Client/Server Computing e RPC

Slides are mainly taken from «*Operating Systems: Internals and Design Principles*”, *8/E* William Stallings (Chapter 16).

Sistemi di Calcolo (II semestre) – Roberto Baldoni

Client/Server Computing

- Client machines are generally single-user PCs or workstations that provide a highly user-friendly interface to the end user
- Each server provides a set of shared services to the clients
- The server enables many clients to share access to the same database and enables the use of a high-performance computer system to manage the database

Generic Client/Server Environment

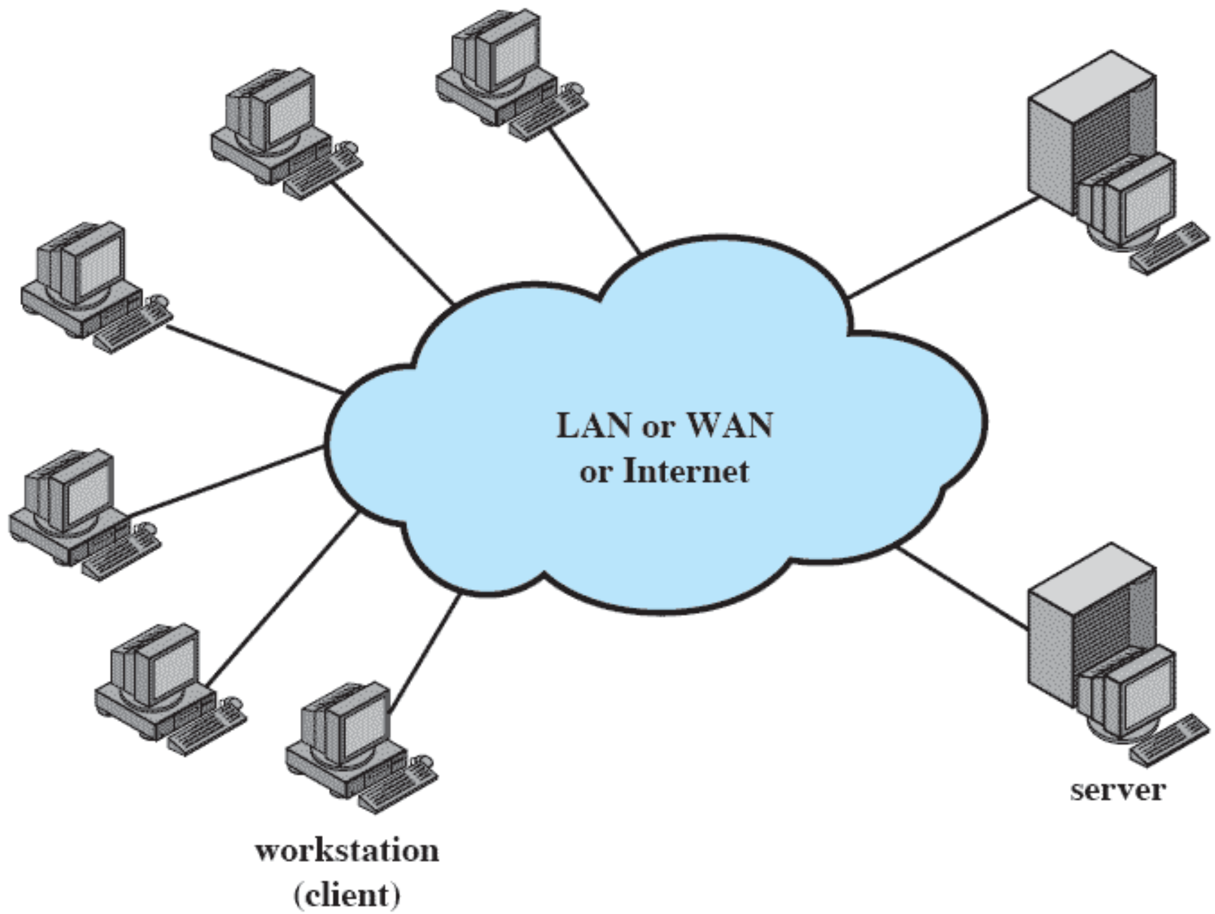


Figure 16.1 Generic Client/Server Environment

Client/Server Applications

- Basic software is an operating system running on the hardware platform
- Platforms and the operating systems of client and server may differ
- These lower-level differences are irrelevant as long as a client and server share the same communications protocols and support the same applications

Generic Client/Server Architecture

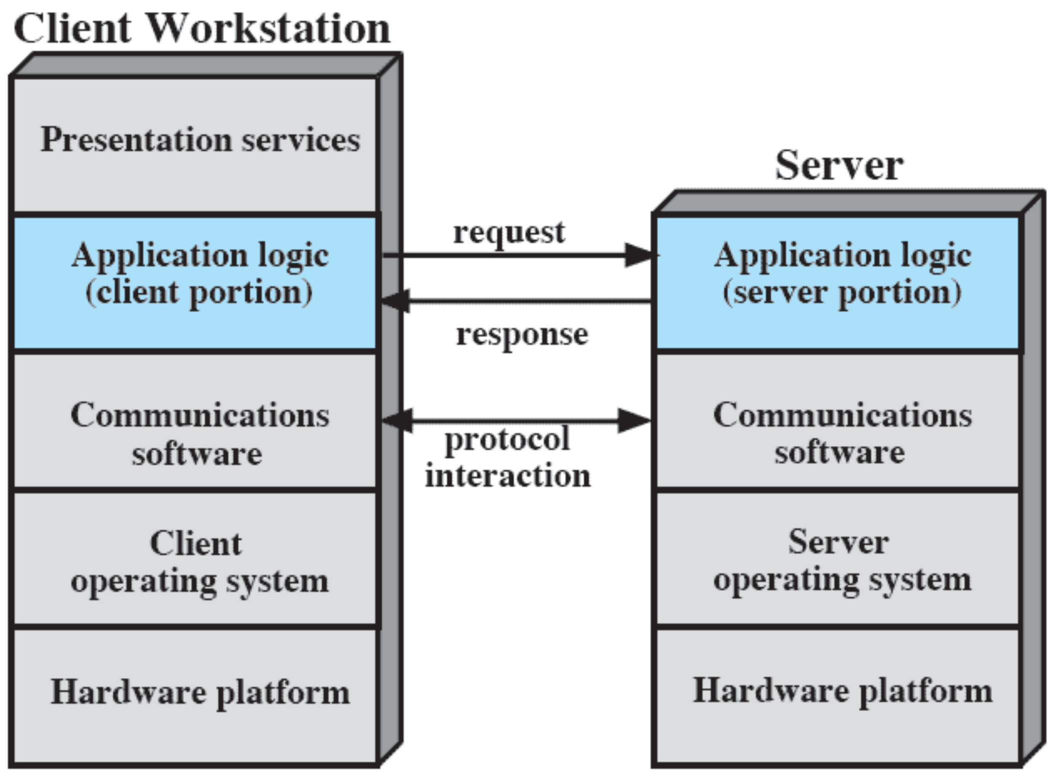


Figure 16.2 Generic Client/Server Architecture

Client/Server Applications

- Bulk of applications software executes on the server
- Application logic is located at the client
- Presentation services in the client

Database Applications

- The server is a database server
- Interaction between client and server is in the form of transactions
  - the client makes a database request and receives a database response
- Server is responsible for maintaining the database

Client/Server Architecture for Database Applications

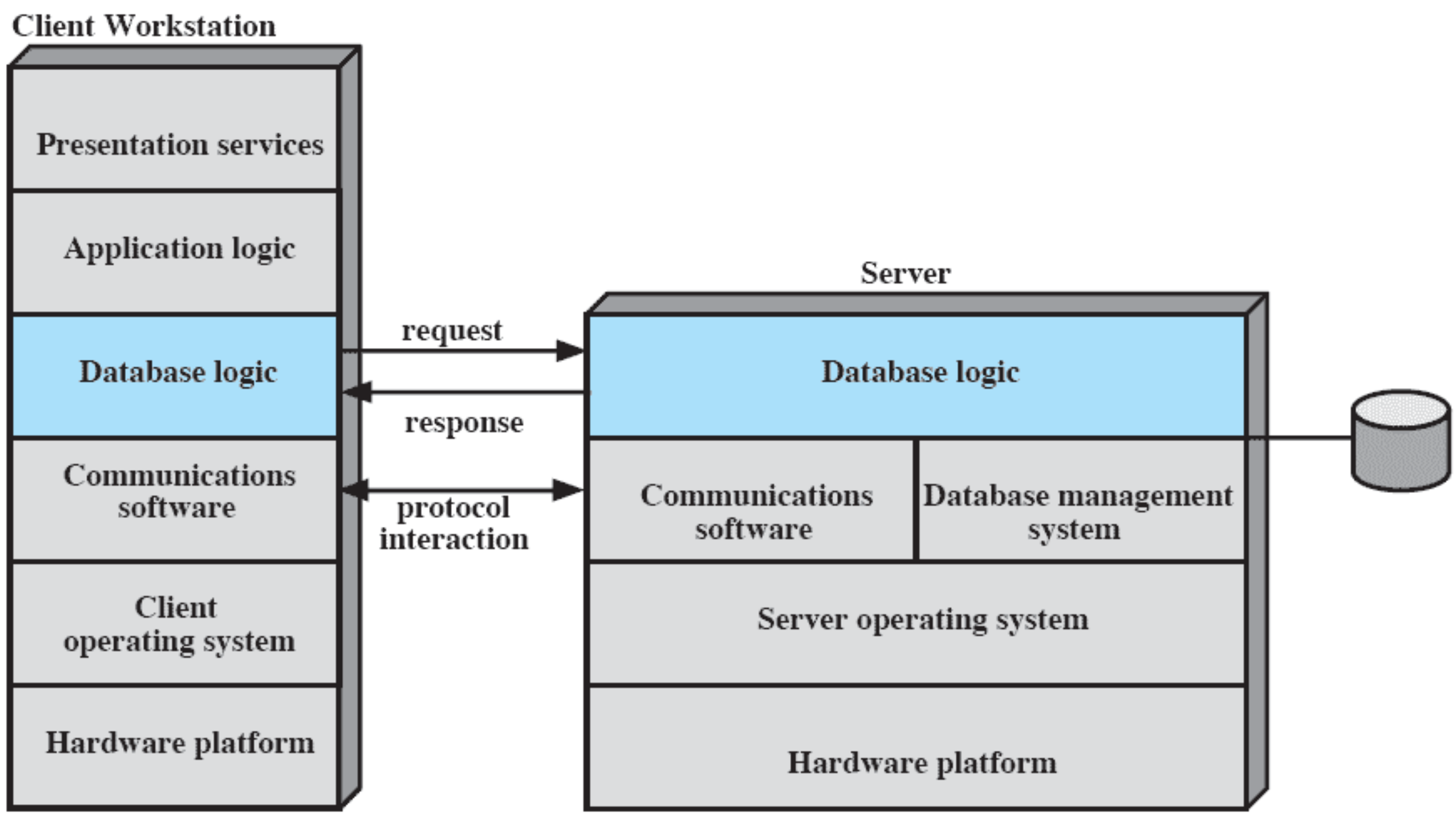
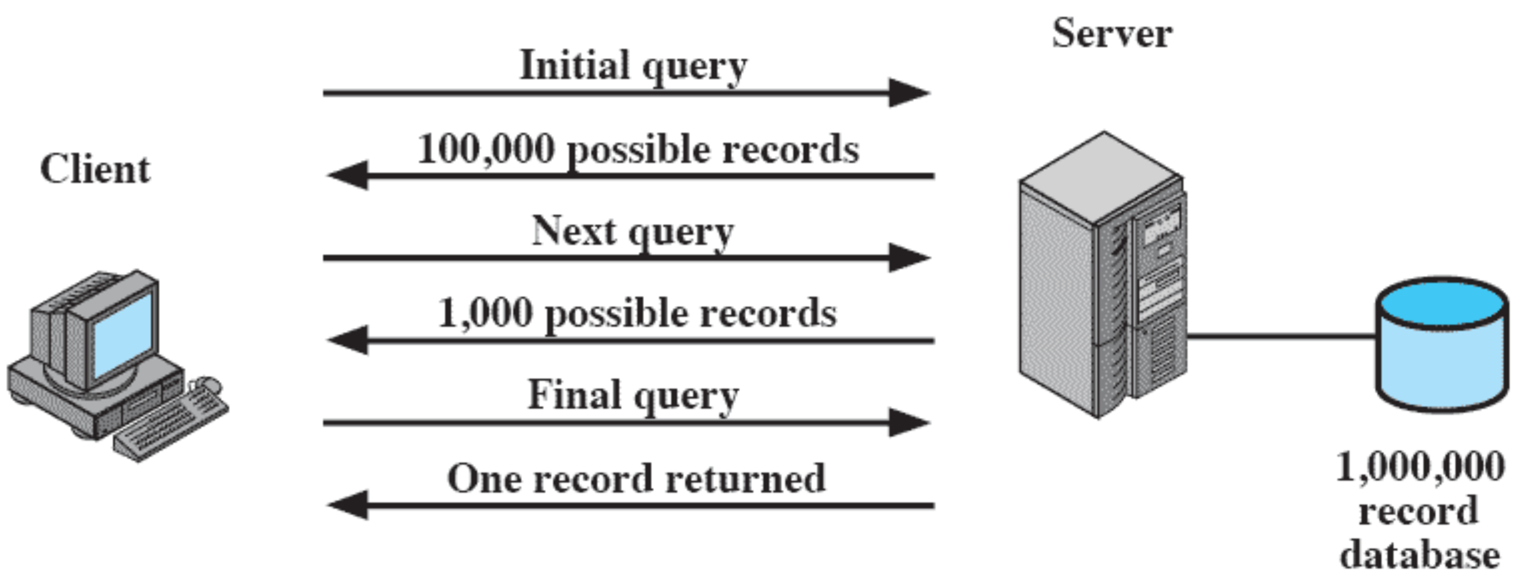


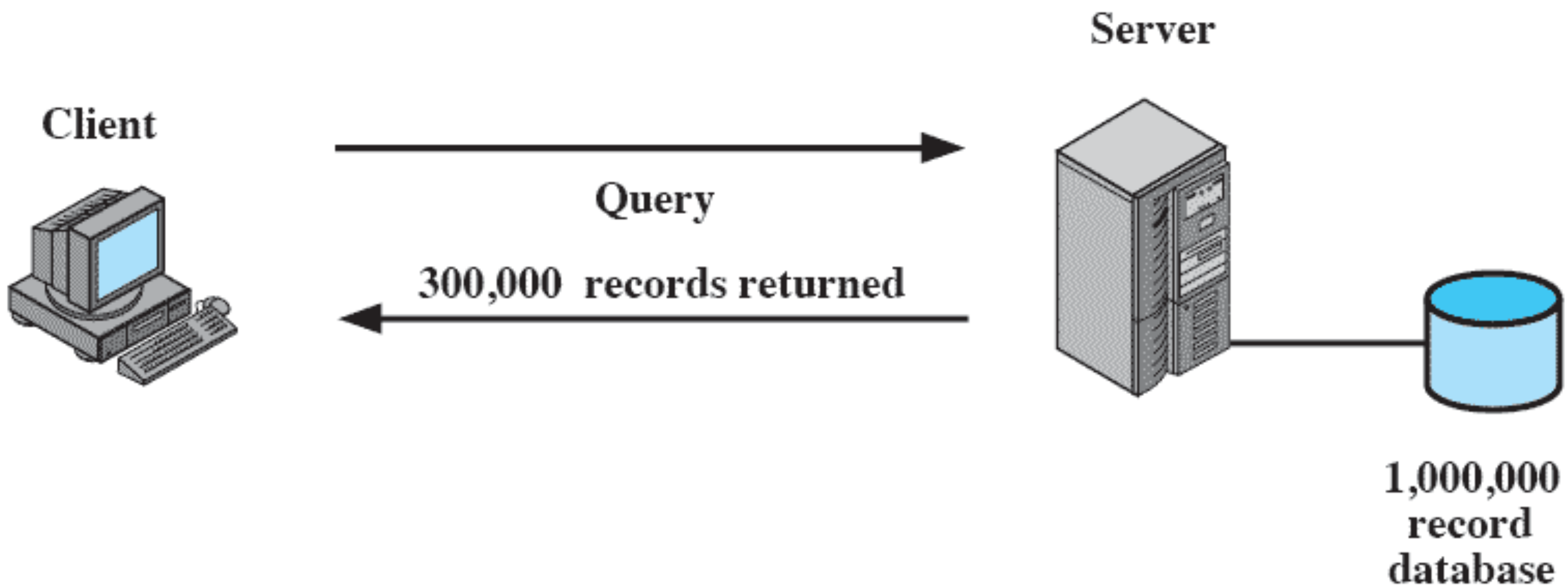
Figure 16.3 Client/Server Architecture for Database Applications

Client/Server Database Usage



(a) Desirable client/server use

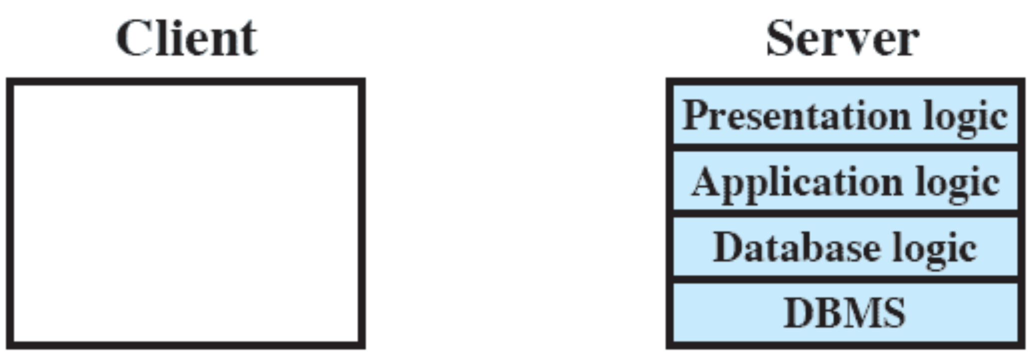
Client/Server Database Usage



(b) Misused client/server

Classes of Client/Server Applications

- Host-based processing
  - Not true client/server computing
  - Traditional mainframe environment



(a) Host-based processing

Classes of Client/Server Applications

- Server-based processing
  - Server does all the processing
  - Client provides a graphical user interface

(b) Server-based processing

Classes of Client/Server Applications

- Client-based processing
  - All application processing done at the client
  - Data validation routines and other database logic functions are done at the server

(c) Cooperative processing

Classes of Client/Server Applications

- Cooperative processing
  - Application processing is performed in an optimized fashion
  - Complex to set up and maintain

(d) Client-based processing

Three-tier Client/Server Architecture

- Application software distributed among three types of machines
  - User machine
    - Thin client
  - Middle-tier server
    - Gateway
    - Convert protocols
    - Merge/integrate results from different data sources
  - Backend server

Three-tier Client/Server Architecture

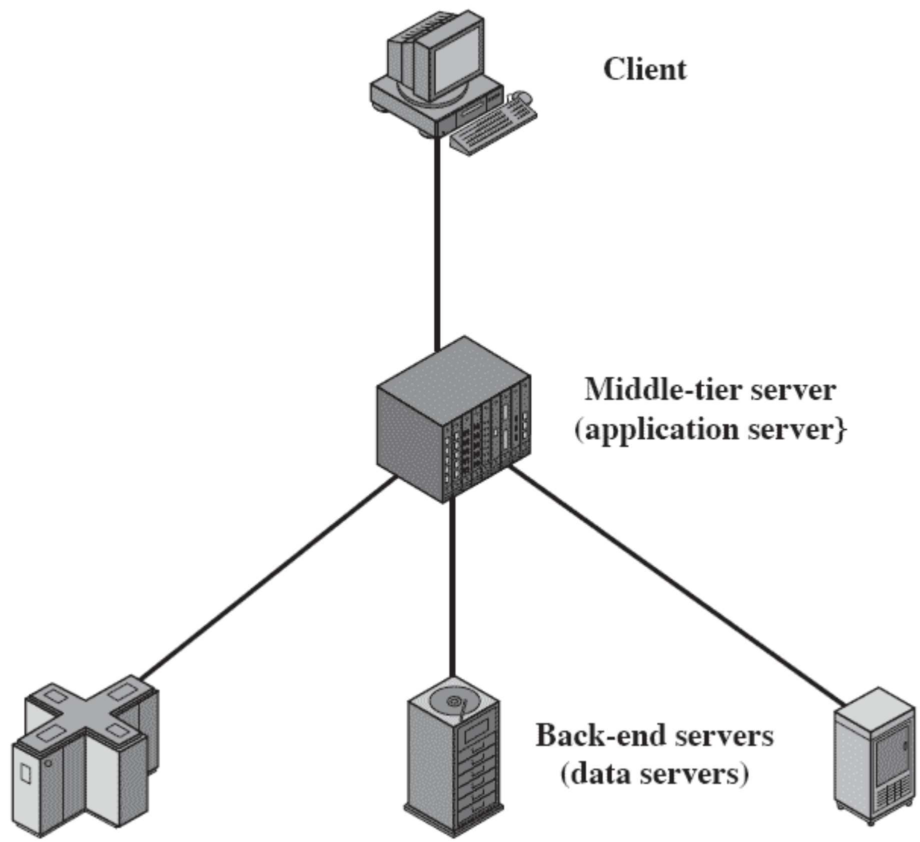
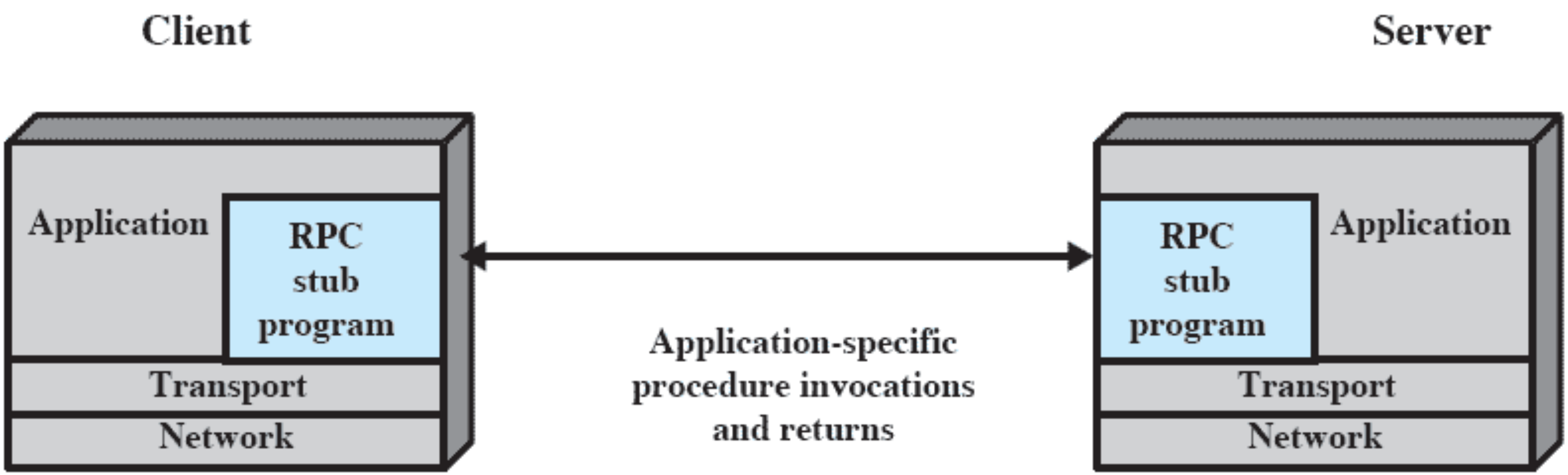


Figure 16.6 Three-tier Client/Server Architecture

Remote Procedure Calls

Distributed Message Passing

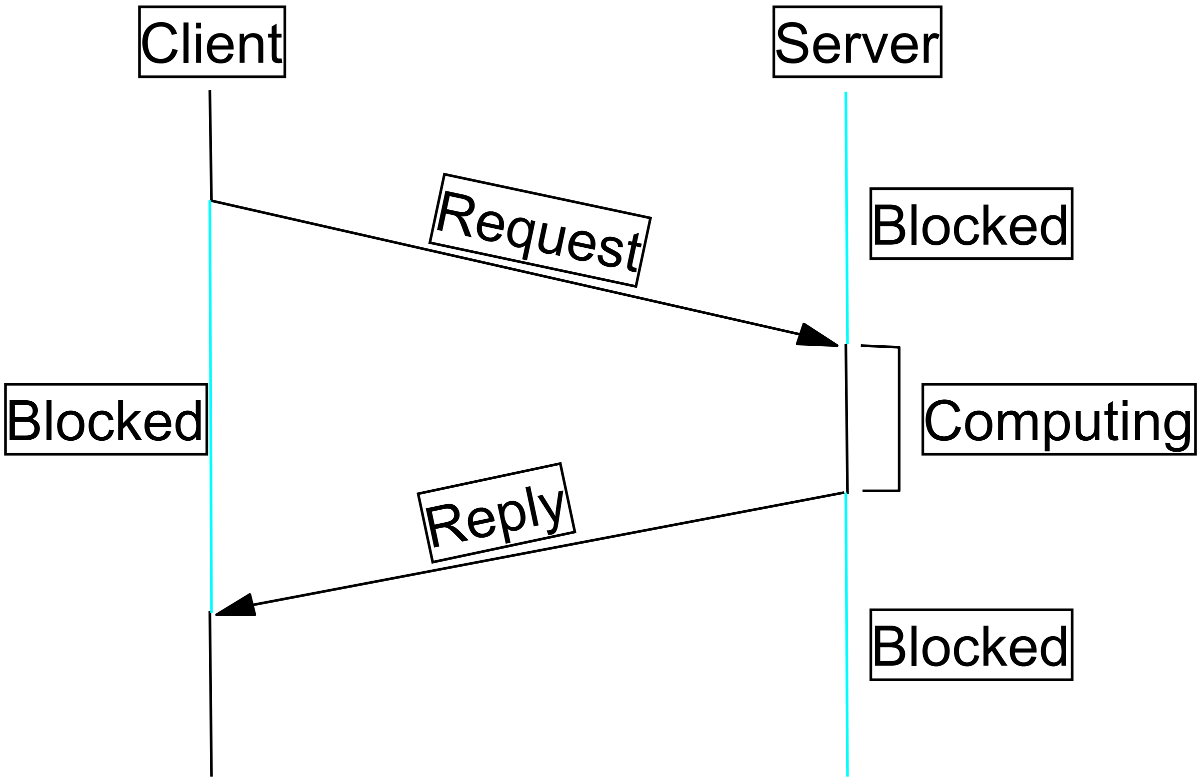


(b) Remote Procedure Calls

Remote Procedure Calls

- Allow programs on different machines to interact using simple procedure call/return semantics
- Widely accepted
- Standardized
  - Client and server modules can be moved among computers and operating systems easily

RPC Timeline



Remote Procedure Call Mechanism

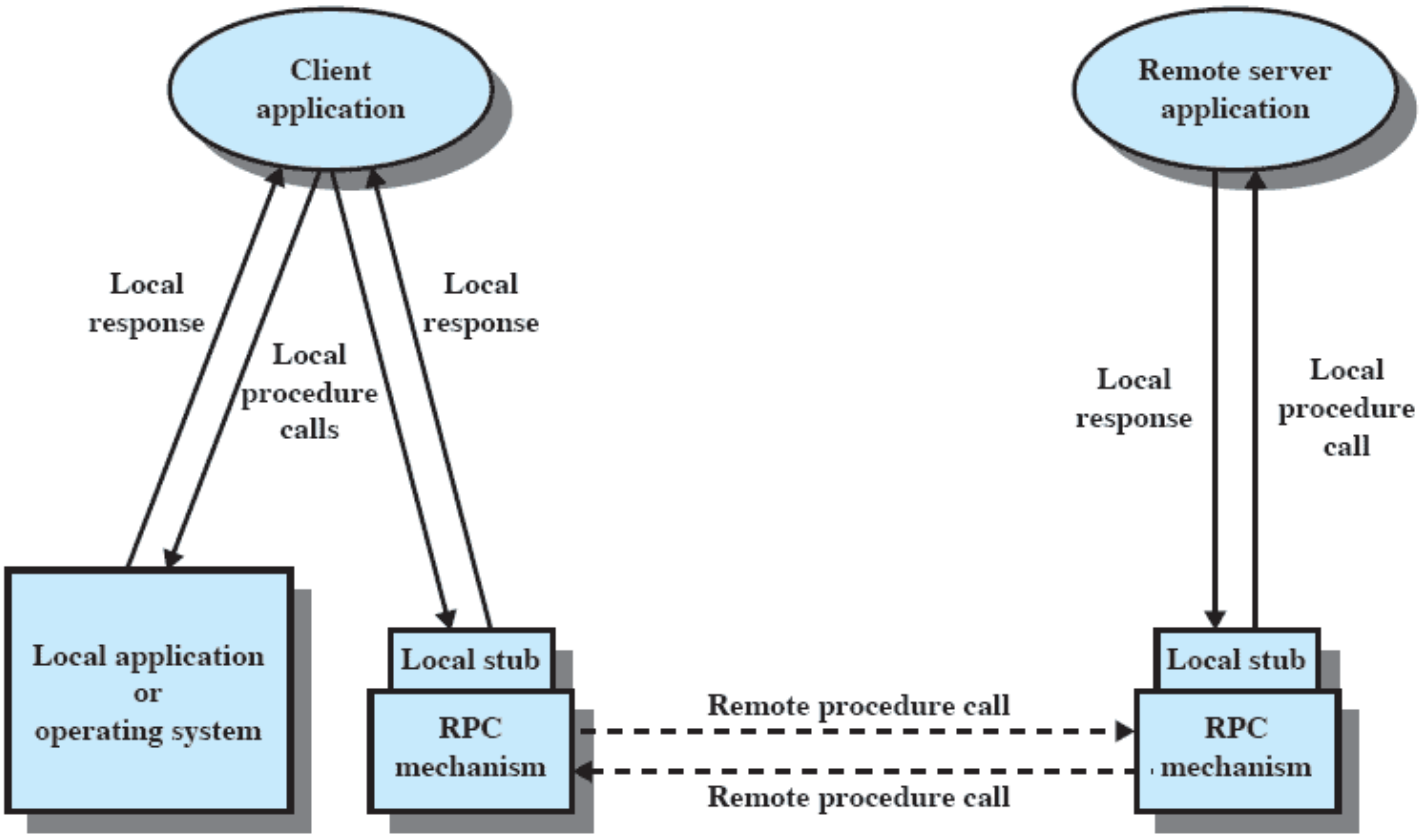


Figure 16.12 Remote Procedure Call Mechanism

Remote Procedure Call

Una *Chiamata a Procedure Remota* (RPC) trasforma l'interazione Client/Server in una chiamata a procedura, simile a quella locale, nascondendo al programmatore la maggiore parte dei meccanismi implementativi che la compongono, come:

- **l'interscambio di messaggi,**
- **la localizzazione del server che fornisce il servizio**
- **le possibili differenti rappresentazioni dei dati delle macchine coinvolte nell'interazione.**

RPC

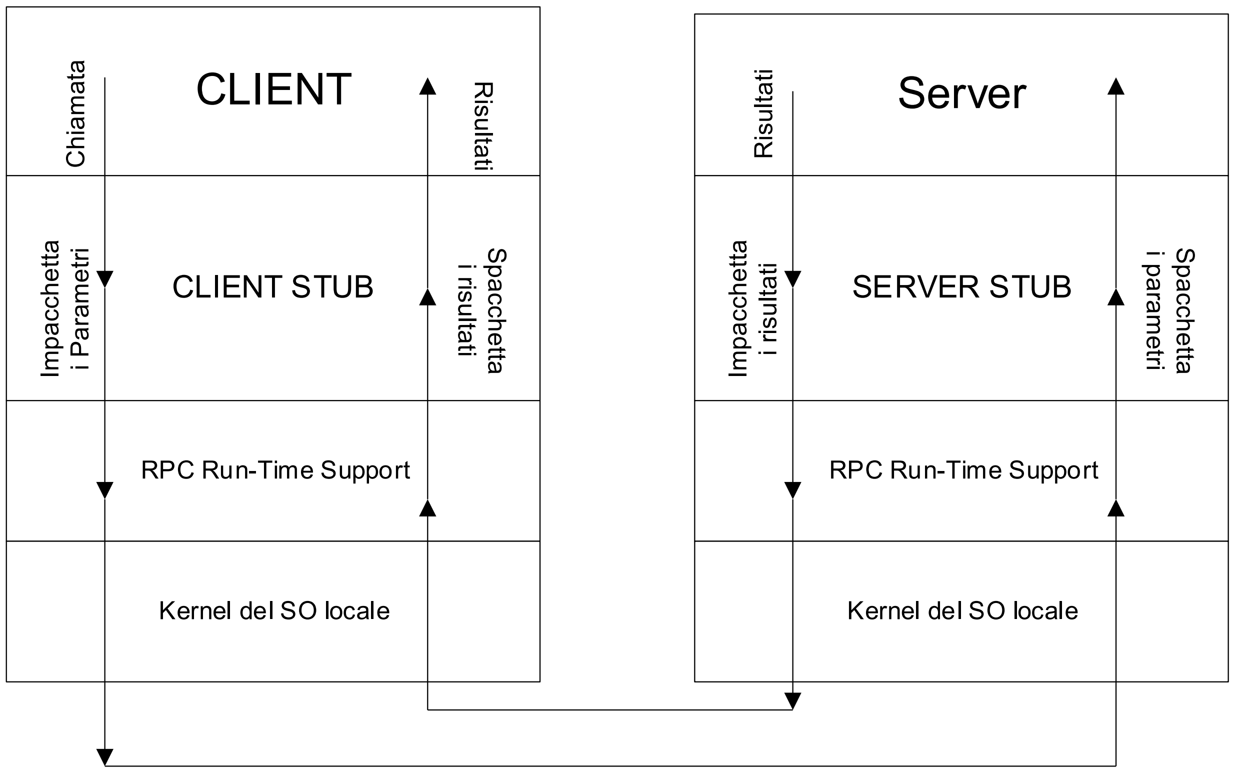
Questo mascheramento avviene in quattro fasi:

- **A tempo di scrittura del codice**. Le RPC usate/fornite dovranno essere dichiarate esplicitamente dal programmatore attraverso *import/export* delle definizioni delle interfacce.
- **A tempo di esecuzione.** Ogni macchina su cui è in esecuzione un programma client e/o server dovrà avere un supporto a tempo si esecuzione per le RPC (RPC run-time support) in grado di eseguire alcune operazioni delle RPC come ad esempio la localizzazione del server o la registrazione di un nuovo servizio offerto da un nuovo server.
- **A tempo di compilazione**. Durante la compilazione per ogni chiamata a procedura remota vengono agganciate linee di codice al programma originario (stub) che permettono operazioni standard sui dati (impacchettamento e codifica universalmente riconosciuta) e le chiamate al RPC run-time support;

Meccanismi per RPC

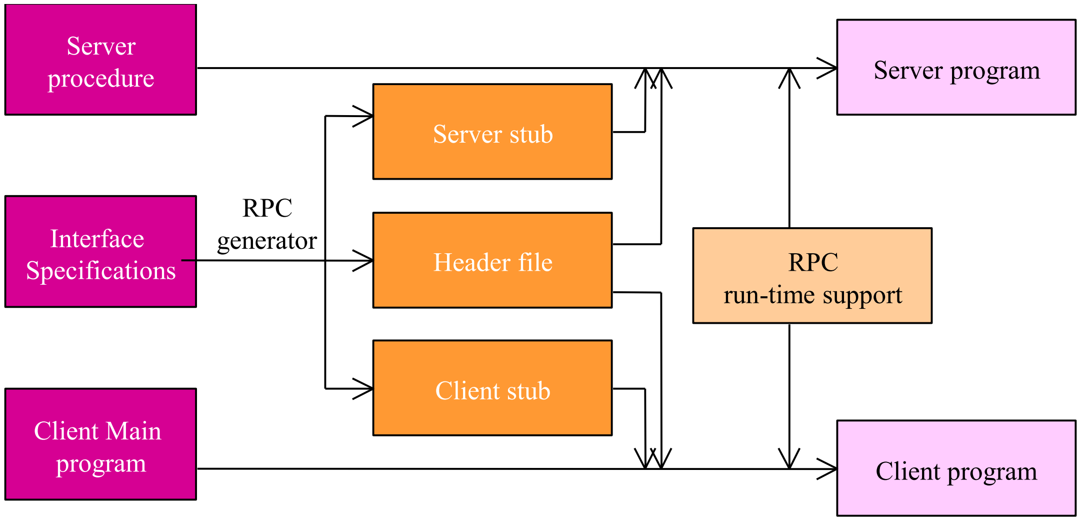
- Un protocollo che nasconde le insidie della rete (perdita di pacchetti e riordinamento dei messaggi)
- Un meccanismo per impacchettare gli argomenti dal lato chiamante e per spacchettarli dal lato chiamato

RPC



CLIENT
Chiamata
Risultati
CLIENT STUB
Impacchetta i Parametri
Spacchetta i risultati
RPC Run-Time Support
Kernel del SO locale
Server
Risultati
SERVER STUB
Impacchetta i risultati
Spacchetta i parametri
RPC Run-Time Support
Kernel del SO locale

RPC


Server procedure
Interface Specifications
Client Main program
RPC generator
Server stub
Header file
Client stub
RPC run-time support
Server program
Client program

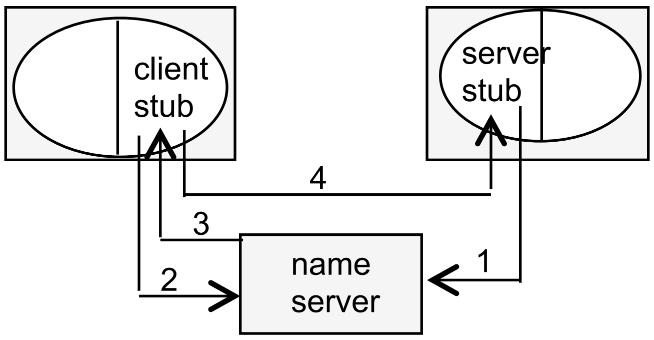
Localizzazione del server

Metodo Statico. Cablare all’interno del client l’indirizzo (IP address) del server.

Metodo Dinamico. Lo stub del client, mentre impacchetta i dati, invia concorrentemente un broadcast richiedendo l’indirizzo di un server in grado di eseguire la RPC desiderata. Il supporto run-time delle RPC di ogni machina risponde se il servizio richiesto e’ fornito da un suo server in esecuzione.

Localizzazione del server

Name Server._Il client alla ricerca di un server consulta una entità, name server, la quale gestisce una lista di associazioni server-servizi.



Passaggio dei Parametri

Call by Reference – sconsigliato

Call by Copy/Restore. Copia una variabile *a,* da parte dello stub del client, nel pacchetto dati (come se fosse passata per valore). Il nuovo valore di *a*, restituito dal server nei parametri di ritorno della RPC sarà copiato, dallo stub del client, nella cella di memoria che contiene la variabile *a*.

CLIENT SIDE	**SERVER SIDE**
begin	**procedure doppioincr (var x,y: integer)**
…..	**begin**
a=0;	***…….***
doppioincr(a,a);	***x:= x+2;***
writeln (a); ...	***y:= y+3;***
end	**end**

Risultato: Call by ref, a="5"

Call by copy/restore a= "2" o "3"

dipendente dall'implementazione dello stub del client

Semantica delle RPC

"At least once"

- Time-out stub del client
- Ritrasmissione

"At most once"

- Time-out stub del client
- Codice di errore di ritorno

"Exactly once"

Semantica delle RPC

“Exactly once”

Lato server

– Immagazzinare tutti i risultati delle RPC nel server (logging)
– Se arriva al server una richiesta già effettuata il risultato dovrà essere preso dal file di log

Lato Client

– Numerare tutte le richeste dai client (sequence number)
– Numero di reincarnazione (add 1 ad ogni restart del client)
– A seguito di un guasto un client invia il numero di reincarnazione corrente prima di cominciare ad eseguire le RPC (per uccidere le RPC pendig della incarnazione precedente.

Sottosistema di Comunicazione

TCP – troppo costoso in fase di connessione

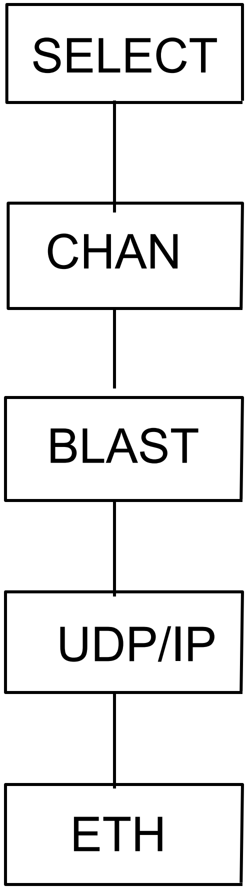
UDP – nessun costo di connessione ma si deve gestire al di sopra an protocollo per l’invio affidabile dei dati e per il dispatching delle RPC all’interno dello stesso host

IP – dobbiamo gestire anche il multiplexing/demultiplexing dei pacchetti all’interno del singolo host oltre ai problemi che derivano dall’utilizzo di UDP

Gestione di pacchetti di riscontro

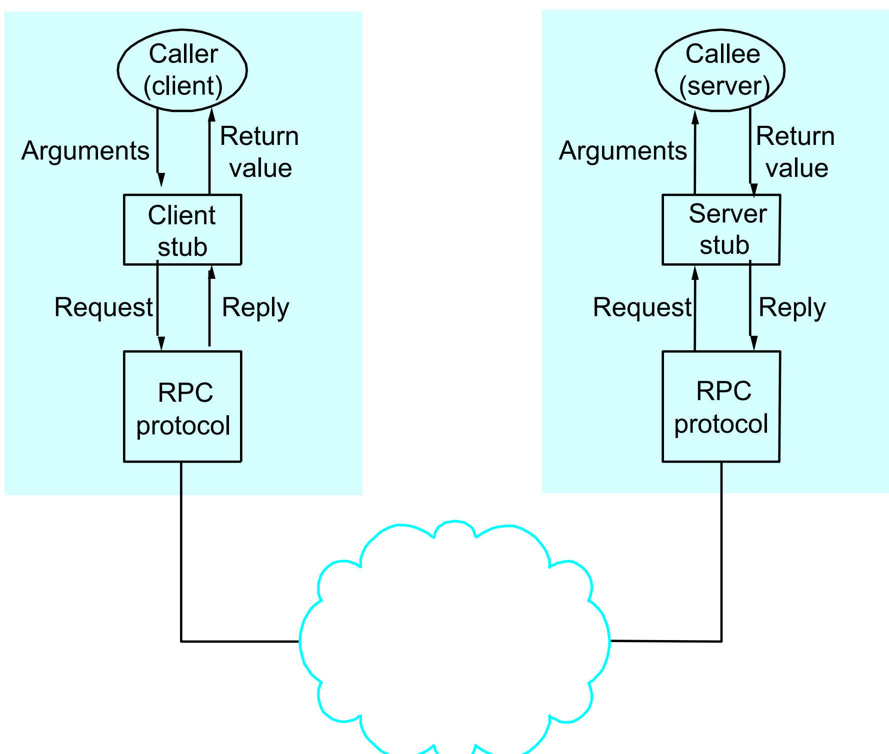
- Stop and wait
- Blast (tutti i pacchetti sono inviati in sequenza ed il server invia un ack in ricezione dell’ultimo pacchetto)

Simple RPC Stack



RCP Components (an example)

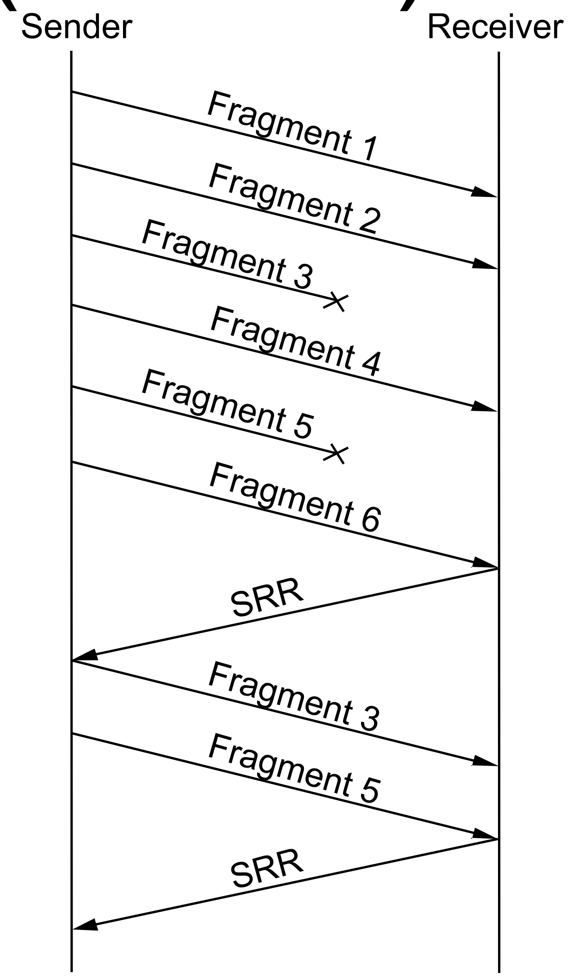
- Protocol Stack
  - BLAST: **fragments** and reassembles large messages
  - CHAN: synchronizes request and reply **messages** (at most once semantic)
  - SELECT: dispatches request to the **correct process**
- Stubs



Similar to SunRPC

Bulk Transfer (BLAST)

- Strategy
  - selective retransmission
  - partial acknowledgements
  - Use of three timers
    - DONE
    - LAST_FRAG
    - RETRY



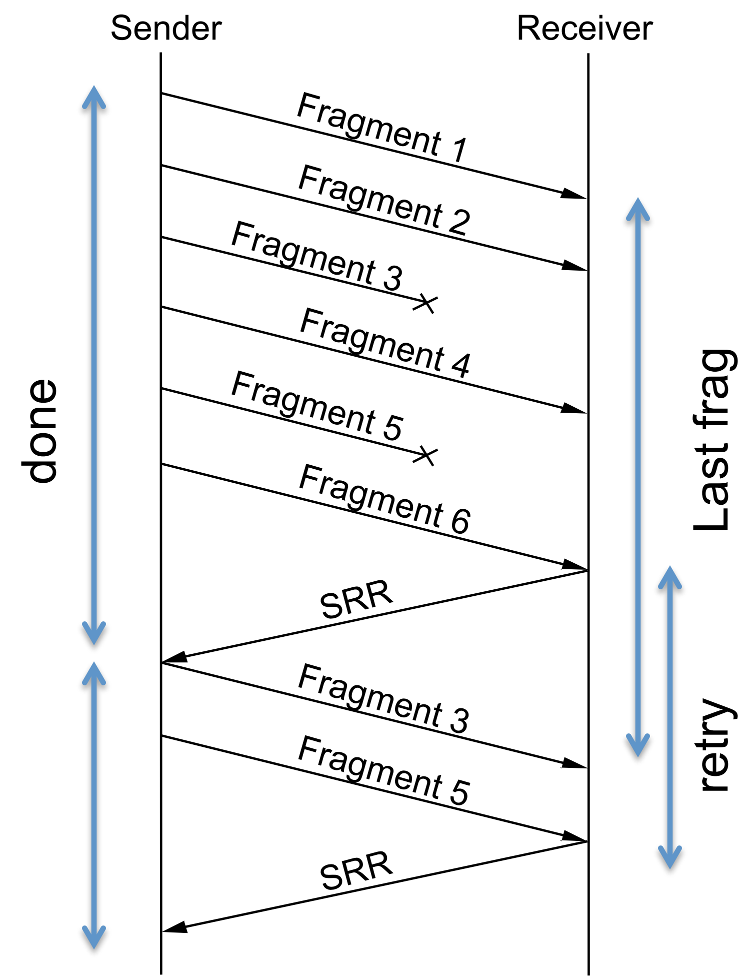
BLAST Details

- Sender:
  - Store fragments in local memory, send all fragments, set timer DONE
  - if receive SRR, send missing fragments and reset DONE
  - If receive SRR “all fragments have been received”, then sender frees fragments
  - if timer DONE expires, free fragments (sender gives up)

BLAST Details (cont)

- Receiver:
  - when first fragment arrives, set timer LAST_FRAG
  - when all fragments present, reassemble and pass up and send SRR back
  - four exceptional conditions:
    - if last fragment arrives but message not complete
      - send SRR and set timer RETRY
    - if timer LAST_FRAG expires
      - send SRR and set timer RETRY
    - if timer RETRY expires for first or second time
      - send SRR and set timer RETRY
    - if timer RETRY expires a third time
      - give up and free partial message

Bulk Transfer (BLAST)



BLAST (iii)

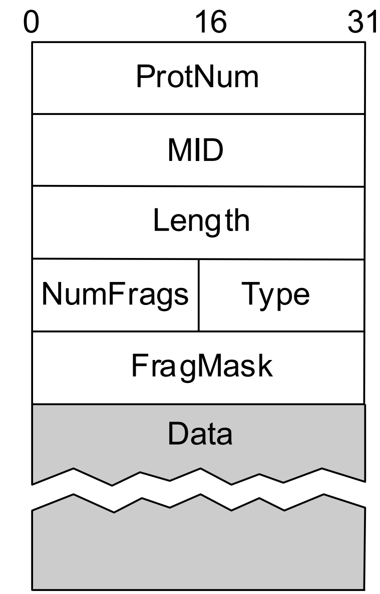
- Performance of BLAST in “nice” conditions does not depend on how carefully timers are set
  - DONE can be a fairly large value
  - RETRY is used to retransmit SRR messages. However when things are bad, performance is the last thing in mind.
  - LAST_FRAG is used to retransmit SRR messages when the last frag is dropped by the network (unlikely event)

BLAST (iv)

- BLAST is persistent in asking retrasmission of missing packets (designed to deliver large messages)
- BLAST does not guarantee anything on the delivery of the complete message. Assume a message composed by two fragments and these fragments are lost. The message will never be delivered. The sender’s DONE timer will expires and the sender gives up
- BLAST does not have capability to resend the complete message. This can be done by an upper layer protocol. Question: Why?
- Answer: preferable resending only those packets that are missing rather than having to retransmit the complete message when one fragment is lost

BLAST Header Format

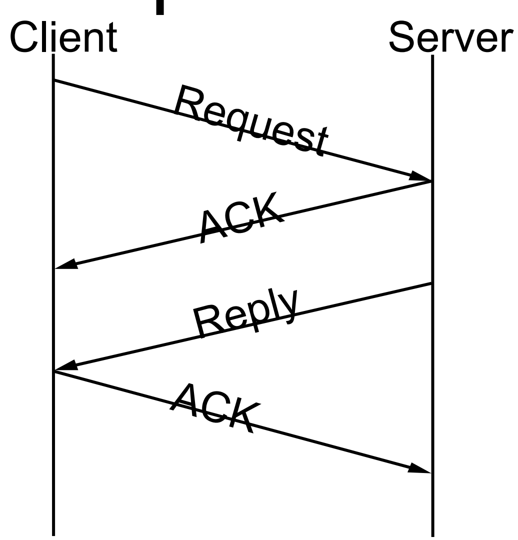
- MID must protect against wrap around (all fragments of a message have the same MID)
- TYPE = DATA or SRR
- NumFrags indicates number of fragments
- FragMask distinguishes among fragments
  - if Type=DATA, identifies this fragment
  - if Type=SRR, identifies missing fragments
  - Max 32 fragments per message



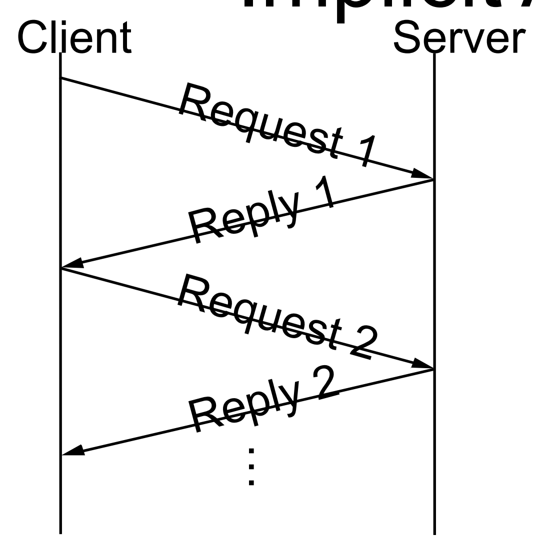
Request/Reply (CHAN)

- Guarantees message delivery
- Synchronizes client with server
- Supports *at-most-once* semantics

Simple case



Implicit Acks



CHAN Details

- To account opportunity of message loss, each message (REQ, REPLY) is stored till the ACK for it has arrived.
- Otherwise set a timer RETRANSMIT and and resend the message each time the timer expires
- Retrasmission implies message duplication at recipient side:
  - use message id (MID) field to distinguish

CHAN more Details

- Slow (long running) server
  - client periodically sends “are you alive” probe, or
  - server periodically sends “I’m alive” notice
- Want to support multiple outstanding calls
  - use channel id (CID) field to distinguish
- Machines crash and reboot
  - use boot id (BID) field to distinguish

- Use RETRANSMIT (client), RETRANSMIT (Server) and PROBE (Client)
- Retransmit on a LAN can be set to 20msec and a WAN suffer the same as TCP

CHAN Header Format

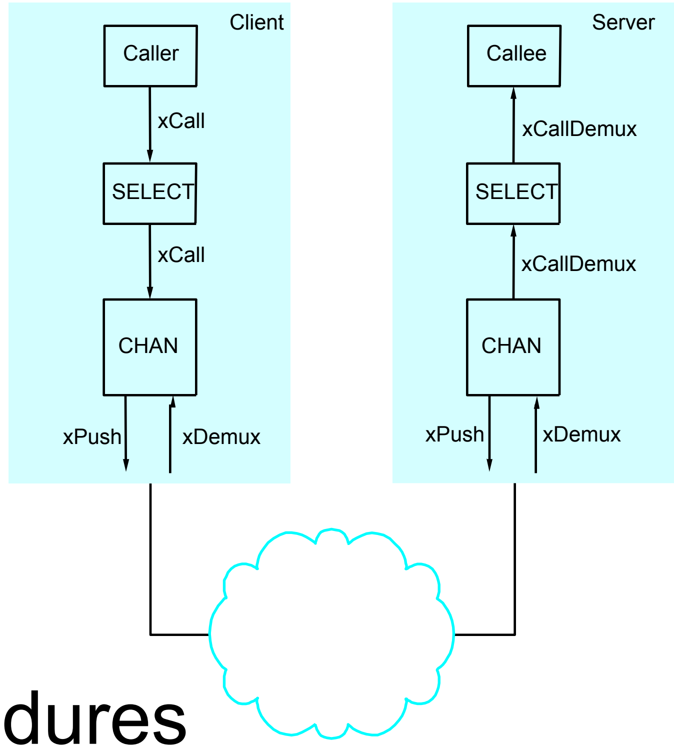
```
typedef struct {
   u_short  Type;    /* REQ, REP, ACK, PROBE */
   u_short  CID;     /* unique channel id */
   int      MID;     /* unique message id */
   int      BID;     /* unique boot id */
   int      Length;  /* length of message */
   int      ProtNum; /* high-level protocol */
} ChanHdr;


typedef struct {
   u_char    type;         /* CLIENT or SERVER */
   u_char    status;       /* BUSY or IDLE */
   int       retries;      /* number of retries */
   int       timeout;      /* timeout value */
   XkReturn  ret_val;      /* return value */
   Msg       *request;     /* request message */
   Msg       *reply;       /* reply message */
   Semaphore reply_sem;    /* client semaphore */
   int       mid;          /* message id */
   int       bid;          /* boot id */
} ChanState;
```

Differences between TCP/IP and CHAN/BLAST/IP

- Assume the network cannot be partitioned forever and that both sender and receiver do not fail
- Does CHAN/BLAST/IP protocol stack ensure a message will be delivered to the destination?
- Does TCP ensure this property?

Dispatcher (SELECT)

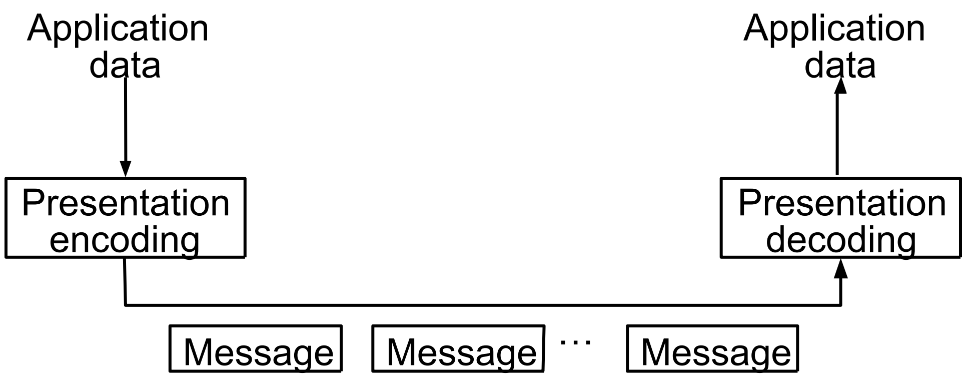
- Dispatch to appropriate procedure
- Synchronous counterpart to UDP



- Address Space for Procedures
  - flat: unique id for each possible procedure
  - hierarchical: program + procedure number

Presentation Formatting

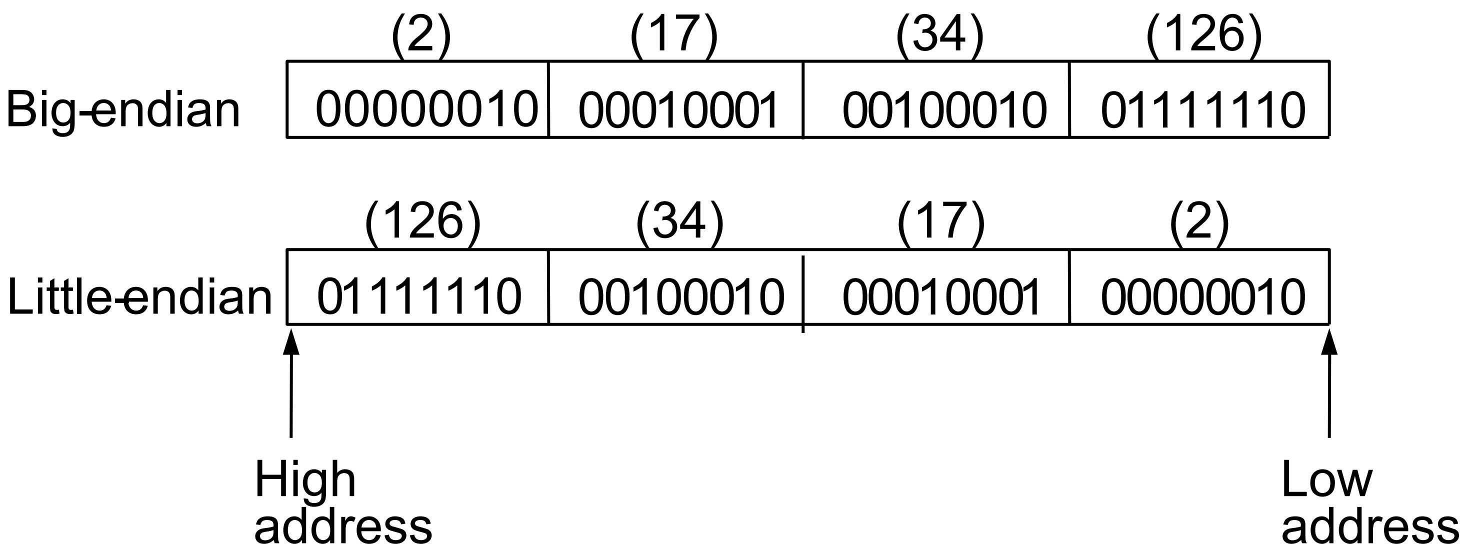
- Marshalling (encoding) application data into messages
- Unmarshalling (decoding) messages into application data



- Data types we consider
  - integers
  - floats
  - strings
  - arrays
  - structs

Difficulties

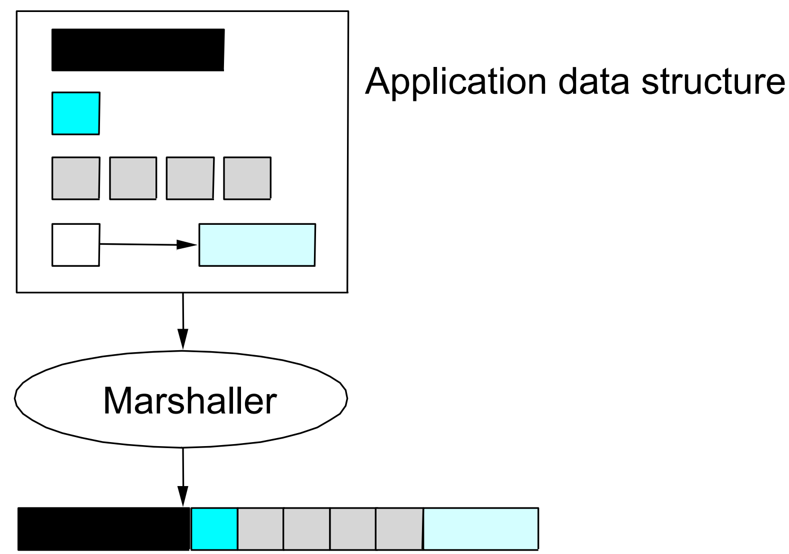
- Representation of base types
  - floating point: IEEE 754 versus non-standard
  - integer: big-endian versus little-endian



- Different representation of integers (1,2,4 bytes)

Taxonomy

- Data types
  - base types (e.g., ints, floats); must convert
  - flat types (e.g., structures, arrays); must pack
  - complex types (e.g., record); must linearize

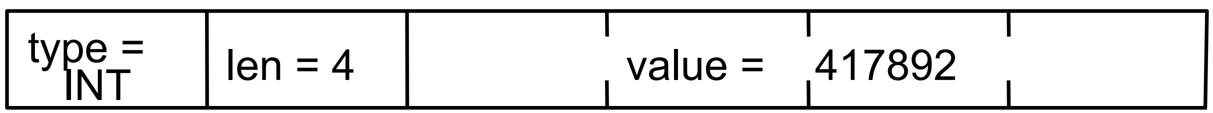


- Conversion Strategy
  - canonical intermediate form
  - receiver-makes-right (an *N* x *N* solution)

Taxonomy (cont)

- How a receiver knows which type of data is in the packet

- Tagged versus untagged data



  – Type, len, architecture

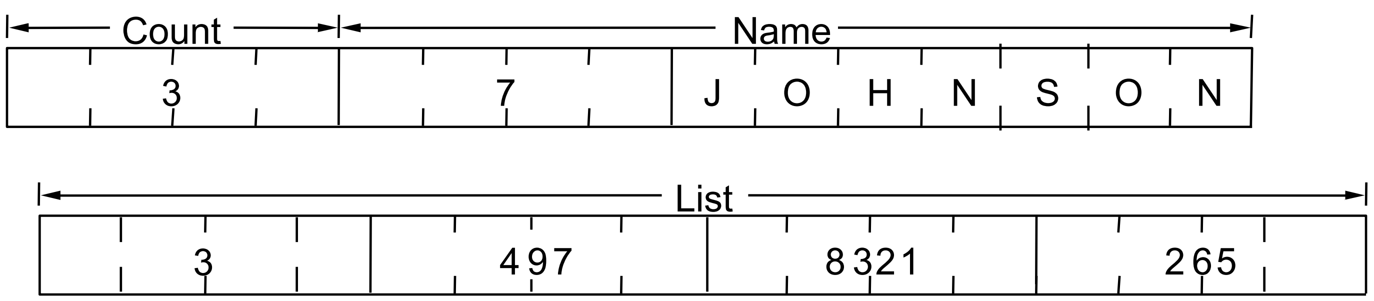
- Untagged data
  – No variable size data structures
  – End-to-end presentation formatting

eXternal Data Representation (XDR)

- Defined by Sun for use with SunRPC
- C type system
- Canonical intermediate form
- Untagged (except array length)

Example of encoding a string and a vector

```
#define MAXNAME 256;
#define MAXLIST 100;

struct item {
   int     count;
   char    name[MAXNAME];
   int     list[MAXLIST];
};
```

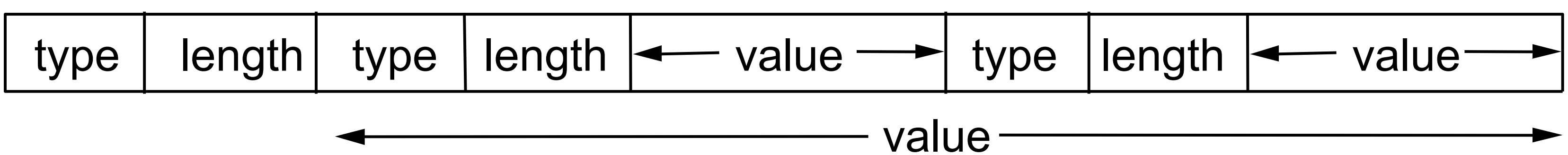


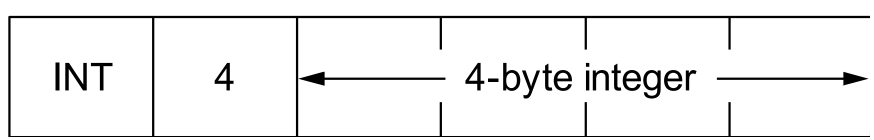
Abstract Syntax Notation One (ASN-1)

- An ISO standard
- Essentially the C type system
- Canonical intermediate form
- Tagged
- BER: Basic Encoding Rules

  `(tag, length, value)`

- Nested representation of data structures





Standard for SNMP

Network Data Representation (NDR)

- Defined by DCE
- Essentially the C type system
- Receiver-makes-right (architecture tag)
- Individual data items untagged

– IntegerRep
  - 0 = big-endian
  - 1 = little-endian

– CharRep
  - 0 = ASCII
  - 1 = EBCDIC

– FloatRep
  - 0 = IEEE 754
  - 1 = VAX
  - 2 = Cray
  - 3 = IBM

0	4	8	16	24	31
IntegrRep	CharRep	FloatRep	Extension 1	Extension 2	