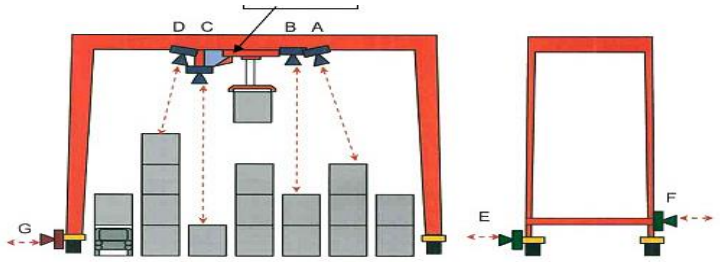
Transportation Research Board (TRB) Conference

Unloading and Pre-Marshalling Algorithms with Java Computer Animation for Terminal Yard Operations

January 10-14, 2016



OLD DOMINION UNIVERSITY

IDEA FUSION

Ivan Makohon
Graduate Student*
Modeling, Simulation & Visualization Engineering (MSVE) Department
Phone: 757-481-0832
imako001@odu.edu

Mecit Cetin (corresponding author)
Associate Professor*
Civil & Environmental Engineering (CEE) Department
Phone: 757-683-6700
mcetin@odu.edu

Manwo Ng
Assistant Professor*
Department of Information Technology and Decision Sciences
Phone: 757-683-6665
mng@odu.edu

Duc T. Nguyen
Professor*
CEE and MSVE Departments
Phone: 757-683-3761
dnguyen@odu.edu

*Old Dominion University

Abstract

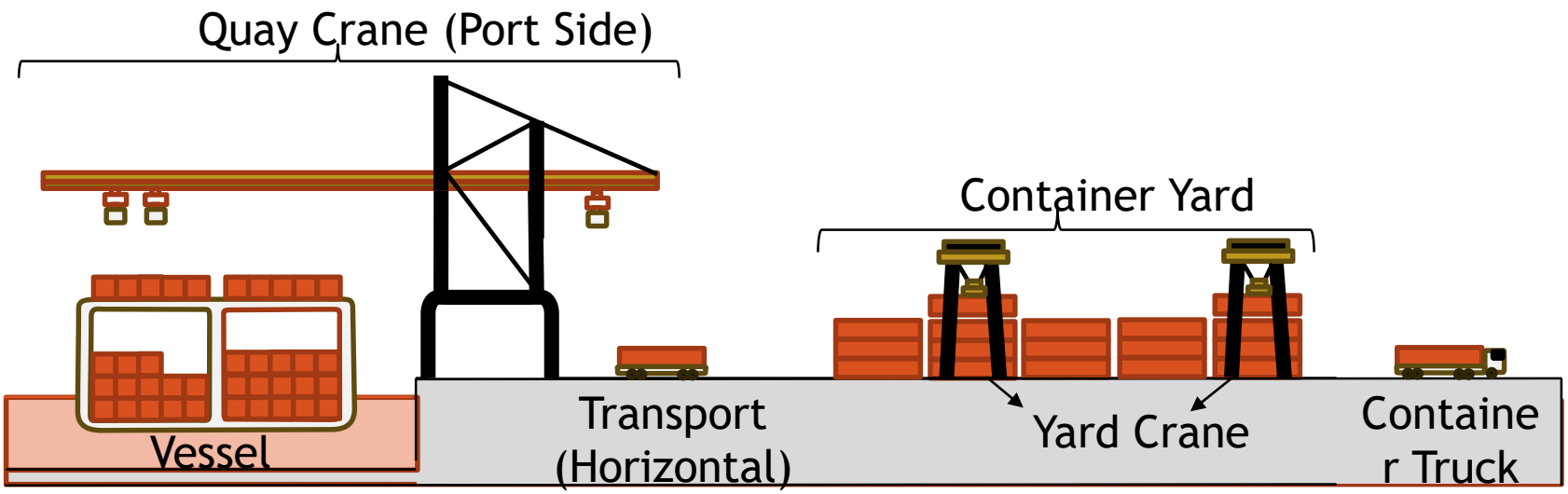
- Unloading and pre-marshalling (or reshuffling) operations have been considered as basic/important tasks in port terminals' activities, since the economic impacts can be significant. The primary goal for this paper is to develop simple heuristic unloading and pre-marshalling algorithms. Detailed explanations for the proposed algorithms are given in the form of step-by-step numerical procedures, so that the reported results can be reproduced by the readers. Several numerical examples (including the ones considered by other researchers) are used to validate the proposed unloading and pre-marshalling algorithms.

- A secondary goal for this paper is to develop a user-friendly and easy-to-use Java Graphical User Interface (GUI) software application that provides precise and clear step-by-step instructions with visual animation and voice for the unloading and pre-marshalling algorithms for Terminal Container Yard operations. The developed Java software can be used as an additional/educational tool that would serve both academia students and transportation professionals in solving any complex dynamic Terminal Container Yard layout visually with provided steps to show the "final/optimal" results. **A demo video of the both the unloading and pre-marshalling algorithm's animation and result can be viewed online from any web browser using the website provided in reference [*4*].**

- *Keywords*: Terminal, Yard, Cargo, Container, Unloading, Pre-marshalling, Reshuffling, Algorithm, Java, Animation

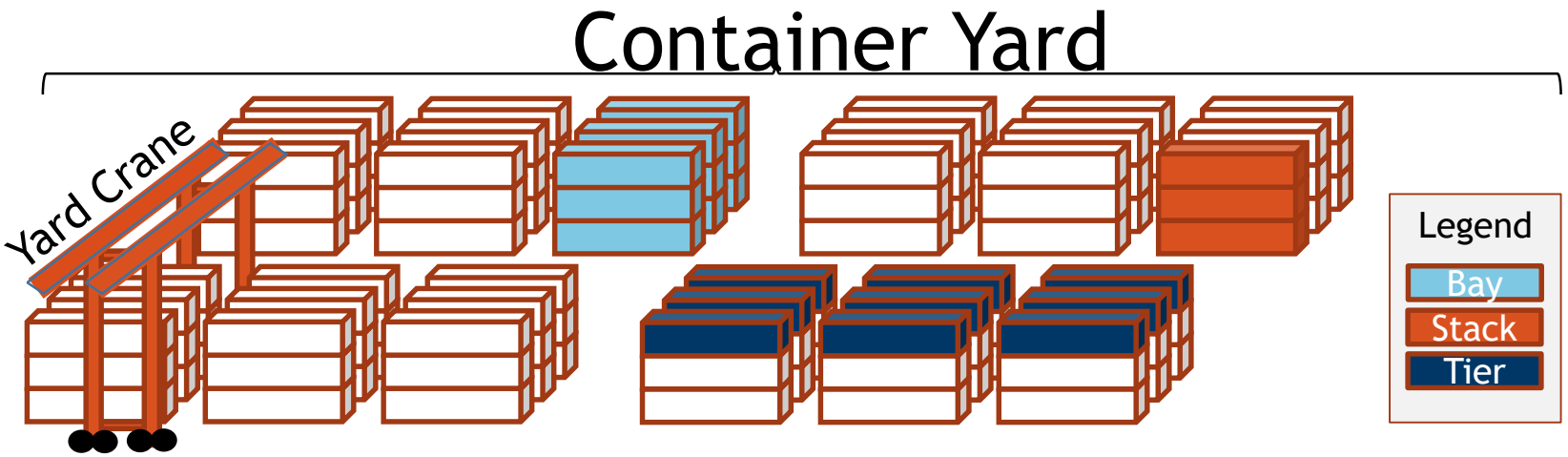
Overview (1 of 2)

- Container Terminal Yard Operations:
  - Quay Crane (Port-side) unload cargo containers from docked vessels pier side to horizontal transport trucks which transports them to the container yards.
  - Container Yards are storage yards for cargo containers.
  - Container trucks transport cargo containers to their destinations.



Overview (1 of 2)

- **Container Terminal Yard:**
  - **Crane** The yard crane that services bays.
  - **Bay** Contains a set of container stacks.
  - **Stack** Contains stackable cargo containers up to MNCPS.
  - **Tier** Height (level) of the stack.



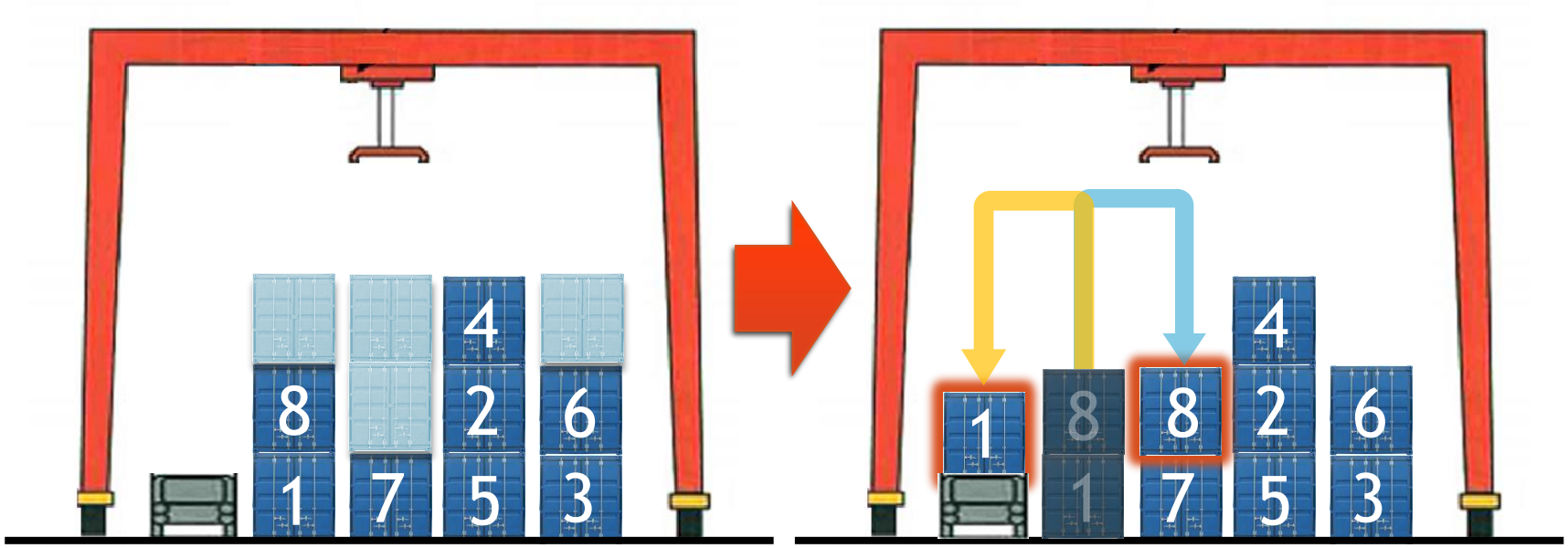
Introduction

- Container Terminal Yards are an import storage point since it links the seaside with landside transportation. It's an important storage point for loading/unloading cargo containers between vessels and vehicles (i.e. trucks or trains) for further distribution.
- Increasing traffic of cargo container shipments around the world which is why critical planning decision with great importance is needed for optimizing the terminal container yard's unloading and pre-marshalling (or reshuffling) for cargo containers within the container yard.
- Very similar to the Blocks World planning domain problem which consists of a finite number of blocks stacked into columns. The goal is to turn the initial state into a goal state with the minimal number of moves. This would save time in port operations which would lead to significant economic gains.
- Two algorithms (unloading and pre-marshalling) are revisited with the primary goal to demonstrate simple heuristic algorithm for port operations to visually showing step-by-step animation to solve for the unloading and pre-marshalling problem.

Unloading Algorithm

- The Unloading Algorithm approach is based upon having the container stacks reshuffled to get to the priority cargo container when a vehicle arrives.
  - When a vehicle arrives for pick up, the container stack is then reshuffled to get to the priority cargo container so that it can be unloaded onto the vehicle.
  - The container stacks remain the same until the next vehicle arrives.

Unloading Algorithm (1 of 5)

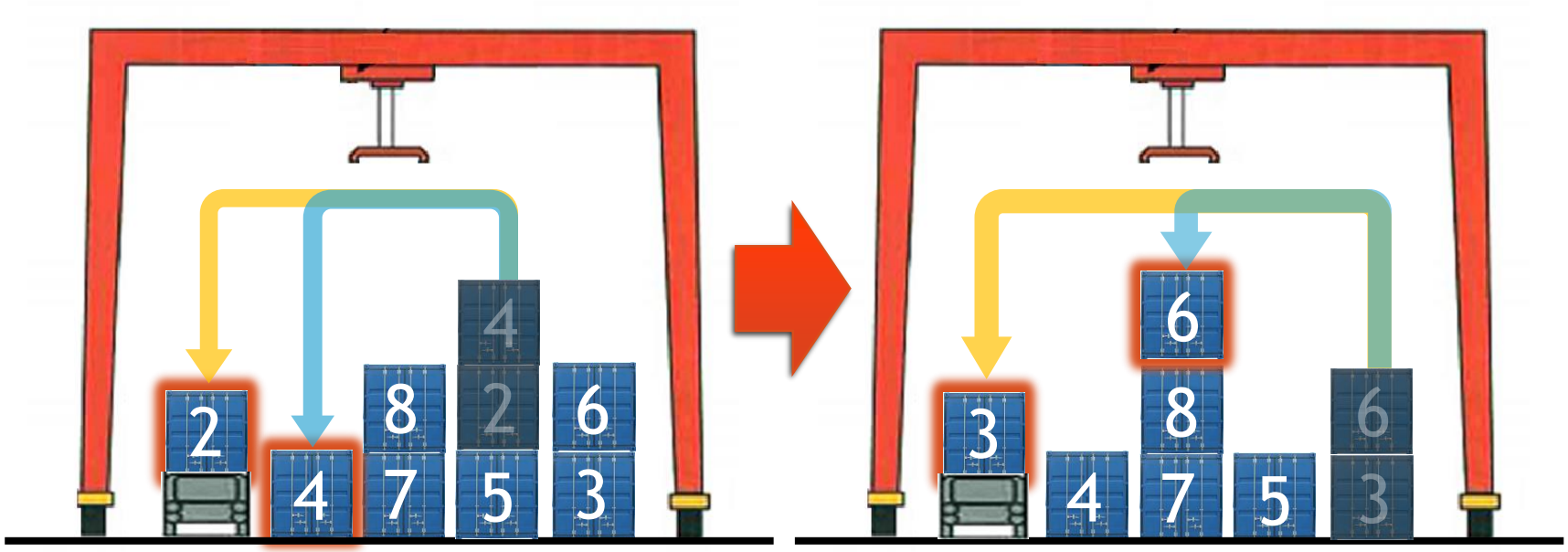


Initial Layout
NS = 4, MNCPS = 3
NS > MNCPS
NM = 0, NR = 0

1. Move 8 to Stack 2
2. Unload 1

NM = 1, NR = 1

Unloading Algorithm (2 of 5)



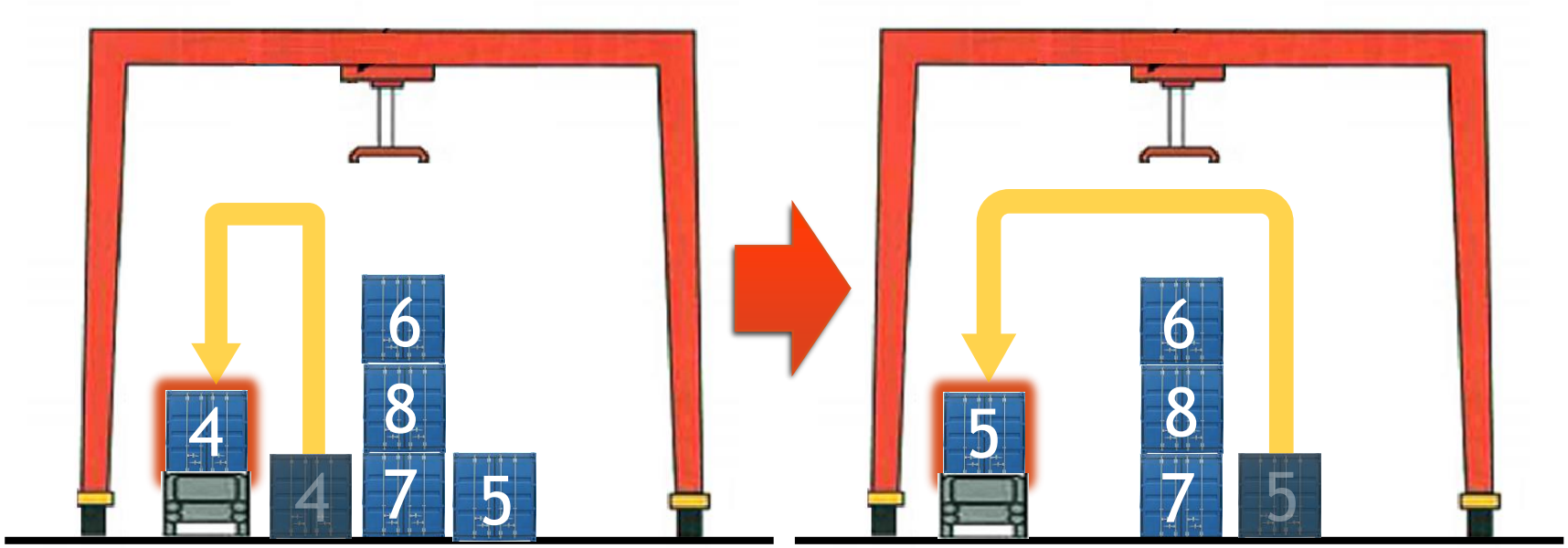
3. Move 4 to Stack 1
4. Unload 2

NM = 2, NR = 2

5. Move 6 to Stack 2
6. Unload 3

NM = 3, NR = 3

Unloading Algorithm (3 of 5)



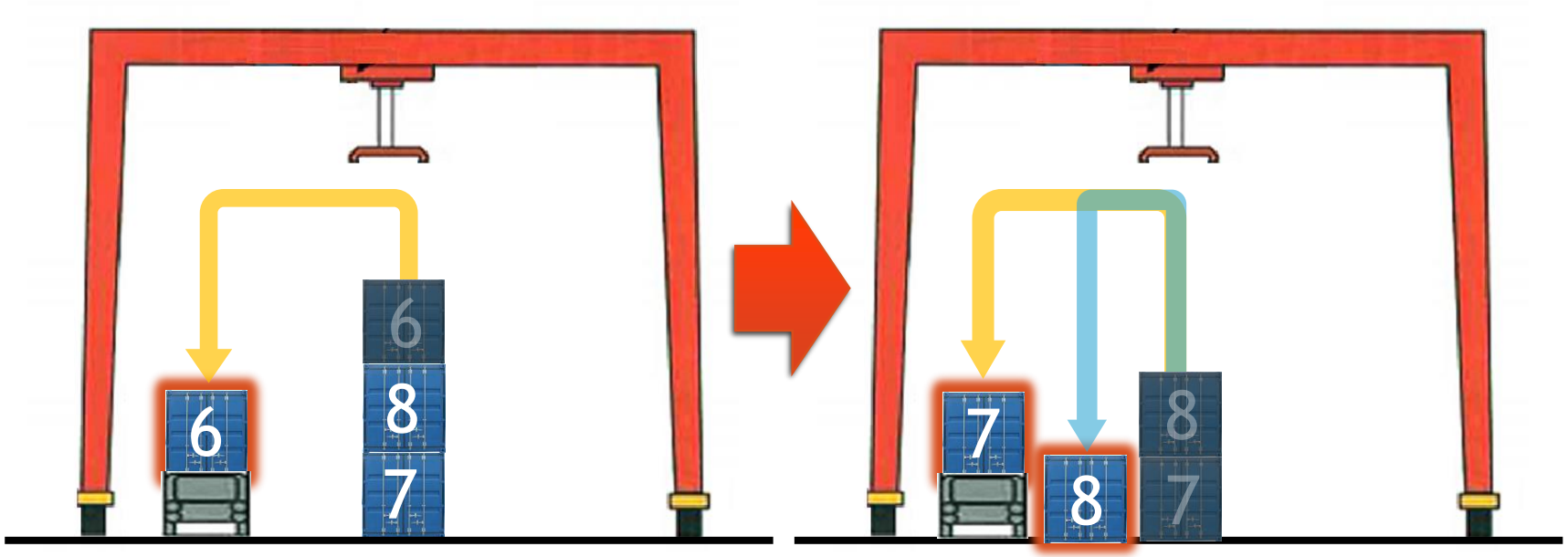
7. Unload 4

NM = 3, NR = 4

8. Unload 5

NM = 3, NR = 5

Unloading Algorithm (4 of 5)



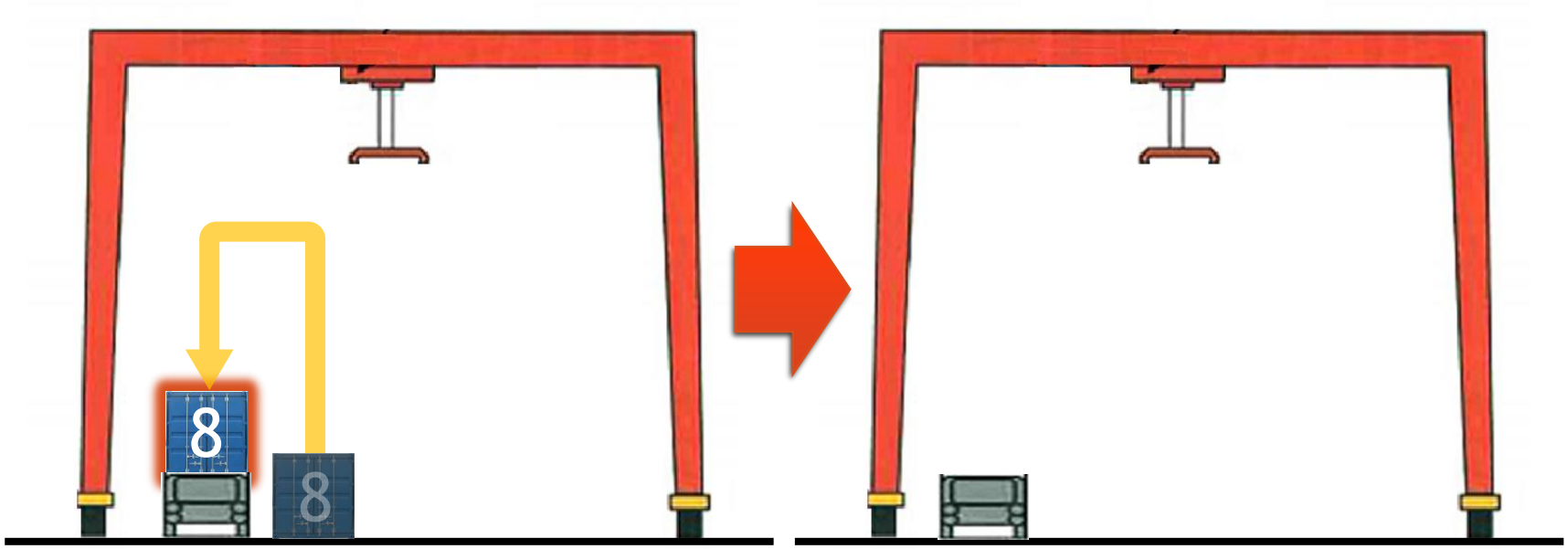
9. Unload 6

NM = 3, NR = 6

10. Move 8 to Stack 1
11. Unload 7

NM = 4, NR = 7

Unloading Algorithm (5 of 5)



12. Unload 8

NM = 4, NR = 8

13. Done Unloading

NM = 4, NR = 8

Unloading Algorithm (1 of 4)

- **Layout Validation**: Determine if there's enough empty spaces to perform the algorithm.
  - If the number of available empty spaces (NAES) is greater-than or equal to the number of empty spaces required (NESR) minus 1. The NESR-1 should allow us to get to the priority cargo container for removal.
  - If the number of container stack (NS) is greater-than 1 container stacks. Need at least 2 or more container stacks to be able to do unloading. NAES should provide enough empty spaces initially to get to the priority cargo container if it's on the bottom.

Unloading Algorithm (2 of 4)

- Step 0: Initialization
  - Initialize the Number of Stacks (NS)
  - Initialize the Maximum Number of Containers Per Stack (MNCPS)
  - Initialize the initial stack layout
  - Initialized variable for determining the number of spaces available
- Step 1: Verify the Number of available spaces required (do only once)
  - Simple check to determine if number available spaces is greater-than equal to the number of number of empty spaces required (in our case we use MNCPS)

Unloading Algorithm (3 of 4)

- Step 2: Find next cargo container to be removed
- Step 2.1: Determine if cargo container is on top of the stack
  - If stacks are empty
    - We are done.
  - Else if next cargo container to be removed is on top
    - Proceed to Step 4: Remove Container
  - Else if next cargo container not on top
    - Proceed to Step 3: Move Container
  - Proceed to Step 2: Find next cargo container to be removed

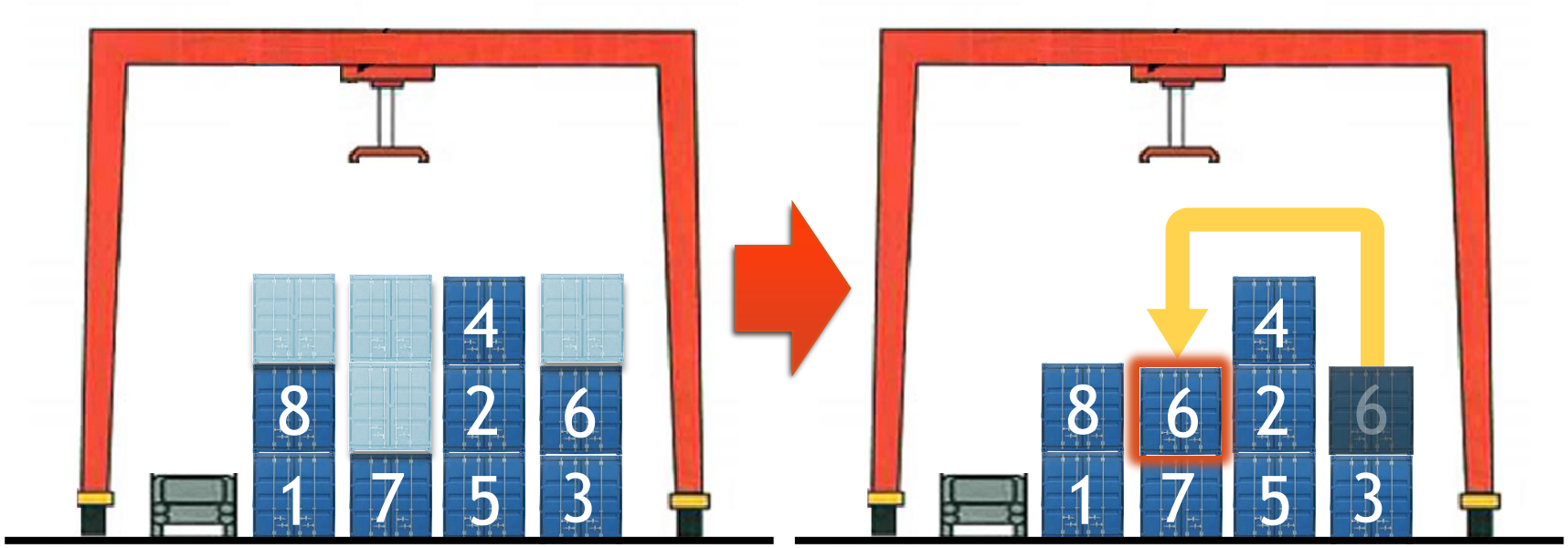
Unloading Algorithm (4 of 4)

- Step 3: Move Container
  - Determine the stack with the highest-lowest cargo container
  - Move cargo container to that stack
- Step 4: Remove Container
  - Remove (pop) the cargo container from the stack

Pre-Marshalling (Reshuffling) Algorithm

- The pre-marshalling (reshuffling) algorithm approach is based on having the container stacks ordered before the first vehicle arrives so that the priority container is on top.
  - No more reshuffling is required since the container stacks are reshuffled with the priority containers on top prior to the first vehicle arriving.
  - As each vehicle arrives, the priority container is on top and is unloading onto the vehicle.

Pre-Marshalling Algorithm (1 of 3)

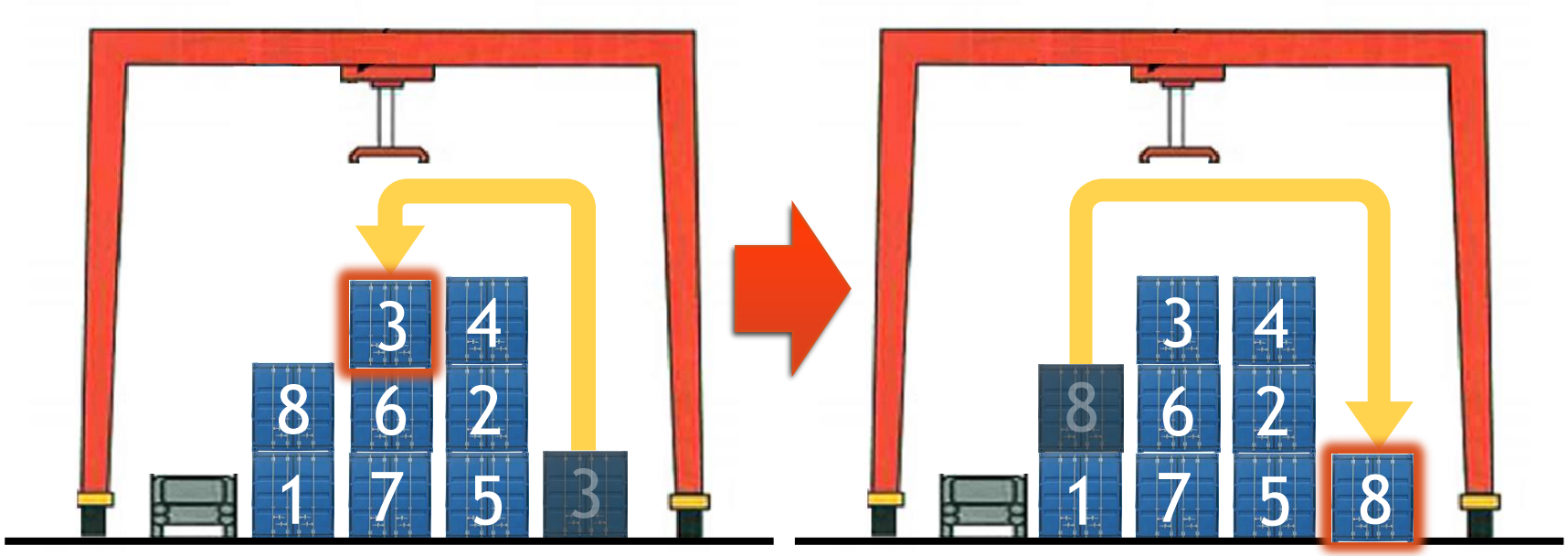


Initial Layout
NS = 4; MNCPS = 3
NS > MNCPS
NM = 0, NR = 0

1. Move 6 to Stack 2

NM = 1, NR = 0

Pre-Marshalling Algorithm (2 of 3)



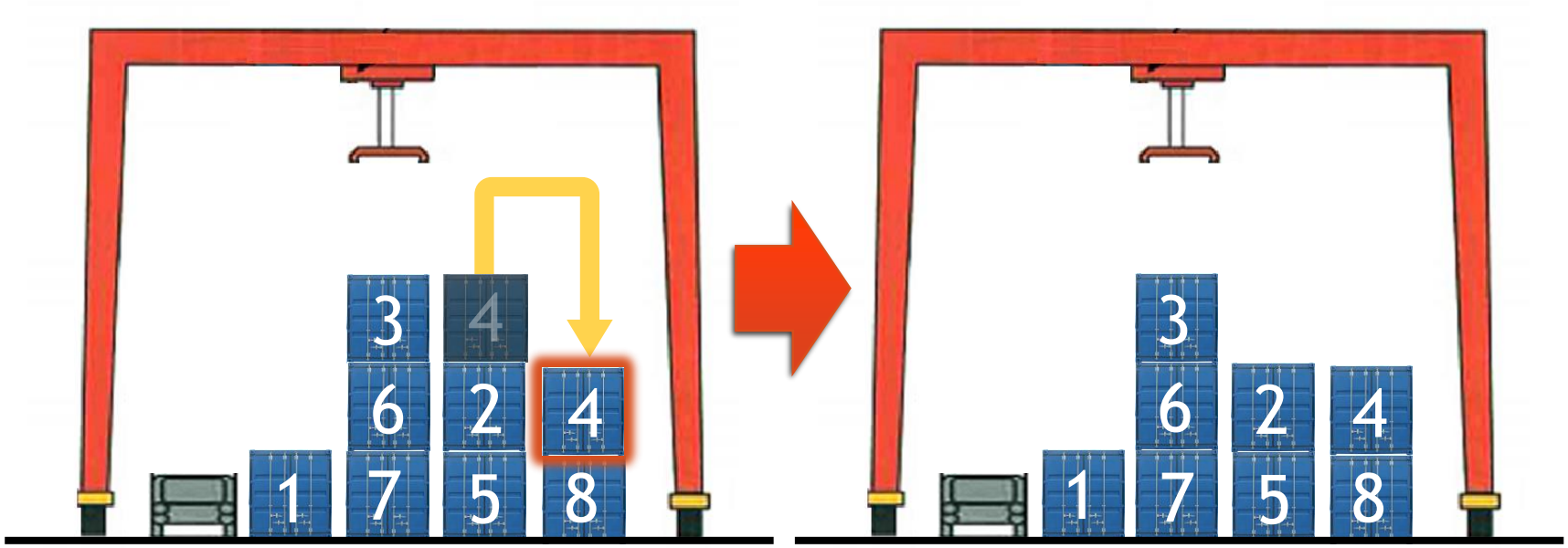
2. Move 3 to Stack 2

NM = 2, NR = 0

3. Move 8 to Stack 4

NM = 3, NR = 0

Pre-Marshalling Algorithm (3 of 3)



4. Move 4 to Stack 4

NM = 4, NR = 0

5. Done Reshuffling
6. Begin Unloading

NM = 4, NR = 0 → 8

Pre-Marshalling Algorithm (1 of 4)

- **Layout Validation**: Determine if there's enough empty spaces to perform the algorithm.
  - If the number of available empty spaces (NAES) is greater-than or equal to the number of empty spaces required (NESR). Need NAES in order to do reshuffling without removal.
  - If the number of container stacks (NS) is greater-than 2 container stacks. Need at least 3 or more container stacks to be able to do reshuffling. Note: some layouts can be solved without these validations for reshuffling, though these verifications avoid deadlock cases.

Pre-Marshalling Algorithm (2 of 4)

- Step 0: Initialization
  - Initialize the Number of Stacks (NS)
  - Initialize the Maximum Number of Containers Per Stack (MNCPS)
  - Initialize the initial stack layout
  - Initialized variable for determining the number of spaces available
- Step 1: Verify the Number of available spaces required (do only once)
  - Simple check to determine if number available spaces is greater-than equal to the number of number of empty spaces required (in our case we use MNCPS)

Pre-Marshalling Algorithm (3 of 4)

- Step 2: Reshuffle Stacks
  - Determine if all the stacks are in order, if so proceed to Step 4: Removing Containers From Stacks.
  - Calculate costs for all stacks (determine the number of moves)
  - Find container with the lowest cost value (number of moves).
  - If cost value is greater-than 0.0
    - Perform Step 2.1
  - Else if cost is equal-to 0.0
    - Perform Step 3: Move Containers (info is provided from Calculating Costs for all stacks)
  - Perform Step 2: Reshuffle Stacks (Recursion)
- Step 2.1: Determine if there's a Stack with One container that can be moved
  - If so then try to find a stack that's in order with a container with the highest-lowest container that this container being moved. Also, check stack to determine if it's less-than MNCPS.

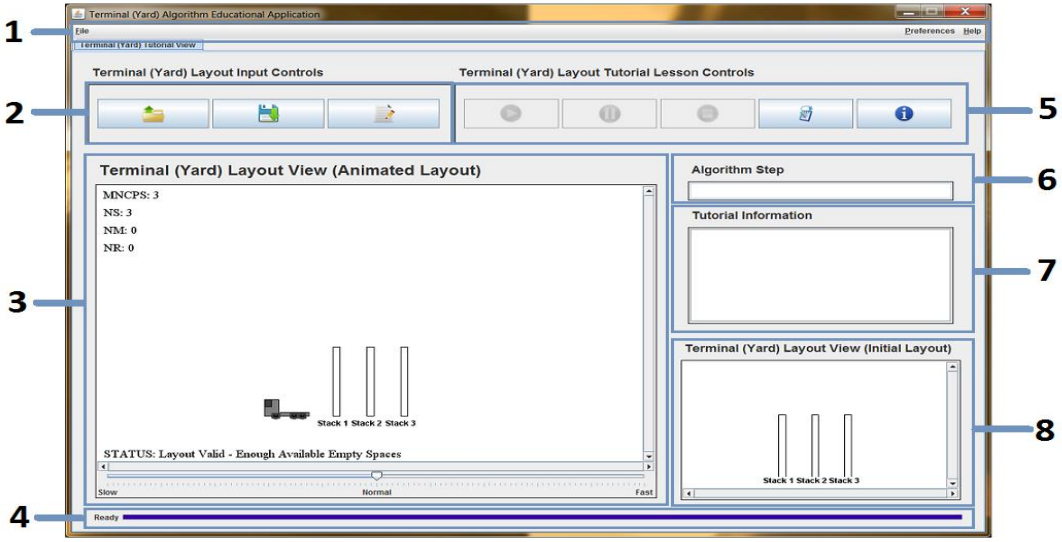
Pre-Marshalling Algorithm (4 of 4)

- Step 3: Move Container
  - Move cargo container to container stack. Basically, removes (pops) the cargo container from the existing (source) stack and adds (pushes) it to the new (destination) stack.
- Step 4: Removing Containers From Stacks
  - Stack should be in order (lowest (on top) and highest (on bottom)).
  - Loop through container stack and find the next lowest cargo container to be removed.

Java Computer Animation (1 of 3)

- A user-friendly and easy-to-use Java Graphical User Interface (GUI) software application that provides precise and clear step-by-step instructions with visual animation and voice for the unloading and pre-marshalling algorithms for Terminal Container Yard operations.
- The software can be used as an additional/educational tool that would serve both academia students and transportation professionals in solving any complex dynamic Terminal Container Yard layout visually with provided steps to show the “final/optimal” results.

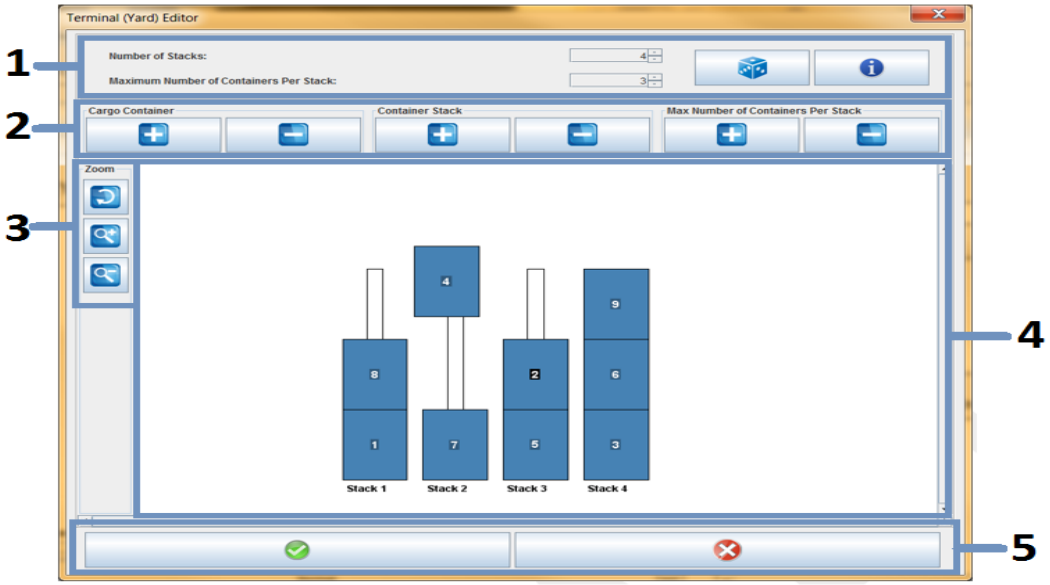
Java Computer Animation (2 of 3)



MAIN Graphical User Interface (GUI):

1. Menu Buttons (File, Preferences, and Help)
2. Input Control Buttons
3. Terminal Yard Layout (Animation View)
4. Status View
5. Lesson Control Buttons
6. Algorithm Step
7. Tutorial Information View
8. Terminal Yard Layout (Initial View)

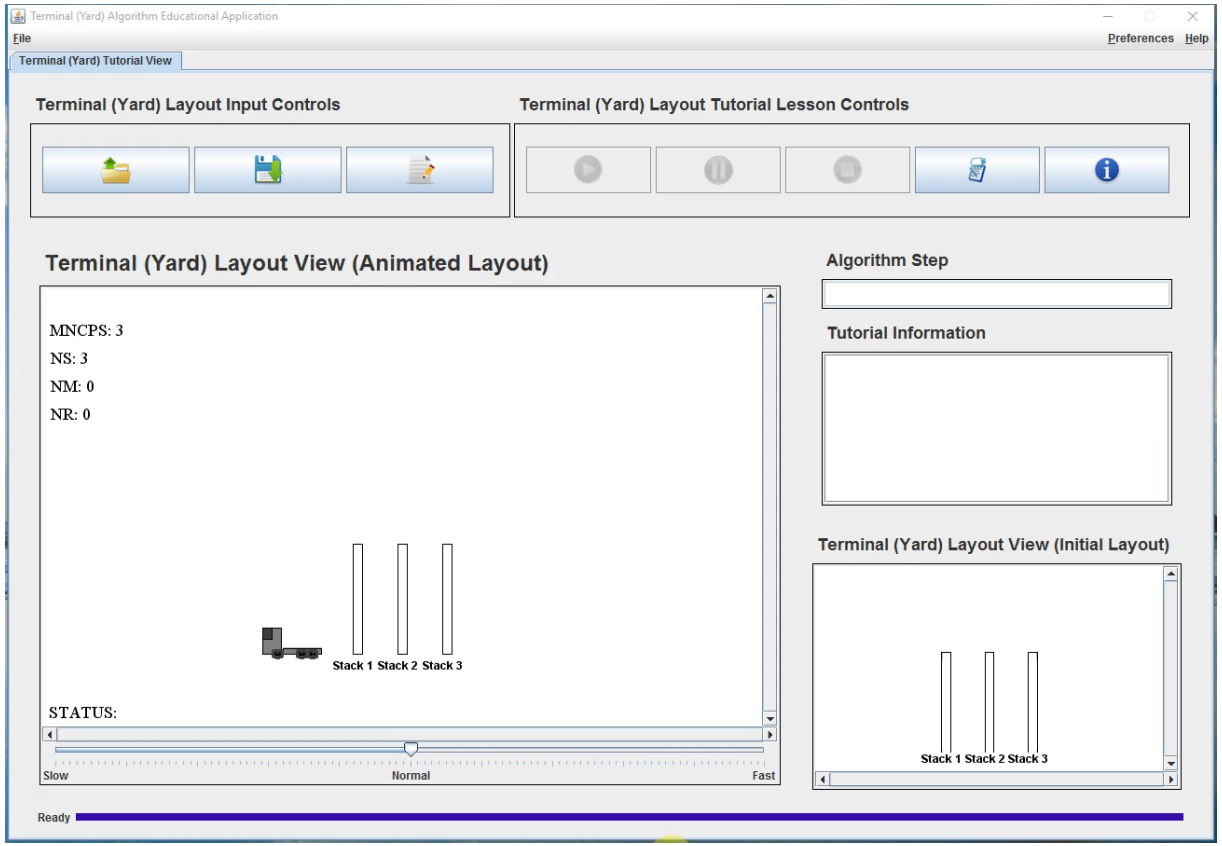
Java Computer Animation (3 of 3)



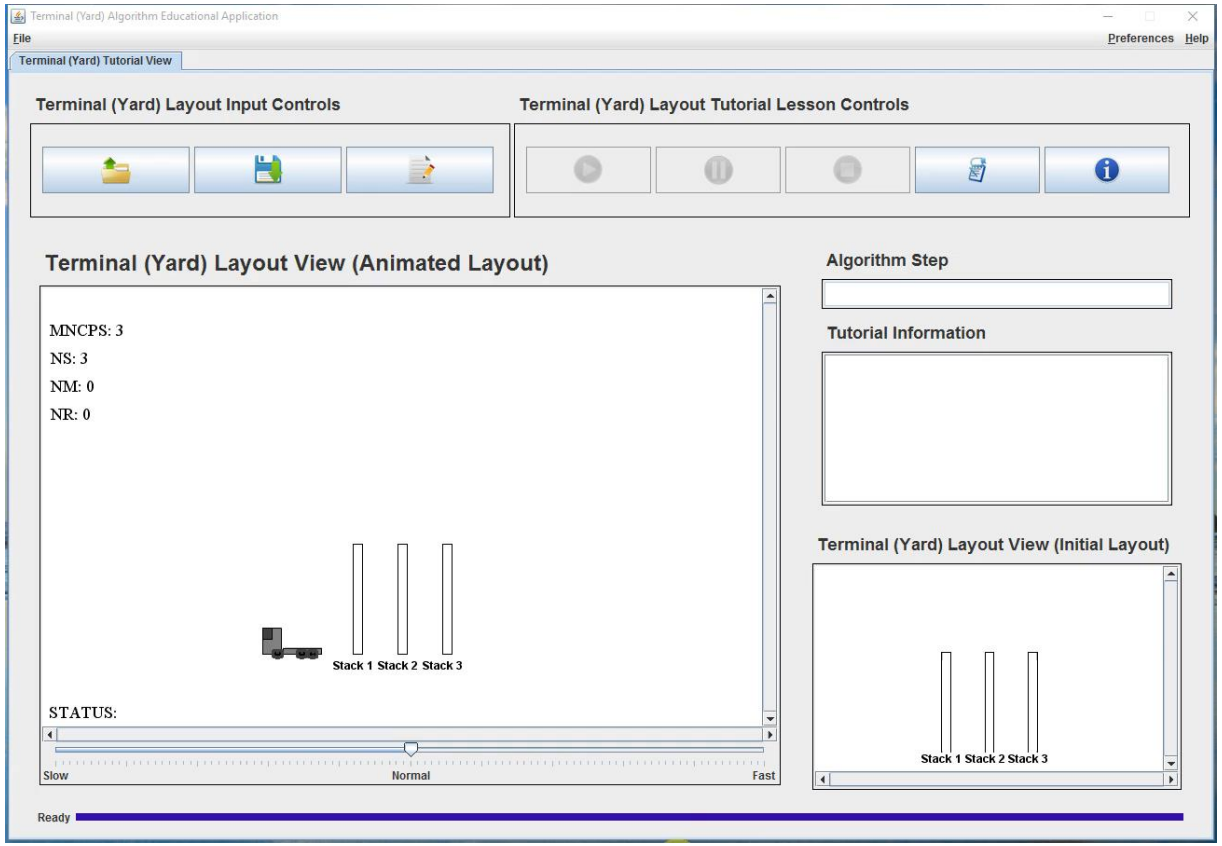
Terminal Yard Editor GUI:

1. Display & Information View
2. Editor Buttons
3. Zoom Buttons
4. Visualization Editor View
5. Confirmation/Close Button

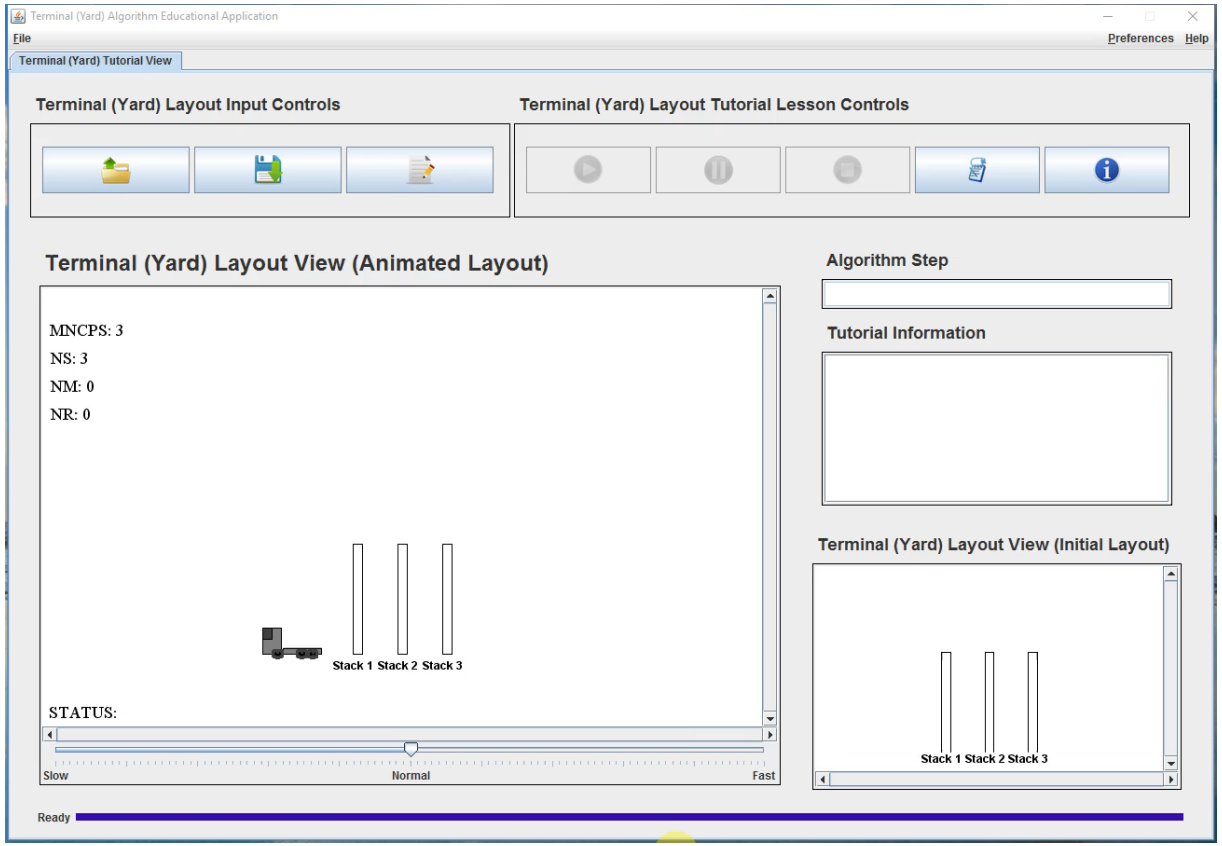
Unloading Algorithm Demo



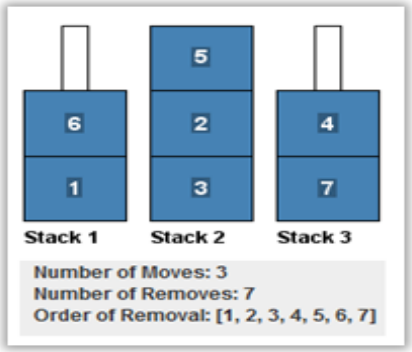
Pre-Marshalling Algorithm Demo

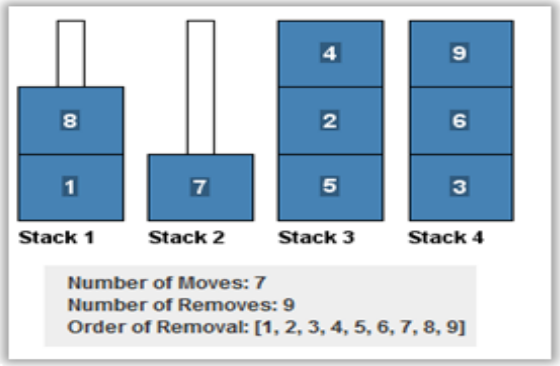


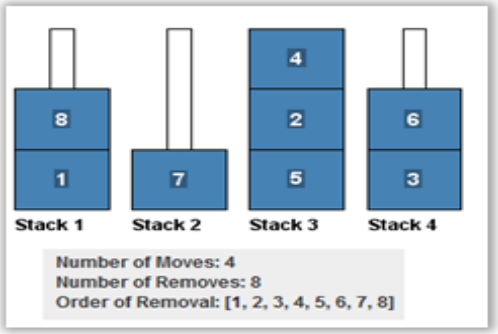
Terminal Yard Editor Demo

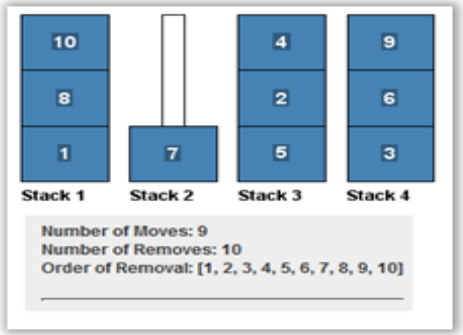


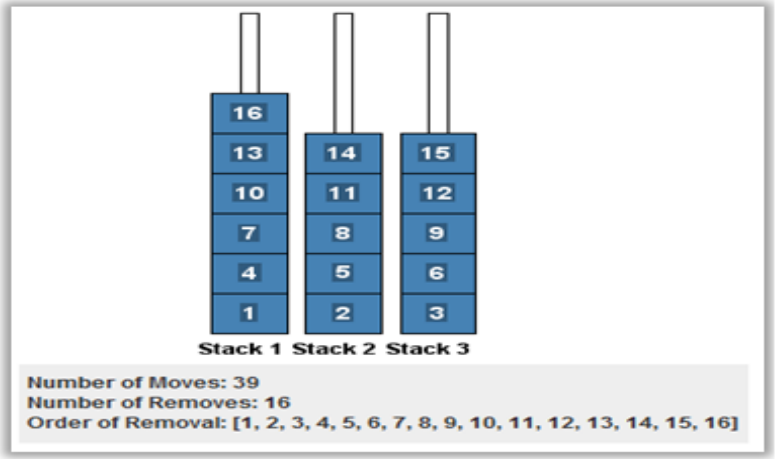
Experimental Results

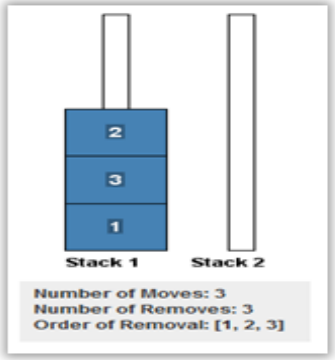












Questions/Comments

Acronyms

- **NS** Number of Stacks
- **MNCPS** Maximum Number of Containers Per Stack
- **NAES** Number of Available Empty Spaces (a.k.a. NAES = TNS – NCC)
- **NESR** Number of Empty Spaces Required (a.k.a. NESR = MNCPS)
- **NCC** Number of Cargo Containers
- **TNS** Total Number of Spaces (a.k.a. TNS = M x N)
- **NM** Number of Moves
- **NR** Number of Removes
- **M** The number of Rows (a.k.a. MNCPS)
- **N** The number of Columns (a.k.a. NS)

Extra Slides

Additional Slides

Pre-Marshalling Algorithm

Pre-Marshalling (Reshuffling) (Examples)

HW Set #1 – Problem 6 Special Team's Project

- Referring to the instructor's (26 page) notes on "Port Container Terminals", and based on the explanation/discussion and illustrated example shown on page 17-26 (Pre-marshalling, or Reshuffling algorithm):
  - Write a "General" step-by-step computer program (any computer language) for the reshuffling operations.
  - Solve for the initial storage layout:
    - NS = 4
    - MNCPS = b_max = 3
    - NS > MNCPS

		4	
8		2	6
1	7	5	3

  - Solve for the initial storage layout:
    - NS = 4
    - MNCPS = b_max = 3
    - NS > MNCPS

		4	9
8		2	6
1	7	5	3

NS = Number of Stacks

MNCPS = Maximum Number of Containers Per Stack

HW Set #1 – Problem 6 (h) - Solution

- Solve for the initial storage layout:
  - NS = 4
  - MNCPS = b_max = 3
  - NS > MNCPS

		4	
8		2	6
1	7	5	3

- **Iteration # 1**
  - Moved Container 6 to Stack 2

*	*	4	*
8	6	2	*
1	7	5	3

*	3	4	*
8	6	2	*
1	7	5	*

- **Iteration # 2**
  - Moved Container 3 to Stack 2

HW Set #1 – Problem 6 (h) - Solution

- **Iteration # 3**
  - Moved Container 8 to Stack 4

*	3	4	*
*	6	2	*
1	7	5	8

*	3	*	*
*	6	2	4
1	7	5	8

- **Iteration # 4**
  - Moved Container 4 to Stack 4

- **Iteration # 5**
  - Removed Container 1

*	3	*	*
*	6	2	4
*	7	5	8

*	3	*	*
*	6	*	4
*	7	5	8

- **Iteration # 6**
  - Removed Container 2

* = Denotes empty space

HW Set #1 – Problem 6 (h) - Solution

- **Iteration # 7**
  - Removed Container 3

*	*	*	*
*	6	*	4
*	7	5	8

*	*	*	*
*	6	*	*
*	7	5	8

- **Iteration # 8**
  - Removed Container 4

- **Iteration # 9**
  - Removed Container 5

*	*	*	*
*	6	*	*
*	7	*	8

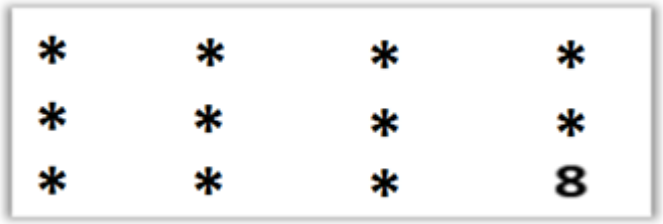
*	*	*	*
*	*	*	*
*	7	*	8

- **Iteration # 10**
  - Removed Container 6

* = Denotes empty space

HW Set #1 – Problem 6 (h) - Solution

- **Iteration # 11**
  - Removed Container 7



- **Iteration # 12**
  - Removed Container 8

* = Denotes empty space

HW Set #1 – Problem 6 (h) - Solution

- Solve for the initial storage layout:
  - NS = 4
  - MNCPS = b_max = 3
  - NS > MNCPS

		4	9
8		2	6
1	7	5	3

- **Iteration # 1**
  - Moved Container 4 to Stack 2

*	*	*	9
8	4	2	6
1	7	5	3

- **Iteration # 2**
  - Moved Container 8 to Stack 3

*	*	8	9
*	4	2	6
1	7	5	3

HW Set #1 – Problem 6 (h) - Solution

- **Iteration # 3**
  - Moved Container 1 to Stack 2

*	1	8	9
*	4	2	6
*	7	5	3

*	1	*	9
*	4	2	6
8	7	5	3

- **Iteration # 4**
  - Moved Container 8 to Stack 1

- **Iteration # 5**
  - Moved Container 1 to Stack 1

*	*	*	9
1	4	2	6
8	7	5	3

9	*	*	*
1	4	2	6
8	7	5	3

- **Iteration # 6**
  - Moved Container 9 to Stack 1

* = Denotes empty space

HW Set #1 – Problem 6 (h) - Solution

- **Iteration # 7**
  - Moved Container 6 to Stack 2

9	**6**	*	*
1	4	2	*
8	7	5	3

9	6	**3**	*
1	4	2	*
8	7	5	*

- **Iteration # 8**
  - Moved Container 3 to Stack 3

- **Iteration # 9**
  - Moved Container 9 to Stack 4

*	6	3	*
1	4	2	*
8	7	5	**9**

*	*	3	*
1	4	2	**6**
8	7	5	9

- **Iteration # 10**
  - Moved Container 6 to Stack 4

* = Denotes empty space

HW Set #1 – Problem 6 (h) - Solution

- **Iteration # 11**
  - Moved Container 3 to Stack 4

*	*	*	3
1	4	2	6
8	7	5	9

- **Iteration # 12**
  - Removed Container 1

*	*	*	3
*	4	2	6
8	7	5	9

- **Iteration # 13**
  - Removed Container 2

*	*	*	3
*	4	*	6
8	7	5	9

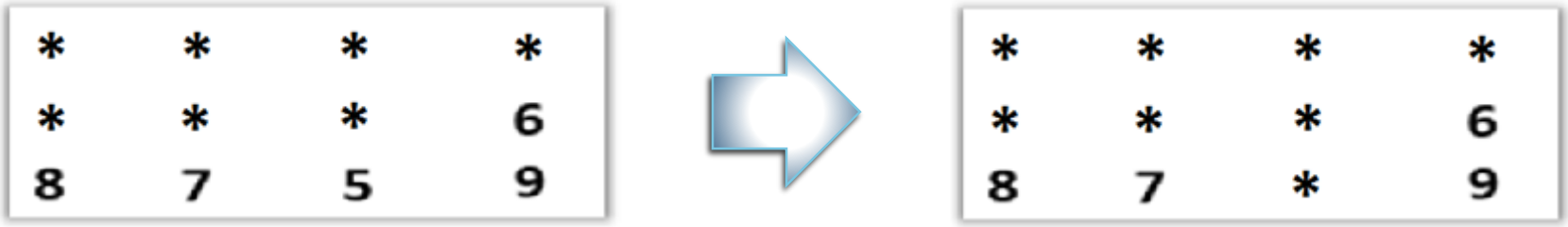
- **Iteration # 14**
  - Removed Container 3

*	*	*	*
*	4	*	6
8	7	5	9

* = Denotes empty space

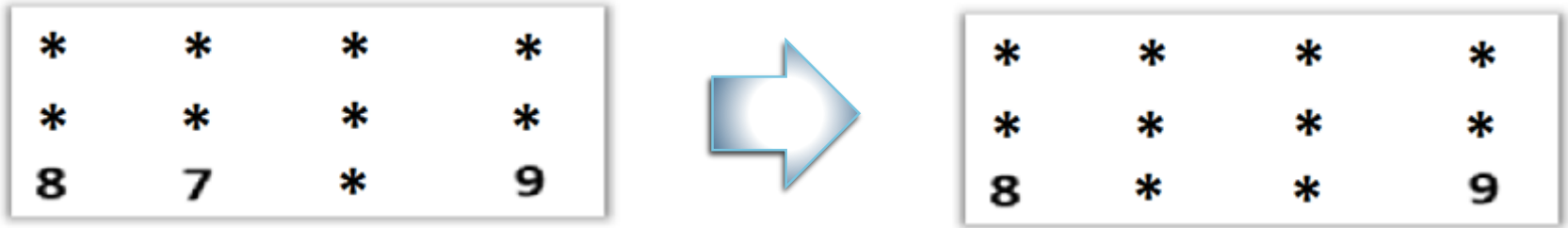
HW Set #1 – Problem 6 (h) - Solution

- **Iteration # 15**
  - Removed Container 4



- **Iteration # 16**
  - Removed Container 5

- **Iteration # 17**
  - Removed Container 6



- **Iteration # 18**
  - Removed Container 7

* = Denotes empty space

HW Set #1 – Problem 6 (h) - Solution

- **Iteration # 19**
  - Removed Container 8

*	*	*	*
*	*	*	*
*	*	*	9

- **Iteration # 20**
  - Removed Container 9

*	*	*	*
*	*	*	*
*	*	*	*

* = Denotes empty space

Results

- Solve for the initial storage layout
  - NS = 4
  - MNCPS = b_max = 3
  - NS > MNCPS

- Results:
  - Number of Moves: 4
  - Number of Removes: 8
  - Remove [Order]: 1, 2, 3, 4, 5, 6, 7, 8
  - Start Time: 2015.04.18.17.022.14
  - End Time: 2015.04.18.17.022.14

- Solve for the initial storage layout
  - NS = 4
  - MNCPS = b_max = 3
  - NS > MNCPS

- Results:
  - Number of Moves: 11
  - Number of Removes: 9
  - Remove [Order]: 1, 2, 3, 4, 5, 6, 7, 8, 9
  - Start Time: 2015.04.18.17.023.12
  - End Time: 2015.04.18.17.023.12

Unloading Algorithm

Unloading
(Examples)

HW Set #1 – Problem 5 Special Team's Project

- Referring to the instructor's (26 page) notes on "Port Container Terminals", and based on the explanation/discussion and illustrated example shown on page 13 (see Figure 11: Unloading Operations, and based on your results in Problem 4a,b,c):
  - Write a "General" step-by-step computer program (any computer language) for the unloading operations.
  - Solve for the initial storage layout
    - NS = 4
    - MNCPS = b_max = 3
    - NS > MNCPS

	2		
8	7	4	
1	5	3	6

  - Solve for the initial storage layout
    - NS = 3
    - MNCPS = b_max = 4
    - NS < MNCPS

	2	
6	7	4
1	5	3

NS = Number of Stacks

MNCPS = Maximum Number of Containers Per Stack

HW Set #1 – Problem 6 (e) - Solution

- Solve for the initial storage layout
  - NS = 4
  - MNCPS = b_max = 3
  - NS > MNCPS

	2		
8	7	4	
1	5	3	6

- **Iteration # 1**
  - GOAL: Remove Container 1
  - Moved Container 8 to Stack 4

*	2	*	*
*	7	4	8
1	5	3	6

*	2	*	*
*	7	4	8
*	5	3	6

- **Iteration # 2**
  - Removed Container 1

* = Denotes empty space

HW Set #1 – Problem 6 (e) - Solution

- **Iteration # 3**
  - Removed Container 2

*	*	*	*
*	7	4	8
*	5	3	6

*	*	*	*
*	7	*	8
4	5	3	6

- **Iteration # 4**
  - GOAL: Remove Container 3
  - Moved Container 4 to Stack 1

- **Iteration # 5**
  - Removed Container 3

*	*	*	*
*	7	*	8
4	5	*	6

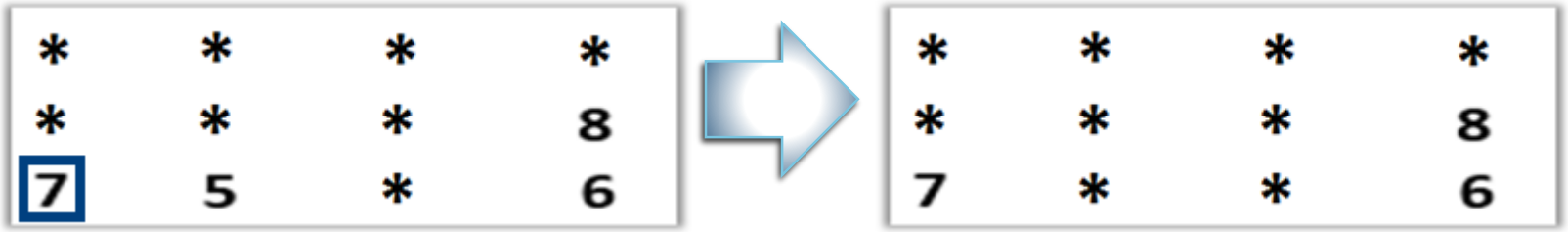
*	*	*	*
*	7	*	8
*	5	*	6

- **Iteration # 6**
  - Removed Container 4

* = Denotes empty space

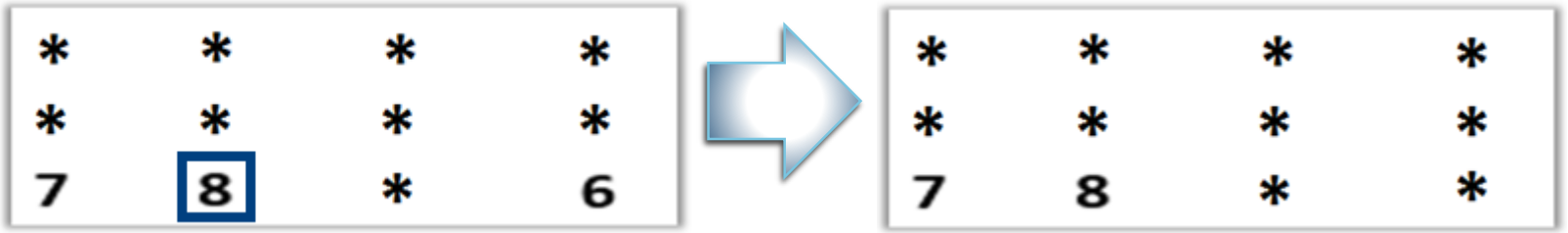
HW Set #1 – Problem 6 (e) - Solution

- **Iteration # 7**
  - GOAL: Remove Container 5
  - Moved Container 7 to Stack 1



- **Iteration # 8**
  - Removed Container 5

- **Iteration # 9**
  - GOAL: Remove Container 6
  - Moved Container 8 to Stack 2

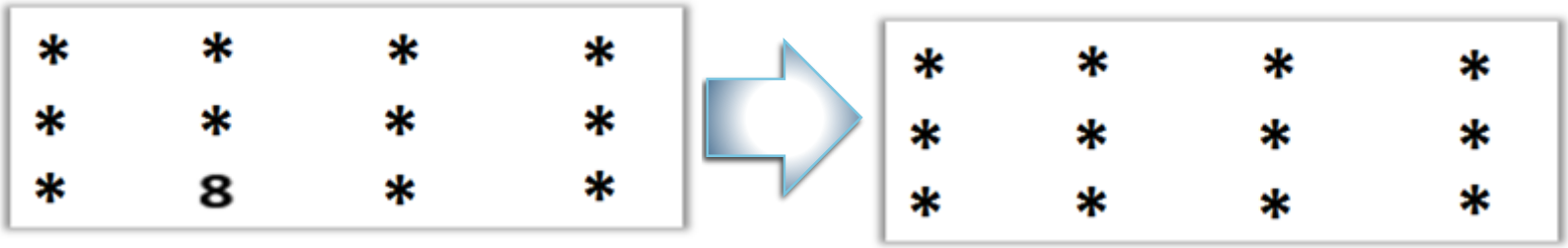


- **Iteration # 10**
  - Removed Container 6

* = Denotes empty space

HW Set #1 – Problem 6 (e) - Solution

- **Iteration # 11**
  - Removed Container 7



- **Iteration # 12**
  - Removed Container 8

* = Denotes empty space

HW Set #1 – Problem 6 (f) - Solution

- Solve for the initial storage layout
  - NS = 3
  - MNCPS = b_max = 4
  - NS < MNCPS

	2	
6	7	4
1	5	3

- **Iteration # 1**
  - GOAL: Remove Container 1
  - Moved Container 6 to Stack 4

*	2	6
*	7	4
1	5	3

- **Iteration # 2**
  - Removed Container 1

*	2	6
*	7	4
*	5	3

* = Denotes empty space

HW Set #1 – Problem 6 (f) - Solution

- **Iteration # 3**
  - Removed Container 2

*	*	6
*	7	4
*	5	3

- **Iteration # 4**
  - GOAL: Remove Container 3
  - Moved Container 6 to Stack 1

*	*	*
*	7	4
6	5	3

- **Iteration # 5**
  - GOAL: Remove Container 3
  - Moved Container 4 to Stack 2

*	4	*
*	7	*
6	5	3

- **Iteration # 6**
  - Removed Container 3

*	4	*
*	7	*
6	5	*

HW Set #1 – Problem 6 (f) - Solution

- **Iteration # 7**
  - Removed Container 4

- **Iteration # 8**
  - GOAL: Remove Container 5
  - Moved Container 7 to Stack 3

- **Iteration # 9**
  - Removed Container 5

- **Iteration # 10**
  - Removed Container 6

HW Set #1 – Problem 6 (f) - Solution

- **Iteration # 11**
  - Removed Container 7

Results

- Solve for the initial storage layout
  - NS = 4
  - MNCPS = b_max = 3
  - NS > MNCPS

	2		
8	7	4	
1	5	3	6

- Results:
  - Number of Moves: 4
  - Number of Removes: 8
  - Remove [Order]: 1, 2, 3, 4, 5, 6, 7, 8
  - Start Time: 2015.04.18.15.022.43
  - End Time: 2015.04.18.15.022.43

- Solve for the initial storage layout
  - NS = 3
  - MNCPS = b_max = 4
  - NS < MNCPS

	2	
6	7	4
1	5	3

- Results:
  - Number of Moves: 4
  - Number of Removes: 7
  - Remove [Order]: 1, 2, 3, 4, 5, 6, 7
  - Start Time: 2015.04.18.15.027.13
  - End Time: 2015.04.18.15.027.13