KU LEUVEN

# Algorithms for Decision Support

## Lab 25-26

Greet Vanden Berghe

Lisa Garcia Tercero (lisa.garciatercero@kuleuven.be)

Augustin Delecluse (augustin.delecluse@kuleuven.be)

# Practical Information

- **5 labs** of 3h each – last session: presentation of results
- **1 project** to be completed during lab sessions + working offline
- Work in groups of **2**
- Deliver reports and present your model in the final session

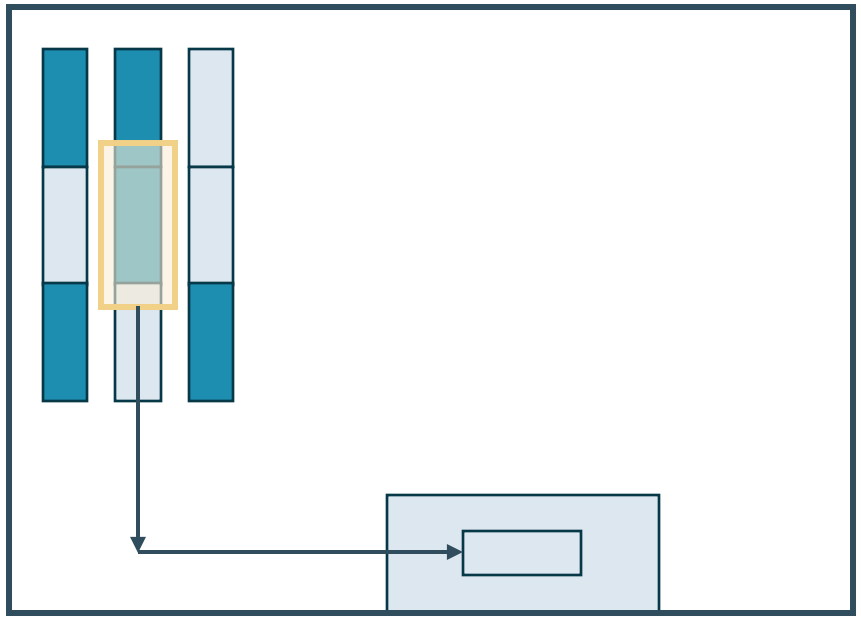
# Project

## Container loading and unloading on a terminal

Picture: PSA Antwerp

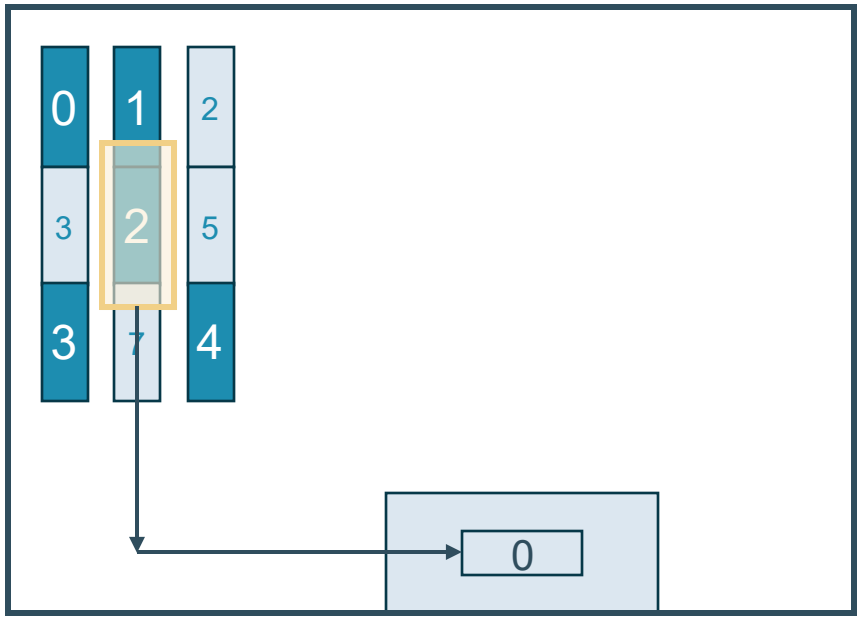
# Project

**Dispatching rules** for stradle carriers: transfer containers from crane to yard and vice versa

# Problem features

- Initital location of the cranes, containers, and straddle carriers
- List of demands for each crane: loading and unloading




- Load 0 2 → load container 2 onto crane section 0
- Load 0 3 → load container 3 onto crane section 0
- Unload 0 5 2 → unload container 5 from crane section 0 to storage section 2
- …

# Problem Input

| | |
|---|---|
| 30 75 | Size of the map (width x height) |
| Crane section | |
| 1 | Number of cranes |
| 0 21 44 30 50 1 0 24 46 | |

# Problem Input

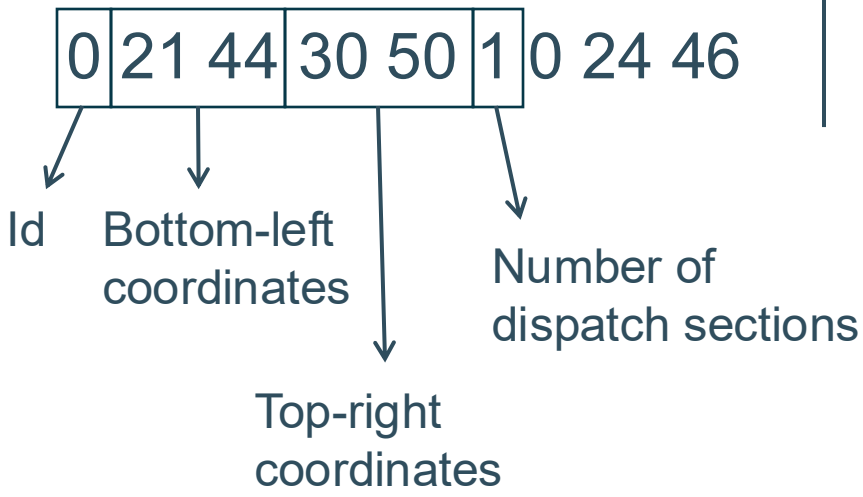
30 75

Size of the map (width x height)

Crane section

1

Number of cranes



# Problem Input

30 75

Size of the map (width x height)

Crane section

1

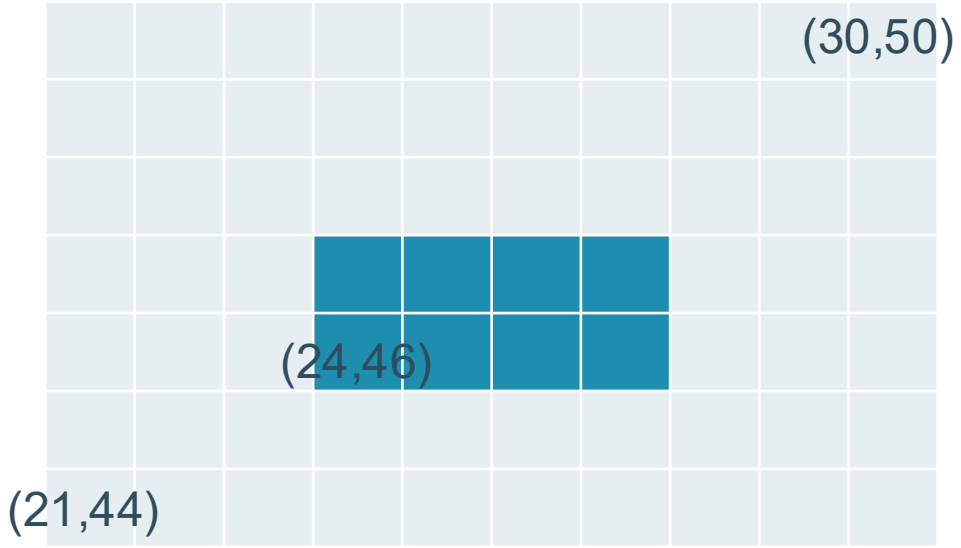
Number of cranes

0 21 44 30 50 1 0 24 46

Dispatch sections are always 4x2

Dispatch id

Bottom-left coordinates



# Problem Input

30 75

Size of the map (width x height)

Crane section

1

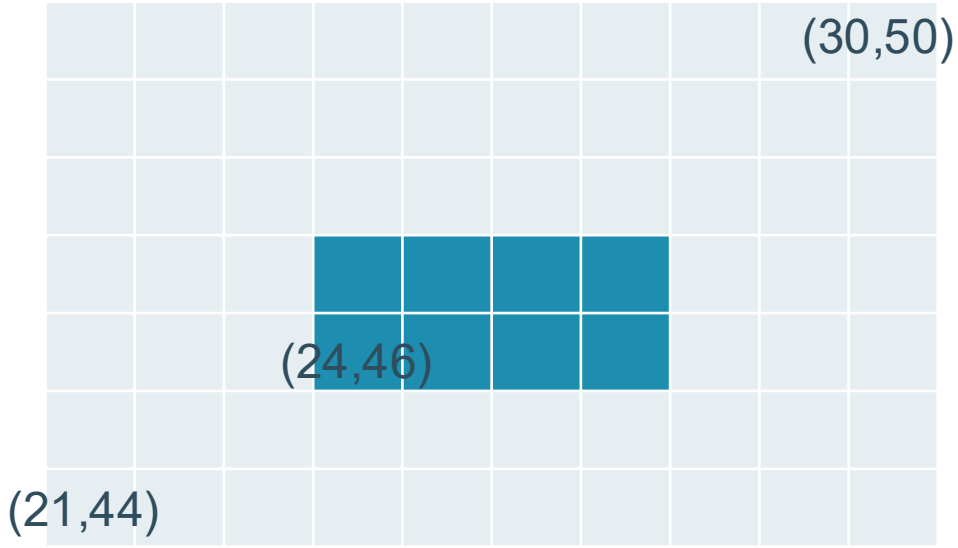
Number of cranes

0 21 44 30 50 1 0 24 46

Dispatch sections are always 4x2

Dispatch id

Bottom-left coordinates

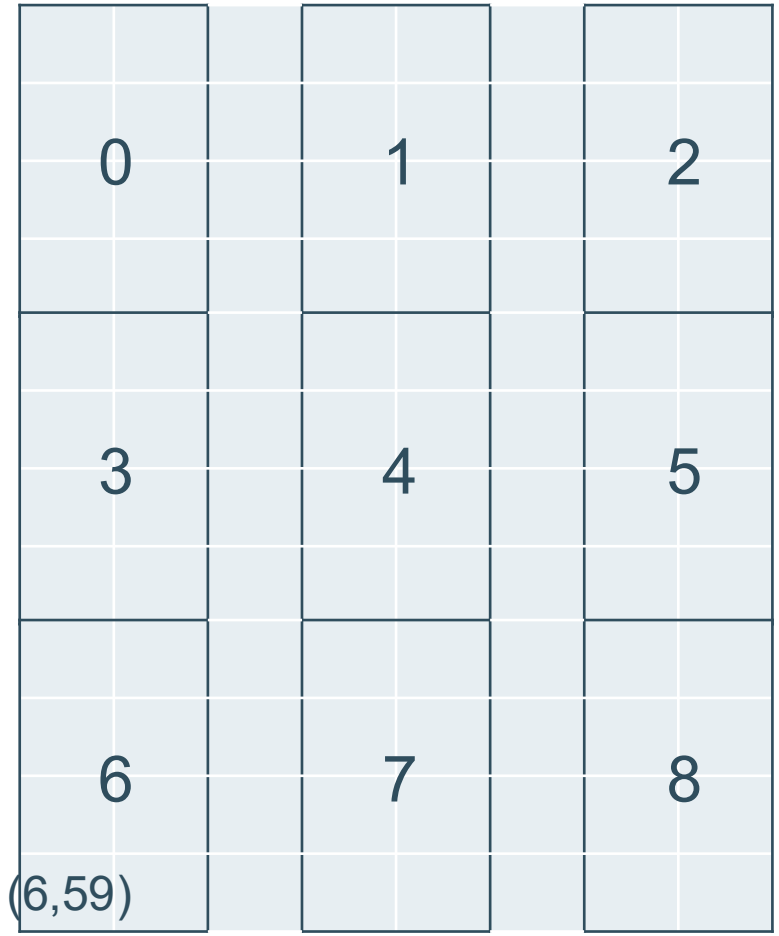



! Crane can only load/unload to ship if carrier is outside crane section

# Problem Input

Storage section
9
0 6 67
1 9 67
2 12 67
3 6 63
4 9 63
5 12 63
6 6 59
…

Number of storages
Storage id, bottom-left coordinates
→ always 2x4



# Problem Input

Carrier section

1

0 0 8 61

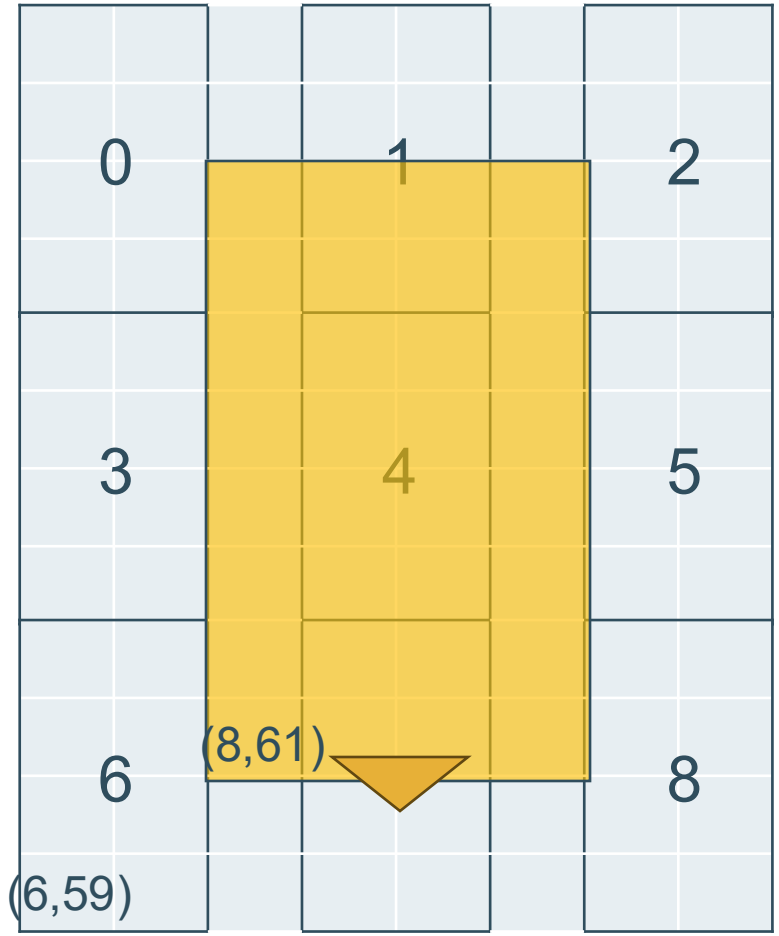
Number of straddle carriers

Id, crane assignment,

bottom left coordinates

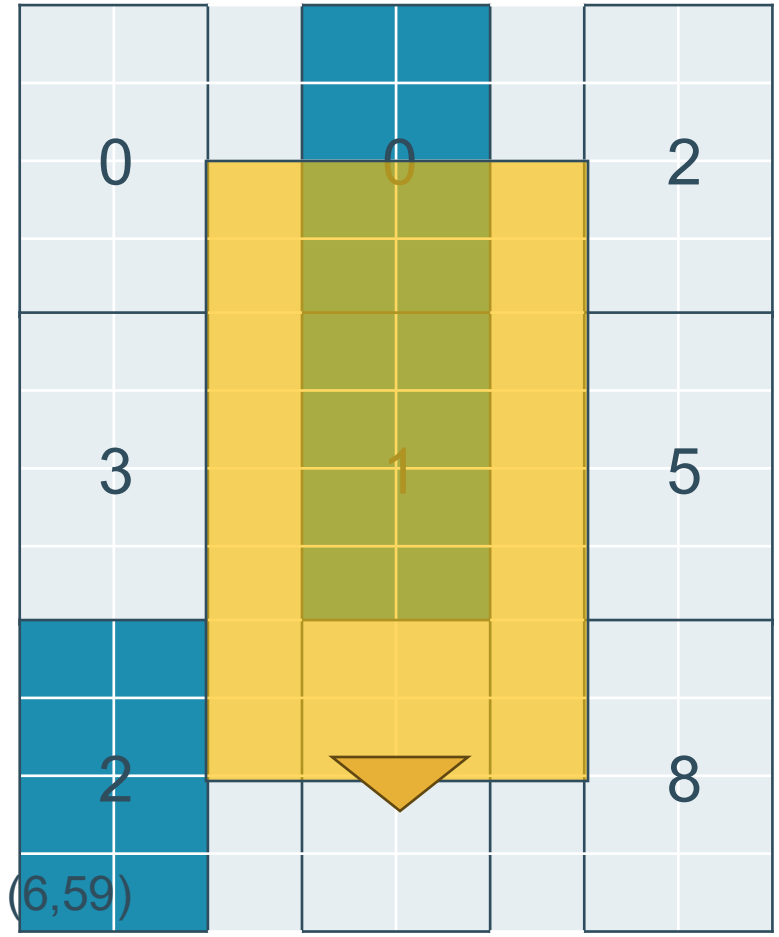
→ A carrier is always 4x8 (can rotate)

and is originally oriented downwards



# Problem Input

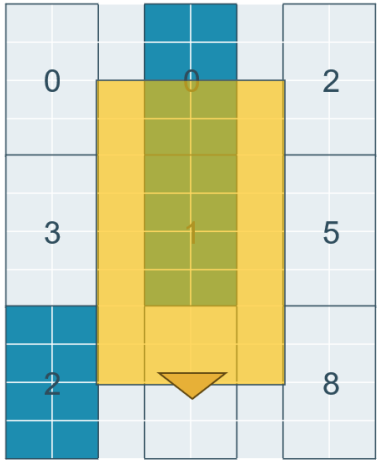
| | |
|---|---|
| Container section | |
| 3 | Number of containers initially present in the storage |
| 0 1 | Container id, storage id |
| 1 4 | → A container is always 2x4 |
| 2 6 | |

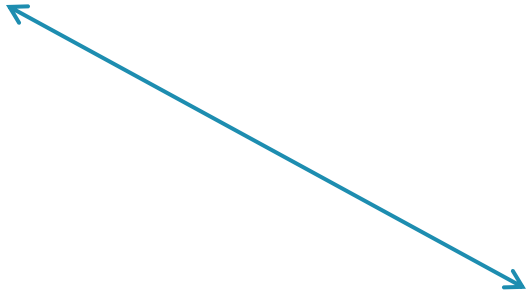


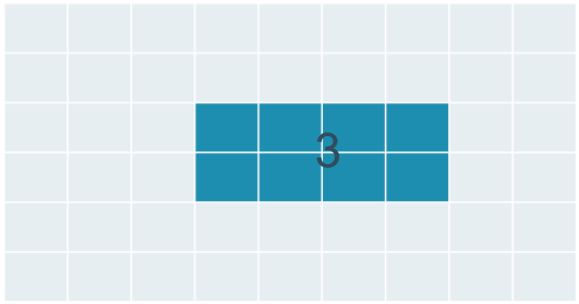
# Problem Input

| | |
|---|---|
| Demand section | |
| 2 | Total number of new containers that will arive in the port |
| Demand crane 0 | Demand for each crane |
| 1 0 | Number of ships incoming + list with id of each ship |
| Ship 0 | Demand for each ship |
| 4 | Number of operations this ship will require |
| Unload 0 3 0 | Pick up at discharge 0 container 3 and put it at storage 0 |
| Unload 0 4 3 | Pick up at discharge 0 container 4 and put it at storage 3 |
| Load 0 2 | Bring container 2 to discharge 0 |
| Load 0 1 | Bring container 1 to discharge 0 |

# Problem input






Unload 0 3 0
Unload 0 4 3
Load 0 2
Load 0 1

# Problem output

- List of **dispatching rules** for the carriers

| Carrier 0 | Time stamp, operation |
|---|---|
| 0 move 18 | at t=0, move 18 units (downwards) |
| 18 face right | at t=18, rotate to the right |
| 19 move 16 | at t=19, move 16 units (right) |
| 35 load | at t=35, load the container onto the carrier |
| … | |
| | Possible operations: move, face, load, unload |

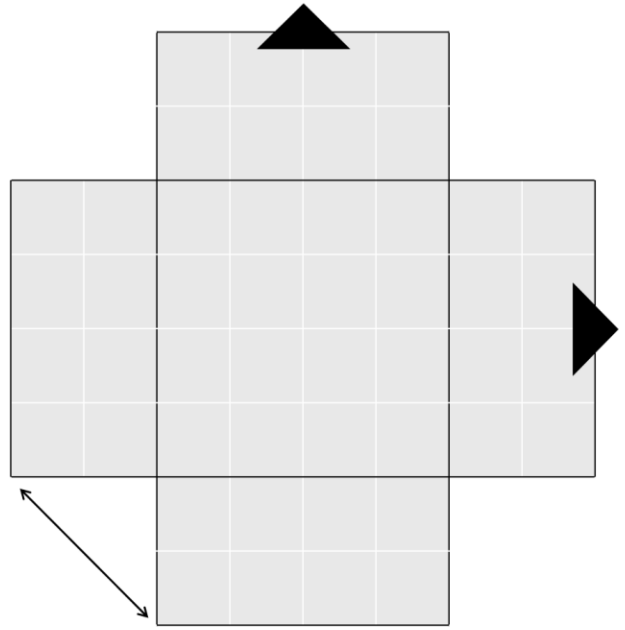
# Problem output

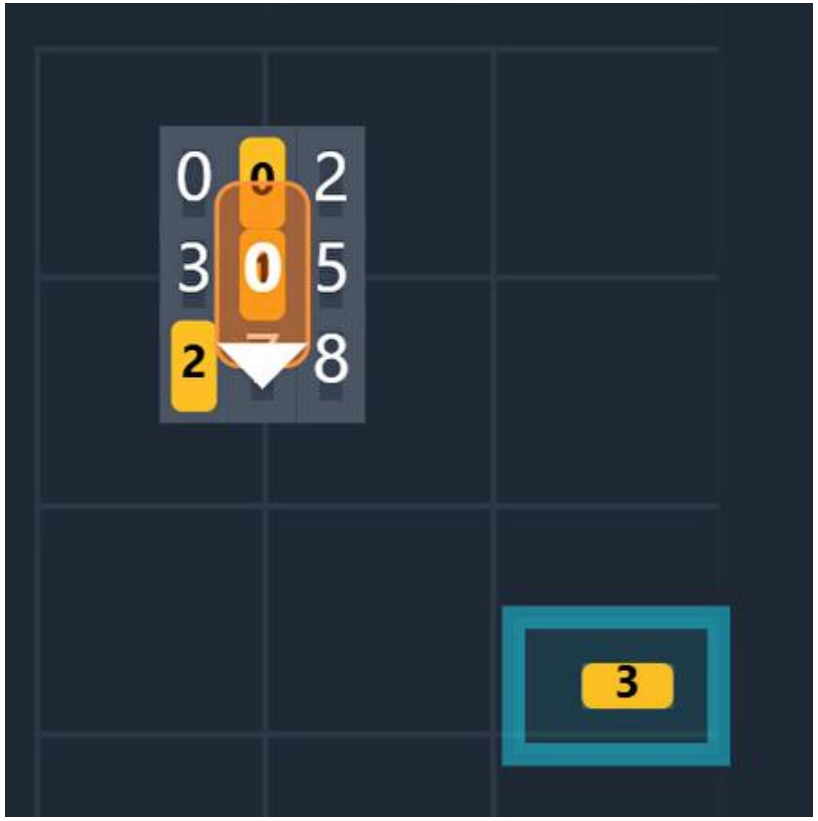
- List of **dispatching rules** for the carriers
- Possible operations: move, face, load, unload
- Each operation takes 1 unit of time
  - Except moving by X tiles, which takes X units of time

- Note: you can also move « backwards »
  - If you are facing right, you can move left by giving negative steps
    - « move -10 »
  - This takes less time than first facing left and then moving 10 steps

# Problem output

- « face » a different direction = rotate the carrier
- Since the carrier is 4x8, this is not straightforward
- The **center** of the carrier remain in the same place
- The bottom left coordinate moves by 2 units diagonally






Unload 0 3 0
Unload 0 4 3
Load 0 2
Load 0 1

# Problem restrictions

- Stradle carriers can **only carry one container** at a time
- Storage sections can store **at most two (stacked) containers** at a time

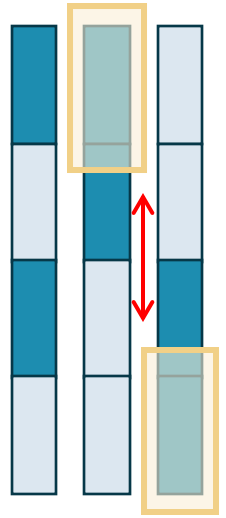
  →How to access the container on the bottom?

  →What if a storage is full?

- Containers are (un)loaded to/from the crane

  when no carrier is within its crane section

- Carriers cannot collide: **conflicts**



# Evaluation

- **Efficiency**

  How fast does the algorithm run + how much memory does it require?

  Use **efficient data structures** to store and access information

- **Flexibility**

  Solve for diverse instance types, also hidden instances

  Problem modifications **will** occur at a later stage. Prepare for the unknown!

# Evaluation

| Lab session | Content | To deliver |
|---|---|---|
| 1 | Introduction to the problem | |
| 2 | | **Before week of 3rd lab:** Intermediary report + current codebase |
| 3 | Problem refinement | |
| 4 | Problem refinement | |
| 5 | Presentation | |
| Later | | Final report + codebase |

# Resources provided

- Today:
  - Toy instance + corresponding feasible solution
    - 1 crane, 1 carrier, 4 demands to fulfill
  - 2 Small instances
- *Ideally on* Friday (but perhaps on Monday 13/10)
  - Checker with visualizer
    - Called from the command line
    - Display instance
    - Check + display your solution over time
  - More instances to try

# Tips & tricks to get started

- First focus on parsing the input: How do you store information? Which data structures to use?
- Take it step by step, start with the toy instance (with only 1 carrier)
  1. Implement the parsing
  2. Fulfill the first transportation demand
  3. Fulfill the remaining demands
- Use the checker at each step to assess your code
- Once satisfied, go to a small instance (2+ carriers) and handle collisions
- **Do not** try to solve a whole (larger) instance at once