CSE 331: Algorithms & Complexity

"Closest Pair"

Prof. Charlie Anne Carlson (She/Her)

Lecture 26

Monday Nov 3, 2025

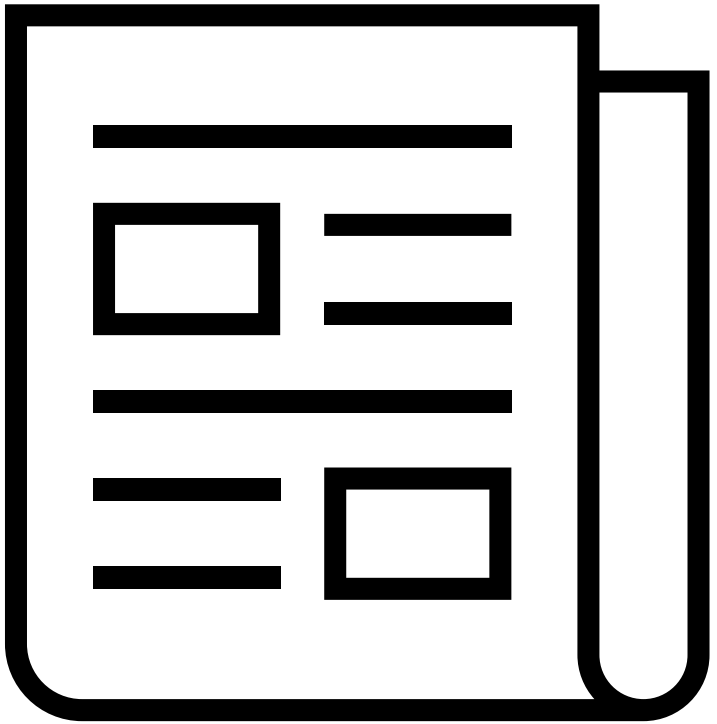
University at Buffalo®

Schedule

1. Course Updates
2. Closest Pair

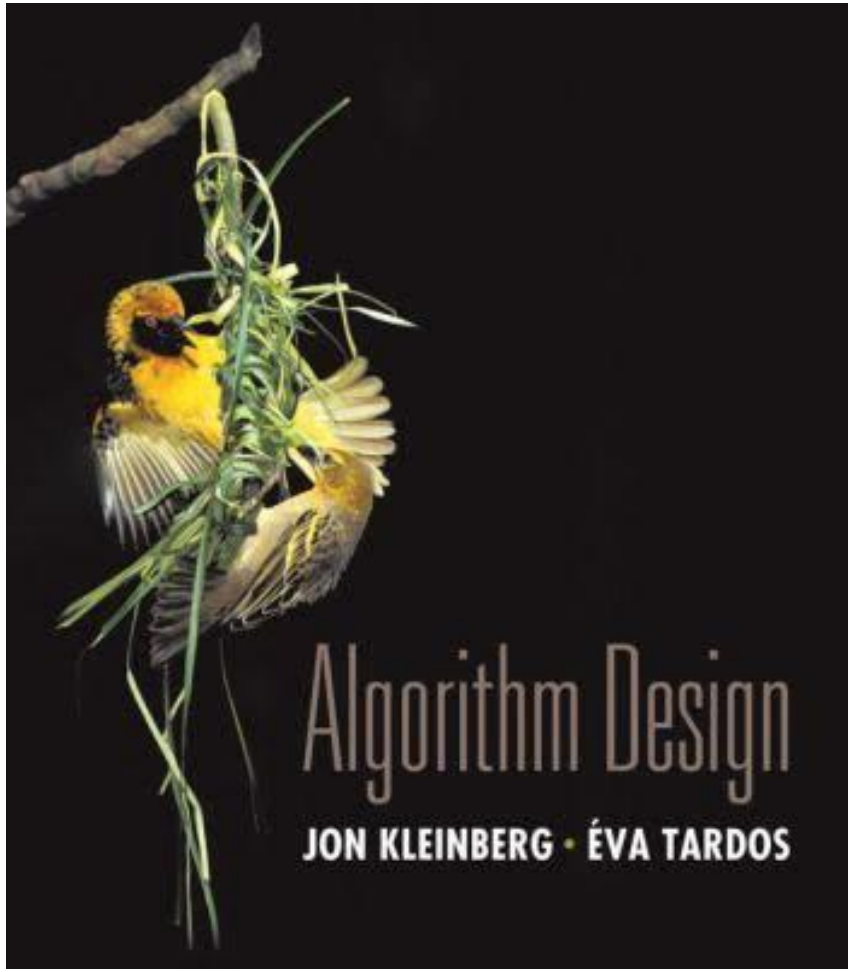
Course Updates

- HW 6 Out
  - Autolab Soon
  - Due November 11th
- Group Project
  - Code 1 & 2 Due Today
  - Reflections 1 & 2 Due Today
- Check Piazza for Google Form Review Link



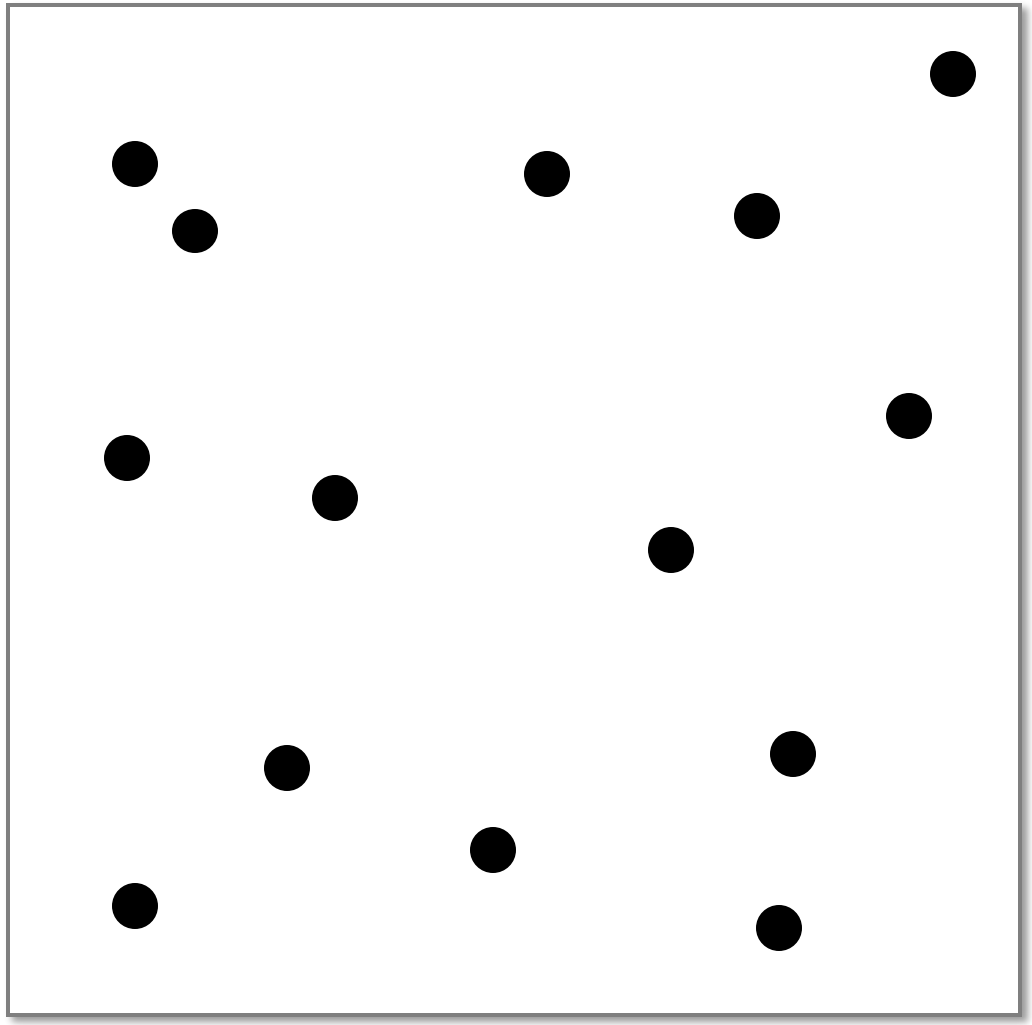
Reading

- You should have read:
  - Finished 5.5
  - Finished 5.4
  - Finished Unraveling the mystery behind the identity
- Before Next Class:
  - Start 6.1
  - Start 6.2



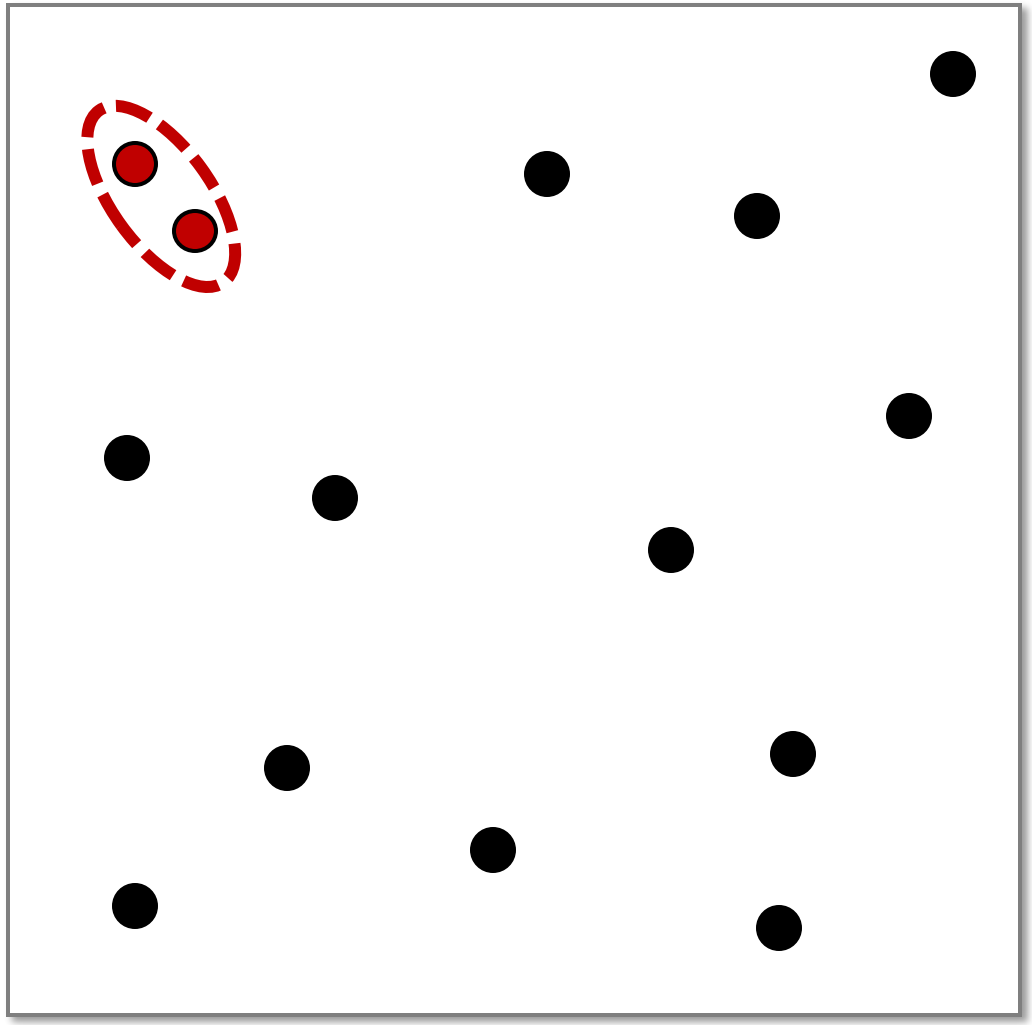
Closest Pair 2D

- **Input**: Given n points in the plane.
- **Goal**: Find pair of points with smallest Euclidean distance between them.



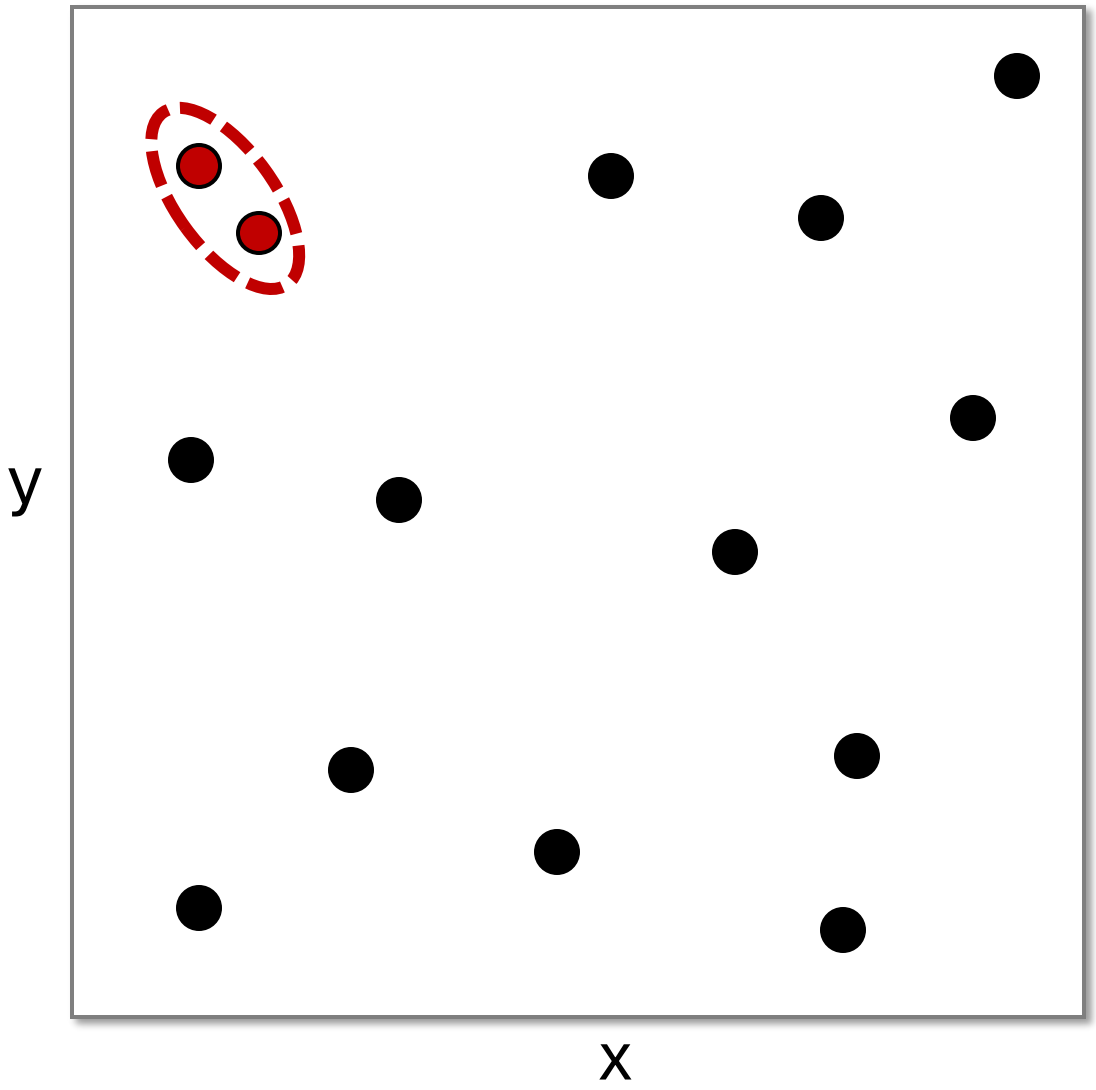
Closest Pair 2D

- **Input**: Given n points in the plane.
- **Goal**: Find pair of points with smallest Euclidean distance between them.



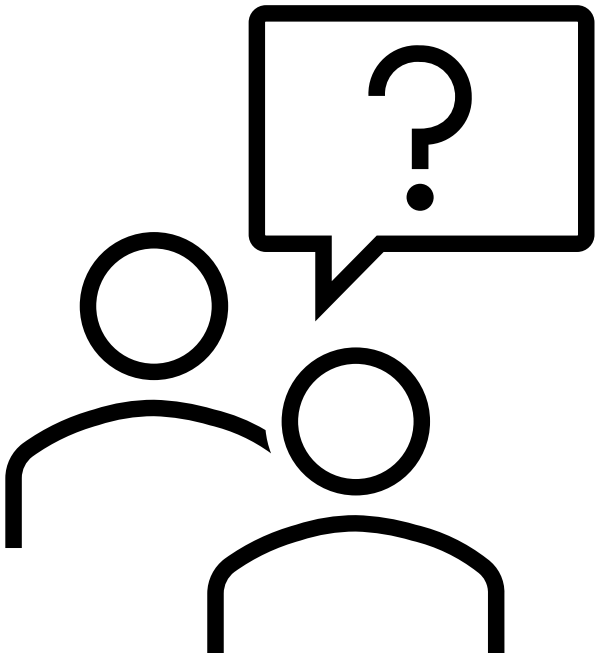
Closest Pair 2D

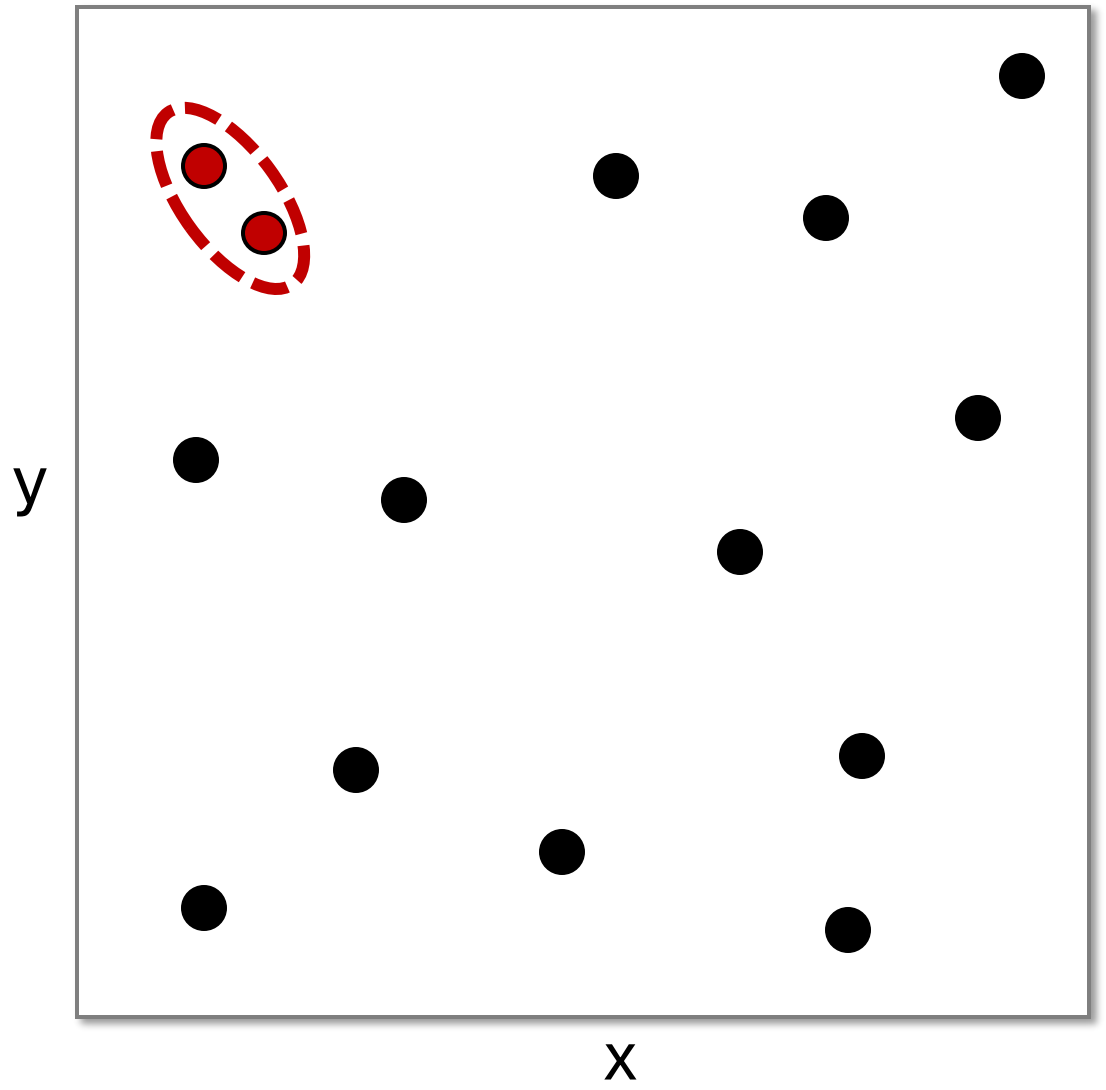
- **Input**: Given n points in the plane.
- **Goal**: Find pair of points with smallest Euclidean distance between them.
- **Assumptions:**
  - No two points have same x and y coordinate.



Closest Pair 2D

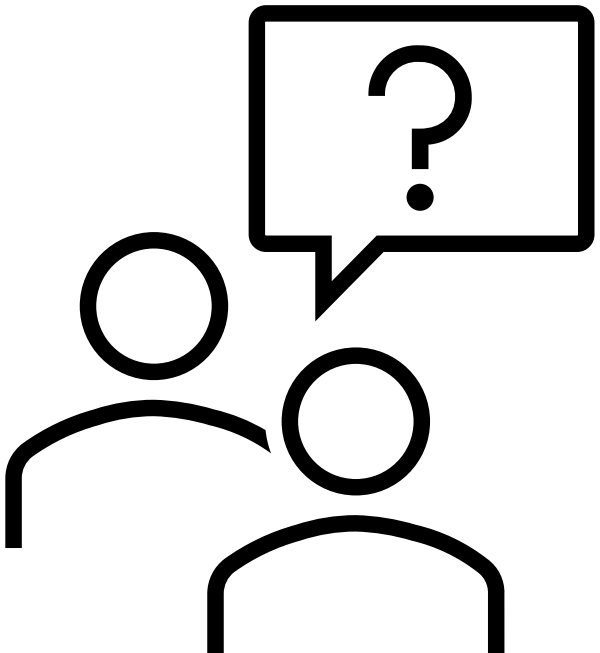
- **Question:** What is a simple algorithm that solves this problem in polynomial time?

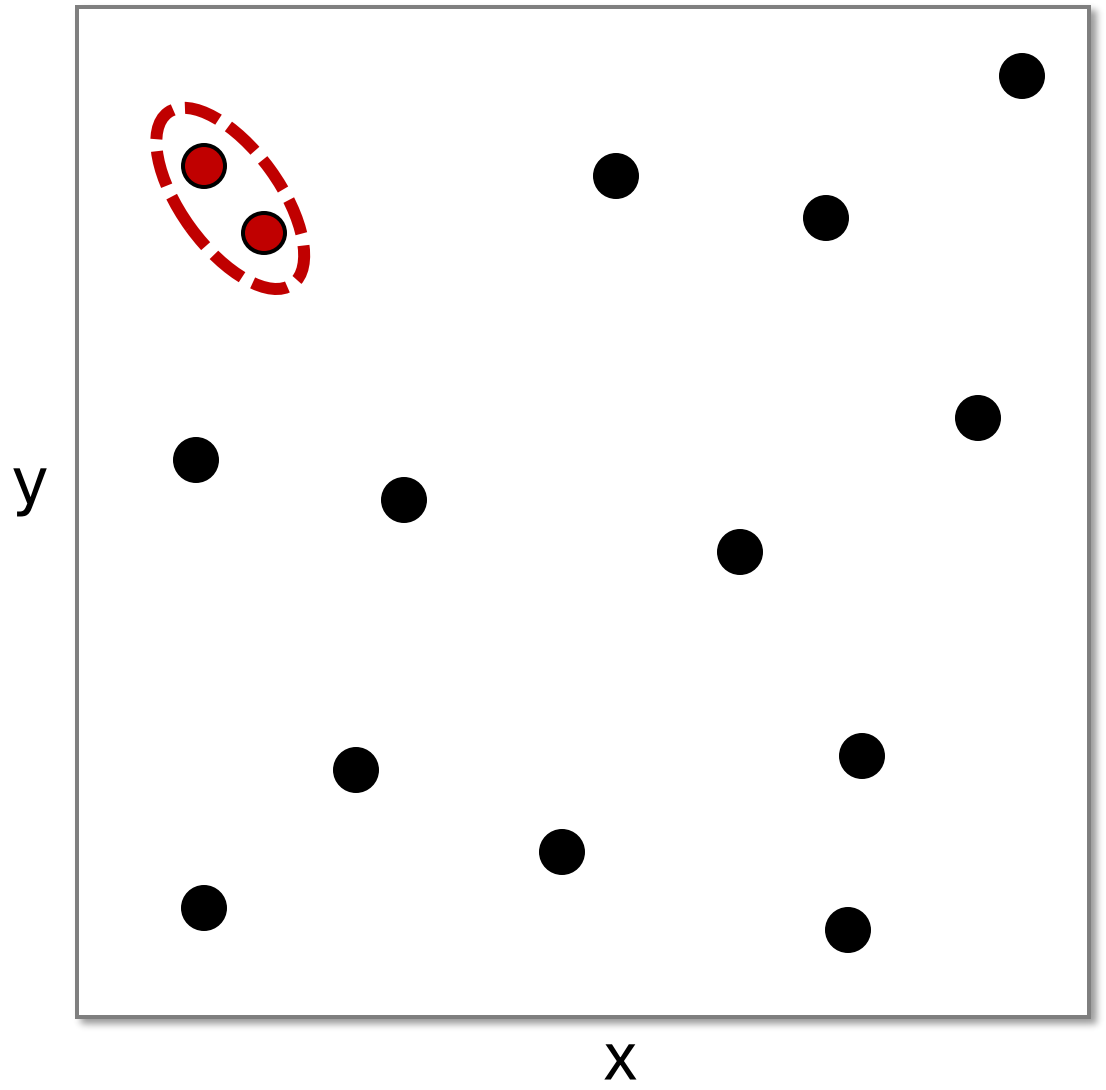




Closest Pair 2D

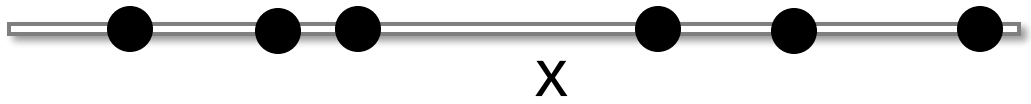
- **Answer:** Check all pairs in O(n^2) time.





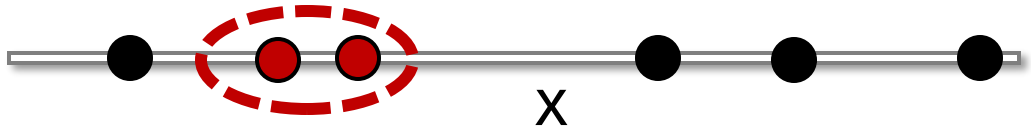
Closest Pair 1D

- **Input**: Given n points ON the LINE.
- **Goal**: Find pair of points with smallest Euclidean distance between them.
- **Assumptions:**
  - No two points have same x coordinate.



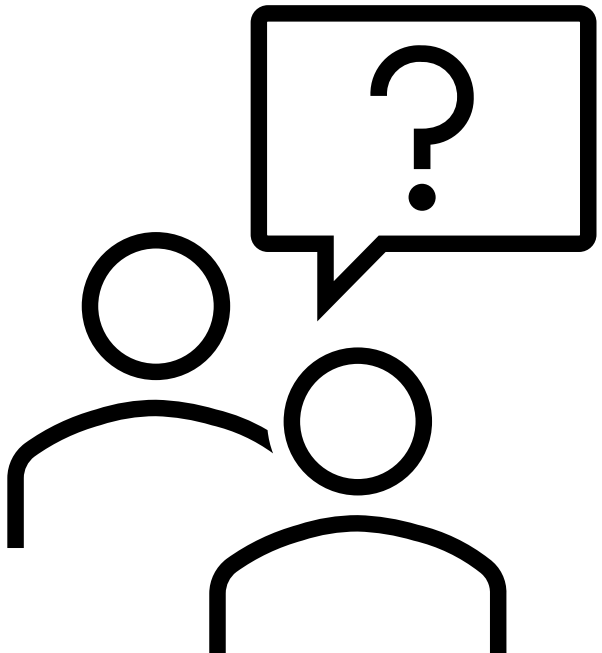
Closest Pair 1D

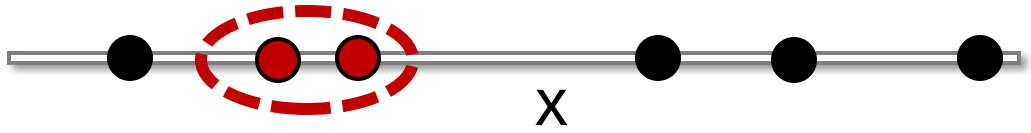
- **Input**: Given n points ON the LINE.
- **Goal**: Find pair of points with smallest Euclidean distance between them.
- **Assumptions:**
  - No two points have same x coordinate.



Closest Pair 1D

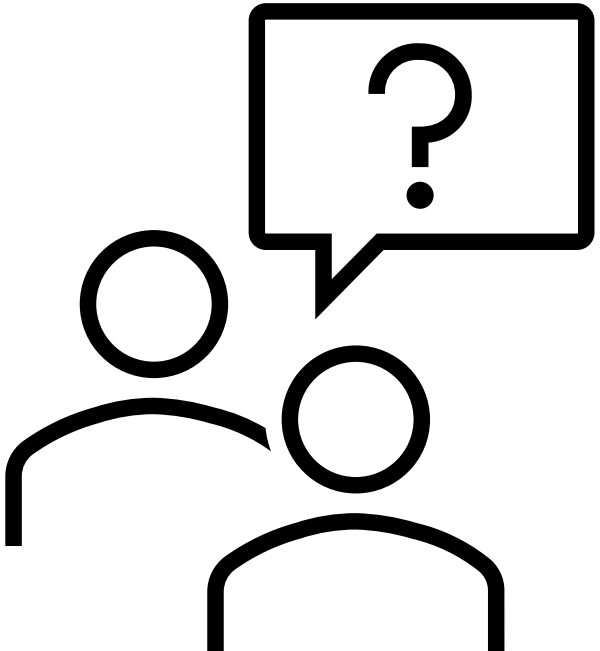
- **Question:** What is a simple algorithm that solves this problem in polynomial time?

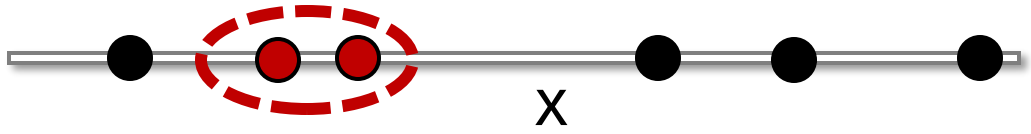




Closest Pair 1D

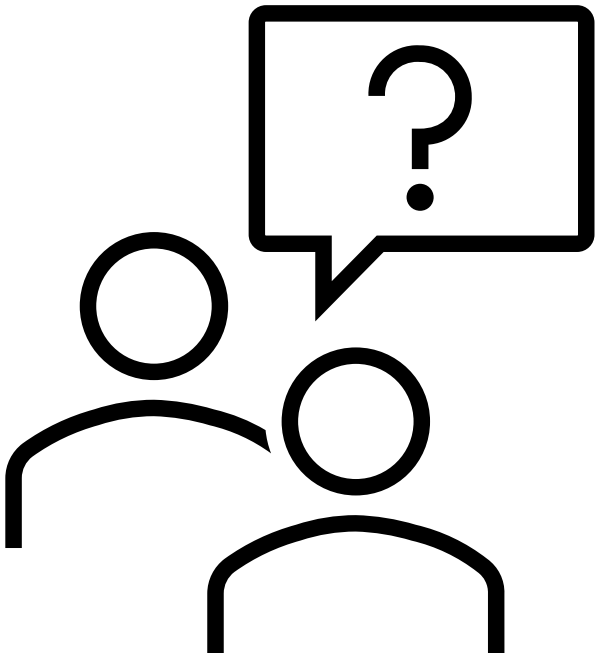
- **Answer:** Sort the list by x coordinate in O(nlog(n)) time and then loop over to find shortest distance in O(n) time.

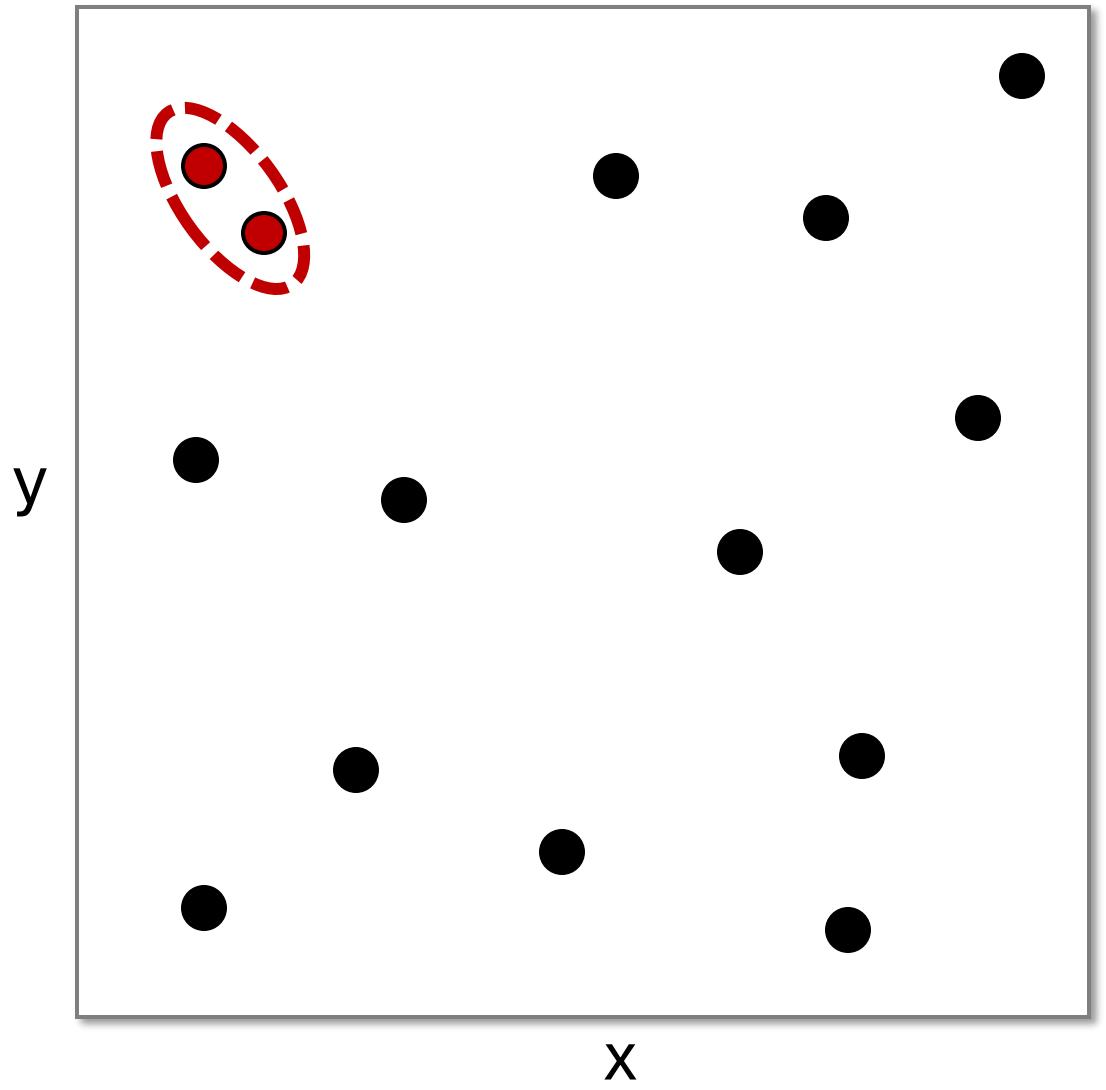




Closest Pair 2D

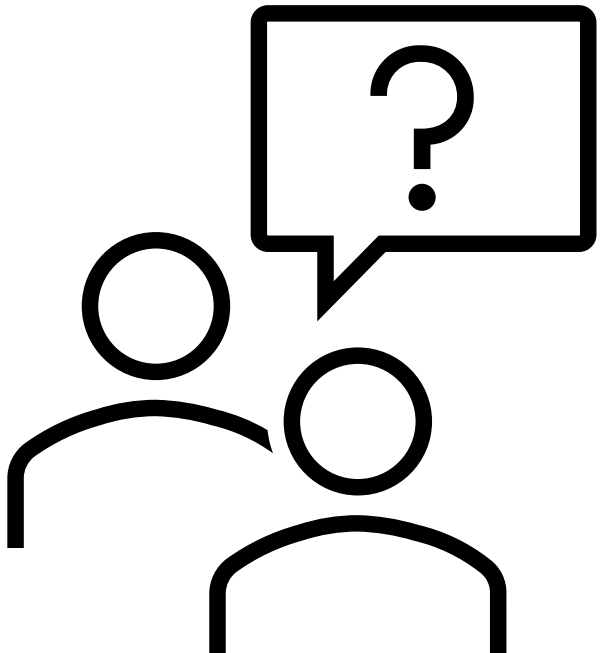
- **Question:** Can we use sorting to help with the 2D version of this problem?

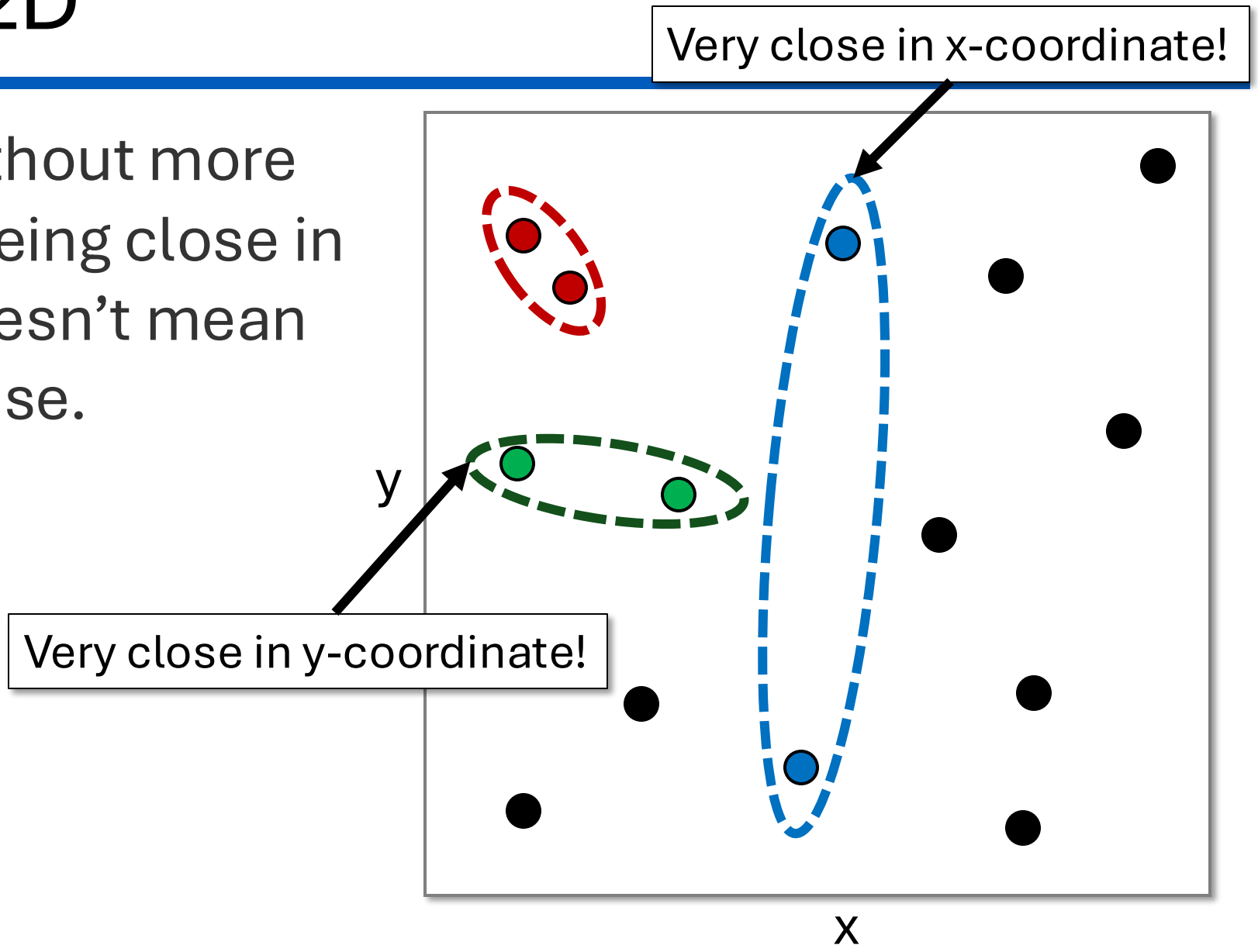




Closest Pair 2D

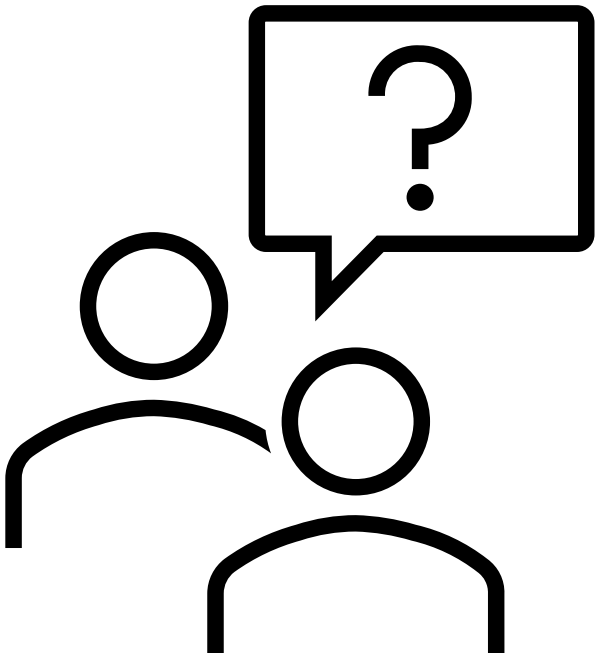
- **Answer:** Not without more work because being close in x-coordinate doesn't mean they are very close.

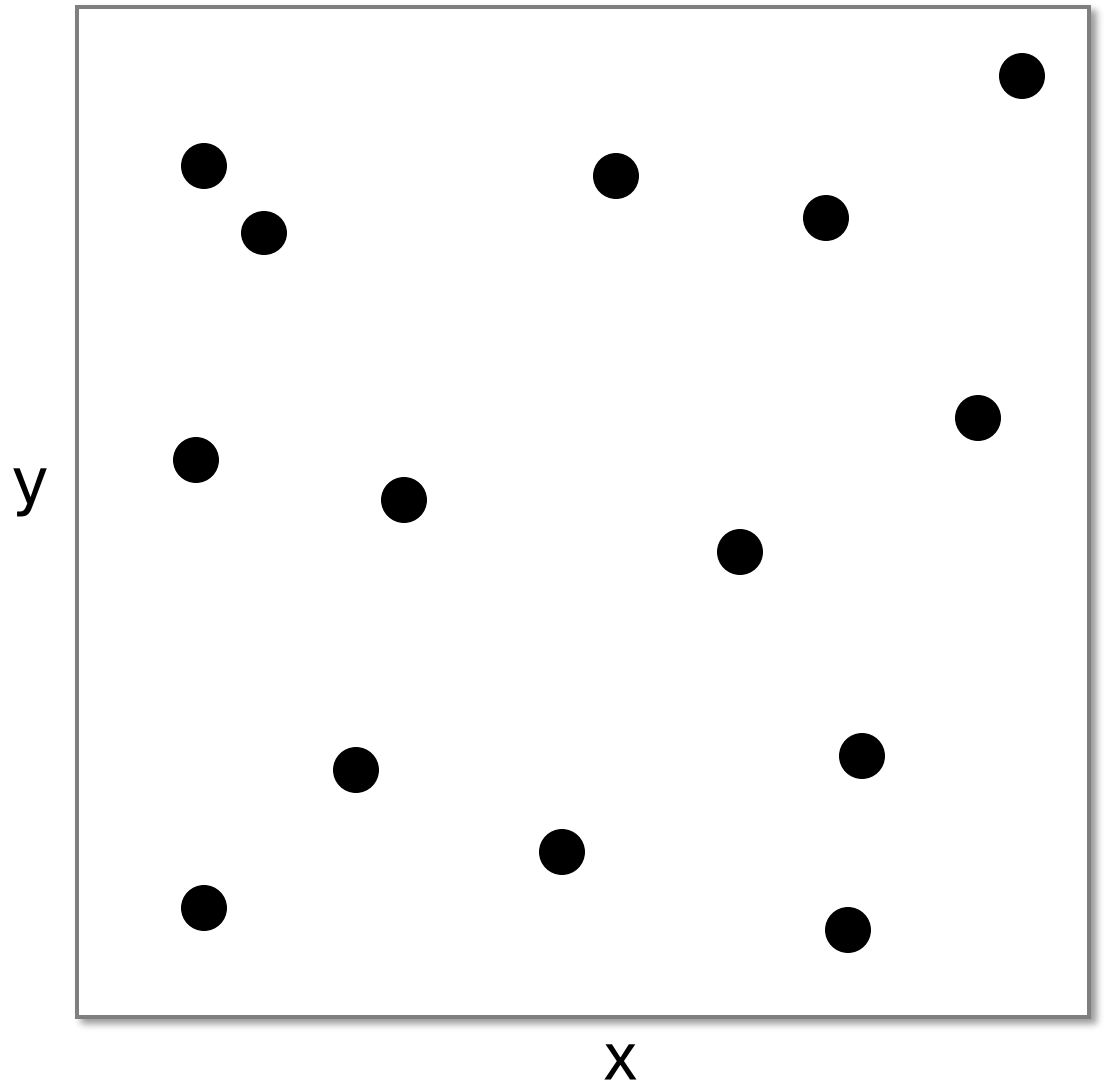




Divide & Conquer Closest Pair

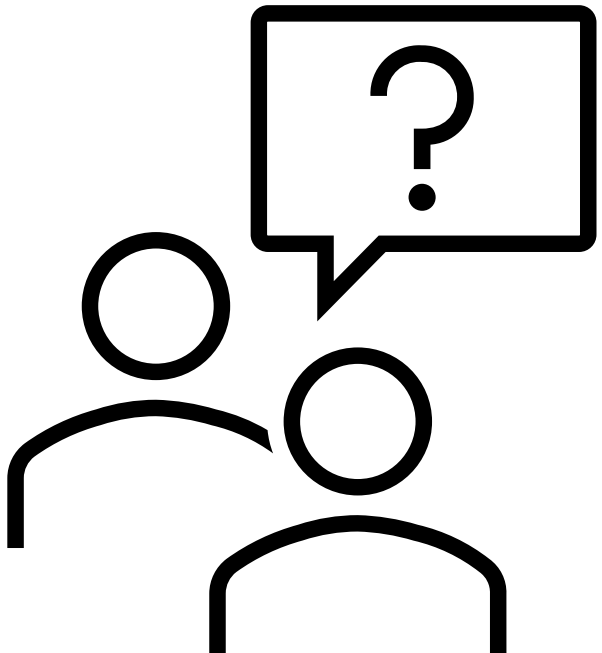
- **Question:** How can we do divide and conquer here?

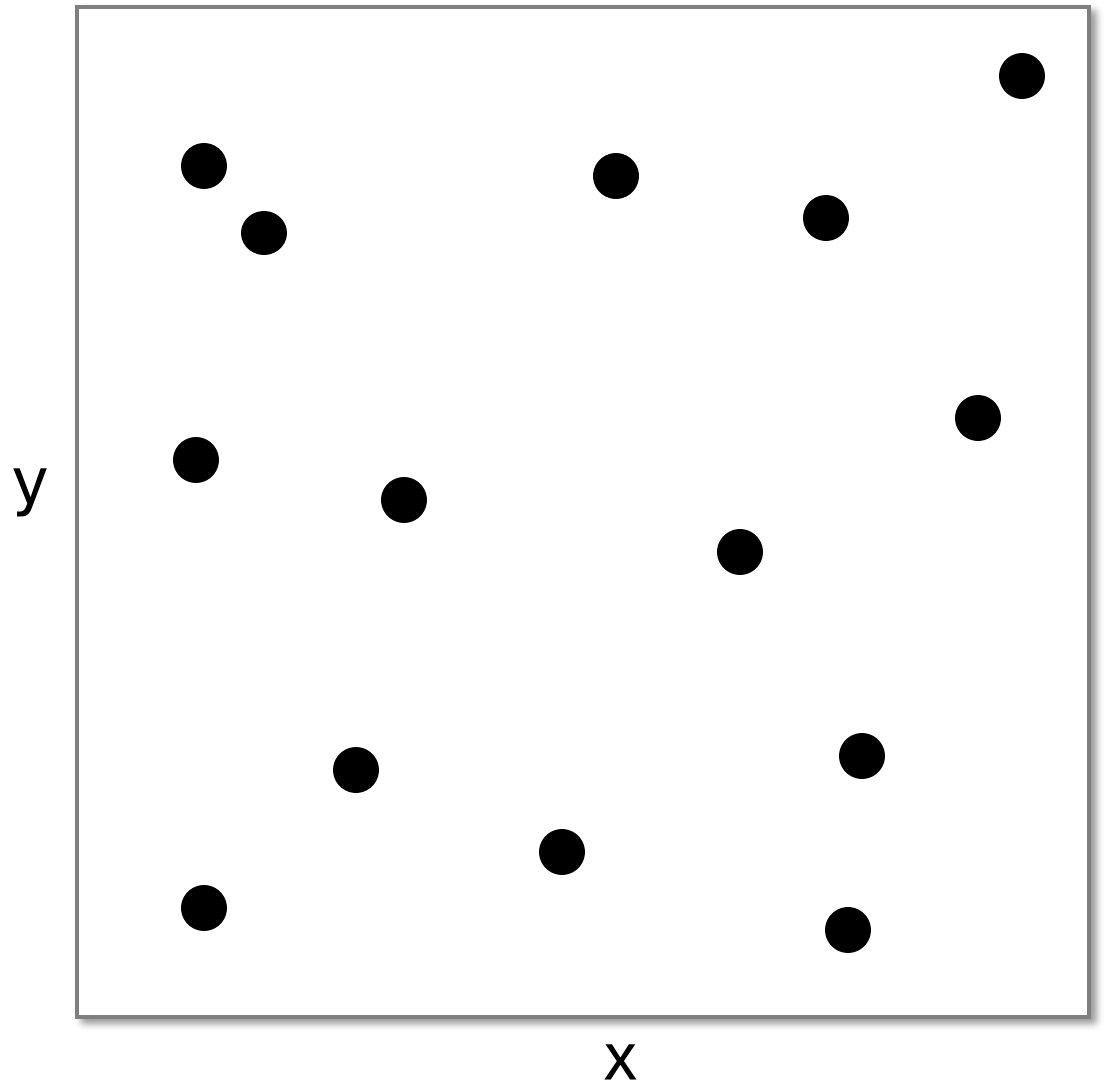




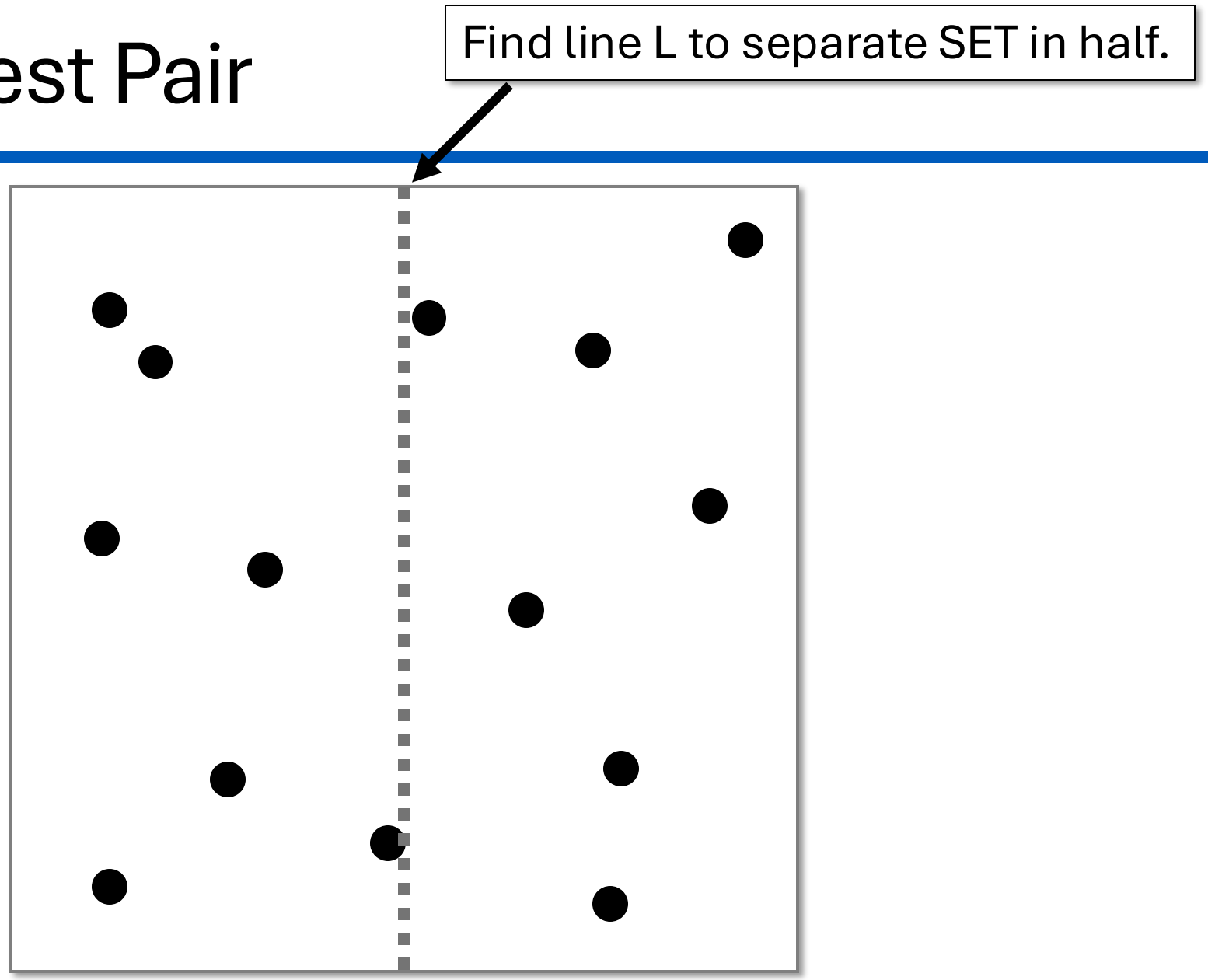
Divide & Conquer Closest Pair

- **Answer:** Break plane into pieces and recurse. Then merge the answers together.

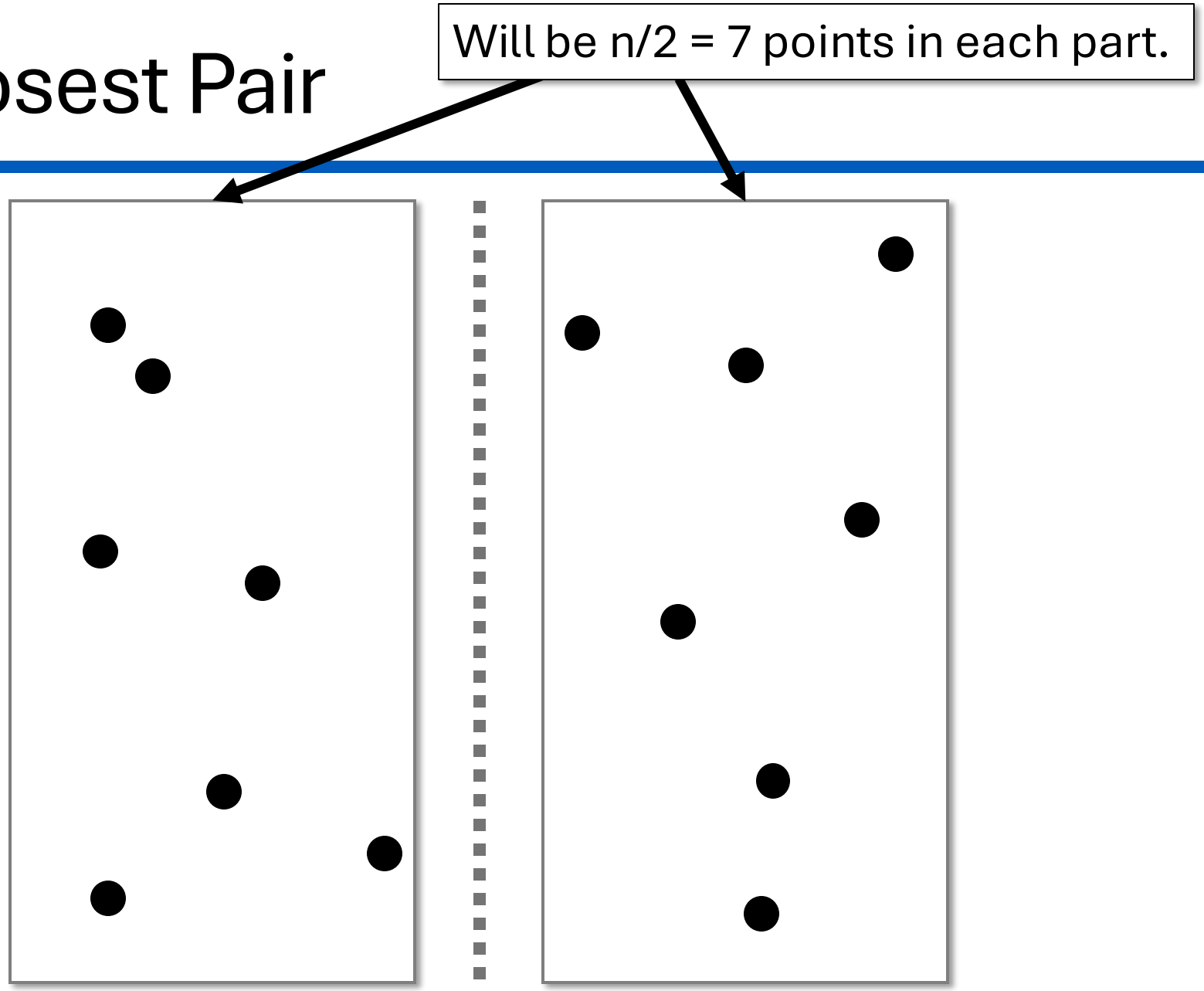




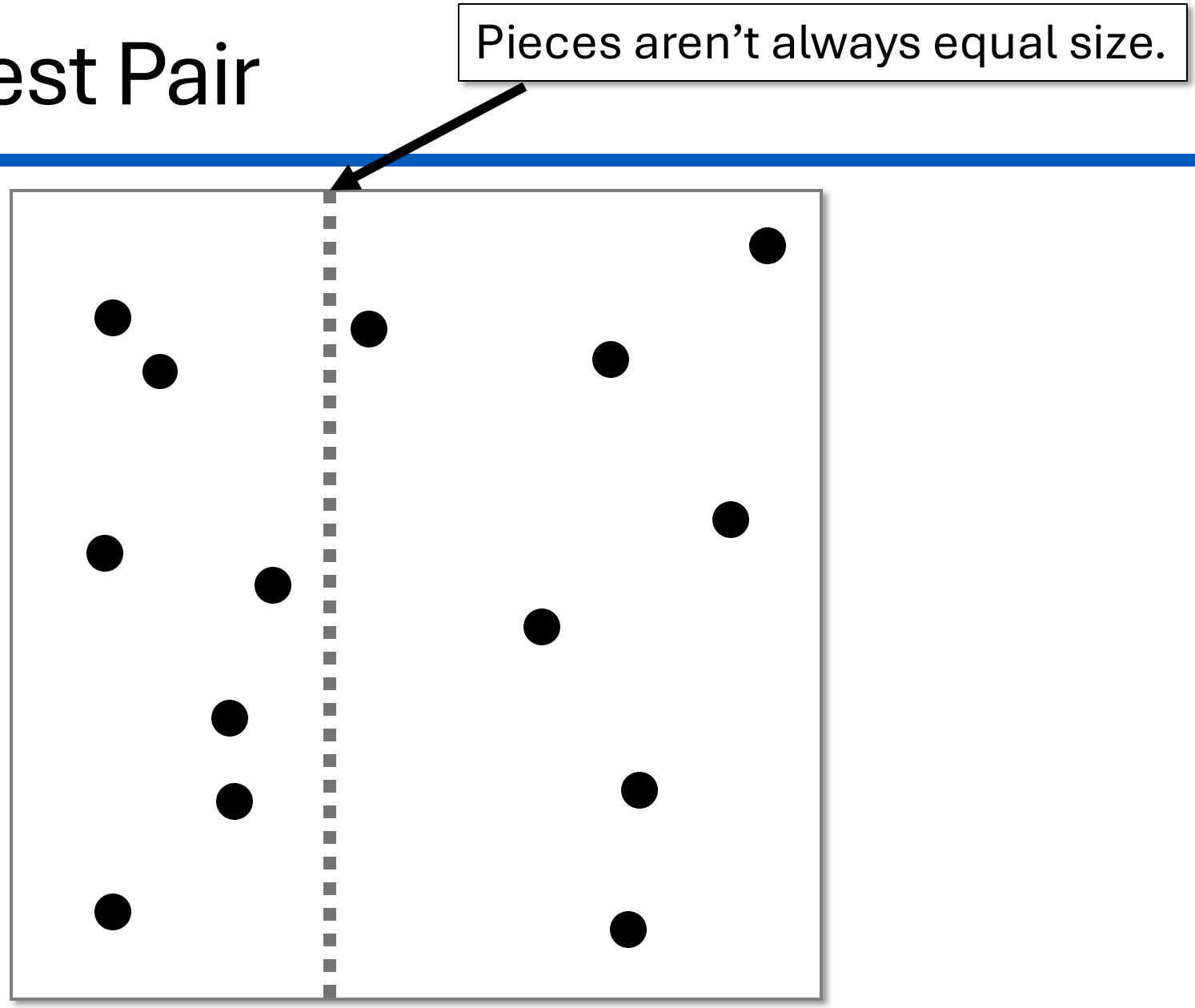
Divide Closest Pair



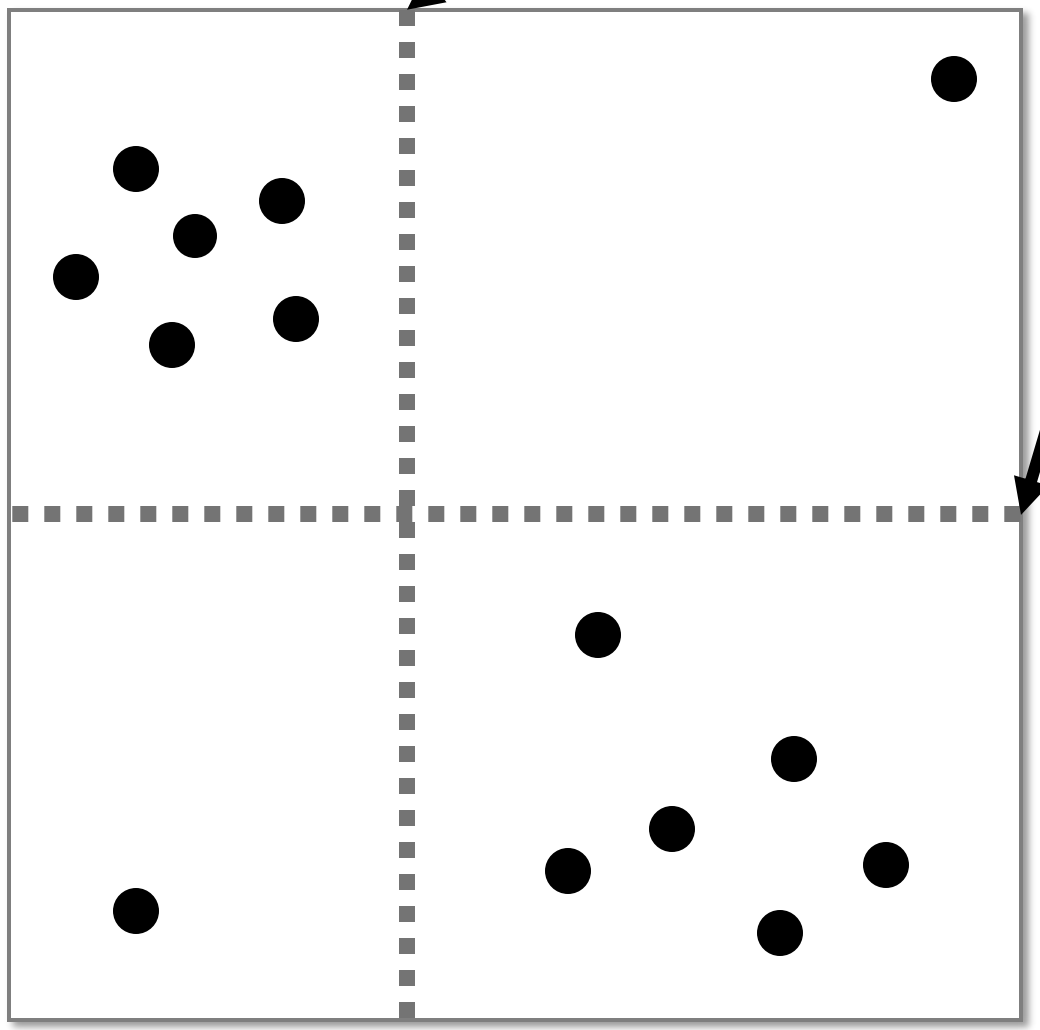
Divide Closest Pair



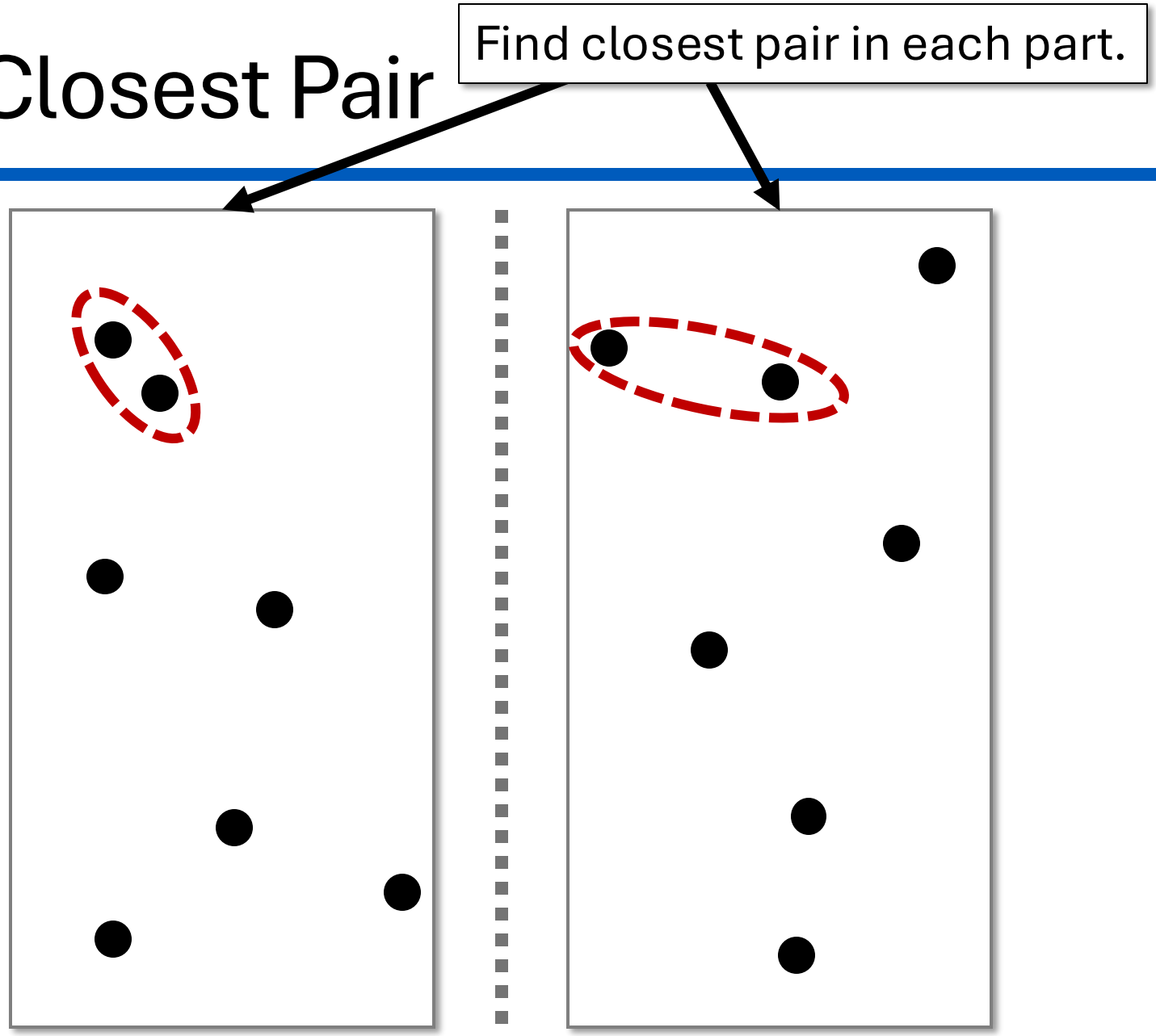
Divide Closest Pair



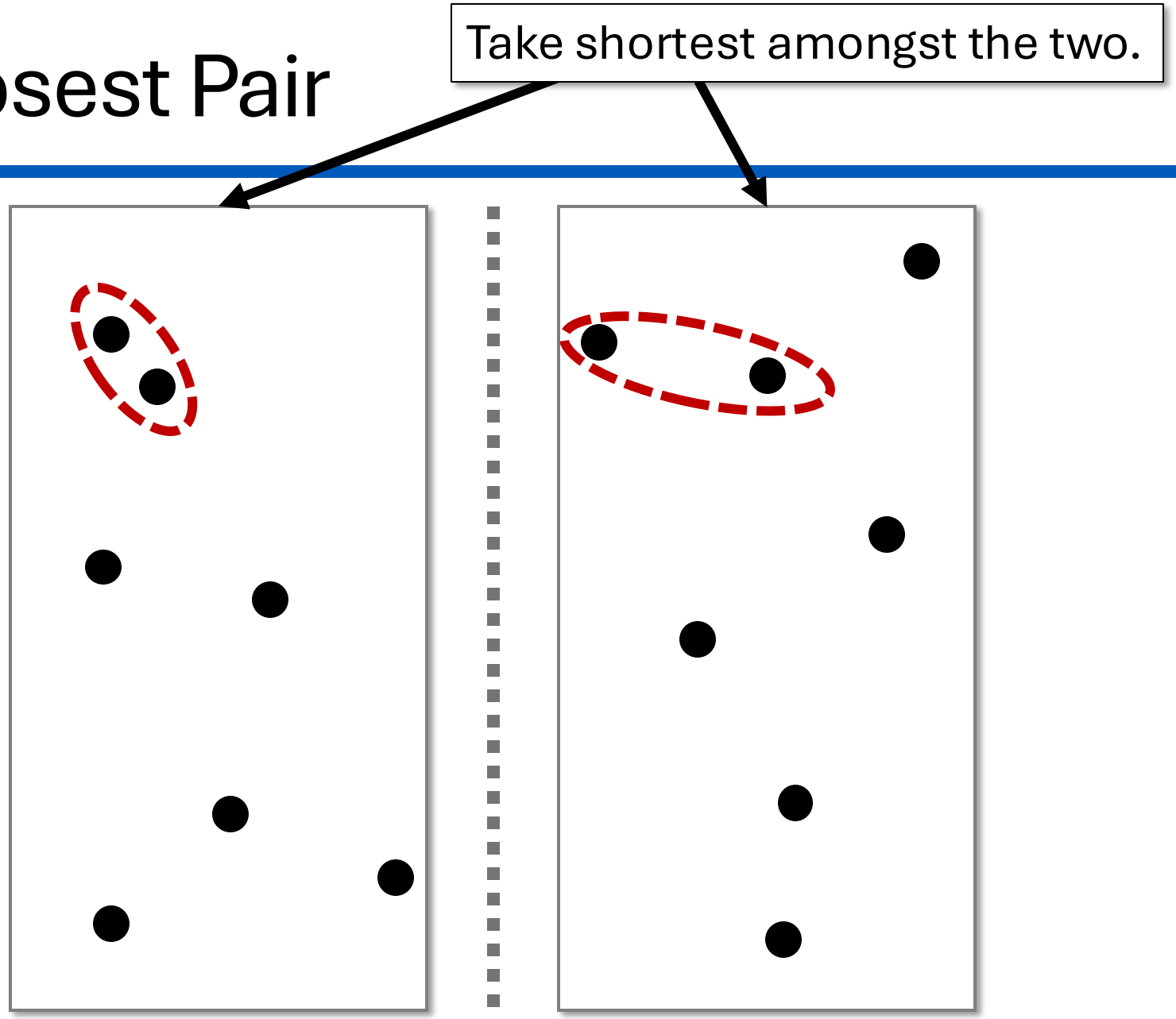
Divide Closest Pair



Conquer Closest Pair

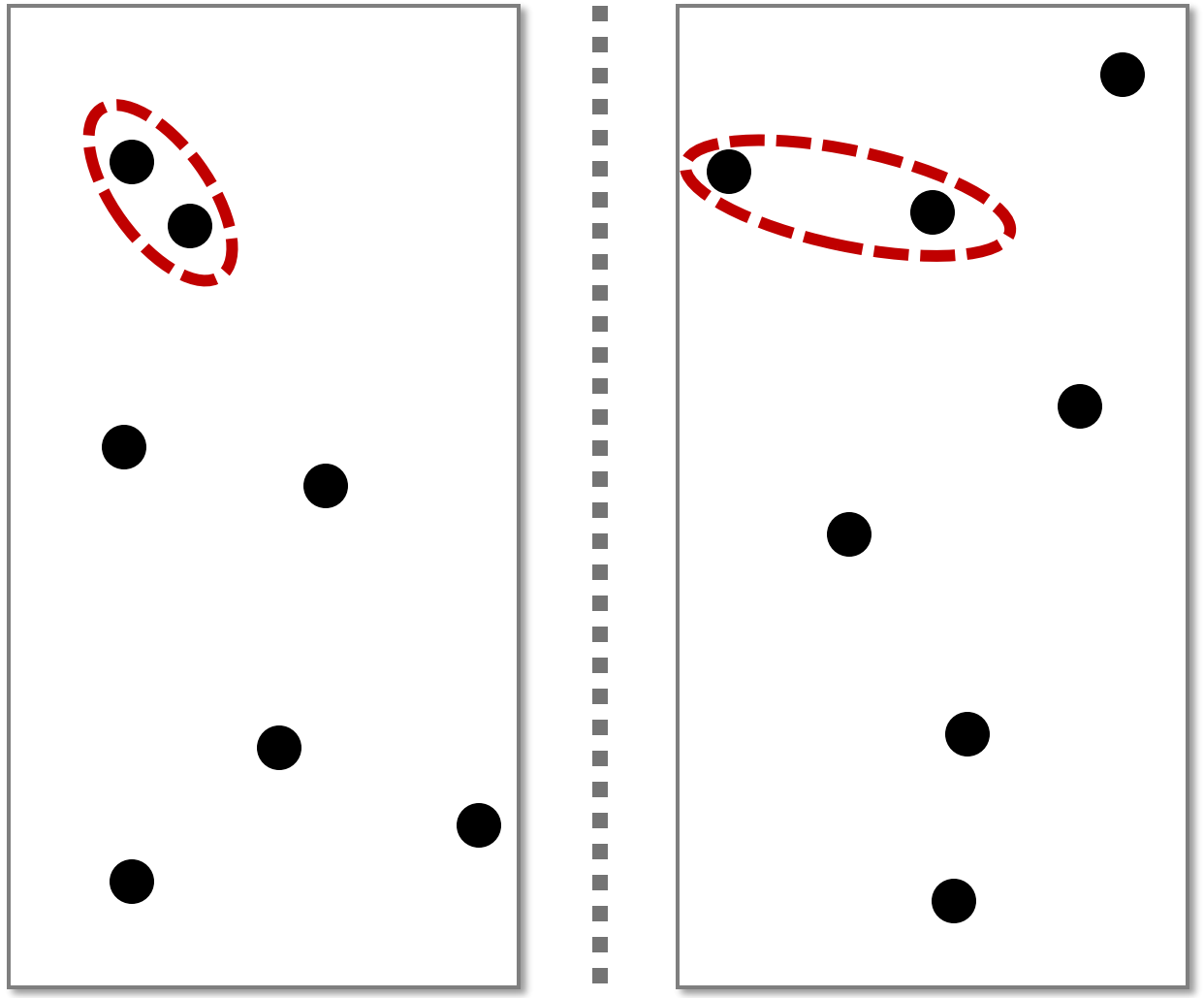


Merge Closest Pair



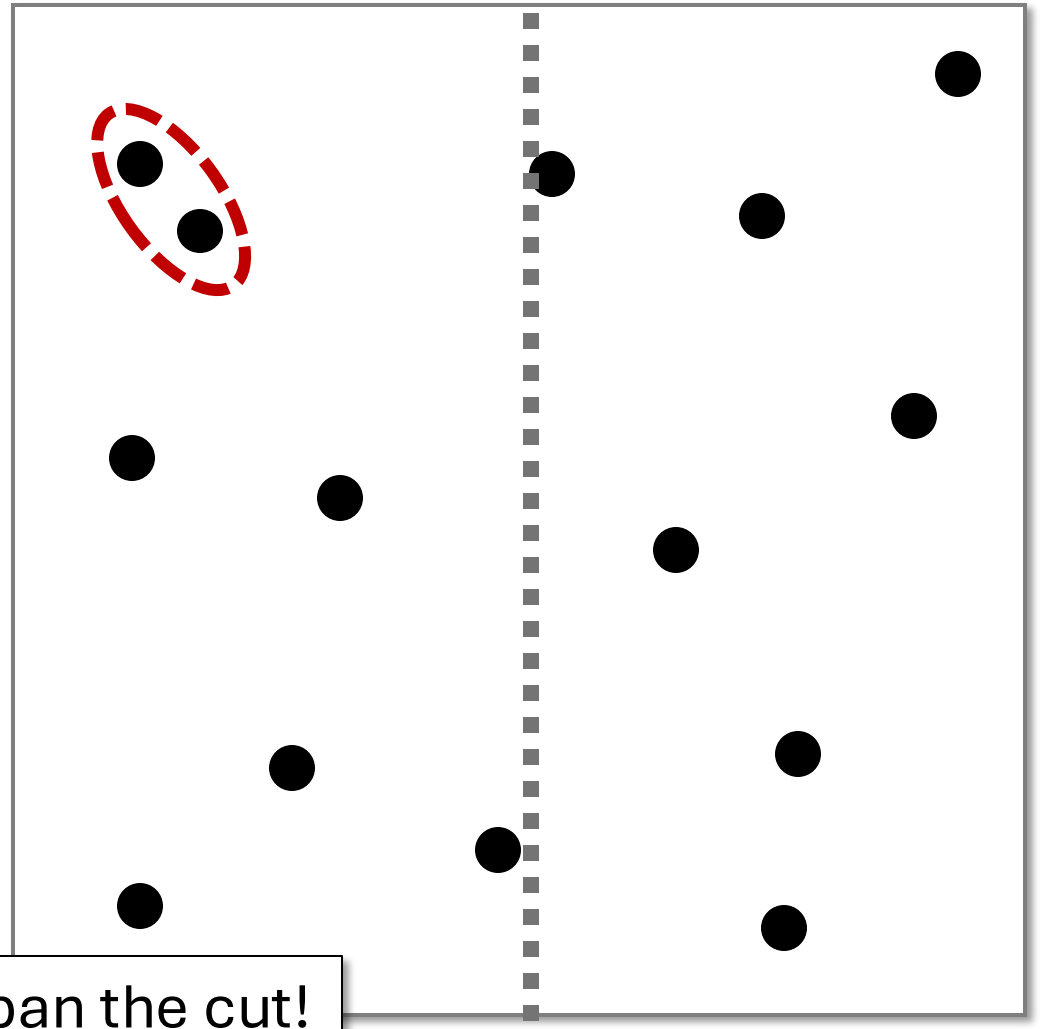
Closest Pair Algo Idea

- Break plane into two parts with equal number of points using line L.
- Recursively find closest pair in each part.
- Return the nearest of the two pairs.
- **Question**: Does this work?



Closest Pair Algo Idea

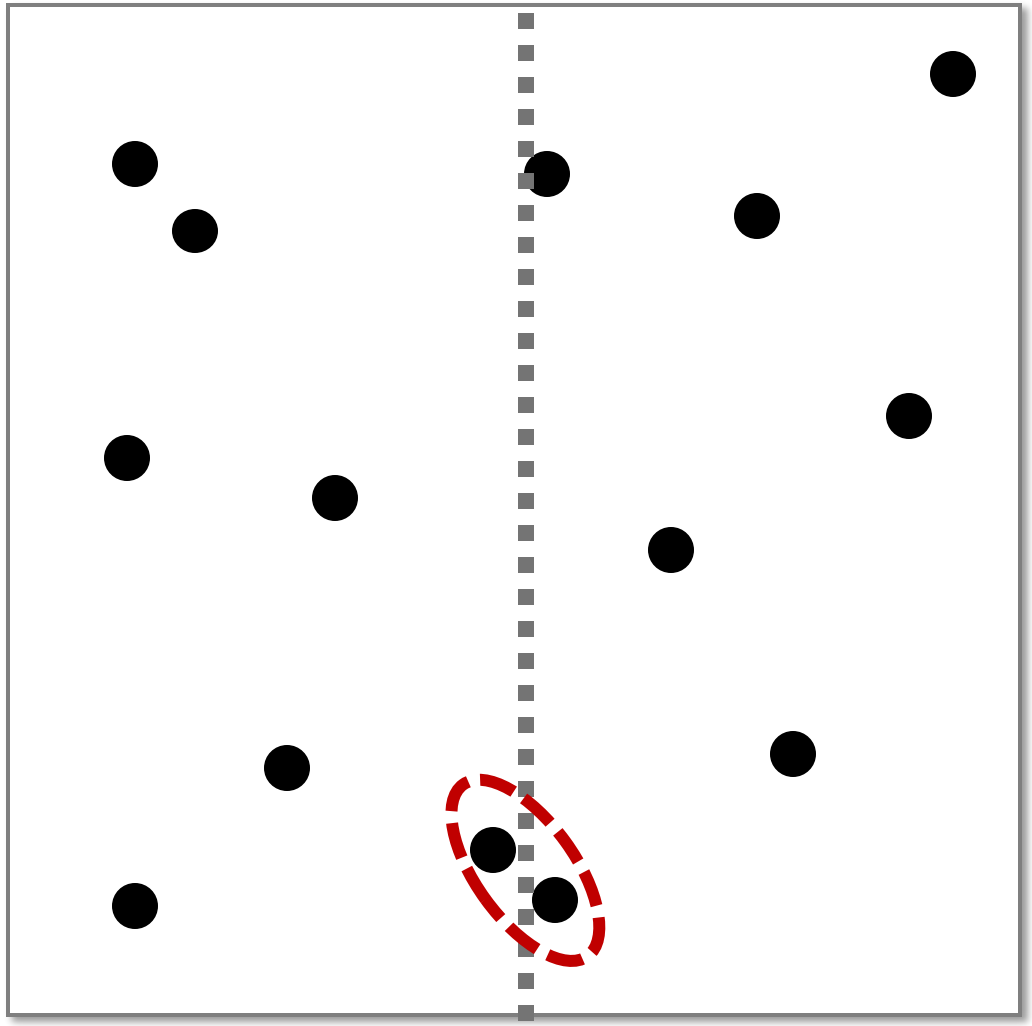
- Break plane into two parts with equal number of points using line L.
- Recursively find closest pair in each part.
- Return the nearest of the two pairs.
- **Question**: Not always!



The closest pair may span the cut!

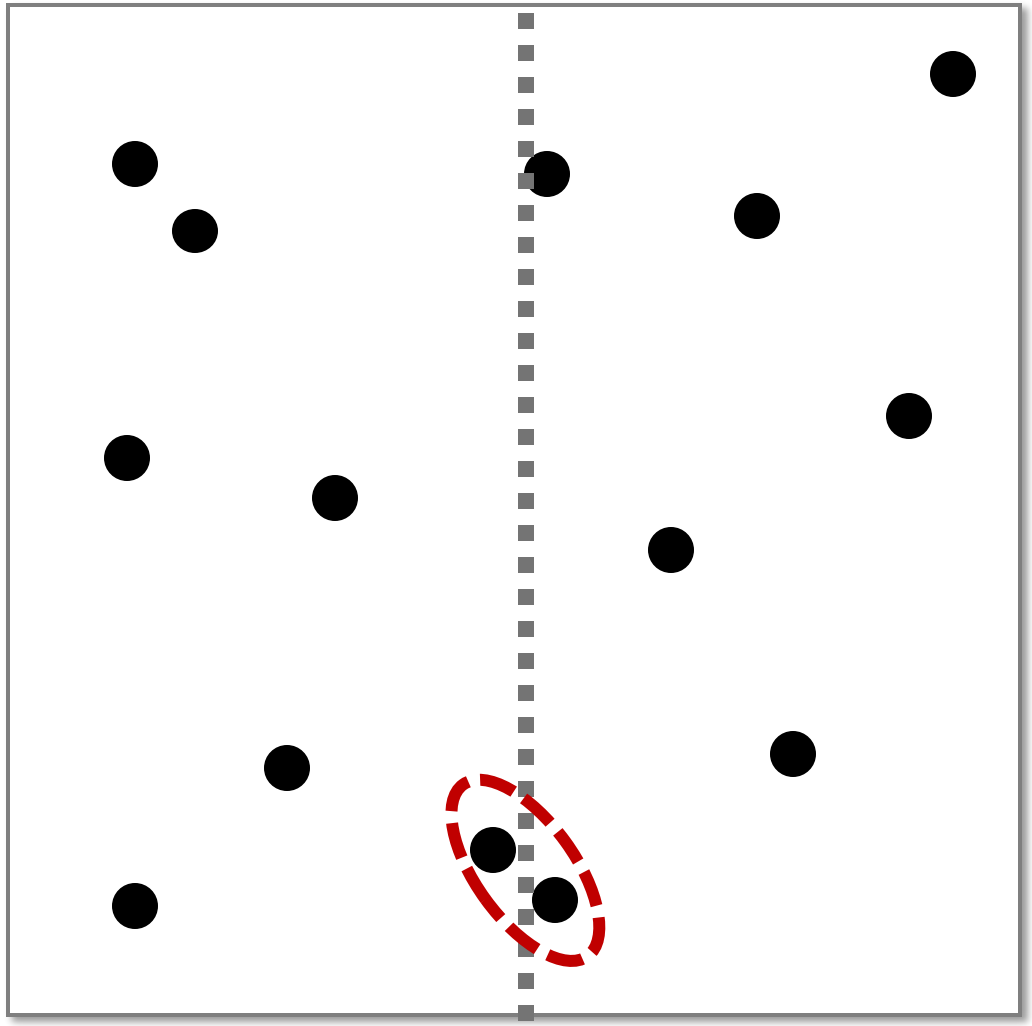
Closest Pair Algo Idea

- So, our merge step idea doesn't work.
  - We need to consider pairs that span the line L.
- **Question**: What is a naïve way of considering those vertices?



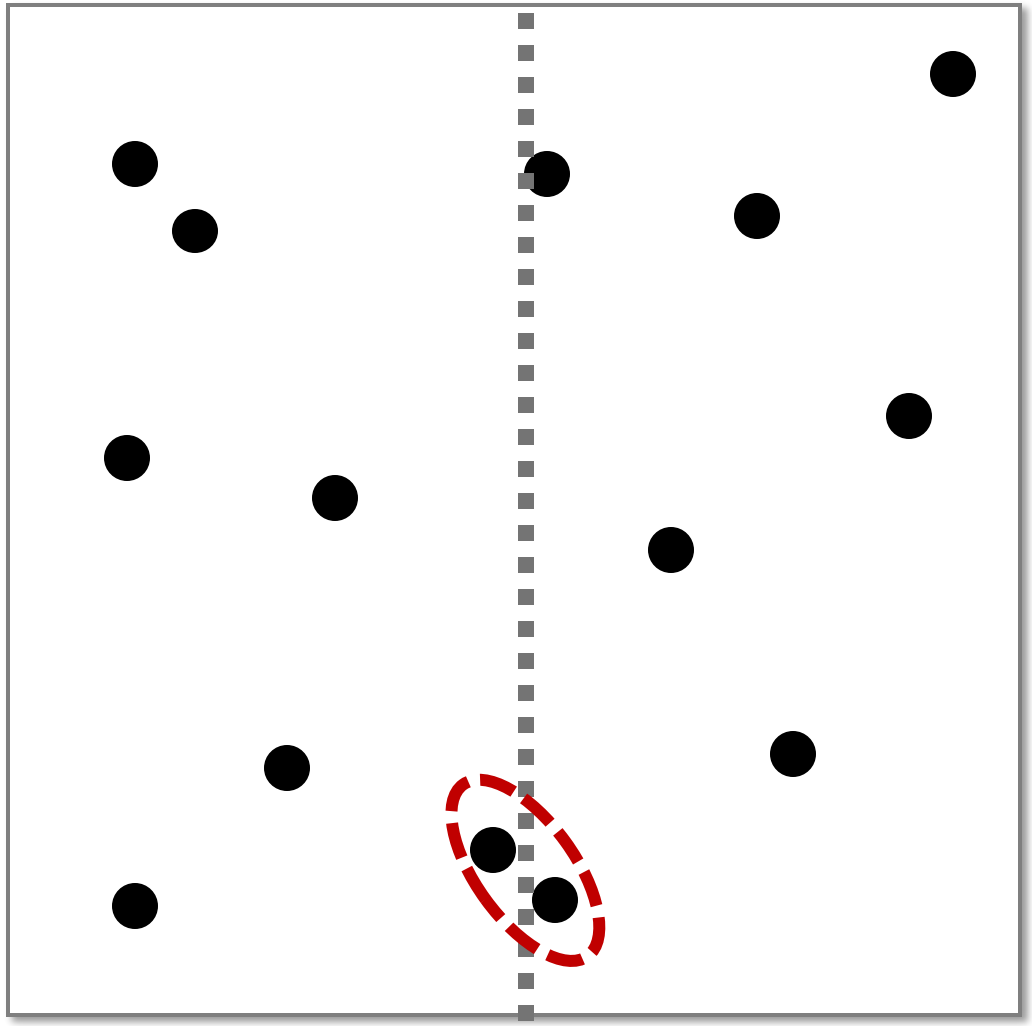
Closest Pair Algo Idea

- So, our merge step idea doesn't work.
  - We need to consider pairs that span the line L.
- **Answer**: We could just check every pair but that is O(n^2) again and we don't like that.

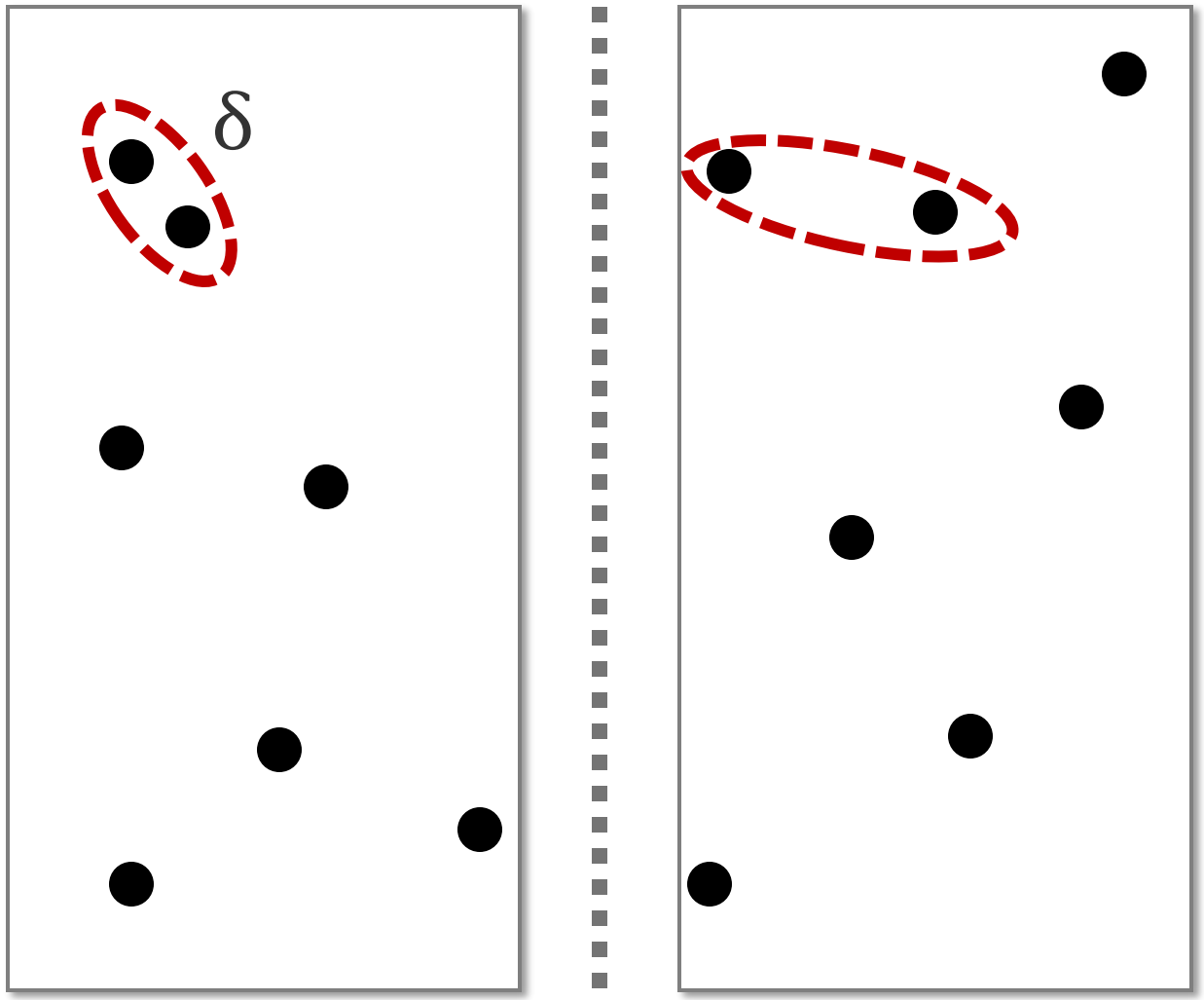


Closest Pair Algo Idea

- So, our merge step idea doesn't work.
  - We need to consider pairs that span the line L.
- **Answer**: We could just check every pair but that is O(n^2) again and we don't like that.

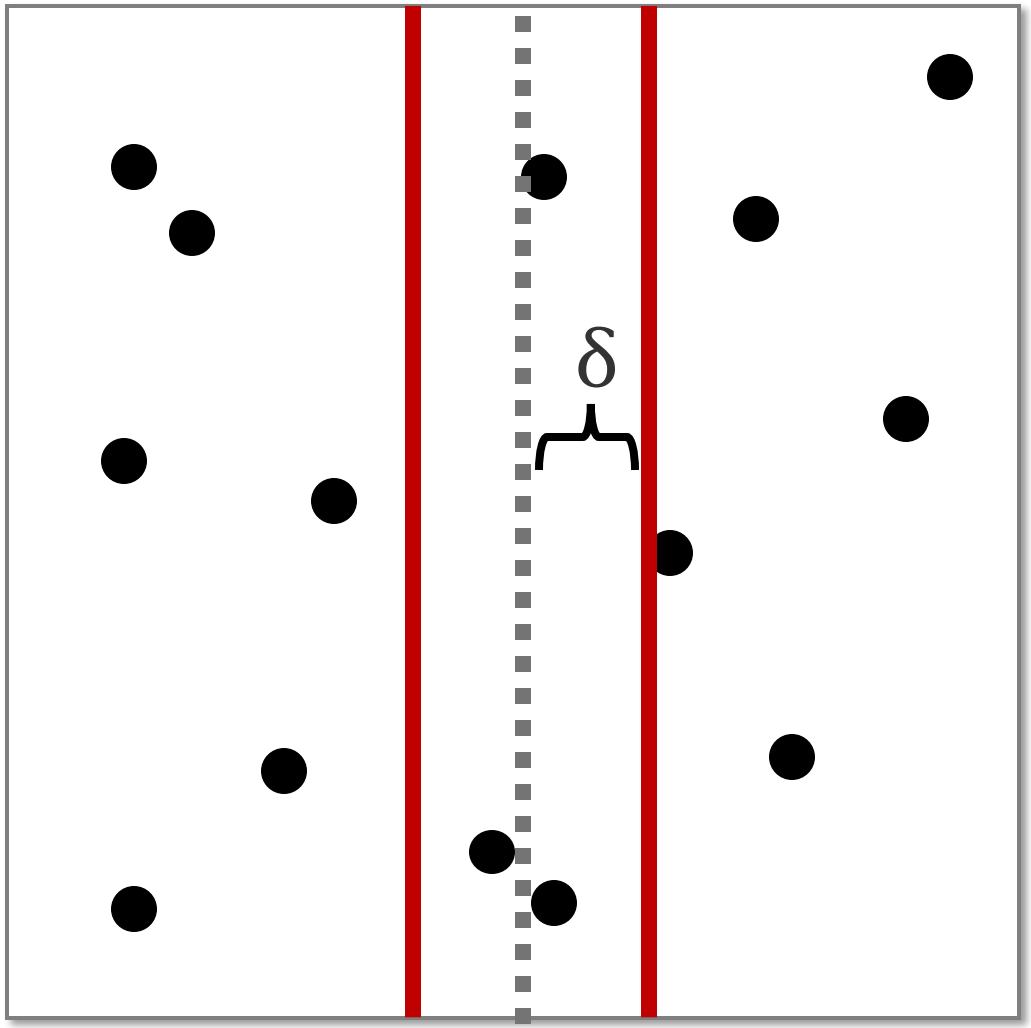


New Algo Idea

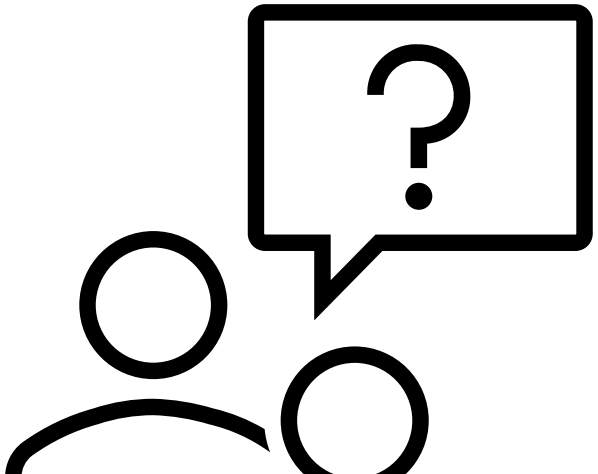


- Still divide into two smaller parts Q and R.
- Still find closest pairs in each part, q and r.
- Define δ to be the minimum distance between the two pairs.
- Now we need to find if any pair is closer.

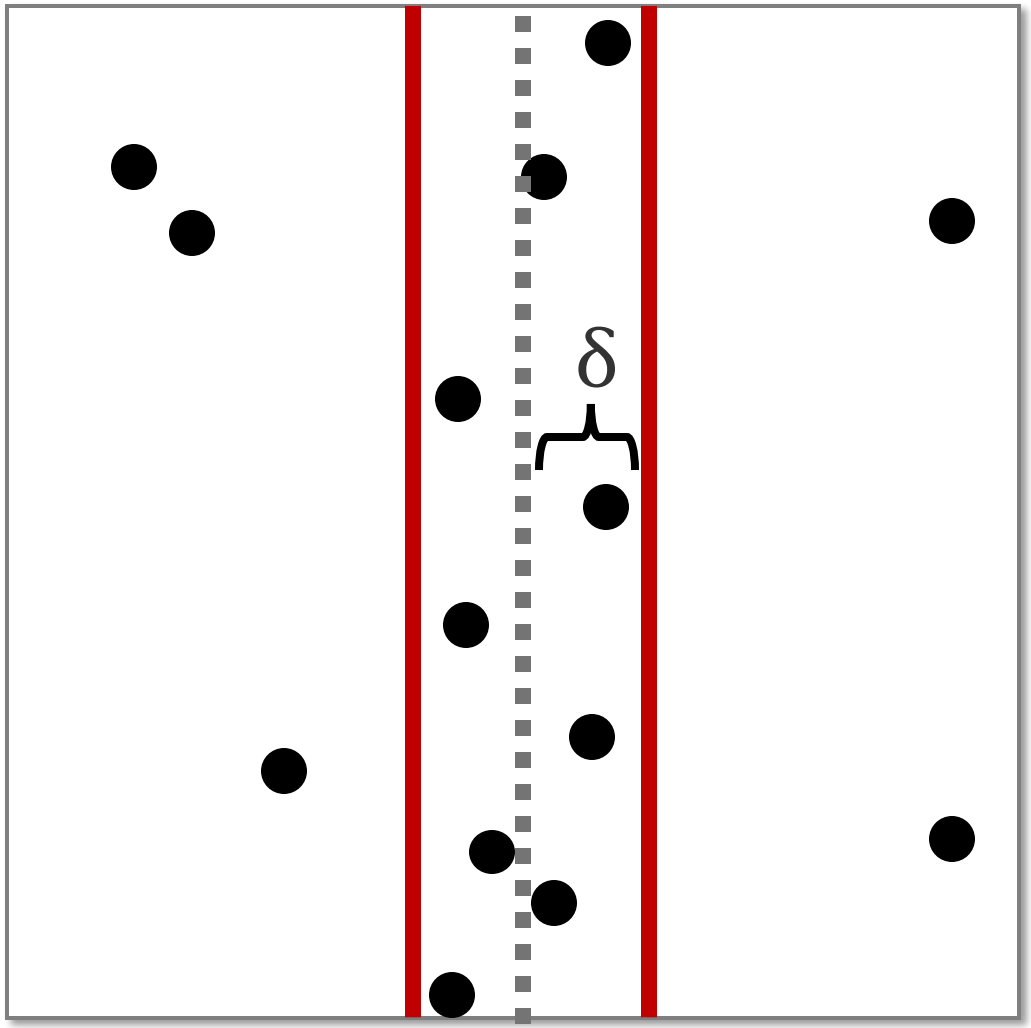
Finding Closer Pairs



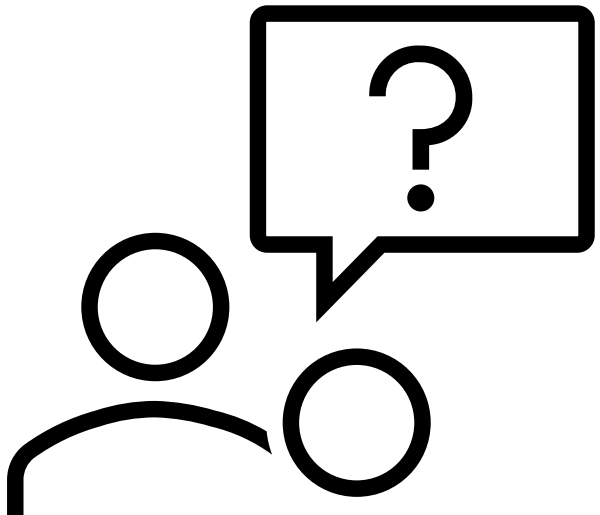
- Consider slice S of length 2δ around L.
- **Question**: How long does it take to check all pairs that are in the slice?



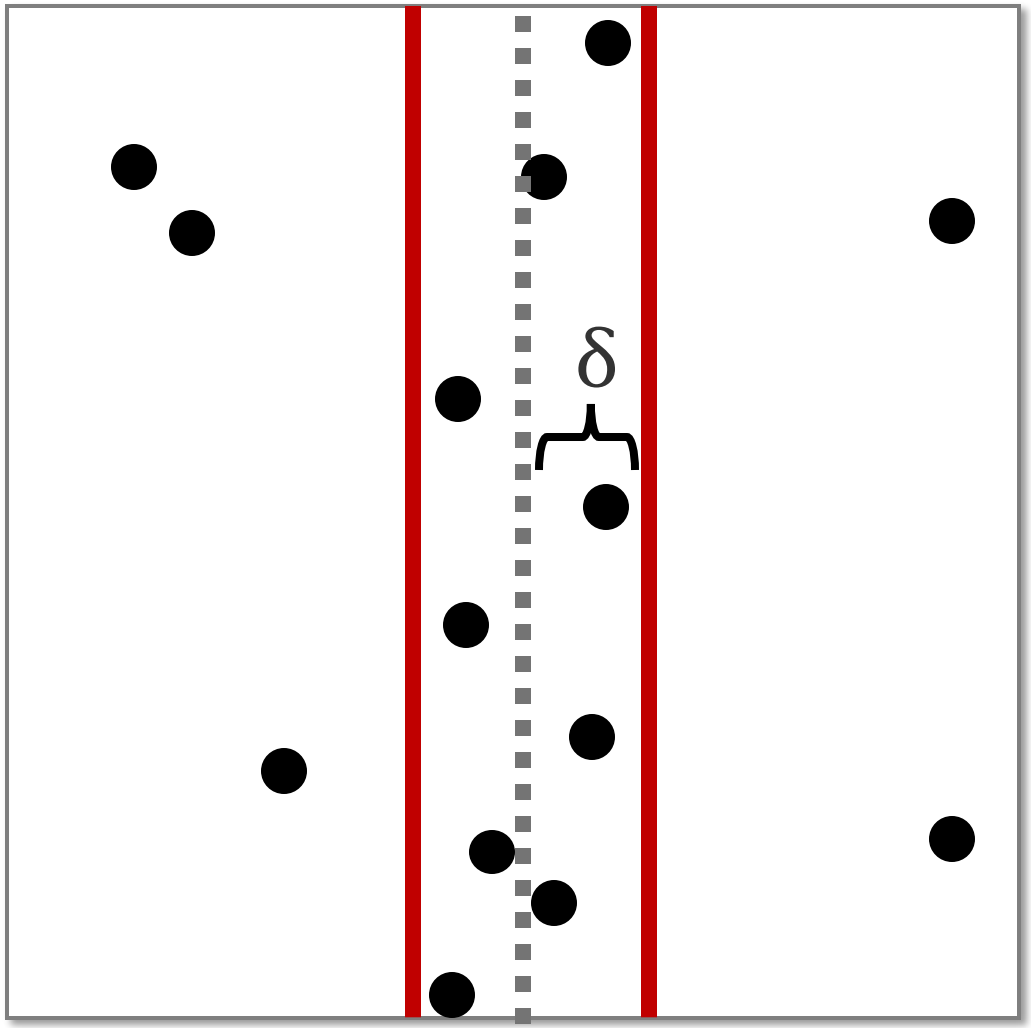
Finding Closer Pairs



- Consider slice S of length 2δ around L.
- **Answer**: If you do it naively then it could still take O(n^2).



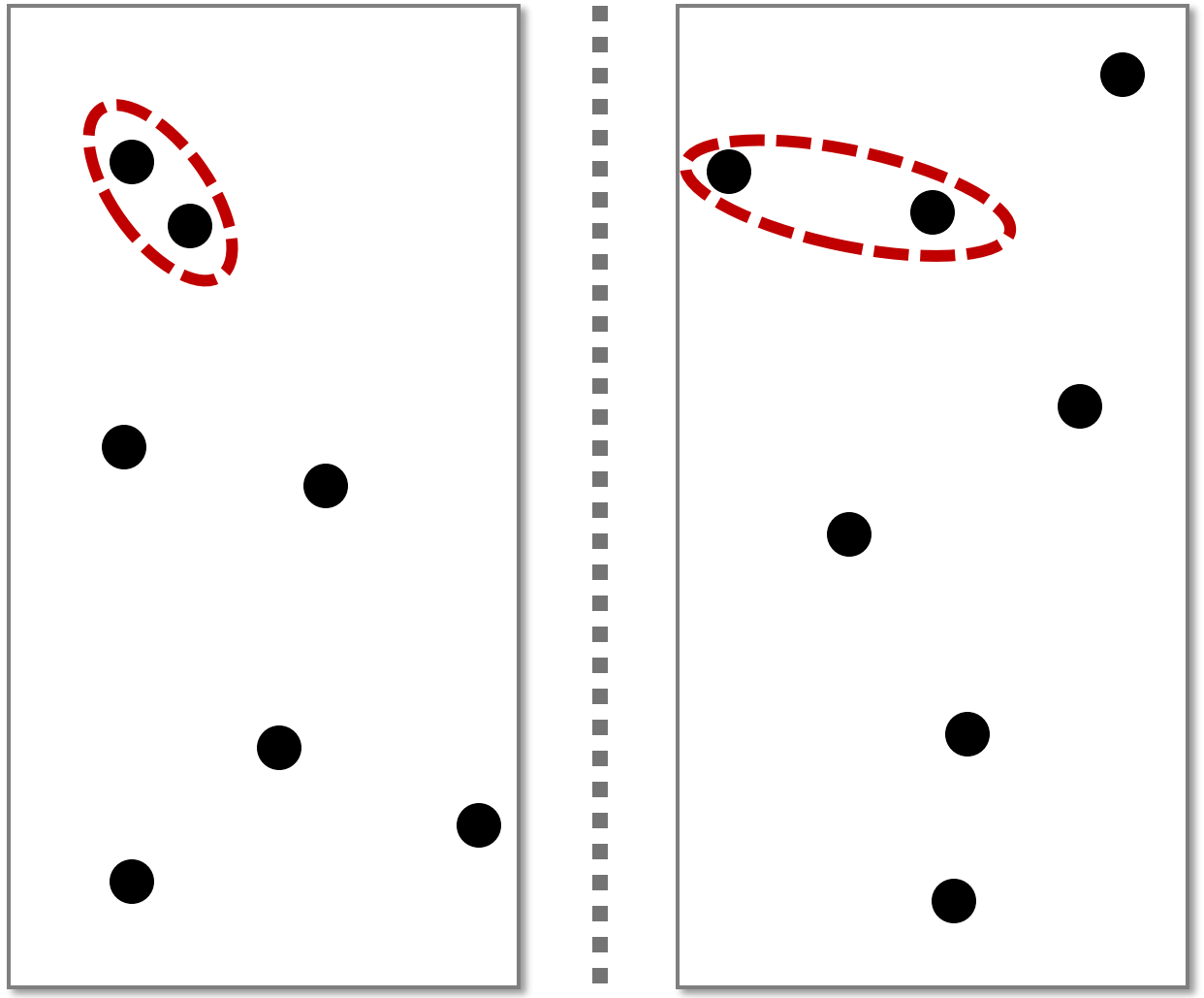
Finding Closer Pairs



- Let's assume that you had an algorithm that could check the slice in O(n) time.
- **Question:** Can you come up with a O(n log(n)) algorithm for Closest Pair 2D?

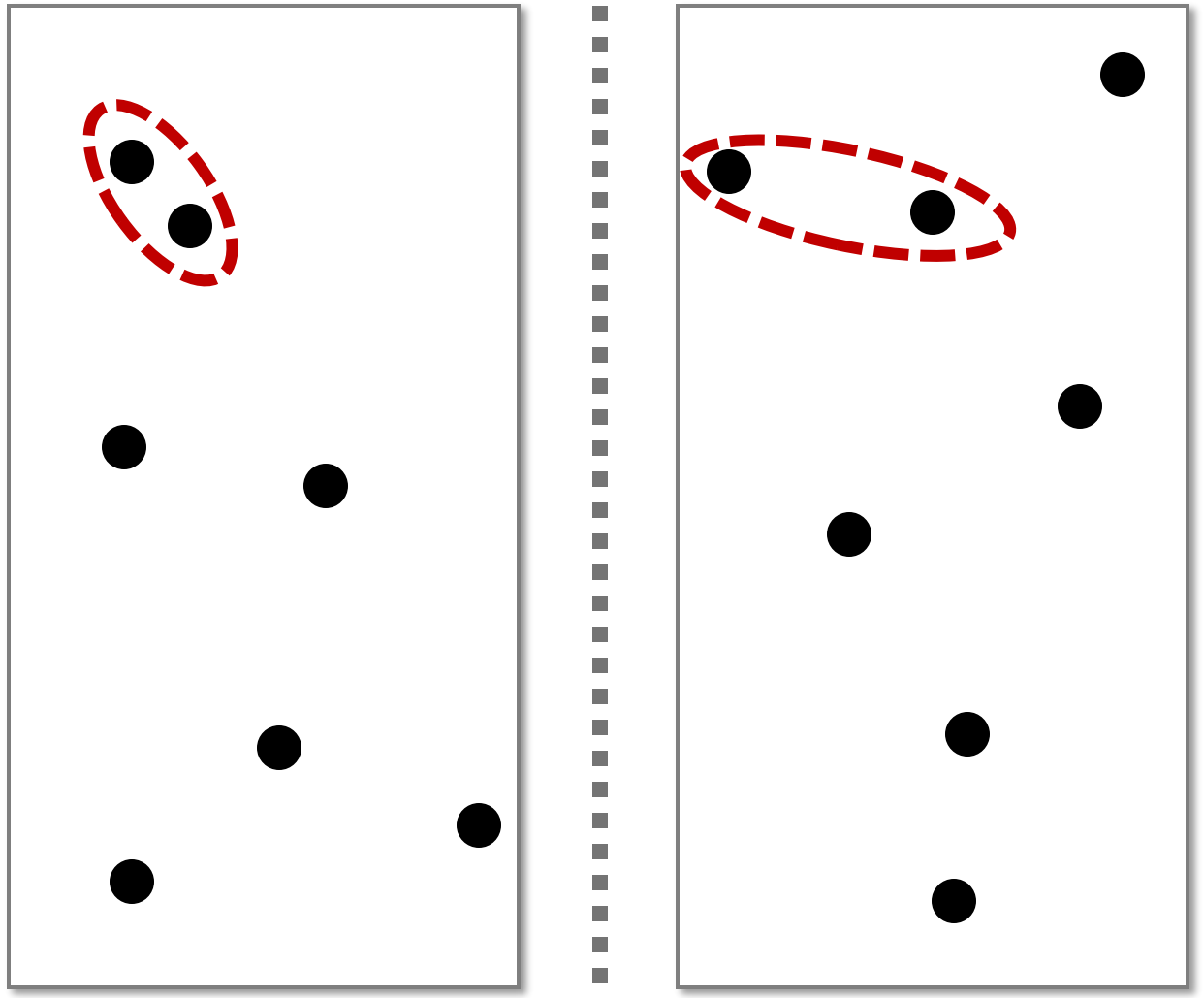
Closest Pair Algo Idea

- Break plane into two parts with equal number of points using line L.
  - Can be O(n) time if you sort by x-coordinate first.
  - Find n/2 item with smallest x-coordinate for one side…



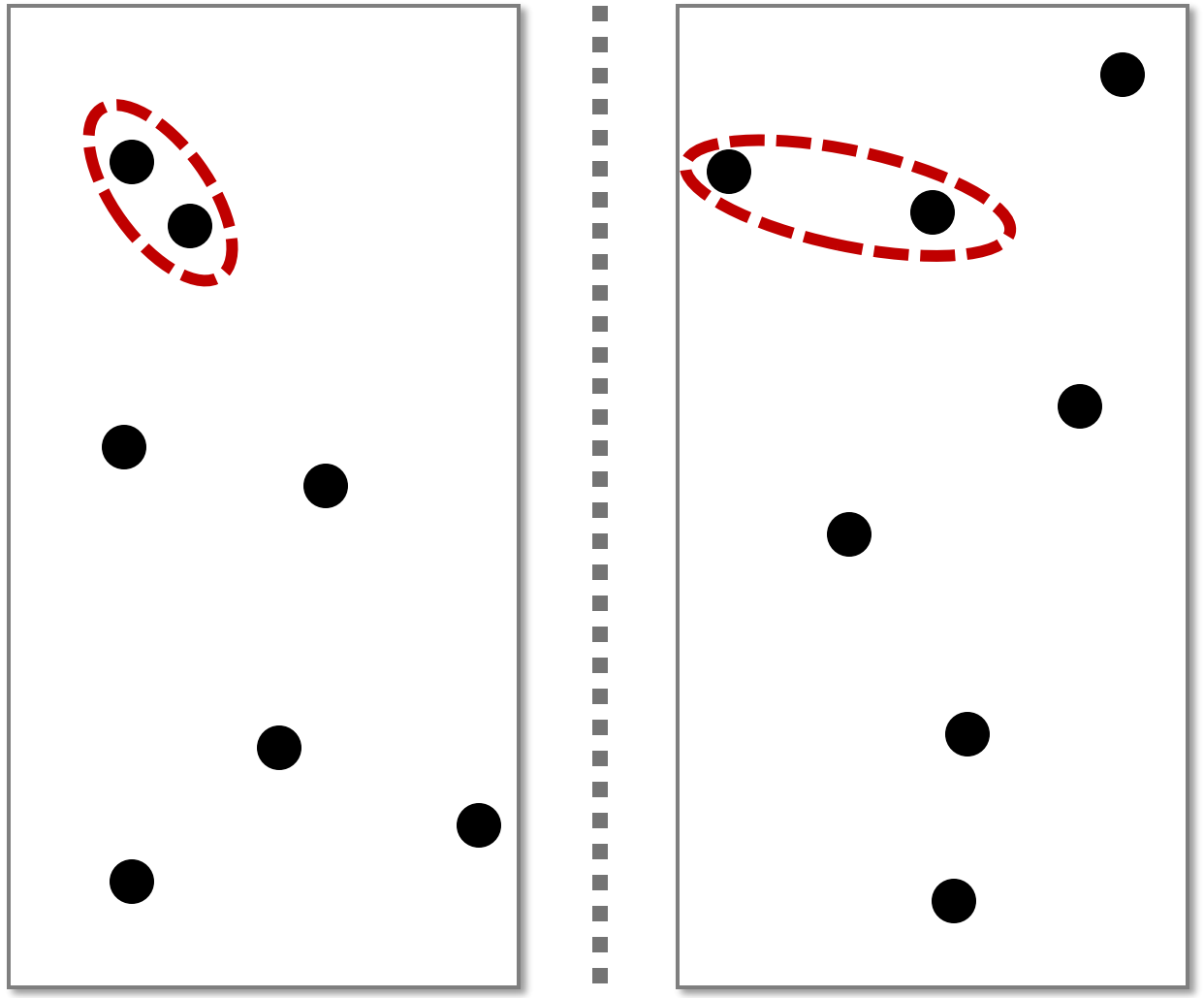
Closest Pair Algo Idea

- Recurse on each of the smaller planes to get closest pair.
  - This takes 2T(n/2) time if you assume n even.
- Use the min distance from these two closest pairs to define slice S.



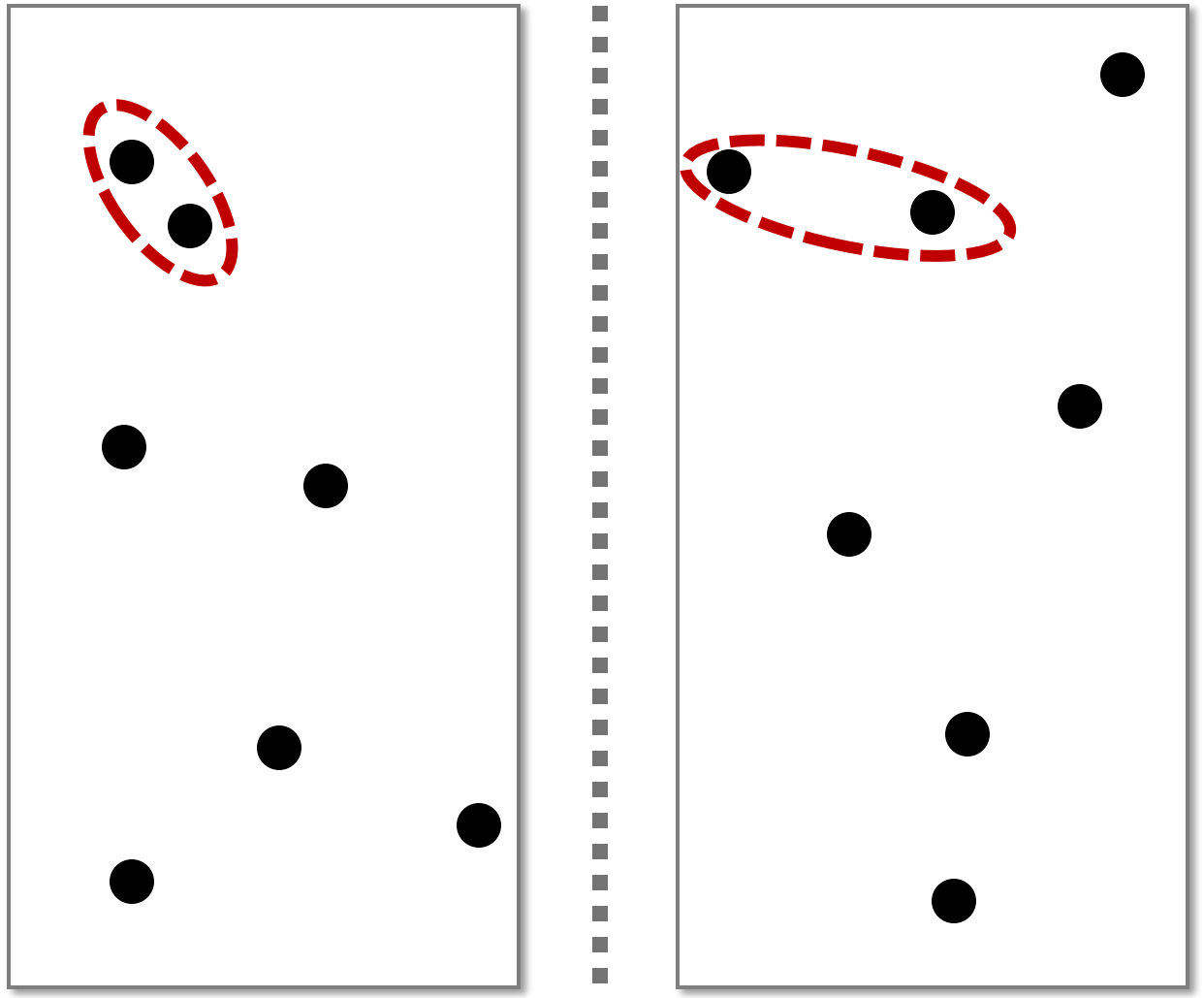
Closest Pair Algo Idea

- Assume you can find closest pair in slice in time O(n).
- If slice has closer point, return it. Otherwise return closest pair from amongst the two recursive calls.
- Don’t forget base case!



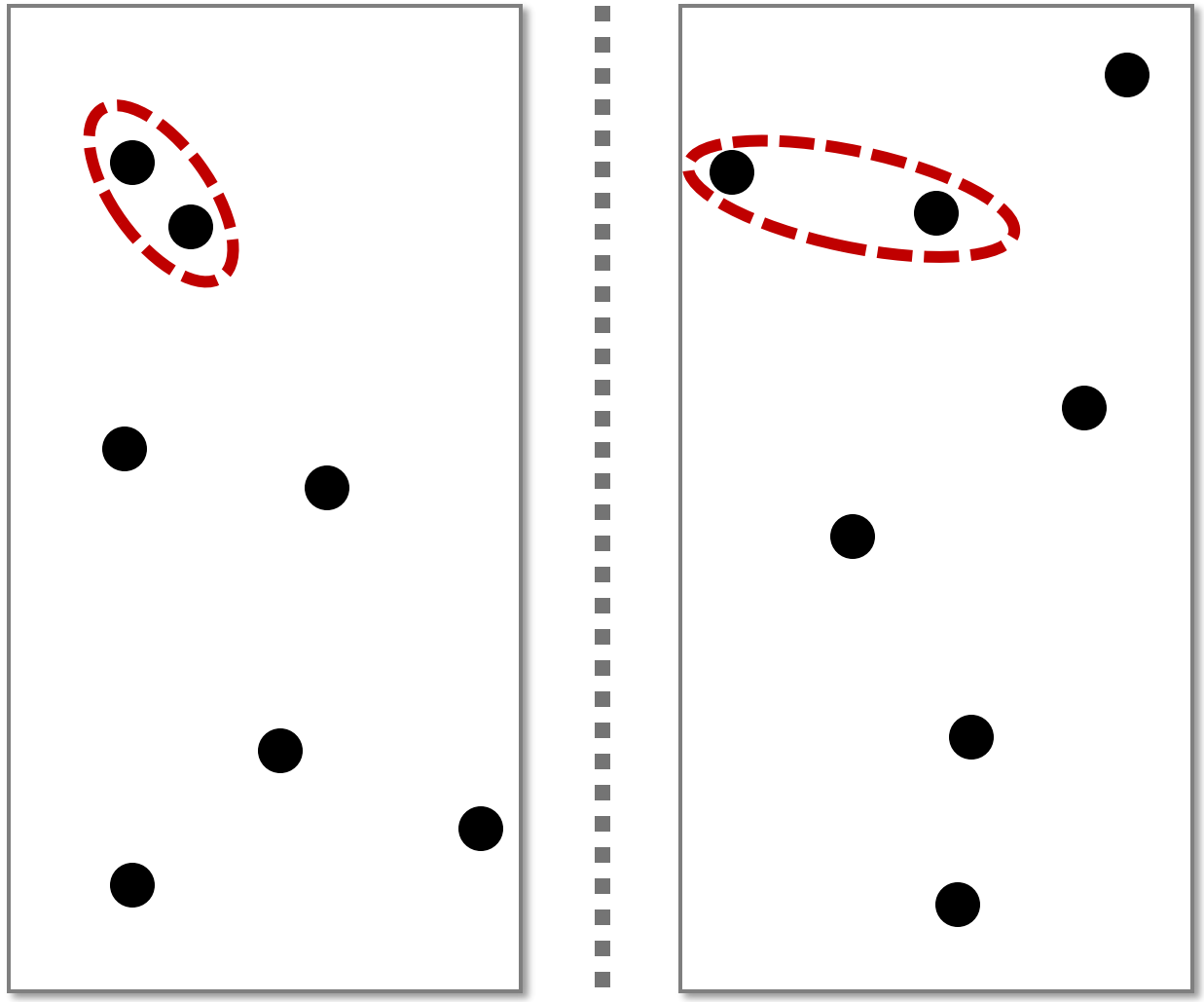
Closest Pair Algo Idea

- Observe that you only need to sort once.
- **Question**: What is the runtime of our algorithm if we assume that we can search the slice in O(n) time?



Closest Pair Algo Idea

- Observe that you only need to sort once.
- **Answer**: Since the base case can be handled in constant time, we have the same recurrence as mergesort and thus, the same running time.



Closest-Pair(P):

- Input: n 2D points $P = \{p_1, p_2, \ldots, p_n\}$, $p_i = (x_i, y_i)$
  - Sorted to get P_x and P_y

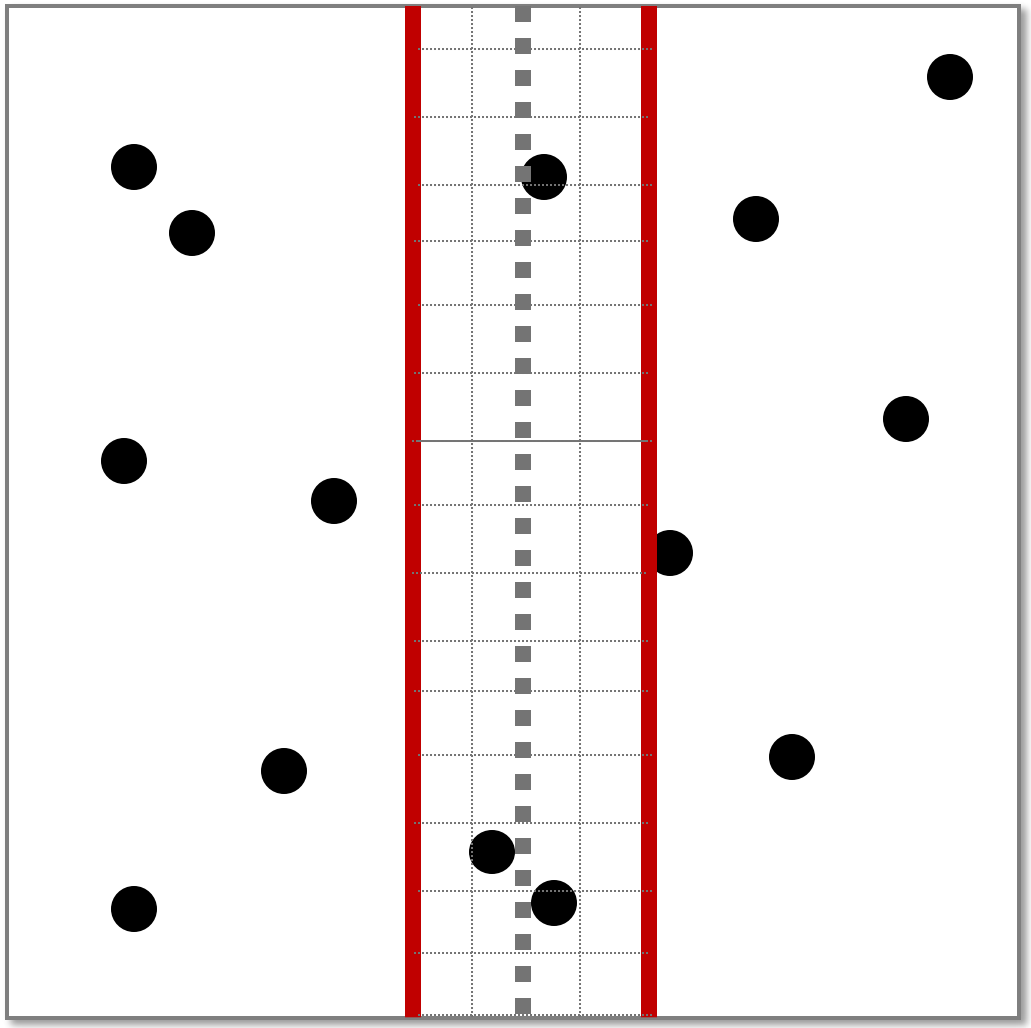
1. If $n \leq 4$, the do brute force.
2. Let Q be first half of P_x and let R be rest
3. Compute $Q_x, Q_y, R_x,$ and R_y using P_x and P_y
4. (q_x, q_y) = Closest-Pair(Q_x, Q_y)
5. (r_x, r_y) = Closest-Pair(R_x, R_y)
6. $\delta = \min((q_x, q_y), (r_x, r_y))$
7. Let S be points (x_i, y_i) in P s.t. $|x_i - x^*| \leq \delta$
8. (r_x, r_y) = Closets-Pair(S_x, S_y)
9. Return min$((q_x, q_y), (r_x, r_y), (r_x, r_y))$

Closest-Pair(P):

- Input: n 2D points $P = \{p_1, p_2, \ldots, p_n\}$, $p_i = (x_i, y_i)$
  - Sorted to get P_x and P_y

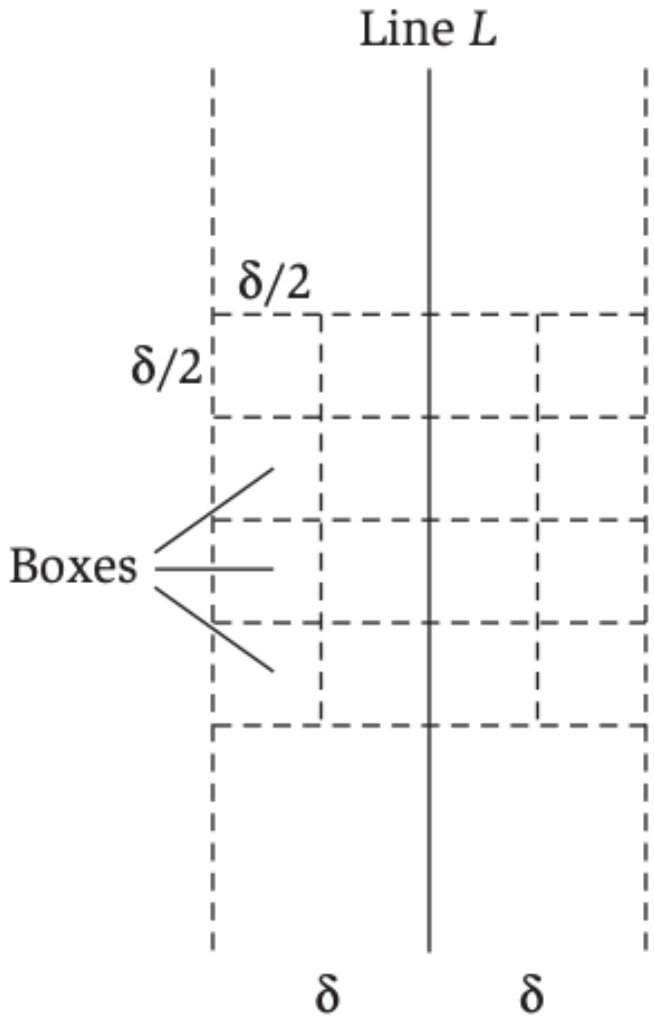
1. If $n \leq 4$, the do brute force.
2. Let Q be first half of P_x and let R be rest
3. Compute $Q_x, Q_y, R_x,$ and R_y using P_x and P_y
4. (q_x, q_y) = Closest-Pair(Q_x, Q_y)
5. (r_x, r_y) = Closest-Pair(R_x, R_y)
6. $\delta = \min((q_x, q_y), (r_x, r_y))$
7. Let S be points (x_i, y_i) in P s.t. $|x_i - x^*| \leq \delta$
8. (r_x, r_y) = Find-Closer(S, δ)
9. Return $\min((q_x, q_y), (r_x, r_y), (r_x, r_y))$

Clever Closer Pairs



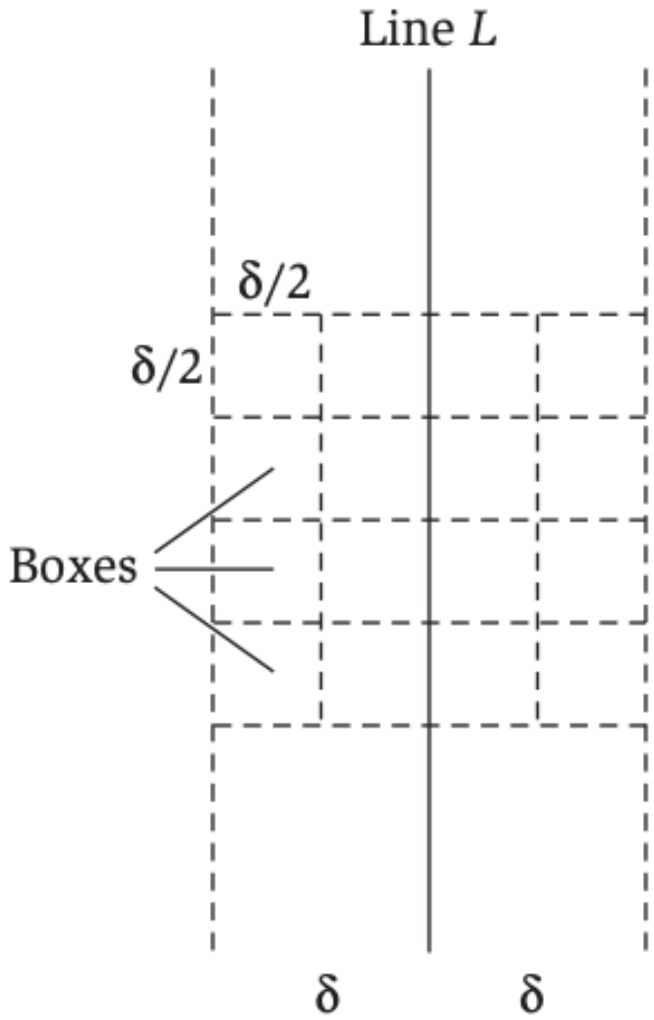
- We can show that a closer pair must be in S (See 5.8 in Section 5.4)
- We break S into blocks of size $\delta/2$ by $\delta/2$

Find-Closer



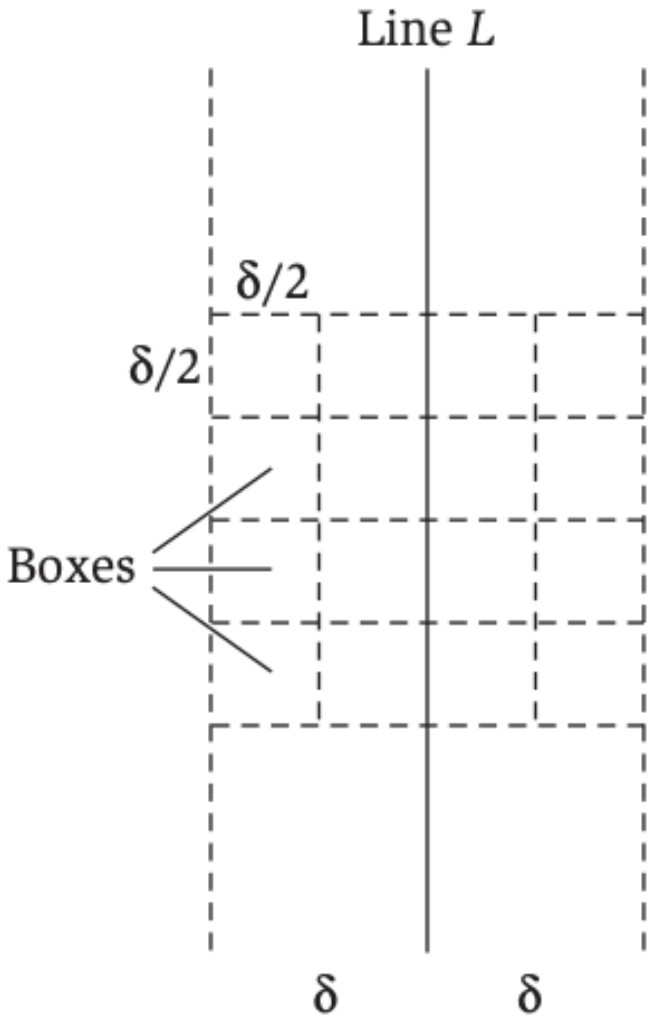
- We break S into blocks of size $\delta/2$ by $\delta/2$.
- **Question**: How many points can be in one box?

Find-Closer



- We break S into blocks of size $\delta/2$ by $\delta/2$.
- **Answer**: At most one point can be in a box since otherwise, it would contradict that the nearest pair on either side was distance δ away.

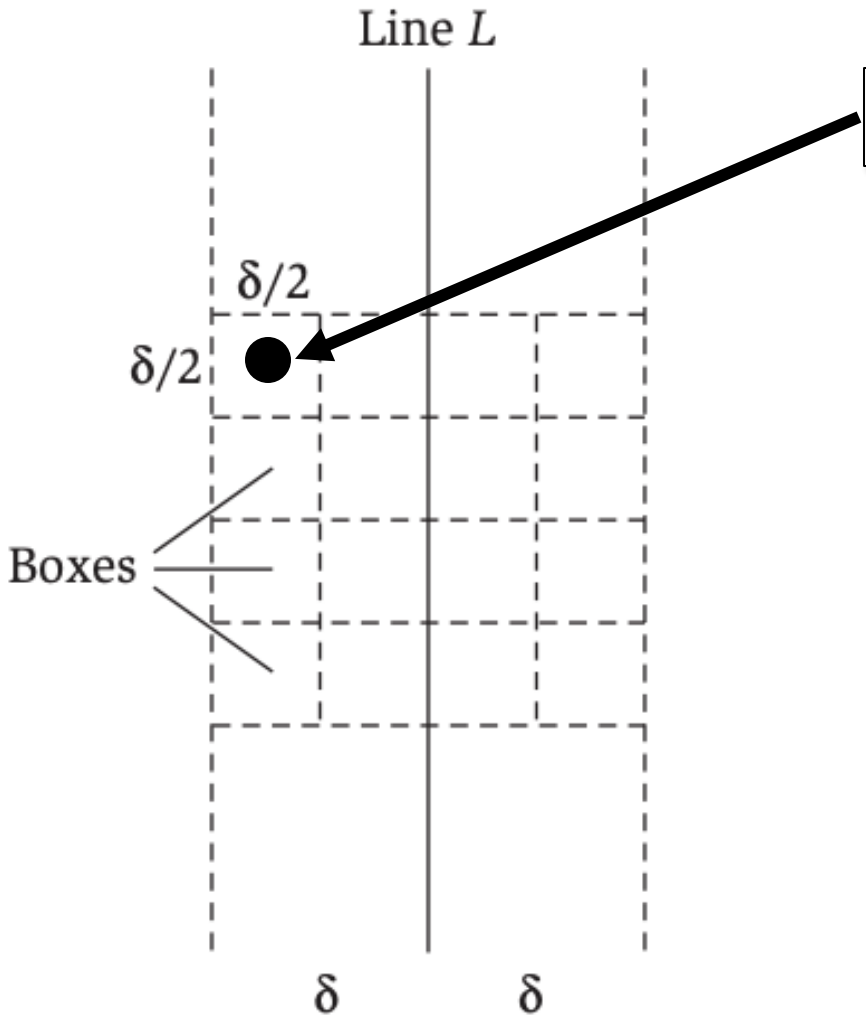
Kickass Property Lemma (KT 5.10 in S 5.4)



Lemma: If $s, s' \in S$ have the property that $d(s, s') \leq \delta$ then s and s' are within 15 positions of each other in the list S when sorted by y-coordinates, S_y.

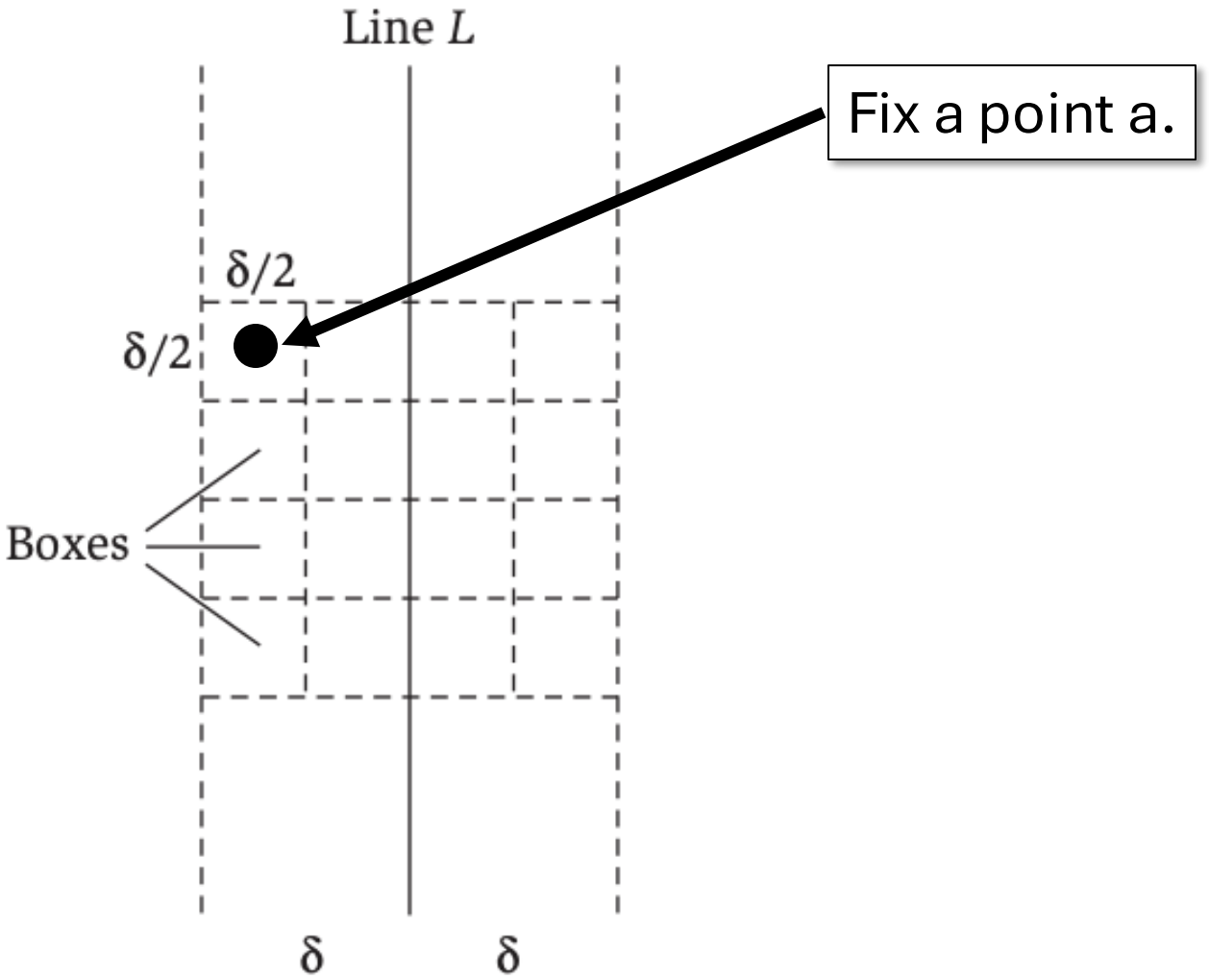
That is, if you sort the points in the slice by their y-coordinates, then any pair that is closer than δ must be within 15 spaces of each other in the sorted list.

Kickass Property Lemma Picture



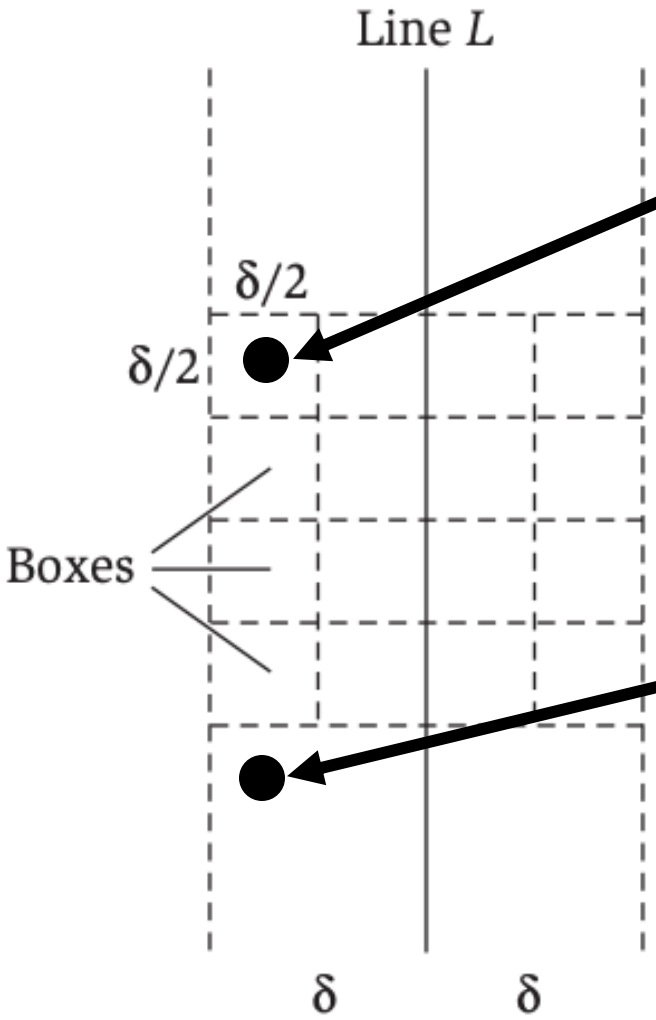
Fix a point a.

Kickass Property Lemma Picture



Recall at most one point per box.

Kickass Property Lemma Picture



Fix a point a.

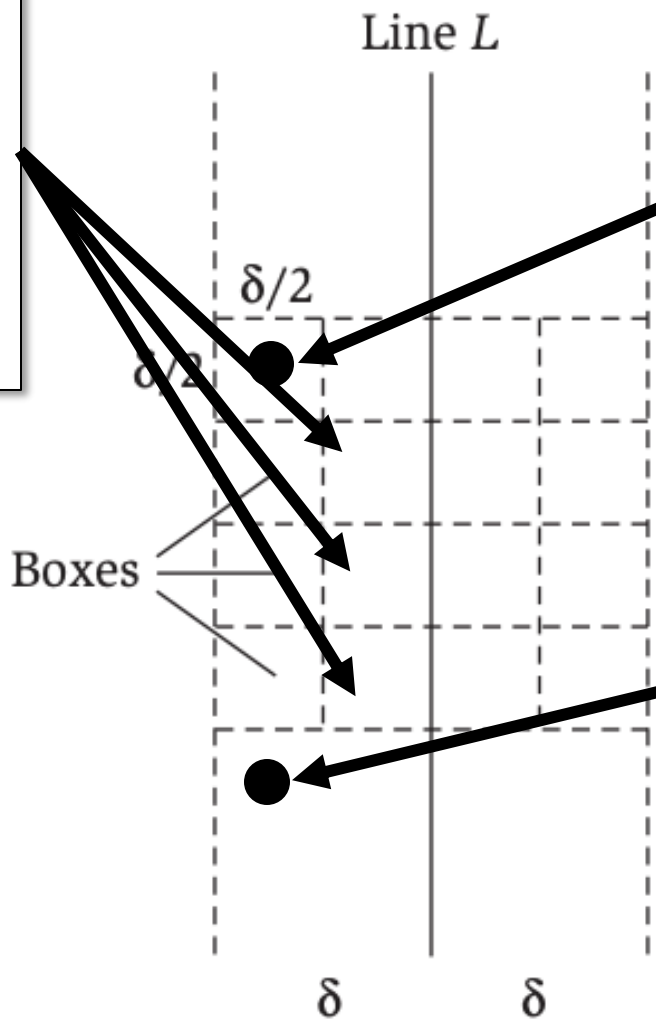
Recall at most one point per box.

Any point b that is at least 15 spaces away in sorted array must be down here.

Kickass Property Lemma Picture

That means there are always at least three stacked boxes between them, i.e. distance $> 3\delta/2$

Recall at most one point per box.



Fix a point a.

Any point b that is at least 15 spaces away in sorted array must be down here.