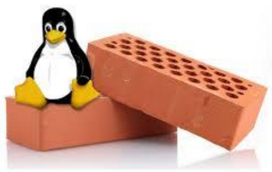
UNIVERSITY OF TORONTO

An Analysis of Performance Evolution of Linux's Core Operations

Xiang (Jenny) Ren, Kirk Rodrigues, Luyuan Chen, Camilo Vega, Michael Stumm and Ding Yuan

How has kernel performance been evolving?

Studying Linux kernel's performance evolution



Most time consuming kernel functions

LEBench | microbenchmark

Test Name	Input
Context switch	N/A
`fork`	0, 12K writeable pages
Thread create	N/A
Page fault	in region of 1, 10K pages
`read`, `write`	1, 10, 12K pages
`mmap`, `munmap`	1, 10, 10K pages
`send`, `recv`	1, 96K bytes
`select`, `poll`, `epoll`	10, 1K file descriptors

Studying Linux kernel's performance evolution

Software setup

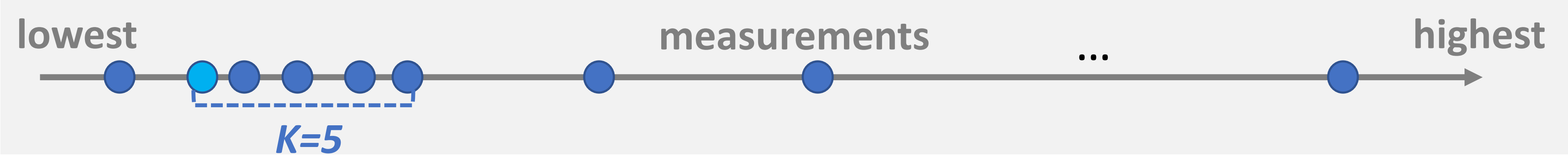
- **Linux v3.0 to v4.20** (**41** versions, covering **7 years**)
- **Ubuntu** distribution, default configuration

Machine setup

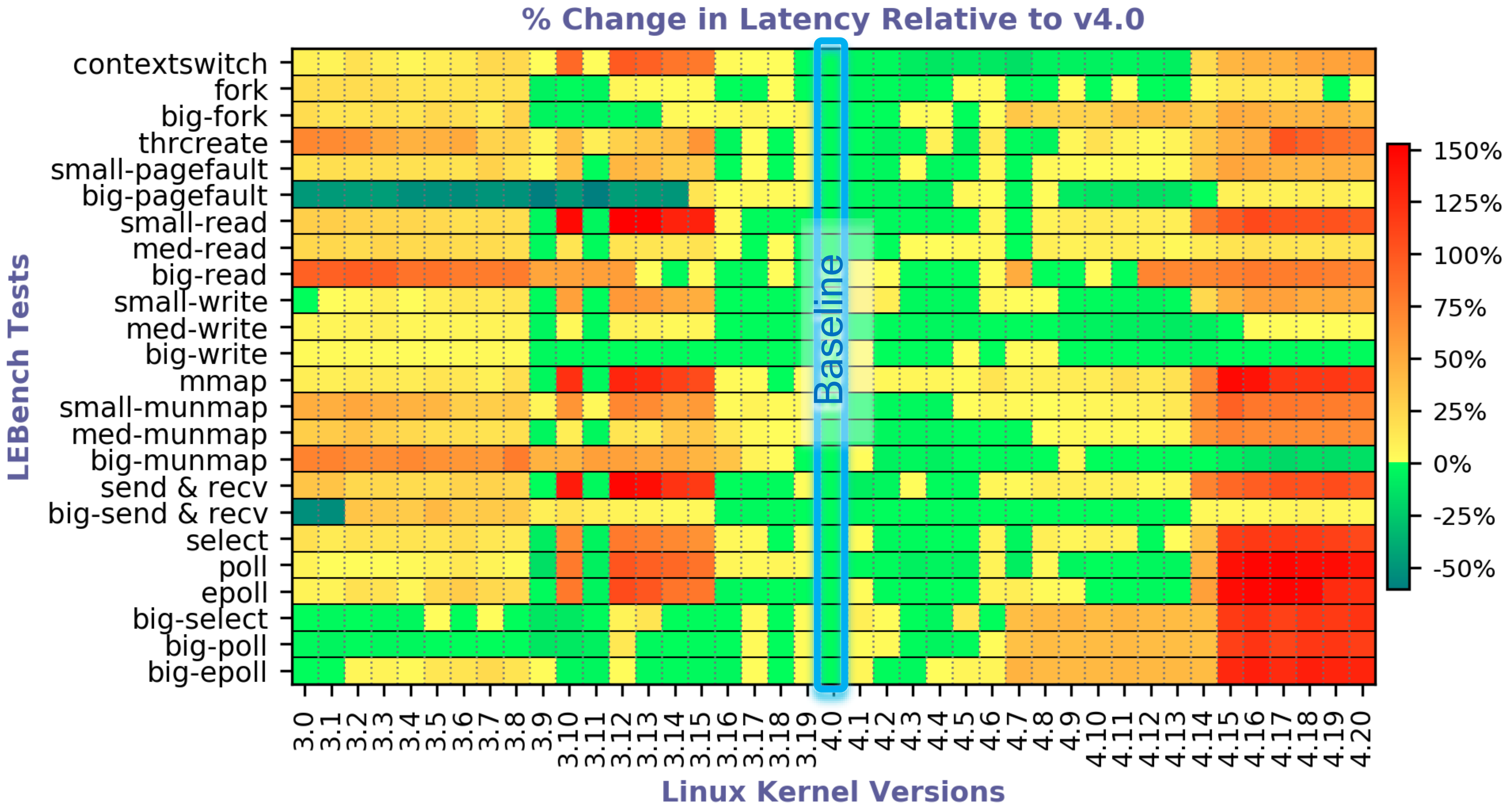
- LEBench run on 1 machine setup:
  2.40GHz Intel Xeon processor, 128GB 1866MHz DDR4 RAM, 960GB SSD

Result collection

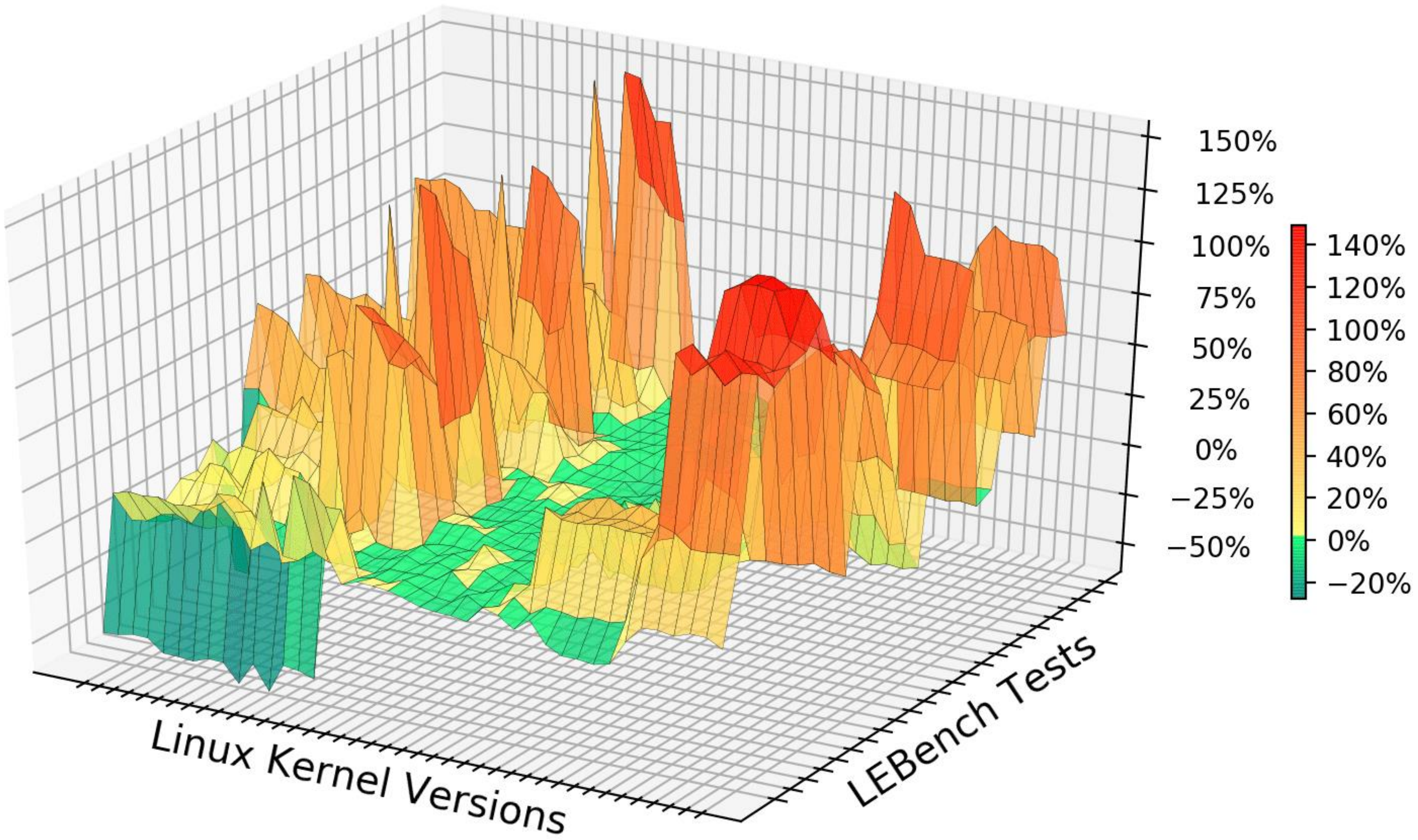
- *K-best* method to remove measurement outliers from 10K runs



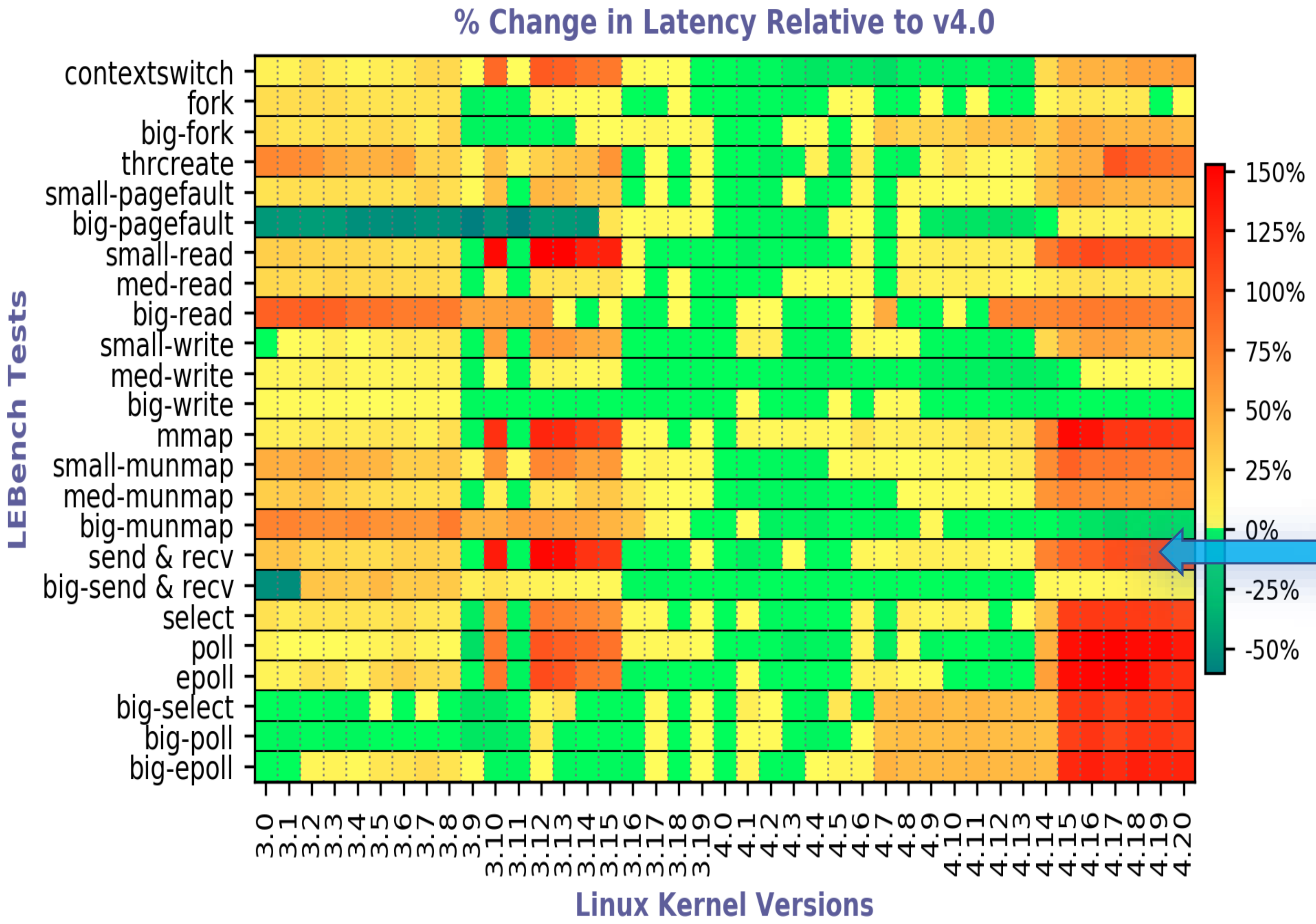
Linux core functions' performance evolution



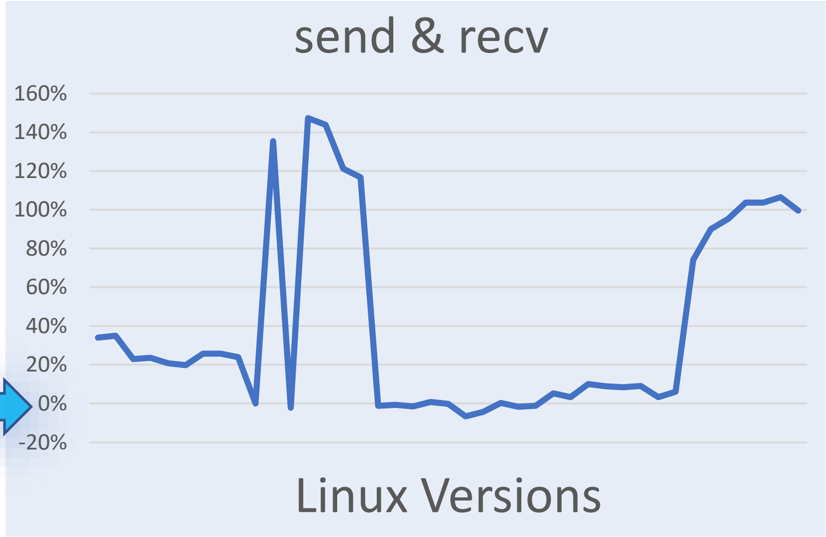
Linux core functions' performance evolution



Linux core functions' performance evolution



- ➔ **67%** encounter slowdown **>50%**
- ➔ **42%** encounter slowdown **>100%**



- ➔ **92%** slower than v4.0 (baseline)
- ➔ **75%** slower than v3.0 (earliest)

Outline

Q: How has performance of Linux's core functions been evolving?

- Linux's core functions' performance displays high variance

Q: What causes performance fluctuations?

Q: What can we do about the root causes?

Outline

Q: How has performance of Linux's core functions been evolving?

- Linux's core functions' performance displays high variance

Q: What causes performance fluctuations?

Q: What can we do about the root causes?

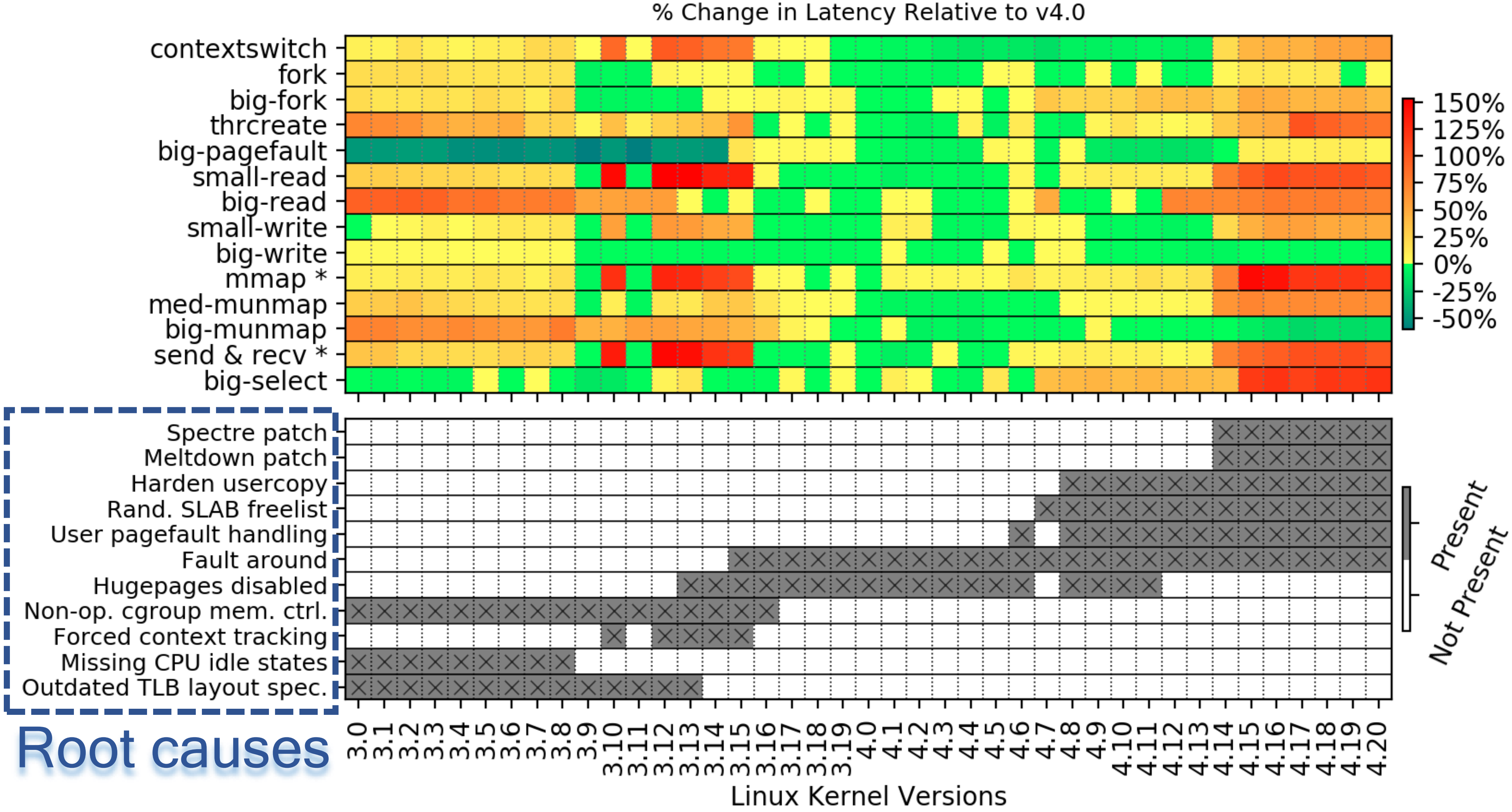
Diagnosing performance root causes

➔ **Step 1** Investigate most significant performance change

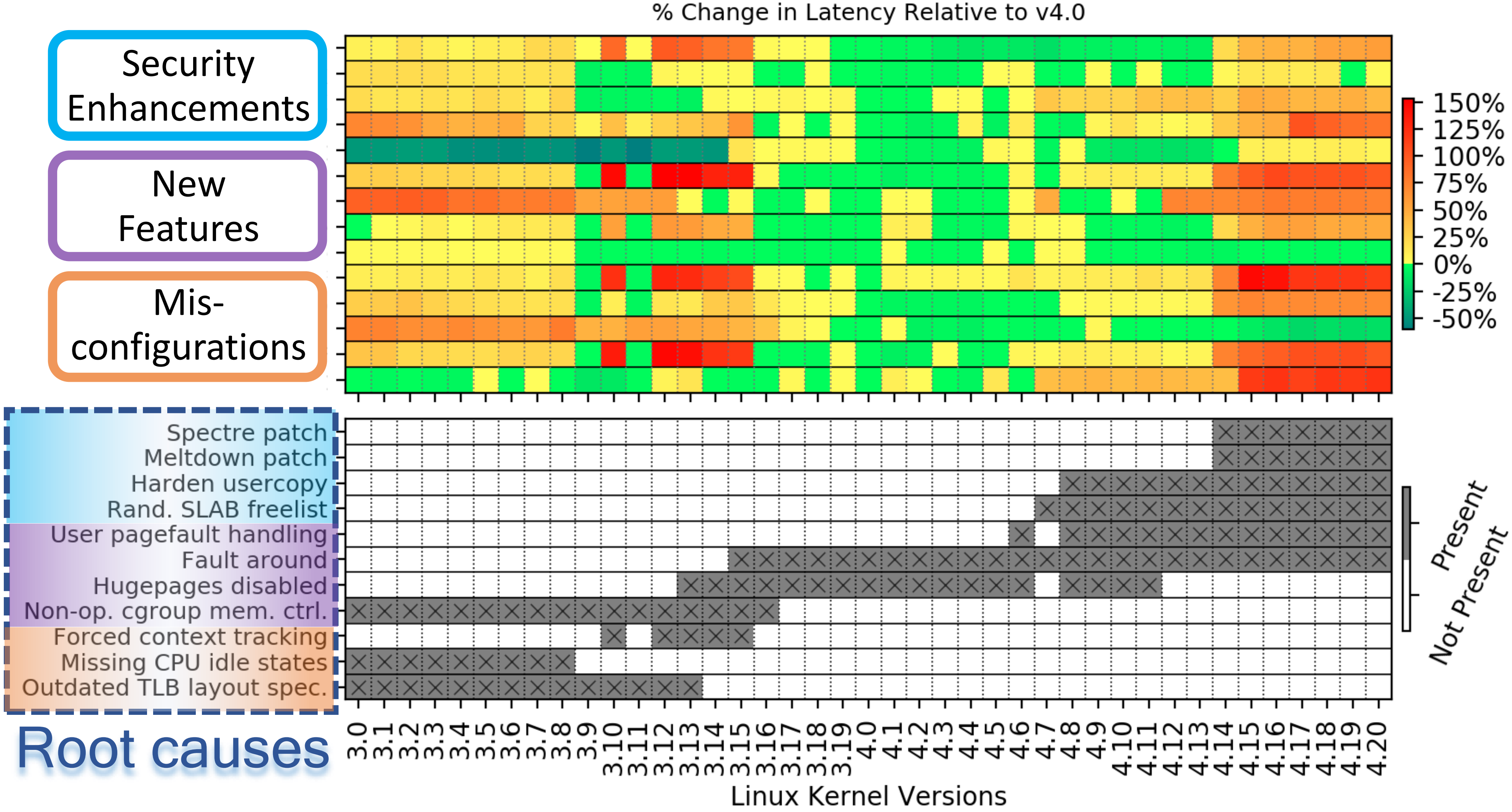
➔ **Step 2** Disable the diagnosed root cause

Repeat until no more than 10% performance change

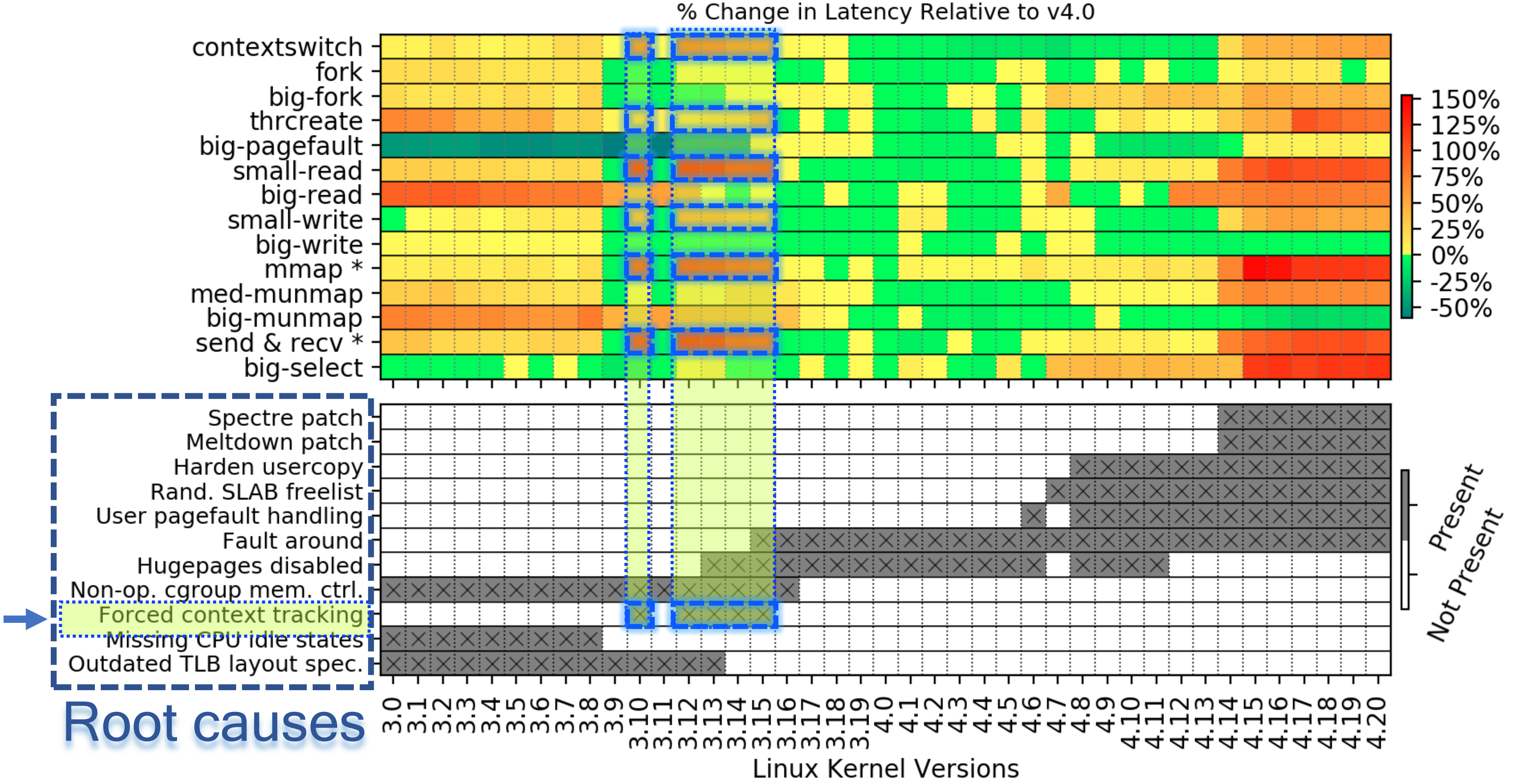
What causes performance fluctuations?



What causes performance fluctuations?



What causes performance fluctuations?



Outline

Q: How has performance of Linux's core functions been evolving?

- Linux's core function performance displays high variance

Q: What causes performance fluctuations?

- Most performance variations explained by 11 root causes
- Root causes fall under *security*, *functionality*, and *misconfiguration*

Q: What can we do about the root causes?

Outline

Q: How has performance of Linux's core functions been evolving?

- Linux's core function performance displays high variance

Q: What causes performance fluctuations?

- Most performance variations explained by 11 root causes
- Root causes fall under *security*, *functionality*, and mis-*configuration*

Q: What can we do about the root causes?

Many opportunities to improve performance

	Root Causes	*Optimize*	*Configure*
Security Enhancements	Meltdown patch	○	
	Spectre patch	●	
	Rand. SLAB freelist		
	Harden usercopy	●	
New Features	Fault around		■
	Hugepages disabled		■
	Cgroup mem. controller	○	
	User pagefault handling		
Misconfigs	Forced context tracking		□
	Missing CPU idle states		□
	Outdated TLB Spec.		□

○ Optimized by Linux developers

● We found further optimization

■ We found better configuration

□ Misconfigs eventually fixed by Linux/Ubuntu developers

Many opportunities to improve performance

	Root Causes	*Optimize*	*Configure*
Security Enhancements	Meltdown patch	○	
	Spectre patch	●	
	Rand. SLAB freelist		
	Harden usercopy	●	
New Features	Fault around		■
	Hugepages disabled		■
	Cgroup mem. controller	○	
	User pagefault handling		
Misconfigs	Forced context tracking		□
	Missing CPU idle states		□
	Outdated TLB Spec.		□

○ Optimized by Linux developers

● We found further optimization

■ We found better configuration

□ Misconfigs eventually fixed by Linux/Ubuntu developers

Many opportunities to improve performance

	Root Causes	*Optimize*	*Configure*
Security Enhancements	Meltdown patch	○	
	Spectre patch	●	
	Rand. SLAB freelist		
	Harden usercopy	●	
New Features	Fault around		■
	Hugepages disabled		■
	Cgroup mem. controller	○	
	User pagefault handling		
Misconfigs	Forced context tracking		□
	Missing CPU idle states		□
	Outdated TLB Spec.		□

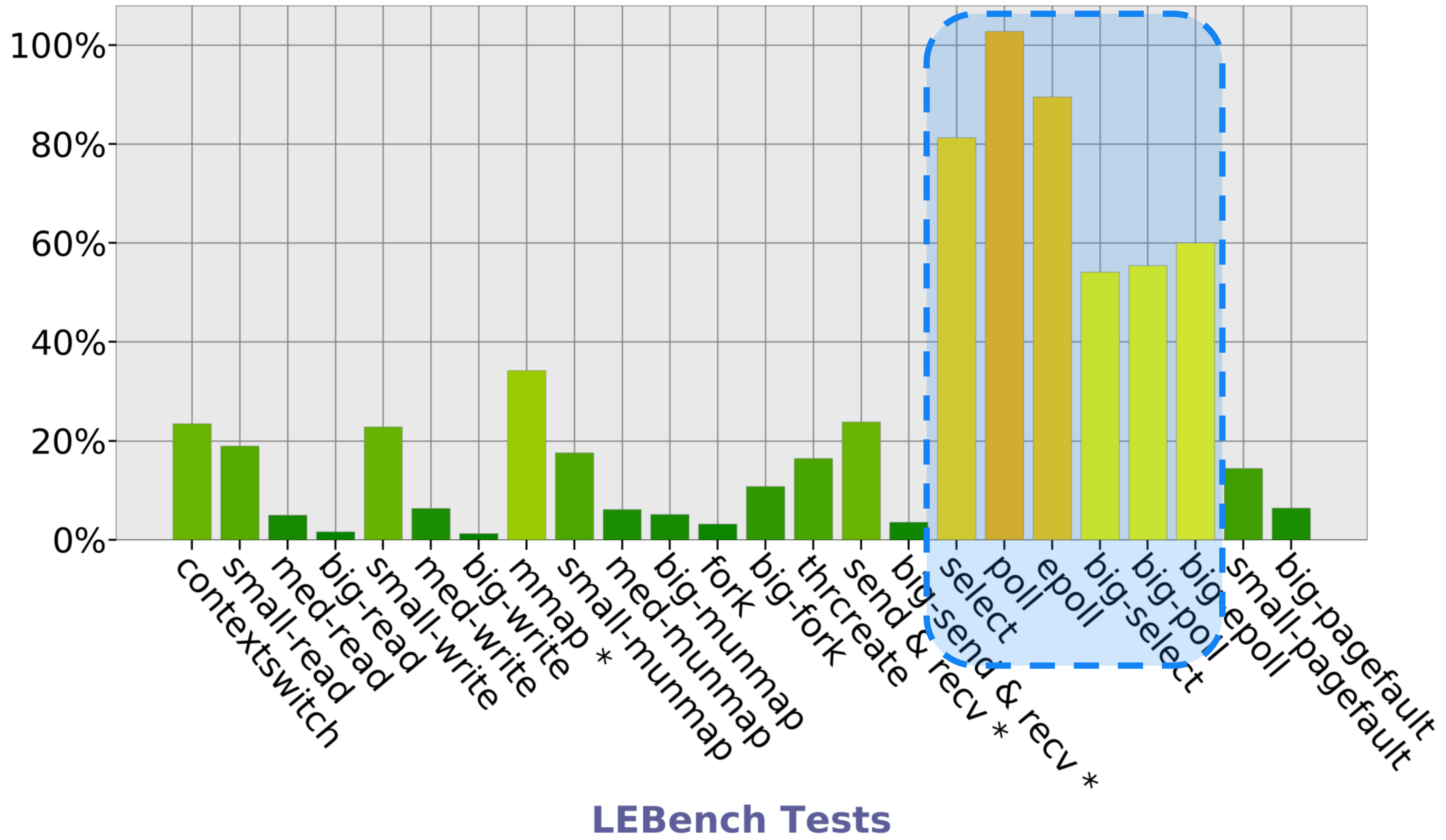
○ Optimized by Linux developers

● We found further optimization

■ We found better configuration

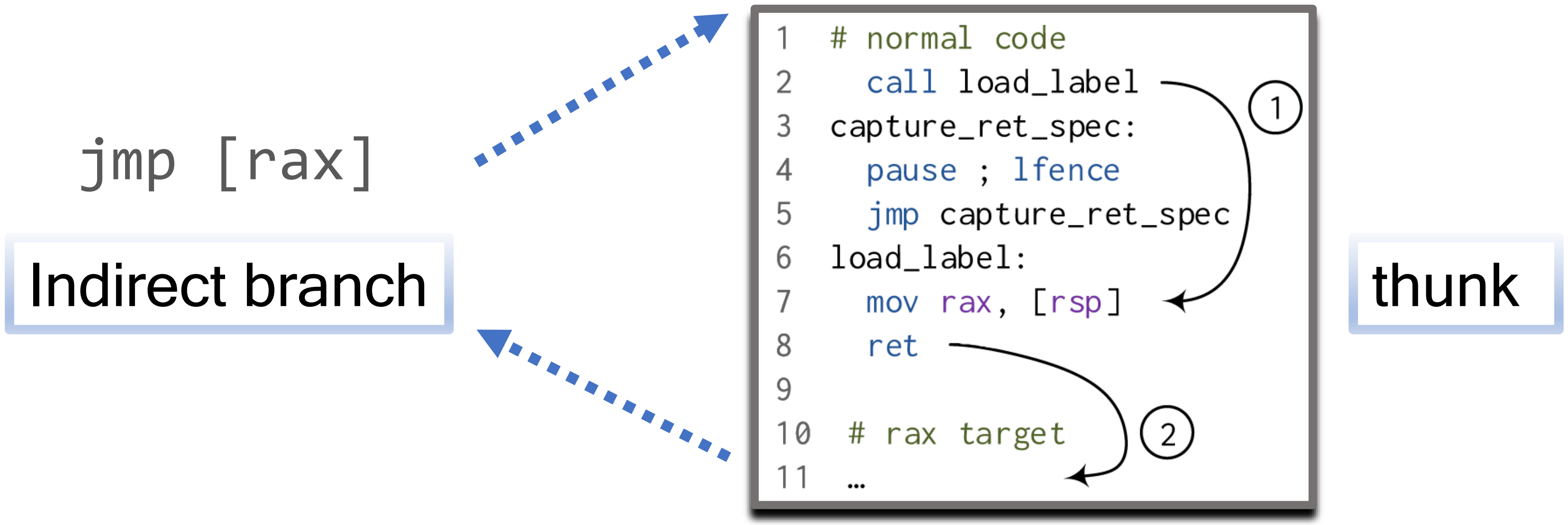
□ Misconfigs eventually fixed by Linux/Ubuntu developers

Case study I: The Spectre patch’s overhead



Case study I: The Spectre patch - Retpoline

- **Spectre V2** exploits indirect branches to leak privileged data
  - Tricks branch predictor into speculatively execute arbitrary address
- Linux mitigates Spectre V2 with gcc patch **Retpoline**
- Retpoline replaces indirect branches with “thunk” instructions



Case study I: The Spectre patch's overhead

- Cost ~ 30-35 cycles per original indirect branch
- Heavily affects `select/poll/epoll` whose polling logic executes indirect branches

fs/select.c

```
int do_select(...) {
  for (;;) {
      ...
      mask = (*f_op->poll)(f.file, wait);
  }
  ...
}
```

net/socket.c

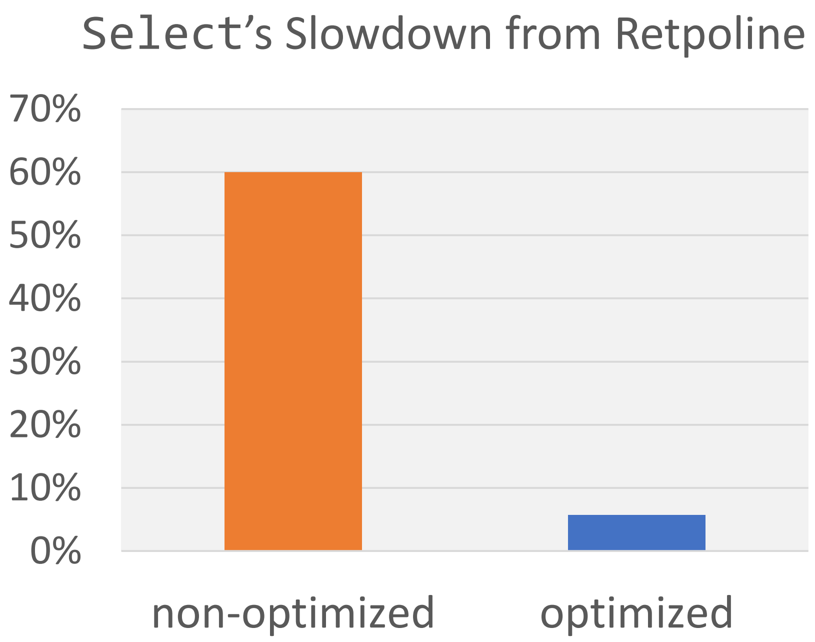
```
const struct file_operations
                  socket_file_ops = {
 .poll =  sock_poll,
 ...
};
```

- 95% of `select`'s slowdown caused by 3 branches in tight loops

Case Study I: Removing Retpoline's overhead

- We design a simple patch: replace the 3 indirect branches with direct branches, which are not vulnerable

```
  for (;;) {
    ...
-   mask = (*f_op->poll)(f.file, wait);
+   if ((*f_op->poll) == sock_poll)
+     mask = sock_poll(f.file, wait);
+   else if ((*f_op->poll) == pipe_poll)
+     mask = pipe_poll(f.file, wait);
+   else if ((*f_op->poll) == timerfd_poll)
+     mask = timerfd_poll(f.file, wait);
+   else
+     mask = (*f_op->poll)(f.file, wait);
    ...
  }
```



Many opportunities to improve performance

Root Causes		***Optimize***	***Configure***
Security Enhancements	Meltdown patch	○	
	Spectre patch	●	
	Rand. SLAB freelist		
	Harden usercopy	●	
New Features	Fault around		■
	Hugepages disabled		■
	Cgroup mem. controller	○	
	User pagefault handling		
Misconfigs	Forced context tracking		□
	Missing CPU idle states		□
	TLB layout Specification		□

○ Optimized by Linux developers

● We found further optimization

■ We found better configuration

□ Misconfigs eventually fixed by Linux/Ubuntu developers

Many opportunities to improve performance

Root Causes		*Optimize*	*Configure*
Security Enhancements	Meltdown patch	○	
	Spectre patch	●	
	Rand. SLAB freelist		
	Harden usercopy	●	
New Features	Fault around		■
	Hugepages disabled		■
	Cgroup mem. controller	○	
	User pagefault handling		
Misconfigs	Forced context tracking		□
	Missing CPU idle states		□
	TLB layout Specification		□

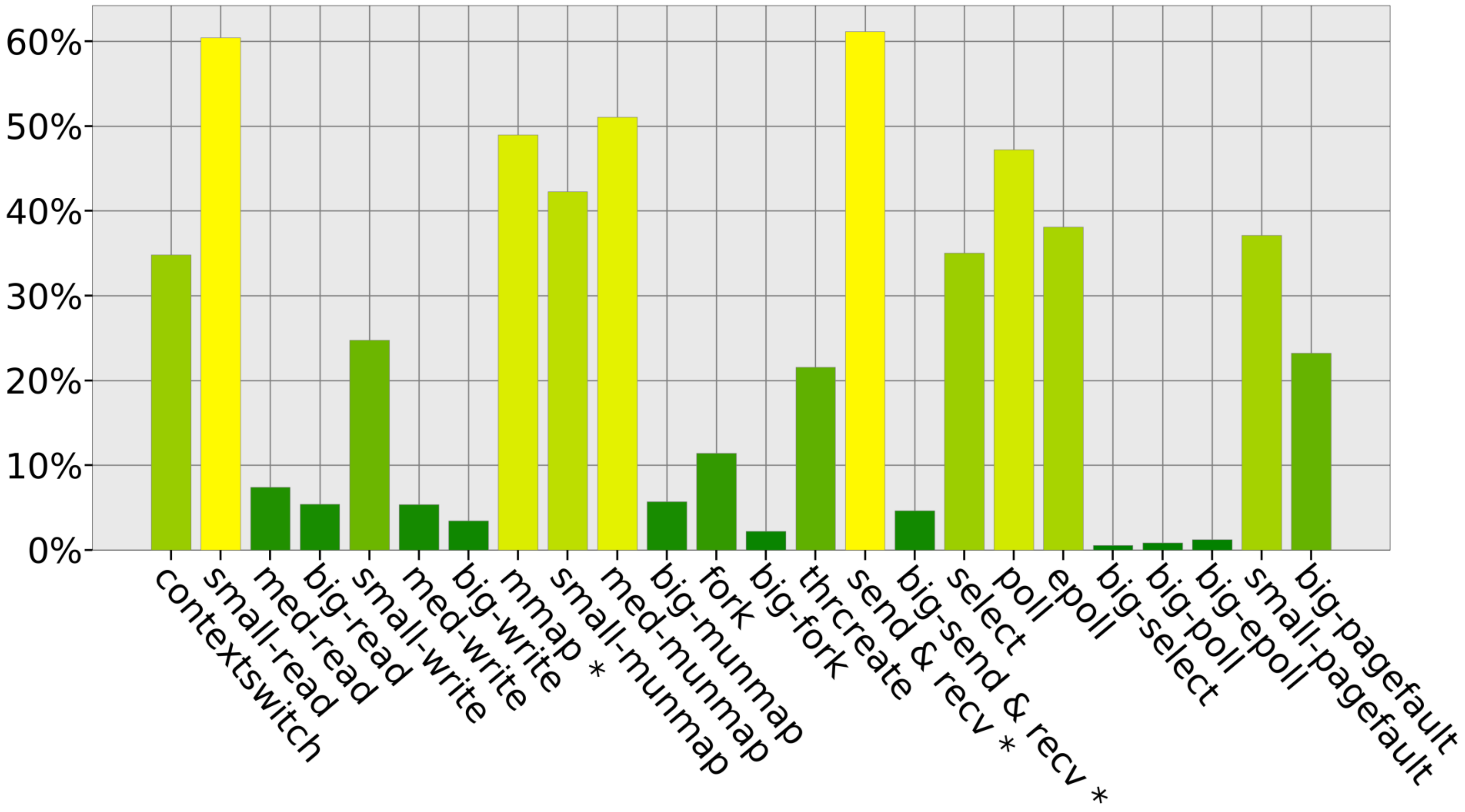
○ Optimized by Linux developers

● We found further optimization

■ We found better configuration

□ Misconfigs eventually fixed by Linux/Ubuntu developers

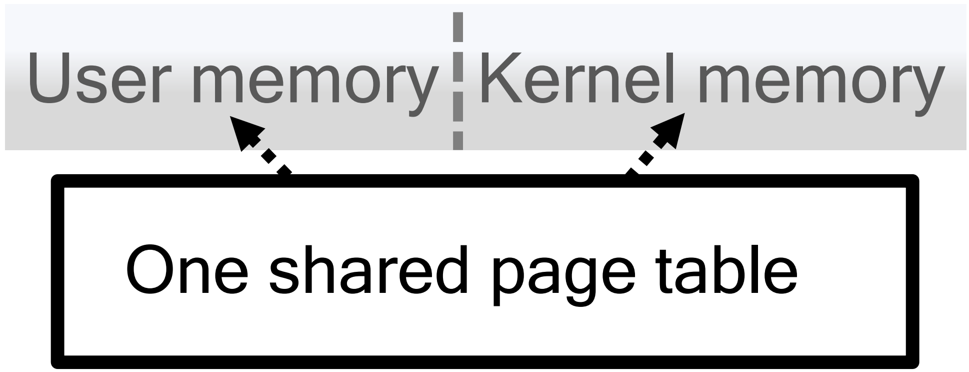
Case study II: The Meltdown patch's overhead



LEBench Tests

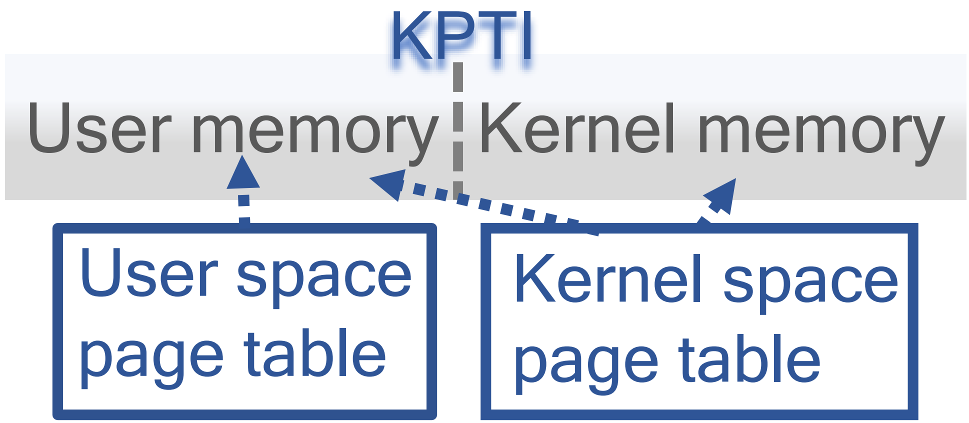
Case study II: The Meltdown patch - KPTI

- The **Meltdown** exploit could leak kernel memory to userspace
  - Exploits data left in cache by unauthorized loads
- Linux's mitigation: **Kernel Page Table Isolation (KPTI)**
- KPTI keeps a separate page table for kernel and user space



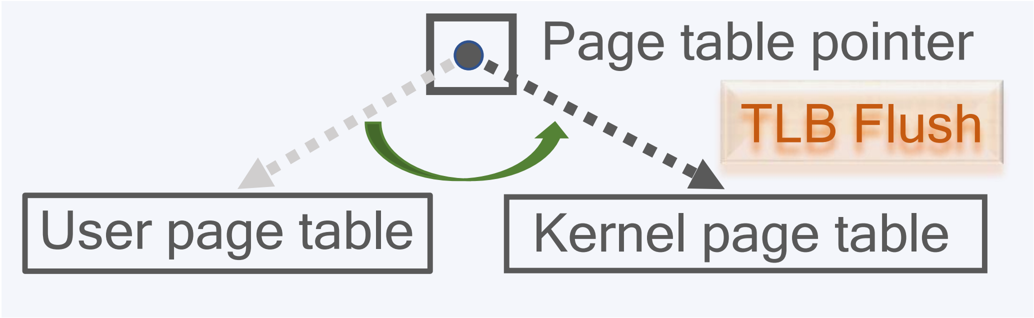
Case study II: The Meltdown patch - KPTI

- The **Meltdown** exploit could leak kernel memory to userspace
  - Exploits data left in cache by unauthorized loads
- Linux's mitigation: **Kernel Page Table Isolation (KPTI)**
- KPTI keeps a separate page table for kernel and user space

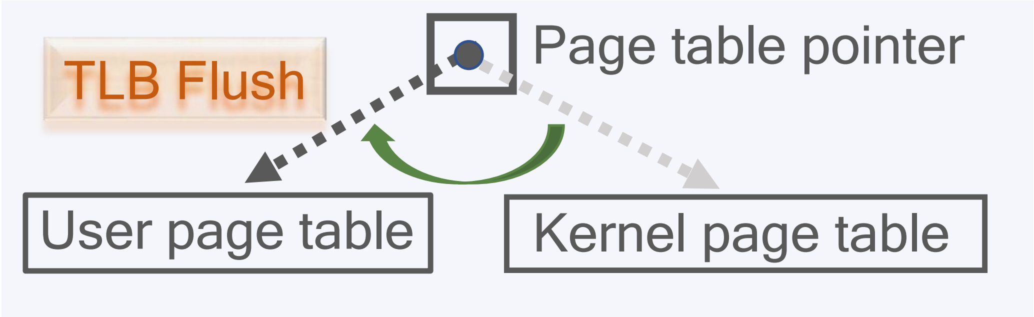


Case study II: The Meltdown patch's overhead

Entering the kernel:



Leaving the kernel:



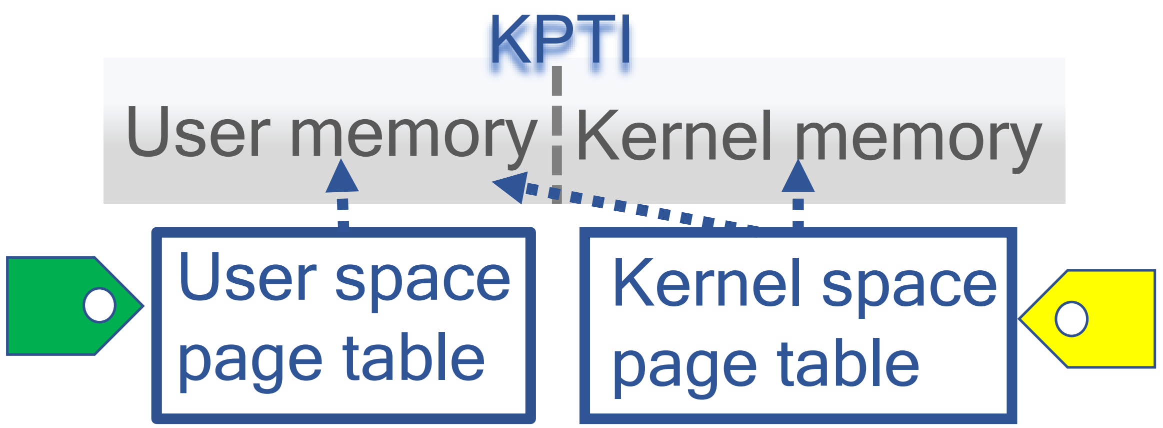
A round-trip to the kernel incurs:

- **2 page table pointer swaps**
  constant cost: ~400 cycles
- **2 TLB flushes**
  ~700-6000 cycles (read tests)

Case study II: Optimizing the Meltdown patch

Linux dev optimized using h/w feature - Process Context IDentifier (PCID):

- Tag kernel/user entries with diff PCIDs
- Allow both entries to coexist in the TLB



A round-trip to the kernel incurs:

- **2 page table pointer swaps**
  constant cost: ~400 cycles
- ~~**2 TLB flushes**~~
  ~~~700-6000 cycles (read tests)~~
  ~~~

Many opportunities to improve performance

	Root Causes	*Optimize*	*Configure*
Security Enhancements	Meltdown patch	○	
	Spectre patch	●	
	Rand. SLAB freelist		
	Harden usercopy	●	
New Features	Fault around		■
	Hugepages disabled		■
	Cgroup mem. controller	○	
	User pagefault handling		
Misconfigs	Forced context tracking		□
	Missing CPU idle states		□
	TLB layout Specification		□

○ Optimized by Linux developers

● We found further optimization

■ We found better configuration

□ Misconfigs eventually fixed by Linux/Ubuntu developers

Many opportunities to improve performance

Root Causes		***Optimize***	***Configure***
Security Enhancements	Meltdown patch	○	
	Spectre patch	●	
	Rand. SLAB freelist		
	Harden usercopy	●	
New Features	Fault around		■
	Hugepages disabled		■
	Cgroup mem. controller	○	
	User pagefault handling		
Misconfigs	Forced context tracking		□
	Missing CPU idle states		□
	TLB layout Specification		□

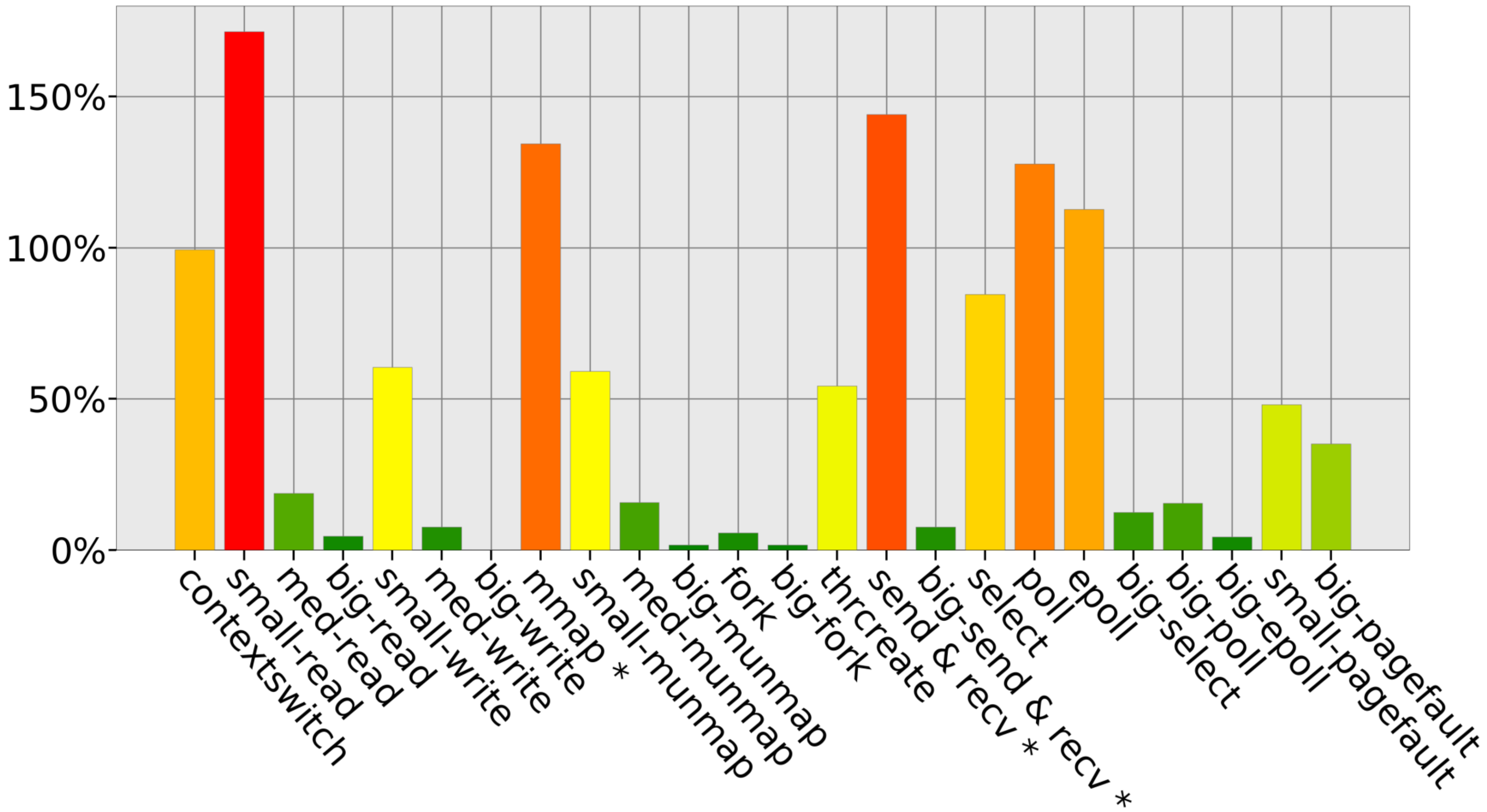
○ Optimized by Linux developers

● We found further optimization

■ We found better configuration

□ Misconfigs eventually fixed by Linux/Ubuntu developers

Case study III: Forced Context Tracking (FCT)



LEBench Tests

Case study III: Forced Context Tracking (FCT)

Reduced Scheduling Clock Ticks (**RSCT**)

- Send fewer/no scheduling interrupts to a core

Depends on

Context Tracking

- Handles tasks done at scheduling interrupts
- Done during kernel entry/exit via system calls etc.

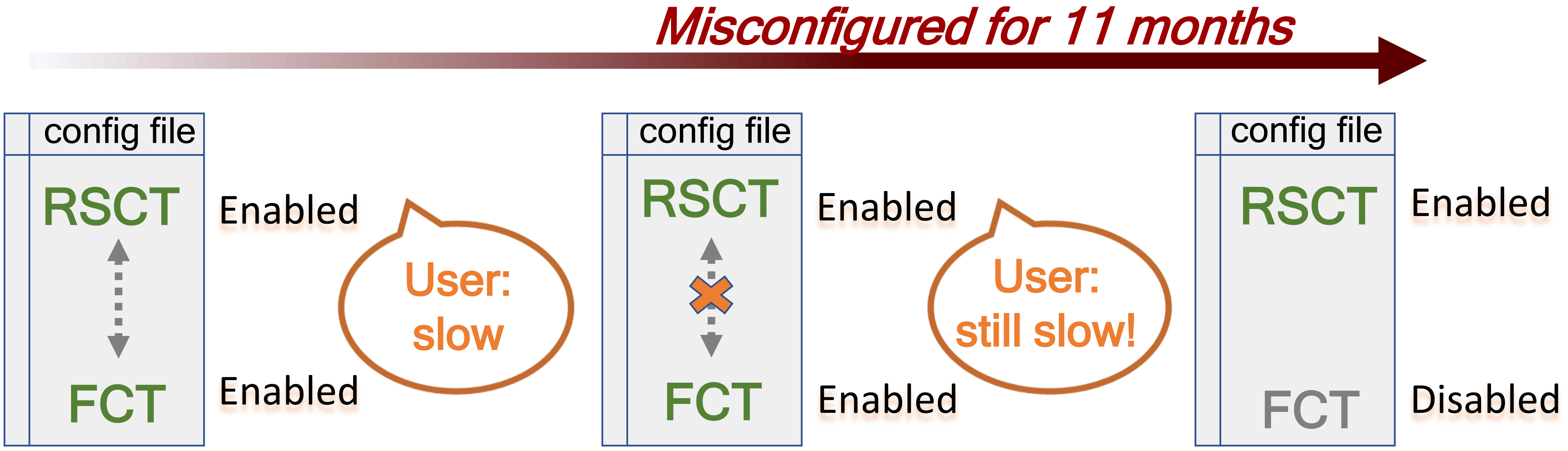
Enables Debugging

Forced Context Tracking (**FCT**)

- Adds 200-300ns for every kernel entry/exit
- Enabled by mistake in release versions

Case study III: Fixing FCT's misconfiguration

RSCT: Reduced Scheduling Clock Ticks **FCT**: Forced Context Tracking



FCT enabled through dependency

Dependency removed, failed to disable FCT

FCT finally disabled

Additional evaluations

- Evaluated Redis, Apache, Nginx macrobenchmarks
  - Experience very similar degrees of slowdown to LEBench
- Reproduced LEBench results on a different machine setup

Related Work

- OS Performance studies on different hardware architectures [Ousterhout'90, Anderson'91, Rosenblum'95]
- OS microbenchmark (lmbench) & macrobenchmark suites (lkp)
- Linux performance regressions [Chen'07]

Our contribution:

- Systematic study of performance evolution of Linux's core functions

Conclusion

- LEBench – a microbenchmark for core Linux functions
- Linux performance displays high variance over time
- 11 root causes explain most of the performance changes
- Much slowdown avoidable by optimizing and re-configuring

Thanks!

https://github.com/LinuxPerfStudy/LEBench