IPC-Direct: Fast and Compatible Inter-Process Communication in User Space
SOSP'17 SRC #39

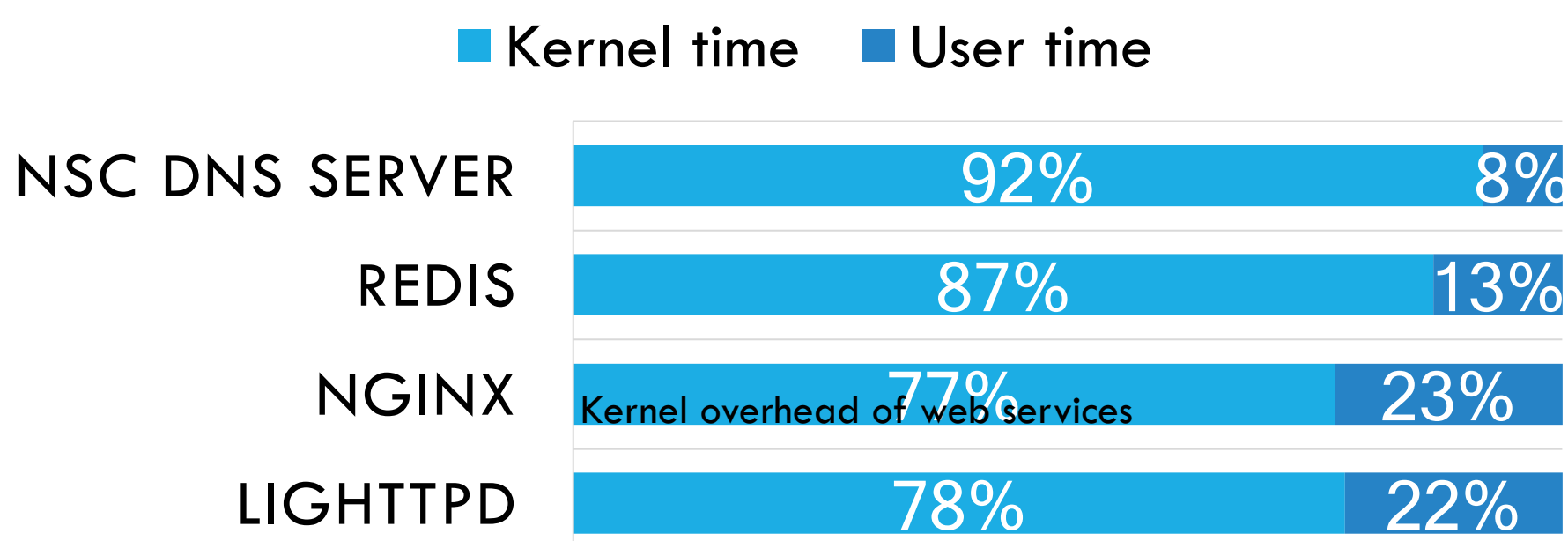Bojie Li, Tianyi Cui, Zibo Wang, Lintao Zhang
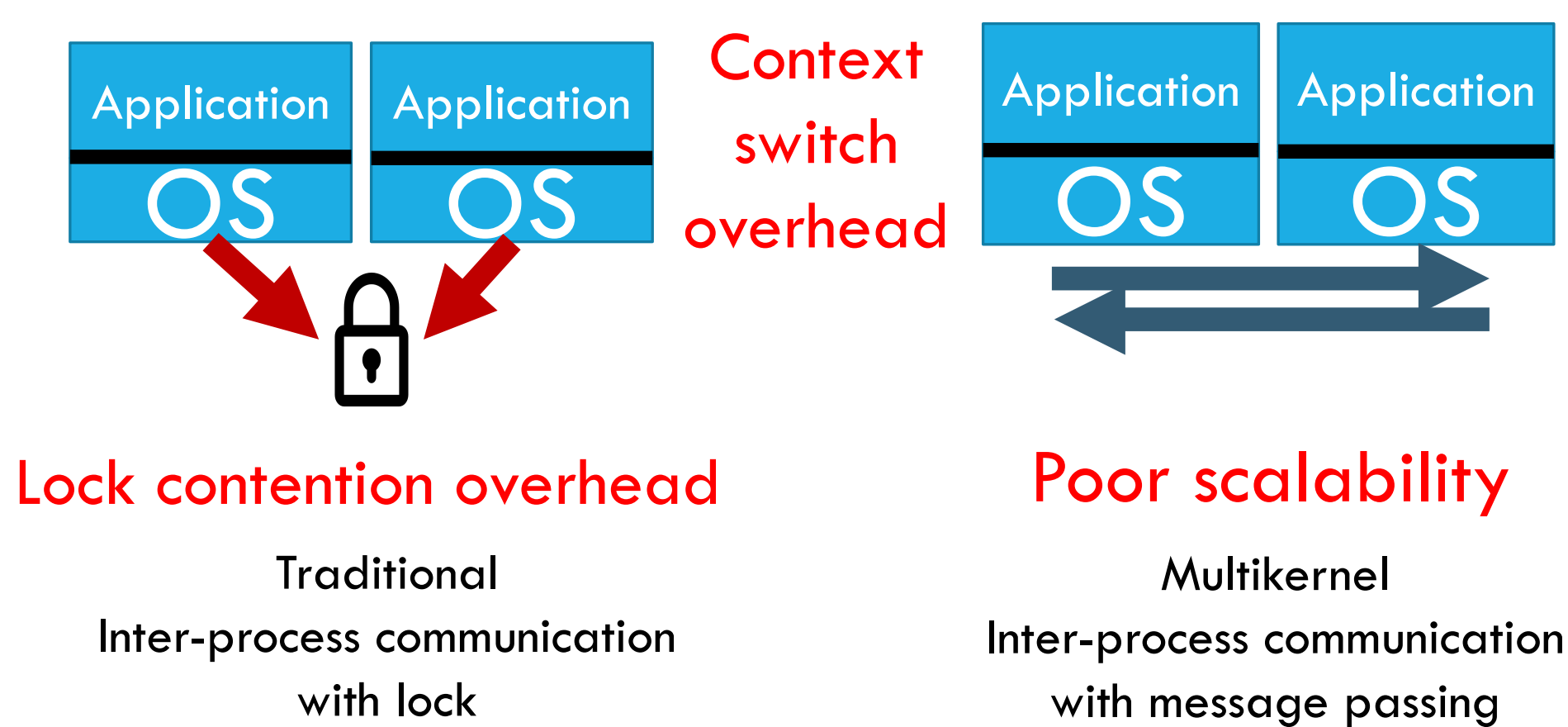
Microsoft **Research**

Kernel overhead dominates web service performance

sources of kernel overhead:
- unnecessary processing in kernels
- context switch
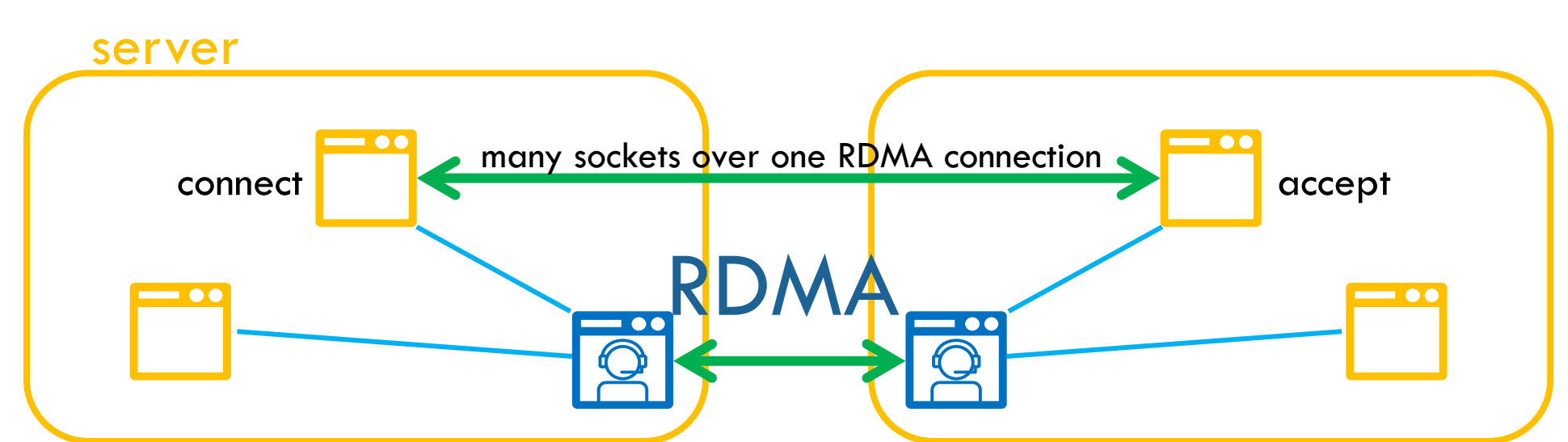- distributed coordination has high contention

■ Kernel time ■ User time

NSC DNS SERVER	92% / 8%
REDIS	87% / 13%
NGINX	77% / 23%
LIGHTTPD	78% / 22%

Kernel overhead of web services

Bottleneck of the current design

Application Application
OS OS

Context switch overhead

Application Application
OS OS

Lock contention overhead
Traditional
Inter-process communication with lock

Poor scalability
Multikernel
Inter-process communication with message passing

Our design: use dedicated coordination core for monitor process

No context switch, better locality

Application	Application	User space monitor process

Shared memory
OS   OS

scalable
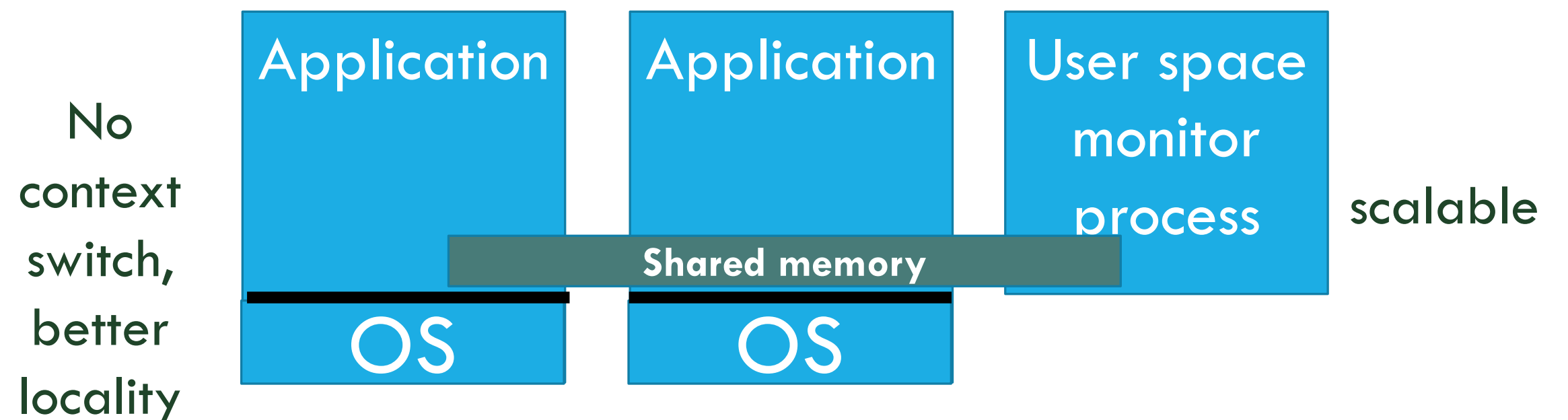
No lock overhead
Inter-process communication with centralized coordinator

server

connect — many sockets over one RDMA connection — accept

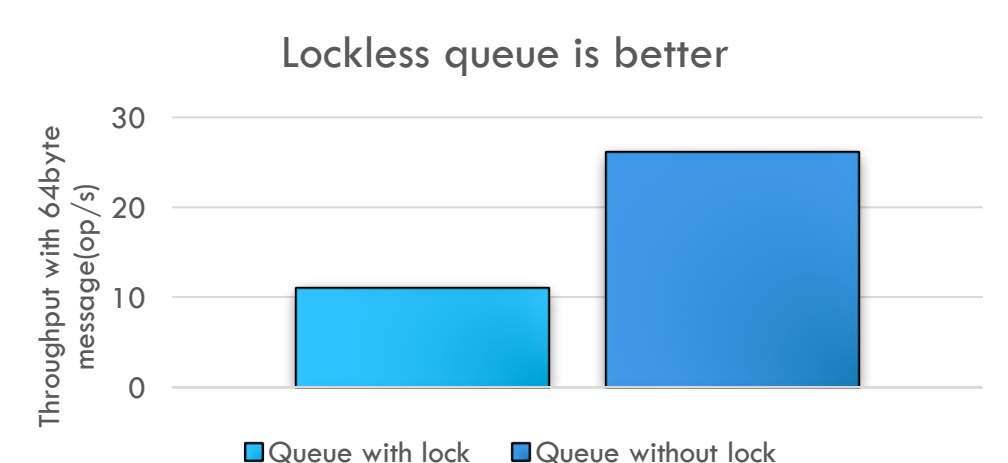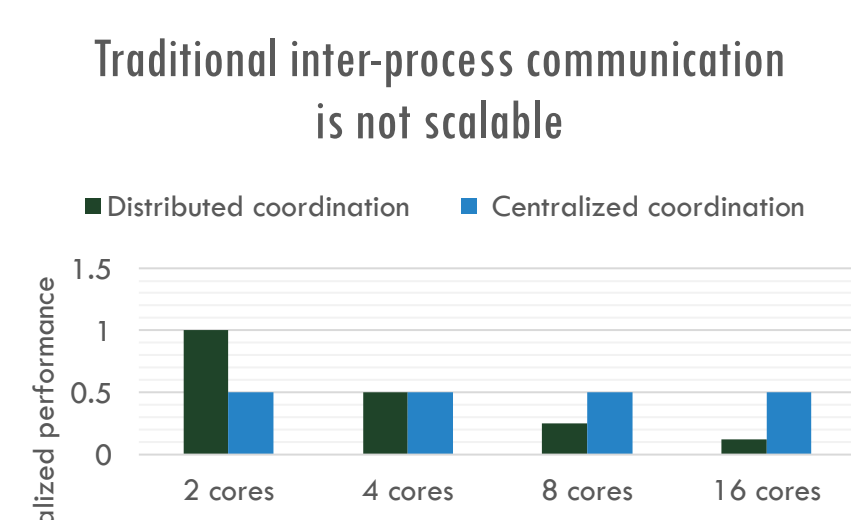RDMA

Scale to multiple server with RDMA

Our advantage
- High throughput
- Low latency
- Compatible with POSIX API (using LD_PRELOAD)

Two possible implementations of IPC-Direct

Polling from different request queue

Monitor Process: Request Parser → Dispatcher → Event Generator
Request metadata / Event metadata

Service Process:
Process 1: Event queue / Application / Metadata / Request queue — Data → Shared Buffer → Data
Process 3: Event queue / Metadata / Application / Request queue
Process 2: Event queue / Application / Metadata / Request queue — Data → Shared Buffer → Data

User Mode
Kernel Mode

Using one shared request queue with atomic operation

Monitor Process: Request Parser → Dispatcher → Event Generator
Request / Shared Request queue / Event metadata

Service Process:
Process 1: Event queue / Application — Data → Shared Buffer → Data
Process 3: Event queue / Metadata / Application
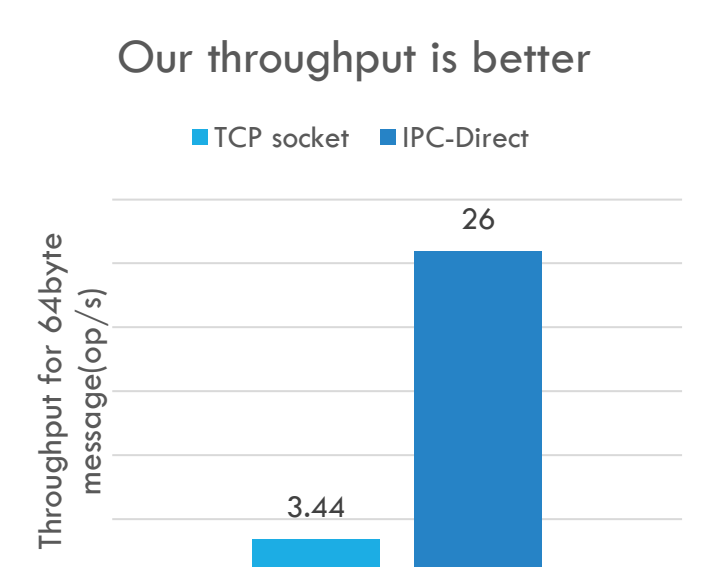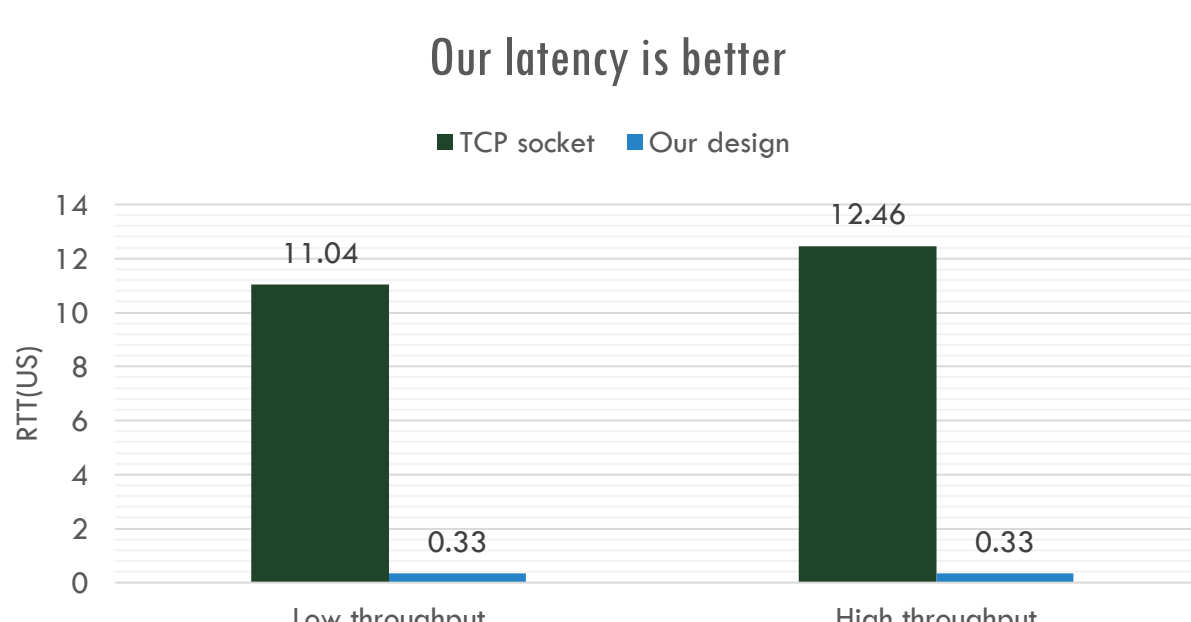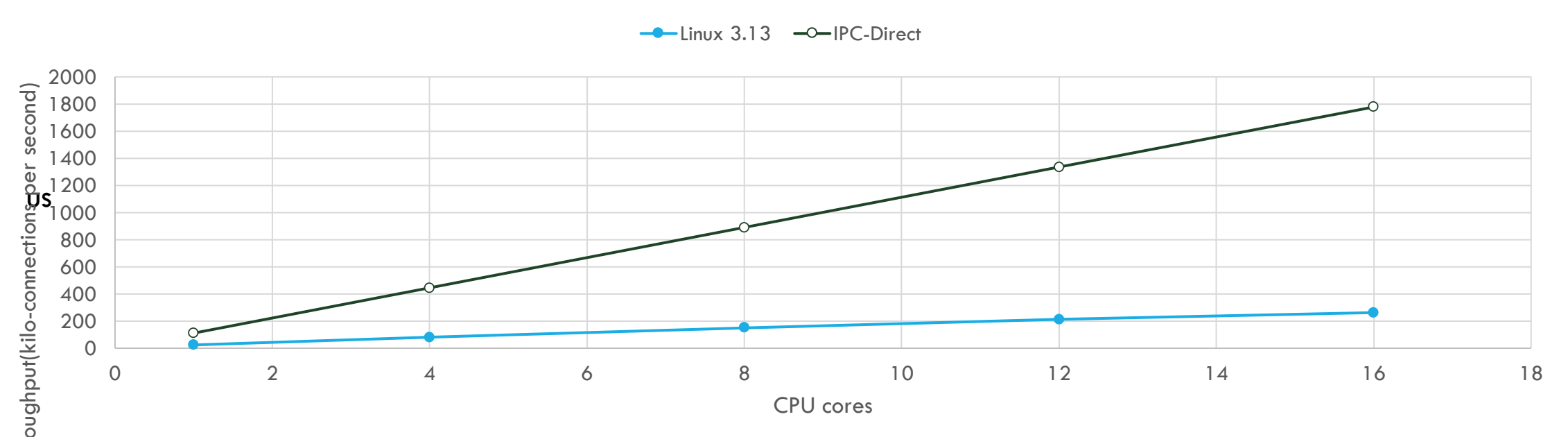Process 2: Event queue / Application — Data → Shared Buffer → Data

User Mode
Kernel Mode

- Using divided queue for each process and polling from monitor
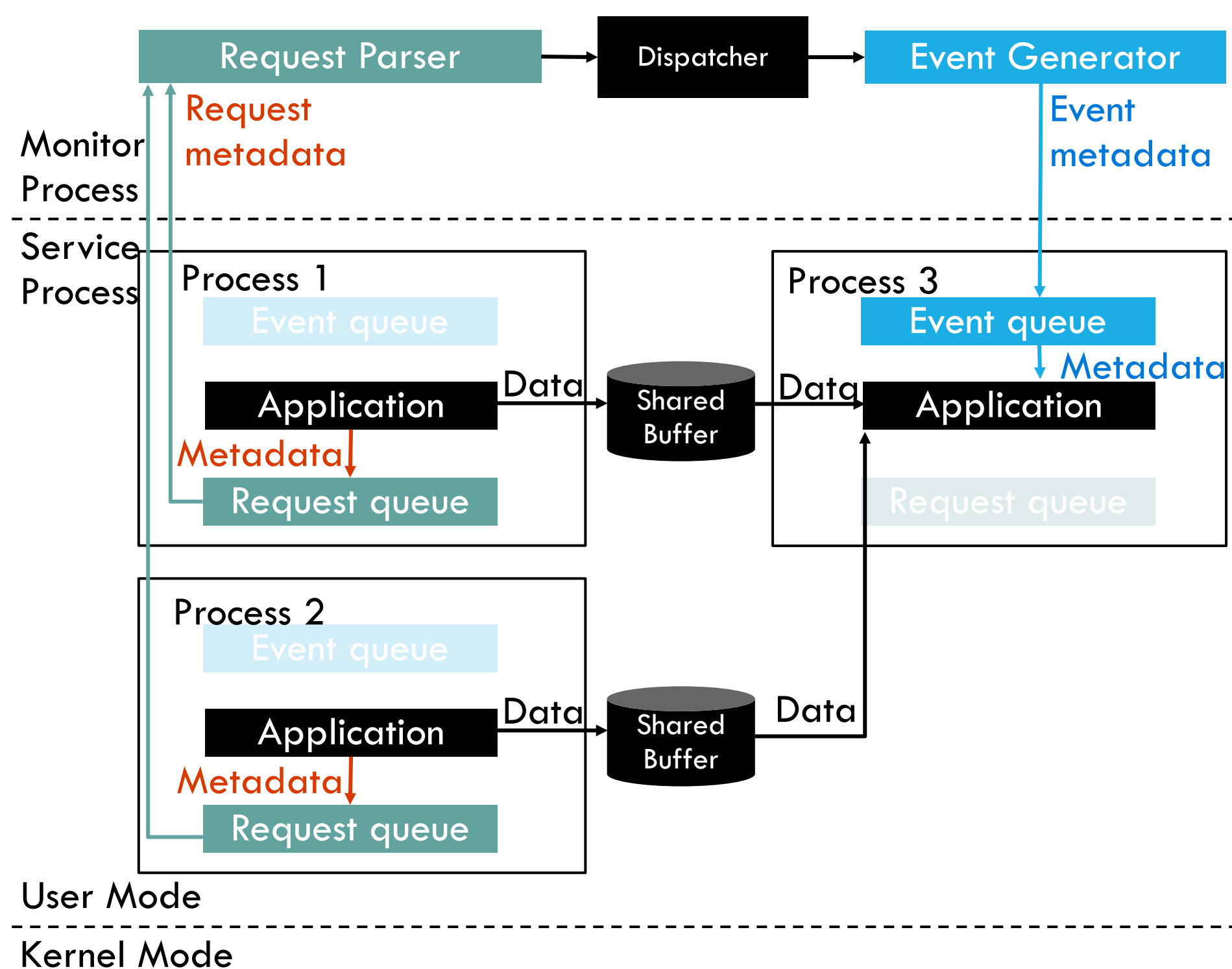- Using shared queue for all processes and using atomic operation to tackle contention

Microbenchmarks

Traditional inter-process communication is not scalable
■ Distributed coordination ■ Centralized coordination
(bar chart: 2 cores, 4 cores, 8 cores, 16 cores; normalized performance)

Lockless queue is better
Throughput with 64byte message(op/s)
■ Queue with lock ■ Queue without lock

Connection setup time for Nginx (Estimated)
— Linux 3.13 —○— IPC-Direct
Throughput(kilo-connections/per second) vs CPU cores

Our latency is better
■ TCP socket ■ Our design
RTT(US)
Low throughput: 11.04 / 0.33
High throughput: 12.46 / 0.33

Our throughput is better
■ TCP socket ■ IPC-Direct
Throughput for 64byte message(op/s)
3.44 / 26