中国科学院大学
University of Chinese Academy of Sciences

论文Review: Theory and Practice of Finding Eviction Sets (S&P 19') CCF-A

Pepe Vila[1,3], Boris Köpf [2], José F. Morales[1]

1 institute imdea software
2 Microsoft
3 UNIVERSIDAD POLITÉCNICA DE MADRID POLITÉCNICA

汪**，罗**，李敬，赵**，常**

Jun. 1, 2021

辛丑孟夏廿一

Huairou, Beijing

中国科学院计算技术研究所 INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES

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

Executive summary

Problem：

- **寻找小驱逐集是microarchitectural攻击的核心步骤**
  - ▶现在的方法效率低、且零散、不成系统体系

Key Idea：

- **群组检测**

Mechanisms ：

- **先分组，分组数维持恒定，再观察每组里是不是去掉它剩下的依然是驱逐集，如果是的话，就把这个组去掉。**

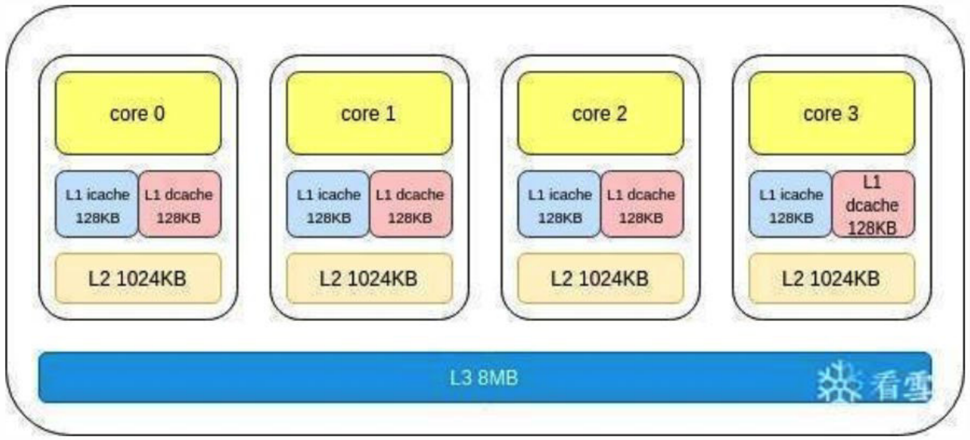
Results ：

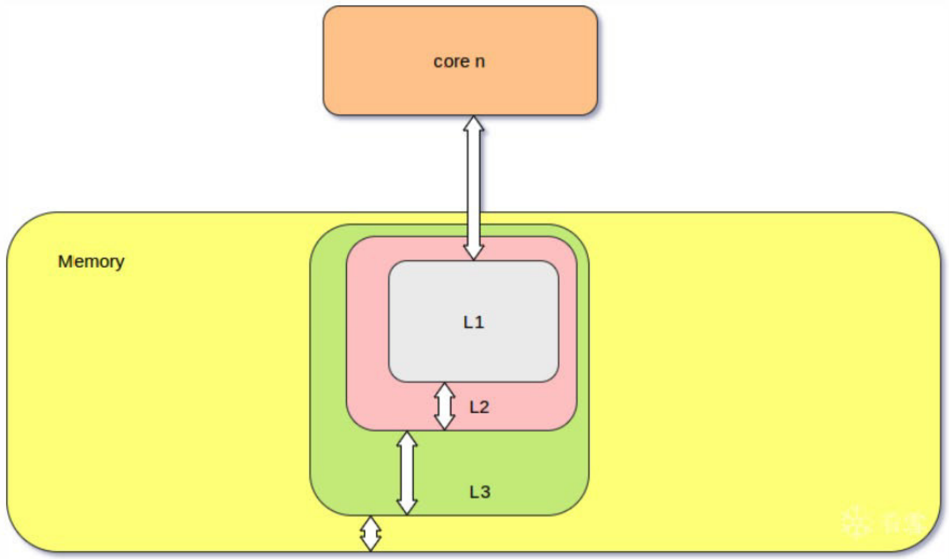
- **Find eviction sets in O(n) compared to previous O(n^2)**
- **严格的实证评估证明**

Overview

Background, Problem & Goal

Cache Architecture

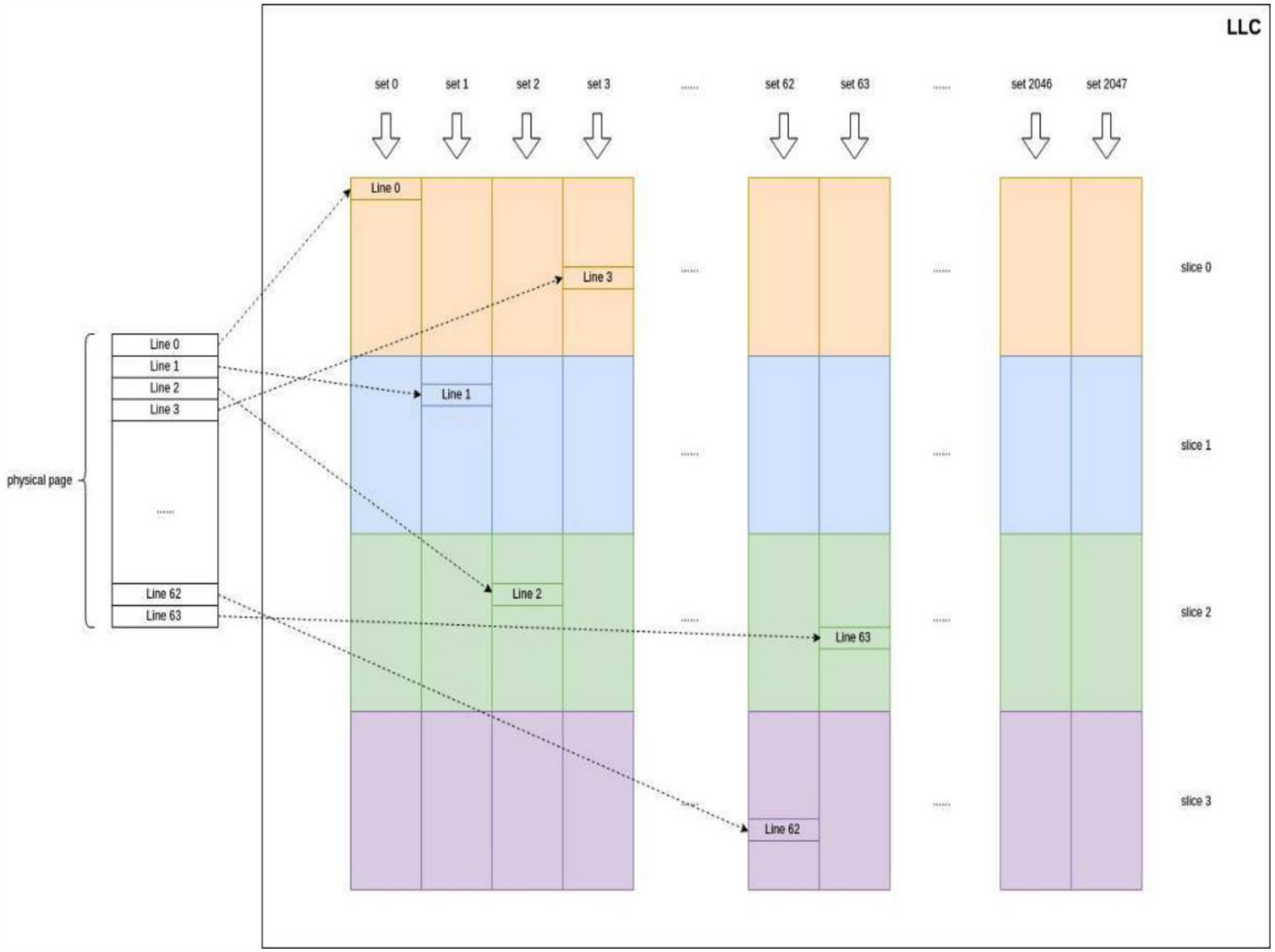




Haswell缓存架构和Intel缓存嵌套结构

Source: https://bbs.pediy.com/thread-256190.htm

L3 Cache (LLC) Architecture

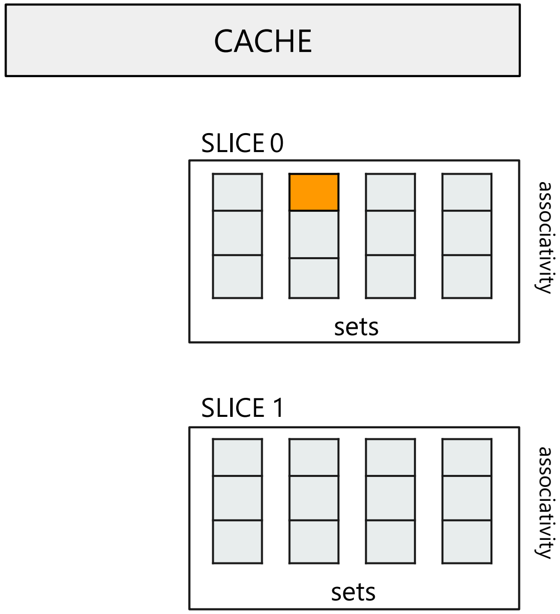


Intel core i7 4790的L3结构

Source: https://bbs.pediy.com/thread-256190.htm

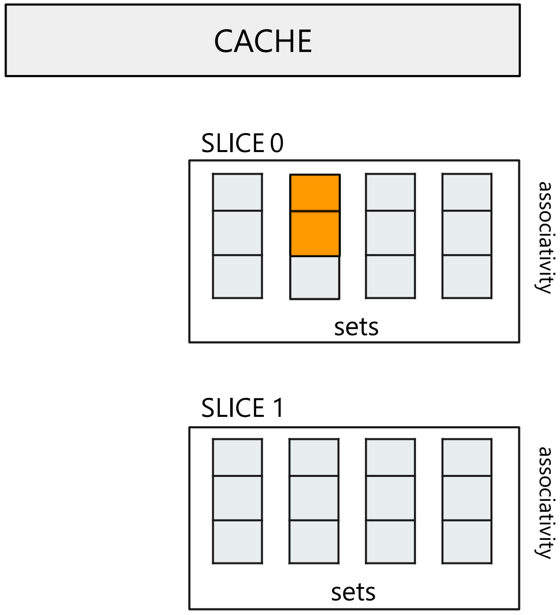
驱逐集 Eviction Sets

Find addresses that collide in cache: i.e. addresses mapped into the same cache set



驱逐集 Eviction Sets (Cont.)

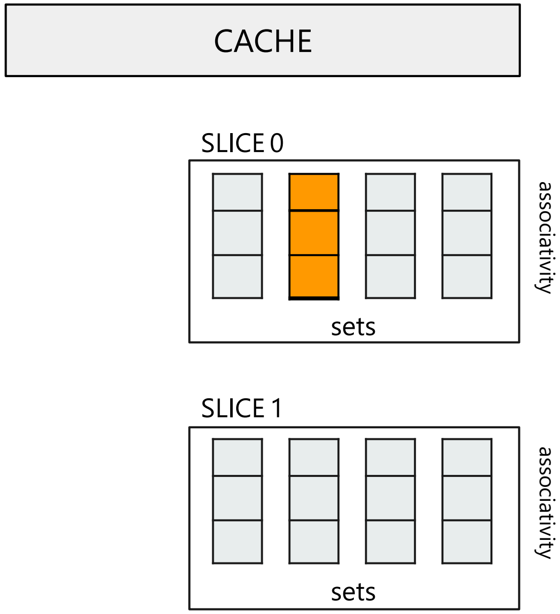
Find addresses that collide in cache: i.e. addresses mapped into the same cache set



驱逐集 Eviction Sets (Cont.)

Find addresses that collide in cache: i.e. addresses mapped into the same cache set

Find associativity many colliding addresses: i.e. an **eviction set**



Recap: evict-base Attack

Efficient attacks require small eviction sets

Spectre

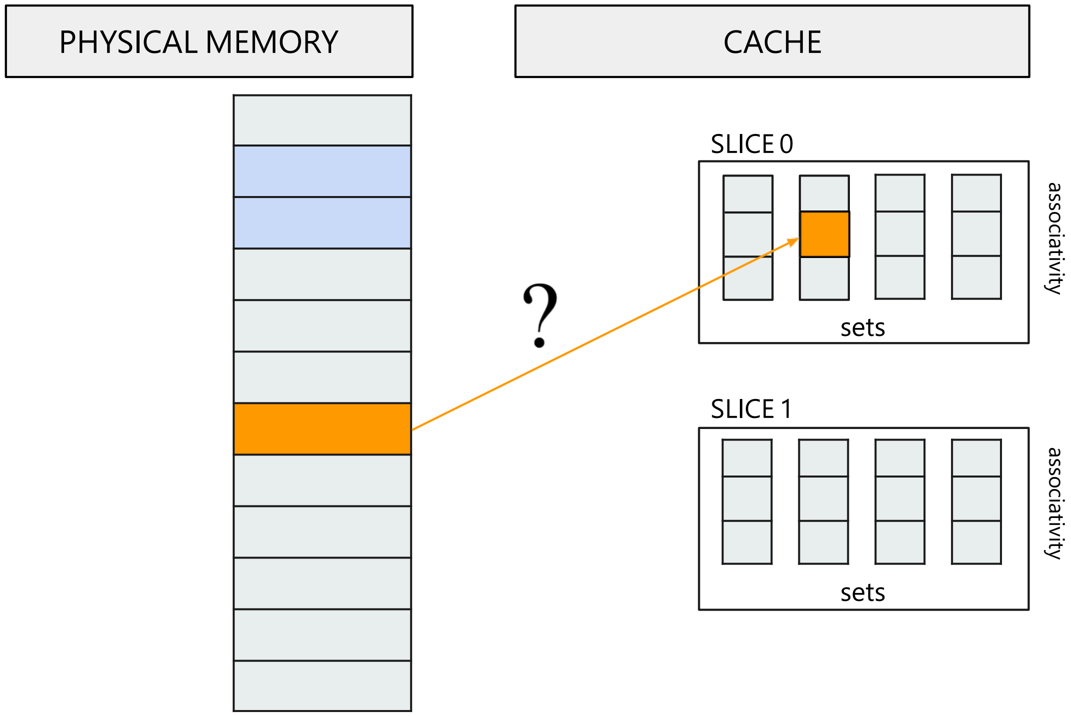
Prime+Probe

Rowhammer

NOT academic stunt AGAIN

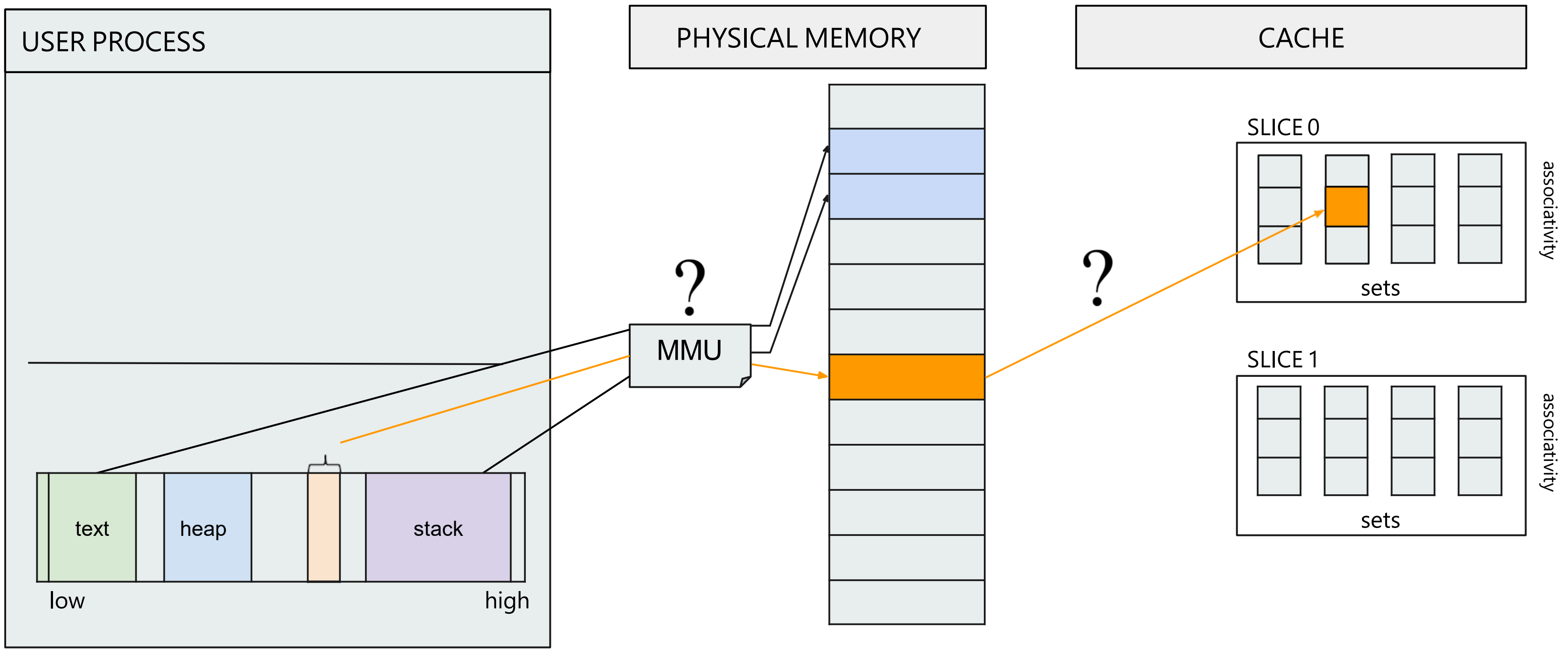
Problem

Potentially unknown mapping from physical address to cache set



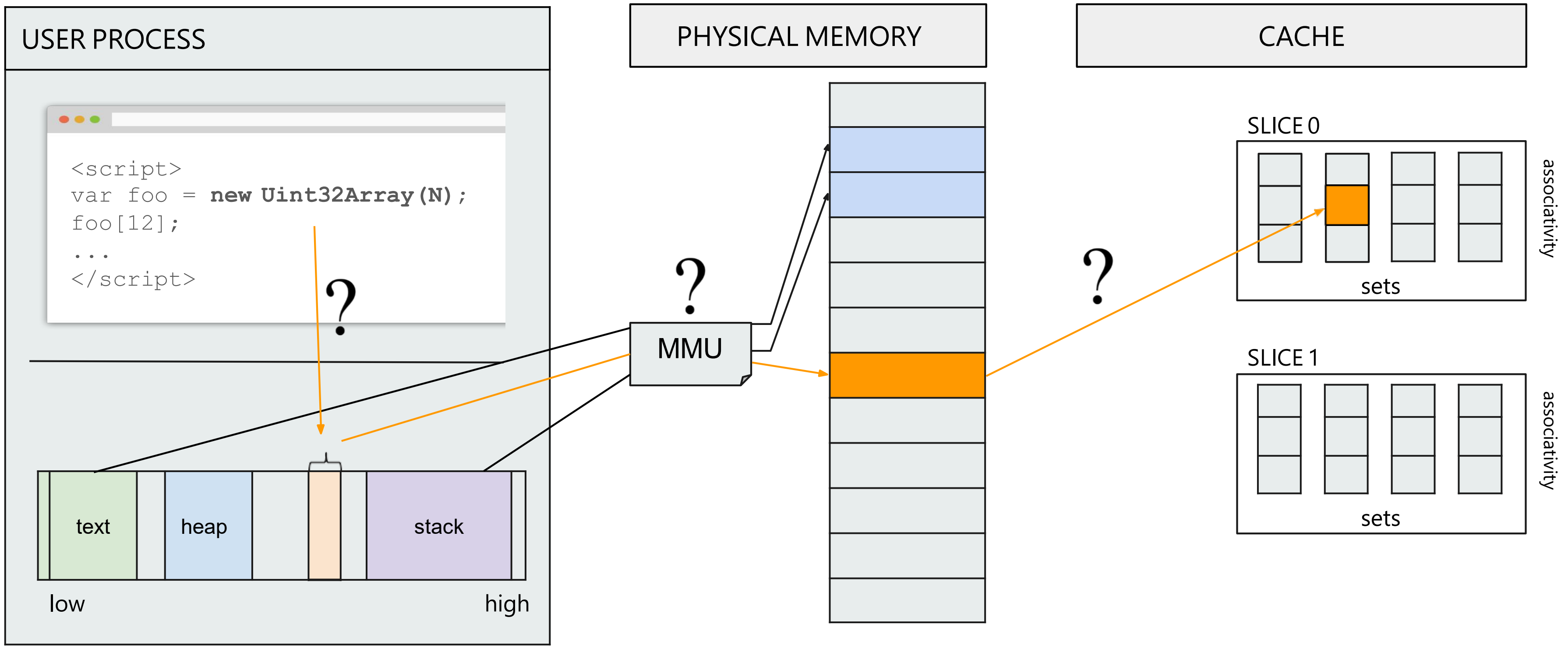
Problem (Cont.)

Unknown translation from virtual to physical addresses



Problem (Cont.)

In some scenarios, even unknown virtual address



Find associativity many elements (e.g. JS array indices) that collide in cache.

Goal

系统地从理论和实践证明两方面研究寻找最小驱逐集的新方法

天下武功 唯快不破

具体而言：

- 寻找最小驱逐集的算法
- 时间复杂度：线性时间
- 算法的实验评估

Goal (Cont.)

系统地从理论和实践证明两方面研究寻找最小驱逐集的新方法

前人工作：

寻找驱逐集的动态和静态方法：动态方法使用计时测量来识别冲突地址，而不知道LLC slice哈希函数或物理地址；而静态方法使用反向工程散列和关于物理地址的(部分)信息来计算逐出集。

Daniel Gruss and Clémentine Maurice and Stefan Mangard, “Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript,” in DIMVA, Springer, 2016.

完全静态：

在没有slicing的cpu(如ARM)中，可以直接使用来自页面映射接口的信息找到逐出集

静态/动态的大页面：

依靠2MB的大页面来绕过虚拟地址映射到缓存集的问题。

F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, “Last-Level Cache Side-Channel Attacks Are Practical,” in Proceedings of the 2015 IEEE Symposium on Security and Privacy, SP ’15, (Washington, DC, USA), pp. 605–622, IEEE Computer Society, 2015.
G. Irazoqui, T. Eisenbarth, and B. Sunar, “S$A: A Shared Cache Attack That Works Across Cores and Defies VM Sandboxing – and Its Application to AES,” in Proceedings of the 2015 IEEE Symposium on Security and Privacy, SP ’15, (Washington, DC, USA), pp. 591–604, IEEE Computer Society, 2015.

Goal (Cont.)

系统地从理论和实践证明两方面研究寻找最小驱逐集的新方法

执行了第一个来自JavaScript的缓存攻击，其中使用常规4KB的页面，指针不能直接使用。它利用浏览器对大缓冲区的页面对齐分配的知识，构造具有相同页面偏移位的初始集。

] Y. Oren, V. P. Kemerlis, S. Sethumadhavan, and A. D. Keromytis, “The Spy in the Sandbox: Practical Cache Attacks in JavaScript and Their Implications,” in CCS, ACM, 2015.

目标是Windows 10上的Microsoft Edge，不能依赖于大页面，但是他们发现Windows内核为来自不同物理内存池的分配页面，这些物理内存池通常属于相同的缓存集。因此能够通过访问几个相距128KB的地址(通常位于同一个缓存集中)有效地找到逐出集(不是最小的)。

E. Bosman, K. Razavi, H. Bos, and C. Giuffrida, “Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector,” in IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016, pp. 987–1004, 2016.

多次迭代Test 3来找到小的逐出集，并丢弃所有始终热门的元素(即始终产生缓存命中)，从而打破虚拟机隔离。

J. Horn, “Reading privileged memory with a side-channel.” https://googleprojectzero.blogspot.com.es/2018/01/reading-privileged-memory-with side.html, 2018.

Goal (Cont.)

系统地从理论和实践证明两方面研究寻找最小驱逐集的新方法

Slicing函数的逆向工程：
具有LLC切片功能的现代cpu 6使用专有的散列函数将块分布到切片中，这导致了对它们进行逆向工程的尝试。

抗抖动/扫描替换策略：
替换策略，如插入策略或DRRIP，已知在导致扫描或抖动的工作负载方面比PLRU性能更好。然而，它们也使驱逐不那么可靠，并且超出了我们当前的模型。

集合索引随机化:
并发工作为随机化缓存提出了一些新的设计，其中缓存集用键控函数进行索引，完全使攻击者无法控制物理地址位。

"L3 cache mapping on Sandy Bridge CPUs." http://lackingrhoticity.blogspot.com.es/2015/04/l3-cache-mapping-on-sandy-bridge-cpus.html, 2015.

D. Trilla, C. Hernández, J. Abella, and F. J. Cazorla,"Cache side channel attacks and time predictability in high performance critical real time systems," in Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29, 2018, pp. 98:1–98:6, 2018.
M. K. Qureshi, "CEASER: Mitigating Conflict-Based Cache Attacks via Encrypted-Address and Remapping," in IEEE/ACM International Symposium on Microarchitecture - MICRO 2018

Key Approach & Ideas

Finding minimal eviction sets

1

Find a large eviction set for an address V:

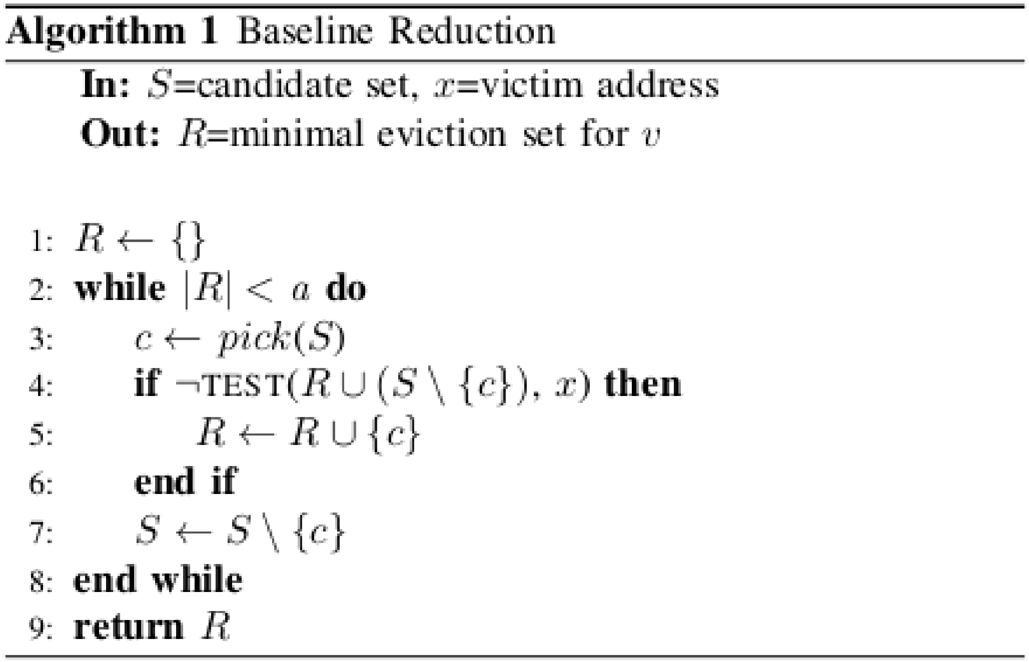
- Pick "enough" addresses at random
- Timing test: $a_v\ a_0\ a_1\ \dots\ a_n\ \boxed{a_v}$

2

Reduce initial large eviction set into its minimal core

Mechanisms & Implementation

老方法 Baseline algorithm



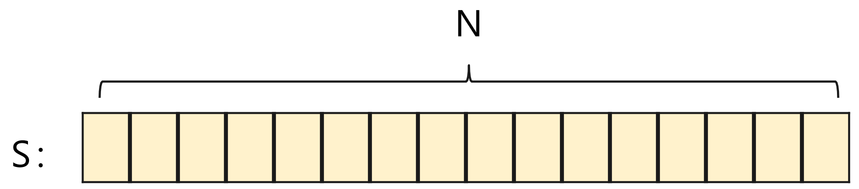
Algorithm 1 Baseline Reduction

```
    In: $S$=candidate set, $x$=victim address
    Out: $R$=minimal eviction set for $v$

1: $R \leftarrow \{\}$
2: while $|R| < a$ do
3:     $c \leftarrow pick(S)$
4:     if $\neg$TEST$(R \cup (S \setminus \{c\}), x)$ then
5:         $R \leftarrow R \cup \{c\}$
6:     end if
7:     $S \leftarrow S \setminus \{c\}$
8: end while
9: return $R$
```

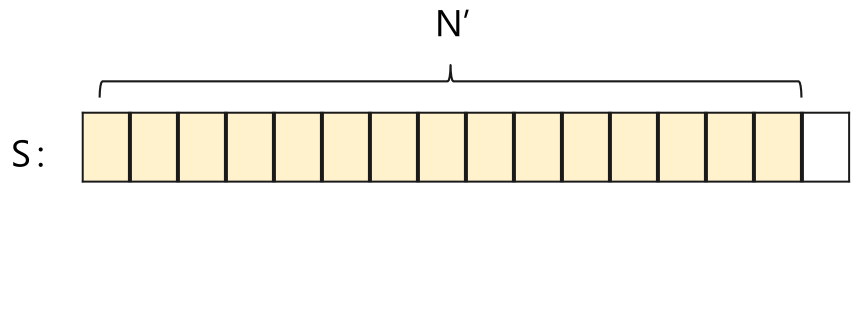
算法接收一个任意地址x和地址集合S作为参数输入，S可以将地址x从L3 Cache中驱逐但不是最小驱逐集。首先从S中取出任意地址c，测试地址集S-{c}是否仍能将x从缓存中刷新掉，如果S-{c}不能刷新x则表示地址c和x处于同一个slice，那么便将c记录到集合R中，如果S-{c}仍能刷新x则表示c是多余的，直接丢弃掉！需要注意的是测试地址集S-{c}是否仍能将x从缓存中刷新掉时（第4行）需要将集合R并入测试地址的集合，因为R中都是有效的地址。如此不断循环，直到R的元素个数等于L3 Cache的关联度a，表示此时从S中得到了最小驱逐集。

老方法 Baseline algorithm (Cont.)



Start with large enough eviction set S of size N

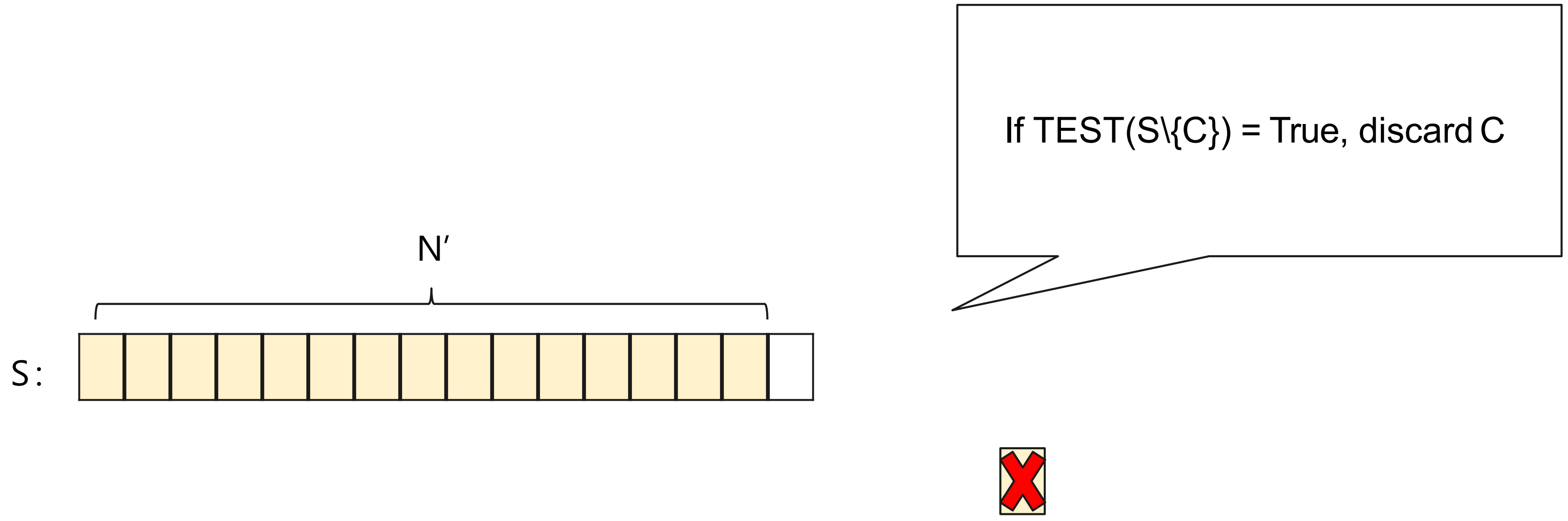
老方法 Baseline algorithm (Cont.)



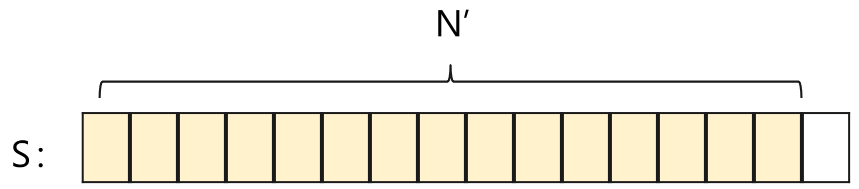
Pick candidate element C, and

Test if remaining set TEST(S\{C}) is still an eviction set

老方法 Baseline algorithm (Cont.)

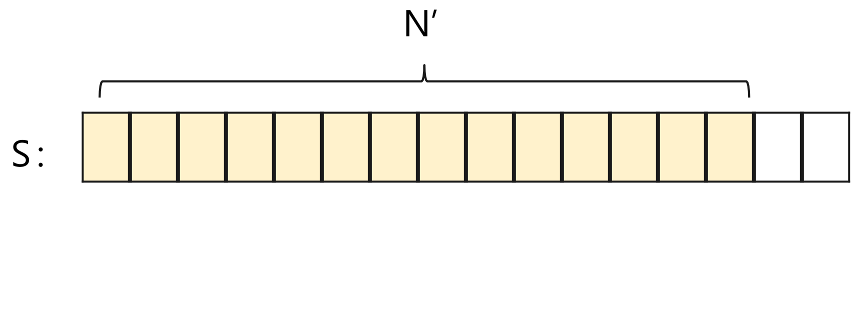


老方法 Baseline algorithm (Cont.)



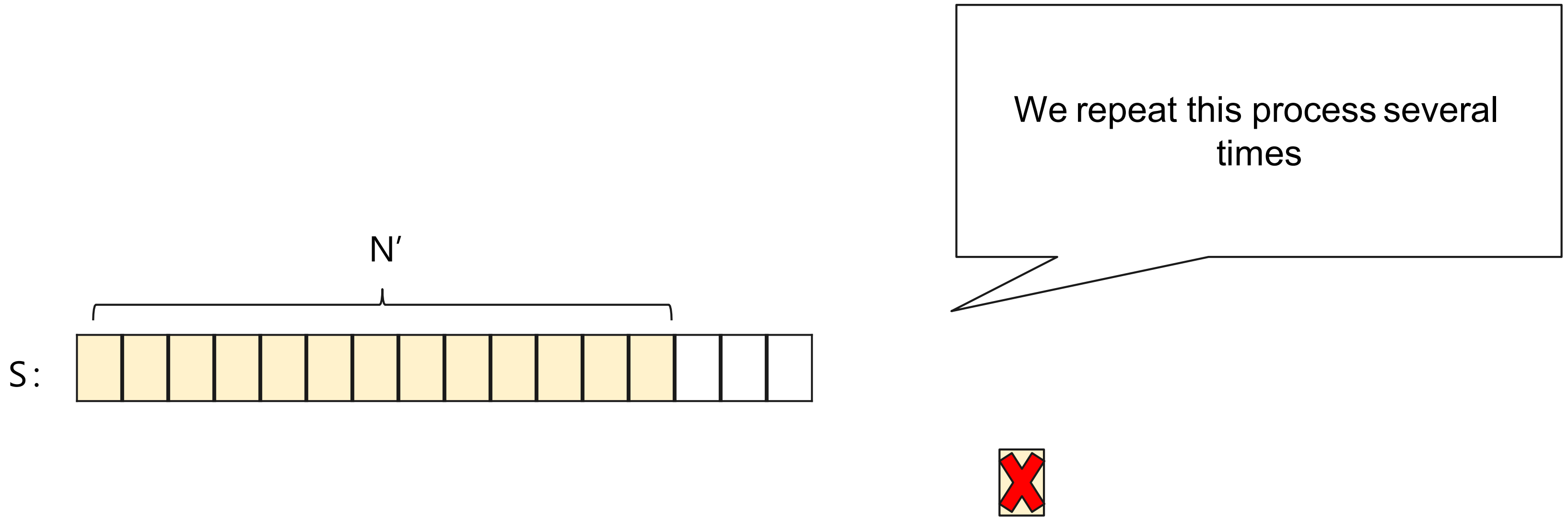
and continue with N’=N-1

老方法 Baseline algorithm (Cont.)

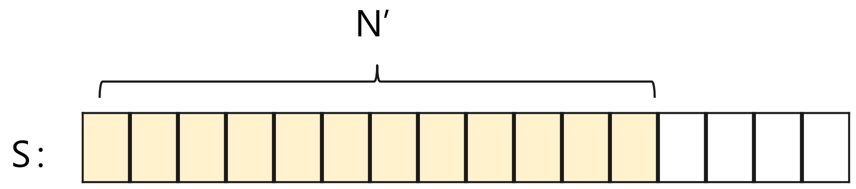


We repeat this process several times

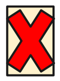
老方法 Baseline algorithm (Cont.)



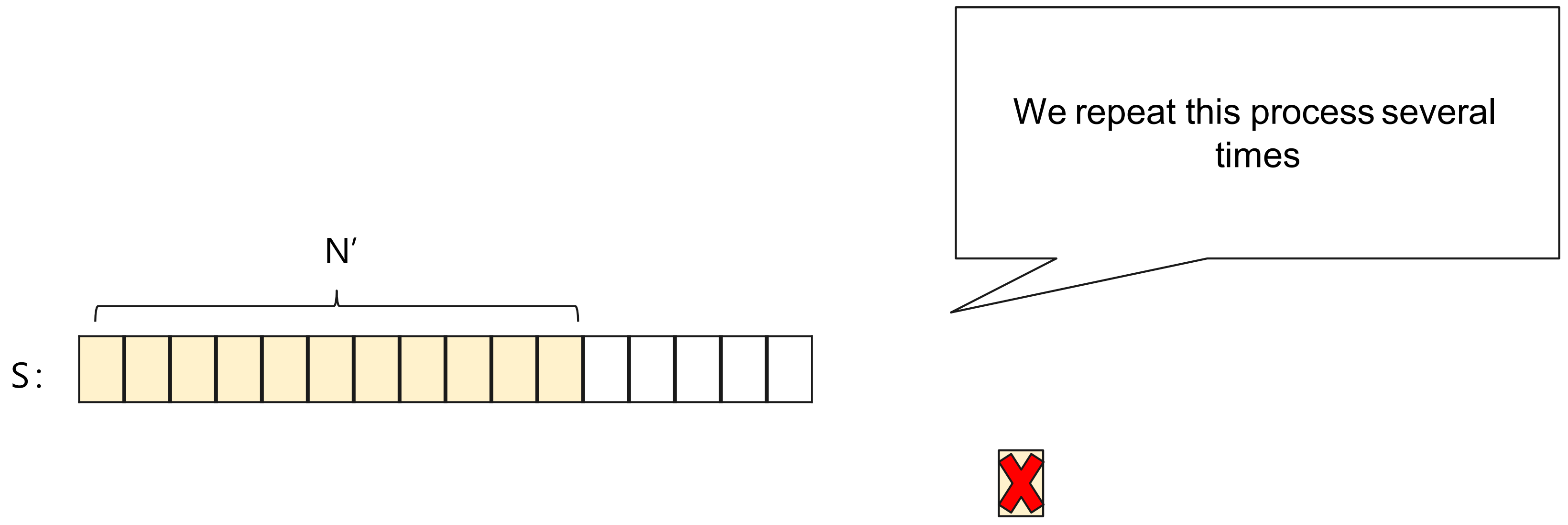
老方法 Baseline algorithm (Cont.)



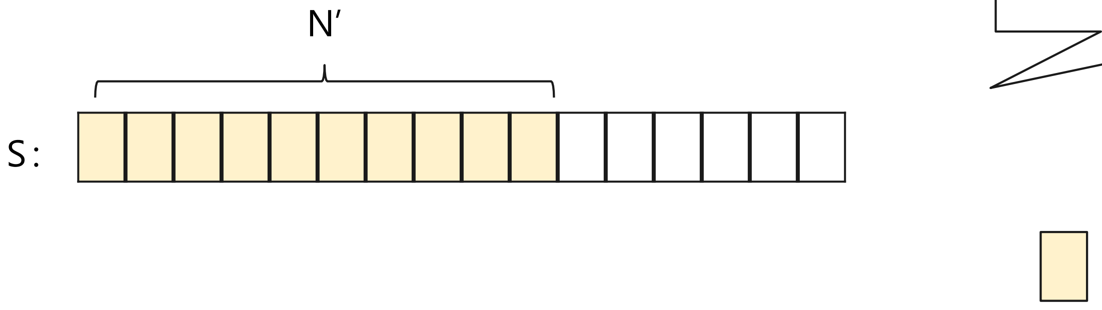
We repeat this process several times



老方法 Baseline algorithm (Cont.)



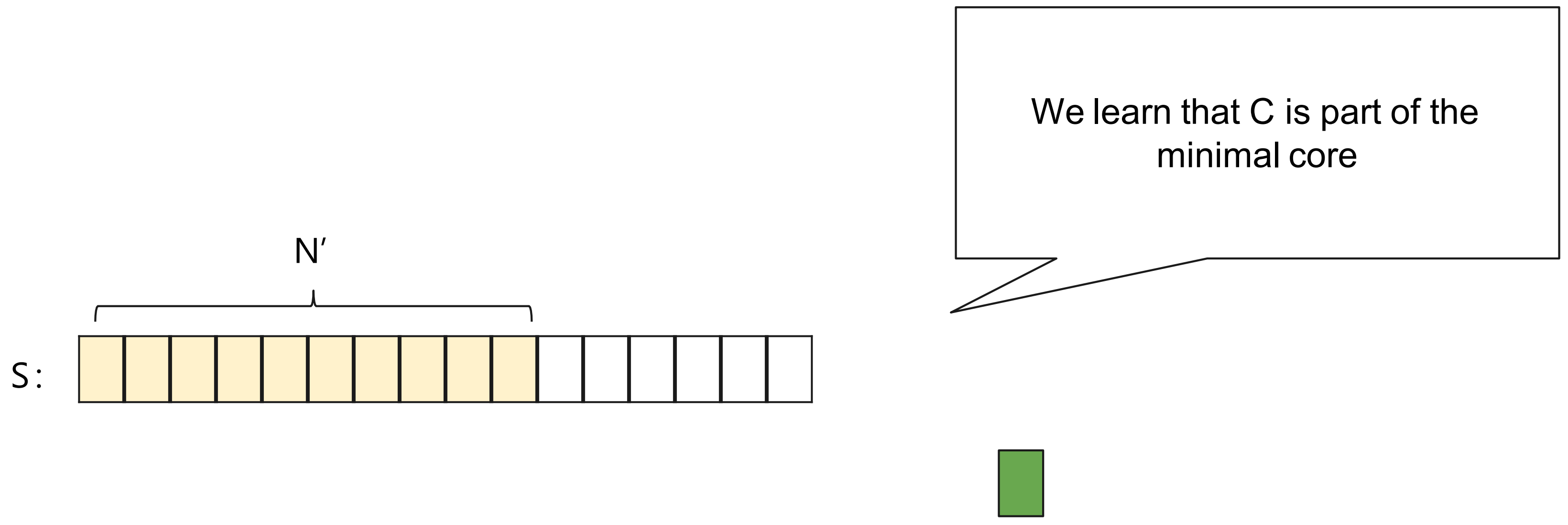
老方法 Baseline algorithm (Cont.)



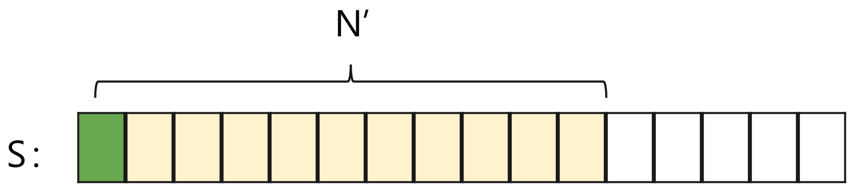
Until we find an element C such that when removed the remaining set stops being an eviction set:

TEST(S\{C}) = False

老方法 Baseline algorithm (Cont.)

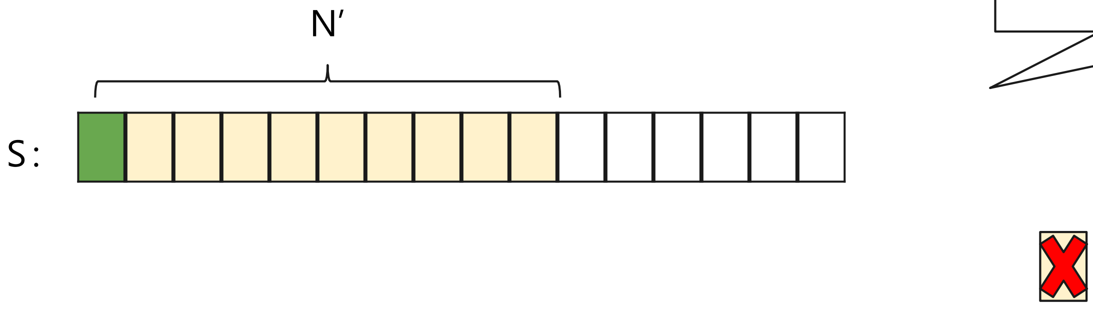


老方法 Baseline algorithm (Cont.)



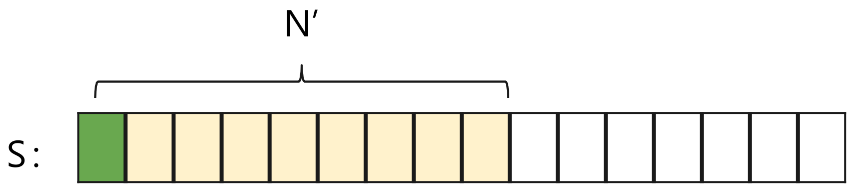
We keep track of it, and insert it again in S

老方法 Baseline algorithm (Cont.)

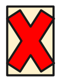


We repeat this process several times

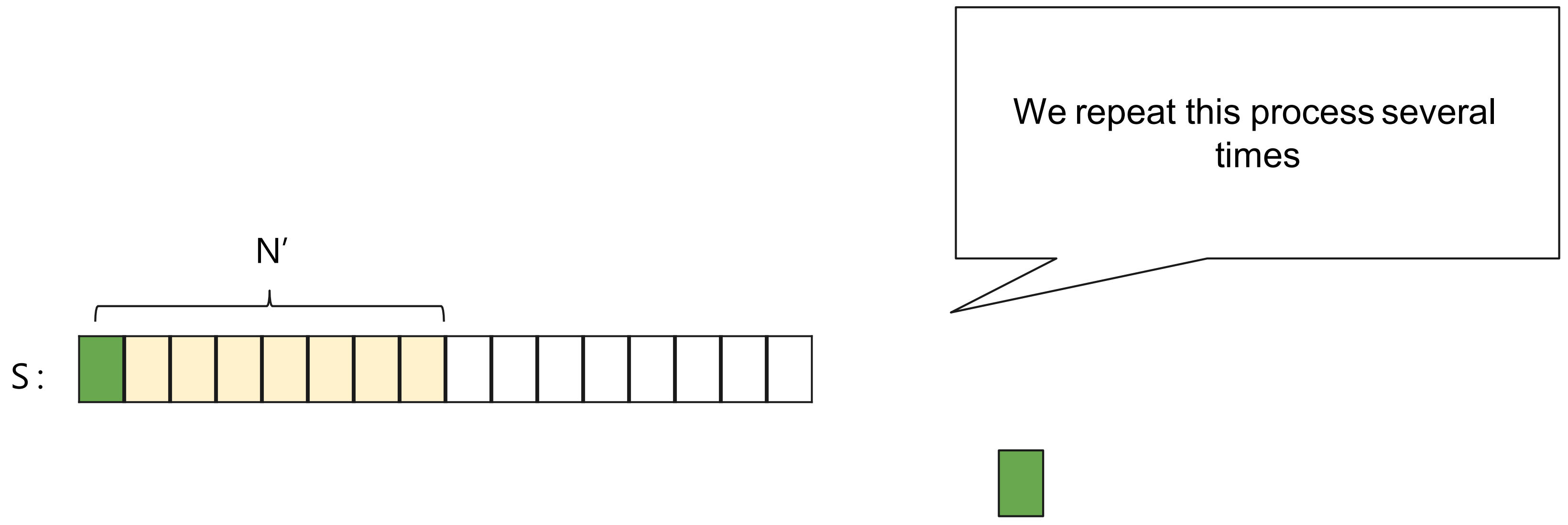
老方法 Baseline algorithm (Cont.)



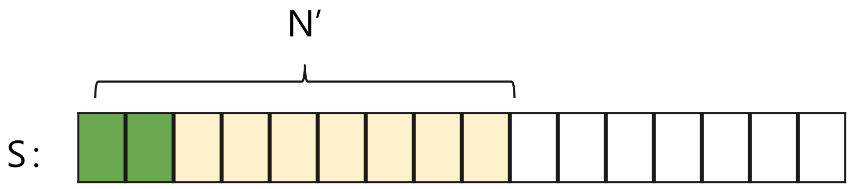
We repeat this process several times



老方法 Baseline algorithm (Cont.)

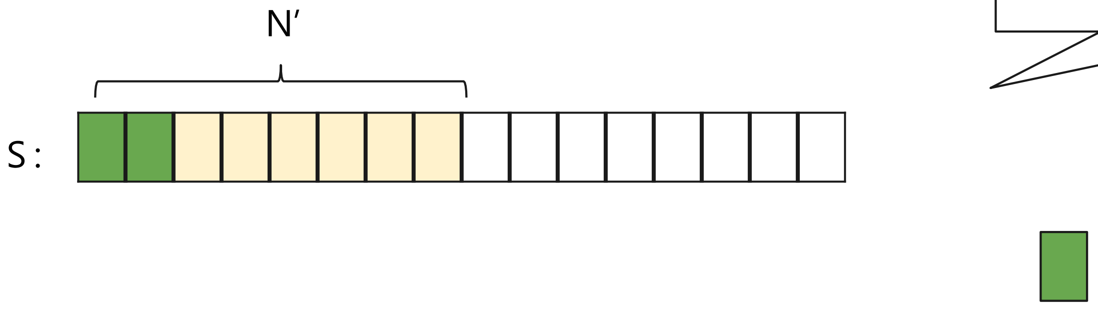


老方法 Baseline algorithm (Cont.)



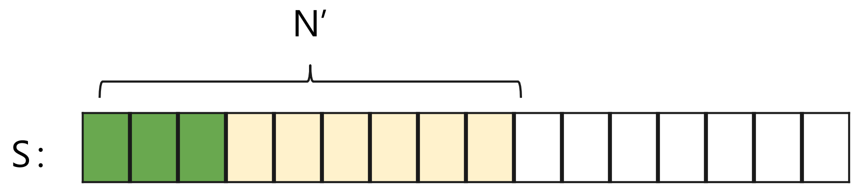
We repeat this process several times

老方法 Baseline algorithm (Cont.)



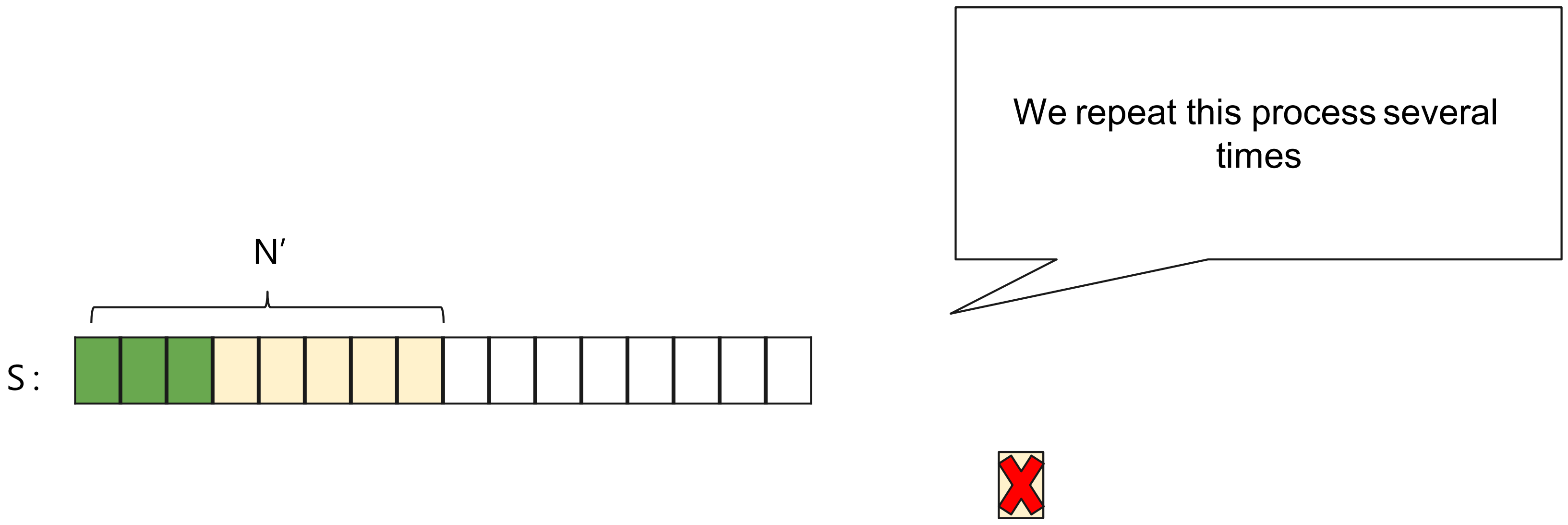
We repeat this process several times

老方法 Baseline algorithm (Cont.)

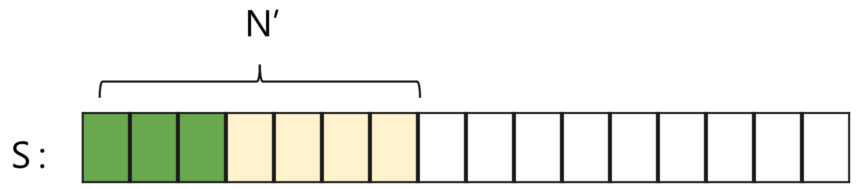


We repeat this process several times

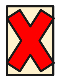
老方法 Baseline algorithm (Cont.)



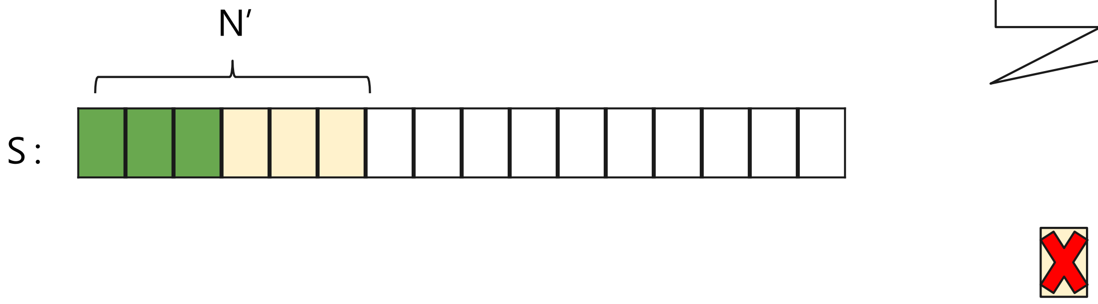
老方法 Baseline algorithm (Cont.)



We repeat this process several times

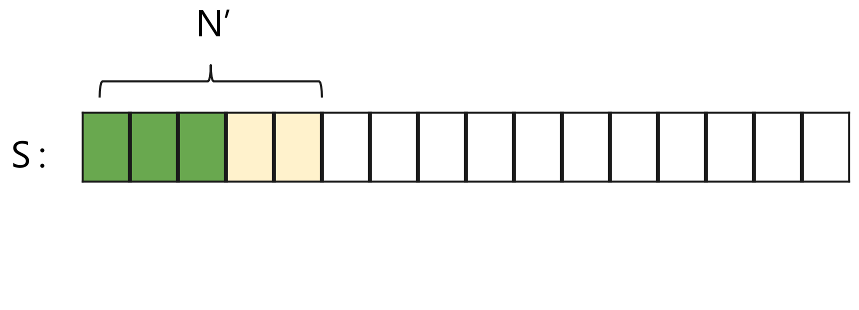


老方法 Baseline algorithm (Cont.)



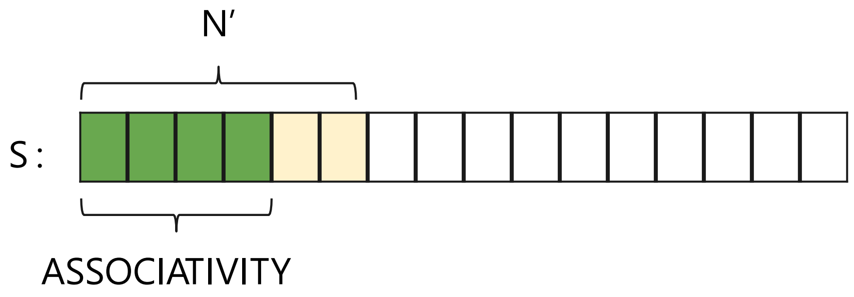
We repeat this process several times

老方法 Baseline algorithm (Cont.)



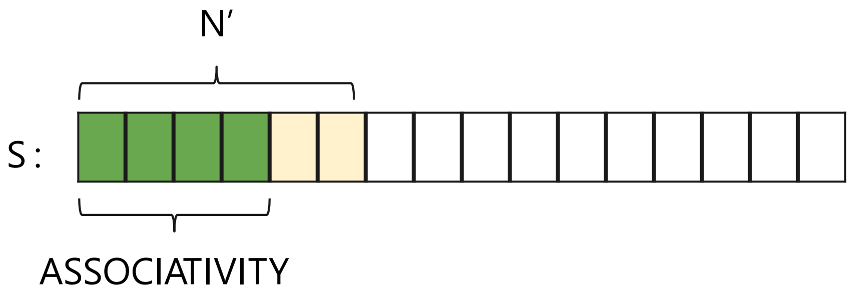
We repeat this process several times

老方法 Baseline algorithm (Cont.)



Until we have identified ASSOCIATIVITY many elements representing the eviction set's core!

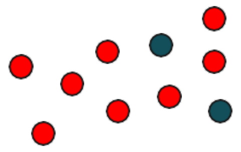
老方法 Baseline algorithm (Cont.)



$\boxed{O(N^2)\text{ memory accesses}}$

Threshold Group Testing

Group testing problem by Robert Dorfman (1943)

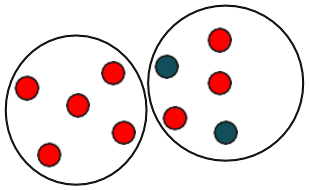


Blood samples

(10 individual tests)

Threshold Group Testing (Cont.)

Group testing problem by Robert Dorfman (1943)



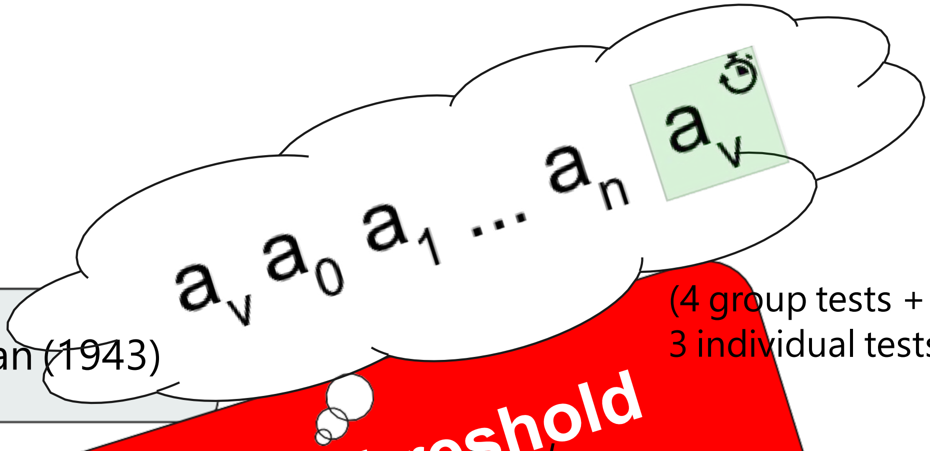
(4 group tests +
3 individual tests)

Blood samples

Generalization by Peter Damaschke (2006):

- Positive test only if at least "u" defectives
- Negative test only if at most "l" defectives
- Random otherwise

Threshold Group Testing (Cont.)



(4 group tests +
3 individual tests)

Group testing problem by Robert Dorfman (1943)

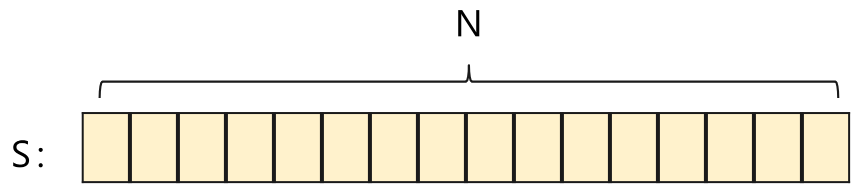
Blood samples

Generalization by Peter Damaschke (2006):

- Positive test if at least "u" defectives
- Negative test if at most "l" defectives
- Random answer otherwise

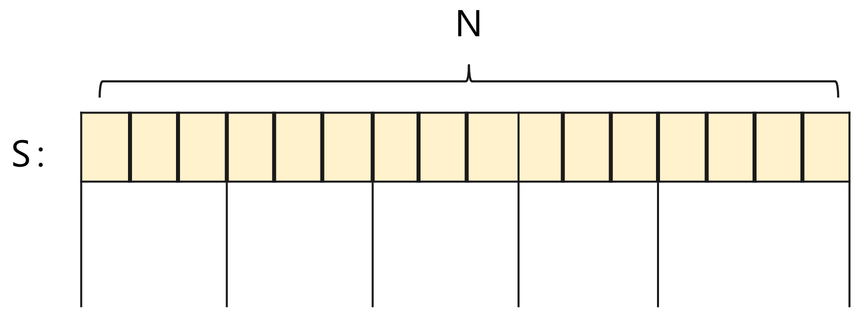
Observation: Our test is a threshold group test!

Group-testing algorithm



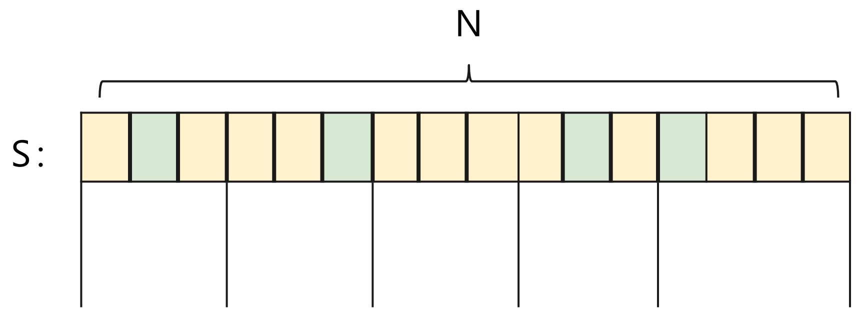
Start with large enough eviction set S of size N

Group-testing algorithm (Cont.)



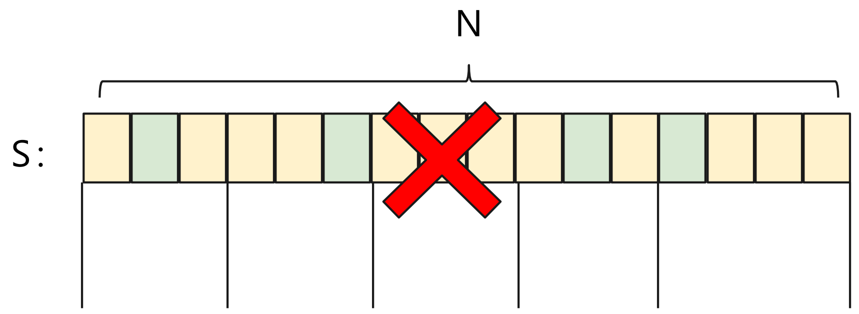
Split S in ASSOCIATIVITY+1 subsets

Group-testing algorithm (Cont.)



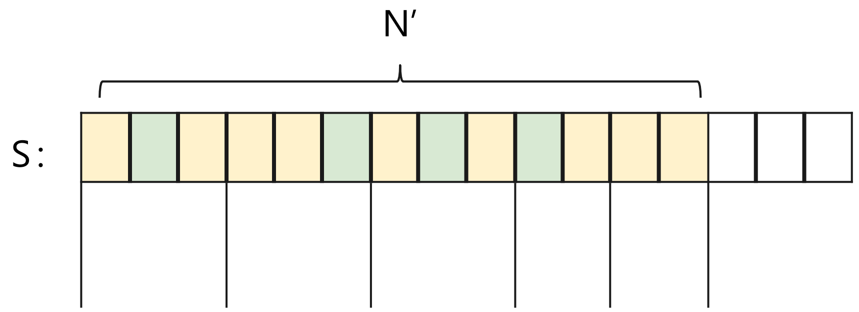
In the worst case, there exists a union of ASSOCIATIVITY subsets being an eviction set

Group-testing algorithm (Cont.)



We can discard N/(ASSOCIATIVITY+1) elements per iteration

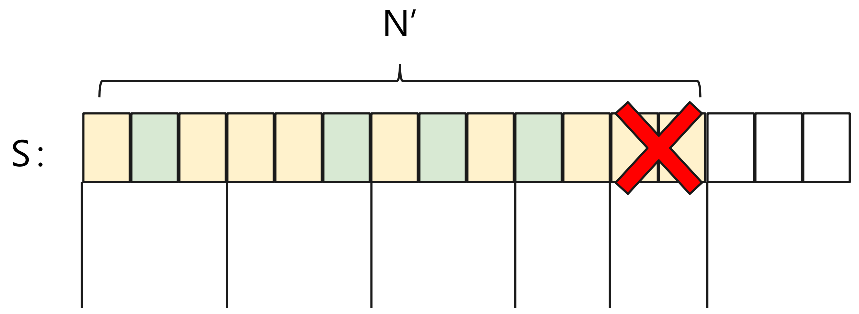
Group-testing algorithm (Cont.)



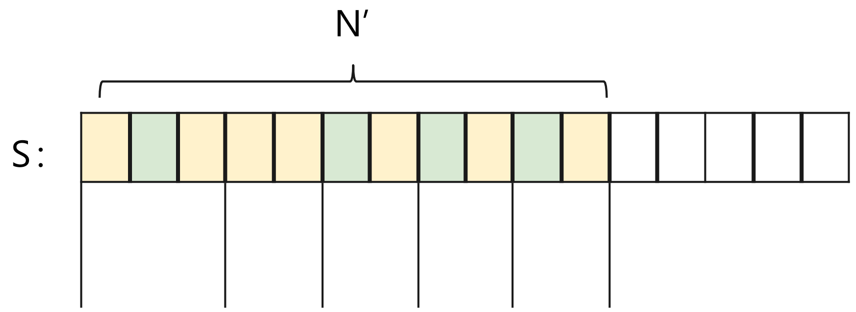
We repeat this process until we have ASSOCIATIVITY many elements

Group-testing algorithm (Cont.)

We repeat this process until we have ASSOCIATIVITY many elements

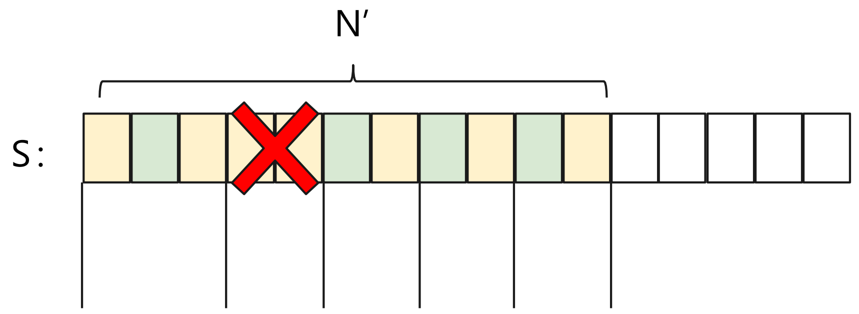


Group-testing algorithm (Cont.)



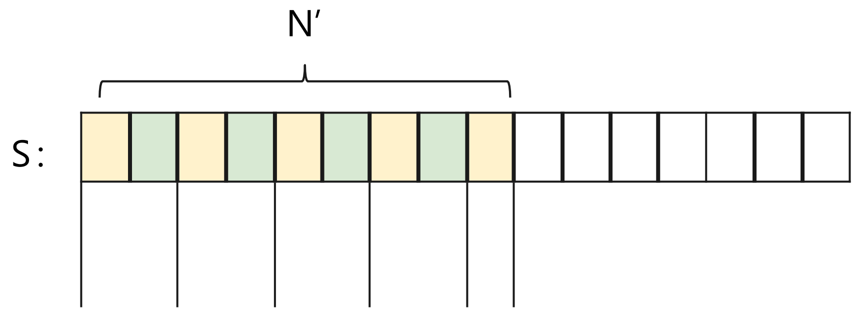
We repeat this process until we have ASSOCIATIVITY many elements

Group-testing algorithm (Cont.)



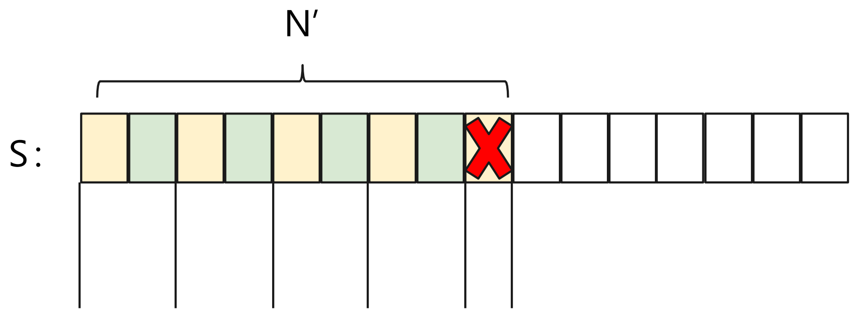
We repeat this process until we have ASSOCIATIVITY many elements

Group-testing algorithm (Cont.)



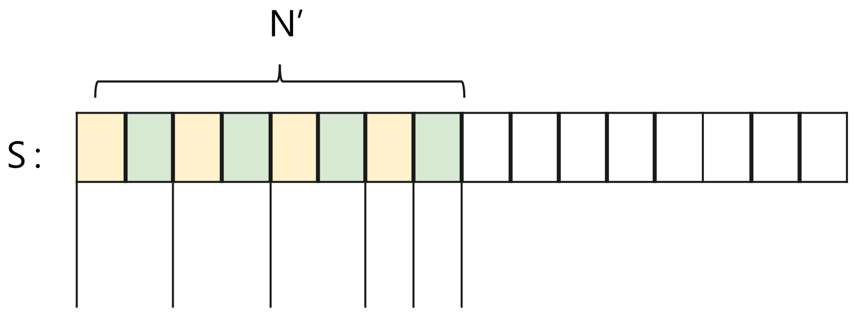
We repeat this process until we have ASSOCIATIVITY many elements

Group-testing algorithm (Cont.)



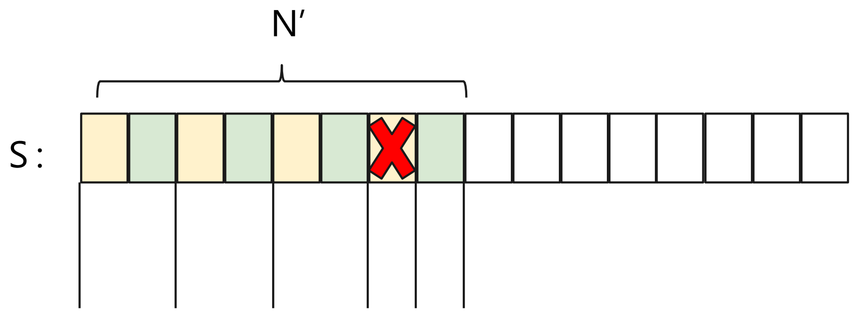
We repeat this process until we have ASSOCIATIVITY many elements

Group-testing algorithm (Cont.)



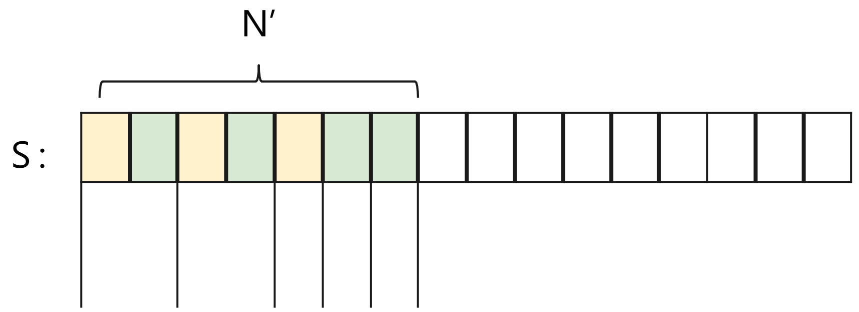
We repeat this process until we have ASSOCIATIVITY many elements

Group-testing algorithm (Cont.)



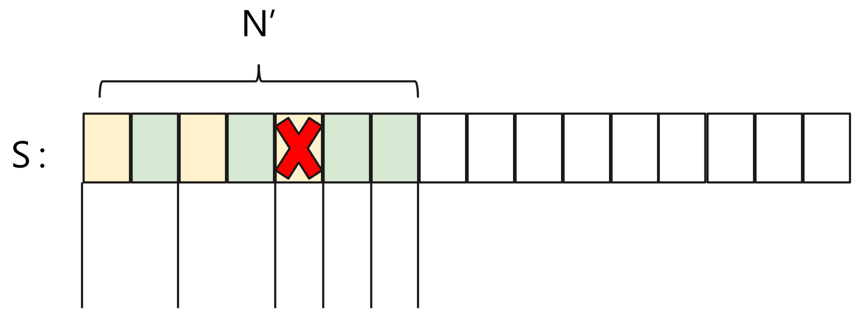
We repeat this process until we have ASSOCIATIVITY many elements

Group-testing algorithm (Cont.)



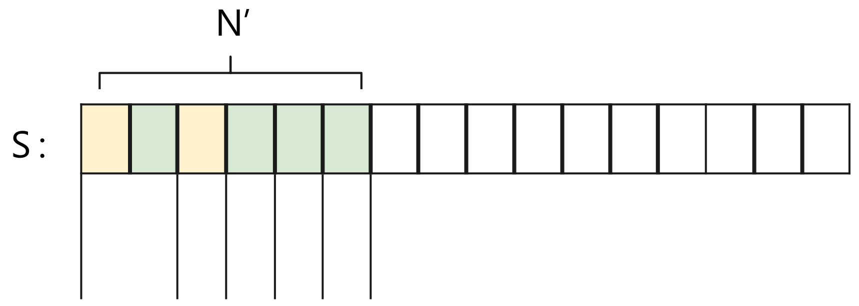
We repeat this process until we have ASSOCIATIVITY many elements

Group-testing algorithm (Cont.)



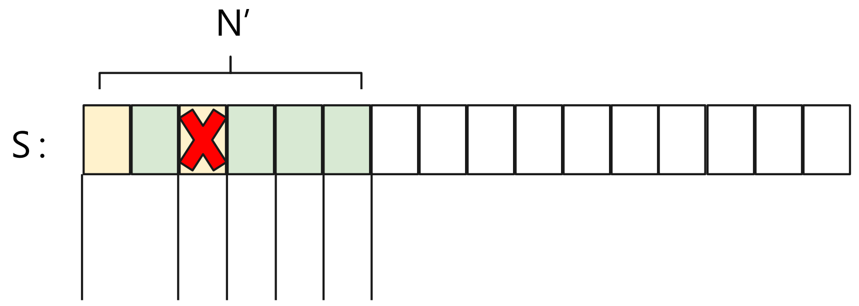
We repeat this process until we have ASSOCIATIVITY many elements

Group-testing algorithm (Cont.)



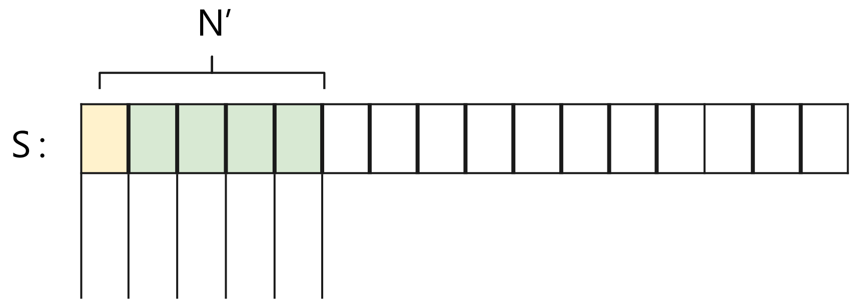
We repeat this process until we have ASSOCIATIVITY many elements

Group-testing algorithm (Cont.)



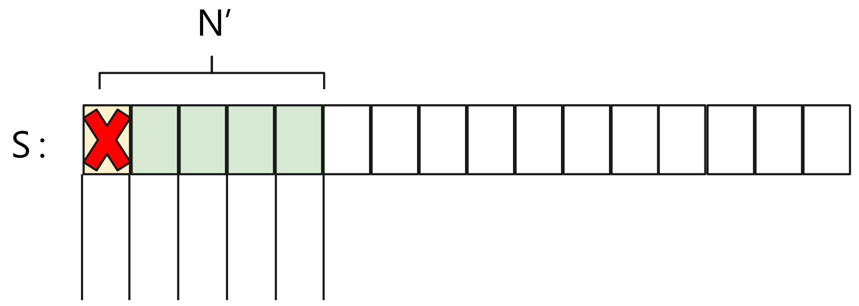
We repeat this process until we have ASSOCIATIVITY many elements

Group-testing algorithm (Cont.)



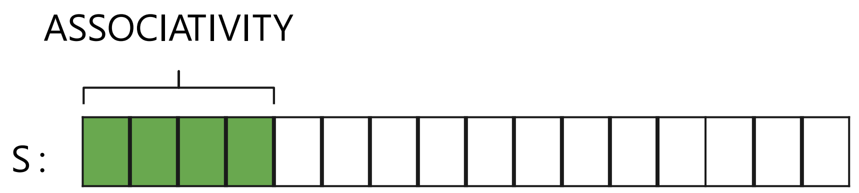
We repeat this process until we have ASSOCIATIVITY many elements

Group-testing algorithm (Cont.)



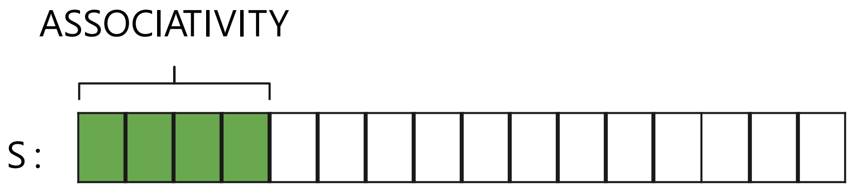
We repeat this process until we have ASSOCIATIVITY many elements

Group-testing algorithm (Cont.)



We find our minimal eviction set!

Group-testing algorithm (Cont.)



O(N) mem accesses

Novelty

创新要素

- 站在巨人的肩膀上
- 分而治之
- 数学上的可证明性
- 理论和实践双全
- 跨学科思维

Results & Evaluation

Performance Evaluation

分析平台:我们在运行Linux 4.9的两个不同cpu上评估了我们的算法。
（1）英特尔i5-6500 4 × 3.20 GHz (Skylake)，16gb RAM, 6mb LLC和8192 12路缓存集。
（2）英特尔i7-4790 8 x 3.60GHz GHz (Haswell)，8gb的RAM, 和8mb LLC与8192 16路缓存集。

鲁棒性评估
驱逐率：我们的测试在随机选择的固定大小的集合上返回真值的相对频率。
减少率：我们成功地将随机选择的固定大小的集合减少到最小驱逐集的相对频率。

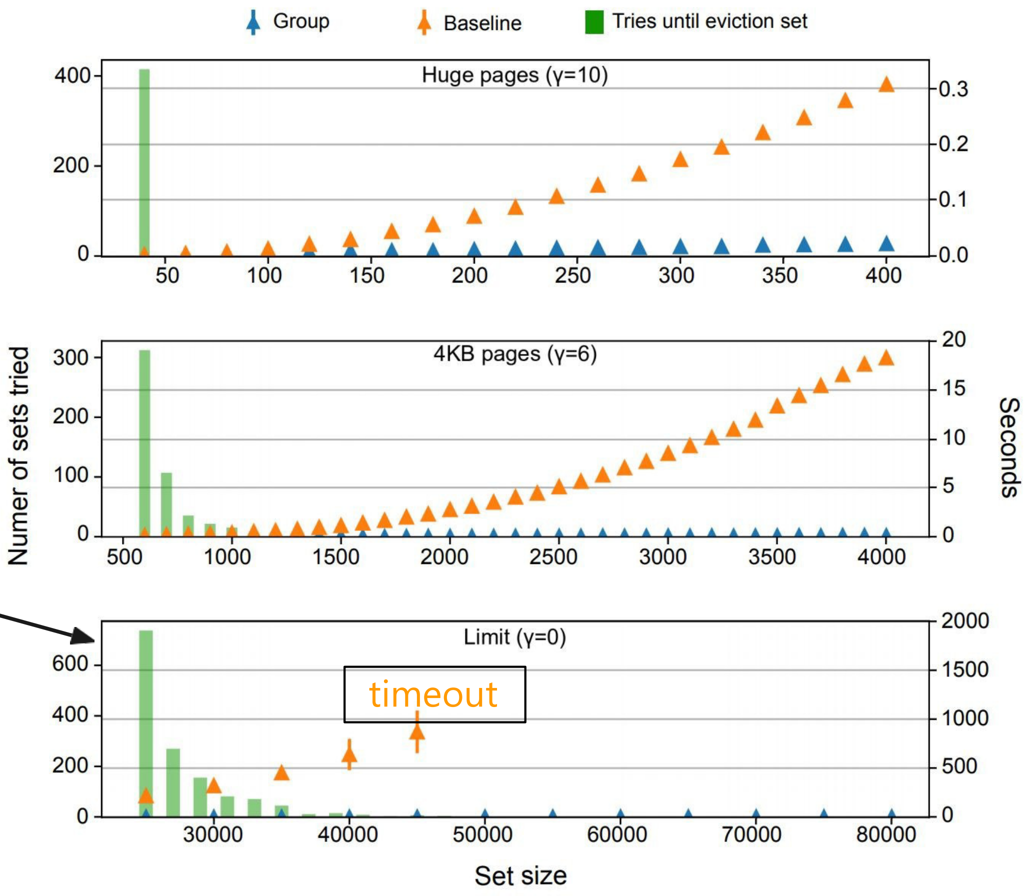
Performance Evaluation

Experiments on Skylake i5-6500 with 6MB cache (8192 sets x 12 assoc)

Tool (C/x86):
https://github.com/cgvwzq/evsets

O(n) vs. O(n^2) advantage shows up in practice!

Finding minimal eviction sets is practical without knowledge on any bits of the set index!

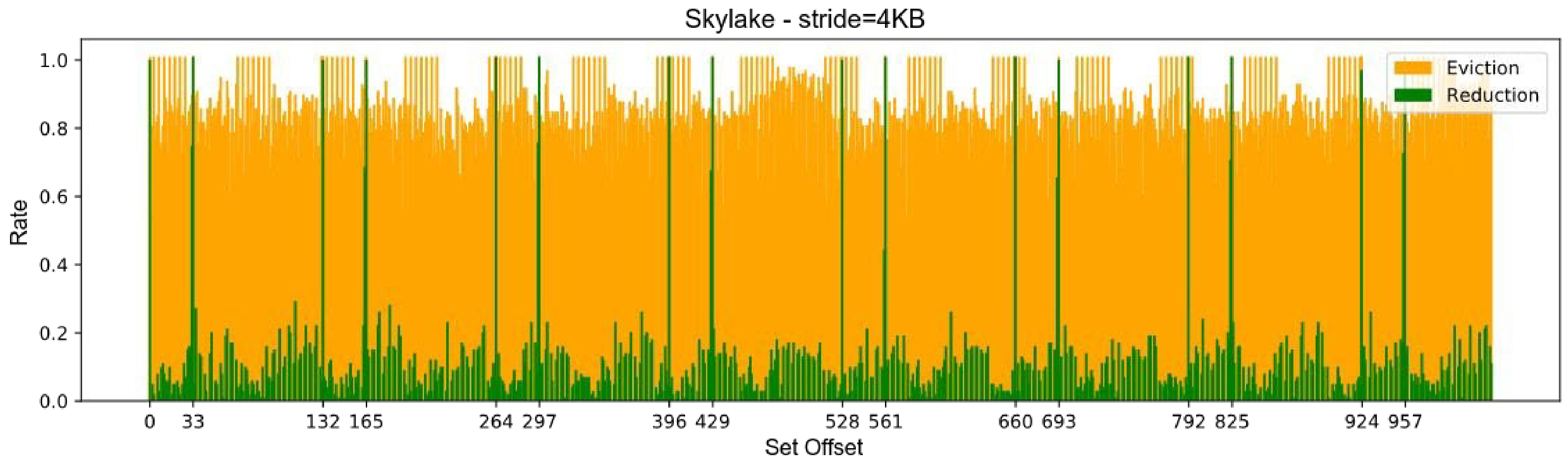


Y-right (lines): Average running time for eviction set reduction
Y-left (columns): Cost of finding an initial eviction set of certain size
X: Eviction set size in number of addresses

Robustness Evaluation

Experiments on Skylake i5-6500 with 6MB cache (8192 sets x 12 assoc)

Modern replacement policies break our test assumption and introduce errors.



X: Cache set offset (each points aggregates all slices)
Y: Average success rate for
Green: reduction rate w/o error correcting mechanisms.

Yellow: Test rate reliability

Strengths & Weaknesses

Strength

- Systematic study of the problem of finding eviction sets
- Find eviction sets in O(n) compared to previous O(n2)
- Reliability and performance evaluation of algorithms in real hardware
- Simple, novel mechanism to solve an important problem
- Effective and low hardware overhead
- 证明详实，工作充分

Weakness

- 技术上依然效仿先前
- 未来利用并行、并发、多线程优化、解决老问题

Discussion

Group Testing against Covid-19

为了缓解COVID-19全球大流行，其中最重要的关切就是检测。

2020年7月，当第三波Covid-19疫情袭击香港时，有些专家建议需要将检测能力提高到每天2万个测试，然而彼时的香港每天的测试量只能支持4,000至5,000个检测。为了在相同数量资源的条件下提高使用效率，我们考虑使用群组检测方法。在这样的过程中，样本按群组进行测试。如果某测试群组中的所有样本均为阴性，则检测结果归于阴性。如果群组中至少有一个样本呈阳性，则该群组证明为阳性。

组分得较大意味着结果归于阳性的机会更大，这就需要做进一步的检测。

另一方面，群组分得较细虽可减小进一步细分群组和重新检测的可能性，

但是群组较小又会增加所需测试的数量。

<u>作出方案决策时还有另一个必须考虑的因素：测试结果有可能是错的。</u>

Source: IMMC 2021中华赛A题（秋季赛）

差不多的改进工作

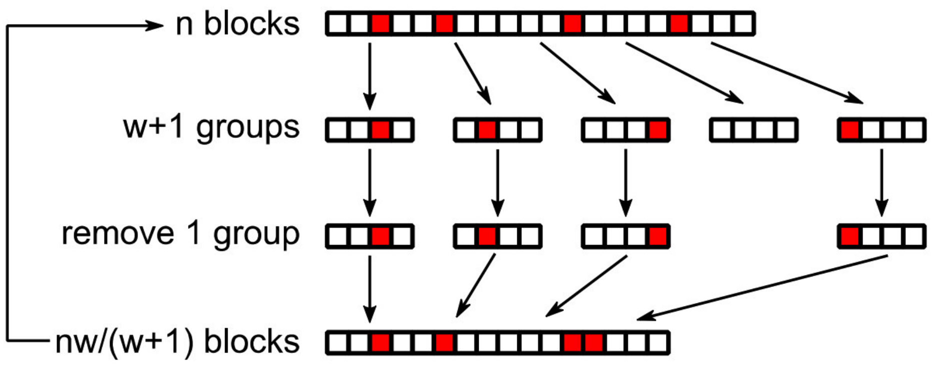
快速驱逐集寻找算法

Algorithm 2: Prune with split: prune_split(C, x, w, l)

Input: C, candidate set; x, target address; w, number of ways; l, split parameter.

Output: Minimal eviction set for x.

```
1  function prune_split(C,x,w,l)
2      while |C| > w do
3          {G_1,...,G_l} ← split(C,l)
4          foreach G in {G_1,...,G_l} do
5              if test(C \ G, x) then
6                  C ← C \ G
7              end
8          end
9      end
10     return C
11 end
```



使用快速的驱逐集寻找算法将时间复杂度从O(N^2)降低到了O(wN)。

P. Vila, B. Köpf, J. F. Morales. Theory and Practice of Finding Eviction Sets. S&P'19, 39-54.
W. Song, P. Liu. Dynamically finding minimal eviction sets can be quicker than you think for side-channel attacks against the LLC. RAID'19, 427-442.

Problem：

■ **寻找小驱逐集是microarchitectural攻击的核心步骤**
  ▶现在的方法效率低、且零散、不成系统体系

Key Idea：

■ **群组检测**

Mechanisms ：

■ **先分组，分组数维持恒定，再观察每组里是不是去掉它剩下的依然是驱逐集，如果是的话，就把这个组去掉。**

Results ：

■ **Find eviction sets in O(n) compared to previous O(n^2)**
■ **严格的实证评估证明**

感谢批评指正
THANKS