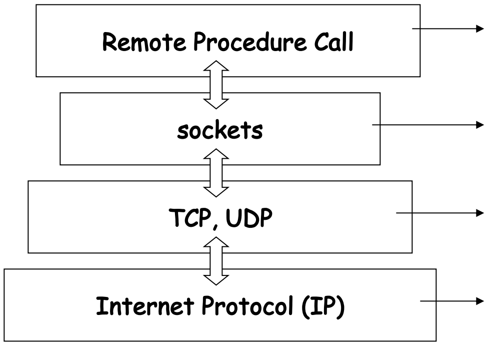
REMOTE PROCEDURE CALL

RPC as a Programming Abstraction



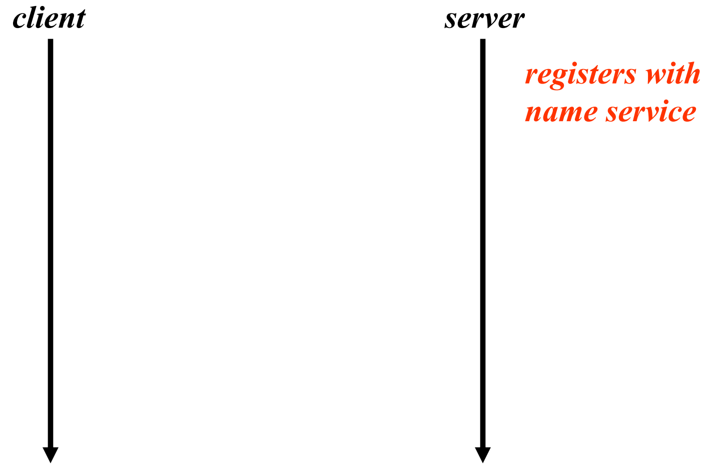
Remote Procedure Call:
hides communication details behind
a procedure call and helps bridge
heterogeneous platforms

sockets:
operating system level interface to the
underlying communication protocols

TCP, UDP:
User Datagram Protocol (UDP) transports
data packets without guarantees
Transmission Control Protocol (TCP)
verifies correct delivery of data streams

Internet Protocol (IP):
moves a packet of data from one node
to another

The basic RPC protocol



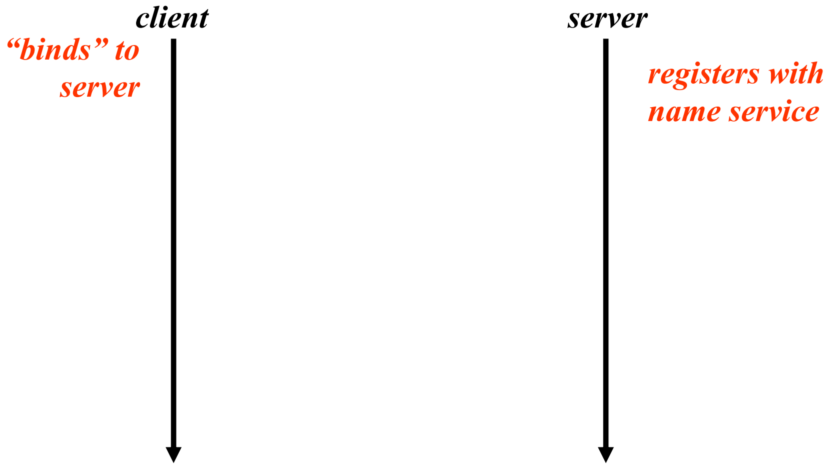
client

server

registers with name service

53

The basic RPC protocol



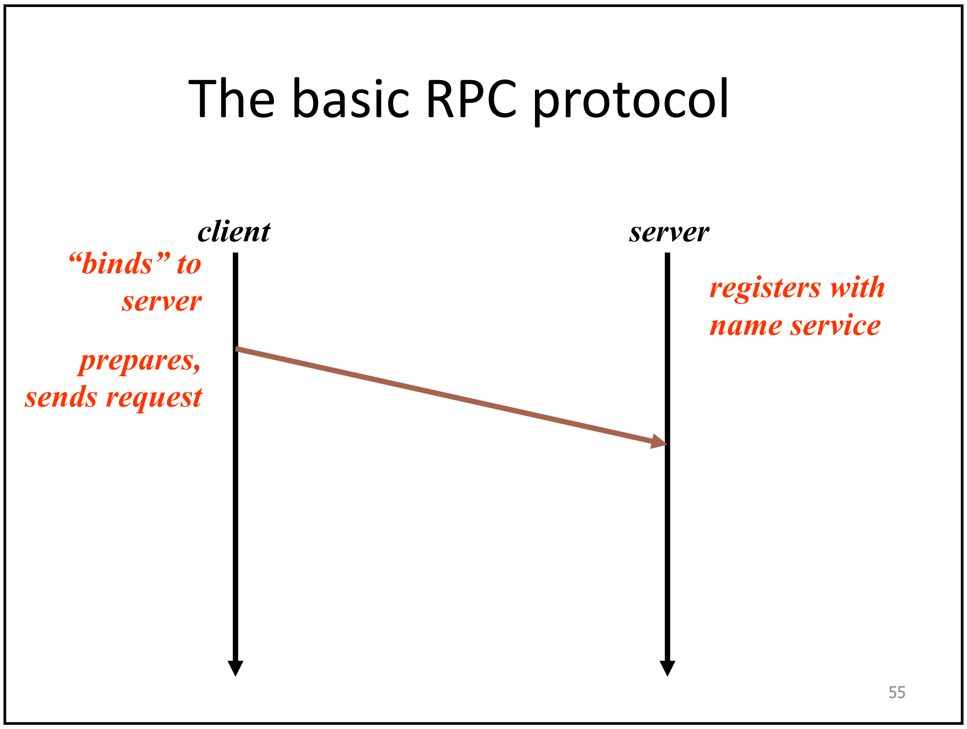
client

"binds" to server

server

registers with name service

54



The basic RPC protocol

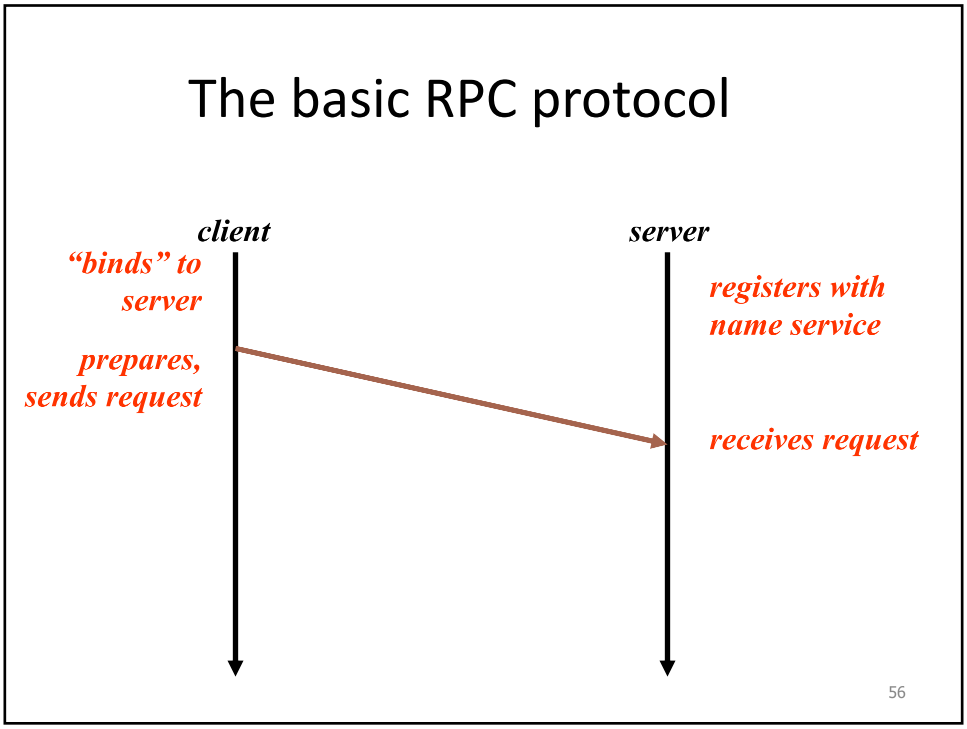
client

server

"binds" to server

registers with name service

prepares, sends request



The basic RPC protocol

client

server

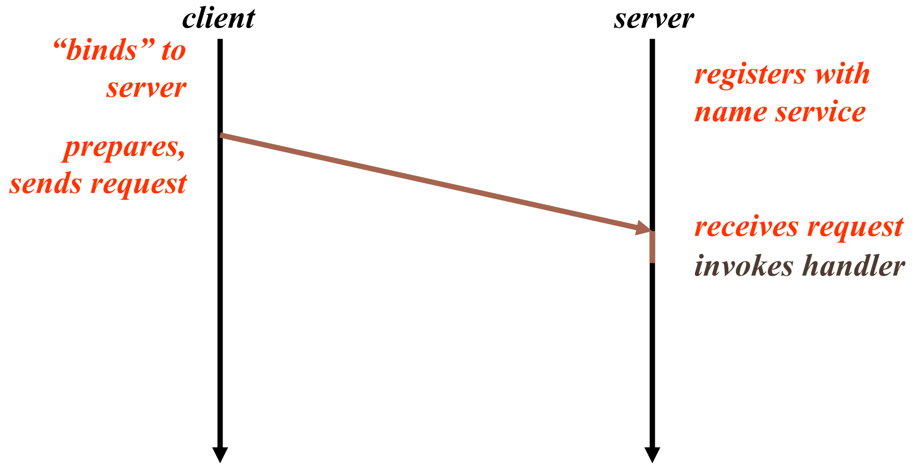
"binds" to server

registers with name service

prepares, sends request

receives request

The basic RPC protocol



client

server

"binds" to server

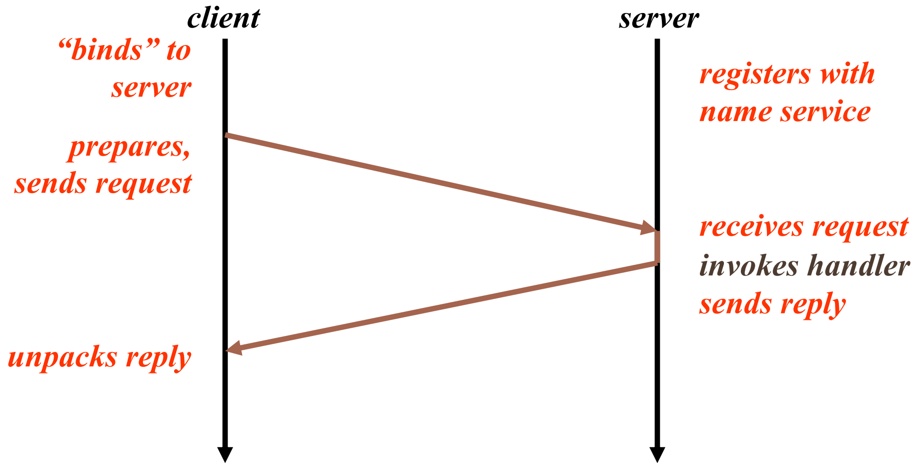
prepares, sends request

registers with name service

receives request

invokes handler

The basic RPC protocol



client

server

"binds" to server

prepares, sends request

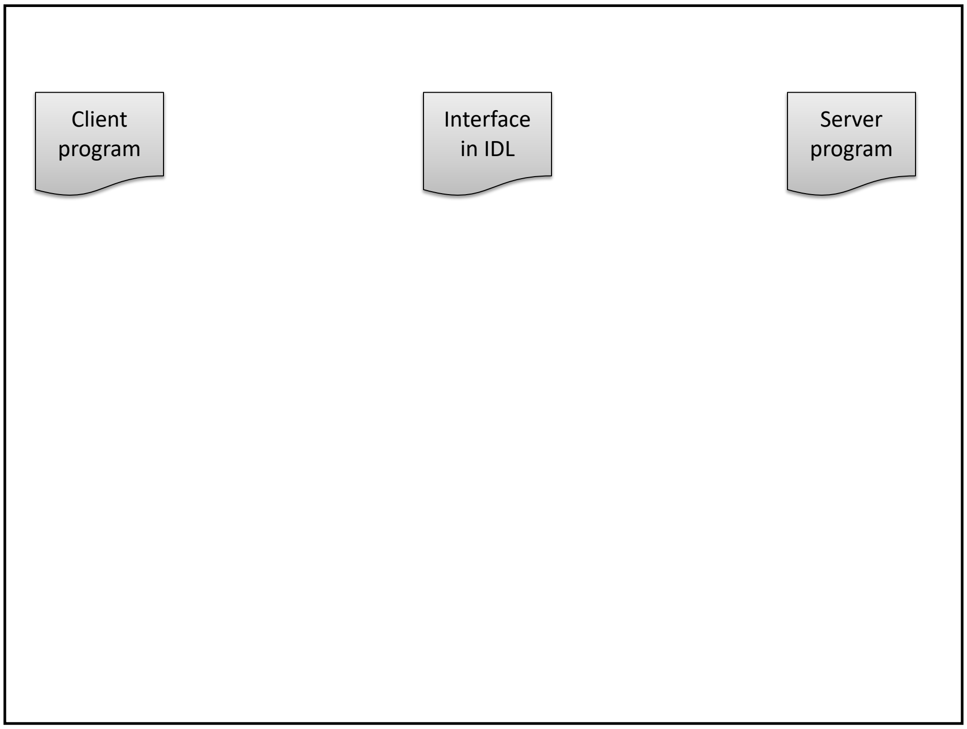
unpacks reply

registers with name service

receives request

invokes handler

sends reply

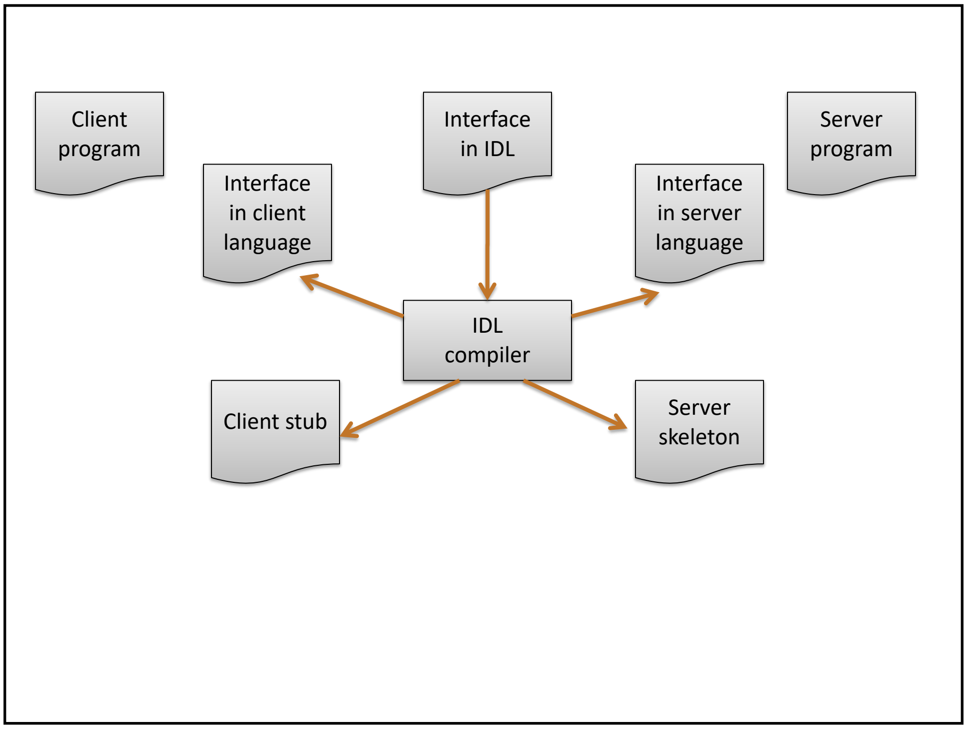


Client program

Interface in IDL

Server program

59



Client program

Interface in IDL

Server program

Interface in client language

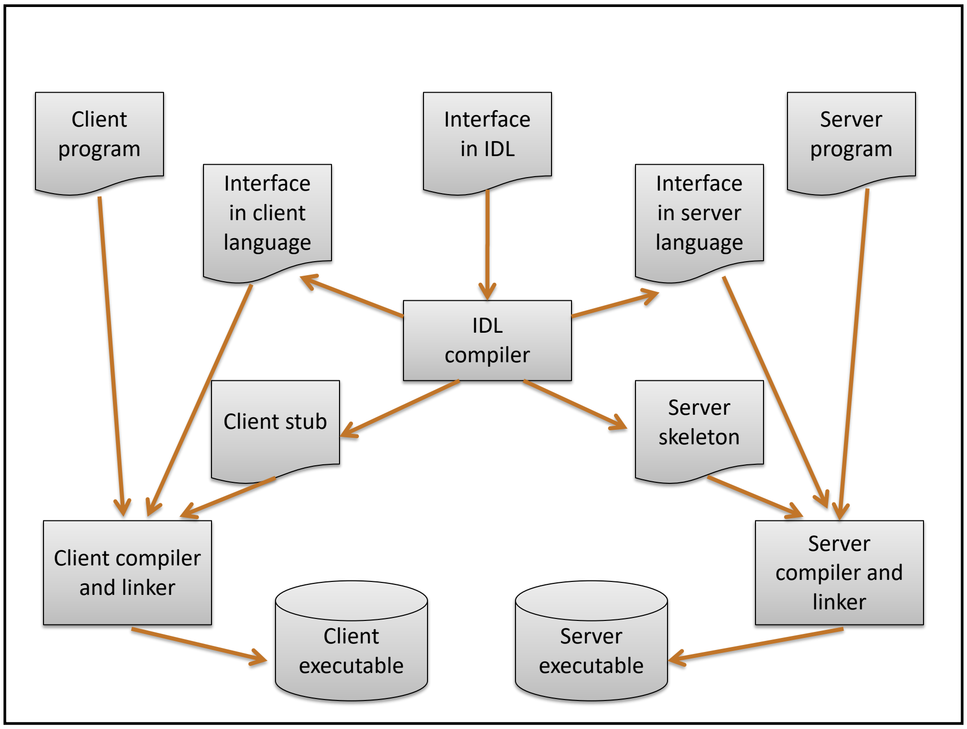
IDL compiler

Interface in server language

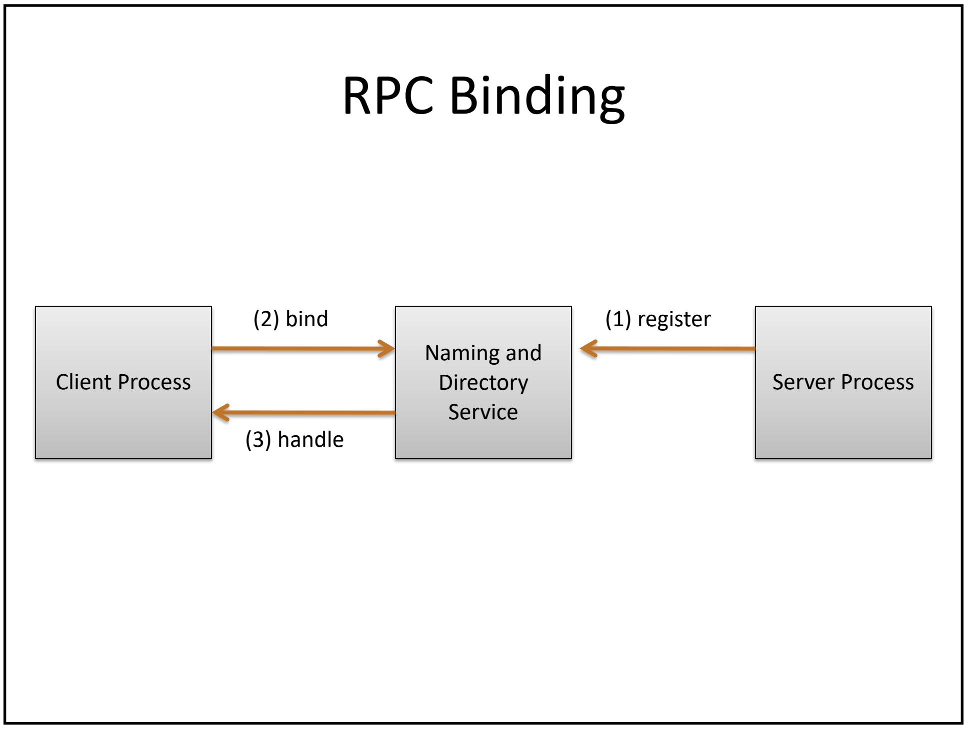
Client stub

Server skeleton

60



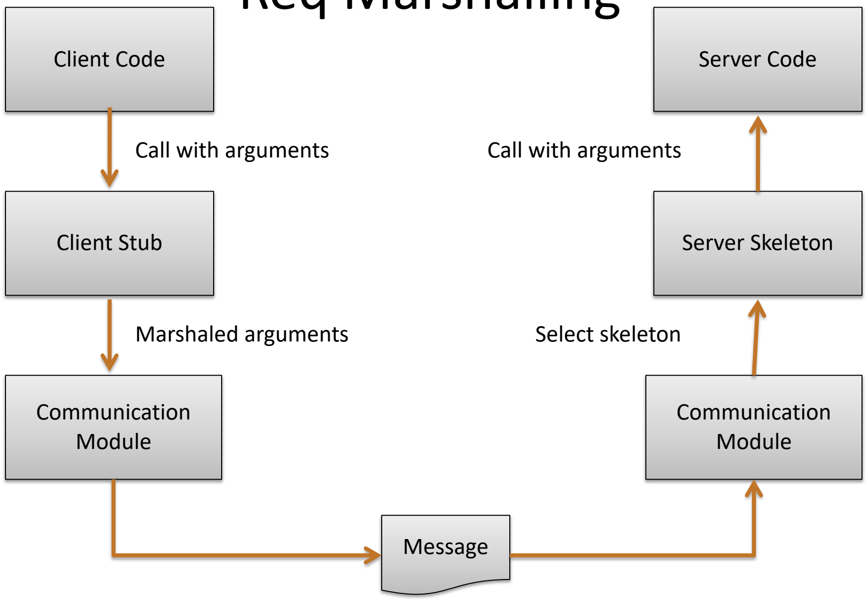
61



RPC Binding

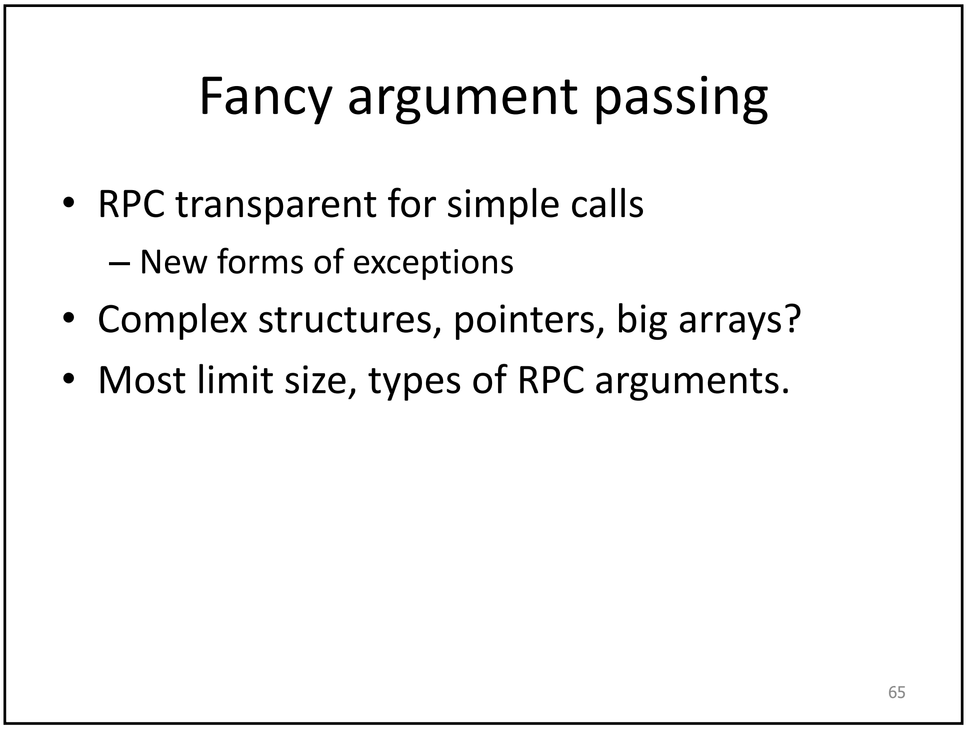
62

Req Marshalling



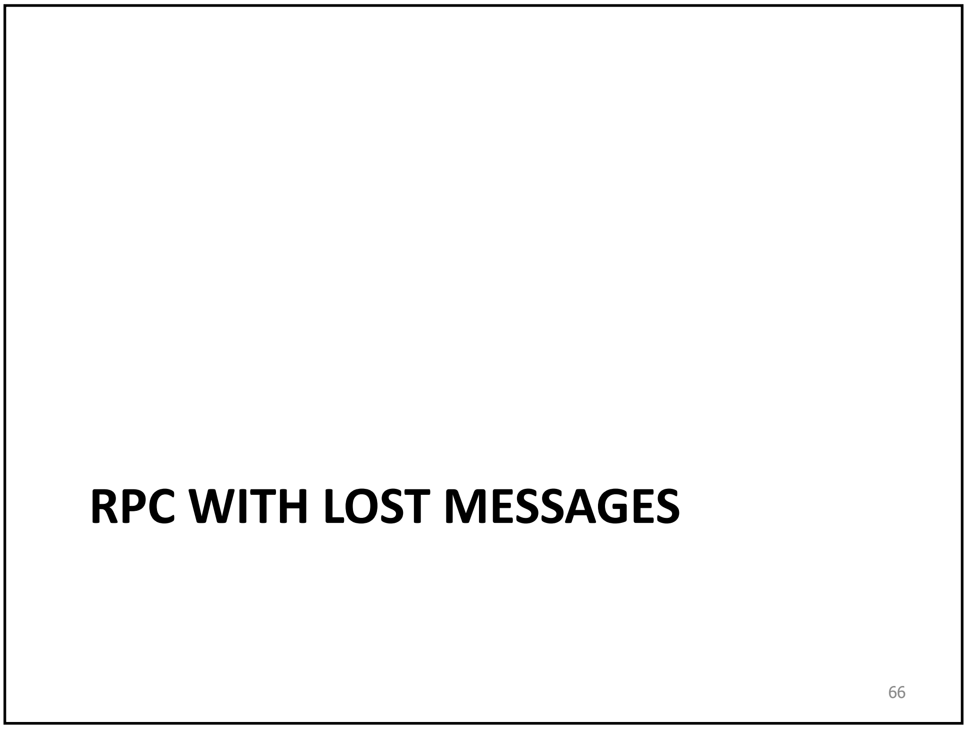
Data in messages

- Marshalling
- Byte ordering issues (big-endian versus little endian), strings (some CPUs require padding), alignment, etc
- As fast as possible
  - – yet must also be very general

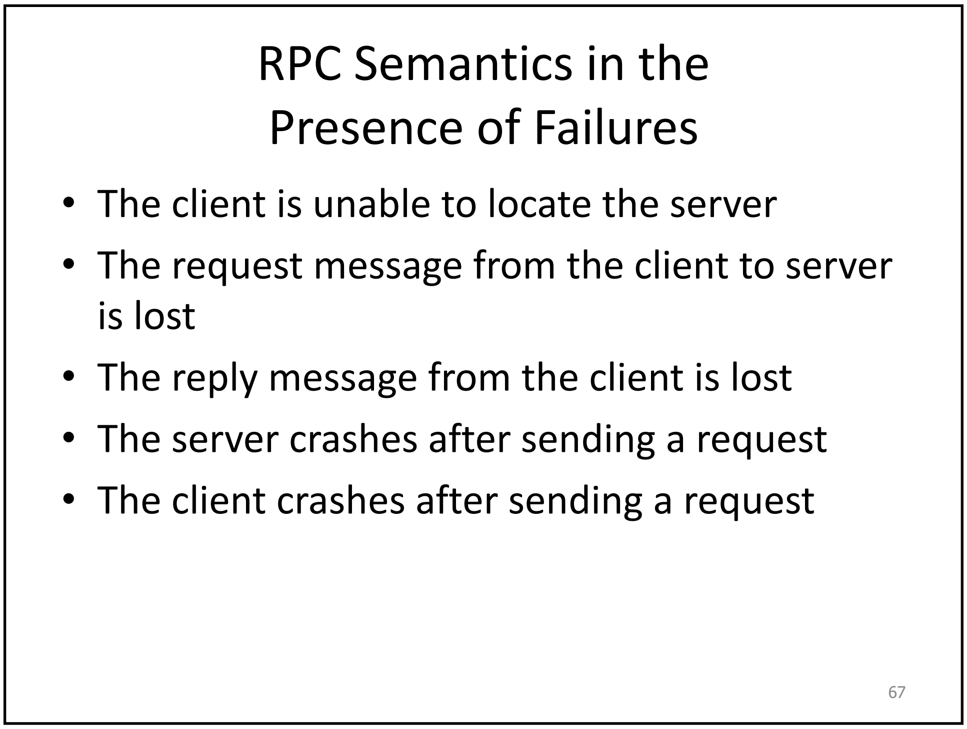


Fancy argument passing

- RPC transparent for simple calls
  - New forms of exceptions
- Complex structures, pointers, big arrays?
- Most limit size, types of RPC arguments.



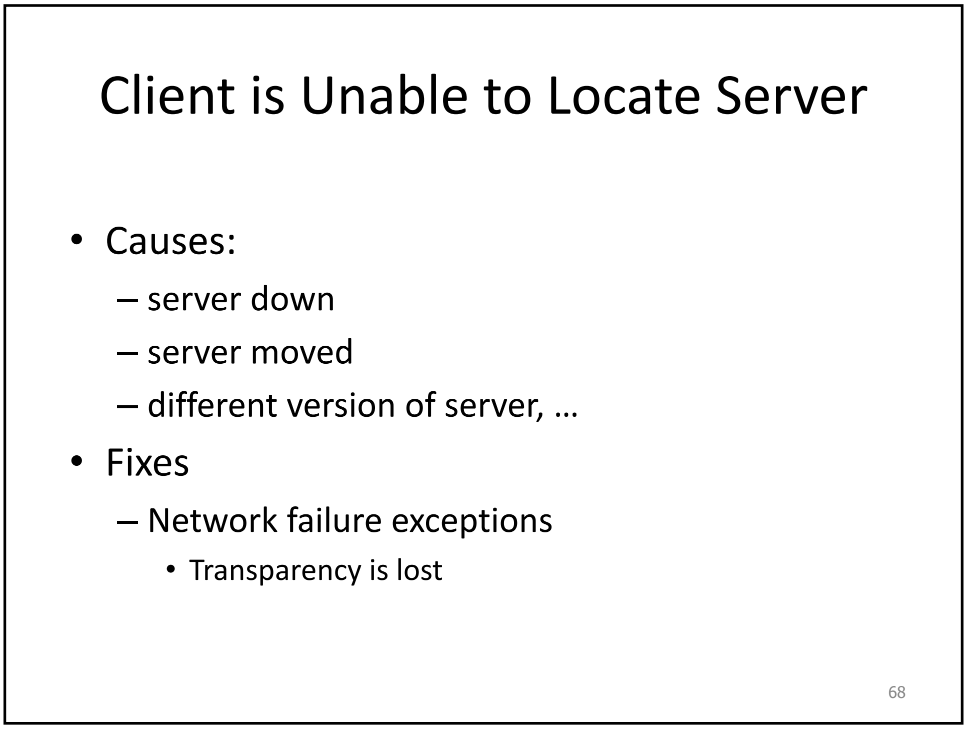
RPC WITH LOST MESSAGES



RPC Semantics in the Presence of Failures

- The client is unable to locate the server
- The request message from the client to server is lost
- The reply message from the client is lost
- The server crashes after sending a request
- The client crashes after sending a request

67

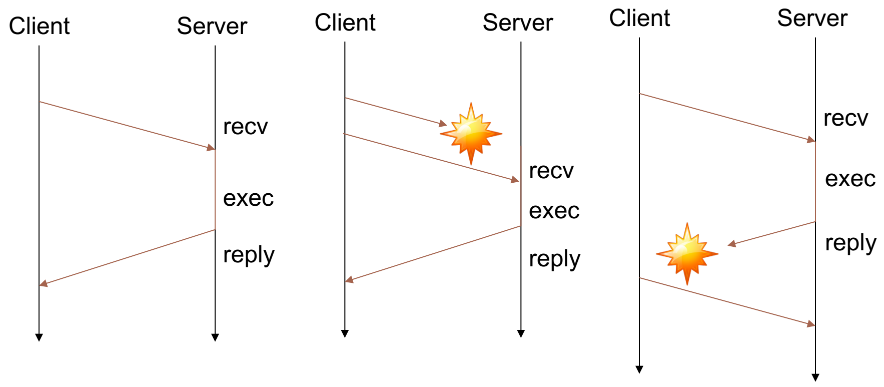


Client is Unable to Locate Server

- Causes:
  – server down
  – server moved
  – different version of server, …
- Fixes
  – Network failure exceptions
    - Transparency is lost

68

Messages Lost

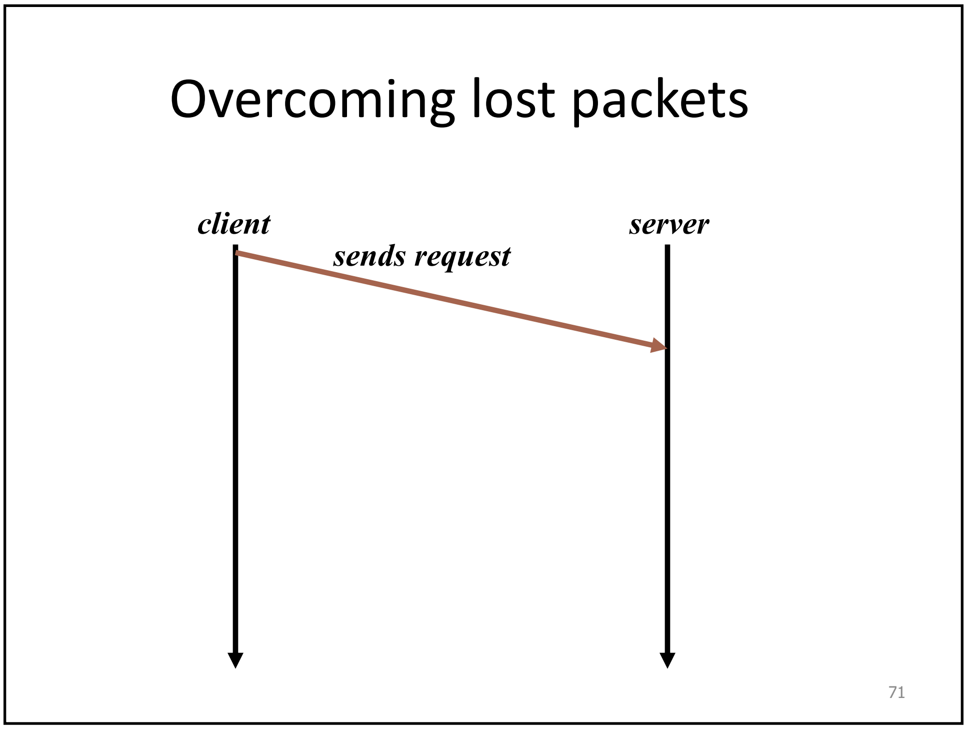


69

Lost Request Message

- Easiest to deal with
- Just retransmit the message!
- If multiple message are lost then
  - “client is unable to locate server” error

70



Overcoming lost packets

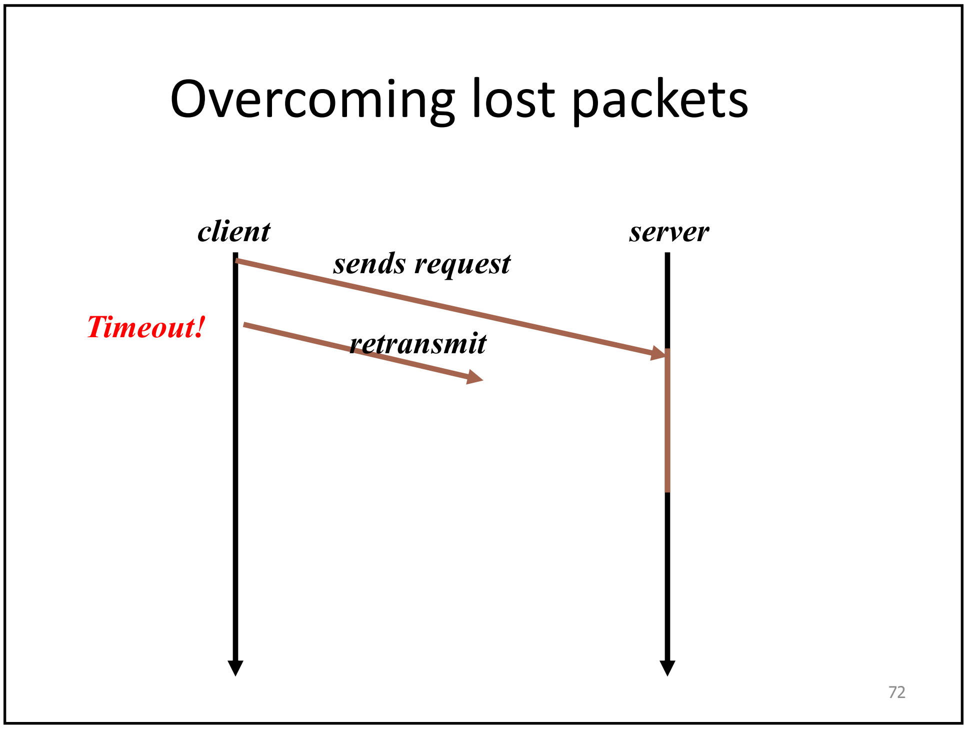
client

server

sends request

71

71



Overcoming lost packets

client

server

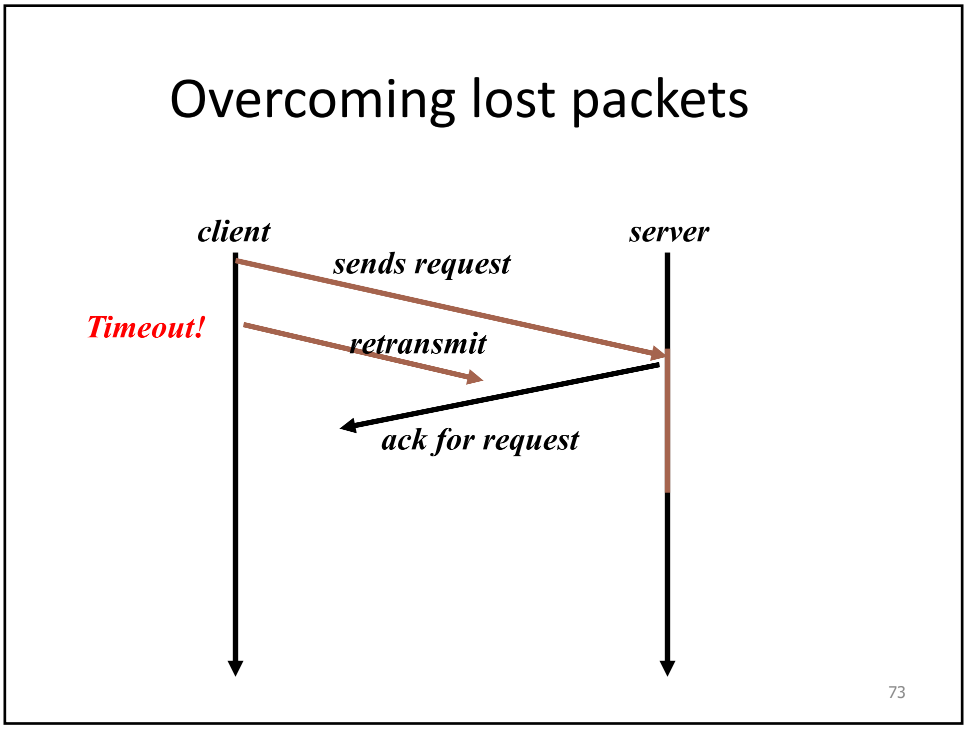
sends request

Timeout!

retransmit

72

72



Overcoming lost packets

client

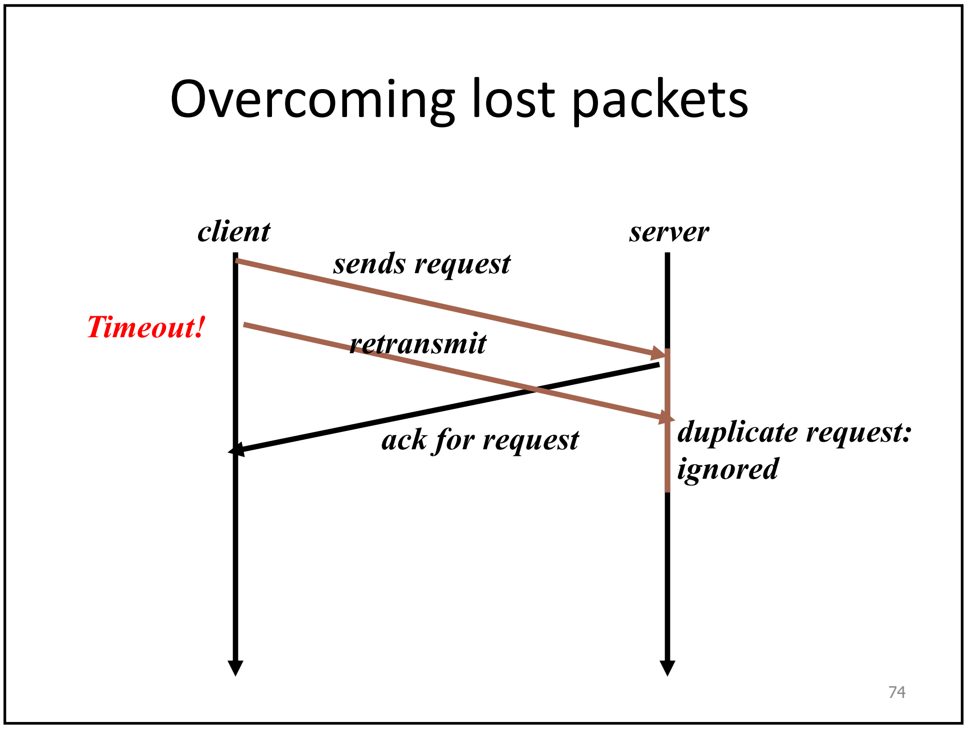
server

sends request

Timeout!

retransmit

ack for request



Overcoming lost packets

client

server

sends request

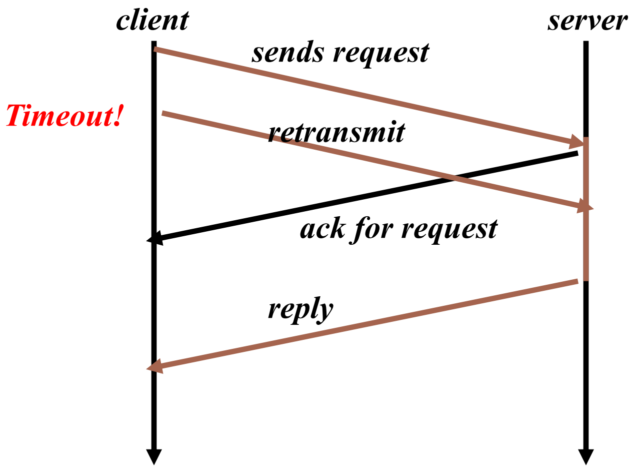
Timeout!

retransmit

ack for request

duplicate request:
ignored

Overcoming lost packets



client

server

sends request

Timeout!

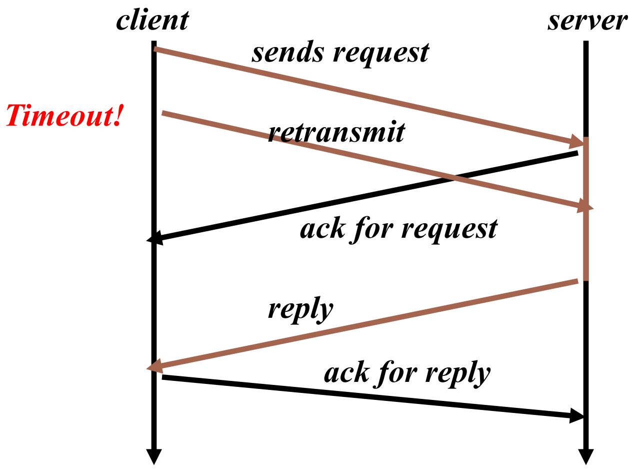
retransmit

ack for request

reply

75

Overcoming lost packets



client

server

sends request

Timeout!

retransmit

ack for request

reply

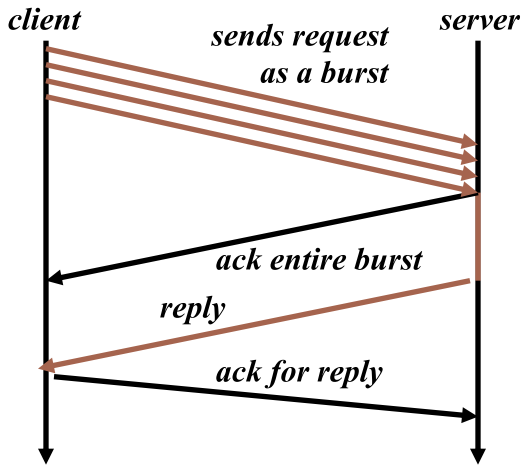
ack for reply

76

Costs in fault-tolerant version?

- Acks are expensive.
  - Suppress the initial ack
- Retransmission is costly.
  - Tune the delay
- Big messages
  - ack a burst at a time

Big packets



client

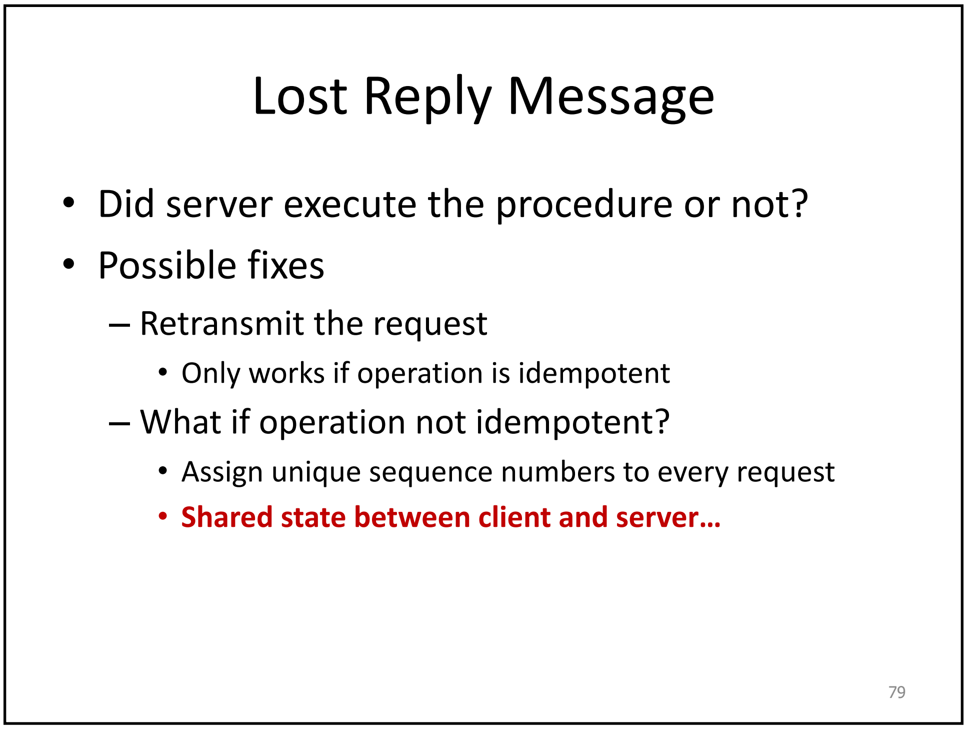
server

sends request as a burst

ack entire burst

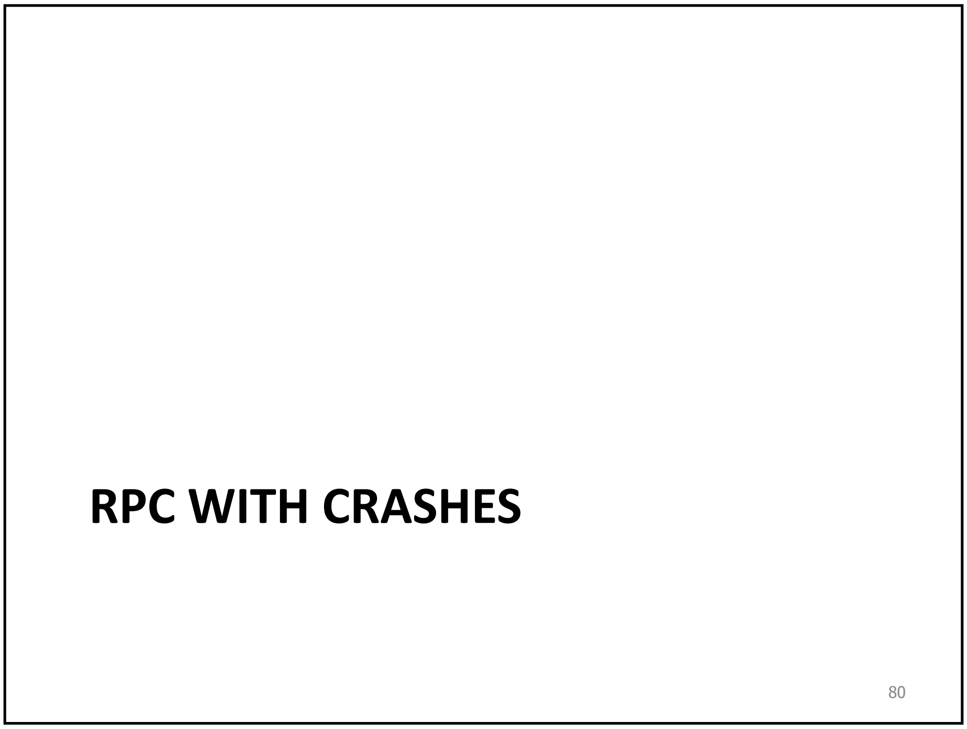
reply

ack for reply



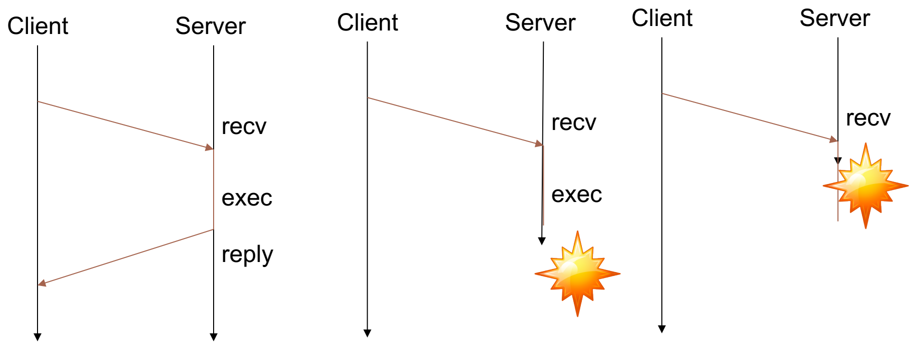
Lost Reply Message

- Did server execute the procedure or not?
- Possible fixes
  - Retransmit the request
    - Only works if operation is idempotent
  - What if operation not idempotent?
    - Assign unique sequence numbers to every request
    - **Shared state between client and server…**



RPC WITH CRASHES

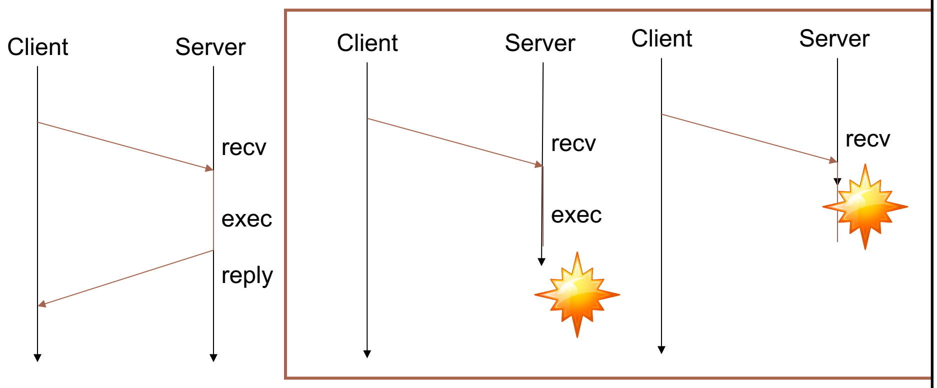
Server Crashes



81

81

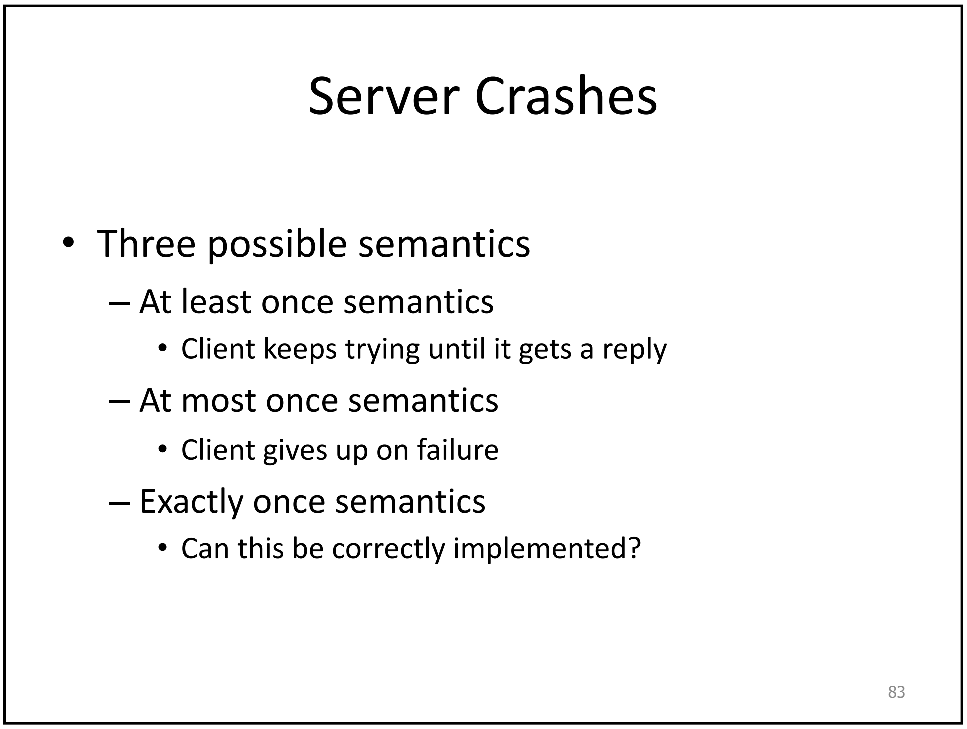
Server Crashes



Client cannot distinguish these scenarios

82

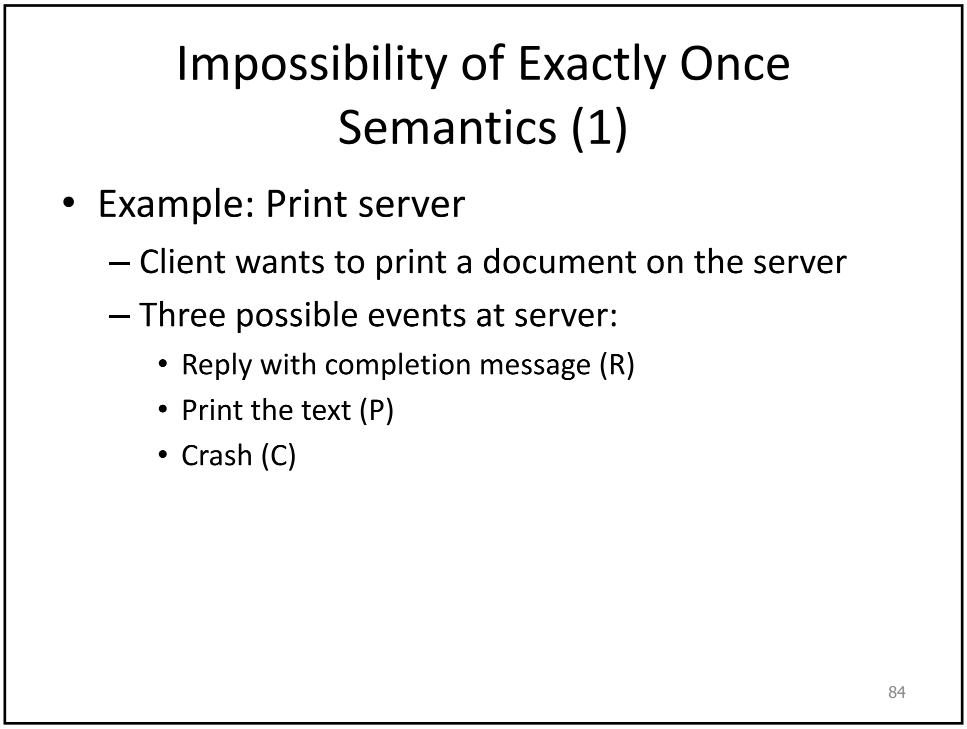
82



Server Crashes

- Three possible semantics
  - At least once semantics
    - Client keeps trying until it gets a reply
  - At most once semantics
    - Client gives up on failure
  - Exactly once semantics
    - Can this be correctly implemented?

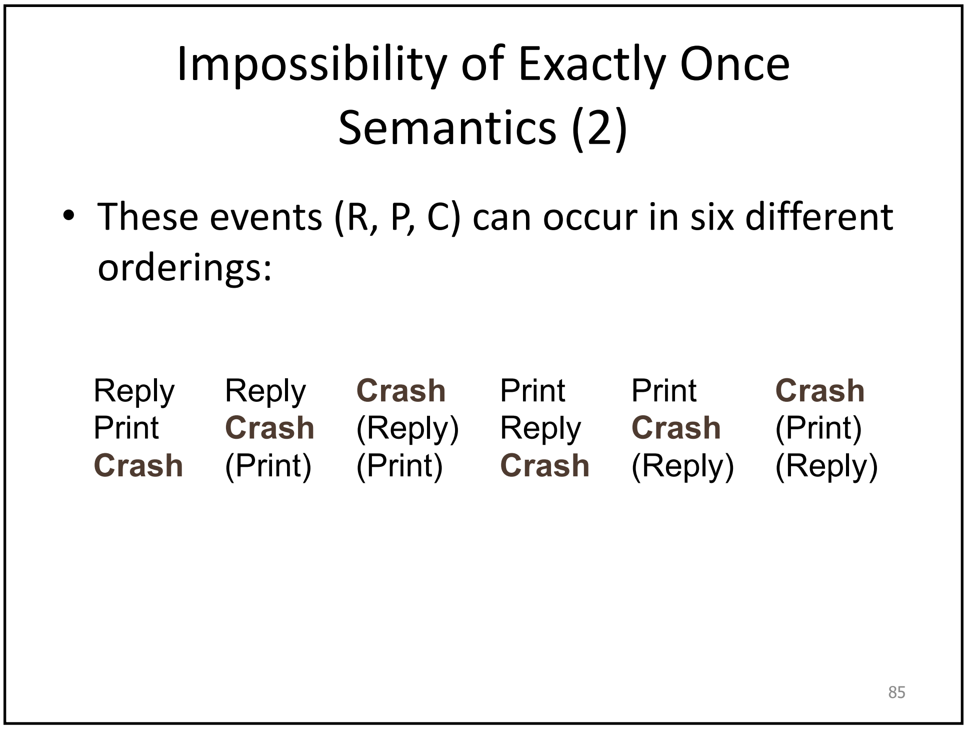
83



Impossibility of Exactly Once Semantics (1)

- Example: Print server
  - Client wants to print a document on the server
  - Three possible events at server:
    - Reply with completion message (R)
    - Print the text (P)
    - Crash (C)

84

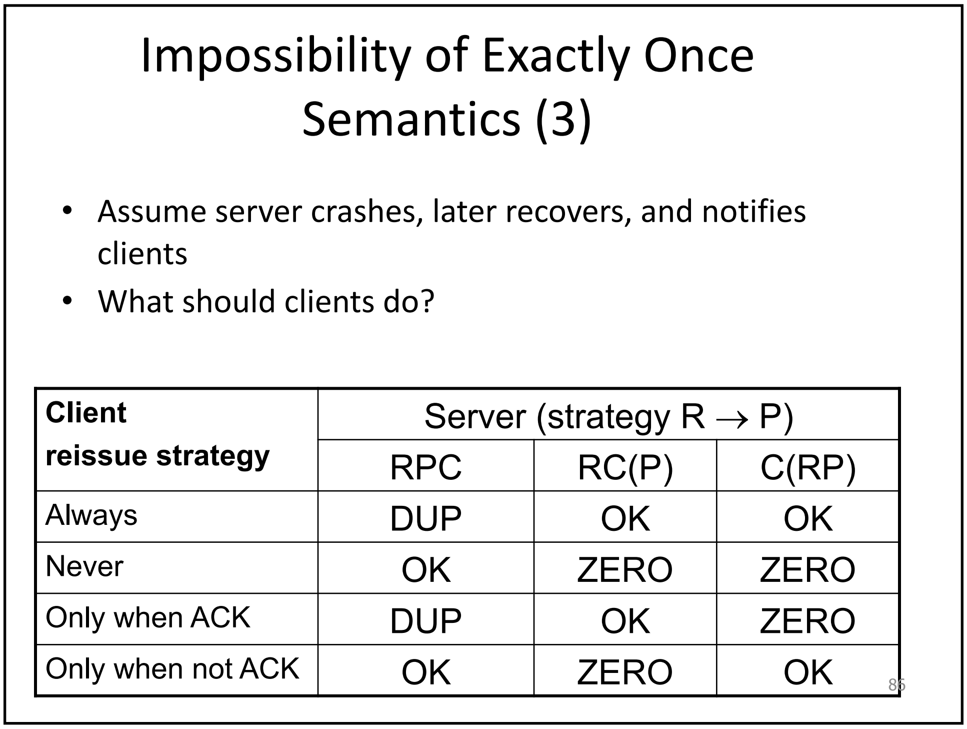


Impossibility of Exactly Once Semantics (2)

- These events (R, P, C) can occur in six different orderings:

Reply	Reply	**Crash**	Print	Print	**Crash**
Print	**Crash**	(Reply)	Reply	**Crash**	(Print)
Crash	(Print)	(Print)	**Crash**	(Reply)	(Reply)

85

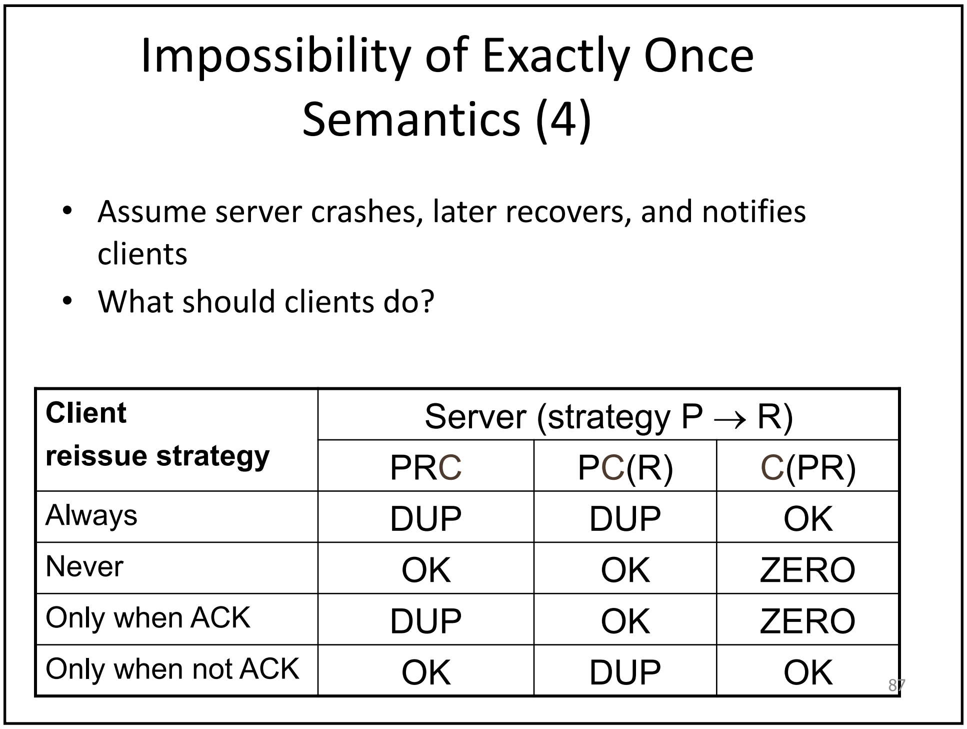


Impossibility of Exactly Once Semantics (3)

- Assume server crashes, later recovers, and notifies clients
- What should clients do?

Client reissue strategy	Server (strategy R → P)		
	RPC	RC(P)	C(RP)
Always	DUP	OK	OK
Never	OK	ZERO	ZERO
Only when ACK	DUP	OK	ZERO
Only when not ACK	OK	ZERO	OK

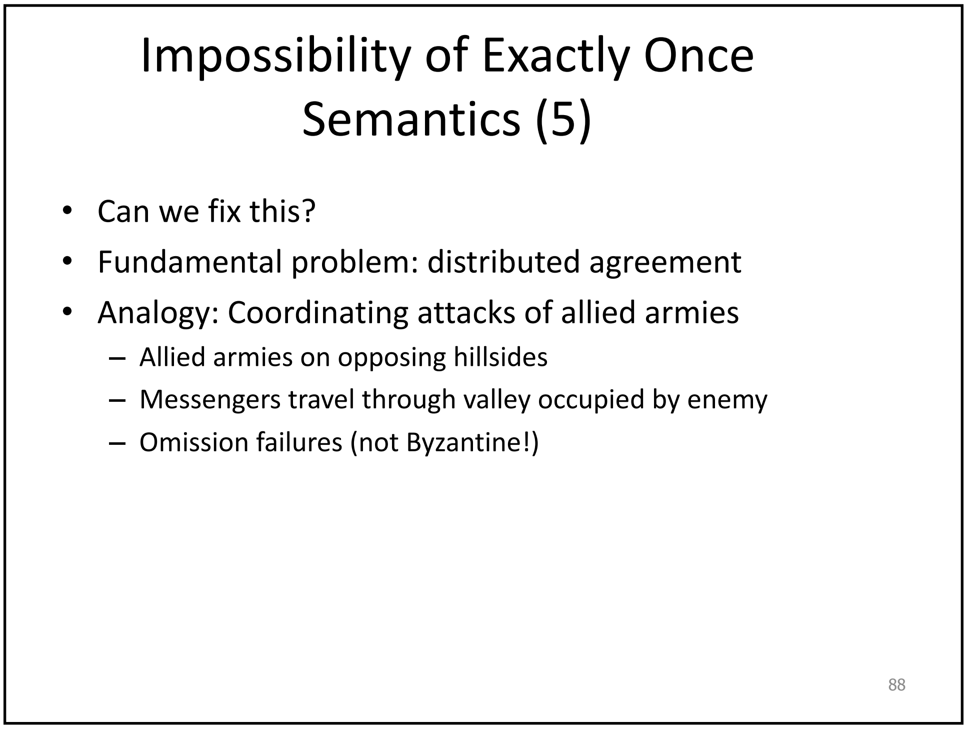
86



Impossibility of Exactly Once Semantics (4)

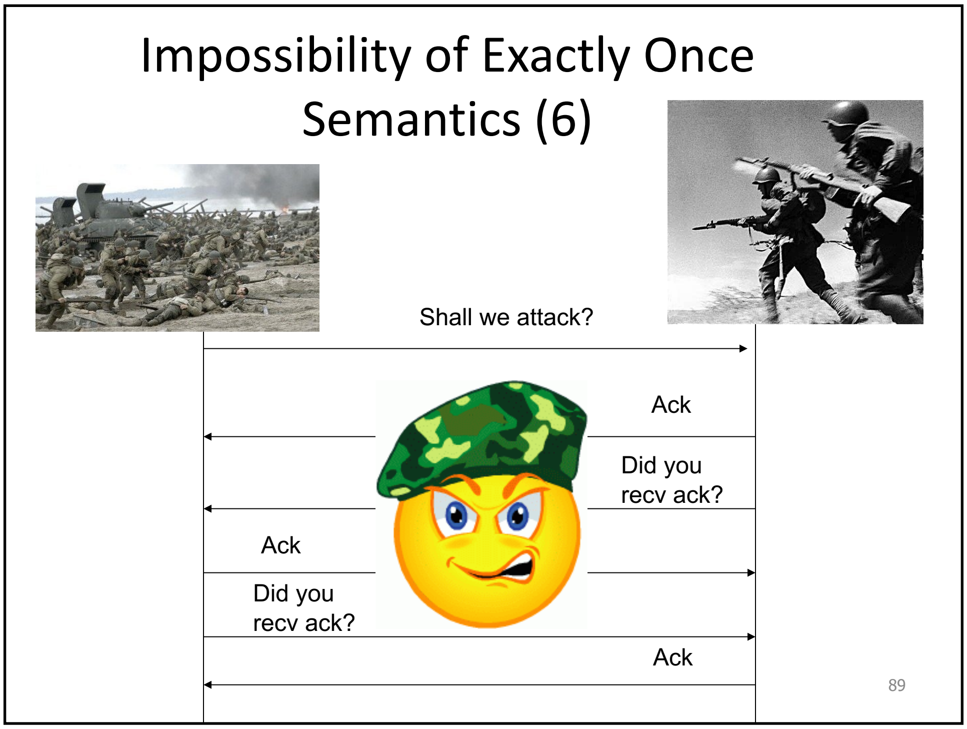
- Assume server crashes, later recovers, and notifies clients
- What should clients do?

Client reissue strategy	Server (strategy P → R)		
	PRC	PC(R)	C(PR)
Always	DUP	DUP	OK
Never	OK	OK	ZERO
Only when ACK	DUP	OK	ZERO
Only when not ACK	OK	DUP	OK



Impossibility of Exactly Once Semantics (5)

- Can we fix this?
- Fundamental problem: distributed agreement
- Analogy: Coordinating attacks of allied armies
  - Allied armies on opposing hillsides
  - Messengers travel through valley occupied by enemy
  - Omission failures (not Byzantine!)



Impossibility of Exactly Once Semantics (6)

Shall we attack?

Ack

Did you recv ack?

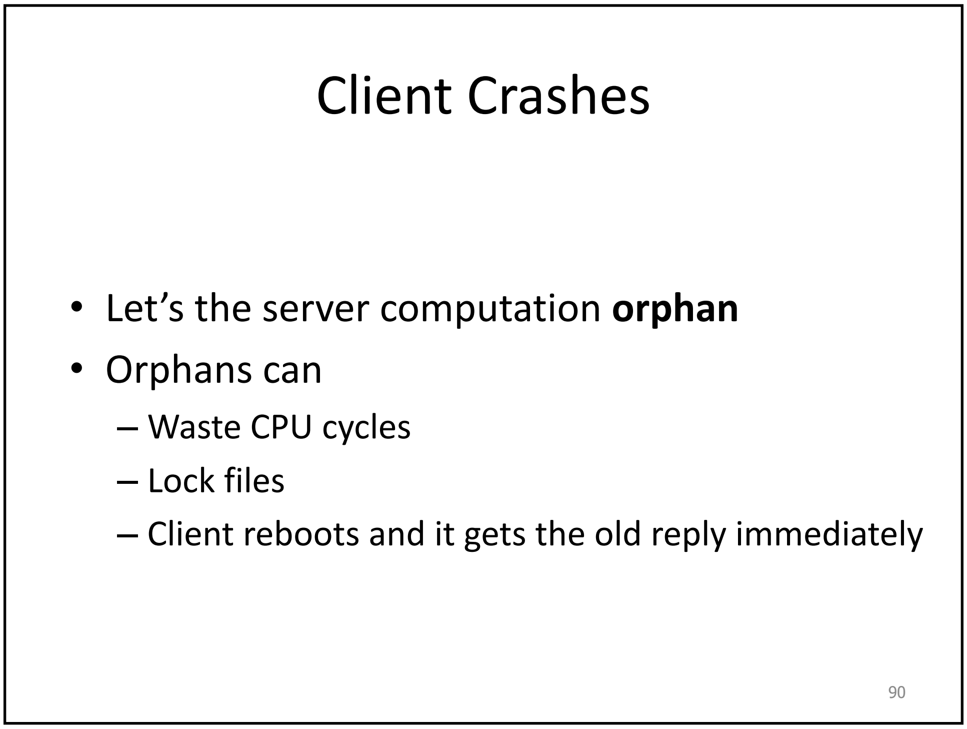
Ack

Did you recv ack?

Ack

89

89



Client Crashes

- Let’s the server computation **orphan**
- Orphans can
  - – Waste CPU cycles
  - – Lock files
  - – Client reboots and it gets the old reply immediately

90

90

Client Crashes: Possible Solutions

- Extermination:
  - – Client log, kill orphan on reboot
  - – Disadvantage: log overhead
- Reincarnation:
  - – Client broadcasts new “epoch” when reboots
  - – New epoch: servers kill orphans
  - – Disadvantage: network partition

Client Crashes: Possible Solutions

- Expiration:
  - – Each RPC is given a lease T
  - – Must renew lease before expiration
  - – If client reboots after T sec, all orphans are gone
  - – Problem: what is a good value of T?