Evolving RPC for Active Storage

Muthian Sivathanu
Andrea C. Arpaci-Dusseau
Remzi H. Arpaci-Dusseau

University of Wisconsin-Madison

Architecture of the future

- Everything is “active”
  - Cheaper, faster processing power
  - Example: “smart” disks
- Everything is “distributed”
  - Network between processors and devices
  - Example: Network-attached storage
- Need: Effective software paradigms
  - Leverage power of active components
  - But remains easy to use

Software systems of the future

- Need: Tools to build “active”, “distributed” systems
  - Pragmatic: Easy for system developers to use
  - Powerful: Exploit active nature of systems
- Active systems permit extensibility
  - Download code to device
  - Tailor to needs of applications/system
  - Simplicity, maintainability
    - Provide primitives, allow clients to compose interface
- Traditional “distributed” systems built w/ RPC
  - Simple, easy-to-use communication paradigm
  - But not designed for “active” world
- Build better distributed systems w/ “active” components

Scriptable RPC

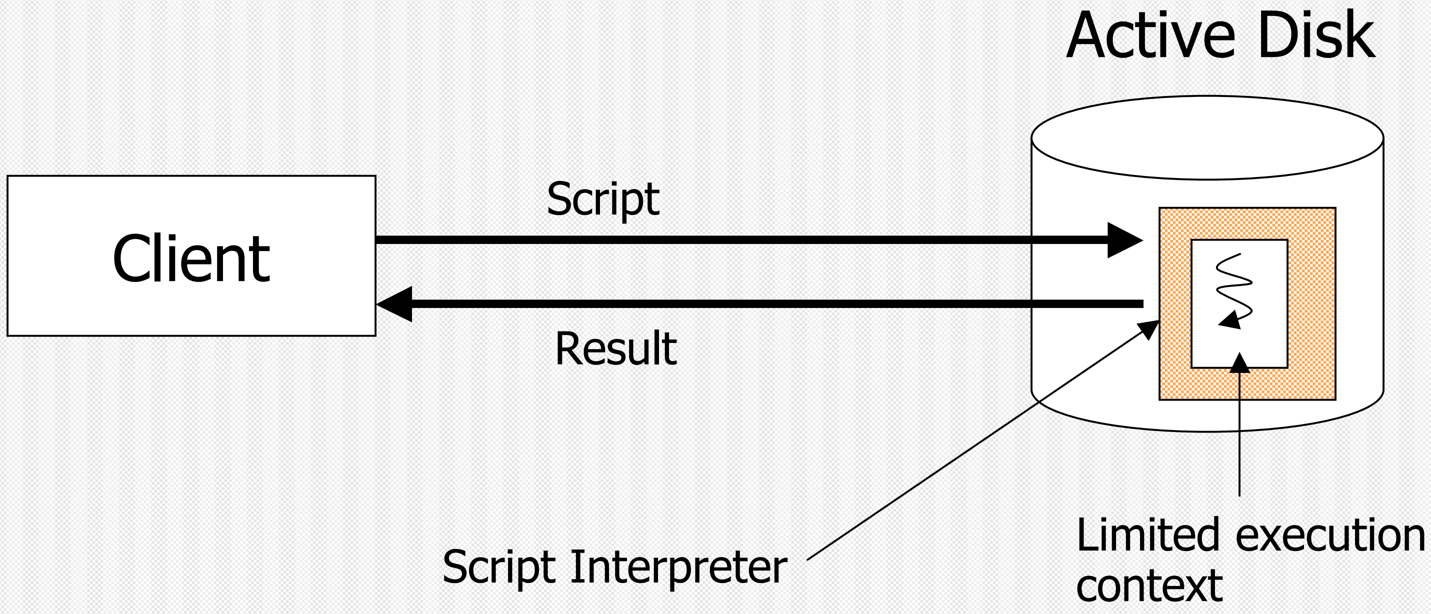
- SRPC: Paradigm for extensible distributed systems
  - Pragmatic: RPC-like development process
  - Powerful: Exploit active components easily
- Case study: Active storage
  - High Performance
    - Efficient “composition” of primitives
  - Rapid addition of new functionality
    - Powerful**:** Advanced consistency semantics over NFS
    - Simple**:** Substantial functionality in < 20 lines of code
  - Simplicity in design
    - Obviate distributed locking, crash recovery
- Compelling paradigm for future systems

Outline

- *Motivation*
- Scriptable RPC
- Case Study: Active Storage
  - Performance
  - Functionality
  - Simplicity
- Summary

Scriptable RPC (SRPC)

- Evolve Remote Procedure Call (RPC)
- Augment RPC interface of “server” with a scripting capability



- Prototype uses Tcl as the scripting language

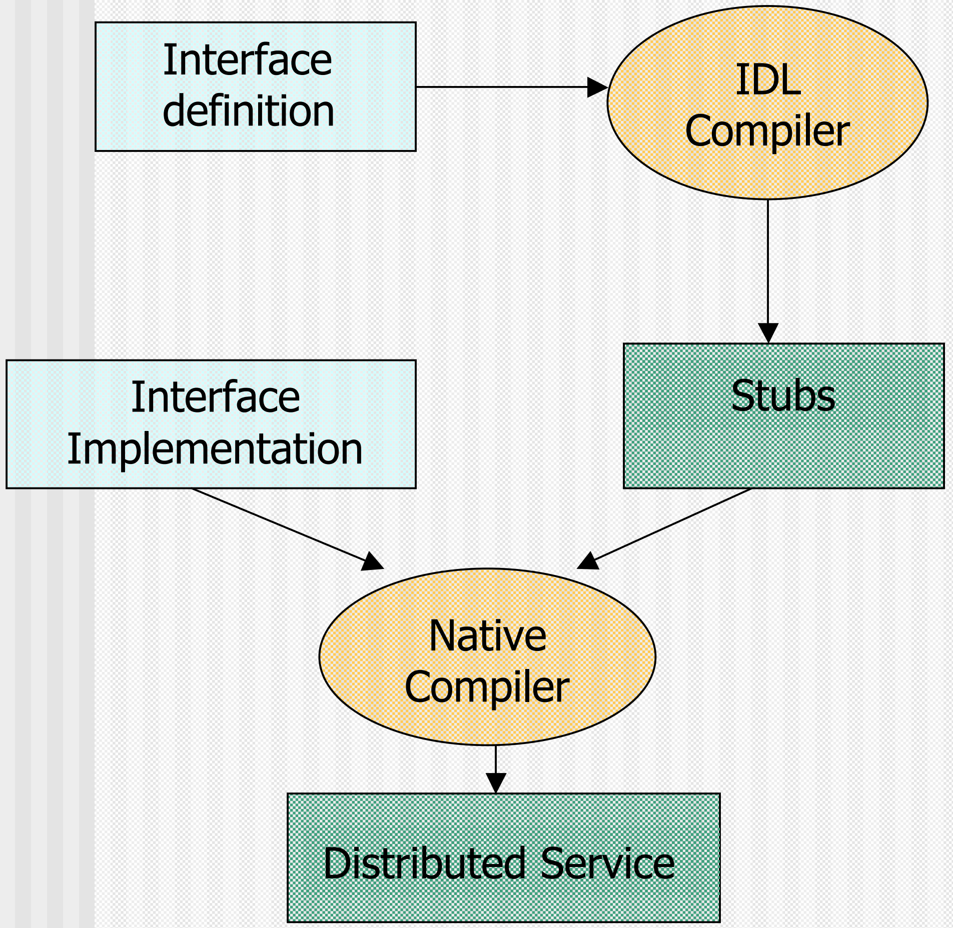
SRPC : Key issues

- Migration path
- Efficient execution of scripts
- Safety

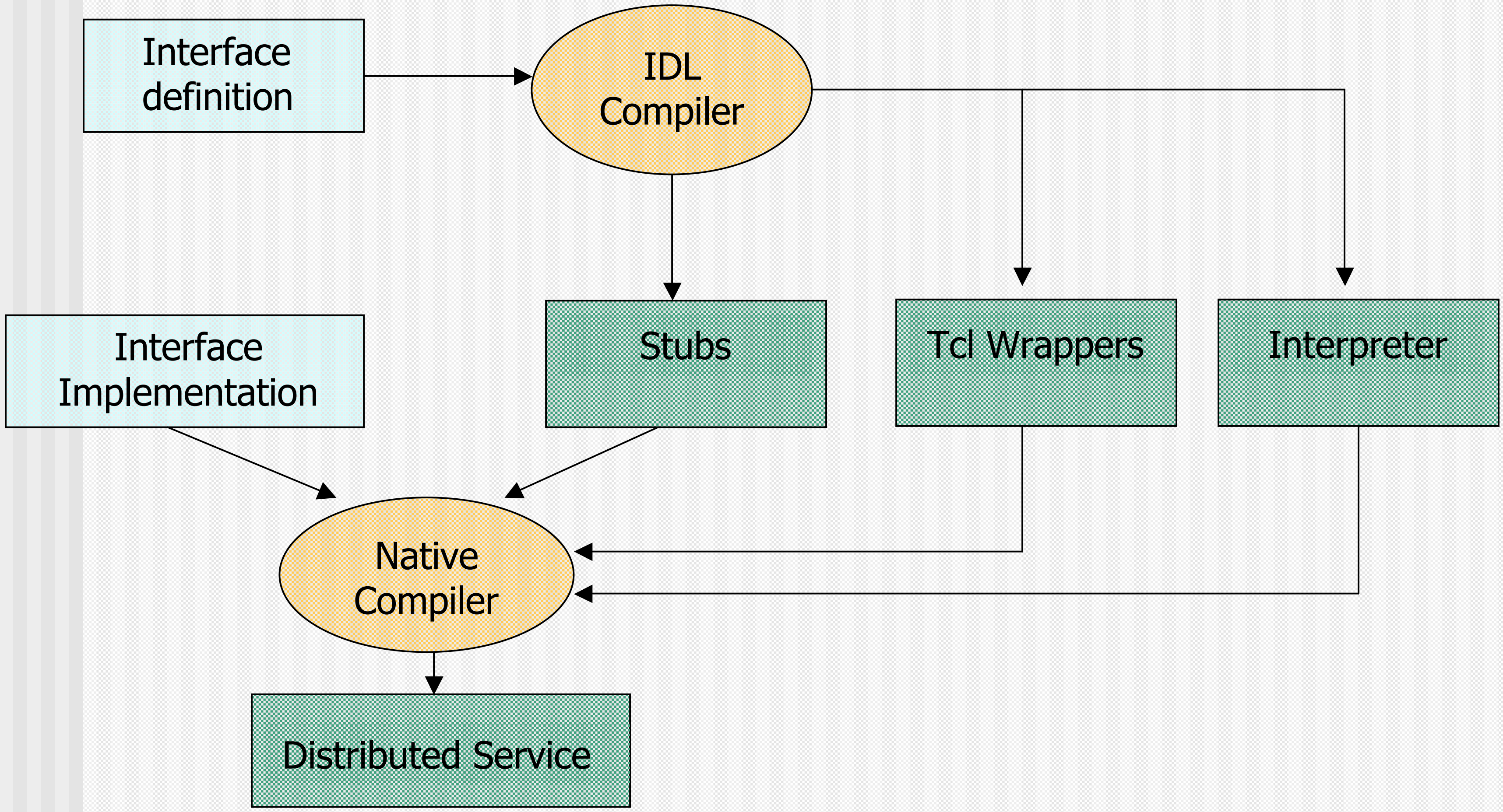
Migration path

- Make transition to new paradigm less intrusive
  - Code to embed scripting into server automatically generated
  - Existing unmodified clients co-exist with *scripting* clients
  - Development process exactly the same

Development: RPC



Development: SRPC



Efficient execution of scripts

- Hide script interpretation overhead
- Script caching
  - Exploit efficient Tcl bytecode representation
- Concurrency
  - Multiple interpreters run simultaneously
- “Fast” standard library of primitives
  - Implemented in C

Safety

- Guard against misbehaving client scripts
- Limited execution environment: SafeTcl
  - Even while loops can be turned off
- Runtime type-checking
  - Prevent illegal memory references
- Automatic tracking of locks
  - Safe concurrent execution

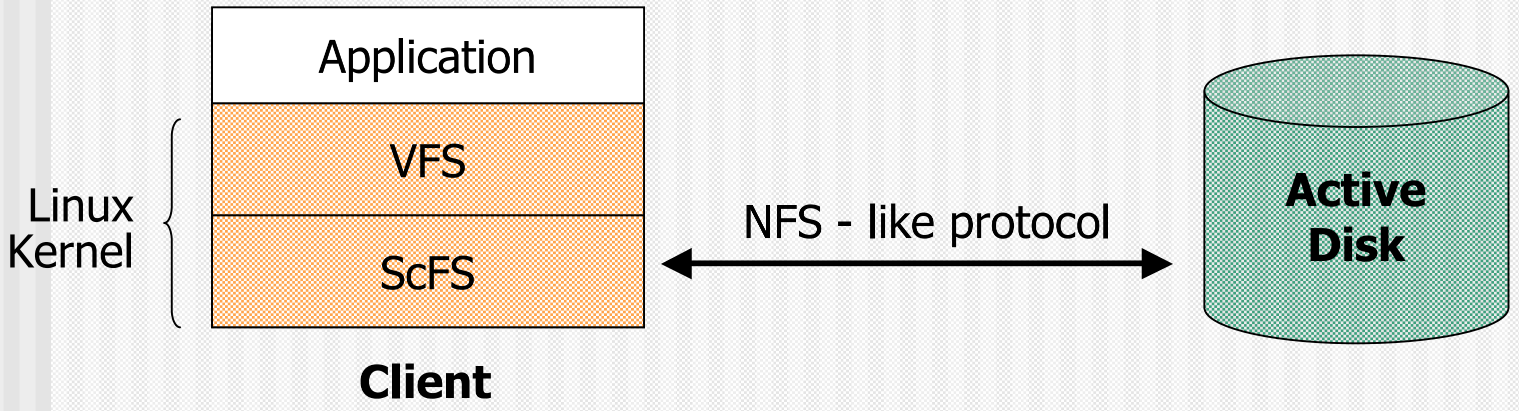
Outline

- *Motivation*
- *Scriptable RPC*
- Case Study: Active Storage
  - Performance
  - Functionality
  - Simplicity
- Summary

Case Study: Active storage

- Utilize CPU power at disks for client-specific processing
- Previous approaches
  - Demonstrate performance benefits
  - But, require radically new architectures
    - No migration path for existing services
  - Limited class of applications
    - Parallel database primitives

Evaluation environment



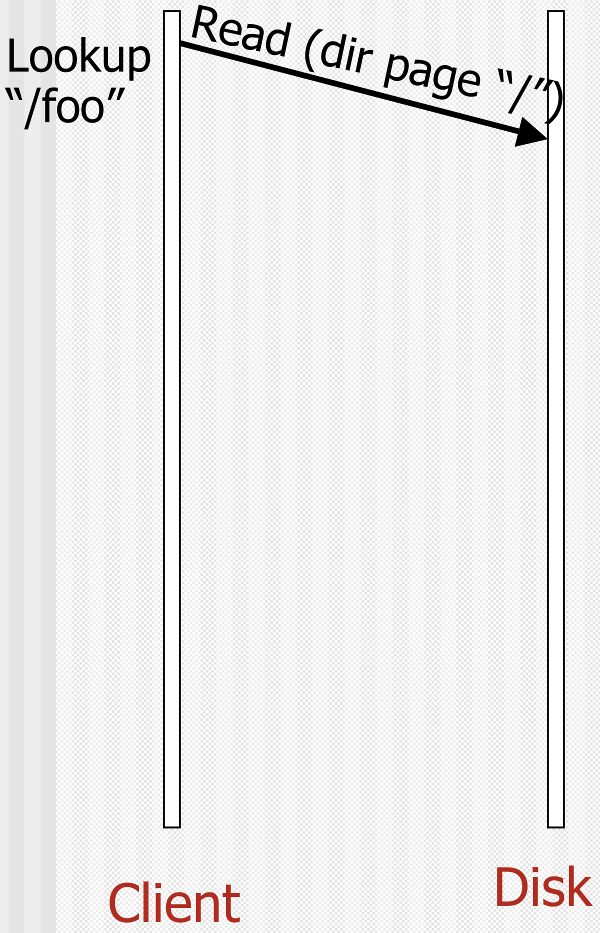
- Platform
  - P-III 550 MHz machines, 1GB mem, 100 Mb/s net
  - Linux kernel v2.2.19
- Case studies enhance ScFS using SRPC

ScFS: Performance enhancements

- Combine dependent sequence of operations into single script
  - Reduction in network round-trips needed for a logical operation
    - Benefit sensitive to network delay
    - Significant savings over dialup, wide-area
    - Even across overloaded “fast” networks
  - Reduction in total network traffic
  - Helps overcome limitations in interface

Pathname lookup

RPC



Lookup
"/foo"

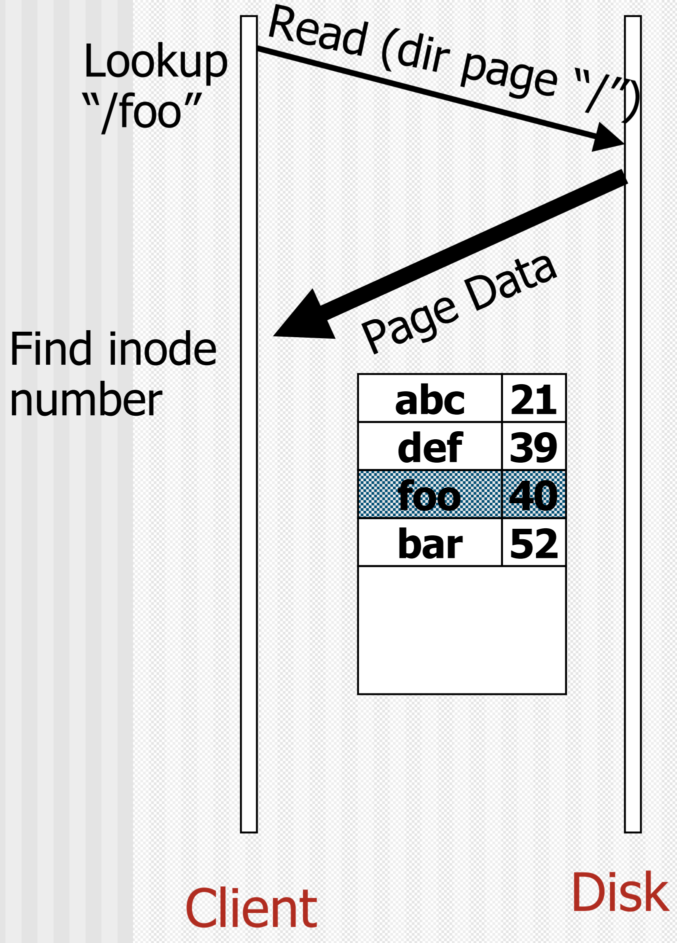
Read (dir page "/")

Client

Disk

Pathname lookup

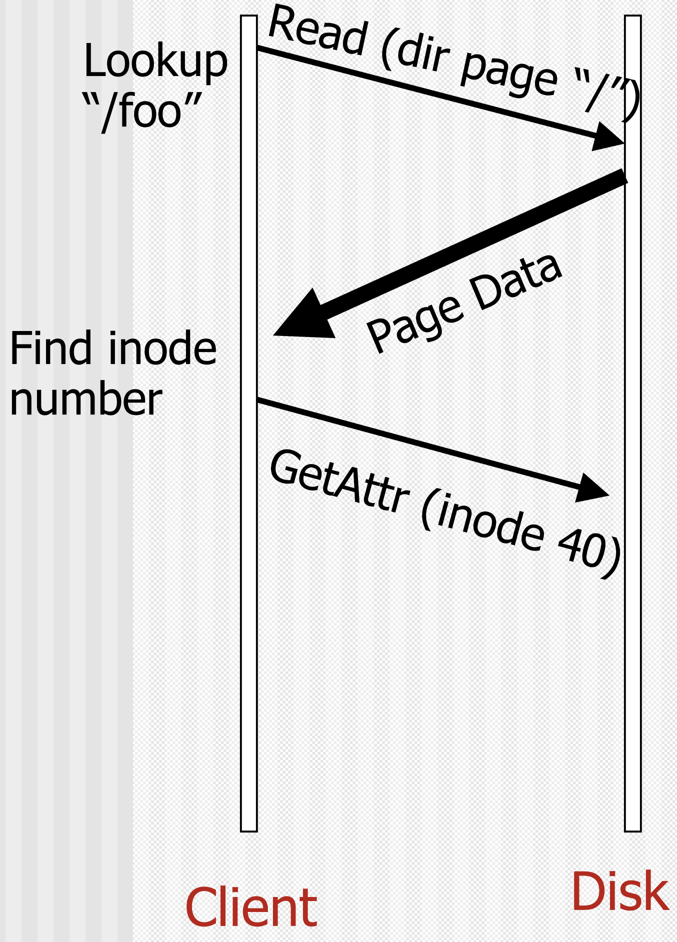
RPC



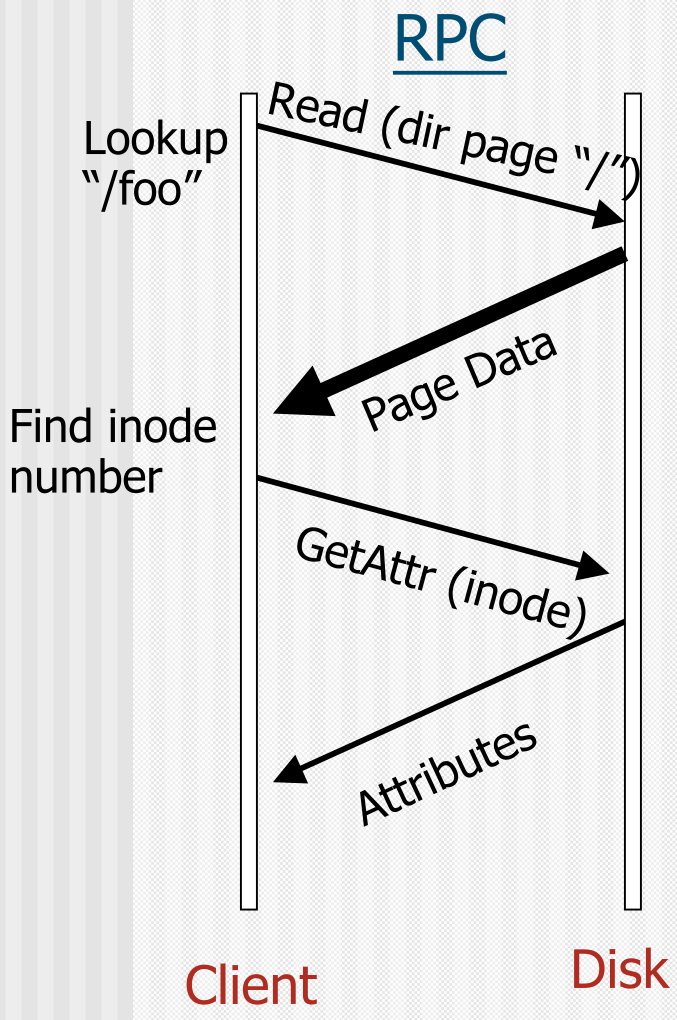
abc	21
def	39
foo	40
bar	52

Pathname lookup

RPC

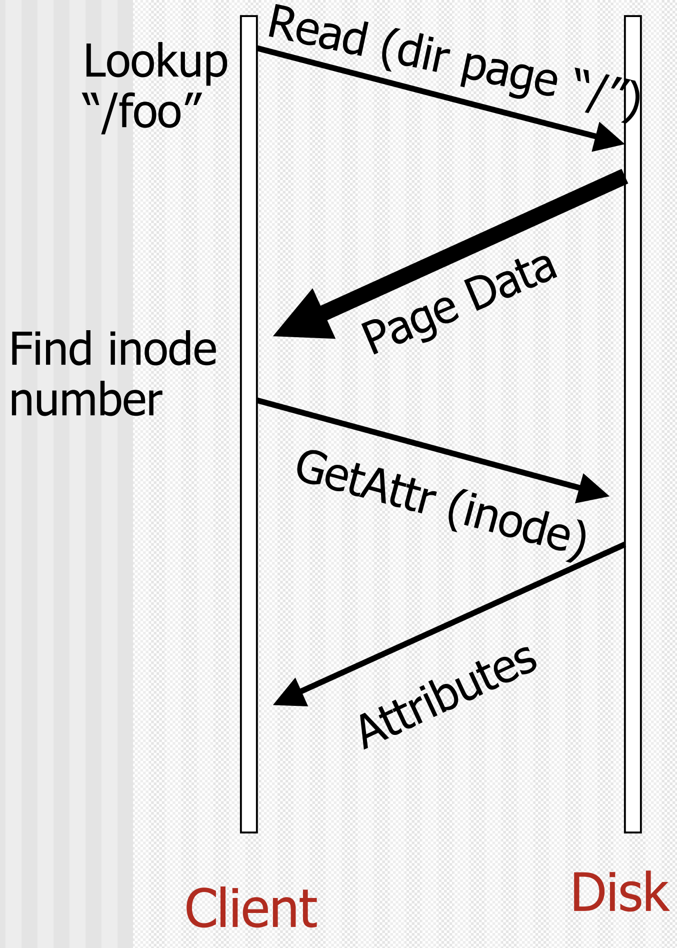


Pathname lookup

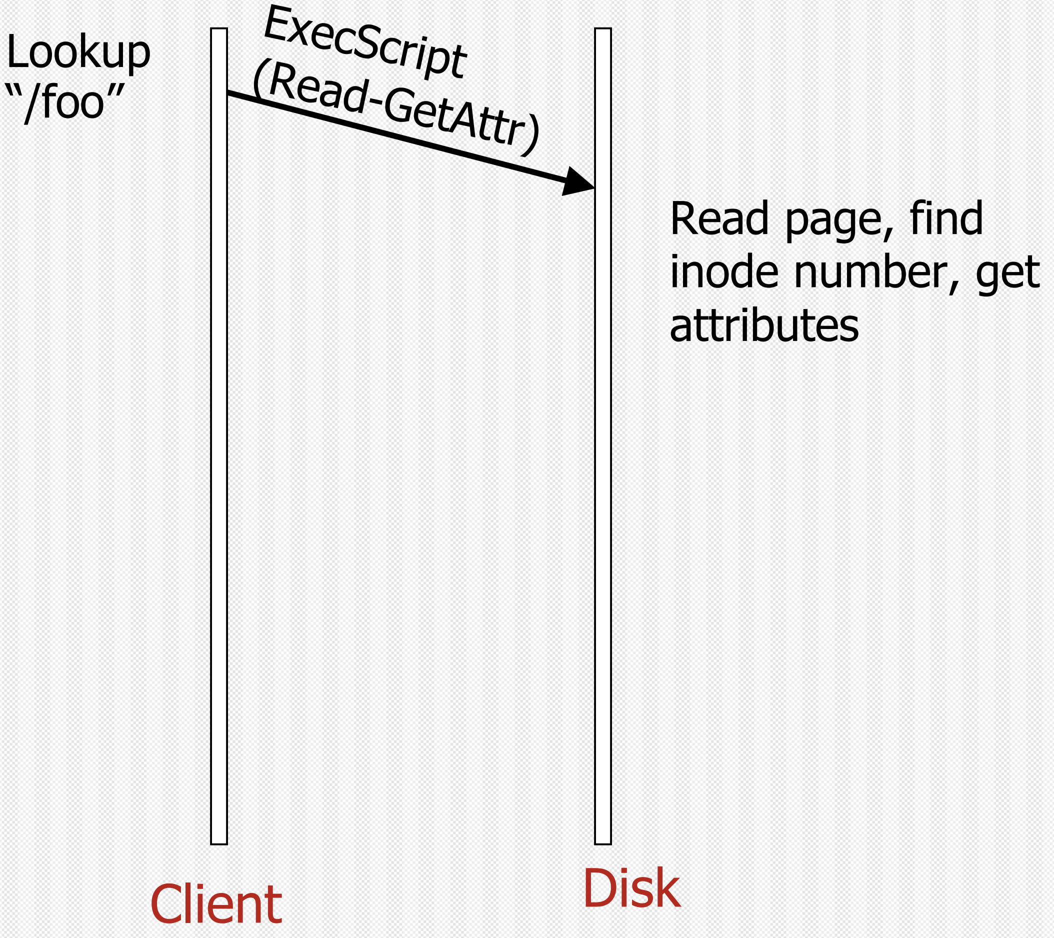


Pathname lookup

RPC

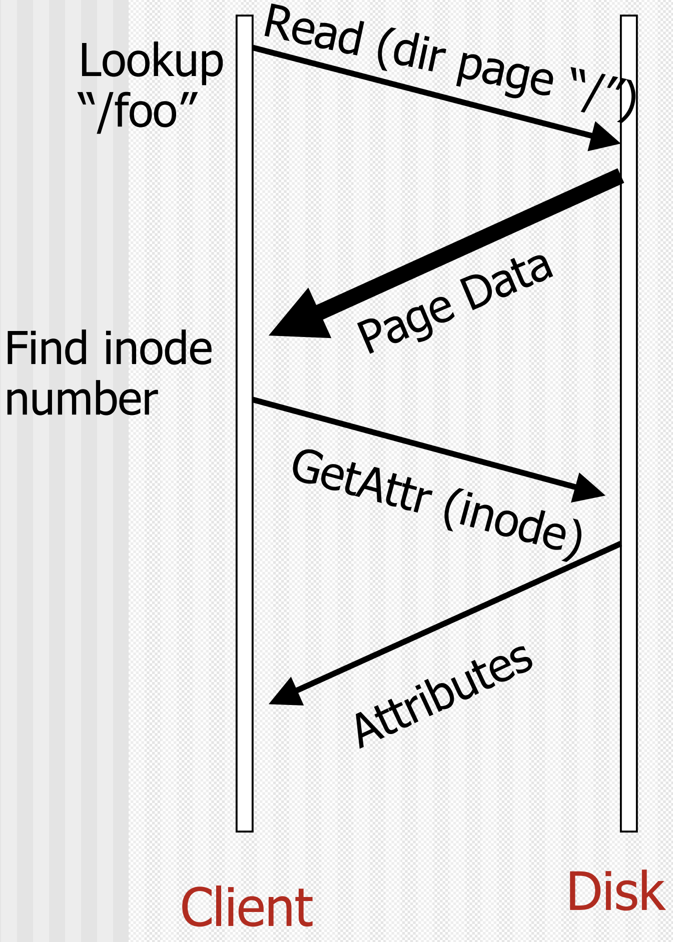


SRPC

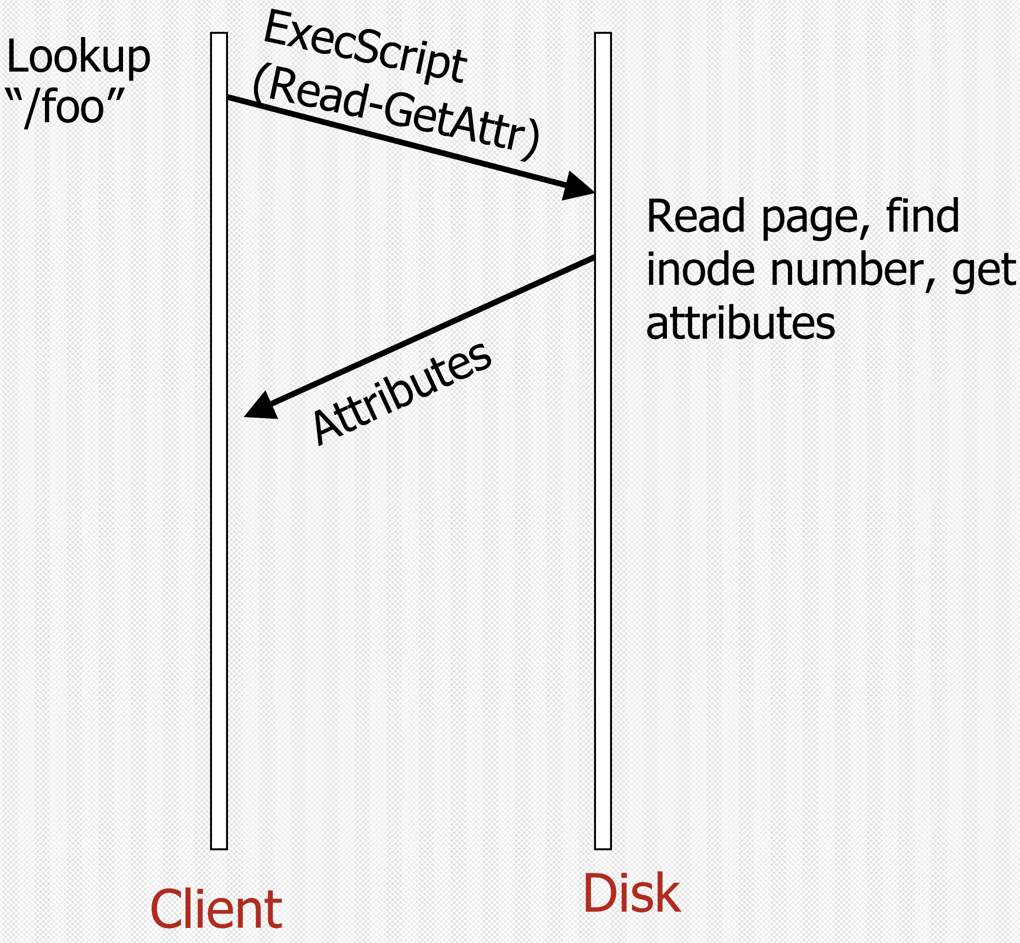


Pathname lookup

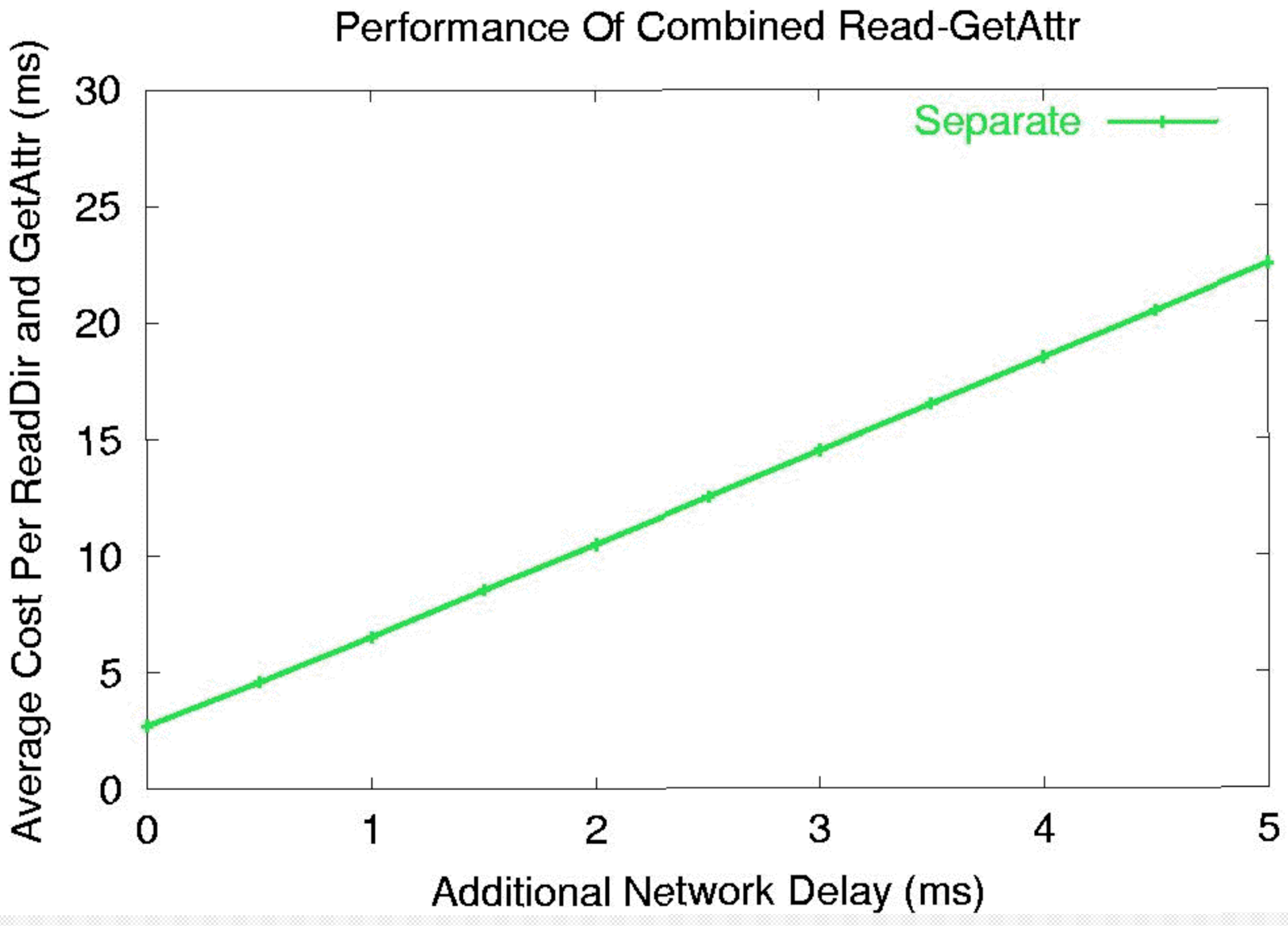
RPC



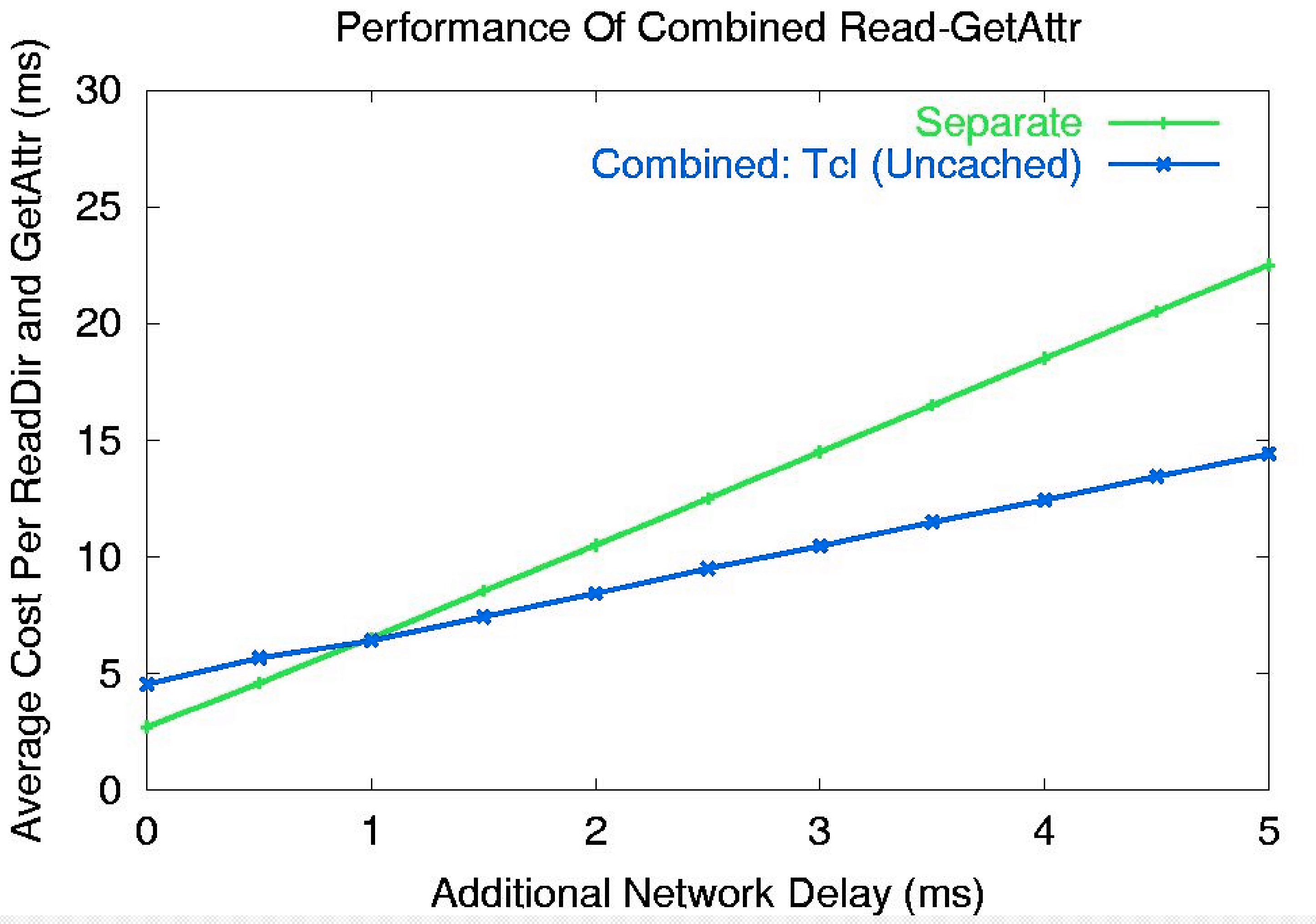
SRPC


Lookup "/foo"
ExecScript (Read-GetAttr)
Read page, find inode number, get attributes
Attributes
Client
Disk

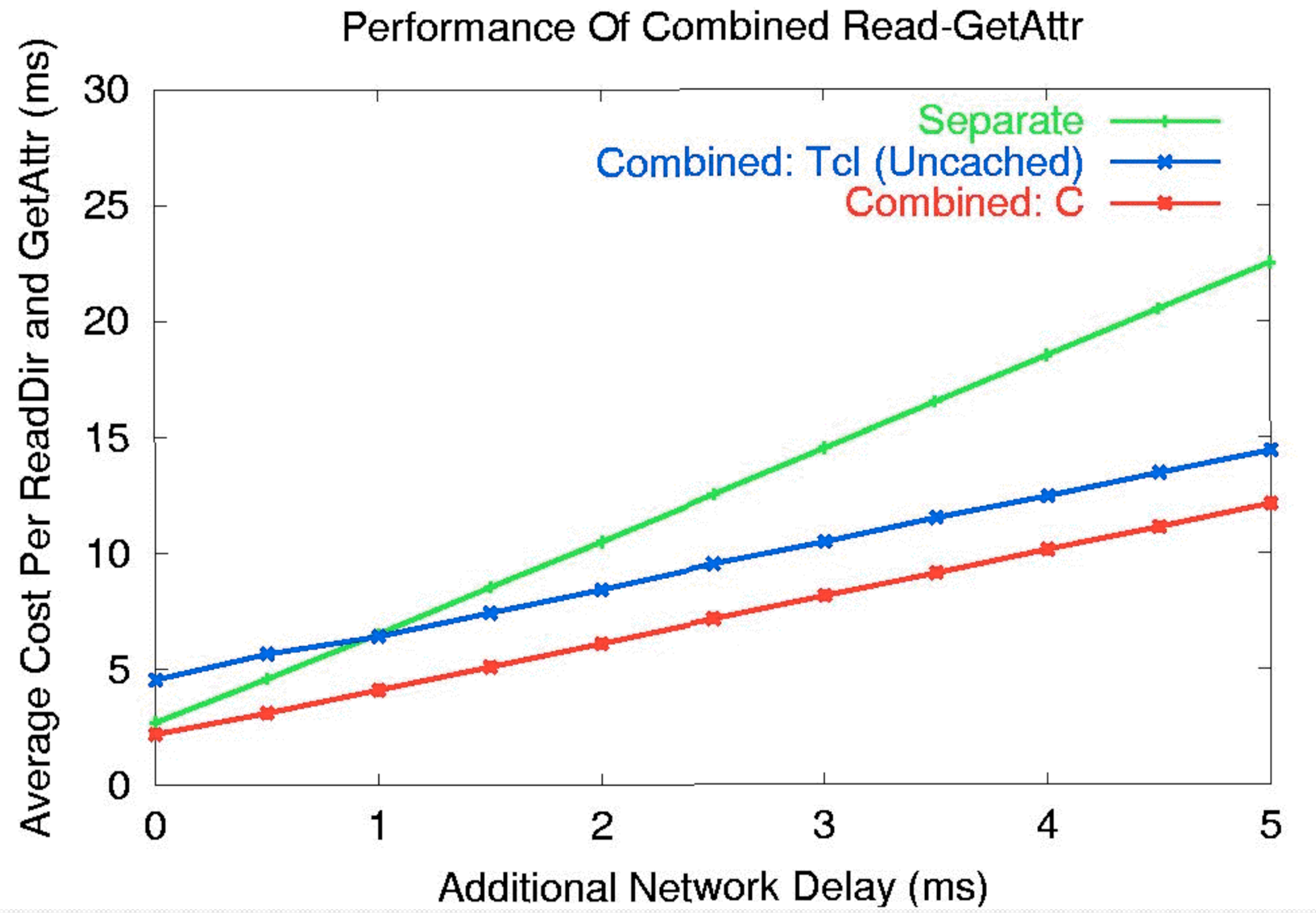
Pathname lookup: Benefits



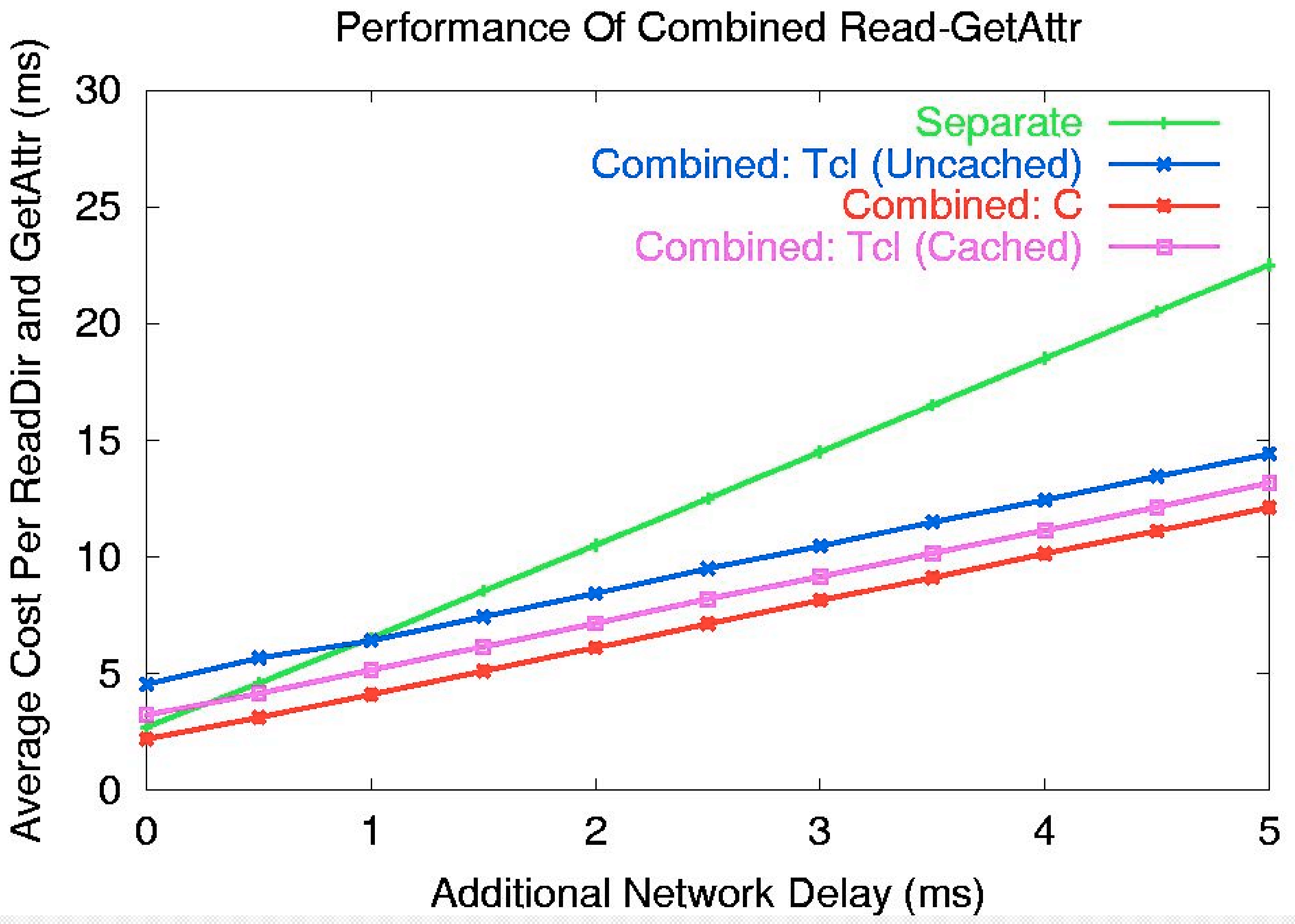
Pathname lookup: Benefits



Pathname lookup: Benefits



Pathname lookup: Benefits



Performance: Summary

- Examples only illustrative
  - Other “compositions” possible too!
- Micro-benchmarks
  - Benefit due to reduced network roundtrips
- Macro-benchmarks
  - Postmark: 54% less network traffic
  - TPC-B: 96% less network traffic
- Facilitates working around minimal interfaces

Outline

- *Motivation*
- *Scriptable RPC*
- *Case Study: Active Storage*
  - *Performance*
  - Functionality
  - Simplicity
- Summary

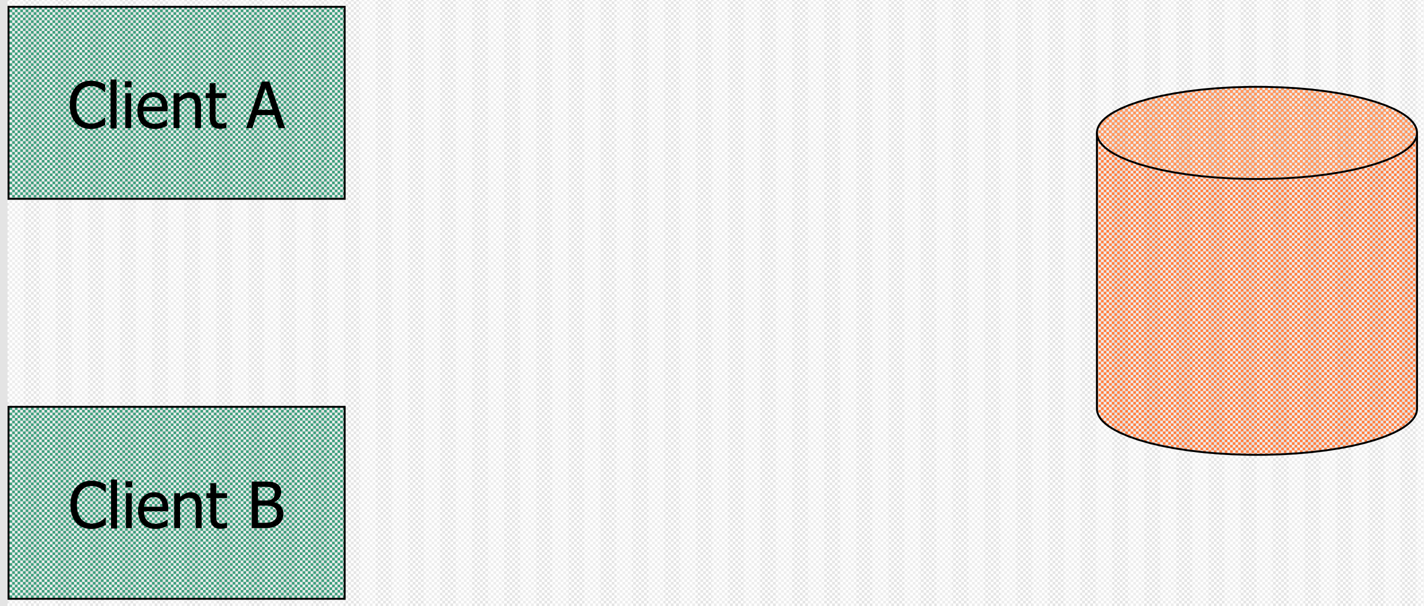
ScFS: Functionality enhancements

- Implement enhanced *virtual protocols* over physical protocols
  - State can be added to stateless protocols
  - System provides primitives
    - Clients compose them into desired functionality
- Examples
  - AFS consistency semantics over NFS
  - Sprite consistency semantics over NFS

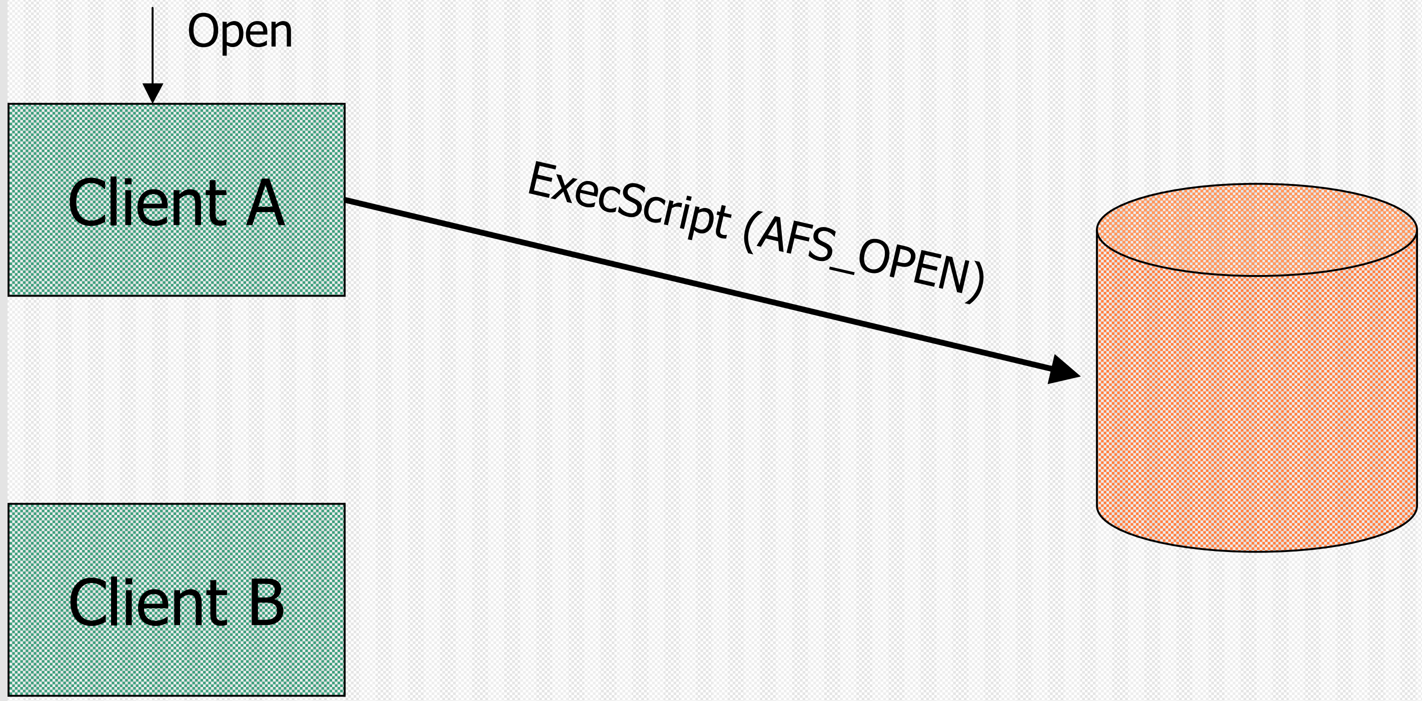
Consistency semantics: NFS vs AFS

- NFS
  - Stateless server
  - Client checks periodically for updates
- AFS
  - Write-on-close semantics
  - Server tracks clients caching a file
    - Notifies clients when modified file written
  - Requires server-side state, participation
    - Cannot implement using existing paradigms

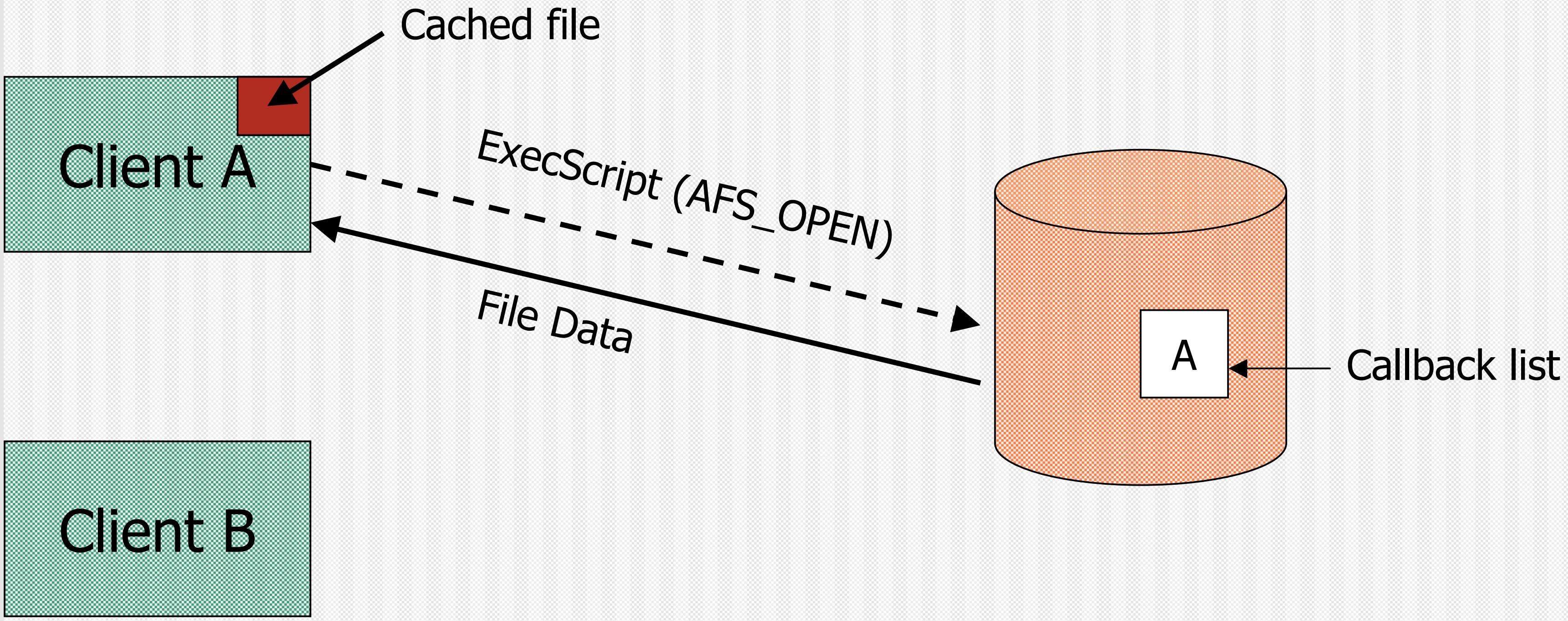
Scripted AFS consistency



Scripted AFS consistency



Scripted AFS consistency



Cached file

Client A

ExecScript (AFS_OPEN)

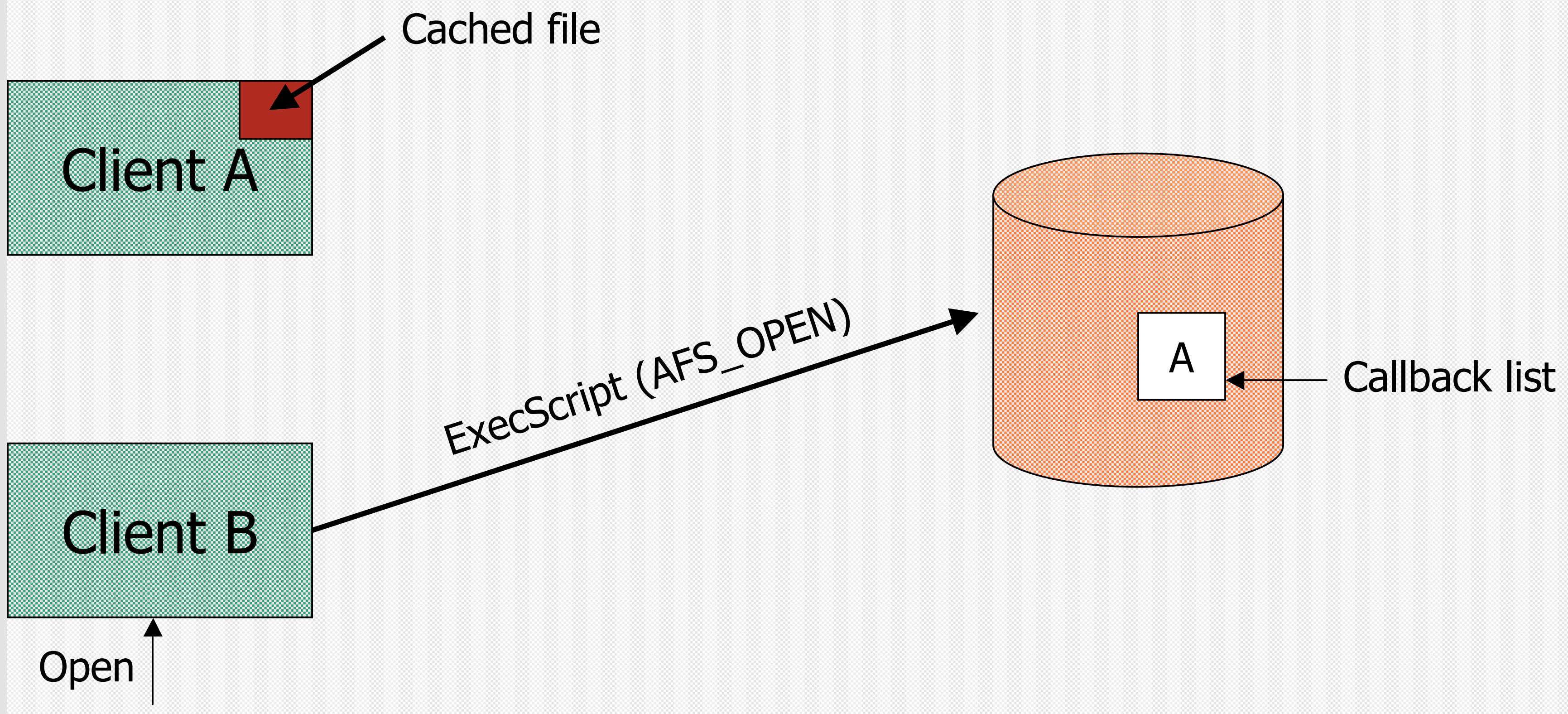
File Data

A

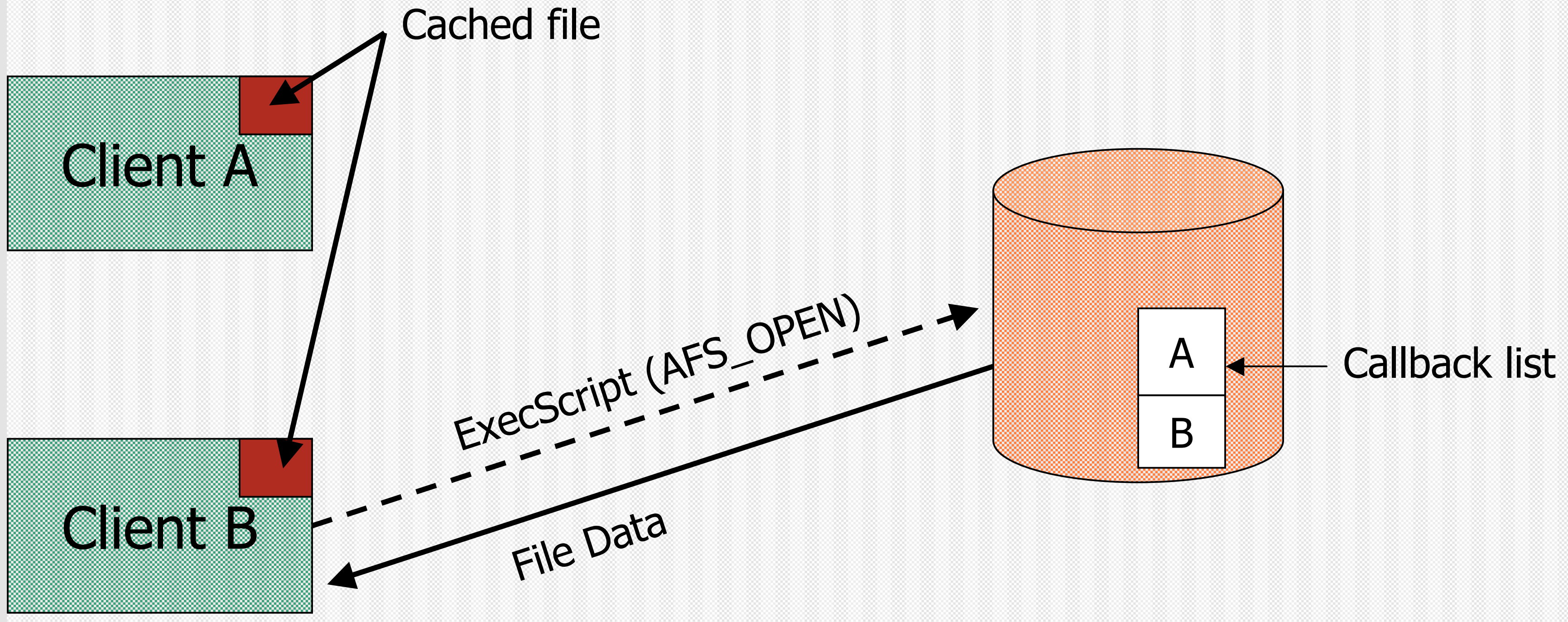
Callback list

Client B

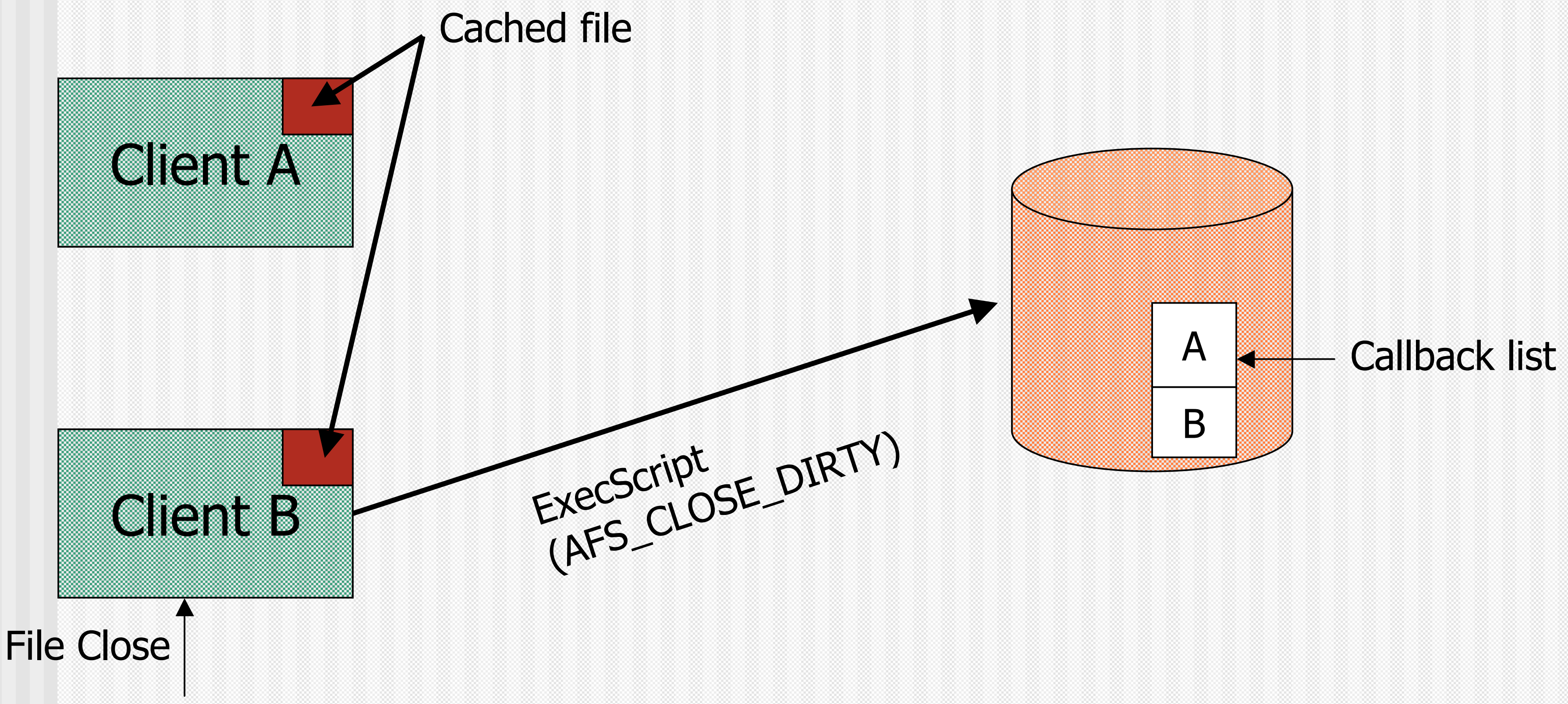
Scripted AFS consistency



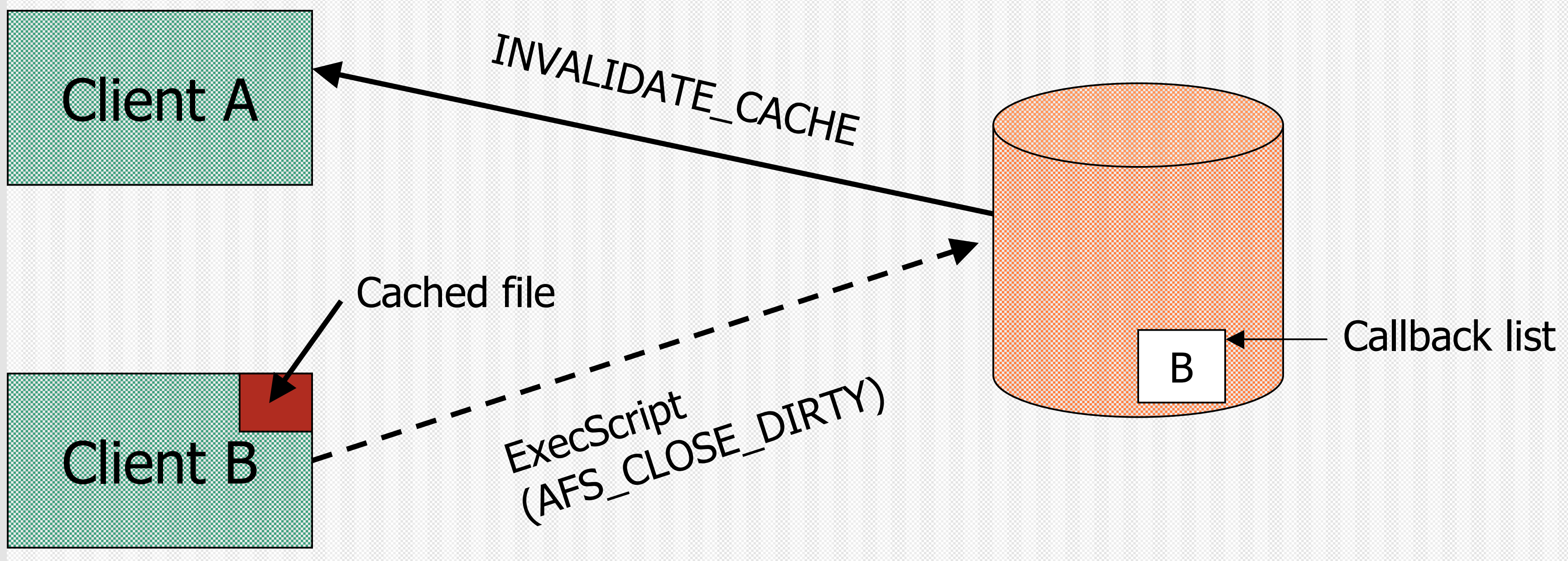
Scripted AFS consistency



Scripted AFS consistency



Scripted AFS consistency



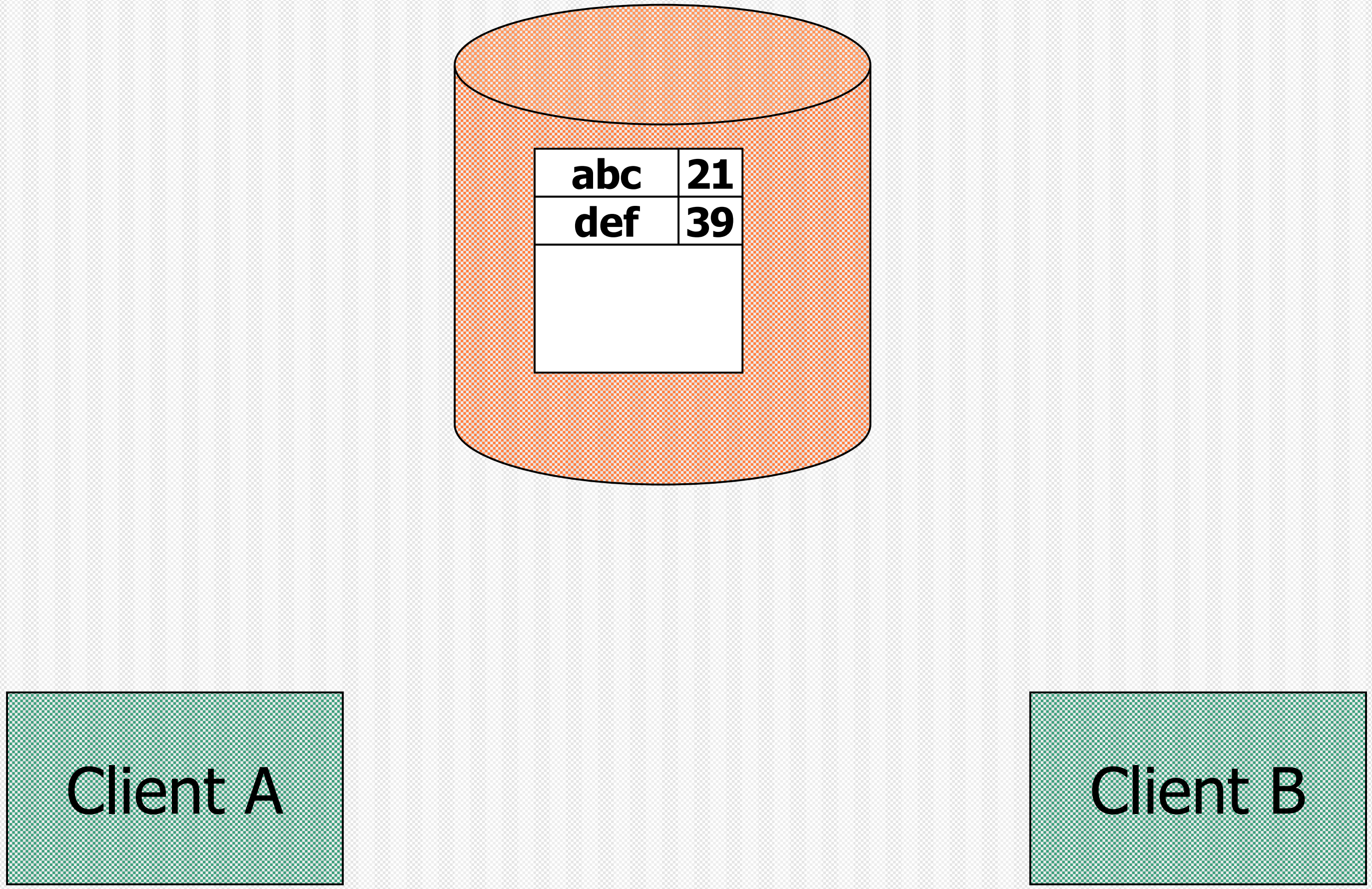
Functionality: Summary

- SRPC: Powerful
  - Possible to add complex functionality
  - Even those requiring augmenting server state
- SRPC: Simple
  - AFS consistency
    - 2 scripts, < 10 lines each
  - Sprite consistency
    - 3 scripts, < 20 lines each
- Simple base system, compact scripts to extend it

ScFS: Simplicity enhancements

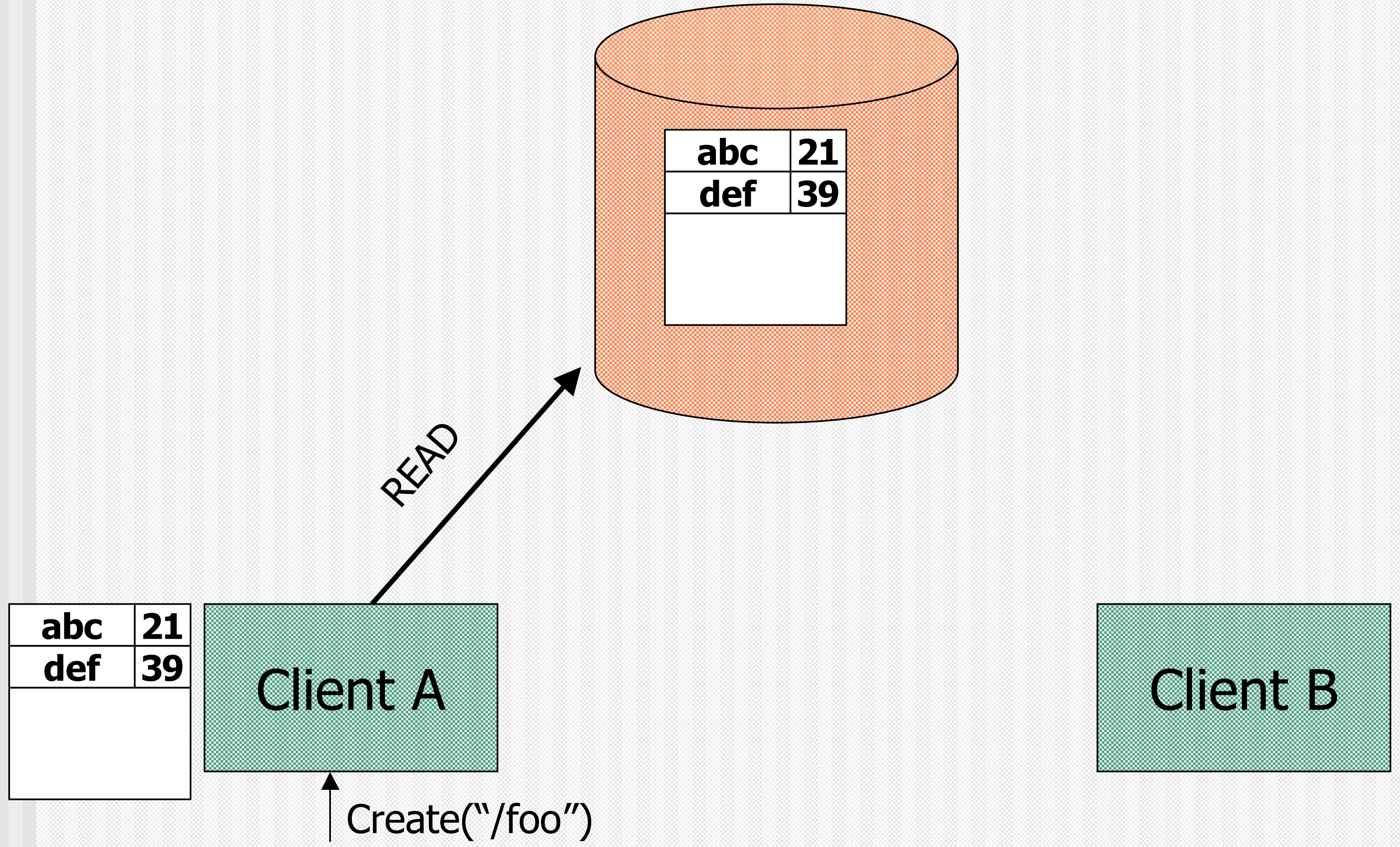
- Ability to group operations at server
  - Simplifies implementation of atomic sets of operations
  - Often, obviates need for distributed locks, distributed crash recovery
- Example - concurrent directory updates

Concurrent directory updates



abc	21
def	39

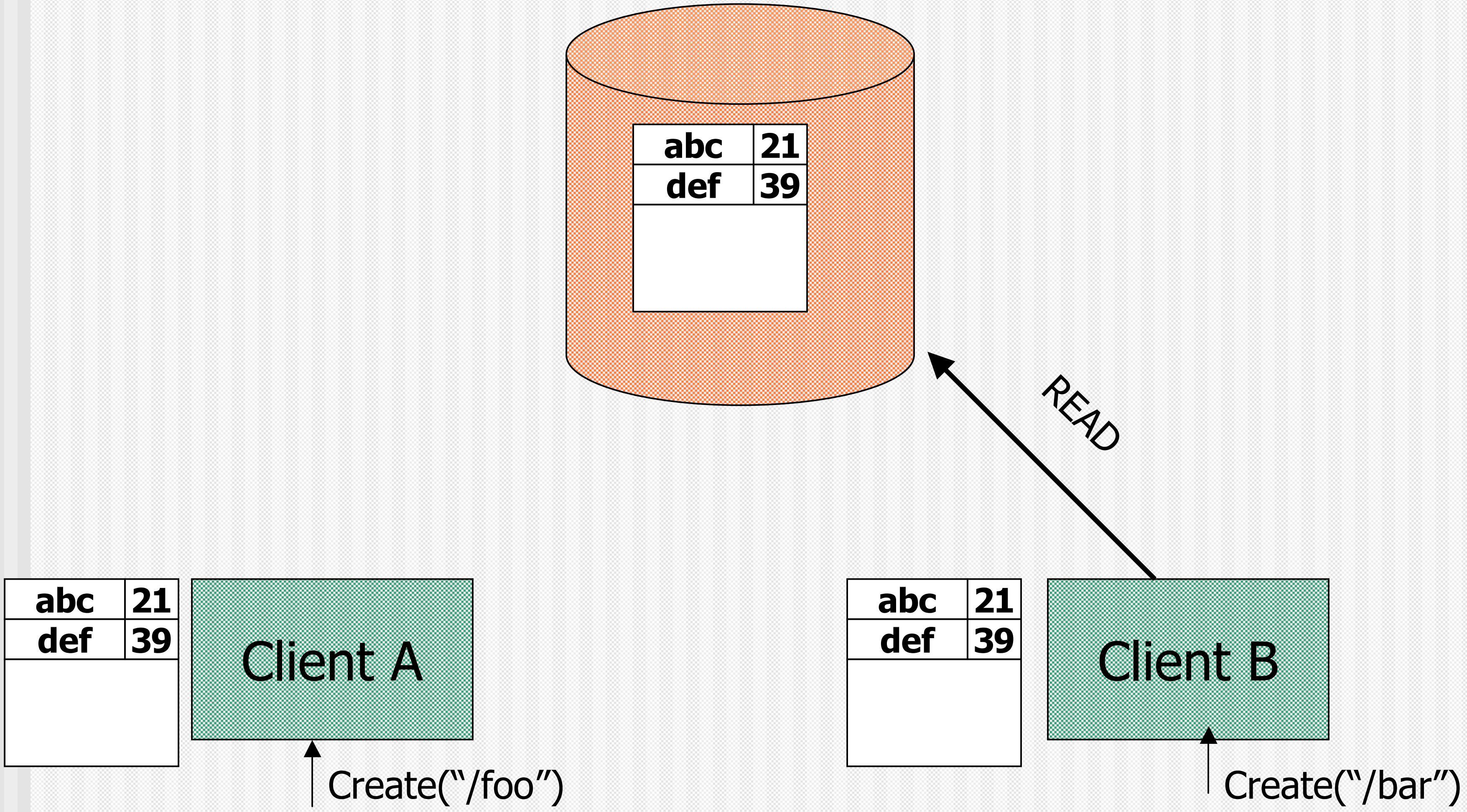
Concurrent directory updates



abc	21
def	39

abc	21
def	39

Concurrent directory updates

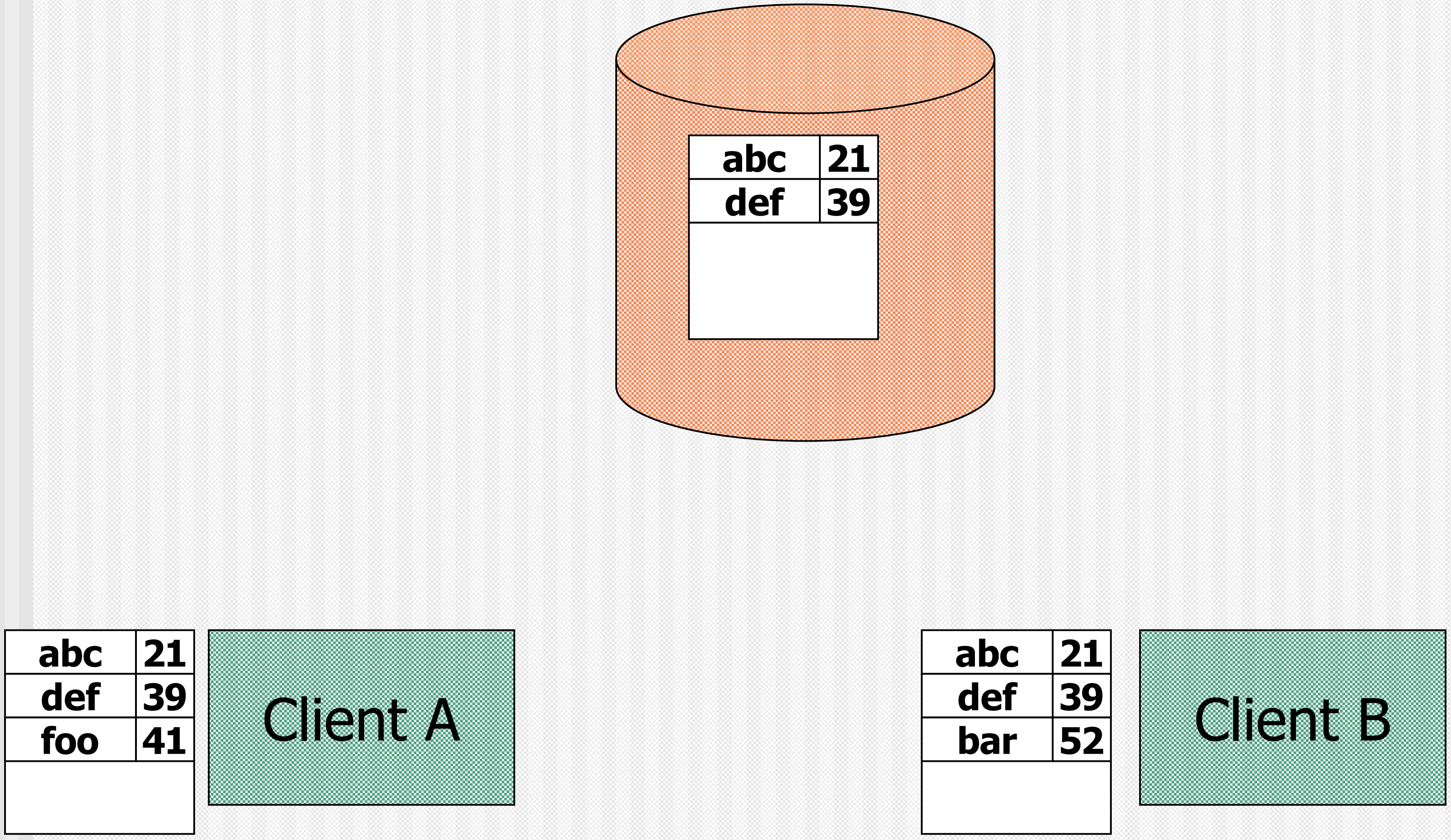


abc	21
def	39

abc	21
def	39

abc	21
def	39

Concurrent directory updates

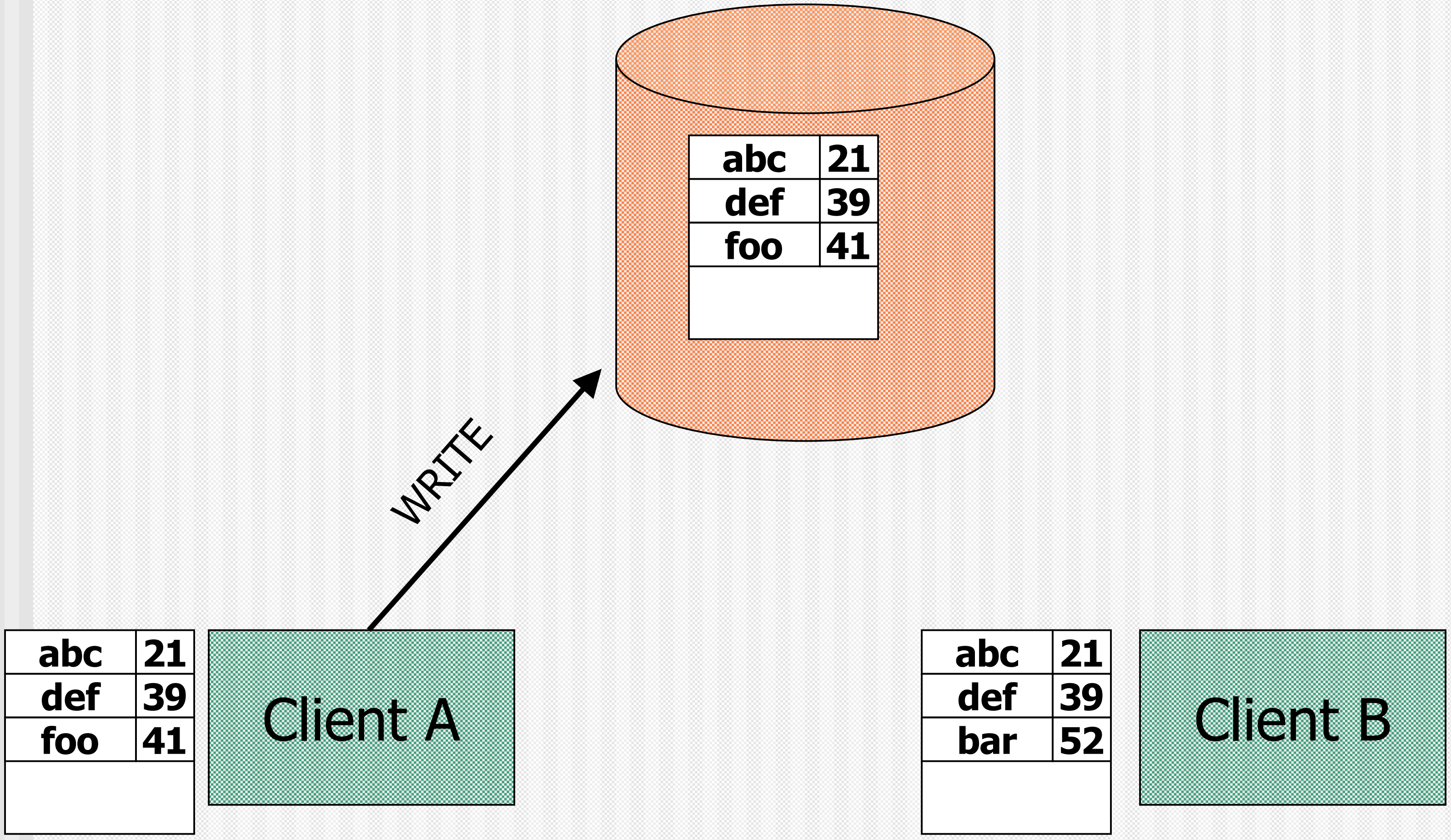


abc	21
def	39

abc	21
def	39
foo	41

abc	21
def	39
bar	52

Concurrent directory updates



abc	21
def	39
foo	41

WRITE

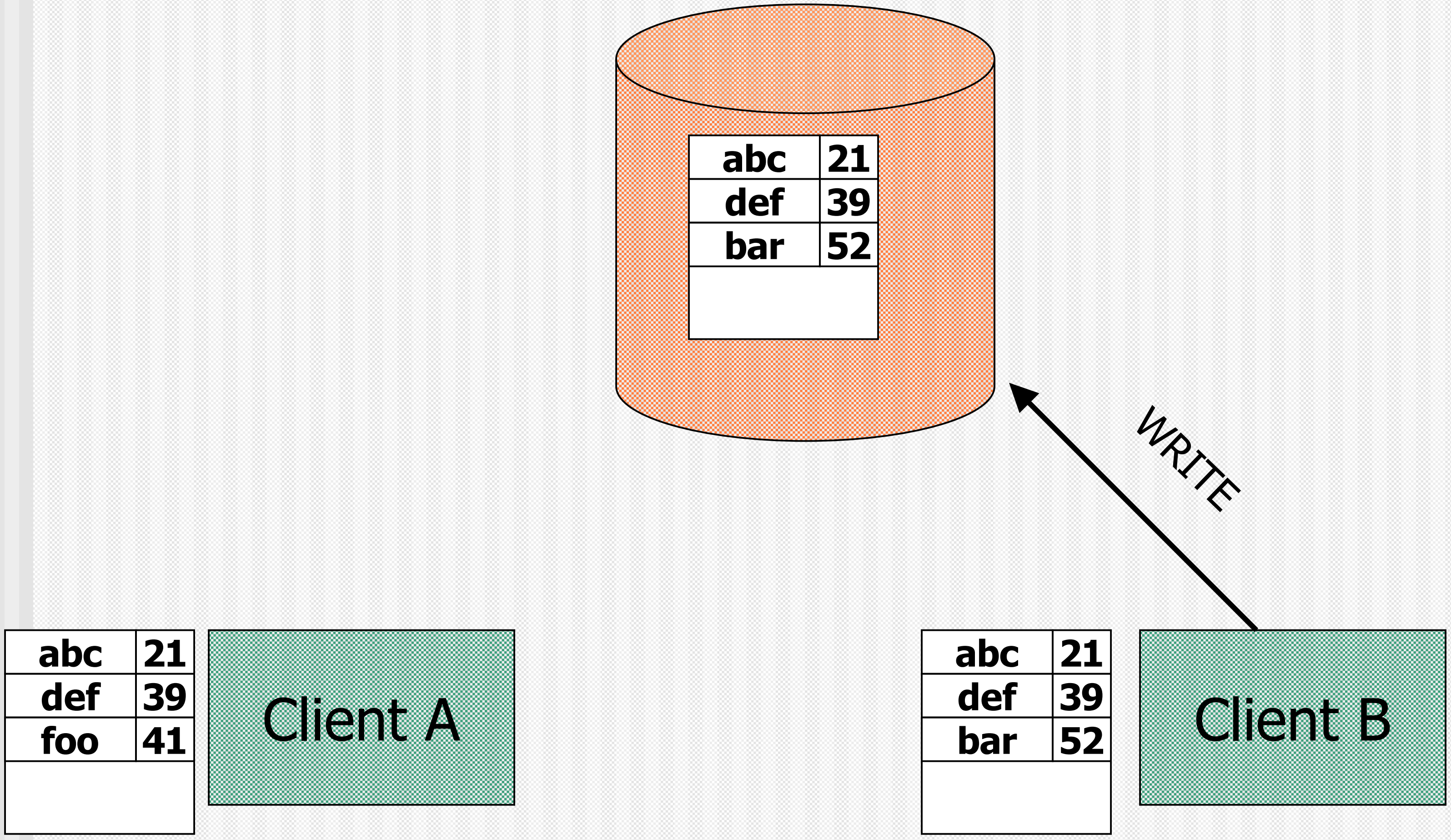
abc	21
def	39
foo	41

Client A

abc	21
def	39
bar	52

Client B

Concurrent directory updates



abc	21
def	39
bar	52

abc	21
def	39
foo	41

abc	21
def	39
bar	52

Concurrent directory updates

- Non-scripting
  - Distributed locking, distributed crash recovery
    - Clients acquire locks before read-modify-write
    - Recover from client failures while holding locks
- SRPC
  - Script acquires in-memory lock at server
  - Just enforce mutual exclusion within single address space

Summary

- SRPC
  - High Performance, rapid extensibility, simplicity
  - Makes effective use of “active” architecture
- All scripts less than 20 lines of code
  - Some implement non-trivial functionality
- Fewer lines of code =>
  - Fewer bugs, more robust systems
  - Ease of building systems with active components
- Don’t have a complex system catering to all client requirements
  - Provide primitives, enhance with compact scripts

Questions ?

Wisconsin Network Disks Group

http://www.cs.wisc.edu/wind