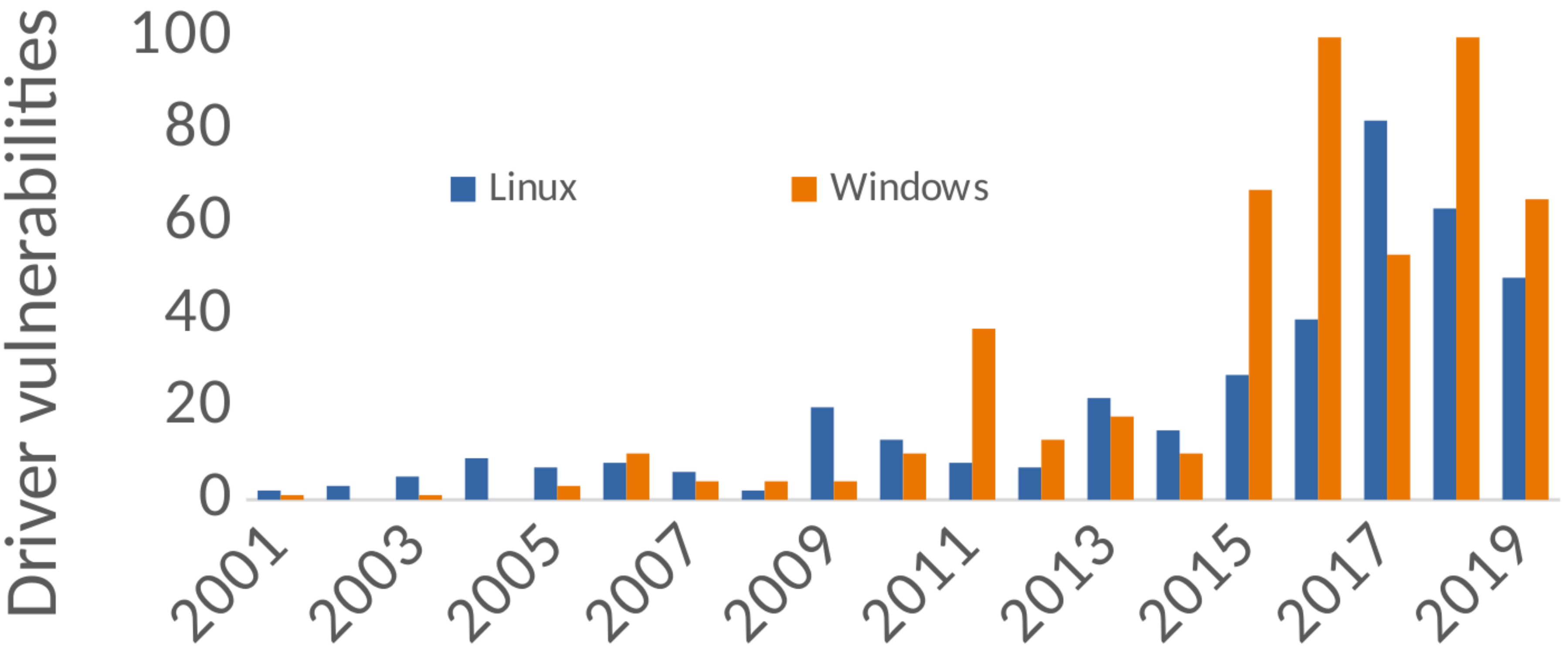
Adelie: Continuous Address Space Layout Re-randomization for Linux Drivers

Ruslan Nikolaev, Hassan Nadeem,
Cathlyn Stone, Binoy Ravindran

Security vulnerabilities in OSs continues to rise



Number of CVEs for device drivers

Attacks

- Control-flow attacks
- Return-Oriented Programming (ROP)
  - ASLR mitigates against traditional ROP
  - More elaborate ROP attacks are still possible
  - KASLR is limited

Contributions

- ✓ Extend KASLR support in Linux
- ✓ Implement stack re-randomization, address encryption, and continuous ASLR on Linux modules

Goals

Generality: Transform all modules to the 64-bit KASLR model

Performance: Avoid costs of copying code and data (re-randomization)

Entropy: Kernel modules can be any distance apart from each other

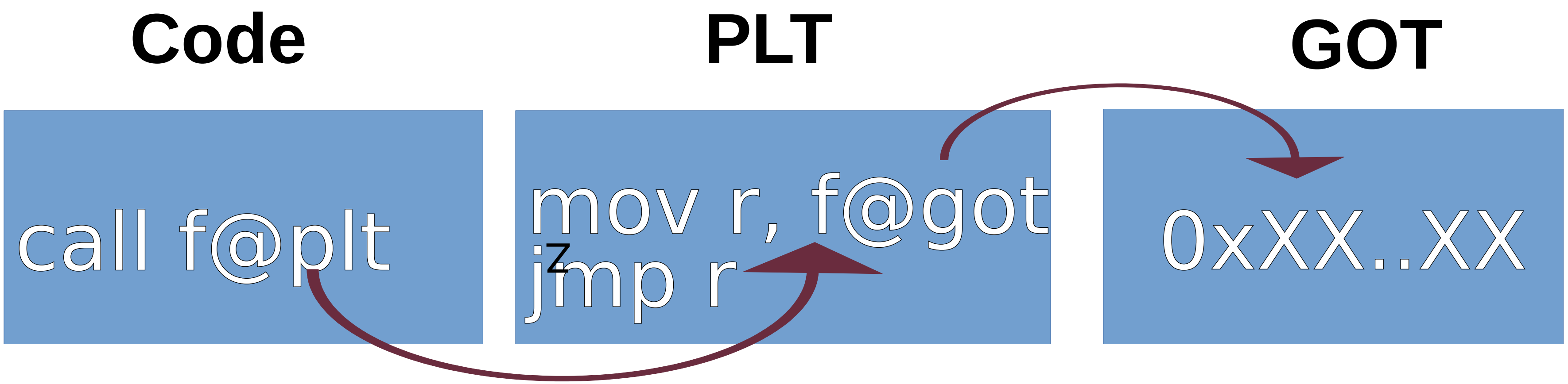
Security: Protect against code reuse attacks

Extending KASLR

- We use a preliminary PIE patch for the Linux kernel
- Cannot use PIE for kernel modules
- We use a more general PIC model for modules, which is similar to shared libraries (with GOT and PLT support)
- Extends KASLR to 64 bits
- Avoids costly absolute-address models such as mcmodel=large

Extending KASLR

- Compilers rely on procedure linkage tables (PLT) and global offset tables (GOT) to call external functions and retrieve external addresses
- We use these to support multiple mappings to code during ongoing re-randomization

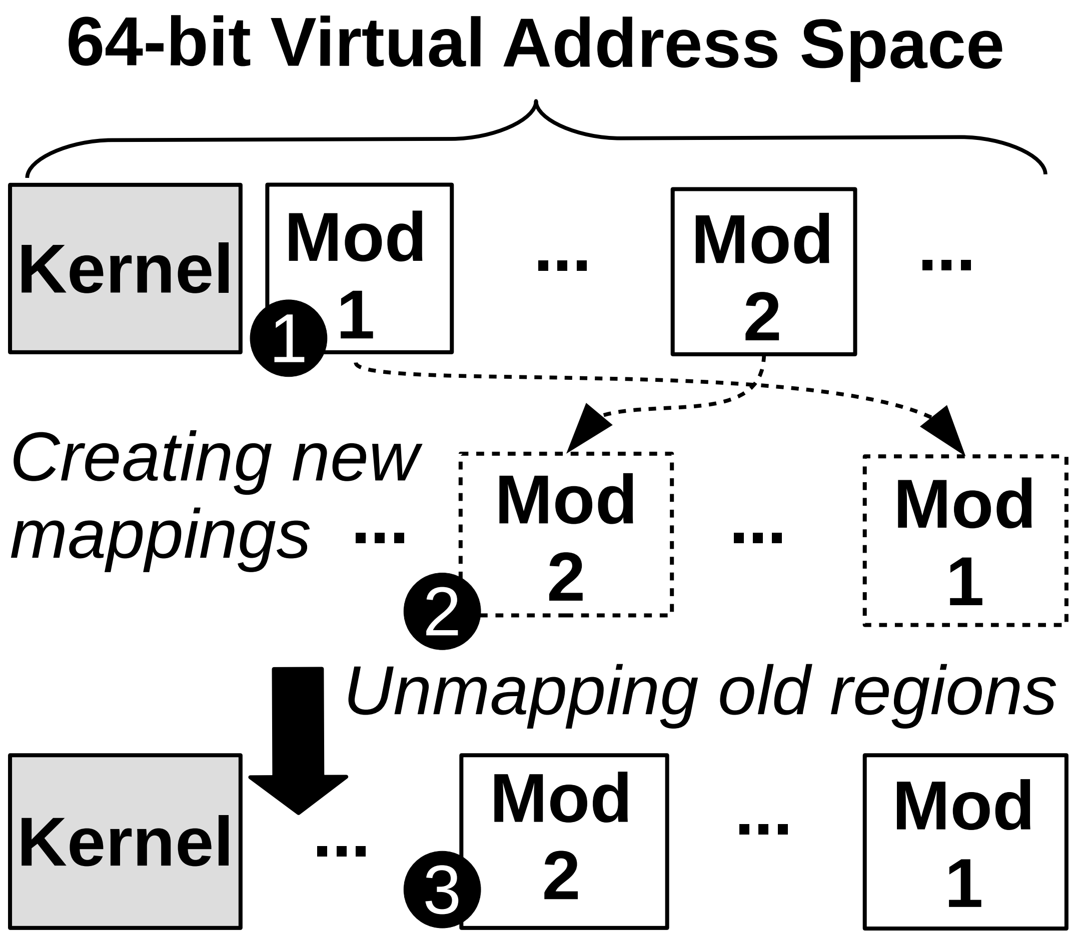


Optimizations

- Spectre-V2
  - Affects indirect calls
  - Impacts the PIC model
    - Optimizations are crucial

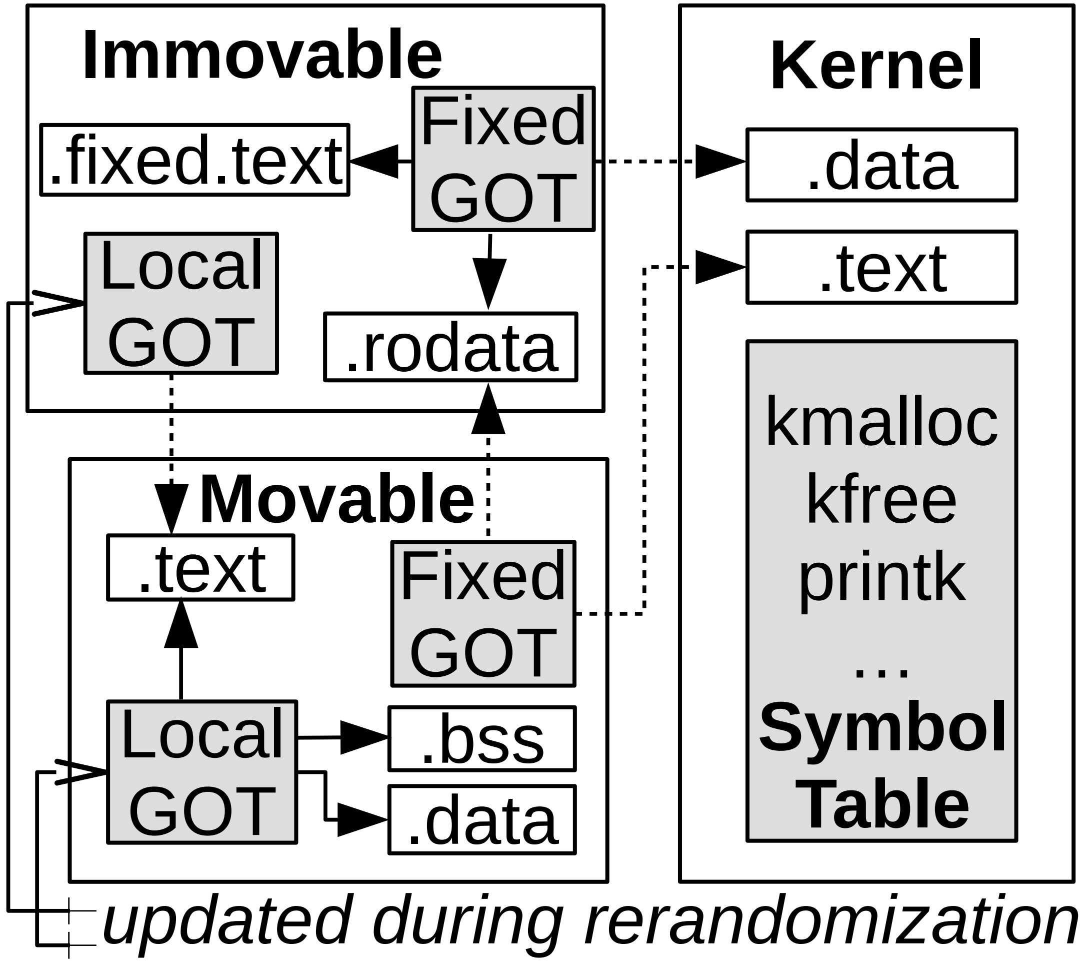
* The picture is taken from Wikipedia

Continuous Module Re-Randomization



Continuous Module Re-Randomization

Use a zero-copying mechanism and organize modules into movable and immovable parts



Continuous Module Re-Randomization

- Use **delayed unmapping** to control address space lifetime
- Track pending calls in a scalable manner with as little overhead as possible
- Enclose operations that access potentially disappearing memory blocks with calls to `mr_start` and `mr_finish`

Continuous Module Re-Randomization

Wrap externally accessible functions in re-randomizable modules, continuously re-randomize stacks

```
long func(long arg) { … }
```

code transformation

```
long func_real(long arg)  [Movable]
{ … } // Renamed function

long func(long arg)     [Immovable]
{
    mr_start();
    get_new_stack();
    long ret = func_real(arg);
    return_old_stack();
    mr_finish();
    return ret;
}               kernel_ref(&func);
```

Continuous Module Re-Randomization

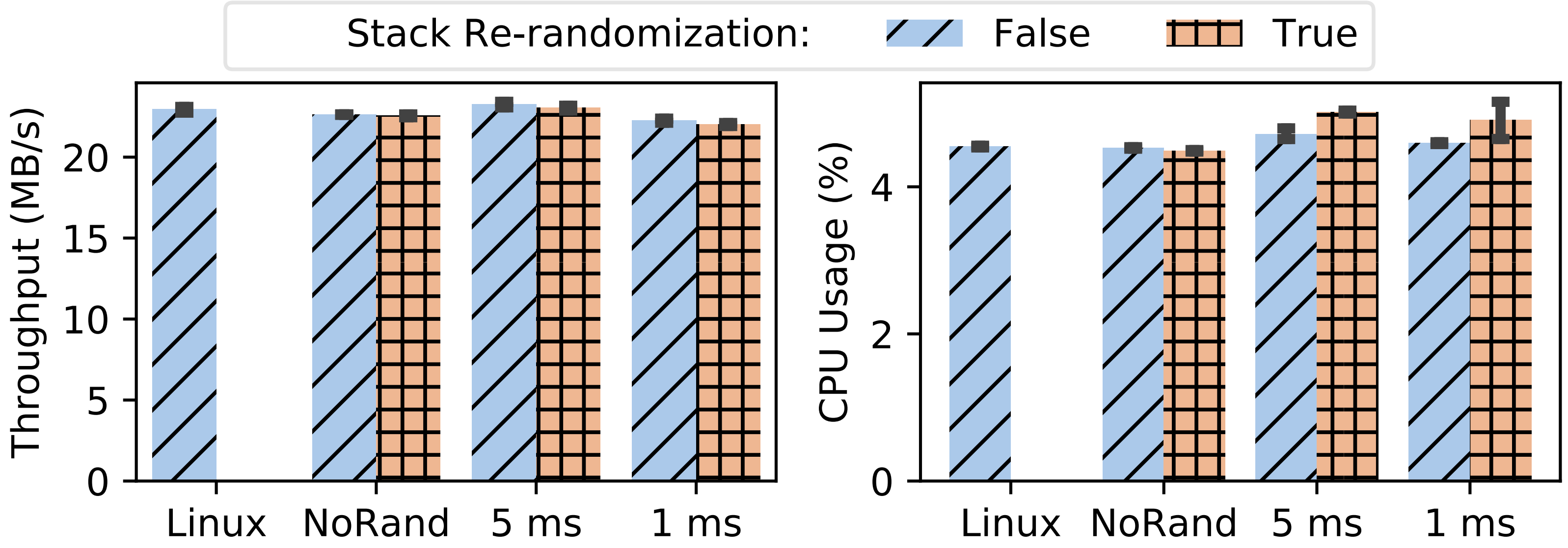
Wrap externally accessible functions in re-randomizable modules, continuously re-randomize stacks

```
get_new_stack (wrapper):
  %rbp = %rsp;    // save stack
  stk = pop_stack_this_cpu();
  if (!stk) stk = alloc_stack();
  %rsp = stk;
return_old_stack (wrapper):
  stk = %rsp;
  %rsp = %rbp;   // restore stack
  push_stack_this_cpu(stk);
prologue/epilogue (non-static):
  mov key@GOTPCREL(%rip), %r11
  xor %r11, (%rsp)  // en/decrypt
  xor %r11, %r11    // %r11 = 0
prologue/epilogue (static):
  push %rbp
  mov key@GOTPCREL(%rip), %rbp
  xor %rbp, 8(%rsp) // en/decrypt
  pop %rbp
```

Evaluation

	Server (for Evaluation)	Load Generator
CPUs	Xeon Silver 4114 2.20GHz	Core i7 4770 3.40GHz
Cores	2x10, no HyperThreading	1x4, no HyperThreading
L1/L2 cache	64 / 1024 KB per core	64 / 256 KB per core
L3 cache	14080 KB	8192 KB
Memory	96 GB	16 GB
Network	Intel E1000E 1GbE	Intel E1000E 1GbE
Storage	Samsung 970 EVO NVMe	Samsung 860 EVO SSD
USB 3.0	Intel C620 xHCI	N/A

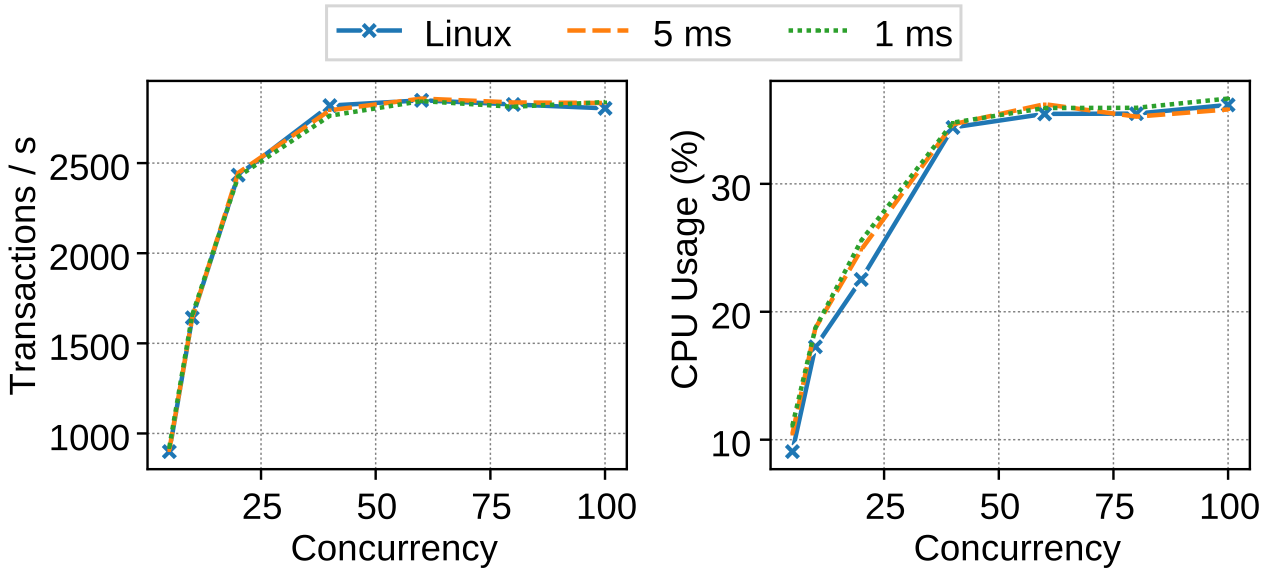
Evaluation



NVMe microbenchmark

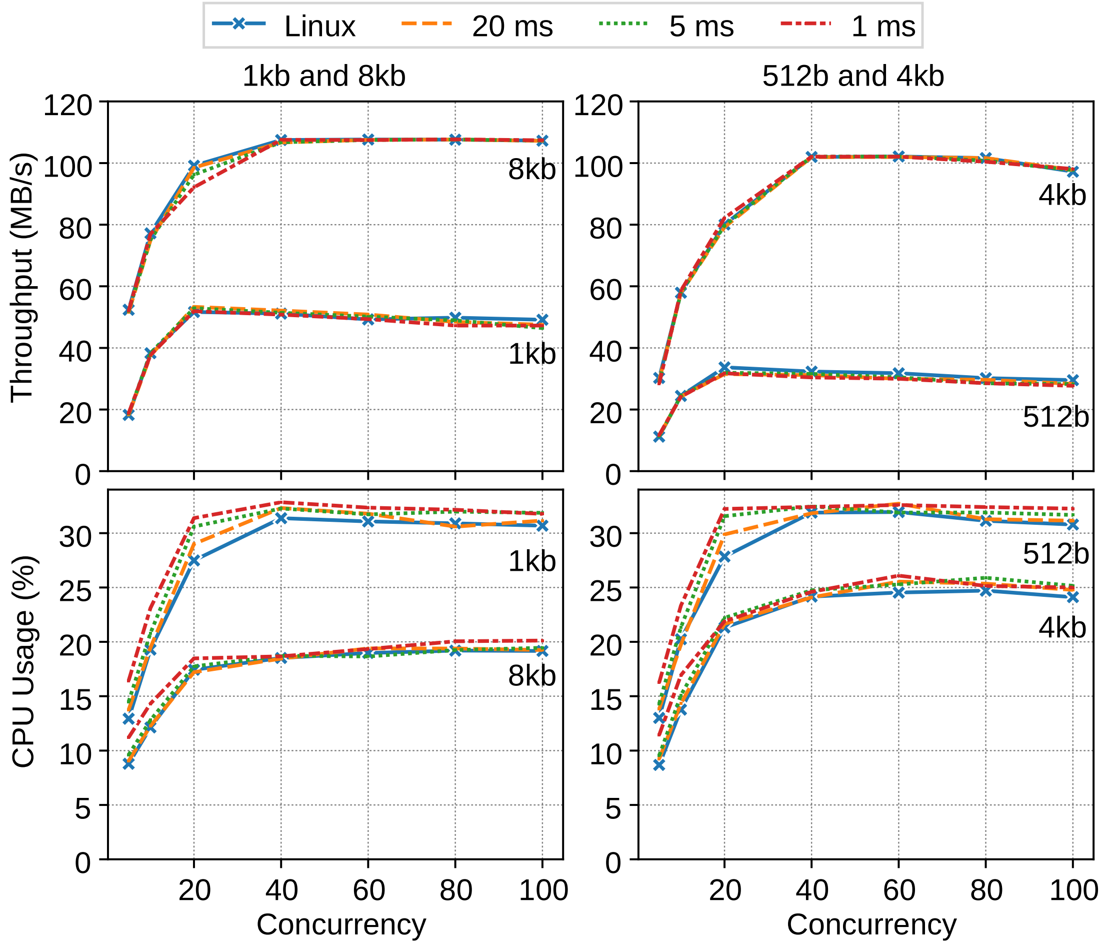
Evaluation

mySQL



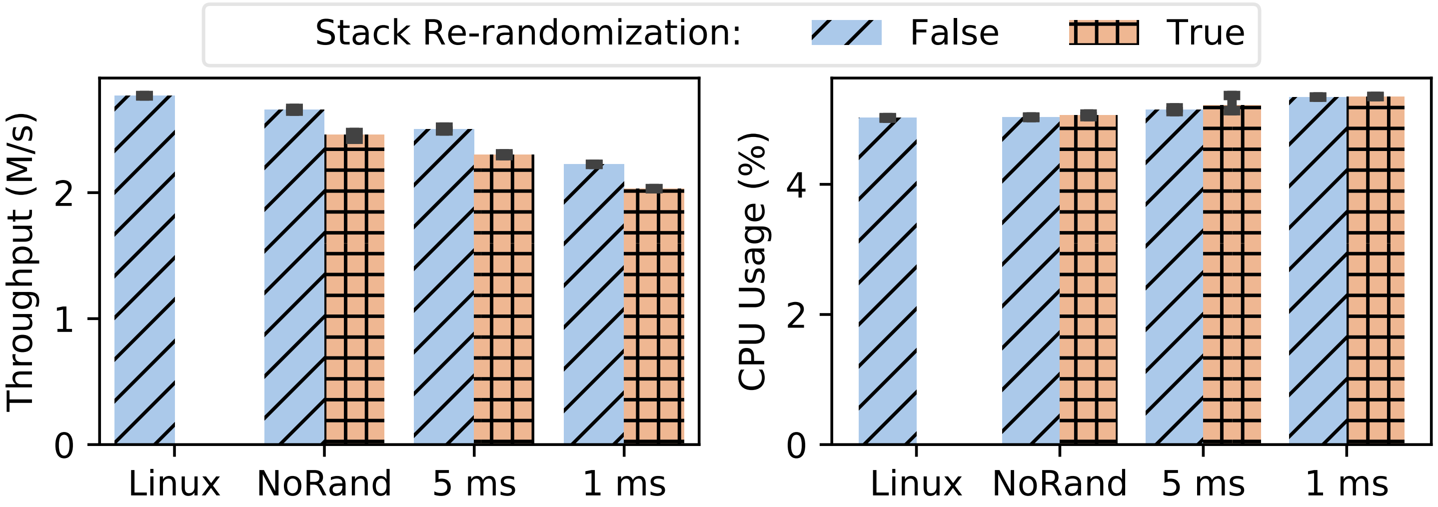
Evaluation

Apache



Evaluation

Extreme ioctl test



Thank you!

Ruslan Nikolaev
rnikola@psu.edu
The Pennsylvania State University

Hassan Nadeem
hnadeem@vt.edu
Virginia Tech

Cathlyn Stone
stonecat@vt.edu
Virginia Tech

Binoy Ravindran
binoy@vt.edu
Virginia Tech

Adelie's source code is available at: https://github.com/adelie-kaslr

PennState

VIRGINIA TECH