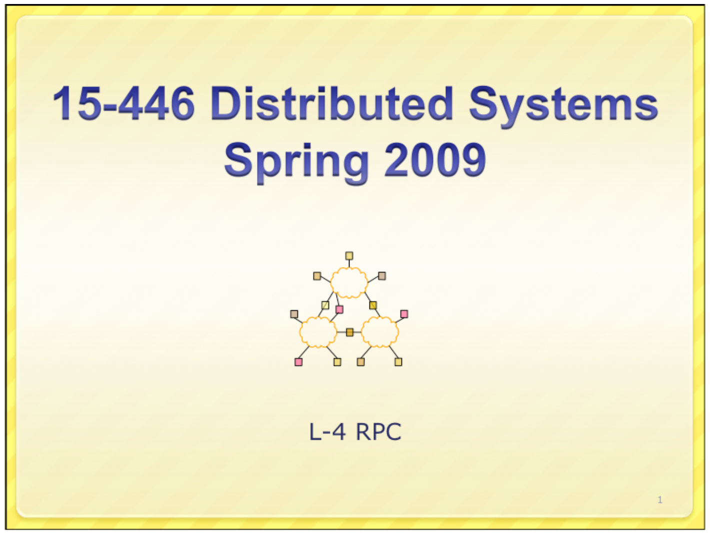
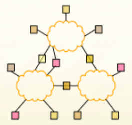
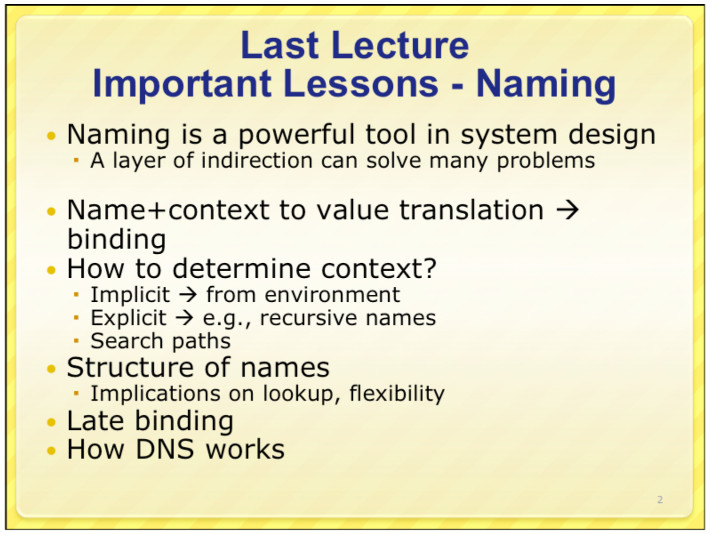


15-446 Distributed Systems Spring 2009

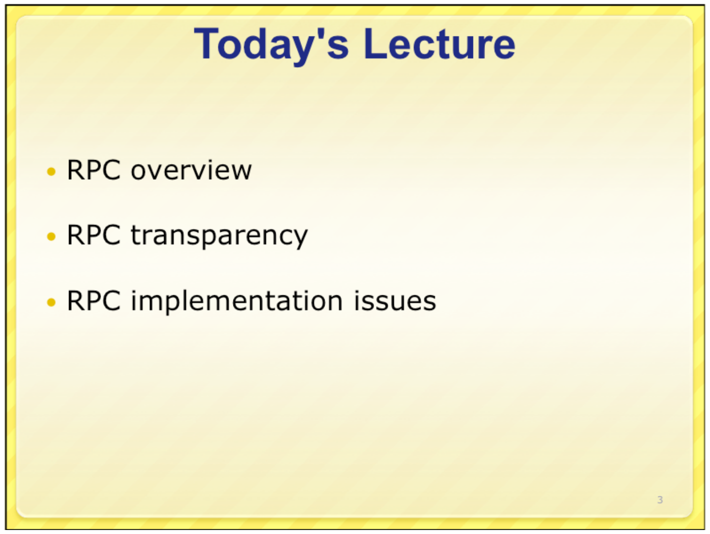


L-4 RPC



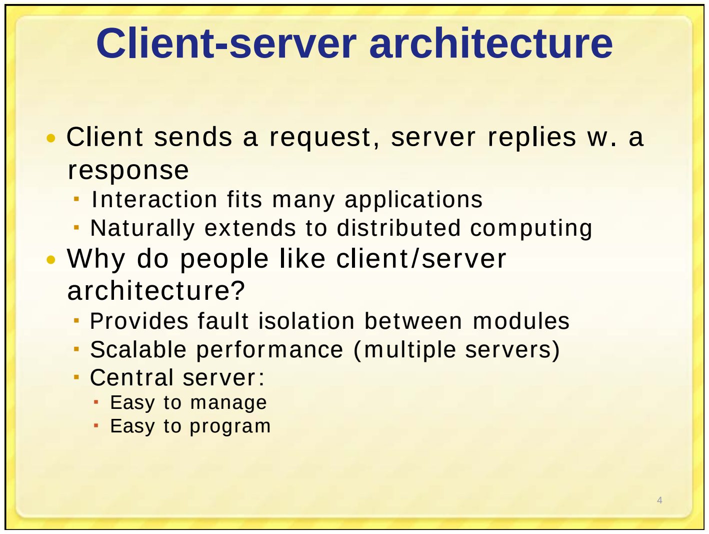
Last Lecture Important Lessons - Naming

- Naming is a powerful tool in system design
  - A layer of indirection can solve many problems
- Name+context to value translation → binding
- How to determine context?
  - Implicit → from environment
  - Explicit → e.g., recursive names
  - Search paths
- Structure of names
  - Implications on lookup, flexibility
- Late binding
- How DNS works



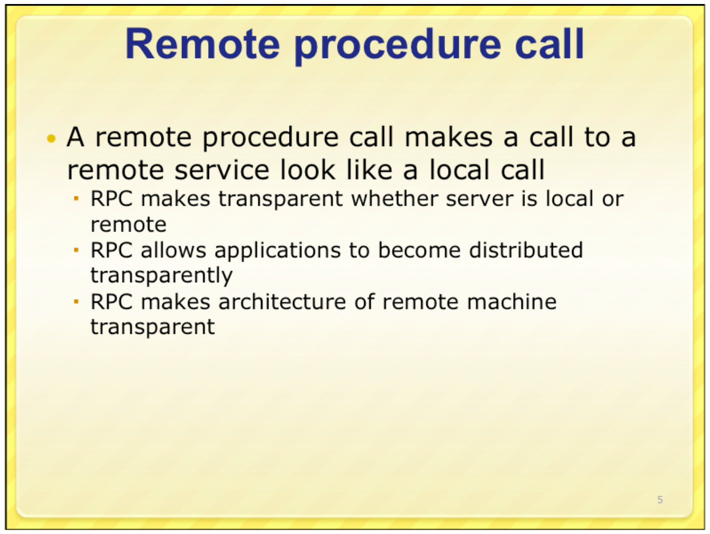
Today's Lecture

- RPC overview
- RPC transparency
- RPC implementation issues



Client-server architecture

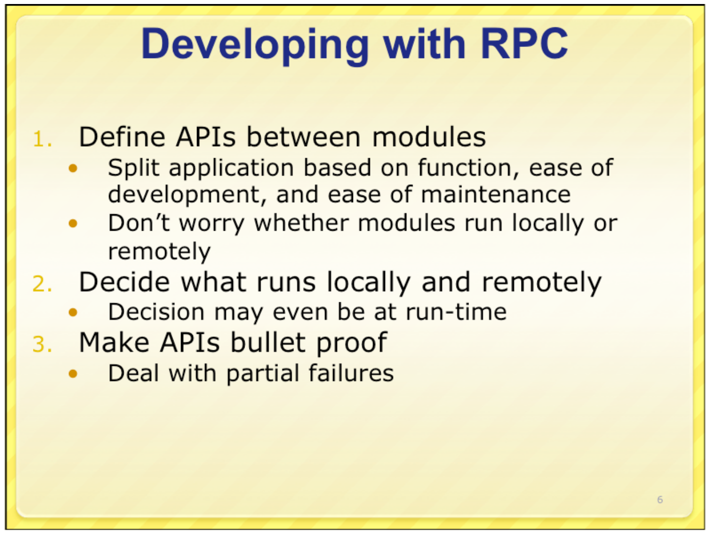
- Client sends a request, server replies w. a response
  - Interaction fits many applications
  - Naturally extends to distributed computing
- Why do people like client/server architecture?
  - Provides fault isolation between modules
  - Scalable performance (multiple servers)
  - Central server:
    - Easy to manage
    - Easy to program



Remote procedure call

- A remote procedure call makes a call to a remote service look like a local call
  - RPC makes transparent whether server is local or remote
  - RPC allows applications to become distributed transparently
  - RPC makes architecture of remote machine transparent

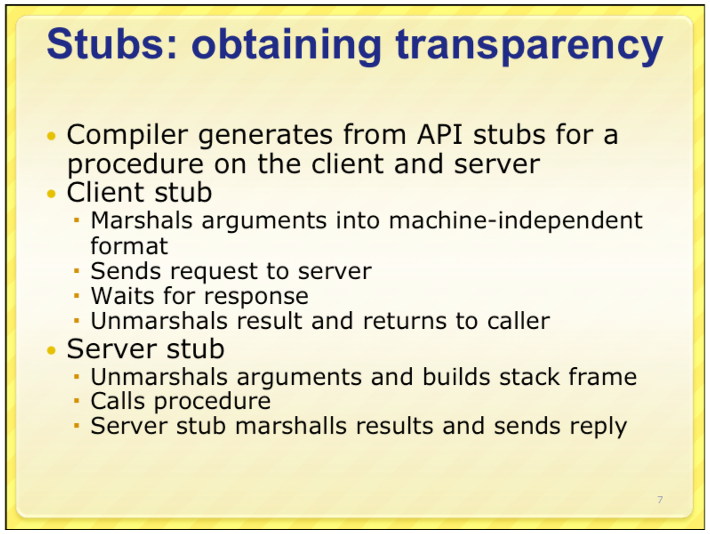
5



Developing with RPC

1. Define APIs between modules
   - Split application based on function, ease of development, and ease of maintenance
   - Don't worry whether modules run locally or remotely
2. Decide what runs locally and remotely
   - Decision may even be at run-time
3. Make APIs bullet proof
   - Deal with partial failures

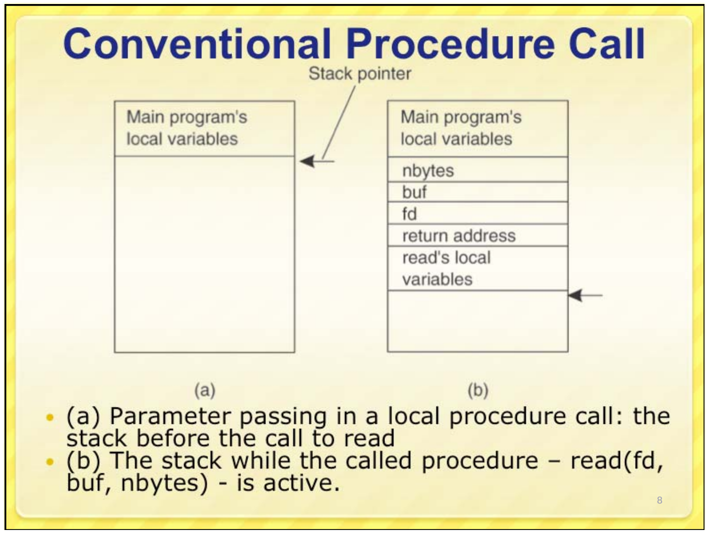
6



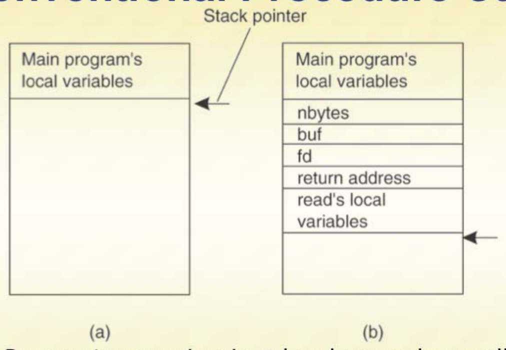
Stubs: obtaining transparency

- Compiler generates from API stubs for a procedure on the client and server
- Client stub
  - Marshals arguments into machine-independent format
  - Sends request to server
  - Waits for response
  - Unmarshals result and returns to caller
- Server stub
  - Unmarshals arguments and builds stack frame
  - Calls procedure
  - Server stub marshalls results and sends reply

7



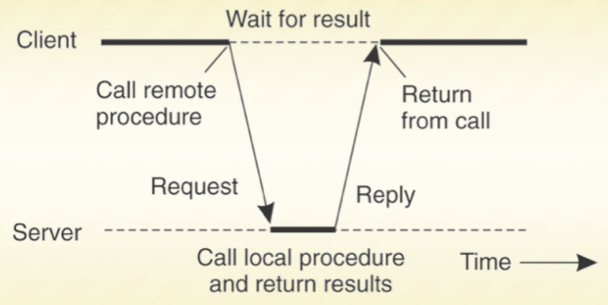
Conventional Procedure Call



- (a) Parameter passing in a local procedure call: the stack before the call to read
- (b) The stack while the called procedure – read(fd, buf, nbytes) - is active.

8

Client and Server Stubs



- Principle of RPC between a client and server program.

Remote Procedure Calls (1)

- A remote procedure call occurs in the following steps:

1. The client procedure calls the client stub in the normal way.
2. The client stub builds a message and calls the local operating system.
3. The client's OS sends the message to the remote OS.
4. The remote OS gives the message to the server stub.
5. The server stub unpacks the parameters and calls the server.

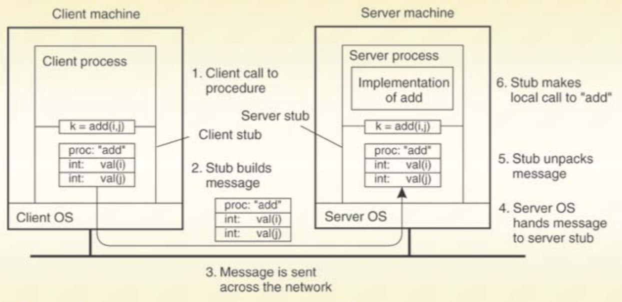
Continued …

Remote Procedure Calls (2)

- A remote procedure call occurs in the following steps (continued):

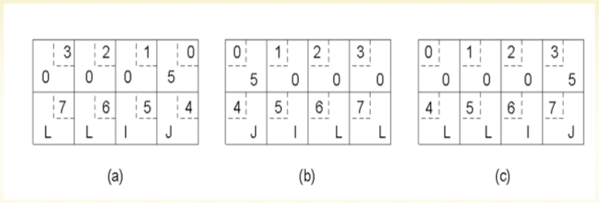
6. The server does the work and returns the result to the stub.
7. The server stub packs it in a message and calls its local OS.
8. The server's OS sends the message to the client's OS.
9. The client's OS gives the message to the client stub.
10. The stub unpacks the result and returns to the client.

Passing Value Parameters (1)



- The steps involved in a doing a remote computation through RPC.

Passing Value Parameters (2)



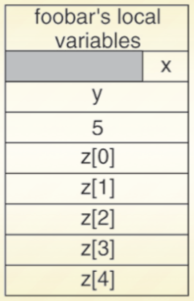
a) Original message on the Pentium
b) The message after receipt on the SPARC
c) The message after being inverted. The little numbers in boxes indicate the address of each byte

Parameter Specification and Stub Generation

- (a) A procedure
- (b) The corresponding message.

```
foobar( char x; float y; int z[5] )
{
    ....
}
```

(a)

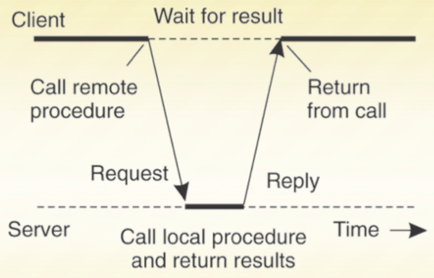


foobar's local variables	
	x
y	
5	
z[0]	
z[1]	
z[2]	
z[3]	
z[4]	

(b)

14

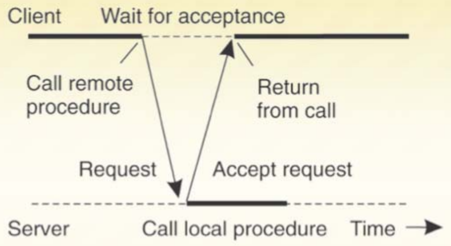
Asynchronous RPC (1)



- The interaction between client and server in a traditional RPC.

15

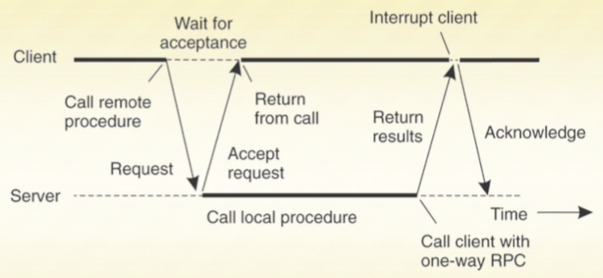
Asynchronous RPC (2)



- The interaction using asynchronous RPC.

16

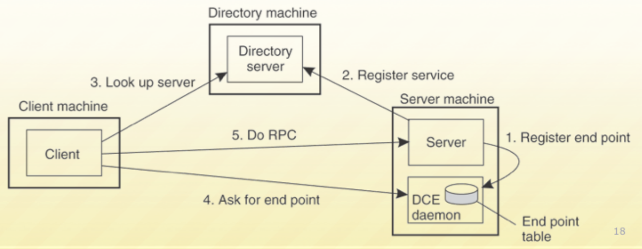
Asynchronous RPC (3)



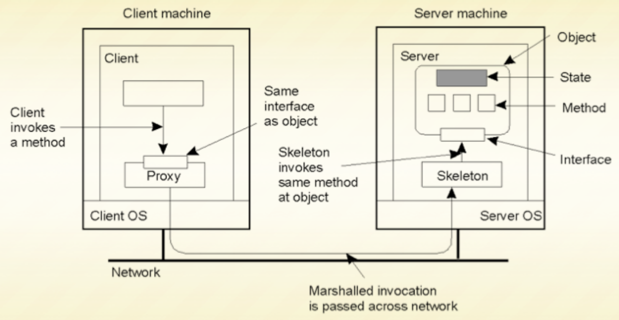
- A client and server interacting through two asynchronous RPCs.

Binding a Client to a Server

- Registration of a server makes it possible for a client to locate the server and bind to it
- Server location is done in two steps:
  - Locate the server's machine.
  - Locate the server on that machine.



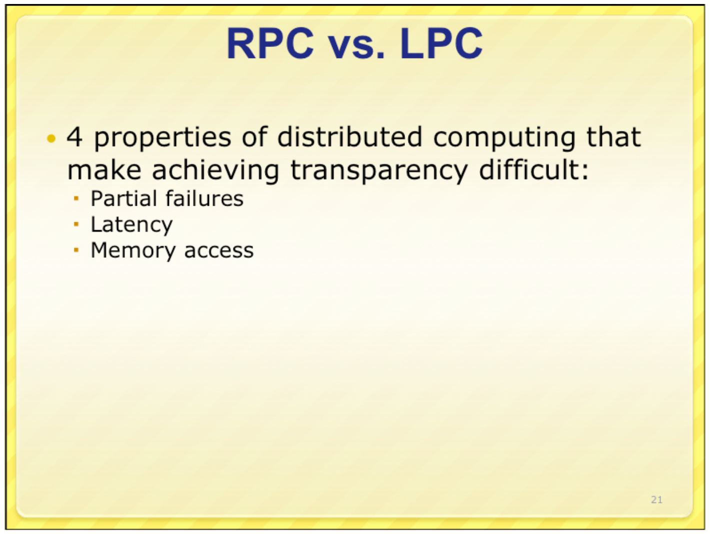
Distributed Objects



- Common organization of a remote object with client-side proxy.

Today's Lecture

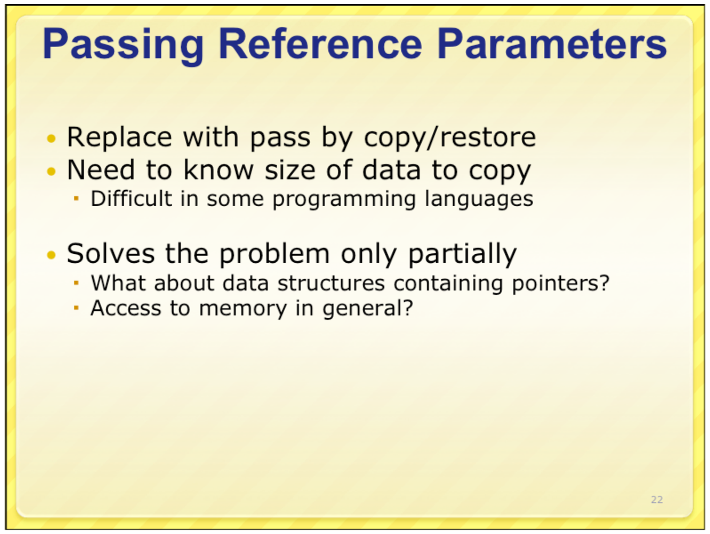
- RPC overview
- RPC transparency
- RPC implementation issues



RPC vs. LPC

- 4 properties of distributed computing that make achieving transparency difficult:
  - Partial failures
  - Latency
  - Memory access

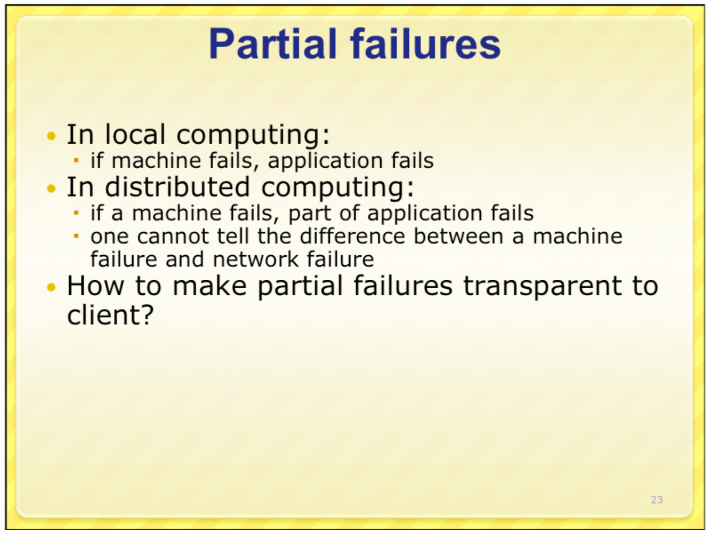
21



Passing Reference Parameters

- Replace with pass by copy/restore
- Need to know size of data to copy
  - Difficult in some programming languages
- Solves the problem only partially
  - What about data structures containing pointers?
  - Access to memory in general?

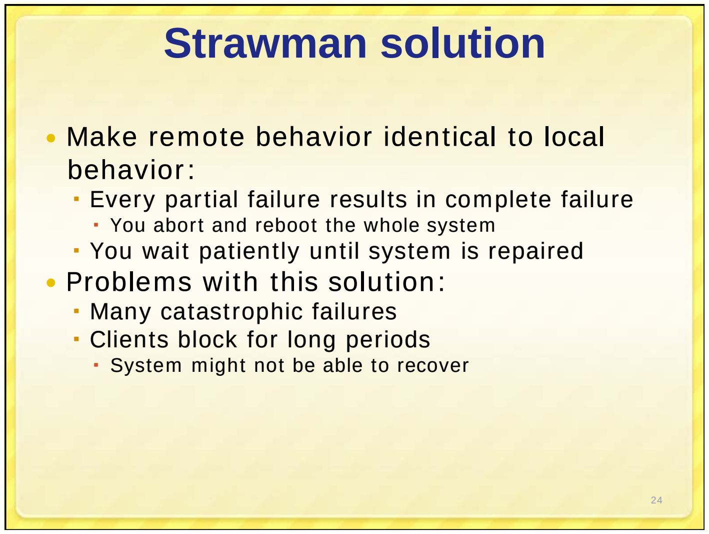
22



Partial failures

- In local computing:
  - if machine fails, application fails
- In distributed computing:
  - if a machine fails, part of application fails
  - one cannot tell the difference between a machine failure and network failure
- How to make partial failures transparent to client?

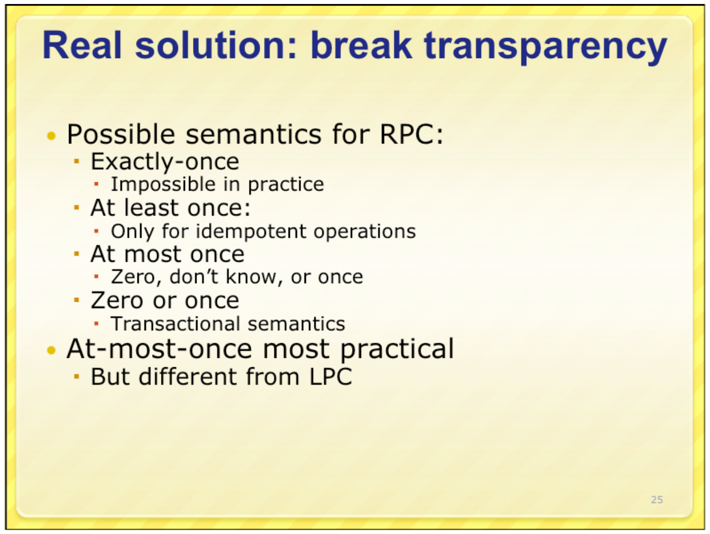
23



Strawman solution

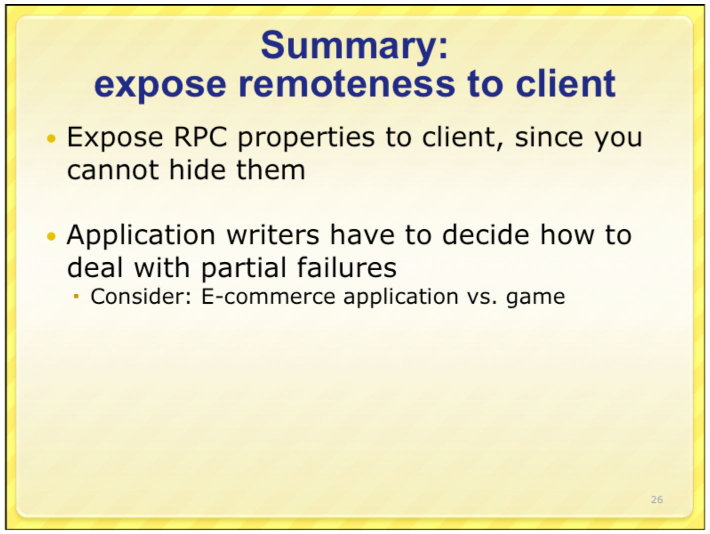
- Make remote behavior identical to local behavior:
  - Every partial failure results in complete failure
    - You abort and reboot the whole system
  - You wait patiently until system is repaired
- Problems with this solution:
  - Many catastrophic failures
  - Clients block for long periods
    - System might not be able to recover

24



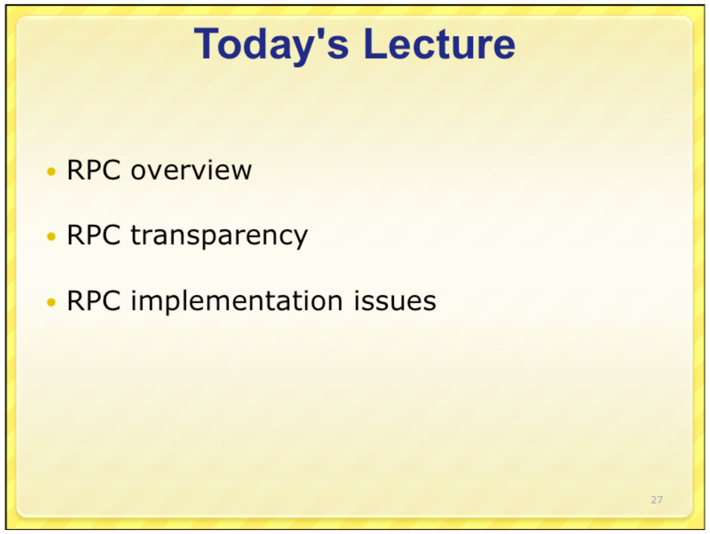
Real solution: break transparency

- Possible semantics for RPC:
  - Exactly-once
    - Impossible in practice
  - At least once:
    - Only for idempotent operations
  - At most once
    - Zero, don't know, or once
  - Zero or once
    - Transactional semantics
- At-most-once most practical
  - But different from LPC



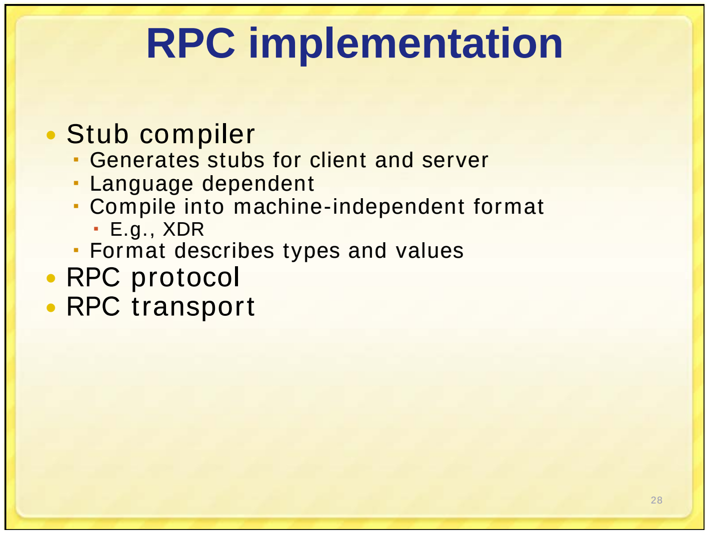
Summary: expose remoteness to client

- Expose RPC properties to client, since you cannot hide them
- Application writers have to decide how to deal with partial failures
  - Consider: E-commerce application vs. game



Today's Lecture

- RPC overview
- RPC transparency
- RPC implementation issues

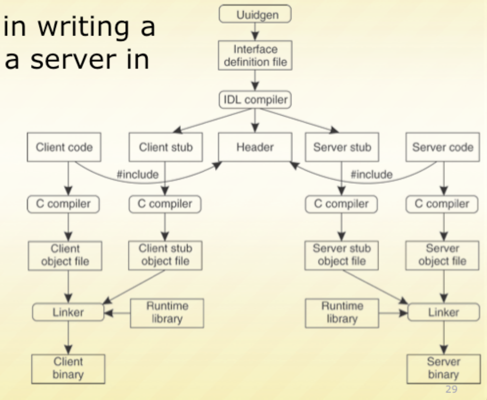


RPC implementation

- Stub compiler
  - Generates stubs for client and server
  - Language dependent
  - Compile into machine-independent format
    - E.g., XDR
  - Format describes types and values
- RPC protocol
- RPC transport

Writing a Client and a Server (1)

- The steps in writing a client and a server in DCE RPC.



29

Writing a Client and a Server (2)

- Three files output by the IDL compiler:
- A header file (e.g., interface.h, in C terms).
- The client stub.
- The server stub.

30

RPC protocol

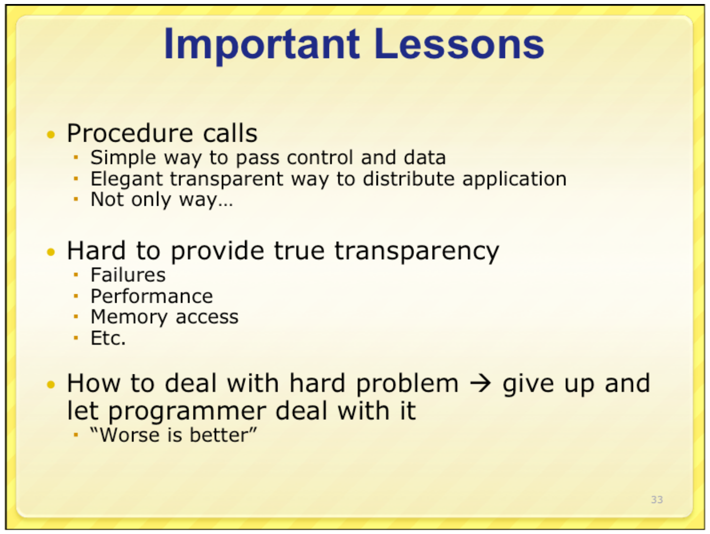
- Guarantee at-most-once semantics by tagging requests and response with a nonce
- RPC request header:
  - Request nonce
  - Service Identifier
  - Call identifier
- Protocol:
  - Client resends after time out
  - Server maintains table of nonces and replies

31

RPC transport

- Use reliable transport layer
  - Flow control
  - Congestion control
  - Reliable message transfer
- Combine RPC and transport protocol
  - Reduce number of messages
    - RPC response can also function as acknowledgement for message transport protocol

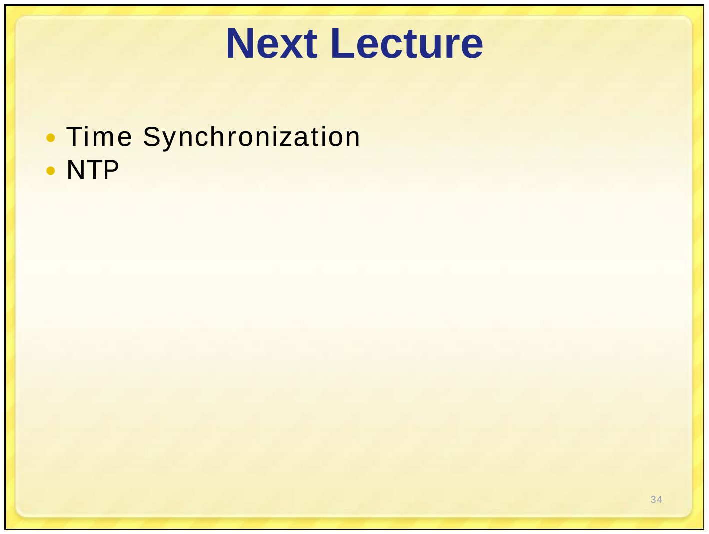
32



Important Lessons

- Procedure calls
  - Simple way to pass control and data
  - Elegant transparent way to distribute application
  - Not only way…
- Hard to provide true transparency
  - Failures
  - Performance
  - Memory access
  - Etc.
- How to deal with hard problem → give up and let programmer deal with it
  - "Worse is better"

33



Next Lecture

- Time Synchronization
- NTP

34