Theory and Practice of Finding Eviction Sets

Pepe Vila
IMDEA Software Institute

Boris Köpf
Microsoft Research

José F. Morales
IMDEA Software Institute

institute IMDEA software

Microsoft

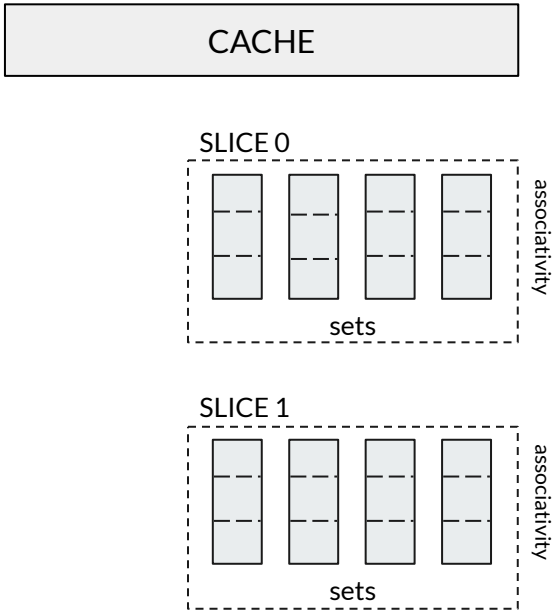
vwzq.net

@cgvwzq

github.com/cgvwzq

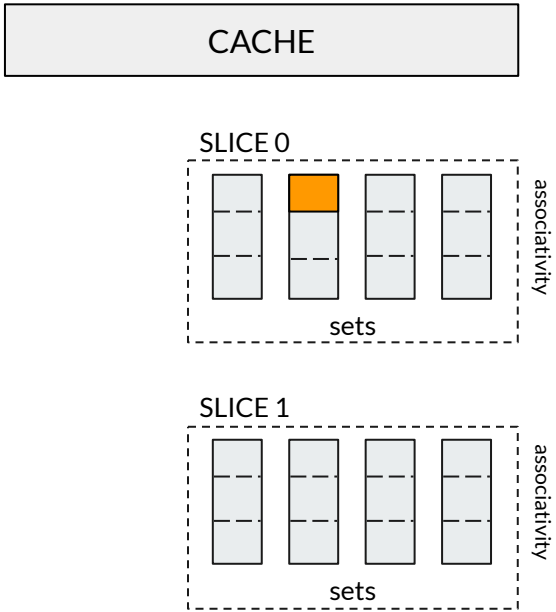
Eviction Sets

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



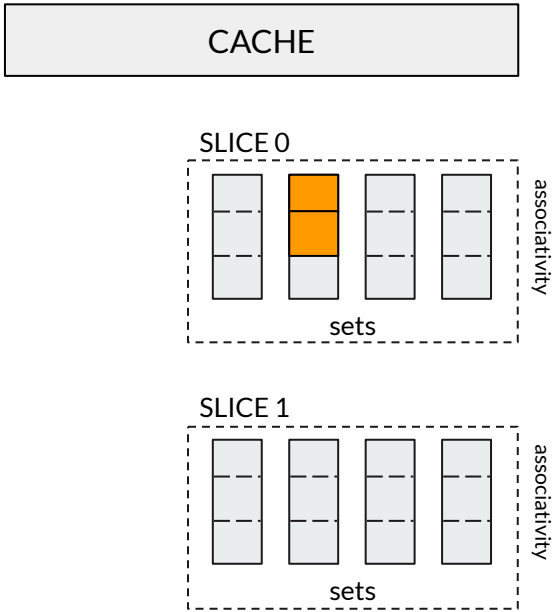
Eviction Sets

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



Eviction Sets

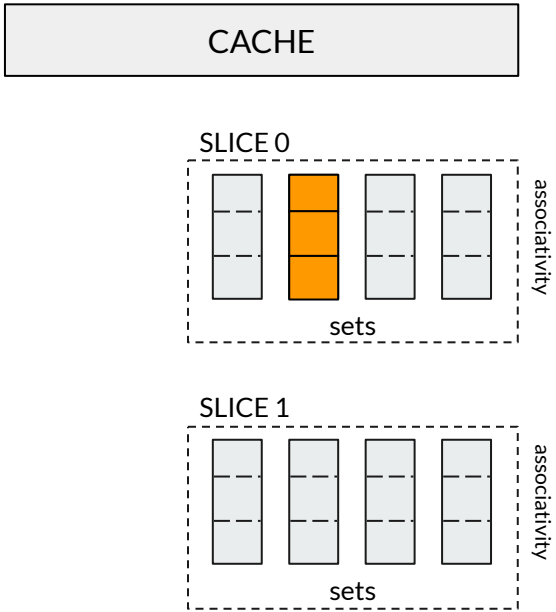
Find addresses that collide in cache: i.e. addresses mapped into the same cache set



Eviction Sets

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**



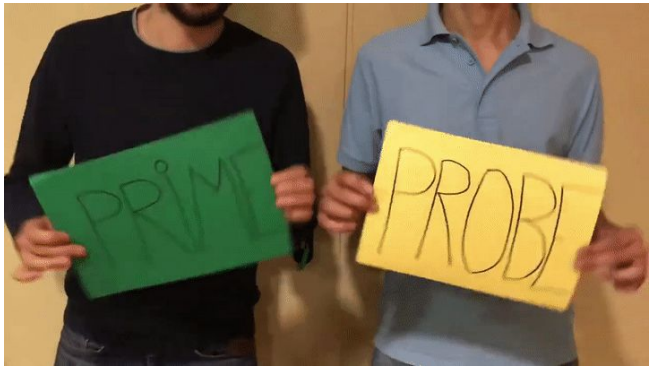
Attacks

Efficient attacks require small eviction sets

Attacks

Efficient attacks require small eviction sets

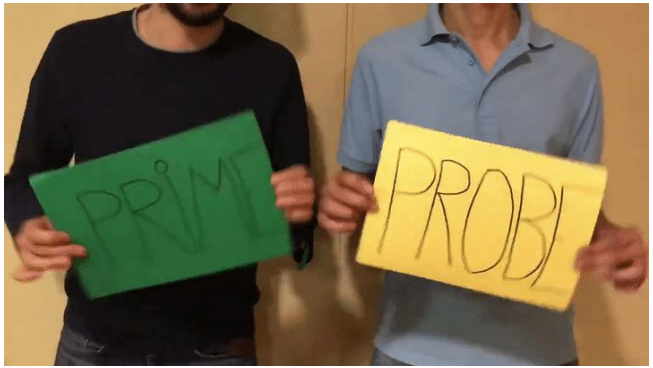
Prime+Probe



Attacks

Efficient attacks require small eviction sets

Prime+Probe

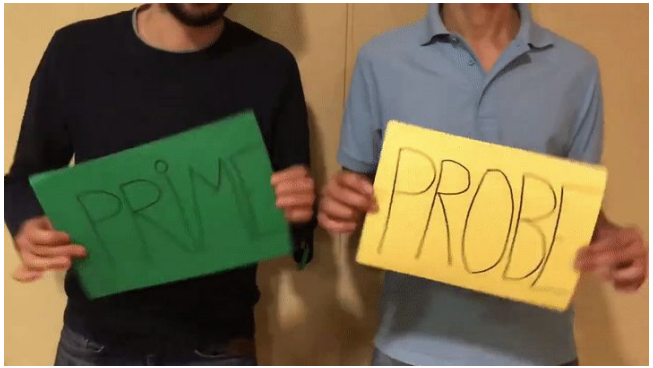


Rowhammer

Attacks

Efficient attacks require small eviction sets

Prime+Probe

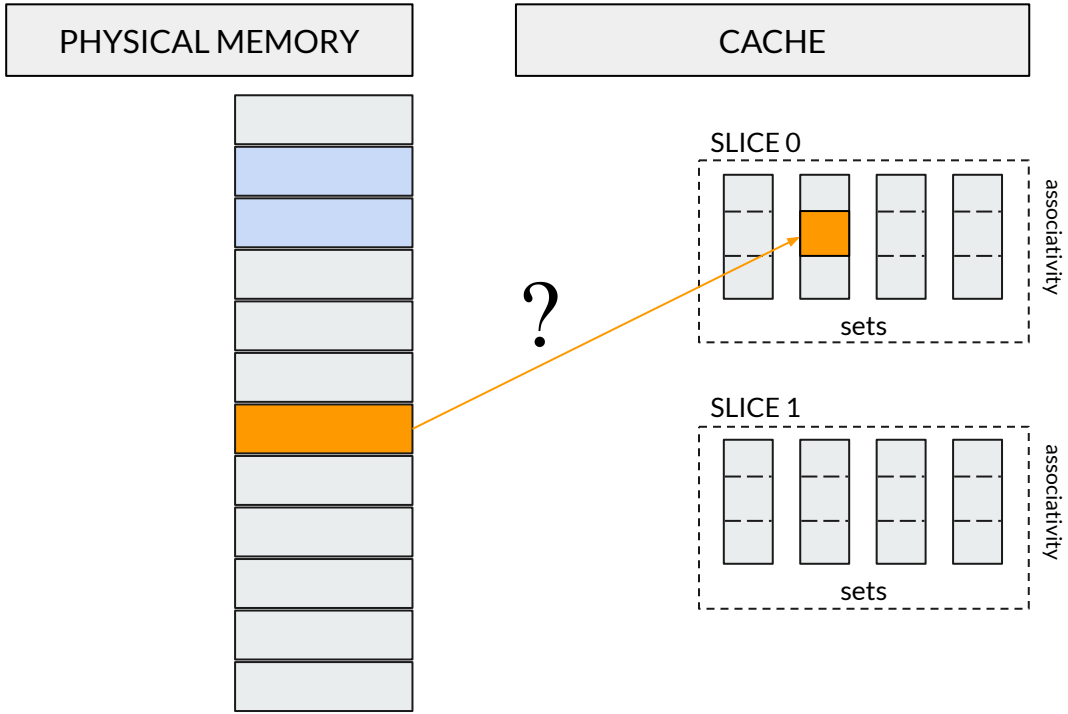


Rowhammer

Spectre

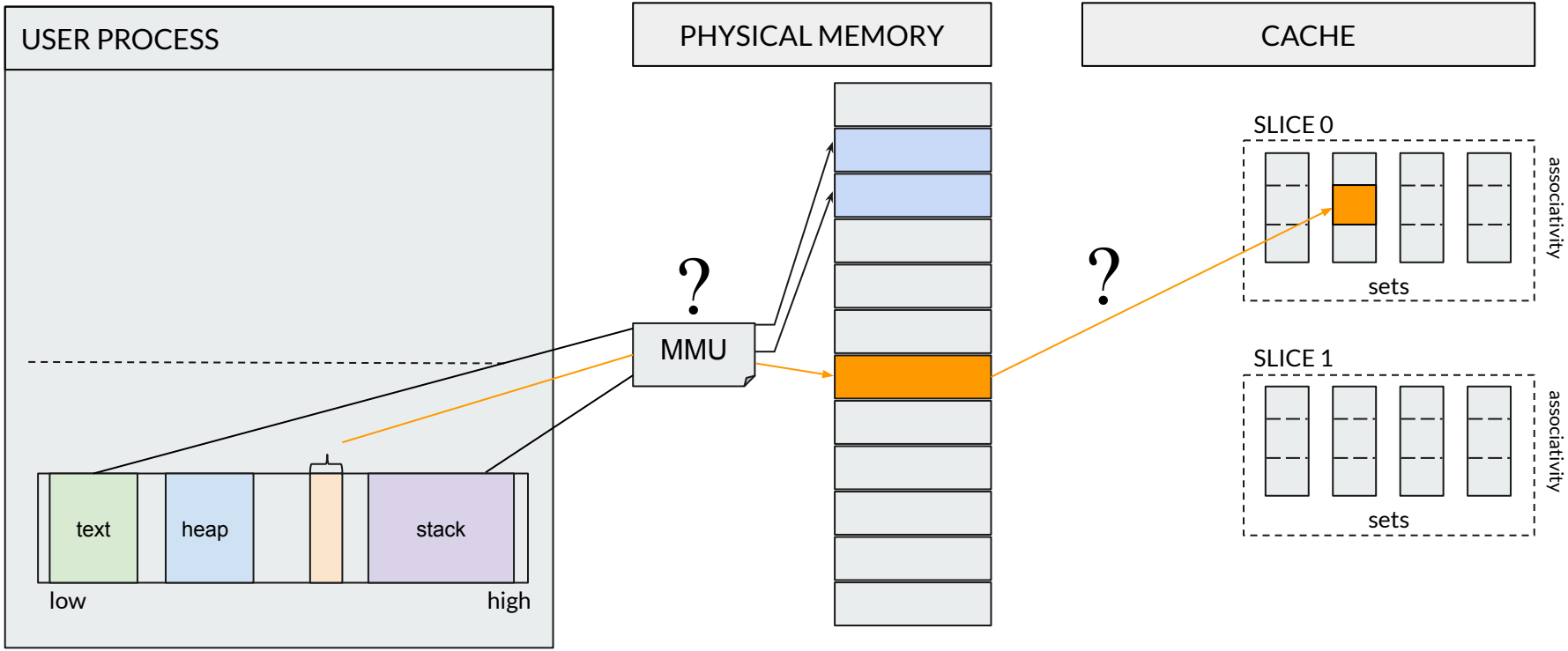
Problem

Potentially unknown mapping from physical address to cache set



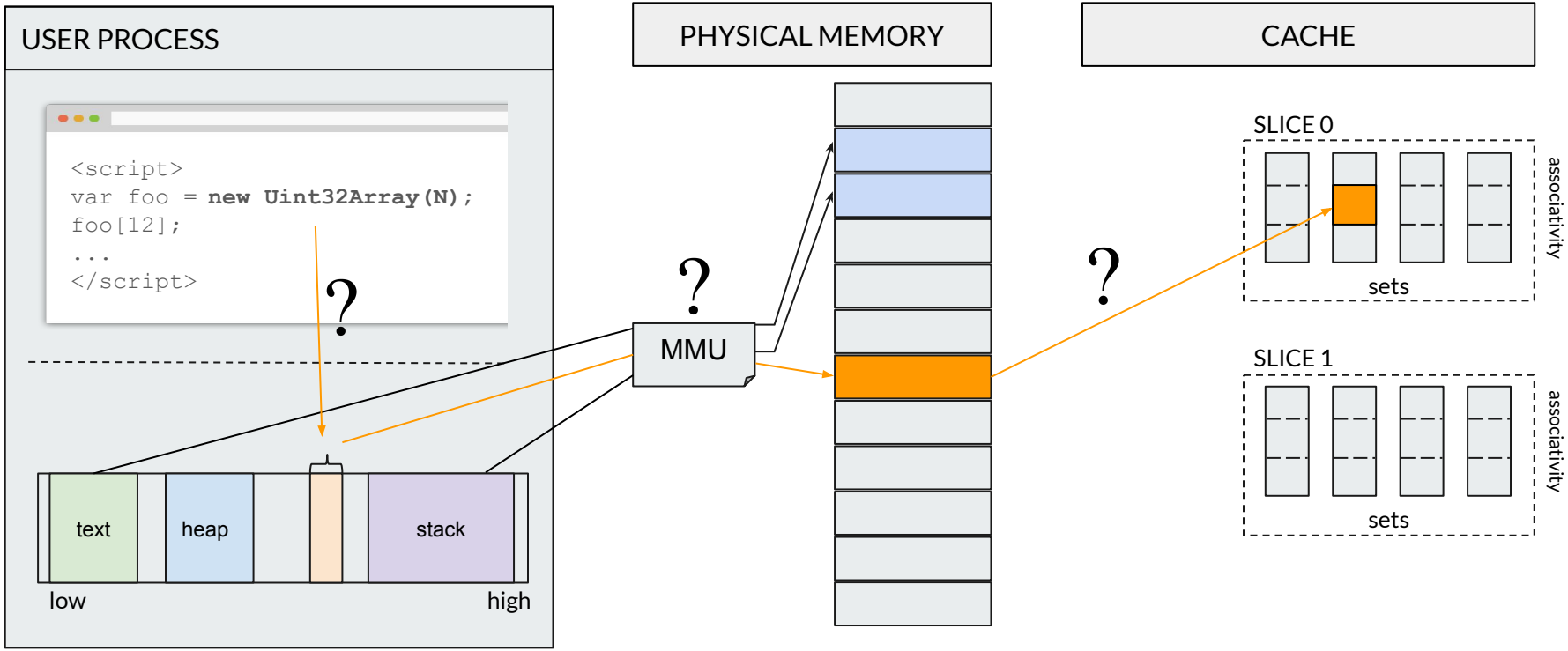
Problem

Unknown translation from virtual to physical addresses

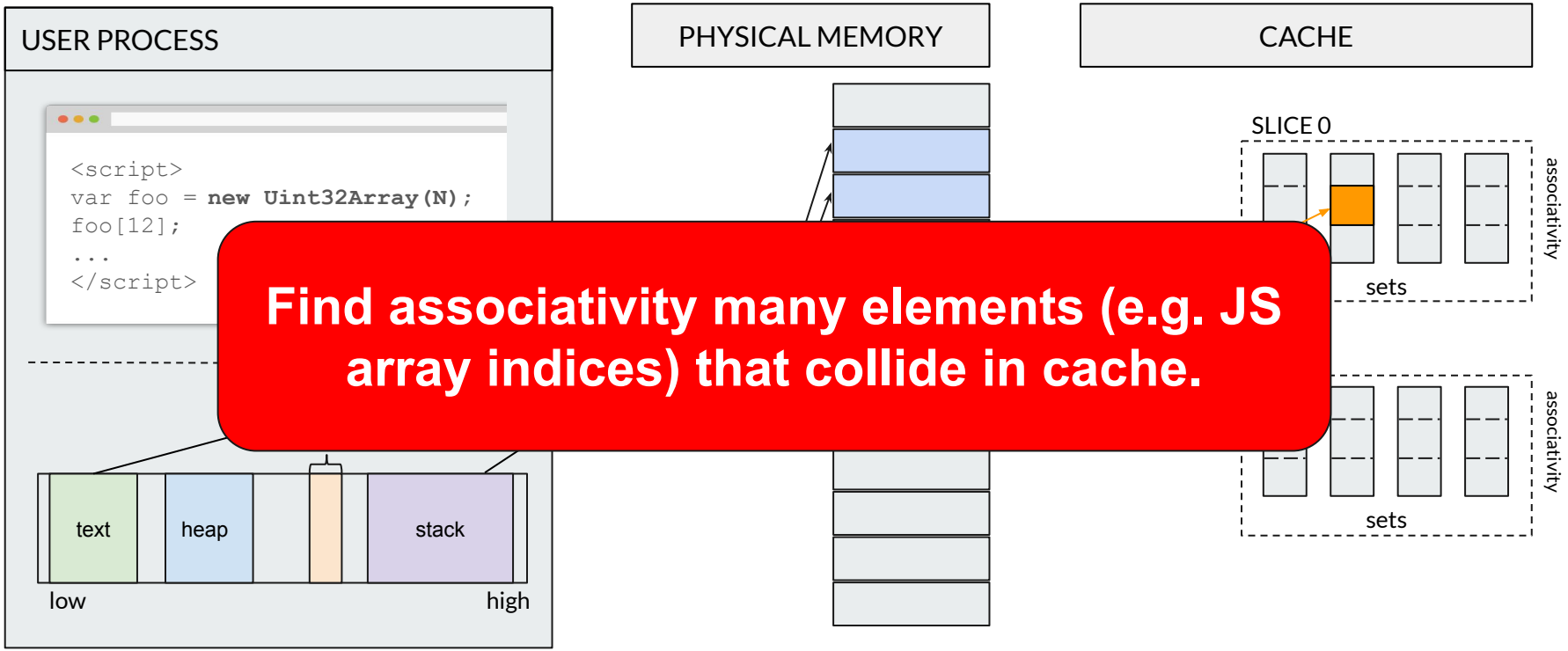


Problem

In some scenarios, even unknown virtual address



Problem


USER PROCESS
<script>
var foo = new Uint32Array(N);
foo[12];
...
</script>
text
heap
stack
low
high
PHYSICAL MEMORY
CACHE
SLICE 0
associativity
sets
associativity
sets
Find associativity many elements (e.g. JS array indices) that collide in cache.

Contributions

Systematic study of the problem of finding eviction sets

Contributions

Systematic study of the problem of finding eviction sets

Find eviction sets in O(n) compared to previous $O(n^2)$

Contributions

Systematic study of the problem of finding eviction sets

Find eviction sets in O(n) compared to previous $O(n^2)$

Reliability and performance evaluation
of algorithms in real hardware

Finding minimal eviction sets

1

Find a large eviction set for an address V:

- Pick “enough” addresses at random
- Timing test: $a_v\ a_0\ a_1\ \ldots\ a_n$ 

Finding minimal eviction sets

1

Find a large eviction set for an address V:

- Pick “enough” addresses at random
- Timing test: $a_v\ a_0\ a_1\ \ldots\ a_n$  a_v

2

Reduce initial large eviction set into its minimal core

Finding minimal eviction sets

1

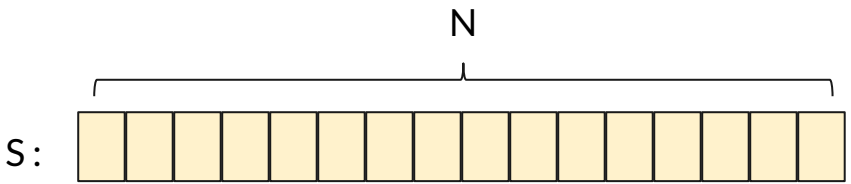
Find a large eviction set for an address V:

- Pick “enough” addresses at random
- Timing test: $a_v\ a_0\ a_1\ \ldots\ a_n\ a_v$

2

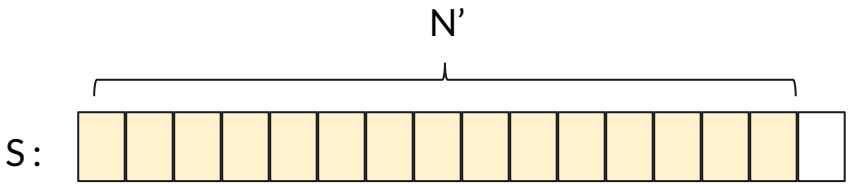
Reduce initial large eviction set into its minimal core

Baseline algorithm



Start with large enough eviction set S of size N

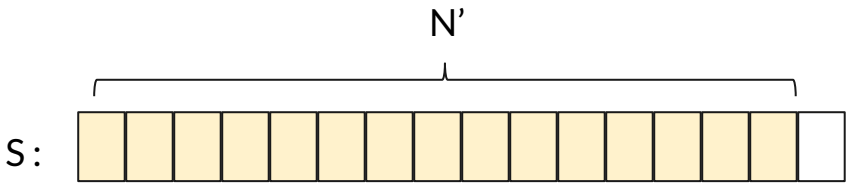
Baseline algorithm



Pick candidate element C, and

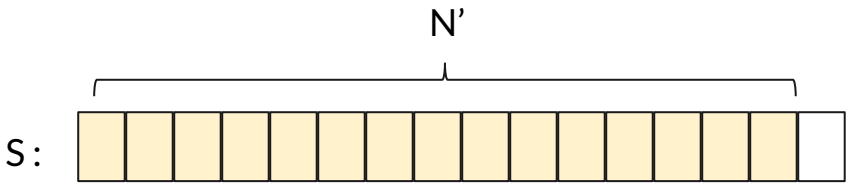
Test if remaining set TEST(S\{C}) is still an eviction set

Baseline algorithm



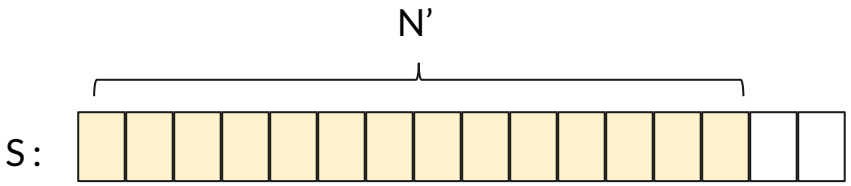
If TEST(S\{C}) = True, discard C

Baseline algorithm



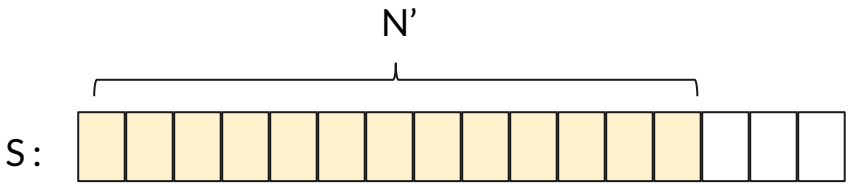
and continue with N’=N-1

Baseline algorithm



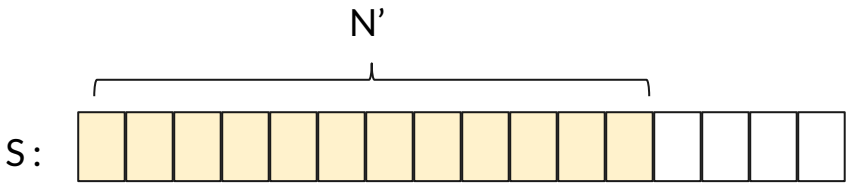
We repeat this process several times

Baseline algorithm



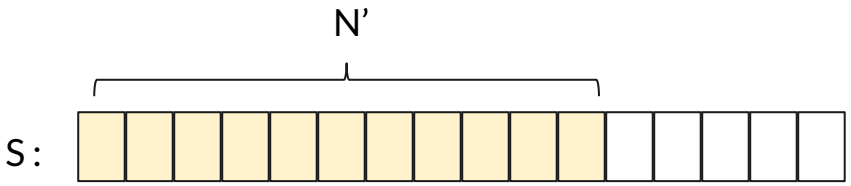
We repeat this process several times

Baseline algorithm



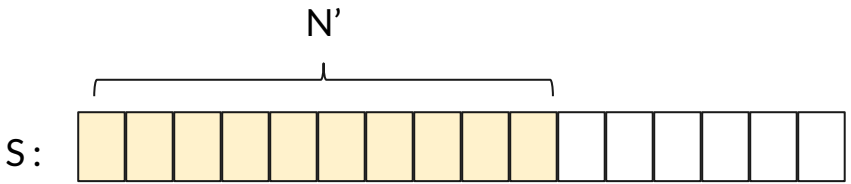
We repeat this process several times

Baseline algorithm



We repeat this process several times

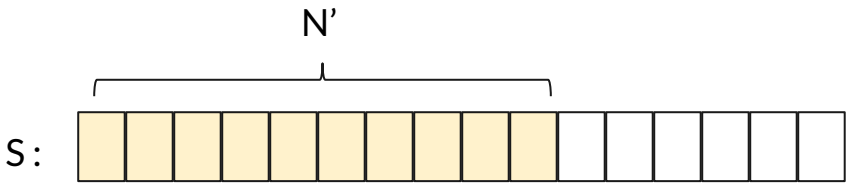
Baseline algorithm



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

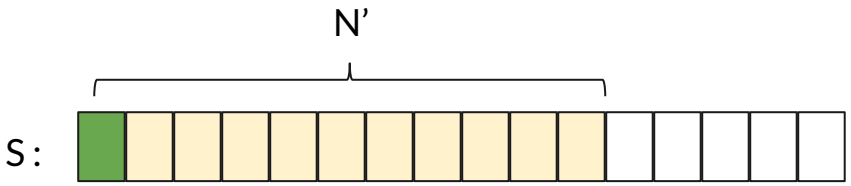
TEST(S\{C}) = False

Baseline algorithm



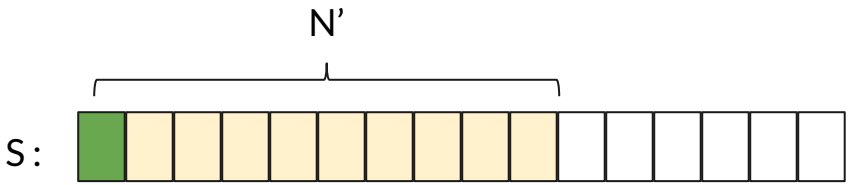
We learn that C is part of the minimal core

Baseline algorithm



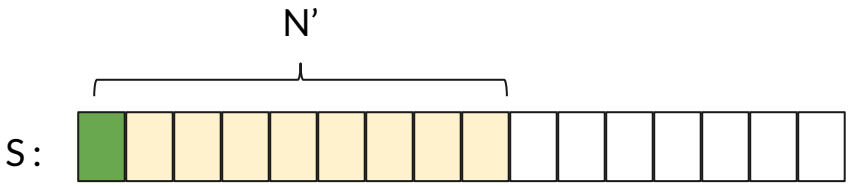
We keep track of it, and insert it again in S

Baseline algorithm



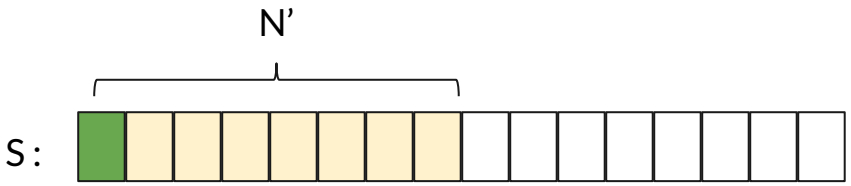
We repeat this process several times

Baseline algorithm



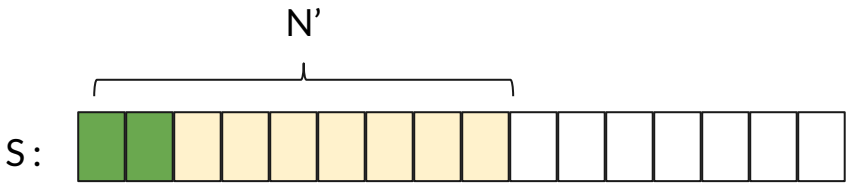
We repeat this process several times

Baseline algorithm



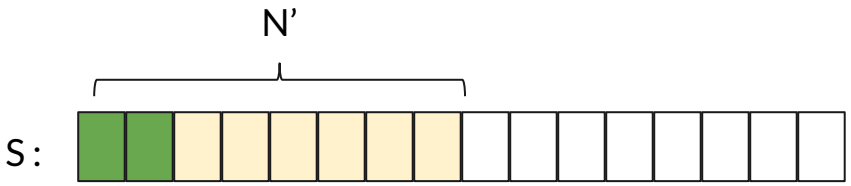
We repeat this process several times

Baseline algorithm



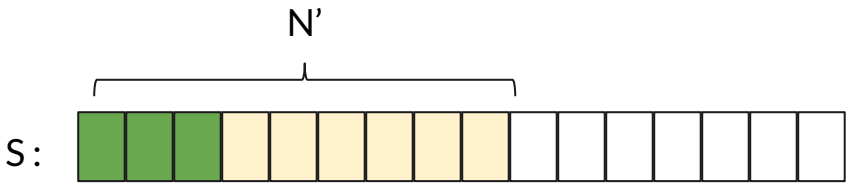
We repeat this process several times

Baseline algorithm



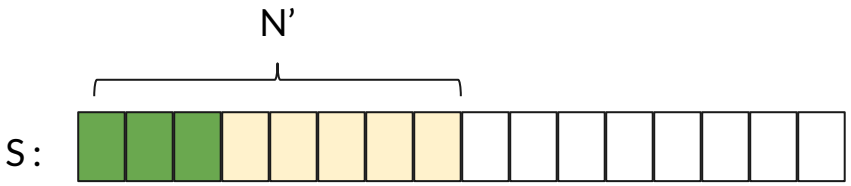
We repeat this process several times

Baseline algorithm



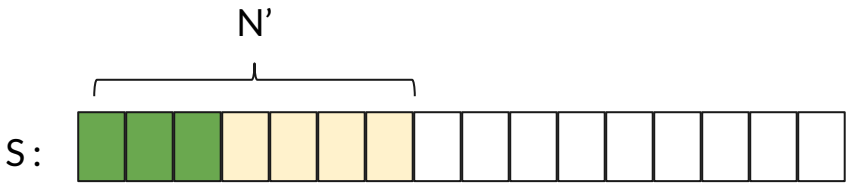
We repeat this process several times

Baseline algorithm



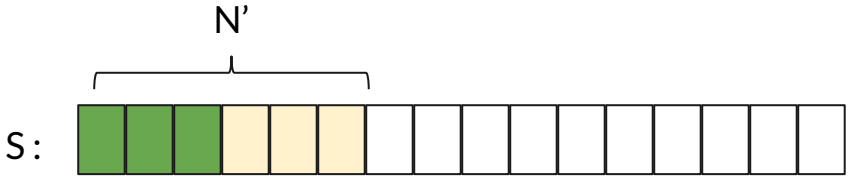
We repeat this process several times

Baseline algorithm



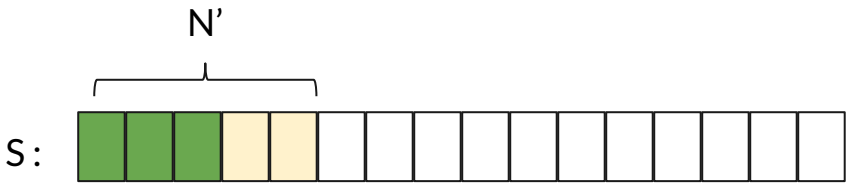
We repeat this process several times

Baseline algorithm



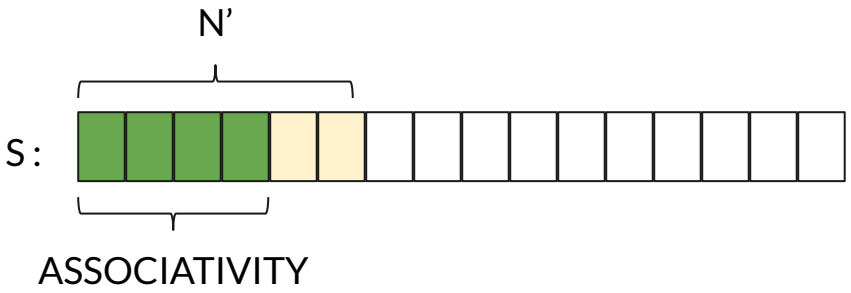
We repeat this process several times

Baseline algorithm



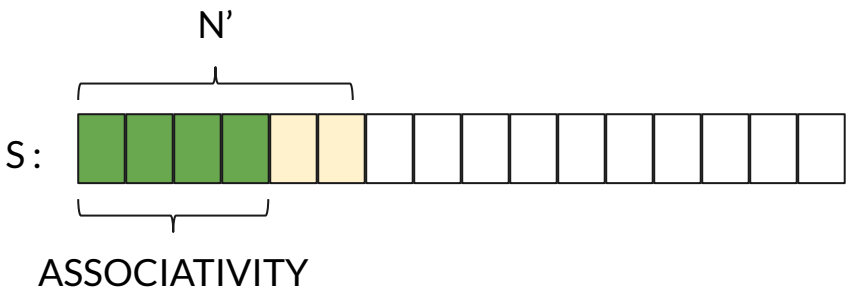
We repeat this process several times

Baseline algorithm



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

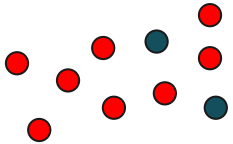
Baseline algorithm



$O(N^2)$ memory accesses

Threshold Group Testing

Group testing problem by Robert Dorfman (1943)

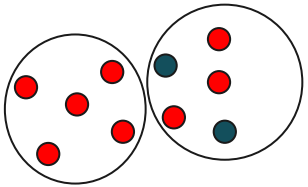


(10 individual tests)

Blood samples

Threshold Group Testing

Group testing problem by Robert Dorfman (1943)



(4 group tests +
3 individual tests)

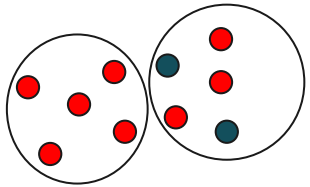
Blood samples

Threshold Group Testing

Group testing problem by Robert Dorfman (1943)

Generalization by Peter Damaschke (2006):

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



(4 group tests +
3 individual tests)

Blood samples

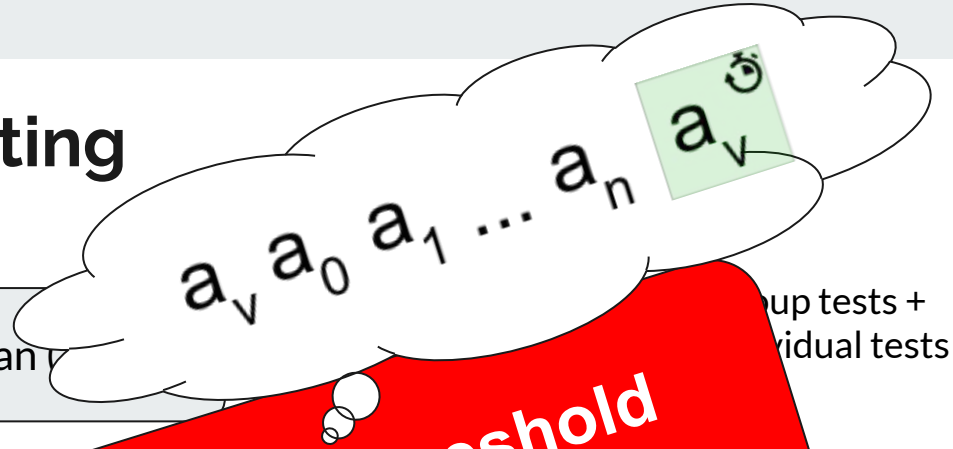
Threshold Group Testing

Group testing problem by Robert Dorfman (

...up tests +
...idual tests)

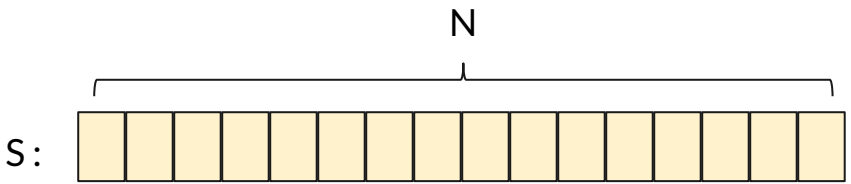
Generalization by Pete

- P
- N
- Ra



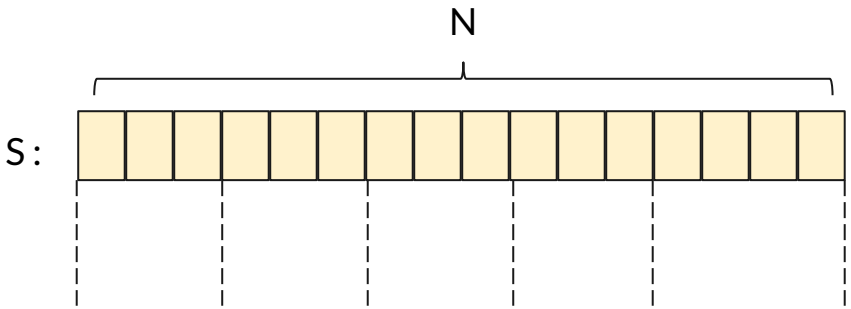
Observation: Our test is a threshold group test!

Group-testing algorithm



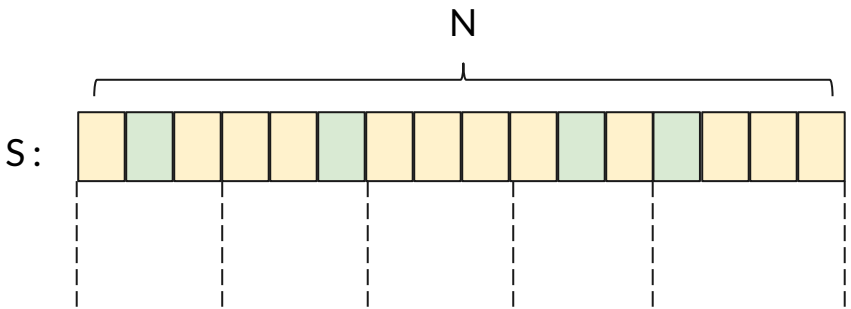
Start with large enough eviction set S of size N

Group-testing algorithm



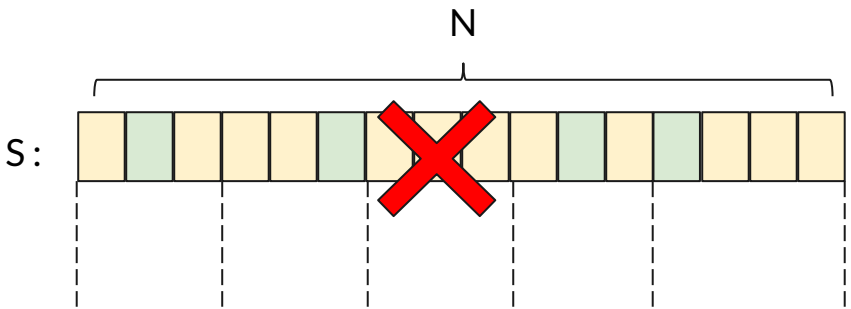
Split S in ASSOCIATIVITY+1 subsets

Group-testing algorithm



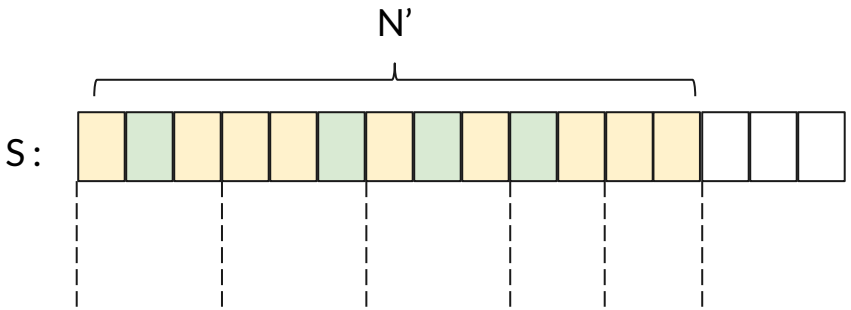
In the worst case, there exists a union of ASSOCIATIVITY subsets being an eviction set

Group-testing algorithm



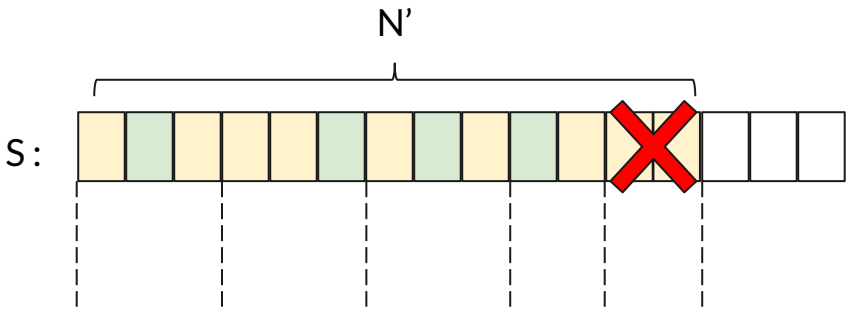
We can discard
N/(ASSOCIATIVITY+1) elements
per iteration

Group-testing algorithm



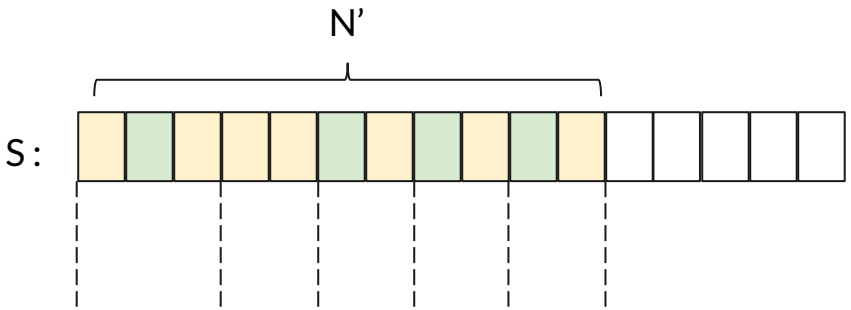
We repeat this process until we have ASSOCIATIVITY many elements

Group-testing algorithm



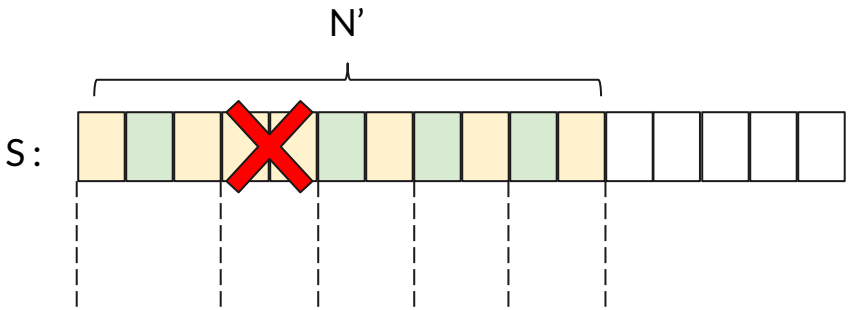
We repeat this process until we have ASSOCIATIVITY many elements

Group-testing algorithm



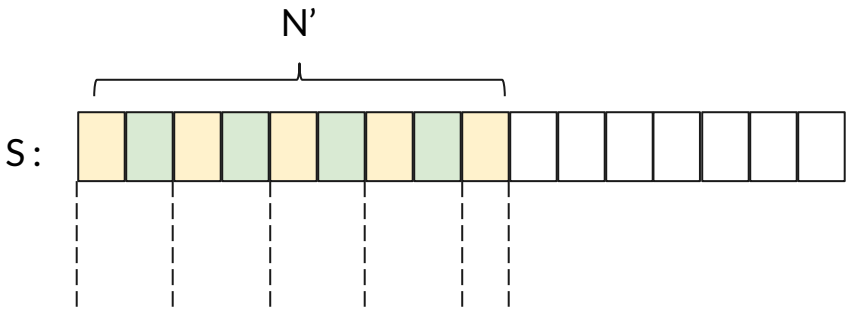
We repeat this process until we have ASSOCIATIVITY many elements

Group-testing algorithm



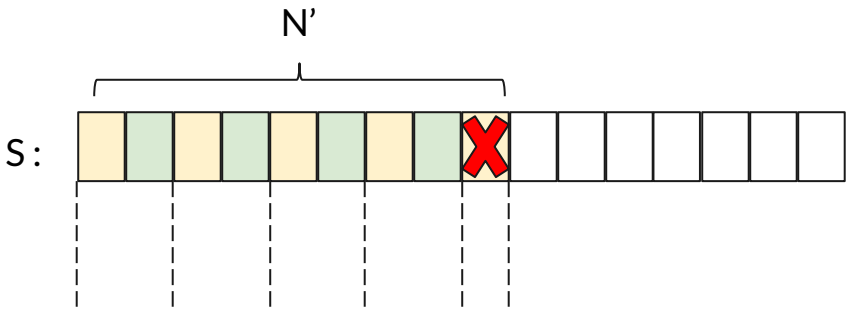
We repeat this process until we have ASSOCIATIVITY many elements

Group-testing algorithm



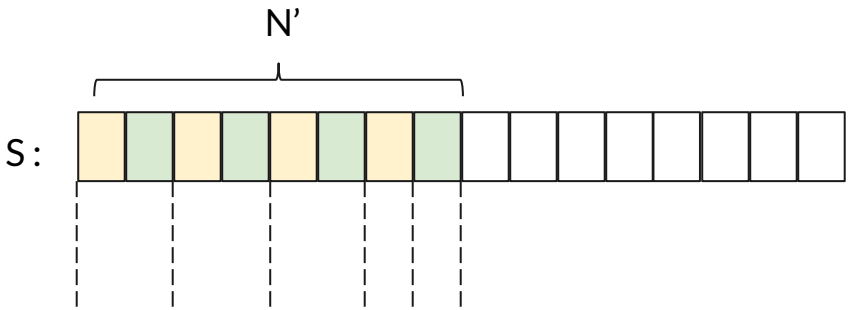
We repeat this process until we have ASSOCIATIVITY many elements

Group-testing algorithm



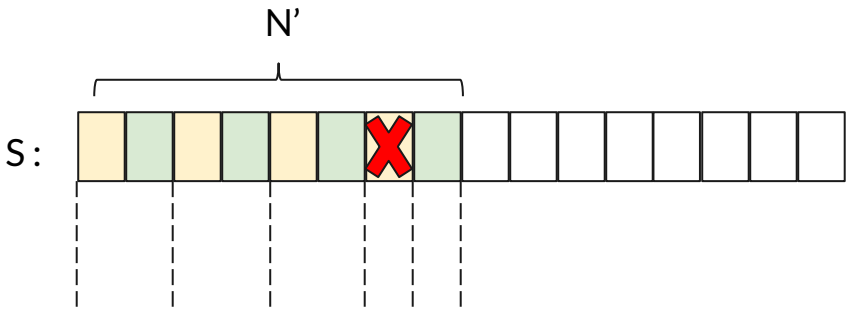
We repeat this process until we have ASSOCIATIVITY many elements

Group-testing algorithm



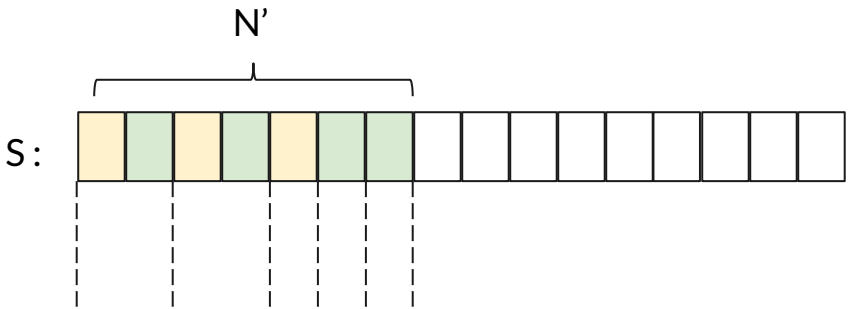
We repeat this process until we have ASSOCIATIVITY many elements

Group-testing algorithm



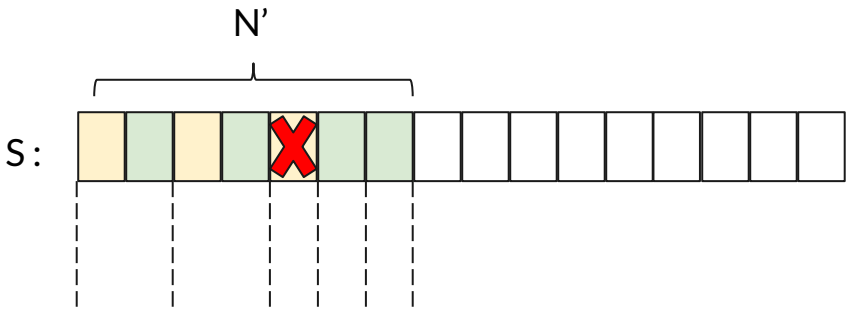
We repeat this process until we have ASSOCIATIVITY many elements

Group-testing algorithm



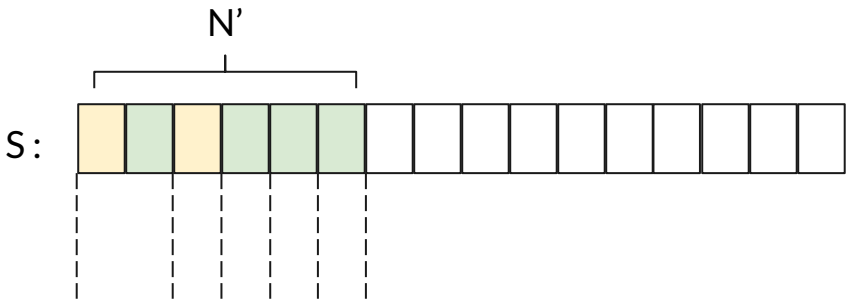
We repeat this process until we have ASSOCIATIVITY many elements

Group-testing algorithm



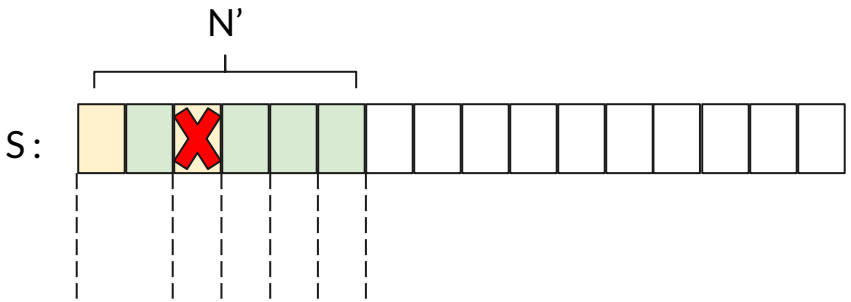
We repeat this process until we have ASSOCIATIVITY many elements

Group-testing algorithm



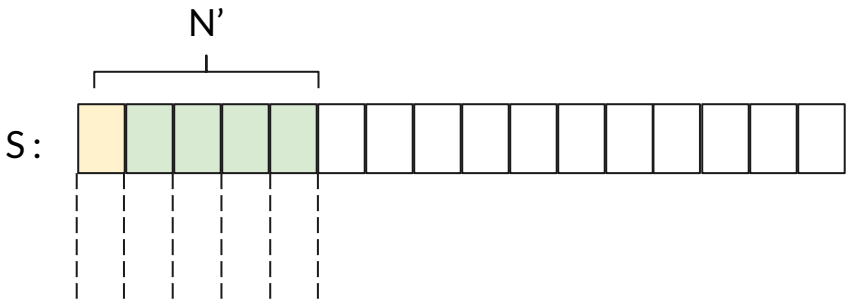
We repeat this process until we have ASSOCIATIVITY many elements

Group-testing algorithm



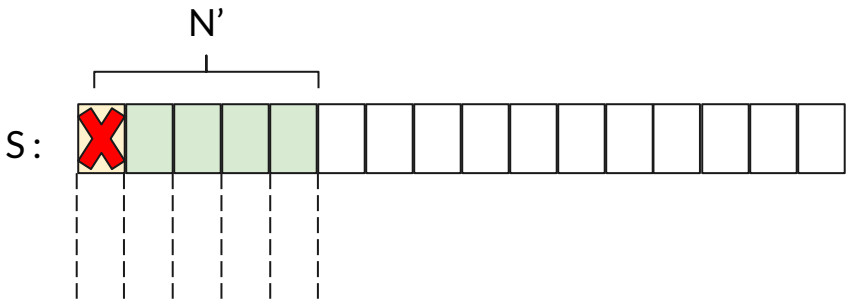
We repeat this process until we have ASSOCIATIVITY many elements

Group-testing algorithm



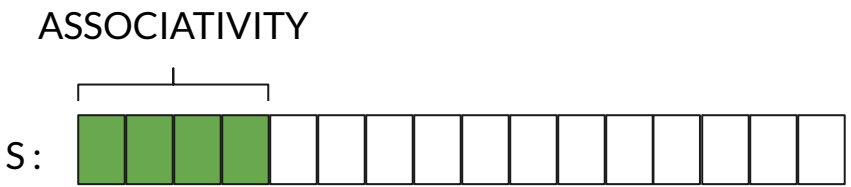
We repeat this process until we have ASSOCIATIVITY many elements

Group-testing algorithm



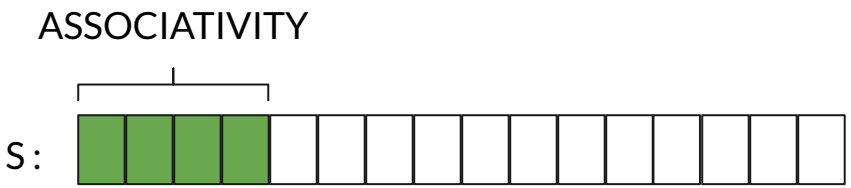
We repeat this process until we have ASSOCIATIVITY many elements

Group-testing algorithm



We find our minimal eviction set!

Group-testing algorithm



O(N) mem accesses

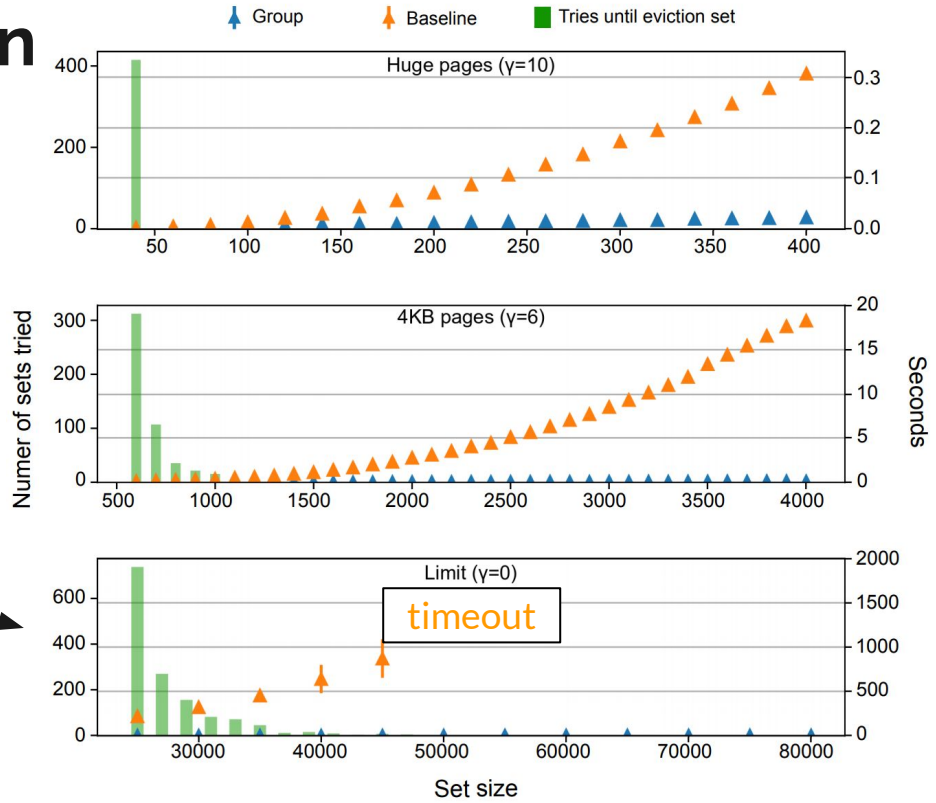
Experiments on Skylake i5-6500 with 6MB cache (8192 sets x 12 assoc)

Performance Evaluation

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

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

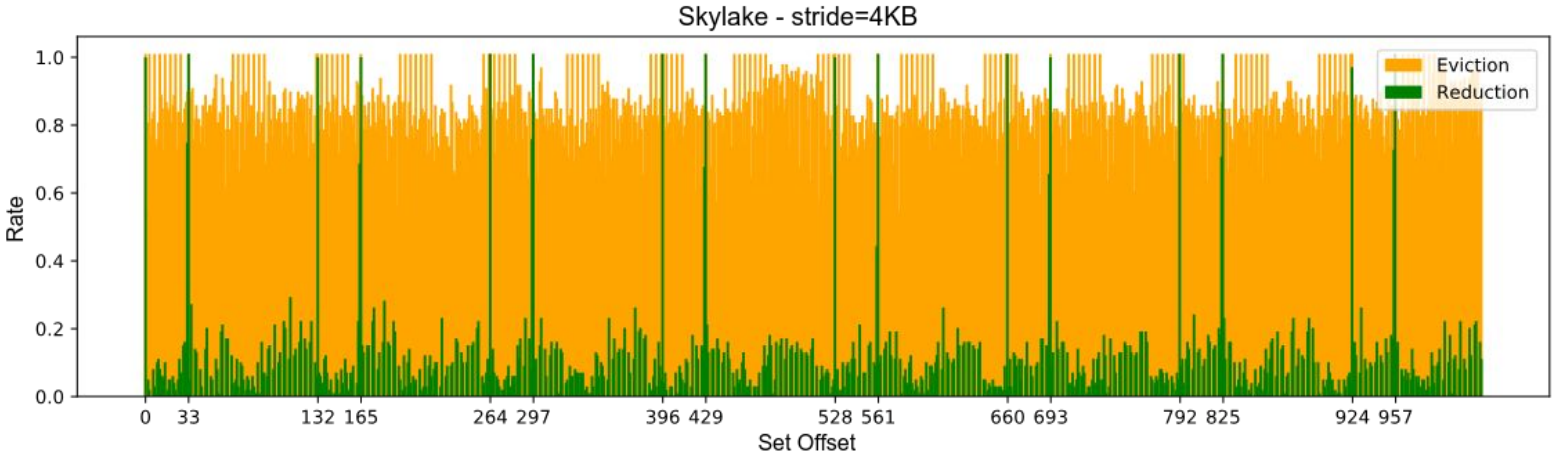
Finding minimal eviction sets is practical without knowledge on any bits of the set index!



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

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

Live demo (Bonus Material!) 😍

Find minimal eviction sets on Chrome with JS and Wasm

Conclusions

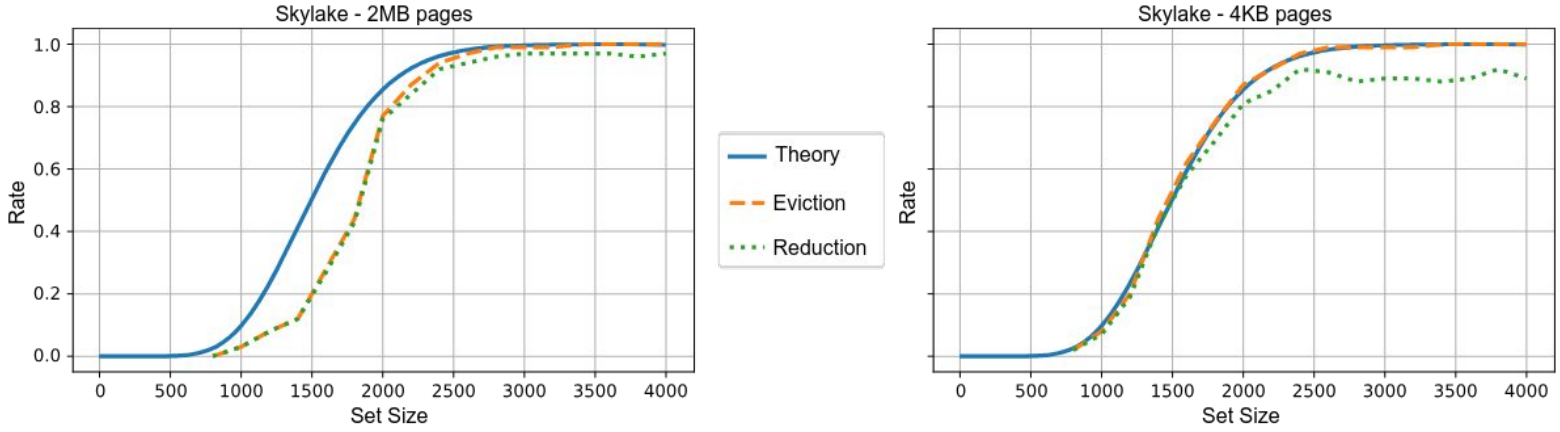
Finding minimal eviction sets is a threshold group-testing problem: new insight for research on principled countermeasures

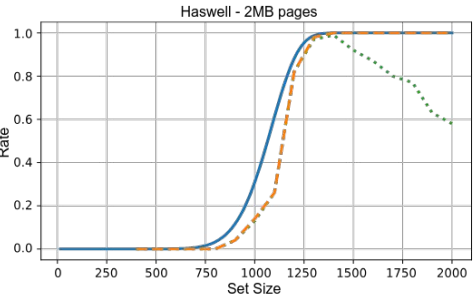
Novel linear-time algorithm makes attacks faster and enables them in scenarios previously considered impractical

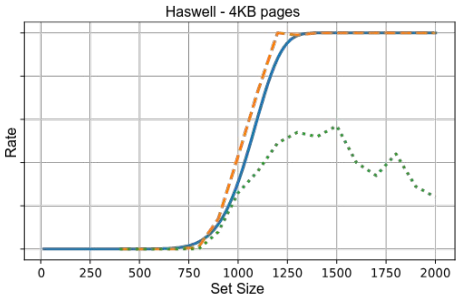
Thanks for your attention

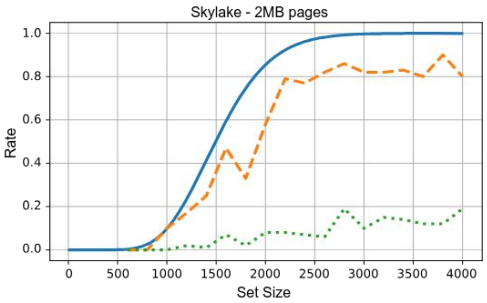
Questions?

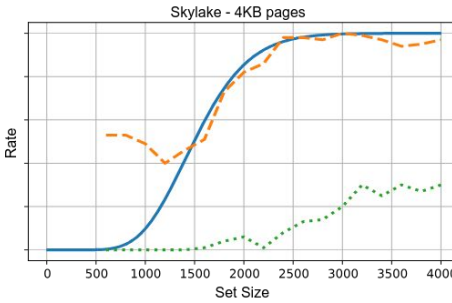
Noise on difference machines and cache sets











Demo verification scripts

`run.sh`:

```
google-chrome-beta --user-data-dir=/tmp/tmp.u9lo18kaTh
--js-flags='--allow-natives-syntax --experimental-wasm-bigint'
http://localhost:8000/ | ./verify_addr.sh
```

`--allow-natives-syntax`: used for printing found indices to stdout
`--experimental-wasm-bigint`: only for convenience, will have default support soon

`verify_addr.sh`:

- find chrome's PID
- use `pmap` to find base virtual address for JS buffer
- read JS indices and add them to virtual address base
- execute `./virt_to_phys` to translate virtual to physical addresses using `/proc/pid/pagemap`
- extract slice and cache index set from physical address (uses Intel's reverse engineered hash function)

Example of TEST() in Wasm

(on V8 v7.4.0)

traverse.wat

```
(func $x (param $ptr i64)
  (loop $iter
    (set_local $ptr
      (i64.load
        (i32.wrap/i64 (get_local $ptr))))
    (br_if $iter
      (i32.eqz (i64.eqz (get_local $ptr)))))
)
```

TurboFan x86_64 output

```
0  55                          push rbp
1  4889e5                      movq rbp,rsp
4  6a0a                        push 0xa
6  56                          push rsi
7  4883ec10                    subq rsp,0x10
b  488b9ea7000000              movq rbx,[rsi+0xa7]
12 6666660f1f840000000000      nop
1d 0f1f00                      nop
20 488b96c7000000              movq rdx,[rsi+0xc7]
27 483922                      cmpq [rdx],rsp
2a 0f8340000000                jnc <+0x70>
30 8bc0                        movl rax,rax
32 49ba0000000001000000        movq r10,0x100000000
3c 4c3bd0                      cmpq r10,rax
3f 7320                        jnc <+0x61>
                               ...
61 488b0403                    movq rax,[rbx+rax*1]
65 4883f800                    cmpq rax,0x0
69 75b5                        jnz <+0x20>
6b 488be5                      movq rsp,rbp
6e 5d                          pop rbp
6f c3                          retl
```

prologue

stack guard

sign check

epilogue