Rethink the Sync

Ed Nightingale
Kaushik Veeraraghavan
Peter Chen
Jason Flinn

University of Michigan

Problem

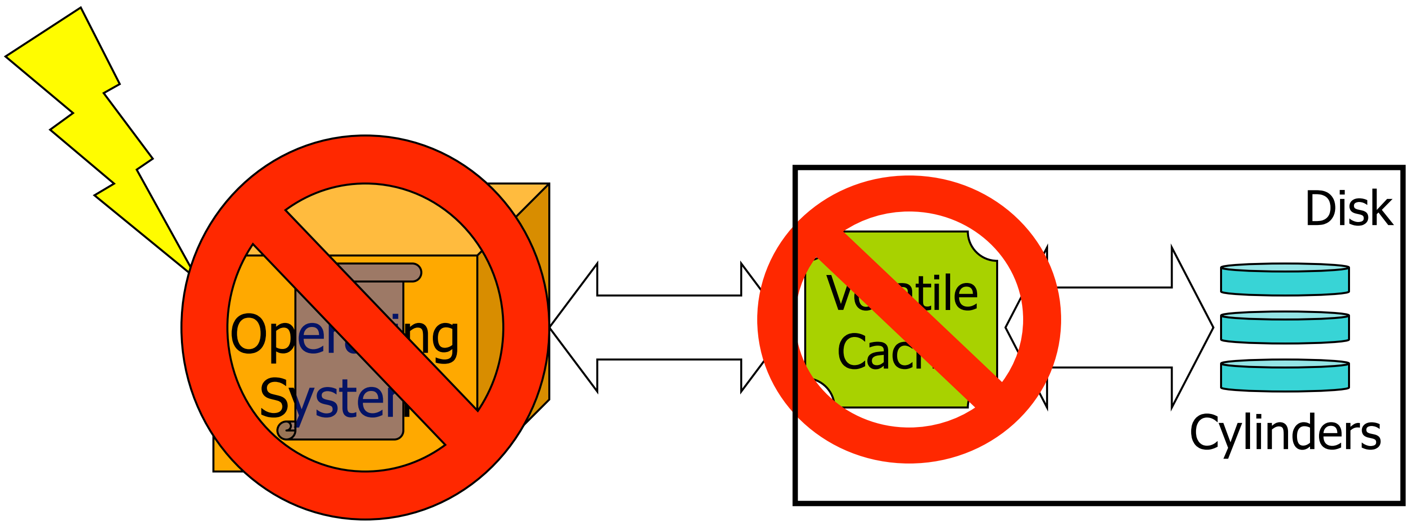
- Asynchronous I/O is a poor abstraction for:
  - Reliability
  - Ordering
  - Durability
  - Ease of programming
- Synchronous I/O is superior but 100x slower
  - Caller blocked until operation is complete

Solution

- Synchronous I/O can be fast
- New model for synchronous I/O
  - External synchrony
  - Same guarantees as synchronous I/O
  - Only 8% slower than asynchronous I/O

When a sync() is really async

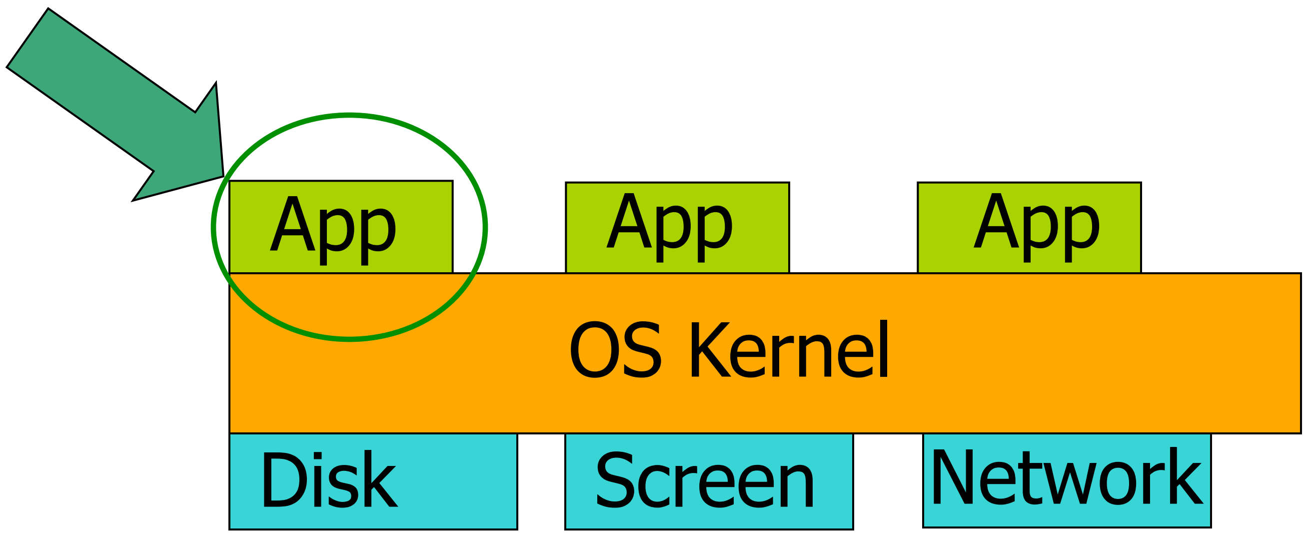
- On sync() data written only to volatile cache
  - 10x performance penalty and data NOT safe



- 100x slower than asynchronous I/O if disable cache

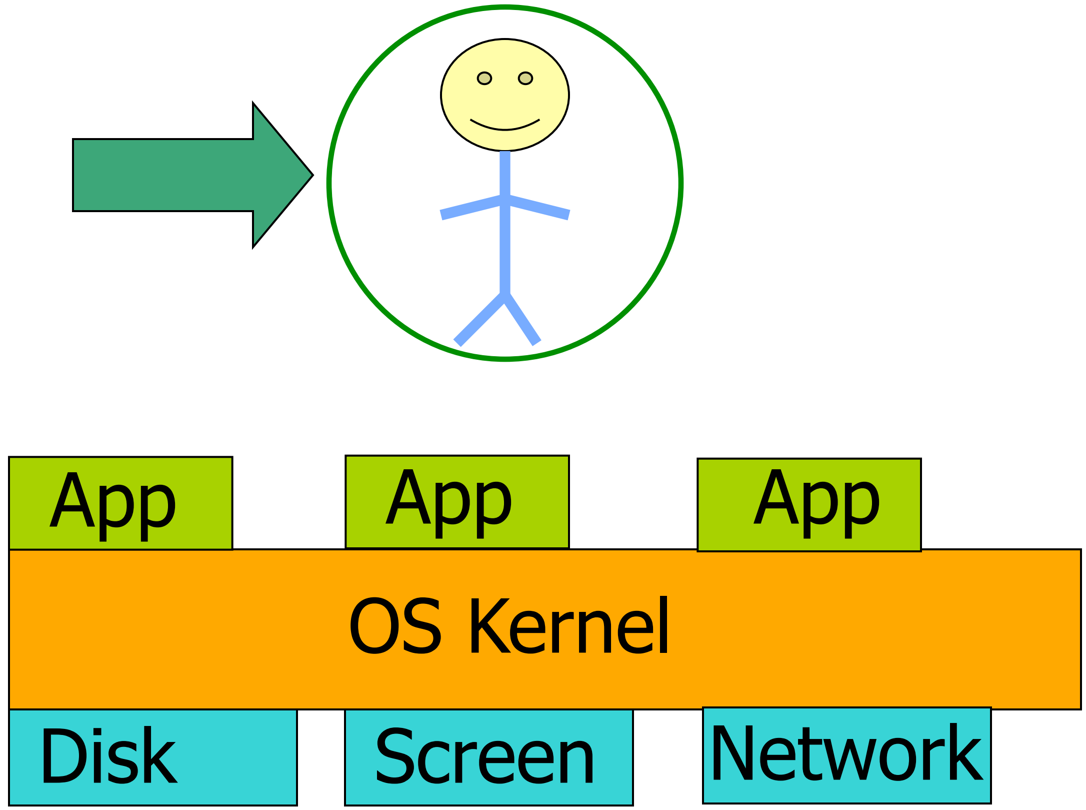
To whom are guarantees provided?

- Synchronous I/O definition:
  - Caller blocked until operation completes



- Guarantee provided to application

To whom are guarantees provided?

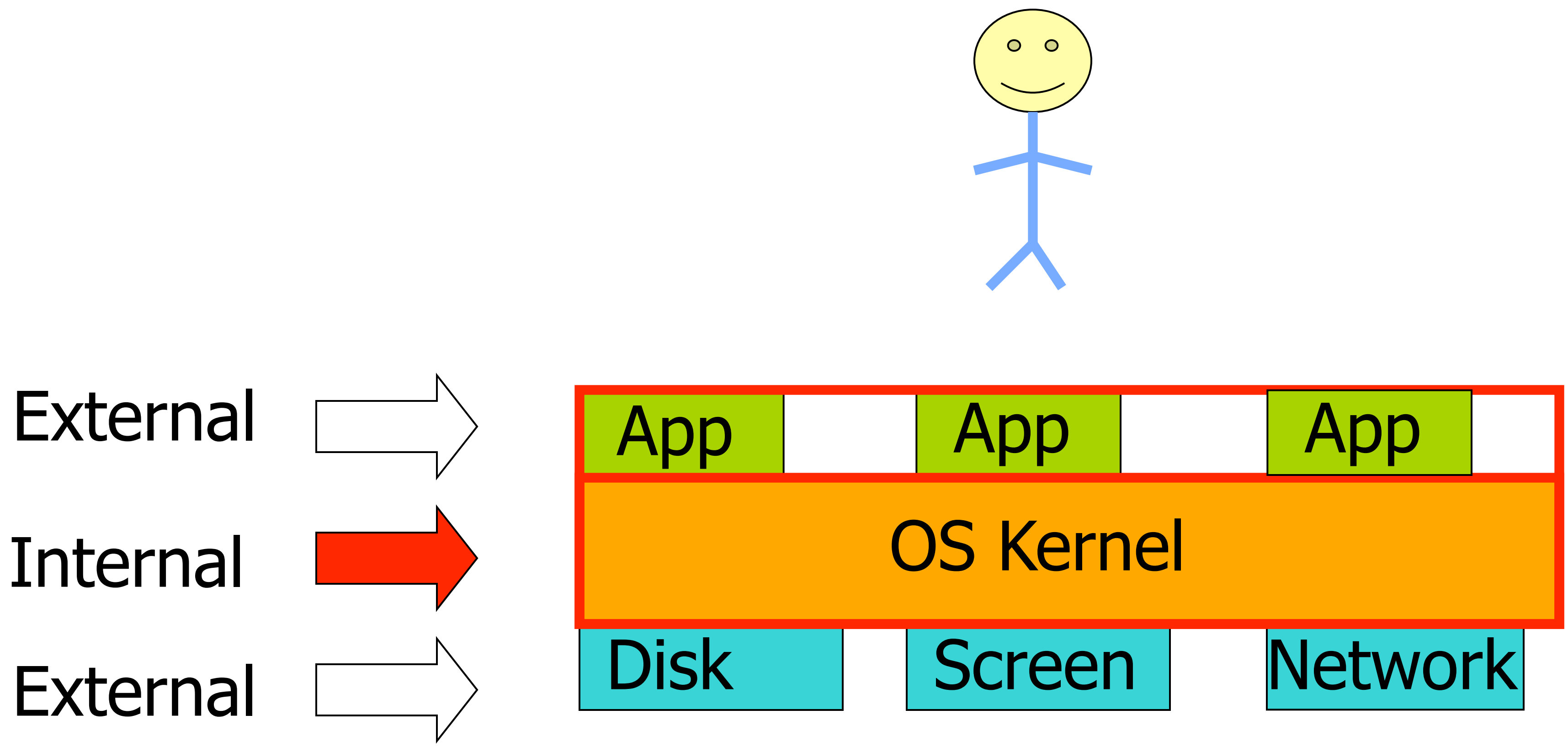


- Guarantee really provided to the user

Providing the user a guarantee

- User *observes* operation has completed
  - User may examine screen, network, disk…
- Guarantee provided by synchronous I/O
  - Data durable when operation observed to complete
- To observe output it must be externally visible
  - Visible on external device

Why do applications block?



- Application is external therefore block on syscall
- Since application is internal no need to block

A new model of synchronous I/O

- Provide guarantee directly to user
  - Rather than via application
- Called externally synchronous I/O
  - Indistinguishable from traditional sync I/O
  - Approaches speed of asynchronous I/O

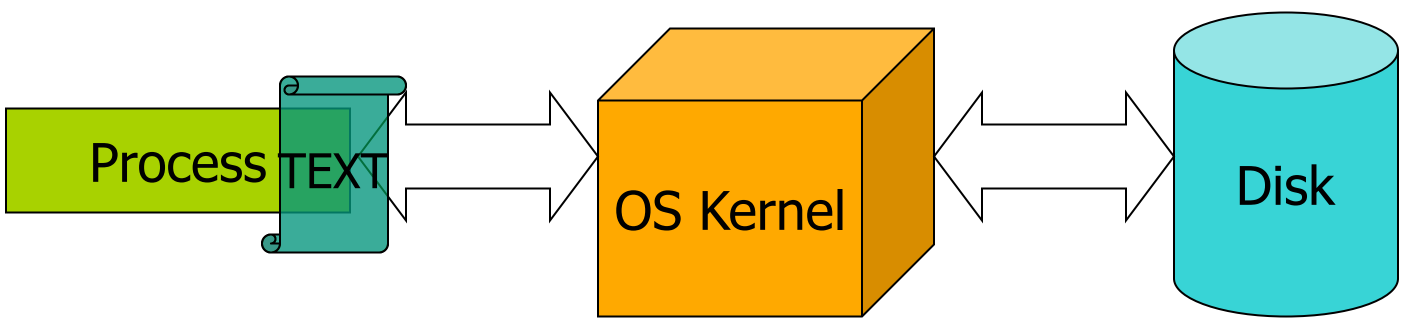
Example: Synchronous I/O

```
write(buf_1);
write(buf_2);
print("work done");
foo();
```

Application blocks

Application blocks

```
%work done
%
```



Observing synchronous I/O

```
write(buf_1);
write(buf_2);
print("work done");
foo();
```

Depends on 1st write

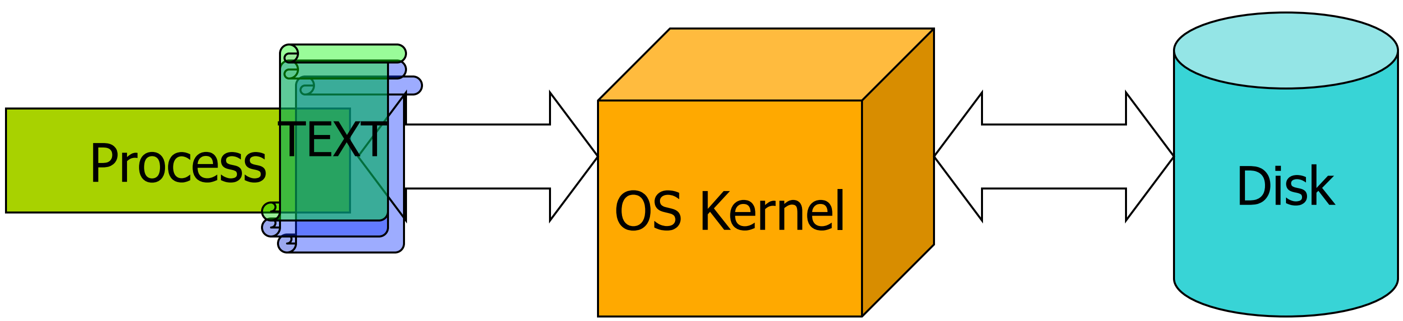
Depends on 1st & 2nd write

- Sync I/O externalizes output based on causal ordering
  - Enforces causal ordering by blocking an application
- Ext sync: Same causal ordering without blocking applications

Example: External synchrony

```
write(buf_1);
write(buf_2);
print("work done");
foo();
```

```
%work done
%
```



Tracking causal dependencies

- Applications may communicate via IPC
  - Socket, pipe, fifo etc.
- Need to propagate dependencies through IPC
- We build upon Speculator [SOSP '05]
  - Track and propagate causal dependencies
  - Buffer output to screen and network

Tracking causal dependencies

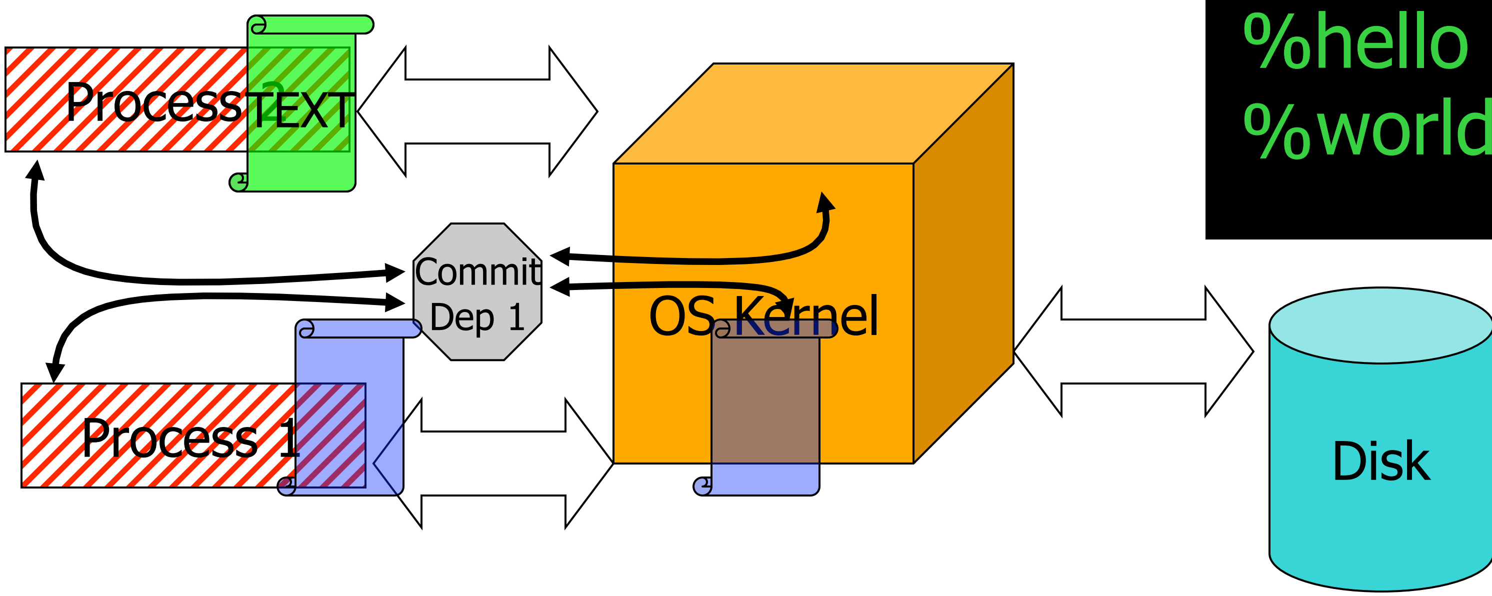
Process 1

```
101   write(file1);
102   do_something();
```

Process 2

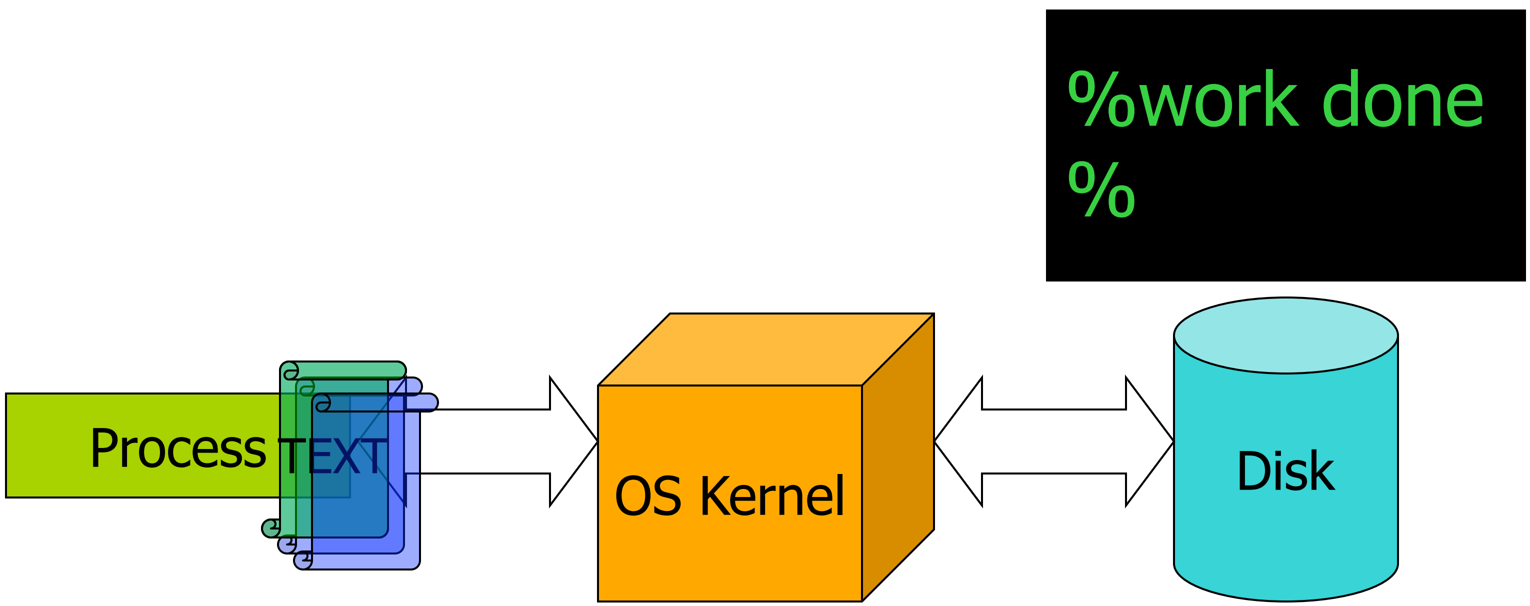
```
101   print ("hello");
102   read(file1);
103   print("world");
```

%hello
%world



Output triggered commits

- Maximize throughput until output buffered
- When output buffered, trigger commit
  - Minimize latency only when important



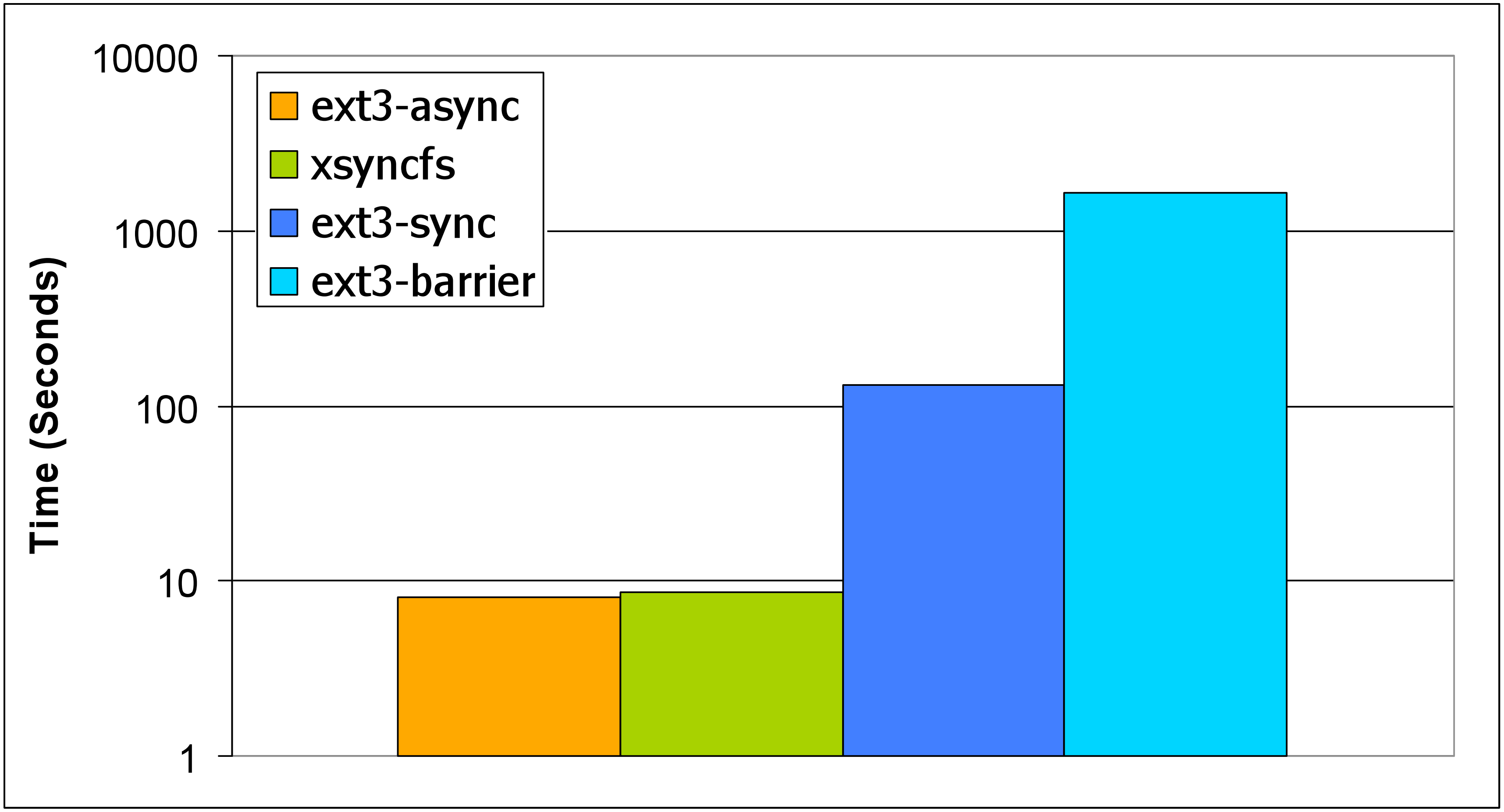
Evaluation

- Implemented ext sync file system Xsyncfs
  - Based on the ext3 file system
  - Use journaling to preserve order of writes
  - Use write barriers to flush volatile cache
- Compare Xsyncfs to 3 other file systems
  - Default asynchronous ext3
  - Default synchronous ext3
  - Synchronous ext3 with write barriers

When is data safe?

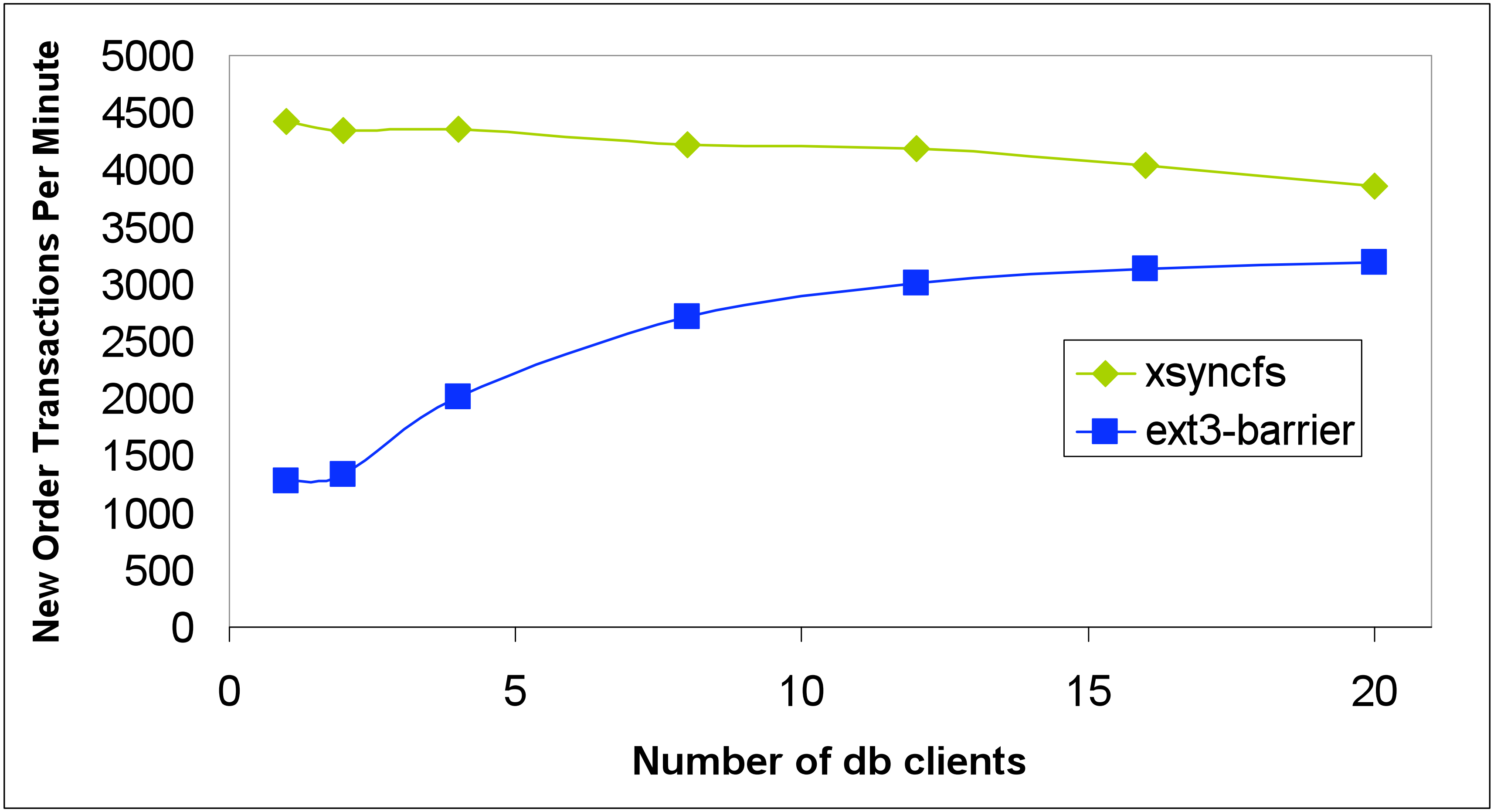
File System Configuration	Data durable on write()	Data durable on fsync()
Asynchronous	No	Not on power failure
Synchronous	Not on power failure	Not on power failure
Synchronous w/ write barriers	Yes	Yes
External synchrony	Yes	Yes

Postmark benchmark



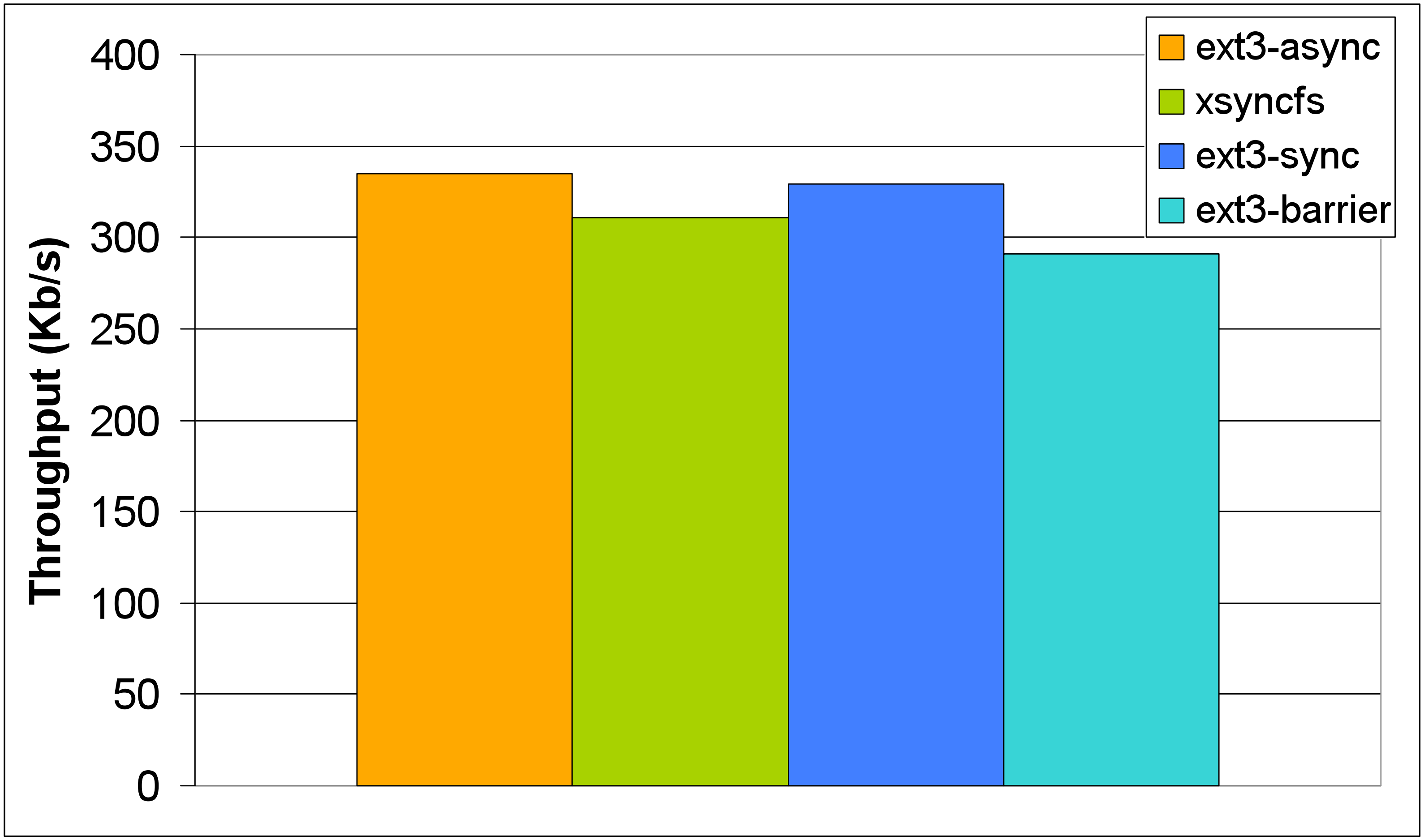
- Xsyncfs within 7% of ext3 mounted asynchronously

The MySQL benchmark



- Xsyncfs can group commit from a single client

Specweb99 throughput



- Xsyncfs within 8% of ext3 mounted asynchronously

Specweb99 latency

Request size	ext3-async	xsyncfs
0-1 KB	0.064 seconds	0.097 seconds
1-10 KB	0.150 second	0.180 seconds
10-100 KB	1.084 seconds	1.094 seconds
100-1000 KB	10.253 seconds	10.072 seconds

- Xsyncfs adds no more than 33 ms of delay

Conclusion

- Synchronous I/O can be fast
- External synchrony performs with 8% of async
- Questions?