SOFTWARE ARCHITECTURE
8. NETWORK SYSTEM

Tatsuya Hagino
hagino@sfc.keio.ac.jp

lecture URL

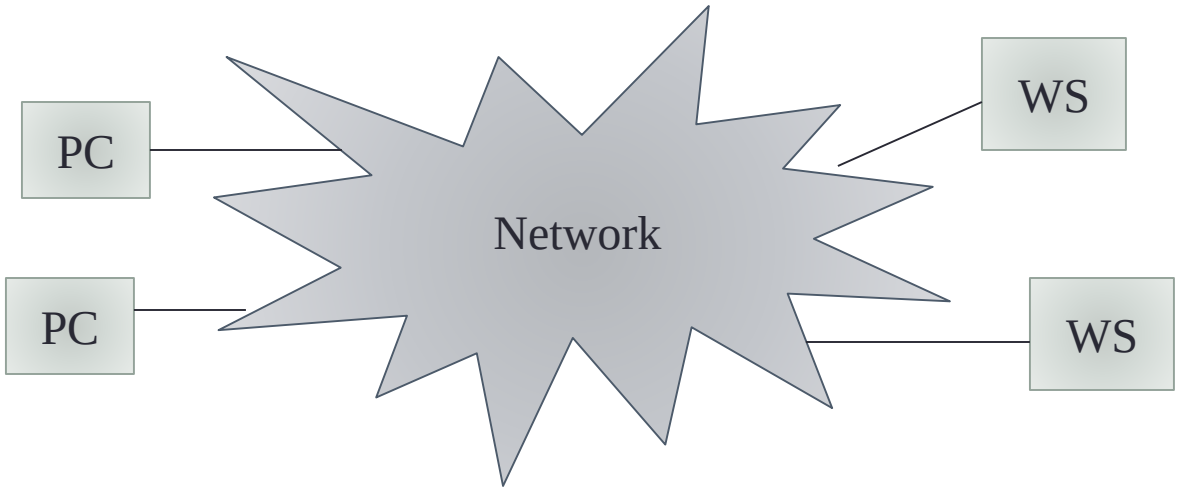
https://vu5.sfc.keio.ac.jp/slide/

Software

- Fundamental software
  - Operating System
- Standalone software
  - shell
  - compiler
  - interpreter
- Network software
  - WEB
  - electric mail
  - chat / instant message
  - IP telephone / VOIP

Distributed Systems

- Multiple autonomous computers cooperate to solve some problem by communicating each other through a computer network.
  - ``Distributed Systems, 5th edition'' by George Coulouris, Jean Dollimore, Tim Kindberg from Pearson Education



Example of Distributed Systems

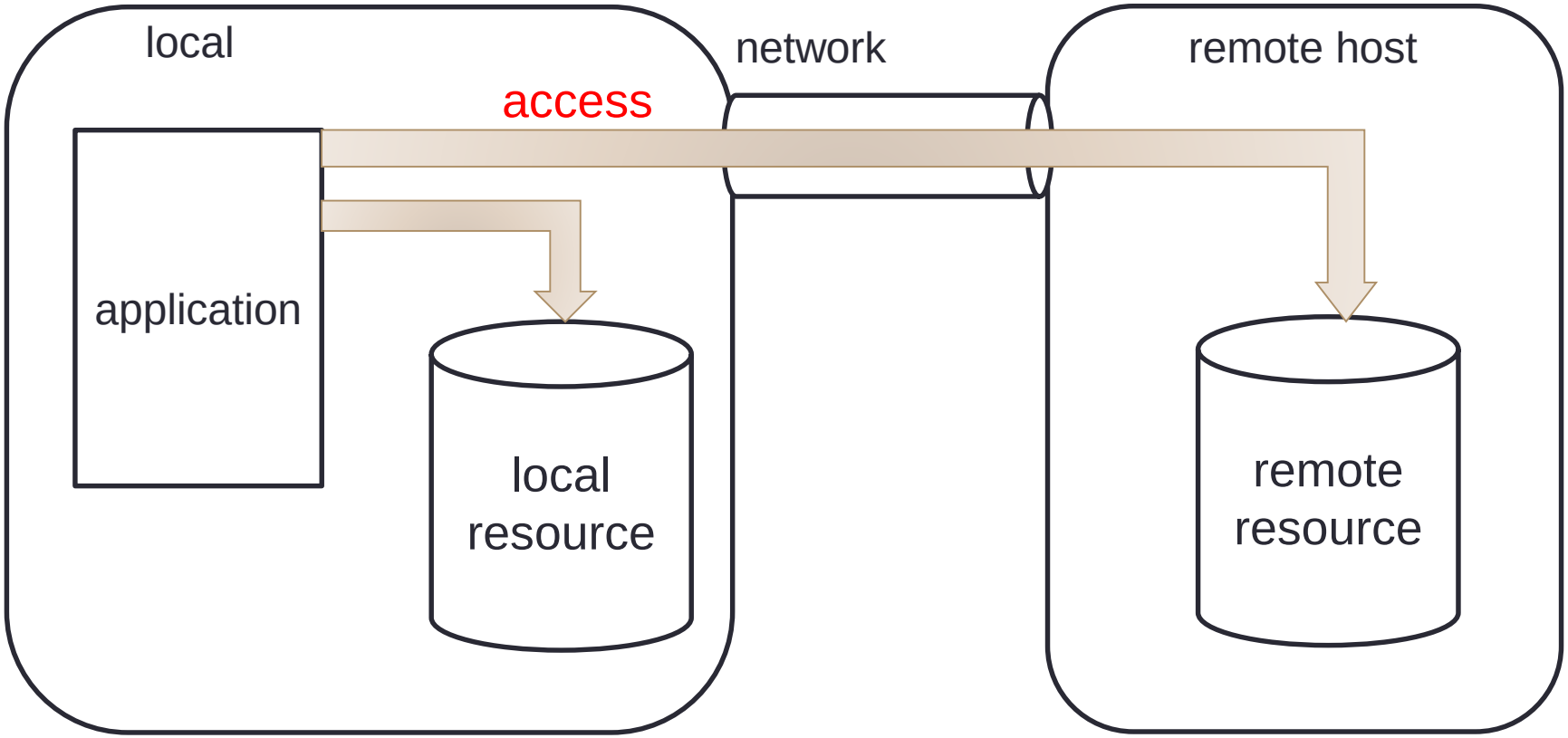
- Network applications
  - electric mail
  - electric news
  - World Wide Web
- Business systems
  - airplane ticket reservation system
  - ATM (Automatic Teller Machine) at banks
  - warehouse management system
- PC connected to LAN
  - file sharing
  - printer sharing
  - remote access
- Teleconference systems
  - e-learning
  - H.323 teleconference
  - CSCW (Computer Supported Cooperative Work)

Distributed Systems

- Resource sharing
  - What resource is shared.
  - Who owns resource.
- Openness
  - Is everyone can join?
  - Close systems can never become large.
- Parallel processing
  - Multiple things happen at the same time.
  - Have to support parallel processing.
- Fault tolerance
  - Cannot assume all the machines work perfectly.
  - Need to cope with failure.
- Transparency
  - As if it were not distributed.
    - access transparency
    - location transparency
    - concurrent transparency
    - replication transparency
    - failure transparency
    - relocation transparency
    - performance transparency
    - scale transparency

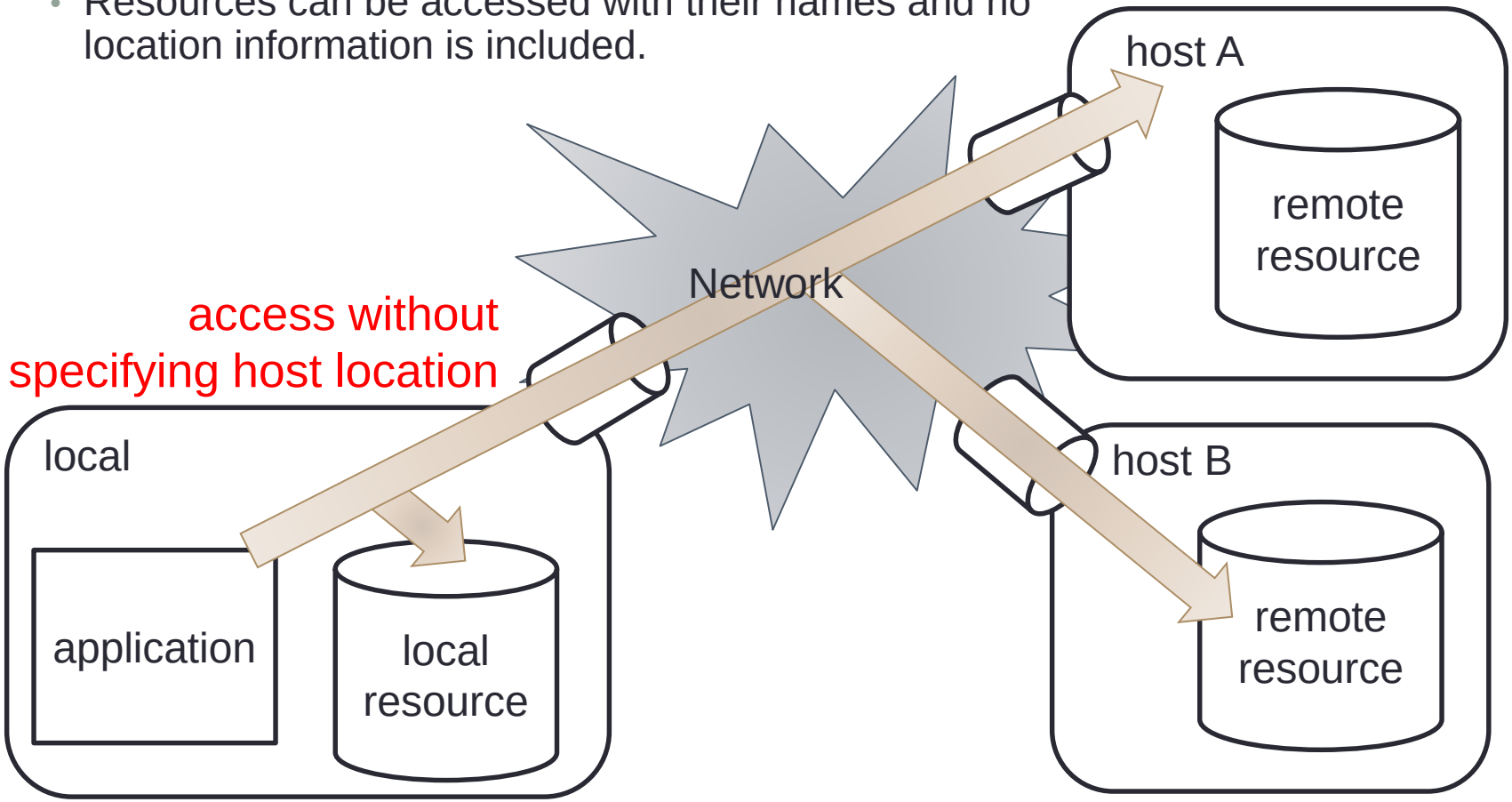
Access Transparency

- Access local and remote resources in the same way.
  - No special access method for remote resources.



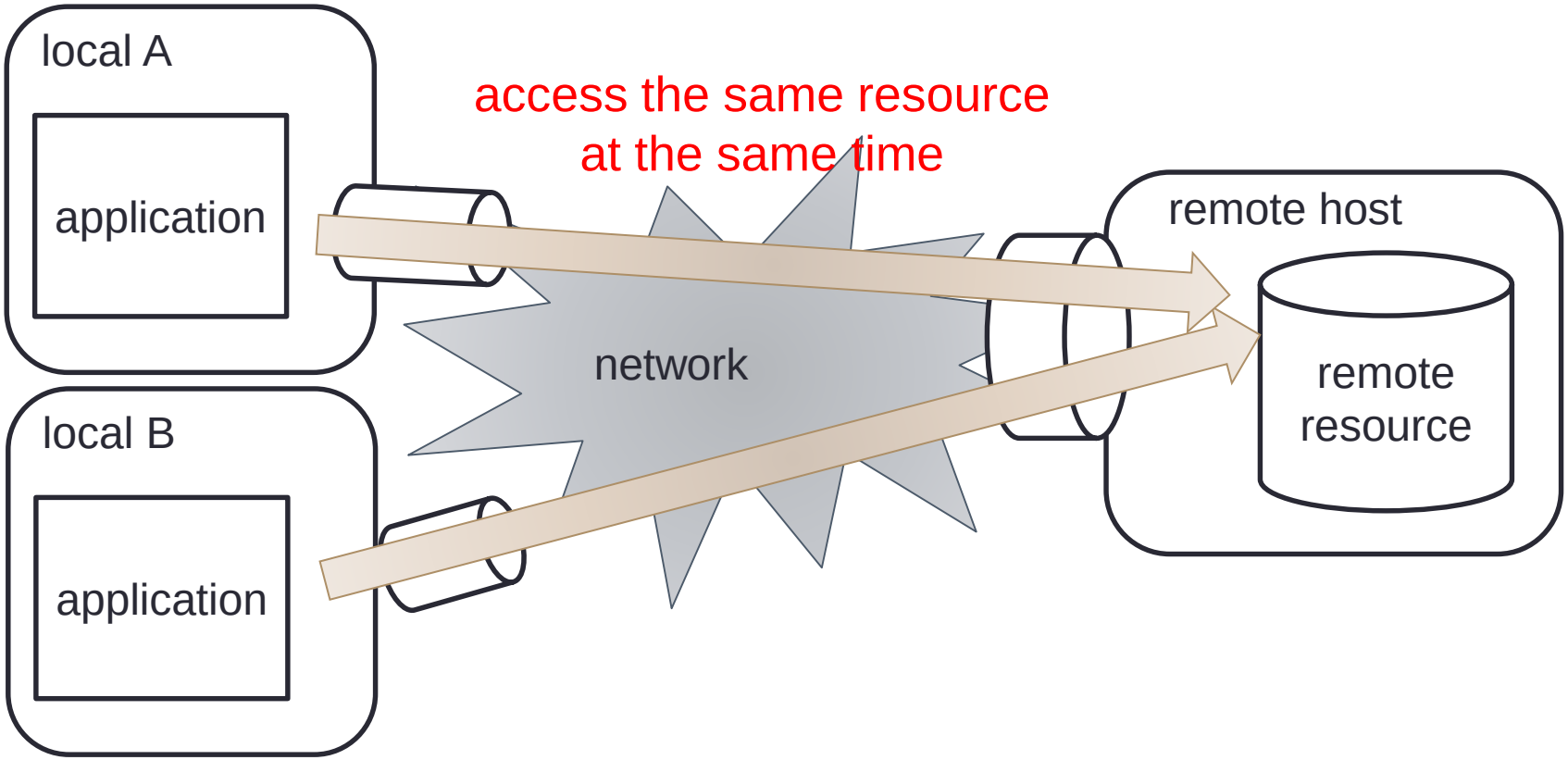
Location Transparency

- Access resources without knowing their location.
  - No need to specify the location to access.
  - Resources can be accessed with their names and no location information is included.



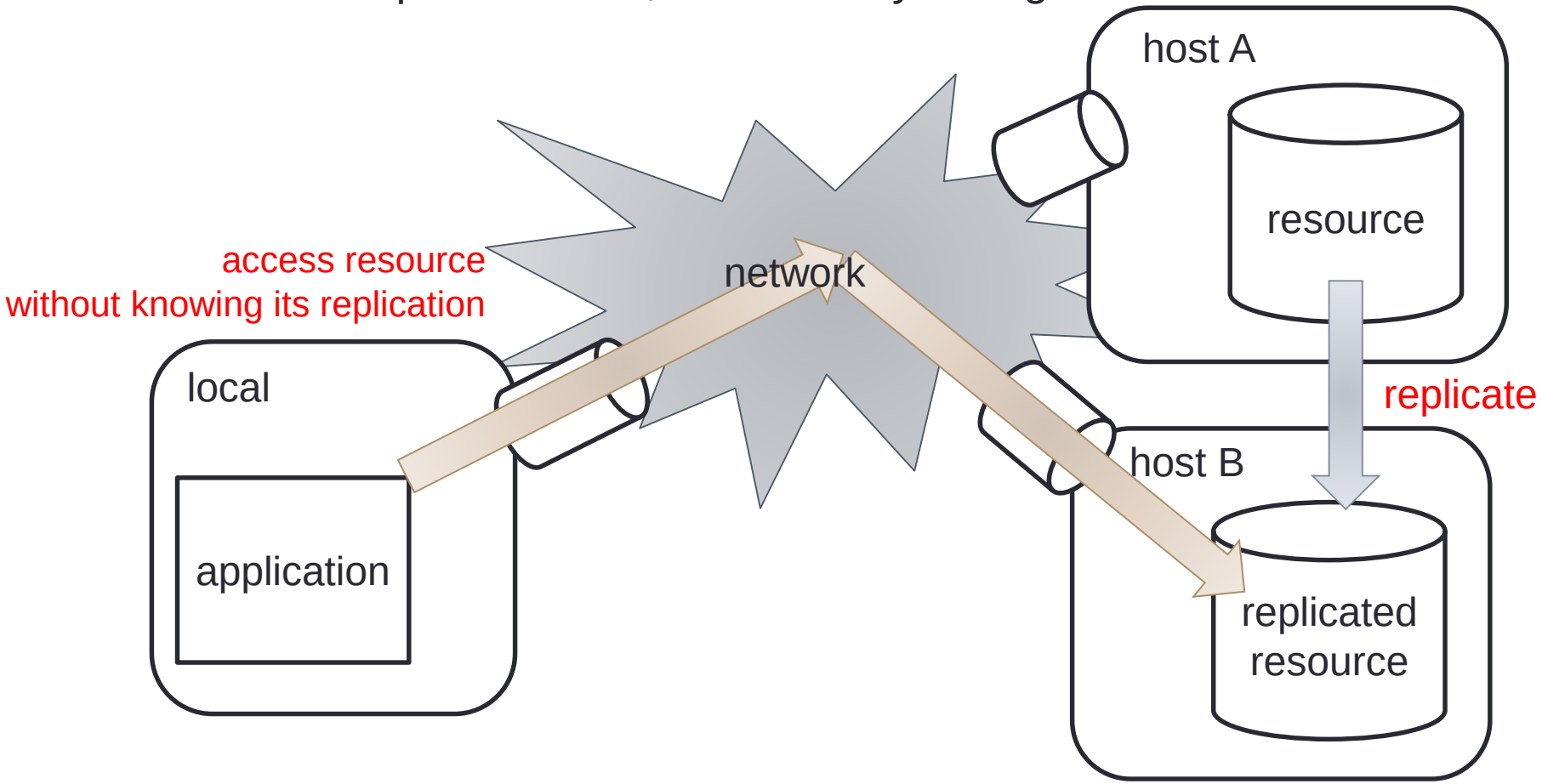
Concurrent Transparency

- Multiple access and manipulation at the same time.
  - No need to wait others to finish but use together.



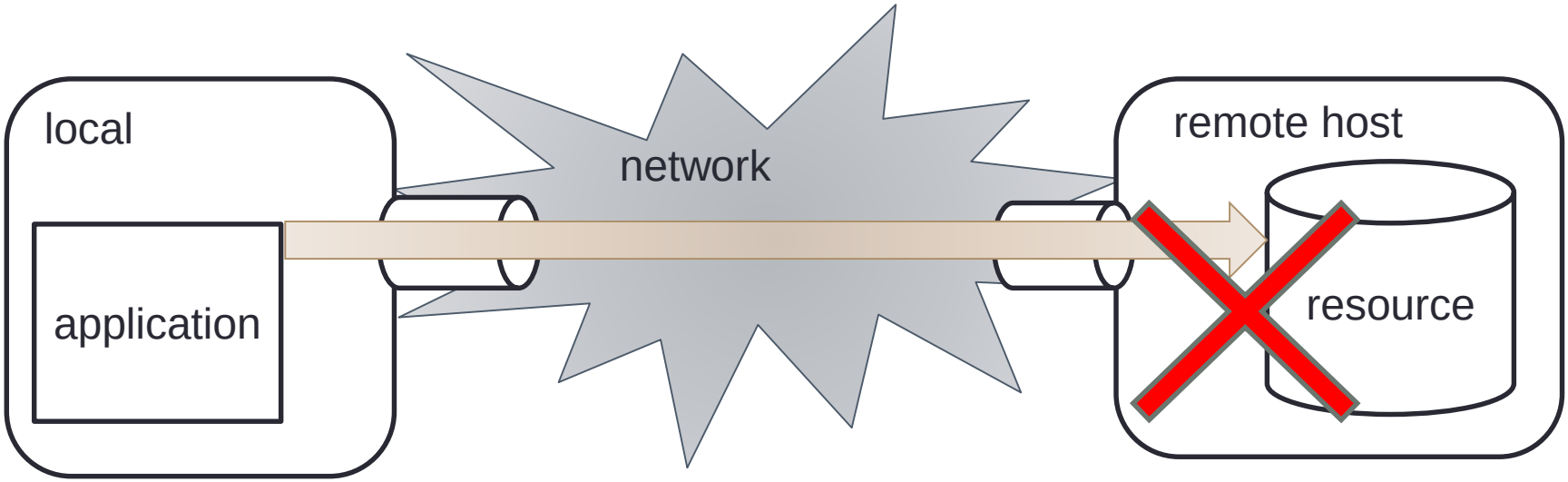
Replication Transparency

- Resource may be replicated among multiple locations.
  - From user's point of view, there is only a single resource.



Failure Transparency

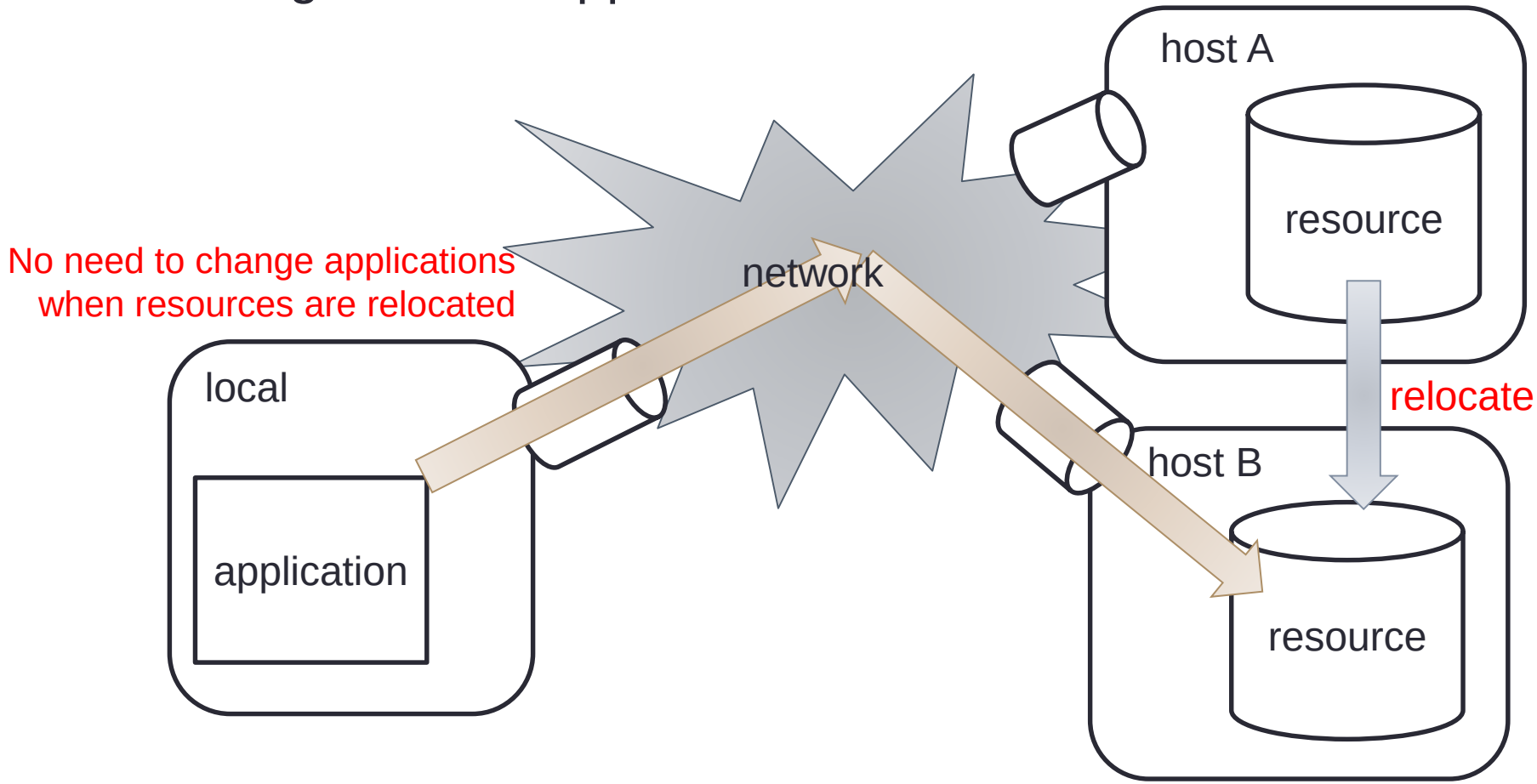
- Hide failure from users
  - Recover from failure automatically.



access and process
even when
hardware or software
failure happens

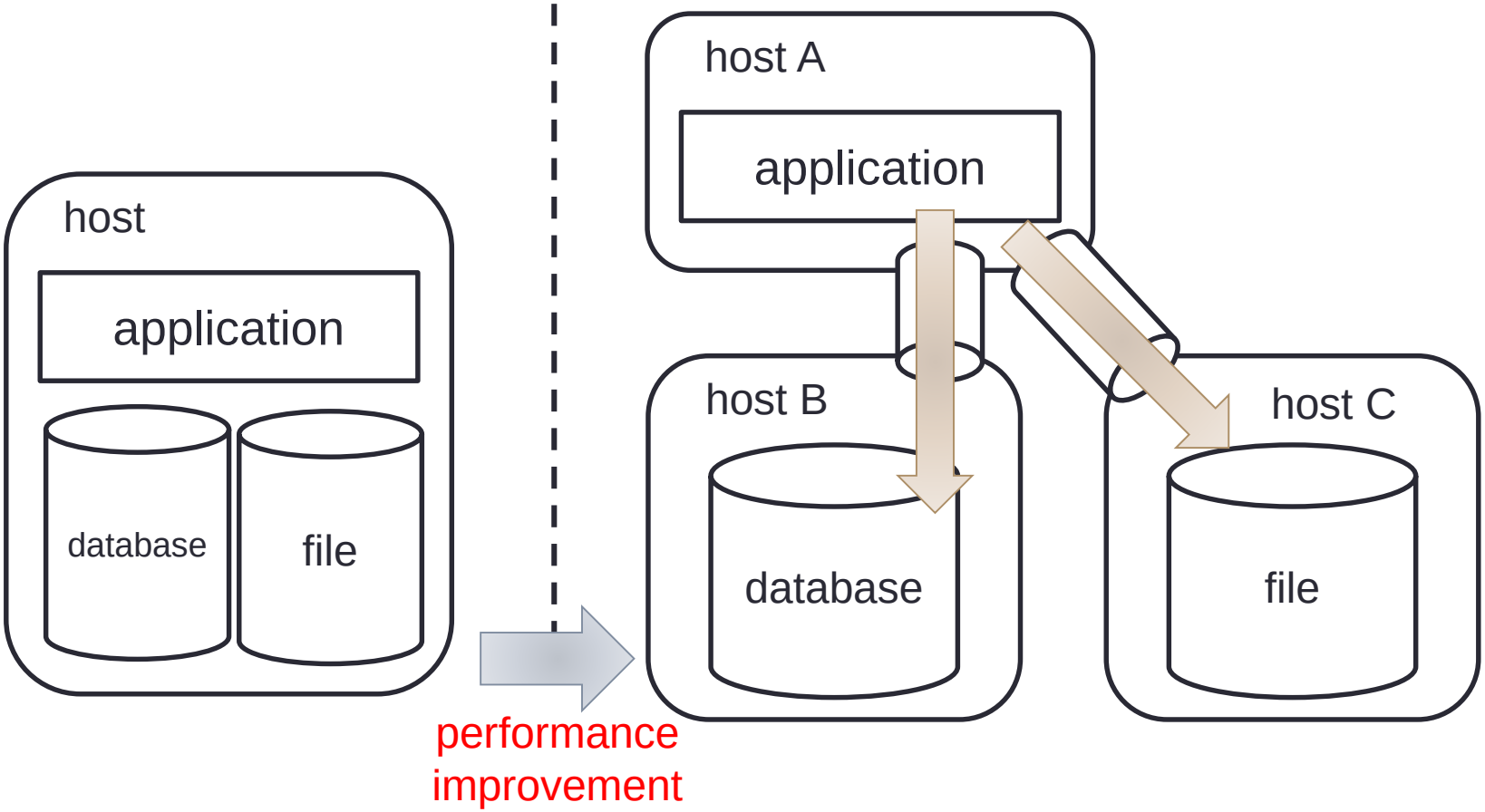
Relocation Transparency

- Resources can be moved to other location without affecting users or applications.



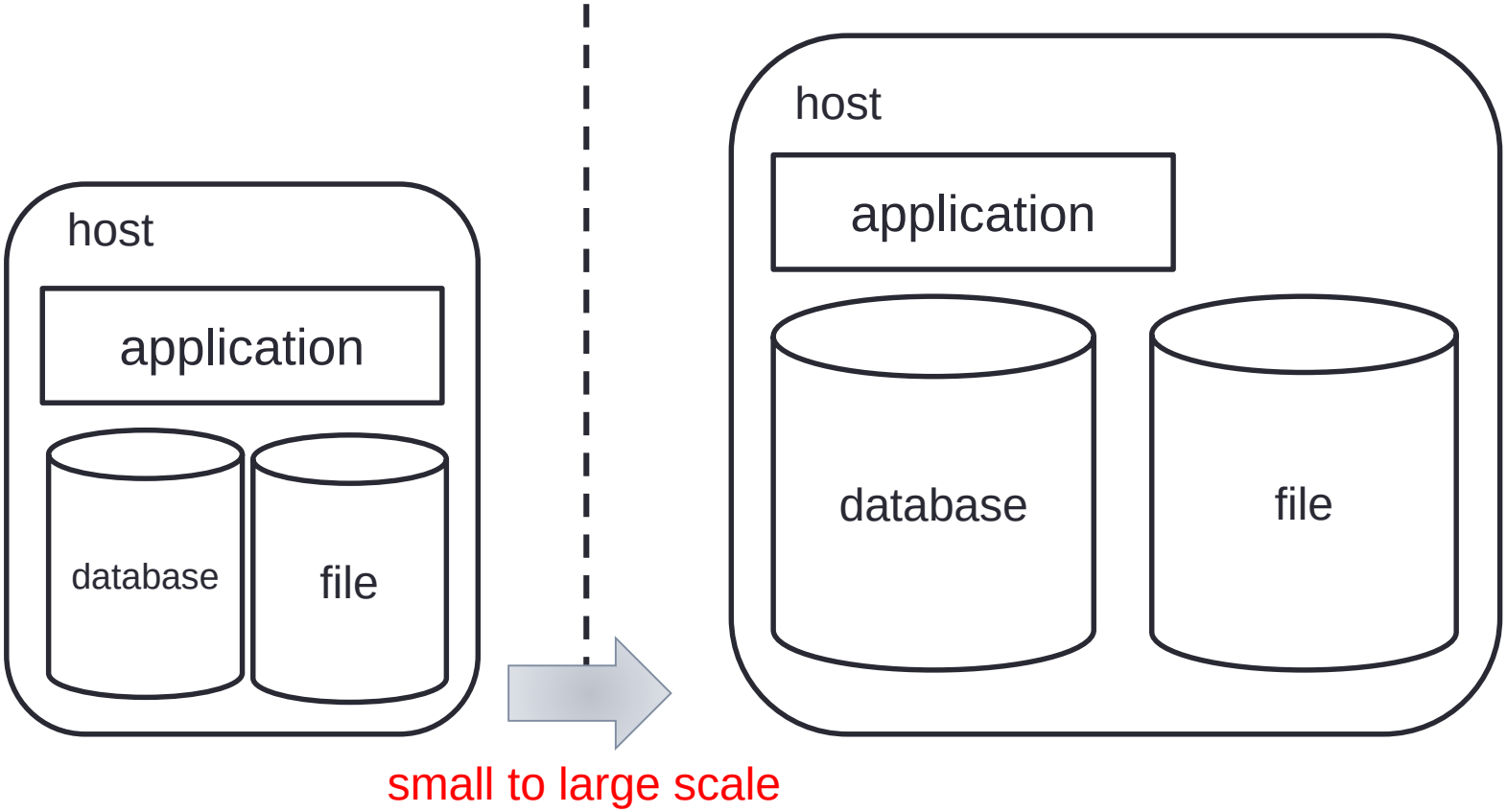
Performance Transparency

- System can be reconfigured in order to increase its performance.



Scale Transparency

- System can be scaled up without changing its application structure.



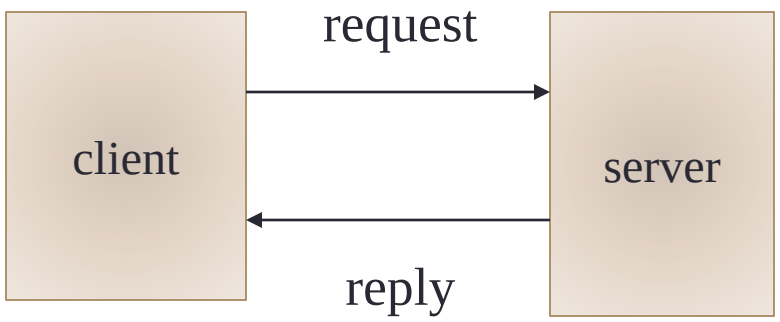
Communication in Distributed Systems

- Distributed systems need communication among distributed machines.

- Communication Model
  - Client Server Model
  - RPC
  - Function Shipping
  - Group Multicast
  - P2P

Client Server Model

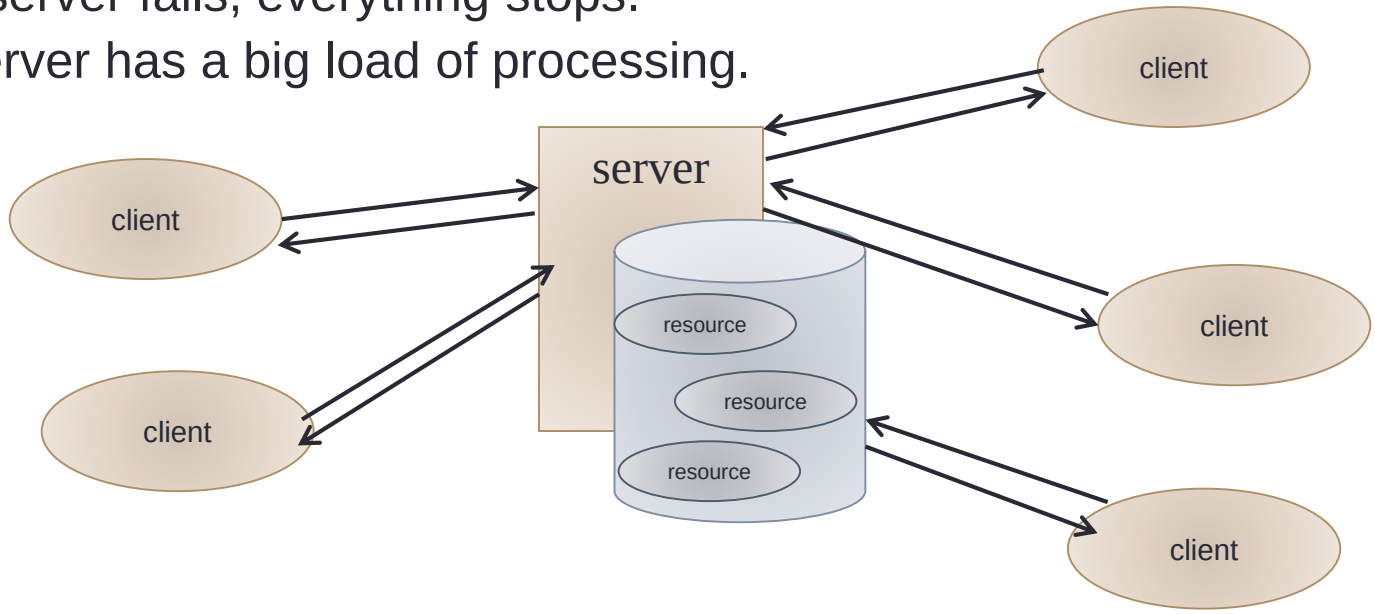
- Server
  - provide service
  - manage resources
- Client
  - request service
- Flow of process
  - A client send a request to a server,
  - The server process the request, and
  - The server replies to the client.



Client Server Model

- Merits
  - Easy to manage resources.
  - Shared resources may be updated correctly using locking in the server.

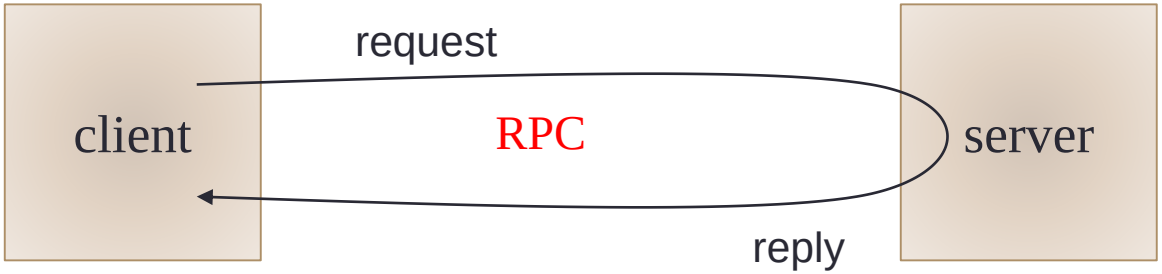
- Problems
  - Server centric
  - If server fails, everything stops.
  - Server has a big load of processing.



RPC (Remote Procedure Call)

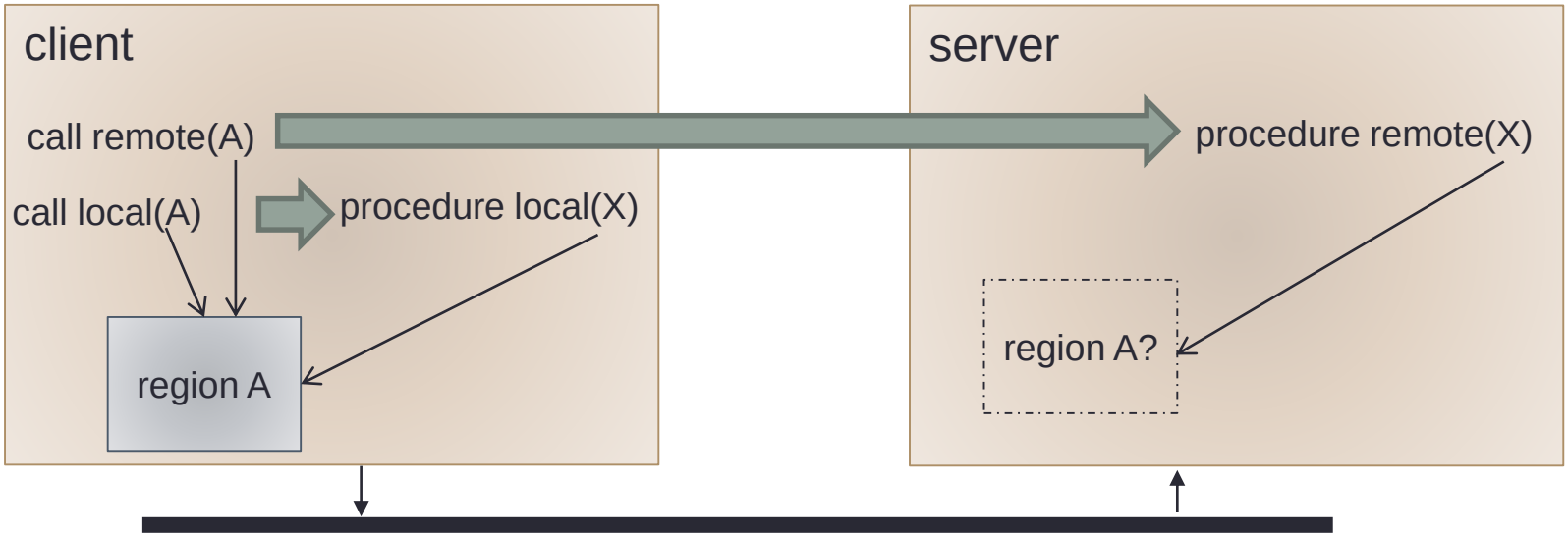
- Special case of client server model
  - In general, service request and reply does not need to match.
  - Usually, a client sends a request and it waits for the reply.
  - The server waits a request and usually replies after the request is processed.
  - It is as if a client calls a procedure in the server.

- RPC (Remote Procedure Call)
  - Remote procedure call compared with normal local procedure call.
  - For a client, it looks like an ordinary procedure call inside the same machine.
  - The difference is just the procedure is in remote host or local.
  - The underlying protocol can be optimized for RPC use.
    - The reply can be used to notify the acceptance of the request.



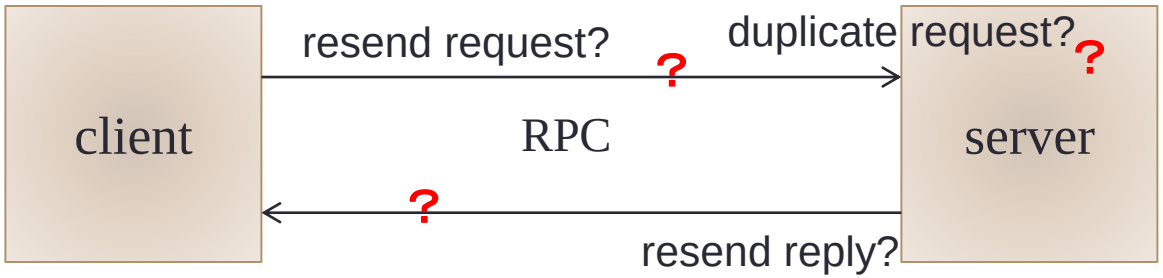
RPC vs Local Procedure (1)

- Pointers cannot be send to remote procedures.
  - Pointers are often used to pass large regions to local procedures (call-by-reference).
  - In RPC, clients and serves are not sharing the same memory space. Pointers cannot be referenced in servers.
  - RPC needs to use call-by-value.



RPC vs Local Procedure (2)

- Need to handle failure
  - Resend a request message?
    - The request message might not reach to the server.
  - Delete duplicate request messages?
    - If request messages are resent, the server may receive duplicate request messages and may need to delete them.
  - Resend a reply message?
    - The replay may not reach to the client.
    - The server needs to keep the reply in order for resend.



RPC Semantics

- Maybe Call
  - No resend of request messages.
    - No duplicate messages -> no need for deletion.
  - No resend of reply messages.

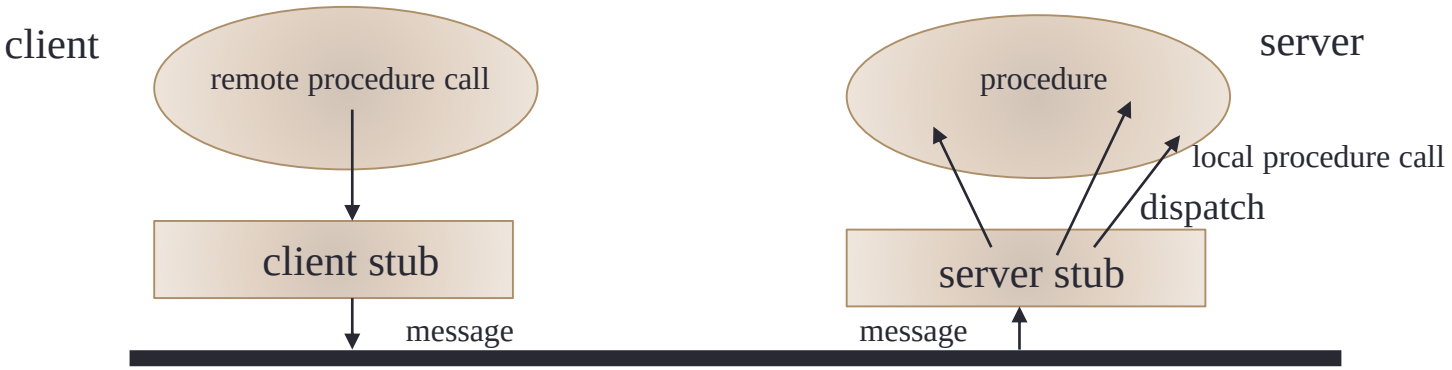
- At-Least-Once Call
  - Resend request messages.
  - No duplicate message deletion.
  - A client keeps sending the request message until it receives its reply.
  - The server processes the request at least once.
  - Good for a no side effect idempotent processing.

- At-Most-Once Call
  - Resend request messages.
  - The server checks the duplication of messages.
  - The server process the request at most once.
  - Good for a transaction processing.

RPC Implementation

- The underlying layer of RPC can be implemented for any RPC.
  - Create a request message.
  - Analyze a request message and calls the appropriate procedure.
  - Create a reply message.
  - Request and reply messages only depend on the interface (i.e. arguments and reply data type).

- Interface Definition Language
  - Specify RPC input output parameters
  - Generate stub automatically
    - Create messages
    - Analyze messages and invoke appropriate procedures.



RPC Interface Definition

- SunRPC interface definition

add.x

```
struct intpair { int a; int b; };
program ADD {
  version ADDVARS {
    int PRINT(string) = 1; /* procedure number = 1 */
    int ADD(intpair) = 2;  /* procedure number = 2 */
  } = 5;                   /* version number = 5 */
} = 0x20000099;            /* program number = 0x20000099 */
```

rpcgen -C add.x

add_clnt.c

```
...
int *print_5(char **argp, CLIENT *clnt)
{
  static int clnt_res;

  memset((char *)&clnt_res, 0, sizeof (clnt_res));
  if (clnt_call(clnt, PRINT,
     (xdrproc_t) xdr_wrapstring, (caddr_t) argp,
     (xdrproc_t) xdr_int, (caddr_t) &clnt_res,
     TIMEOUT) != RPC_SUCCESS) {
       return (NULL);
  }
  return (&clnt_res);
}
```

add_svc.c

```
...
add_5(struct svc_req *rqstp, SVCXPRT *transp)
{
  union {
    char *print_5_arg;
    intpair add_5_arg;
  } argument;
  char *result;
  ...
  switch (rqstp->rq_proc) {
  case PRINT:
      xdr_argument = (xdrproc_t) xdr_wrapstring;
      xdr_result = (xdrproc_t) xdr_int;
      local = (char *(*)(char *, struct svc_req
*)) print_5_svc;
       break;
```

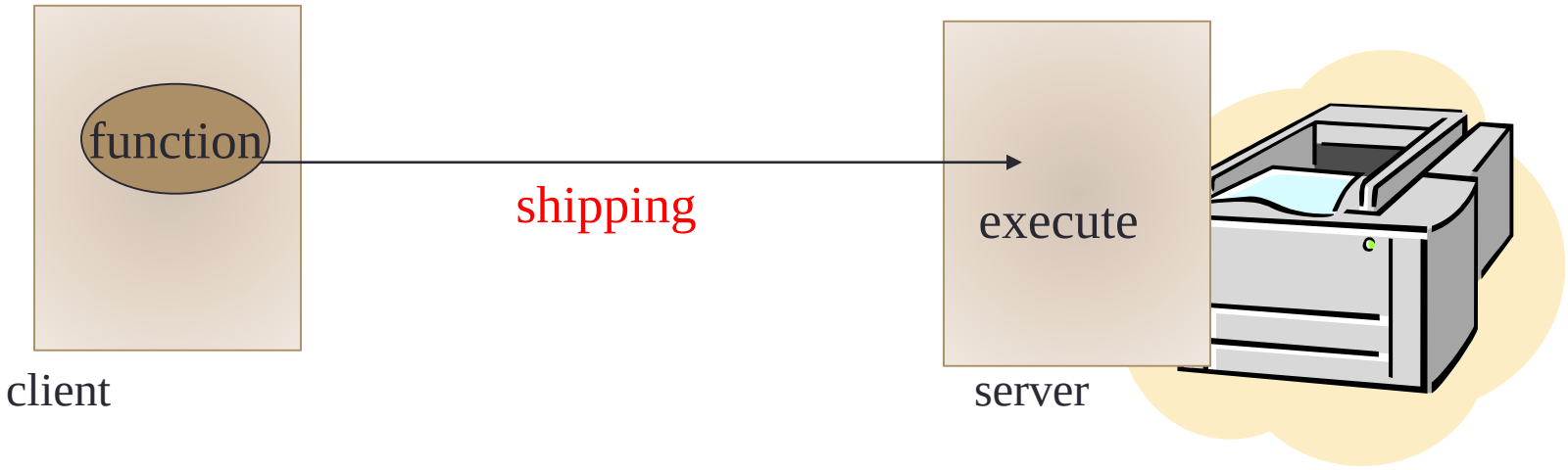
RPC

- Merits
  - Programs can use remote procedures in the same way as local procedures.
  - Stubs are generated automatically from interface definition.
- Problems
  - Need to handle failure.
  - Need to define interface beforehand.
    - Cannot call remote procedures which are not defined.
  - Procedures are called one by one.
    - Cannot be combined.

Function Shipping

- Send a set of instructions (or a program) rather than request.
  - Not limited to specific procedures.
  - Multiple processes may be packed as one program.
  - Sever is an interpreter of the instructions.

- Example
  - PostScript printer
  - NeWS window system (Display PostScript)



PostScript

- Page description language
  - Drawing instructions
  - Stack oriented programming language
  - Use reverse Polish notation.

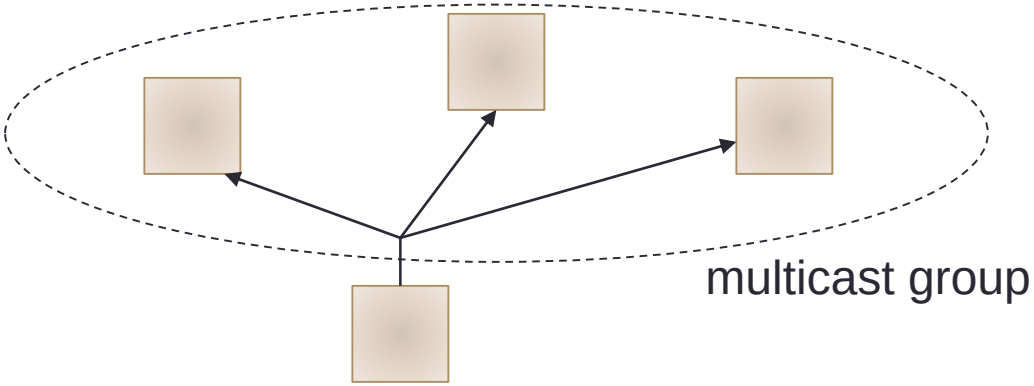
```
%!
% macro (draw rectangle) ; usage: left top width height RRECT
/RRECT { newpath 4 copy pop pop moveto dup 0 exch rlineto exch 0 rlineto
neg 0 exch rlineto closepath pop pop } def

100 100 100 150 RRECT
.5 setgray
fill

100 300 moveto
/Helvetica findfont
12 scalefont
setfont
.5 0 .5 0 setcmykcolor
(test string) show

showpage
```

Asymmetricity of Client Server Model

- Server needs to handle resource management.
  - Easy to implement.
  - Server becomes bottle neck.
  - Server needs to be protected from hackers.
- no. of servers <<< no. of clients
- Sever is usually huge.
  - Client has mobility.
  - Cannot process anything without connecting to server.

Group Multicast

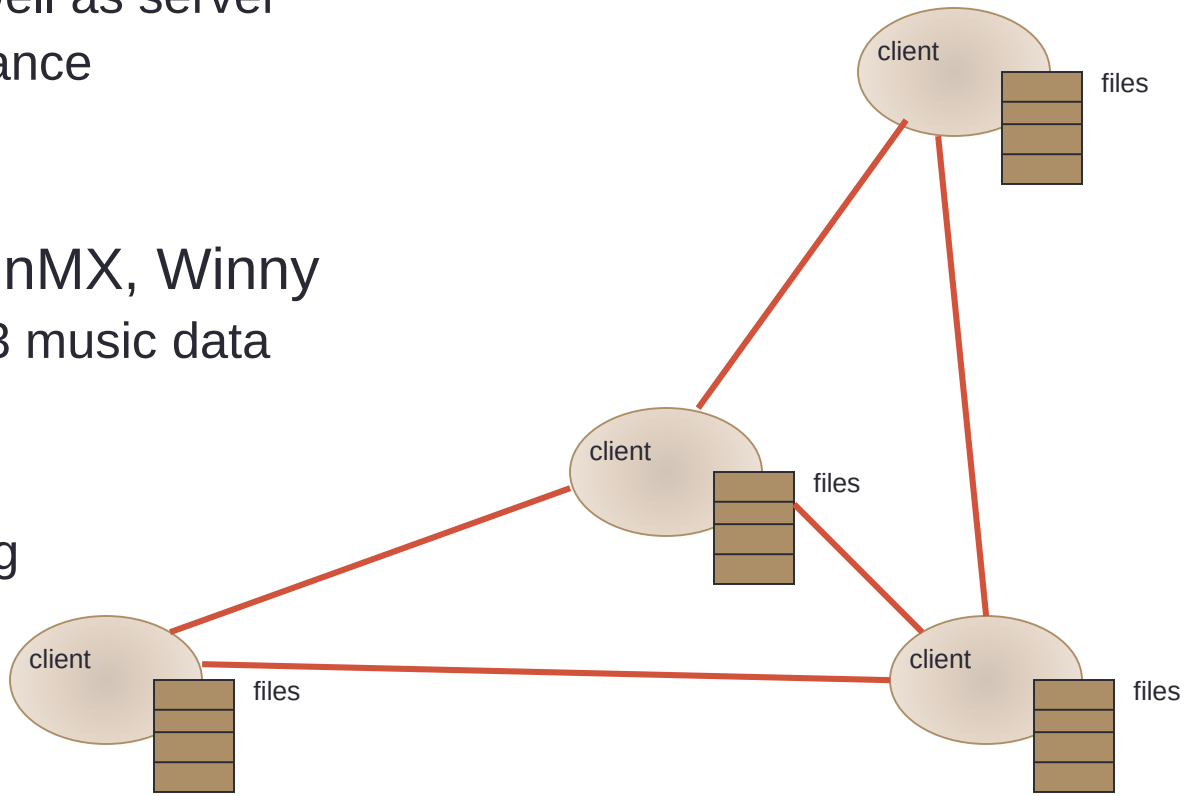
- Communicate with multiple machines at the same time.



- Find resource location
  - Multicast a request and one server with the resource replies.
- Update replicated resources at once.
  - Multicast update request to multiple servers at once.
- Fault tolerance
  - A group is acting as a server.
  - Still works even if one server fails.

P2P (Peer to Peer)

- Not asymmetric like client server model.
- Clients communicate each other.
  - Client as well as server
  - Fault tolerance
  - Anonymity
- Napster, WinMX, Winny
  - Share MP3 music data
- Gnutella
  - File sharing



Summary

- Distributed Systems
- Transparency
  - access transparency
  - location transparency
  - concurrent transparency
  - replication transparency
  - failure transparency
  - relocation transparency
  - performance transparency
  - scale transparency
- Communication Model
  - Client Server Model
  - RPC
  - Function Shipping
  - Group Multicast
  - P2P