

# CS 518

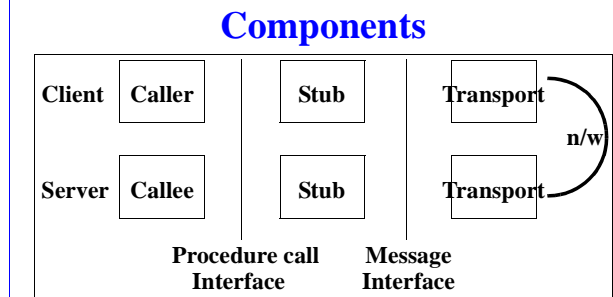
## RPC: Communication Semantics and Its Implication on Implementation

Randy Wang  
Fall 2002  
Princeton University

- ### RPC vs. TCP
- Request/response style of interaction
    - TCP: one way long streams
  - Lightweight
    - Did “everything” (including their own transport)
    - Few extra messages
    - TCP: typically heavily layered
    - TCP: many extra messages
      - Managing connection
      - Managing reliable transmission
  - Little state
    - TCP: lots of state

## Procedure Call Semantics

- Synchronous
- Little data
- Latency is important
- Implementation exploits these semantics

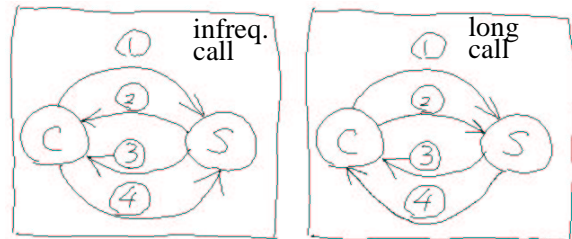


- Caller, callee, procedure call interface: (illusion of) conventional mesa semantics
- Stubs: transform between arguments/results and messages
- Transport: reliable delivery
  - What’s the definition of “reliable”?

## Packet Protocol: Goals

- Latency is the name of the game
- Minimize state

## Packet Protocol: Reliable Messages



- Reliable messages: messages must be acked
- Common case: one packet in each direction
  - Reply acks request
  - Next request acks previous request
  - Benefits from procedure semantics
- If not acked, retransmit
- Long calls: client pings
- Large messages
  - Ack per packet
  - Adds round trip time to each packet
  - Loses bandwidth
  - Allows them to minimize state

## Packet Protocol: State

- Need one word of sequence #
  - Distinguish between new request and retransmission of old request
  - Does not assume idempotency of calls (“un-NFS”)
- Server: “call ID table”
  - Server state can be flushed
  - What’s the right magic number for server state flushing?
- Client: options of granularity for multiplexing conversations
  - Host, process, thread (not just “active” ones)
  - Sprite “channels”
  - Maximize re-use while allowing concurrency
- Server process re-use
  - Pool of pre-spawned processes
  - Maximize re-use vs. allowing concurrency