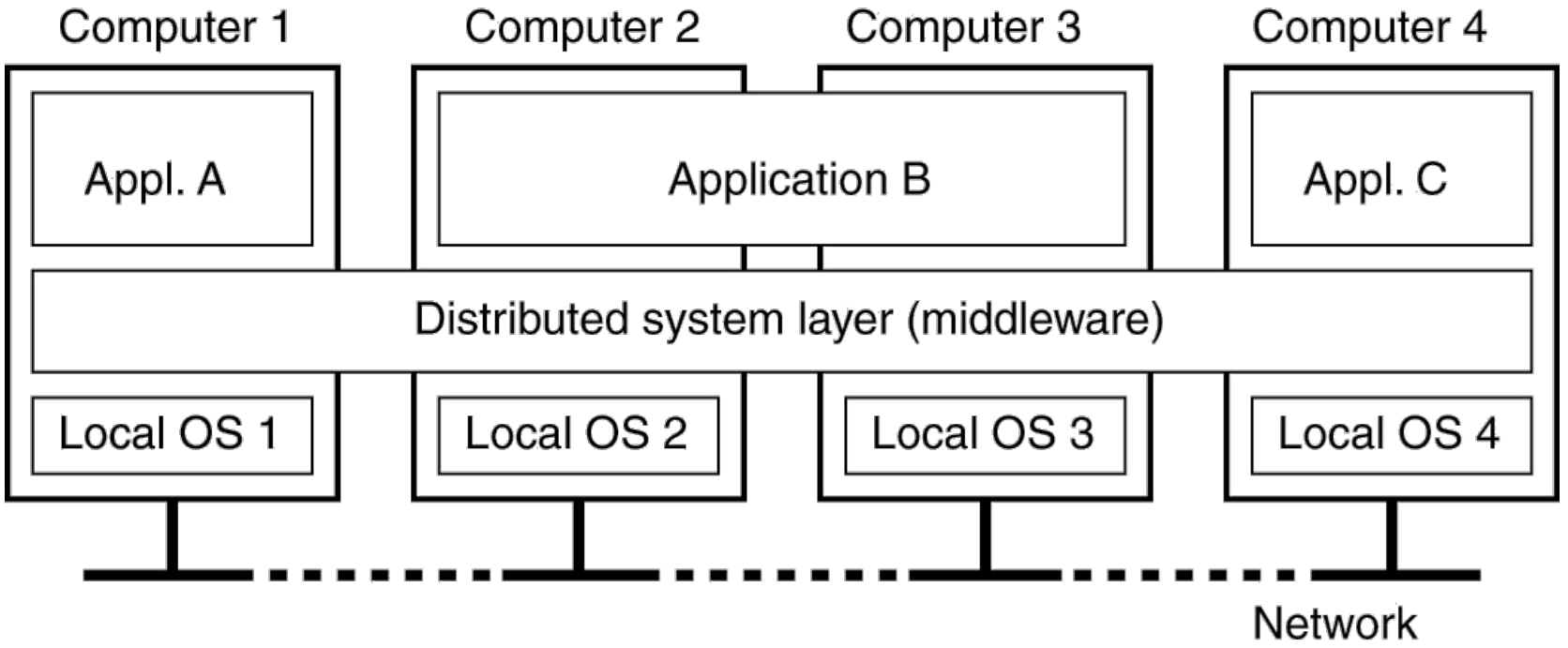
Now back to Distributed Systems: remember?

- A distributed system is:

"A collection of independent computers that appears to its users as a sin gle coherent system"

"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable." – Leslie La mport

Distributed Systems



- The middleware layer extends over multiple machine s, and offers each application the same interface.

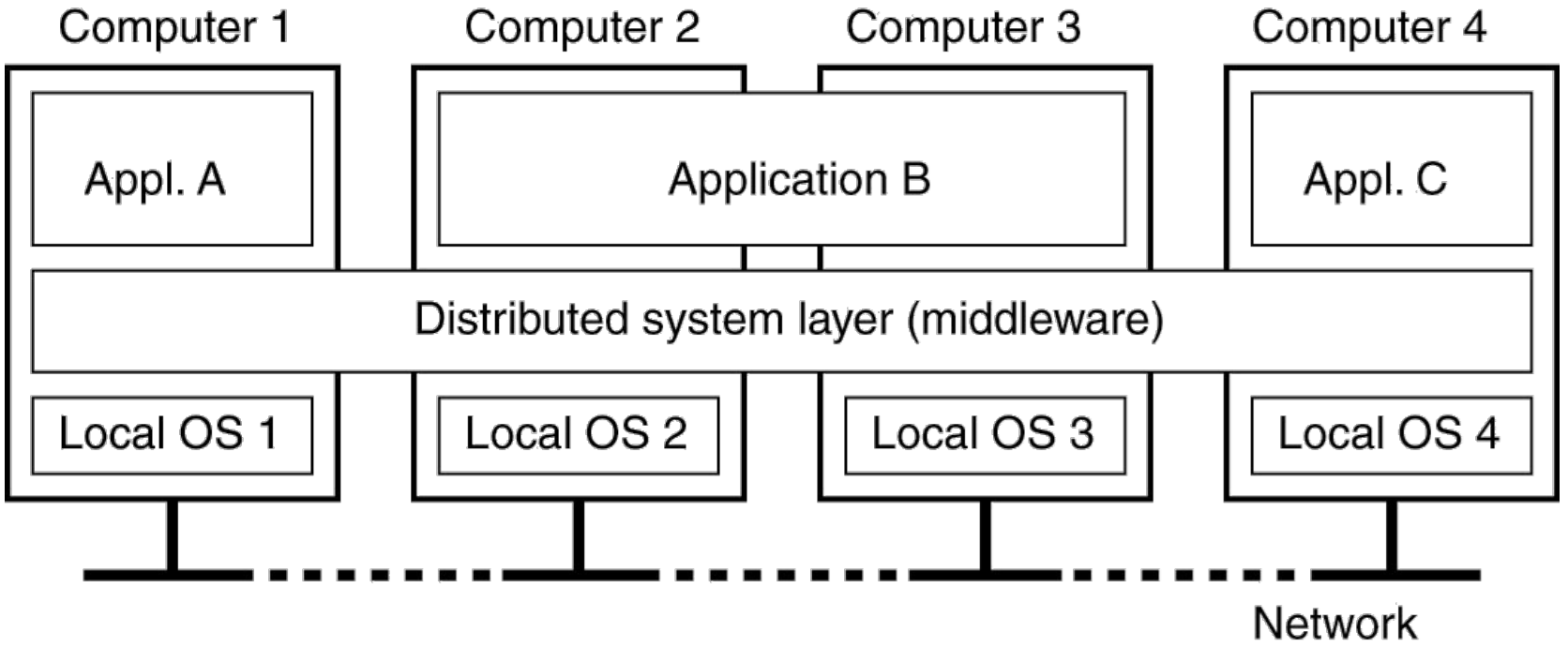
What does it do?

Hide complexity to programmers/users
Hide the fact that its processes and resources are physically distributed across multiple machines.

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource is replicated
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource

Transparency in a Distributed System

How?



- The middleware layer extends over multiple machine s, and offers each application the same interface.

Starter for Today

- Splitting computation across the network

What programming abstractions work well to split work among multiple networked computers?

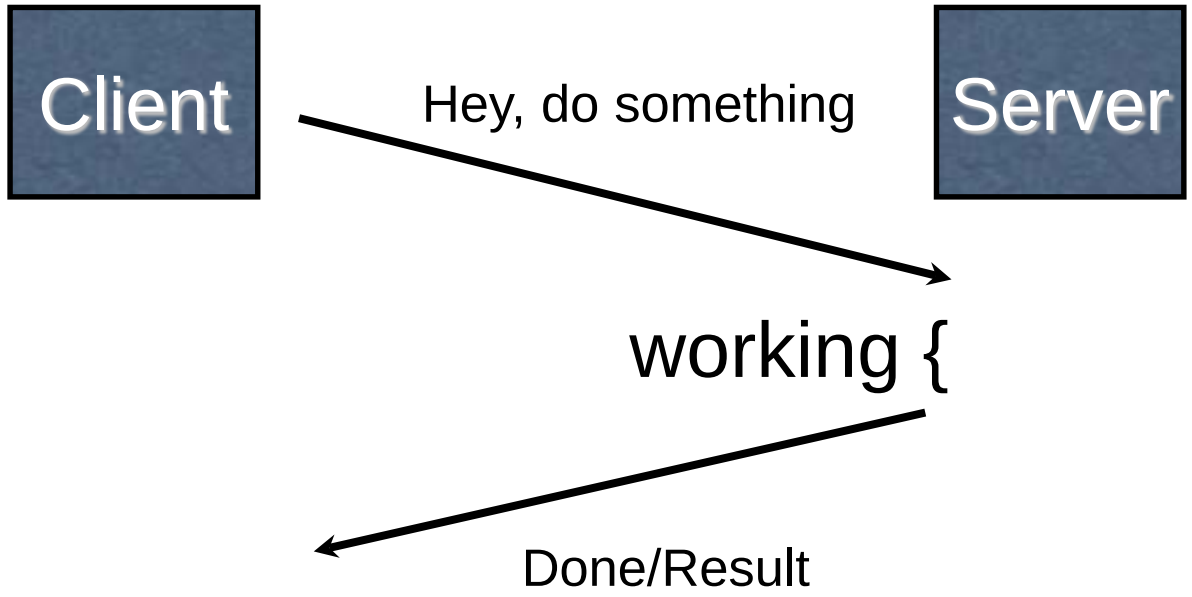
Many ways

- Request-reply protocols
- Remote procedure calls (RPC)
- Remote method invocation (RMI)

Recommended reading :

Distributed Systems: Concepts and Design 5th edition, by Coulouris, et al. (CDK5) chapter 5

Request-reply protocols



Request-reply protocols

- eg, your PA1 (binary protocol)

```
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, handle timeout, etc.

Request-reply protocols: text protocol

- HTTP
  - See the HTTP/1.1 standard:
  - `http://www.w3.org/Protocols/rfc2616/rfc2616.html`
  - Done with your PA2 yet?

Remote Procedure Call (RPC)

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

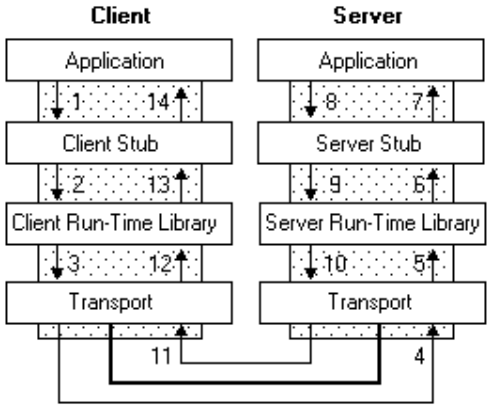


figure from Microsoft MSDN

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

RPC Goals

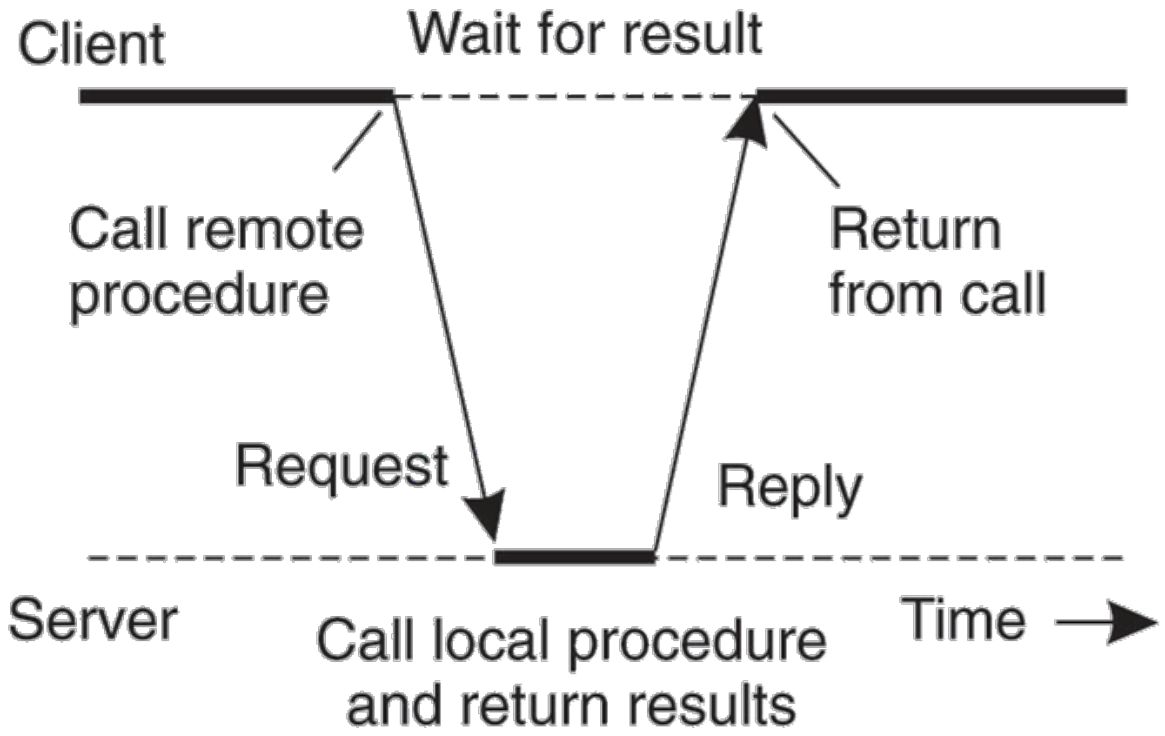
- Ease of programming
- Hide complexity
- Automate a lot of task of implementing
- Familiar model for programmers (just make a function call)

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

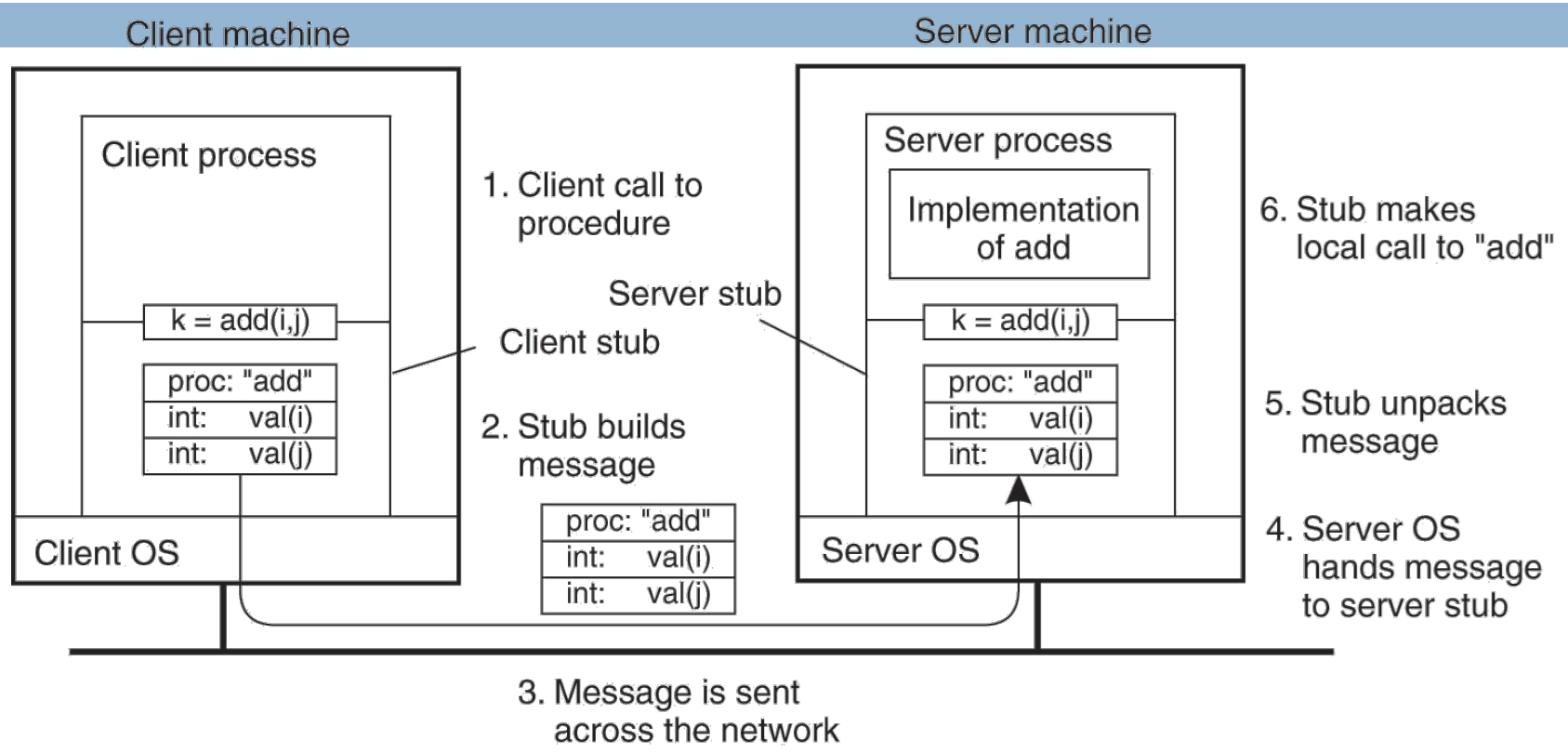
- A remote procedure call makes a call to a remote service look like a l ocal call
  - RPC makes transparent whether server is local or remote
  - RPC allows applications to become distributed transparently
  - RPC makes architecture of remote machine transparent

RPC



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

Passing Value Parameters (1)



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

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

Marshaling and Unmarshaling

- (From example) hotnl() -- “host to network-byte-order, long”.
  - network-byte-order (big-endian) standardized to deal with cross-platform variance
- Note how we arbitrarily decided to send the string by sending its length followed by L bytes of the string? That’s marshalling, too.
- Floating point…
- Nested structures? (Design question for the RPC system - do you support them?)
- Complex datastructures? (Some RPC systems let you send lists and maps as first-order objects)

“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 definition language.
  - Lots of these. Some look like C, some look like XML, … details don’t matter much.