ELP

Jan 2024

pierre.francois@insa-lyon.fr

Communications de service

ELP

- GO (PFR), ELM(TRO), JS (SFR)
- Groups
  - 2-3 ppl. MUST remain the same through the projects
  - Register on moodle before the holidays
    - Provide link to github repository
      - 3 directories: js, go, elm
    - Names
- Deadline for last commit
  - 05/02/2024

GO

- **Debugging** support provided on demand in my office/discord, not by email
  My office/discord:
    - Show up randomly: maybe
    - Appointment: always
- **“Cheating” is allowed**
  - exchange hints with other groups, show some code examples.
  - Ask ChatGPT
  - **NO BLIND COPY/PASTE, big trouble if you can’t explain your own code (as in, <=7/20)**

GoLang mini-project

pierre.francois@insa-lyon.fr

- Golang
  - Bitcoin fork peer-to-peer crawler → IP mapping
  - Data crunching for International Committee of the Red Cross
  - Whatever I was doing in Perl I did in Python
  - Whatever I was doing in Python I now do in Golang
    - Same goals
      - “Better”
      - Faster

Golang project Objectives

- Get the hands on the Go language
  - “I did some golang at school” vs.
    “I have delivered a PoC concurrent server in go”
- Implement an algorithm that benefits from concurrency
  - Define input/output data
  - Describe concurrency approach, implement with goroutines
- Do networking stuff
  - Implement a Client-Server application
  - TCP session pool management using go concurrency
- Evaluate performance

Sources

- Installing go
  - https://golang.org/doc/tutorial/getting-started
  - Install VSCode / GoLand
- Learning go
  - https://tour.golang.org/basics/1
- Assess and keep track of project progress
  - GIT, GIT, GIT
  - ~~Feature tracking: The ugly spreadsheet approach~~ Git issues

Target application

Implements an algorithm to solve a problem by taking advantage of concurrency

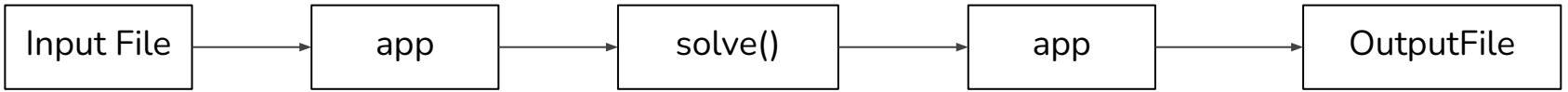
One problem → Multiple functions running to jointly solve the problem

Provides the execution of the algorithm as a service over TCP

Multiple clients → Multiple runs of the algorithm “in //”

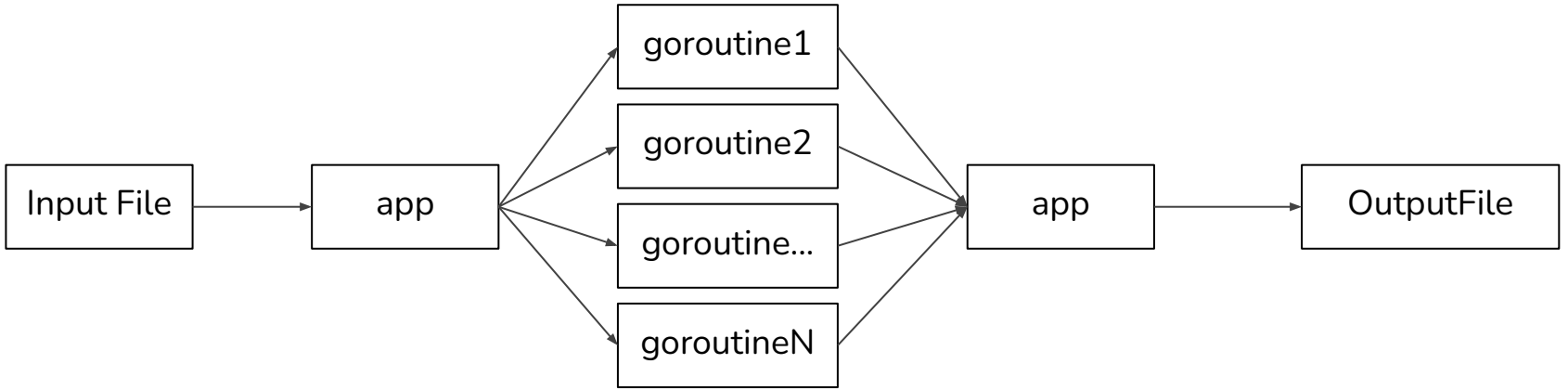
Target application design (0)

Local, non-concurrent application



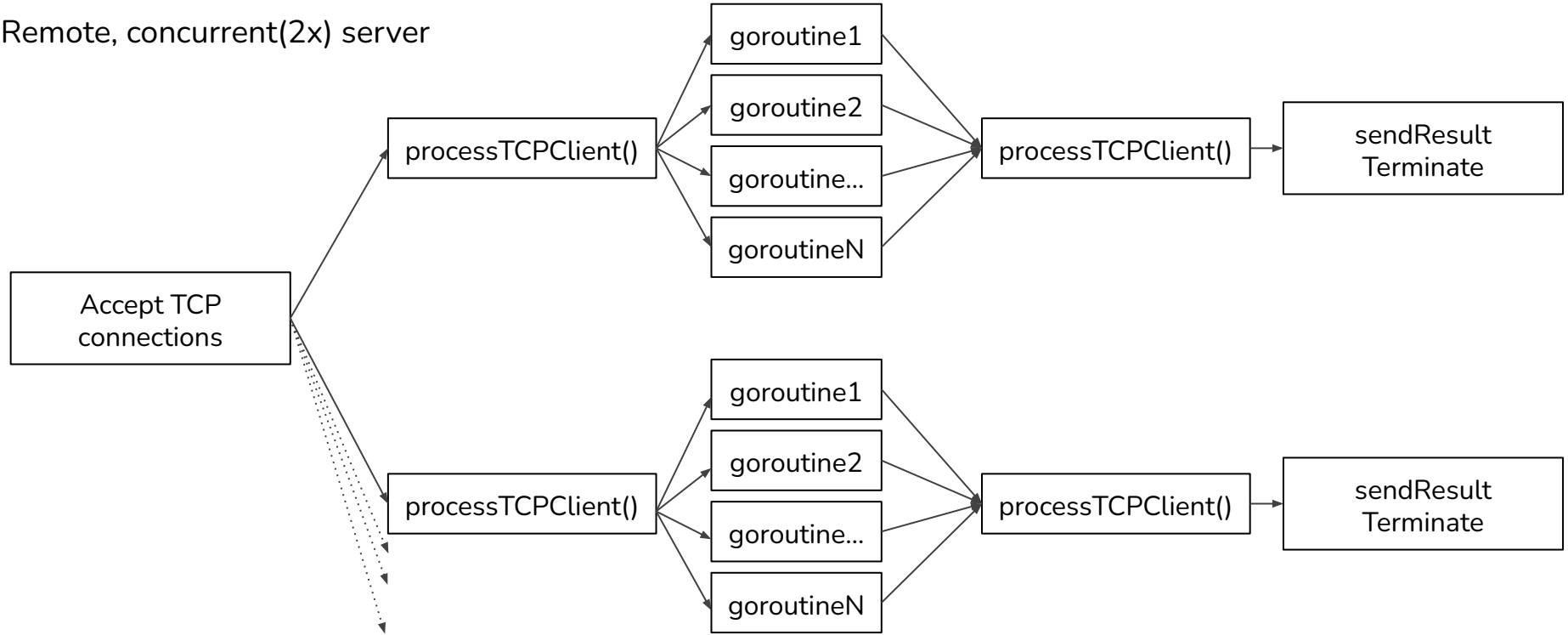
Target application design (1)

Local, concurrent application



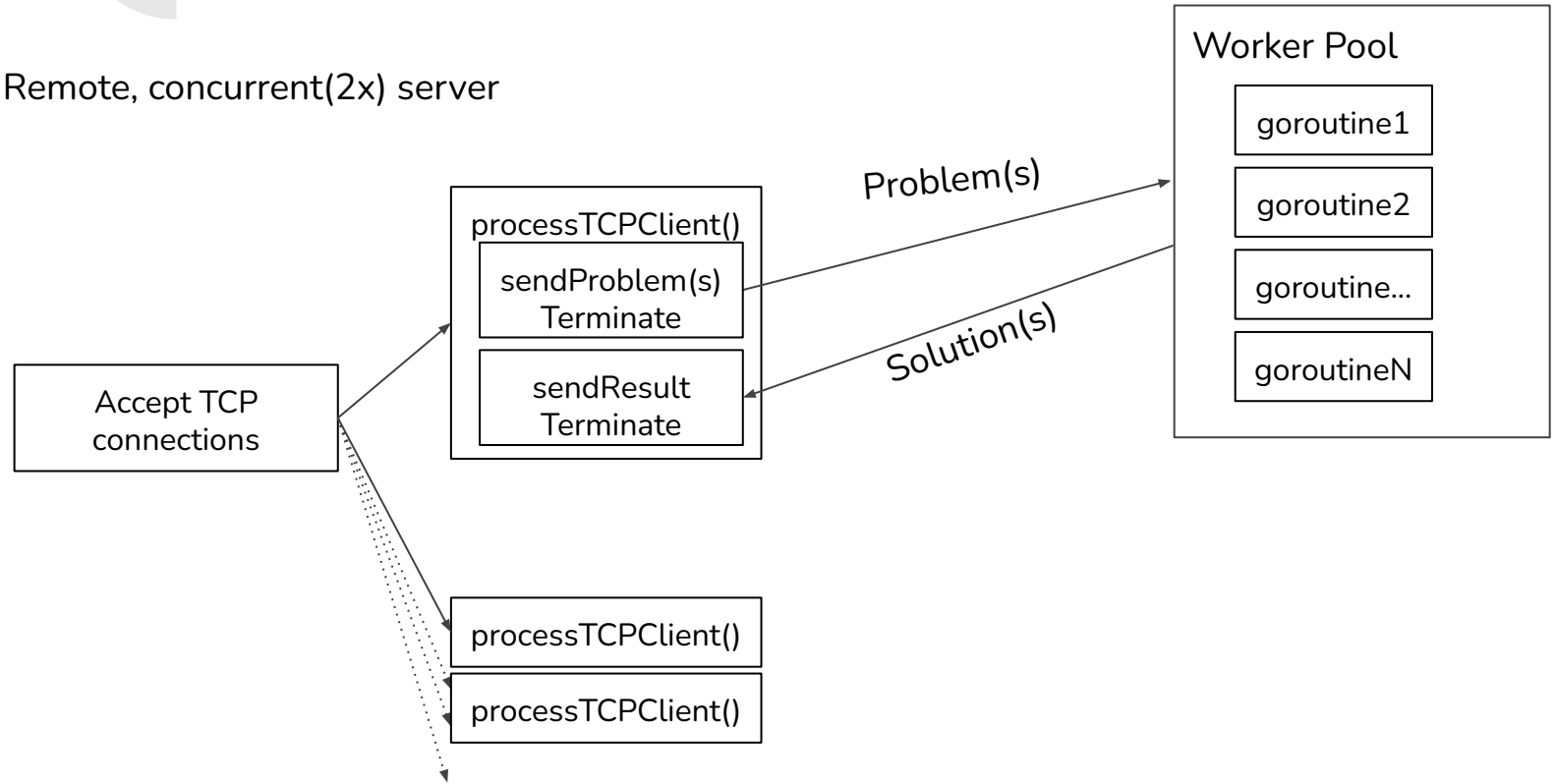
Target application design (2)

Remote, concurrent(2x) server



Target application design (3)

Remote, concurrent(2x) server



Target application

One of

- Huge matrix multiplication
- Levenshtein distance to find doublons and match names from various sources
  - ~(L^2)*(K^2)
- A non linear graph algorithm (less ez)
  - All-Pair Shortest Paths (N x Dijkstra)
  - Random walks (e.g. use it to approximate Google's pagerank)
  - Community detection algorithms
- Image filters
- Self assigned application
  - Propose something (not too complicated)
  - Must be acked by me ASAP

We have a group, now what?

Not necessarily in this order:

- Decide the app you want to implement
- Get some go running on your machine
  - Read the lines of a text file
  - Extract and print the last word of each line
- Struggling setting up go on your machine?
  - get help from colleagues
  - tour.golang.org
- Create a git repository for your project
  - Create your first issues

Session 2 Go routines

- Goroutines
- Wait groups
- Channels

Go routines

- func myFunction()
- go myFunction()

Wait groups

- wg.Add()
- wg.Done()
- wg.Wait()
- Do not pass by value !

Channels

- Wait group limitations
- myChannel := make chan (<type>, int) // Create a Channel
- myChannel <- “coucou” // Push
- x <- myChannel // or x:=<-myChannel //Pop
- Protected datastructure
- Blocking datastructure
    - On push
    - On pop

Session 3 TCP

- TCP sockets, Concurrent TCP server, env. Limitations
  - Whiteboard
  - Code together
  - Discuss parsing content read from TCP connection
  - Discuss blocking read
- Project
  - Status report review
  - Code rush

TCP

- Bidirectional byte delivery mechanism between applications
  TCP flow: Source IP, Source TCP Port, Destination IP, Destination TCP port
- Connection oriented
- Reliable
- Ordered

- No message boundaries: stream of bytes, application defined message boundaries within the stream of bytes

Client

- Connect

  conn, err := net.Dial("tcp", portString) //portString: “IP:Port”, eg “127.0.0.1:80”
- Disconnect

  conn.Close()

  defer conn.Close()
- Get yourself a reader on the connection, read some characters

  reader := bufio.NewReader(conn)

  message:= reader.ReadString(‘\n’)
- Write content on the connection

  io.WriteString(conn, fmt.Sprintf("Coucou %d\n", i))

Server

- **Take** a TCP port on the machine and ask connection attempts to that port to be redirected to your app

  ln, err := net.Listen("tcp", portString) //":8000"
- Accept a new connection on that port

  conn, errconn := ln.Accept()
- Close the connection of a client

  conn.Close()

- Receive/Send byte: same as for the client

Multi-client server

- Accept a new connection on that port

  conn, errconn := ln.Accept()
- Deal with that client in a go routine
  - go handleClient(conn) //To be written by yourself
  - Warning: Do not panic upon error, you have other clients to deal with
  - Warning: if you have a variable that depends on the connection, the scope of that variable should be per connection

Blocking functions

- Write is blocking until the bytes have been passed to your server/client net stack
  (You won’t feel it unless you write a huge amount of bytes)
- Read is blocking until you received the necessary bytes from the remote host
  You will feel it

Env Limitations

- A connection is a file
- An application can only have a limited number of open files

```
Pierres-MacBook-Pro-2:~ pfrancois$ ulimit -n
256
```

- Trying to open an additional connection will lead to an error

- Know it. Change config if needed