

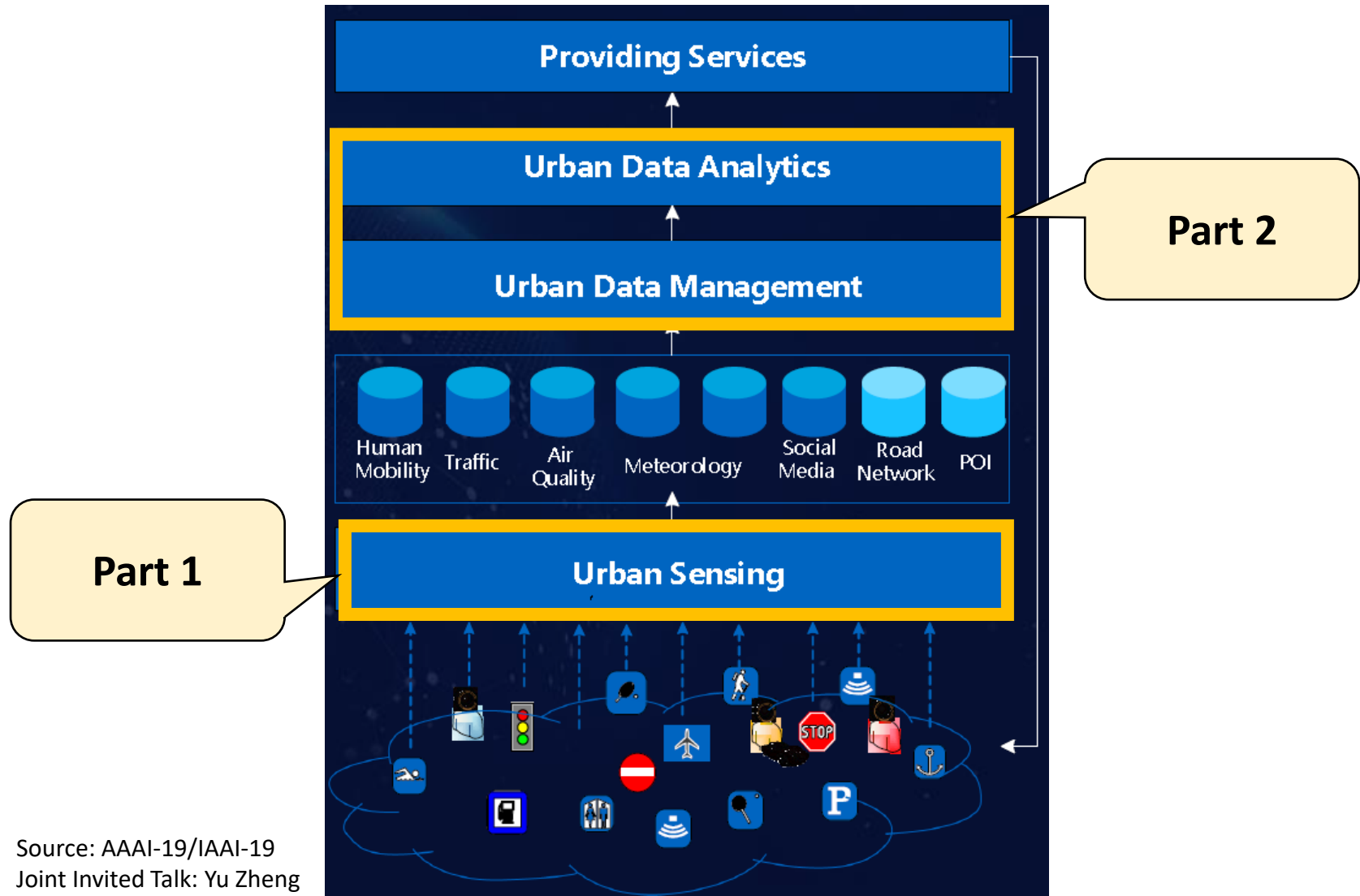
Part 2 - 01: Urban Data Management (1) (Spatial Data)

Long Cheng

Assistant Professor

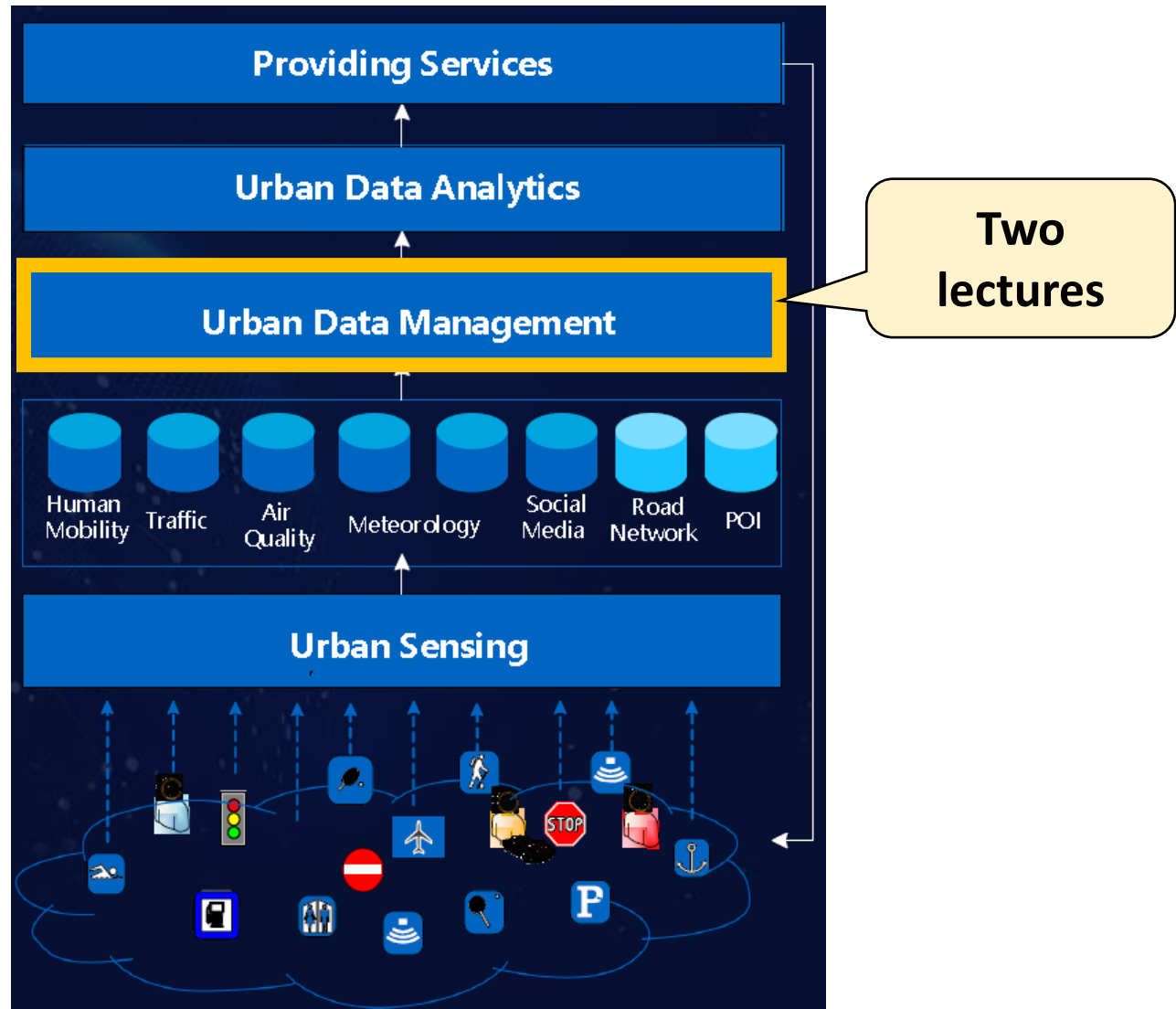
c.long@ntu.edu.sg

Urban Computing: Overview



Source: AAAI-19/IAAI-19
Joint Invited Talk: Yu Zheng

Urban Computing: Overview



Source: AAAI-19/IAAI-19
Joint Invited Talk: Yu Zheng

Urban Data Management

Preprocessing

- How to preprocess the urban data?

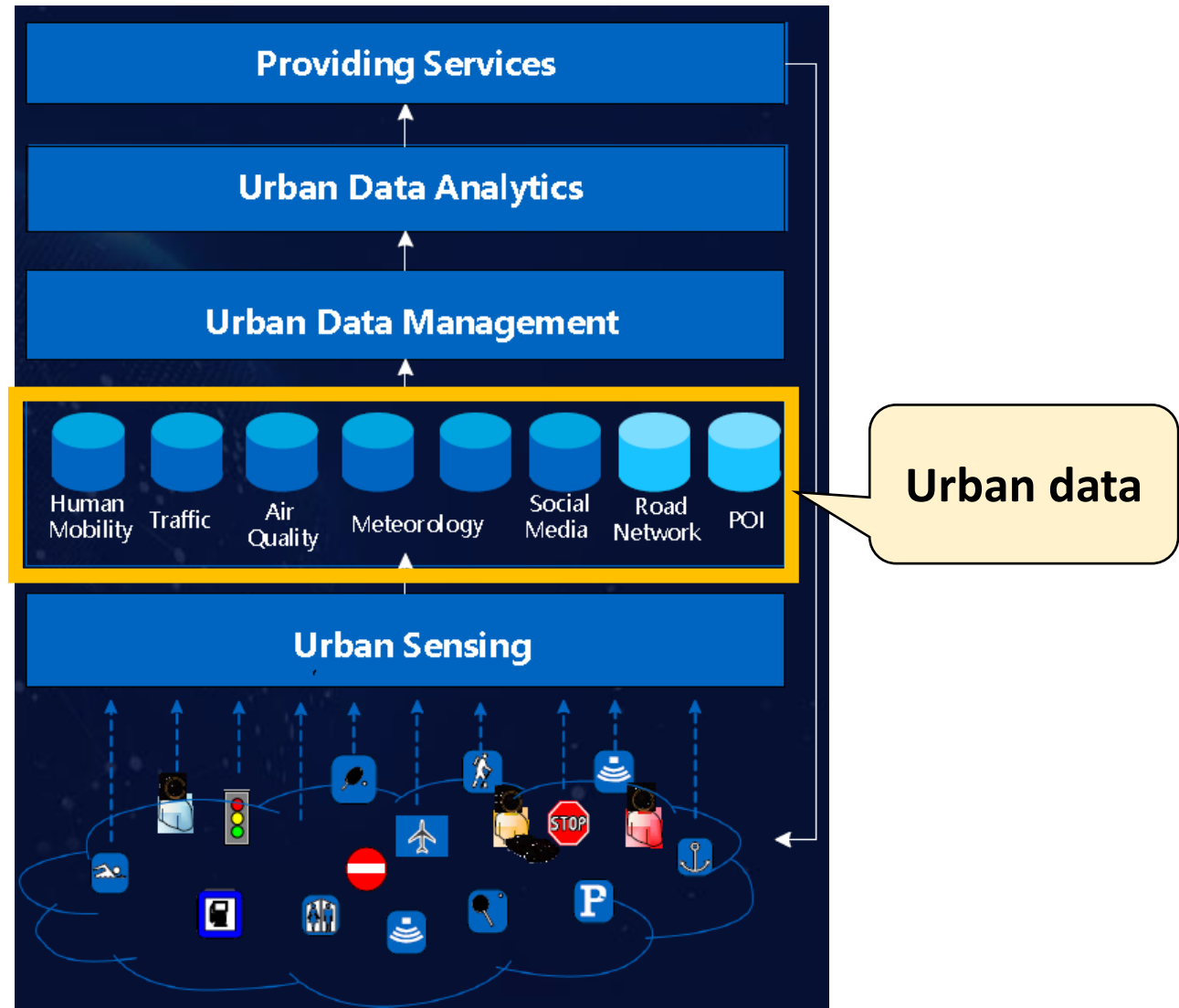
Indexing

- How to build indexes on the urban data?

Query Processing

- How to answer queries on the urban data

Urban Computing: Overview



Source: AAAI-19/IAAI-19
Joint Invited Talk: Yu Zheng

Urban Data: Examples



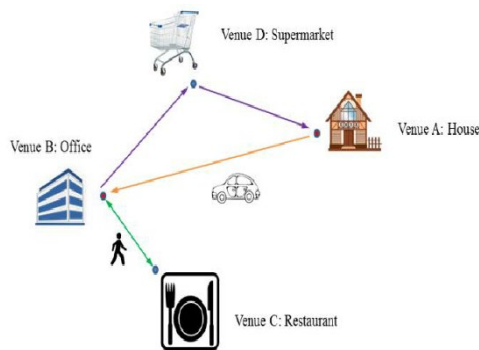
Point-of-interest



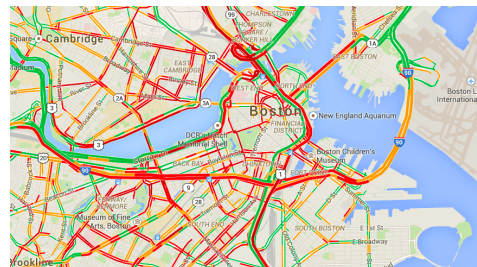
Road network



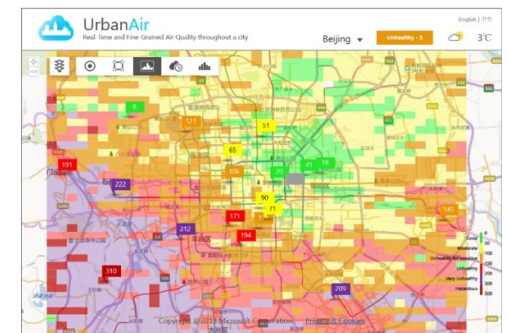
Vehicle trajectories



User mobility



Road traffic



Air quality

Urban Data: Categorization

Spatial Data

- Coordinates (location)
- Fixed

Spatio-temporal Data

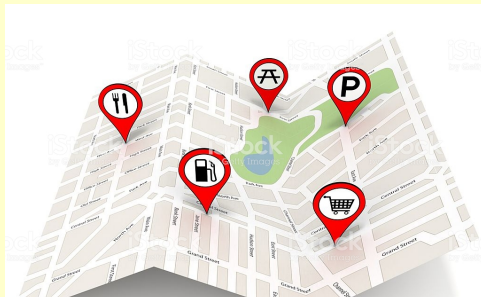
- Coordinates (location)
- Time stamps
- Dynamic

Network Data

- Nodes and edges
- Fixed structure
- Dynamic features

Urban Data: Categorization

Spatial Data



Point-of-interest

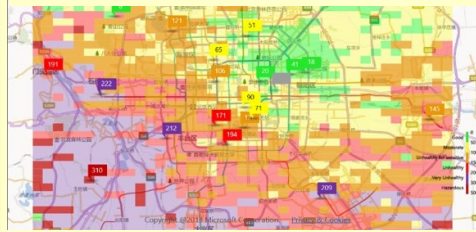


Road network

Spatio-temporal Data



Vehicle trajectories

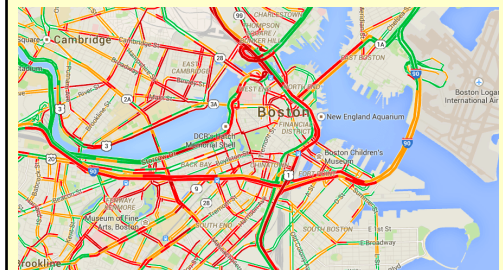


Air quality

Network Data

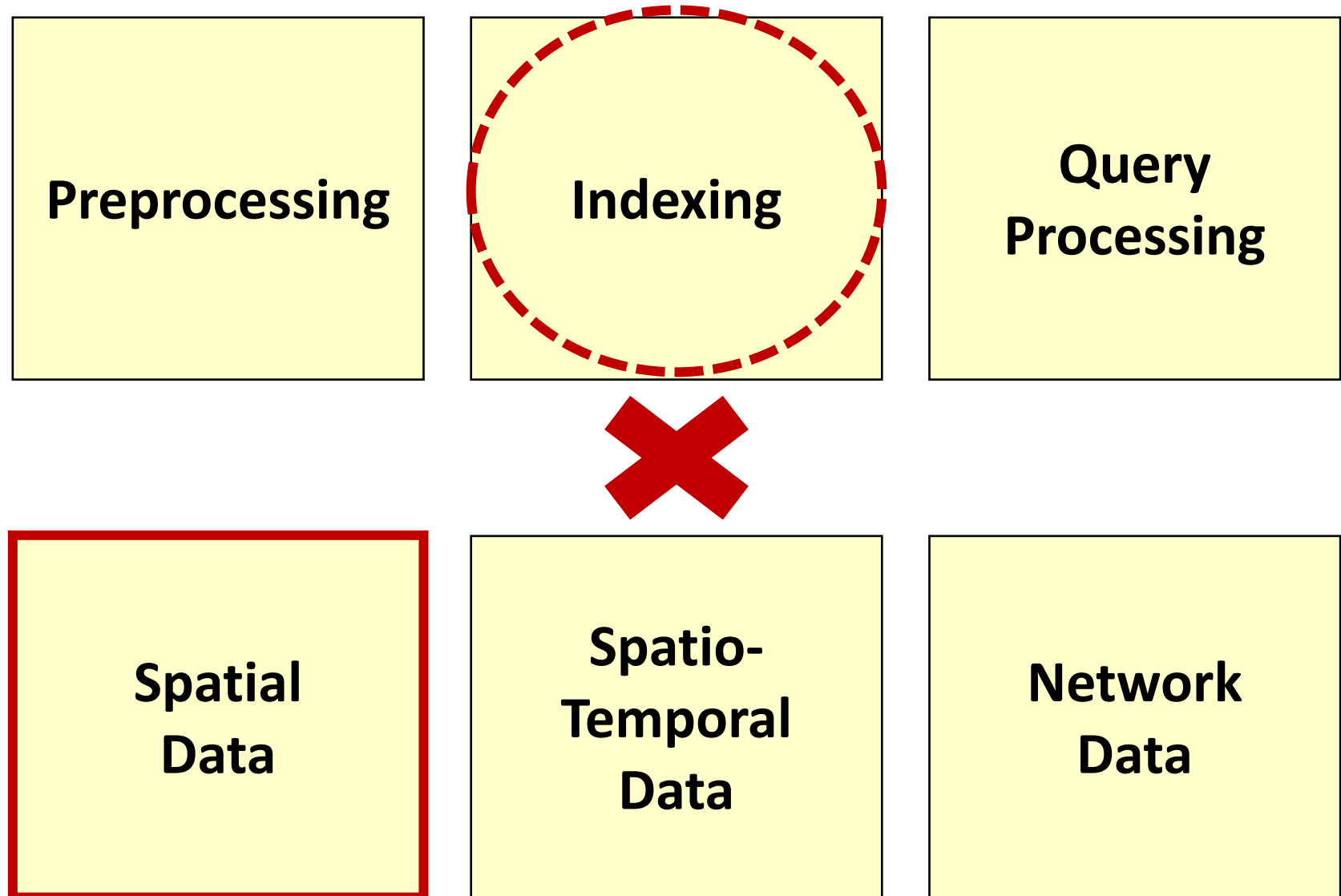


Road network



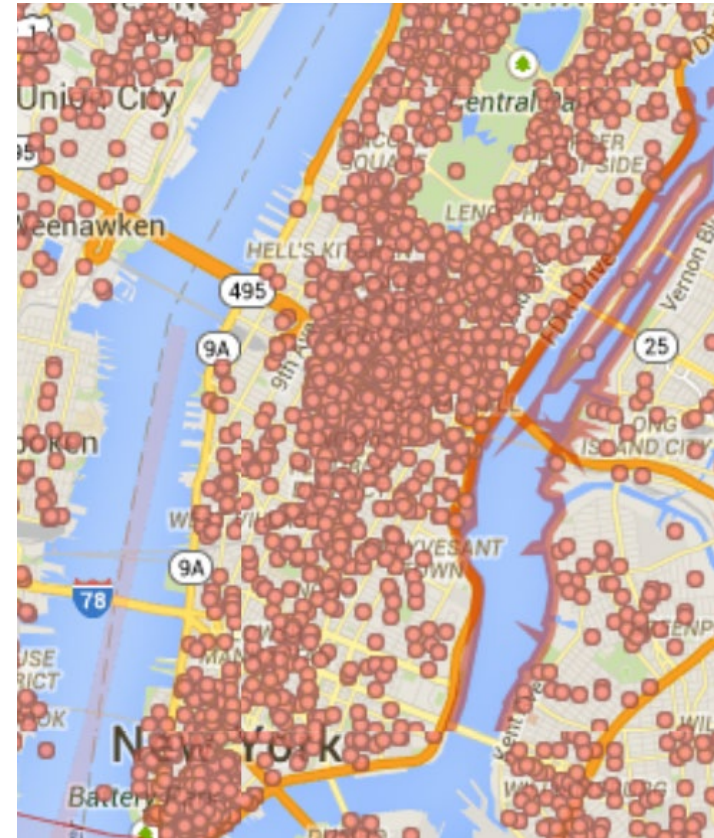
Road traffic

Urban Data Management



Indexing: Motivation

What is the nearest restaurant near my location?



**Point-of-interests at
Manhattan, NYC
(a fraction)**

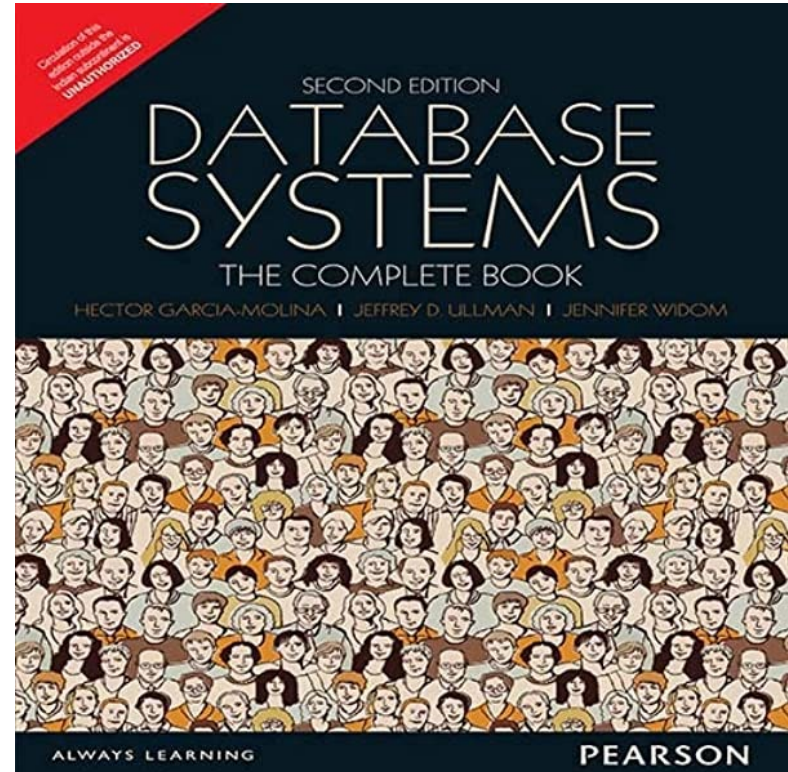
Indexing: Motivation

What is the nearest restaurant near my location?

- A **full scan** would be costly
- **Indexing** => **only a small fraction** is checked

Indexing: Motivation

1. Suppose you want to read the part related to **indexes** in the book
2. How to find that part fast?



Indexing: Motivation

We use the **index**!!

xxvi *TABLE OF CONTENTS*

13.6.5 Pinned Records and Blocks	600
13.6.6 Exercises for Section 13.6	602
13.7 Variable-Length Data and Records	603
13.7.1 Records With Variable-Length Fields	604
13.7.2 Records With Repeating Fields	605
13.7.3 Variable-Format Records	607
13.7.4 Records That Do Not Fit in a Block	608
13.7.5 BLOBs	608
13.7.6 Column Stores	609
13.7.7 Exercises for Section 13.7	610
13.8 Record Modifications	612
13.8.1 Insertion	612
13.8.2 Deletion	614
13.8.3 Update	615
13.8.4 Exercises for Section 13.8	615
13.9 Summary of Chapter 13	615
13.10 References for Chapter 13	617

14 Index Structures 619

14 Index Structures ... 619

14.1 Index-Organized Databases	620
14.1.1 Sequential Files	621
14.1.2 Dense Indexes	621
14.1.3 Sparse Indexes	622
14.1.4 Multiple Levels of Index	623
14.2 The Structure of B-Trees	634
14.2.1 The Structure of B-Trees	634
14.2.2 Applications of B-Trees	637
14.2.3 Lookup in B-Trees	639
14.2.4 Range Queries	639
14.2.5 Insertion Into B-Trees	640
14.2.6 Deletion From B-Trees	642
14.2.7 Efficiency of B-Trees	645
14.2.8 Exercises for Section 14.2	646
14.3 Hash Tables	648
14.3.1 Secondary-Storage Hash Tables	649
14.3.2 Insertion Into a Hash Table	649
14.3.3 Hash-Table Deletion	650
14.3.4 Efficiency of Hash Table Indexes	651
14.3.5 Extensible Hash Tables	652
14.3.6 Insertion Into Extensible Hash Tables	653

Indexing (Spatial Data)

R tree

- Hierarchical collection of rectangles organizing spatial data

k-d tree

- Recursively partition a space based on x and y in an interleaved fashion

Quad tree

- Recursively partition a space into 4 regions evenly

R Tree



The world's most popular open source database



[Contact MySQL](#) | [Login](#) | [Register](#)

[MYSQL.COM](#) [DOWNLOADS](#) [DOCUMENTATION](#) [DEVELOPER ZONE](#)



[MySQL Server](#) [MySQL Enterprise](#) [Workbench](#) [InnoDB Cluster](#) [MySQL NDB Cluster](#) [Connectors](#) [More](#)

MySQL 8.0 Reference Manual / ... / Creating Spatial Indexes

version 8.0

11.4.10 Creating Spatial Indexes

For **InnoDB** and **MyISAM** tables, MySQL can create spatial indexes using syntax similar to that for creating regular indexes, but using the **SPATIAL** keyword. Columns in spatial indexes must be declared **NOT NULL**. The following examples demonstrate how to create spatial indexes:

- With **CREATE TABLE**:

```
1 CREATE TABLE geom (g GEOMETRY NOT NULL SRID 4326, SPATIAL INDEX(g));
```

- With **ALTER TABLE**:

```
1 CREATE TABLE geom (g GEOMETRY NOT NULL SRID 4326);
```

SPATIAL INDEX creates an R-tree index

SPATIAL INDEX creates an R-tree index. For storage engines that support nonspatial indexing of spatial columns, the engine creates a B-tree index. A B-tree index on spatial values is useful for exact-value lookups, but not for range scans.

[Documentation Home](#)

MySQL 8.0 Reference Manual

- [Preface and Legal Notices](#)
- [General Information](#)
- [Installing and Upgrading MySQL](#)
- [Tutorial](#)
- [MySQL Programs](#)
- [MySQL Server Administration](#)
- [Security](#)
- [Backup and Recovery](#)
- [Optimization](#)
- [Language Structure](#)
- [Character Sets, Collations, Unicode](#)
- [Data Types](#)
 - [Numeric Data Types](#)
 - [Date and Time Data Types](#)
 - [String Data Types](#)
 - [Spatial Data Types](#)
 - [Spatial Data Types](#)
 - [The OpenGIS Geometry Model](#)



R Tree

Spatial and Graph Developer's Guide

←

🏠 < >

Page 15

🔍

☐ This Book ☐ This Release

Table of Contents

🔍 Oracle Spatial and Graph Developer's Guide

Preface

Changes in This Release for Oracle Spatial and Graph Developer's Guide

Part I Conceptual and Usage Information

🔍 **Spatial Concepts**

🔍 Spatial Data Types and Metadata

🔍 SQL Multimedia Type Support

🔍 Loading Spatial Data

🔍 Indexing and Querying Spatial Data

🔍 Coordinate Systems (Spatial Reference Systems)

🔍 Linear Referencing System

🔍 Spatial Analysis and Mining

🔍 Extending Spatial Indexing Capabilities

Part II Spatial Web Services

🔍 Introduction to Spatial Web Services

🔍 Geocoding Address Data

1.7 Indexing of Spatial Data

The introduction of spatial indexing capabilities into the Oracle database engine is a key feature of the Spatial and Graph product. A spatial index, like any other index, provides a mechanism to limit searches, but in this case the mechanism is based on spatial criteria such as intersection and containment. A spatial index is needed to:

- Find objects within an indexed data space that interact with a given point or area of interest (window query)
- Find pairs of objects from within two indexed data spaces that interact spatially with each other (spatial join)

Testing of spatial indexes with many workloads and operators is ongoing, and results and recommendations will be documented as they become available.

The following sections explain the concepts and options associated with R-tree indexing.

1.7.1 R-Tree Indexing

A spatial R-tree index can index spatial data of up to four dimensions. An R-tree index approximates each geometry by a single rectangle that minimally encloses the geometry (called the minimum bounding rectangle, or MBR), as shown in [Figure 1-3](#).

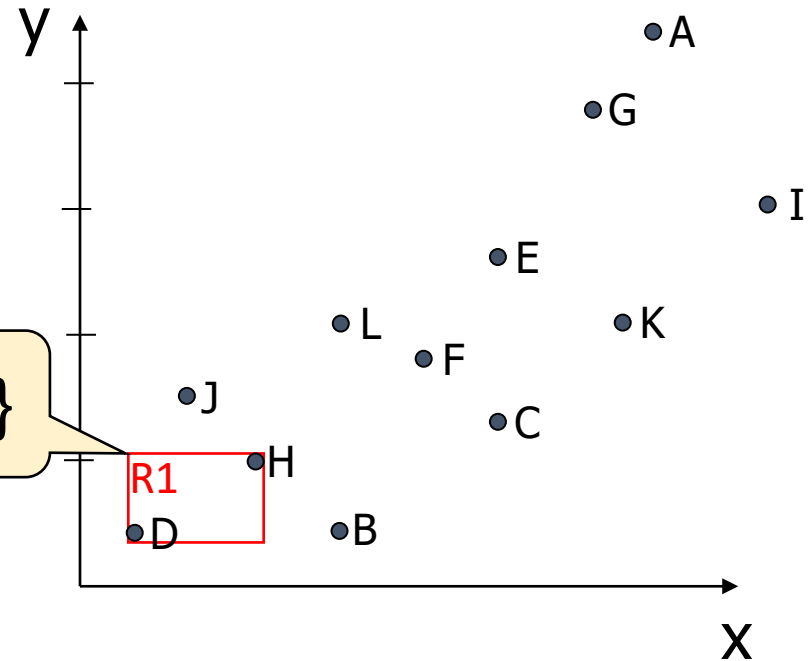
Figure 1-3 MBR

R-Tree Indexing

R Tree

- Group points D and H
- Obtain a **minimum bounding rectangle** (MBR)

{D, H}



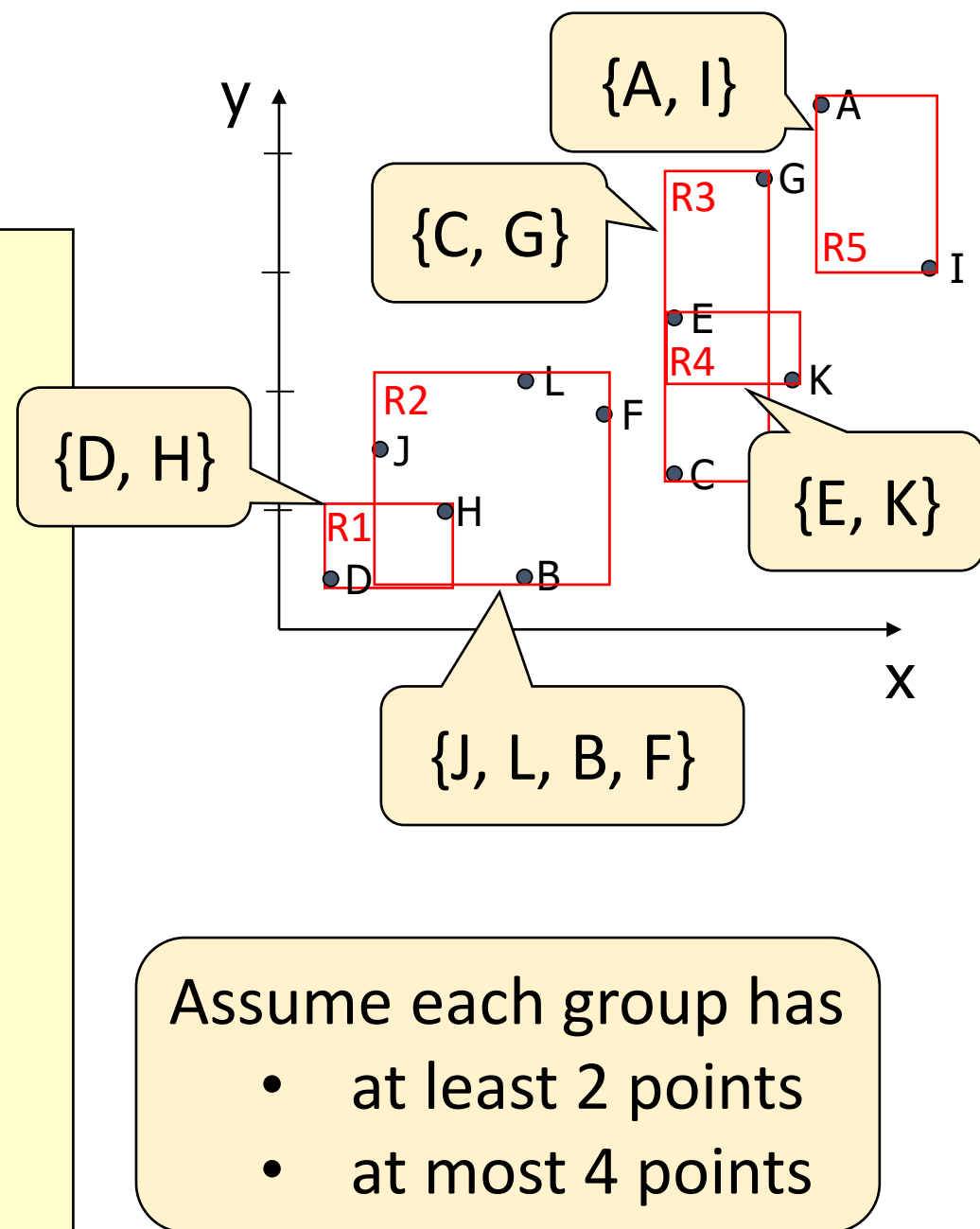
Settings => Balanced

Assume each group has

- at least 2 points
- at most 4 points

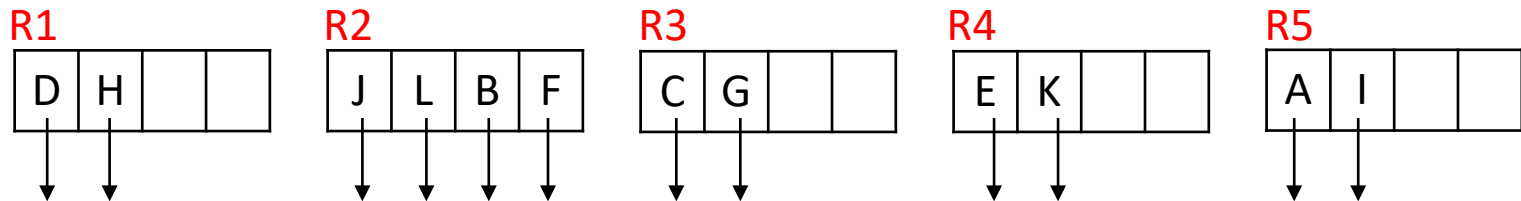
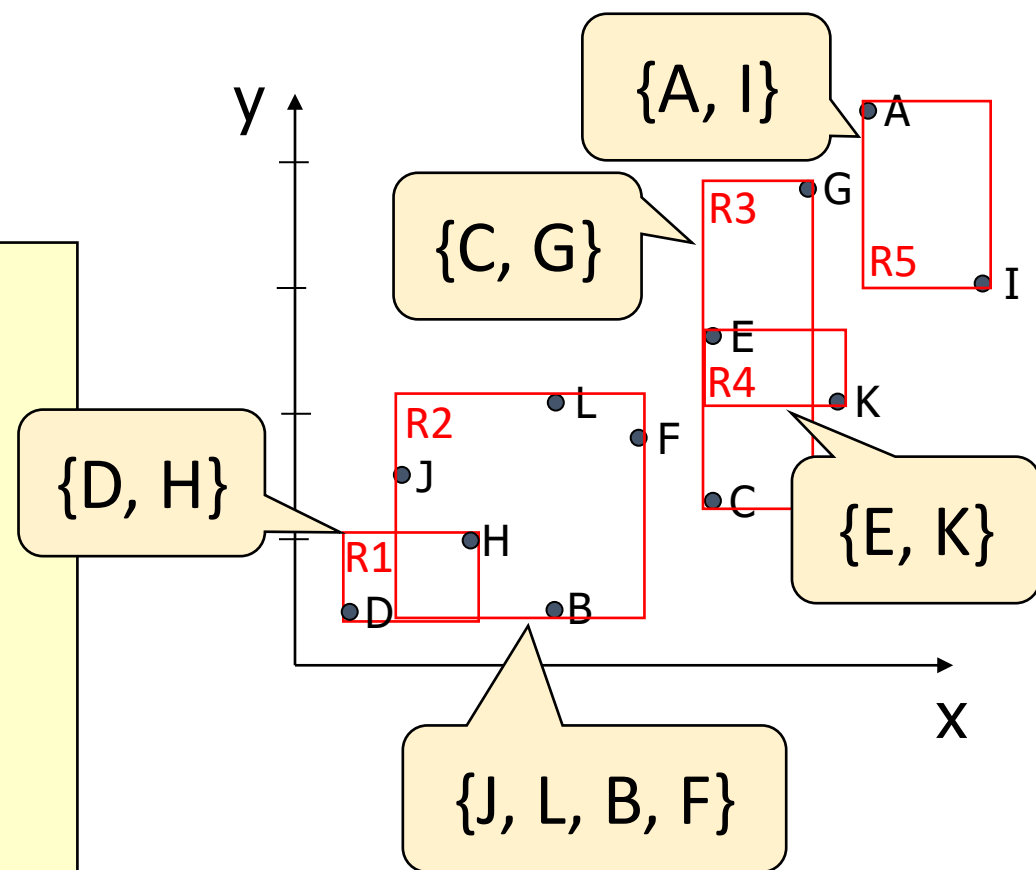
R Tree

- Similarly, group other points
- Obtain corresponding MBRs



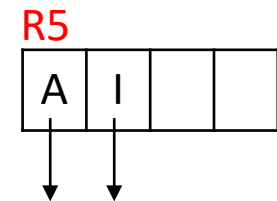
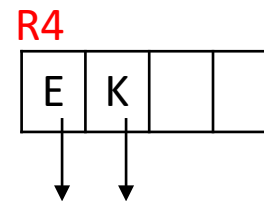
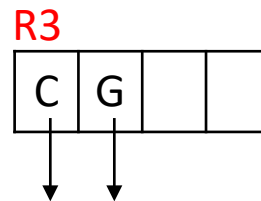
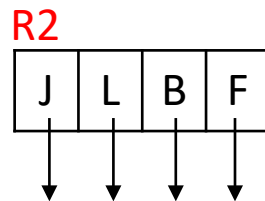
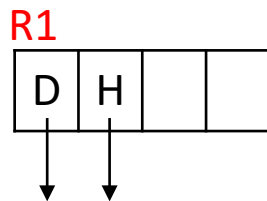
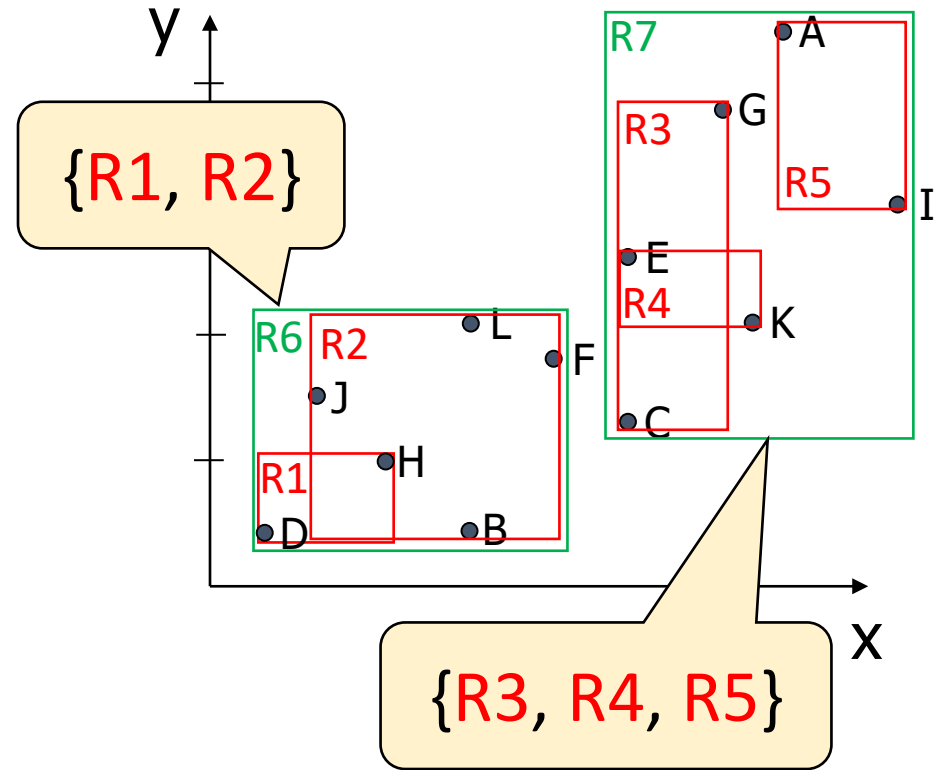
R Tree

- Create a node for each group
- Each node corresponds to an **array** of pairs of **<point, pointer>**



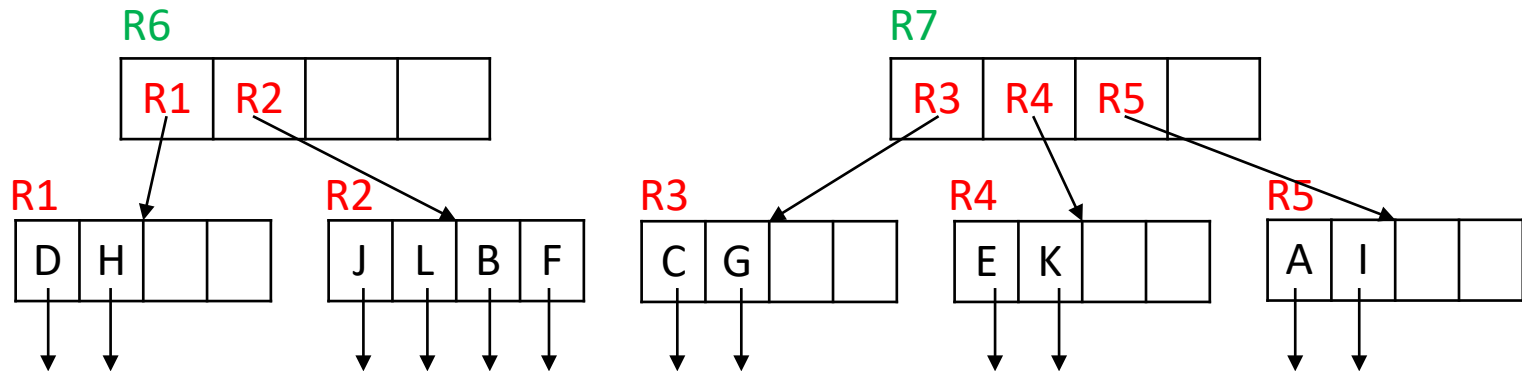
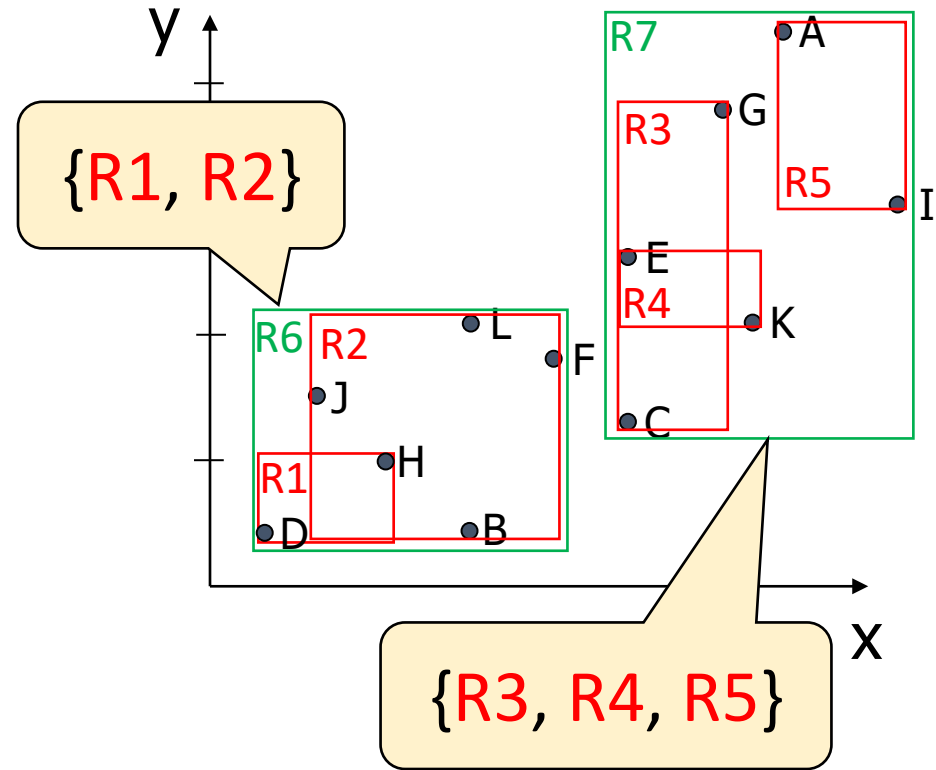
R Tree

- Recursively, group the MBRs
- Obtain larger MBRs



R Tree

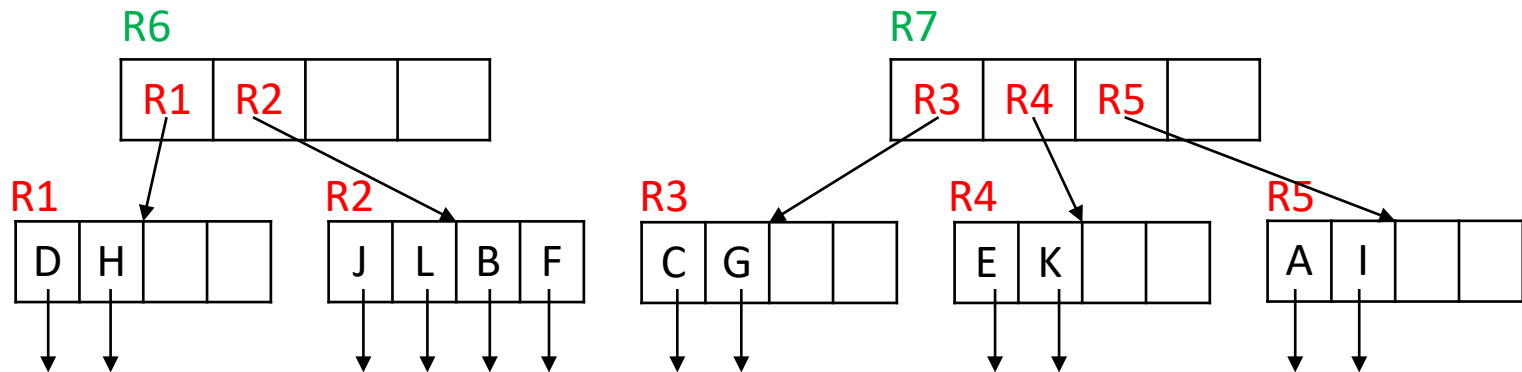
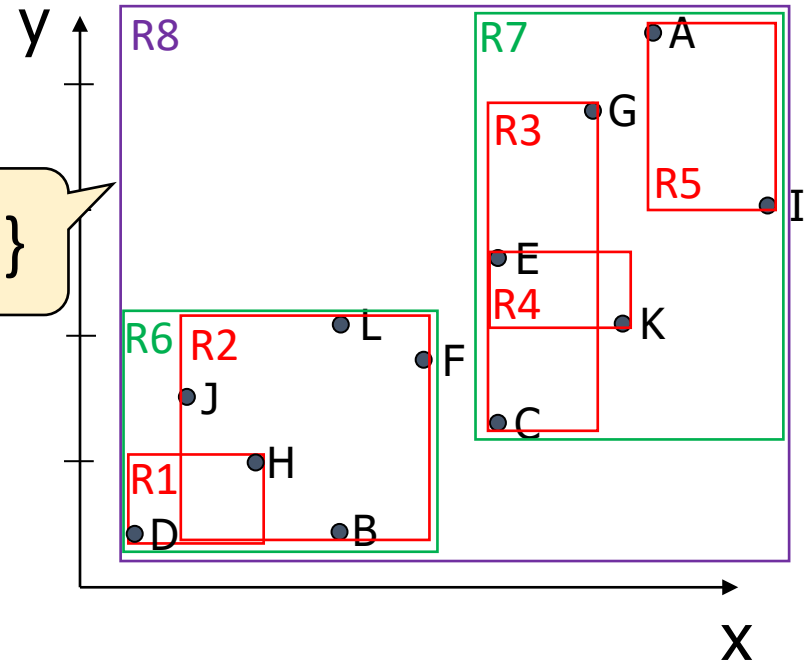
- Create a node for each group
- Each node corresponds to an **array** of pairs of $\langle \text{MBR}, \text{pointer} \rangle$



R Tree

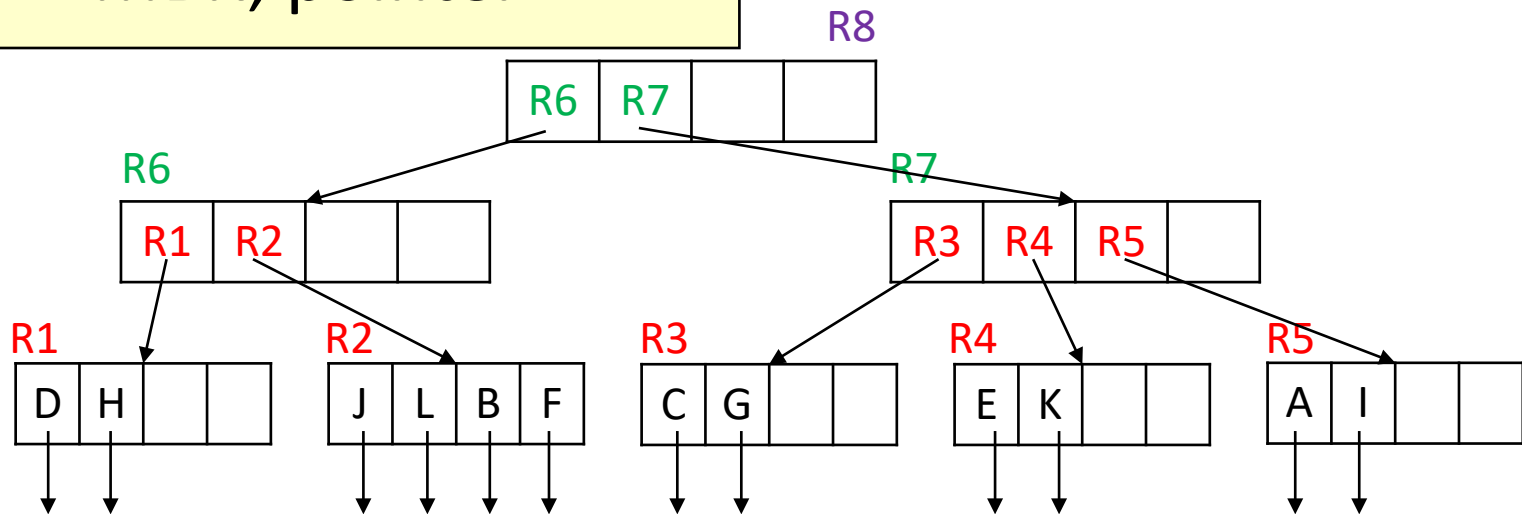
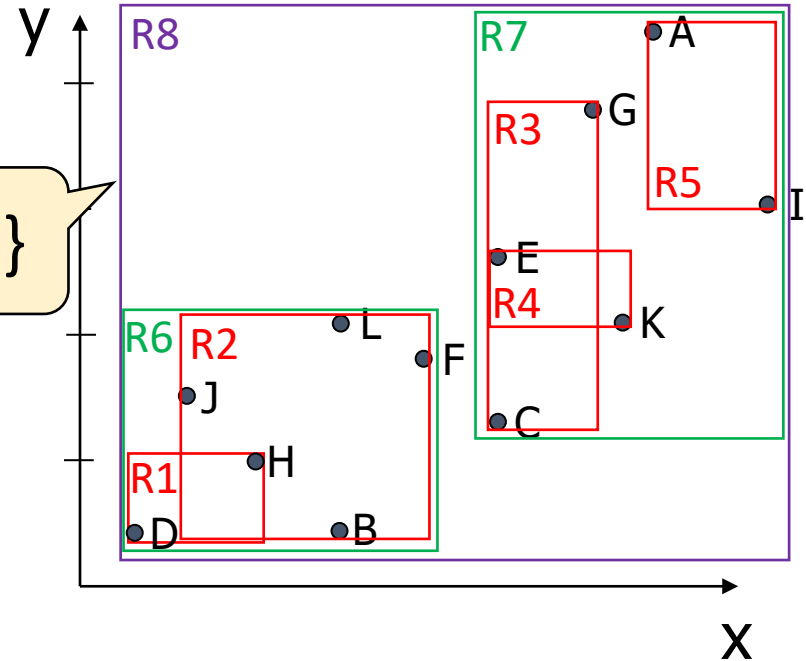
- Recursively, group the MBRs
- Obtain larger MBRs

{R6, R7}

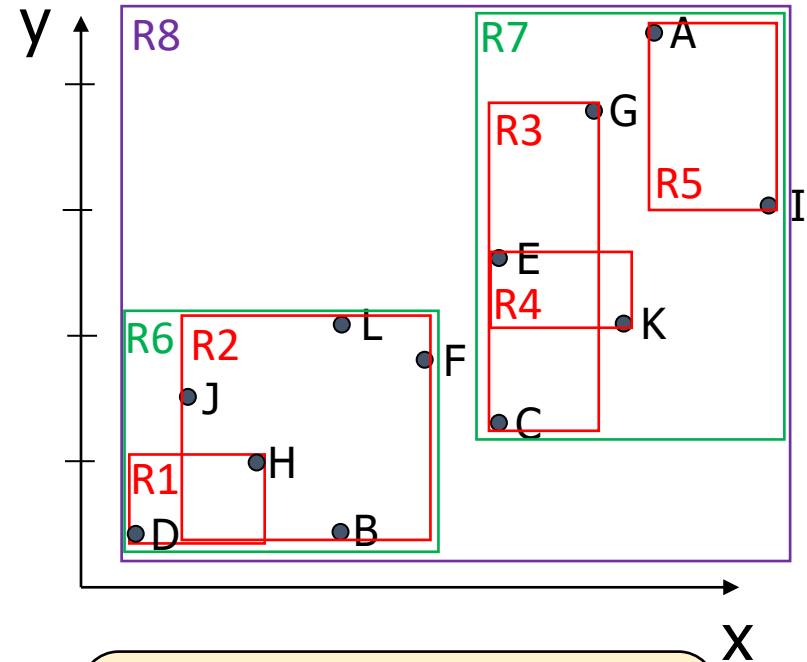
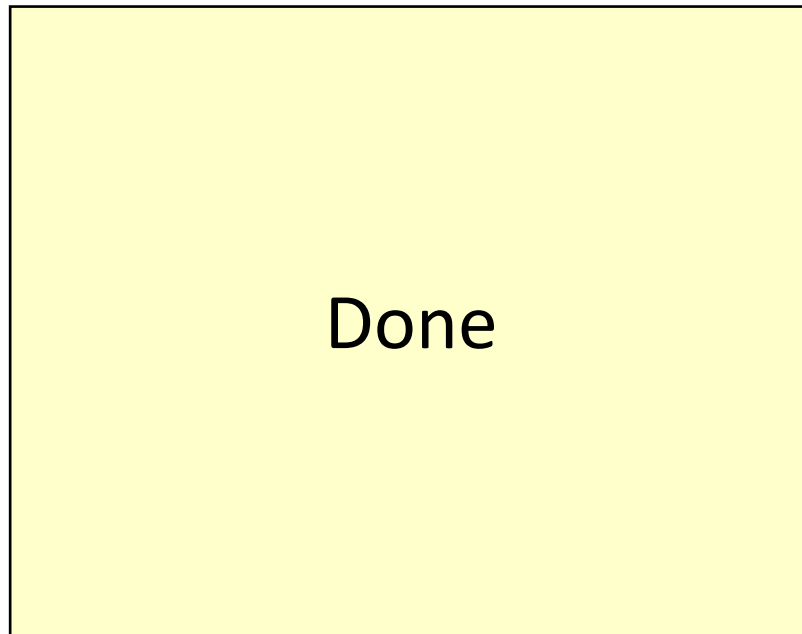


R Tree

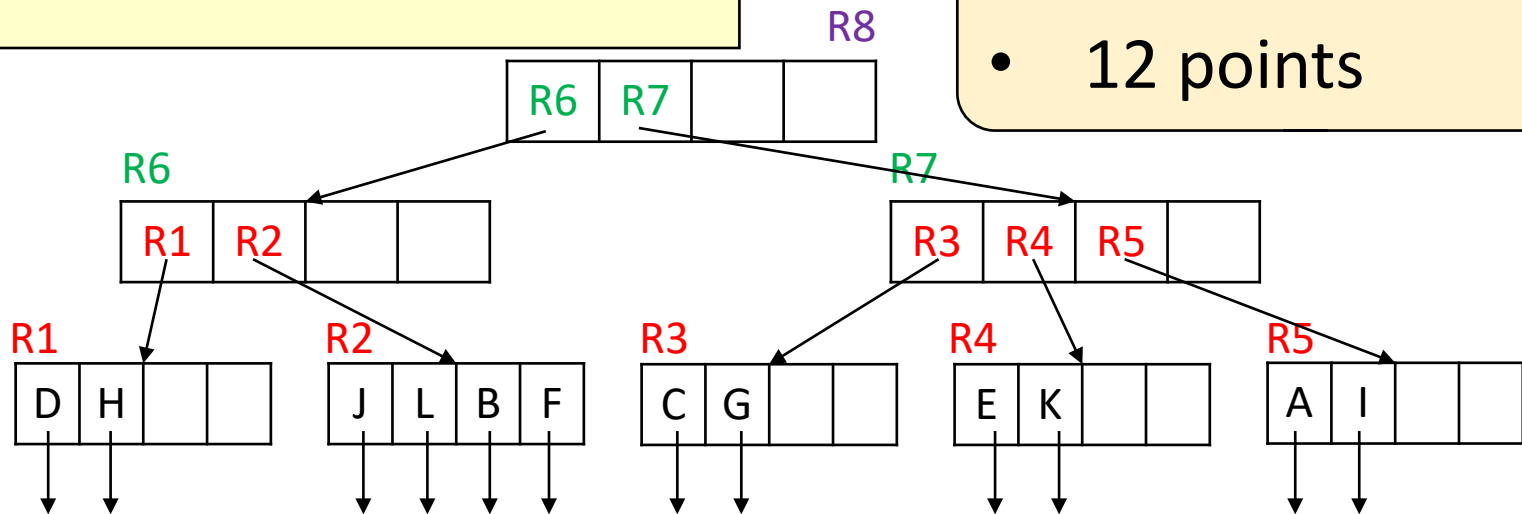
- Create a node for $\{R6, R7\}$
- Each node corresponds to an **array** of pairs of $\langle \text{MBR}, \text{pointer} \rangle$



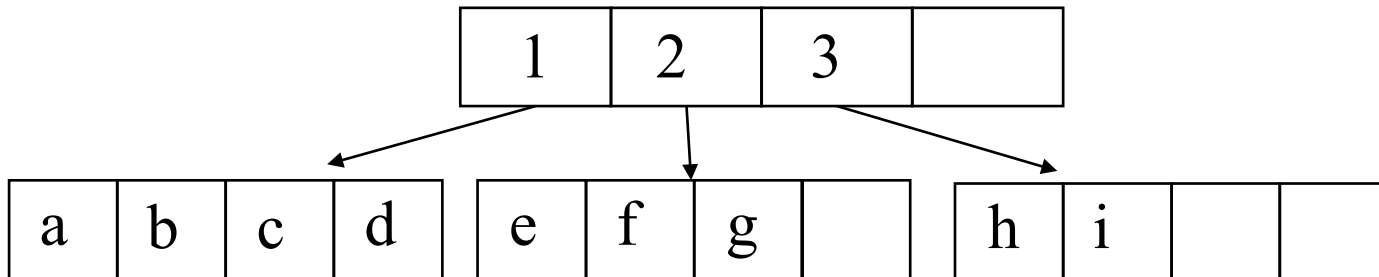
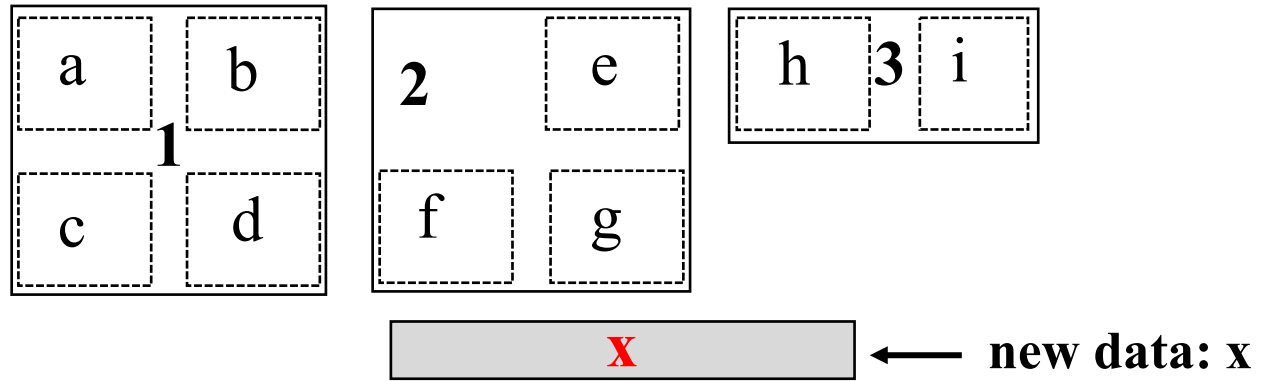
R Tree



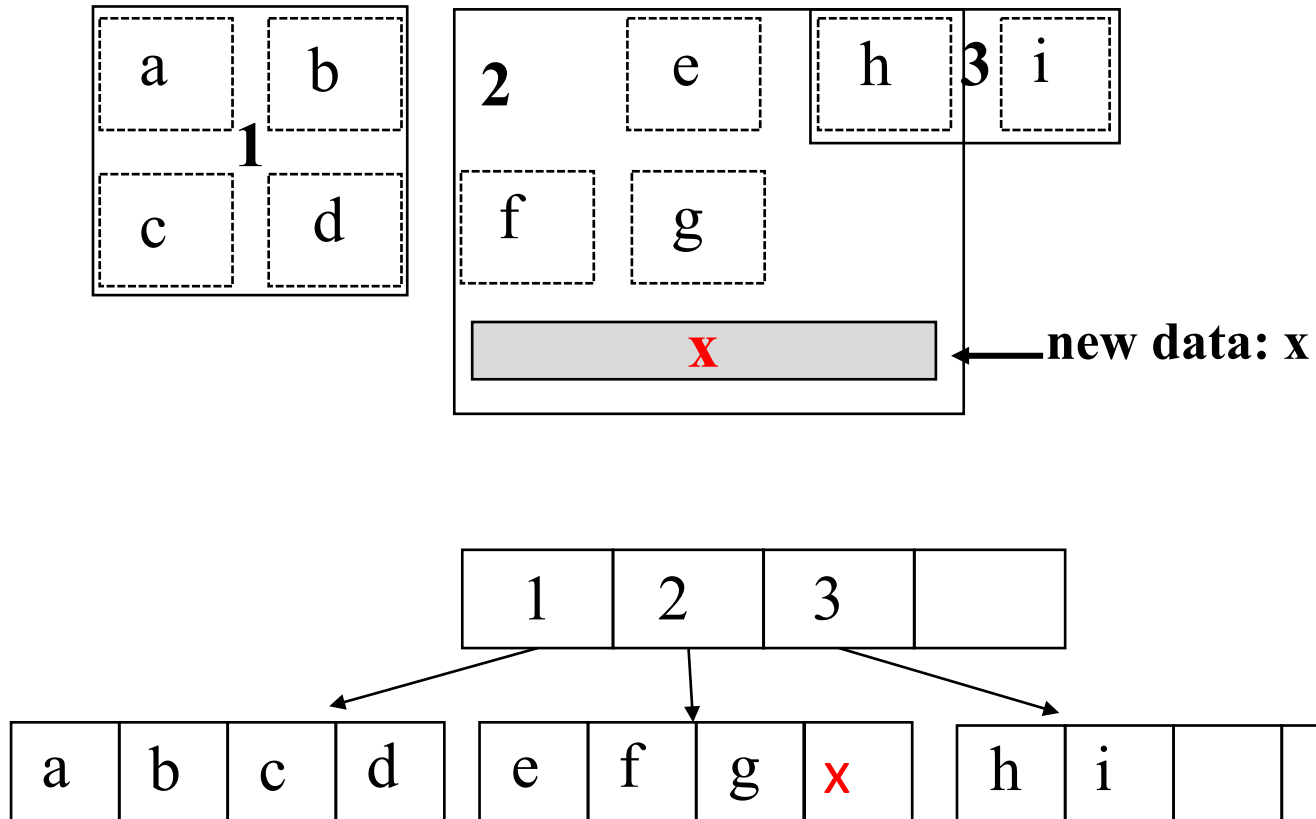
- 8 nodes
- 12 points



R Tree: Insertion



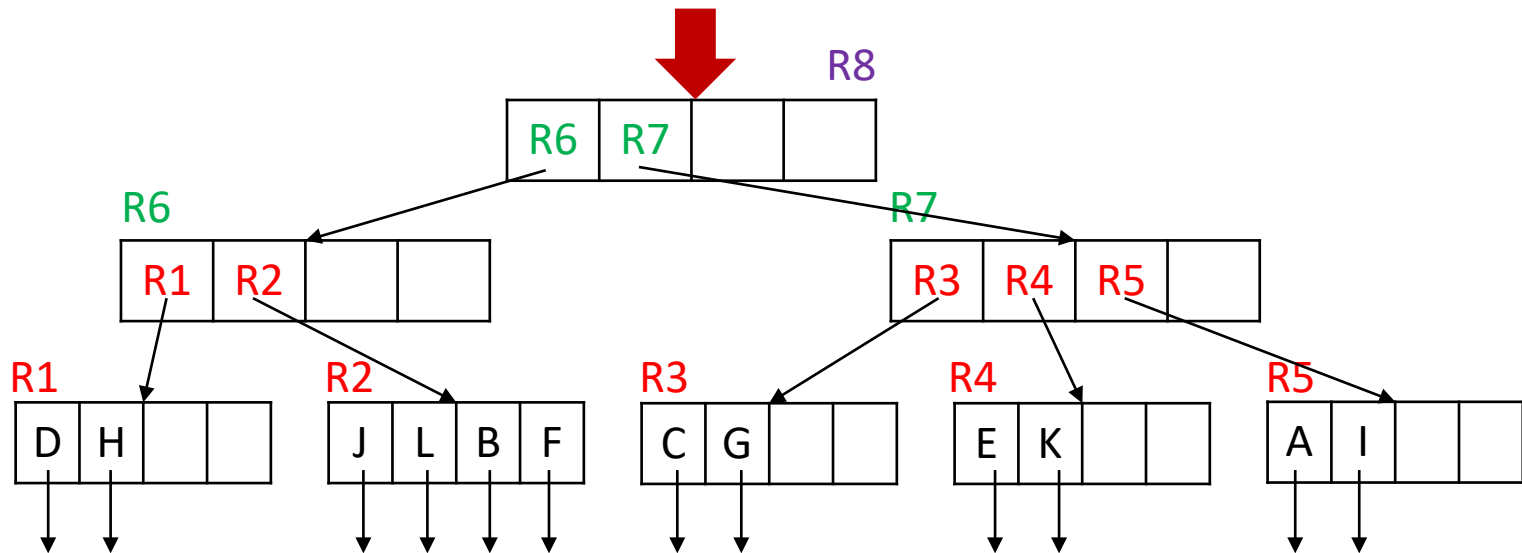
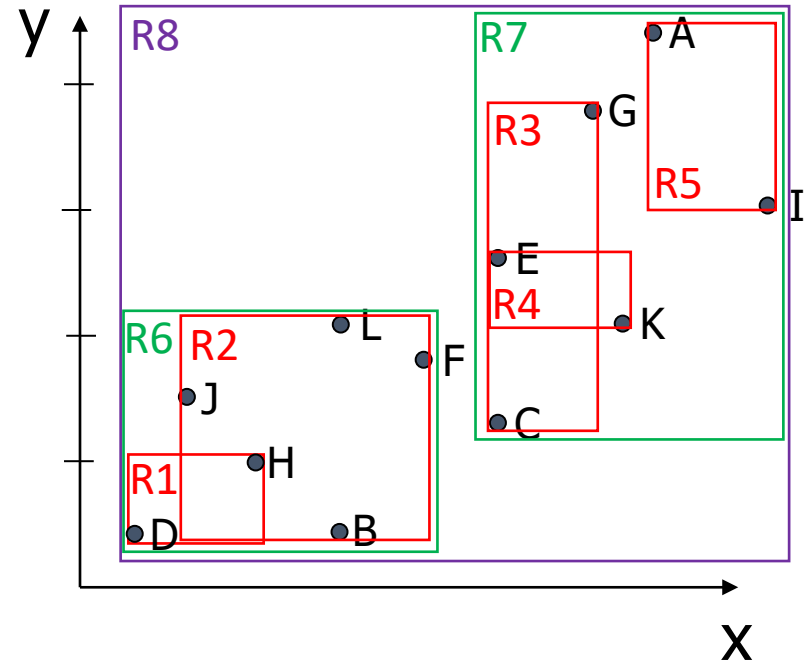
R Tree: Insertion



R Tree: Query Processing

Search point K

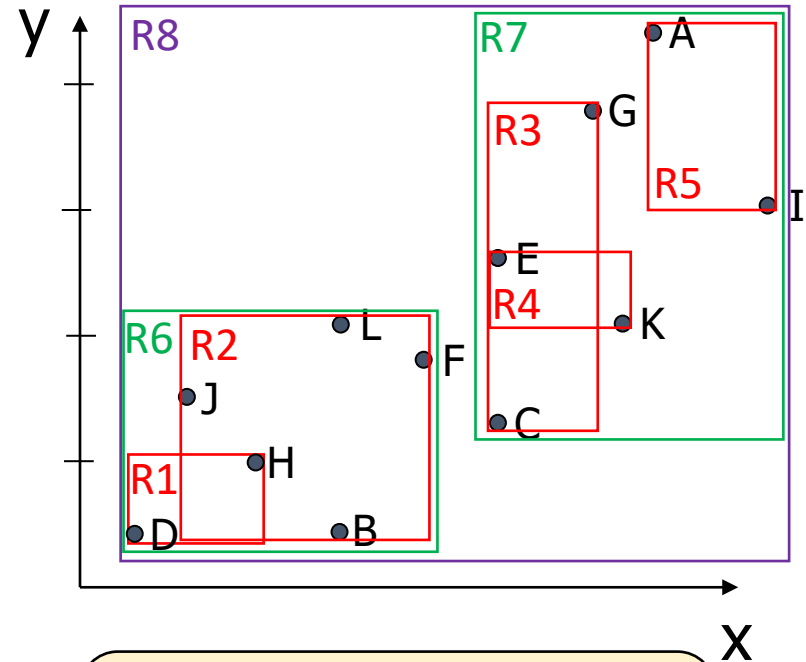
1. Check if the root node intersects K, and if so, proceed



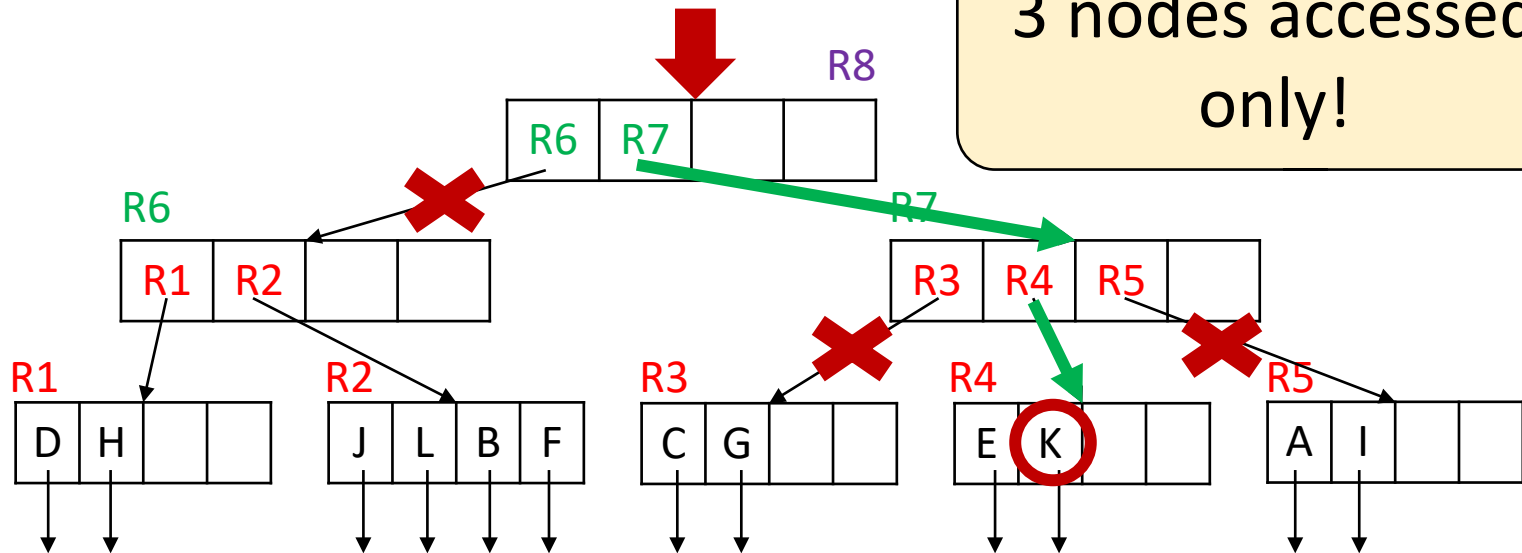
R Tree: Query Processing

Search point K

2. Enumerate all children with the MBR intersecting K



3 nodes accessed only!



Indexing (Spatial Data)

R tree

- Hierarchical collection of rectangles organizing spatial data

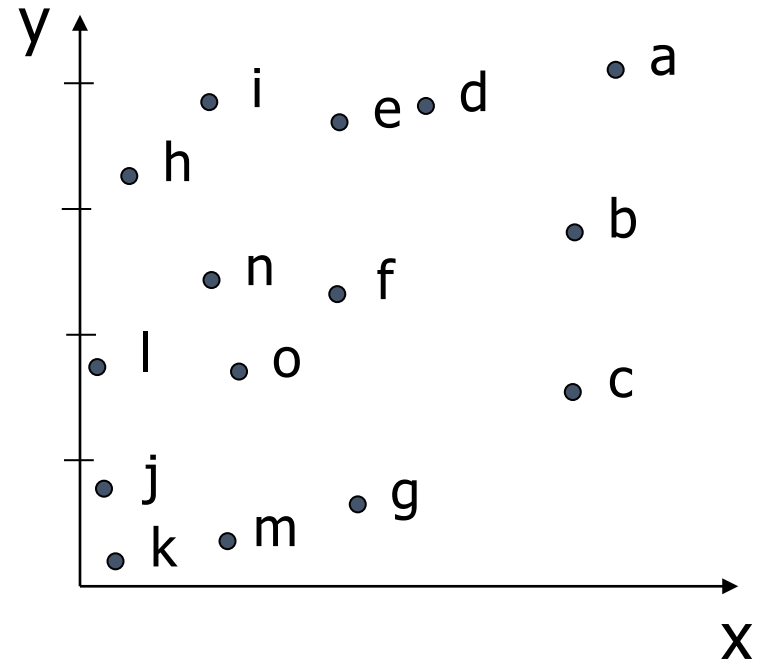
k-d tree

- Recursively partition a space based on x and y in an interleaved fashion

Quad tree

- Recursively partition a space into 4 regions evenly

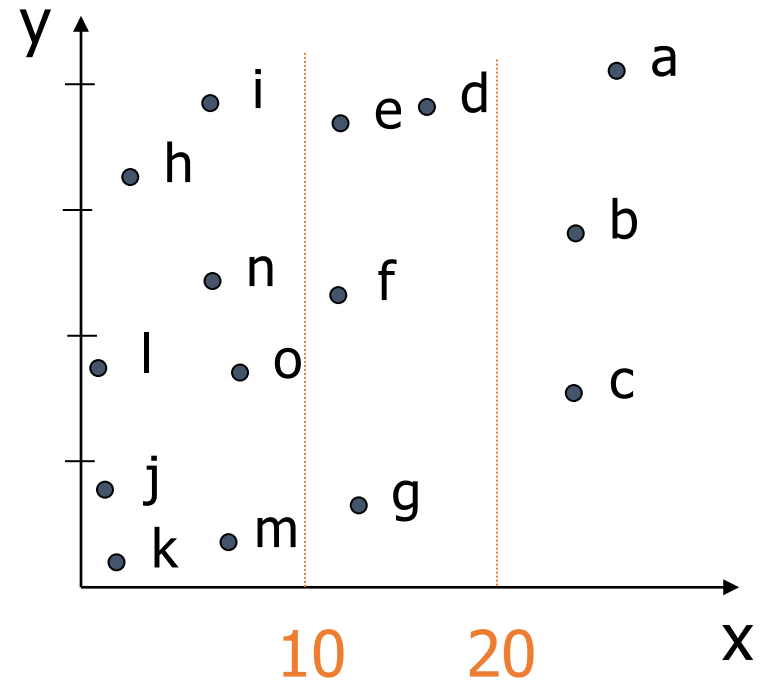
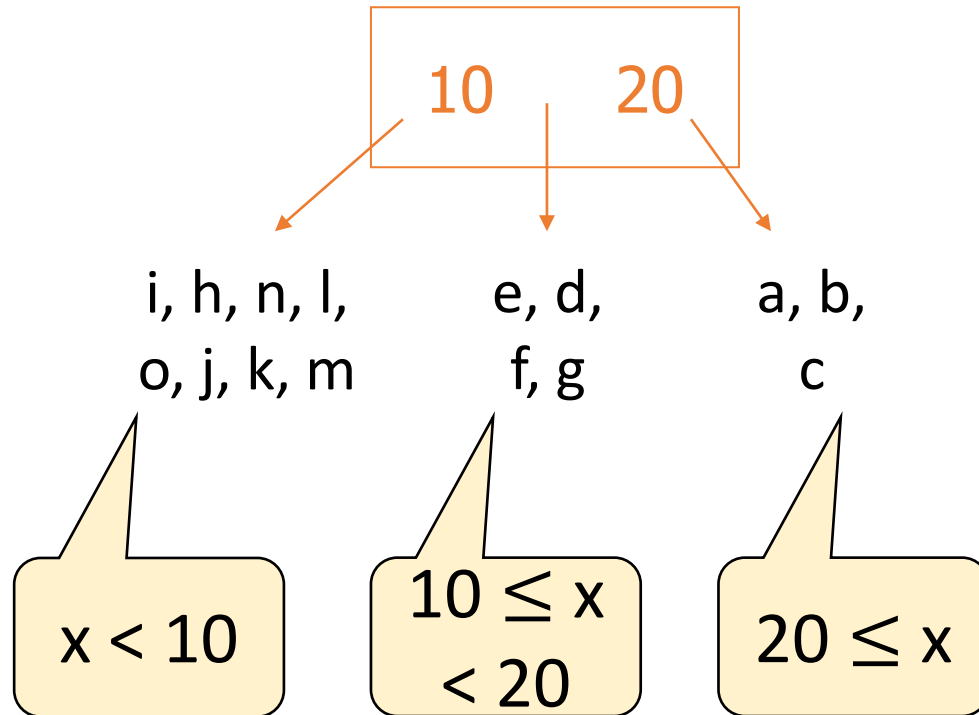
k-d Tree



Input data

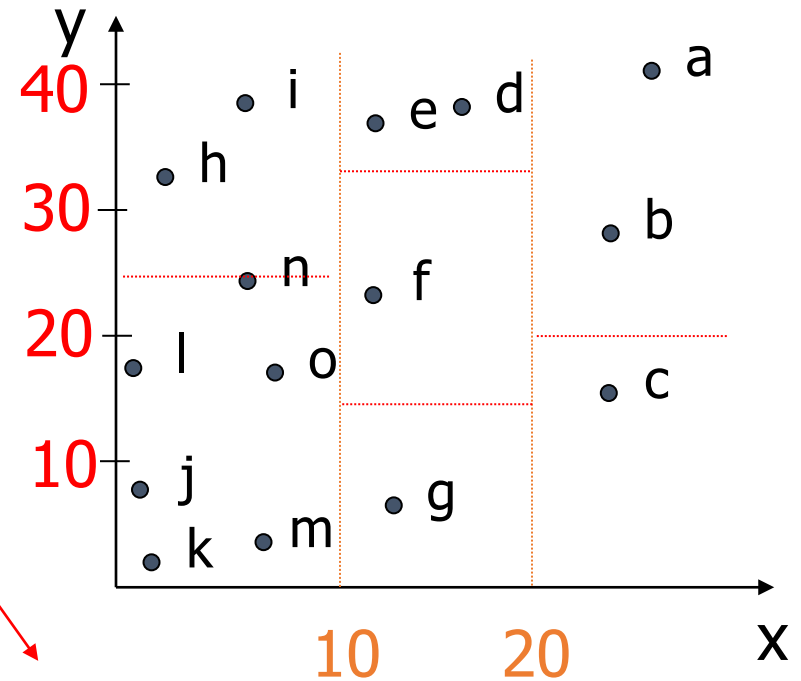
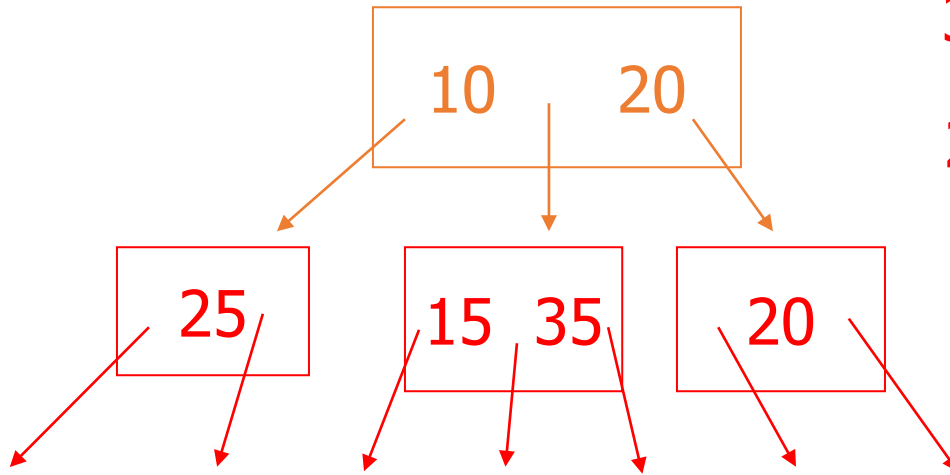
15 records

k-d Tree



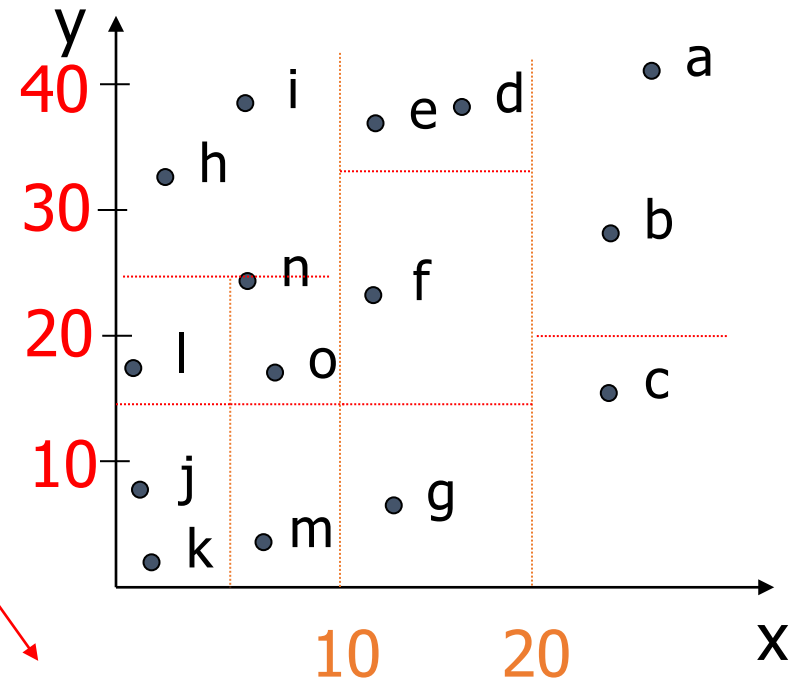
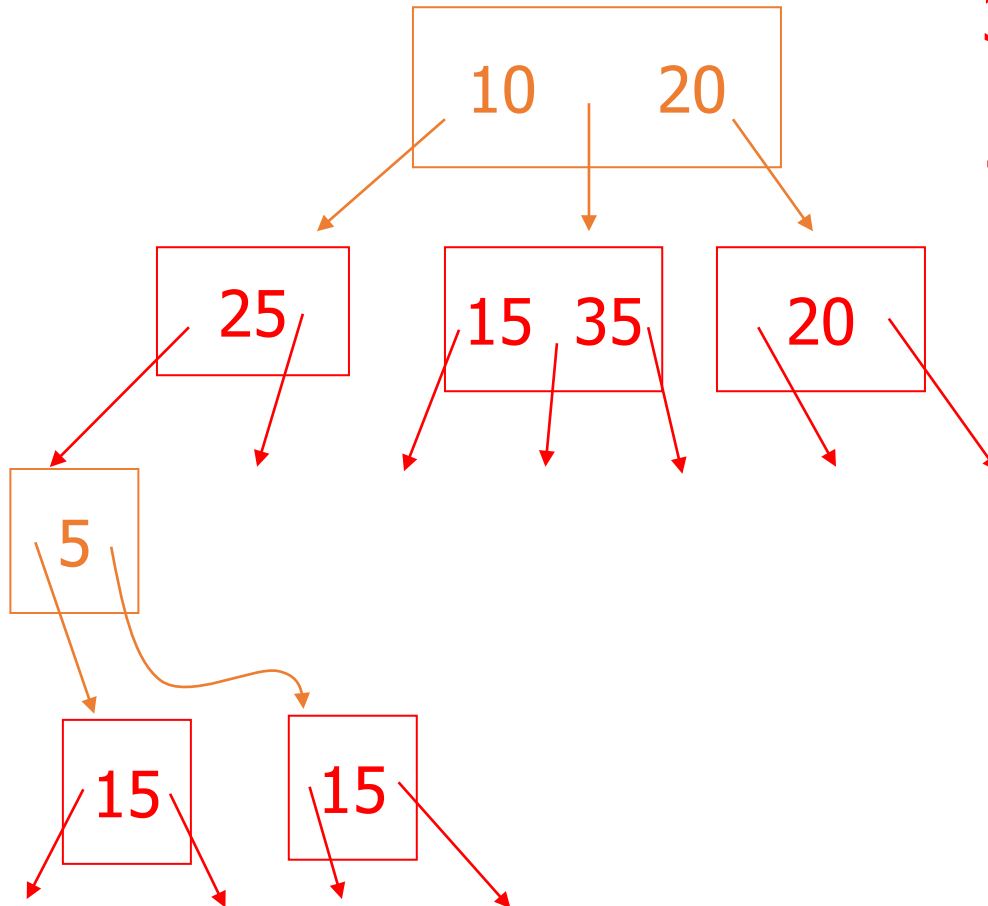
Build an index on x

k-d Tree



Build an index on y
(for each partition
of data on x)

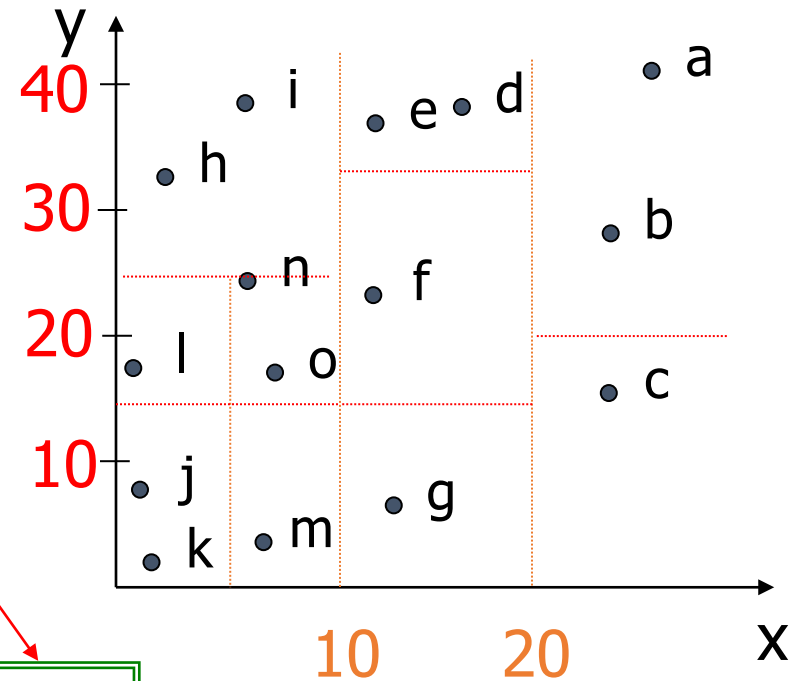
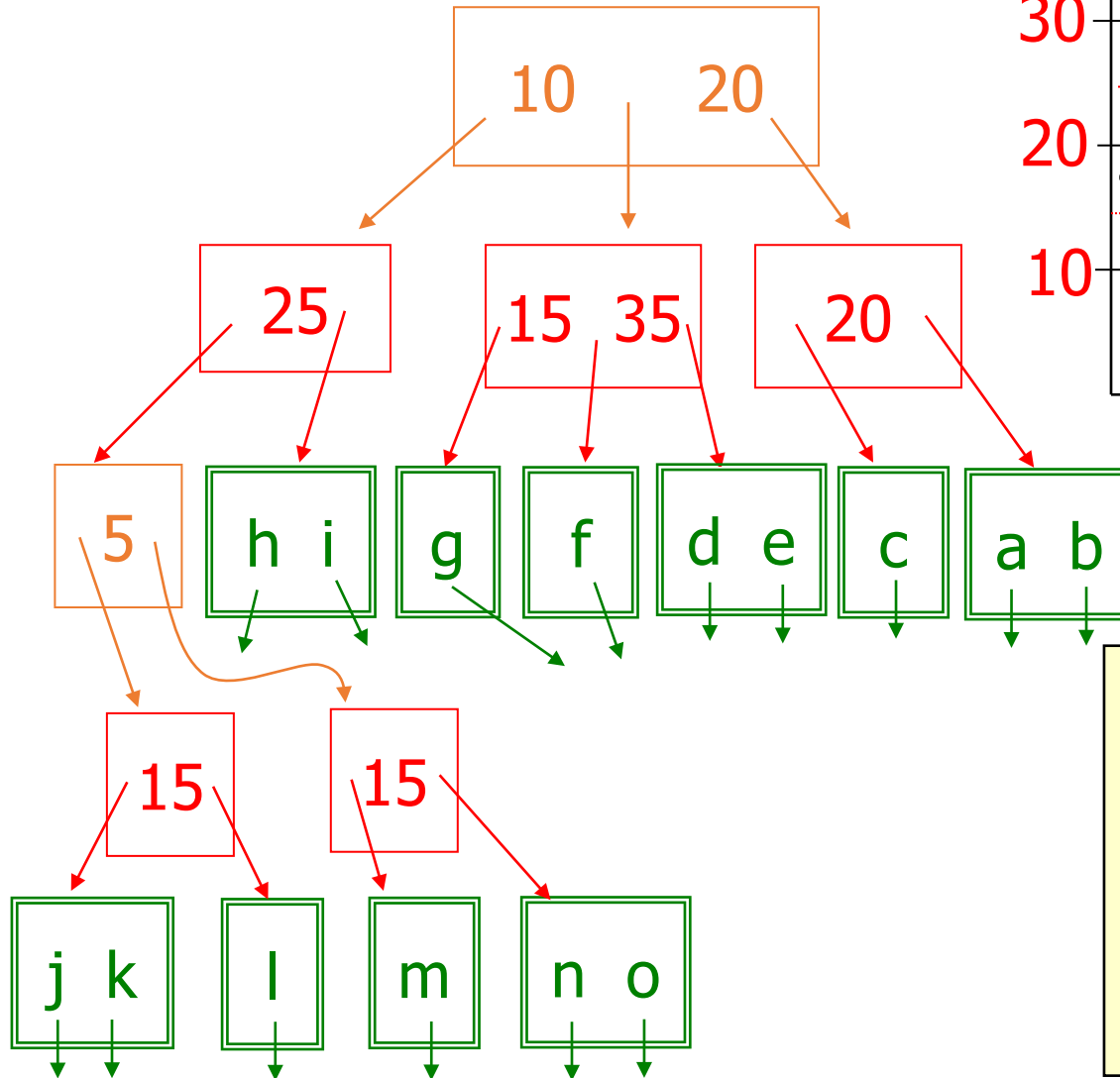
k-d Tree



Build indexes on y
and x iteratively

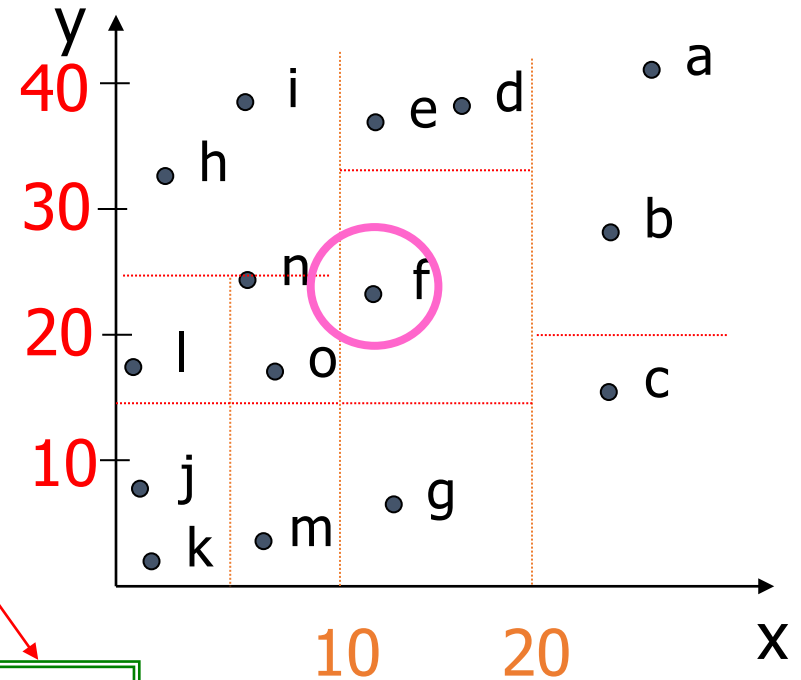
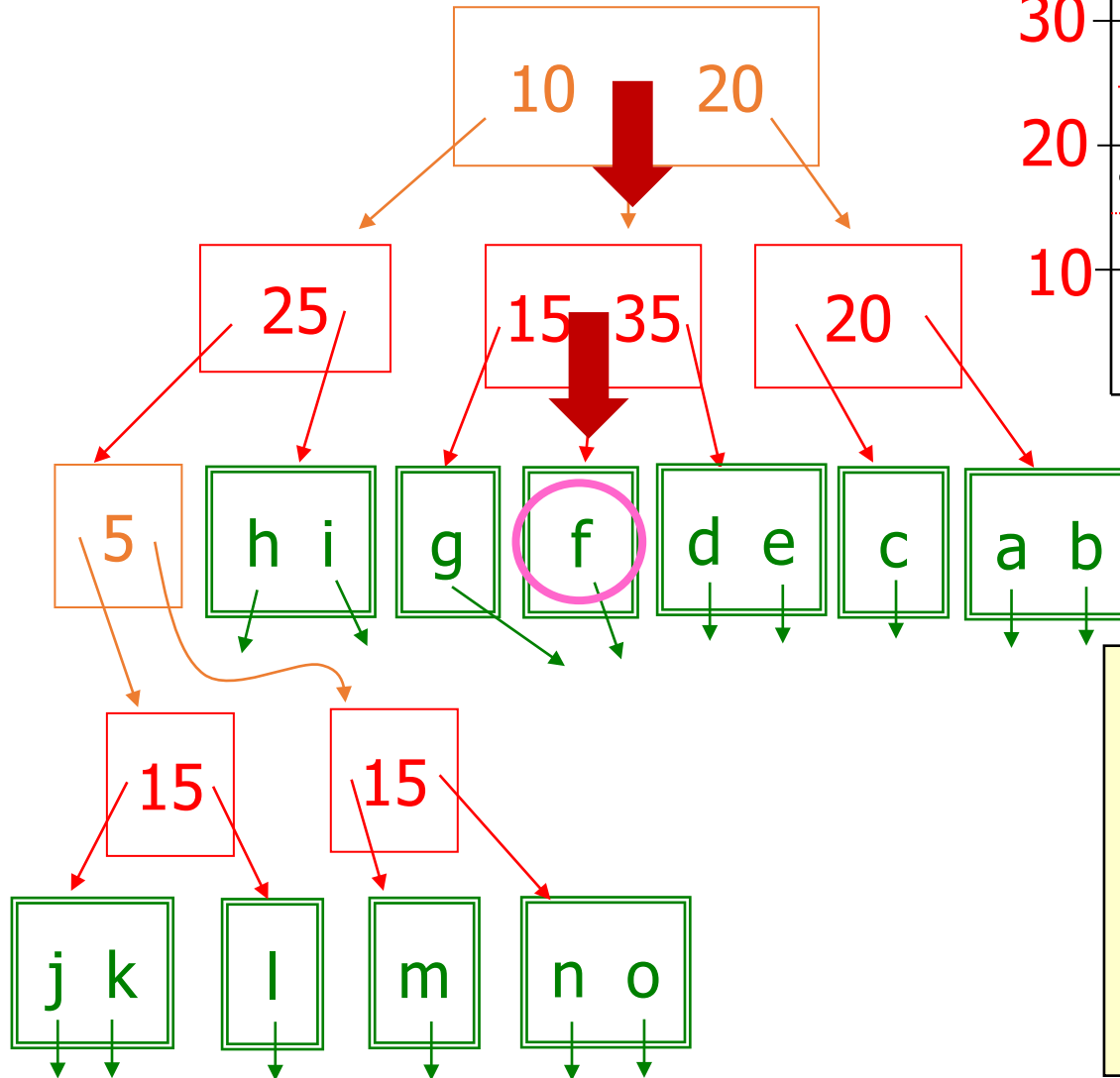
until each partition
involves at most 2
records

k-d Tree



Index at the end

k-d Tree



Search the point f
<12, 24>

Indexing (Spatial Data)

R tree

- Hierarchical collection of rectangles organizing spatial data

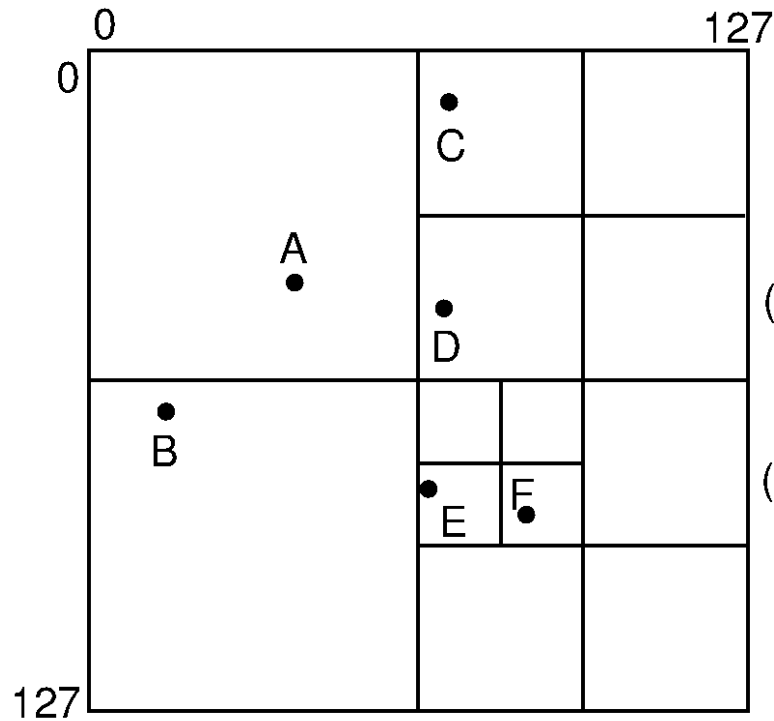
k-d tree

- Recursively partition a space based on x and y in an interleaved fashion

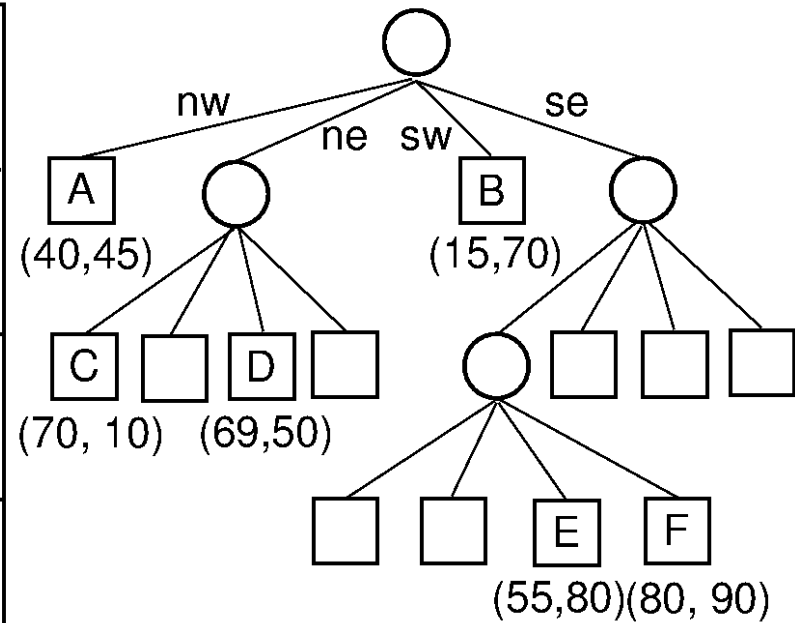
Quad tree

- Recursively partition a space into 4 regions evenly

Quad Tree



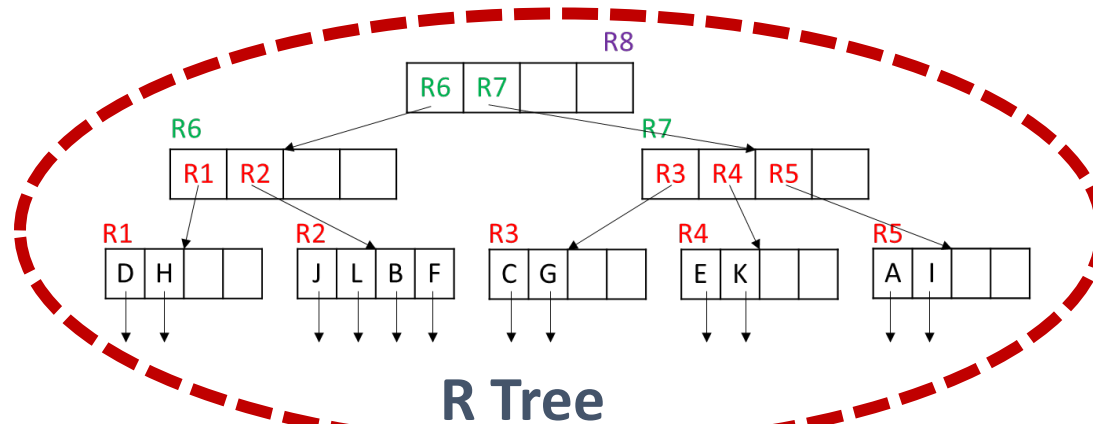
(a)



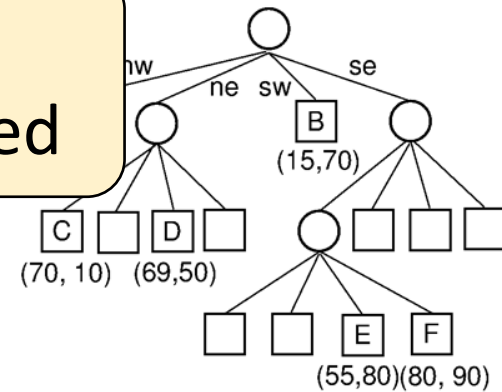
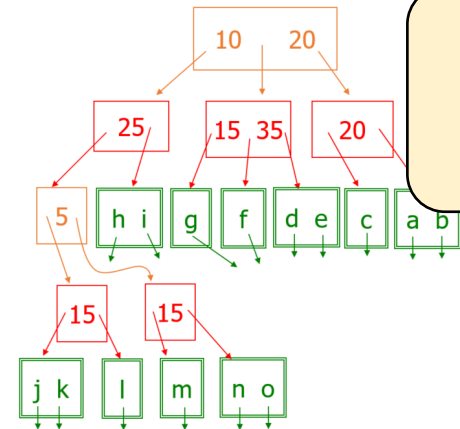
(b)

Source: https://opensa-server.cs.vt.edu/ODSA/Books/CS3/html/_images/PRexamp.png

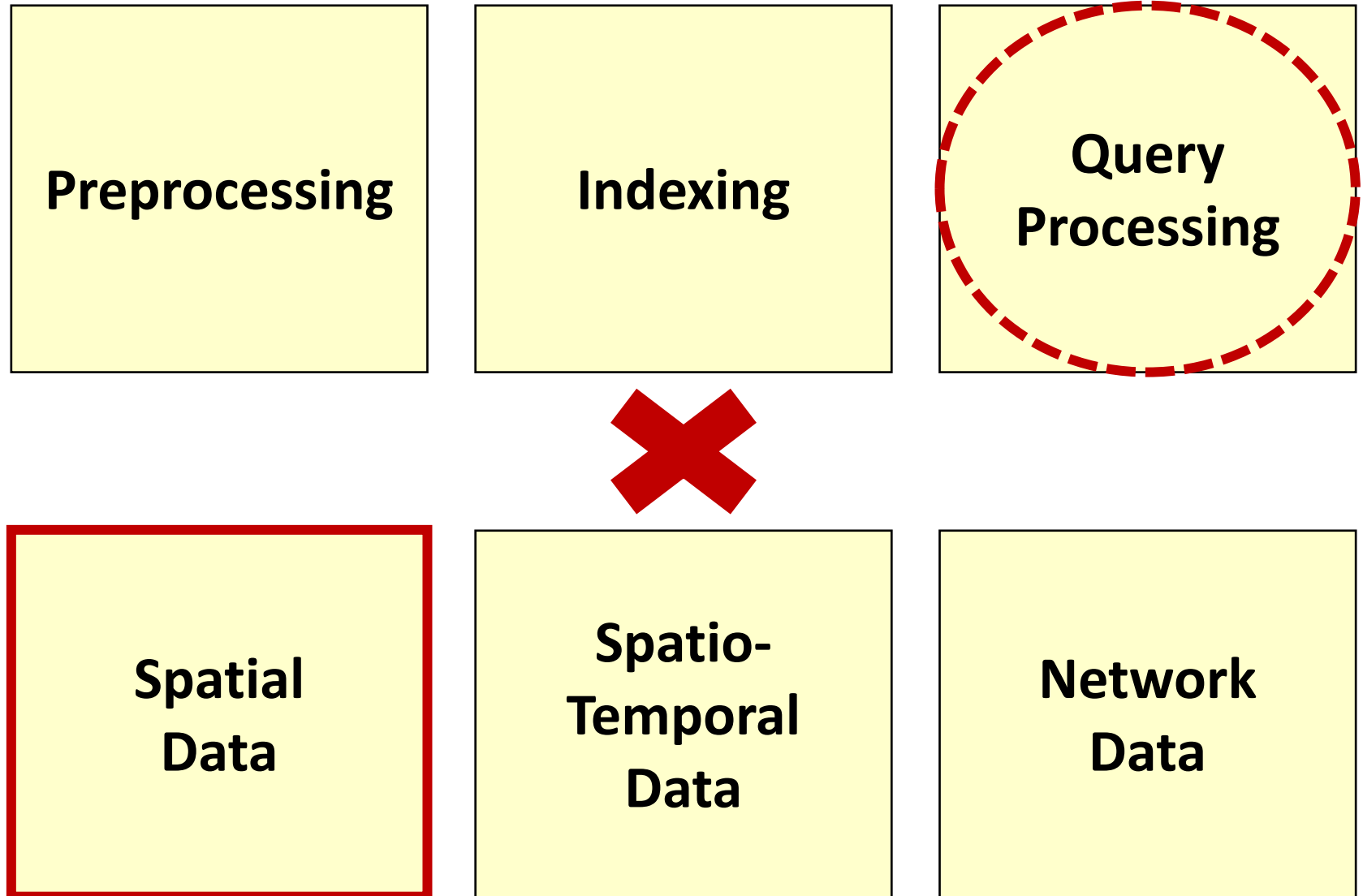
Indexing (Spatial Data): Summarization



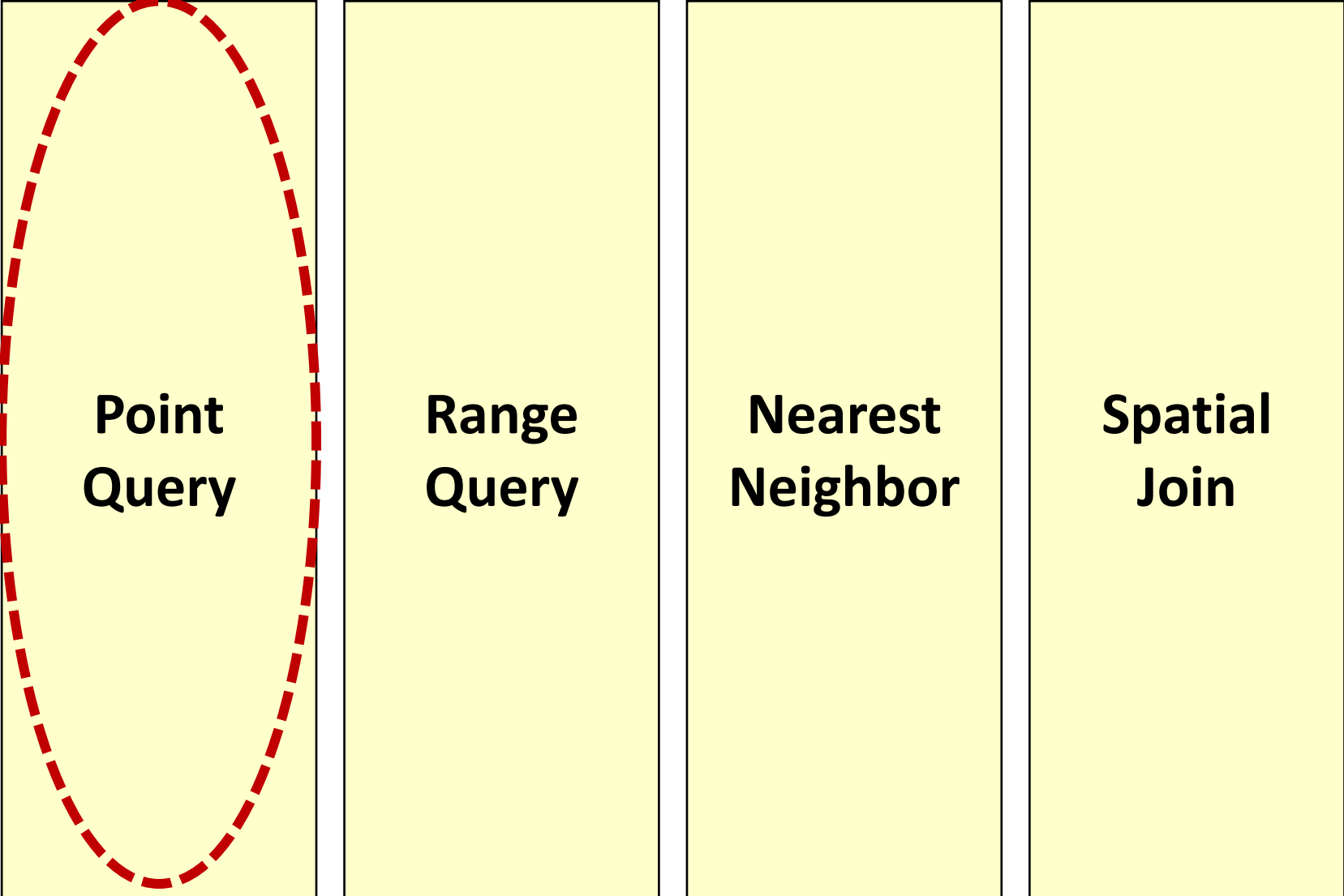
Always balanced!
Most commonly used



Urban Data Management



Query Processing (Spatial Data)



**Point
Query**

**Range
Query**

**Nearest
Neighbor**

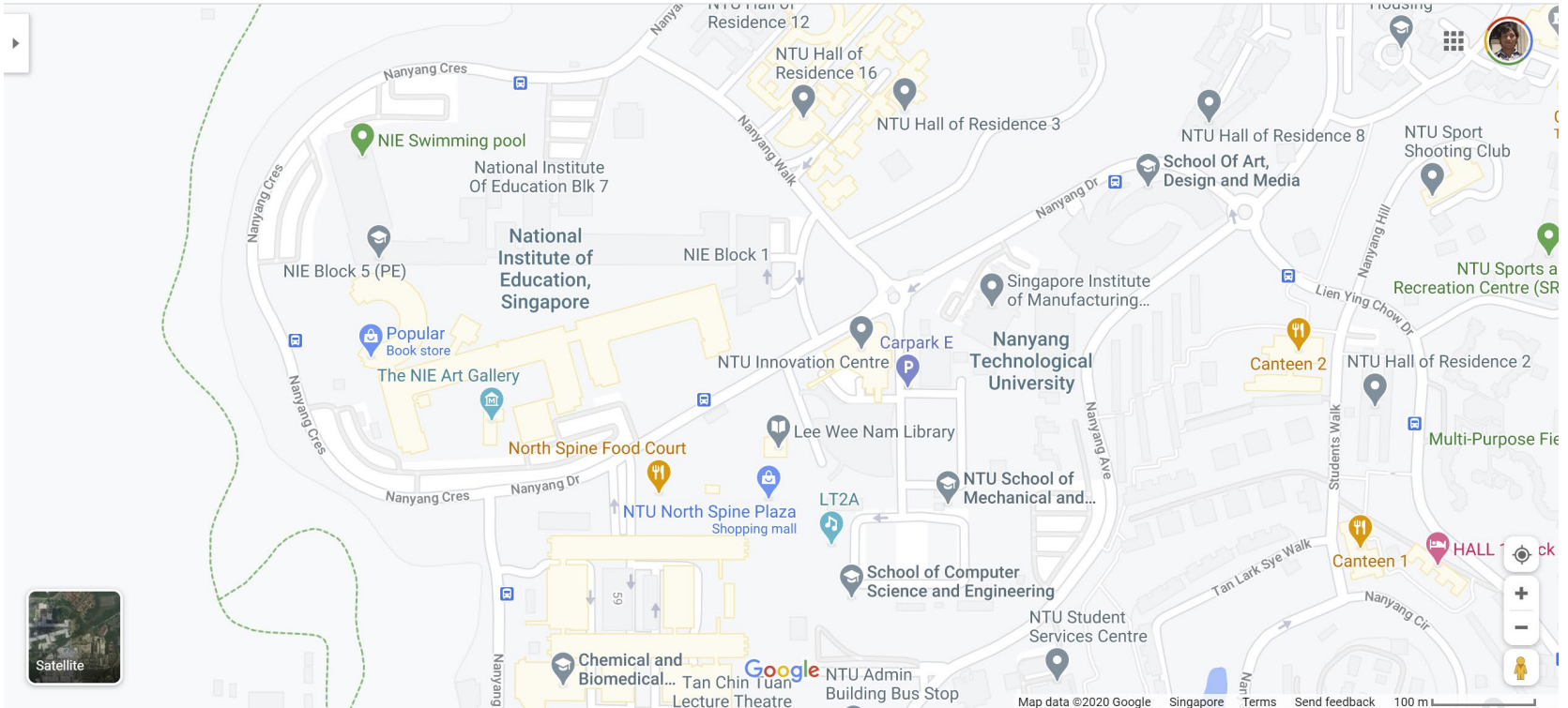
**Spatial
Join**

Point Query: Definition

Point Query

- Given a location
- Return a property (e.g., place name) of the location

Point Query: Example



Google Maps

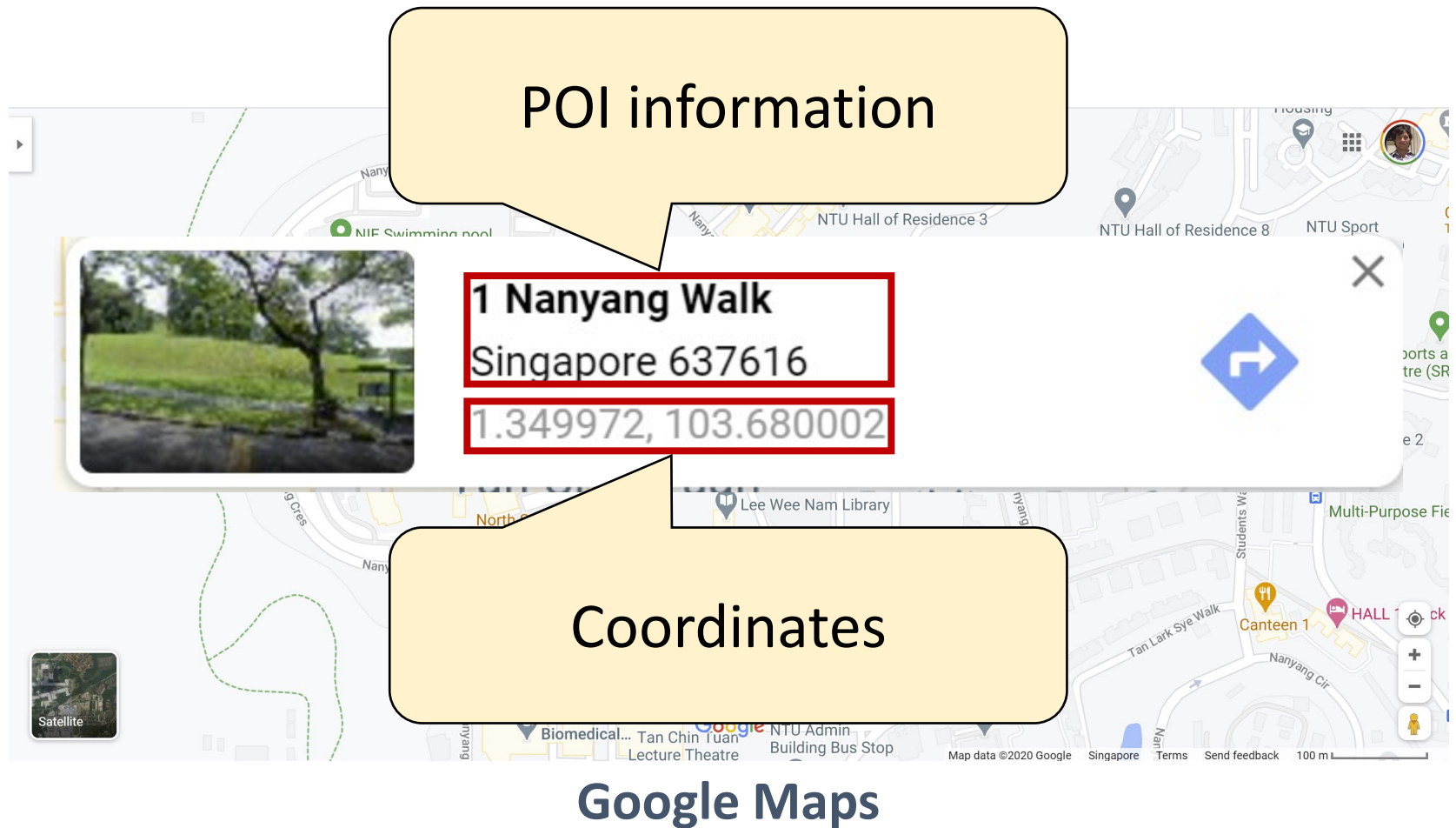
Point Query: Example



Google Maps

Point Query: Example

POI information



1 Nanyang Walk
Singapore 637616
1.349972, 103.680002

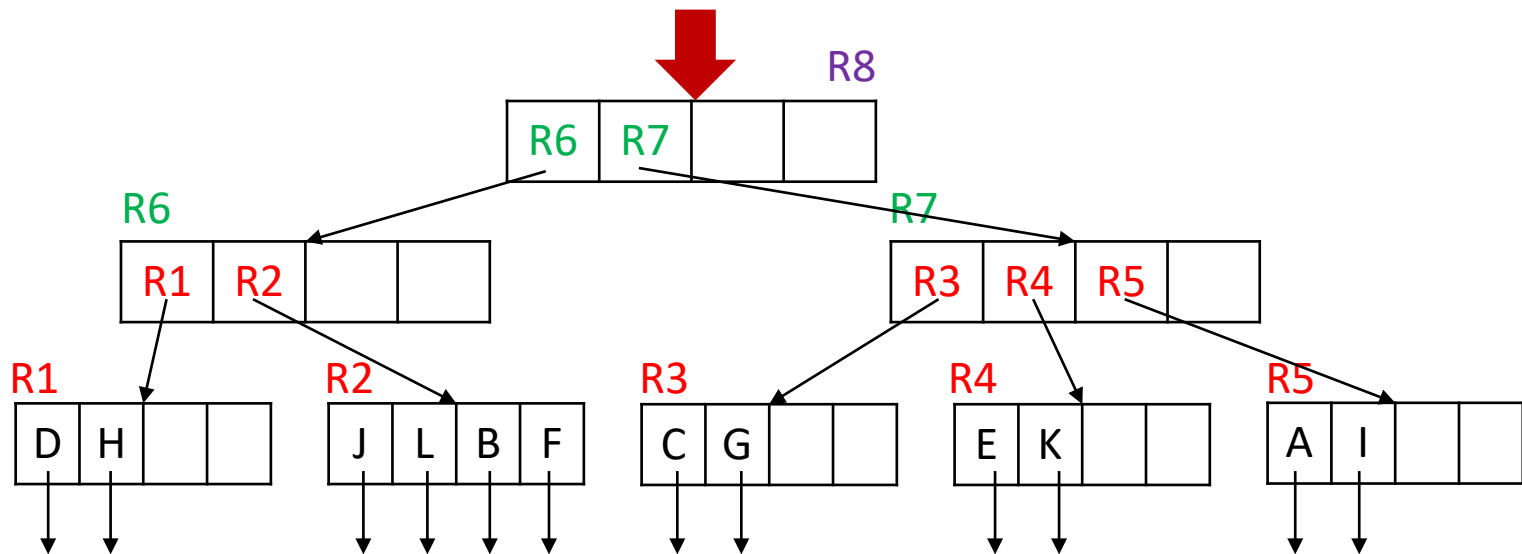
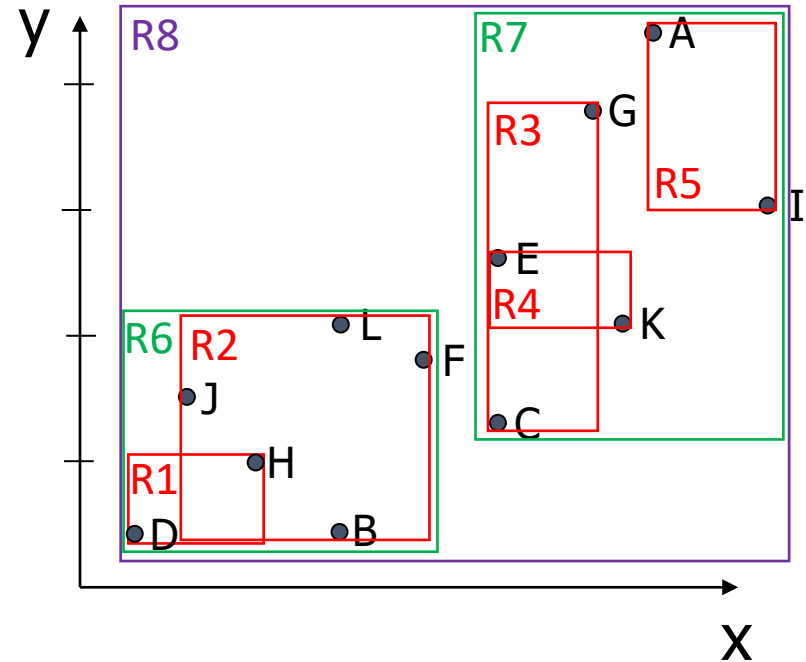
Coordinates

Google Maps

Point Query: Algorithm (With R Tree)

Search point K

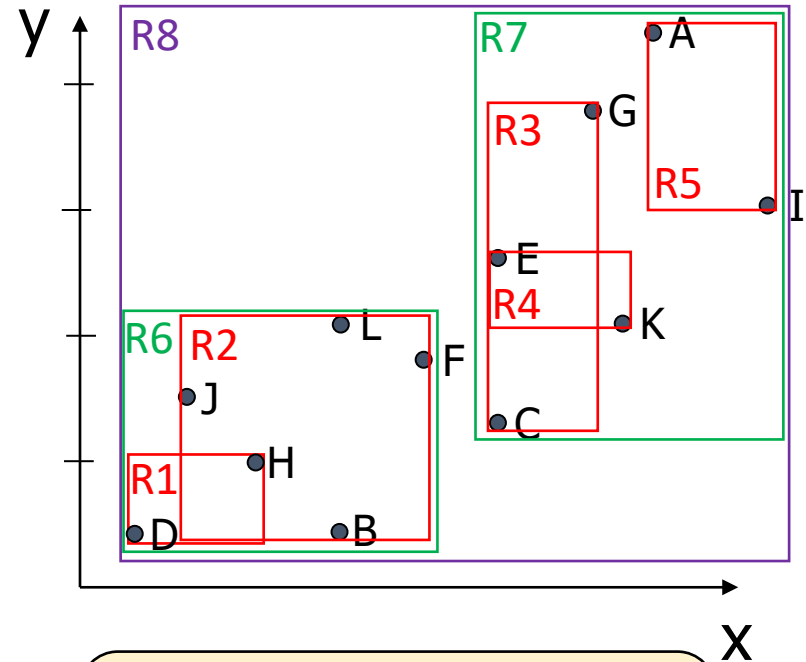
1. Check if the root's MBR intersects the point, if so, proceed



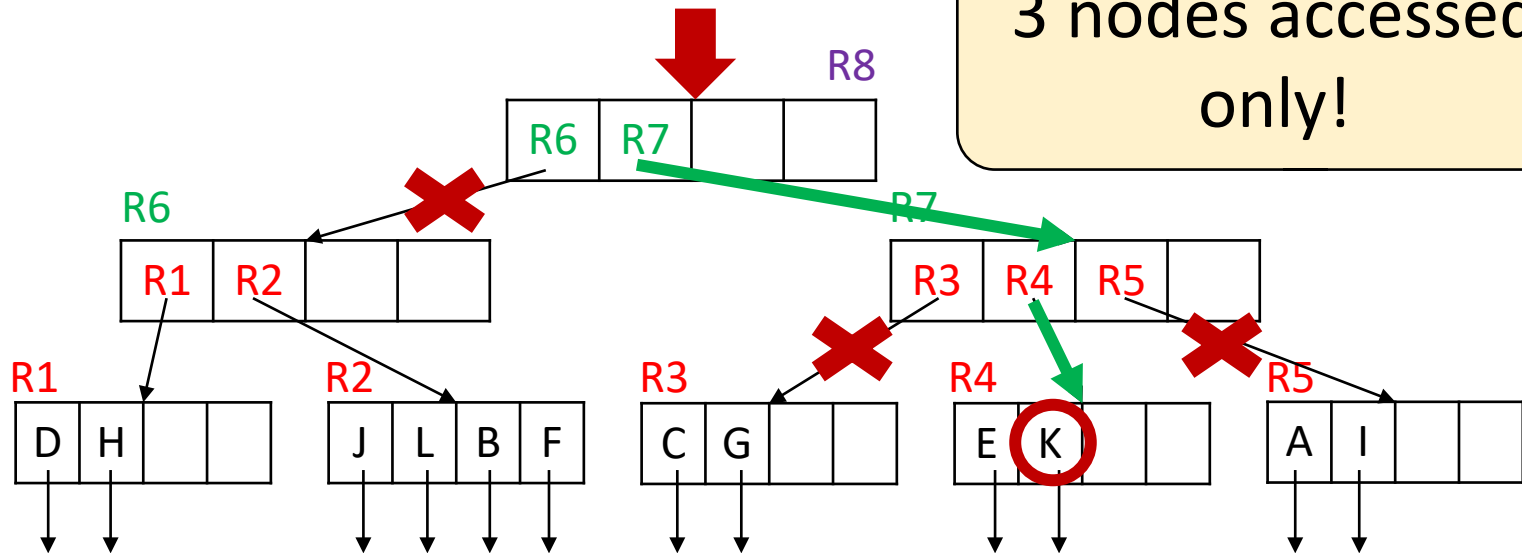
Point Query: Algorithm (With R Tree)

Search point K

2. Enumerate all children with the MBR intersecting K



3 nodes accessed only!



Query Processing (Spatial Data)

**Point
Query**

**Range
Query**

**Nearest
Neighbor**

**Spatial
Join**

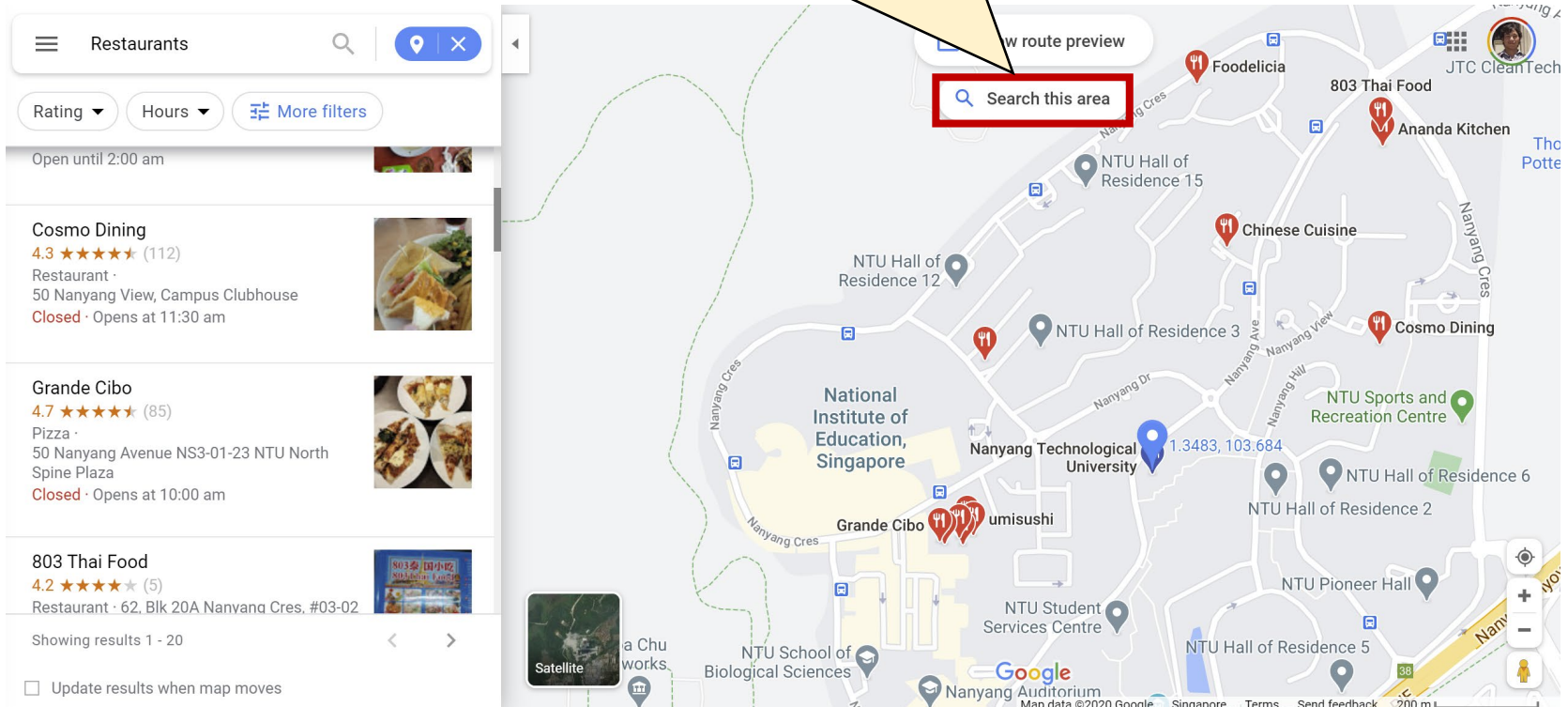
Range Query: Definition

Range Query:

- Given a region (e.g., a circle or a rectangle)
- Return all spatial objects within the region

Range Query: Example

“Search this area”

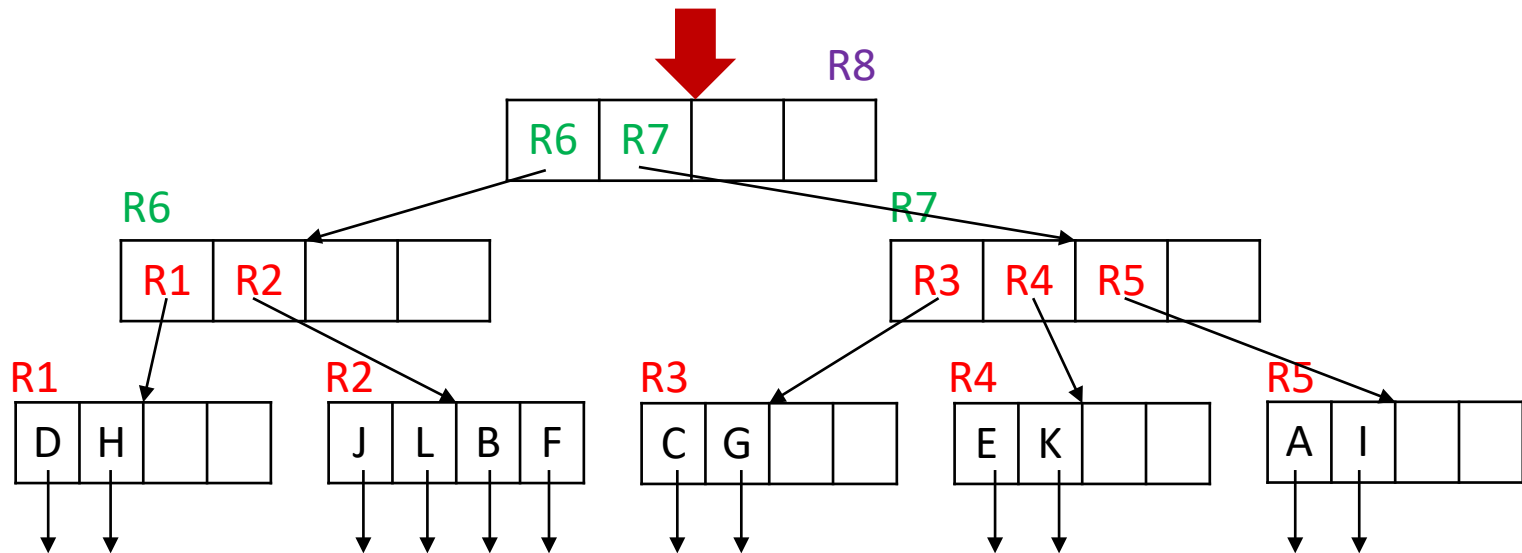
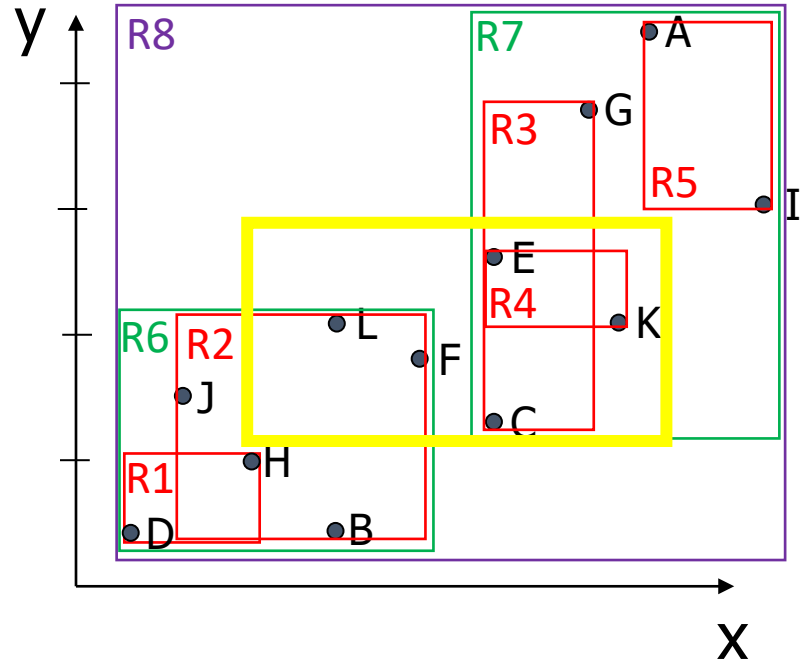


Google Maps

Range Query: Algorithm (With R Tree)

Search within the yellow box

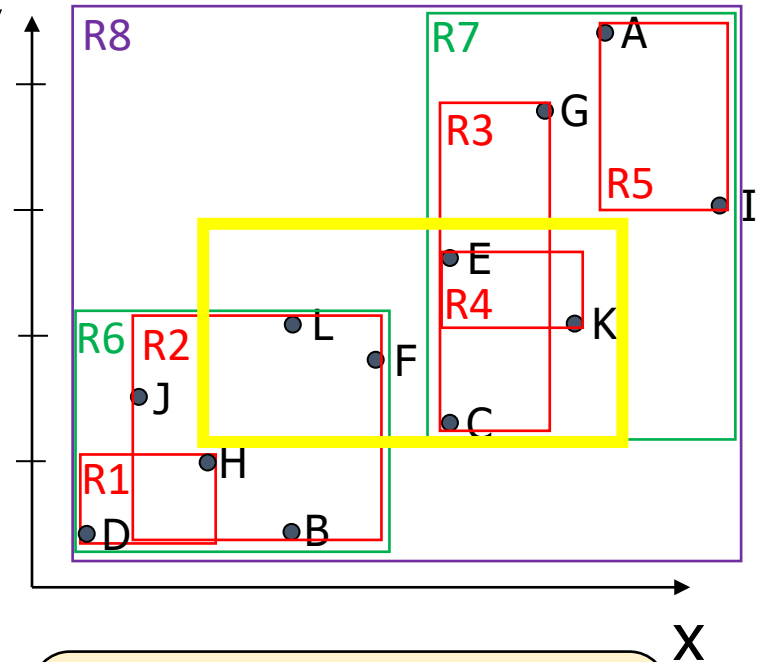
1. Check if the root's MBR intersects the point, if so, proceed



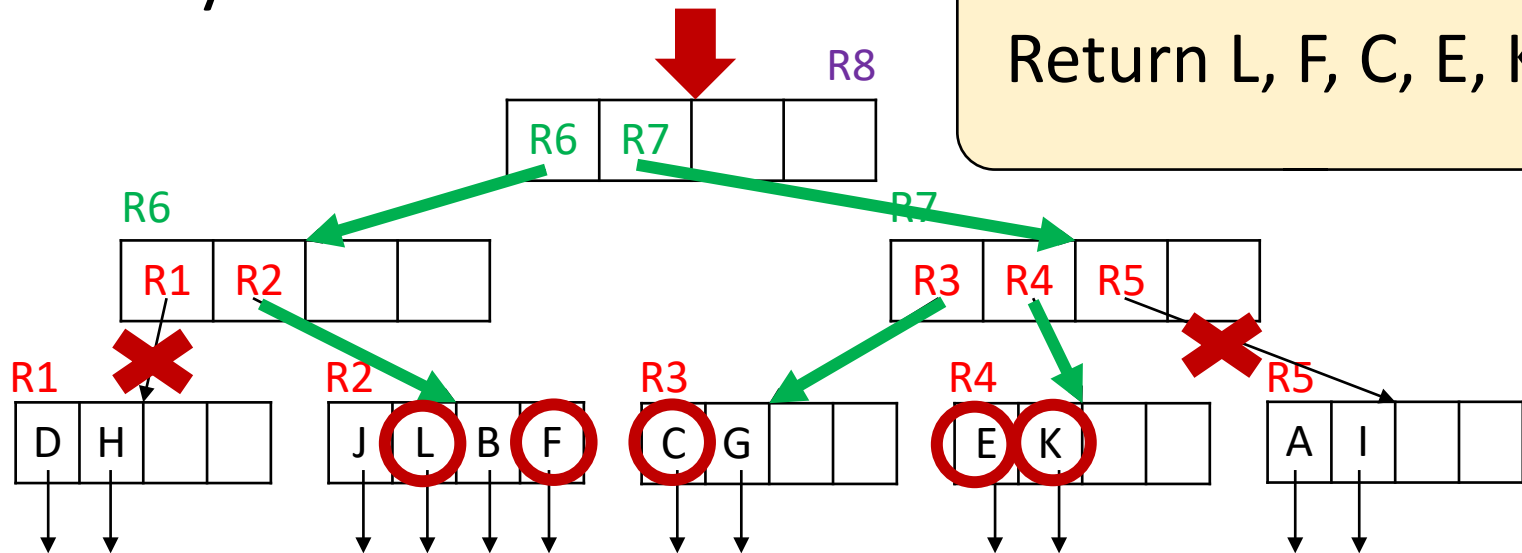
Range Query: Algorithm (With R Tree)

Search within the yellow box

2. Enumerate all children with the MBR intersecting the yellow box



Return L, F, C, E, K



Query Processing (Spatial Data)

**Point
Query**

**Range
Query**

**Nearest
Neighbor**

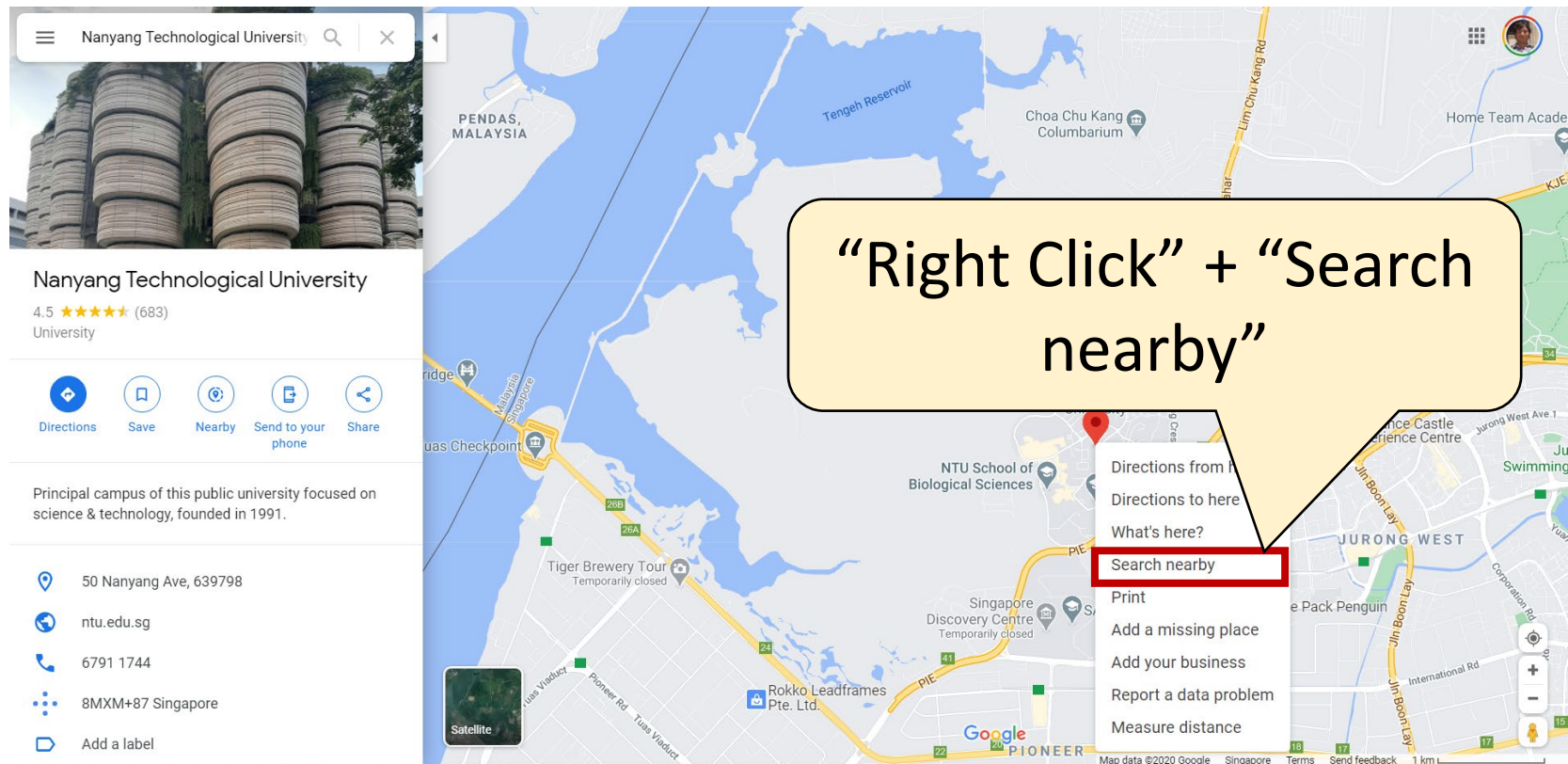
**Spatial
Join**

Nearest Neighbor: Definition

Nearest Neighbor:

- Given a query location
- Return the object **nearest** to the query location

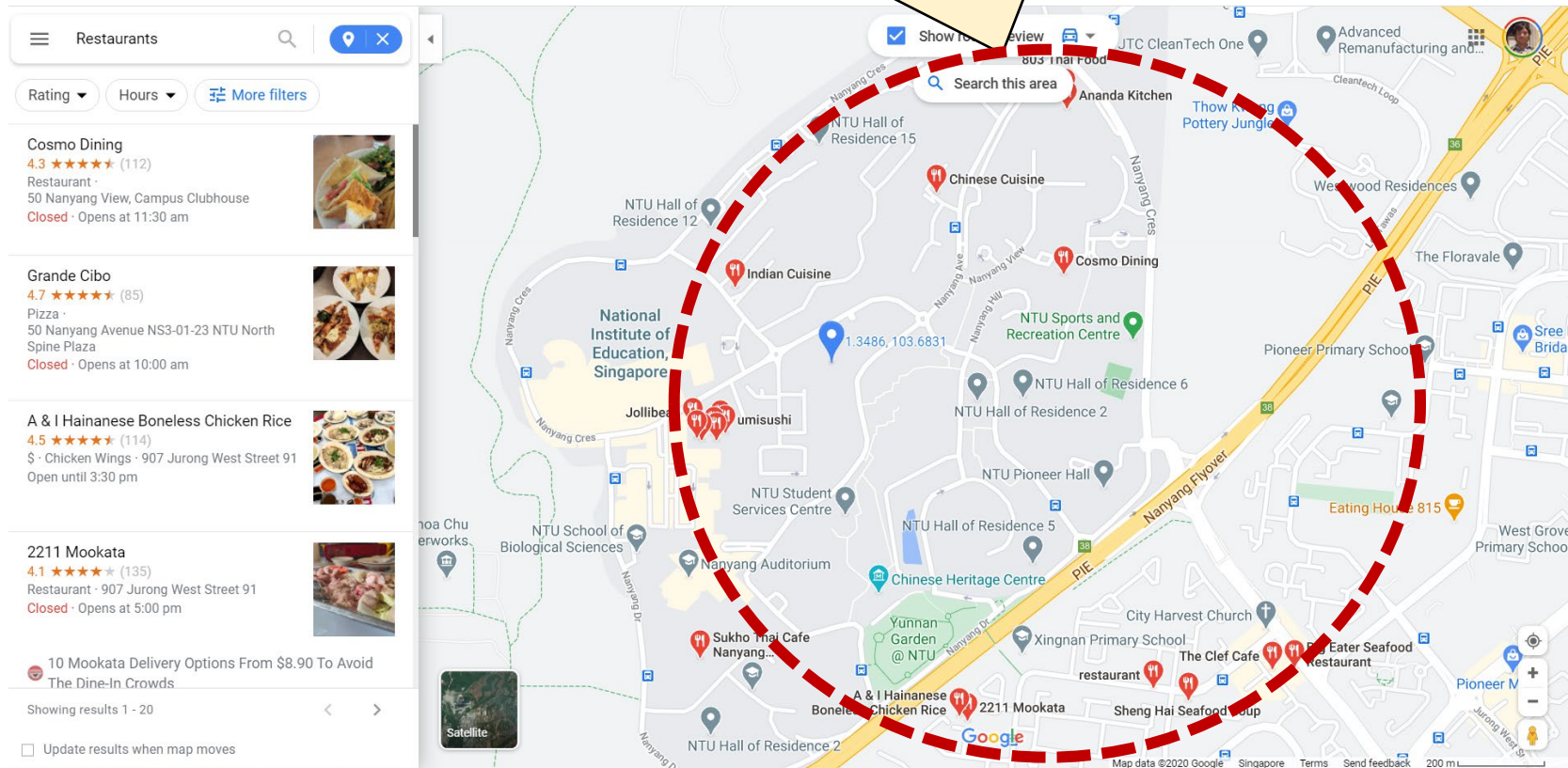
Nearest Neighbor: Example



Google Maps

Nearest Neighbor: Example

20 nearest restaurants



Google Maps

Nearest Neighbor: Algorithm (With R Tree)

**Filter-and-
Refine
Algorithm**

**Best-First
Search
Algorithm**

Filter and Refine Algorithm

Filter and Refine:

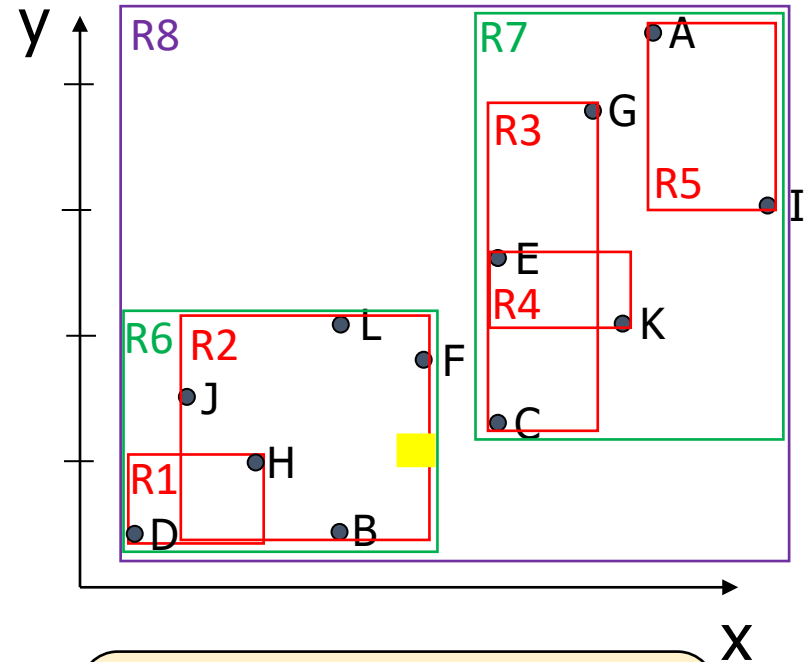
1. Fetch the node that contain the query location
2. R = minimum distance between the query location and an object in the fetched node
3. Find all objects within distance R from the query location (via a Range Query)
4. Return among the found objects the one that is the closest to the query location

Filter and Refine Algorithm

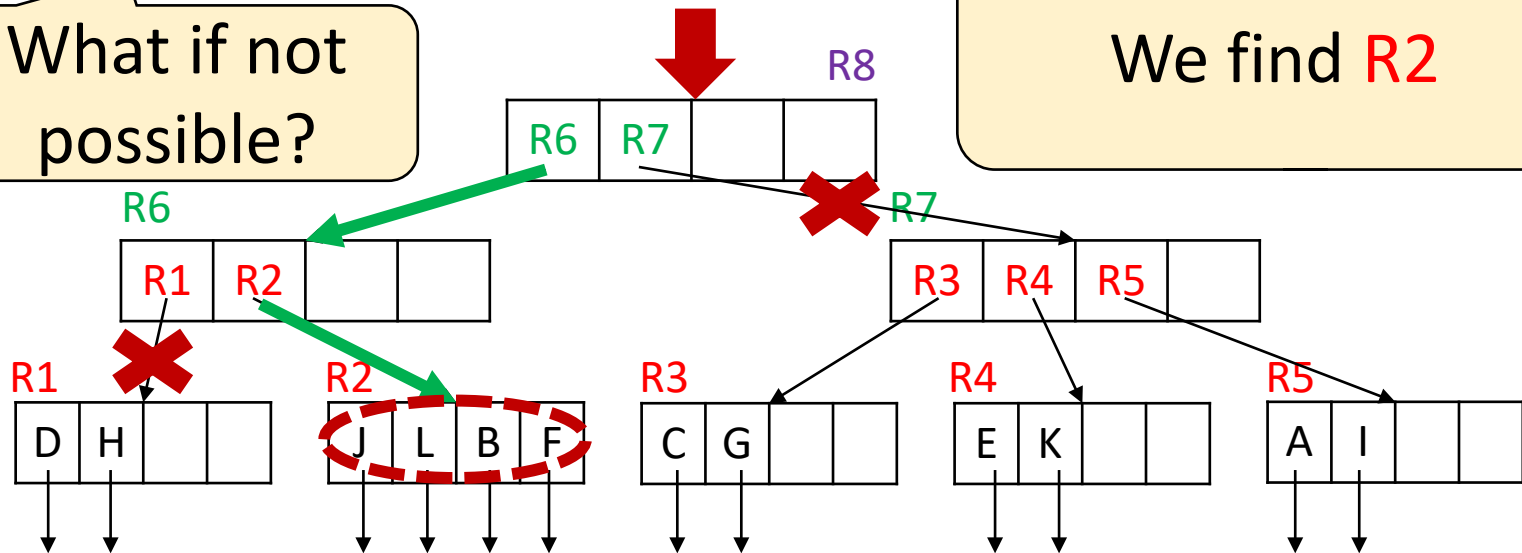
Find the object nearest to yellow dot

1. Fetch the leaf node containing the yellow dot

What if not possible?



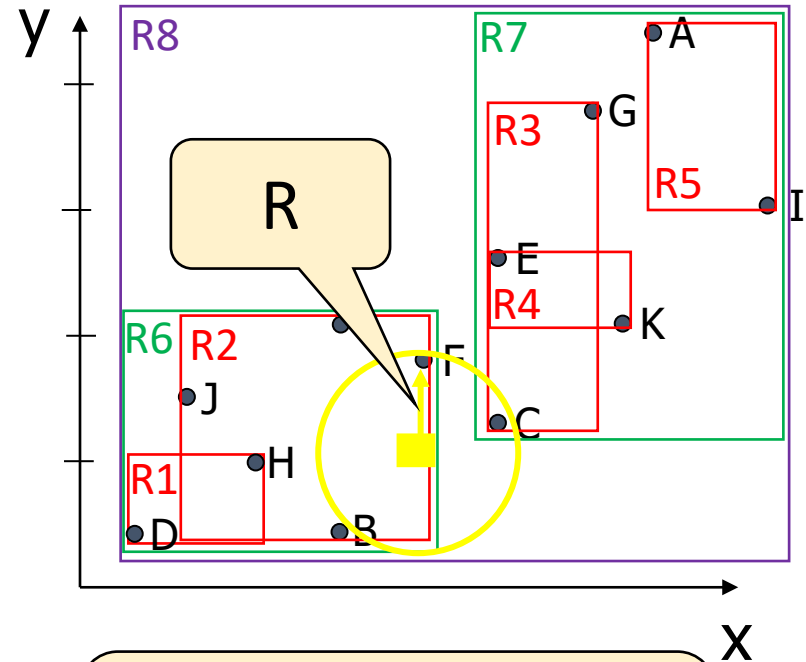
We find **R2**



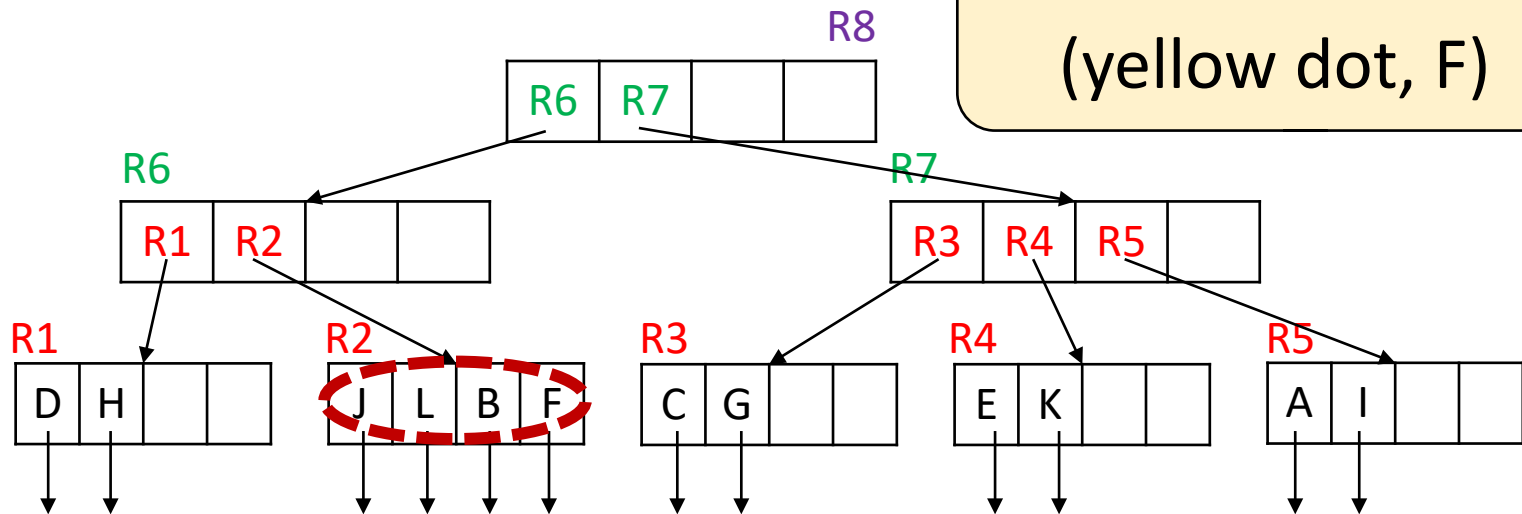
Filter and Refine Algorithm

Search within the yellow box

2. $R = \min$ distance between the yellow dot and an object in $R2$



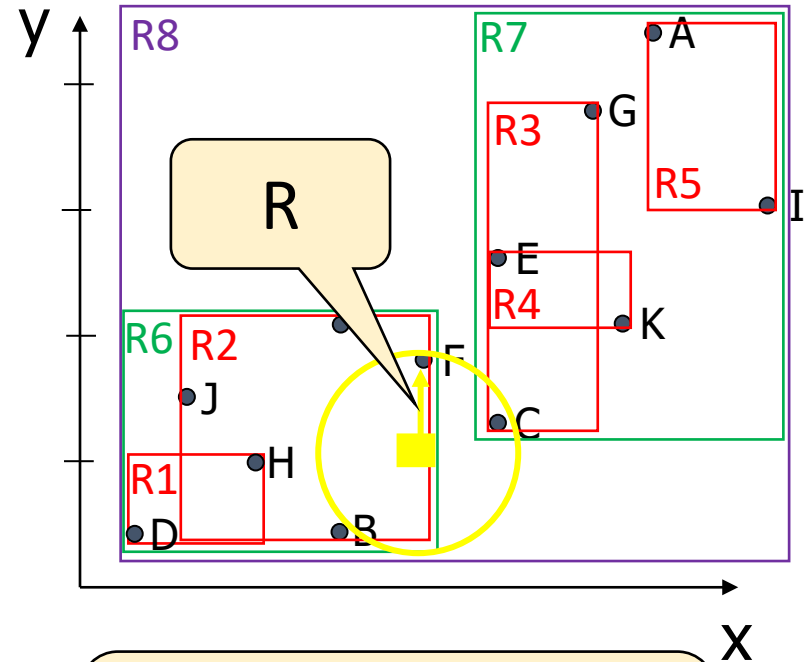
$R = \text{distance}$
(yellow dot, F)



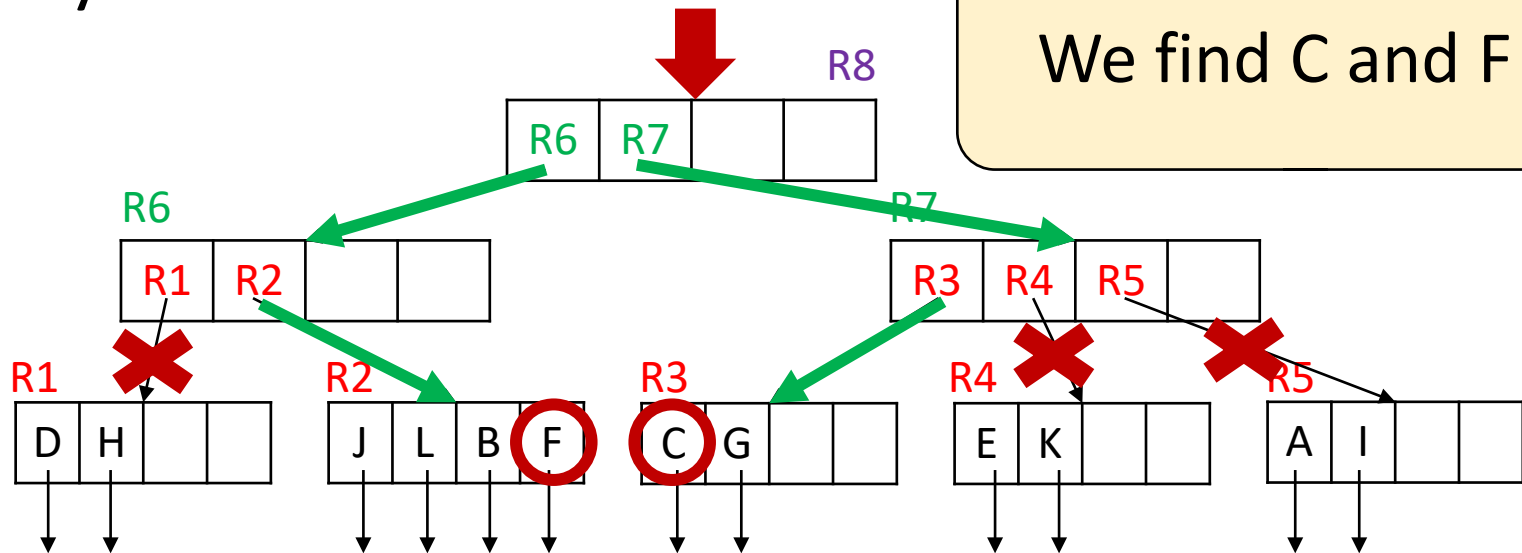
Filter and Refine Algorithm

Search within the yellow box

3. Perform a range query within R distance from the yellow dot



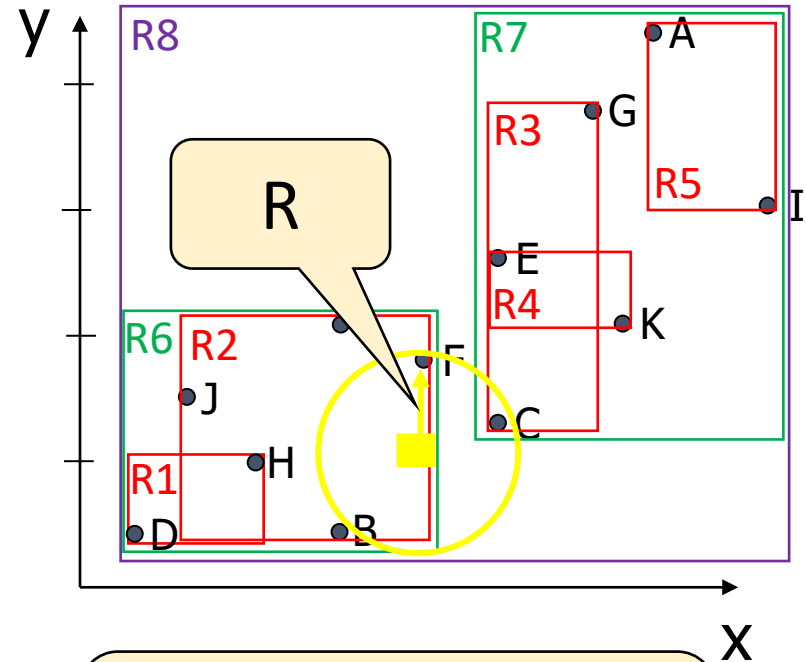
We find C and F



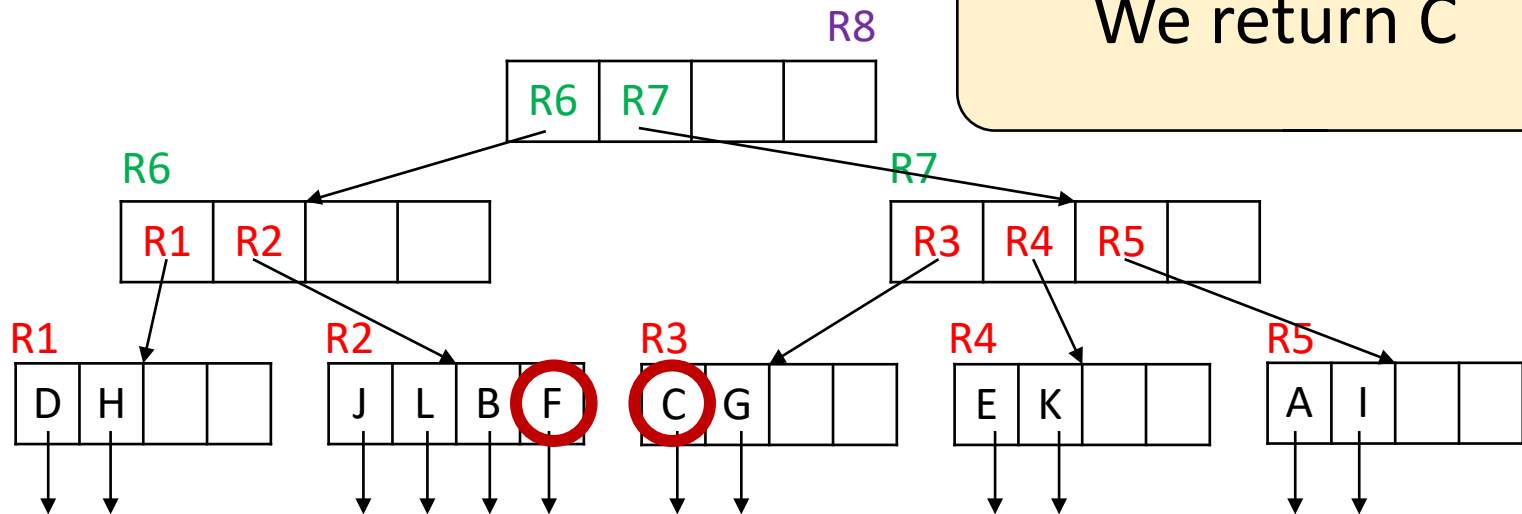
Filter and Refine Algorithm

Search within the yellow box

4. Return among the found objects the one that is the closest to the dot



We return C



Nearest Neighbor: Algorithm (With R Tree)

**Filter-and-
Refine
Algorithm**

**Best-First
Search
Algorithm**

Best-First Search Algorithm

Best-First Search:

- Maintain an **upper bound** of the distance between the query location and its nearest neighbor, denoted by **UB**
- Explore only those nodes with their MBRs' **minimum** distance to the query location **at most** UB, and in an ascending order of the minimum distances (i.e., best-first search)

Best-First Search Algorithm

Find the object nearest to yellow dot

Min dist (R6, dot) 0

Min dist (R2, dot) 0

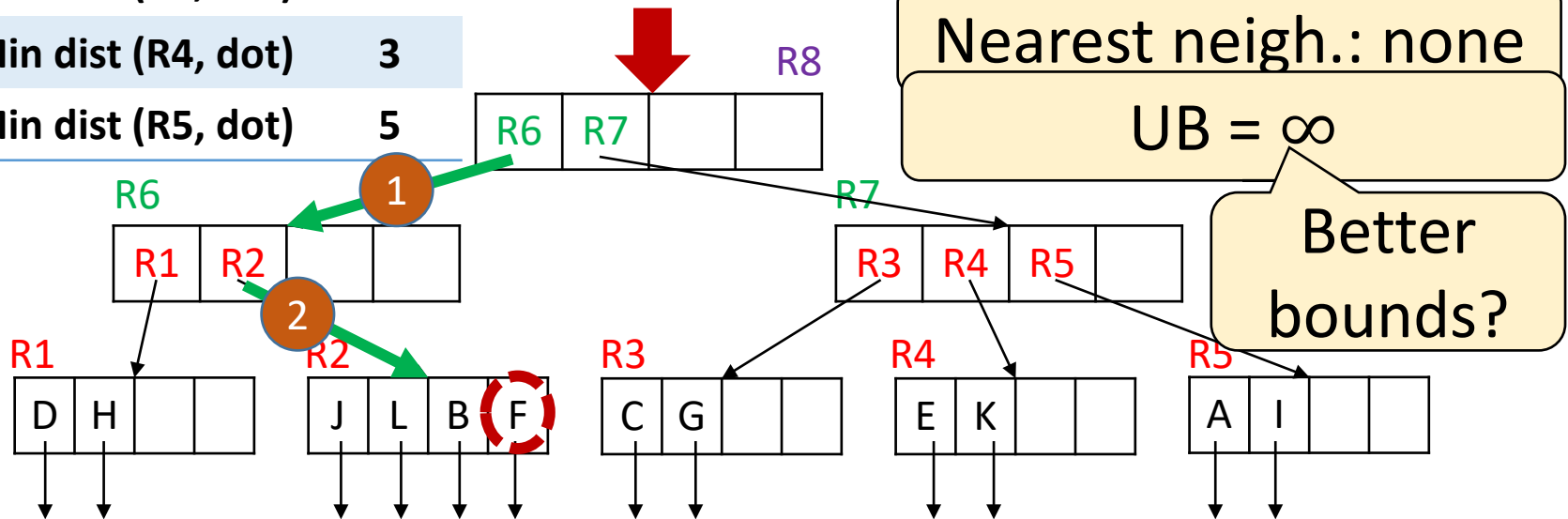
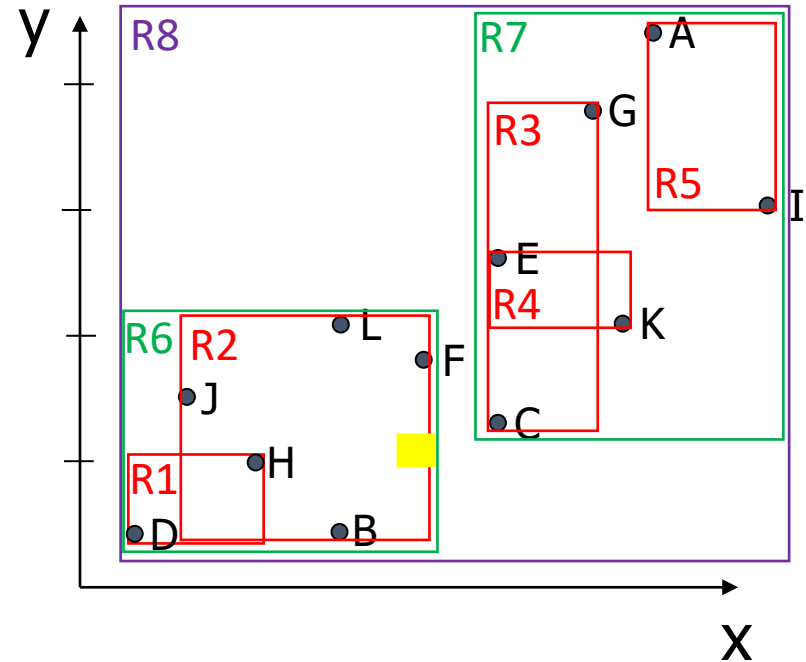
Min dist (R7, dot) 1

Min dist (R3, dot) 1

Min dist (R1, dot) 2.5

Min dist (R4, dot) 3

Min dist (R5, dot) 5



Best-First Search Algorithm

Find the object nearest to yellow dot

Min dist (R6, dot) 0

Min dist (R2, dot) 0

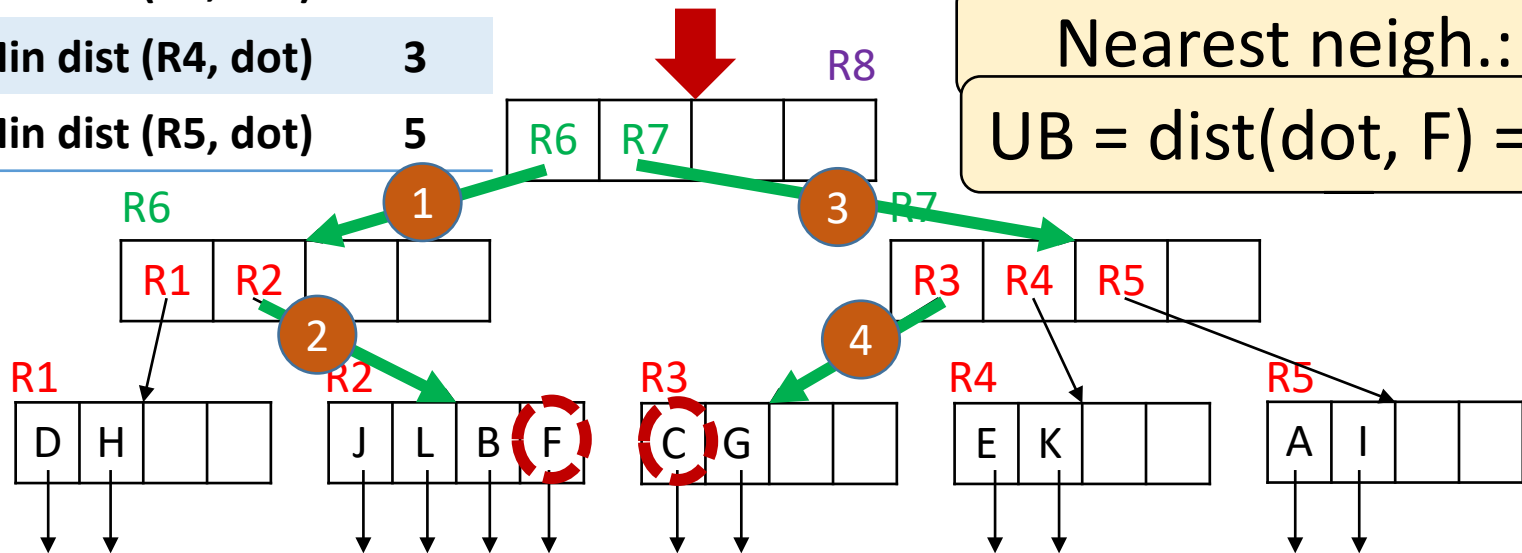
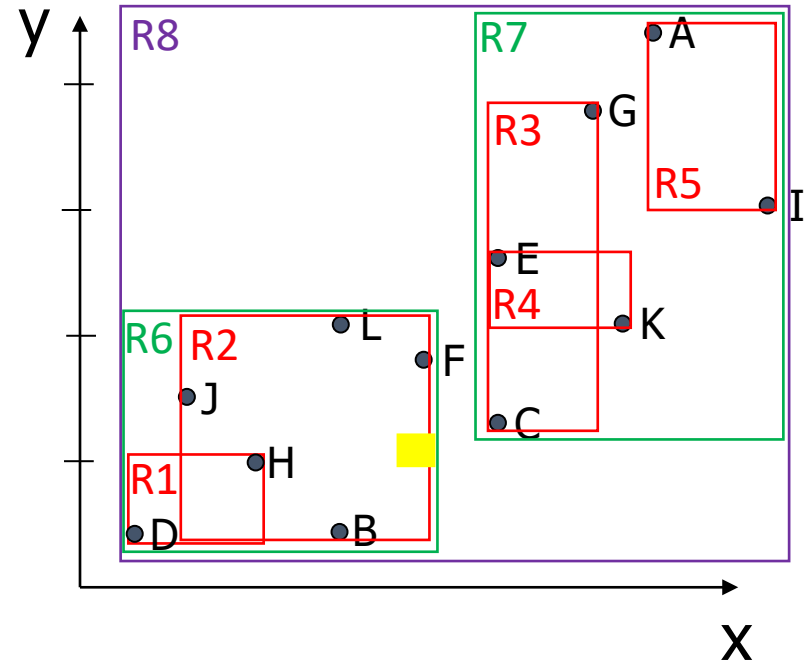
Min dist (R7, dot) 1

Min dist (R3, dot) 1

Min dist (R1, dot) 2.5

Min dist (R4, dot) 3

Min dist (R5, dot) 5



Nearest neigh.: F

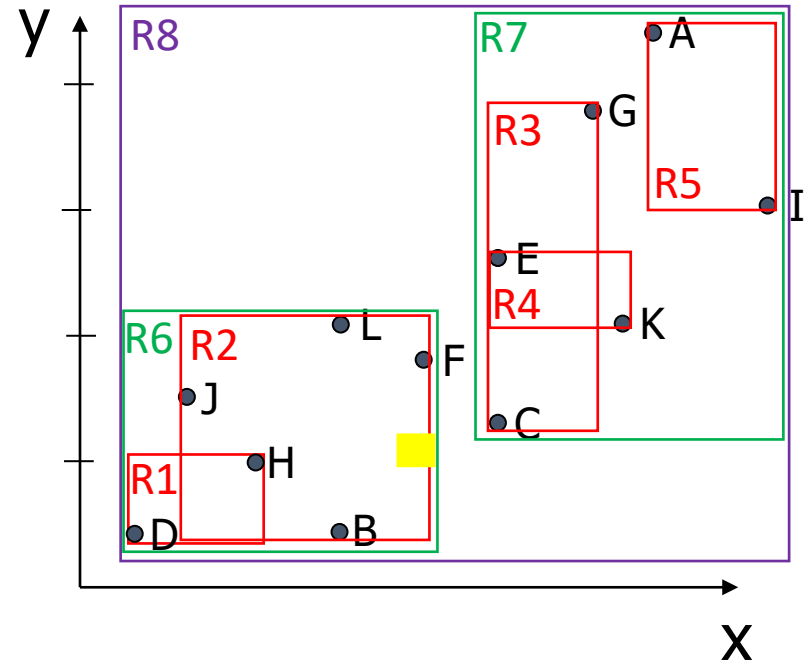
$UB = \text{dist}(\text{dot}, F) = 1.5$

Best-First Search Algorithm

Find the object nearest to yellow dot

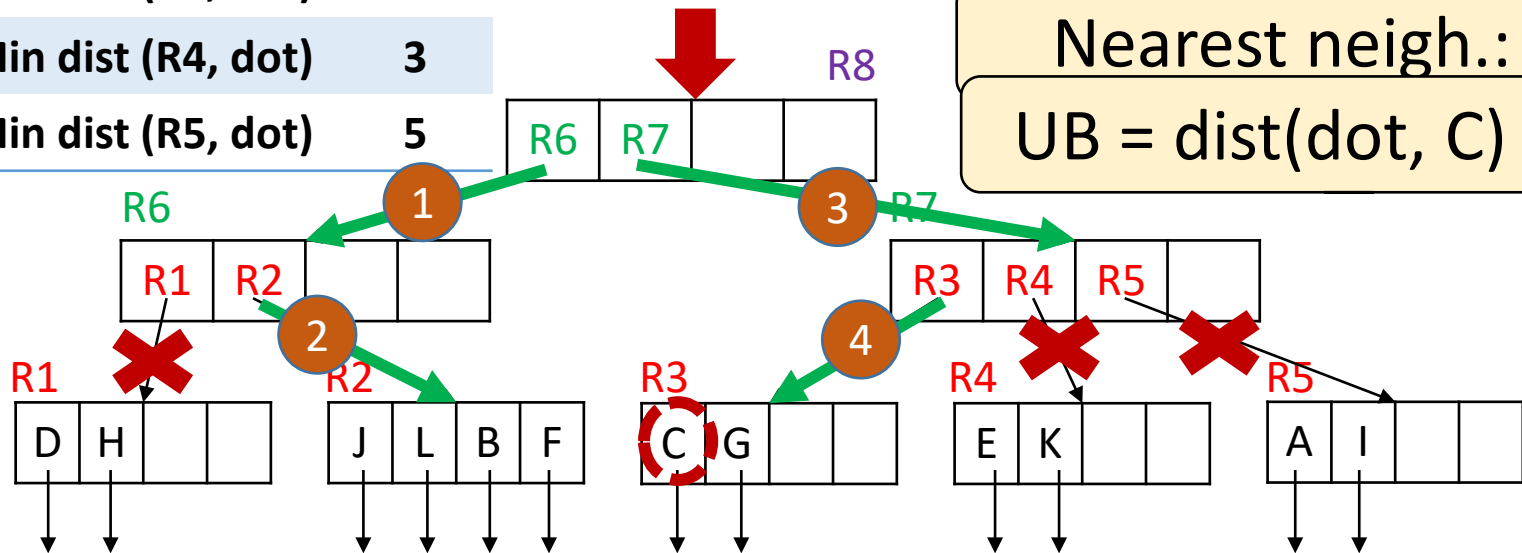
Min dist (R6, dot) 0Min dist (R2, dot) 0Min dist (R7, dot) 1Min dist (R3, dot) 1Min dist (R1, dot) 2.5

Min dist (R4, dot) 3

Min dist (R5, dot) 5

Nearest neigh.: C

$$UB = \text{dist}(\text{dot}, C) = 1$$



Query Processing (Spatial Data)

**Point
Query**

**Range
Query**

**Nearest
Neighbor**

**Spatial
Join**

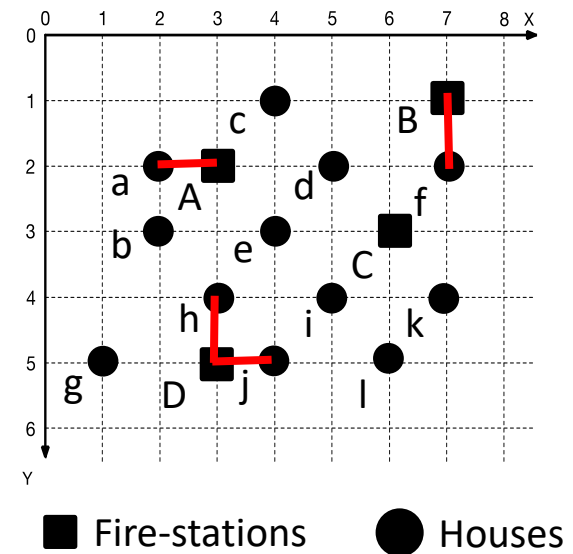
Spatial Join: Definition

Spatial Join:

- Given two sets of spatial objects
- Find pairs of objects, each from one set, which are close to each other

Spatial Join: Example

Find pairs of fire stations and houses, which are within distance of 1

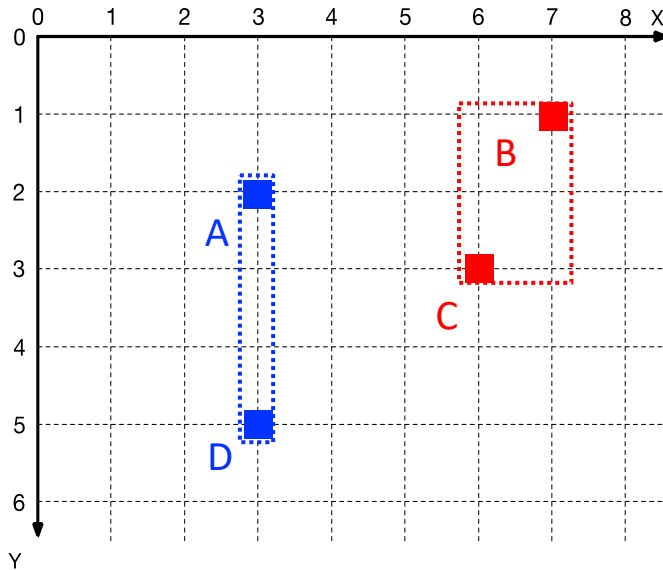


Spatial Join: Algorithm (With R Tree)

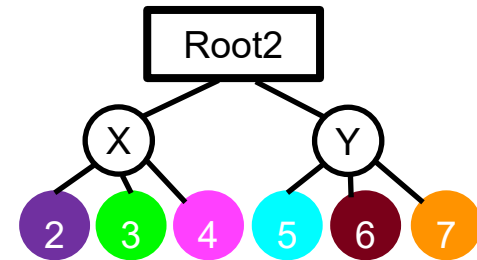
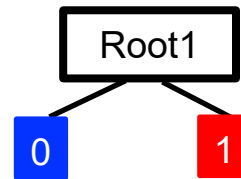
Spatial Join Algorithm:

- Check if the rectangles of the roots satisfy the join predicate, and if so, start from the pair of (root1, root2)
- For each pair, **recursively** examine the pairs of their children if they intersect
- Return the pairs of objects that satisfy the join predicate

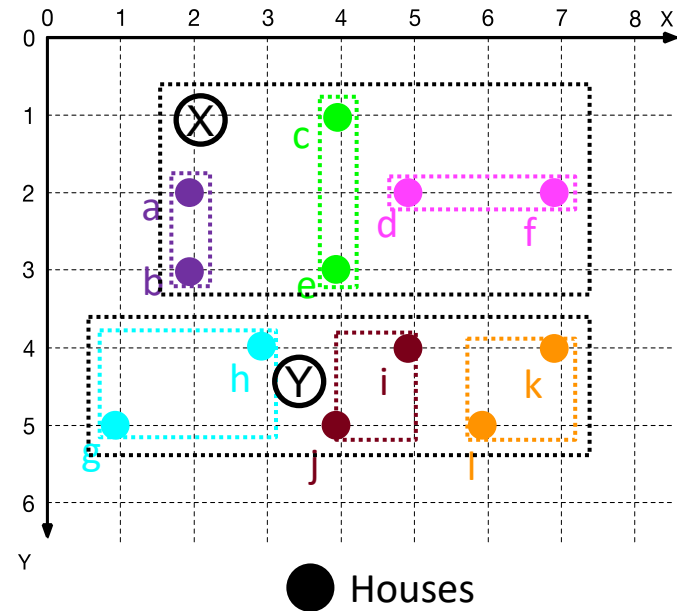
Spatial Join: Algorithm (With R Tree)



■ Fire-stations

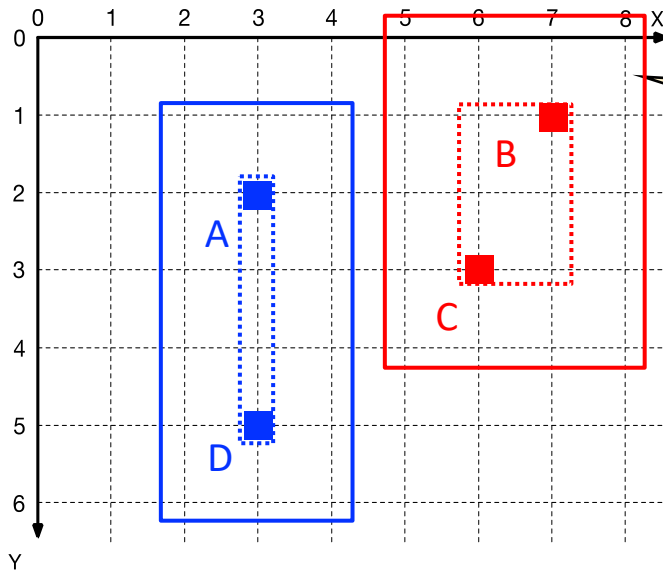


Find pairs of fire stations and houses, which are within distance of 1



● Houses

Spatial Join: Algorithm (With R Tree)



■ Fire-stations

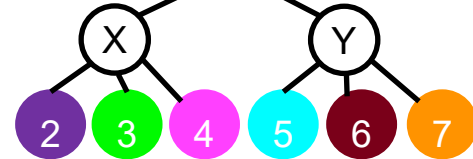
Root1

0

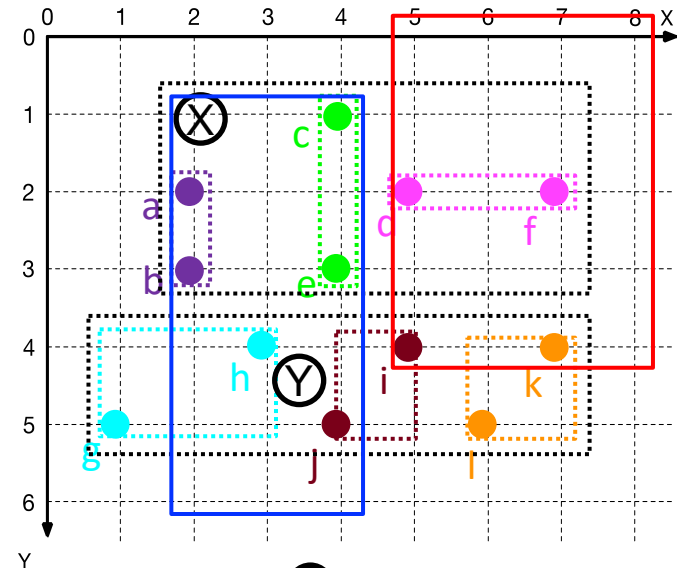
Use a bigger MBR by 1
(specific for the query)

1

Root2

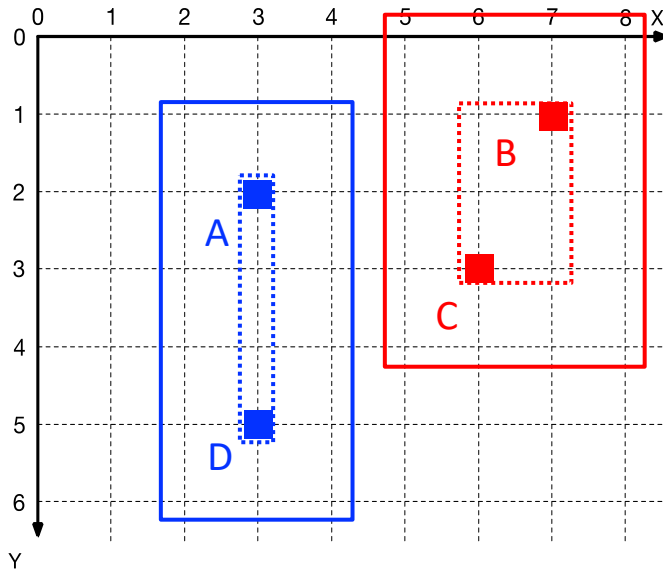


(Root1, Root2)

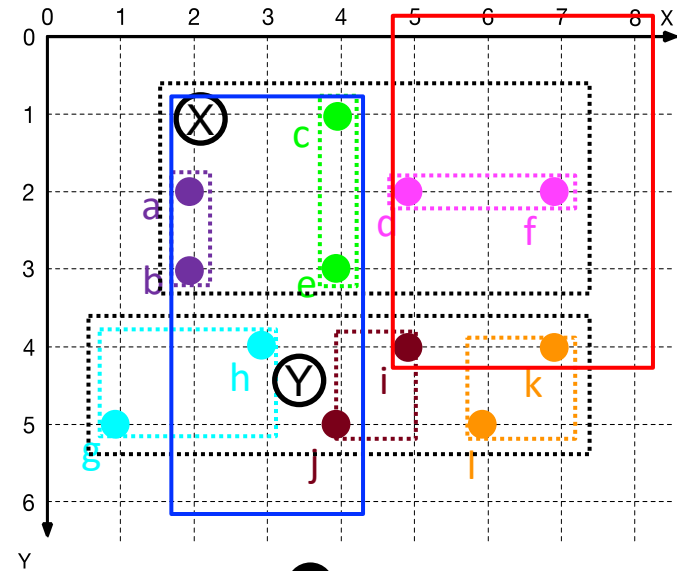
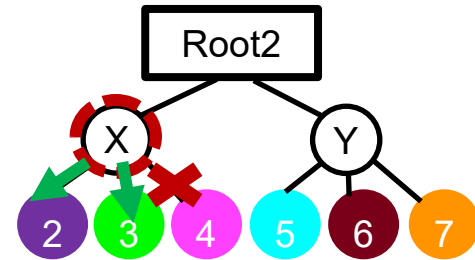
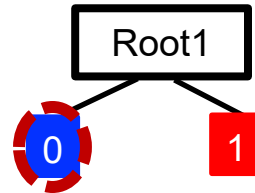


● Houses

Spatial Join: Algorithm (With R Tree)



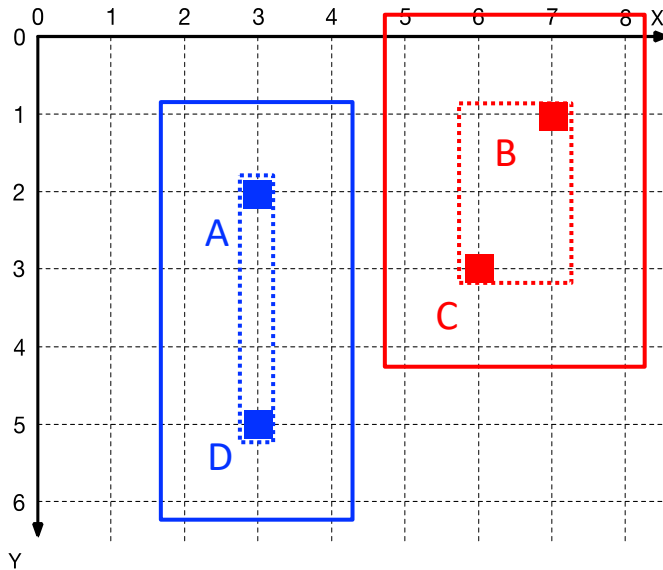
■ Fire-stations



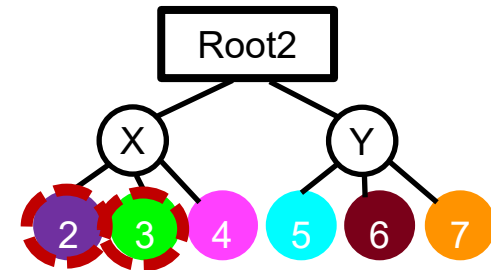
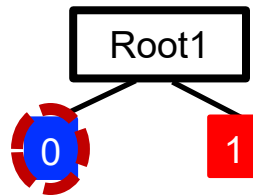
● Houses

(0, X)
(0, Y)
(1, X)
(1, Y)

Spatial Join: Algorithm (With R Tree)

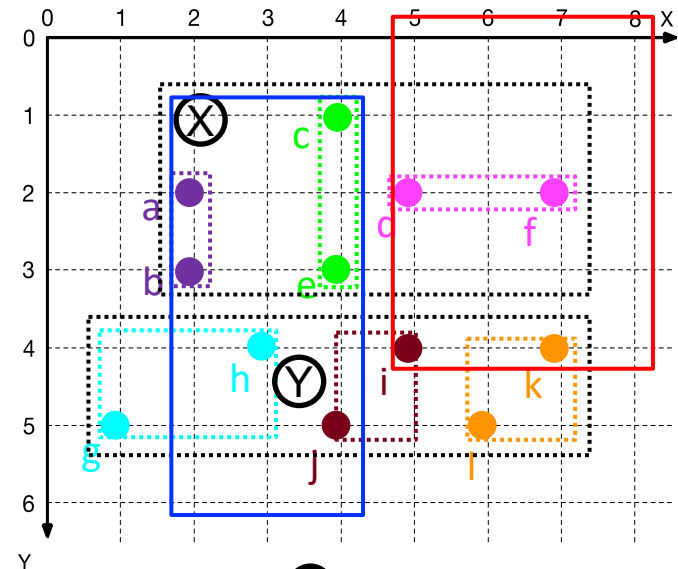


■ Fire-stations



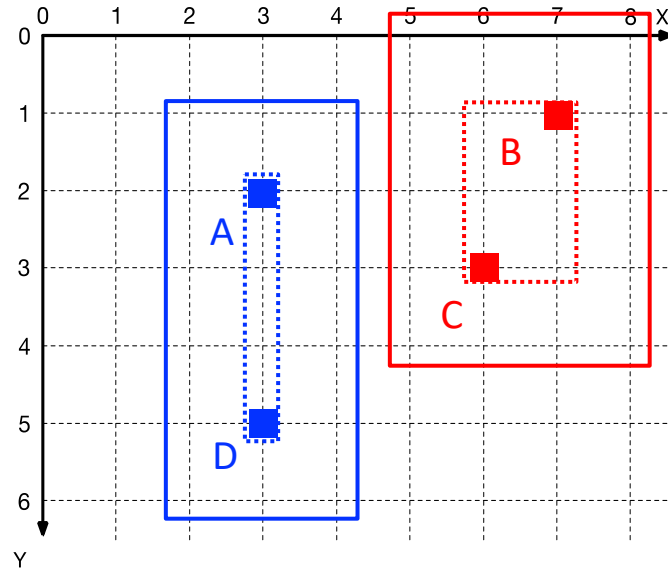
(0, 2)

(0, 3)

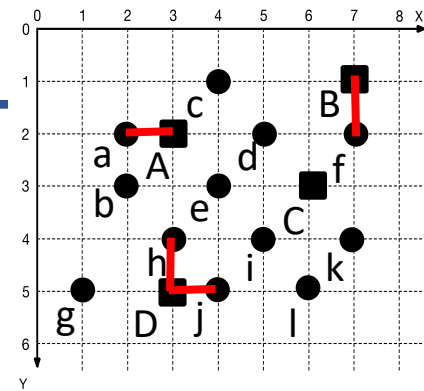
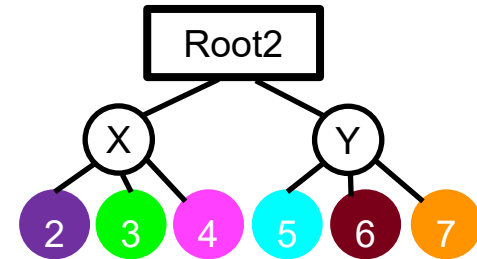
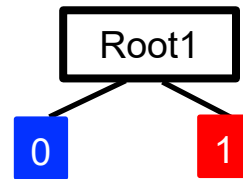


● Houses

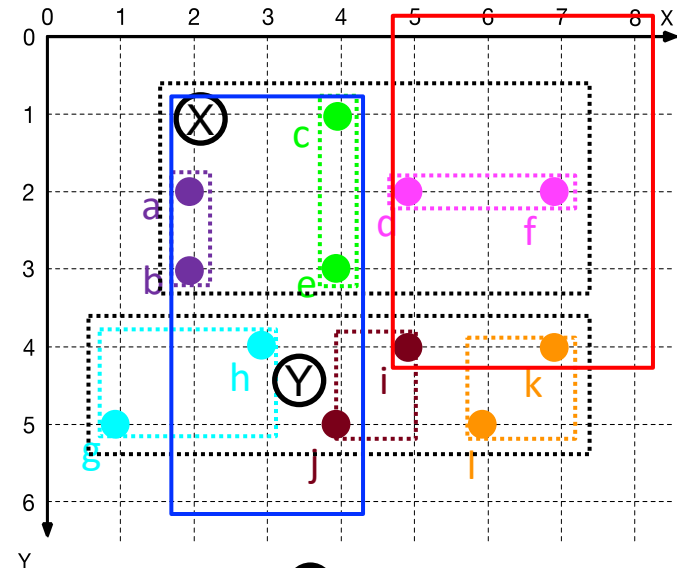
Spatial Join: Algorithm (With R T



■ Fire-stations

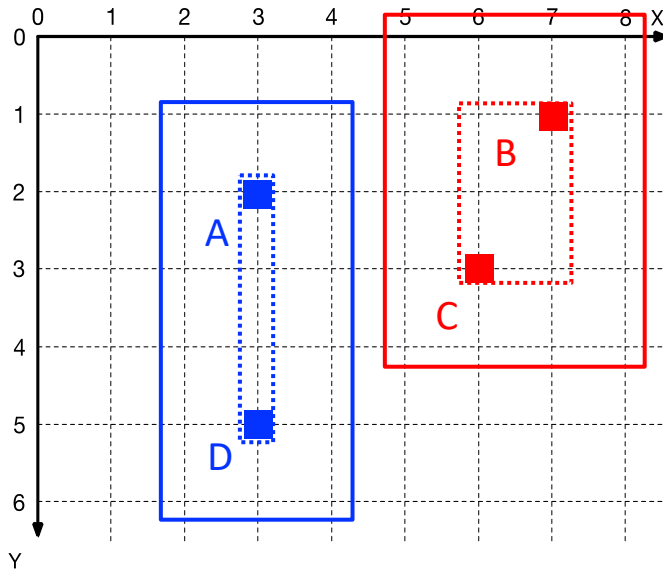


Return (A, a)

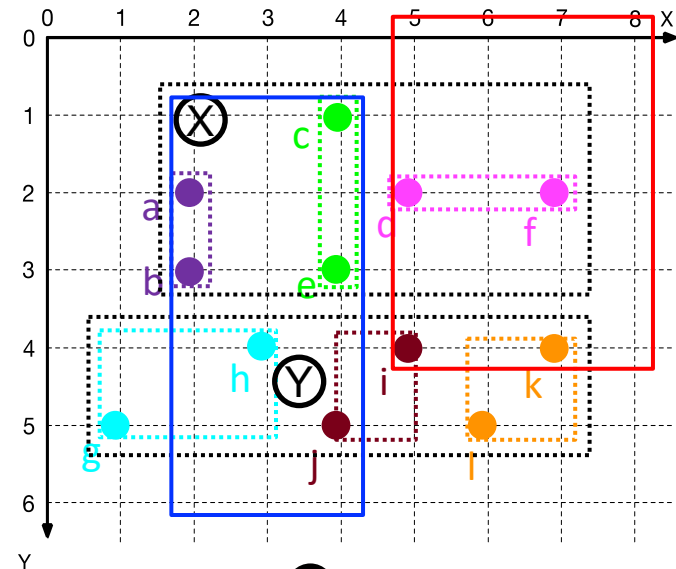
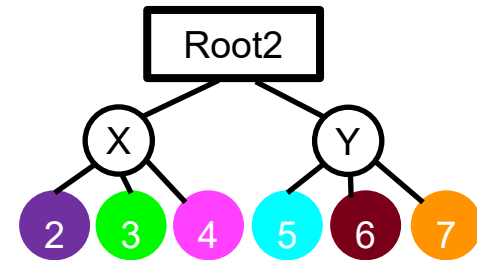
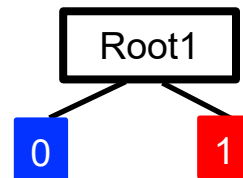


● Houses

Spatial Join: Algorithm (With R Tree)



■ Fire-stations



● Houses

$(0, X)$
 $(0, Y)$
 $(1, X)$
 $(1, Y)$

Similar process

Query Processing (Spatial Data)

**Point
Query**

**Range
Query**

**Nearest
Neighbor**

**Spatial
Join**

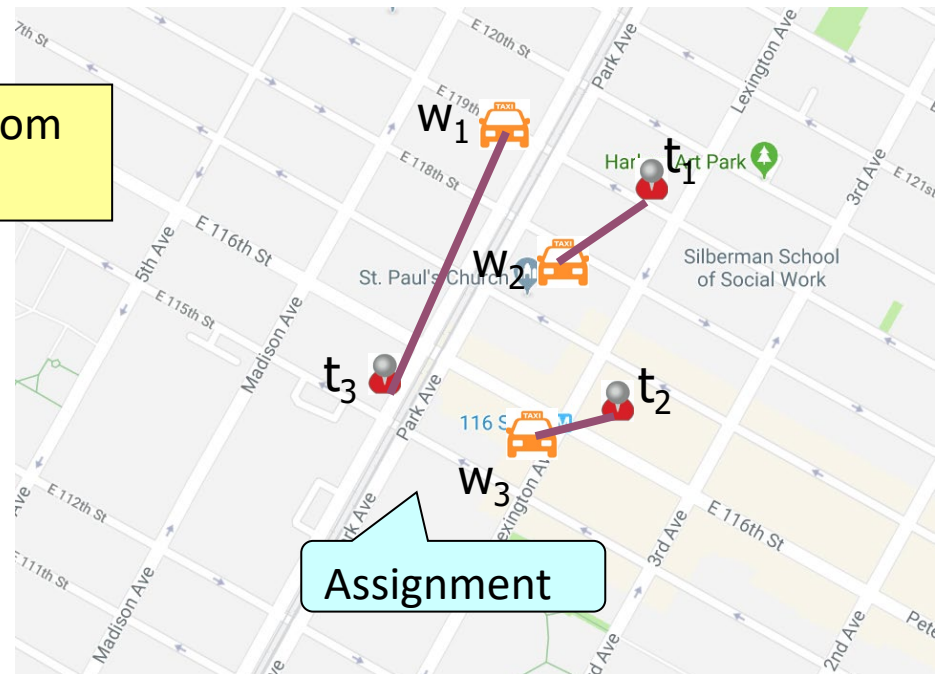
**Spatial
matching
(not tested)**

Spatial Matching: Introduction

We have a few possible **assignments** from tasks to workers

 — task

 — worker

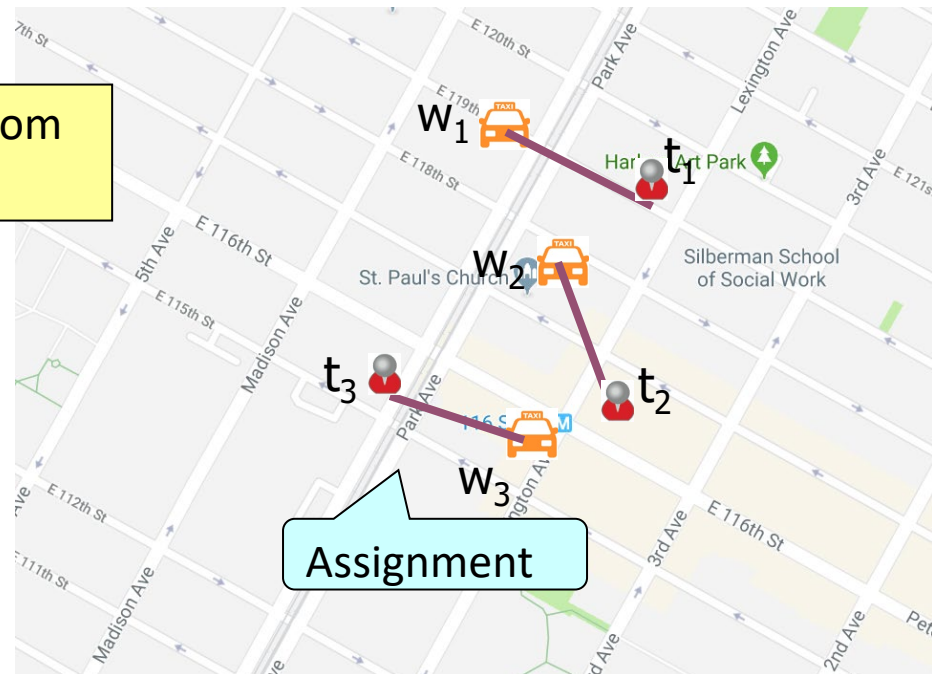


Spatial Matching: Introduction

We have a few possible **assignments** from tasks to workers

 — task

 — worker

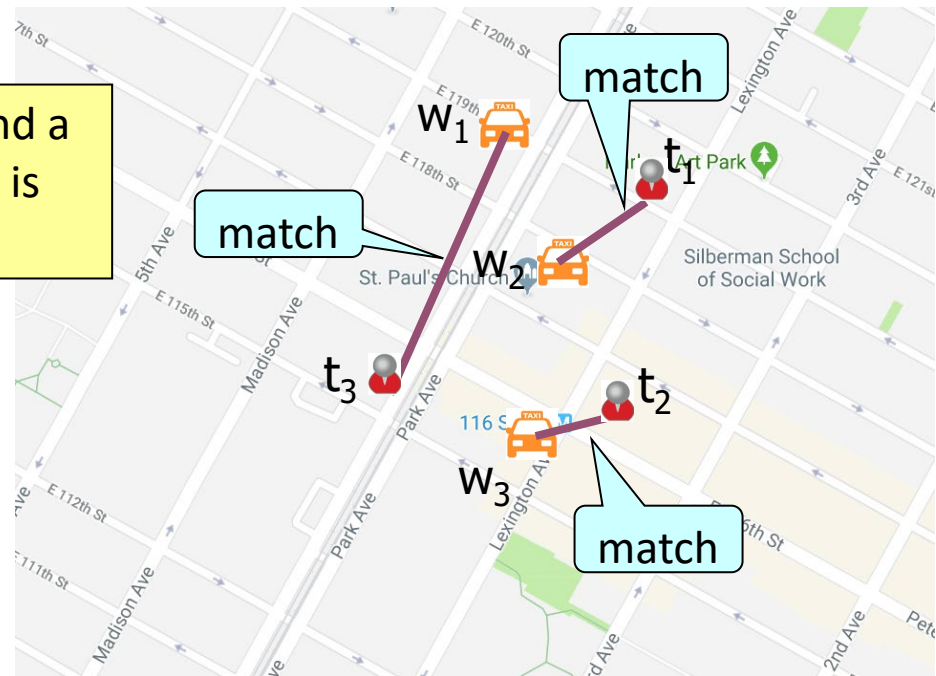


Spatial Matching: Introduction

In an assignment, each pair of a task and a worker that are assigned to each other is called a **match**

 — task

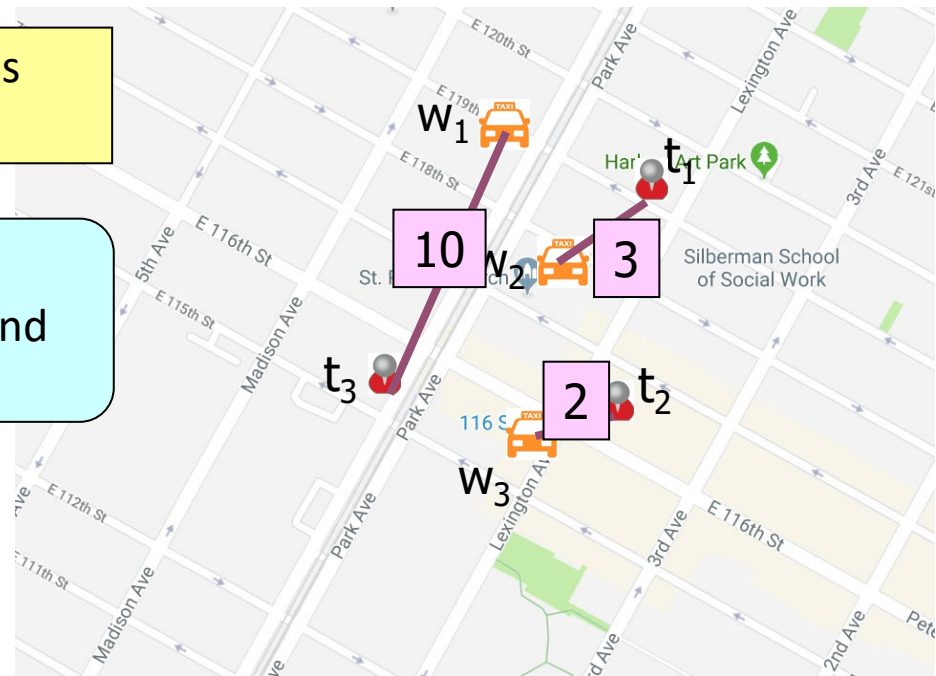
 — worker



Spatial Matching: Introduction

A match involving a task and a worker is usually associated with a **cost**

For ease of illustration, we use the **Euclidean distance** between the task and the worker as the cost



Spatial Matching: Problem Definition

Problem (Spatial matching):

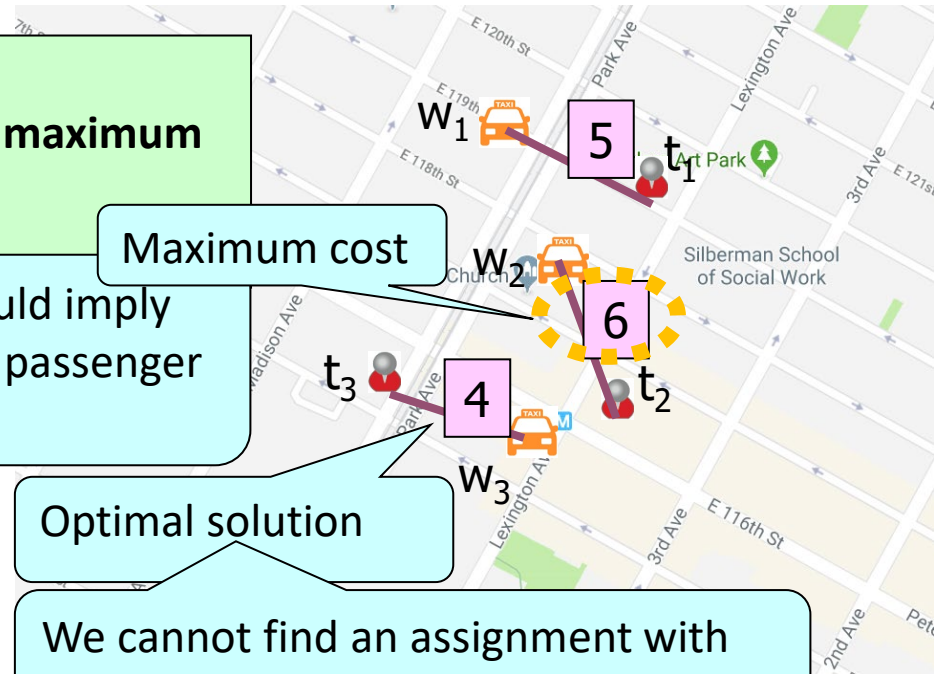
Find an assignment such that the **maximum cost** for a task is the **smallest**

Mapping back the taxi service, this could imply that the worst-case waiting time for a passenger is optimized

Maximum cost

Optimal solution

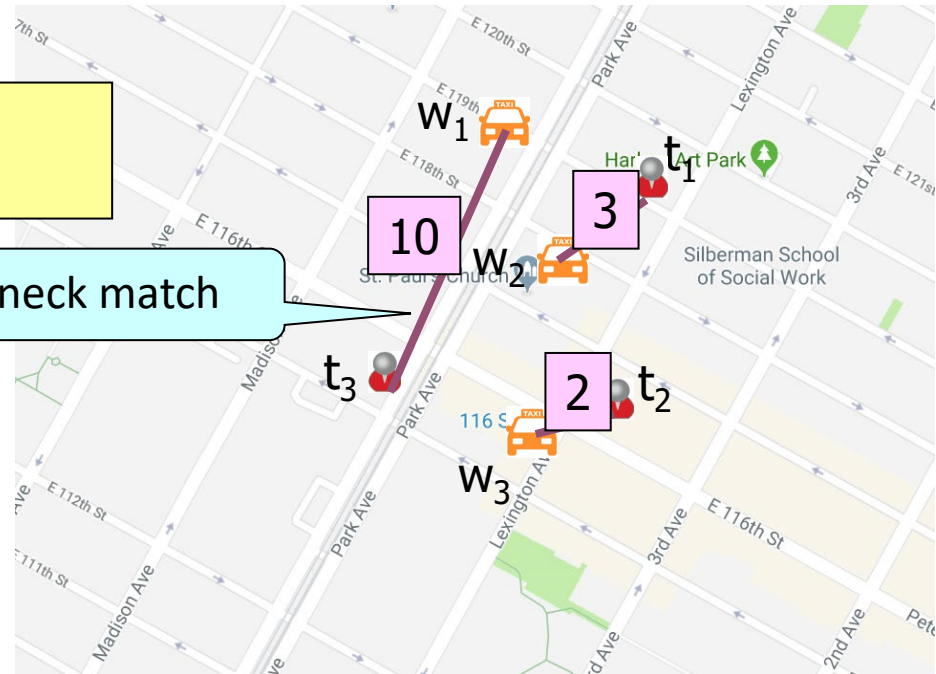
We cannot find an assignment with its maximum cost smaller than 6



Spatial Matching: Swap-Chain

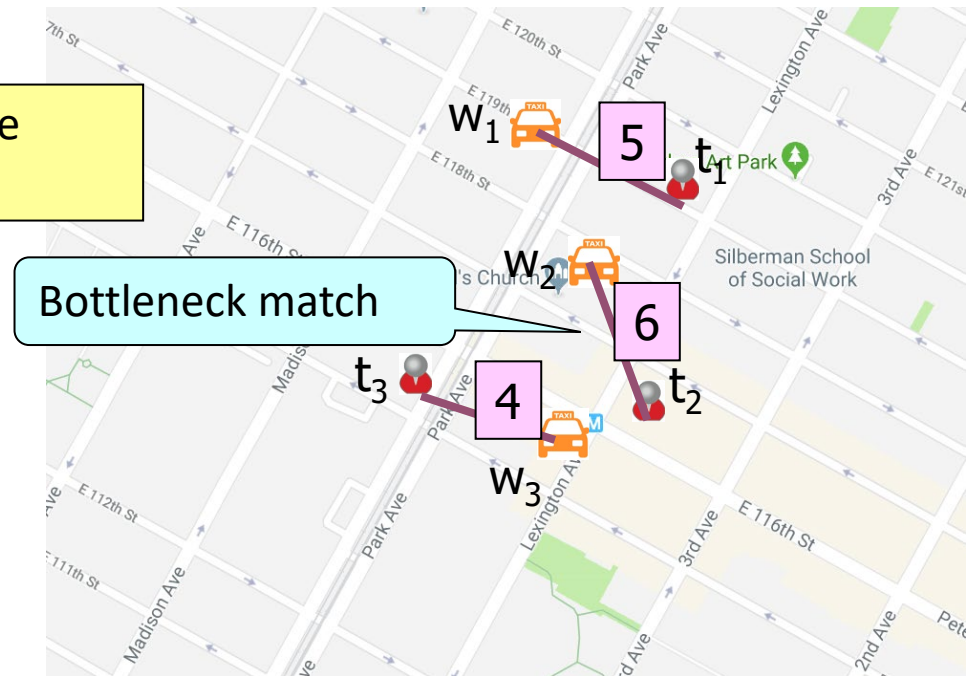
Bottleneck match: the match with the maximum cost

Bottleneck match



Spatial Matching: Swap-Chain

Bottleneck match: the match with the maximum cost



Spatial Matching: Swap-Chain

Algorithm (Swap-Chain):

Initialize an assignment

Repeat

Adjust the current assignment such that the new assignment has a smaller maximum cost

Until this is not possible

Assignment Initialization

Assignment Adjustment

Spatial Matching: Swap-Chain

Algorithm (Swap-Chain):

Initialize an assignment

Repeat

Adjust the current assignment such that the

Assignment Initialization

Different trade-offs between the workload and the quality of initialized assignment (max. cost)

They do not affect the correctness of the algorithm

Method 1:

Assign for each task a **random** worker that has not been matched

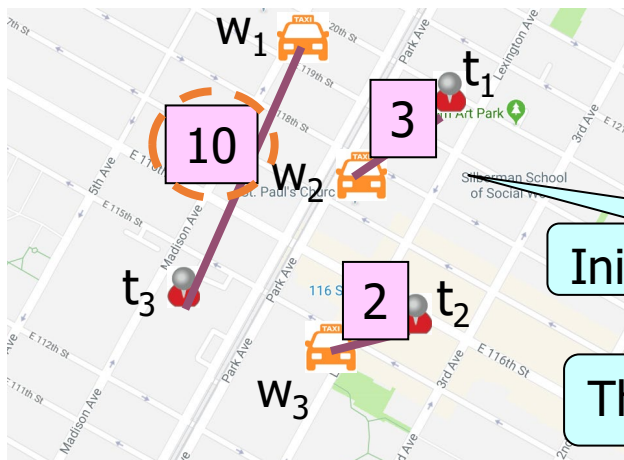
Method 2:

Assign for each task the **nearest** worker that has not been matched

Spatial Matching: Swap-Chain

Method 2:

Assign for each task the **nearest** worker that has not been matched



Initialized assignment

This is **NOT** an optimal solution!

Spatial Matching: Swap-Chain

Algorithm (Swap-Chain):

Initialize an assignment

Assignment Initialization

Repeat

Adjust the current assignment such that the new assignment has a smaller maximum cost

Until this is not possible

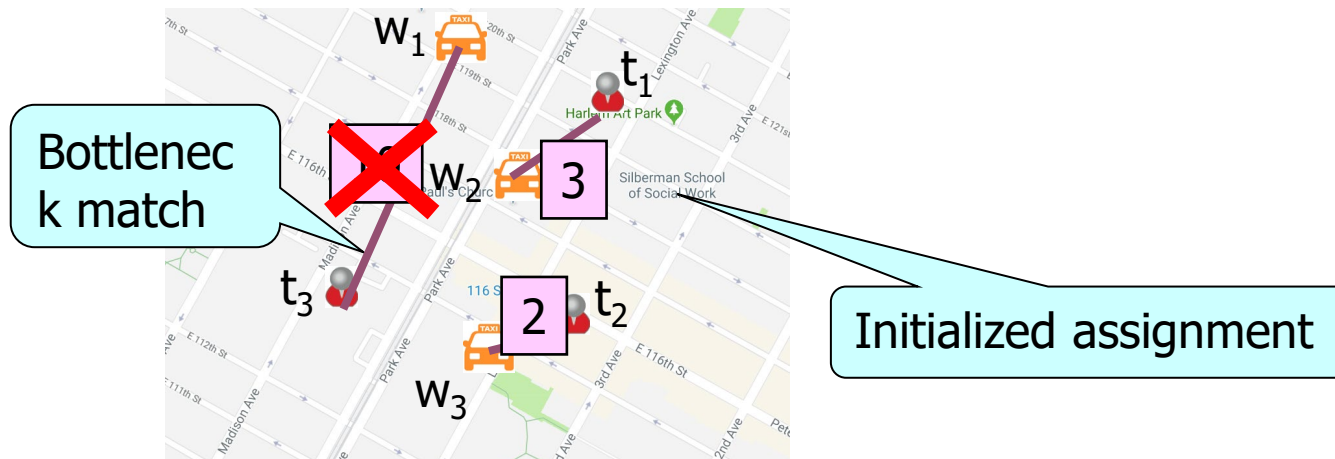
Assignment Adjustment

Spatial Matching: Swap-Chain

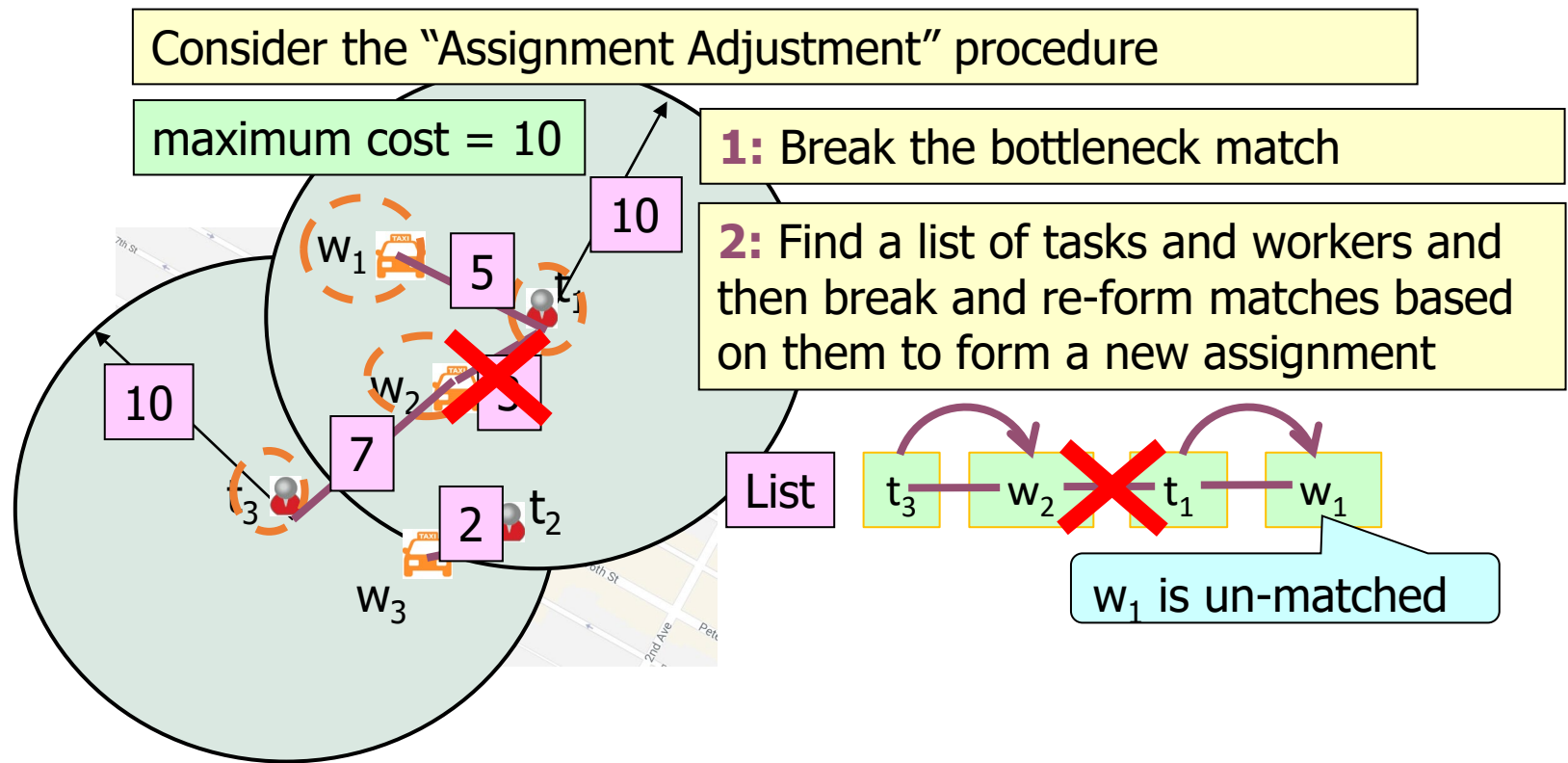
Consider the “Assignment Adjustment” procedure

maximum cost = 10

1: Break the bottleneck match



Spatial Matching: Swap-Chain

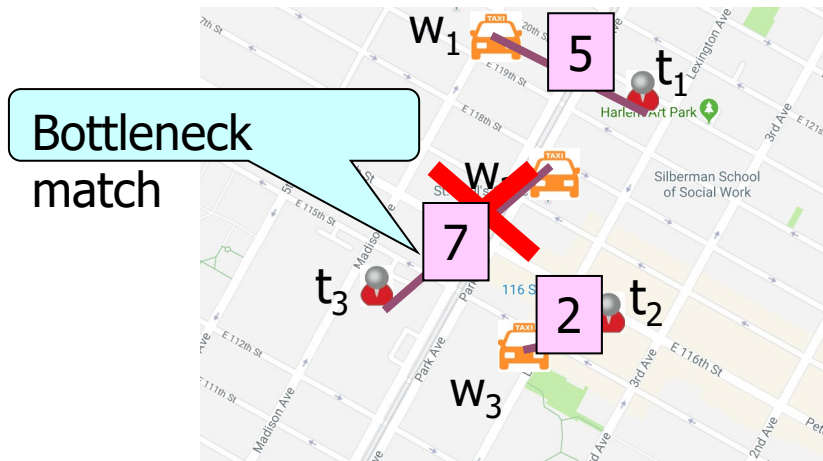


Spatial Matching: Swap-Chain

It repeats the "Assignment Adjustment" procedure

maximum cost = 7

1: Break the bottleneck match



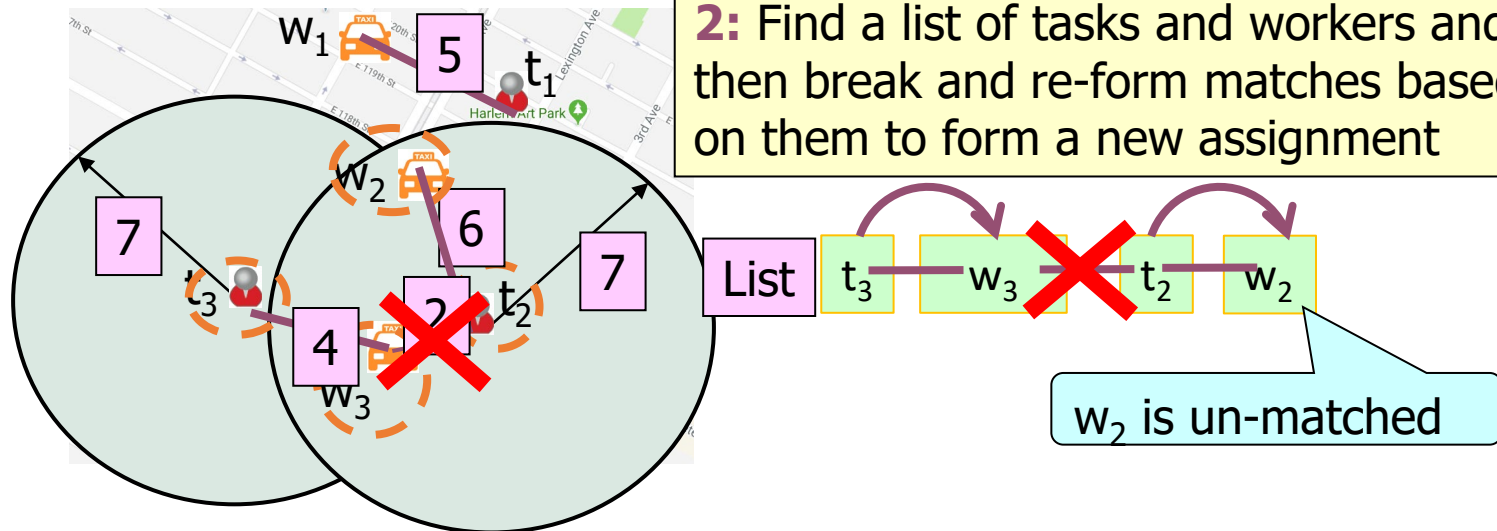
Spatial Matching: Swap-Chain

It repeats the “Assignment Adjustment” procedure

maximum cost = 7

1: Break the bottleneck match

2: Find a list of tasks and workers and then break and re-form matches based on them to form a new assignment

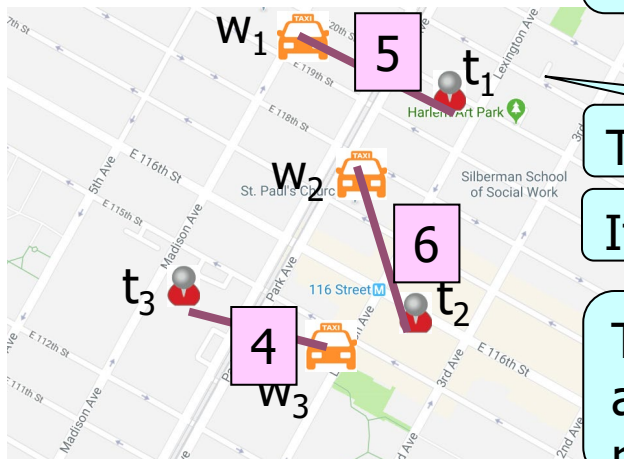


Spatial Matching: Swap-Chain

It cannot do the "Assignment Adjustment" procedure

maximum cost = 6

It cannot find a list if it breaks the bottleneck match



The final solution

It is proved this is the **optimal** solution!

This is interesting since Swap-Chain adopts a **local search** strategy but returns an **optimal** solution

Recap

- Urban Data
- Urban Data Indexing (Spatial)
- Urban Data Query Processing (Spatial)
- Spatial Matching

Next Lecture

Part 2 – 02: Urban Data Management (2) (Spatio-Temporal and Network Data)