# Speculative Parallelization of a Randomized Incremental Convex Hull Algorithm

M. Cintra (i), D. Llanos (ii), **B. Palop (ii)**

(i) School of Informatics, U. Edinburgh, UK

(ii) Dpt. of Computer Science, U. Valladolid, Spain

# Keywords

“Convex Hull”, “Speculative Parallelization”, “Randomized Incremental Algorithm”

- **Why the Convex hull?**
- **Why Speculative Parallelization?**
- **Why Randomized Algorithms?**

# Keywords

"Convex Hull", "Speculative Parallelization", "Randomized Incremental Algorithm"

- **Why the Convex hull?**

  Frequently used simple structure whose computation is bottleneck for many others

- **Why Speculative Parallelization?**

- **Why Randomized Algorithms?**

# Keywords

"Convex Hull", "Speculative Parallelization", "Randomized Incremental Algorithm"

- **Why the Convex hull?**

  Frequently used simple structure whose computation is bottleneck for many others

- **Why Speculative Parallelization?**

  No need to design specific parallel algorithms

- **Why Randomized Algorithms?**

# Keywords

"Convex Hull", "Speculative Parallelization", "Randomized Incremental Algorithm"

- **Why the Convex hull?**

  Frequently used simple structure whose computation is bottleneck for many others

- **Why Speculative Parallelization?**

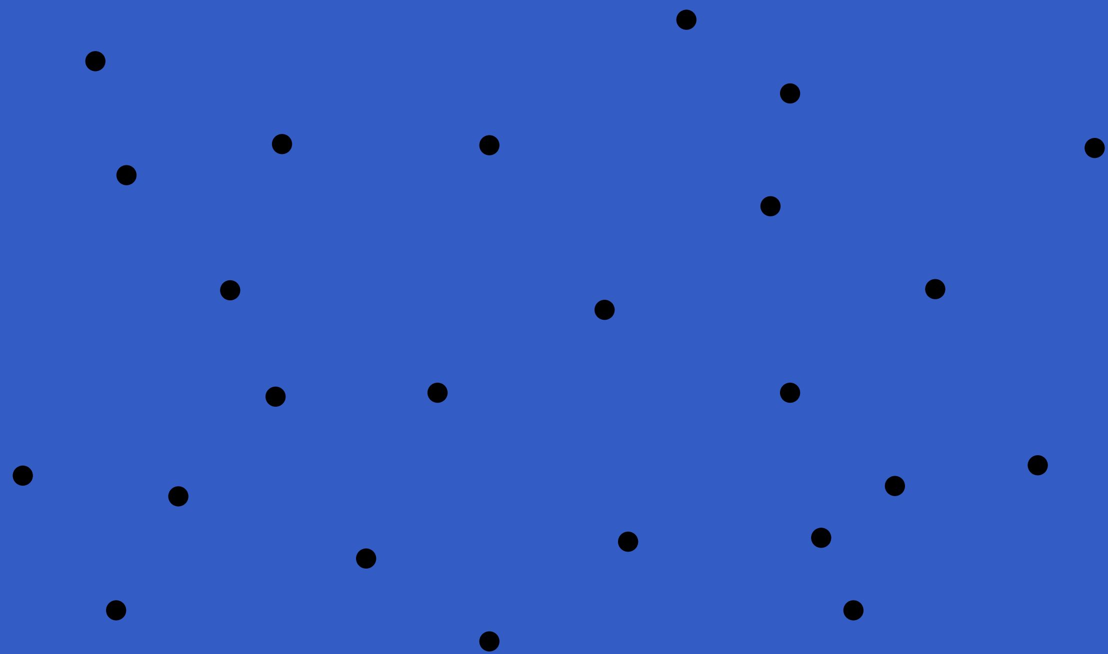
  No need to design specific parallel algorithms

- **Why Randomized Algorithms?**

  Lower time bounds are expected in most cases

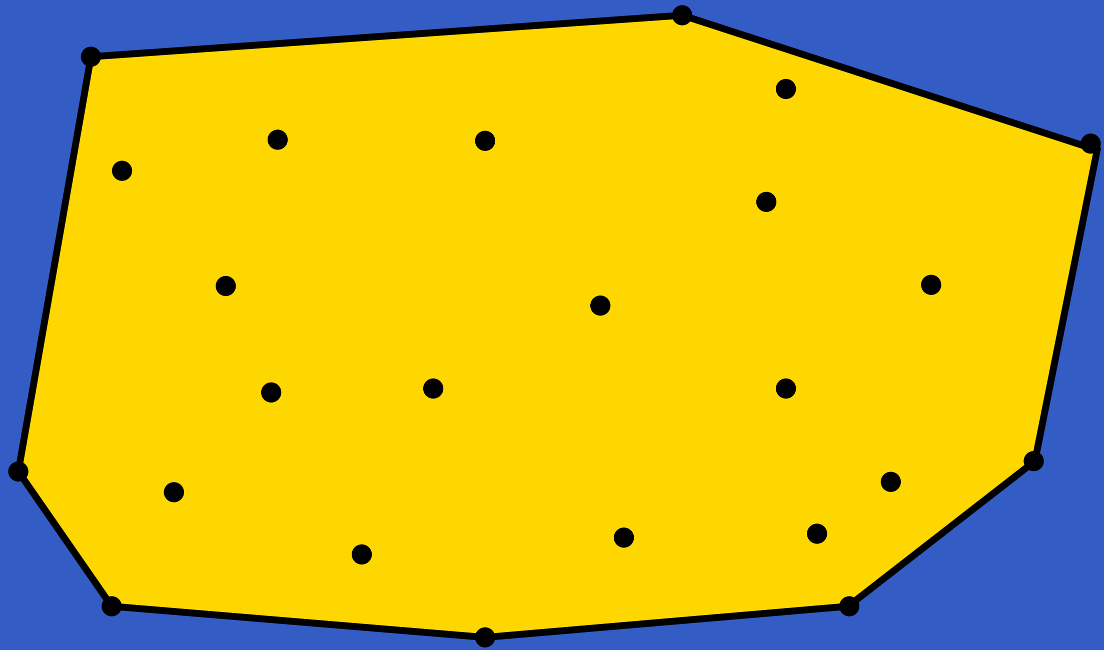
# Convex Hull of a set of points

- Definition: Given a set $S$ of points in the plane, $CH(S)$ is the smallest convex set containing $S$.



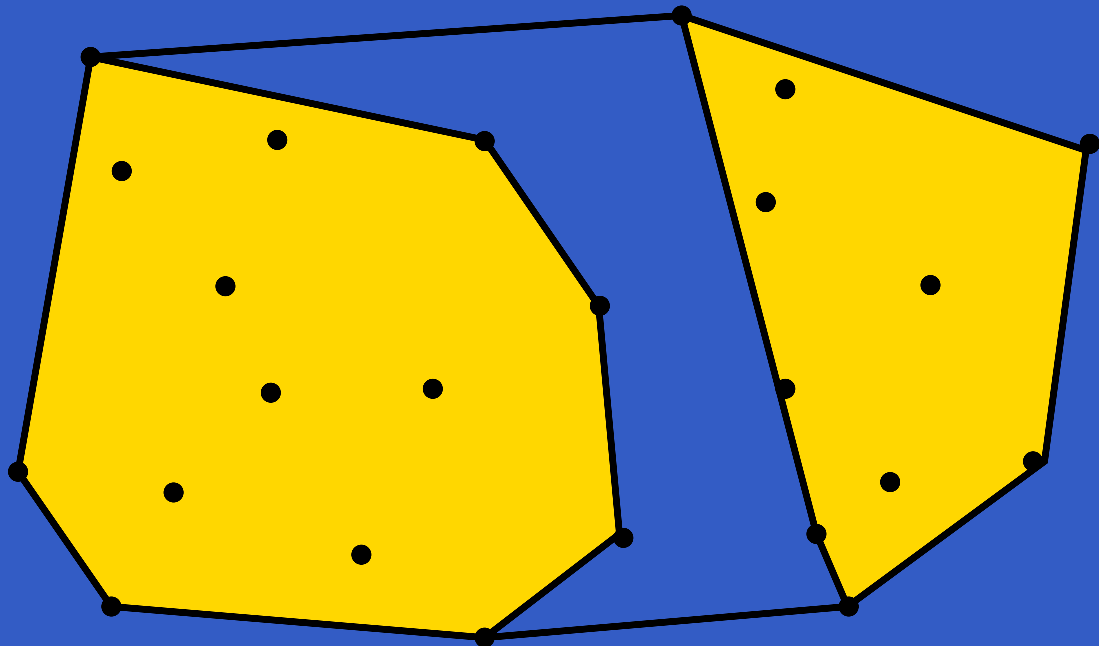
# Convex Hull of a set of points

- Definition: Given a set $S$ of points in the plane, $CH(S)$ is the smallest convex set containing $S$.
- Construction:



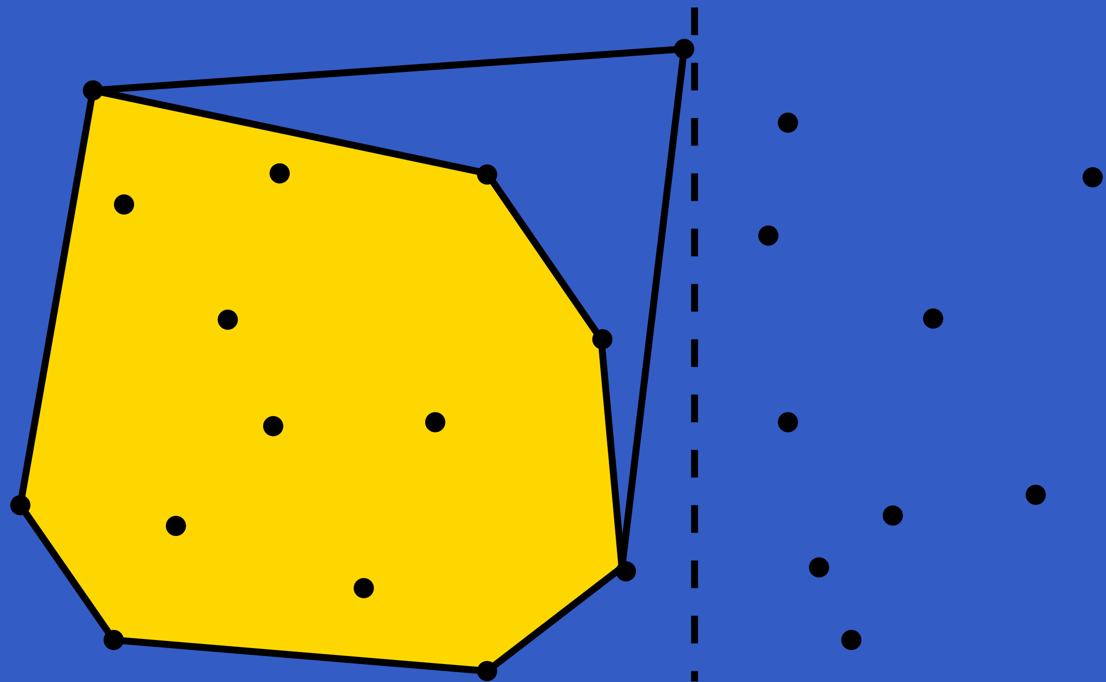
# Convex Hull of a set of points

- Definition: Given a set $S$ of points in the plane, $CH(S)$ is the smallest convex set containing $S$.
- Construction: Divide and conquer



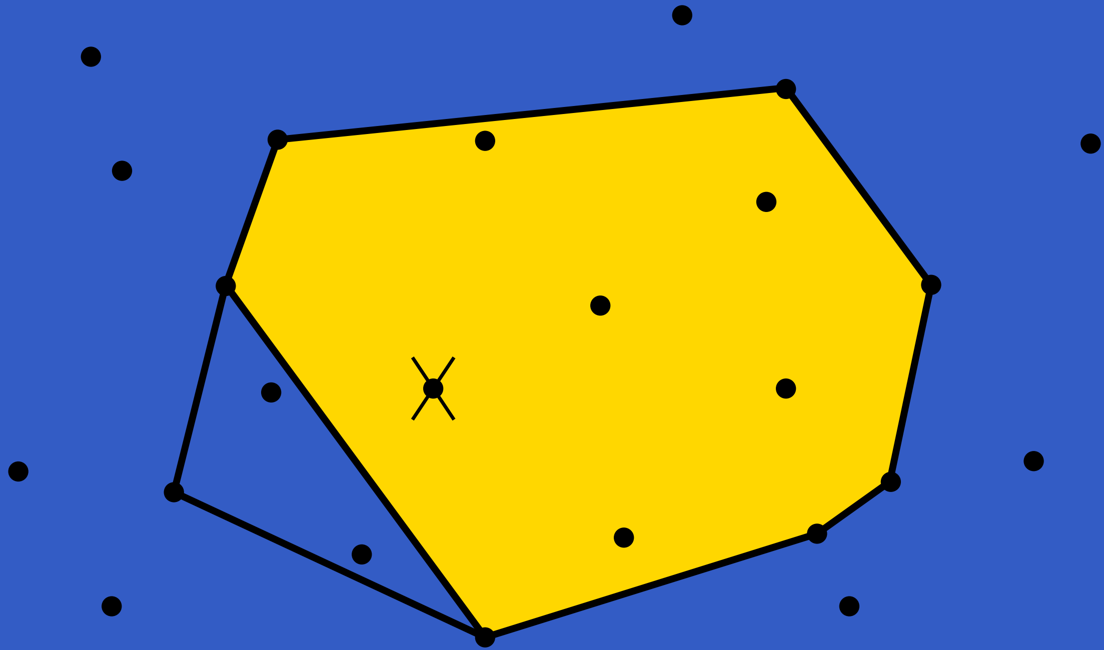
# Convex Hull of a set of points

- Definition: Given a set $S$ of points in the plane, $CH(S)$ is the smallest convex set containing $S$.
- Construction: Divide and conquer, Sweep line



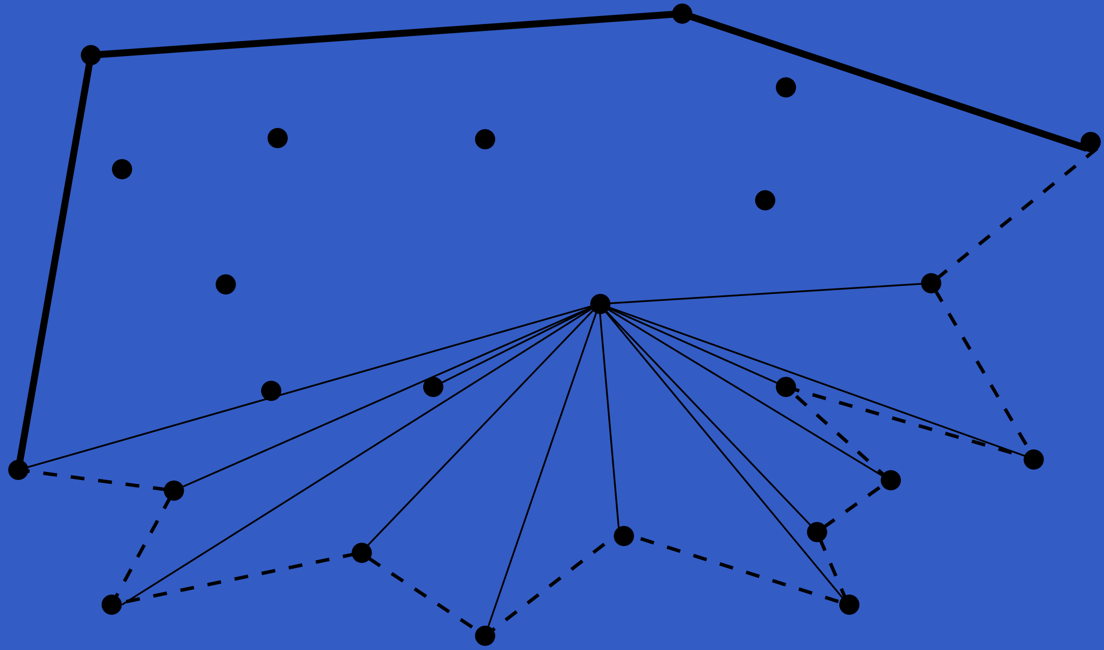
# Convex Hull of a set of points

- Definition: Given a set $S$ of points in the plane, $CH(S)$ is the smallest convex set containing $S$.
- Construction: Divide and conquer, Sweep line, Incremental



# Convex Hull of a set of points

- Definition: Given a set $S$ of points in the plane, $CH(S)$ is the smallest convex set containing $S$.
- Construction: Divide and conquer, Sweep line, Incremental, Graham Scan, etc.

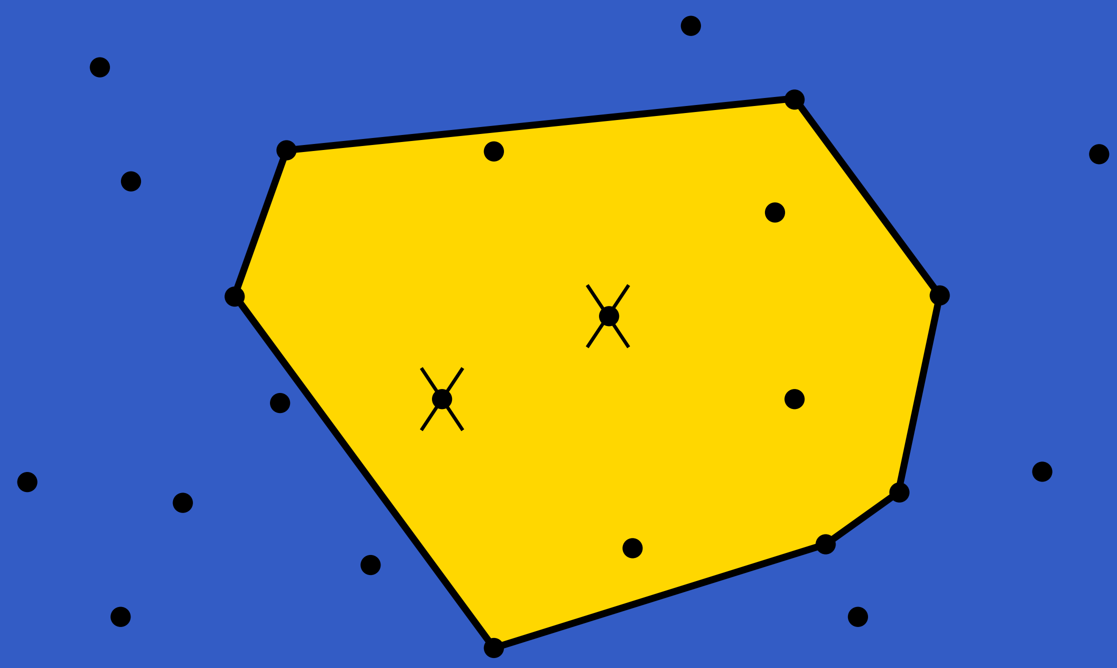


# Parallelization

- By hand: Developing specialized algorithms for each problem and architecture.
- Automatic: Rely on the compiler to obtain a parallel version of the incremental sequential algorithm.

# Speculative Parallelization

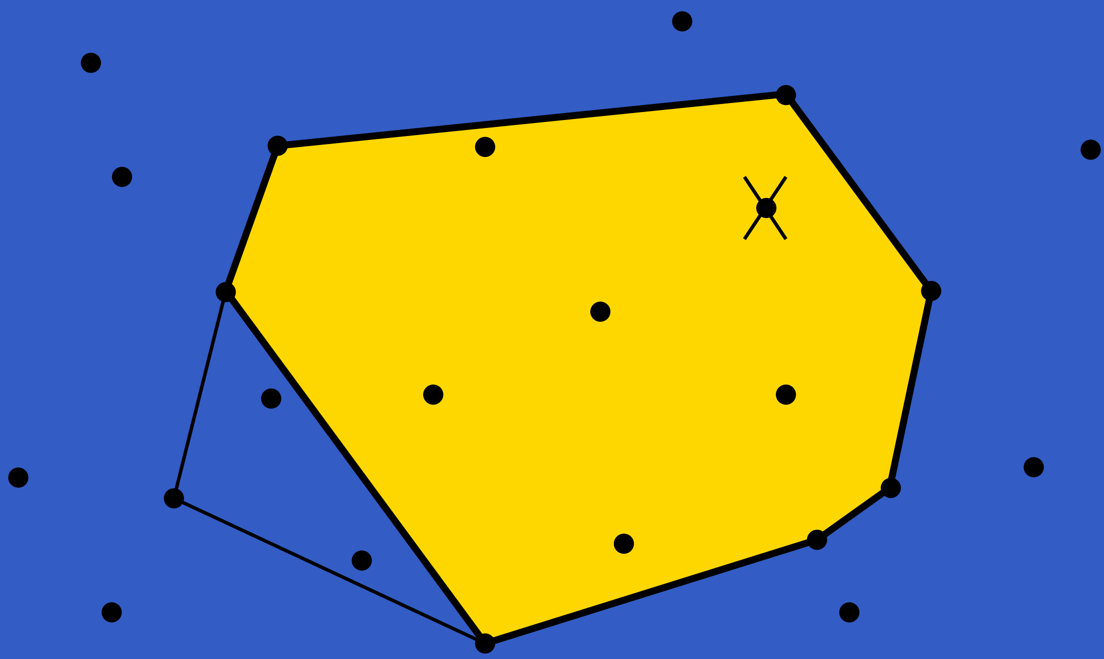
- Optimistic parallel execution of an iterative algorithm



Points inside the current solution. No dependencies found.

# Speculative Parallelization

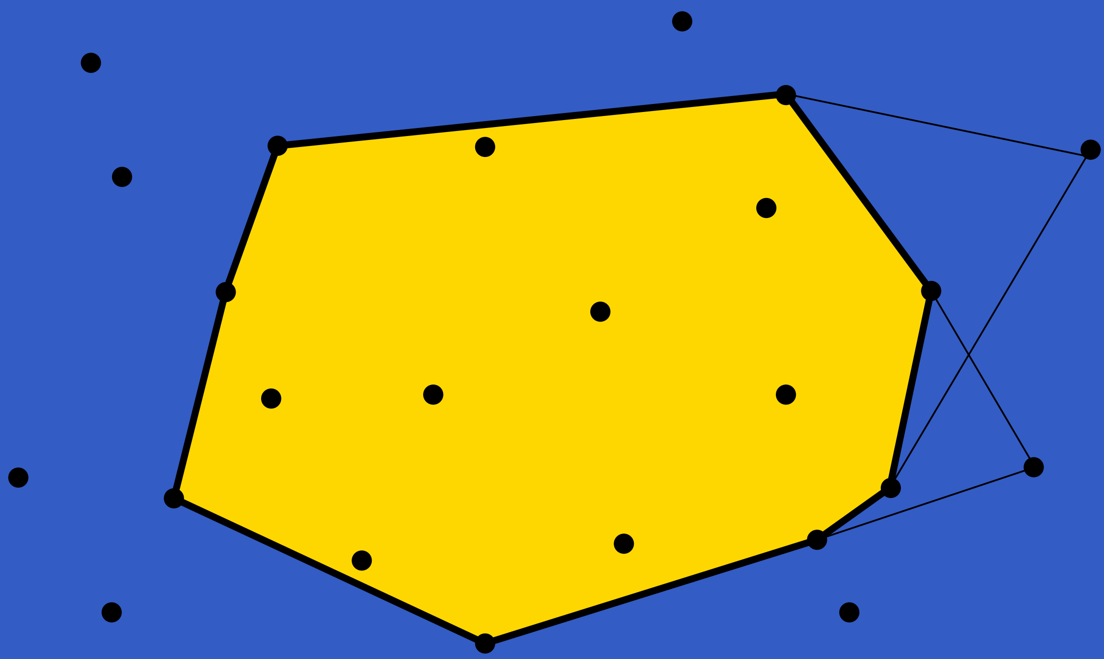
- Optimistic parallel execution of an iterative algorithm



One point inside, one point outside. No dependencies found.

# Speculative Parallelization

- Optimistic parallel execution of an iterative algorithm
- Squashes are produced when dependencies are found at runtime



Points outside the current solution. Dependencies found!

# Speculative Parallelization vs. the Convex Hull

Why being optimistic on a parallel execution of the Convex Hull?

- Output structure depends usually on a small portion of input points

# Speculative Parallelization vs. the Convex Hull

Why being optimistic on a parallel execution of the Convex Hull?

- Output structure depends usually on a small portion of input points

- Given the solution, ALL other input points could be processed in parallel

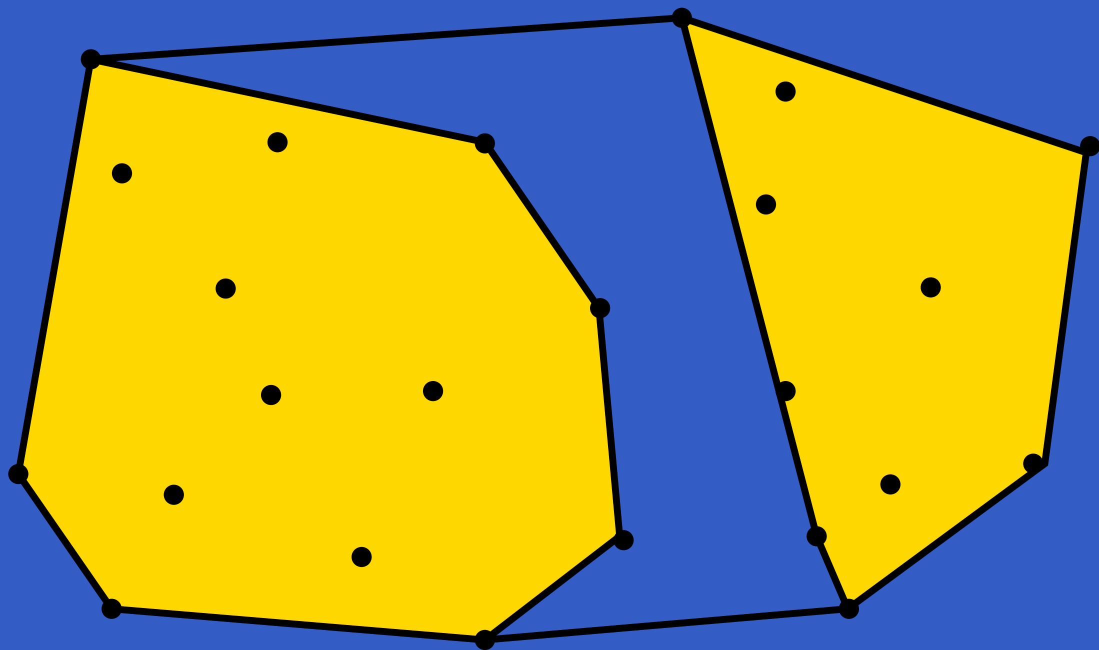
# Speculative Parallelization vs. the Convex Hull

Why being optimistic on a parallel execution of the Convex Hull?

- Output structure depends usually on a small portion of input points
- Given the solution, ALL other input points could be processed in parallel
- Iterative algorithms with a lot of inherent parallelism are excellent candidates for speculative parallelization

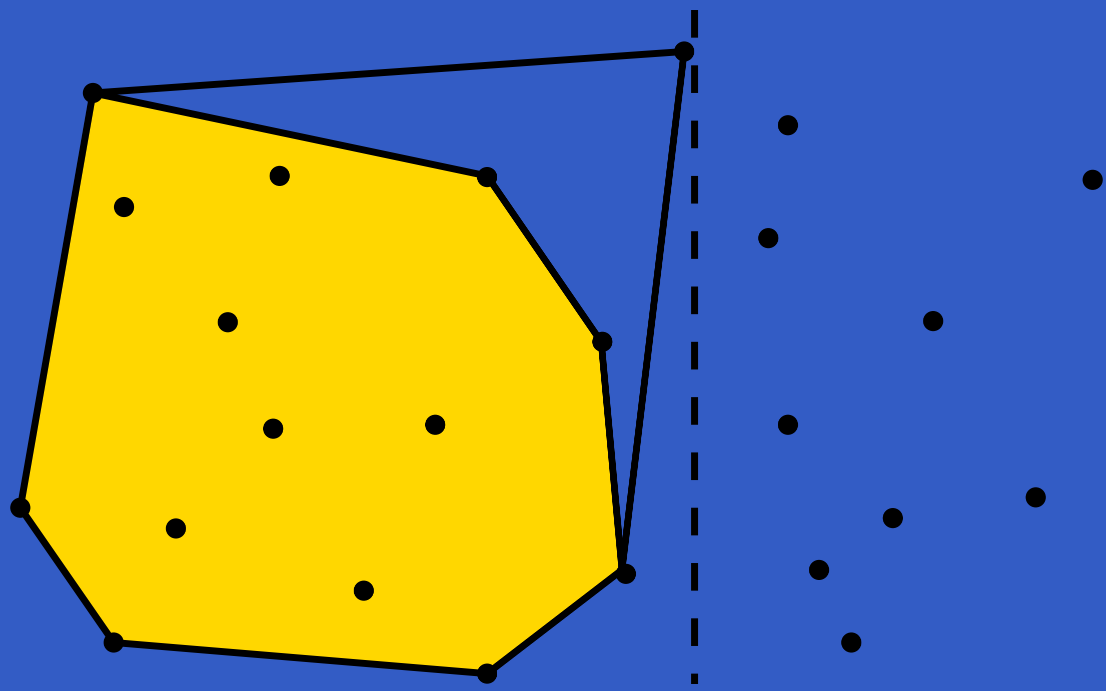
# Convex Hull of a set of points (II)

- Construction: Divide and conquer



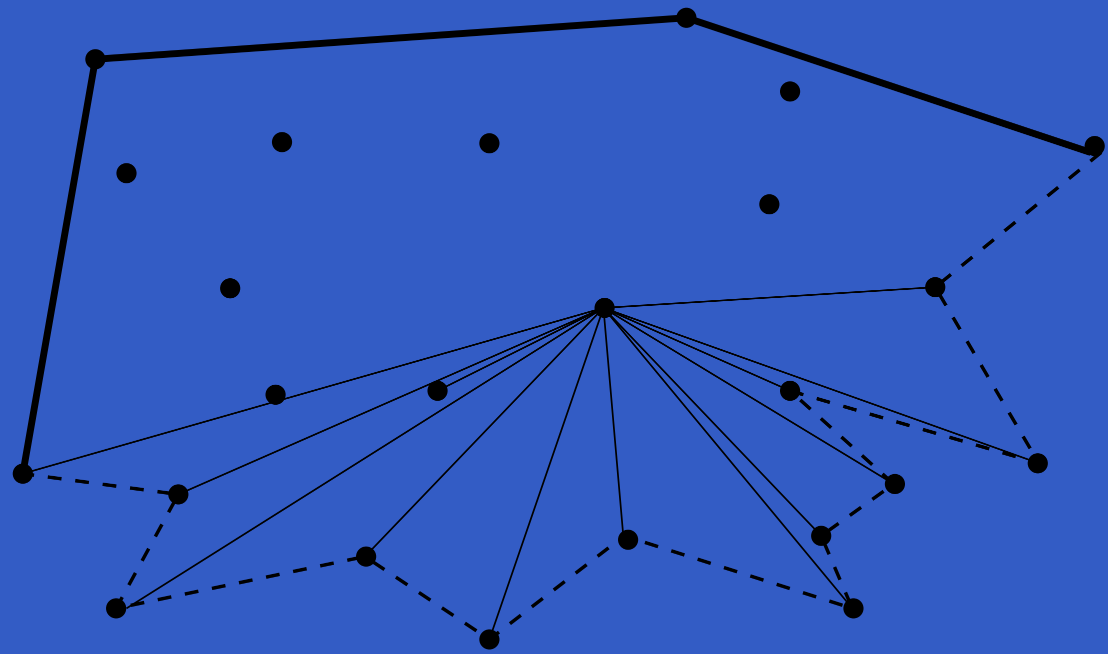
# Convex Hull of a set of points (II)

- Construction: Divide and conquer, Sweep line



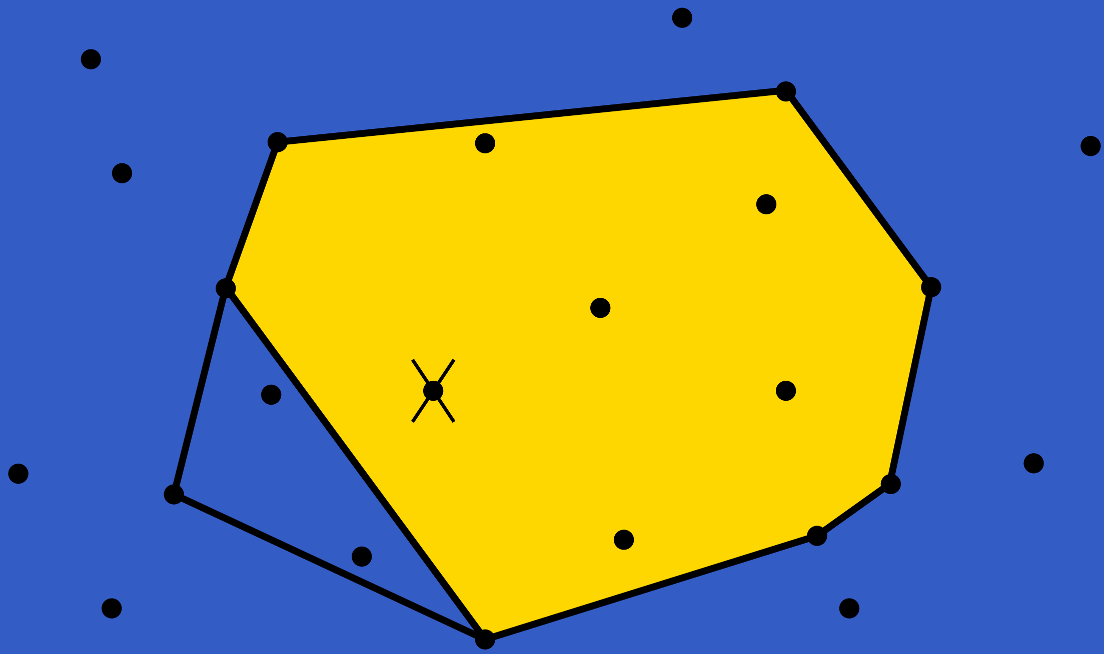
# Convex Hull of a set of points (II)

- Construction: Divide and conquer, Sweep line, Graham Scan



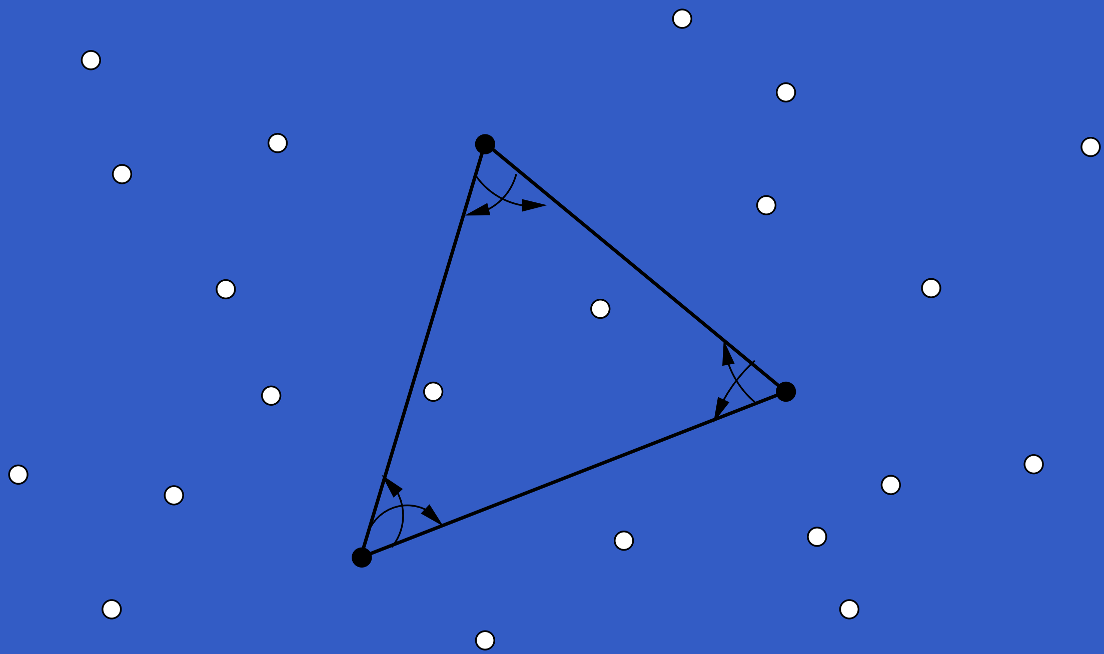
# Convex Hull of a set of points (II)

- Construction: Divide and conquer, Sweep line, Graham Scan, Incremental



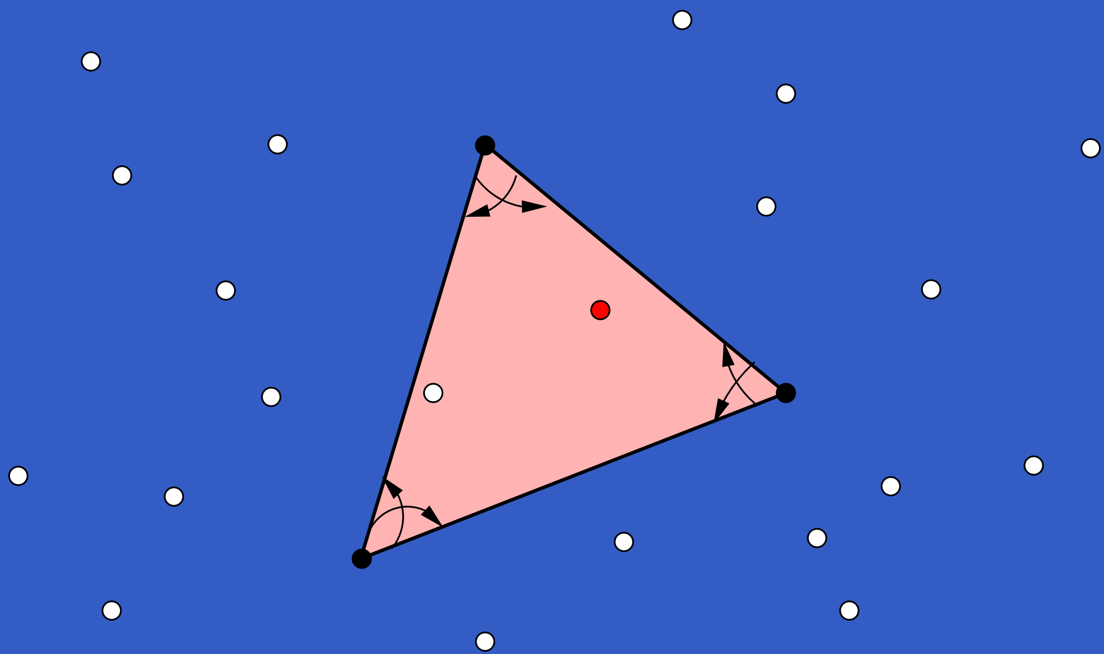
# Clarkson's Incremental Randomized Algorithm (I)

- **First three points build first triangle**
- Locate next point in existing structure
- If inside, process next point
- If outside, connect to structure and process next point



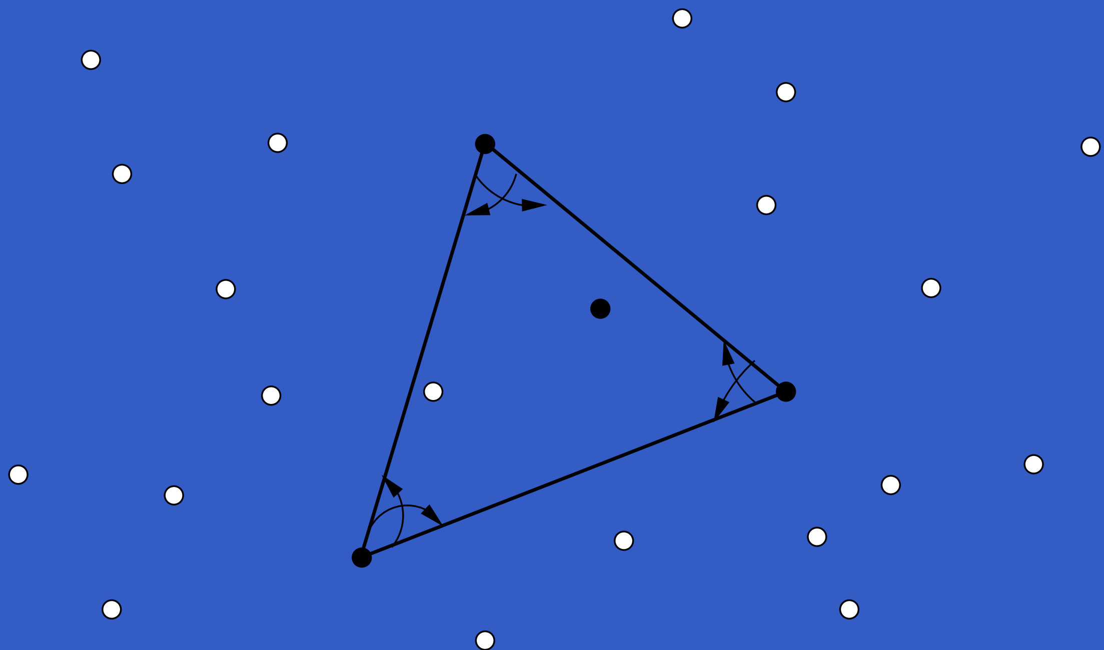
# Clarkson's Incremental Randomized Algorithm (I)

- First three points build first triangle
- **Locate next point in existing structure**
- If inside, process next point
- If outside, connect to structure and process next point



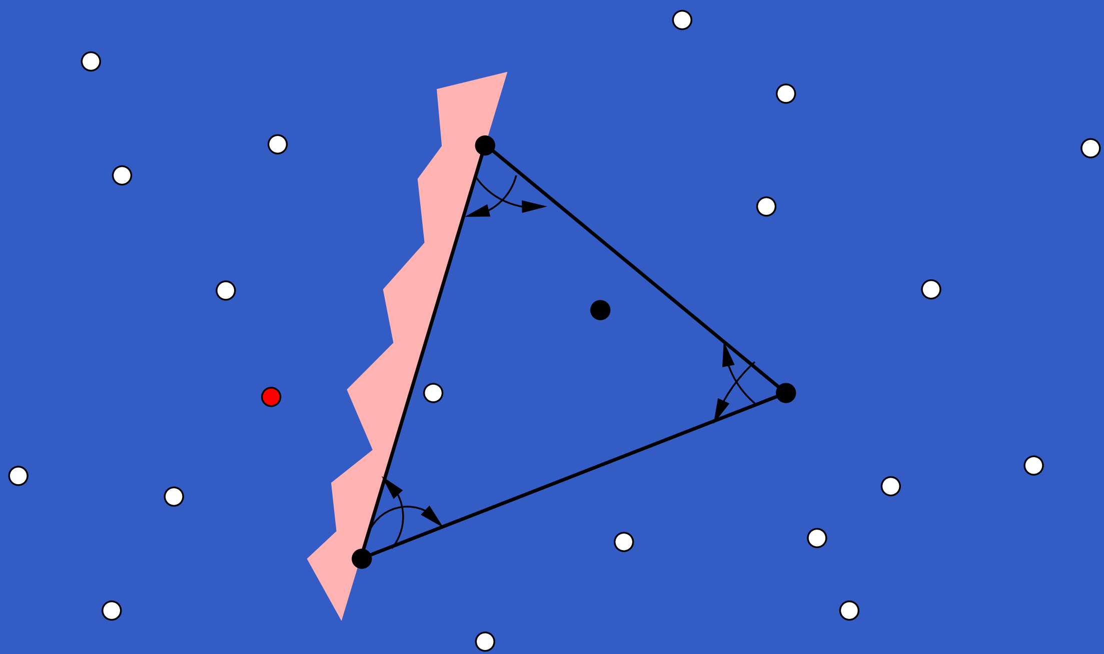
# Clarkson's Incremental Randomized Algorithm (I)

- First three points build first triangle
- Locate next point in existing structure
- **If inside, process next point**
- If outside, connect to structure and process next point



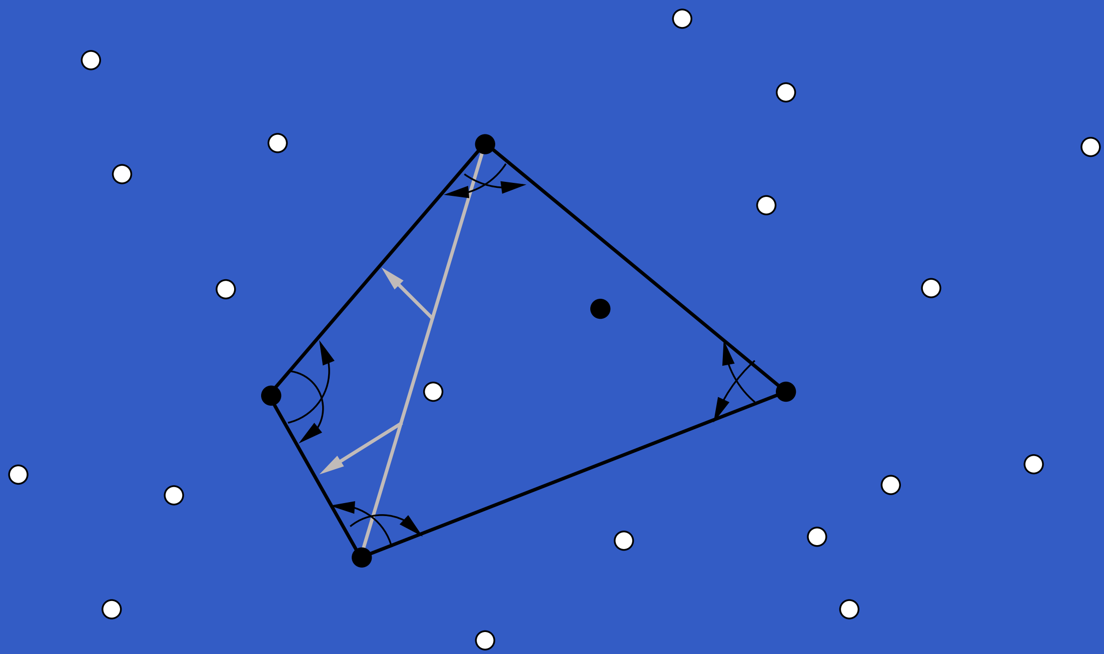
# Clarkson's Incremental Randomized Algorithm (I)

- First three points build first triangle
- **Locate next point in existing structure**
- If inside, process next point
- If outside, connect to structure and process next point



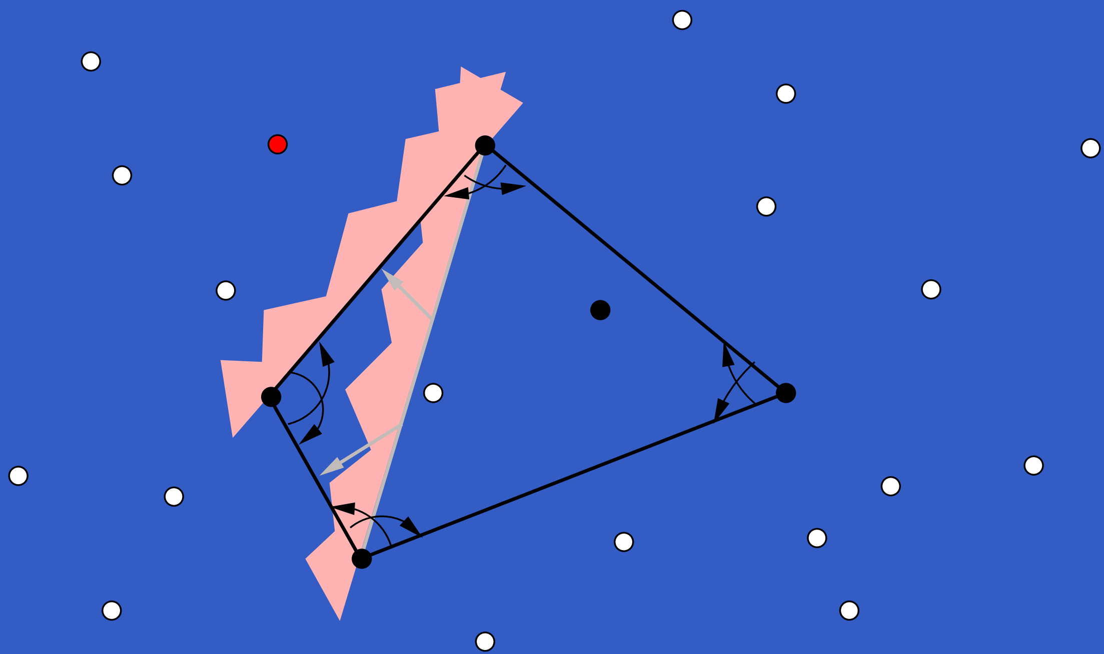
# Clarkson's Incremental Randomized Algorithm (I)

- First three points build first triangle
- Locate next point in existing structure
- If inside, process next point
- **If outside, connect to structure and process next point**



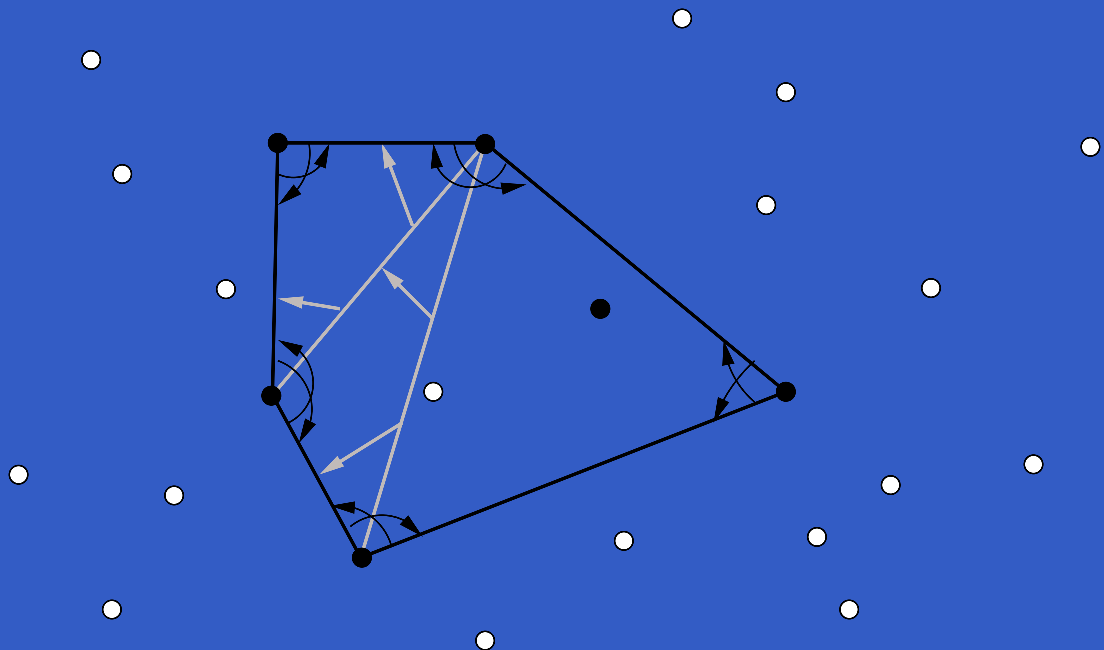
# Clarkson's Incremental Randomized Algorithm (I)

- First three points build first triangle
- **Locate next point in existing structure**
- If inside, process next point
- If outside, connect to structure and process next point



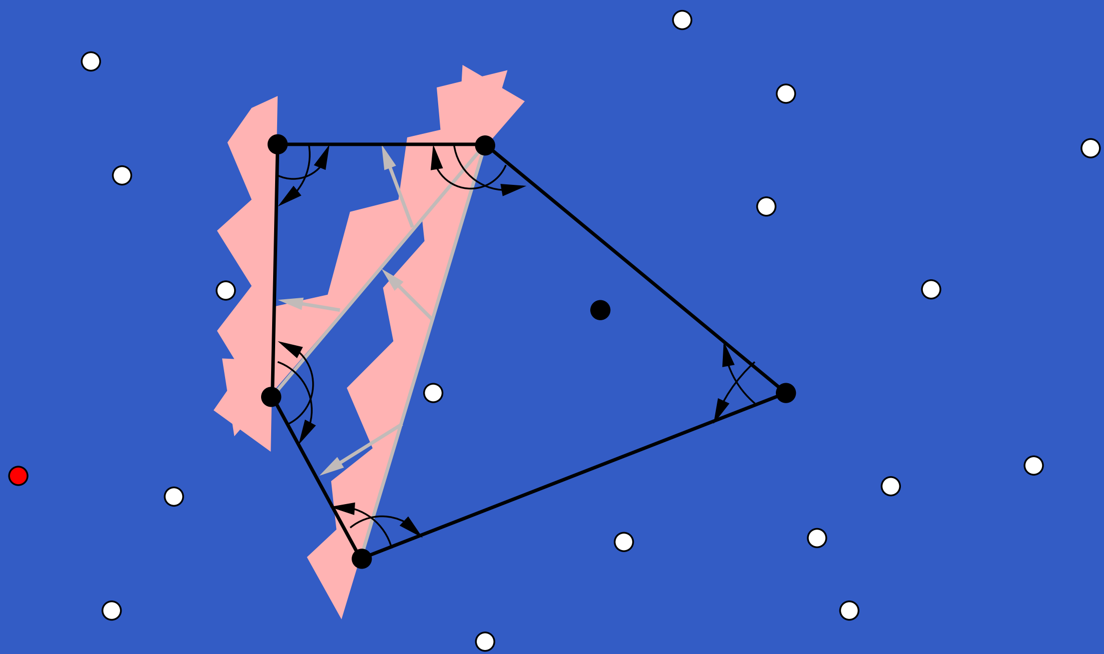
# Clarkson’s Incremental Randomized Algorithm (I)

- First three points build first triangle
- Locate next point in existing structure
- If inside, process next point
- **If outside, connect to structure and process next point**



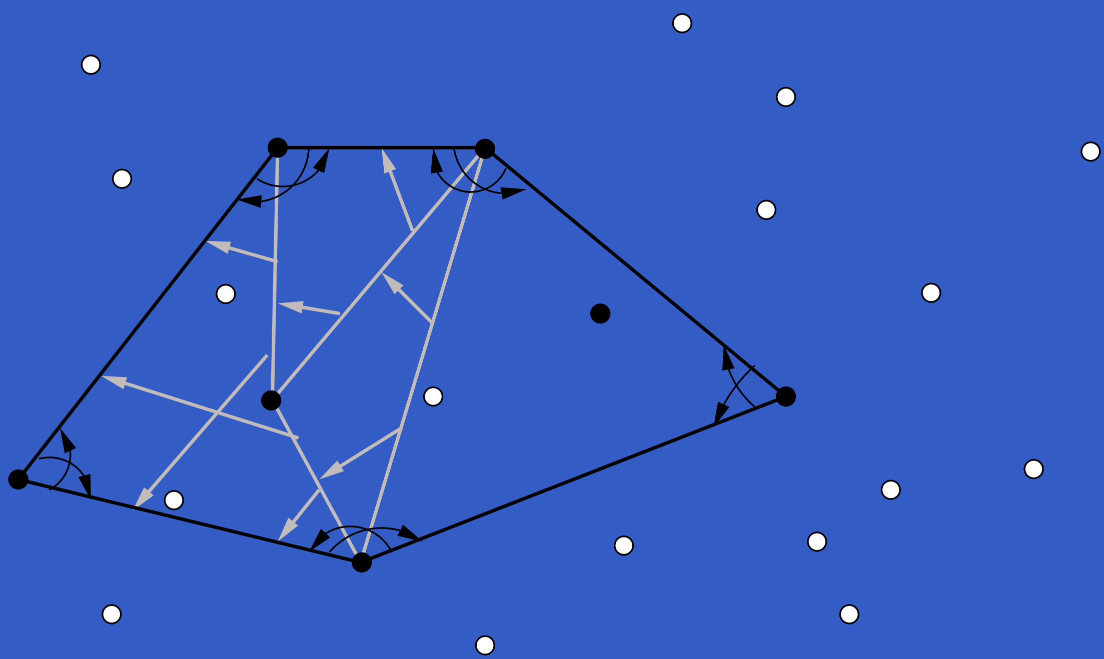
# Clarkson’s Incremental Randomized Algorithm (I)

- First three points build first triangle
- **Locate next point in existing structure**
- If inside, process next point
- If outside, connect to structure and process next point



# Clarkson’s Incremental Randomized Algorithm (I)

- First three points build first triangle
- Locate next point in existing structure
- If inside, process next point
- **If outside, connect to structure and process next point**



# Clarkson's Incremental Randomized Algorithm (II)

- First three points build first triangle $\in O(1)$
- **Locate next point in existing structure**
- If inside, process next point $\in O(1)$
- If outside, connect to structure and process next point $\in O(1)$

The expected number of edges that *see* point $p_{r+1}$ is in $O\log(r)$.

# Clarkson's Incremental Randomized Algorithm (II)

- First three points build first triangle $\in O(1)$
- Locate next point in existing structure $\in O(\log n)$
- If inside, process next point $\in O(1)$
- If outside, connect to structure and process next point $\in O(1)$

$$\boxed{\text{The expected running time is } O(n \log n)}$$

# Setup: Input Sets and System

**Input sets**

- CGAL 2.4 random point generator and shuffle functions with integer coordinates
- 10 and 40 Million points in a square
- 10 and 40 Million points in a disk

# Setup: Input Sets and System

**Input sets**

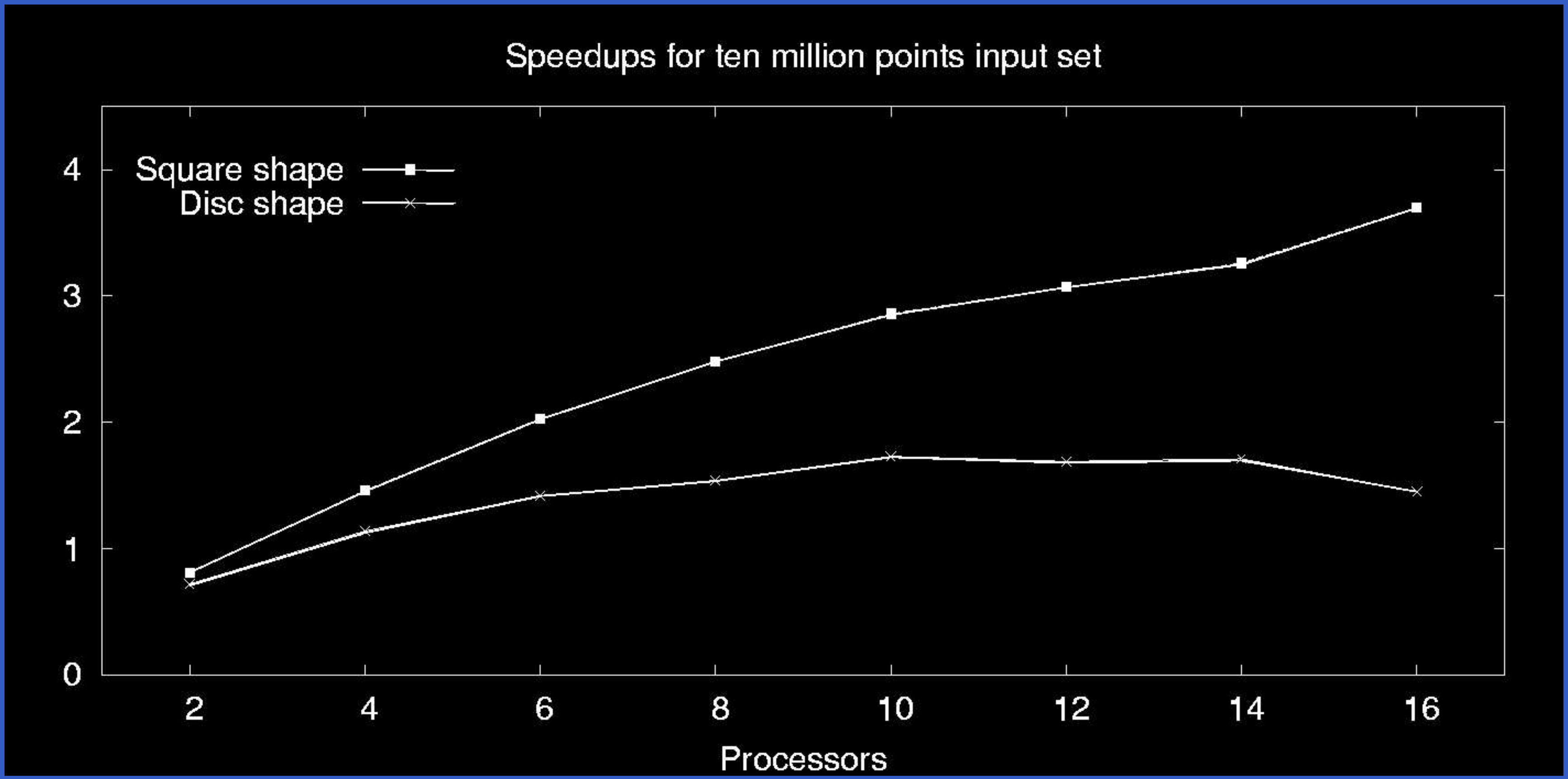
- CGAL 2.4 random point generator and shuffle functions with integer coordinates
- 10 and 40 Million points in a square
- 10 and 40 Million points in a disk

**System**

- Sun Fire 15K symmetric multiprocessor (SMP) with SunOS 5.8
- 900MHz UltraSparc-III processors with 1 GByte of shared memory each
- Forte Developer 7 Fortran 95 compiler

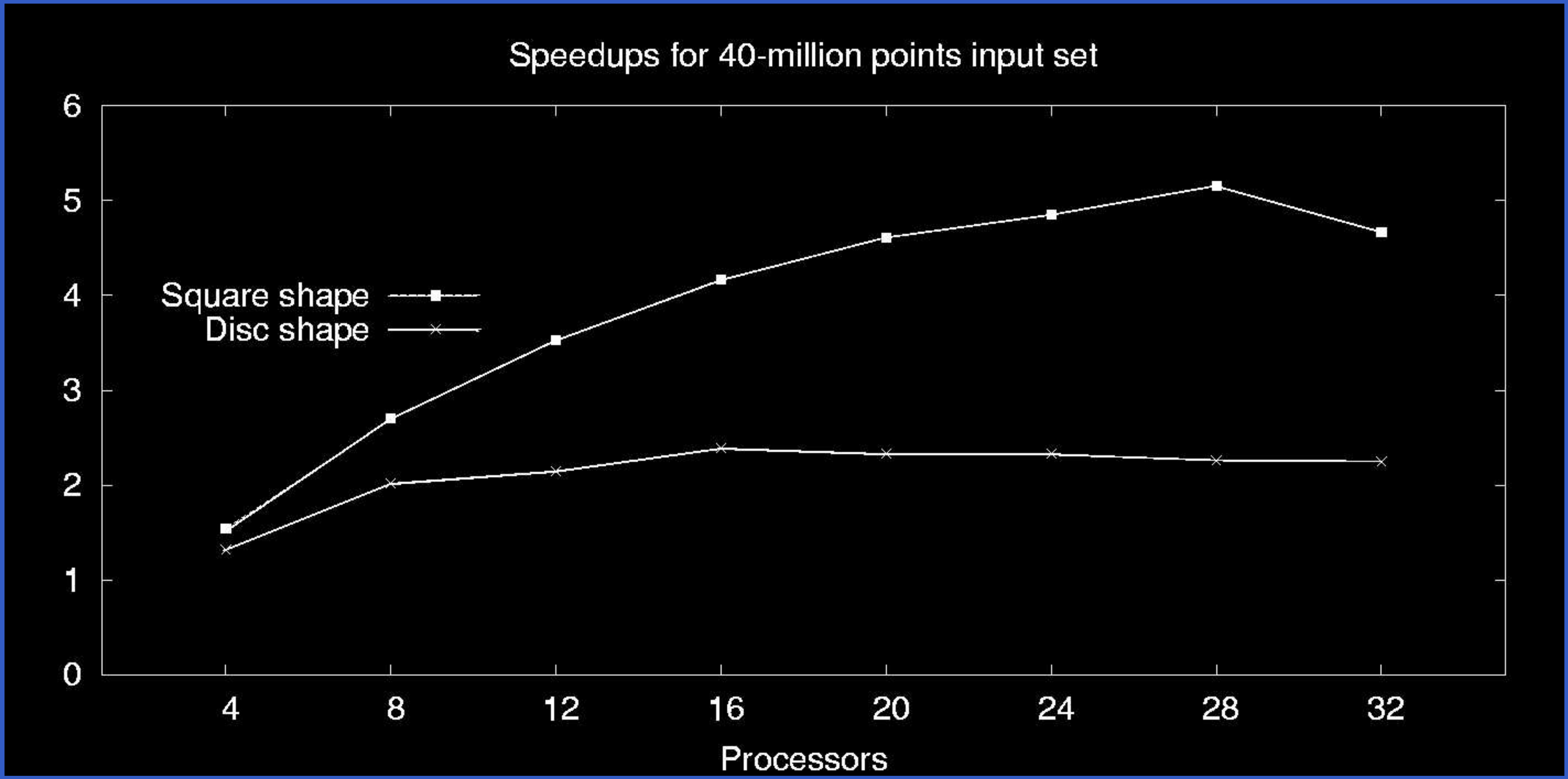
# Obtained speedups

- *Small* input set, with 10 MP



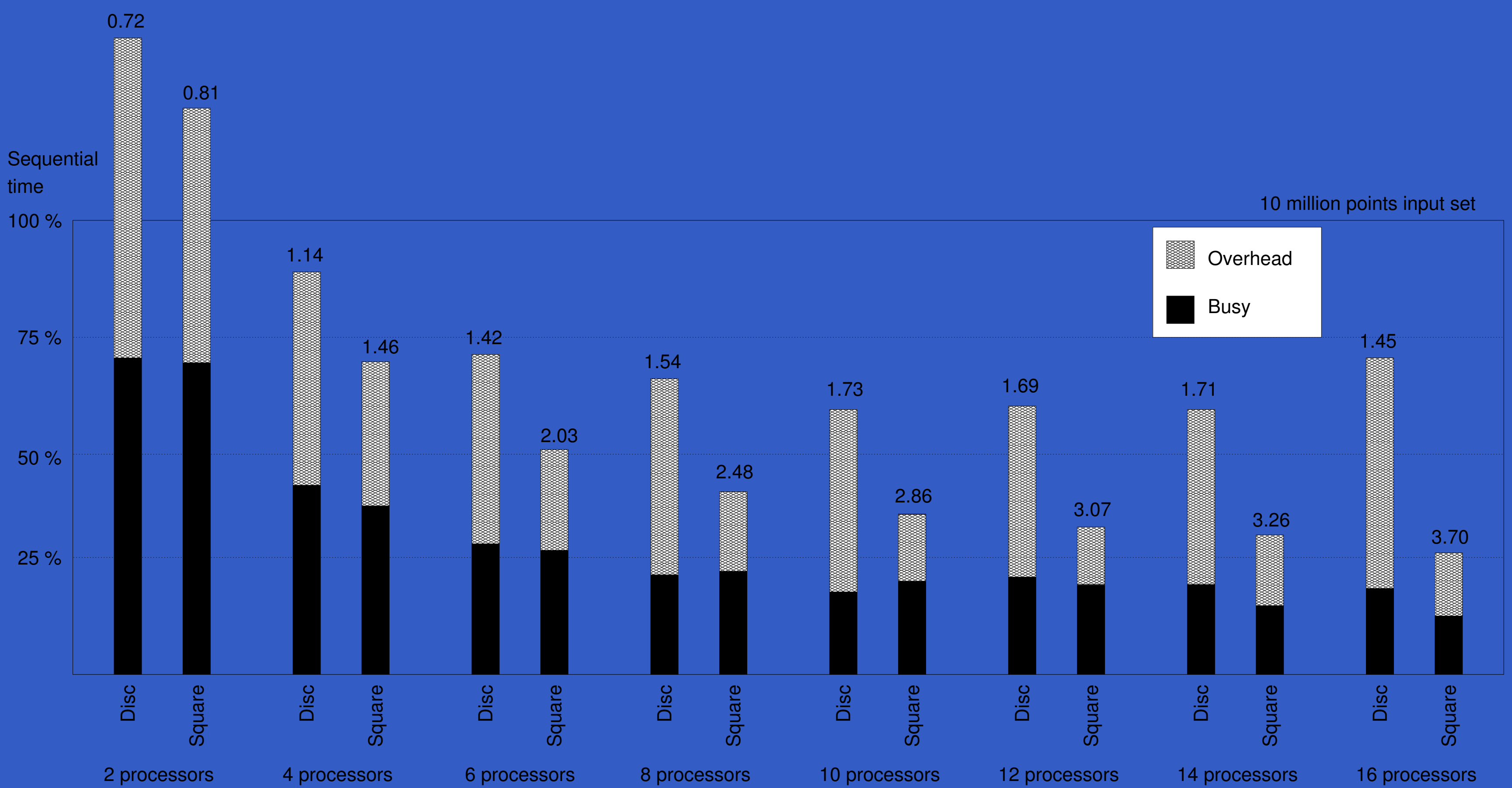
# Obtained speedups

- *Small* input set, with 10 MP
- *Big* input set, with 40 MP



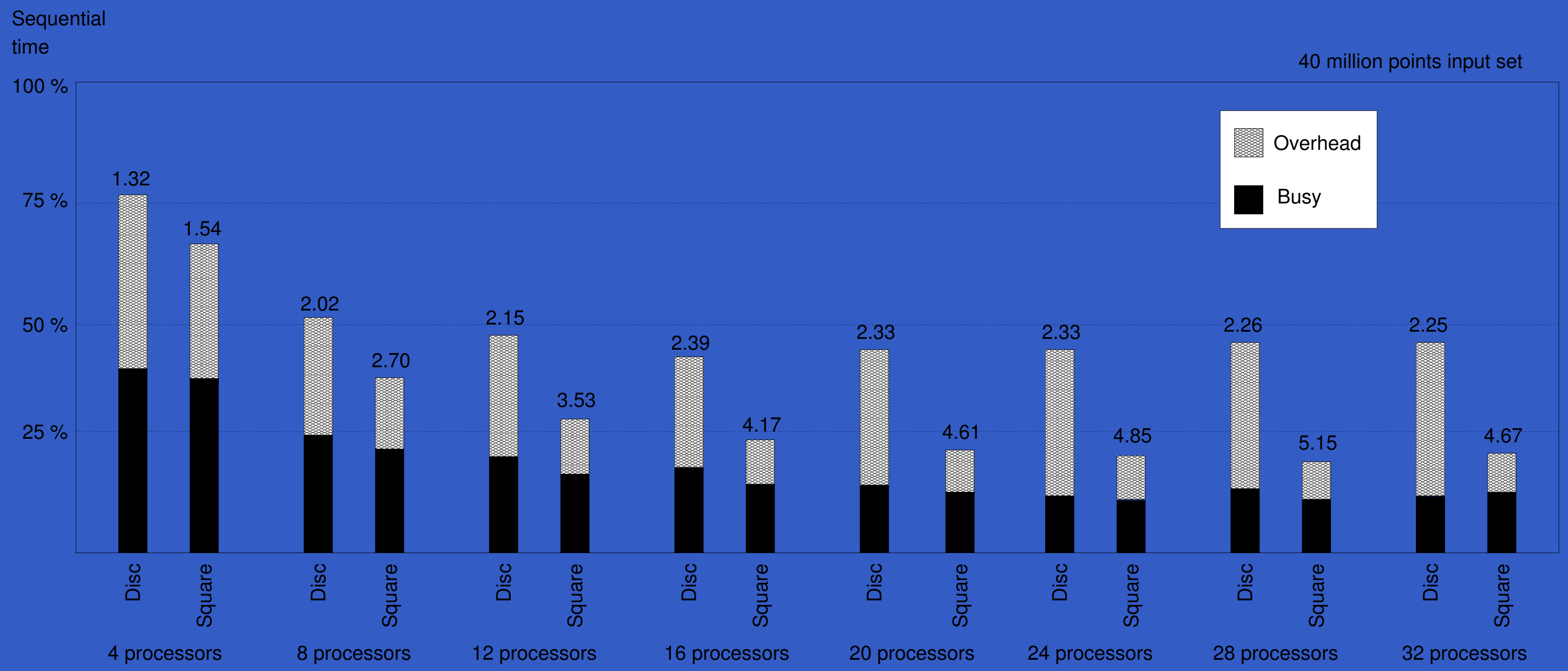
# Results

- Overhead grows with the number of processors



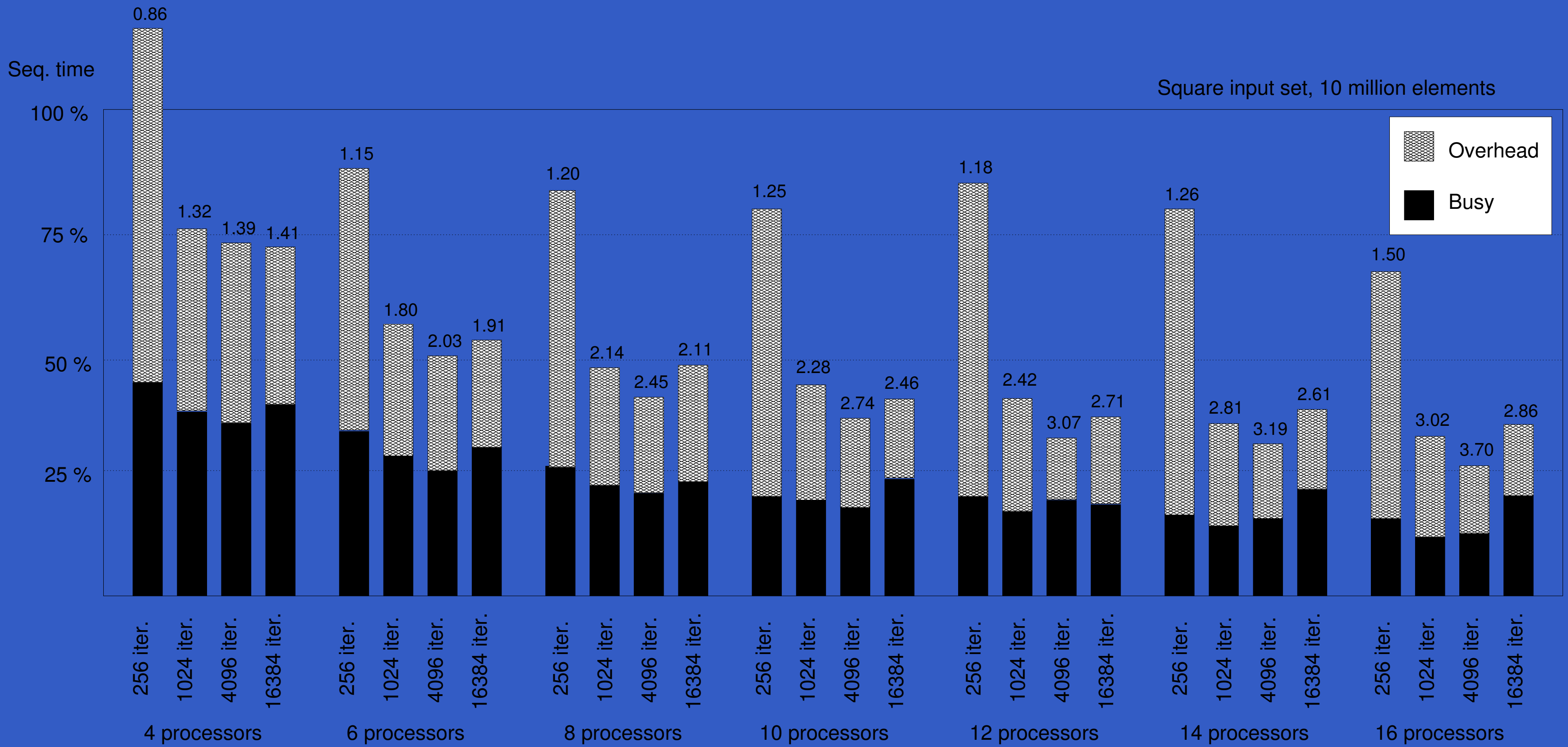
# Results

- Overhead grows with the number of processors
- Bigger sets obtain better speedups



# Results

- Overhead grows with the number of processors
- Bigger sets obtain better speedups
- Performance also depends on the block size



# Conclusions

- Speculative parallelization is a valid technique to extract inherent parallelism from randomized iterative algorithms

# Conclusions

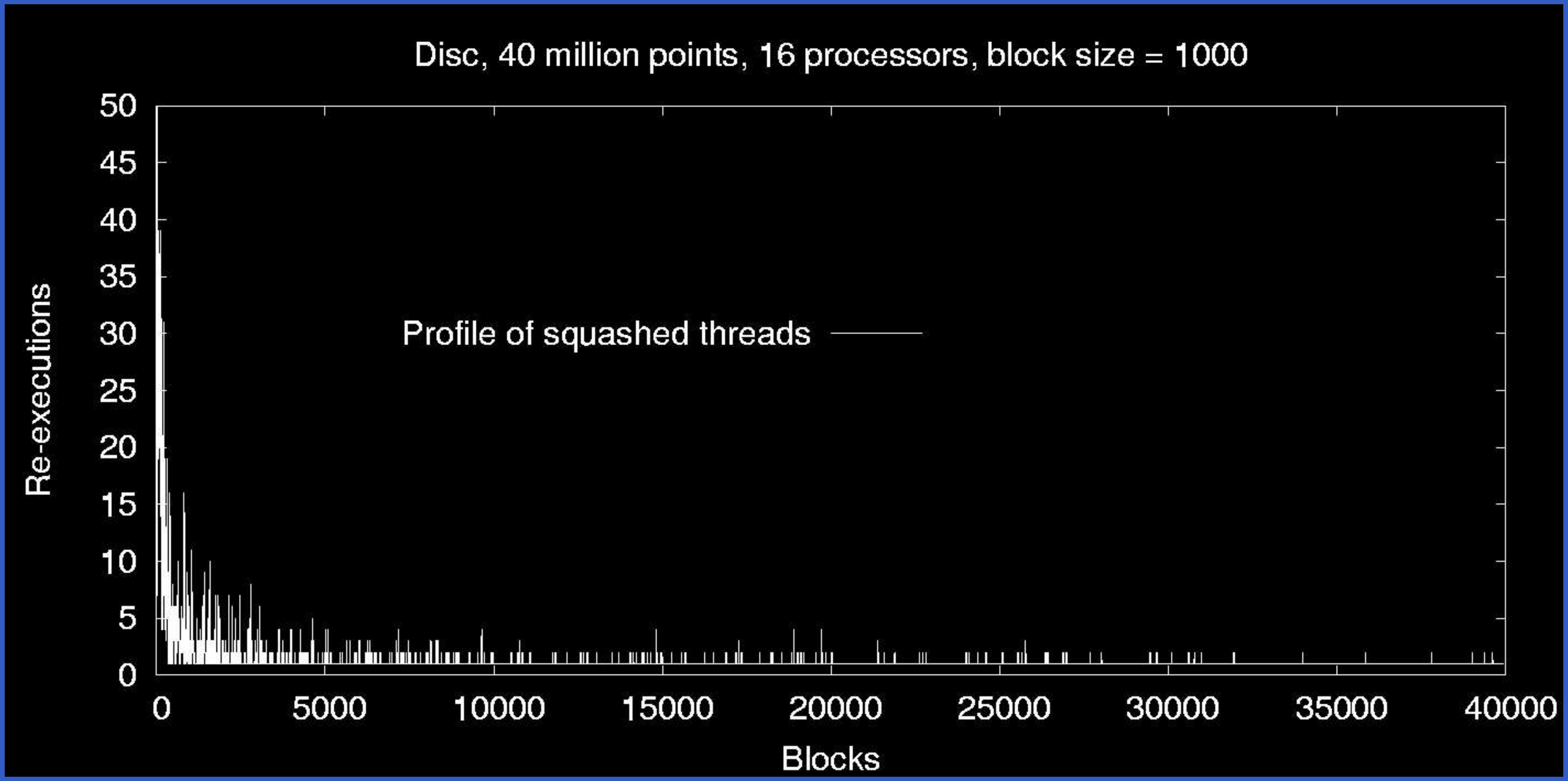
- Speculative parallelization is a valid technique to extract inherent parallelism from randomized iterative algorithms
- No manual parallelization is needed: speculative calls may be inserted automatically

# Conclusions

- Speculative parallelization is a valid technique to extract inherent parallelism from randomized iterative algorithms
- No manual parallelization is needed: speculative calls may be inserted automatically
- Low-cost technique: No need of hardware changes to the processor and/or memory hierarchy

# Ongoing and future work

- Squash graphics show the *hot spots* where many dependencies are found



# Ongoing and future work

- Squash graphics show the *hot spots* where many dependencies are found
- Dynamic scheduling can be applied to decrease the number of dependencies and reexecutions of big blocks