



The Closest-Pair Problem

Algorithmic Thinking

Luay Nakhleh

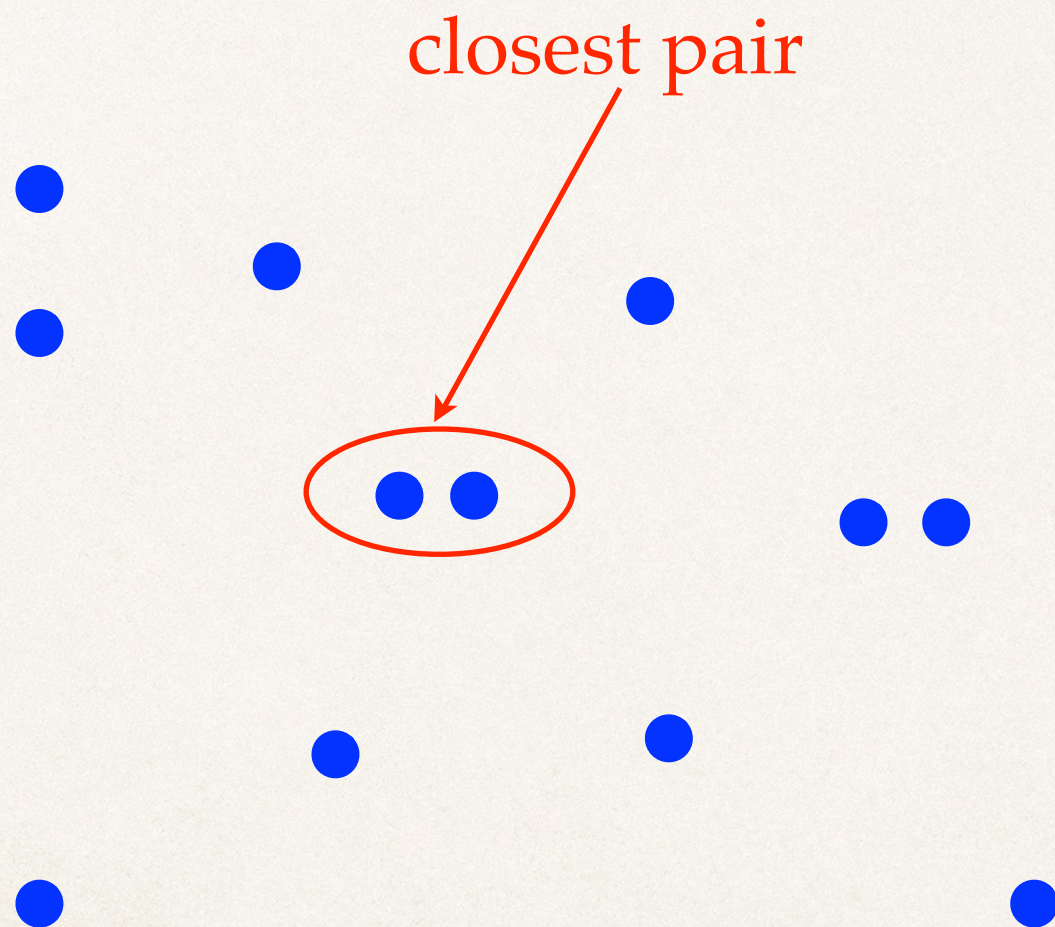
Department of Computer Science

Rice University

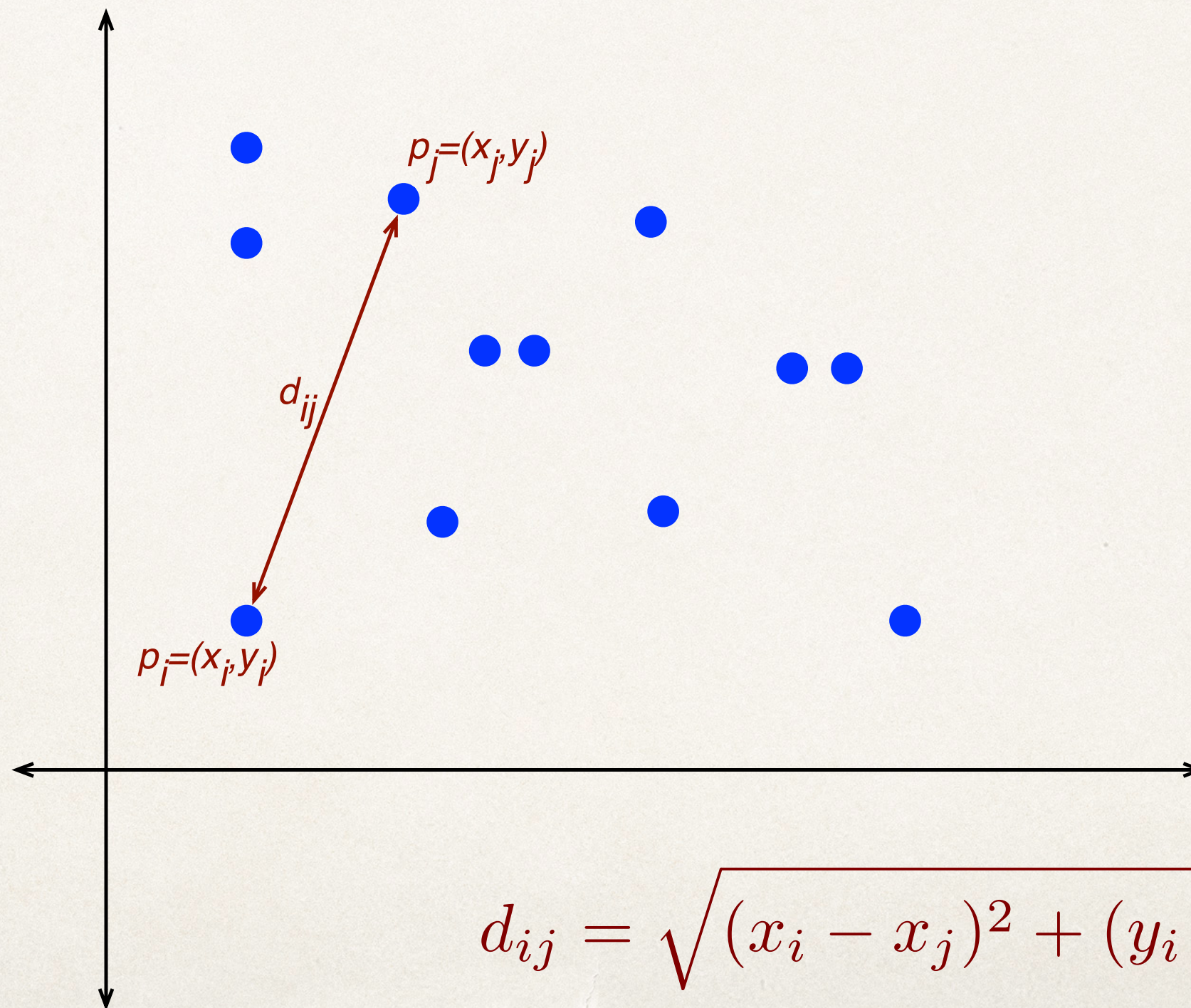
The Problem

- ❖ Input: A set P of (distinct) points in the 2D space, each given by its horizontal (x) and vertical (y) coordinates.
- ❖ Output: A pair of points $p_i, p_j \in P$ ($p_i \neq p_j$) that are closest to each other (under the Euclidian distance).

The Problem



Distance Between Points: The Euclidian Distance



$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

First Attempt: Brute-force

- ❖ A simple brute-force algorithm for the problem:
- ❖ Compute the distance between every two points
- ❖ Return a pair of points with the smallest distance

Algorithm 3: SlowClosestPair.

Input: A set P of (≥ 2) points whose i th point, p_i , is a pair (x_i, y_i) .

Output: A tuple (d, i, j) where d is the smallest pairwise distance of points in P , and i, j are the indices of two points whose distance is d .

```
1  $(d, i, j) \leftarrow (\infty, -1, -1);$   
2 foreach  $p_u \in P$  do  
3   foreach  $p_v \in P$  ( $u \neq v$ ) do  
4      $(d, i, j) \leftarrow \min\{(d, i, j), (d_{p_u, p_v}, u, v)\};$     // min compares the first element of each tuple  
5 return  $(d, i, j);$ 
```

- ❖ What is the running time of the brute-force algorithm?
- ❖ Can we do better?

Divide-and-Conquer

The Divide Phase

Draw a vertical line so that half of the points are to the left of the line and the other half are to the right of the line

Recursively find a closest pair in each half

closest pair
in the left half



closest pair
in the right half

The Merge Phase: A First Attempt

If (p_i, p_j) is a closest pair on the left, and (p_k, p_l) is a closest pair on the right, then return (p_i, p_j) if $d_{ij} < d_{kl}$, and return (p_k, p_l) otherwise.

closest pair
in the left half



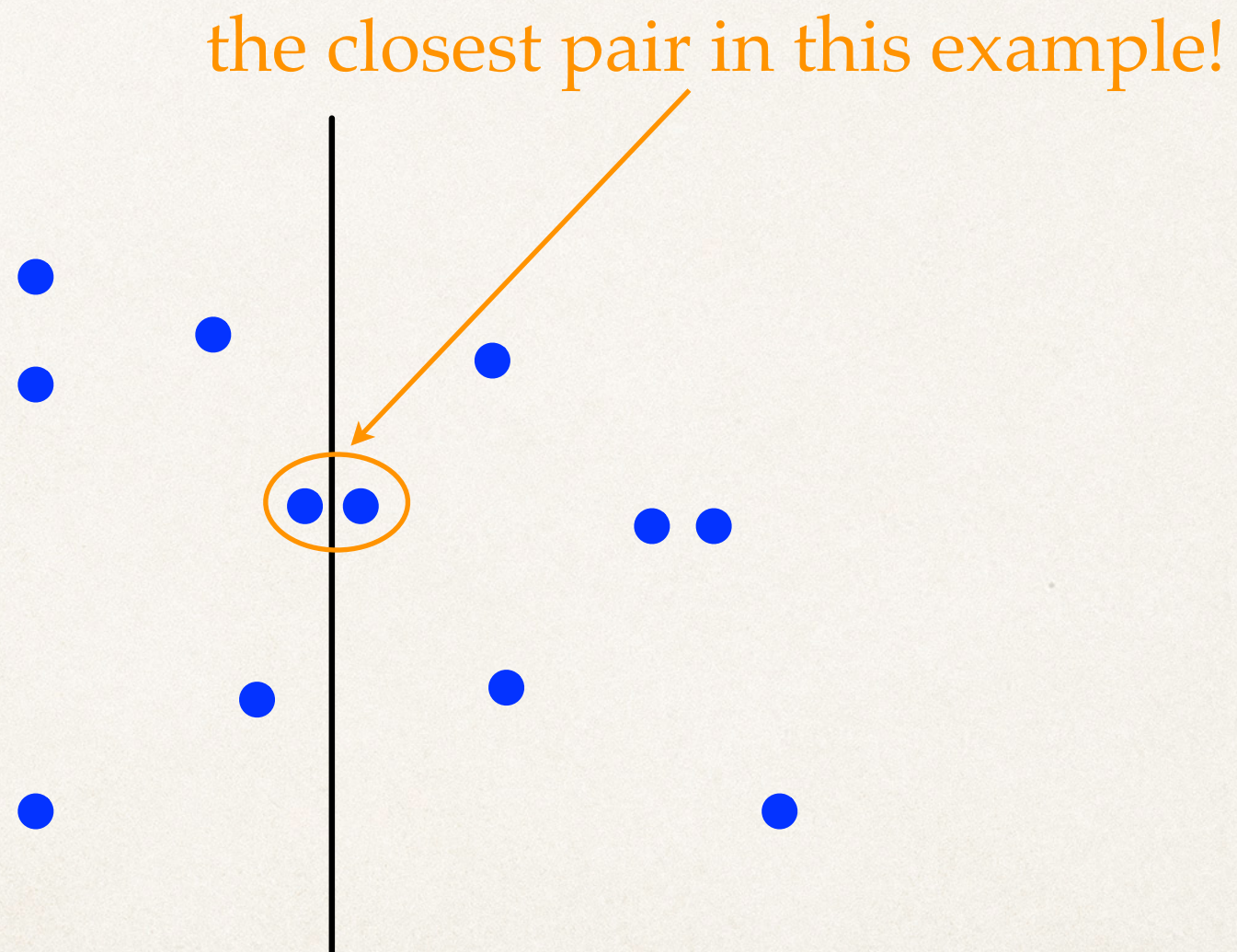
closest pair
in the right half

return

The Merge Phase: A First Attempt

- ❖ This attempt at solving the problem doesn't work if at any iteration of the divide phase, two closest points reside on either side of the vertical line!

The Merge Phase: A First Attempt



The Merge Phase: A Second Attempt

- ❖ Suppose we've found a closest pair (p_i, p_j) on the left, and a closest pair (p_k, p_l) on the right
- ❖ Now, compute the distance between every point on the left and every point on the right, and find a closest such pair of points, say (p_r, p_s)
- ❖ Finally, return a pair out of the three that has the smallest distance, and we're done!

The Merge Phase: A Second Attempt

- ❖ The problem with this second attempt is that it is basically the brute-force algorithm!
- ❖ Can we do the Merge phase more efficiently?
- ❖ The answer is yes!

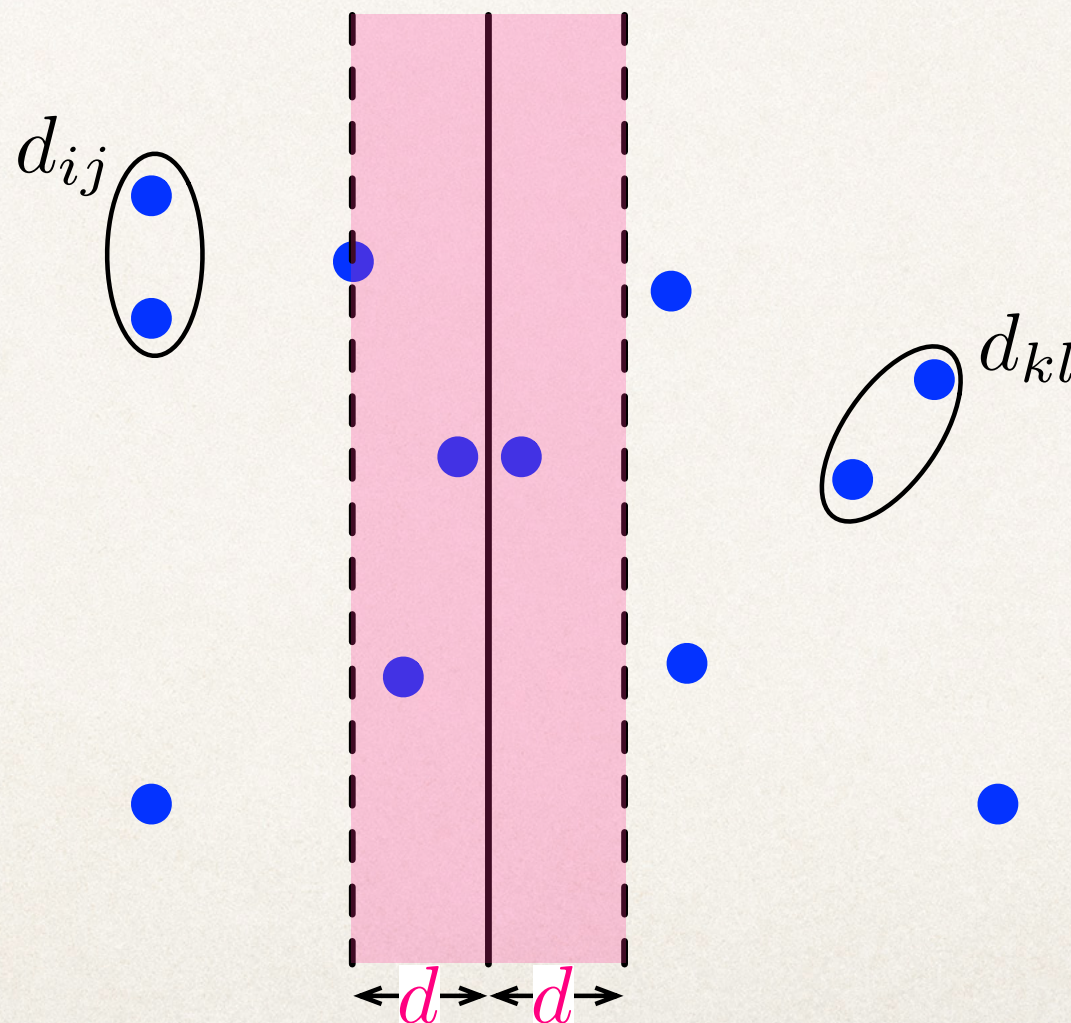
The Merge Phase: A Third Attempt

- ❖ Key observation 1:
 - ❖ If d_{ij} is the smallest pairwise distance on the left and d_{kl} is the smallest pairwise distance on the right, then we need to consider points on either side of the line that are at most $d = \min\{d_{ij}, d_{kl}\}$ distance apart.

The Merge Phase: A Third Attempt

- ✧ Key observation 1:

$$d = \min\{d_{ij}, d_{kl}\}$$



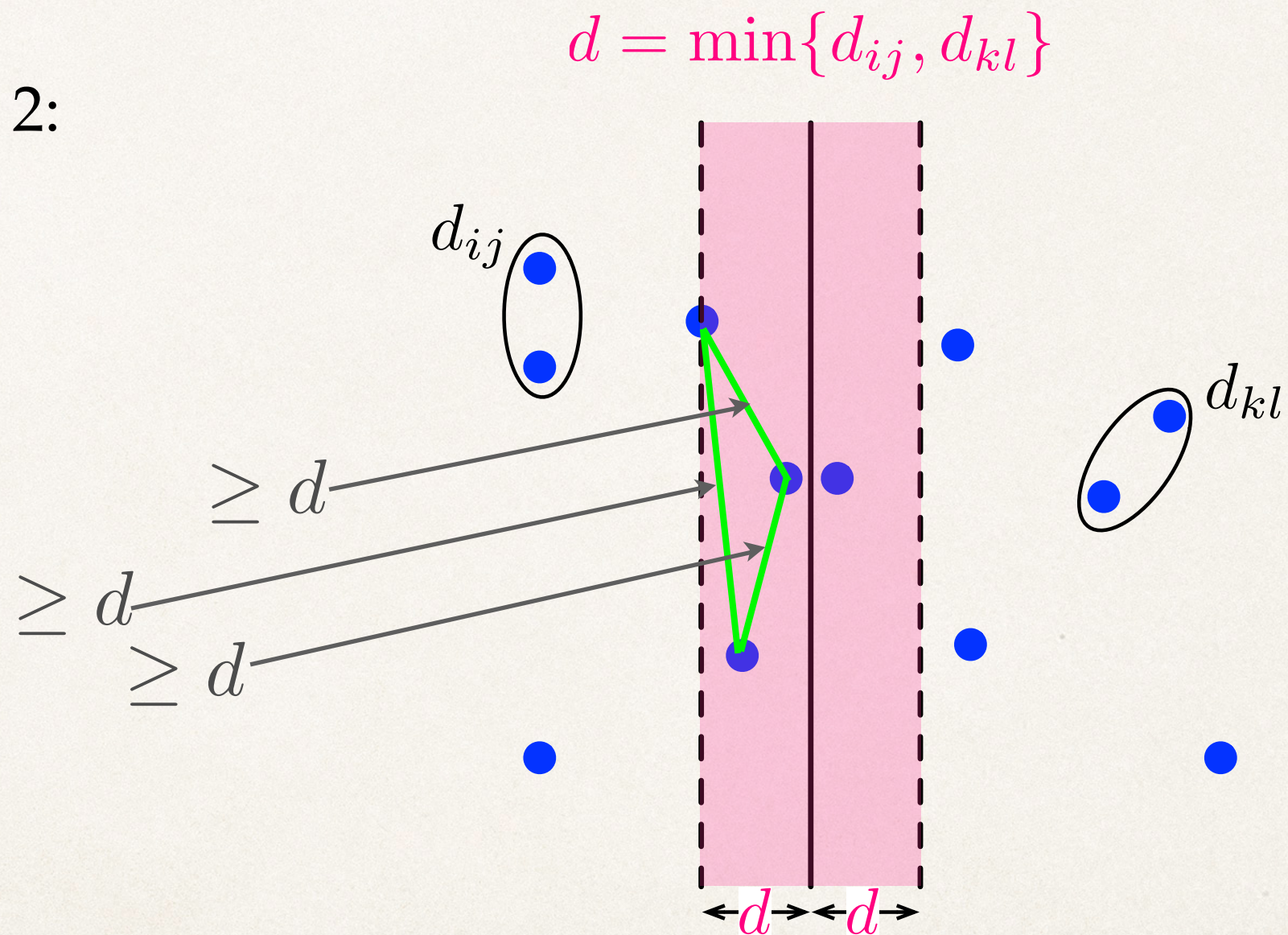
The Merge Phase: A Third Attempt

- ❖ Key observation 2:

- ❖ In each half (left and right) of the (shaded) rectangle, every pair of points is at least distance d apart (why?).

The Merge Phase: A Third Attempt

- ❖ Key observation 2:



Due to this observation, non-consecutive points in each half of the rectangle must be separated by some minimum vertical distance.

The Merge Phase: A Third Attempt

- ❖ Consequence of observation 2:
 - ❖ Let $S[0..m-1]$ be an array (or a list) of all the points inside the rectangle sorted by nondecreasing order of their y -coordinates. If the distance between $S[i]$ and $S[j]$ is smaller than d , then $|i-j| < 4$.

The Merge Phase: A Third Attempt

- ❖ Putting all three observations together gives the idea for efficiently finding a closest pair across the vertical line:
- ❖ Identify all points whose horizontal distance from the vertical line is $\leq d$.
- ❖ Let $S[0..m-1]$ be an array (or a list) of all these points sorted by nondecreasing order of their y -coordinates.
- ❖ Going through the element of S in order, for each element $S[i]$ inspect the next three ones to find the closest to $S[i]$, and record the pair of indices i and j which correspond to the closest pair thus found.

✧ And, now to the full algorithm...

Algorithm 4: FastClosestPair.

Input: A set P of (≥ 2) points whose i th point, p_i , is a pair (x_i, y_i) , **sorted** in nondecreasing order of their horizontal (x) coordinates.

Output: A tuple (d, i, j) where d is the smallest pairwise distance of the points in P , and i, j are the indices of two points whose distance is d .

```
1  $n \leftarrow |P|;$ 
2 if  $n \leq 3$  then
3    $(d, i, j) \leftarrow \text{SlowClosestPair}(P);$ 
4 else
5    $m \leftarrow \lfloor n/2 \rfloor;$ 
6    $P_\ell \leftarrow \{p_i : 0 \leq i \leq m-1\}; P_r \leftarrow \{p_i : m \leq i \leq n-1\};$ 
7    $(d_\ell, i_\ell, j_\ell) \leftarrow \text{FastClosestPair}(P_\ell);$ 
8    $(d_r, i_r, j_r) \leftarrow \text{FastClosestPair}(P_r);$ 
9    $(d, i, j) \leftarrow \min\{(d_\ell, i_\ell, j_\ell), (d_r, i_r + m, j_r + m)\};$ 
10   $mid \leftarrow \frac{1}{2}(x_{m-1} + x_m);$ 
11   $(d, i, j) \leftarrow \min\{(d, i, j), \text{ClosestPairStrip}(P, mid, d)\};$ 
12 return  $(d, i, j);$ 
```

base case

// P_ℓ and P_r are also sorted

divide

// center line of strip

merge

Algorithm 5: ClosestPairStrip.

Input: A set P of points whose i th point, p_i , is a pair (x_i, y_i) ; mid and w , both of which are real numbers.

Output: A tuple (d, i, j) where d is the smallest pairwise distance of points in P whose horizontal (x) coordinates are within w from mid .

- 1 Let S be a list of the set $\{i : |x_i - mid| < w\}$;
 - 2 Sort the indices in S in nondecreasing order of the vertical (y) coordinates of their associated points;
 - 3 $k \leftarrow |S|$;
 - 4 $(d, i, j) \leftarrow (\infty, -1, -1)$;
 - 5 **for** $u \leftarrow 0$ **to** $k - 2$ **do**
 - 6 **for** $v \leftarrow u + 1$ **to** $\min\{u + 3, k - 1\}$ **do**
 - 7 $(d, i, j) \leftarrow \min\{(d, i, j), (d_{p_{S[u]}, p_{S[v]}}, S[u], S[v])\}$;
 - 8 **return** (d, i, j) ;
-