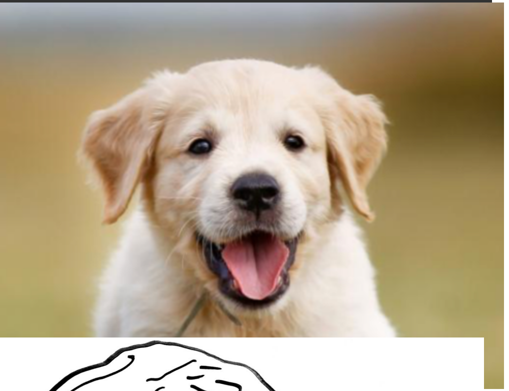
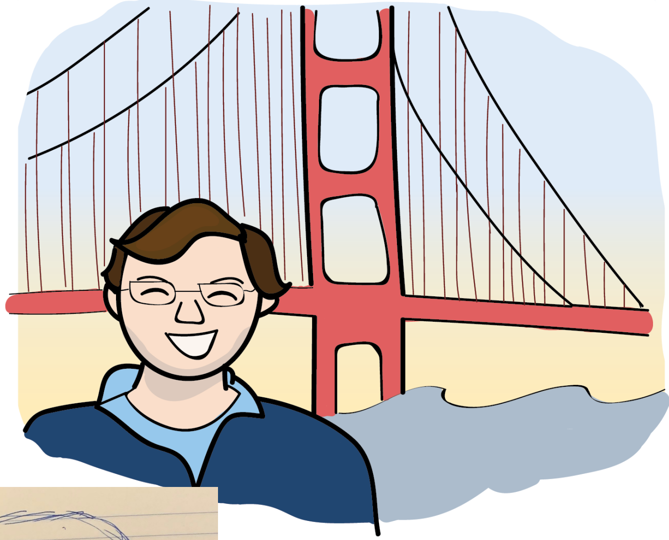
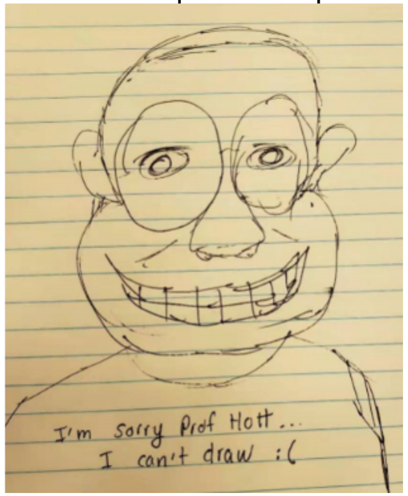
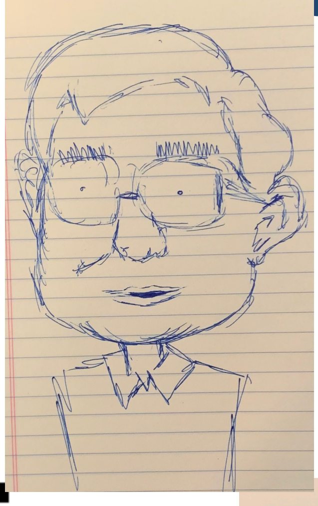
CS4102 Algorithms

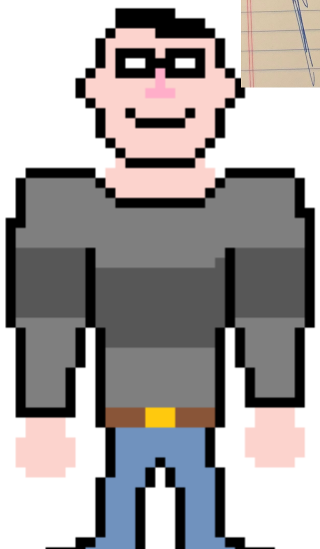
Fall 2019

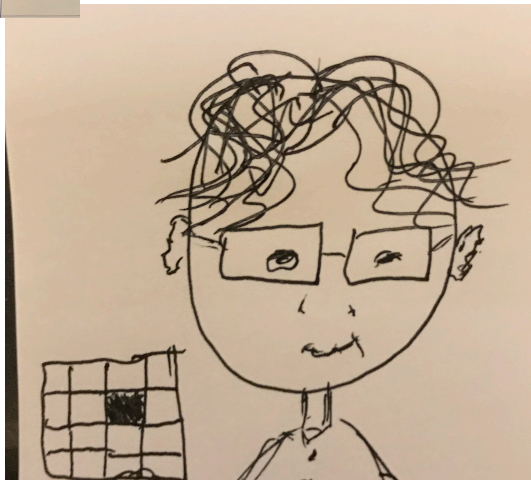












Today’s Keywords

- Reductions
- NP Hard, NP Completeness
- k-Clique
- Convex Hull
- Graham Scan
- Jarvis’ March
- Chan’s Algorithm

CLRS Readings

- Chapter 34

Homeworks

- HW9 due Thursday at 11pm
  - Reductions, Graphs
  - Written (LaTeX)
- HW10C due Thursday at 11pm
  - Implement a problem from HW9
  - No late submissions

Final Exam

- Monday, December 9, 7pm in Maury 209 (our section)
  - Practice exam out! Solutions later this week
  - Review session this weekend (look for an email)
  - SDAC: please schedule for some time on Monday 12/9

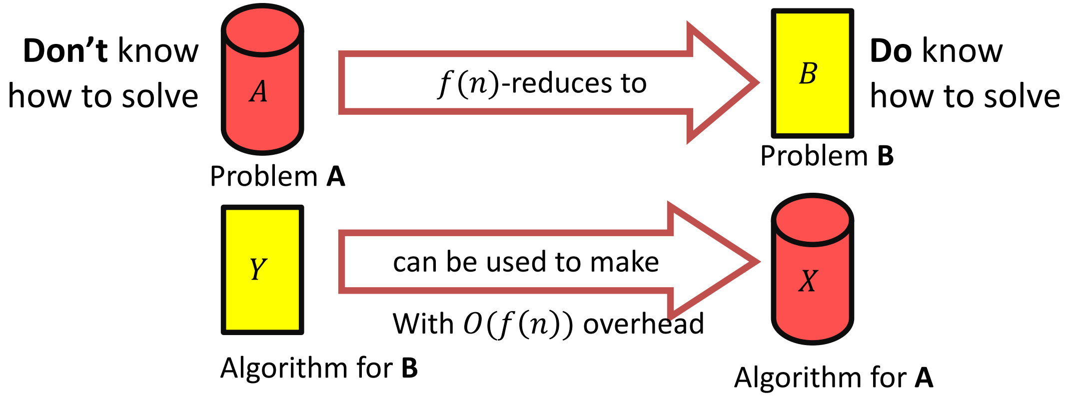
Reductions

- Algorithm technique of supreme ultimate power
- Convert instance of problem A to an instance of Problem B
- Convert solution of problem B back to a solution of problem A

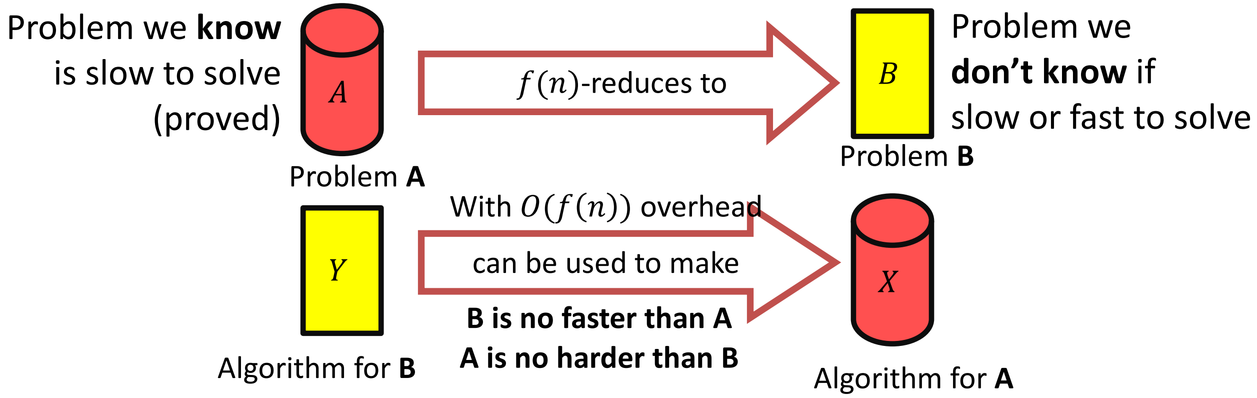
Reductions

Possible uses

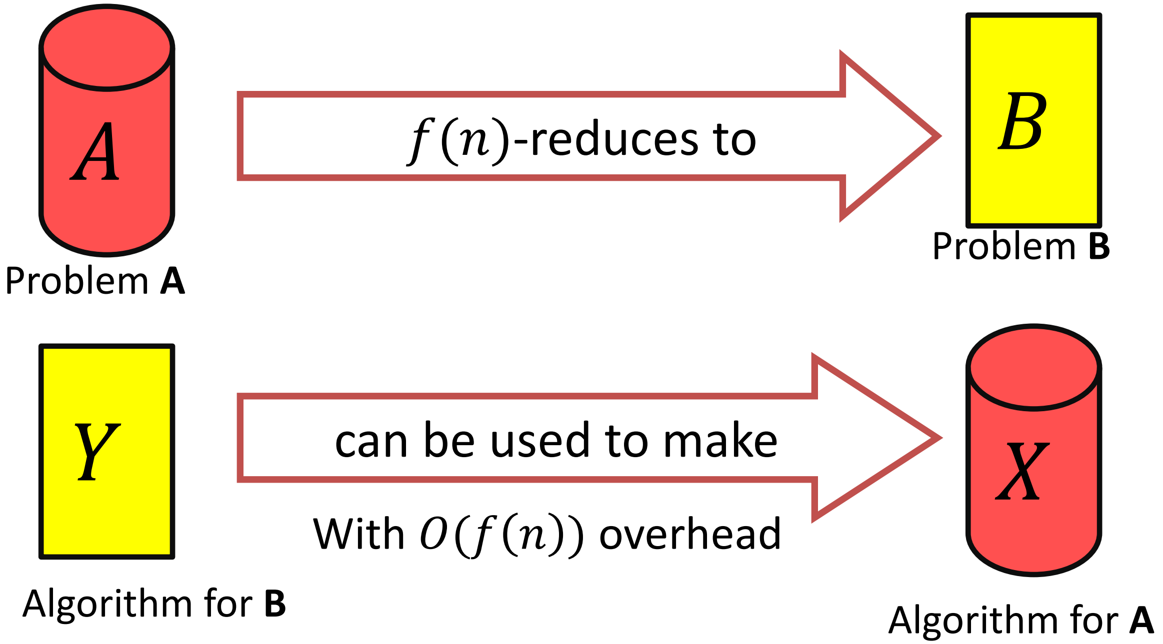
- Use solver for B to solve A



- Prove lower bound for B by showing it's as hard as A



Reduction Proof Notation



A is not a harder problem than B

$$A \le B$$

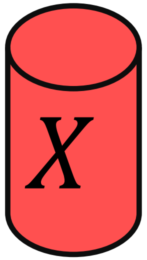
If A requires time $\Omega(f(n))$ time then B also requires $\Omega(f(n))$ time

$$A \le_{f(n)} B$$

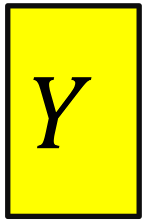
Or we could have solved A faster using B's solver!

Proof of Lower Bound by Reduction

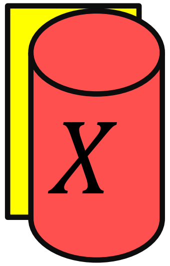
<u>To Show: Y is slow</u>



1. We know X is slow (by a proof)
(e.g., X = some way to open the door)



2. Assume Y is quick [toward contradiction]
(Y = some way to light a fire)

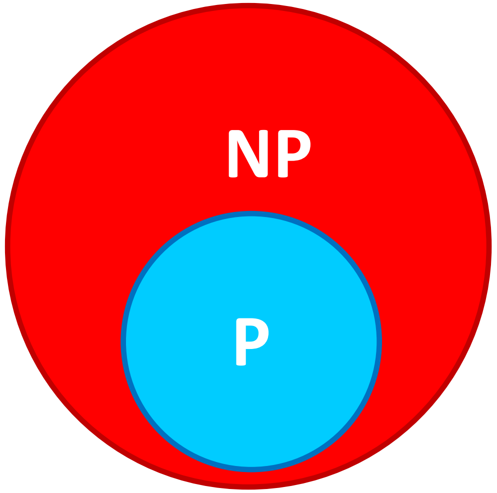


3. Show how to use Y to perform X quickly

4. X is slow, but Y could be used to perform X quickly
conclusion: Y must not actually be quick

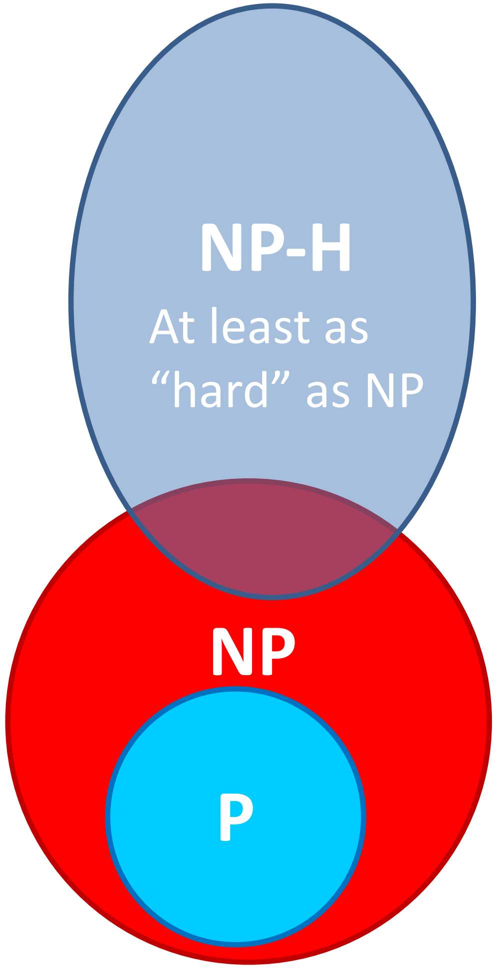
P vs NP

- P
  - Deterministic Polynomial Time
  - Problems solvable in polynomial time
    - $O(n^p)$ for some number p
- NP
  - Non-Deterministic Polynomial Time
  - Problems verifiable in polynomial time
    - $O(n^p)$ for some number p
- Open Problem: Does P=NP?
  - Certainly $P \subseteq NP$

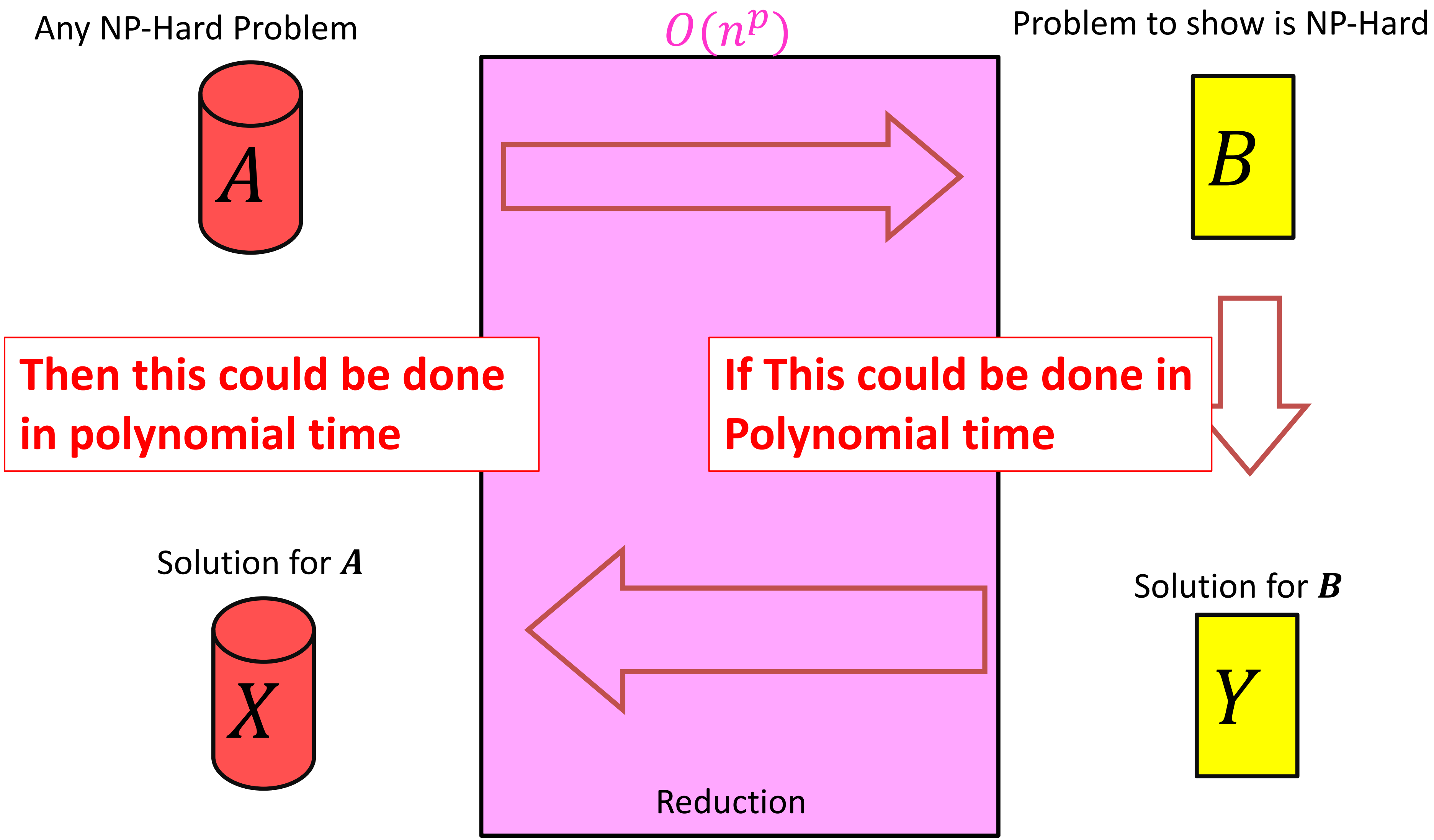


NP-Hard

- How can we try to figure out if P=NP?
- Identify problems at least as "hard" as NP
  - If any of these "hard" problems can be solved in polynomial time, then all NP problems can be solved in polynomial time.
- Definition: NP-Hard:
  - B is NP-Hard if $\forall A \in NP, A \leq_p B$
  - $A \leq_p B$ means A reduces to B in polynomial time

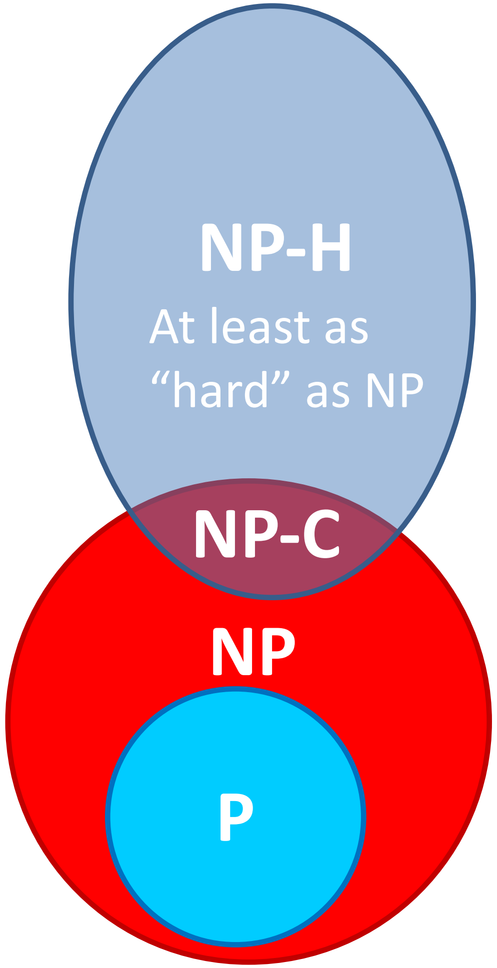


NP-Hardness Reduction



NP-Complete

- “Together they stand, together they fall”
- Problems solvable in polynomial time iff ALL NP problems are
- NP-Complete = NP ∩ NP-Hard
- **How to show a problem is NP-Complete?**
  - Show it belongs to NP
    - Give a polynomial time verifier
  - Show it is NP-Hard
    - Give a reduction from another NP-H problem



We now just need a FIRST NP-Hard problem

3-SAT

- Shown to be NP-Hard by Cook and Levin (independently)
- Given a 3-CNF formula (logical AND of clauses, each an OR of 3 variables), Is there an assignment of true/false to each variable to make the formula true?

$$\underbrace{(x \vee y \vee z)}_{\textbf{Clause}} \wedge (x \vee \bar{y} \vee y) \wedge (u \vee y \vee \bar{z}) \wedge (z \vee \bar{x} \vee u) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$$

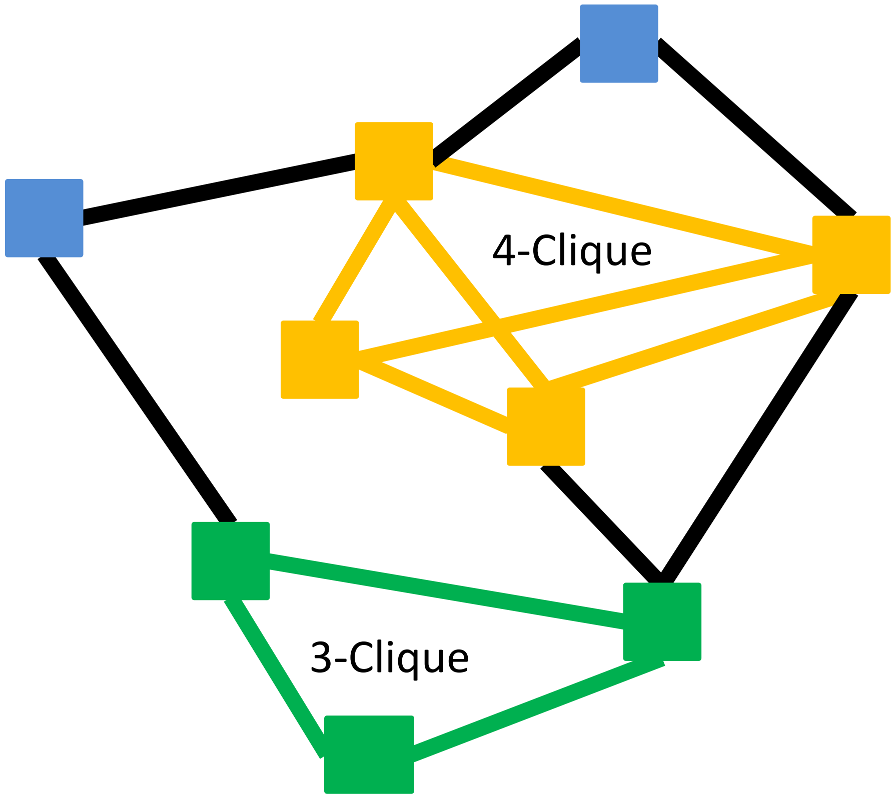
Variables

$$\boldsymbol{x = true}$$
$$\boldsymbol{y = false}$$
$$\boldsymbol{z = false}$$
$$\boldsymbol{u = true}$$

k-Clique Problem

Given a graph G and a number k, is there a *clique* of size k?

- Clique: A complete subgraph



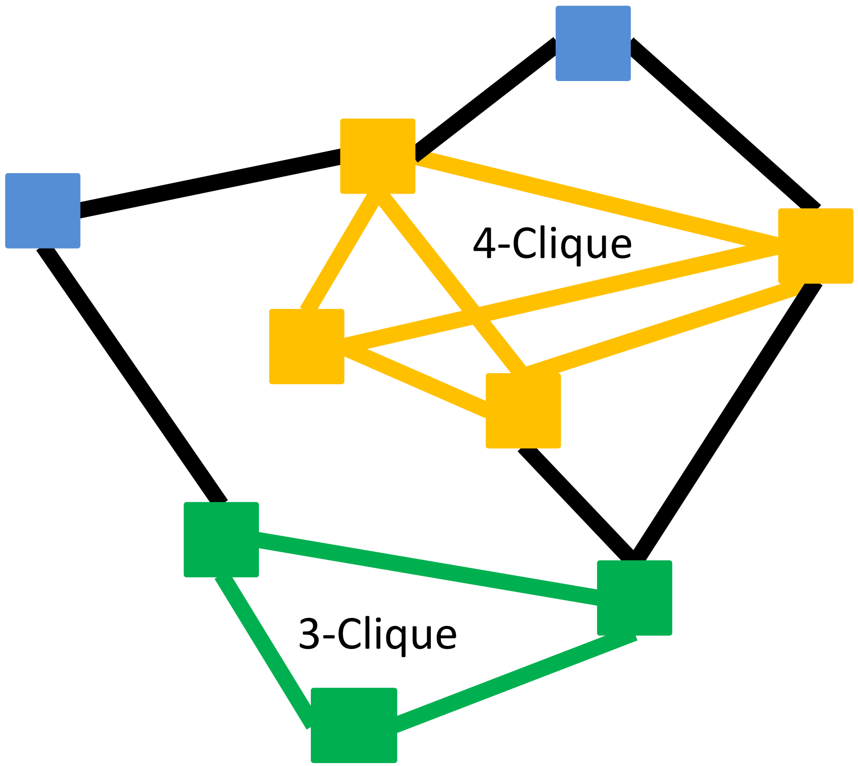
k-Clique is NP-Complete

1. Show that it belongs to NP
   – Give a polynomial time verifier
2. Show it is NP-Hard
   – Give a reduction from a known NP-Hard problem
   – We will show $3SAT \leq_p kClique$

k-Clique is NP

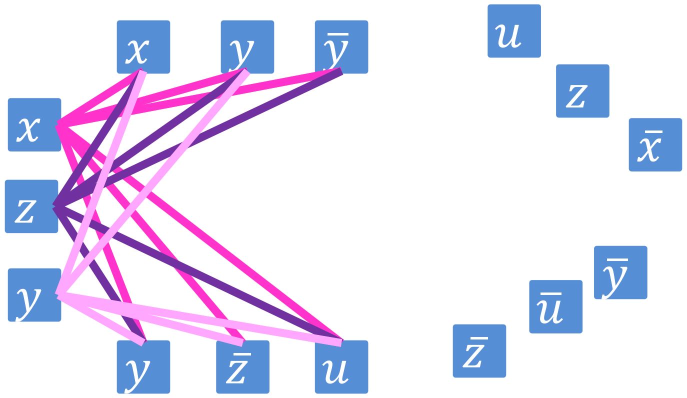
Given a Graph, k, and a potential solution

1. Check that the solution has k nodes
2. Check that every pair of nodes share an edge



Instance of 3SAT to Instance of kClique

$$(x \vee y \vee z) \wedge (x \vee \bar{y} \vee y) \wedge (u \vee y \vee \bar{z}) \wedge (z \vee \bar{x} \vee u) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$$



(also do this for the other clauses, omitted due to clutter)

For each clause, produce a node for each of its three variables

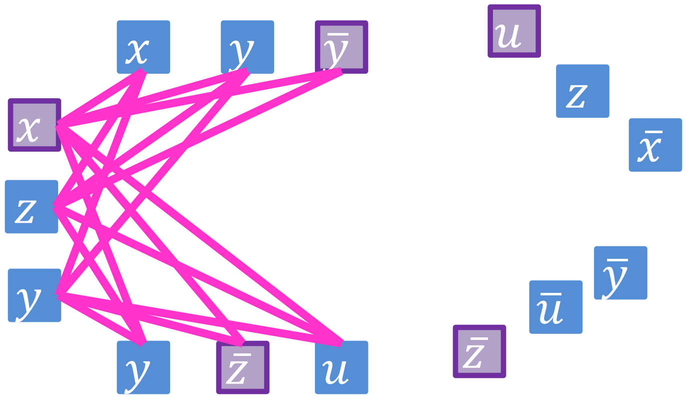
Connect each node to all non-contradictory nodes in the other clauses (i.e., anything that's not its negation)

Let k = number of clauses

There is a k-Clique in this graph **iff** there is a satisfying assignment

kClique ⇒ Satisfying Assignment

$$(x \lor y \lor z) \land (x \lor \bar{y} \lor y) \land (u \lor y \lor \bar{z}) \land (z \lor \bar{x} \lor u) \land (\bar{x} \lor \bar{y} \lor \bar{z})$$



$\boldsymbol{x = true}$
$\boldsymbol{y = false}$
$\boldsymbol{z = false}$
$\boldsymbol{u = true}$

There are k triplets in the graph, and no two nodes in the same triplet are adjacent

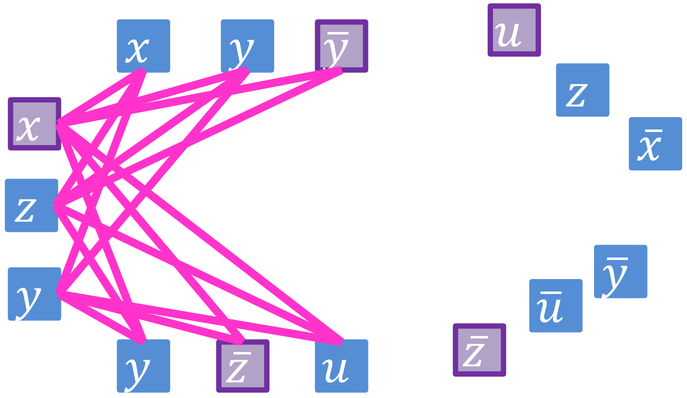
To have a k-Clique, must have one node from each triplet

Cannot select a node for both a variable and its negation

Therefore selection of nodes is a satisfying assignment

Satisfying Assignment $\Rightarrow$ kClique

$$(x \vee y \vee z) \wedge (x \vee \bar{y} \vee y) \wedge (u \vee y \vee \bar{z}) \wedge (z \vee \bar{x} \vee u) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$$

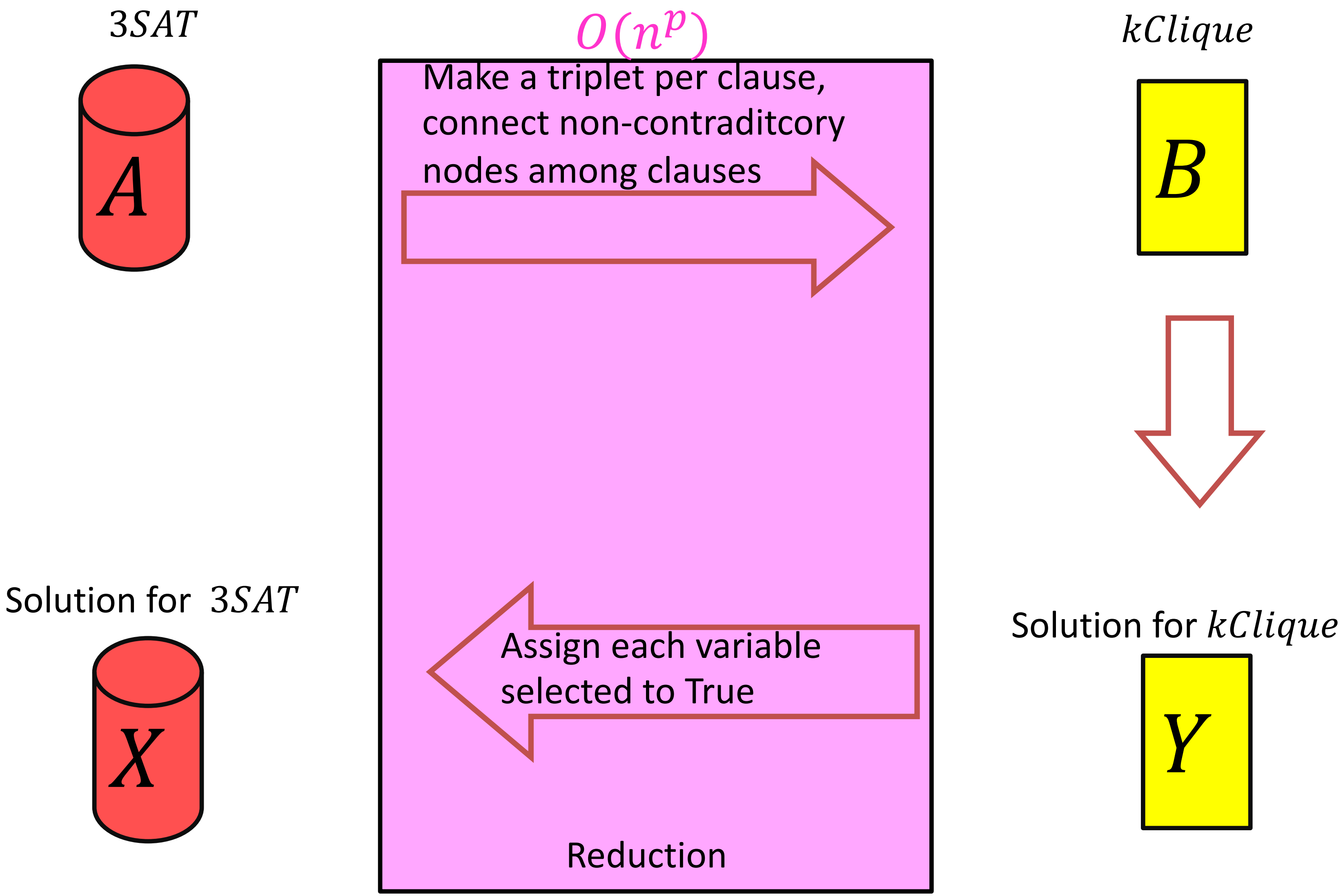


$\boldsymbol{x = true}$
$\boldsymbol{y = false}$
$\boldsymbol{z = false}$
$\boldsymbol{u = true}$

Select one node for a true variable from each clause

There will be k nodes selected
We can't select both a node and its negation
All nodes will be non-contradictory, so they will be pairwise adjacent

$3SAT \leq_p kClique$

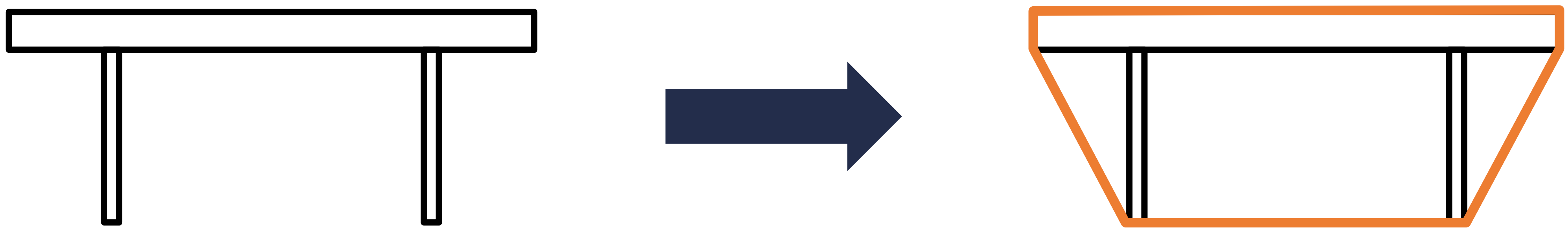


NP-Complete Problems

- We’ve now seen 4 NP-Complete problems
  - 3SAT
  - k-Independent Set
  - k-Vertex Cover
  - k-Clique
- You’ve seen 2 more in HW9
  - Backpacking
  - Subset Sum

One More Reduction

The Convex Hull Problem



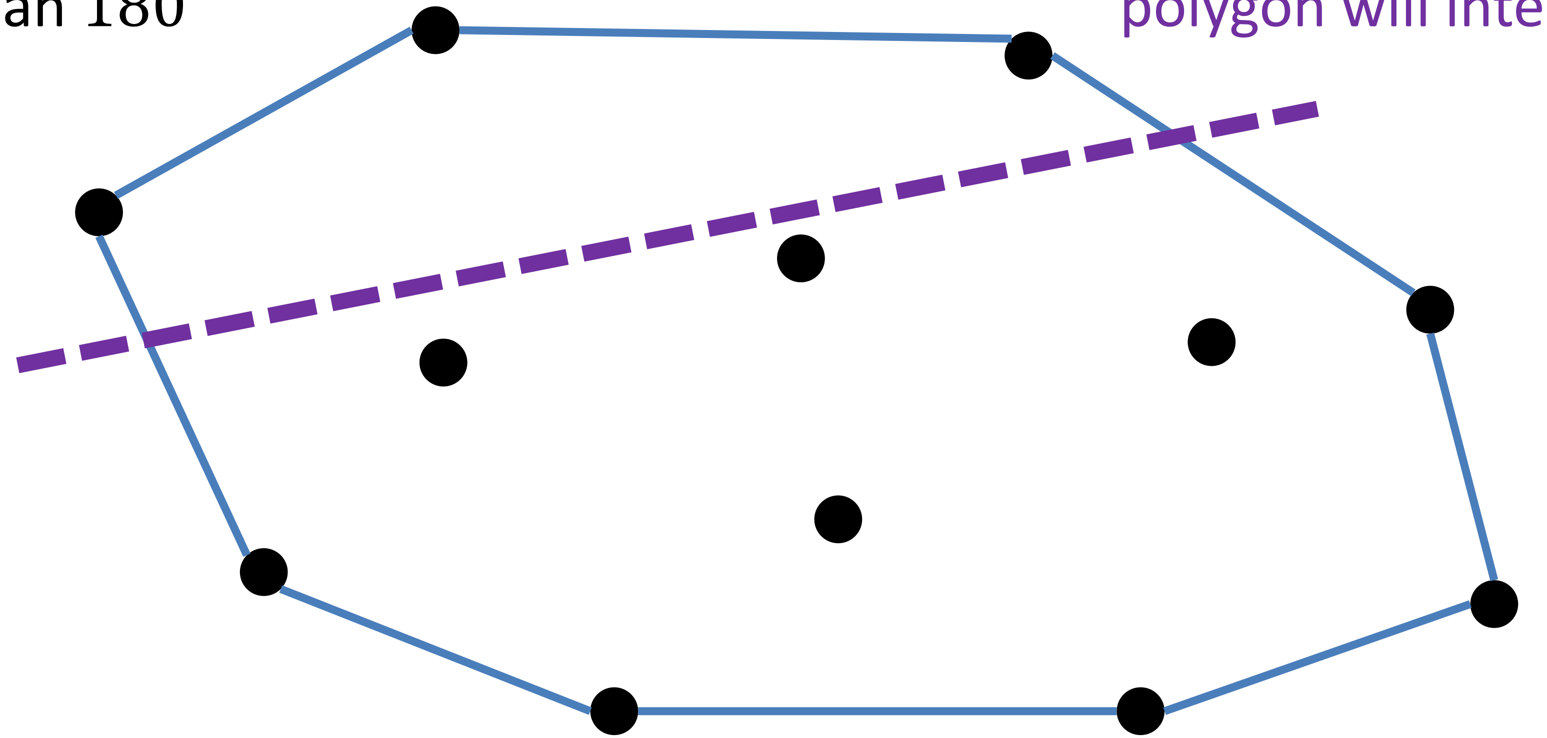
Problem: find the smallest convex polygon that bounds a shape (or more generally, a collection of points)

Example application: collision detection in computer graphics, vision, robotics; also useful for solving other problems, especially in computational geometry (e.g., furthest pair of points)

The Convex Hull Problem

Convex polygon: all interior angles are less than $180°$

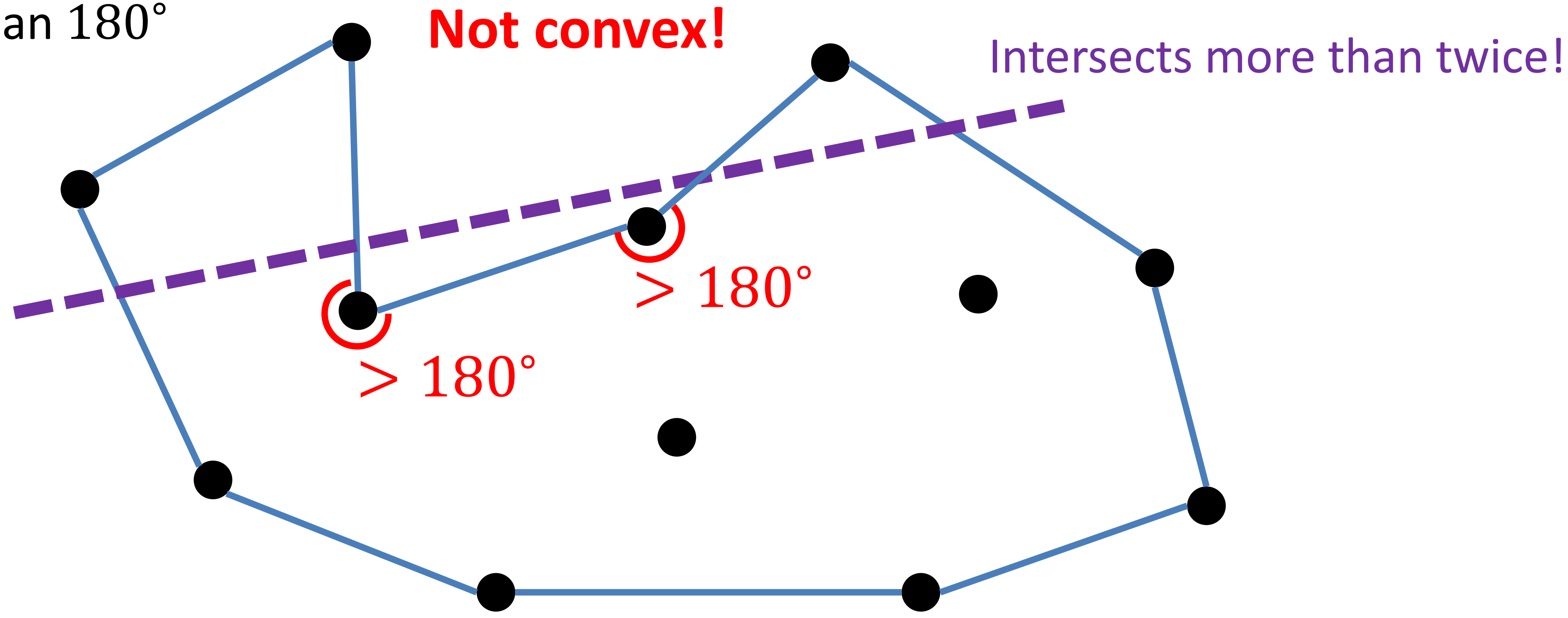
Equivalently: line drawn through polygon will intersect exactly twice



Given a set of n points, find the smallest convex polygon such that every point is either on the boundary or in the interior of the polygon

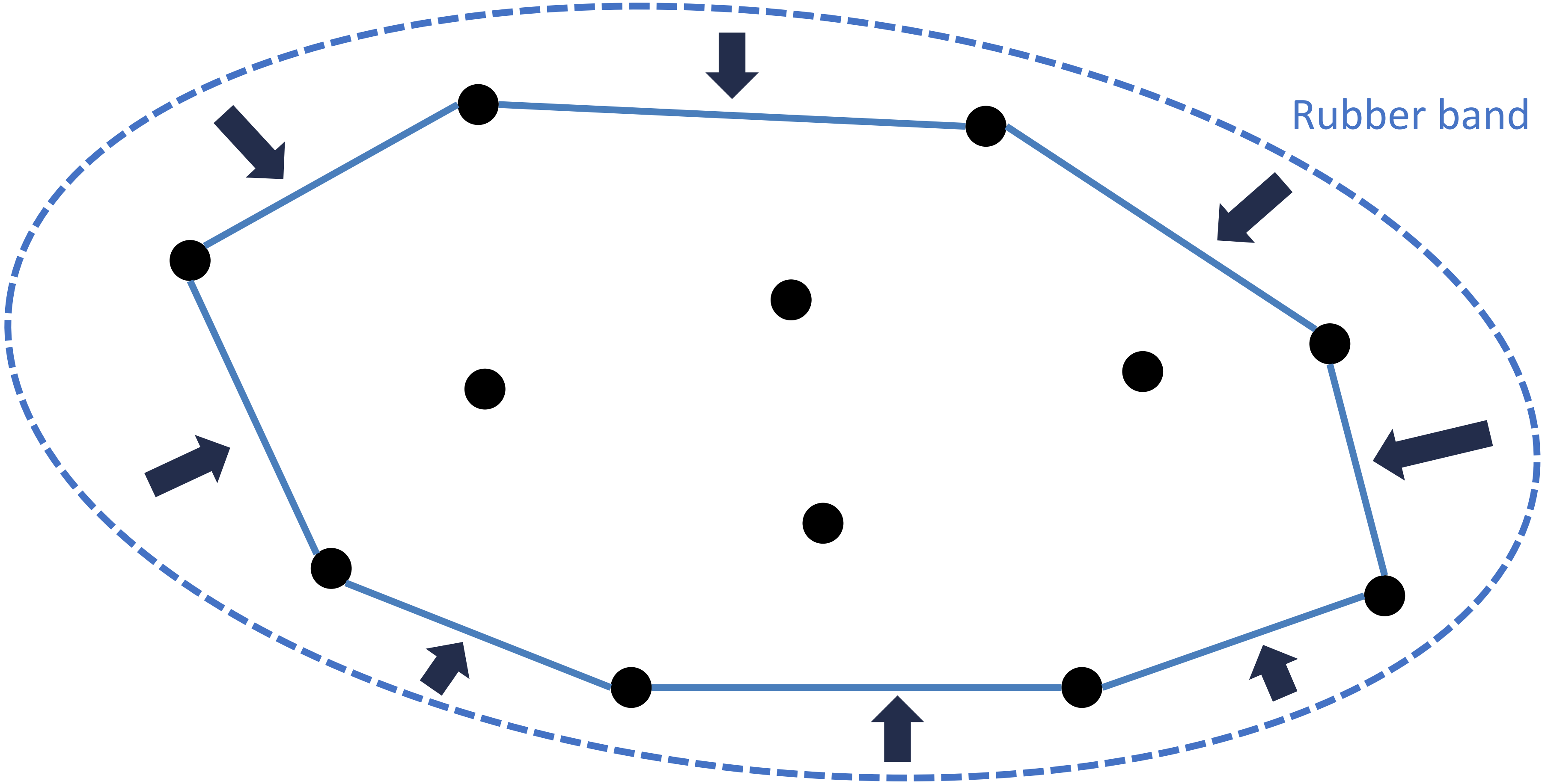
The Convex Hull Problem

Convex polygon: all interior angles are less than $180°$



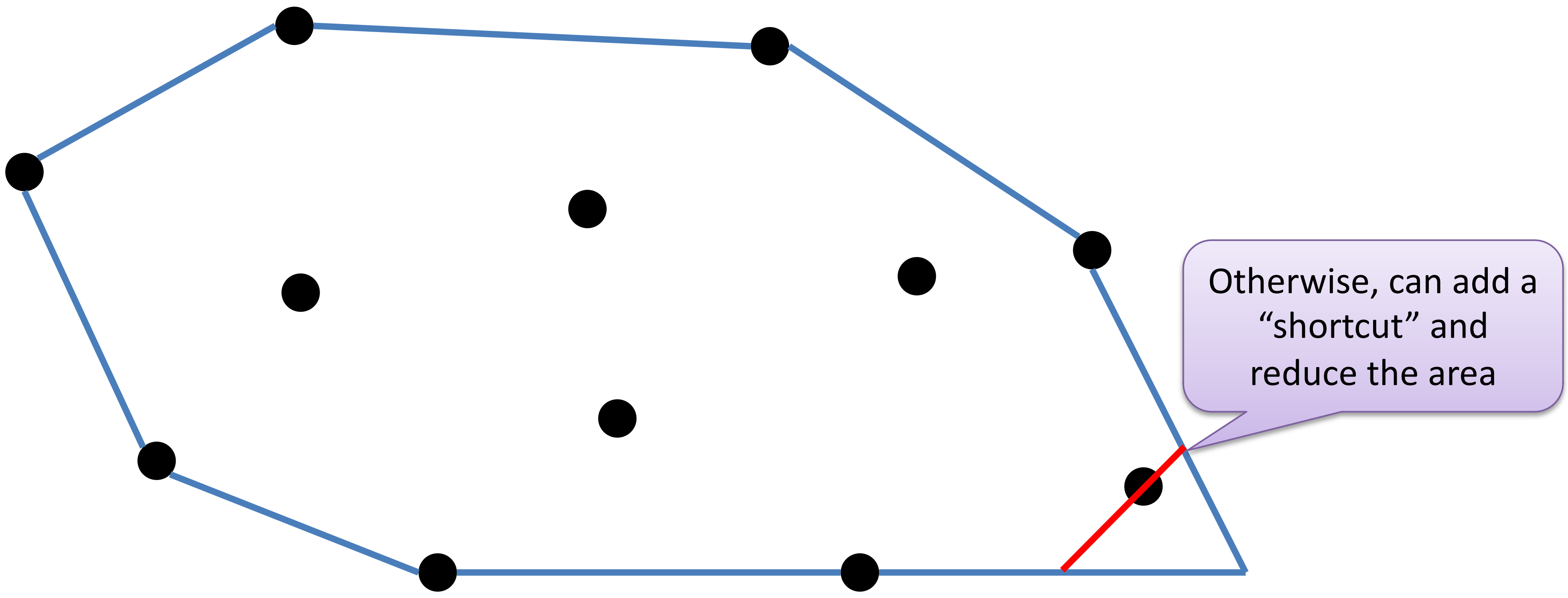
Given a set of n points, find the smallest convex polygon such that every point is either on the boundary or in the interior of the polygon

The Convex Hull Problem



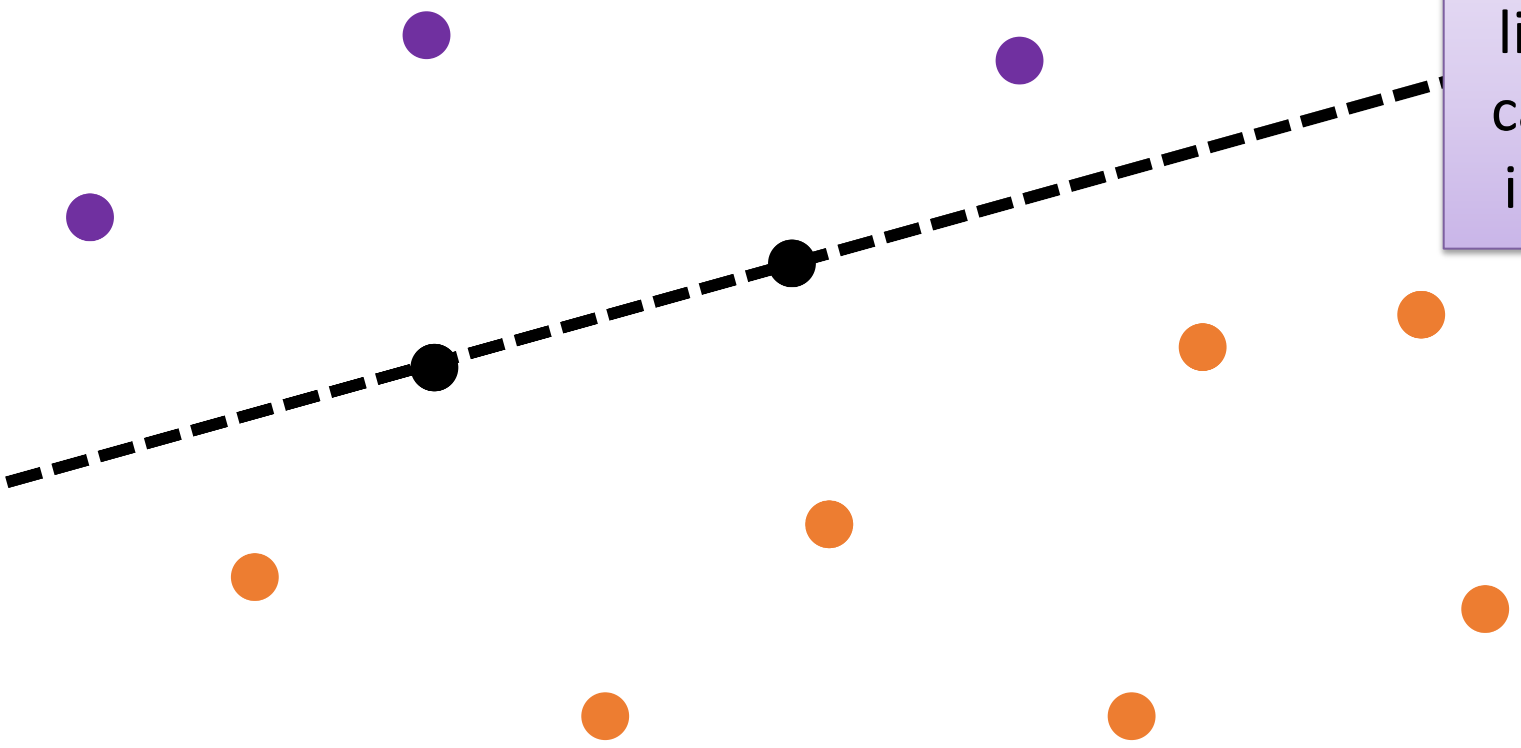
Rubber band analogy: imagine the points are nails sticking out of a board and wrapping a rubber band to encompass the nails; convex hull is resulting shape

The Convex Hull Problem



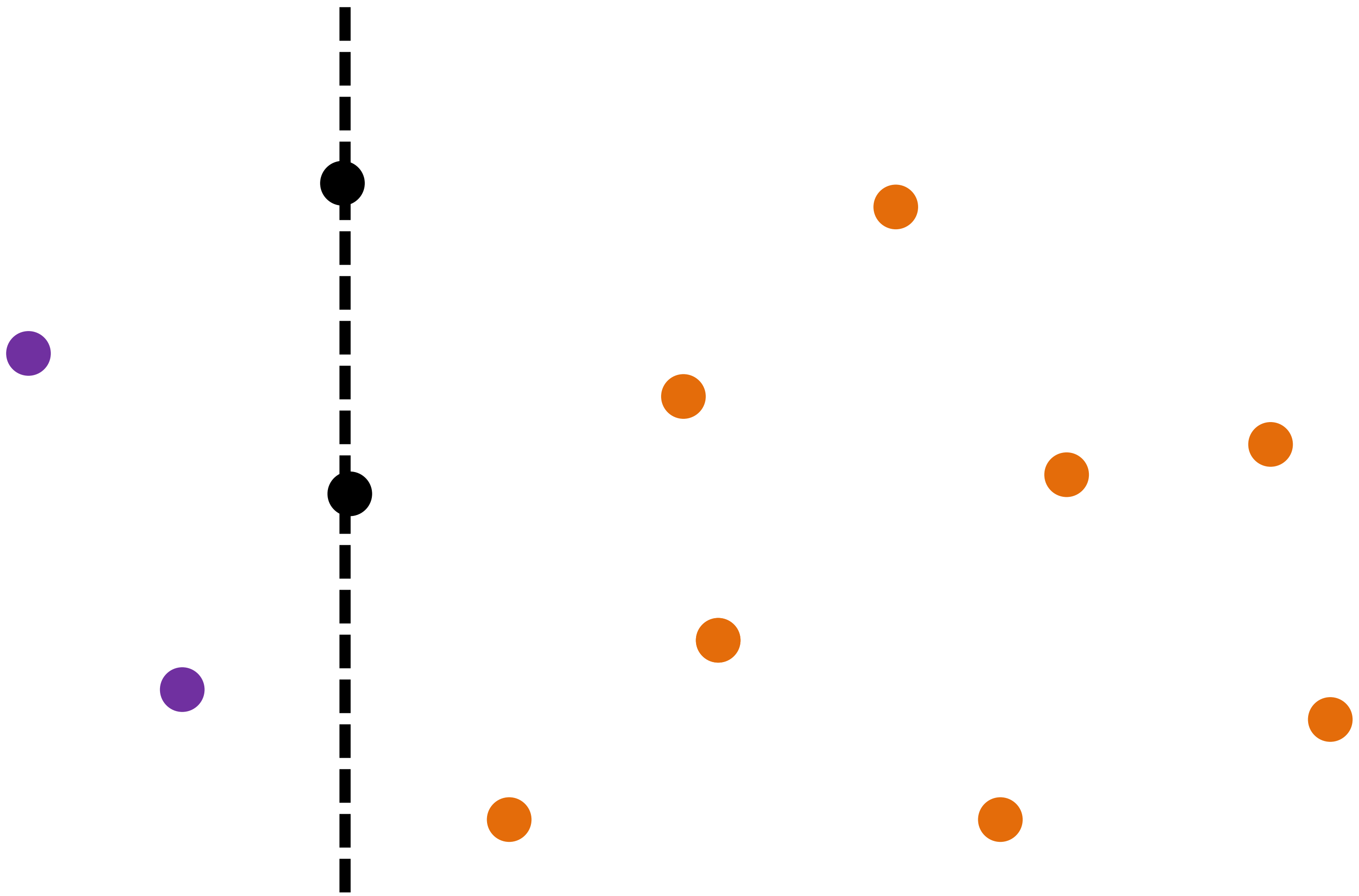
Observation: every point on the convex hull is one of the input points

A Brute Force Approach

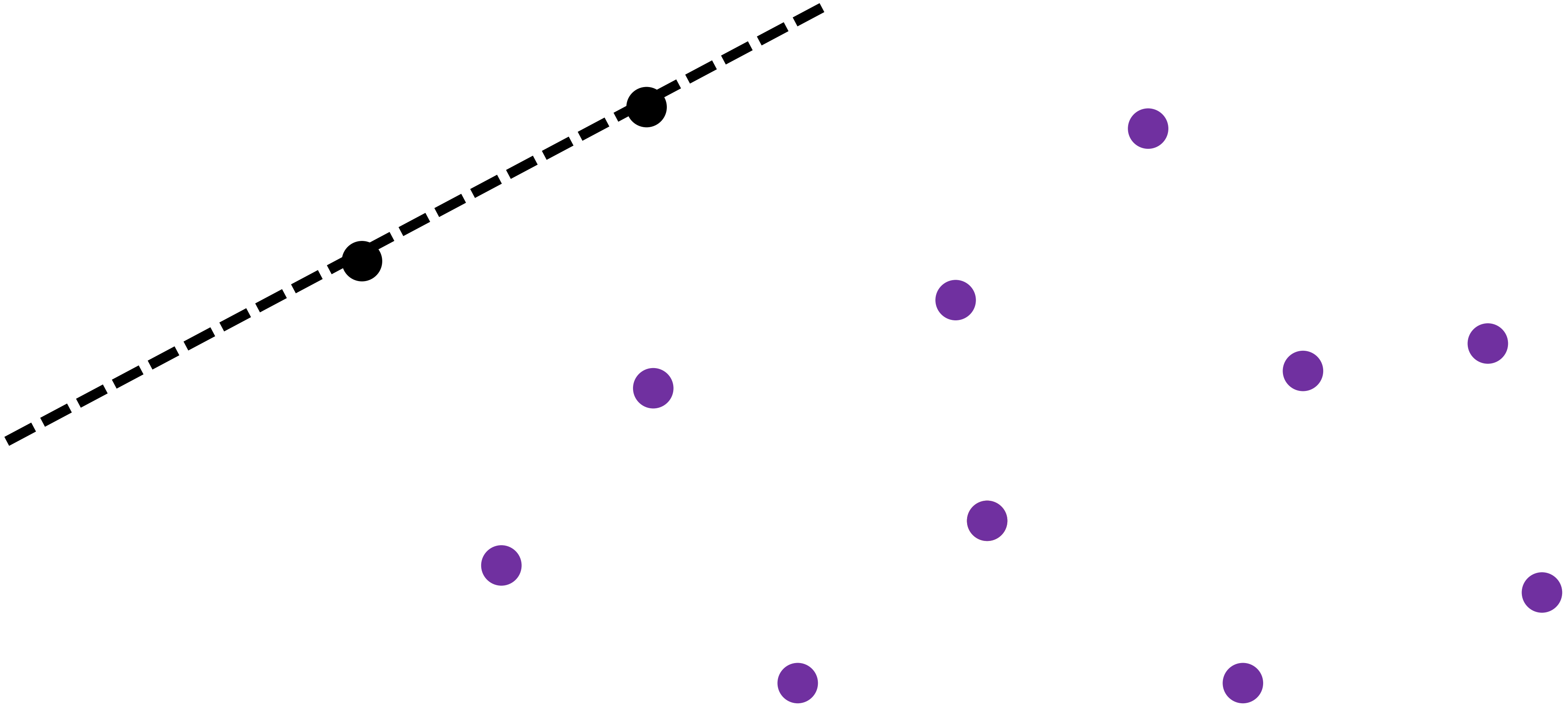


If there are points on both sides of the line, then the **pair** cannot be an edge in the convex hull

A Brute Force Approach



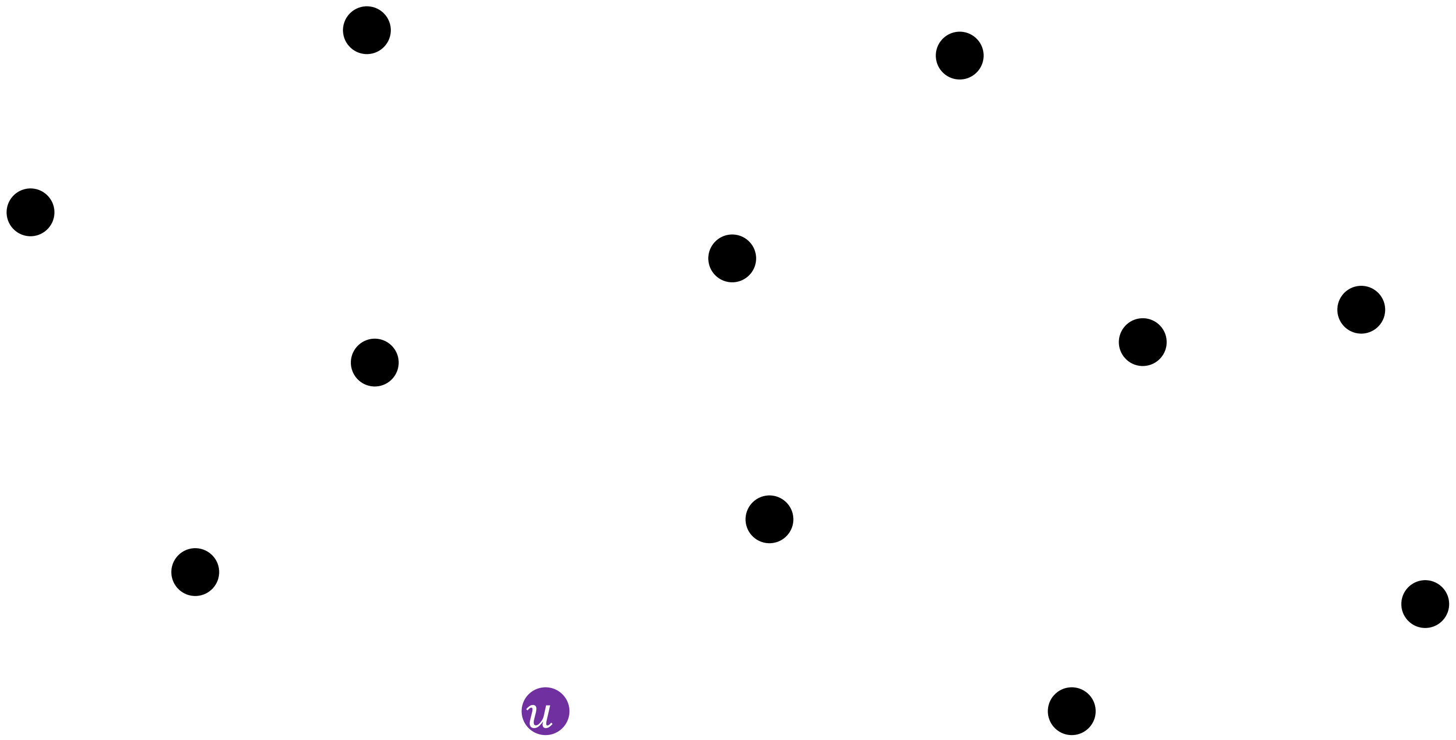
A Brute Force Approach



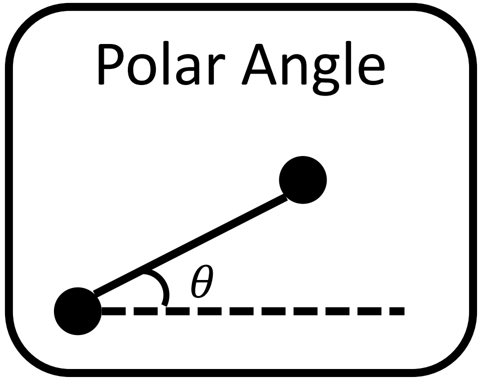
Brute force approach: for every pair of points, check if all other points are on the same side of the line $O(n^3)$

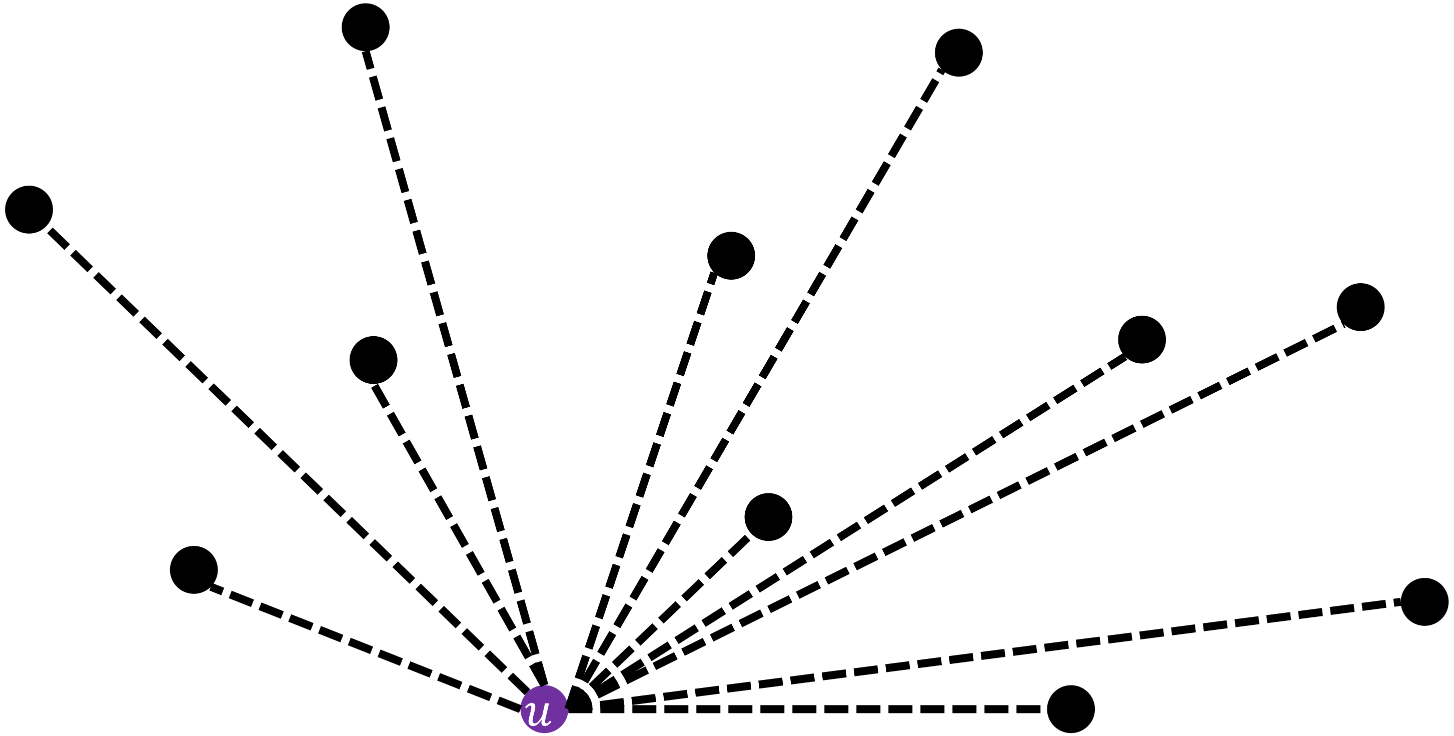
Graham's Algorithm

Observation: Extremal points must be part of the convex hull
(e.g., bottom-most point, left-most point, etc.)



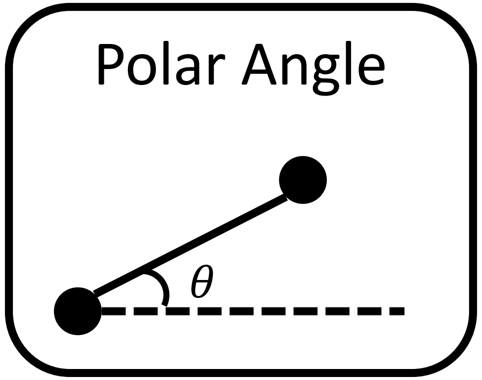
Graham's Algorithm

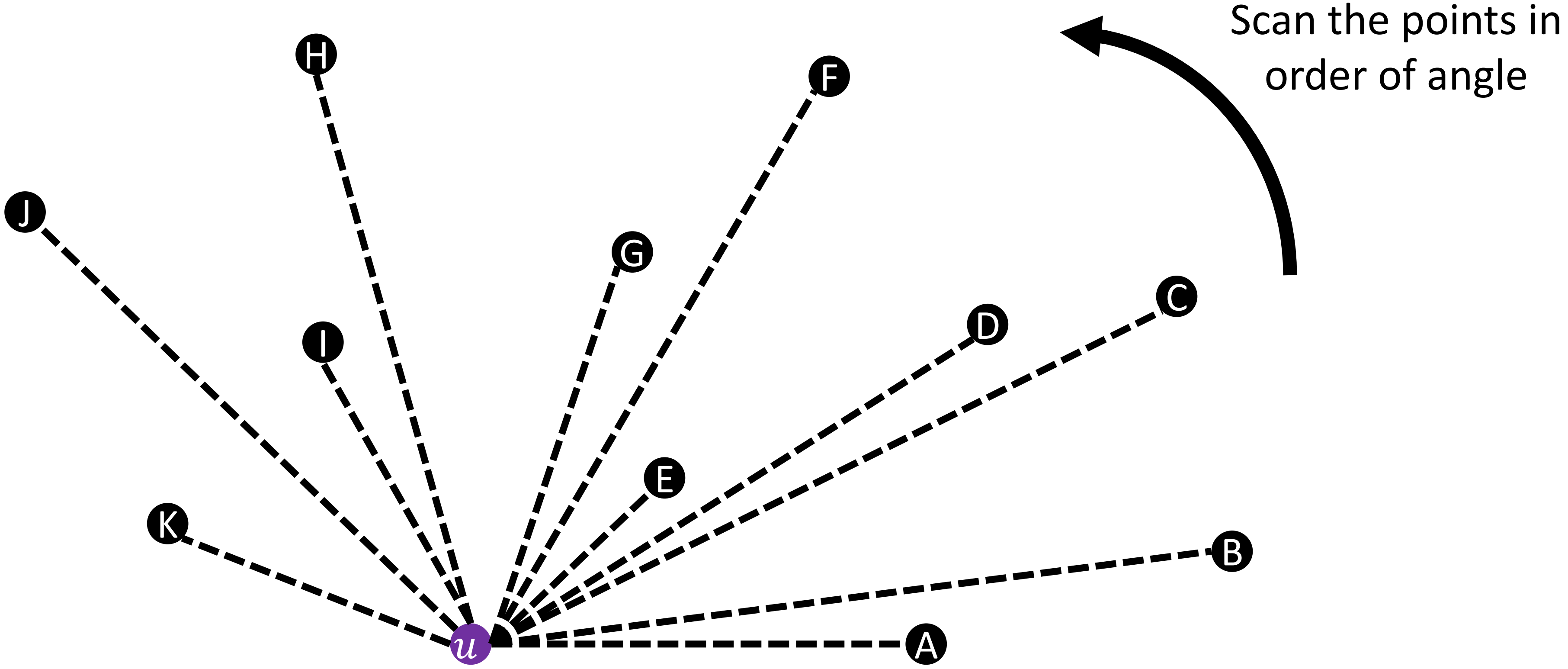




Consider the (polar) angle formed between base point u and every other point

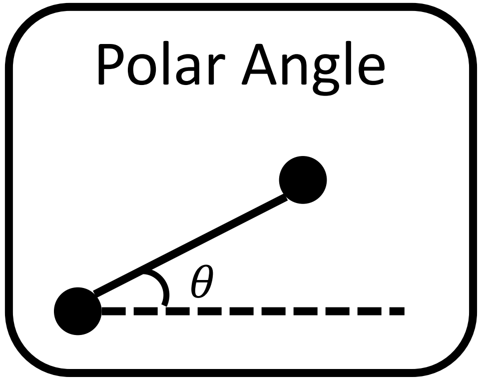
Graham’s Algorithm

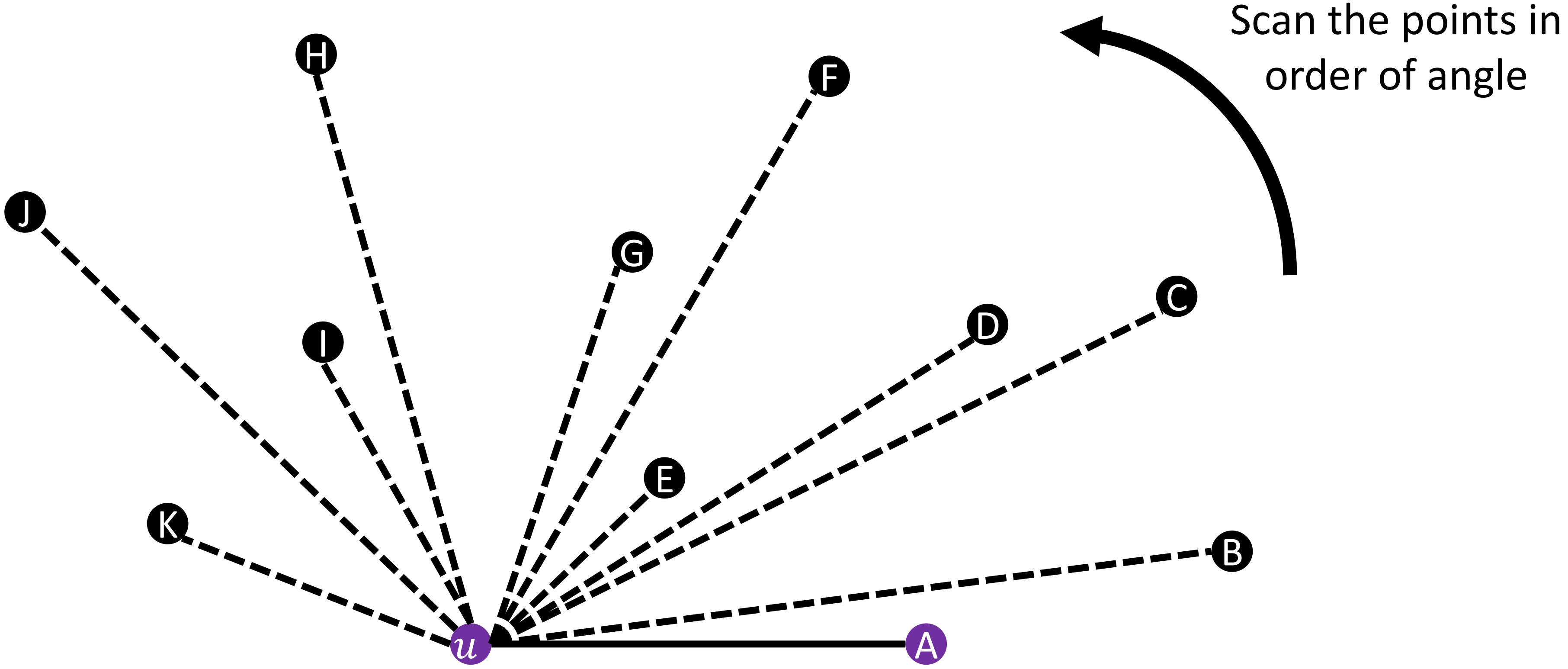




Idea: In order of angle, add points to the convex hull as long as it <u>preserves</u> convexity

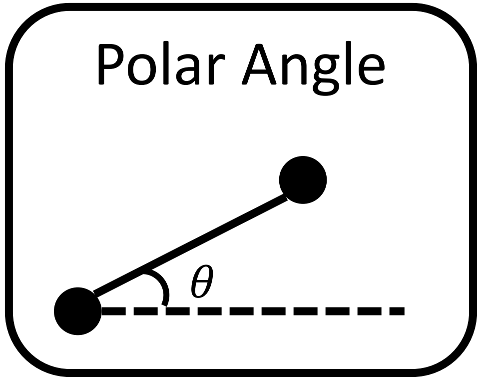
Graham's Algorithm

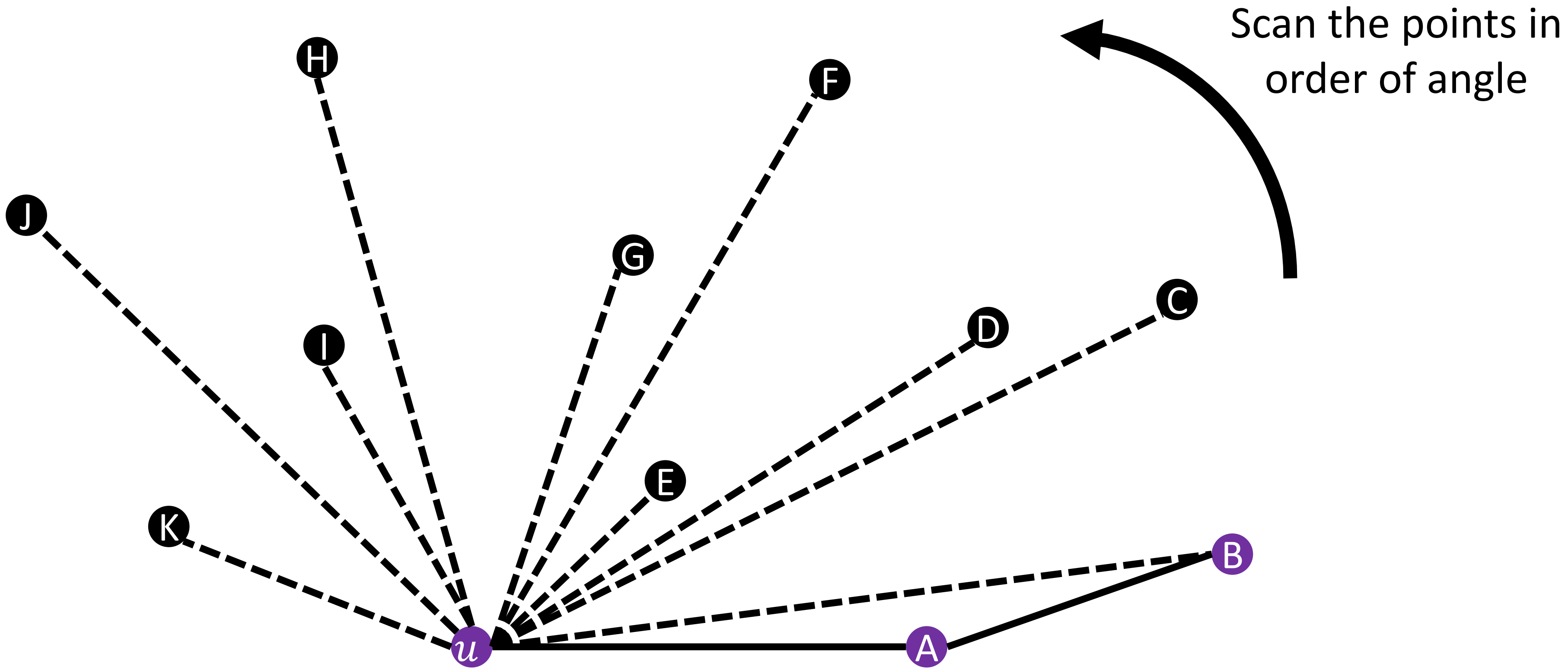




Idea: In order of angle, add points to the convex hull as long as it preserves convexity

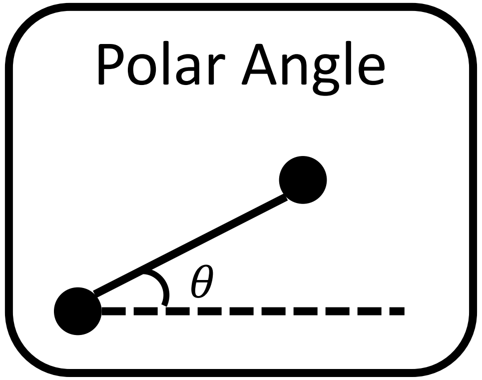
Graham's Algorithm

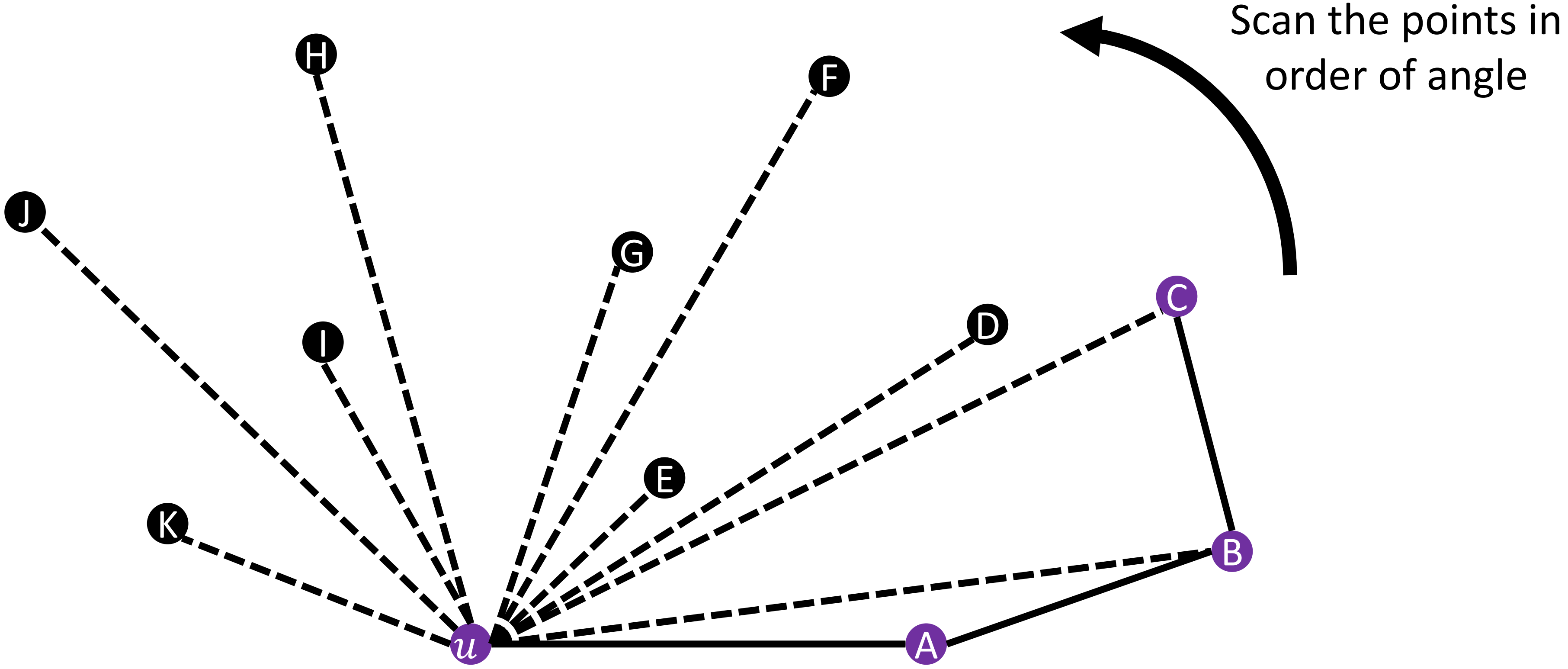




Idea: In order of angle, add points to the convex hull as long as it preserves convexity

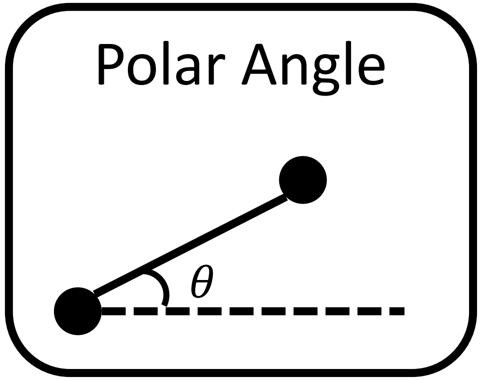
Graham's Algorithm

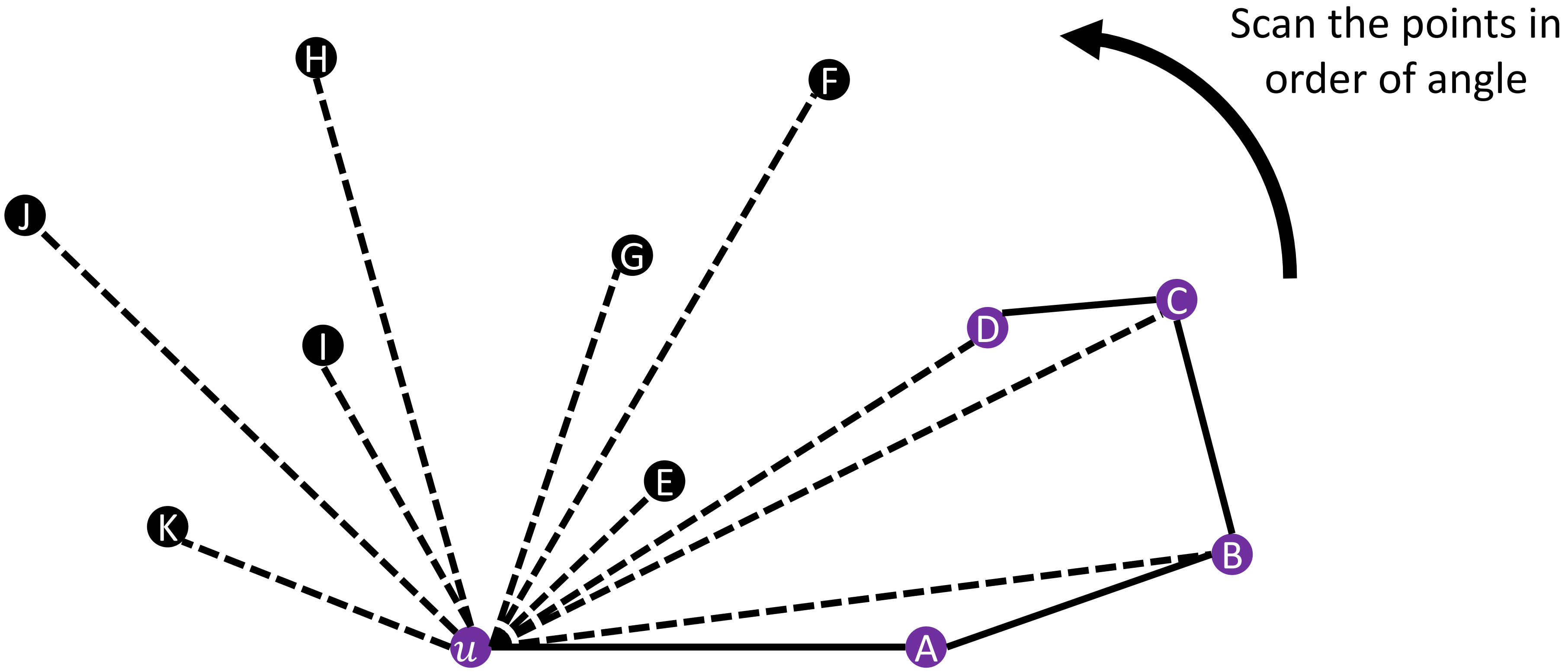




Idea: In order of angle, add points to the convex hull as long as it preserves convexity

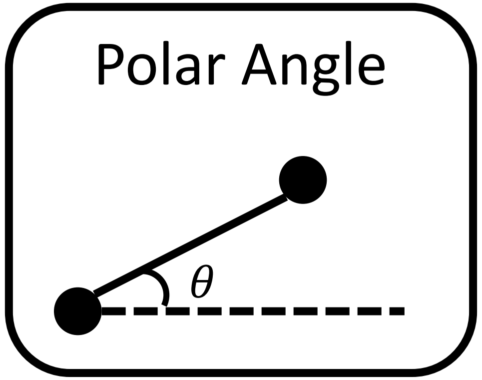
Graham's Algorithm

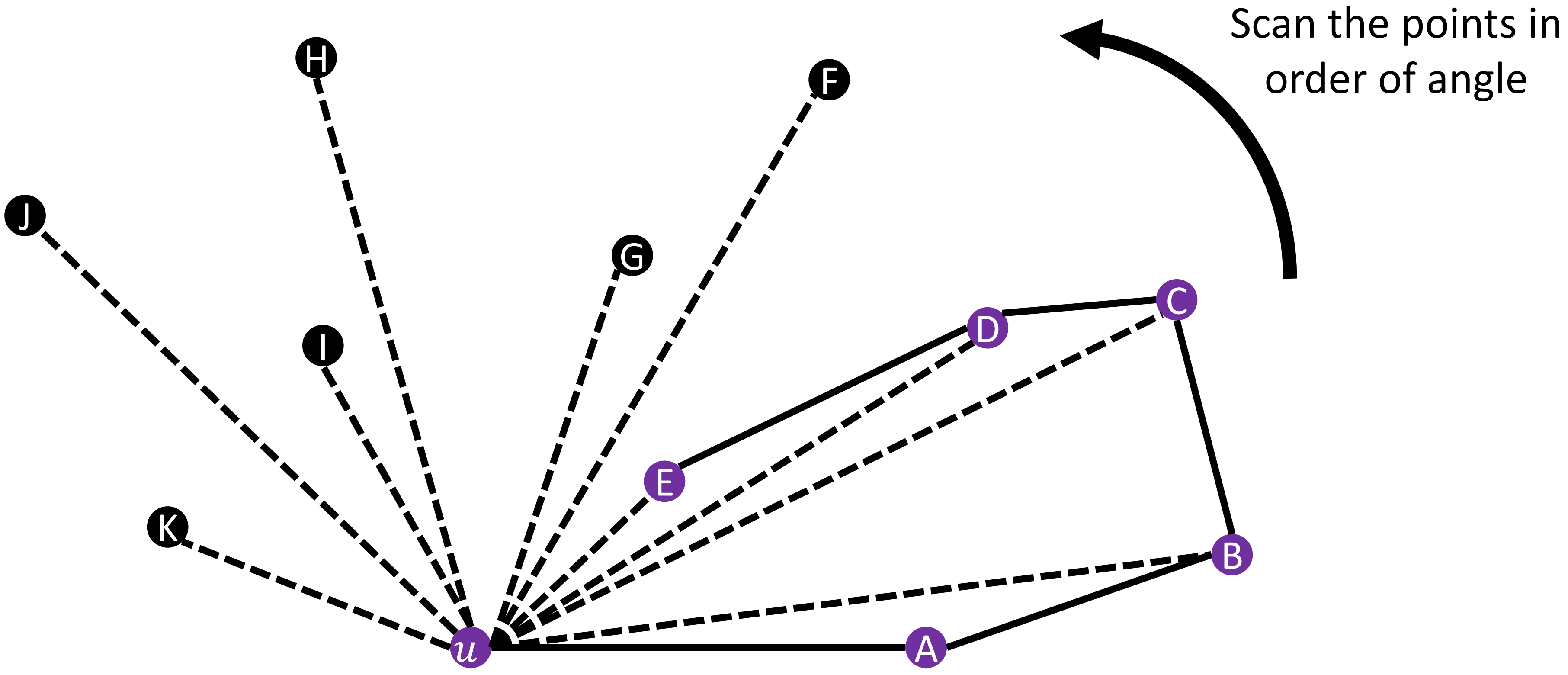




Idea: In order of angle, add points to the convex hull as long as it preserves convexity

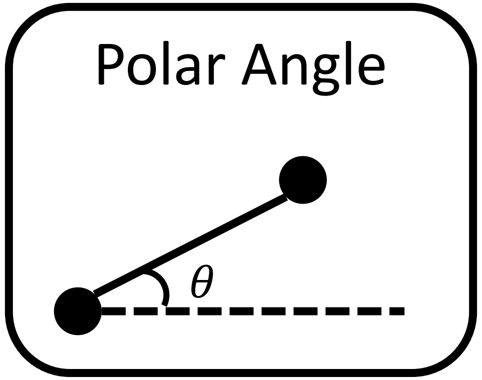
Graham's Algorithm



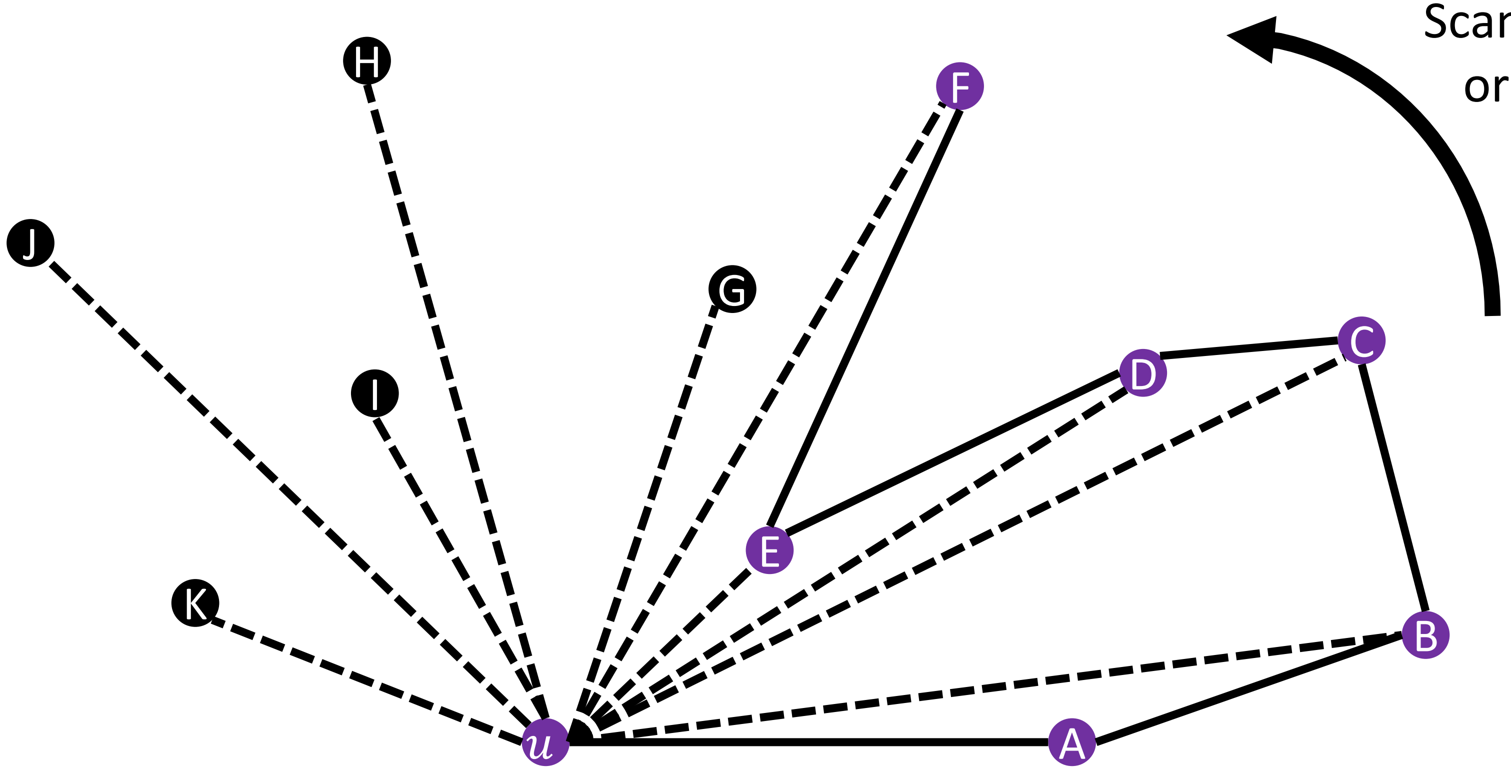


Idea: In order of angle, add points to the convex hull as long as it preserves convexity

Graham's Algorithm



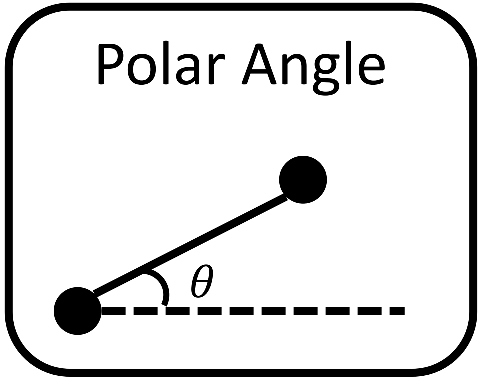
Not convex anymore!



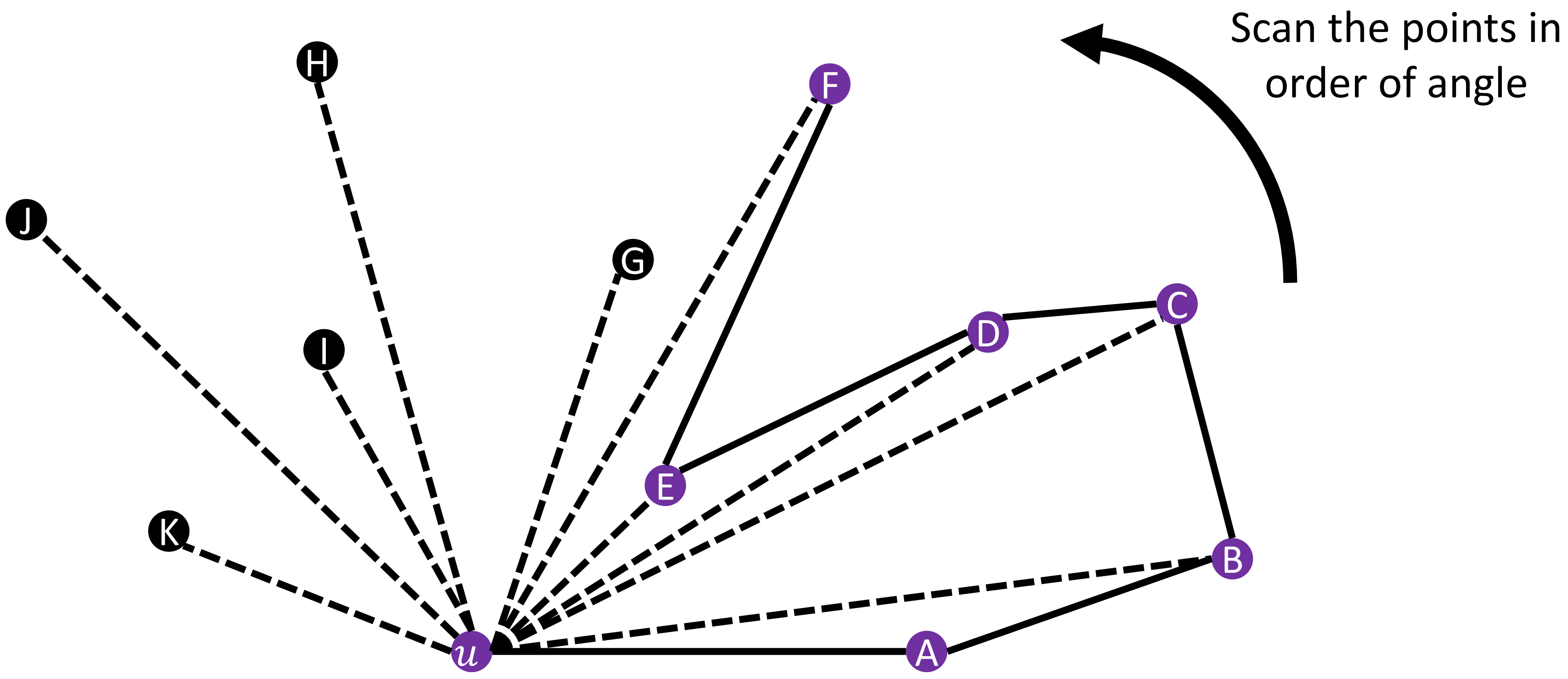
Scan the points in order of angle

Idea: In order of angle, add points to the convex hull as long as it <u>preserves</u> convexity

Graham's Algorithm

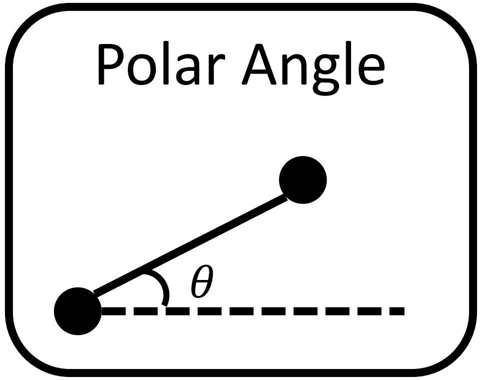


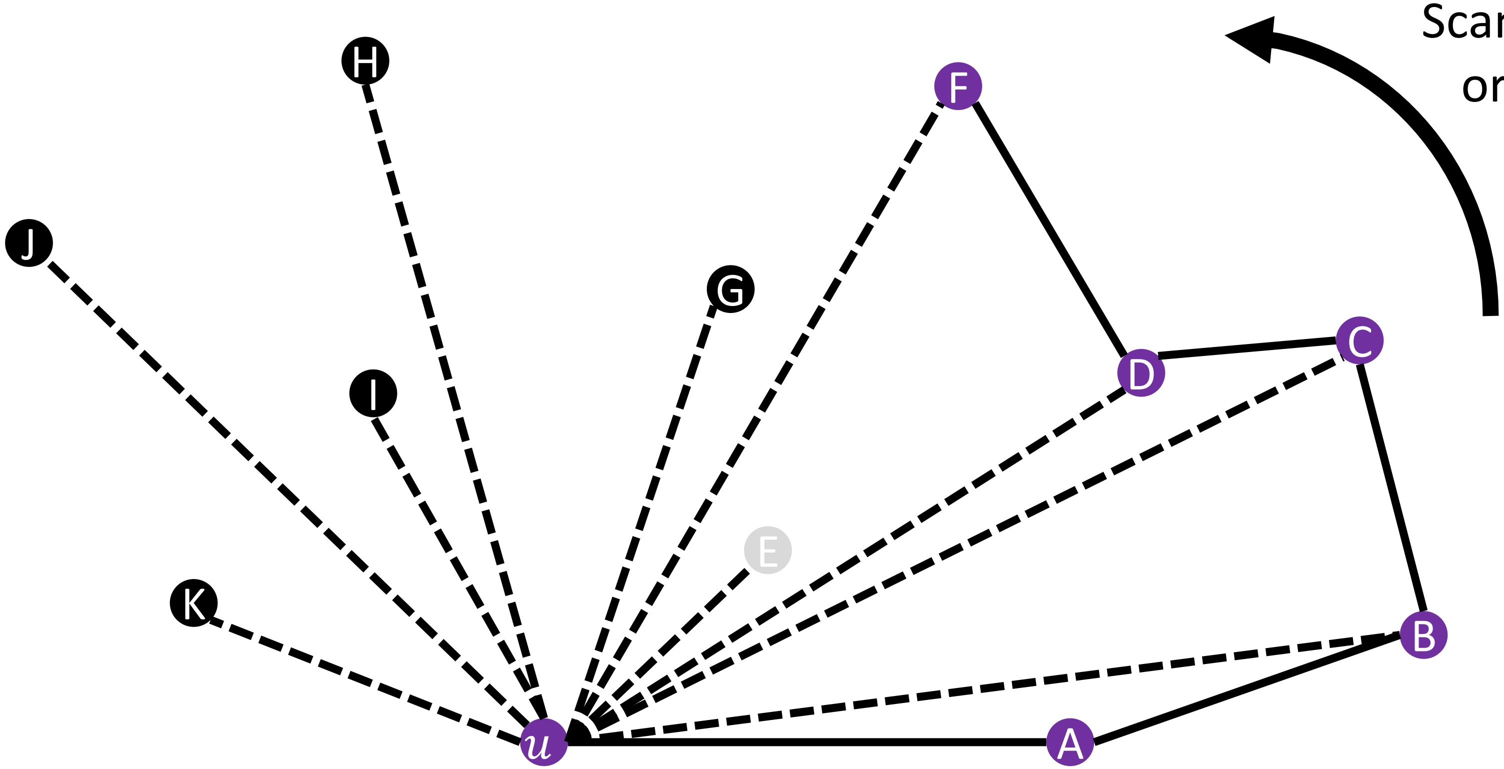
Not convex anymore!



Idea: Try extending the convex hull from the previous vertex if we are unable to extend from the current one

Graham's Algorithm

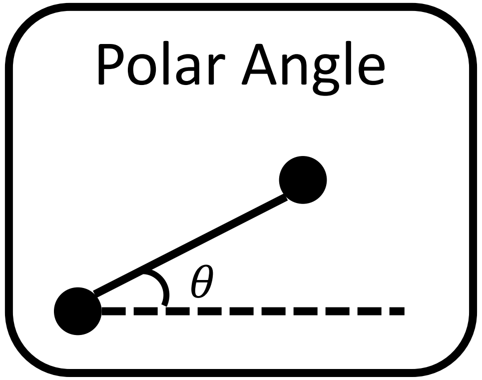




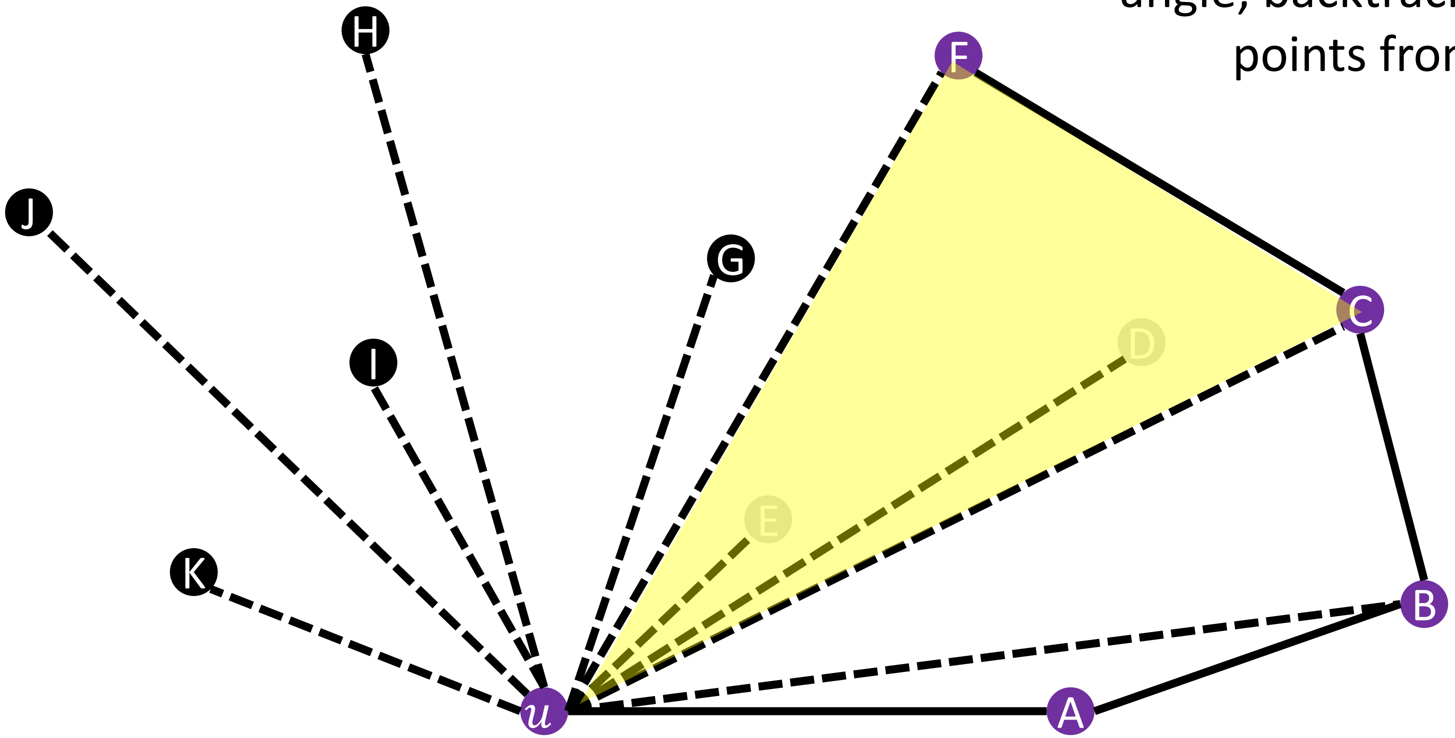
Scan the points in order of angle

Idea: Try extending the convex hull from the previous vertex if we are unable to extend from the current one

Graham’s Algorithm

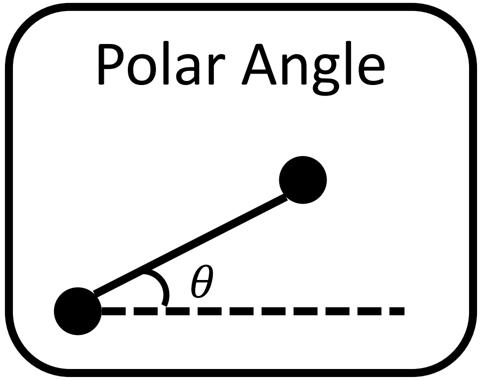


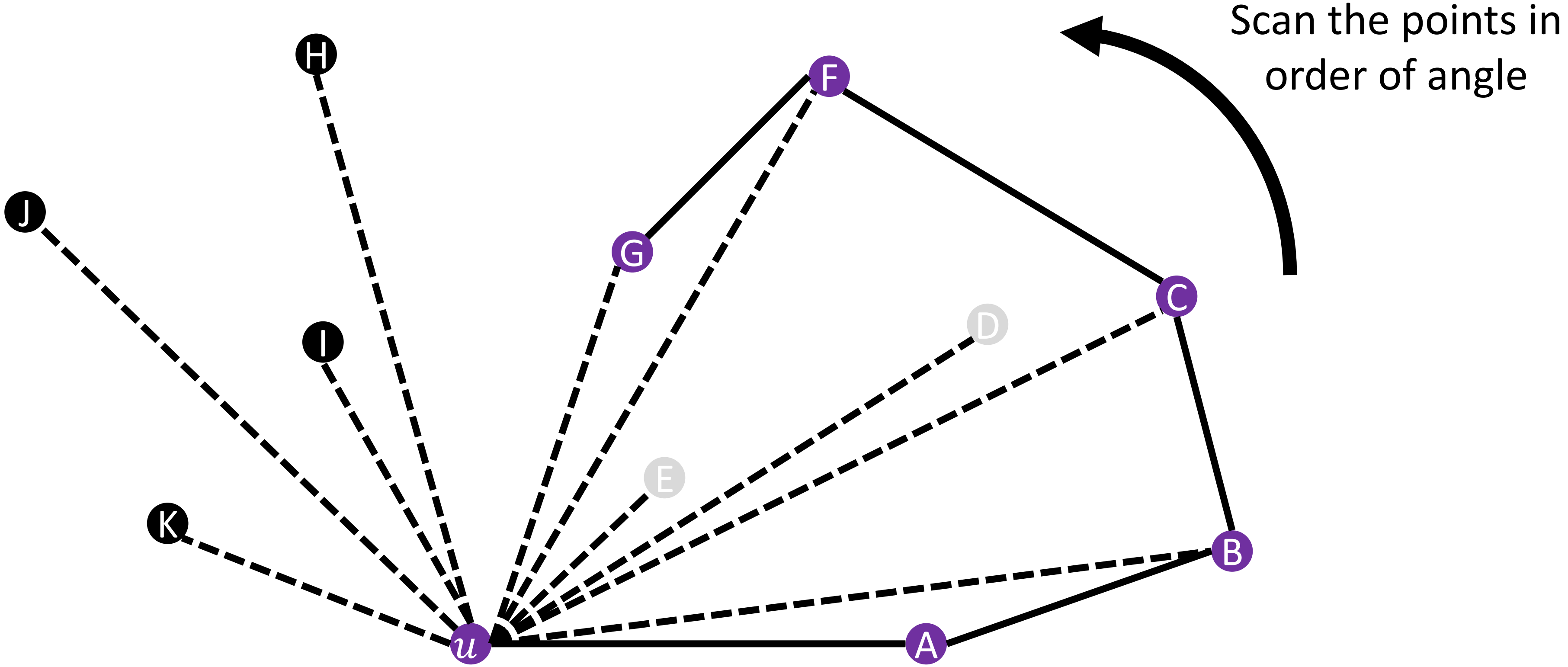
Observe: since points are sorted by angle, backtracking will <u>never</u> remove points from the convex hull



Idea: Try extending the convex hull from the previous vertex if we are unable to extend from the current one

Graham's Algorithm

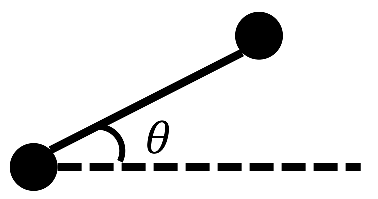




Idea: In order of angle, add points to the convex hull as long as it <u>preserves</u> convexity

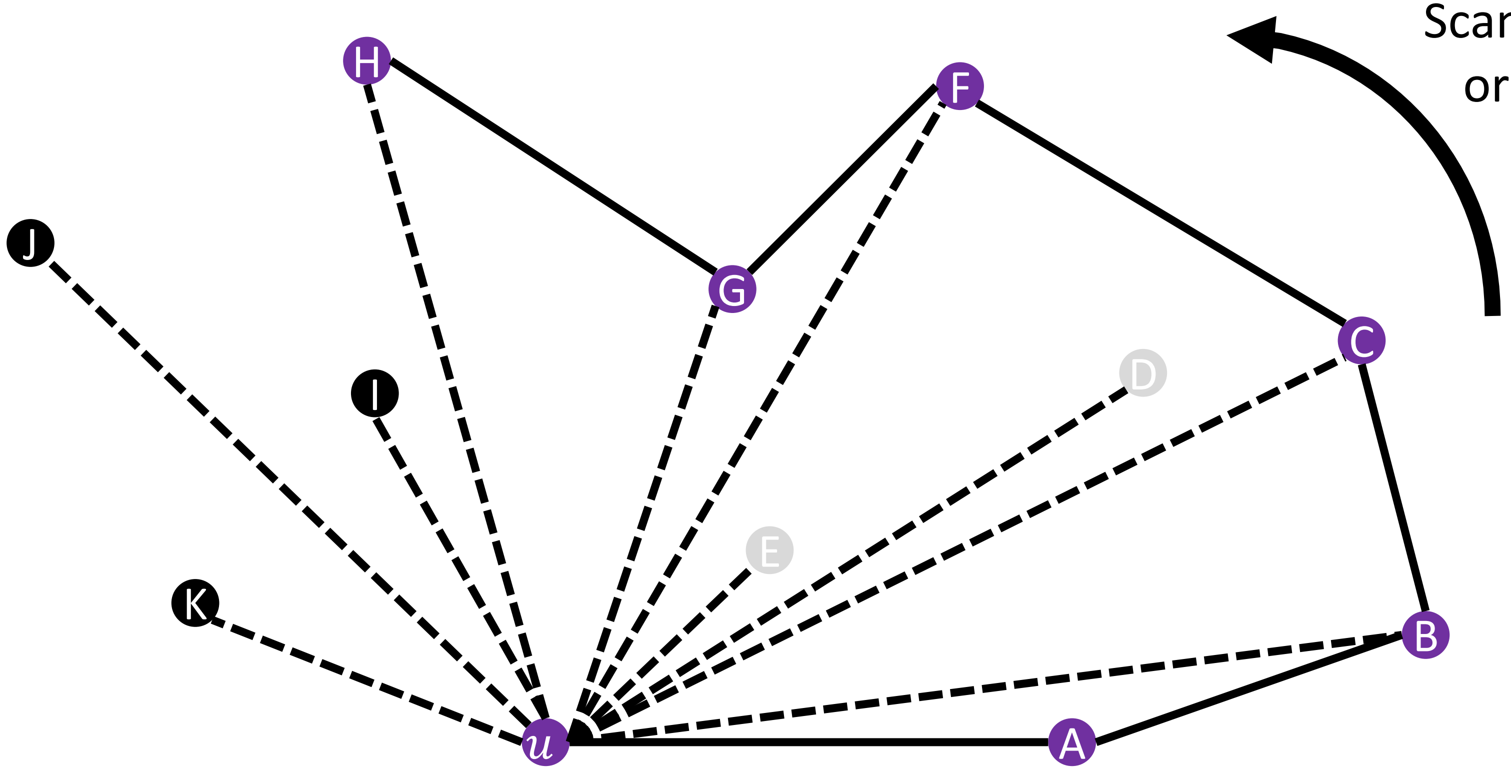
Graham's Algorithm

Polar Angle



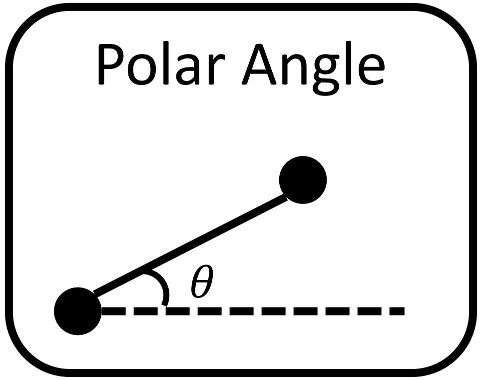
Not convex anymore!

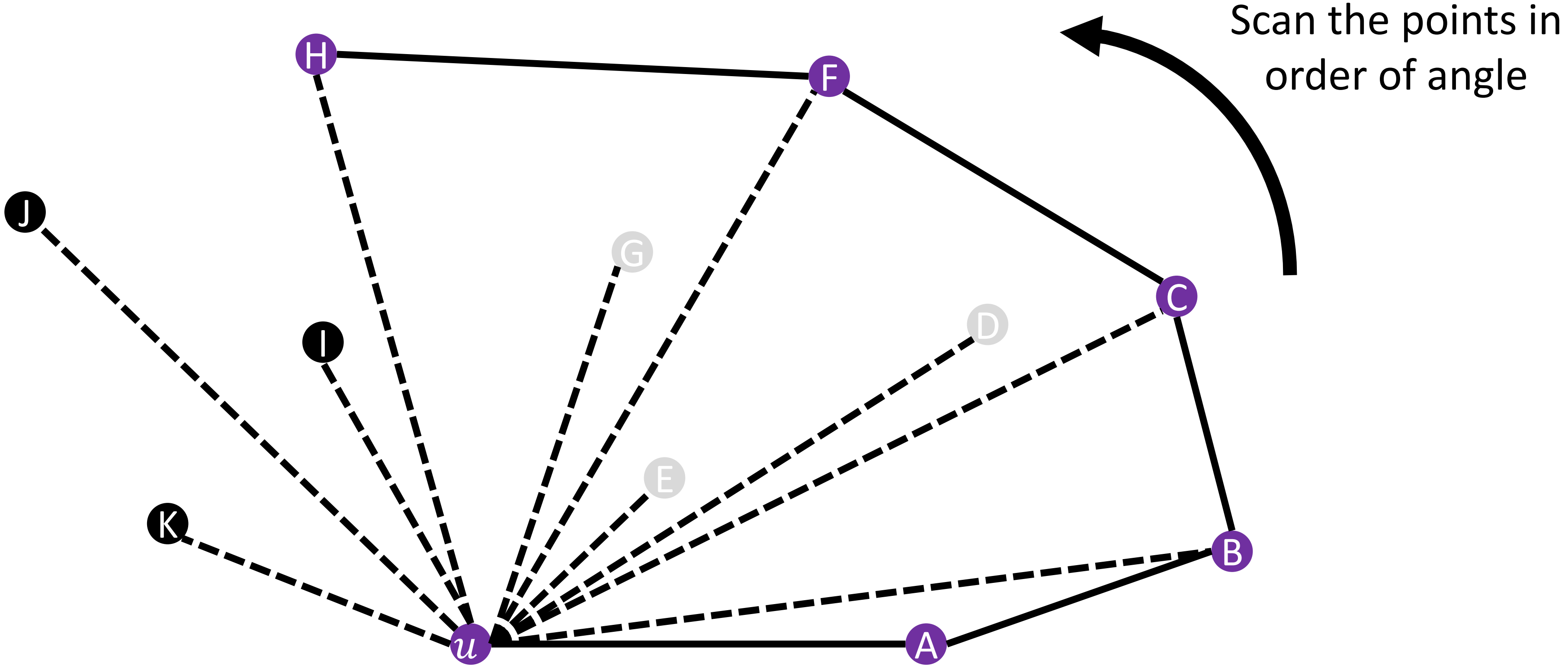
Scan the points in order of angle



Idea: In order of angle, add points to the convex hull as long as it preserves convexity

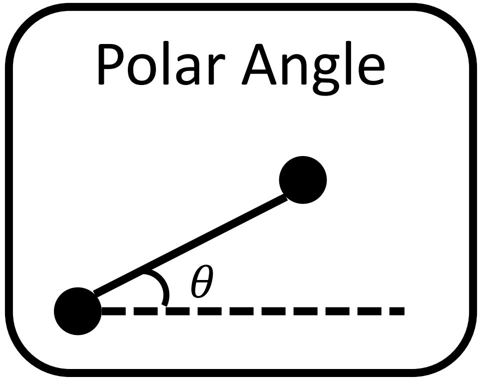
Graham's Algorithm

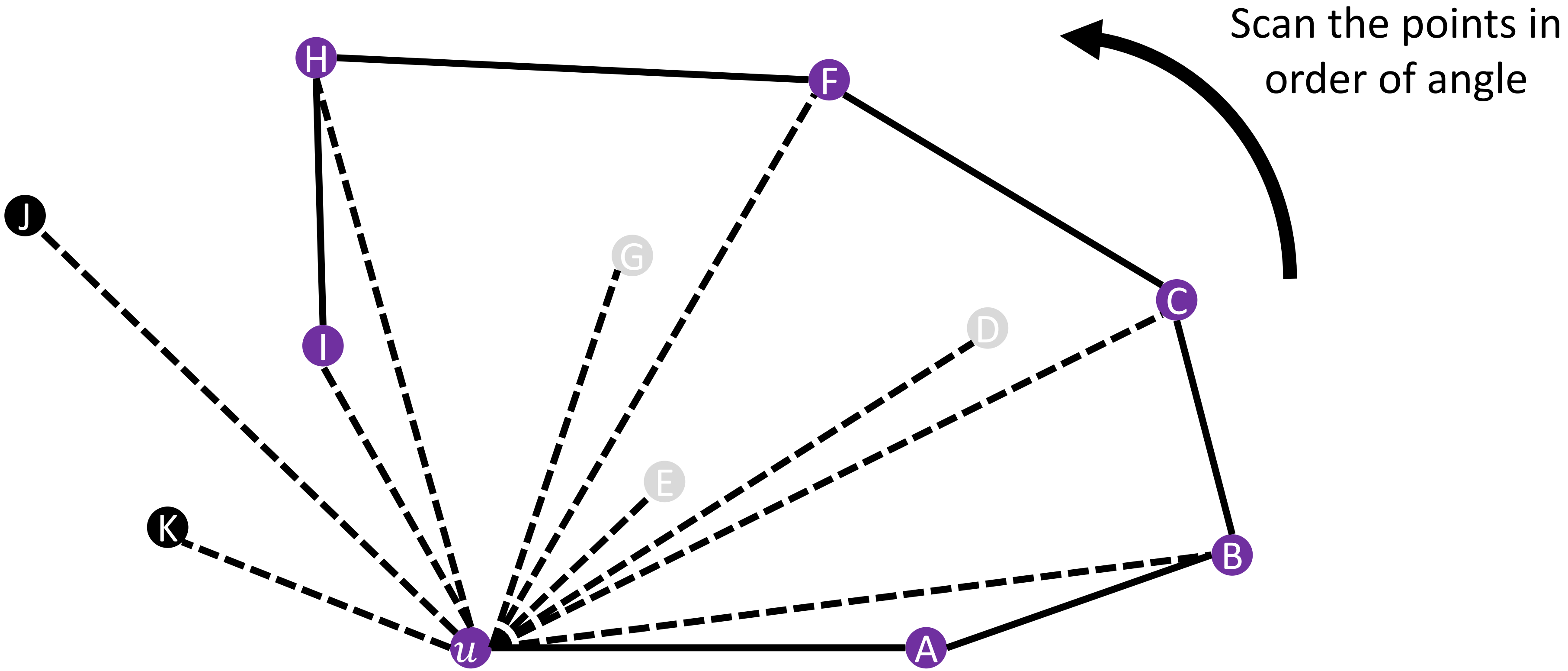




Idea: In order of angle, add points to the convex hull as long as it <u>preserves</u> convexity

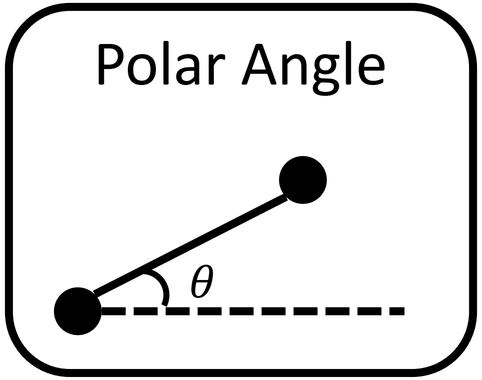
Graham's Algorithm





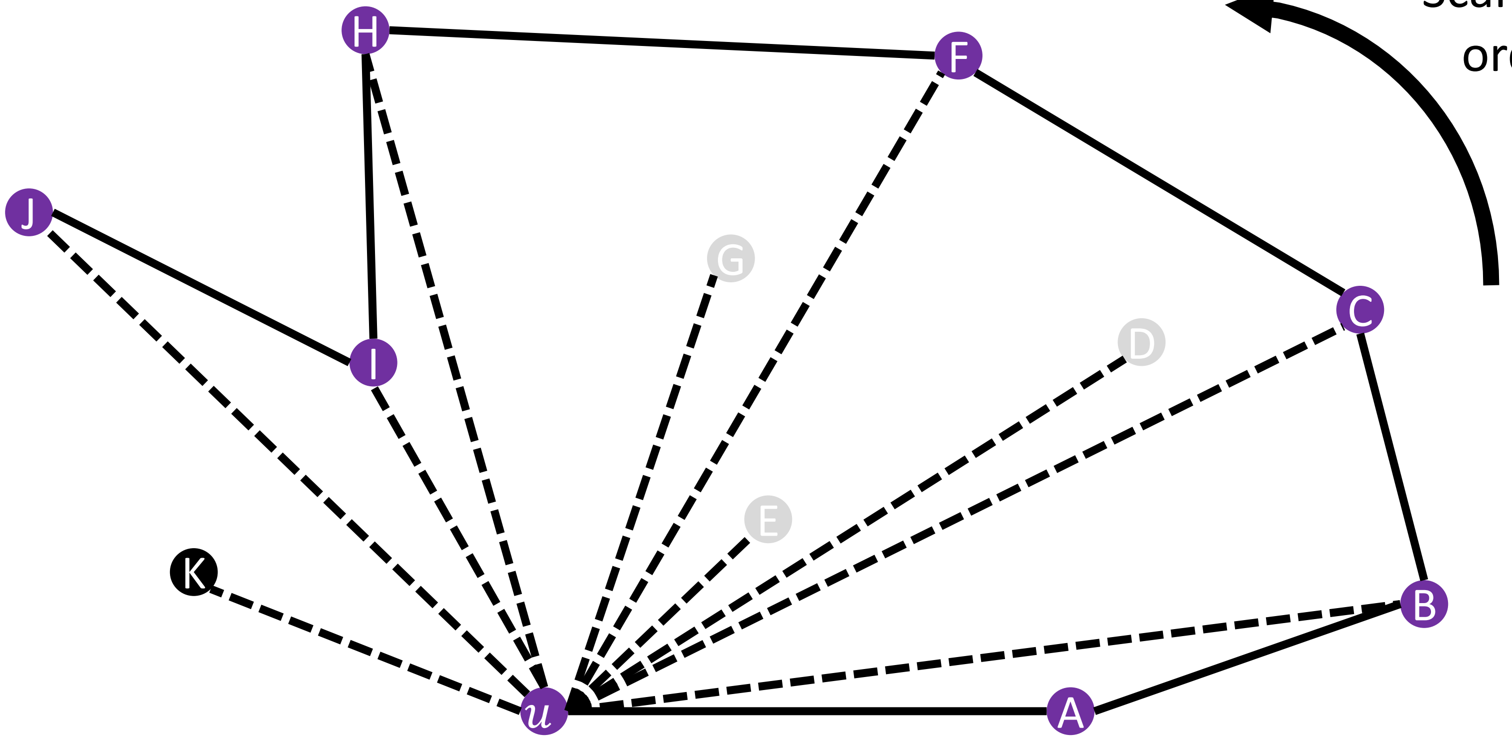
Idea: In order of angle, add points to the convex hull as long as it <u>preserves</u> convexity

Graham's Algorithm



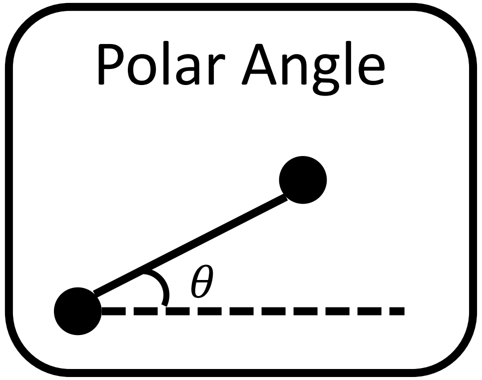
Not convex anymore!

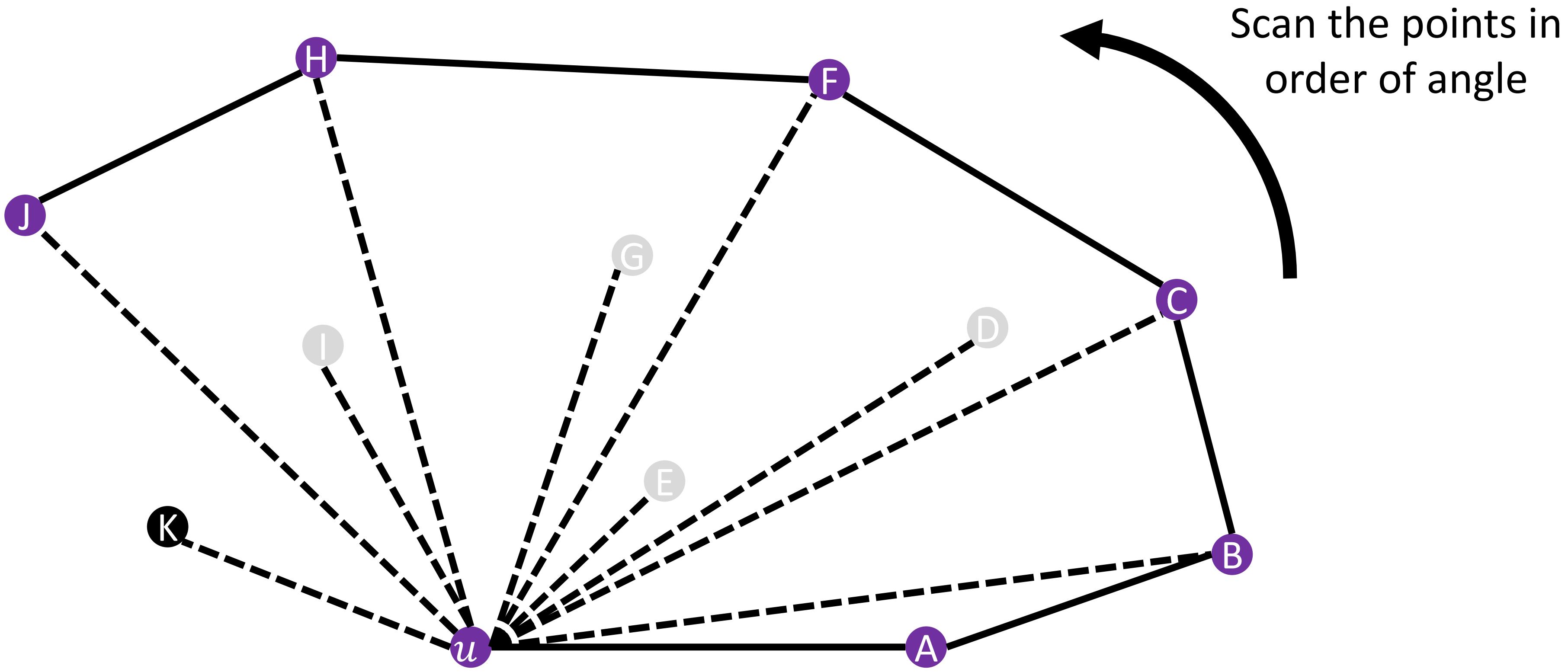
Scan the points in order of angle



Idea: In order of angle, add points to the convex hull as long as it <u>preserves</u> convexity

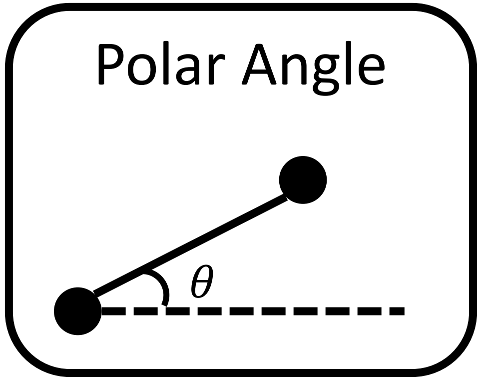
Graham's Algorithm

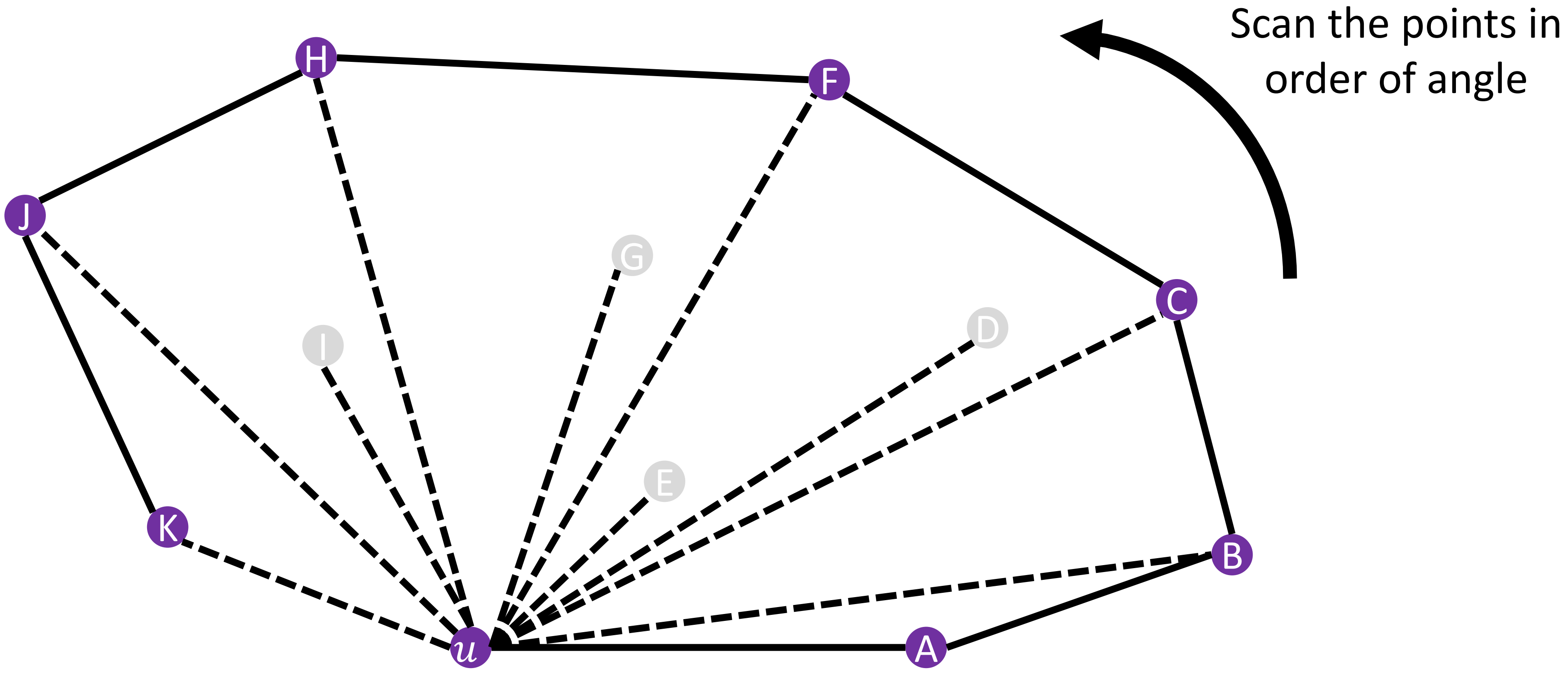




Idea: In order of angle, add points to the convex hull as long as it preserves convexity

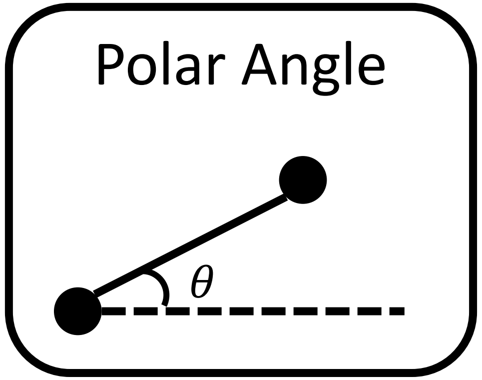
Graham's Algorithm

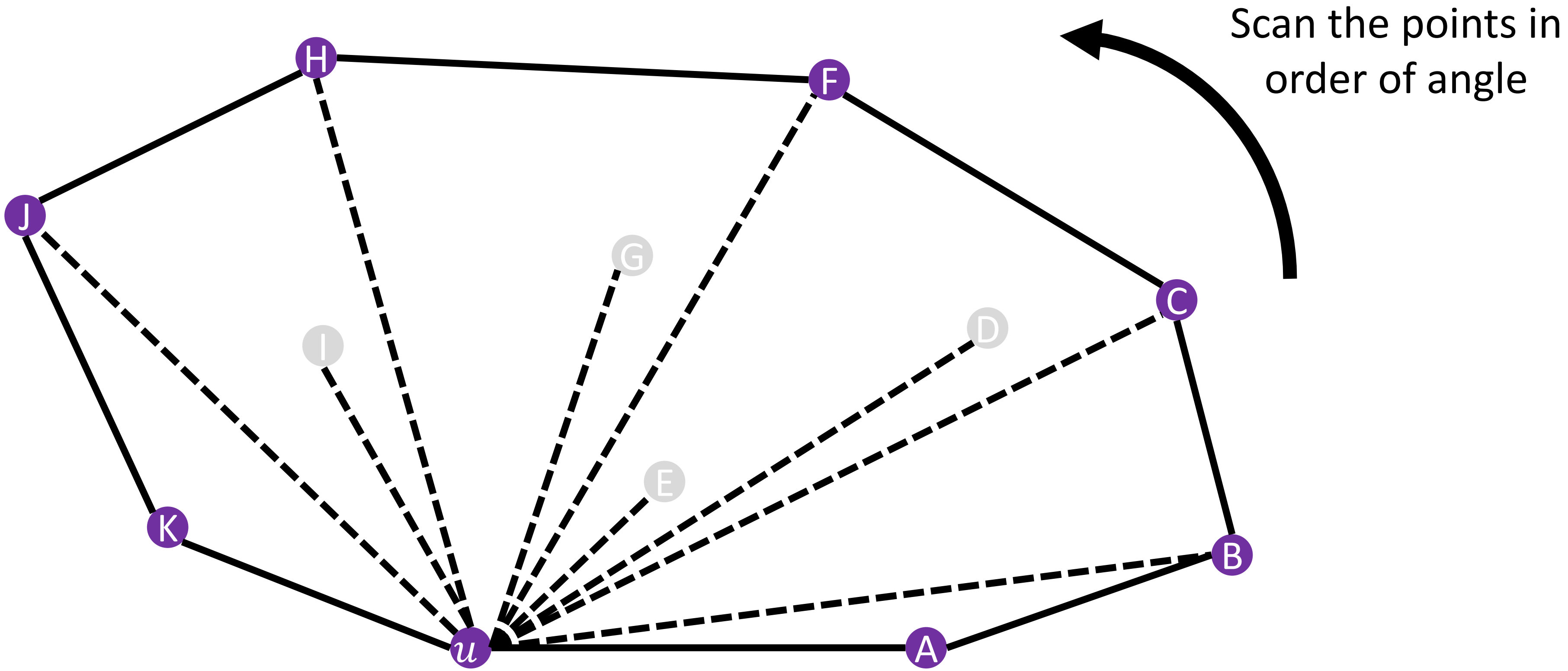




Idea: In order of angle, add points to the convex hull as long as it preserves convexity

Graham's Algorithm



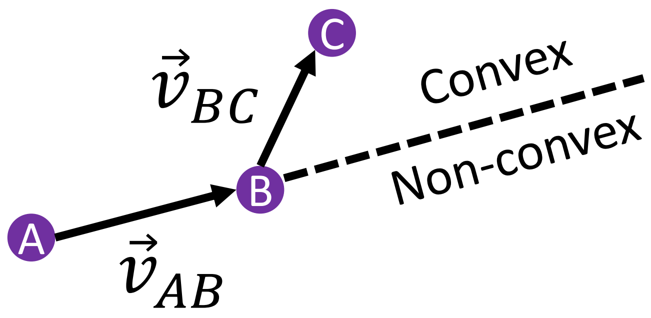


Idea: In order of angle, add points to the convex hull as long as it <u>preserves</u> convexity

Graham's Algorithm

1. Let p_1 be the point with the smallest y-coordinate (and smallest x-coordinate if multiple points have the same minimum-y coordinate)
2. Add p_1 to the convex hull C (represented as an ordered list)
3. Sort all of the points based on their angle relative to p_1
4. For each of the points p_i in sorted order:
    - Try adding p_i to the convex hull C
    - If adding p_i makes C non-convex, then remove the last component of C and repeat this check

How to implement this?



Imagine driving from $A \rightarrow B$

- $B \rightarrow C$ is convex if need to take a "left turn" to reach C
- $B \rightarrow C$ is non-convex if need to take a "non-left turn"

Decide "left turn" vs. "right turn" by computing the sign of the (vector) cross product between $\vec{v}_{AB}$ and $\vec{v}_{BC}$

Graham's Algorithm

1. Let p_1 be the point with the smallest y-coordinate (and smallest x-coordinate if multiple points have the same minimum-y coordinate)
2. Add p_1 to the convex hull C (represented as an ordered list)
3. Sort all of the points based on their angle relative to p_1
4. For each of the points p_i in sorted order:
   - Try adding p_i to the convex hull C
   - If adding p_i makes C non-convex, then remove the last component of C and repeat this check

Which data structure to use?

Need to be able to insert elements and remove in order of <u>most-recent</u> insertion

Can implement both operations in <u>constant-time</u> using a <u>stack</u>

Graham's Algorithm

1. Let p_1 be the point with the smallest y-coordinate (and smallest x-coordinate if multiple points have the same minimum-y coordinate)
2. Add p_1 to the convex hull C (represented as *a stack*)
3. Sort all of the points based on their angle relative to p_1
4. For each of the points p_i in sorted order:
   - Try adding p_i to the convex hull C
   - If adding p_i makes C non-convex, then remove the last component of C and repeat this check

Correctness?

See CLRS 33.3

Running Time of Graham's Algorithm

1. Let p_1 be the point with the smallest y-coordinate (and smallest x-coordinate if multiple points have the same minimum-y coordinate) $O(n)$
2. Add p_1 to the convex hull C (represented as a stack) $O(1)$
3. Sort all of the points based on their angle relative to p_1 $O(n \log n)$
4. For each of the points p_i in sorted order: $O(n)$
   - Try adding p_i to the convex hull C
   - If adding p_i makes C non-convex, then remove the last component of C and repeat this check

$O(n \log n)$

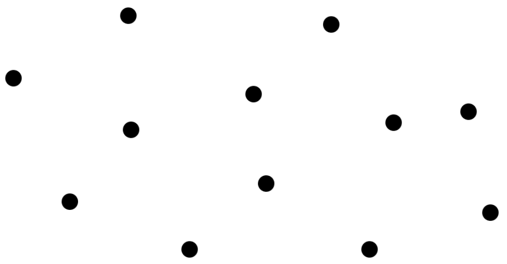
Running Time of Graham's Algorithm

1. Let p_1 be the point with the smallest y-coordinate (and smallest x-coordinate if multiple points have the same minimum-y coordinate) $O(n)$
2. Add p_1 to the convex hull C (represented as a stack) $O(1)$
3. Sort all of the points based on their angle relative to p_1 $O(n \log n)$
4. For each of the points p_i in sorted order: $O(n)$
   - Try adding p_i to the convex hull C
   - If adding p_i makes C non-convex, then remove the last component of C and repeat this check

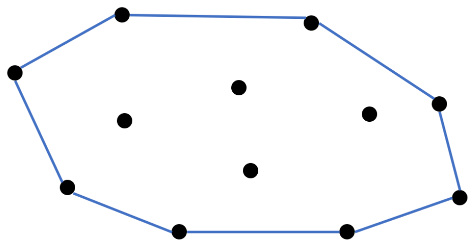
We have essentially <u>reduced</u> the problem of computing a convex hull to the problem of sorting! $O(n \log n)$

Convex Hull to Sorting Reduction

convex hull



convex hull



Map instances of problem ***A*** to instances of ***B***

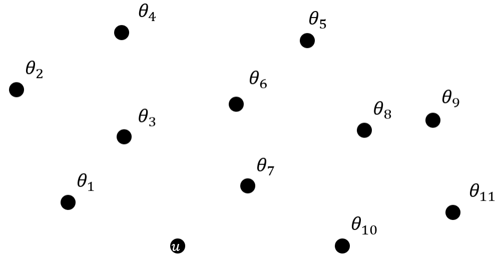
$O(n)$

Map solutions of problem ***B*** to solutions of ***A***

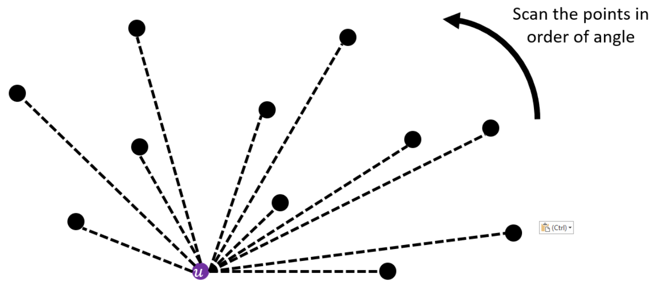
$O(n)$

Reduction

sorting



points sorted by angle



convex hull $\leq$ sorting

convex hull can be reduced to sorting in $O(n)$ time

Running Time of Graham's Algorithm

1. Let p_1 be the point with the smallest y-coordinate (and smallest x-coordinate if multiple points have the same minimum-y coordinate) — $O(n)$
2. Add p_1 to the convex hull C (represented as a stack) — $O(1)$
3. Sort all of the points based on their angle relative to p_1 — $O(n \log n)$
4. For each of the points p_i in sorted order: — $O(n)$
   - Try adding p_i to the convex hull C
   - If adding p_i makes C non-convex, then remove the last component of C and repeat this check

$O(n \log n)$

Running time of Graham's algorithm: same as best sorting algorithm

Can we do better (without going through sorting)?

Running Time of Graham's Algorithm

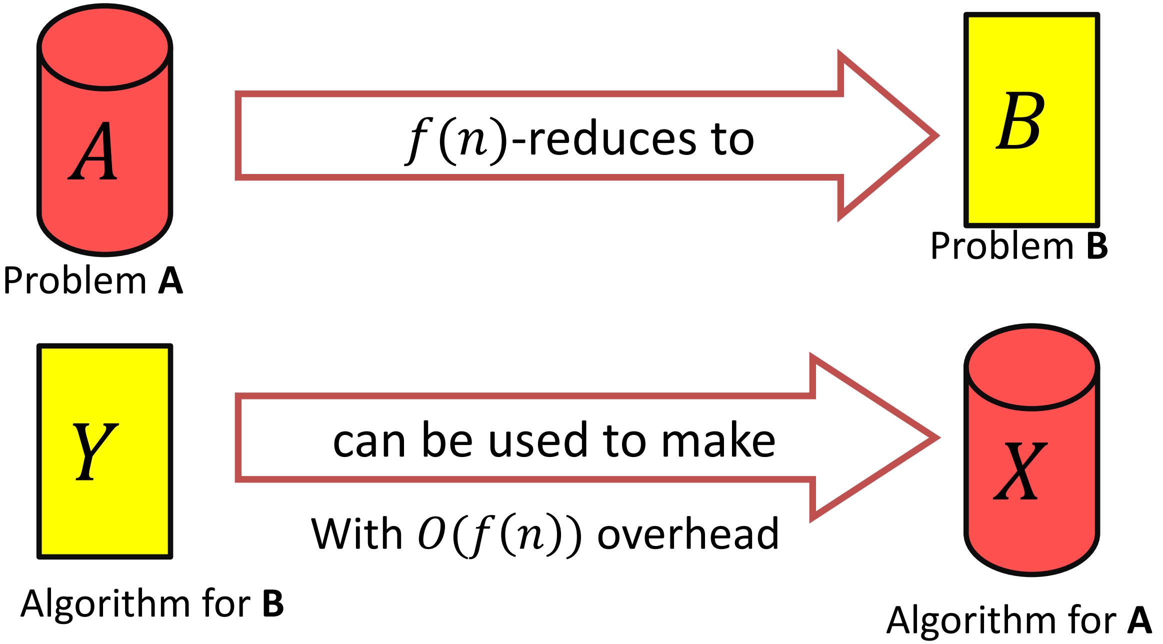
1. Let p_1 be the point with the smallest y-coordinate (and smallest x-coordinate if multiple points have the same minimum-y coordinate) $O(n)$
2. Add p_1 to the convex hull C (represented as a stack) $O(1)$
3. Sort all of the points based on their angle relative to p_1 $O(n \log n)$
4. For each of the points p_i in sorted order: $O(n)$
   - Try adding p_i to the convex hull C
   - If adding p_i makes C non-convex, then remove the last component of C and repeat this check

Trivial lower bound: $\Omega(n)$

ne as best sorting algorithm

Can we do better (without going through sorting)?

Worst Case Lower Bound Proofs



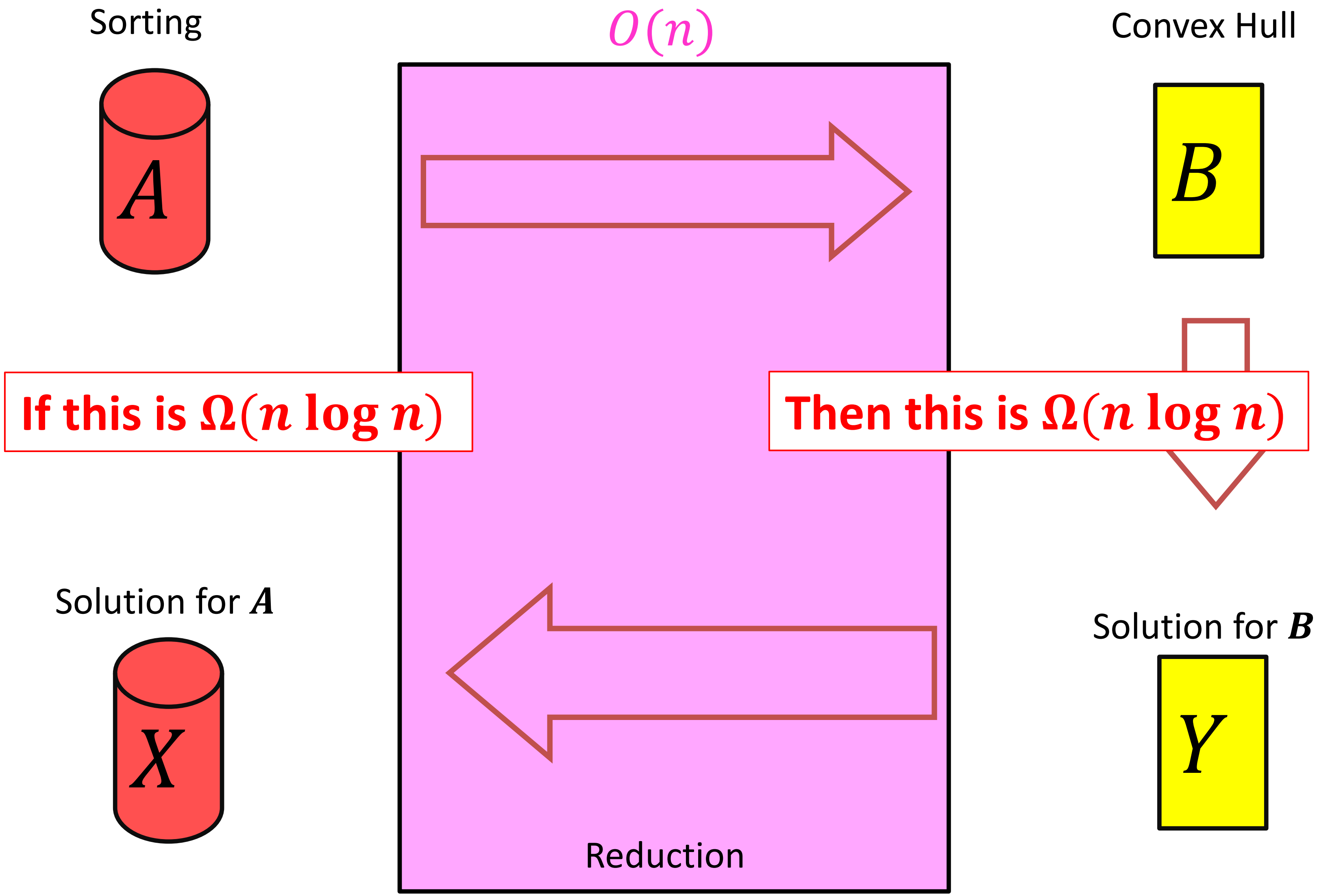
A is not a harder problem than B

$$A \leq B$$

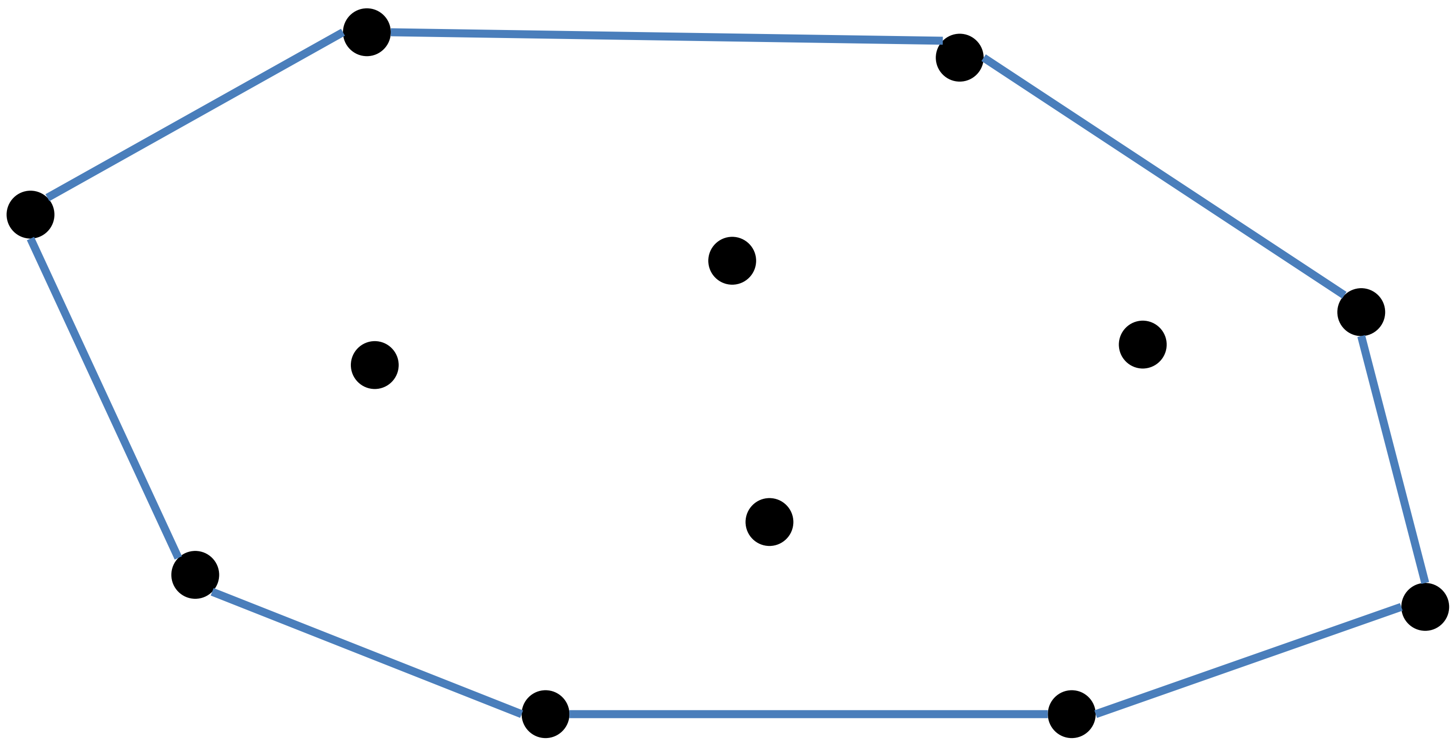
If we know that A **cannot** be solved in $O(f(n))$ time

If there is a $O(f(n))$ reduction from A to B, then B **cannot** be solved in $O(f(n))$ time

Sorting to Convex Hull Reduction

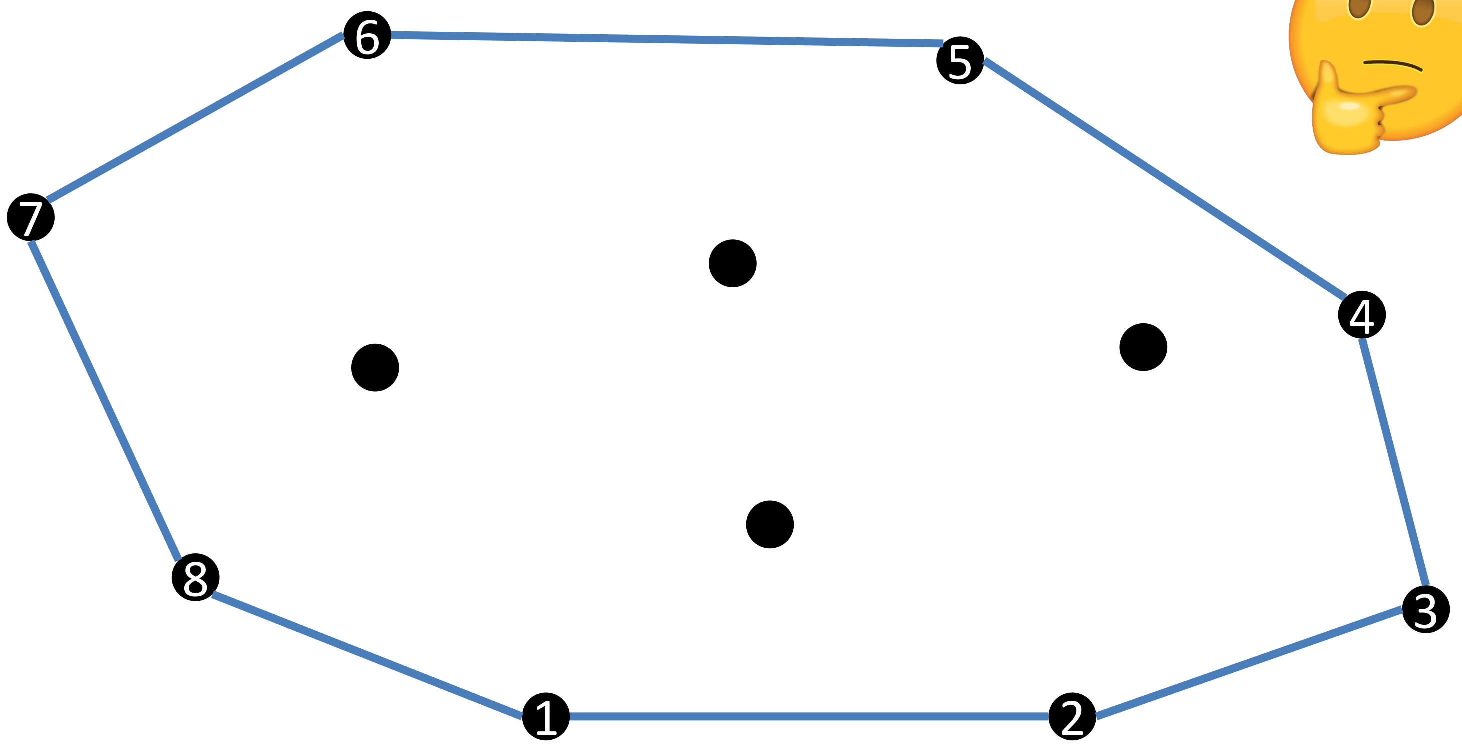


Sorting to Convex Hull Reduction



Observe: convex hull consists of a subset of points in a prescribed order

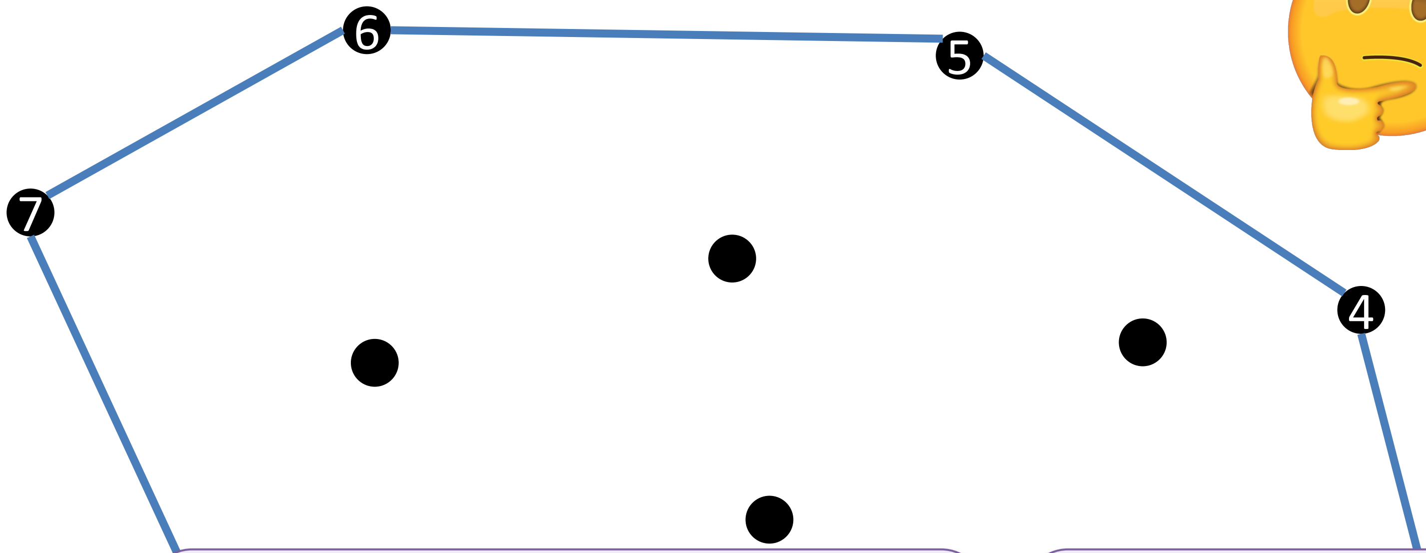
Sorting to Convex Hull Reduction



Can we use this to sort?

Observe: convex hull consists of a subset of points in a prescribed order

Sorting to Convex Hull Reduction



Can we use this to sort?

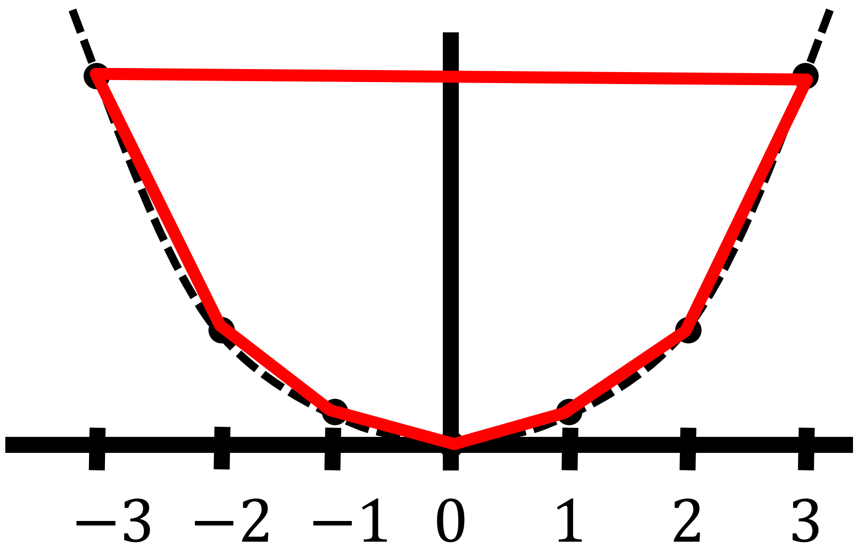
To get full sorted ordering, convex hull should contain all of the points (i.e., values in the set)

Want order of points in convex hull to be the order of elements in sorted order

Observe: convex hull consists of a subset of points in a prescribed order

Sorting to Convex Hull Reduction

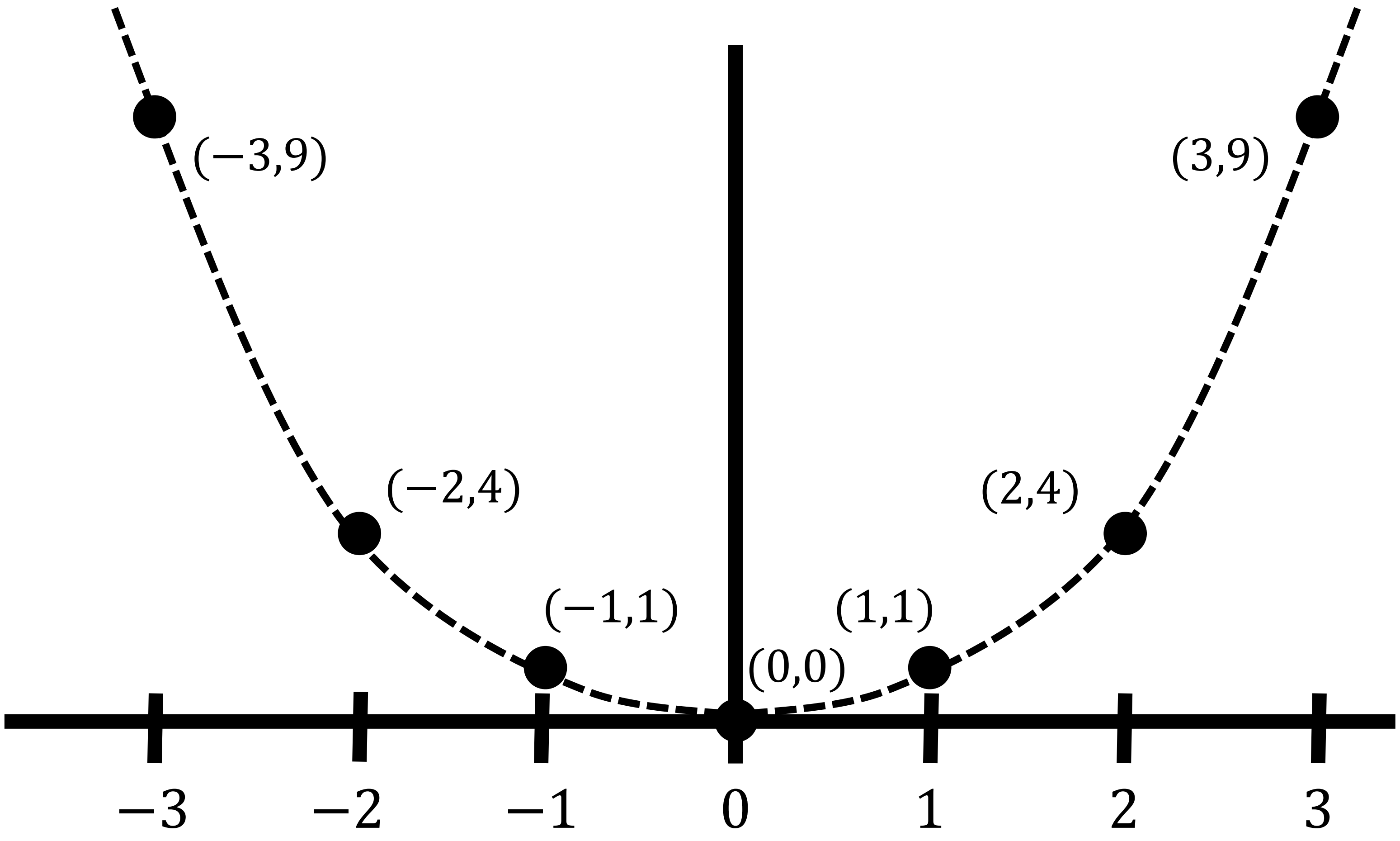
- **Goal:** need a way to map list of (numeric) values onto a convex hull instance
  - Given: -2 1 -3 0 2 3 -1
  - Create some convex hull instance where all points on the convex hull



How do we map values to points?

Sorting to Convex Hull Reduction

Given: $-2 \quad 1 \quad -3 \quad 0 \quad 2 \quad 3 \quad -1$



$$x \quad \Rightarrow \quad (x, x^2)$$

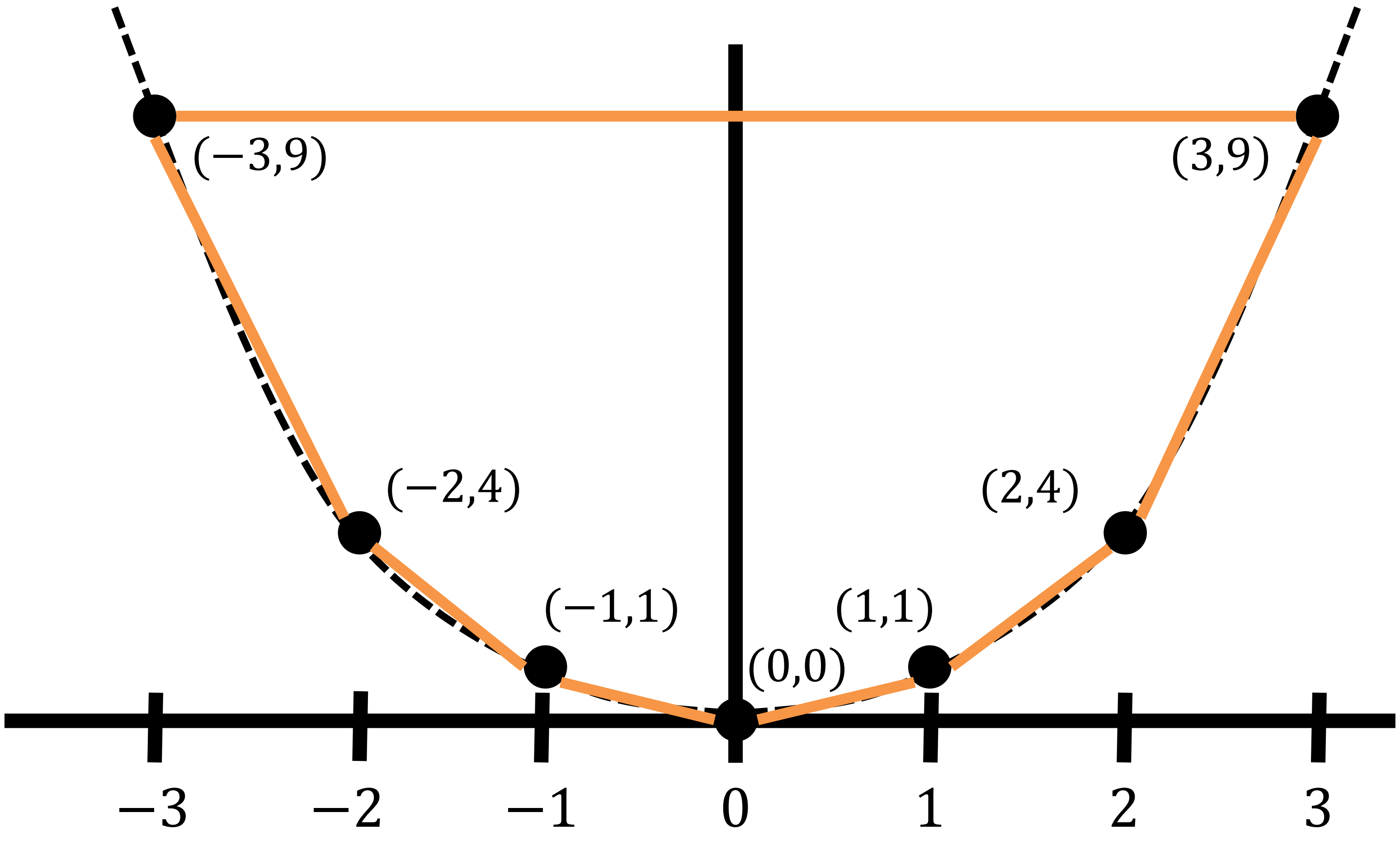
sorting CH

Creates a parabola!

Sorting to Convex Hull Reduction

Given: $-2 \quad 1 \quad -3 \quad 0 \quad 2 \quad 3 \quad -1$

Claim: order of elements in convex hull coincide with elements in sorted order



Sorting to Convex Hull Reduction

- Reduction Construction
  - Convert each element to a 2D point, $x \Rightarrow (x, x^2)$ $O(n)$
  - *Run Convex Hull algorithm*
  - Find minimum x-coordinate in convex hull points $O(n)$
  - List convex hull points' x-coordinate in prescribed order $O(n)$

Reduction cost: $O(n)$

Convex Hull to Sorting Reduction

sorting

$-2 \quad 1 \quad -3 \quad 0 \quad 2 \quad 3 \quad -1$

Create 2D points

$O(n)$

Find minimum x

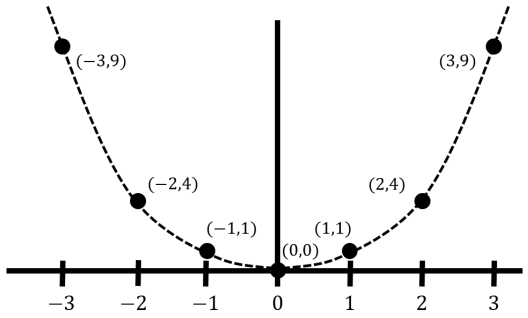
Read CH solution in order

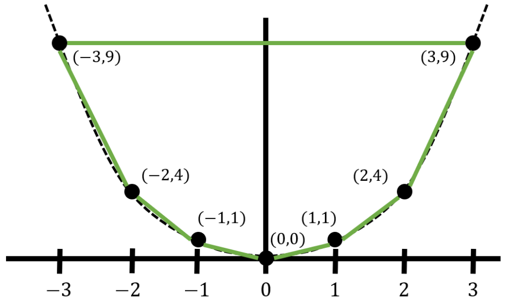
$O(n)$

Reduction

$-3 \quad -2 \quad -1 \quad 0 \quad 1 \quad 2 \quad 3$

convex hull





sorting numeric values $\leq$ convex hull

sorting numeric values can be reduced to convex hull in $O(n)$ time

Convex Hull to Sorting Reduction

sorting

$-2 \quad 1 \quad -3 \quad 0 \quad 2 \quad 3 \quad -1$

Create 2D points

$O(n)$

If this is $\Omega(n \log n)$

Then this is $\Omega(n \log n)$

Find minimum x

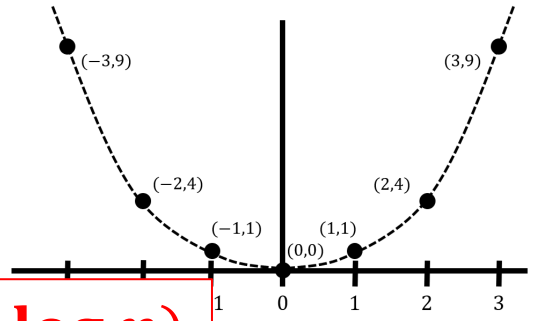
Read CH solution in order

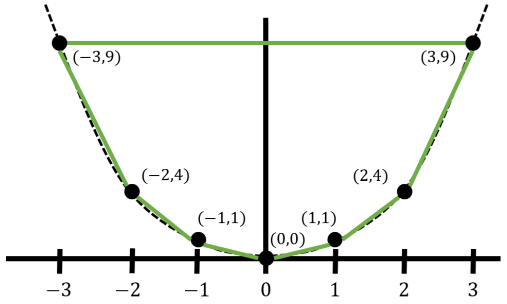
$O(n)$

$-3 \quad -2 \quad -1 \quad 0 \quad 1 \quad 2 \quad 3$

Reduction

convex hull

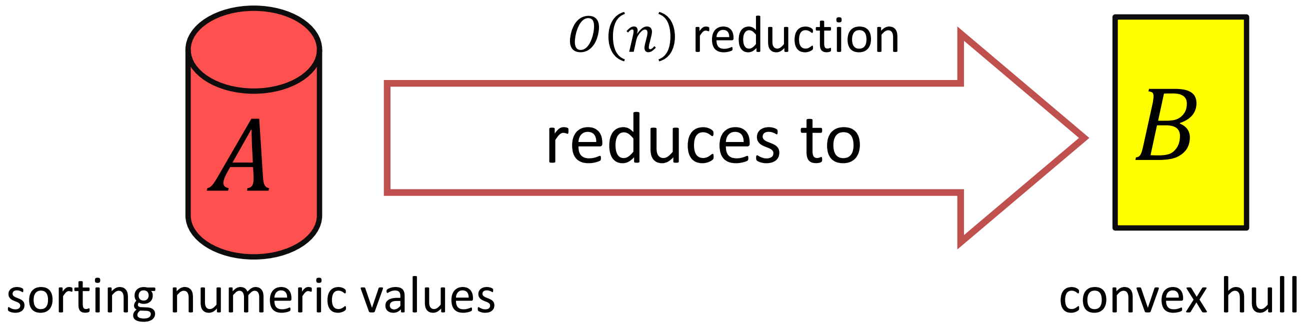




sorting numeric values $\leq$ convex hull

sorting numeric values can be reduced to convex hull in $O(n)$ time

Lower Bound for Convex Hull



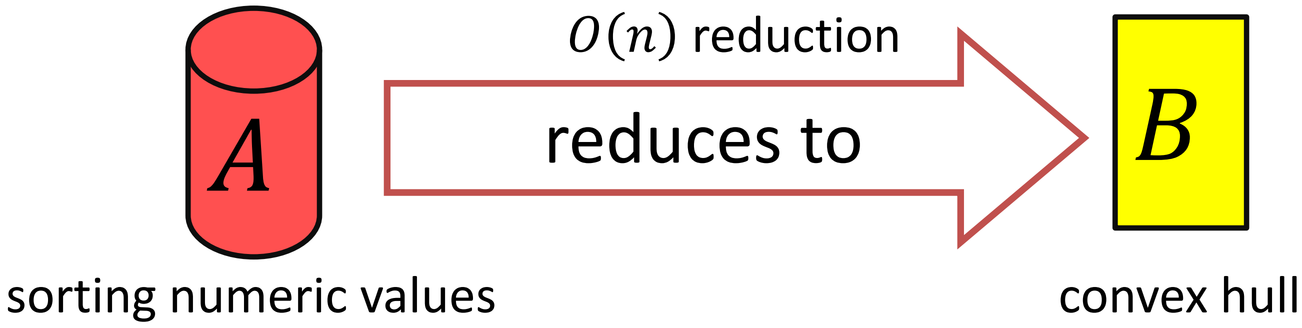
Conclusion: a lower bound for sorting translates into one for convex hull

Our lower bound for sorting: $\Omega(n \log n)$ for comparison-based sorts

Our reduction is not a comparison sort algorithm

$\Omega(n \log n)$ lower bound for sorting also holds in an "algebraic decision tree model"
(i.e., decisions can be an algebraic function of inputs)
Implies $\Omega(n \log n)$ lower bound for computing convex hull in this model

Lower Bound for Convex Hull



Conclusion: a lower bound for sorting translates into one for convex hull

Our lower bound for sorting: $\Omega(n \log n)$ for comparison-based sorts

Our reduction is not a

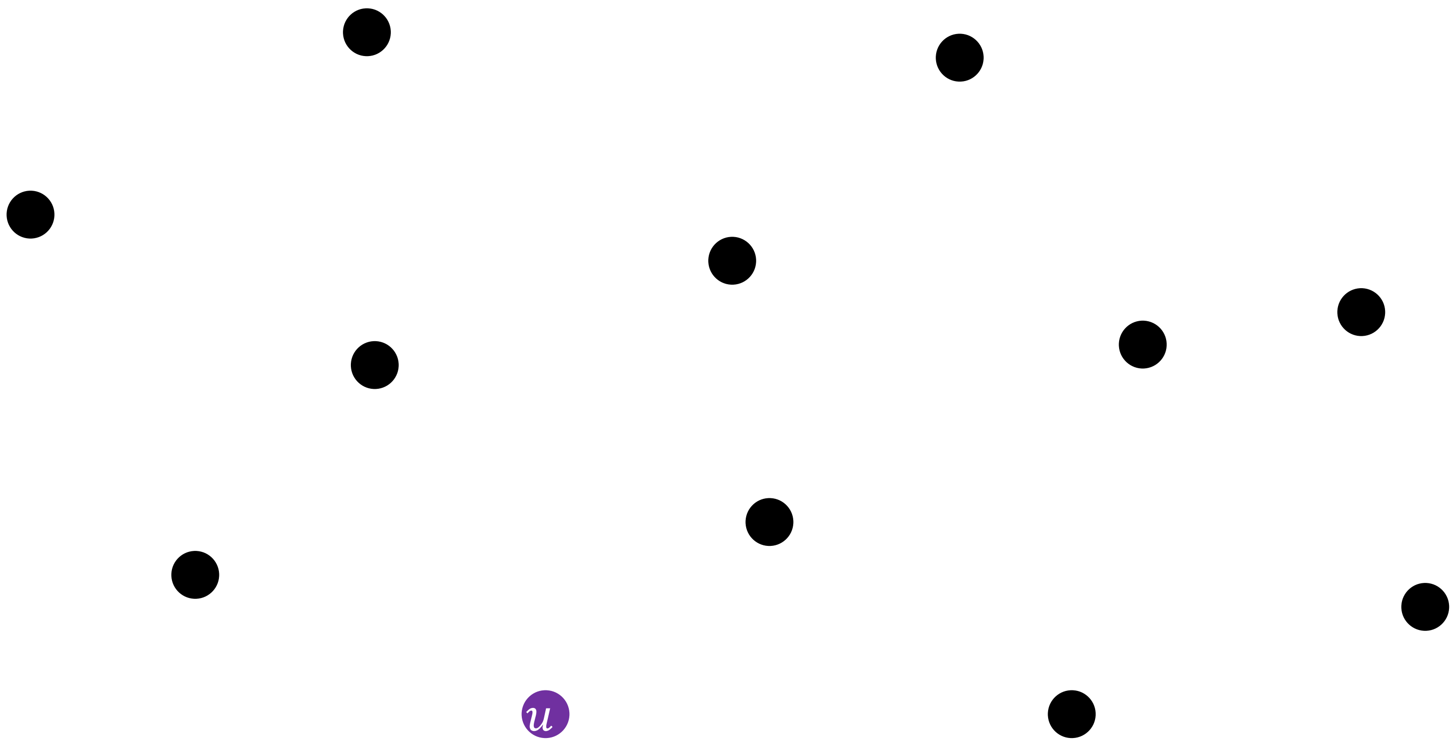
In fact, this lower bound holds even for algorithms that just identify the set of points on the convex hull (and not necessarily their order)!

$\Omega(n \log n)$ lower bound for s e model"

(i.e., decisions can be

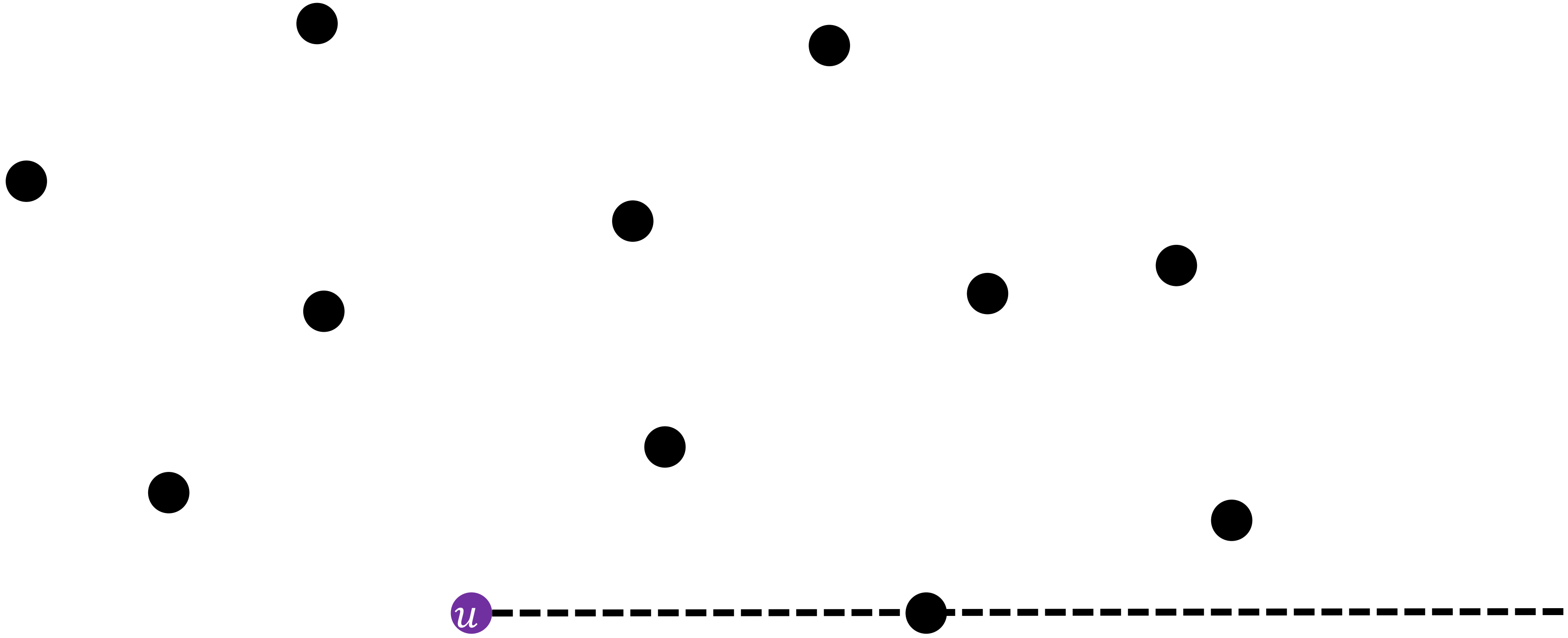
Implies $\Omega(n \log n)$ lower bound for computing convex hull in this model

Jarvis' Algorithm (Gift Wrapping Method)



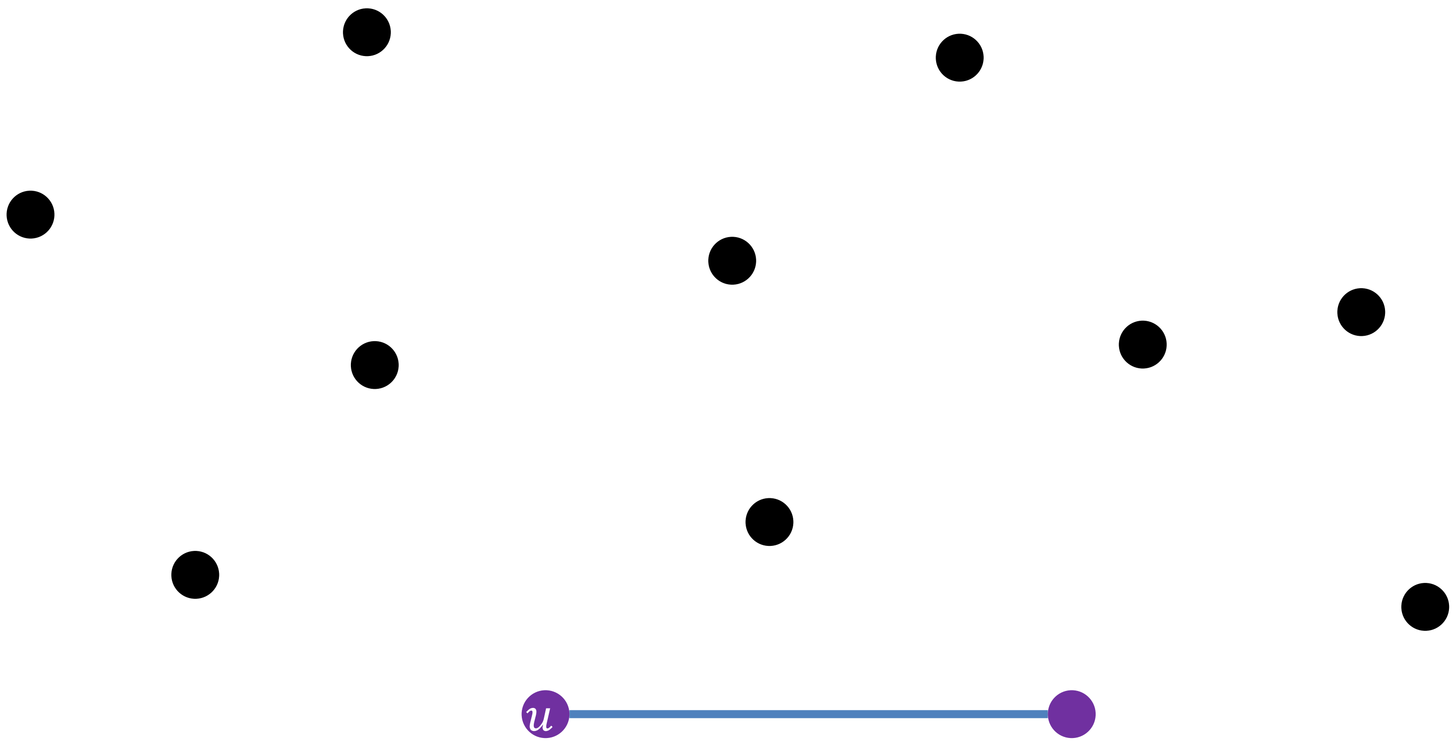
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



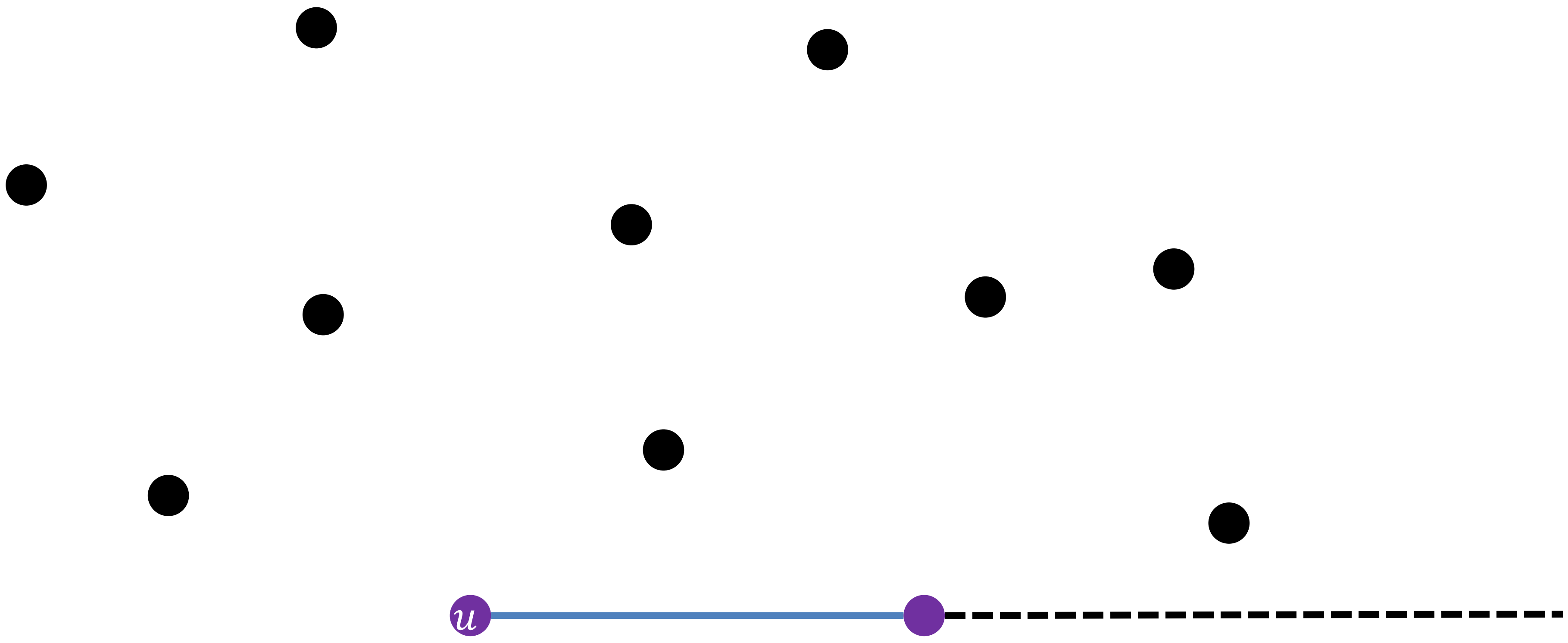
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



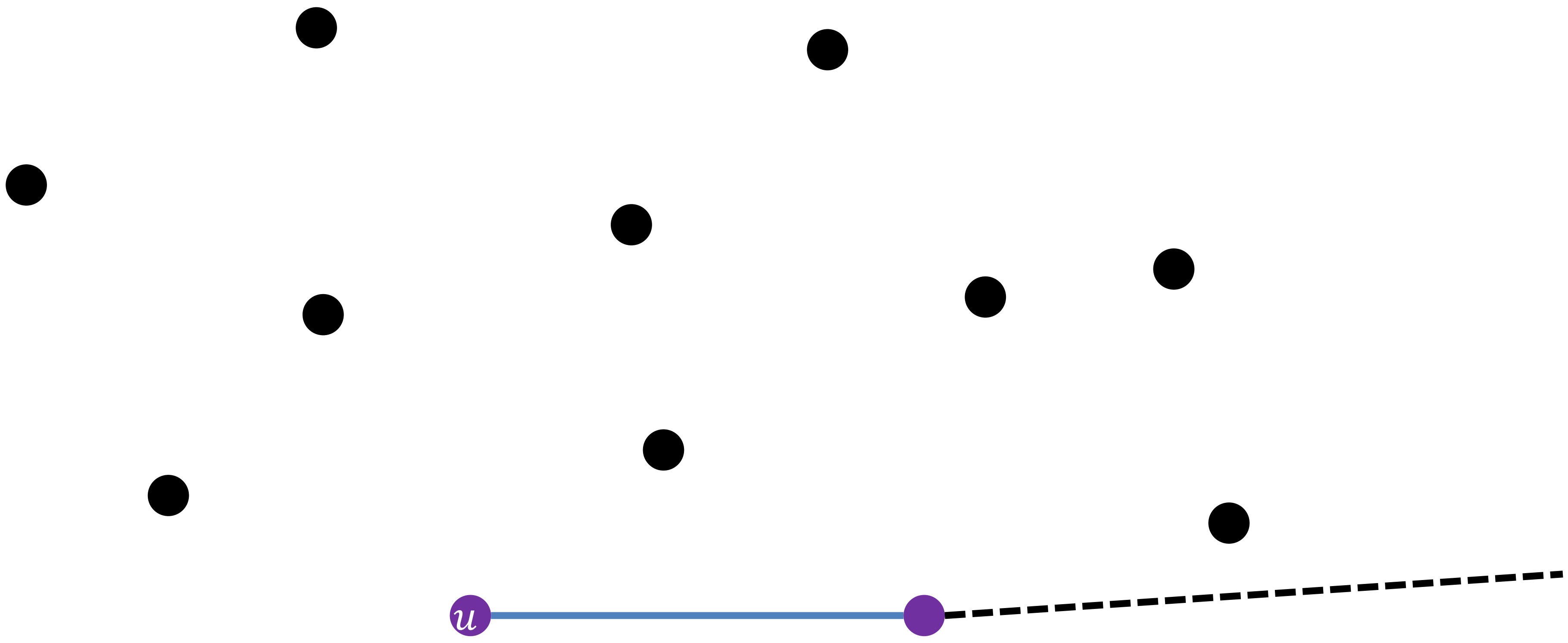
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



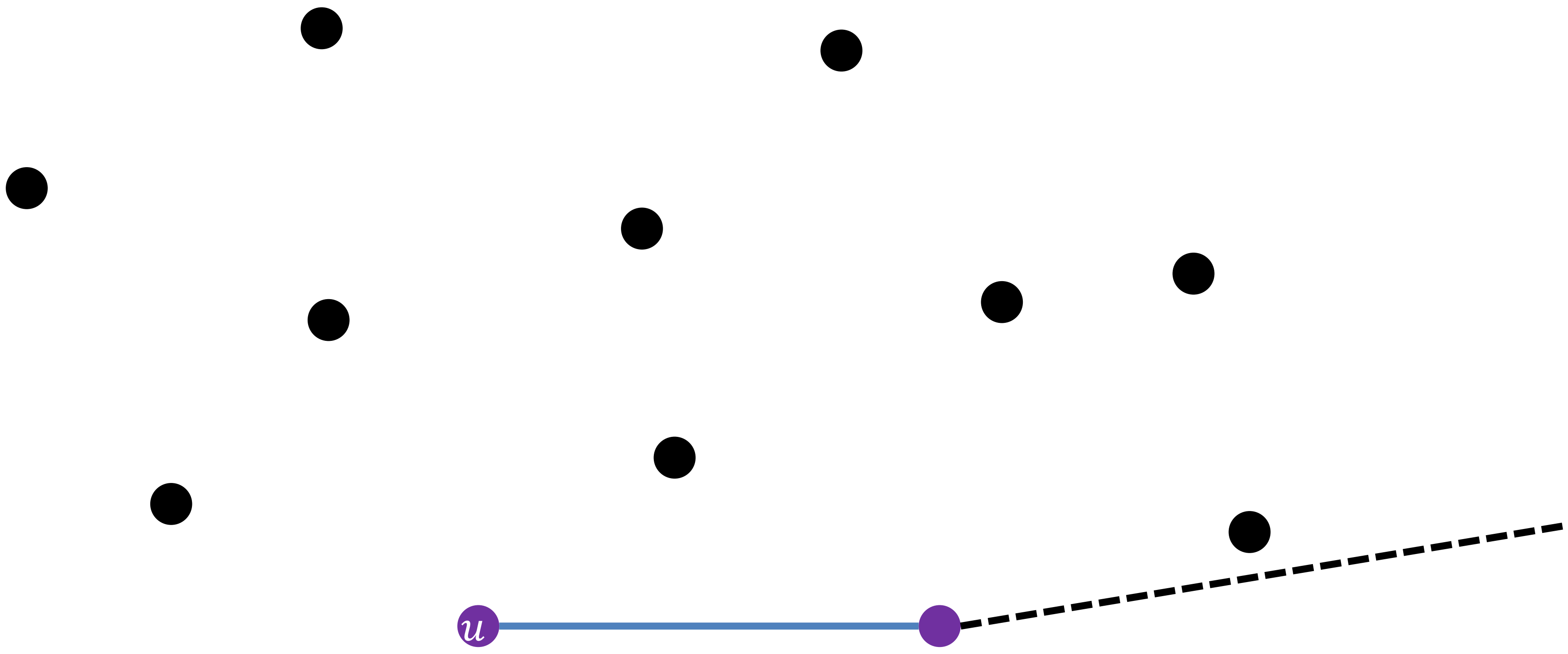
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



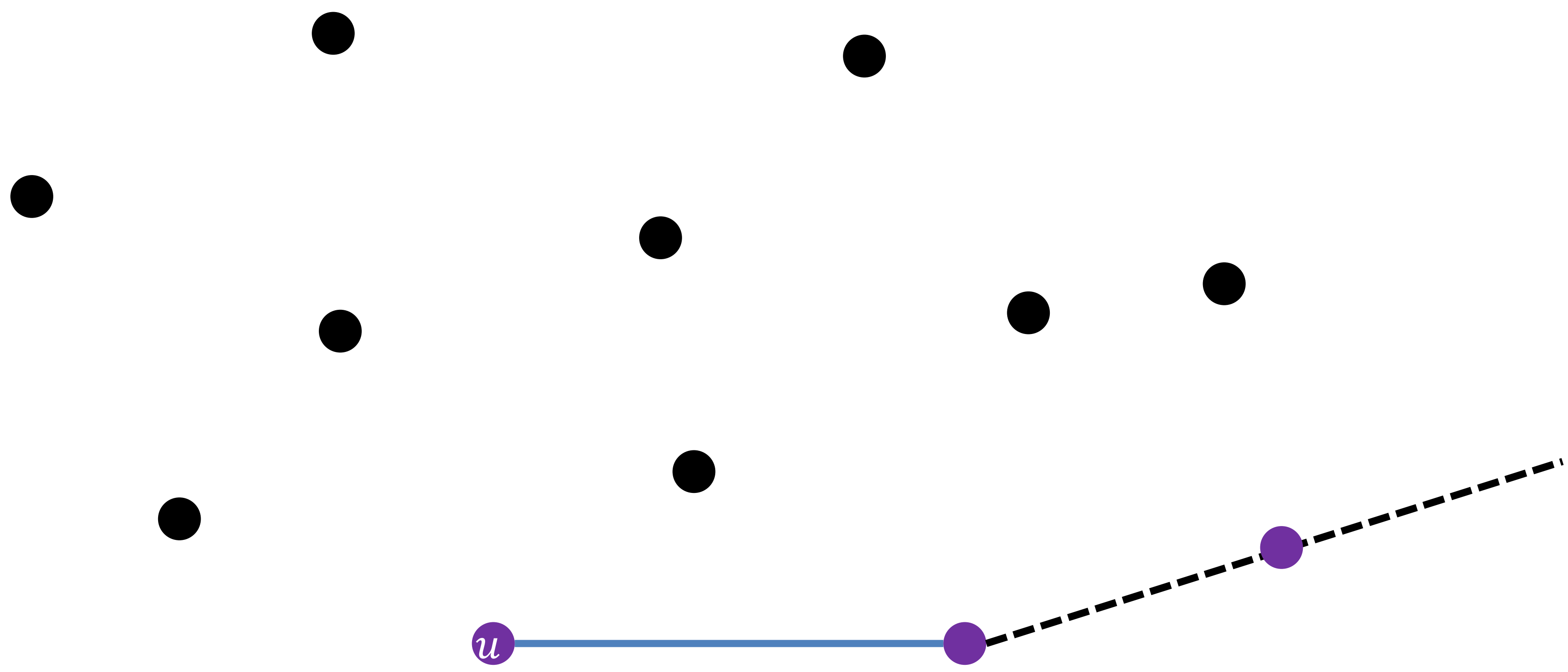
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



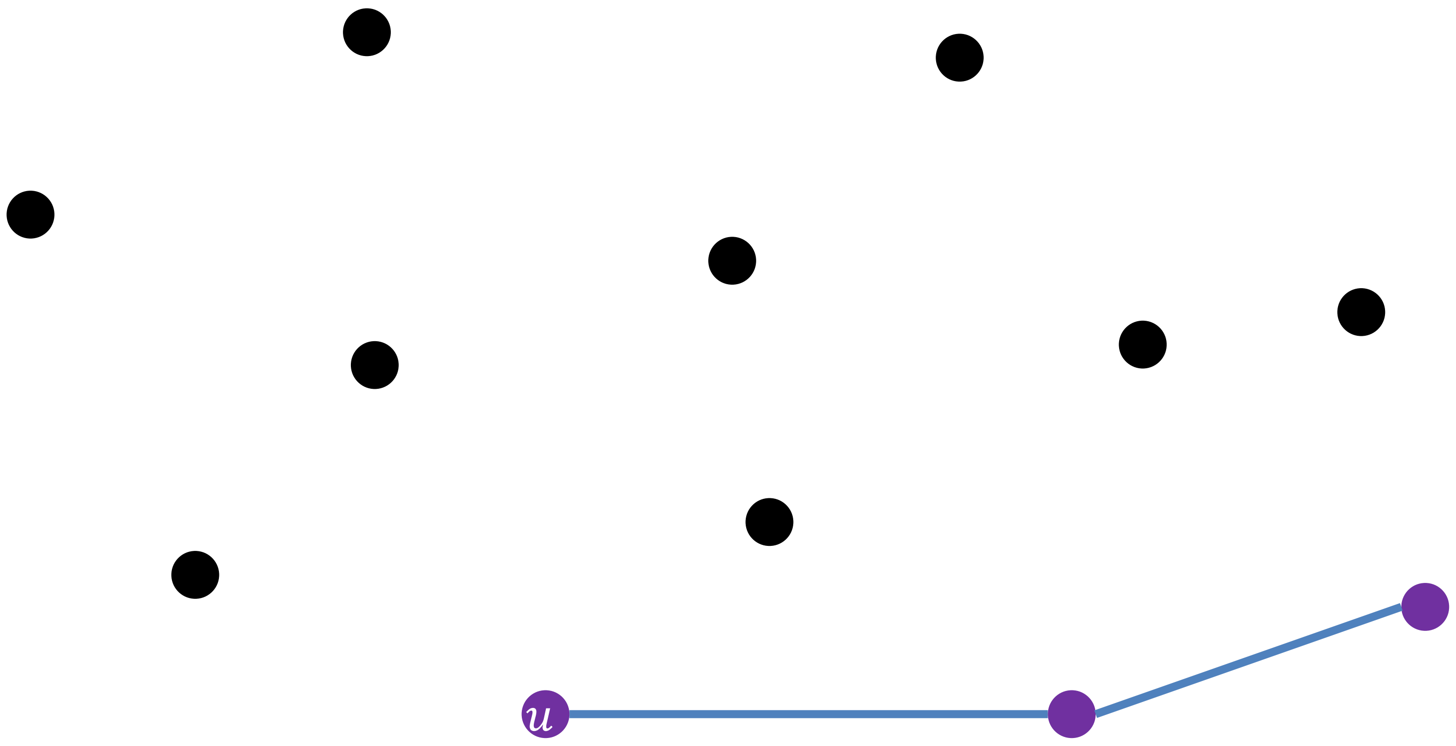
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



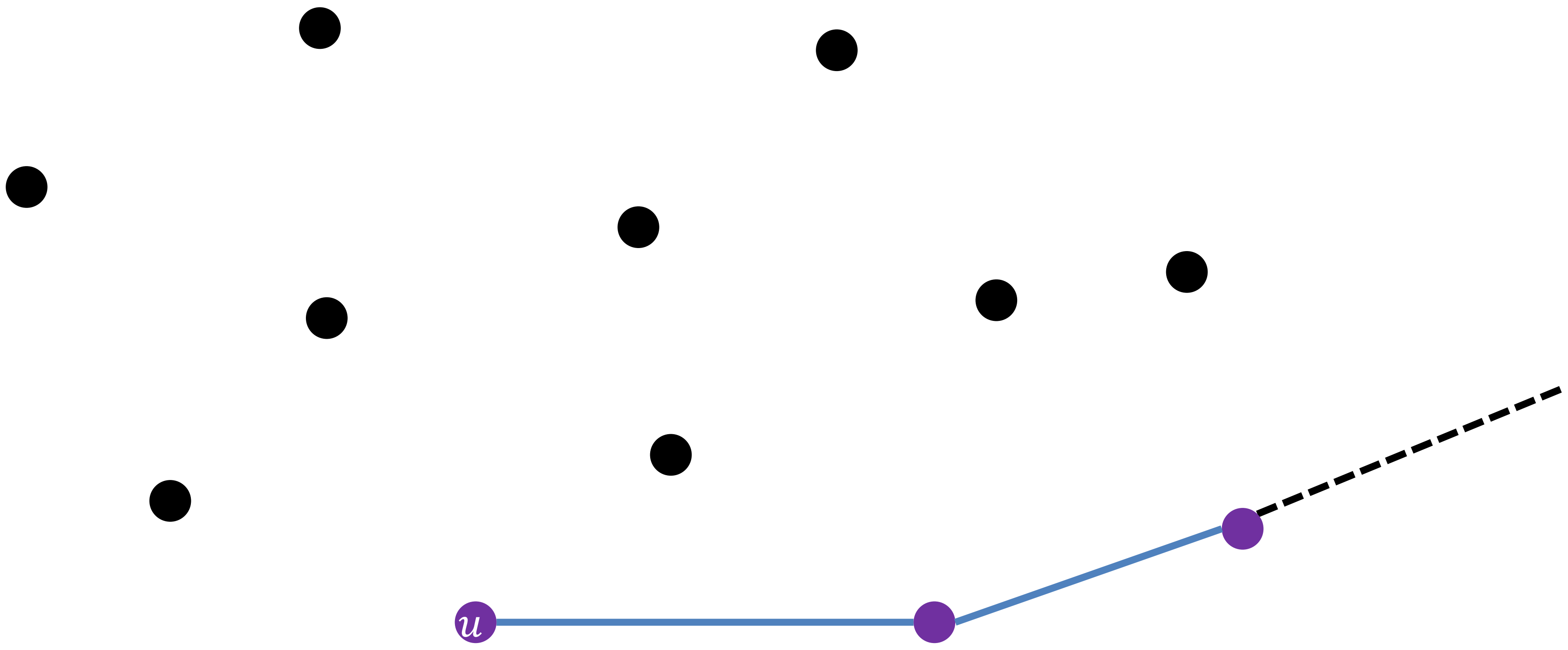
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



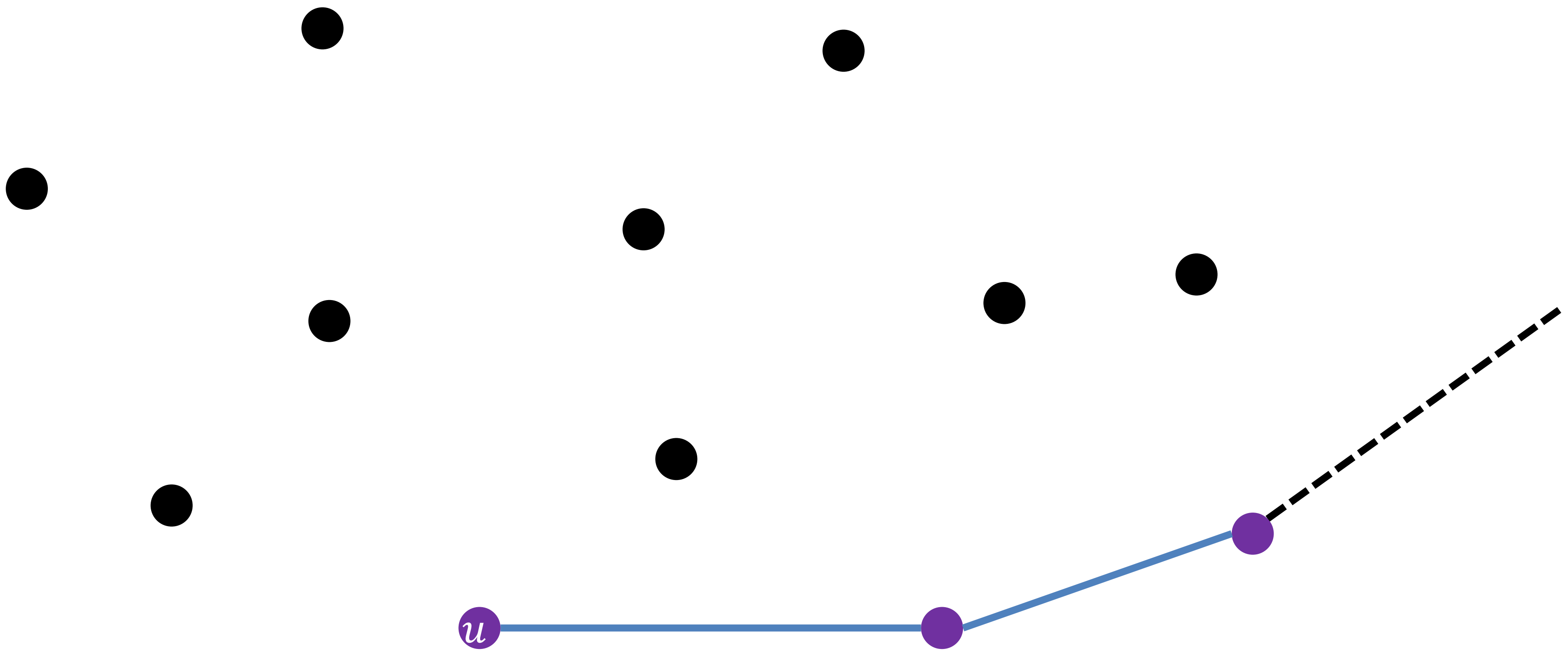
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



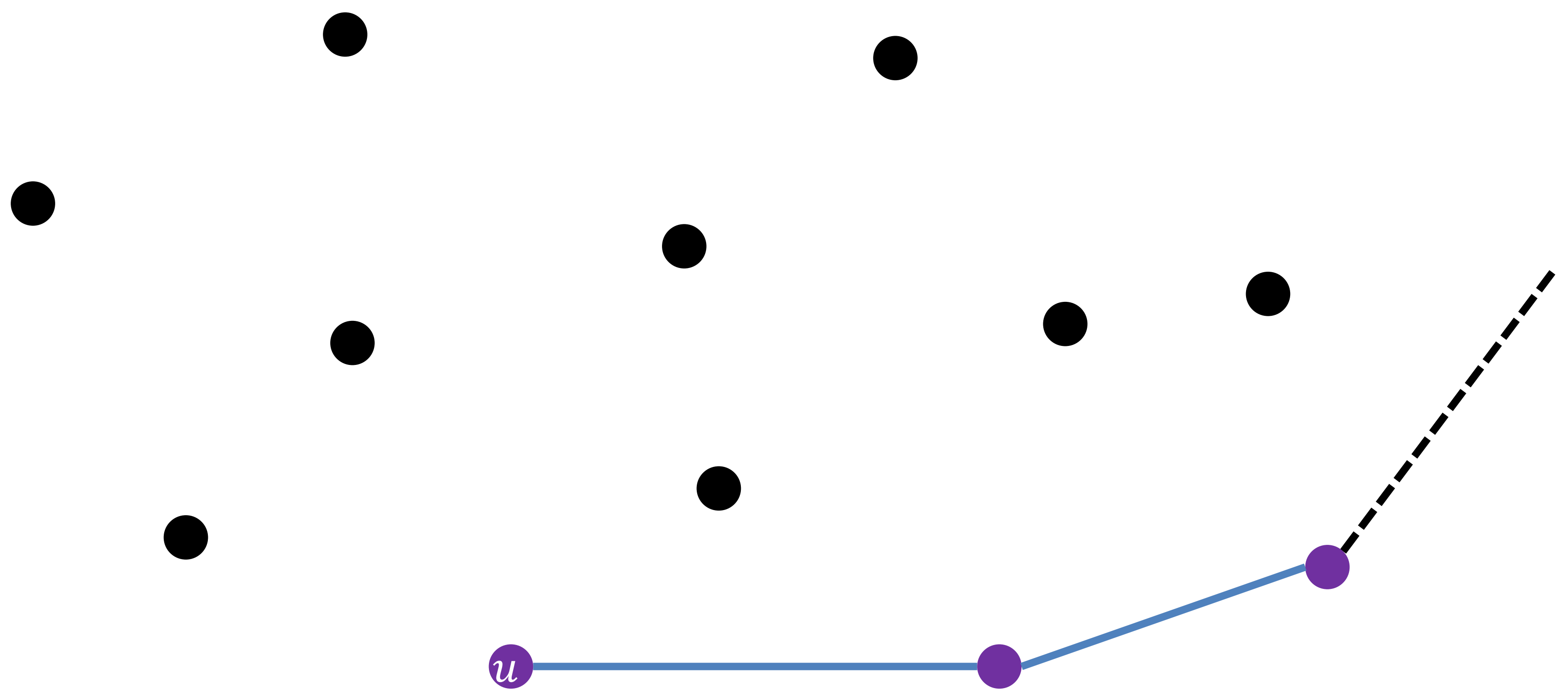
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



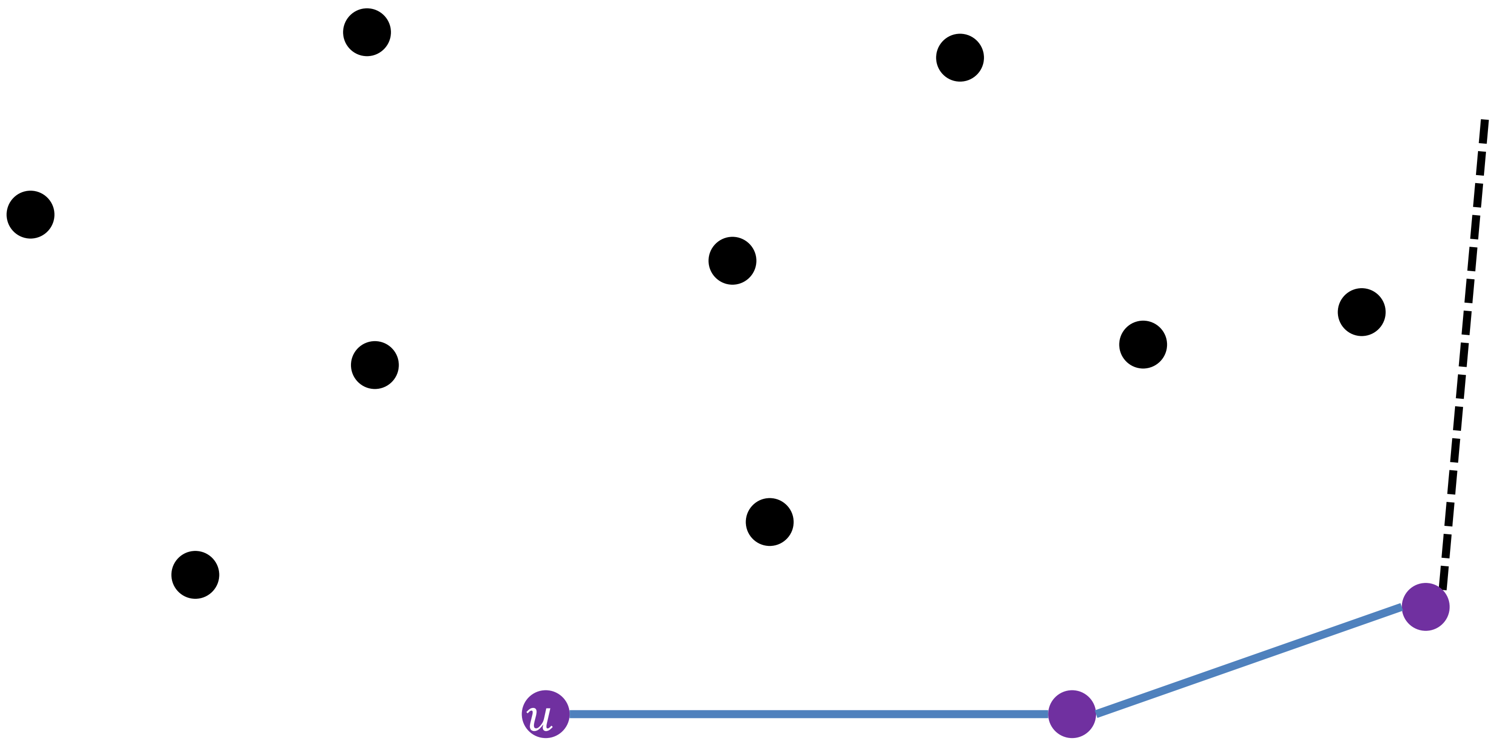
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



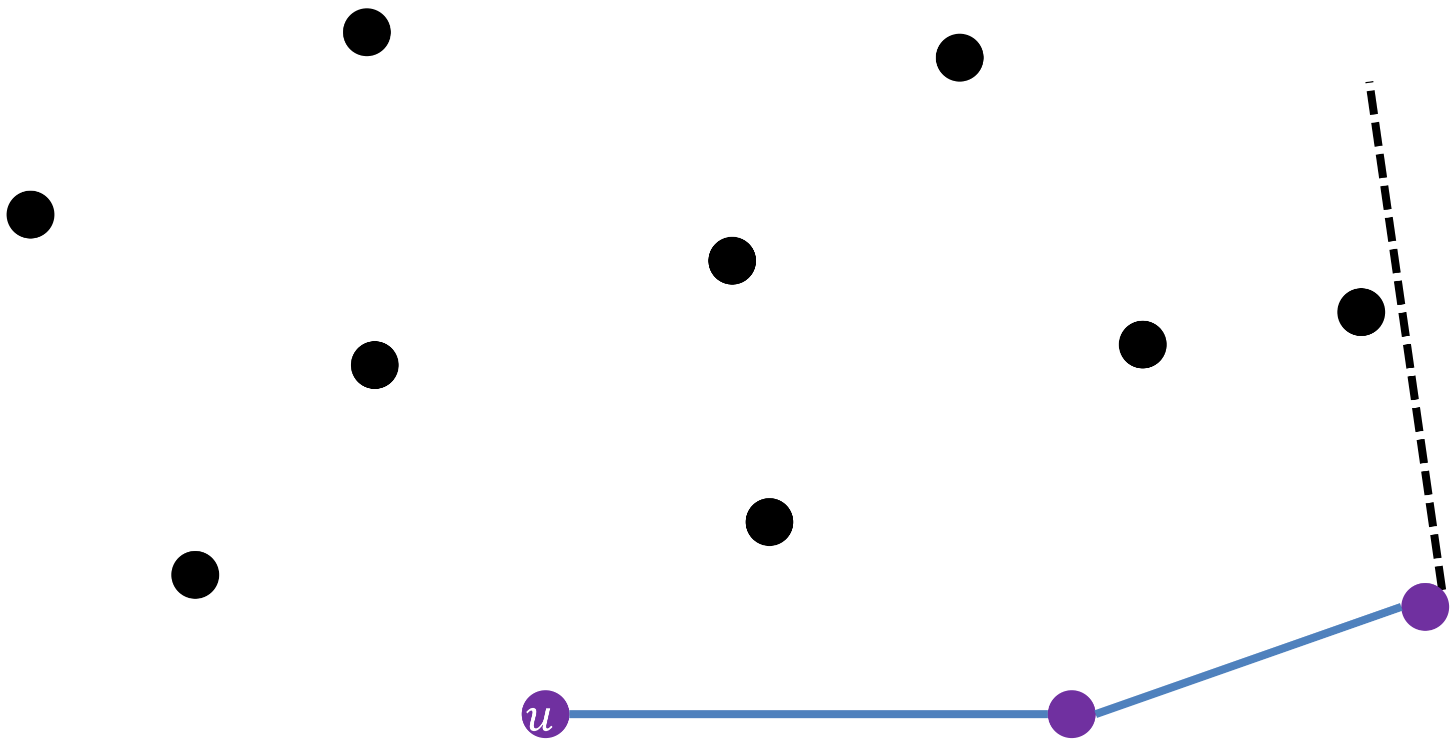
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



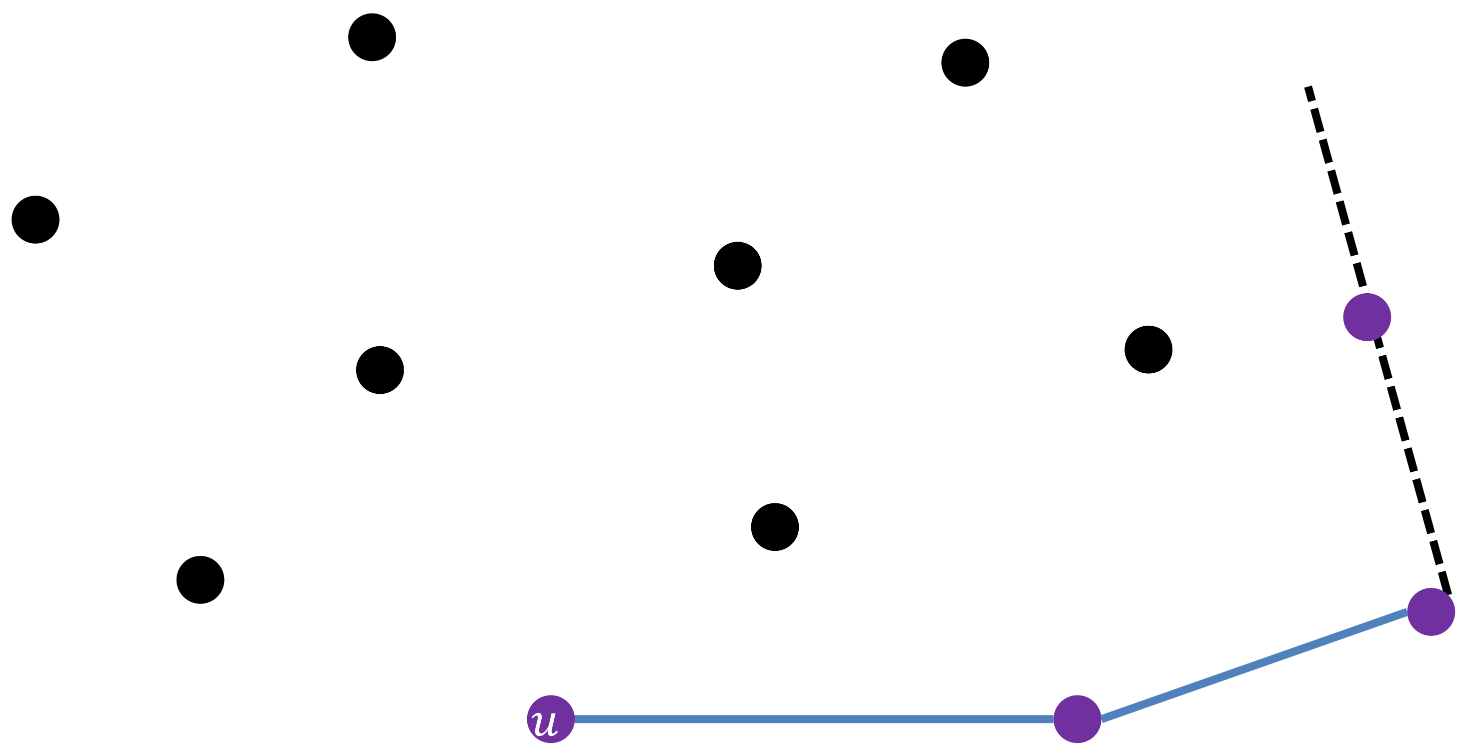
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



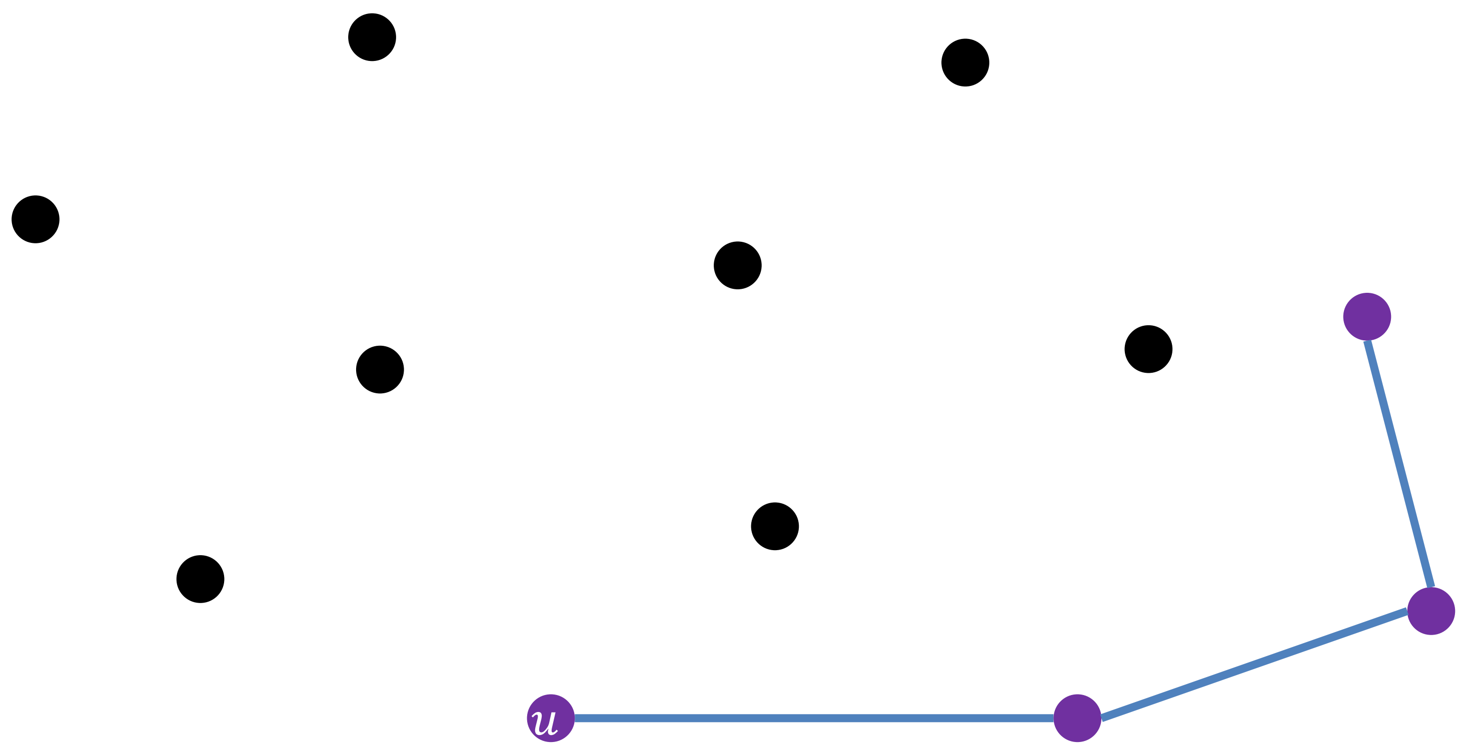
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



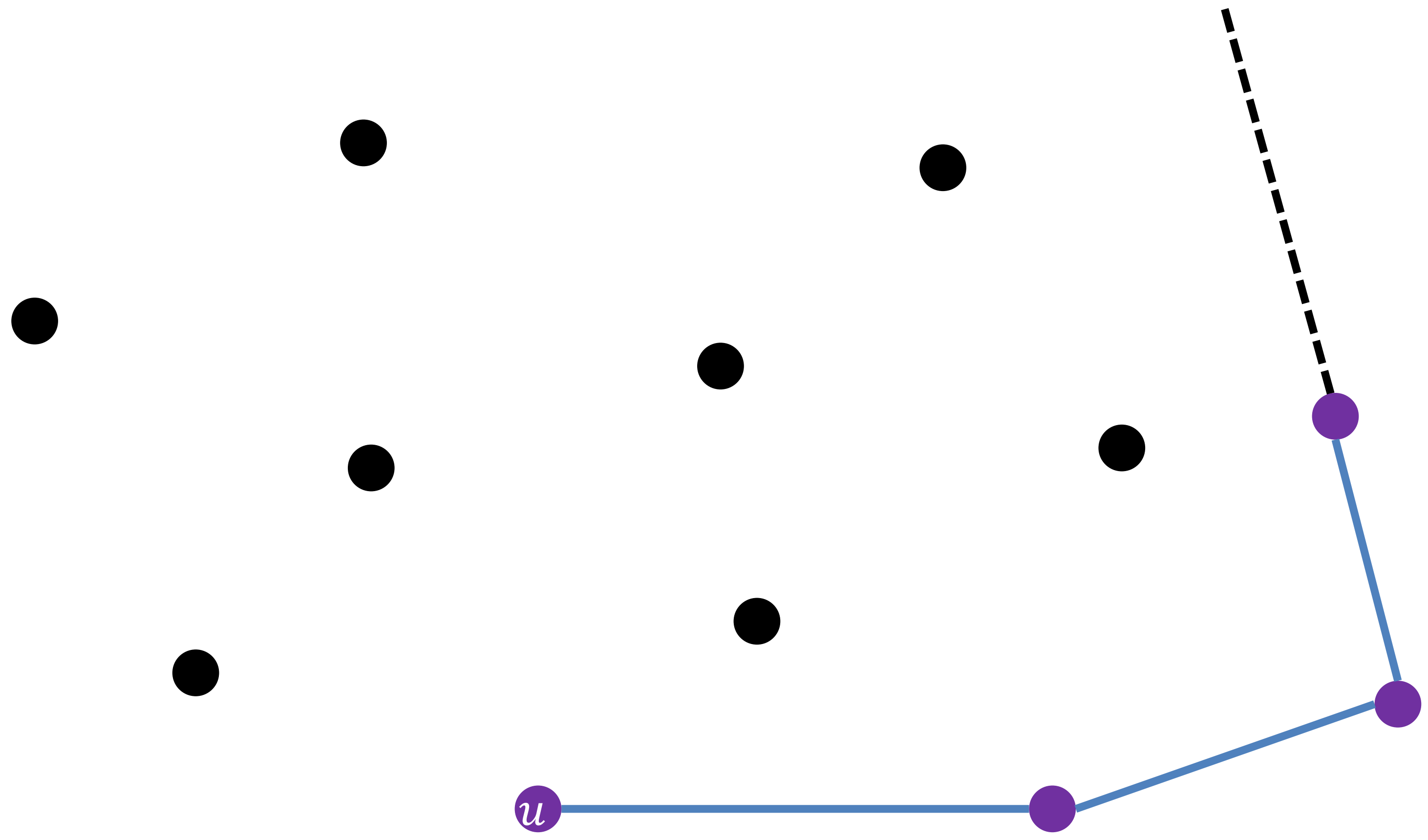
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



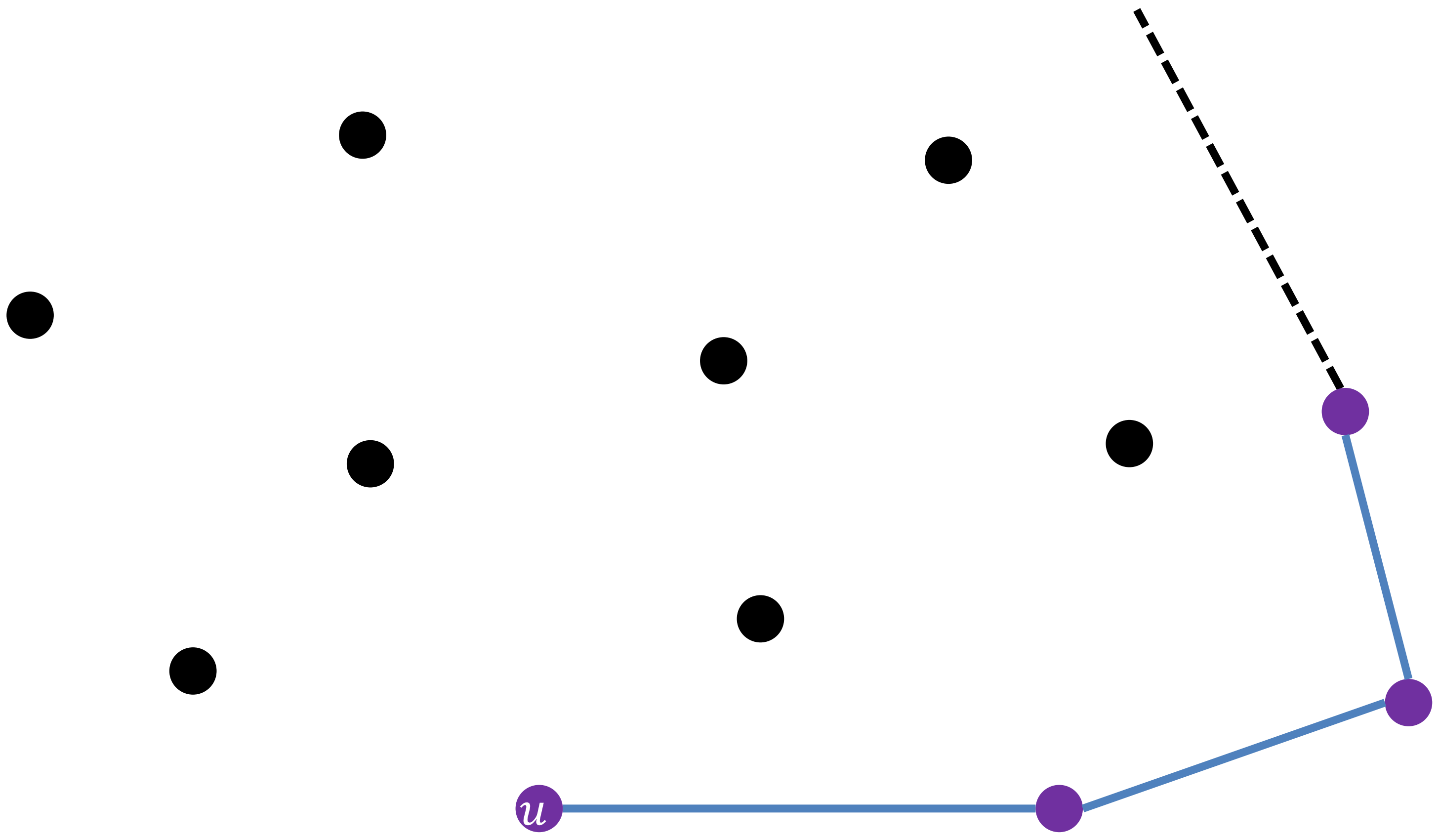
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



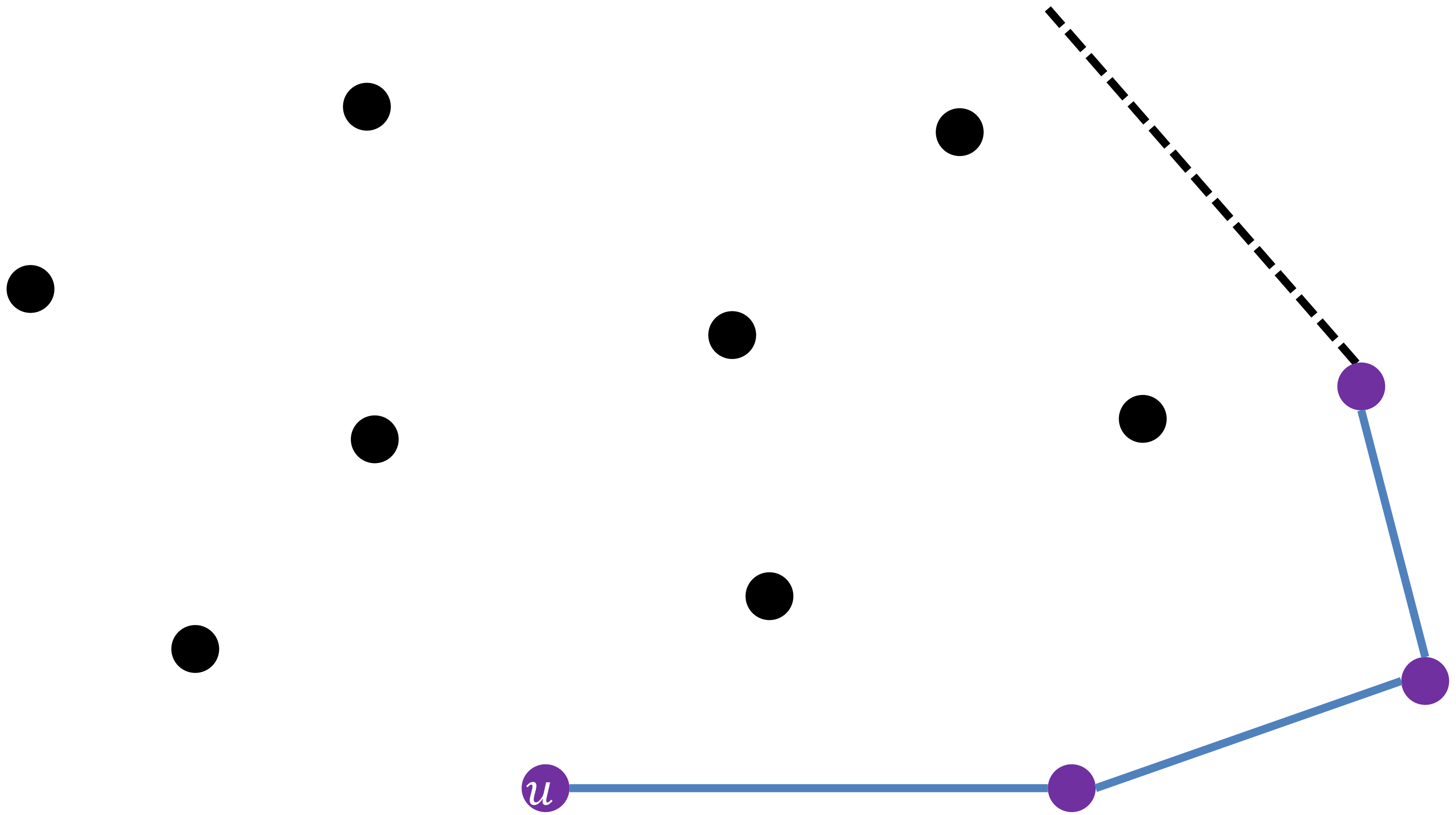
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



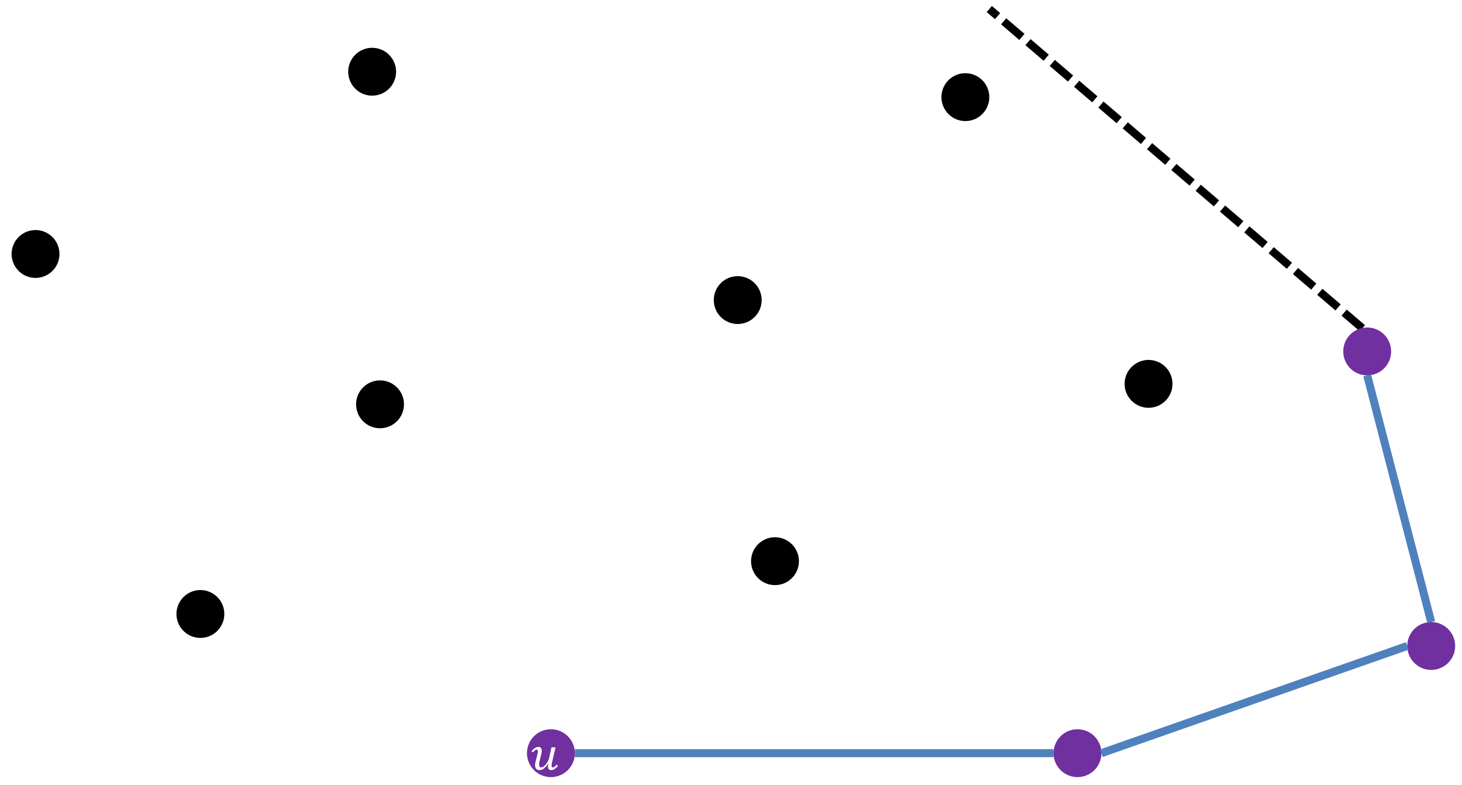
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



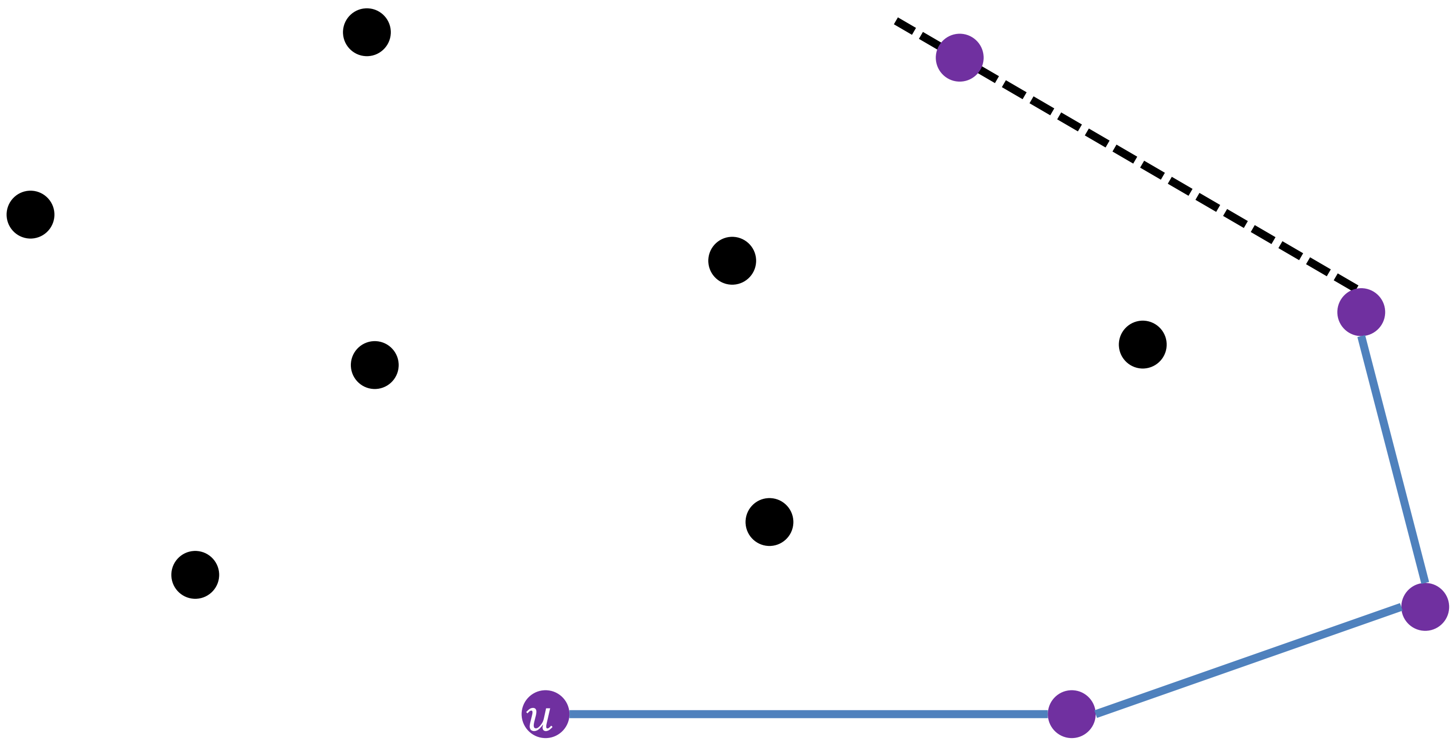
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



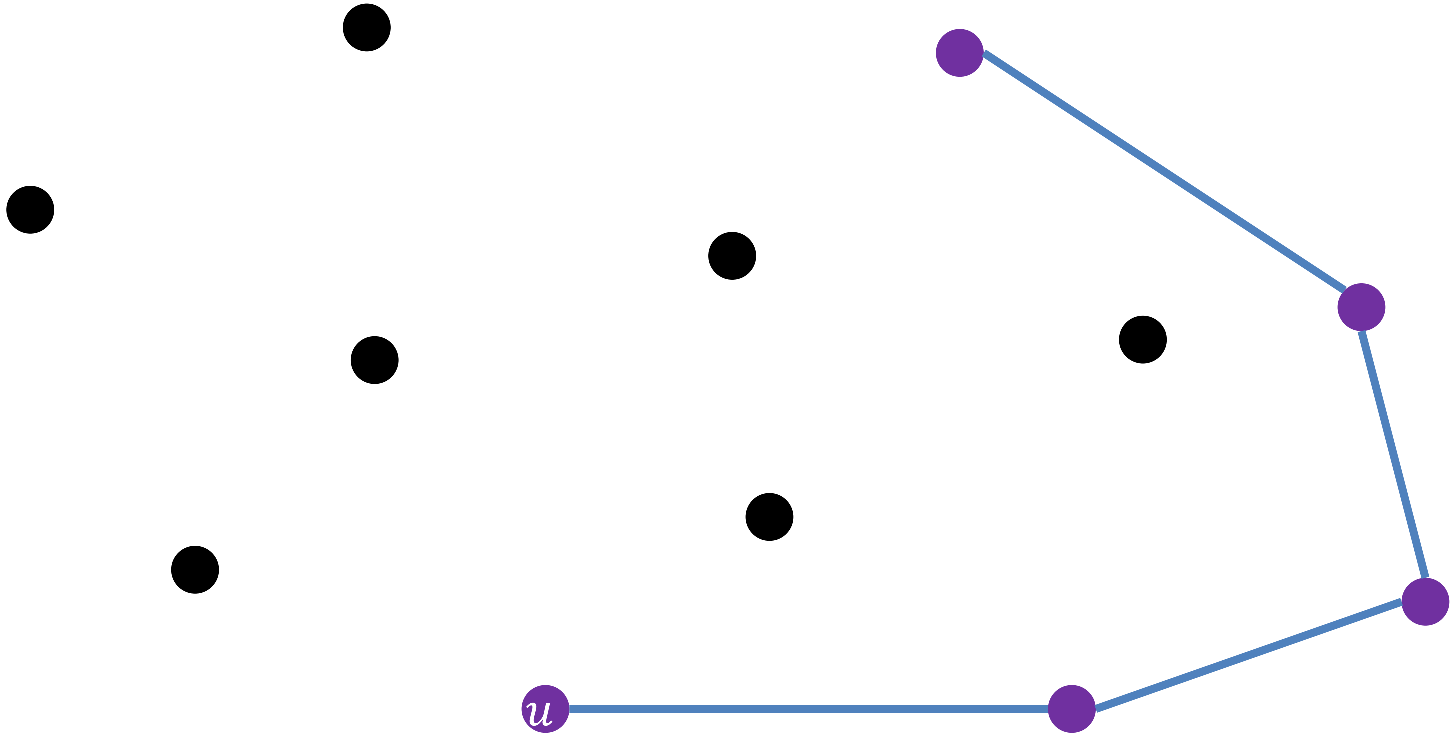
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



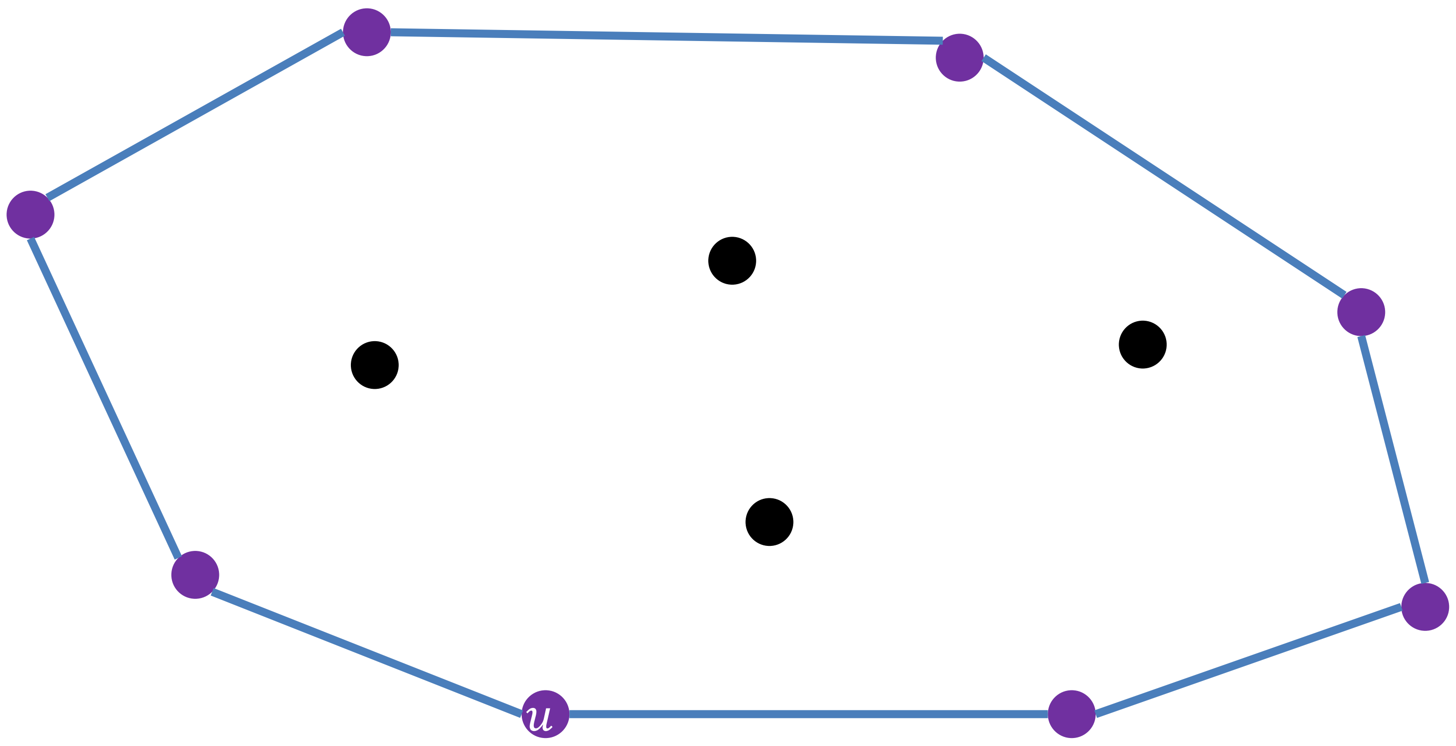
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



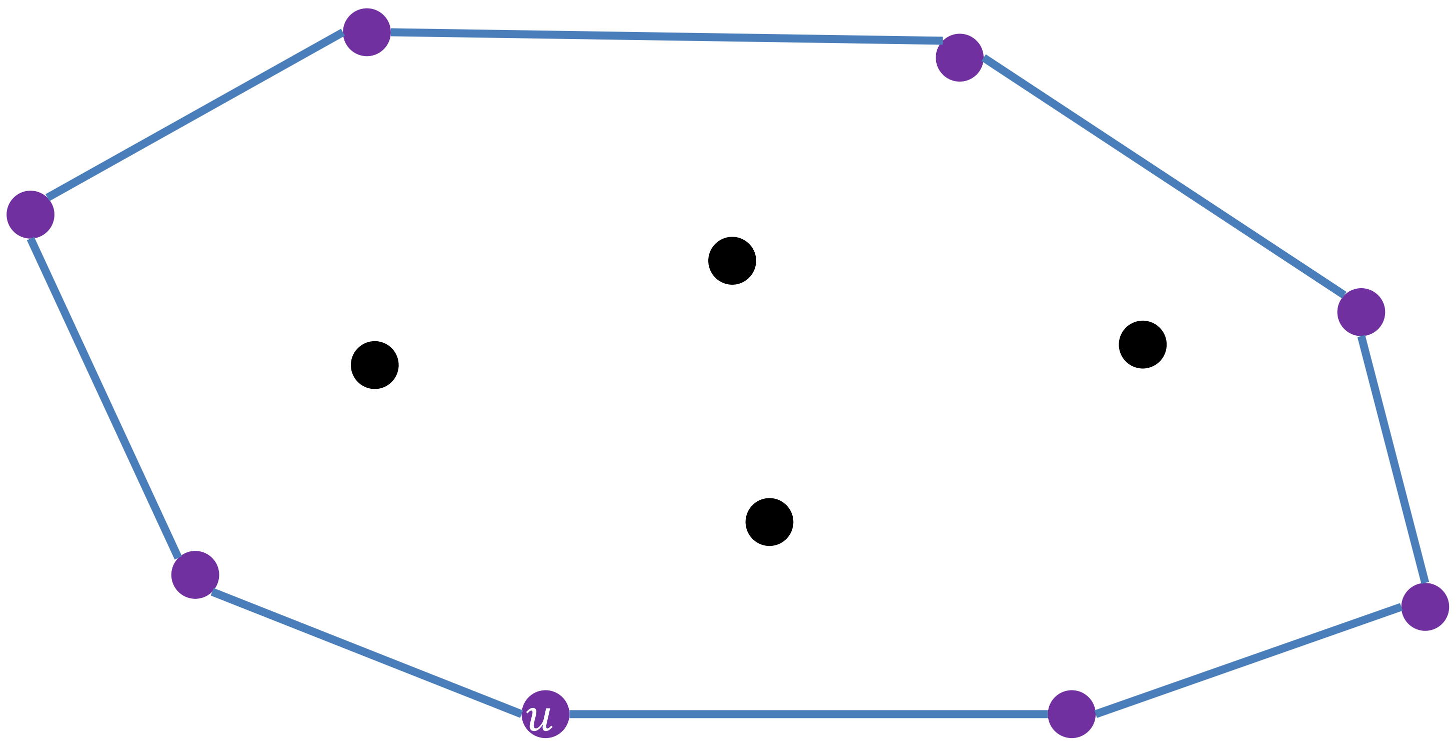
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)

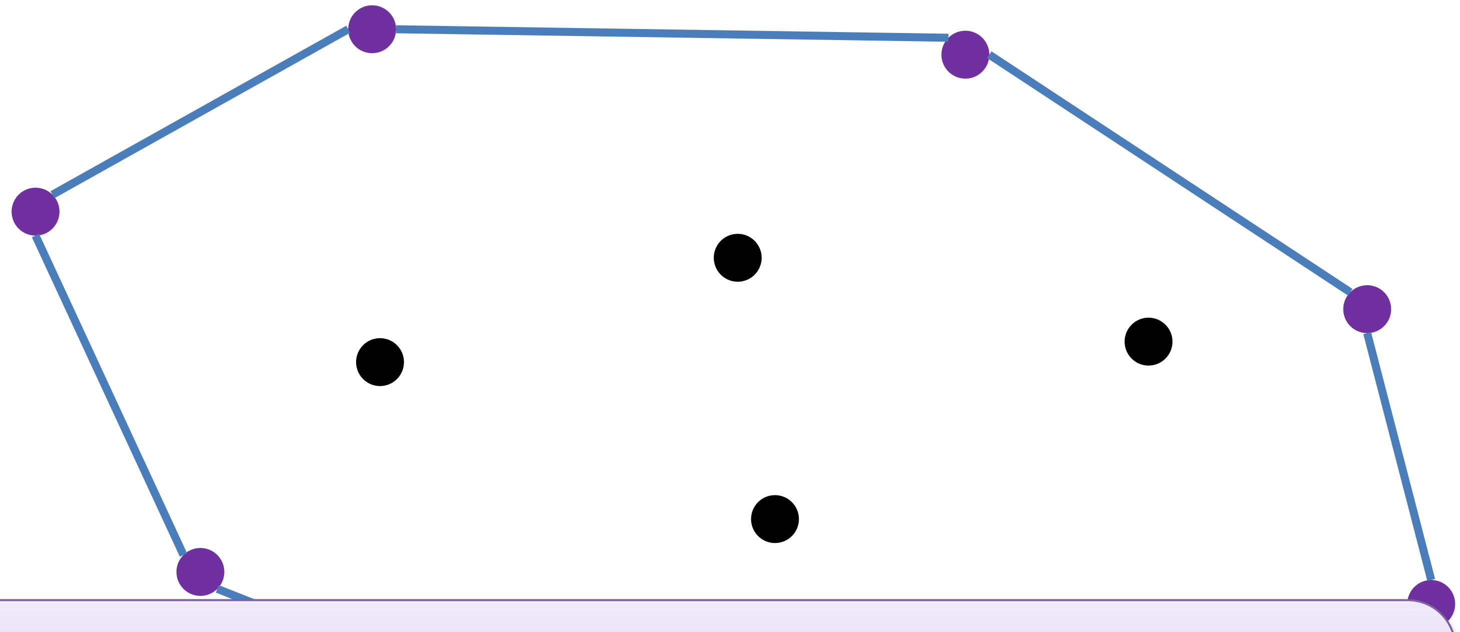


Can find the "next" point using a linear scan

Number of iterations: number of points on convex hull

Run time: $O(nh)$ where h is the number of points on the convex hull

Jarvis' Algorithm (Gift Wrapping Method)



Output-dependent running time (similar to Ford-Fulkerson)

- Can be better than Graham's Algorithm when $h \ll \log n$
- **Worst case:** $h = n$, so $O(n^2)$

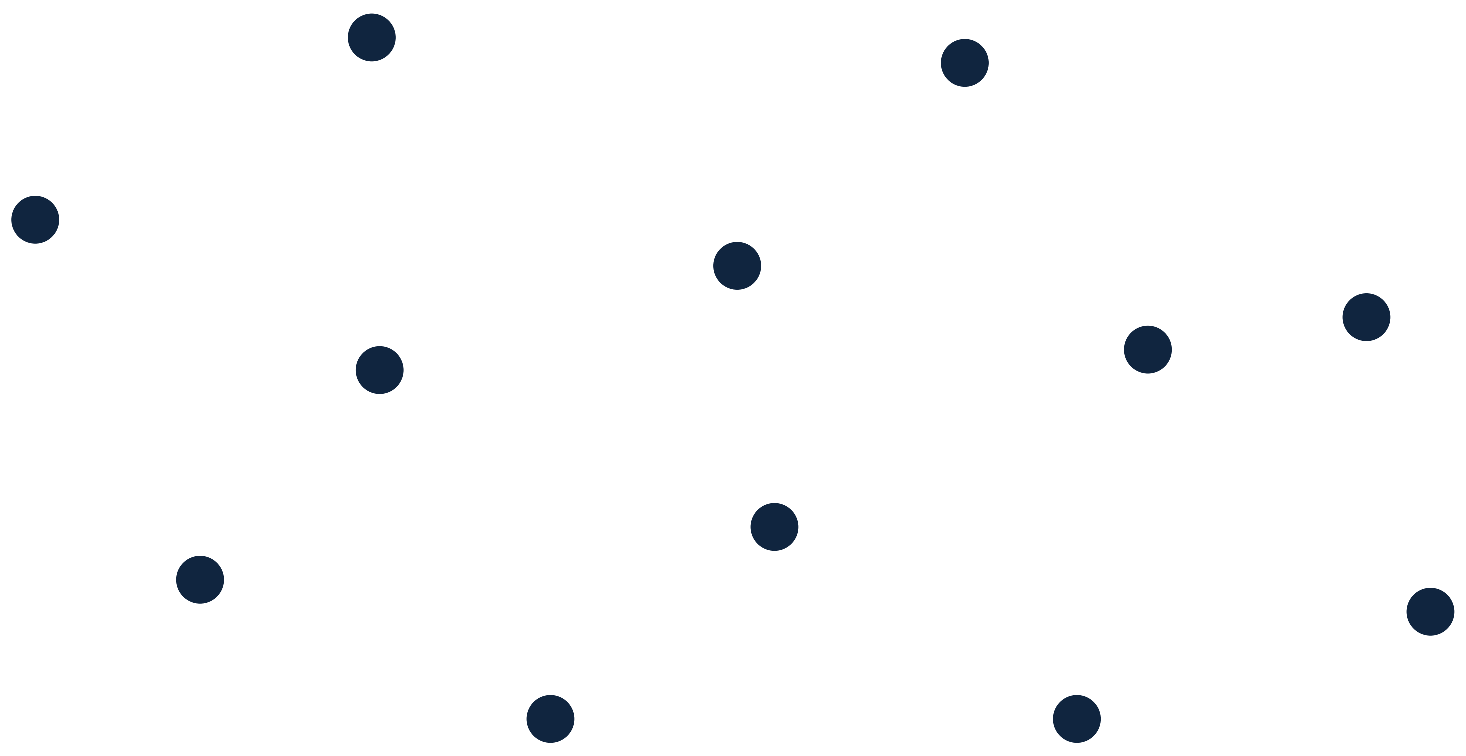
Ca

Number of iterat... mber of points on convex hull

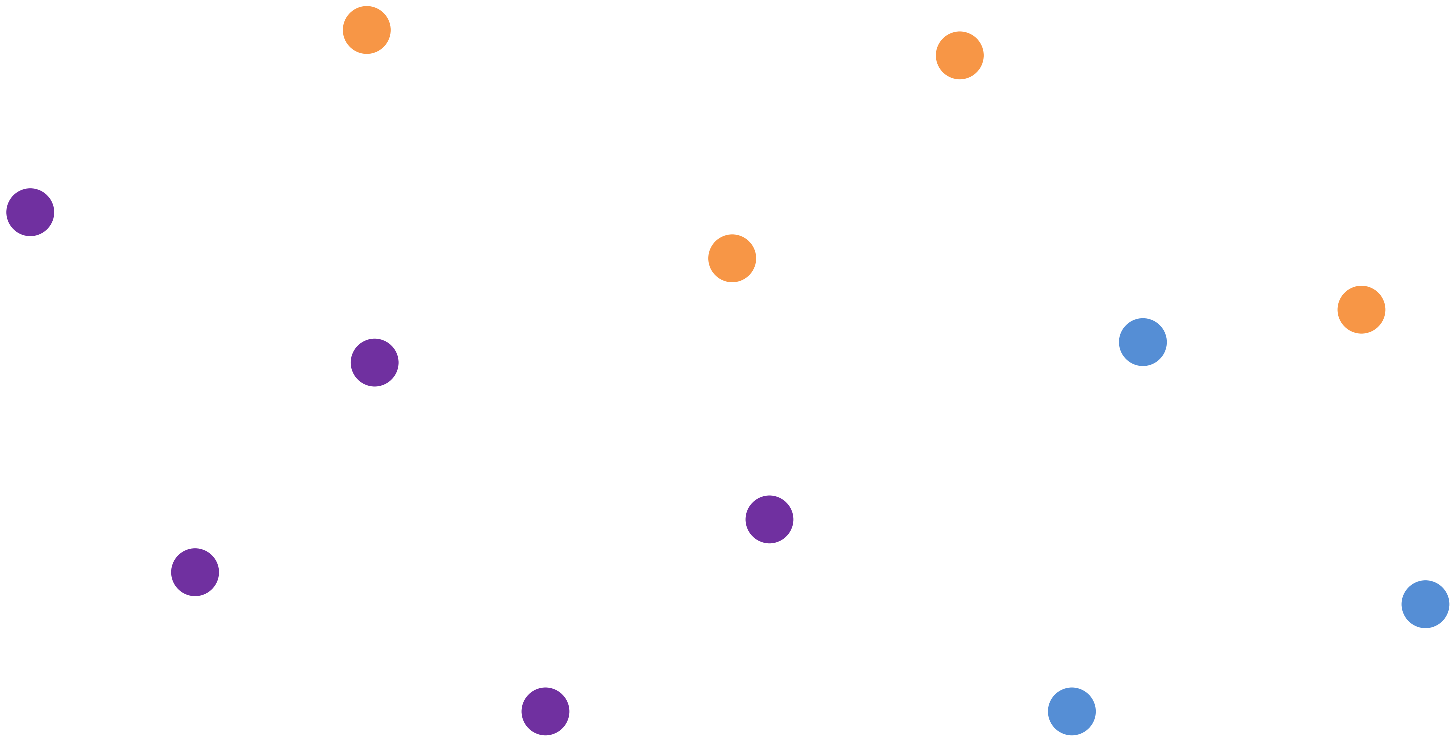
Run time: $O(nh)$ where h is the number of points on the convex hull


GRAHAM SCAN: O(NLOGN),
OR JARVIS MARCH O(NH)?
Why don't we have both?
CHAN'S ALGORITHM:
O(NLOGH)
imgflip.com

Chan’s Algorithm

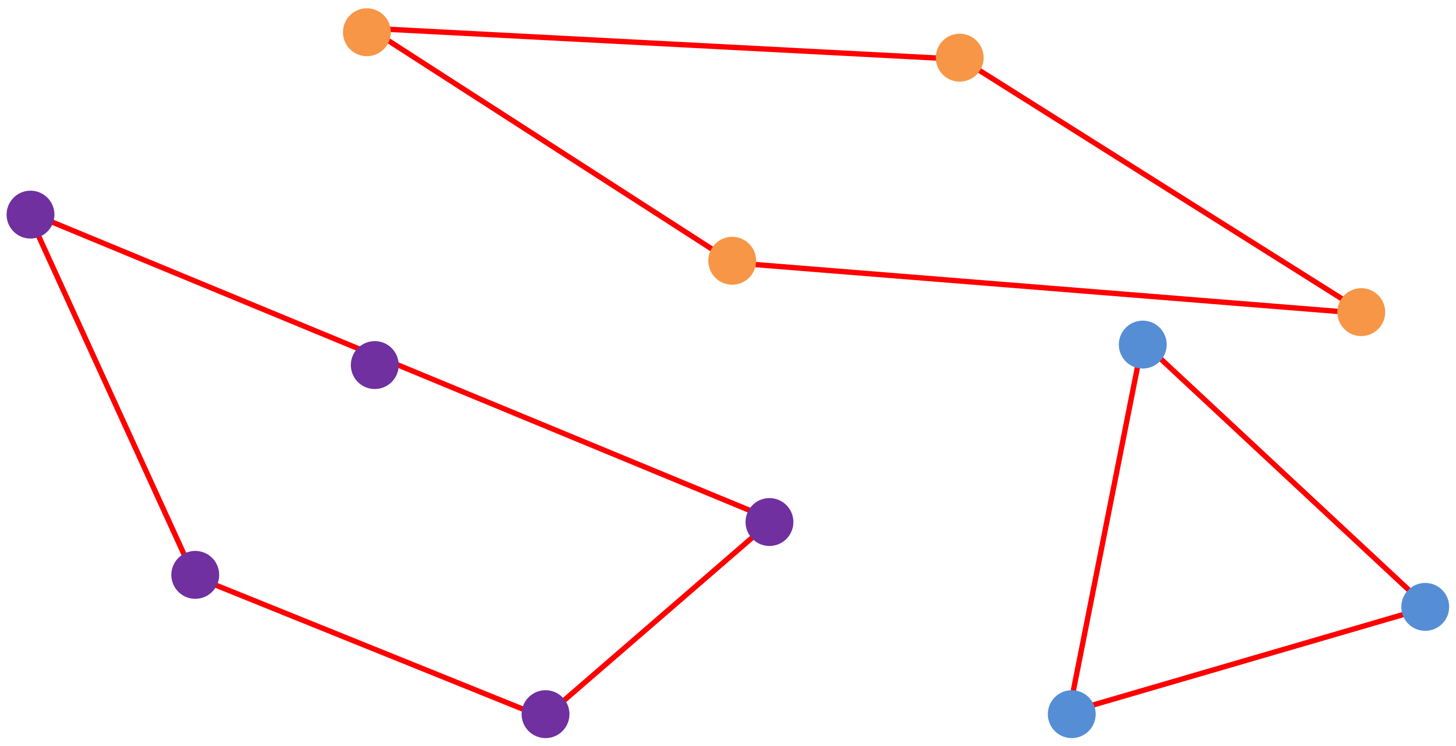


Chan’s Algorithm



Divide into smaller subsets

Chan’s Algorithm

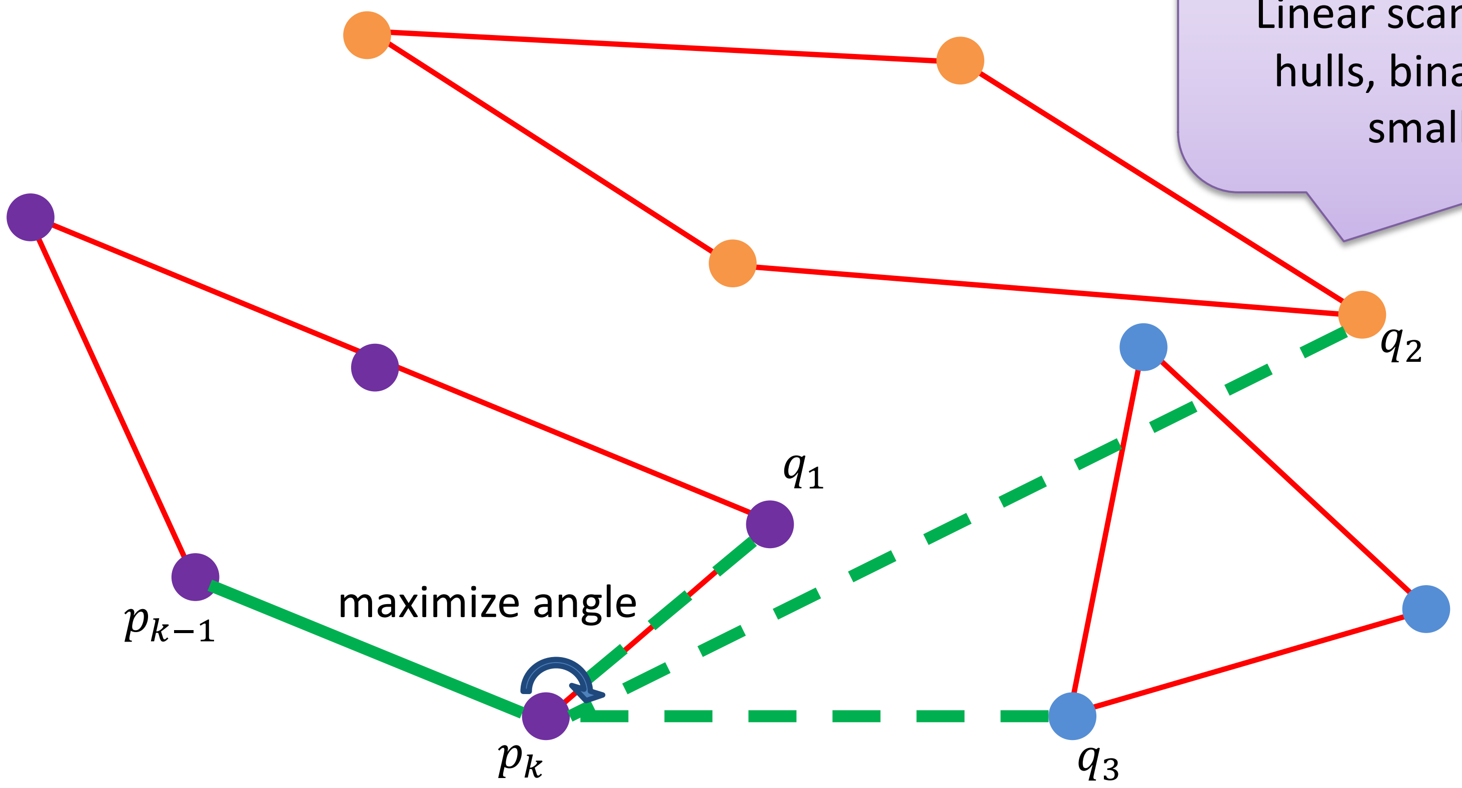


Use Graham’s Algorithm to **conquer** the smaller subsets

Chan's Algorithm

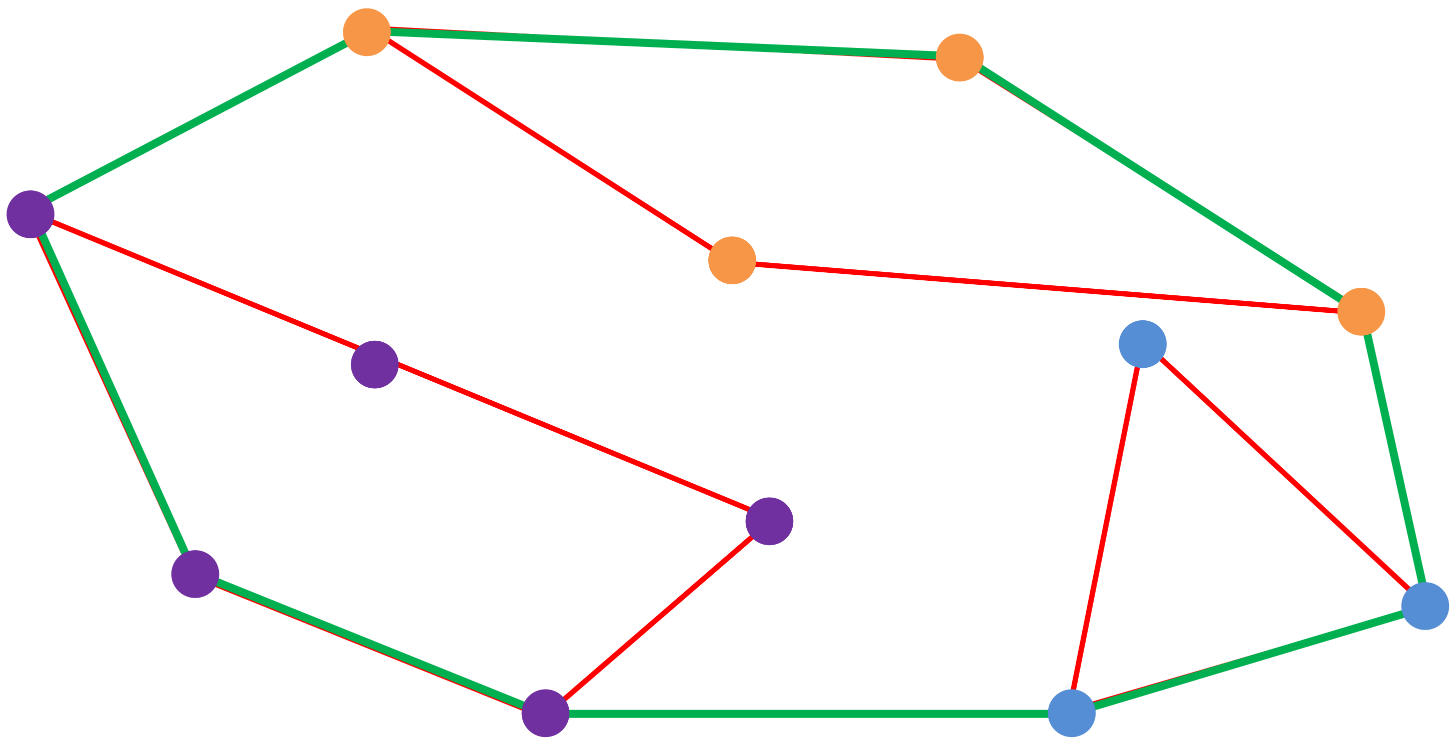
Jarvis checks all other points, but...

Linear scan over smaller convex hulls, binary search inside the smaller convex hulls



Use Jarvis' Algorithm to **combine** the solutions to the smaller subsets

Chan's Algorithm



Use Jarvis' Algorithm to **combine** the solutions to the smaller subsets

Chan's Algorithm

Combines Graham's Algorithm *and* Jarvis' Algorithm
Given points P, size of subsets m, guess of number of hull points H

Partition P into subsets $P_1, P_2, \ldots, P_{\lceil n/m \rceil}$ of size at most m

Divide

for $i = 1, \ldots, \lceil n/m \rceil$
    Compute $conv(P_i)$ using *Graham's Algorithm,* store in counter-clockwise order

Conquer with Graham's Algorithm

$p_0 \leftarrow (0, -\infty)$
$p_1 \leftarrow$ rightmost point of P
for $k = 1, \ldots, H$ (each hull point)
    for $i = 1, \ldots, \lceil n/m \rceil$ (each subset)
        Use binary search on $conv(P_i)$ to find point q_i
        that maximizes the angle $\angle p_{k-1} p_k q_i$
    $p_{k+1} \leftarrow q \in \{q_1, q_2, \ldots, q_{\lceil \frac{n}{m} \rceil}\}$ that maximizes the angle $\angle p_{k-1} p_k q$ *(Jarvis' Algorithm)*
    if $p_{k+1} = p_1$, then return $conv(P) = \{p_1, p_2, \ldots, p_k\}$

Combine with Jarvis' Algorithm

$O(n \log h)$
Where h is the number of hull points

https://link.springer.com/content/pdf/10.1007%2FBF02712873.pdf