The impact of Meltre and Specdown on microkernel systems

Matthias Lange, Kernkonzept GmbH, FOSDEM 2019

“We need to talk about Meltre and Specdown.”

–Conf call with customer, early 2018

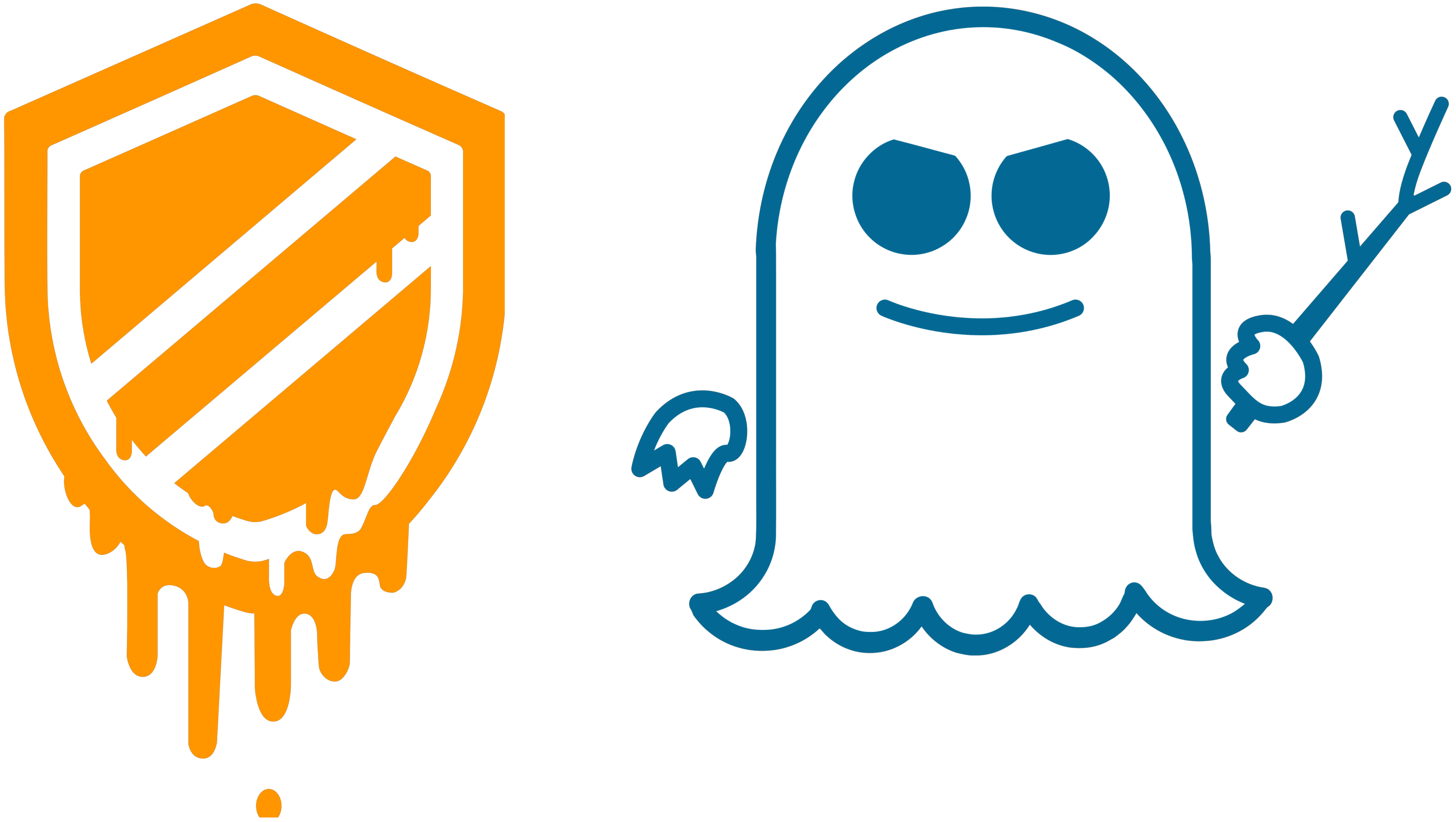
The impact of Meltdown and Spectre on the L4Re microkernel system

Questions

- Where we prepared?
- Did microkernel design principles protect or help us?
- What’s the impact of implemented mitigations?

Questions - Spoiler

- Where we prepared? **No**
- Did microkernel design principles protected or helped us? **A little bit**
- What’s the impact of implemented mitigations? 😥

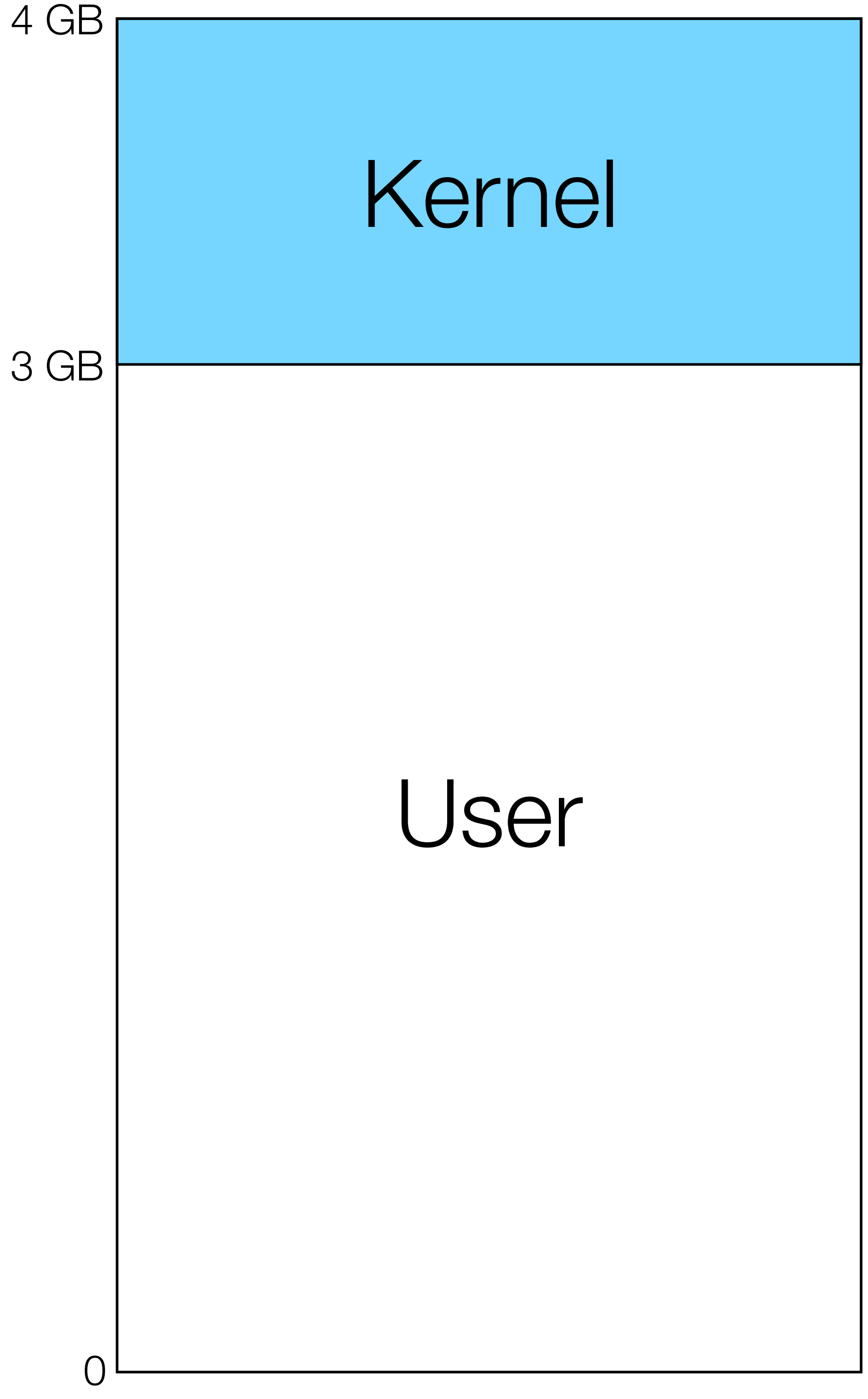


Meltdown & Spectre

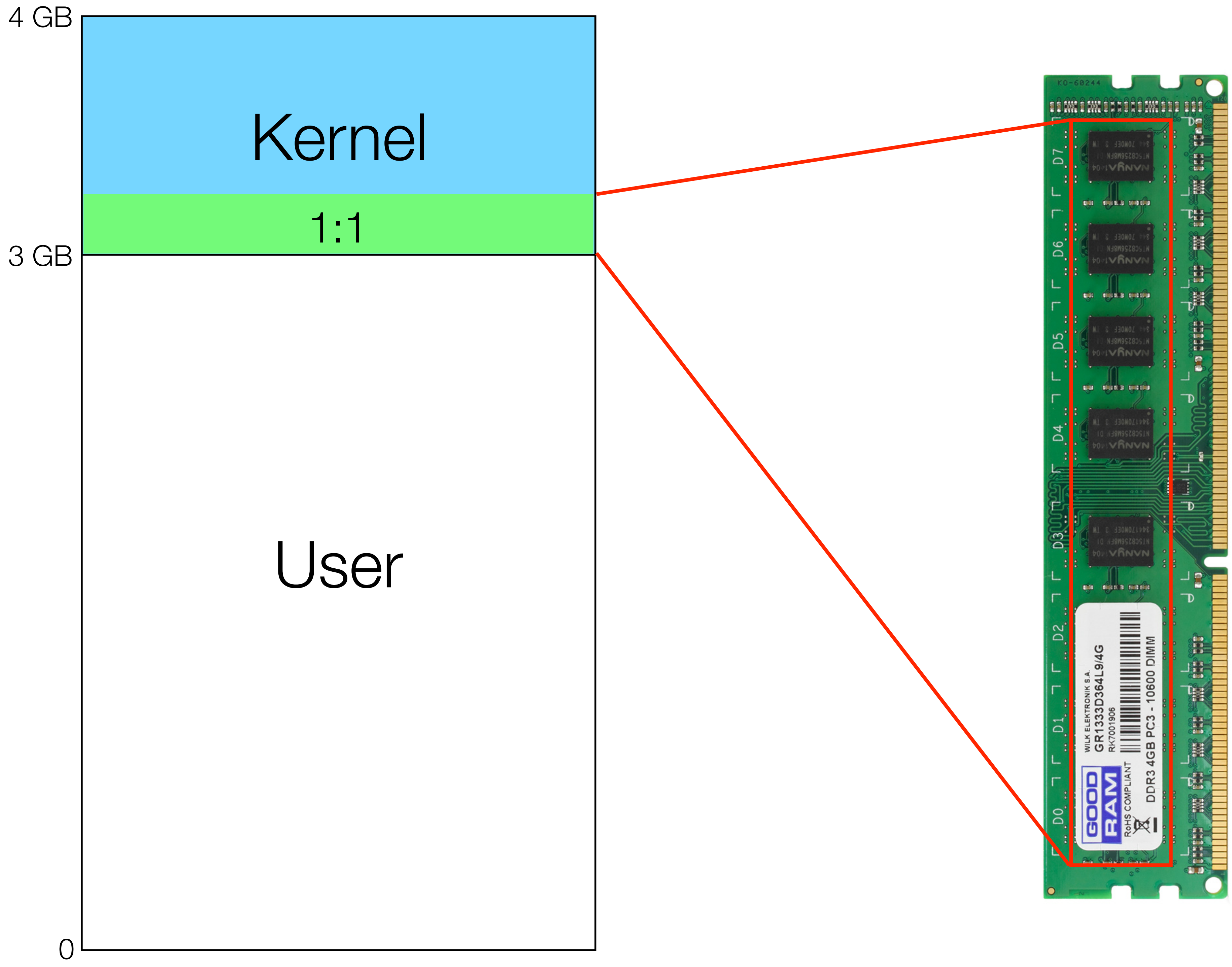
Set of vulnerabilities in modern CPUs

Meltdown

Classic virtual address space layout



Classic virtual address space layout



4 GB

Kernel

1:1

3 GB

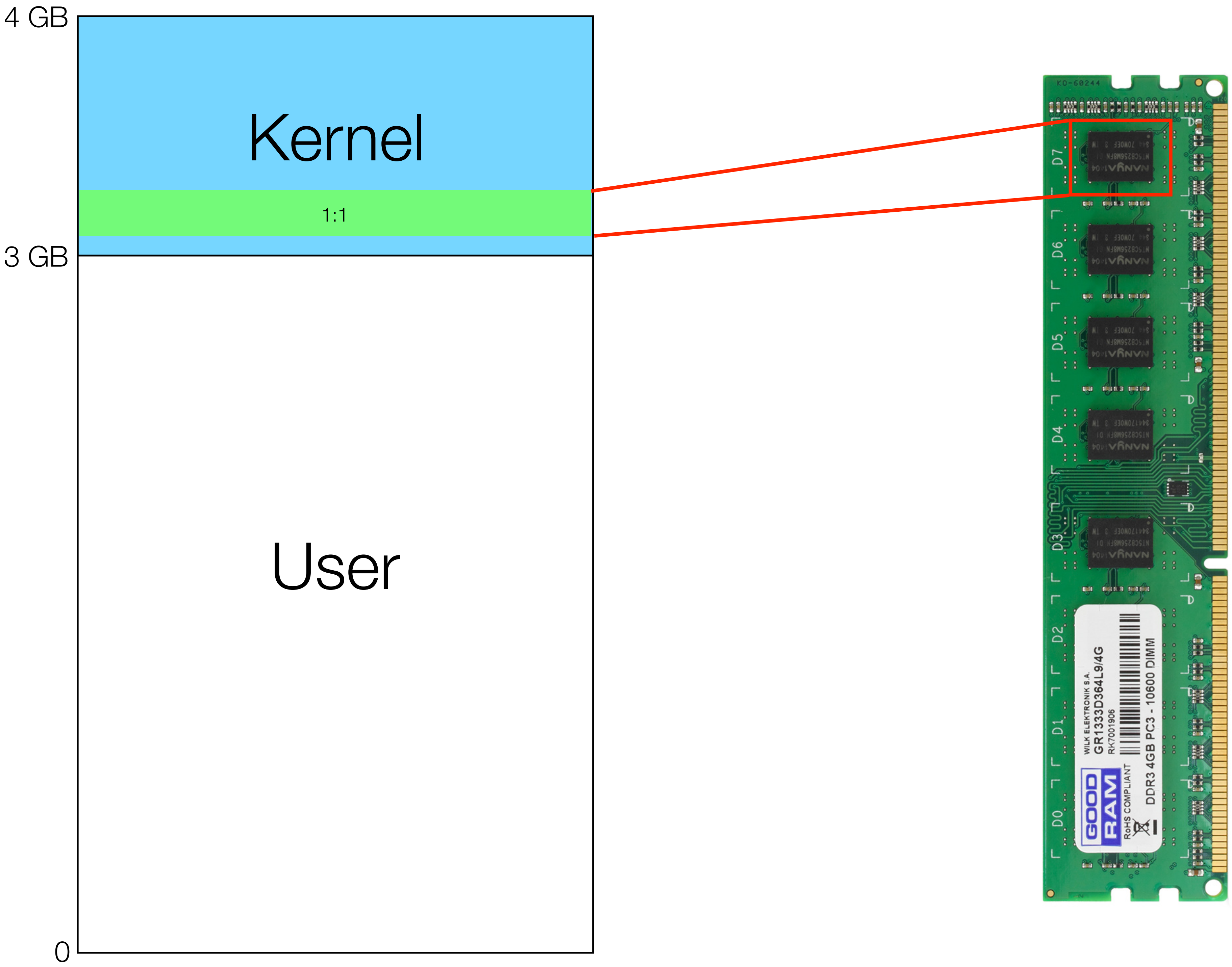
User

0

L4Re’s virtual address space layout

- Fiasco reserves fixed amount of memory for itself
    - Not all physical memory is mapped in the kernel
    - Uses big pages for mapping
    - Mapping may include user memory

L4Re’s virtual address space layout



4 GB

Kernel

1:1

3 GB

User

0

Solution: Kernel address space

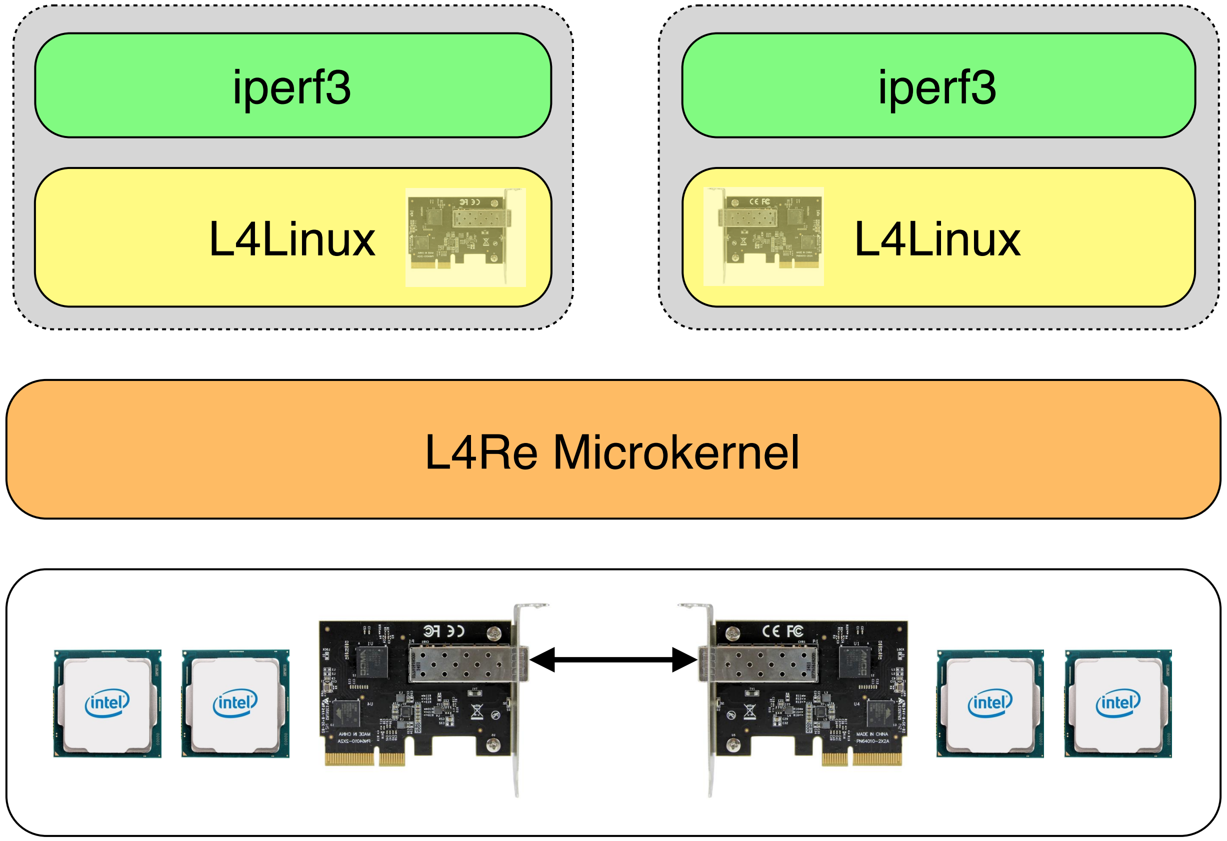
- Move kernel into its own address space
  - Fiasco uses a CPU local address space
- User address space only maps absolutely necessary parts
  - GDT, TSS, entry / exit stack, UTCBs

Benchmarks - PTI

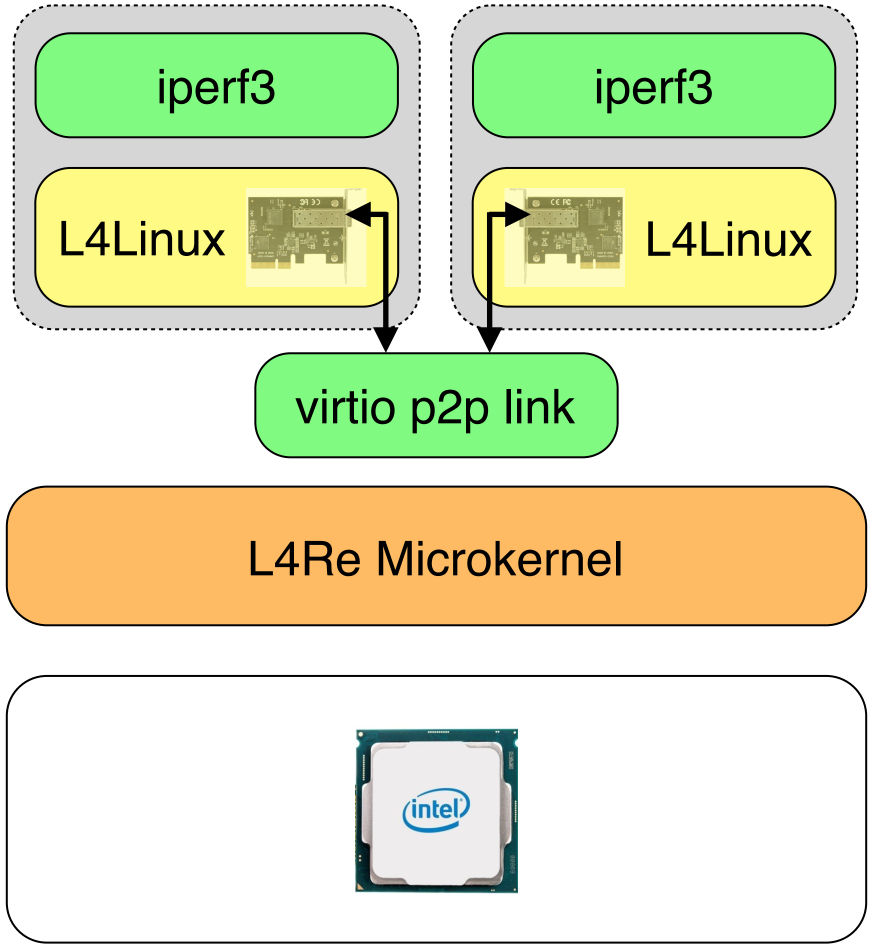
Benchmarks - Meta

- Baseline
  - Fiasco GitHub commit 566cc120, January 1st, 2018
- Head
  - Fiasco GitHub commit 591c8c0b, January 7th, 2019
- Compiler: kernel clang 6, userland gcc 7.3
- Core i7-5700EQ, 2.60GHz
- Contact me if interested in raw data

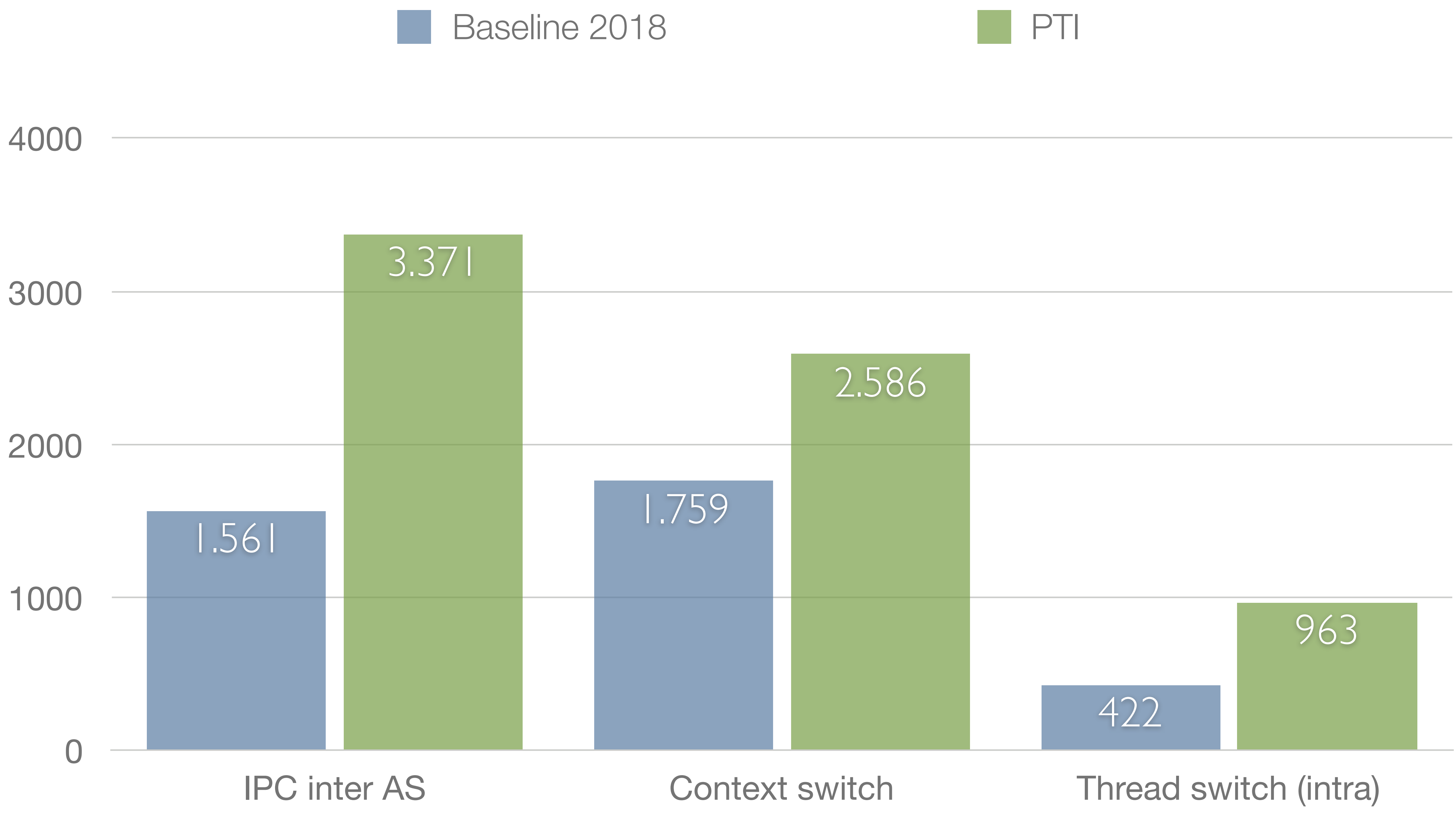
Benchmarks - Scenario 1



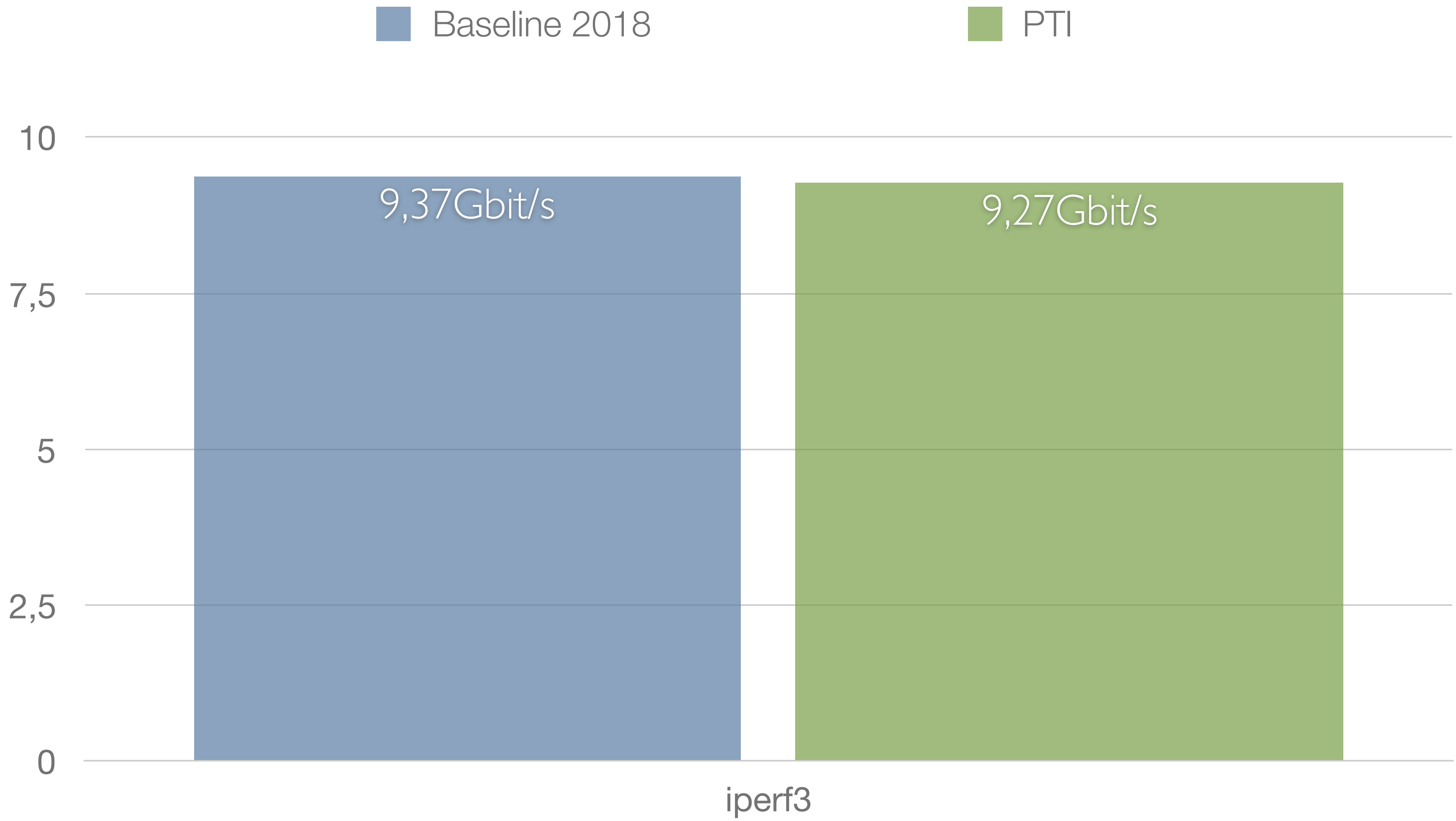
Benchmarks - Scenario 2



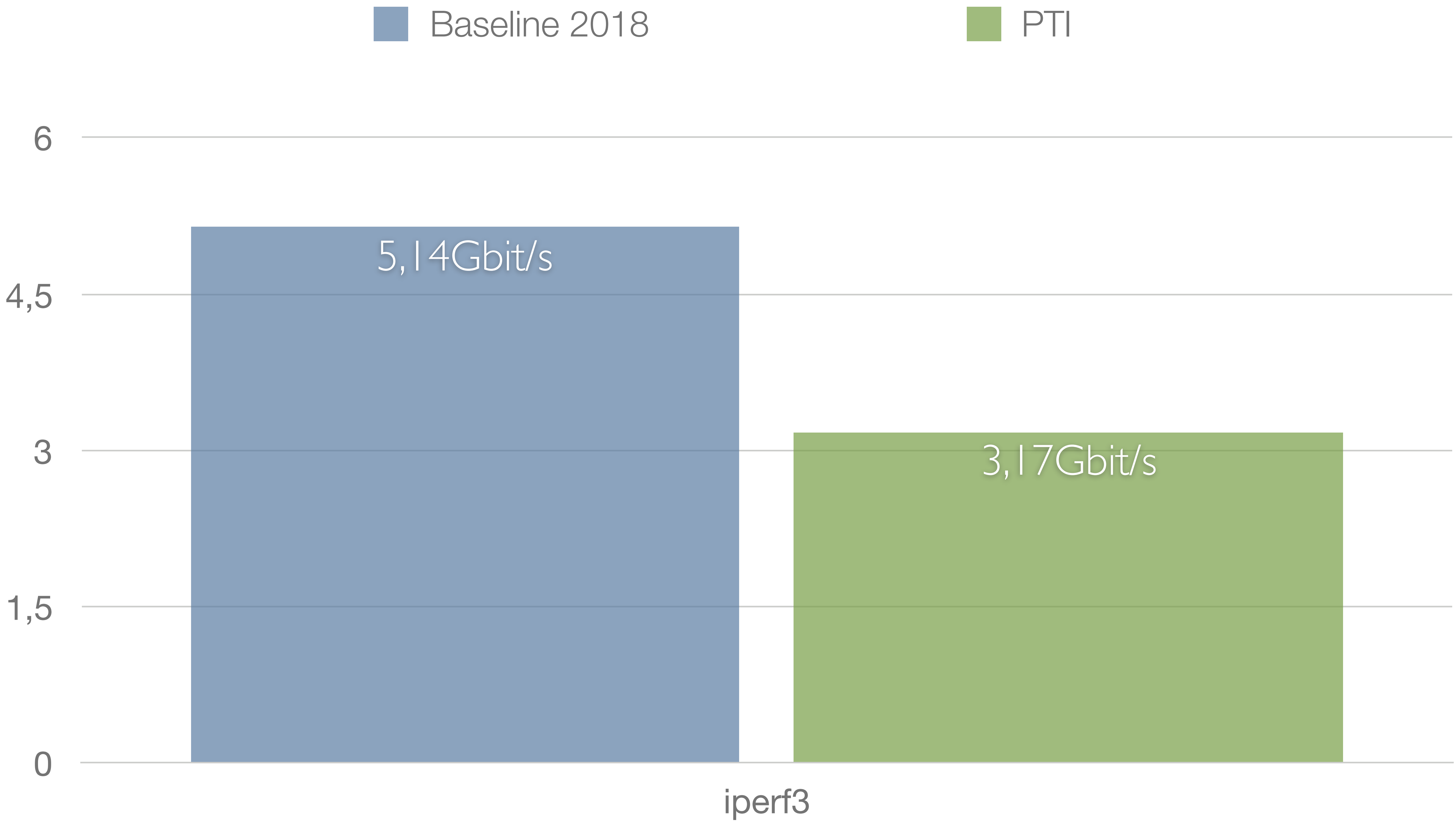
Micro benchmarks - pingpong, PTI



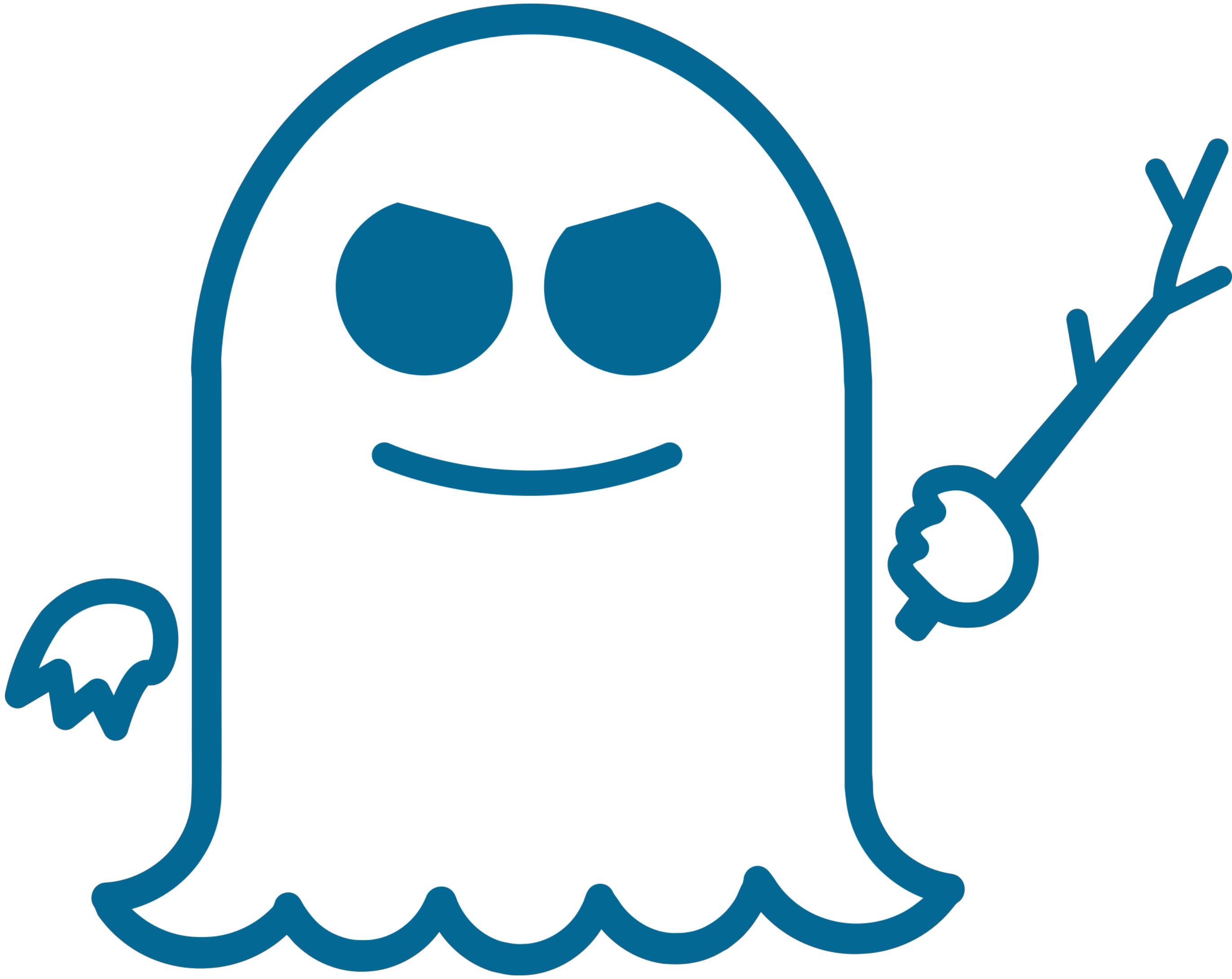
Benchmarks - Scenario 1, PTI



Benchmarks - Scenario 2, PTI

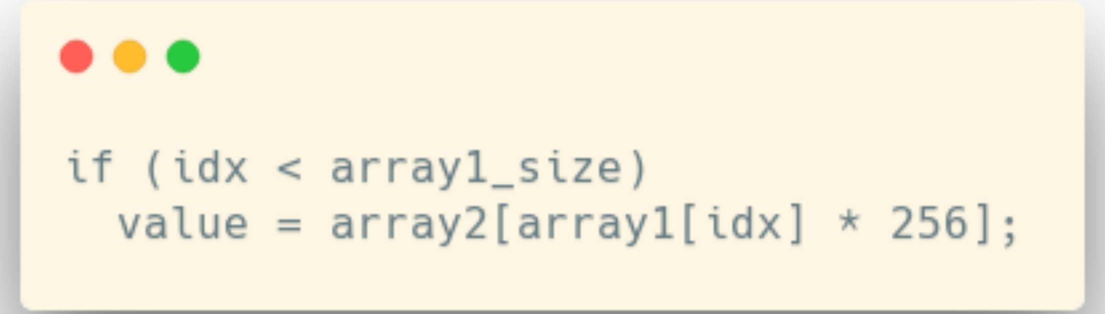


Spectre



Spectre

- Indirect branch prediction speculatively access data causing side effects



```
if (idx < array1_size)
  value = array2[array1[idx] * 256];
```

Spectre NG

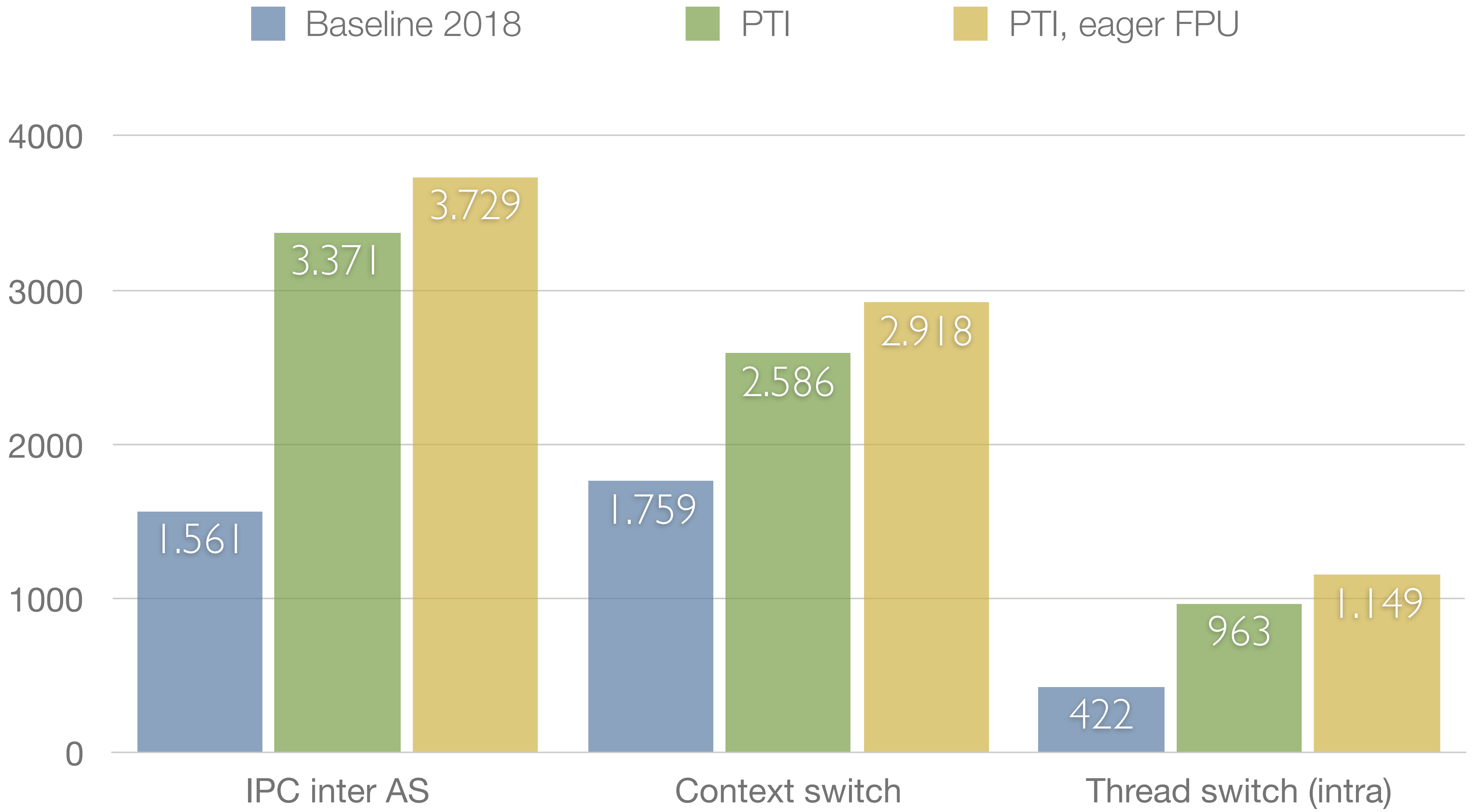
- Speculative access to FPU state while current context is not the owner
- Fiasco uses lazy FPU switching

Spectre NG - Mitigation

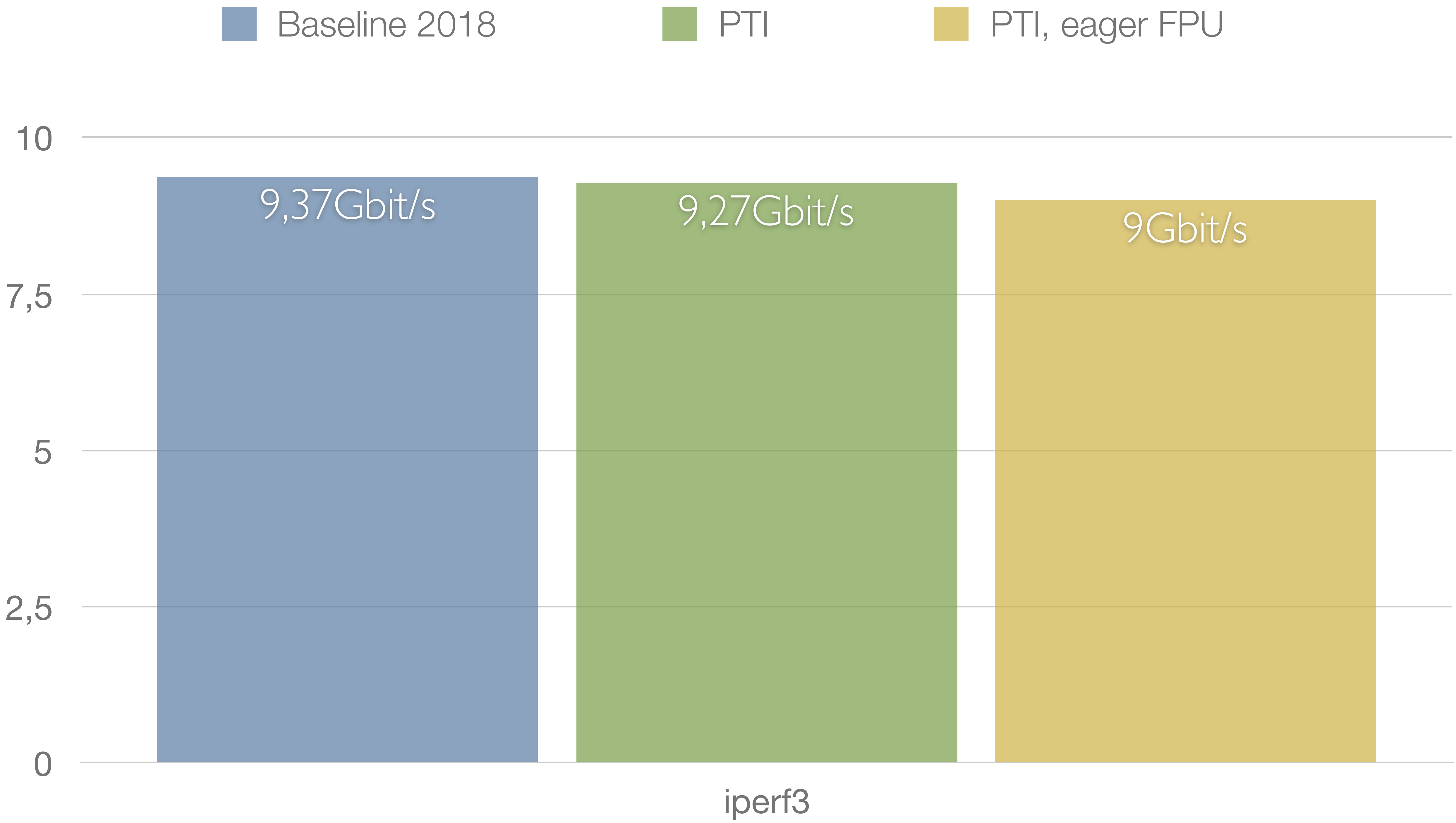
- Fiasco now supports eager switching on x86
- Does this incur any performance loss?

Benchmarks - Eager FPU switching

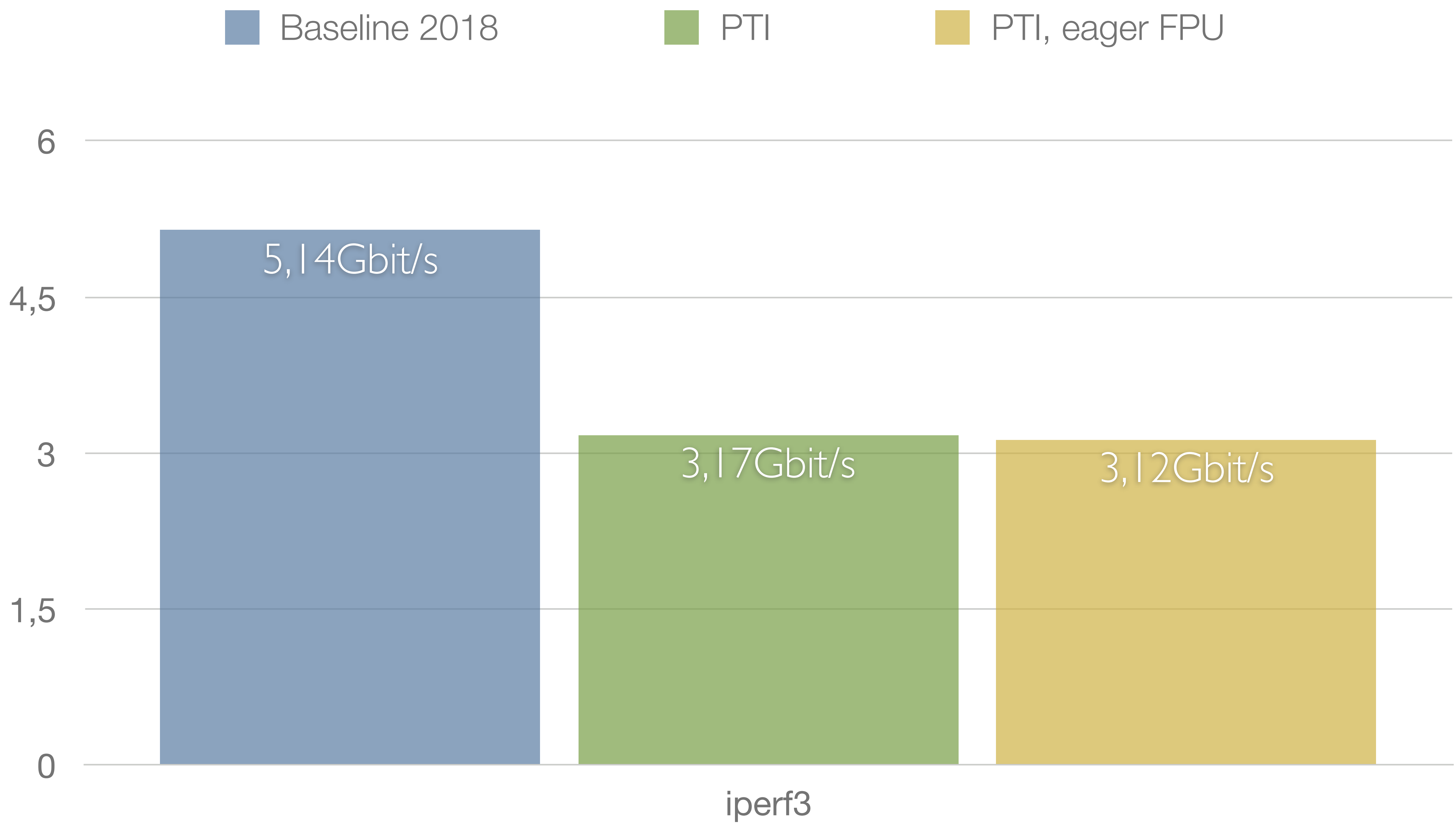
Micro benchmarks - pingpong, PTI, eager FPU



Benchmarks - Scenario 1, PTI, eager FPU



Benchmarks - Scenario 2, PTI, eager FPU



Spectre continued

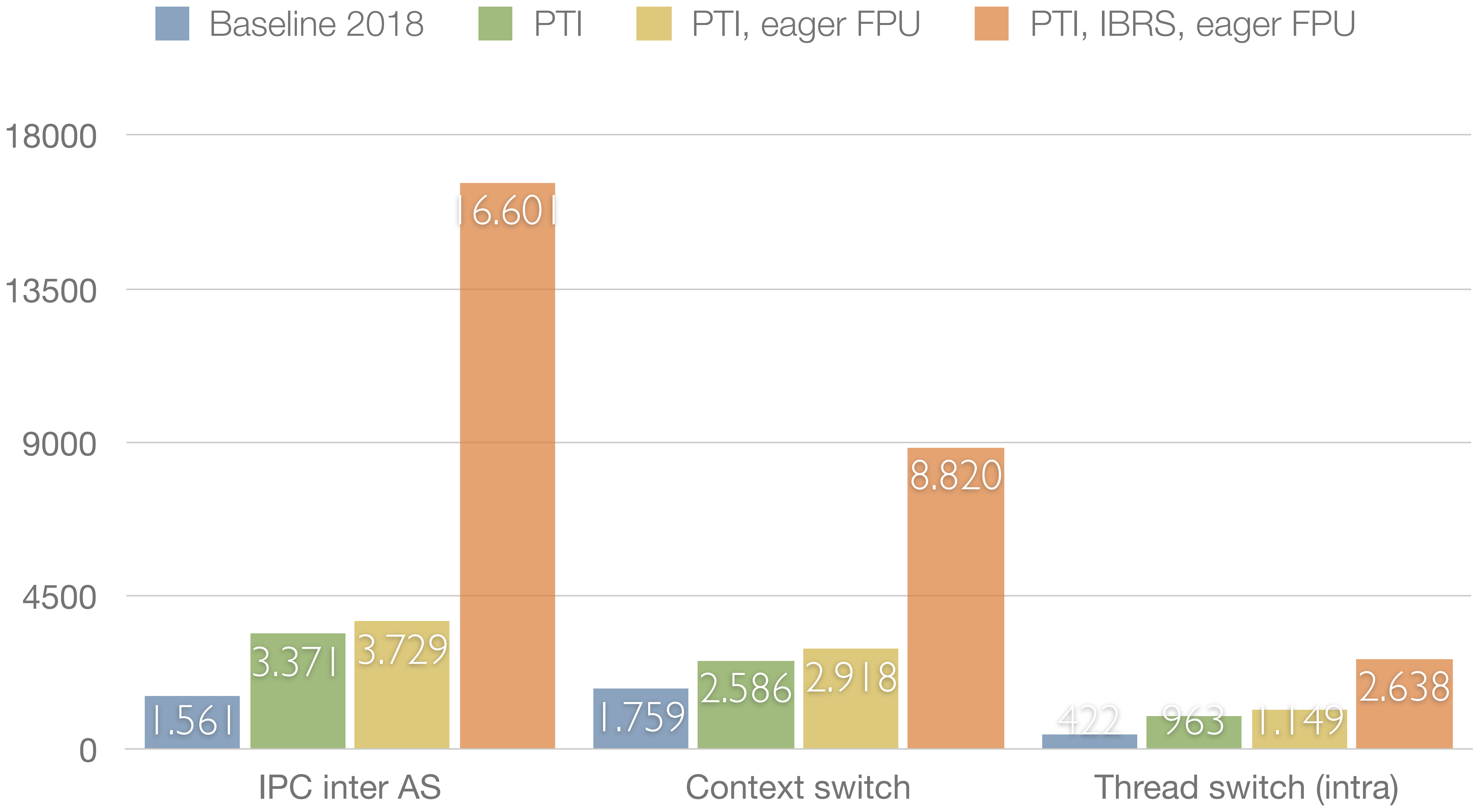
- Most variants do not work across process boundaries
- Usually code execution required

Spectre continued - Mitigations

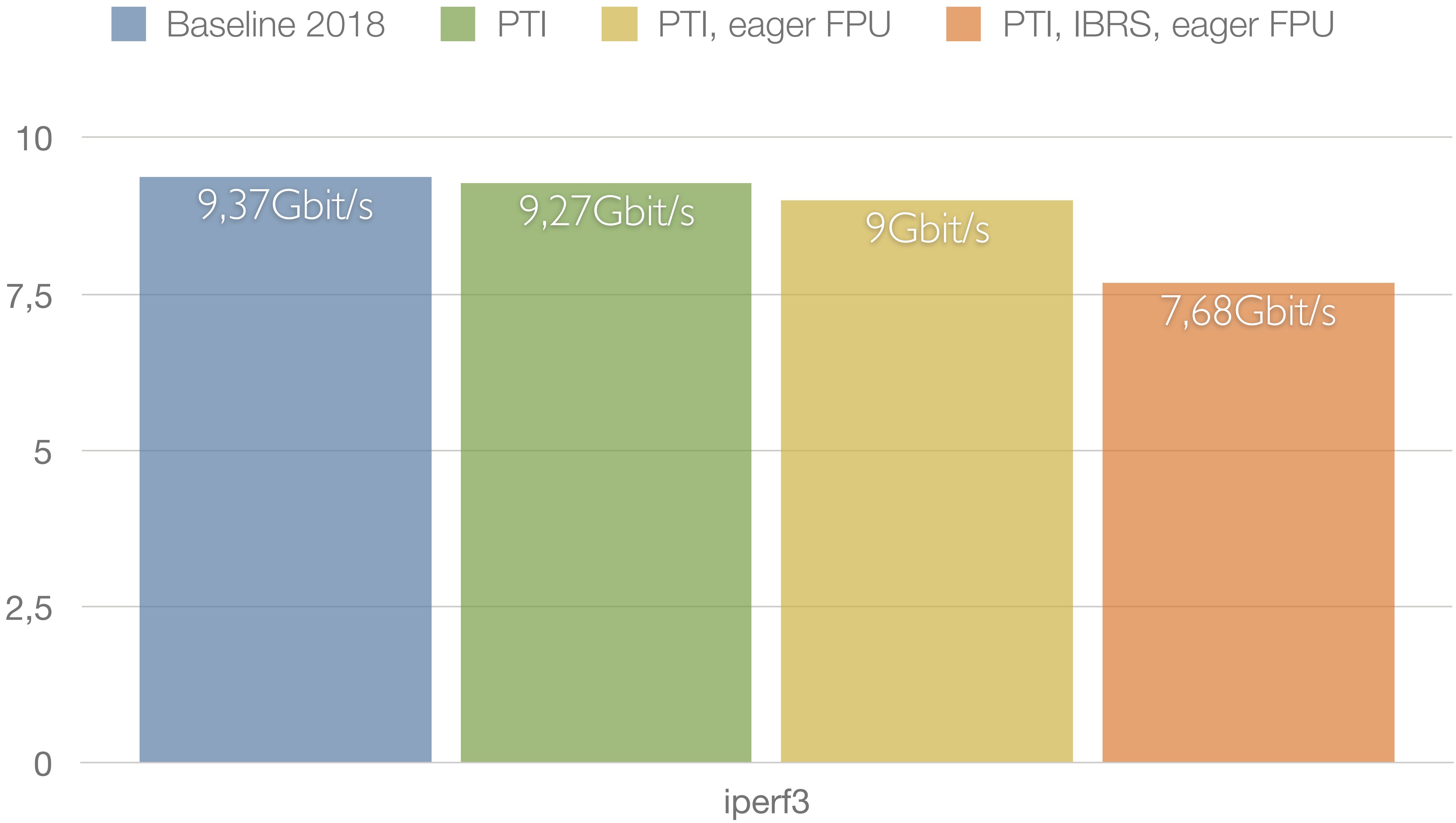
- Fiasco mitigations
  - Indirect branch prediction barrier at kernel entry
  - Full prediction barrier at context switch
  - (microcode loading functionality)

Benchmarks - IBRS 😥

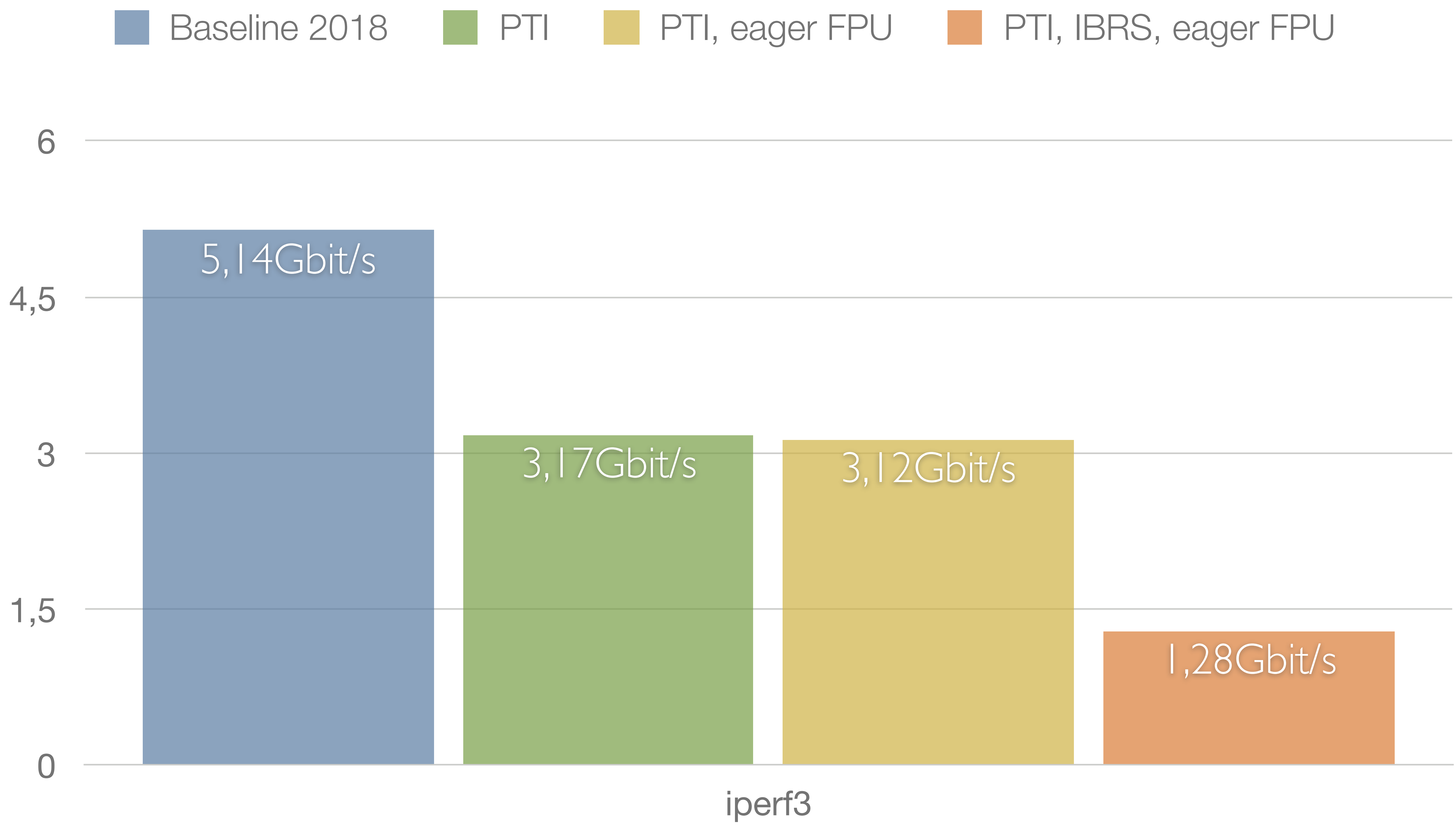
Micro benchmarks - pingpong, IBRS



Benchmarks - Scenario 1, IBRS



Benchmarks - Scenario 2, IBRS



Foreshadow

L1 Terminal Fault

L1 Terminal Fault

- Affects OS / SMM, VT-x and SGX
- SGX not supported in L4Re
  - Don’t care
- SMM needs to protect itself

L1 Terminal Fault - L4Re mitigations

- OS
    - Fiasco is not vulnerable
    - We zero our PTEs
- VT-x is nasty
    - Microcode update
        - New MSR and new instruction for L1D flush
    - Flush L1D on every vmresume

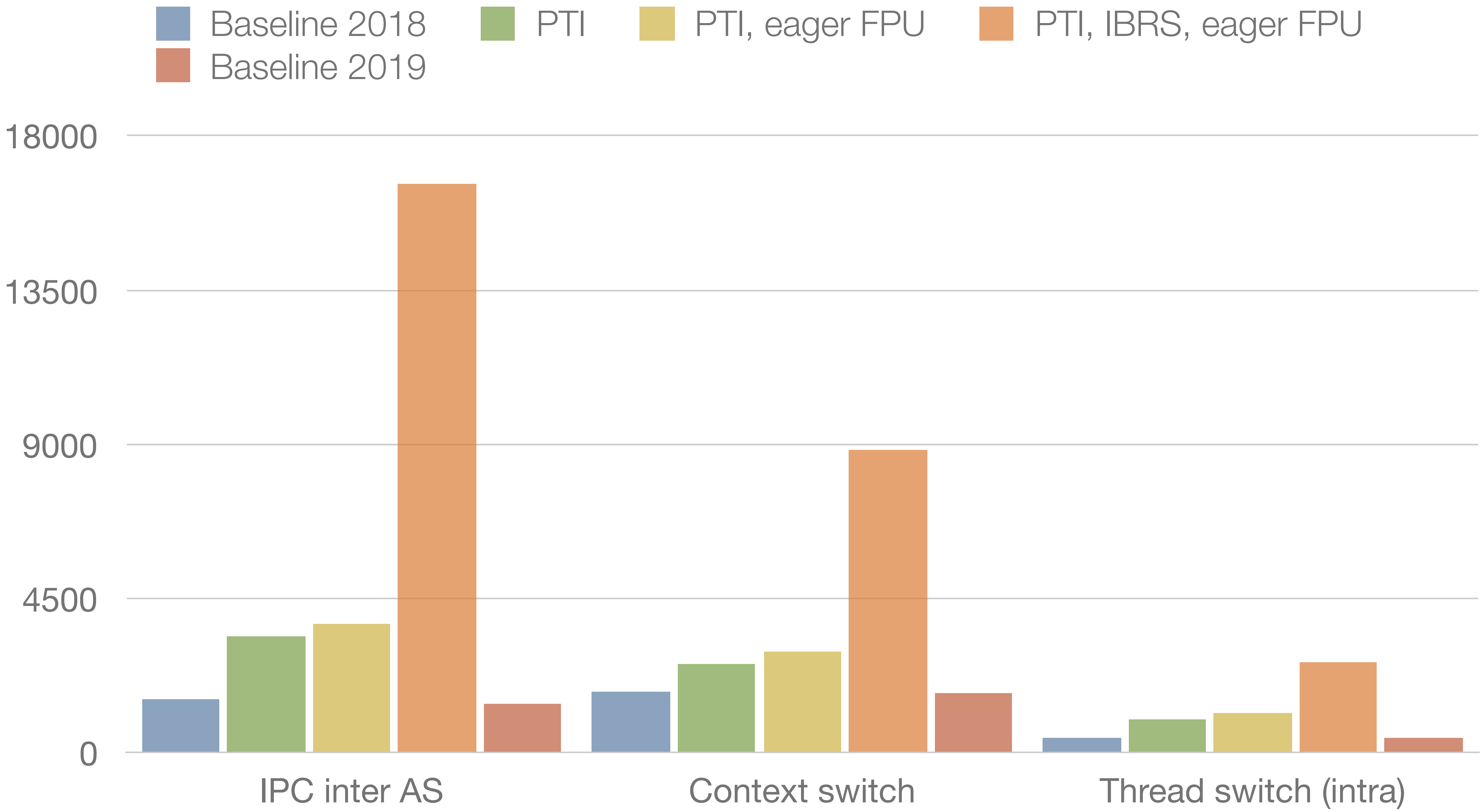
Benchmarks - Sorry, no benchmarks for L1TF.

But there is one more thing …

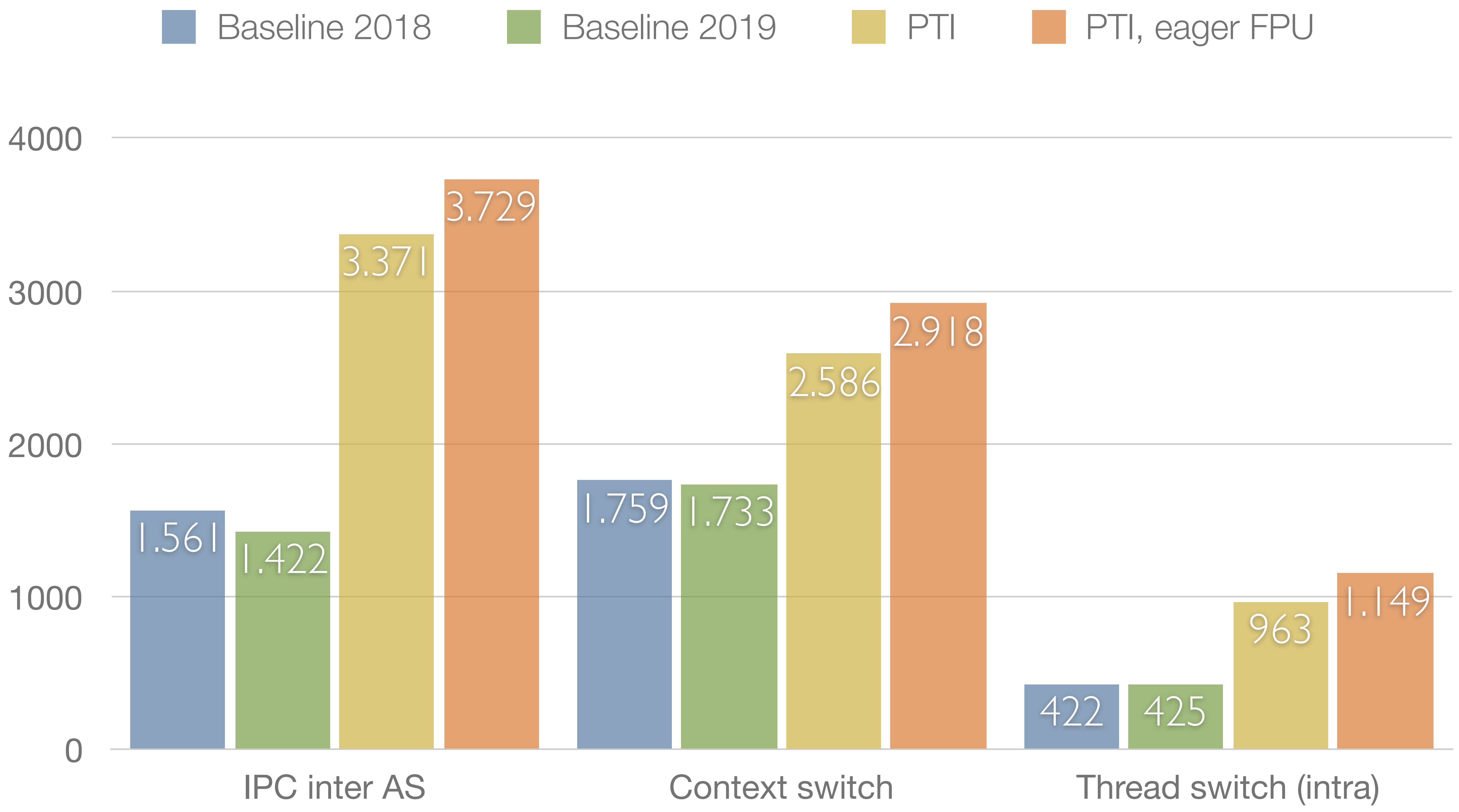
One more thing

- All features / mitigations are configurable
- You can turn off
    - PTI
    - Eager FPU
    - IBRS
- How does this compare to the 2018 baseline?

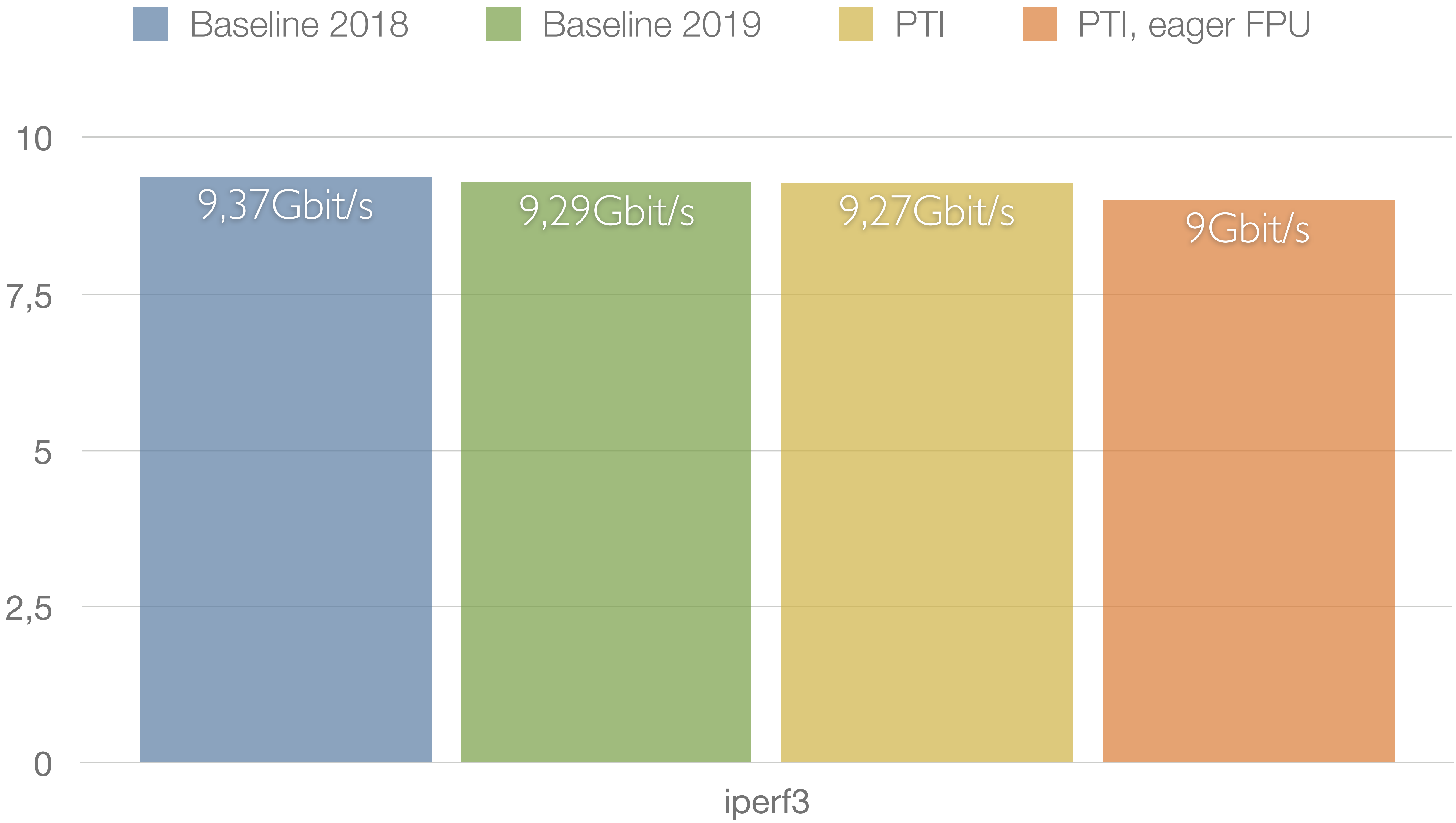
Micro benchmarks - pingpong



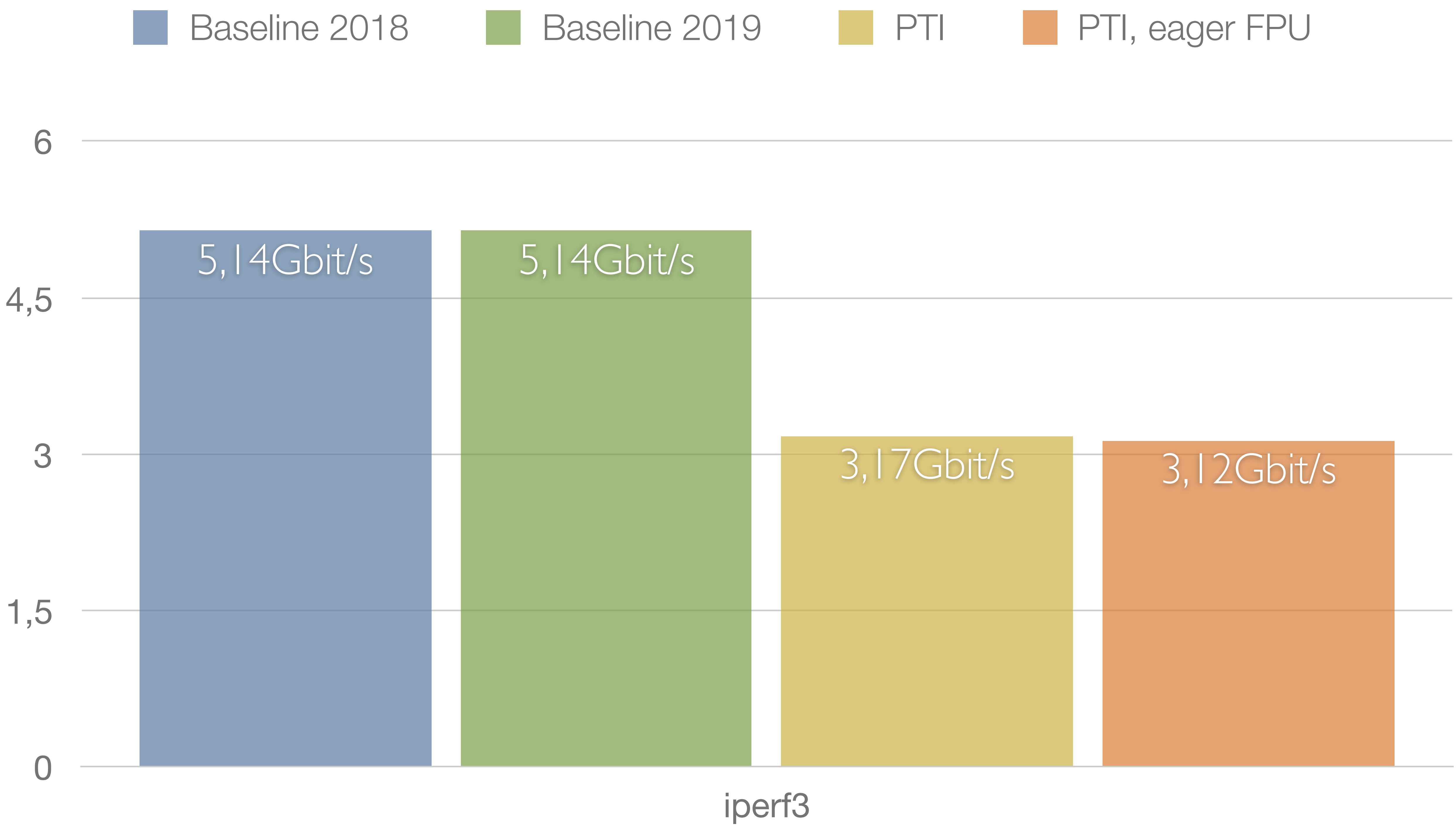
Micro benchmarks - pingpong



Benchmarks - Scenario 1



Benchmarks - Scenario 2



Conclusion

> “Fiasco is still not the fastest microkernel in the world.”

– Me

Conclusion

- Some bugs did not hit as hard
- “missing” features helped us
- Dramatic performance impact
  - Consider alternatives compared to microcode
- Reconsider existing legacy implementations
  - Removed IO page fault
- What to expect in the future? How can we proactively act?
- gcc vs. clang

THANK YOU