Exception-Less System Calls for Event-Driven Servers

Livio Soares and Michael Stumm

University of Toronto

Talk overview

- At OSDI'10: **exception-less system calls**
  - Technique targeted at highly threaded servers
  - Doubled performance of Apache
- Event-driven servers are popular
  - Faster than threaded servers

We show that **exception-less system calls** make event-driven server *faster*

- memcached speeds up by 25-35%
- nginx speeds up by 70-120%

Event-driven server architectures

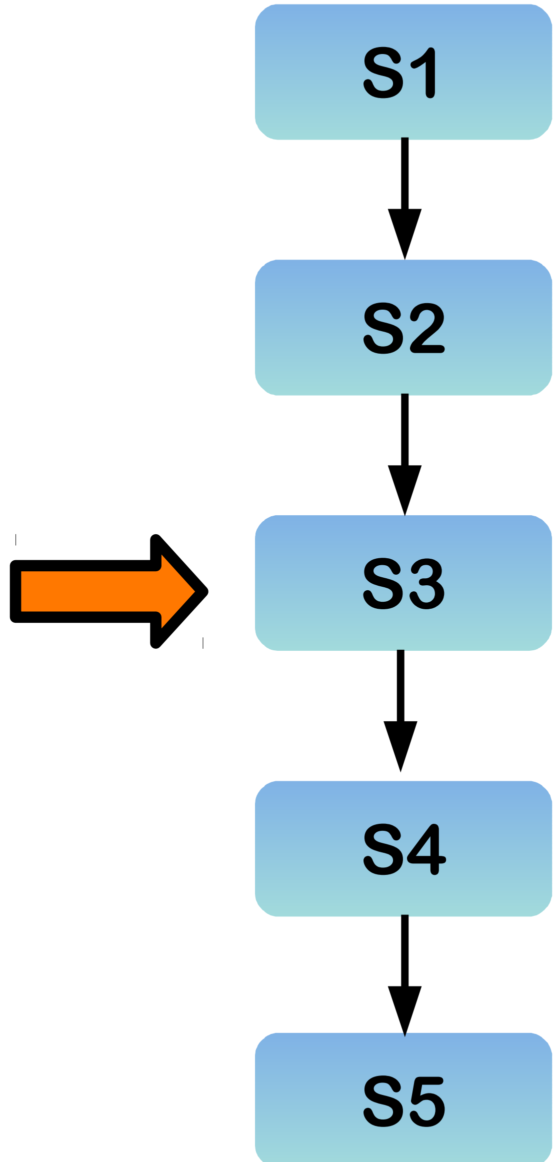
- Supports I/O concurrency with a single execution context
  - Alternative to thread based architectures
- At a high-level:
  - Divide program flow into non-blocking **stages**
  - After each stage register interest in event(s)
  - Notification of event is asynchronous, driving next stage in the program flow
  - To avoid idle time, applications multiplex execution of multiple independent stages

Example: simple network server

```
void server() {
    ...
    ...
    fd = accept();
    ...
    ...
    read(fd);
    ...
    ...
    write(fd);
    ...
    ...
    close(fd);
    ...
    ...
}
```

Example: simple network server

```
void server() {
    ...
         S1
    ...
   fd = accept();
    ...
         S2
    ...
   read(fd);
    ...
         S3
    ...
   write(fd);
    ...
         S4
    ...
   close(fd);
    ...
         S5
    ...
}
```



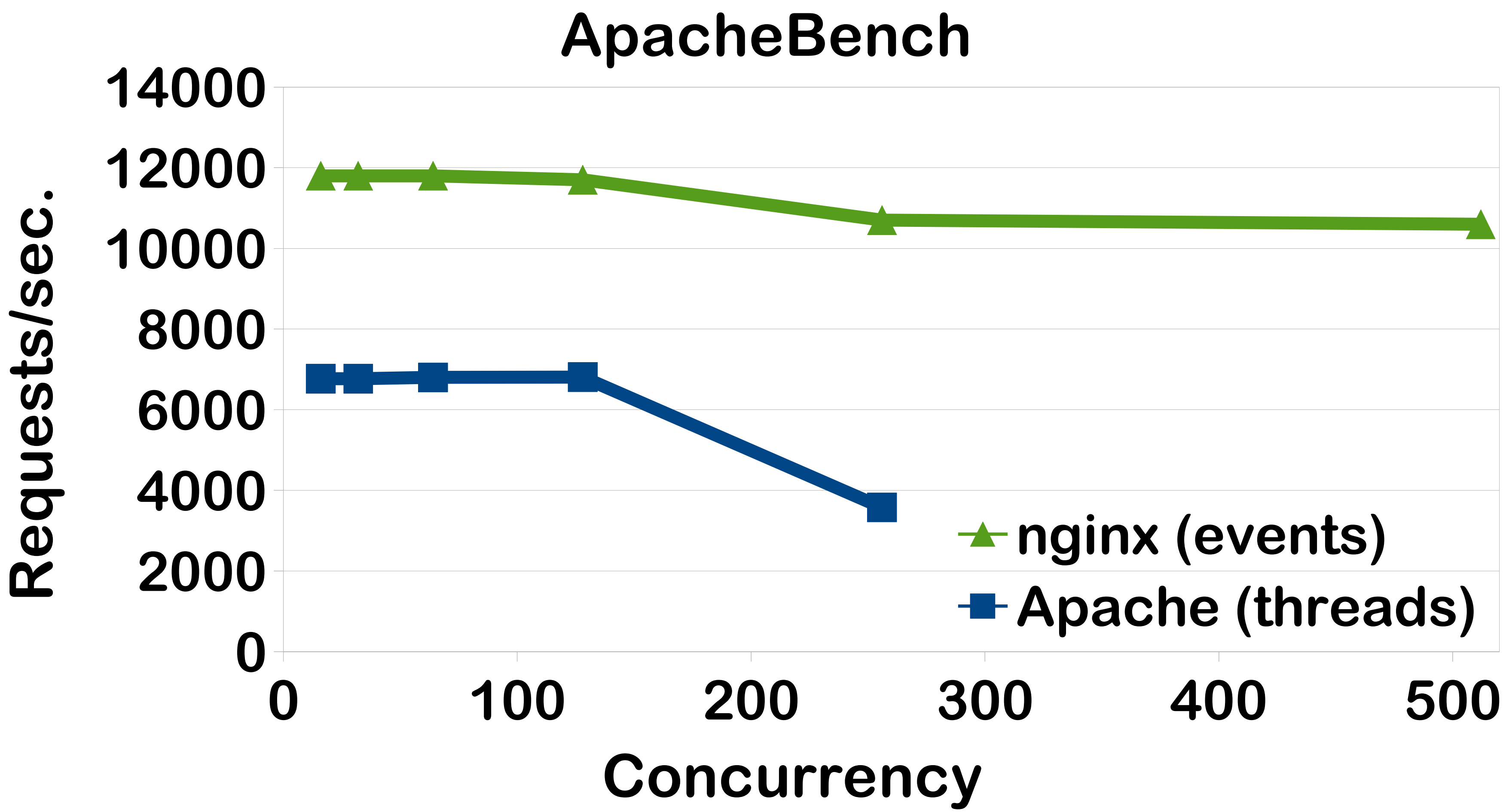
UNIX options:

Non-blocking I/O

`poll()`
`select()`
`epoll()`

Async I/O

Performance: events vs. threads



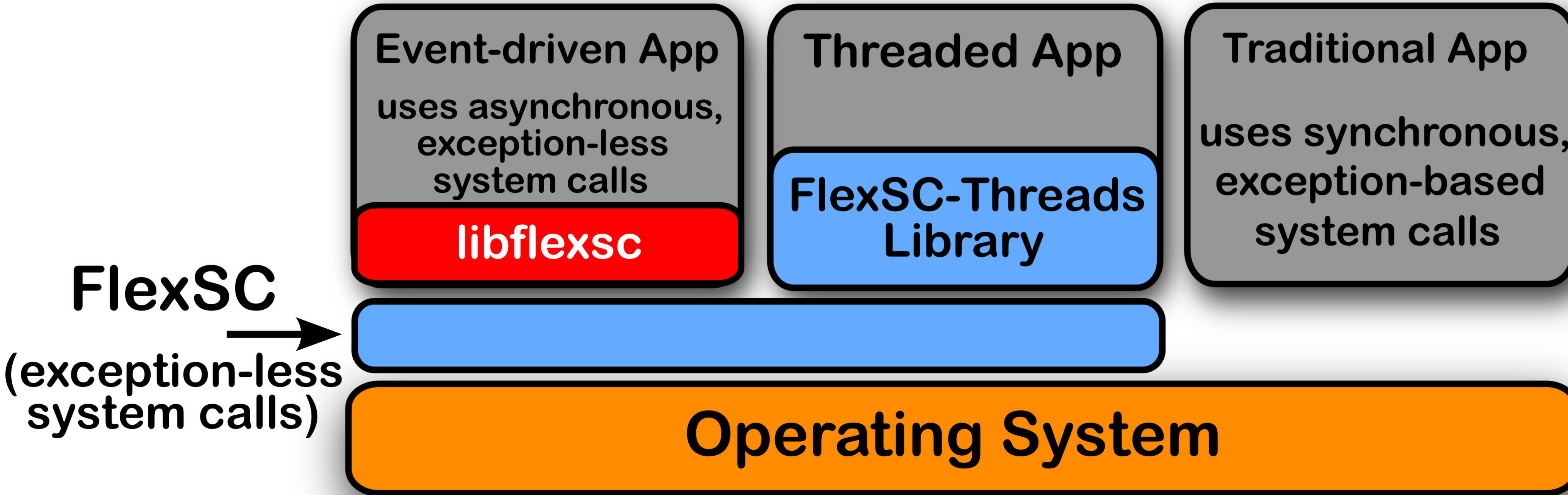
nginx delivers 1.7x the throughput of Apache; gracefully copes with high loads

Issues with UNIX event primitives

- Do not cover all system calls
  - Mostly work with file-descriptors (files and sockets)
- Overhead
  - Tracking progress of I/O involves both application and kernel code
  - Application and kernel communicate frequently

Previous work shows that fine-grain mode switching can **half** processor efficiency

FlexSC component overview



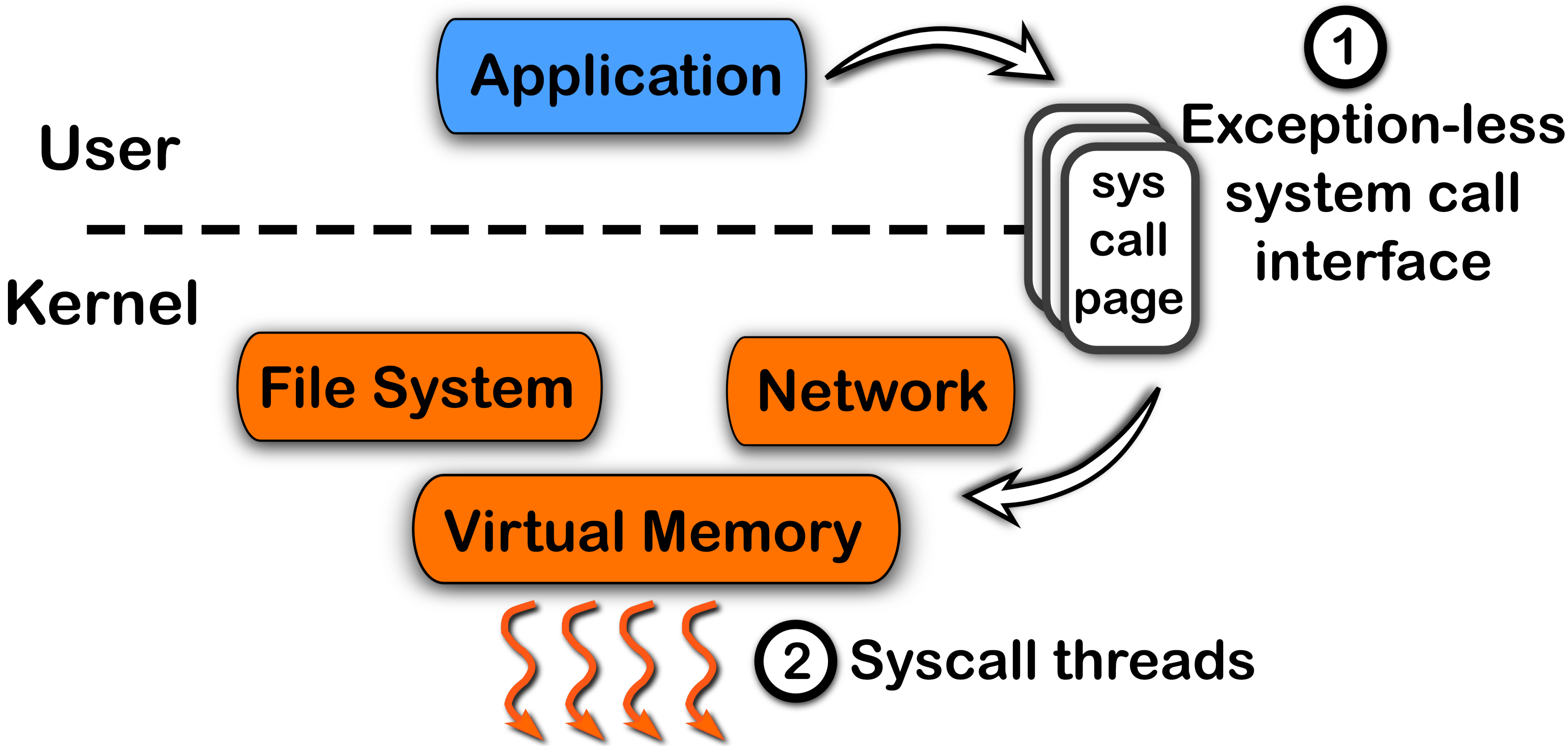
FlexSC and FlexSC-Threads presented at OSDI 2010

This work: libflexsc for event-driven servers
1) memcached throughput increase of up to 35%
2) nginx throughput increase of up to 120%

Benefits for event-driven applications

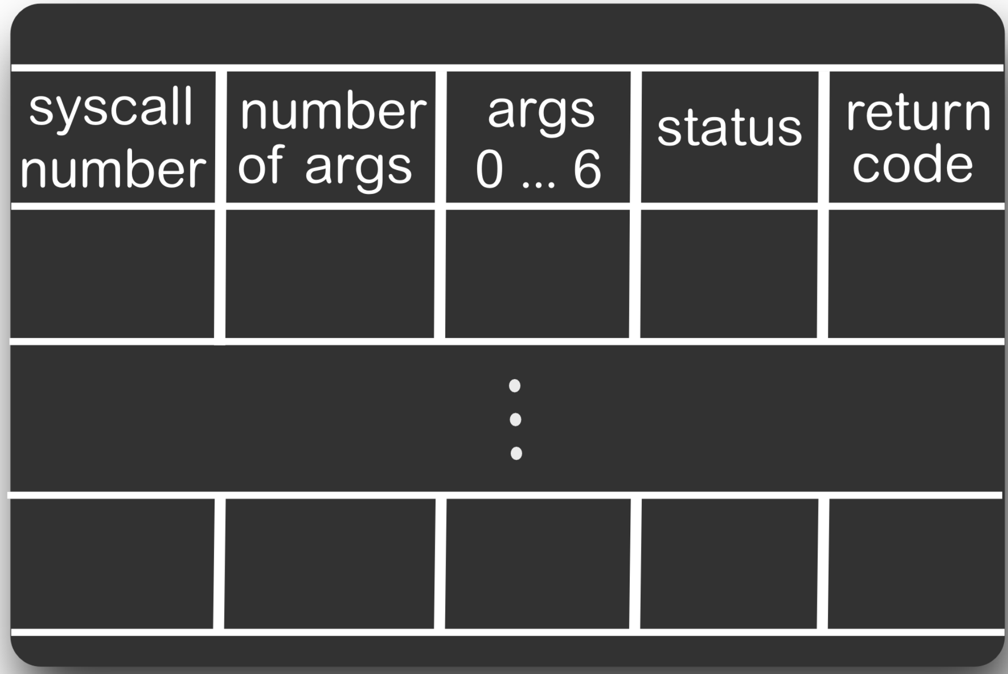
1) General purpose
   - Any/all system calls can be asynchronous
2) Non-intrusive kernel implementation
   - Does not require per syscall code
3) Facilitates multi-processor execution
   - OS work is automatically distributed
4) Improved processor efficiency
   - Reduces frequent user/kernel mode switches

Summary of exception-less syscalls



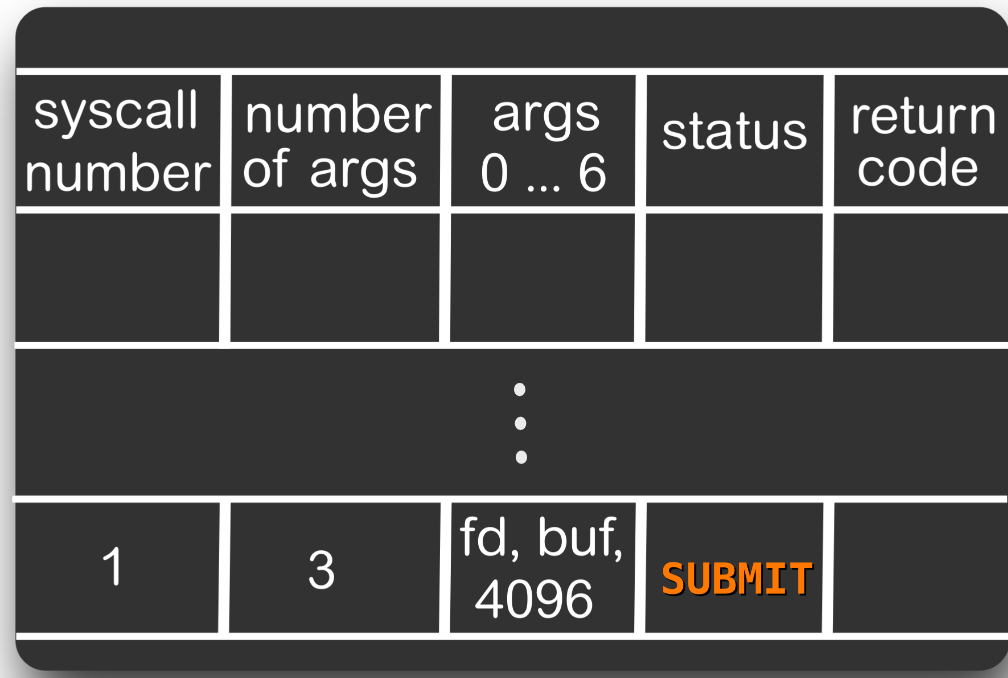
Exception-less interface: syscall page

```
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;
```



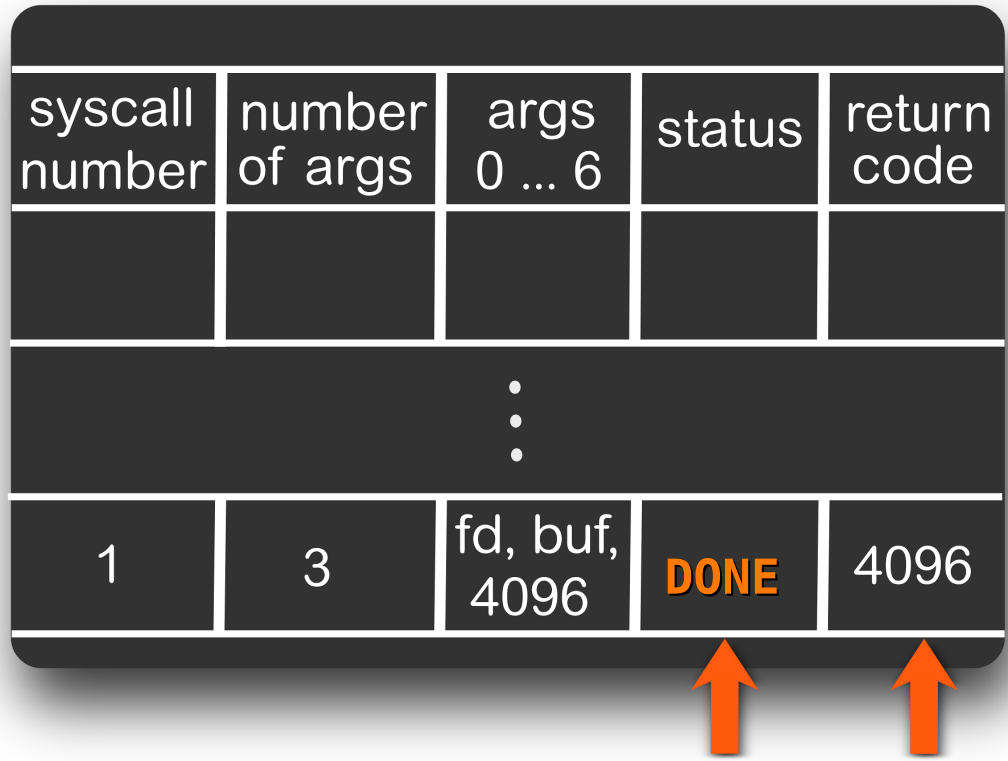
Exception-less interface: syscall page

```c
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;
```



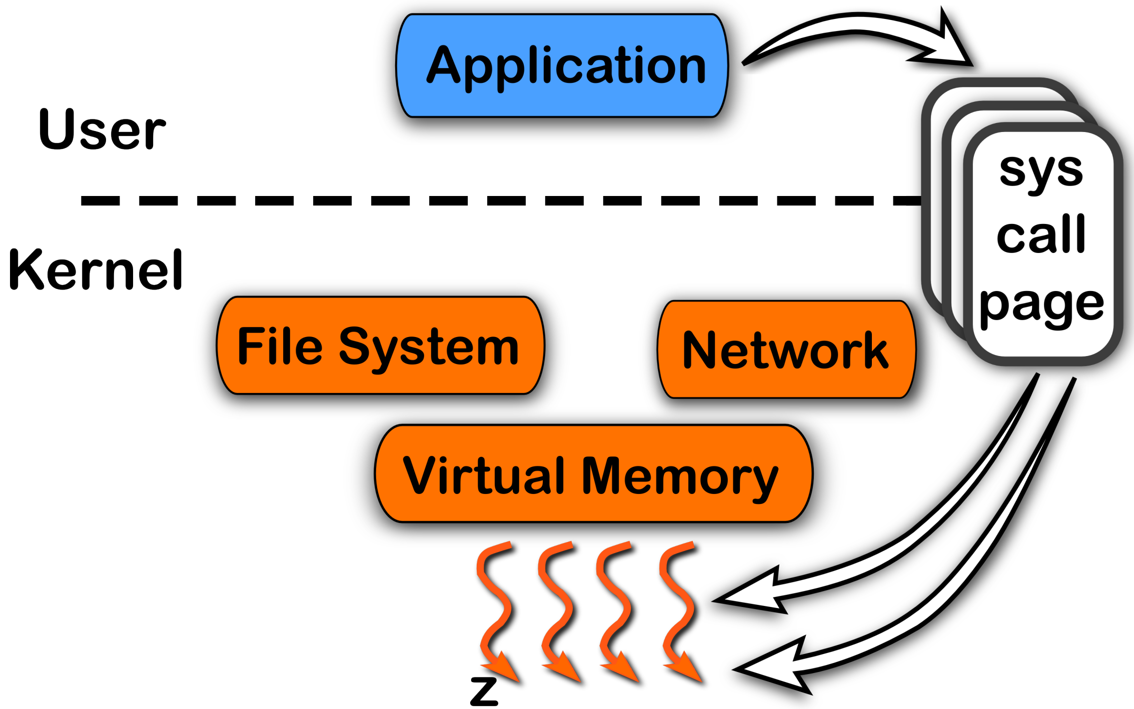
Exception-less interface: syscall page

```
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;
```

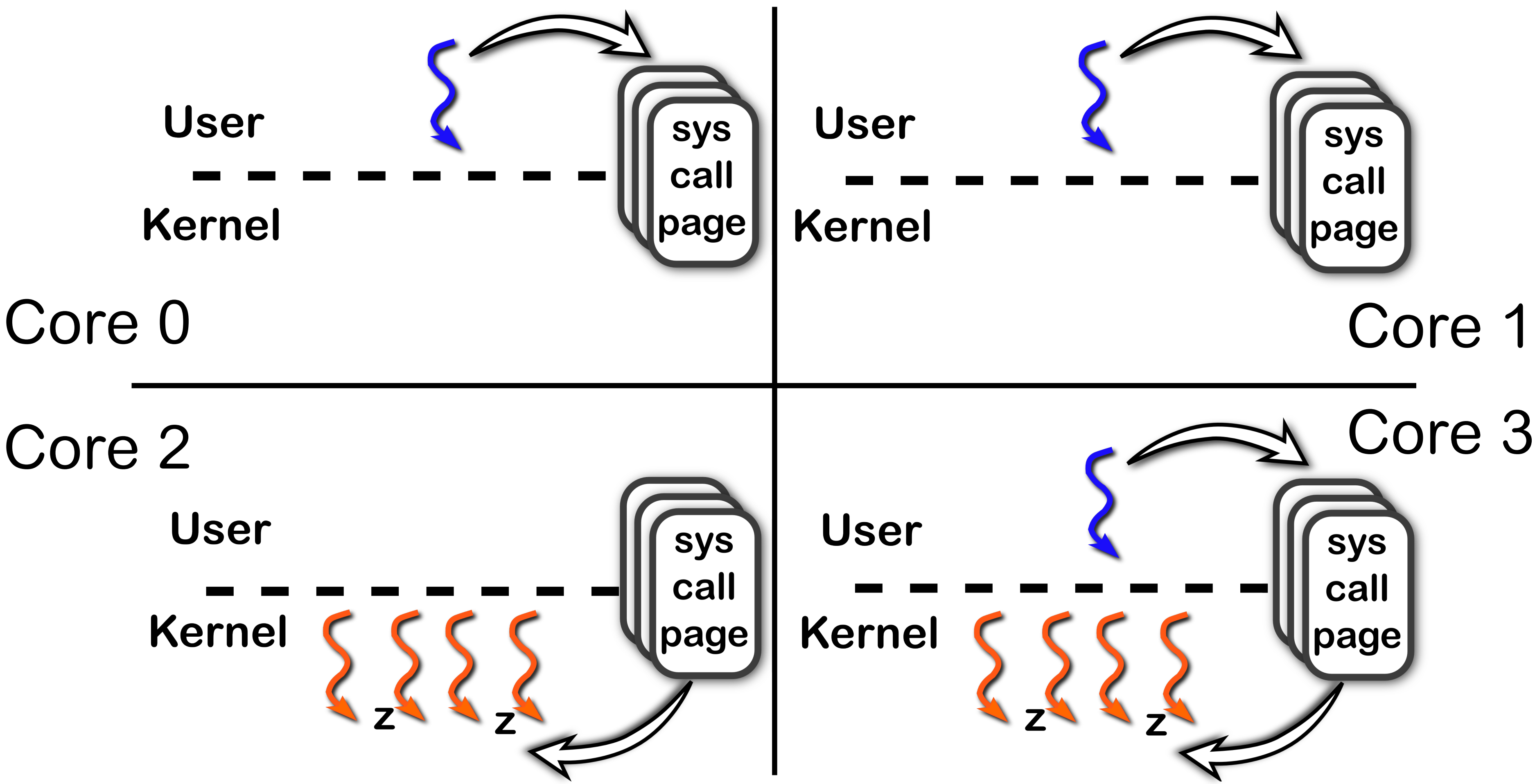


Syscall threads

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



Dynamic multicore specialization



1) FlexSC makes specializing cores simple
2) Dynamically adapts to workload needs

libflexsc: async syscall library

- Async syscall and notification library
- Similar to *libevent*
  - But operates on syscalls instead of file-descriptors
- Three main components:

  1) Provides main loop (dispatcher)

  2) Support asynchronous syscall with associated callback to notify completion

  3) Cancellation support

Main API: async system call

```
struct flexsc_cb {
    void (*callback)(struct flexsc_cb *); /* event handler */
    void *arg;                            /* auxiliary var */
    int64_t ret;                          /* syscall return */
}

int flexsc_##SYSCALL(struct flexsc_cb *, ... /*syscall args*/);
```

```
/* Example: asynchronous accept */
struct flexsc_cb cb;
cb.callback = handle_accept;
flexsc_accept(&cb, master_sock, NULL, 0);

void handle_accept(struct flexsc_cb *cb) {
   int fd = cb->ret;
   if (fd != -1) {
       struct flexsc_cb read_cb;
       read_cb.callback = handle_read;
       flexsc_read(&read_cb, fd, read_buf, read_count);
   }
}
```

memcached port to libflexsc

- memcached: in-memory key/value store
  - Simple code-base: 8K LOC
  - Uses libevent

- Modified 293 LOC
- Transformed libevent calls to libflexsc
- Mostly in one file: memcached.c
- Most memcached syscalls are socket based

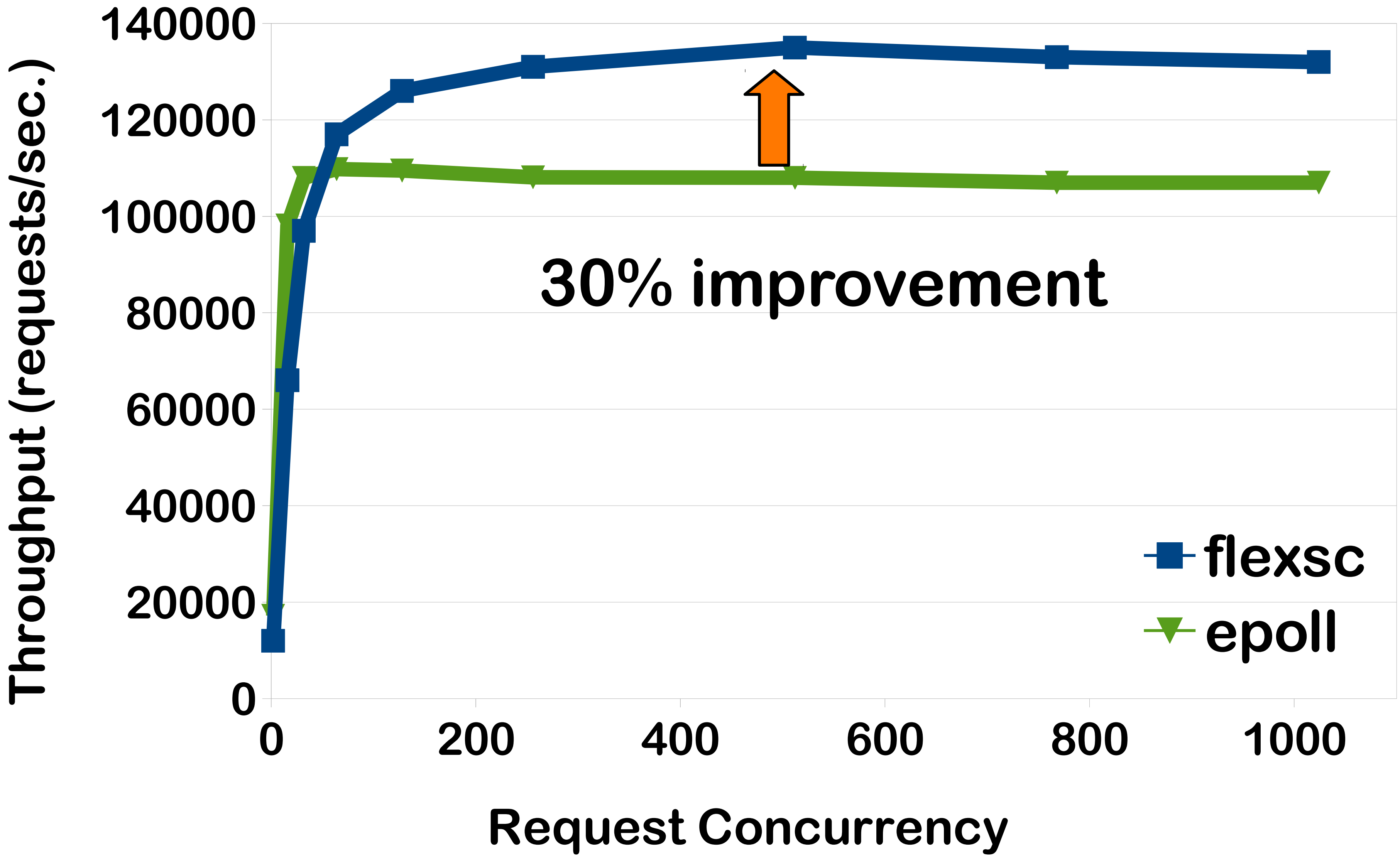
nginx port to libflexsc

- Most popular event-driven webserver
  - Code base: 82K LOC
  - Natively uses both non-blocking (epoll) I/O and asynchronous I/O
- Modified 255 LOC
- Socket based code already asynchronous
- Not all file-system calls were asynchronous
  - e.g., open, fstat, getdents
- Special handling of stack allocated syscall args

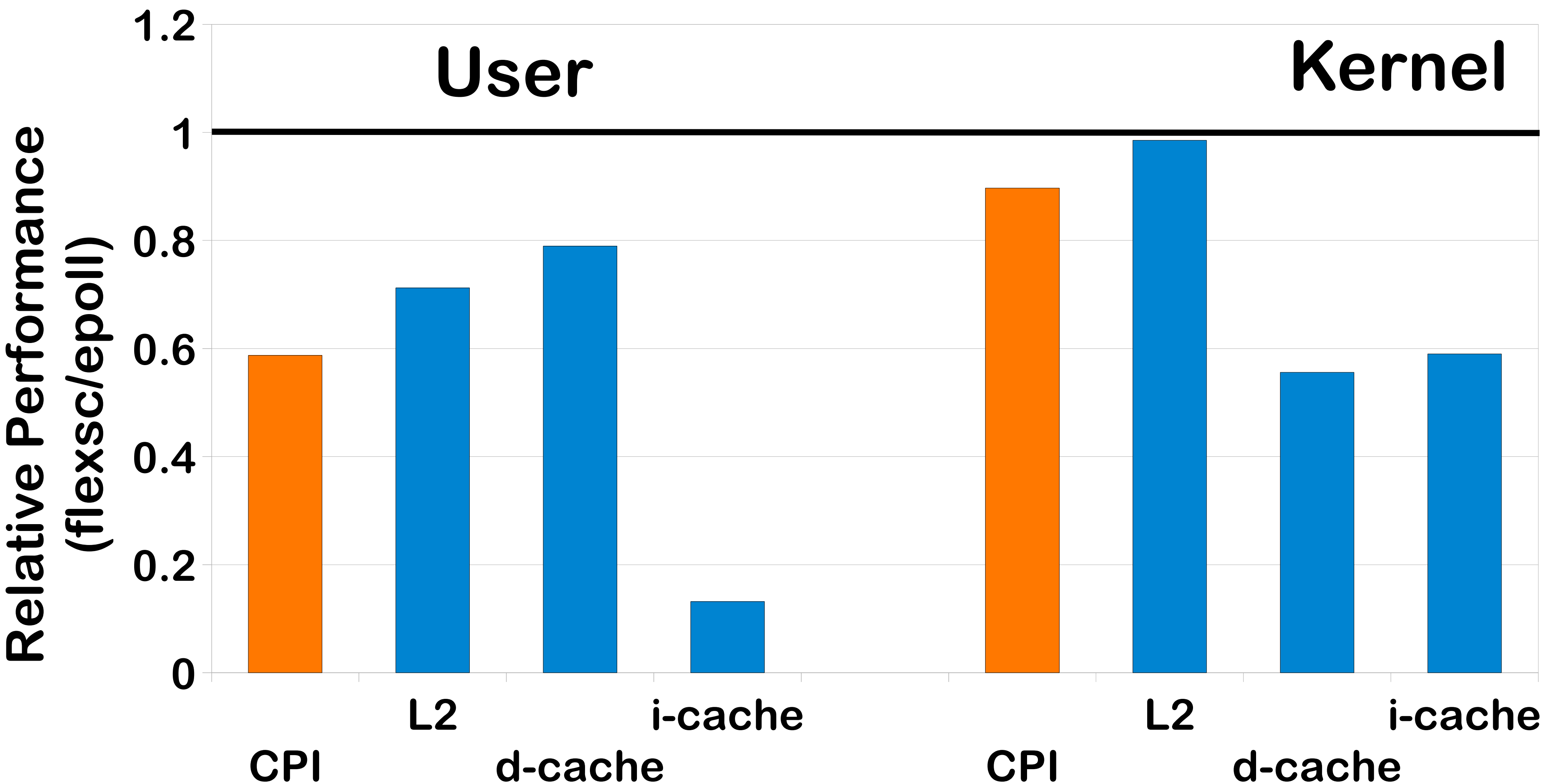
Evaluation

- Linux 2.6.33
- Nehalem (Core i7) server, 2.3GHz
  - 4 cores
- Client connected through 1Gbps network
- Workloads
  - memslap on memcached (30% user, 70% kernel)
  - httperf on nginx (25% user, 75% kernel)
- Default Linux (“**epoll**”) vs.
  libflexsc (“**flexsc**”)

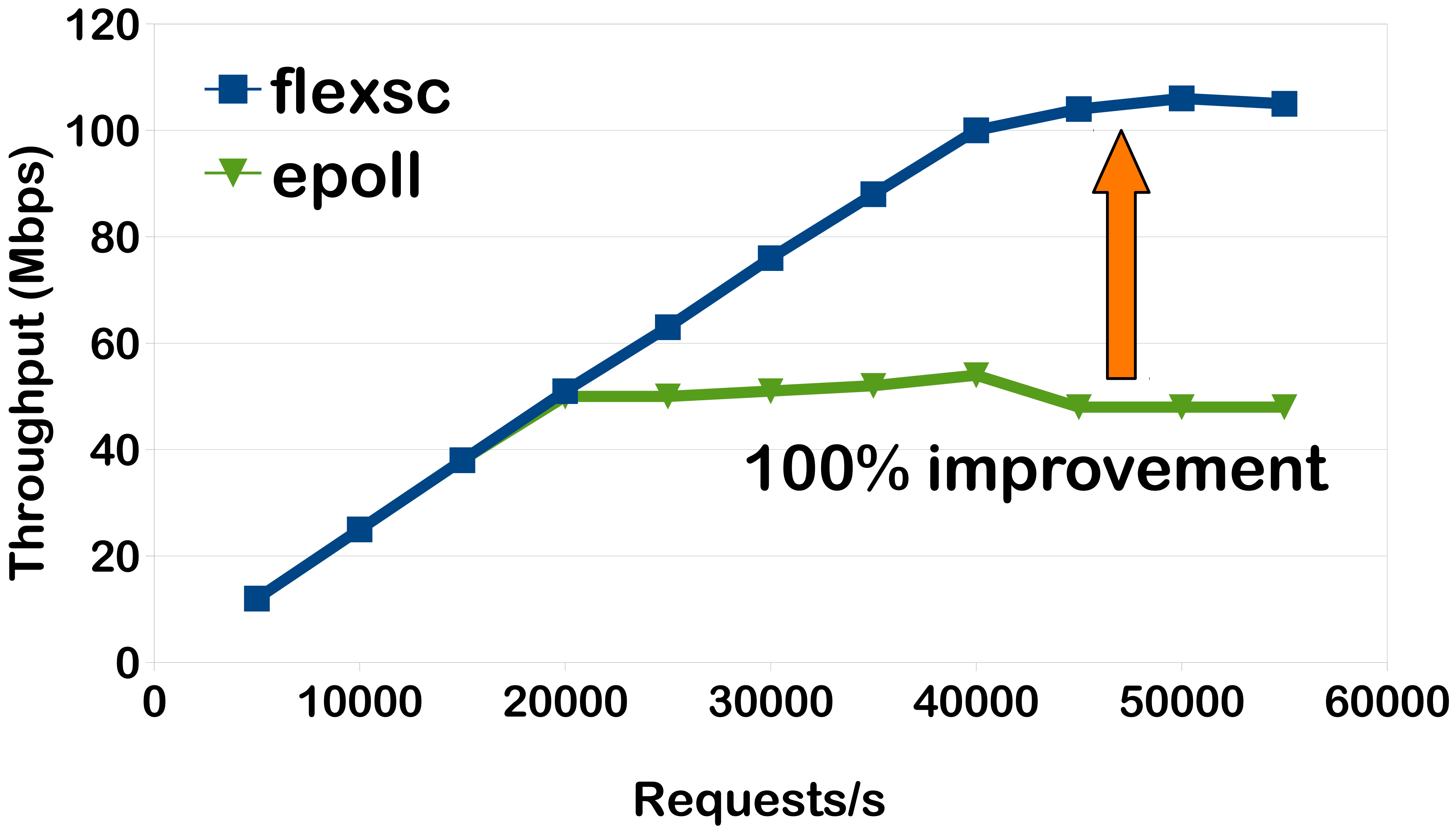
memcached on 4 cores



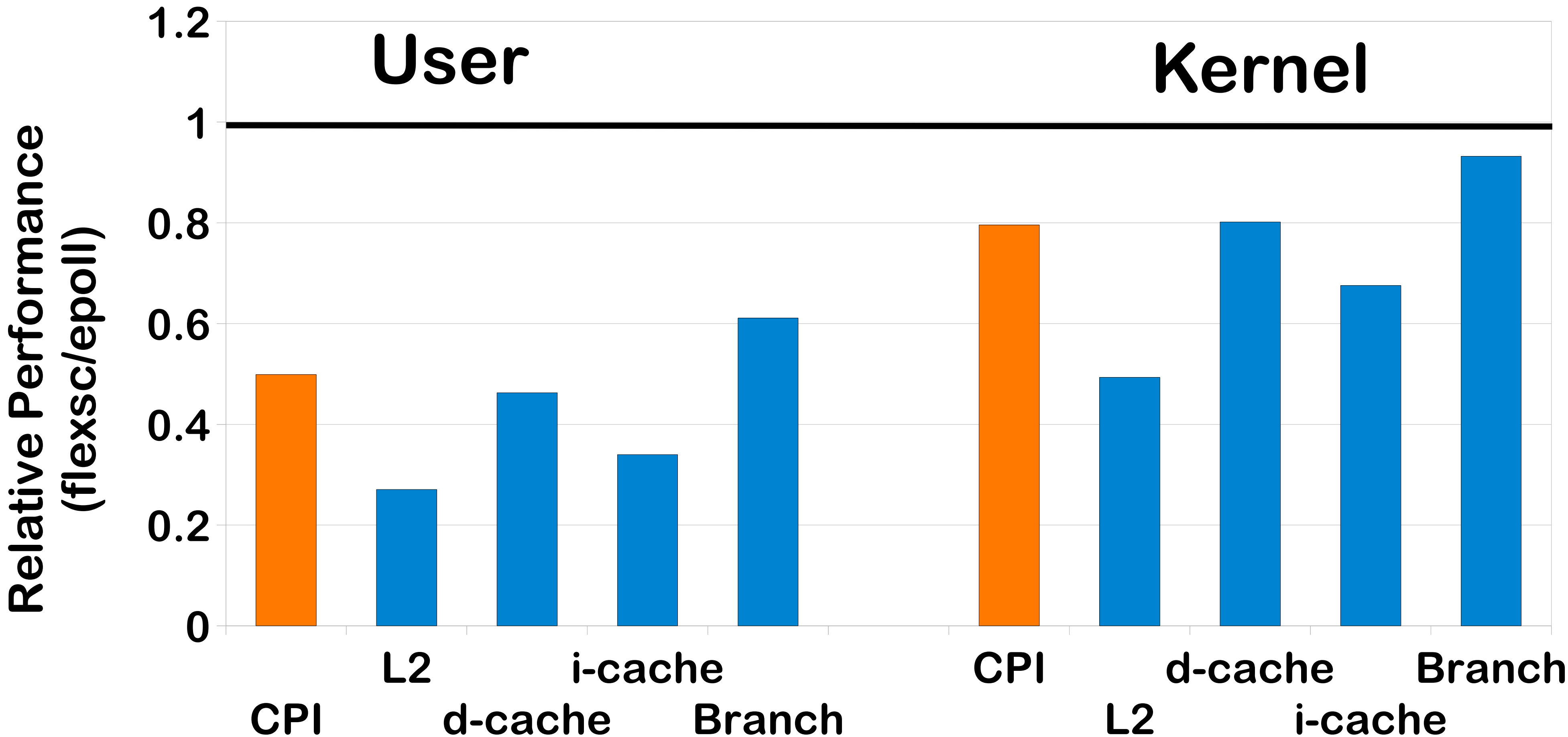
memcached processor metrics



httperf on nginx (1 core)



nginx processor metrics



Concluding remarks

- Current event-based primitives add overhead
  - I/O operations require frequent communication between OS and application
- **libflexsc**: exception-less syscall library
  1) General purpose
  2) Non-intrusive kernel implementation
  3) Facilitates multi-processor execution
  4) Improved processor efficiency
- Ported memcached and nginx to libflexsc
  - Performance improvements of 30 - 120%

Exception-Less System Calls for Event-Driven Servers

Livio Soares and Michael Stumm

University of Toronto

Backup Slides

Difference in improvements

Why does nginx improve more than memcached?

1) Frequency of mode switches:

Server	memcached	nginx
Frequency of syscalls (in instructions)	3,750	1,460

2) nginx uses greater diversity of system calls

- ➔ More interference in processor structures (caches)

3) Instruction count reduction

- ➔ nginx with epoll() has connection timeouts

Limitations

- Scalability (number of outstanding syscalls)
  - Interface: operations perform linear scan
  - Implementation: overheads of syscall threads non-negligible
- Solutions
  - Throttle syscalls at application or OS
  - Switch interface to scalable message passing
  - Provide exception-less versions of async I/O
  - Make kernel fully non-blocking

Latency (ApacheBench)