Communication models in distributed systems

Prof. Claudio Bettini

EveryWare Lab, Dipartimento di Informatica
Università degli Studi di Milano

Copyright

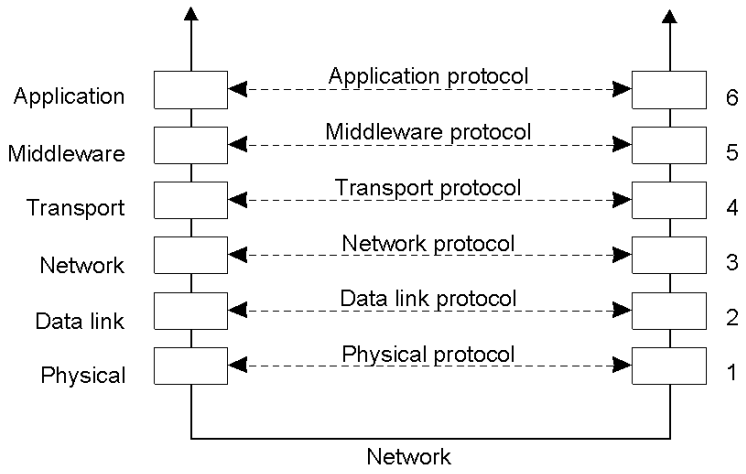
Outline

- Types of communication
  - persistent/transient, synchronous/asynchronous
- Transient message oriented communication: Sockets
- Persistent asynchronous message oriented communication: queuing systems and message brokers
- Remote procedure call

Every Ware Lab

Middleware Protocols

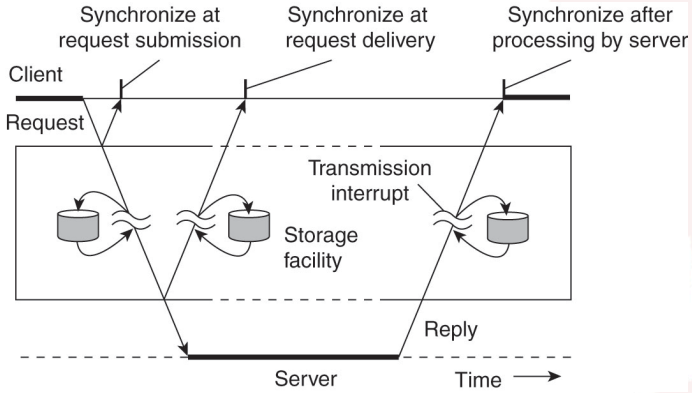
An adapted reference model for networked communication.



Every Ware Lab

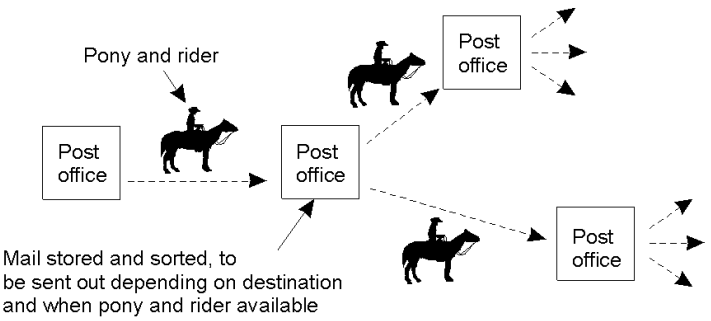
Types of Communication

- Viewing middleware as an intermediate (distributed) service in application-level communication.

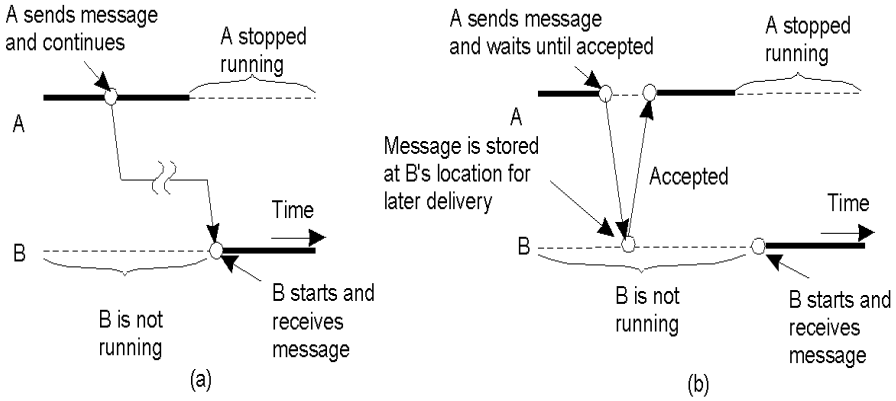


Persistence and Synchronicity in Communication

- Persistent communication of letters back in the days of the Pony Express.



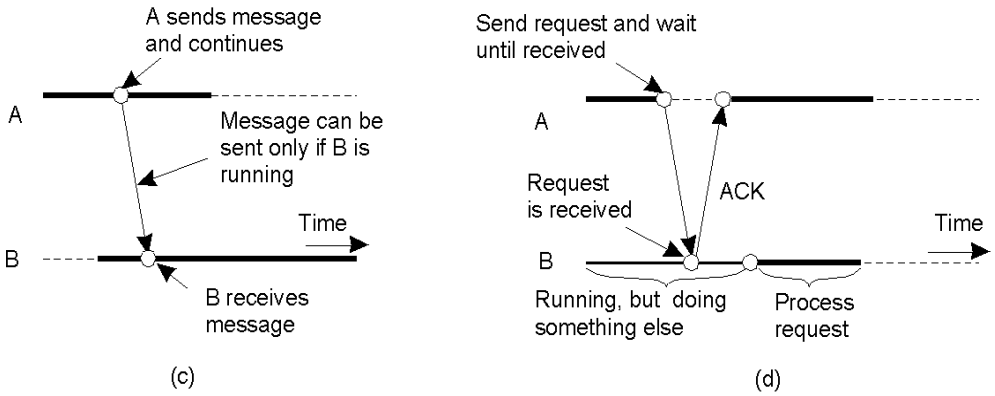
Persistence and Synchronicity in Communication



a) Persistent asynchronous communication

b) Persistent synchronous communication

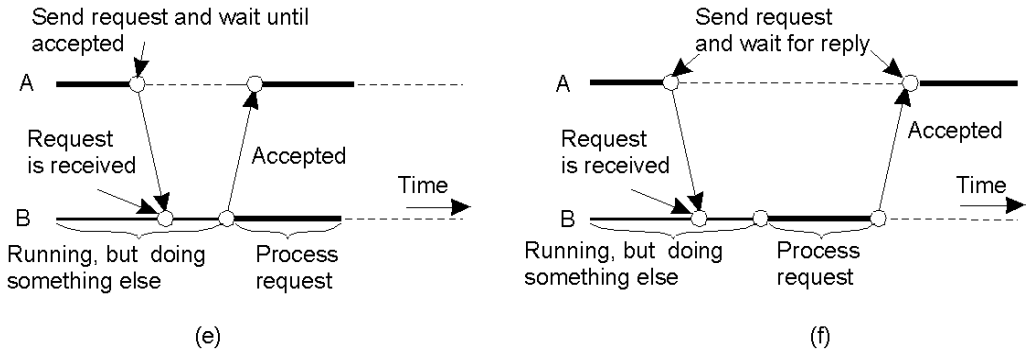
Persistence and Synchronicity in Communication



c) Transient asynchronous communication

d) Receipt-based transient synchronous communication

Persistence and Synchronicity in Communication



e) Delivery-based transient synchronous communication at message delivery

f) Response-based transient synchronous communication

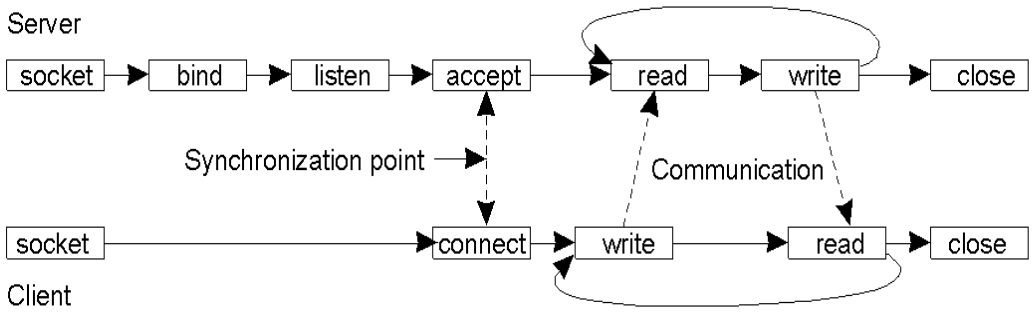
Communication middleware

- Message Oriented Communication
  - synchronous/asynchronous
  - transient/persistent
- Remote Procedure Call and Remote Object Invocation
  - transient synchronous/asynchronous
- Streams: communication of continuous data

Message Oriented Transient Communication: Berkeley Sockets

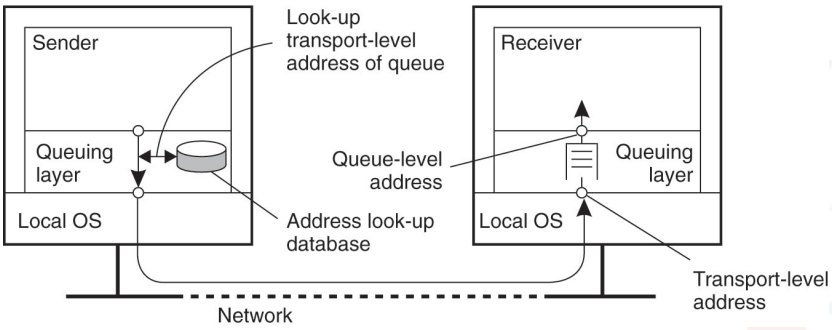
Primitive	Meaning
Socket	Create a new communication endpoint
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive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

Berkeley Sockets



Connection-oriented communication pattern using sockets (socket primitives in C language).

Message Oriented Persistent Communication: Queuing Systems

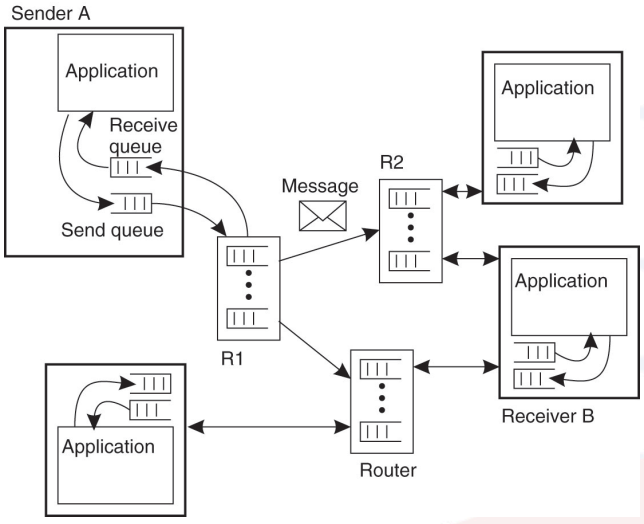


The relationship between queue-level addressing and network-level addressing.

15

Message-queuing system with routers

General organization of a message-queuing system with routers



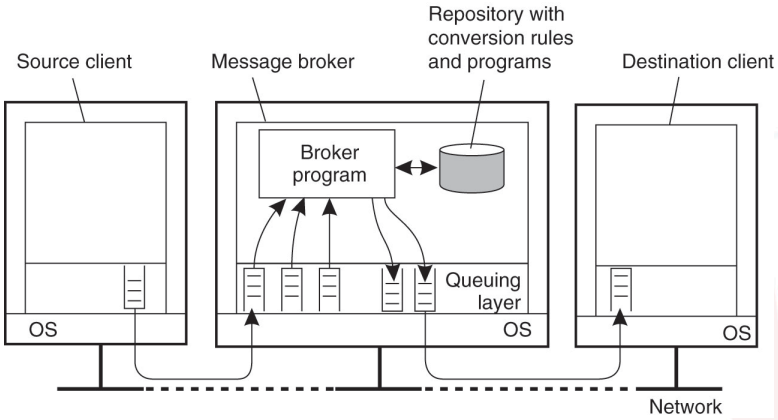
16

Message-Queuing Model

Primitive	Meaning
Put	Append a message to a specified queue
Get	Block until the specified queue is nonempty, and remove the first message
Poll	Check a specified queue for messages, and remove the first. Never block
Notify	Install a handler to be called when a message is put into the specified queue

Basic interface to a queue in a message-queuing system.

Message Brokers



The general organization of a message broker in a message-queuing system.

When considering message queuing systems

- When asynchronous communication is preferable
- When scalability can be increased by admitting delays in requests and responses
- When the producer is faster than the consumer
- When implementing a publish-subscribe communication pattern

Messaging protocols

- XMPP (Extensible Messaging and Presence Protocol) https://xmpp.org
  - Free and open, used for instant messaging (used by Google Talk e FB till 2014 and many others)
- MQTT (Message Queue Telemetry Transport) https://mqtt.org/
  - publish/subscribe model
  - a lightweight messaging protocol designed for constrained devices
  - OASIS standard, candidate for M2M and IOT; also used by FB Messenger
- AMQP (Advanced Message Queuing Protocol) https://www.amqp.org/
  - ISO standard, general purpose
  - suitable for a broad range of messaging-middleware infrastructures, and also for peer-to-peer data transfer.

Popular message-broker software

- Eclipse Mosquitto
  - opensource, lightweight, supports MQTT
- RabbitMQ
  - most popular, opensource, supports AMQP and many other protocols
- Apache ActiveMQ and Kafka
- Google Cloud Pub/Sub
- Amazon MQ

RabbitMQ

- A multi-language open-source implementation of AMQP (and other protocols)
- It consists of
  - A Broker (server written in Erlang) usually hosted in cloud
  - Clients (available in many languages)
  - Clients communicate with the broker through AMQP

RabbitMQ Message flow

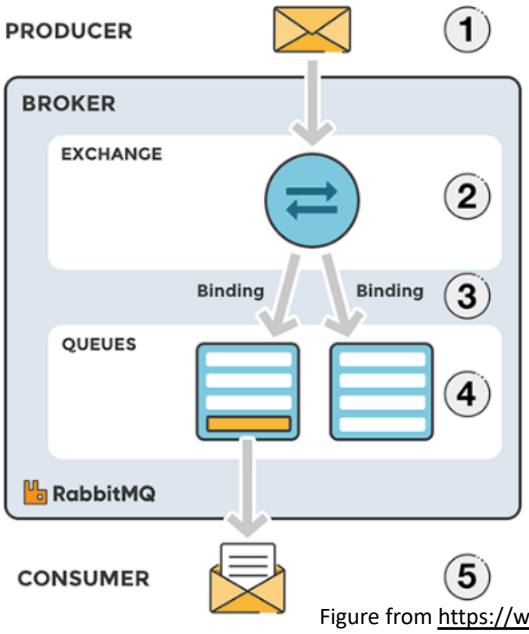


Figure from https://www.cloudamqp.com/

Every Ware Lab

Every Ware Lab

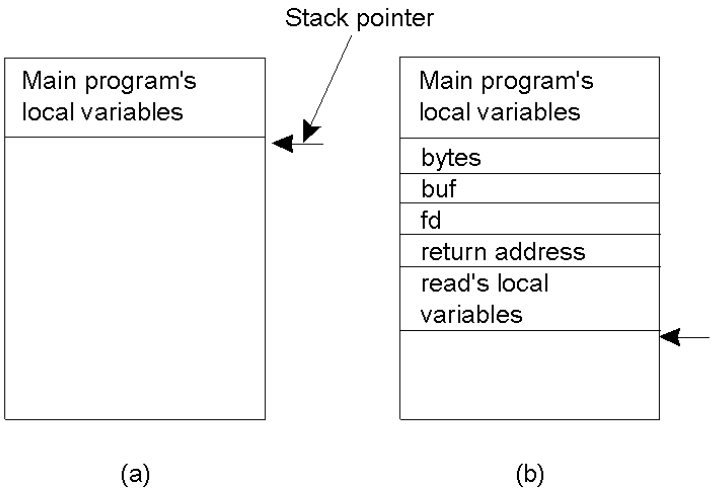
UNIVERSITÀ DEGLI STUDI DI MILANO

Remote Procedure Call

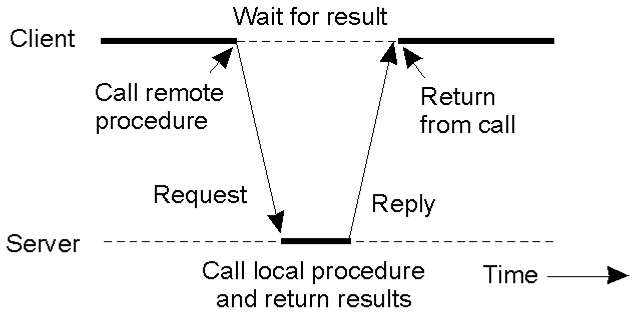
Conventional Procedure Call

a) Parameter passing in a local procedure call: the stack before the call to `read(fd,buf,nbytes)`

b) The stack while the called procedure is active



Client and Server Stubs

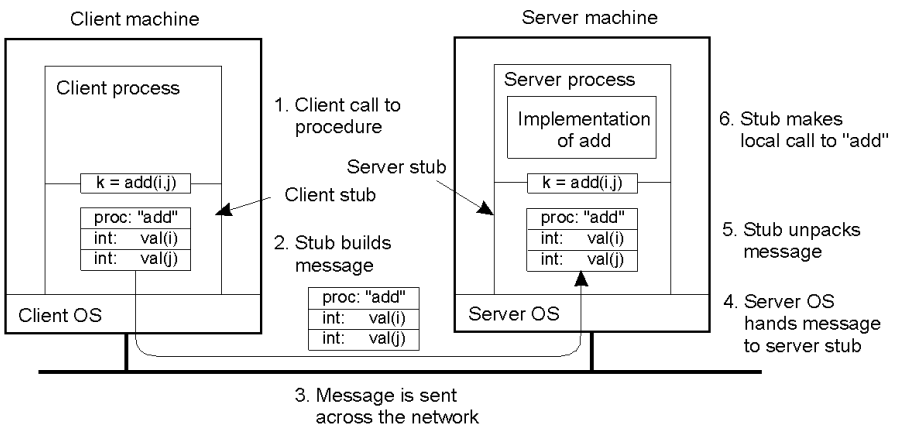


Principle of RPC between a client and server program.

Steps of a Remote Procedure Call

1. Client procedure calls client stub in normal way
2. Client stub builds message, 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, calls server
6. Server does work, returns result to the stub
7. Server stub packs it in message, calls local OS
8. Server's OS sends message to client's OS
9. Client's OS gives message to client stub
10. Stub unpacks result, returns to client

Passing Value Parameters (1)



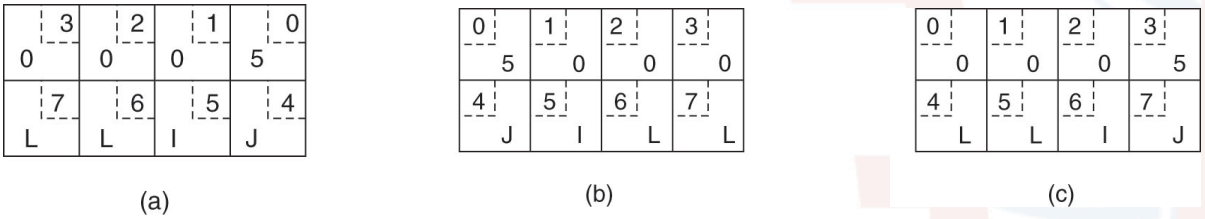
Steps involved in doing remote computation through RPC

Marshalling and Unmarshalling

- *Parameter marshalling*: the procedure to format RPC parameters in a message
- *Parameter unmarshalling*: the inverse procedure of extracting the parameters from the message
- Marshalling requires the serialization of objects, i.e. their coding in a sequence of bytes.
  - Serialization can be in binary (like in gRPC and Java RMI) or in strings (like REST WS in JSON)

Passing Value Parameters (2)

Problem with passing values in RPC:
Source and destination may have different interpretation of data formats

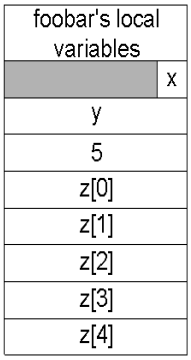


a) Original message on the Pentium
b) The message after receipt on the SPARC
c) The message after being inverted. The little numbers in boxes indicate the address of each byte

Parameter Specification and Stub Generation

```
foobar( char x; float y; int z[5] )
{
   ....
}
```

(a)

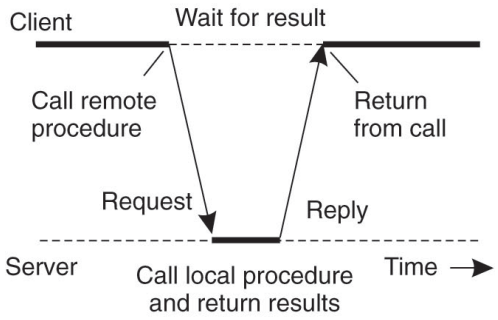


(b)

(a) A procedure

(b) z[5] is passed by copy/restore. It cannot be passed by reference

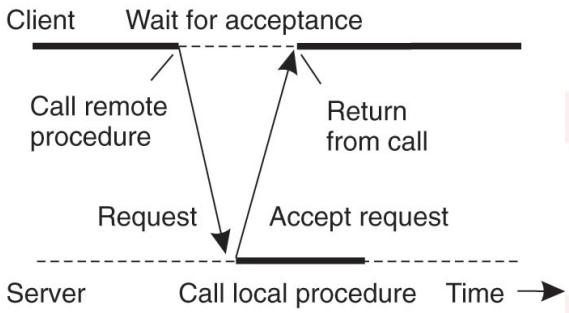
Synchronous vs asynchronous RPC (1)



(a)

The interaction between client and server in a traditional (synchronous) RPC.

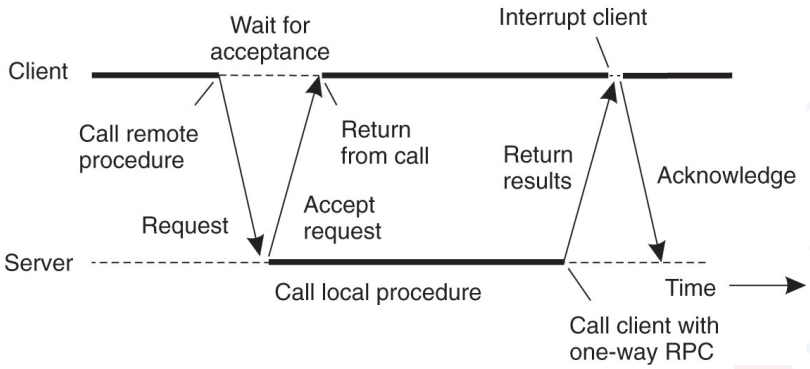
Synchronous vs asynchronous RPC (2)



(b)

The interaction using asynchronous RPC.

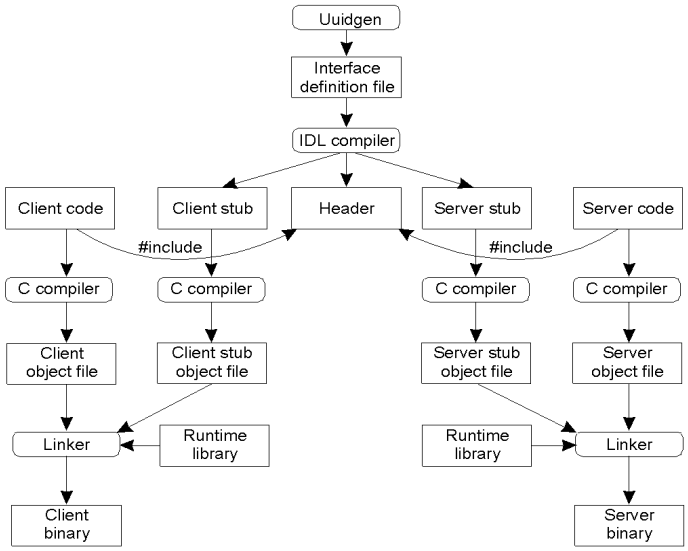
Deferred synchronous RPC (3)



A client and server interacting through two asynchronous RPCs.

Writing a Client and a Server

The steps in writing a client and a server for RPC.



Every Ware Lab

Modern RPC example: gRPC

- Open source remote procedure call (RPC) framework
- Largely used to build low latency, scalable distributed systems
- Supporting many languages (C/C++,Java, Go, Python, ...)
- Originally developed by Google
- https://grpc.io/

Every Ware Lab

Example of IDL and stub generation in gRPC

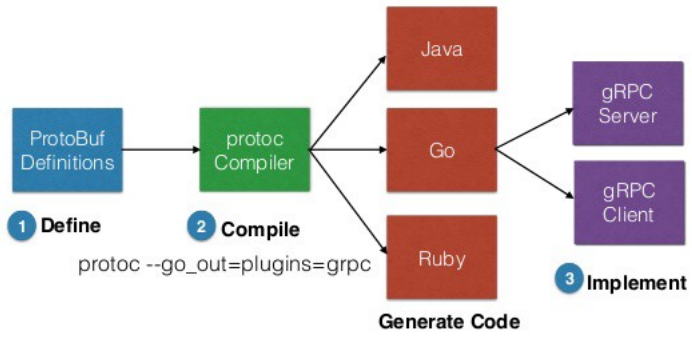


Figure from https://devopedia.org/grpc

Binding a Client to a Server

- Registration of a server makes it possible for a client to locate the server and bind to it.
- Server location is done in two steps:
  - Locate the server's machine.
  - Locate the server on that machine.

Binding a Client to a Server