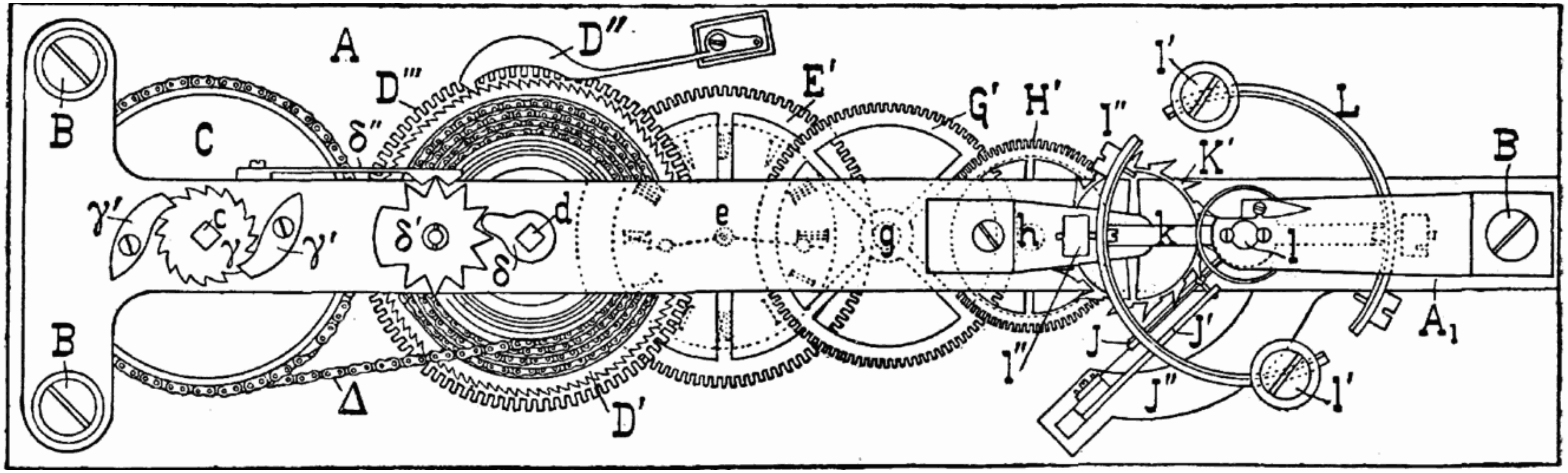
FlexSC

Flexible System Call Scheduling with Exception-Less System Calls

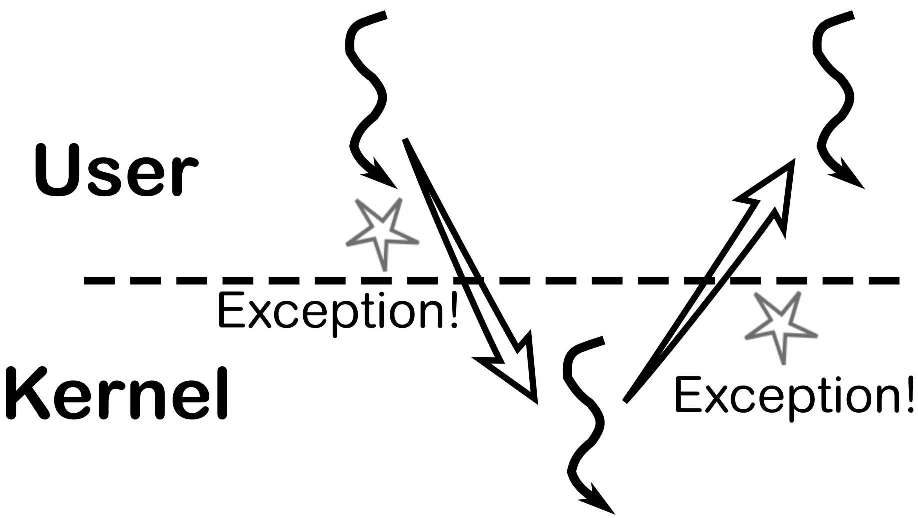
Livio Soares and Michael Stumm

University of Toronto



Motivation

The **synchronous** system call interface is a legacy from the single core era



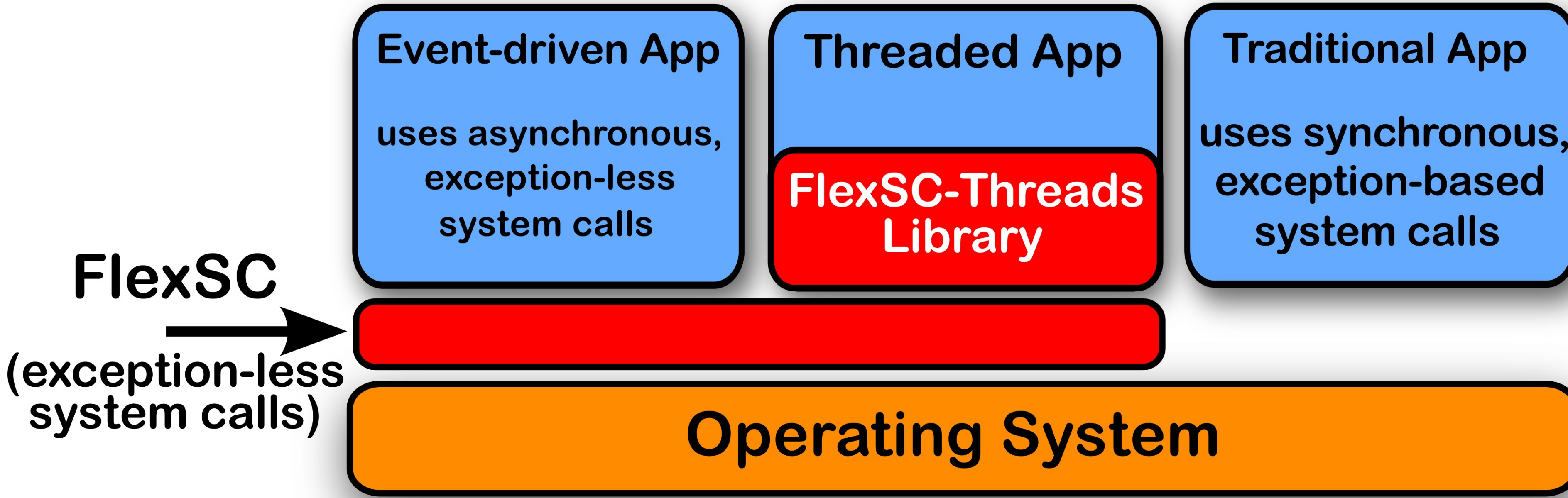
Expensive! Costs are:

- ➔ **direct**: mode-switch
- ➔ **indirect**: processor structure pollution

FlexSC implements **efficient and flexible** system calls for the multicore era

FlexSC overview

Two contributions: FlexSC and FlexSC-Threads



Results in:

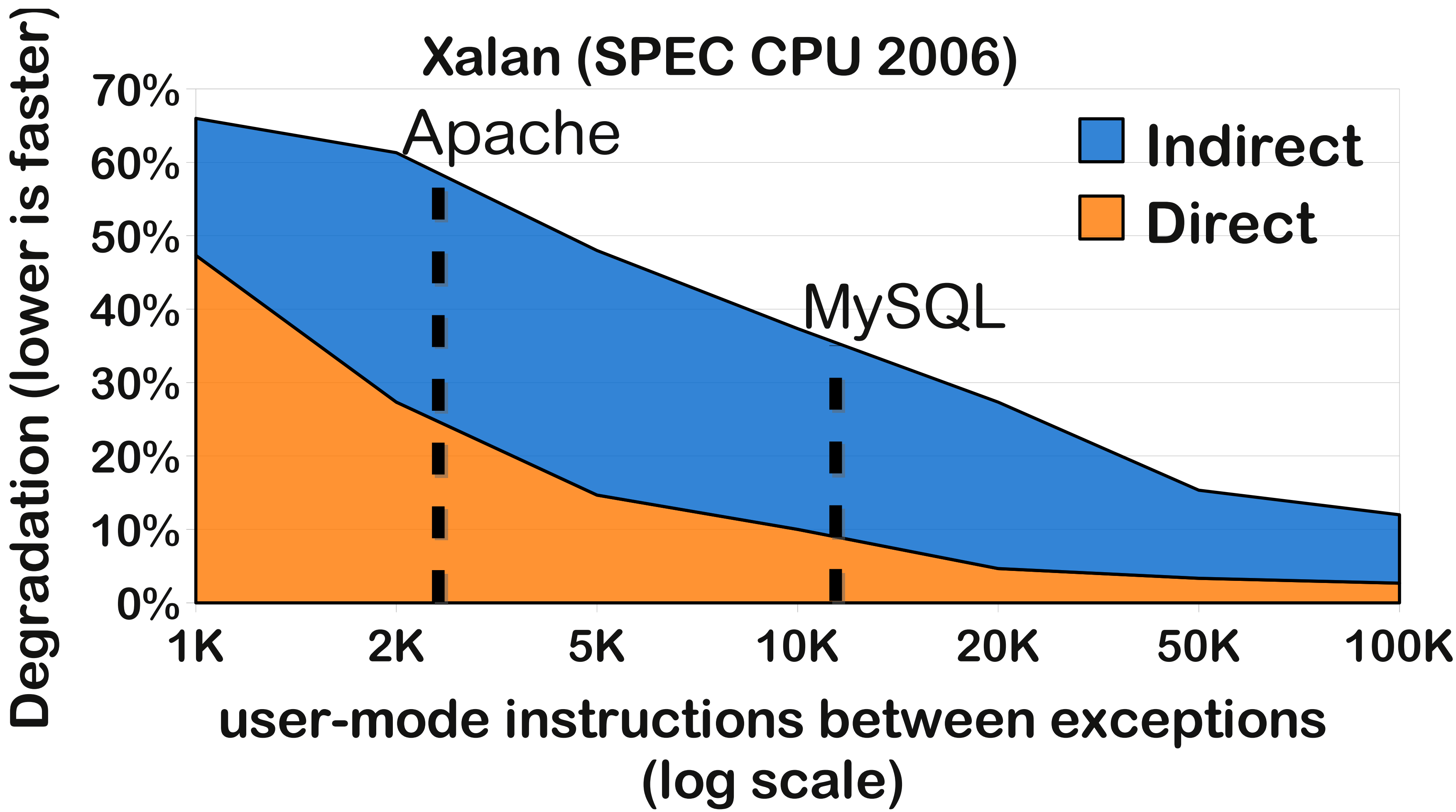
1) MySQL throughput increase of up to 40% and latency reduction of 30%
2) Apache throughput increase of up to 115% and latency reduction of 50%

Performance impact of synchronous syscalls

- Xalan from SPEC CPU 2006
  - Virtually no time in the OS
- Linux on Intel Core i7 (Nehalem)
- Injected exceptions with varying frequencies
  - **Direct**: emulate null system call
  - **Indirect**: emulate “write()” system call
- Measured only user-mode time
  - Kernel time ignored

Ideally, user-mode performance is unaltered

Degradation due to sync. syscalls



System calls can **half** processor efficiency;
indirect cause is major contributor

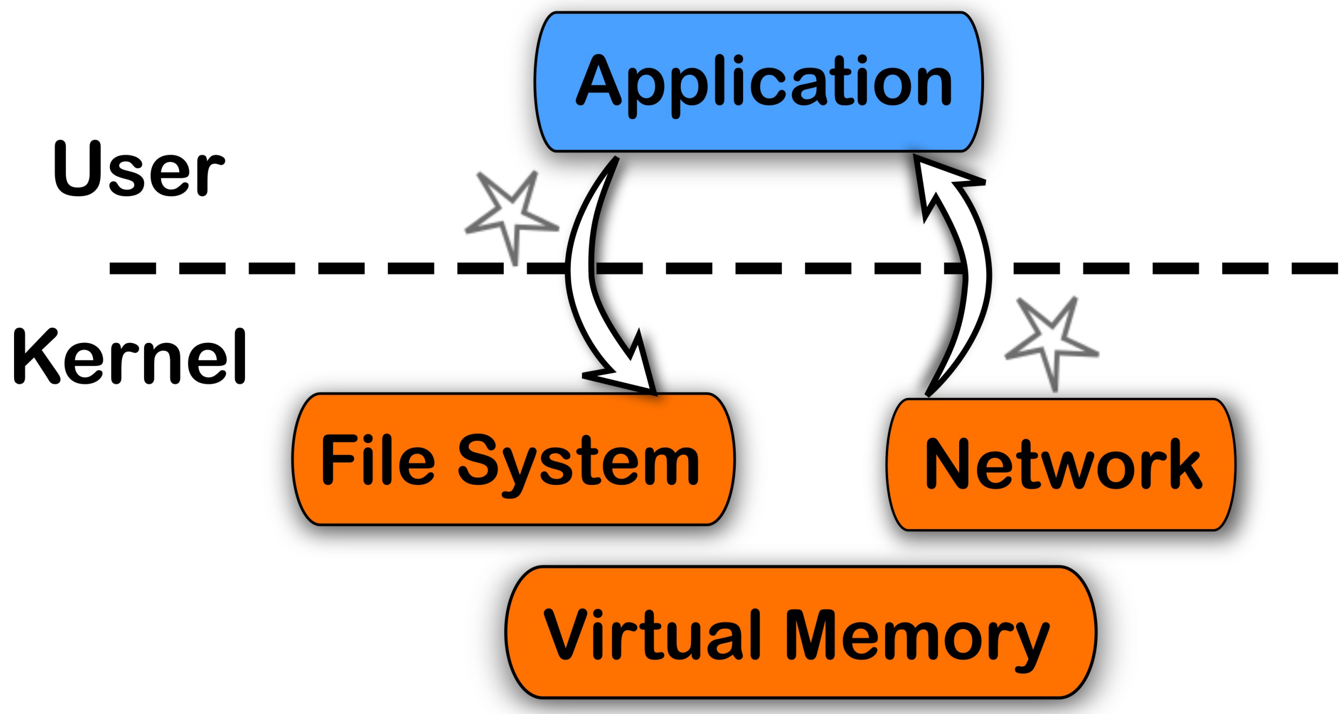
Processor state pollution

- ➔ Key source of performance impact

- ➔ On a Linux write() call:
  - ➔ up to 2/3rd of the L1 data cache and data TLB are **evicted**

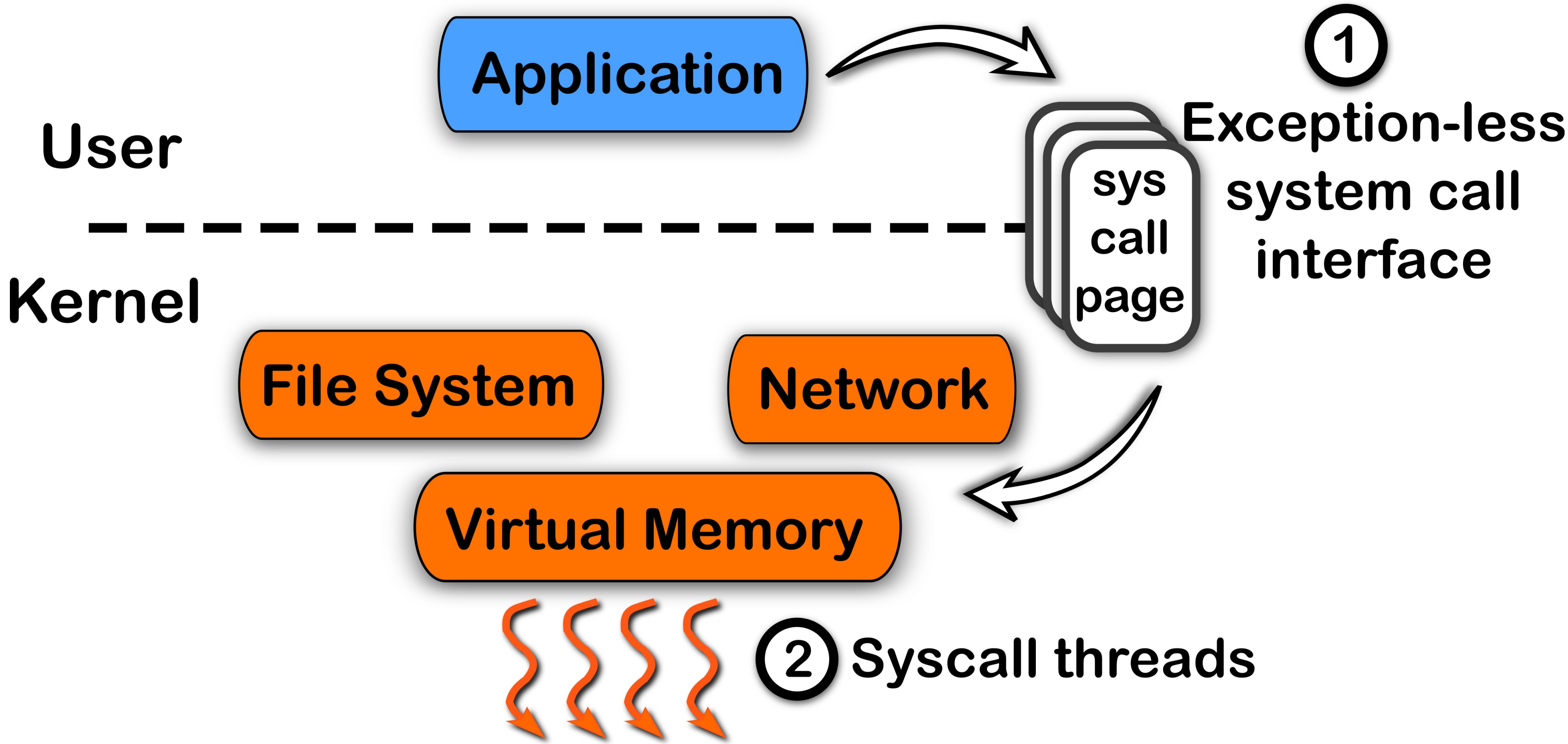
- ➔ Kernel performance equally affected
  - ➔ Processor efficiency for OS code is also cut in **half**

Synchronous system calls are expensive



Traditional system calls are synchronous and use exceptions to cross domains

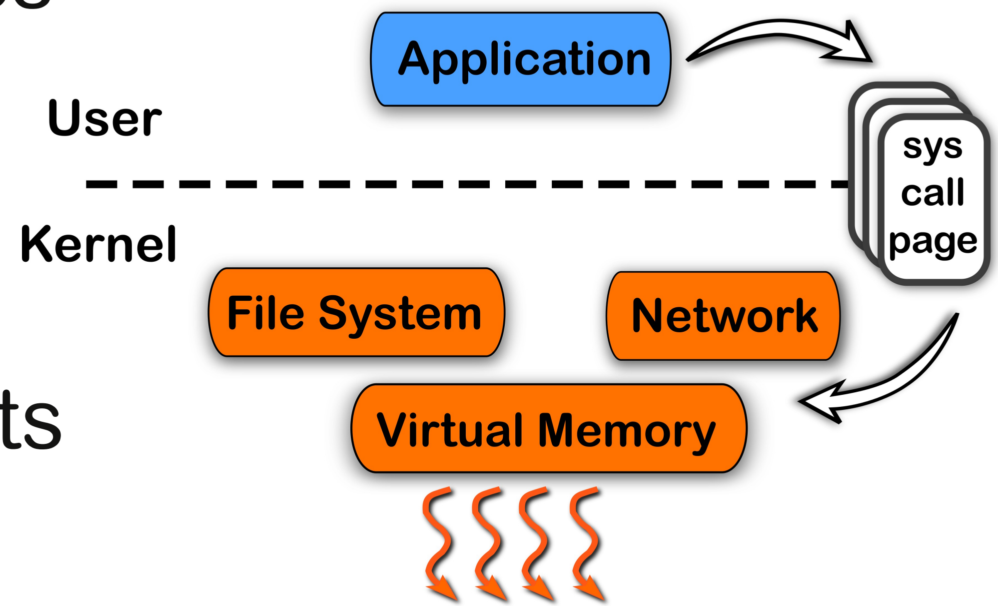
Alternative: side-step the boundary



Exception-less syscalls remove synchronicity by decoupling invocation from execution

Benefits of exception-less system calls

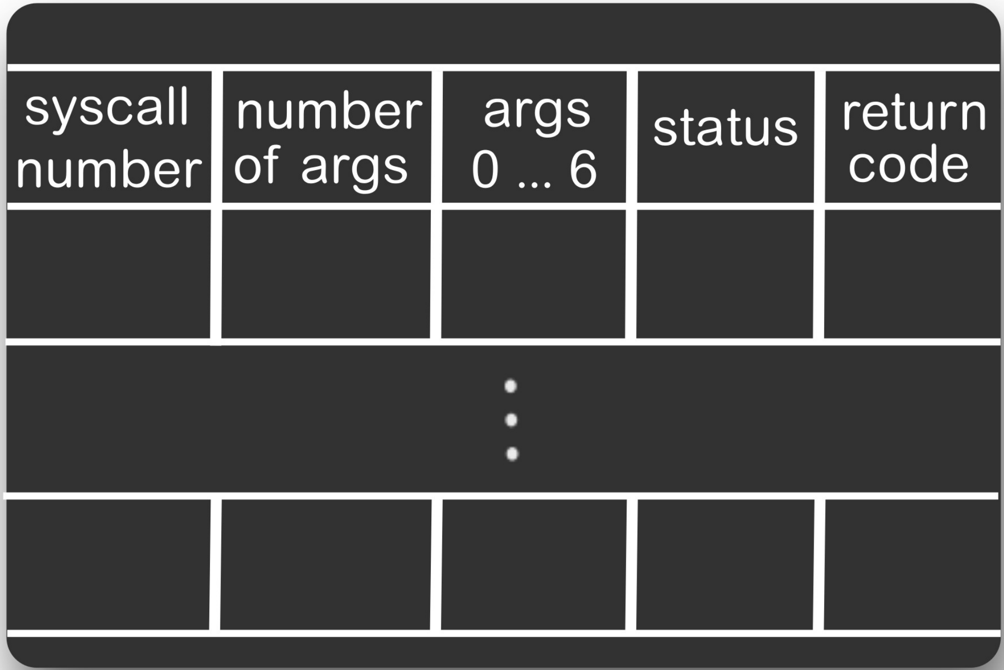
- Significantly reduce direct costs
  - Fewer mode switches
- Allow for batching
  - Reduce indirect costs



- Allow for dynamic multicore specialization
  - Further reduce direct and indirect costs

Exception-less interface: syscall page

```
write(fd, buf, 4096);



entry = free_syscall_entry();

/* write syscall */
entry->syscall = 1;
entry->num_args = 3;
entry->args[0] = fd;
entry->args[1] = buf;
entry->args[2] = 4096;
entry->status = SUBMIT;

while (entry->status != DONE)
   do_something_else();

return entry->return_code;
```



Exception-less interface: syscall page

```
write(fd, buf, 4096);

entry = free_syscall_entry();

/* write syscall */
entry->syscall = 1;
entry->num_args = 3;
entry->args[0] = fd;
entry->args[1] = buf;
entry->args[2] = 4096;
entry->status = SUBMIT;

while (entry->status != DONE)
   do_something_else();

return entry->return_code;
```

syscall number	number of args	args 0 ... 6	status	return code
		⋮		
1	3	fd, buf, 4096	SUBMIT	

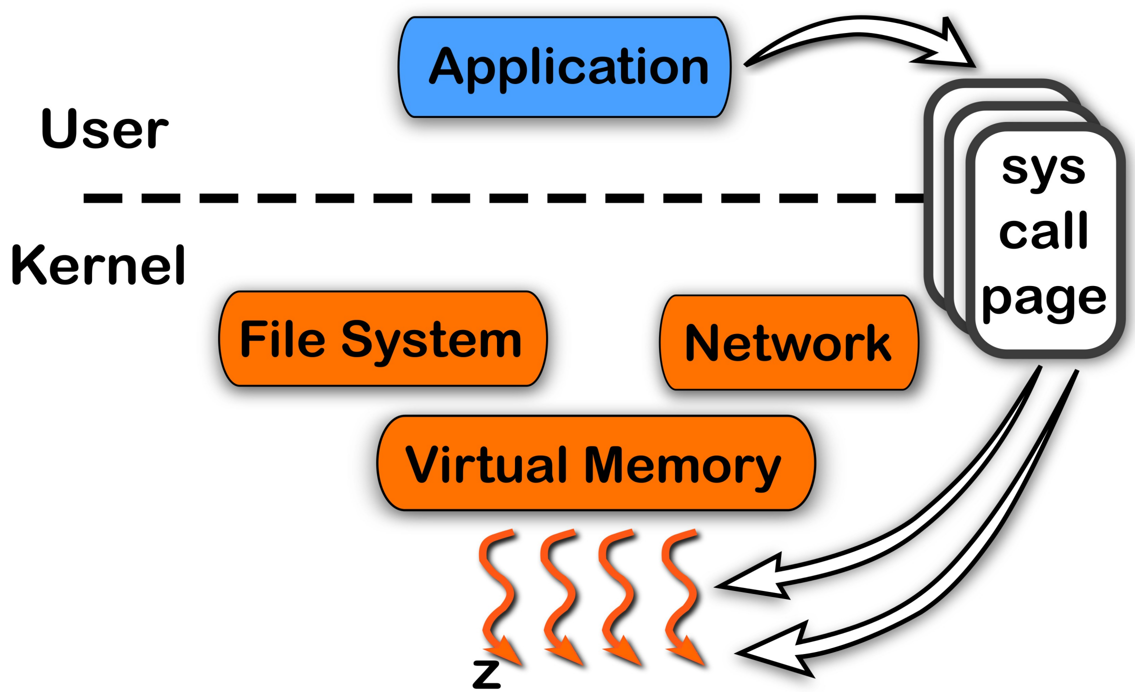
Exception-less interface: syscall page

```
write(fd, buf, 4096);

entry = free_syscall_entry();

/* write syscall */
entry->syscall = 1;
entry->num_args = 3;
entry->args[0] = fd;
entry->args[1] = buf;
entry->args[2] = 4096;
entry->status = SUBMIT;

while (entry->status != DONE)
   do_something_else();

return entry->return_code;
```

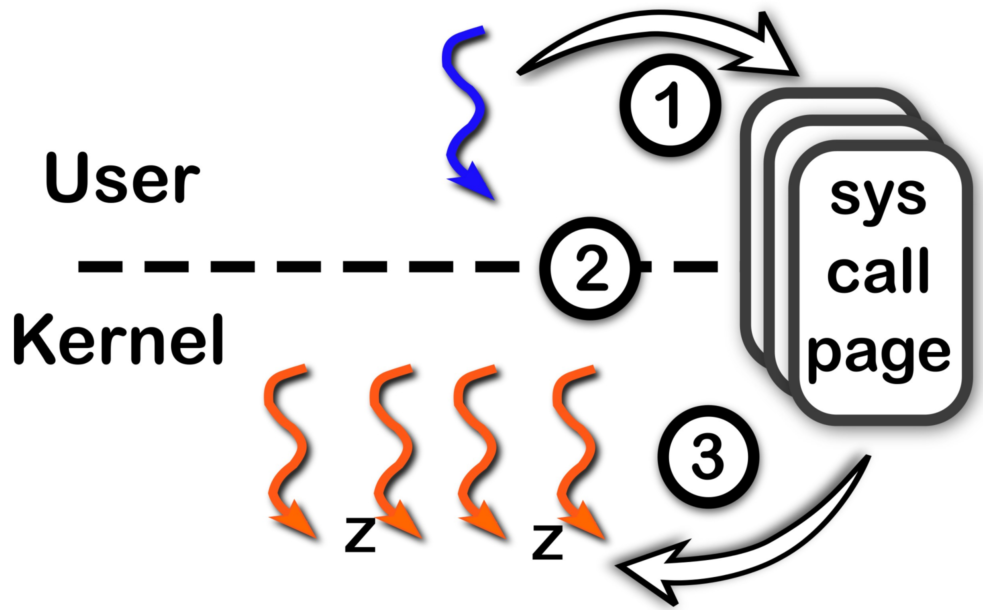
syscall number	number of args	args 0 ... 6	status	return code
		⋮		
1	3	fd, buf, 4096	DONE	4096

Syscall threads

- Kernel-only threads
  - Part of application process
- Execute requests from syscall page
- Schedulable on a per-core basis



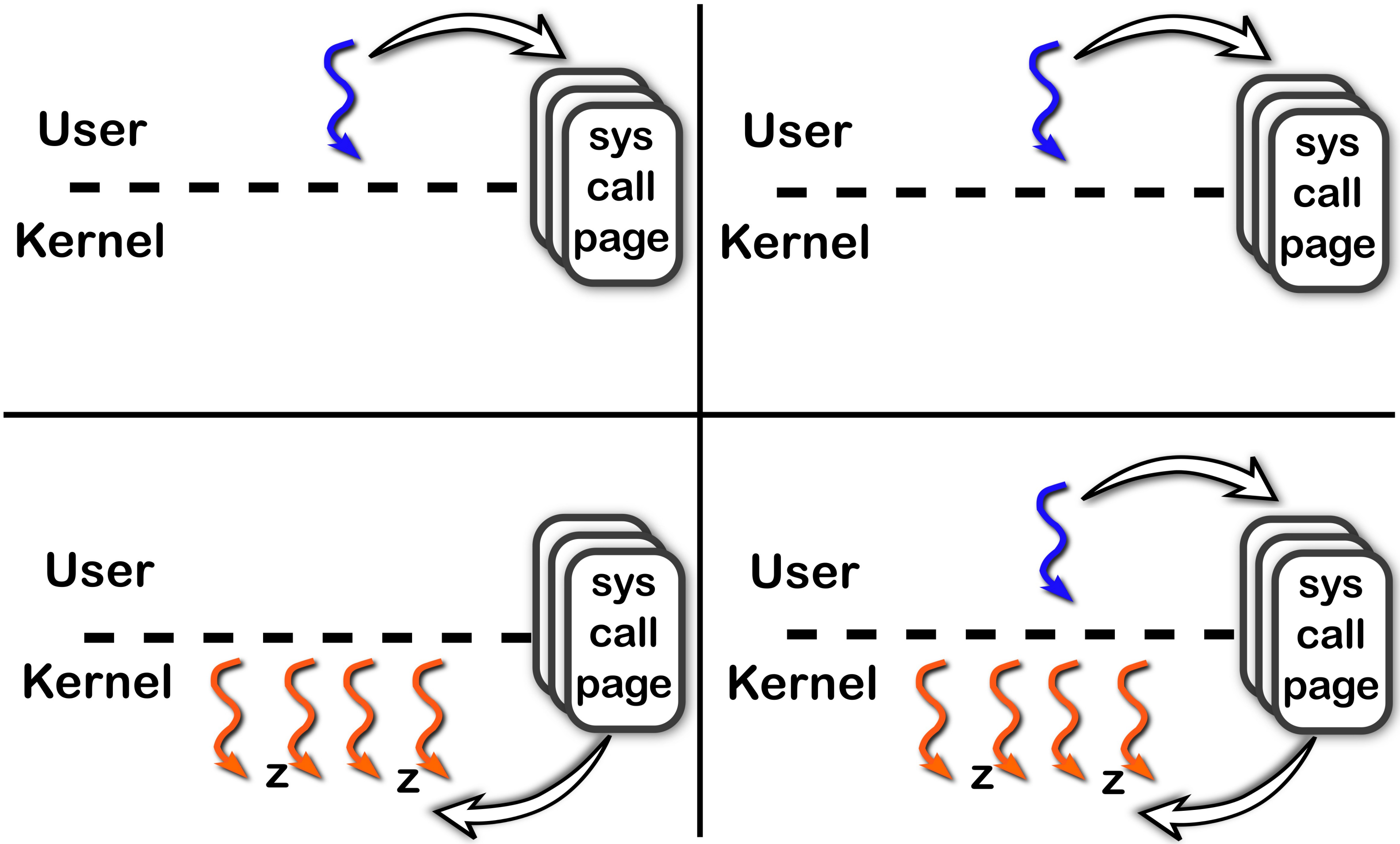
System call batching



① Request as many system calls as possible
② Switch to kernel-mode
③ Start executing all posted system calls

Avoids direct and indirect costs,
even on a single core

Dynamic multicore specialization



FlexSC makes specializing cores simple
Dynamically adapts to workload needs

What programs can benefit from FlexSC?

Event-driven servers

(e.g., memcached, nginx webserver)

- ➔ Use asynchoronous calls, similar to FlexSC
- ➔ Can use FlexSC *directly*
- ➔ Mix sync and exception-less system calls

Multi-threaded servers: **FlexSC-Threads**

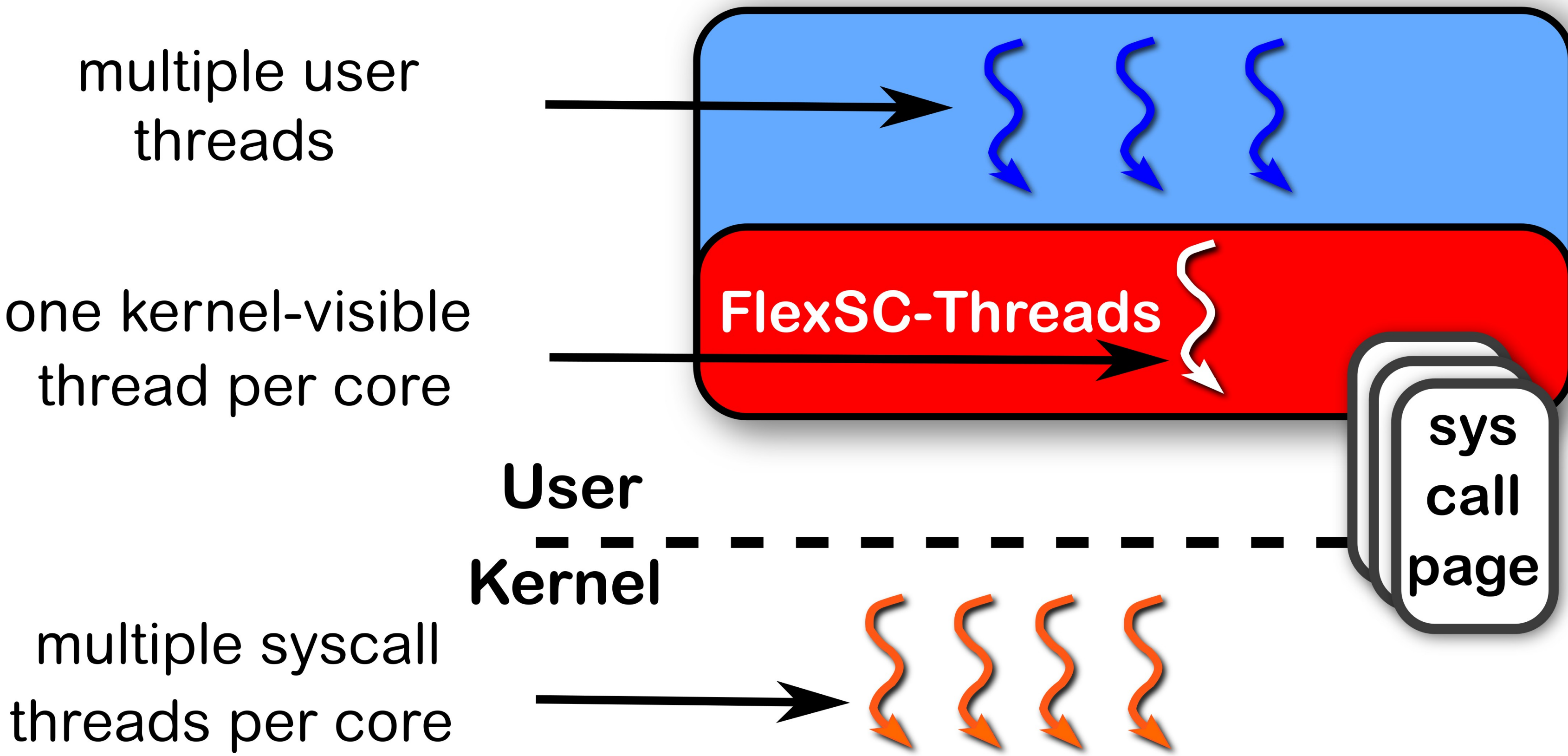
- ➔ Thread library, compatible with Pthreads
- ➔ No changes to app. code or recompilation required
- ➔ Transparently converts legacy syscalls into exception-less ones

FlexSC-Threads library

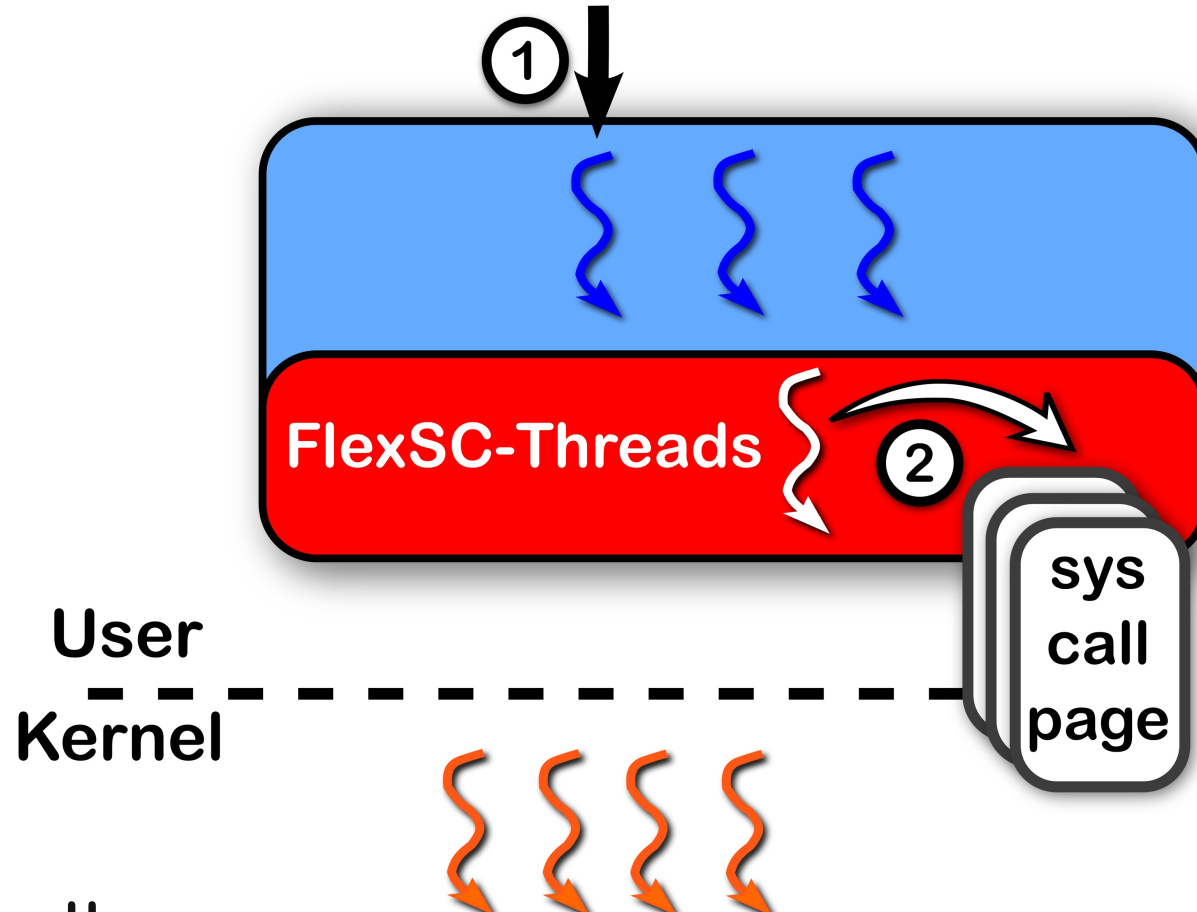
- Hybrid (M-on-N) threading model
  - One kernel visible thread per core
  - Many user threads per kernel-visible thread
- Redirects system calls (*libc* wrappers)
  - Posts exception-less syscall to syscall page
  - Switches to other user-level thread
  - Resumes thread upon syscall completion

Benefits of exception-less syscalls while maintaining sequential syscall interface

FlexSC-Threads in action



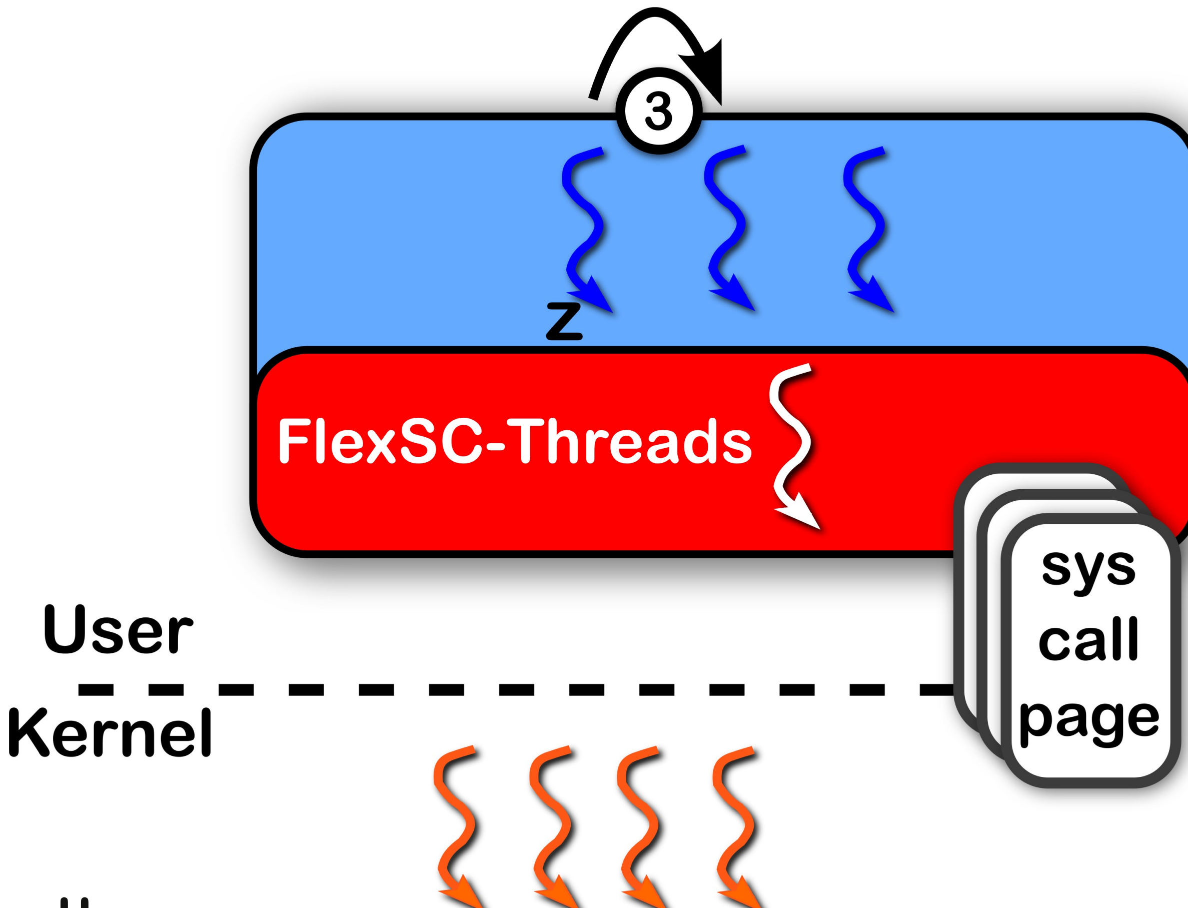
FlexSC-Threads in action



On a syscall:

1. Post request to system call page
2. Block user-level thread

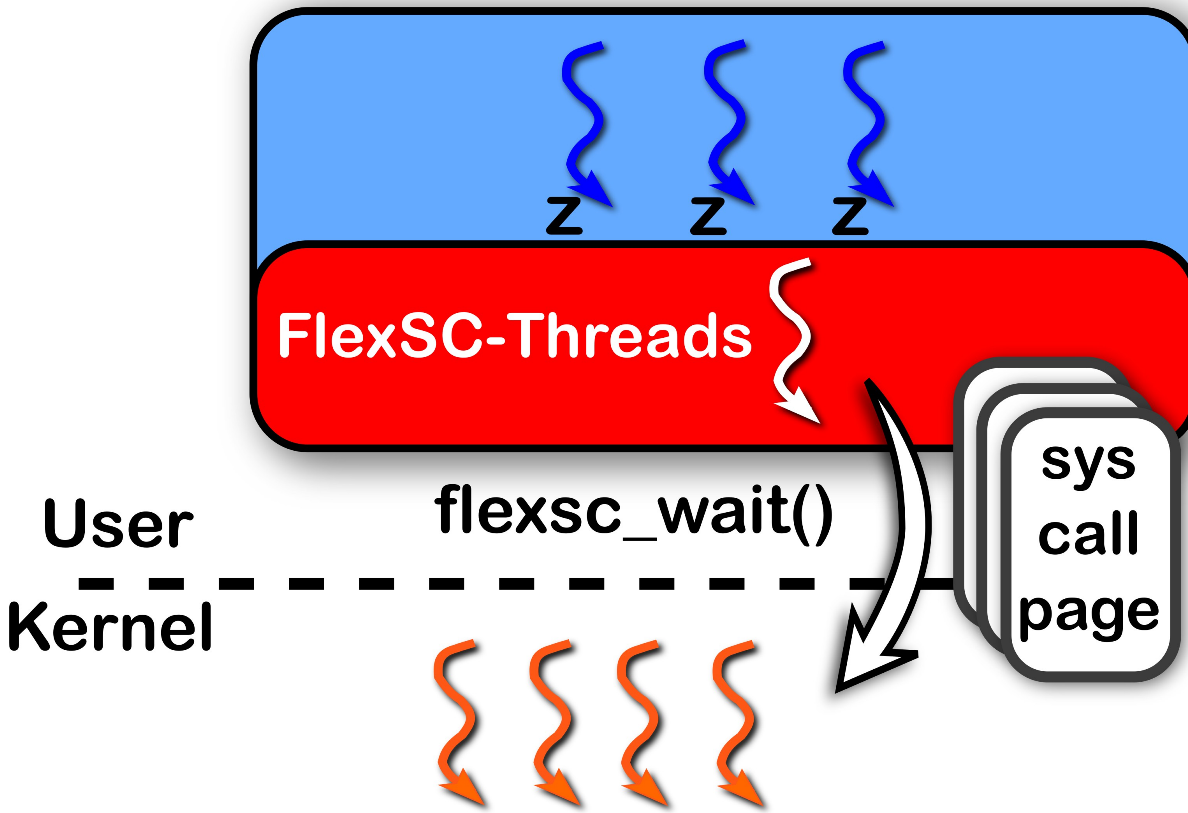
FlexSC-Threads in action



On a syscall:

① Post request to system call page
② Block user-level thread
③ Switch to next ready thread

FlexSC-Threads in action



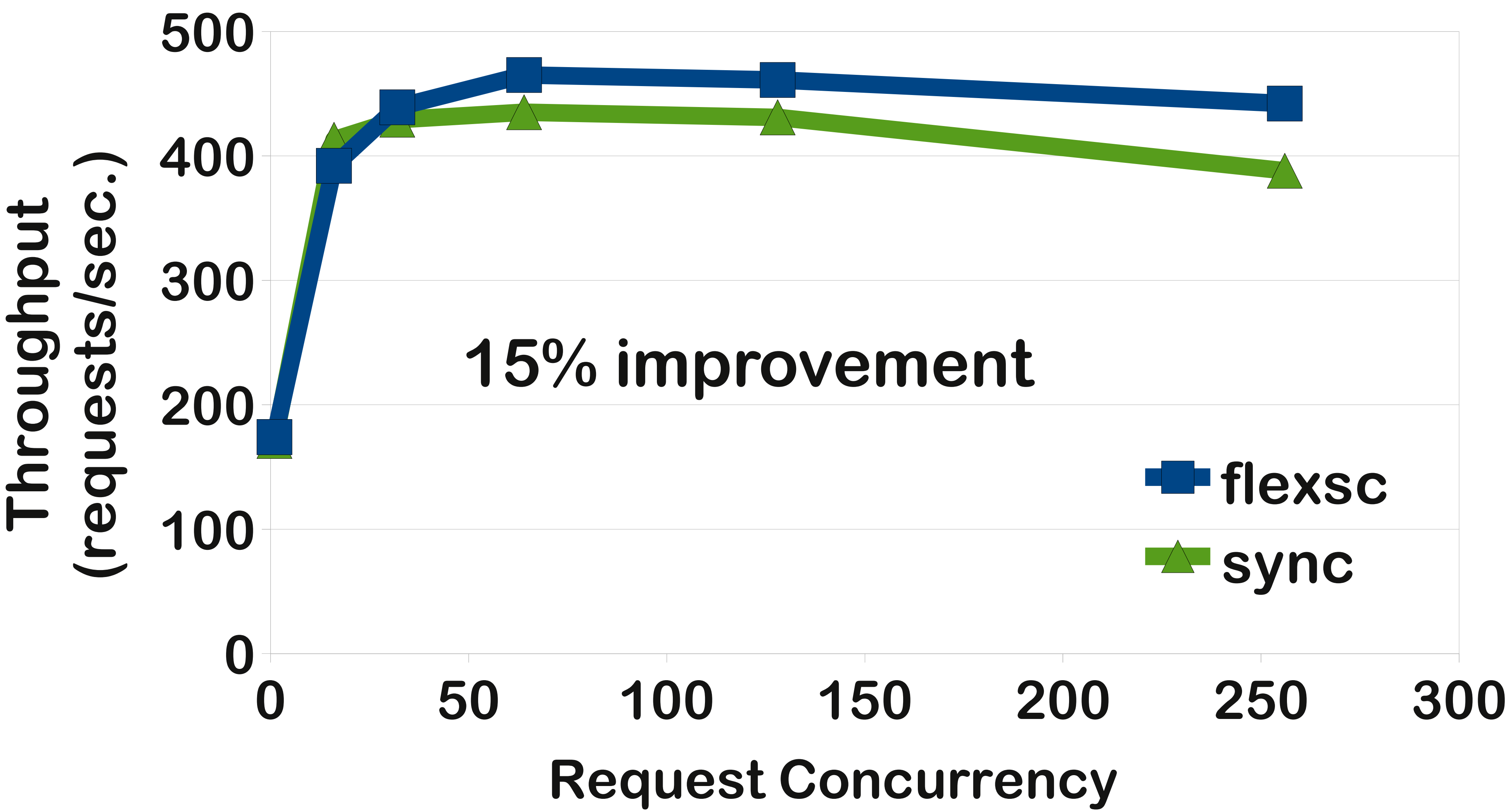
If all user-level threads become blocked:

1) enter kernel
2) wait for completion of at least 1 syscall

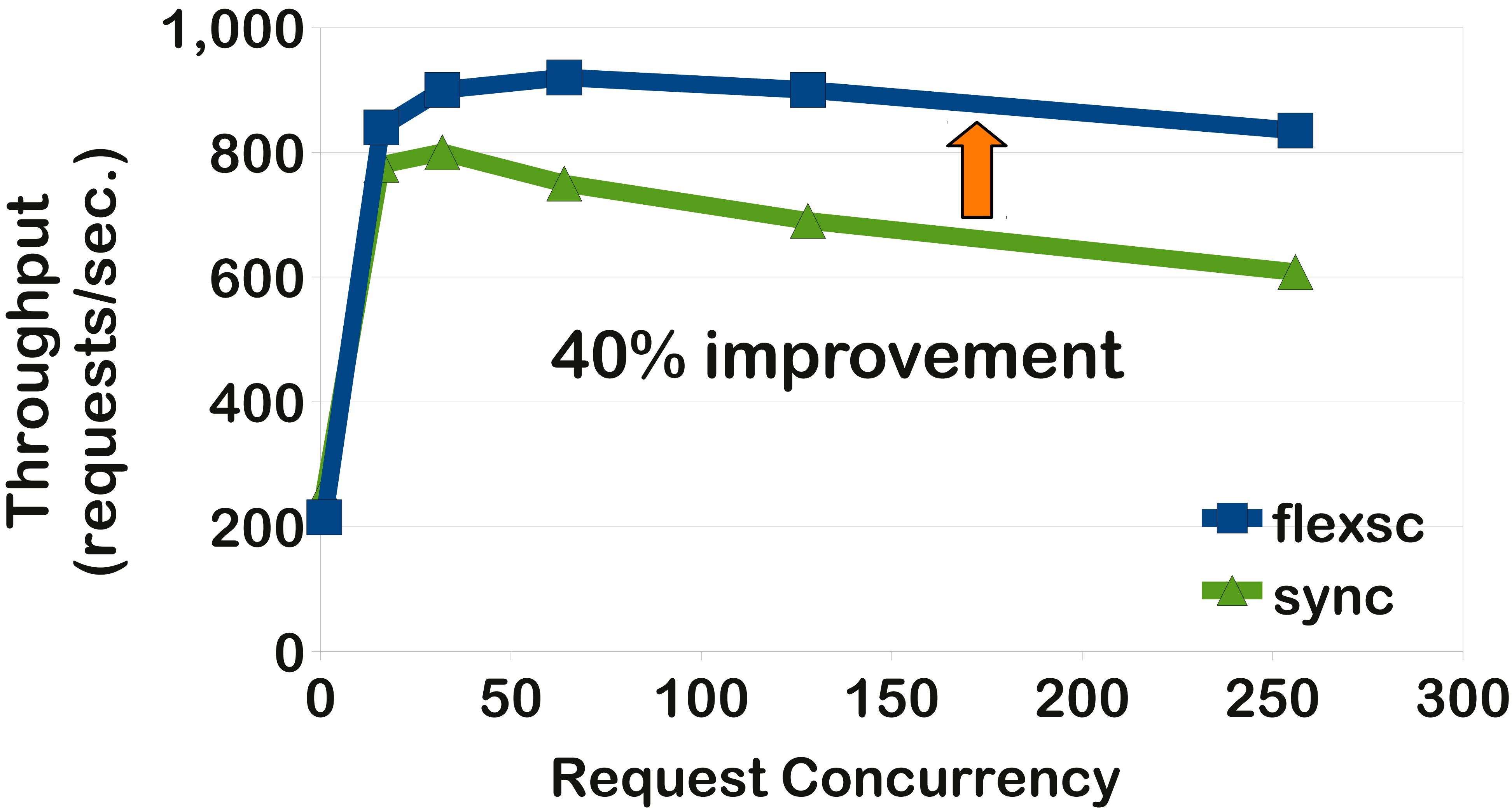
Evaluation

- Linux 2.6.33
- Nehalem (Core i7) server, 2.3GHz
  - 4 cores on a chip
- Clients connected on 1 Gbps network
- Workloads
  - Sysbench on MySQL (80% user, 20% kernel)
  - ApacheBench on Apache (50% user, 50% kernel)
- Default Linux NTPL ("**sync**") vs.
  FlexSC-Threads ("**flexsc**")

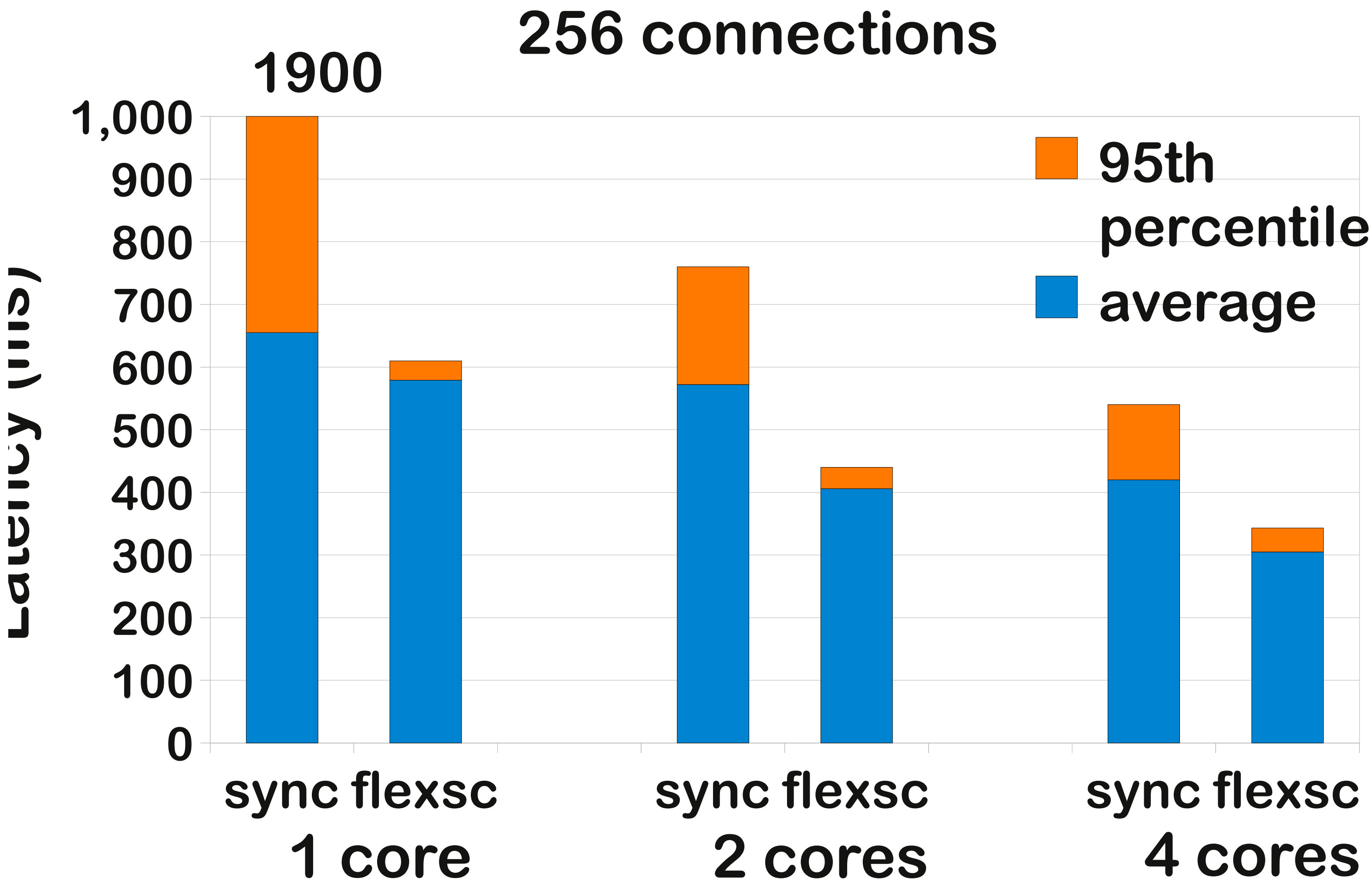
Sysbench: "OLTP" on MySQL (1 core)



Sysbench: “OLTP” on MySQL (4 cores)

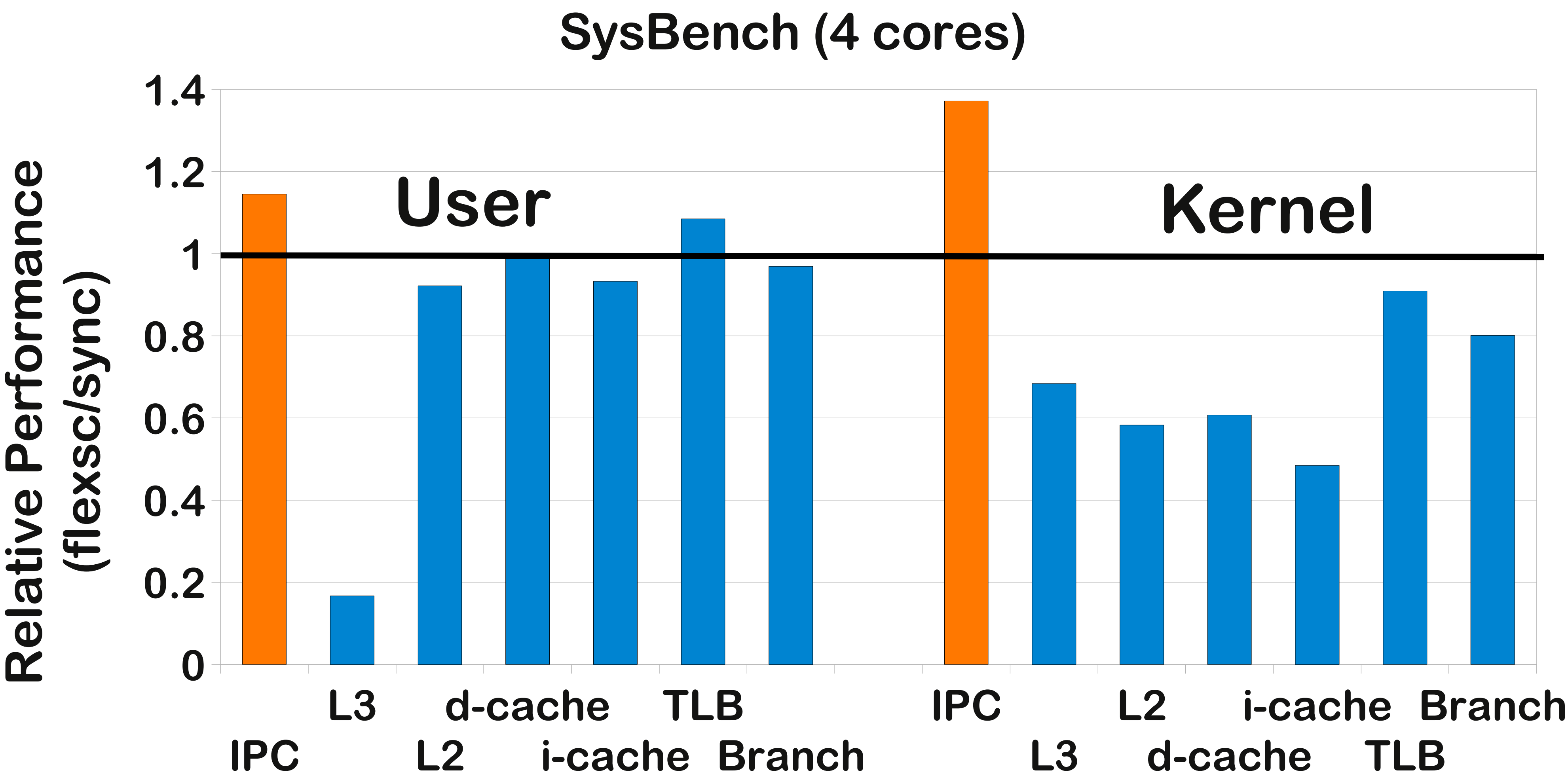


MySQL latency per client request



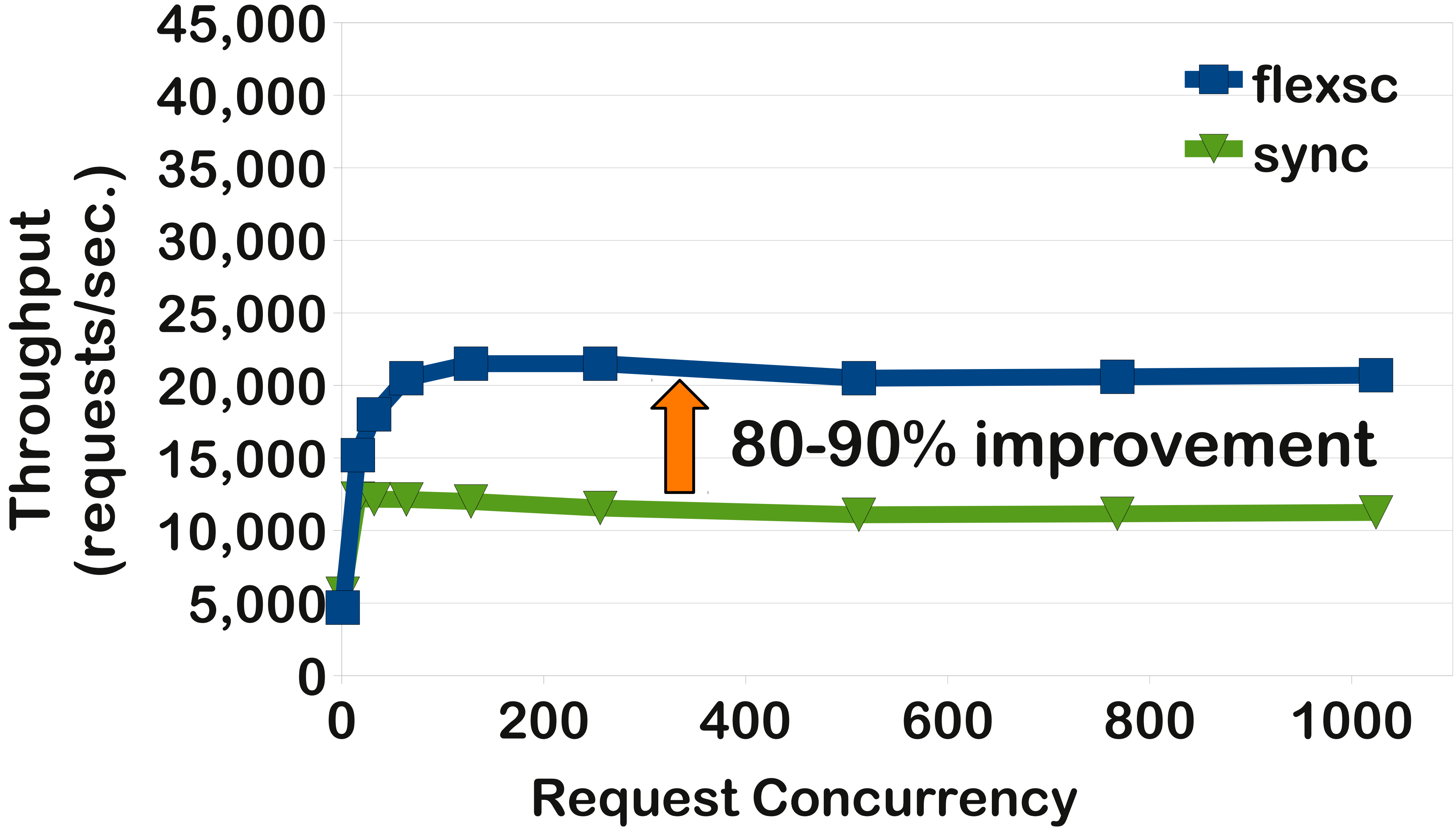
Up to 30% reduction of average request latencies

MySQL processor metrics

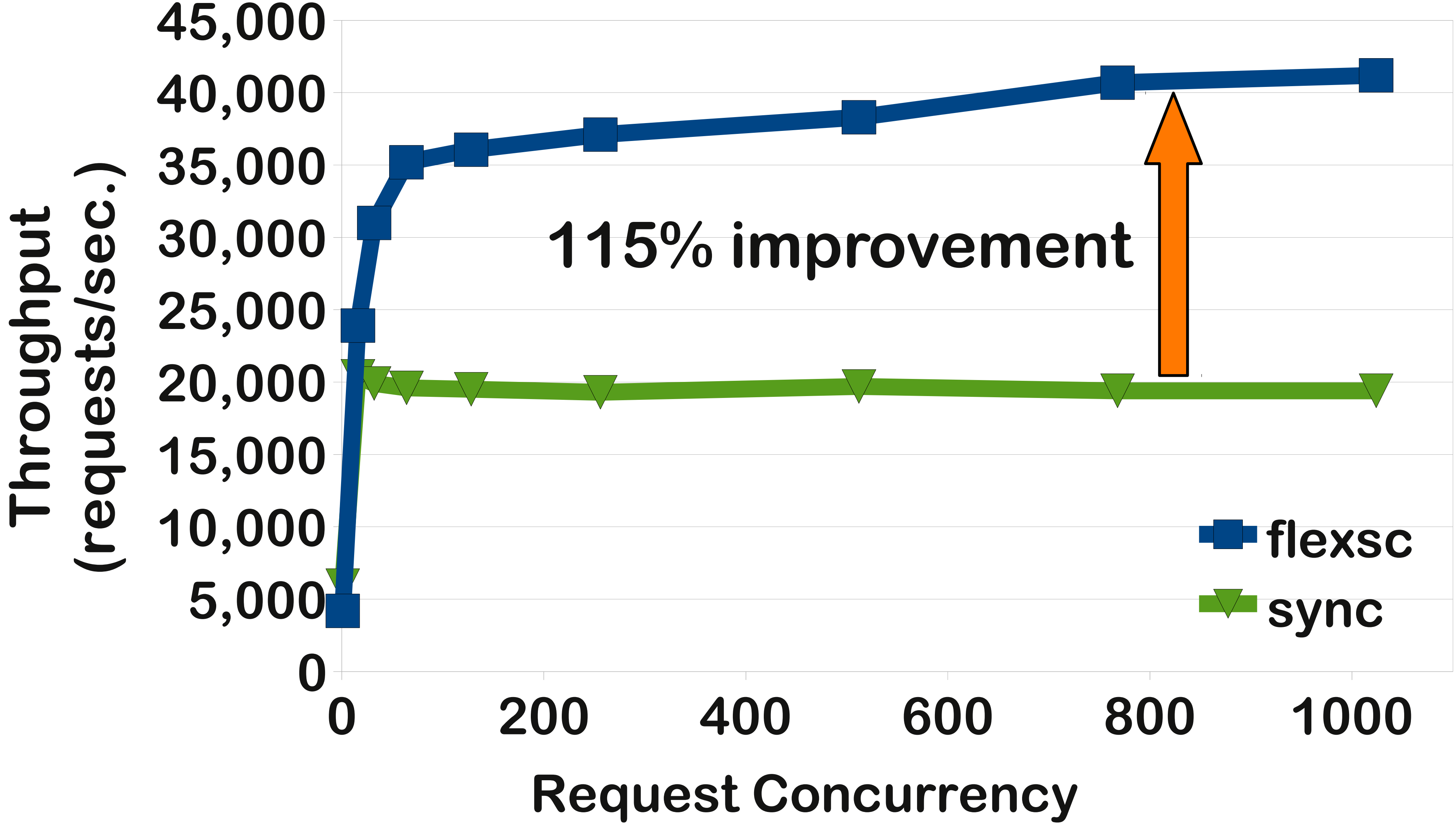


Performance improvements consequence of more efficient processor execution

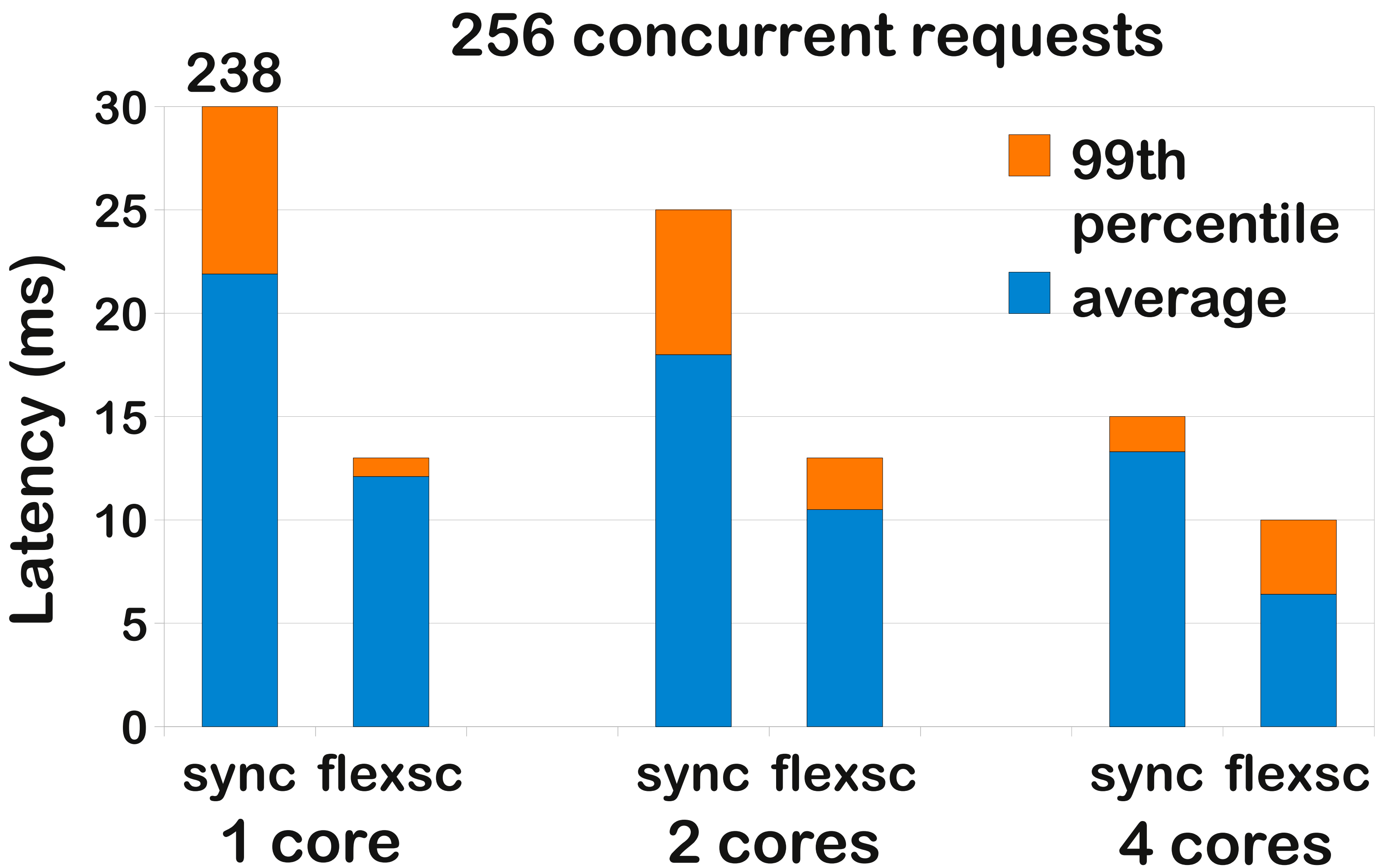
ApacheBench throughput (1 core)



ApacheBench throughput (4 cores)

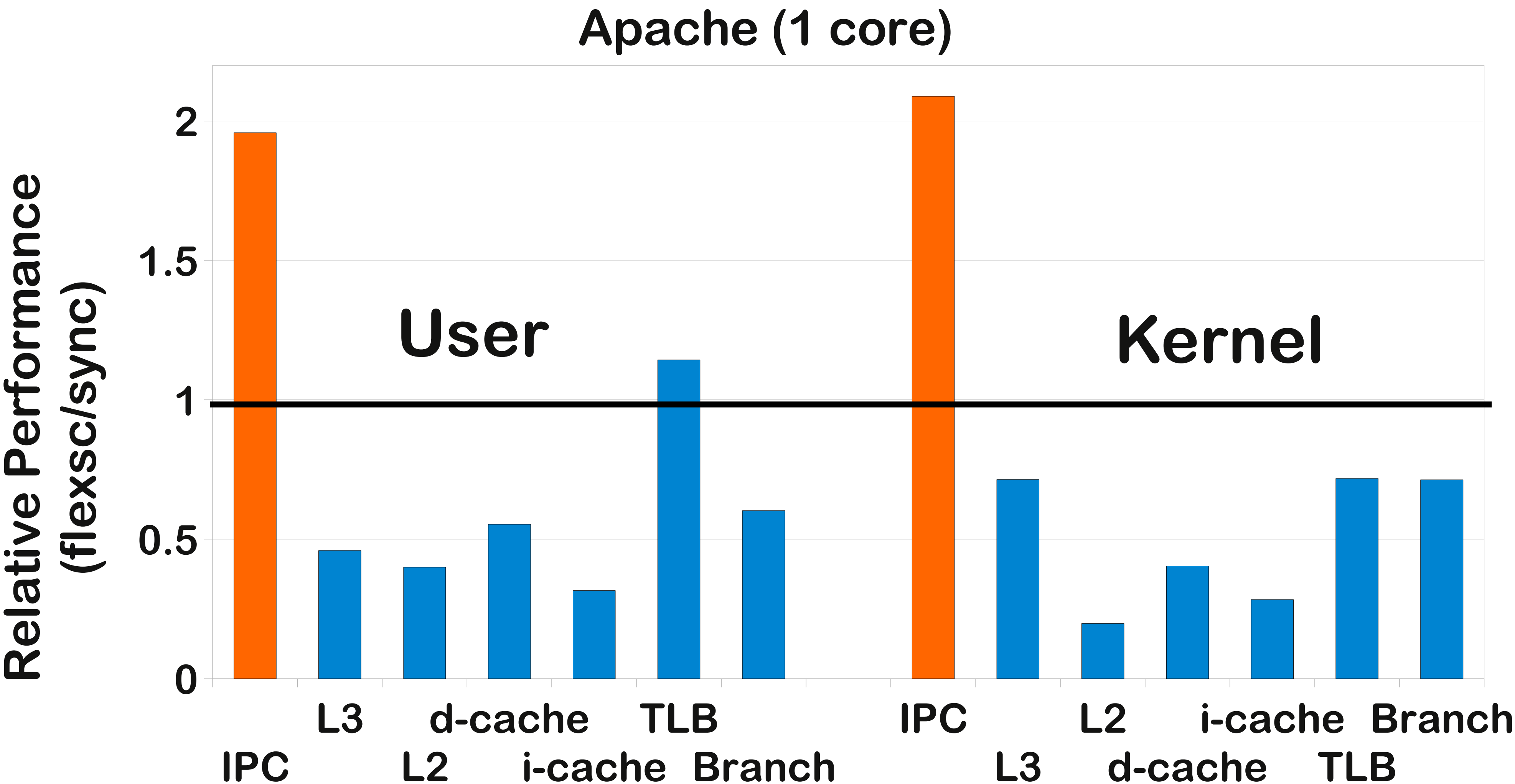


Apache latency per client request



Up to 50% reduction of average request latencies

Apache processor metrics



Processor efficiency doubles for kernel and user-mode execution

Discussion

- New OS architecture not necessary
  - Exception-less syscalls can coexist with legacy ones
- Foundation for non-blocking system calls
  - select() / poll() in user-space
  - Interesting case of non-blocking free()
- Multicore ***ultra***-specialization
  - TCP Servers (Rutgers; Iftode et.al), FS Servers
- Single-ISA asymmetric cores
  - OS-friendly cores (HP Labs; Mogul et. al)

Concluding Remarks

- System calls degrade server performance
  - Processor pollution is inherent to synchronous system calls
- **Exception-less syscalls**
  - Flexible and efficient system call execution
- **FlexSC-Threads**
  - Leverages exception-less syscalls
  - No modifications to multi-threaded applications
- Throughput & latency gains
  - 2x throughput improvement for Apache and BIND
  - 1.4x throughput improvement for MySQL

FlexSC

Flexible System Call Scheduling with Exception-Less System Calls

Livio Soares and Michael Stumm

University of Toronto