CS 138: Communication II

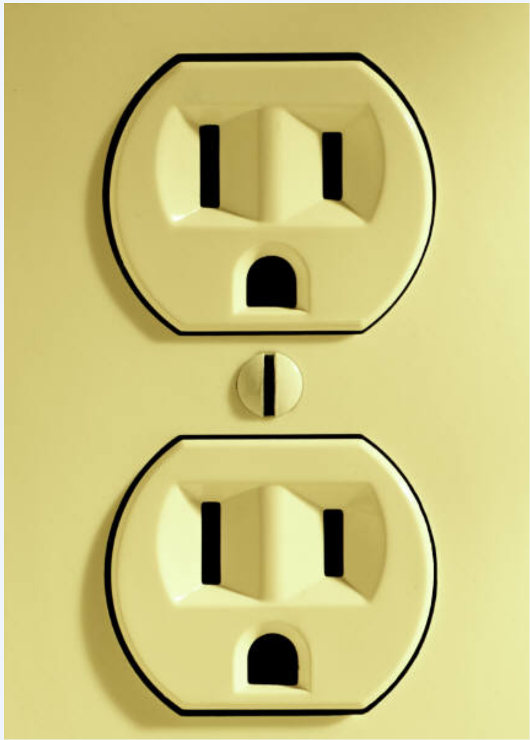
Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Today's Lecture

- Sockets
- RPC
  – Overview
  – Challenges
  – Examples

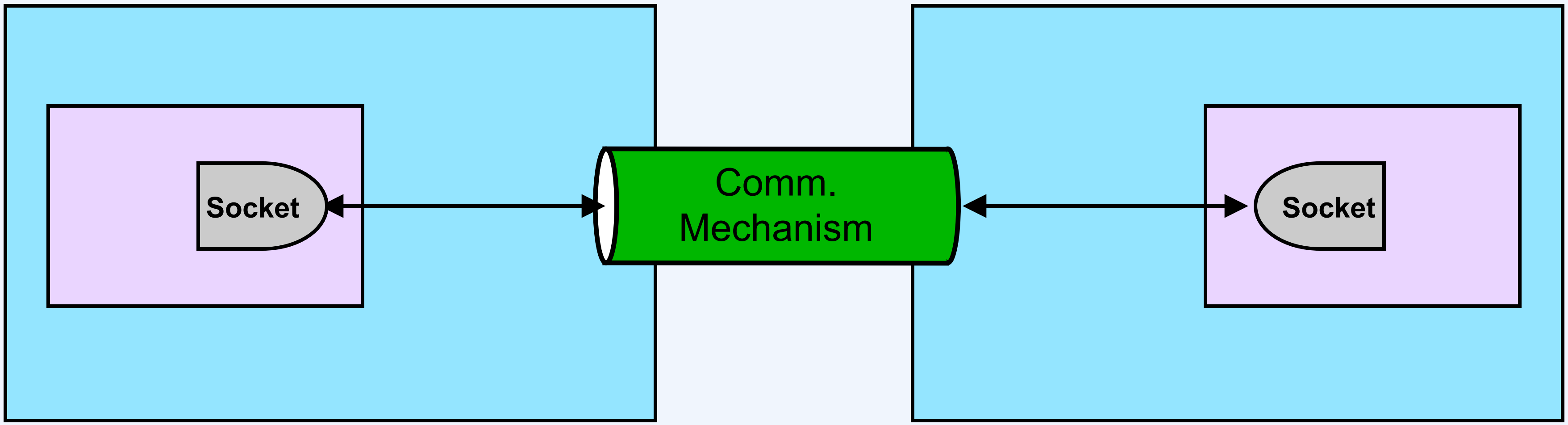
Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Sockets



Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Sockets



Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Socket Parameters

- Styles of communication:
  - stream: reliable, two-way byte streams
  - datagram: unreliable, two-way record-oriented
  - sequenced packet: reliable, two-way record-oriented
  - etc.
- Communication domains
  - UNIX
    - endpoints (sockets) named with file-system pathnames
    - supports stream and datagram
  - Internet
    - endpoints named with IP addresses
    - supports stream and datagram
  - others
- Protocols
  - the means for communicating data
  - e.g., TCP/IP, UDP/IP

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.

Some content from David Andersen.

Limitations

- Strictly an interface to the transport layer
  - (or lower)
- Reliability
  - if the receiving machine is temporarily not available, will sent data eventually reach it?
  - how is the sender notified if sent data does not arrive at destination machine?
  - how is the sender notified if sent data does not arrive at destination application?

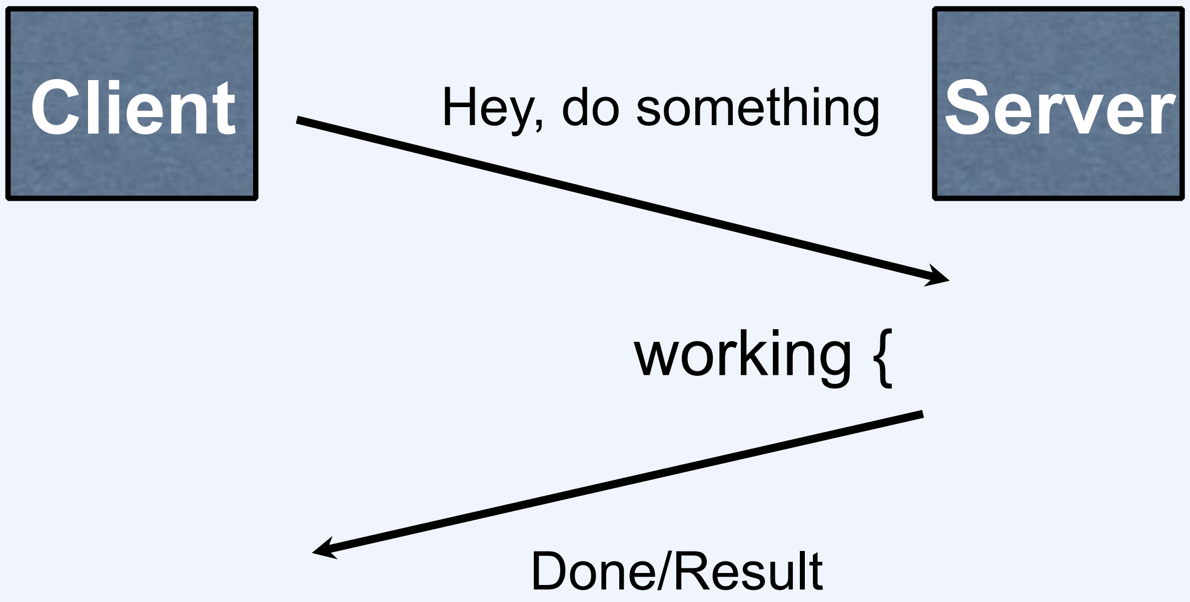
Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Writing Distributed Programs

- Concerns
  – transparency
  – portability
  – interoperability
- Solutions
  – RPC
  – RMI

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Common communication pattern



Writing it by hand...

- **eg, if you had to write a, say, password cracker**

```
struct foomsg {
  u_int32_t len;
}

send_foo(char *contents) {
   int msglen = sizeof(struct foomsg) + strlen(contents);
   char buf = malloc(msglen);
   struct foomsg *fm = (struct foomsg *)buf;
   fm->len = htonl(strlen(contents));
   memcpy(buf + sizeof(struct foomsg),
          contents,
          strlen(contents));
   write(outsock, buf, msglen);
}
```

Then wait for response, etc.

Content from David Andersen

RPC

- A type of client/server communication
- Attempts to make remote procedure calls look like local ones

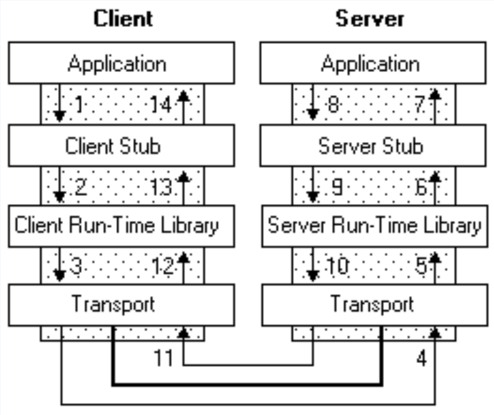


figure from Microsoft MSDN

```
{ ...
   foo()
}
void foo() {
  invoke_remote_foo()
}
```

Go Example

- **Need some setup in advance of this but…**

```
// Synchronous call
args := &server.Args{7,8}
var reply int
err = client.Call("Arith.Multiply", args,
&reply)
if err != nil {
      log.Fatal("arith error:", err)
}
fmt.Printf("Arith: %d*%d=%d", args.A, args.B,
reply)
```

RPC Goals

- Ease of programming
- Hide complexity
- Automates task of implementing distributed computation
- Familiar model for programmers (just make a function call)

Historical note: Seems obvious in retrospect, but RPC was only invented in the ‘80s. See Birrell & Nelson, “Implementing Remote Procedure Call” ... or Bruce Nelson, Ph.D. Thesis, Carnegie Mellon University: Remote Procedure Call., 1981 :)

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

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.

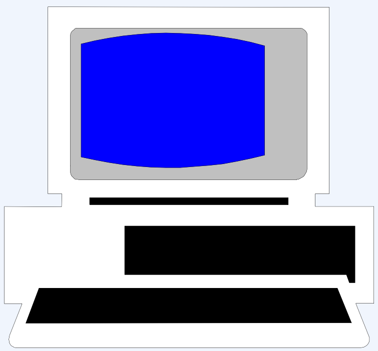
Some content from David Andersen.

But it’s not always simple

- Calling and called procedures run on different machines, with different address spaces
  - And perhaps different environments .. or operating systems ..
- Must convert to local representation of data
- Machines and network can fail

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Local Procedure Calls



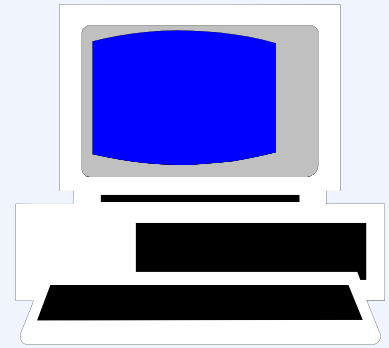
```
// Client code
    . . .
result = procedure(arg1, arg2);
    . . .
```

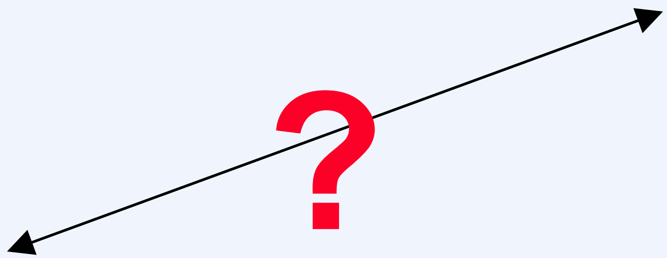
```
// Server code
result_t procedure(a1_t arg1, a2_t arg2) {
    . . .
    return(result);
}
```

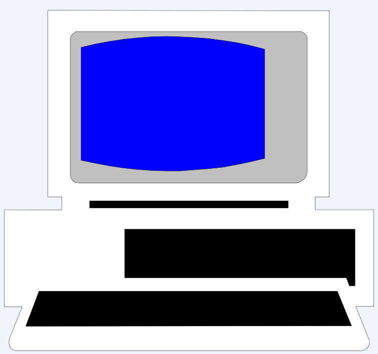

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Remote Procedure Calls (1)

```
// Client code
      . . .
result = procedure(arg1, arg2);
      . . .
```





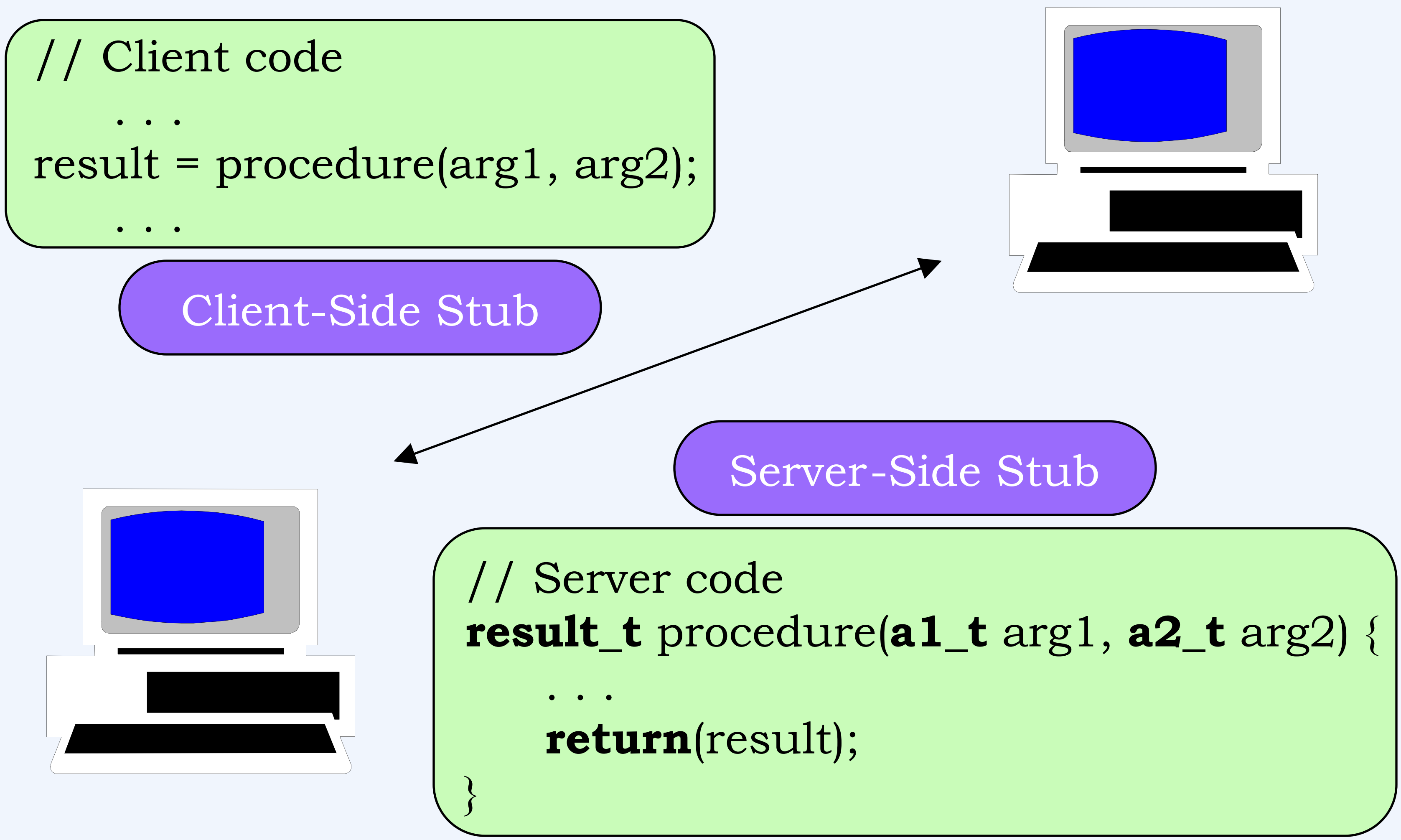


```
// Server code
result_t procedure(a1_t arg1, a2_t arg2) {
      . . .
      return(result);
}
```


Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.

Some content from David Andersen.

Remote Procedure Calls (2)



Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

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 marshals results and sends reply

“stubs” and IDLs

- RPC stubs do the work of marshaling and unmarshaling data
- But how do they know how to do it?
- Typically: Write a description of the function signature using an IDL -- interface description language.
  - Lots of these. Some look like C, some look like XML, ... details don’t matter much.

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)

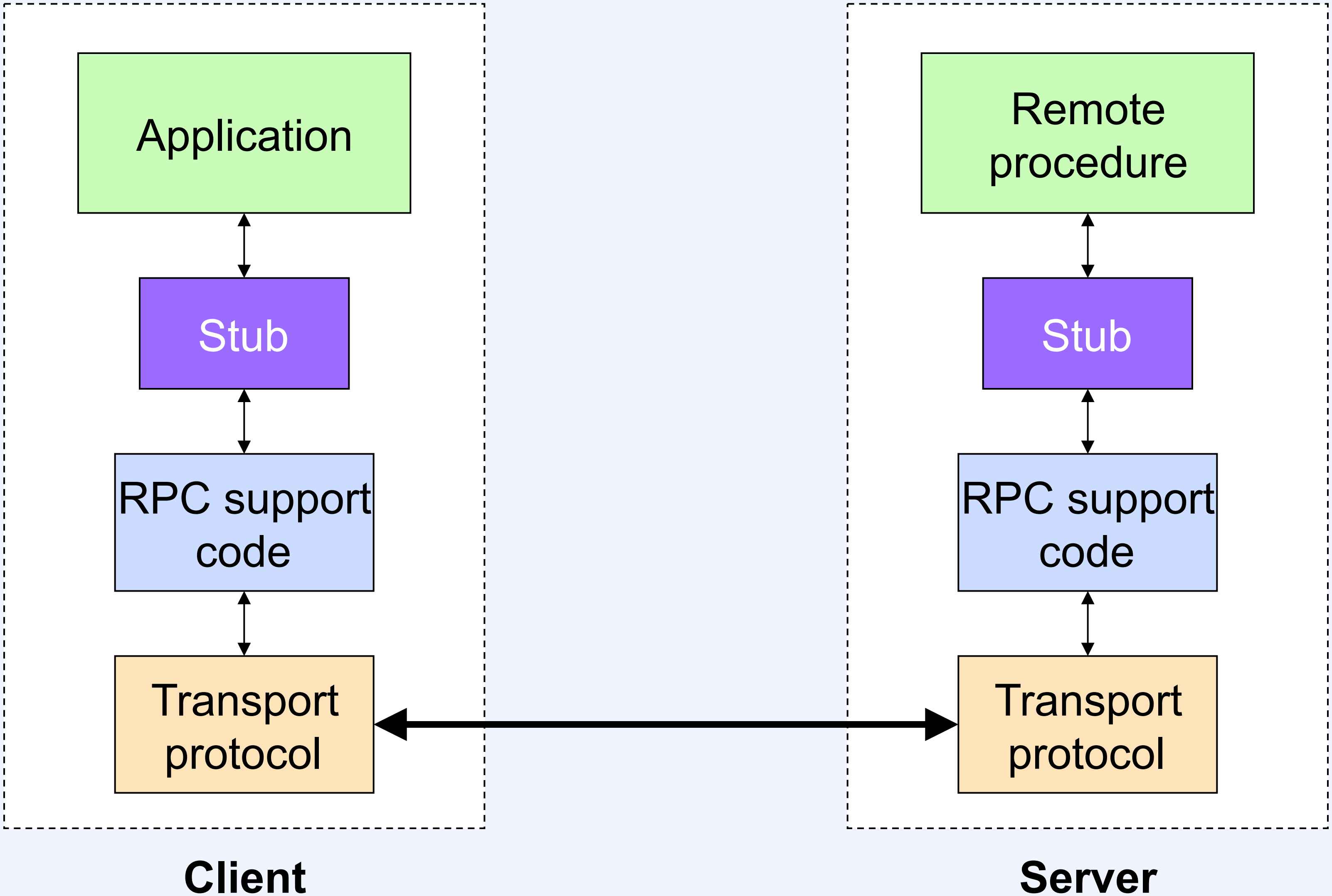
- A remote procedure call occurs in the following steps (continued):

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.

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.

Some content from David Andersen.

Block Diagram



Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.

Some content from David Andersen.

Today's Lecture

- Sockets
- RPC
  - Overview
  - Challenges
  - Examples

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Local vs. Distributed

- Latency
- Memory access
- Partial failure
- Concurrency

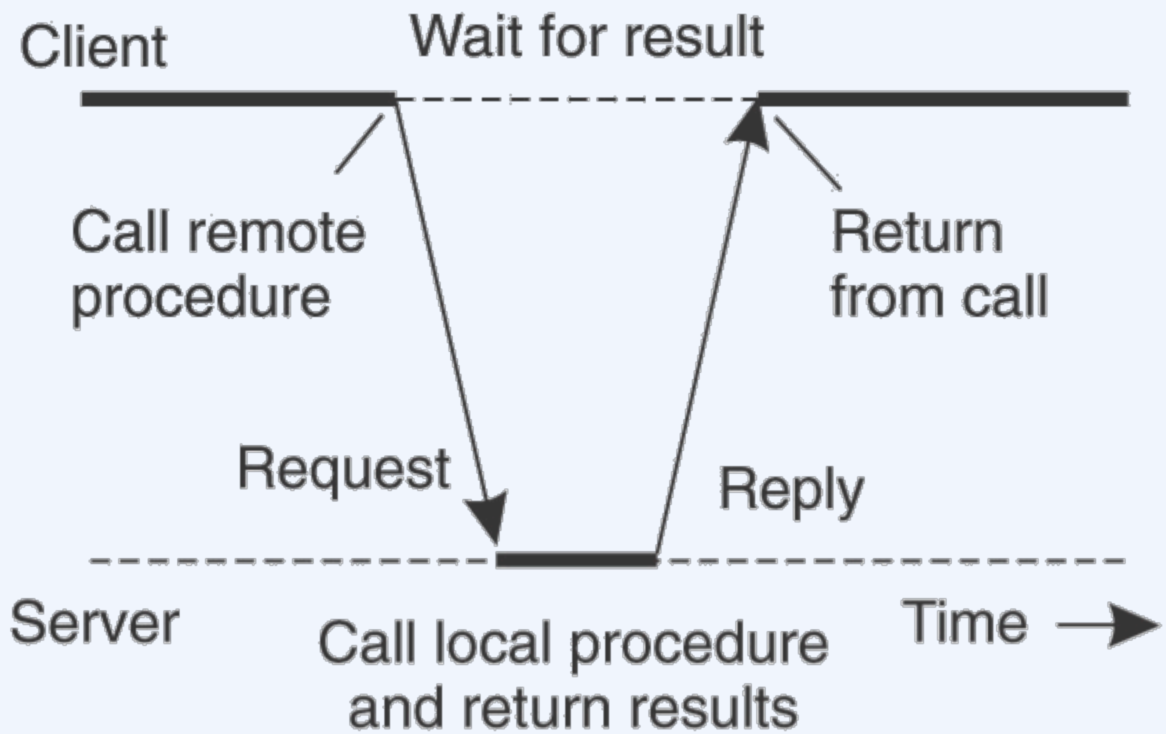
Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Latency

- **Remote invocation of objects takes much longer than local invocation**
  - **can this be ignored at first and dealt with later?**

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

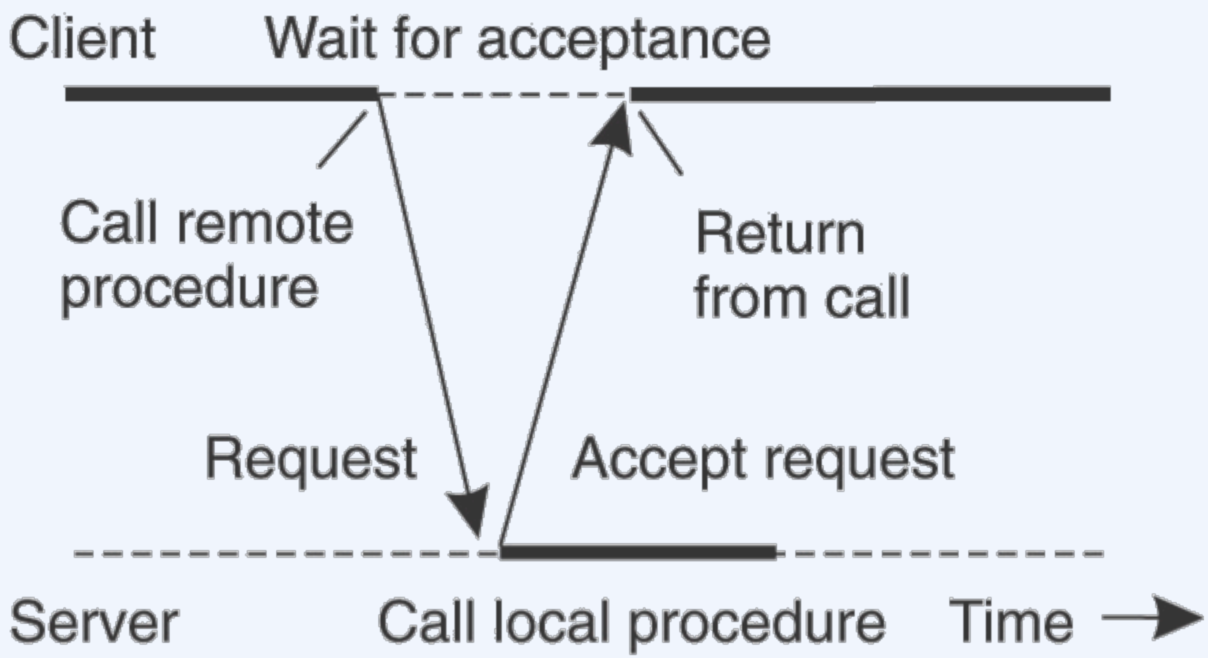
Synchronous RPC



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

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

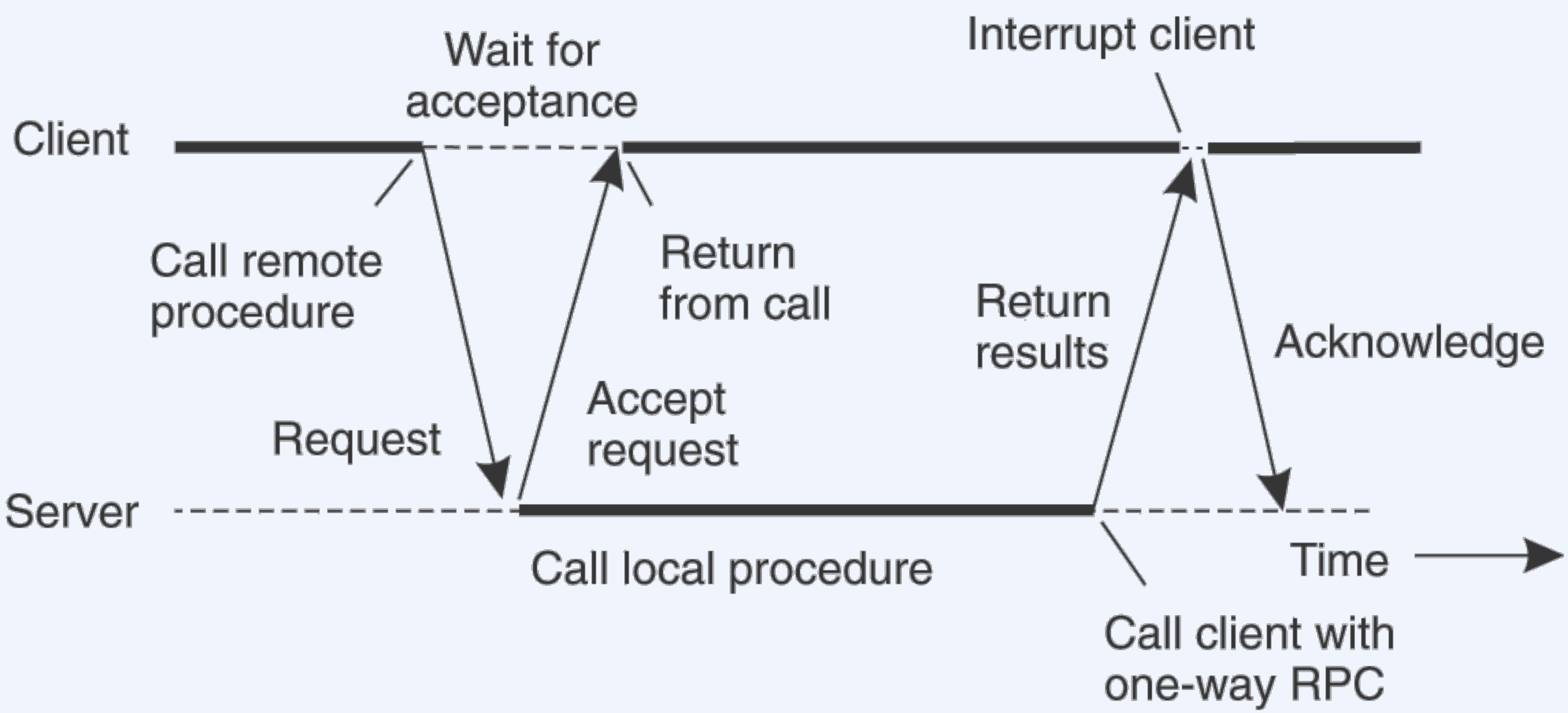
Asynchronous RPC (1)



- **The interaction using asynchronous RPC.**

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Asynchronous RPC (2)



- **A client and server interacting through two asynchronous RPCs.**

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Concurrency

- Distributed programs have the same concurrency issues as multithreaded programs
  - false
    - all threads are under control of a common OS
    - synchronization is easy

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

RPC failures

- Request from cli → srv lost
- Reply from srv → cli lost
- Server crashes after receiving request
- Client crashes after sending request

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.

Some content from David Andersen.

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?

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.

Some content from David Andersen.

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

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

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

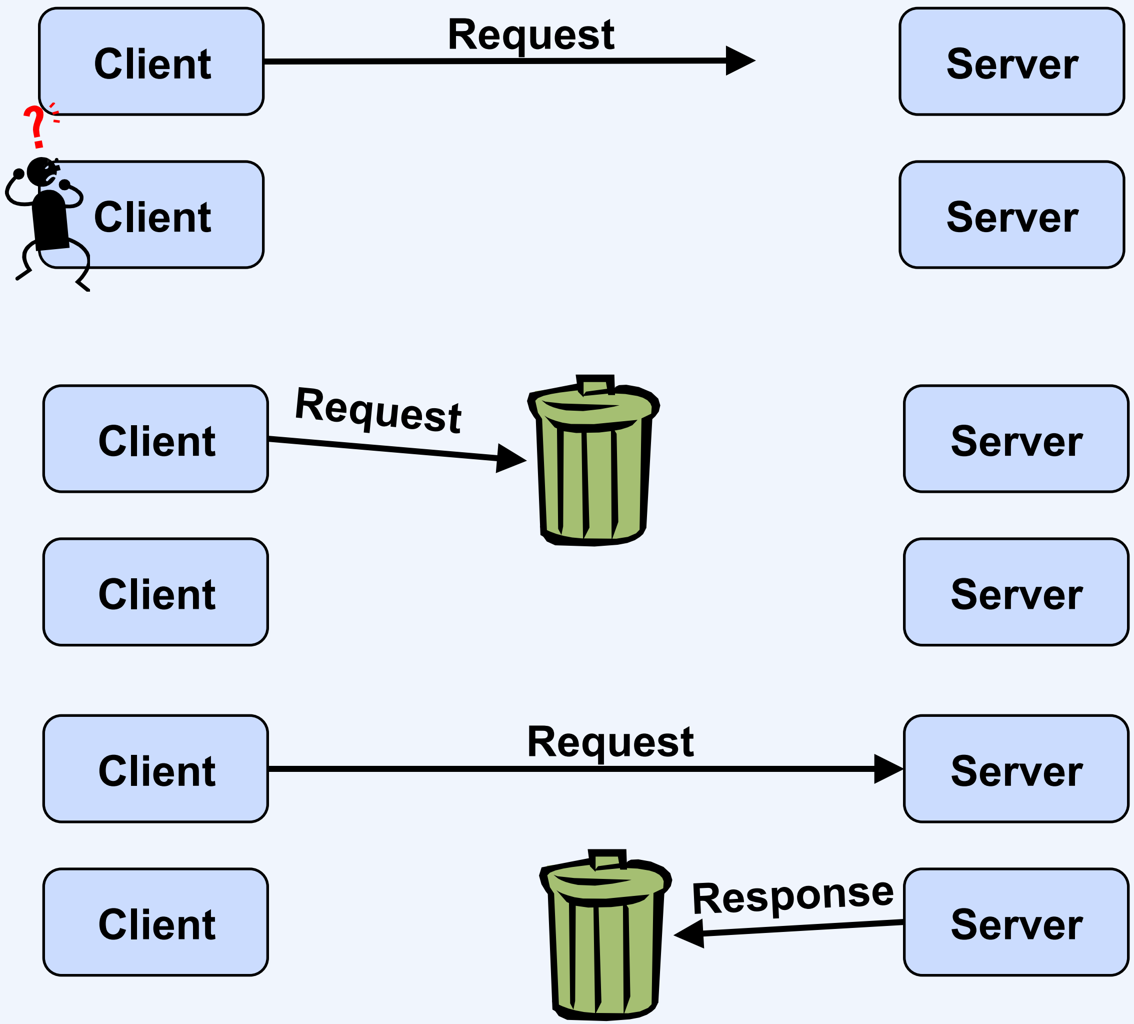
Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Real solution: break transparency

- <u>At-least-once</u>: Just keep retrying on client side until you get a response.
  – Server just processes requests as normal, doesn't remember anything. Simple!
- <u>At-most-once</u>: Server might get same request twice...
  – Must re-send *previous* reply and not process request (implies: keep cache of handled requests/responses)
  – Must be able to identify requests
  – Strawman: remember *all* RPC IDs handled. -> Ugh! Requires infinite memory.
  – Real: Keep sliding window of valid RPC IDs, have client number them sequentially.

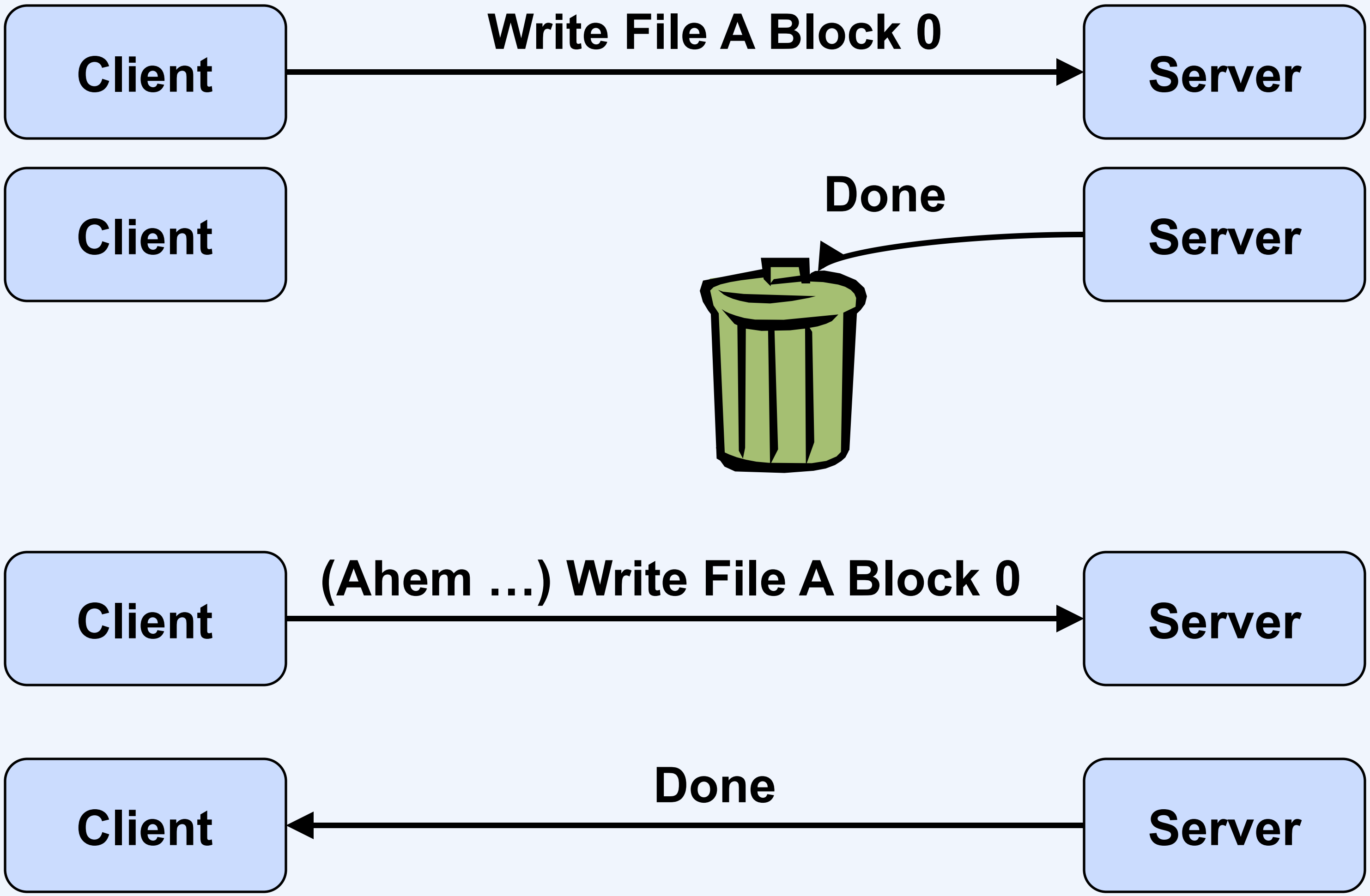
Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Uncertainty



Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

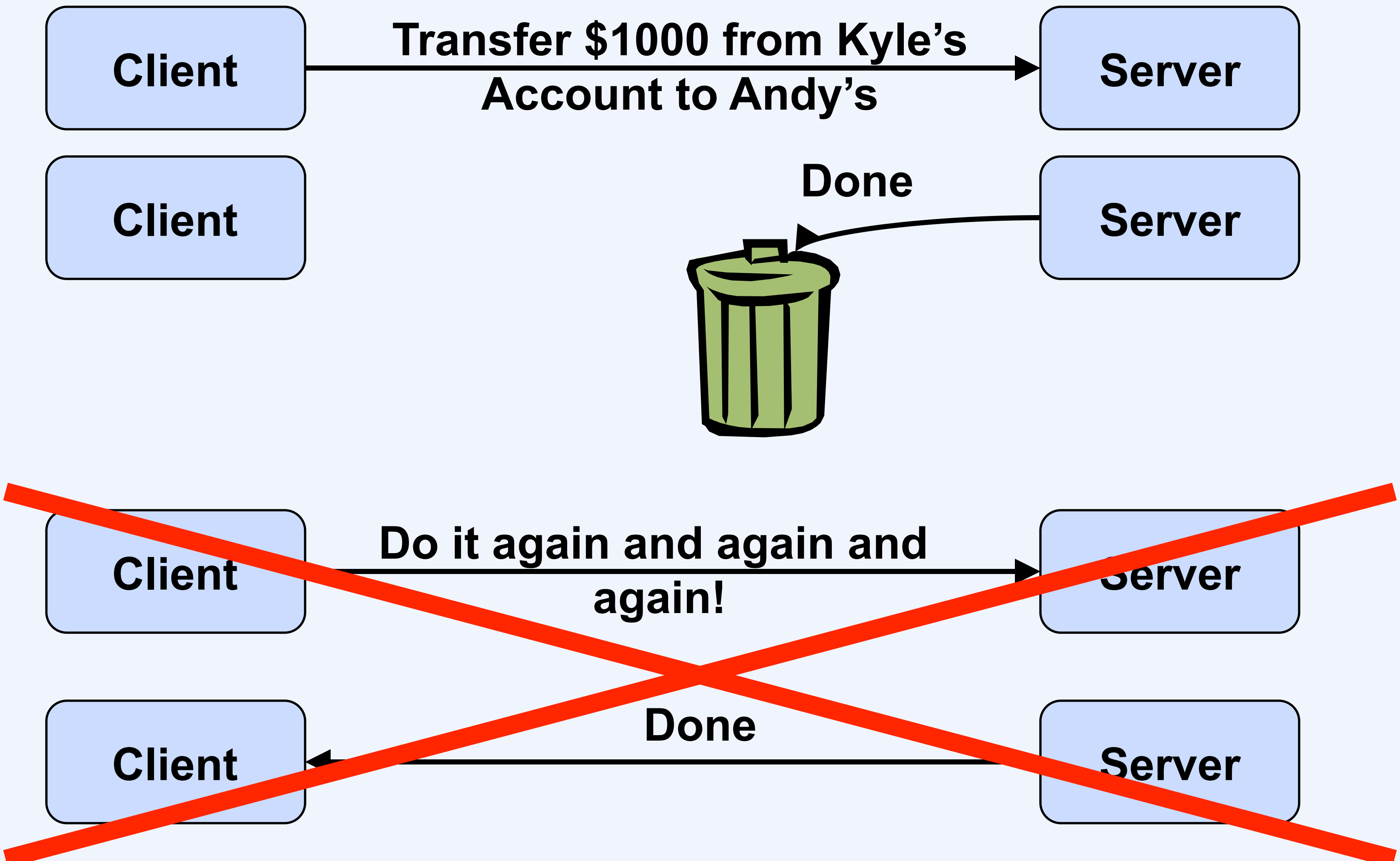
Idempotent Procedures



At-least-once semantics

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

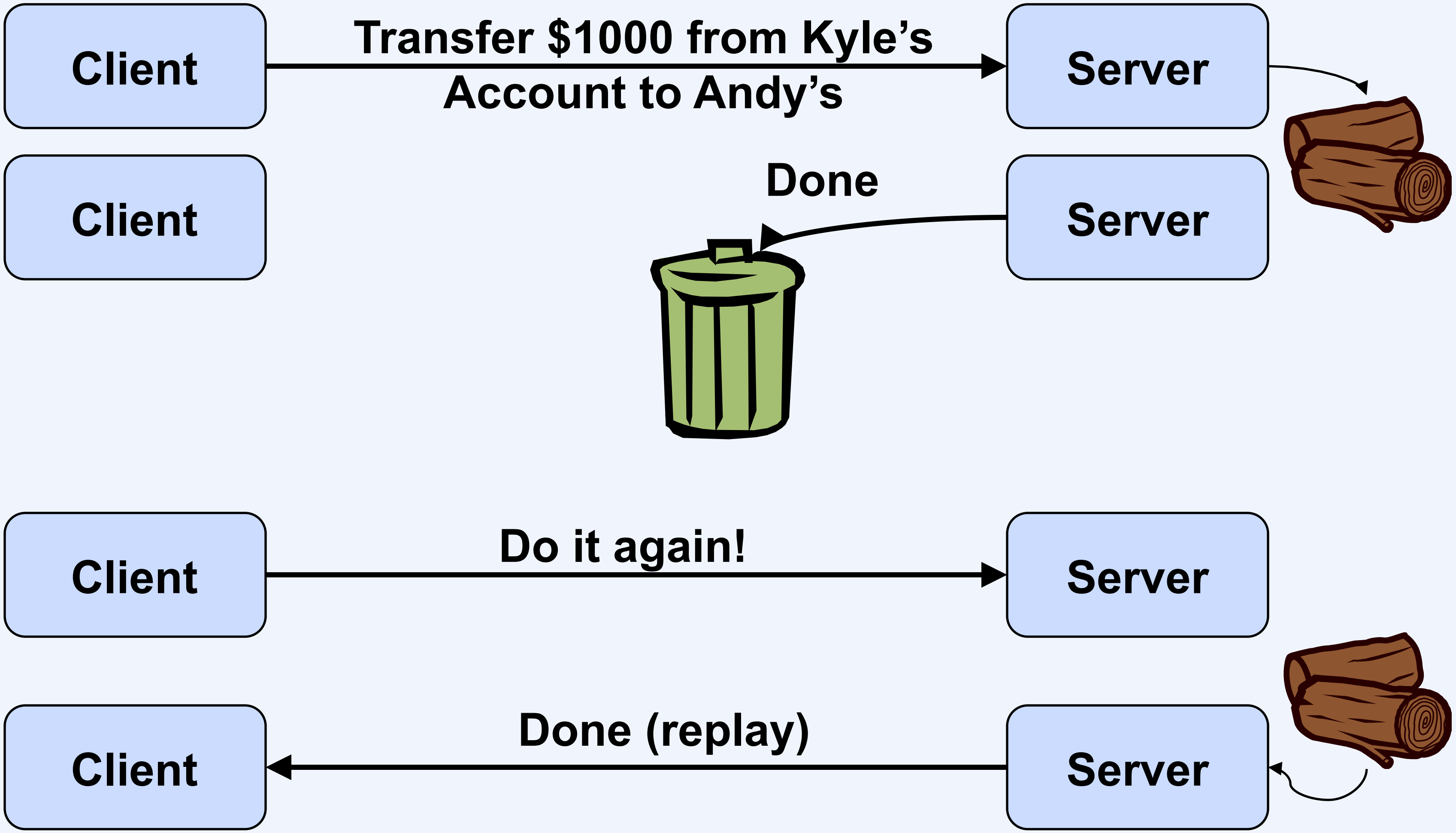
Non-Idempotent Procedures



At-most-once semantics

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

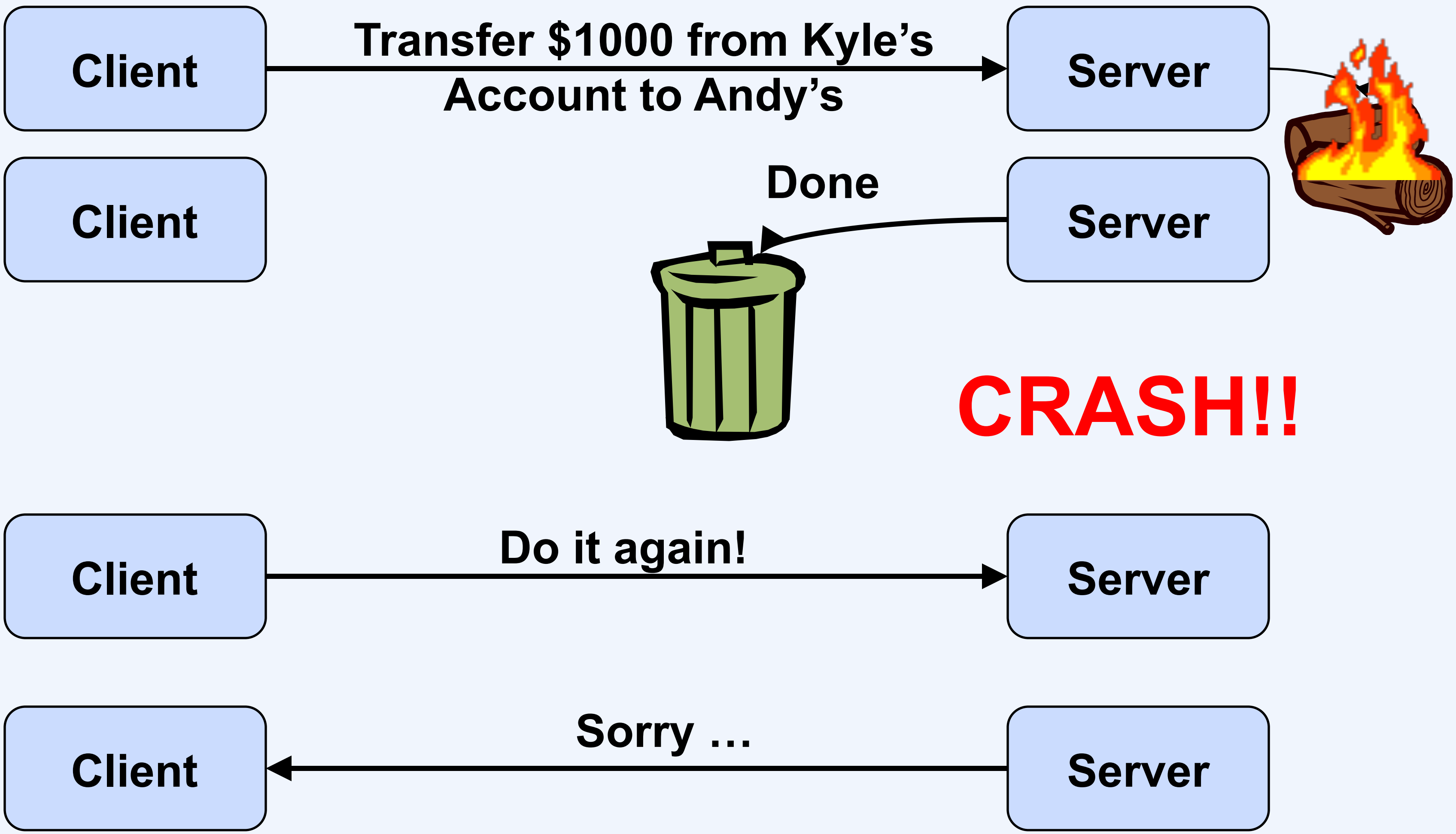
Maintaining History



At-most-once semantics

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

No History



At-most-once semantics

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Coping

Fault tolerance measures			Invocation Semantics
Retransmit request message	Duplicate filtering	Re-execute procedure or retransmit reply	
No	Not applicable	Not applicable	Maybe
Yes	No	Re-execute procedure	At-least-once
Yes	Yes	Retransmit reply	At-most-once

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Exactly-Once?

- Sorry - no can do *in general*.
- Imagine that message triggers an external physical thing (say, a robot fires a nerf dart at the professor)
- The robot could crash immediately before or after firing and lose its state. Don't know which one happened. Can, however, make this window very small.

Memory Access

- Pointers work locally
- Can they be made to work remotely?
  - yes …
  - but don't use a remote pointer thinking it's just like a local pointer

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.

Some content from David Andersen.

Implementation Concerns

- As a general library, performance is often a big concern for RPC systems
- Major source of overhead: copies and marshaling/unmarshaling overhead
- Zero-copy tricks:
  - Representation: Send on the wire in native format and indicate that format with a bit/byte beforehand. What does this do? Think about sending uint32 between two little-endian machines (DEC RPC)
  - Scatter-gather writes (writev() and friends)

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

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

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"

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Today's Lecture

- Sockets
- RPC
  - Overview
  - Challenges
  - Examples

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Two styles of RPC implementation

- Shallow integration. Must use lots of library calls to set things up:
  - How to format data
  - Registering which functions are available and how they are invoked.
- Deep integration.
  - Data formatting done based on type declarations
  - (Almost) all public methods of object are registered.
- Go is the latter.

Example: Sun XDR (RFC 4506)

- *External Data Representation* for SunRPC
- Types: most of C types
- No tags (except for array lengths)
  - Code needs to know structure of message
- Usage:
  - Create a program description file (.x)
  - Run rpcgen program
  - Include generated .h files, use stub functions
- Very C/C++ oriented
  - Although encoders/decoders exist for other languages

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Example

```
typedef struct {
    int         comp1;
    double      comp2;
    long long   comp3[6];
    char        *annotation;
} value_t;

typedef struct {
    value_t     element;
    value_t     *next;
} list_t;

char add(int key, value_t value);
char remove(int key, value_t value);
list_t query(int key);
```


Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

A Specification

```
typedef struct value {
        int                 comp1;
        double              comp2;
        hyper               comp3[6];
        string              annotation<255>;
} value_t;

typedef struct list {
        value_t       element;
        struct list   *next;
} list_t;

program DB {
    version DBVERS {
            bool add(int key, value_t value) = 1;
            bool remove(int key, value_t value) = 2;
            list_t query(int key) = 3;
    } = 1;
} = 0x2000000A;
```

- **Rpcgen generates marshalling/unmarshalling code, stub functions, you fill out the actual code**

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

XDR Primitive Types

- Integer
- Unsigned integer
- Boolean
- Hyper integer
- Unsigned hyper integer
- Fixed-length opaque data
- Variable-length opaque data
- String

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

XDR Structured Types

- Fixed-length array
- Variable-length array
- Discriminated union
- Linked lists

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Generated Header File

```
struct value {
    int comp1;
    double comp2;
    int64_t comp3[6];
    char *annotation;
};
typedef struct value value;

typedef value value_t;

struct list {
    value_t element;
    struct list *next;
};
typedef struct list list;
```

```
typedef list list_t;

struct add_1_argument {
    int key;
    value_t value;
};
typedef struct add_1_argument
        add_1_argument;

struct remove_1_argument {
    int key;
    value_t value;
};
typedef struct remove_1_argument
        remove_1_argument;
```


Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Placing Calls

```
    result_1 = add_1(add_1_key, add_1_value, clnt);
    if (result_1 == (bool_t *) NULL) {
        clnt_perror (clnt, "call failed");
    }
    result_2 = remove_1(remove_1_key, remove_1_value, clnt);
    if (result_2 == (bool_t *) NULL) {
        clnt_perror (clnt, "call failed");
    }
    result_3 = query_1(query_1_key, clnt);
    if (result_3 == (list_t *) NULL) {
        clnt_perror (clnt, "call failed");
    }
    clnt_destroy (clnt);
}
```


Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

DCE RPC

- Designed by Apollo and Digital in the 1980s
  - both companies later absorbed by HP
- Does everything ONC RPC can do, and more
- Basis for Microsoft RPC

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Same Example …

```
typedef struct {
    double      comp1;
    int         comp2;
    long long       comp3;
    char        *annotation;
} value_t;

char add(int key, value_t value);
char remove(int key, value_t value);
int query(int key, int number, value_t values[ ]);
```


Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.

Some content from David Andersen.

An Interface Specification

```
interface db {
  typedef struct {
    double comp1;
    long   comp2;
    hyper  comp3;
    [string, ptr]
      ISO_LATIN_1
        *annotation;
  } value_t;

  boolean add(
    [in] long     key,
    [in] value_t     value
  );

  boolean remove(
    [in] long     key,
    [in] value_t     value
  );

  long query(
    [in] long     key,
    [in] long     number,
    [out, size_is(number)]
      value_t  values[ ]
  );
}
```


Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

An Interface Specification
(notes continued)

```
interface db {
  typedef struct {
    double comp1;
    long   comp2;
    hyper  comp3;
    [string, ptr]
      ISO_LATIN_1
        *annotation;
  } value_t;

  boolean add(
    [in] long     key,
    [in] value_t  value
  );

  boolean remove(
    [in] long     key,
    [in] value_t  value
  );

  long query(
    [in] long     key,
    [in] long     number,
    [out, size_is(number)]
      value_t  values[ ]
  );
}
```


Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.

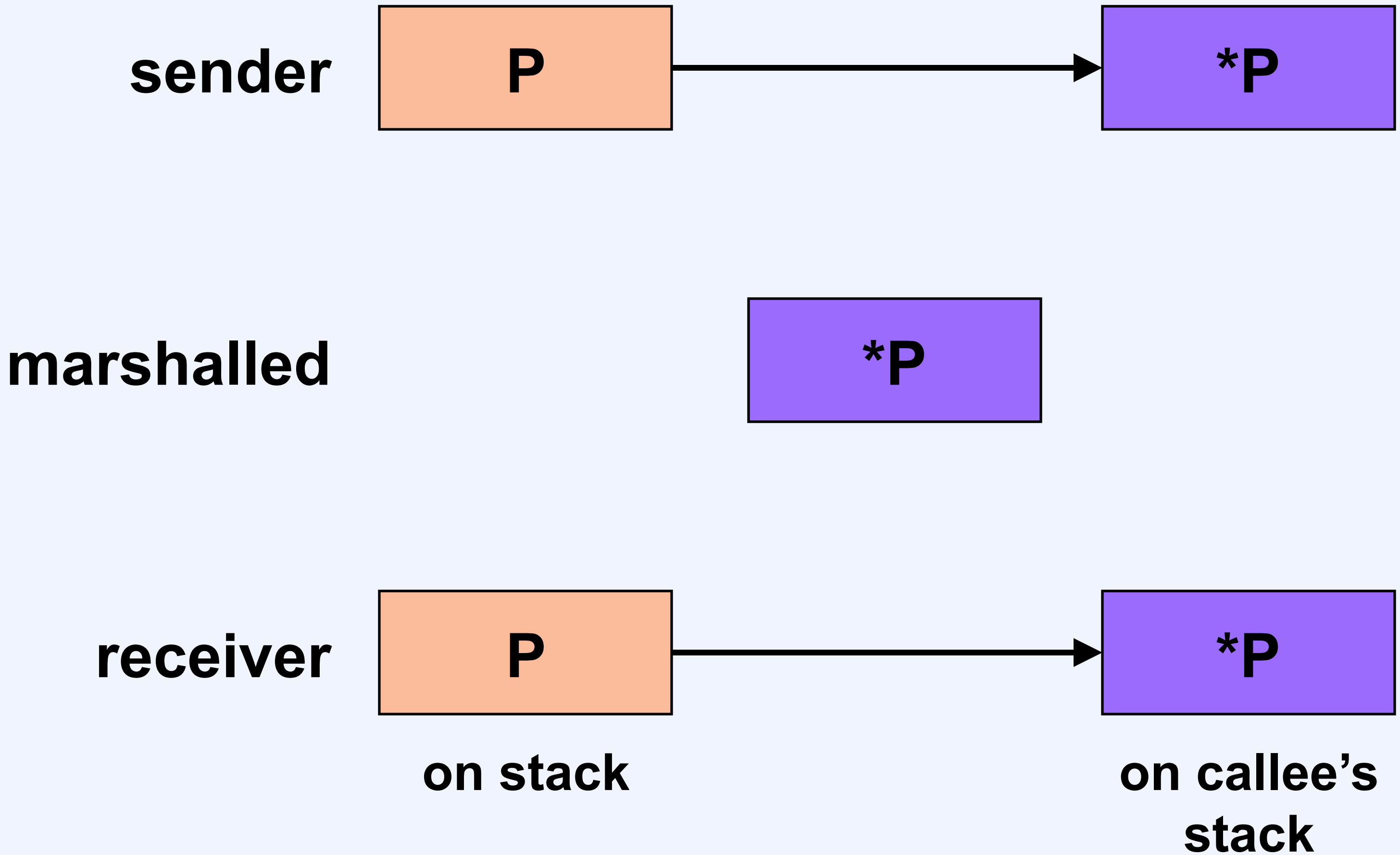
Some content from David Andersen.

Representing an Array

Length	Item 1	Item 2	…	Item n

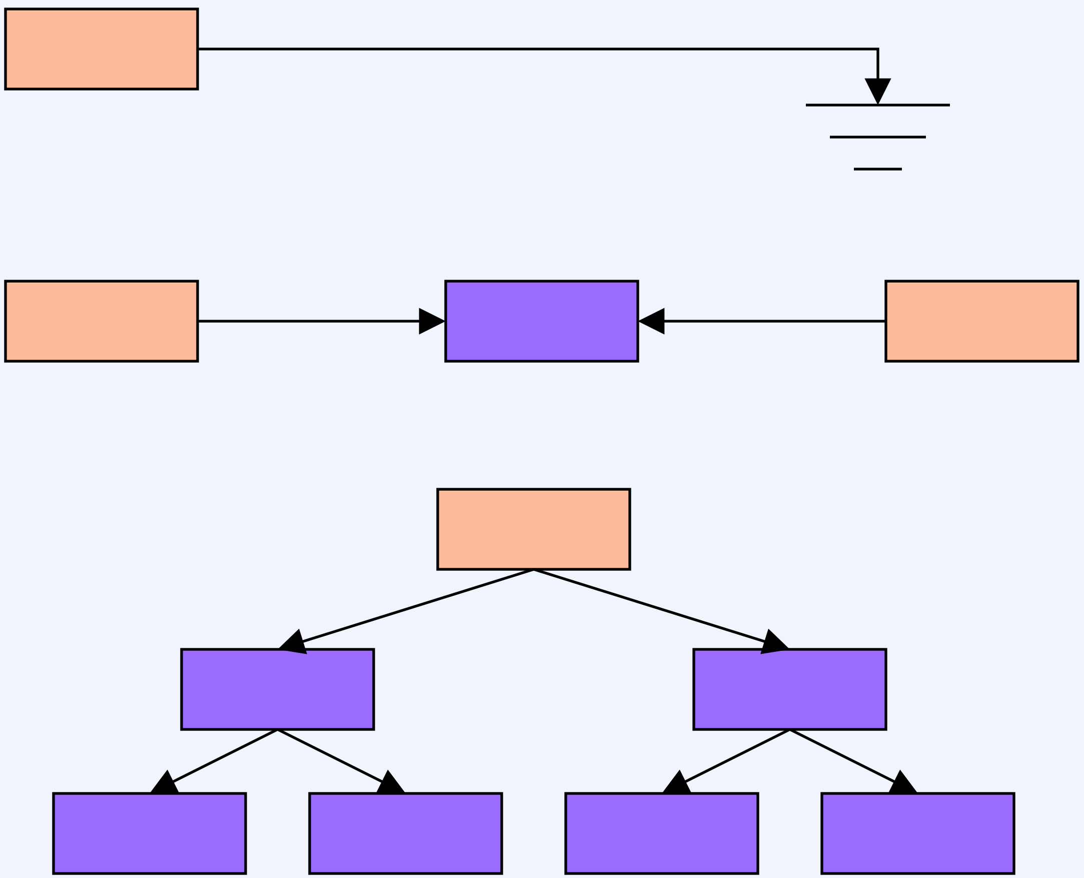
Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Representing Pointers



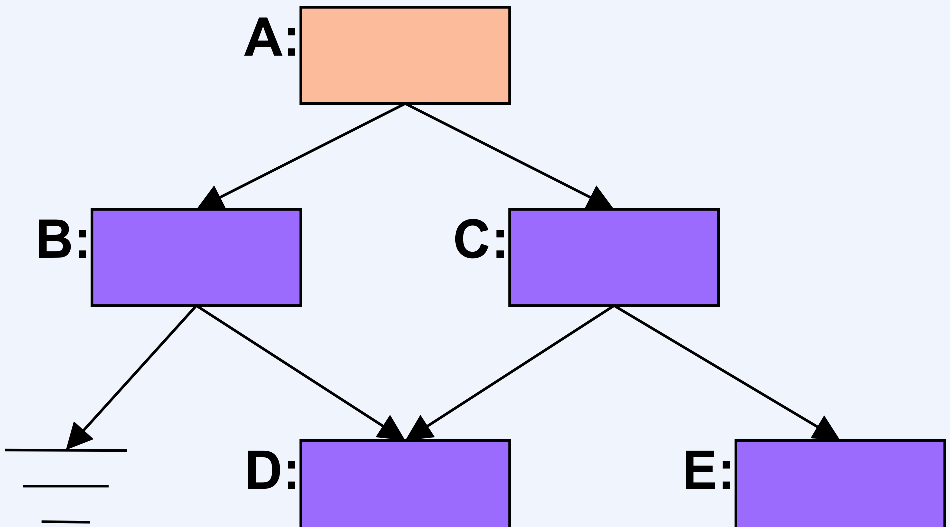
Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Complications



Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

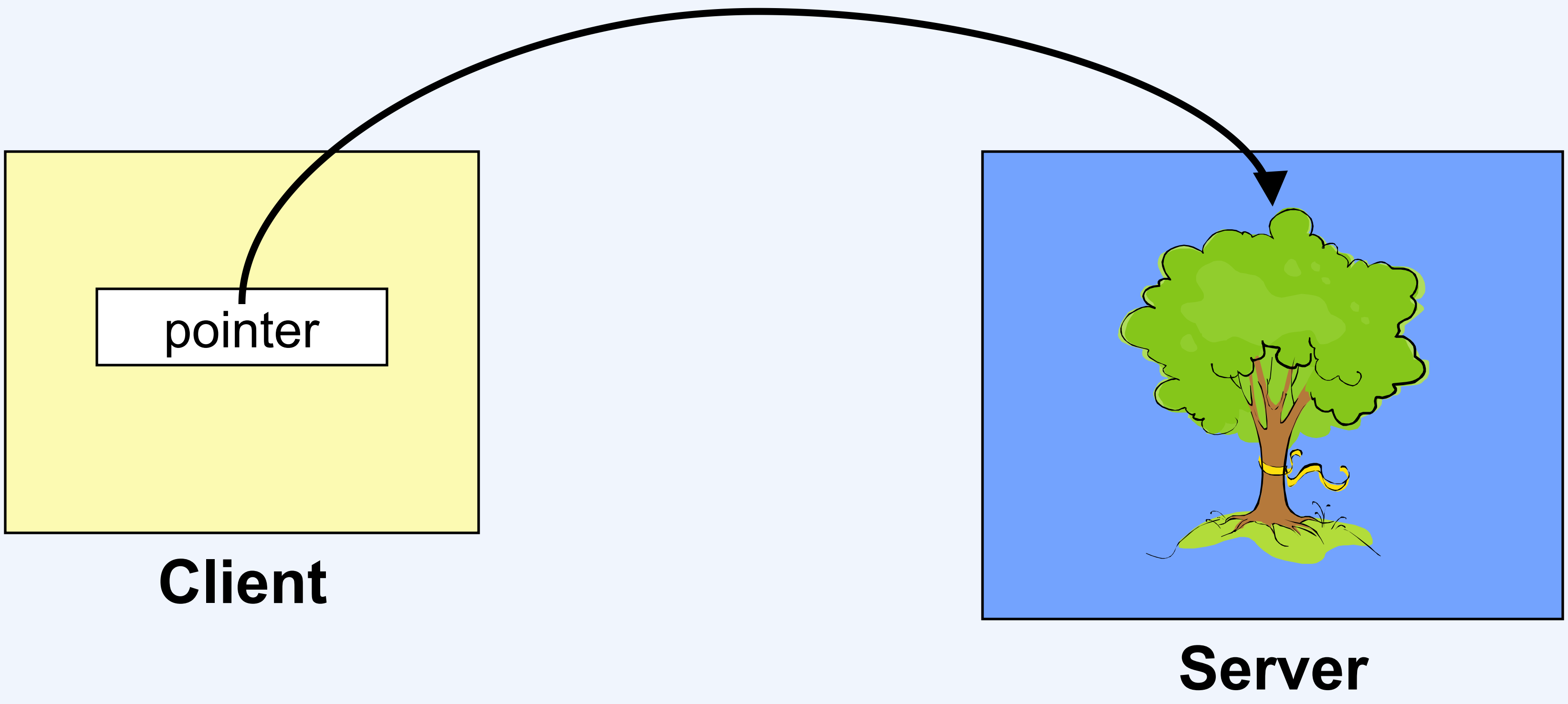
Marshalling Unrestricted Pointers



0 (A):	2
	4
2 (B):	-1
	6
4 (C):	6
	8
6 (D):	
8 (E):	

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Referring to Server State



Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Maintaining Client State on Servers

```
interface trees {
  typedef [context_handle] void *tree_t;

  void create (
      [in]  long value,
      [out] tree_t pine
  );

  void insert (
      [in] long value,
      [in, out] tree_t pine
  );
}
```


Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Go RPC

```
package server

type Args struct {
    A, B int
}
type Quotient struct {
    Quo, Rem int
}
type Arith int

func (t *Arith) Multiply(args *Args, reply *int) error {
    *reply = args.A * args.B
    return nil
}
func (t *Arith) Divide(args *Args, quo *Quotient) error {
    if args.B == 0 {
        return errors.New("divide by zero")
    }
    quo.Quo = args.A / args.B
    quo.Rem = args.A % args.B
    return nil
}
```


Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Server Startup

```
arith := new(Arith)
rpc.Register(arith)
rpc.HandleHTTP()
l, e := net.Listen("tcp", ":1234")
if e != nil {
    log.Fatal("listen error:", e)
}
go http.Serve(l, nil)
```


Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Client Startup

```
client, err := rpc.DialHTTP("tcp", serverAddress + ":1234")
if err != nil {
    log.Fatal("dialing:", err)
}
```


Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Client Call

```
// Synchronous call
args := &server.Args{7,8}
var reply int
err = client.Call("Arith.Multiply", args,
&reply)
if err != nil {
      log.Fatal("arith error:", err)
}
fmt.Printf("Arith: %d*%d=%d", args.A, args.B,
reply)
```


Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Client Call

```
// Asynchronous call
quotient := new(Quotient)
divCall := client.Go("Arith.Divide", args,
quotient, nil)
replyCall := <-divCall.Done   // will be equal to
divCall
// check errors, print, etc.
```


Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

RMI: a Remote Interface

```
package compute;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Compute extends Remote {
    <T> T executeTask(Task<T> t) throws RemoteException;
}
```


Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

RMI: the Argument

```
package compute;

public interface Task<T> {
    T execute();
}
```


Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

RMI: the Server (1)

```
package engine;

import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import compute.Compute;
import compute.Task;

public class ComputeEngine implements Compute {
    public ComputeEngine() { super(); }
    public <T> T executeTask(Task<T> t) {
        return t.execute();
    }
```


Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

RMI: the Server (2)

```
public static void main(String[] args) {
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager());
        }
        try {
            String name = "Compute";
            Compute engine = new ComputeEngine();
            Compute stub =
                (Compute) UnicastRemoteObject.exportObject(engine, 0);
            Registry registry = LocateRegistry.getRegistry();
            registry.rebind(name, stub);
            System.out.println("ComputeEngine bound");
        } catch (Exception e) {
            System.err.println("ComputeEngine exception:");
            e.printStackTrace();
        }
    }
}
```


Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

RMI: the Client

```
public class ComputePi {
    public static void main(String args[]) {
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager());
        }
        try {
            String name = "Compute";
            Registry registry = LocateRegistry.getRegistry(args[0]);
            Compute comp = (Compute) registry.lookup(name);
            Pi task = new Pi(Integer.parseInt(args[1]));
            BigDecimal pi = comp.executeTask(task);
            System.out.println(pi);
        } catch (Exception e) {
            System.err.println("ComputePi exception:");
            e.printStackTrace();
        }
    }
}
```


Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

RMI: the Client's Compute Object

```
package client;

import compute.Task;
import java.io.Serializable;
import java.math.BigDecimal;

public class Pi implements Task<BigDecimal>, Serializable {
    private final int digits;
    public Pi(int digits) {this.digits = digits;} // constructor
    // lots of stuff deleted …
    public BigDecimal execute() {
        return computePi(digits);
    }
    // lots more stuff deleted …
}
```


Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.

Other examples

- Protocol Buffers (mostly an IDL)
- Thrift. Developed at Facebook. Now part of Apache Open Source. Supports multiple data encodings & transport mechanisms. Works across multiple languages.
- Avro. Also Apache standard. Created as part of Hadoop project. Uses JSON. Not as elaborate as Thrift.

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca.
Some content from David Andersen.