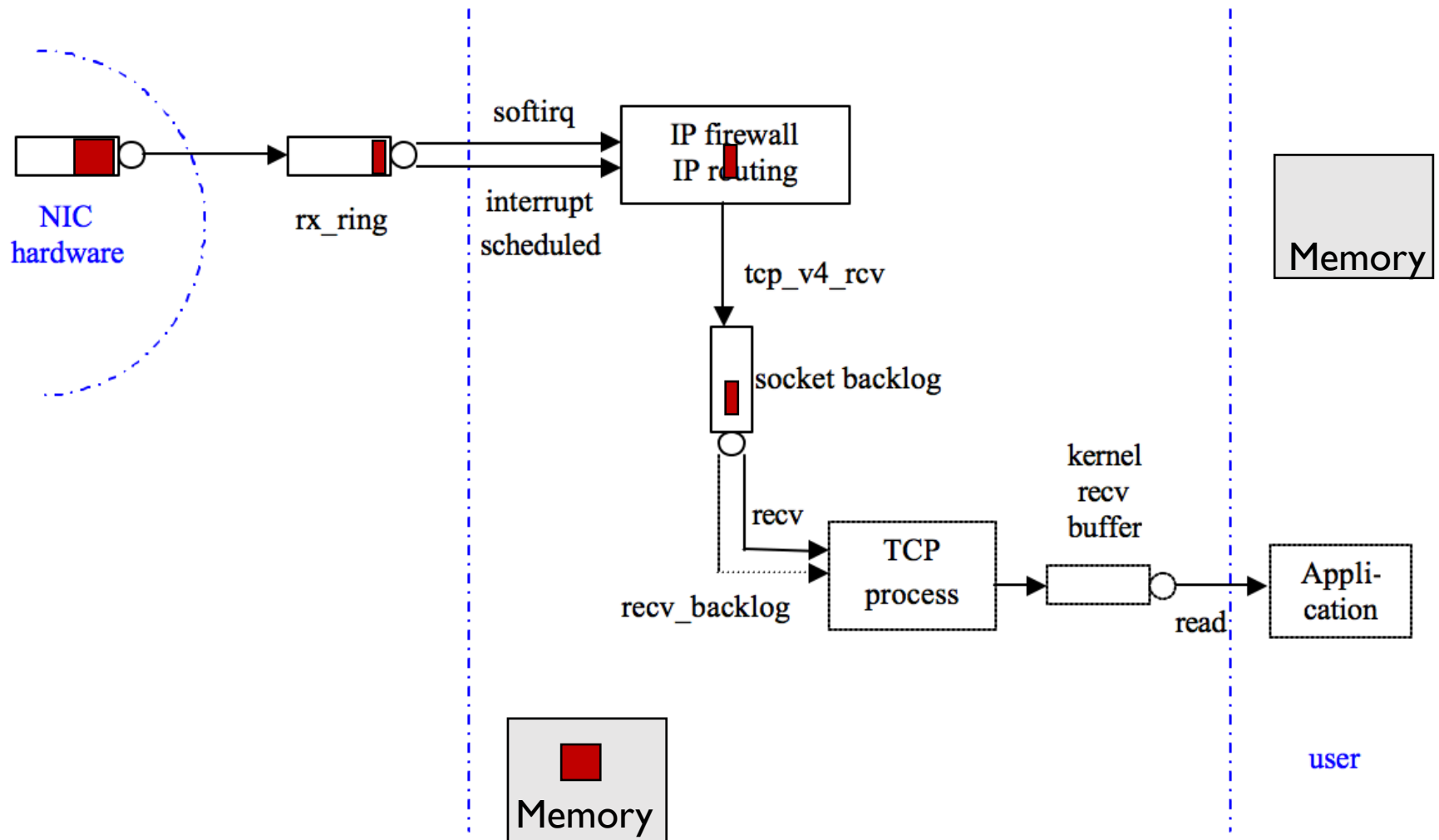


High Performance Network Stack

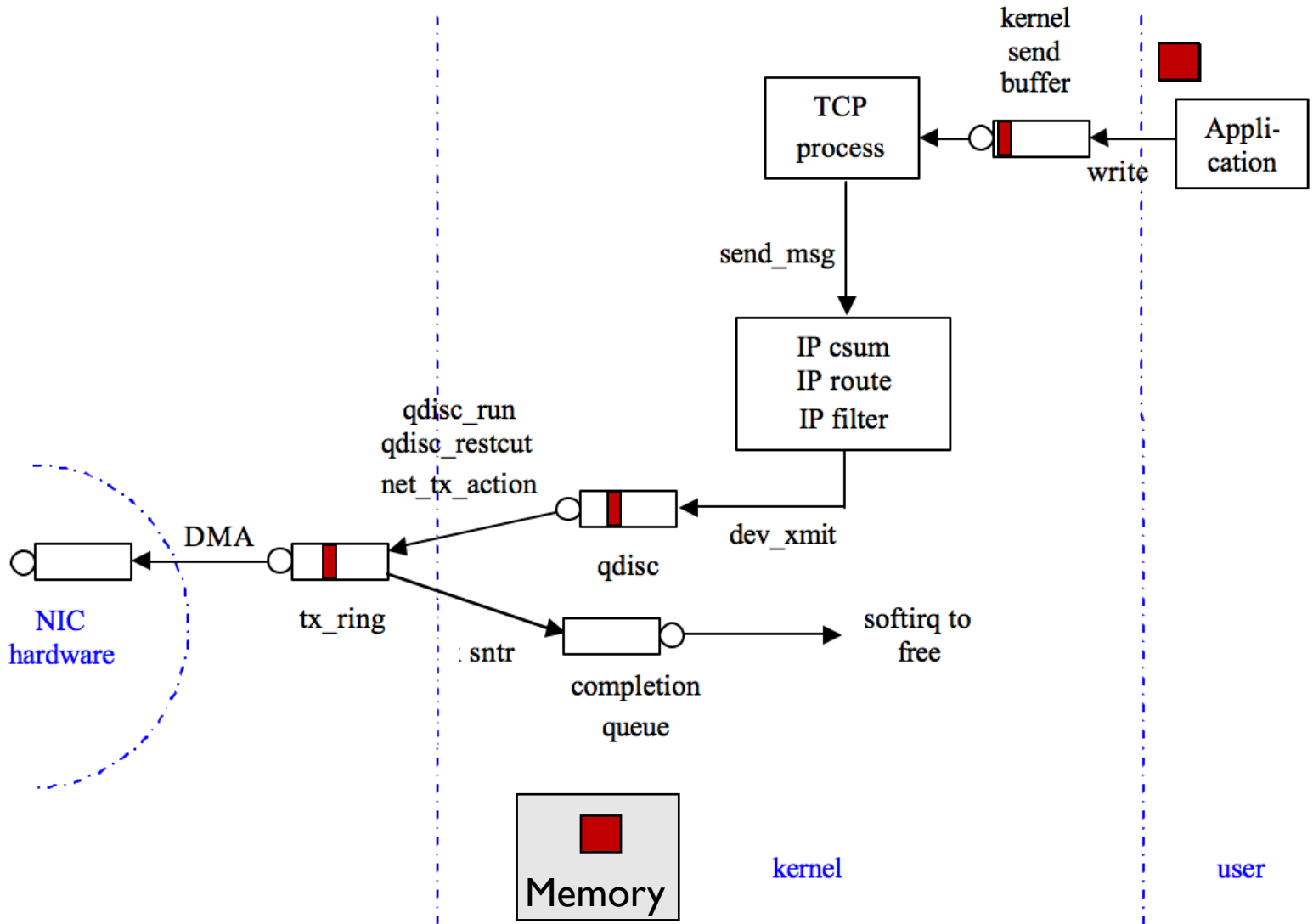
ECE/CS598HPN

Radhika Mittal

Rx Processing in the kernel



Tx Processing in the kernel



Various sources of performance overheads

MegaPipe: A New Programming Interface for Scalable Network I/O

Sangjin Han, Scott Marshal,
Byung-Gon Chun, Sylvia Ratnasamy

OSDI'12

Content borrowed from Sangjin's OSDI talk

Two Types of Network Workloads

- **Bulk Transfer**
 - Large files (HDFS)

- **Message-oriented**
 - Short connections or small messages (HTTP, RPCs, DB, key-value stores, etc)

Two Types of Network Workloads

- **Bulk Transfer**

- Large files (HDFS)
- A half CPU core can saturate 10Gbps link

- **Message-oriented**

- Short connections or small messages (HTTP, RPCs, DB, key-value stores, etc)
- CPU-intensive


BSD Socket API Performance Issues

```
n_events = epoll_wait(...); // wait for I/O readiness
for (...) {
    ...
    new_fd = accept(listen_fd); // new connection
    ...
    bytes = recv(fd2, buf, 4096); // new data for fd2
```

- Issues with message-oriented workloads
 - System call overhead


BSD Socket API Performance Issues

```
n_events = epoll_wait(...); // wait for I/O readiness
for (...) {
    ...
    new_fd = accept(listen_fd); // new connection
    ...
    bytes = recv(fd2, buf, 4096); // new data for fd2
}
```

- Issues with message-oriented workloads
 - System call overhead
 - Shared listening socket 

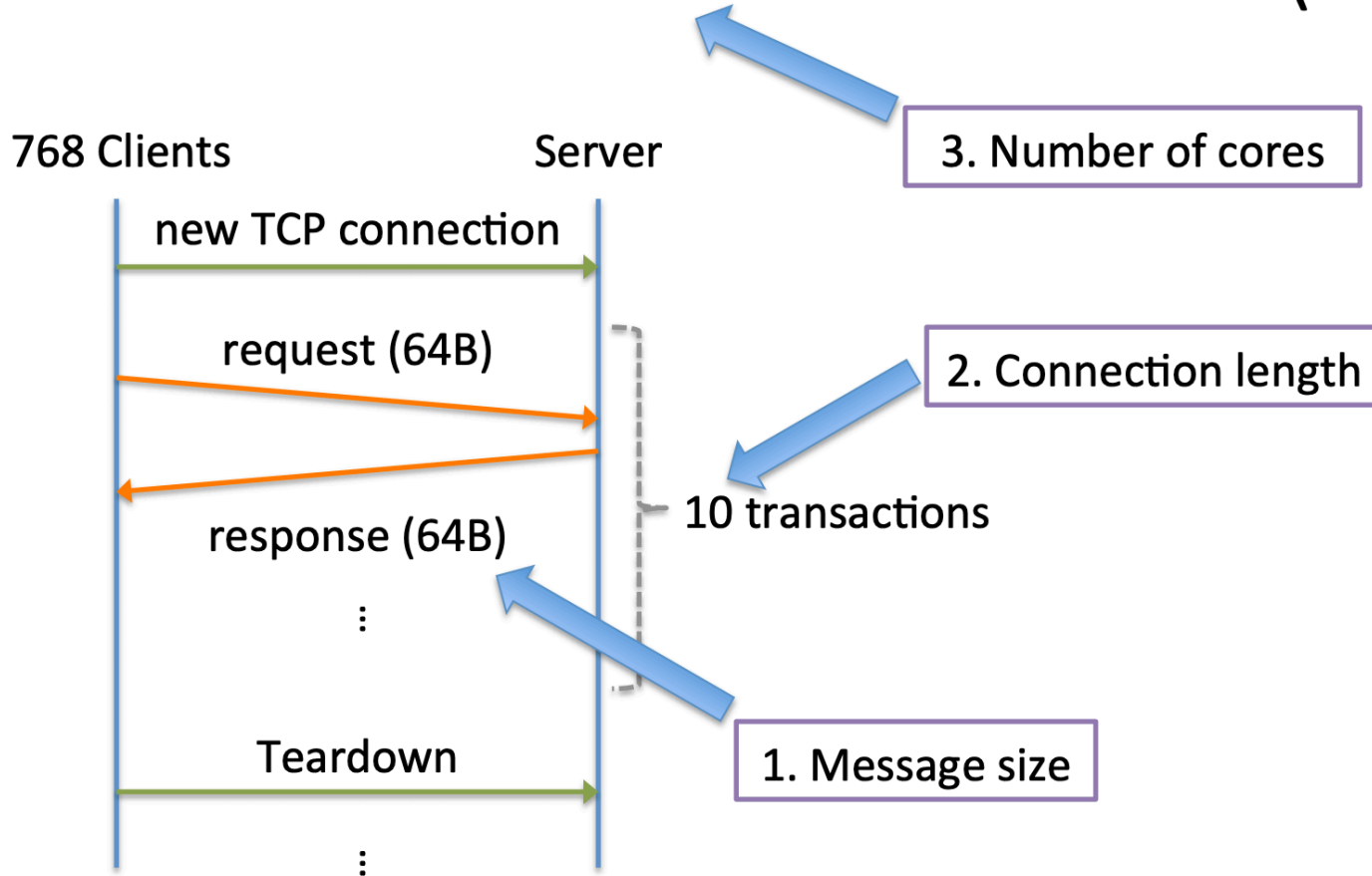
BSD Socket API Performance Issues

```
n_events = epoll_wait(...); // wait for I/O readiness
for (...) {
    ...
    new_fd = accept(listen_fd); // new connection
    ...
    bytes = recv(fd2, buf, 4096); // new data for fd2
}
```

- Issues with message-oriented workloads
 - System call overhead
 - Shared listening socket
 - File abstraction overhead 

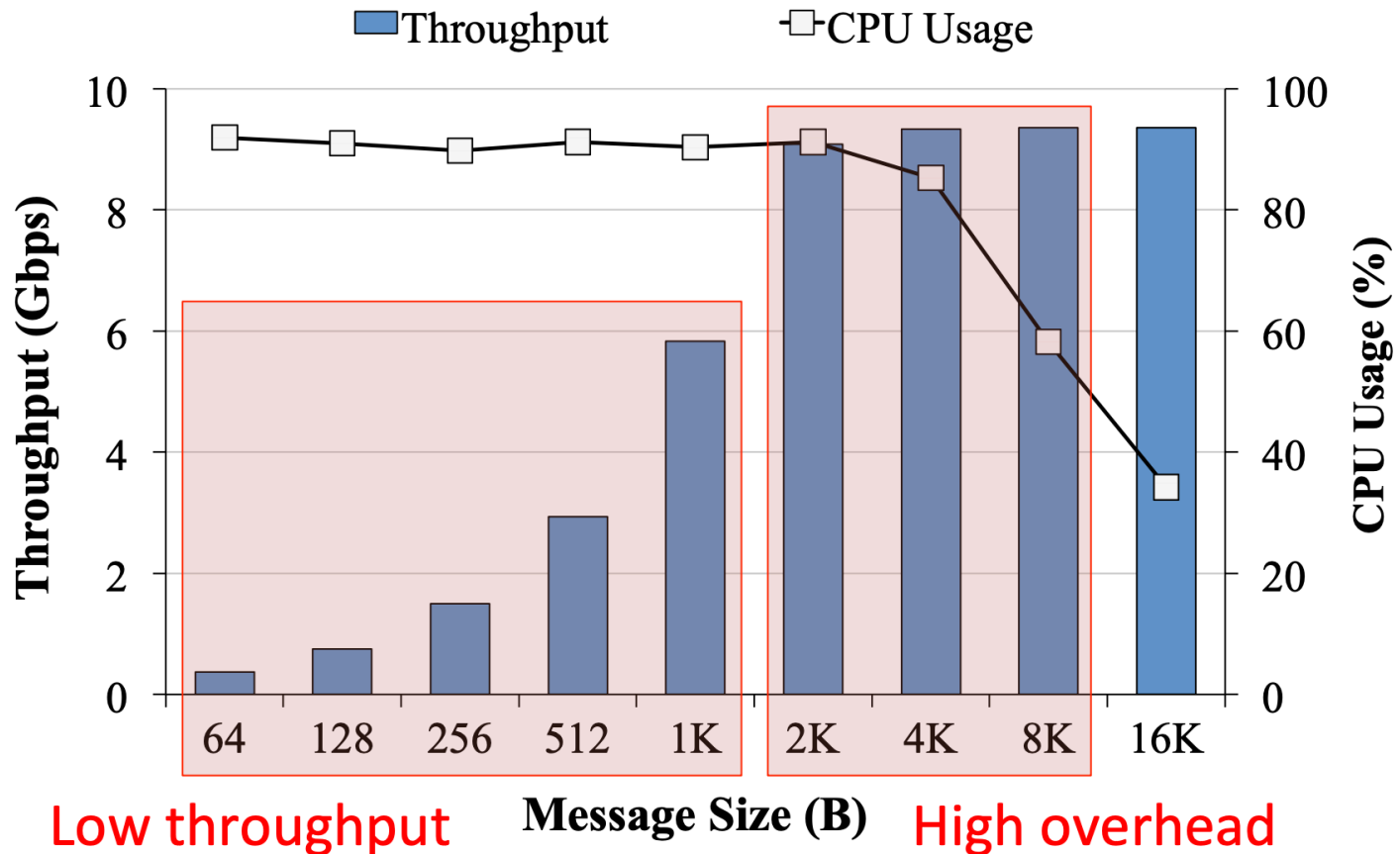
Microbenchmarks: how bad?

RPC-like test on an 8-core Linux server (with epoll)



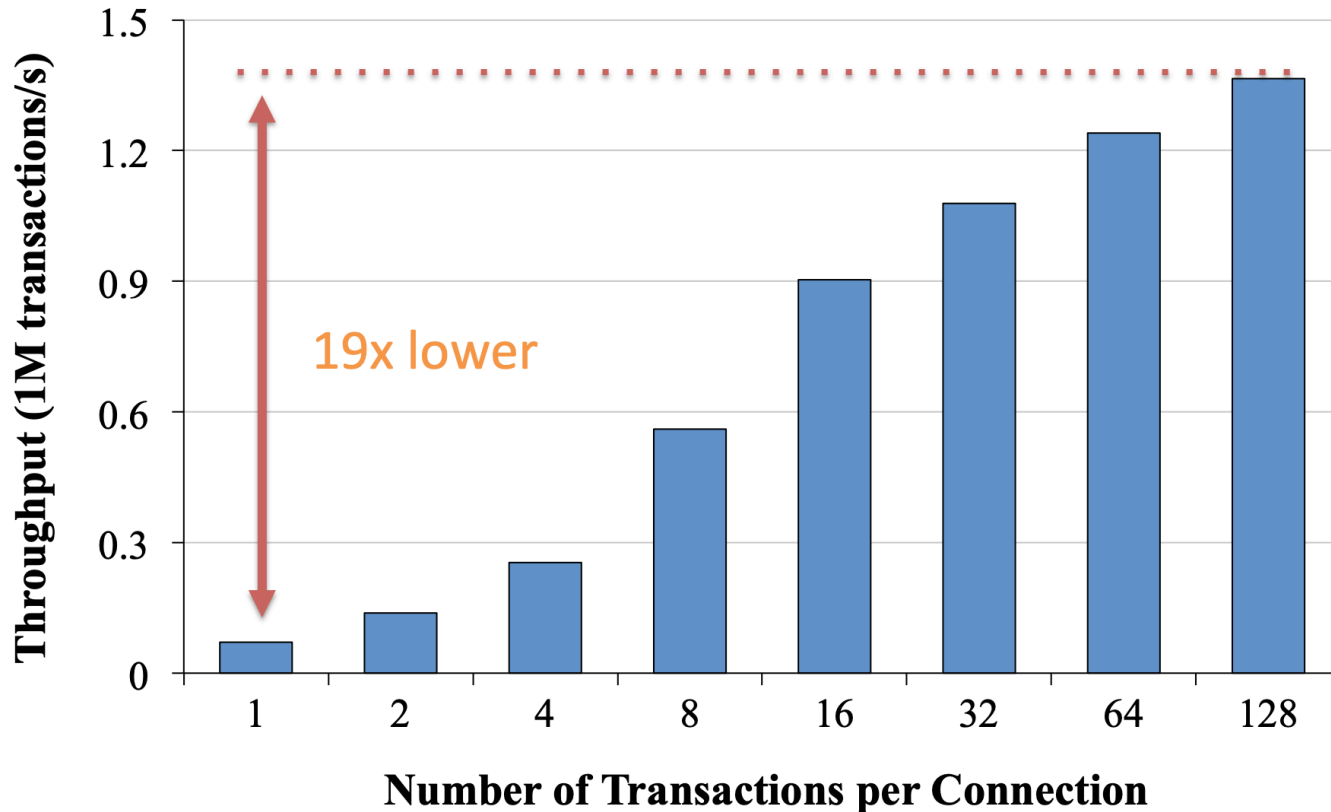
Microbenchmarks: how bad?

1. Small Messages Are Bad



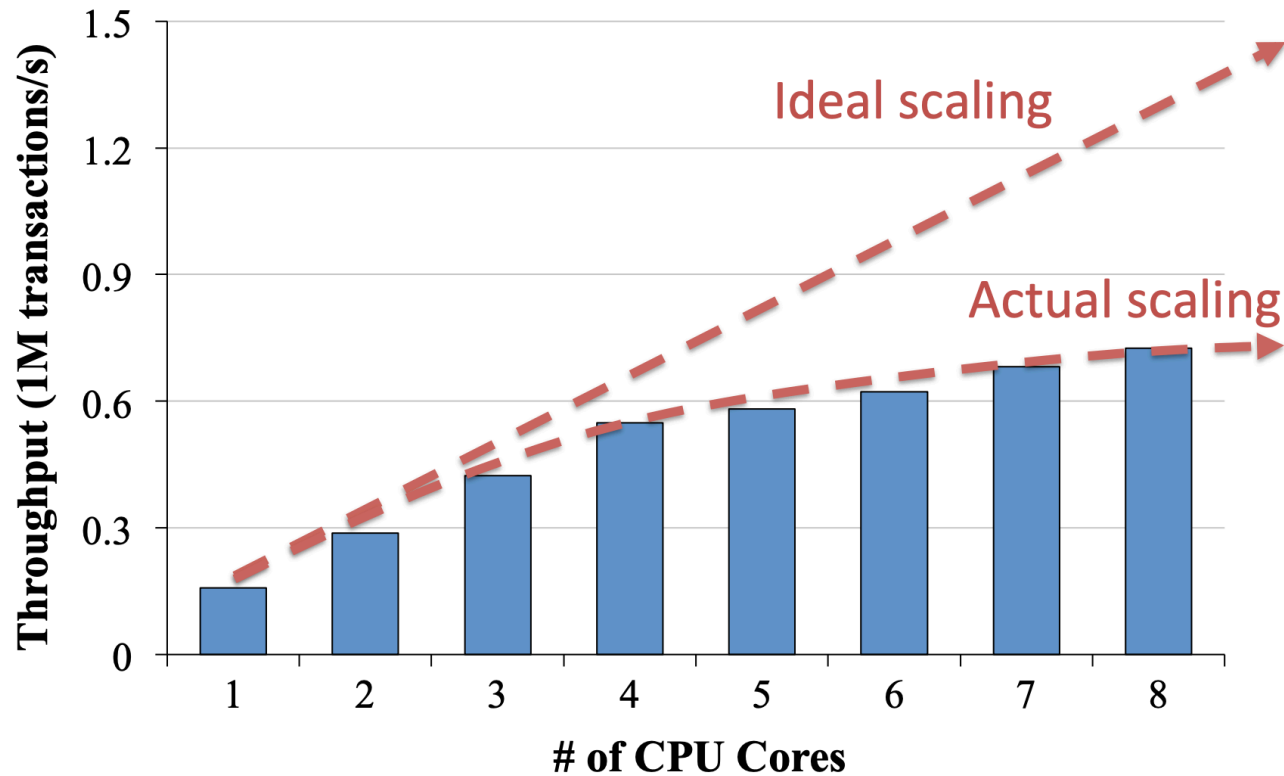
Microbenchmarks: how bad?

2. Short Connections Are Bad



Microbenchmarks: how bad?

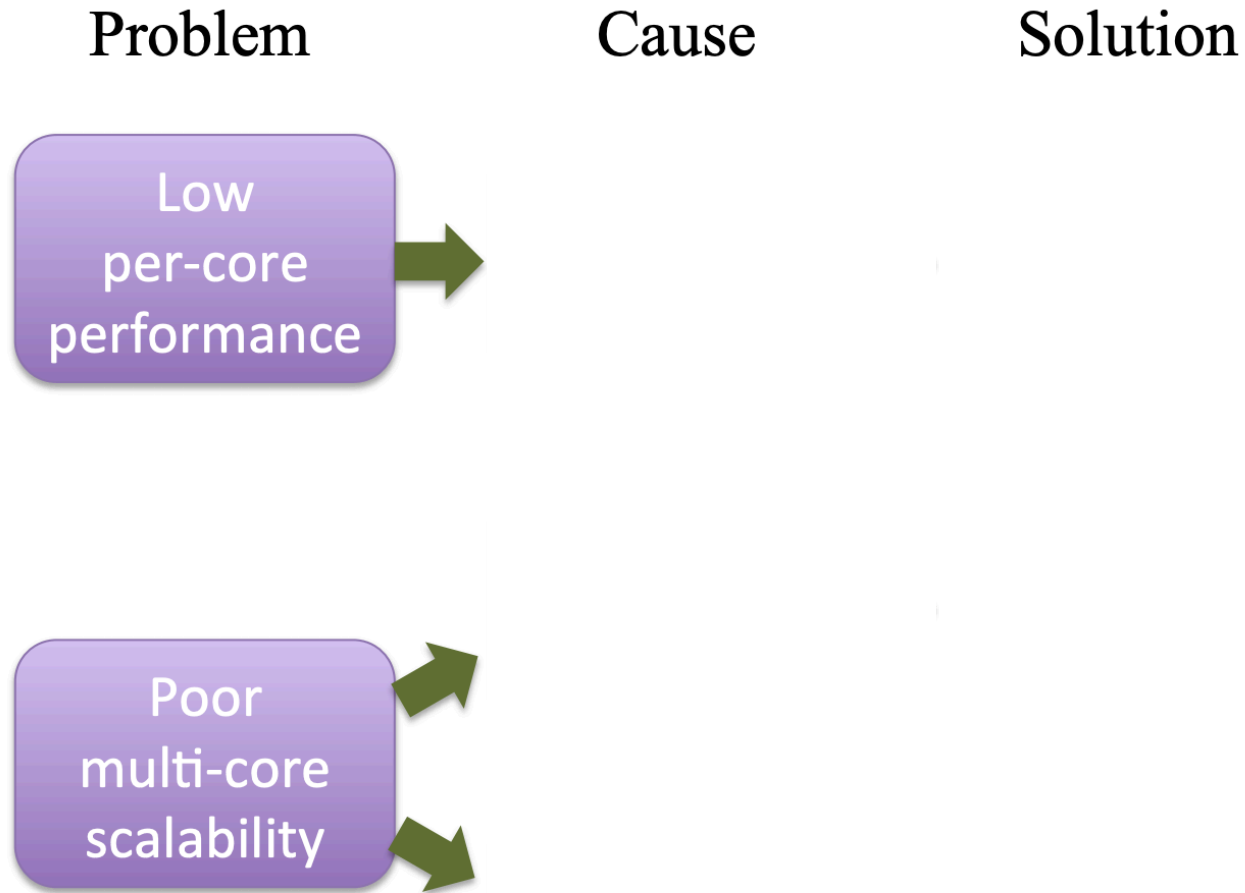
3. Multi-Core Will Not Help (Much)



MegaPipe Design

Focus: low-overhead and multi-core scalability.

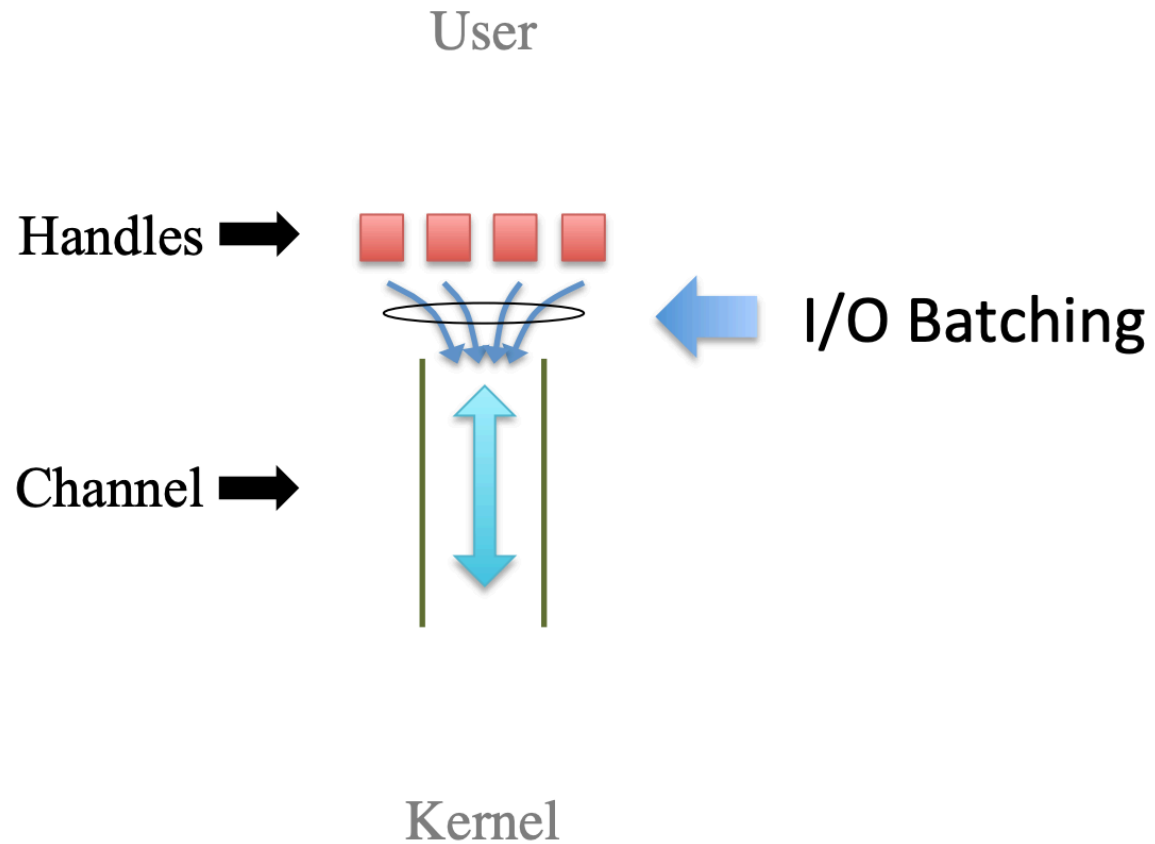
MegaPipe: Overview



Key Primitives

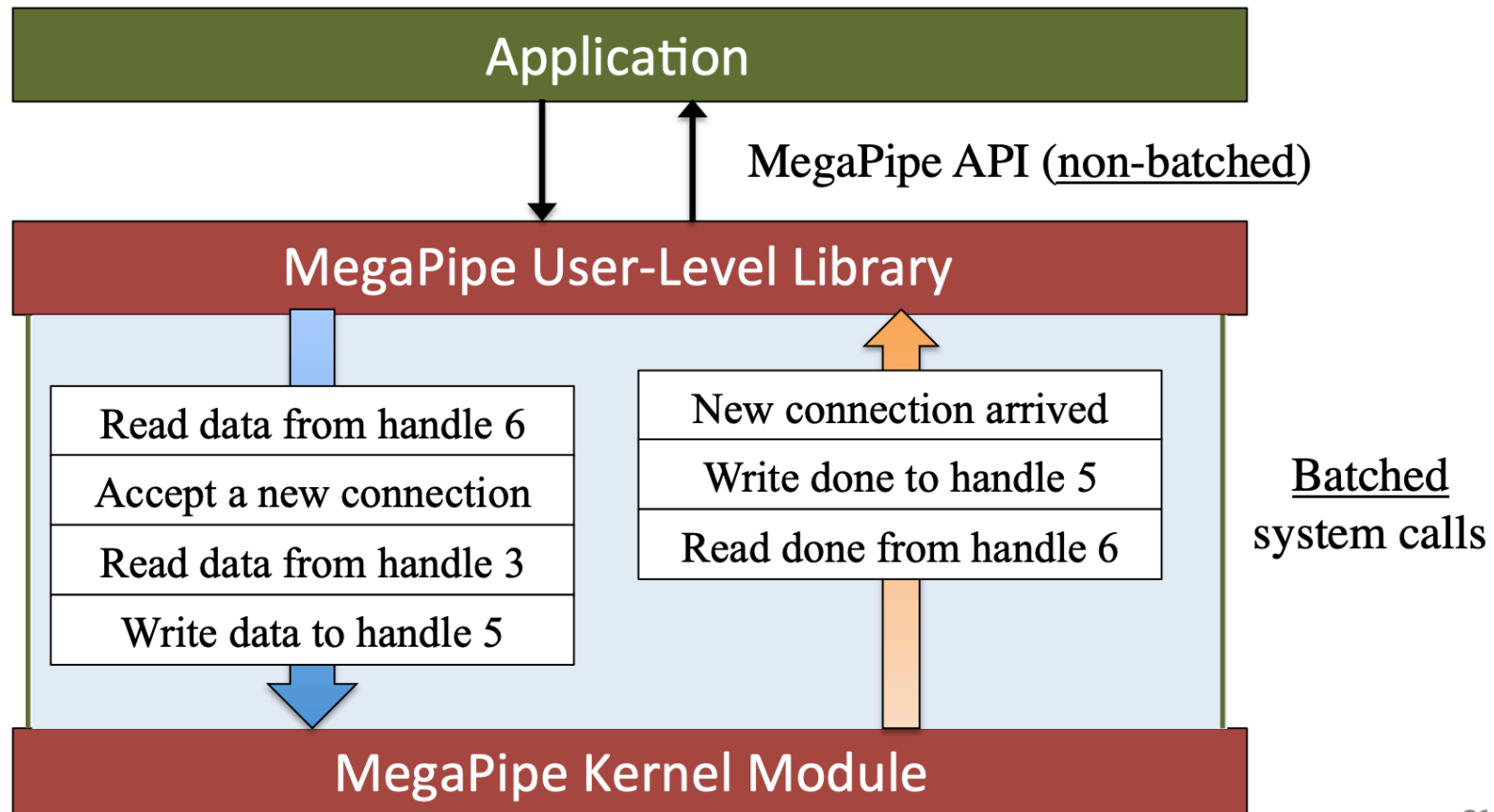
- Handle
 - Similar to file descriptor
 - But only valid within a channel
 - TCP connection, pipe, disk file, ...
- Channel
 - **Per-core**, bidirectional pipe between user and kernel
 - Multiplexes I/O operations of its handles

How channels help?

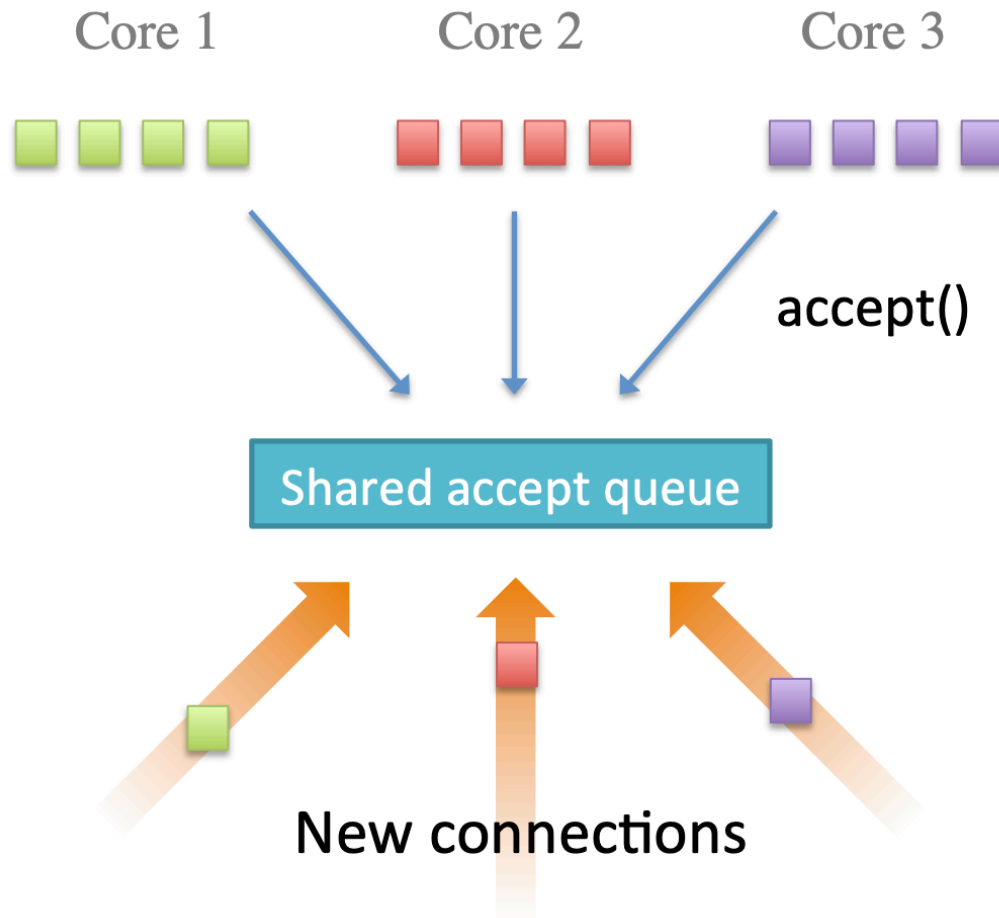


I/O Batching

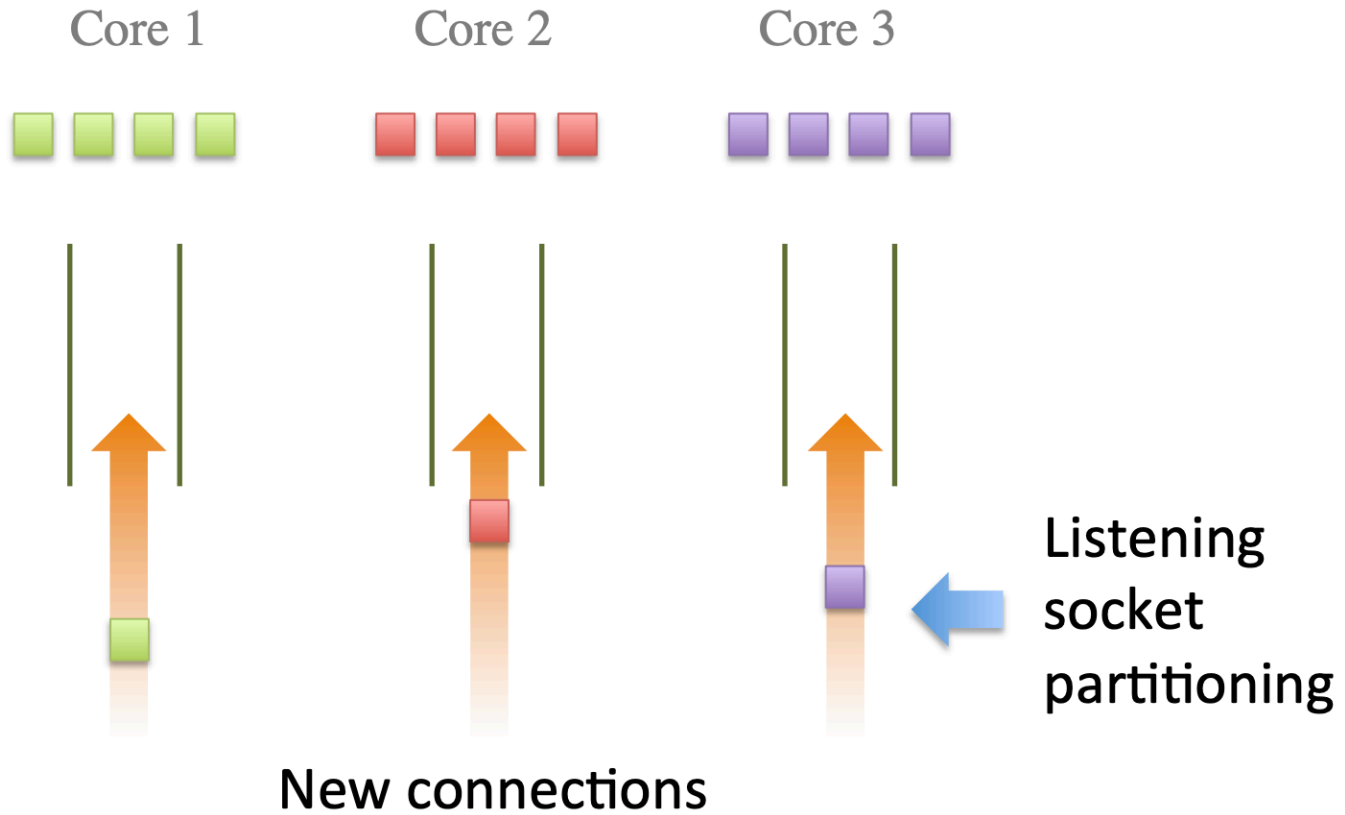
- Transparent batching
 - Exploits parallelism of independent handles



How channels help?

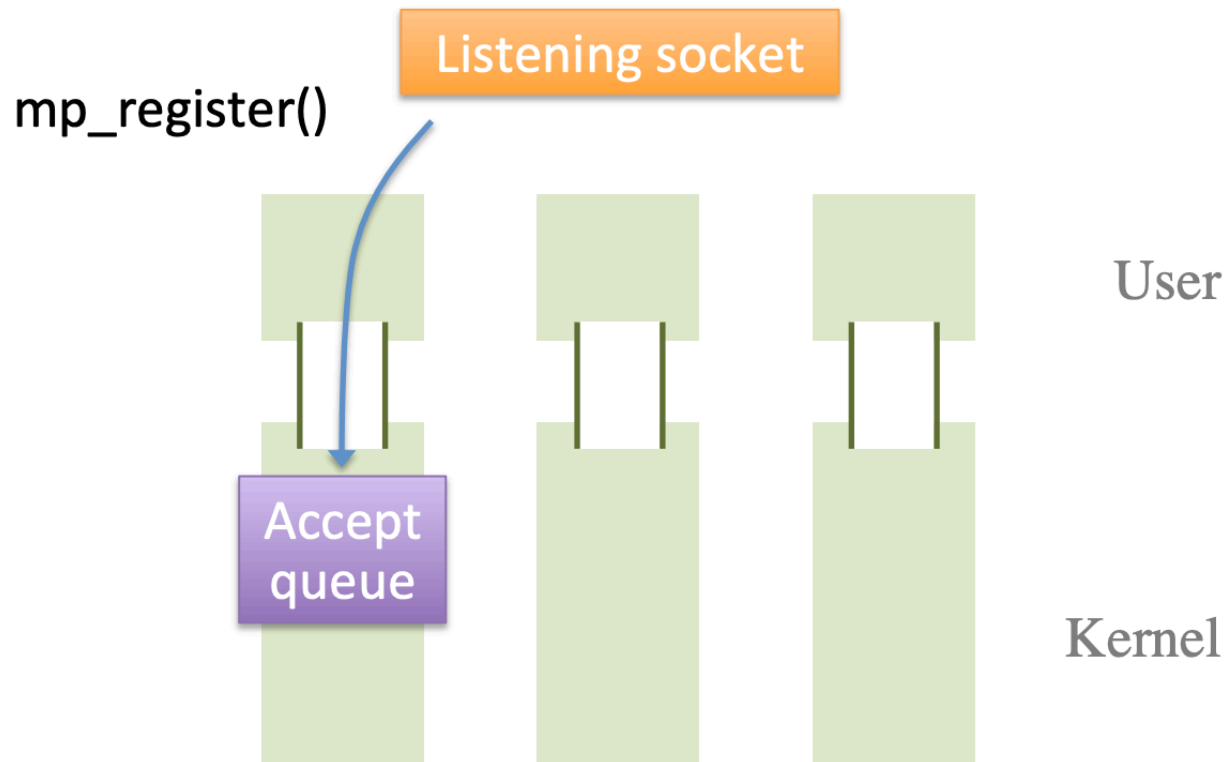


How channels help?



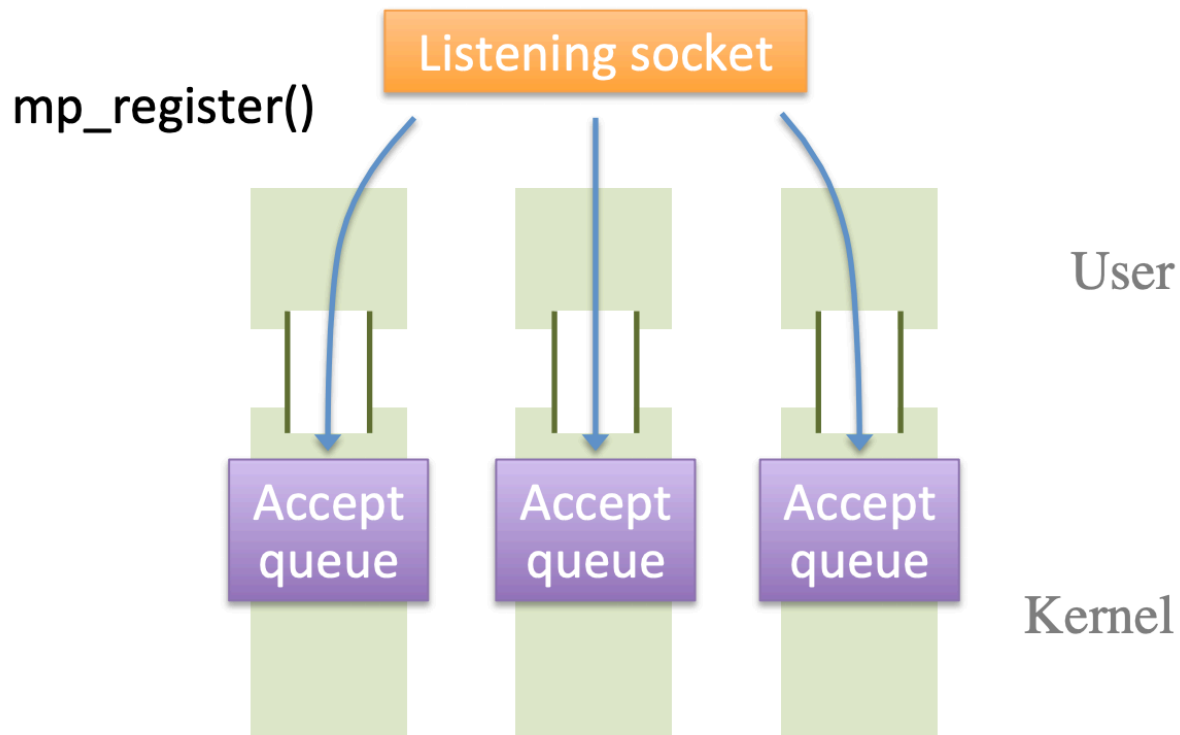
2. Listening Socket Partitioning

- Per-core accept queue for each channel
 - Instead of the globally shared accept queue



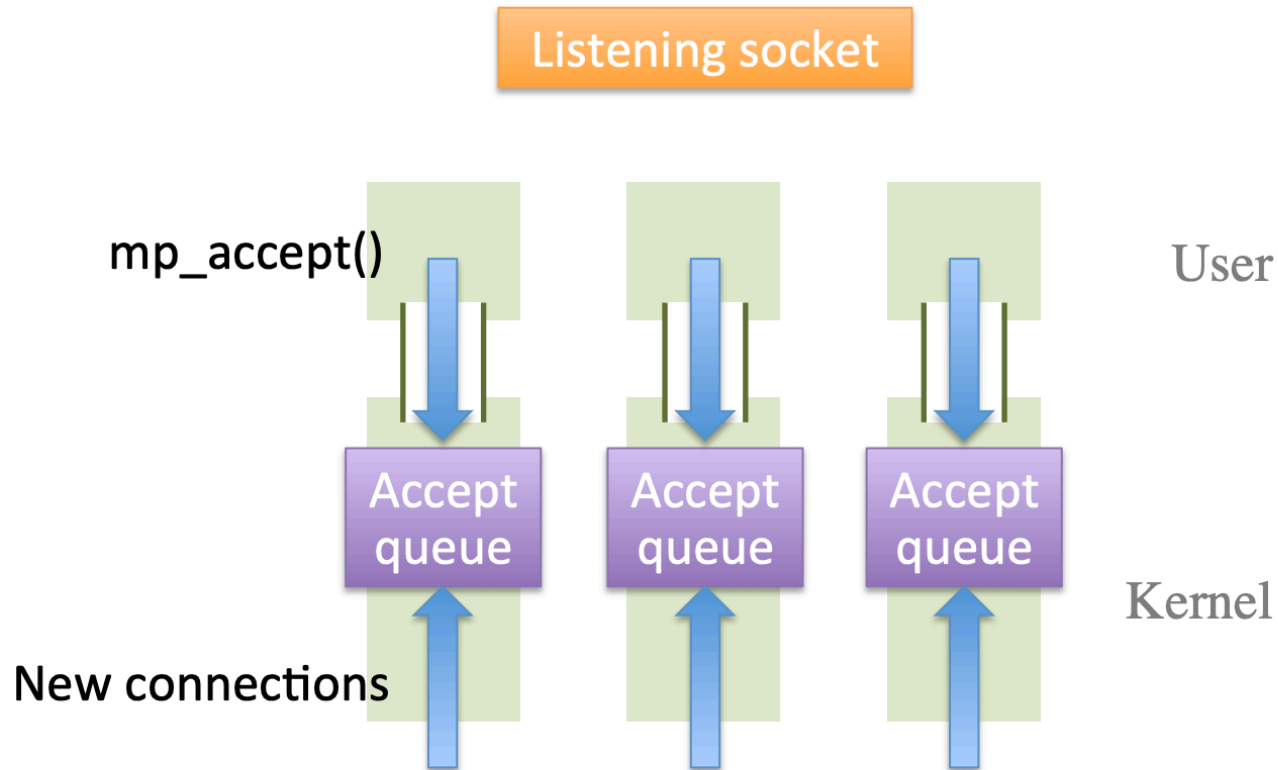
2. Listening Socket Partitioning

- Per-core accept queue for each channel
 - Instead of the globally shared accept queue

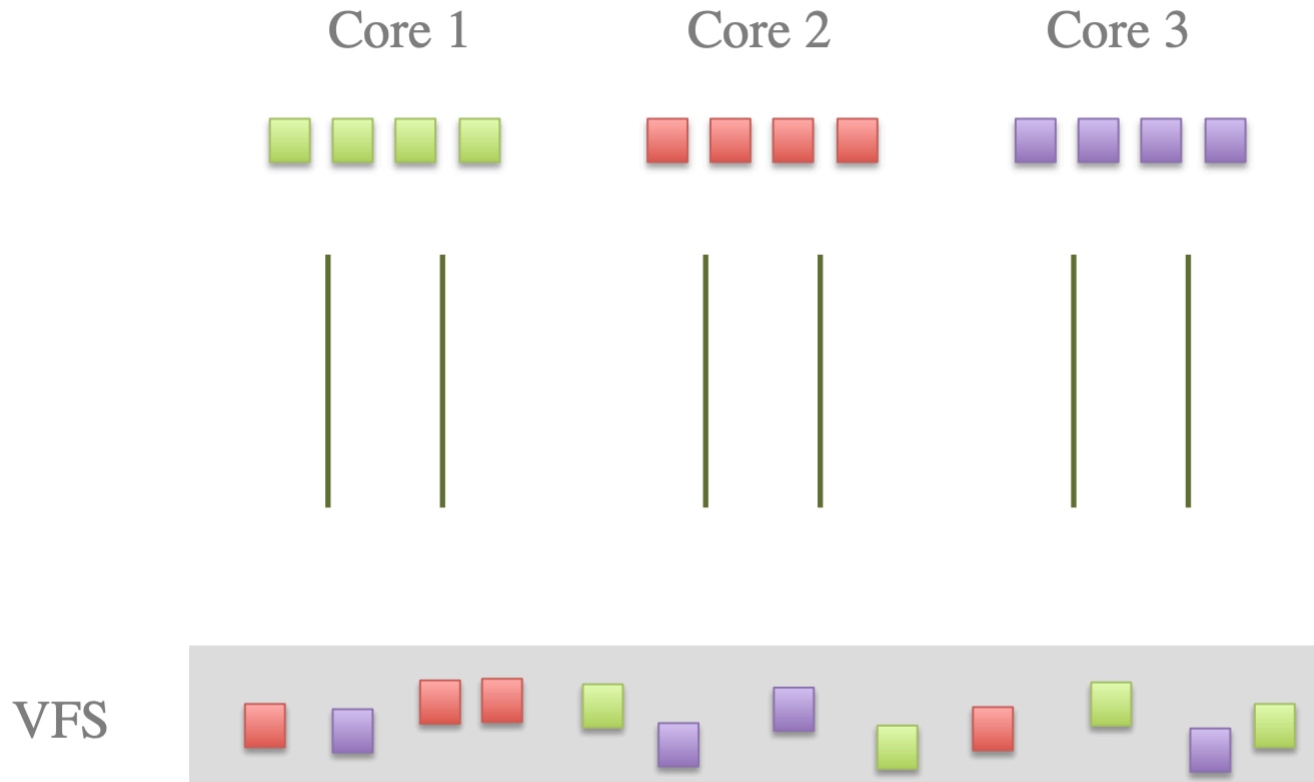


2. Listening Socket Partitioning

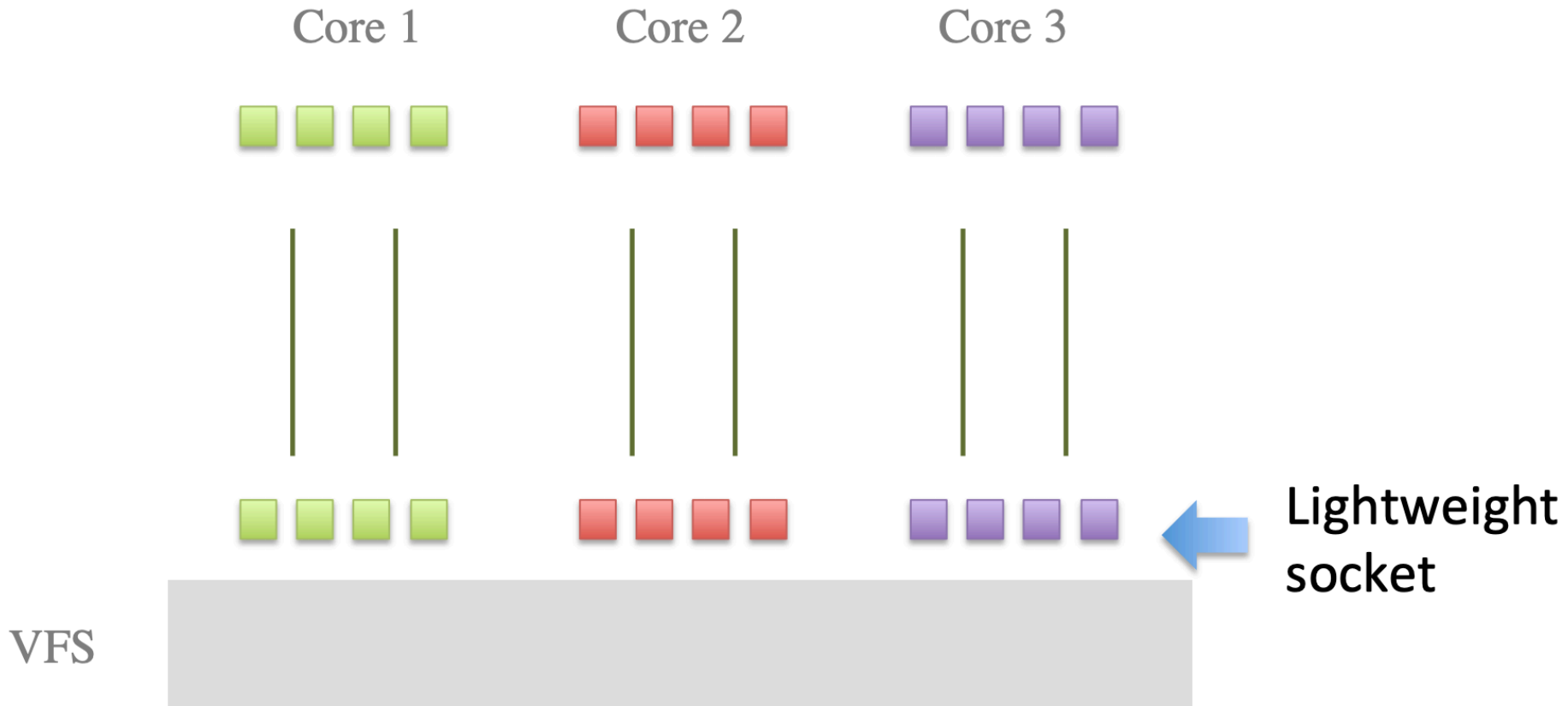
- Per-core accept queue for each channel
 - Instead of the globally shared accept queue



How channels help?

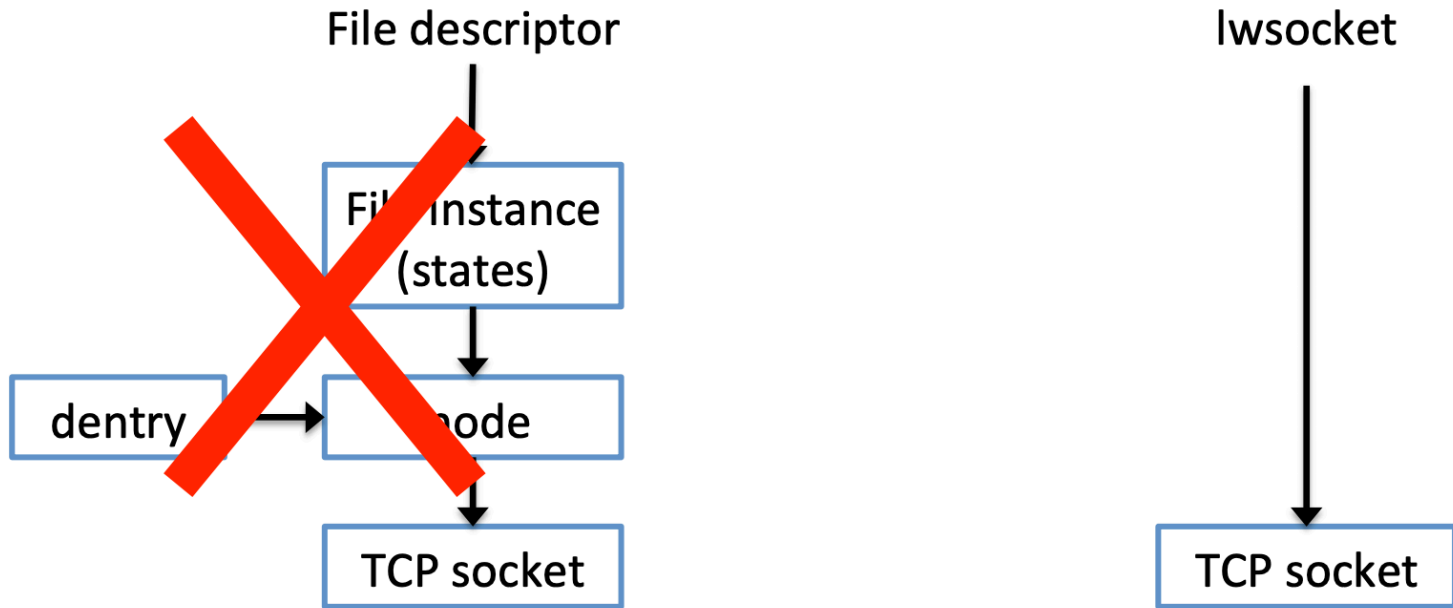


How channels help?



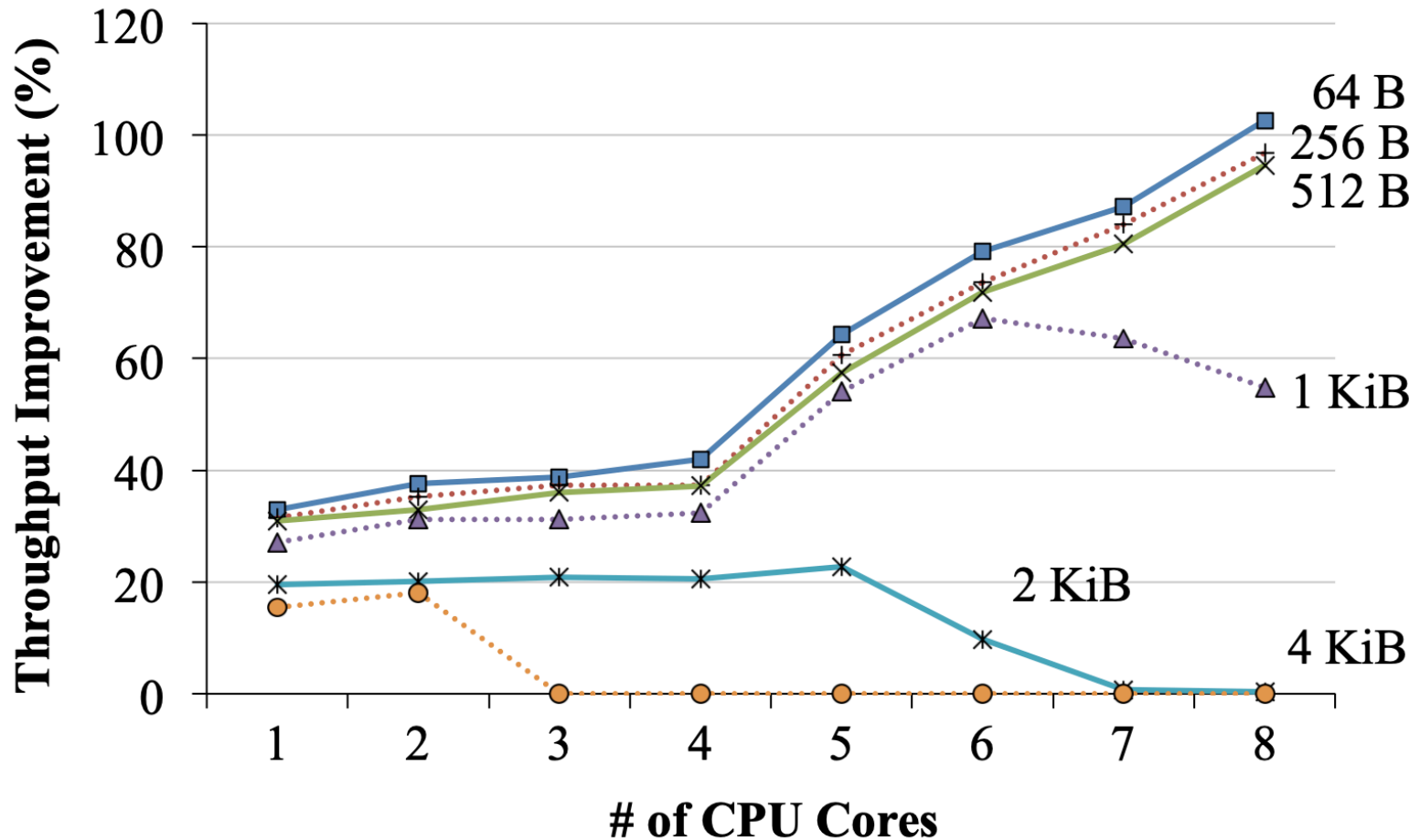
3. Light-weight Sockets

- Common-case optimization for sockets
 - Sockets are ephemeral and rarely shared
 - Bypass the VFS layer
 - Convert into a regular file descriptor only when necessary



Evaluation: Microbenchmarks

- Throughput improvement with various message sizes



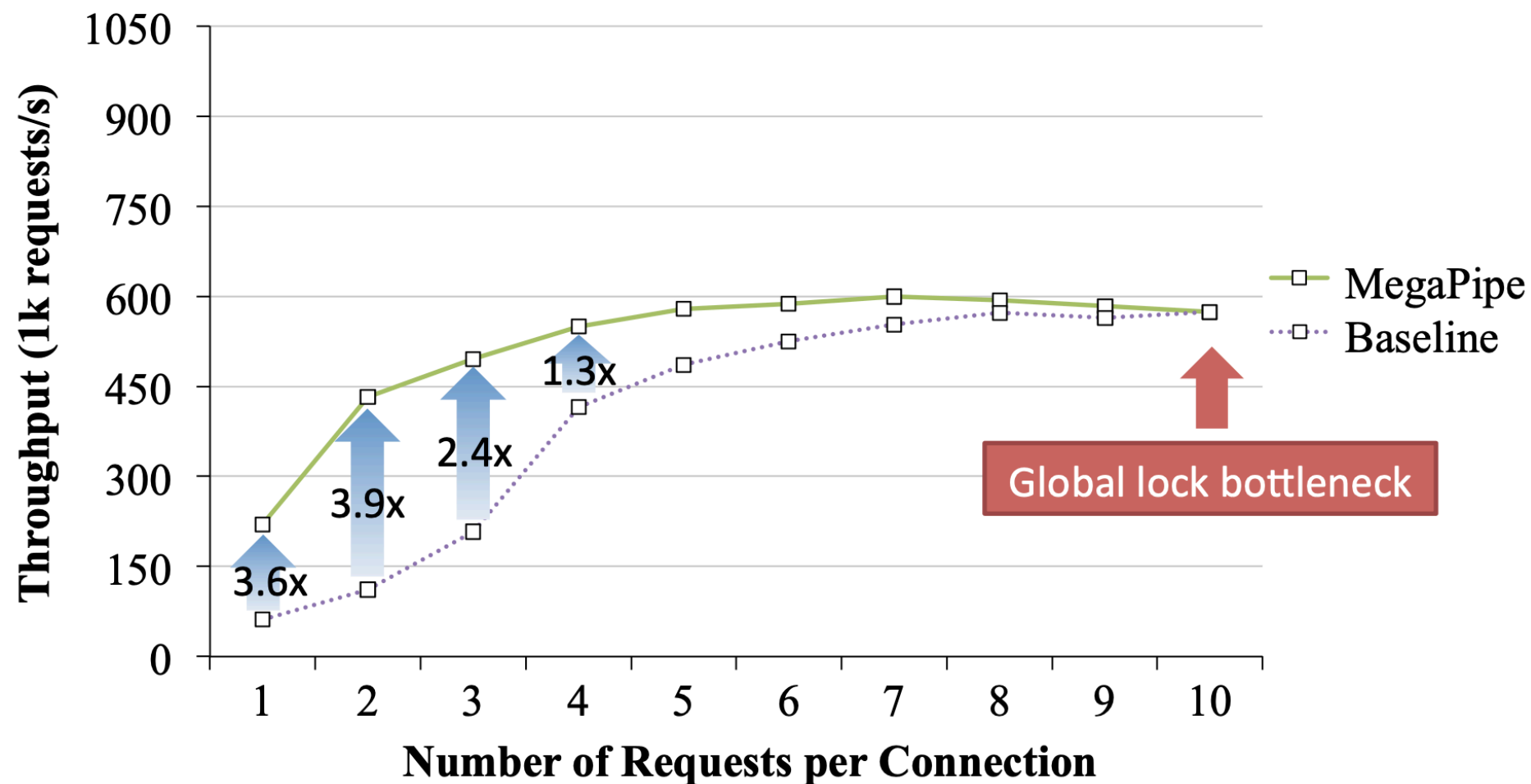
Evaluation: Macrobenchmarks

- memcached
 - In-memory key-value store
 - Limited scalability
 - Object store is shared by all cores with a global lock
- nginx
 - Web server
 - Highly scalable
 - Nothing is shared by cores, except for the listening socket

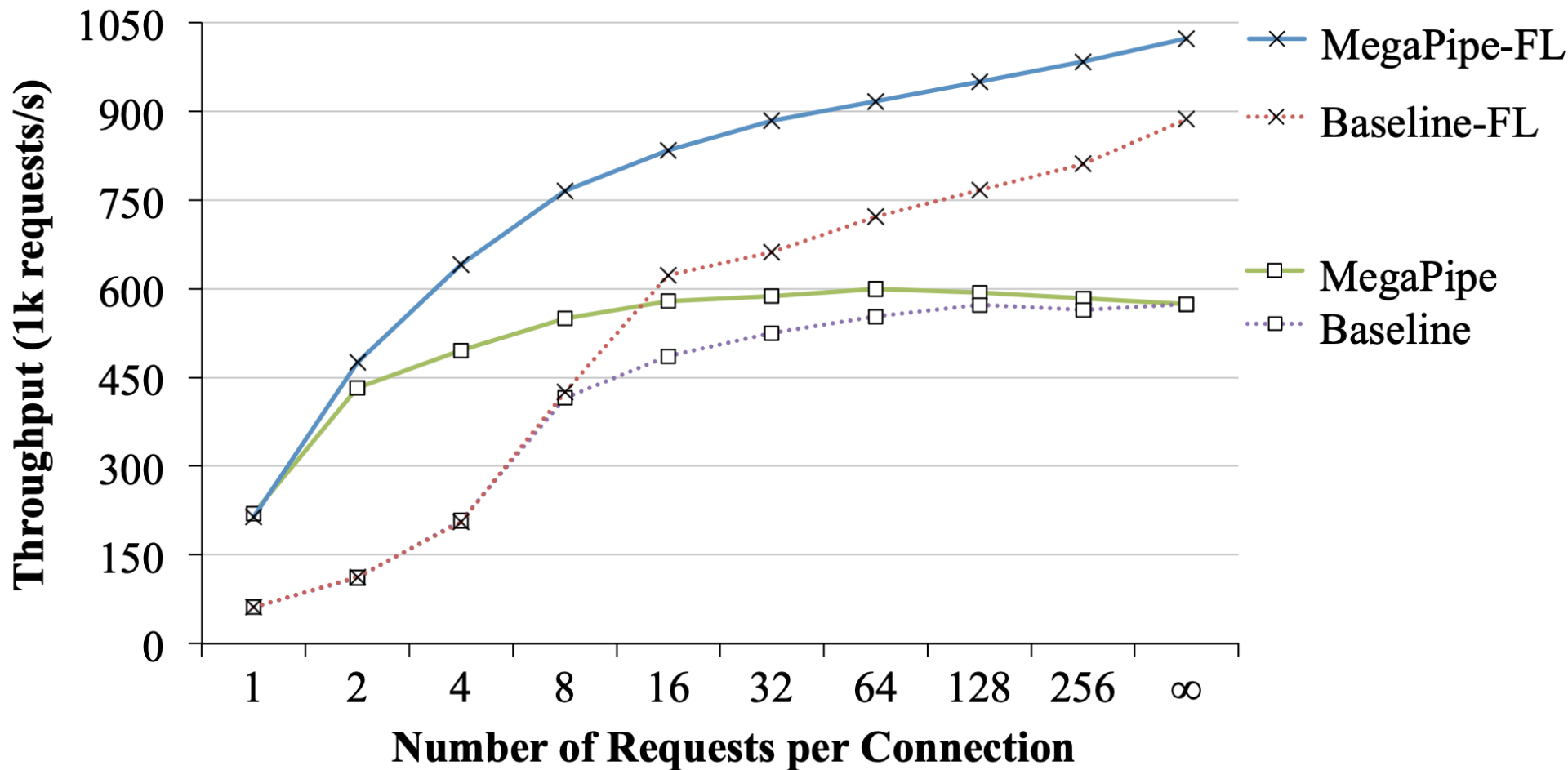
Evaluation: Macrobenchmarks

- memcached
 - In-memory key-value store
 - Limited scalability
 - Object store is shared by all cores with a global lock
- nginx
 - Web server
 - Highly scalable
 - Nothing is shared by cores, except for the listening socket

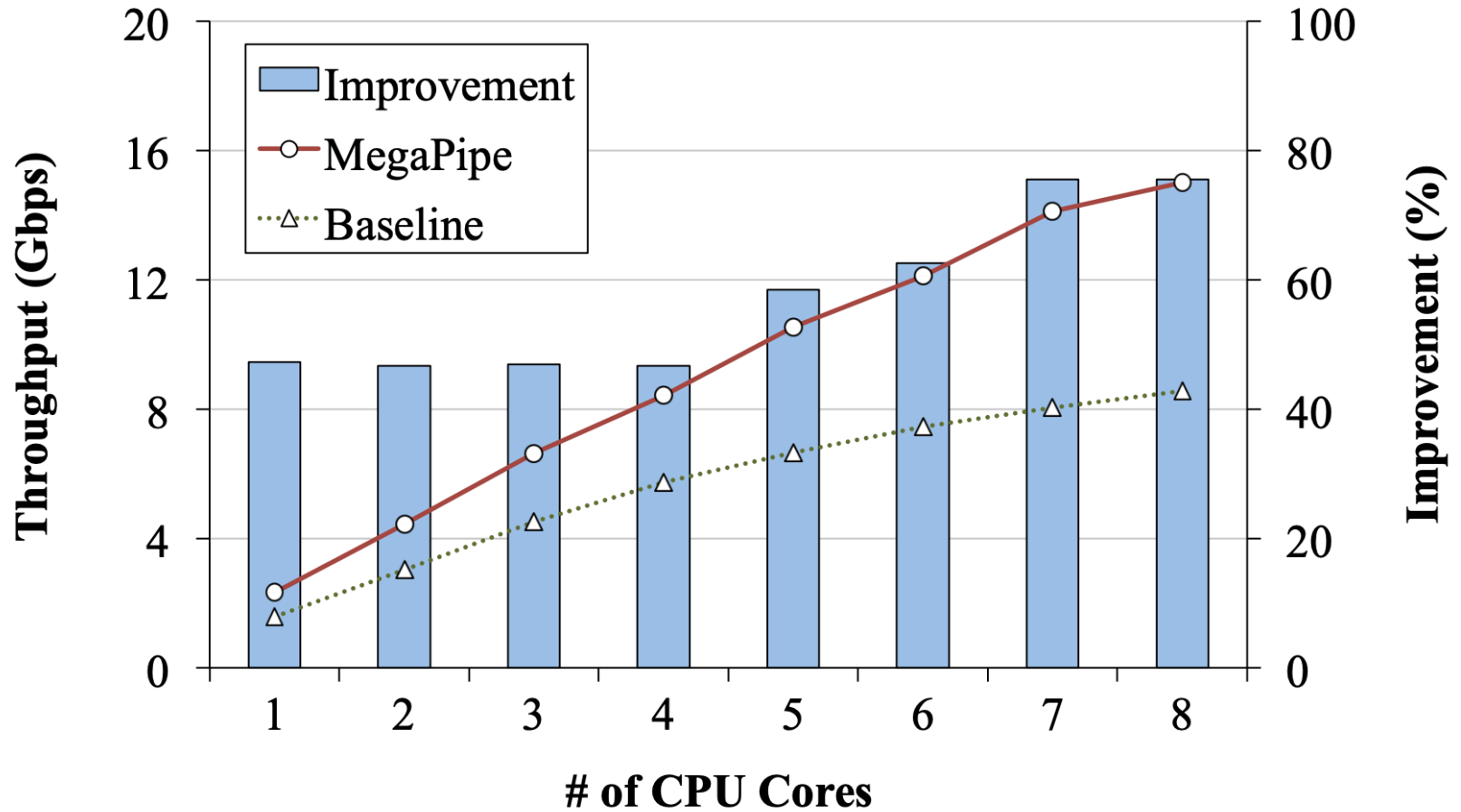
Evaluation: memcached



Evaluation: memcached



Evaluation: nginx



Conclusion

- Short connections or small messages:
 - High CPU overhead
 - Poorly scaling with multi-core CPUs
- MegaPipe
 - Key abstraction: per-core channel
 - Enabling three optimization opportunities:
 - Batching, partitioning, lwsocket
 - 15+% improvement for memcached, 75% for nginx

Your Opinions

Pros:

- Light-weight socket, batching, listening socket partitioning.
- Thorough evaluation of performance bottlenecks.
- Significant performance improvement (for nginx).

Your Opinions

Cons:

- Lack of backwards-compatibility.
- How much effort is required to port an application to use MegaPipe?
- Batching may impact latency.
- What do we lose out on by using lwsockets?
- Does not support (dynamic) load-balancing for partitioned sockets.
- Scaling beyond 8 cores?
- Kernel modifications may be difficult.
- Why not use MPI or RDMA?

Your Opinions

Ideas:

- Secure accept queue sharing with access control
- Is MegaPipe useful beyond network I/O?
- Beyond Linux?
- Load balancing for socket partitioning.
- Lower syscall cost.
- Combining RouteBricks with MegaPipe.
- What hardware optimizations can be applied?
- Network IO interface that is both high performance and POSIX-compliant.
- Automate application modifications to use MegaPipe.

Discuss!

What other sources of performance overhead remain?