Speculation

Supriya Vadlamani

CS 6410 Advanced Systems

Introduction : Authors

- Edmund B Nightingale
  - Microsoft Research(currently)
  - Speculator -thesis
- Jason Flinn
  - University of Michigan
- Peter M Chen
  - University of Michigan

Edmund B. Nightingale, Peter M. Chen and Jason Flinn

SPECULATIVE EXECUTION IN DISTRIBUTED FILE SYSTEM

Distributed File System : basics

- *allows access to files located on another remote host as though working on the actual host computer*

- Does it perform worse than local file system?
  - YES , Synchronous I/O
    - Cache Coherence :

      Sync n/w messages provide consistency
    - Data Safety

      Sync disk writes provide safety

Proposal

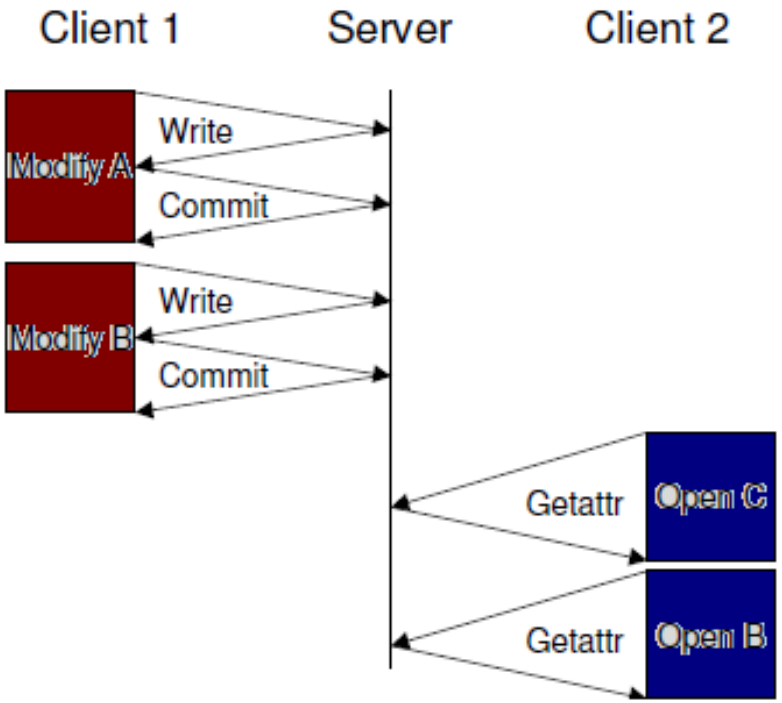
- Solutions :
  1. Sacrifice guarantees to gain speed
  2. Speculation with OS support
- Speculation
  - DFS can be safe, consistent and fast
    - Light weight checkpoints
    - Speculative execution
    - Tracking causal dependencies

Conditions for success

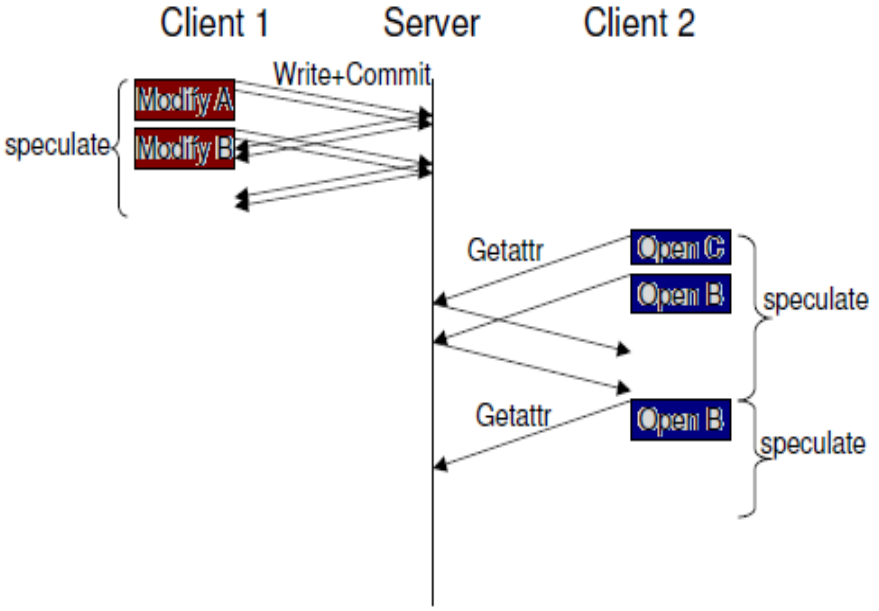
- Time for lightweight Check point < n/w round trip time
  - Around 52 microseconds
- Modern Computers can afford to execute speculatively
  - Spare resources: CPU cycles and memory
- File system clients can predict the outcome correctly.
  - Conflicts are rare
  - Very few concurrent updates

Speculation in NFS

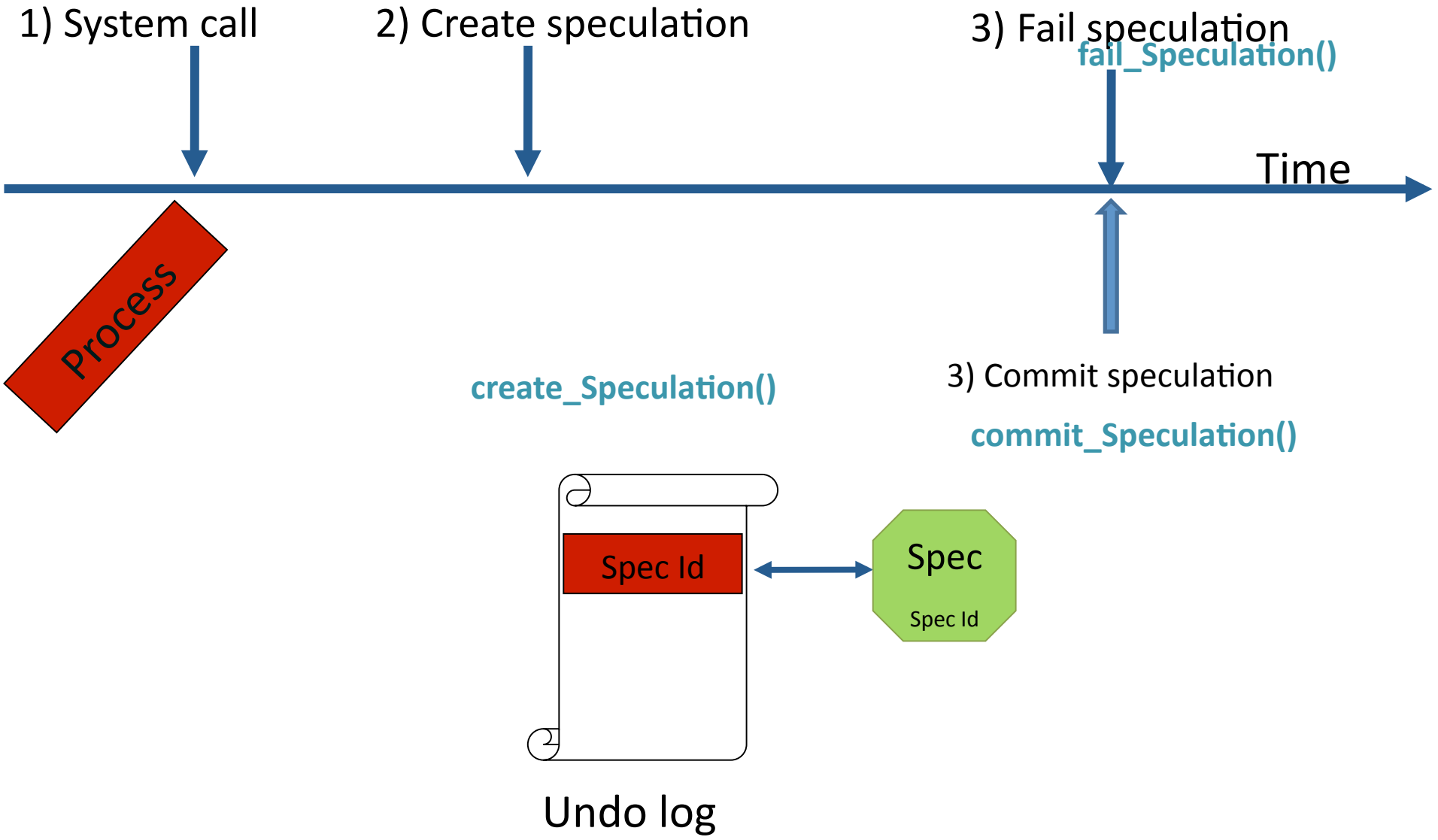
NFS without speculation



NFS with speculation



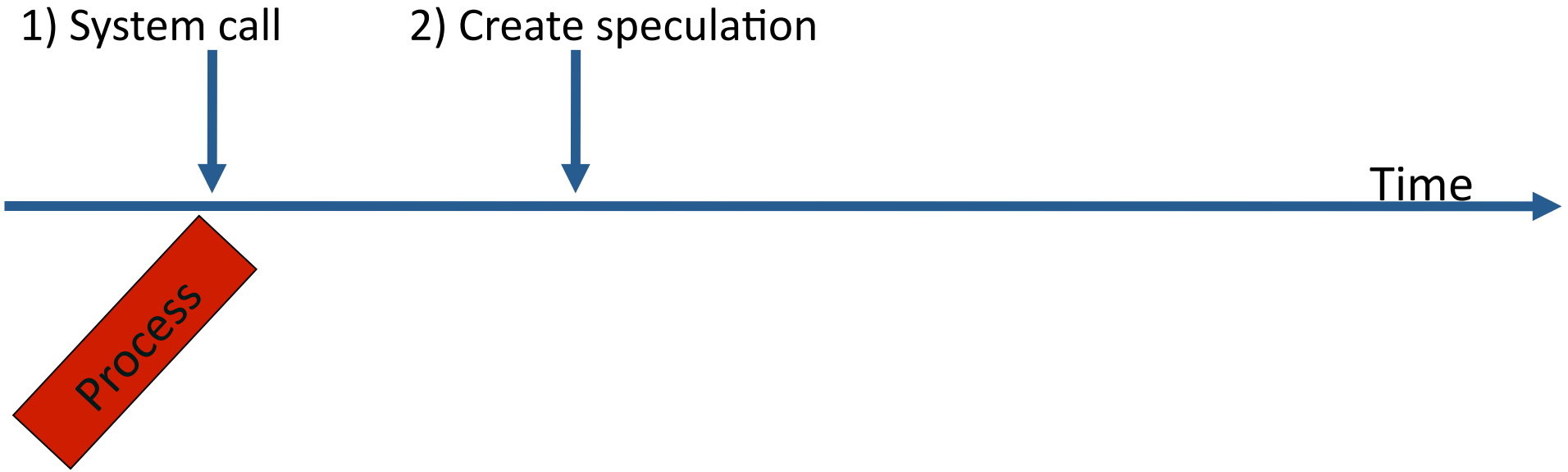
Speculation Interface

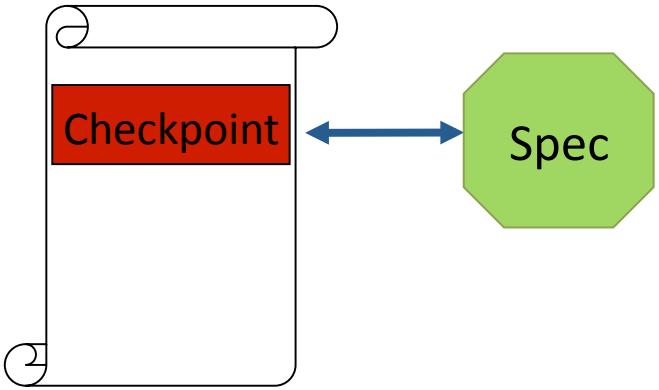


Advantages of the Interface

- Application independent
- Speculation success/failure cab be declared by any process
- Abstraction over the hypothesis underlying each speculation

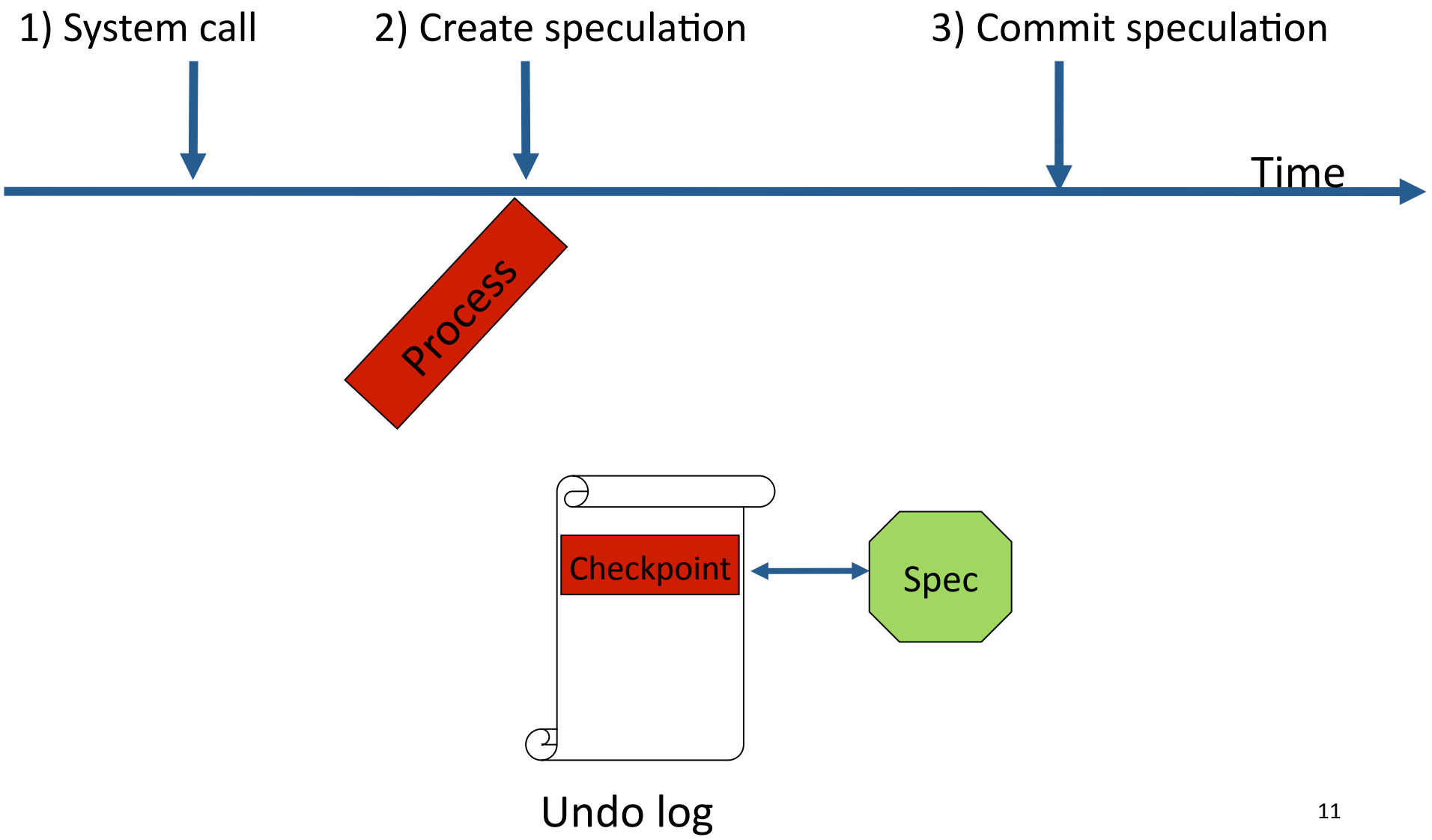
Implementing Speculation



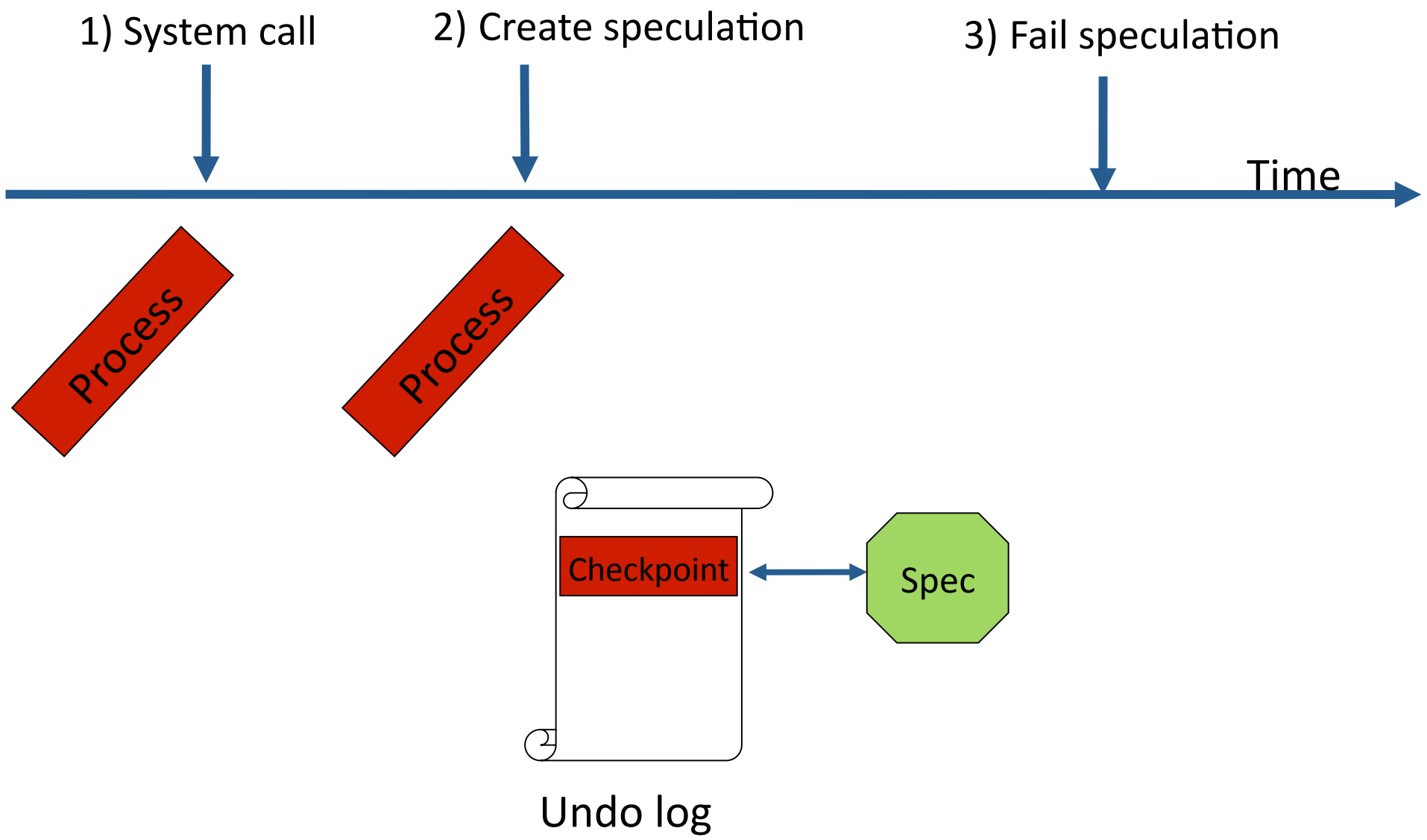


Undo log

Speculation Success



Speculation Failure



Implementing Speculation

- Data Structures
  - Speculation structure
    - Set of kernel objects
      - Depend on speculation
  - Undo log
    - Ordered list of speculative objects

Implementing Speculation

- Speculative process can
  - Call System calls that don't modify state (Getpid())
    - Can modify calling process' state : dup2
  - Can perform file system operations
    - If flag is set
- Correct Speculation Execution
  - Invisible to user or external device
  - Process shouldn't view speculative state unless it is speculatively dependent

Implementing Speculation

Ensuring Correctness

- Issues:
  - External state
    - Displaying a message on to the console
    - Sending a message over the network
- Solutions:
  - Propagate dependencies
  - Buffer
  - Block the process

Multi-process Speculation
[Speculative processes and IPC]

Processes often cooperate

Example: “make” forks children to compile, link, etc.

Would block if speculation limited to one task

Allow kernel objects to have speculative state

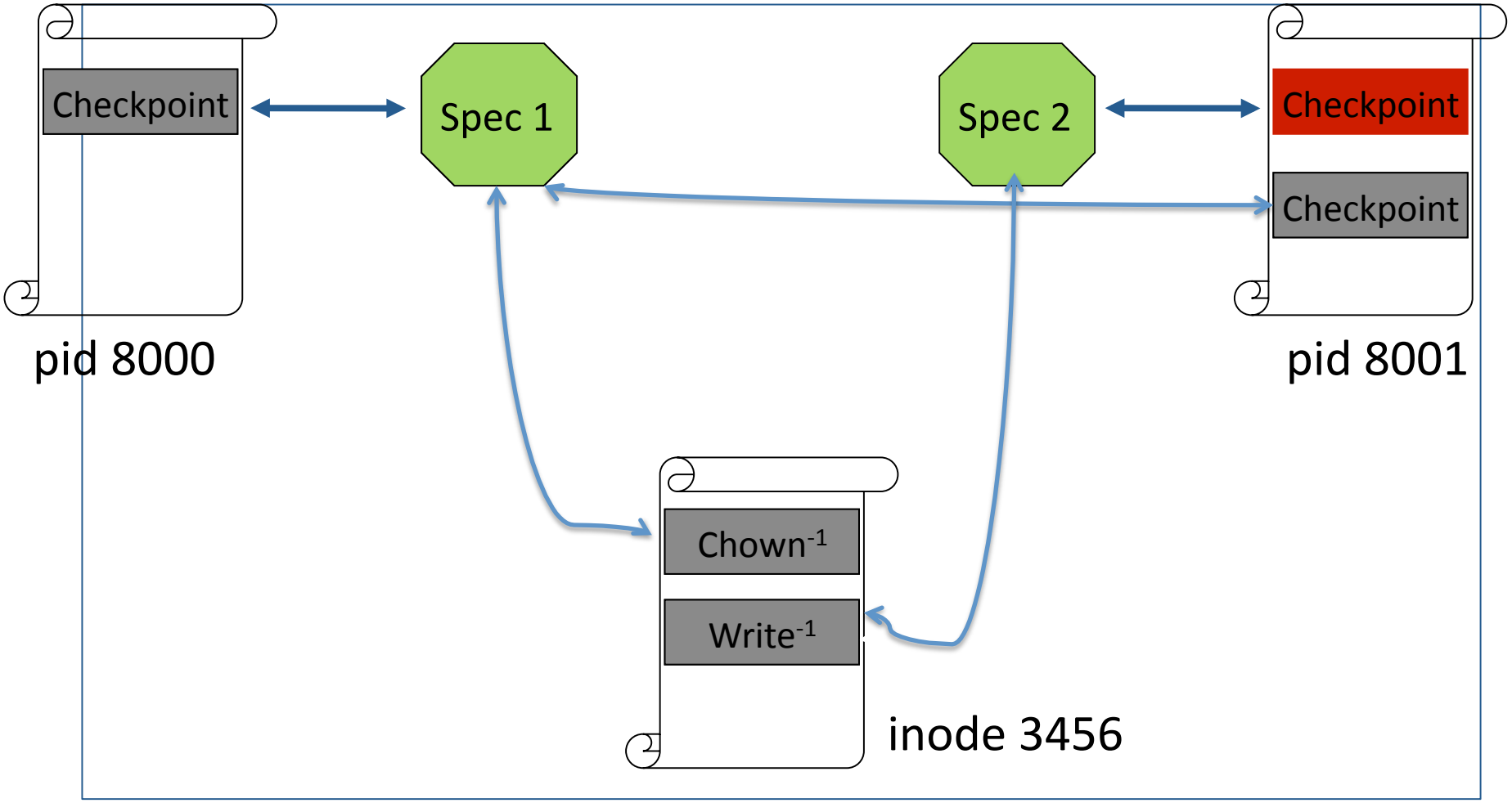
Examples: inodes, signals, pipes, Unix sockets, etc.

Propagate dependencies among objects

Objects rolled back to prior states when specs fail

Multi-process Speculation

[Speculative processes and IPC]



Multi-process Speculation
[Speculative processes and IPC]

Speculator supports:

- DFS objects
- RAMFS
- Local file system objects - Ext3
- Pipes & FIFOs
- Unix Sockets
- Signals
- Fork & Exit

Handling Mutating Operations

Client 1

1. cat foo > bar

Client 2

2. cat bar

bar depends on cat foo

What does client 2 view in bar?

Simple Solution:

restricted nature of communication in a server-based DFS

Handling Mutating Operations

- Server always knows the true state of the file system;
  - **Client** includes the hypothesis underlying that speculation.
  - **Server:** Evaluates the hypothesis underlying the speculation
    - If hypothesis is valid.
      - Mutation is performed

    Else
      - fails the mutation

Handling Mutating Operations

- Eg: **BlueFS client**：check version RPC [version number of its cached copy foo]

  Server: checks this version number against current version

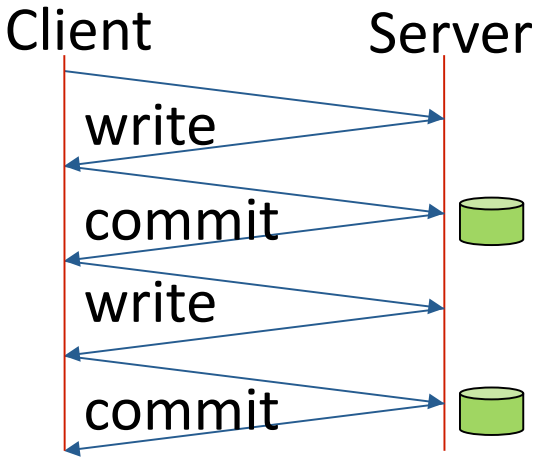
  fails the speculation if the two differ

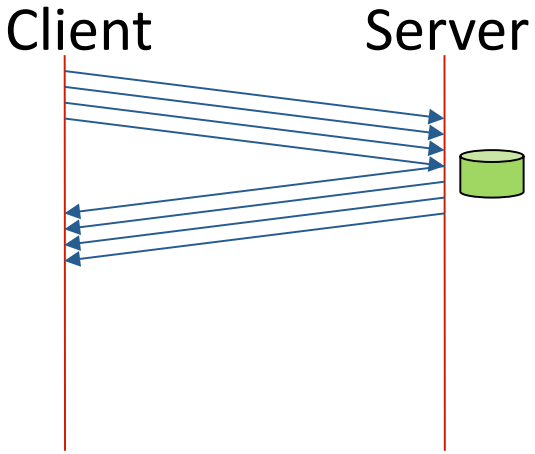
  Server: If previously failed any of the listed speculations, it fails the mutation.

- **Causal dependencies:**
  - set of speculations associated with the undo log of prior processes
  - List returned by create speculation and included in any speculative RPC sent to the server.

Speculative group commit

- Parallelize writes to disk by grouping them





SPEC NFS

- preserves existing NFS semantics, including close-to-open consistency.
- Issues the same RPCs, many of these RPCs are speculative

NFS: Security is still an issue!

Blue FS

Features:

- Single copy semantics
- Synchronous I/O

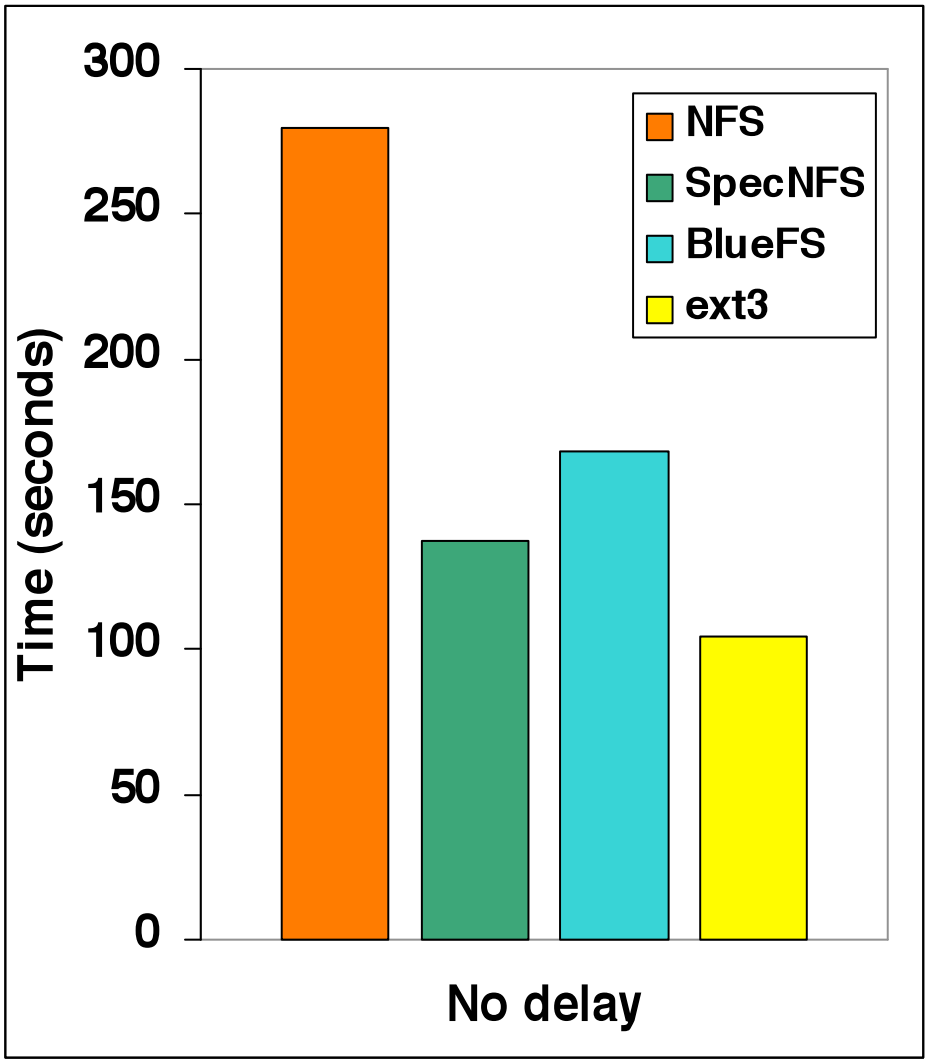
Each file, directory, etc. has version number

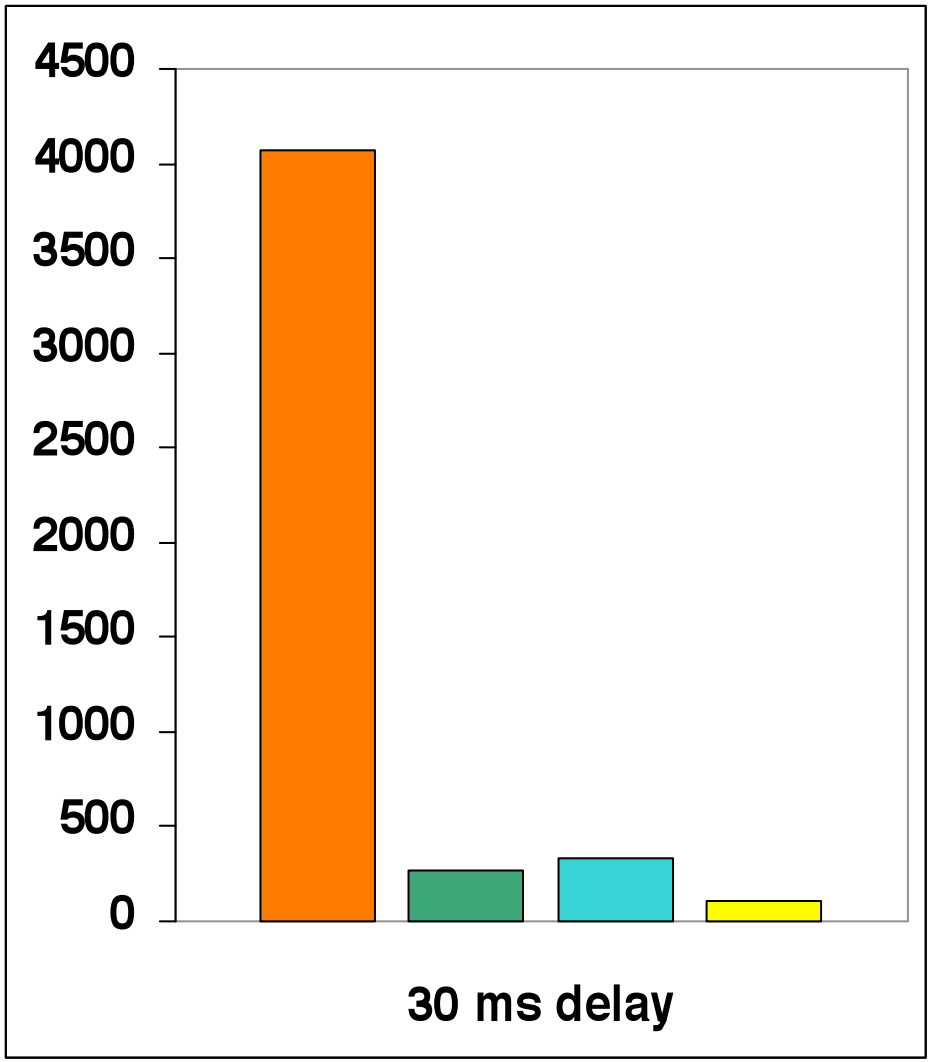
- Incremented on each mutating op (e.g. on write)
- Checked prior to all operations.
- Many ops speculate and check version async

Evaluation

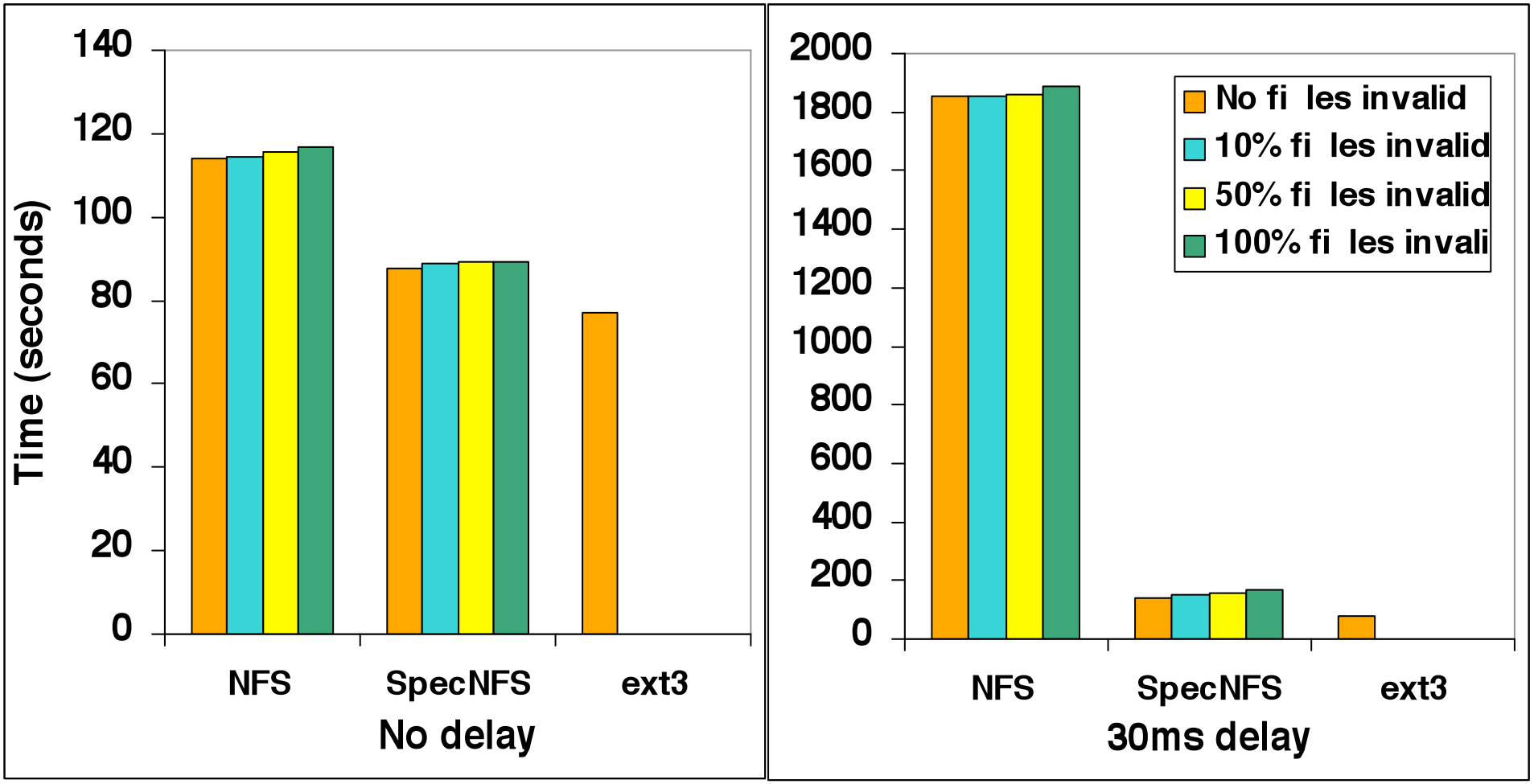
- Two Dell Precision 370 desktops as the client and file server
- Each machine has a 3 GHz Pentium 4 processor, 2GB DRAM, and a 160GB disk.
- To insert delays, we route packets through a Dell Optiplex GX270 desktop running the NISTnet [4] network emulator.
- Ping time between client and server is 229 s.

Results : Apache Benchmark

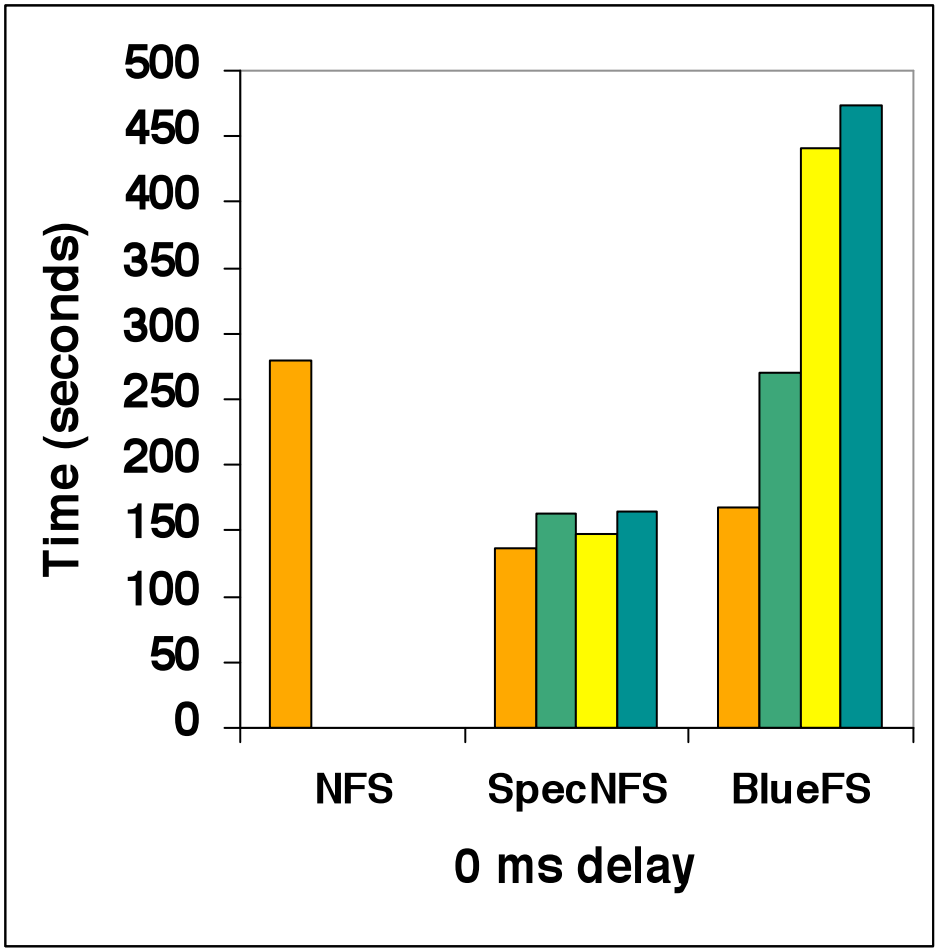


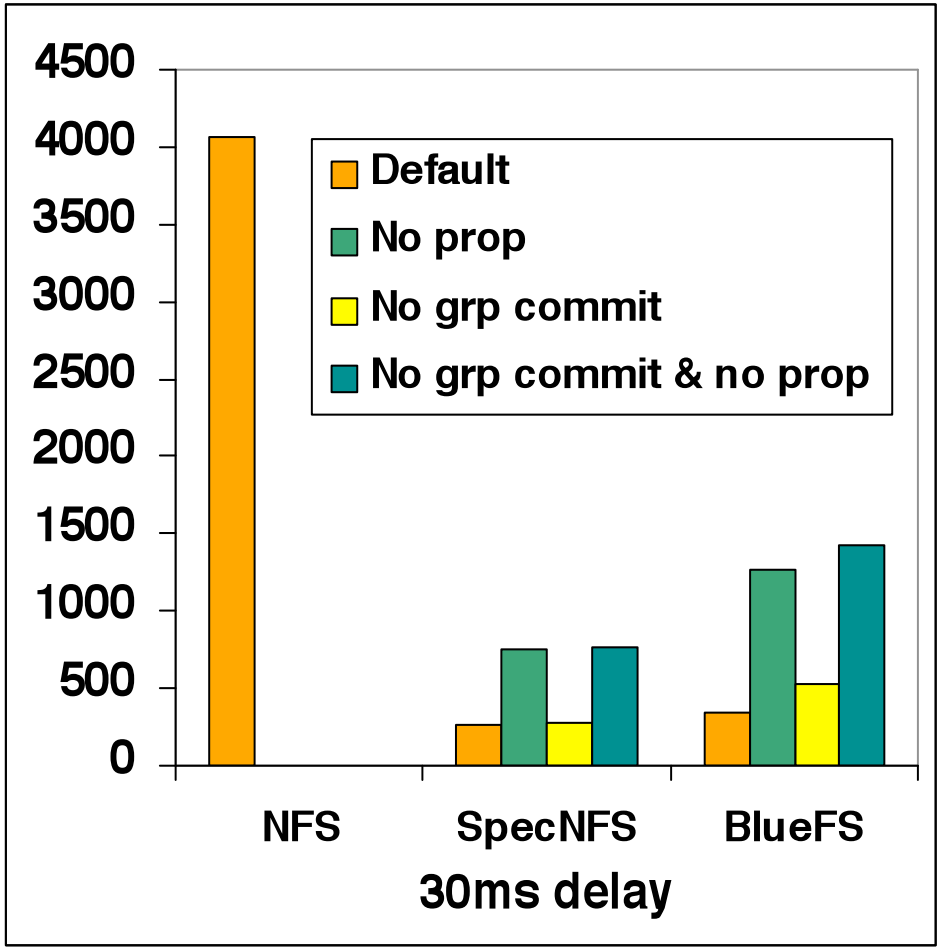


Cost of Rollback



Group Commit & Sharing State





Discussion

- Speculation: not a new concept
  - Used for hardware
  - Is speculation in OS a good idea?
- Server handles the speculation
  - Server crashes ?

Edmund B. Nightingale, Kaushik Veeraraghavan ,Peter M. Chen and Jason Flinn

RETHINK THE SYNC

Synchronous I/O v/s Asynchronous I/O

- Very slow
- Applications a frequently blocked (decreases performance by 2 orders of magnitude)

- Fast, not safe
- Does not block an application
- Complicates applns tht require durability and reliability

Despite, the Asynchronous I/O ‘s poor guarantees, users prefer asynchronous I/O because synchronous I/O is too slow!

External Synchrony

- External Synchrony
  - Provides the reliability & simplicity of synchronous system
  - Closely approaches the performance of asynchronous system

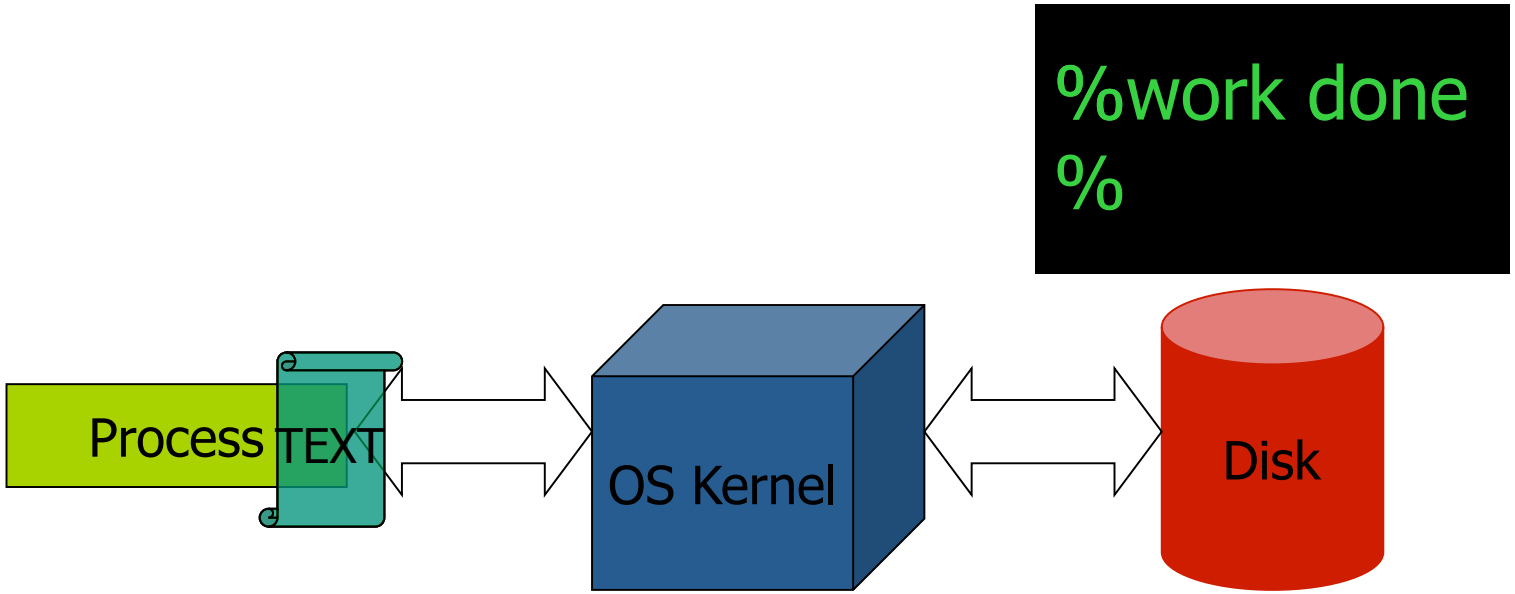
- Synchronous I/O: Application centric view
- External Synchrony: User centric view

Example: Synchronous I/O

```
101   write(buf_1);
102   write(buf_2);
103   print("work done");
104   foo();
```

Application blocks

Application blocks



Observing synchronous I/O

```
101   write(buf_1);
102   write(buf_2);
103   print("work done");
104   foo();
```

Depends on 1st write

Depends on 1st & 2nd write

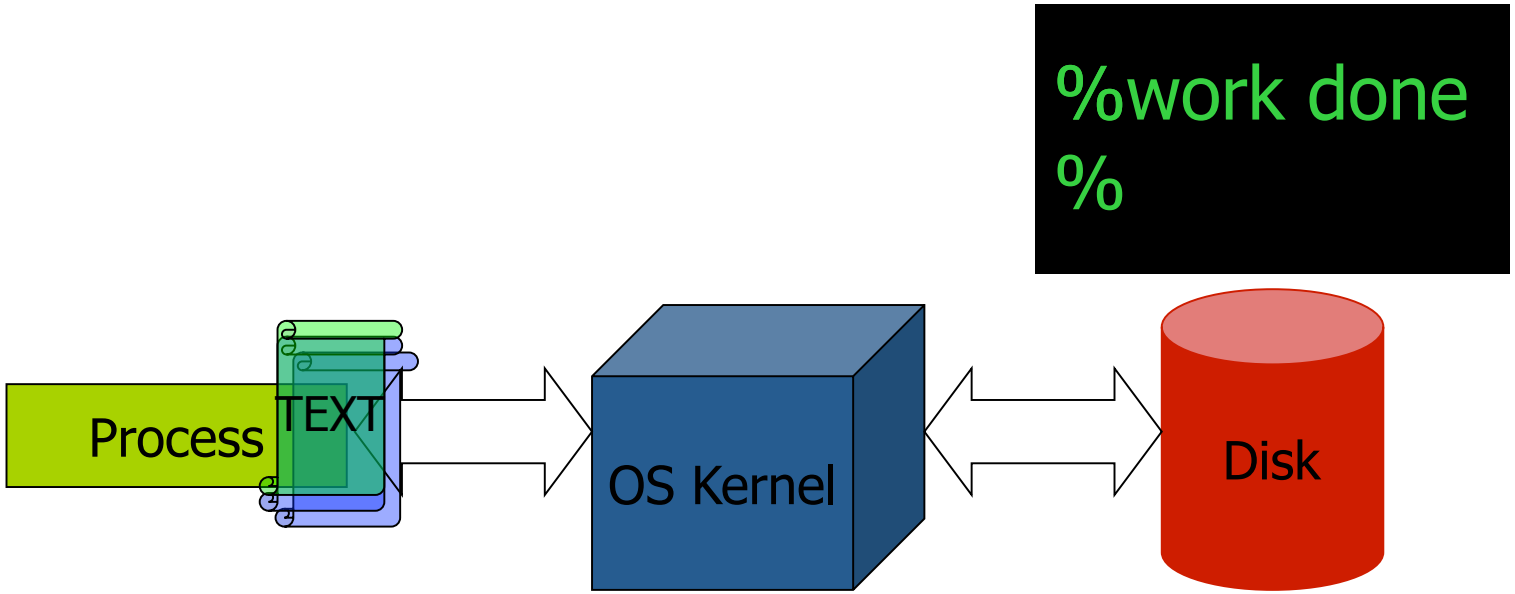
Sync I/O externalizes output based on causal ordering

Enforces causal ordering by blocking an application

Ext sync: Same causal ordering without blocking applications

Example: External synchrony

```
101   write(buf_1);
102   write(buf_2);
103   print("work done");
104   foo();
```



Optimizations to External Synchrony

- Two modifications are grouped and committed as a single file system transaction

- Buffer screen output

Causal dependencies need to be resolved to between file system modifications and external output

Limitations of External Synchrony

- Externally synchronous system can propagate failures using a speculator to checkpoint a process before modifications
- User may have temporal expectations about modifications committed to a disk
- Modifications to data in two different file system cannot be committed in a single transaction

Evaluation

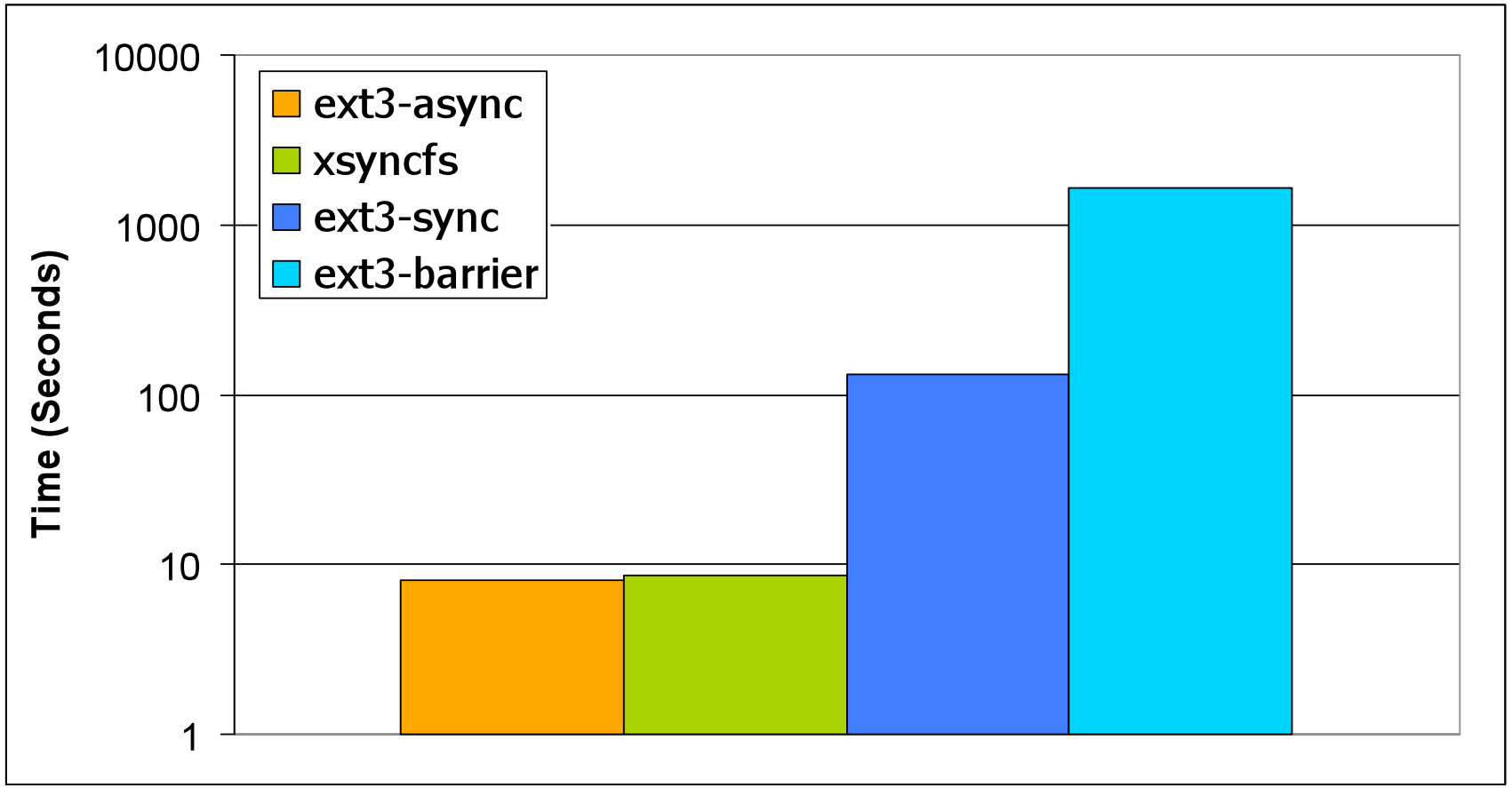
Implemented ext sync file system Xsyncfs

- Based on the ext3 file system
- Use journaling to preserve order of writes
- Use write barriers to flush volatile cache

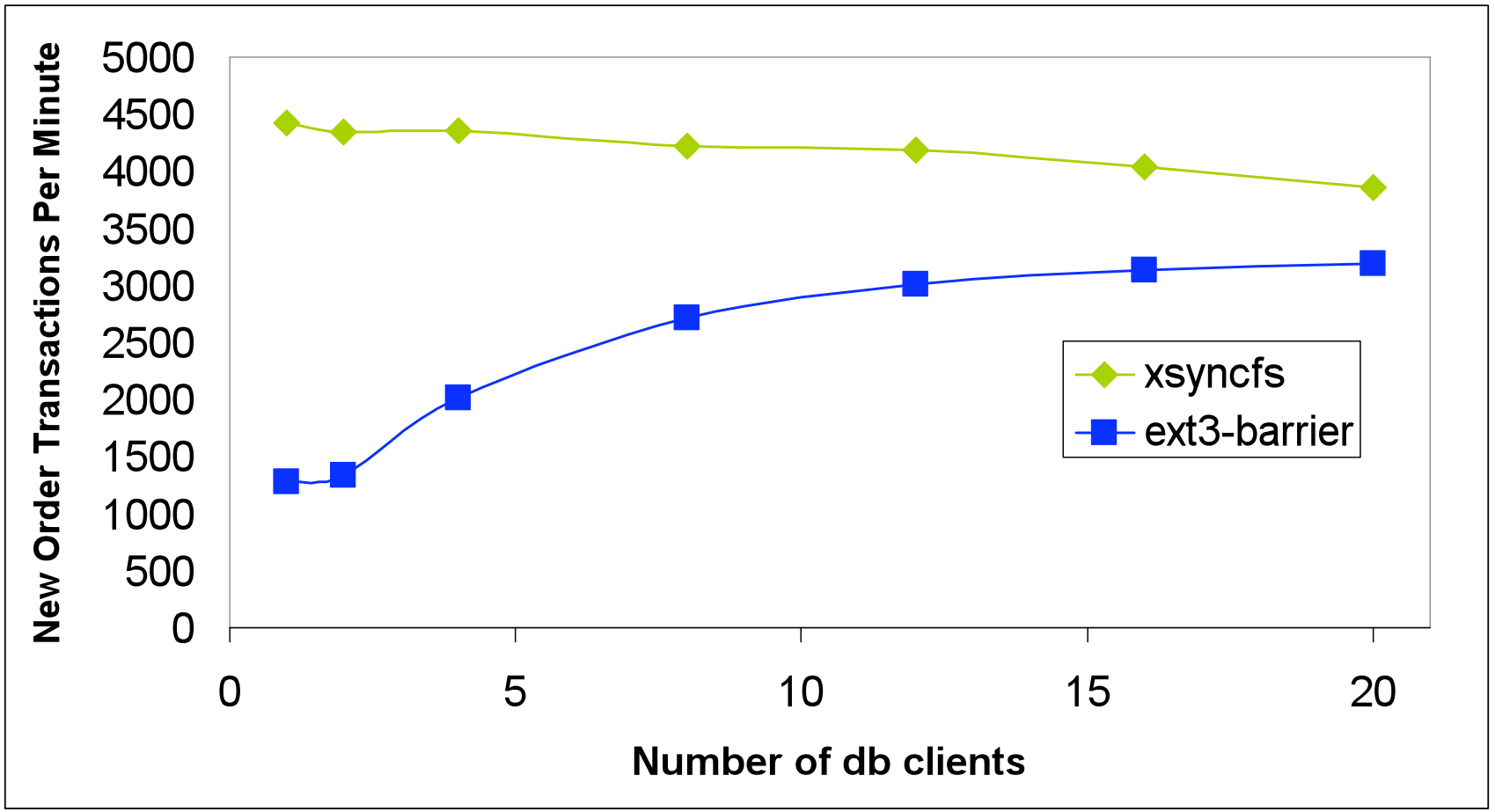
Compare Xsyncfs to 3 other file systems

- Default asynchronous ext3
- Default synchronous ext3
- Synchronous ext3 with write barriers

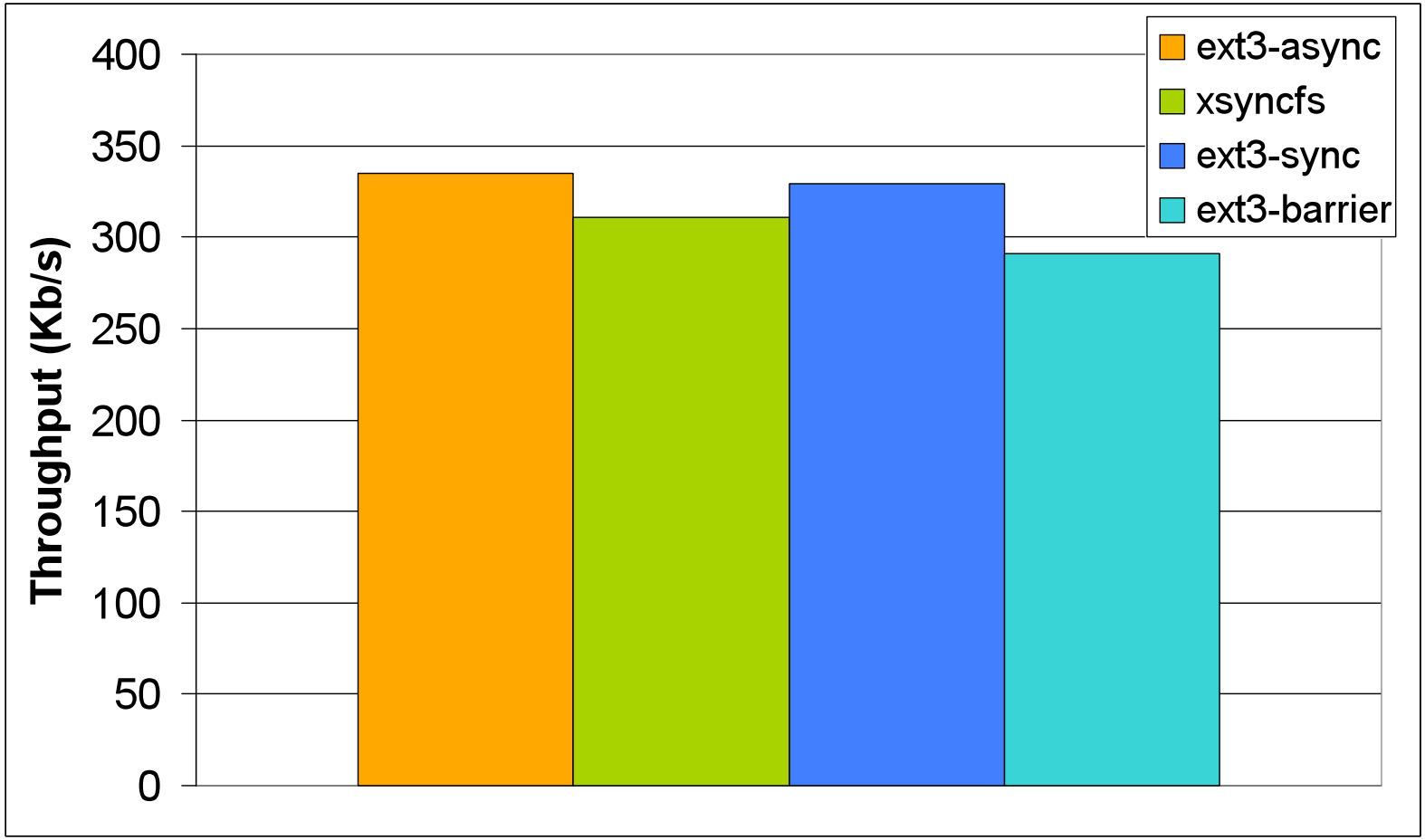
Postmark Benchmark



MYSQL Benchmark



SPEC web99 Throughput



Discussion

- Xsyncfs:
  - Performs several orders better than existing solns.
  - Why isn't it widely used?
- Checkpoints:
  - How far can you go?

Conclusion

- Speculation in DFS
  - Spec NFS
    - Security is still an issue
    - Same authors examined security issues wrt speculation in later works
  - Blue FS
    - Consistent and single copy semantics

- Rethink the sync : Xsyncfs
    - a new model for local file I/O that provides the reliability and simplicity of synchronous I/O
    - approximates the performance of asynchronous I/O
    - But limitations of external synchrony exist

References

- Edmund B. Nightingale, Peter M. Chen, Jason Flinn, "Speculative Execution in a Distributed File System", *Proceedings of the 2005 Symposium on Operating Systems Principles (SOSP)*, October 2005
- Edmund B. Nightingale, Kaushik Veeraraghavan, Peter M. Chen, Jason Flinn, "Rethink the sync", *Proceedings of the 2006 Symposium on Operating Systems Design and Implementation (OSDI)*, November 2006
- Edmund B. Nightingale, Daniel Peek, Peter M. Chen, Jason Flinn, "Parallelizing security checks on commodity hardware", *Proceedings of the 2008 International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, March 2008
- Wikipedia