

SP22 CSE599d: Hardware Security

DAVID KOHLBRENNER



Today: Caches and Projects

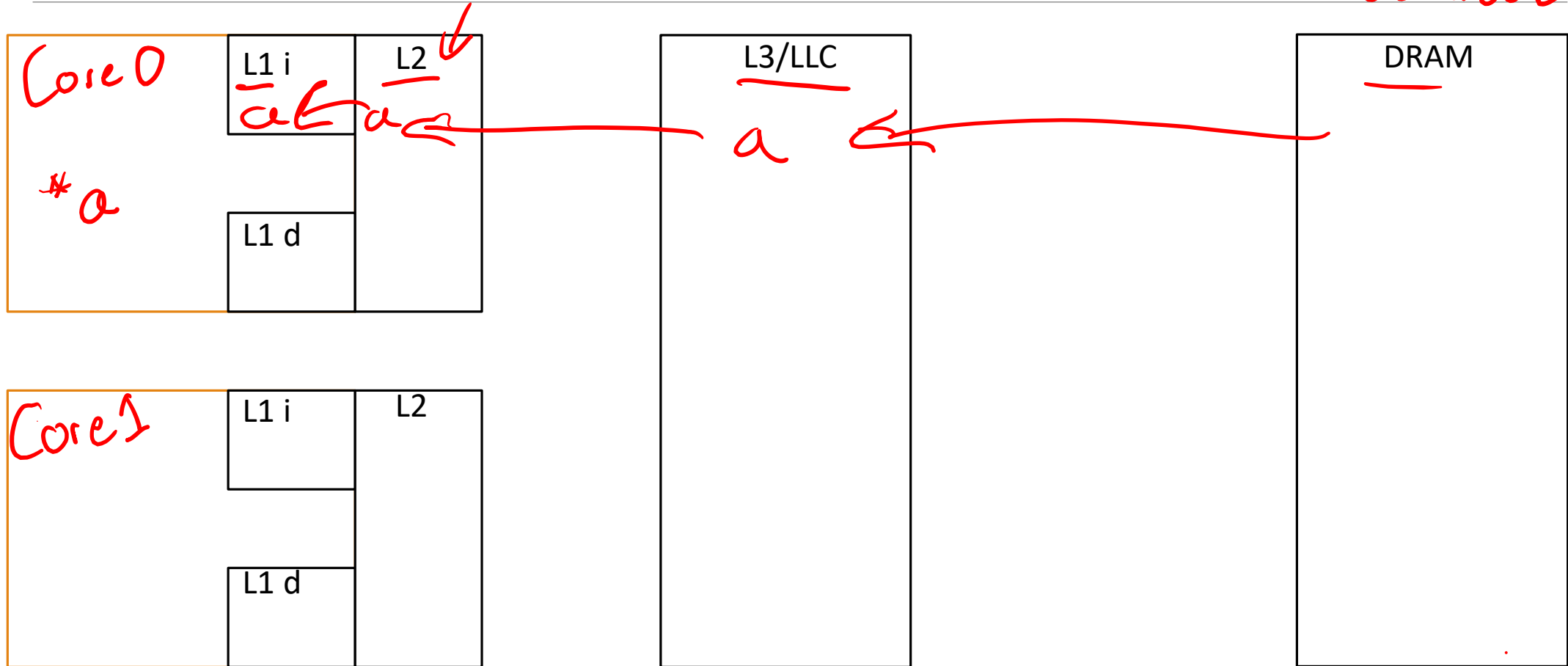
Recap

You'll need to:

- Sign up for (3?) presentations during the course, 2 people per-day
- Form a group (2-3 people) for a project
- We'll start meeting next week (or the week after) for project planning

Cache basics

100-200c



Different caches for different designs

Generally 1-3 levels of caching

Inclusive vs Exclusive vs Non-inclusive

Cache lines

Cache eviction policies

...

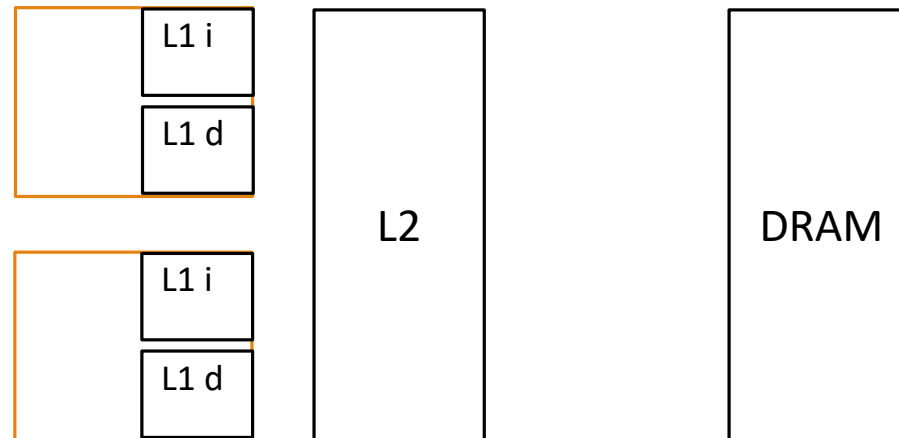


Cache Missing for Fun and Profit

~2005

Mostly about the Pentium 4

- 128 cache lines of 64 bytes each, organized into 32 4-way associative sets



Aside: Difficulty of measurement

How do we measure 'how long' something takes?

What if 'something' is *really* fast?

granularity

Start = time()

action()

Stop = time()

Colin's solution

RDTSC – `ReadTimeStampCounter`

- Fills `edx,eax` with current timestamp

Still not good enough

“This alone would not be measurable, thanks to the long latency of the RDTSC (read time stamp counter) instruction [...]”

Colin's solution

RDTSC – ReadTimeStampCounter

- Fills edx, eax with current timestamp

Still not good enough


“This alone would not be measurable, thanks to the long latency of the RDTSC (read time stamp counter) instruction , **but this problem is resolved by adding some high-latency instructions – for example, integer multiplications – into the critical path**”


→ rdtsc
x ← *a
x ← x * 3
x ← x * 2


Measuring effectively (in 2022)


The Improved Benchmarking Method


The solution to the problem presented in [Section 0](#) is to add a CPUID instruction just after the RDTSCP and the two mov instructions (to store in memory the value of edx and eax). The implementation is as follows:

```
asm volatile ("CPUID\n\t"  
             "RDTSC\n\t"  
             "mov %%edx, %0\n\t"  
             "mov %%eax, %1\n\t": "=r" (cycles_high), "=r" (cycles_low)::  
"%rax", "%rbx", "%rcx", "%rdx");  
/*****/  
/*call the function to measure here*/  
/*****/  
asm volatile("RDTSCP\n\t"  
             "mov %%edx, %0\n\t"  
             "mov %%eax, %1\n\t"  
             "CPUID\n\t": "=r" (cycles_high1), "=r" (cycles_low1)::  
"%rax", "%rbx", "%rcx", "%rdx");
```

CPUID 

RDTSC 

mov [start_high], edx 

mov [start_low], eax 

call measure_me * a

RDTSCP b

mov [end_high], edx —

mov [end_low], eax —

CPUID —

Back to caches

SMT (or Hyperthreading[®] Intel)

Idea: Why not run two different threads at the same time!

→ Thread 0 is sometimes waiting for memory, etc

Let Thread 1 use that time!

Result: Thread 0 and Thread 1 compete for L1 cache 😊

(Thanks to Dean, Susan, and Hank for making it practical)

Back to caches (really)

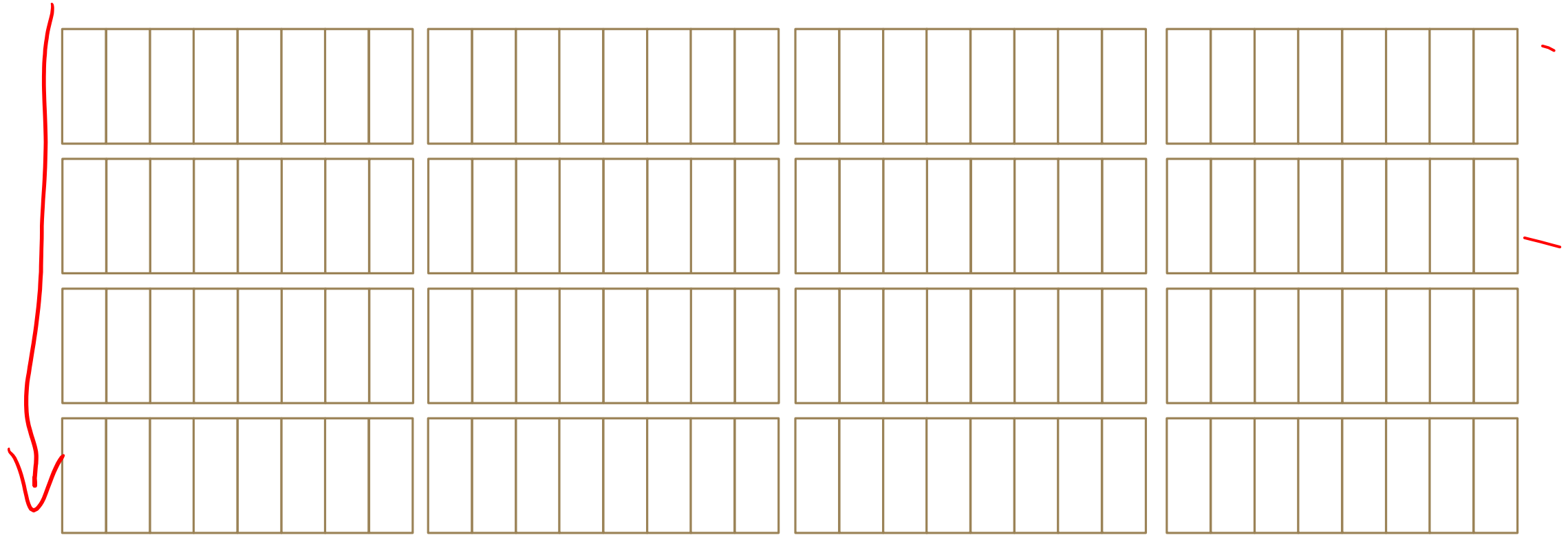
spy = +0
sender = +1

%32

L1 Cache misses as covert channel

32-bit

sets



L1 Cache misses as covert channel

“Using this code, 32 bits can be reliably transmitted from the Trojan to the Spy in roughly 5000 cycles with a bit error rate of under 25%; using an appropriate error correcting code, this provides a covert channel of 400 kilobytes per second on a 2.8 GHz processor.”

L2 Cache misses as covert channel

L2: “4096 cache lines of 128 bytes each, organized into 512 8-way associative sets”

Now needs to worry about TLB as well

Also prefetchers

L1 Cache misses as side-channel

Apply what we learned to attack RSA implementations

L1 Cache misses as side-channel

Apply what we learned to attack RSA implementations

OpenSSL (0.9.7c) uses several vulnerable patterns:

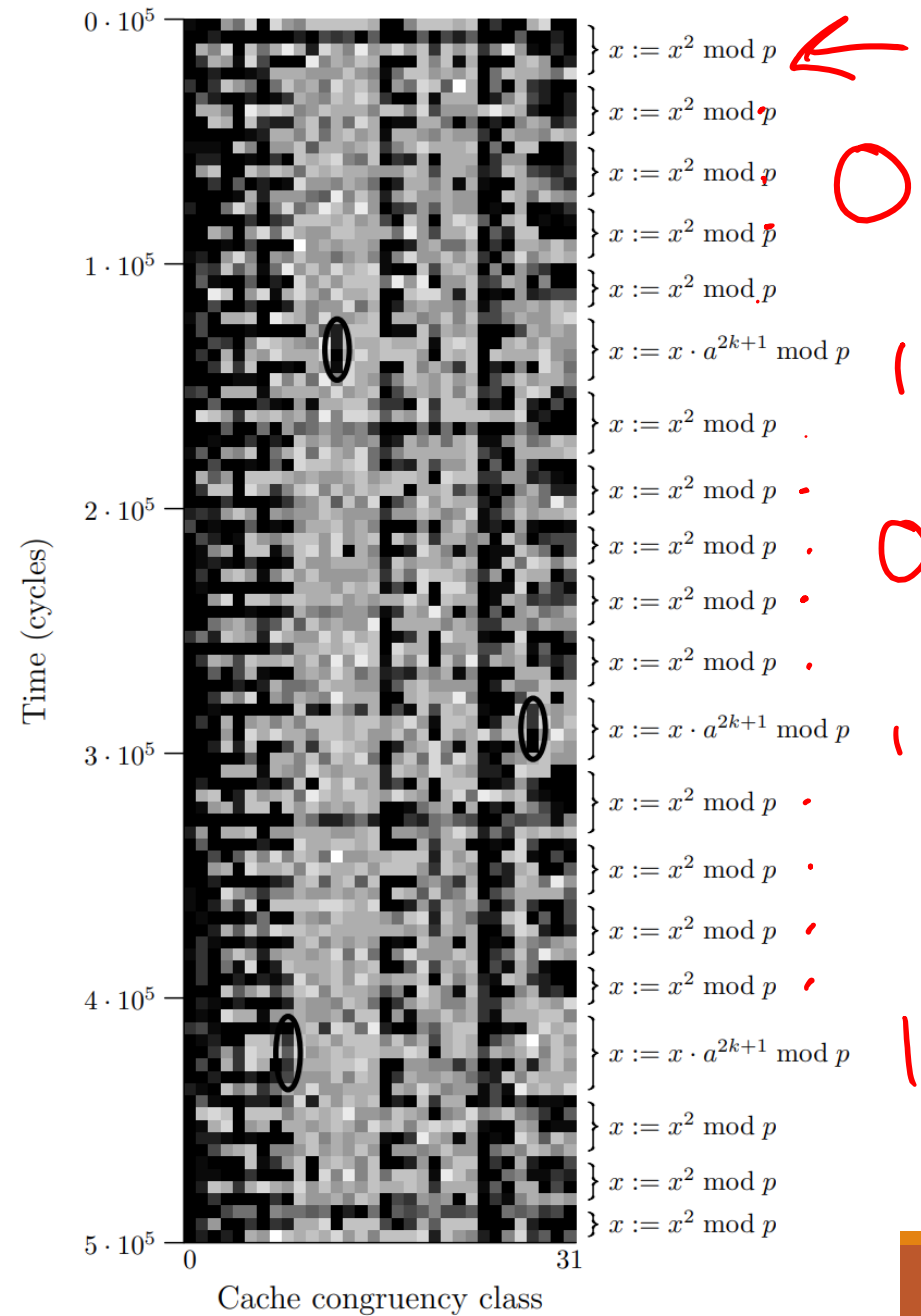
- Decomposes modular exponentiation into repeated squares or multiplications
- Multiplications are against precomputed values

$$M^d \bmod N$$

x^2

$a_n * x$

CACHE MISSING FOR FUN AND PROFIT



a

Result of attack

310 out of 512 bits of each RSA modulus

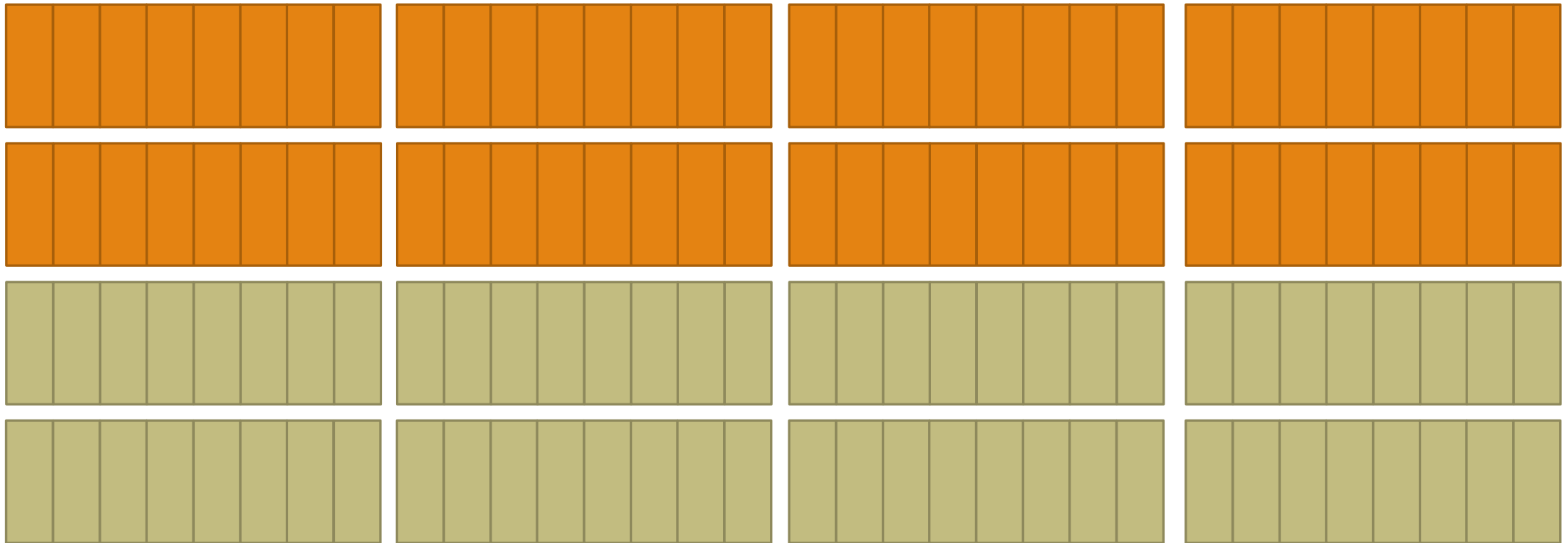
Can now solve the rest!

Solutions

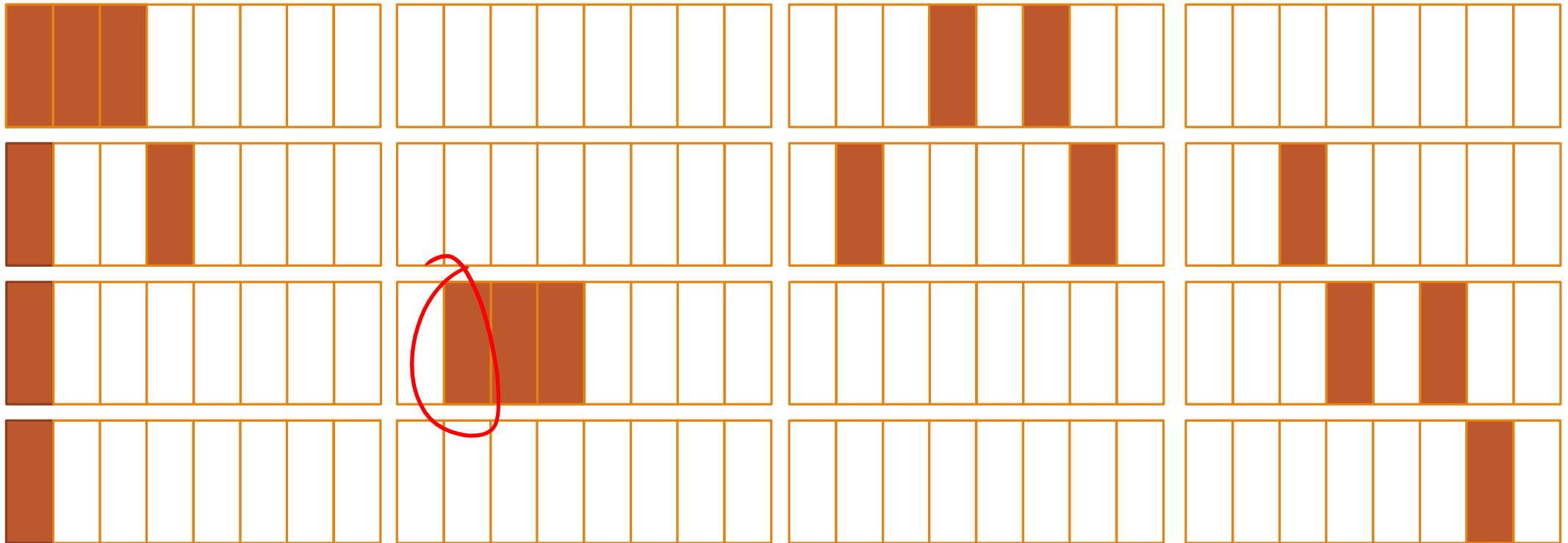
Isolation

set

34



Isolation (pt2)



Isolation (pt3)

Or, refuse to schedule 2 different threads at the same time

(Q: Does this work?)

Removal of channel

Rewrite code to not leak based on cache behavior

“This would be a dramatic divergence from existing practice , and would require that some existing algorithms be thrown out or reworked considerably”

“This approach has the disadvantage of requiring a vast amount of code to be audited if it is to be carried out comprehensively”

$a_0 \rightarrow$
 $a_1 \rightarrow$
 a_2
 $\rightarrow \vdots$
 $a_{secret}(2)$

Mitigation of channel

Remove ability to measure time (or, limit it)

fuzzy time