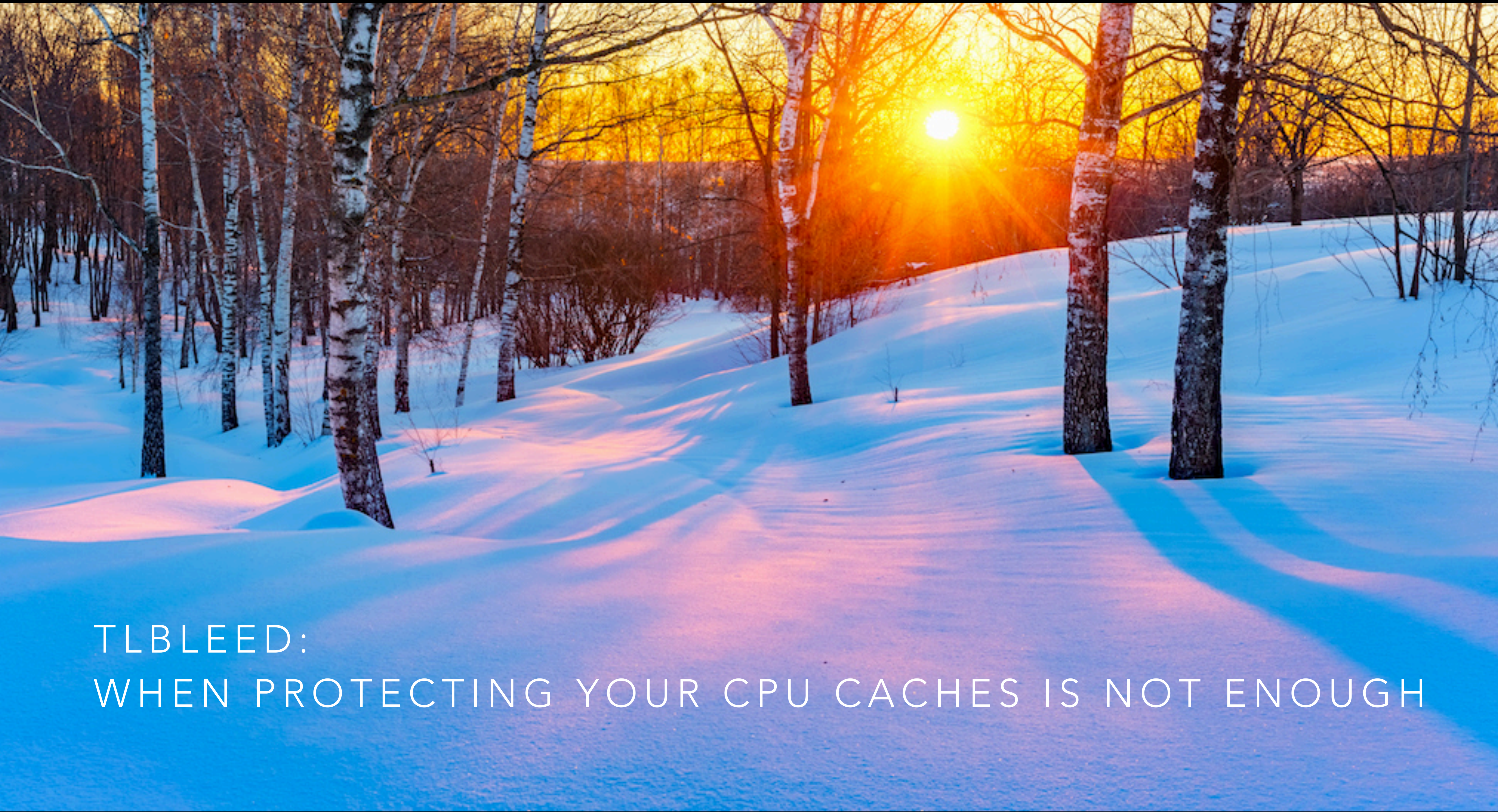
BEN GRAS @BJG
KAVEH RAZAVI @KAVEHRAZAVI
CRISTIANO GIUFFRIDA, HERBERT BOS
VRIJE UNIVERSITEIT AMSTERDAM

HARDWEAR.IO 2018



TLBLEED: WHEN PROTECTING YOUR CPU CACHES IS NOT ENOUGH

ABOUT US

VUSec

ABOUT US

- VUsec - Security Research group at VU Amsterdam

ABOUT US

- VUsec - Security Research group at VU Amsterdam
- Academic group researching systems software security

ABOUT US

- VUsec - Security Research group at VU Amsterdam
- Academic group researching systems software security
- We do software hardening, exploitation

ABOUT US

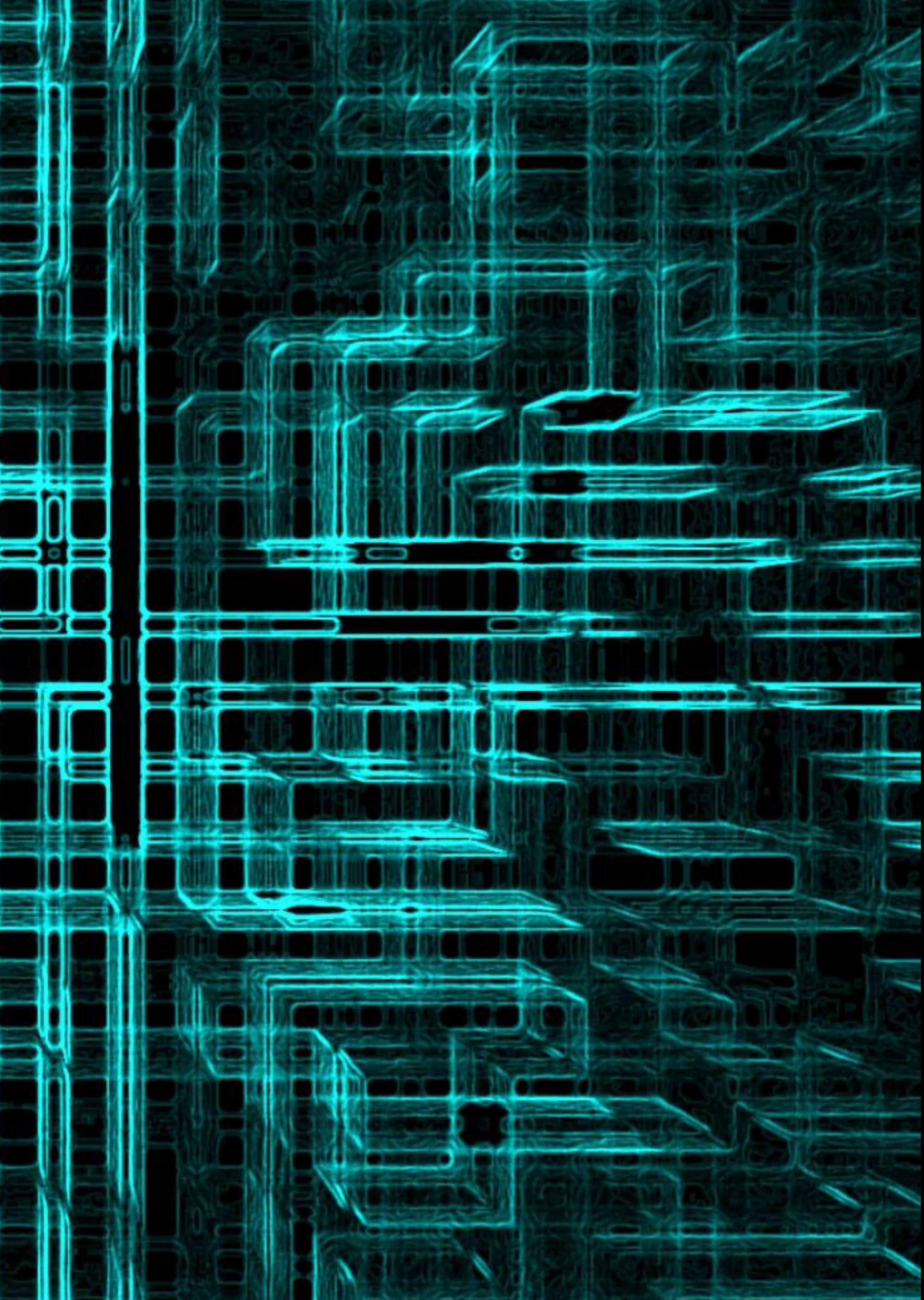
- VUsec - Security Research group at VU Amsterdam
- Academic group researching systems software security
- We do software hardening, exploitation
- Hardware attacks, side channels

ABOUT US

- VUsec - Security Research group at VU Amsterdam
- Academic group researching systems software security
- We do software hardening, exploitation
- Hardware attacks, side channels

OVERVIEW

- Cache attacks
- Cache defences
- TLBleed
- Evaluation
- Reception



CACHE ATTACKS

SIDE CHANNELS

SIDE CHANNELS

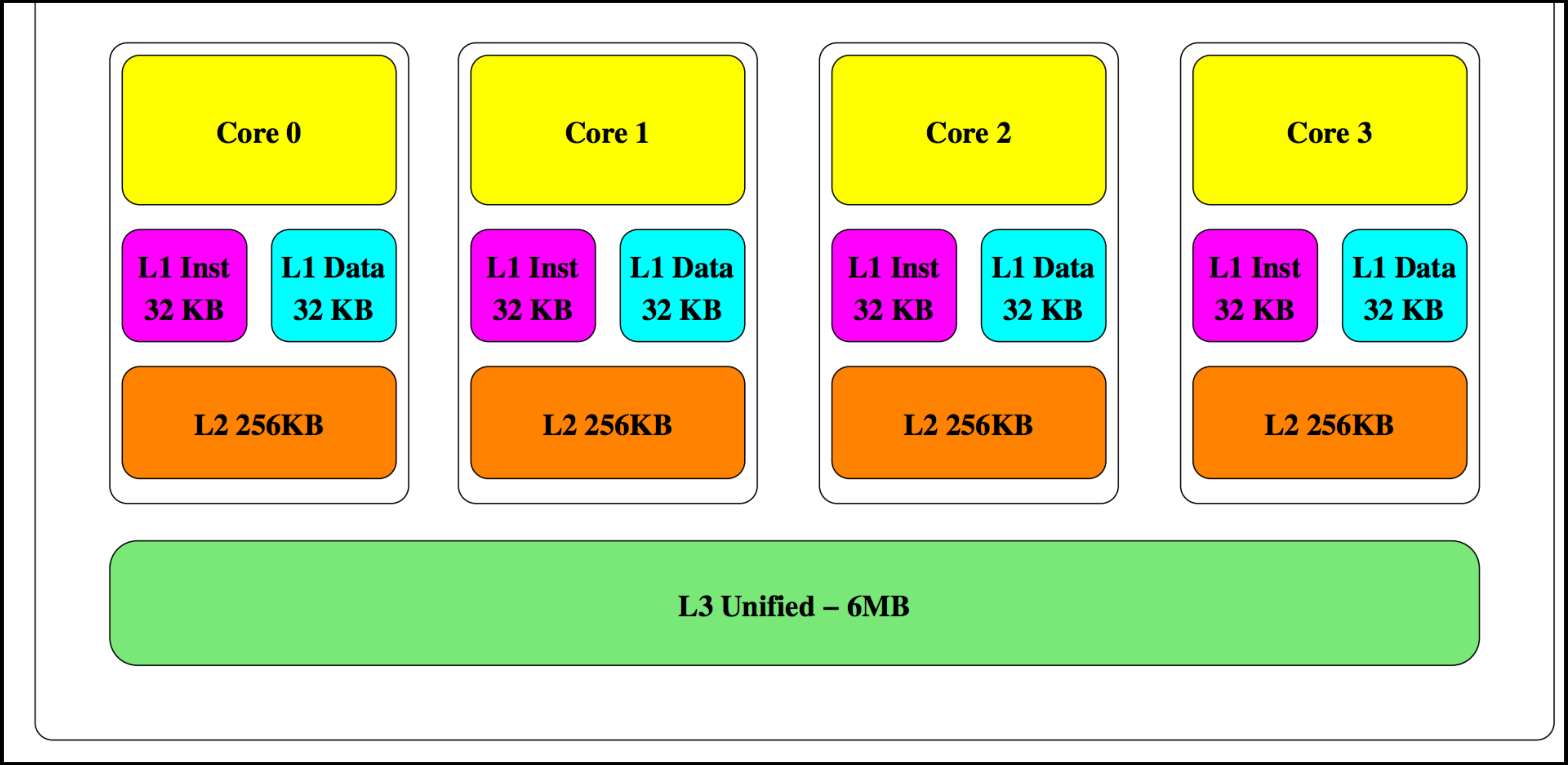
- Leak secrets outside the regular interface

SIDE CHANNELS

- Leak secrets outside the regular interface

INSIDE A CPU

INSIDE A CPU


Core 0
L1 Inst 32 KB
L1 Data 32 KB
L2 256KB
Core 1
L1 Inst 32 KB
L1 Data 32 KB
L2 256KB
Core 2
L1 Inst 32 KB
L1 Data 32 KB
L2 256KB
Core 3
L1 Inst 32 KB
L1 Data 32 KB
L2 256KB
L3 Unified – 6MB

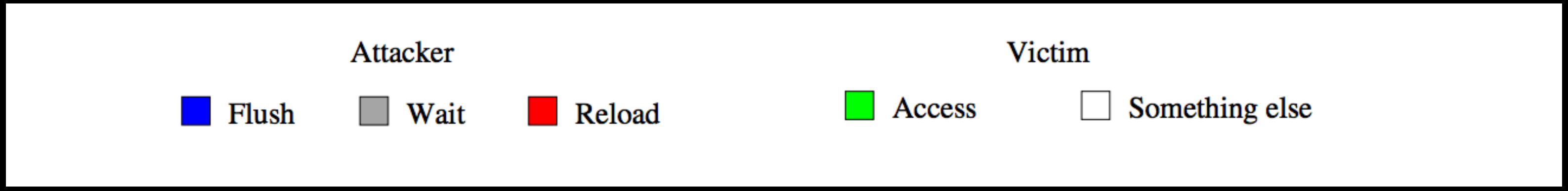
SIDE CHANNEL ATTACKS ON SHARED RESOURCES

- There are **shared** resources between processes
- RAM, CPU cache, TLB, computational resources ..
- Covert channels
- Sometimes: Side channels (spying)

EXAMPLE: FLUSH+RELOAD

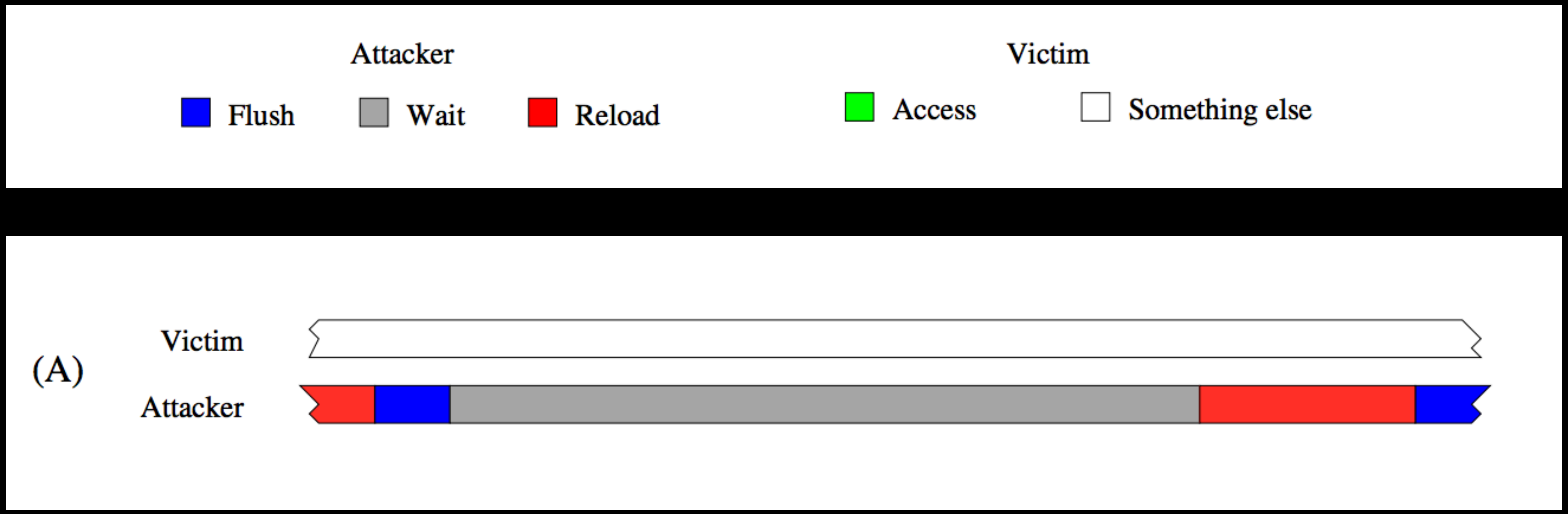
- Work by Yuval Yarom, Katrina Falkner

EXAMPLE: FLUSH+RELOAD



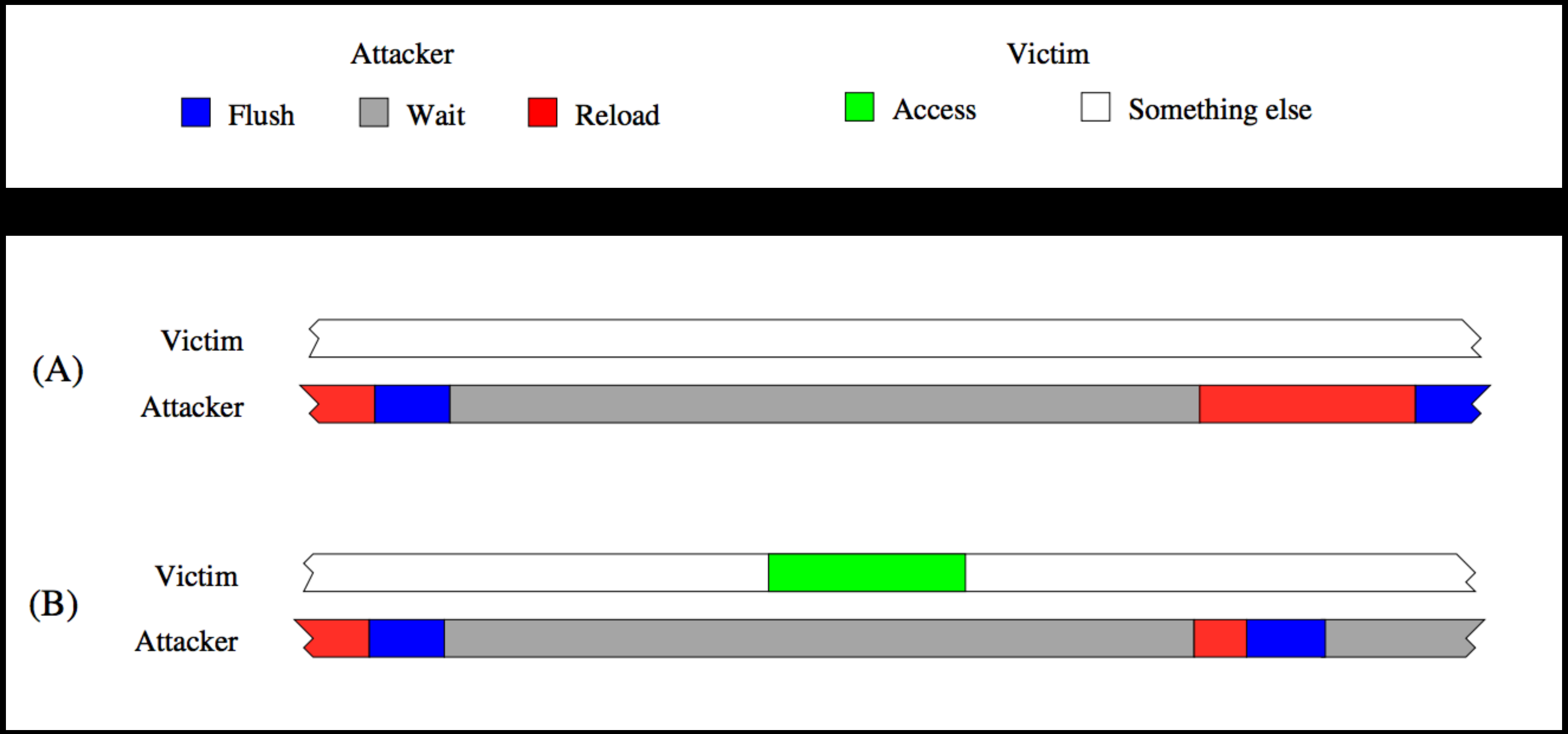
- Work by Yuval Yarom, Katrina Falkner

EXAMPLE: FLUSH+RELOAD



- Work by Yuval Yarom, Katrina Falkner

EXAMPLE: FLUSH+RELOAD



- Work by Yuval Yarom, Katrina Falkner

EXAMPLE: FLUSH+RELOAD

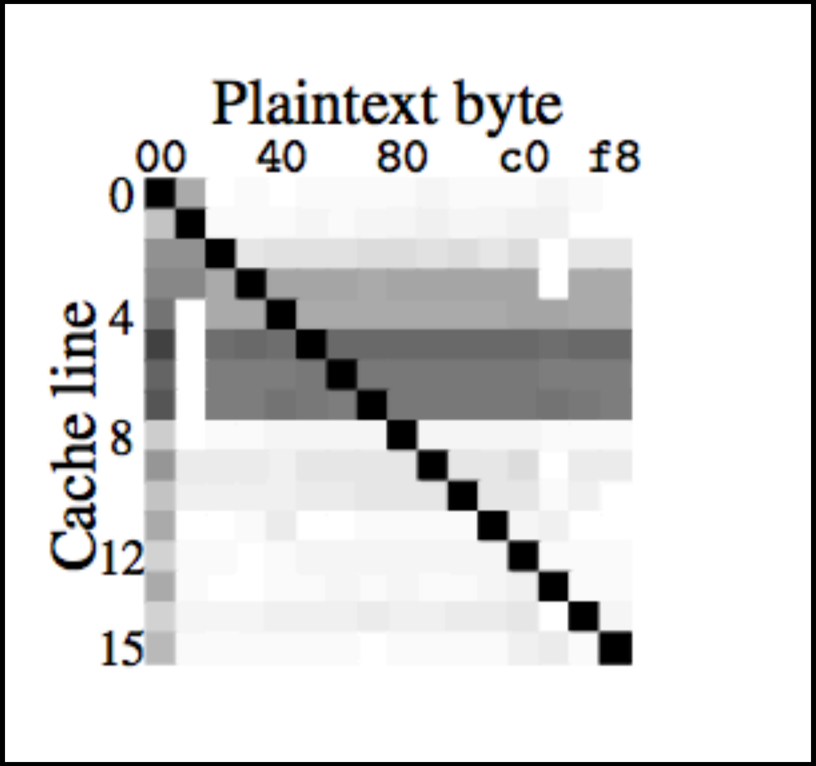
- Can attack AES implementation with T tables
- A table lookup happens $T_j\,[x_i = p_i \oplus k_i\,]$
- p_i is a plaintext byte, k_i a key byte

EXAMPLE: FLUSH+RELOAD

- Again: secrets are betrayed by memory accesses
- Known plaintext + accesses = key recovery

EXAMPLE: FLUSH+RELOAD

- Again: secrets are betrayed by memory accesses
- Known plaintext + accesses = key recovery

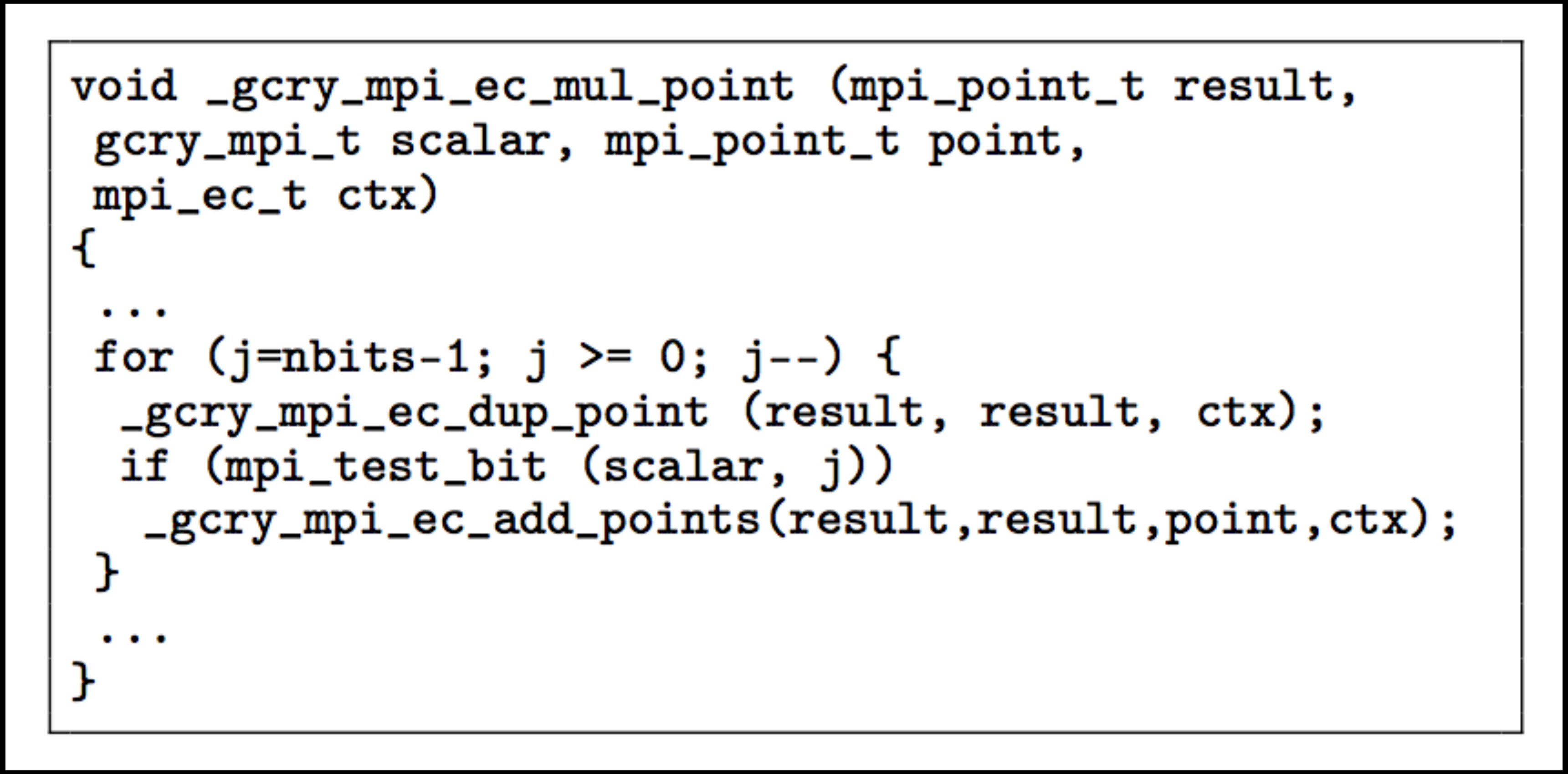


EXAMPLE: LIBGCRYPT ECC

- Not side channel proof version:

EXAMPLE: LIBGCRYPT ECC

- Not side channel proof version:

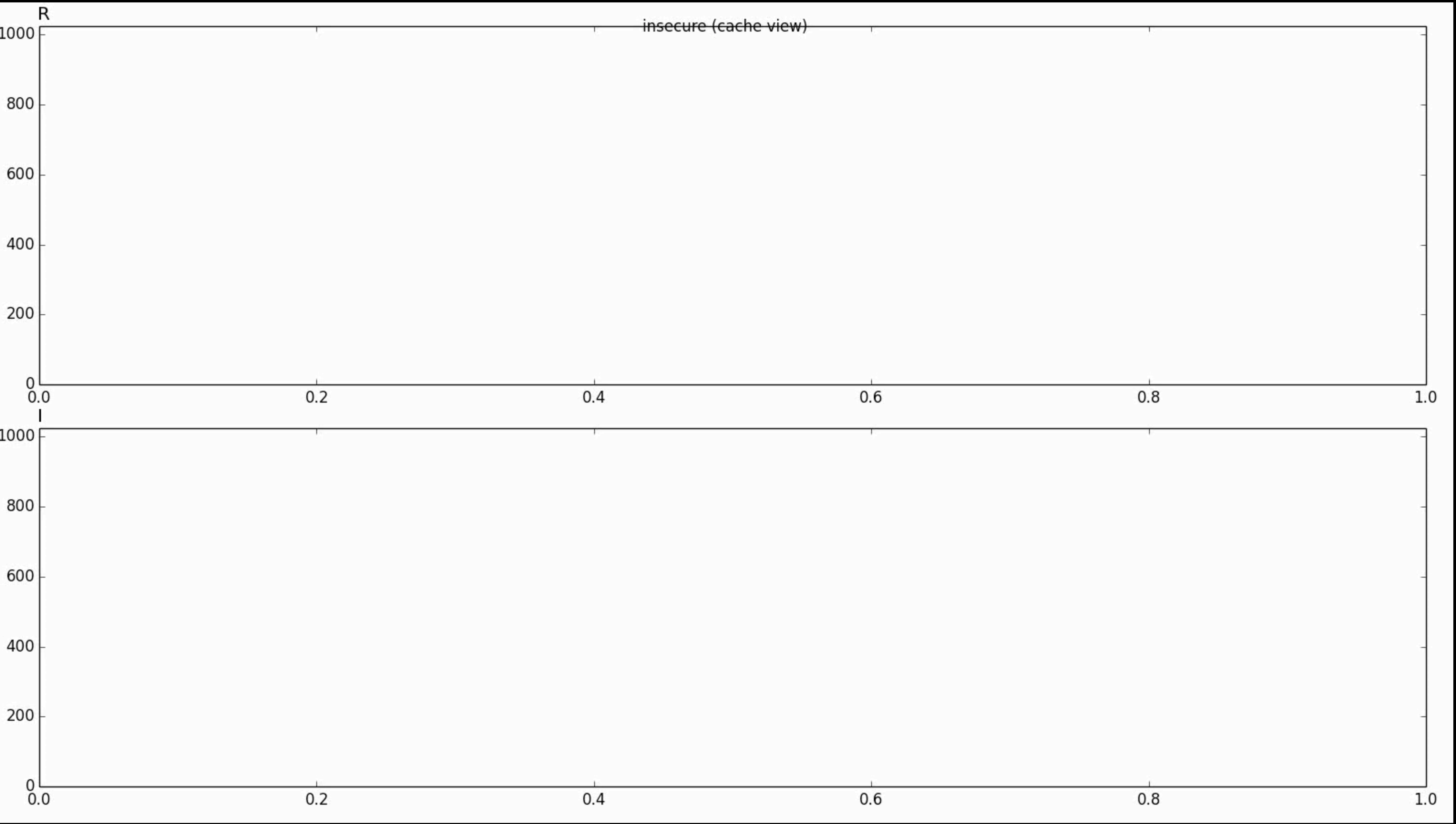


```
void _gcry_mpi_ec_mul_point (mpi_point_t result,
 gcry_mpi_t scalar, mpi_point_t point,
 mpi_ec_t ctx)
{
 ...
 for (j=nbits-1; j >= 0; j--) {
  _gcry_mpi_ec_dup_point (result, result, ctx);
  if (mpi_test_bit (scalar, j))
   _gcry_mpi_ec_add_points(result,result,point,ctx);
 }
 ...
}
```

EXAMPLE: LIBGCRYPT ECC

- Not side channel proof version:

EXAMPLE: LIBGCRYPT ECC

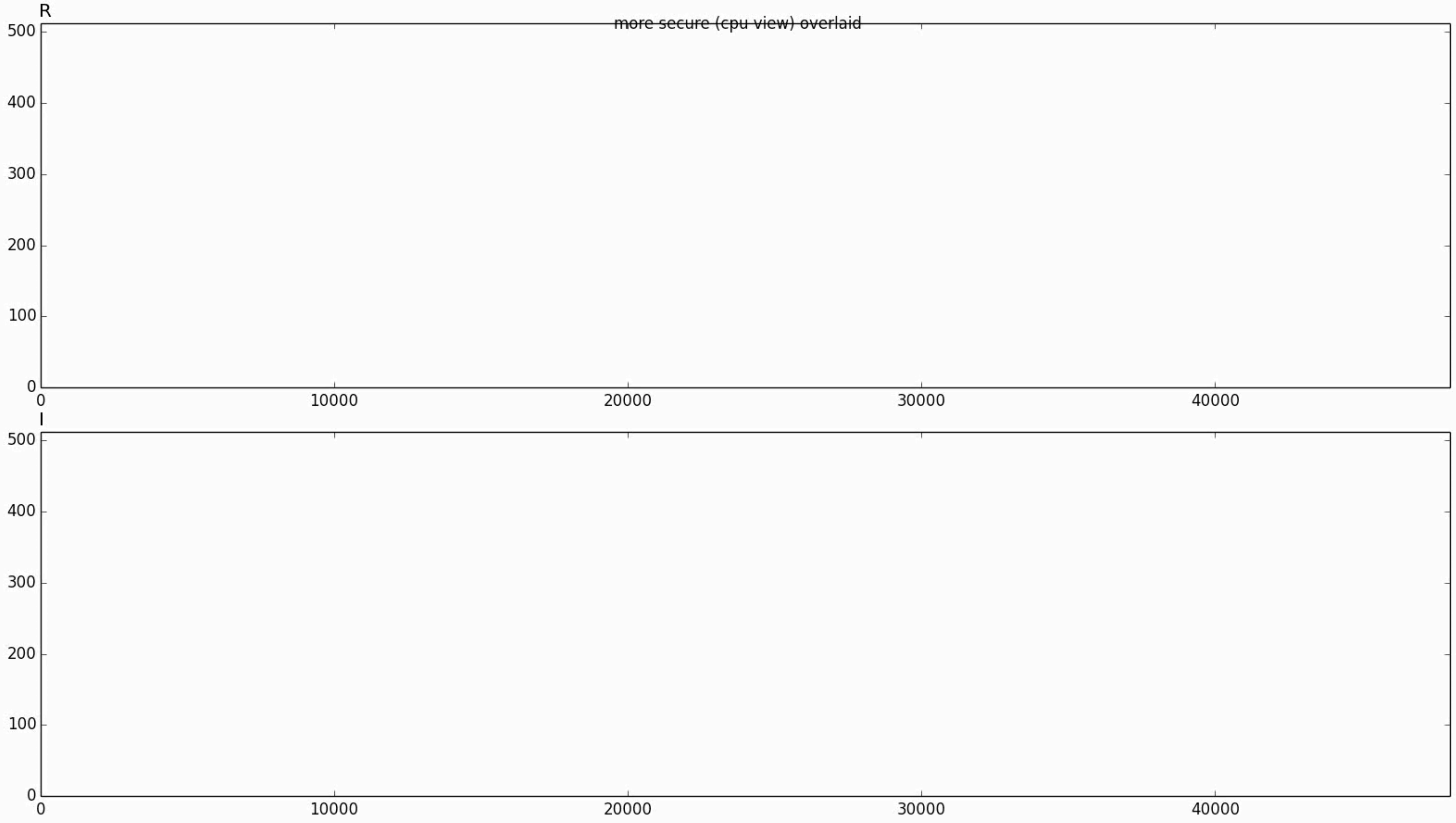


EXAMPLE: LIBGCRYPT ECC

EXAMPLE: LIBGCRYPT ECC

- More side channel proof version

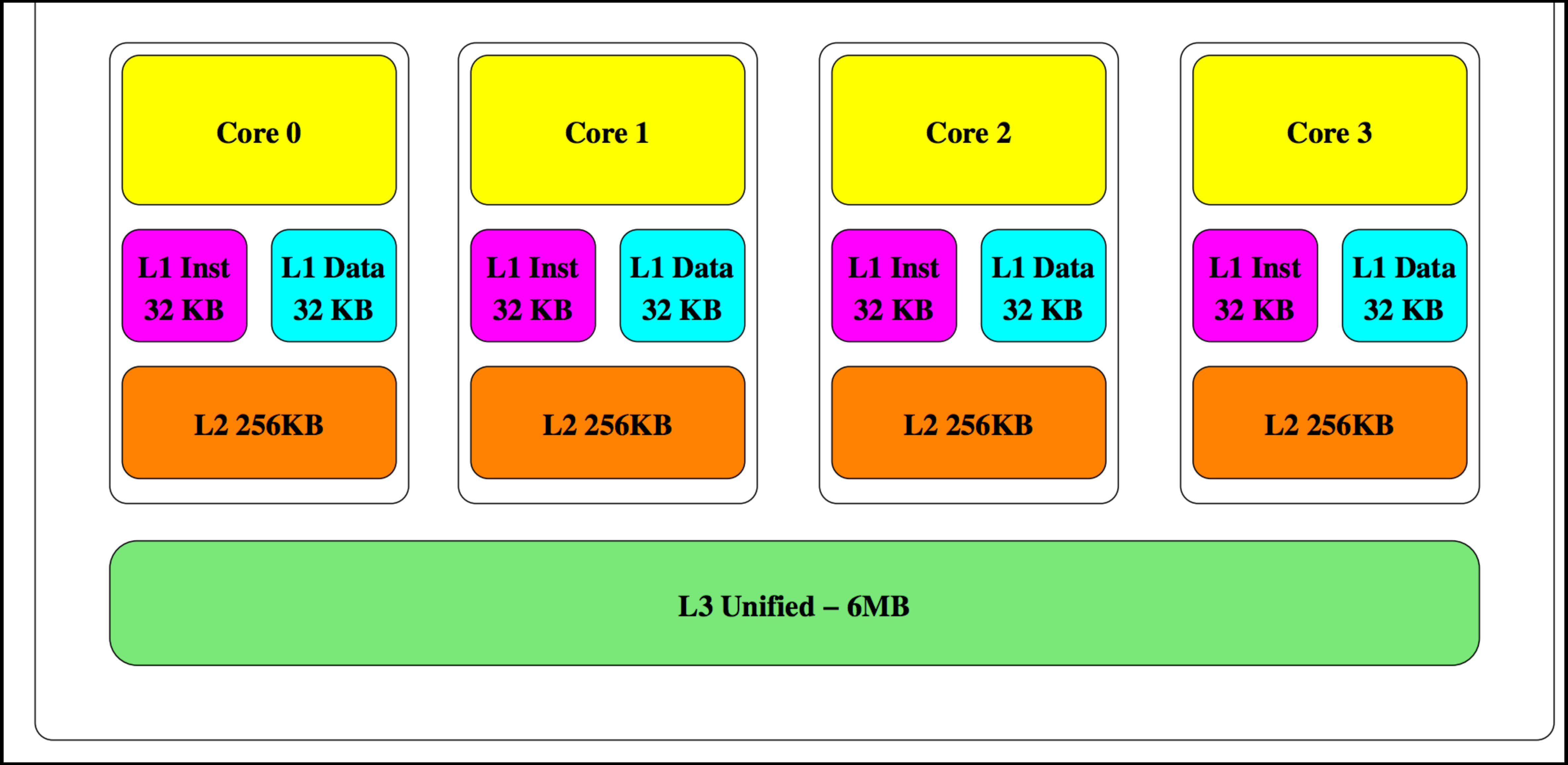
EXAMPLE: LIBGCRYPT ECC



CACHE DEFENCES

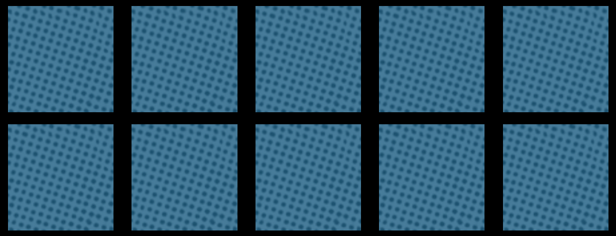
CACHE PARTIONING

CACHE PARTIONING



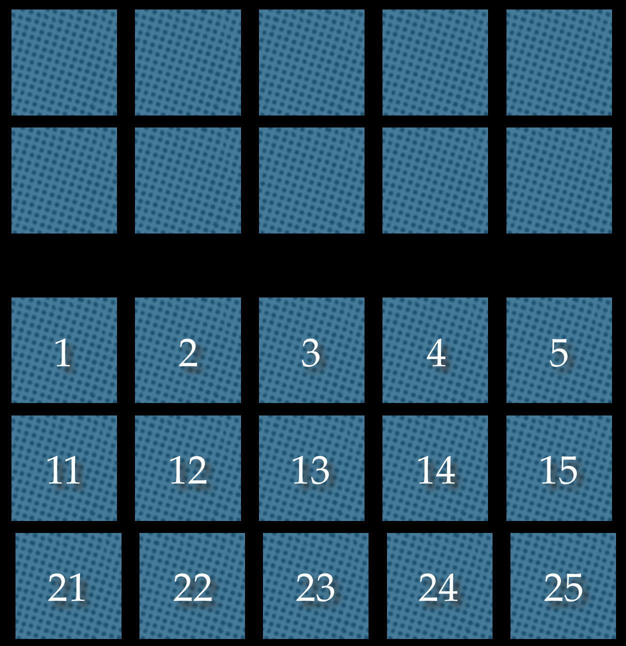
CACHE COLOURING

- Figure out page colors
- These map to shared cache sets
- Do not share same colors across security boundaries
- Kernel arranges this



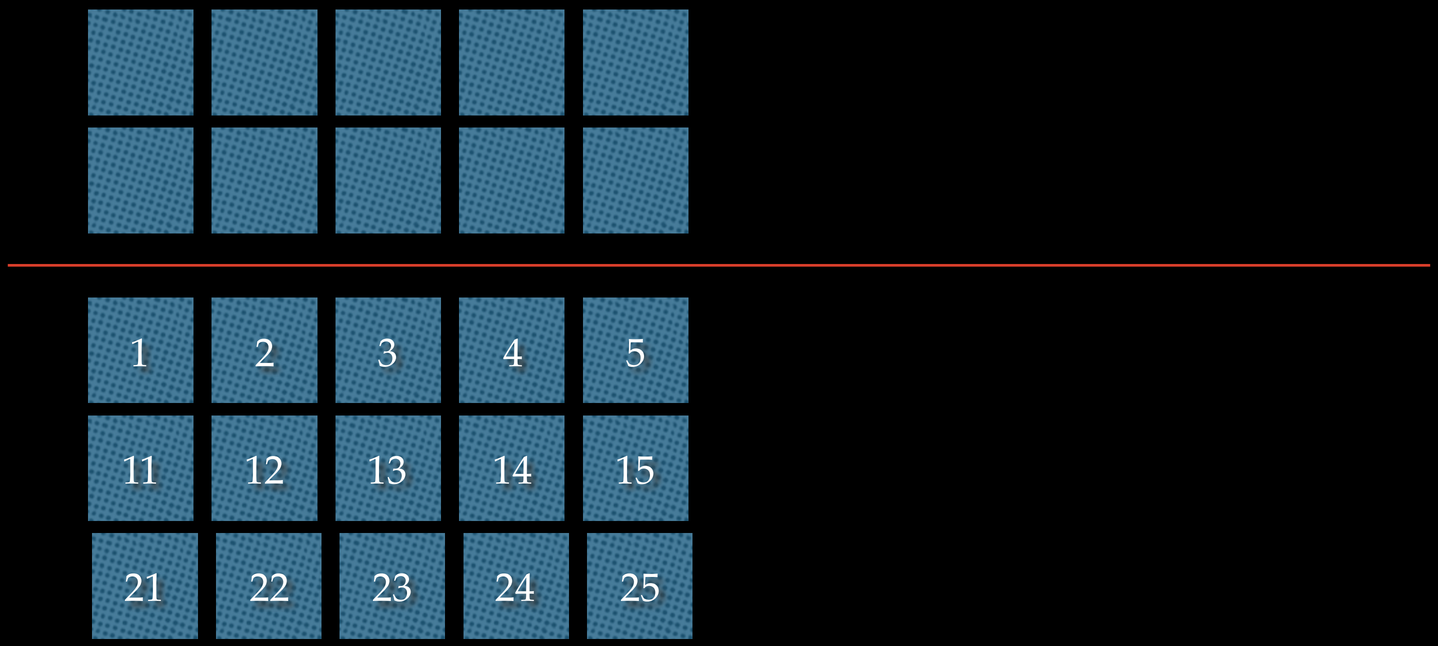
CACHE COLOURING

- Figure out page colors
- These map to shared cache sets
- Do not share same colors across security boundaries
- Kernel arranges this



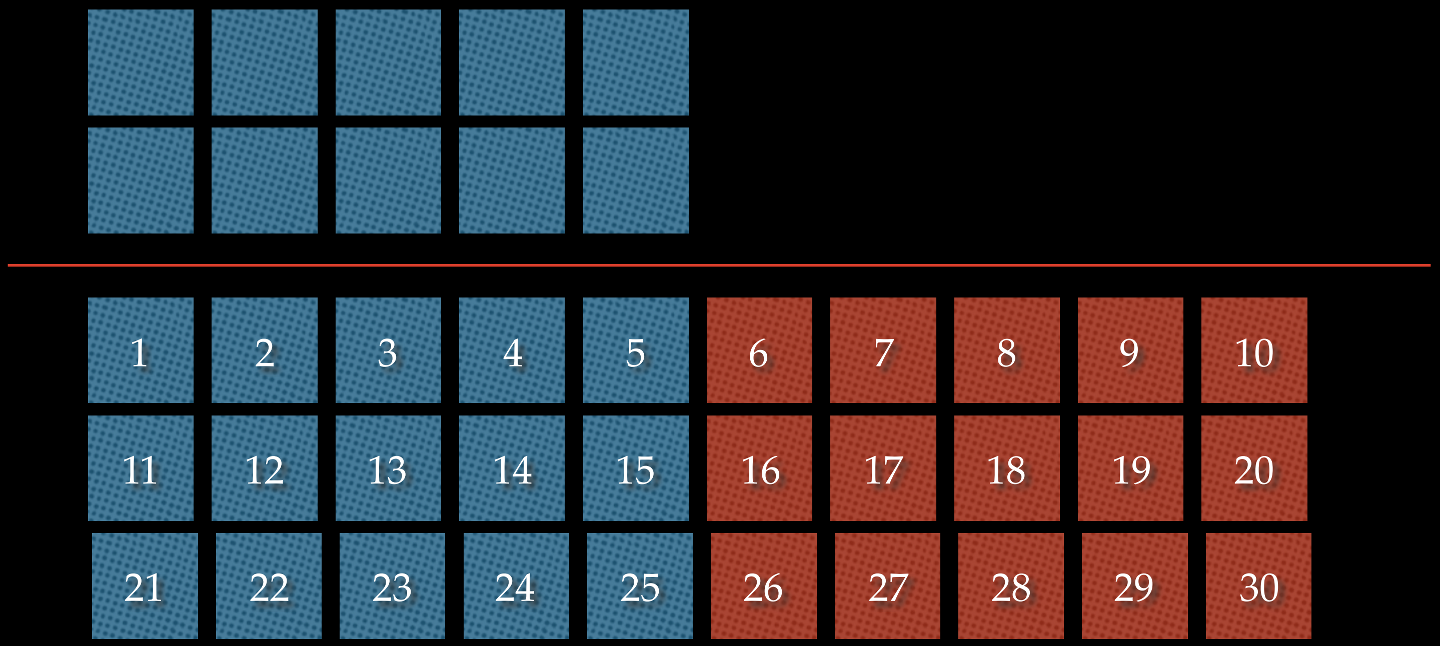
CACHE COLOURING

- Figure out page colors
- These map to shared cache sets
- Do not share same colors across security boundaries
- Kernel arranges this



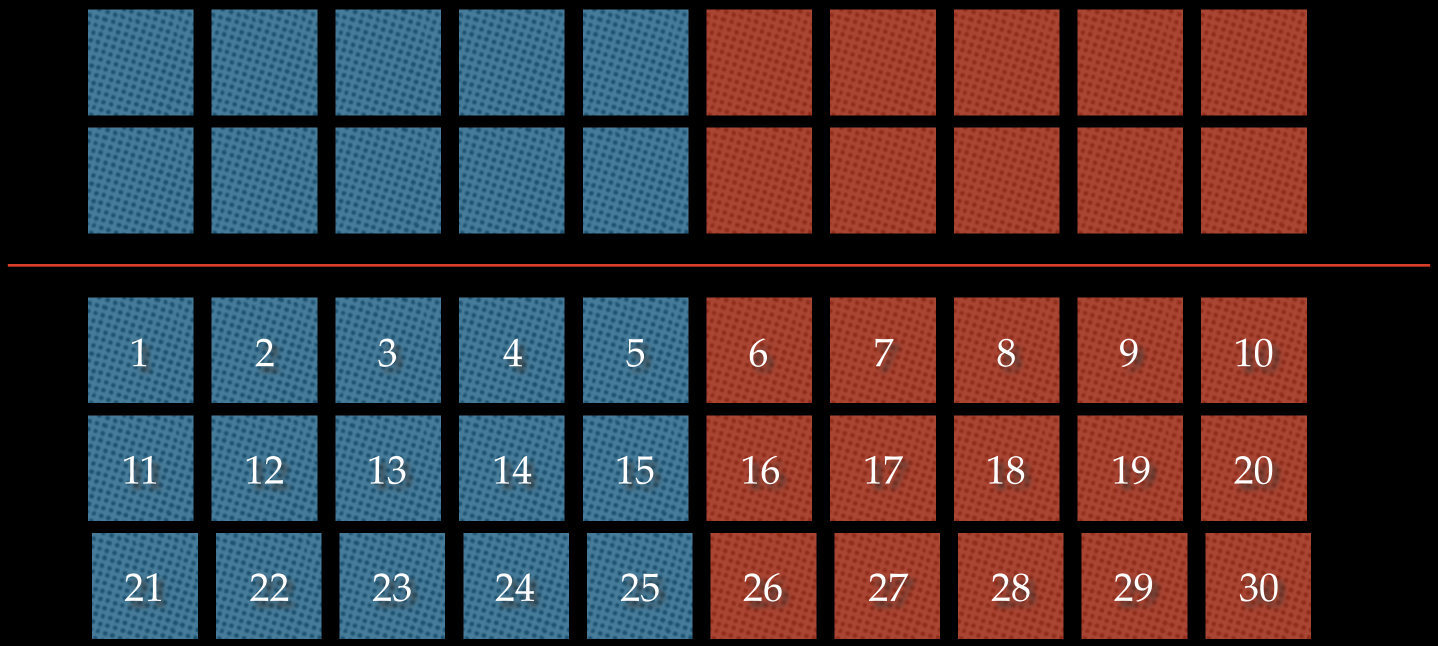
CACHE COLOURING

- Figure out page colors
- These map to shared cache sets
- Do not share same colors across security boundaries
- Kernel arranges this



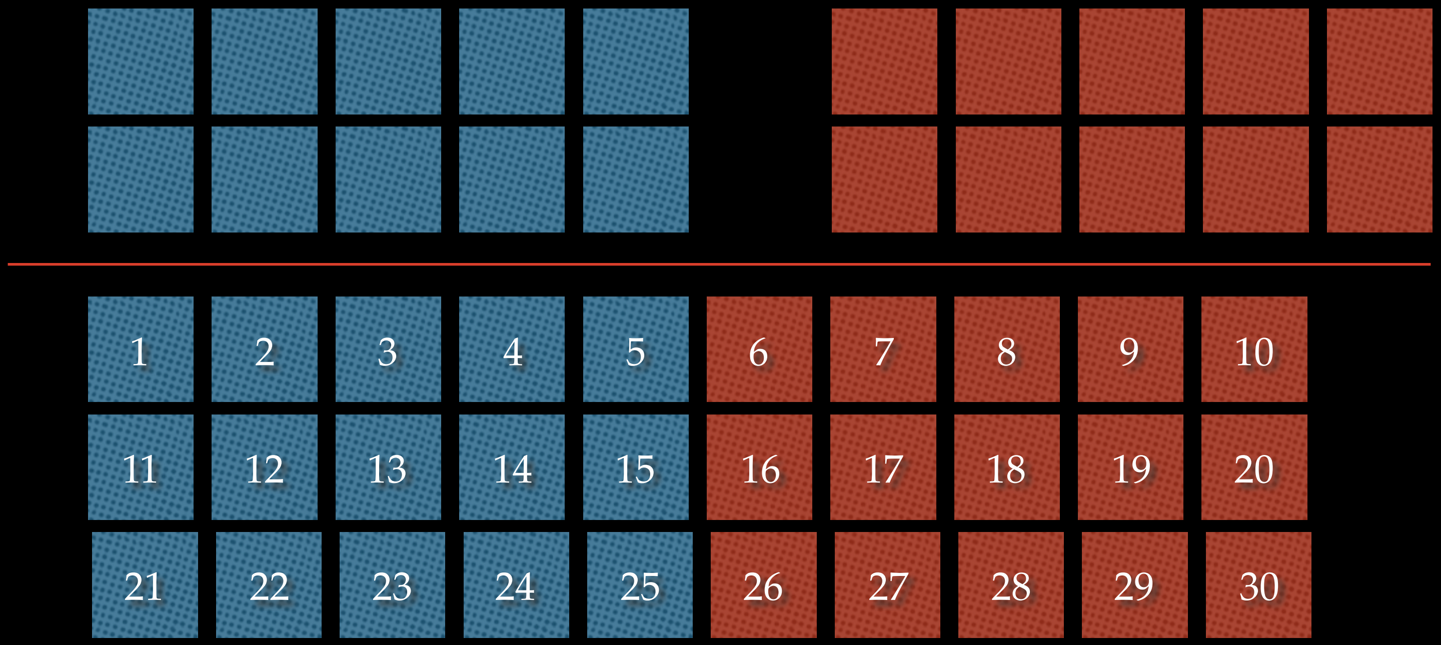
CACHE COLOURING

- Figure out page colors
- These map to shared cache sets
- Do not share same colors across security boundaries
- Kernel arranges this



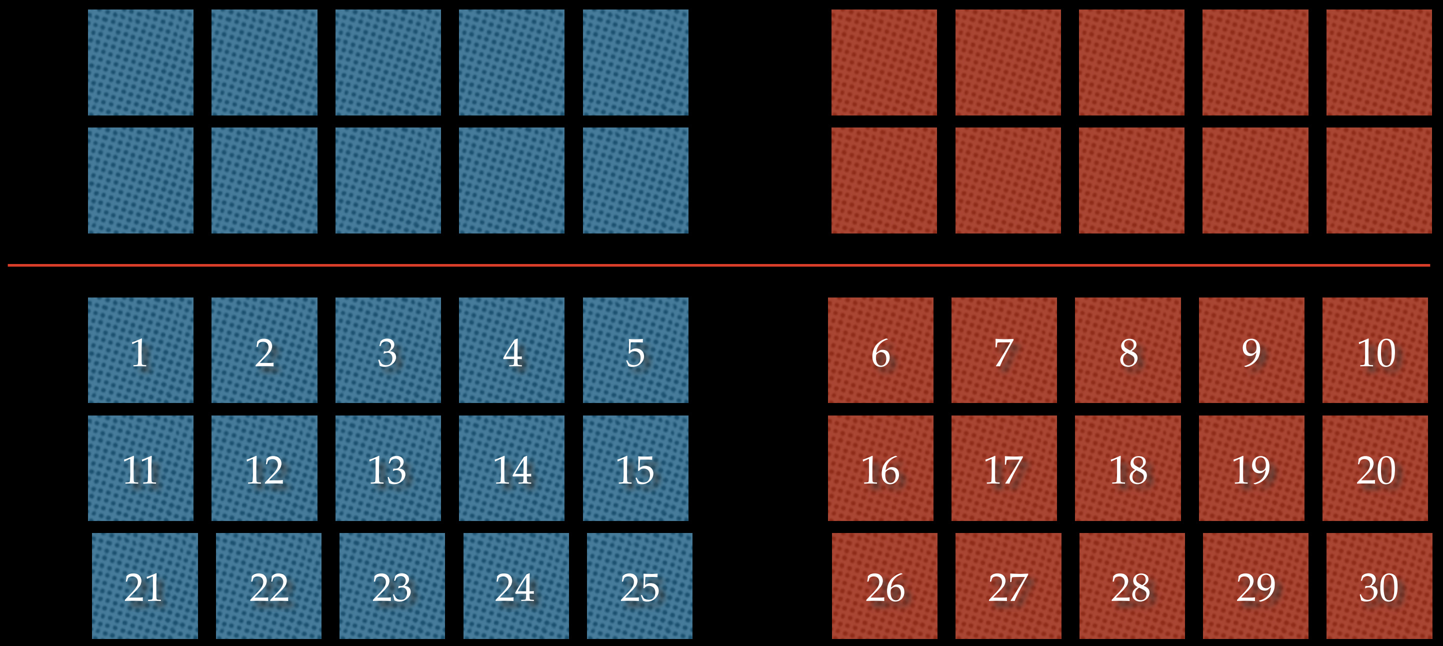
CACHE COLOURING

- Figure out page colors
- These map to shared cache sets
- Do not share same colors across security boundaries
- Kernel arranges this



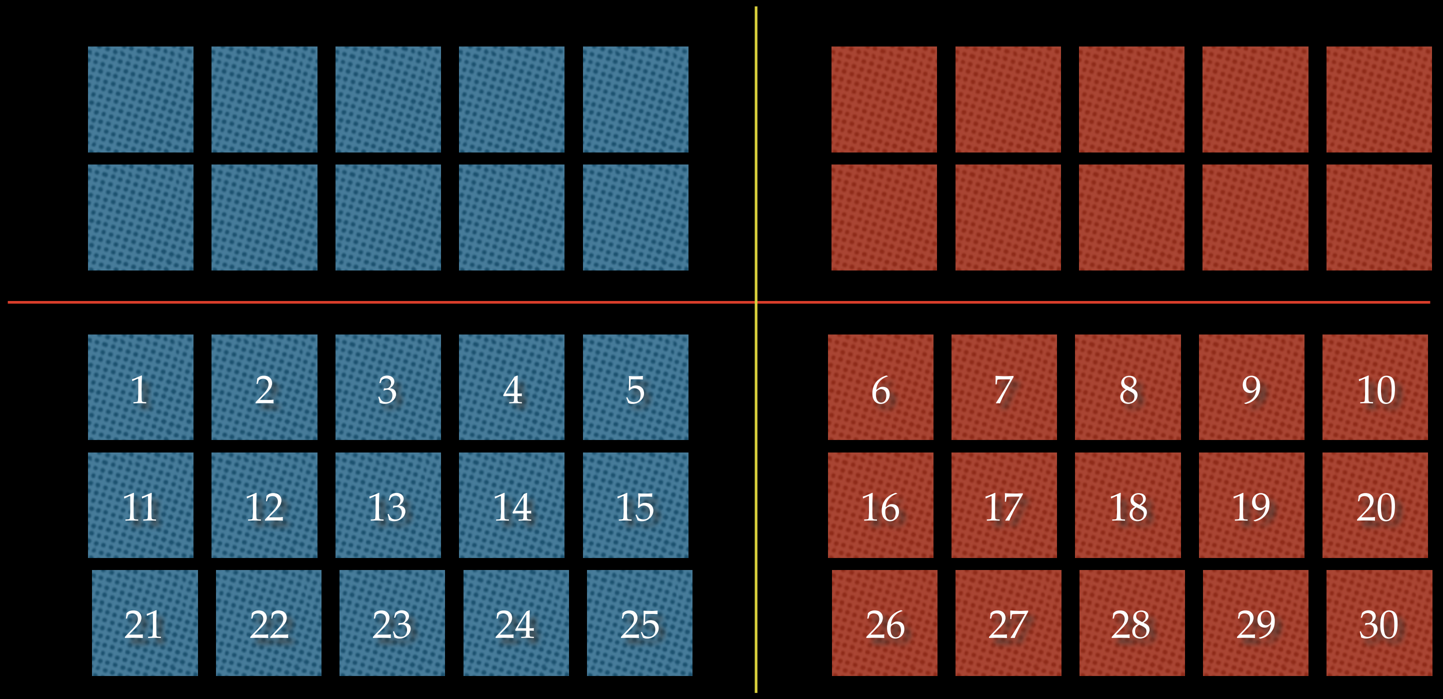
CACHE COLOURING

- Figure out page colors
- These map to shared cache sets
- Do not share same colors across security boundaries
- Kernel arranges this



CACHE COLOURING

- Figure out page colors
- These map to shared cache sets
- Do not share same colors across security boundaries
- Kernel arranges this

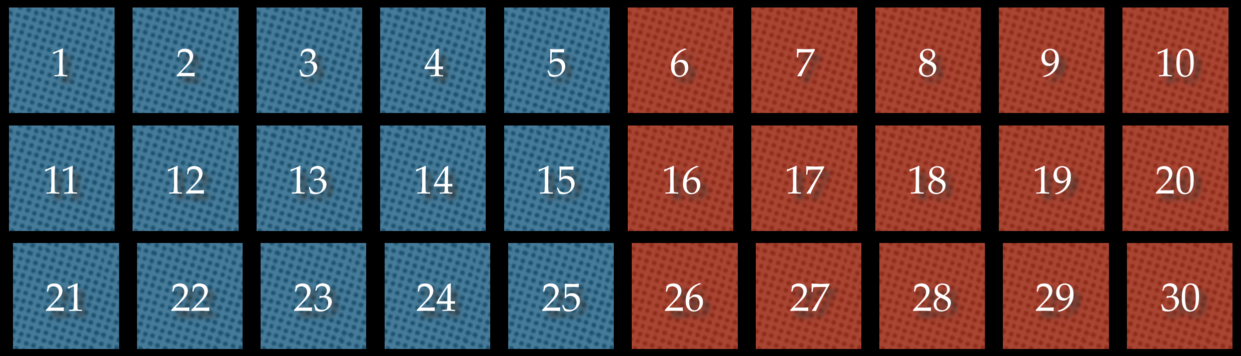


CACHE PARTITIONING: CAT

- Intel CAT: Cache Allocation Technology
- Intended for predictable performance for VMs
- Partitions caches in *ways*
- Hardware feature

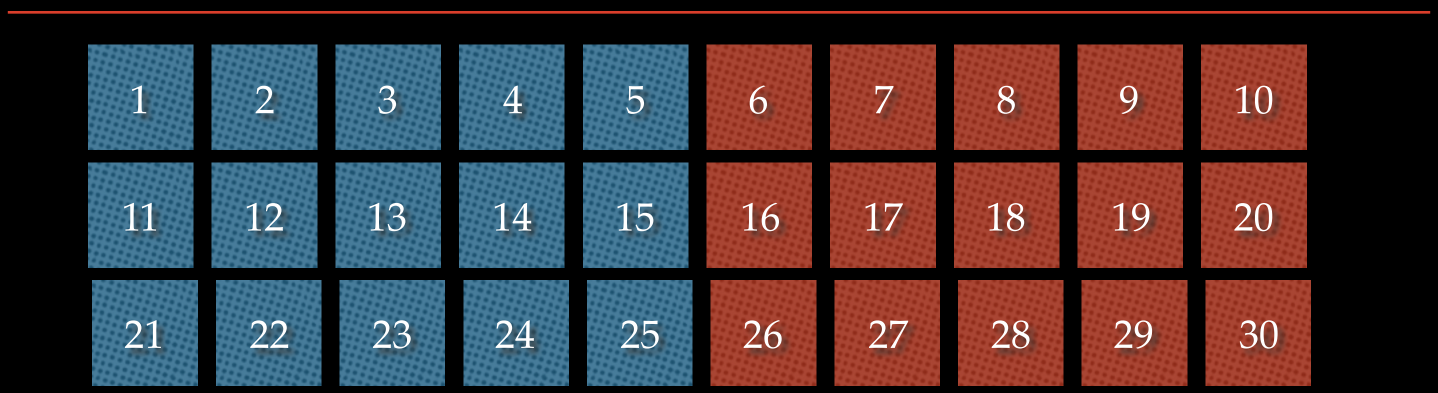
CACHE PARTITIONING: CAT

- Intel CAT: Cache Allocation Technology
- Intended for predictable performance for VMs
- Partitions caches in *ways*
- Hardware feature



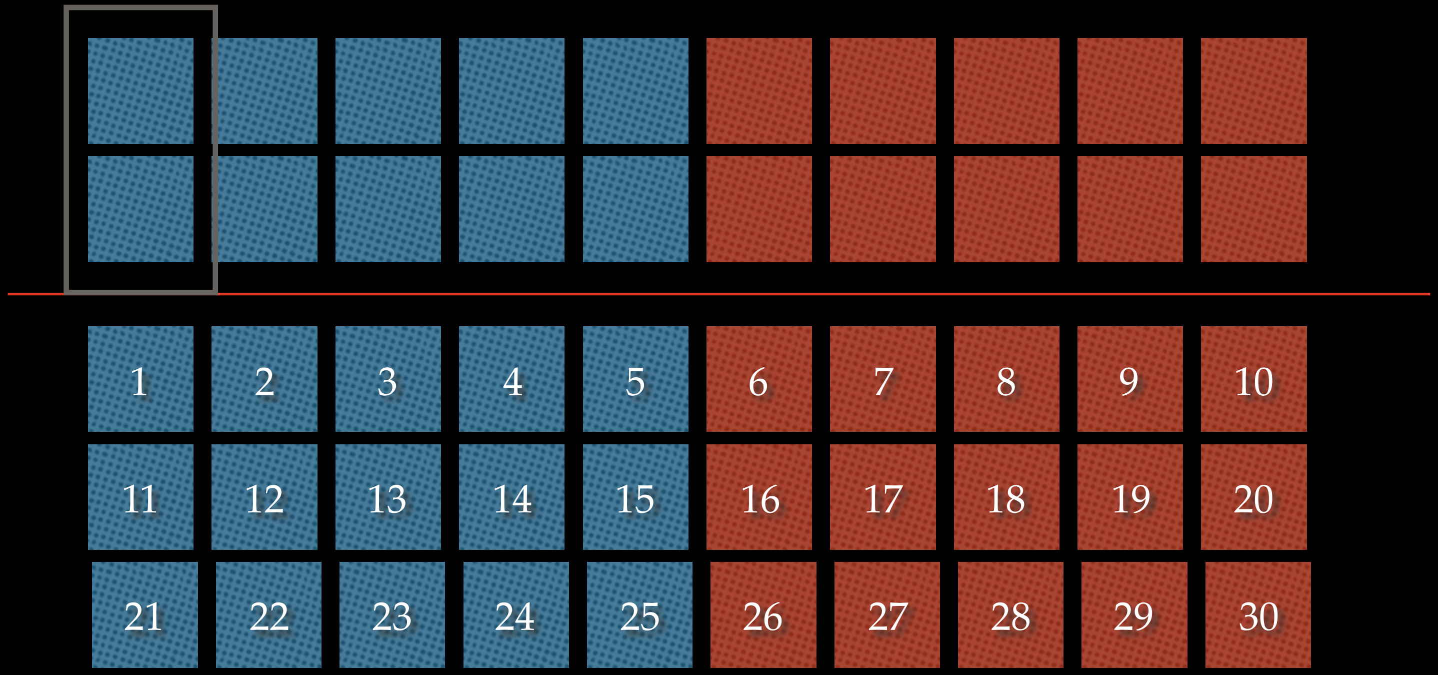
CACHE PARTITIONING: CAT

- Intel CAT: Cache Allocation Technology
- Intended for predictable performance for VMs
- Partitions caches in *ways*
- Hardware feature



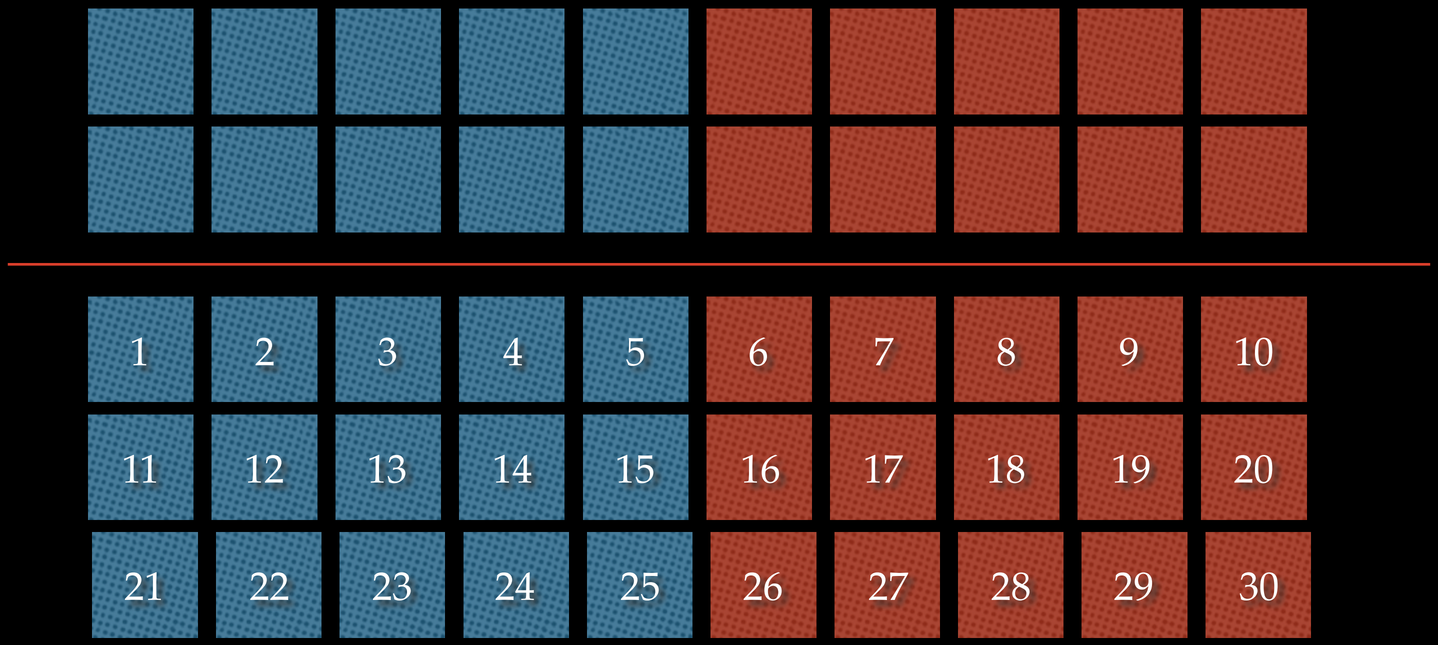
CACHE PARTITIONING: CAT

- Intel CAT: Cache Allocation Technology
- Intended for predictable performance for VMs
- Partitions caches in *ways*
- Hardware feature



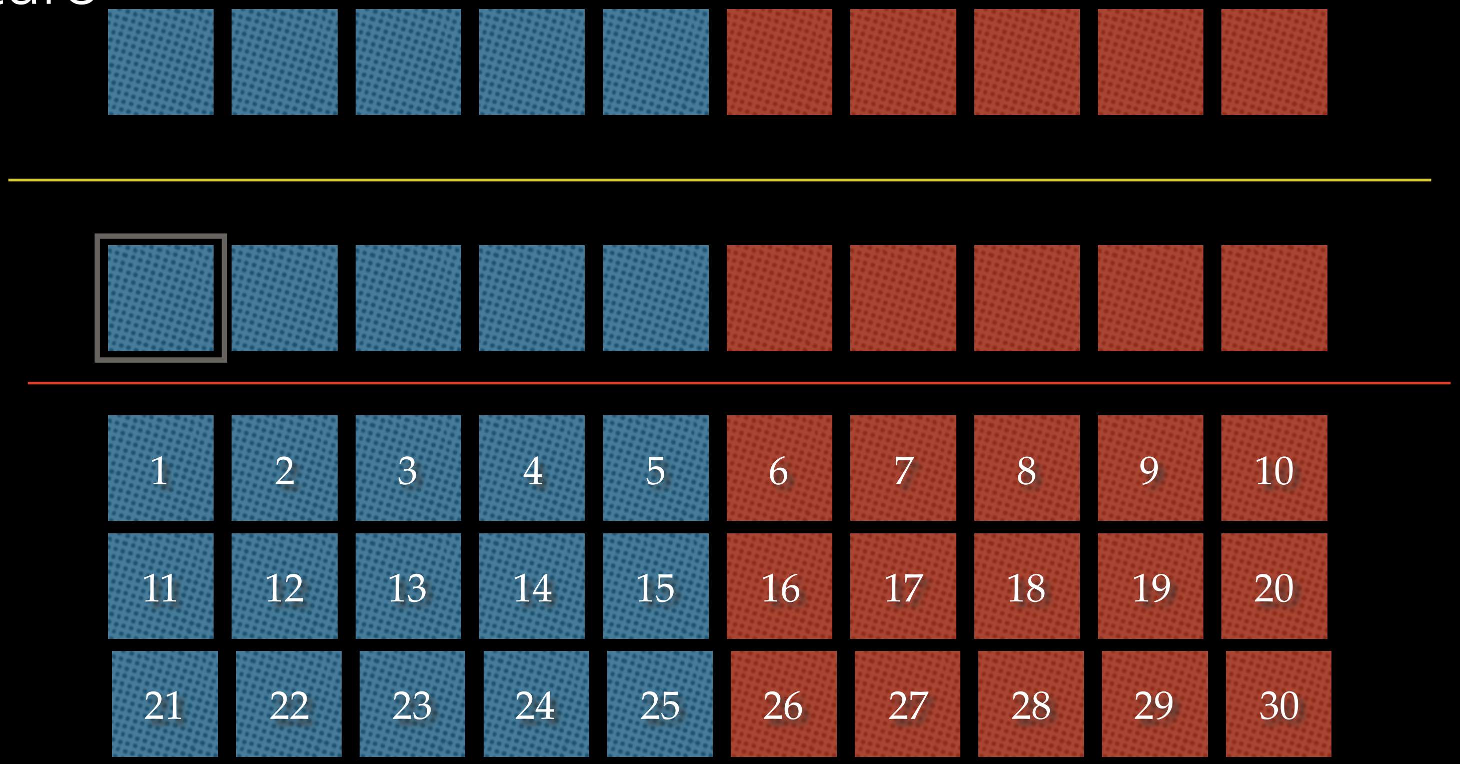
CACHE PARTITIONING: CAT

- Intel CAT: Cache Allocation Technology
- Intended for predictable performance for VMs
- Partitions caches in *ways*
- Hardware feature



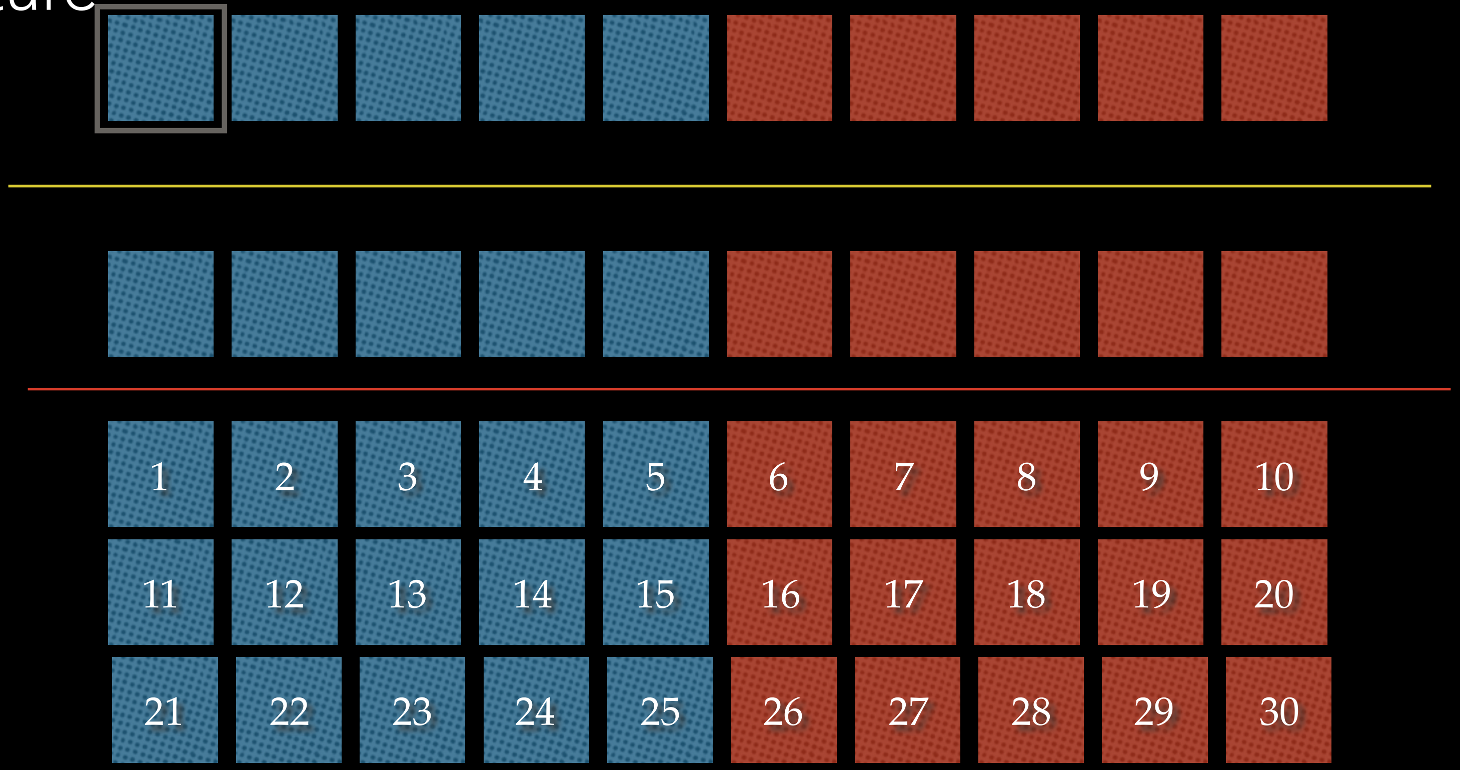
CACHE PARTITIONING: CAT

- Intel CAT: Cache Allocation Technology
- Intended for predictable performance for VMs
- Partitions caches in *ways*
- Hardware feature



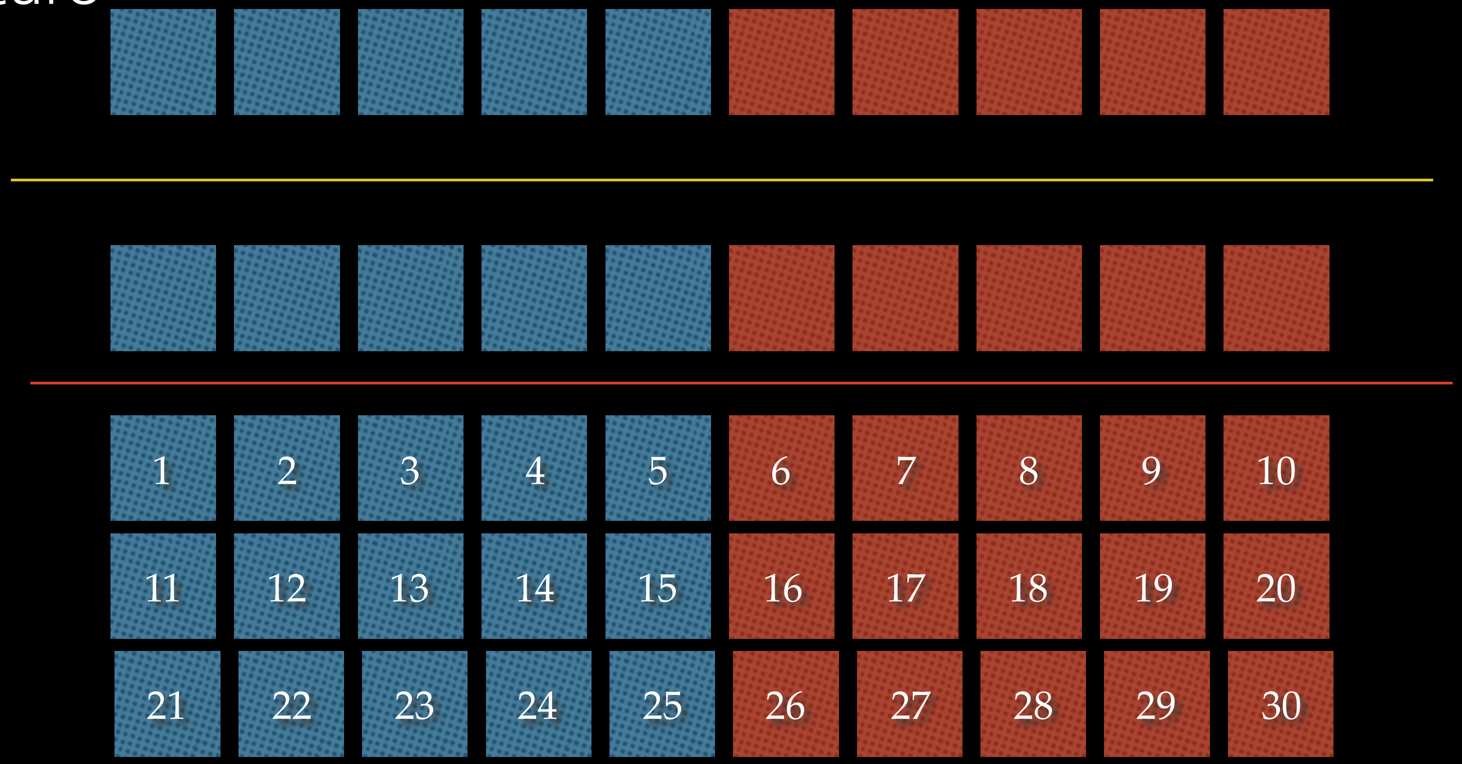
CACHE PARTITIONING: CAT

- Intel CAT: Cache Allocation Technology
- Intended for predictable performance for VMs
- Partitions caches in *ways*
- Hardware feature



CACHE PARTITIONING: CAT

- Intel CAT: Cache Allocation Technology
- Intended for predictable performance for VMs
- Partitions caches in *ways*
- Hardware feature



CACHE PARTITIONING: TSX

CACHE PARTITIONING: TSX

- Intel TSX: Transactional Synchronization Extensions

CACHE PARTITIONING: TSX

- Intel TSX: Transactional Synchronization Extensions
- Intended for hardware transactional memory

CACHE PARTITIONING: TSX

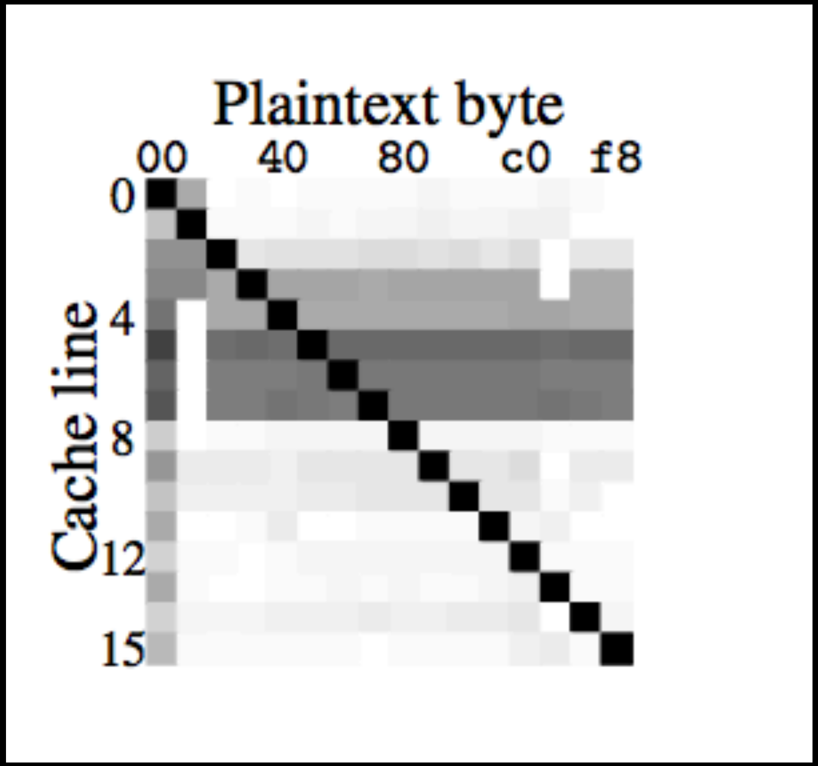
- Intel TSX: Transactional Synchronization Extensions
- Intended for hardware transactional memory
- Transaction working set should fit in cache, otherwise auto-abort

CACHE PARTITIONING: TSX

- Intel TSX: Transactional Synchronization Extensions
- Intended for hardware transactional memory
- Transaction working set should fit in cache, otherwise auto-abort
- We can use this as a defence

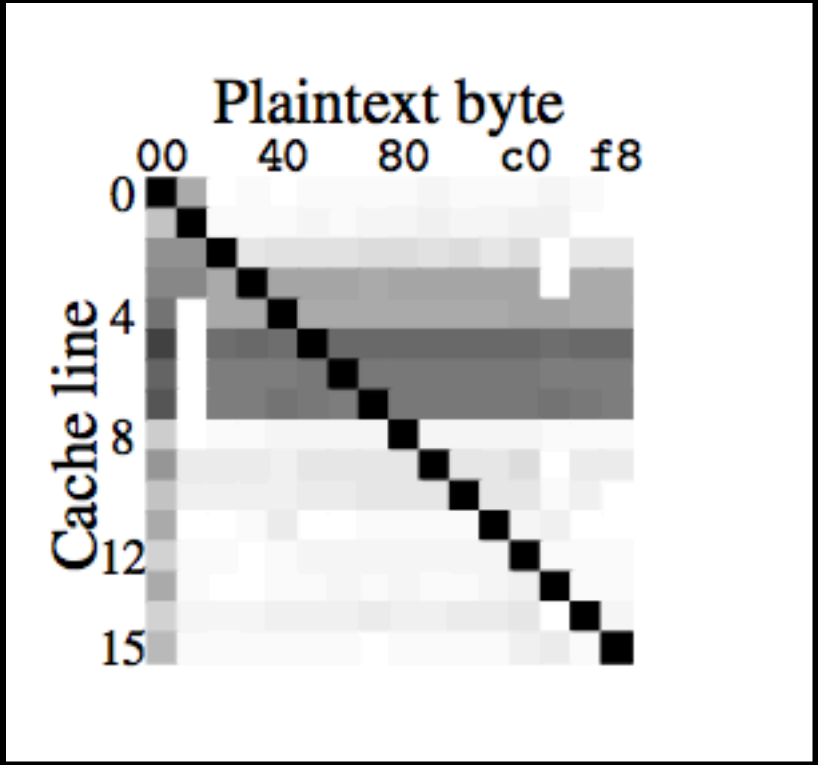
CACHE PARTITIONING: TSX

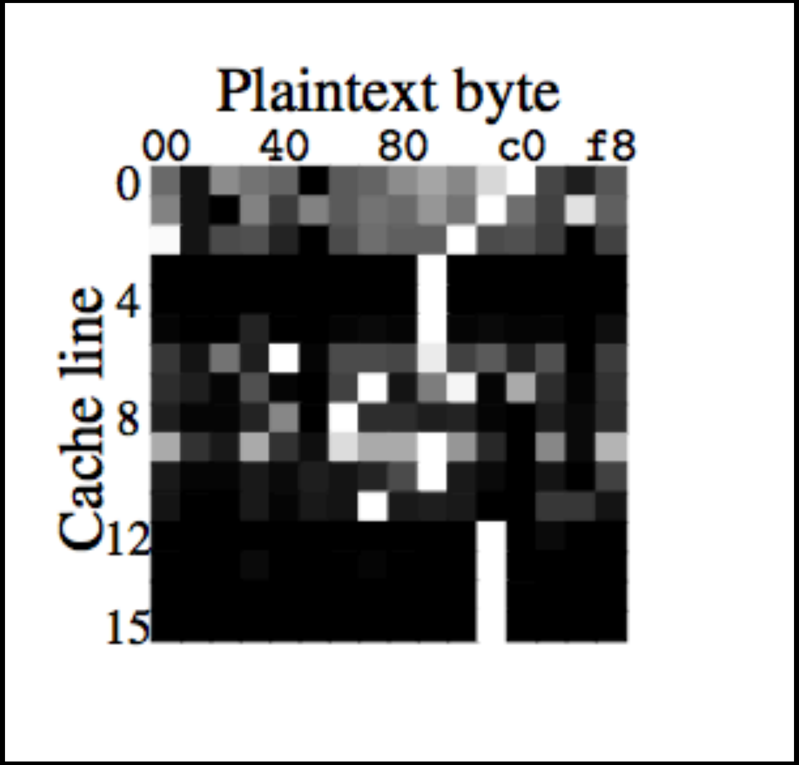
- Intel TSX: Transactional Synchronization Extensions
- Intended for hardware transactional memory
- Transaction working set should fit in cache, otherwise auto-abort
- We can use this as a defence



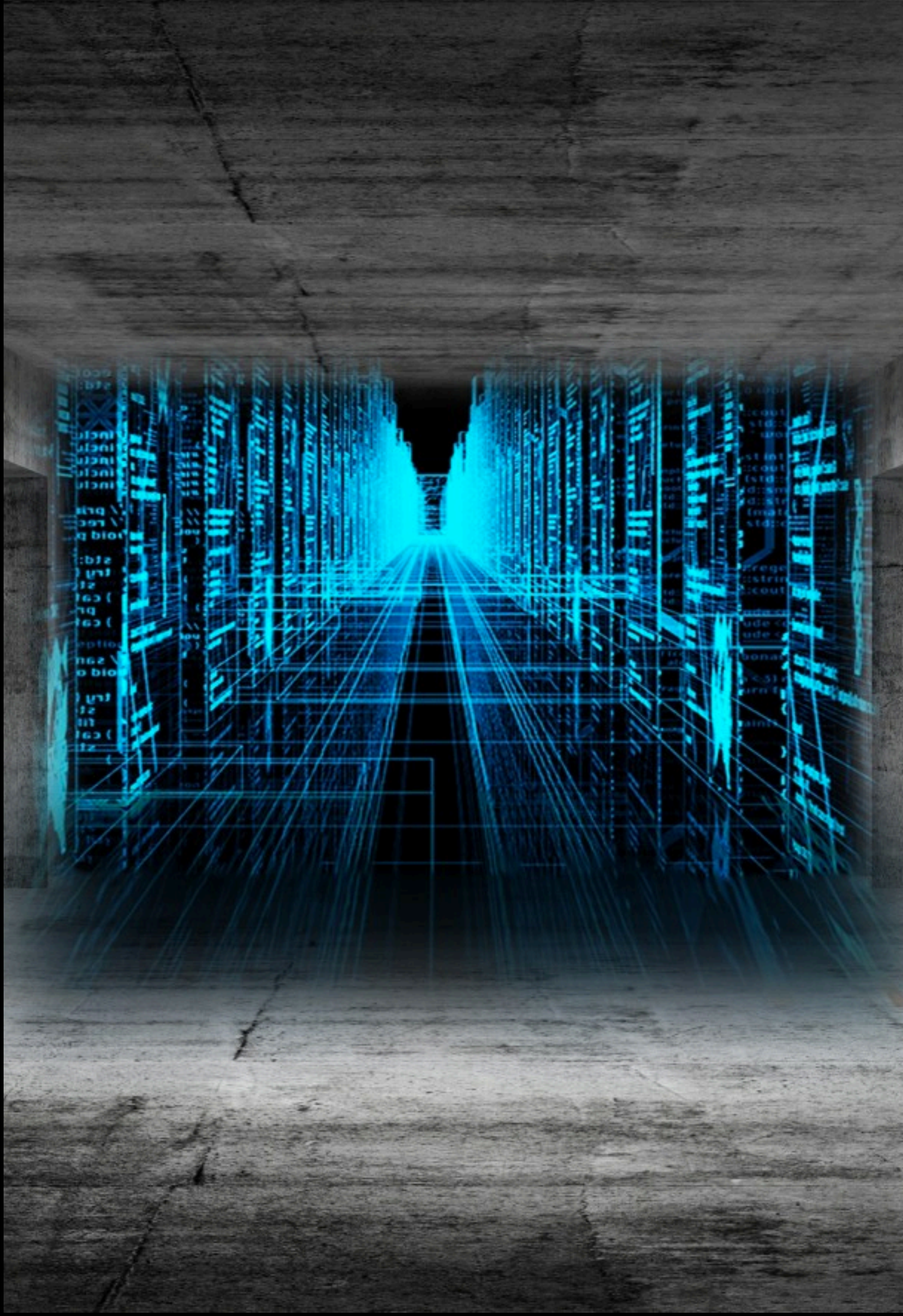
CACHE PARTITIONING: TSX

- Intel TSX: Transactional Synchronization Extensions
- Intended for hardware transactional memory
- Transaction working set should fit in cache, otherwise auto-abort
- We can use this as a defence





TLBLEED

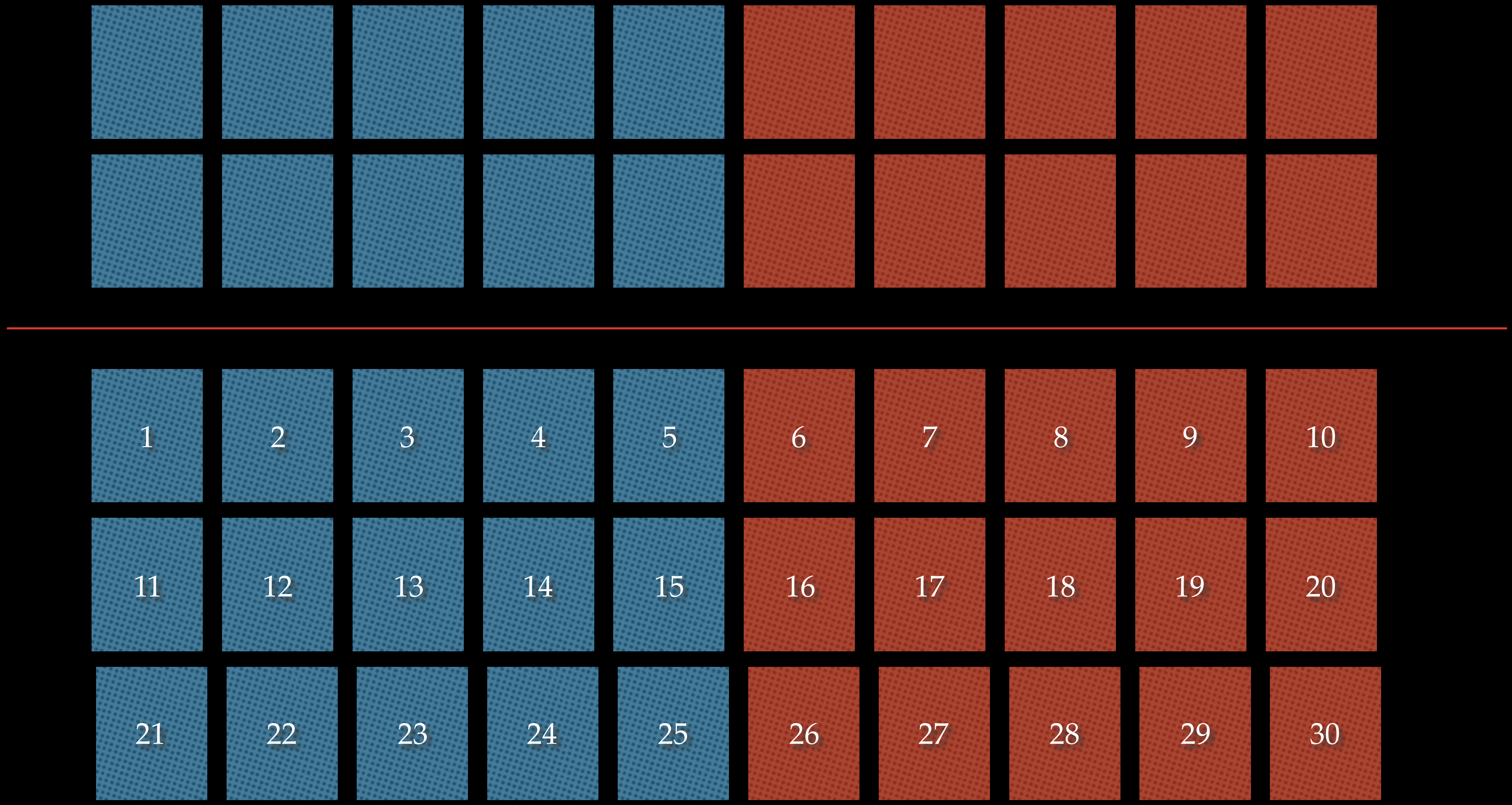


TLBLEED: TLB AS SHARED STATE

TLBLEED: TLB AS SHARED STATE

- Other structures than cache shared between threads?
- What about the TLB?
- Documented: TLB has L1iTLB, L1dTLB, and L2TLB
- They have sets and ways
- Not documented: structure

TLB IS JUST ANOTHER CACHE



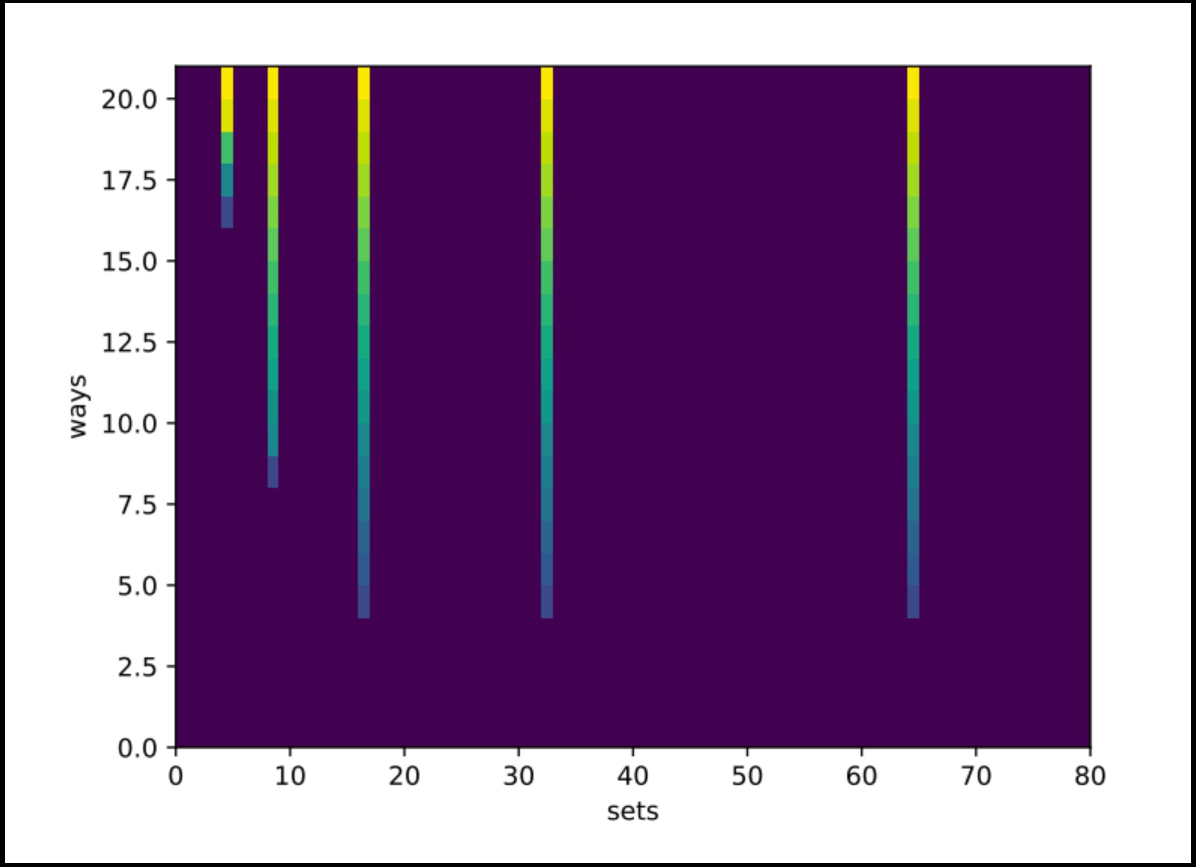
TLBLEED: TLB AS SHARED STATE

TLBLEED: TLB AS SHARED STATE

- Let’s experiment with performance counters
- Try linear structure first
- All combinations of ways (set size) and sets (stride)
- Smallest number of ways is it
- Smallest corresponding stride is number of sets

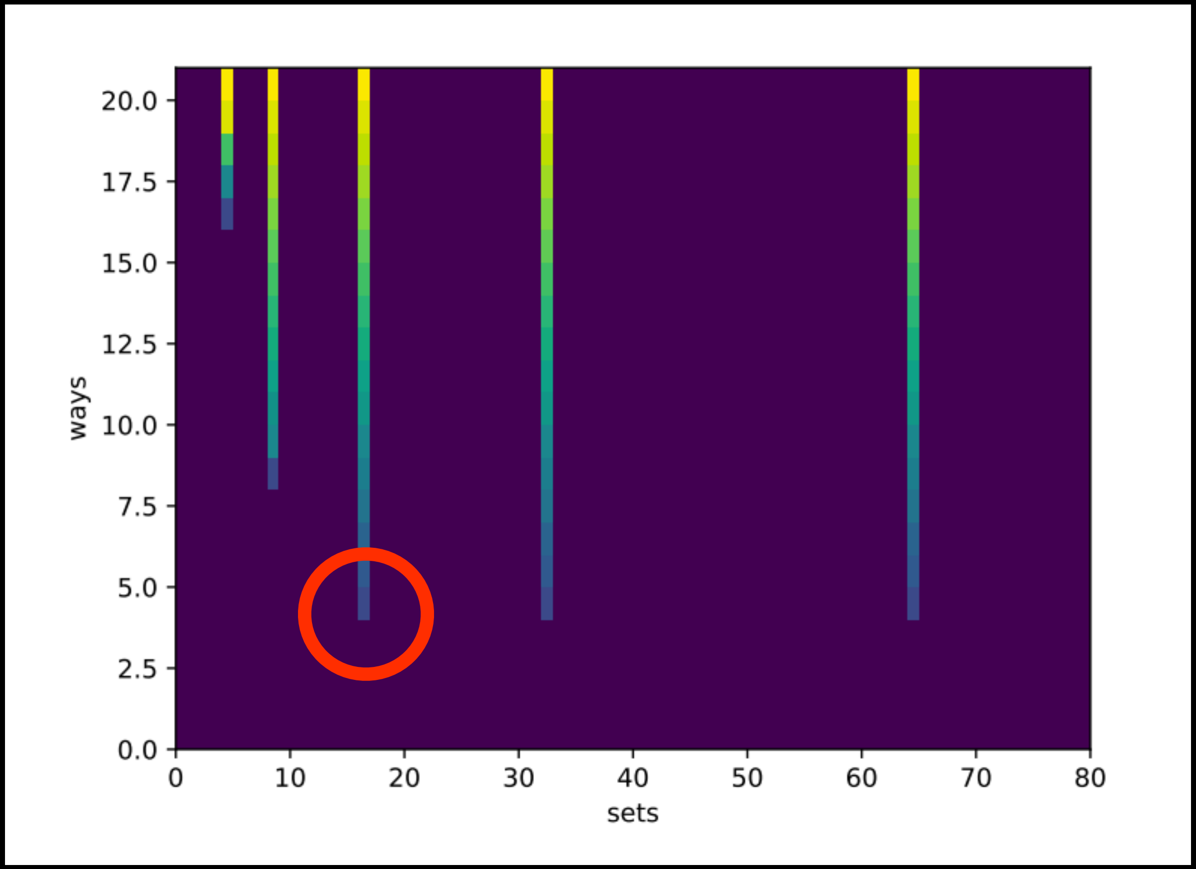
TLBLEED: TLB AS SHARED STATE

- Let's experiment with performance counters
- Try linear structure first
- All combinations of ways (set size) and sets (stride)
- Smallest number of ways is it
- Smallest corresponding stride is number of sets



TLBLEED: TLB AS SHARED STATE

- Let's experiment with performance counters
- Try linear structure first
- All combinations of ways (set size) and sets (stride)
- Smallest number of ways is it
- Smallest corresponding stride is number of sets



TLBLEED: TLB AS SHARED STATE

TLBLEED: TLB AS SHARED STATE

- For L2TLB:
  We reverse engineered a more complex hash function

TLBLEED: TLB AS SHARED STATE

- For L2TLB:
  We reverse engineered a more complex hash function

- Skylake XORs 14 bits, Broadwell XORs 16 bits

TLBLEED: TLB AS SHARED STATE

- For L2TLB:
  We reverse engineered a more complex hash function

- Skylake XORs 14 bits, Broadwell XORs 16 bits

- Represented by this matrix, using modulo 2 arithmetic

TLBLEED: TLB AS SHARED STATE

- For L2TLB:
  We reverse engineered a more complex hash function
- Skylake XORs 14 bits, Broadwell XORs 16 bits
- Represented by this matrix, using modulo 2 arithmetic

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

TLBLEED: TLB AS SHARED STATE

TLBLEED: TLB AS SHARED STATE

- Let’s experiment with performance counters

TLBLEED: TLB AS SHARED STATE

- Let’s experiment with performance counters

- Now we know the structure..
  Are TLB’s shared between hyperthreads?

TLBLEED: TLB AS SHARED STATE

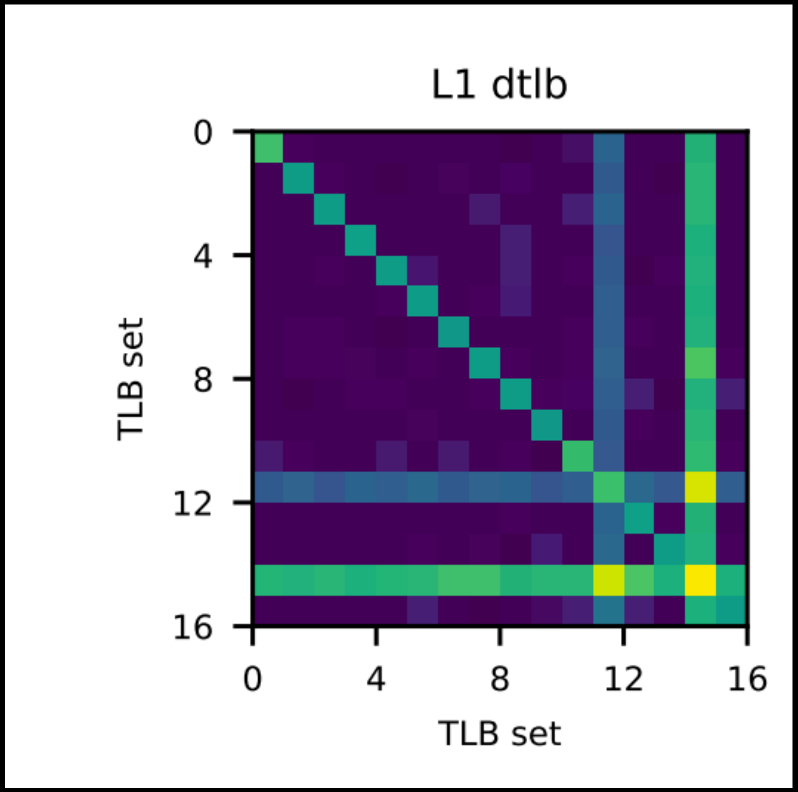
- Let’s experiment with performance counters

- Now we know the structure..
  Are TLB’s shared between hyperthreads?

- Let’s experiment with misses when accessing the same set

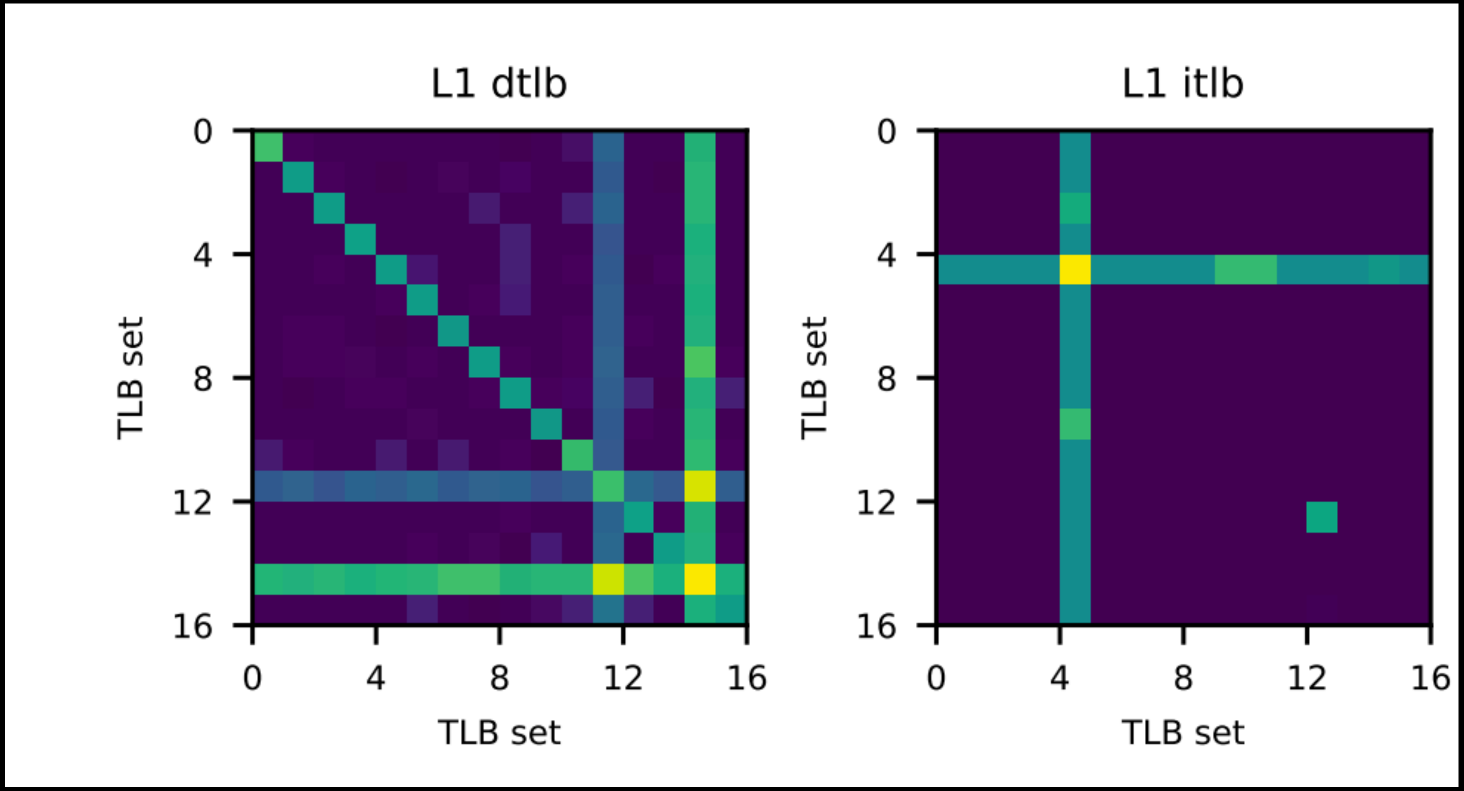
TLBLEED: TLB AS SHARED STATE

- Let's experiment with performance counters
- Now we know the structure..
  Are TLB's shared between hyperthreads?
- Let's experiment with misses when accessing the same set



TLBLEED: TLB AS SHARED STATE

- Let’s experiment with performance counters
- Now we know the structure..
  Are TLB’s shared between hyperthreads?
- Let’s experiment with misses when accessing the same set

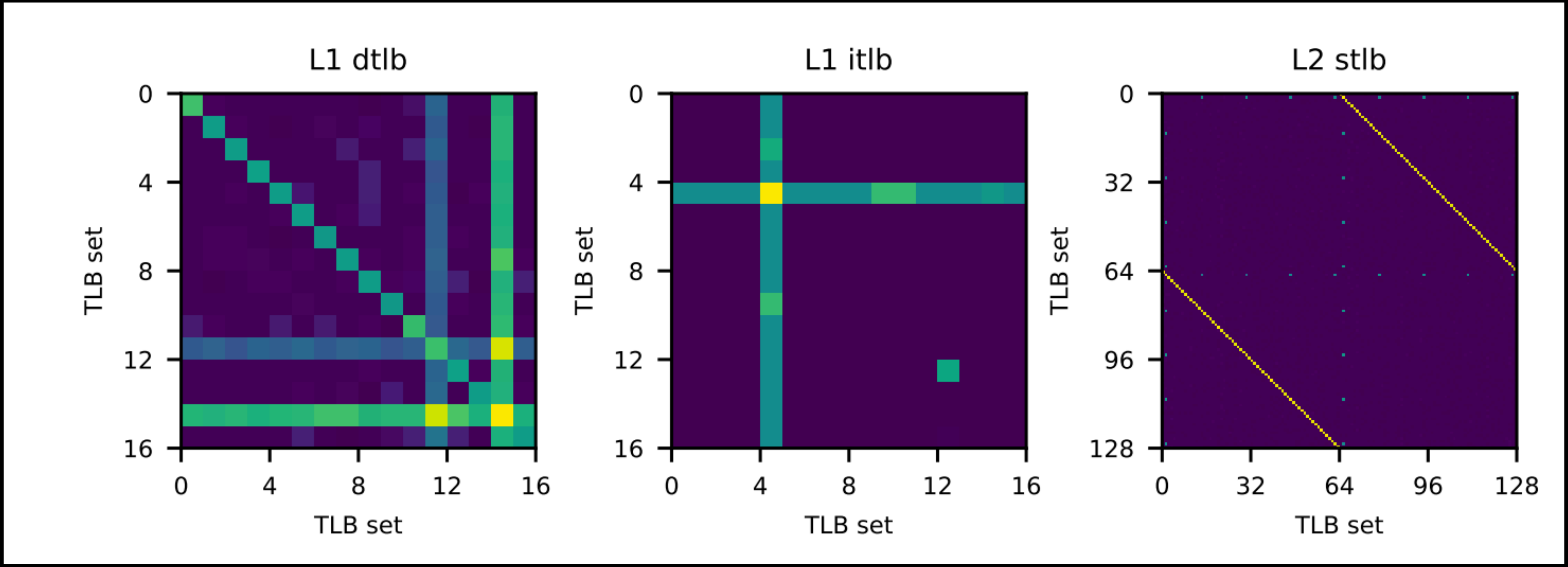


TLBLEED: TLB AS SHARED STATE

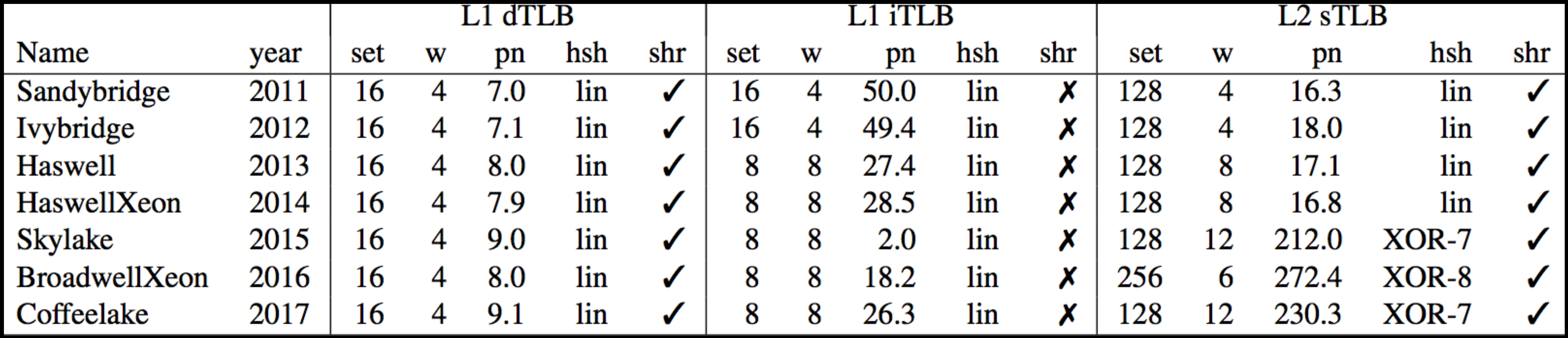
- Let’s experiment with performance counters

- Now we know the structure..
  Are TLB’s shared between hyperthreads?

- Let’s experiment with misses when accessing the same set



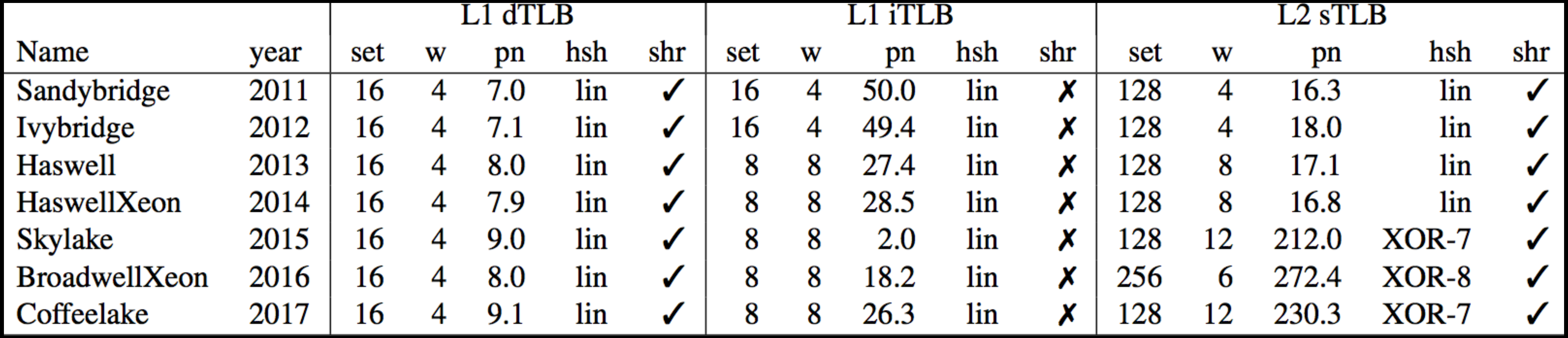
TLBLEED: TLB AS SHARED STATE



		L1 dTLB					L1 iTLB					L2 sTLB				
Name	year	set	w	pn	hsh	shr	set	w	pn	hsh	shr	set	w	pn	hsh	shr
Sandybridge	2011	16	4	7.0	lin	✓	16	4	50.0	lin	✗	128	4	16.3	lin	✓
Ivybridge	2012	16	4	7.1	lin	✓	16	4	49.4	lin	✗	128	4	18.0	lin	✓
Haswell	2013	16	4	8.0	lin	✓	8	8	27.4	lin	✗	128	8	17.1	lin	✓
HaswellXeon	2014	16	4	7.9	lin	✓	8	8	28.5	lin	✗	128	8	16.8	lin	✓
Skylake	2015	16	4	9.0	lin	✓	8	8	2.0	lin	✗	128	12	212.0	XOR-7	✓
BroadwellXeon	2016	16	4	8.0	lin	✓	8	8	18.2	lin	✗	256	6	272.4	XOR-8	✓
Coffeelake	2017	16	4	9.1	lin	✓	8	8	26.3	lin	✗	128	12	230.3	XOR-7	✓

TLBLEED: TLB AS SHARED STATE

- We find more TLB properties
- Size, structure, sharing, miss penalty, hash function



		L1 dTLB					L1 iTLB					L2 sTLB				
Name	year	set	w	pn	hsh	shr	set	w	pn	hsh	shr	set	w	pn	hsh	shr
Sandybridge	2011	16	4	7.0	lin	✓	16	4	50.0	lin	✗	128	4	16.3	lin	✓
Ivybridge	2012	16	4	7.1	lin	✓	16	4	49.4	lin	✗	128	4	18.0	lin	✓
Haswell	2013	16	4	8.0	lin	✓	8	8	27.4	lin	✗	128	8	17.1	lin	✓
HaswellXeon	2014	16	4	7.9	lin	✓	8	8	28.5	lin	✗	128	8	16.8	lin	✓
Skylake	2015	16	4	9.0	lin	✓	8	8	2.0	lin	✗	128	12	212.0	XOR-7	✓
BroadwellXeon	2016	16	4	8.0	lin	✓	8	8	18.2	lin	✗	256	6	272.4	XOR-8	✓
Coffeelake	2017	16	4	9.1	lin	✓	8	8	26.3	lin	✗	128	12	230.3	XOR-7	✓

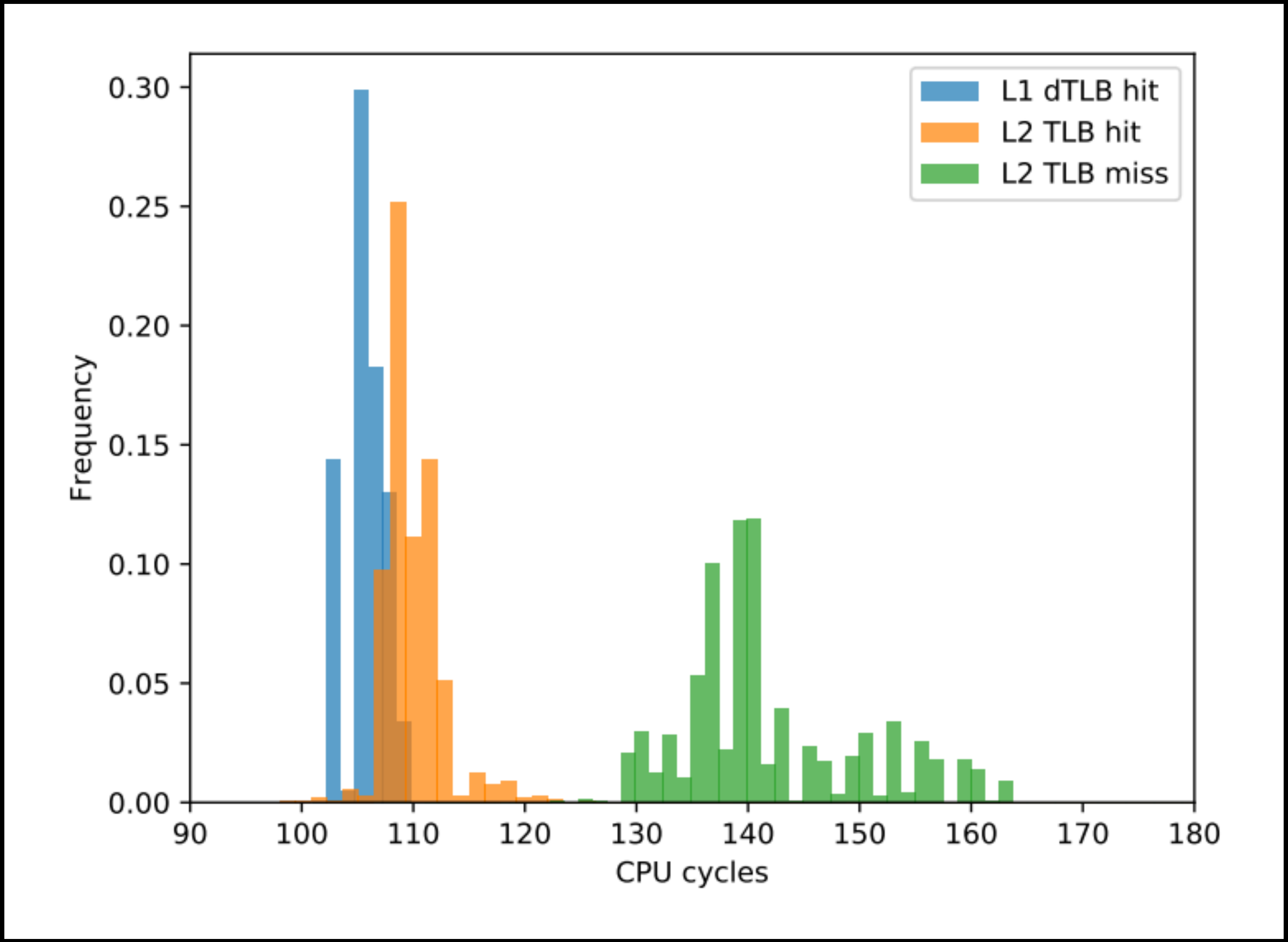
TLBLEED: TLB AS SHARED STATE

TLBLEED: TLB AS SHARED STATE

- Can we use only latency?
- Map many virtual addresses to same physical page

TLBLEED: TLB AS SHARED STATE

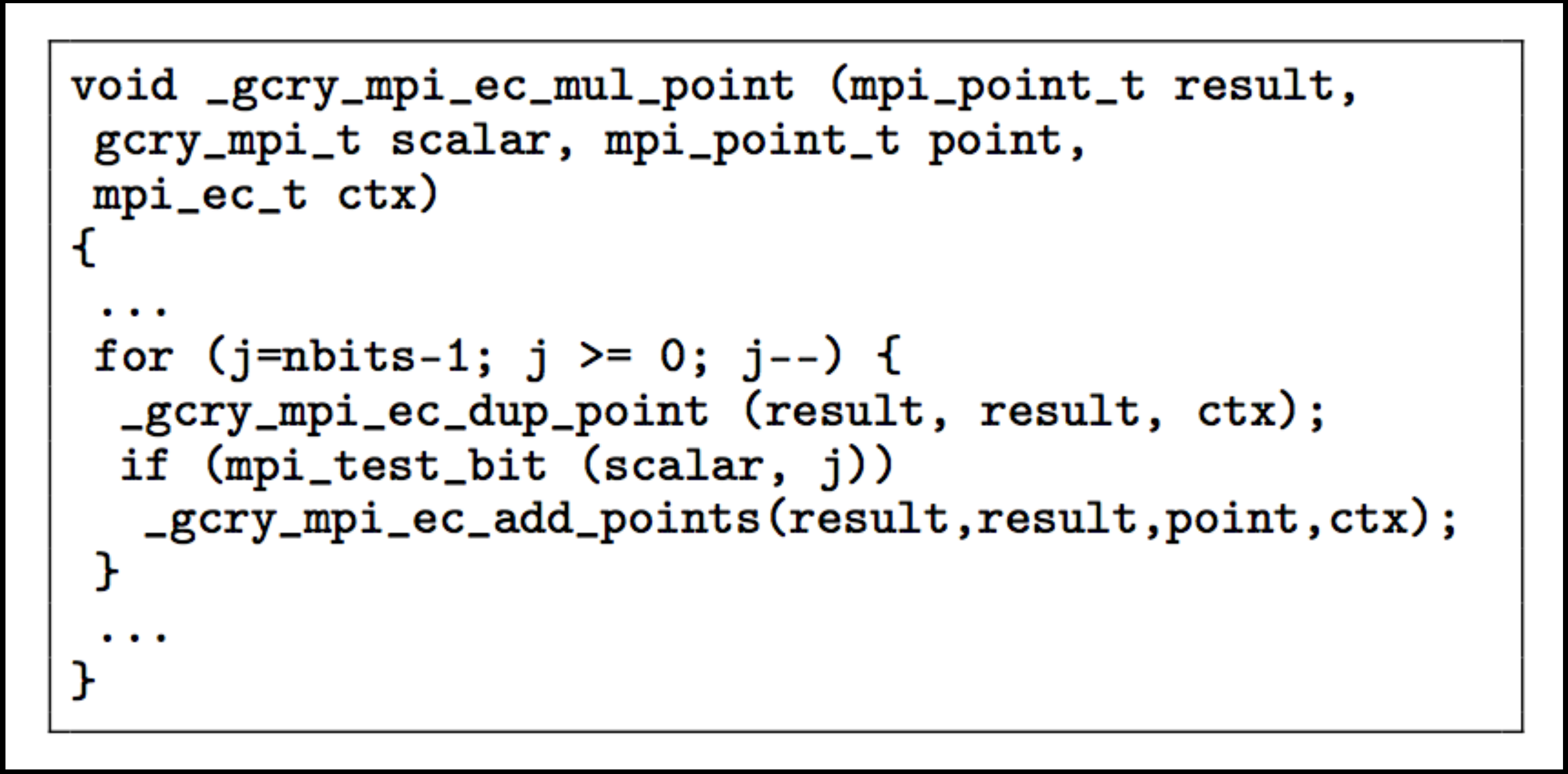
- Can we use only latency?
- Map many virtual addresses to same physical page



TLBLEED: TLB AS SHARED STATE

TLBLEED: TLB AS SHARED STATE

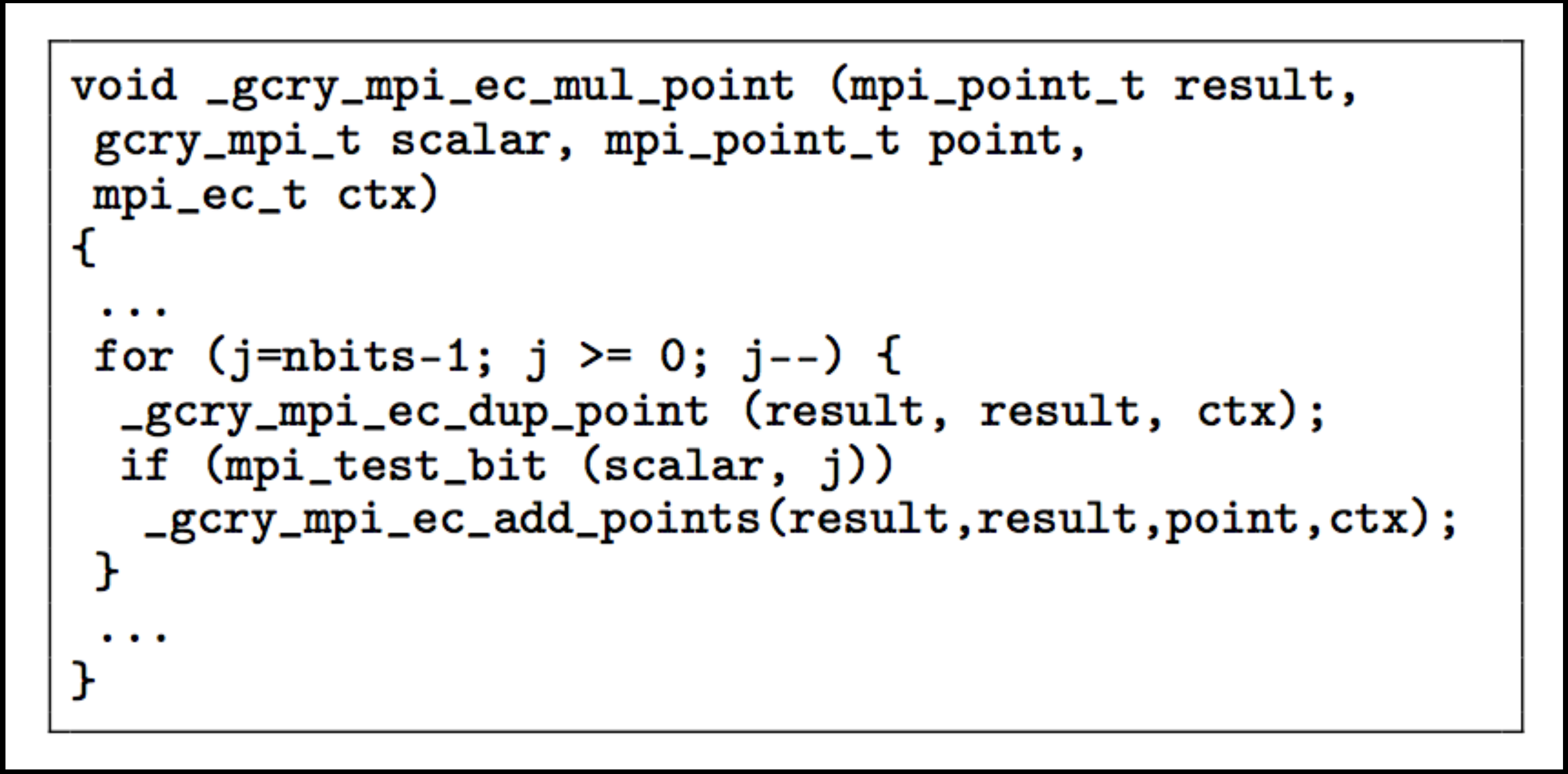
- Let’s observe EdDSA ECC key multiplication



```
void _gcry_mpi_ec_mul_point (mpi_point_t result,
 gcry_mpi_t scalar, mpi_point_t point,
 mpi_ec_t ctx)
{
 ...
 for (j=nbits-1; j >= 0; j--) {
  _gcry_mpi_ec_dup_point (result, result, ctx);
  if (mpi_test_bit (scalar, j))
   _gcry_mpi_ec_add_points(result,result,point,ctx);
 }
 ...
}
```

TLBLEED: TLB AS SHARED STATE

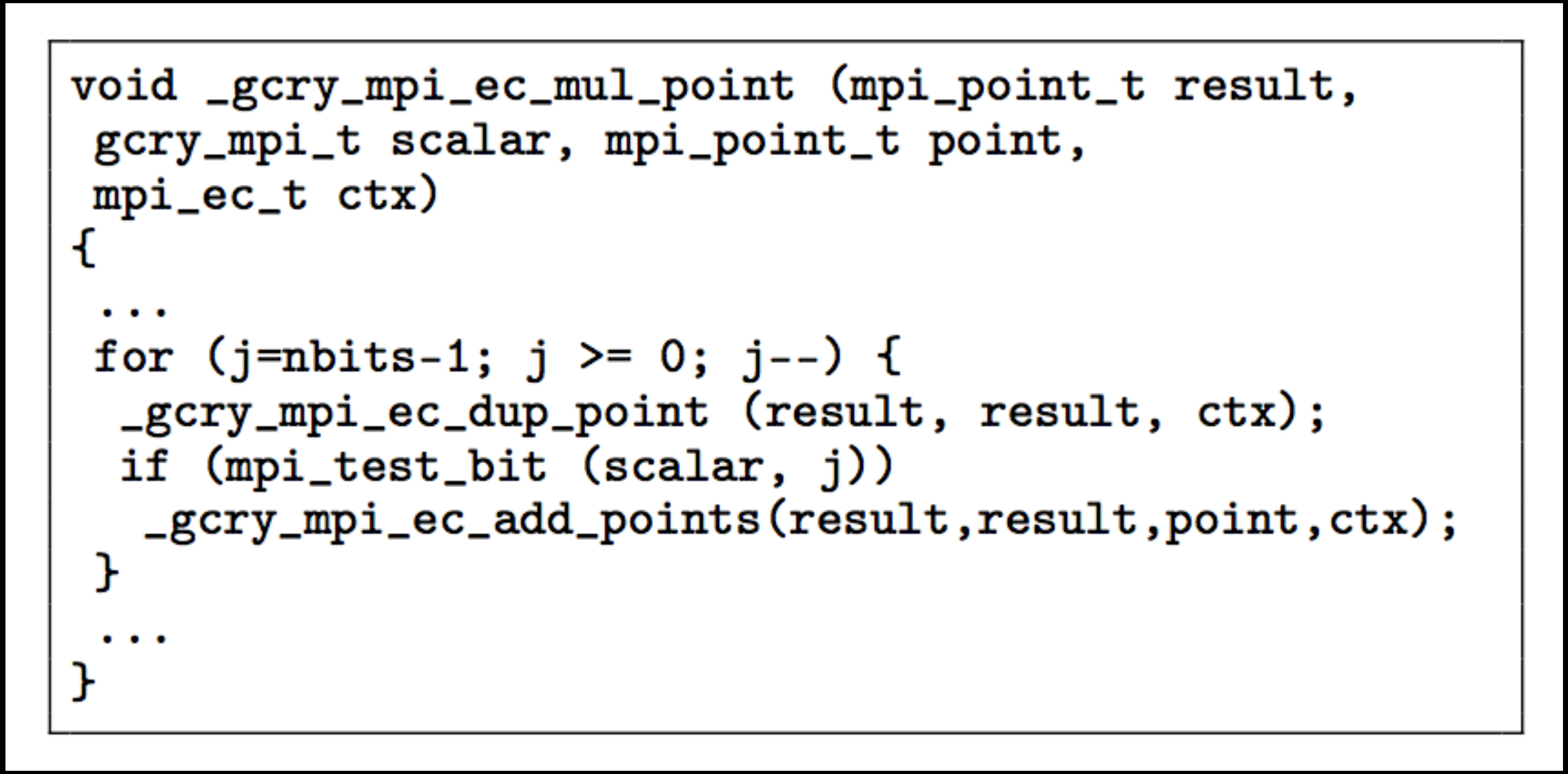
- Let's observe EdDSA ECC key multiplication
- Scalar is secret and ADD only happens if there's a 1



```
void _gcry_mpi_ec_mul_point (mpi_point_t result,
 gcry_mpi_t scalar, mpi_point_t point,
 mpi_ec_t ctx)
{
 ...
 for (j=nbits-1; j >= 0; j--) {
  _gcry_mpi_ec_dup_point (result, result, ctx);
  if (mpi_test_bit (scalar, j))
   _gcry_mpi_ec_add_points(result,result,point,ctx);
 }
 ...
}
```

TLBLEED: TLB AS SHARED STATE

- Let’s observe EdDSA ECC key multiplication
- Scalar is secret and ADD only happens if there’s a 1
- But: we can not use code information! Only data..!



```
void _gcry_mpi_ec_mul_point (mpi_point_t result,
 gcry_mpi_t scalar, mpi_point_t point,
 mpi_ec_t ctx)
{
 ...
 for (j=nbits-1; j >= 0; j--) {
  _gcry_mpi_ec_dup_point (result, result, ctx);
  if (mpi_test_bit (scalar, j))
   _gcry_mpi_ec_add_points(result,result,point,ctx);
 }
 ...
}
```

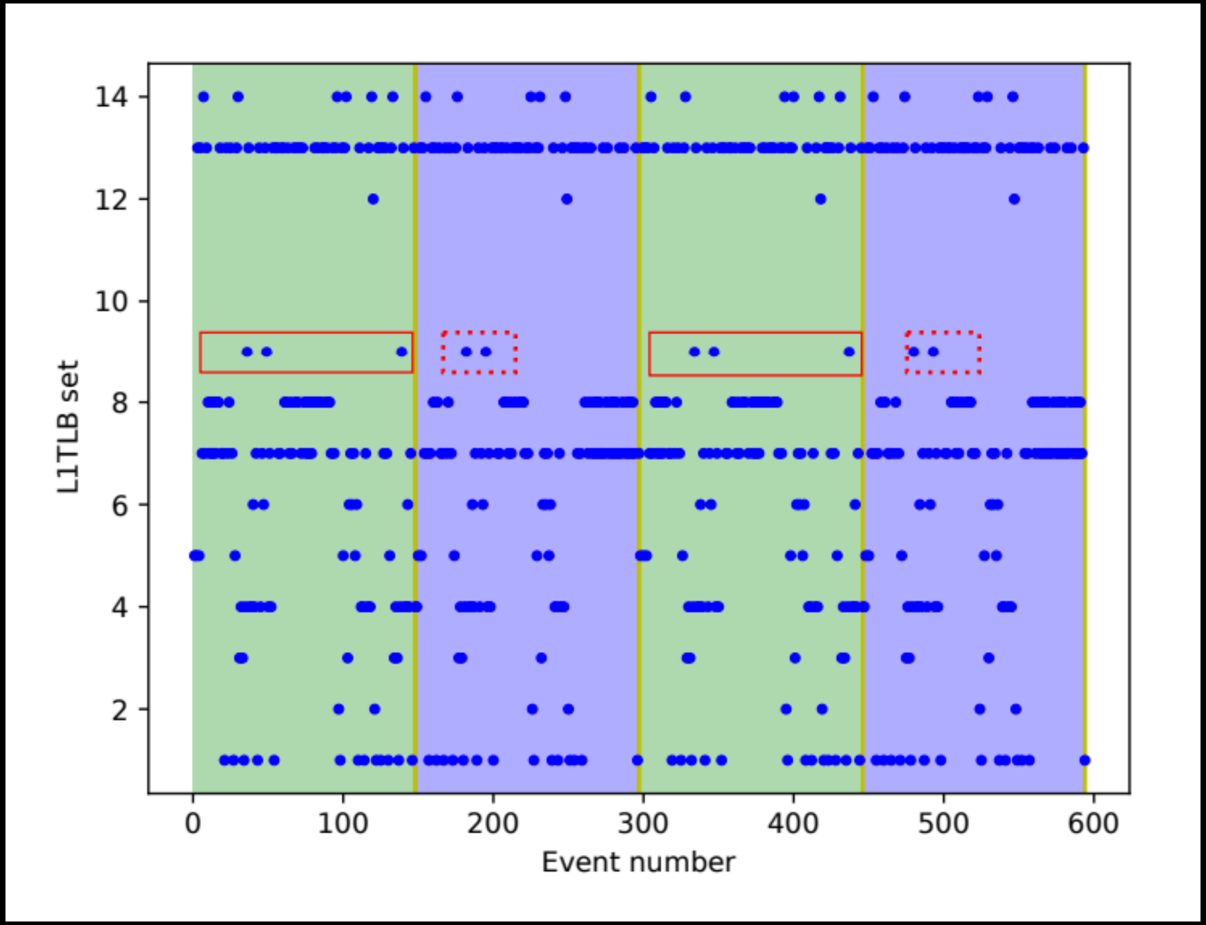
TLBLEED: TLB AS SHARED STATE

TLBLEED: TLB AS SHARED STATE

- Let’s find the spatial L1 DTLB separation
- There isn’t any
- Too much activity in both blue/green cases

TLBLEED: TLB AS SHARED STATE

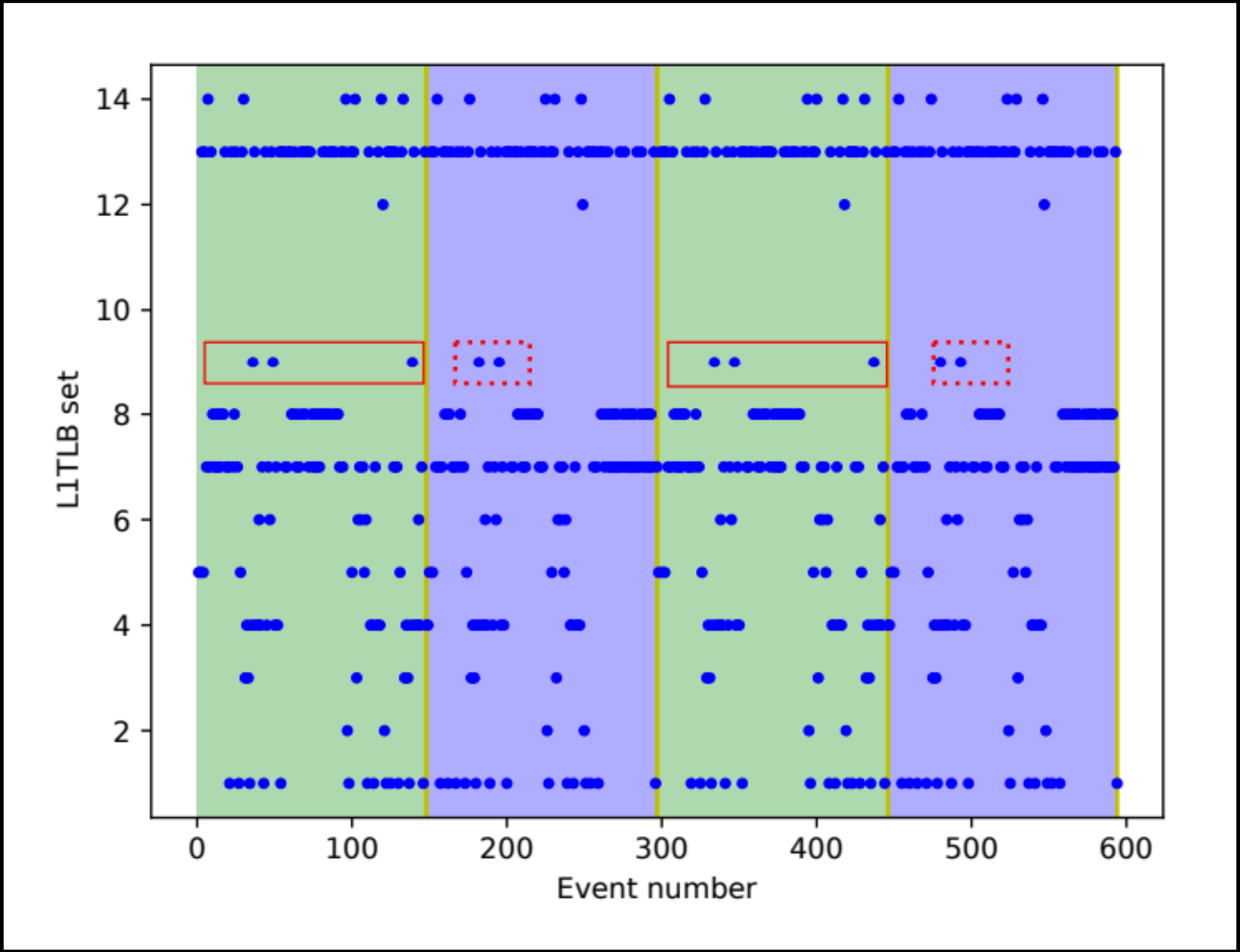
- Let's find the spatial L1 DTLB separation
- There isn't any
- Too much activity in both blue/green cases



TLBLEED: TLB AS SHARED STATE

TLBLEED: TLB AS SHARED STATE

TLBLEED: TLB AS SHARED STATE



TLBLEED: TLB AS SHARED STATE

TLBLEED: TLB AS SHARED STATE

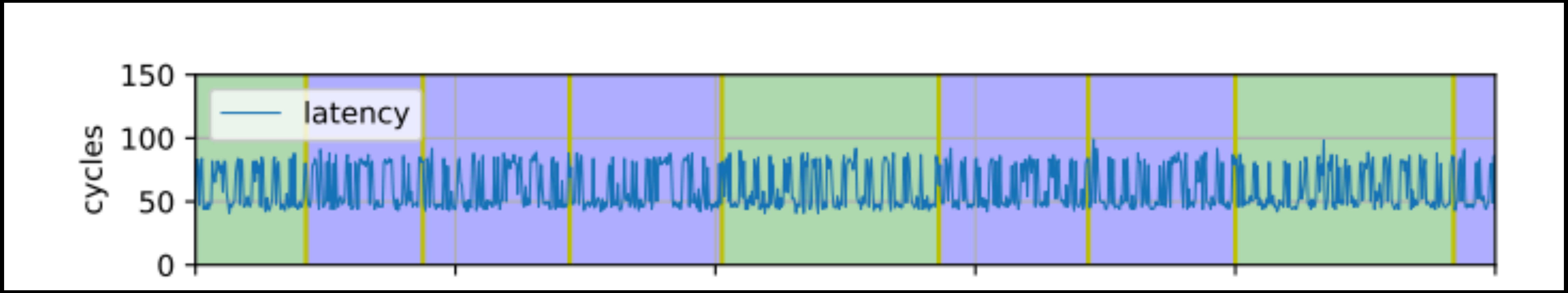
- Monitor a single TLB set and use temporal information

TLBLEED: TLB AS SHARED STATE

- Monitor a single TLB set and use temporal information
- Use machine learning (SVM classifier) to tell the difference

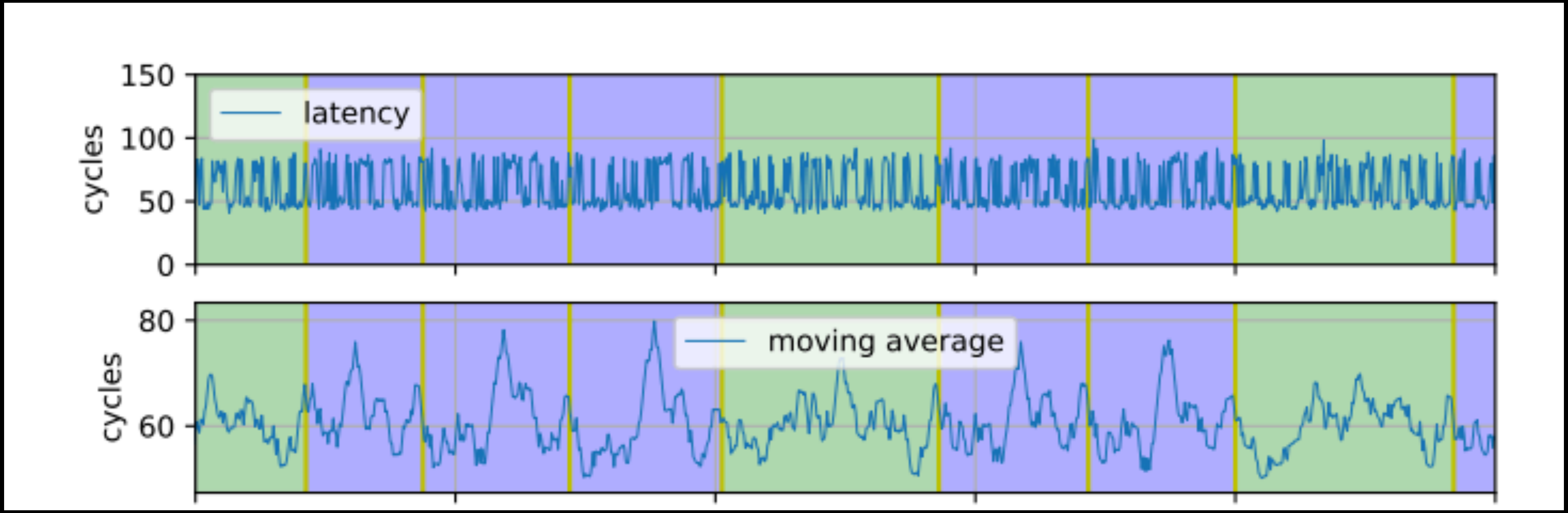
TLBLEED: TLB AS SHARED STATE

- Monitor a single TLB set and use temporal information
- Use machine learning (SVM classifier) to tell the difference



TLBLEED: TLB AS SHARED STATE

- Monitor a single TLB set and use temporal information
- Use machine learning (SVM classifier) to tell the difference

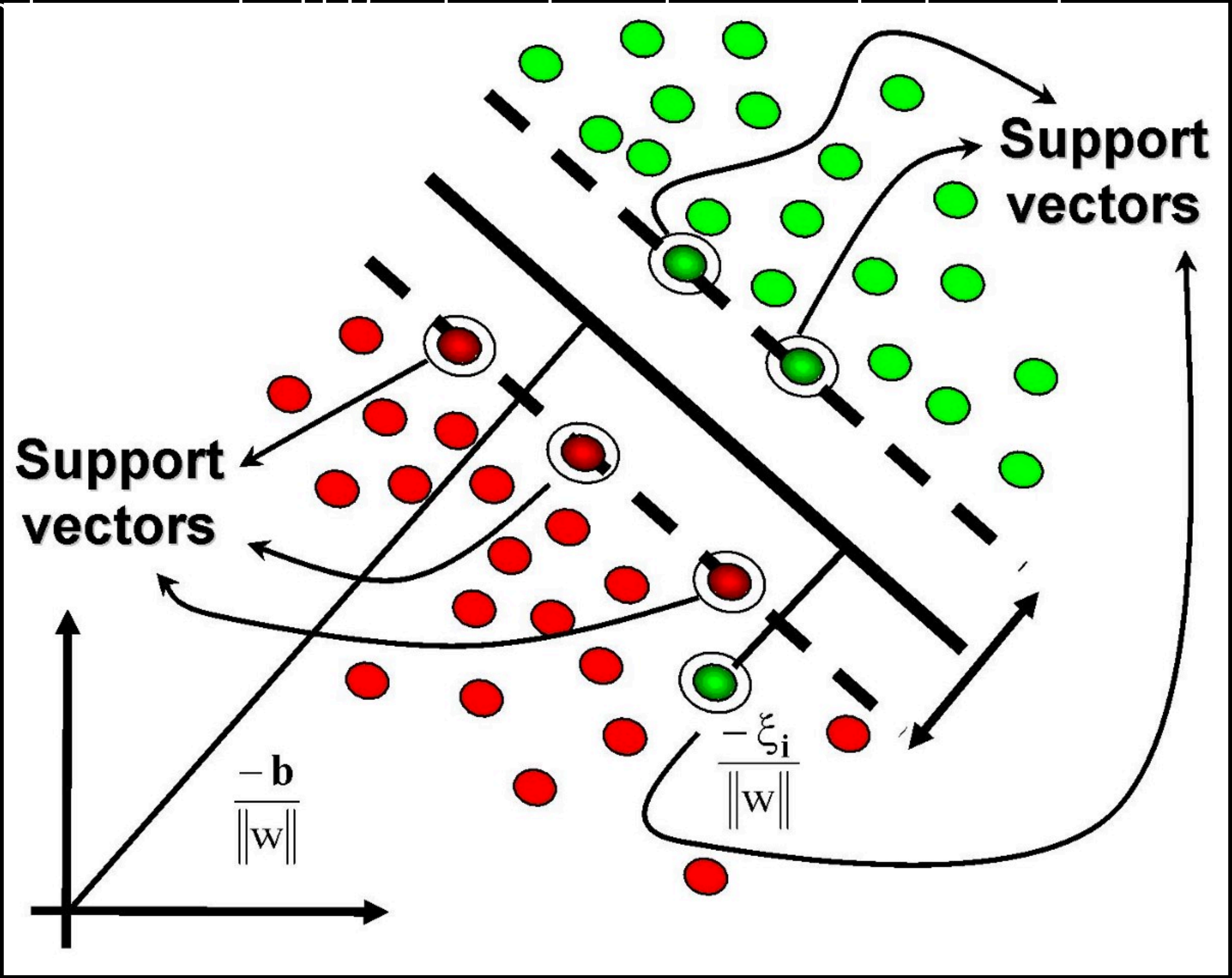


TLBLEED: TLB AS SHARED STATE

- Monitor a single TLB set and use temporal information
- Use machine learning (SVM classifier) to tell the difference

TLBLEED: TLB AS SHARED STATE

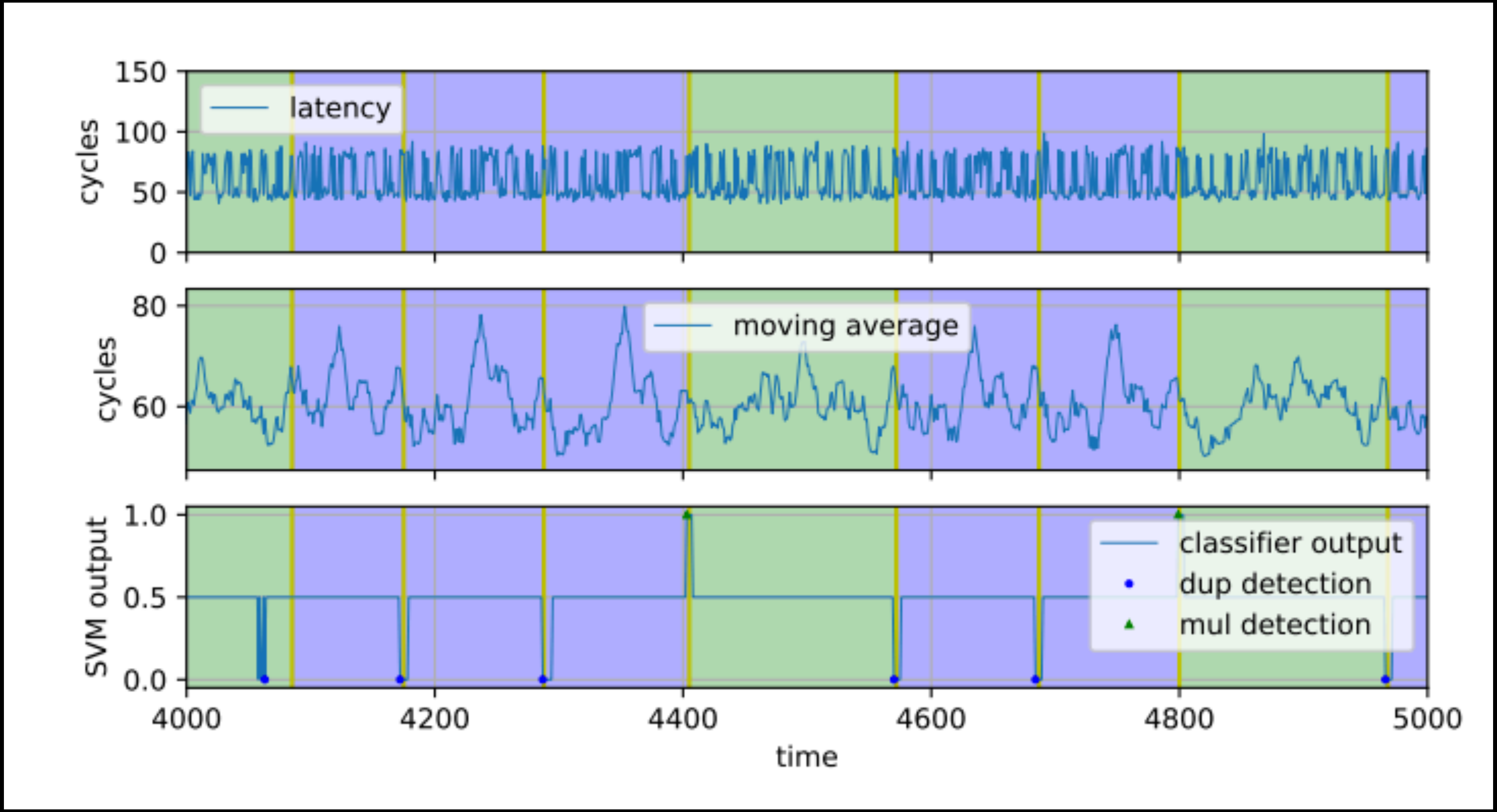
- Mon
- Use

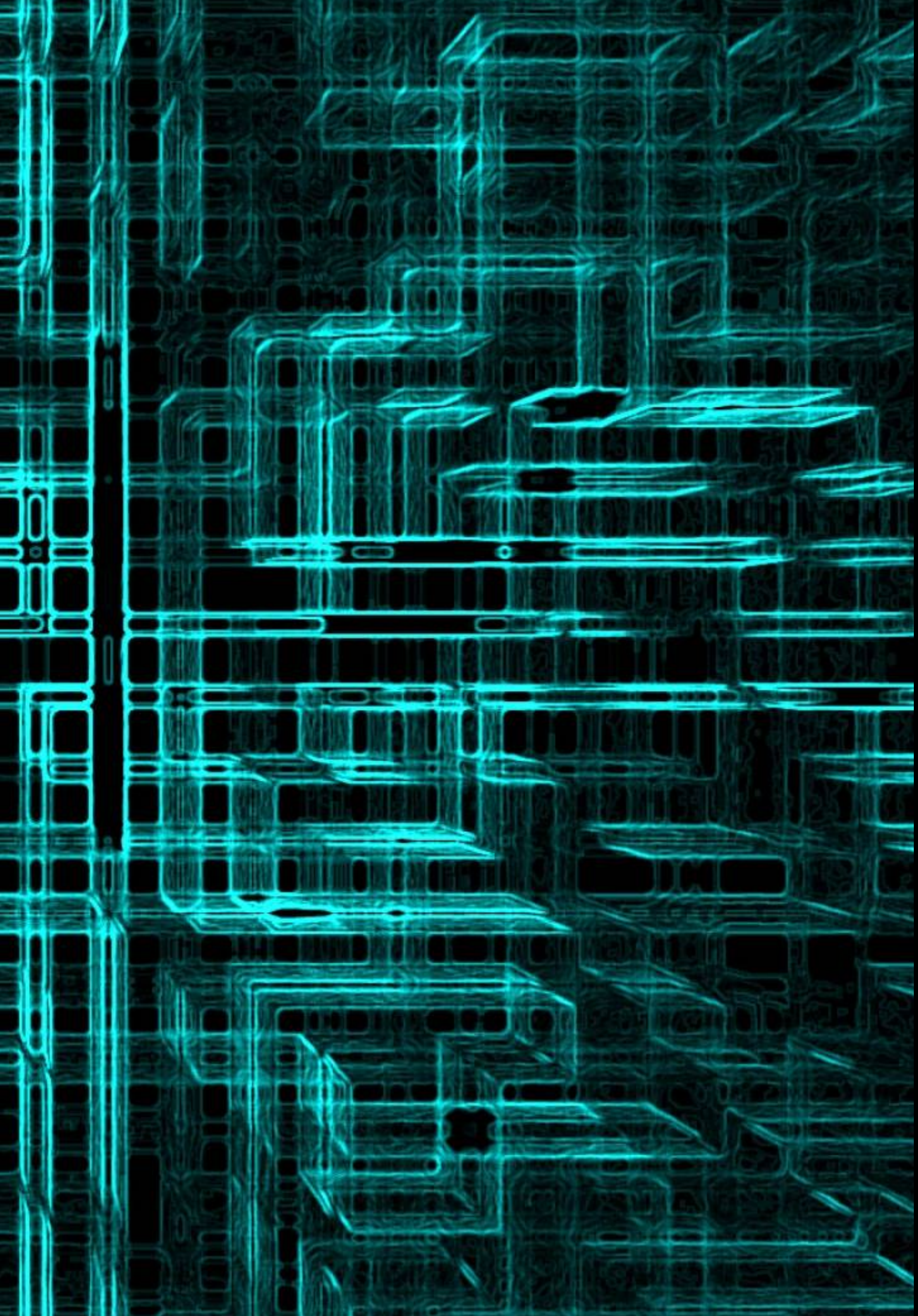


TLBLEED: TLB AS SHARED STATE

- Monitor a single TLB set and use temporal information

- Use machine learning (SVM classifier) to tell the difference

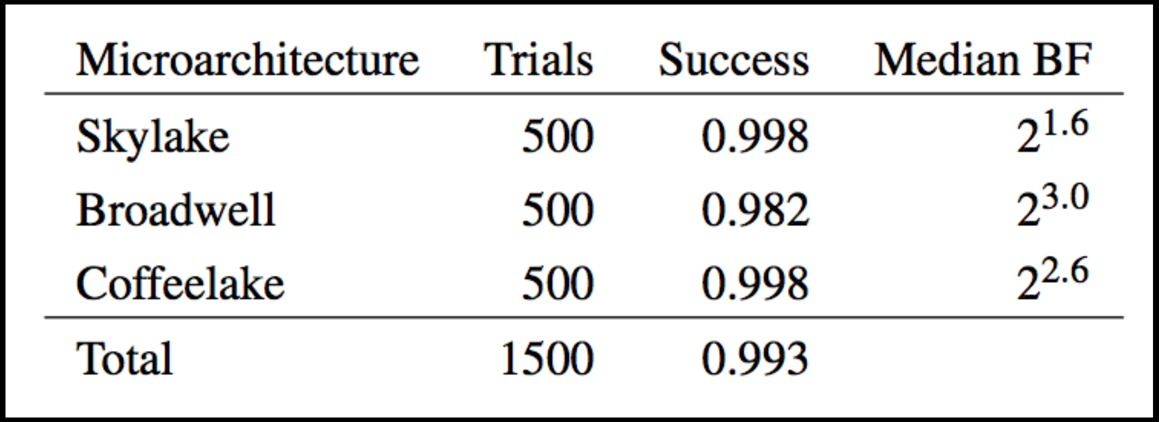




EVALUATION

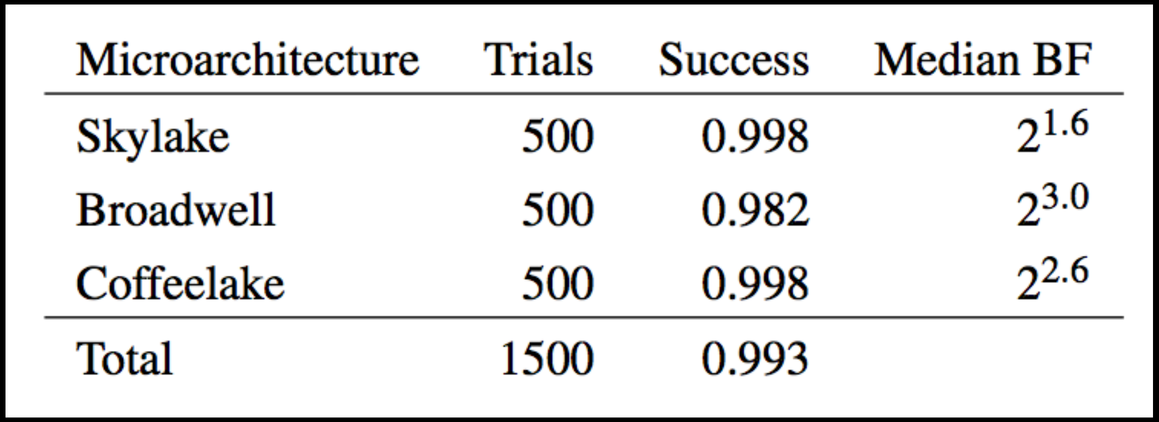
TLBLEED RELIABILITY

TLBLEED RELIABILITY



Microarchitecture	Trials	Success	Median BF
Skylake	500	0.998	$2^{1.6}$
Broadwell	500	0.982	$2^{3.0}$
Coffeelake	500	0.998	$2^{2.6}$
Total	1500	0.993	

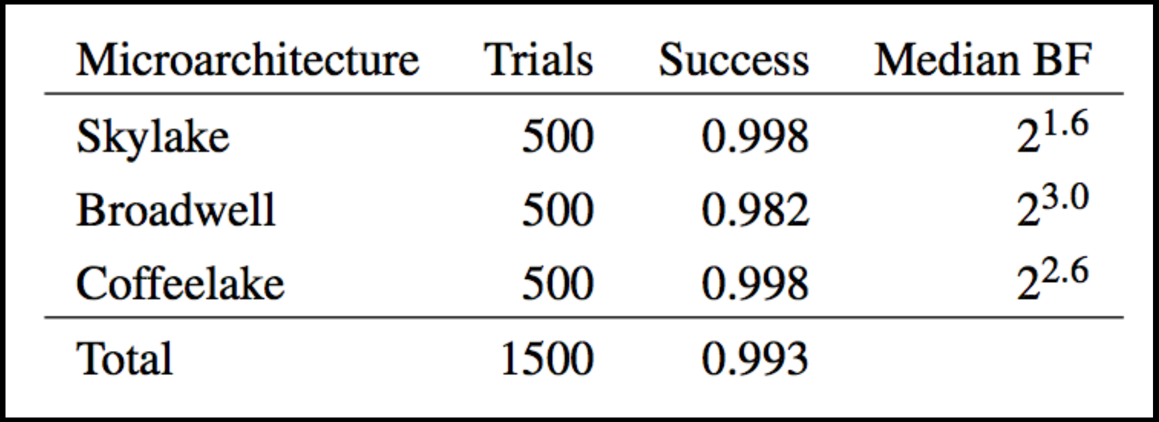
TLBLEED RELIABILITY



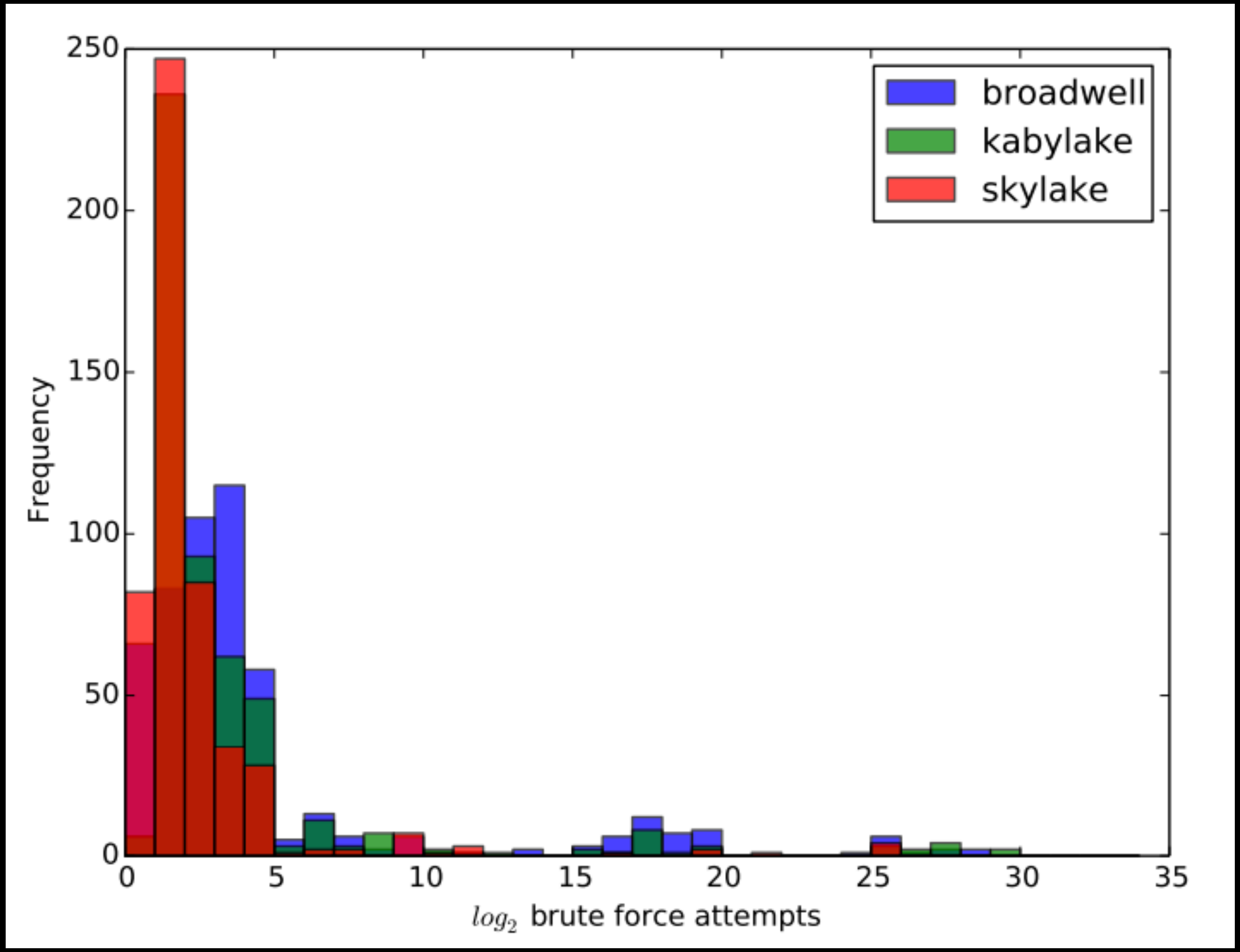
Microarchitecture	Trials	Success	Median BF
Skylake	500	0.998	$2^{1.6}$
Broadwell	500	0.982	$2^{3.0}$
Coffeelake	500	0.998	$2^{2.6}$
Total	1500	0.993	

- Single trace capture: 1ms
- Median end-to-end time: 17s

TLBLEED RELIABILITY

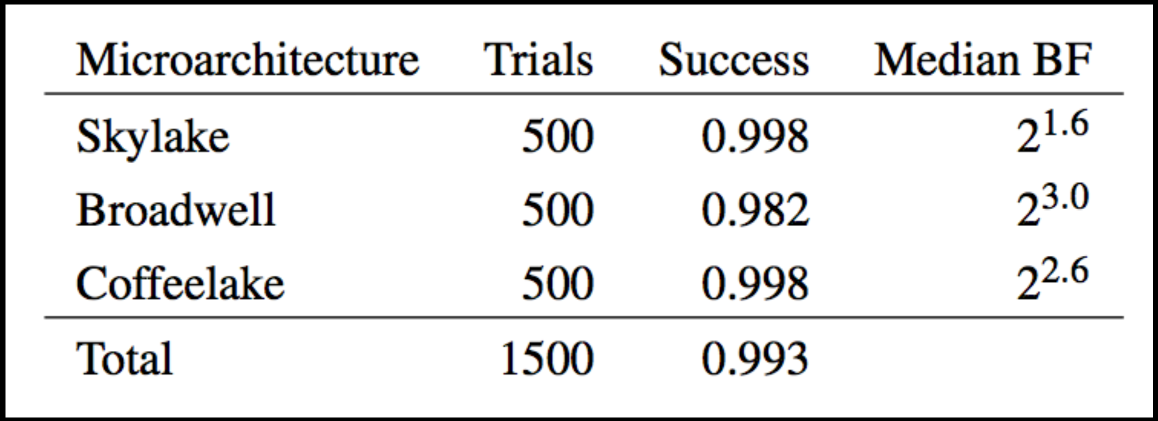


Microarchitecture	Trials	Success	Median BF
Skylake	500	0.998	$2^{1.6}$
Broadwell	500	0.982	$2^{3.0}$
Coffeelake	500	0.998	$2^{2.6}$
Total	1500	0.993	

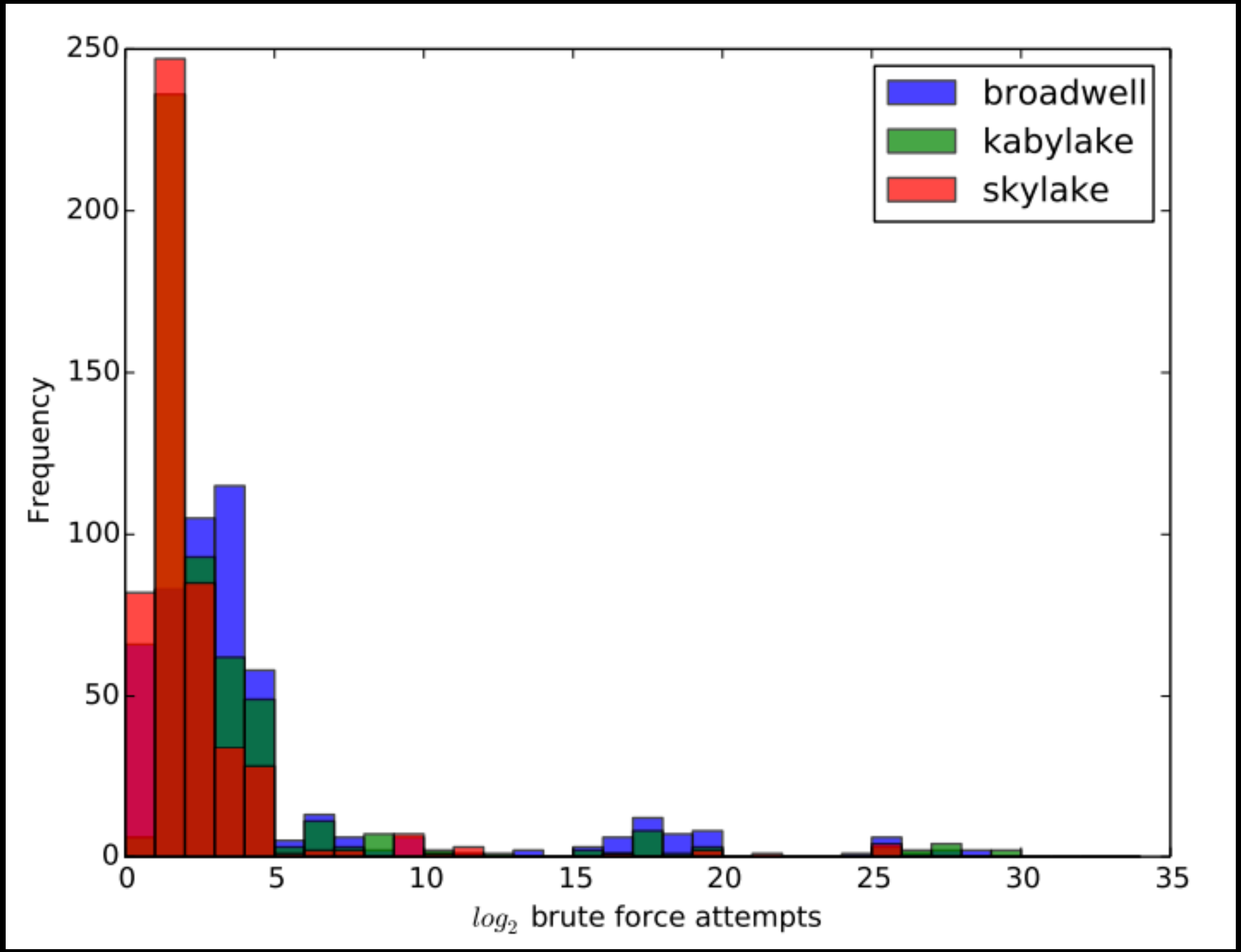


- Single trace capture: 1ms
- Median end-to-end time: 17s

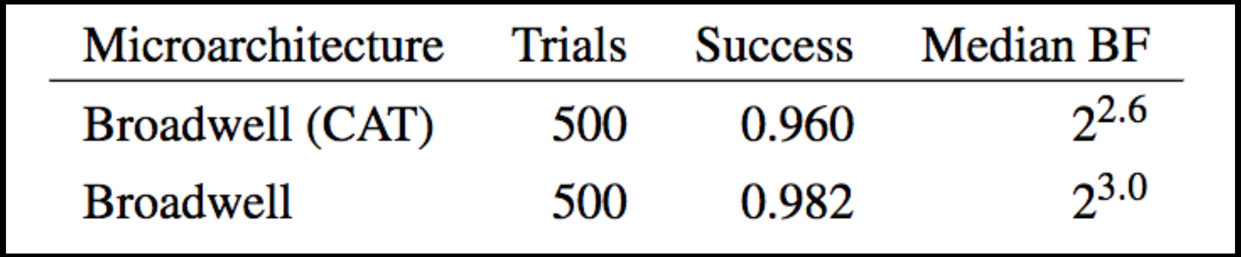
TLBLEED RELIABILITY



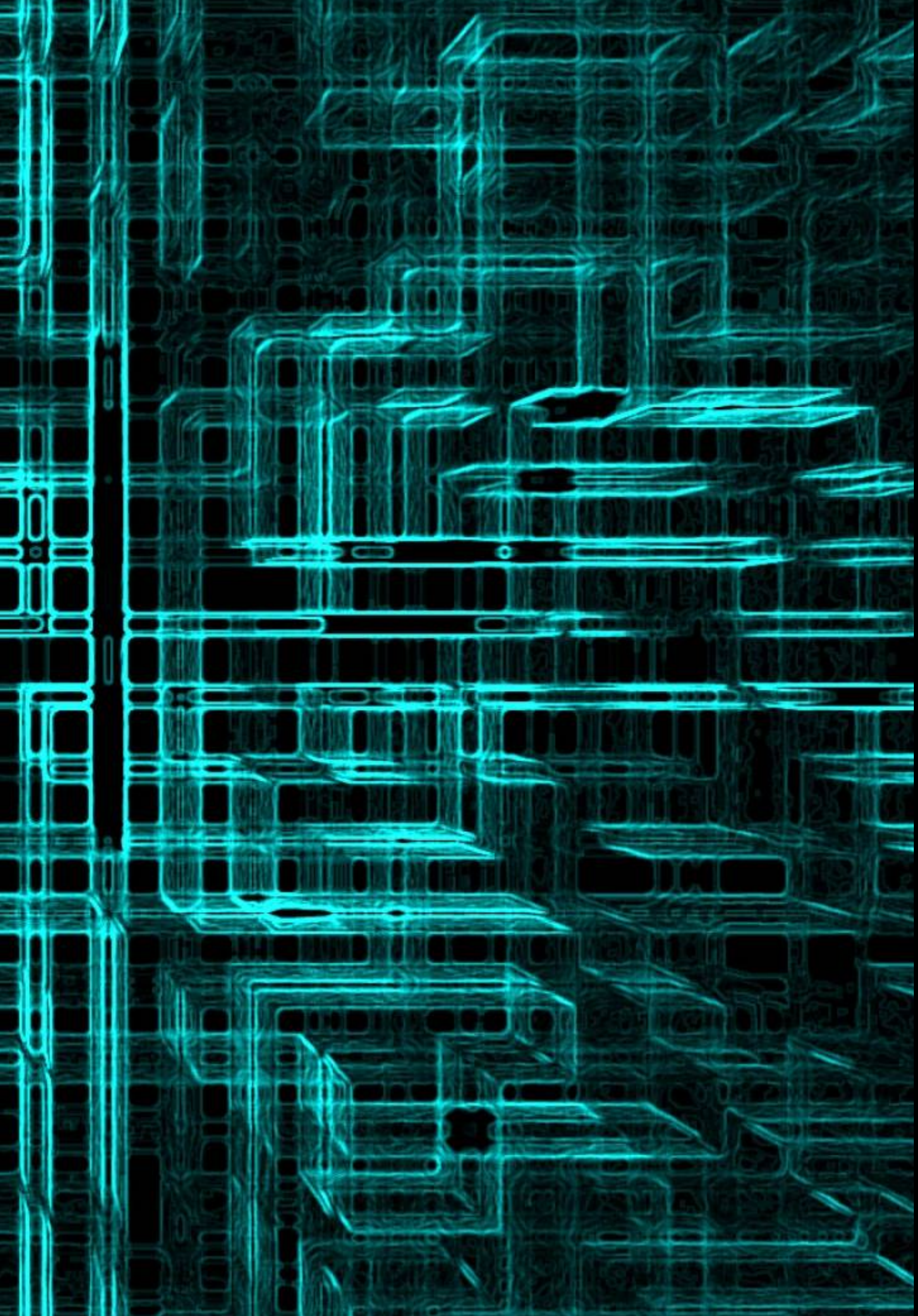
Microarchitecture	Trials	Success	Median BF
Skylake	500	0.998	$2^{1.6}$
Broadwell	500	0.982	$2^{3.0}$
Coffeelake	500	0.998	$2^{2.6}$
Total	1500	0.993	



- Single trace capture: 1ms
- Median end-to-end time: 17s



Microarchitecture	Trials	Success	Median BF
Broadwell (CAT)	500	0.960	$2^{2.6}$
Broadwell	500	0.982	$2^{3.0}$



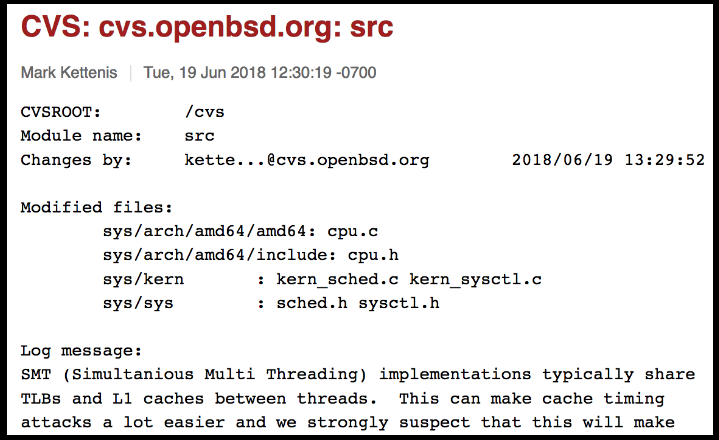
RECEPTION

RECEPTION

- Intel: same power as cache attacks
- OpenBSD disabled Intel HT
- Widespread media coverage, logo thanks to TheRegister
- Wikipedia

RECEPTION

- Intel: same power as cache attacks
- OpenBSD disabled Intel HT
- Widespread media coverage, logo thanks to TheRegister
- Wikipedia



CVS: cvs.openbsd.org: src

Mark Kettenis | Tue, 19 Jun 2018 12:30:19 -0700

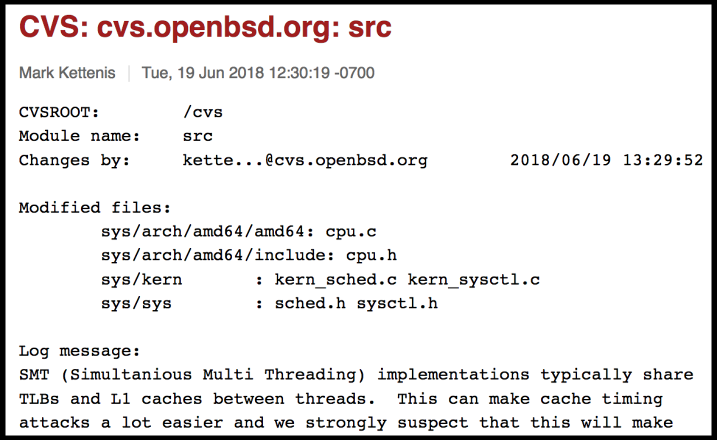
```
CVSROOT:        /cvs
Module name:    src
Changes by:     kette...@cvs.openbsd.org        2018/06/19 13:29:52

Modified files:
        sys/arch/amd64/amd64: cpu.c
        sys/arch/amd64/include: cpu.h
        sys/kern        : kern_sched.c kern_sysctl.c
        sys/sys         : sched.h sysctl.h

Log message:
SMT (Simultanious Multi Threading) implementations typically share
TLBs and L1 caches between threads.  This can make cache timing
attacks a lot easier and we strongly suspect that this will make
```

RECEPTION

- Intel: same power as cache attacks
- OpenBSD disabled Intel HT
- Widespread media coverage, logo thanks to TheRegister
- Wikipedia



CVS: cvs.openbsd.org: src

Mark Kettenis | Tue, 19 Jun 2018 12:30:19 -0700

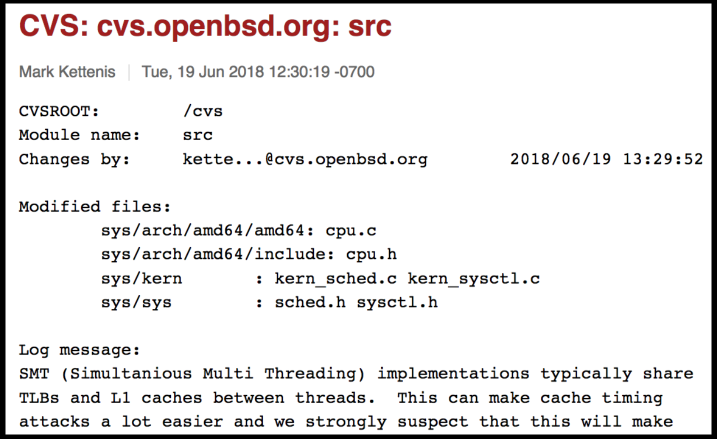
```
CVSROOT:        /cvs
Module name:    src
Changes by:     kette...@cvs.openbsd.org        2018/06/19 13:29:52

Modified files:
        sys/arch/amd64/amd64: cpu.c
        sys/arch/amd64/include: cpu.h
        sys/kern       : kern_sched.c kern_sysctl.c
        sys/sys        : sched.h sysctl.h

Log message:
SMT (Simultanious Multi Threading) implementations typically share
TLBs and L1 caches between threads.  This can make cache timing
attacks a lot easier and we strongly suspect that this will make
```

RECEPTION

- Intel: same power as cache attacks
- OpenBSD disabled Intel HT
- Widespread media coverage, logo thanks to TheRegister
- Wikipedia



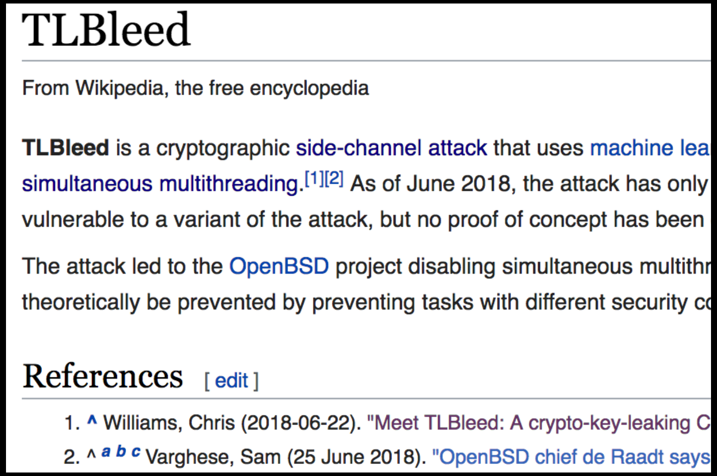
CVS: cvs.openbsd.org: src

Mark Kettenis | Tue, 19 Jun 2018 12:30:19 -0700

```
CVSROOT:        /cvs
Module name:    src
Changes by:     kette...@cvs.openbsd.org        2018/06/19 13:29:52

Modified files:
        sys/arch/amd64/amd64: cpu.c
        sys/arch/amd64/include: cpu.h
        sys/kern        : kern_sched.c kern_sysctl.c
        sys/sys         : sched.h sysctl.h

Log message:
SMT (Simultanious Multi Threading) implementations typically share
TLBs and L1 caches between threads.  This can make cache timing
attacks a lot easier and we strongly suspect that this will make
```



TLBleed

From Wikipedia, the free encyclopedia

TLBleed is a cryptographic side-channel attack that uses machine lea simultaneous multithreading.[1][2] As of June 2018, the attack has only vulnerable to a variant of the attack, but no proof of concept has been

The attack led to the OpenBSD project disabling simultaneous multithr theoretically be prevented by preventing tasks with different security c

References [edit]

1. ^ Williams, Chris (2018-06-22). "Meet TLBleed: A crypto-key-leaking C
2. ^ a b c Varghese, Sam (25 June 2018). "OpenBSD chief de Raadt says

CONCLUSION

CONCLUSION

- Practical, reliable, high resolution side channels exist outside the cache

CONCLUSION

- Practical, reliable, high resolution side channels exist outside the cache
- They bypass defences

CONCLUSION

- Practical, reliable, high resolution side channels exist outside the cache
- They bypass defences
- @bjg @kavehrazavi

CONCLUSION

- Practical, reliable, high resolution side channels exist outside the cache
- They bypass defences

- @bjg @kavehrazavi
- @vu5ec

CONCLUSION

- Practical, reliable, high resolution side channels exist outside the cache
- They bypass defences

- @bjg @kavehrazavi
- @vu5ec
- www.vusec.net

CONCLUSION

- Practical, reliable, high resolution side channels exist outside the cache
- They bypass defences
- @bjg @kavehrazavi
- @vu5ec
- www.vusec.net
- Thank you for listening