Project Clojure:
Flight Reservation System

Multicore programming

Flight reservation

Flight:

```
{:id 0 :from “BRU” :to “ATL”
 :pricing [[600 150 0]
           [650  50 0]
           [700  50 0]]}
```

There are
150 seats available and 0 taken at €600;
50 seats available and 0 taken at €650;
50 seats available and 0 taken at €700.

Customer:

```
{:id 0 :from “BRU” :to “ATL”
 :seats 5 :budget 600}
```

I would like to book 5 seats for at most €600 per seat.

Updated flight:

```
{:id 0 :from “BRU” :to “ATL”
 :pricing [[600 145 5]
           [650  50 0]
           [700  50 0]]}
```

Sales

Every once in a while, the prices of **all** flights are reduced.

No sales

```
{:id 0 :from “BRU” :to “ATL”
 :pricing [[600 145 5]
           [650  50 0]
           [700  50 0]]}
```

TIME_BETWEEN_SALES

TIME_OF_SALES

Sales −20%

```
{:id 0 :from “BRU” :to “ATL”
 :pricing [[480 145 5]
           [520  50 0]
           [560  50 0]]}
```

Implementation

Parallelize to process customers concurrently

- **Correctness**: no corrupt data or race conditions
- **Performance**: maximize throughput (process all customers in minimal time)

Use any of Clojure's concurrency mechanisms (futures, agents, atoms, refs, promises…)

Correctness > performance

Sequential implementation given

Evaluation

Correctness

add unit tests to check thread-safety

e.g. no overbooked flights, only one reservation/customer, correct pricing

Performance

benchmarks to measure throughput with varying parameters

e.g. # flights, # customers, length/frequency of sales, # cores

some sample input is given, you can make new ones

Report

Table of contents in assignment sheet:

- Implementation
  - Which concurrency mechanism(s) and why?
  - How did you parallelize?
  - How do you ensure correctness?
  - What are potential performance bottlenecks?
- Evaluation
  - Correctness
  - Performance
- Insight questions

Details

Deadline: **Friday, 6th of May**, 23:59

Submit code and report (ZIP) on PointCarré

Project defense in June

⅓ of final grade

Assignment and sequential implementation on PointCarré