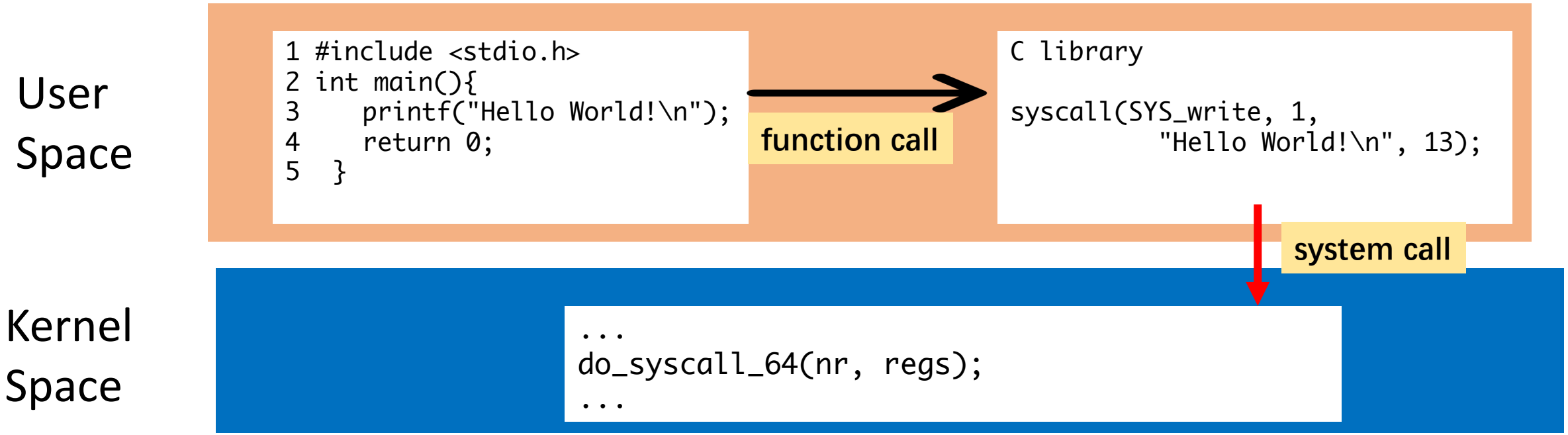
Userspace Bypass: Accelerating Syscall-intensive Applications

Zhe Zhou[1], Yanxiang Bi,[1] **Junpeng Wan**[1, 3], Yangfan Zhou[1], and Zhou Li[2]

[1]Fudan University, [2]University of California, Irvine, [3]Purdue University

Background – System Call

System call is a mechanism that allows a process to request services or functionality from kernel.



Background – System Call Cost

Direct cost

- **Context Switch**
  - Saving and restoring the CPU states
- **Syscall instructions**
  - E.g. syscall/sysret

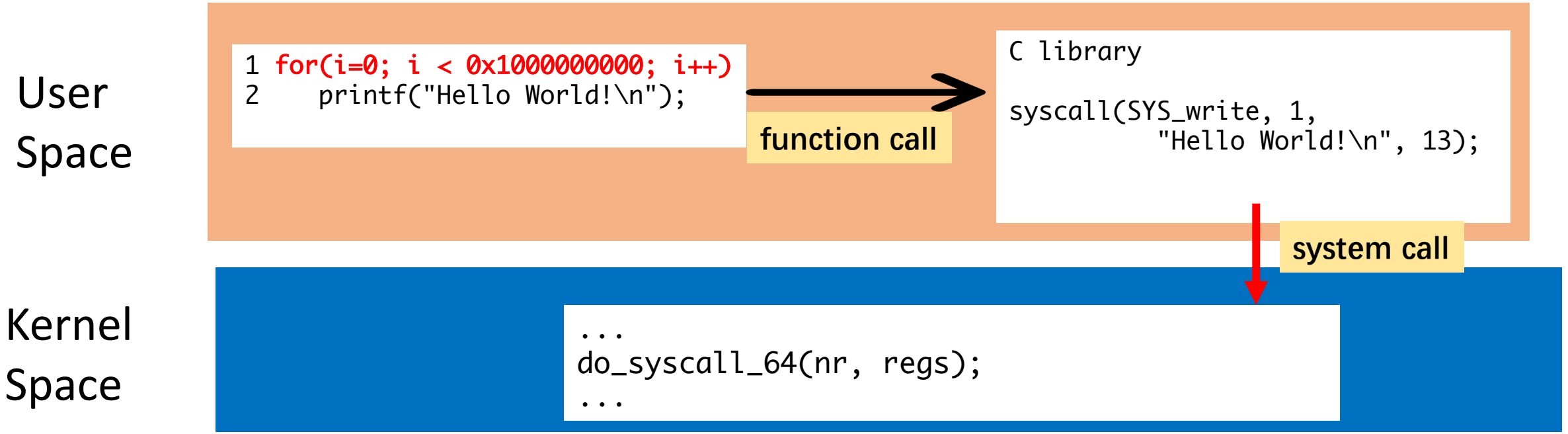
➢ Syscall without operations costs ~ 992 CPU cycles

Indirect cost

- **Cache Pollution**
  - L1 cache and TLB polluted by syscalls
- **Out of Order Execution (OOE) of CPU Stall**
  - To guarantee the execution order
- **Kernel Page Table Isolation(KPTI)**
  - Page table walking, TLB miss

Background – System Call

System call overhead is usually negligible, but sometimes it is significant.
Happens in applications with a huge I/O demand, e.g. Redis and Nginx.



State-of-the-art Solutions

- Asynchronous syscalls
  - Asynchronous system calls, *OLS'07*
- Syscall batching
  - Cassyopia: Compiler assisted system optimization, *HotOS'03*
  - Io_uring: Efficient IO with io_uring
- Unikernel
  - Unikernels: Library operating systems for the cloud, *ACM SIGARCH Computer Architecture News'13*
  - Cubicleos: a library OS with software componentization for practical isolation, *ASPLOS'21*
  - Evaluating the performance of user-space and kernel-space web servers, *CASCON'04*
- In-kernel sandbox
  - eBPF
  - Privbox: Faster system calls through sandboxed privileged execution, *ATC'22*
- Kernel bypass
  - DPDK

State-of-the-art Solutions

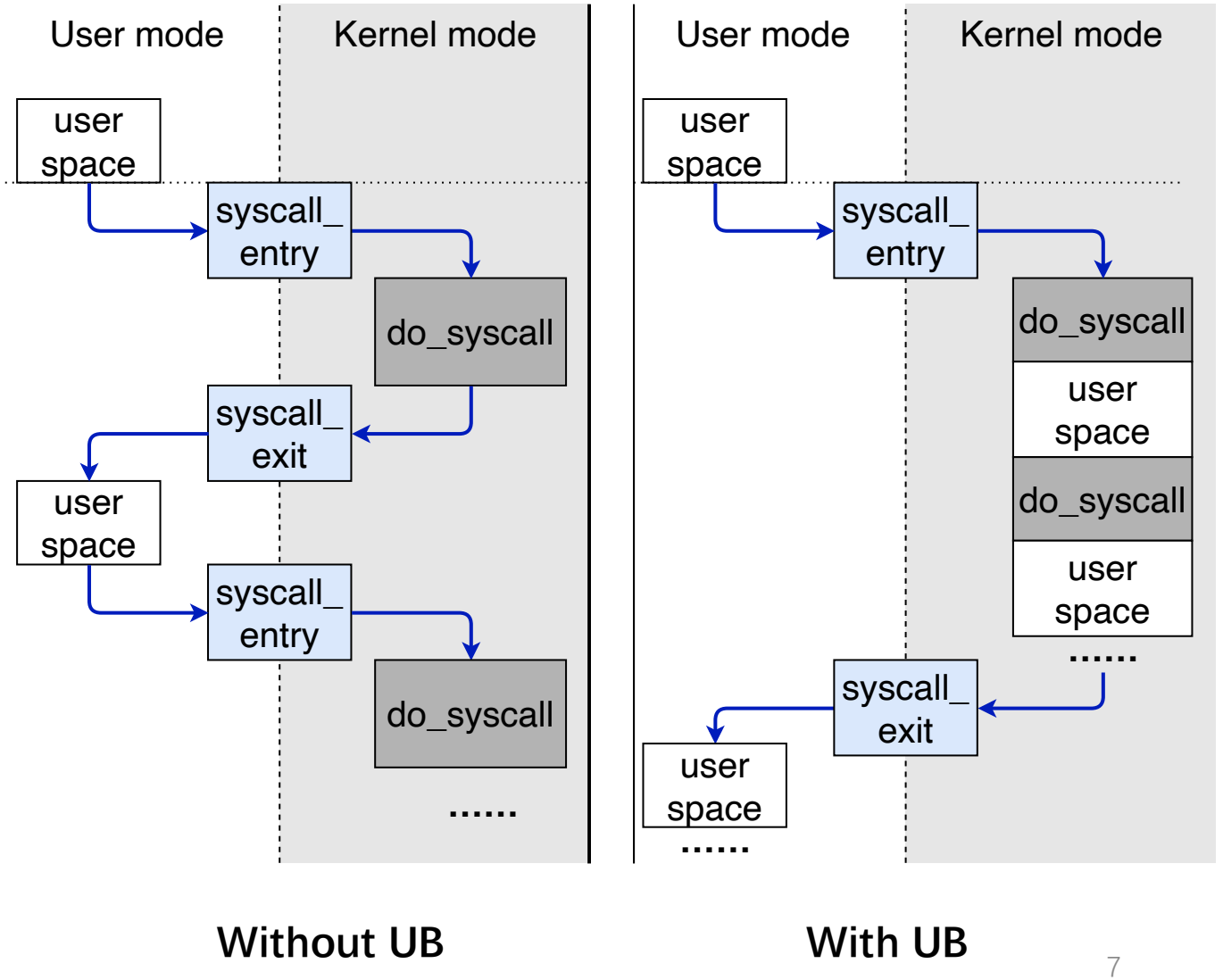
- Asynchronous syscalls
  - Asynchronous system calls, *OLS'07*
- Syscall batching
  - Cassyopia: Compiler ass
  - Io_uring: Efficient IO wi
- Unikernel
  - Unikernels: Library oper *mputer Architecture News'13*
  - Cubicleos: a library OS v olation, *ASPLOS'21*
  - Evaluating the performa ers, *CASCON'04*
- In-kernel sandbox
  - eBPF
  - Privbox: Faster system calls through sandboxed privileged execution, *ATC'22*
- Kernel bypass
  - DPDK

Development efforts

Userspace Bypass (UB)

No development efforts to reduce system call cost

UB **transparently** translates user space instructions to kernel space for syscall-intensive applications.



UB Design

Hot Syscall Profiler

- Run in kernel mode
- Identify hot syscalls

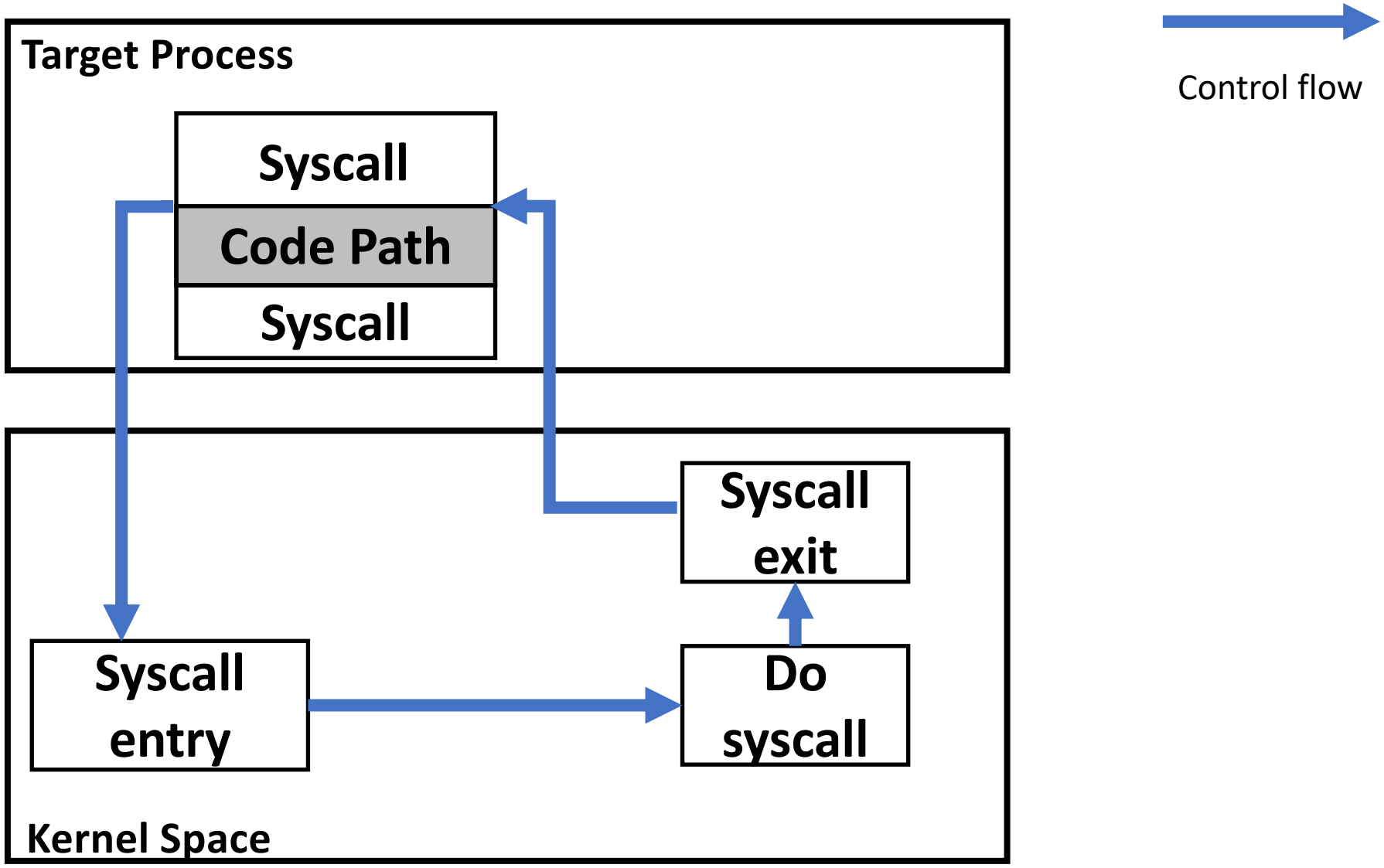
BTC Translator

- Run in user mode
- Translate user space instructions between hot syscalls into BTC
- Security guarantees

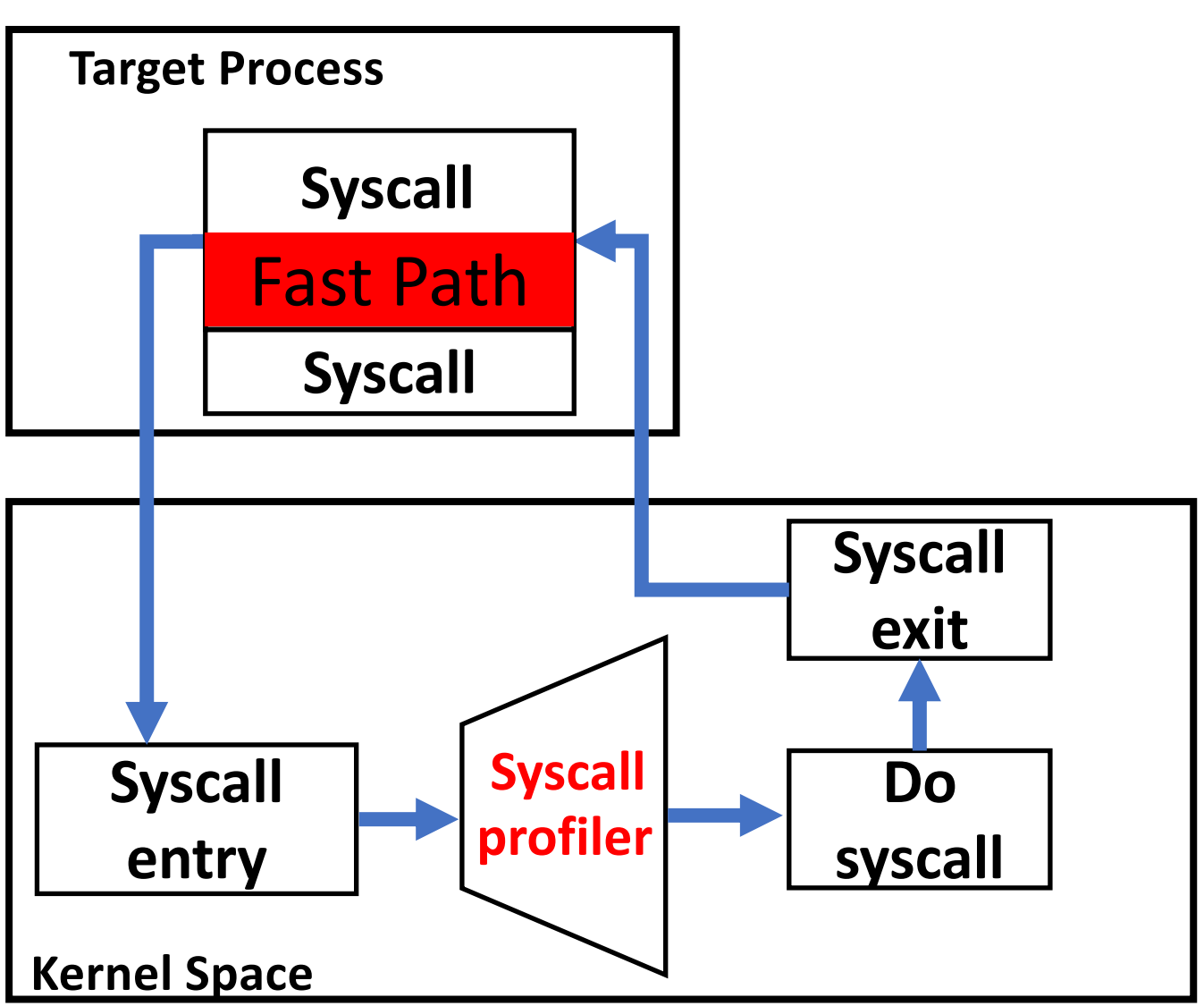
BTC Runtime

- Run in kernel mode
- Execute BTC

UB Design – Regular Syscall



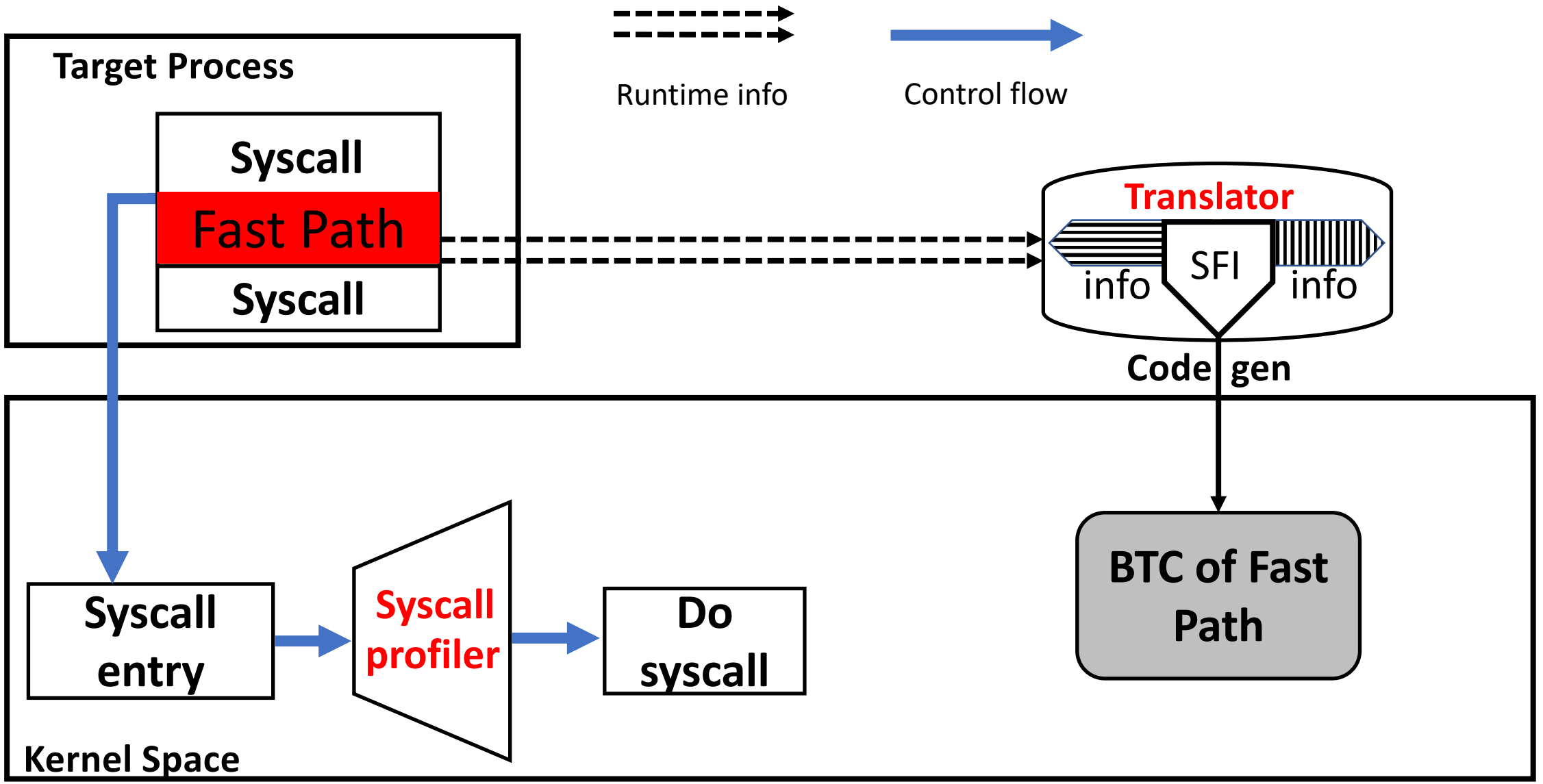
UB Design – Hot Syscall profiler



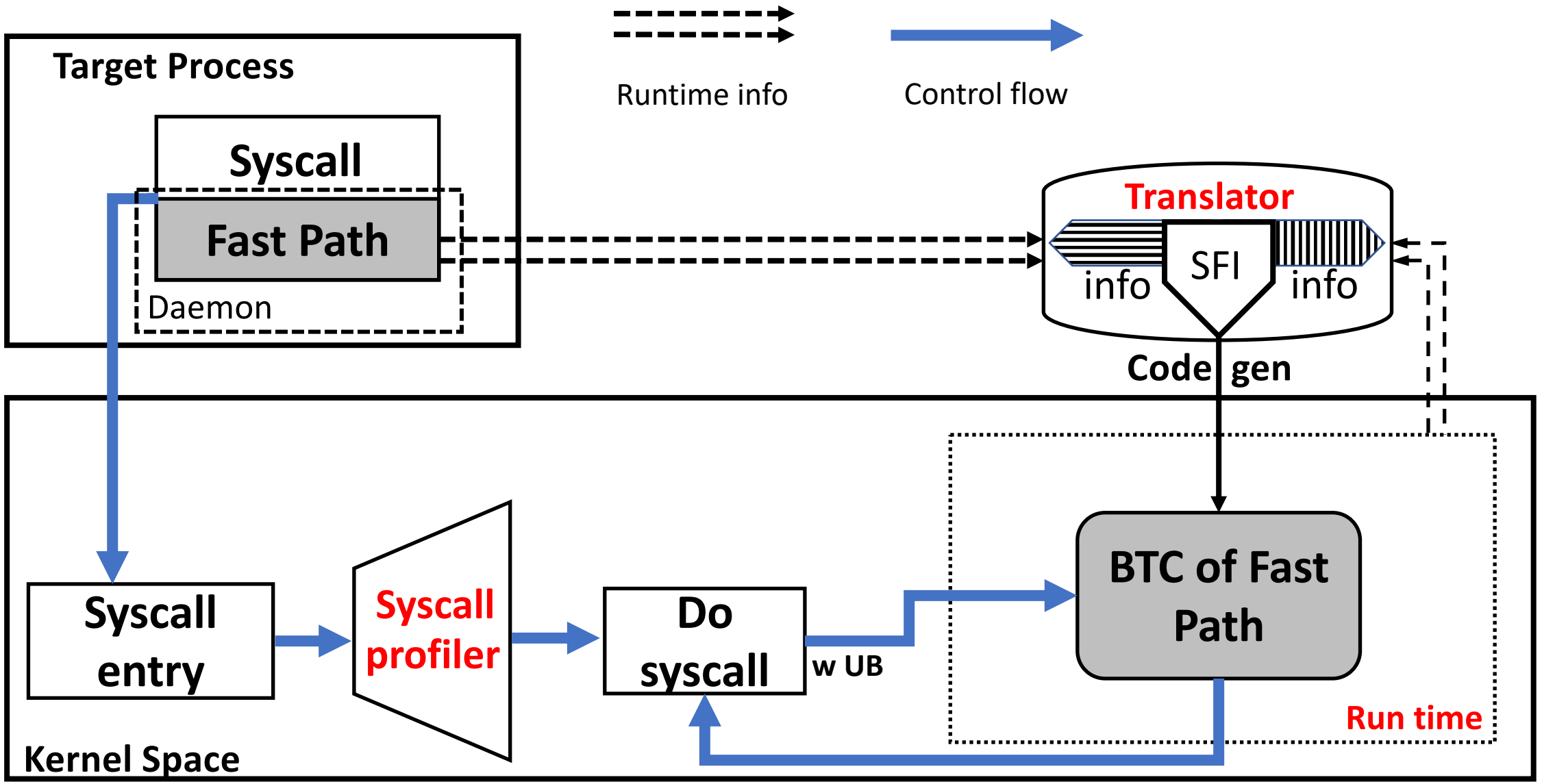
Control flow

- **Syscall sampling & Coarse-grained profiling**
  - Find syscall intensive thread candidates
- **Fine-grained profiling**
  - Find hot syscalls inside these threads

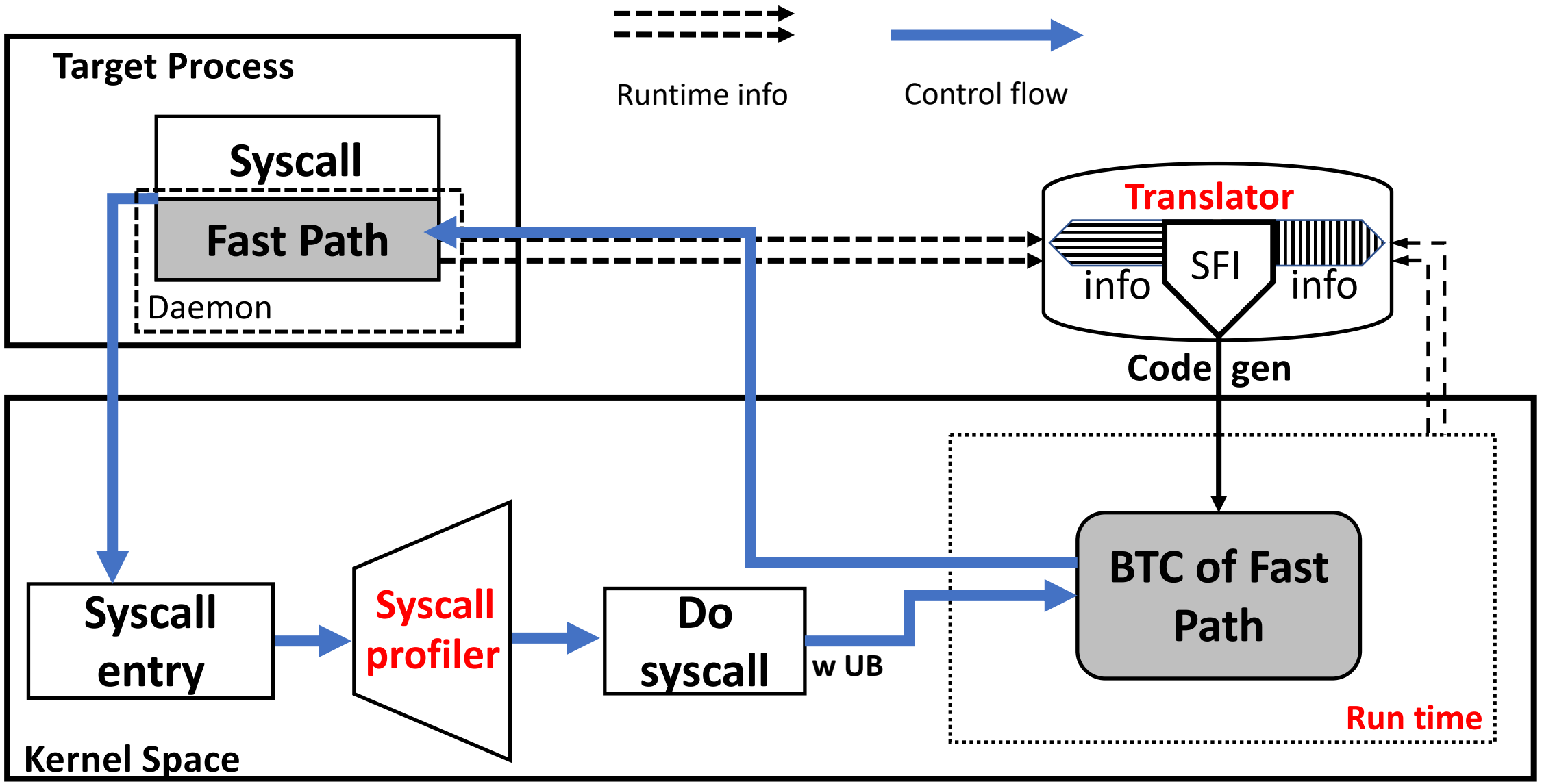
UB Design – BTC Translator



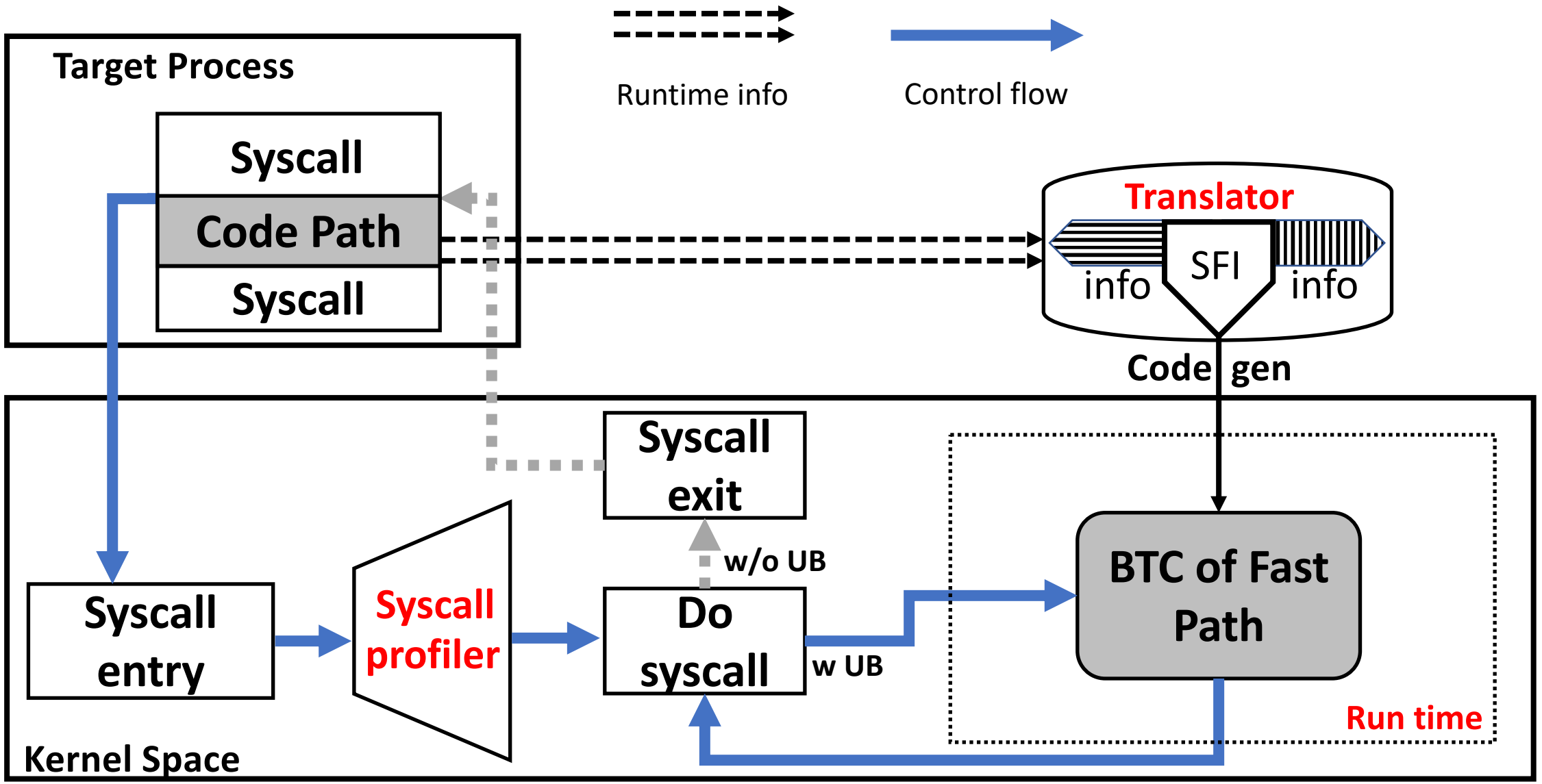
UB Design – BTC Runtime



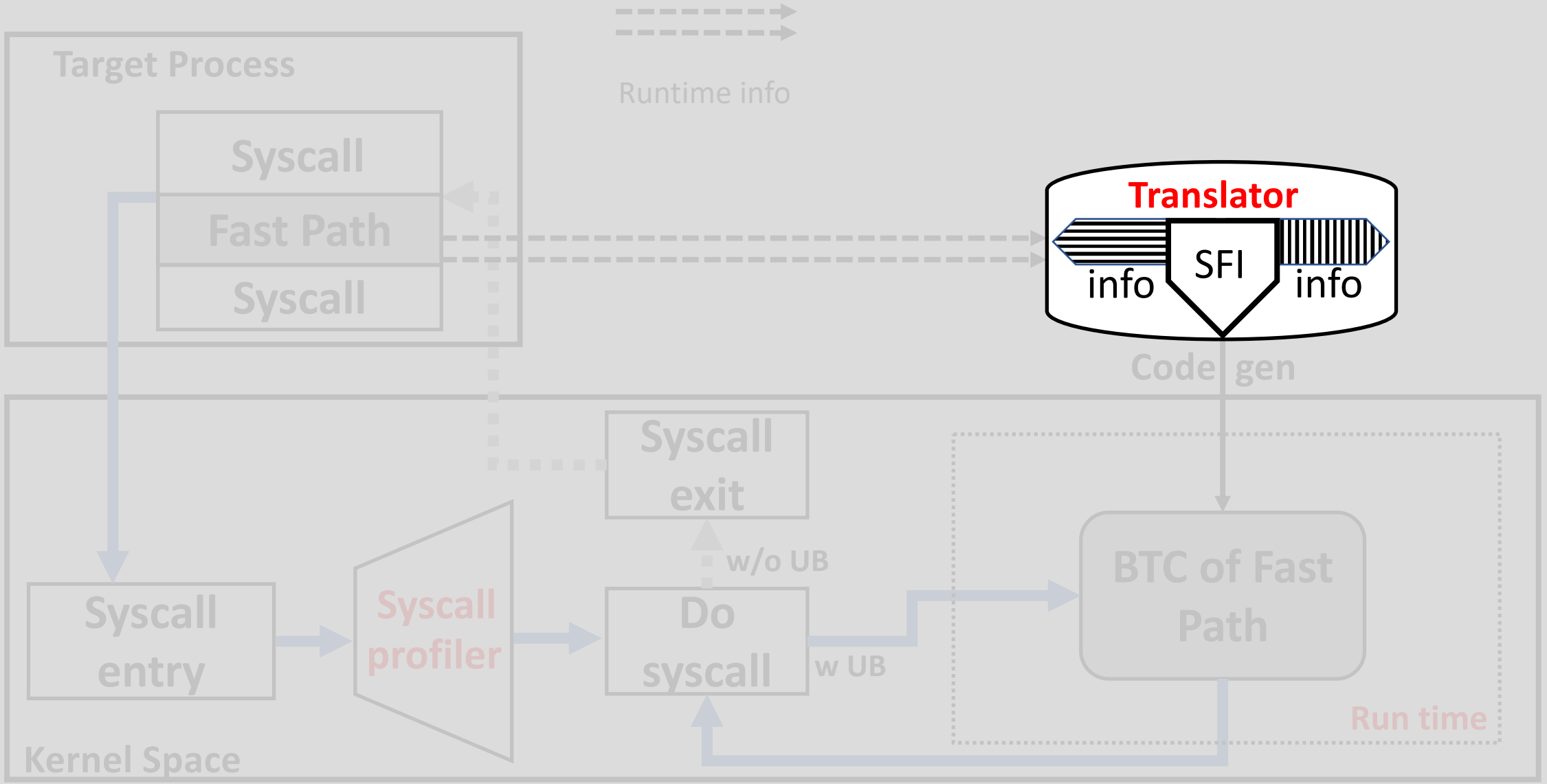
UB Design – BTC Runtime



UB Design

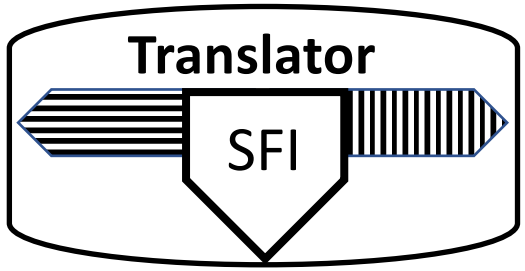

Runtime info
Control flow
Target Process
Syscall
Code Path
Syscall
Translator
info
SFI
info
Code gen
Syscall exit
w/o UB
Syscall entry
Syscall profiler
Do syscall
w UB
BTC of Fast Path
Run time
Kernel Space

UB Design


Runtime info
Target Process
Syscall
Fast Path
Syscall
Translator
info
SFI
info
Code gen
Syscall exit
w/o UB
Syscall entry
Syscall profiler
Do syscall
w UB
BTC of Fast Path
Run time
Kernel Space

BTC Translator

Function: Disassemble the fast path binary and compile them to BTC



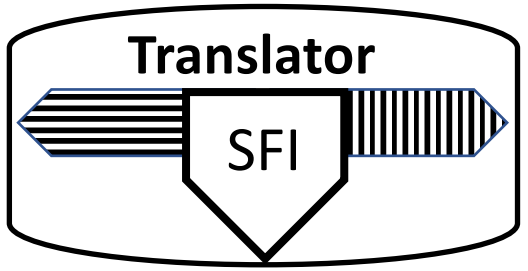
Problem: Buggy and malicious userspace code could corrupt kernel

Solution: Implement SFI (Software Fault Isolation) [1] in the BTC translator.

- Register Remapping - protect kernel registers and stack
- Instruction Sanitization - avoid privilege escalation
- Memory Access Sanitization - prevent unauthorized access to the kernel memory
- Branch Sanitization – prevent unauthorized kernel code execution

1. Native client: A sandbox for portable, untrusted x86 native code

BTC Translator – JIT Style



- Direct branches are translated at start because the target address is known when translating
- Indirect branches will be translated until the target address is known during runtime

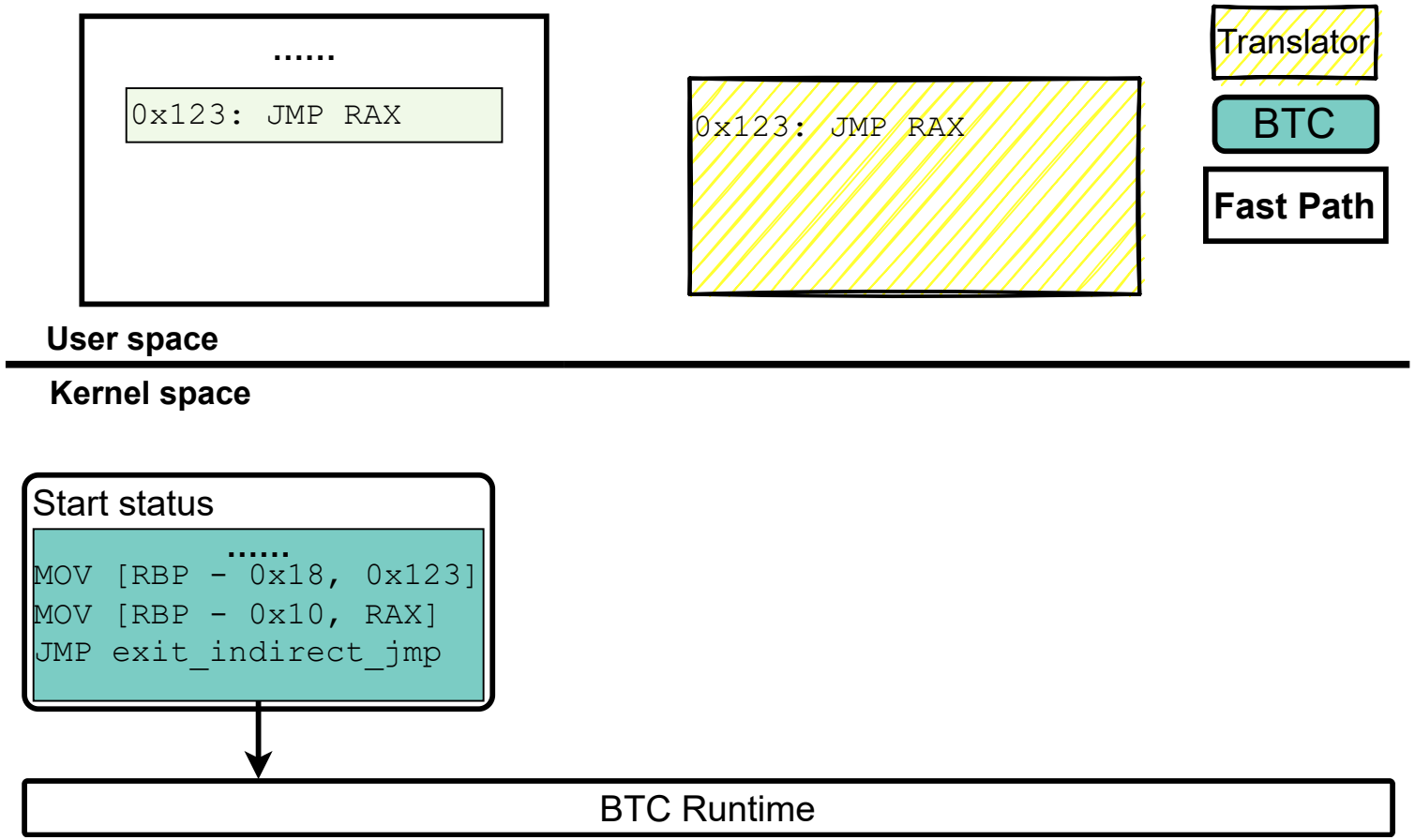
```
jmp label;
ret

label:
mov rax, 1
ret
```

Direct Branch

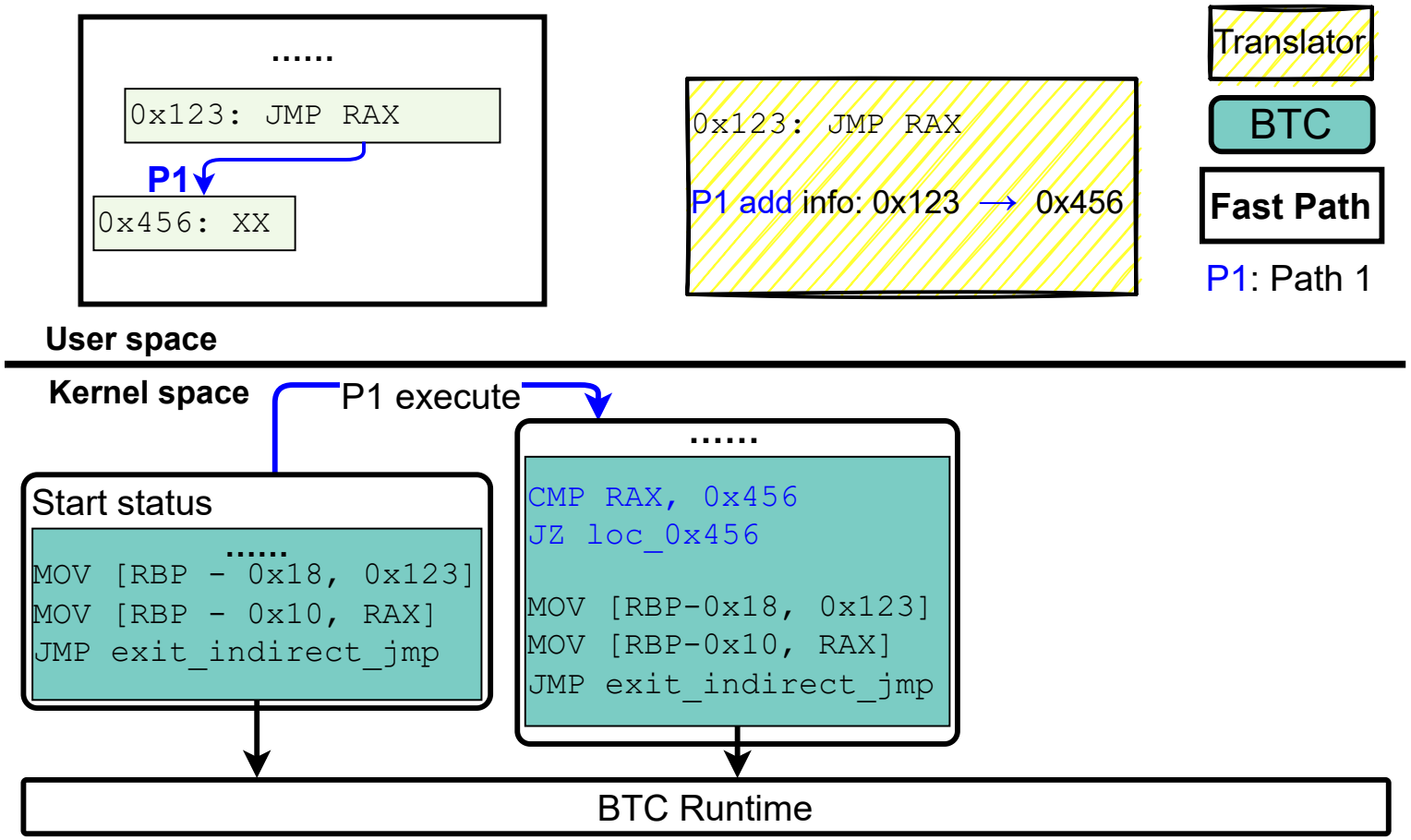
```
jmp rax
ret

label1:
mov rax, 1
ret

label2:
mov rax, 2
ret
```

Indirect Branch

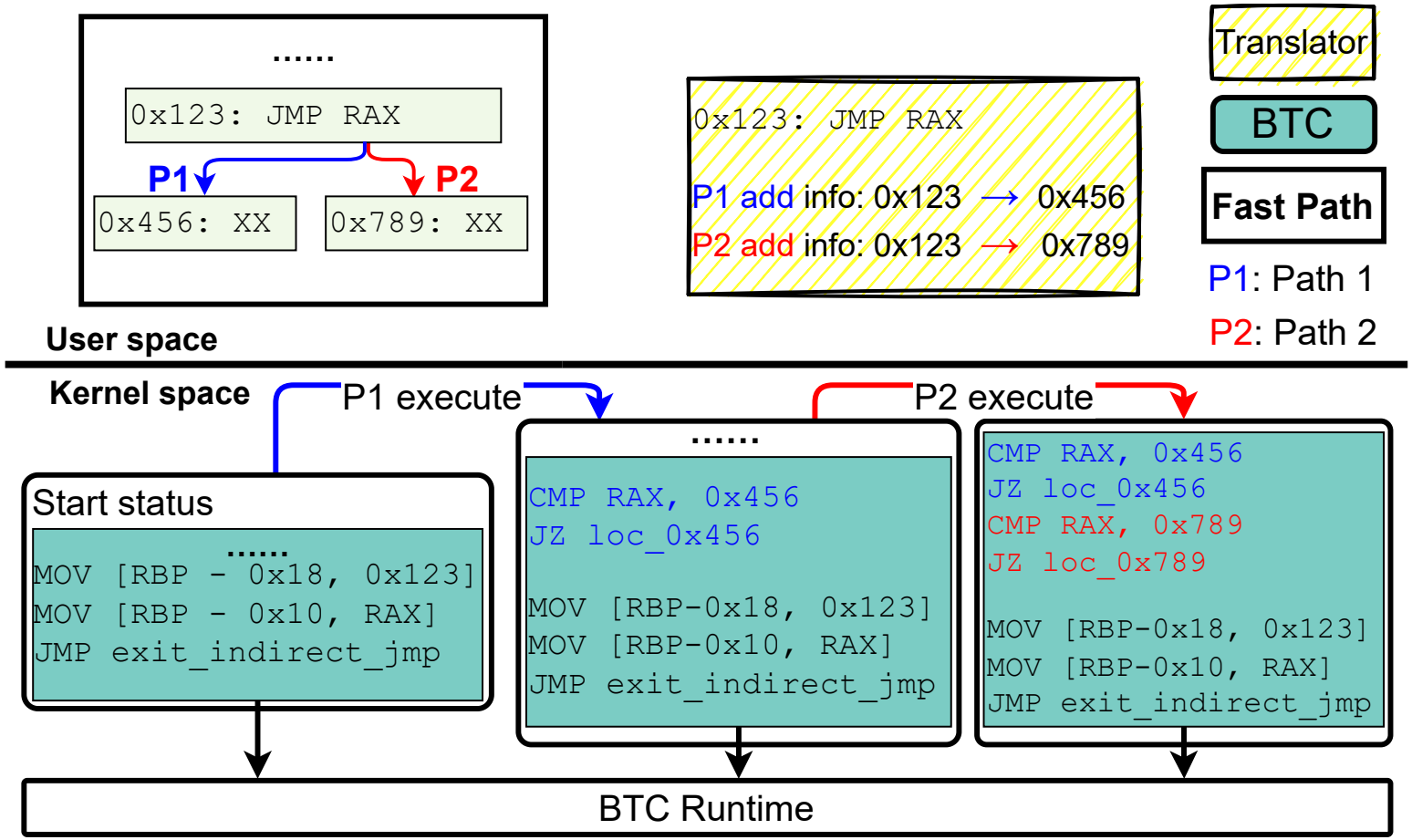
BTC Translator – Indirect Branch



BTC Translator – Indirect Branch



BTC Translator – Indirect Branch



Implementation

- The hot syscall identifier and BTC runtime written in **C**
- The BTC translator written in **Python**
- Modify Linux kernel for less than 30 LOC

Evaluation

The acceleration of

- I/O micro-benchmark
  - Applications that purely perform file I/O operations
- Raw socket network packet process
- Redis
- Nginx

Settings

- Linux KPTI On / Off × Physical machine/ Virtual Machine

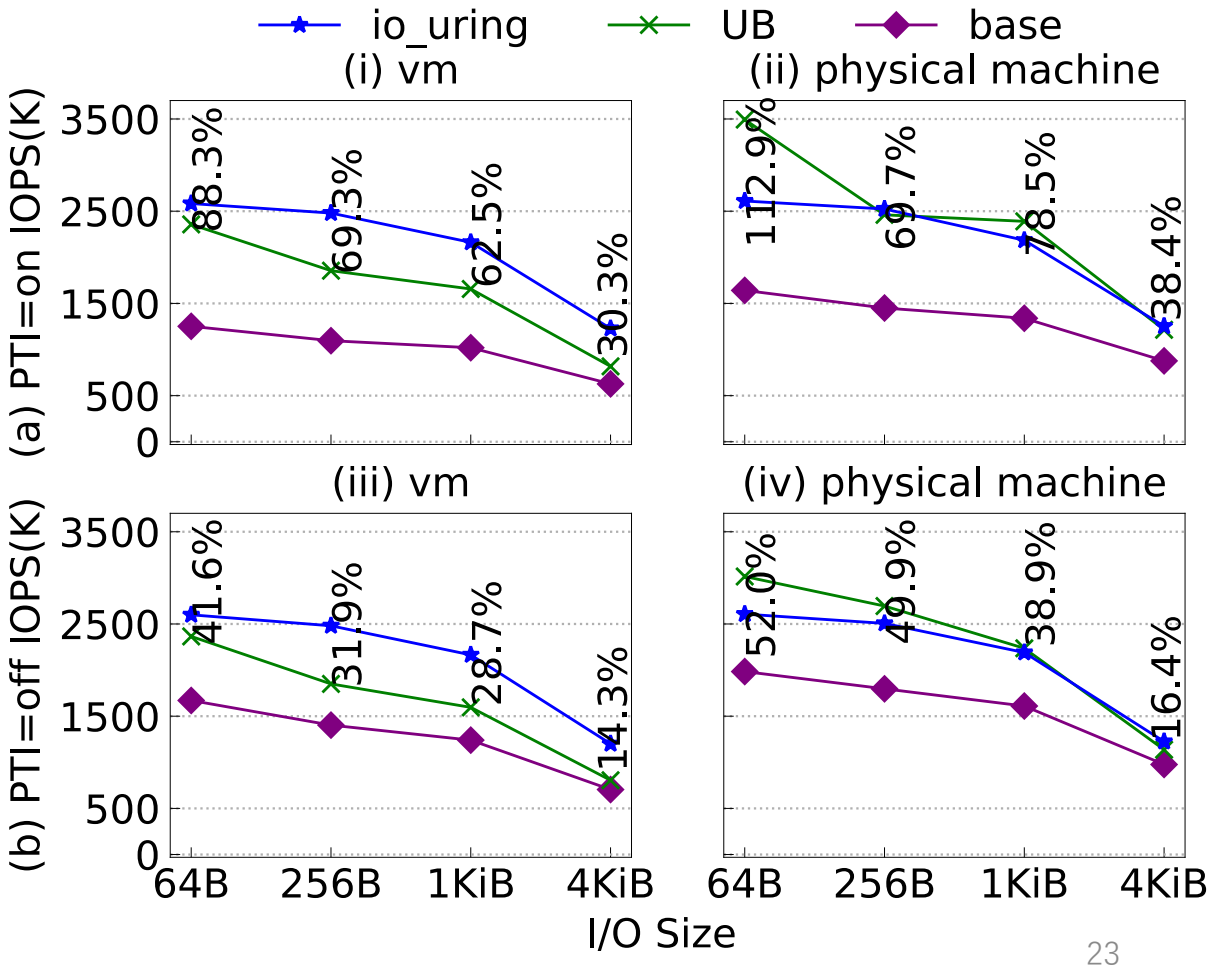
Evaluation - I/O Micro-benchmark

In-memory file access (syscall `read`)

```
for(i = 0; i < DOTS; i++){
    uint64_t ret = pread(fd, buf_arr[i % USE_PAGES],
                         buffer_size, offset);
    offset = next_pos(offset);
}
```

- Get **comparable performance** with io_uring

- KPTI ON
  - 40% - 110% on Physical machine
  - 30% - 90% on VM
- KPTI OFF
  - 16% - 52% on Physical machine
  - 14% - 41% on VM



Evaluation

A brief view of the acceleration rate

- Raw socket => 30% - 40%
- Redis => -5% - 16%
- Nginx => -1% - 13%

(Physical machine with KPTI on)

	Test	VM	Physical
w/ PTI	**In-mem**	30.3% – 88.3%	38.4% – 112.9%
	Redis GET	-3.7% – 10.8%	-5.4% – 6.4%
	Redis SET	-0.4% – 12.4%	-3.2% – 16.1%
	Nginx	0.4% – 10.9%	-1.4% – 13.4%
	Socket	31.5% – 34.3%	30.9% – 38.6%
w/o PTI	**In-mem**	14.3% – 41.6%	16.4% – 52.0%
	Redis GET	-2.0% – 4.6%	-6.4% – 3.9%
	Redis SET	-5.5% – 4.9%	-0.9% – 2.8%
	Nginx	-1.2% – -0.3%	-0.2% – 3.0%
	Socket	14.5% – 17.8%	9.2% – 19.8%

Conclusion

- We propose Userspace Bypass(UB) which makes syscall much cheaper
- UB requires no extra development efforts
- UB requires minimal system architecture changes

Code available at: *https://github.com/glarer/UserspaceBypass*

THANKS!