Remote Procedure Call

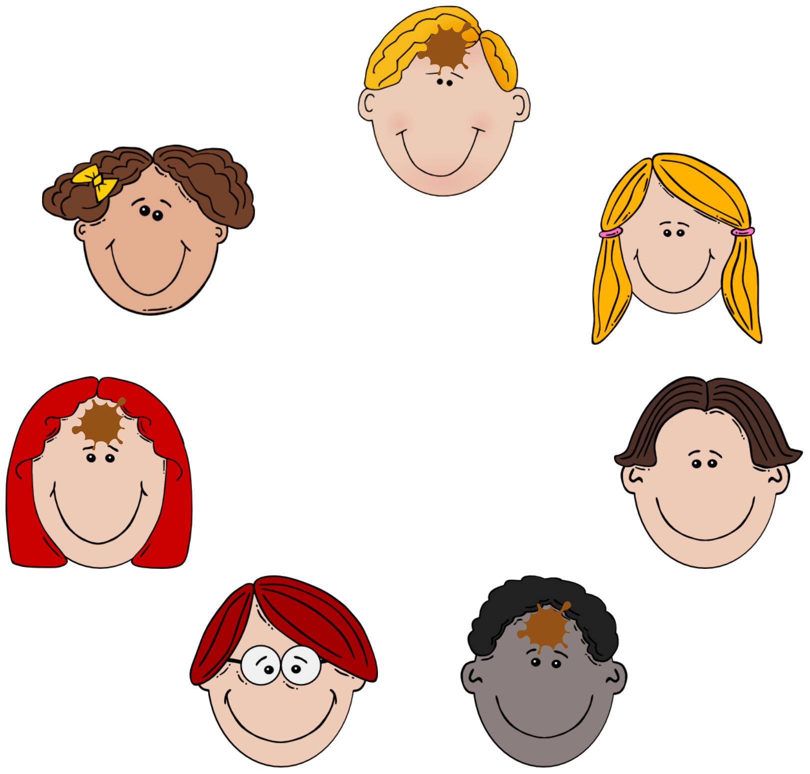
Arvind Krishnamurthy

Course Logistics

- Everyone should have a gitlab account
  - Let us know if you don’t have one
- Make sure you have signed up for Piazza
- Lab 1 due next Thursday
  - Submission through Canvas
- Blog post for Friday’s reading
  - Submission through Canvas

Muddy Foreheads

- n children, k get mud on their foreheads
- Children sit in circle.
- Teacher announces, "Someone has mud on their forehead."
  - Someone == 1 or more
  - No on can see their own forehead
  - k is not "common knowledge"

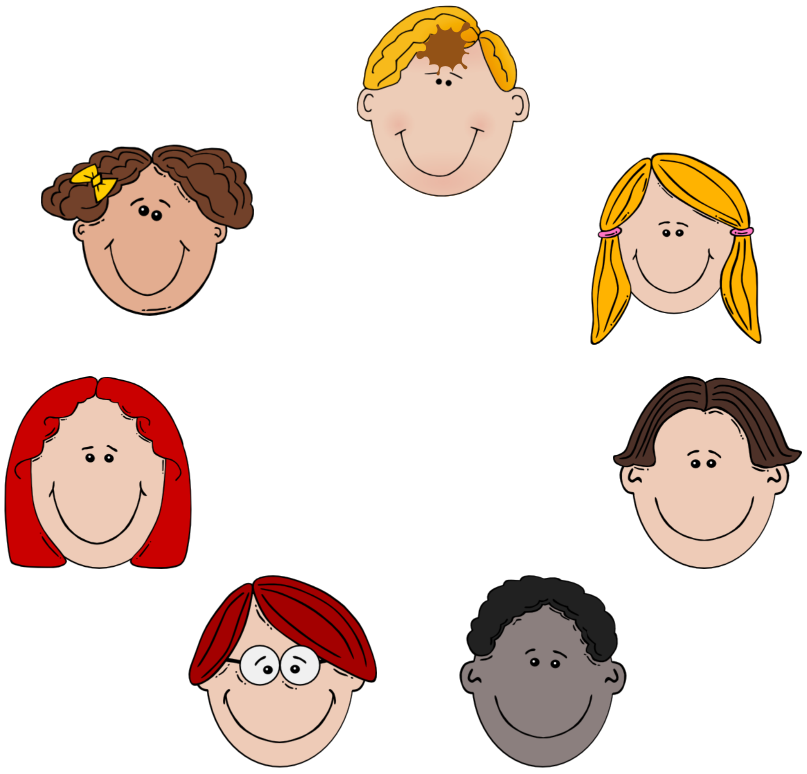


Muddy Foreheads

- n children, k get mud on their foreheads
- Children sit in circle.
- Teacher announces, "Someone has mud on their forehead."
- Teacher repeatedly asks, "Raise your hand if you know you have mud on your forehead."
- What happens?

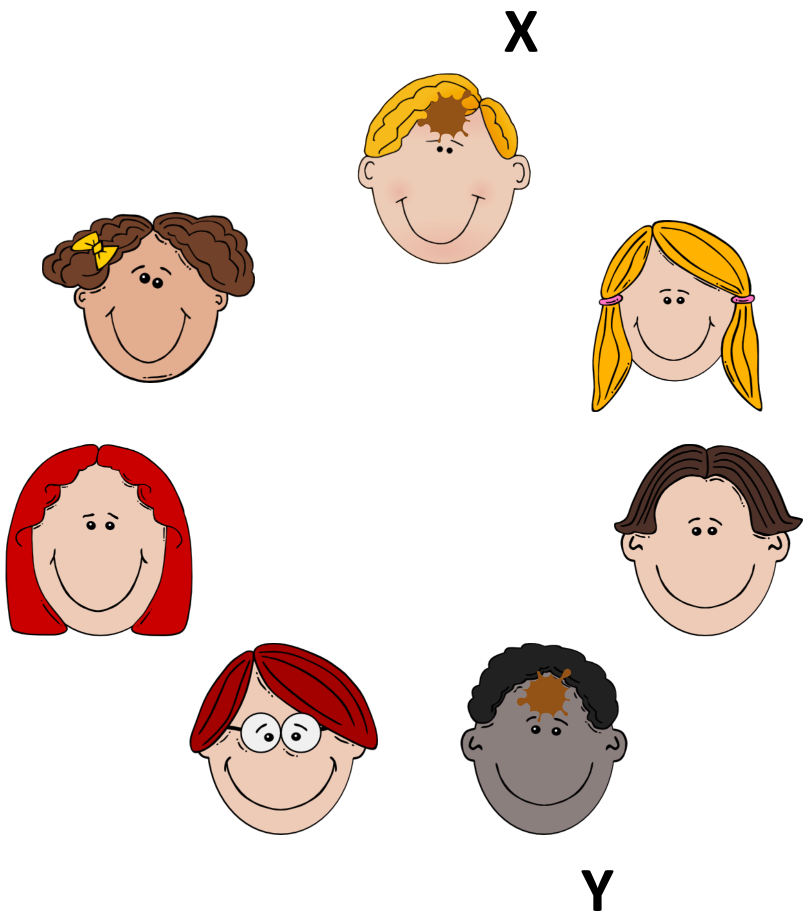
Muddy Foreheads

- n children, k get mud on their foreheads
- Children sit in circle.
- Teacher announces, "Someone has mud on their forehead."
- Teacher repeatedly asks, "Raise your hand if you know you have mud on your forehead."
- What happens?



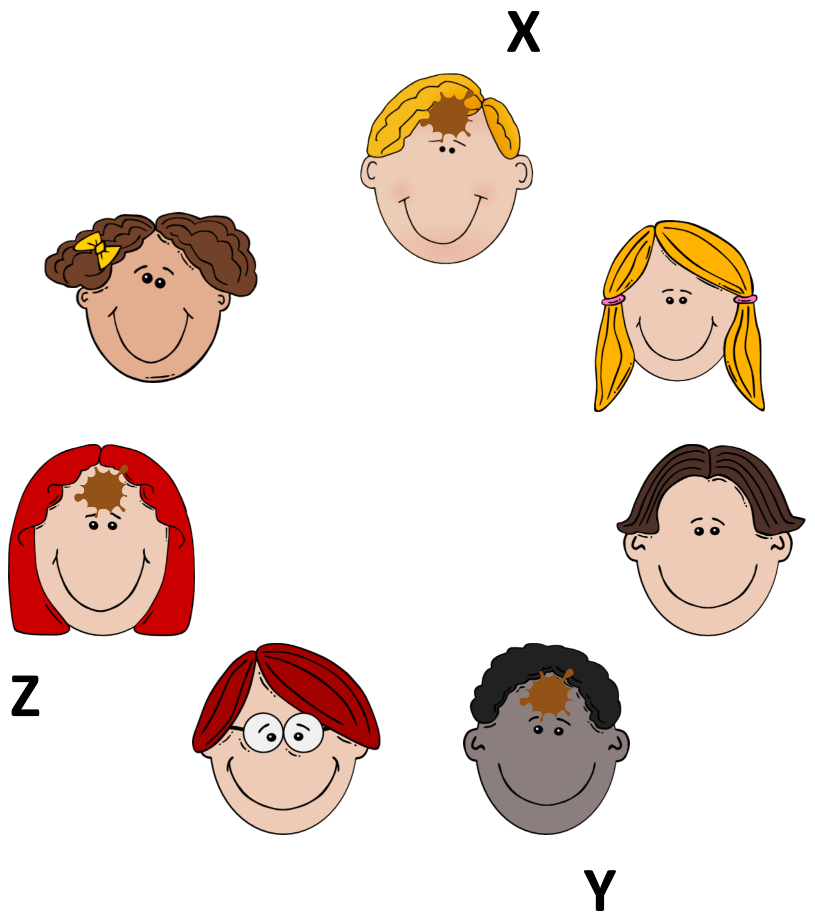
Muddy Foreheads

- n children, k get mud on their foreheads
- Children sit in circle.
- Teacher announces, "Someone has mud on their forehead."
- Teacher repeatedly asks, "Raise your hand if you know you have mud on your forehead."
- What happens?



Muddy Foreheads

- n children, k get mud on their foreheads
- Children sit in circle.
- Teacher announces, "Someone has mud on their forehead."
- Teacher repeatedly asks, "Raise your hand if you know you have mud on your forehead."
- What happens?



Muddy Foreheads (contd.)

- Claim:
  - The first k-1 times the teacher asks, all children will reply “No”
  - The k-th time all dirty children will reply “Yes”
- Reasoning by considering cases and using induction:
  - k=1: the child with a muddy forehead will say yes
  - k=2: let X and Y have muddy foreheads
    - Each sees exactly one other person with muddy forehead
    - In round 1, X noticed Y didn’t say “Yes”
      - Possible only because Y must have seen a child with a muddy forehead ==> X must have mud

The Muddy Forehead "Paradox"

If $k > 1$, the teacher didn't say anything anyone didn't already know!

Why Are Distributed Systems Hard?

- Asynchrony
  - Different nodes run at different speeds
  - Messages can be unpredictably, arbitrarily delayed
- Failures (partial and ambiguous)
  - Parts of the system can crash
  - Can’t tell crash from slowness
- Concurrency and consistency
  - Replicated state, cached on multiple nodes
  - How to keep many copies of data consistent?

Why Are Distributed Systems Hard?

- Performance
  - Have to efficiently coordinate many machines
  - Performance is variable and unpredictable
  - Tail latency: only as fast as slowest machine
- Testing and verification
  - Almost impossible to test all failure cases
  - Proofs (emerging field) are really hard
- Security
  - Need to assume adversarial nodes

MapReduce Computational Model

For each key k with value v, compute a new set of key-value pairs:

map (k,v) → list(k’,v’)

For each key k’ and list of values v’, compute a new (hopefully smaller) list of values:

reduce (k’,list(v’)) → list(v’’)

User writes map and reduce functions.

Framework takes care of parallelism, distribution, and fault tolerance.

MapReduce (or ML or …) Architecture

- Scheduler accepts MapReduce jobs
  - finds a MapReduce master and set of avail workers
- For each job, MapReduce master <array>
  - farms tasks to workers; restarts failed jobs; syncs task completion
- Worker <array>
  - executes Map and Reduce tasks
- Storage <array>
  - stores initial data set, intermediate files, end results

Remote Procedure Call (RPC)

A request from the client to execute a function on the server.

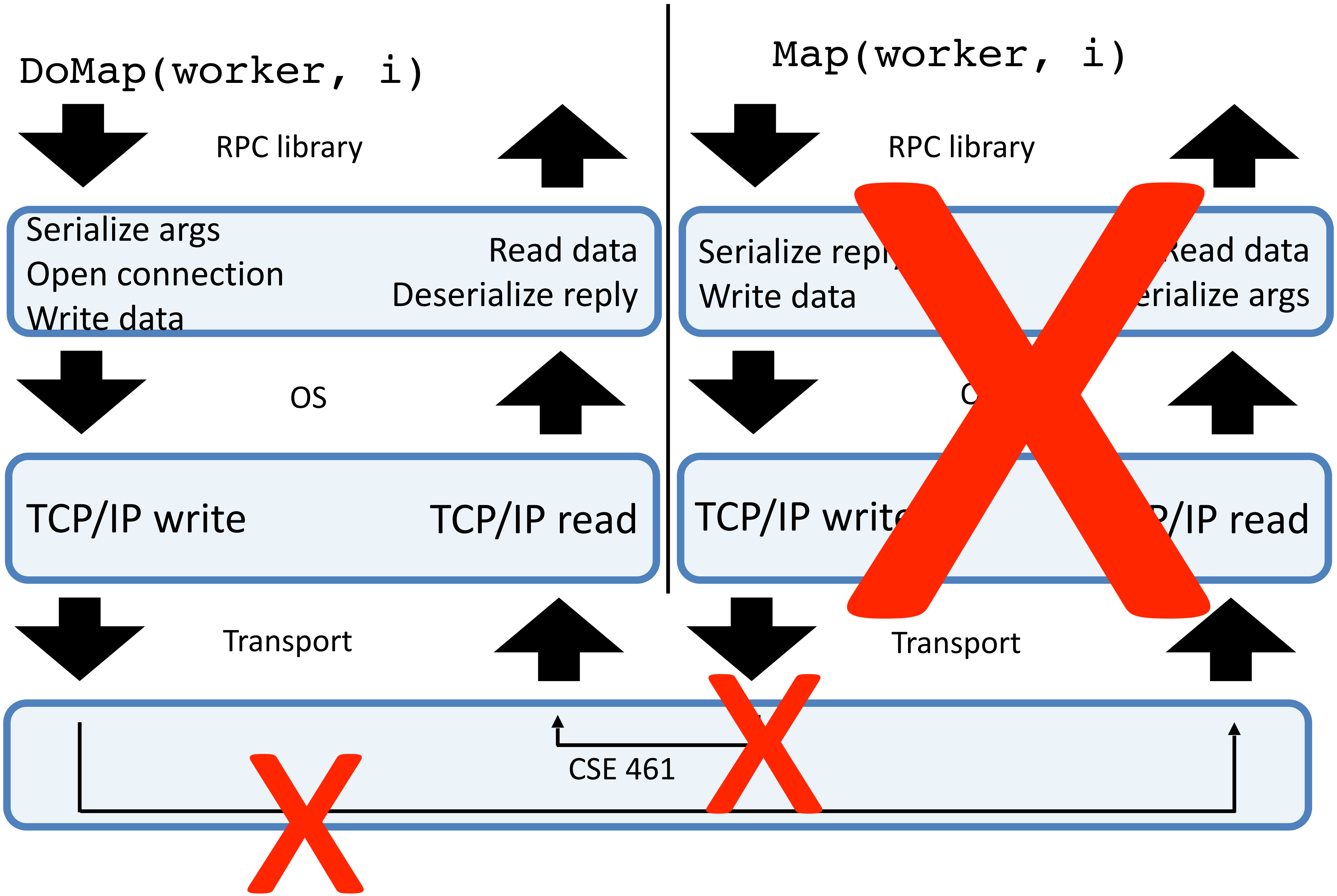
- To the client, looks like a procedure call
- To the server, looks like an implementation of a procedure call

Remote Procedure Call (RPC)

A request from the client to execute a function on the server.

- On client
  - Ex: result = DoMap(worker, i)
  - Parameters marshalled into a message (can be arbitrary types)
  - Message sent to server (can be multiple pkts)
  - Wait for reply
- On server
  - message is parsed
  - operation DoMap(i) invoked
  - Result marshalled into a message (can be multiple pkts)
  - Message sent to client

RPC implementation



RPC vs. Procedure Call

- What is equivalent of:
  - The name of the procedure?
  - The calling convention?
  - The return value?
  - The return address?

RPC vs. Procedure Call

Binding

- Client needs a connection to server
- Server must implement the required function
- What if the server is running a different version of the code?

Performance

- procedure call: maybe 10 cycles = ~3 ns
- RPC in data center: 10 microseconds => ~1K slower
- RPC in the wide area: millions of times slower

RPC vs. Procedure Call

Failures

- What happens if messages get dropped?
- What if client crashes?
- What if server crashes?
- What if server crashes after performing op but before replying?
- What if server appears to crash but is slow?
- What if network partitions?

Semantics

- Semantics = meaning

- reply == ok => ???
- reply != ok => ???

Semantics

- At least once (NFS, DNS)
  - true: executed at least once
  - false: maybe executed, maybe multiple times
- At most once
  - true: executed once
  - false: maybe executed, but never more than once
- Exactly once
  - true: executed once
  - false: never returns false

At Least Once

RPC library waits for response for a while

If none arrives, re-send the request

Do this a few times

Still no response -- return an error to the application

Non-replicated key/value server

Client sends Put k v

Server gets request, but network drops reply

Client sends Put k v again

- should server respond "yes"?
- or "no"?

What if op is “append”?

Does TCP Fix This?

- TCP: reliable bi-directional byte stream between two endpoints
  – Retransmission of lost packets
  – Duplicate detection
- But what if TCP times out and client reconnects?
  – Browser connects to Amazon
  – RPC to purchase book
  – Wifi times out during RPC
  – Browser reconnects

When does at-least-once work?

- If no side effects
  - read-only operations (or idempotent ops)
- Example: MapReduce
- Example: NFS
  - readFileBlock
  - writeFileBlock

At Most Once

Client includes unique ID (UID) with each request

- use same UID for re-send

Server RPC code detects duplicate requests

- return previous reply instead of re-running handler

```
if seen[uid] {
    r = old[uid]
} else {
    r = handler()
    old[uid] = r
    seen[uid] = true
}
```

Some At-Most-Once Issues

How do we ensure UID is unique?

- Big random number?
- Combine unique client ID (IP address?) with seq #?
- What if client crashes and restarts? Can it reuse the same UID?
- In labs, nodes never restart
- Equivalent to: every node gets new ID on start

When Can Server Discard Old RPCs?

Option 1:

- Never?

Option 2:

- unique client IDs
- per-client RPC sequence numbers
- client includes "seen all replies <= X" with every RPC

Option 3: only allow client one outstanding RPC at a time

- arrival of seq+1 allows server to discard all <= seq

Labs use Option 3

What if Server Crashes?

If at-most-once list of recent RPC results is stored in memory, server will forget and accept duplicate requests when it reboots

- Does server need to write the recent RPC results to disk?
- If replicated, does replica also need to store recent RPC results?

In Labs, server gets new address on restart

- Client messages aren’t delivered to restarted server