Data Structure and Analysis
Offline Session 1

Il-Chul Moon
Department of Industrial and Systems Engineering
KAIST
icmoon@kaist.ac.kr

Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Session Schedule

- 1300-1415 : Summary lecture on
  - Python basics
  - Object oriented paradigm
  - Linked List, Stack and Queue
  - Recursions and Dynamic Programming

- 1430-1545 : Exercise on AI Blackjack programming

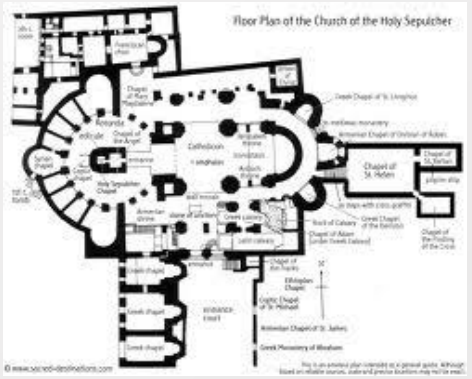
Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

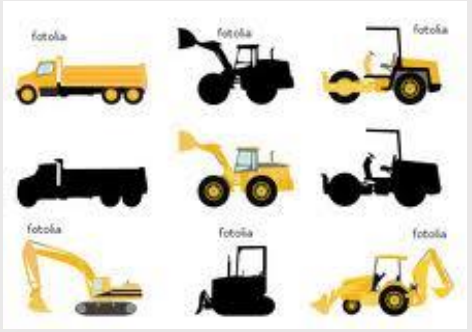
SUMMARY LECTURE

Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Programming and DS&A

- What is programming to data structure and algorithm?
  - Programming is an implementation tool
  - Conceptual thinking and design
    - Where to put the restroom
    - How to find the restroom
  - Practical design and implementation
    - What to use for designing the restroom
    - How to move to the restroom
- Both are important
  - Should pursue good design and good implementation
  - Good design and bad implementation?
  - Bad design and good implementation?





Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

List

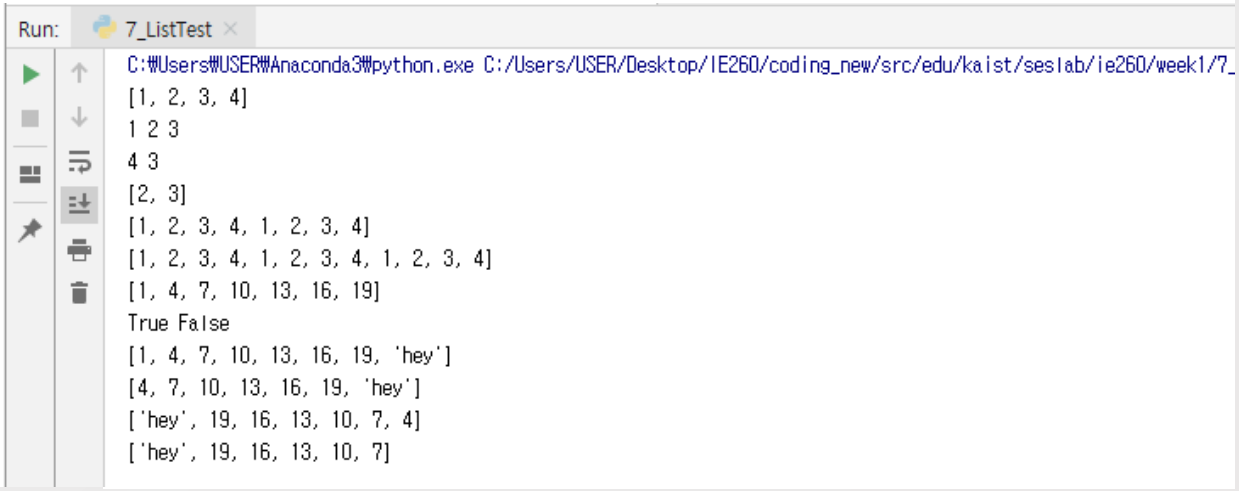
List is another type of sequence variables

```
lstTest = [1, 2, 3, 4]
print(lstTest)
print(lstTest[0], lstTest[1], lstTest[2])
print(lstTest[-1], lstTest[-2])
print(lstTest[1:3])
print(lstTest+lstTest)
print(lstTest*3)

lstTest = list(range(1, 20, 3))
print(lstTest)
print(4 in lstTest, 100 in lstTest)
lstTest.append('hey')
print(lstTest)
del lstTest[0]
print(lstTest)
lstTest.reverse()
print(lstTest)
lstTest.remove(4)
print(lstTest)
```

See how the operators work

range(x,y,z) == x:y:z
You will use this function many, many times

in and not in comes pretty handy



Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Dictionary

- Dictionary is also a collection variable type
  - However, it is not sequential
  - It works by a pair of keys and values
    - A set of (key 1, value 1), (key 2, value 2), (key 3, value 3)…
    - Exact syntax is { key1:value1, key2:value2, key3:value3 …)

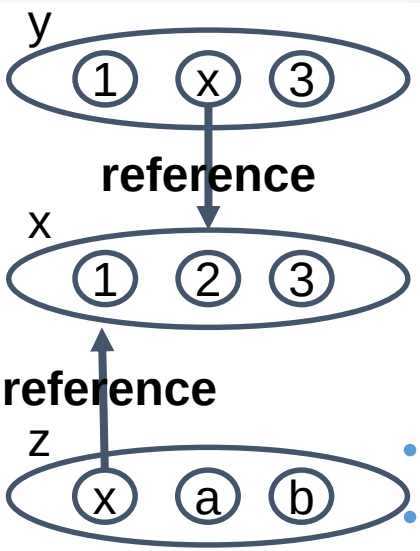
```
dicTest = {1: 'one', 2: 'two', 3: 'three'}
print(dicTest[1])
dicTest[4] = 'four'
print(dicTest)
dicTest[1] = 'hana'
print(dicTest)
print(dicTest.keys())
print(dicTest.values())
print(dicTest.items())
```

```
Run:  9_DictionaryTest
C:₩Users₩USER₩Anaconda3₩python.exe C:/Users/USER/Desktop/IE260/coding_n
one
{1: 'one', 2: 'two', 3: 'three', 4: 'four'}
{1: 'hana', 2: 'two', 3: 'three', 4: 'four'}
dict_keys([1, 2, 3, 4])
dict_values(['hana', 'two', 'three', 'four'])
dict_items([(1, 'hana'), (2, 'two'), (3, 'three'), (4, 'four')])
```

Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Assignment and Equivalence

```
x = [1, 2, 3]
y = [100, x, 120]
z = [x, 'a', 'b']

print('x : ', x)
print('y : ', y)
print('z : ', z)

x[1] = 1717

print('₩nx : ', x)
print('y : ', y)
print('z : ', z)

x[1] = 2
x2 = [1, 2, 3]
if x == x2:
    print("Values are equivalent")
else:
    print("Values are not equivalent")

if x is x2:
    print("Values are stored at the same place")
else:
    print("Values are not stored at the same place")

if x[1] is y[1][1]:
    print("Values are stored at the same place")
else:
    print("Values are not stored at the same place")
```




```
Run:  15_AssignmentAndEquivalence
C:₩Users₩USER₩Anaconda3₩python.exe C:/Users/USER/Desktop/IE260/coding_new/src/edu/kaist/seslab
x :  [1, 2, 3]
y :  [100, [1, 2, 3], 120]
z :  [[1, 2, 3], 'a', 'b']

x :  [1, 1717, 3]
y :  [100, [1, 1717, 3], 120]
z :  [[1, 1717, 3], 'a', 'b']
Values are equivalent
Values are not stored at the same place
Values are stored at the same place
```

- One variable's value is changed
- But, you see three changes
- Why this happened?
  - Because of references
  - x has two references from y and z
  - The values of y and z are determined by x, and x is changed
    - See the ripple effects
- **==**
  - Checks the equivalence of two referenced values
- **is**
  - Checks the equivalence of two referenced objects' IDs

Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

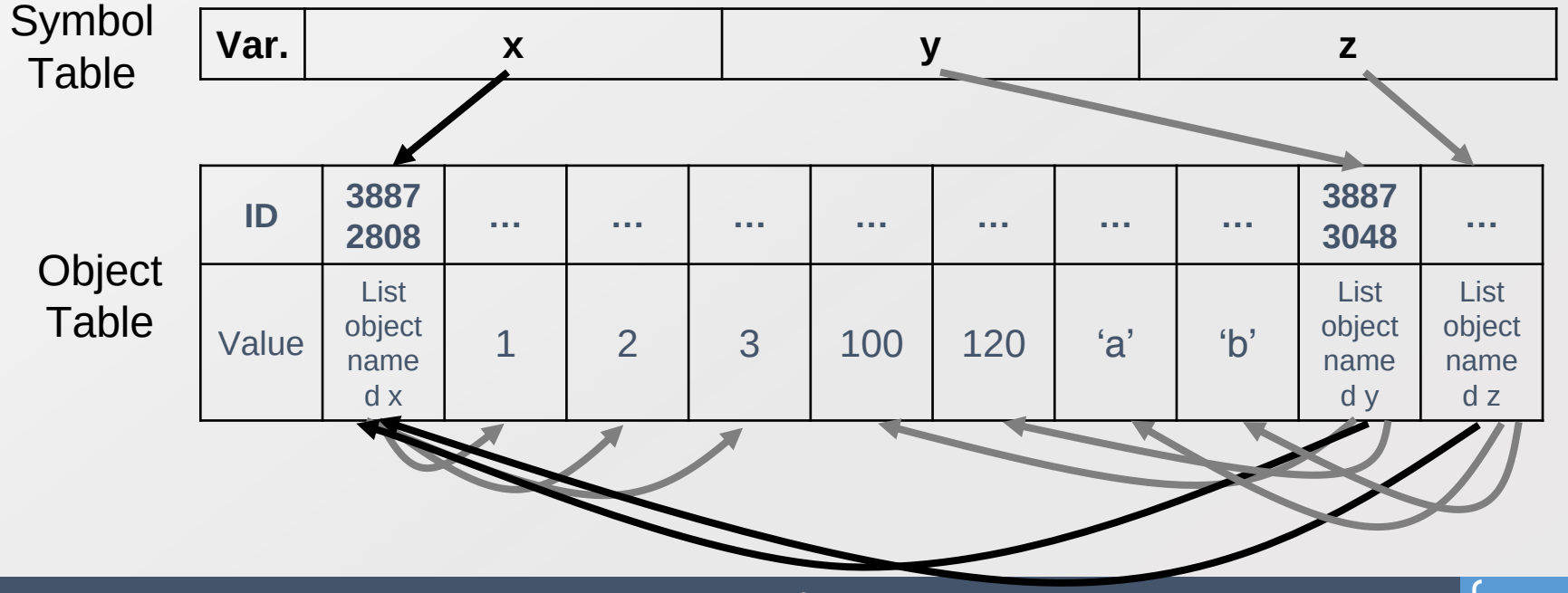
References, Symbol Table, and Object Table

```
import sys

x = [1, 2, 3]
y = [100, x, 120]
z = [x, 'a', 'b']

print(sys.getrefcount(x))
print(sys.getrefcount(y))

print(id(x))
print(id(y[1]))
print(id(y))

print(x is y[1])
print(x is y)
```

```
Run:  16_UnderstandingReference
C:\Users\USER\Anaconda3\python
4
2
2519459127816
2519459127816
2519459127880
True
False
```

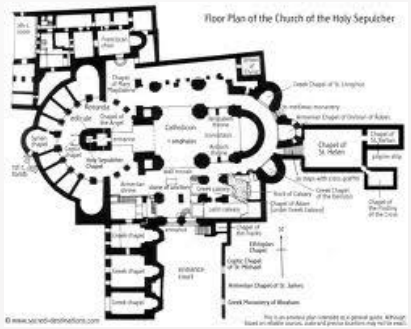
ID(variable**)** returns the referenced object ID
ID(variable1**)==ID(**variable2**)**
→ variabl1 **is** variable2

sys.getrefcount(variable**)** returns the number of references of an object ID + 1



Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Class and Instance



instantiation

```
class MyHome:
    colorRoof = 'red'
    stateDoor = 'closed'
    def paintRoof(self,color):
        self.colorRoof = color
    def openDoor(self):
        self.stateDoor = 'open'
    def closeDoor(self):
        self.stateDoor = 'close'
    def printStatus(self):
        print ("Roof color is", self.colorRoof, ", and door is", self.stateDoor)


homeAtDaejeon = MyHome()
homeAtSeoul = MyHome()
homeAtSeoul.openDoor()
homeAtDaejeon.paintRoof('blue')
homeAtDaejeon.printStatus()
homeAtSeoul.printStatus()
```

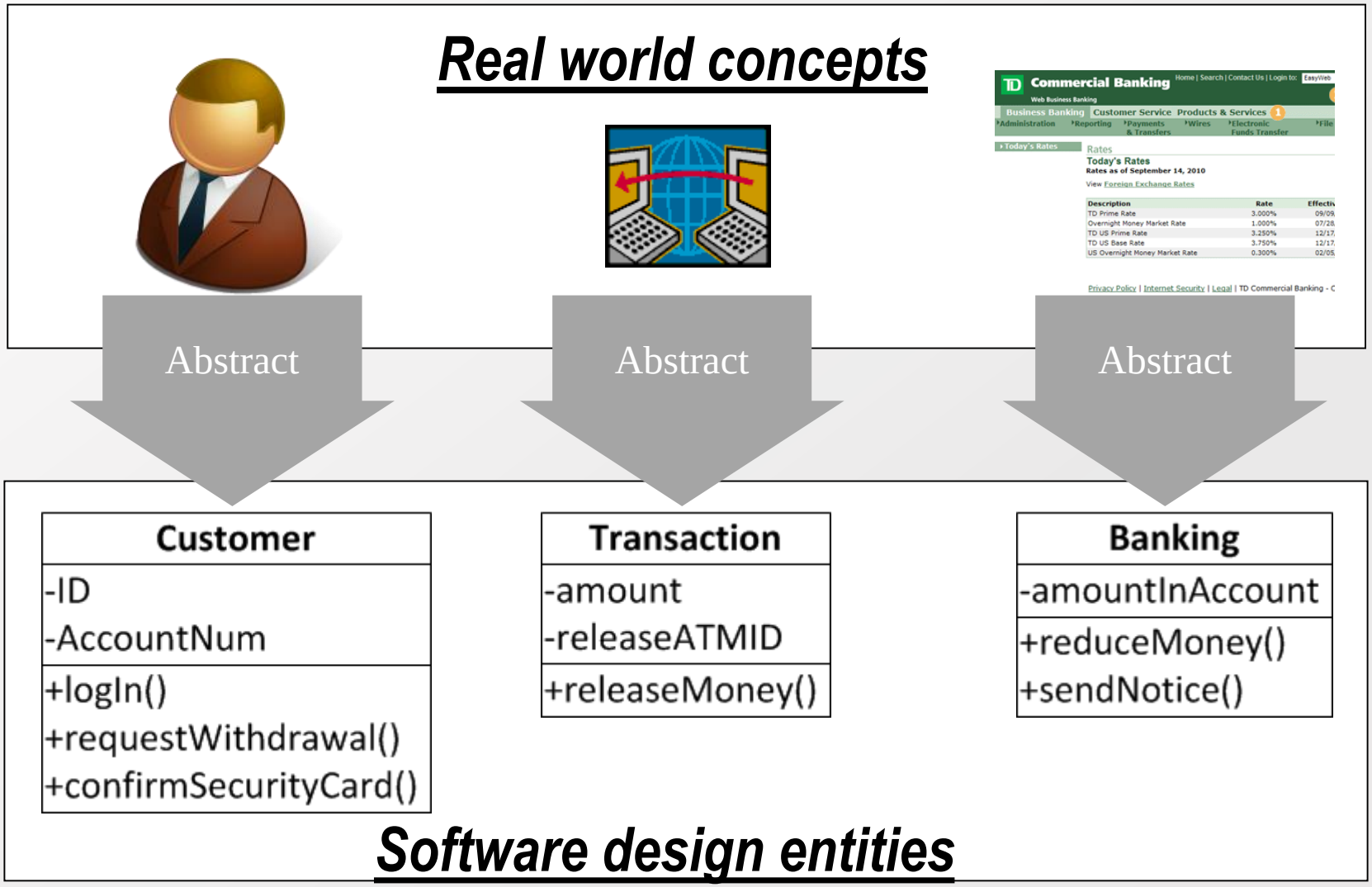
```
Run:  17_ClassInstanceAndObject
C:₩Users₩USER₩Anaconda3₩python.exe C:/User
Roof color is blue , and door is closed
Roof color is red , and door is open
```

See how to define a class
→ **class** classname**:**

See how to instantiate a class
→ var **=** classname**(**param**)**

Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

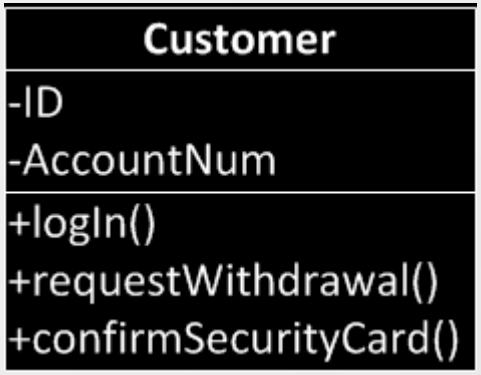
Object-Oriented Design



Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

What are Class and Instance?

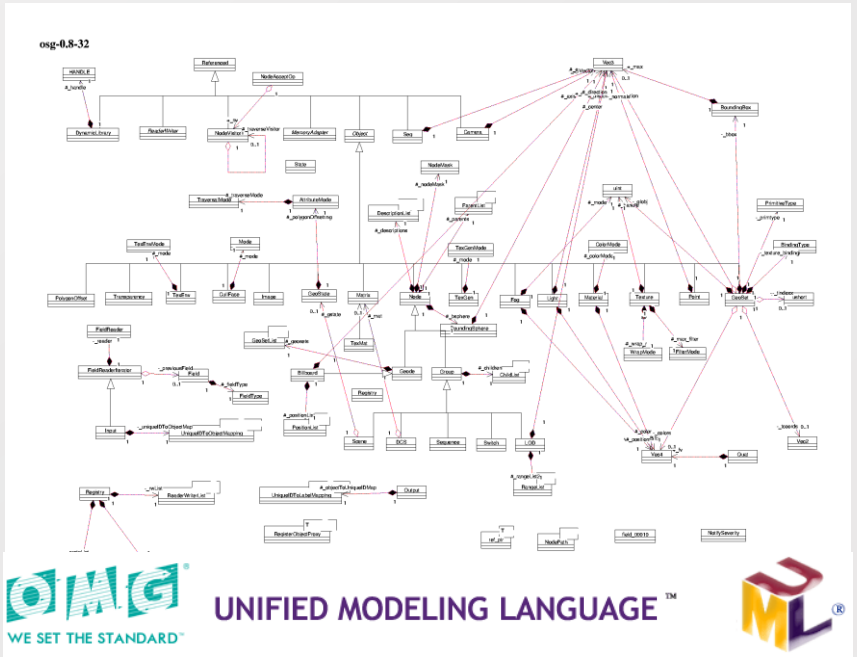
- Class vs. Instance
- Class
  - Result of design and implementation
  - Conceptualization
  - Corresponds to design abstractions
- Instance
  - Result of execution
  - Realization
  - Corresponds to real world entities



Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

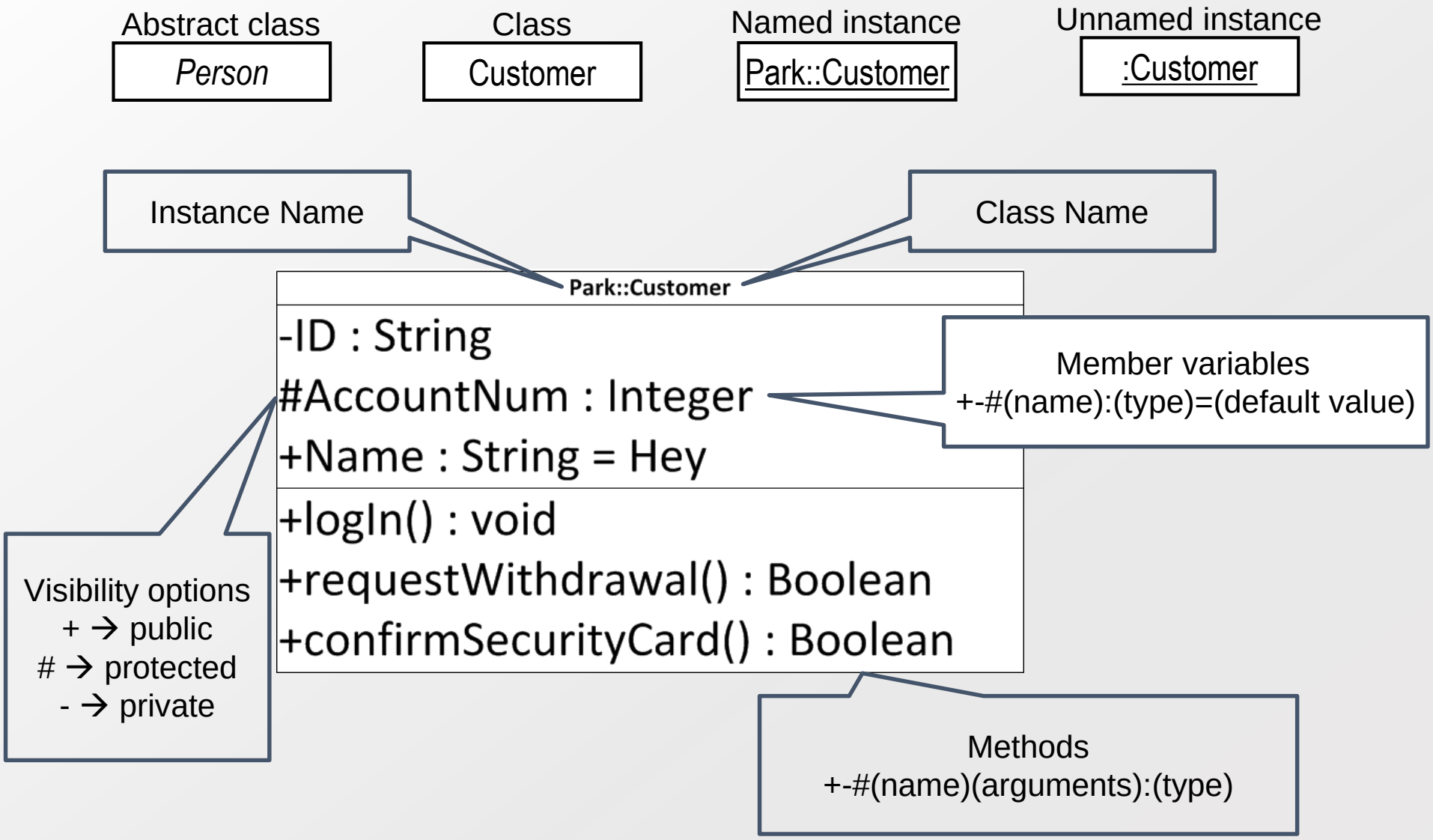
Software Design as House Floorplan

- After your graduation, some of you will be constructors of software
  - Mainly design
  - Some coding
- Need to learn how to communicate your colleagues
  - Learn standard
  - Learn how to represent your design to your boss
- In software engineering,
  - UML is the standard



Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

UML Notation : Class and Instance



Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Encapsulation

- Object = Data + Behavior
  - Data : field, member variable, attribute
  - Behavior : method, member function, operation
- Delegating the implementation responsibility!
  - Bring me a sausage, and I don't care how you made it
- Utilizing the visibility
  - private: seen only within the class
  - protected: seen only within the class and its descendants
  - public: seen everywhere
- Python does not support the visibility options!

Building Architecture

I care overall composition

Room
-location -size
+openDoor() +openWindow()

Interior Designer

I care inside implementation

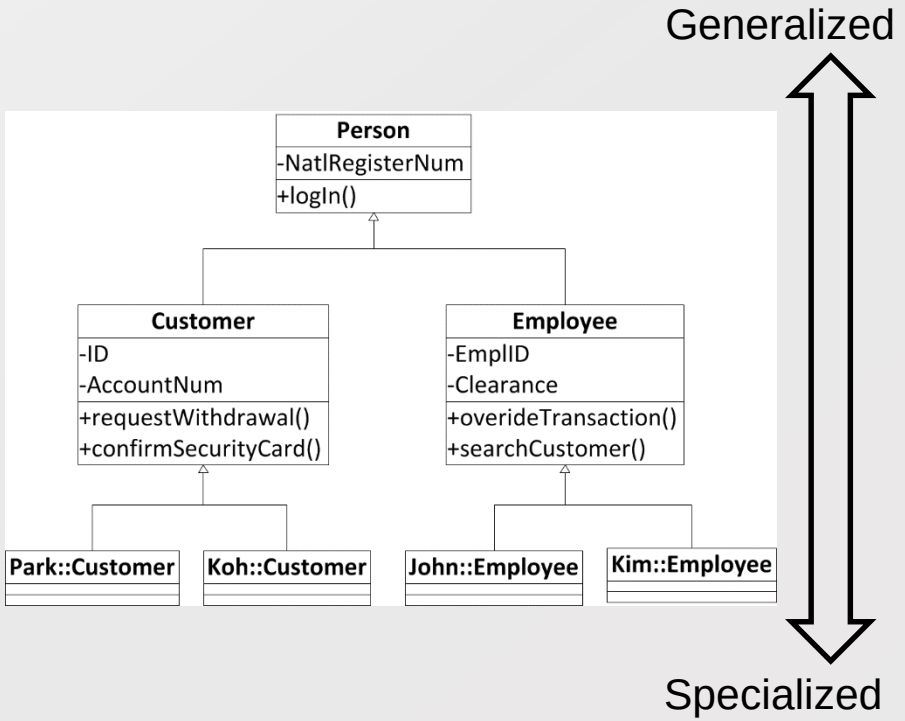
Class Definition

Interface as a specification

Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Inheritance

- Inheritance
  - Giving my attributes to my descendants
    - My attributes include
      - Member variables
      - Methods
  - My descendants may have new attributes of their own
  - My descendants may mask the received attributes
  - But, if not specified, sons follow their father
- Superclass
  - My ancestors, specifically my father
  - Generalized from the conceptual view
- Subclass
  - My descendants, specifically my son
  - Specialized from the conceptual view
- How about having a mother?
  - Yes. It is possible in Python

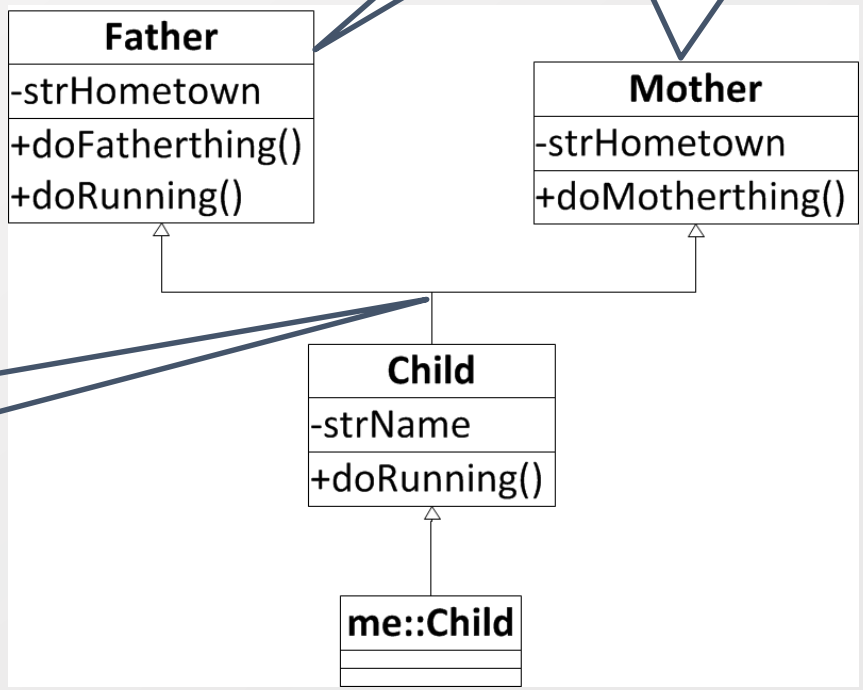


Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Inheritance in Python

```
class Father(object):
    strHometown = "Jeju"
    def __init__(self):
        print("Father is created")
    def doFatherThing(self):
        print("Father's action")
    def doRunning(self):
        print("Slow")

class Mother(object):
    strHometown = "Seoul"
    def __init__(self, paramHome):
        self.strHometown = paramHome
        print("Mother is created")
    def doMotherThing(self):
        print("Mother's action")

class Child(Father, Mother):
    strName = "Moon"
    def __init__(self):
        super(Child,self).__init__()
        print("Child is created")
    def doRunning(self):
        print("Fast")


me = Child()
me.doFatherThing()
me.doMotherThing()
me.doRunning()
print(me.strHometown)
print(me.strName)
```

Base Class (Super Class)



Multiple Inheritance

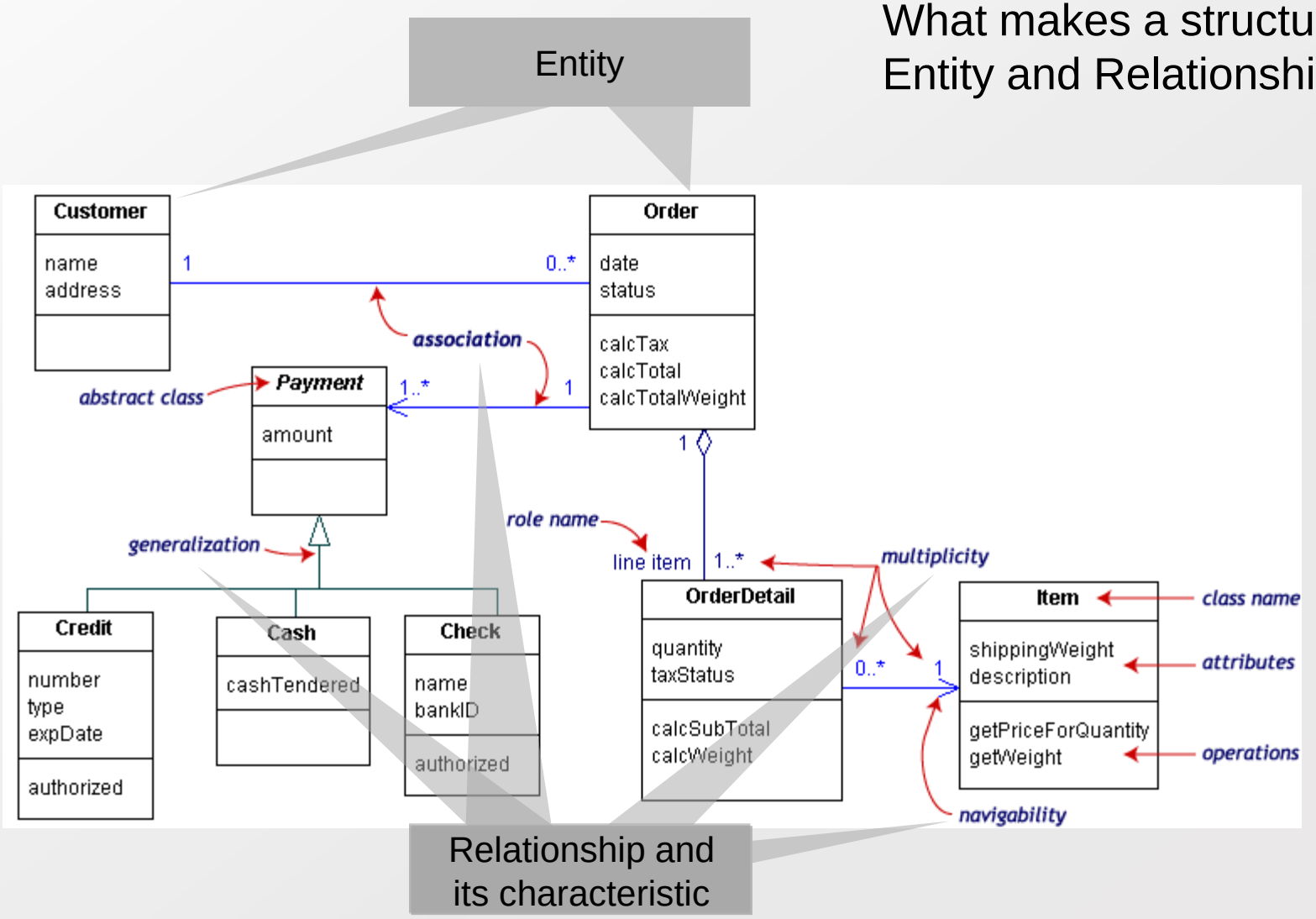
```
Run:  Inheritance
C:\Users\USER\Anaconda3\python.exe C:/Users/USER/Desktop/
Father is created
Child is created
Father's action
Mother's action
Fast
Jeju
Moon
```

1. See Child has Father's and Mother's attributes
2. See Child overwrite Father's method by his own

Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Structure of Classes in Class Diagram

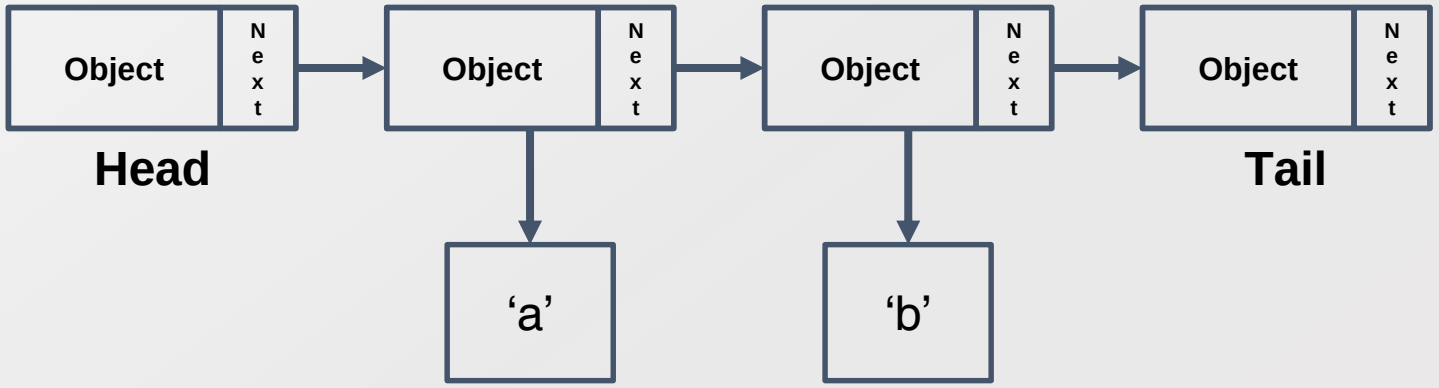
What makes a structure?
Entity and Relationship



Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Basic Structure: Singly Linked List

- Construct a singly linked list with nodes and references
  - A node consists of
    - A variable to hold a reference to its next node
    - A variable to hold a reference to its value object
  - Special nodes: Head and Tail
    - You can construct the singly linked list without them
    - But, using them makes search, insert and delete more convenient
  - Generally, requires more coding than array

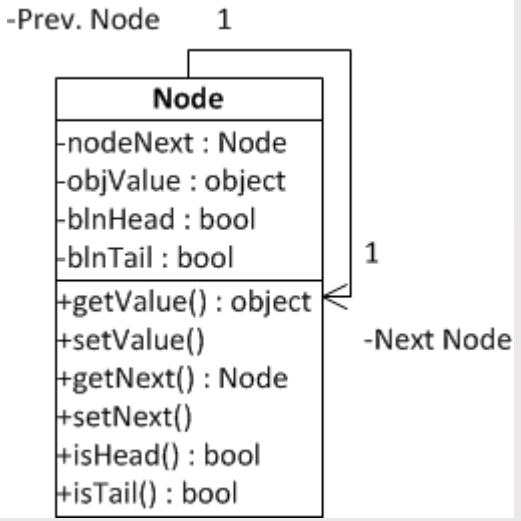


Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Implementation of Node Class

- Member variables
  - Variable to reference the next node
  - Variable to hold a value object
  - (Optional) Variable to check whether it is a head or not
  - (Optional) Variable to check whether it is a tail or not
- Member functions
  - Various set/get methods

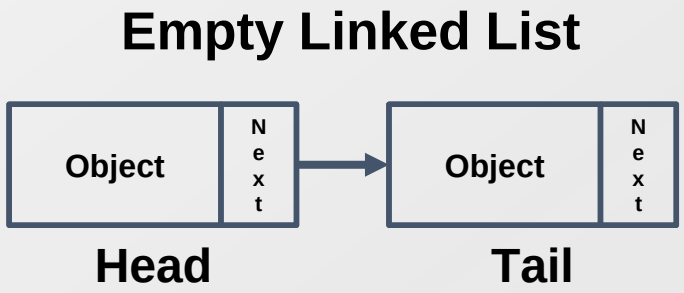
```
class Node:
    nodeNext = None
    nodePrev = ''
    objValue = ''
    blnHead = False
    blnTail = False
    def __init__(self, objValue = '', nodeNext = None, blnHead = False, blnTail = False):
        self.nodeNext = nodeNext
        self.objValue = objValue
        self.blnHead = blnHead
        self.blnTail = blnTail
    def getValue(self):
        return self.objValue
    def setValue(self, objValue):
        self.objValue = objValue
    def getNext(self):
        return self.nodeNext
    def setNext(self, nodeNext):
        self.nodeNext = nodeNext
    def isHead(self):
        return self.blnHead
    def isTail(self):
        return self.blnTail

node1 = Node(objValue = 'a')
nodeTail = Node(blnTail = True)
nodeHead = Node(blnHead = True, nodeNext = node1)
```

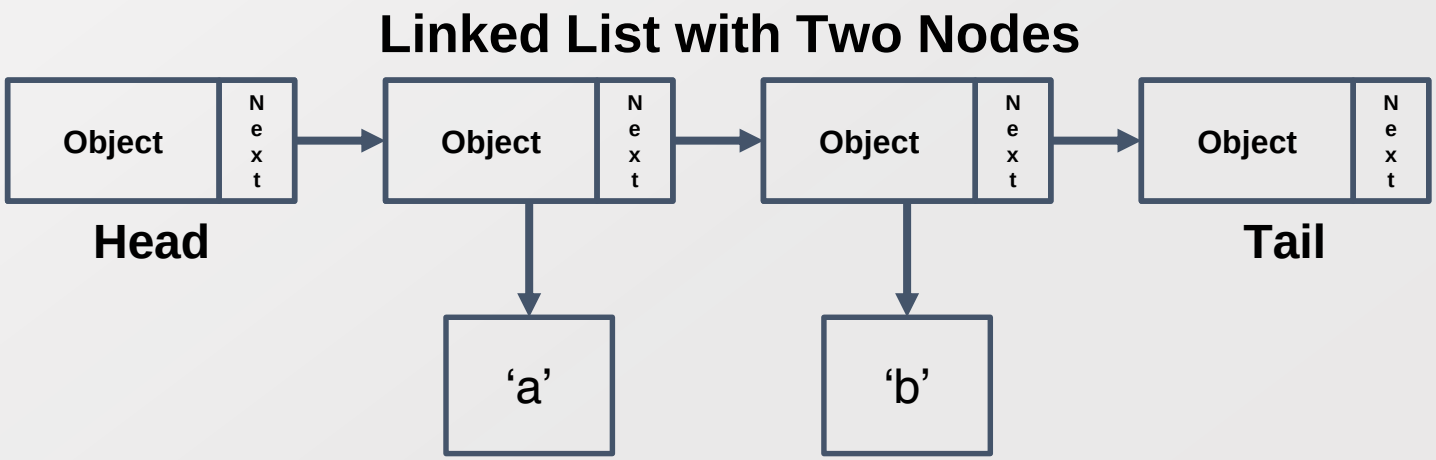


Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Head and Tail

- Specialized node
  - Head : Always at the first of the list
  - Tail : Always at the last of the list
  - These are the two corner stone showing the start and the end of the list
- These are optional nodes.
  - Linked list works okay without these
  - However, having these makes implementation very convenient
  - Any example?

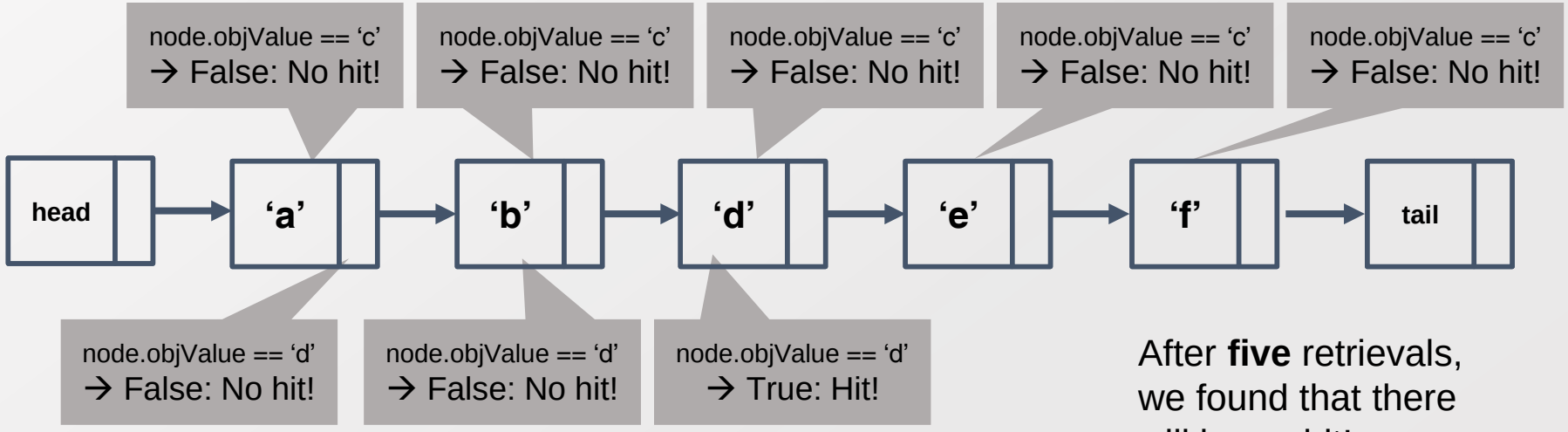




Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Search Procedure in Singly Linked List

- Again, let's find 'd' and 'c' from the list
- Just like an array, navigating from the first to the last until hit is the only way
- No difference in the search pattern, though you cannot use index any further!
  - Your list implementation may include the index function, but it is not required in the linked list



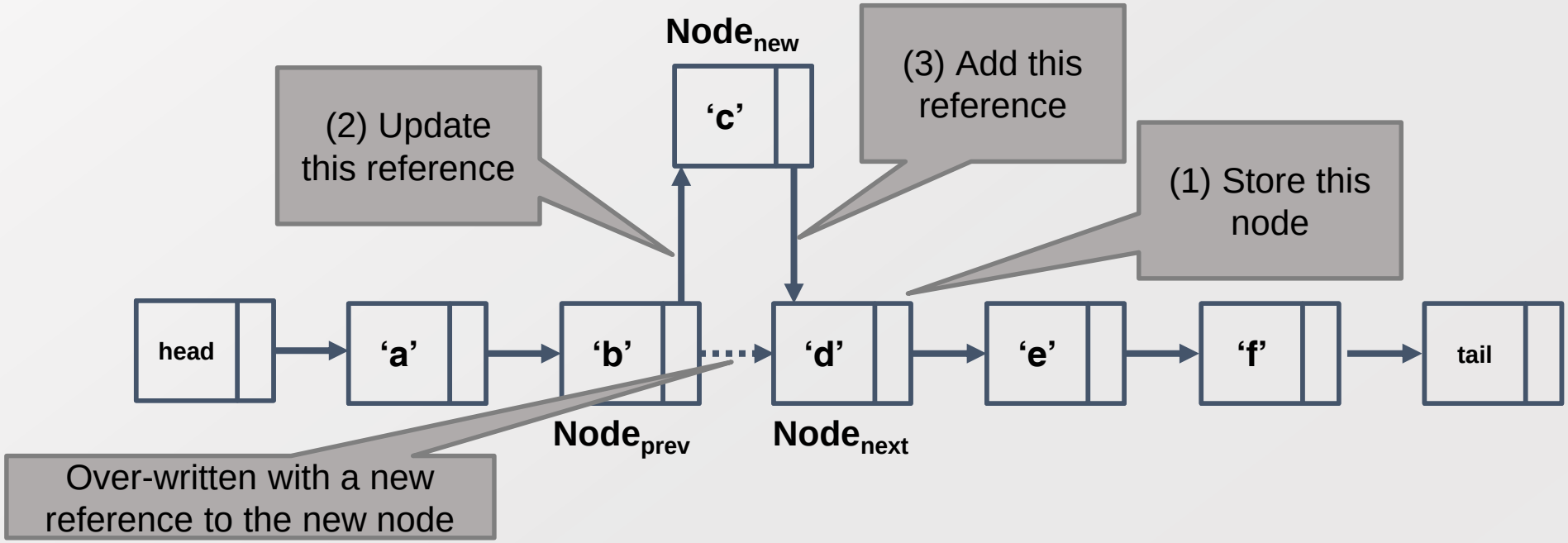
After **three** retrievals,
we found the hit!
(maximum N retrievals)

After **five** retrievals,
we found that there
will be no hit!
(always N retrievals)

Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Insert Procedure in Singly Linked List

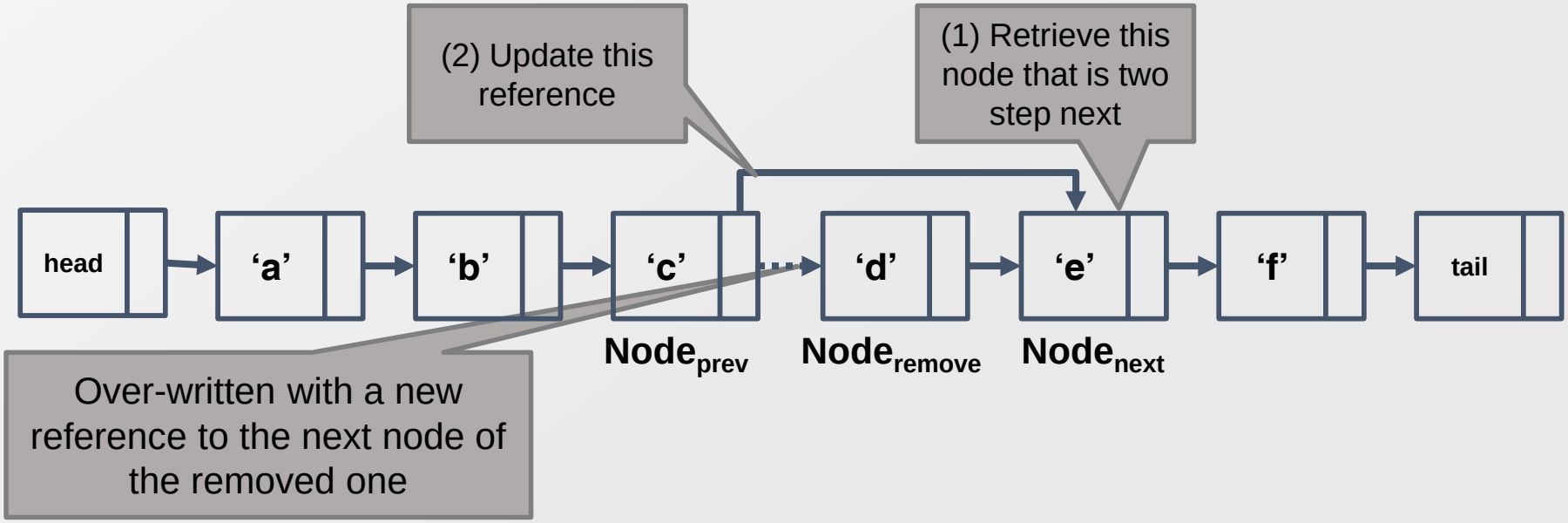
- This is the moment that you see the power of a linked list
- Last time, you need N retrievals to insert a value in the array list
- This time, you need only three operations
  - With an assumption that you have a reference to the node, $Node_{prev}$ that you want to put your new node next
  - First, you store a Node, or a $Node_{next}$, pointed by a reference from $Node_{prev}$'s nodeNext member variable
  - Second, you change a reference from $Node_{prev}$'s nodeNext to $Node_{new}$
  - Third, you change a reference from $Node_{new}$'s nodeNext to $Node_{next}$



Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Delete Procedure in Singly Linked List

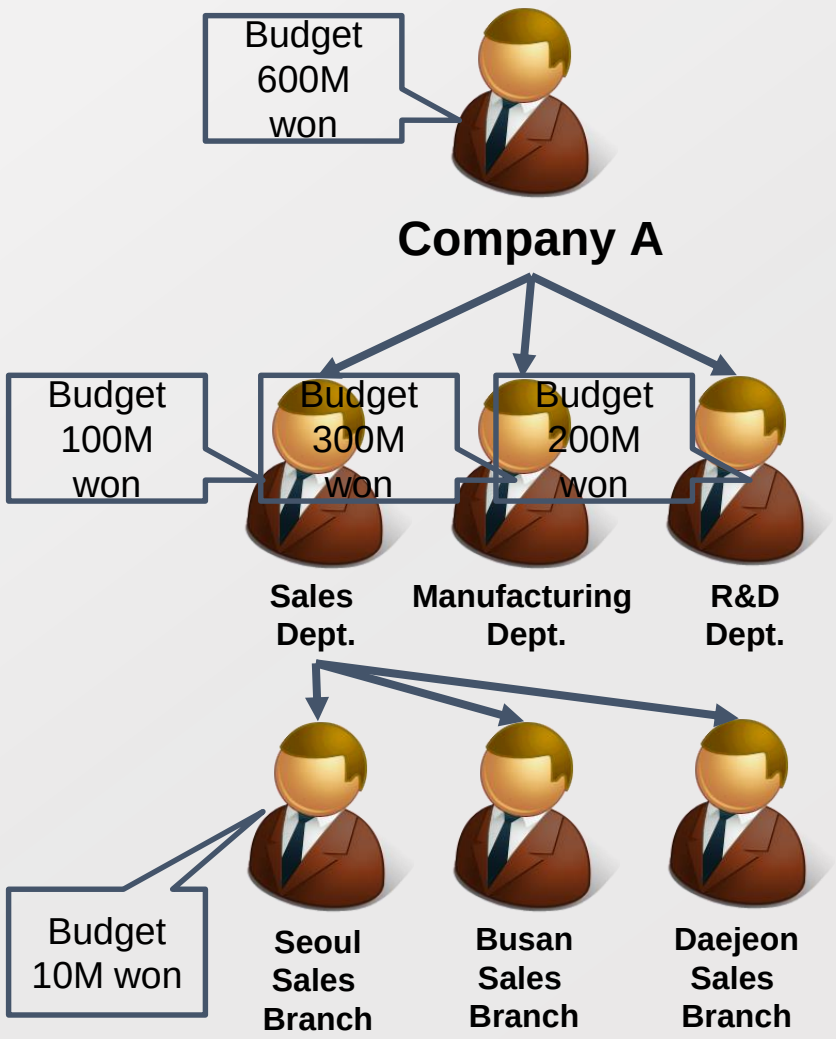
- This is the another moment that you see the power of a linked list
- Last time, you need N retrievals to delete a value in the array list
- This time, you need only three operations
  - With an assumption that you have a reference to the node, $Node_{prev}$ that you want to remove the node next
  - First, you retrieve $Node_{next}$ that is two steps next from $Node_{prev}$
  - Second, you change a reference from $Node_{prev}$'s nodeNext to $Node_{next}$
- The node will be removed because there is no reference to $Node_{remove}$



Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Repeating Problems and Divide and Conquer

- Calculating a budget of a company?
  - Departments consist of the company
  - Departments within departments
- Can't avoid the below structures
  - class Department
    - dept = [sales, manu, randd]
    - def **calculateBudget**(self)
      - Sum = 0
      - For itr in range(0, numDepartments)
        - Sum = sum + dept[itr].**calculateBudget**()
      - Return sum

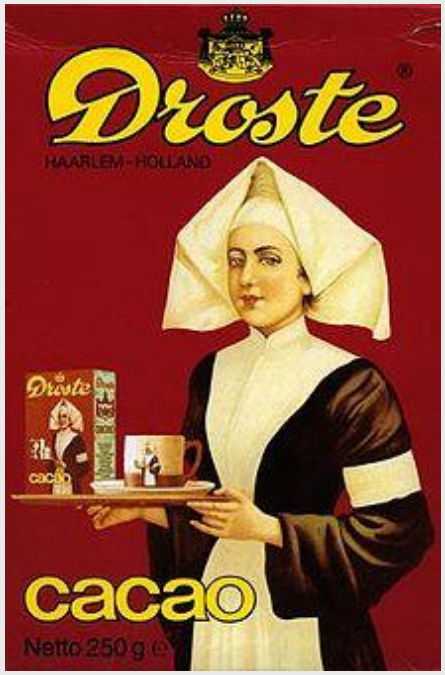


Russian Doll:
Dolls within dolls

Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

More Examples…

- Factorial
  - $Factorial(n) = \begin{cases} 1 & if\ n = 0 \\ n \times (n-1) \times \cdots \times 2 \times 1 & if\ n > 0 \end{cases}$
  - Repeating problems?
    - $Factorial(n) = \begin{cases} 1 & if\ n = 0 \\ n \times Factorial(n-1) & if\ n > 0 \end{cases}$
- Great Common Divisor
  - GCD(32,24) = 8
  - Euclid's algorithm
    - GCD(A, B)=GCD(B, A mod B)
    - GCD(A, 0)=A
- Commonality
  - Repeating function calls
  - Reducing parameters
  - Just like the mathematical induction



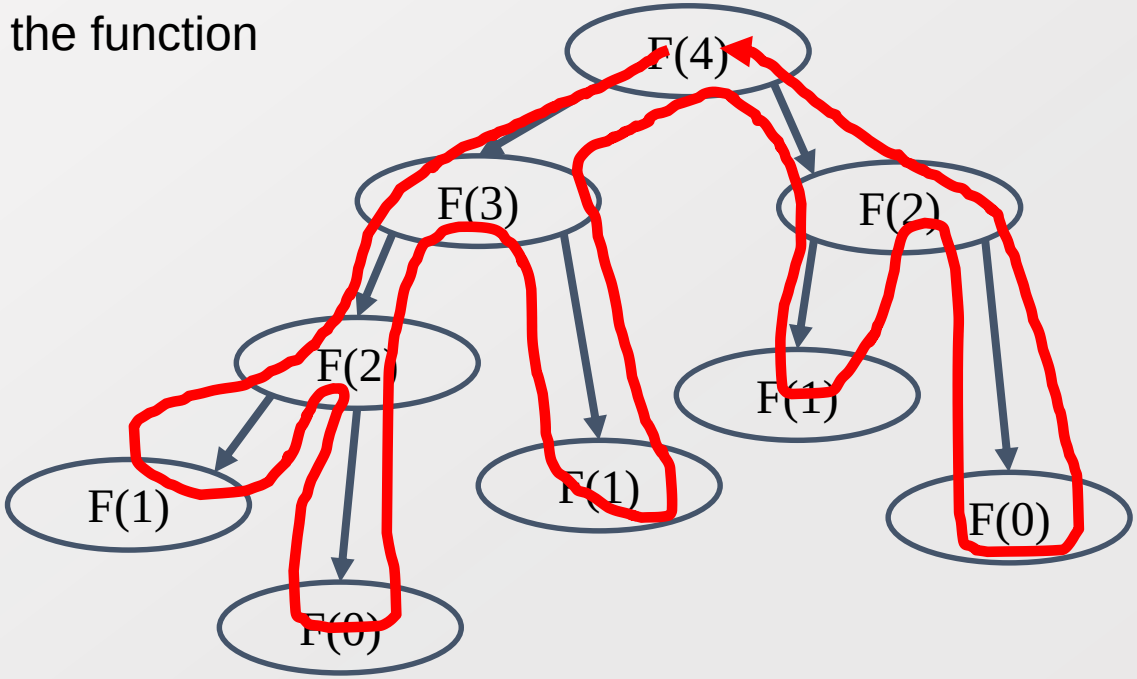
Self-Similar

Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Recursion

- A programming method to handle the repeating items in a self-similar way
  - Often in a form of
    - Calling a function within the function
      - def functionA(target)
        - ….
        - functionA(target')
        - ….
        - if (escapeCondition)
          - Return A;

Program Execution Flow




```
def Fibonacci(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    intRet = Fibonacci(n-1) + Fibonacci(n-2)
    return intRet

for itr in range(0, 10):
    print(Fibonacci(itr), end=" ")
```

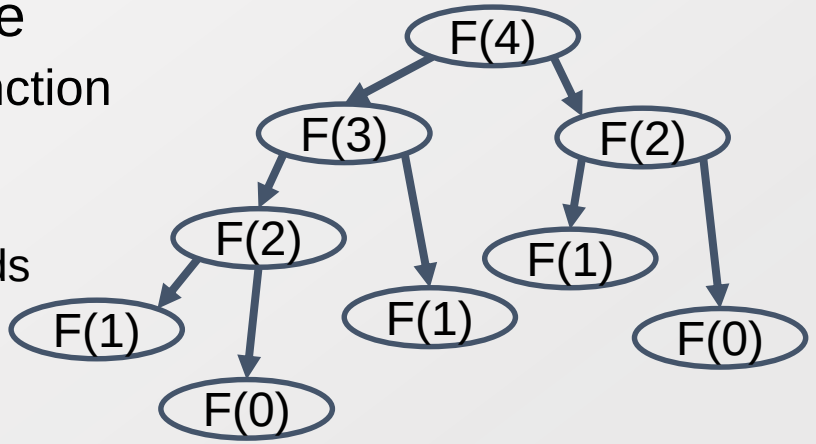
```
Run:  FibonacciSequence
C:₩Users₩USER₩Anaconda3₩python.exe C:/Us
0 1 1 2 3 5 8 13 21 34
```

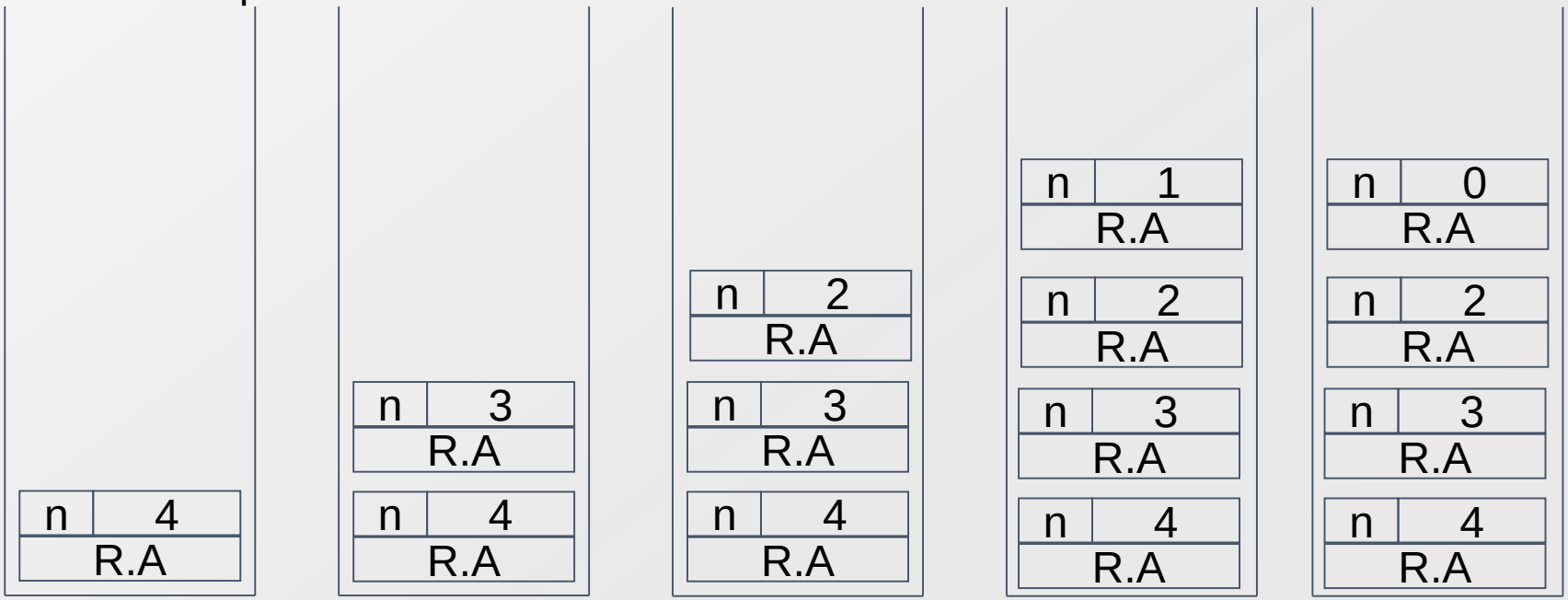
$$Fibonacci(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ Fibonacci(n-1) + Fibonacci(n-2) & n \geq 2 \end{cases}$$

Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Recursions and Stackframe

- Recursion of functions
  - Increase the items in the stackframe
    - Stackframe is a stack storing your function call history
      - Push: When a function is invoked
      - Pop: When a function hits return or ends
      - What to store?
        - Local variables and function call parameters

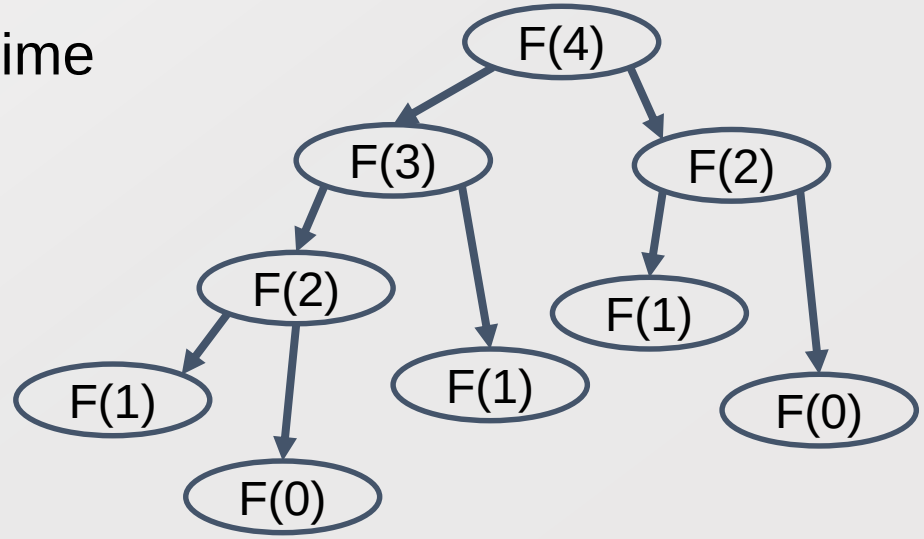




Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Problems in Recursions of Fibonacci Sequence

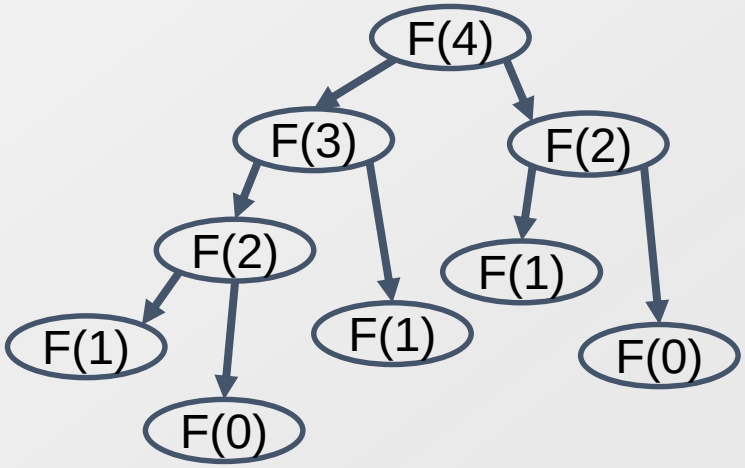
- Problems in recursions
  - Excessive function calls
    - Calling functions again and again
    - Even though the function is executed before with the same parameters
- For instance, Fibonacci(4)
  - Has two repeated calls of F(0)
  - Has three repeated calls of F(1)
  - Has two repeated calls of F(2)
- These are unnecessarily taking time and space
- How to solve this problem?



Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Dynamic Programming

- Dynamic programming:
  - A general algorithm design technique for solving problems defined by or formulated as **recurrences with overlapping sub-instances**
  - In this context, Programming == Planning
- Main storyline
  - Setting up a recurrence
    - Relating a solution of a larger instance to solutions of some smaller instances
    - Solve small instances once
    - Record solutions in a table
    - Extract a solution of a larger instance from the table

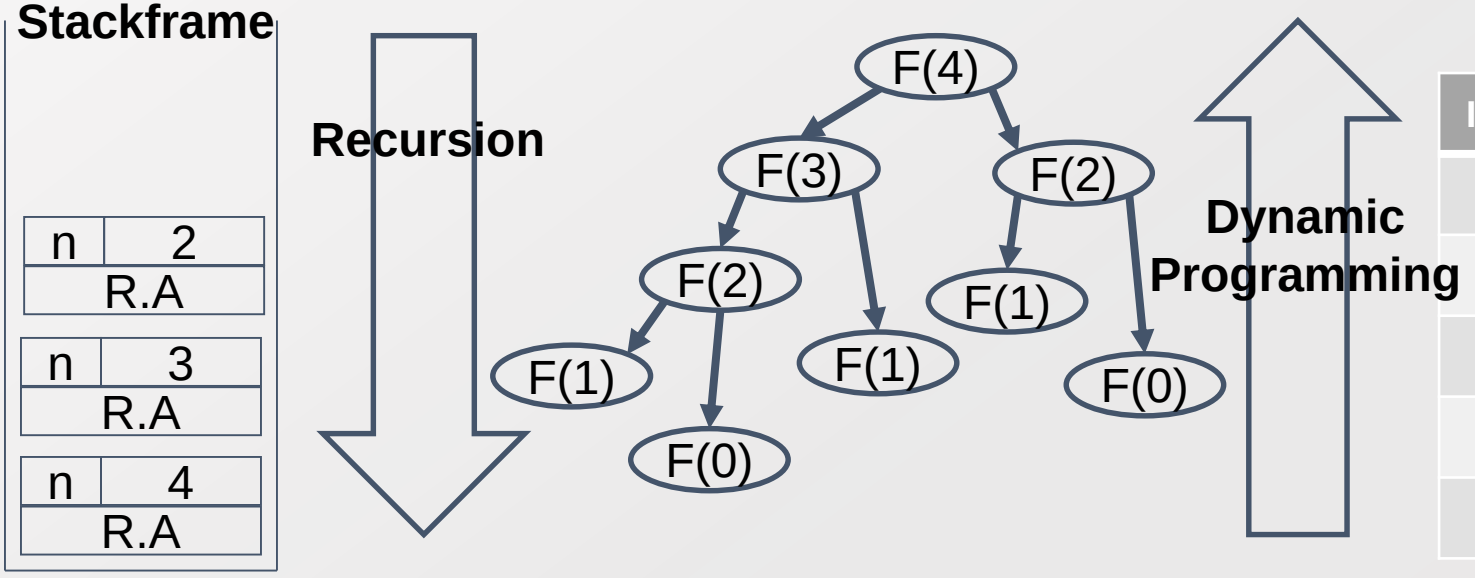


Instance	Solution
F(0)	0
F(1)	1
F(2)	1
F(3)	2
F(4)	?

Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Memoization

- Key technique of dynamic programming
  - Simply put
    - Storing the results of previous function calls to reuse the results again in the future
  - More philosophical sense
    - Bottom-up approach for problem-solving
      - Recursion: Top-down of divide and conquer
      - Dynamic programming: Bottom-up of storing and building



Memoization

Instance	Solution
F(0)	0
F(1)	1
F(2)	1
F(3)	2
F(4)	3

Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Implementation Example: Fibonacci Sequence in DP

```
def FibonacciDP(n):
    dicFibonacci = {}
    dicFibonacci[0] = 0
    dicFibonacci[1] = 1
    for itr in range(2, n + 1):
        dicFibonacci[itr] = dicFibonacci[itr-1] + dicFibonacci[itr-2]
    return dicFibonacci[n]

for itr in range(0, 10):
    print(FibonacciDP(itr), end=" ")
```

Setting up a memoization table

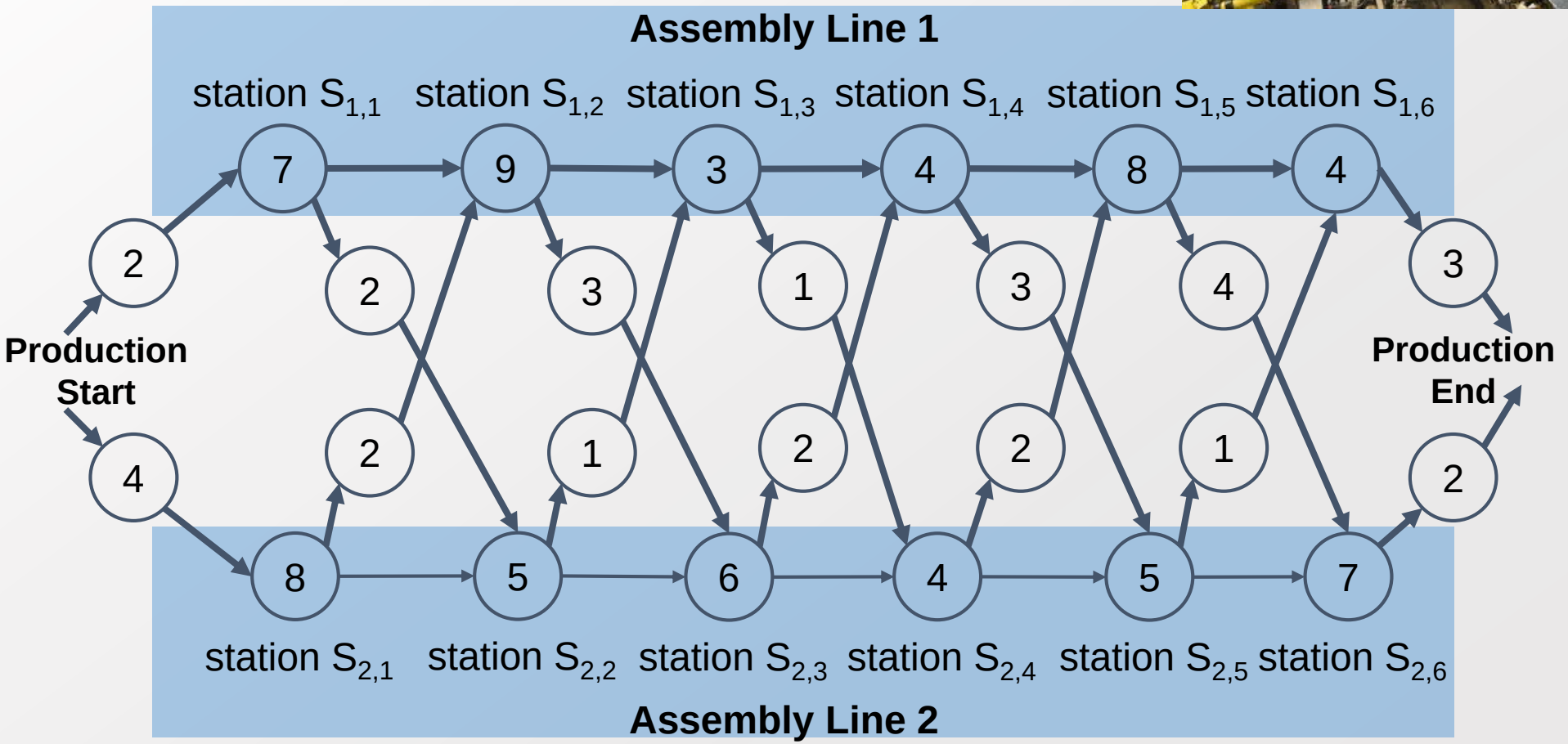
Building up a bigger solutions

Execution Part

- Use a dictionary collection variable type for memoization
  - Memoization
    - Storing a fibonacci number for a particular index
- Now,
  - We have a new space requirement, the dictionary or the table, of O(N)
  - We have reduced execution time from $O(2^n)$ to O(N)

Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

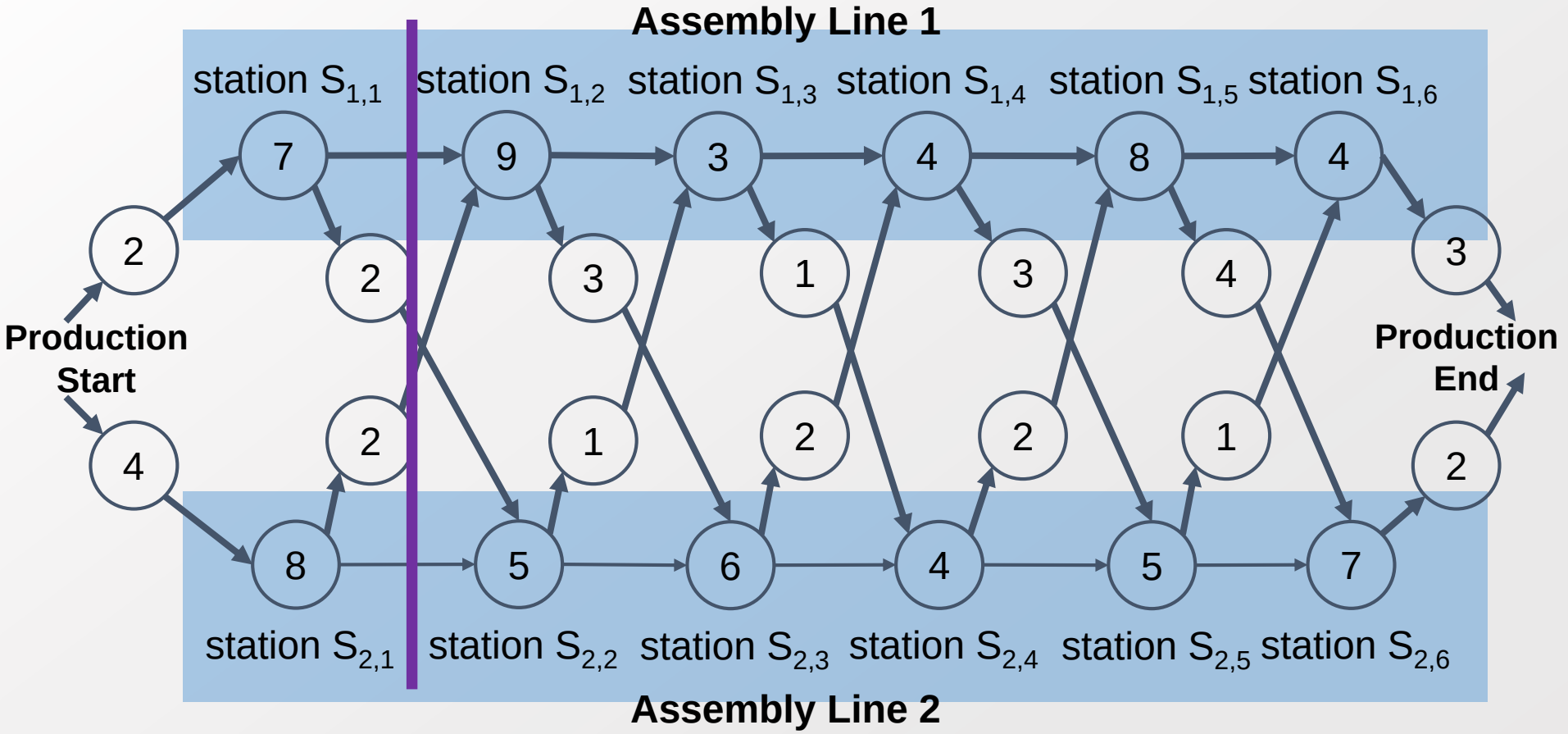
Assembly Line Scheduling



Goal: Computing the fastest production route

Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Process of Assembly Line Scheduling

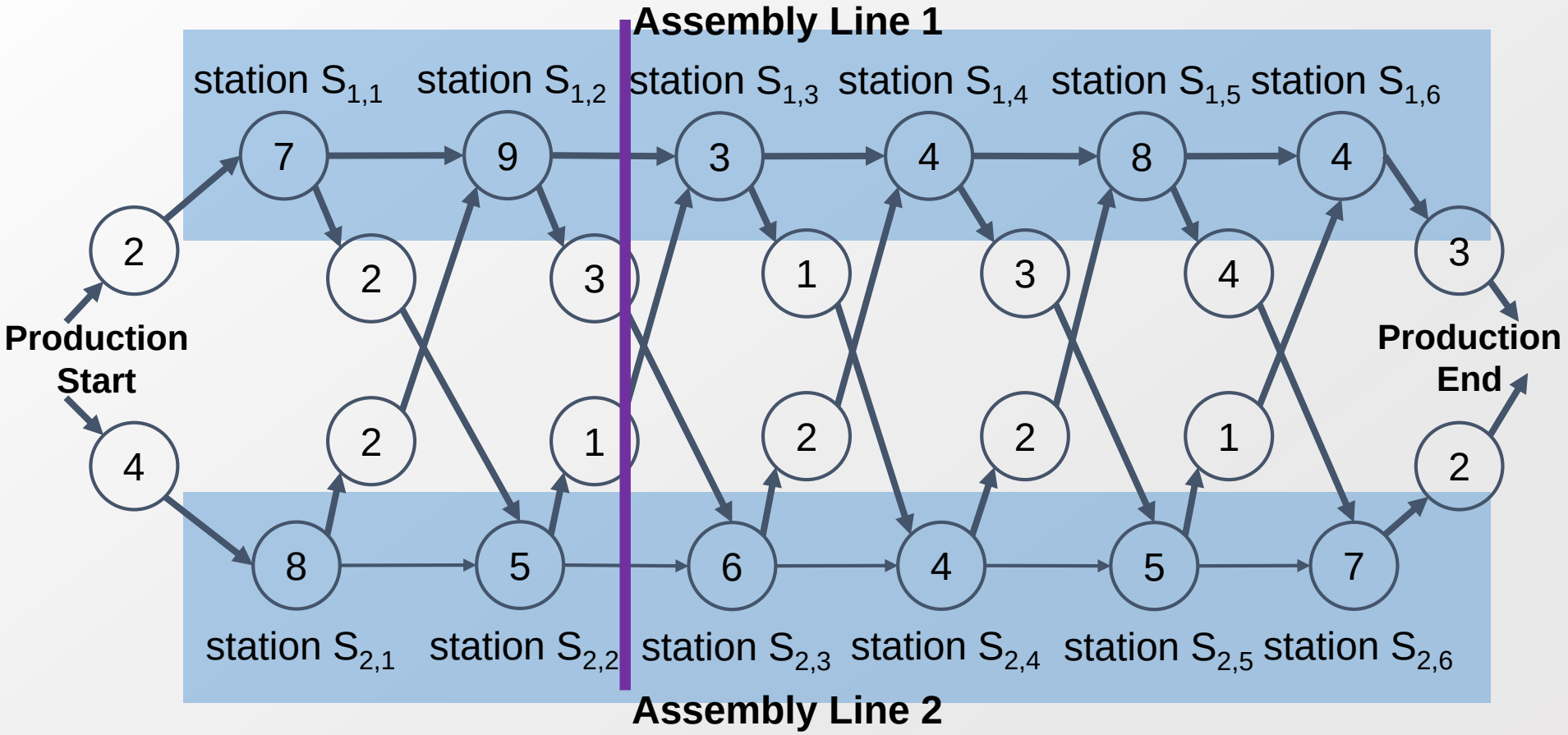


Time	1	2	3	4	5	6
L1	9					
L2	12					

Trace	1	2	3	4	5	6
L1	S					
L2	S					

Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Process of Assembly Line Scheduling

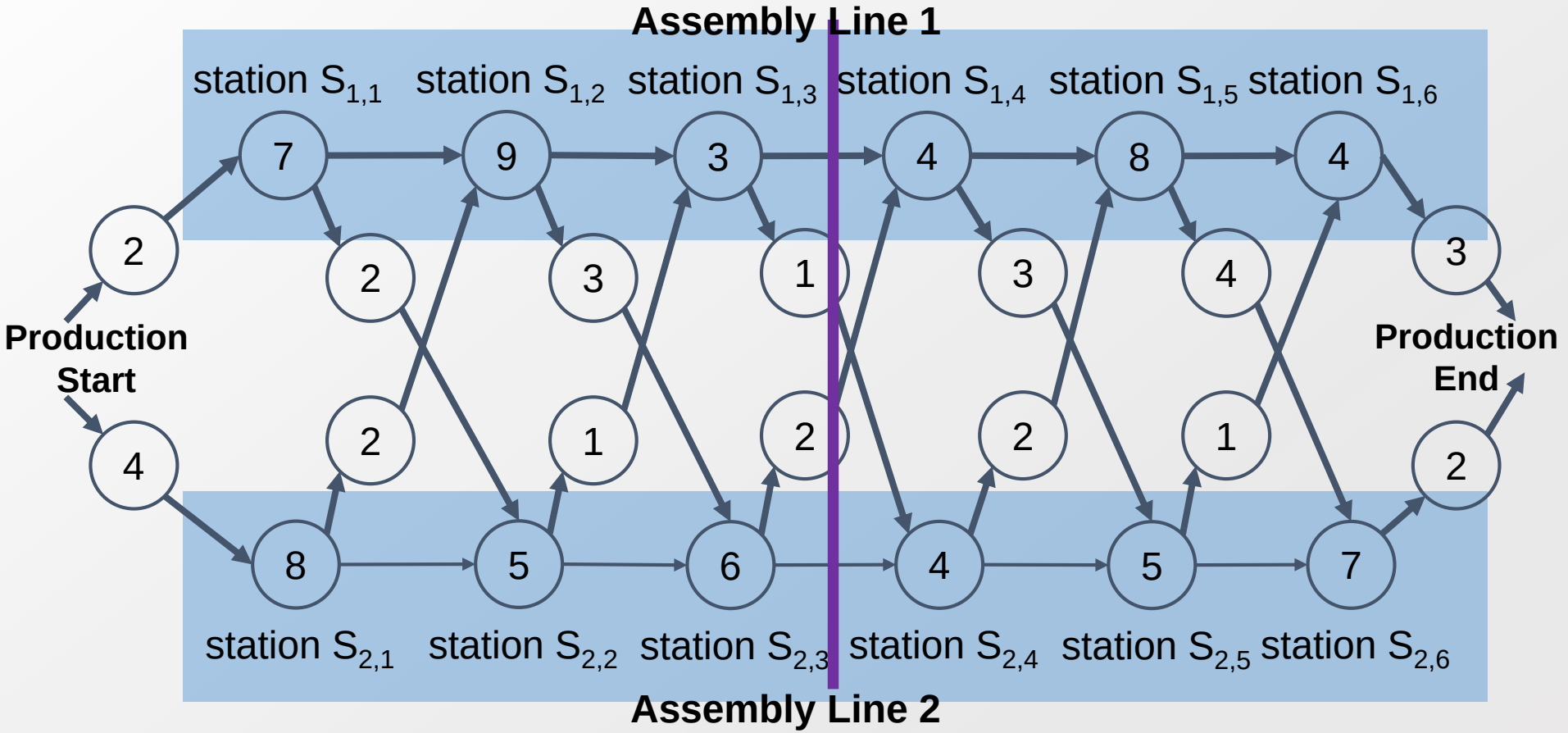


Time	1	2	3	4	5	6
L1	9	18				
L2	12	16				

Trace	1	2	3	4	5	6
L1	S	1				
L2	S	1				

Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Process of Assembly Line Scheduling

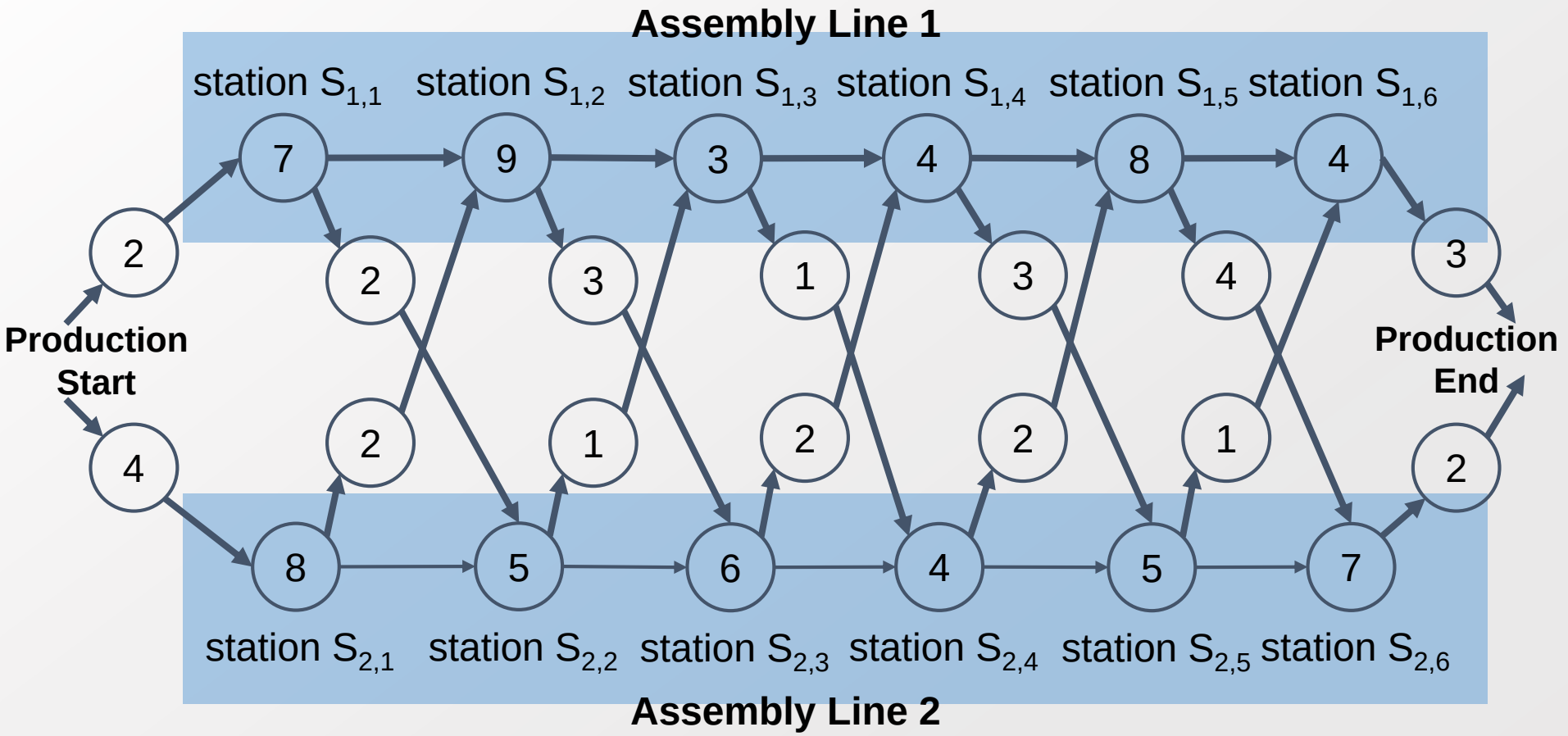


Time	1	2	3	4	5	6
L1	9	18	20			
L2	12	16	22			

Trace	1	2	3	4	5	6
L1	S	1	2			
L2	S	1	2			

Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Process of Assembly Line Scheduling



Time	1	2	3	4	5	6
L1	9	18	20	24	32	35
L2	12	16	22	25	30	37

Trace	1	2	3	4	5	6
L1	S	1	2	1	1	2
L2	S	1	2	1	2	2

Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

EXERCISE

Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Introduction

- Building an AI playing Blackjack game
  - Customized game rule
    - Two players only : AI and player
    - Start with no cards
    - Cards with values : J=11, Q=12, K=13, A=1
    - Can see five cards in advance for each round
    - Each round has two decisions
      - First, Player : Hit or Pass : Hit means receiving a card
      - Second, AI : Hit or Pass : Hit means receiving a card
    - Winner
      - First reach 21 for the card sum
        - Beyond 21 == Lose
      - Both pass for a single round
        - Higher score wins
        - Same score means tie

Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Expected Result

- After running ‘Blackjack.py’
- C:\Users\win\Anaconda3\python.exe "D:/SESLab-Teaching/Blackjack.py"
- ----------
- Round : 0
- ['7-Spade', '7-Diamond', 'K-Spade', '8-Spade', '10-Clover']
- AI Current Score : 0
- AI's Hand : []
- Player Current Score : 0
- Player's Hand : []
- Player Cards to Play : 7-Spade
- (H)it or (P)ass ??? Type!
- H
- AI Observed Cards : ['7-Diamond', 'K-Spade', '8-Spade', '10-Clover']
- INSIDE AI : AI Expected Total Score : 21
- INSIDE AI : AI Selected Cards : ['K-Spade', '8-Spade']
- AI Cards to Play : 7-Diamond
- AI Pass
- AI Current Score : 0
- Player Current Score : 7
- ----------
- Round : 1
- ['7-Diamond', 'K-Spade', '8-Spade', '10-Clover', 'Q-Spade']
- AI Current Score : 0
- AI's Hand : []
- Player Current Score : 7
- Player's Hand : ['7-Spade']
- Player Cards to Play : 7-Diamond
- (H)it or (P)ass ??? Type!
- H
- AI Observed Cards : ['K-Spade', '8-Spade', '10-Clover', 'Q-Spade']
- INSIDE AI : AI Expected Total Score : 21
- INSIDE AI : AI Selected Cards : ['K-Spade', '8-Spade']
- AI Cards to Play : K-Spade
- AI Hit
- AI Current Score : 13
- Player Current Score : 14
- ----------
- Round : 2
- ['8-Spade', '10-Clover', 'Q-Spade', 'A-Diamond', '6-Heart']
- AI Current Score : 13
- AI's Hand : ['K-Spade']
- Player Current Score : 14
- Player's Hand : ['7-Spade', '7-Diamond']
- Player Cards to Play : 8-Spade
- (H)it or (P)ass ??? Type!
- P
- AI Observed Cards : ['8-Spade', '10-Clover', 'Q-Spade', 'A-Diamond', '6-Heart']
- INSIDE AI : AI Expected Total Score : 21
- INSIDE AI : AI Selected Cards : ['K-Spade', '8-Spade']
- AI Cards to Play : 8-Spade
- AI Hit
- AI Current Score : 21
- Player Current Score : 14
- AI Win!

- Process finished with exit code 0

Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Expected Result

Round : 0

['7-Spade', '7-Diamond', 'K-Spade', '8-Spade', '10-Clover'] **Player can observe these cards**

AI Current Score : 0 **Current score of AI**

AI's Hand : [] **AI has these cards**

Player Current Score : 0 **Current score of player**

Player's Hand : [] **Player has these cards**

Player Cards to Play : 7-Spade **Player has to make a decision whether or not to get this card**

(H)it or (P)ass ??? Type! **Make decision of player's action**

Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Expected Result

Round : 0

['7-Spade', '7-Diamond', 'K-Spade', '8-Spade', '10-Clover'] **Player can observe these cards**

AI Current Score : 0 **Current score of AI**

AI's Hand : [] **AI has these cards**

Player Current Score : 0 **Current score of player**

Player's Hand : [] **Player has these cards**

Player Cards to Play : 7-Spade **Player has to make a decision whether or not to get this card**

(H)it or (P)ass ??? Type! **Make decision of player's action**

H

Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Expected Result

Round : 0
['7-Spade', '7-Diamond', 'K-Spade', '8-Spade', '10-Clover'] **Player can observe these cards**
AI Current Score : 0 **Current score of AI**
AI's Hand : [] **AI has these cards**
Player Current Score : 0 **Current score of player**
Player's Hand : [] **Player has these cards**
Player Cards to Play : 7-Spade **Player has to make a decision whether or not to get this card**
(H)it or (P)ass ??? Type! **Make decision of player's action**
H
AI Observed Cards : ['7-Diamond', 'K-Spade', '8-Spade', '10-Clover'] **AI can observe these cards**
INSIDE AI : AI Expected Total Score : 21 **Expected score based on the strategy of AI**
INSIDE AI : AI Selected Cards : ['K-Spade', '8-Spade'] **Selected cards based on the strategy of AI**
AI Cards to Play : 7-Diamond **If AI hits, then AI will get this card**
AI Pass
AI Current Score : 0
Player Current Score : 7 **Current score of end of round**

Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Dynamic Programming for Blackjack

- Core problem
  - How to find the best combination to reach 21 with the current observations and the current hand
  - This is a customized problem of Knapsack problem
    - English Wikipedia provides a good description
- Key variable
  - self.tblSelect
    - [index = self.scoreAI]
    - selected cards from the observed cards + current hand(self.lstAIHand)
  - self.tblRemaining
    - [index = self.scoreAI]
    - cards for selection (initially at the method beginning : self.observedCards), each element is a list
  - self.tblBestSelect
    - the selection set closest to 21

Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST

Memoization Table for Blackjack

Index =Score	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
tblSelect	-							7	8		10			K		7, 8		7, 10	8, 10
tblRemaining	7, K, 8, 10							K, 8, 10	7, K, 10		7, K, 8			7, 8, 10		K, 10		K, 8,	7, K

- Structure the memorization table
  - See the score is used as an index
  - Make the remaining and the selected card lists
- Retrace from 21 score
  - The first selection will be the closest to 21

AI Current Score : 0
AI's Hand : []
Player Current Score : 0
Player's Hand : []
Player Cards to Play : 7-Spade
(H)it or (P)ass ??? Type!
H
AI Observed Cards : ['7-Diamond', 'K-Spade', '8-Spade', '10-Clover']
INSIDE AI : AI Expected Total Score : 21
INSIDE AI : AI Selected Cards : ['K-Spade', '8-Spade']

Copyright © 2019 by Il-Chul Moon, AAILab, Dept. of Industrial and Systems Engineering, KAIST