Advanced Operating Systems: Three Easy Pieces

Distributed Operating Systems

Outline

- What is Distributed System
- Key characteristics of Distributed Systems
- Challenges with distributed System
- Build reliable messaging over unreliable layers
- What is an RPC
- Distributed File System:
  - NFS Architecture
  - Client caching

What is Distributed System

What is a Distributed System

A distributed system is one where a machine I've never heard of can cause my program to fail.

— ***Leslie Lamport***

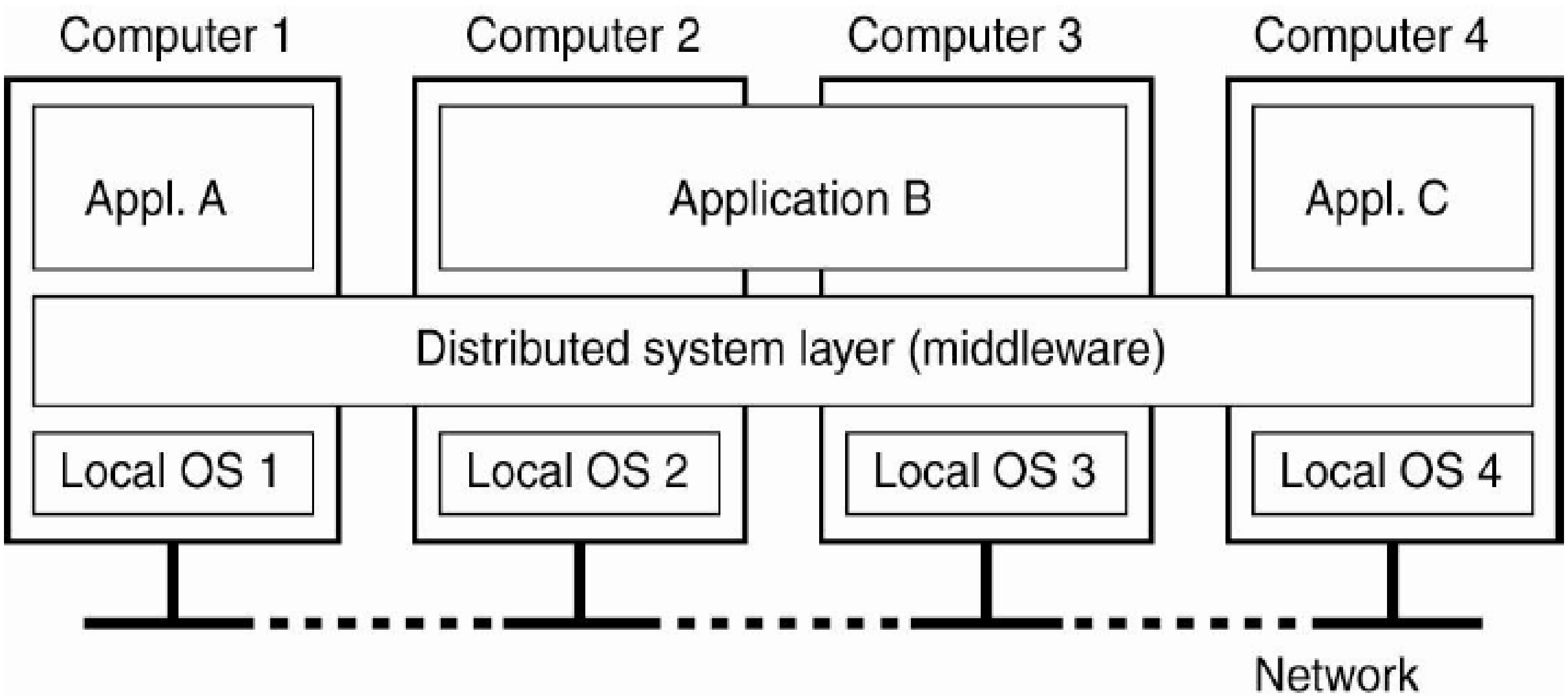
Definition:

More than 1 machine working together to solve a problem

Examples:

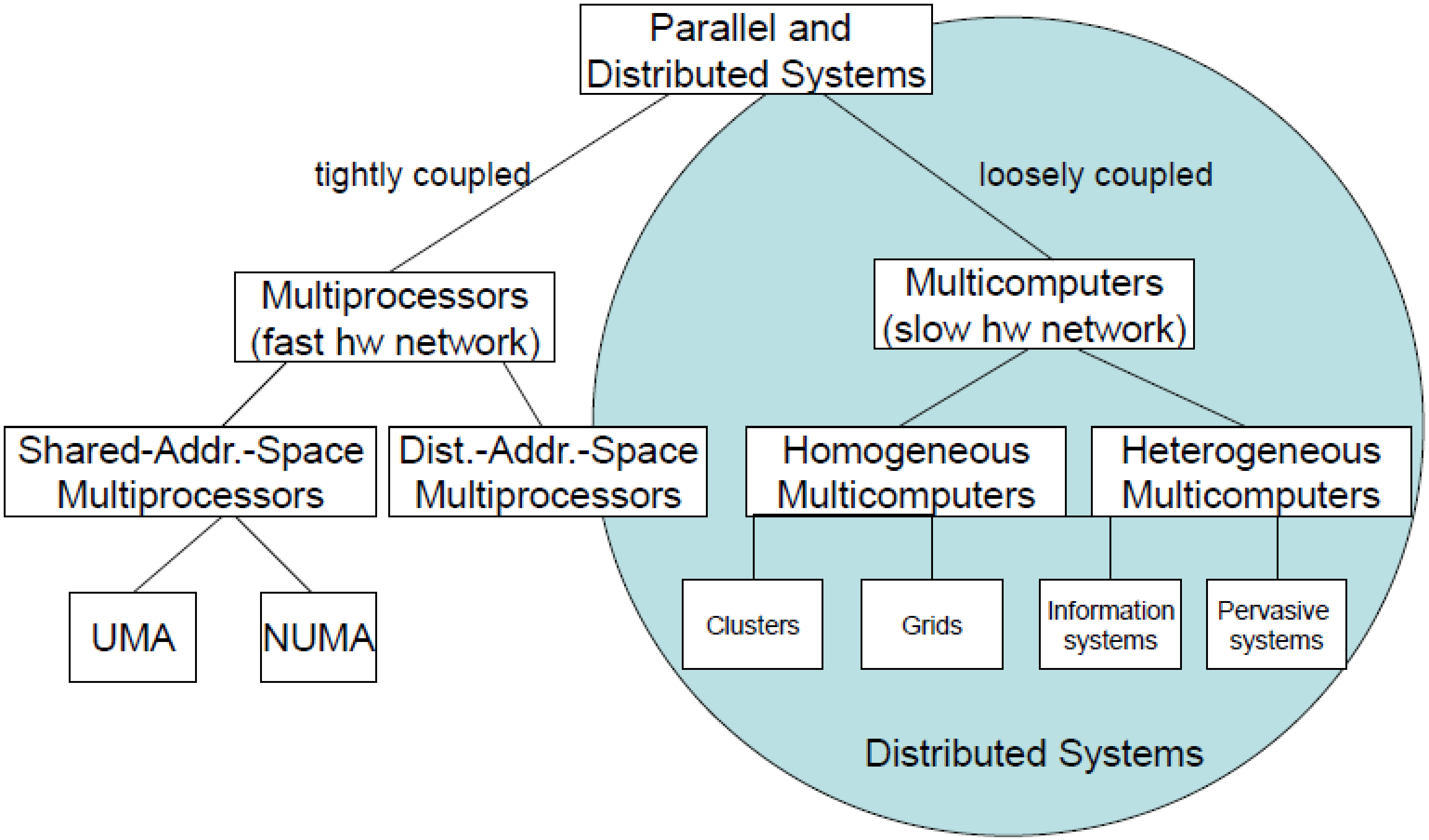
- **Client/server:** web server and web client
- **Cluster:** page rank computation
- **Peer-2-Peer:** Twitter, etc.

High-level Architecture (1)

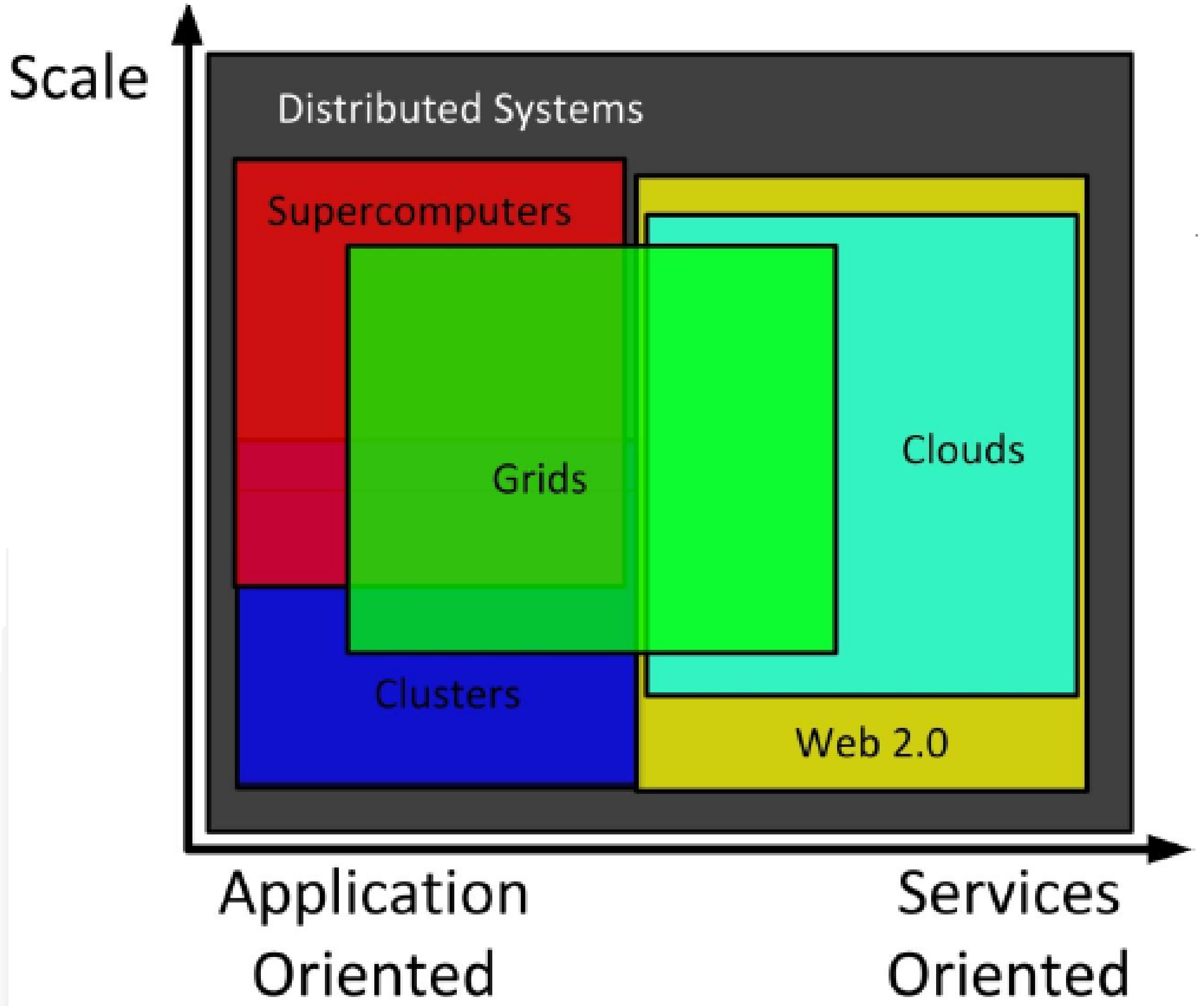


A distributed system organized as middleware. The middleware layer extends over multiple machines, and offers each application the same interface.

High-level Architecture (2)



Distributed Systems: Clusters, Grids, Clouds, and Supercomputers



Why go Distributed?

- More computing power
- More storage capacity
- Fault tolerance
- Data sharing
- Why Cloud

Key Characteristics of Distributed Systems

Key Characteristics

- Support for resource sharing
- Openness
- Concurrency
- Scalability
- Fault tolerance (reliability)
- Transparency (distribution transparency)

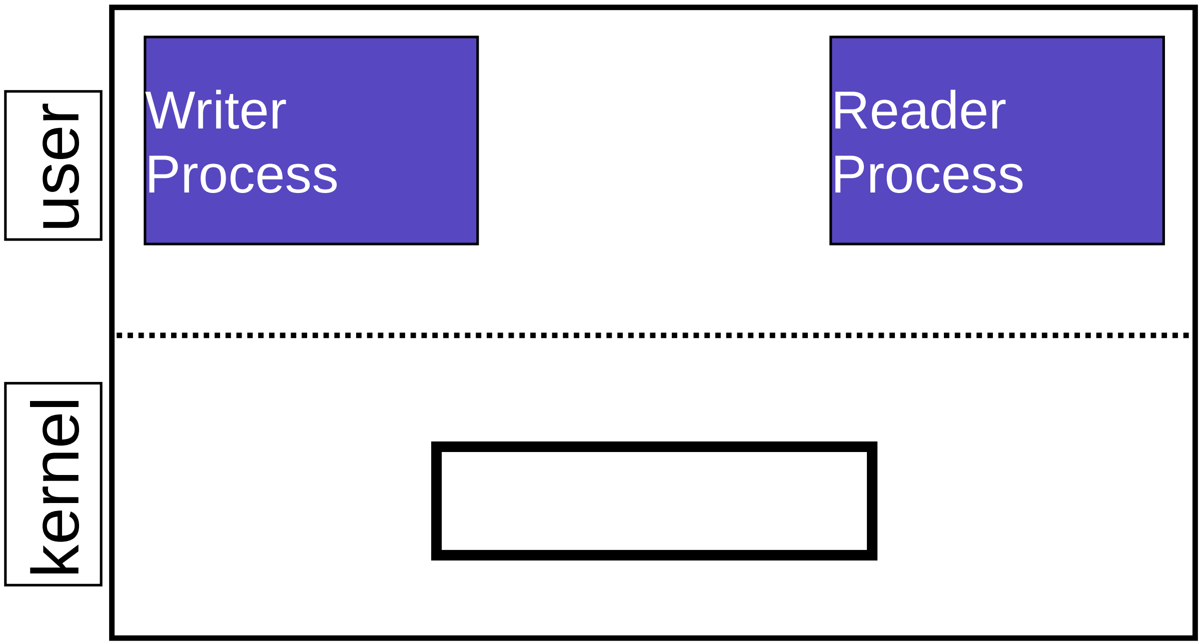
Challenges with Distributed System

Challenges with Distributed Systems?

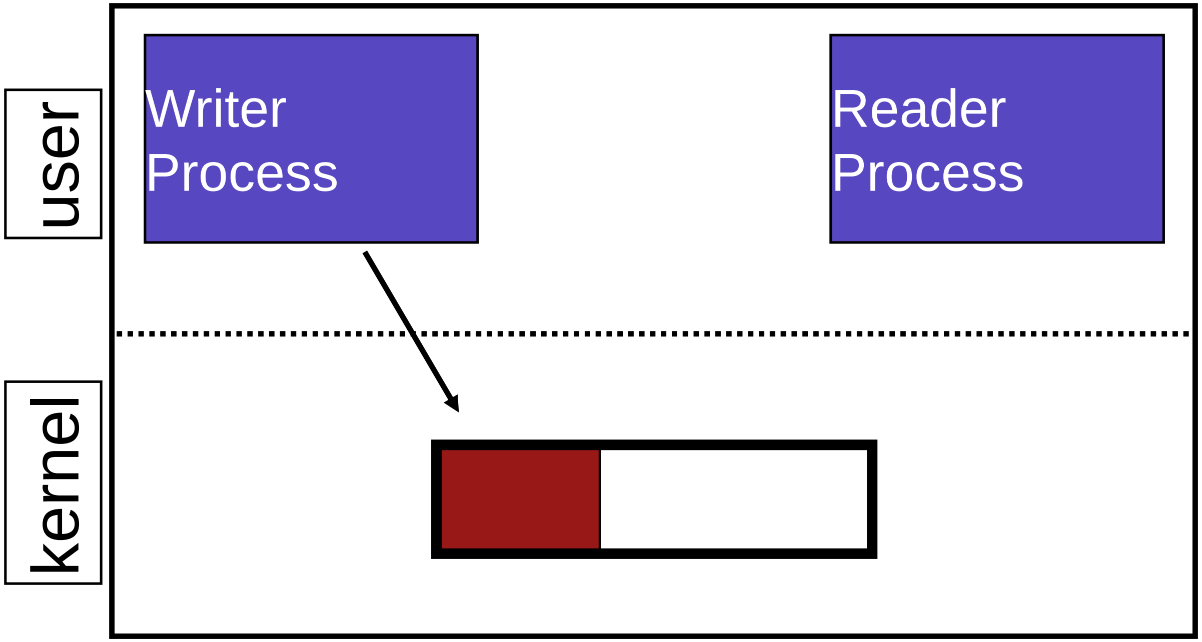
- **Global consistent system state**; network delays
- **System failure**: need to worry about partial failure
- **Communication failure**: links are unreliable
  - bit errors
  - packet loss
  - node/link failure
- **Motivation example:**

  Why are network sockets less reliable than Unix pipes (no network)?

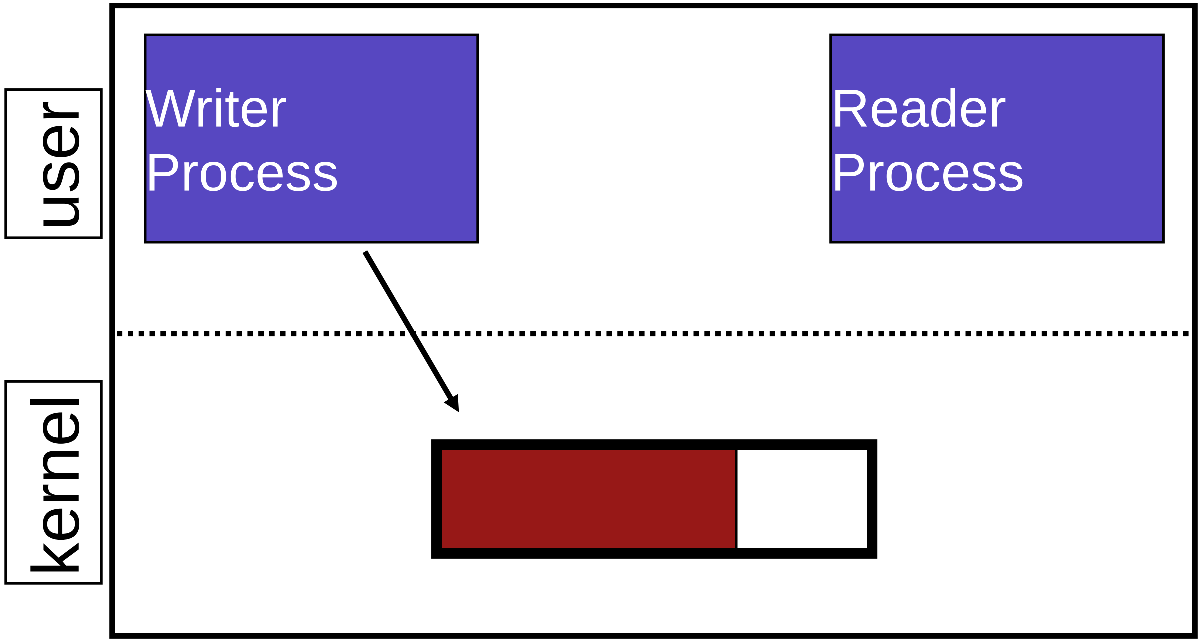
Pipe



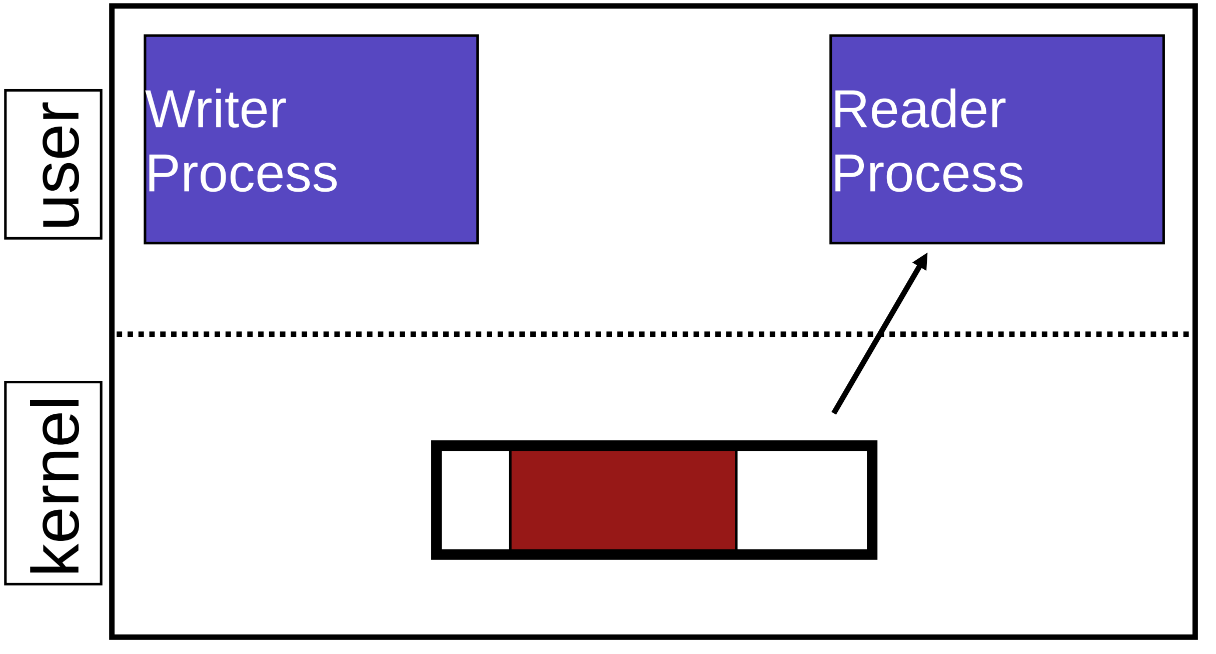
Pipe



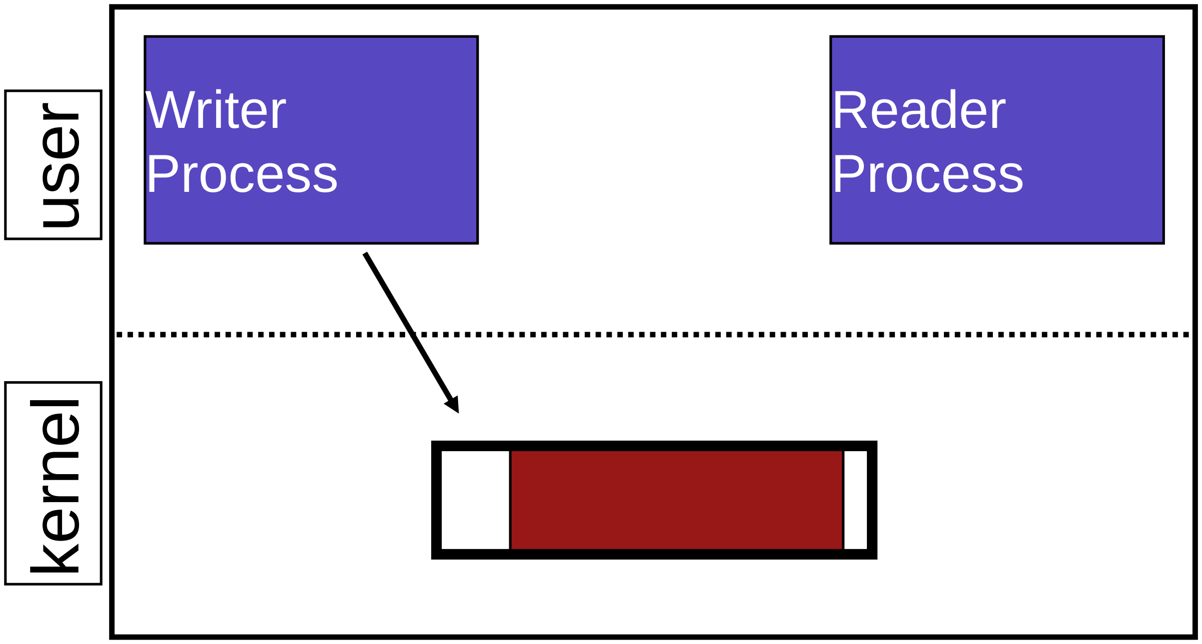
Pipe



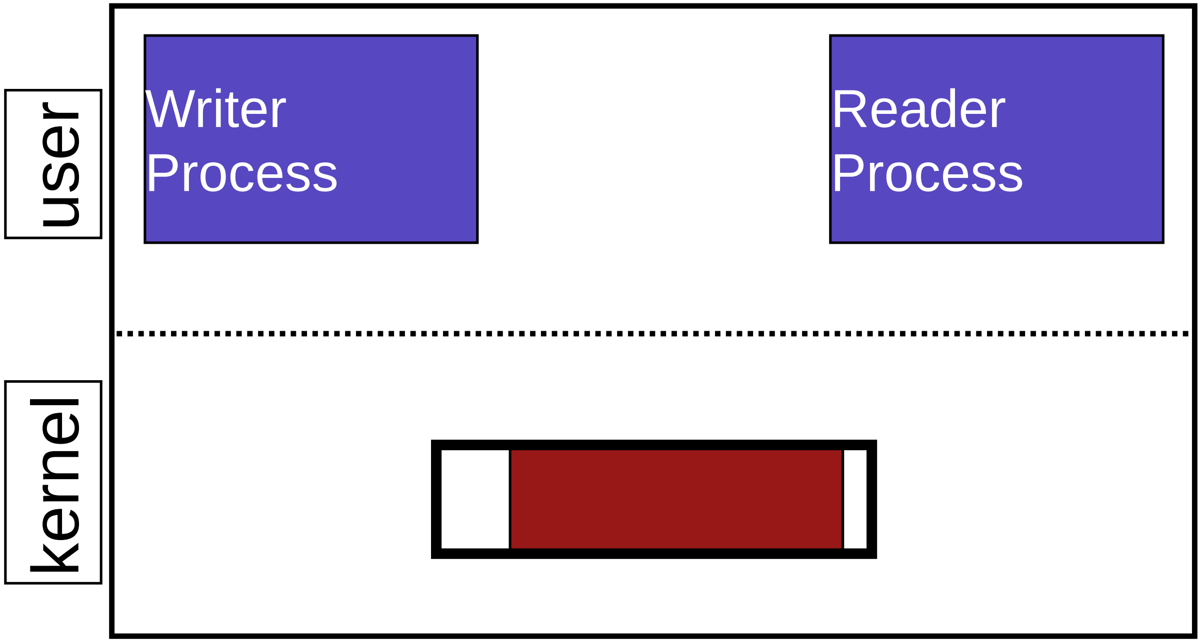
Pipe



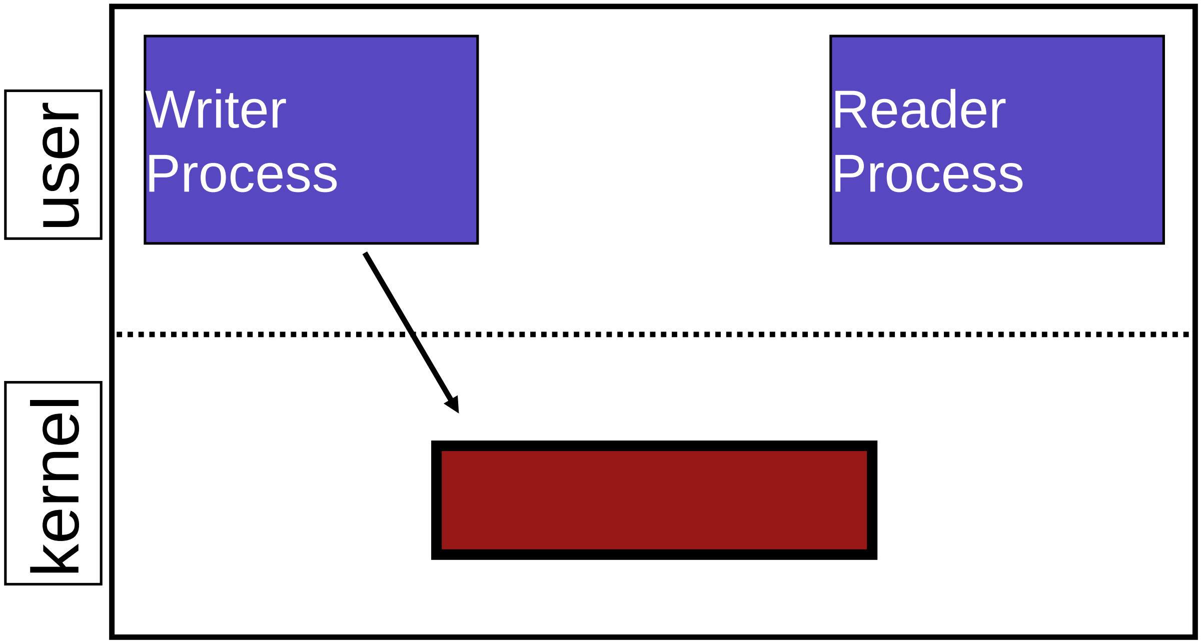
Pipe



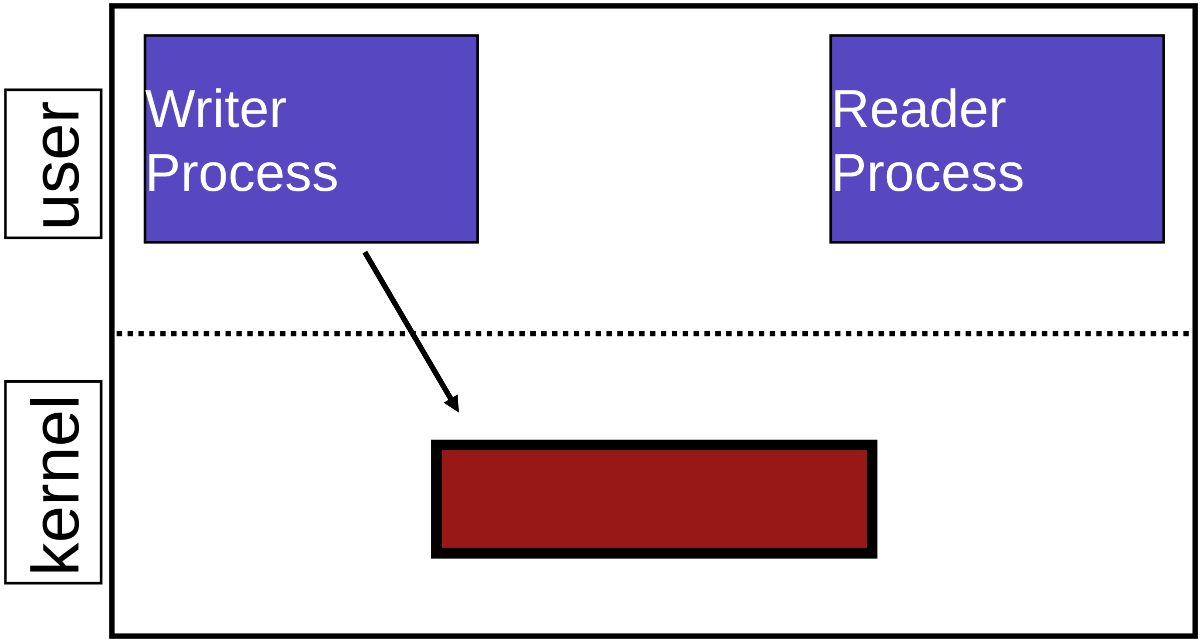
Pipe



Pipe

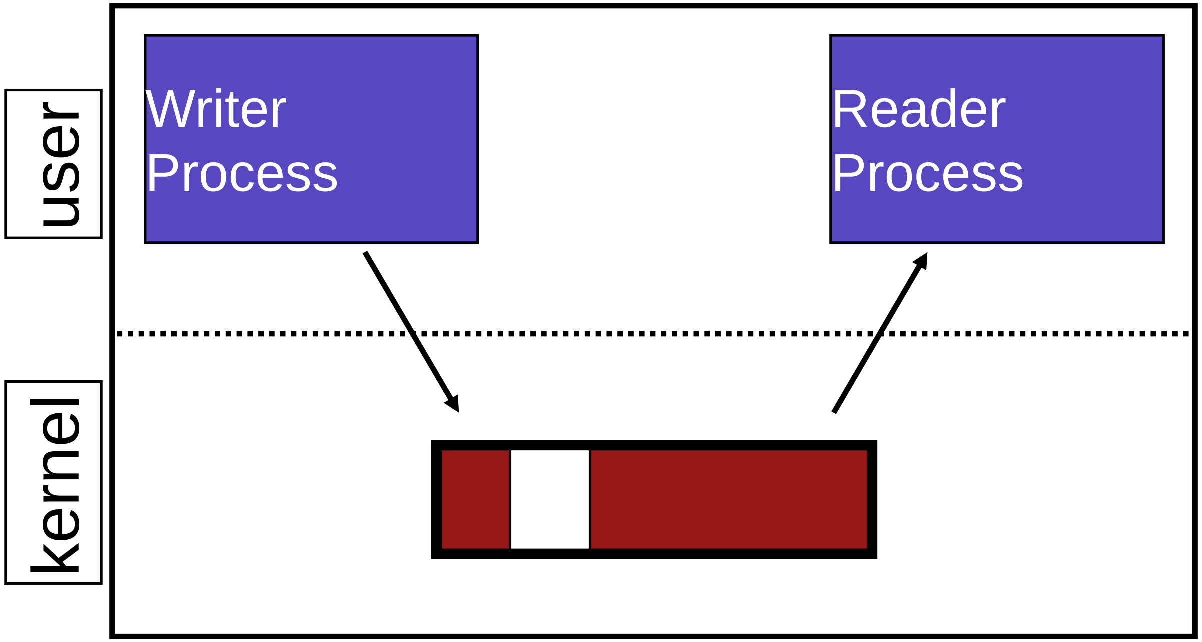


Pipe



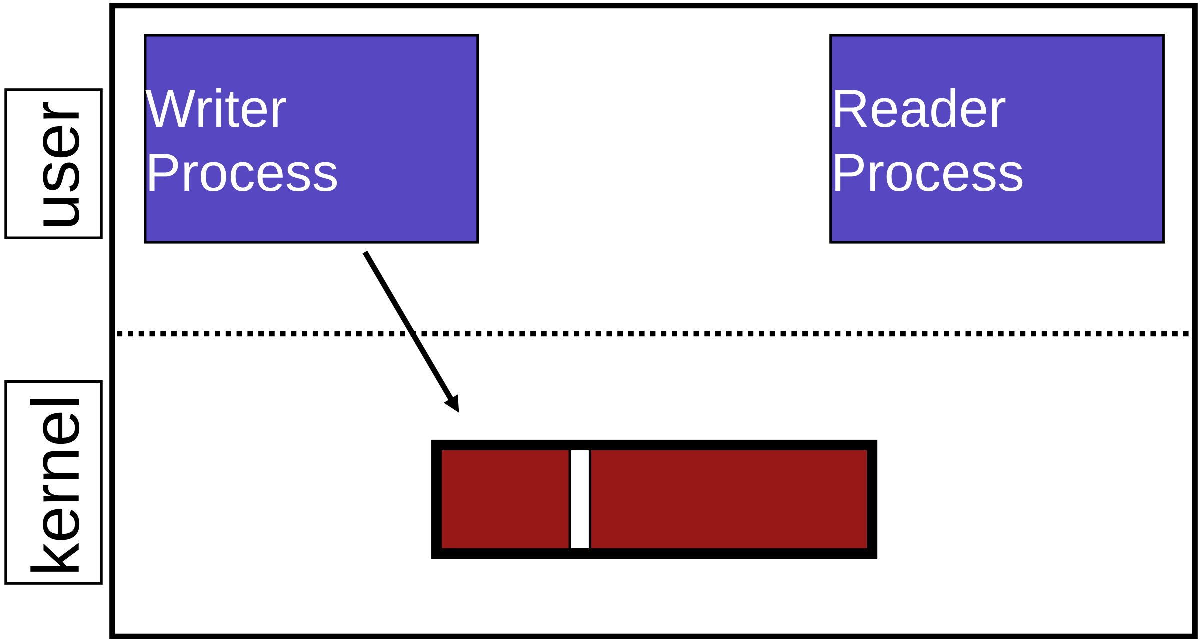
write waits for space

Pipe

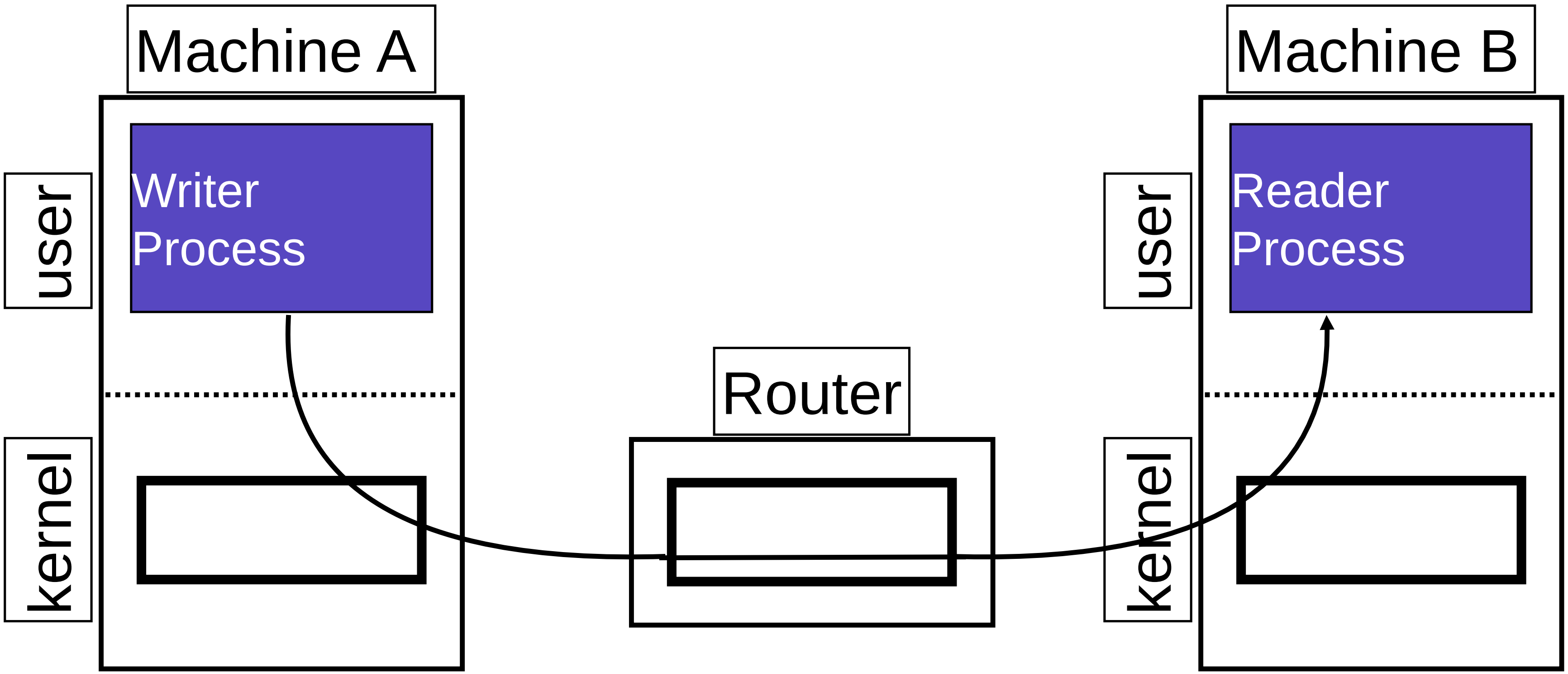


write waits for space

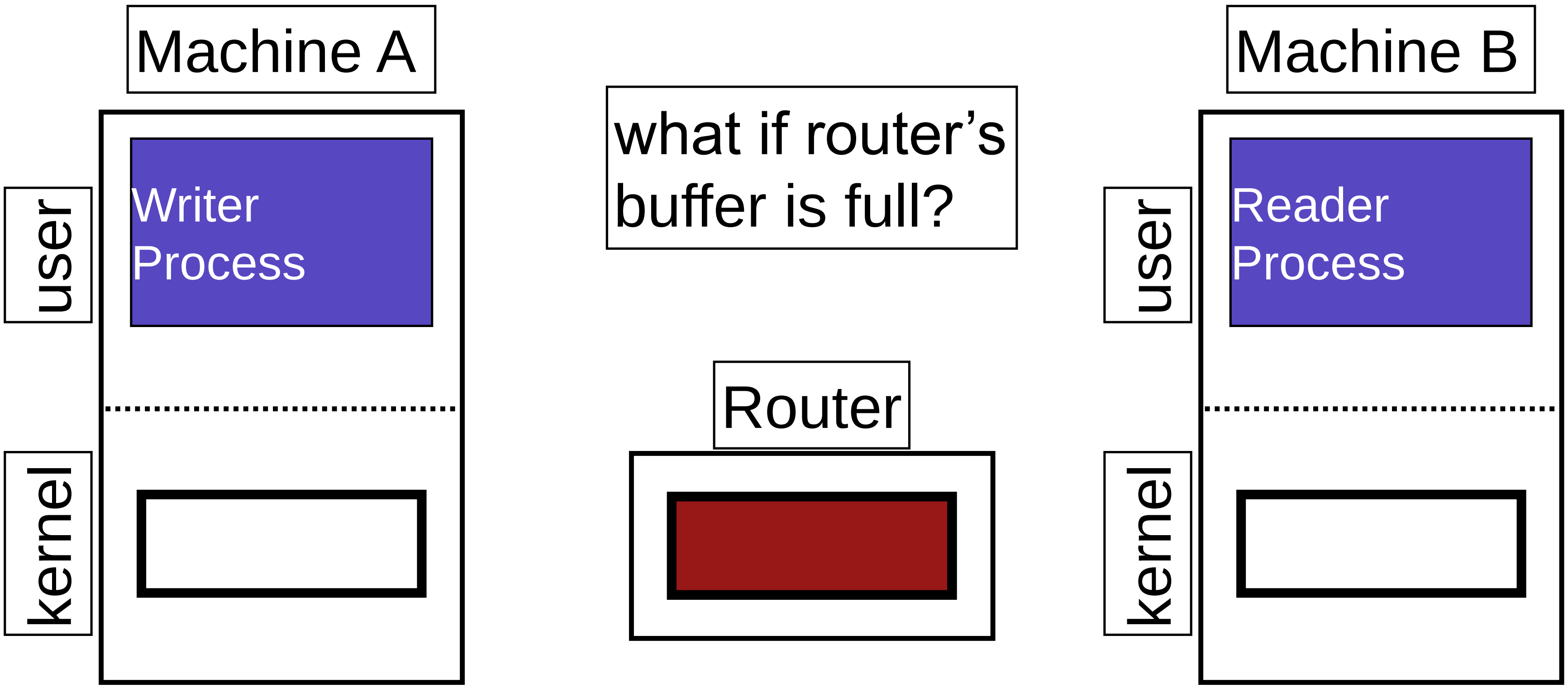
Pipe



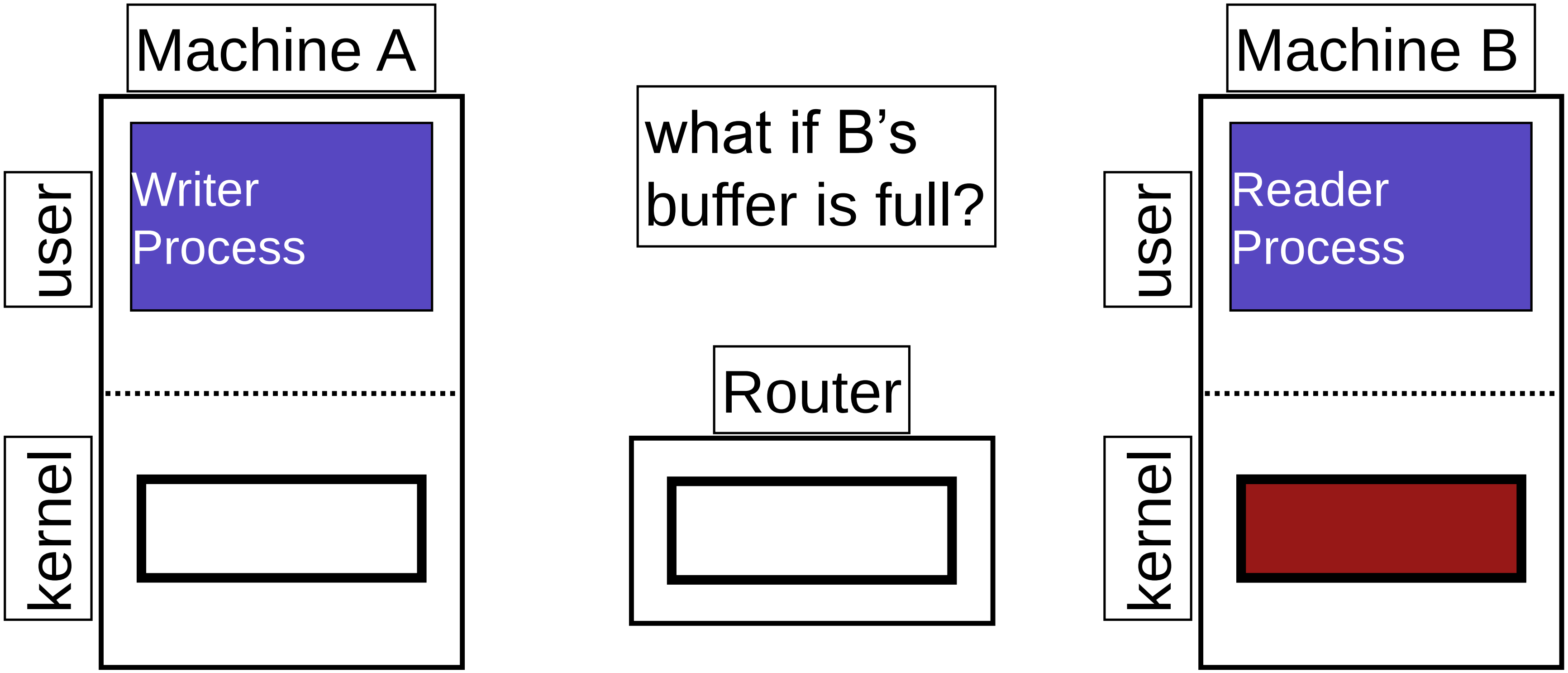
Network Socket



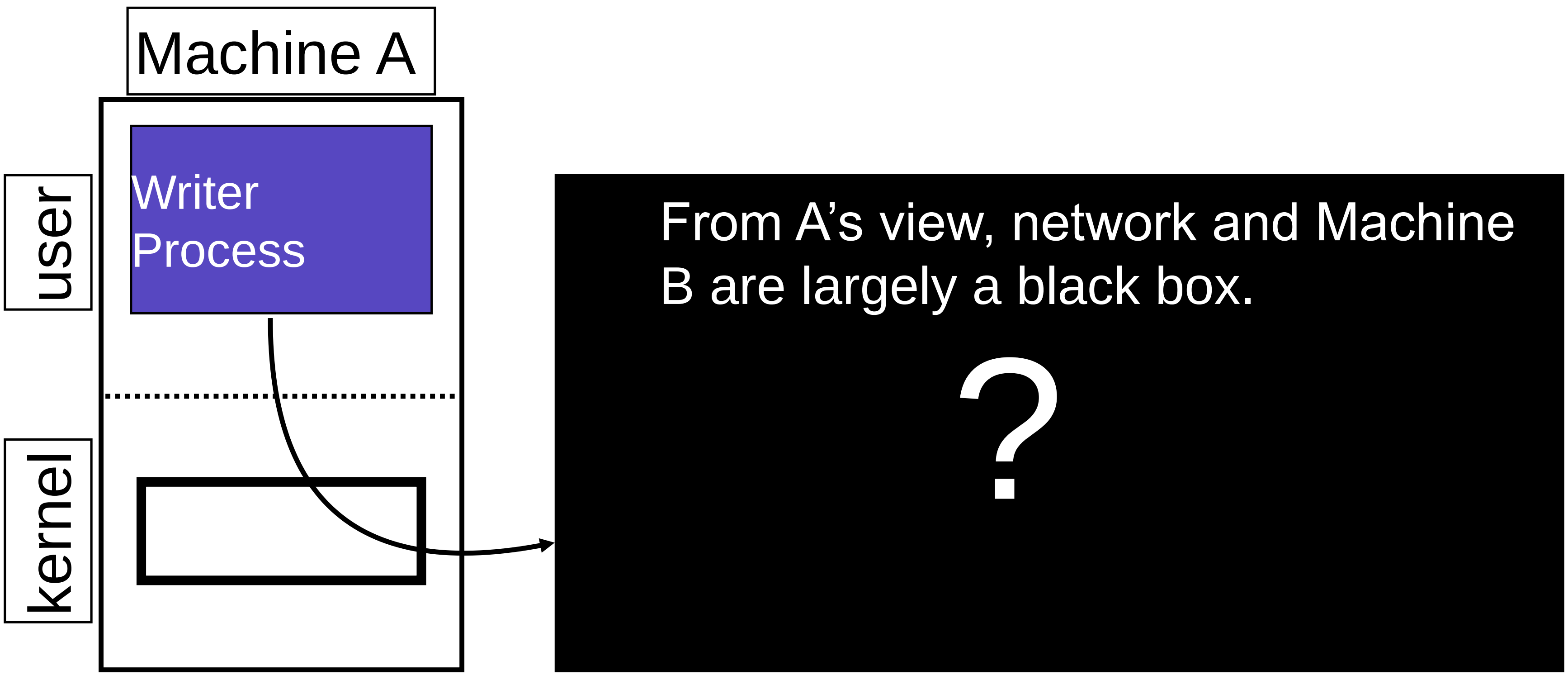
Network Socket



Network Socket



Network Socket



Communication Overview

- **Raw messages:** UDP
- **Reliable messages:** TCP
- **Remote procedure call:** RPC

Raw Message: UDP

UDP: User Datagram Protocol

API:

- reads and writes over socket file descriptors
- messages sent from/to ports to a target process on a machine

Provide minimal reliability features:

- messages may be lost
- messages may be **reordered**
- messages may be duplicated
- **only protection:** data integrity - checksums to ensure data not corrupted

Raw Message: UDP

Advantages:

- Lightweight
- Some applications make better reliability decisions themselves (e.g., video conferencing programs)

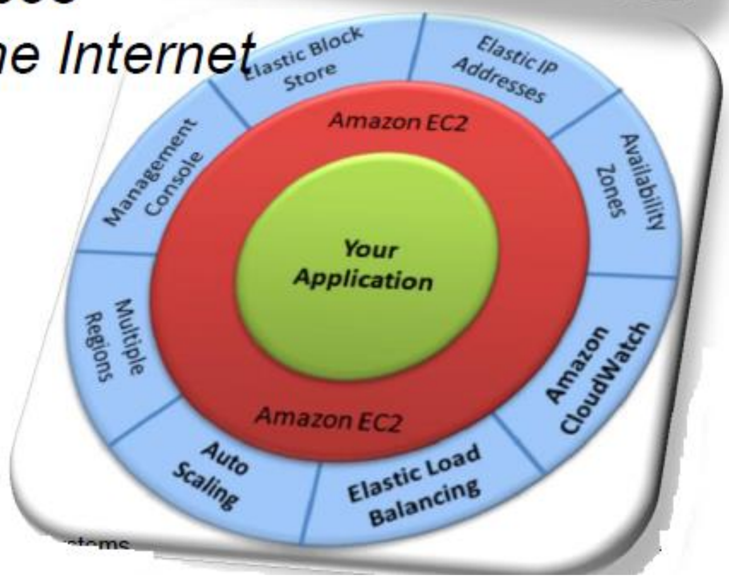
Disadvantages:

- Shift the reliability burden to the application. More difficult to write applications correctly

Why Cloud

- *A large-scale distributed computing paradigm driven by:*
  1. *economies of scale*
  2. *virtualization*
  3. *dynamically-scalable resources*
  4. *delivered on demand over the Internet*

Clouds ~ hosting

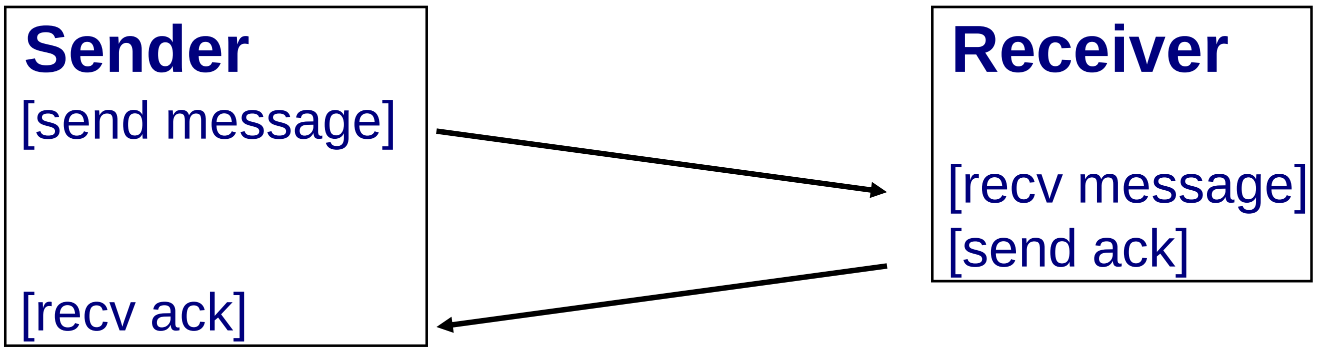


Build Reliable Messaging over Unreliable Layers

Reliable Messages: Layering Strategy

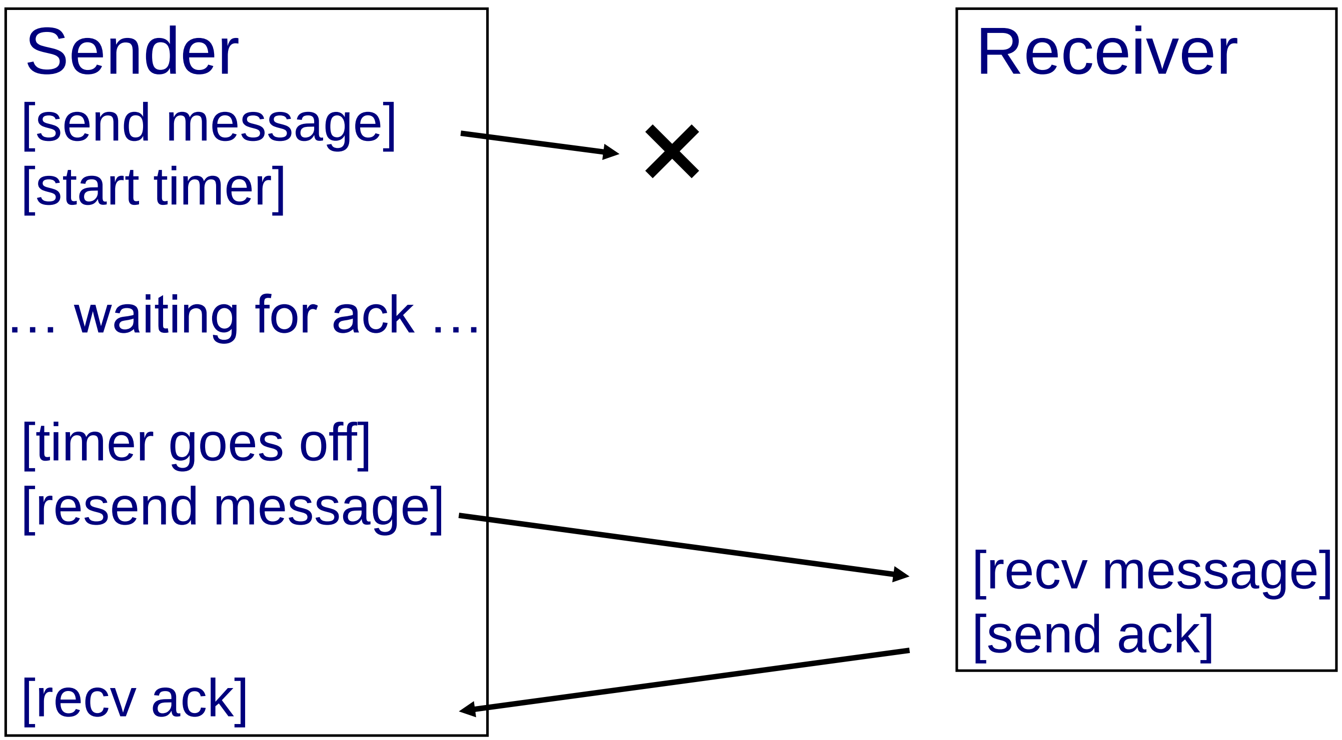
- **TCP:** Transmission Control Protocol
- **Using software**, build reliable, logical connections over unreliable hardware connections
- **Techniques:**
  - **Acknowledgment** (ACK)
  - **Timeout**
  - **Remember sent Messages**: having msg-id as part of the message header

Technique #1: ACK



Sender knows (ACK) message was received

Technique #2: TIMEOUT



Technique #2: TIMEOUT

- **How long to wait?** Timeout goes off, resend the message

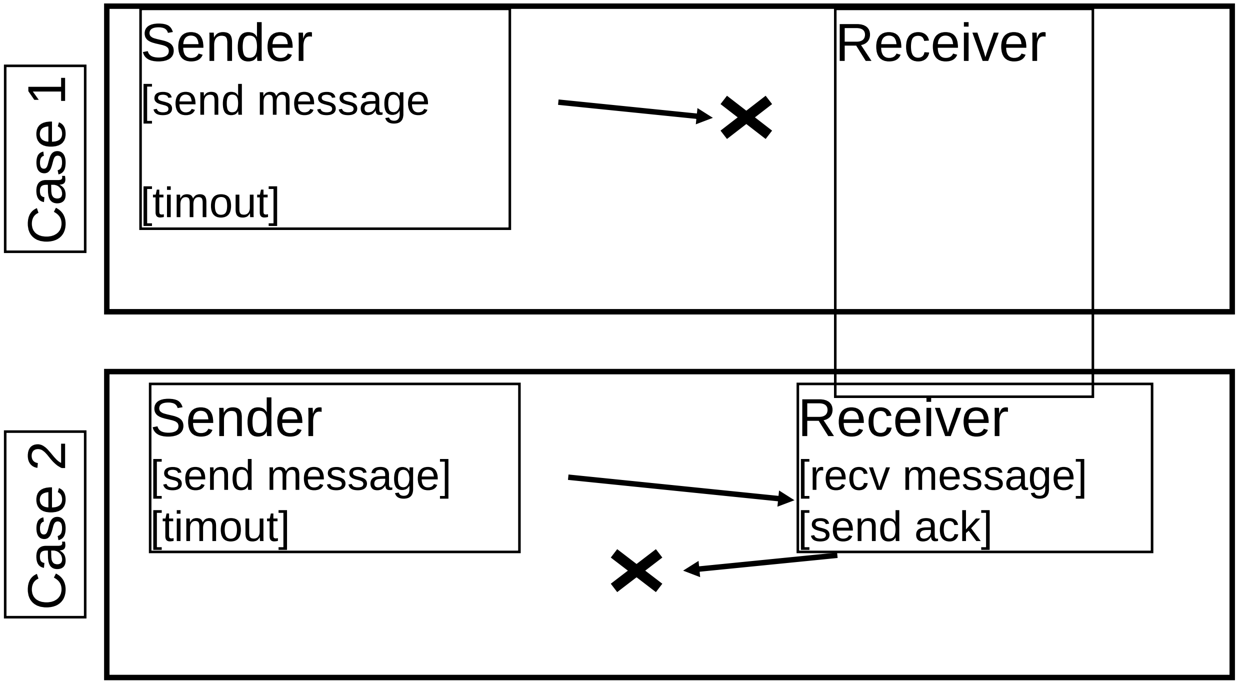
- **How long to wait: Too long?**
  - System feels unresponsive – poor r.p.t

- **How long to wait: Too short?**
  - Messages needlessly re-sent
  - Messages may have been dropped due to overloaded server. Resending makes overload worse!

Technique #2: TIMEOUT

- **One strategy:** be adaptive
- **Adjust timeout** based on how long acks usually take
- **For each missing ack**, wait longer between retries
- **What does a lost ack really mean**?

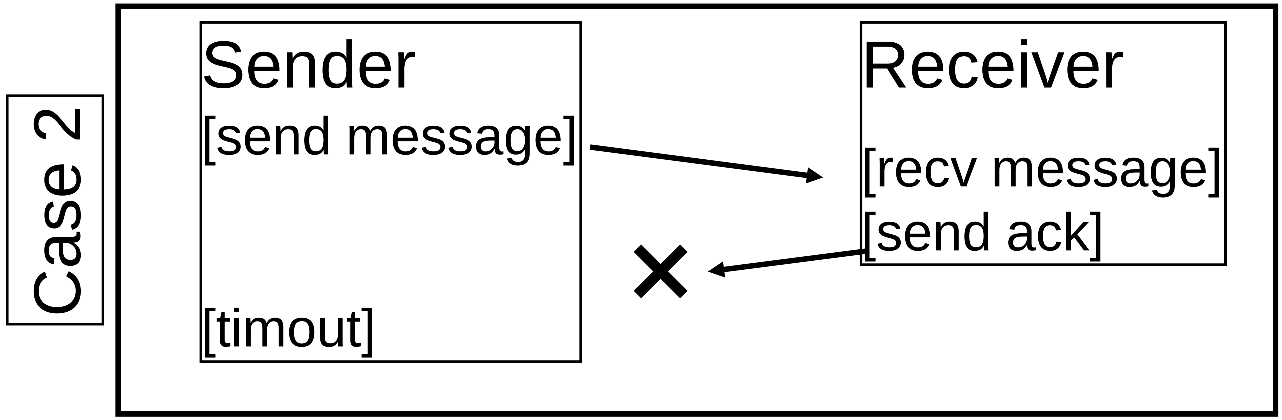
How do you Handle lost ACK



Lost ACK:
How can sender tell between these two cases?

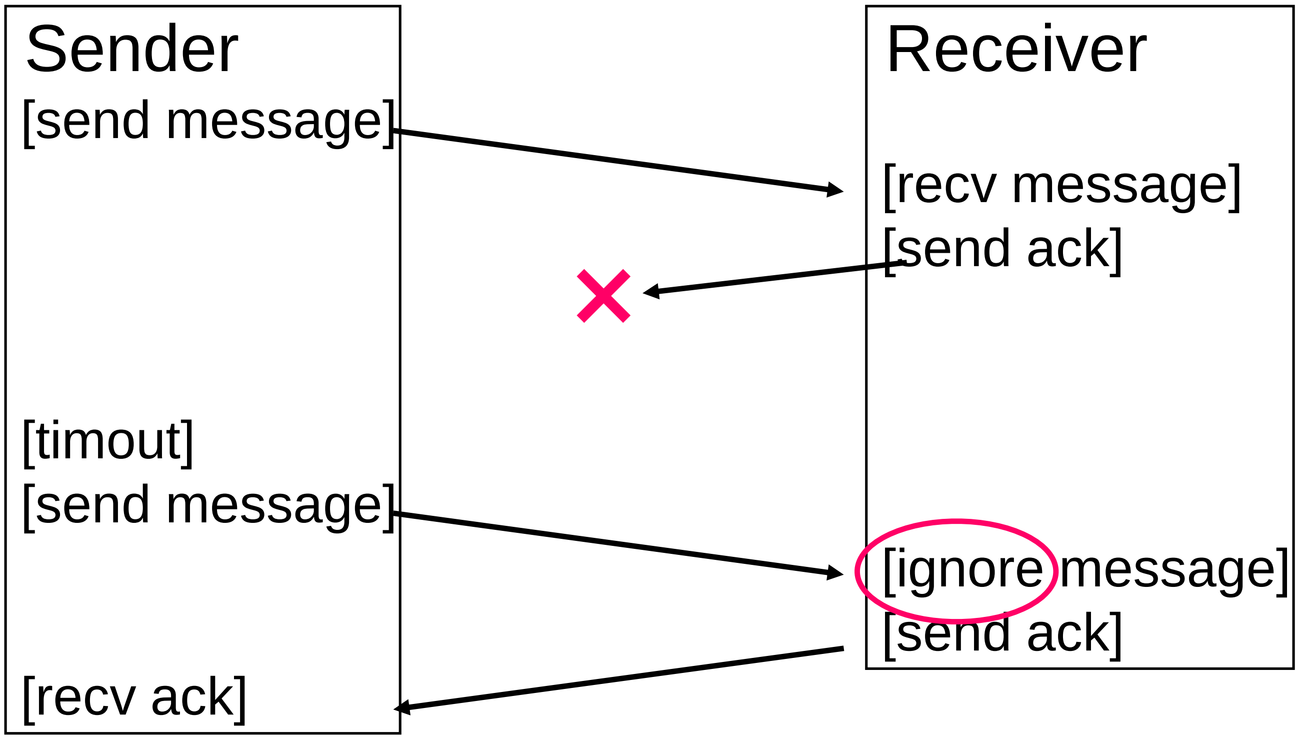
- **ACK**: message received exactly once
- **No ACK:** message may or may not have been received
- **What if message is a command to increment counter? non-idempotent…**

PROPOSED SOLUTION



- **Proposal:**
  Sender could send an **AckAck** msg to receiver so receiver knows whether to retry sending an Ack
- Sound good? No

Technique #3: REMEMBERS MESSAGES



Q) How does receiver know when to ignore a request?
Ans) Sequence number

Transmission Control Protocol: TCP

- Most popular protocol is based on seq-nums
- Receiver (server) Buffers (cache) response messages in the kernel to deliver to receiver/user in order
- Timeouts are adaptive

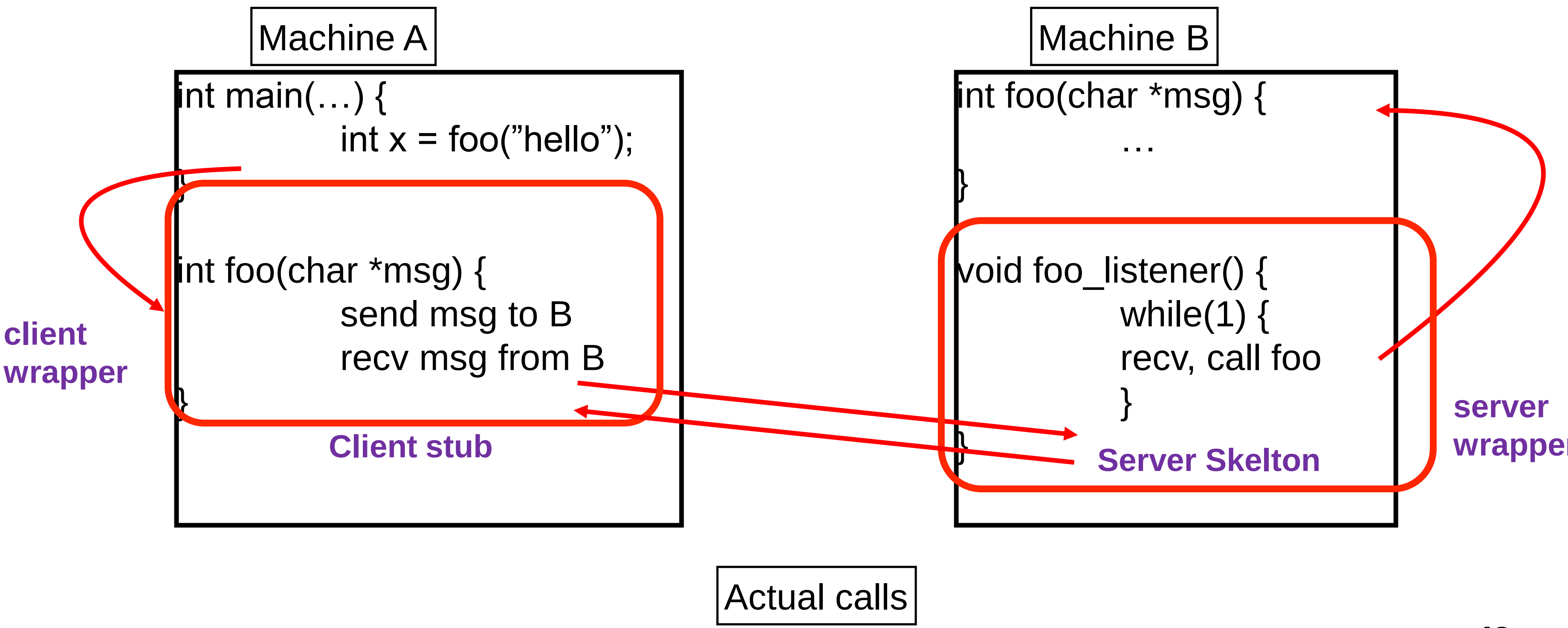
Communications Overview

- Raw messages: UDP
- Reliable messages: TCP
- Remote procedure call: RPC

What is an RPC

Remote Procedure Call: RPC

- **What could be easier** than calling a function?
- **Strategy**: create wrappers so calling a function on another machine feels just like calling a local function
- Very common abstraction



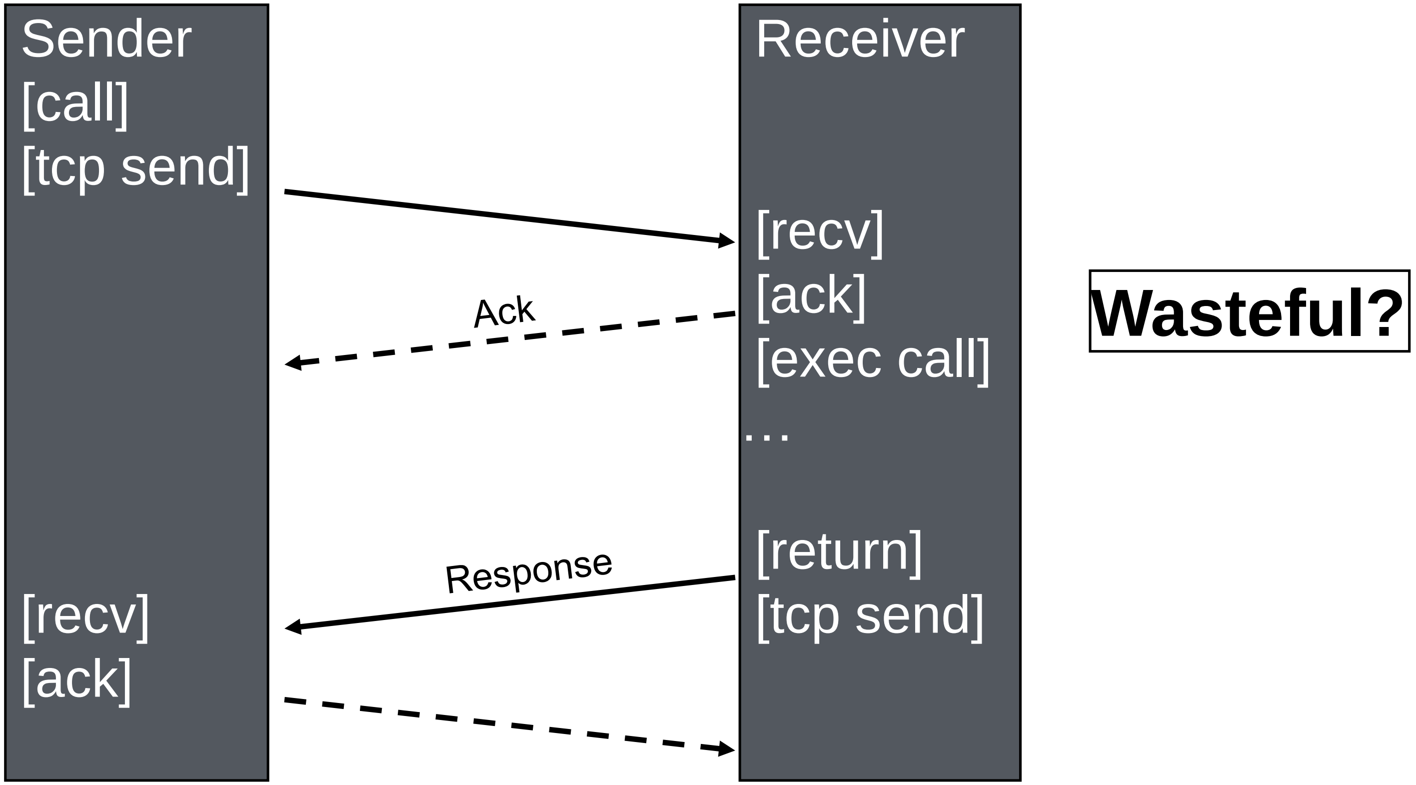
RPC Tools

- RPC packages help with two components:
  1. **Runtime library**
     - Thread pool
     - Socket listeners call functions on server (worker threads)

  2. **Stub generation / compiler**
     - Create wrappers automatically
     - Many tools available (rpcgen, thrift, protobufs)*

* rpcgen: https://en.wikipedia.org/wiki/RPCGEN
Thrift (Apache RPC): https://thrift.apache.org/
Protobufs (lang independent for serializing data structures):
https://en.wikipedia.org/wiki/Protocol_Buffers
https://developers.google.com/protocol-buffers/

RPC Over TCP



- Use UDP and piggybacking the ACK to be part of the returned server message back to the client; unless serving the sender request takes long time.
- If the server function takes log time then send separate ACK.

Distributed File System

NFS Architecture

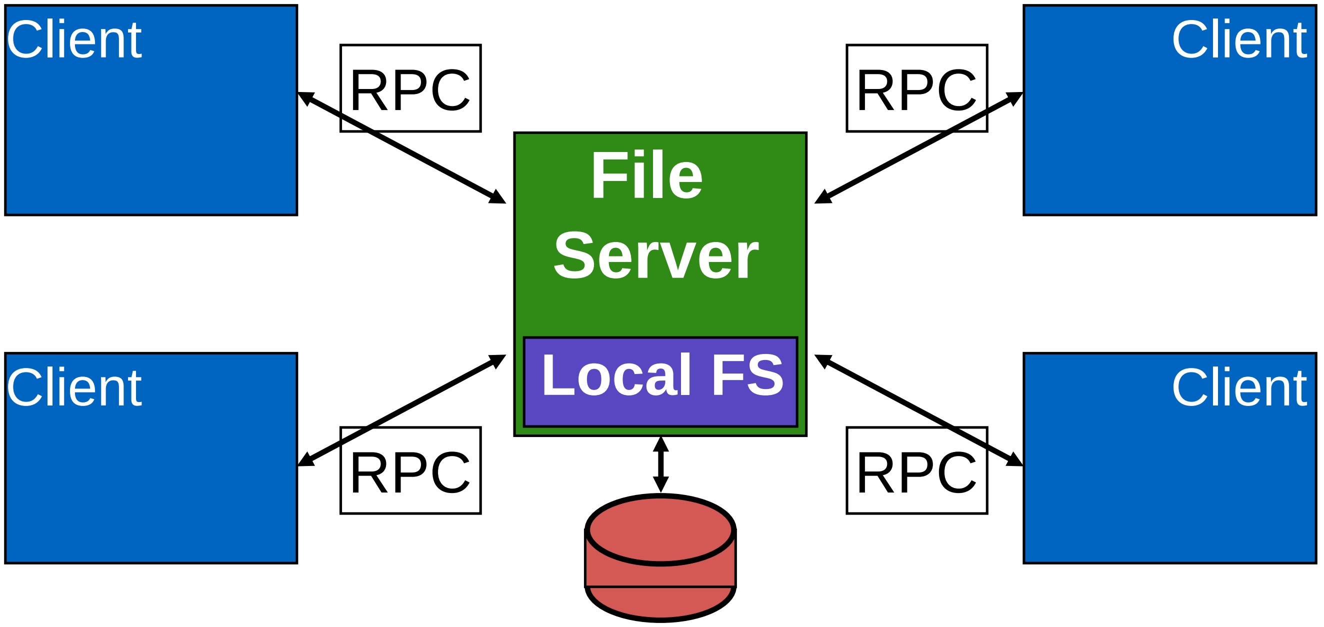
Distributed File Systems

- **File systems** are great use case for distributed systems
- **Local FS**:
  processes on same machine access shared files
- **Network FS**:
  processes on different machines access shared files in the same way
- **Network File System Goals:**
  - **Fast and simple** crash recovery
  - **Transparent access;** normal UNIX semantics
  - **Reasonable performance**

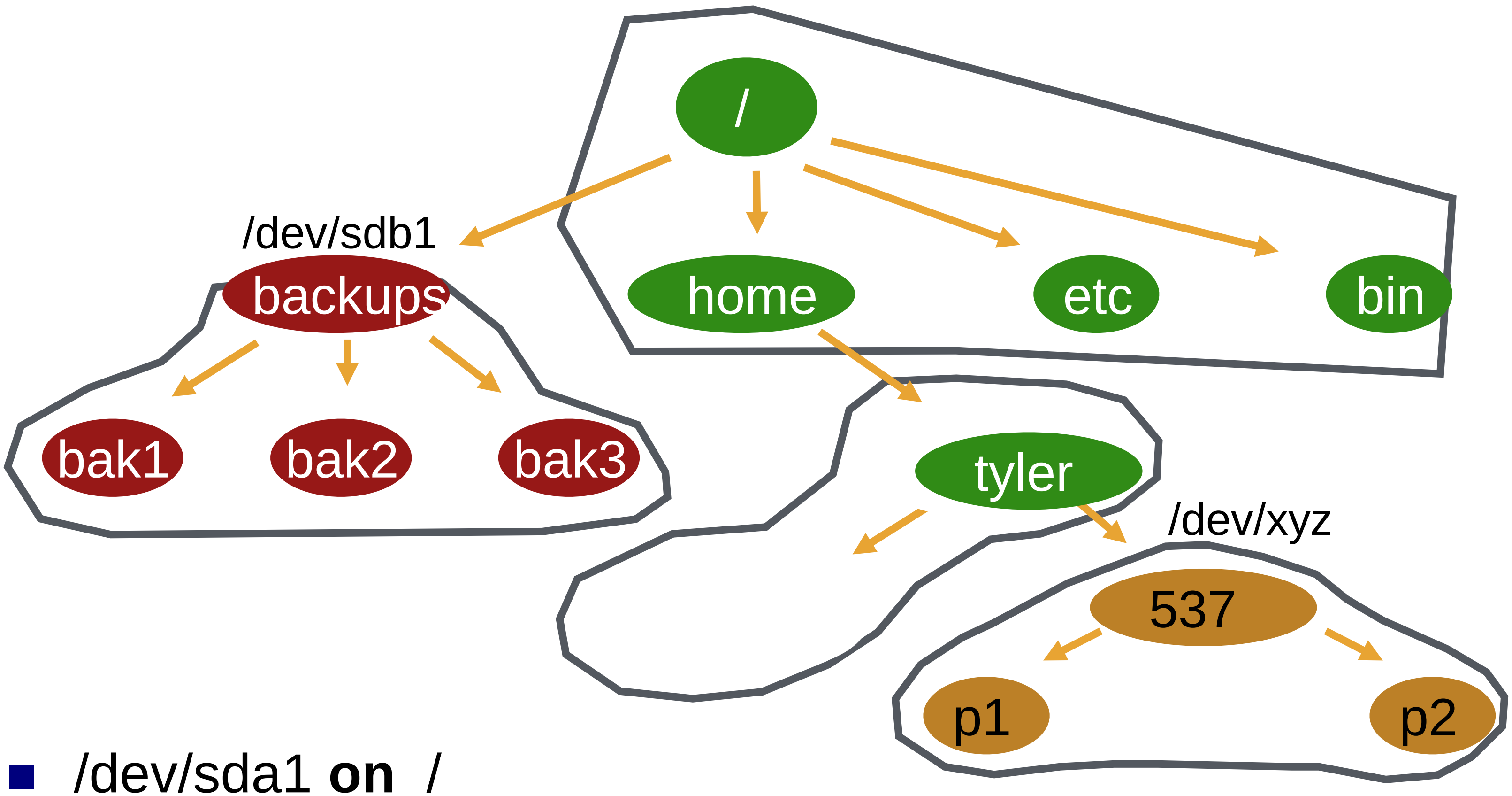
Overview

- **NFS Architecture**
- **Network API:** use RPC
- **Write Buffering** (delayed write to disk)
- **Cache**

NFS Architecture

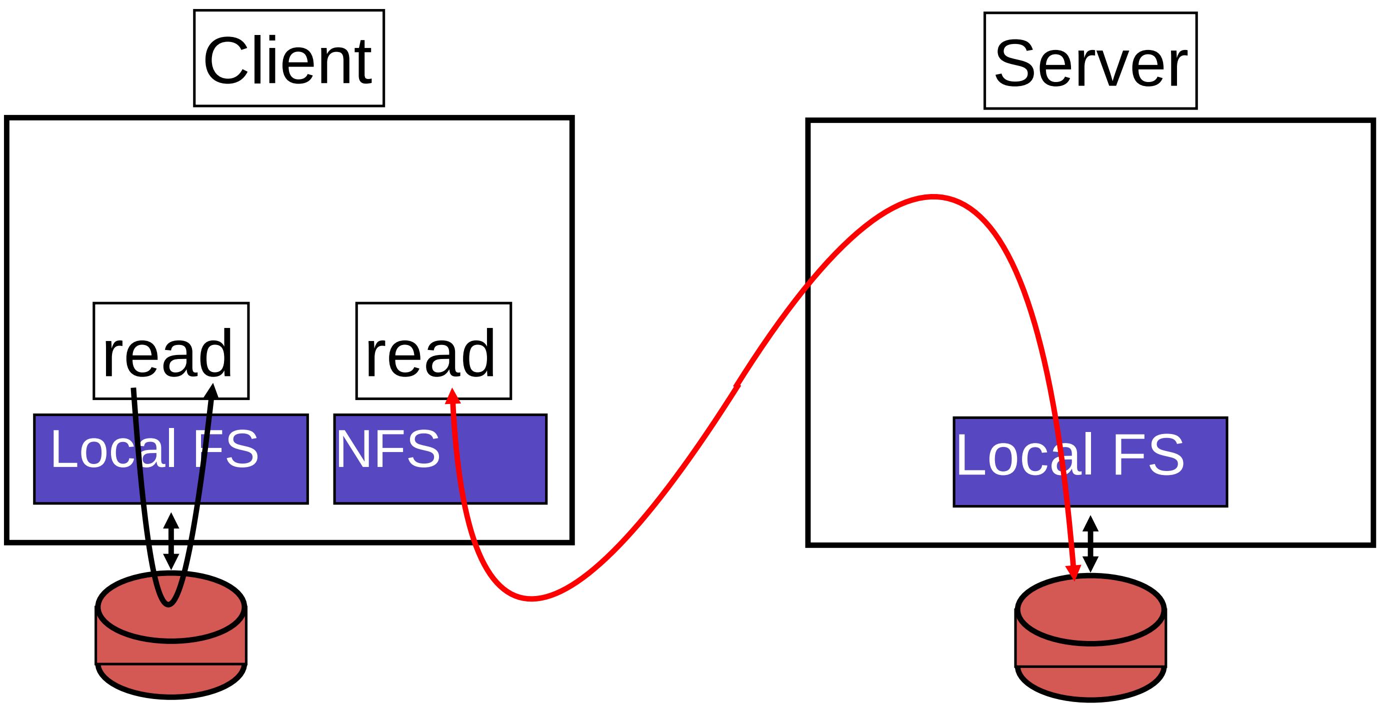


Mount

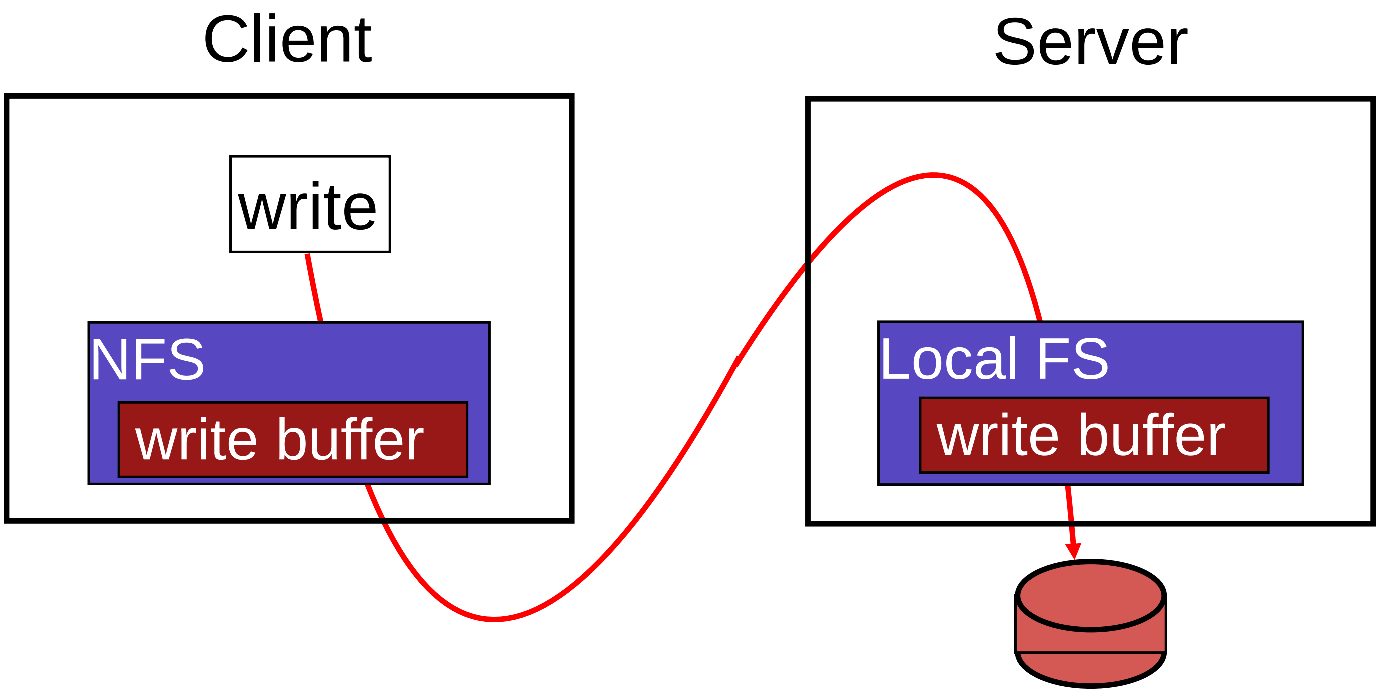


- /dev/sda1 **on** /
- /dev/sdb1 **on** /backups
- /dev/xyz **on** /home/tyler

General Strategy: Export FS

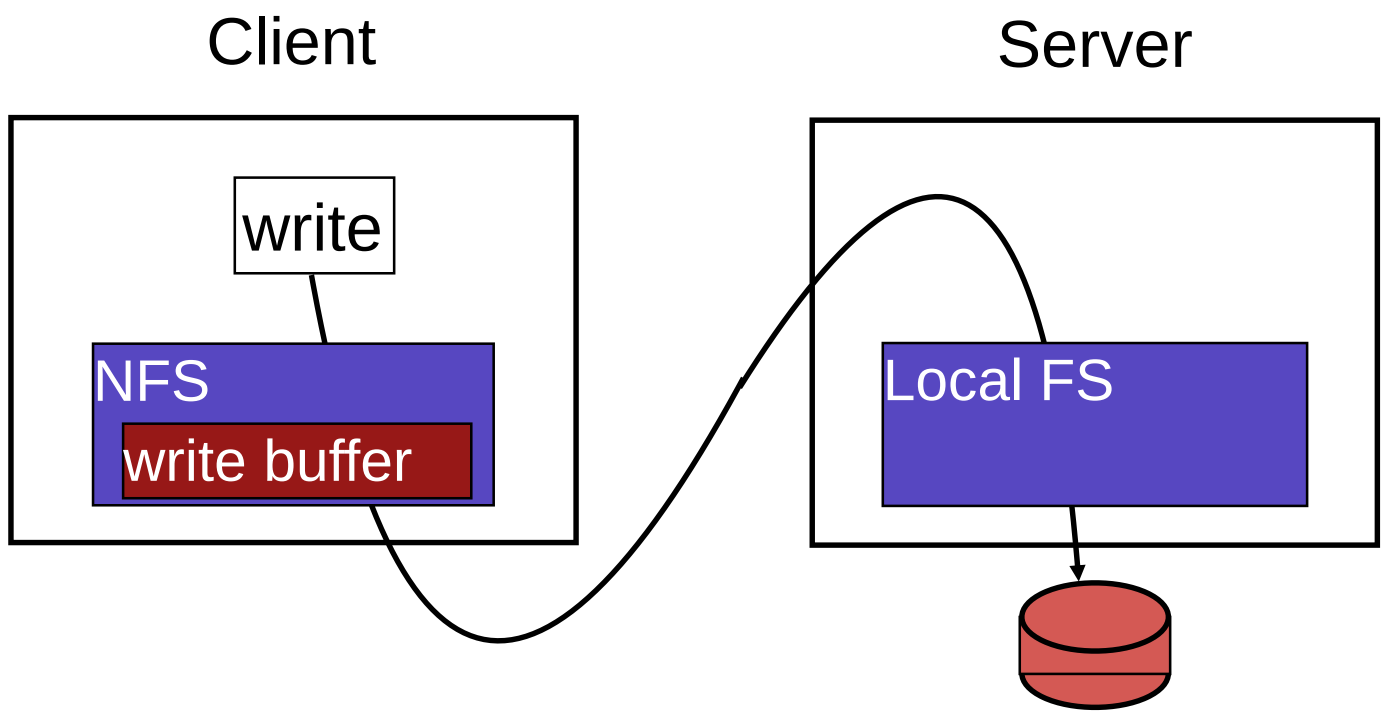


Write Buffers (delayed write)



- Server acknowledges write once data in the server cache but before the write is pushed to disk
- What happens if server crashes before writing to disk?

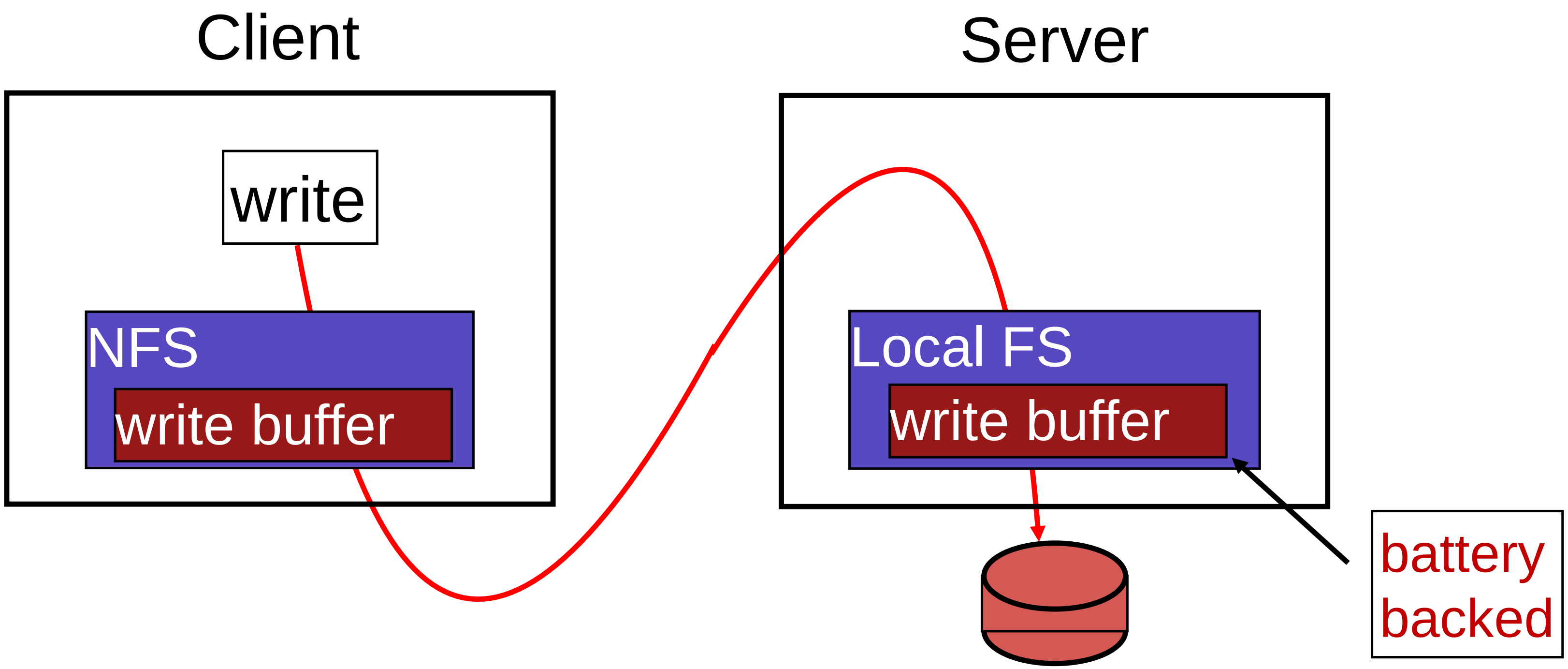
Write Buffers



1. Don’t use server write buffer
   (persist data to disk before acknowledging write)

Problem: Slow!

Write Buffers



2. use persistent write buffer (more $$ expensive)

Client Caching

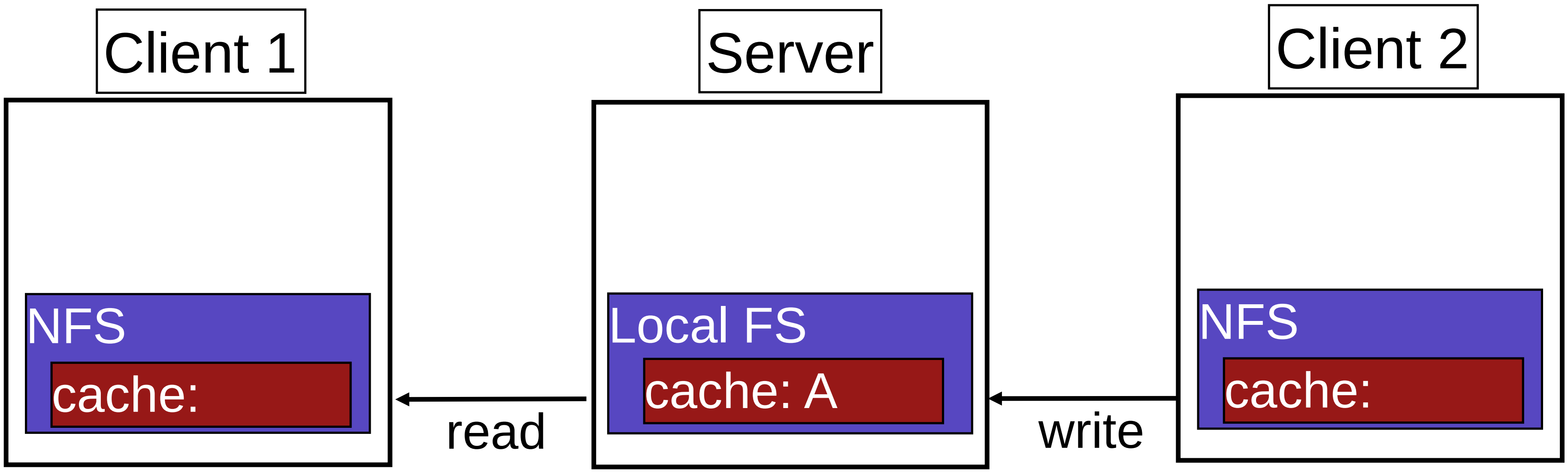
Cache Consistency

NFS can cache data in three places:

- Server memory
- Client memory
- Client disk

How to make sure all versions are in sync?

Distributed Cache



Cache consistency is expensive

One possible solution is to invalidate cache entries after a given time; it means that you minimize the chances for reading out-of-date data!

END