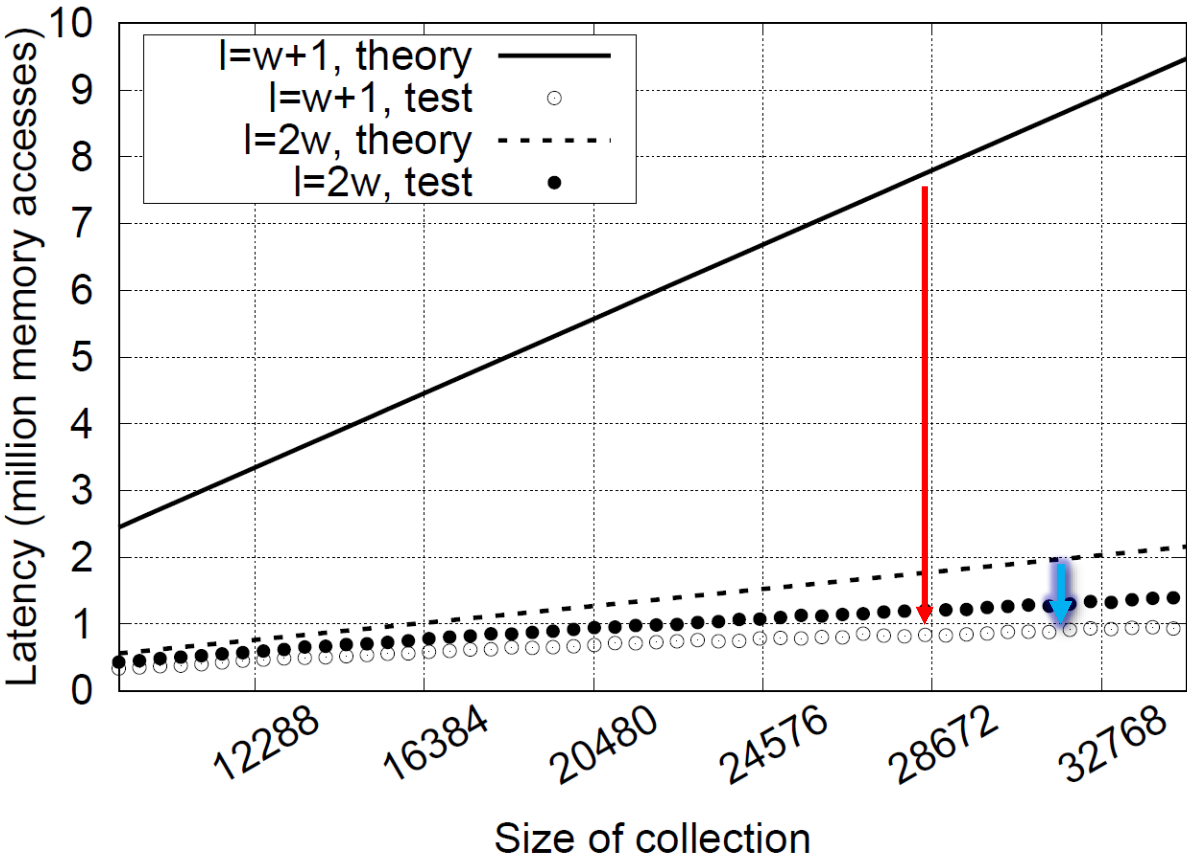
- Original method [Liu2015 at S&P, Oren2015 at CCS]
  - Remove one cache block per iteration $O(n^2)$
- Optimized method (group pruning) [Vila 2019 at S&P]
  - Assume we have an initial eviction set with n blocks for a 4-way cache.
  - By dividing them into $w + 1$ groups, **time complexity is reduced to $O(w^2 n)$.**

Is this a good estimation?



The actual latency is much smaller!

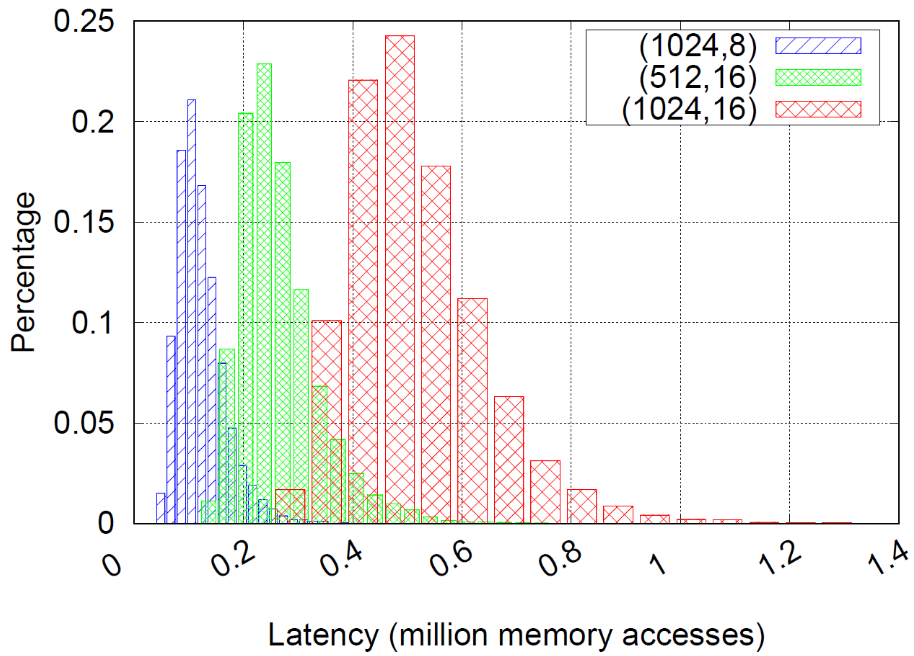
- The actual latency is much smaller
  - Early termination effect: terminate the iteration whenever the first removable group is found.
- Divide by $2w$
  - Use $2w$ rather than $w+1$ reduce the theoretical bound to $(\mathbf{4w}-\mathbf{2})\boldsymbol{n} \rightarrow \boldsymbol{O}(\boldsymbol{wn})$
  - Much closer to the actual latency
  - Actual test using $2w$ is slightly worse than $w+1$ due to the reduced early termination effect.
- Even $(4w-2)n$ is not good enough!

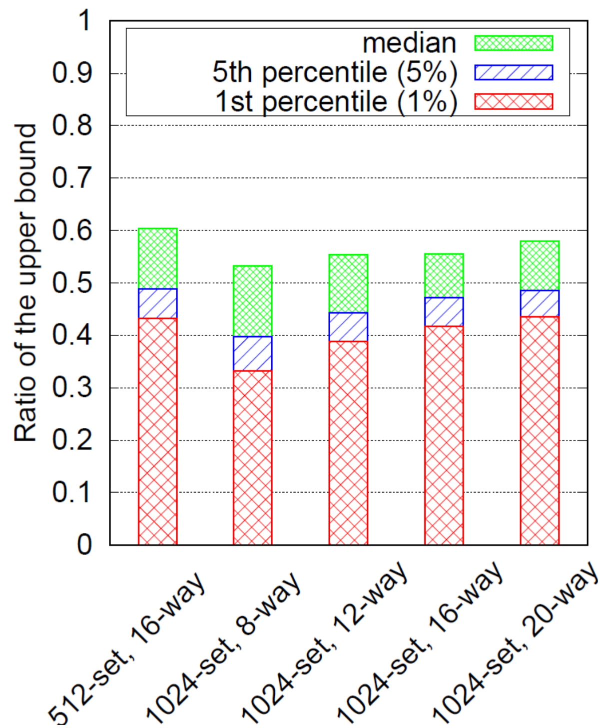


The long tail distribution of latency

- The actual latency distribution is a long tail.
  - For a defense, what actually matter is the location of the left boundary (1st percentile, 1% of attacks).
  - For a 1024-set 16-way randomized cache, 1st percentile $\approx 25n, n = 11500$
    0.2% of n^2, around $18 \cdot s \cdot w$!

This is much faster than we ever thought!



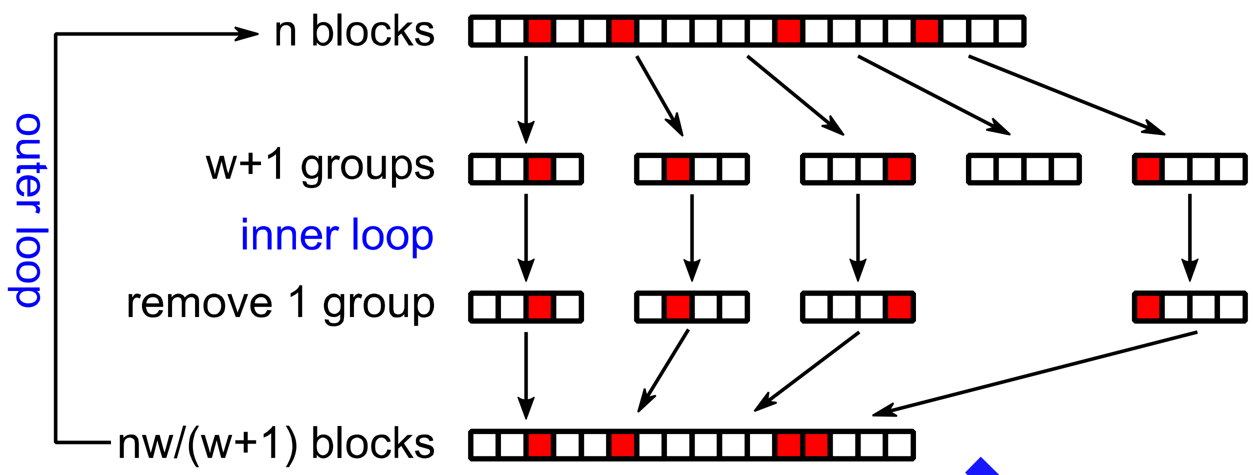


What about Actual Processors?

- Applying the dynamic search on three Intel processors.

	i7-3770	Xeon-4110	i7-8700
Architecture	IvyBridge	Sky Lake	Coffee Lake
Cores	4	8	6
Threads	8	16	12
LLC Size	8 MB	11 MB	12 MB
Cache Way	16	11	16
Memory	4 GB	32 GB	32 GB
OS	Ubuntu 16.04	Ubuntu 16.04	Ubuntu 18.04

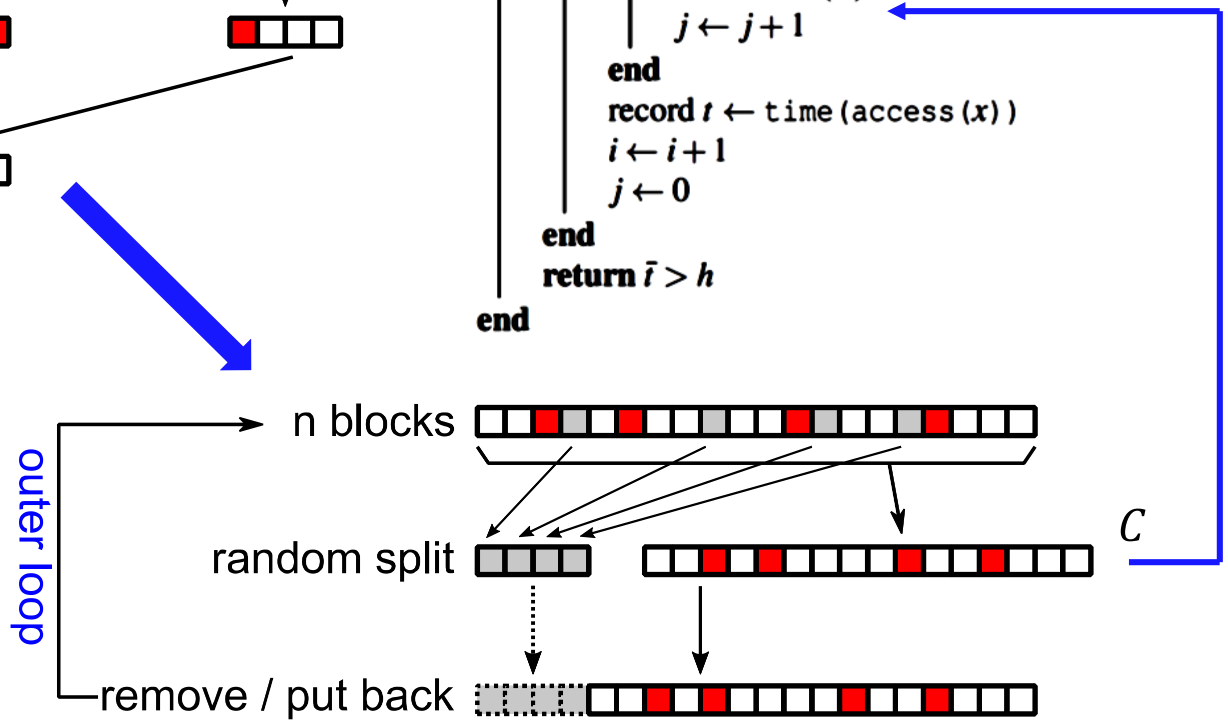
Improve the Pruning Algorithm




```
function test(C,x)
    i ← 0, j ← 0
    while i < b do
        access (x)
        while j < d do
            traverse (C)
            j ← j + 1
        end
        record t ← time(access(x))
        i ← i + 1
        j ← 0
    end
    return t̄ > h
end
```

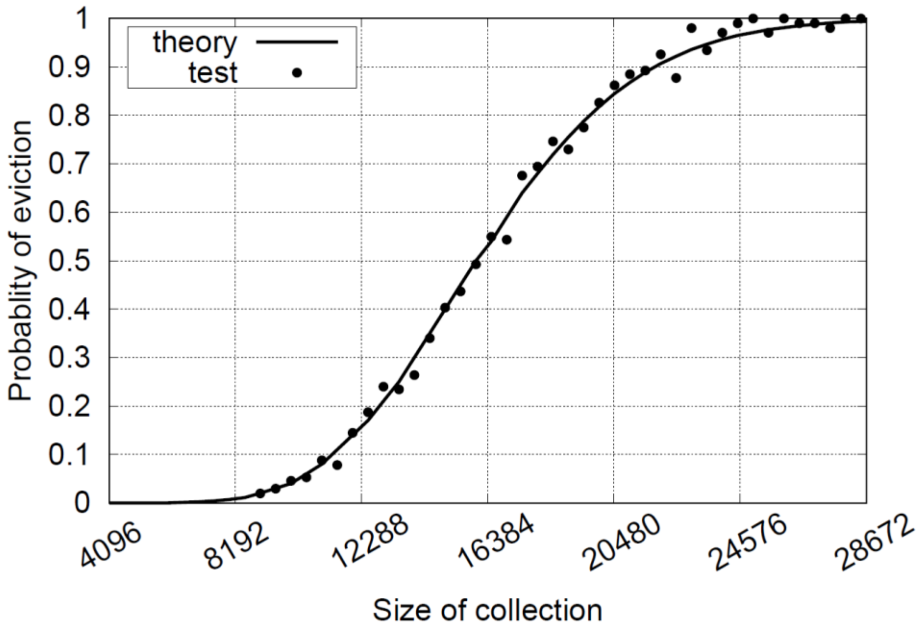
Test C with repeat parameter (b, d)

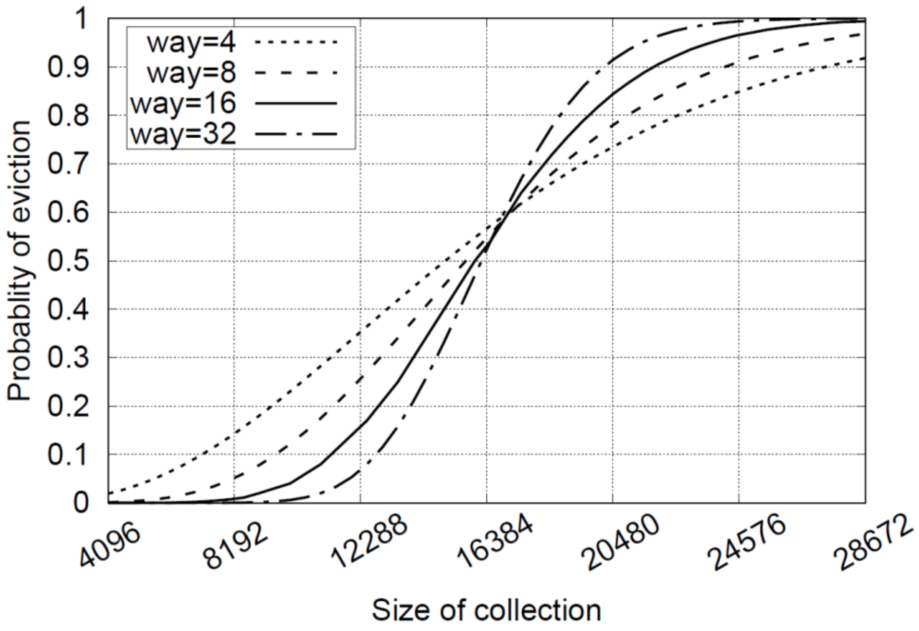
- Random split 1/(w+1)
- Simpler loop control
- Better tolerance to noise



The Optimal Candidate Set Size?

- How many random cache blocks are enough to get a large eviction set?





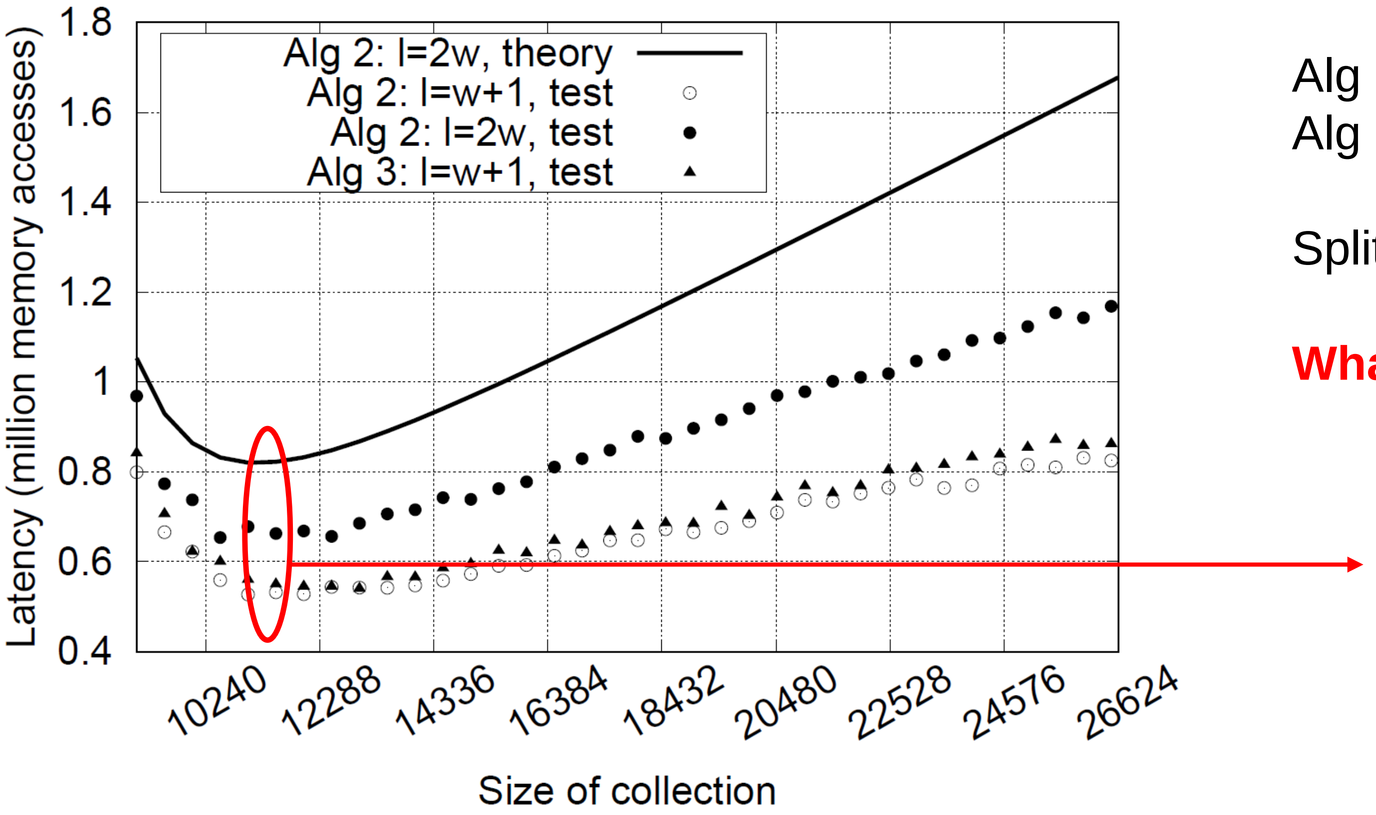
1024-set 16-way cache
$\sim 16\text{K} \rightarrow 50\%$ probability of eviction

512-set 32-way
1024-set 16-way
2048-set 8-way
4096-set 4-way

Magic 60%

The Optimal Candidate Set Size?

- How many random cache blocks are enough? **Slightly less than s*w.**



Alg 2: Group prune [Vila2019]
Alg 3: Random split [this paper]

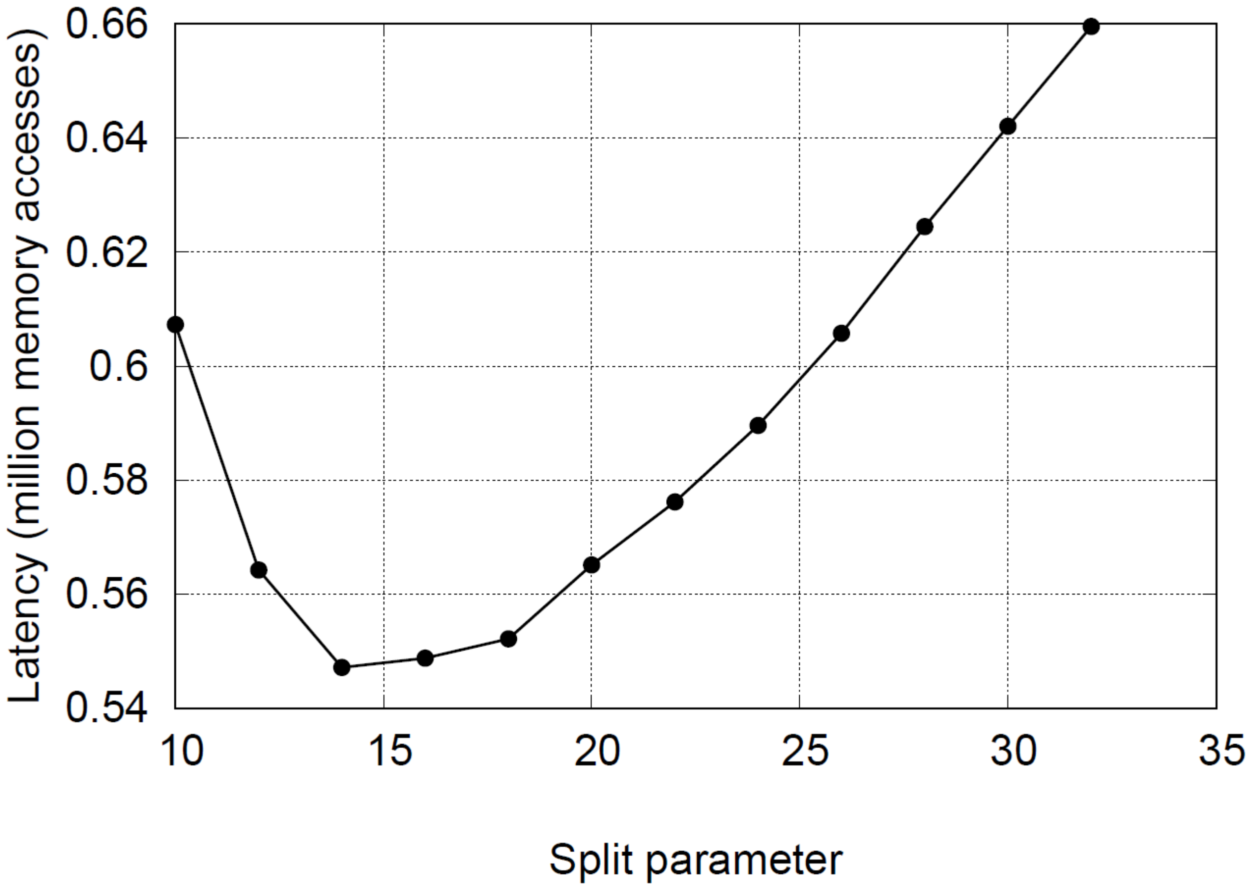
Split ratio: w+1 is better than 2w

What is the best split ratio?

Less than 50% chance in finding a candidate set but much shorter time in pruning.

What is the Best Split Rate?

- Is w+1 the best split rate? **No.**



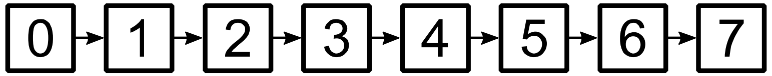
1024-set 16-way

The best split rate ~14

Slightly less than w+1.

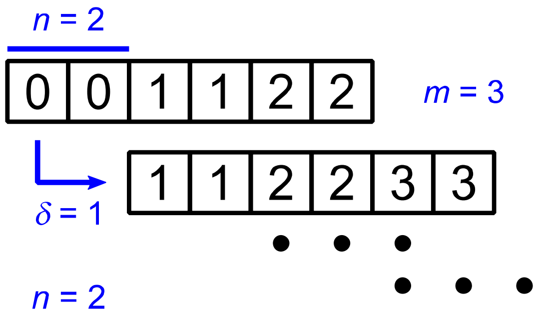
What is the Best Traverse Function?

- Start from Ivy bridge (2012), anti-threshing replacement is utilized.



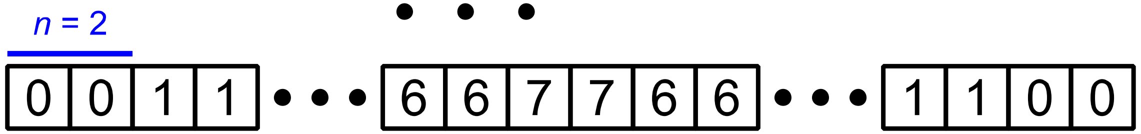
- Traverse strategy [Gruss2016]

$$strategy(m = 3, n = 2, \delta = 1)$$
$$list(m, n) = strategy(m, n, 1)$$



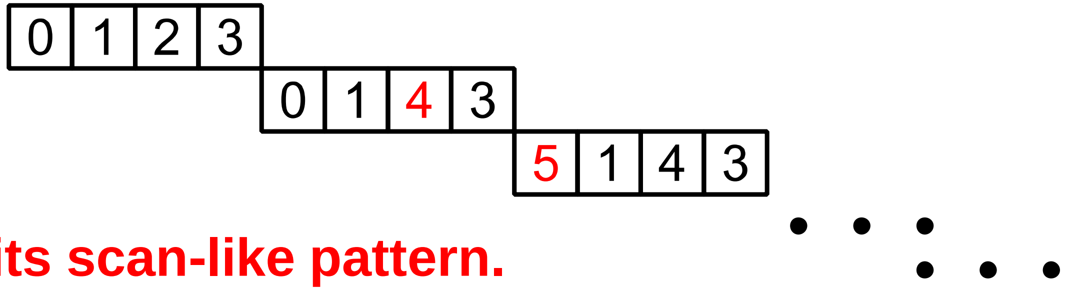
- Round traverse [Liu2015]

$$round(n = 2)$$



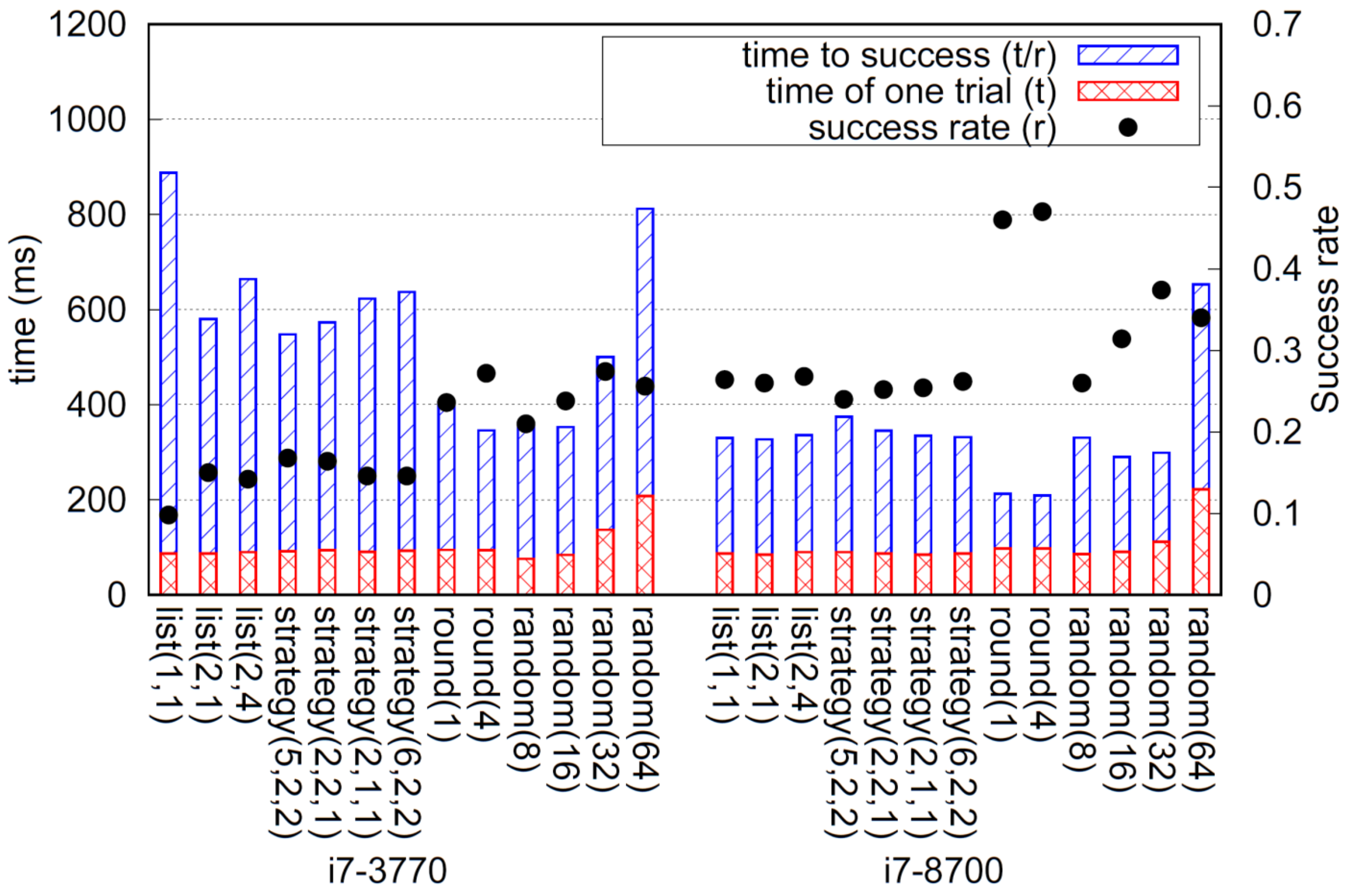
- Random traverse [this paper]

$$random(4)$$



They key: disguise its scan-like pattern.

What is the Best Traverse Function?



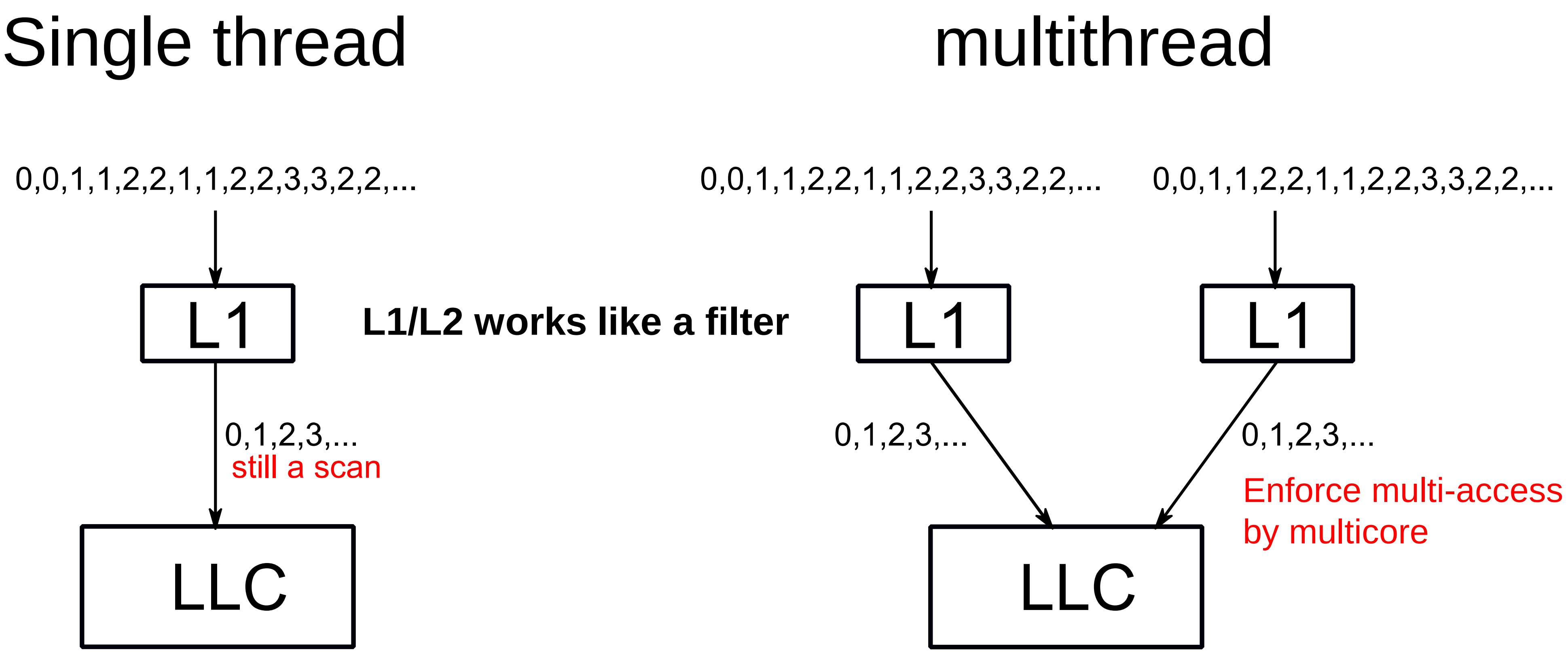
Time to success = $\frac{\text{time of one trial}}{\text{success rate}}$

i7-3770: round(4) and random(16)

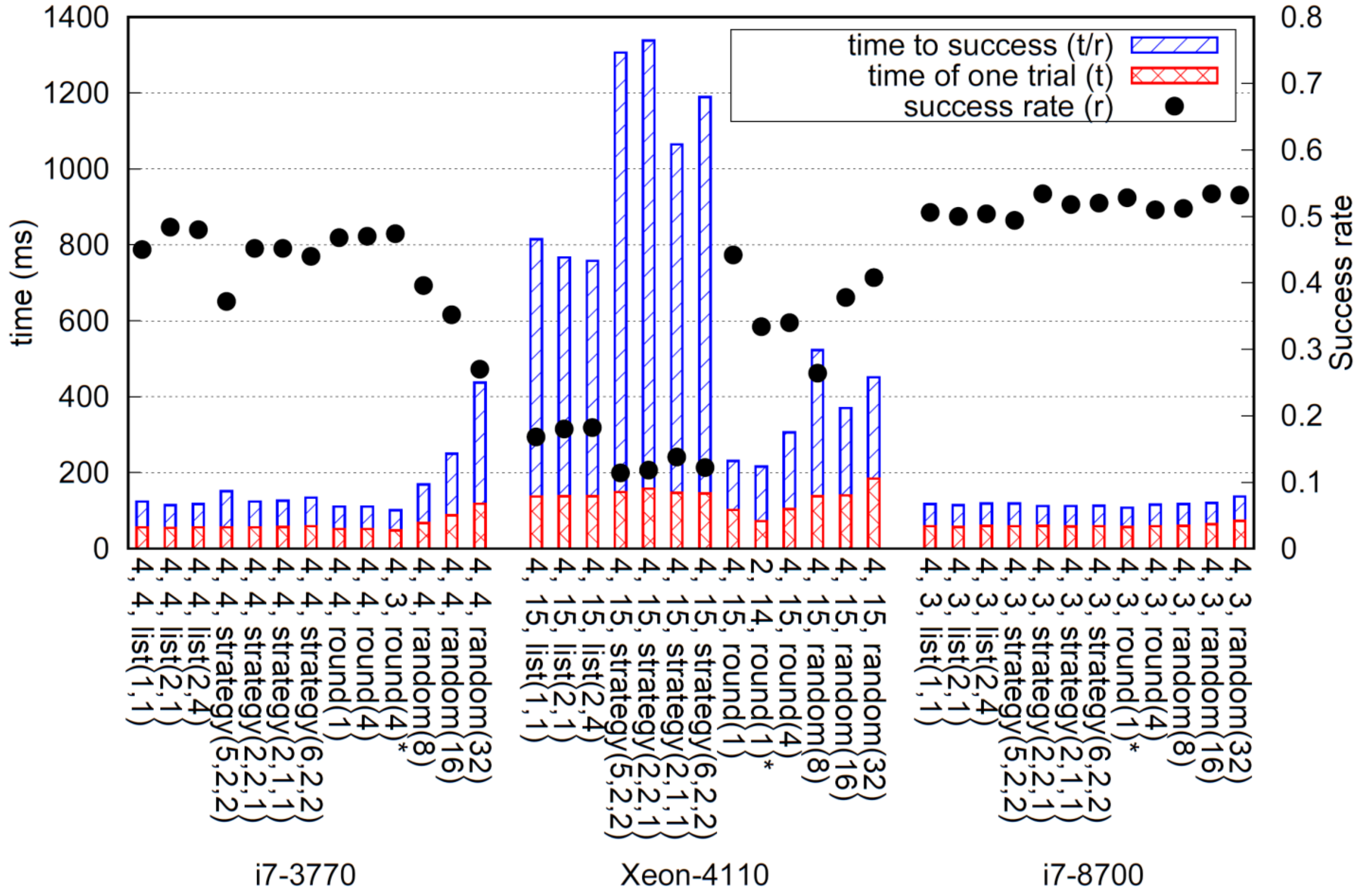
i7-8700: round(4)

Xeon-4110: failed!

Improve the Success Rate: Multithread Traverse



Now We Succeed on Xeon-4110



i7-3770: round(4) Xeon-4110: round(1) i7-8700: round(1)

Summary of Techniques

Parameter	Good Setting	Proposed by
prune algorithm	Algorithm 3	this paper
repeat parameter	$b = 4, d = 4$	[7, 26]
multithreading	enable	this paper
traverse function	$round(1)$	[18, 26]
rollback	$rb(2)$	[26]
reuse failed set	$ru(1)$	this paper
TLB preload	enable	[5]
huge page	enable	[26]
retry limit	$rt(4w)$	this paper
candidate set	$n = s \cdot w/64$	this paper
split parameter	$l = w$	this paper

Results: When VA to PA mapping is unknown

- **Finding eviction sets at the page granularity**

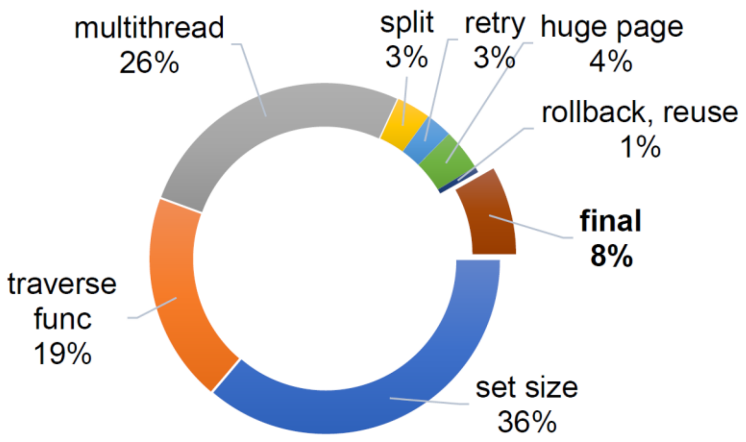
	Single Thread Normal Page	Single Thread Huge Page	Multithread Normal Page	Multithread Huge Page
i7-3770	0.150 s	0.091 s	**0.085 s**	0.060 s
Xeon-4110	Failed	Failed	**0.170 s**	0.134 s
I7-8700	0.202 s	0.123 s	**0.095 s**	0.061 s

- **Compare with optimized [Vila2019]**

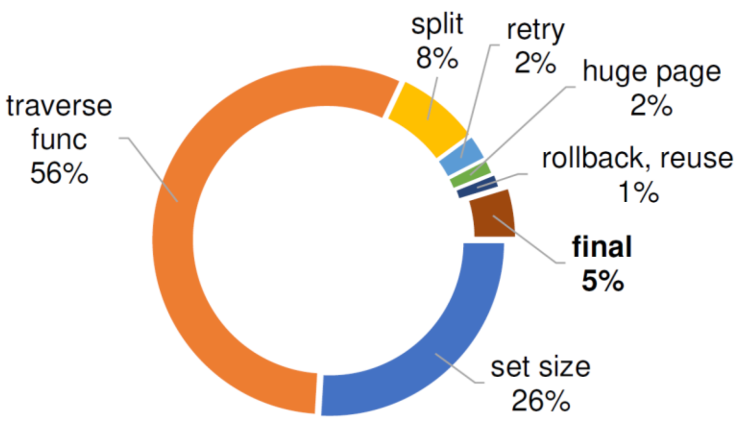
	Normal Page		Huge Page	
	Latency	Reduction	Latency	Reduction
i7-3770	0.477 s	**-82.1%**	0.219 s	**-72.6%**
i7-8700	0.244 s	**-61.1%**	0.186 s	**-67.2%**

Improve success rate from ~60% to ~90%.

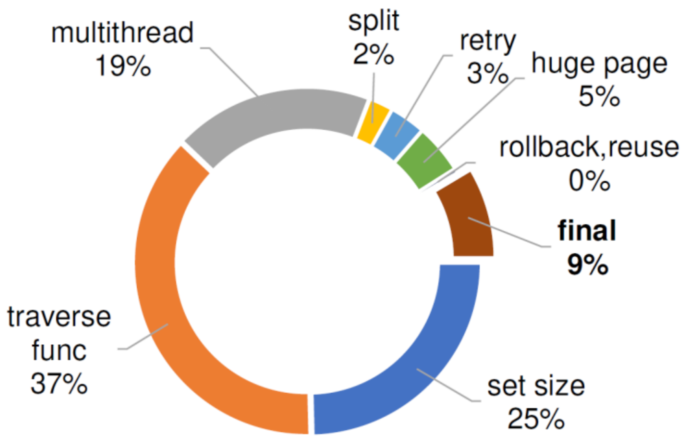
Results: Contribution of Individual Techniques



i7-3770: from 0.732s to 0.060s (8.2%).



Xeon-4110: from 2.838s to 0.134s (4.7%).



i7-8700: from 0.715s to 0.061s (8.5%).

Results: When LLC is Randomized

- **Finding eviction sets at the cache block granularity.**

	Granularity	Multithread	Huge Page	Candidate Size	Repeat Parameter (b,d)	Traverse Function	Retry (rb)	Split Parameter (l)	rollack (rb)	Reuse Failed Set (ru)	TLB Preload	Success rate	Time of a Single Trial	Time to Success
i7-3770	**64B**	Y	N	162000	$(5,7)$	$round(4)$	80	16	2	10	Y	0.390	43.98s	1.88m
Xeon-4110	**64B**	Y	N	281600	$(5,18)$	$round(1)$	48	9	2	50	Y	0.980	1.70m	1.74m
i7-8700	**128B**	Y	N	112000	$(4,3)$	$round(4)$	64	16	2	10	Y	0.100	63.3s	10.6m

  - We Succeed both on i7-3770 and Xeon-4110 but failed on i7-8700.
  - Although it is slow, it is a demonstration that we can find eviction sets on a (statically) randomized LLC.

Conclusion and Future Works

- **Contributions:**
  - Reduce the bound from $O(w^2 n)$ to $O(wn)$.
  - CEASER has overestimated the latency (confirmed by [Qureshi2019 at ISCA]).
  - New techniques to reduce the latency to ~0.1 second.
  - Multithread traversing (Xeon-4110, non-inclusive LLC [Yan2019 at SP]).
  - First time to find eviction set without fixing page offset.

- **Future works:**
  - Non-inclusive LLC (AMD snooping protocol) [Yan2019 at S&P]
  - Skewed random LLC [Werner2019 at Security, Qureshi2019 at ISCA]

- Opensource
  - The ideal cache model: https://github.com/comparch-security/cache-model
  - Tests on Intel processors: https://github.com/comparch-security/smart-cache-evict

Thank you!
Any Questions?

中國科學院 信息工程研究所
INSTITUTE OF INFORMATION ENGINEERING,CAS