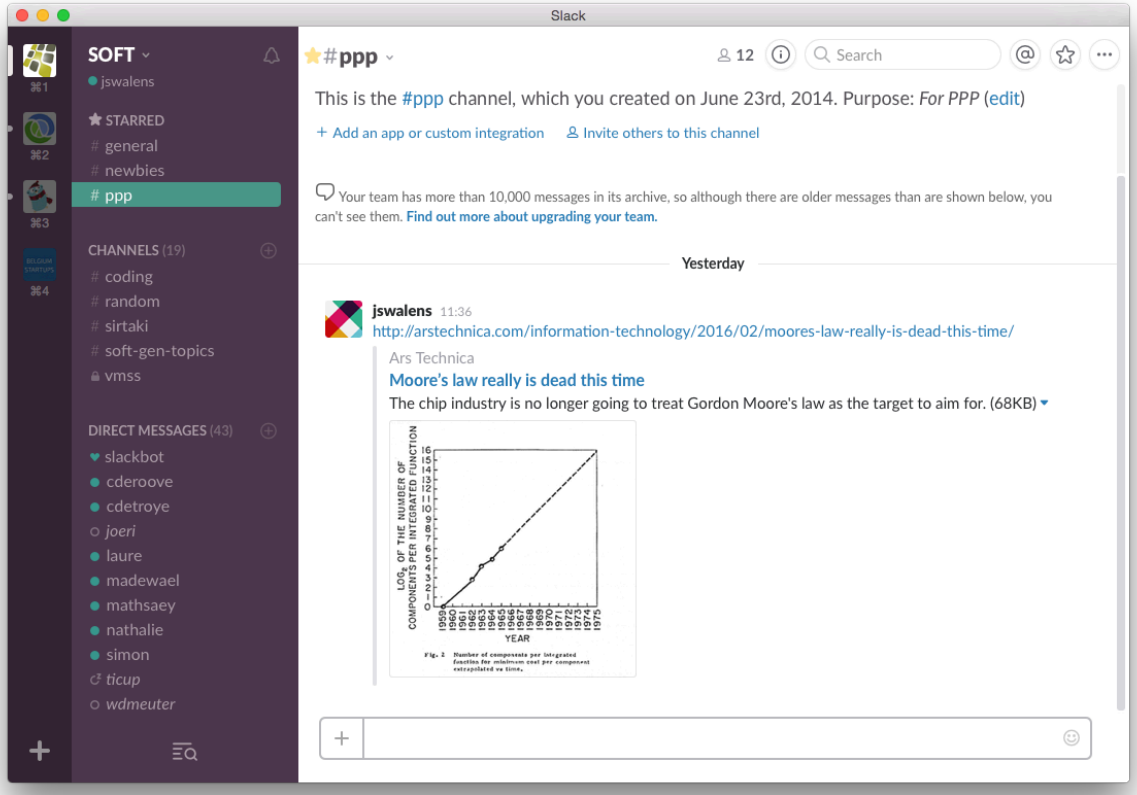
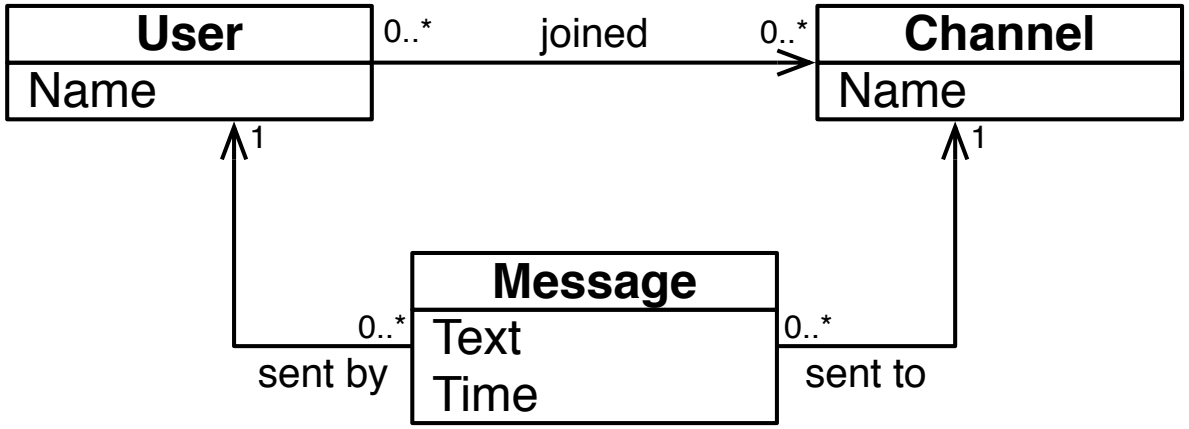
Multicore Programming

Project Erlang

Slack

Scalable chat room (Slack)





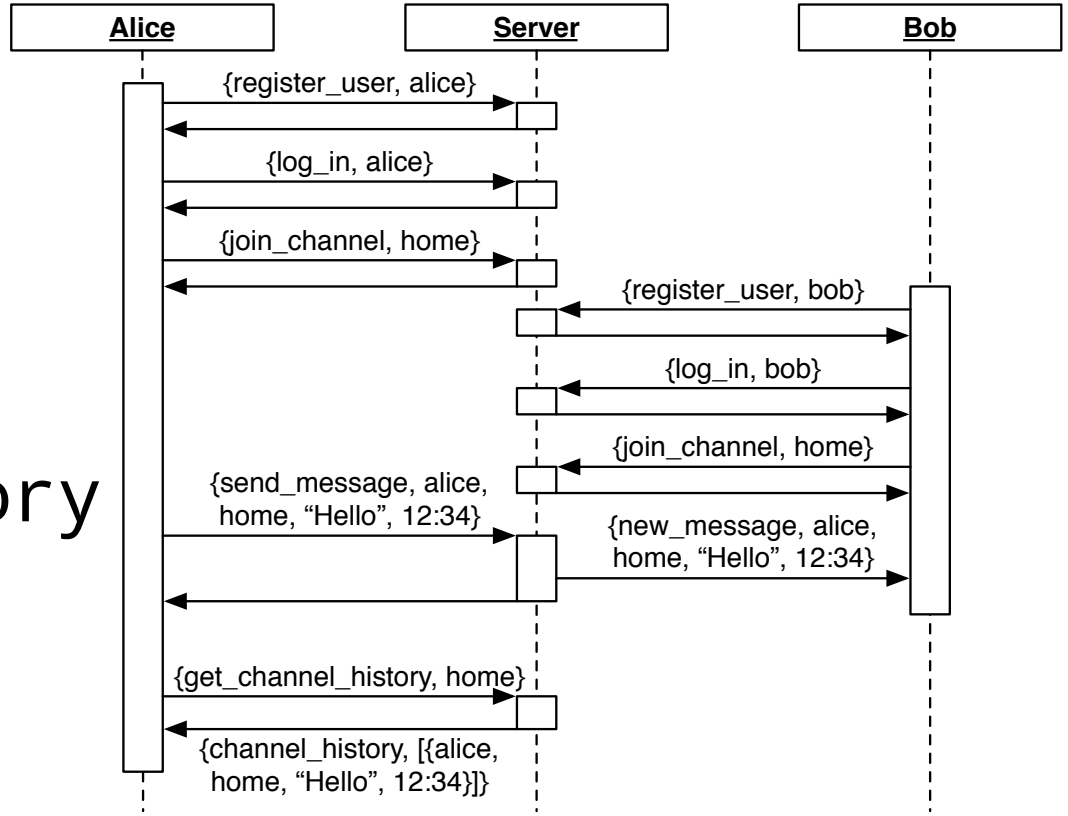
API

Client → server:

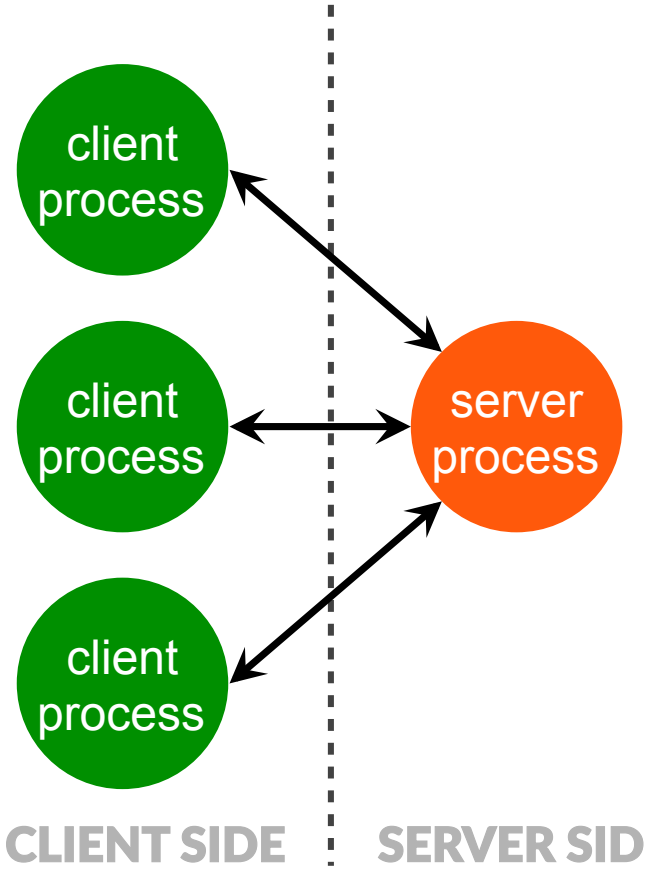
- `register_user`
- `log_in`
- `join_channel`
- `send_message`
- `get_channel_history`

Server → client:

- `new_message`



Example Implementation



- 1 process / client → don't change, for benchmarking
- 1 server process → you'll need to change this

Scalability over Consistency

Allowed to sacrifice consistency for scalability
E.g:

- Some requests might return stale data
- Data may be duplicated and be inconsistent for short periods
- Keep both channels for user as well as users for channel, both lists sometimes inconsistent

Describe motivations in report

Evaluation

Benchmarks: generate number of client processes, each do a number of operations (`send_message`, `get_channel_history`)

Latency & throughput of operations, in variety of situations

Best case, worst case, in between

Allowed to share benchmark code on PointCarré forum (only this, nothing else!)

Evaluation

Serenity = 64-core server at lab

Time slot between March 21st and April 6st
(will send out Doodle)

Remote log in: demo in lab session

Report may contain experiments on Serenity as well as local (≥ 4 cores)

Report

Table of contents in assignment sheet:

Implementation

- How do you ensure scalability?
- Where did you sacrifice consistency?
- Include diagrams to show design

Evaluation

- Describe set-up, metrics
- Include plots of results

Insight questions

- Hypothetical extensions

Details

Deadline: Friday 8th of April, 23:59

Submit ZIP with implementation & report on PointCarré

Project defense in June

⅓ of your final grade

Assignment sheet on PointCarré