Algorithmic Data Science: Distributed Computation

Fernando Rosas

UNIVERSITY
OF SUSSEX

Recap of last class

- Processes are programs (sequences of instructions that often take inputs and generate outputs), which run either in parallel or concurrently sharing CPU cores.
- Processes may have one or multiple threads — which are lightweight processes themselves.
- Processes have a life cycle, which includes creation, being ready, running, waiting, and termination.
- Processes can be synchronised/coordinated via locks or other methods.

Overview for today

- What are data centres
  - How to synchronise access to shared resources
  - What is a remote procedure call
  - What is a fault tolerant file system
- How authentication and privacy can be enabled
  - What is hashing
  - How do encryption works

What is a data centre

What is a data centre

As data scientists, you will access a data centre via:

- a link that connects an external I/O system into a Data Centre
- high bandwidth - 1 GBit/s or higher
- Low latency

You will be a *client*, and the data centre will be a *server.*

What is a data centre

As client, you may use “machines” that are are:

- Within the data centre
- Often virtual, running on a physical machine
- Most likely running versions of Linux
- Your processes will send messages to each other to communicate
- These message will use the TCP protocol for reliable delivery

Client-server architecture

Scenario: a server offer services desired by (potentially many) clients. All requests from clients and services provided by the server are delivered over a network.

How it works:

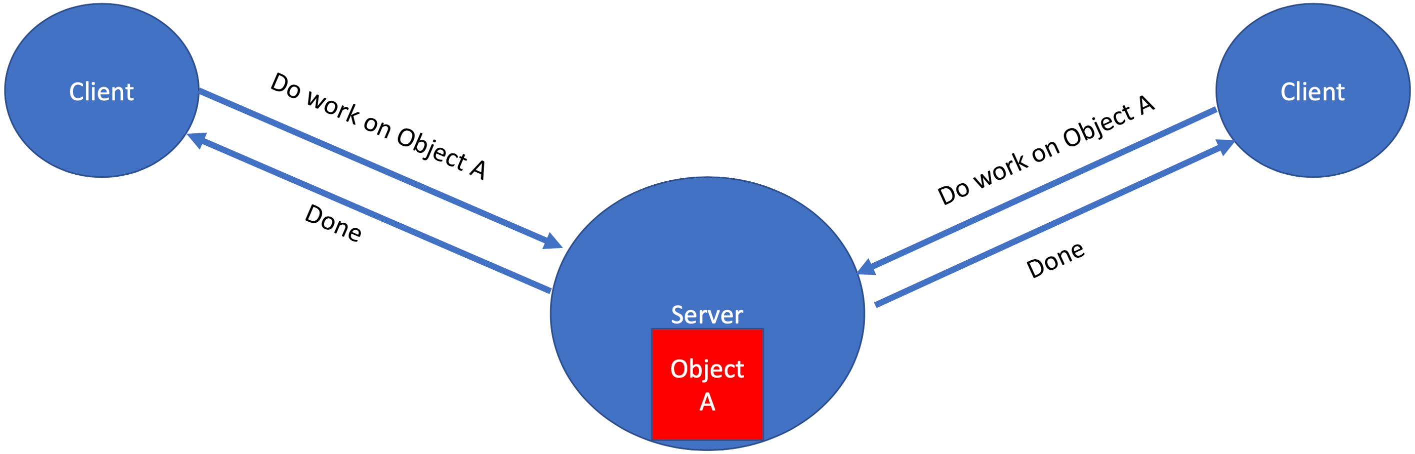
- First, the client sends their request via a network-enabled device
- Then, the network server accepts and processes the user request
- Finally, the server delivers the reply to the client

… this is similar to how processes share a CPU core!

Synchronisation across machines

Data centres are usually accessed and used by multiple users/clients.

How can we ensure that the different machines don't corrupt *Object A*?



Challenge: here there is no centralised scheduler…

Synchronisation across machines

→ One solution: use file locking (similar to locks for processes…)

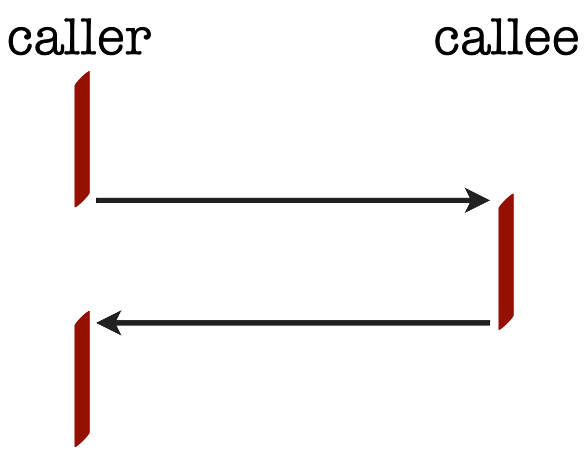
- Often distributed systems hold multiple copies of the same file. So a lock implies messages notifying that all copies of the file are locked as well.
- Locking only takes place after confirmation of local locking has been made.

Remote Procedure (function) Call

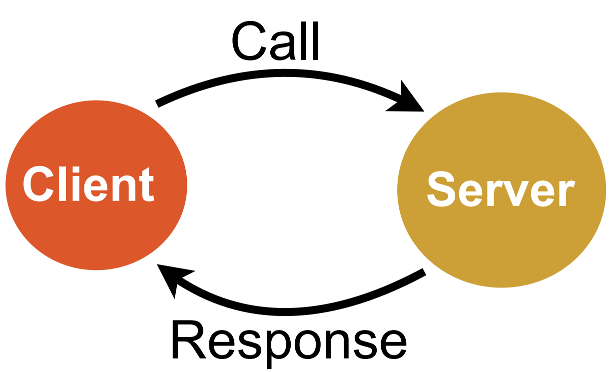
RPC: Calling a function that is executed in a server

- Abstraction, such that is transparent for the user if function runs locally or elsewhere.
- Needs to account for possible problems with online messaging (e.g. send error if no answer arrives in a given amount of time…)

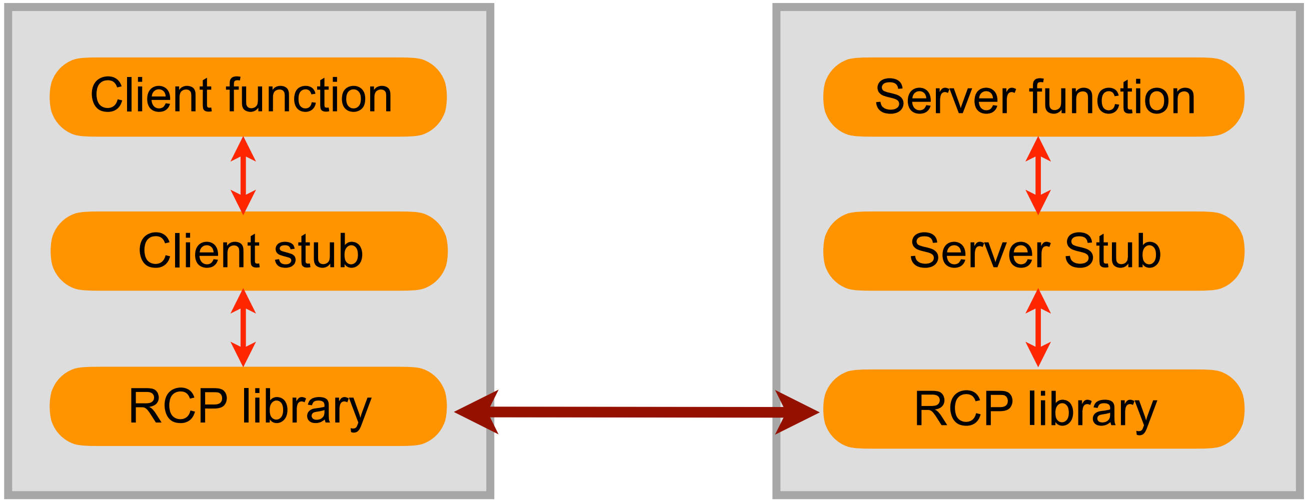
function call



remote function call



Remote Procedure Call



Popular RPC frameworks: Avro, gRPC, apache thrift…

Fault-tolerant distributed systems

Fault-tolerance: If a machine fails, then we can still provide a service

— > **key idea**: have *redundancy*

For example, multiple machines providing the same service (such as storing data)

- Probability of total failure (such as data being lost) is reduced, since data is replicated across multiple machines.
- If probability of failure is p for a given machine, then probability of loss of service with *n* machines is p^n, and the availability of the service is $1 - p^n$.
- If the mean time between failure for 3 machines is 5 days, and repair time is four hours, then assuming independence of failure $p = \dfrac{4}{5 \times 24} = 0.03$, the availability is $1 - 0.03^3 = 99.96\,\%$

In this case, one replicates data and monitor operations to enable fault-tolerance

The Google File System (GFS)

Example: GFS is designed for massive data that rarely gets updated

- 64MB chunks are replicated over chunk servers.
- Meta Data (including where each chunk is, and which chunk is what file) is stored on master servers.
- Master servers monitor the work of chunk servers.

Securing Machines

Goal: Prevent misuse of distributed systems

- Definitions
- Hashing
- Authentication
- Symmetric encryption
- Private and public key encryption

Security definitions

Misuse can be either *accidental* or *intentional*.

- Security is to prevent intentional misuse.
- Protection is to prevent either.

Security is made of three key pieces:

- **Authentication**: identify who each user is
- **Authorisation**: control who is allowed to do what
- **Enforcement**: ensure that users only do what they are allowed to do

Authentication

Most common approach to implement — via passwords

- Acts as a shared secret between two parties. Since only I know password, machine can assume it is me.
- Problem 1 system must keep copy of secret, to check against password. What if malicious user gains access to this list of passwords?
- Encryption Transformation on data that is difficult to reverse - in particular, secure digest functions.

Hash Functions

A secure digest function $h = H(M)$ has the following key properties:

- Given M, is is easy to compute h.
- Given h, it is hard to compute M.
- Given M, it is hard to find another M' such that $H(M) = H(M')$

Usually, any small change in M generates a big change in h…

Typical hashing functions: MD5, SHA-1, SHA-256, SHA-512.

Hashing in Python

```
a = ‘hello world’
h = hash(a)


import hashlib
a = ‘hello world!’
h_md5 = hashlib.md5(a.encode(‘UTF-8’))
h_sha1 = hashlib.sha1(a.encode(‘UTF-8’))
```

Password management via hash

Consider a user needs to authenticate in a server using a password via a public channel.

- Password should not be sent publicly.
- List of passwords should not be stored — could be hacked!

Idea:

- Server only store a list of hashes of passwords.
- Users only send hash of their password, which is then compared against the list.

Hashed-Message Authentication

Can we use message digests to provide authentication?

- Provide a shared secret to both ends of communication K.
- Generate an authentication code for message m, denoted by $(K \mid m)$.
- Hash the message authentication code, $H(K \mid m)$.
- Send message m and hash $H(K \mid m)$.

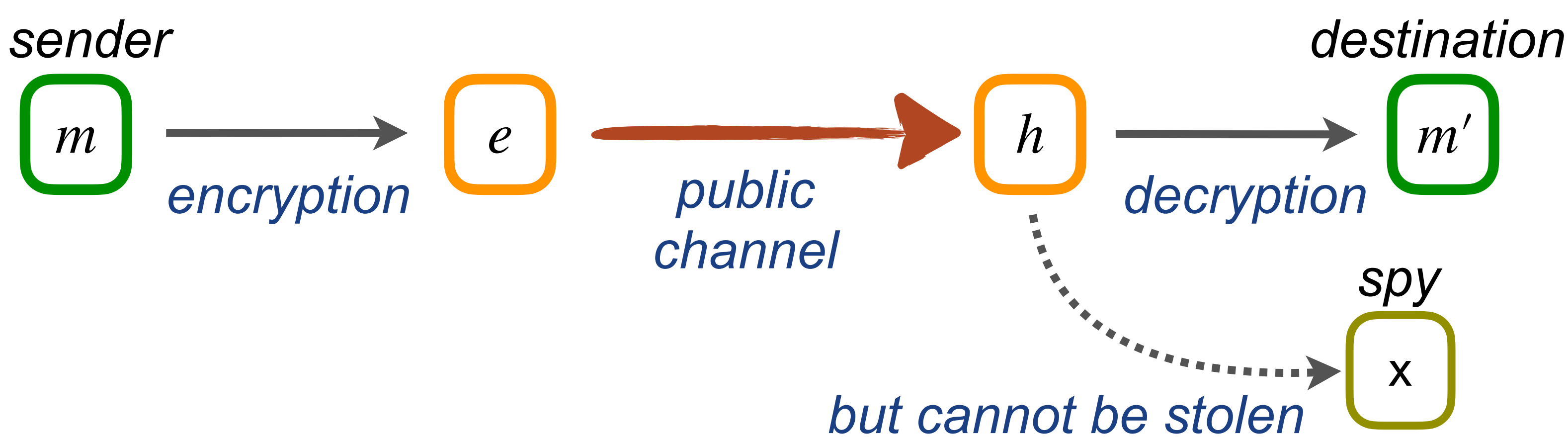
Receiver then regenerates $H(K \mid m)$ and compares with the received hash. Only knowledge of the shared secret and an unchanged message could generate the authentication code $\longrightarrow$ Provides authentication and integrity!

- ✦ Practical algorithms (e.g. HMAC) uses two keys and two hashes to provided added security: $H\left(K_2 \mid H(K_1 \mid m)\right)$.

Hashing vs encryption

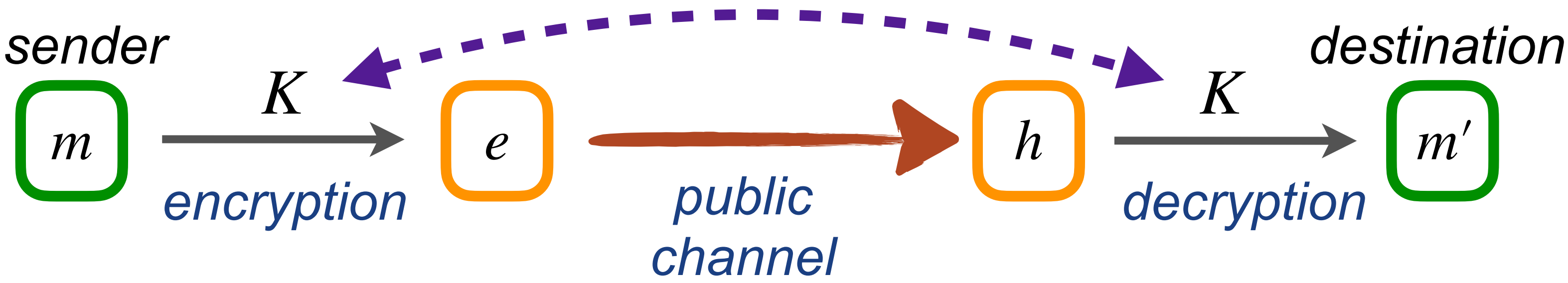
Hashing: a one-way process — easy to do, hard to undo.

Encryption: a two-way process, where decoding is possible under conditions…



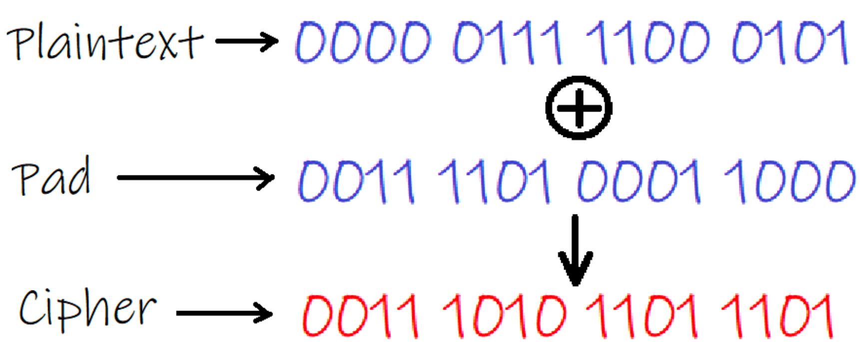
Symmetric encryption

Symmetric: same key is used for coding and decoding.



Simple, but not very flexible…

Example — *one-time pad* (OTP)



Private/public key encryption

Asymmetric: different keys are used for coding and decoding.

Usage 1: private transmission of data

if a receiver makes K_1 public and keeps K_2 private, people can encrypt using K_1 to send information privately.

Usage 2: authentication

if a sender keeps K_1 private and makes K_2 public, people can using K_2 to decrypt and hence authenticate the sender.

Private/public key encryption

Consider two users, ***Alice*** and ***Bob***:

- Both users create a pair (private-public) of keys, and share their public key via an open channel.
- To encrypt, Alice first encrypts with her private key, and then with Bob's public key.
- Bob decrypts the message using Alice's public key and his private key.

This scheme achieves two goals at the same time:

- Privacy: only Bob can decrypt the message because only him has his private key.
- Authentication: the fact that the message was successfully decoded using Alice's public key authenticates that is coming from her.

Conclusions

- **Client-server** architecture is pervasive in distributed systems. Most data science centres are accessed in that way.
- **Distributed systems** have their own ways to offer computational services, coordinate users, and guarantee robustness.
- Security is crucial in distributed system for controlling who can do what (**authentication**) and who can access what information (**privacy**).
- **Encryption** is a key technology to enable both authentication and privacy in distributed systems.