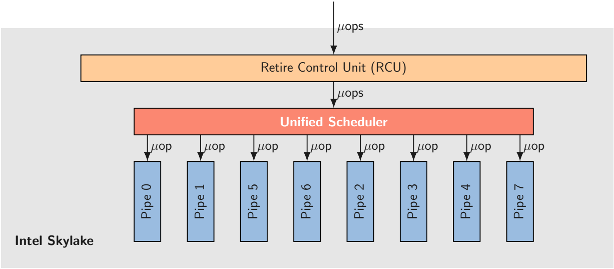
SQUIP

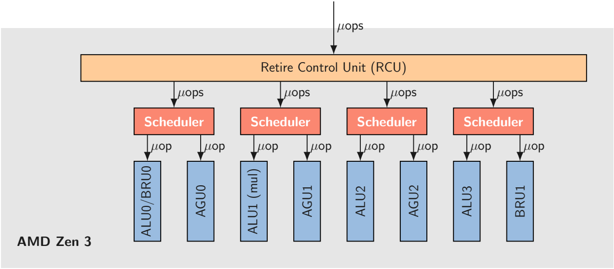
Exploiting the Scheduler Queue Contention Side Channel

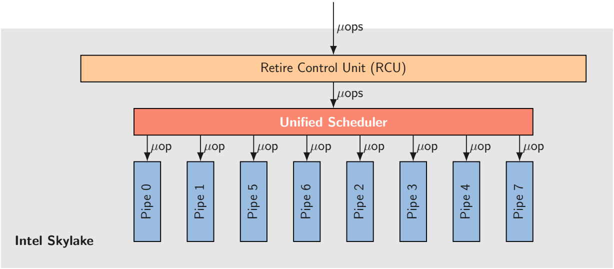
Stefan Gast [1] **Jonas Juffinger** [1] **Martin Schwarzl** [1] **Gururaj Saileshwar** [2] **Andreas Kogler** [1] **Simone Franza** [1] **Markus Köstl** [1] **Daniel Gruss** [1]

2023-05-23

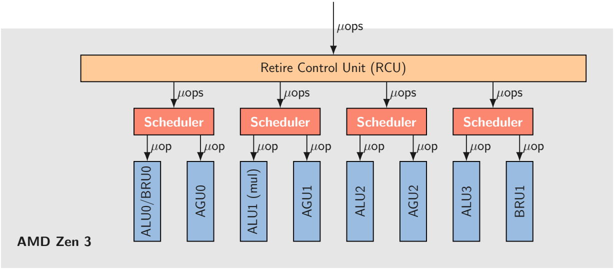
[1] Graz University of Technology
[2] Georgia Institute of Technology


μops
Retire Control Unit (RCU)
μops
Unified Scheduler
μop
μop
μop
μop
μop
μop
μop
μop
Pipe 0
Pipe 1
Pipe 5
Pipe 6
Pipe 2
Pipe 3
Pipe 4
Pipe 7
Intel Skylake

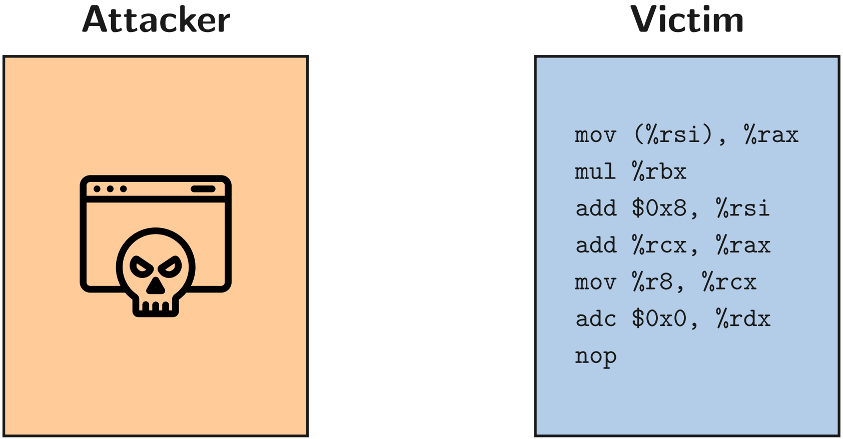

μops
Retire Control Unit (RCU)
μops
μops
μops
μops
Scheduler
Scheduler
Scheduler
Scheduler
μop
μop
μop
μop
μop
μop
μop
μop
ALU0/BRU0
AGU0
ALU1 (mul)
AGU1
ALU2
AGU2
ALU3
BRU1
AMD Zen 3



Can we exploit this difference?

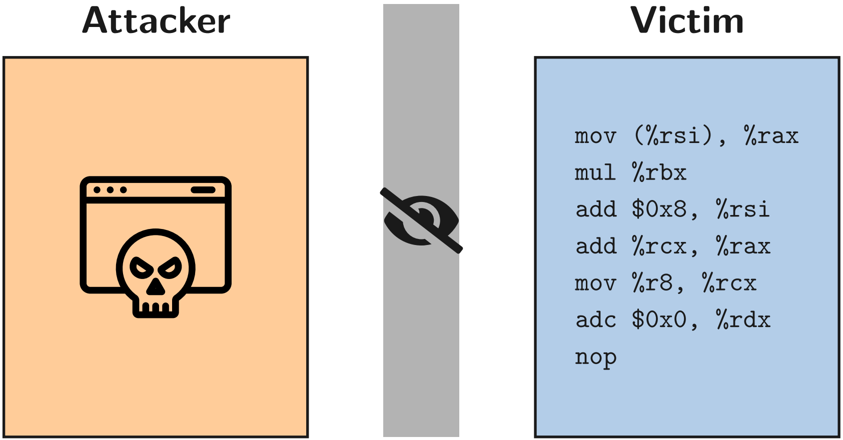


Goal



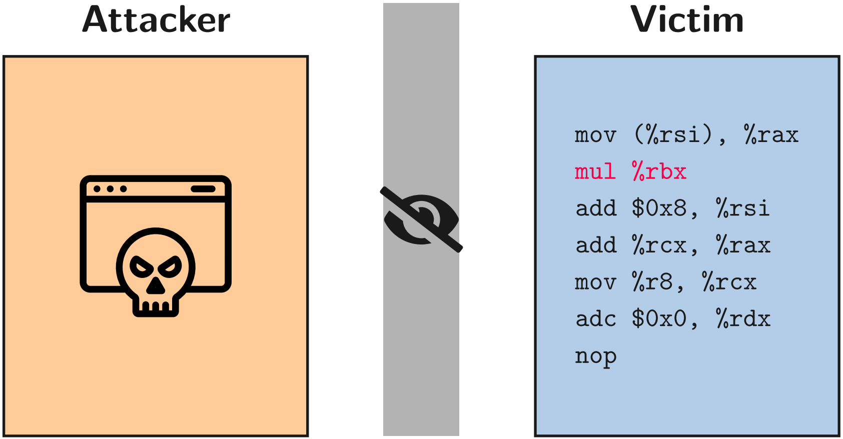
```
mov (%rsi), %rax
mul %rbx
add $0x8, %rsi
add %rcx, %rax
mov %r8, %rcx
adc $0x0, %rdx
nop
```

Goal



```
mov (%rsi), %rax
mul %rbx
add $0x8, %rsi
add %rcx, %rax
mov %r8, %rcx
adc $0x0, %rdx
nop
```

Goal

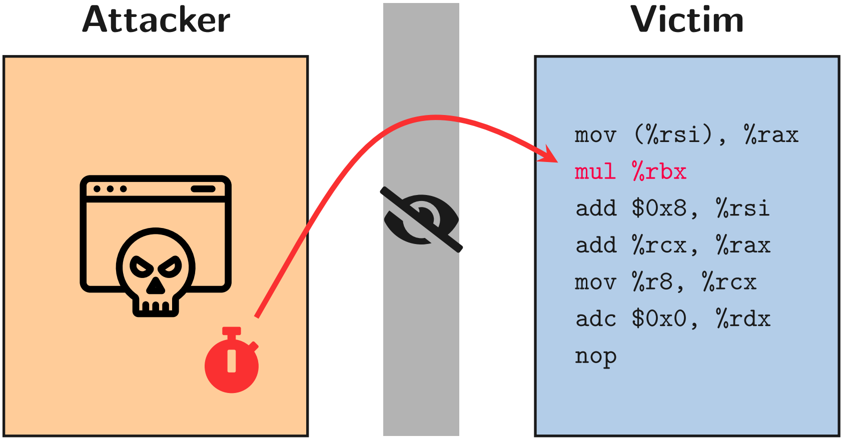


Attacker

Victim

```
mov (%rsi), %rax
mul %rbx
add $0x8, %rsi
add %rcx, %rax
mov %r8, %rcx
adc $0x0, %rdx
nop
```

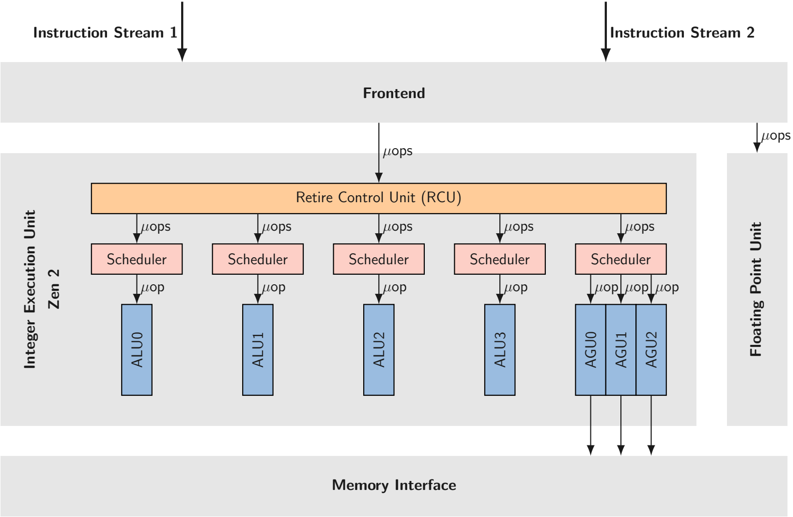
Goal

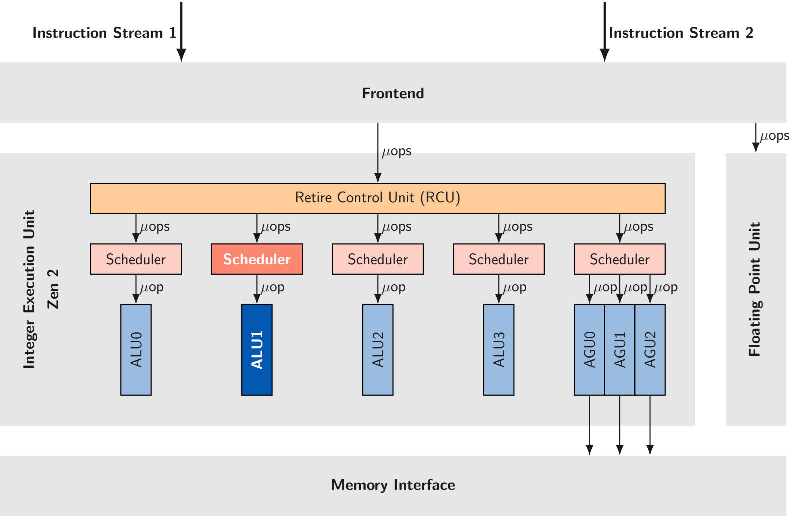


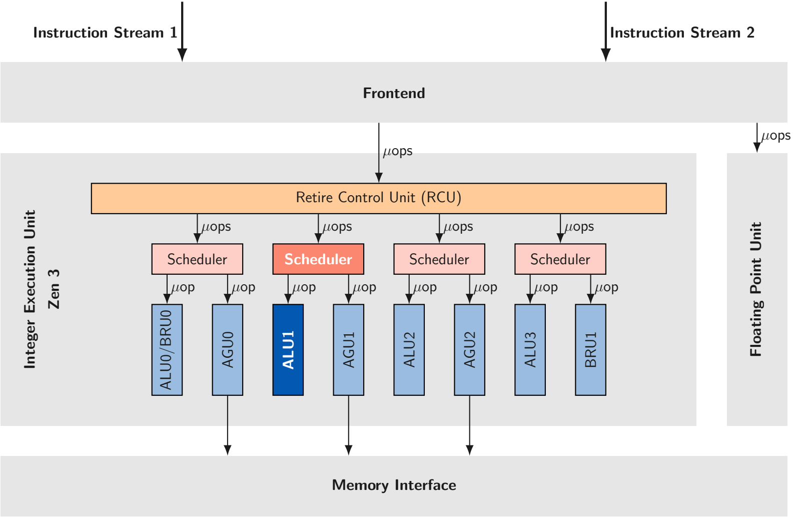
Attacker

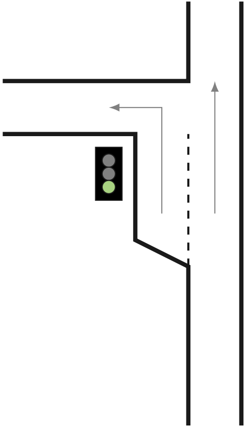
Victim

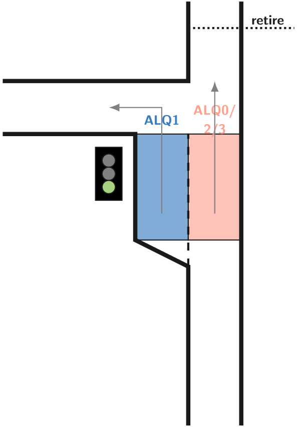
```
mov (%rsi), %rax
mul %rbx
add $0x8, %rsi
add %rcx, %rax
mov %r8, %rcx
adc $0x0, %rdx
nop
```

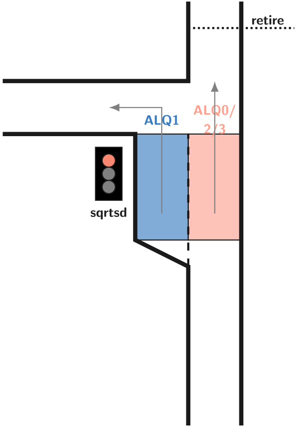

Instruction Stream 1
Instruction Stream 2
Frontend
μops
μops
Integer Execution Unit
Zen 2
Retire Control Unit (RCU)
μops
μops
μops
μops
μops
Scheduler
Scheduler
Scheduler
Scheduler
Scheduler
μop
μop
μop
μop
μop
μop
μop
ALU0
ALU1
ALU2
ALU3
AGU0
AGU1
AGU2
Floating Point Unit
Memory Interface

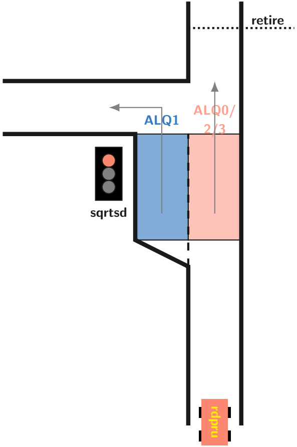

Instruction Stream 1
Instruction Stream 2
Frontend
μops
μops
Integer Execution Unit
Zen 2
Retire Control Unit (RCU)
μops
μops
μops
μops
μops
Scheduler
Scheduler
Scheduler
Scheduler
Scheduler
μop
μop
μop
μop
μop
μop
μop
ALU0
ALU1
ALU2
ALU3
AGU0
AGU1
AGU2
Floating Point Unit
Memory Interface

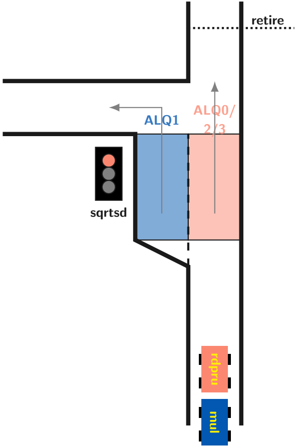

Instruction Stream 1
Instruction Stream 2
Frontend
μops
μops
Integer Execution Unit
Zen 3
Retire Control Unit (RCU)
μops
μops
μops
μops
Scheduler
Scheduler
Scheduler
Scheduler
μop
μop
μop
μop
μop
μop
μop
μop
ALU0/BRU0
AGU0
ALU1
AGU1
ALU2
AGU2
ALU3
BRU1
Floating Point Unit
Memory Interface

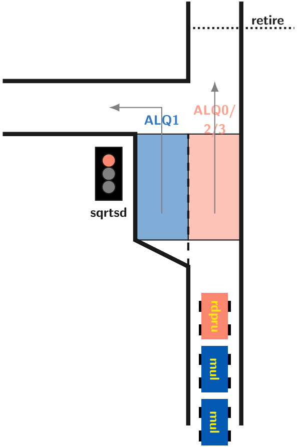


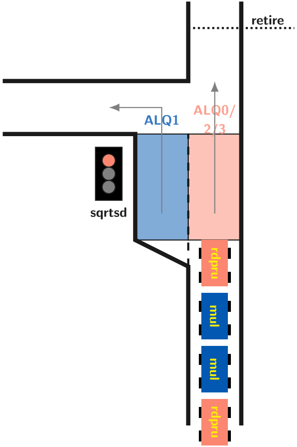

retire
ALQ1
ALQ0/
2/3

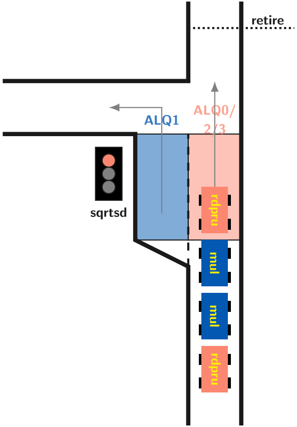

retire
ALQ1
ALQ0/
2/3
sqrtsd

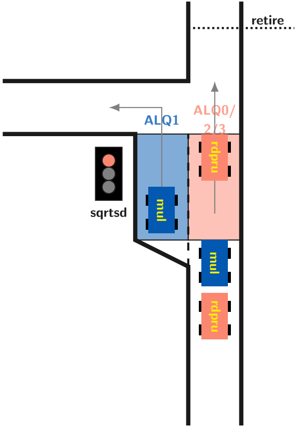

retire
ALQ1
ALQ0/
2/3
sqrtsd
rdpru

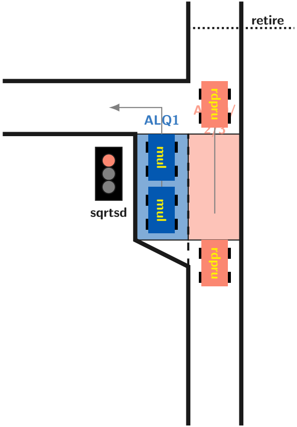

retire
ALQ1
ALQ0/
2/3
sqrtsd
rdpru
mul

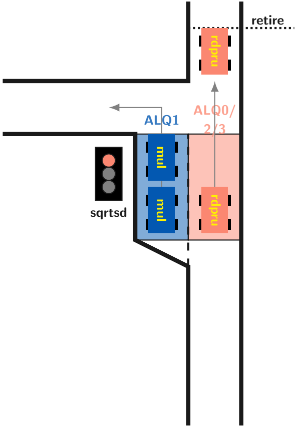

retire
ALQ1
ALQ0/
2/3
sqrtsd
rdpru
mul
mul

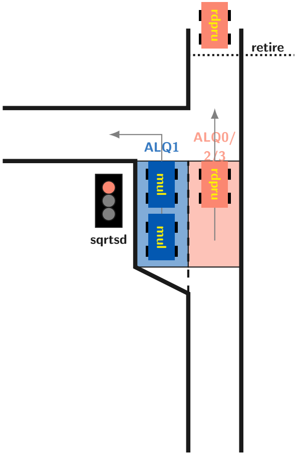

retire
ALQ1
ALQ0/
2/3
sqrtsd
rdpru
mul
mul
rdpru

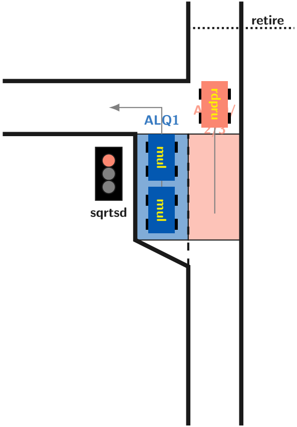

retire
ALQ1
ALQ0/
2/3
sqrtsd
rdpru
mul
mul
rdpru

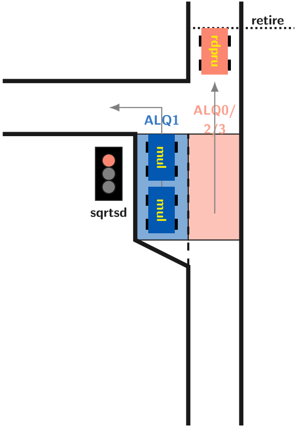

retire
ALQ1
ALQ0/
2/3
rdpru
mul
sqrtsd
mul
rdpru

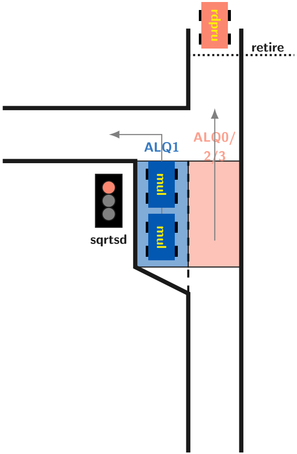

retire
rdpru
A /
2/3
ALQ1
mul
mul
sqrtsd
rdpru

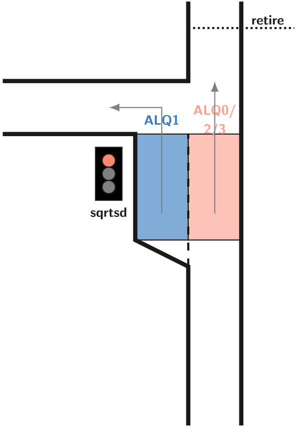

retire
rdpru
ALQ1
ALQ0/
2/3
mul
mul
rdpru
sqrtsd

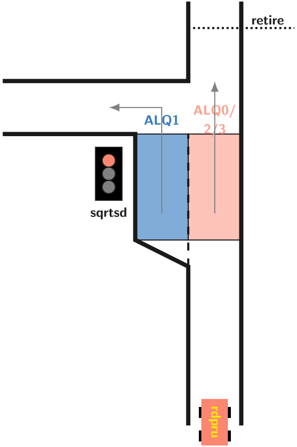

rdpru
retire
ALQ1
ALQ0/
2/3
mul
rdpru
mul
sqrtsd

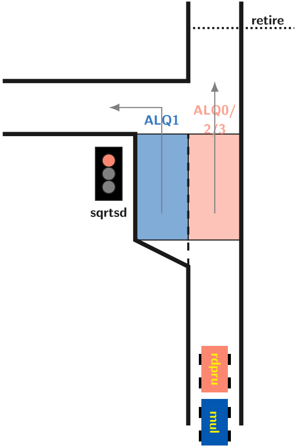

retire
rdpru
A /
2/3
ALQ1
mul
mul
sqrtsd

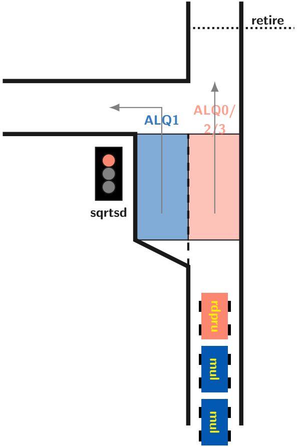

retire
rdpru
ALQ0/
2/3
ALQ1
mul
mul
sqrtsd

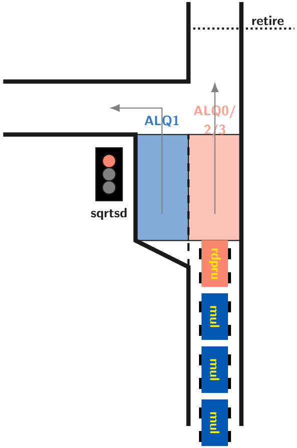

rdpru
retire
ALQ1
ALQ0/
2/3
mul
mul
sqrtsd

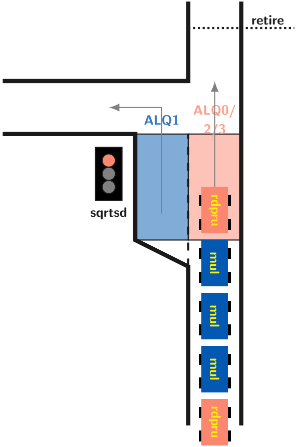

retire
ALQ1
ALQ0/
2/3
sqrtsd

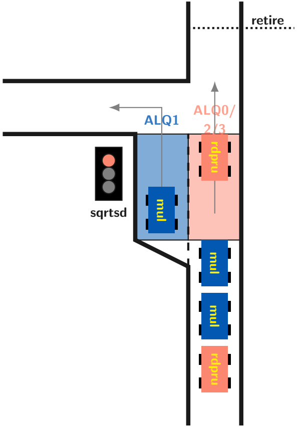

retire
ALQ1
ALQ0/
2/3
sqrtsd
rdpru

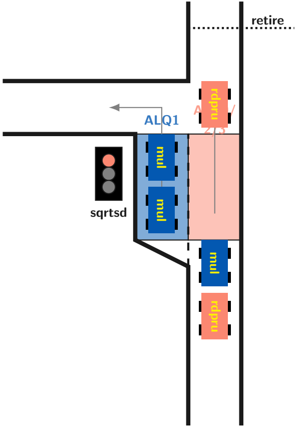

retire
ALQ1
ALQ0/
2/3
sqrtsd
rdpru
mul

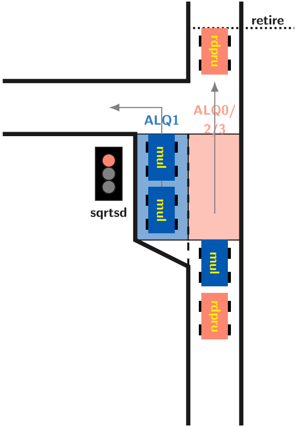

retire
ALQ1
ALQ0/
2/3
sqrtsd
rdpru
mul
mul

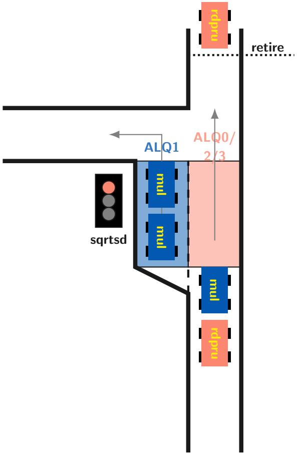

retire
ALQ1
ALQ0/
2/3
sqrtsd
rdpru
mul
mul
mul

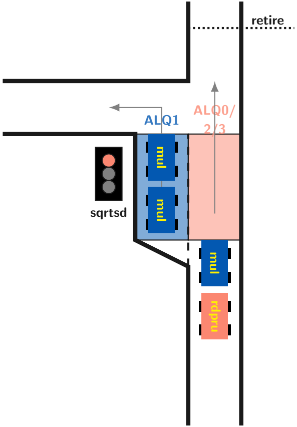

retire
ALQ1
ALQ0/
2/3
sqrtsd
rdpru
mul
mul
mul
rdpru


retire
ALQ1
ALQ0/
2/3
rdpru
mul
sqrtsd
mul
mul
rdpru

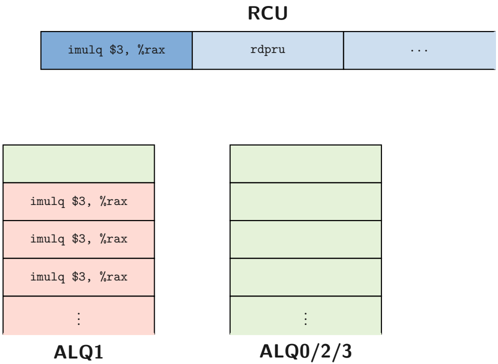

retire
rdpru
A /
2/3
ALQ1
mul
mul
sqrtsd
mul
rdpru


retire
rdpru
ALQ1
ALQ0/
2/3
mul
mul
sqrtsd
mul
rdpru


rdpru
retire
ALQ1
ALQ0/
2/3
mul
mul
sqrtsd
mul
rdpru


retire
ALQ0/
2/3
ALQ1
mul
mul
sqrtsd
mul
rdpru

What if a scheduler queue is full?



RCU

imulq $3, %rax | rdpru | ...

ALQ1

imulq $3, %rax

imulq $3, %rax

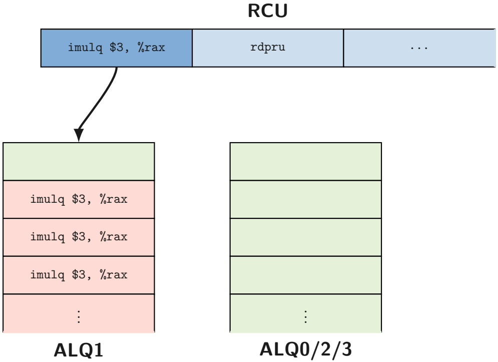
imulq $3, %rax

⋮

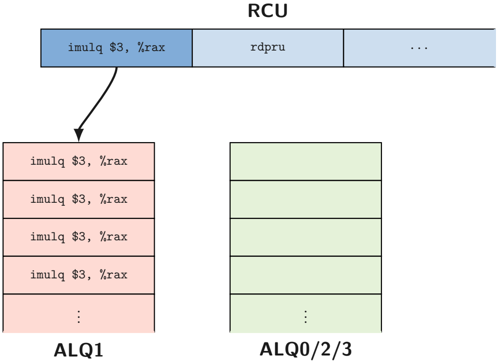
ALQ0/2/3

⋮

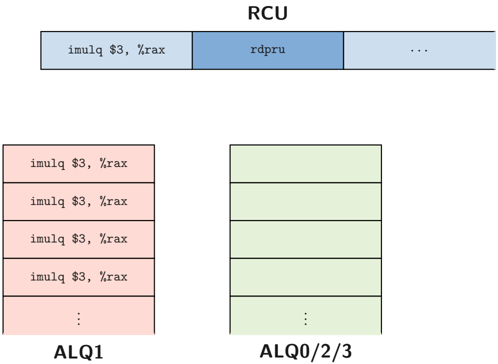
What if a scheduler queue is full?



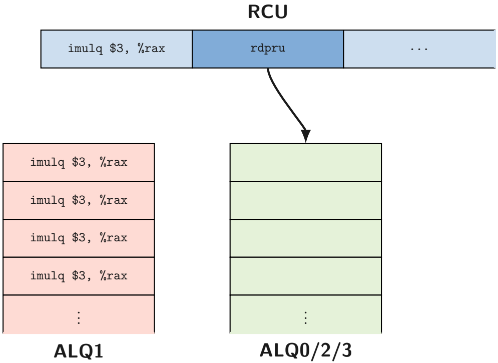
What if a scheduler queue is full?



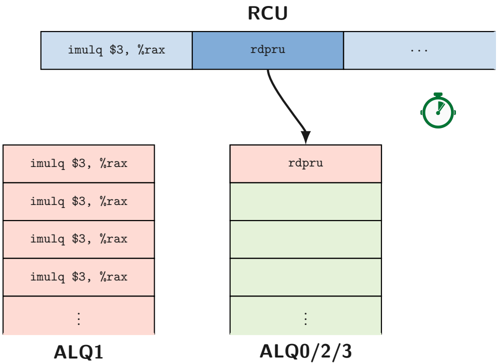
What if a scheduler queue is full?



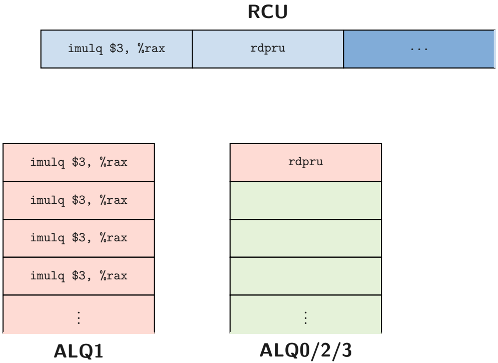
What if a scheduler queue is full?



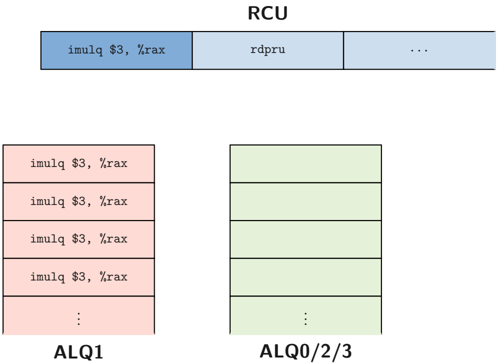
What if a scheduler queue is full?



What if a scheduler queue is full?



What if a scheduler queue is full?



RCU

imulq $3, %rax | rdpru | ...

imulq $3, %rax

imulq $3, %rax

imulq $3, %rax

imulq $3, %rax

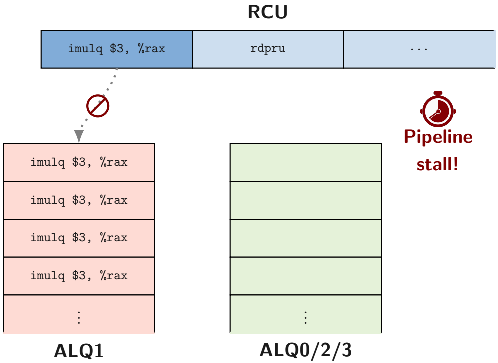
⋮

ALQ1

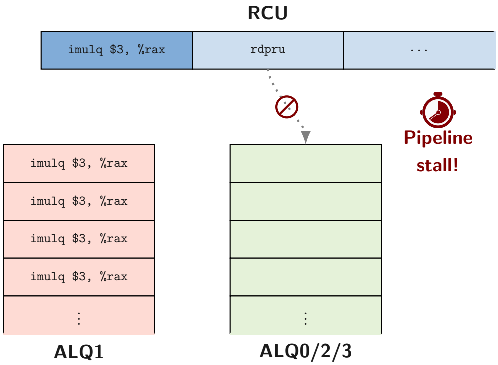
⋮

ALQ0/2/3

What if a scheduler queue is full?



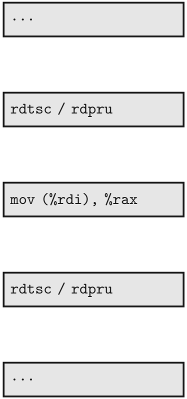
What if a scheduler queue is full?



Time Measurements and Out-of-Order Execution

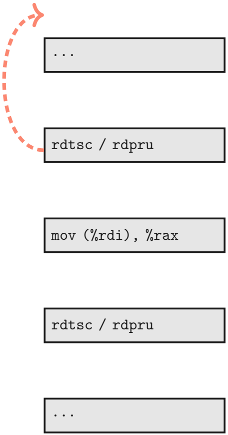
`rdtsc / rdpru`

Time Measurements and Out-of-Order Execution



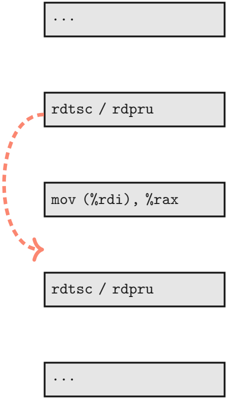
```
...
rdtsc / rdpru
mov (%rdi), %rax
rdtsc / rdpru
...
```

Time Measurements and Out-of-Order Execution



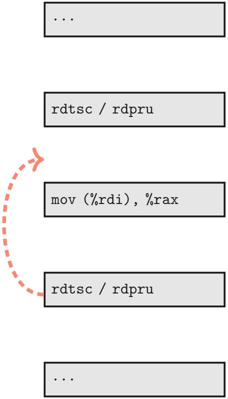
```
...
rdtsc / rdpru
mov (%rdi), %rax
rdtsc / rdpru
...
```

Time Measurements and Out-of-Order Execution



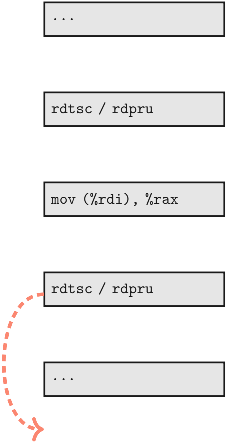
```
...
rdtsc / rdpru
mov (%rdi), %rax
rdtsc / rdpru
...
```

Time Measurements and Out-of-Order Execution



```
...
rdtsc / rdpru
mov (%rdi), %rax
rdtsc / rdpru
...
```

Time Measurements and Out-of-Order Execution

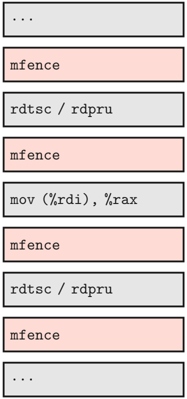


```
...
rdtsc / rdpru
mov (%rdi), %rax
rdtsc / rdpru
...
```

Time Measurements and Out-of-Order Execution

```
...
rdtsc / rdpru
mov (%rdi), %rax
rdtsc / rdpru
...
```

Time Measurements and Out-of-Order Execution



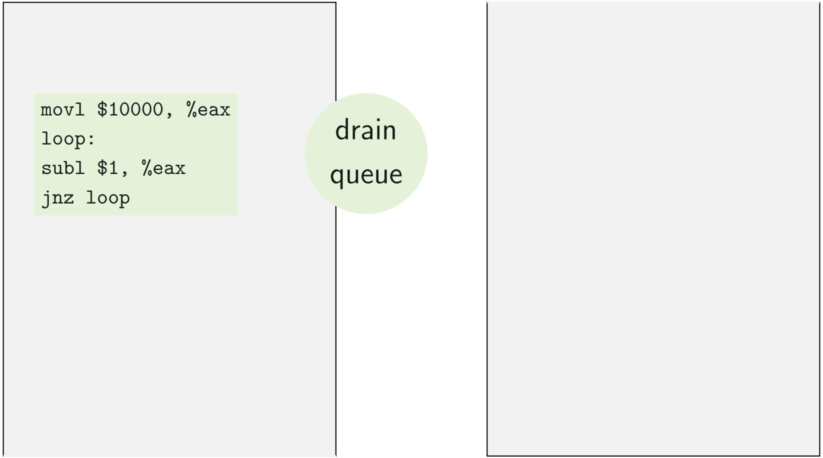
```
...
mfence
rdtsc / rdpru
mfence
mov (%rdi), %rax
mfence
rdtsc / rdpru
mfence
...
```

Adding serializing instructions around rdtsc

Exploiting out-of-order execution of rdtsc

Let’s code that!

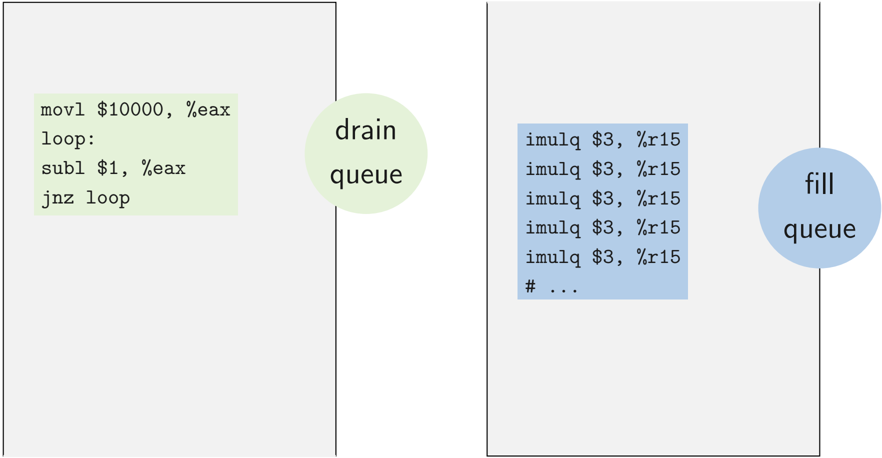
Let’s code that!



```
movl $10000, %eax
loop:
subl $1, %eax
jnz loop
```

drain queue

Let’s code that!



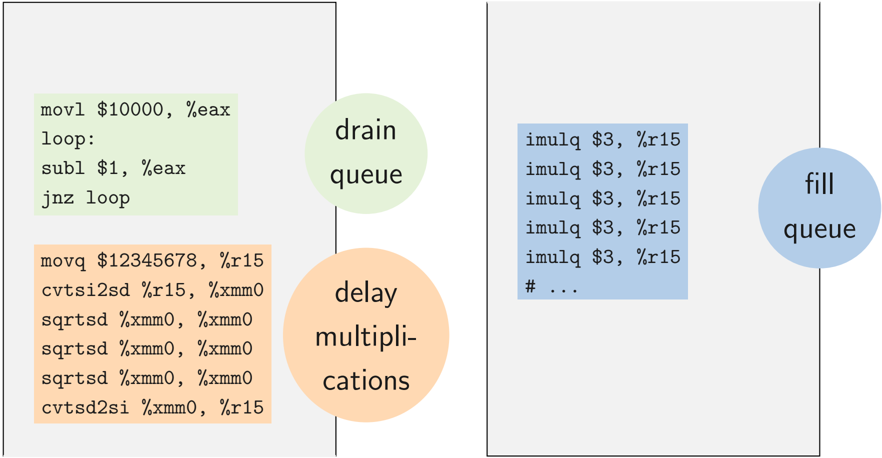
```
movl $10000, %eax
loop:
subl $1, %eax
jnz loop
```

drain queue

```
imulq $3, %r15
imulq $3, %r15
imulq $3, %r15
imulq $3, %r15
imulq $3, %r15
# ...
```

fill queue

Let’s code that!



```
movl $10000, %eax
loop:
subl $1, %eax
jnz loop
```

drain queue

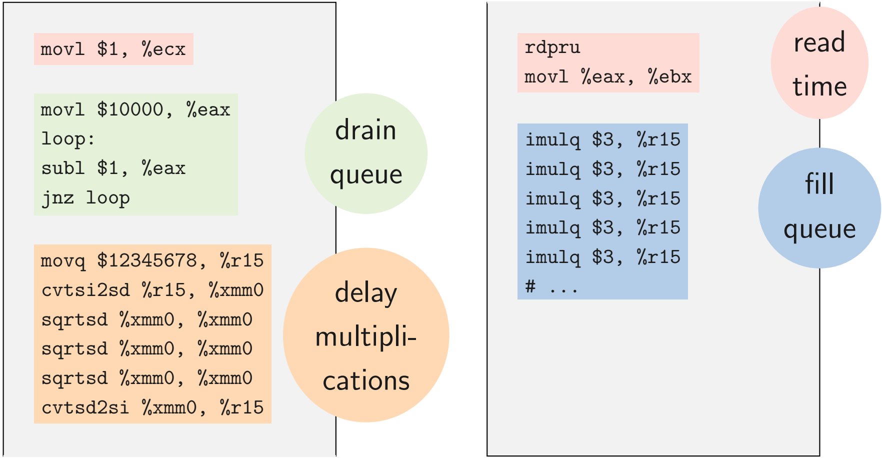
```
movq $12345678, %r15
cvtsi2sd %r15, %xmm0
sqrtsd %xmm0, %xmm0
sqrtsd %xmm0, %xmm0
sqrtsd %xmm0, %xmm0
cvtsd2si %xmm0, %r15
```

delay multiplications

```
imulq $3, %r15
imulq $3, %r15
imulq $3, %r15
imulq $3, %r15
imulq $3, %r15
# ...
```

fill queue

Let’s code that!



```
movl $1, %ecx
```

```
movl $10000, %eax
loop:
subl $1, %eax
jnz loop
```

drain queue

```
movq $12345678, %r15
cvtsi2sd %r15, %xmm0
sqrtsd %xmm0, %xmm0
sqrtsd %xmm0, %xmm0
sqrtsd %xmm0, %xmm0
cvtsd2si %xmm0, %r15
```

delay multipli-cations

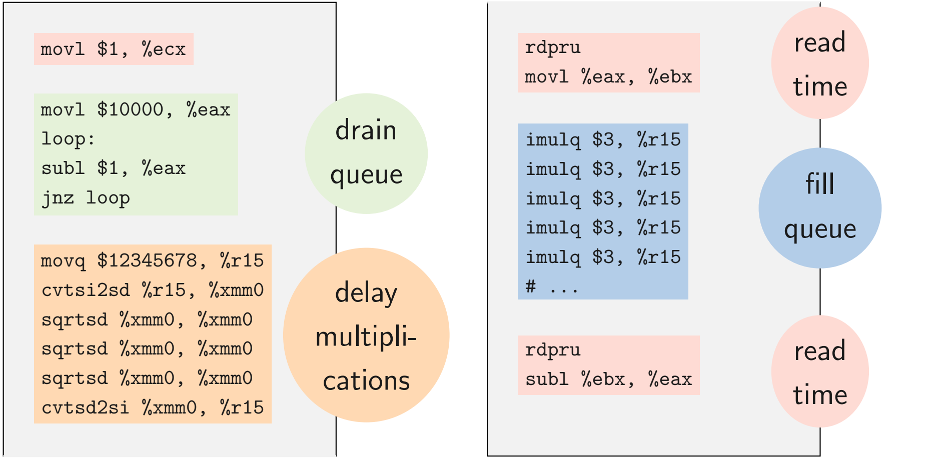
```
rdpru
movl %eax, %ebx
```

read time

```
imulq $3, %r15
imulq $3, %r15
imulq $3, %r15
imulq $3, %r15
imulq $3, %r15
# ...
```

fill queue

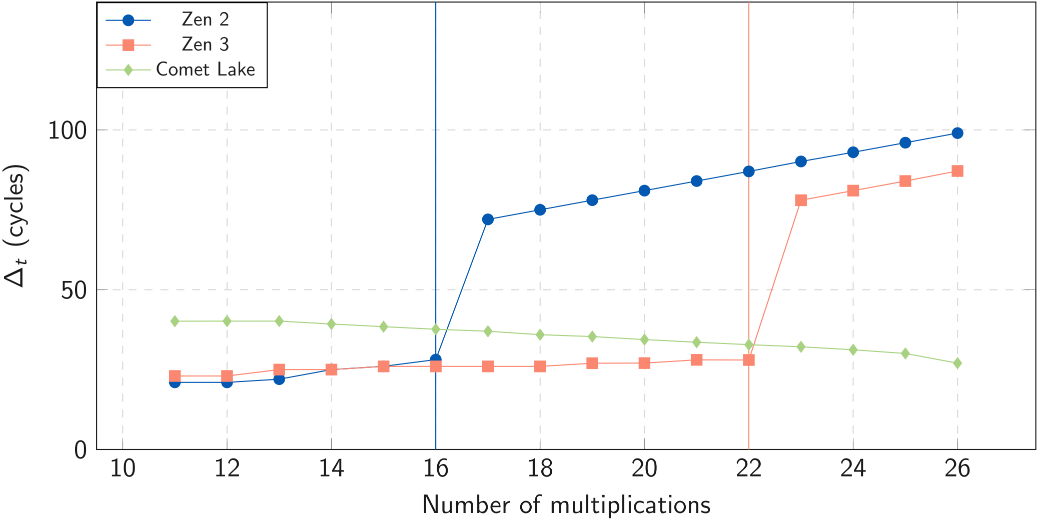
Let’s code that!



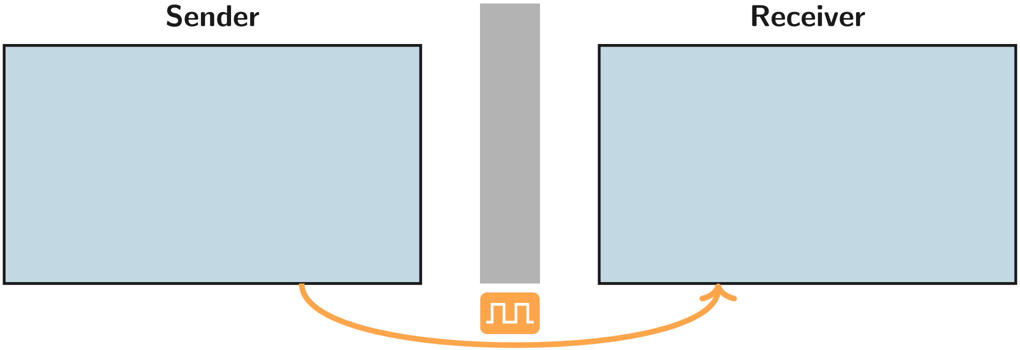
```
movl $1, %ecx

movl $10000, %eax
loop:
subl $1, %eax
jnz loop

movq $12345678, %r15
cvtsi2sd %r15, %xmm0
sqrtsd %xmm0, %xmm0
sqrtsd %xmm0, %xmm0
sqrtsd %xmm0, %xmm0
cvtsd2si %xmm0, %r15
```

```
rdpru
movl %eax, %ebx

imulq $3, %r15
imulq $3, %r15
imulq $3, %r15
imulq $3, %r15
imulq $3, %r15
# ...

rdpru
subl %ebx, %eax
```

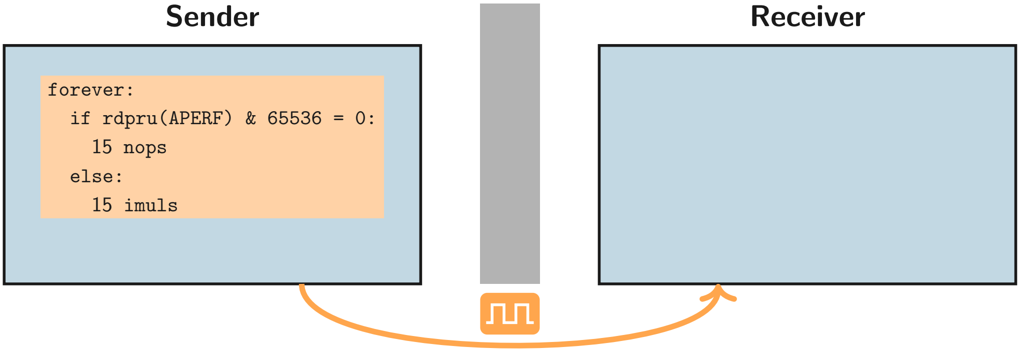
Timing Differences on Zen 2, Zen 3 and Comet Lake



Observing Multiplications of the Sibling Thread

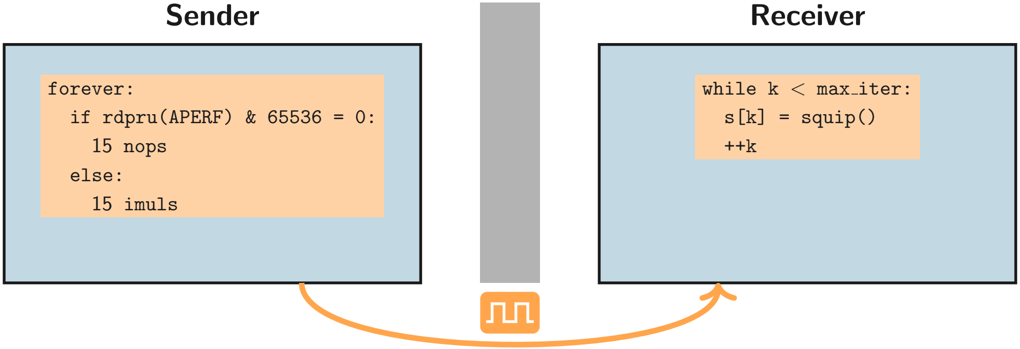


Observing Multiplications of the Sibling Thread



```
forever:
  if rdpru(APERF) & 65536 = 0:
    15 nops
  else:
    15 imuls
```

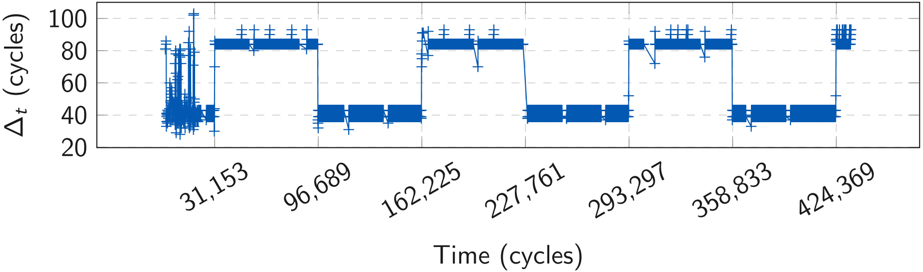
Observing Multiplications of the Sibling Thread



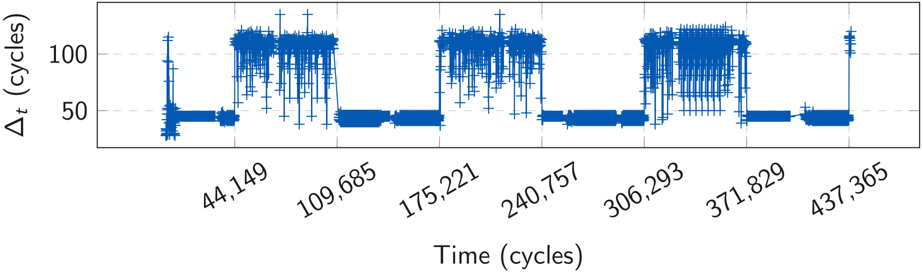
```
forever:
  if rdpru(APERF) & 65536 = 0:
    15 nops
  else:
    15 imuls
```

```
while k < max_iter:
  s[k] = squip()
  ++k
```

Observing Multiplications of the Sibling Thread (Zen 2)



Observing Multiplications of the Sibling Thread (Zen 3)



Covert Channel Evaluation

Scenario	CPU	Raw Tx Rate	Error Rate

10 000 random messages, each with 32 kbit payload

Covert Channel Evaluation

Scenario	CPU	Raw Tx Rate	Error Rate
Cross-Process	Ryzen 7 3700X (Zen 2)	$2.195\,\mathrm{Mbit\,s^{-1}}$	$0.71\,\%$
	Ryzen 7 5800X (Zen 3)	$2.700\,\mathrm{Mbit\,s^{-1}}$	$0.62\,\%$

10 000 random messages, each with 32 kbit payload

Covert Channel Evaluation

Scenario	CPU	Raw Tx Rate	Error Rate
Cross-Process	Ryzen 7 3700X (Zen 2)	2.195 Mbit s^{-1}	0.71 %
	Ryzen 7 5800X (Zen 3)	2.700 Mbit s^{-1}	0.62 %
Cross-VM	Ryzen 7 3700X (Zen 2)	0.873 Mbit s^{-1}	3.18 %
	Ryzen 7 5800X (Zen 3)	0.892 Mbit s^{-1}	0.75 %
	EPYC 7443 (Zen 3)	0.874 Mbit s^{-1}	0.96 %

10 000 random messages, each with 32 kbit payload

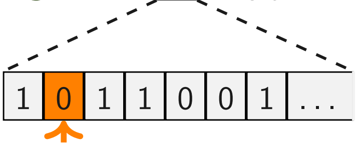
Covert Channel Evaluation

Scenario	CPU	Raw Tx Rate	Error Rate
Cross-Process	Ryzen 7 3700X (Zen 2)	2.195 Mbit s^{-1}	0.71 %
	Ryzen 7 5800X (Zen 3)	2.700 Mbit s^{-1}	0.62 %
Cross-VM	Ryzen 7 3700X (Zen 2)	0.873 Mbit s^{-1}	3.18 %
	Ryzen 7 5800X (Zen 3)	0.892 Mbit s^{-1}	0.75 %
	EPYC 7443 (Zen 3)	0.874 Mbit s^{-1}	0.96 %
Cross-VM (SEV)	EPYC 7443 (Zen 3)	0.873 Mbit s^{-1}	1.47 %

10 000 random messages, each with 32 kbit payload

Attacking RSA (Square+Multiply)

$$S = M^{\boxed{d}} \bmod n$$

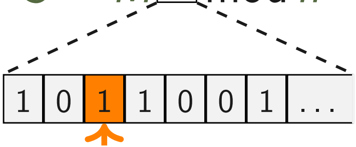


1	0	1	1	0	0	1	...

$$\boxed{\text{Result}} = \underbrace{\boxed{\text{Result}} \times \boxed{\text{Result}}}_{\text{square}}$$

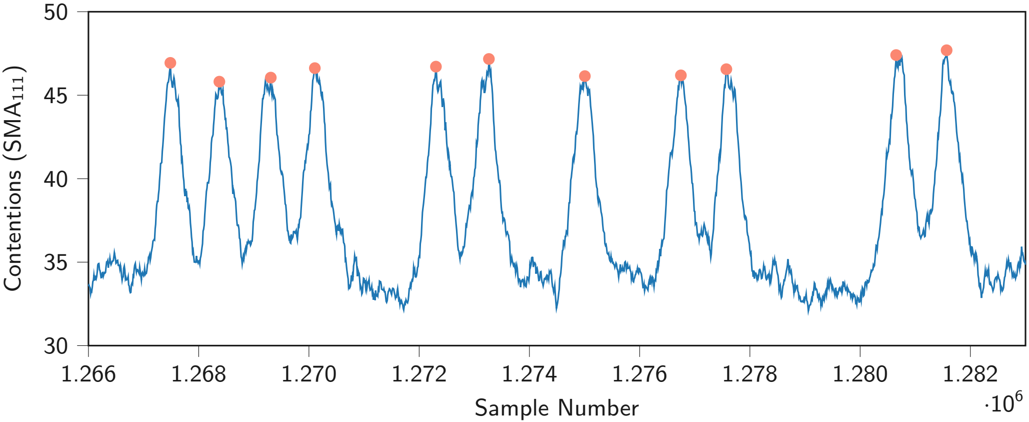
Attacking RSA (Square+Multiply)

$$S = M^{\boxed{d}} \bmod n$$

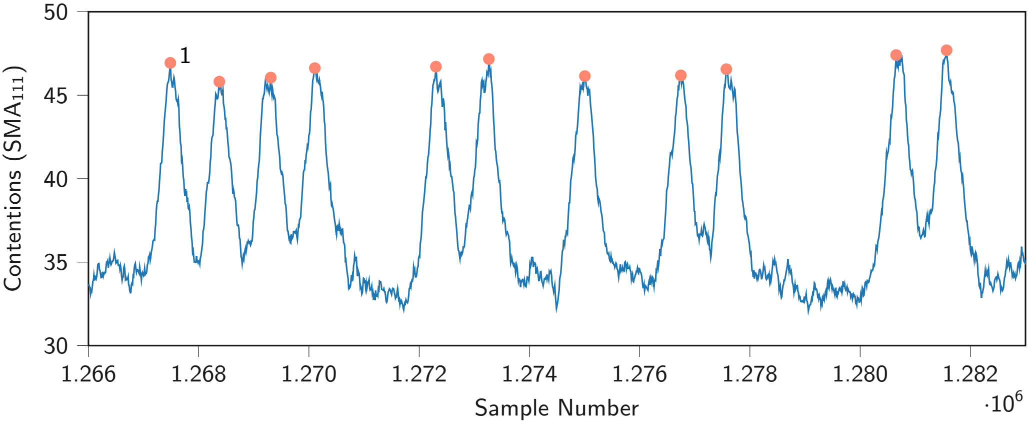


$$\boxed{\text{Result}} = \underbrace{\boxed{\text{Result}} \times \boxed{\text{Result}}}_{\text{square}} \underbrace{\times \boxed{\text{M}}}_{\text{multiply}}$$

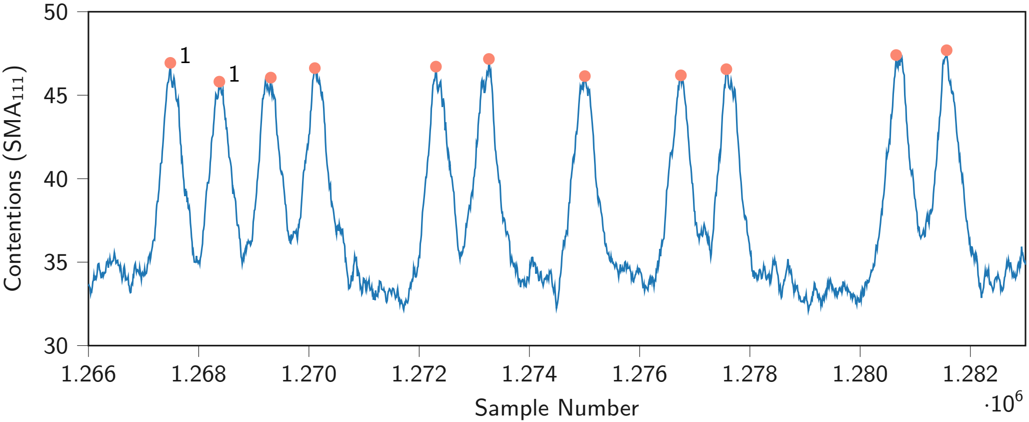
Attacking RSA: Key Recovery



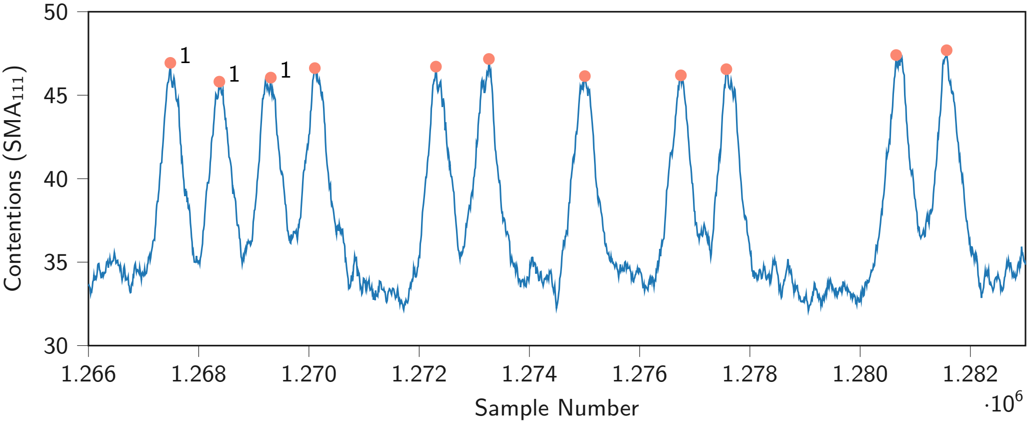
Attacking RSA: Key Recovery



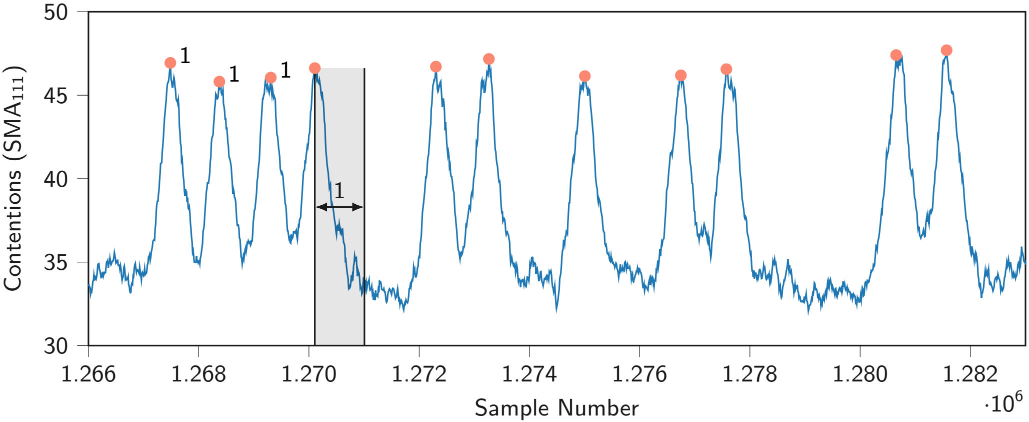
Attacking RSA: Key Recovery



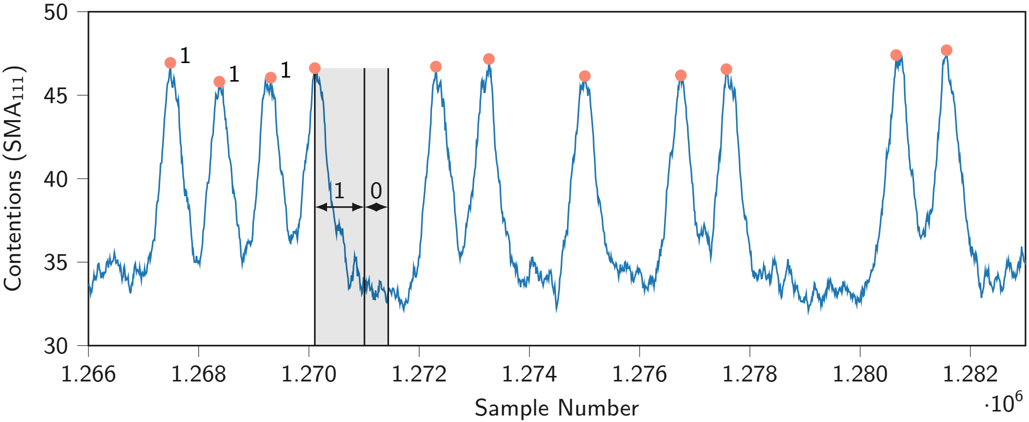
Attacking RSA: Key Recovery



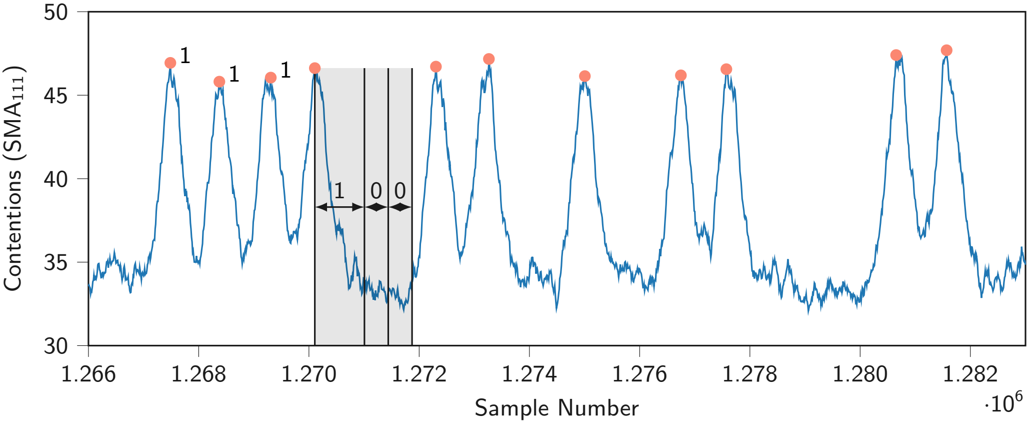
Attacking RSA: Key Recovery



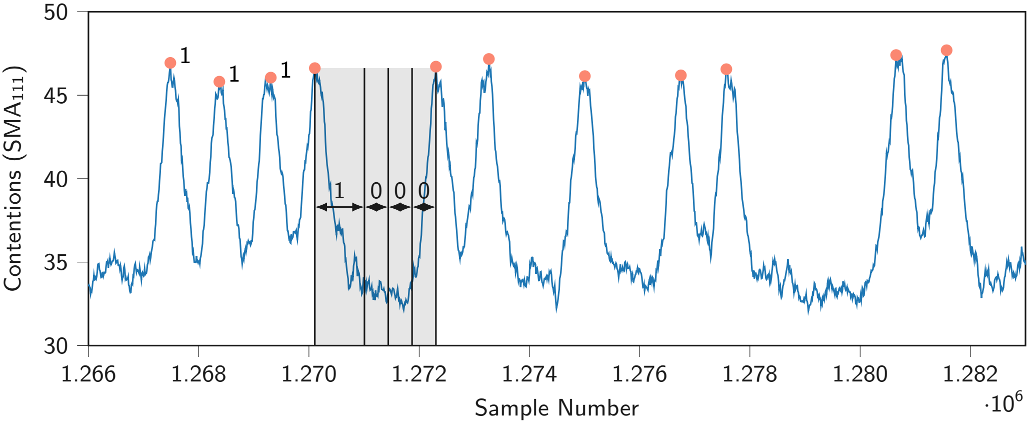
Attacking RSA: Key Recovery



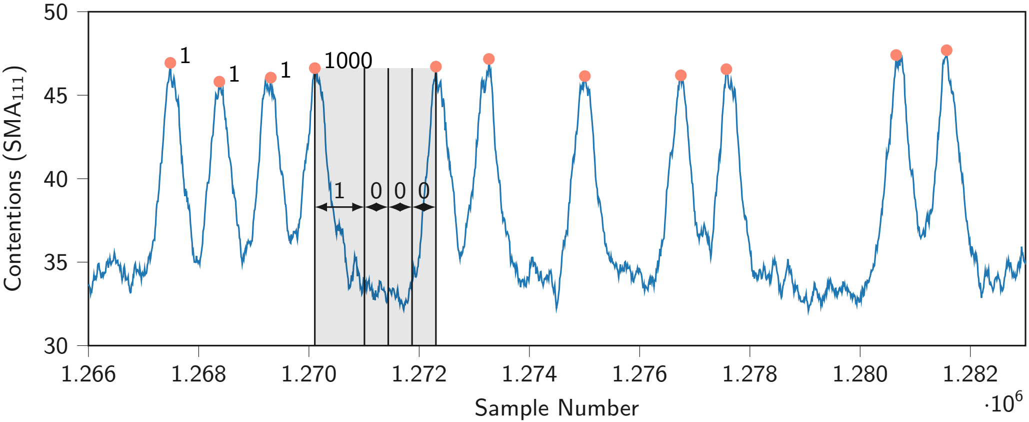
Attacking RSA: Key Recovery



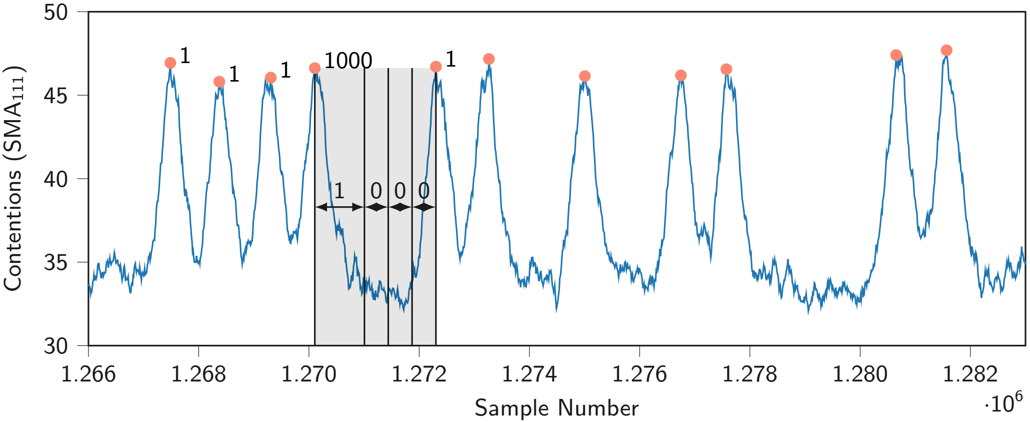
Attacking RSA: Key Recovery



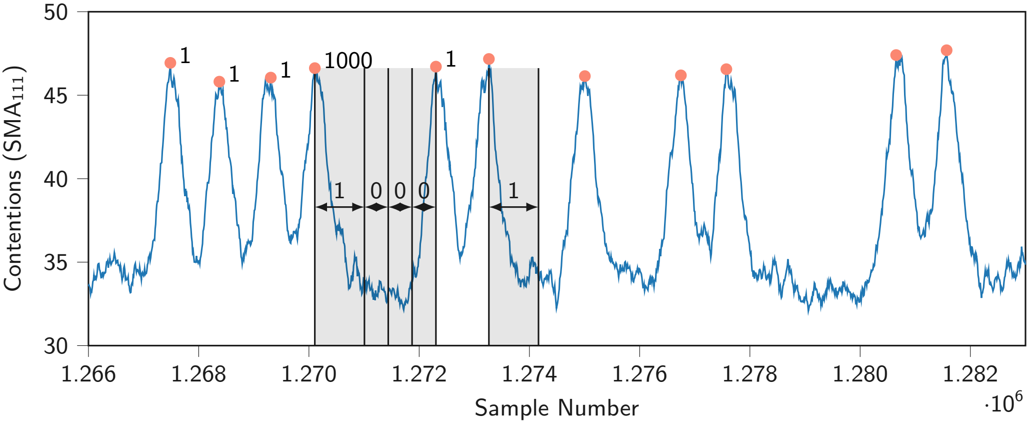
Attacking RSA: Key Recovery



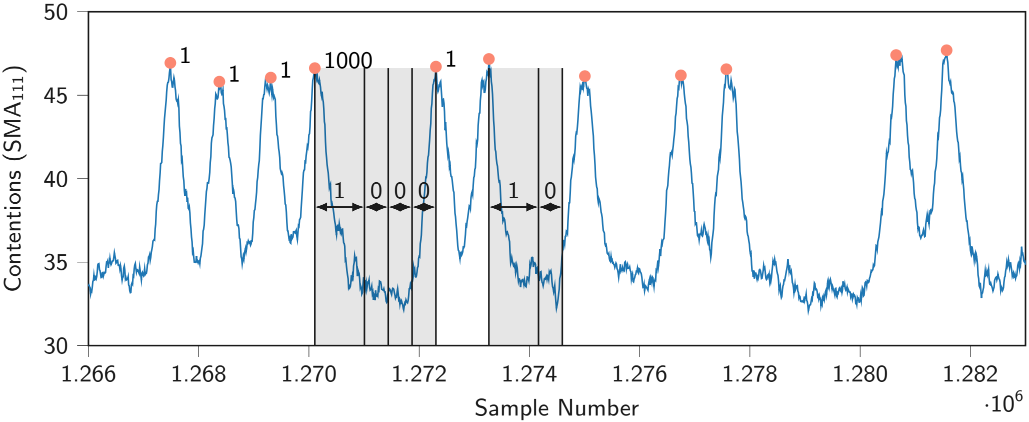
Attacking RSA: Key Recovery



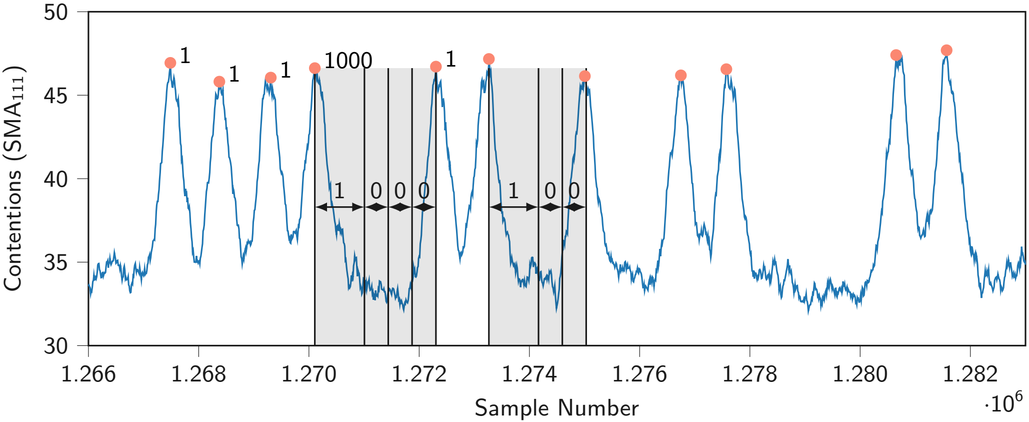
Attacking RSA: Key Recovery



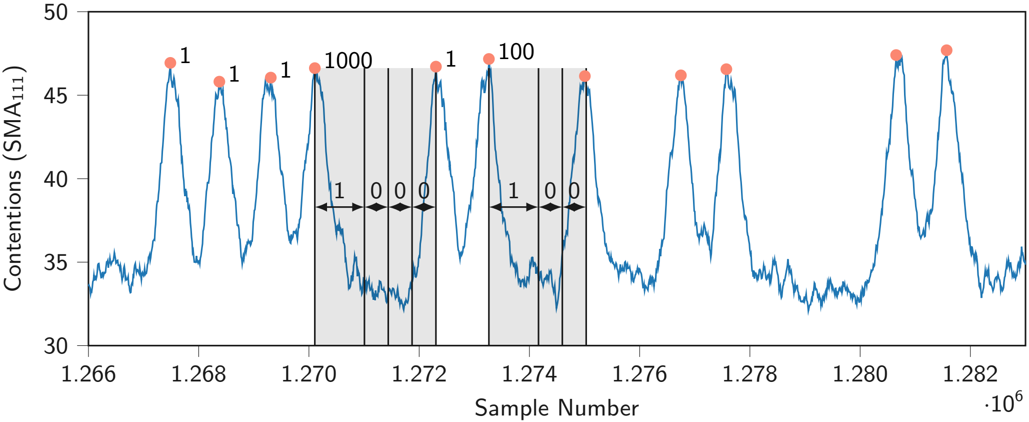
Attacking RSA: Key Recovery



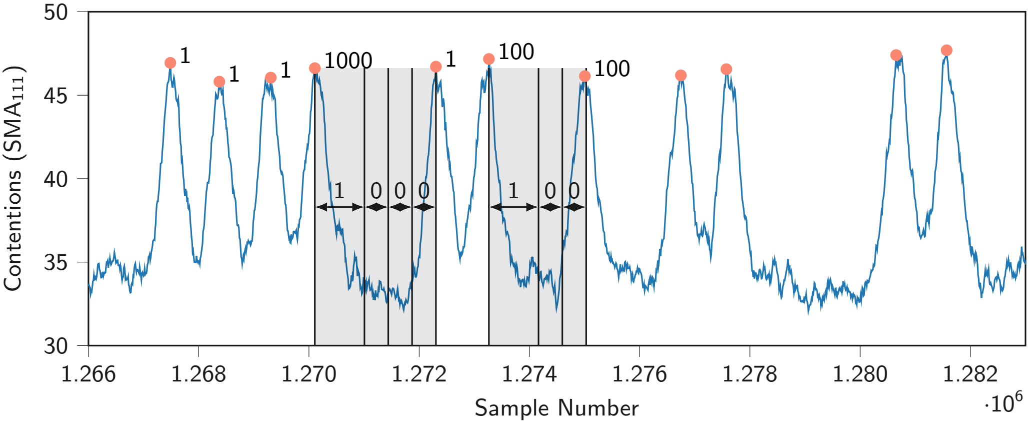
Attacking RSA: Key Recovery



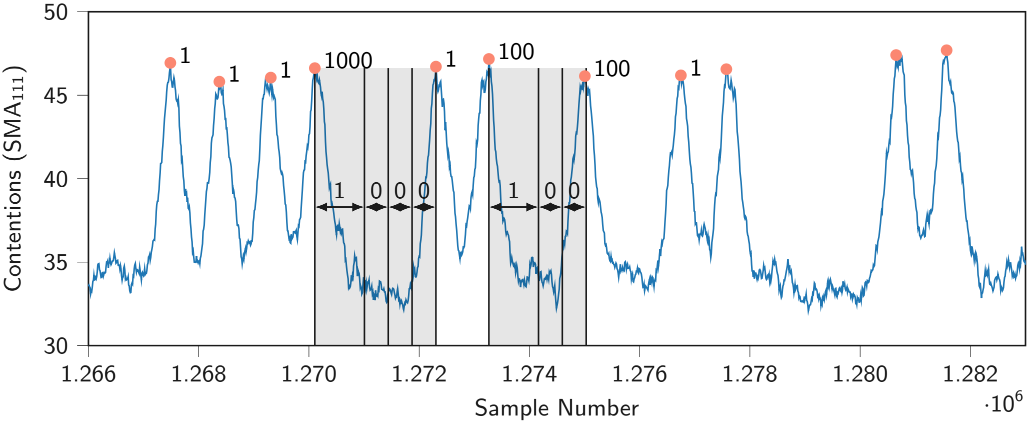
Attacking RSA: Key Recovery



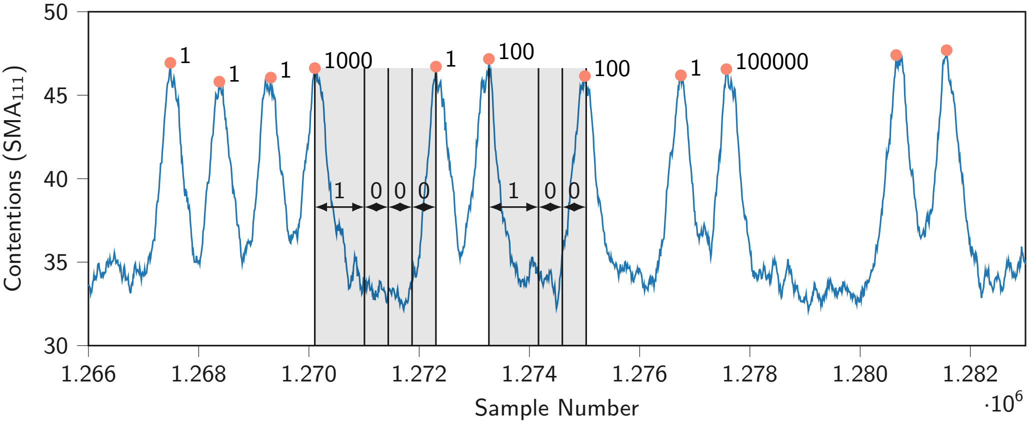
Attacking RSA: Key Recovery



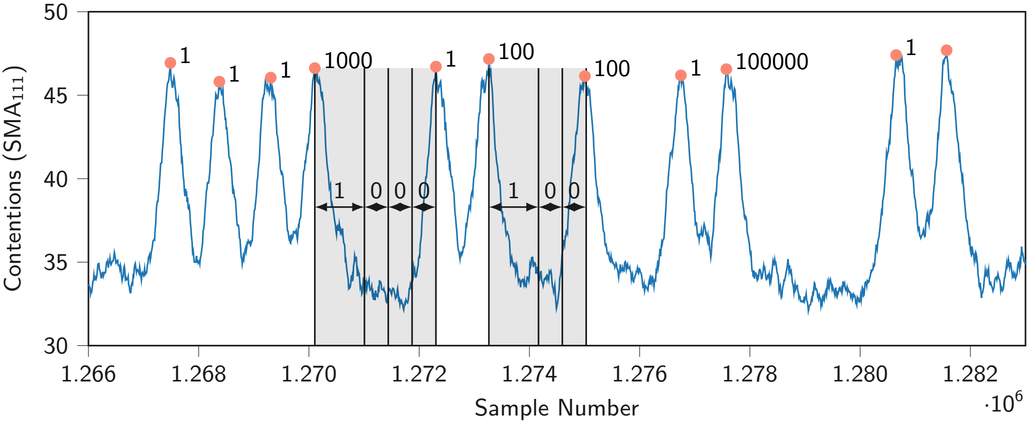
Attacking RSA: Key Recovery



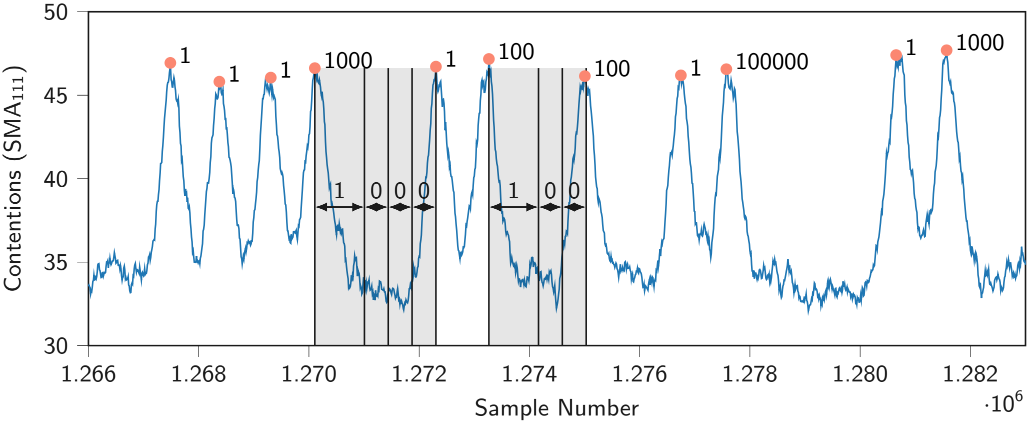
Attacking RSA: Key Recovery



Attacking RSA: Key Recovery



Attacking RSA: Key Recovery

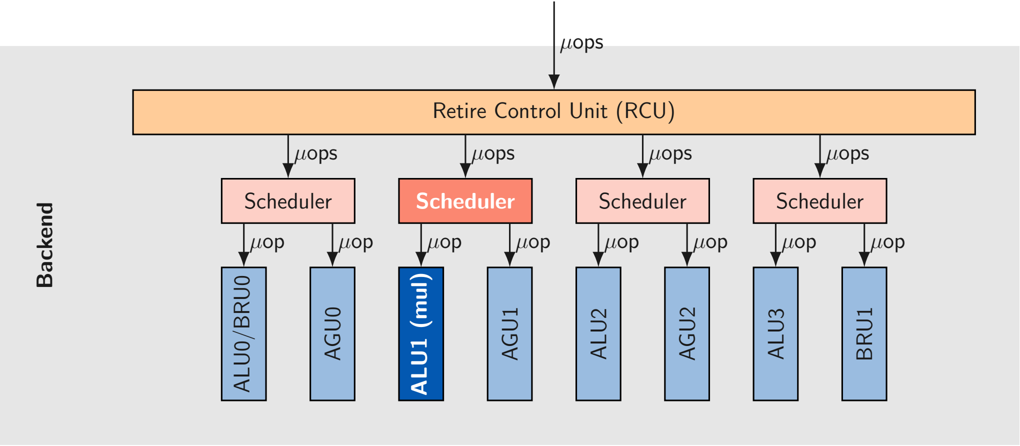


Attacking RSA: Evaluation

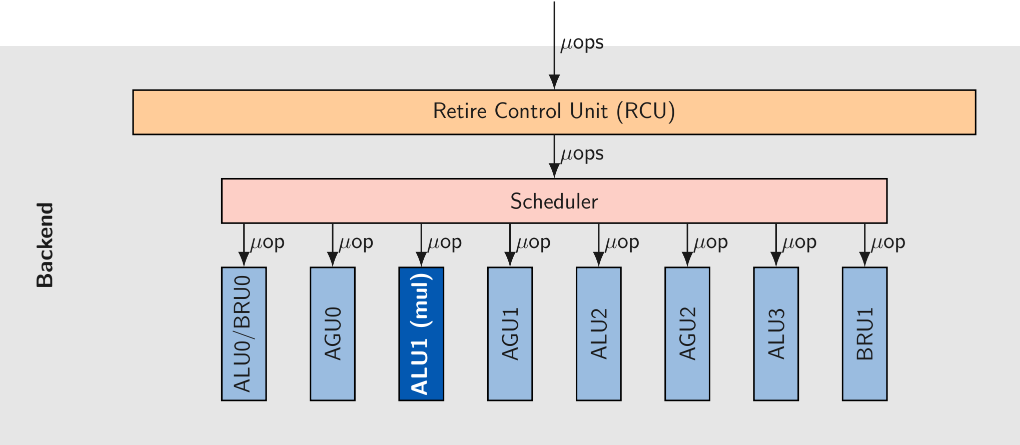
Scenario	Edit Distance	Error Rate	Recording Time
Cross-Process	4.9 bit	0.12 %	41 min
Cross-VM	17.8 bit	0.5 %	38 min

- 10 keys, generated with `openssl genrsa`
- CPU: AMD Ryzen 7 5800X (Zen 3)
- 50500 traces per key

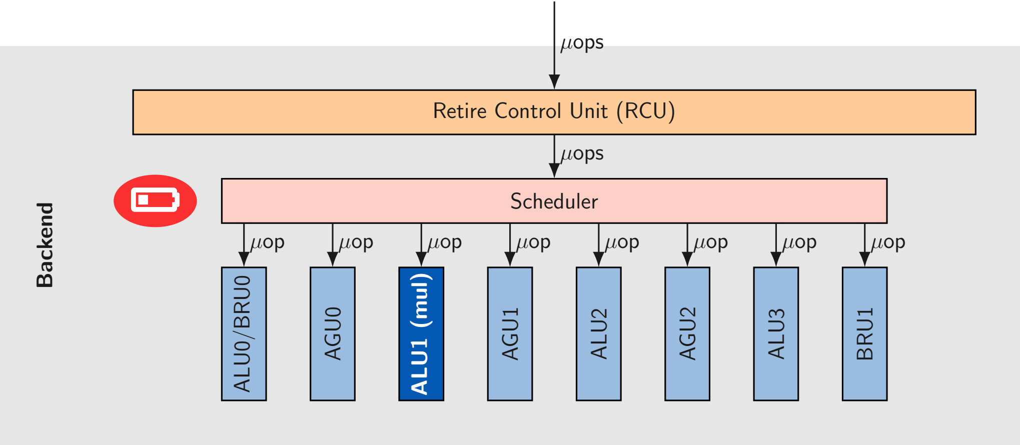
Countermeasures: Hardware



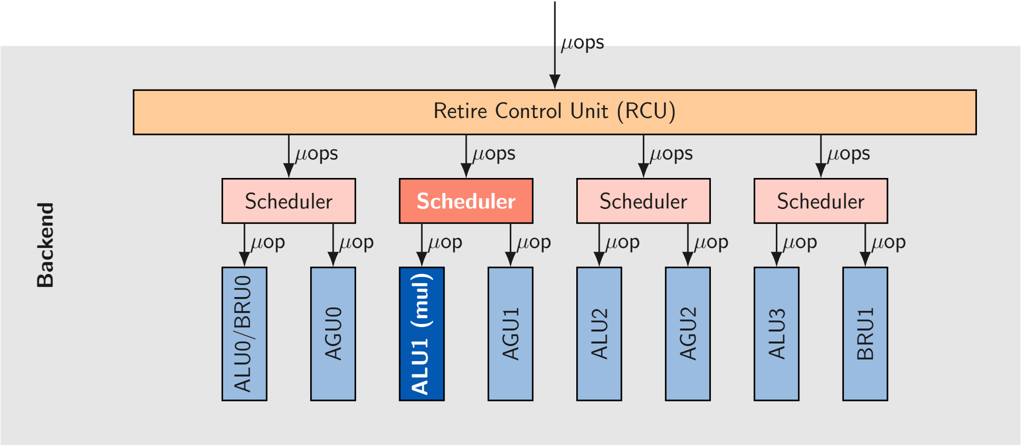
Countermeasures: Hardware



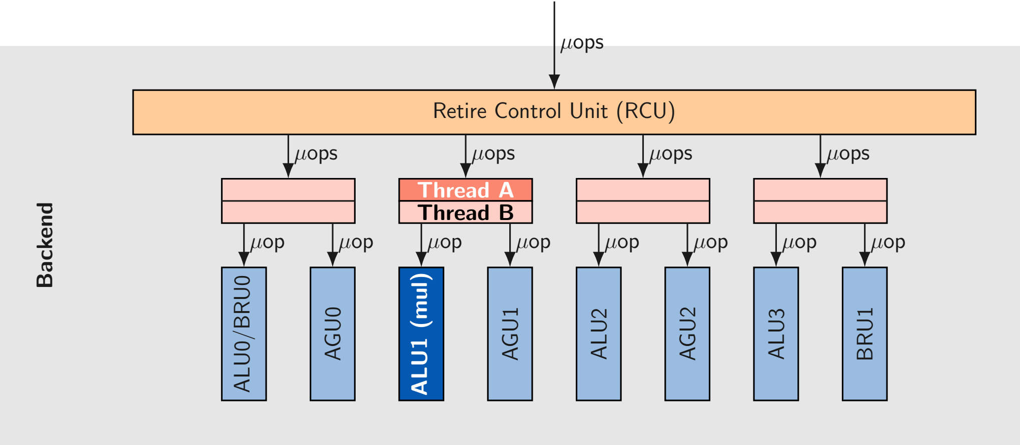
Countermeasures: Hardware



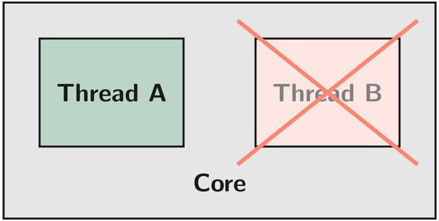
Countermeasures: Hardware



Countermeasures: Hardware



Countermeasures: Software



SQUIP

Exploiting the Scheduler Queue Contention Side Channel

Stefan Gast **Jonas Juffinger** **Martin Schwarzl** **Gururaj Saileshwar**
Andreas Kogler **Simone Franza** **Markus Köstl** **Daniel Gruss**

2023-05-23

✉ stefan.gast@iaik.tugraz.at
notbobbytables@infosec.exchange
www.stefangast.eu