Blockchain, Communication: Layering, RPC, Sockets

10 November 2026
Lecture 3

Slide Credits: Maarten van Steen

Topics for Today

- Blockchain basics
- Communication
  - Layered Communication
  - Types of Communication
- RPC and RMI
- Sockets

A little aside

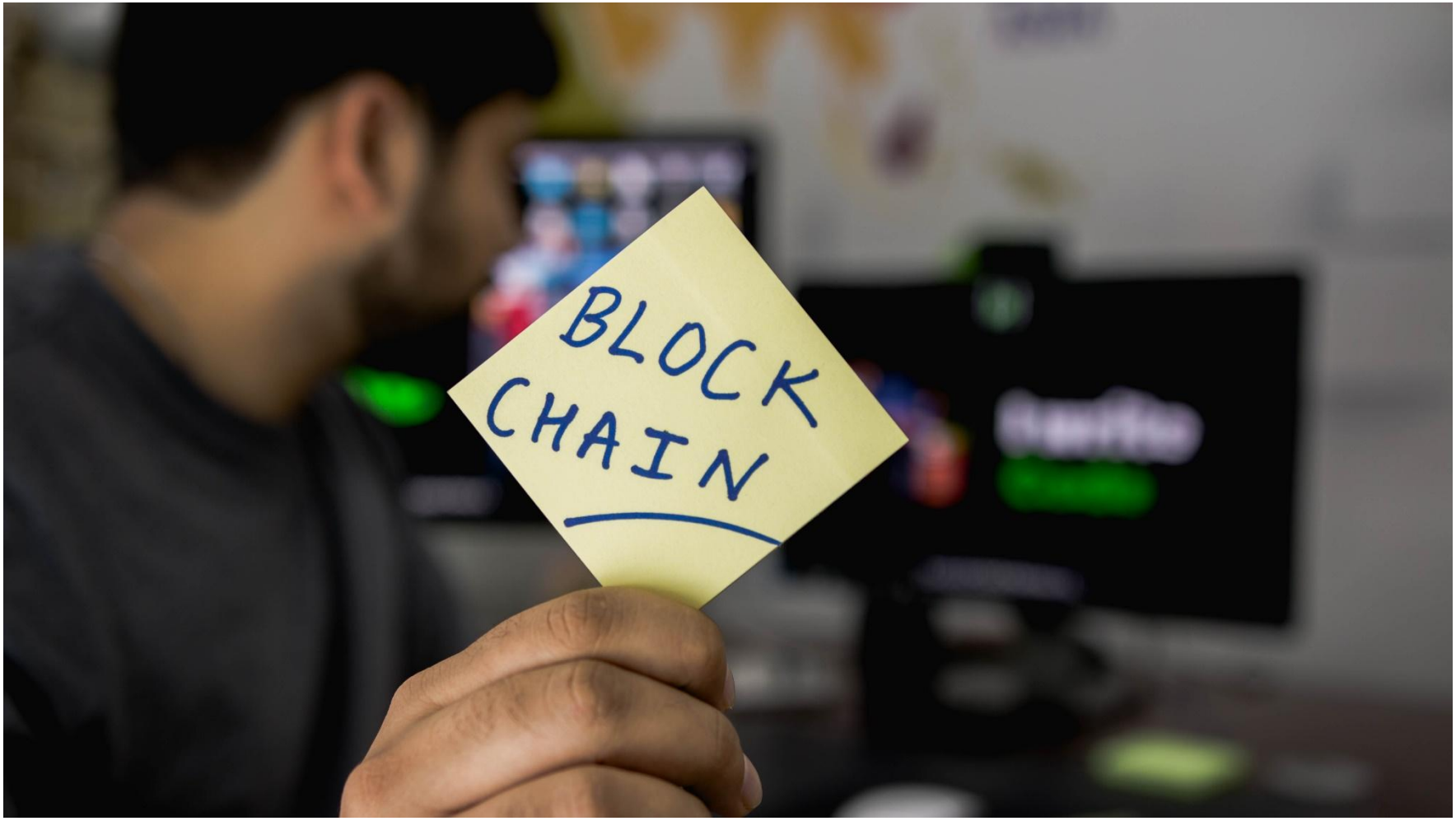
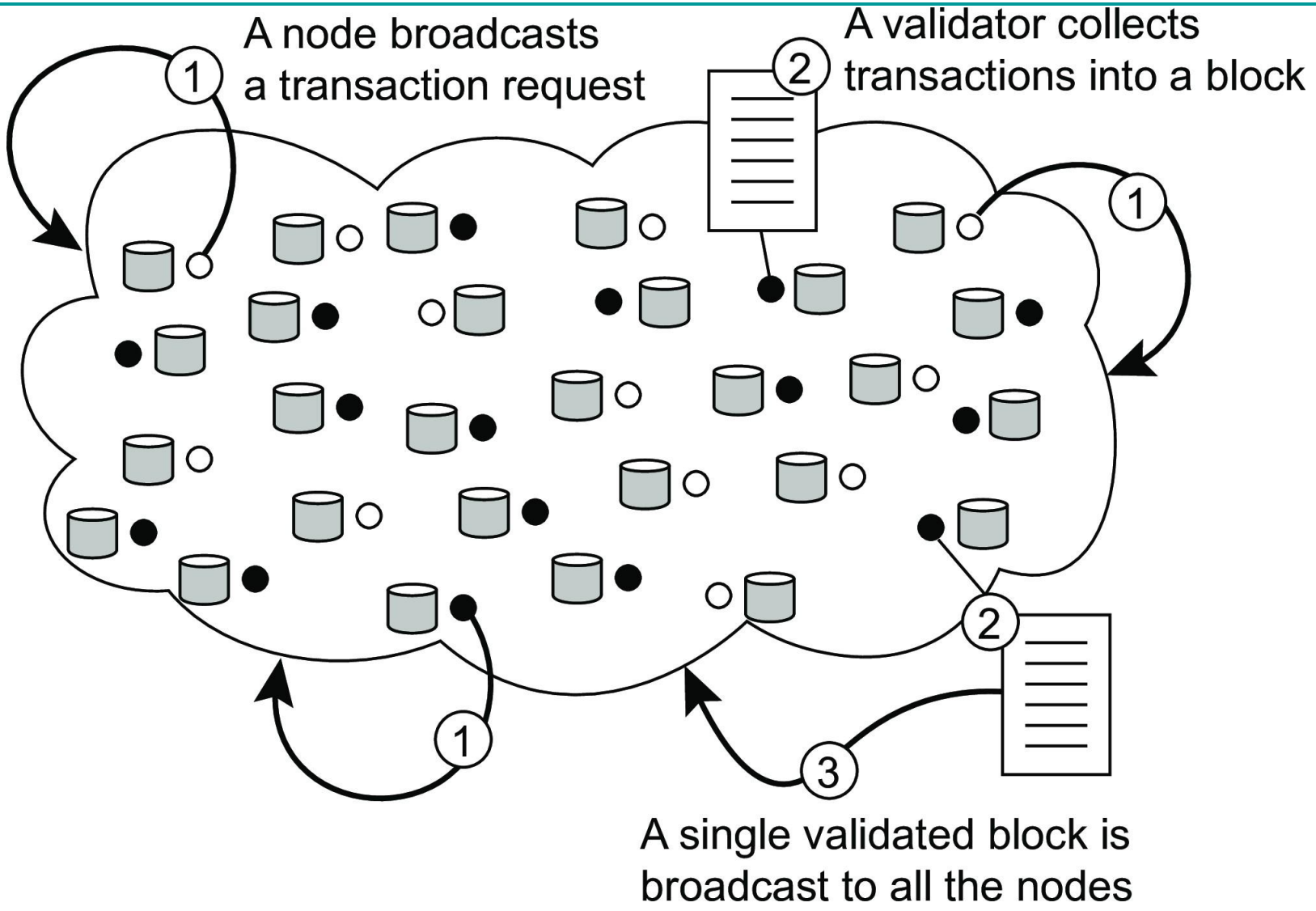
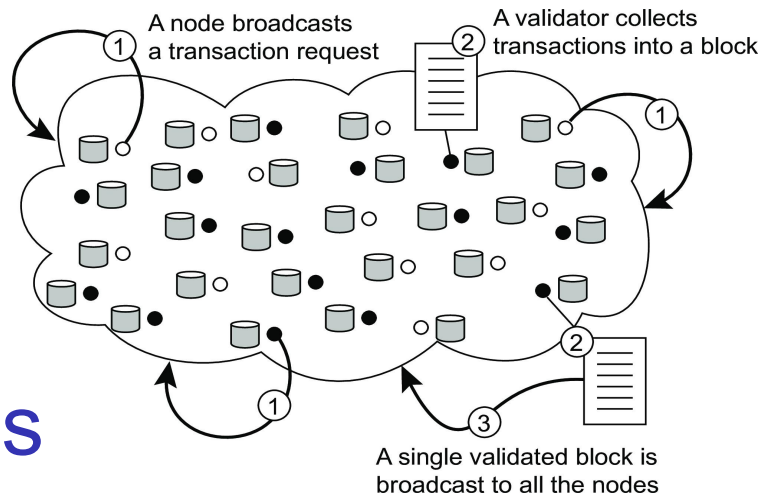


Photo by Hitesh Choudhary on Unsplash

Blockchain Basics

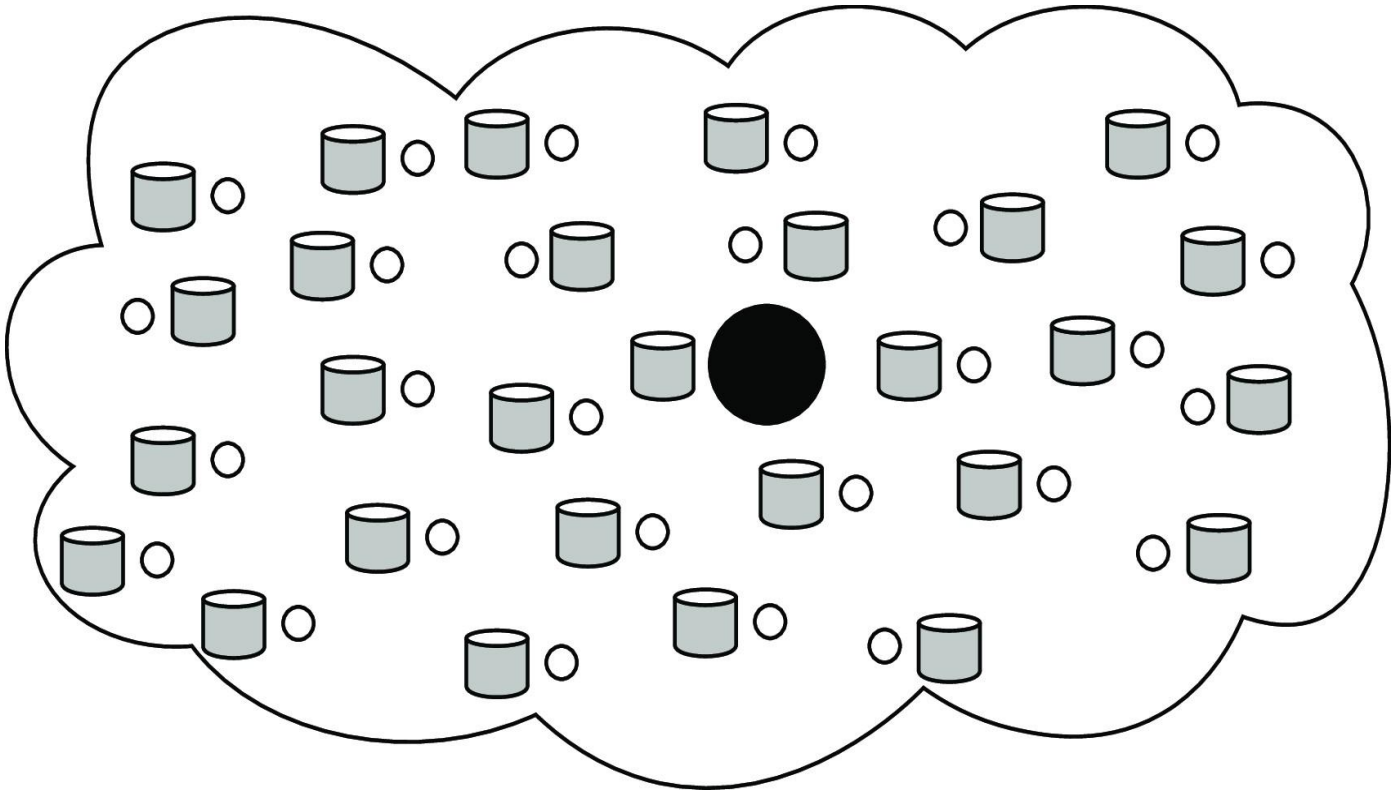


Blockchain Basics



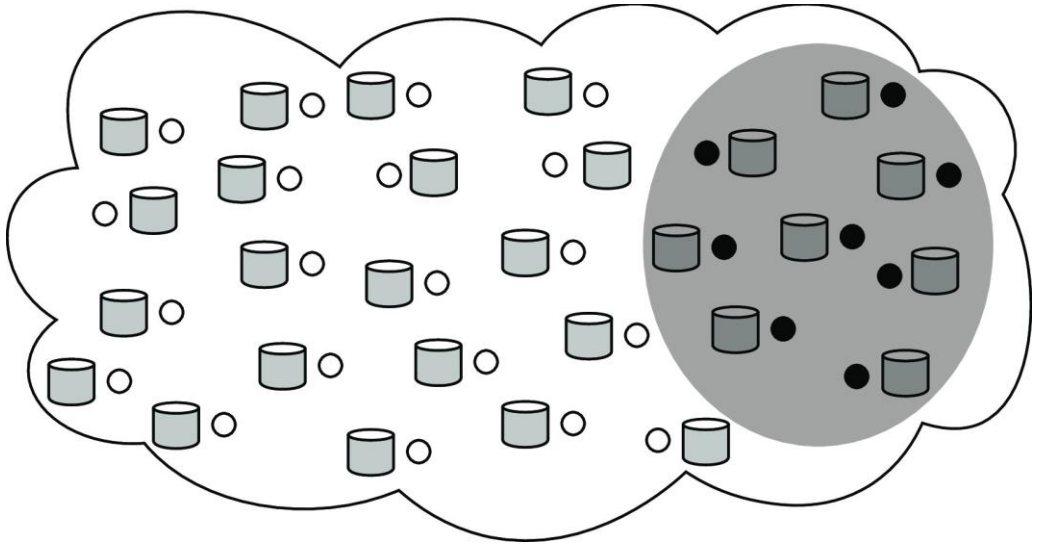
- Observations
  - Blocks are organized into an unforgeable append-only chain
  - Each block in the blockchain is immutable ⇒ massive replication
  - The real snag lies in who is allowed to append a block to a chain

Appending a block: Centralized



- A single entity decides on which validator can go ahead and append a block.
- Does not fit the design goals of blockchains.

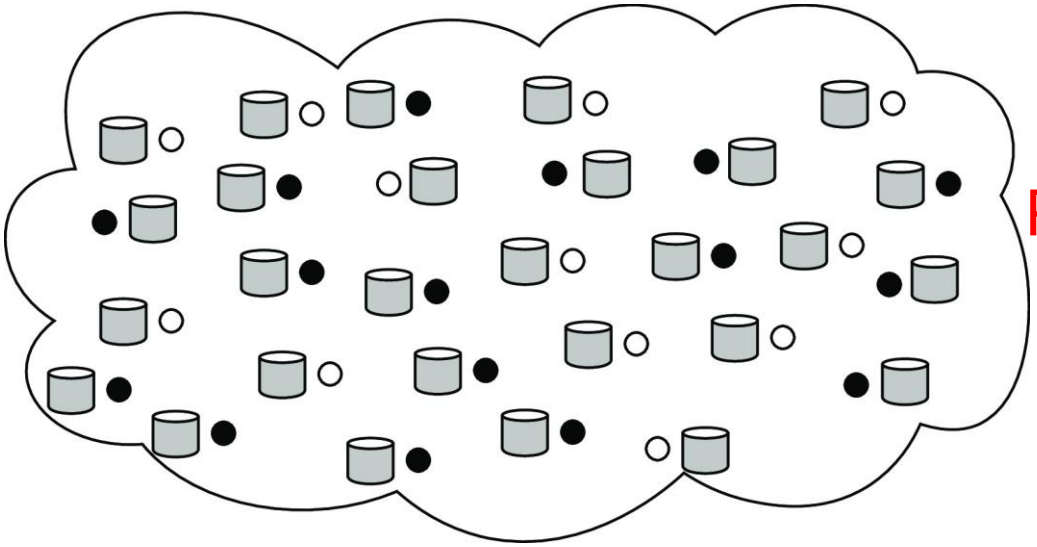
Appending a block: Distributed



Permissioned

- A selected, relatively small group of servers jointly reach consensus on which validator can go ahead.
- None of the servers needs to be trusted, if $\approx \frac{2}{3}$ behave according to specification.
- In practice, only 10s of servers can be accommodated.

Appending a block: Distributed



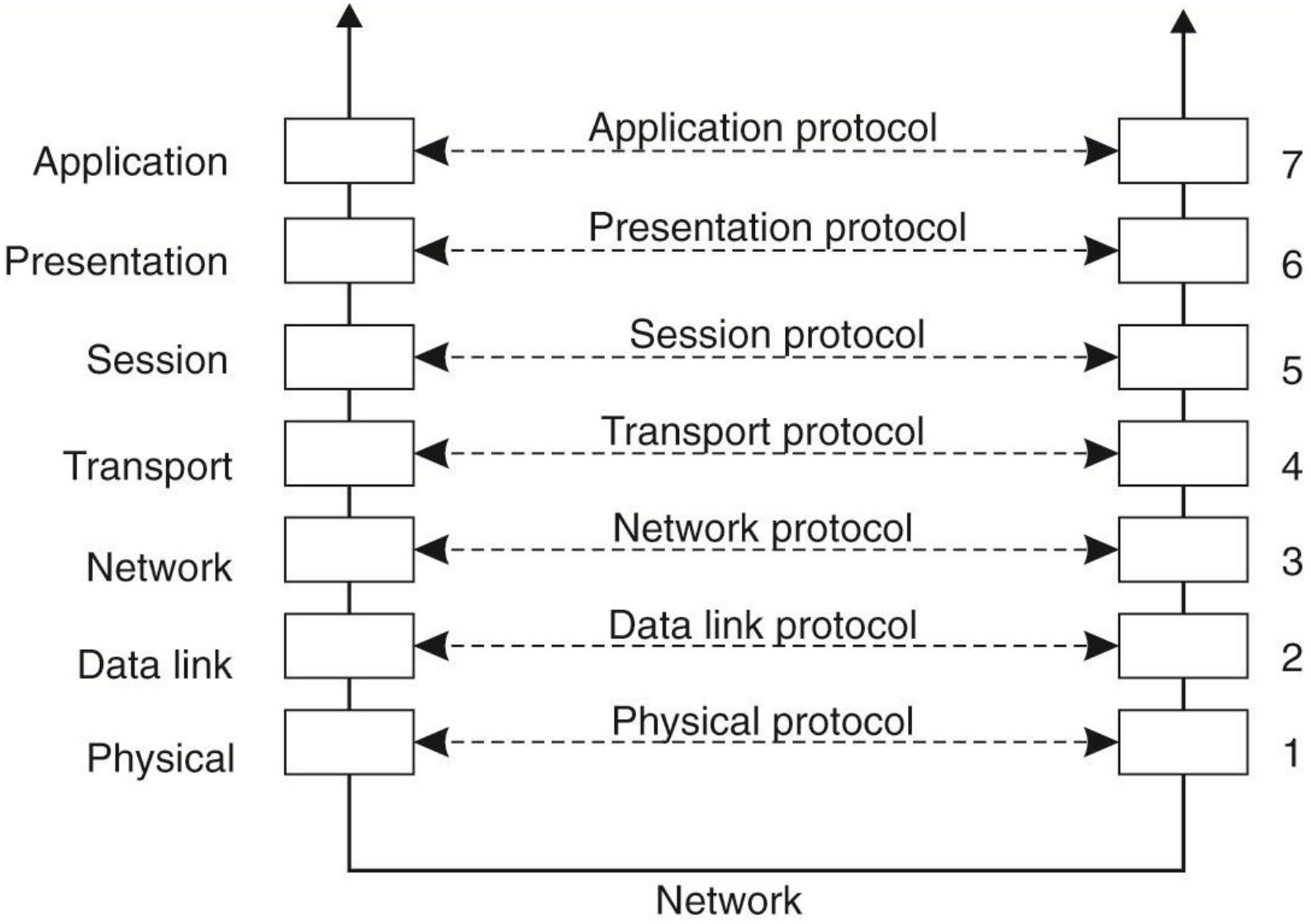
Permissionless

- Participants collectively engage in a leader election. Only the elected leader is allowed to append a block of validated transactions.
- Large-scale, decentralized leader election that is fair, robust, secure, etc. is not trivial.

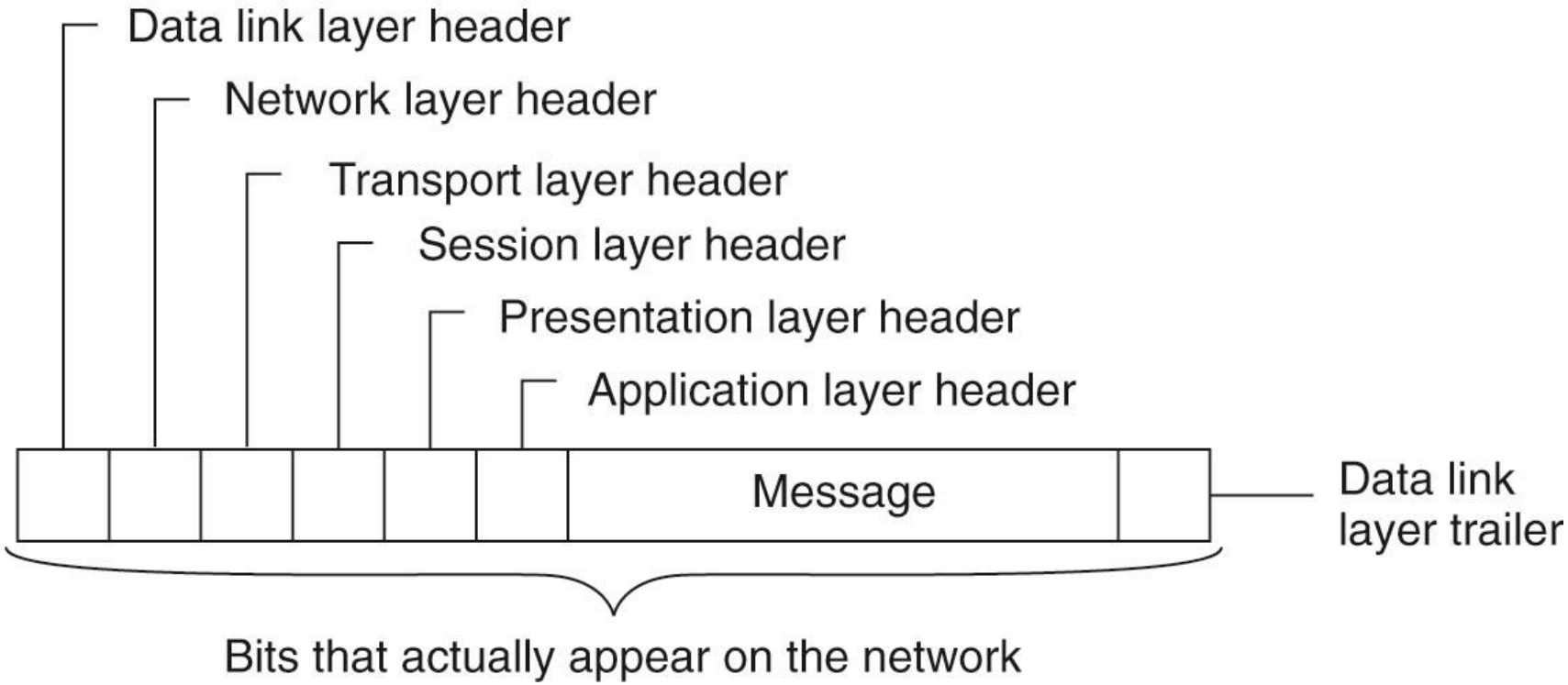
So Far

- Blockchain basics
- Communication
  - Layered Communication
  - Types of Communication
- RPC and RMI
- Sockets

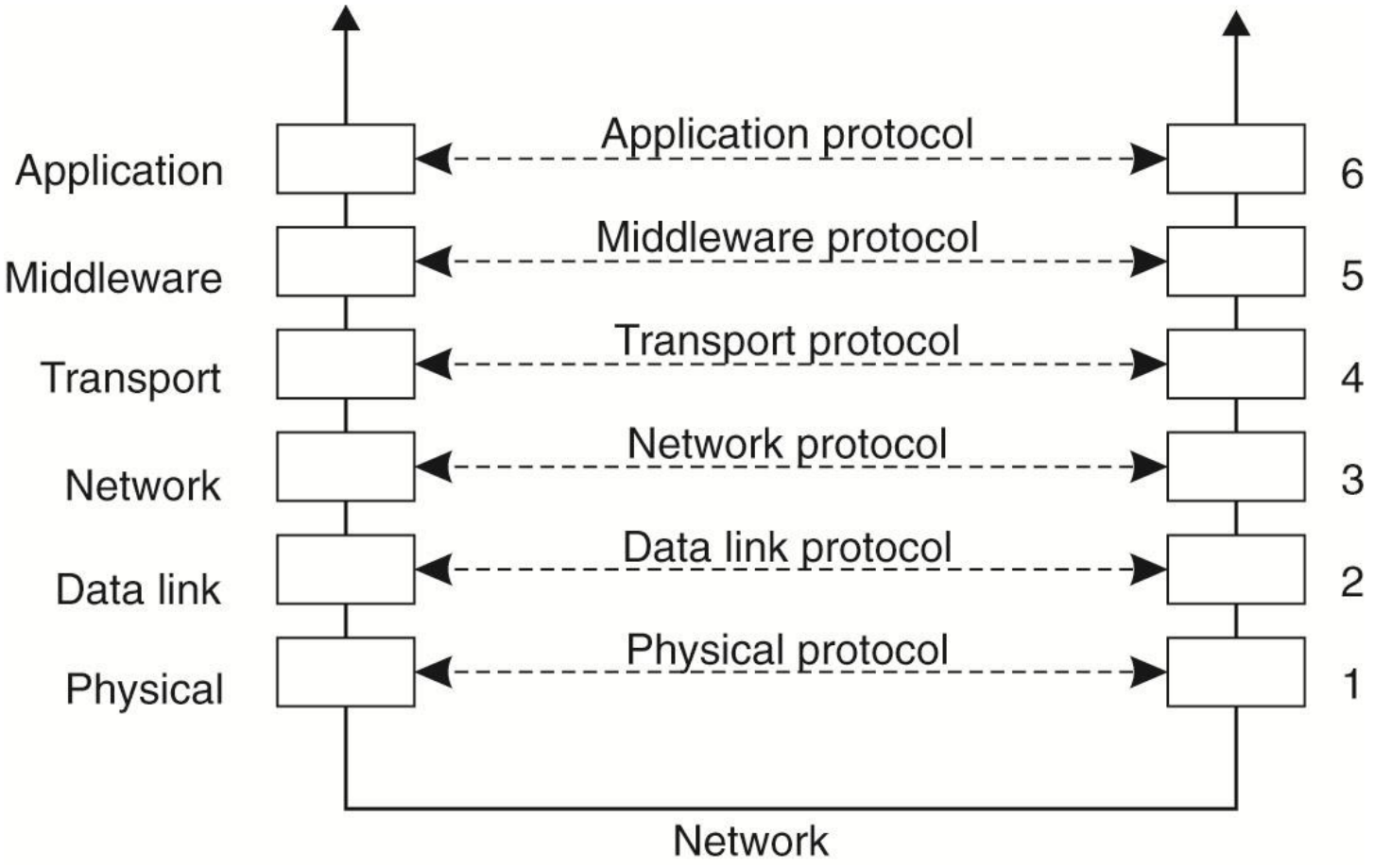
Basic Networking Model



Layered Protocols



Adding Middleware



Middleware Layer

- **Observation**: Middleware is invented to provide **common** services and protocols that can be used by many different applications:
  - A rich set of **communication protocols**, but which allow different applications to communicate
  - **(Un)marshaling** of data, necessary for integrated systems
  - **Naming protocols**, so that different applications can easily share resources
  - **Security protocols**, to allow different applications to communicate in a secure way
  - **Scaling mechanisms**, such as support for replication and caching

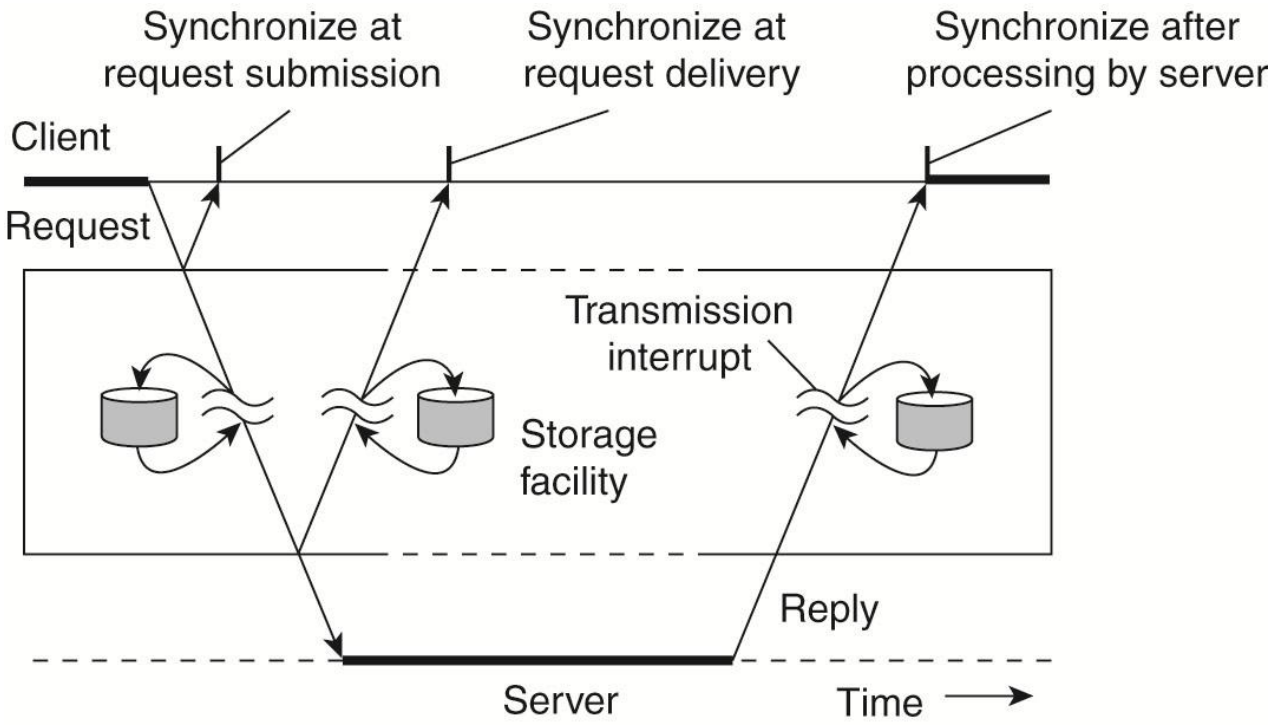
- **Note**: what remains are truly **application-specific** protocols

- **Question**: Such as...?

So Far

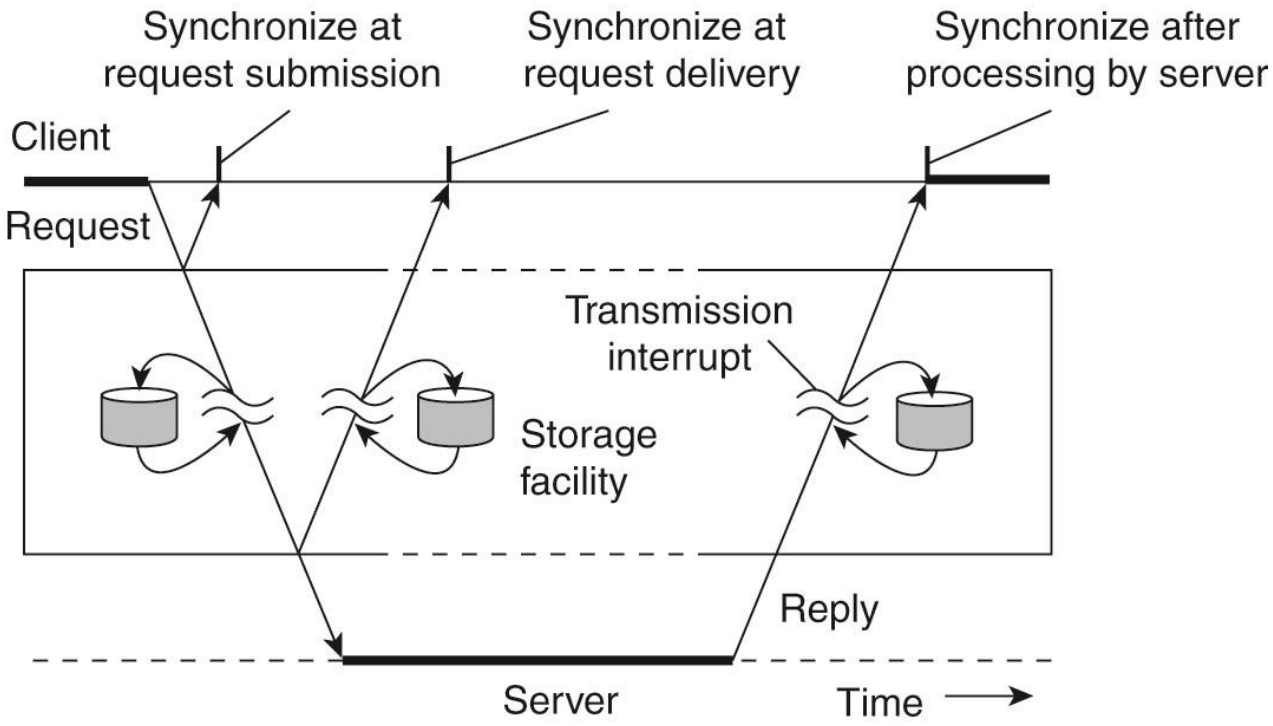
- Blockchain basics
- Communication
  - Layered Communication
  - Types of Communication
- RPC and RMI
- Sockets

Types of Communication (1/3)



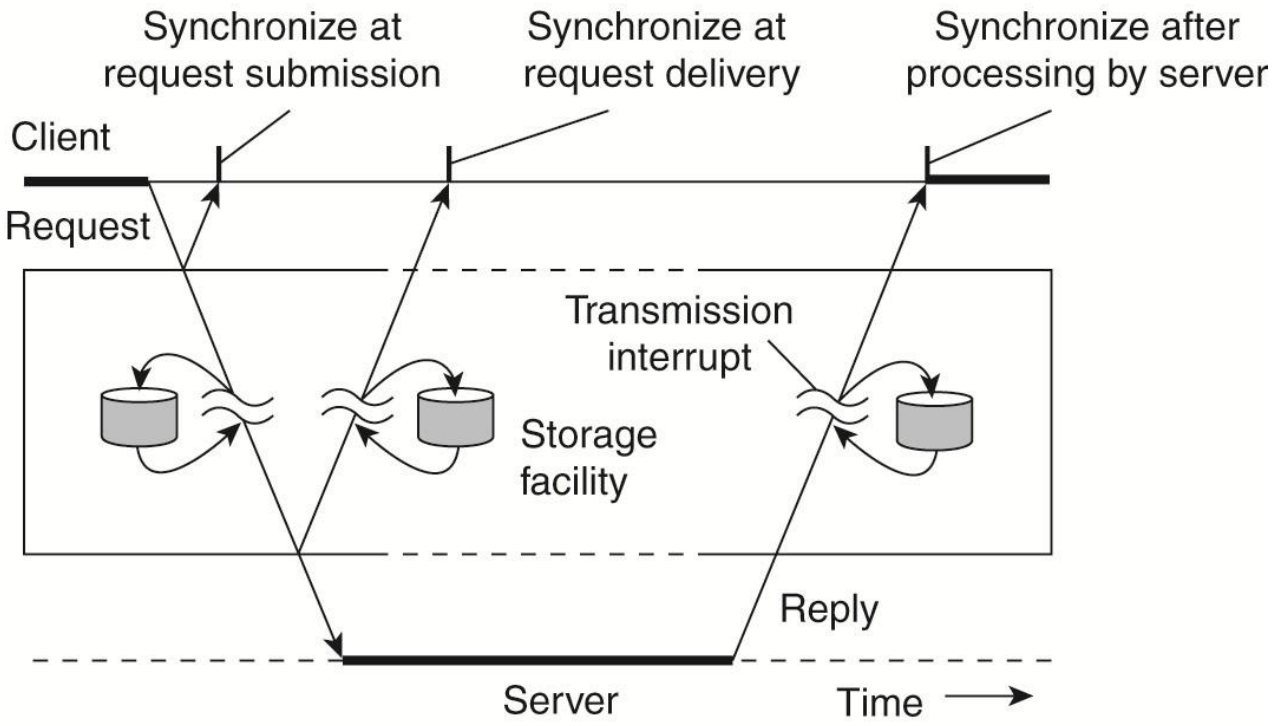
- **Distinguish**:
  - **Transient** versus **persistent** communication
  - **Asynchronous** versus **synchronous** communication

Types of Communication (2/3)



- **Transient communication:** A message is discarded by a communication server as soon as it cannot be delivered at the next server, or at the receiver.
- **Persistent communication**: A message is stored at a communication server as long as it takes to deliver it at the receiver.

Types of Communication (3/3)



- **Places for synchronization:**
  - At request submission
  - At request delivery
  - After request processing

Client/Server



- **Some observations**: Client/Server computing is generally based on a model of **transient synchronous communication**:

Client and server must be active at the time of communication	Client issues request and blocks until it receives reply	Server essentially waits only for incoming requests, and subsequently processes them

- **Drawbacks of synchronous communication:**

Client cannot do any other work while waiting for reply	Failures must be dealt with immediately (the client is waiting)	In many cases the model is simply not appropriate (mail, news)

Image source: https://www.zazzle.com/funny+server+accessories

Messaging

- **Message-oriented middleware**: Aims at high-level **persistent asynchronous communication**:

Processes send each other messages, which are queued	Sender need not wait for immediate reply, but can do other things	Middleware often ensures fault tolerance



Image source: http://tattoo-journal.com/40-lovely-mom-tattoo-designs/

So Far

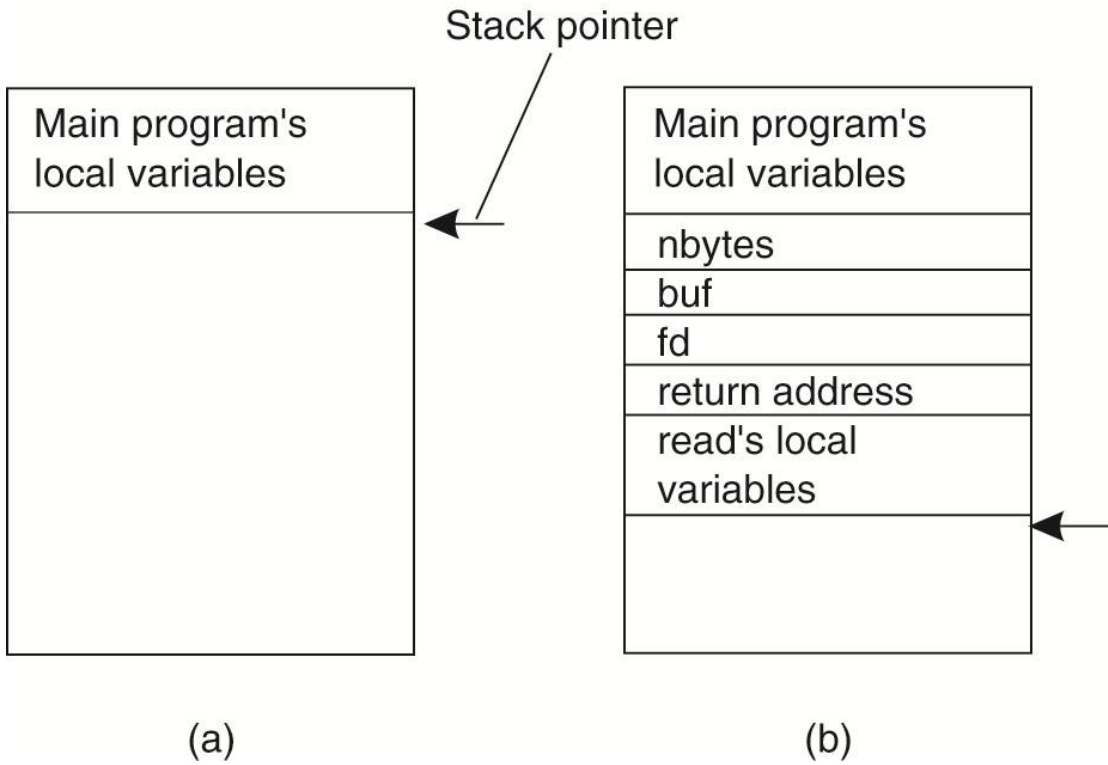
- Blockchain basics
- Communication
  - Layered Communication
  - Types of Communication
- RPC and RMI
- Sockets

Conventional Procedure Call

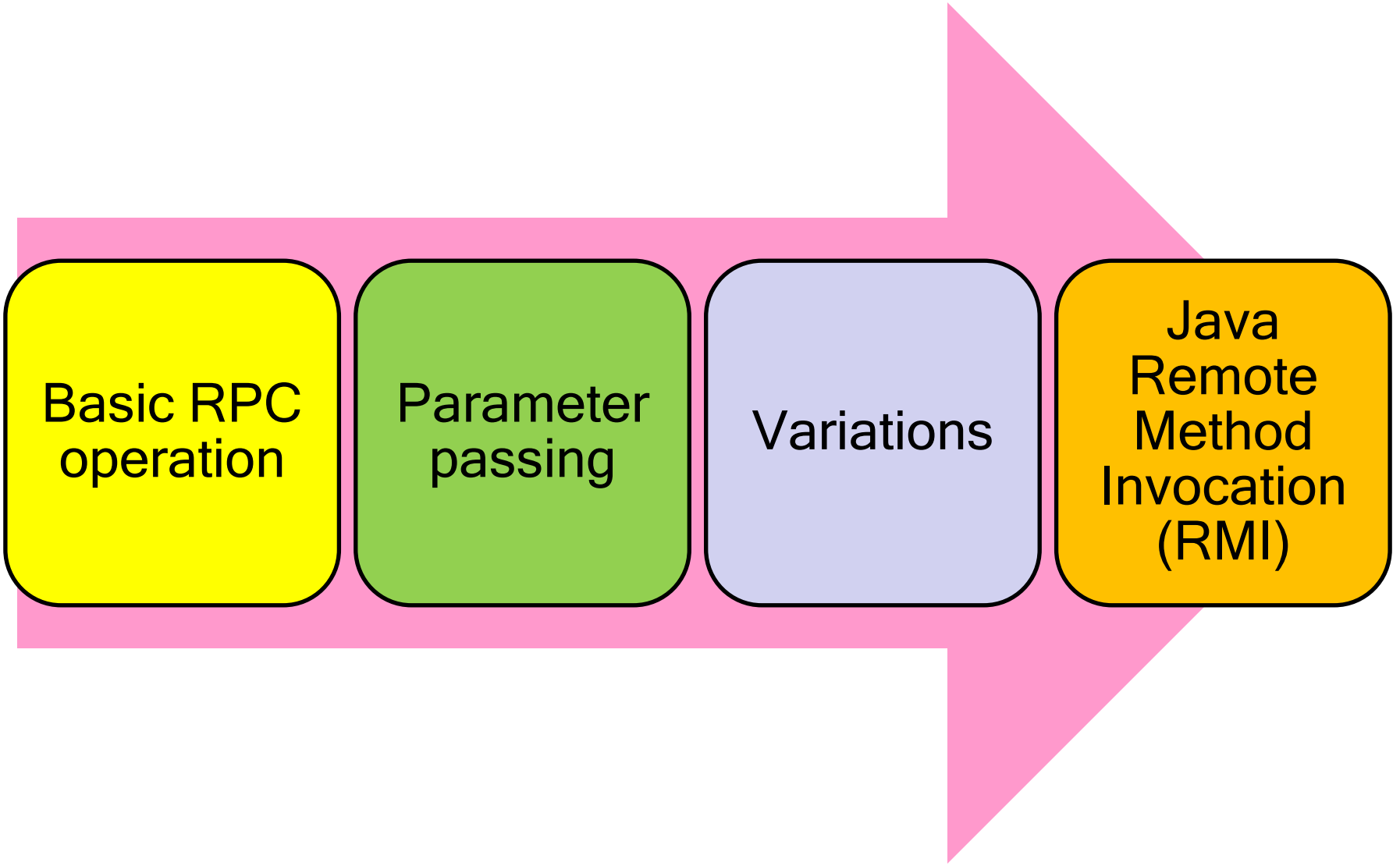
Parameter passing in a local procedure call:

(a) the stack before the call to read.

(b) The stack while the called procedure is active.



Remote Procedure Call (RPC)

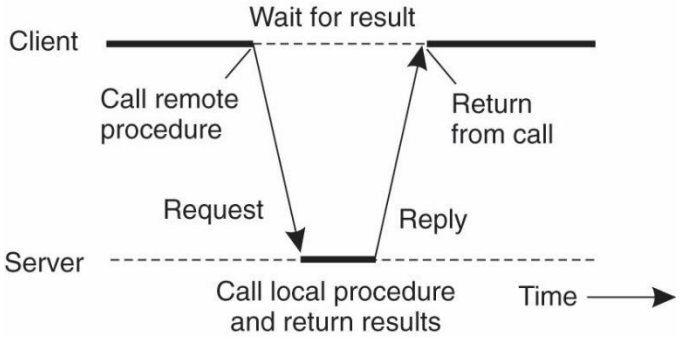


Basic RPC Operation (1/3)

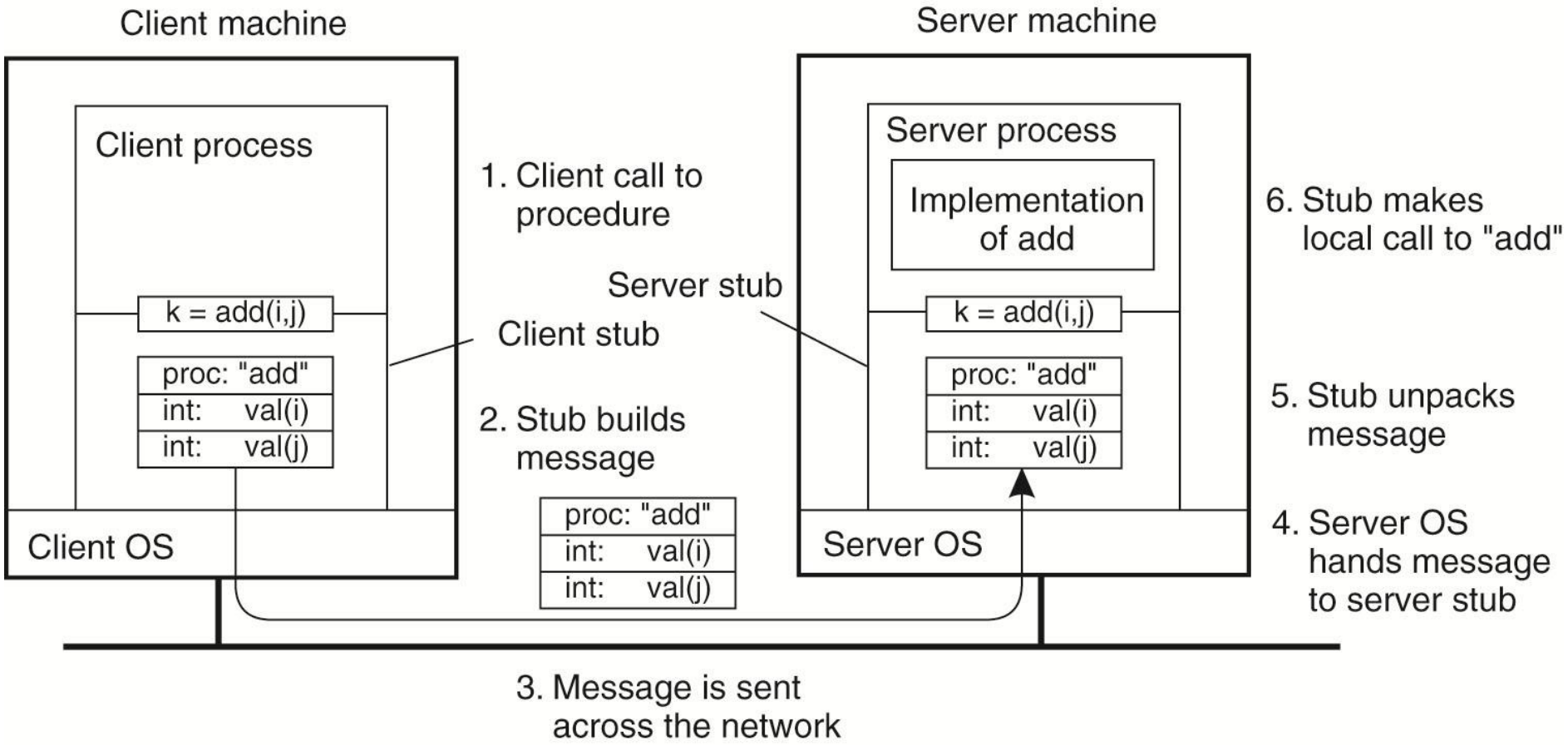
Observations:

- Application developers are familiar with simple procedure model
- Well-engineered procedures operate in isolation (black box)
- There is no fundamental reason not to execute procedures on separate machine

Conclusion: communication between caller & callee can be hidden by using procedure-call mechanism.

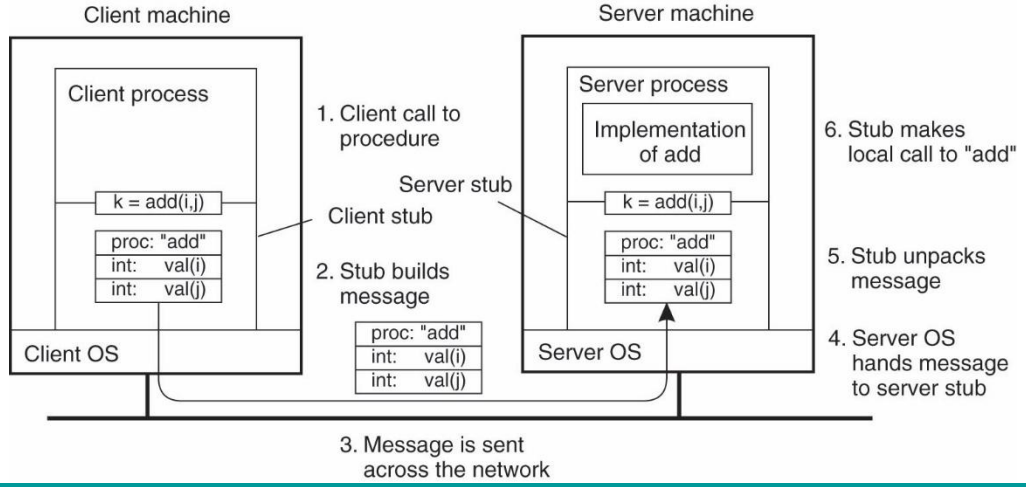


Basic RPC Operation (2/3)



Basic RPC Operation (3/3)

1. Client procedure calls client stub as usual.
2. Client stub builds message and calls local OS.
3. Client's OS sends message to remote OS.
4. Remote OS gives message to server stub.
5. Server stub unpacks parameters and calls server.
6. Server does work and returns result to the stub.
7. Server stub packs it in message and calls OS.
8. Server's OS sends message to client's OS.
9. Client's OS gives message to client stub.
10. Client stub unpacks result and returns to the client.



RPC: Parameter Passing (1/2)

Parameter marshaling: There's more than just wrapping parameters into a message:

Client and server machines may have different data representations (think of byte ordering)

Wrapping a parameter means transforming a value into a sequence of bytes

Client and server must agree on the encoding:

- How are basic data values represented (integers, floats, characters)
- How are complex data values represented (arrays, unions)

Client and server need to properly interpret messages transforming them into machine-dependent representations.

RPC: Parameter Passing (2/2)

RPC parameter passing:

- RPC assumes **copy in/copy out** semantics: while procedure is executed, nothing can be assumed about parameter values (only Ada supports this model).
- RPC assumes all data that is to be operated on is passed by parameters. Excludes passing **references to (global) data**.

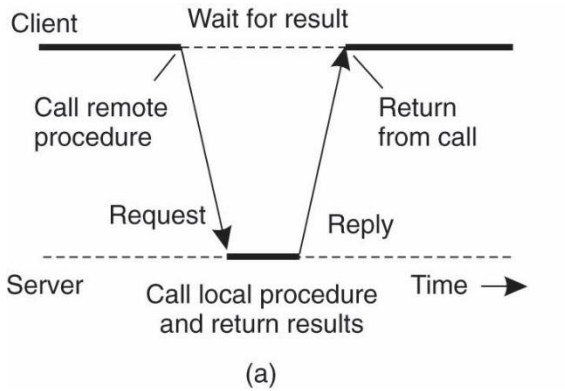
Conclusion: full access transparency cannot be realized.

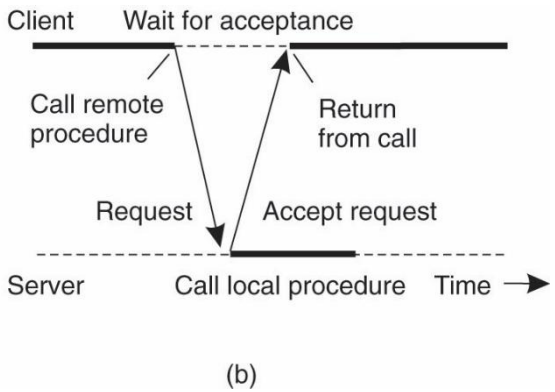
Observation: If we introduce a **remote reference** mechanism, access transparency can be enhanced:

- Remote reference offers unified access to remote data
- Remote references can be passed as parameter in RPCs

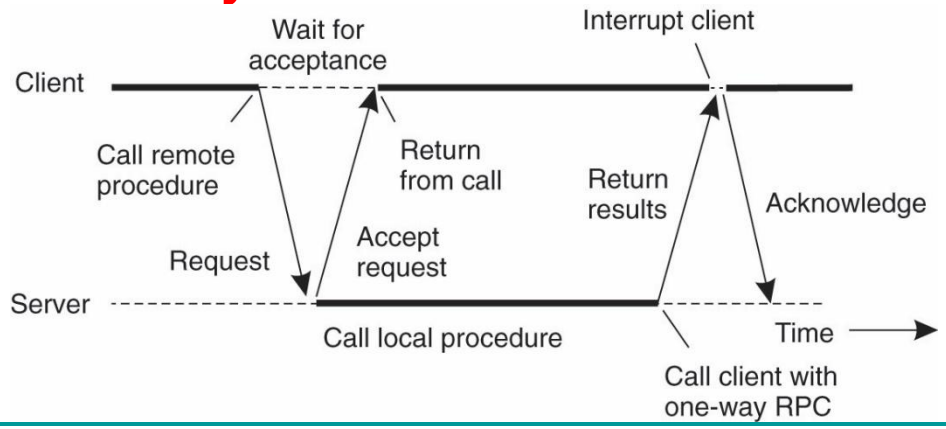
Asynchronous RPCs

Essence: Try to get rid of the strict request-reply behavior, but let the client continue without waiting for an answer from the server.



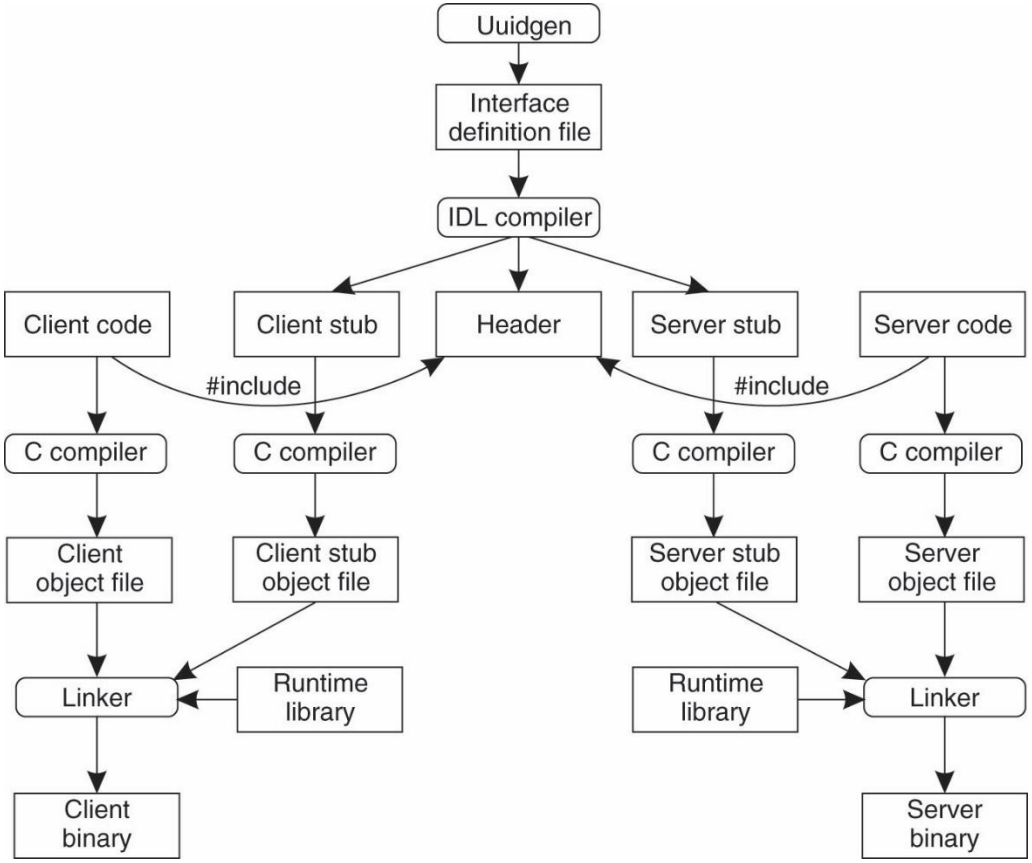


Variation: **deferred synchronous RPC**:



RPC in Practice

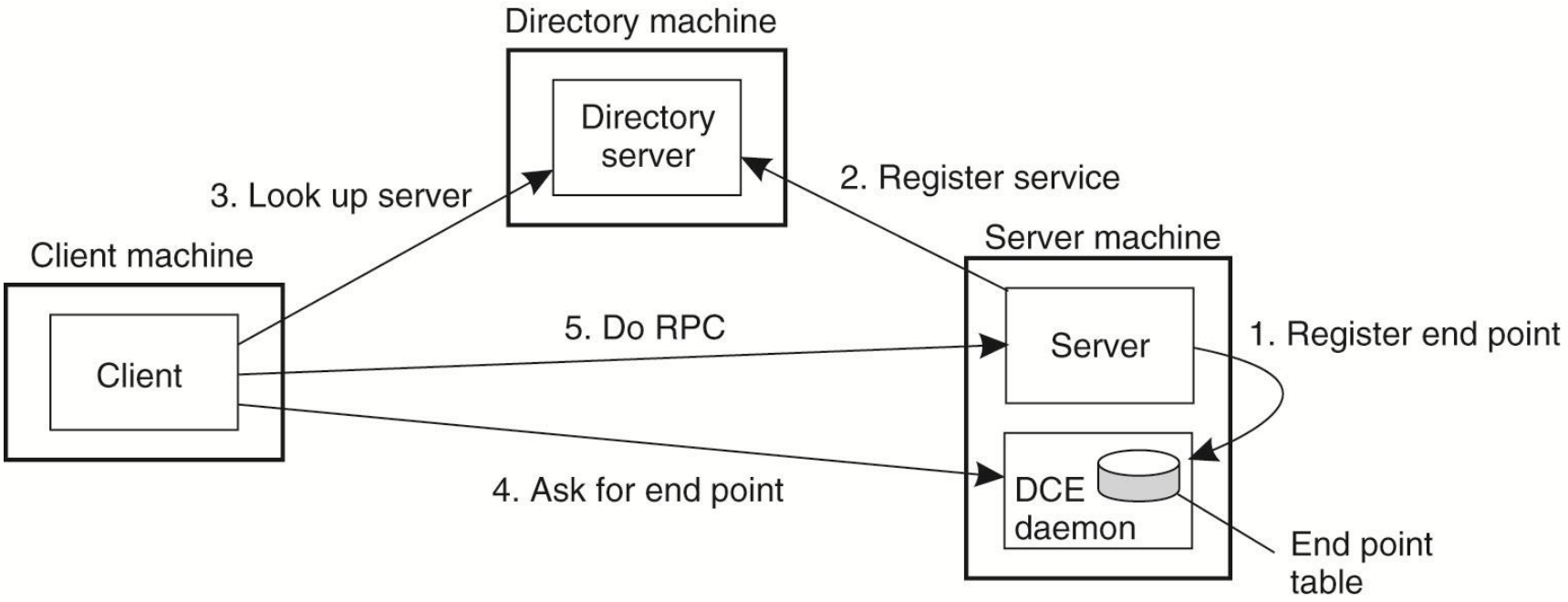
Essence: Let the developer concentrate on only the client- and server-specific code; let the RPC system (generators and libraries) do the rest.



Client-to-Server Binding (DCE)

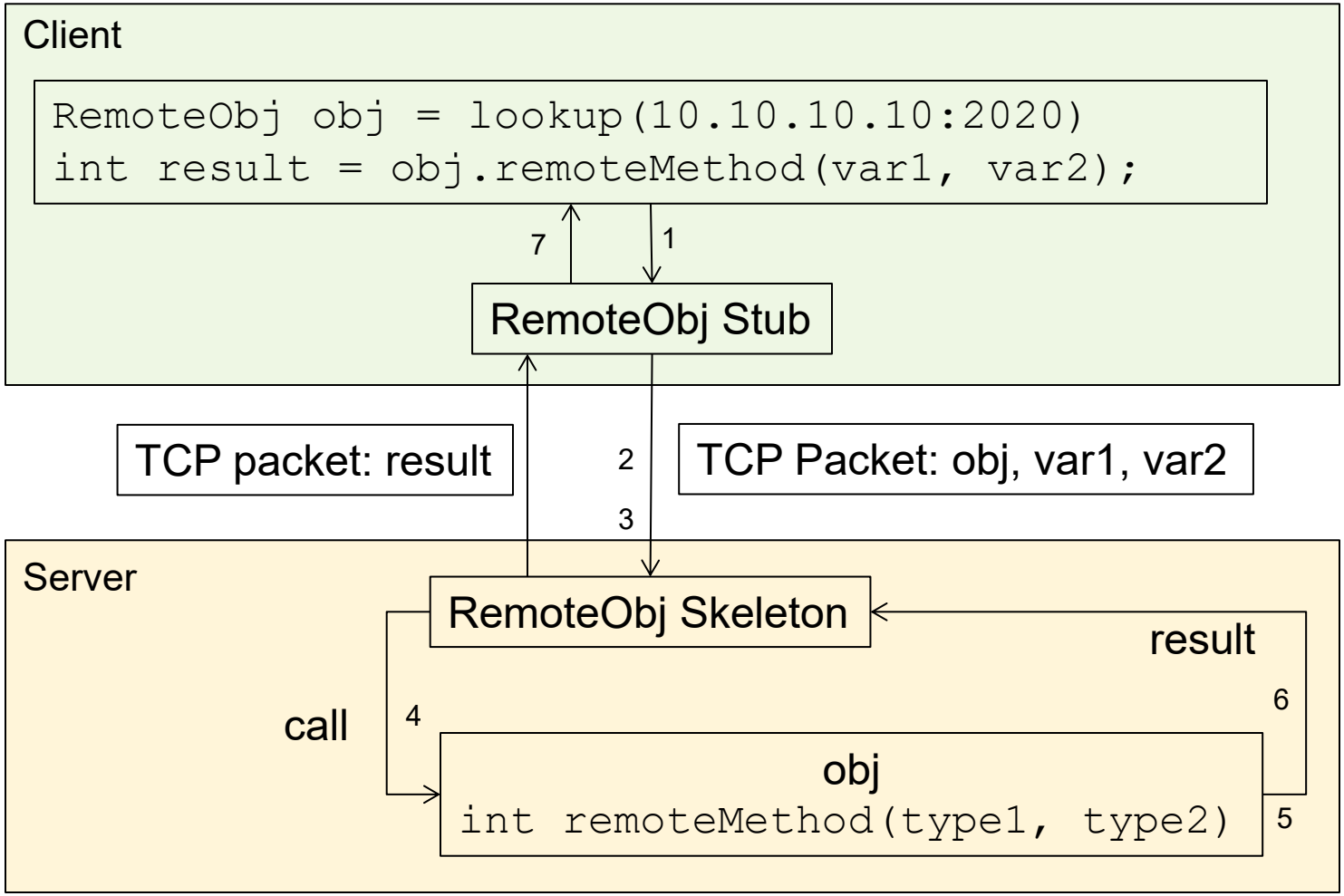
Issues: (1) Client must locate server machine, and (2) locate the server.

Example: DCE uses a separate daemon for each server machine.



Remote Method Invocation Basics

Assume client stub and server skeleton are in place



Remote Method Invocation Basics

1. Client invokes method at stub
2. Stub marshals request and sends it to server
3. Server ensures referenced object is active:
   1. Create separate process to hold object
   2. Load the object into server process
4. Request is unmarshaled by object's skeleton, and referenced method is invoked
5. If request contained an object reference, invocation is applied recursively (i.e., server acts as client)
6. Result is marshaled and passed back to client
7. Client stub unmarshals reply and passes result to client application

RMI Parameter Passing

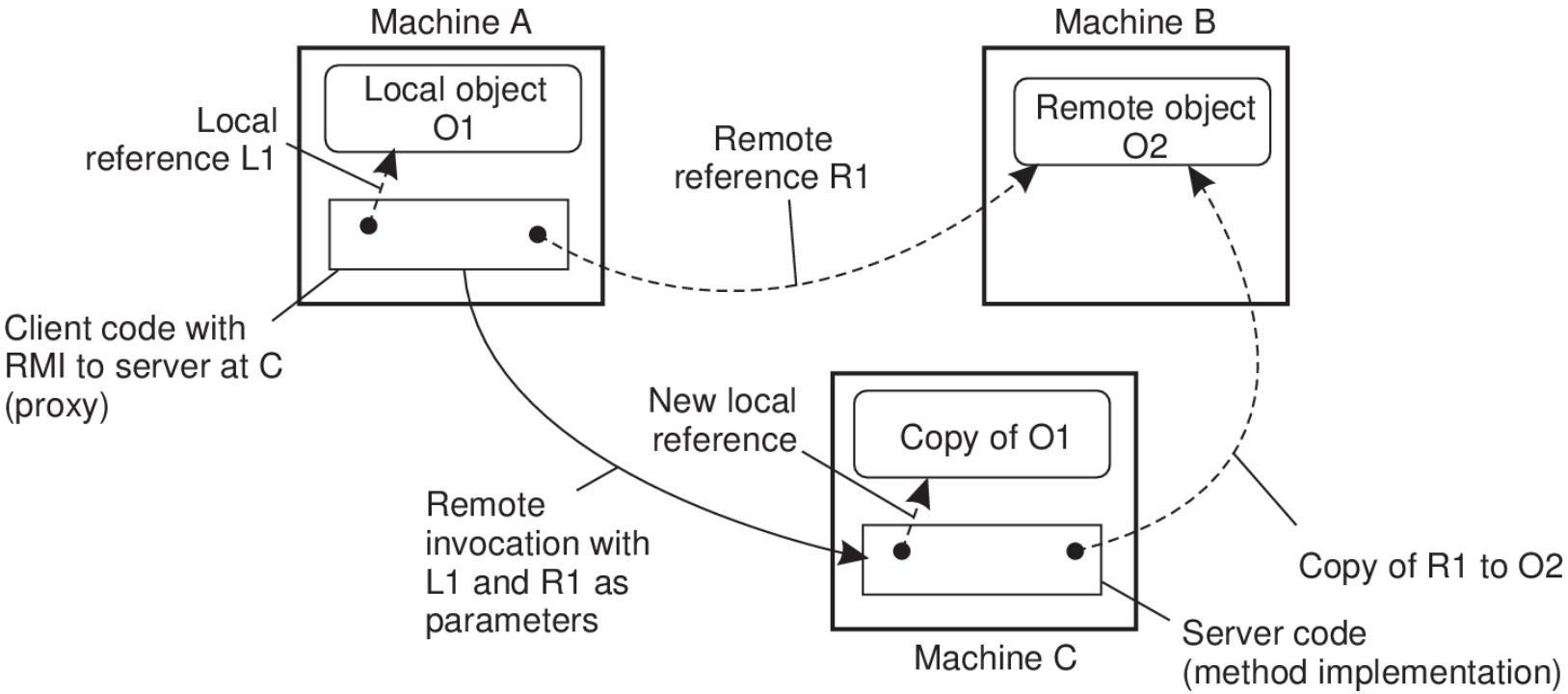
Object-by-value: A client may also pass a complete object as parameter value:

- An object has to be marshaled:
  - Marshall its state
  - Marshall its methods, or give a reference to where an implementation can be found

- Server unmarshals object. Note that we have now created a copy of the original object.

- Object-by-value passing tends to introduce nasty problems

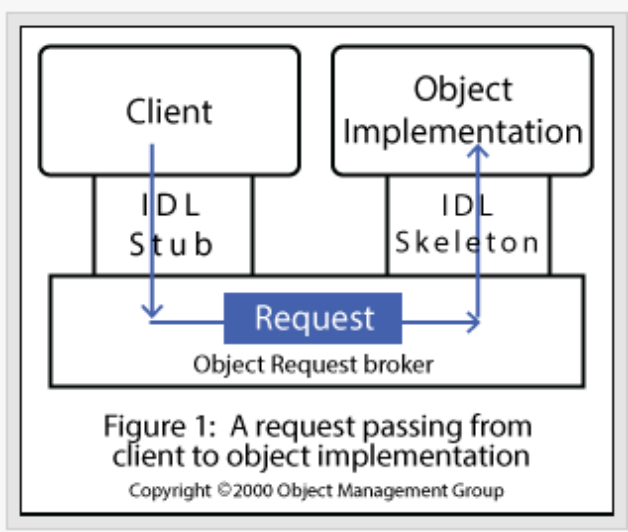
RMI Parameter Passing



Note: System-wide object references generally contain server address and port to which adapters listens, and local object ID.

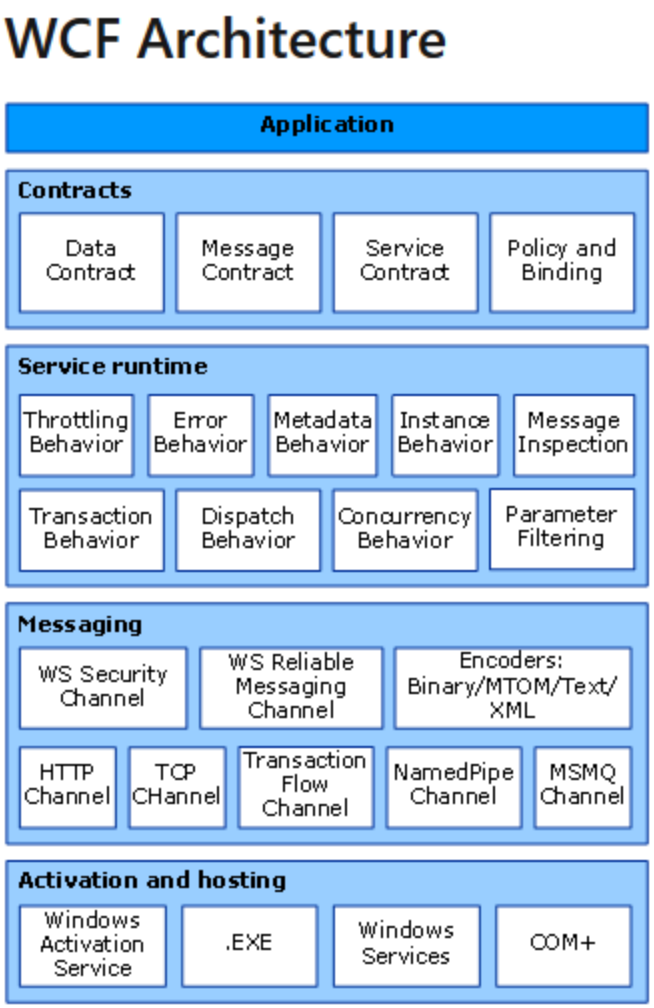
Extra: May also contain information on protocol between client and servers (TCP, UDP, SOAP, JSON, etc.)

Others: CORBA



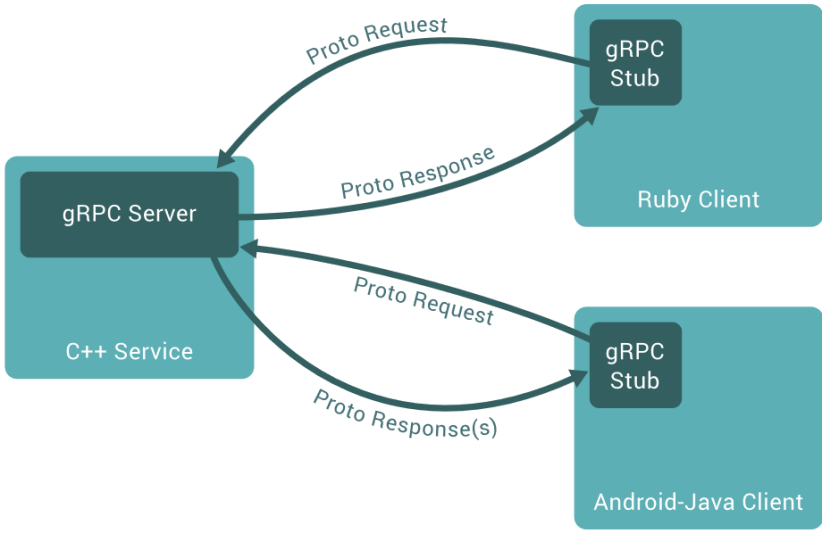
- https://corba.org
- Multi-language support (COBOL, Ada, Lisp, Java, Python, etc.)
- Legacy technology, but still works

Others: Windows Communication Foundation



- https://docs.microsoft.com/en-us/dotnet/framework/wcf
- Microsoft based technology
- Web Services based
- Support for .NET languages and Windows, should work on Linux

Others: gRPC



- https://grpc.io/
- Language independent (multi-platform)
- Uses binary underlying protocol (ProtoBuf from Google)

So Far

- Blockchain basics
- Communication
  - Layered Communication
  - Types of Communication
- RPC and RMI
- Sockets

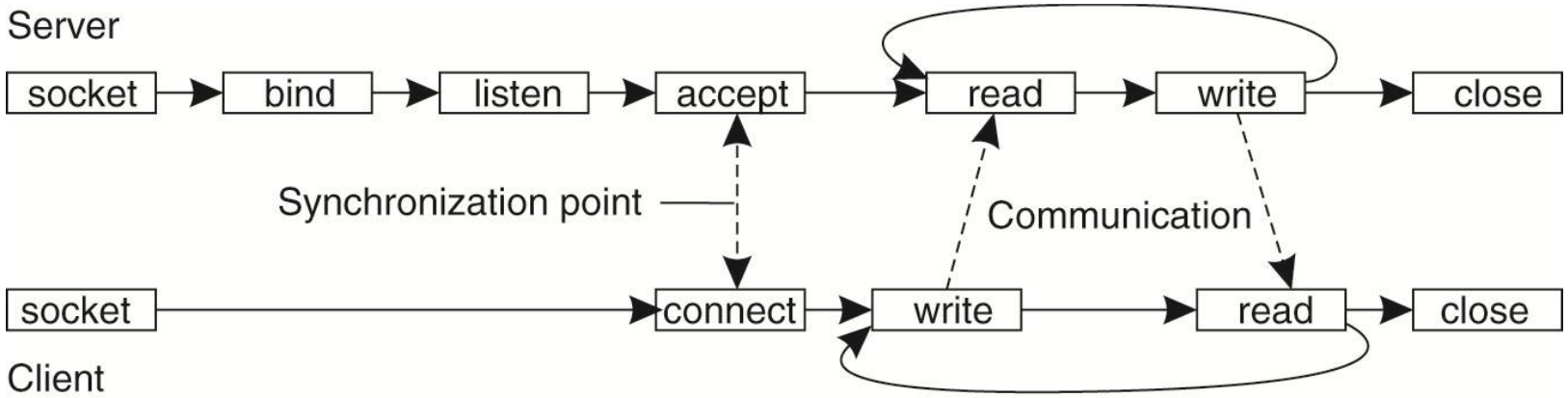
Message-Oriented Communication

Transient Messaging	Message-Queuing System	Examples
• Sockets	• Message Brokers	• IBM MQ (formerly WebSphere MQ) • Others

Transient Messaging: Sockets

Example: Consider the Berkeley **socket interface**, which has been adopted by all Posix systems, as well as Windows 95/NT/2000/XP/Vista:

Primitive	Meaning
Socket	Create a new communication end point
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection



Conclusion

- Blockchain basics
- Communication
  - Layered Communication
  - Types of Communication
- RPC and RMI
- Sockets