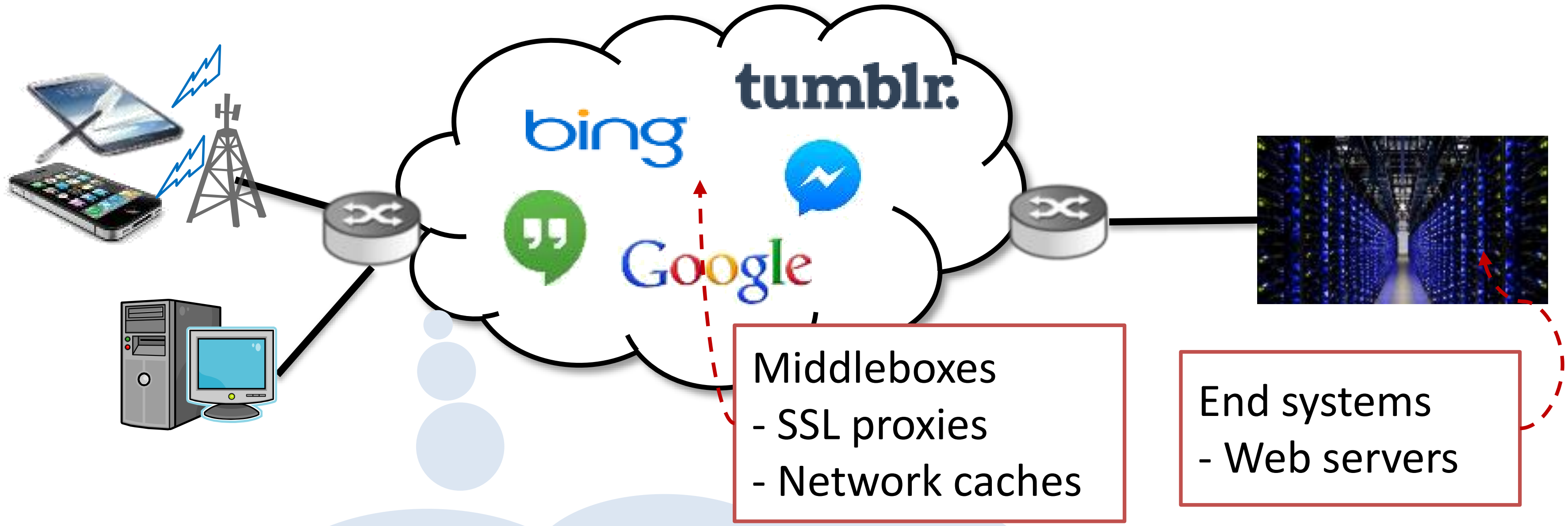
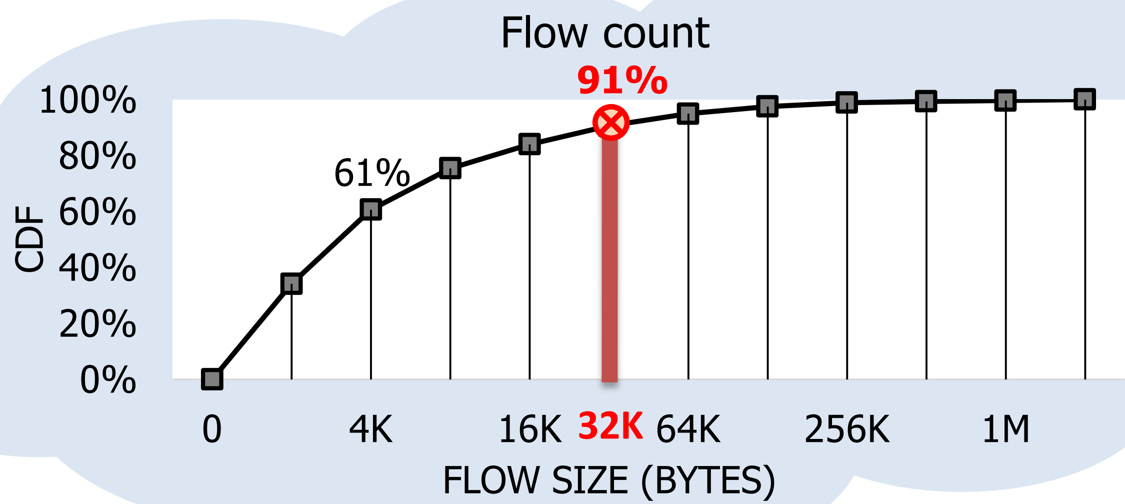
mTCP: A Highly Scalable User-level TCP Stack for Multicore Systems

EunYoung Jeong, **Shinae Woo**, Muhammad Jamshed, Haewon Jeong
Sunghwan Ihm*, Dongsu Han, and KyoungSoo Park

KAIST   * Princeton University

Needs for Handling Many Short Flows

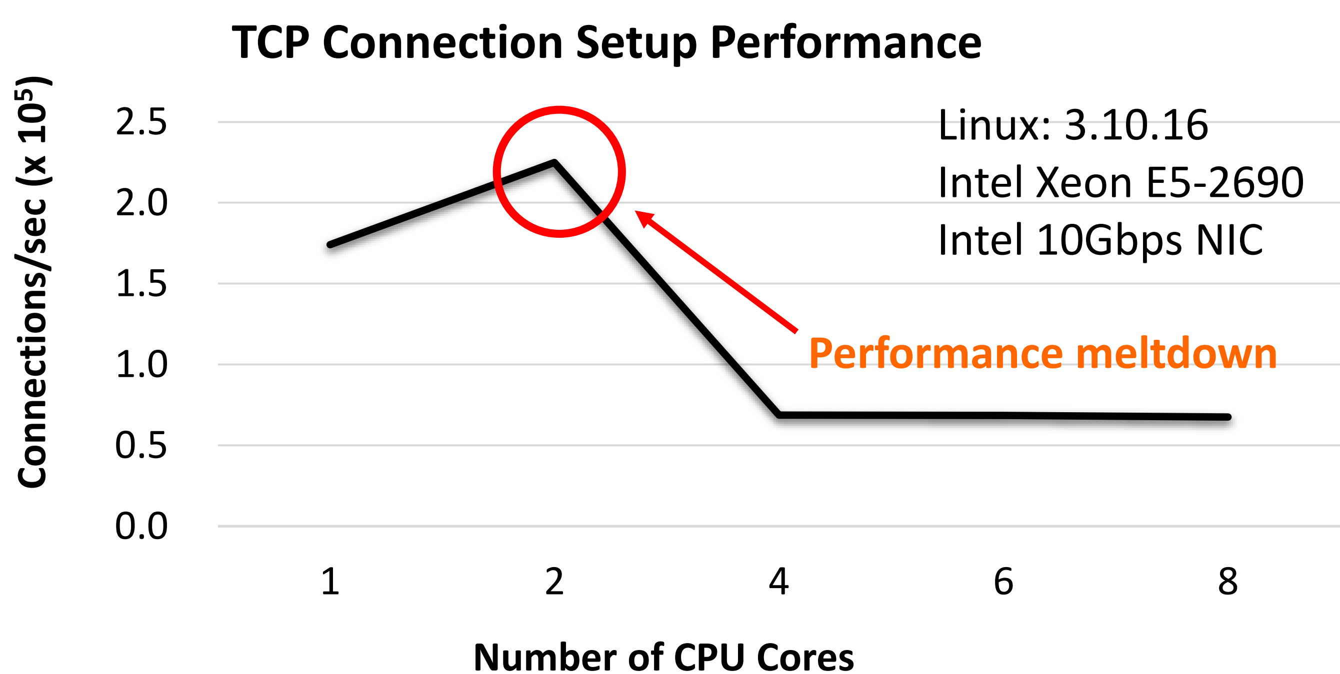




* **Commercial cellular traffic for 7 days**
Comparison of Caching Strategies in Modern Cellular Backhaul Networks,
MOBISYS 2013

Unsatisfactory Performance of Linux TCP

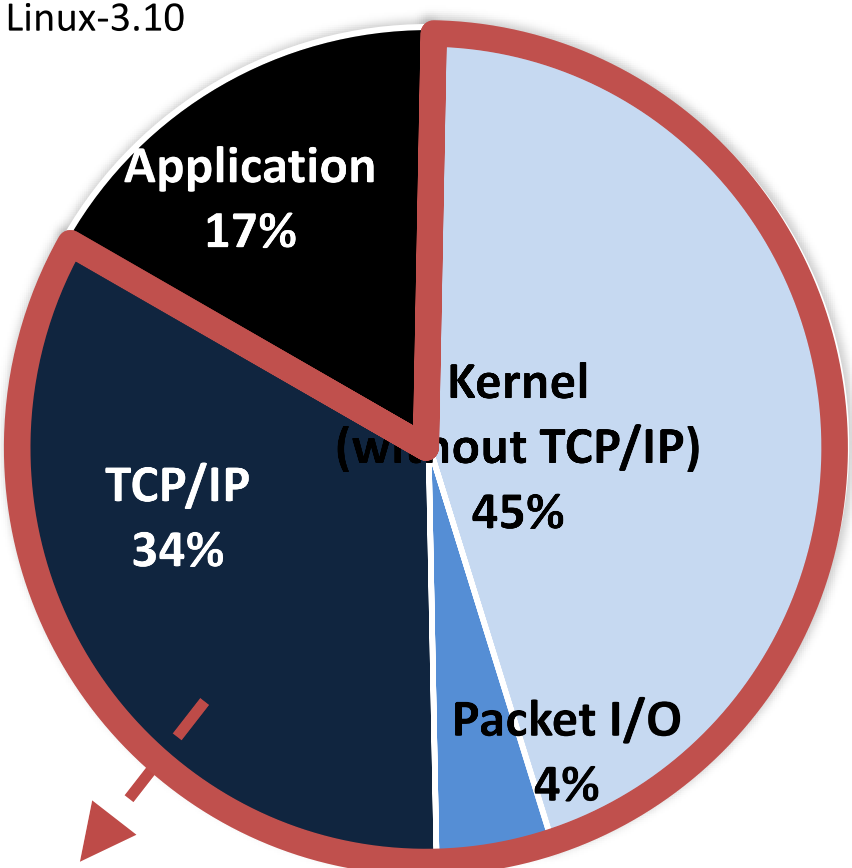
- Large flows: Easy to fill up 10 Gbps
- Small flows: Hard to fill up 10 Gbps regardless of # cores
  - Too many packets:
    14.88 Mpps for 64B packets in a 10 Gbps link
  - Kernel is not designed well for multicore systems



Kernel Uses the Most CPU Cycles

CPU Usage Breakdown of Web Server

Web server (Lighttpd) Serving a 64 byte file
Linux-3.10



83% of CPU usage spent inside kernel!

Performance bottlenecks

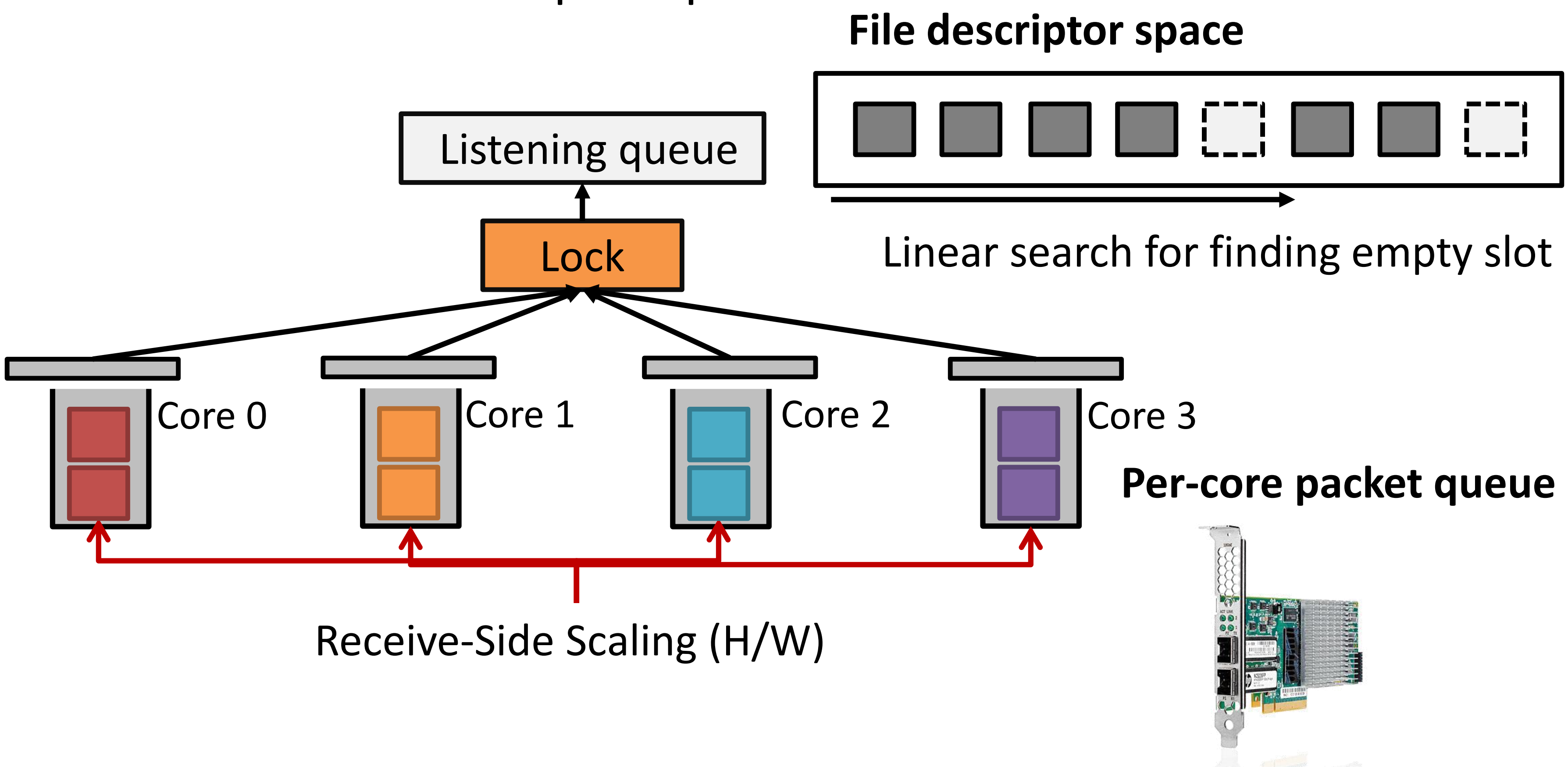
1. Shared resources
2. Broken locality
3. Per packet processing

Bottleneck removed by **mTCP**

1) Efficient use of CPU cycles for TCP/IP processing
→ 2.35x more CPU cycles for app
2) 3x ~ 25x better performance

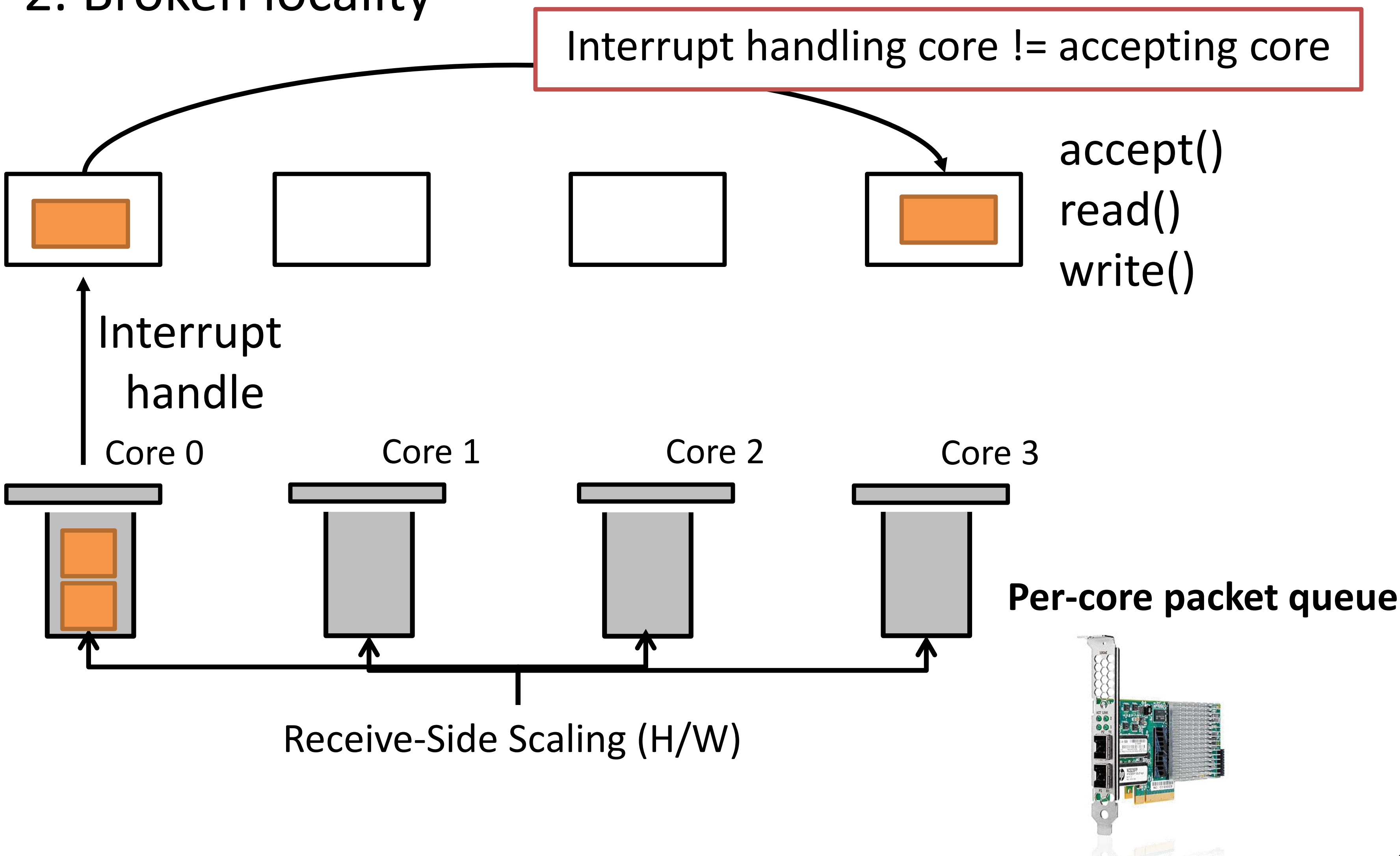
Inefficiencies in Kernel from Shared FD

1. Shared resources
   - Shared listening queue
   - Shared file descriptor space



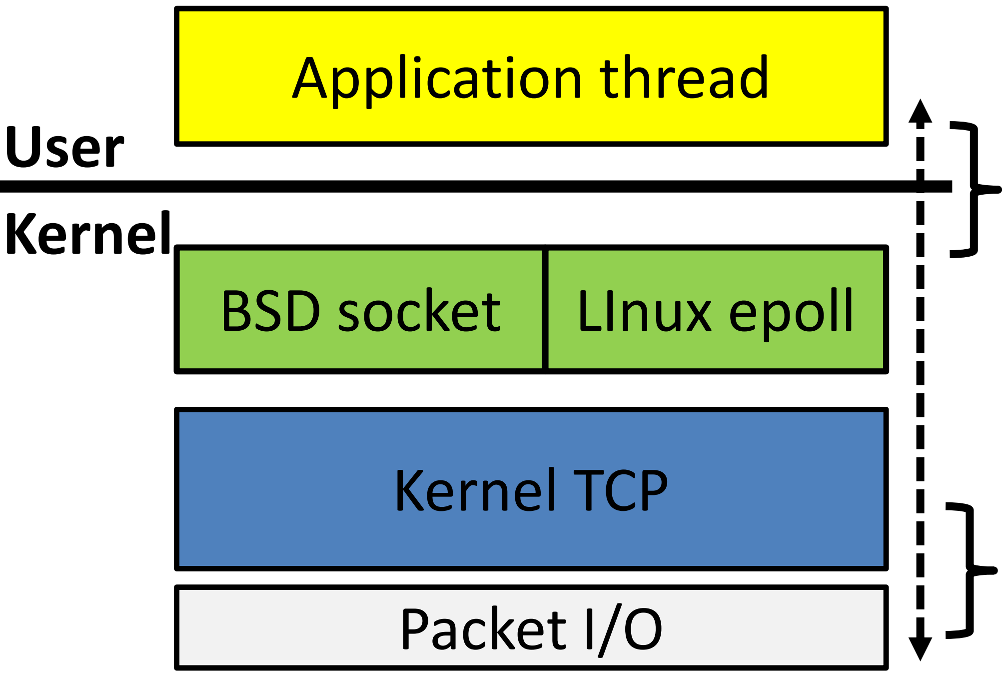
Inefficiencies in Kernel from Broken Locality

2. Broken locality



Inefficiencies in Kernel from Lack of Support for Batching

3. Per packet, per system call processing



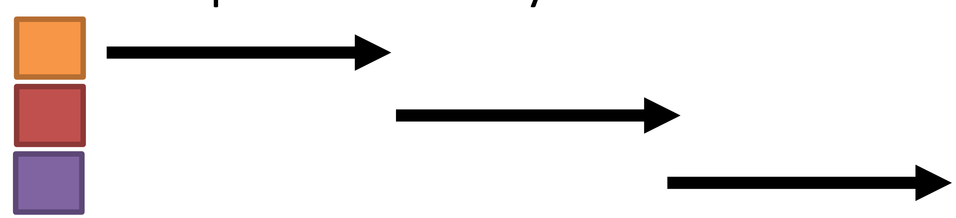
accept(), read(), write()

Inefficient per system call processing

Frequent mode switching
Cache pollution

Inefficient per packet processing

Per packet memory allocation



Previous Works on Solving Kernel Complexity

	Listening queue	Connection locality	App <-> TCP comm.	Packet I/O	API
Linux-2.6	Shared	No	Per system call	Per packet	BSD
Linux-3.9 SO_REUSEPORT	**Per-core**	No	Per system call	Per packet	BSD
Affinity-Accept	**Per-core**	**Yes**	Per system call	Per packet	BSD
MegaPipe	**Per-core**	**Yes**	**Batched system call**	Per packet	**custom**

→ Still, 78% of CPU cycles are used in kernel!

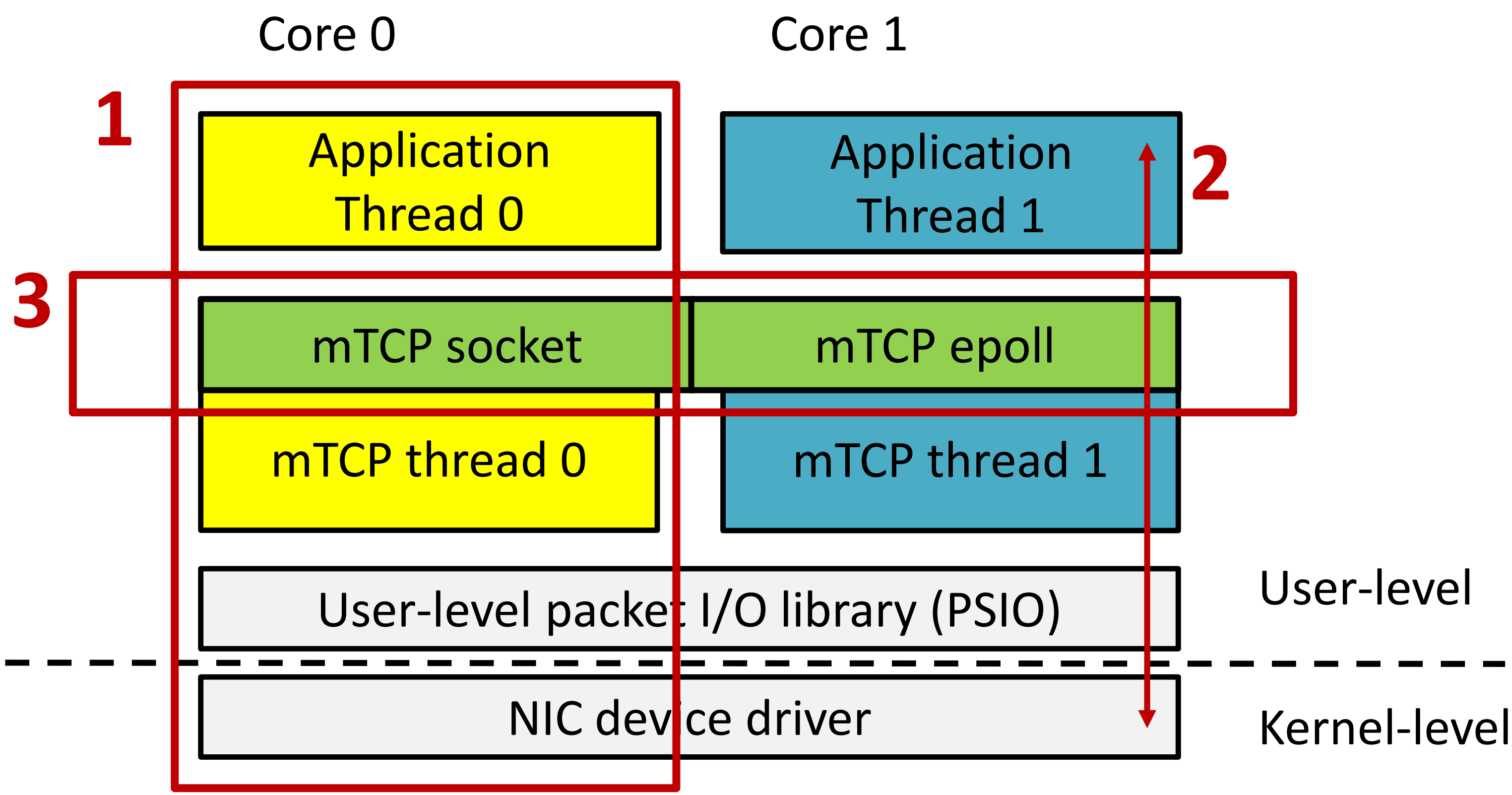
How much **performance improvement** can we get if we implement **a user-level TCP stack** with all optimizations?

Clean-slate Design Principles of mTCP

- mTCP: A high-performance user-level TCP designed for multicore systems
- Clean-slate approach to divorce kernel's complexity

Problems		Our contributions
1. Shared resources 2. Broken locality	➡	Each core works independently – No shared resources – Resources affinity
3. Lack of support for batching	➡	Batching from flow processing from packet I/O to user API
	✚	Easily portable APIs for compatibility

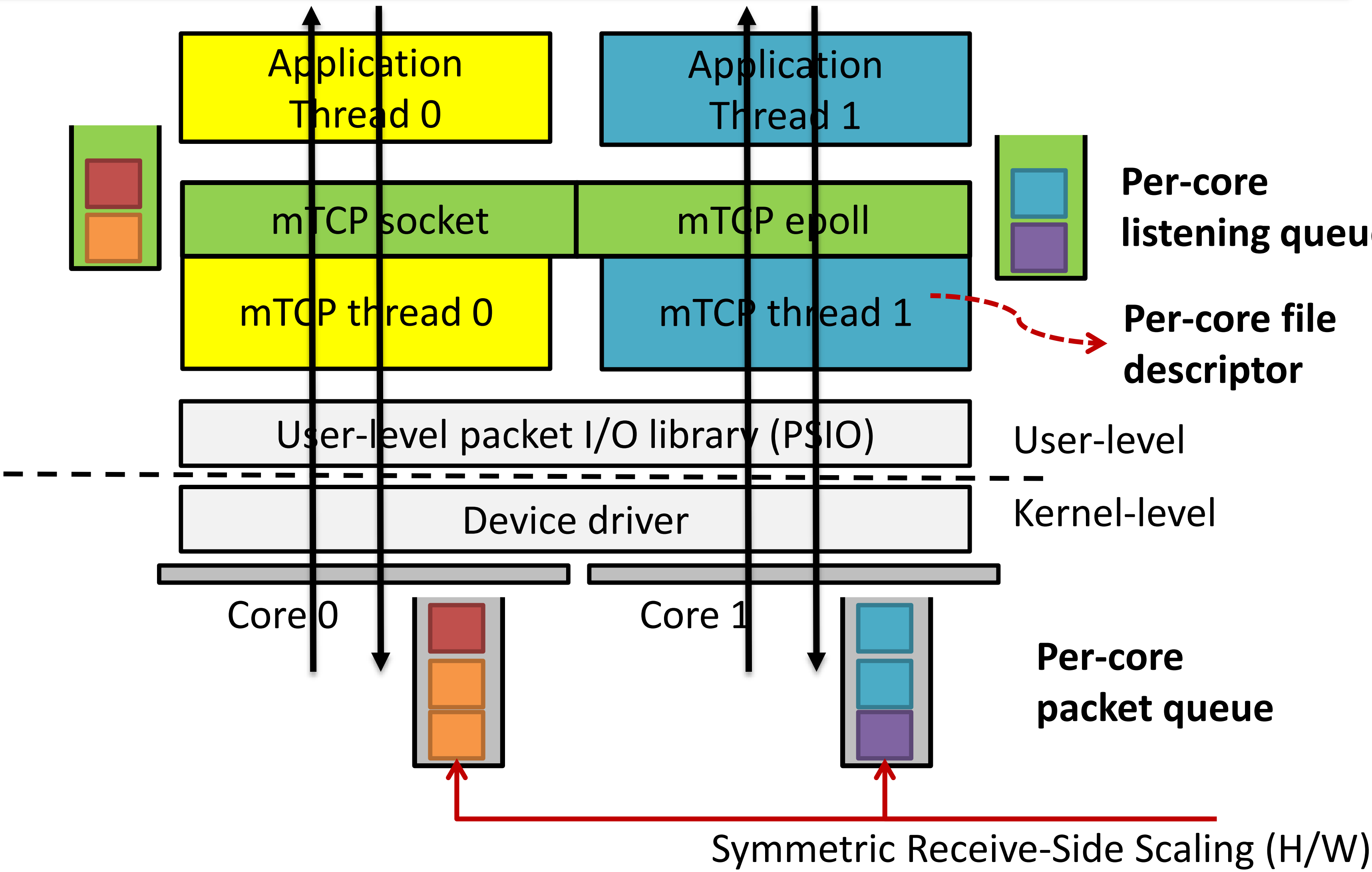
Overview of mTCP Architecture



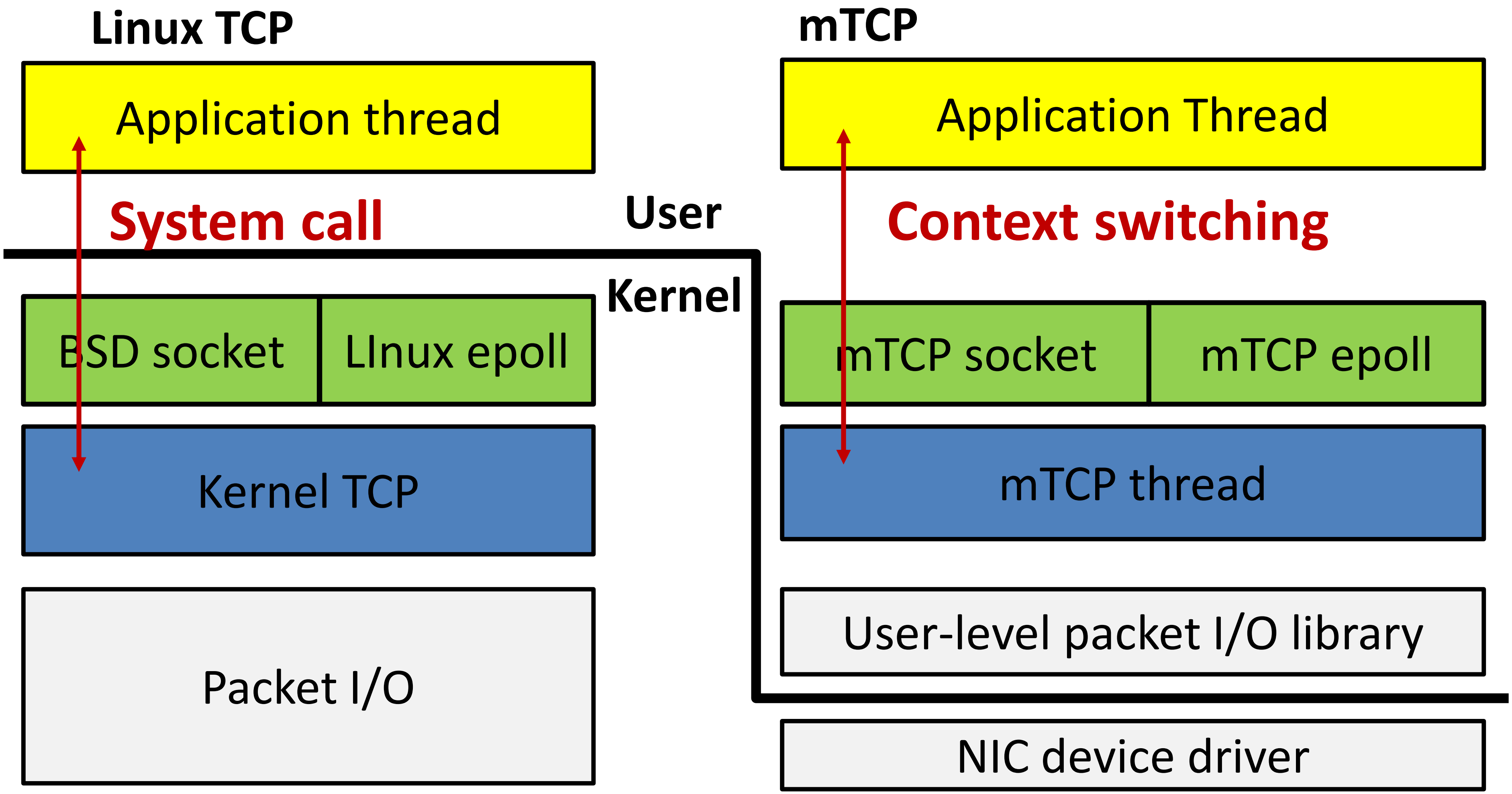
1. Thread model: Pairwise, per-core threading
2. Batching from packet I/O to application
3. mTCP API: Easily portable API (BSD-like)

- [SIGCOMM'10] PacketShader: A GPU-accelerated software router, http://shader.kaist.edu/packetshader/io_engine/index.html

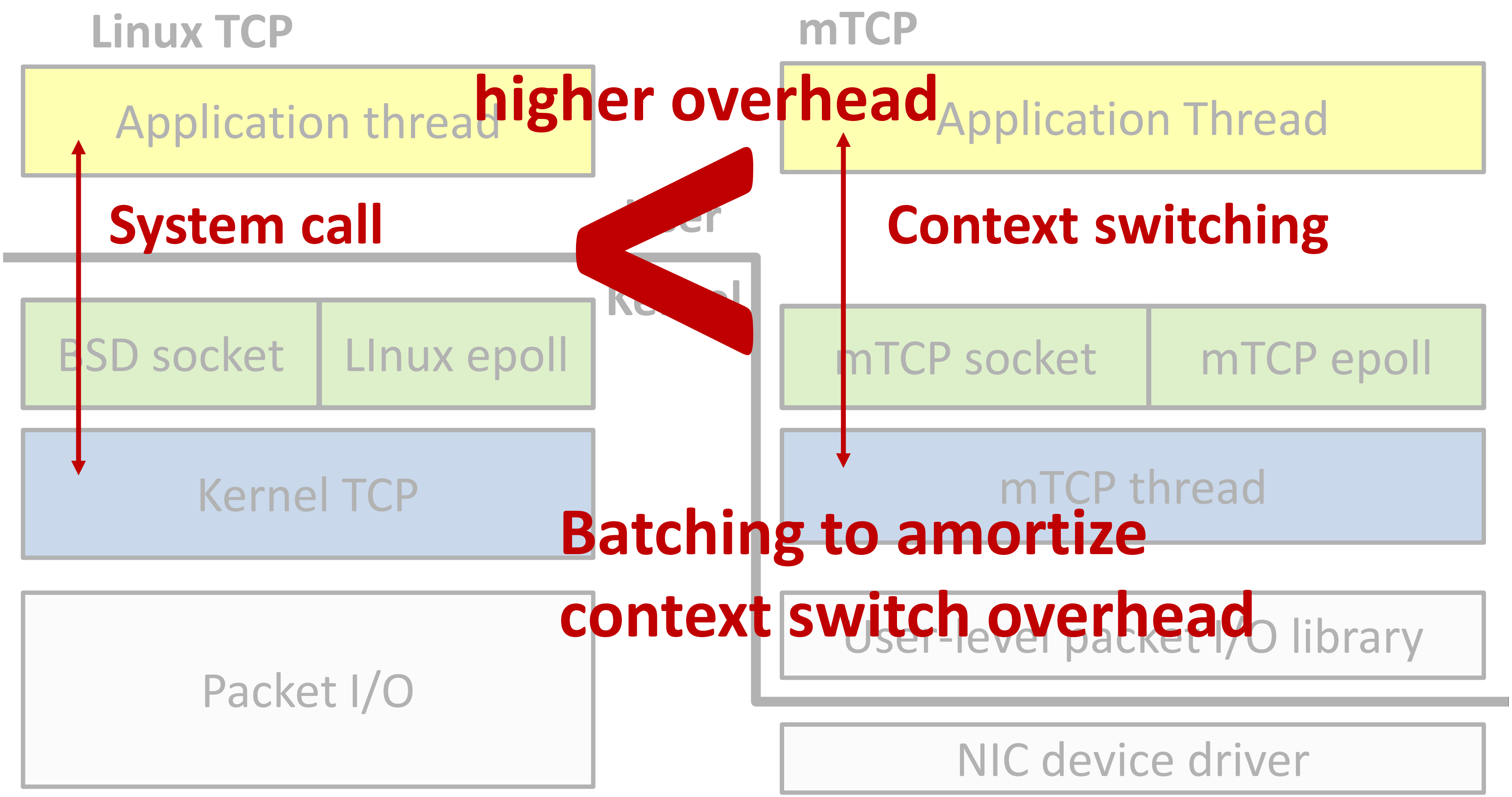
1. Thread Model: Pairwise, Per-core Threading



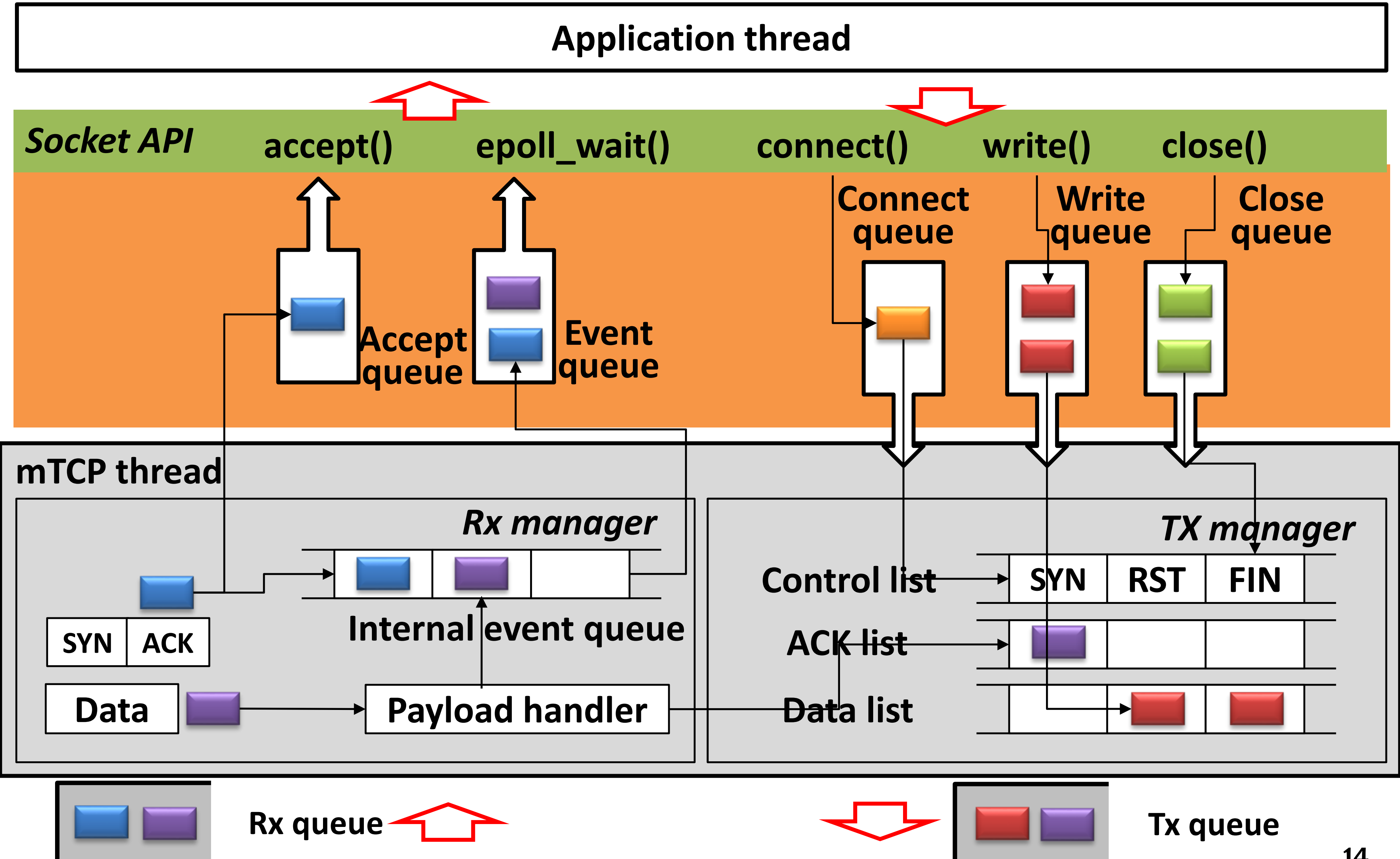
From System Call to Context Switching



From System Call to Context Switching



2. Batching process in mTCP thread



3. mTCP API: Similar to BSD Socket API

- Two goals: Easy porting + keeping popular event model
- Ease of porting
  - Just attach "mtcp_" to BSD socket API
  - **socket() → mtcp_socket()**, **accept() → mtcp_accept()**, etc.
- Event notification: Readiness model using **epoll()**
- Porting existing applications
  - Mostly less than 100 lines of code change

Application	Description	Modified lines / Total lines
Lighttpd	An event-driven web server	**65** / 40K
ApacheBench	A webserver performance benchmark tool	**29** / 66K
SSLShader	A GPU-accelerated SSL proxy [NSDI '11]	**43** / 6,618
WebReplay	A web log replayer	**81** / 3,366

Optimizations for Performance

- Lock-free data structures
- Cache-friendly data structure
- Hugepages for preventing TLB missing
- Efficient TCP timer management
- Priority-based packet queuing
- Lightweight connection setup
- ……

Please refer to our paper ☺

mTCP Implementation

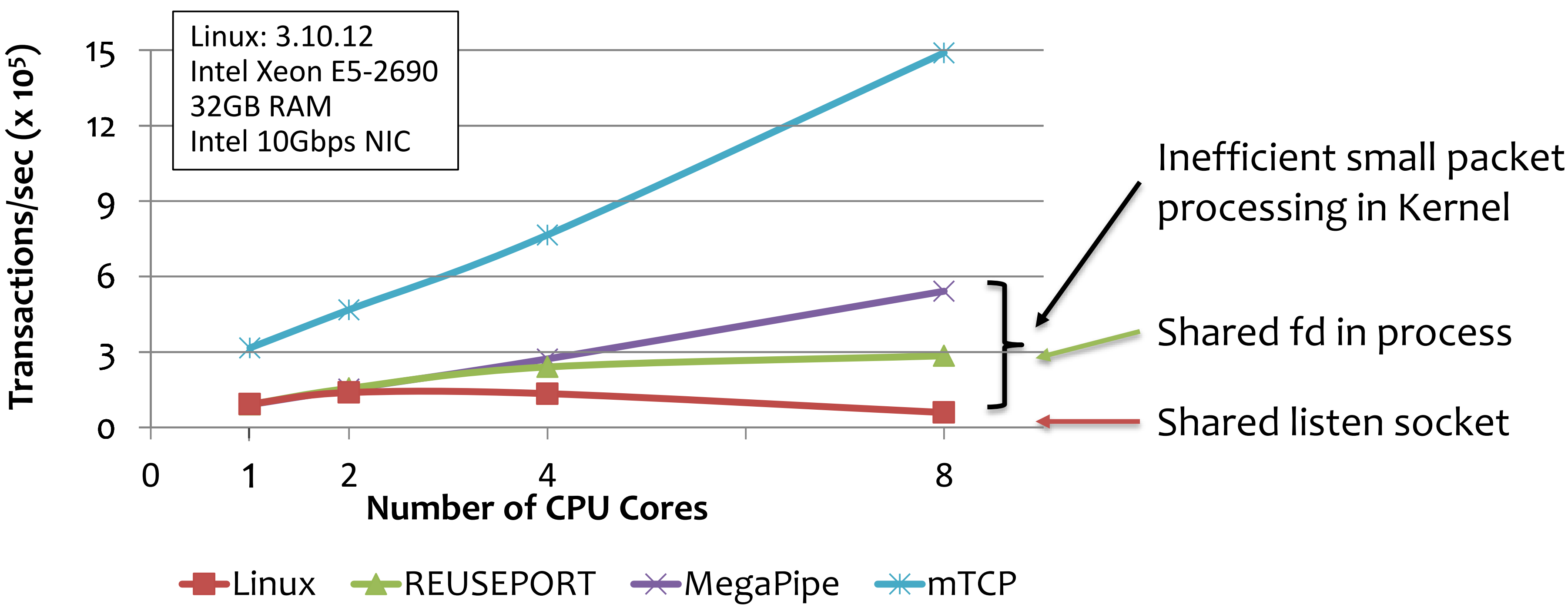
- 11,473 lines (C code)
  - Packet I/O, TCP flow management, User-level socket API, Event system library
- 552 lines to patch the PSIO library
  - Support event-driven packet I/O: ps_select()
- TCP implementation
  - Follows RFC793
  - Congestion control algorithm: NewReno
- Passing correctness test and stress test with Linux TCP stack

Evaluation

- Scalability with multicore
  - Comparison of performance of multicore with previous solutions
- Performance improvement on ported applications
  - Web Server (Lighttpd)
    – Performance under the real workload
  - SSL proxy (SSL Shader, NSDI 11)
    – TCP bottlenecked application

Multicore Scalability

- 64B ping/pong messages per connection
- Heavy connection overhead, small packet processing overhead
- 25x Linux, 5x SO_REUSEPORT*[LINUX3.9], 3x MegaPipe*[OSDI'12]



* [LINUX3.9] https://lwn.net/Articles/542629/
* [OSDI'12] MegaPipe: A New Programming Interface for Scalable Network I/O, Berkeley

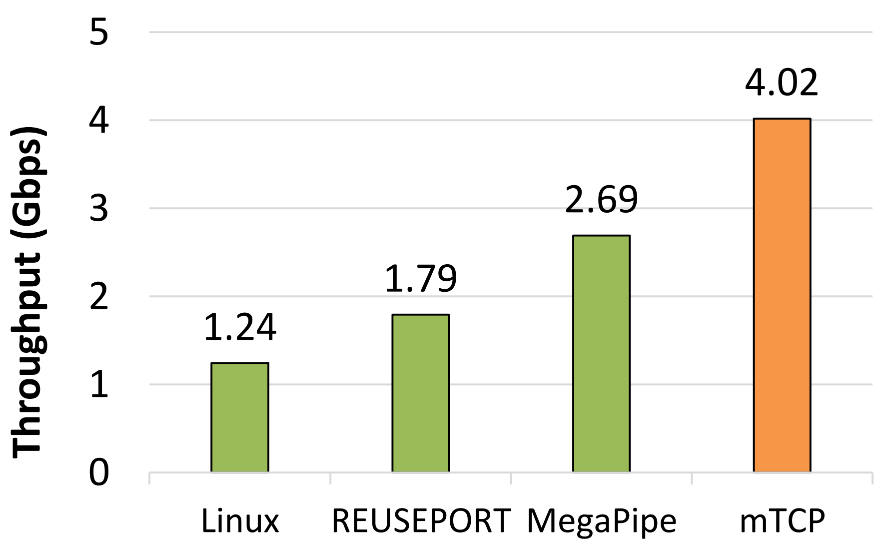
Performance Improvement on Ported Applications

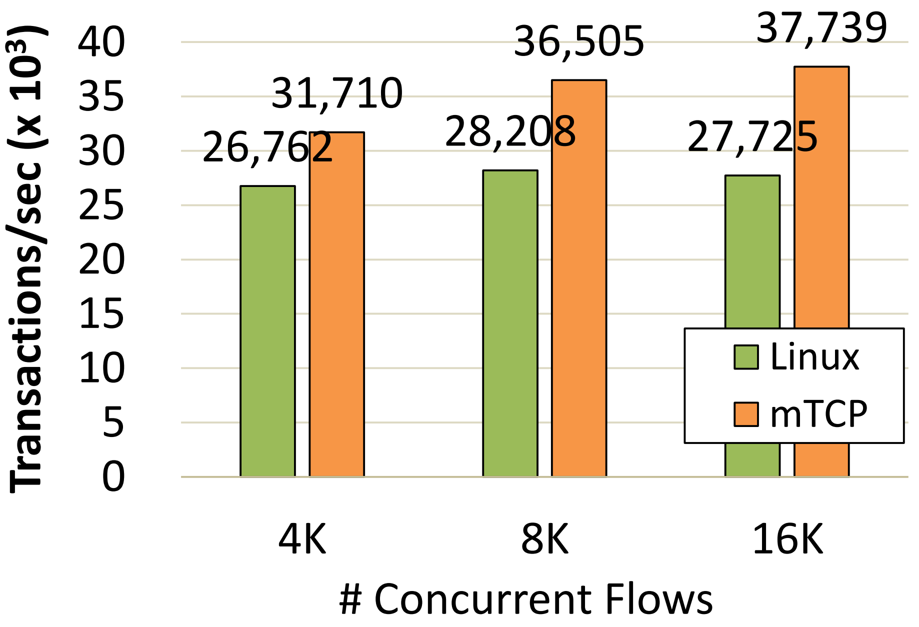
Web Server (Lighttpd)

- Real traffic workload: Static file workload from SpecWeb2009 set
- **3.2x** faster than Linux
- **1.5x** faster than MegaPipe

SSL Proxy (SSLShader)

- Performance Bottleneck in TCP
- Cipher suite
  1024-bit RSA, 128-bit AES, HMAC-SHA1
- Download 1-byte object via HTTPS





Conclusion

- mTCP: A high-performing user-level TCP stack for multicore systems
  - Clean-slate user-level design to overcome inefficiency in kernel
- Make full use of extreme parallelism & batch processing
  - Per-core resource management
  - Lock-free data structures & cache-aware threading
  - Eliminate system call overhead
  - Reduce context switch cost by event batching
- Achieve high performance scalability
  - Small message transactions: **3x to 25x better**
  - Existing applications: 33% (SSLShader) to 320% (lighttpd)

Thank You

Source code is available at

http://shader.kaist.edu/mtcp/

https://github.com/eunyoung14/mtcp