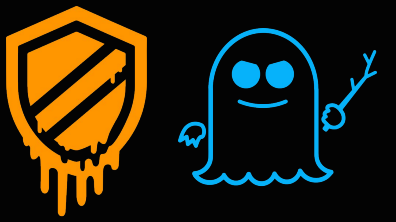


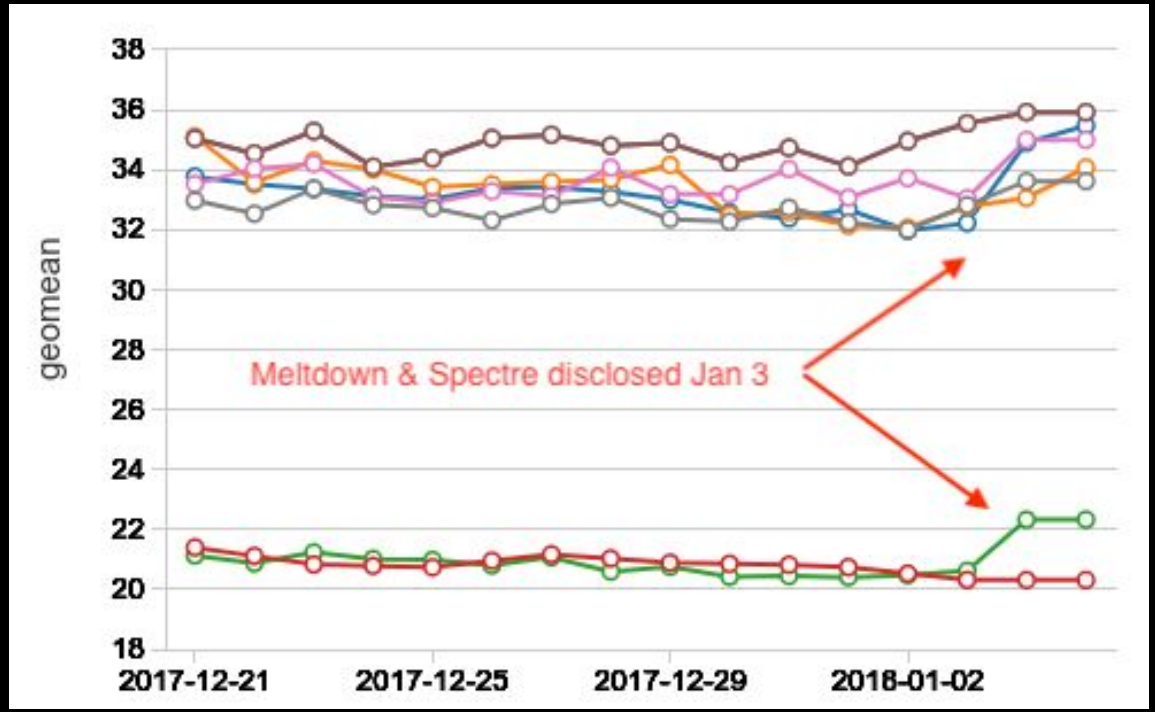
Meltdown, Spectre and Apache Spark™ Performance

Chris Stevens

June 5, 2018

databricks

Databricks Performance on AWS



3-5% performance degradation

Overview

Goal: Understand the 3-5% degradation

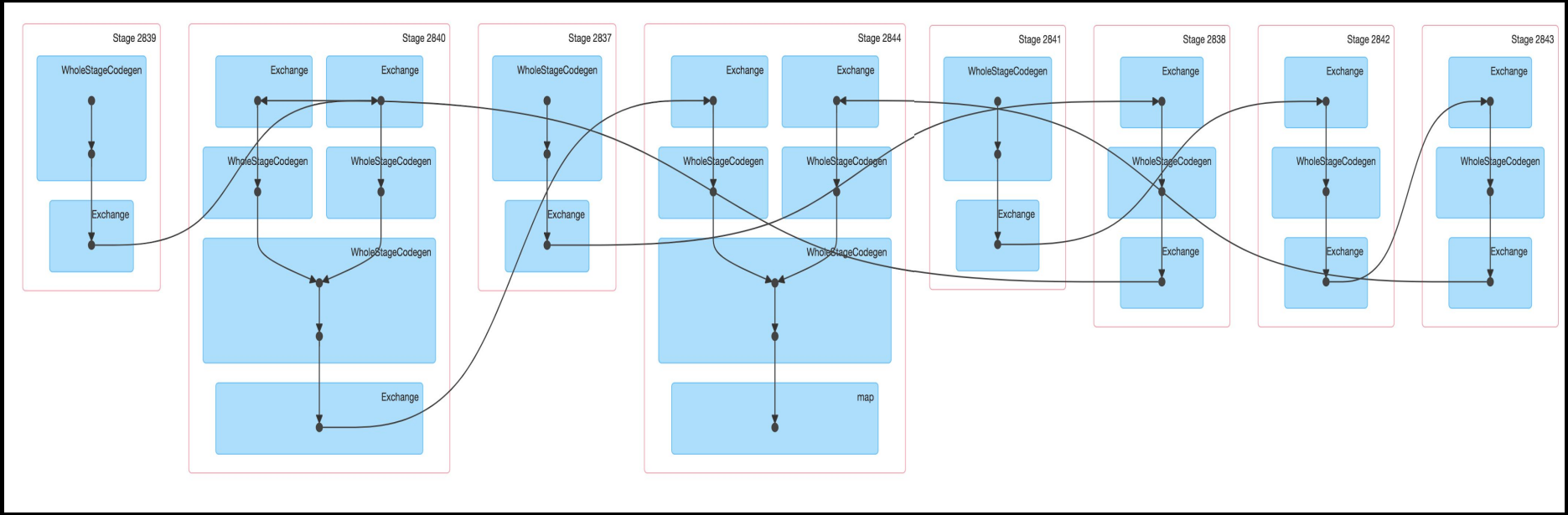
Steps:

- TPC-DS Benchmarks
- System Analysis
- Breakdown the exploits and patches
  - Meltdown
  - Spectre V1 - Bounds Check Bypass
  - Specter V2 - Branch Target Injection

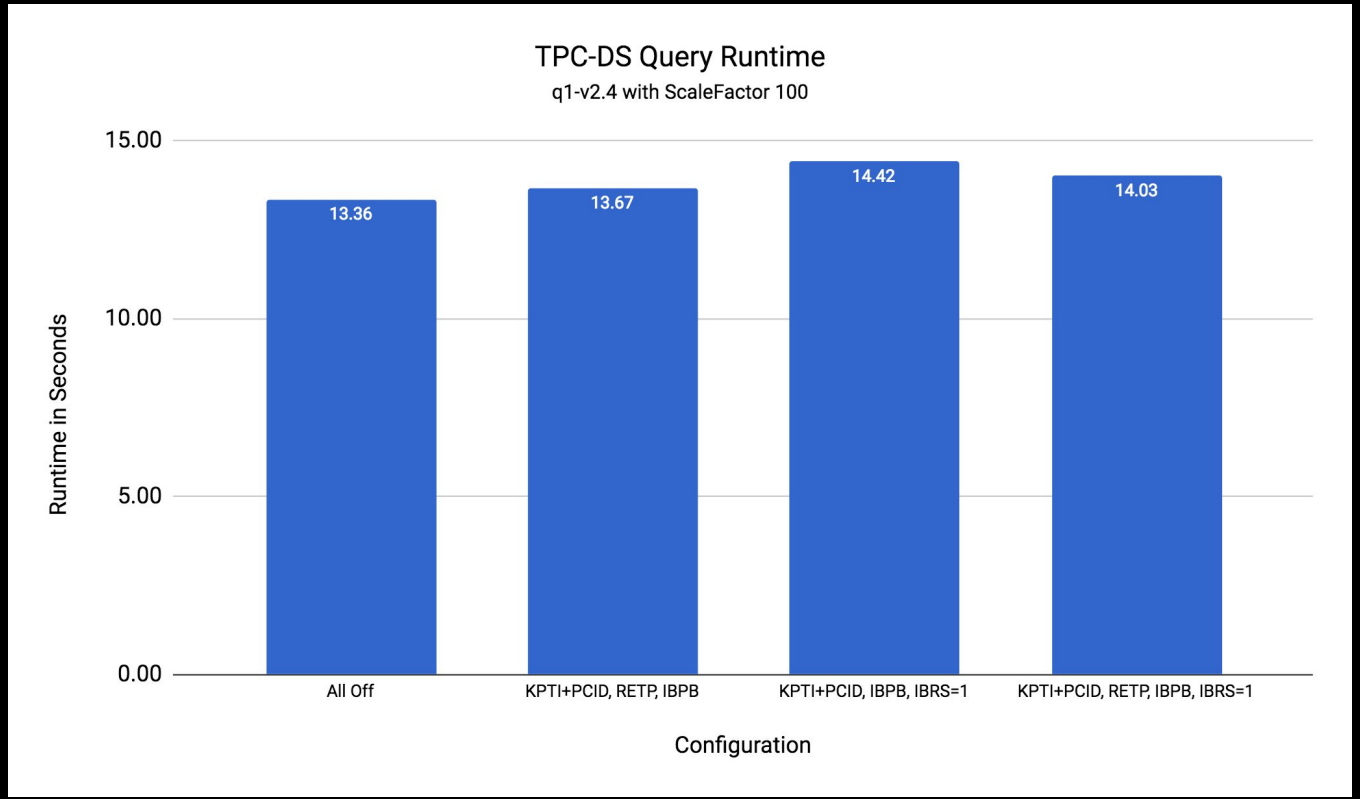
TPC-DS: q1-v2.4

```
WITH customer_total_return AS
  (SELECT sr_customer_sk AS ctr_customer_sk,
          sr_store_sk AS ctr_store_sk,
          sum(sr_return_amt) AS ctr_total_return
   FROM store_returns, date_dim
   WHERE sr_returned_date_sk = d_date_sk
   AND d_year = 2000
   GROUP BY sr_customer_sk, sr_store_sk)
SELECT c_customer_id
  FROM customer_total_return ctr1, store, customer
  WHERE ctr1.ctr_total_return >
    (SELECT avg(ctr_total_return)*1.2
     FROM customer_total_return ctr2
     WHERE ctr1.ctr_store_sk = ctr2.ctr_store_sk)
  AND s_store_sk = ctr1.ctr_store_sk
  AND s_state = 'TN'
  AND ctr1.ctr_customer_sk = c_customer_sk
  ORDER BY c_customer_id LIMIT 100
```

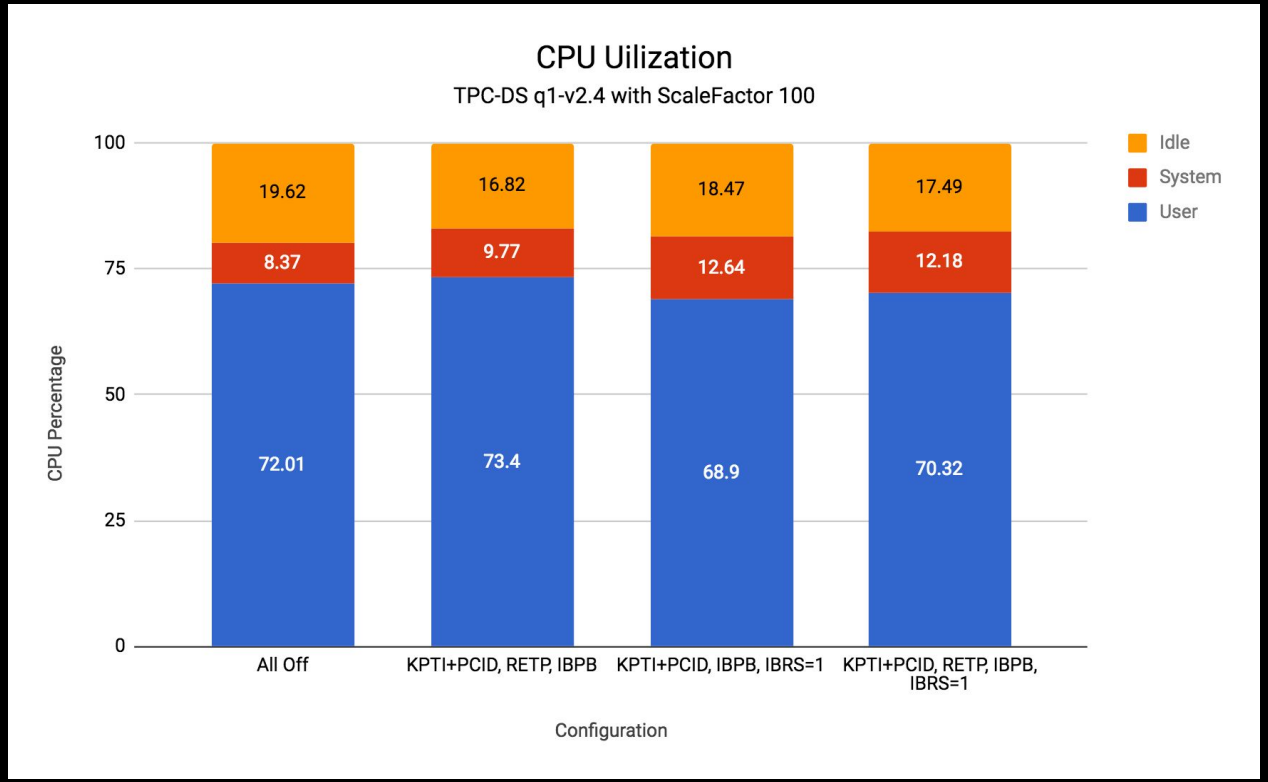
TPC-DS: q1-v2.4


Stage 2839
WholeStageCodegen
Exchange
Stage 2840
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
WholeStageCodegen
Exchange
Stage 2837
WholeStageCodegen
Exchange
Stage 2844
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
WholeStageCodegen
map
Stage 2841
WholeStageCodegen
Exchange
Stage 2838
Exchange
WholeStageCodegen
Exchange
Stage 2842
Exchange
WholeStageCodegen
Exchange
Stage 2843
Exchange
WholeStageCodegen
Exchange

TPC-DS: q1-v2.4

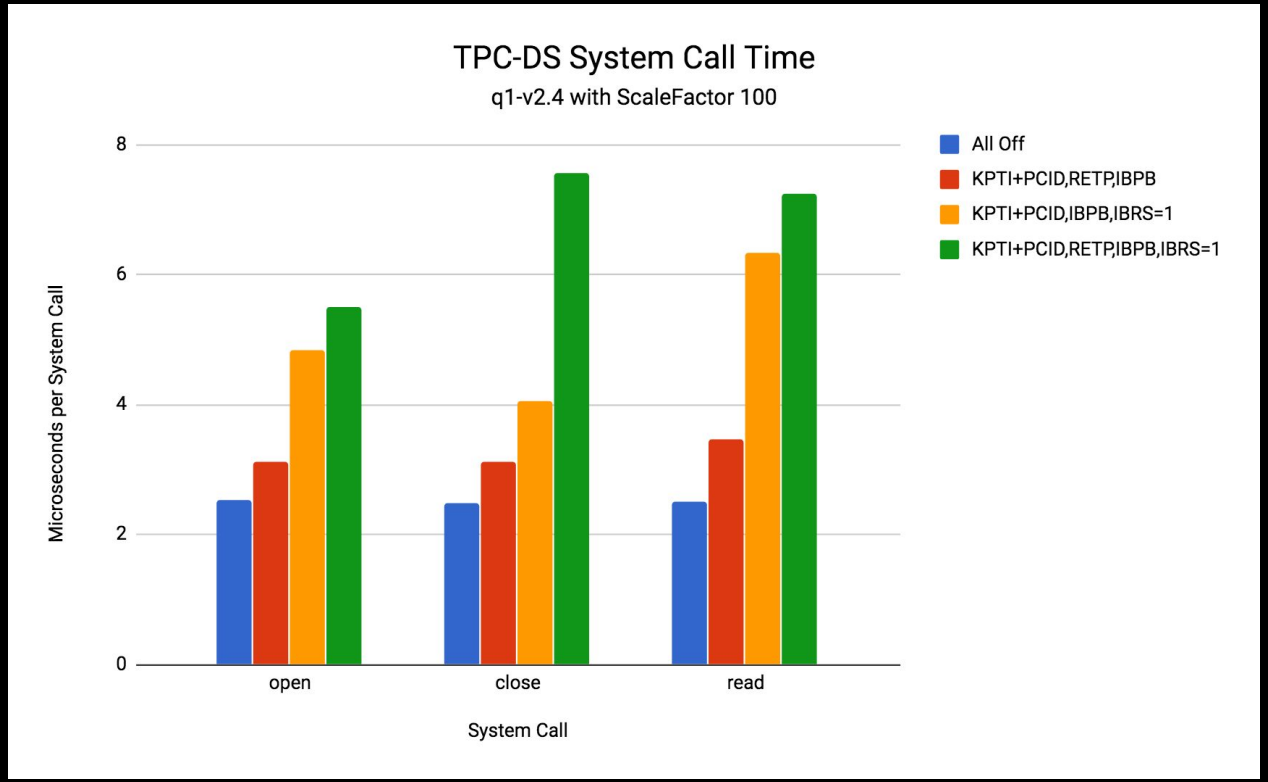

TPC-DS Query Runtime
q1-v2.4 with ScaleFactor 100
15.00
10.00
5.00
0.00
Runtime in Seconds
13.36
13.67
14.42
14.03
All Off
KPTI+PCID, RETP, IBPB
KPTI+PCID, IBPB, IBRS=1
KPTI+PCID, RETP, IBPB, IBRS=1
Configuration

q1-v2.4 CPU Utilization



mpstat -P ALL

TPC-DS: q1-v2.4



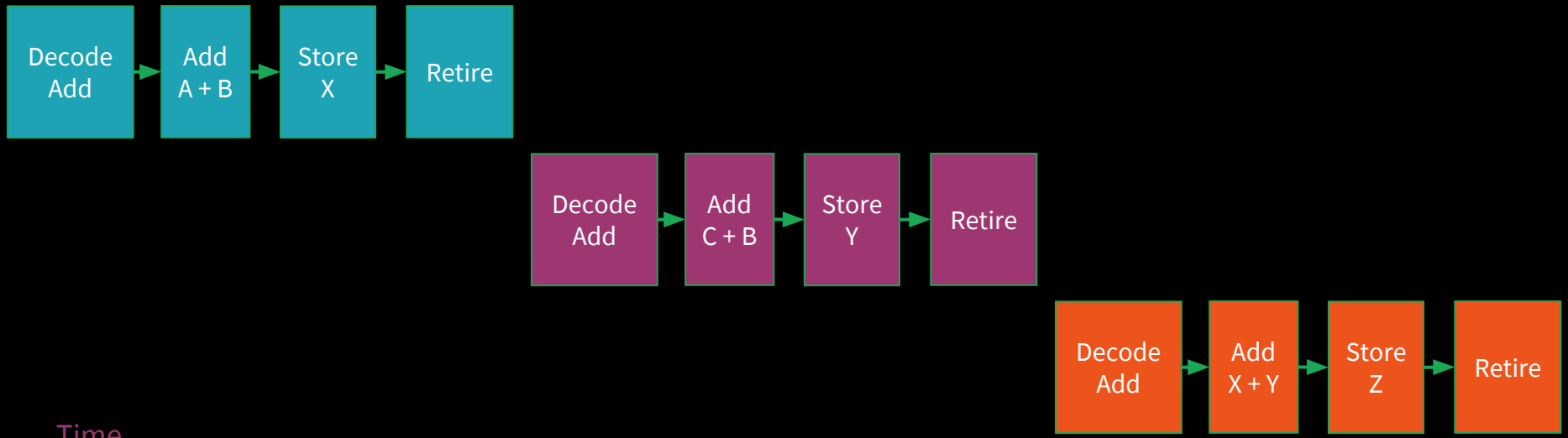

```
strace -f -c -p PID
```

Exploits Background

Out-of-Order Execution + Side Channel Attacks

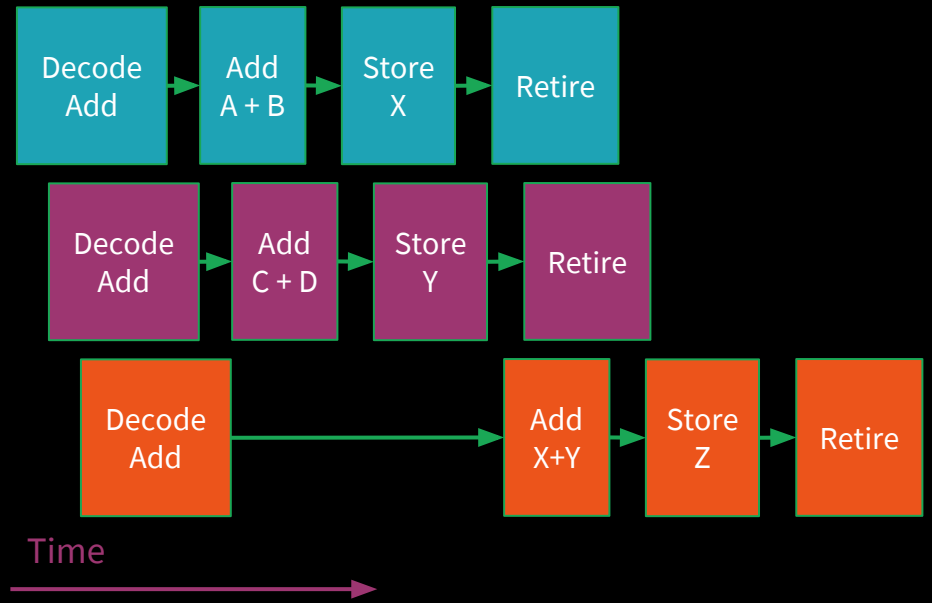
In-order Execution

```
x = a + b
y = c + d
z = x + y
```

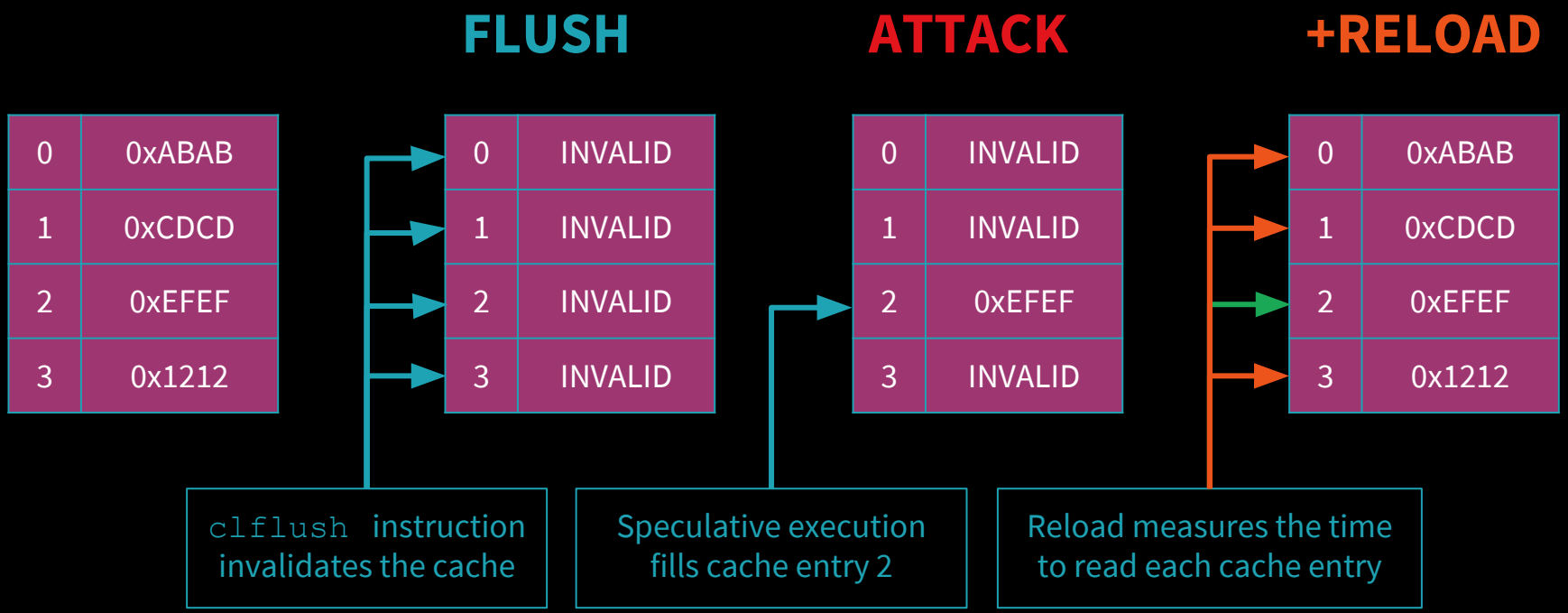


Out-of-order Execution

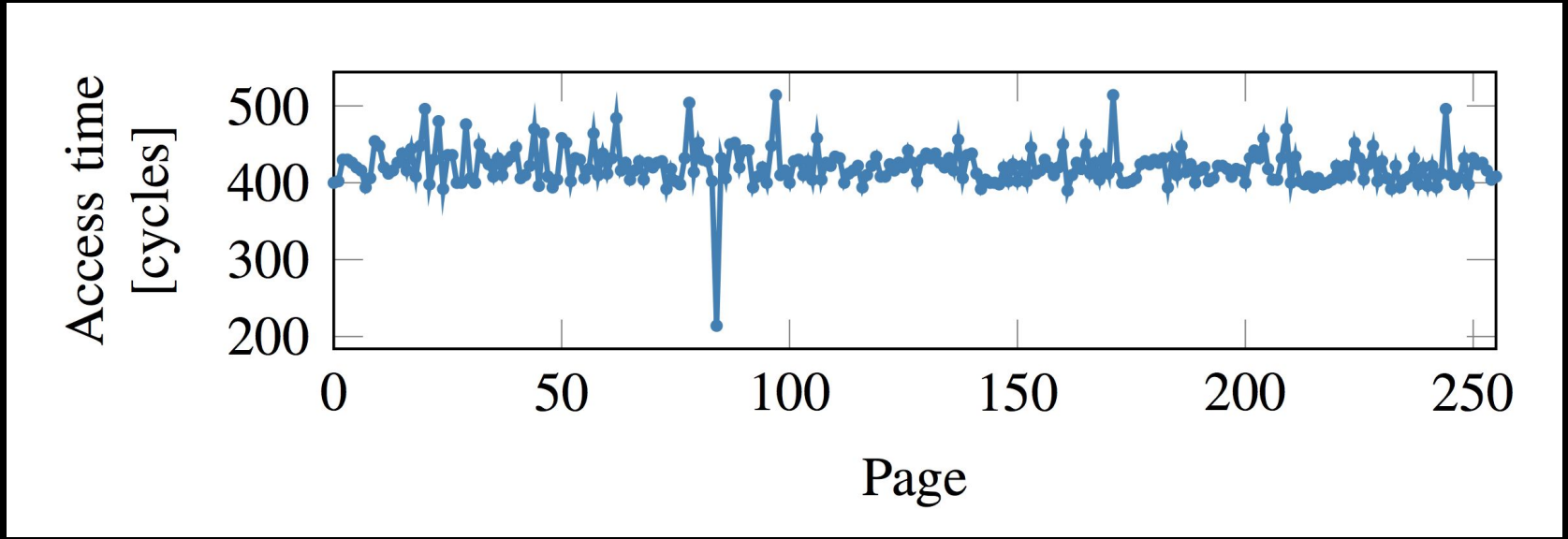
```
x = a + b
y = c + d
z = x + y
```



Side-Channel Attacks



Side-Channel Attacks



https://meltdownattack.com/meltdown.pdf

Meltdown

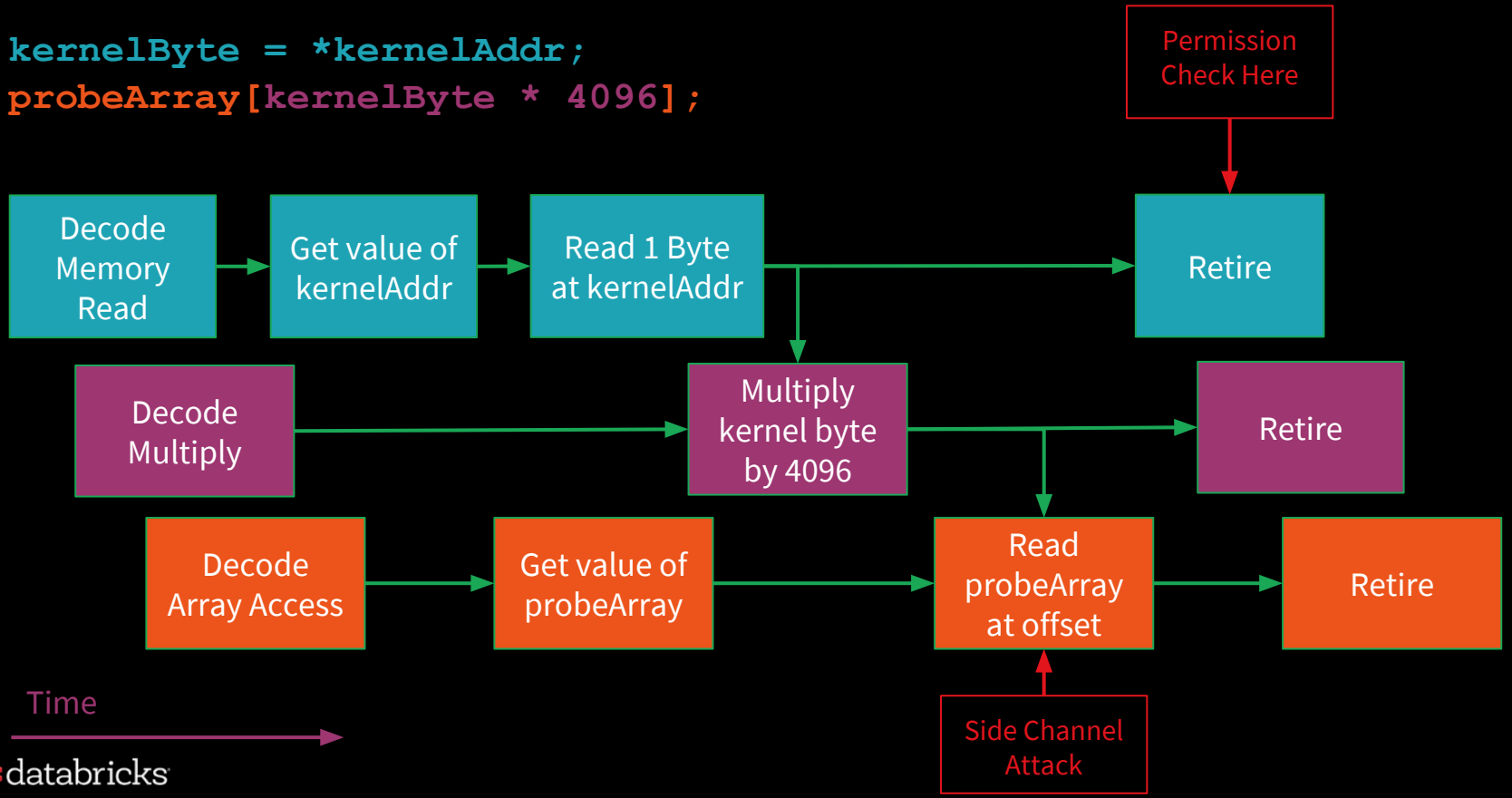
databricks

Meltdown

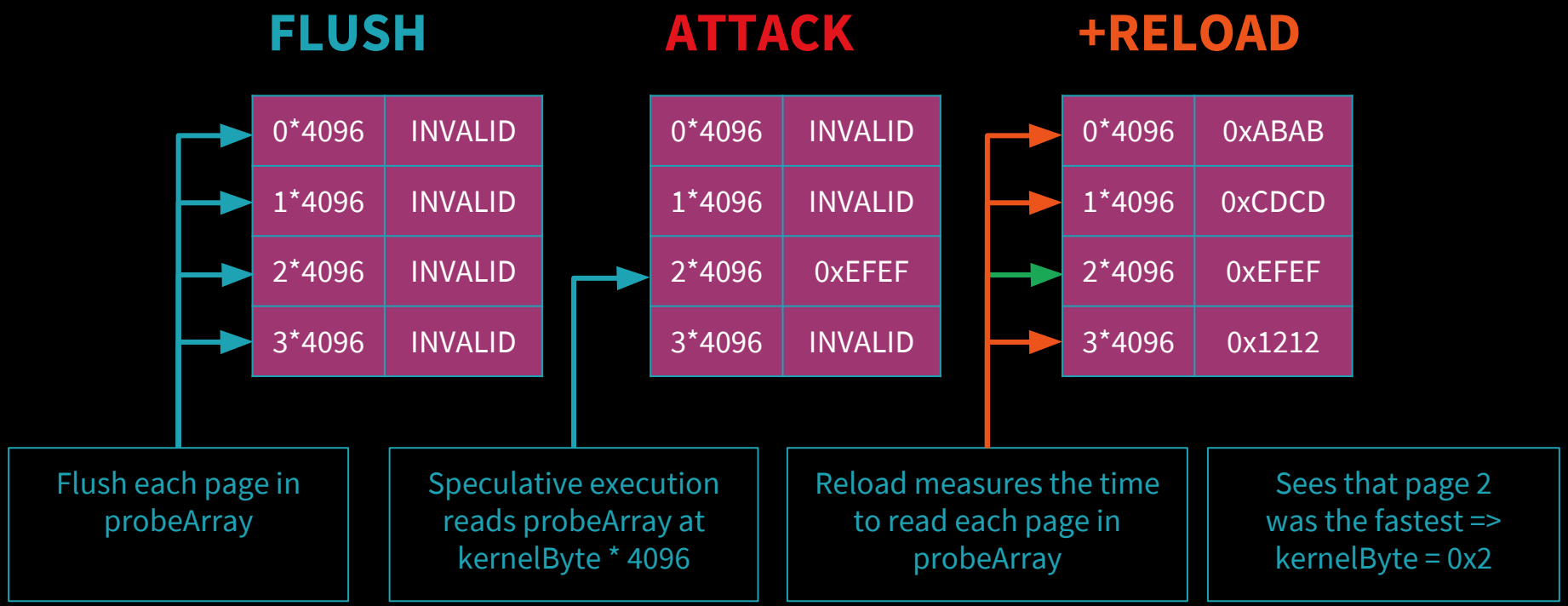
```
kernelByte = *kernelAddr;
probeArray[kernelByte * 4096];
```

Meltdown

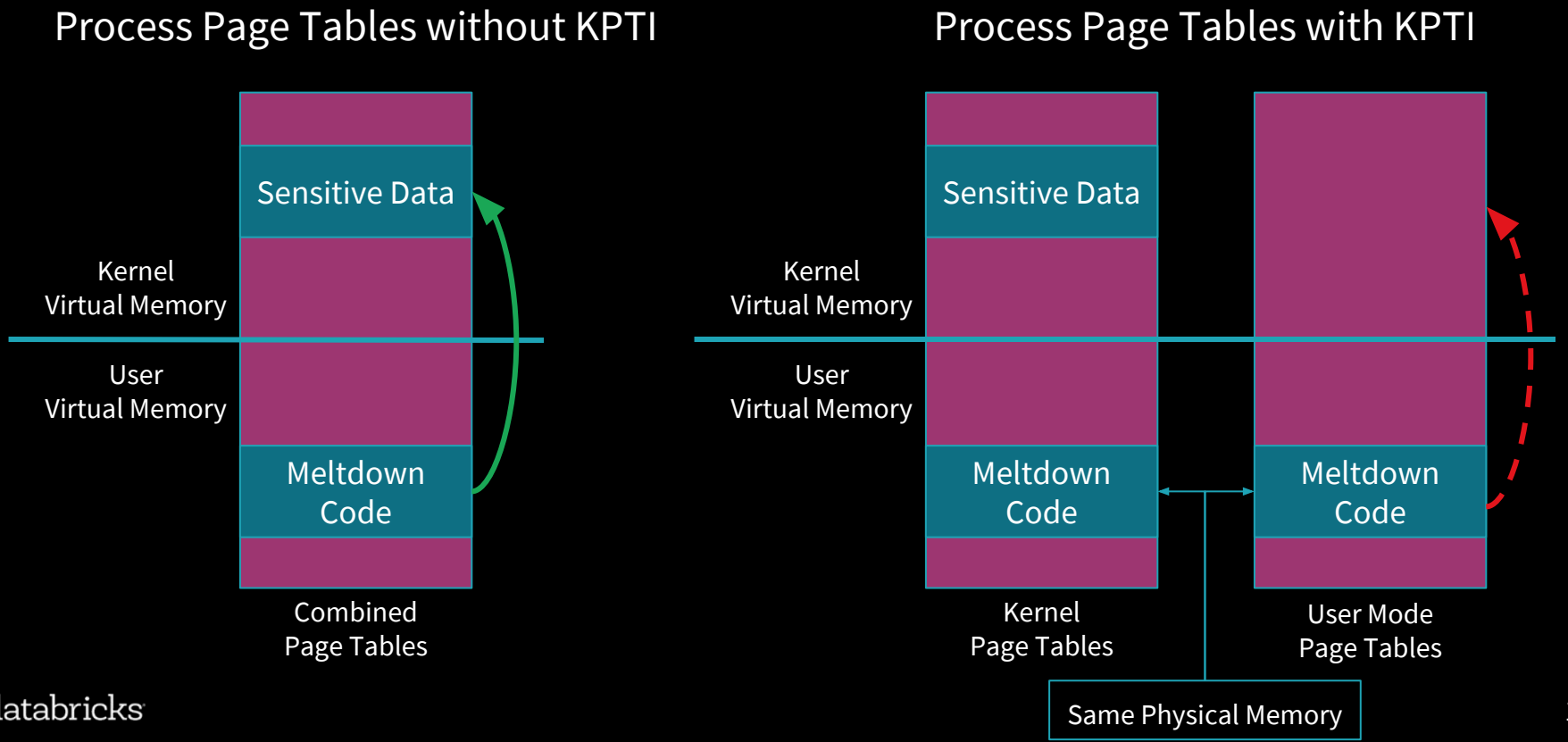
```
kernelByte = *kernelAddr;
probeArray[kernelByte * 4096];
```



Side-Channel Attack

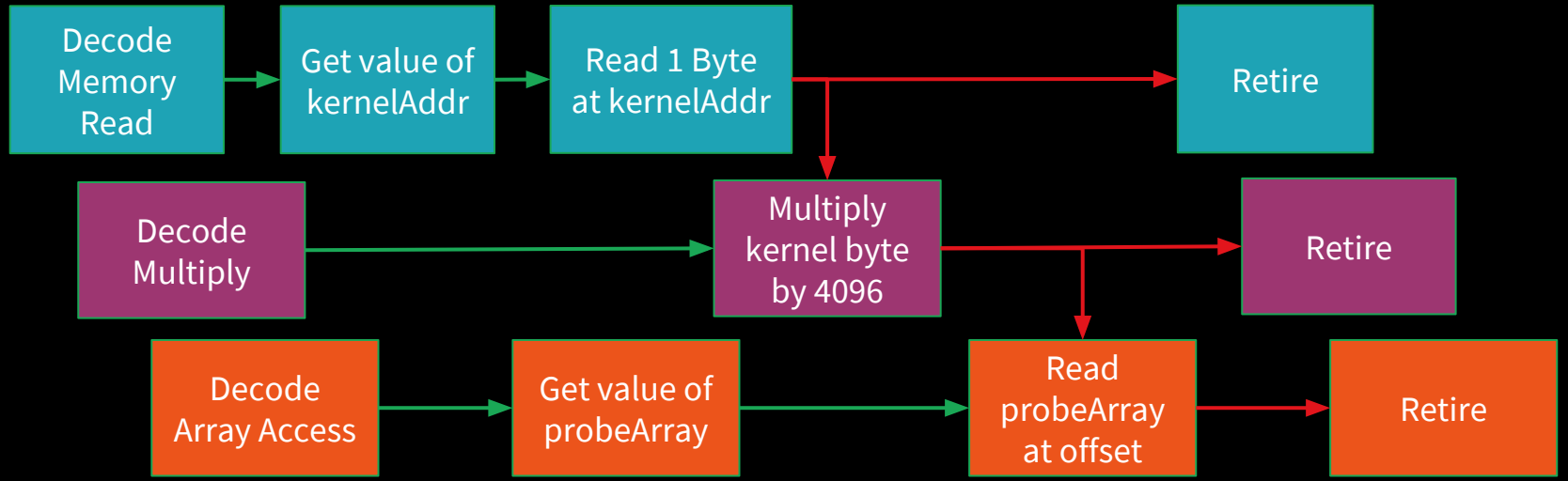


Kernel Page-Table Isolation

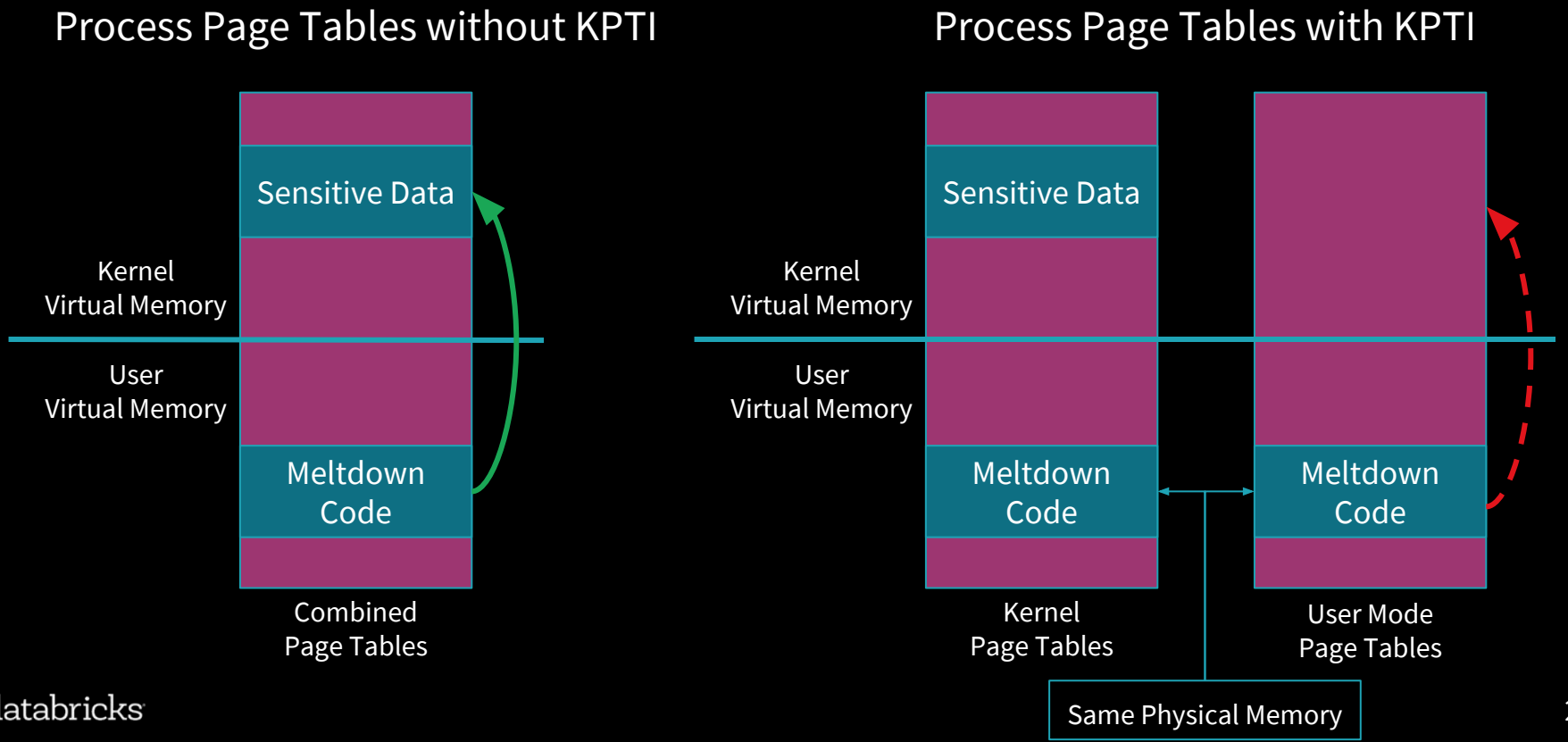


Meltdown

```
kernelByte = *kernelAddr;
probeArray[kernelByte * 4096];
```



Kernel Page-Table Isolation



TLB before KPTI

```
while True:
  print *UserAddress
  sys_clock_gettime
```

TLB MISS

Virtual Address	Physical Address
-	-
-	-

`print *UserAddress`

TLB MISS

Virtual Address	Physical Address
UserAddress	Page 1
-	-

`sys_clock_gettime`

TLB HIT

Virtual Address	Physical Address
UserAddress	Page 1
KernelTime	Page 2

`print *UserAddress`

TLB with KPTI

```
while True:
  print *UserAddress
  sys_clock_gettime
```

TLB MISS

Virtual Address	Physical Address
-	-
-	-

`print *UserAddress`

TLB MISS

Virtual Address	Physical Address
-	-
-	-

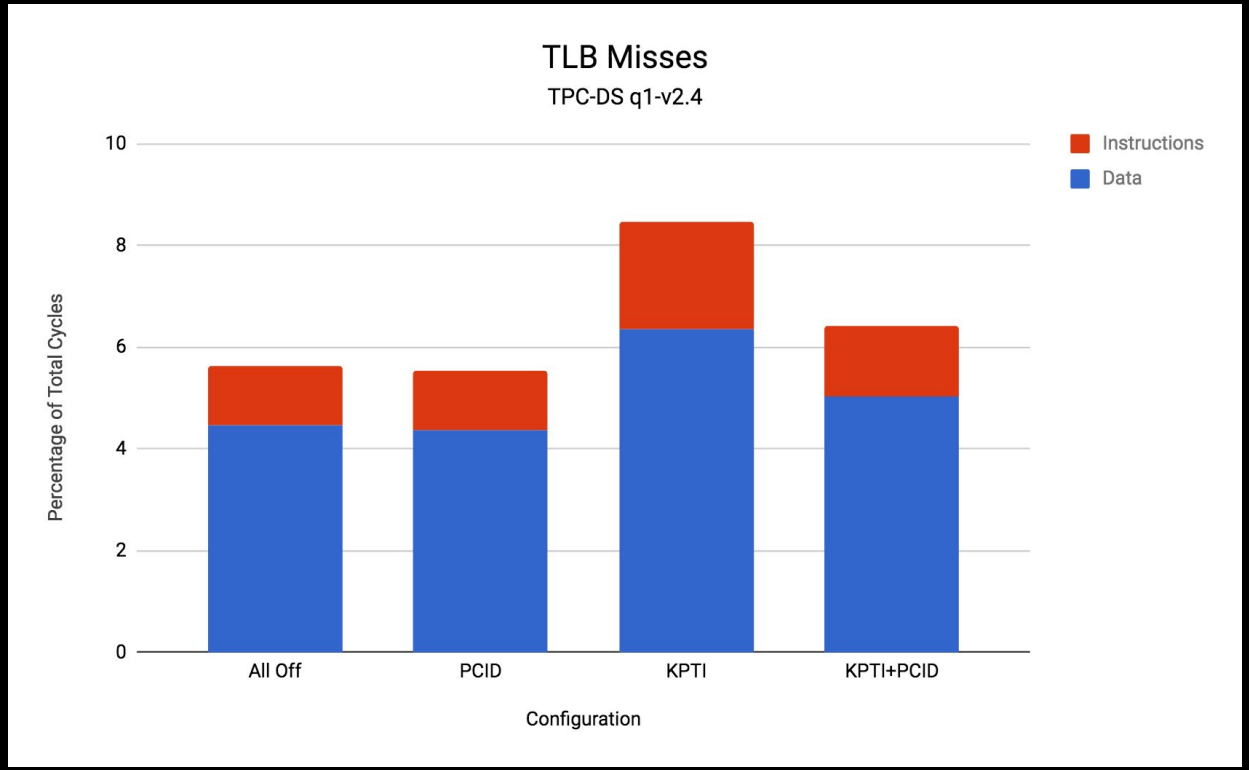
`sys_clock_gettime`

TLB MISS

Virtual Address	Physical Address
-	-
-	-

`print *UserAddress`

Meltdown TLB Misses




```
perf stat -e dtlb-load-misses
```

TLB with KPTI and PCID

```
while True:
  print *UserAddress
  sys_clock_gettime
```

TLB MISS

Virtual Address	PCID	Physical Address
-	-	-
-	-	-

`print *UserAddress`

TLB MISS

Virtual Address	PCID	Physical Address
UserAddress	1	Page 1
-	-	-

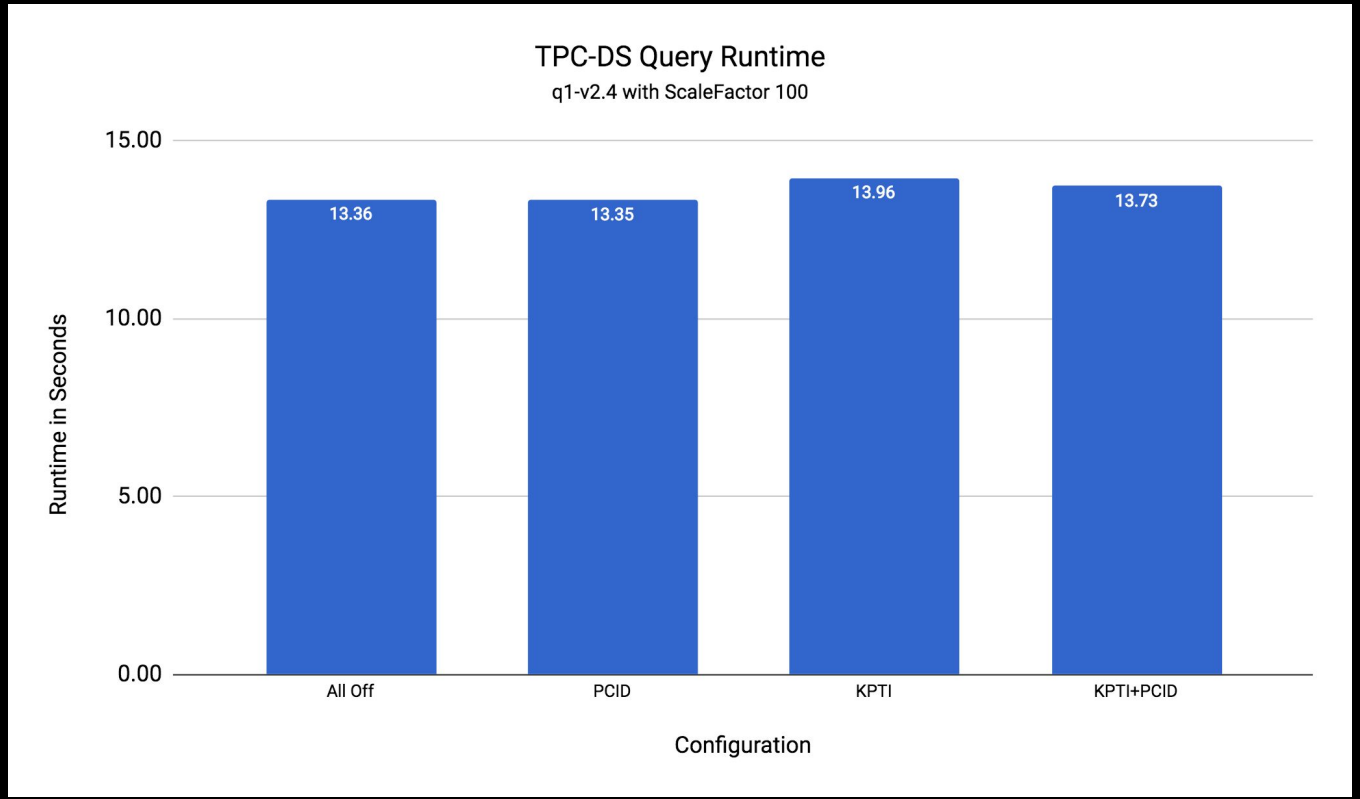
`sys_clock_gettime`

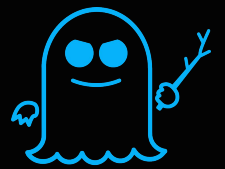
TLB HIT

Virtual Address	PCID	Physical Address
UserAddress	1	Page 1
KernelTime	0	Page 2

`print *UserAddress`

Meltdown Runtime



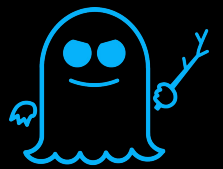


Spectre V1

Bounds Check Bypass

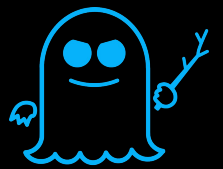
databricks

Bounds Check Bypass

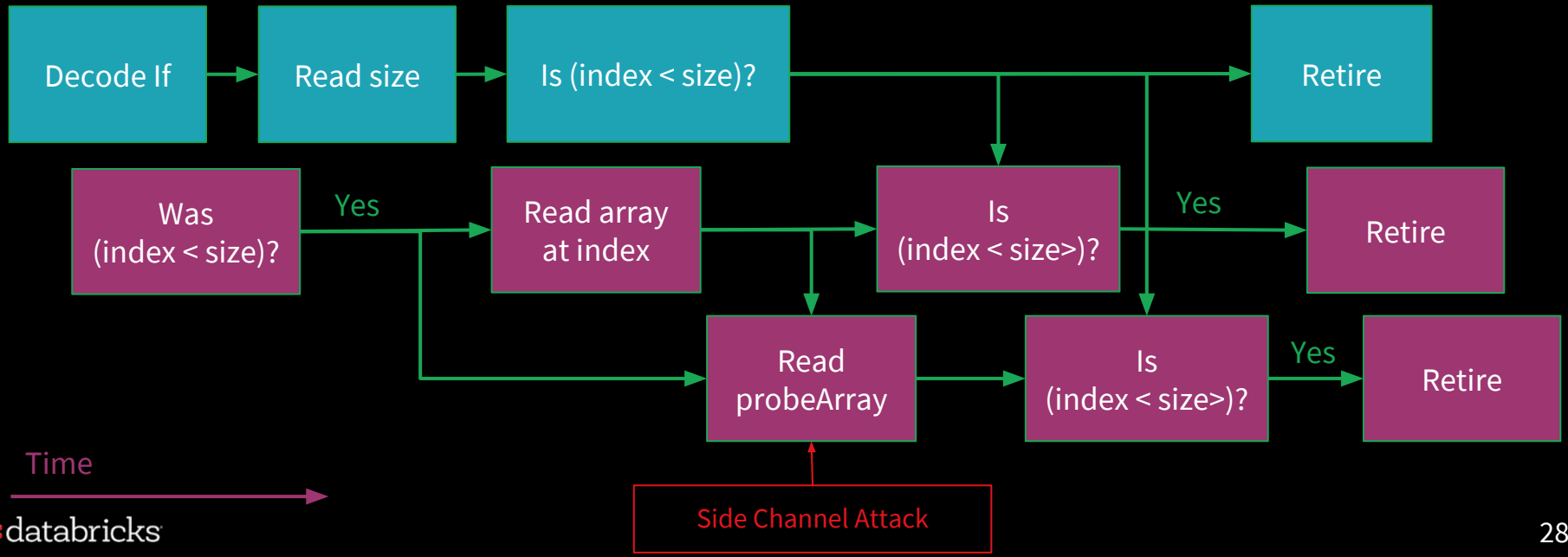


```
if (index < size) {
  val = array[index];
  probeArray[val];
}
```

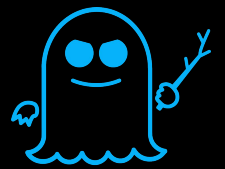
Bounds Check Bypass



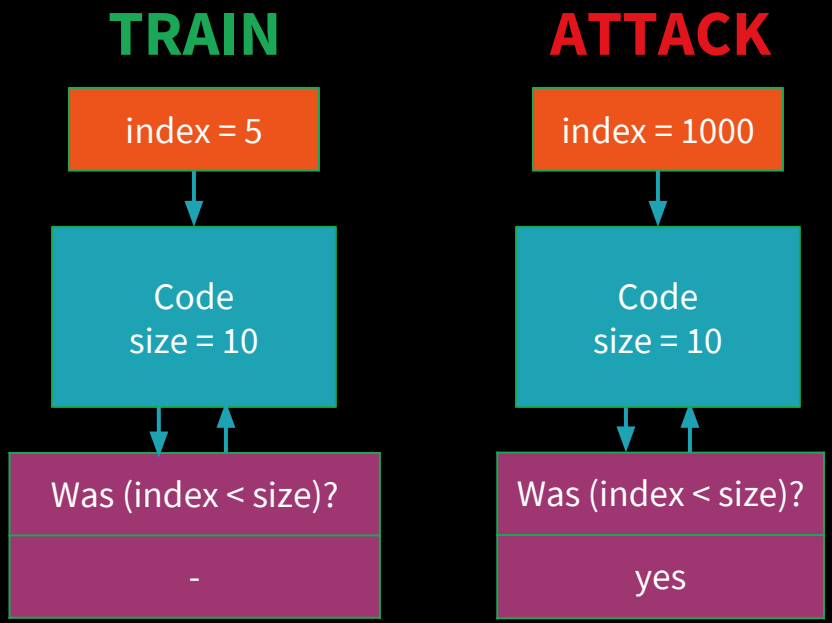
```
if (index < size) {
  val = array[index];
  probeArray[val];
}
```



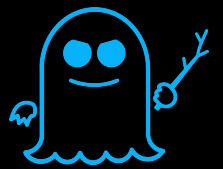
Bounds Check Bypass



```
if (index < size) {
  val = array[index];
  probeArray[val];
}
```



Observable Speculation Barrier



- Protects against Spectre V1 - Bounds Check Bypass
- ~14 in the Ubuntu kernel and drivers
- Stops speculative array access with the LFENCE barrier

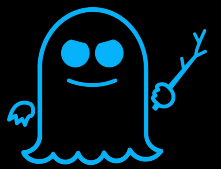
Before

```
if (index < size) {
  val = array[index];
  probeArray[val];
}
```

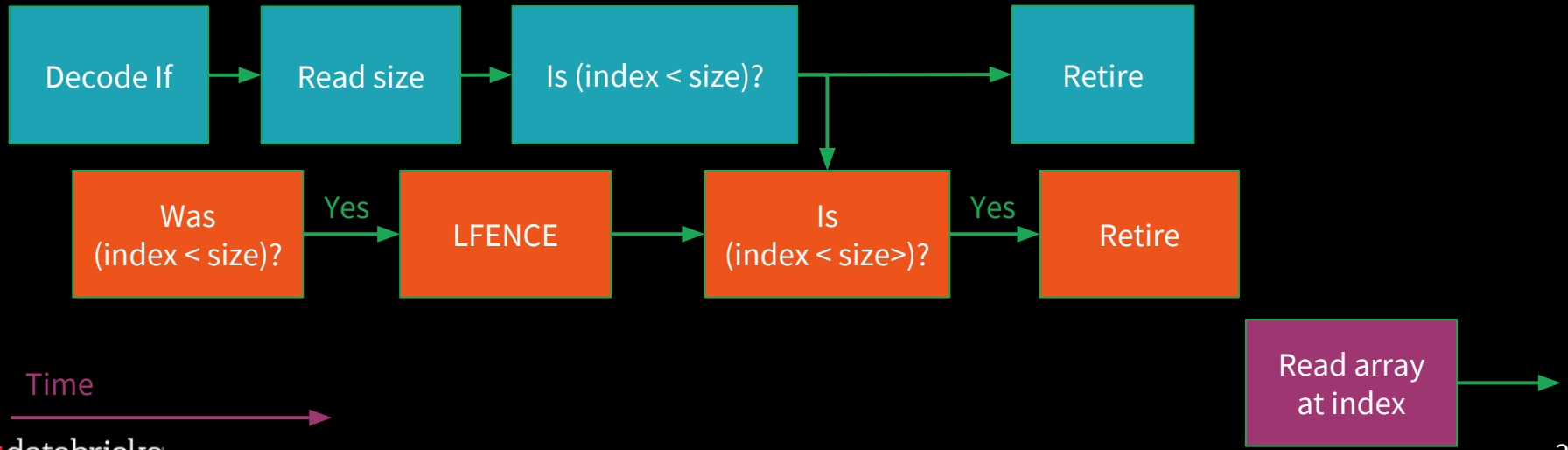
After

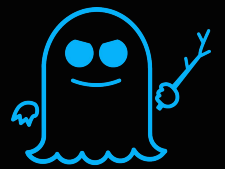
```
if (index < size) {
  osb();
  val = array[index];
  probeArray[val];
}
```

Observable Speculation Barrier



```
if (index < size) {
  osb()
  val = array[index];
  probeArray[val];
}
```



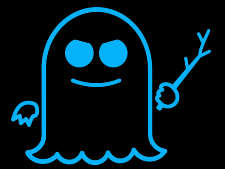


Spectre V2

Branch Target Injection

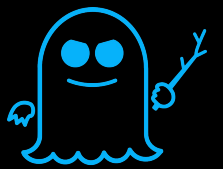
databricks

Branch Target Injection

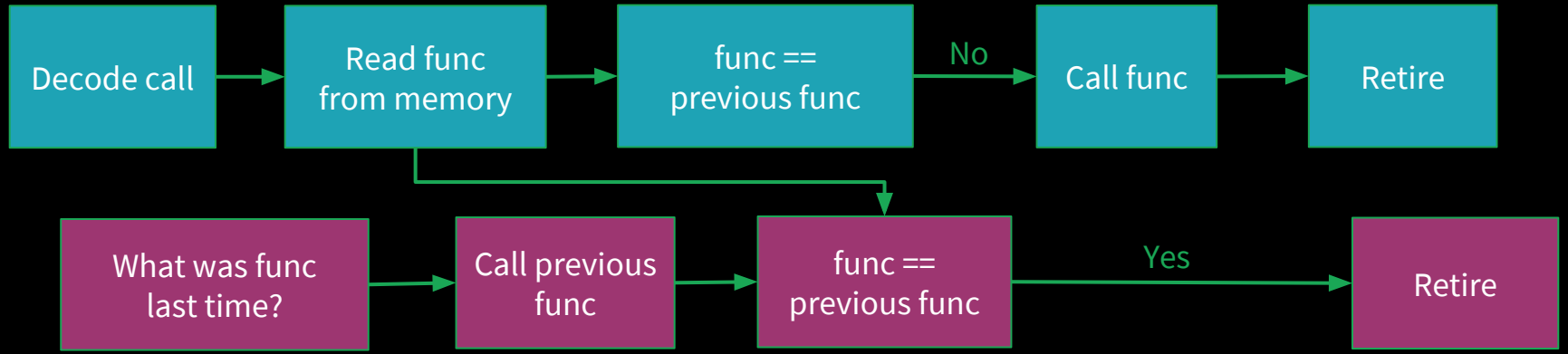


```
def call_func(func, arg):
  func(arg)
```

Branch Target Injection

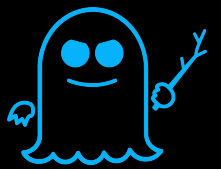


```
def call_func(func, arg):
  func(arg)
```

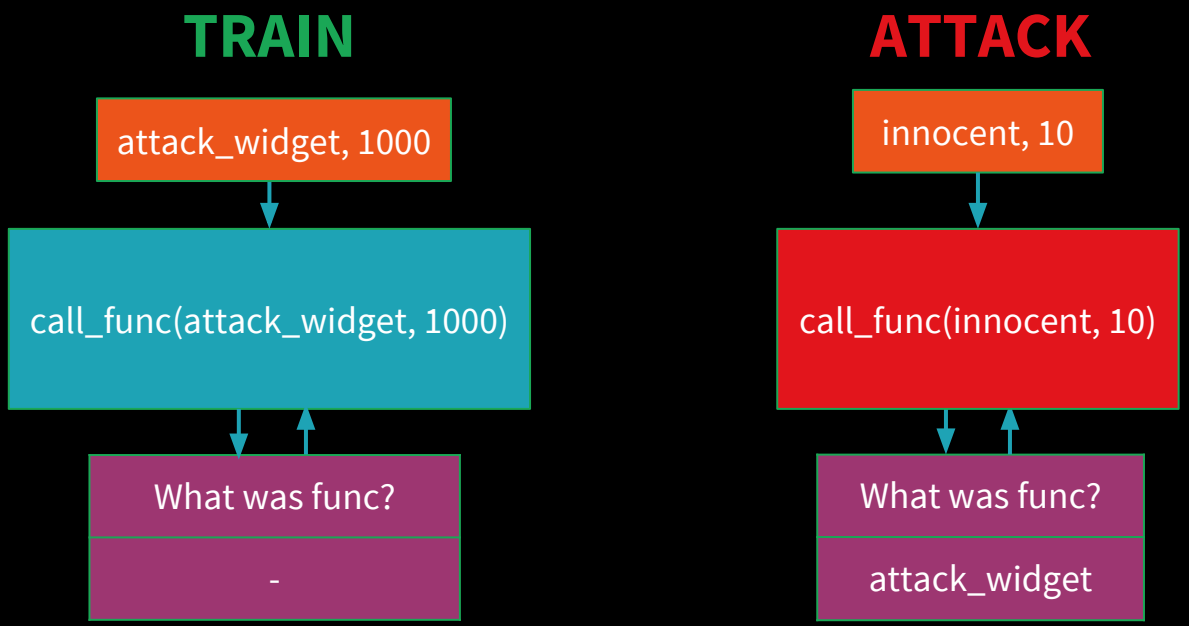


Time

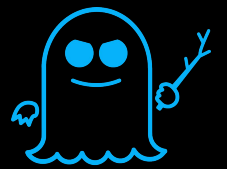
Branch Target Injection



```
def call_func(func, arg):
  func(arg)
```



Intel Microcode Updates

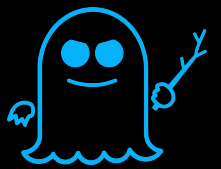


- Protect against Spectre V2 - Branch Target Injection

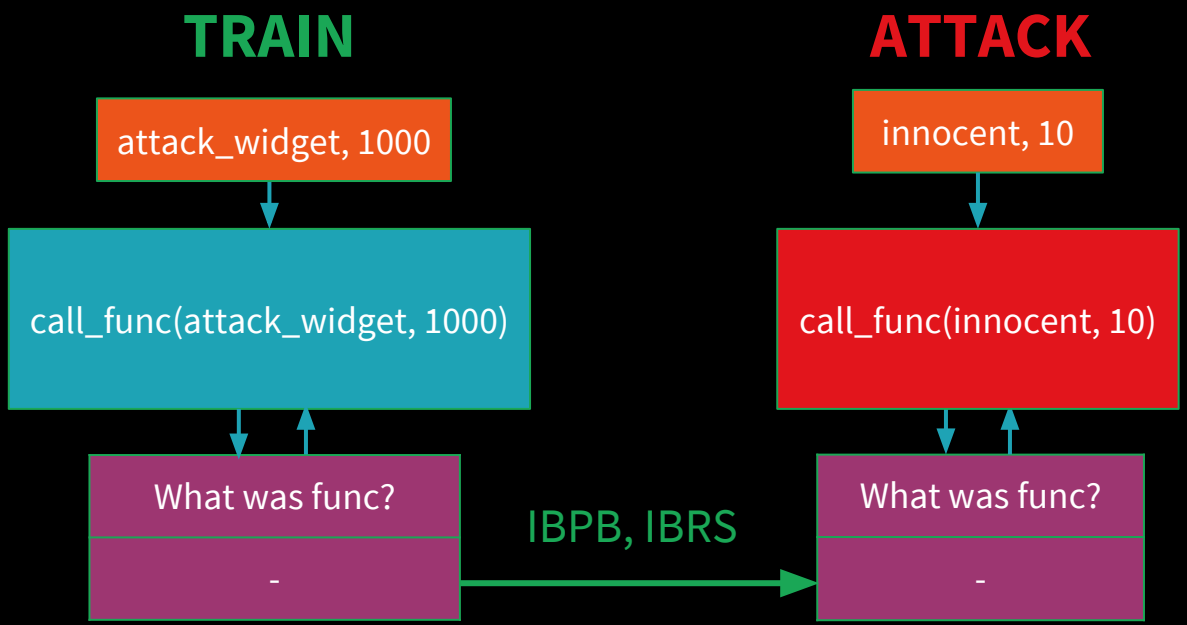
- Indirect Branch Restricted Speculation (IBRS)
  - Stops attacks from code running at lower privilege levels
  - Stops attacks from code running on the sibling hyperthread (STIBP)

- Indirect Branch Prediction Barrier (IBPB)
  - Stops attacks from code running at the same privilege level
  - Inserted at User-to-User and Guest-to-Guest transitions

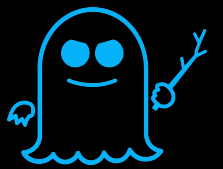
IBPB and IBRS Patches



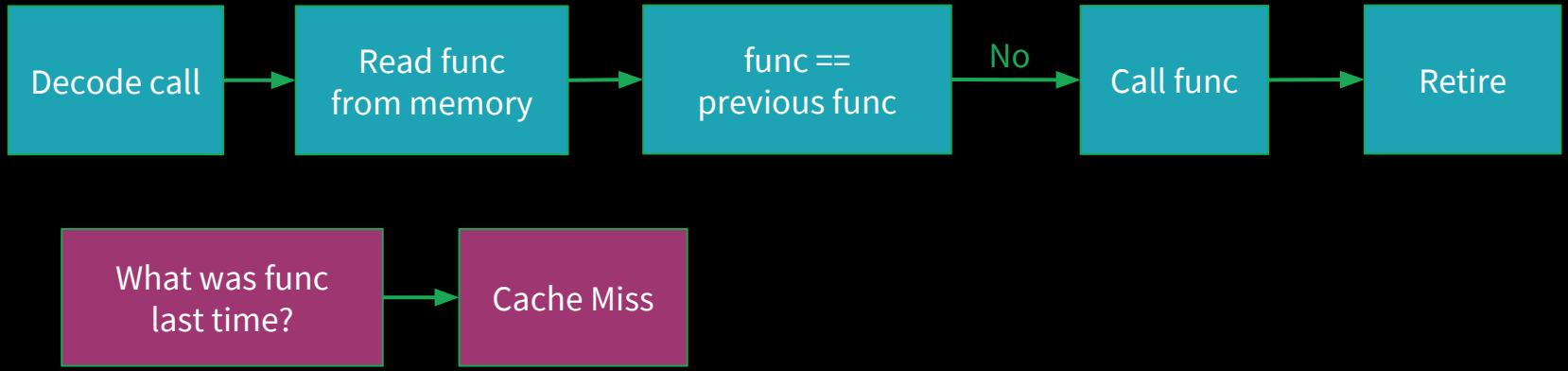
```
def call_func(func, arg):
  func(arg)
```



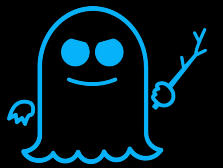
Branch Target Injection

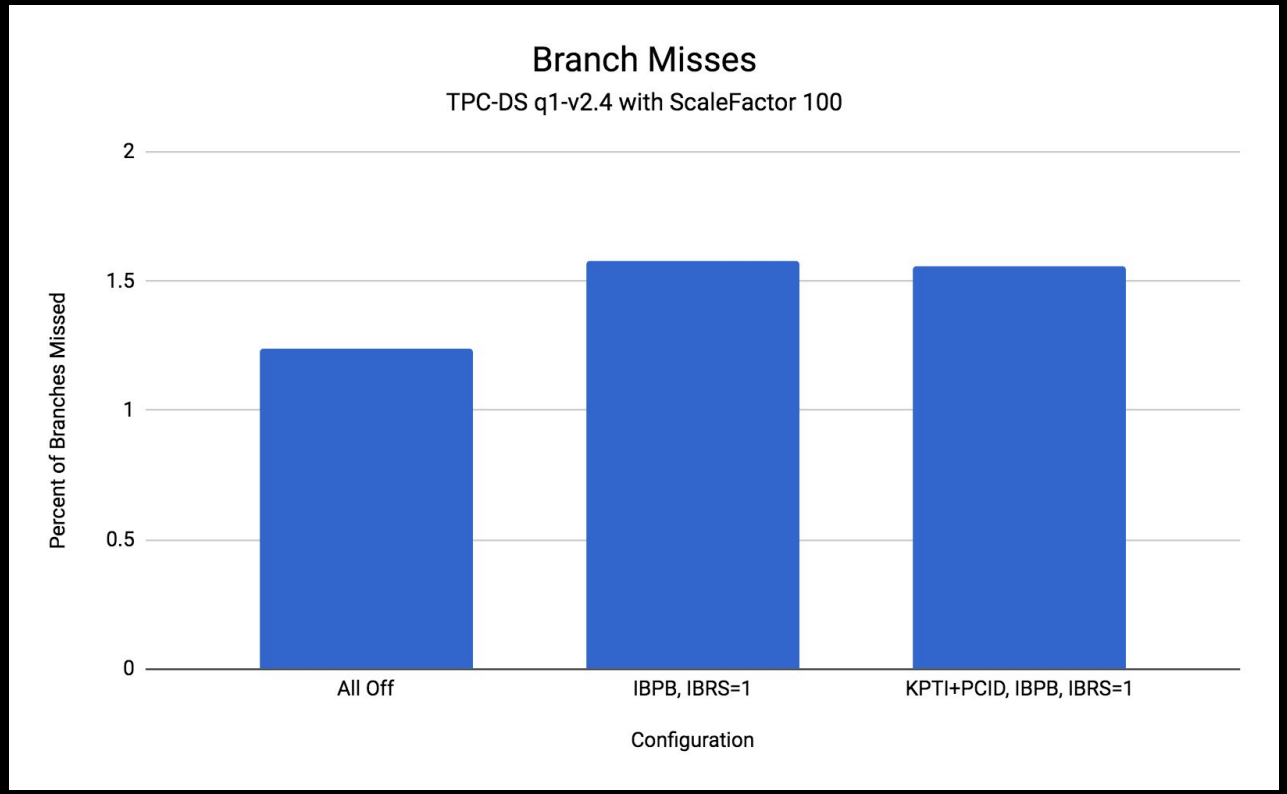


```
def call_func(func, arg):
  func(arg)
```



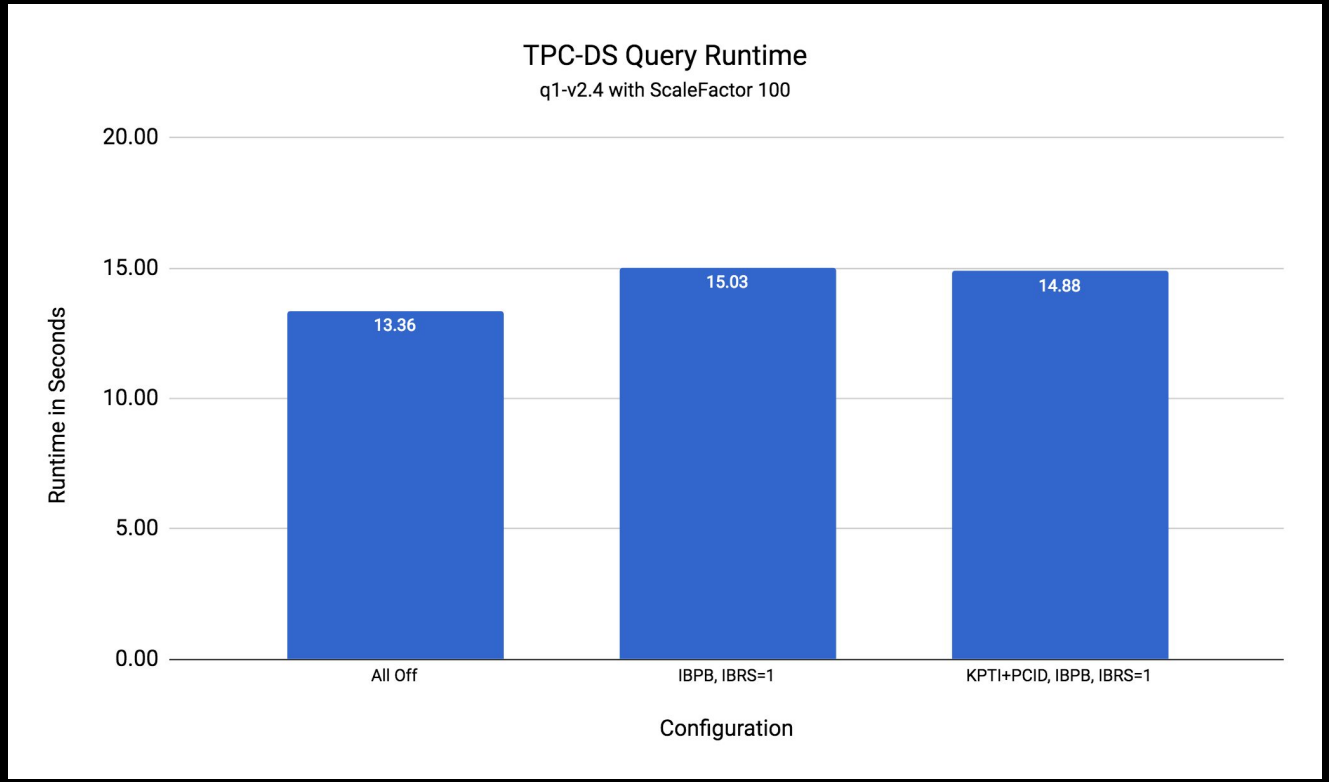
Spectre V2 Branch Misses



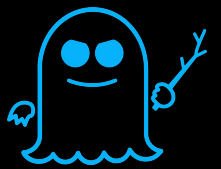


`perf stat -e branch-misses`

Spectre V2 Runtime



Retpoline



- Protects indirect branches/calls from speculative execution
- Uses an infinite loop to do it

Before

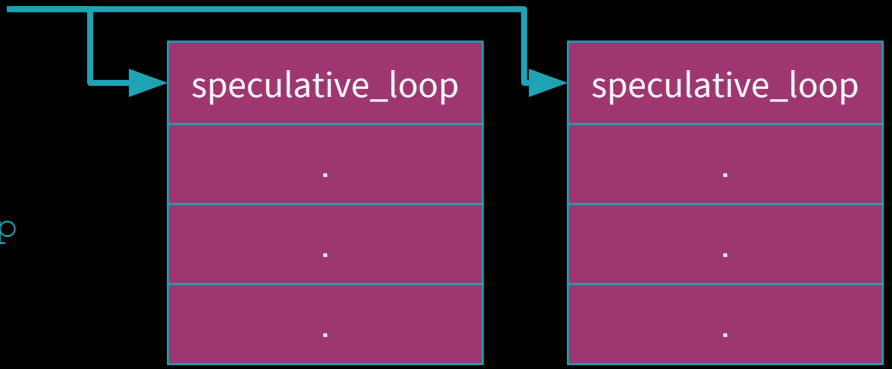
```
jmp rax
```

After

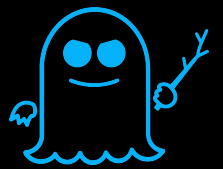
```
  call set_up_target

speculative_loop:
  pause
  lfence
  jmp speculative_loop

set_up_target:
  mov rax, [rsp]
  ret
```

Return Stack Buffer

Stack



Retpoline



- Protects indirect branches/calls from speculative execution
- Uses an infinite loop to do it

Before

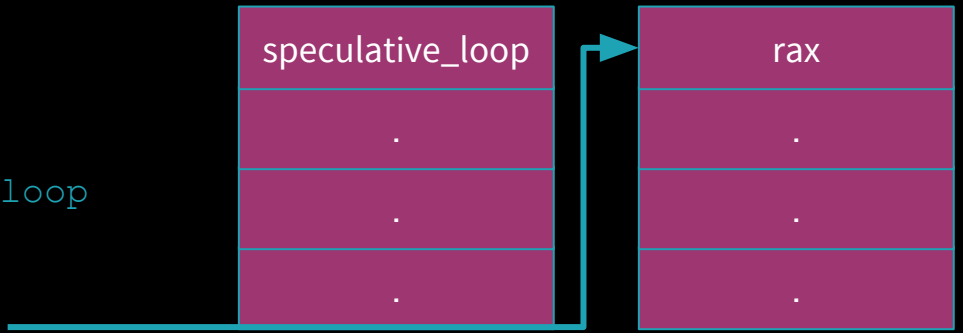
```
jmp rax
```

After

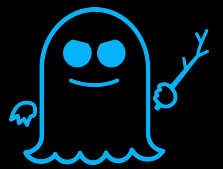
```
  call set_up_target

speculative_loop:
  pause
  lfence
  jmp speculative_loop

set_up_target:
  mov rax, [rsp]
  ret
```

Return Stack Buffer

Stack



Retpoline



- Protects indirect branches/calls from speculative execution
- Uses an infinite loop to do it

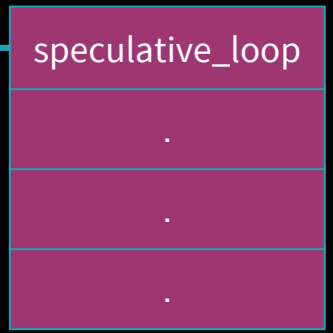
Before

```
jmp rax
```

After

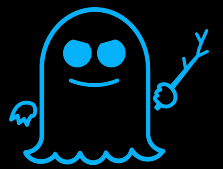
```
  call set_up_target

speculative_loop:
  pause
  lfence
  jmp speculative_loop

set_up_target:
  mov rax, [rsp]
  ret
```

Return Stack Buffer



Stack

Retpoline



- Protects indirect branches/calls from speculative execution
- Uses an infinite loop to do it

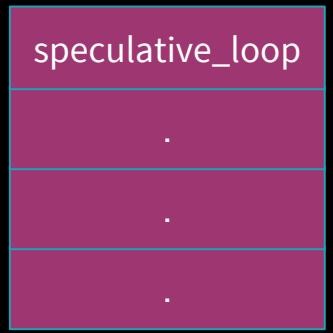
Before

```
jmp rax
```

After

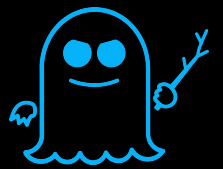
```
  call set_up_target

speculative_loop:
  pause
  lfence
  jmp speculative_loop

set_up_target:
  mov rax, [rsp]
  ret
```

Return Stack Buffer



Stack

Retpoline



- Protects indirect branches/calls from speculative execution
- Uses an infinite loop to do it

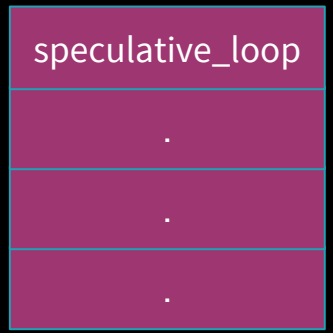
Before

```
jmp rax
```

After

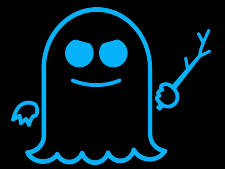
```
  call set_up_target

speculative_loop:
  pause
  lfence
  jmp speculative_loop

set_up_target:
  mov rax, [rsp]
  ret
```

Return Stack Buffer



Stack

Retpoline (Improved)



- Check against a known target and direct branch

Before

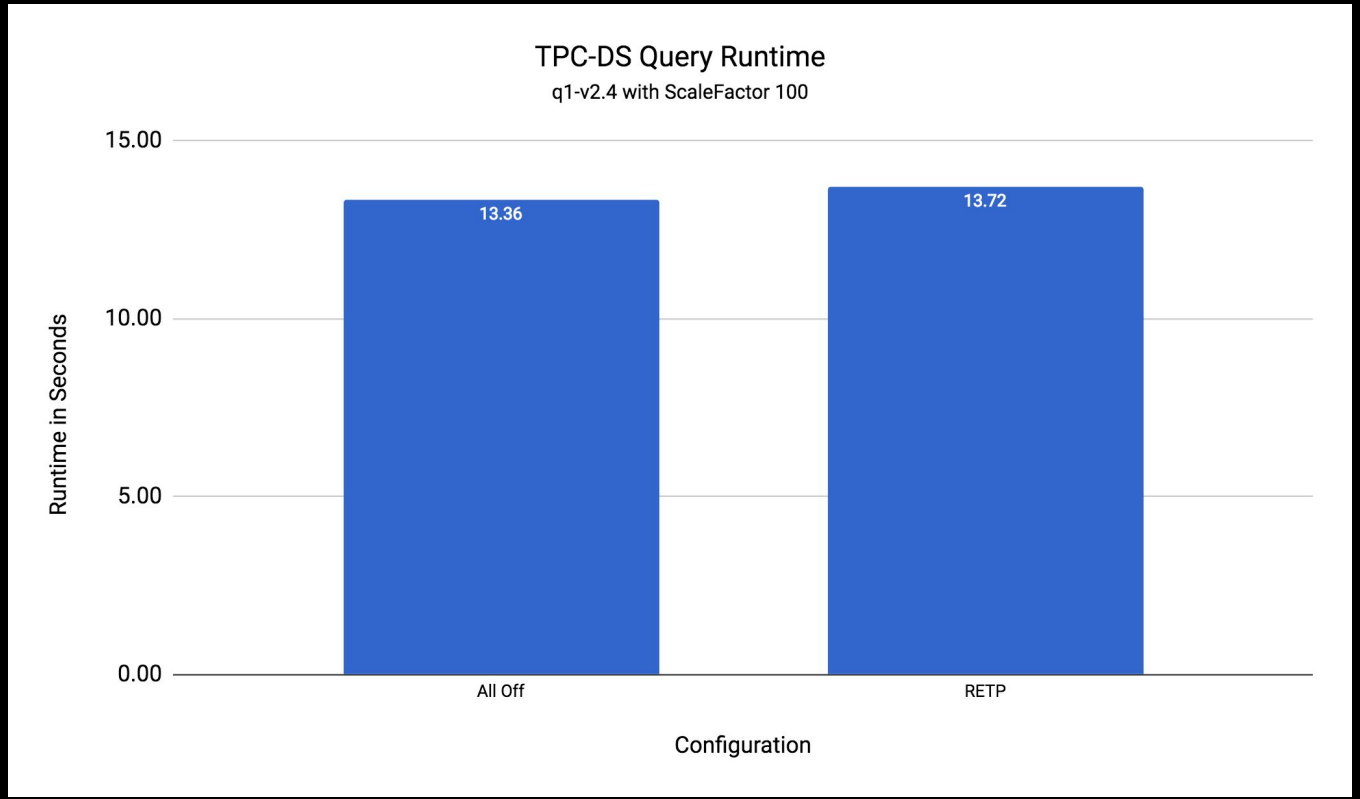
```
jmp rax
```

After

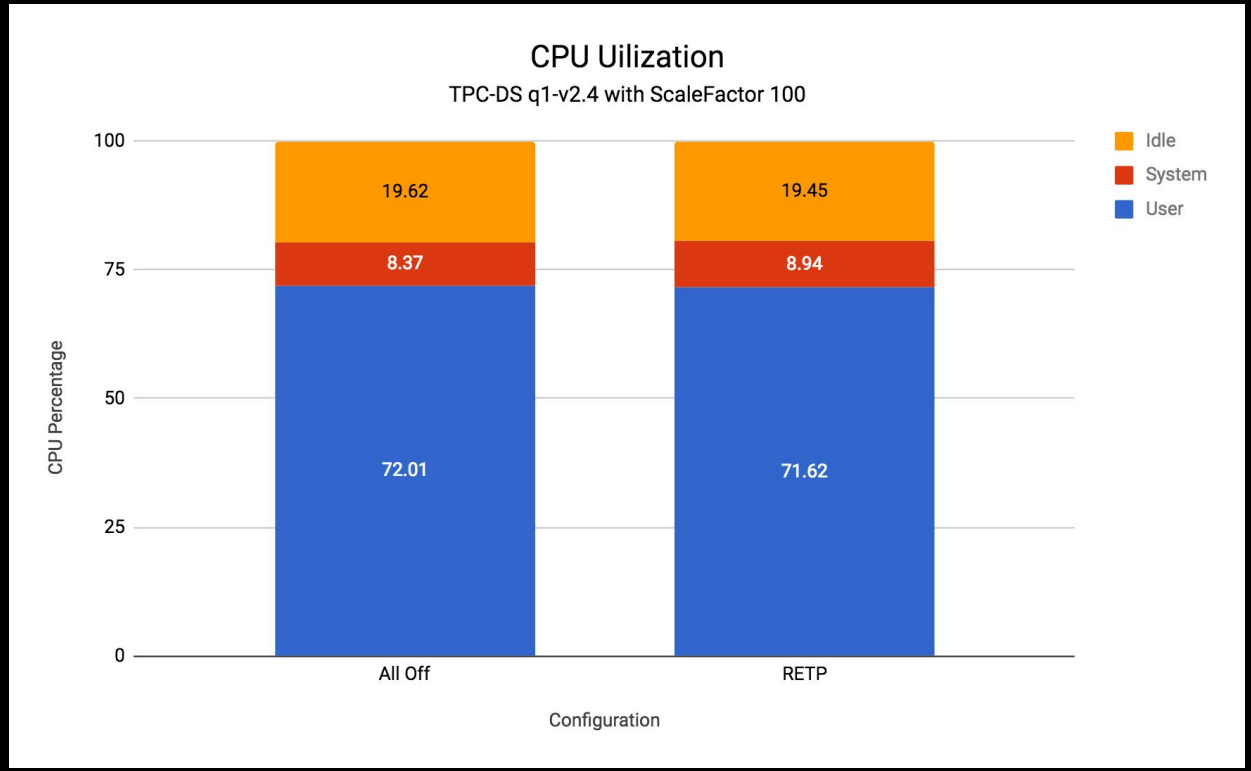
```
  cmp rax, known_target
  jne retpoline
  jmp known_target

retpoline:
  call set_up_target

speculative_loop:
  pause
  jmp speculative_loop

set_up_target:
  mov rax, [rsp]
  ret
```

Direct Branch (fast)

Retpoline Runtime

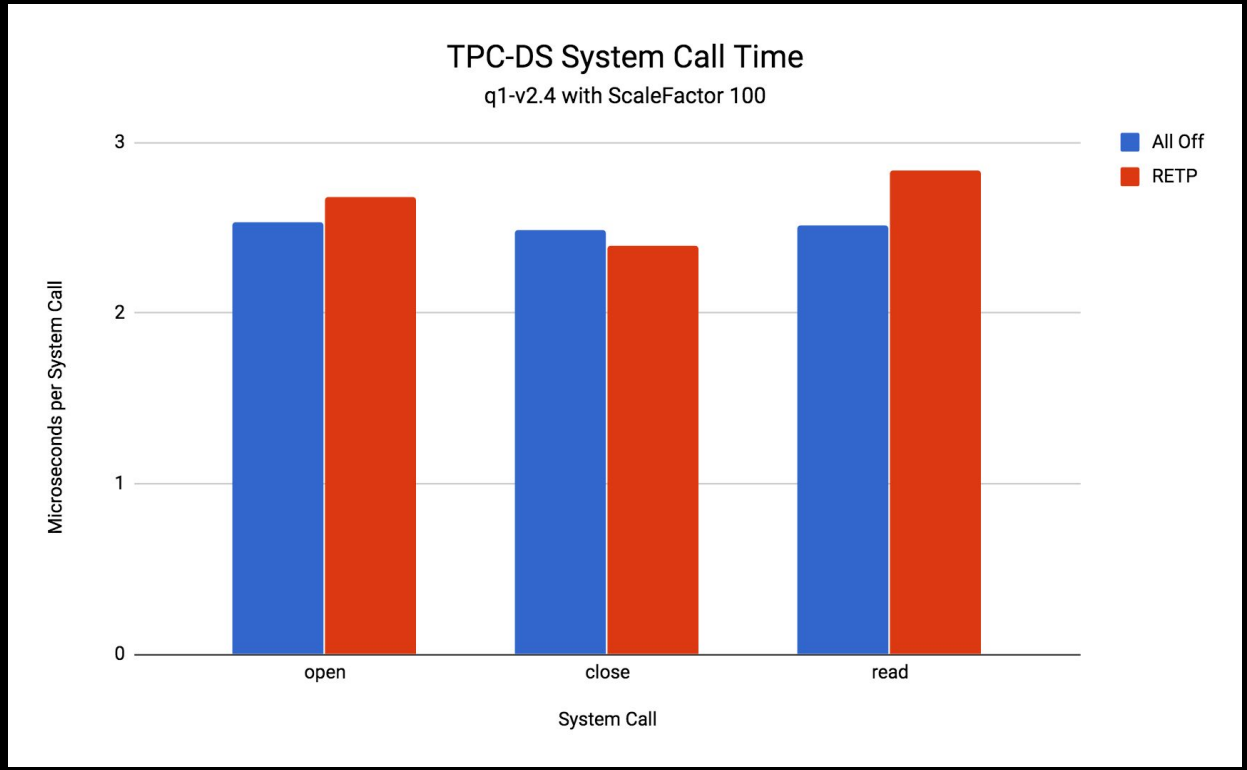


Retpoline CPU Utilization



mpstat -P ALL

Retpoline System Calls




```
strace -f -c -p PID
```

A Tale of Two Queries

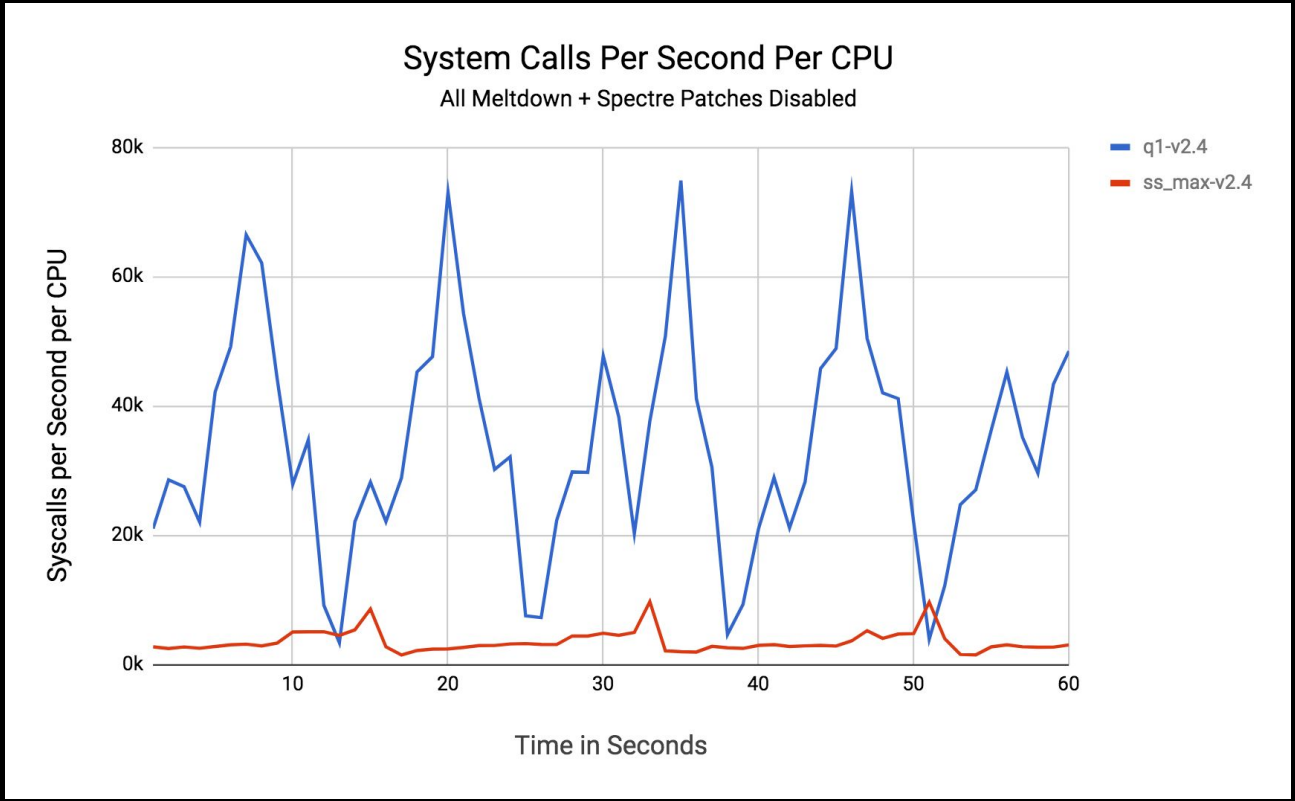
q1-v2.4

```
WITH customer_total_return AS
  (SELECT sr_customer_sk AS ctr_customer_sk,
          sr_store_sk AS ctr_store_sk,
          sum(sr_return_amt) AS ctr_total_return
   FROM store_returns, date_dim
   WHERE sr_returned_date_sk = d_date_sk
   AND d_year = 2000
   GROUP BY sr_customer_sk, sr_store_sk)
SELECT c_customer_id
  FROM customer_total_return ctr1, store, customer
  WHERE ctr1.ctr_total_return >
    (SELECT avg(ctr_total_return)*1.2
     FROM customer_total_return ctr2
     WHERE ctr1.ctr_store_sk = ctr2.ctr_store_sk)
  AND s_store_sk = ctr1.ctr_store_sk
  AND s_state = 'TN'
  AND ctr1.ctr_customer_sk = c_customer_sk
  ORDER BY c_customer_id LIMIT 100
```

ss_max-v2.4

```
SELECT
  count(*) AS total,
  count(ss_sold_date_sk) AS not_null_total,
  count(DISTINCT ss_sold_date_sk) AS unique_days,
  max(ss_sold_date_sk) AS max_ss_sold_date_sk,
  max(ss_sold_time_sk) AS max_ss_sold_time_sk,
  max(ss_item_sk) AS max_ss_item_sk,
  max(ss_customer_sk) AS max_ss_customer_sk,
  max(ss_cdemo_sk) AS max_ss_cdemo_sk,
  max(ss_hdemo_sk) AS max_ss_hdemo_sk,
  max(ss_addr_sk) AS max_ss_addr_sk,
  max(ss_store_sk) AS max_ss_store_sk,
  max(ss_promo_sk) AS max_ss_promo_sk
FROM store_sales
```

q1-v2.4 *vs* ss_max-v2.4




```
perf stat -e raw_syscalls:sys_enter -a -I 1000
```

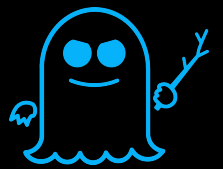
Thank You

chris.stevens@databricks.com
https://www.linkedin.com/in/chriscstevens/

databricks

Backup Slides

array_index_nospec()



- Protects against Spectre V1 - Bounds Check Bypass
- ~50 in the Linux kernel and drivers (not in Ubuntu)
- Stops speculative array access by clamping the index

Before

```
if (index < size) {
  val = array[index];
  probeArray[val];
}
```

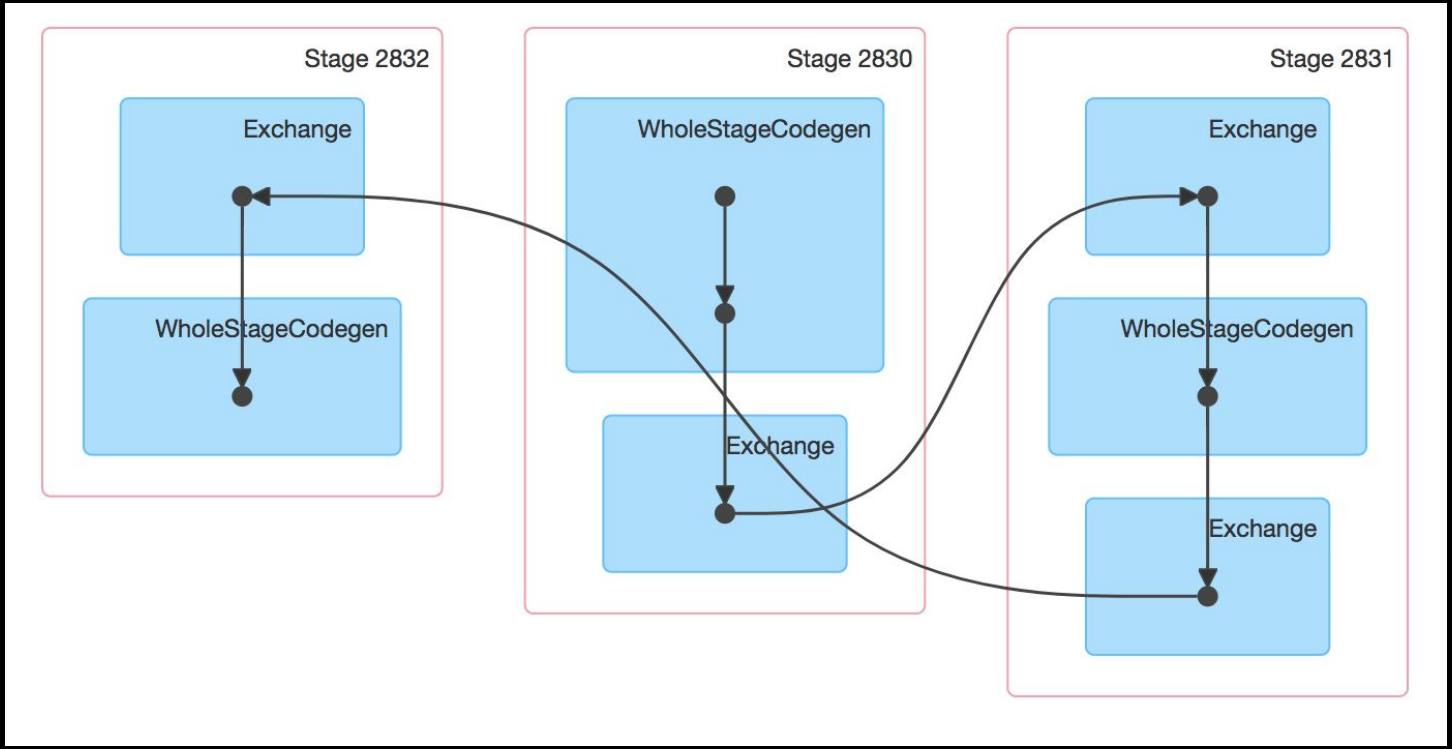
After

```
if (index < size) {
  index = array_index_nospec(index, size);
  val = array[index];
  probeArray[val];
}
```

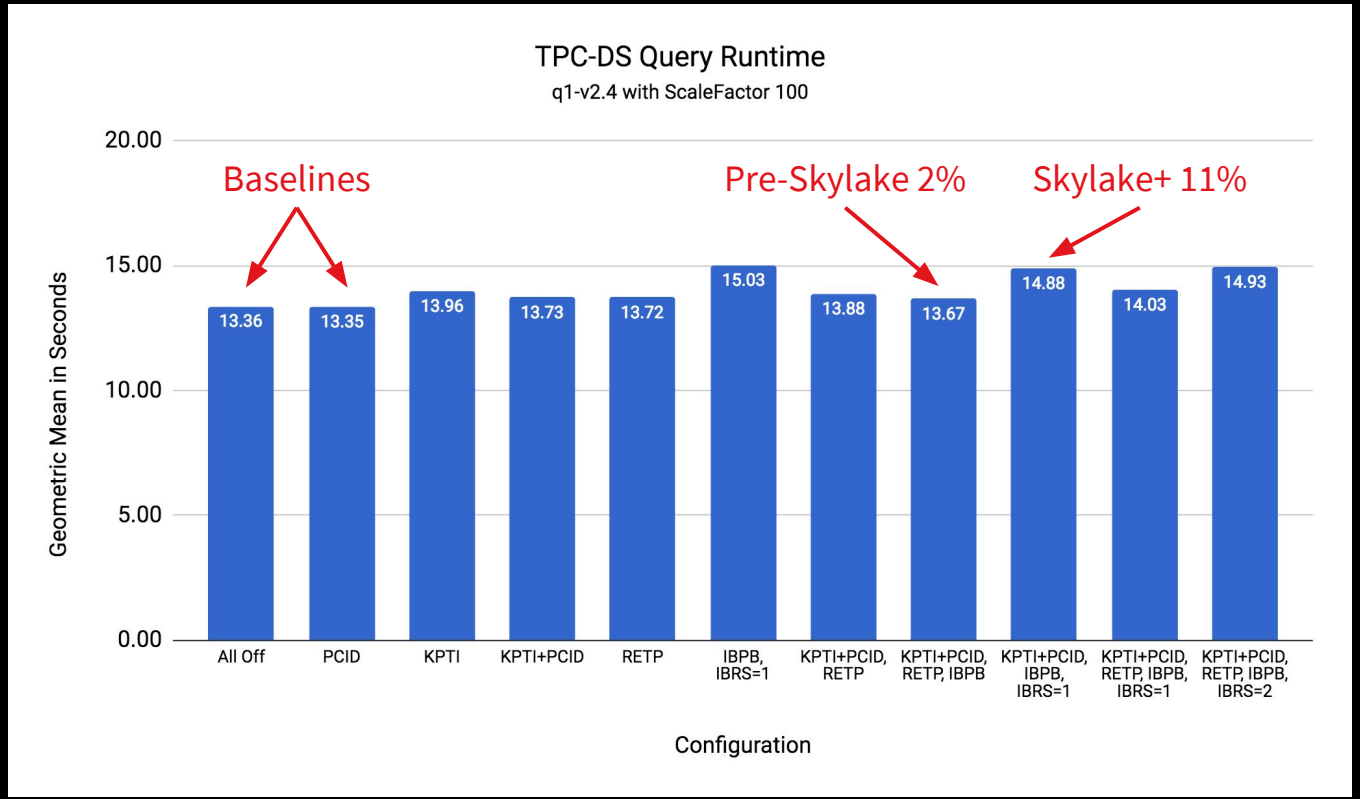
Speculative Store Bypass

- CVE-2018-3639
- “Variant 4”
- Public Date: May 21, 2018
- Details:
  - https://access.redhat.com/security/vulnerabilities/ssbd
- Ubuntu Command Line Parameter:
  - `spec_store_bypass_disable=on`

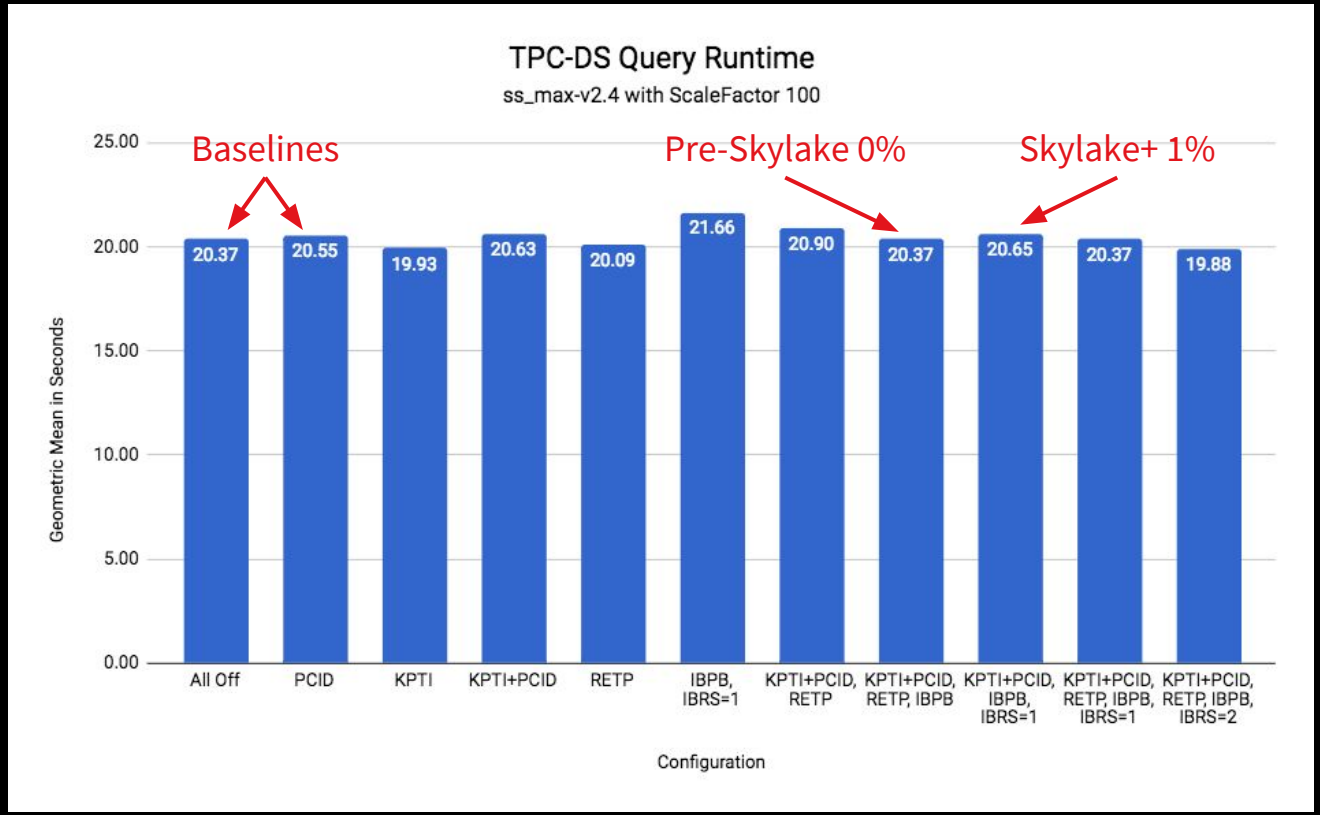
TPC-DS: ss_max-v2.4



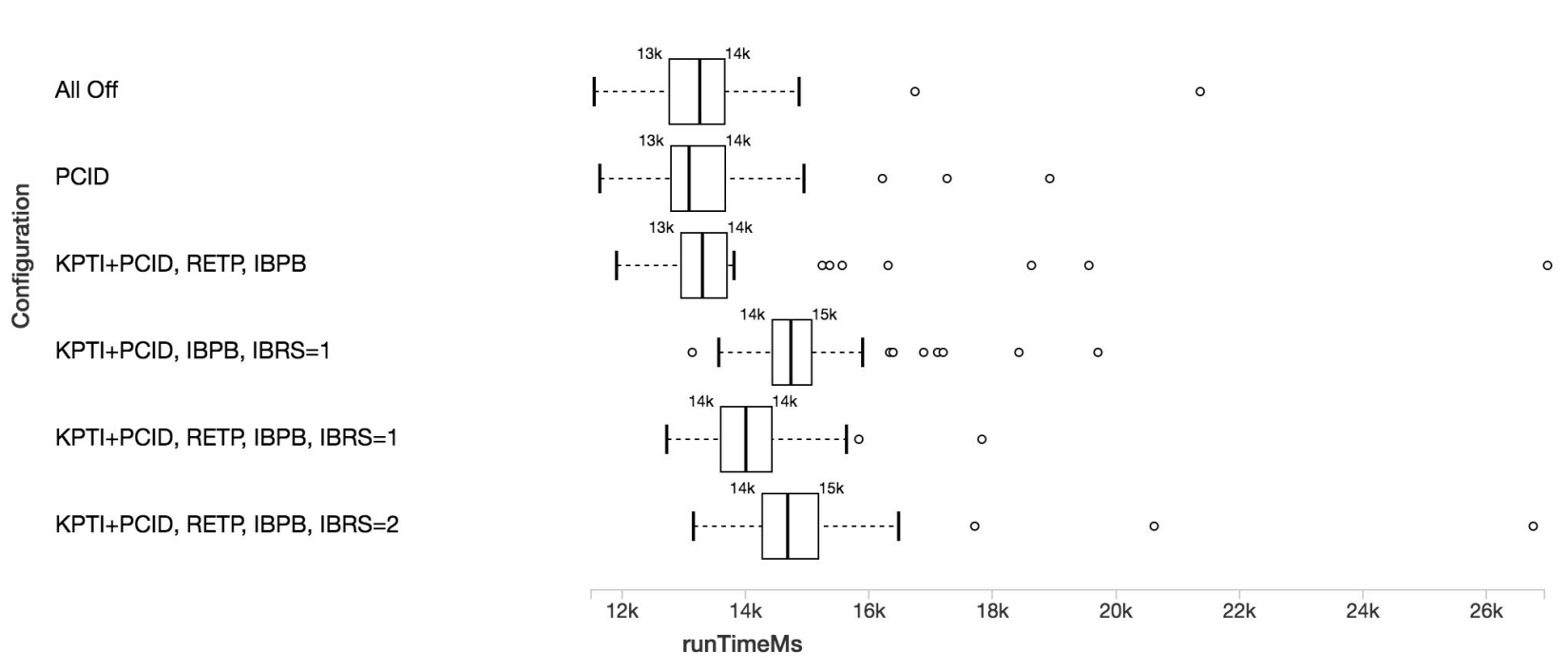
TPC-DS: q1-v2.4


TPC-DS Query Runtime
q1-v2.4 with ScaleFactor 100
Geometric Mean in Seconds
20.00
15.00
10.00
5.00
0.00
Baselines
Pre-Skylake 2%
Skylake+ 11%
13.36
13.35
13.96
13.73
13.72
15.03
13.88
13.67
14.88
14.03
14.93
All Off
PCID
KPTI
KPTI+PCID
RETP
IBPB, IBRS=1
KPTI+PCID, RETP
KPTI+PCID, RETP, IBPB
KPTI+PCID, IBPB, IBRS=1
KPTI+PCID, RETP, IBPB, IBRS=1
KPTI+PCID, RETP, IBPB, IBRS=2
Configuration

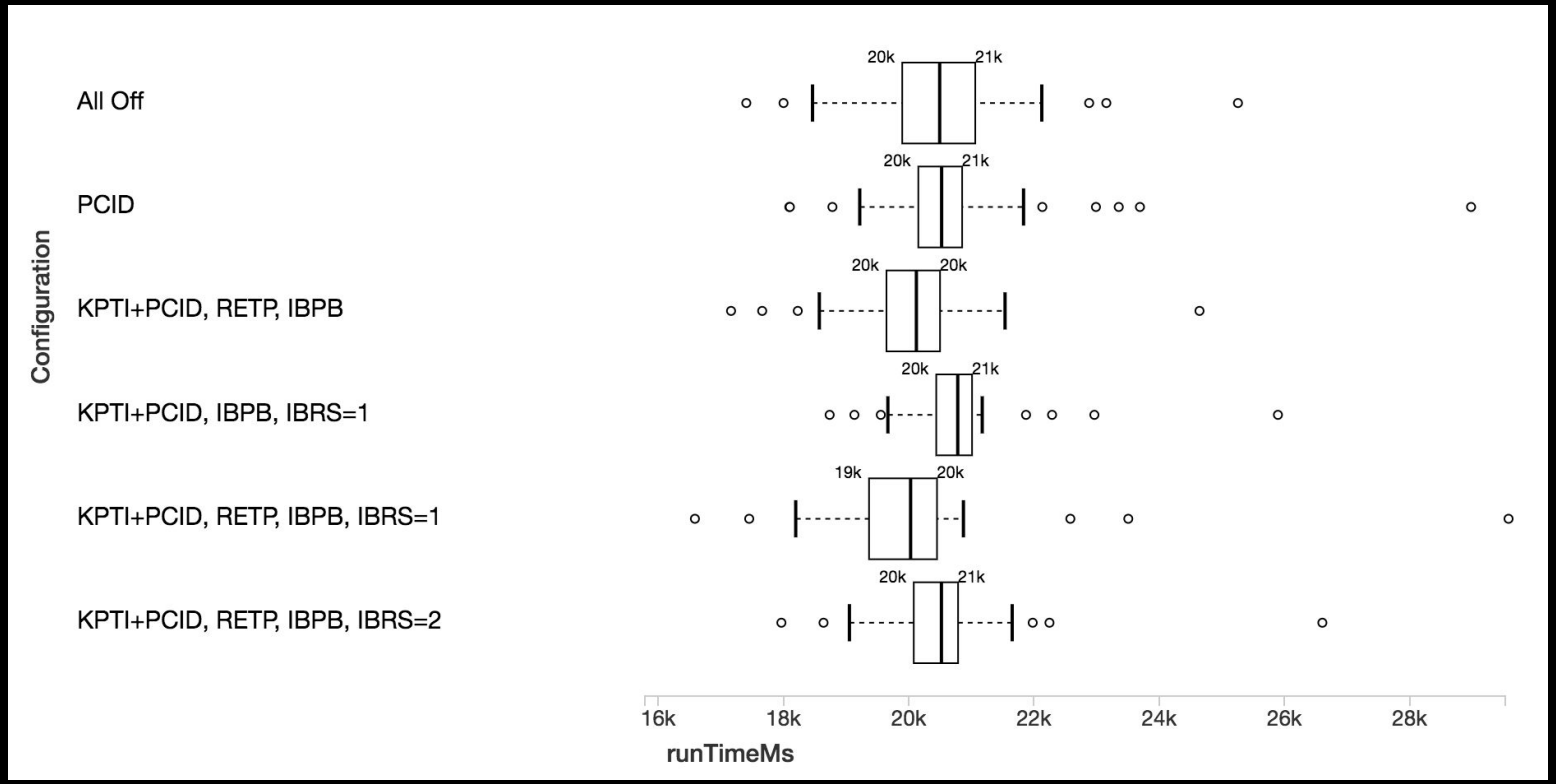
TPC-DS: ss_max-v2.4



q1-v2.4 Runtime Box Plots



ss_max-v2.4 Runtime Box Plots



Single Node TPC-DS Setup

- Intel Core i7-4820k @ 3.70GHz (Ivy Bridge - ca. 2012)
- 8 CPUs
- 8GB of RAM
- Disk: Western Digital Blue
  - WDC10EZEX-08M2NA0
  - 1TB
  - 7200 rpm
  - 146MB/s sequential read (http://hdd.userbenchmark.com/)
- Ubuntu 16.04.4 LTS (Xenial Xerus)
  - 64-bit server image
  - 4.4.0-116-generic Linux kernel

Repositories

- https://github.com/apache/spark.git
- https://github.com/databricks/spark-sql-perf.git
- https://github.com/databricks/tpcds-kit
- https://github.com/speed47/spectre-meltdown-checker
- https://github.com/brendangregg/pmc-cloud-tools
- https://github.com/brendangregg/perf-tools