Fancy File Systems

Bringing It All Together

- We've been studying how an OS manages a single CPU system
- As part of that, it will communicate with I/O devices
- The network is one of those I/O devices
  - Networks are controlled by complex software that enables the proper routing of data and the appearance of reliability
  - As a programmer, we access the network through an abstraction called a socket
- The network may connect machines running a single application, that's a distributed system
  - Loosely-coupled, different machines
  - Event Ordering: Happened-Before Relation, Logical Clocks
  - Atomicity of Transactions: Two Phase Commit (2PC)—the actions eventually happen
  - Electing a Leader: Bully Algorithm
- Parallel systems also exist, and an OS manages those, too
  - Includes multicore systems and SMP systems
  - The system must share a clock, an OS, and physical memory
- Parallel programming techniques are used in both parallel and distributed applications
  - Include shared memory and message passing

Networked and Distributed File Systems

- Provide transparent access to files stored on remote disks
- Issues:
  - *Naming*: How do we locate a file?
  - *Performance*: How well does a distributed file system perform as compared to a local file system?
  - *Failure handling*: How do applications deal with remote server failures?
  - *Consistency*: How do we allow multiple remote clients to access the same files?

NFS

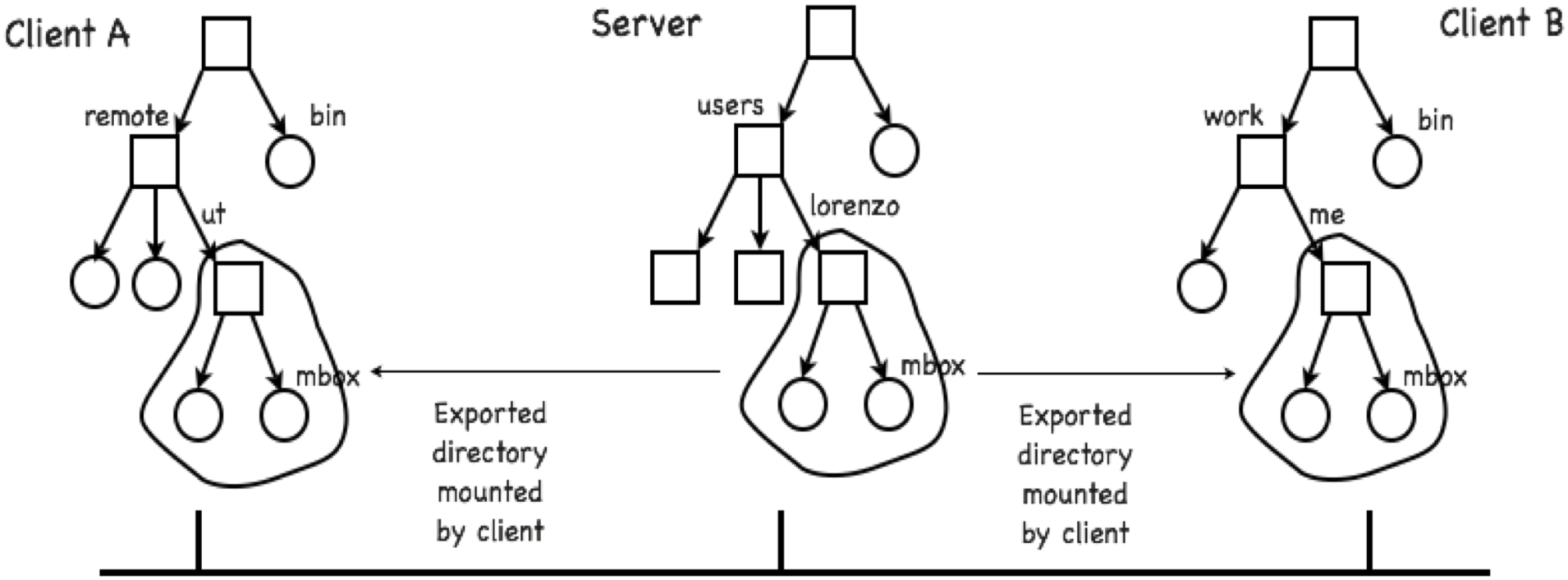
Sun's NFS: Overview

- Network File System (NFS)
- Developed by Sun Microsystems in 1984
  - Most heavily used distributed file system
  - Used by UTCS
- Files are stored on a remote file server
- Client accesses files transparently
  - Users may never even know their files are not local

Finding Files in NFS

- Uses implicit naming for files
  - Location transparency: file name does not include name of the server where the file is stored
  - As opposed to explicit naming:
    - <file server: file name >
    - //anslaptop/Users/ans/Desktop
- Finds the server using a static, location-transparent mapping (NFS Mount protocol)
  - Mount remote directories as local directories
  - Maintain a mount table with directory → server mapping, e.g.,
    - mount zathras:/vol/vol0/users/ans /home/ans

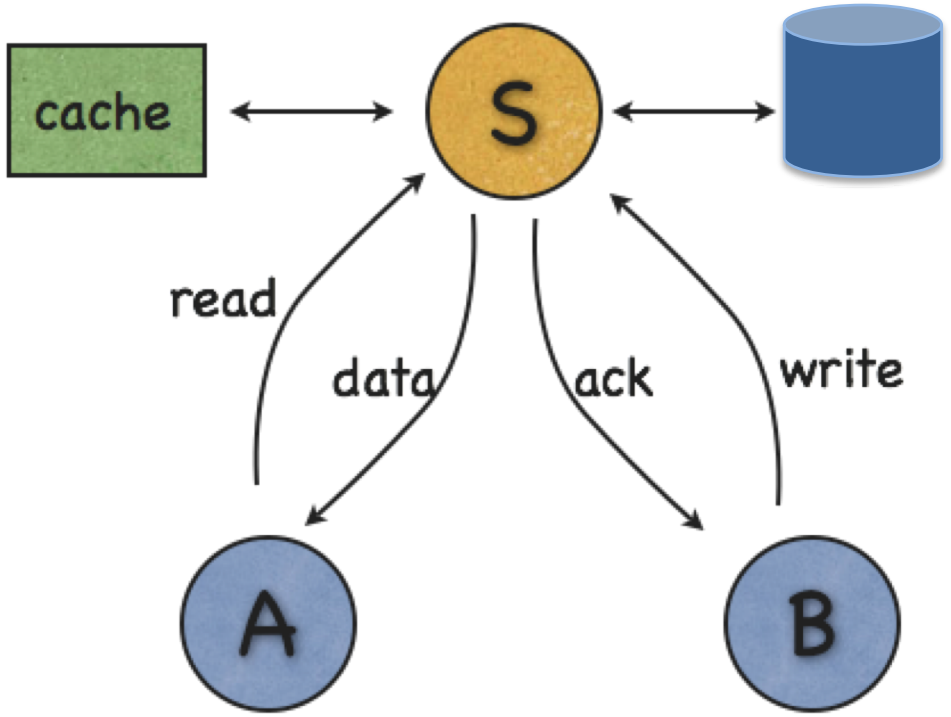
Example: Mounting a Directory



Performance: Simple Case

Straightforward use of RPC

- Use RPC to forward every file system request (e.g., open, seek, read, write, close, etc.) to the remote server
- Remote server executes each operation as a local request
- Remote server responds back with the result



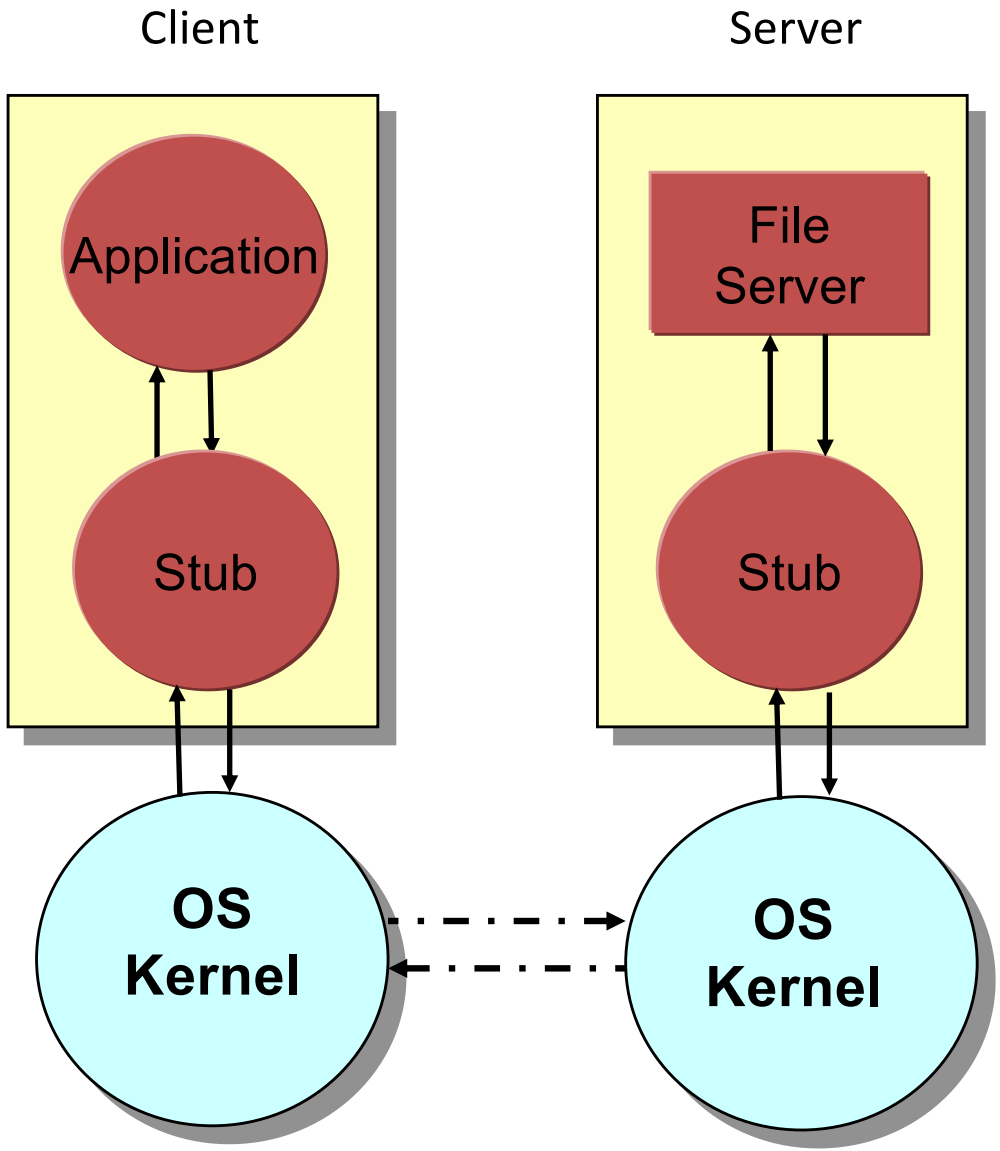
Remote Procedure Calls

Remote Procedure Calls (RPC)

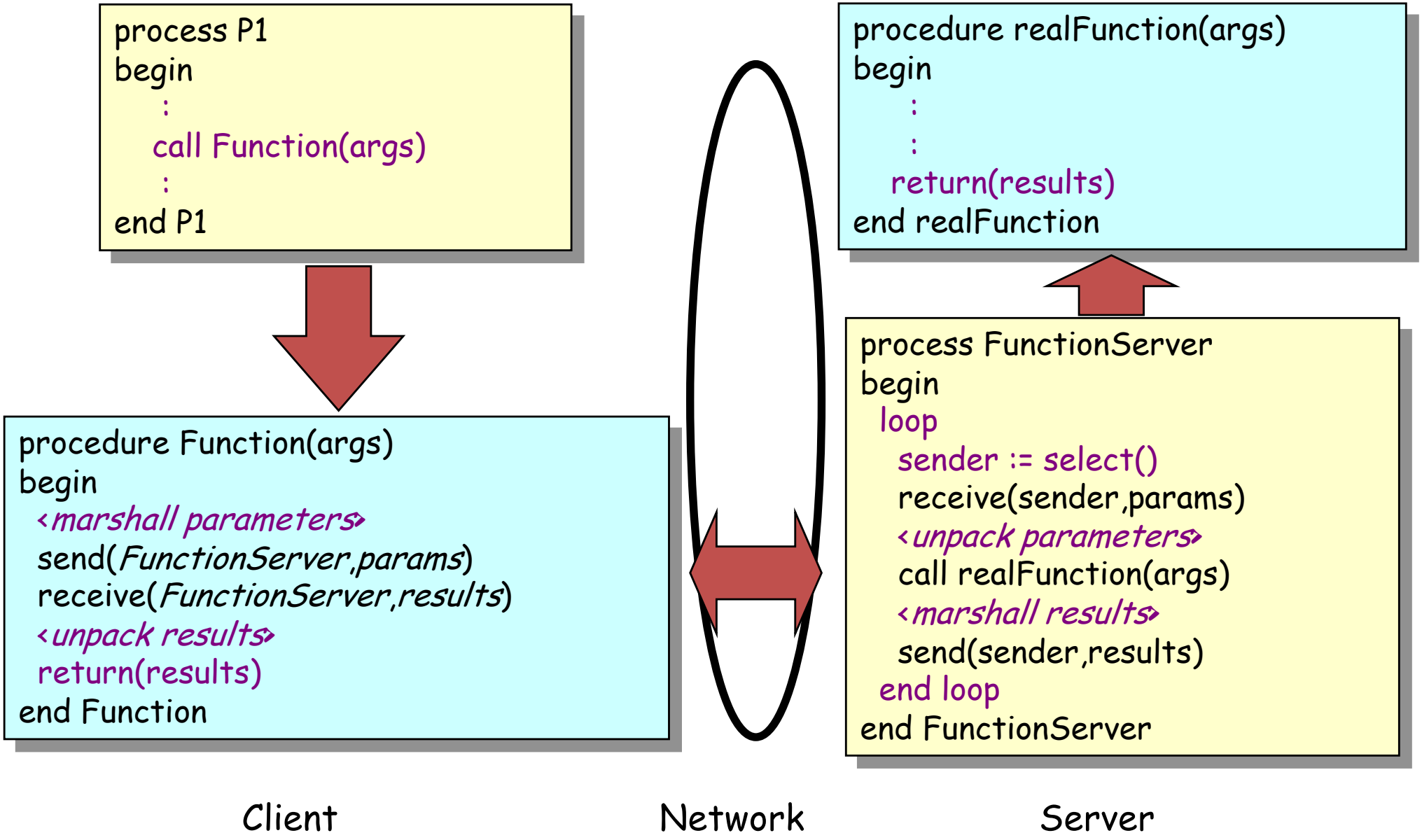
- Clients execute services on a remote server using a procedure call
- Servers export procedures for clients to call
- Communication details are hidden from the client and handled by the *RPC Mechanism*
- Is *NOT* message passing!
  - Not this is *not* message passing, which requires more work for the application

RPC: High-Level

- Remote procedure calls abstract out the *send-wait-reply* paradigm into a “procedure call”
- Remote procedure calls can be made to look like “local” procedure calls by using a *stub* that hides the details of remote communication



RPC: Lower-Level



RPC: Lower-Level

The RPC mechanism uses the procedure *signature* (number and type of arguments and return value)

1. to generate a client stub that bundles RPC arguments and sends them off to the server.
2. to generate the server stub that unpacks the message and makes the procedure call

The stubs actually do the work, and the client and server communicate through the stubs.

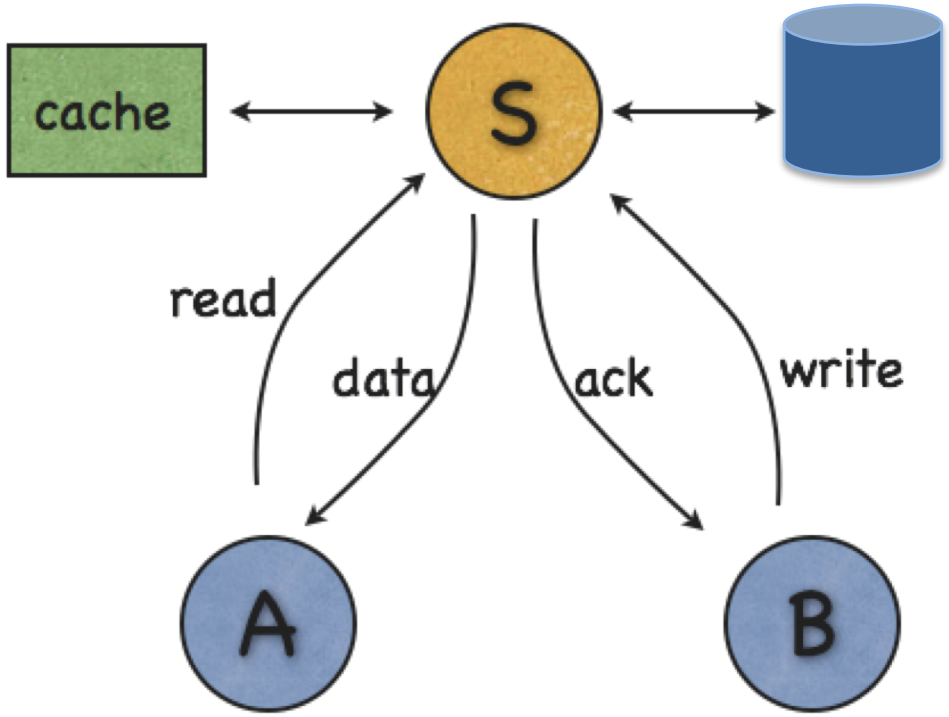
Problems with RPC

- Failure handling
  - A program may hang because of
    - Failure of a remote machine; or
    - Failure of the server application on the remote machine
  - An inherent problem with distributed systems, not just RPC
    - Lamport: “A distributed system is one where you can’t do work because some machine that you have never heard of has crashed”
- Performance
  - Cost of procedure call << same machine RPC << network RPC

Back to NFS

Performance: Simple Case

- Simple case: straightforward use of RPC
  - Use RPC to forward every file system request (e.g., open, seek, read, write, close, etc.) to the remote server
  - Remote server executes each operation as a local request
  - Remote server responds back with the result
- Advantage: Server provides a consistent view of the file system to distributed clients.
  - What does consistent mean?
- Disadvantage: Poor performance



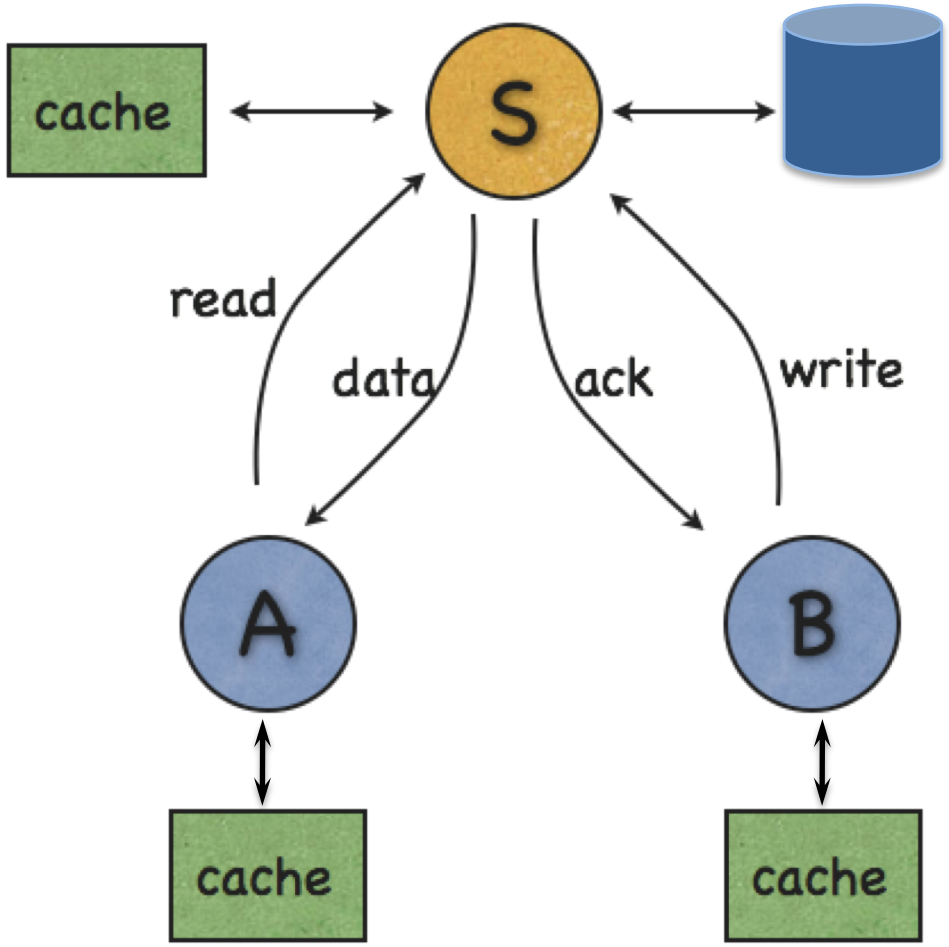
Question

Why does turning every file system operation into an RPC to a remote server perform poorly?

A. Disk latency is larger than network latency
B. Network latency is larger than disk latency
C. No server-side cache
D. No client-side cache

Sun's Network File System (NFS)

- Cache data blocks, file headers, etc. both at client and server
  - Generally, caches are maintained in memory; client-side disk can also be used for caching
  - Cache update policy: write-back or write-through
- *Advantage*: read, write, stat, etc. can be performed locally
  - Reduce network load and
  - Improve client performance



Sun's NFS Issues: Consistency

- What if multiple clients share the same file?
  - Easy if both are reading files …
  - But what if one or more clients start modifying files?

Consistency Models

Consistency Models

- Shared data is kept in a data store
  - a register, a file system, a database...
- Clients access the data store through read and write operations
- Consistency Semantics: a contract between the data store and its clients that specifies the results that clients can expect to obtain when accessing the data store

Coherence, Staleness, and Consistency

- Coherence: restricts order of reads and writes to one location
- Staleness: bounds maximum (real-time) delay between writes and reads to one location
- Consistency: restricts orders of reads and writes across locations

Coherence

- A read should return the result of the latest write to a memory location

```
P1:
for (ii = 0; ii < 100; ii++) {
   write(A,ii);
}
```

```
P2:
while(1){
   printf("%d", read(A);
}
```

- A memory that is not coherent may return

  1 2 3 3 3 4 8 10 9 11 12 13

- How could this happen?
  - writer sends updates via Internet; updates are reordered in route
  - reader switches between two servers (e.g. redirected to a different Akamai node)

Staleness

```
P1:
while(1) {
    sleep(1000ms);
    write (T, A, "At %t price is %d\n");
}
```

```
P2:
while(1){
    sleep (1000ms);
    printf("%s", read (A);
}
```

- Assume perfectly synchronized clocks and a real time OS
- Reads may return

At 1:00:00 price is 10.50
At 1:00:01 price is 10.55
At 1:00:02 price is 10.65
At 1:00:02 price is 10.65
At 1:00:04 price is 13.18….

- Why staleness?
  - Polling interval
  - Cache update/invalidation delayed by network

Consistency

- Restricts order of reads and writes across locations

```
P1:
for (ii = 0; ii < 100; ii++) {
    write(A,ii);
    write(B,ii);
}
```

```
P2:
while(1){
    printf("(%d, %d)", read(A), read(B));
}
```

- A memory that is not consistent may return

(0,0),(1,0),(2,2),(3,3),(4,5),(9,8),(9,9),(9,10),(10,10),(11,10),(11,11),(12,12)

Consistency Semantics

Sequential Consistency

- “The result of any execution is the same as if the operations of all the processes were executed in some sequential order and the operations of each individual process appear in this sequence in the order specified by its program” (Lamport, 1979)
- In other words: create a total order that includes all the operations of the execution, such that:
  - the total order respects the local history of each process
  - every read returns the result of the latest write, according to the total order (data coherence)

Limitations of Strong Consistency

- Implementing strong semantics has intrinsic costs
  - Lipton and Sandberg: in sequential consistency can have either fast reads or fast writes, but not both
- CAP theorem: it is impossible in a distributed system to provide simultaneously
  - Consistency: all nodes see the same data at the same time, even in the presence of updates
  - Availability: every request receives a response
  - Partition Tolerance: the system properties hold even when the system is partitioned
- One can only have two properties out of three...

Weakening Sequential Consistency: FIFO Consistency

"Writes done by a single process are seen by all other processes in the order in which they were issued, but writes from different processes may be seen in a different order by different processes" (PRAM consistency, Lipton and Sandberg 1988)

Weakening Sequential Consistency: Causal Consistency

Writes that are potentially causally related must be seen by all processes in the same order. Concurrent writes may be seen in a different order on different machines. (Hutto and Ahamad, 1990)

Back to NFS

Sun's NFS Issues: Consistency

- What if multiple clients share the same file?
  - Easy if both are reading files …
  - But what if one or more clients start modifying files?
- *Client-initiated weak consistency protocol*
  - Clients poll the server periodically to check if the file has changed
  - When a file changes at a client, server is notified
    - Generally, using a delayed write-back policy
  - Clients on detecting a new version of file at the server obtain a new version
- Consistency semantics determined by the cache update policy and the file-status polling frequency
- Other possibility: server-initiated consistency protocol

Sun's NFS Issues: Failures

- What if the server crashes? Can client wait for the server to come back up and continue as before?
  - Data in server memory can be lost
  - Client state maintained at the server is lost (e.g., seek + read)
  - Messages may be retried
- What if clients crash?
  - Lose modified data in client cache

NFS Protocol: Statelessness

- NFS uses a *stateless* protocol
  - Server maintains *no state* about clients or open files
    - Except as hints to improve performance
  - Each file request must provide complete information
    - Example: ReadAt(inode, position) rather than Read(inode)
- *Idempotent operations*: All requests can be repeated without any adverse effects
  - "at least once" semantics
- Server failures are (almost) transparent to clients
  - When server fails, clients:
    - hang until the server recovers or
    - crash after a timeout
  - Why not return an error?

Question

Accessing files over NFS is much more complicated for user applications than if they were using the local file system.

A. True
B. False

NFS: Summary

- Key features
  - Location-transparent naming
  - Client-side and server-side caching for performance
  - Stateless architecture
  - Client-initiated weak consistency
- Advantages
  - Simple
  - Highly portable
- Disadvantages
  - Lack of strong consistency

Google File System

Google File System: Design Considerations

- Files are huge by traditional standards
  - Multi-GB files are common
  - Billions of objects
- Most modifications are appends
  - Random writes are practically nonexistent
  - Many files are written once and read sequentially
- Two types of reads
  - Large streaming reads
  - Small random reads (in the forward direction)

Google File System: Design Considerations

- Files are stored on a 10K-100K cluster of servers
- Component failures are the norm
  - 1000s of components
  - Running on commodity Linux machines
  - Bugs, human errors, failures of memory, disk, connectors, networking, and power supplies
  - Monitoring, error detection, fault tolerance, automatic recovery
- Sustained bandwidth more important than latency

Design Goal: Scaling!

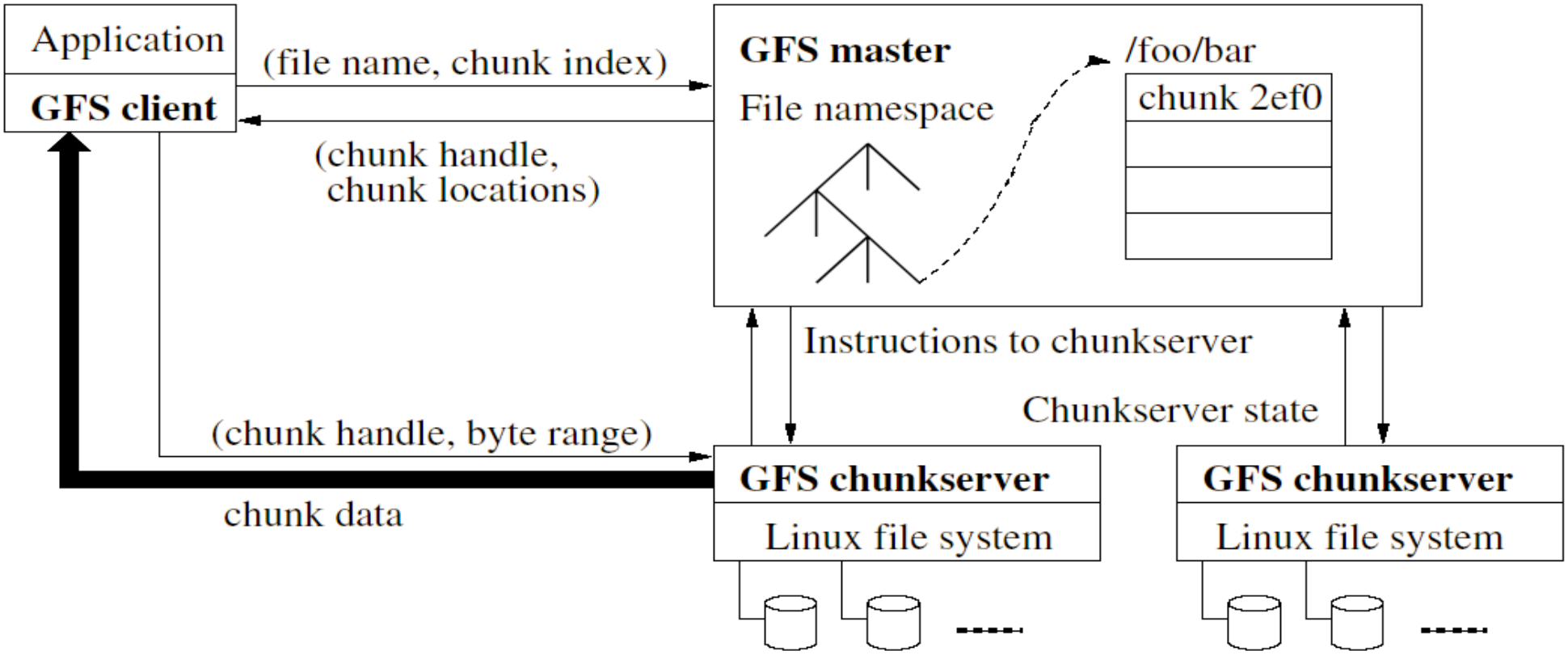
- Many (often large) files to many people
- Three main ways to achieve scalability:
  - Distribute: partition data and computation across multiple machines
  - Replicate: make copies of data available at different machines
  - Cache: allow client processes to access local copies

GFS Design

GFS: Architectural Design

- A GFS cluster
  - A single *master*
    - Stores metadata
    - Manages access control
    - Single point of failure
  - Multiple *chunkservers* per master
    - Reading/writing goes directly through the chunkservers
    - Accessed by multiple *clients*
- A file
  - Divided in fixed-sized *chunks*
    - Chunks are 3-way replicated and stored at chunkservers
    - Each chunk has a 64-bit unique global IDs
    - Chunks are each 64 MB (much larger than a normal FS block!)

GFS Architectural Design



Consistency Model

- Two types of updates
  - Writes (at application-specified offset)
  - Record appends (at offset of GFS's choosing)
- Relaxed consistency
  - Concurrent changes are consistent but undefined
    - *undefined* means that all clients may not see modification in its entirety (change may not appear to be atomic)
  - A record append is atomically committed at least once
    - Even during concurrent updates
    - Offset returned to the client marks beginning of defined region
    - Occasional padding/duplications
- Updates are applied in the same order at all replicas
  - Use chunk version number to detect stale replicas
    - They are garbage collected ASAP
- Client read may temporarily see stale values
  - Staleness limited by timeout on cache entry

Question

The Google File System would be a good choice for the file system on your home computer.

A. True
B. False

Summary

- Distributed file systems have more challenges than local file systems
- Consistency models provide the user with guarantees about how and when updates will be seen in a distributed file system.
- NFS (Sun's version, 1984) uses:
  - RPC and a level of abstraction to provide seamless access for the OS kernel
  - mount to allow multiple file systems to appear to the user as a single file system
- GFS is optimized for very large files with particular access patterns and for a very large number of simultaneous clients