

# Remote Procedure Call

Amin Vahdat

CSE 123b

May 11, 2006

## Annoucements

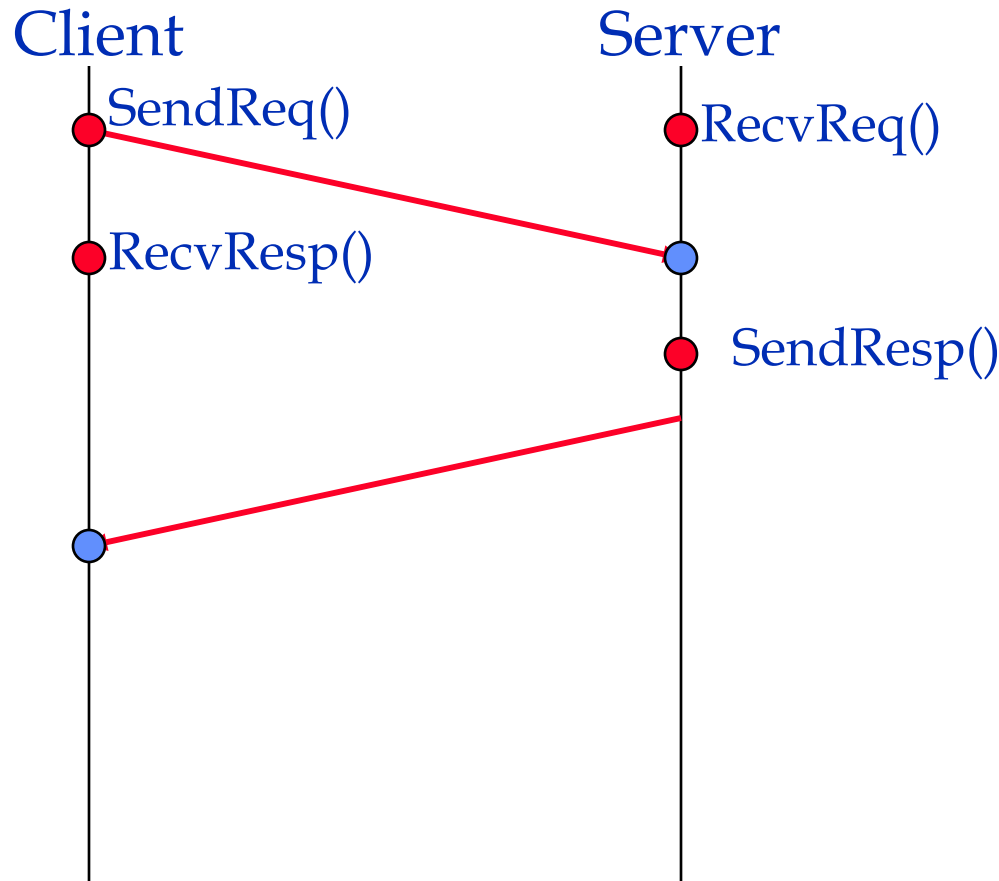
- Second assignment due May 15

# Distributed Programming

# Protocol-to-Protocol Interface

- Send/Receive semantics
  - Synchronous vs. Asynchronous
- Process Model
  - Single process vs. multiple process
  - Avoid context switches
- Buffer Model
  - Avoid data copies

# Client/Server Interaction



- Client sends request
  - Client waits for response (blocking)
- Server wakes up
  - Computes, sends response
  - Response wakes up client
- Server structure?
  - Process, thread, FIFO

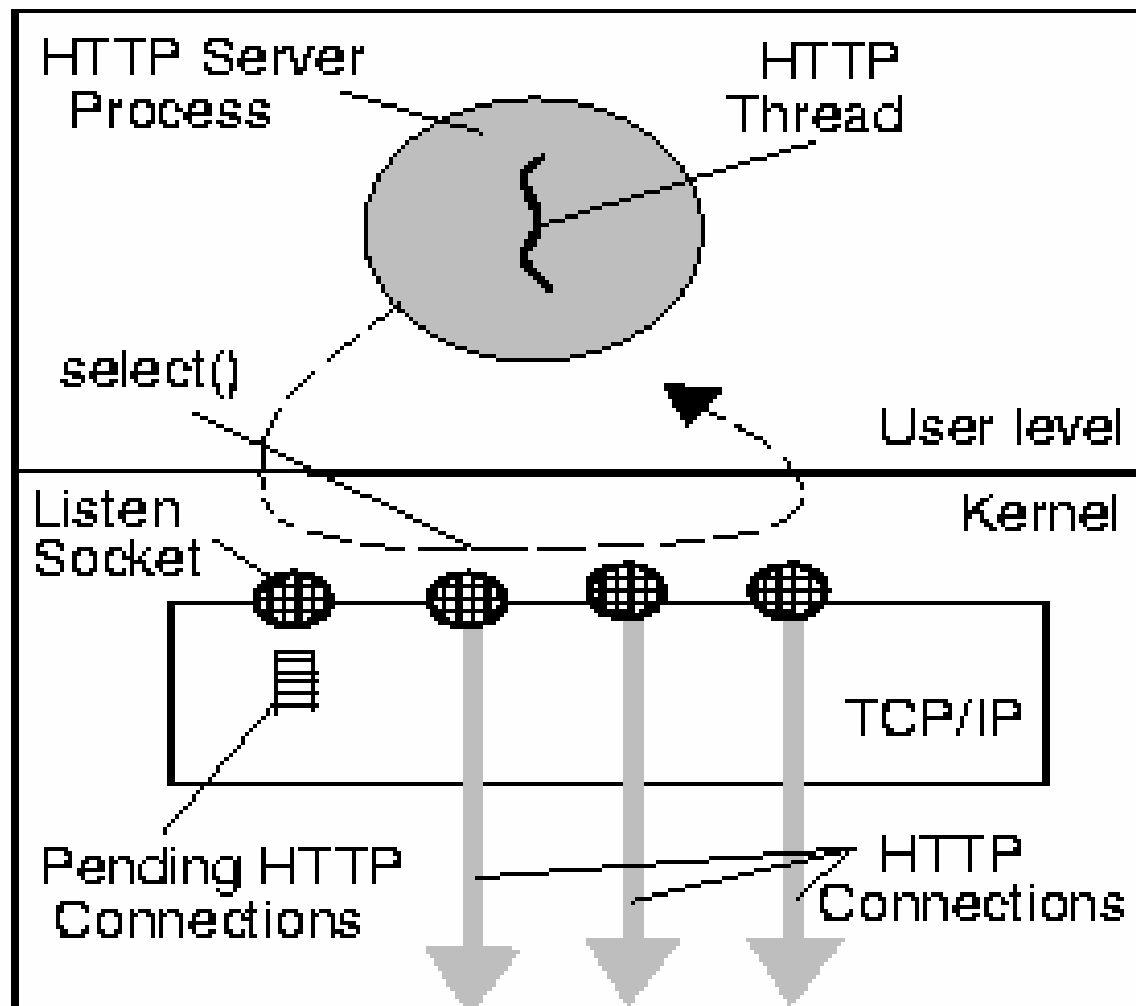
## High Overhead of select()

- Must copy relatively large data structures into and out of kernel each time
  - Specifying which fd's program is interested in
- Common case: very few fd's have any data available
  - Thus array of fd's is relatively sparse
  - Must still iterate over entire list to see what is available
- Performance of early web servers bottlenecked by select()

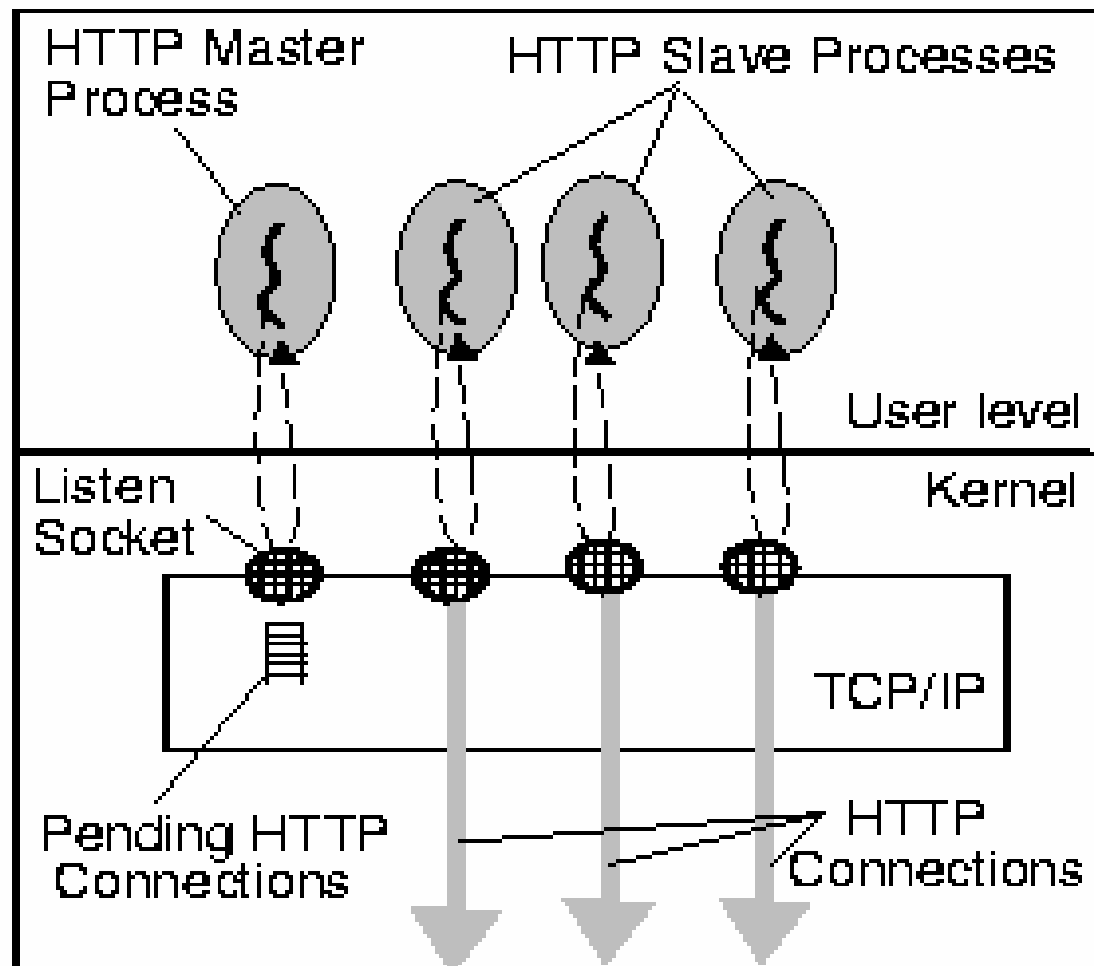
## Alternative: /dev/poll Interface

- /dev/poll is pseudo device driver available on a number of operating systems
  - Open just like normal file, receive fs
  - Manipulate using ioctl()
- Pass into kernel set of target fd's interested
  - Kernel maintains state associated with /dev/poll fd
  - Program only updates kernel when fd's should be added or deleted from set
- ioctl() calls on /dev/poll return only the set of fd's that actually have data available on them

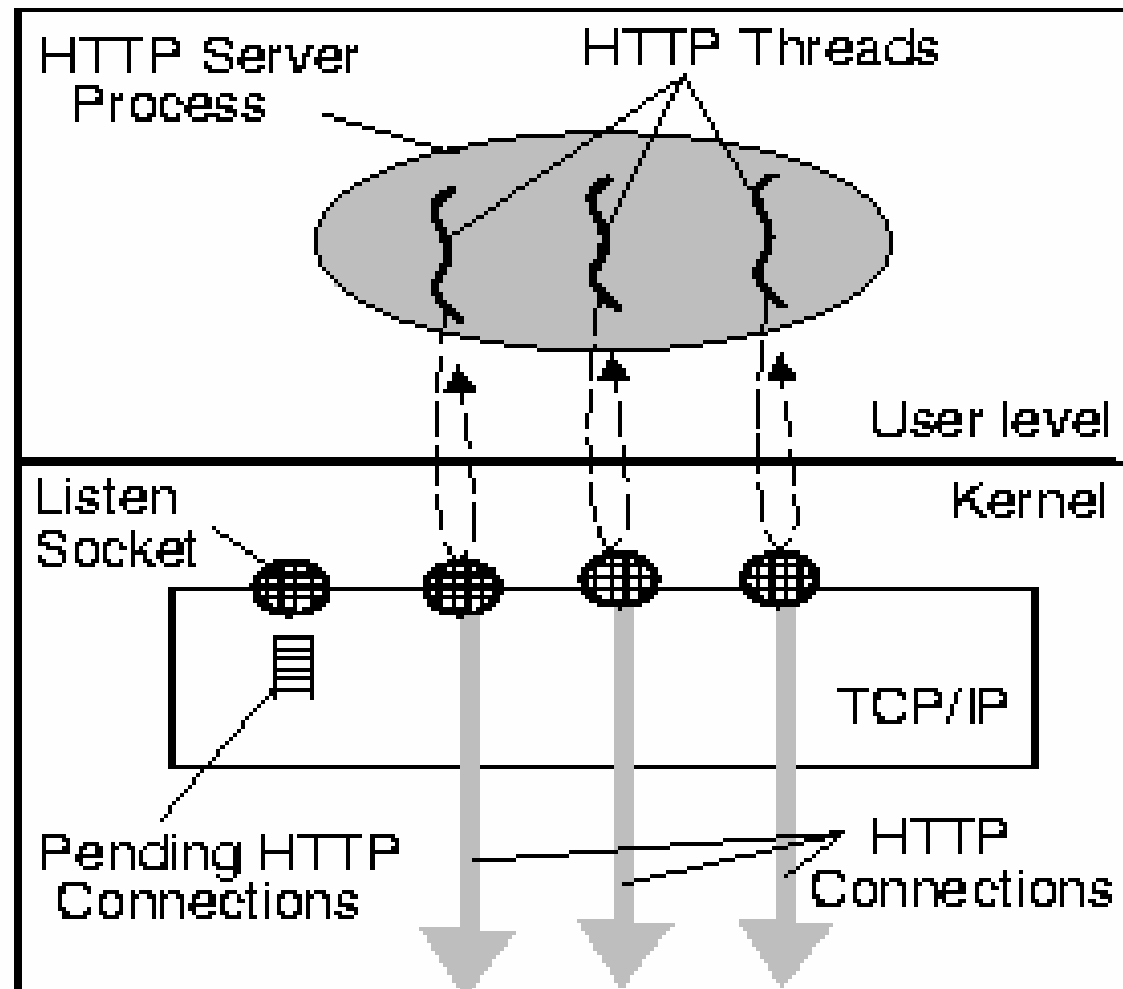
## Single Process, Event-Driven



## Pre-Forked Worker Pool



## Single Process, Multi-threaded



## High Performance Servers: What About Dynamic Content?

- Queries to web servers must spawn work rather than grab static document
- Earliest approach forked off CGI programs
  - High overhead of process creation/initialization
- FastCGI allowed persistent processes
  - Context switch still necessary
- Microsoft/Netscape standards for running code in server
  - Fault isolation is compromised
- Modern network services hand work off to separate machine (2<sup>nd</sup> tier in 3-tier service)

## Remote Procedure Call (RPC)

- Distributed programming challenging
  - Need common primitives/abstraction to hide complexity
  - E.g., file system abstraction to hide block layout, process abstraction for scheduling/fault isolation
- In early 1980's, researchers at PARC noticed most distributed programming took form of *remote procedure call*
- Popular variant: Remote Method Invocation (RMI)
  - Obtain handle to remote object, invoke method on object
  - Transparency goal: remote object appear as if local object

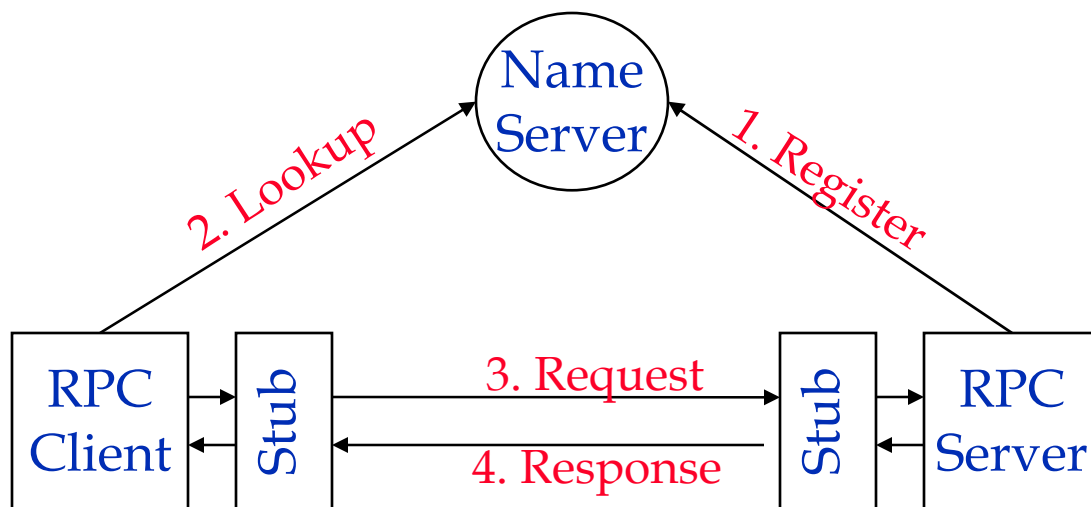
# Remote Procedure Call Issues

- Underlying protocol
  - UDP vs. TCP
  - Advantages/Disadvantages?
- Semantics
  - What happens on network/host failure?
  - No *fate sharing* (compared to local machine case)
- Transparency
  - Hide network communication/failure from programmer
  - With language support, can make remote procedure call “look just like” local procedure call

## Remote Procedure Call Issues

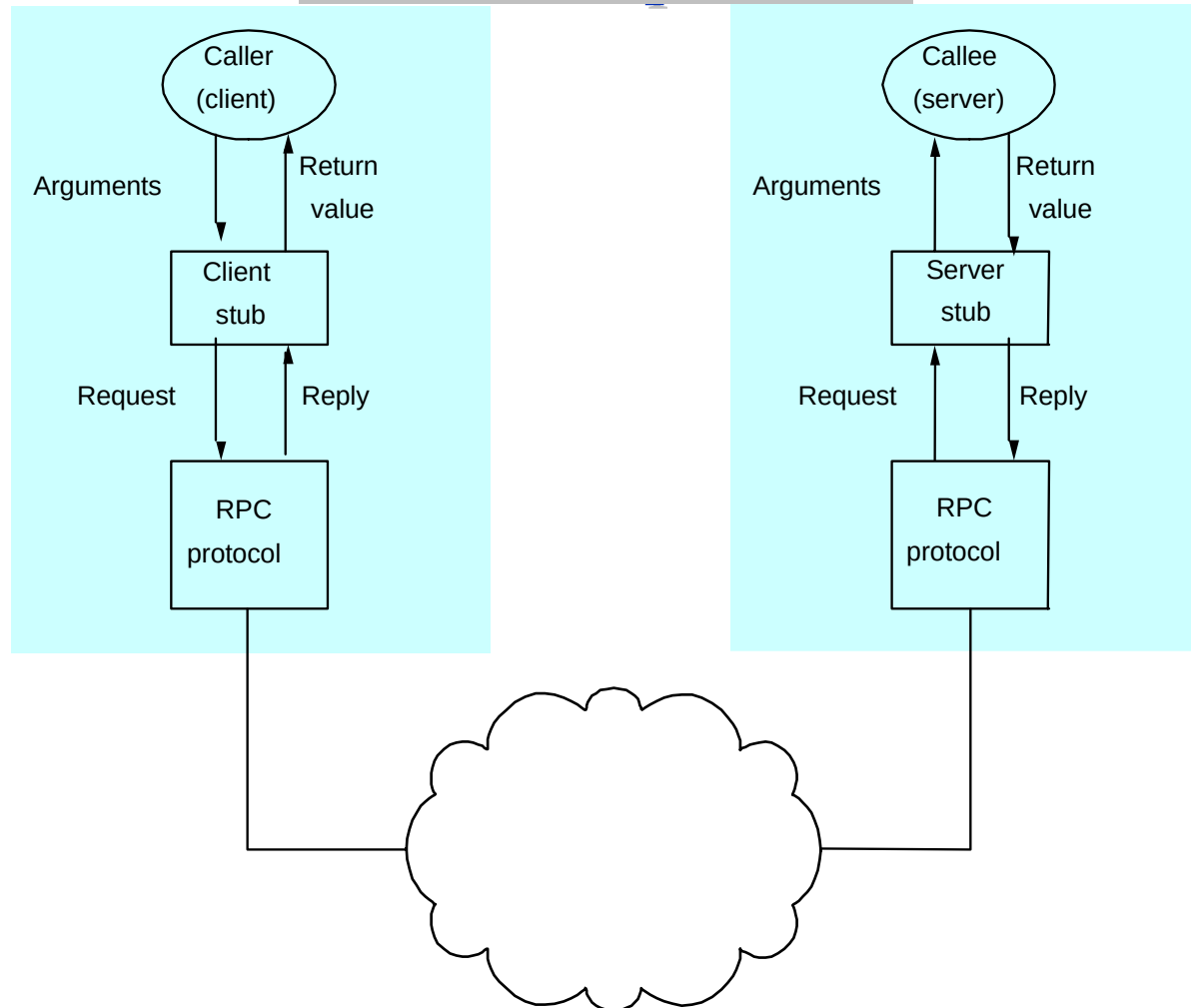
- Idempotent operations
  - Can you re-execute operations without harmful side effects
  - Idempotency means that executing the same procedure does not have visible side effects
- Timeout value
  - How long to wait before re-transmitting request?

# Remote Procedure Call



- Client stub indicates which procedure should run at server
  - Marshals arguments
- Server stub unmarshals arguments
  - Big switch statement to determine local procedure to run

# RPC Components



## RPC Mechanics

- Client issues request by calling stub procedure
  - Stubs can be automatically generated with compiler support
- Stub procedure marshals arguments, transmits requests, blocks waiting for response
  - RPC layer deals with network issues (e.g., TCP vs. UDP)
- Server stub
  - Unmarshals arguments
  - Determines correct local procedure to invoke, computes
  - Marshals results, transmits to client
  - Can also be automatically generated with language support

## RPC Semantics

| Delivery Guarantees |                     |                      | RPC Call Semantics   |
|---------------------|---------------------|----------------------|----------------------|
| Retry Request       | Duplicate Filtering | Retransmit Response  |                      |
| No                  | NA                  | NA                   | <i>Maybe</i>         |
| Yes                 | No                  | Re-execute Procedure | <i>At-least once</i> |
| Yes                 | Yes                 | Retransmit reply     | <i>At-most once</i>  |

# RPC Binding

- Binding
  - Servers register the *service* they provide to a name server
  - Clients lookup services they need from server
  - Bind to server for a particular set of remote procedures
  - Operation informs RPC layer which machine to transmit requests to
- How to locate the RPC name service?

# Presentation Formatting

- Marshalling (encoding) application data into messages
- Unmarshalling (decoding) messages into application data



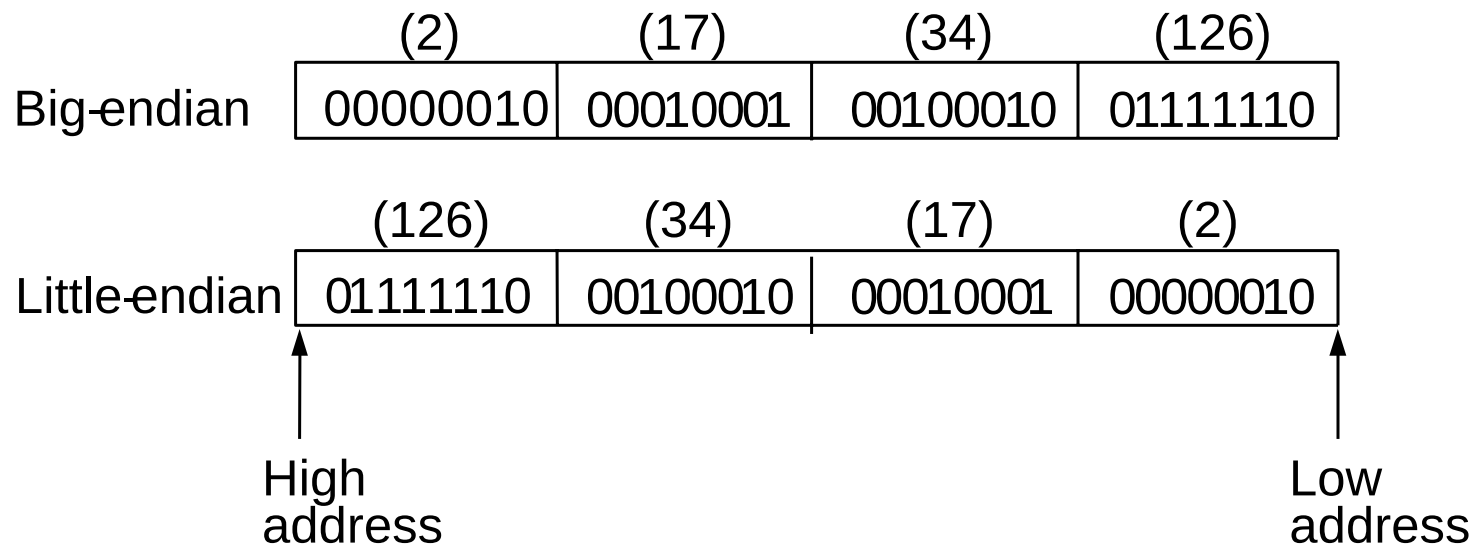
- Data types to consider:
  - integers
  - floats
  - strings
  - arrays
  - structs

## Types of data we do not consider

images  
video  
multimedia documents

## Difficulties

- Representation of base types
  - Floating point: IEEE 754 versus non-standard
  - Integer: big-endian versus little-endian (e.g., 34,677,374)

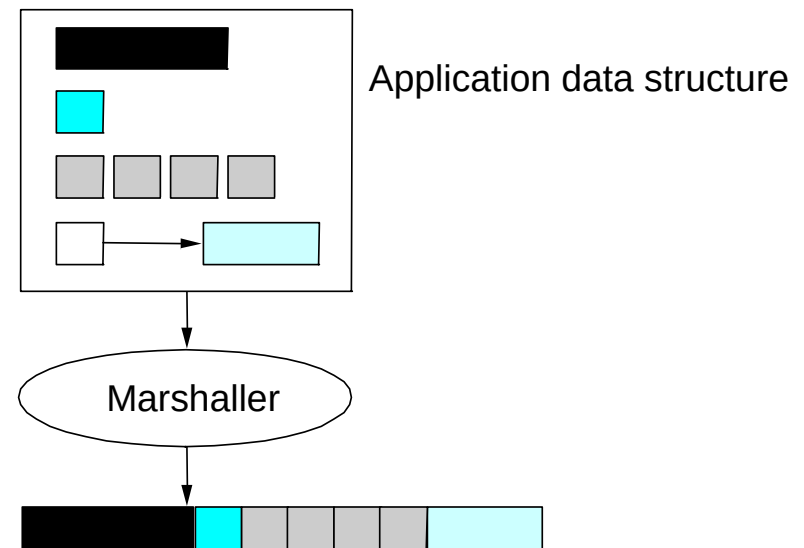


- Compiler layout of structures

# Taxonomy

## ■ Data types

- Base types (e.g., ints, floats); must convert
- Flat types (e.g., structures, arrays); must pack
- Complex types (e.g., pointers); must linearize



## ■ Conversion Strategy

- Canonical intermediate form
- Receiver-makes-right (an  $N \times N$  solution)

# SunRPC

- Originally implemented for popular NFS (network file service)
- XID (transaction id) uniquely identifies request
- Server does not remember last XID it serviced
  - Problem if client retransmits request while reply is in transit
  - Provides *at-least once* semantics

