Lecture 3.

RPC

Last week:
- Causal Order & Traces
- Correctness conditions restrict what traces are admissible (can be produced by a correct protocol)
  ⇒ Safety: Bad state never occurs
  ⇒ Liveness: There is a path to good state
- Next week: Return to talking about traces & correctness

This week: Remote procedure calls
Why?

- The way most practical distributed systems are implemented today
  - gRPC, Thrift, ...

- Found their way into how many of the protocols we will talk about are described

How Do RPCs Affect Protocol Descriptions

On Init:

- A

On Recv Add(x,y):

- Send (A, Sum(x+y))
On Recv SUM(x):
  ...

On INIT:
  x = B.ADD(1, 2)

# CALLS HAVE RETURNS
-> MIGHT BE NULL, BUT SIGNALS COMPLETION

-> GOOD PRACTICE. WHY?

* ALGORITHM & PROGRAM STRUCTURE REQUIRES BEING AWARE
  OF CONCURRENCY

A

**HANDLER FANCY()**:

x = B.ADD(7, 8)

If x > 3 := 0:
  \[ y = C\cdot \text{Mul}(2, 5) \]
Else:
  \[ y = C\cdot \text{Div}(...) \]
RETURN x + y

B

**HANDLER ADD(x, y):**

If x > 3 := 0:
  RETURN x + y

**HANDLER FOO:**

A. FANCY()

C

Sources of Concurrency

- RPC calls from other nodes
CONSIDERATIONS WHEN READING & IMPLEMENTING ALGORITHMS SPECIFIED USING RPC

- CONCURRENCY

\[ \text{for } p \in P \exists \] p. I am Alive ( )

COMPARISON TO PRACTICE TODAY

Birrel & Nelson VS GRPC/Thrift
System Design Considerations When Using RPC

- Concurrency & Overload

*Tracing, Debugging, Reasoning About Performance*
Failures & Failure Handling

A. Fancy

B. FOO

Send(B, FOO, ..., C)

C

I/O
Req \( X \)

Ret \( X \)

\[
\begin{align*}
\text{op}_1 & \leftarrow \text{RPCcall}, \quad \text{op}_2 \\
\text{req} \quad \text{op}_1 & \quad \text{net} \quad \text{op}_1
\end{align*}
\]

\( \text{op}_1 \) and \( \text{op}_2 \) are concurrent

\[
(\text{req} \quad \text{op}_1, \quad \text{net} \quad \text{op}_1) \cap \\
(\text{req} \quad \text{op}_2, \quad \text{net} \quad \text{op}_2) \neq \emptyset
\]
Linearizable Queue

\[ \text{op} \leftarrow \text{enq}(5) \]
\[ \text{op2} \leftarrow \text{enq}(3) \]
\[ \text{op3} \leftarrow \text{deq}() \]