

Microarchitectural Attack

Cache Based Attacks

Dr Milan Patnaik

Indian Institute of Technology Madras, India
Rashtriya Raksha University, India



Outline

- Cache Timing Attacks.
 - Cache Covert Channel.
 - Flush + Reload Attack
- Cache Collision Attacks.
 - Prime + Probe Attack
 - Time Driven Attacks
- Transient Micro-architectural Attacks.
 - Meltdown
 - Spectre

Outline

- Cache Timing Attacks.
 - Cache Covert Channel.
 - Flush + Reload Attack
- Cache Collision Attacks.
 - Prime + Probe Attack
 - Time Driven Attacks
- Transient Micro-architectural Attacks.
 - Meltdown
 - Spectre

Security

- Cryptography
- Passwords
- Information Flow Policies
- Privileged Rings
- ASLR
- Virtual Machines and confinement
- Javascript and HTML5
(due to restricted access to system resources)
- Enclaves (SGX and Trustzone)

Security

- Cryptography
- Passwords
- Information Flow Policies
- Privileged Rings
- ASLR
- Virtual Machines and confinement
- Javascript and HTML5
(due to restricted access to system resources)
- Enclaves (SGX and Trustzone)

Cache timing attack

Branch prediction attack

Speculation Attacks

Row hammer

Fault Injection Attacks

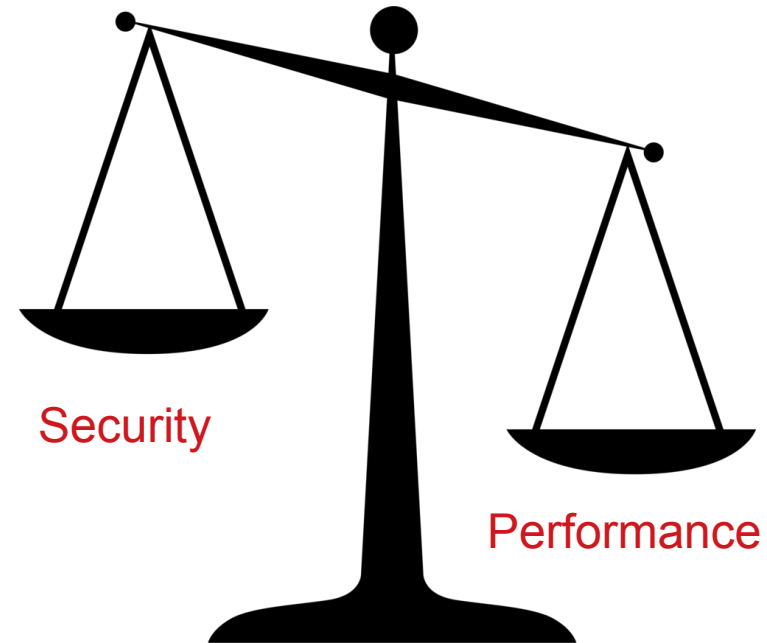
Cold boot attacks

DRAM Row buffer (DRAMA)



Micro-architectural Attacks

- Micro-architectural attacks are caused by:-
-
- Performance optimizations
- Inherent device properties
- Stronger attackers

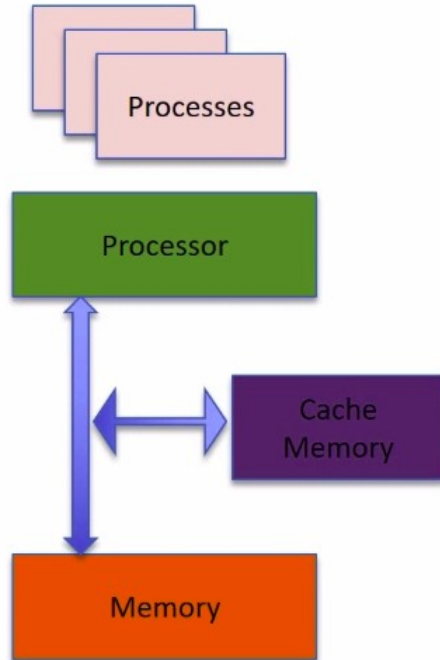


Cache Timing Attacks

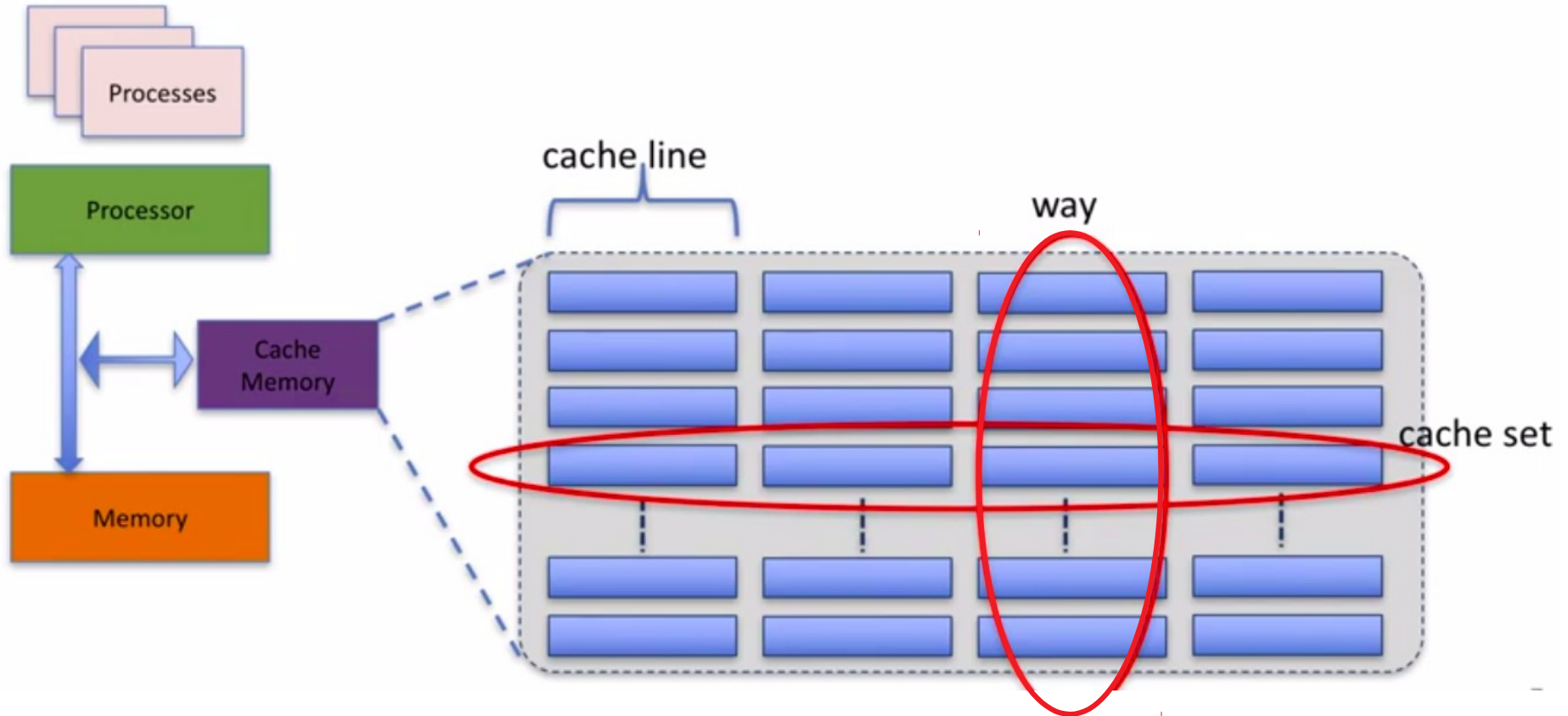
Cache Covert Channels



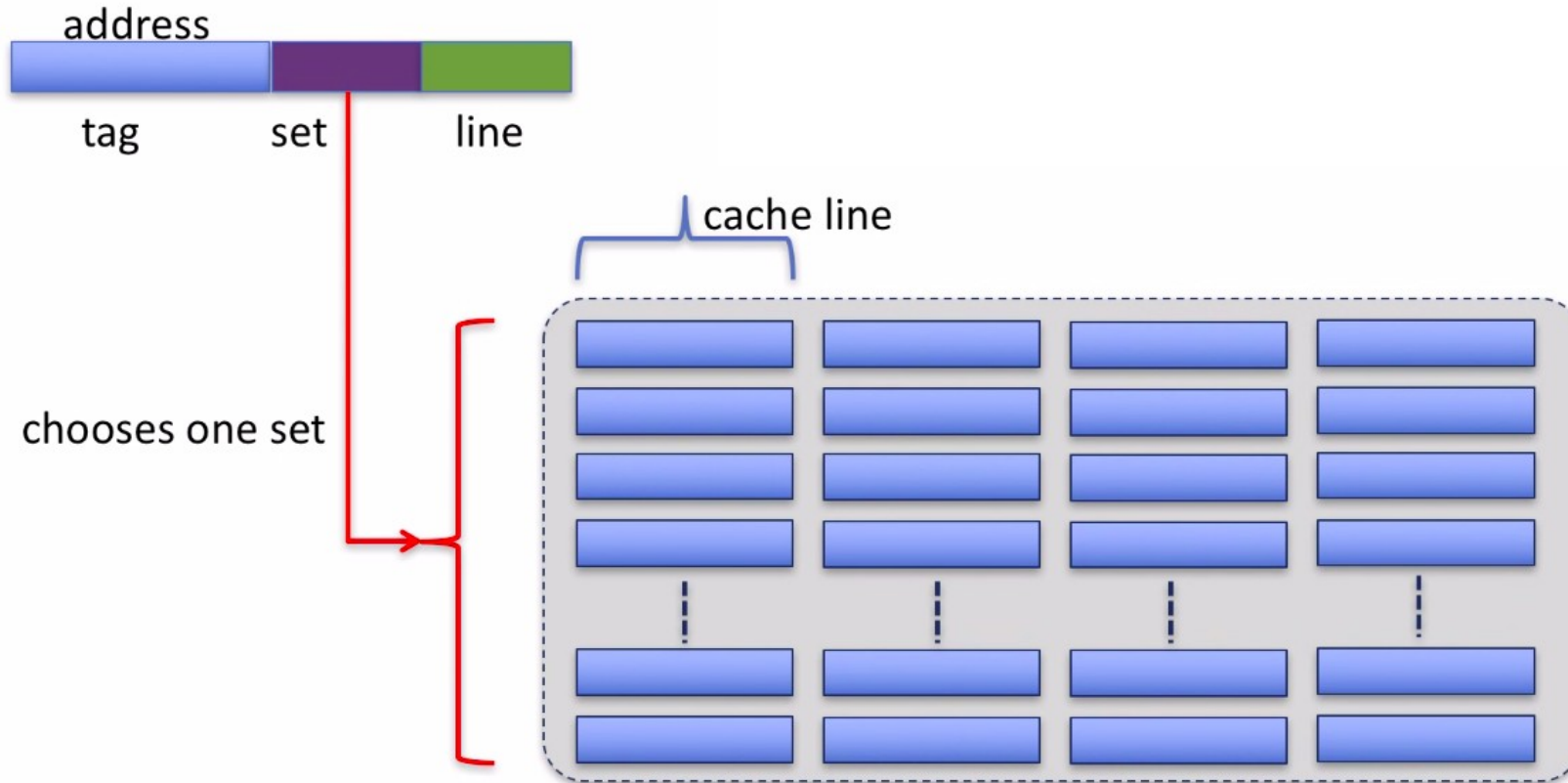
Cache Organisation



Cache Organisation



Cache Organisation



Cache Organisation

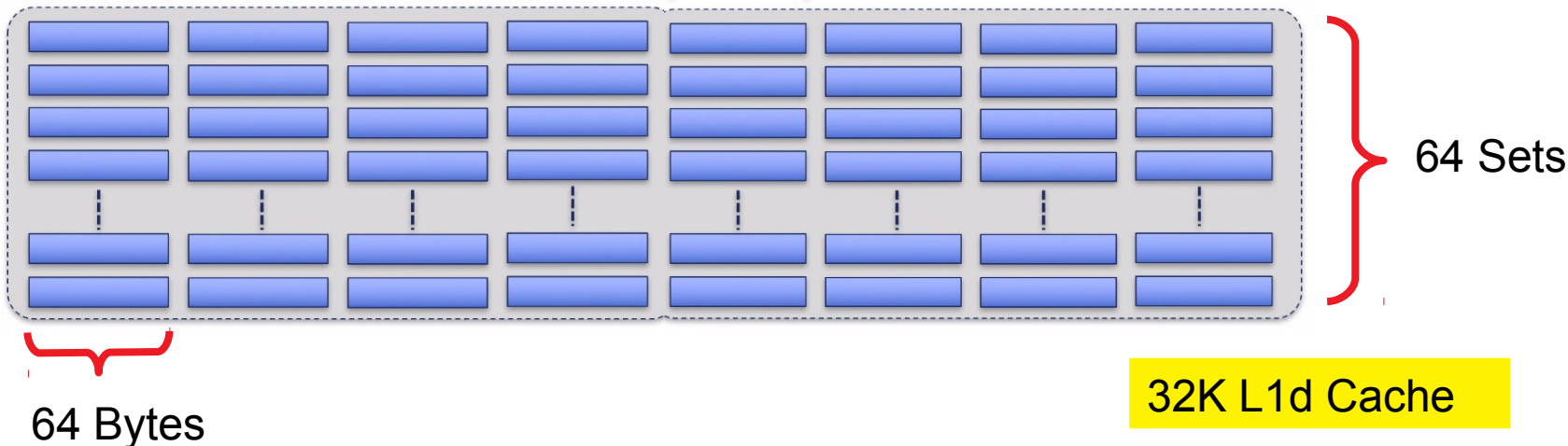
```
maverick@maverick-workforce:~$ lscpu
Architecture:          x86_64
CPU op-mode(s):       32-bit, 64-bit
Byte Order:           Little Endian
CPU(s):               8
On-line CPU(s) list: 0-7
Thread(s) per core:   2
Core(s) per socket:   4
Socket(s):            1
NUMA node(s):        1
Vendor ID:            GenuineIntel
CPU family:           6
Model:                142
Model name:           Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz
Stepping:             11
CPU MHz:              705.790
CPU max MHz:          3900.0000
CPU min MHz:          400.0000
BogoMIPS:             3600.00
Virtualization:       VT-x
L1d cache:            32K
L1i cache:            32K
L2 cache:             256K
L3 cache:             6144K
NUMA node0 CPU(s):   0-7
Flags:                fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe s
yscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 m
onitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand la
hf_lm abm 3dnowprefetch cpuid_fault epb invpcid_single ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid ept_ad fsgsbase tsc_adjust b
mi1 avx2 smep bmi2 erms invpcid mpx rdseed adx smap clflushopt intel_pt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp hwp_notify
hwp_act_window hwp_epp md_clear flush_l1d arch_capabilities
```

Cache Organisation

```
maverick@maverick-workforce: /sys/devices/system/cpu/cpu0/cache/index0$ cat /proc/cpuinfo
processor      : 0
vendor_id    : GenuineIntel
cpu family   : 6
model        : 142
model name   : Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz
stepping     : 11
microcode    : 0xf0
cpu MHz      : 748.275
cache size   : 6144 KB
physical id  : 0
siblings     : 8
core id      : 0
cpu cores    : 4
apicid       : 0
initial apicid : 0
fpu          : yes
fpu_exception : yes
cpuid level  : 22
wp           : yes
flags        : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault epb invpcid_single ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid mpx rdseed adx smap clflushopt intel_pt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_epp md_clear flush_l1d arch_capabilities
bugs         : spectre_v1 spectre_v2 spec_store_bypass mds swaps itlb_multihit srbds mmio_stale_data retbleed
bogomips     : 3600.00
clflush size : 64
cache_alignment : 64
address sizes : 39 bits physical, 48 bits virtual
power management:
```

Cache Organisation

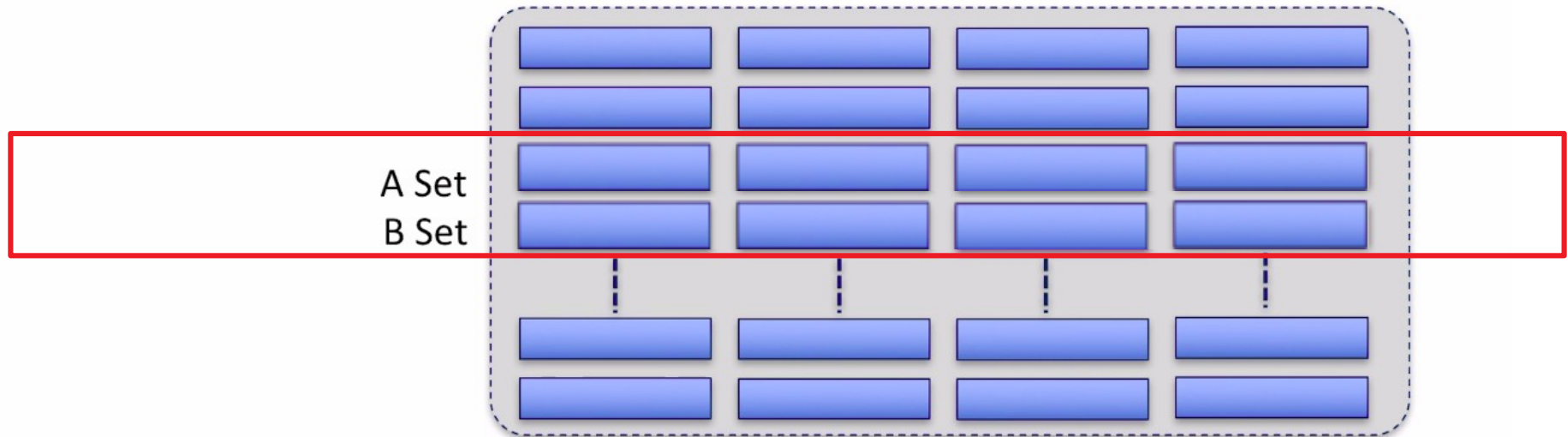
```
maverick@maverick-workforce:/sys/devices/system/cpu/cpu0/cache/index0$ ls
coherency_line_size  level          physical_line_partition  shared_cpu_map  type  ways_of_associativity
id                  number_of_sets  shared_cpu_list          size            uevent
maverick@maverick-workforce:/sys/devices/system/cpu/cpu0/cache/index0$ cat number_of_sets
64
maverick@maverick-workforce:/sys/devices/system/cpu/cpu0/cache/index0$ cat ways_of_associativity
8
maverick@maverick-workforce:/sys/devices/system/cpu/cpu0/cache/index0$ cat coherency_line_size
64
maverick@maverick-workforce:/sys/devices/system/cpu/cpu0/cache/index0$ cat shared_cpu_list
0,4
```



Cache Covert Channels

```
while(1){  
  load A1p2; load A2p2  
  load A3p2; load A4p2  
  load B1p2; load B2p2  
  load B3p2; load B4p2  
}
```

Process P2



Cache Covert Channels

```
while(1){
```

```
  load A1p2; load A2p2
```

```
  load A3p2; load A4p2
```

```
  load B1p2; load B2p2
```

```
  load B3p2; load B4p2
```

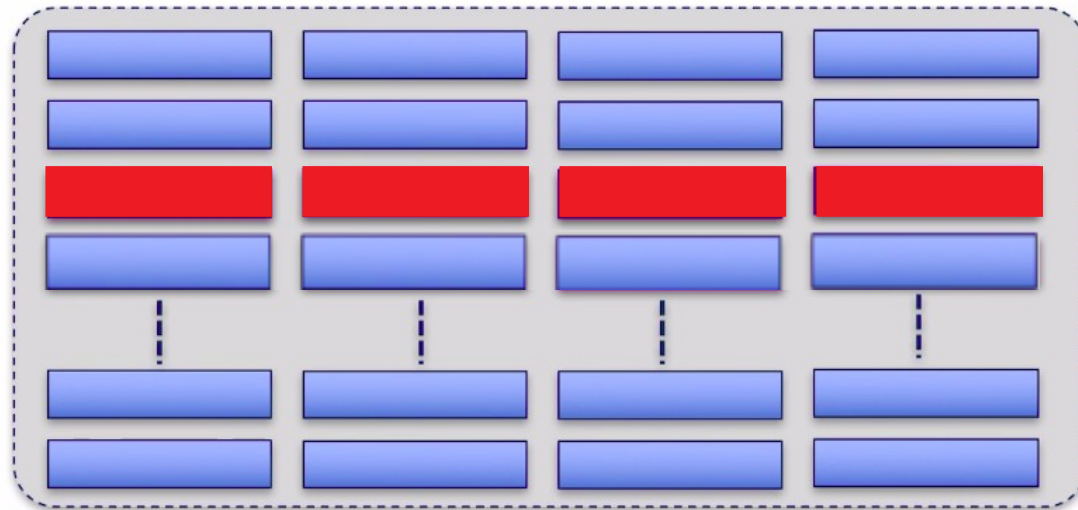
```
}
```

Process P2

Cache Miss Set A

A Set

B Set

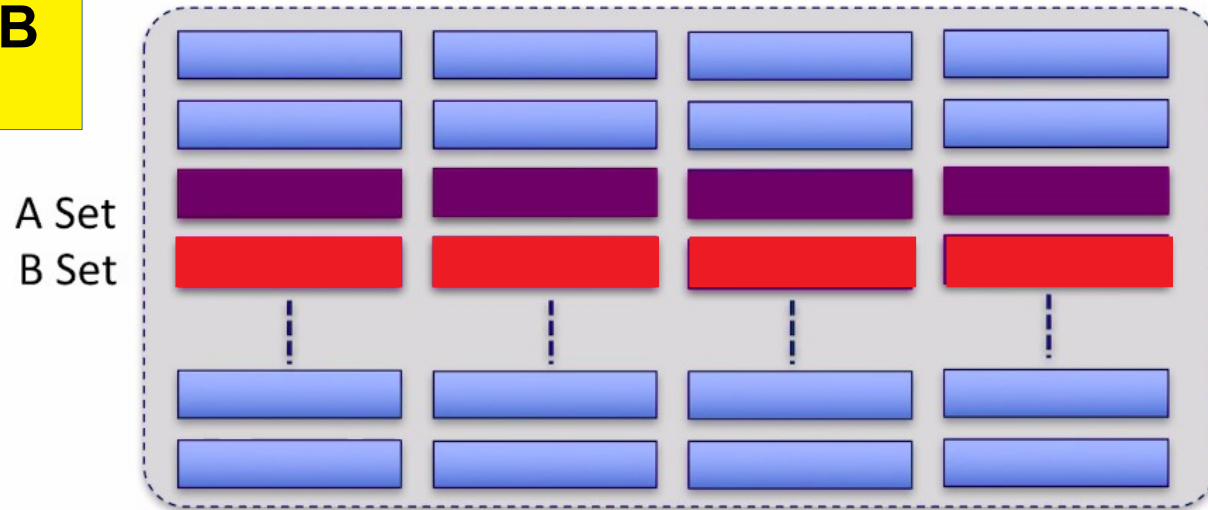


Cache Covert Channels

```
while(1){  
  load A1p2; load A2p2  
  load A3p2; load A4p2  
  load B1p2; load B2p2  
  load B3p2; load B4p2  
}
```

Process P2

Cache Miss Set B



Cache Covert Channels

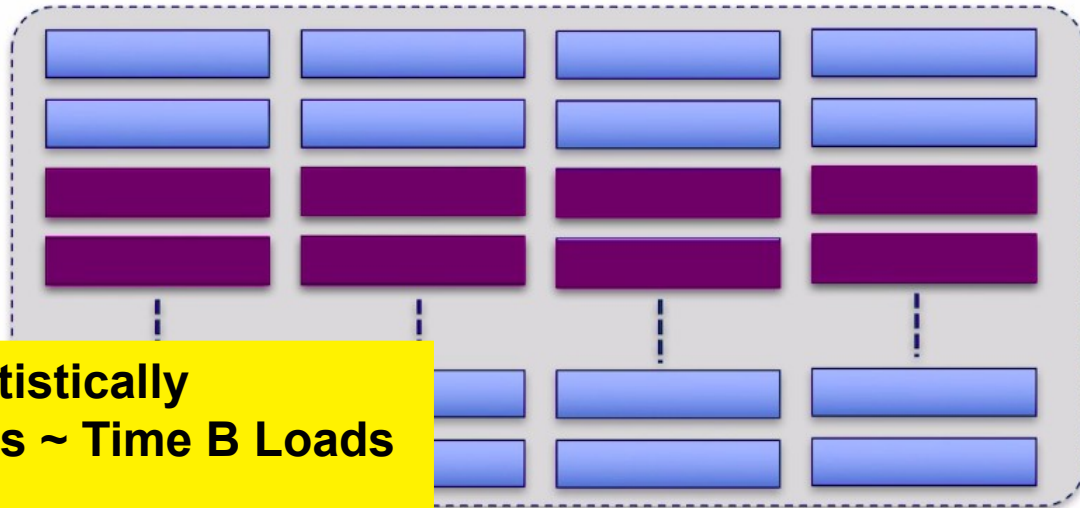
```
while(1){
```

```
  load A1p2; load A2p2  
  load A3p2; load A4p2  
  load B1p2; load B2p2  
  load B3p2; load B4p2
```

Process P2



A Set
B Set



**Statistically
Time A Loads ~ Time B Loads**

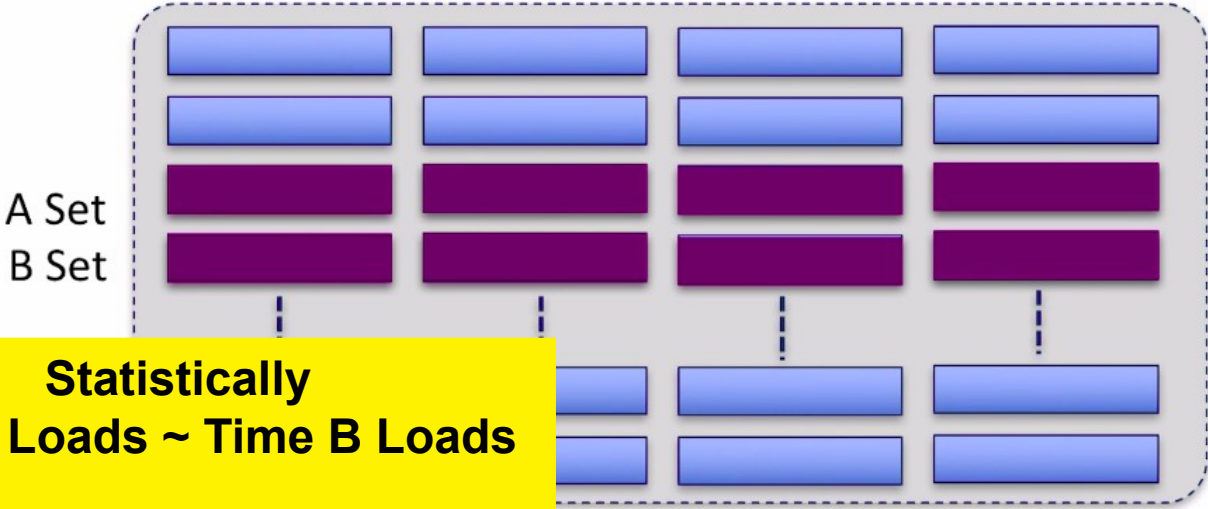
Cache Covert Channels

Process P1

```
if (bit == 1)  
    Load A1p1  
else  
    Load B1p1
```

```
while(1){  
    load A1p2; load A2p2  
    load A3p2; load A4p2  
    load B1p2; load B2p2  
    load B3p2; load B4p2  
}
```

Process P2



**Statistically
Time A Loads ~ Time B Loads**

Cache Covert Channels

Process P1

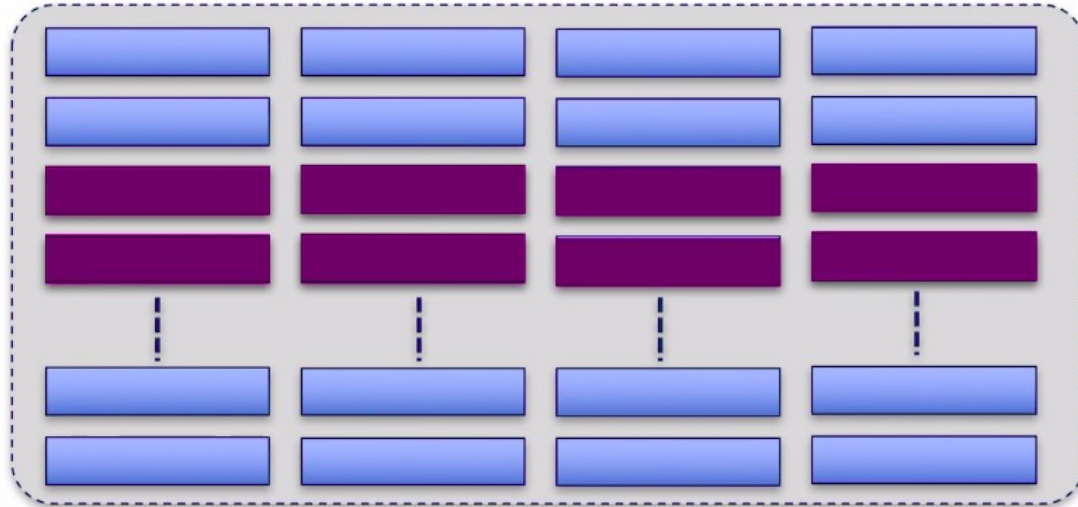
```
if (bit == 1)
  Load A1p1
else
  Load B1p1
```

```
while(1){
  load A1p2; load A2p2
  load A3p2; load A4p2
  load B1p2; load B2p2
  load B3p2; load B4p2
}
```

Process P2



A Set
B Set



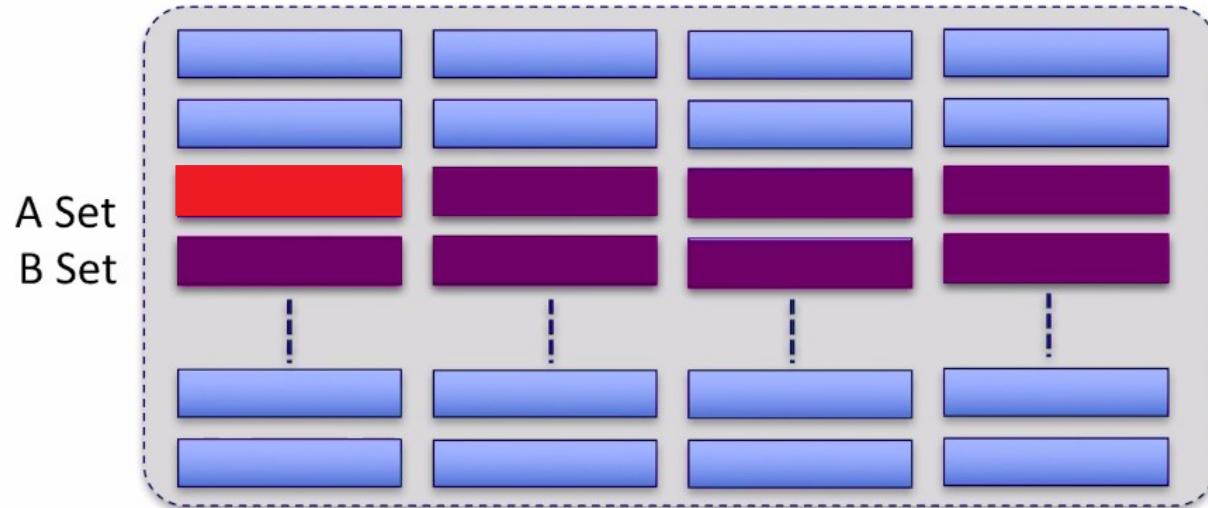
Cache Covert Channels

Process P1

```
if (bit == 1)  
    Load A1p1  
else  
    Load B1p1
```

```
while(1){  
    load A1p2; load A2p2  
    load A3p2; load A4p2  
    load B1p2; load B2p2  
    load B3p2; load B4p2  
}
```

Process P2



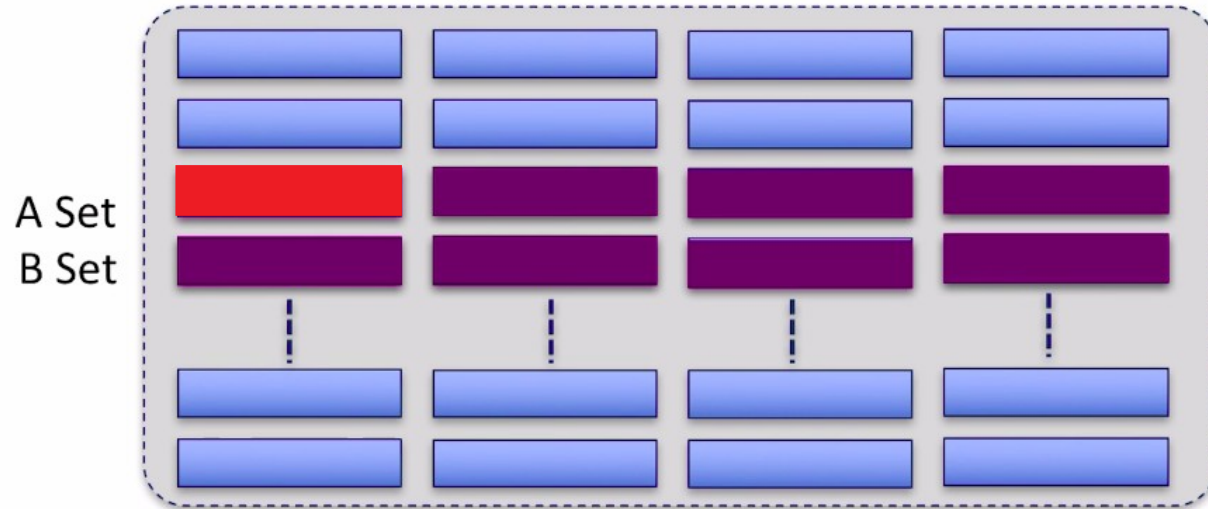
Cache Covert Channels

Process P1

```
if (bit == 1)  
    Load A1p1  
else  
    Load B1p1
```

```
while(1){  
    load A1p2; load A2p2  
    load A3p2; load A4p2  
    load B1p2; load B2p2  
    load B3p2; load B4p2  
}
```

Process P2



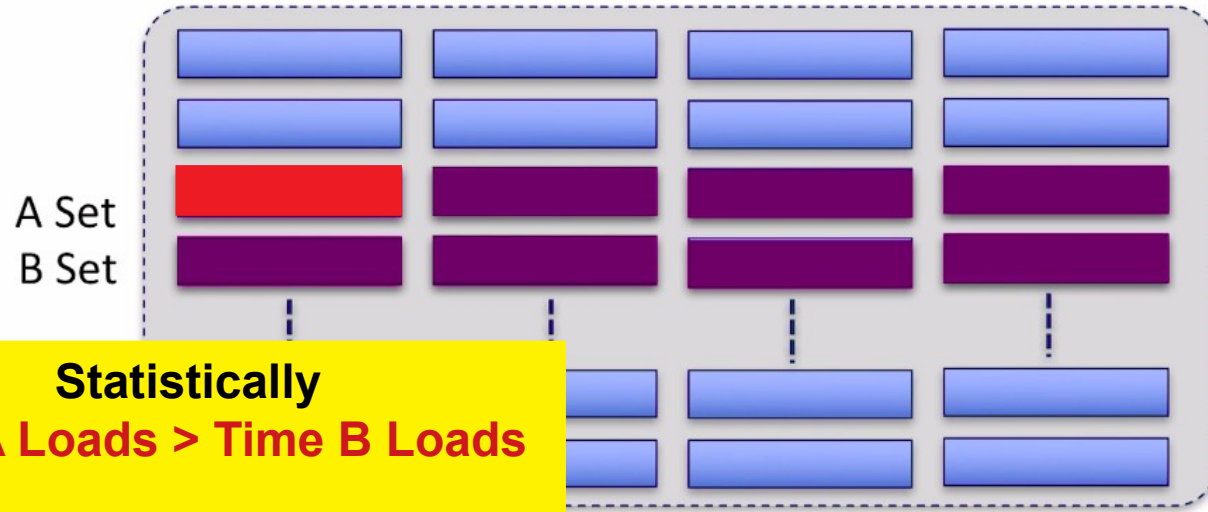
Cache Covert Channels

Process P1

```
if (bit == 1)  
    Load A1p1  
else  
    Load B1p1
```

```
while(1){  
    load A1p2; load A2p2  
    load A3p2; load A4p2  
    load B1p2; load B2p2  
    load B3p2; load B4p2  
}
```

Process P2



Cache Covert Channels

Process P1

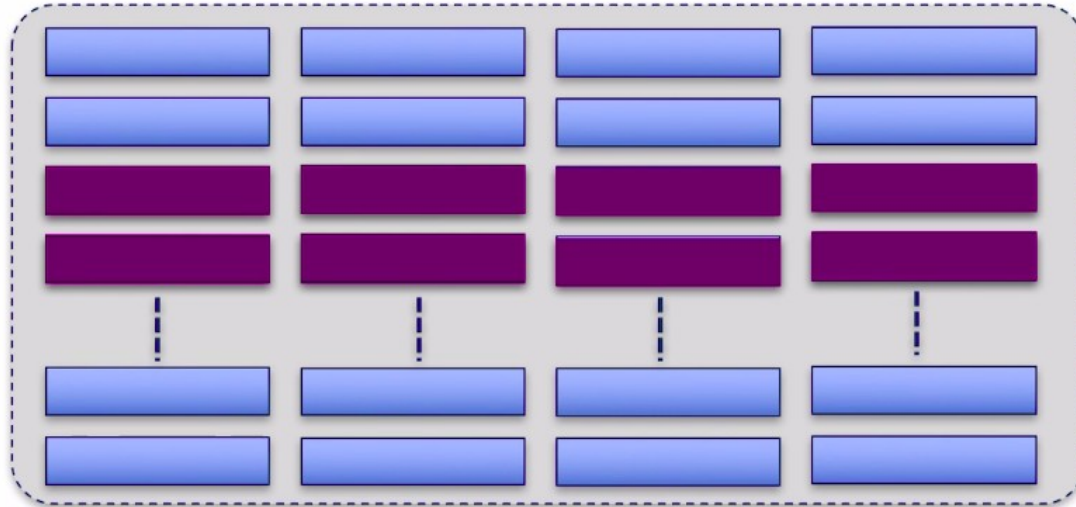
```
if (bit == 1)  
    Load A1p1  
else  
    Load B1p1
```

```
while(1){  
    load A1p2; load A2p2  
    load A3p2; load A4p2  
    load B1p2; load B2p2  
    load B3p2; load B4p2  
}
```

Process P2



A Set
B Set



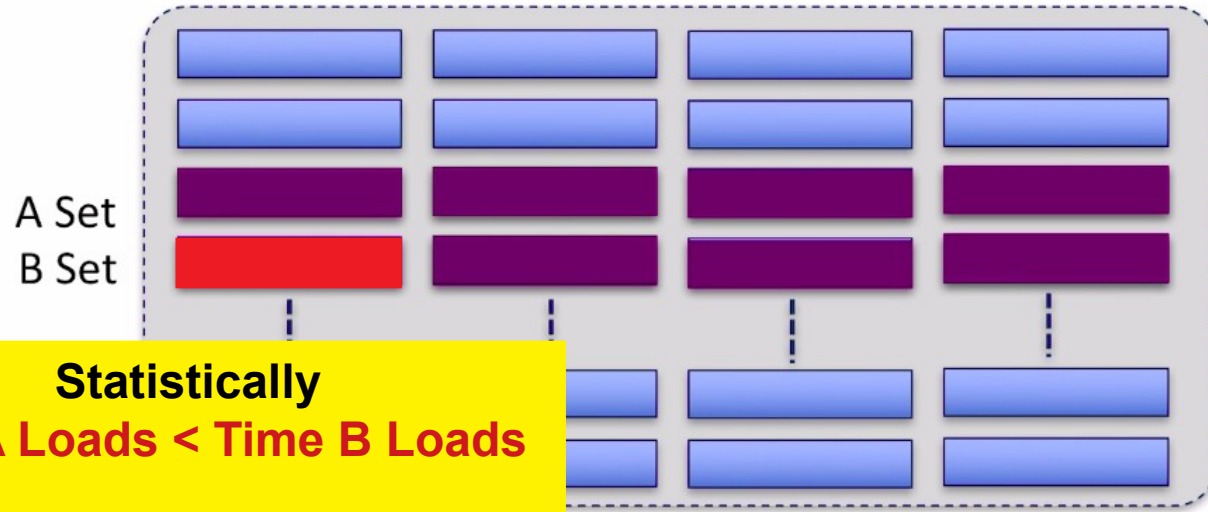
Cache Covert Channels

Process P1

```
if (bit == 1)  
    Load A1p1  
else  
    Load B1p1
```

```
while(1){  
    load A1p2; load A2p2  
    load A3p2; load A4p2  
    load B1p2; load B2p2  
    load B3p2; load B4p2  
}
```

Process P2



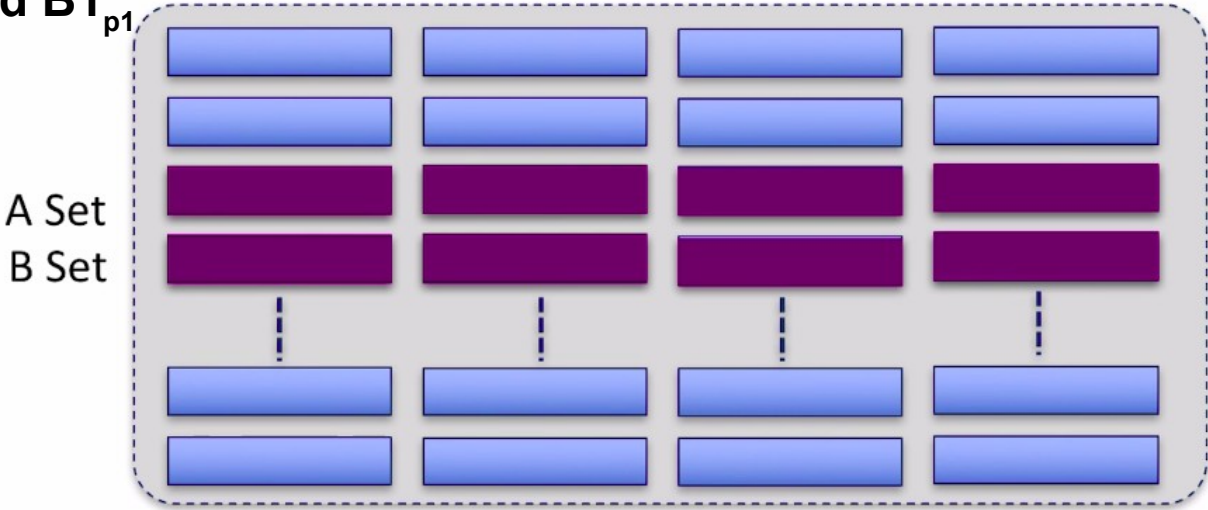
Cache Covert Channels

Process P1

```
bit = message
while (bit[i] != '\0')
  if (bit == 1)
    Load A1p1
  else
    Load B1p1
```

```
while(1){
  load A1p2; load A2p2
  load A3p2; load A4p2
  load B1p2; load B2p2
  load B3p2; load B4p2
}
```

Process P2



Cache Covert Channels

Process P1

```

bit = message
while (bit[i] != '\0')
  If (bit == 1)
    Load A1p1
  else
    Load B1p1

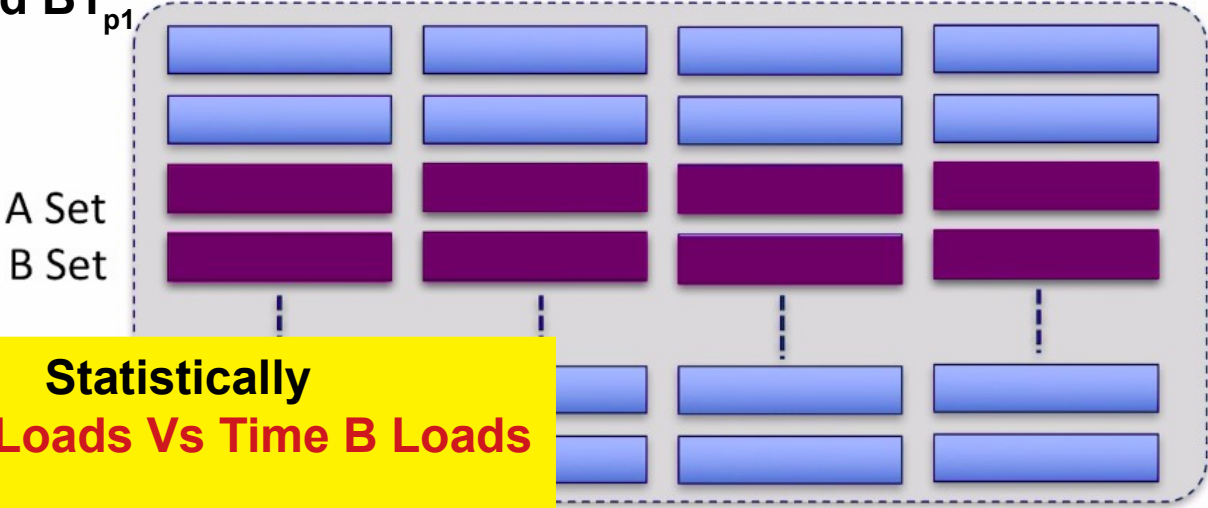
```

```

while(1){
  load A1p2; load A2p2
  load A3p2; load A4p2
  load B1p2; load B2p2
  load B3p2; load B4p2
}

```

Process P2

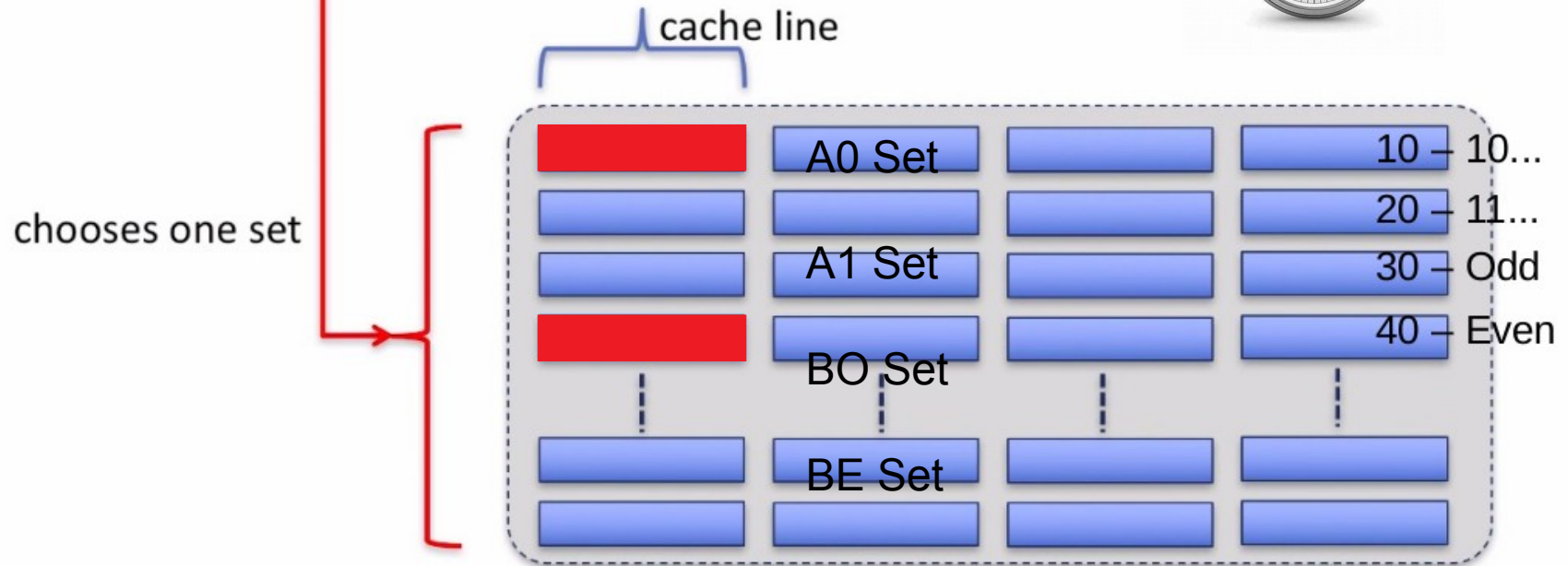


Statistically
Time A Loads Vs Time B Loads

Cache Covert Channels: Send Even 1



1 1 0 1...
EOEO



Cache Covert Channels

- Identifying
 - Cache Covert Channels are difficult
 - Variety of Covert Channels : File, Time etc
- Quantifying
 - Bit rate of communication : bps
- Elimination
 - Careful design
 - Separation
 - Studying characteristic of operations
 - Rate of opening and closing of files

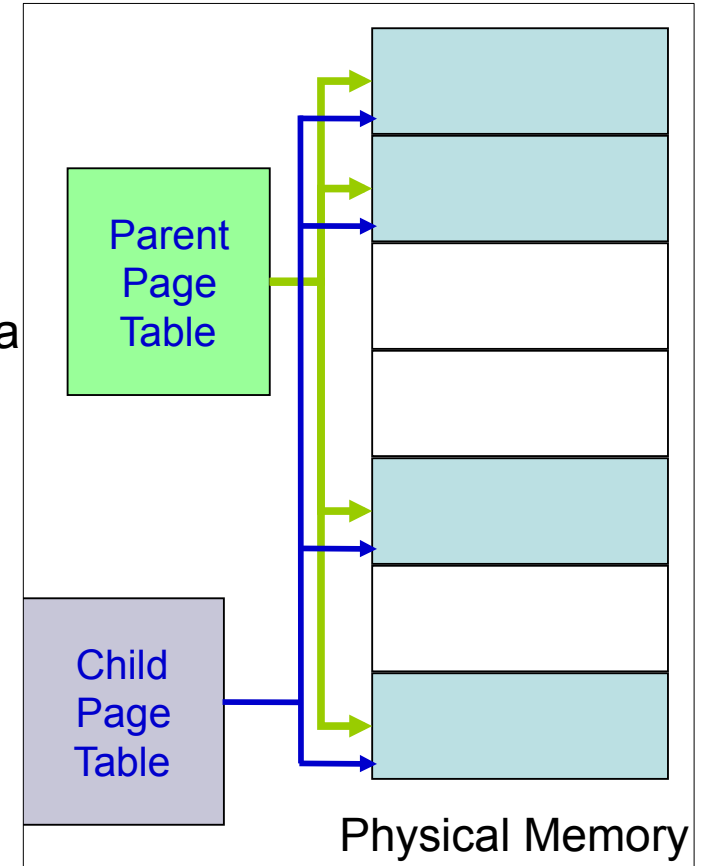
Cache Timing Attacks

Flush + Reload Attack

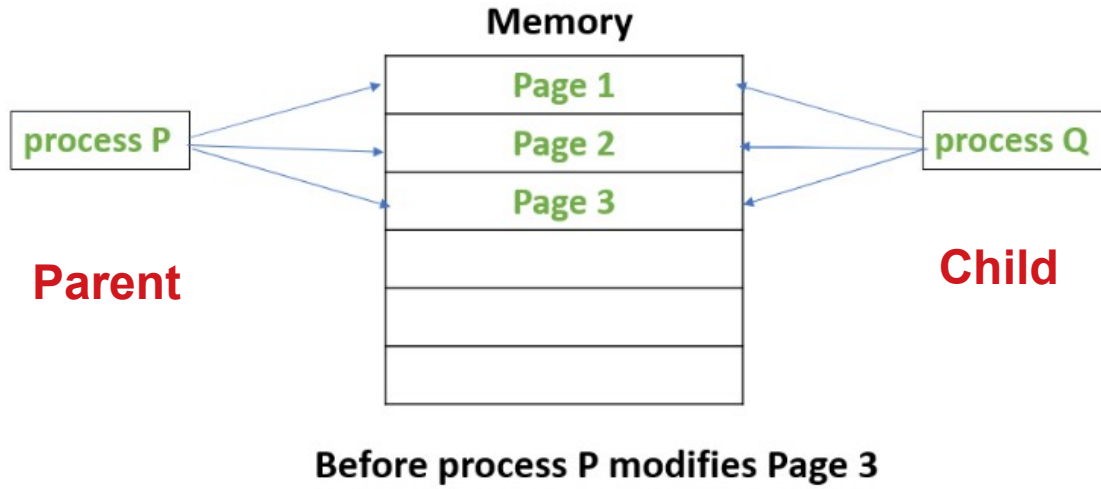


Copy On Write

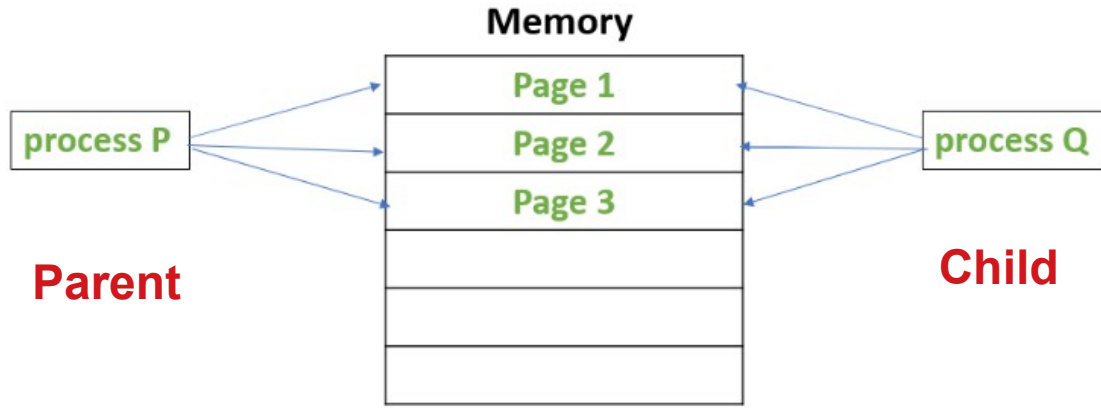
- Child created is an exact replica of the parent process
- Page tables of the parent duplicated in the child
- New pages created only when parent (or child) modifies data
 - Postpone copying of pages as much as possible, thus optimizing performance
 - Thus, common code sections (like libraries) would be shared across processes.



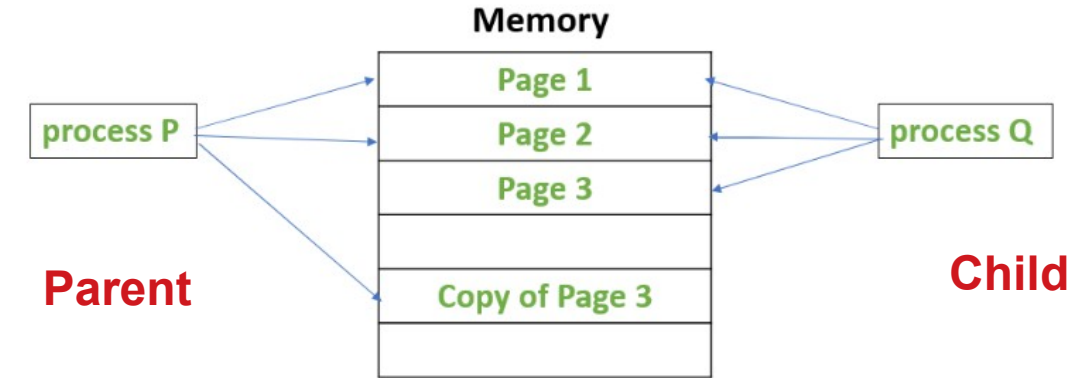
Copy On Write



Copy On Write

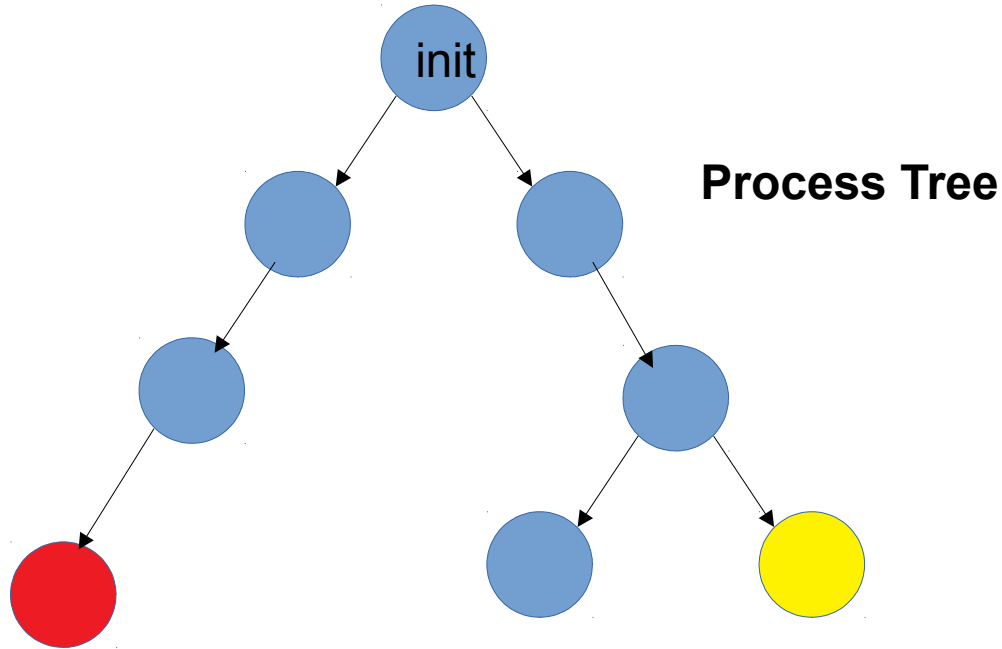


Child



Child

Process Tree



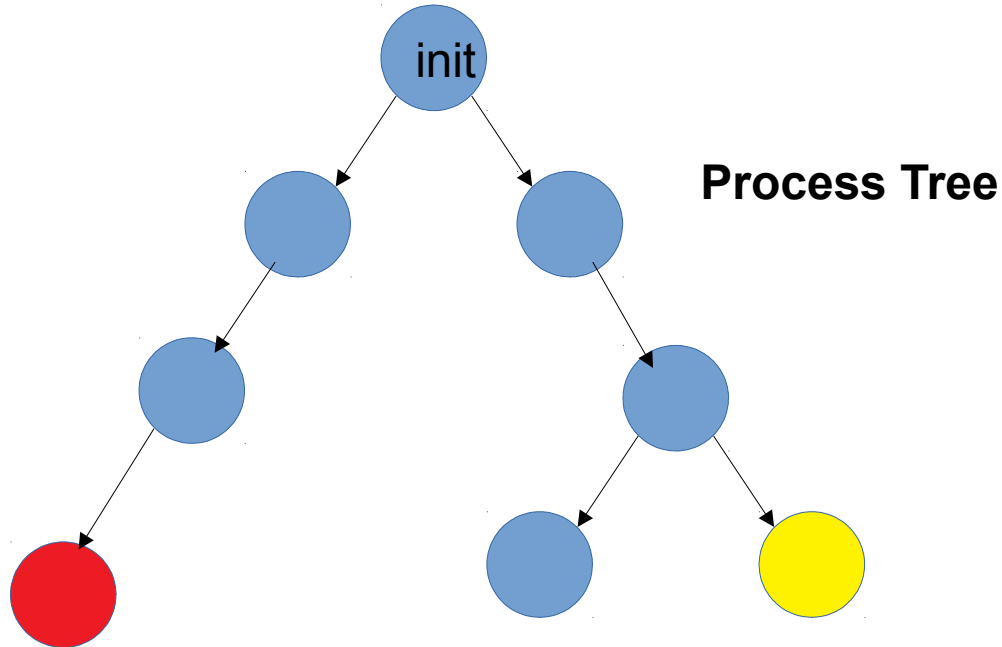
:
SSLEncryption()
:

Virtual Memory
(process 2)

:
SSLEncryption()
:

Virtual Memory
(process 1)

Process Tree



Process Tree



Virtual Memory
(process 2)

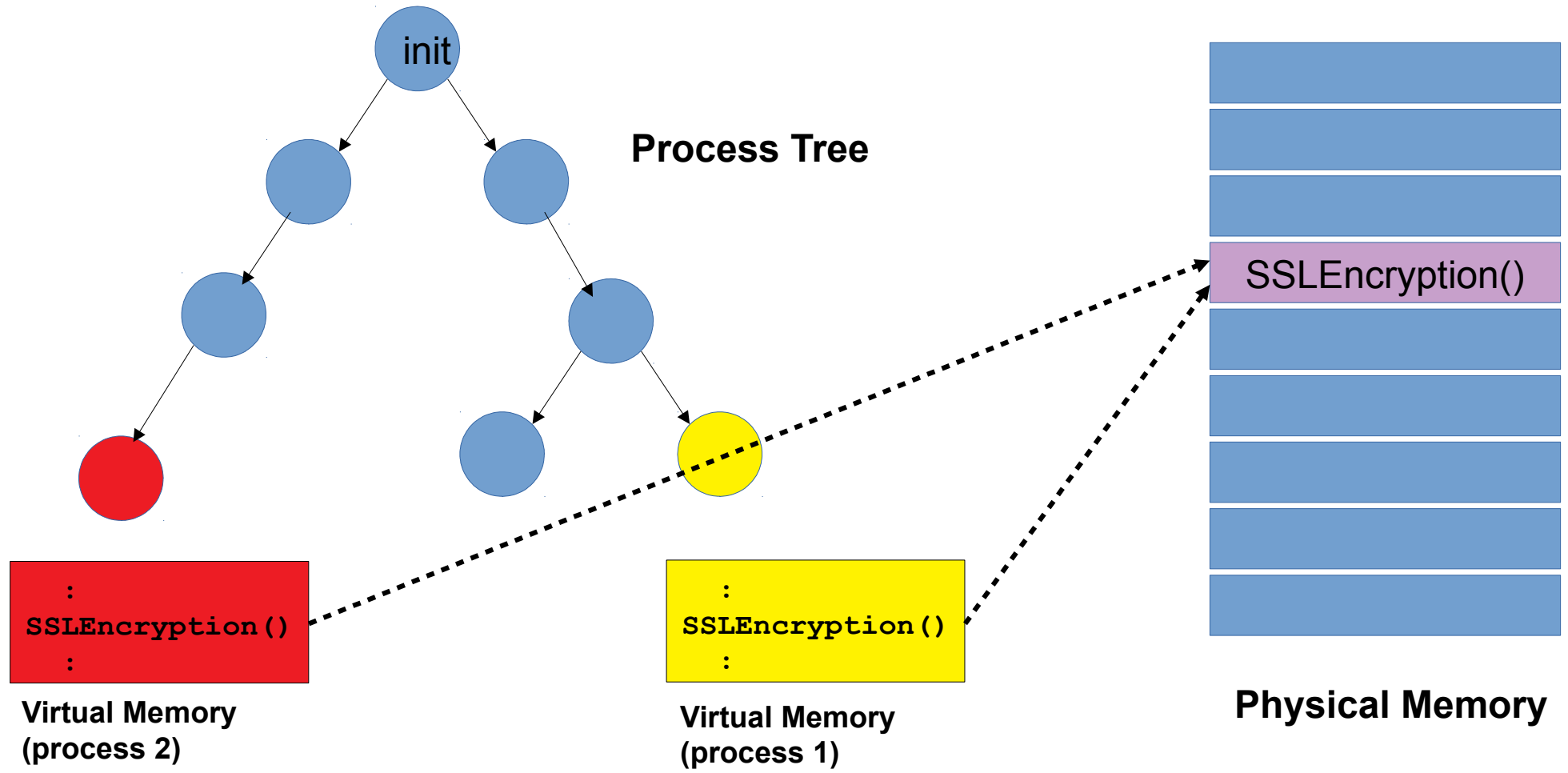


Virtual Memory
(process 1)

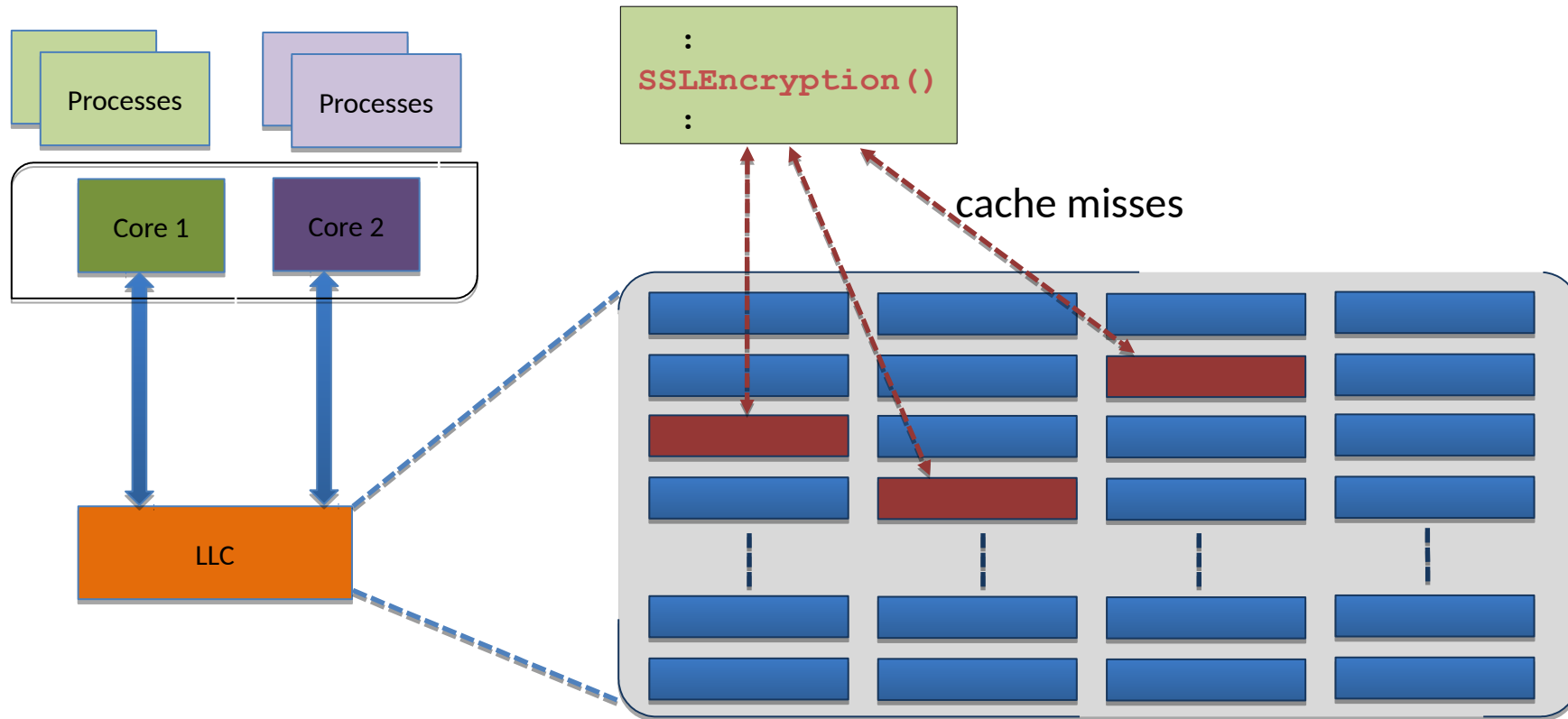


Physical Memory

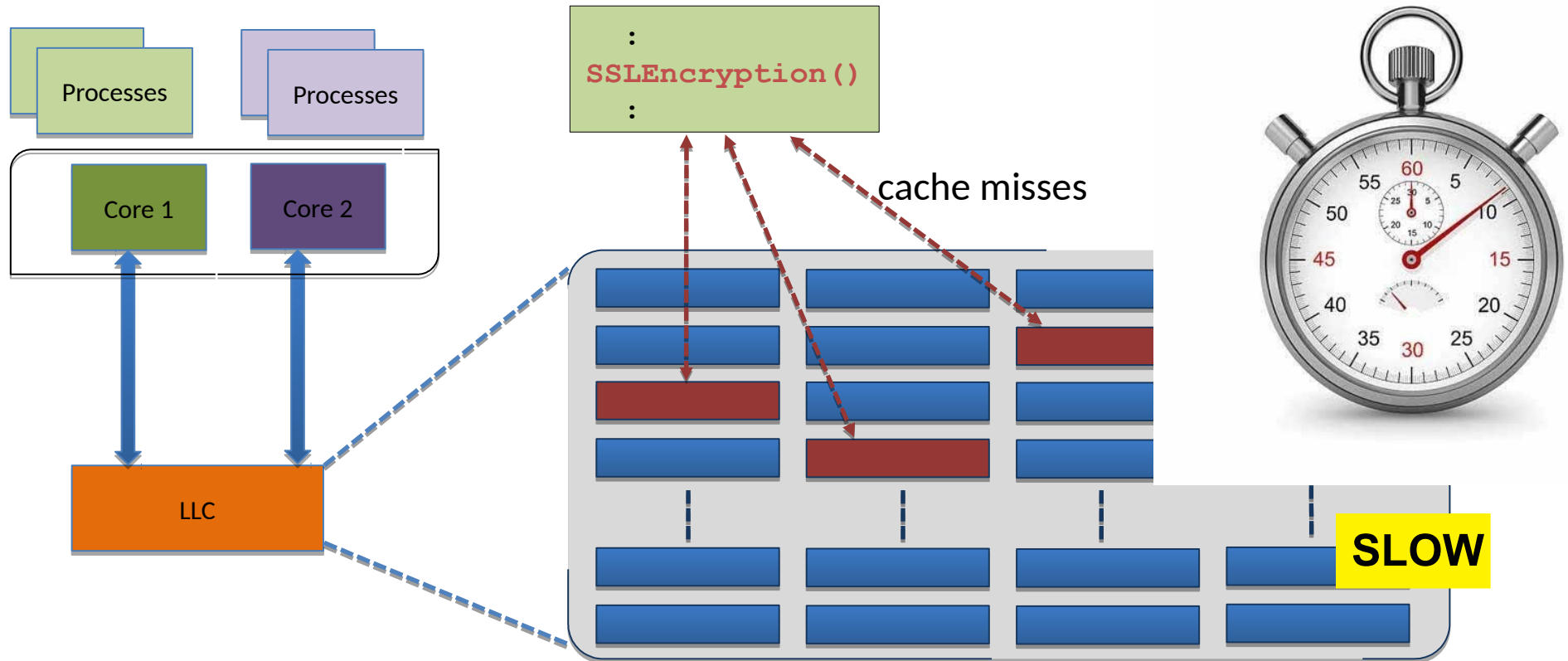
Process Tree



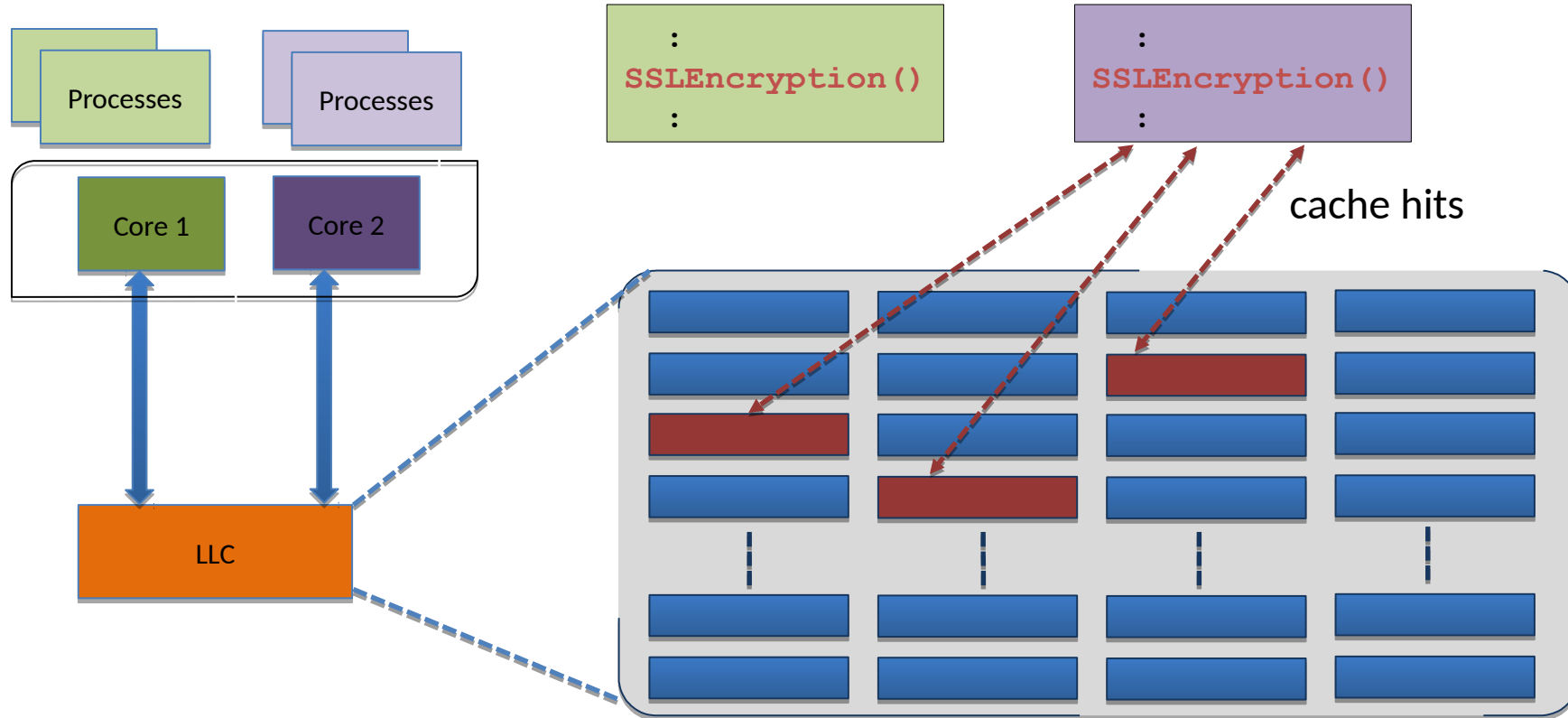
Interaction with LLC



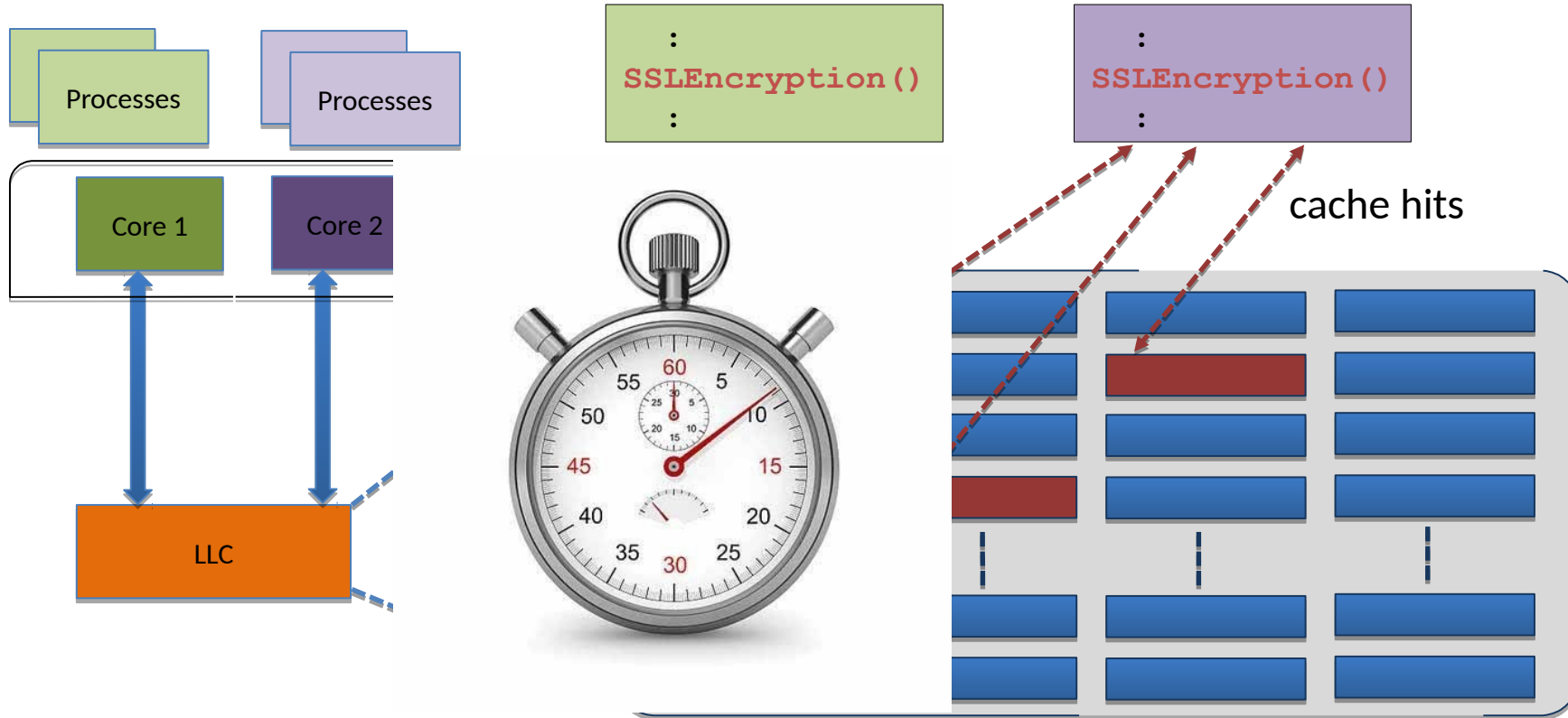
Interaction with LLC



Interaction with LLC



Interaction with LLC



FAST

Flush + Reload Attack

Part of an encryption algorithm

```

1 function exponent( $b, e, m$ )
2 begin
3    $x \leftarrow 1$ 
4   for  $i \leftarrow |e| - 1$  downto 0 do
5      $x \leftarrow x^2$ 
6      $x \leftarrow x \bmod m$ 
7     if ( $e_i = 1$ ) then
8        $x \leftarrow xb$ 
9      $x \leftarrow x \bmod m$ 
10  endif
11  done
12  return  $x$ 
13 end

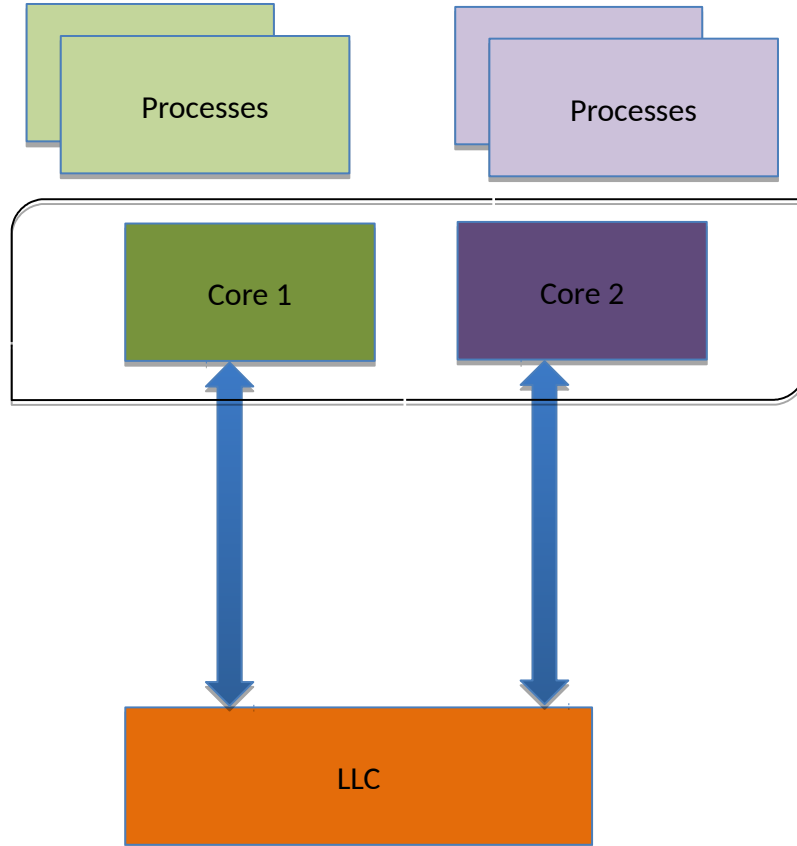
```

executed only when $e_i = 1$

clflush Instruction

Takes an address as input.
Flushes that address from all caches
clflush (line 8)

Flush + Reload Attack

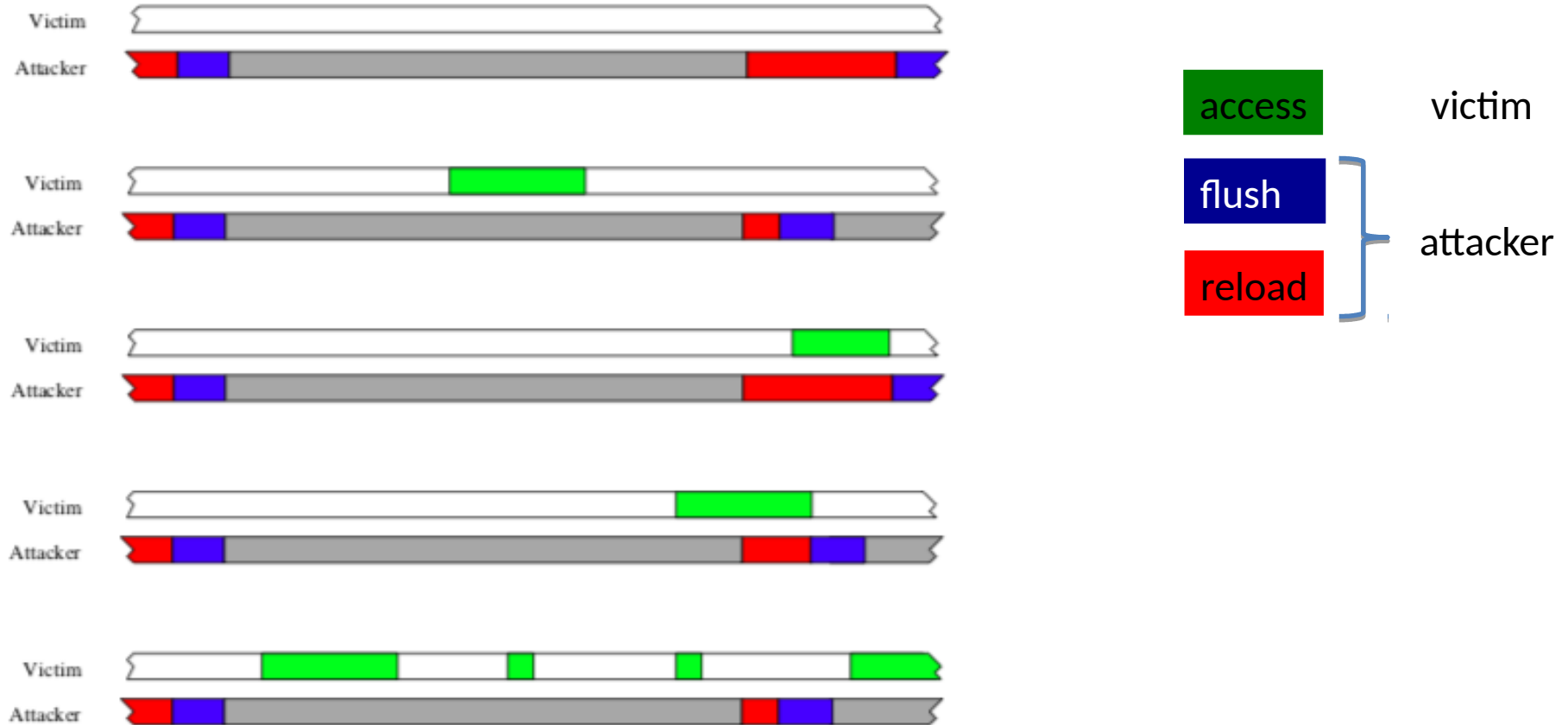


```
:  
SSLEncryption()  
:
```

```
:  
Clflush(line 8)  
:
```

```
1 function exponent( $b, e, m$ )  
2 begin  
3  $x \leftarrow 1$   
4 for  $i \leftarrow |e| - 1$  downto 0 do  
5  $x \leftarrow x^2$   
6  $x \leftarrow x \bmod m$   
7 if ( $e_i = 1$ ) then  
8  $x \leftarrow xb$   
9  $x \leftarrow x \bmod m$   
10 endif  
11 done  
12 return  $x$   
13 end
```

Flush + Reload Attack



Flush + Reload Attack

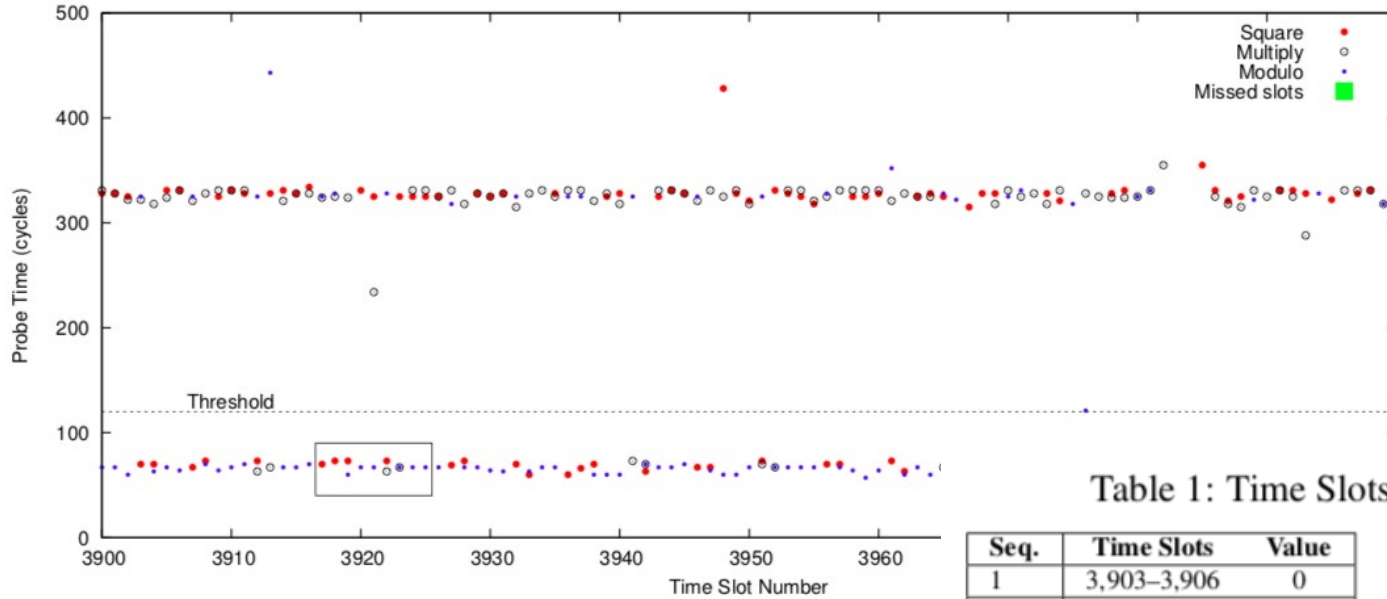


Table 1: Time Slots for Bit Sequence

Seq.	Time Slots	Value
1	3,903–3,906	0
2	3,907–3,916	1
3	3,917–3,926	1
4	3,927–3,931	0
5	3,932–3,935	0
6	3,936–3,945	1
7	3,946–3,955	1

Seq.	Time Slots	Value
8	3,956–3,960	0
9	3,961–3,969	1
10	3,970–3,974	0
11	3,975–3,979	0
12	3,980–3,988	1
13	3,989–3,998	1

Flush + Reload Attack : Counter

- Do not use copy-on-write
 - Implemented by cloud providers
- Permission checks for clflush
 - Do we need clflush?
- Non-inclusive cache memories
 - AMD
 - Intel i9 versions
- Fuzzing Clocks
- Software Diversification
 - Permute location of objects in memory (statically and dynamically)

Cache Collision Attacks

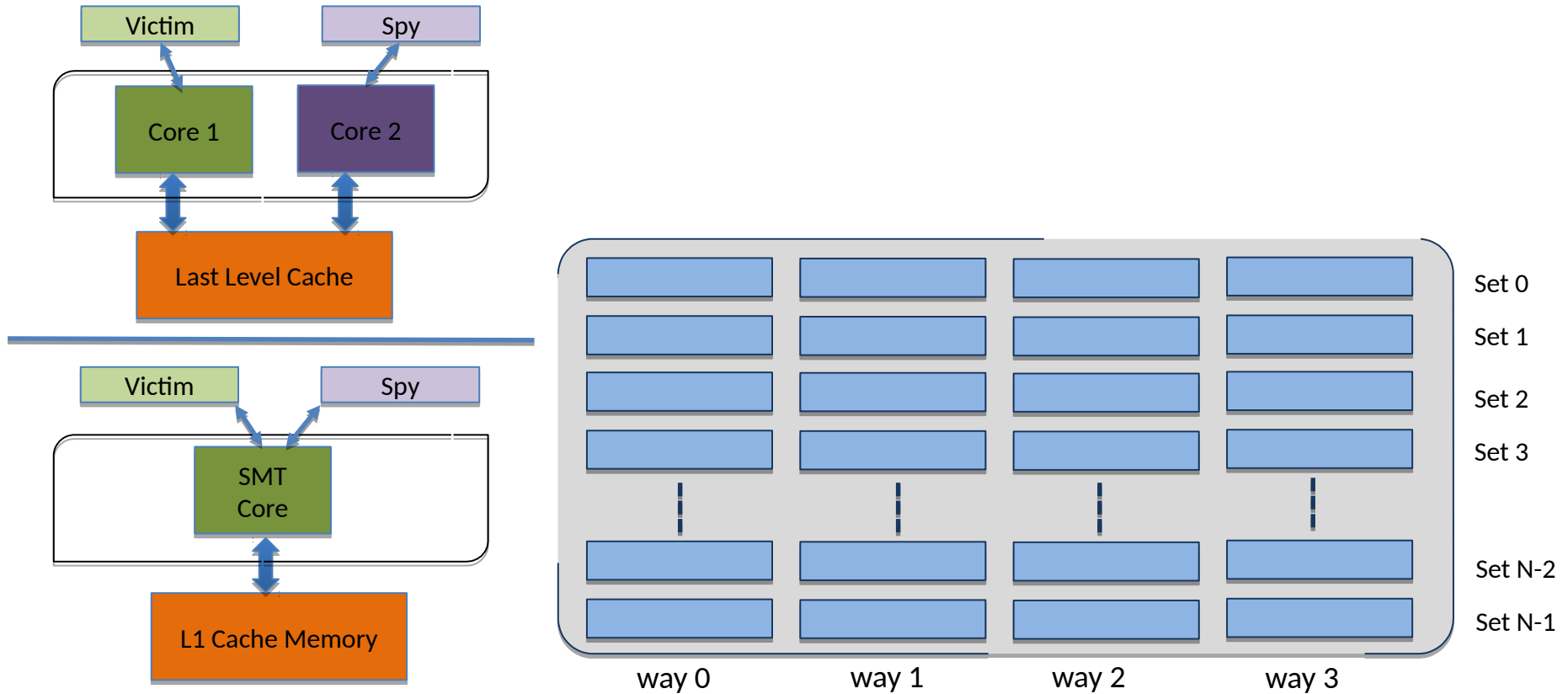
Prime + Probe Attack



Cache Collision Attacks

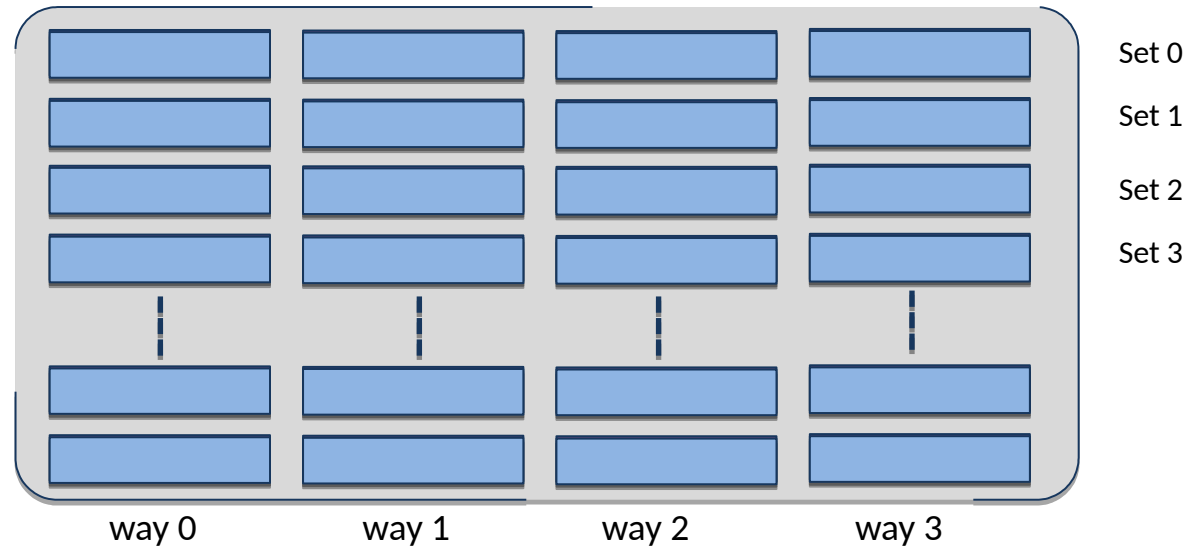
- External Collision Attacks
 - Prime + Probe Attack
- Internal Collision Attacks
 - Time Driven Attacks

Prime + Probe Attack



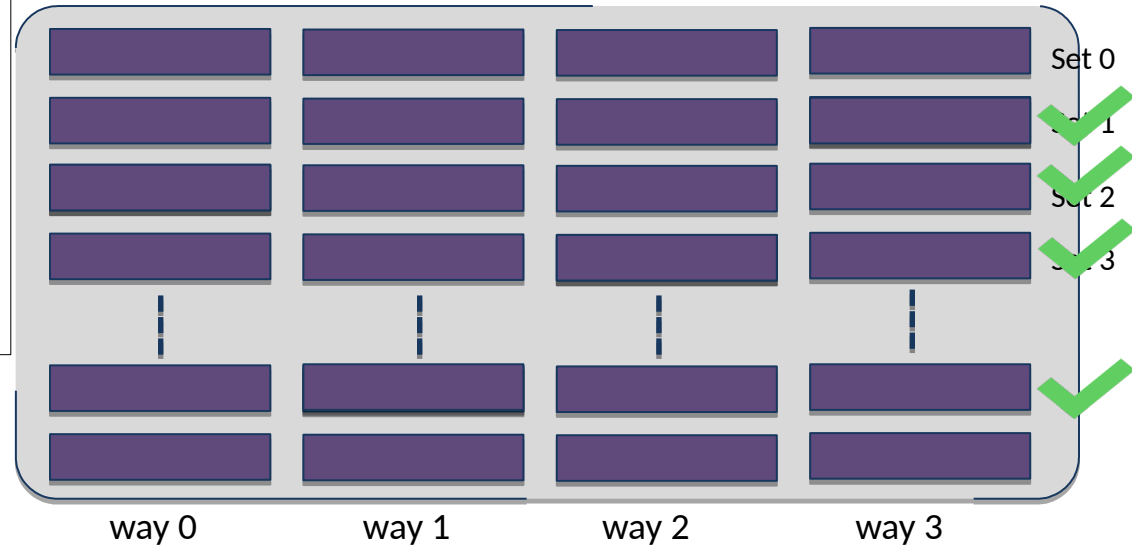
Prime + Probe Attack

```
While(1){  
  for(each cache set){  
    start = time();  
    access all cache ways  
    end = time();  
    access_time = end - start  
  }  
  wait for some time  
}
```



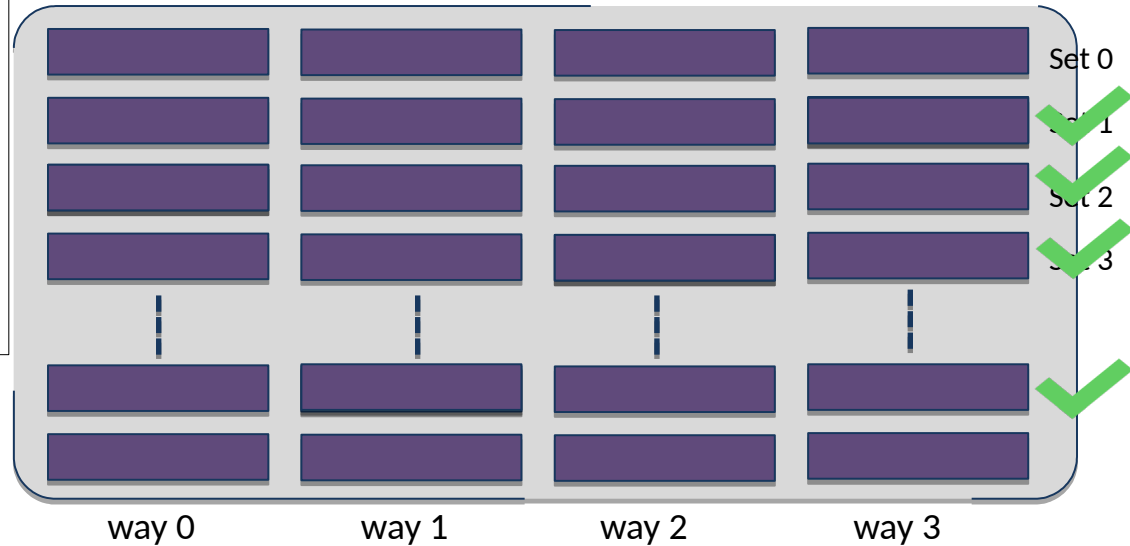
Prime + Probe Attack

```
While(1){  
  for(each cache set){  
    start = time();  
    access all cache ways  
    end = time();  
    access_time = end - start  
  }  
  wait for some time  
}
```



Prime + Probe Attack

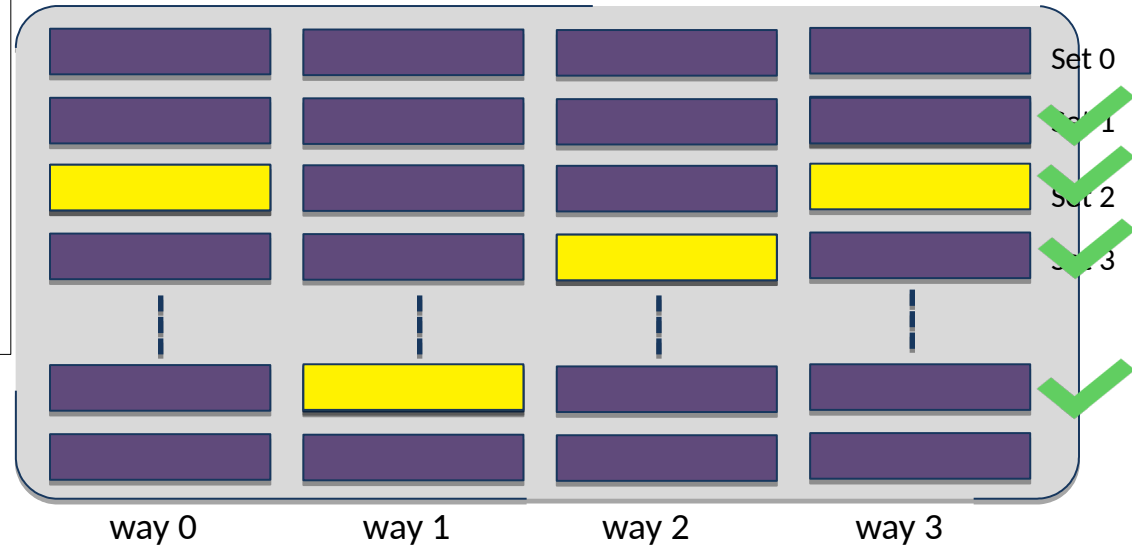
```
While(1){  
  for(each cache set){  
    start = time();  
    access all cache ways  
    end = time();  
    access_time = end - start  
  }  
  wait for some time  
}
```



PRIME

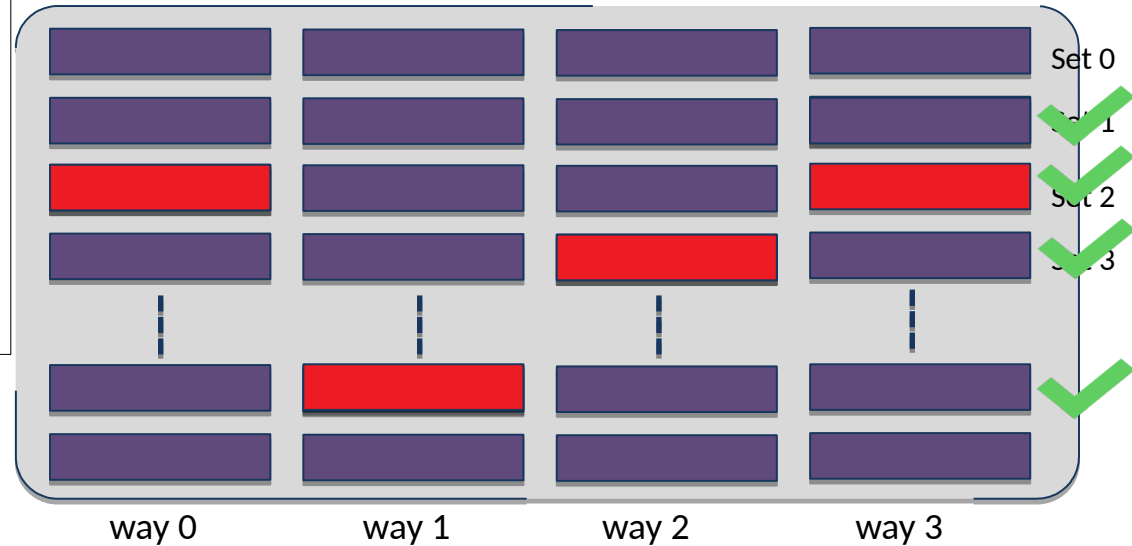
Prime + Probe Attack

```
While(1){  
  for(each cache set){  
    start = time();  
    access all cache ways  
    end = time();  
    access_time = end - start  
  }  
  wait for some time  
}
```



Prime + Probe Attack

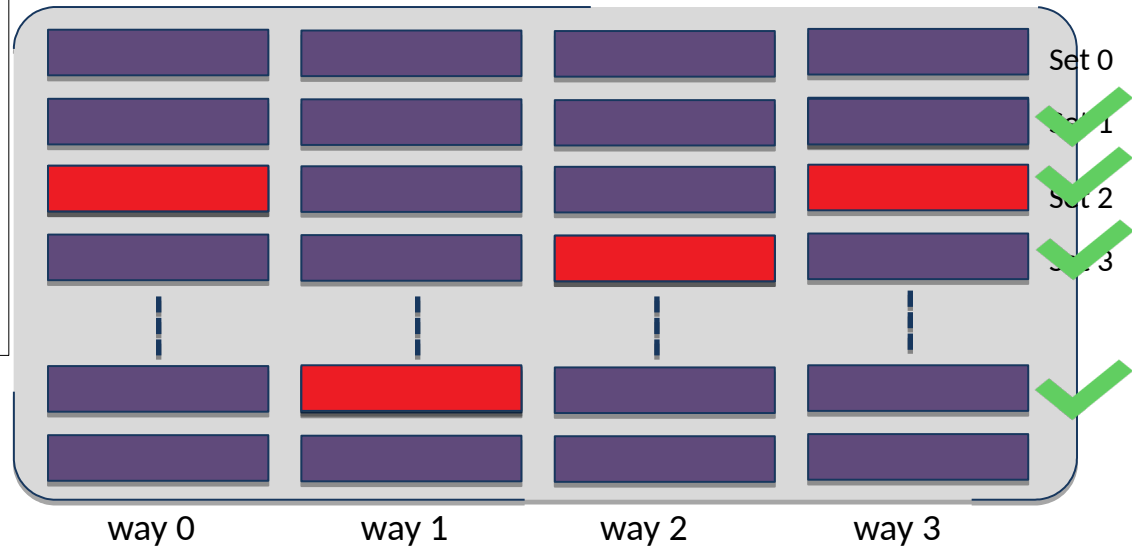
```
While(1){  
  for(each cache set){  
    start = time();  
    access all cache ways  
    end = time();  
    access_time = end - start  
  }  
  wait for some time  
}
```



Prime + Probe Attack

PROBE

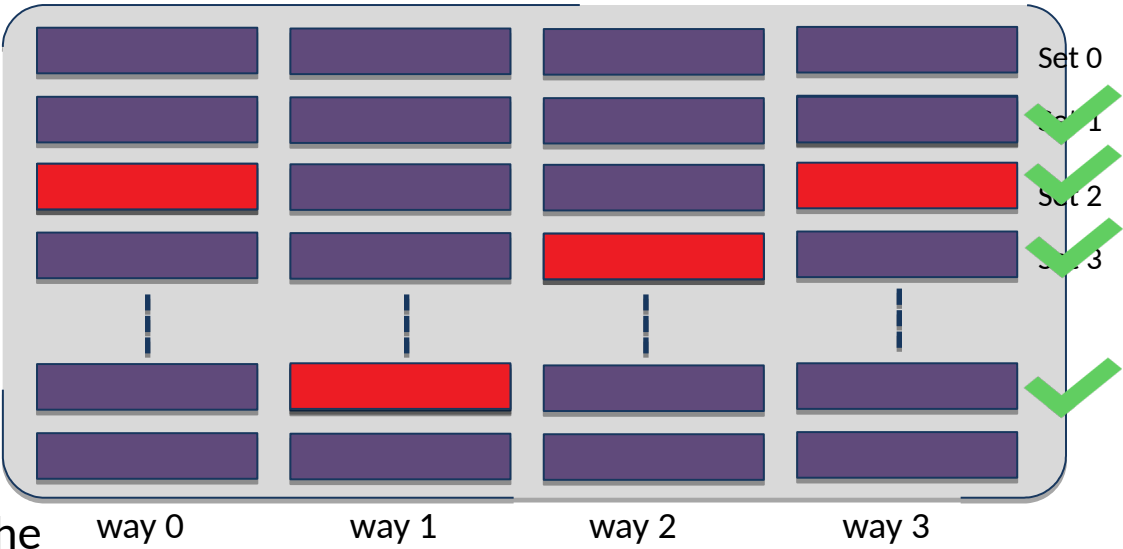
```
While(1){  
  for(each cache set){  
    start = time();  
    access all cache ways  
    end = time();  
    access_time = end - start  
  }  
  wait for some time  
}
```



Prime + Probe Attack

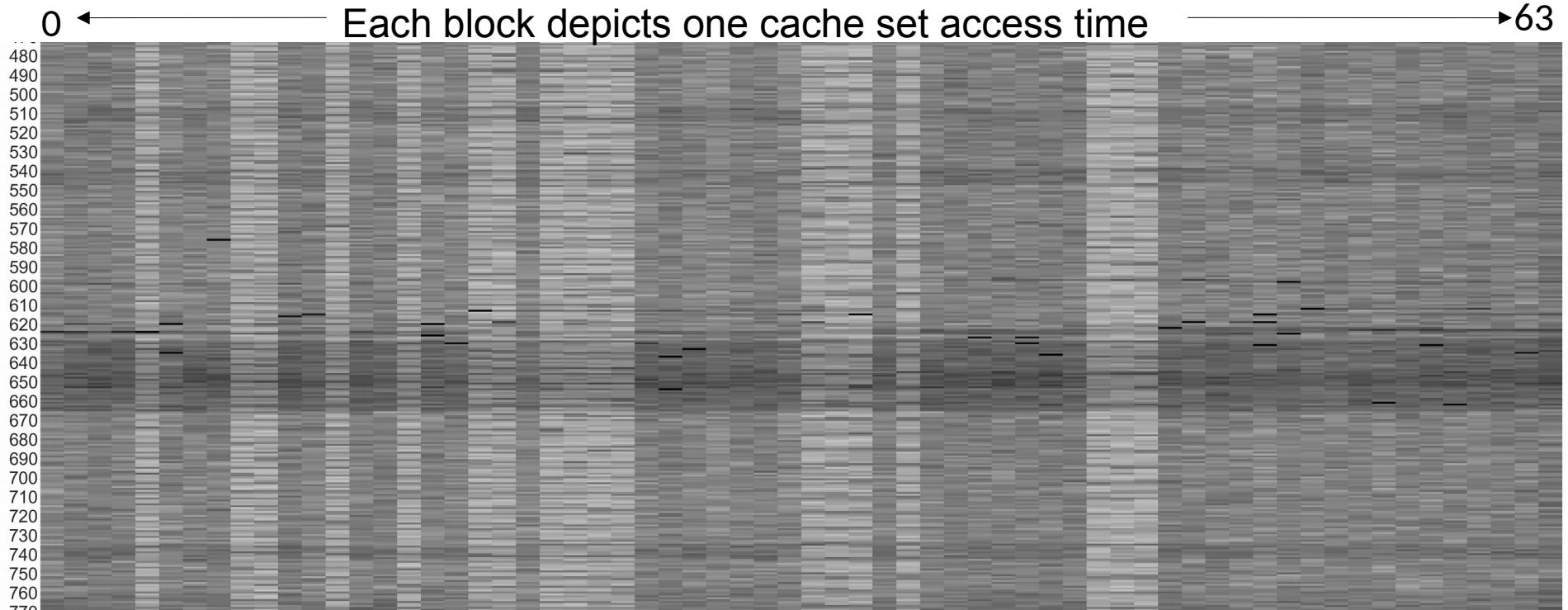
PROBE

```
While(1){  
  for(each cache set){  
    start = time();  
    access all cache ways  
    end = time();  
    access_time = end - start  
  }  
  wait for some time  
}
```



Time taken by sets that have victim data is more due to the cache misses

Prime + Probe Attack



Each row is an iteration of the while loop; darker shades imply higher memory access time

Example Prime+Probe: Cryptography

```
char Lookup[] = {x, x, x, . . . x};

char RecvDecrypt(socket){
    char key = 0x12;
    char pt, ct;

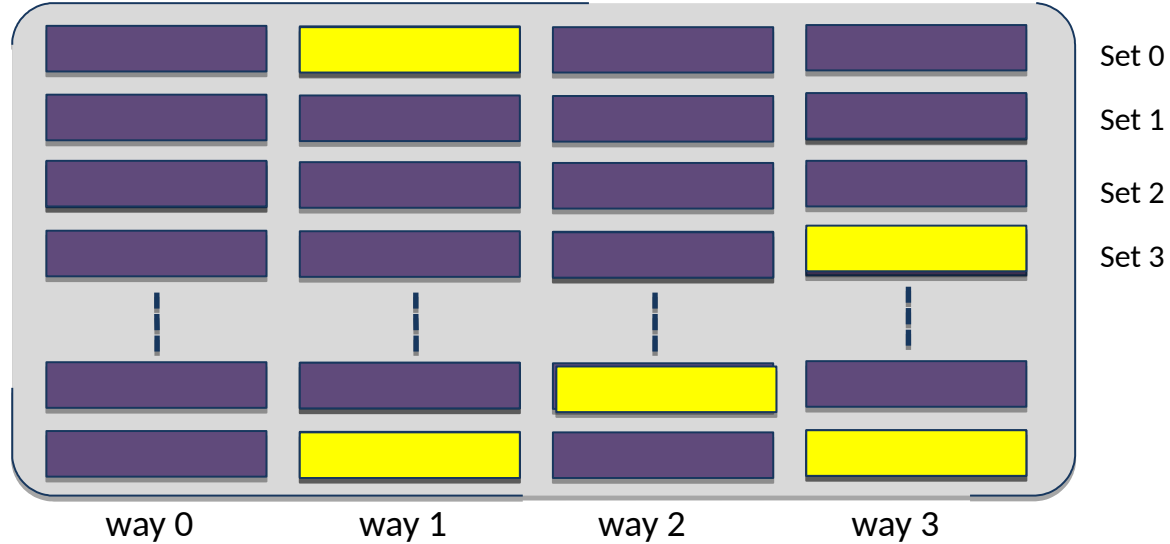
    read(socket, &ct, 1);
    pt = Lookup[key ^ ct];
    return pt;
}
```

Key dependent memory accesses

The attacker know the address of Lookup and the ciphertext (ct)
The memory accessed in Lookup depends on the value of key
Given the set number, one can identify bits of key ^ ct.

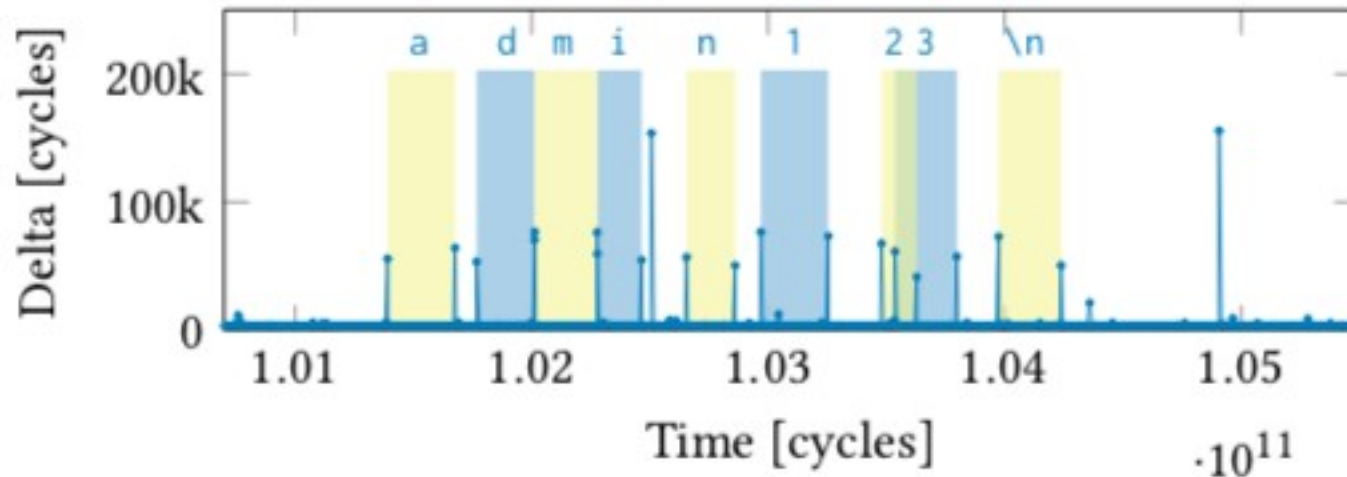
Example Prime+Probe: Keystroke Sniffing

- Keystroke -- interrupt -- kernel mode switch -- ISR execution -- add to keyboard buffer -- ... -- return from interrupt



Example Prime+Probe: Keystroke Sniffing

- Regular disturbance seen in Probe Time Plot
- Period between disturbance used to predict passwords



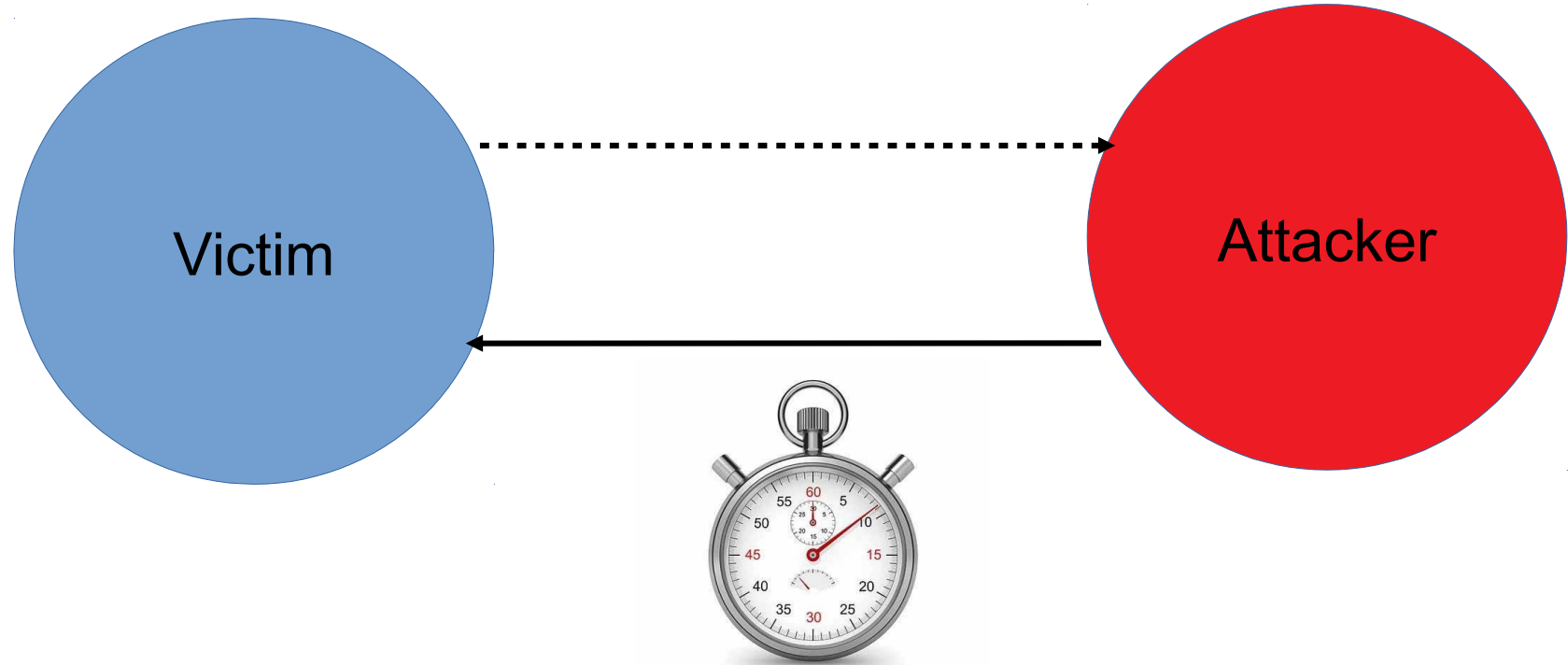
Svetlana Pinet, Johannes C. Ziegler, and F.-Xavier Alario. 2016. Typing Is Writing: Linguistic Properties Modulate Typing Execution. *Psychon Bull Rev* 23, 6

Cache Collision Attacks

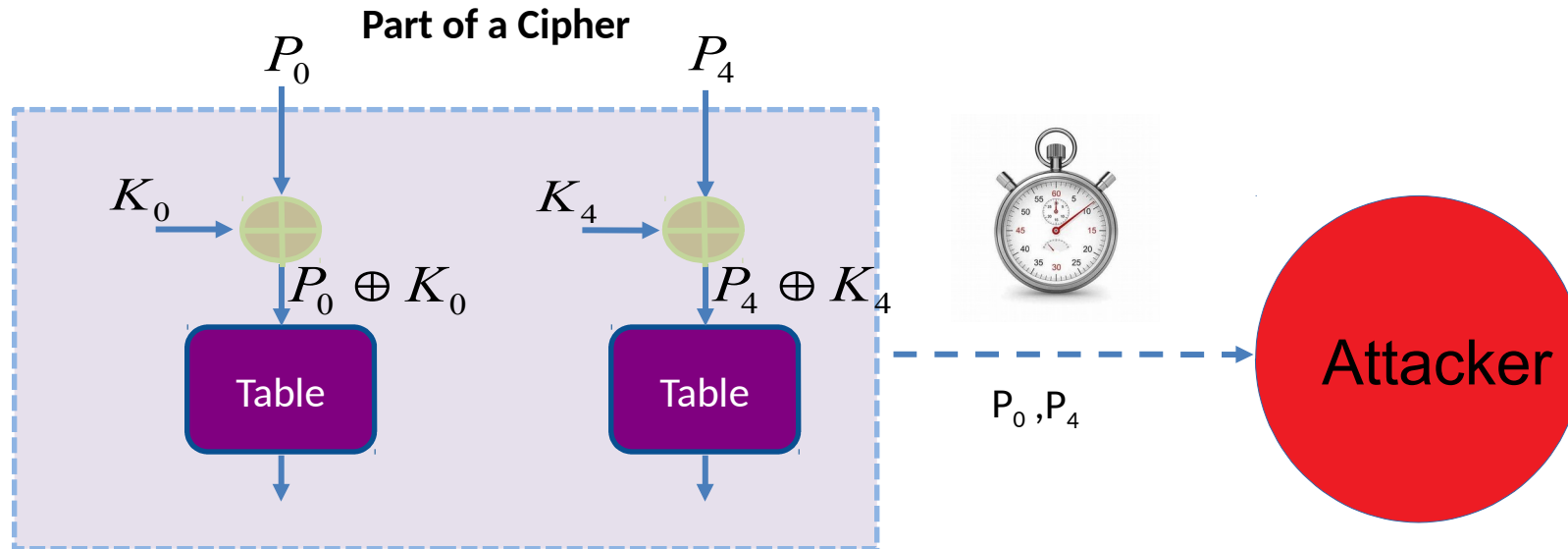
Time Driven Attacks



Time Driven Attacks



Internal Collision : Cipher



If cache hit (less time) :

$$\langle P_0 \oplus K_0 \rangle = \langle P_4 \oplus K_4 \rangle$$
$$\Rightarrow \langle K_0 \oplus K_4 \rangle = \langle P_0 \oplus P_4 \rangle$$

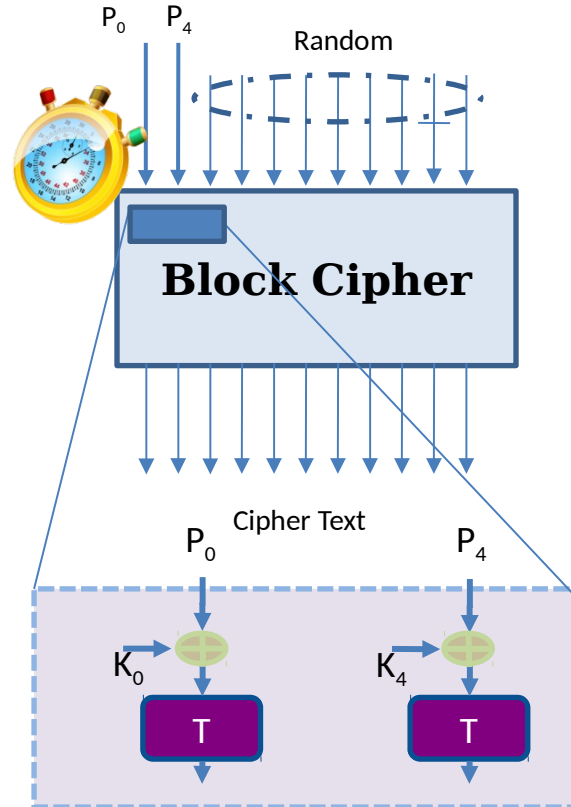
If cache miss (more time):

$$\langle P_0 \oplus K_0 \rangle \neq \langle P_4 \oplus K_4 \rangle$$
$$\Rightarrow \langle K_0 \oplus K_4 \rangle \neq \langle P_0 \oplus P_4 \rangle$$

Internal Collision : Cipher

Suppose
($K_0 = 00$ and $K_4 = 50$)

- $P_0 = 0$, all other inputs are random
- Make N time measurements
- Segregate into buckets based on value of P_4
- Find average time of each bucket
- Find deviation of each average from overall average (DOM)



$$\langle K_0 \oplus K_4 \rangle = \langle P_0 \oplus P_4 \rangle$$

P4	Average Time	DOM
00	2945.3	1.8
10	2944.4	0.9
20	2943.7	0.2
30	2943.7	0.2
40	2944.8	1.3
50	2937.4	-6.3
60	2943.3	-0.2
70	2945.8	2.3
:	:	:
F0	2941.8	-1.7

Average : 2943.57
Maximum : -6.3

Questions

Cache Attacks

