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

Long Cheng

Assistant Professor

c.long@ntu.edu.sg

Urban Computing: Overview

Part 1

Part 2

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

Urban Computing: Overview

Two lectures

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

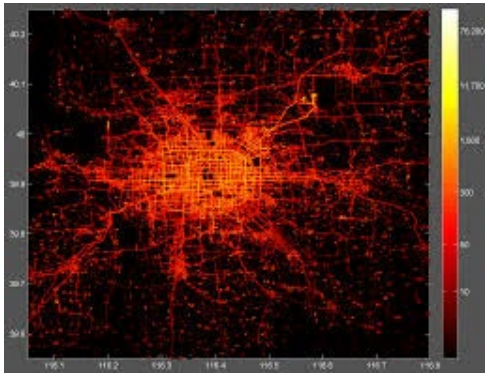
Urban data

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

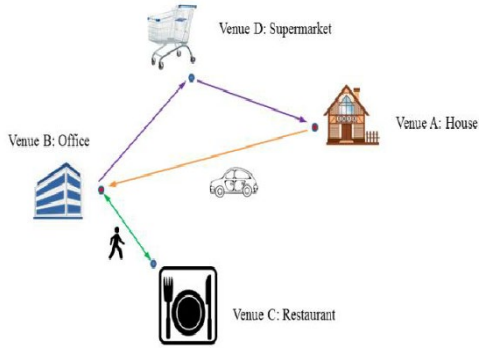
Urban Data: Examples

Point-of-interest

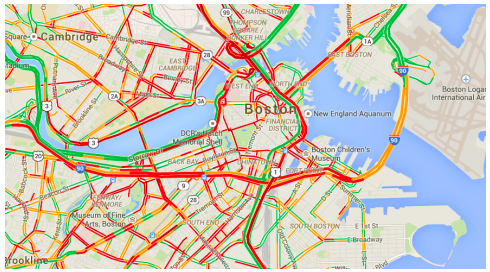
Road network



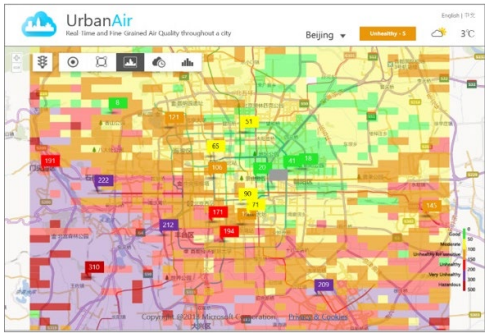
Vehicle trajectories



User mobility



Road traffic



Air quality

Urban Data: Categorization

Spatial Data

- Coordinates (location)
- Fixed

Spatio-temporal Data

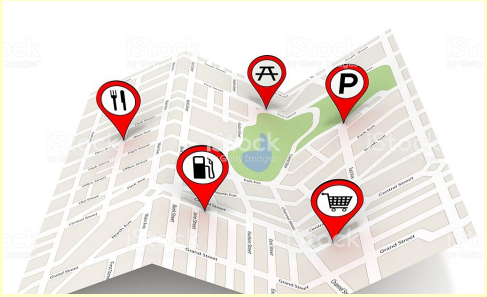
- Coordinates (location)
- Time stamps
- Dynamic

Network Data

- Nodes and edges
- Fixed structure
- Dynamic features

Urban Data: Categorization

Spatial Data

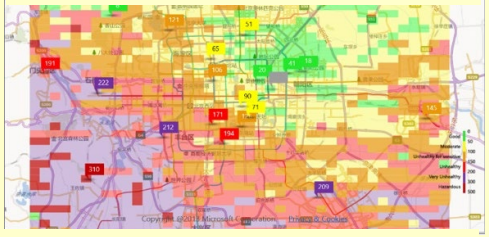


Point-of-interest

Road network

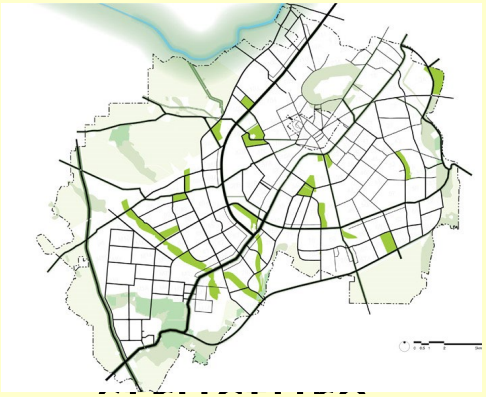
Spatio-temporal Data

Vehicle trajectories

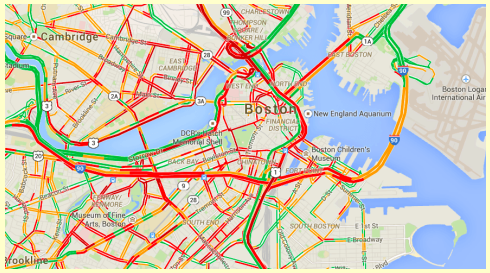


Air quality

Network Data

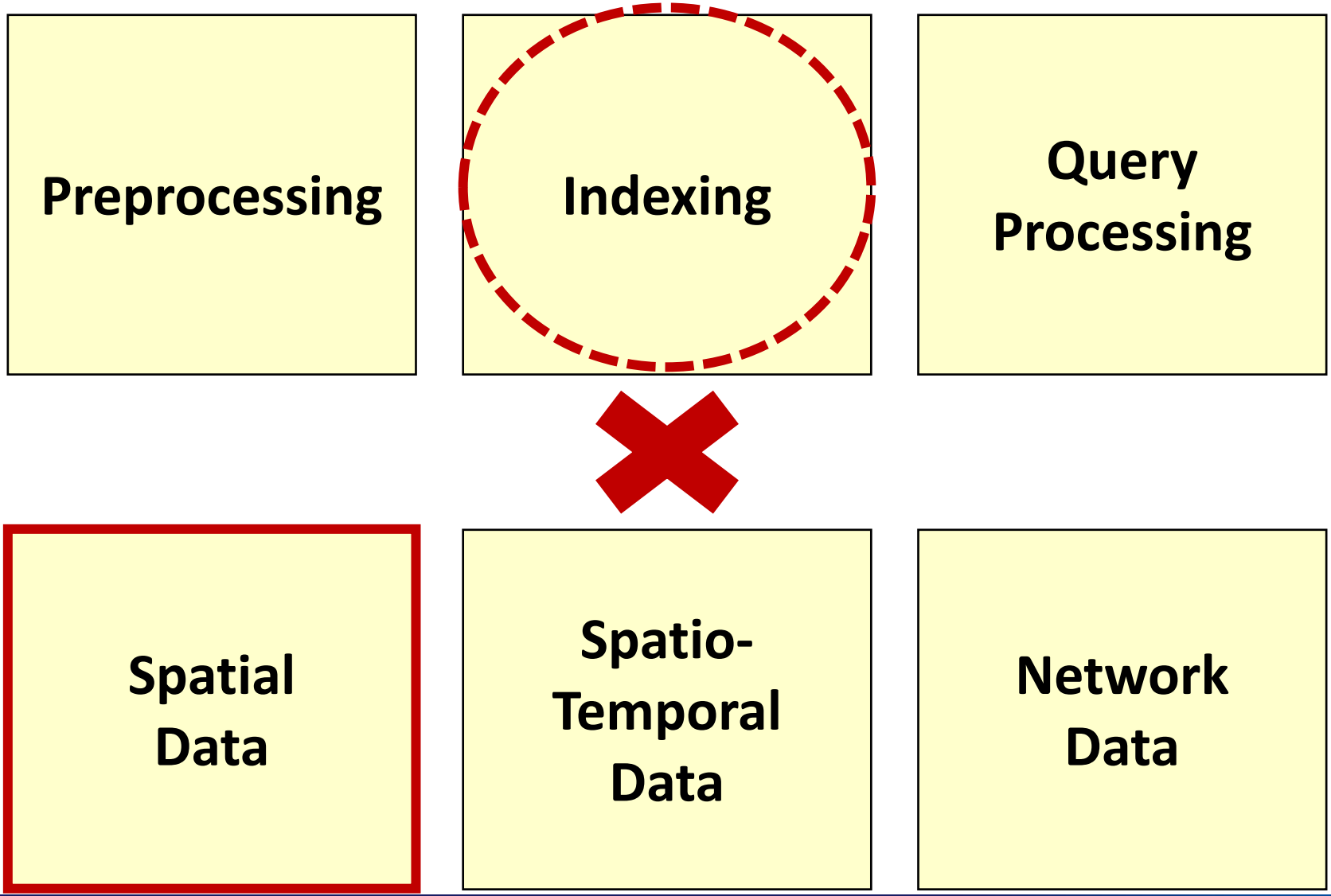


Road network



Road traffic

Urban Data Management



Indexing: Motivation

What is the nearest restaurant near my location?

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

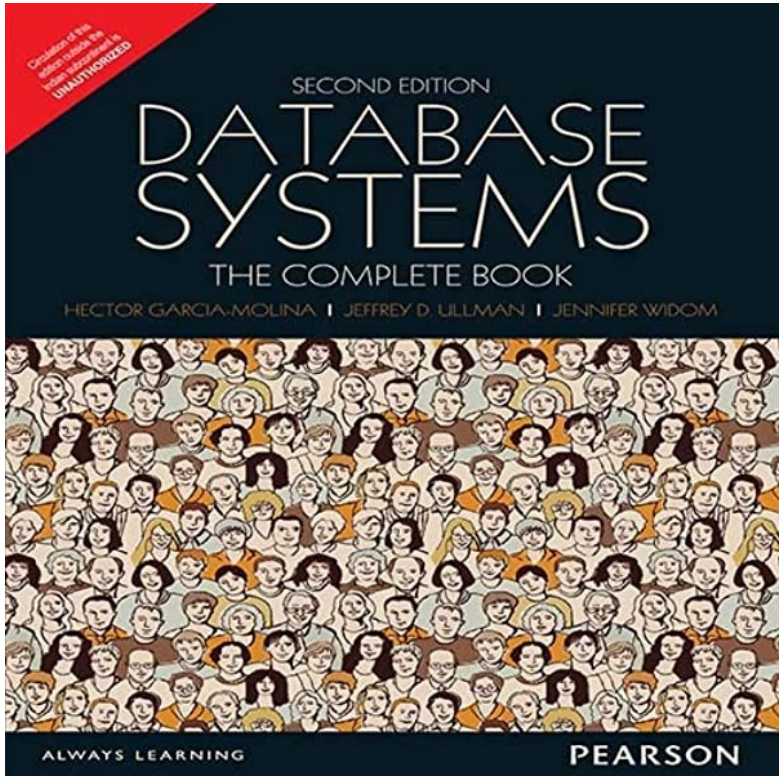
Indexing: Motivation

What is the nearest restaurant near my location?

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

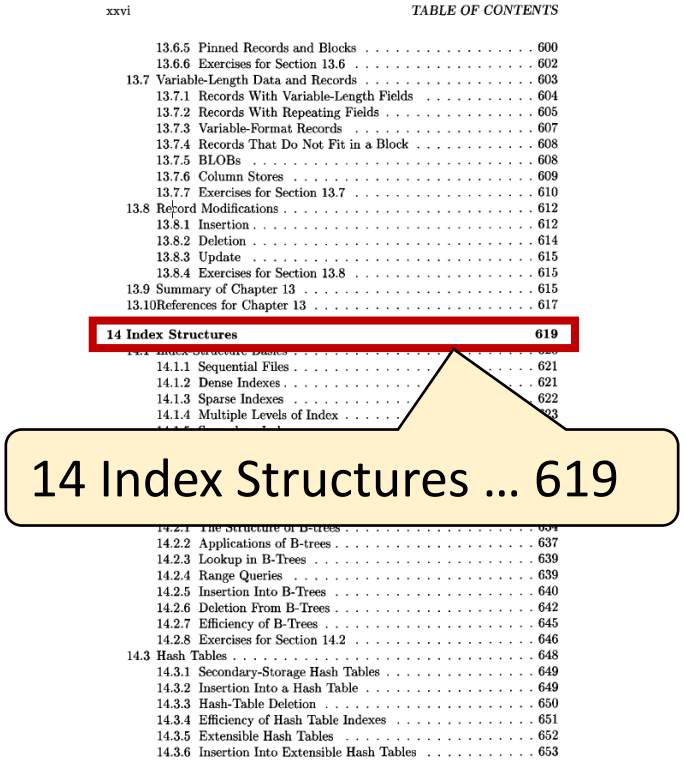
Indexing: Motivation

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**!!



Indexing (Spatial Data)

R tree

- Hierarchical collection of rectangles organizing spatial data

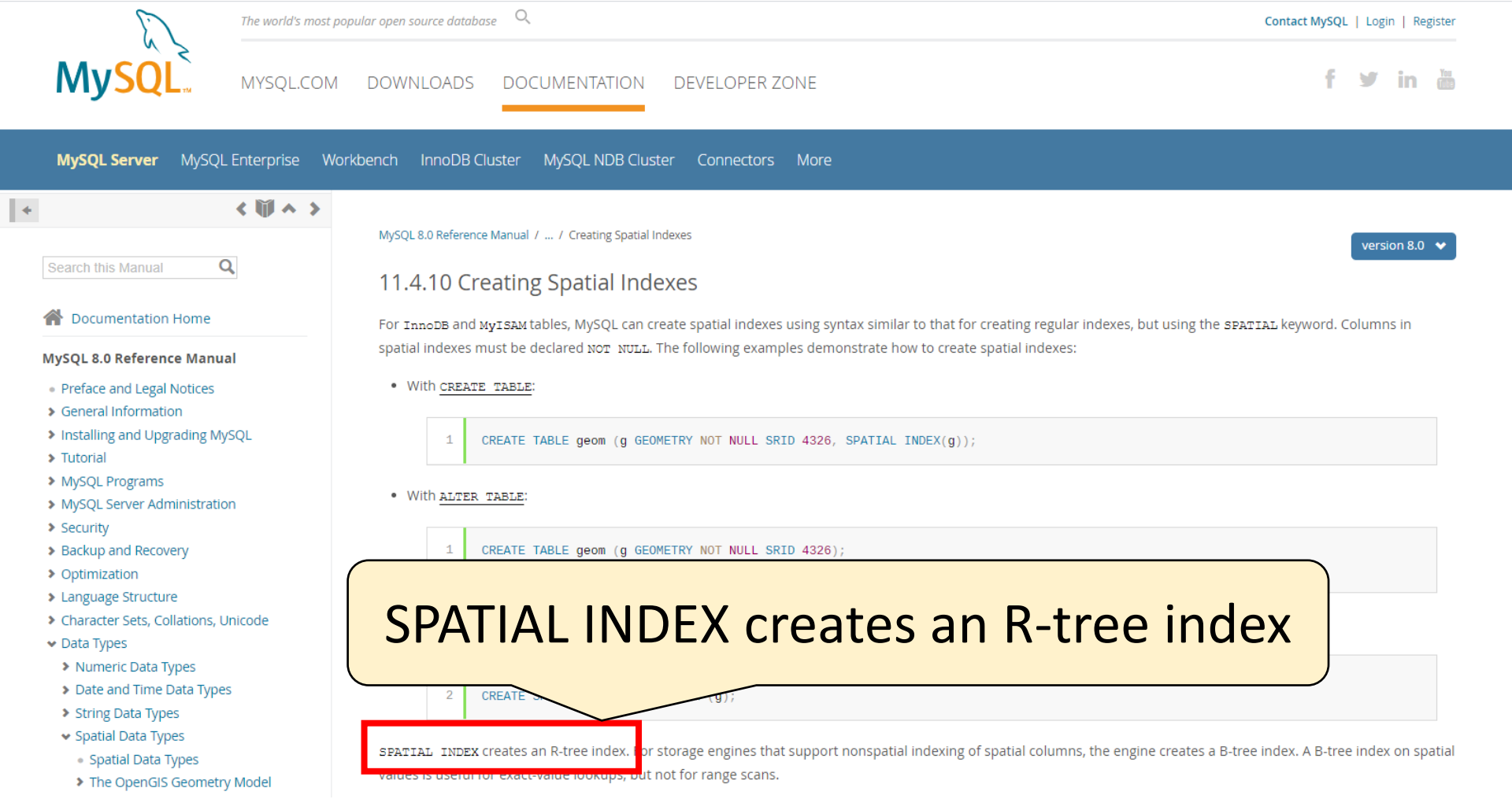
k-d tree

- 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



SPATIAL INDEX creates an R-tree index

R Tree

ORACLE Help Center

Sign In

Home / Database / Oracle Database Online Documentation 12c, Release 1 (12.1) / Unstructured Data and Content Management

Spatial and Graph Developer's Guide

Page 15

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

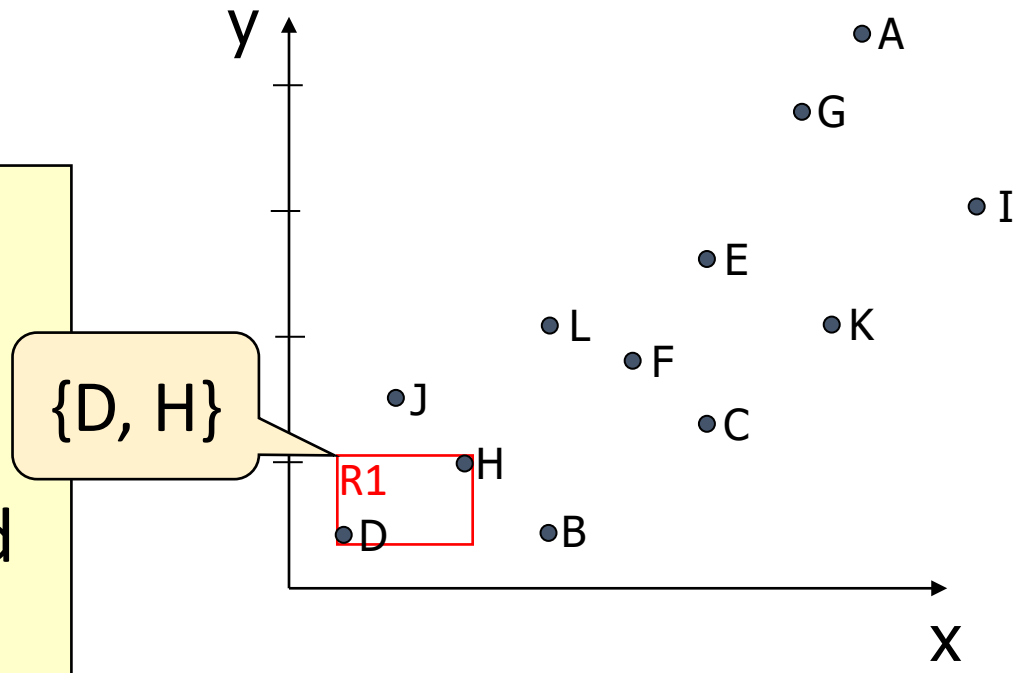
A spatial R-tree in[illegible] can index spatial data of up to four dimensions. An R-tree index approximates each geometry by a single rectangle that minimally encloses the geometry (called [illegible]um bounding rectangle, or MBR), as shown in Figure 1-3.

Figure 1-3 MBR

R-Tree Indexing

R Tree

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

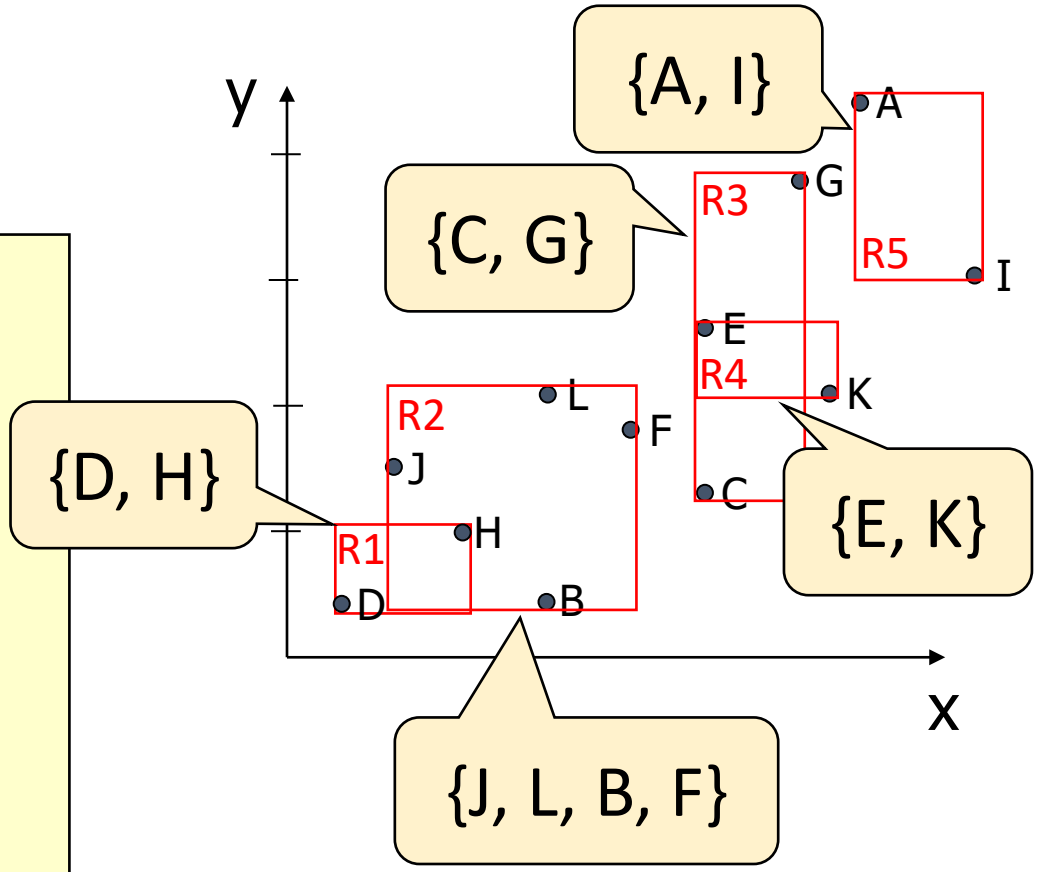


Settings => Balanced

Assume each group has
- at least 2 points
- at most 4 points

R Tree

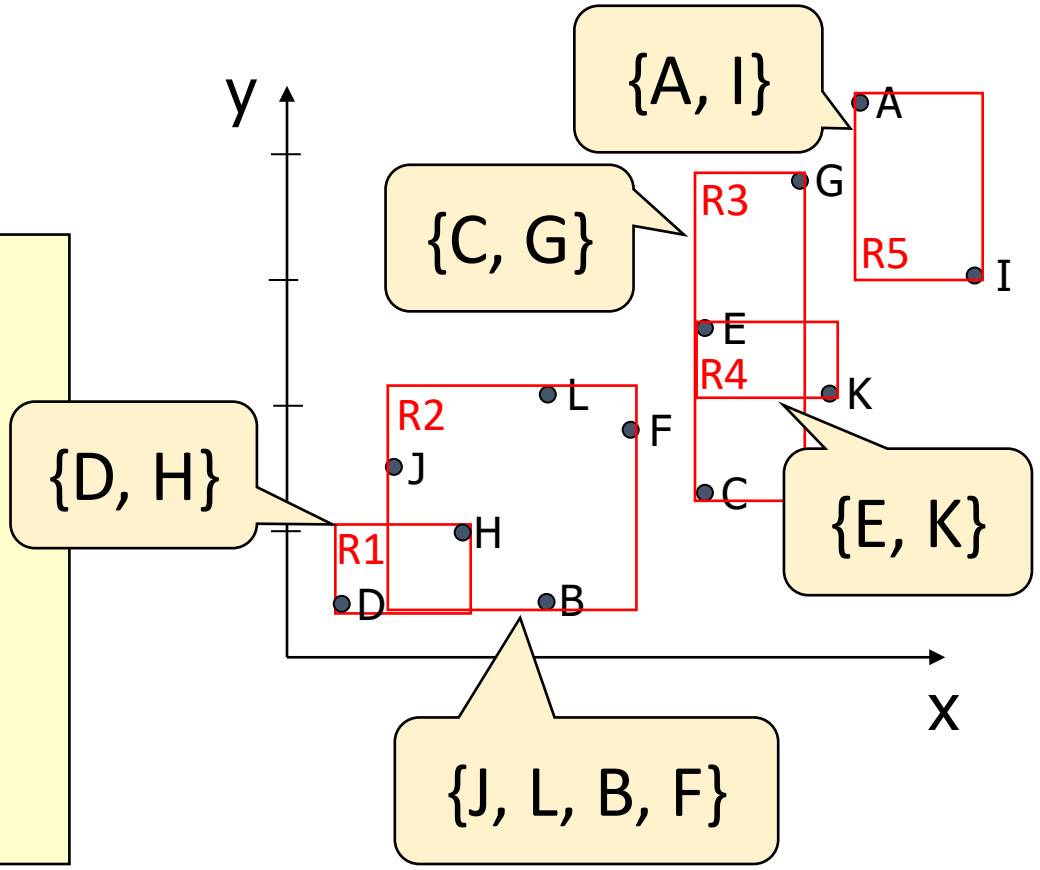
- Similarly, group other points
- Obtain corresponding MBRs

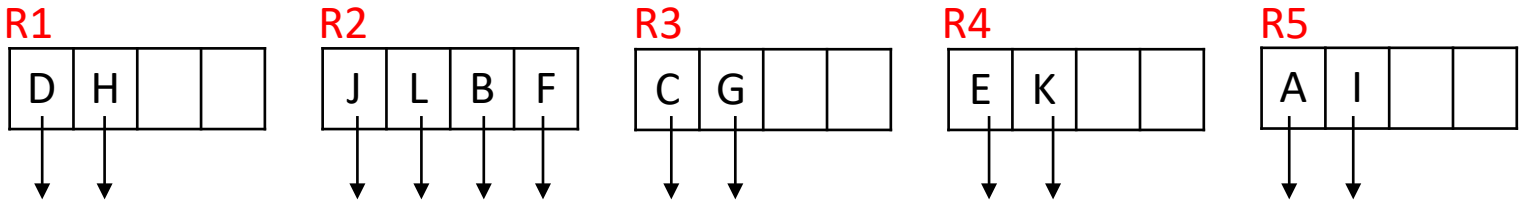


Assume each group has
- at least 2 points
- at most 4 points

R Tree

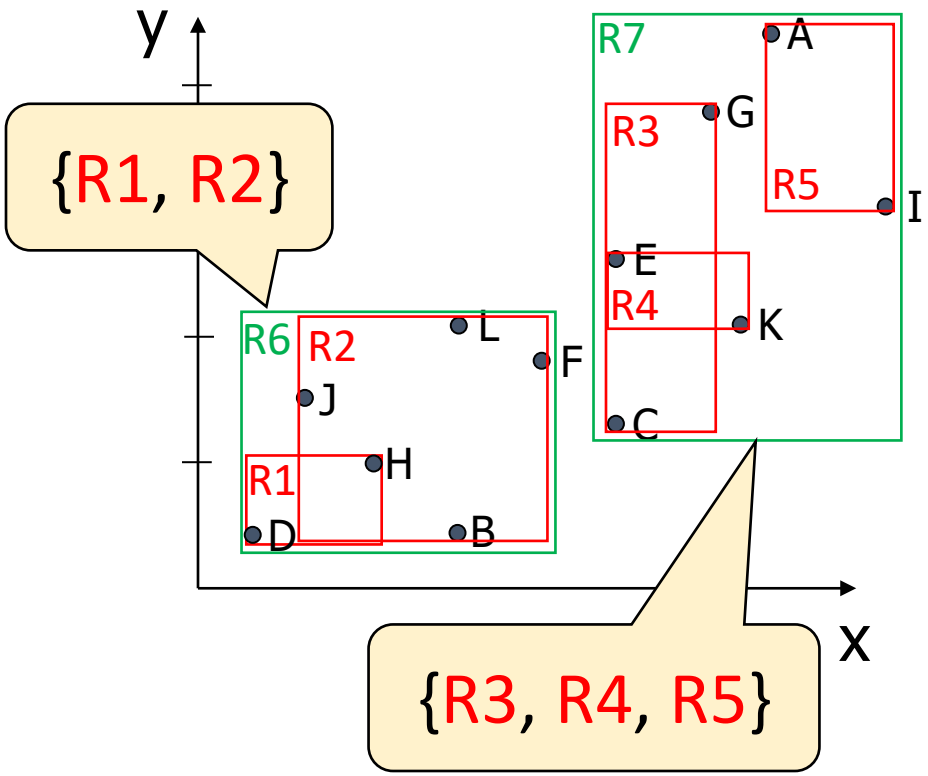
- Create a node for each group
- Each node corresponds to an **array** of pairs of <point, pointer>

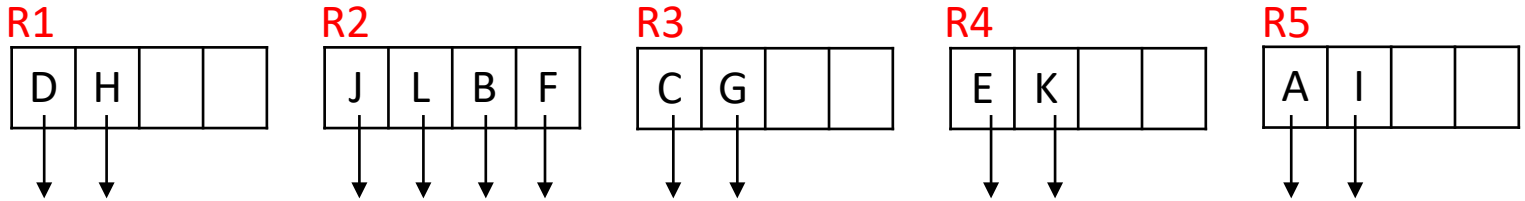




R Tree

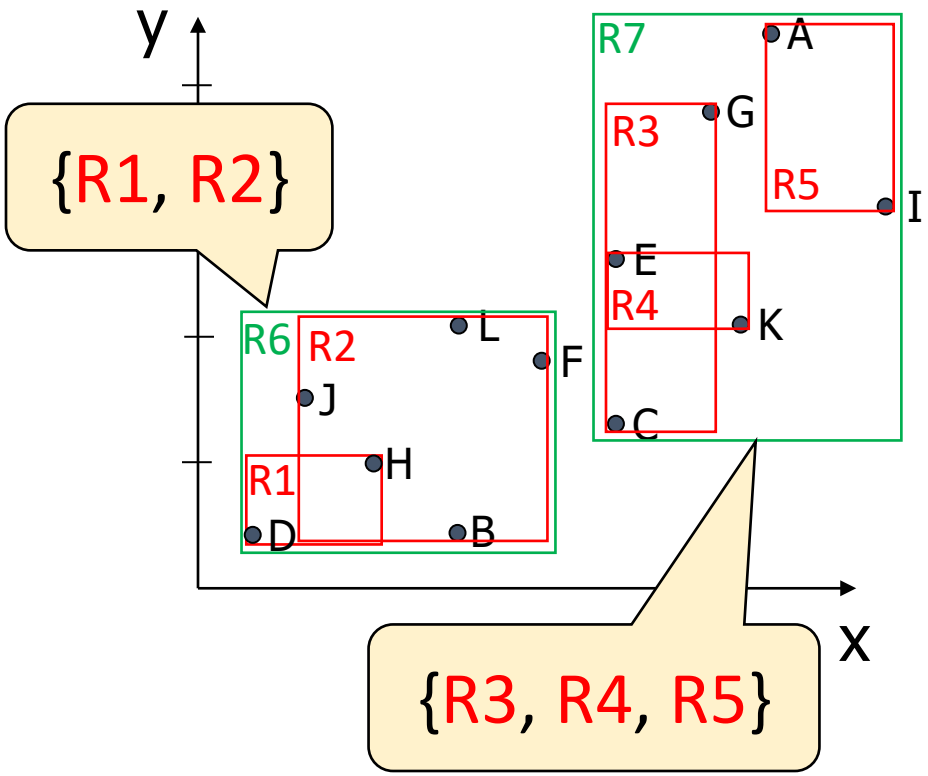
- Recursively, group the MBRs
- Obtain larger MBRs

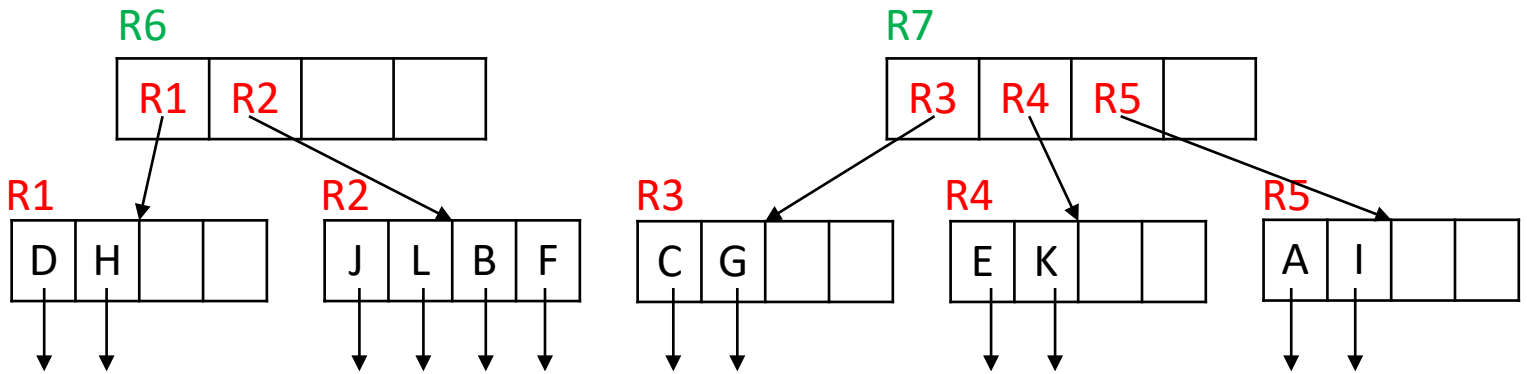




R Tree

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

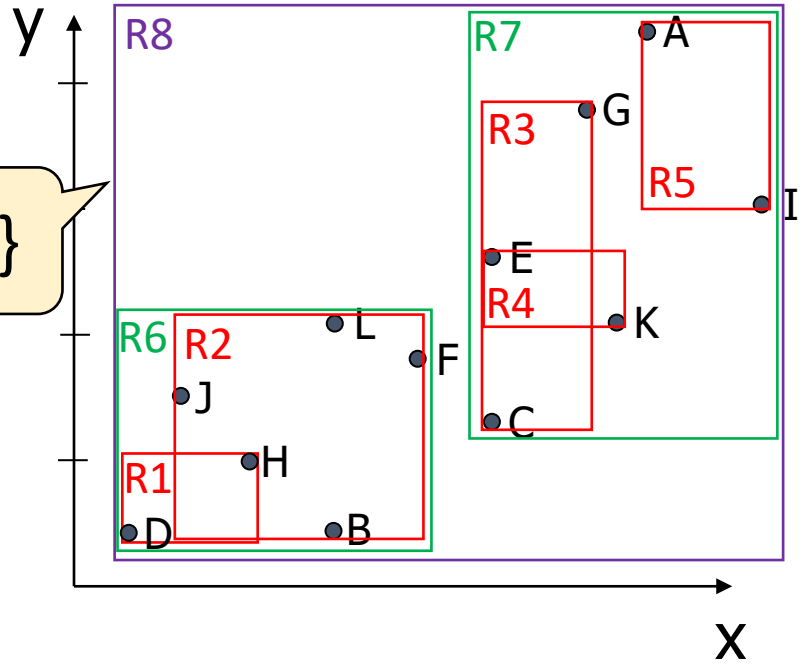


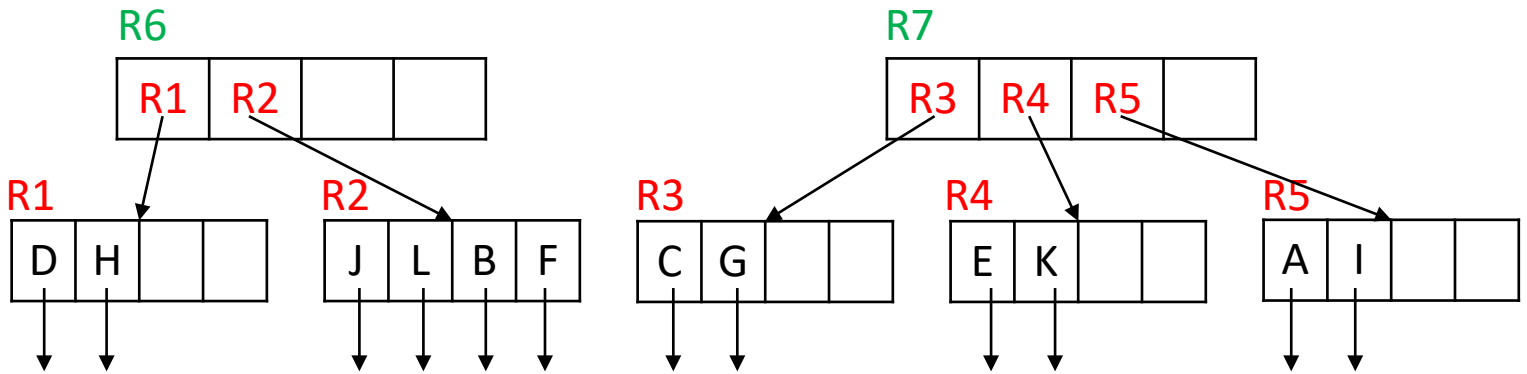


R Tree

- Recursively, group the MBRs
- Obtain larger MBRs

{R6, R7}

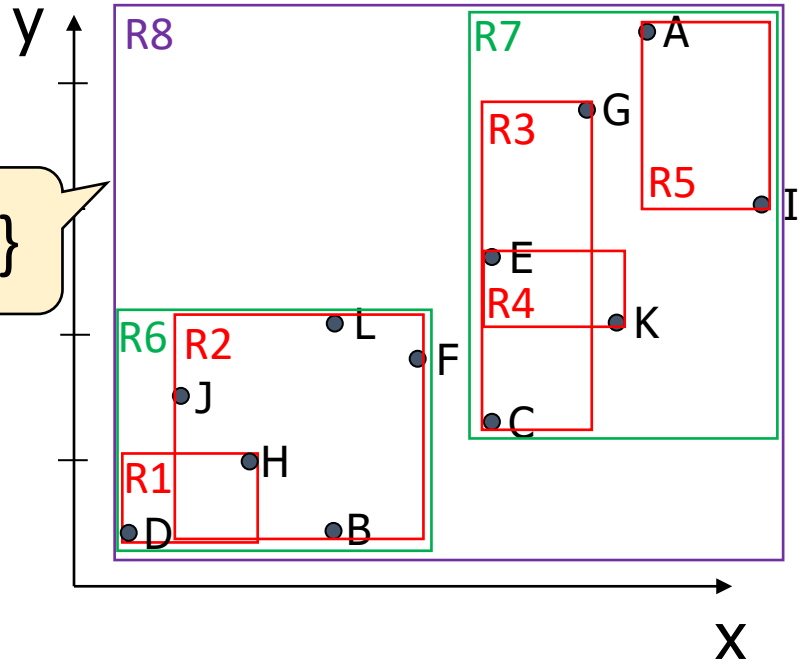


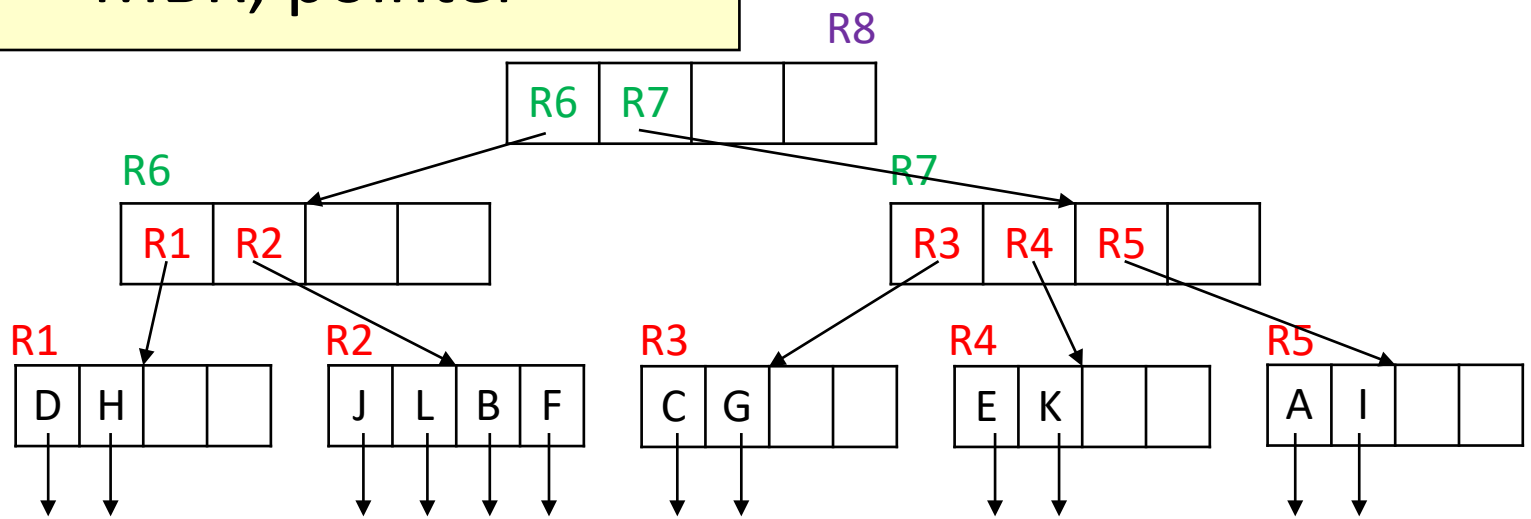


R Tree

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

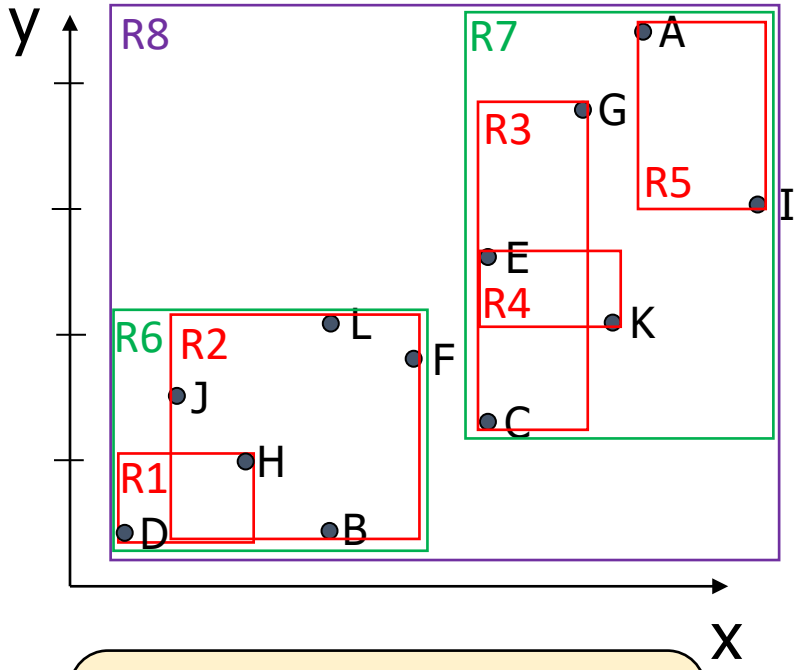
{R6, R7}



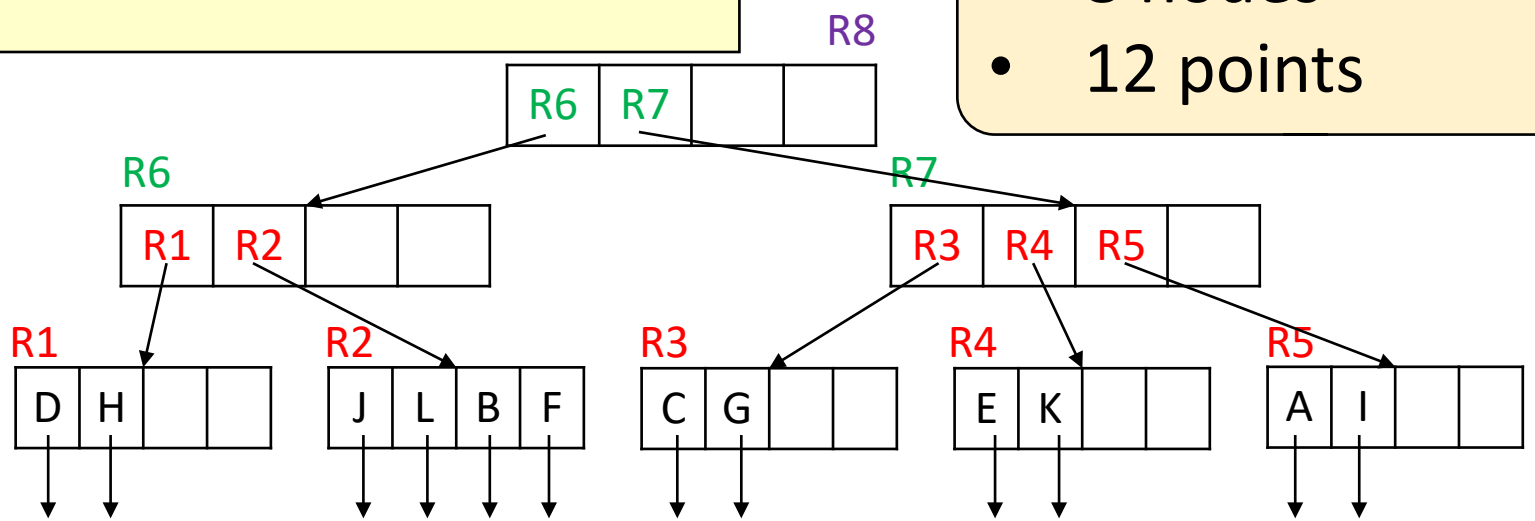


R Tree

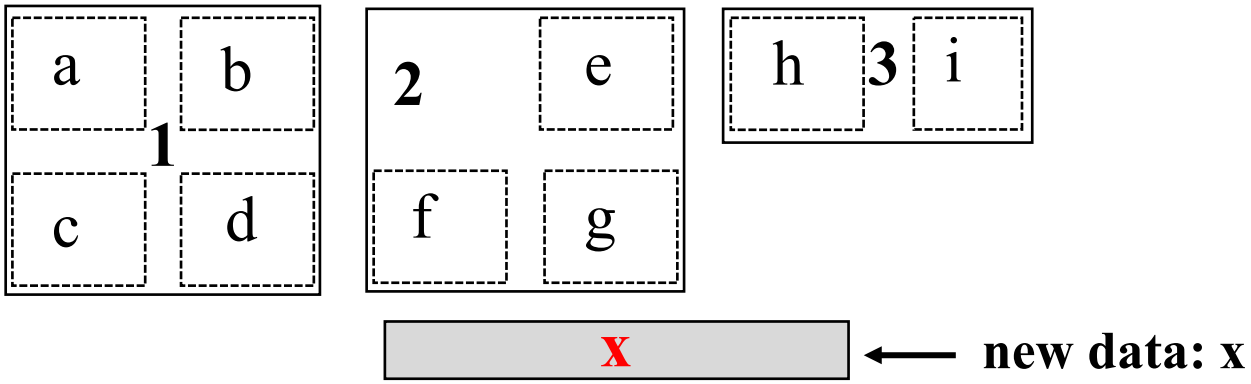
Done

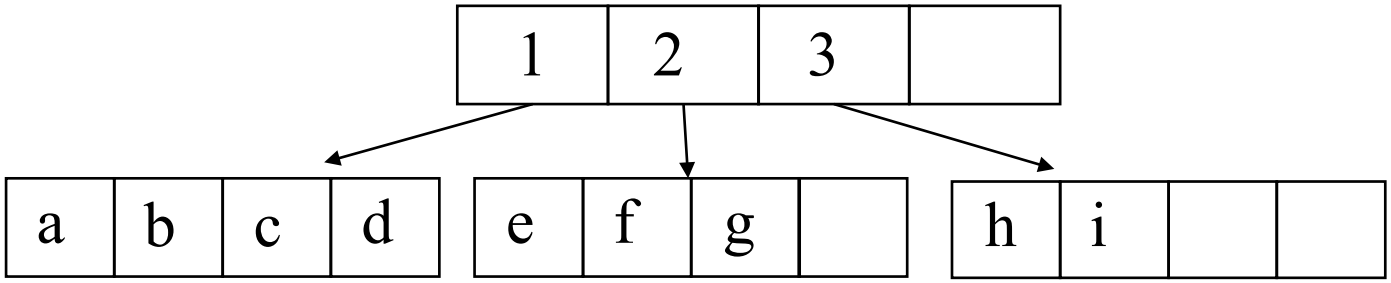


- 8 nodes
- 12 points

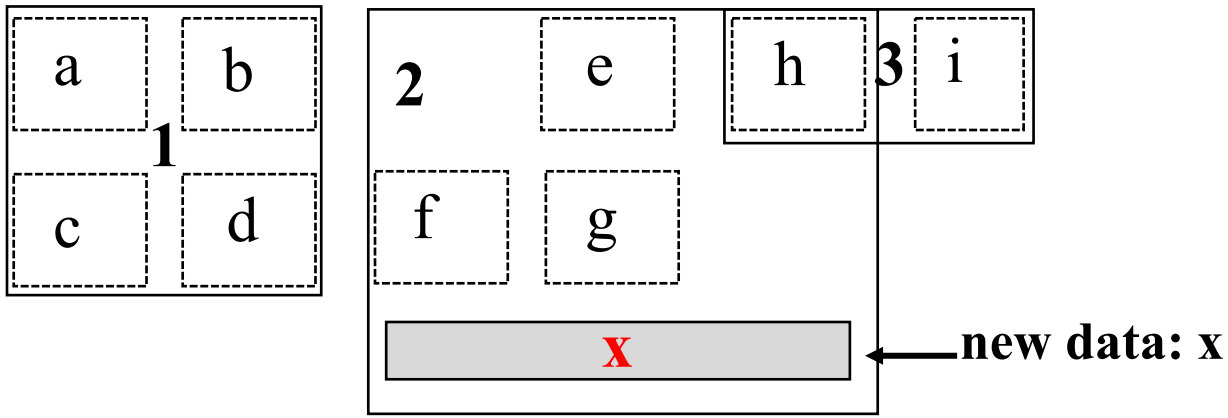


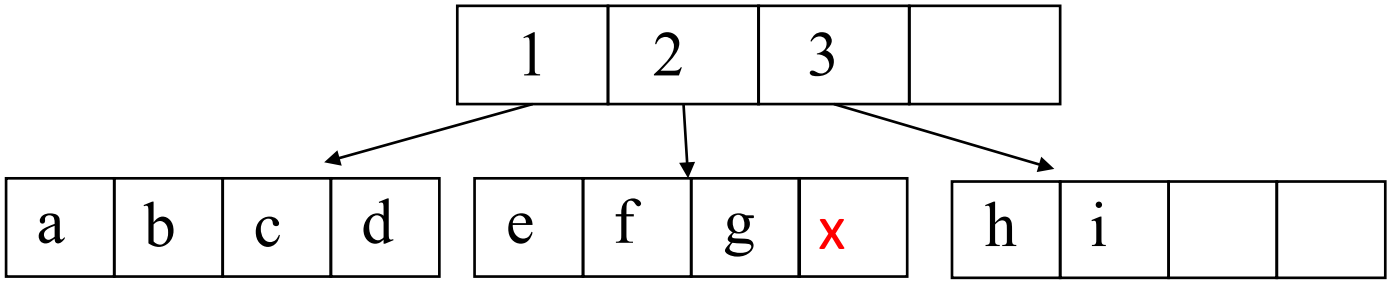
R Tree: Insertion





R Tree: Insertion

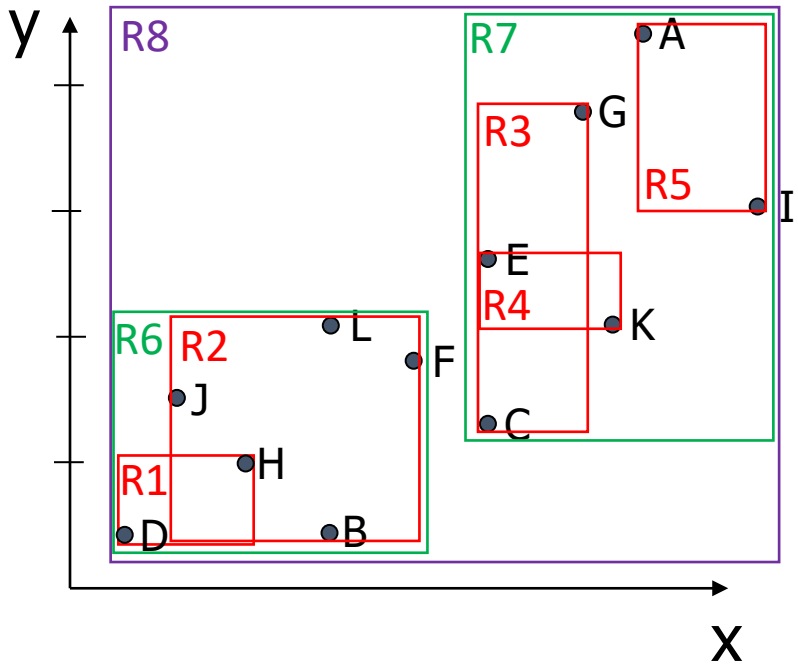


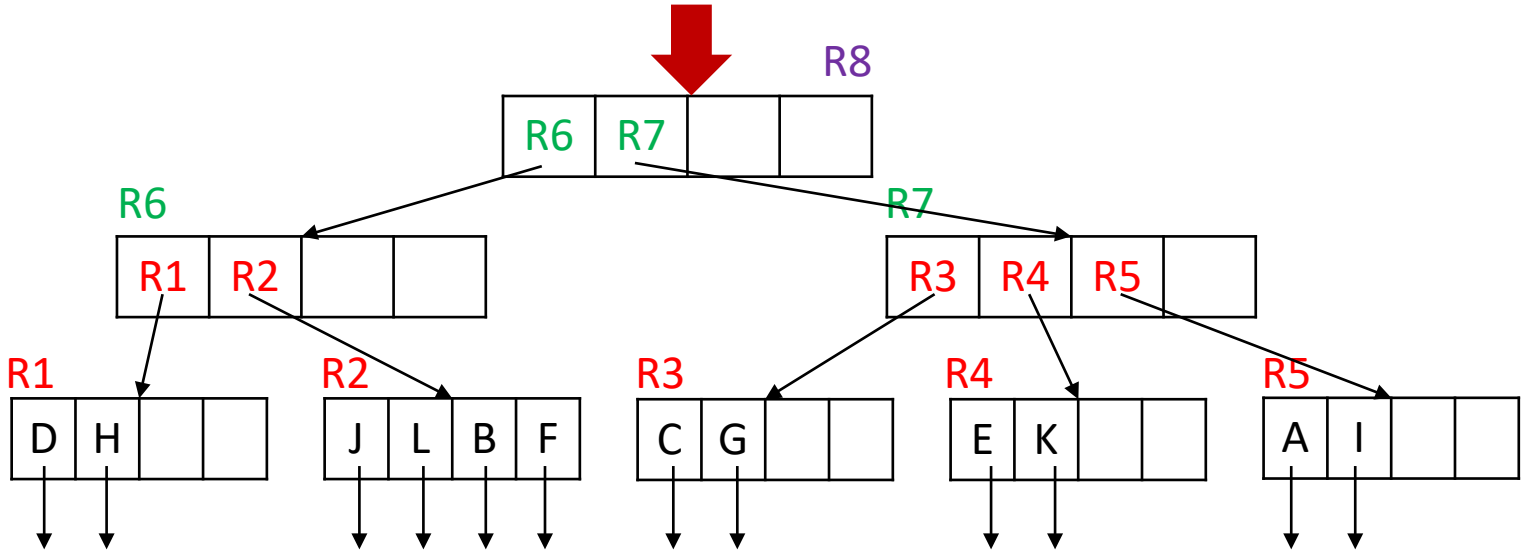


R Tree: Query Processing

Search point K

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

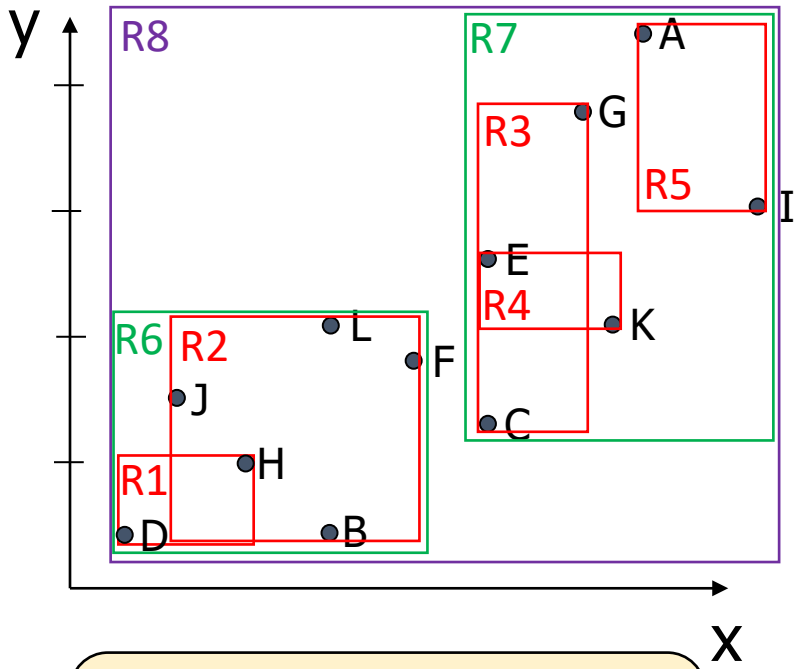




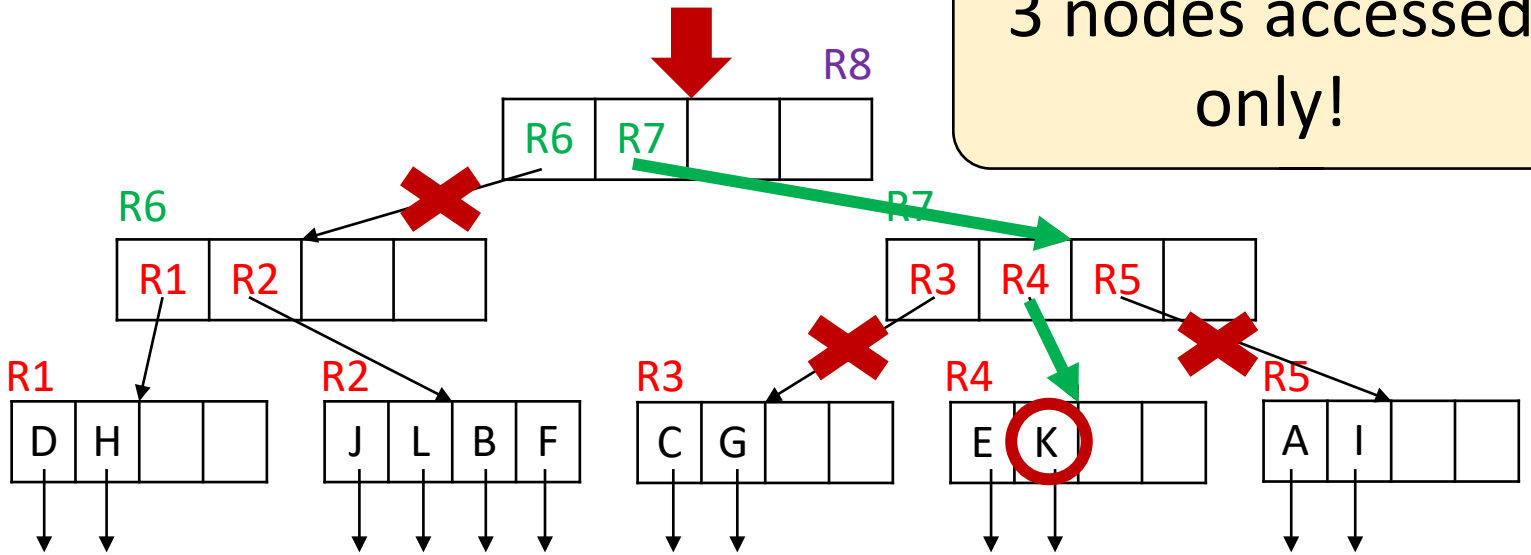
R Tree: Query Processing

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

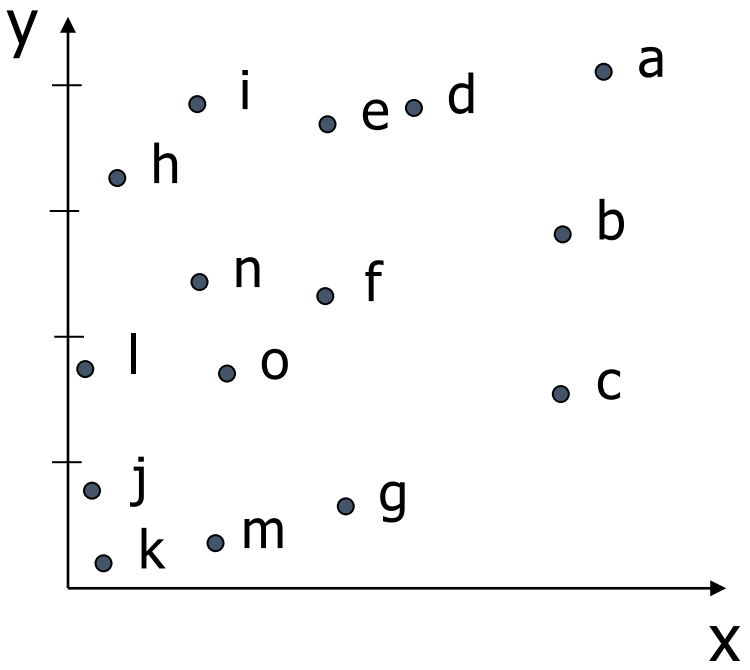
k-d tree

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

Quad tree

- Recursively partition a space into 4 regions evenly

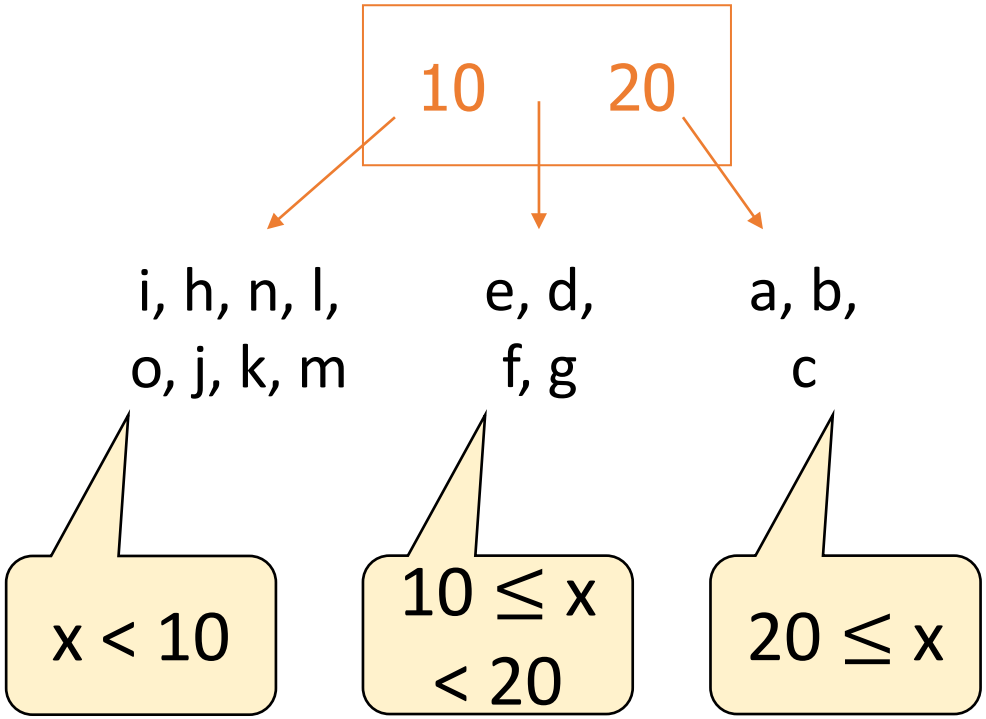
k-d Tree

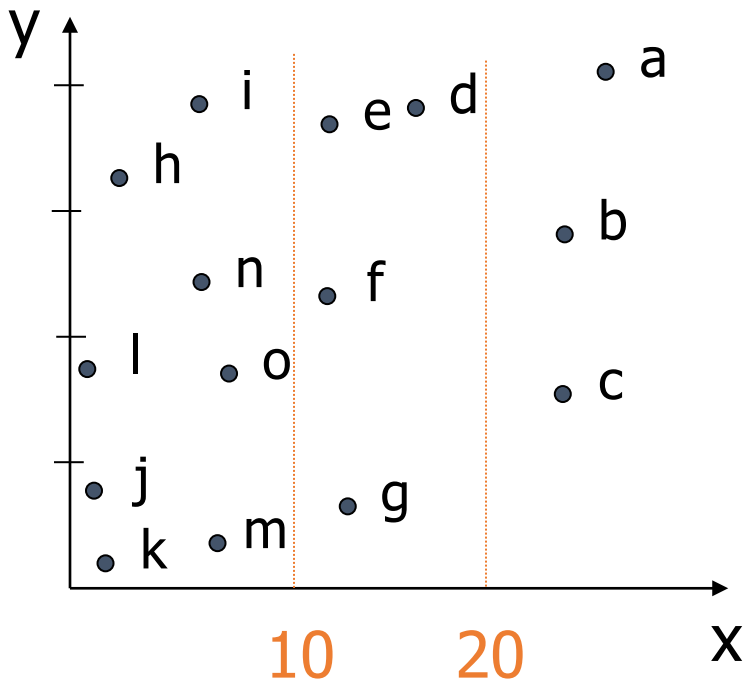


Input data

15 records

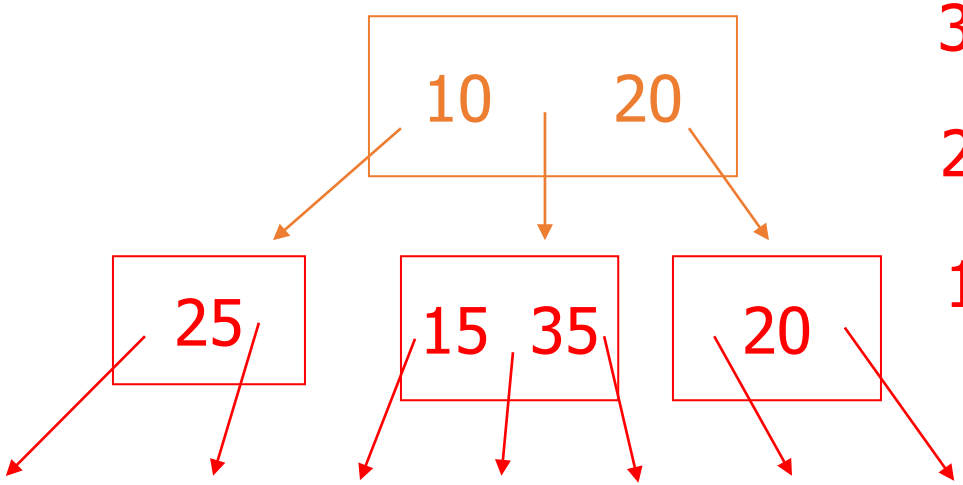
k-d Tree

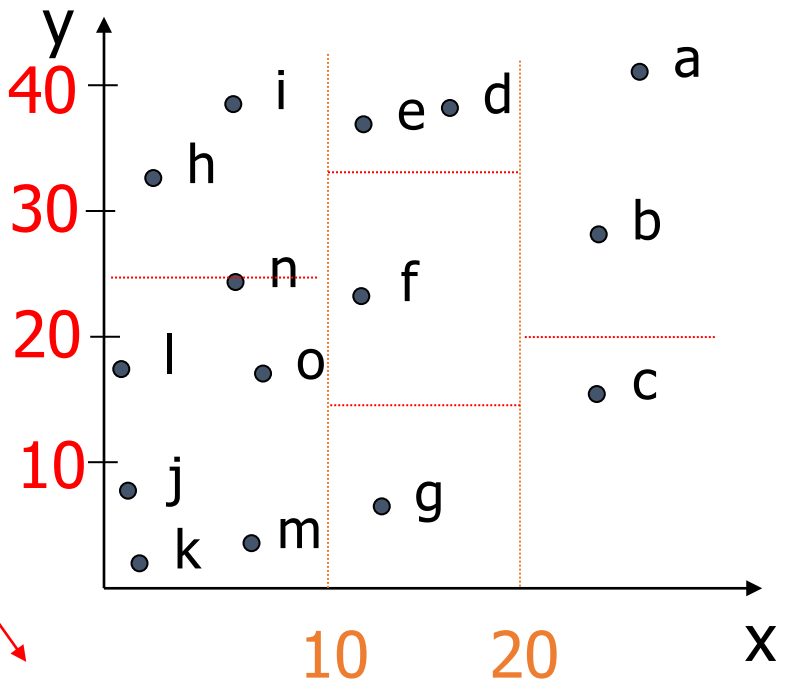




Build an index on x

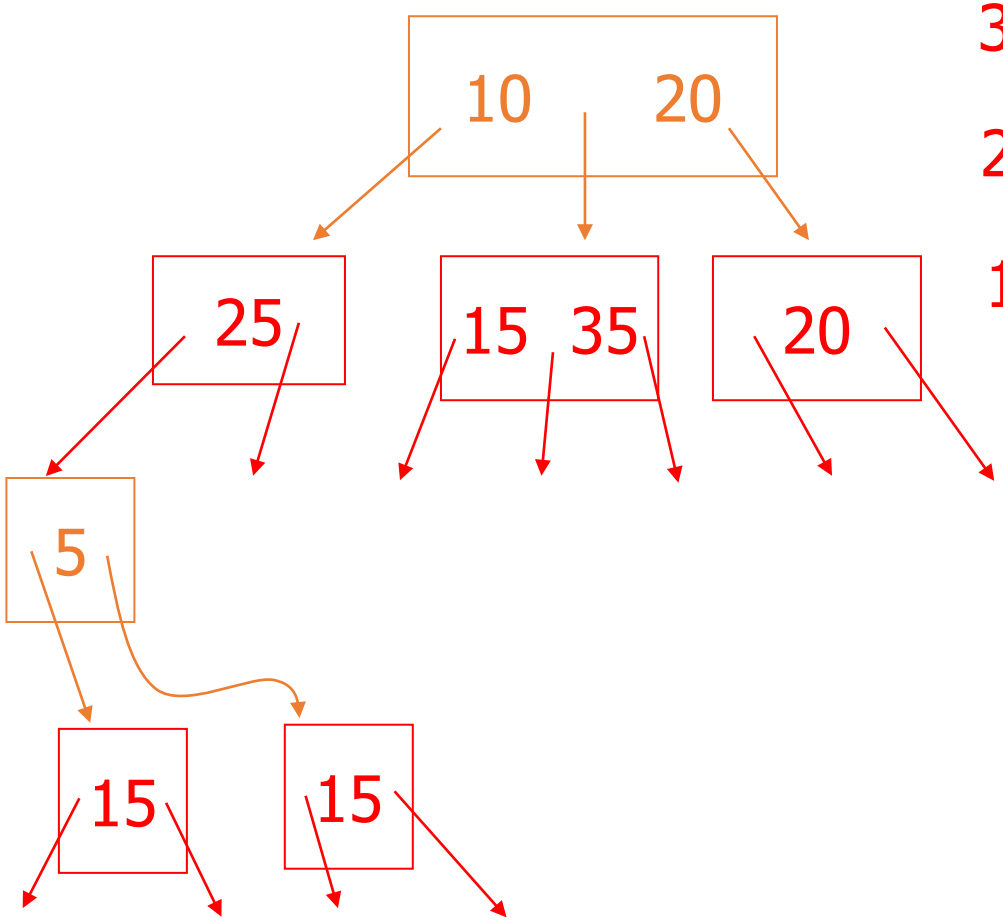
k-d Tree

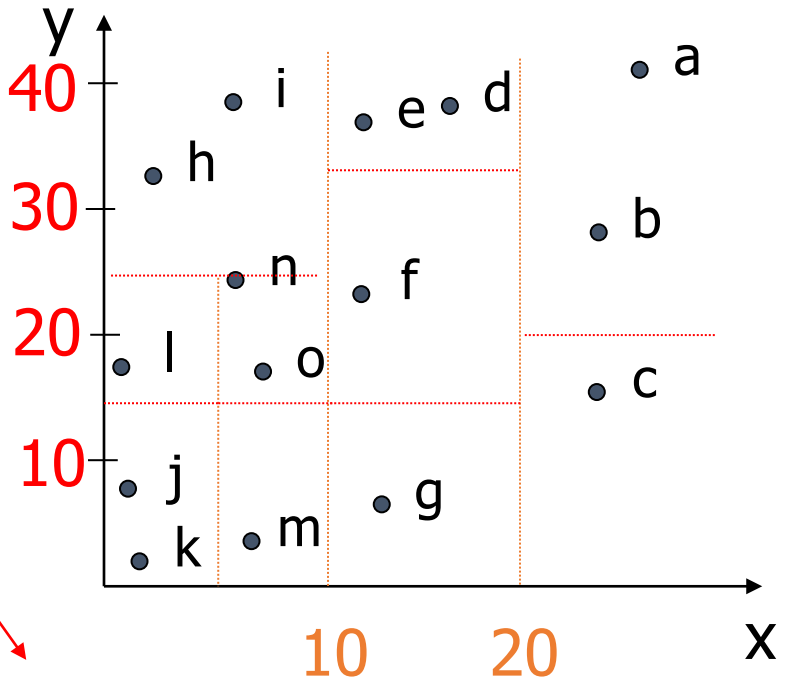




Build an index on y (for each partition of data on x)

k-d Tree

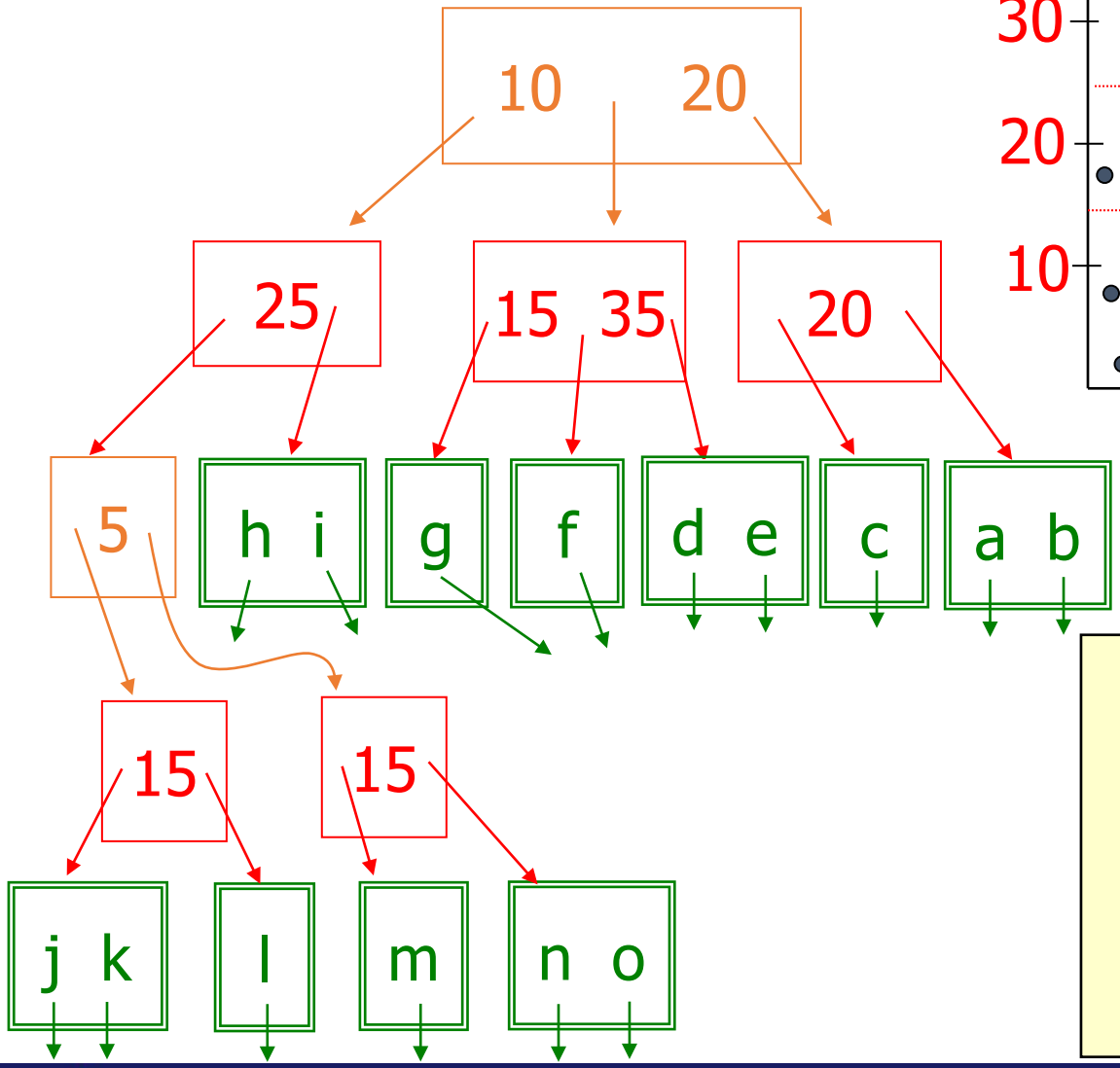


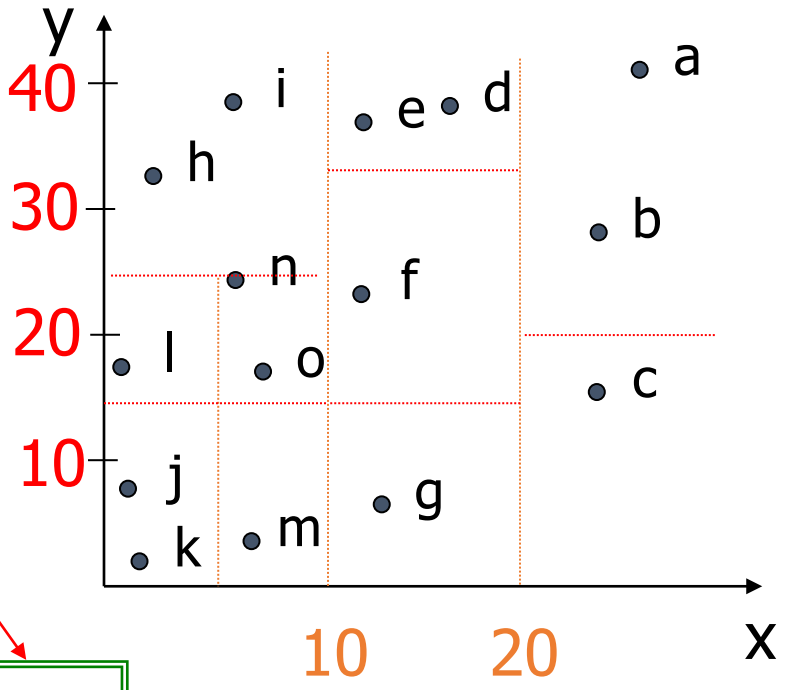


Build indexes on y and x iteratively

until each partition involves at most 2 records

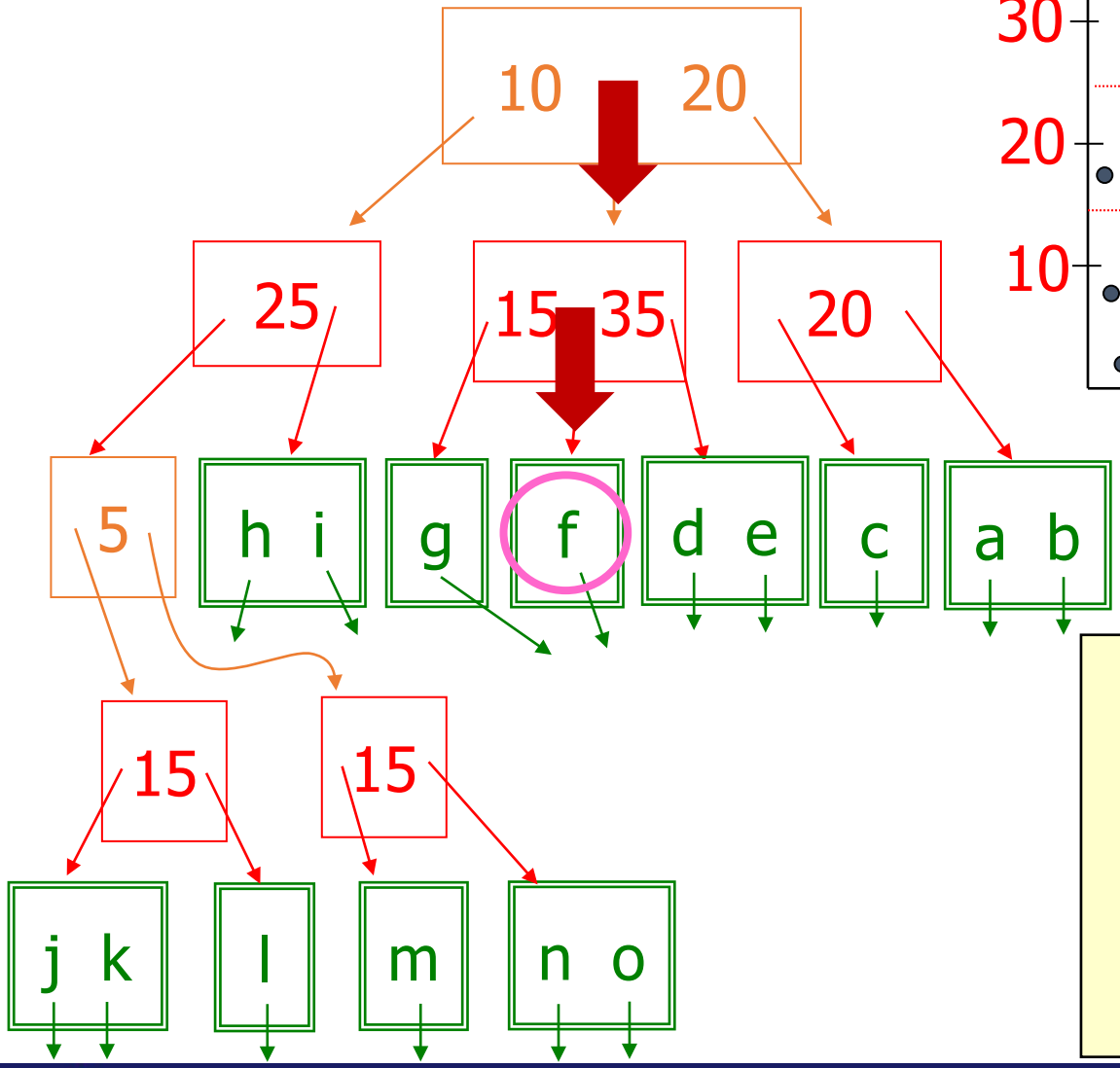
k-d Tree

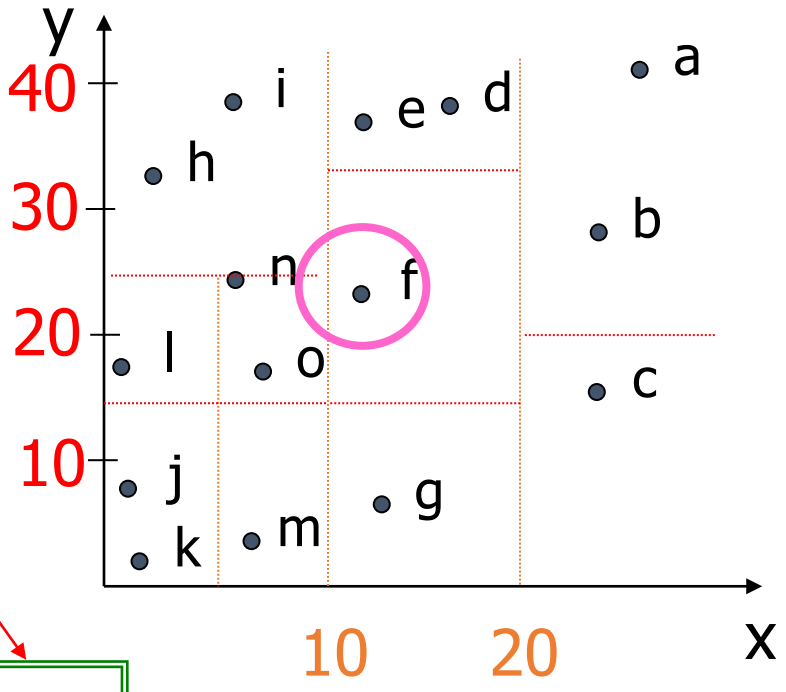




Index at the end

k-d Tree





Search the point f
<12, 24>

Indexing (Spatial Data)

R tree

- Hierarchical collection of rectangles organizing spatial data

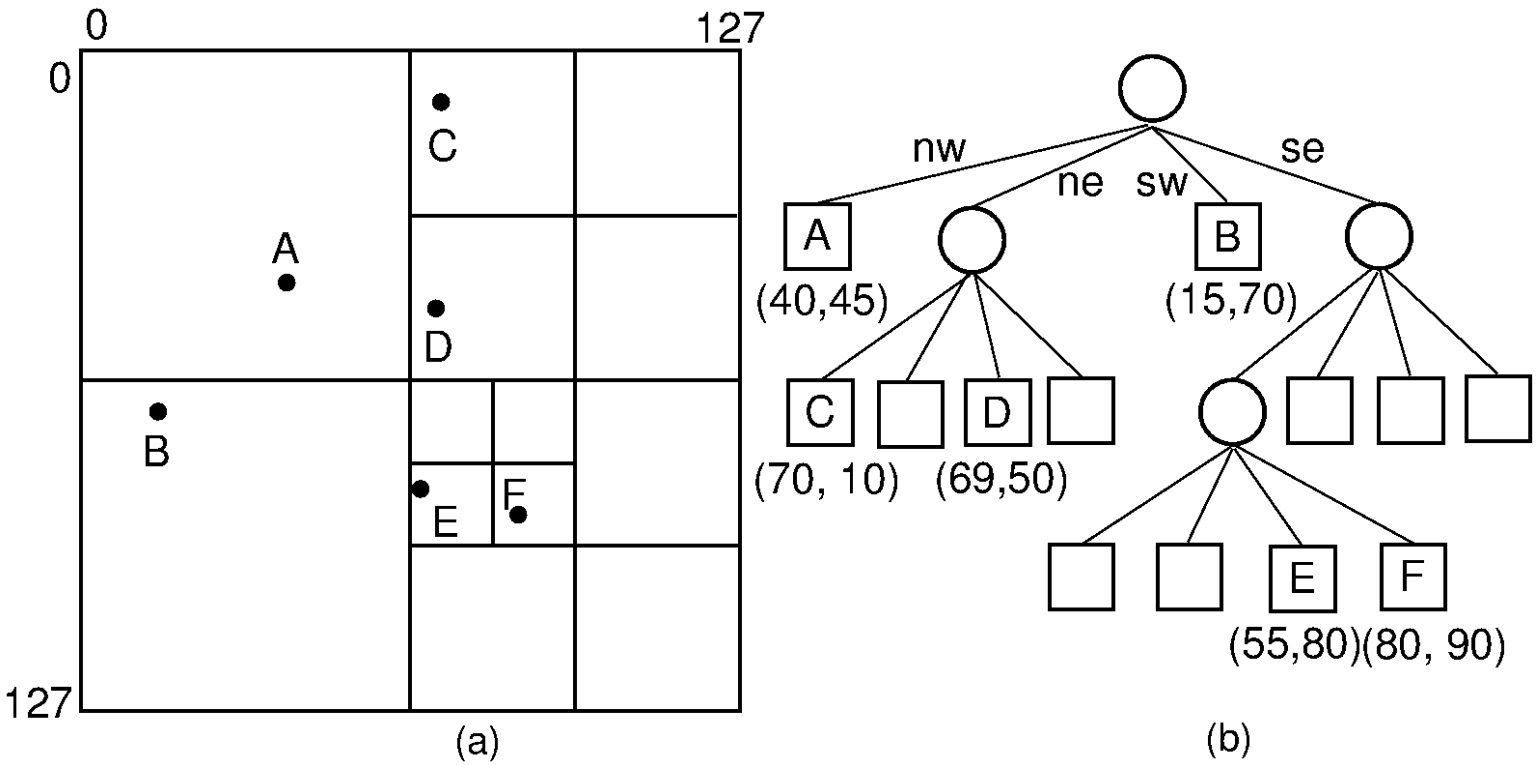
k-d tree

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

Quad tree

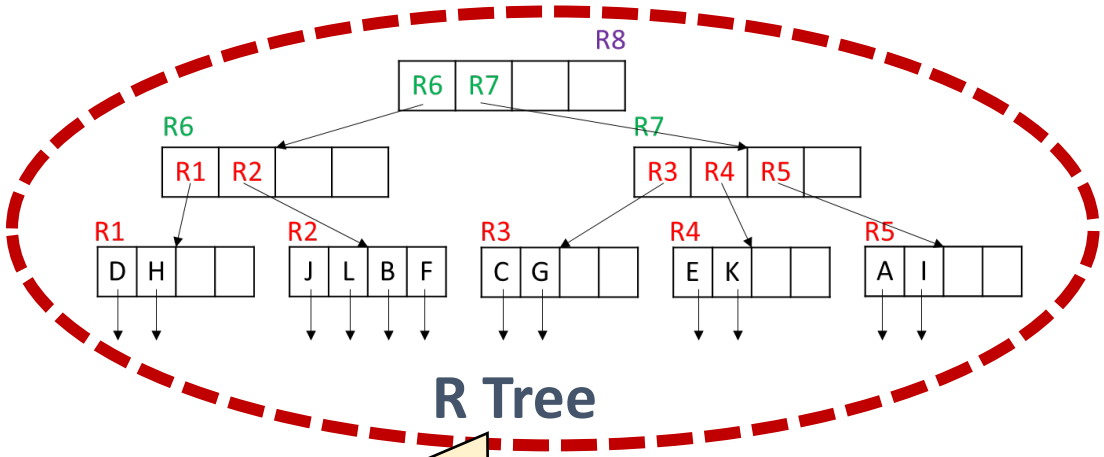
- Recursively partition a space into 4 regions evenly

Quad Tree



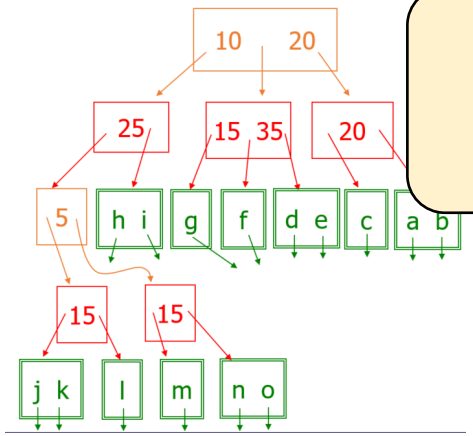
Source: https://opendsa-server.cs.vt.edu/ODSA/Books/CS3/html/_images/PRexamp.png

Indexing (Spatial Data): Summarization

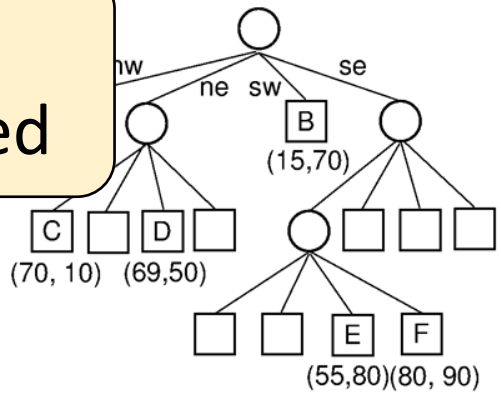


R Tree

Always balanced!
Most commonly used

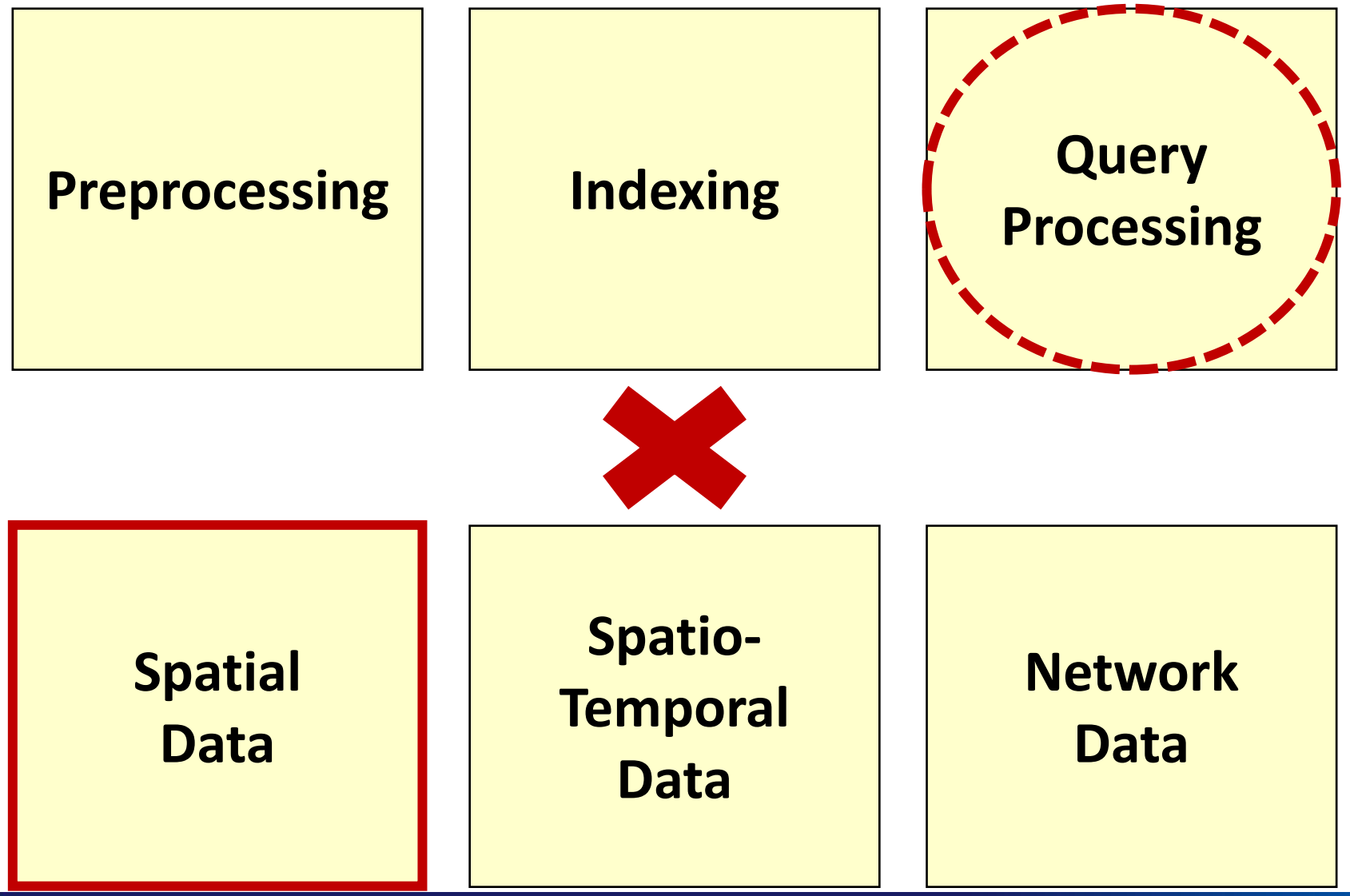


k-d Tree

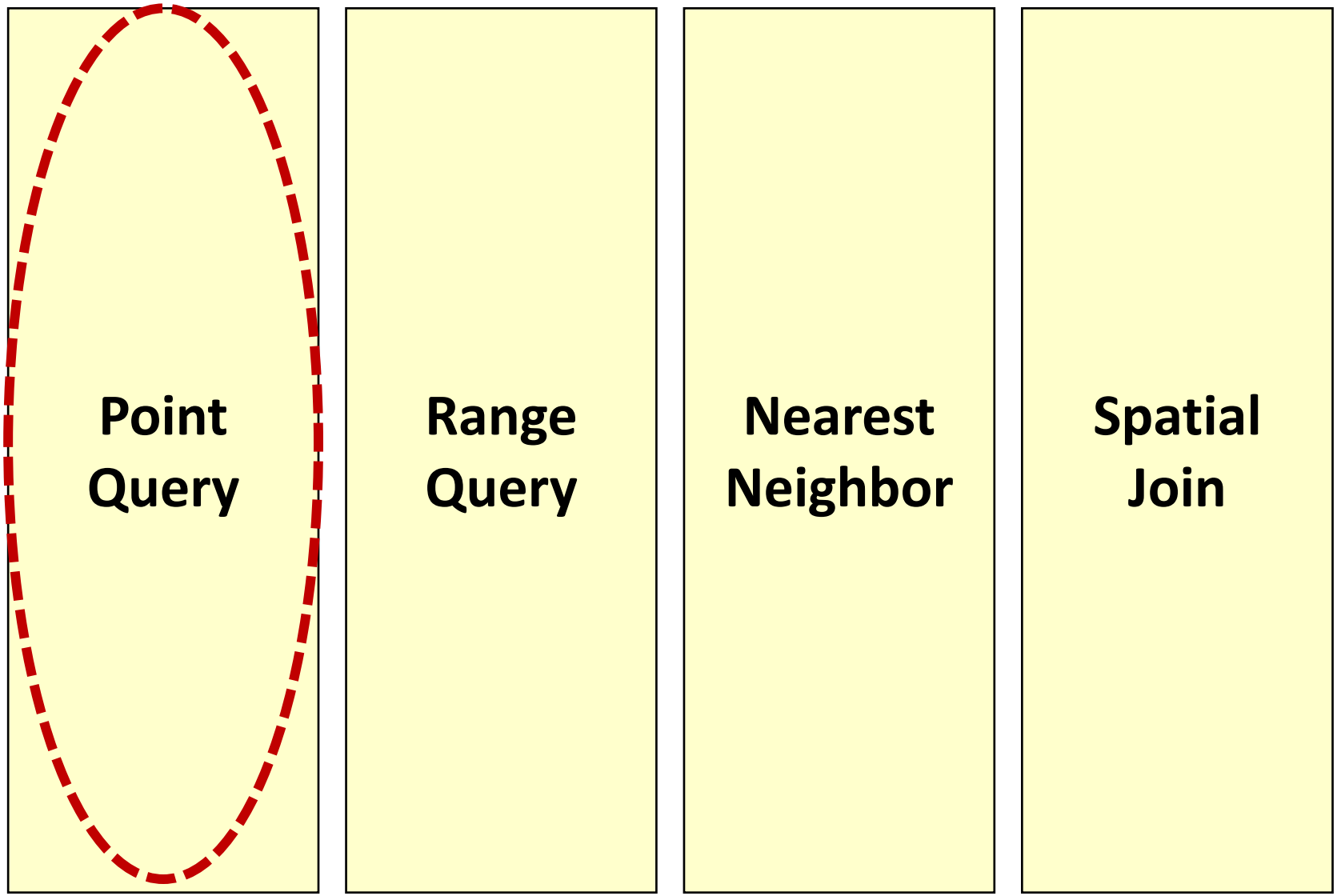


Quad Tree

Urban Data Management



Query Processing (Spatial Data)

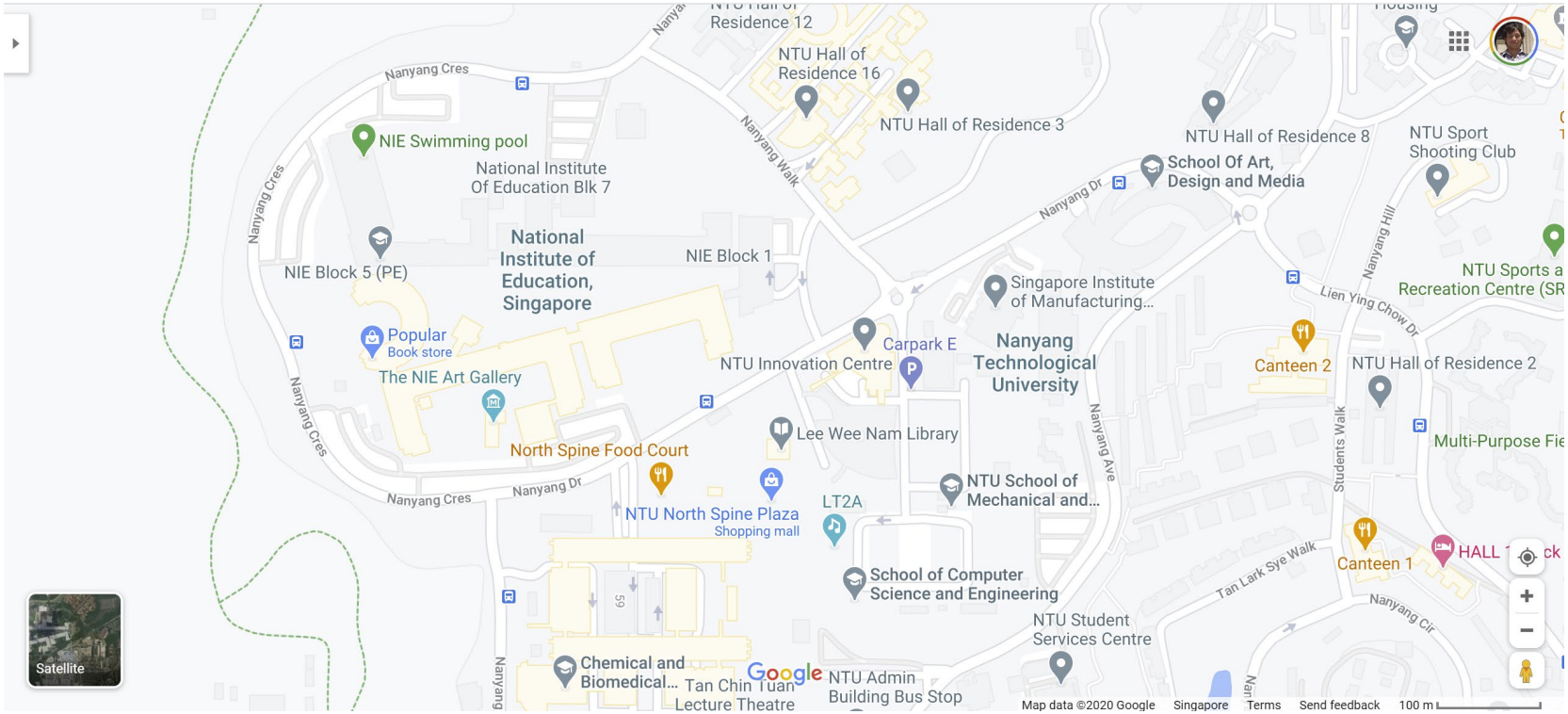


Point Query: Definition

Point Query

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

Point Query: Example

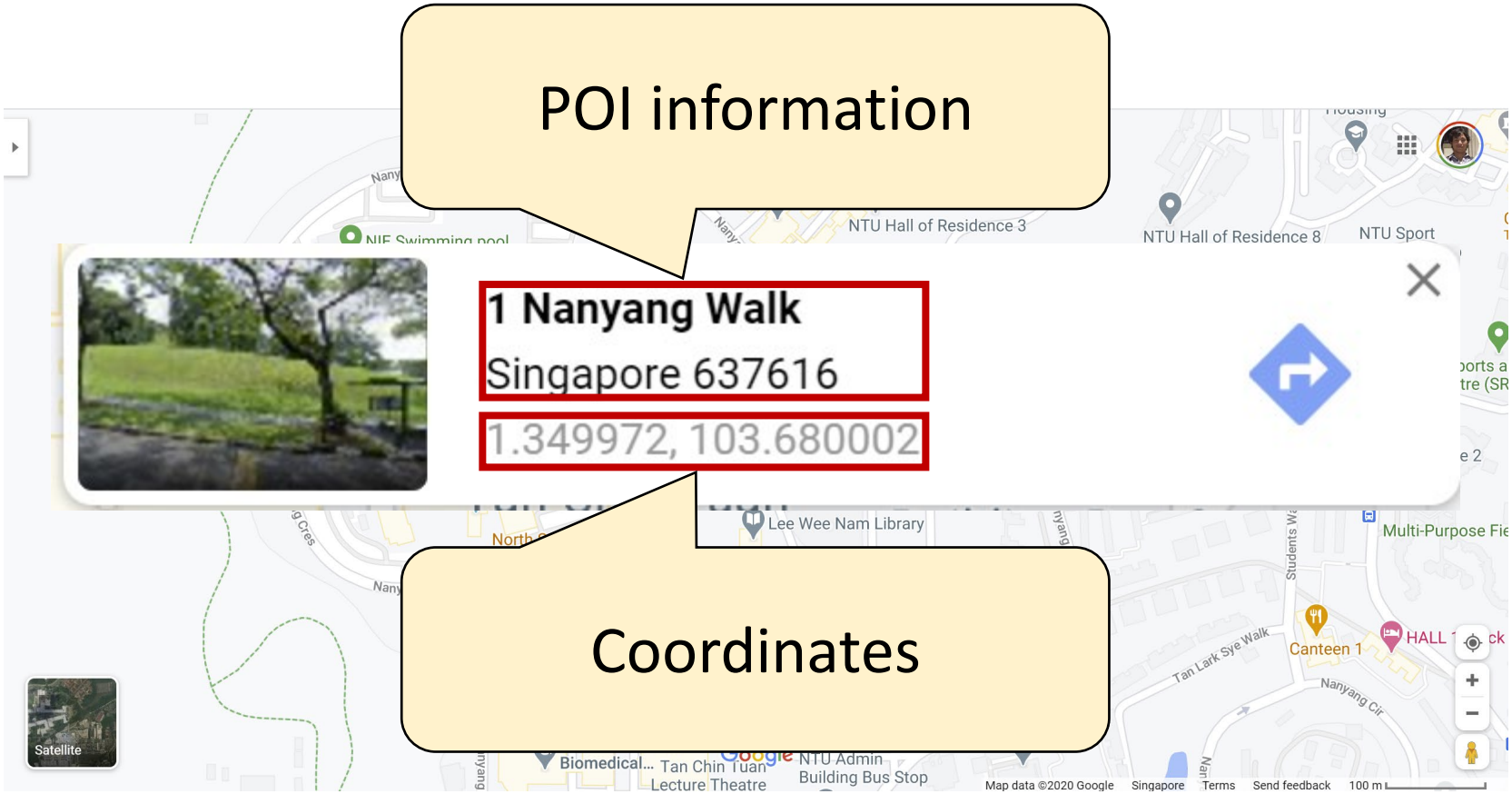


Google Maps

Point Query: Example

Google Maps

Point Query: Example

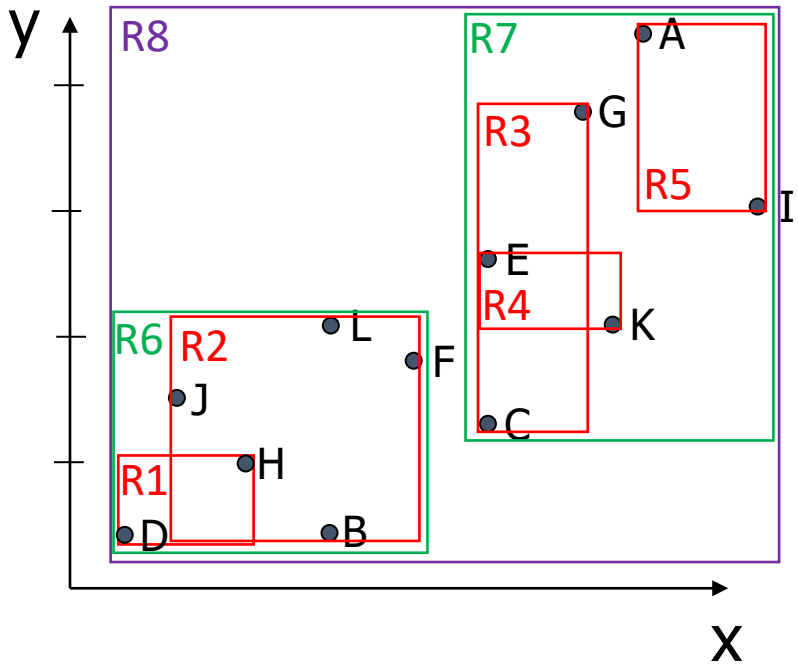


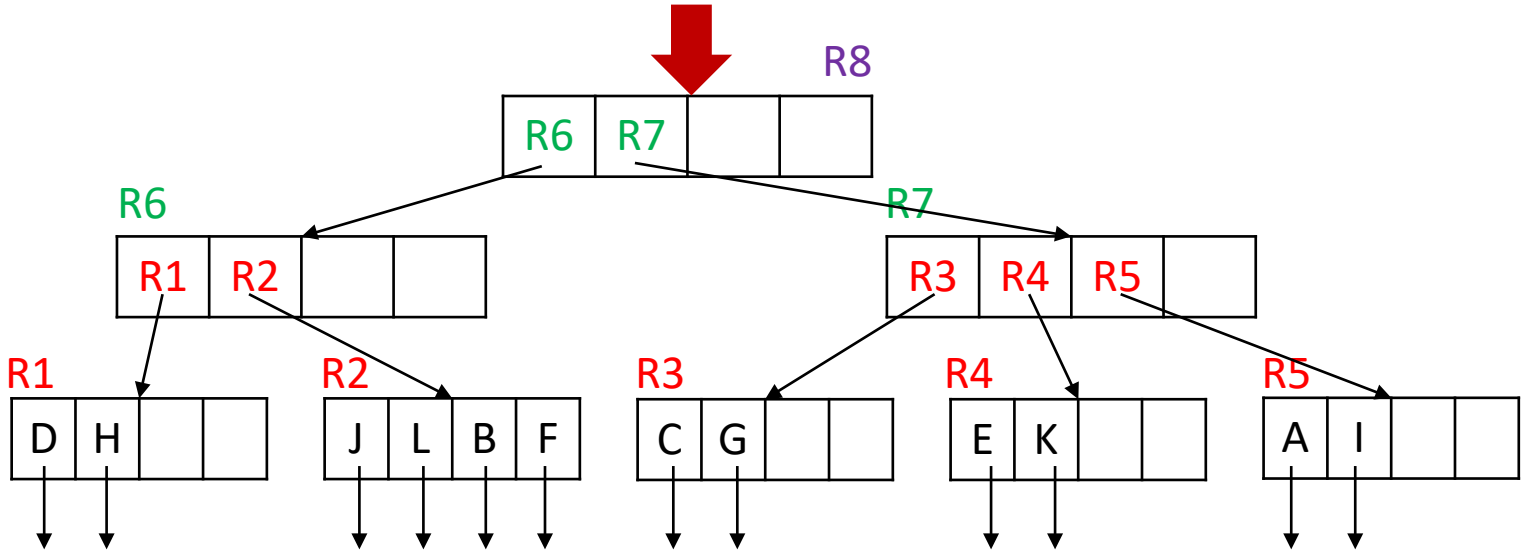
Google Maps

Point Query: Algorithm (With R Tree)

Search point K

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

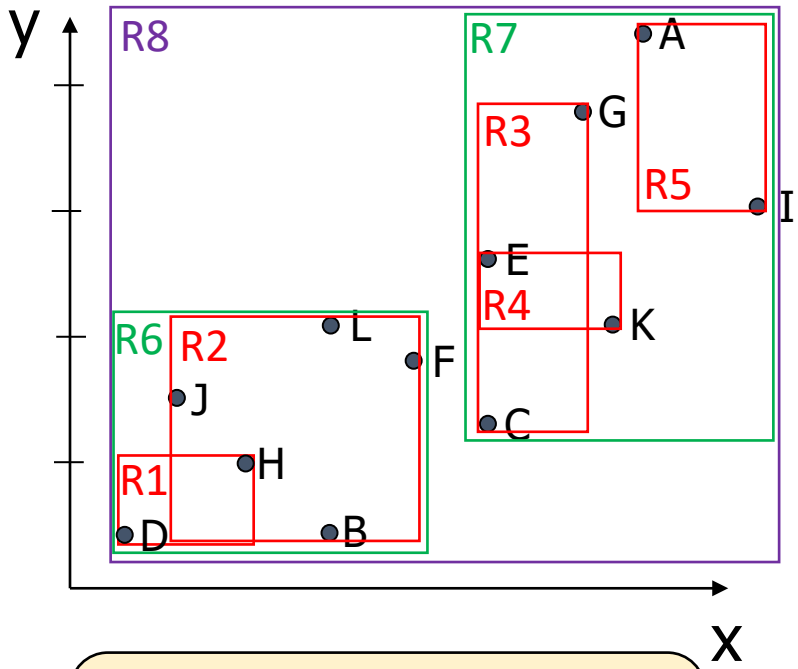




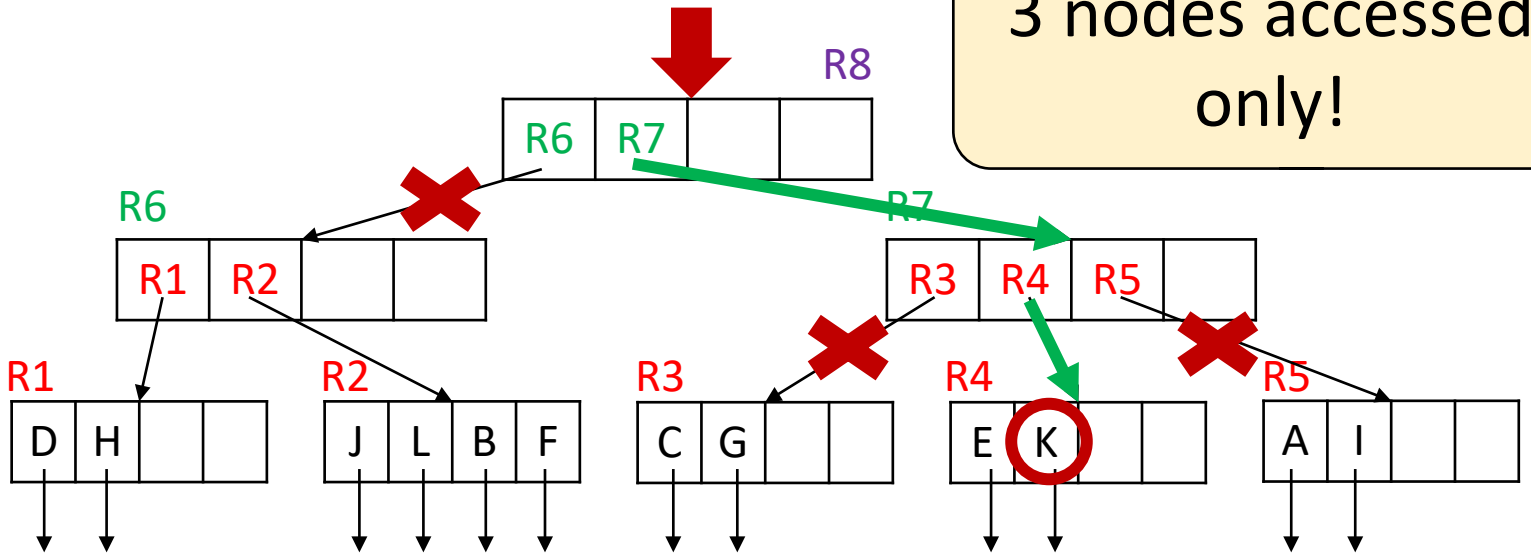
Point Query: Algorithm (With R Tree)

Search point K

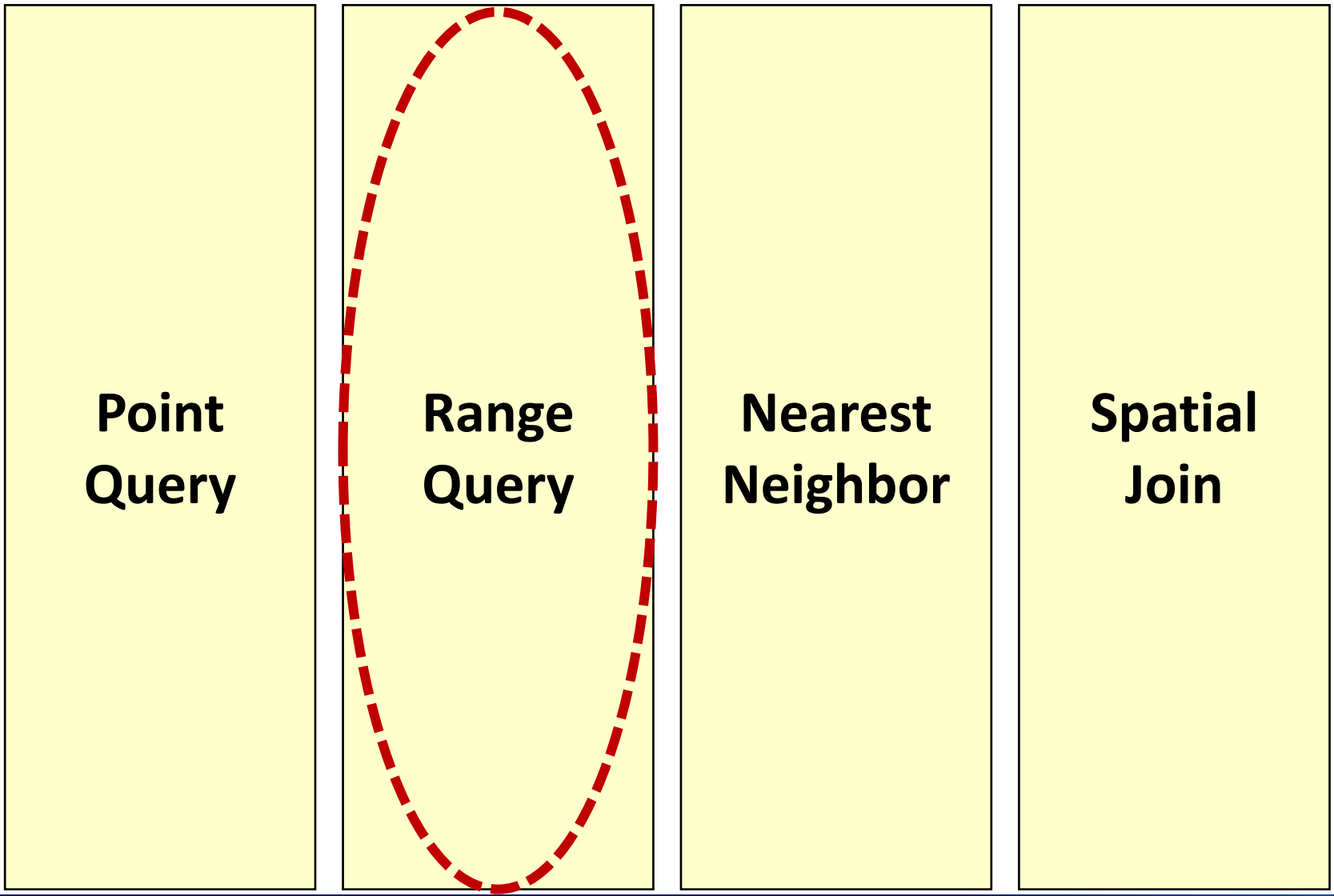
2. Enumerate all children with the MBR intersecting K



3 nodes accessed only!



Query Processing (Spatial Data)

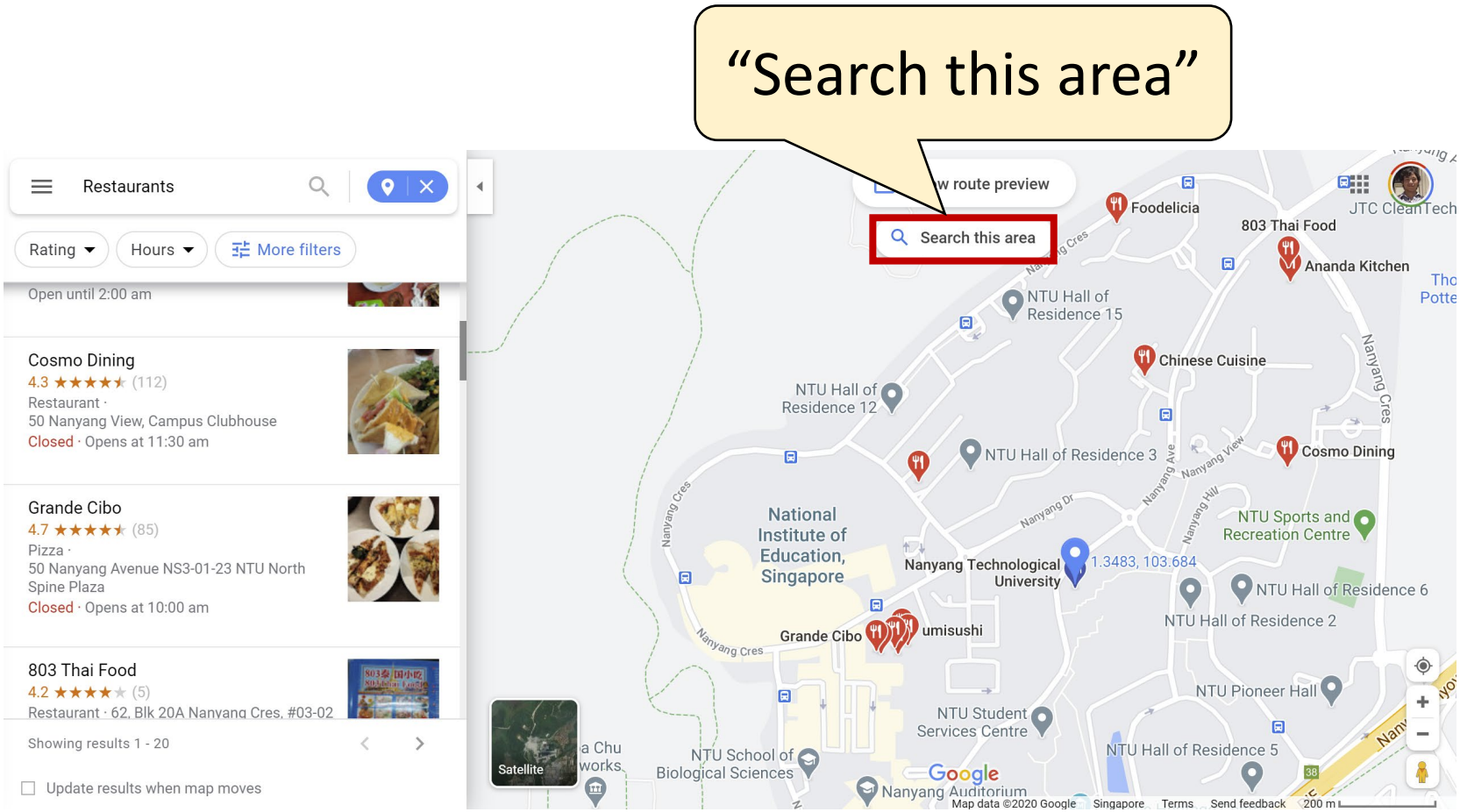


Range Query: Definition

Range Query:

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

Range Query: Example

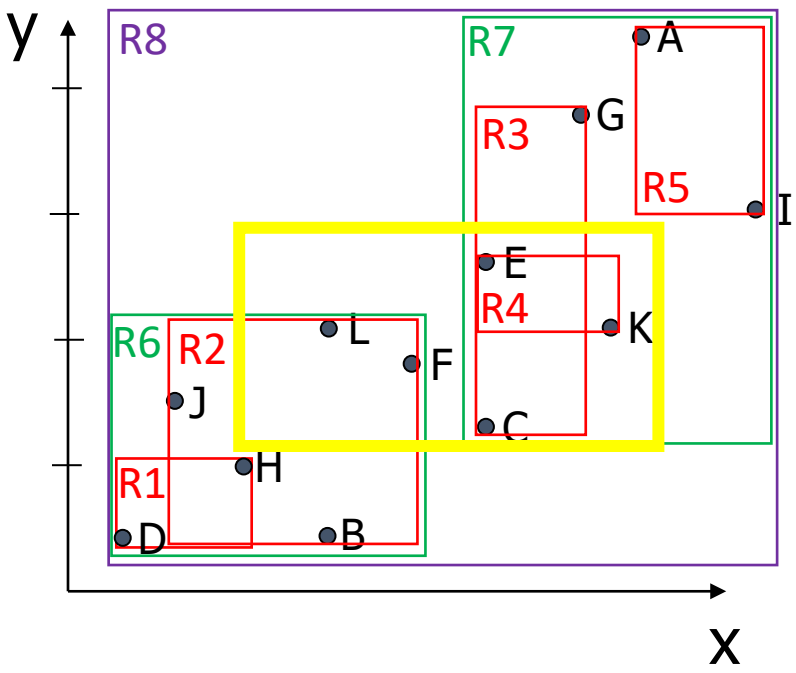


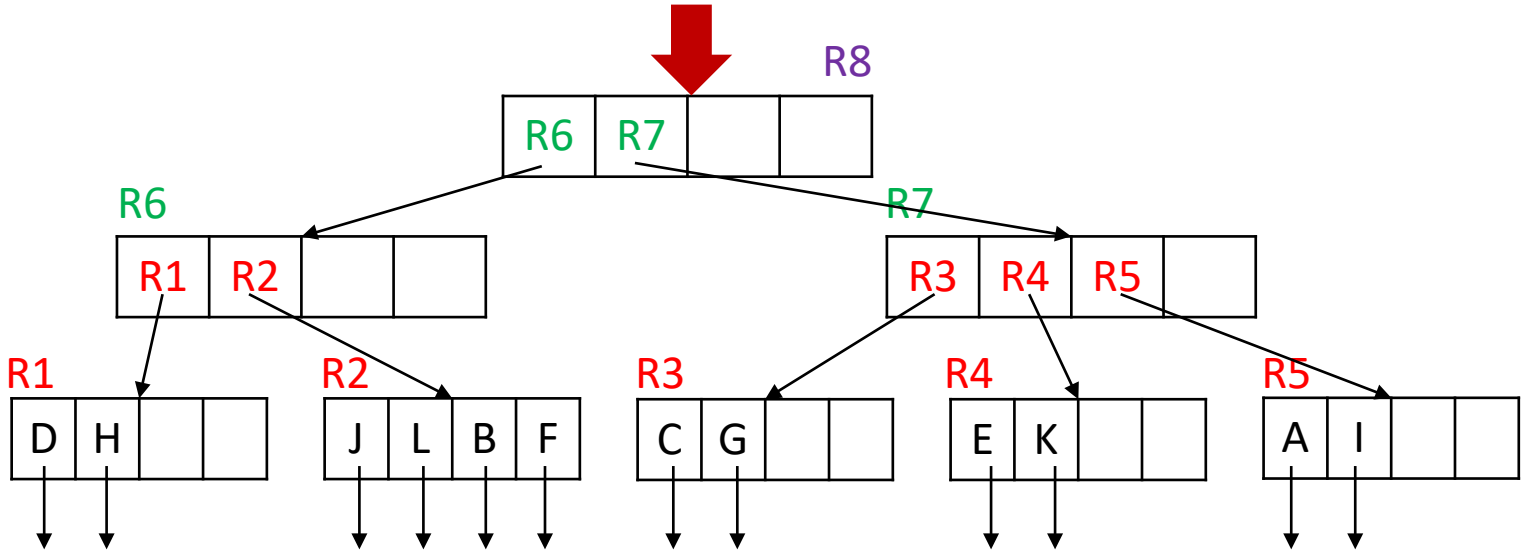
Google Maps

Range Query: Algorithm (With R Tree)

Search within the yellow box

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

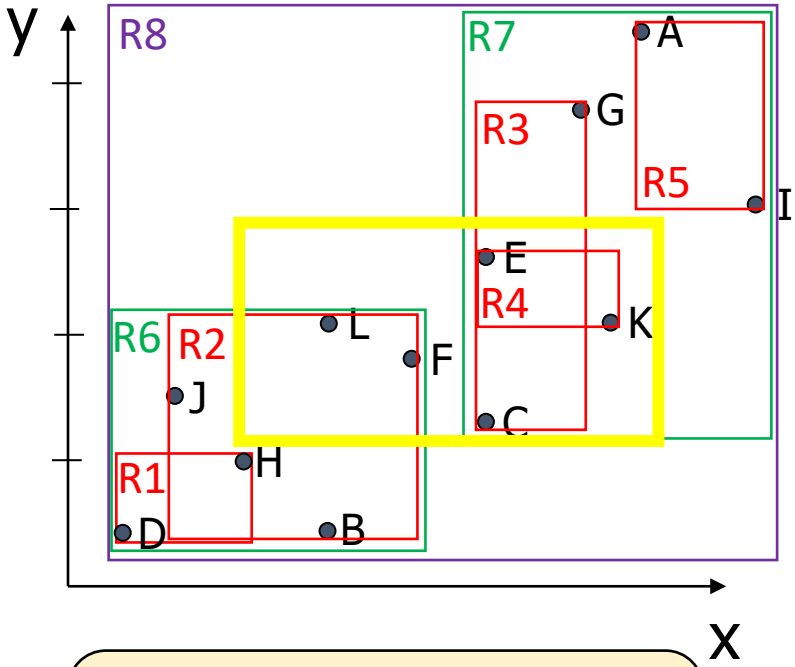




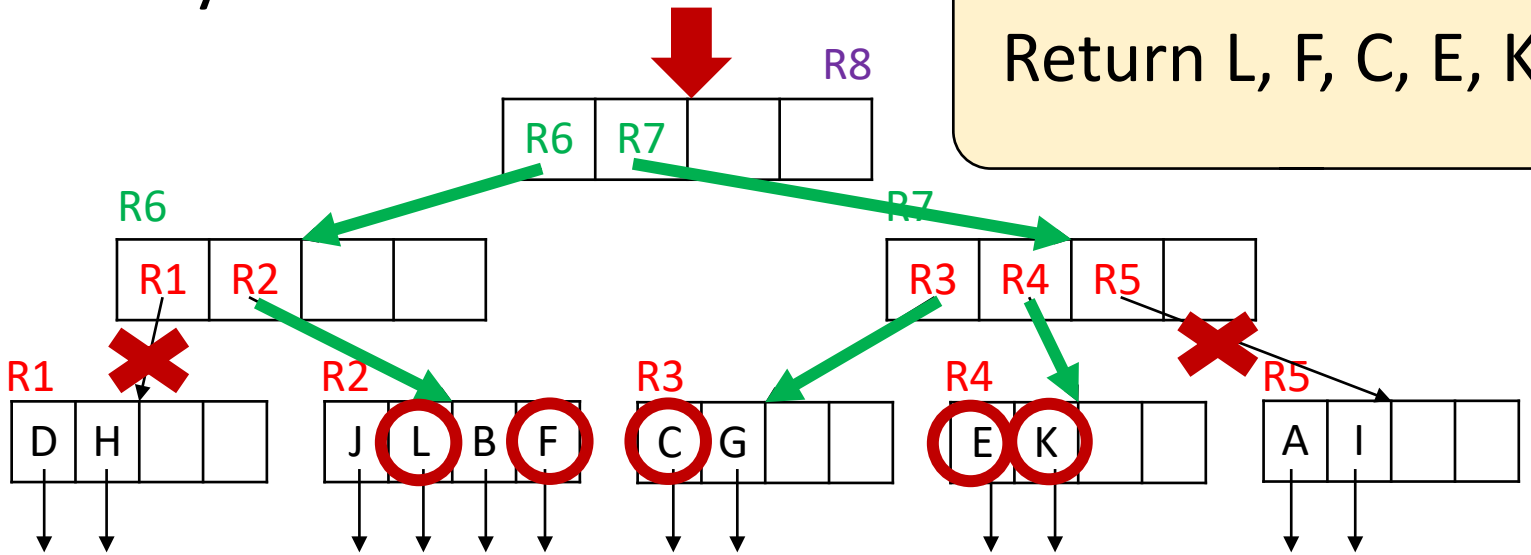
Range Query: Algorithm (With R Tree)

Search within the yellow box

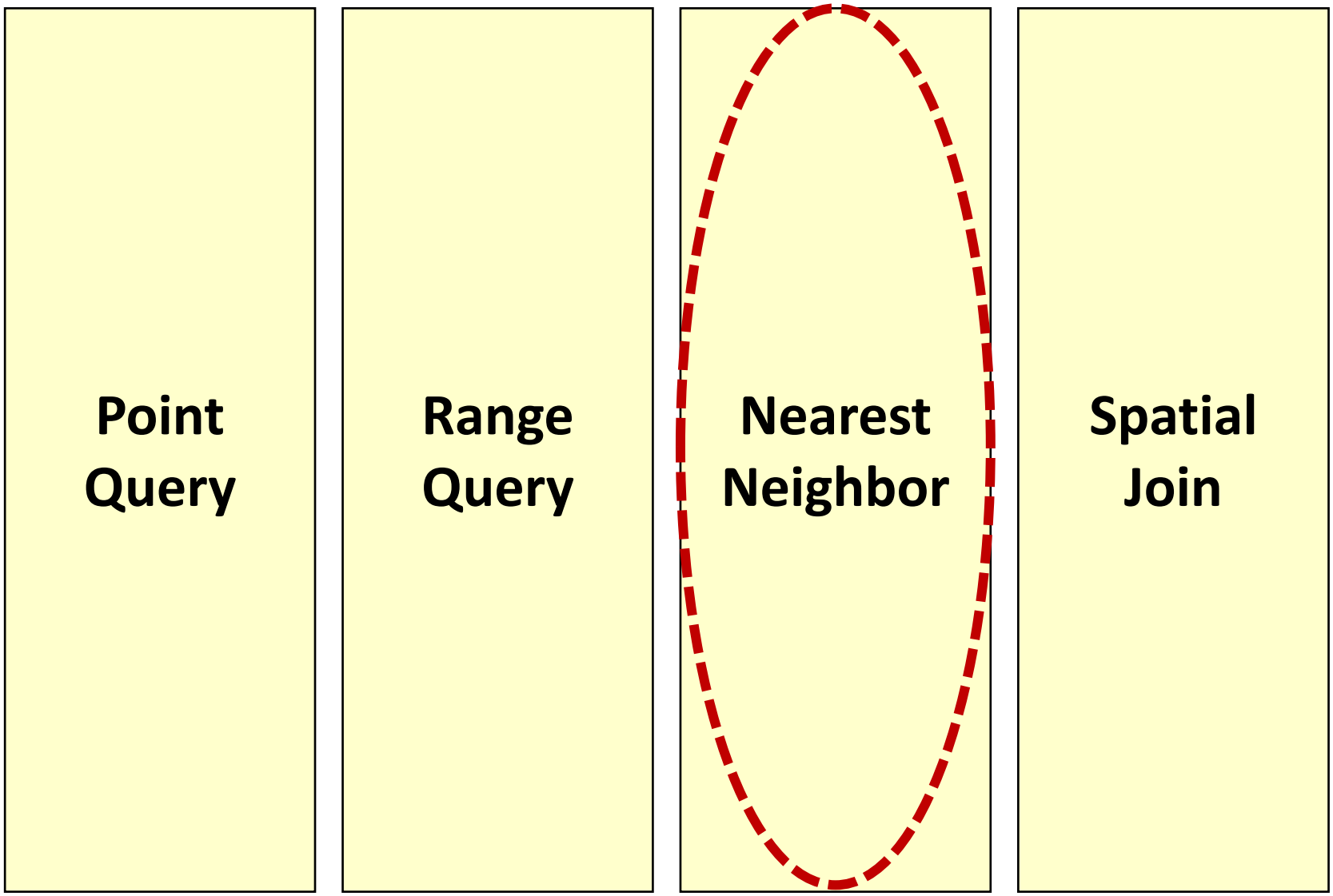
2. Enumerate all children with the MBR intersecting the yellow box



Return L, F, C, E, K



Query Processing (Spatial Data)

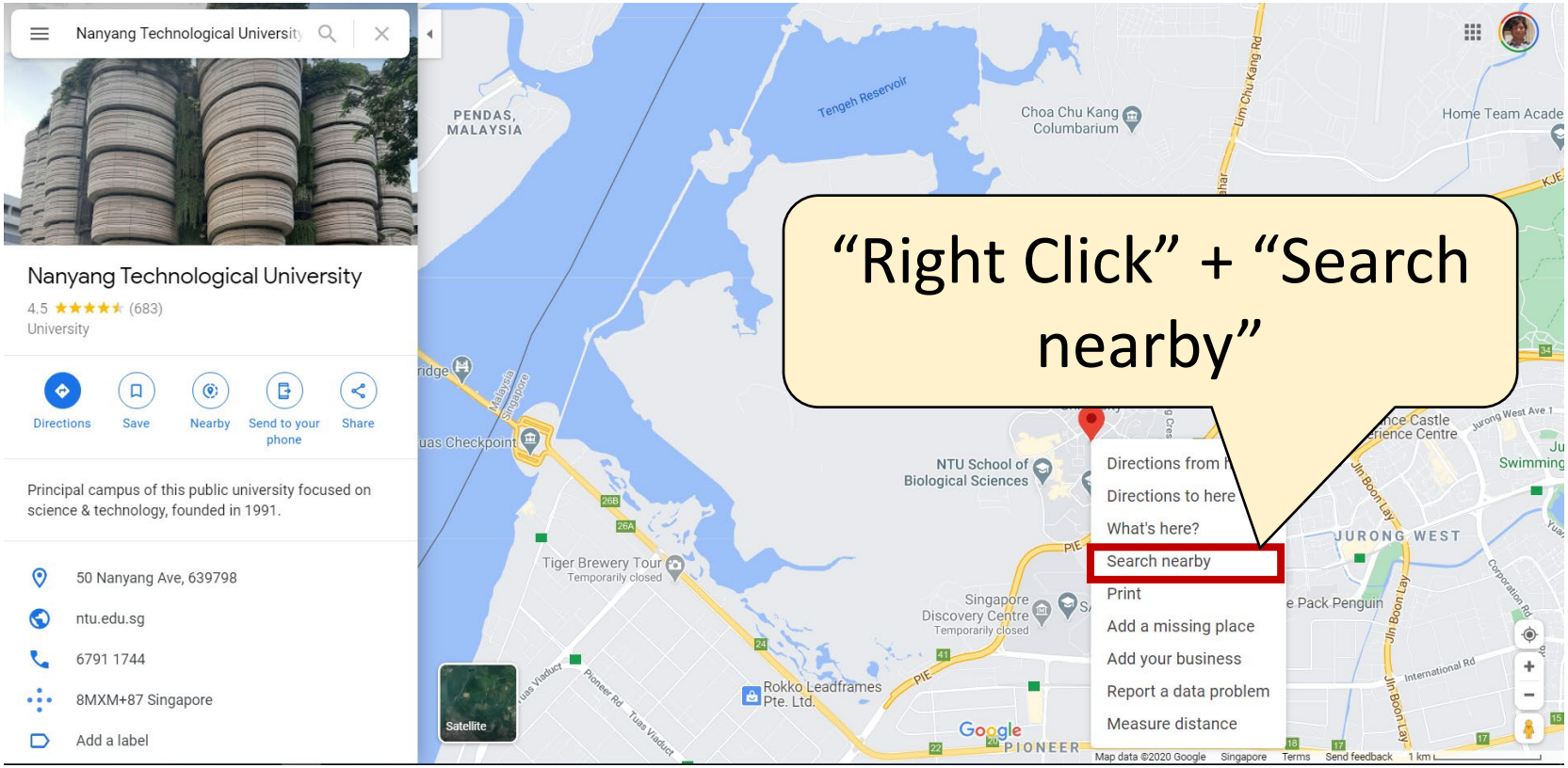


Nearest Neighbor: Definition

Nearest Neighbor:

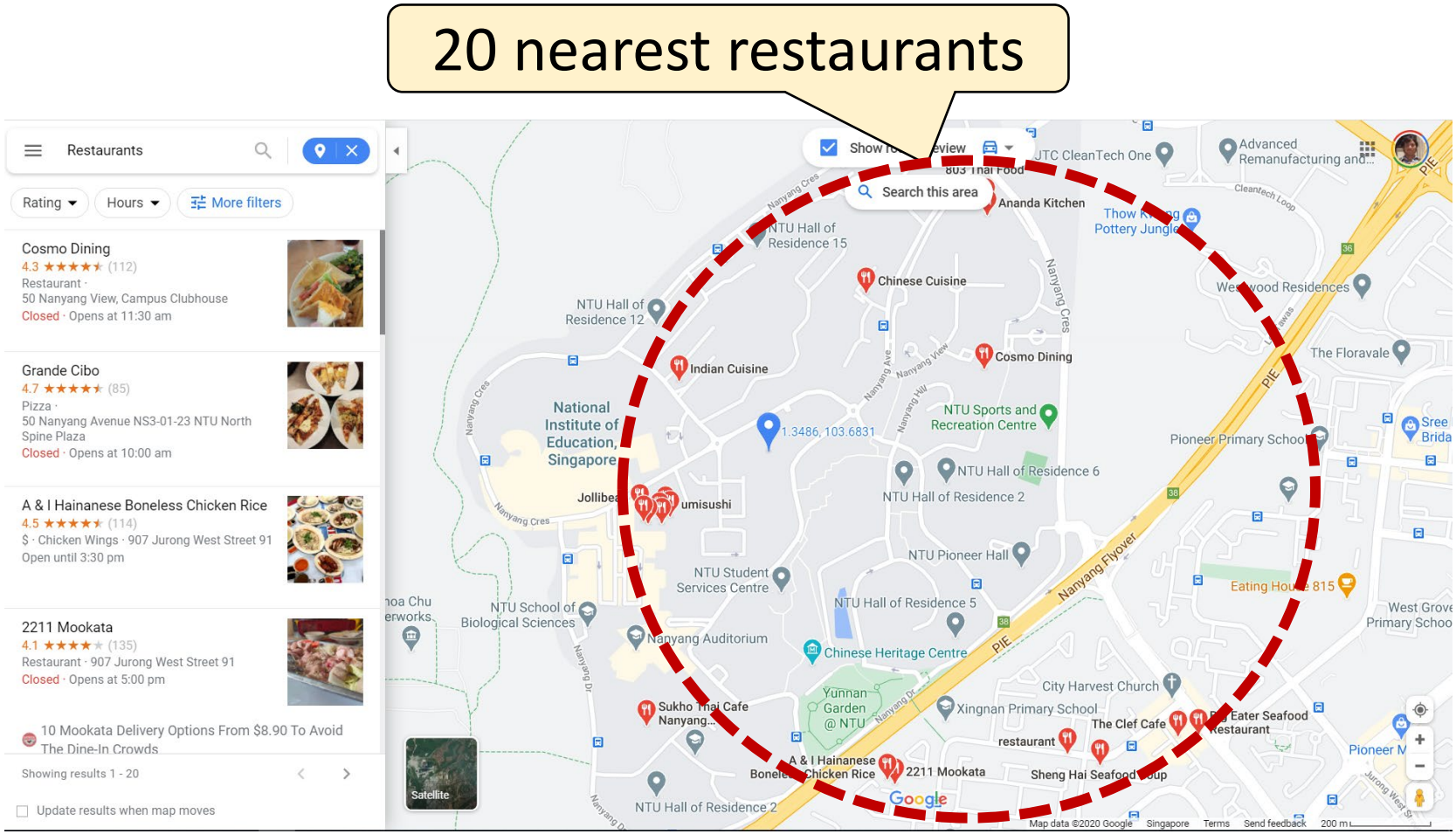
- Given a query location
- Return the object **nearest** to the query location

Nearest Neighbor: Example



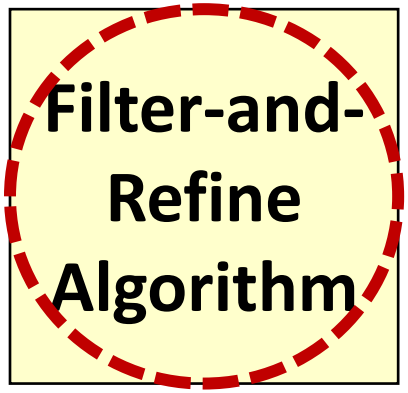
Google Maps

Nearest Neighbor: Example



Google Maps

Nearest Neighbor: Algorithm (With R Tree)



Filter and Refine Algorithm

Filter and Refine:

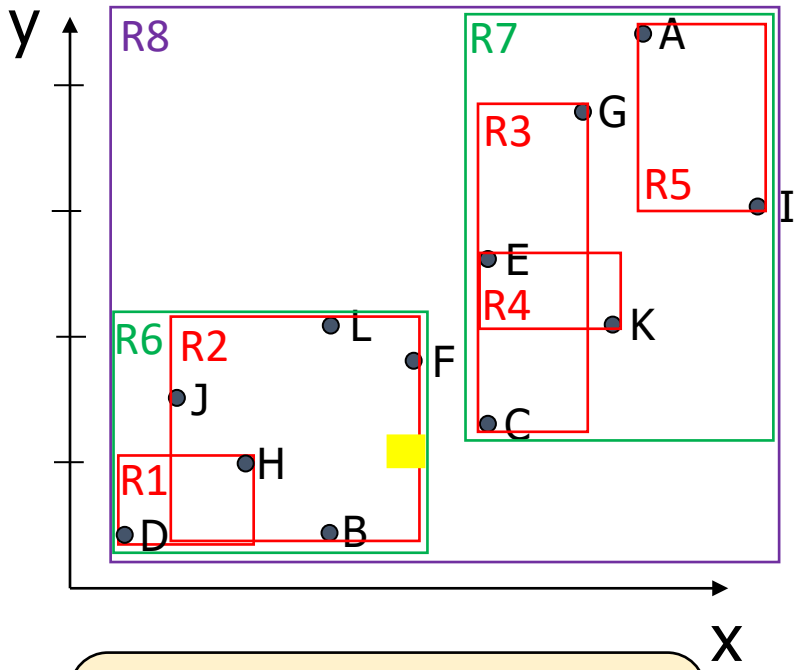
1. Fetch the node that contain the query location
2. R = minimum distance between the query location and an object in the fetched node
3. Find all objects within distance R from the query location (via a Range Query)
4. Return among the found objects the one that is the closest to the query location

Filter and Refine Algorithm

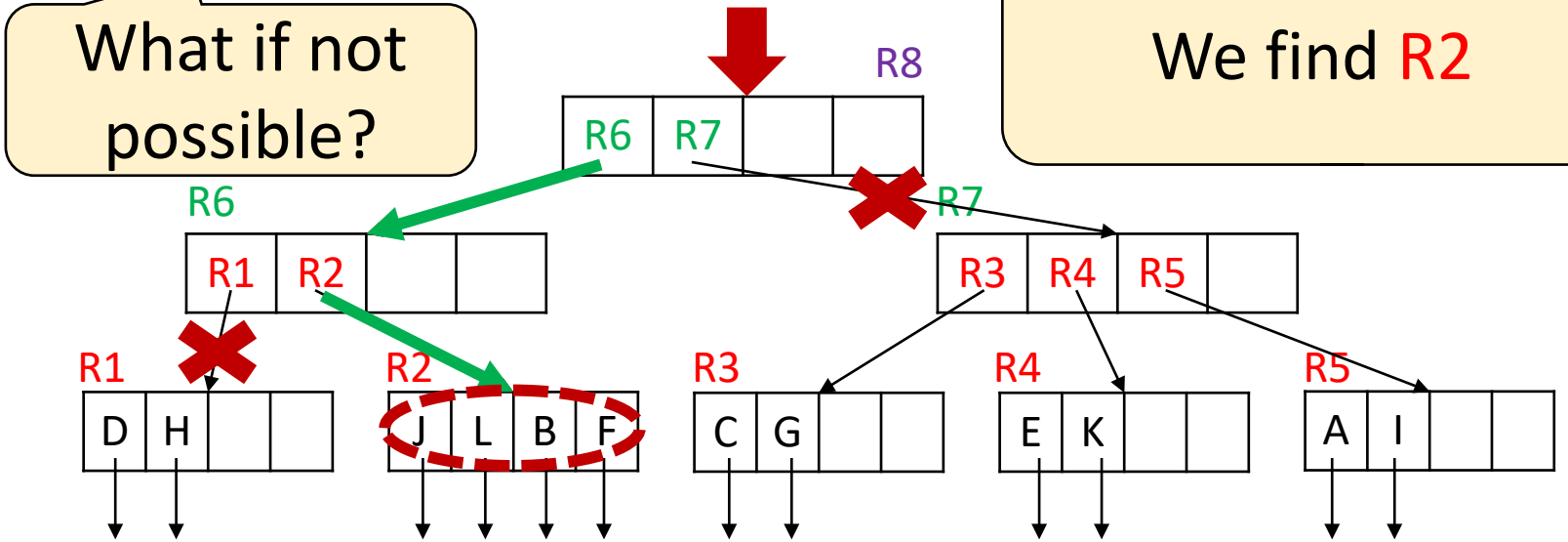
Find the object nearest to yellow dot

1. Fetch the leaf node containing the yellow dot

What if not possible?



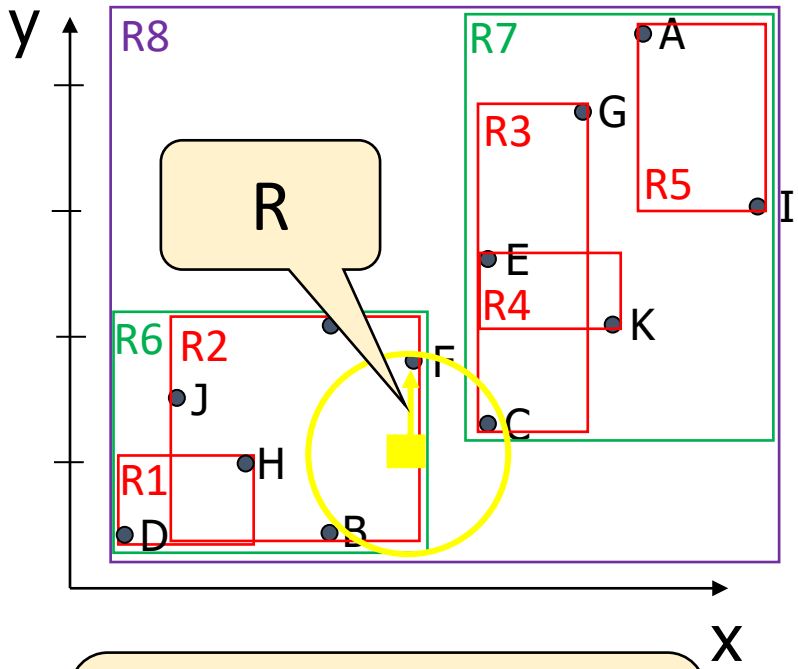
We find R2



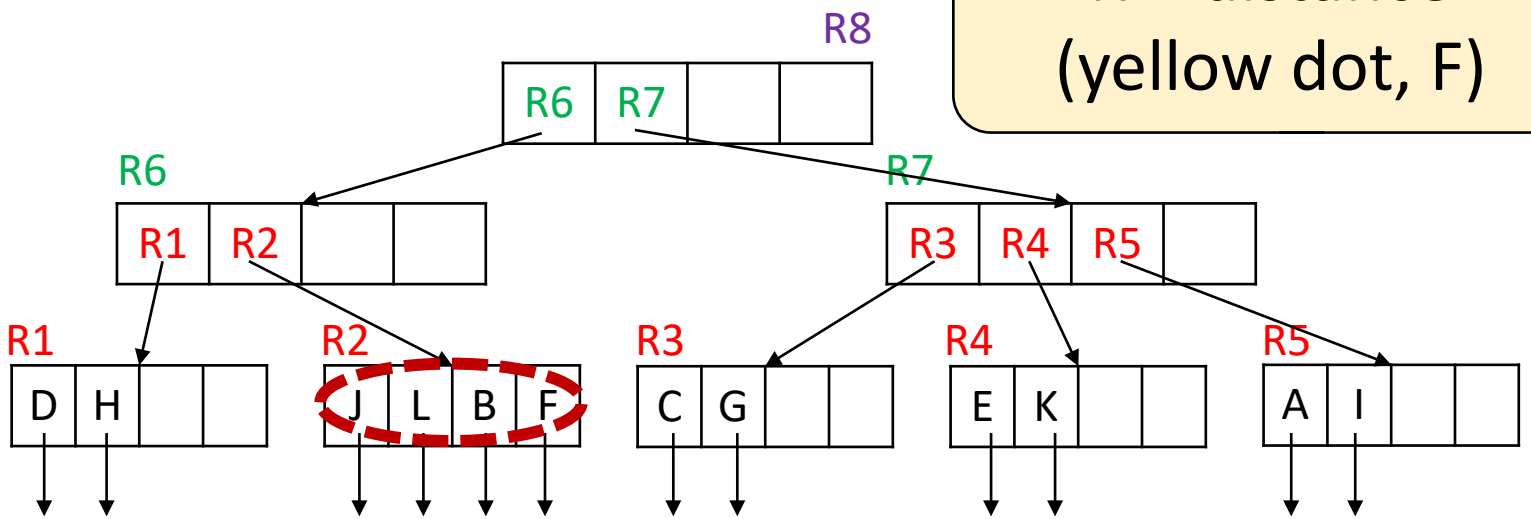
Filter and Refine Algorithm

Search within the yellow box

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



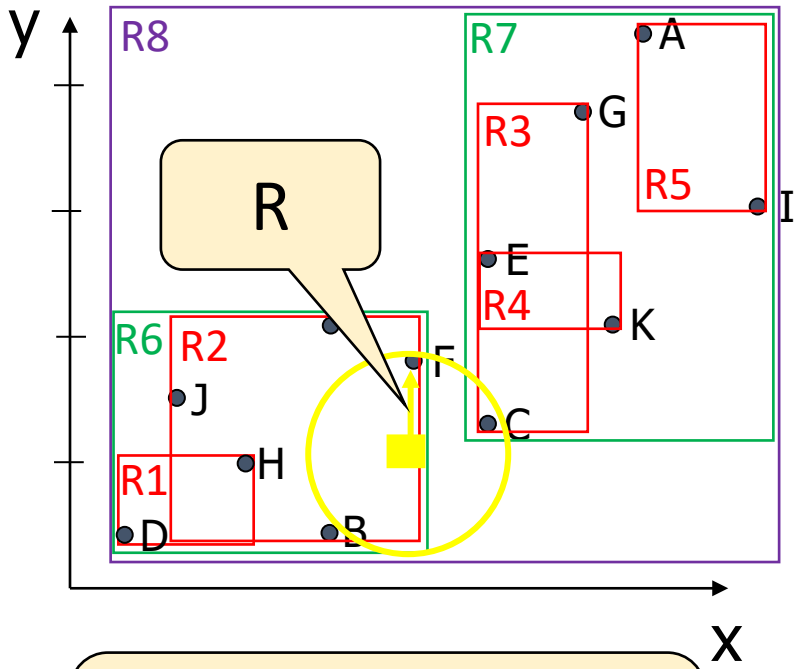
R = distance (yellow dot, F)



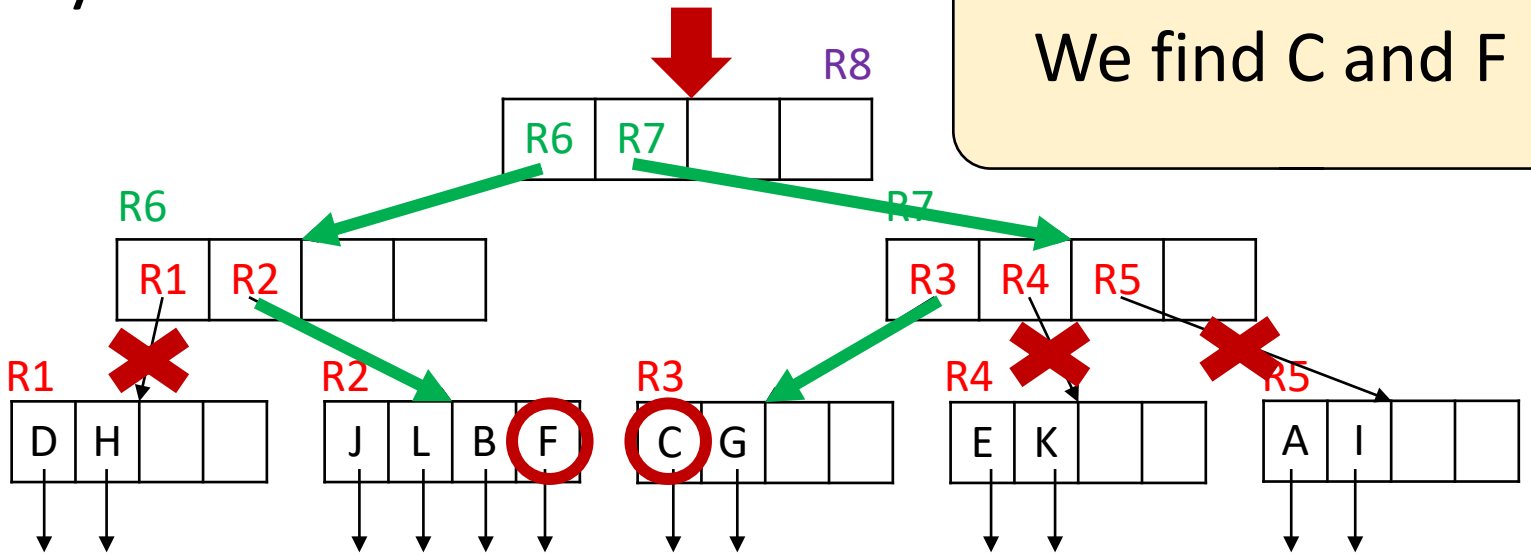
Filter and Refine Algorithm

Search within the yellow box

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



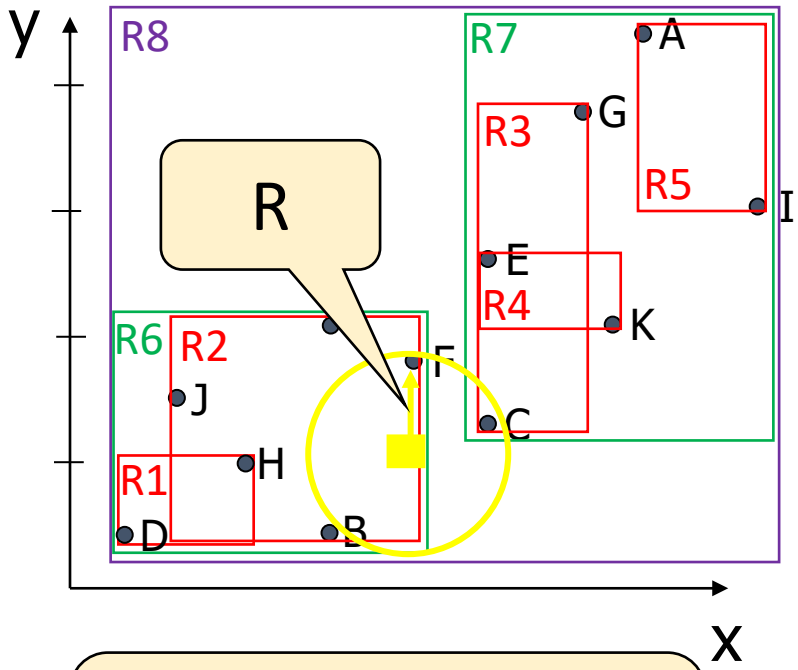
We find C and F



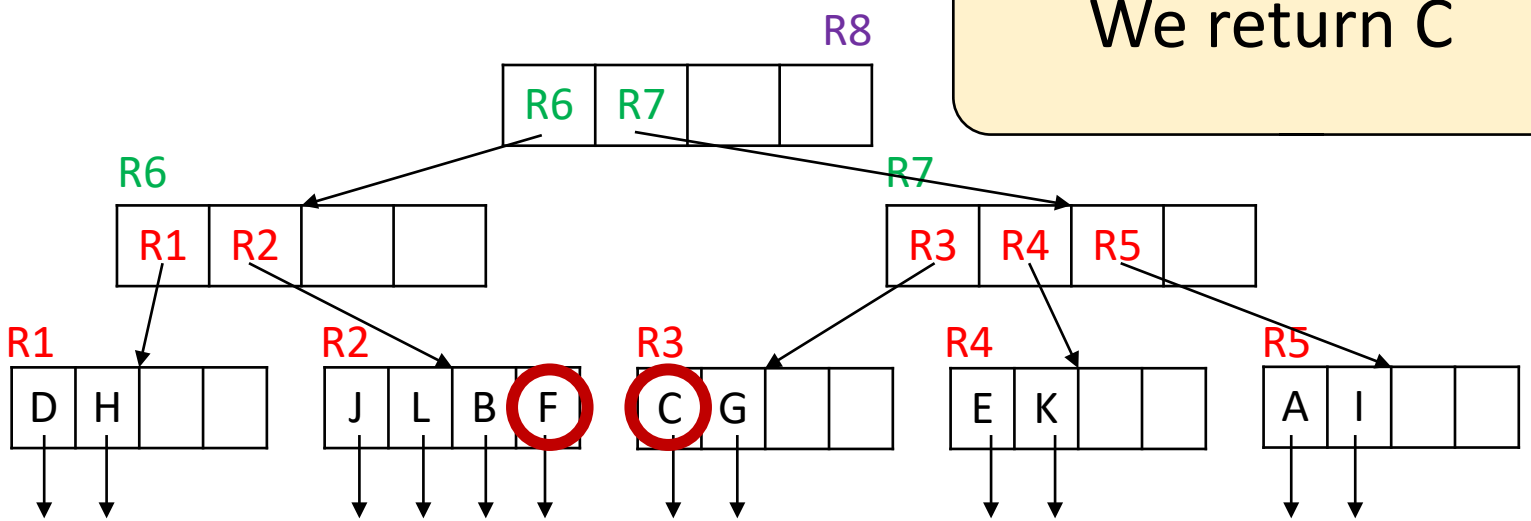
Filter and Refine Algorithm

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

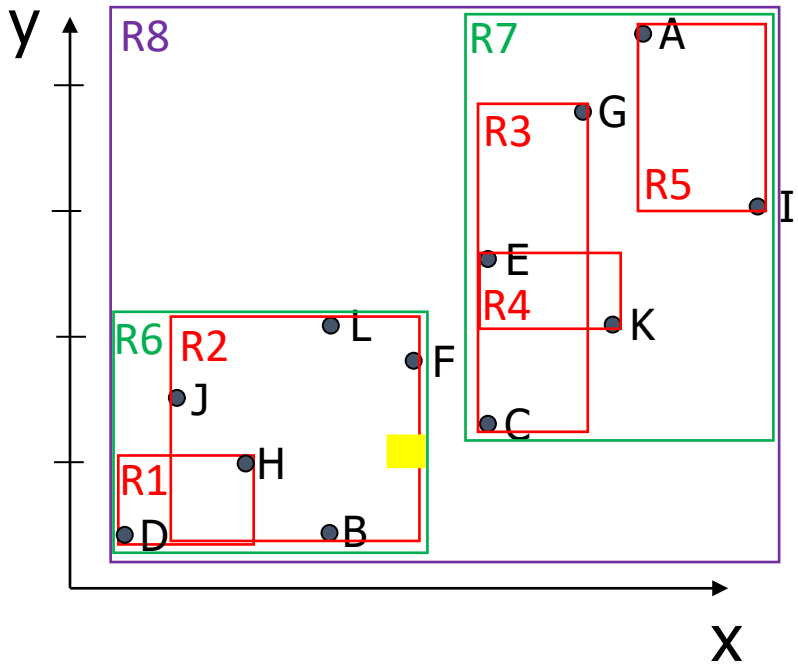
Best-First Search:

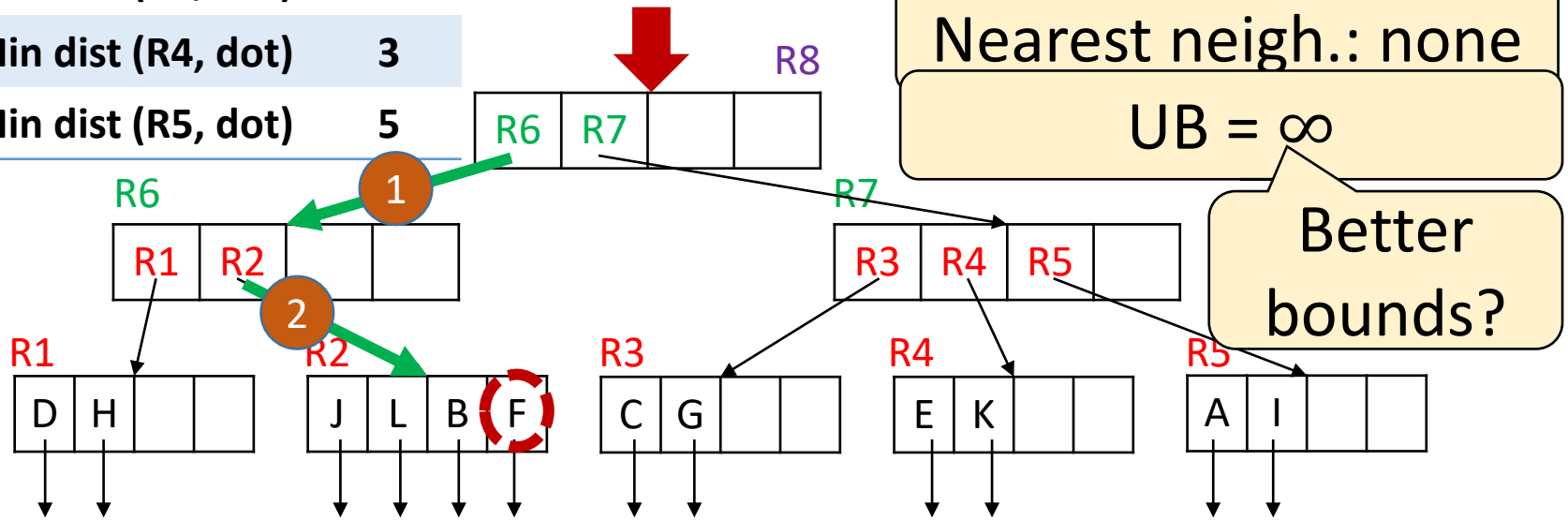
- Maintain an **upper bound** of the distance between the query location and its nearest neighbor, denoted by **UB**
- Explore only those nodes with their MBRs' **minimum** distance to the query location **at most** UB, and in an ascending order of the minimum distances (i.e., best-first search)

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





Nearest neigh.: none

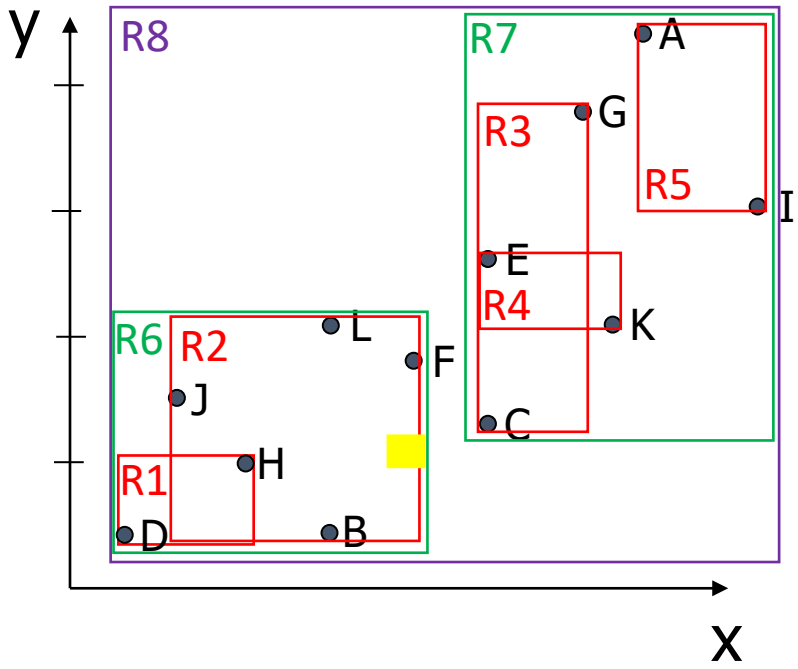
UB = ∞

Better bounds?

Best-First Search Algorithm

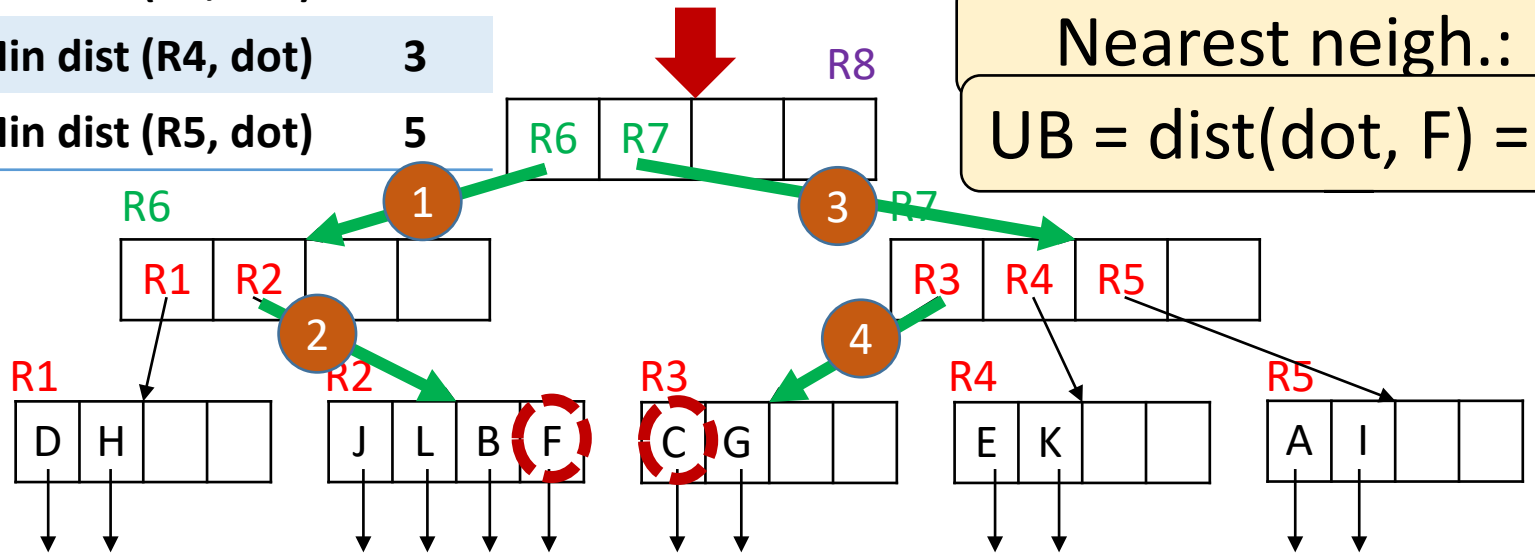
Find the object nearest to yellow dot

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

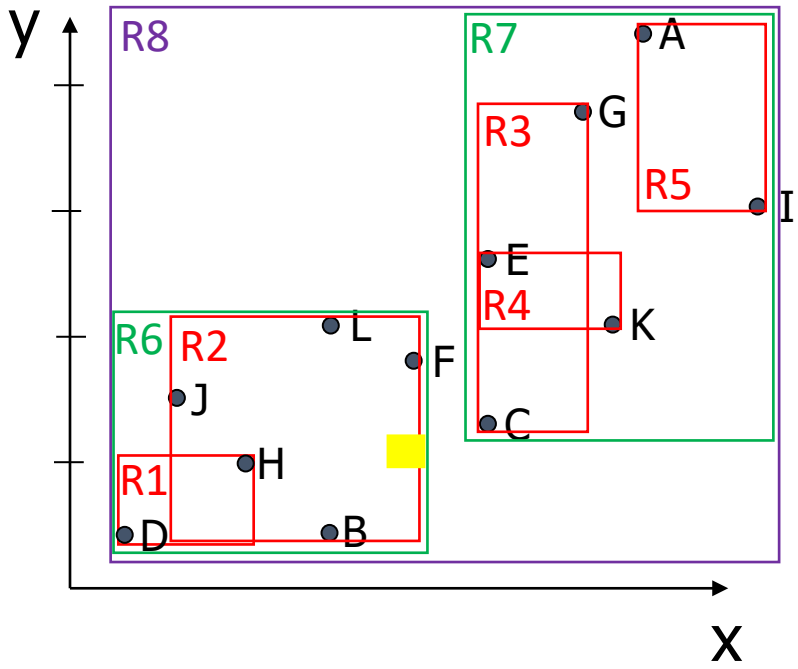
UB = dist(dot, F) = 1.5



Best-First Search Algorithm

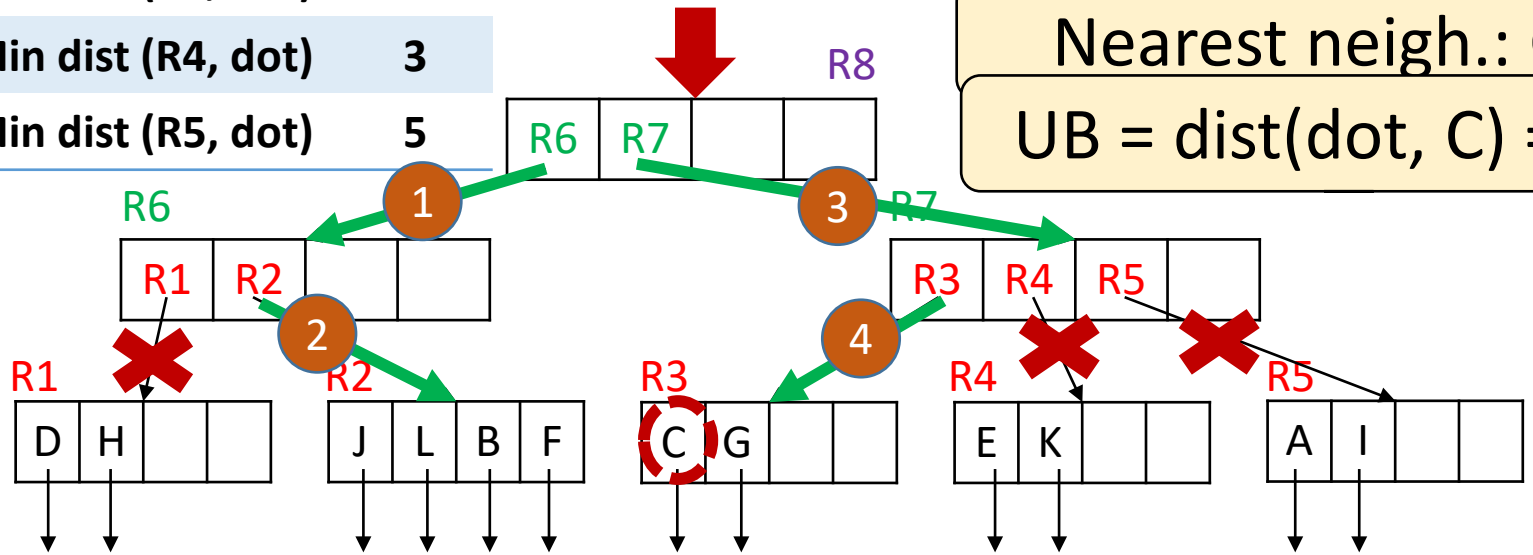
Find the object nearest to yellow dot

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

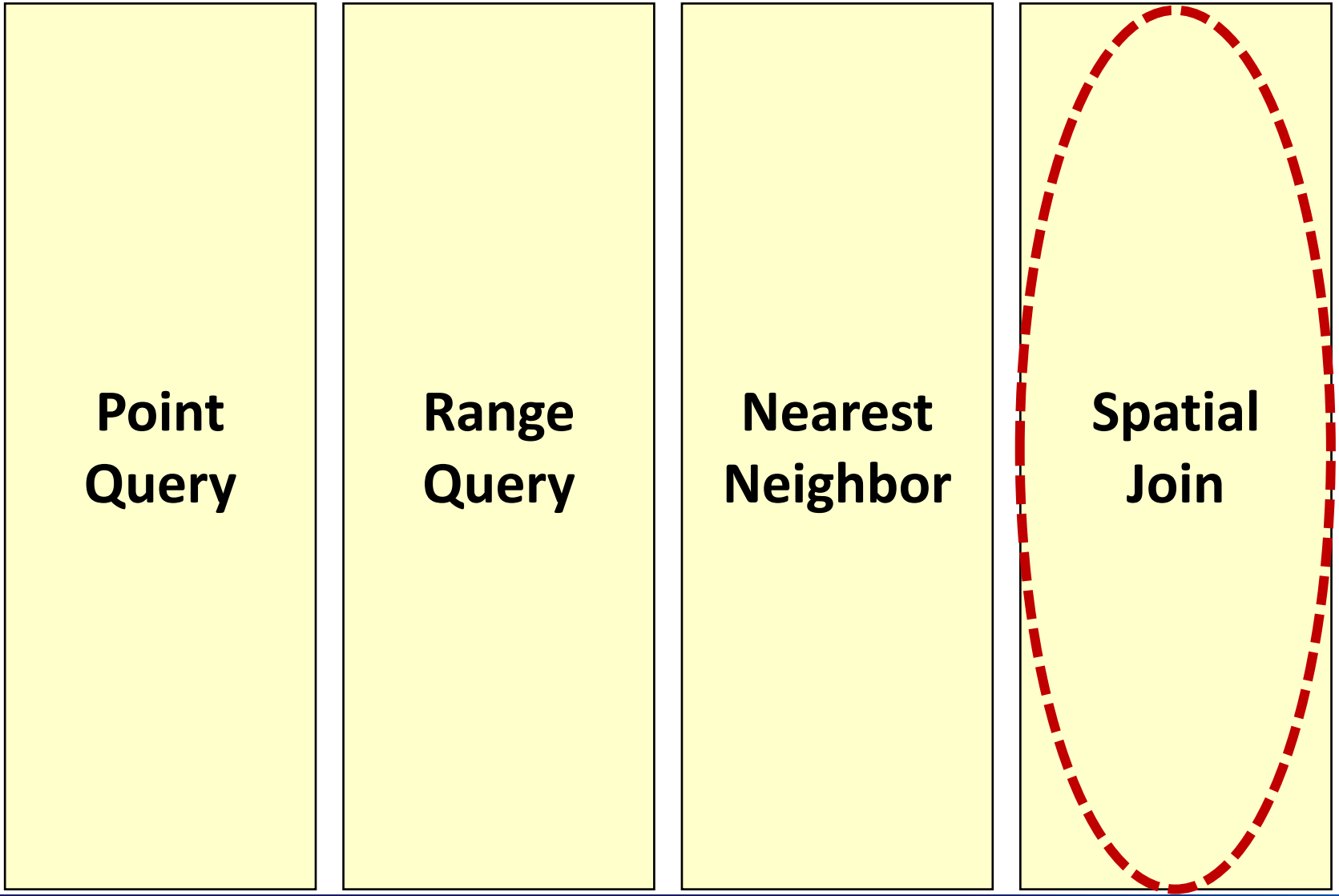


Nearest neigh.: C

UB = dist(dot, C) = 1



Query Processing (Spatial Data)



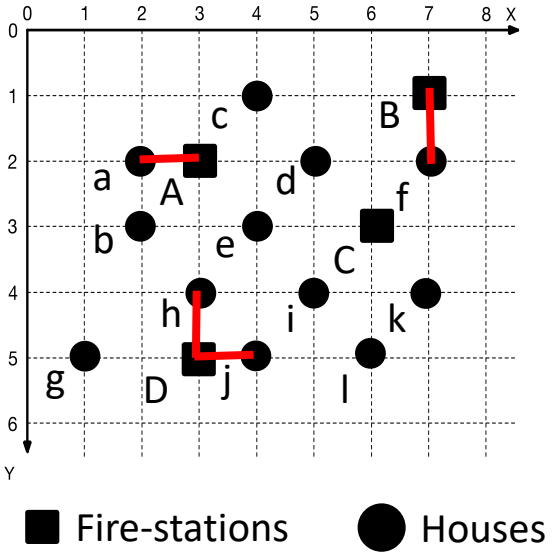
Spatial Join: Definition

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

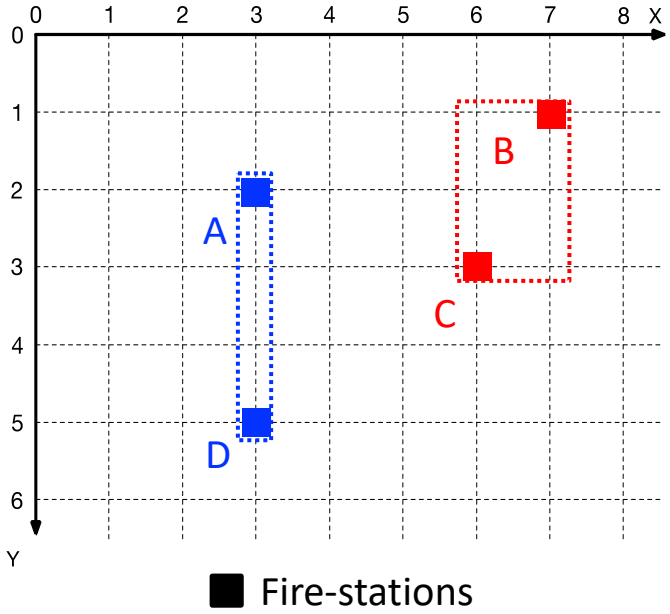


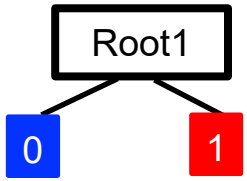
Spatial Join: Algorithm (With R Tree)

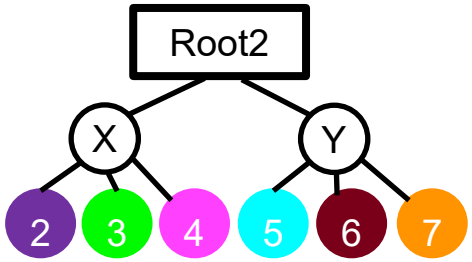
Spatial Join Algorithm:

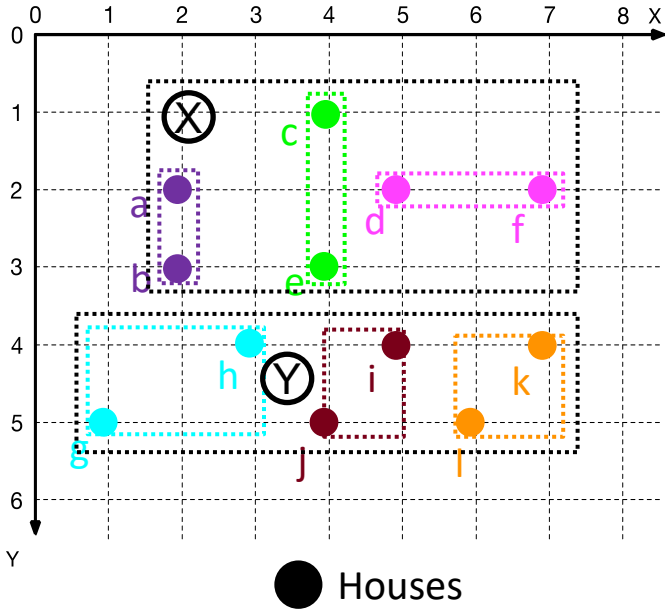
- Check if the rectangles of the roots satisfy the join predicate, and if so, start from the pair of (root1, root2)
- For each pair, **recursively** examine the pairs of their children if they intersect
- Return the pairs of objects that satisfy the join predicate

Spatial Join: Algorithm (With R Tree)



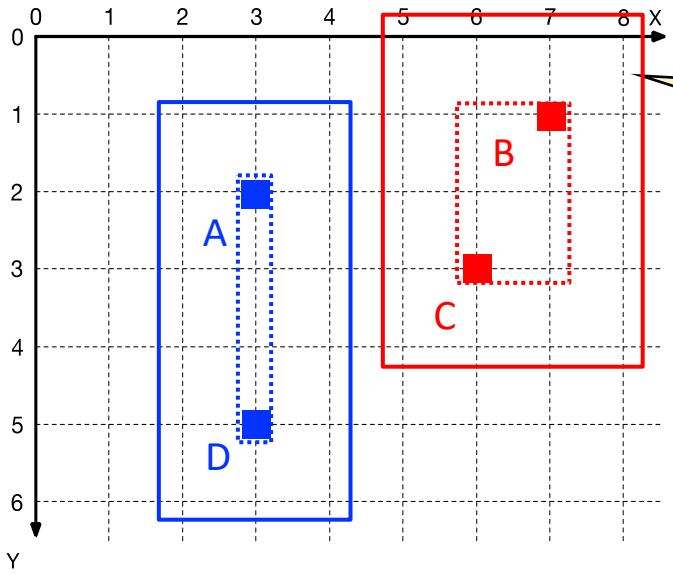




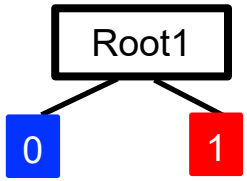


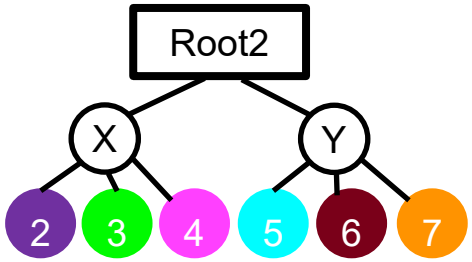
Find pairs of fire stations and houses, which are within distance of 1

Spatial Join: Algorithm (With R Tree)

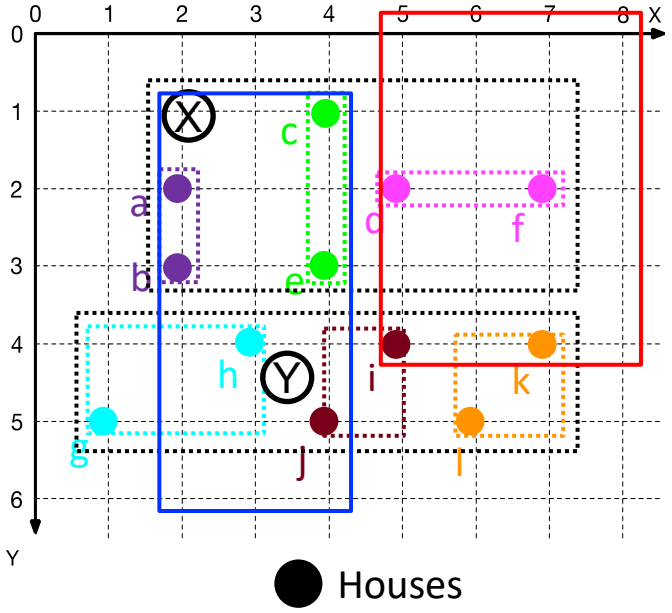


Use a bigger MBR by 1 (specific for the query)

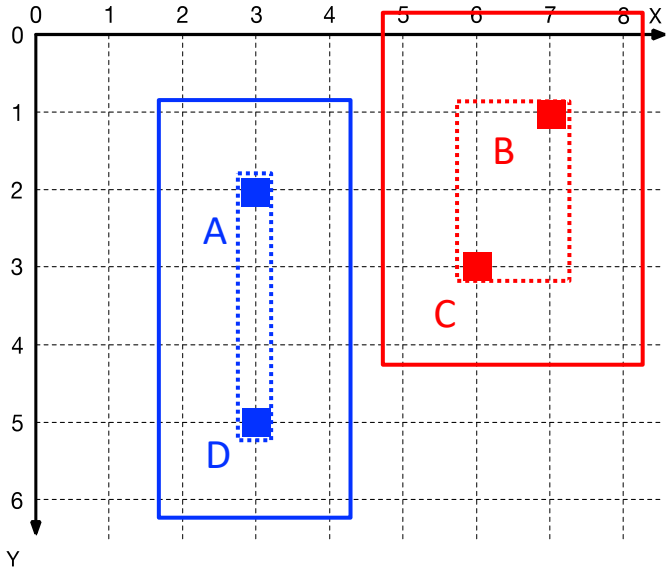


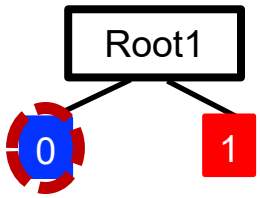


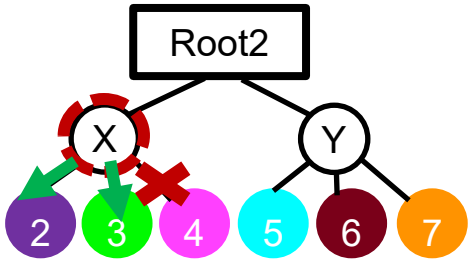
(Root1, Root2)



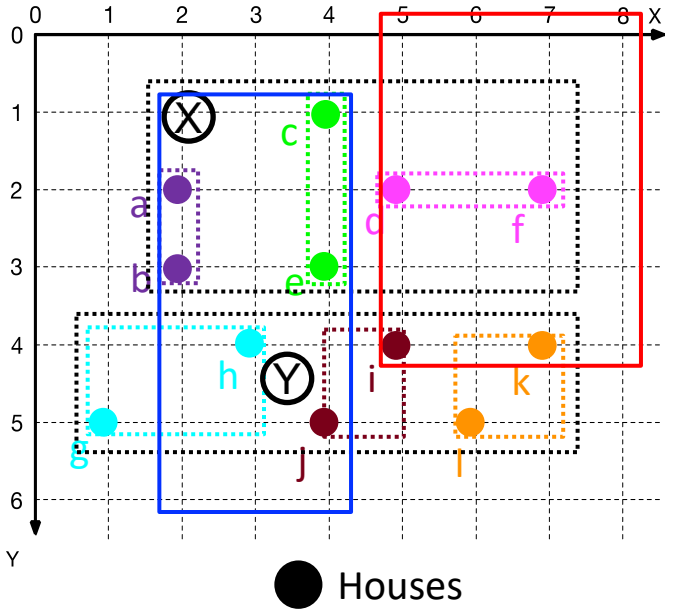
Spatial Join: Algorithm (With R Tree)



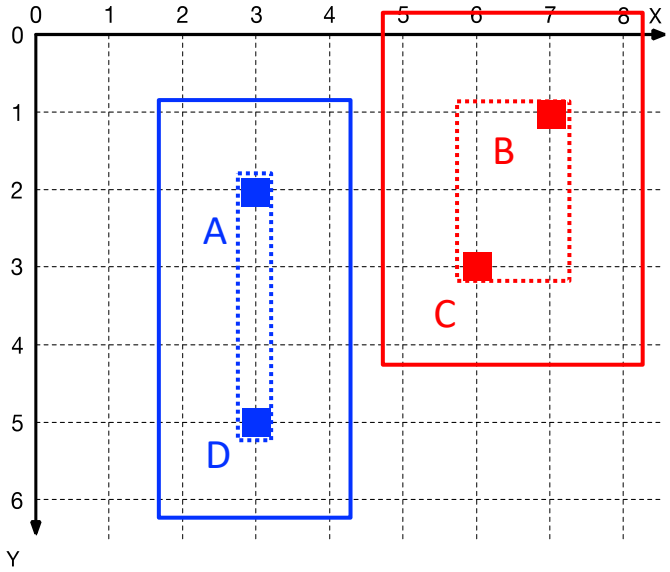


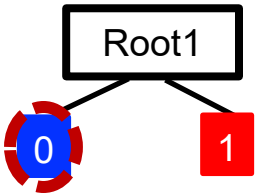


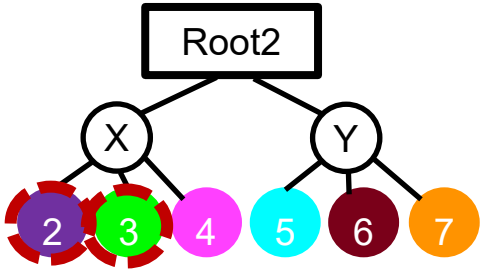
(0, X)
(0, Y)
(1, X)
(1, Y)



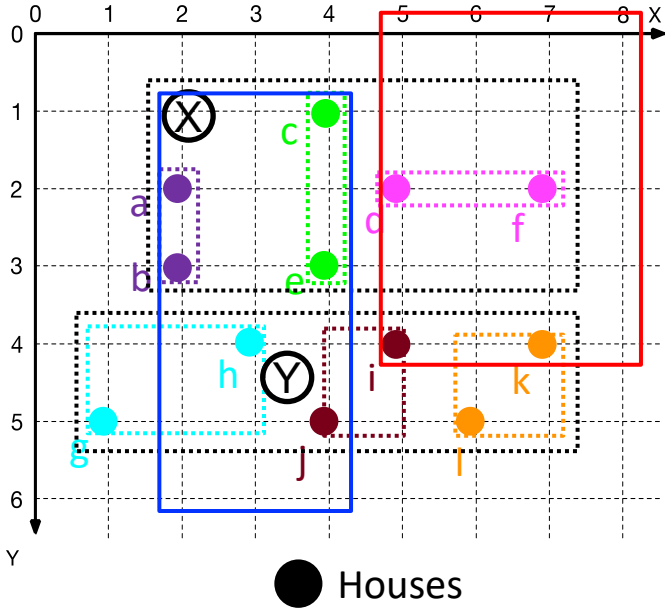
Spatial Join: Algorithm (With R Tree)



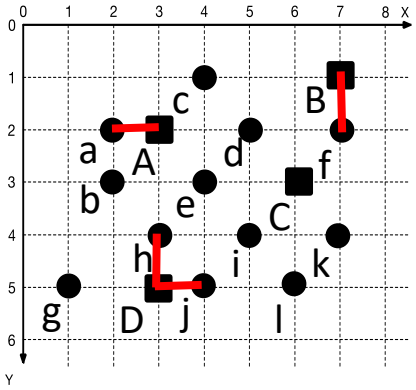


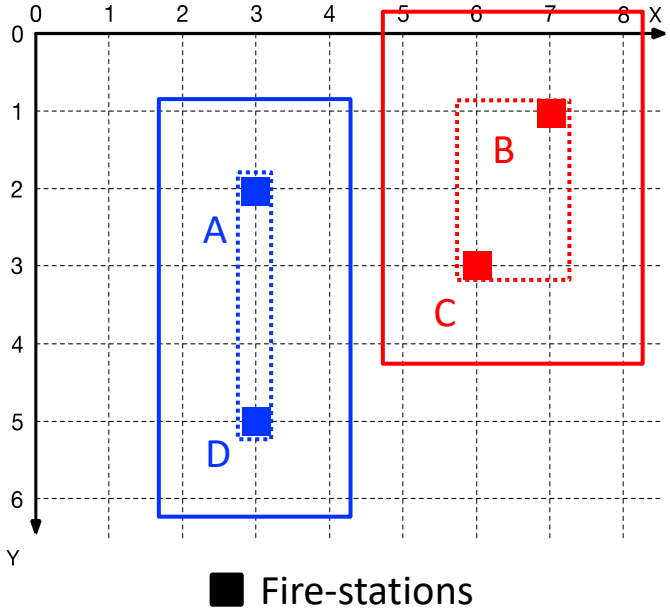


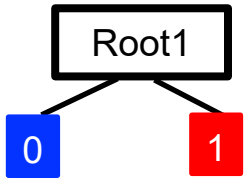
(0, 2)
(0, 3)

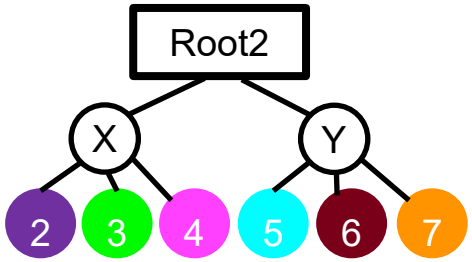


Spatial Join: Algorithm (With R T



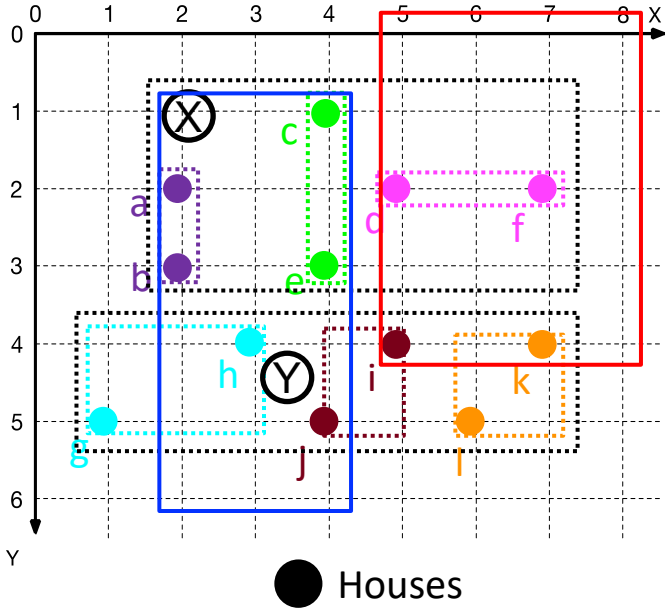






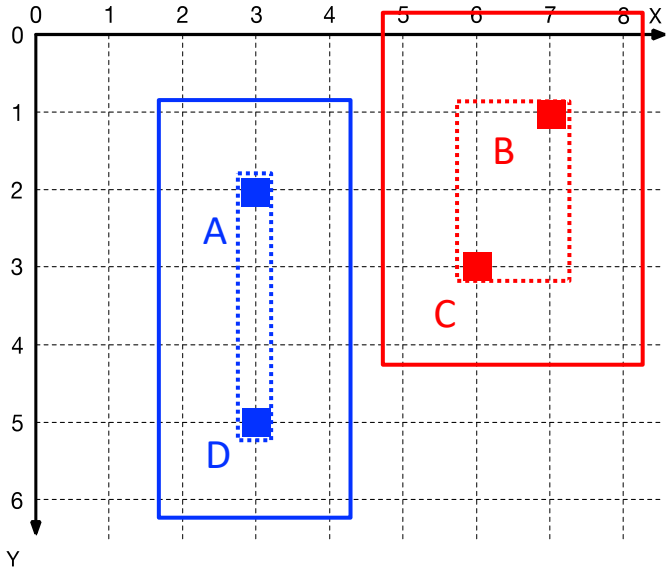
Fire-stations

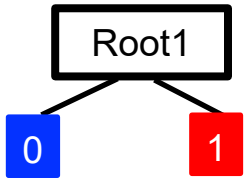
Return (A, a)

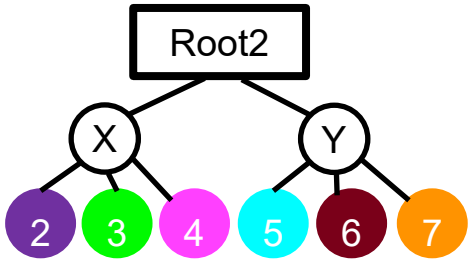


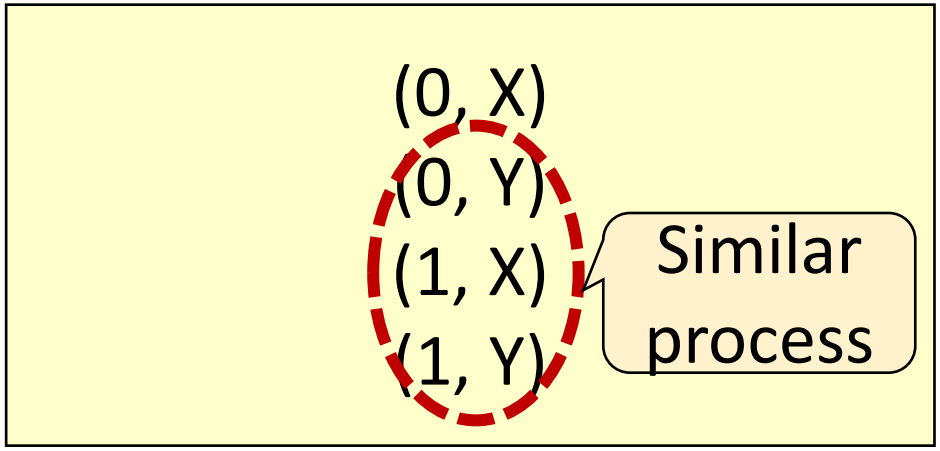
Houses

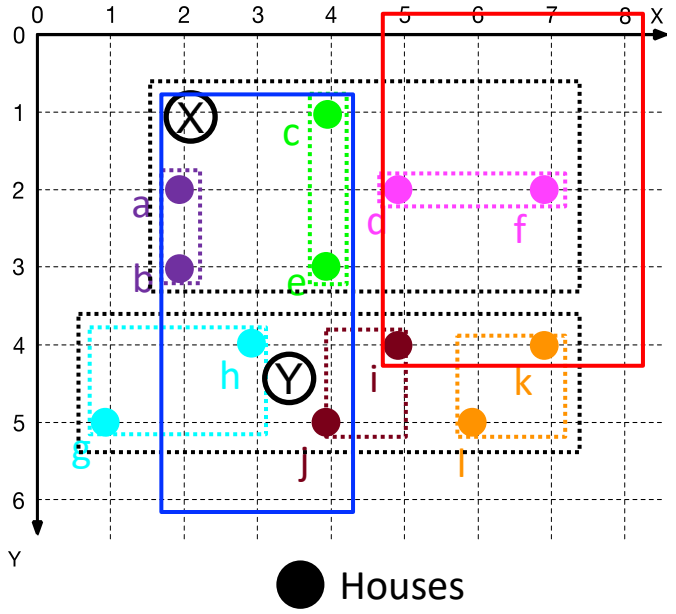
Spatial Join: Algorithm (With R Tree)



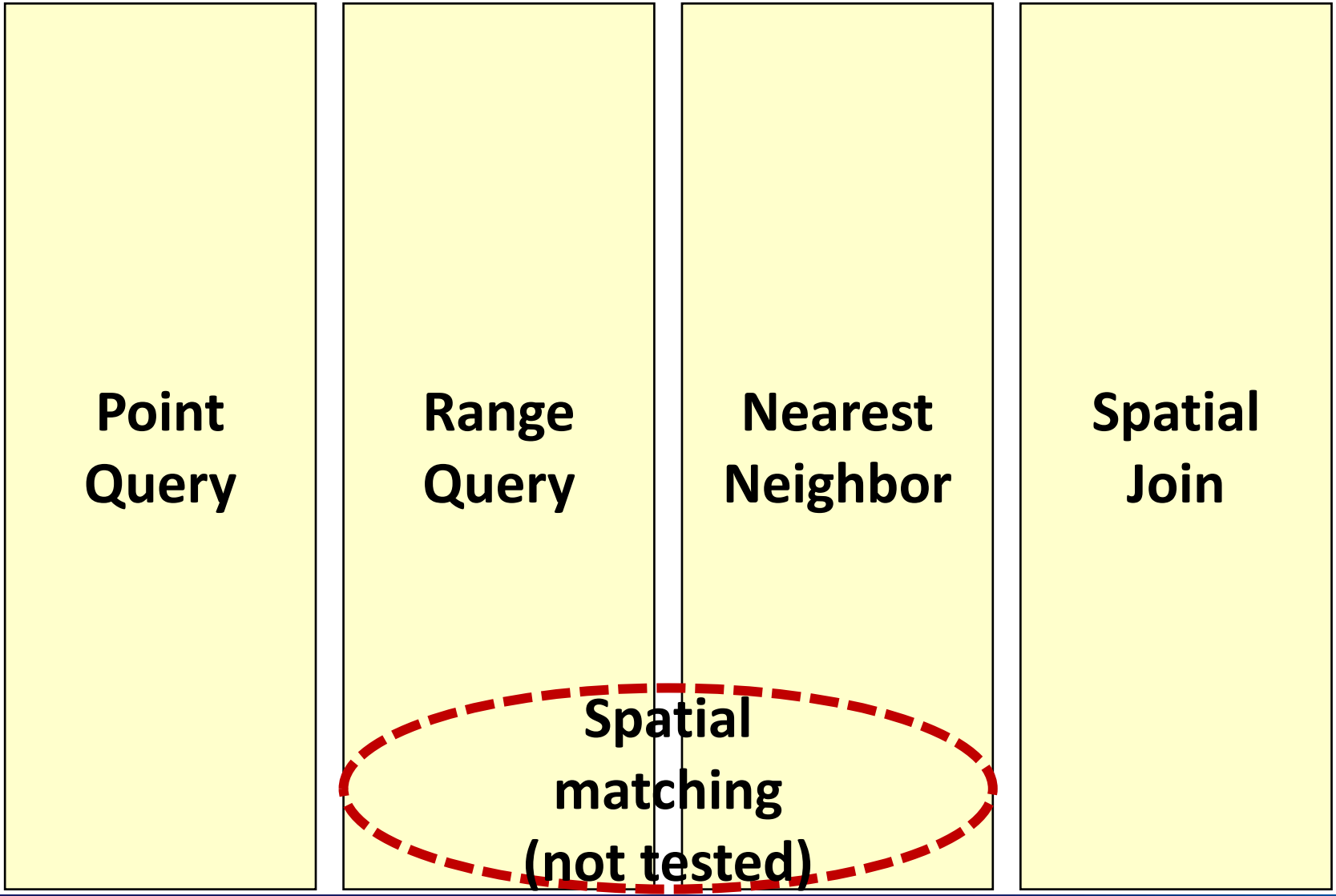






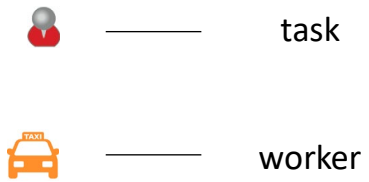


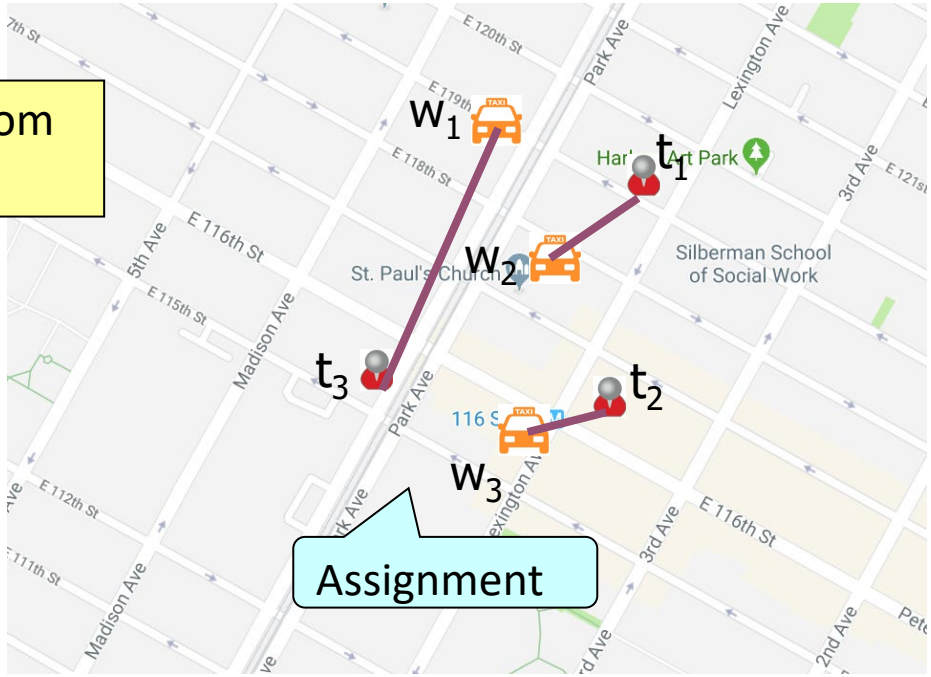
Query Processing (Spatial Data)



Spatial Matching: Introduction

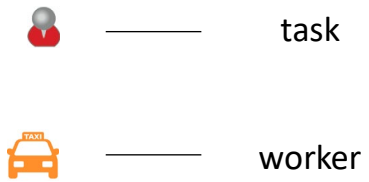
We have a few possible **assignments** from tasks to workers

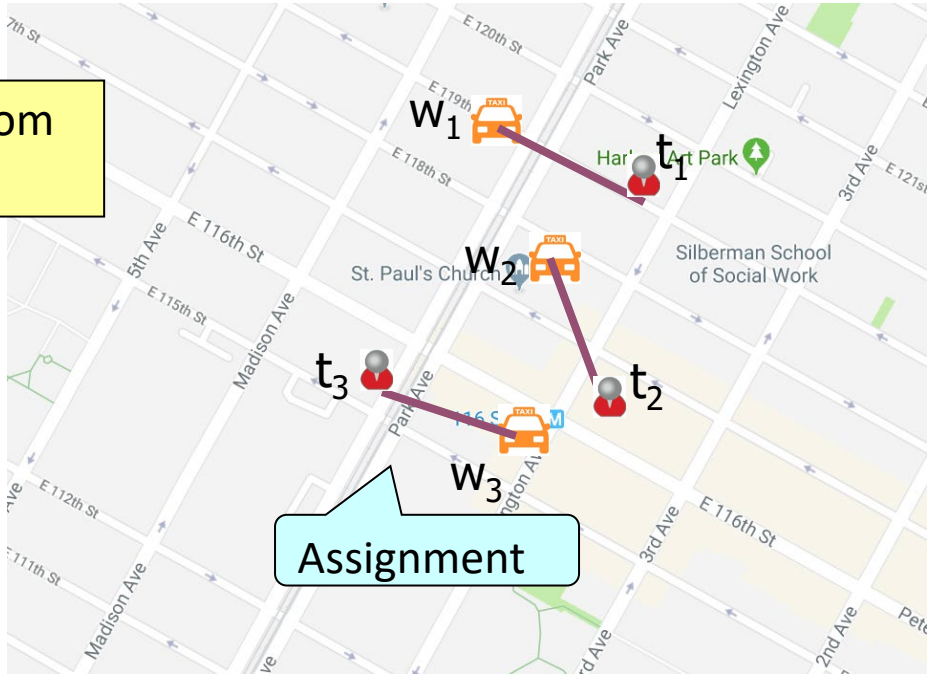




Spatial Matching: Introduction

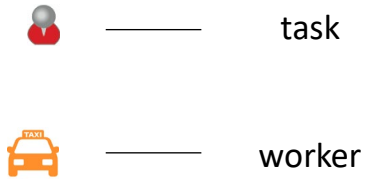
We have a few possible **assignments** from tasks to workers

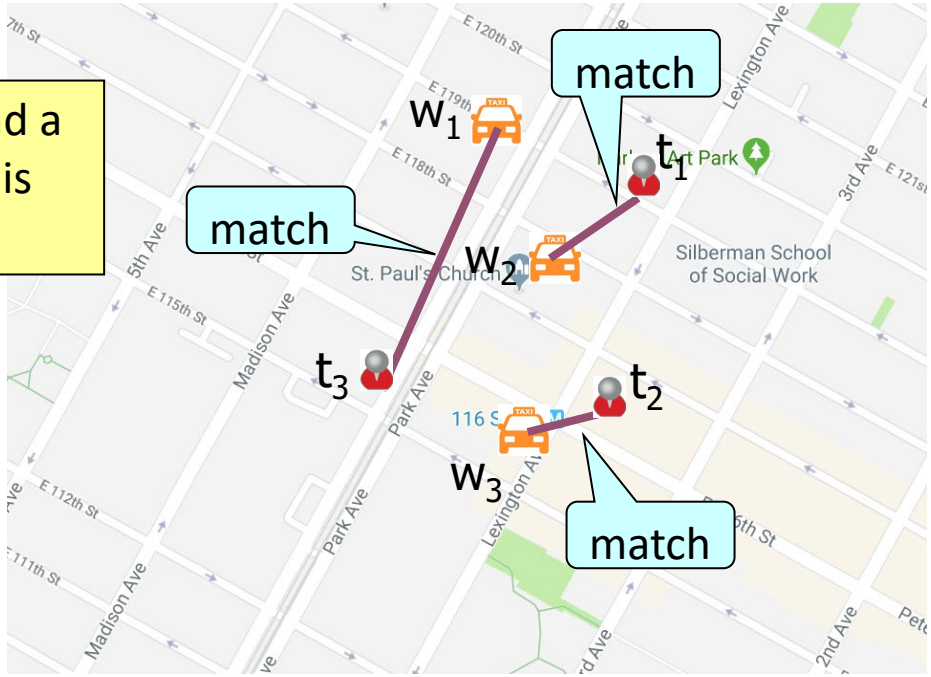




Spatial Matching: Introduction

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

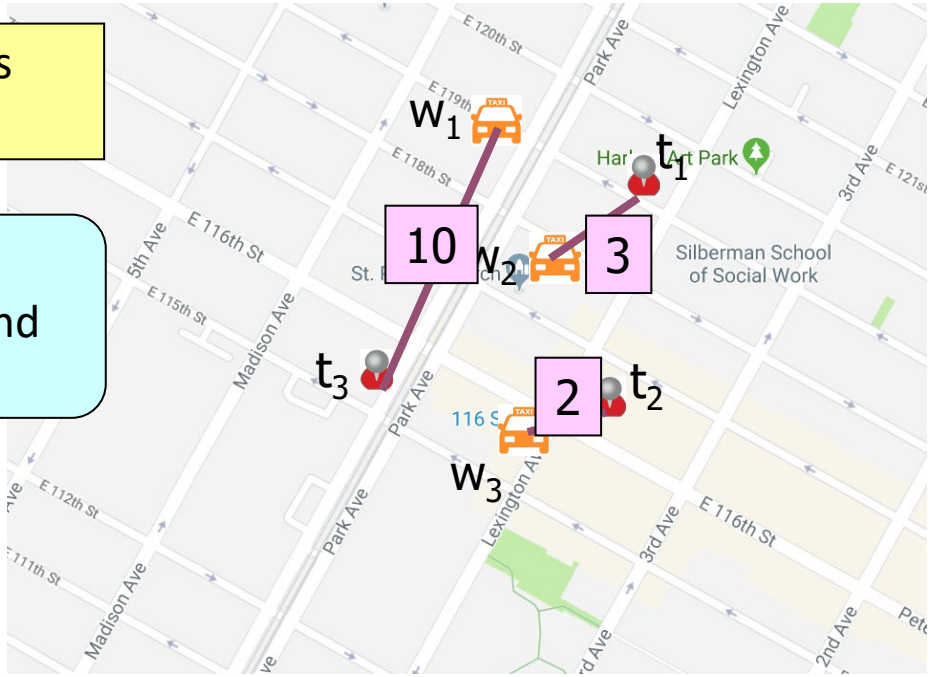




Spatial Matching: Introduction

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

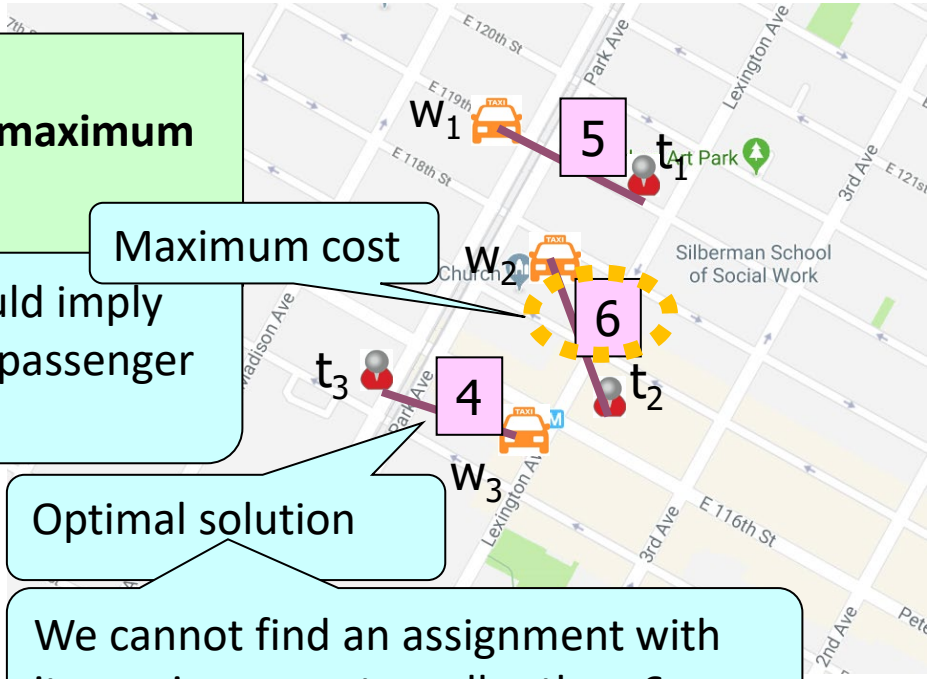


Spatial Matching: Problem Definition

Problem (Spatial matching):

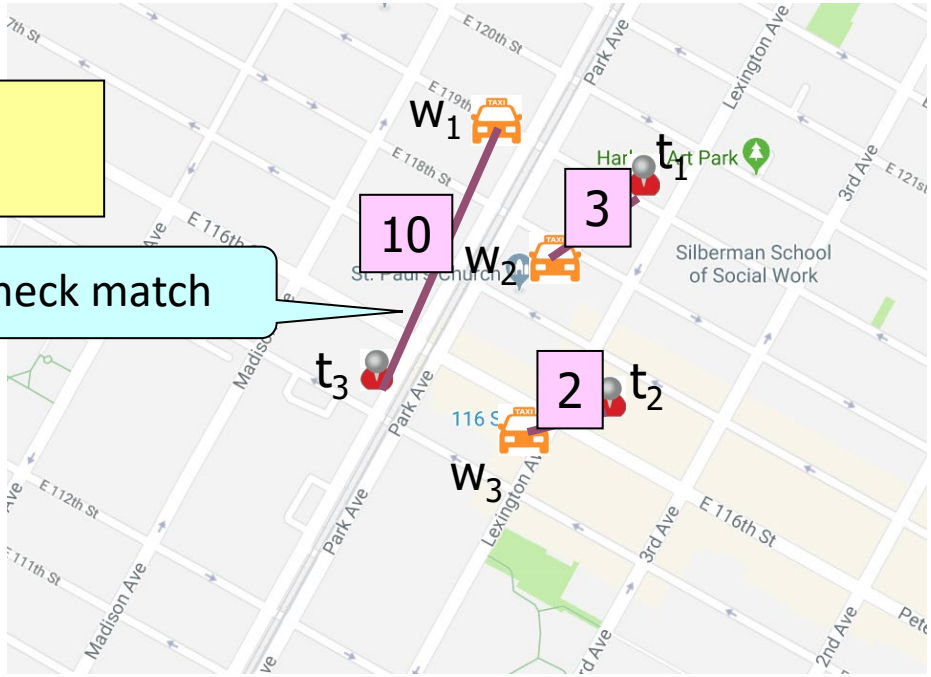
Find an assignment such that the **maximum cost** for a task is the **smallest**

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



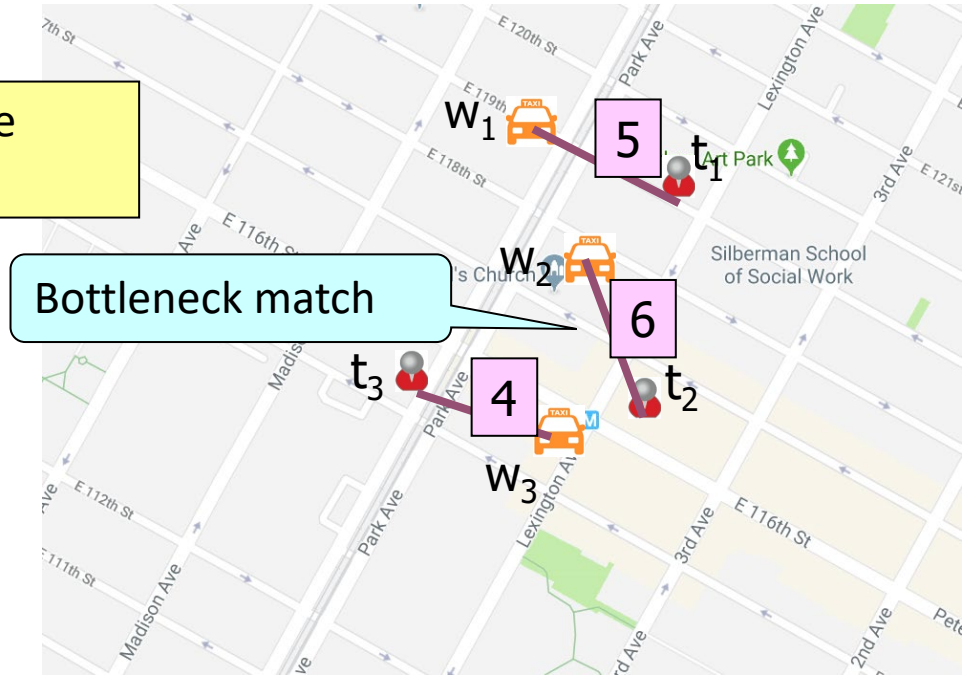
Spatial Matching: Swap-Chain

Bottleneck match: the match with the maximum cost

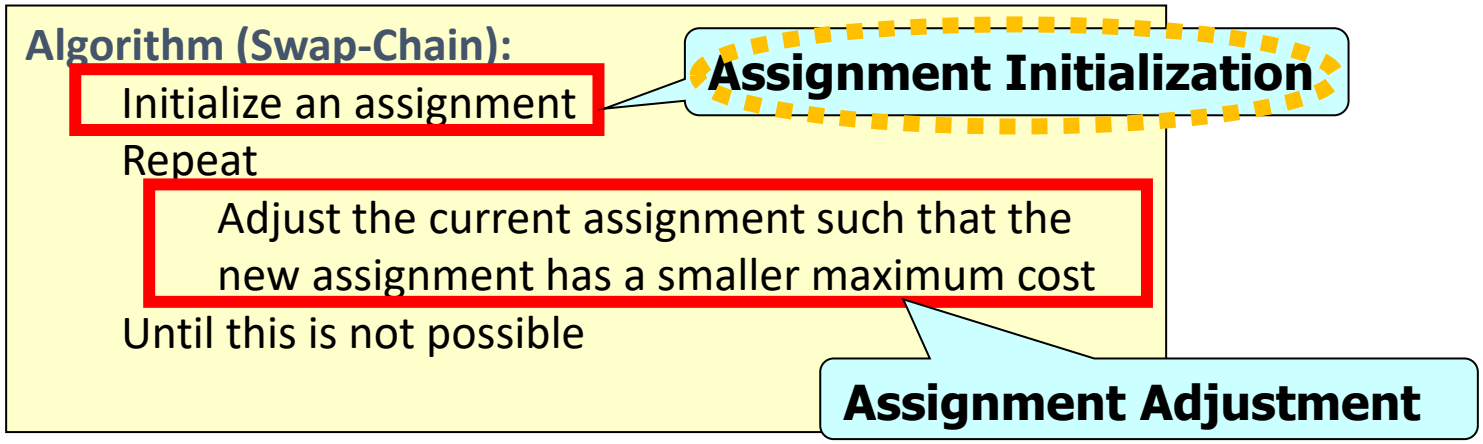


Spatial Matching: Swap-Chain

Bottleneck match: the match with the maximum cost



Spatial Matching: Swap-Chain



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:

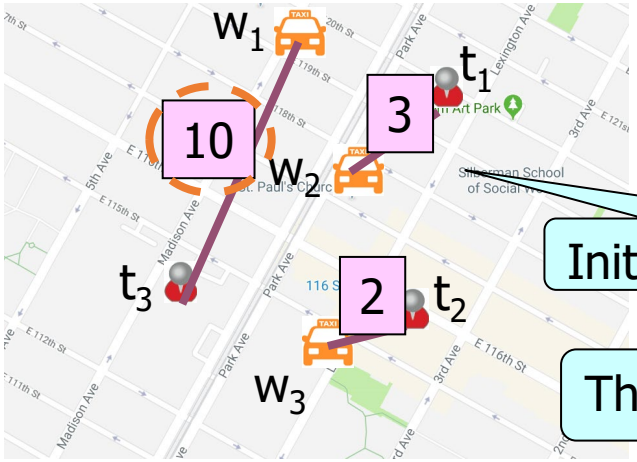
Assign for each task the **nearest** worker that has not been matched

....

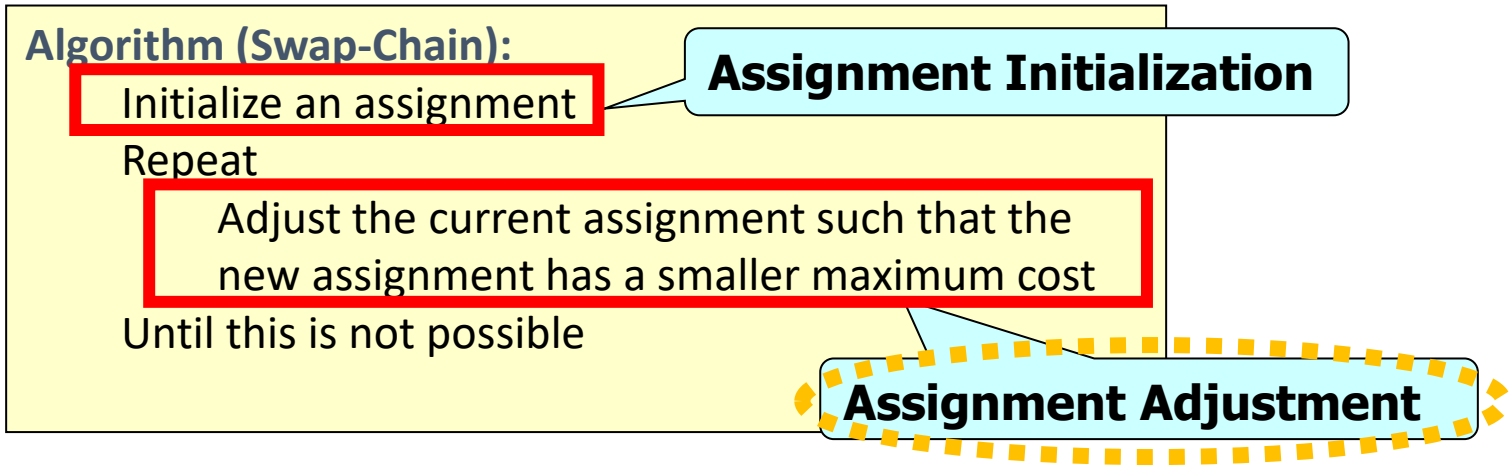
Spatial Matching: Swap-Chain

Method 2:

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



Spatial Matching: Swap-Chain

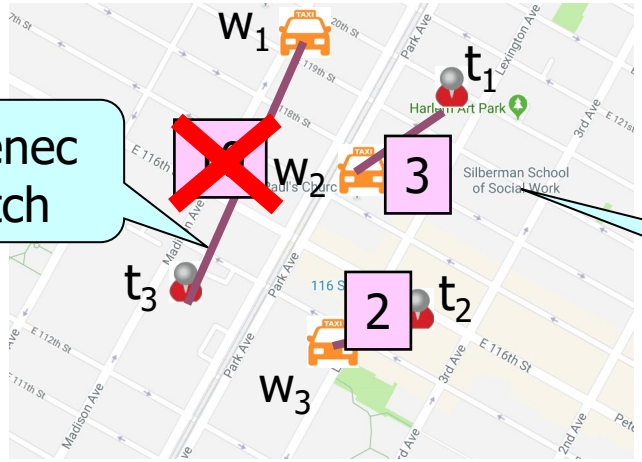


Spatial Matching: Swap-Chain

Consider the "Assignment Adjustment" procedure

maximum cost = 10

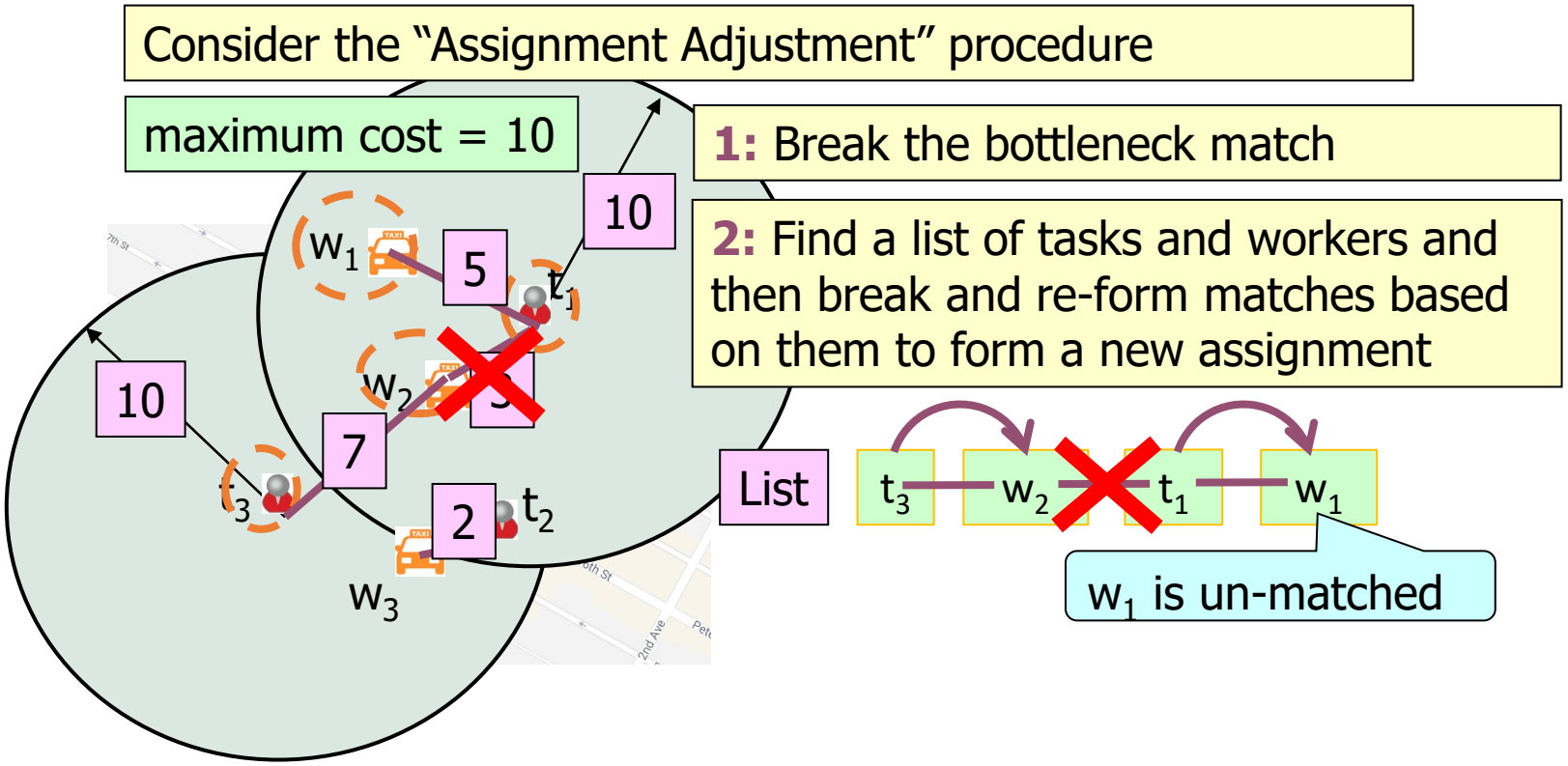
1: Break the bottleneck match



Bottleneck match

Initialized assignment

Spatial Matching: Swap-Chain

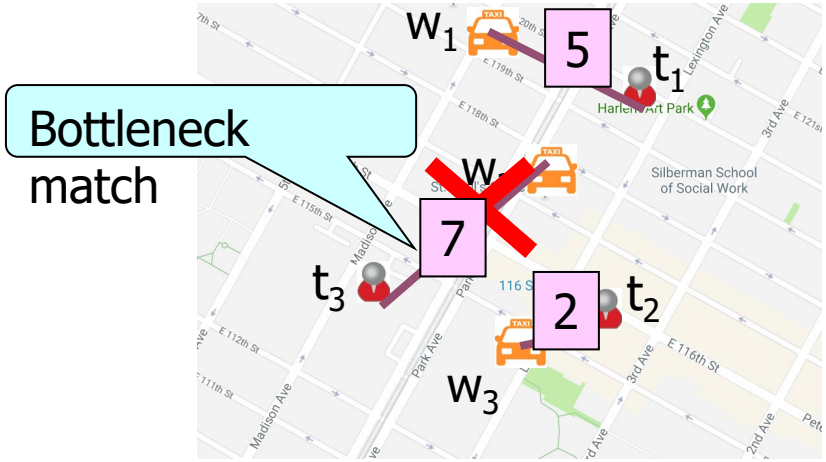


Spatial Matching: Swap-Chain

It repeats the "Assignment Adjustment" procedure

maximum cost = 7

1: Break the bottleneck match



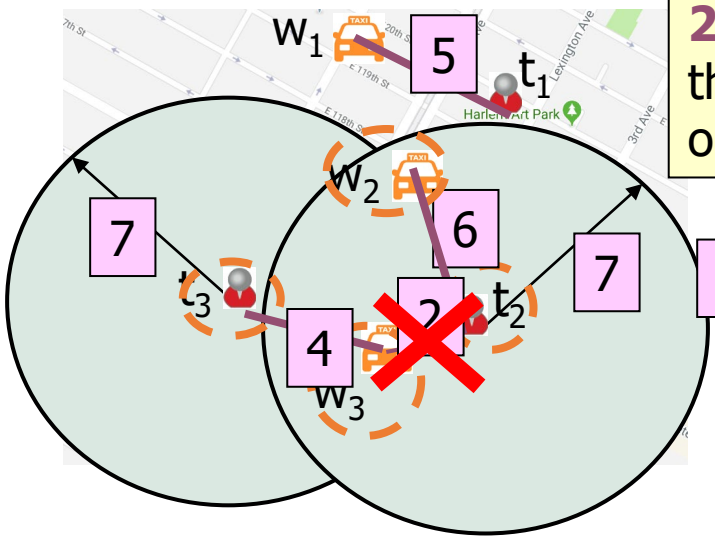
Spatial Matching: Swap-Chain

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



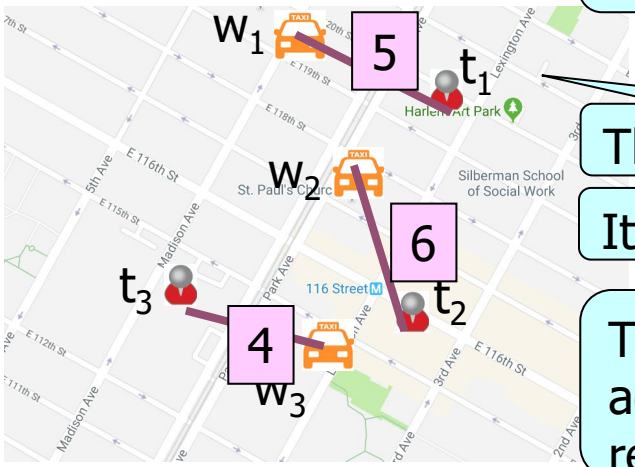
List t_3 — w_3 — t_2 — w_2

w_2 is un-matched

Spatial Matching: Swap-Chain

It cannot do the "Assignment Adjustment" procedure anymore

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)