

# Project Clojure: Flight Reservation System

Multicore programming

# Flight reservation system

One-way ▾

1 passenger ▾

Economy ▾

○ Brussels Brussels

↔

📍

Atlanta ATL

📅

Sat, 2 May

< >

Bags ▾

Stops ▾


Airlines ▾

Under €700 ✕


Times ▾


Connecting airports ▾


More ▾


 Track prices ⓘ

☐

 Date grid

 Price graph

 Nearby airports




Prices are currently **low** – €310 cheaper than usual for your dates.

Details ▾

## All flights

Total price includes taxes + fees for 1 adult. [Additional bag fees](#) and other fees may apply.

Sort by: ↑↓

|  |                      |                     |        |
|--|----------------------|---------------------|--------|
|  <div>06:00 – 19:49<br/>Tap Air Portugal, JetBlue</div> | 19 h 49 m<br>BRU–ATL | 2 stops<br>LIS, BOS | €378 ▾ |
|--|----------------------|---------------------|--------|

2

# Flight reservation

Flight:

```
{:id 0 :from "BRU" :to "ATL"  
 :carrier "Delta"  
 :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"  
 :carrier "Delta"  
 :pricing [[600 145 5]  
           [650  50 0]  
           [700  50 0]]}
```

# Sales

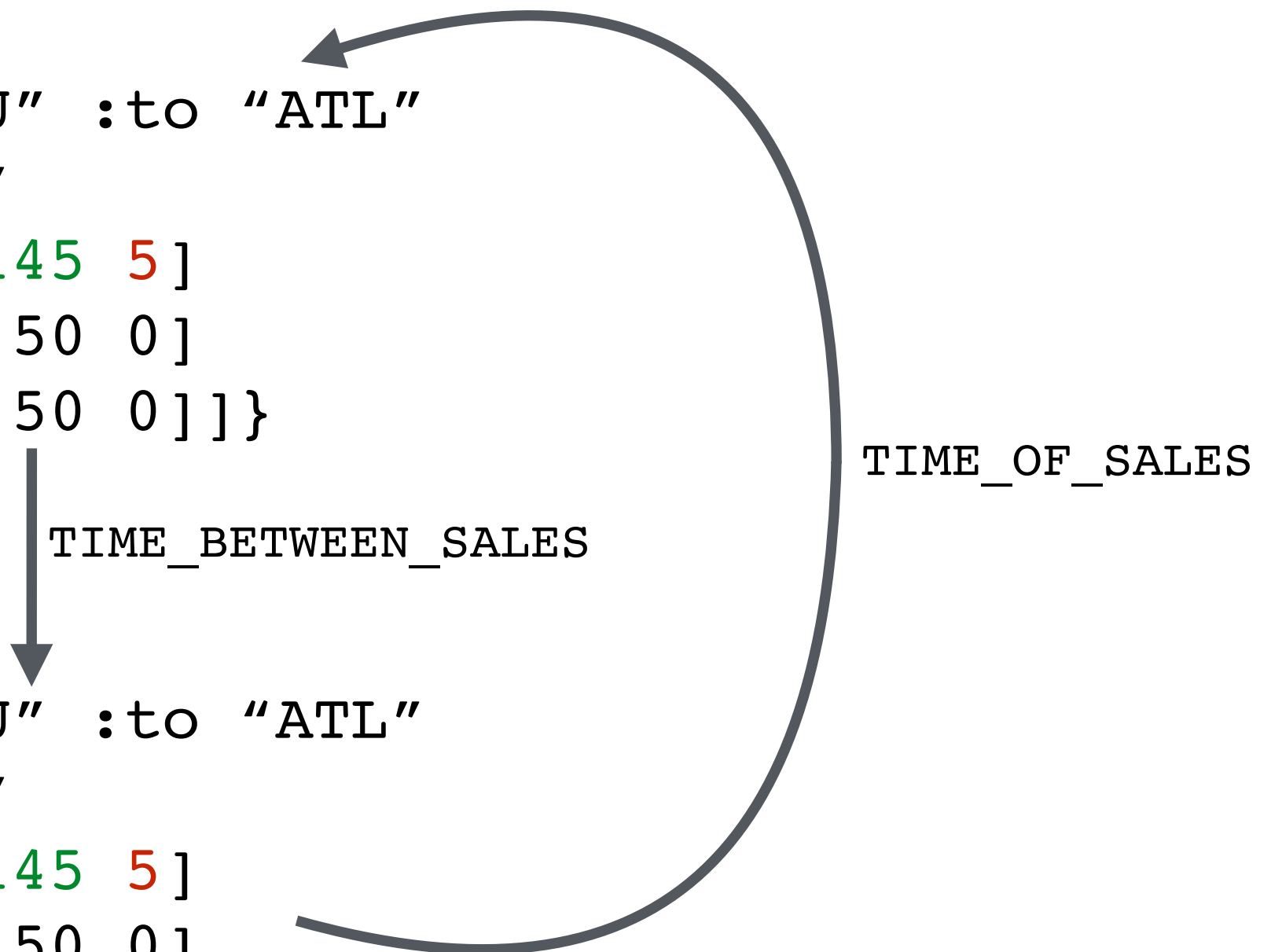
Every once in a while, the prices of all flights for **one, randomly chosen**, carrier are reduced.

No sales:

```
{:id 0 :from "BRU" :to "ATL"  
 :carrier "Delta"  
 :pricing [[600 145 5]  
           [650  50 0]  
           [700  50 0]]}
```

Sales –20%:

```
{:id 0 :from "BRU" :to "ATL"  
 :carrier "Delta"  
 :pricing [[480 145 5]  
           [520  50 0]  
           [560  50 0]]}
```



# Implementation

Sequential implementation given

Parallelize to process customers concurrently

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

Correctness > performance

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

# Evaluation: correctness

Add unit tests to check thread-safety

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

# Evaluation: performance

Create three benchmarks to measure throughput with varying parameters

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

In different scenarios, e.g. “best” case, “typical” case.

E.g. customers all book different flights, many customers for same flight...

Some sample input is given, you can add new ones.

```
(ns input-simple)

(def flights
  [{:id 0
    :from "BRU" :to "ATL"
    :carrier "Delta"
    :pricing [[600 150 0] ; price; available; occupied
              [650 50 0]
              [800 50 0]]}
   ...])

(def customers
  [{:id 0 :from "BRU" :to "ATL" :seats 5 :budget 700}
   {:id 1 :from "BRU" :to "ATL" :seats 5 :budget 550}
   ...])

(def carriers (distinct (map :carrier flights)))

(def TIME_BETWEEN_SALES 50) ; milliseconds
(def TIME_OF_SALES 10)
```

# Evaluation: performance

Perform **three** experiments:

- one experiment to measure speed-up (= throughput as number of cores increases)
- two other experiments, free to choose

Other tip: if you use atoms or transactions, measure how often they re-execute. Relate this back to the chosen parameters.



# 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: **Sunday, 4<sup>th</sup> of June, 23:59**

Submit code and report (ZIP) on Canvas

Project defense in June

$\frac{1}{3}$  of final grade

Assignment and sequential implementation on Canvas