Advanced Algorithms – COMS31900

Approximation algorithms part two

more constant factor approximations

Raphaël Clifford

Slides by Benjamin Sach

Approximation Algorithms Recap

An algorithm A is an α-approximation algorithm for problem P if,

- A runs in polynomial time
- A always outputs a solution with value s within an α factor of Opt

Here P is an optimisation problem with optimal solution of value Opt

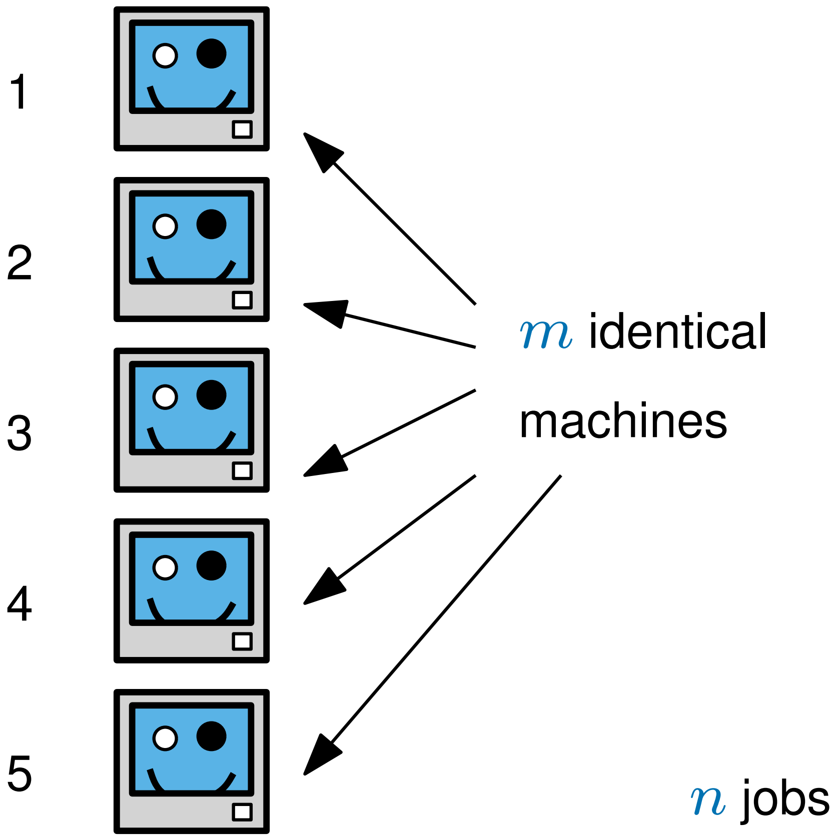
- If P is a *maximisation* problem, $\frac{\mathrm{Opt}}{\alpha} \leqslant s \leqslant \mathrm{Opt}$
- If P is a *minimisation* problem, $\mathrm{Opt} \leqslant s \leqslant \alpha \cdot \mathrm{Opt}$

We have seen

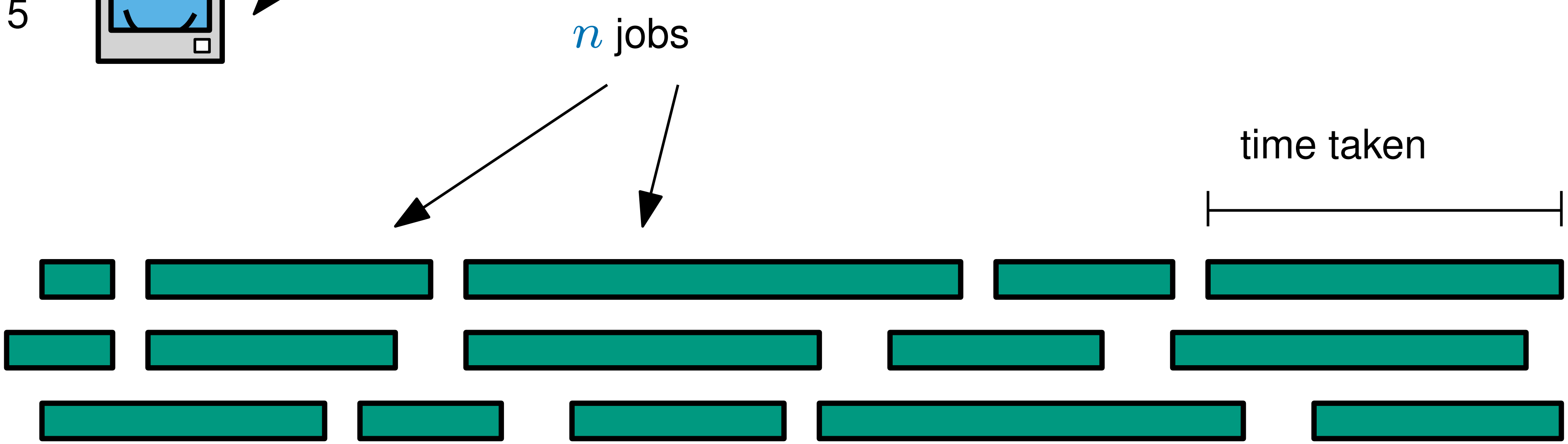
a $3/2$-approximation algorithm for Bin Packing

(and a *faster* 2-approximation)

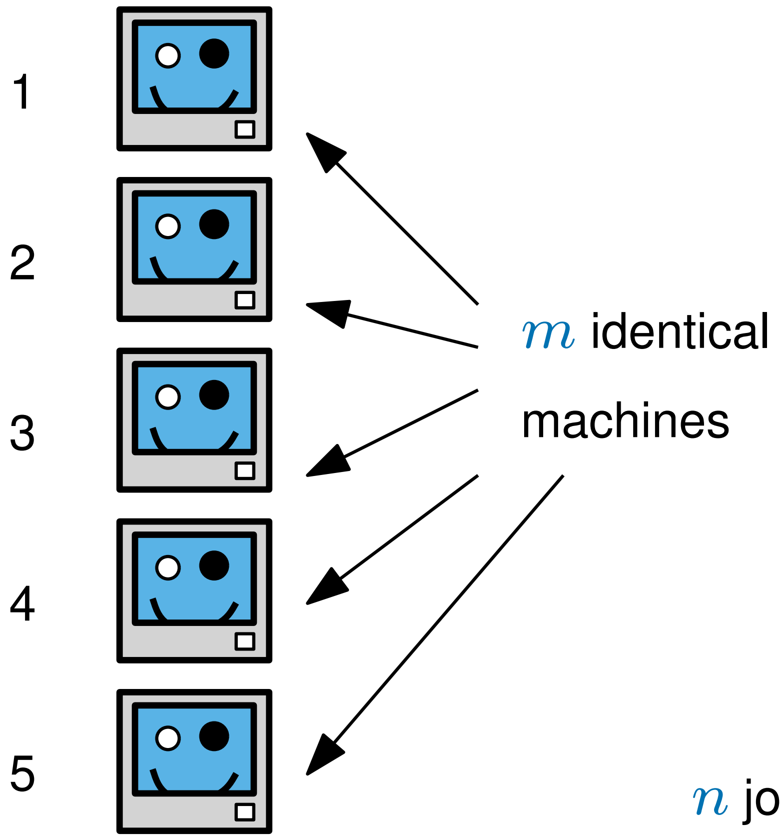
Scheduling Jobs on Parallel Machines



Goal: minimise the (wall-clock) time taken to process all jobs

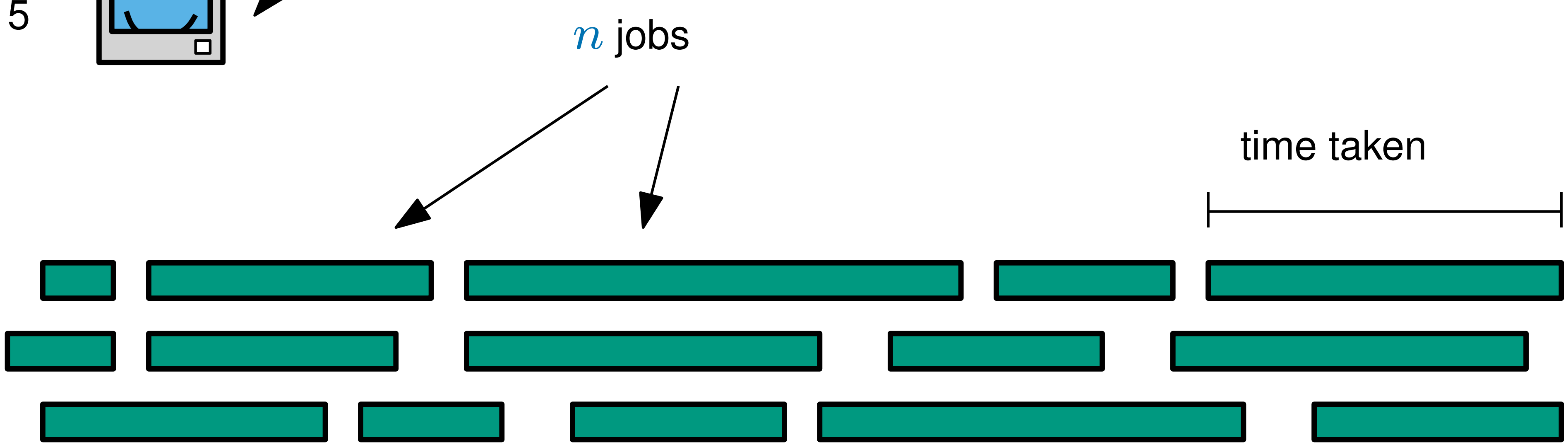


Scheduling Jobs on Parallel Machines

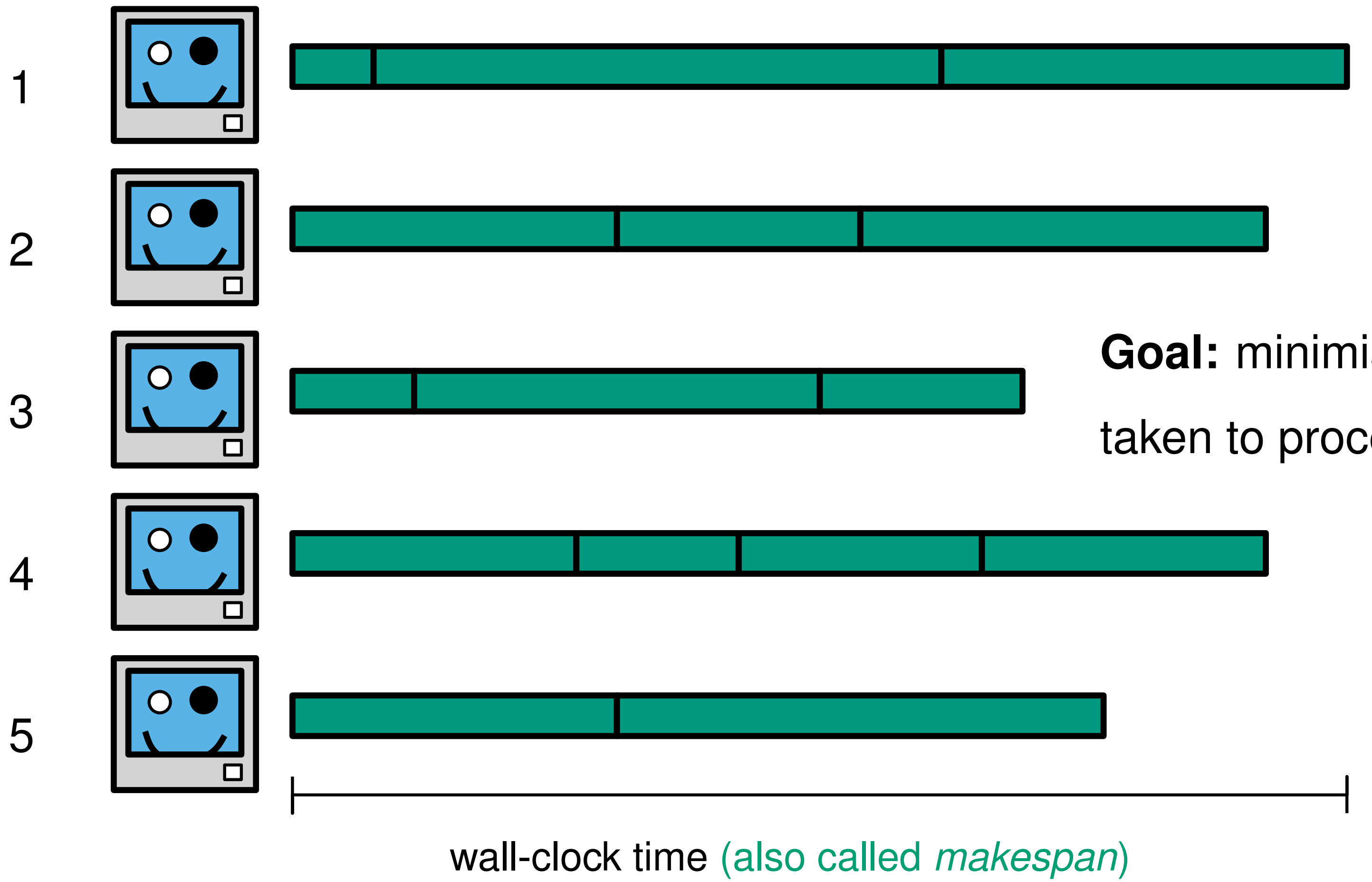


Goal: minimise the (wall-clock) time taken to process all jobs

(it's NP-hard)

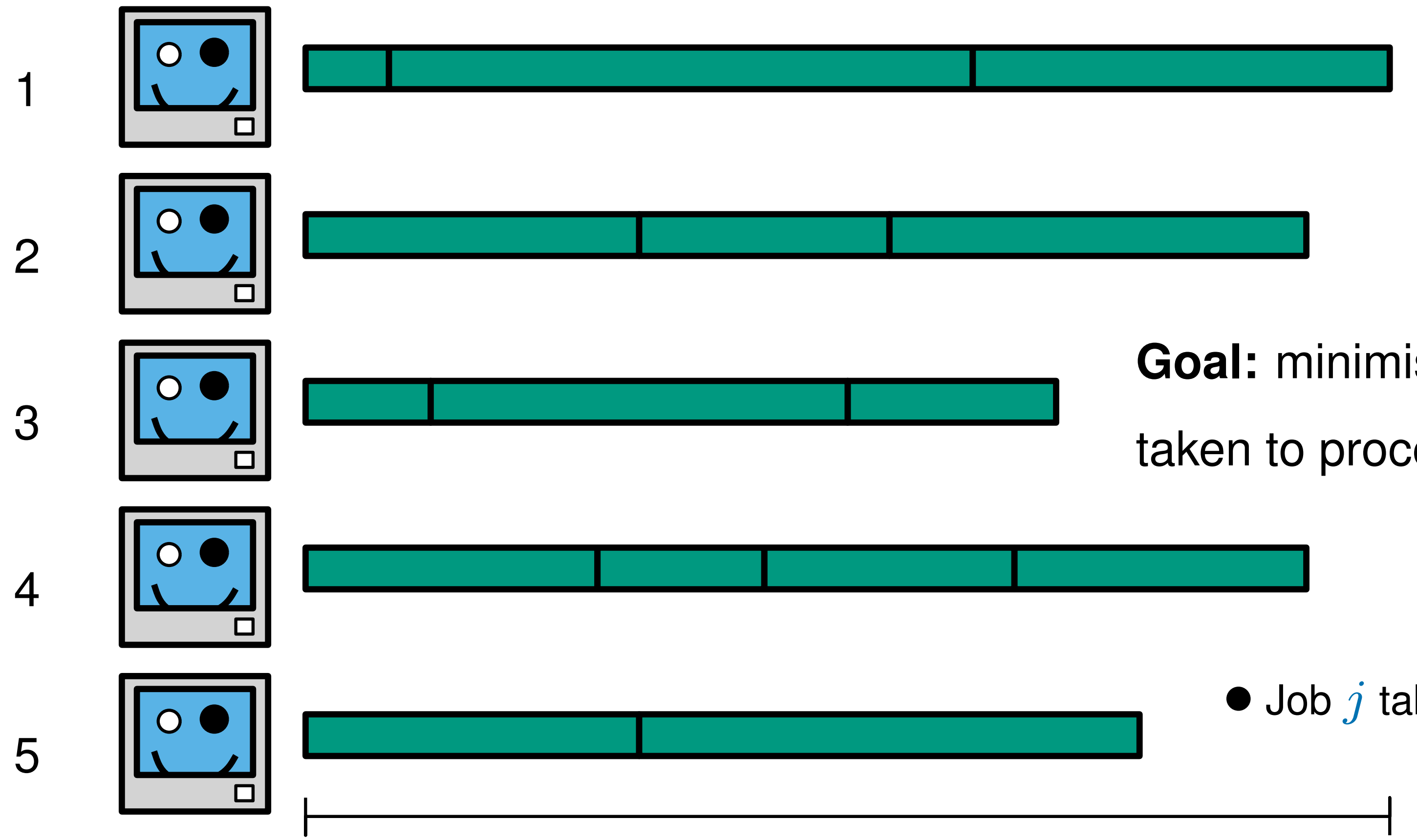


Scheduling Jobs on Parallel Machines



Goal: minimise the (wall-clock) time taken to process all jobs

Scheduling Jobs on Parallel Machines

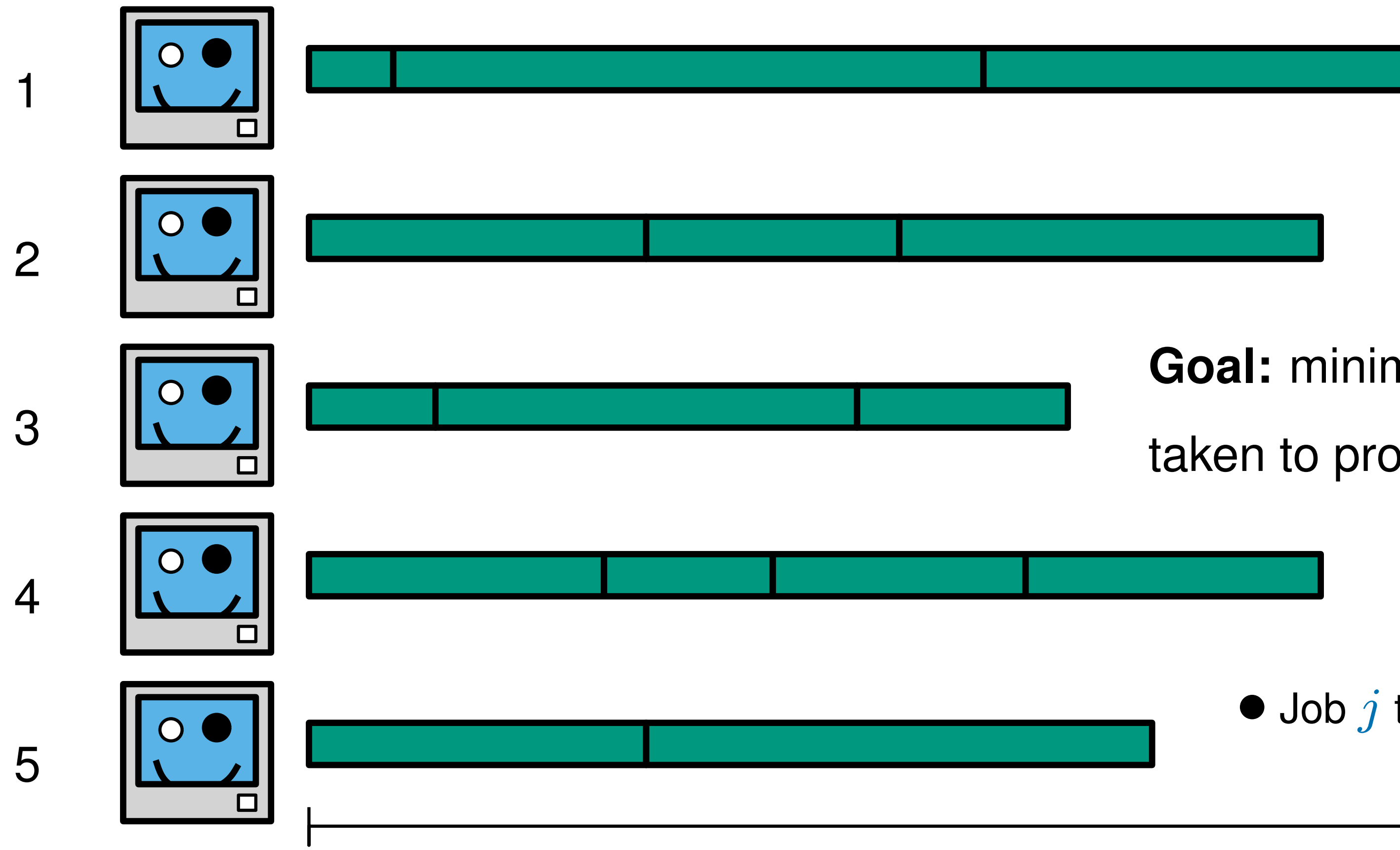


wall-clock time (also called *makespan*)

Goal: minimise the (wall-clock) time taken to process all jobs

- Job j takes t_j time units

Scheduling Jobs on Parallel Machines



Goal: minimise the (wall-clock) time taken to process all jobs

- Job j takes t_j time units

wall-clock time (also called *makespan*)

- We say that $j \in J(i)$ iff job j is assigned to machine i

Scheduling Jobs on Parallel Machines

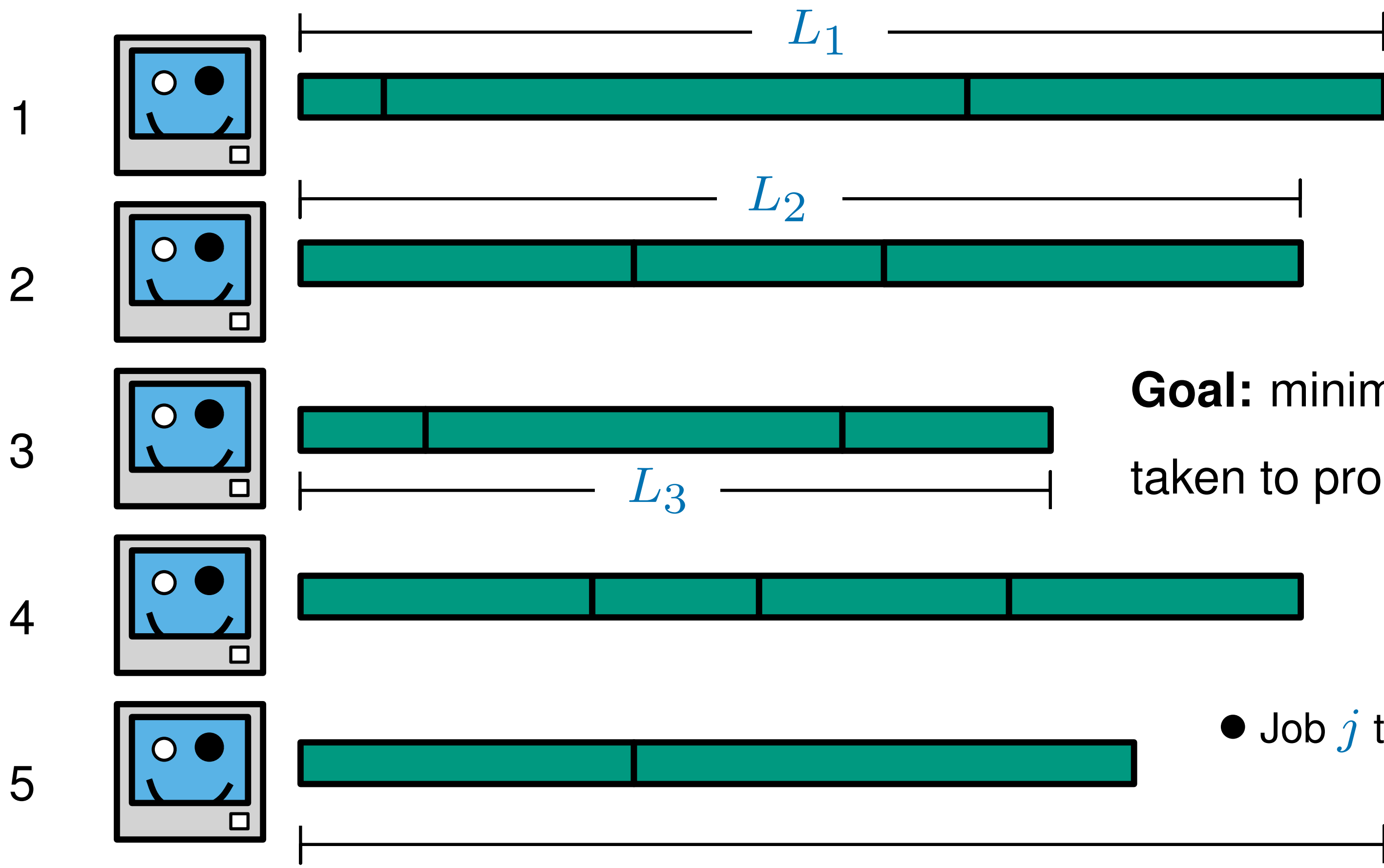
Goal: minimise the (wall-clock) time taken to process all jobs

- Job j takes t_j time units

wall-clock time (also called *makespan*)

- We say that $j \in J(i)$ iff job j is assigned to machine i
- The load of machine i is $L_i = \sum_{j \in J(i)} t_j$

Scheduling Jobs on Parallel Machines



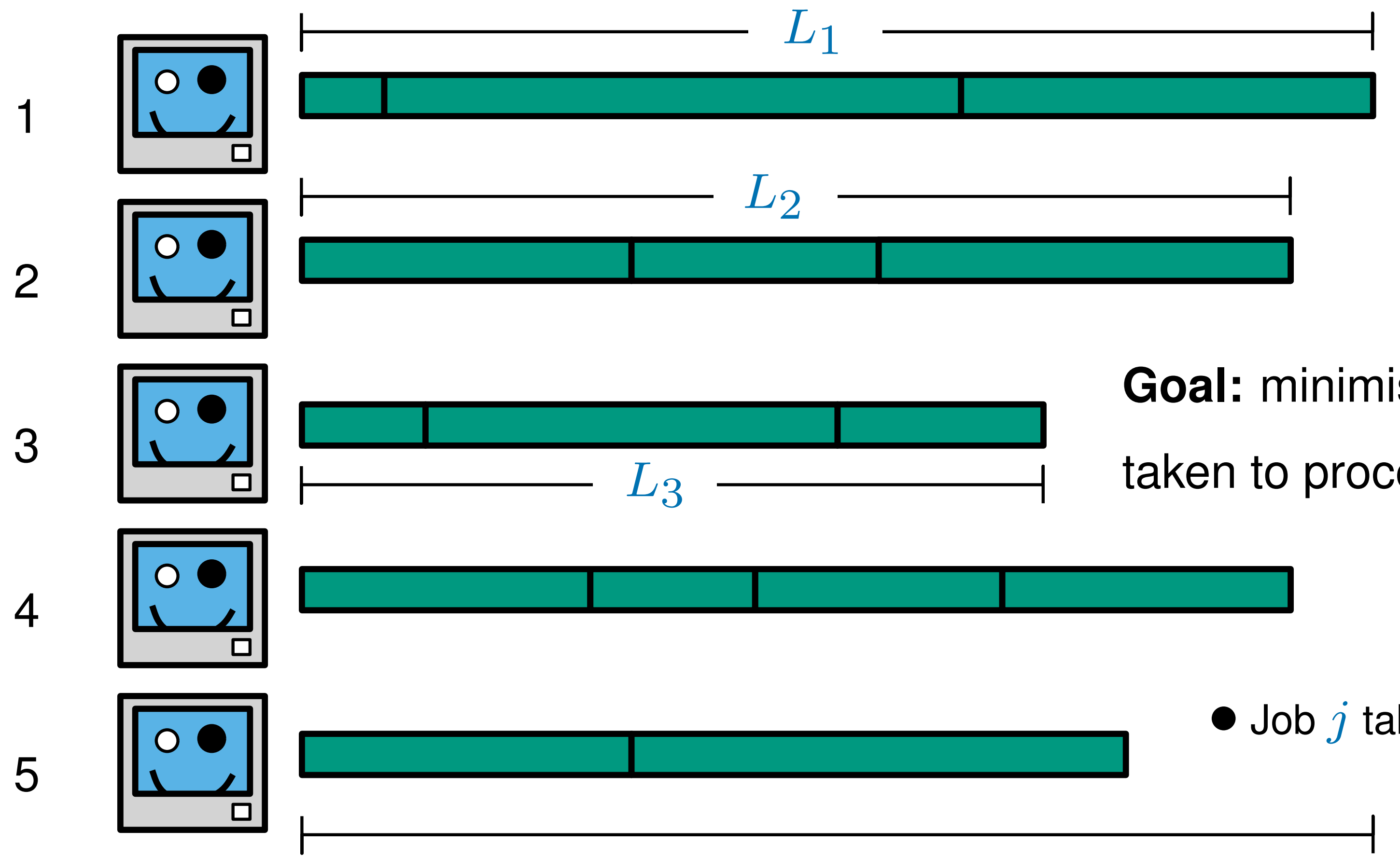
wall-clock time (also called *makespan*)

Goal: minimise the (wall-clock) time taken to process all jobs

- Job j takes t_j time units

- We say that $j \in J(i)$ iff job j is assigned to machine i
- The load of machine i is $L_i = \sum_{j \in J(i)} t_j$

Scheduling Jobs on Parallel Machines



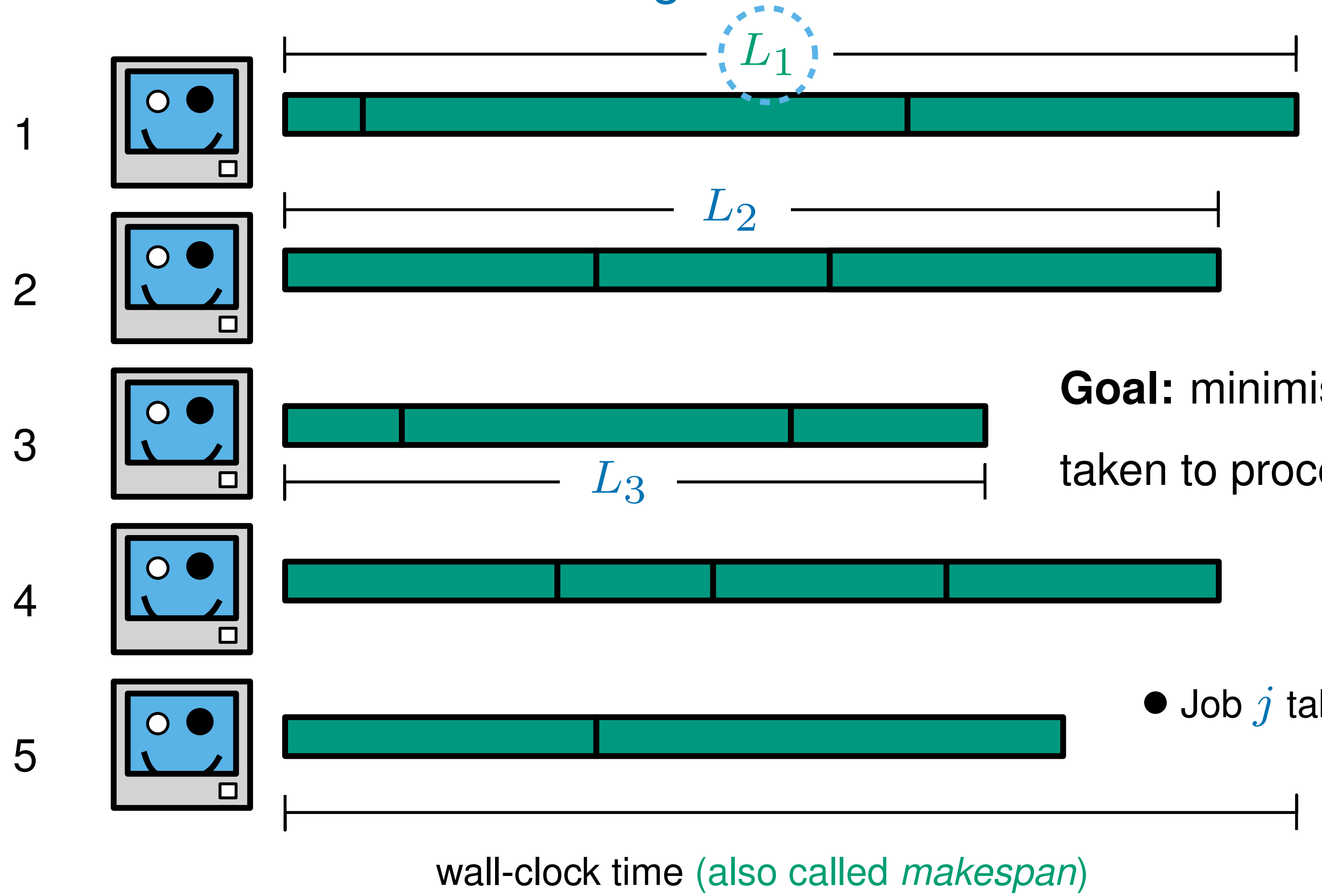
wall-clock time (also called *makespan*)

Goal: minimise the (wall-clock) time taken to process all jobs

- Job j takes t_j time units

- We say that $j \in J(i)$ iff job j is assigned to machine i
- The load of machine i is $L_i = \sum_{j \in J(i)} t_j$
- So the wall-clock time is $\max_i L_i$ (which we want to minimise)

Scheduling Jobs on Parallel Machines



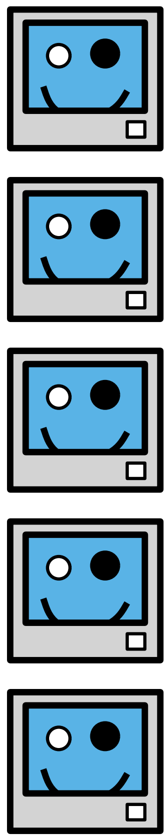
Goal: minimise the (wall-clock) time taken to process all jobs

- Job j takes t_j time units

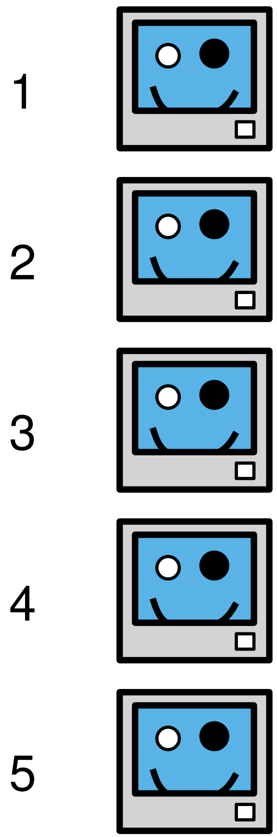
- We say that $j \in J(i)$ iff job j is assigned to machine i
- The load of machine i is $L_i = \sum_{j \in J(i)} t_j$
- So the wall-clock time is $\max_i L_i$ (which we want to minimise)

Scheduling Jobs on Parallel Machines

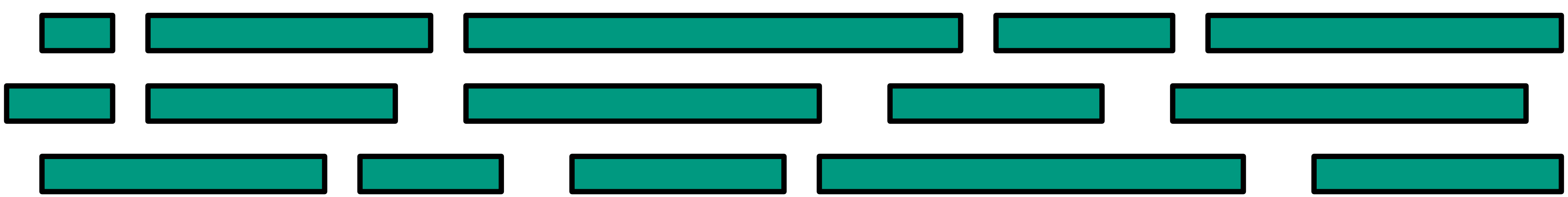
1
2
3
4
5



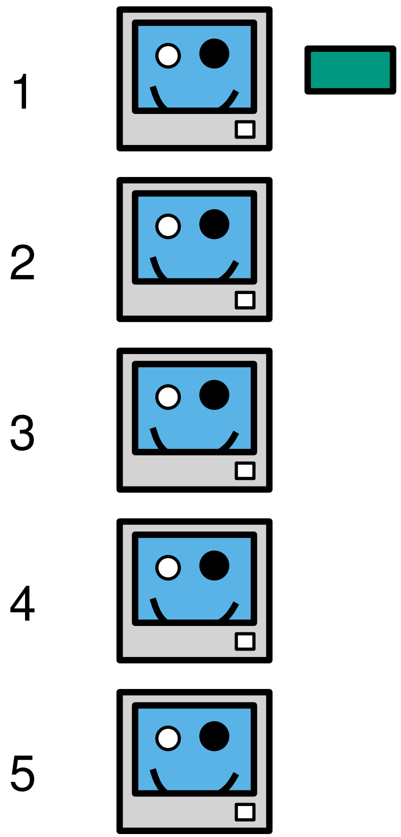
Scheduling Jobs on Parallel Machines



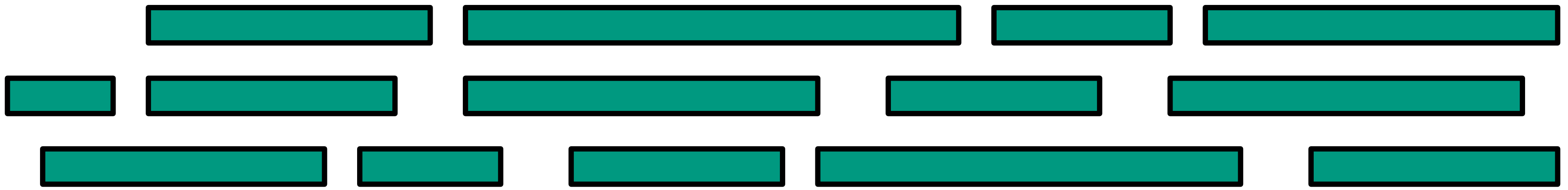
Algorithm: Put job j on the machine i with smallest (current) load



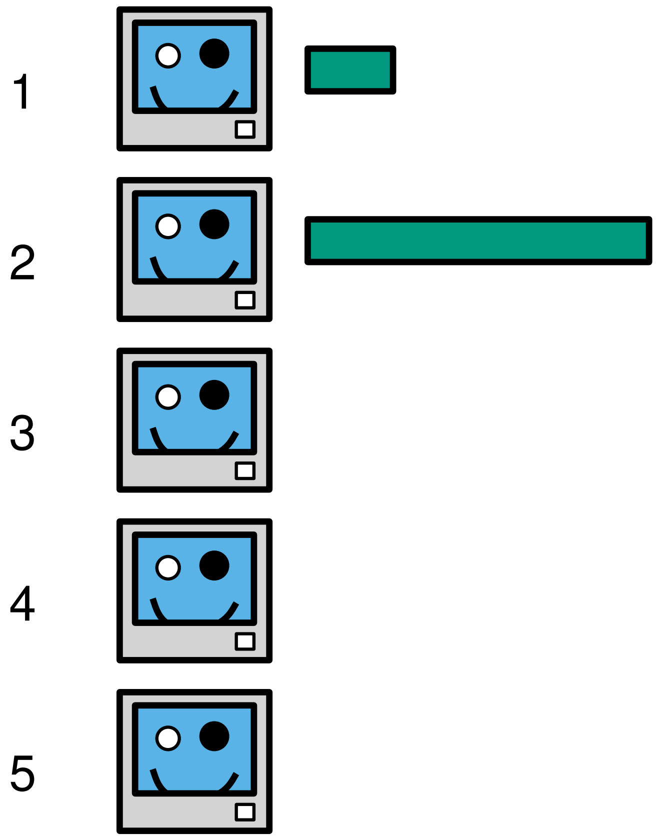
Scheduling Jobs on Parallel Machines



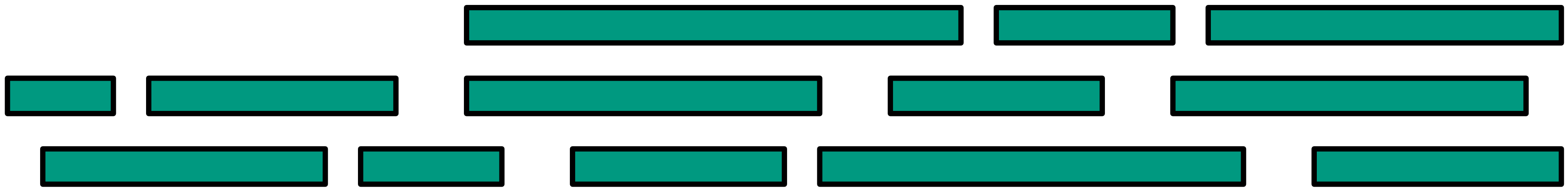
Algorithm: Put job j on the machine i with smallest (current) load



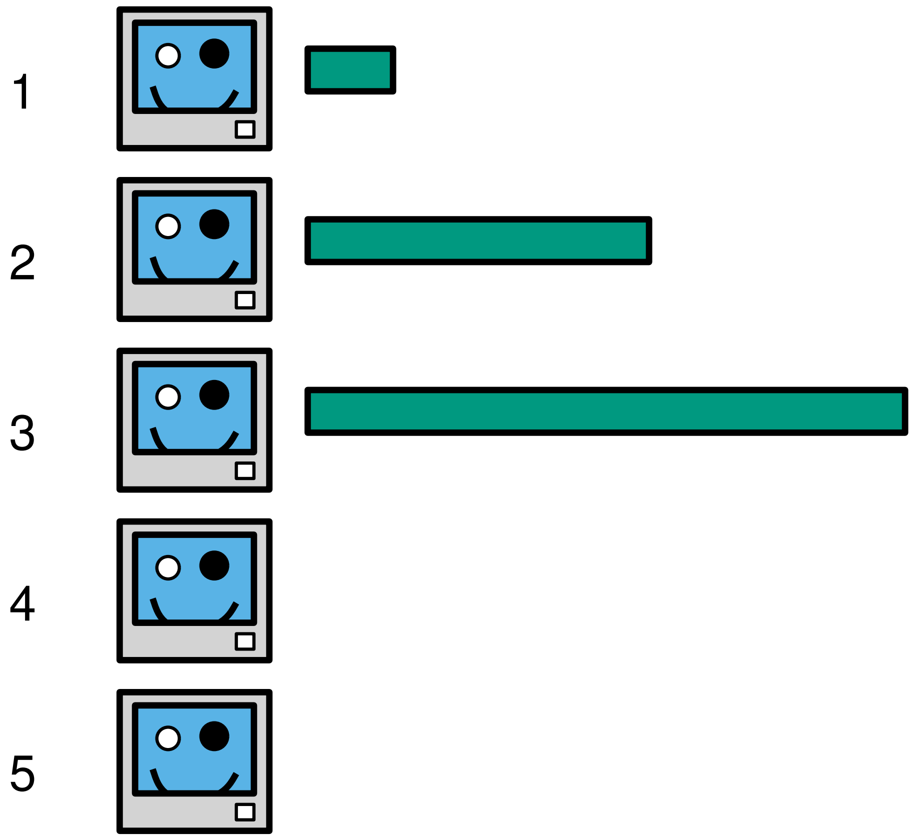
Scheduling Jobs on Parallel Machines



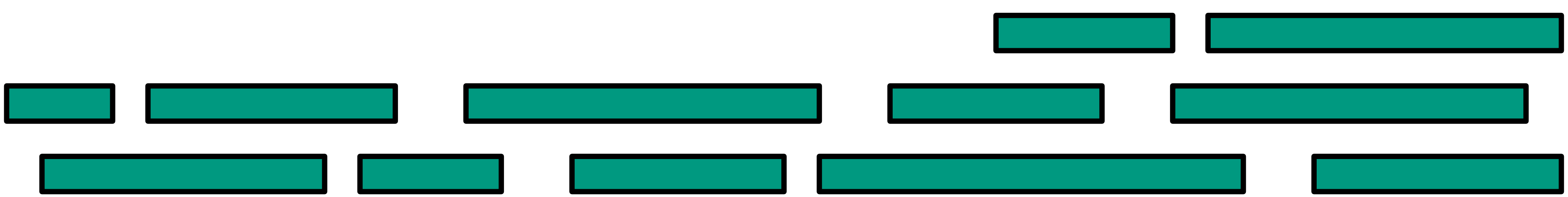
Algorithm: Put job j on the machine i with smallest (current) load



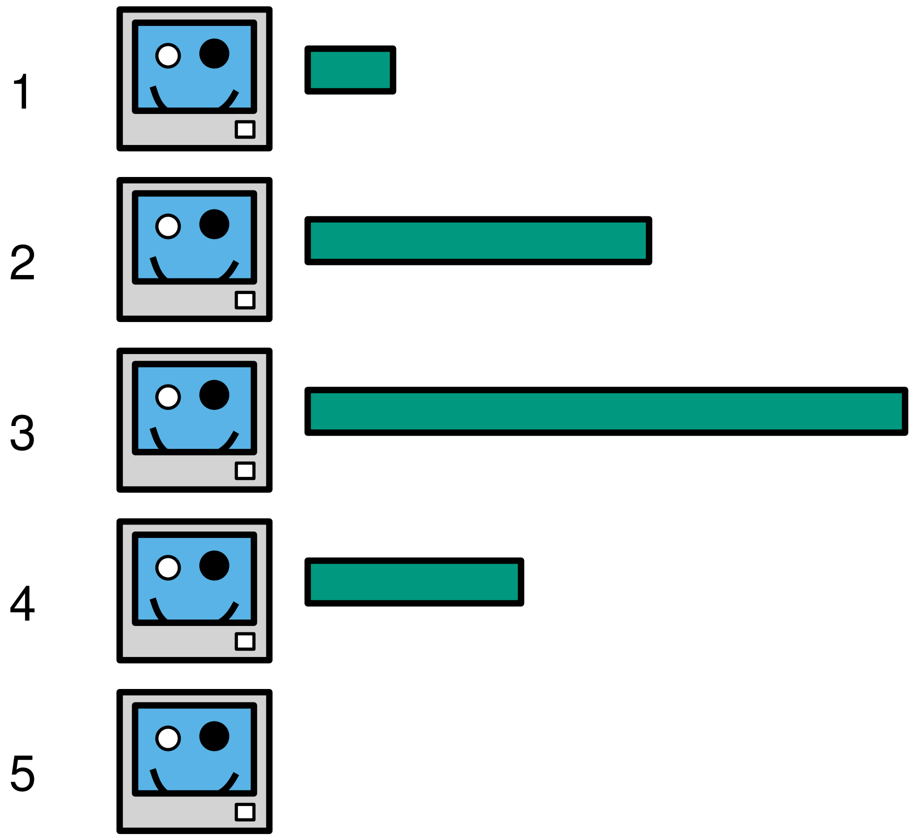
Scheduling Jobs on Parallel Machines



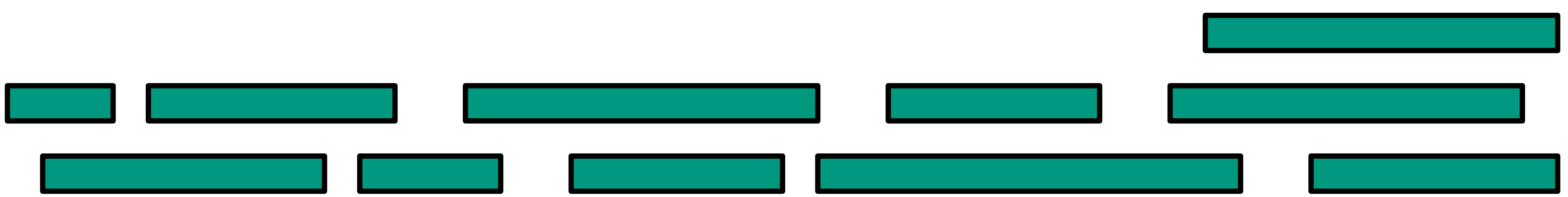
Algorithm: Put job j on the machine i with smallest (current) load



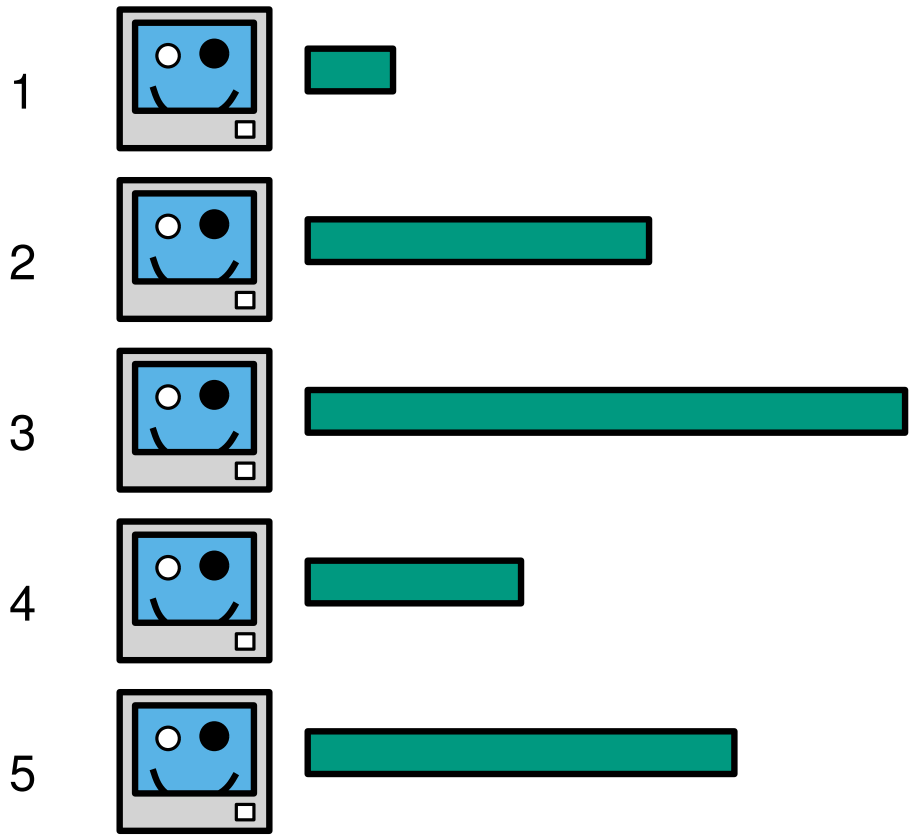
Scheduling Jobs on Parallel Machines



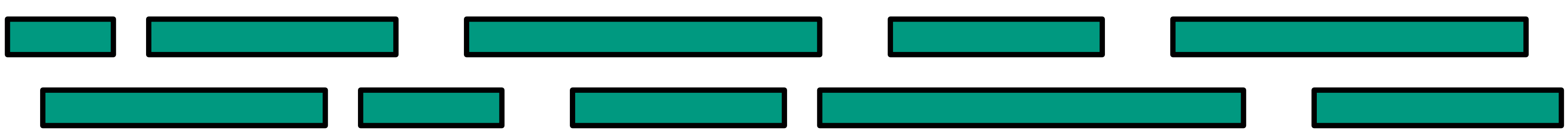
Algorithm: Put job j on the machine i with smallest (current) load



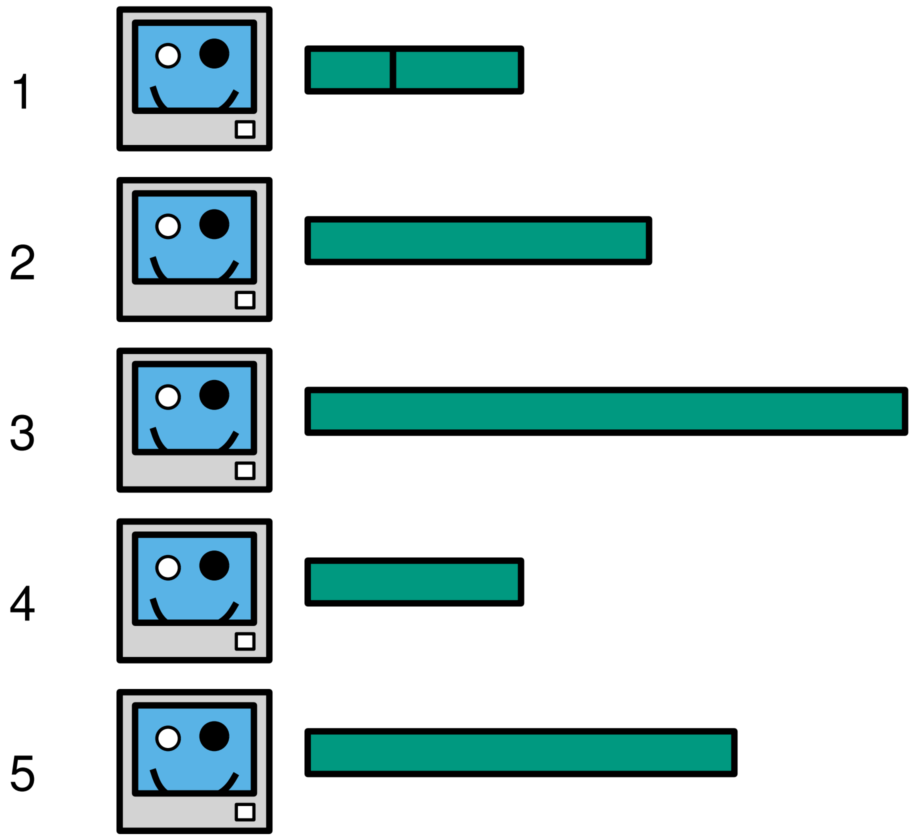
Scheduling Jobs on Parallel Machines



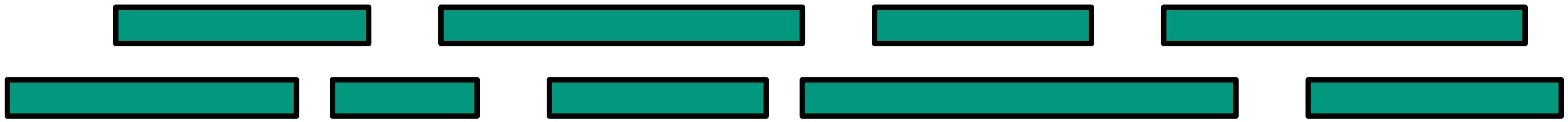
Algorithm: Put job j on the machine i with smallest (current) load



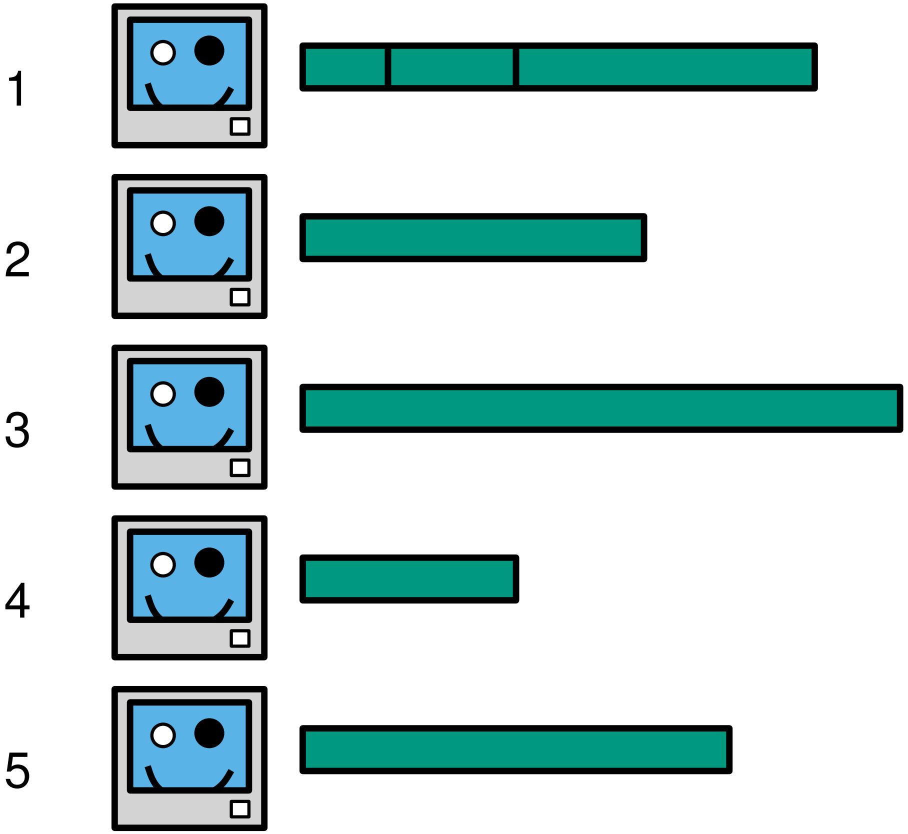
Scheduling Jobs on Parallel Machines



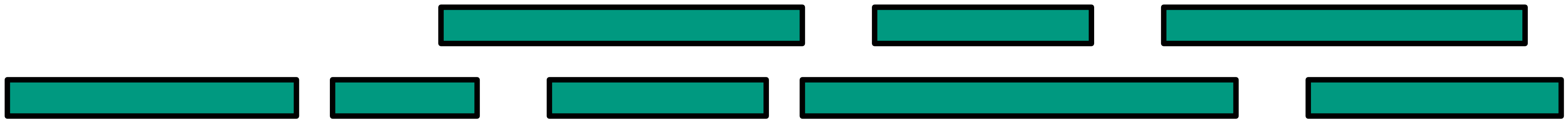
Algorithm: Put job j on the machine i with smallest (current) load



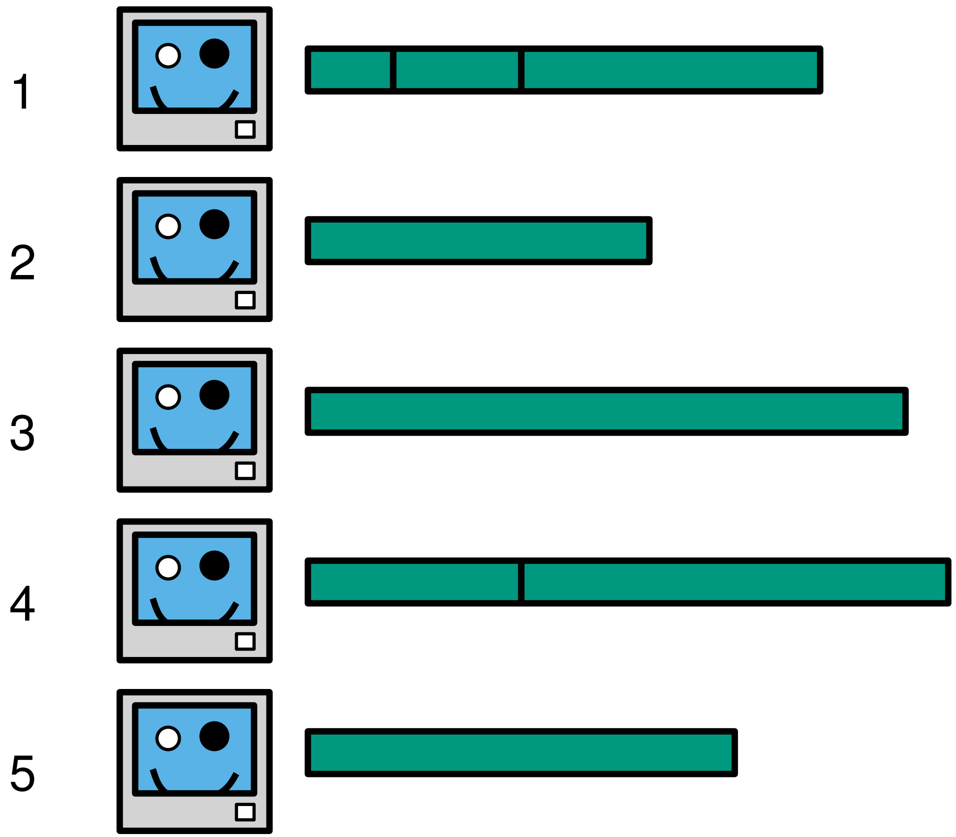
Scheduling Jobs on Parallel Machines



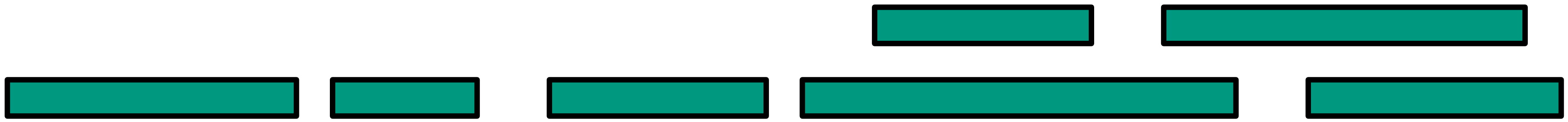
Algorithm: Put job j on the machine i with smallest (current) load



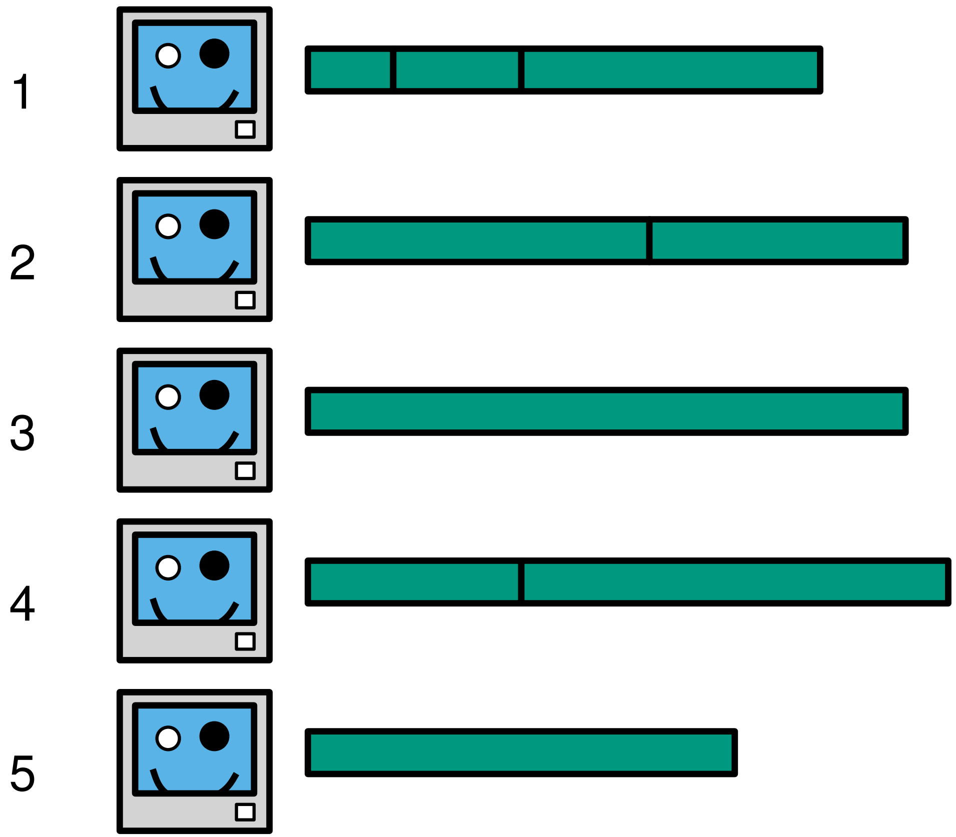
Scheduling Jobs on Parallel Machines



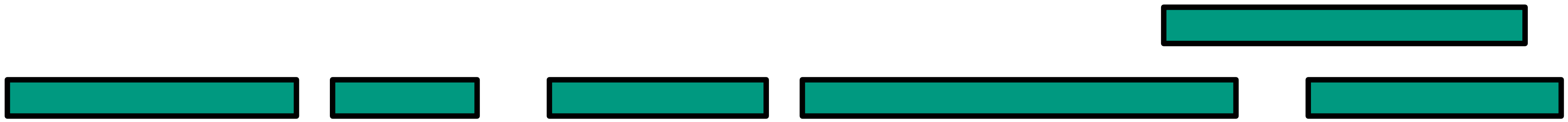
Algorithm: Put job j on the machine i with smallest (current) load



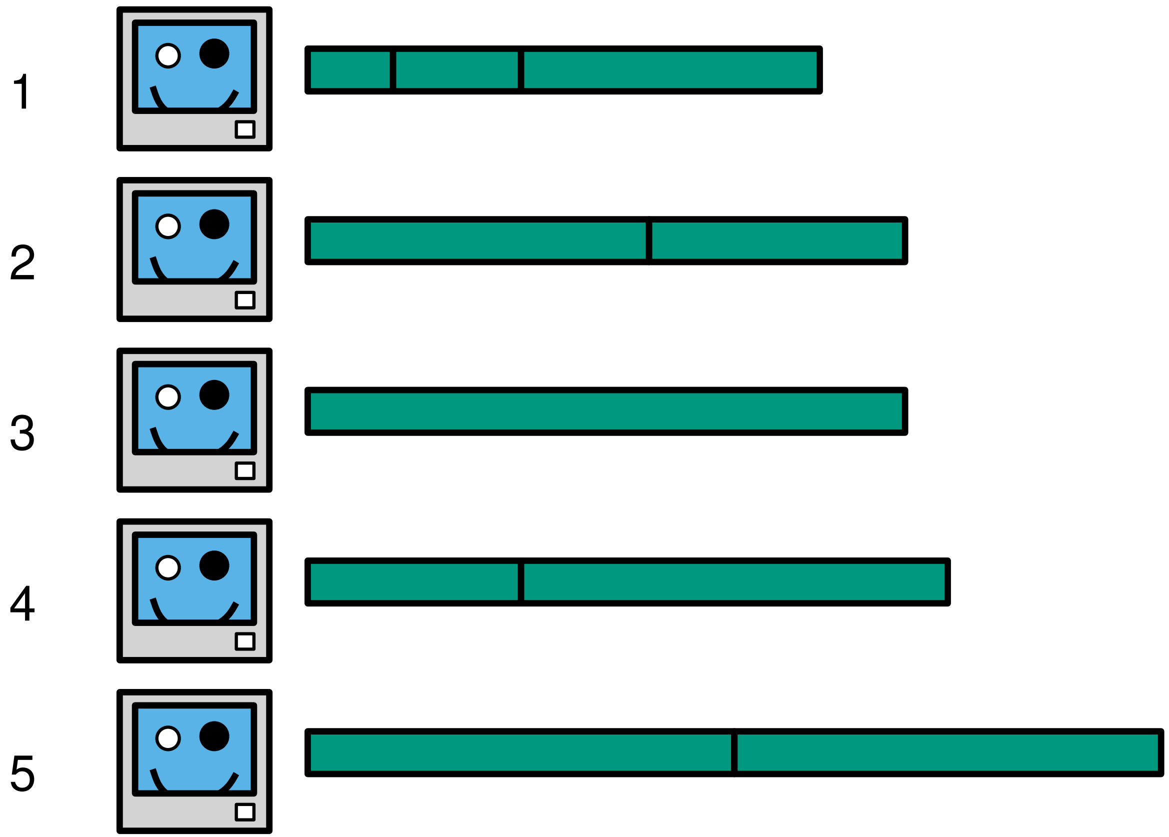
Scheduling Jobs on Parallel Machines



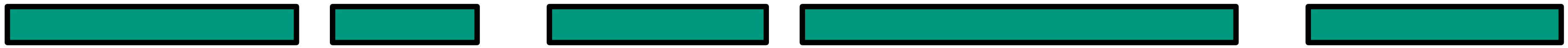
Algorithm: Put job j on the machine i with smallest (current) load



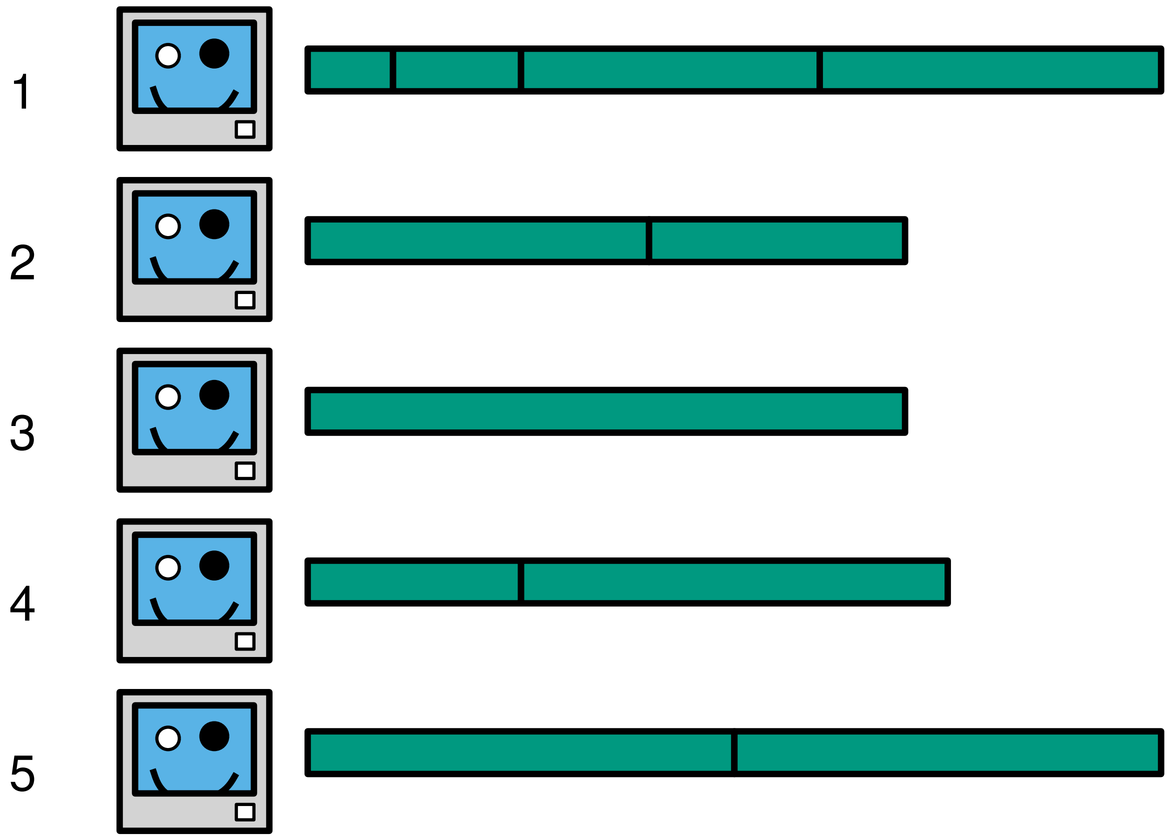
Scheduling Jobs on Parallel Machines



Algorithm: Put job j on the machine i with smallest (current) load

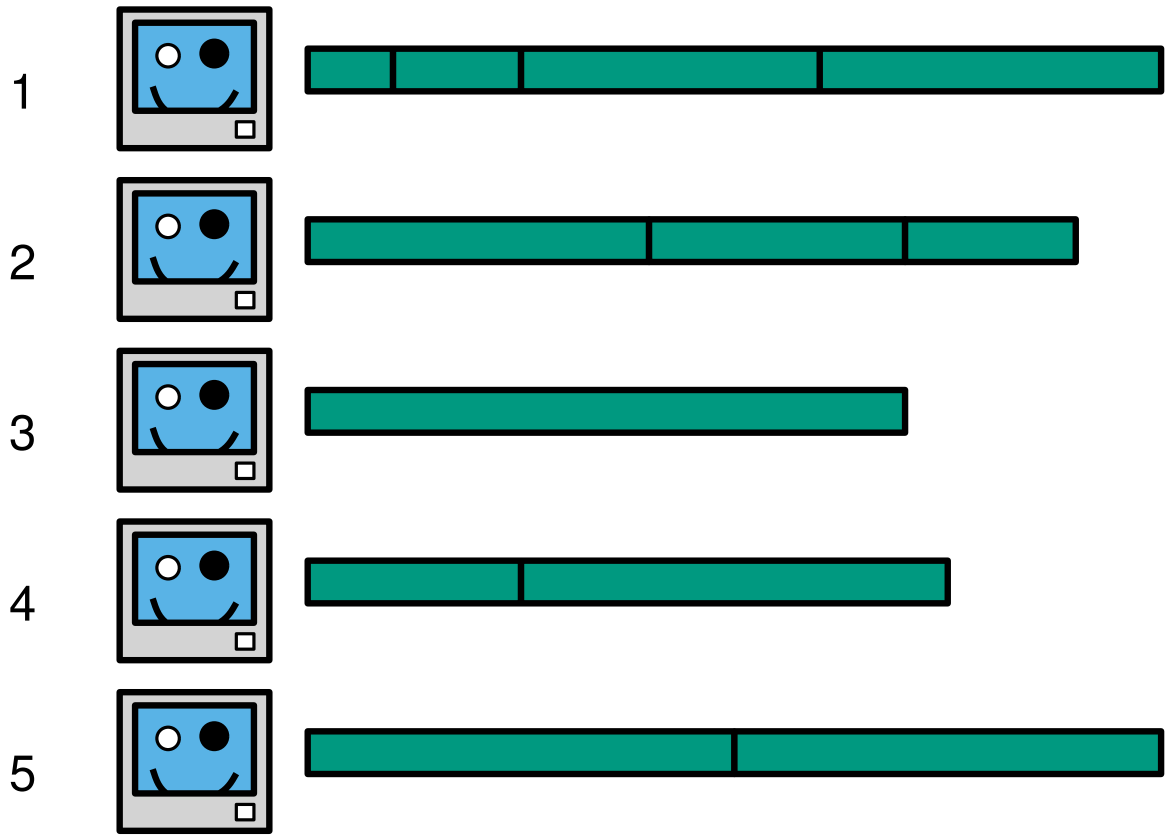


Scheduling Jobs on Parallel Machines



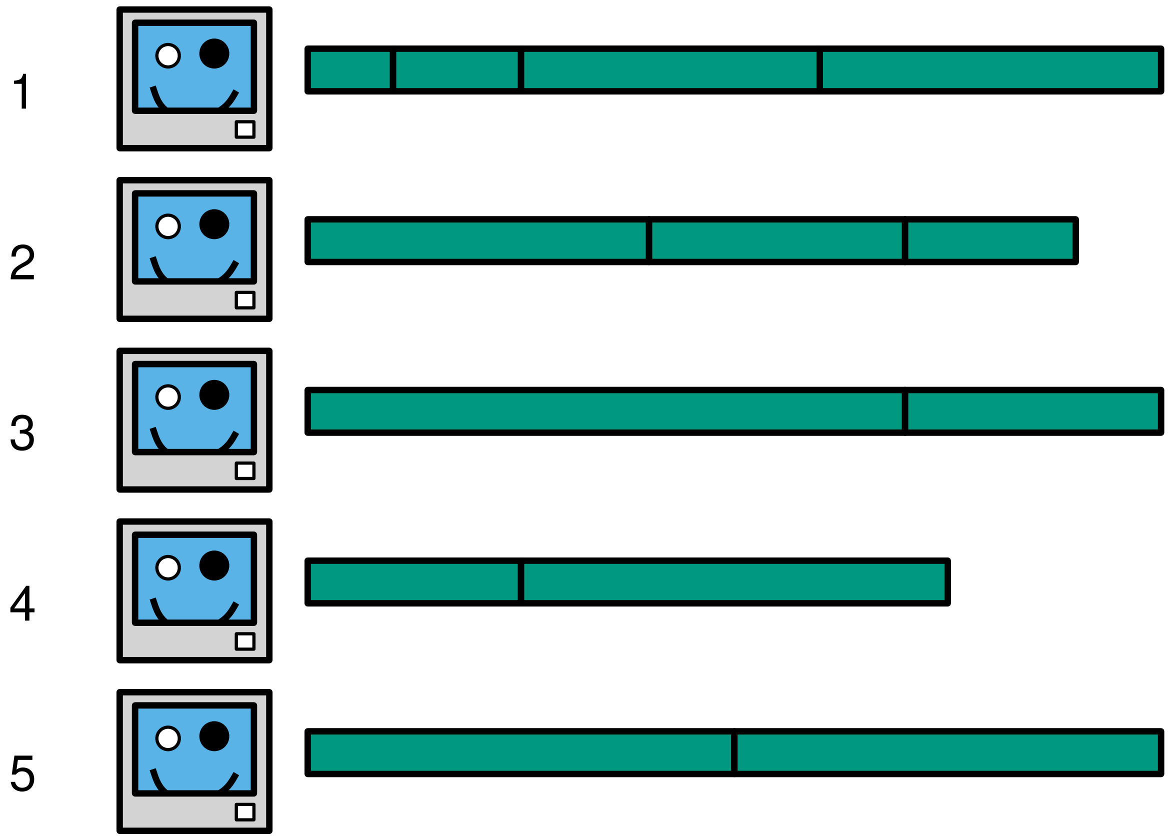
Algorithm: Put job j on the machine i with smallest (current) load

Scheduling Jobs on Parallel Machines



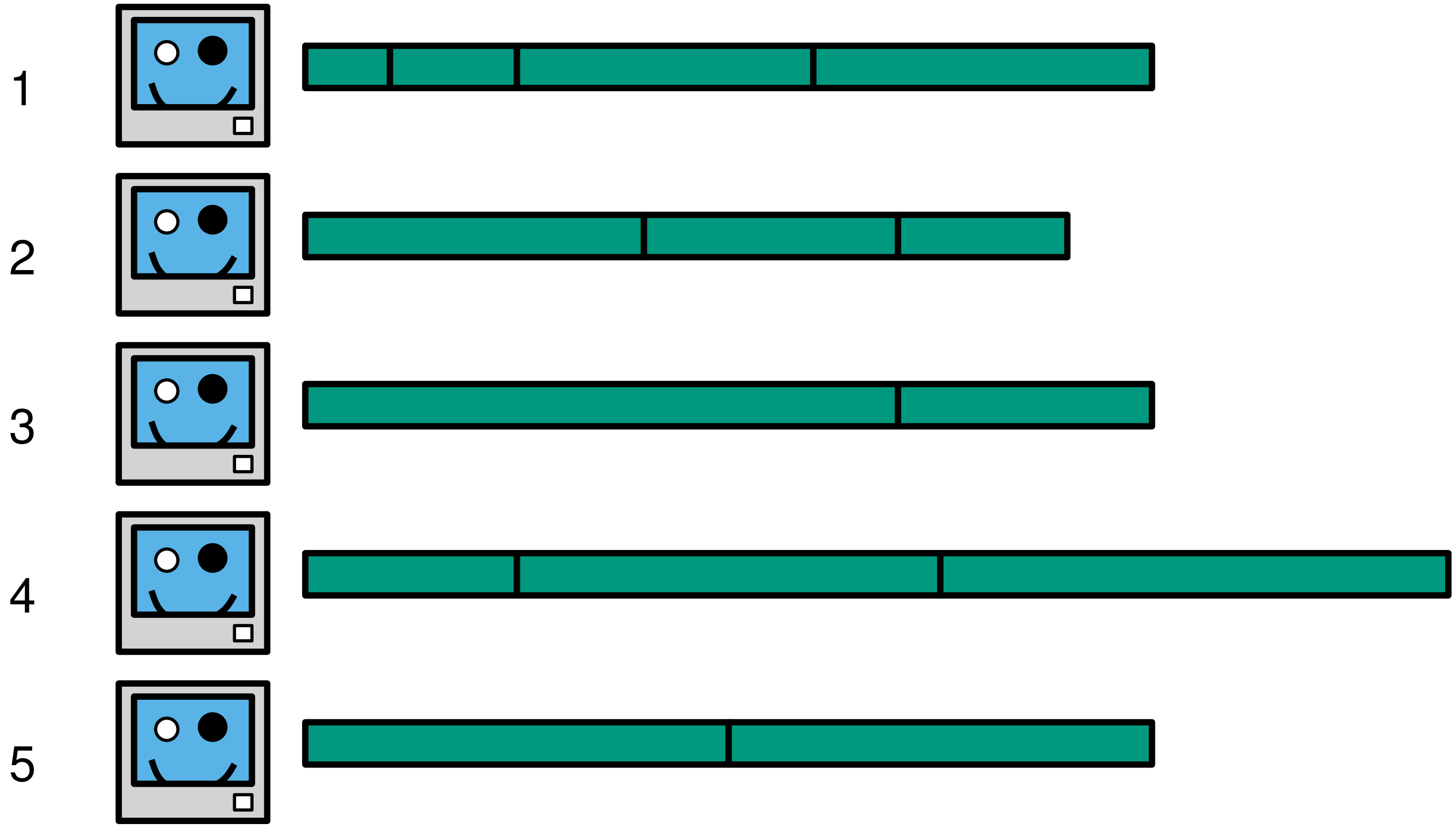
Algorithm: Put job j on the machine i with smallest (current) load

Scheduling Jobs on Parallel Machines



Algorithm: Put job j on the machine i with smallest (current) load

Scheduling Jobs on Parallel Machines



Algorithm: Put job j on the machine i with smallest (current) load

Scheduling Jobs on Parallel Machines

Algorithm: Put job j on the machine i with smallest (current) load

Scheduling Jobs on Parallel Machines

Algorithm: Put job j on the machine i with smallest (current) load

How long does it take to compute this schedule?

Scheduling Jobs on Parallel Machines

m machines

n jobs

Algorithm: Put job j on the machine i with smallest (current) load

How long does it take to compute this schedule?

Scheduling Jobs on Parallel Machines

m machines

n jobs

Algorithm: Put job j on the machine i with smallest (current) load

How long does it take to compute this schedule?

$O(nm)$ time naively, $O(n \log m)$ time using a priority queue

Scheduling Jobs on Parallel Machines

m machines

n jobs

Algorithm: Put job j on the machine i with smallest (current) load

How long does it take to compute this schedule?

$O(nm)$ time naively, $O(n \log m)$ time using a priority queue

(it's also an online solution)

University of BRISTOL

Scheduling Jobs on Parallel Machines

m machines

n jobs

Algorithm: Put job j on the machine i with smallest (current) load

How long does it take to compute this schedule?

$O(nm)$ time naively, $O(n \log m)$ time using a priority queue

(it's also an online solution)

How good is it?

The greedy approximation

Let Opt denote the time taken by the optimal scheduling of jobs

Let T_g denote the time taken by the greedy schedule

Theorem The greedy algorithm given is a 2-approximation algorithm

L_i is the *load* of machine i

Job j takes t_j time units

m machines
n jobs

The greedy approximation

L_i is the *load* of machine i

Let Opt denote the time taken by the optimal scheduling of jobs

Let T_g denote the time taken by the greedy schedule

Theorem The greedy algorithm given is a 2-approximation algorithm

- Before we prove this, we prove two useful facts,

Job j takes t_j time units

m machines

n jobs

The greedy approximation

L_i is the *load* of machine i

Let Opt denote the time taken by the optimal scheduling of jobs

Let T_g denote the time taken by the greedy schedule

Theorem The greedy algorithm given is a 2-approximation algorithm

- Before we prove this, we prove two useful facts,

Fact $\mathrm{Opt} \geqslant \max_j t_j$

Job j takes t_j time units

m machines

n jobs

The greedy approximation

L_i is the *load* of machine i

Let Opt denote the time taken by the optimal scheduling of jobs

Let T_g denote the time taken by the greedy schedule

Theorem The greedy algorithm given is a 2-approximation algorithm

- Before we prove this, we prove two useful facts,

Fact $\mathrm{Opt} \geqslant \max_j t_j$

  - Some machine must process the largest job

Job j takes t_j time units

m machines
n jobs

The greedy approximation

L_i is the *load* of machine i

Let Opt denote the time taken by the optimal scheduling of jobs

Let T_g denote the time taken by the greedy schedule

Theorem The greedy algorithm given is a 2-approximation algorithm

- Before we prove this, we prove two useful facts,

Fact $\mathrm{Opt} \geqslant \max_j t_j$

Job j takes t_j time units

- Some machine must process the largest job

Fact $\mathrm{Opt} \geqslant \frac{\sum_j t_j}{m}$

m machines

n jobs

The greedy approximation

L_i is the *load* of machine i

Let Opt denote the time taken by the optimal scheduling of jobs

Let T_g denote the time taken by the greedy schedule

Theorem The greedy algorithm given is a 2-approximation algorithm

- Before we prove this, we prove two useful facts,

Fact $\mathrm{Opt} \geqslant \max_j t_j$

Job j takes t_j time units

- Some machine must process the largest job

Fact $\mathrm{Opt} \geqslant \dfrac{\sum_j t_j}{m}$

m machines

n jobs

- There is a total of $\sum_j t_j$ time units of work to be done

The greedy approximation

L_i is the *load* of machine i

Let Opt denote the time taken by the optimal scheduling of jobs

Let T_g denote the time taken by the greedy schedule

Theorem The greedy algorithm given is a 2-approximation algorithm

- Before we prove this, we prove two useful facts,

Fact $\mathrm{Opt} \geqslant \max_j t_j$

Job j takes t_j time units

- Some machine must process the largest job

Fact $\mathrm{Opt} \geqslant \frac{\sum_j t_j}{m}$

m machines
n jobs

- There is a total of $\sum_j t_j$ time units of work to be done
- Some machine i must have load L_i at least $\frac{\sum_j t_j}{m}$

The greedy approximation

L_i is the *load* of machine i

Let Opt denote the time taken by the optimal scheduling of jobs

Let T_g denote the time taken by the greedy schedule

Theorem The greedy algorithm given is a 2-approximation algorithm

- Before we prove this, we prove two useful facts,

Fact $\mathrm{Opt} \geqslant \max_j t_j$

Job j takes t_j time units

- ○ Some machine must process the largest job

Fact $\mathrm{Opt} \geqslant \dfrac{\sum_j t_j}{m}$

m machines
n jobs

- ○ There is a total of $\sum_j t_j$ time units of work to be done
- ○ Some machine i must have load L_i at least $\dfrac{\sum_j t_j}{m}$

(the m machines can't **all** *have below average load)*

The greedy approximation

L_i is the *load* of machine i

Let Opt denote the time taken by the optimal scheduling of jobs

Let T_g denote the time taken by the greedy schedule

Theorem The greedy algorithm given is a 2-approximation algorithm

Job j takes t_j time units

m machines

n jobs

The greedy approximation

L_i is the *load* of machine i

Let Opt denote the time taken by the optimal scheduling of jobs

Let T_g denote the time taken by the greedy schedule

Theorem The greedy algorithm given is a 2-approximation algorithm

Proof Consider the machine i with largest load $T_g = L_i$

Job j takes t_j time units

m machines

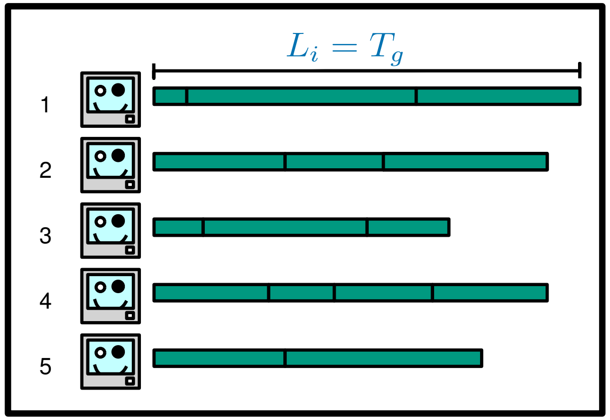
n jobs

The greedy approximation

L_i is the *load* of machine i

Let Opt denote the time taken by the optimal scheduling of jobs

Let T_g denote the time taken by the greedy schedule



Theorem The greedy algorithm given is a 2-approximation algorithm

Proof Consider the machine i with largest load $T_g = L_i$

Job j takes t_j time units

m machines

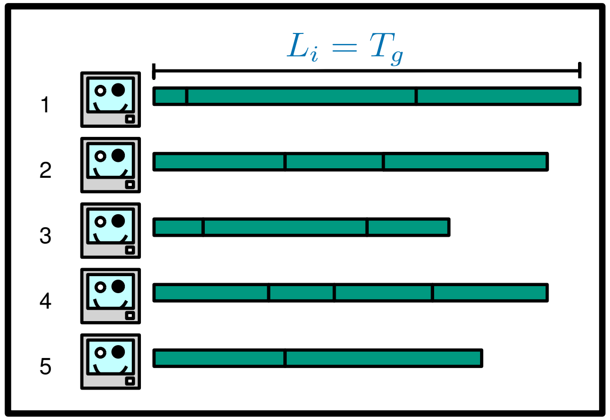
n jobs

The greedy approximation

L_i is the *load* of machine i

Let Opt denote the time taken by the optimal scheduling of jobs

Let T_g denote the time taken by the greedy schedule



Theorem The greedy algorithm given is a 2-approximation algorithm

Proof Consider the machine i with largest load $T_g = L_i$

- Let j denote the last job machine i completes

Job j takes t_j time units

m machines

n jobs

The greedy approximation

Let Opt denote the time taken by the optimal scheduling of jobs

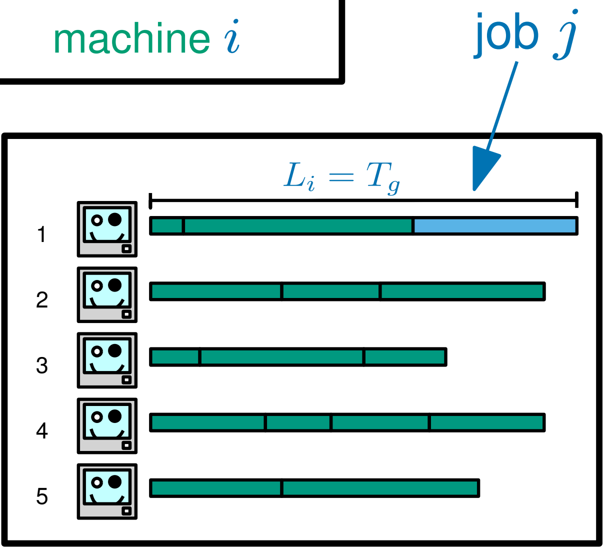
Let T_g denote the time taken by the greedy schedule

Theorem The greedy algorithm given is a 2-approximation algorithm

Proof Consider the machine i with largest load $T_g = L_i$

- Let j denote the last job machine i completes

L_i is the *load* of machine i



Job j takes t_j time units

m machines
n jobs

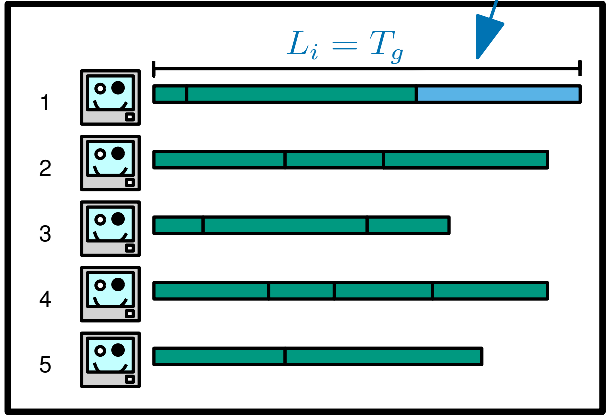
The greedy approximation

L_i is the *load* of machine i

job j

Let Opt denote the time taken by the optimal scheduling of jobs

Let T_g denote the time taken by the greedy schedule



Theorem The greedy algorithm given is a 2-approximation algorithm

Proof Consider the machine i with largest load $T_g = L_i$

- Let j denote the last job machine i completes
- When job j was assigned, machine i had the smallest load, $L_i - t_j$

Job j takes t_j time units

m machines

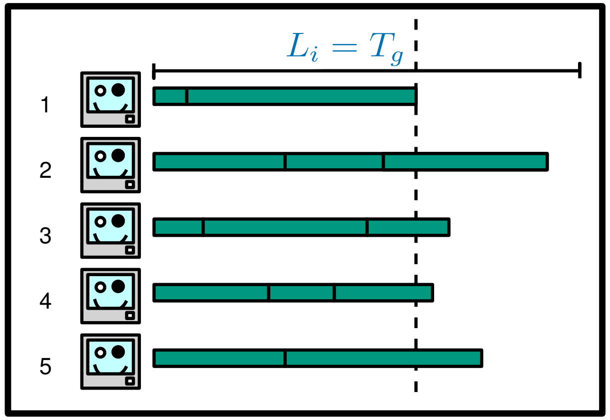
n jobs

The greedy approximation

L_i is the *load* of machine i

Let Opt denote the time taken by the optimal scheduling of jobs

Let T_g denote the time taken by the greedy schedule



Theorem The greedy algorithm given is a 2-approximation algorithm

Proof Consider the machine i with largest load $T_g = L_i$

- Let j denote the last job machine i completes
- When job j was assigned, machine i had the smallest load, $L_i - t_j$

Job j takes t_j time units

m machines

n jobs

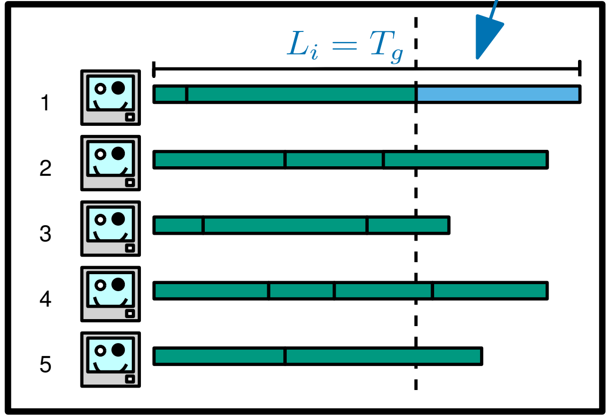
The greedy approximation

L_i is the *load* of machine i

job j

Let Opt denote the time taken by the optimal scheduling of jobs

Let T_g denote the time taken by the greedy schedule



Theorem The greedy algorithm given is a 2-approximation algorithm

Proof Consider the machine i with largest load $T_g = L_i$

- Let j denote the last job machine i completes
- When job j was assigned, machine i had the smallest load, $L_i - t_j$

Job j takes t_j time units

m machines

n jobs

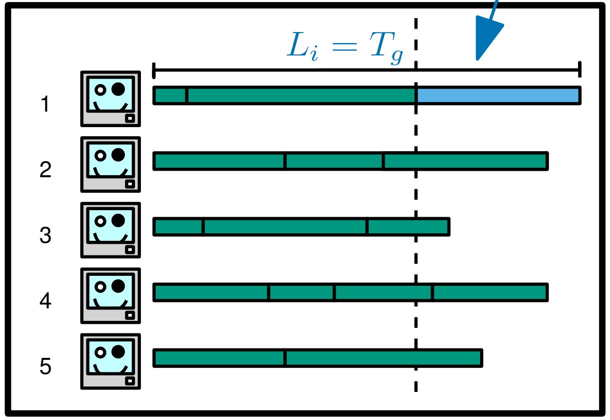
The greedy approximation

L_i is the *load* of machine i

job j

Let Opt denote the time taken by the optimal scheduling of jobs

Let T_g denote the time taken by the greedy schedule



Theorem The greedy algorithm given is a 2-approximation algorithm

Proof Consider the machine i with largest load $T_g = L_i$

- Let j denote the last job machine i completes
- When job j was assigned, machine i had the smallest load, $L_i - t_j$

So...

Job j takes t_j time units

m machines

n jobs

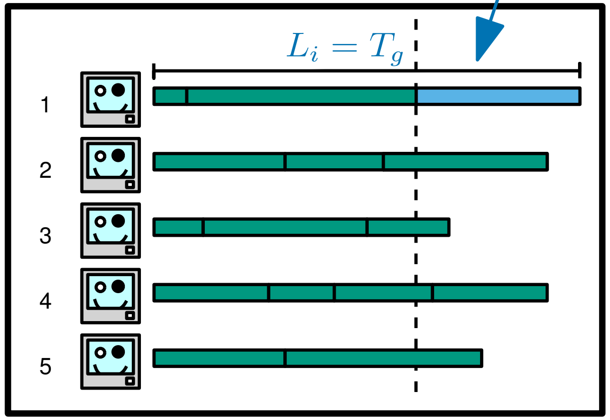
The greedy approximation

L_i is the *load* of machine i

job j

Let Opt denote the time taken by the optimal scheduling of jobs

Let T_g denote the time taken by the greedy schedule



Theorem The greedy algorithm given is a 2-approximation algorithm

Proof Consider the machine i with largest load $T_g = L_i$

- Let j denote the last job machine i completes
- When job j was assigned, machine i had the smallest load, $L_i - t_j$

So...

$$L_i - t_j \leqslant L_k \text{ for all } 1 \leqslant k \leqslant m,$$

Job j takes t_j time units

m machines

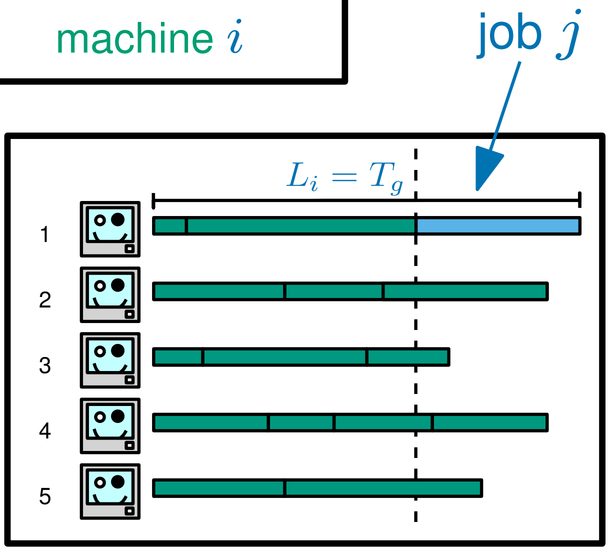
n jobs

The greedy approximation

L_i is the *load* of machine i

Let Opt denote the time taken by the optimal scheduling of jobs

Let T_g denote the time taken by the greedy schedule



Theorem The greedy algorithm given is a 2-approximation algorithm

Proof Consider the machine i with largest load $T_g = L_i$

- Let j denote the last job machine i completes
- When job j was assigned, machine i had the smallest load, $L_i - t_j$

So...

$L_i - t_j \leqslant L_k$ for all $1 \leqslant k \leqslant m$,

Job j takes t_j time units

- If we then sum over all k,

m machines

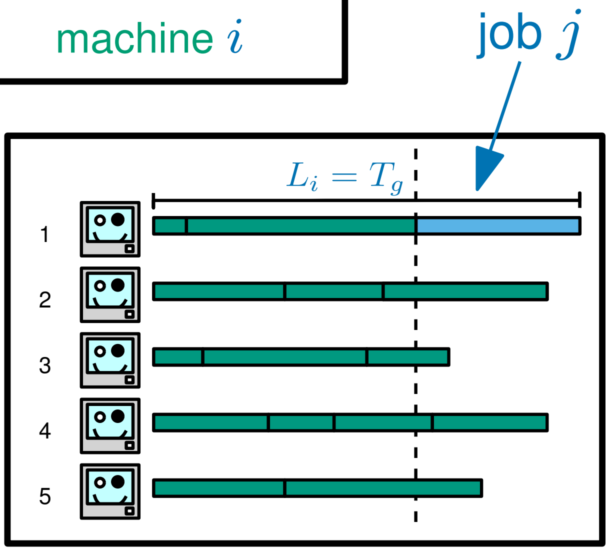
n jobs

The greedy approximation

L_i is the *load* of machine i

Let Opt denote the time taken by the optimal scheduling of jobs

Let T_g denote the time taken by the greedy schedule



Theorem The greedy algorithm given is a 2-approximation algorithm

Proof Consider the machine i with largest load $T_g = L_i$

- Let j denote the last job machine i completes
- When job j was assigned, machine i had the smallest load, $L_i - t_j$

So...

$L_i - t_j \leqslant L_k$ for all $1 \leqslant k \leqslant m$,

Job j takes t_j time units

- If we then sum over all k,

$$m(L_i - t_j) \leqslant \sum_{k=1}^{m} L_k$$

m machines

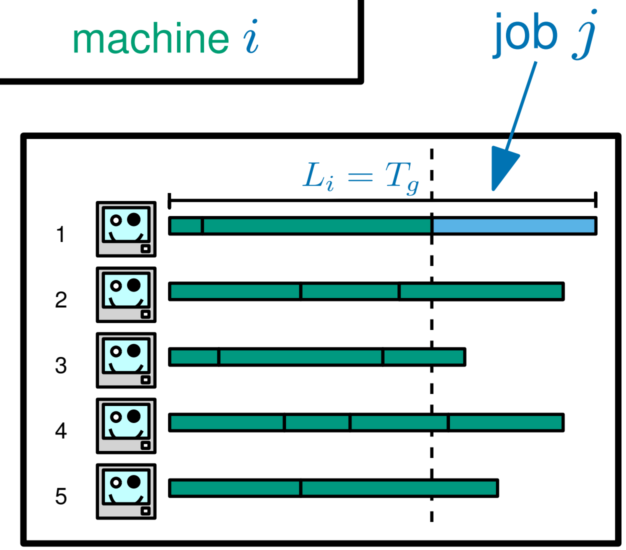
n jobs

The greedy approximation

L_i is the *load* of machine i

Let Opt denote the time taken by the optimal scheduling of jobs

Let T_g denote the time taken by the greedy schedule



Theorem The greedy algorithm given is a 2-approximation algorithm

Proof Consider the machine i with largest load $T_g = L_i$

- Let j denote the last job machine i completes
- When job j was assigned, machine i had the smallest load, $L_i - t_j$

So. . .

$L_i - t_j \leqslant L_k$ for all $1 \leqslant k \leqslant m$,

Job j takes t_j time units

- If we then sum over all k,

$$m(L_i - t_j) \leqslant \sum_{k=1}^{m} L_k$$

m machines

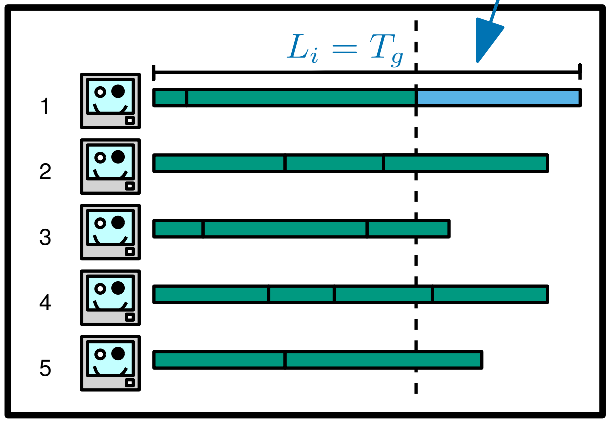
n jobs

so $(L_i - t_j) \leqslant \frac{\sum_{k=1}^{m} L_k}{m} \leqslant \mathrm{Opt}$ (by the second fact)

The greedy approximation

L_i is the *load* of machine i

job j



Let Opt denote the time taken by the optimal scheduling of jobs

Let T_g denote the time taken by the greedy schedule

Theorem The greedy algorithm given is a 2-approximation algorithm

Proof Consider the machine i with largest load $T_g = L_i$

- Let j denote the last job machine i completes
- When job j was assigned, machine i had the smallest load, $L_i - t_j$

So...

$$L_i - t_j \leqslant L_k \text{ for all } 1 \leqslant k \leqslant m,$$

Job j takes t_j time units

- If we then sum over all k,

$$m(L_i - t_j) \leqslant \sum_{k=1}^{m} L_k$$

m machines

n jobs

so $(L_i - t_j) \leqslant \dfrac{\sum_{k=1}^{m} L_k}{m} \leqslant \mathrm{Opt}$ (by the second fact)

Fact $\mathrm{Opt} \geqslant \dfrac{\sum_j t_j}{m}$

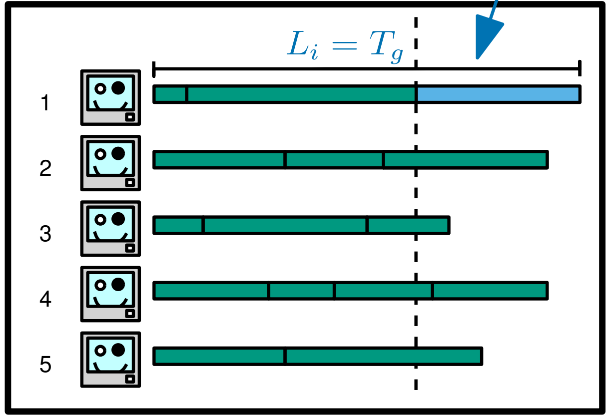
The greedy approximation

L_i is the *load* of machine i

job j

Let Opt denote the time taken by the optimal scheduling of jobs

Let T_g denote the time taken by the greedy schedule



Theorem The greedy algorithm given is a 2-approximation algorithm

Proof Consider the machine i with largest load $T_g = L_i$

- Let j denote the last job machine i completes
- When job j was assigned, machine i had the smallest load, $L_i - t_j$

So...

$$L_i - t_j \leqslant L_k \text{ for all } 1 \leqslant k \leqslant m,$$

Job j takes t_j time units

- If we then sum over all k,

$$m(L_i - t_j) \leqslant \sum_{k=1}^{m} L_k$$

m machines

n jobs

so $(L_i - t_j) \leqslant \frac{\sum_{k=1}^{m} L_k}{m} \leqslant \mathrm{Opt}$ (by the second fact)

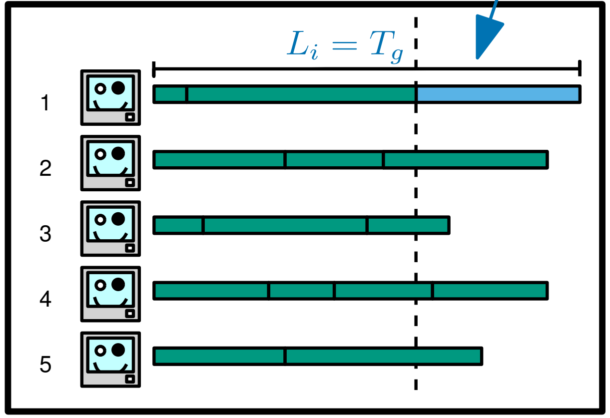
The greedy approximation

L_i is the *load* of machine i

Let Opt denote the time taken by the optimal scheduling of jobs

Let T_g denote the time taken by the greedy schedule

job j



Theorem The greedy algorithm given is a 2-approximation algorithm

Proof Consider the machine i with largest load $T_g = L_i$

- Let j denote the last job machine i completes
- When job j was assigned, machine i had the smallest load, $L_i - t_j$

So...

$$L_i - t_j \leqslant L_k \text{ for all } 1 \leqslant k \leqslant m,$$

Job j takes t_j time units

- If we then sum over all k,

$$m(L_i - t_j) \leqslant \sum_{k=1}^{m} L_k$$

m machines
n jobs

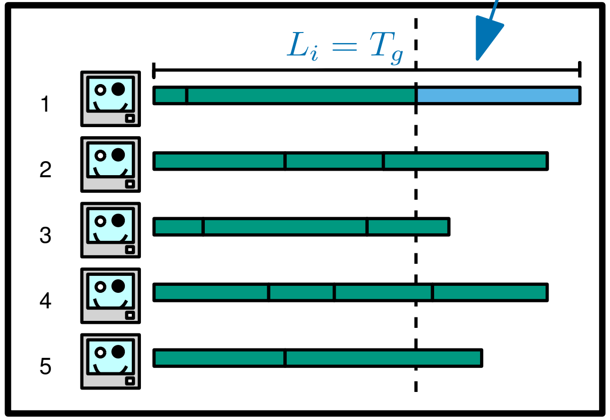
so $(L_i - t_j) \leqslant \frac{\sum_{k=1}^{m} L_k}{m} \leqslant \mathrm{Opt}$ (by the second fact)

also $t_j \leqslant \mathrm{Opt}$ (by the first fact)

The greedy approximation

L_i is the *load* of machine i

job j



Let Opt denote the time taken by the optimal scheduling of jobs

Let T_g denote the time taken by the greedy schedule

Theorem The greedy algorithm given is a 2-approximation algorithm

Proof Consider the machine i with largest load $T_g = L_i$

- Let j denote the last job machine i completes
- When job j was assigned, machine i had the smallest load, $L_i - t_j$

So...

$$L_i - t_j \leqslant L_k \text{ for all } 1 \leqslant k \leqslant m,$$

Job j takes t_j time units

- If we then sum over all k,

$$m(L_i - t_j) \leqslant \sum_{k=1}^{m} L_k$$

m machines

n jobs

so $(L_i - t_j) \leqslant \frac{\sum_{k=1}^{m} L_k}{m} \leqslant \mathrm{Opt}$ (by the second fact)

also $t_j \leqslant \mathrm{Opt}$ (by the first fact)

Fact $\mathrm{Opt} \geqslant \max_j t_j$

The greedy approximation

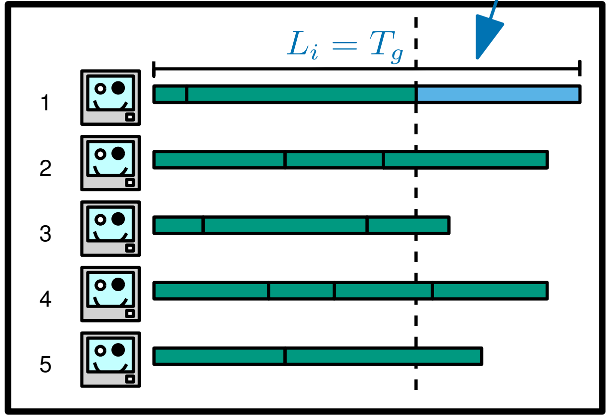
University of BRISTOL

L_i is the *load* of machine i

Let Opt denote the time taken by the optimal scheduling of jobs

Let T_g denote the time taken by the greedy schedule

job j



Theorem The greedy algorithm given is a 2-approximation algorithm

Proof Consider the machine i with largest load $T_g = L_i$

- Let j denote the last job machine i completes
- When job j was assigned, machine i had the smallest load, $L_i - t_j$

So...

$$L_i - t_j \leqslant L_k \text{ for all } 1 \leqslant k \leqslant m,$$

Job j takes t_j time units

- If we then sum over all k,

$$m(L_i - t_j) \leqslant \sum_{k=1}^{m} L_k$$

m machines

n jobs

so $(L_i - t_j) \leqslant \frac{\sum_{k=1}^{m} L_k}{m} \leqslant \mathrm{Opt}$ (by the second fact)

also $t_j \leqslant \mathrm{Opt}$ (by the first fact)

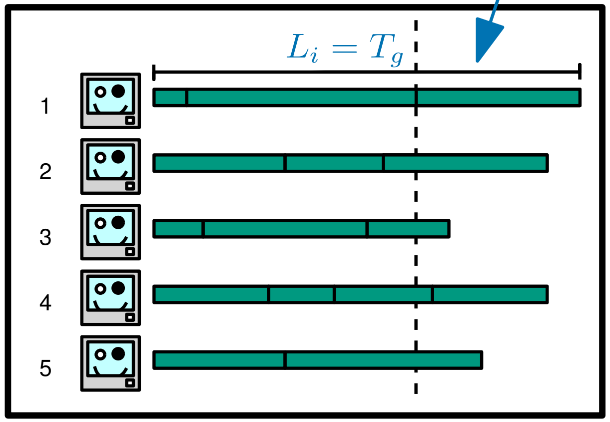
The greedy approximation

L_i is the *load* of machine i

Let Opt denote the time taken by the optimal scheduling of jobs

Let T_g denote the time taken by the greedy schedule

job j



Theorem The greedy algorithm given is a 2-approximation algorithm

Proof Consider the machine i with largest load $T_g = L_i$

- Let j denote the last job machine i completes
- When job j was assigned, machine i had the smallest load, $L_i - t_j$

So...

$$L_i - t_j \leqslant L_k \text{ for all } 1 \leqslant k \leqslant m,$$

Job j takes t_j time units

- If we then sum over all k,

$$m(L_i - t_j) \leqslant \sum_{k=1}^{m} L_k$$

m machines

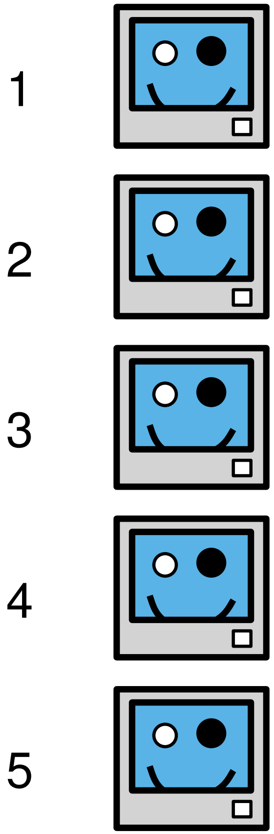
n jobs

so $(L_i - t_j) \leqslant \frac{\sum_{k=1}^{m} L_k}{m} \leqslant \mathrm{Opt}$ (by the second fact)

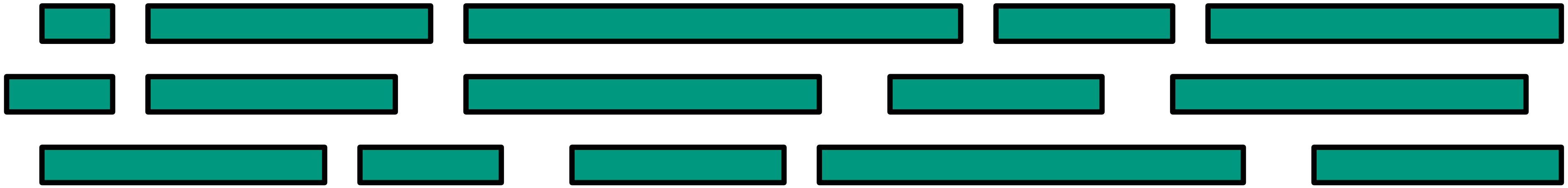
also $t_j \leqslant \mathrm{Opt}$ (by the first fact)

Therefore, $T_g = L_i = (L_i - t_j) + t_j \leqslant \mathrm{Opt} + \mathrm{Opt} = 2\mathrm{Opt}$

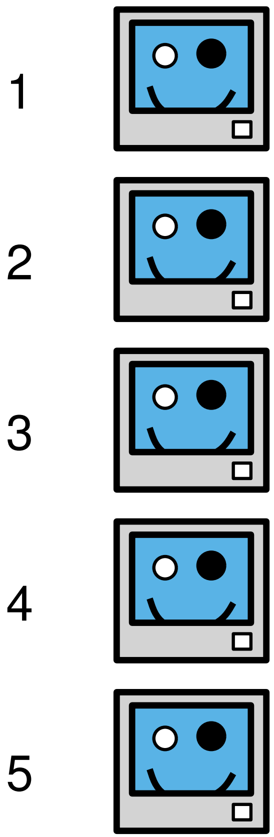
Longest Processing Time (LPT)

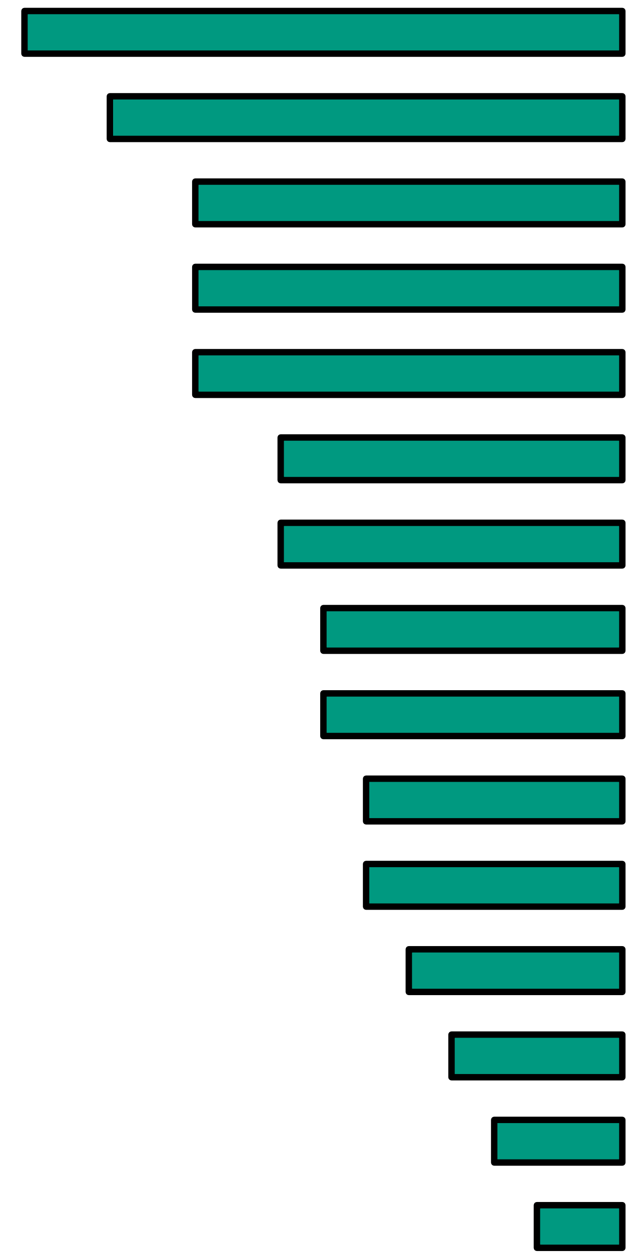


Step 1: Sort the jobs into non-increasing order (job 1 is now largest)



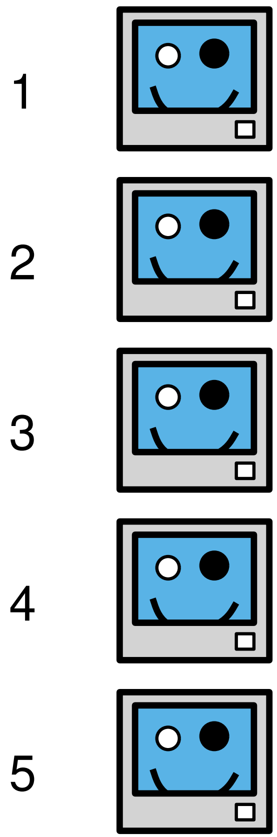
Longest Processing Time (LPT)

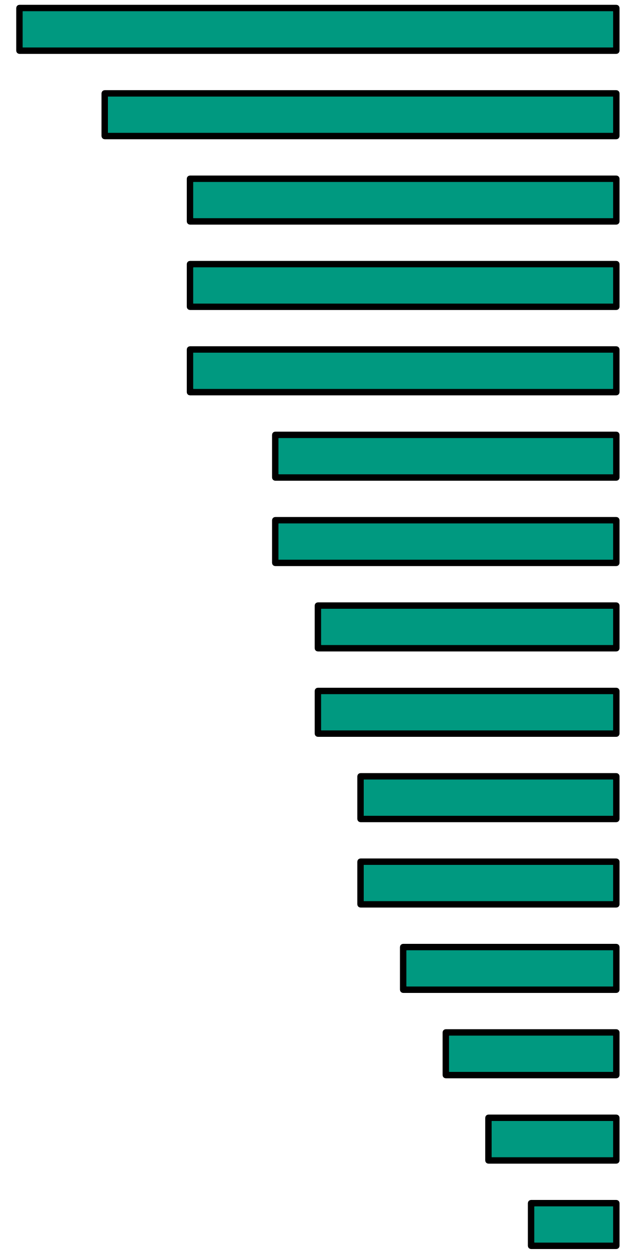




Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Longest Processing Time (LPT)





Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

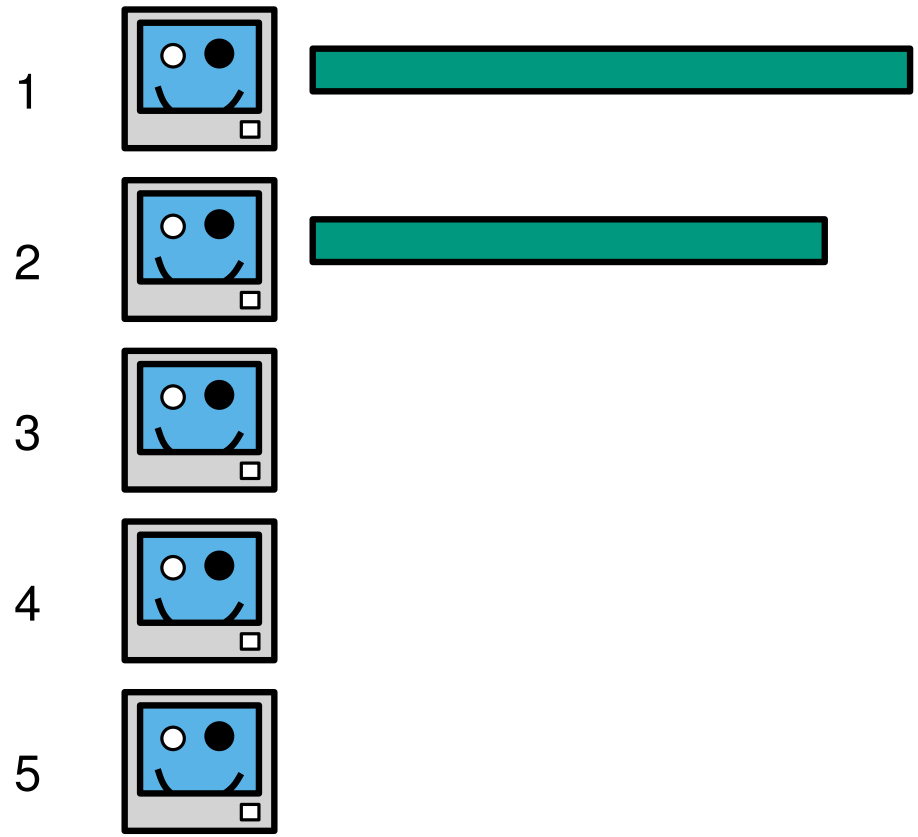
Step 2: Put job j on the machine i with smallest (current) load

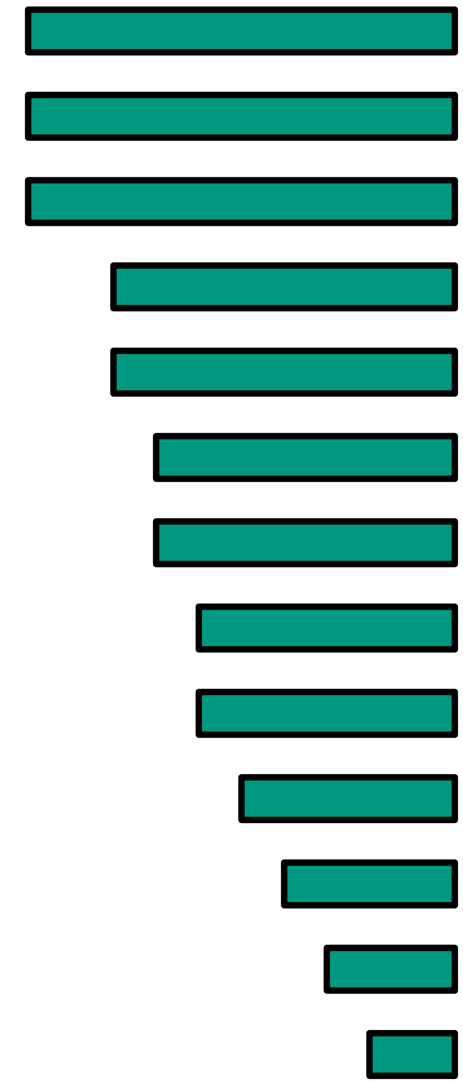
Longest Processing Time (LPT)

Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job j on the machine i with smallest (current) load

Longest Processing Time (LPT)





Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job j on the machine i with smallest (current) load

Longest Processing Time (LPT)

Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job j on the machine i with smallest (current) load

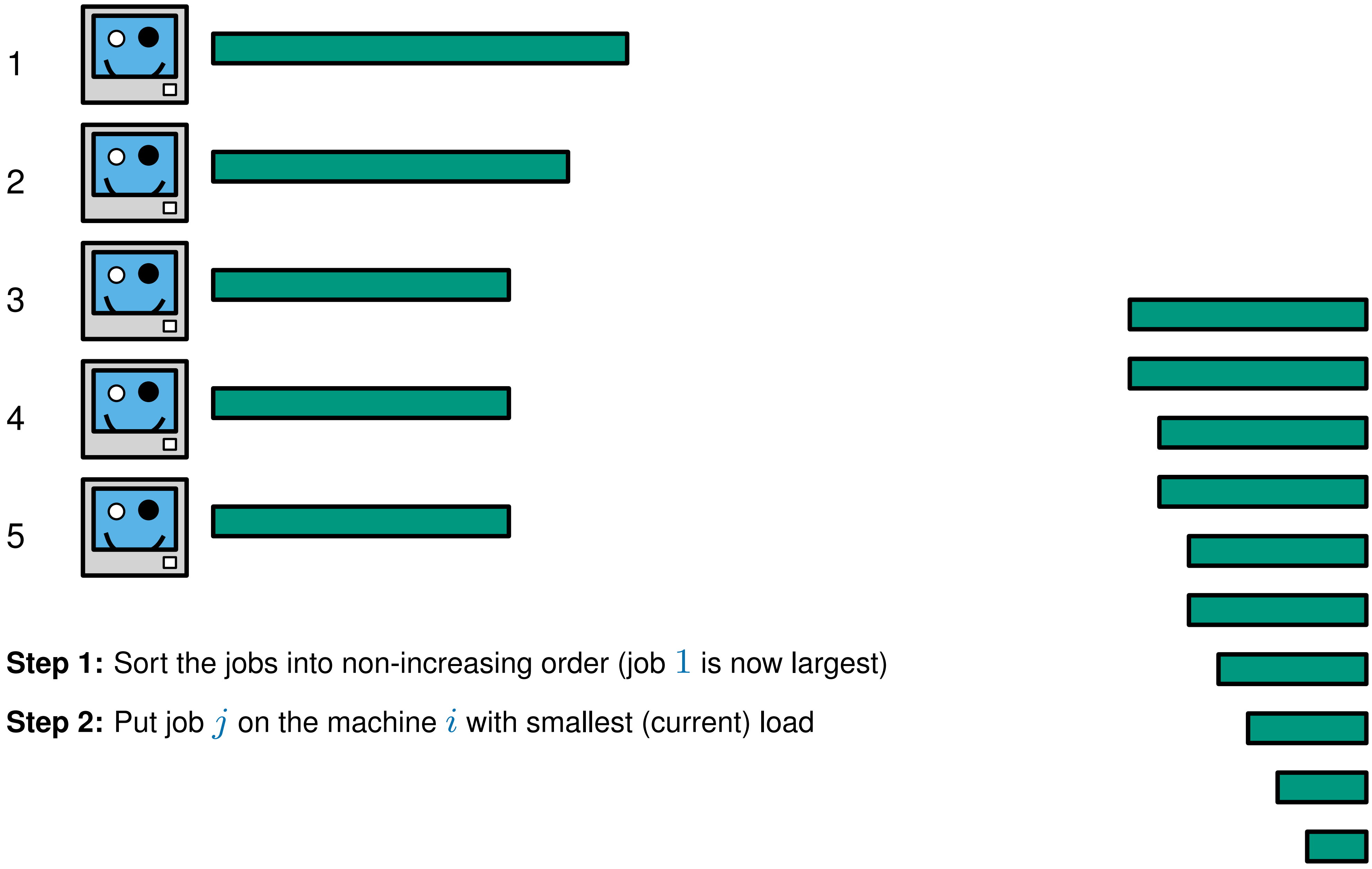
Longest Processing Time (LPT)

Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job j on the machine i with smallest (current) load

University of BRISTOL

Longest Processing Time (LPT)



Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job j on the machine i with smallest (current) load

Longest Processing Time (LPT)

Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

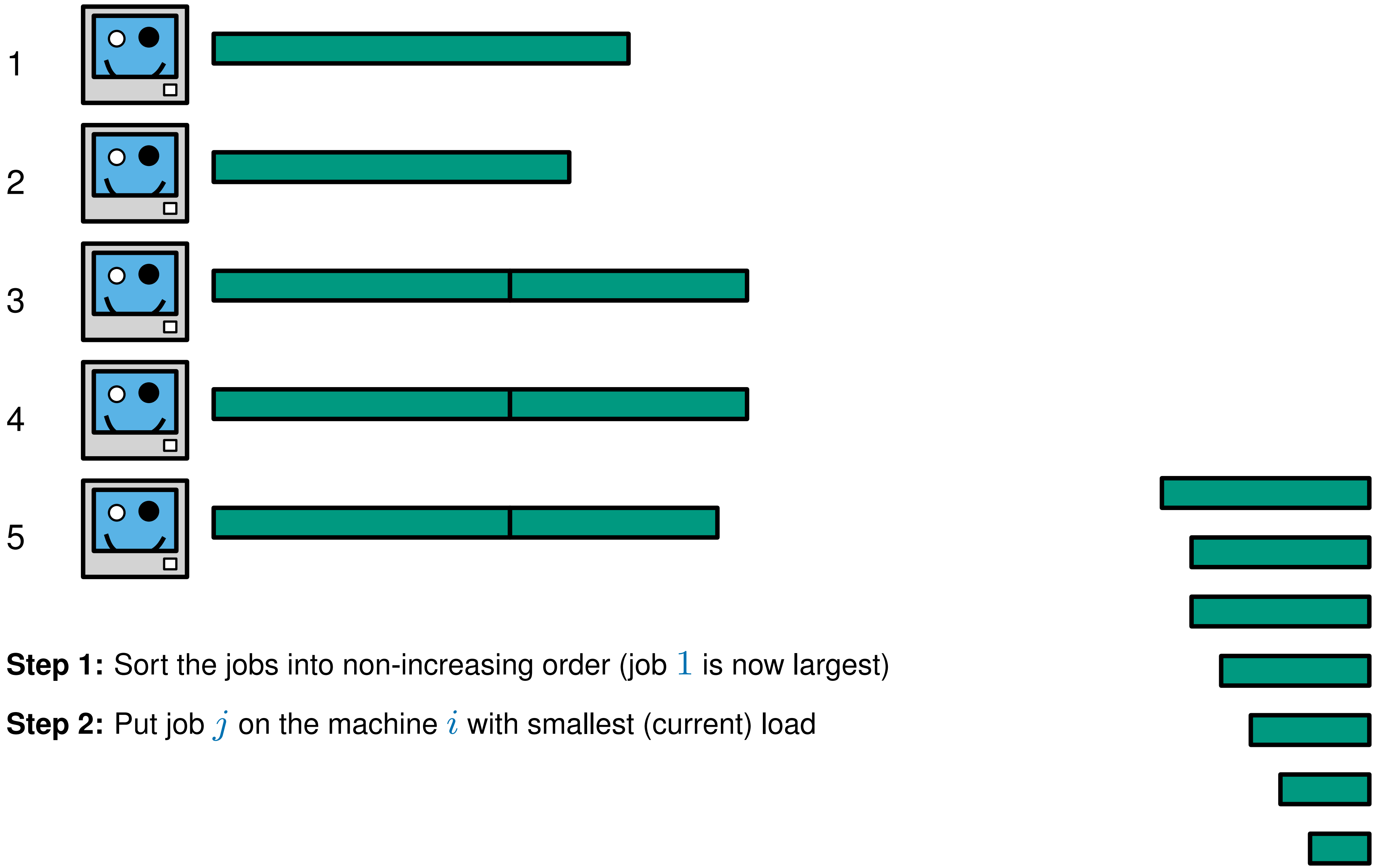
Step 2: Put job j on the machine i with smallest (current) load

Longest Processing Time (LPT)

Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job j on the machine i with smallest (current) load

Longest Processing Time (LPT)



Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job j on the machine i with smallest (current) load

Longest Processing Time (LPT)

Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

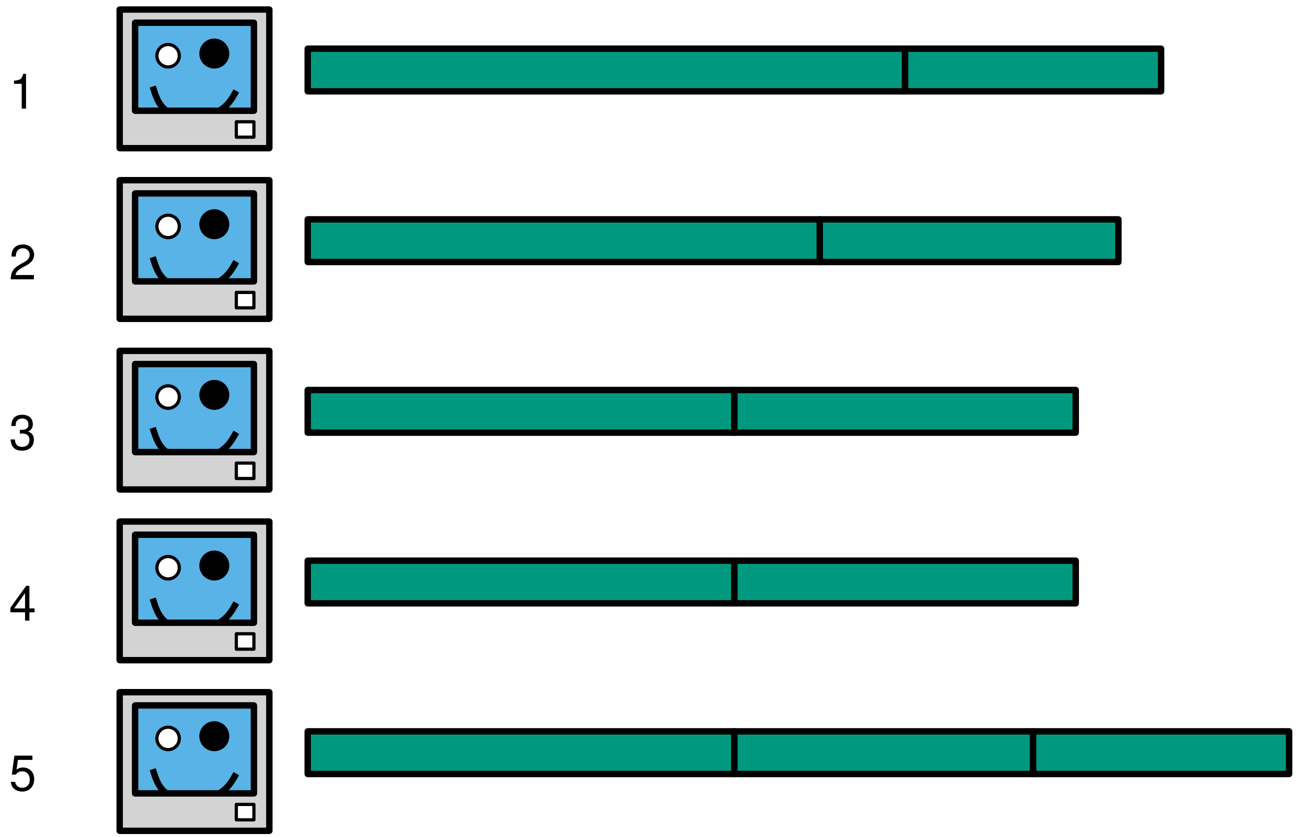
Step 2: Put job j on the machine i with smallest (current) load

Longest Processing Time (LPT)

Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

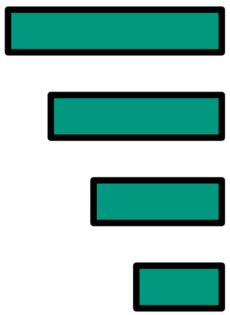
Step 2: Put job j on the machine i with smallest (current) load

Longest Processing Time (LPT)

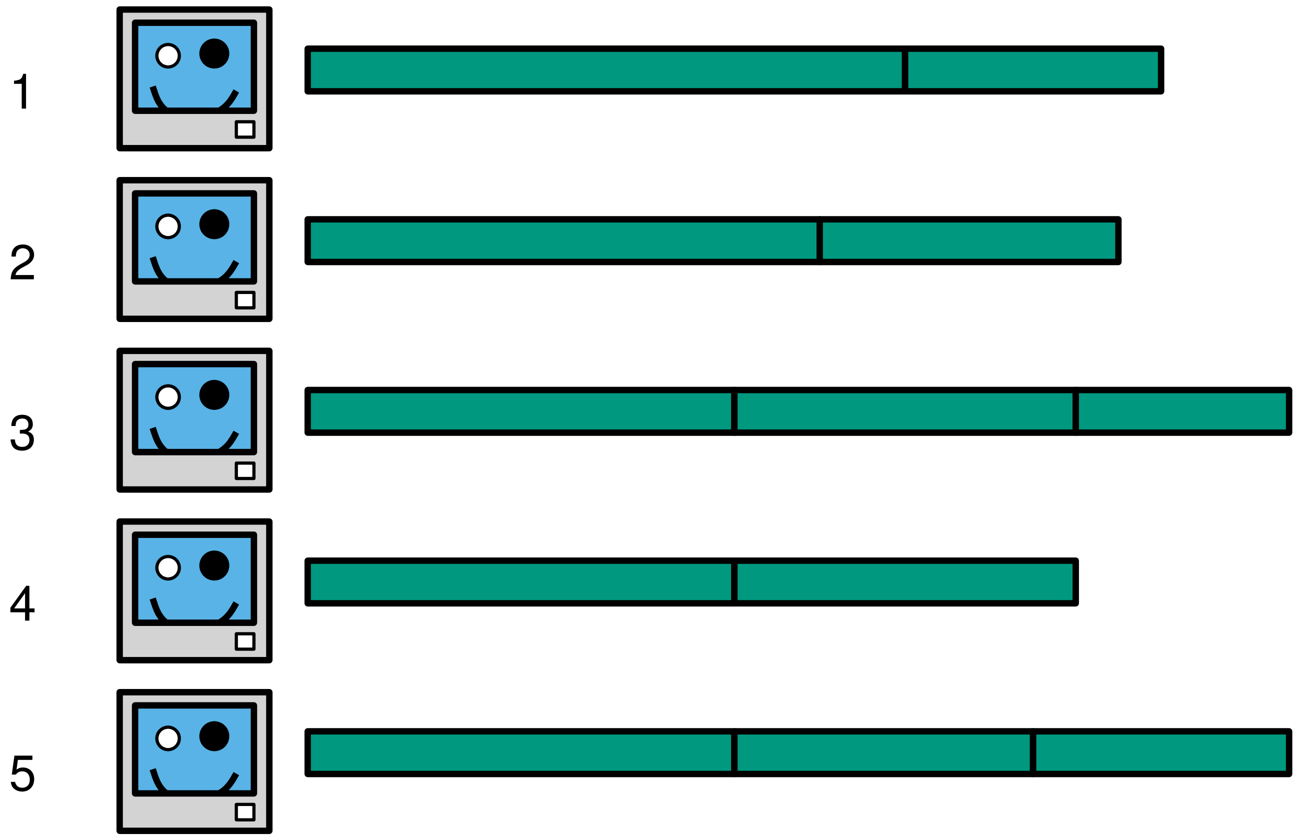


Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job j on the machine i with smallest (current) load

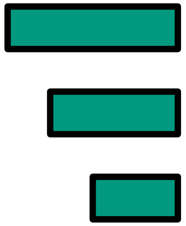


Longest Processing Time (LPT)

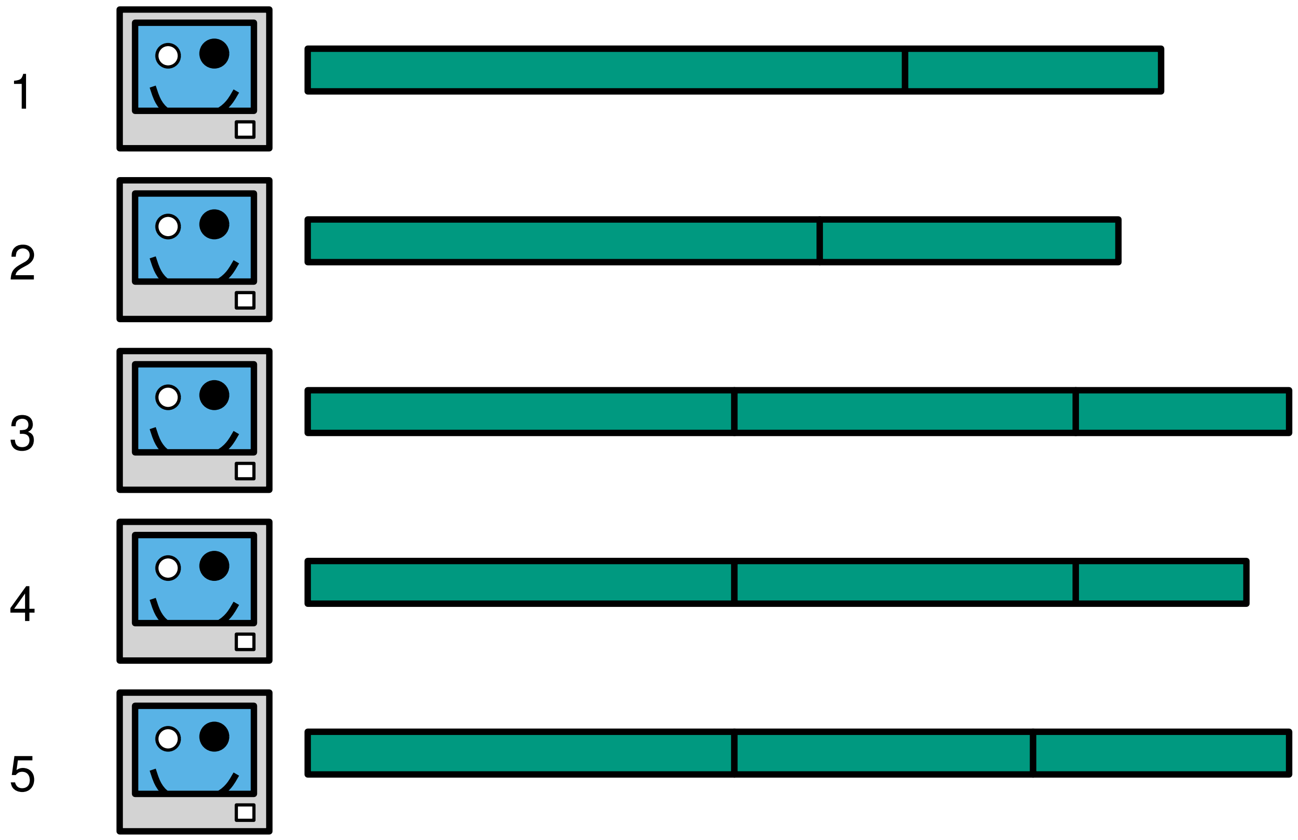


Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job j on the machine i with smallest (current) load

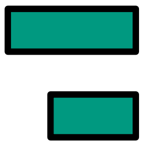


Longest Processing Time (LPT)

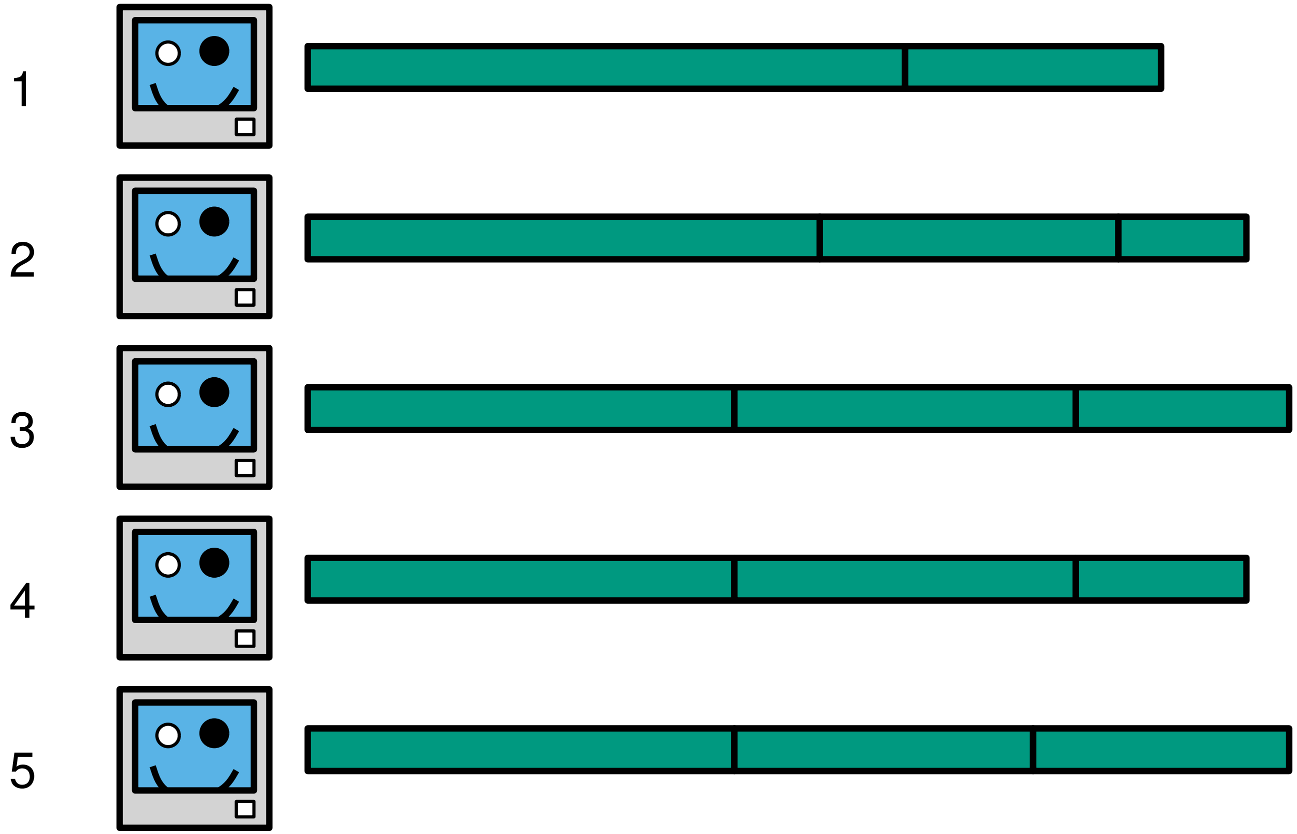


Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job j on the machine i with smallest (current) load



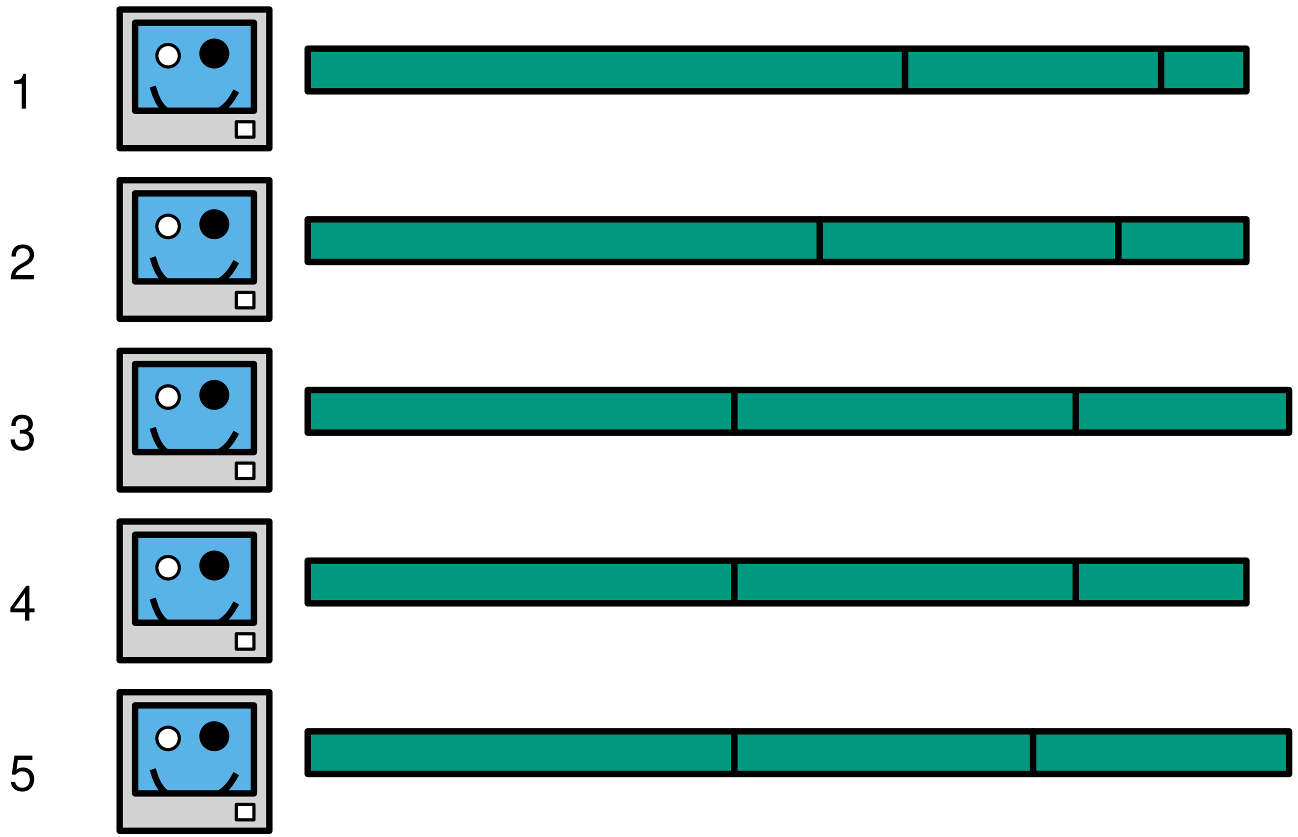
Longest Processing Time (LPT)



Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job j on the machine i with smallest (current) load

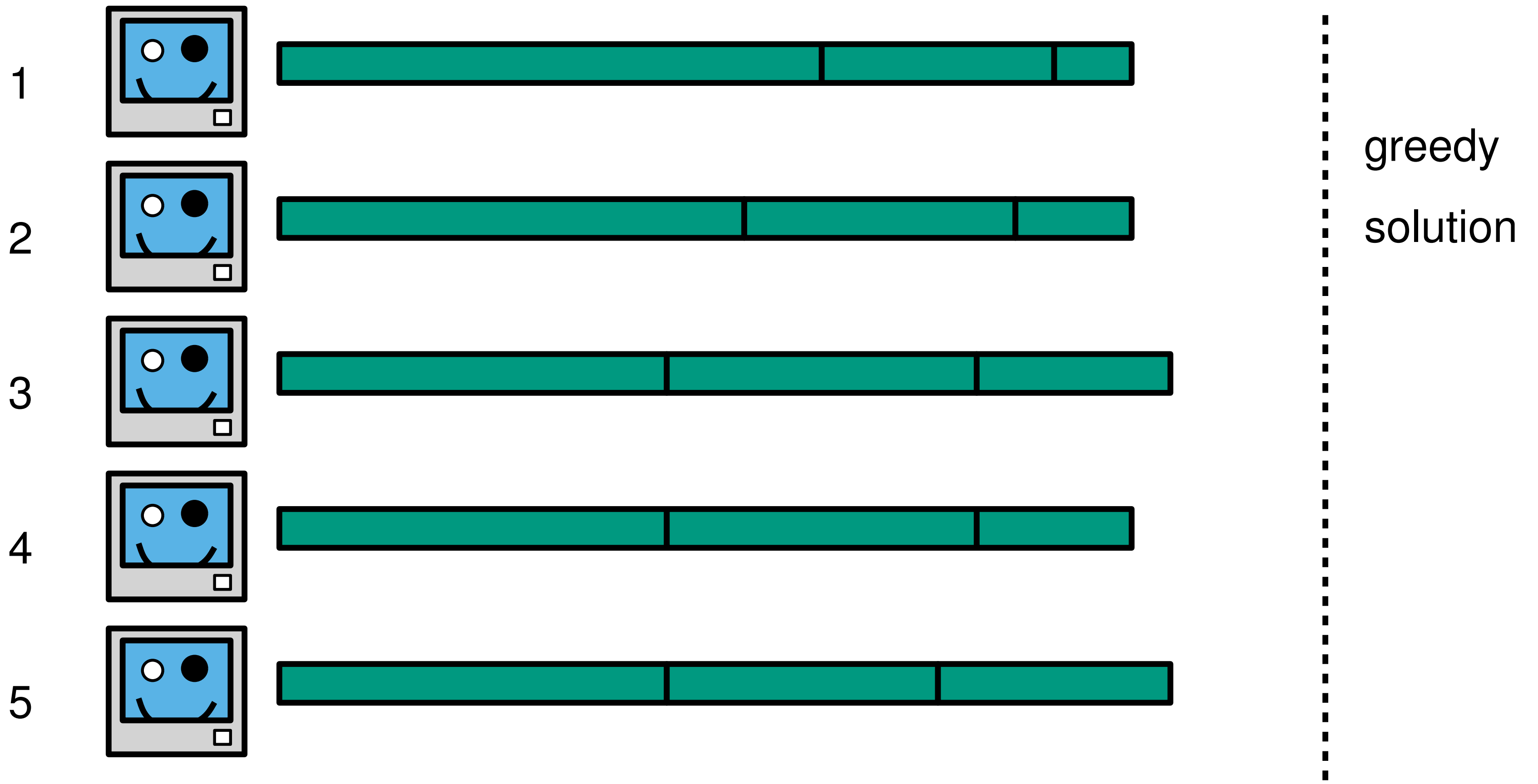
Longest Processing Time (LPT)



Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job j on the machine i with smallest (current) load

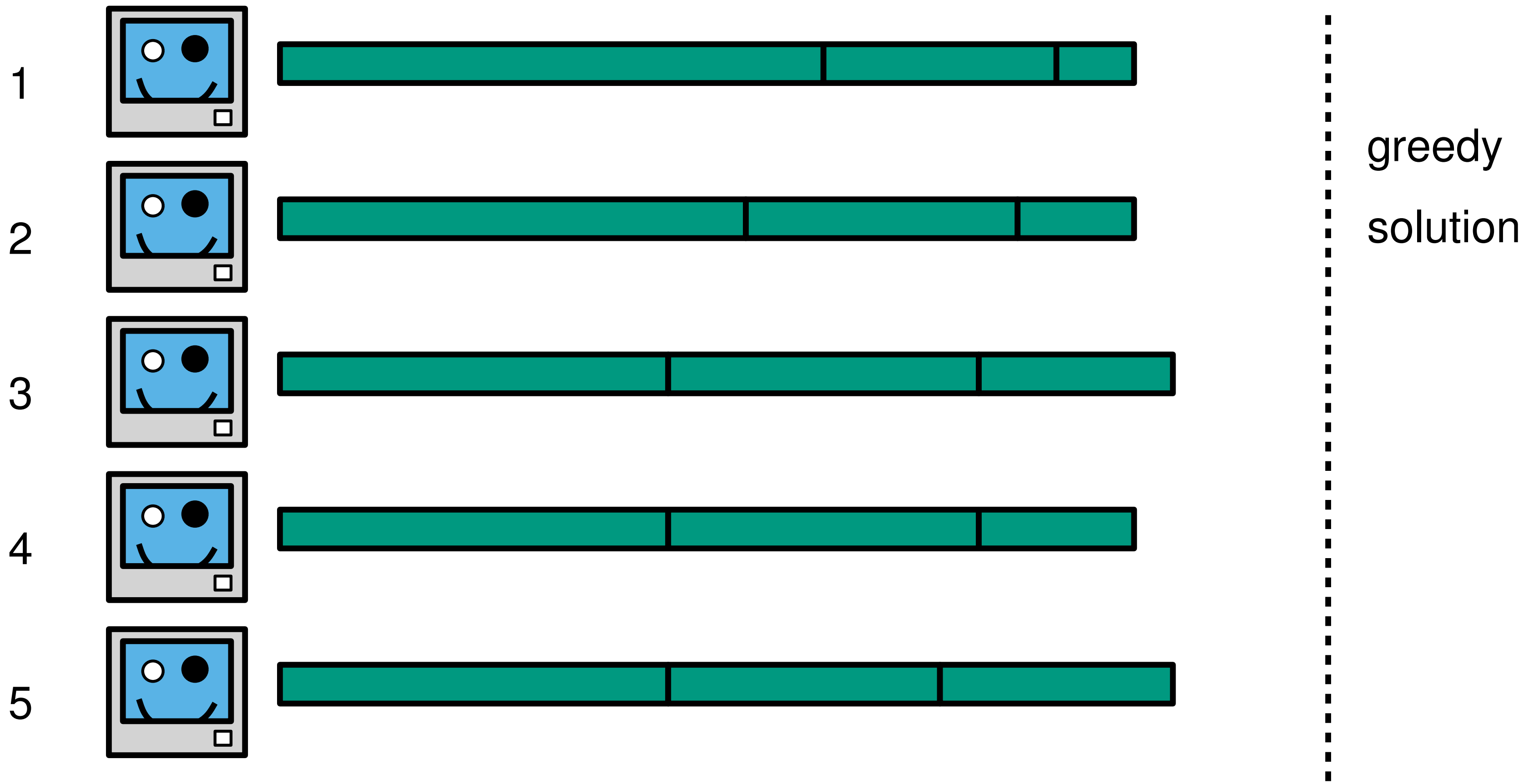
Longest Processing Time (LPT)



Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job j on the machine i with smallest (current) load

Longest Processing Time (LPT)

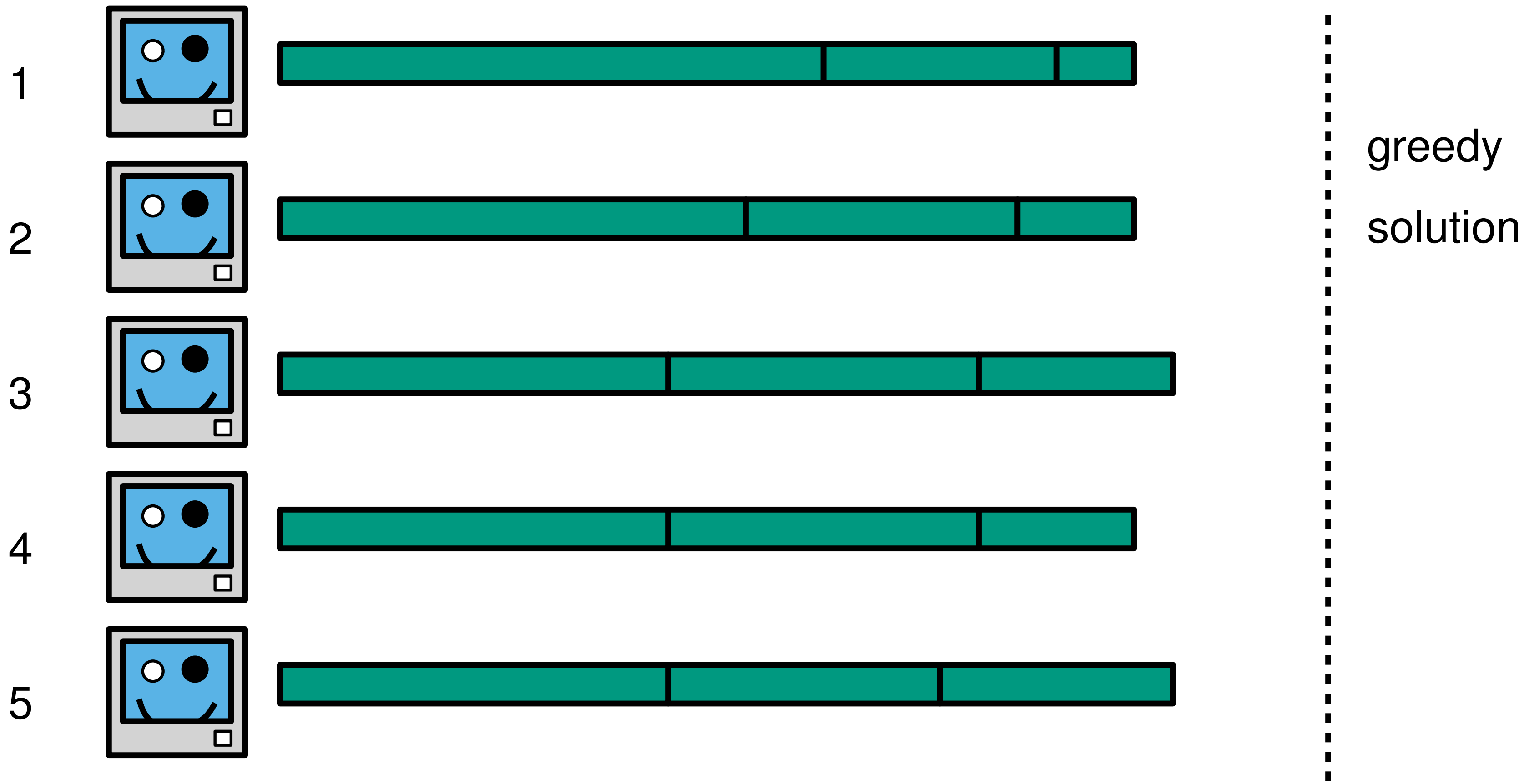


Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job j on the machine i with smallest (current) load

How long does it take to compute this schedule?

Longest Processing Time (LPT)



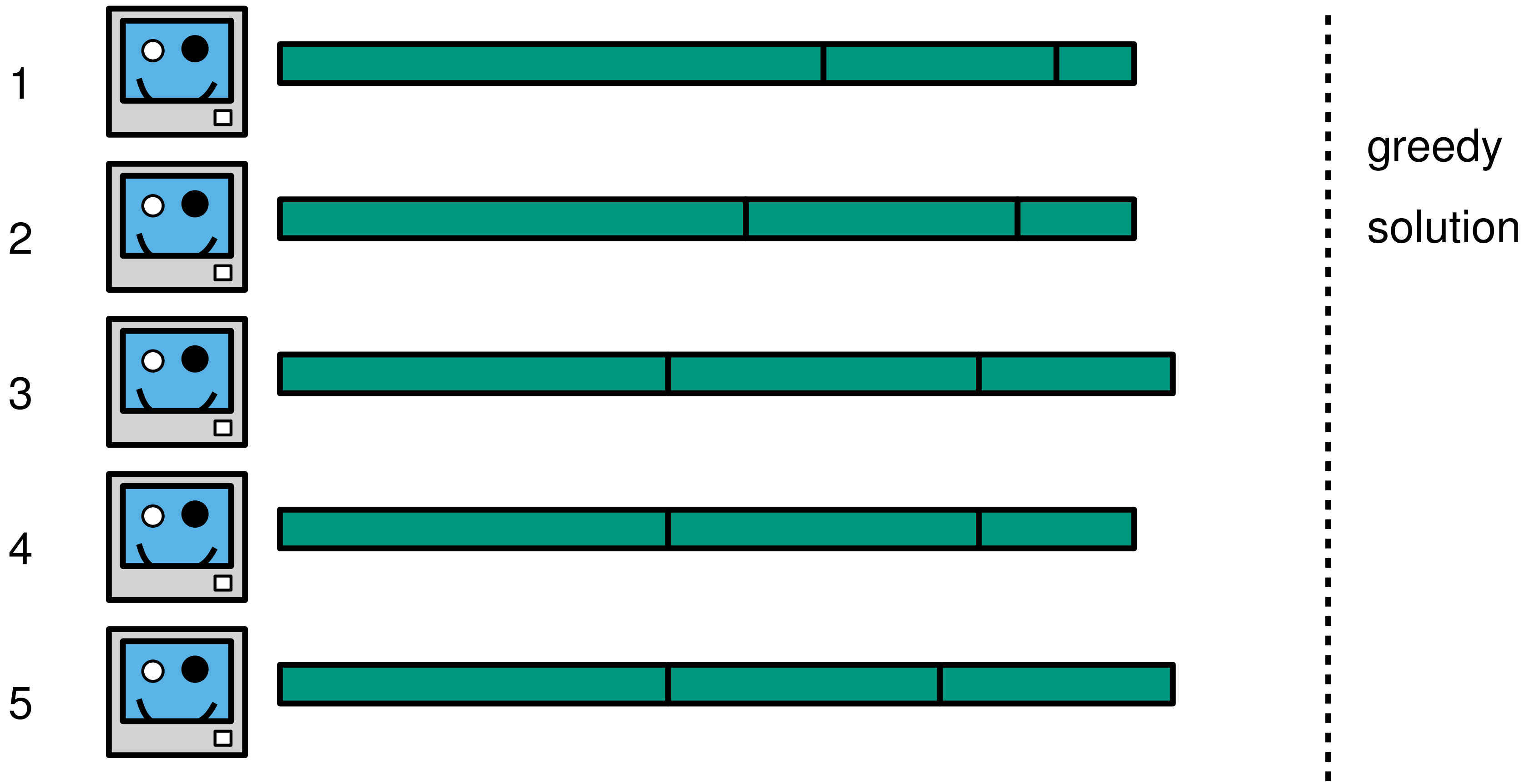
Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job j on the machine i with smallest (current) load

How long does it take to compute this schedule?

$O(n \log n)$ time (to sort the jobs)

Longest Processing Time (LPT)



Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job j on the machine i with smallest (current) load

How long does it take to compute this schedule?

$O(n \log n)$ time (to sort the jobs)

How good is it?

The LPT approximation

- Let T_l denote the time taken by the LPT schedule

Theorem The LPT algorithm is a $3/2$-approximation algorithm

L_i is the *load* of machine i

m machines

n jobs

Job j takes t_j time units

University of BRISTOL

The LPT approximation

- Let T_l denote the time taken by the LPT schedule

Theorem The LPT algorithm is a $3/2$-approximation algorithm

- Before we prove this, we prove another useful fact and a Lemma

L_i is the *load* of machine i

m machines

n jobs

Job j takes t_j time units

University of BRISTOL

The LPT approximation

- Let T_l denote the time taken by the LPT schedule

Theorem The LPT algorithm is a $3/2$-approximation algorithm

- Before we prove this, we prove another useful fact and a Lemma

Fact If there are at most m jobs $(n \leqslant m)$ then LPT is optimal

L_i is the *load* of machine i

m machines
n jobs

Job j takes t_j time units

The LPT approximation

- Let T_l denote the time taken by the LPT schedule

Theorem The LPT algorithm is a $3/2$-approximation algorithm

- Before we prove this, we prove another useful fact and a Lemma

Fact If there are at most m jobs $(n \leqslant m)$ then LPT is optimal

If there are at most m jobs then

LPT gives each job its own machine so $\max_i L_i \leqslant \max_j t_j \leqslant \mathrm{Opt}$

L_i is the *load* of machine i

m machines

n jobs

Job j takes t_j time units

The LPT approximation

- Let T_l denote the time taken by the LPT schedule

Theorem The LPT algorithm is a $3/2$-approximation algorithm

- Before we prove this, we prove another useful fact and a Lemma

Fact If there are at most m jobs $(n \leqslant m)$ then LPT is optimal

If there are at most m jobs then

LPT gives each job its own machine so $\max_i L_i \leqslant \max_j t_j \leqslant \mathrm{Opt}$

Lemma If $n > m$ then $\mathrm{Opt} \geqslant 2t_{(m+1)}$ (after sorting)

L_i is the *load* of machine i

m machines

n jobs

Job j takes t_j time units

The LPT approximation

- Let T_l denote the time taken by the LPT schedule

L_i is the *load* of machine i

Theorem The LPT algorithm is a $3/2$-approximation algorithm

- Before we prove this, we prove another useful fact and a Lemma

m machines

n jobs

Fact If there are at most m jobs $(n \leqslant m)$ then LPT is optimal

If there are at most m jobs then

LPT gives each job its own machine so $\max_i L_i \leqslant \max_j t_j \leqslant \mathrm{Opt}$

Job j takes t_j time units

Lemma If $n > m$ then $\mathrm{Opt} \geqslant 2t_{(m+1)}$ (after sorting)

Proof

The LPT approximation

- Let T_l denote the time taken by the LPT schedule

Theorem The LPT algorithm is a $3/2$-approximation algorithm

- Before we prove this, we prove another useful fact and a Lemma

Fact If there are at most m jobs $(n \leqslant m)$ then LPT is optimal

If there are at most m jobs then

LPT gives each job its own machine so $\max_i L_i \leqslant \max_j t_j \leqslant \mathrm{Opt}$

Lemma If $n > m$ then $\mathrm{Opt} \geqslant 2t_{(m+1)}$ (after sorting)

Proof

- Note that $t_1 \geqslant t_2 \geqslant t_3 \geqslant \ldots t_m \geqslant t_{(m+1)}$

L_i is the *load* of machine i

m machines

n jobs

Job j takes t_j time units

The LPT approximation

- Let T_l denote the time taken by the LPT schedule

L_i is the *load* of machine i

Theorem The LPT algorithm is a $3/2$-approximation algorithm

- Before we prove this, we prove another useful fact and a Lemma

m machines

n jobs

Fact If there are at most m jobs $(n \leqslant m)$ then LPT is optimal

If there are at most m jobs then

LPT gives each job its own machine so $\max_i L_i \leqslant \max_j t_j \leqslant \mathrm{Opt}$

Job j takes t_j time units

Lemma If $n > m$ then $\mathrm{Opt} \geqslant 2t_{(m+1)}$ (after sorting)

Proof

- Note that $t_1 \geqslant t_2 \geqslant t_3 \geqslant \ldots t_m \geqslant t_{(m+1)}$
- One of the m machines must be assigned

  (at least) two of these $m+1$ jobs *under any schedule*

The LPT approximation

- Let T_l denote the time taken by the LPT schedule

Theorem The LPT algorithm is a $3/2$-approximation algorithm

- Before we prove this, we prove another useful fact and a Lemma

Fact If there are at most m jobs $(n \leqslant m)$ then LPT is optimal

If there are at most m jobs then

LPT gives each job its own machine so $\max_i L_i \leqslant \max_j t_j \leqslant \mathrm{Opt}$

Lemma If $n > m$ then $\mathrm{Opt} \geqslant 2t_{(m+1)}$ (after sorting)

Proof

- Note that $t_1 \geqslant t_2 \geqslant t_3 \geqslant \ldots t_m \geqslant t_{(m+1)}$
- One of the m machines must be assigned (at least) two of these $m+1$ jobs *under any schedule*
- So we have that any schedule takes at least $2t_{(m+1)}$ time

L_i is the *load* of machine i

m machines

n jobs

Job j takes t_j time units

The LPT approximation

- Let T_l denote the time taken by the LPT schedule

L_i is the *load* of machine i

Theorem The LPT algorithm is a $3/2$-approximation algorithm

- Before we prove this, we prove another useful fact and a Lemma

m machines

n jobs

Fact If there are at most m jobs $(n \leqslant m)$ then LPT is optimal

If there are at most m jobs then

LPT gives each job its own machine so $\max_i L_i \leqslant \max_j t_j \leqslant \mathrm{Opt}$

Job j takes t_j time units

Lemma If $n > m$ then $\mathrm{Opt} \geqslant 2t_{(m+1)}$ (after sorting)

Proof

- Note that $t_1 \geqslant t_2 \geqslant t_3 \geqslant \ldots t_m \geqslant t_{(m+1)}$
- One of the m machines must be assigned

  (at least) two of these $m+1$ jobs *under any schedule*
- So we have that any schedule takes at least $2t_{(m+1)}$ time

  in particular $\mathrm{Opt} \geqslant 2t_{(m+1)}$

The LPT approximation

L_i is the *load* of machine i

Theorem The LPT algorithm is a $3/2$-approximation algorithm

m machines

n jobs

Job j takes t_j time units

The LPT approximation

L_i is the *load* of machine i

Theorem The LPT algorithm is a $3/2$-approximation algorithm

Proof Consider the machine i with largest load $T_l = L_i$

m machines

n jobs

Job j takes t_j time units

The LPT approximation

L_i is the *load* of machine i

Theorem The LPT algorithm is a $3/2$-approximation algorithm

Proof Consider the machine i with largest load $T_l = L_i$

- Let j denote the last job machine i completes

m machines

n jobs

Job j takes t_j time units

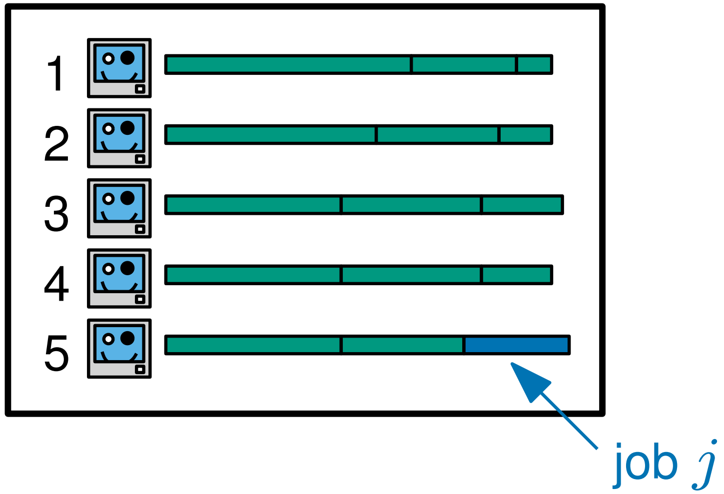
The LPT approximation

L_i is the *load* of machine i

Theorem The LPT algorithm is a $3/2$-approximation algorithm

Proof Consider the machine i with largest load $T_l = L_i$

- Let j denote the last job machine i completes



m machines

n jobs

Job j takes t_j time units

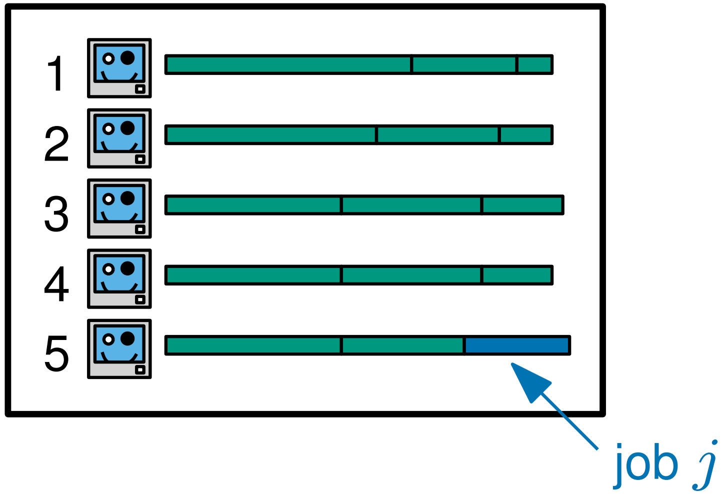
The LPT approximation

L_i is the *load* of machine i

Theorem The LPT algorithm is a $3/2$-approximation algorithm

Proof Consider the machine i with largest load $T_l = L_i$

- Let j denote the last job machine i completes
- Using the same argument as before, we have that,



m machines

n jobs

Job j takes t_j time units

The LPT approximation

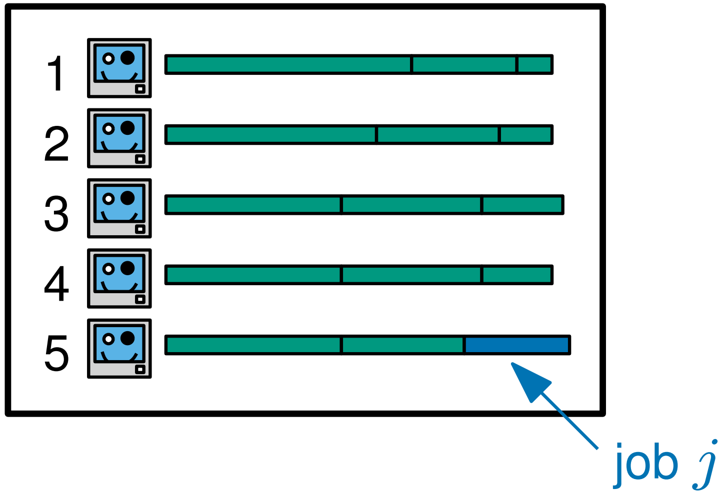
L_i is the *load* of machine i

Theorem The LPT algorithm is a $3/2$-approximation algorithm

Proof Consider the machine i with largest load $T_l = L_i$

- Let j denote the last job machine i completes
- Using the same argument as before, we have that,

$$(L_i - t_j) \leqslant \mathrm{Opt}$$



m machines

n jobs

Job j takes t_j time units

The LPT approximation

L_i is the *load* of machine i

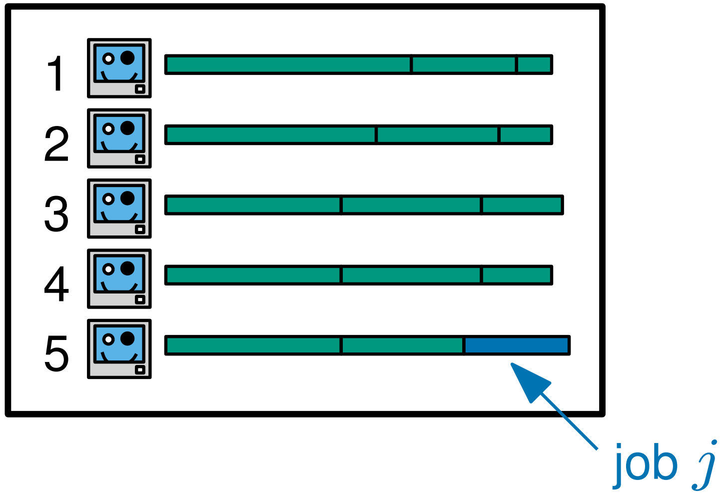
Theorem The LPT algorithm is a $3/2$-approximation algorithm

Proof Consider the machine i with largest load $T_l = L_i$

- Let j denote the last job machine i completes
- Using the same argument as before, we have that,

$$(L_i - t_j) \leqslant \mathrm{Opt}$$

- If $n \leqslant m$ then we are done so assume $n > m$



m machines

n jobs

Job j takes t_j time units

The LPT approximation

L_i is the *load* of machine i

Theorem The LPT algorithm is a $3/2$-approximation algorithm

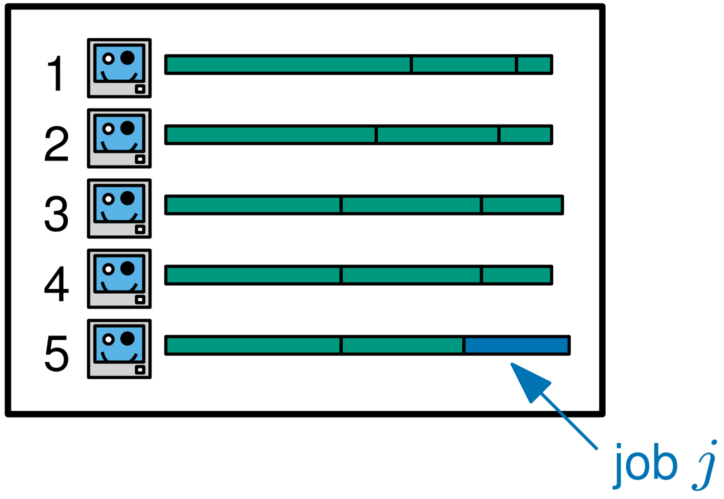
Proof Consider the machine i with largest load $T_l = L_i$

- Let j denote the last job machine i completes
- Using the same argument as before, we have that,

$$(L_i - t_j) \leqslant \text{Opt}$$

- If $n \leqslant m$ then we are done so assume $n > m$

because LPT is optimal in this case



m machines

n jobs

Job j takes t_j time units

The LPT approximation

L_i is the *load* of machine i

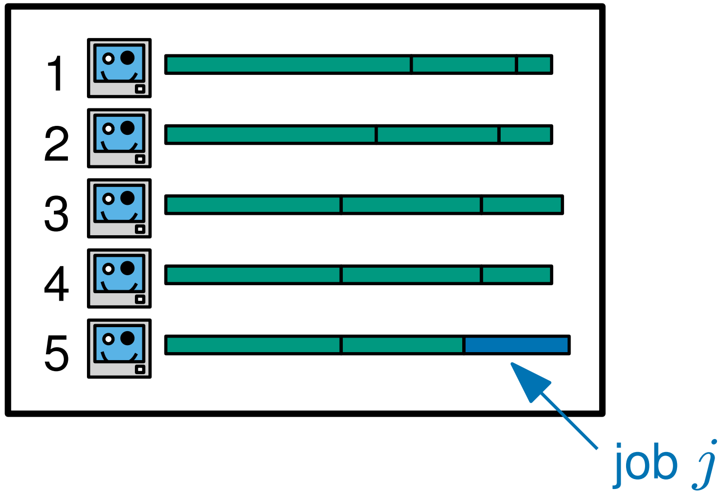
Theorem The LPT algorithm is a $3/2$-approximation algorithm

Proof Consider the machine i with largest load $T_l = L_i$

- Let j denote the last job machine i completes
- Using the same argument as before, we have that,

$$(L_i - t_j) \leqslant \mathrm{Opt}$$

- If $n \leqslant m$ then we are done so assume $n > m$



m machines

n jobs

Job j takes t_j time units

The LPT approximation

L_i is the *load* of machine i

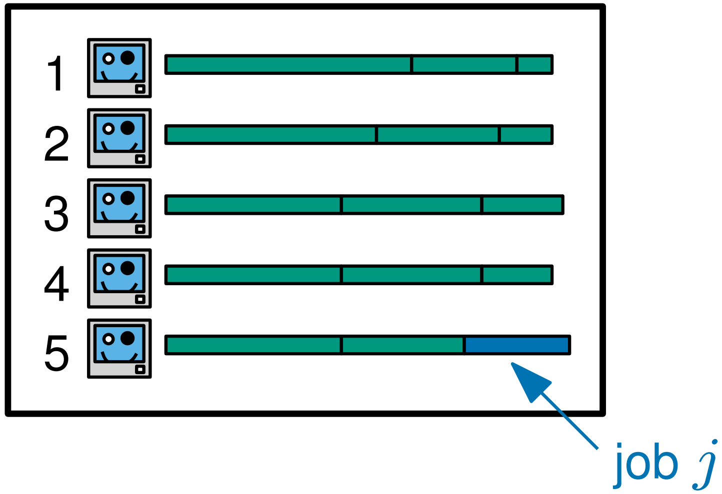
Theorem The LPT algorithm is a $3/2$-approximation algorithm

Proof Consider the machine i with largest load $T_l = L_i$

- Let j denote the last job machine i completes
- Using the same argument as before, we have that,

$$(L_i - t_j) \leqslant \mathrm{Opt}$$

- If $n \leqslant m$ then we are done so assume $n > m$
- Further if $(L_i - t_j) = 0$ then $T_l = L_i = t_j \leqslant \mathrm{Opt}$



m machines

n jobs

Job j takes t_j time units

The LPT approximation

L_i is the *load* of machine i

Theorem The LPT algorithm is a $3/2$-approximation algorithm

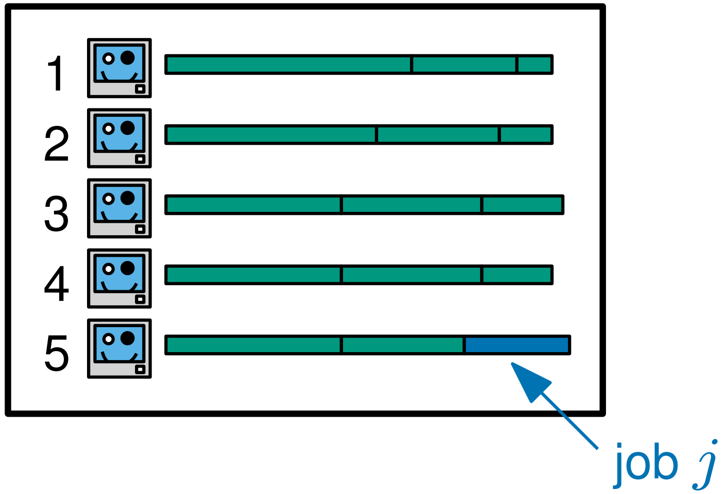
Proof Consider the machine i with largest load $T_l = L_i$

- Let j denote the last job machine i completes
- Using the same argument as before, we have that,

$$(L_i - t_j) \leqslant \mathrm{Opt}$$

- If $n \leqslant m$ then we are done so assume $n > m$
- Further if $(L_i - t_j) = 0$ then $T_l = L_i = t_j \leqslant \mathrm{Opt}$

Fact $\mathrm{Opt} \geqslant \max_j t_j$



m machines
n jobs

Job j takes t_j time units

The LPT approximation

L_i is the *load* of machine i

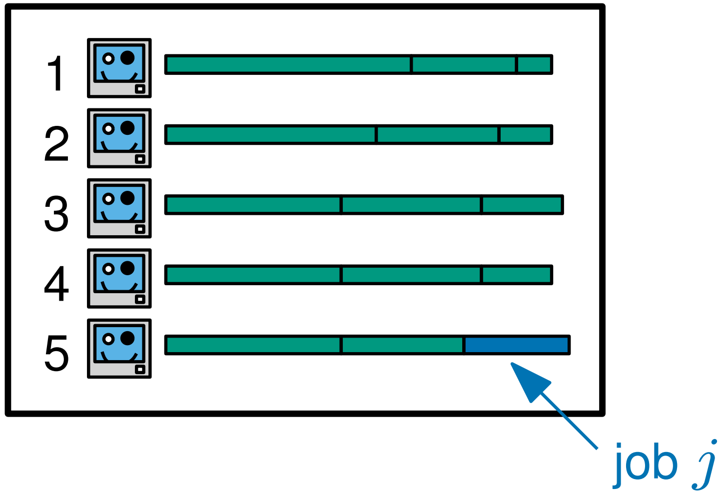
Theorem The LPT algorithm is a $3/2$-approximation algorithm

Proof Consider the machine i with largest load $T_l = L_i$

- Let j denote the last job machine i completes
- Using the same argument as before, we have that,

$$(L_i - t_j) \leqslant \mathrm{Opt}$$

- If $n \leqslant m$ then we are done so assume $n > m$
- Further if $(L_i - t_j) = 0$ then $T_l = L_i = t_j \leqslant \mathrm{Opt}$



m machines

n jobs

Job j takes t_j time units

The LPT approximation

L_i is the *load* of machine i

Theorem The LPT algorithm is a $3/2$-approximation algorithm

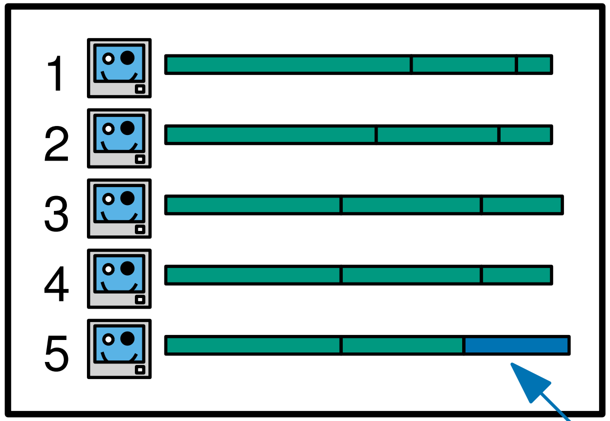
Proof Consider the machine i with largest load $T_l = L_i$

- Let j denote the last job machine i completes
- Using the same argument as before, we have that,

$$(L_i - t_j) \leqslant \text{Opt}$$

- If $n \leqslant m$ then we are done so assume $n > m$
- Further if $(L_i - t_j) = 0$ then $T_l = L_i = t_j \leqslant \text{Opt}$

so assume that $(L_i - t_j) > 0$



job j

m machines

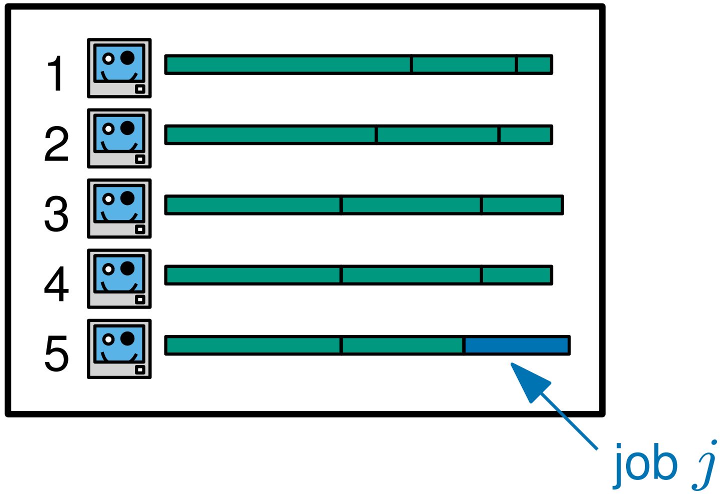
n jobs

Job j takes t_j time units

The LPT approximation

L_i is the *load* of machine i

Theorem The LPT algorithm is a $3/2$-approximation algorithm



Proof Consider the machine i with largest load $T_l = L_i$

- Let j denote the last job machine i completes
- Using the same argument as before, we have that,

$$(L_i - t_j) \leqslant \mathrm{Opt}$$

- If $n \leqslant m$ then we are done so assume $n > m$
- Further if $(L_i - t_j) = 0$ then $T_l = L_i = t_j \leqslant \mathrm{Opt}$ so assume that $(L_i - t_j) > 0$

m machines
n jobs

- Therefore machine i was assigned at least two jobs

Job j takes t_j time units

The LPT approximation

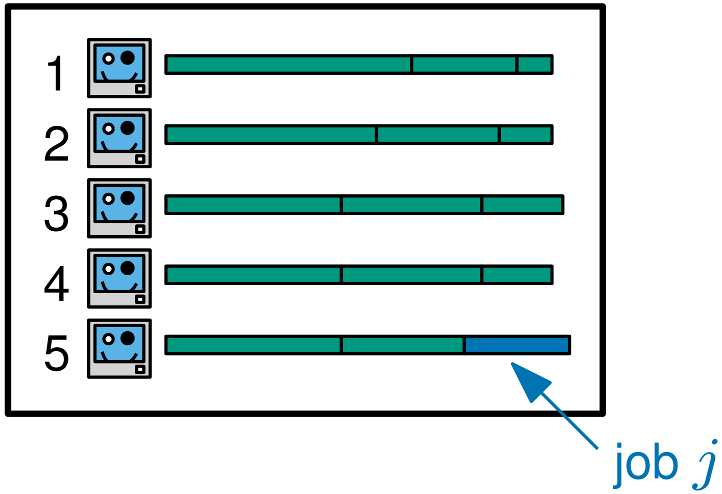
L_i is the *load* of machine i

Theorem The LPT algorithm is a $3/2$-approximation algorithm

Proof Consider the machine i with largest load $T_l = L_i$

- Let j denote the last job machine i completes
- Using the same argument as before, we have that,

$$(L_i - t_j) \leqslant \mathrm{Opt}$$



- If $n \leqslant m$ then we are done so assume $n > m$
- Further if $(L_i - t_j) = 0$ then $T_l = L_i = t_j \leqslant \mathrm{Opt}$

  so assume that $(L_i - t_j) > 0$

m machines

n jobs

- Therefore machine i was assigned at least two jobs

  By the algorithm description, we have that $j \geqslant m + 1$

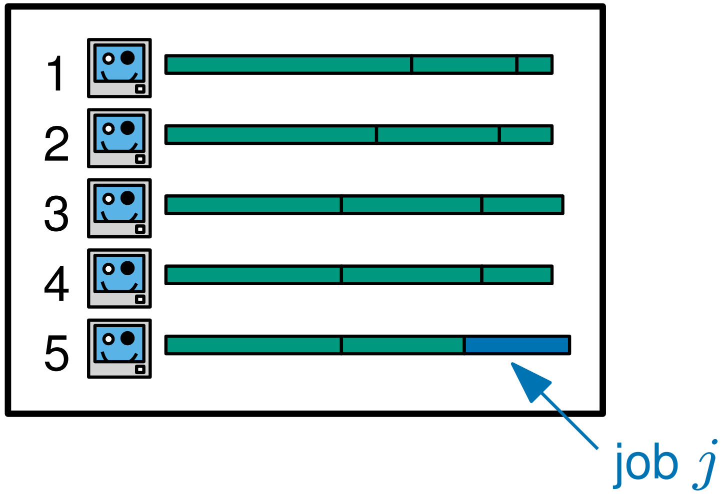
Job j takes t_j time units

The LPT approximation

L_i is the *load* of machine i

Theorem The LPT algorithm is a $3/2$-approximation algorithm

Proof Consider the machine i with largest load $T_l = L_i$



- Let j denote the last job machine i completes
- Using the same argument as before, we have that,

$$(L_i - t_j) \leqslant \mathrm{Opt}$$

- If $n \leqslant m$ then we are done so assume $n > m$
- Further if $(L_i - t_j) = 0$ then $T_l = L_i = t_j \leqslant \mathrm{Opt}$

  so assume that $(L_i - t_j) > 0$

m machines

n jobs

- Therefore machine i was assigned at least two jobs

  By the algorithm description, we have that $j \geqslant m + 1$

Job j takes t_j time units

it doesn't assign a second job to any machine until every machine has at least one job

The LPT approximation

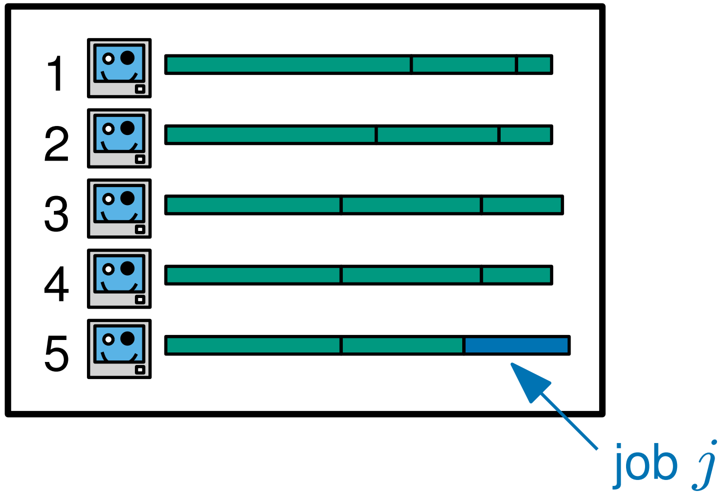
L_i is the *load* of machine i

Theorem The LPT algorithm is a $3/2$-approximation algorithm

Proof Consider the machine i with largest load $T_l = L_i$

- Let j denote the last job machine i completes
- Using the same argument as before, we have that,

$$(L_i - t_j) \leqslant \text{Opt}$$



- If $n \leqslant m$ then we are done so assume $n > m$
- Further if $(L_i - t_j) = 0$ then $T_l = L_i = t_j \leqslant \text{Opt}$

  so assume that $(L_i - t_j) > 0$

m machines

n jobs

- Therefore machine i was assigned at least two jobs

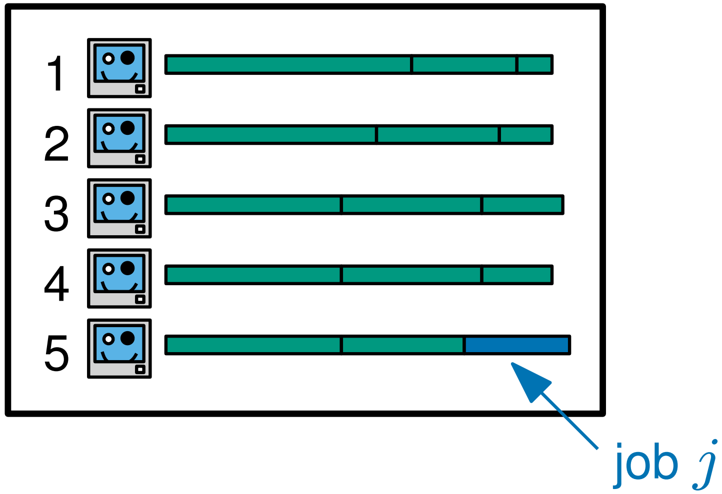
  By the algorithm description, we have that $j \geqslant m + 1$

Job j takes t_j time units

The LPT approximation

L_i is the *load* of machine i

Theorem The LPT algorithm is a $3/2$-approximation algorithm



Proof Consider the machine i with largest load $T_l = L_i$

- Let j denote the last job machine i completes

- Using the same argument as before, we have that,

$$(L_i - t_j) \leqslant \text{Opt}$$

- If $n \leqslant m$ then we are done so assume $n > m$

- Further if $(L_i - t_j) = 0$ then $T_l = L_i = t_j \leqslant \text{Opt}$

  so assume that $(L_i - t_j) > 0$

m machines

n jobs

- Therefore machine i was assigned at least two jobs

  By the algorithm description, we have that $j \geqslant m + 1$

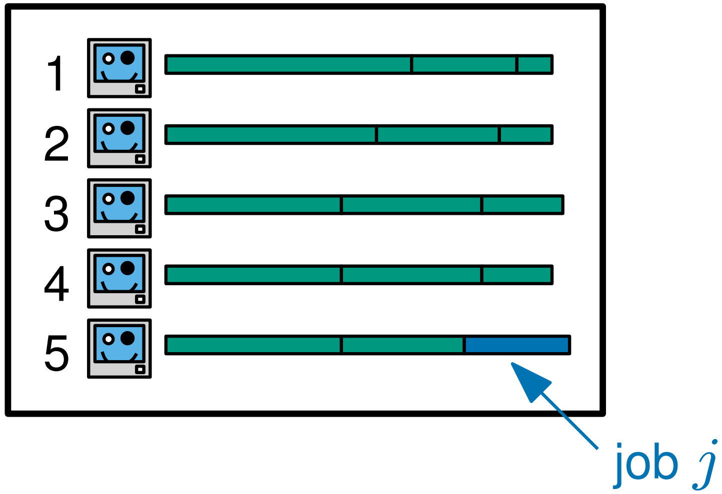
  $t_j \leqslant t_{m+1} \leqslant \text{Opt}/2$ (by the Lemma)

Job j takes t_j time units

The LPT approximation

L_i is the *load* of machine i

Theorem The LPT algorithm is a $3/2$-approximation algorithm



Proof Consider the machine i with largest load $T_l = L_i$

- Let j denote the last job machine i completes
- Using the same argument as before, we have that,

$$(L_i - t_j) \leqslant \text{Opt}$$

- If $n \leqslant m$ then we are done so assume $n > m$
- Further if $(L_i - t_j) = 0$ then $T_l = L_i = t_j \leqslant \text{Opt}$

  so assume that $(L_i - t_j) > 0$

m machines
n jobs

- Therefore machine i was assigned at least two jobs

  By the algorithm description, we have that $j \geqslant m + 1$

$$t_j \leqslant t_{m+1} \leqslant \text{Opt}/2 \text{ (by the Lemma)}$$

Job j takes t_j time units

Lemma If $n > m$ then $\text{Opt} \geqslant 2t_{(m+1)}$ (after sorting)

University of BRISTOL

The LPT approximation

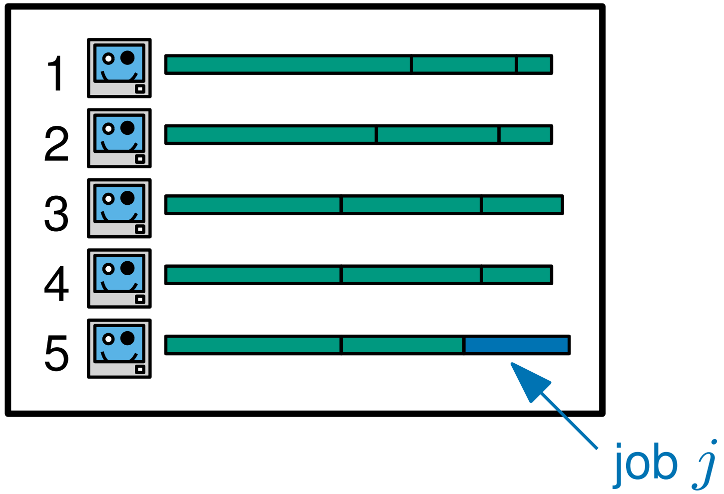
L_i is the *load* of machine i

Theorem The LPT algorithm is a $3/2$-approximation algorithm

Proof Consider the machine i with largest load $T_l = L_i$

- Let j denote the last job machine i completes
- Using the same argument as before, we have that,

$$(L_i - t_j) \leqslant \text{Opt}$$



- If $n \leqslant m$ then we are done so assume $n > m$
- Further if $(L_i - t_j) = 0$ then $T_l = L_i = t_j \leqslant \text{Opt}$

so assume that $(L_i - t_j) > 0$

m machines

n jobs

- Therefore machine i was assigned at least two jobs

By the algorithm description, we have that $j \geqslant m + 1$

$$t_j \leqslant t_{m+1} \leqslant \text{Opt}/2 \text{ (by the Lemma)}$$

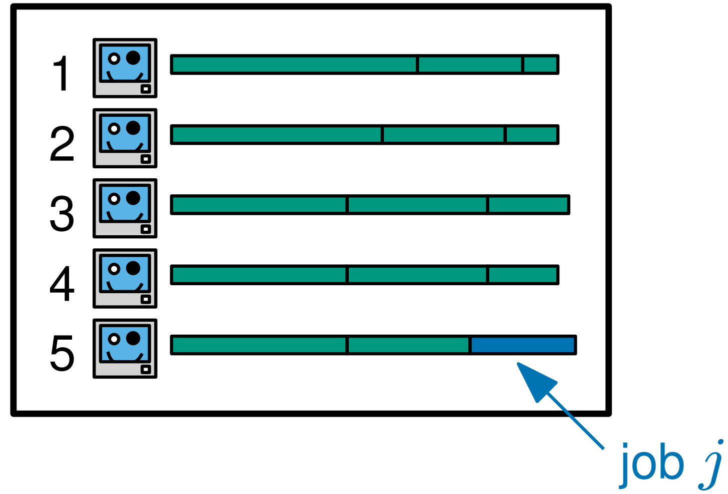
Job j takes t_j time units

The LPT approximation

L_i is the *load* of machine i

Theorem The LPT algorithm is a $3/2$-approximation algorithm

Proof Consider the machine i with largest load $T_l = L_i$



- Let j denote the last job machine i completes
- Using the same argument as before, we have that,

$$(L_i - t_j) \leqslant \mathrm{Opt}$$

- If $n \leqslant m$ then we are done so assume $n > m$
- Further if $(L_i - t_j) = 0$ then $T_l = L_i = t_j \leqslant \mathrm{Opt}$

  so assume that $(L_i - t_j) > 0$

m machines
n jobs

- Therefore machine i was assigned at least two jobs

  By the algorithm description, we have that $j \geqslant m + 1$

$$t_j \leqslant t_{m+1} \leqslant \mathrm{Opt}/2 \text{ (by the Lemma)}$$

Job j takes t_j time units

Therefore, $T_l = L_i = (L_i - t_j) + t_j \leqslant \mathrm{Opt} + \mathrm{Opt}/2 = (3/2) \cdot \mathrm{Opt}$

Scheduling conclusions

m machines

n jobs

Theorem The greedy algorithm is a 2-approximation algorithm

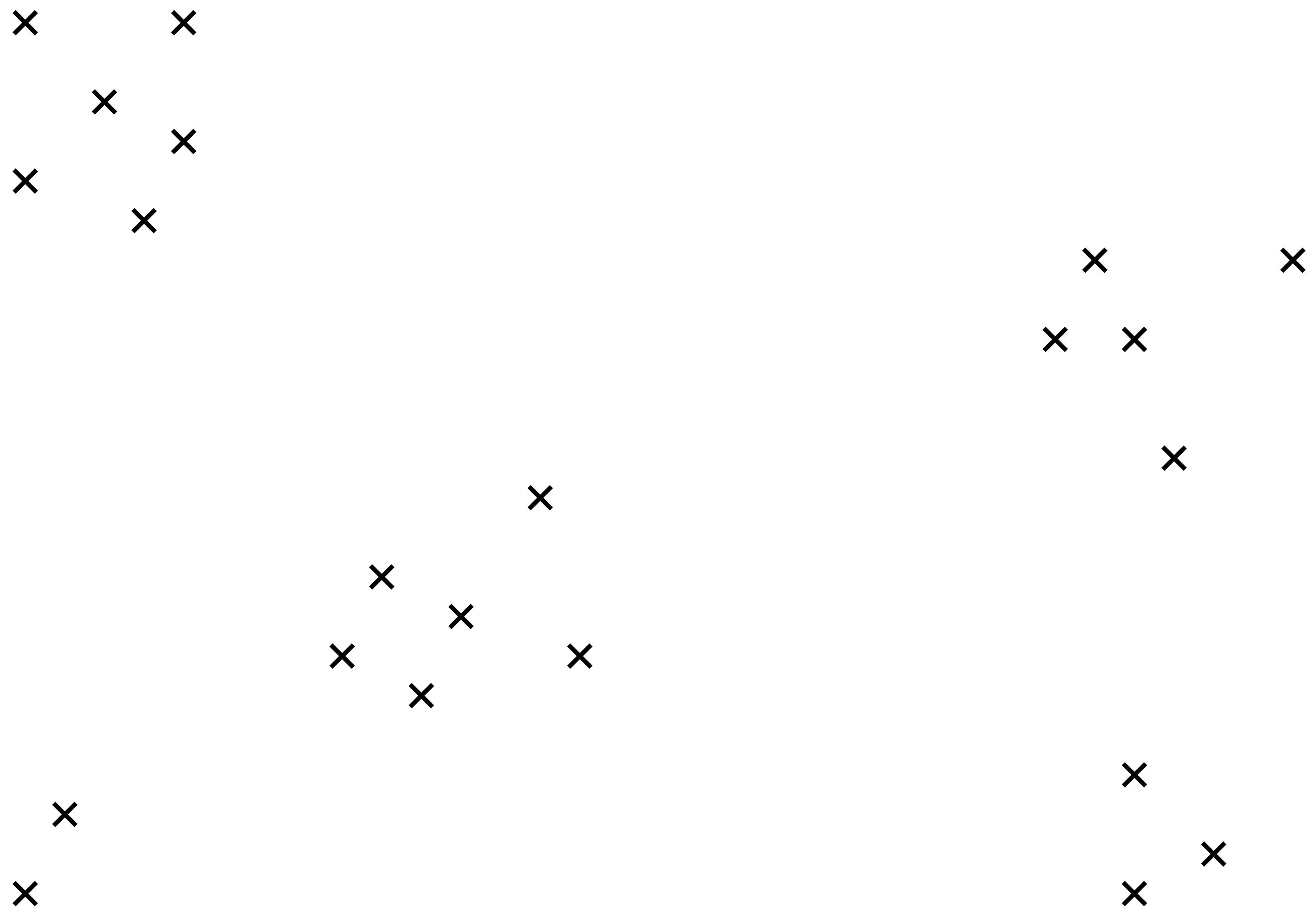
which runs in $O(n \log m)$ time and it's online

Theorem The LPT algorithm is a $3/2$-approximation algorithm

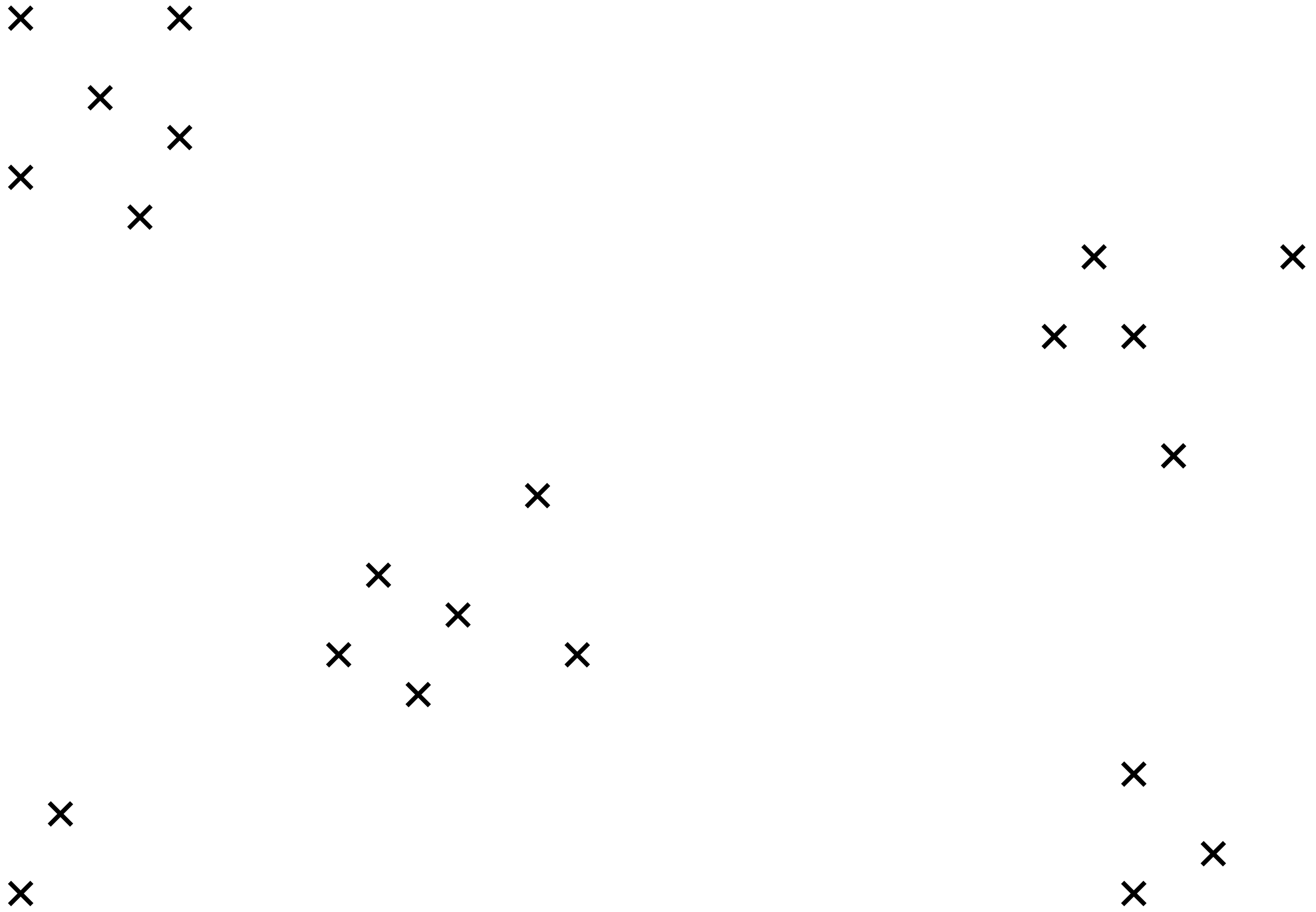
which runs in $O(n \log n)$ time

In fact, LPT is a $4/3$-approximation algorithm (using better analysis)

k-centers

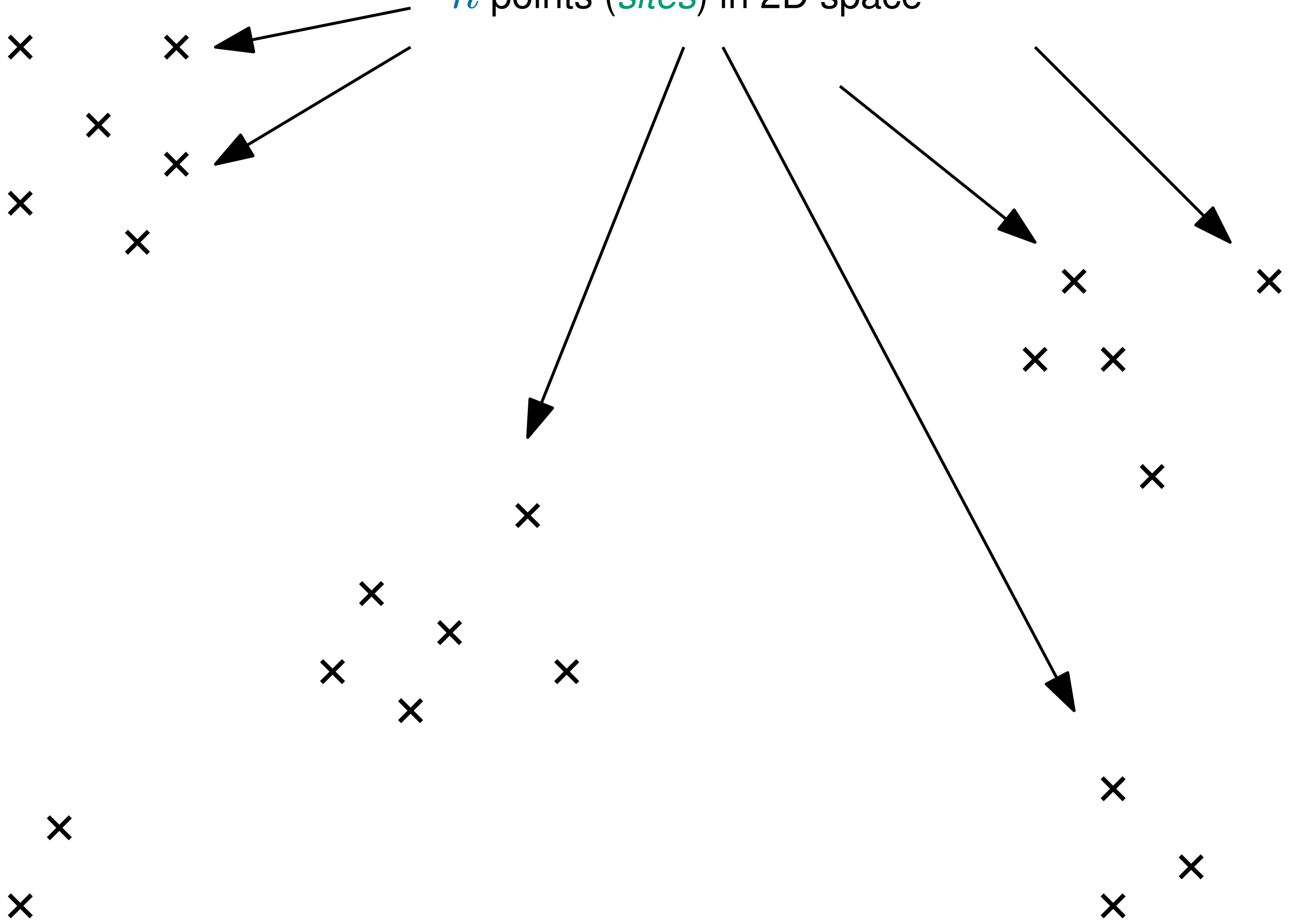


k-centers

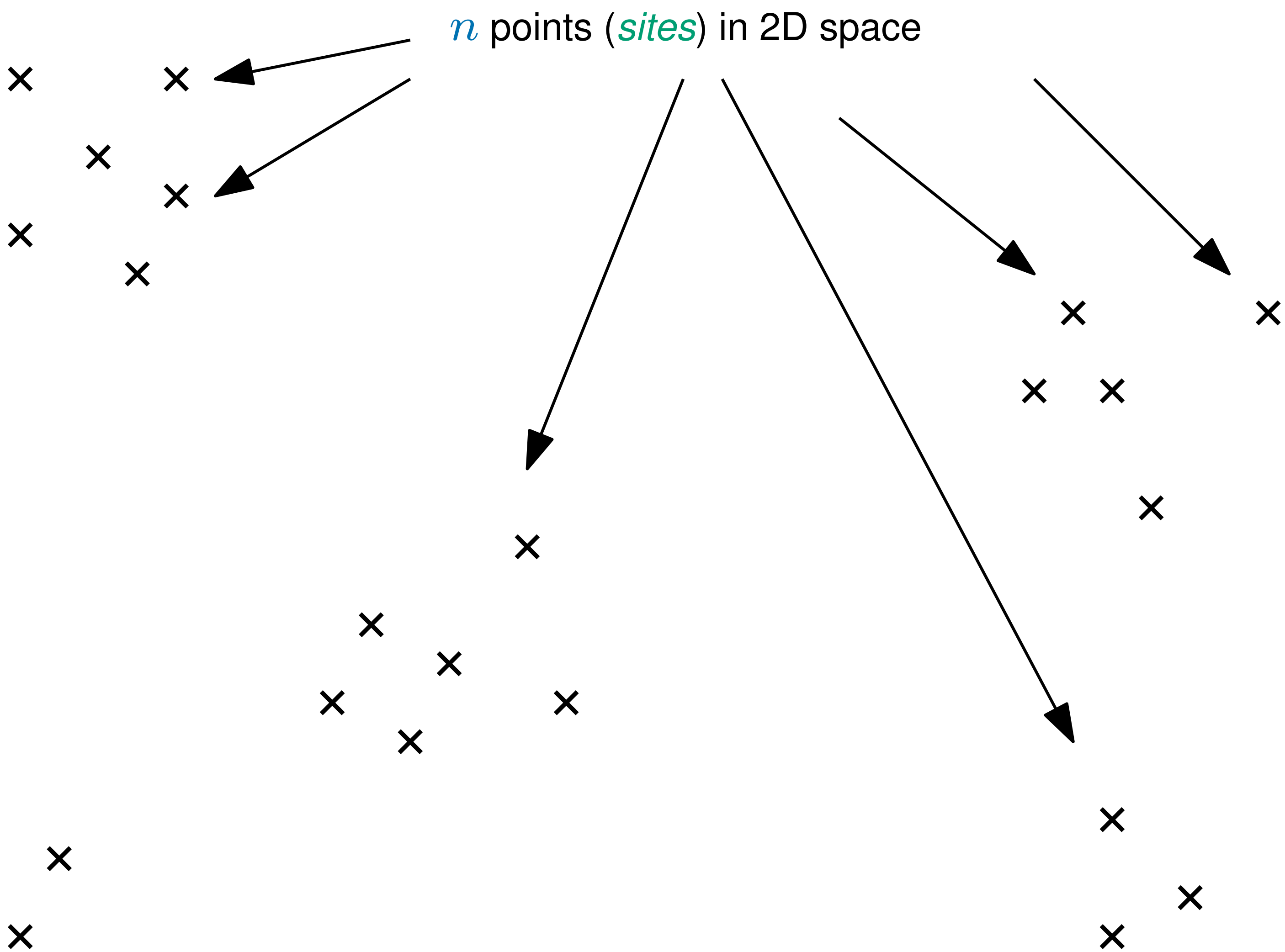


k-centers

n points (*sites*) in 2D space

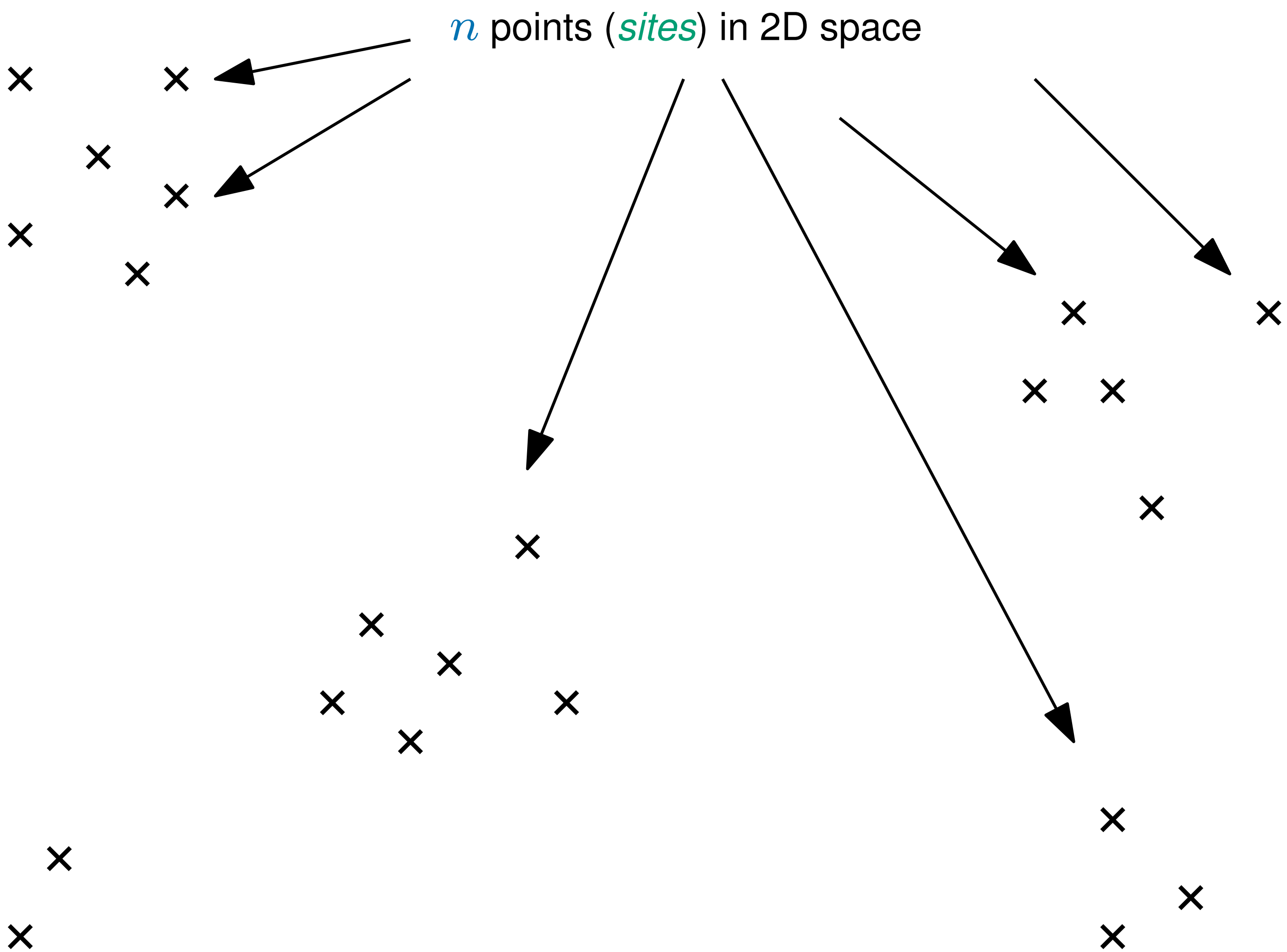


k-centers



The distance between points s_i, s_j is $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

k-centers

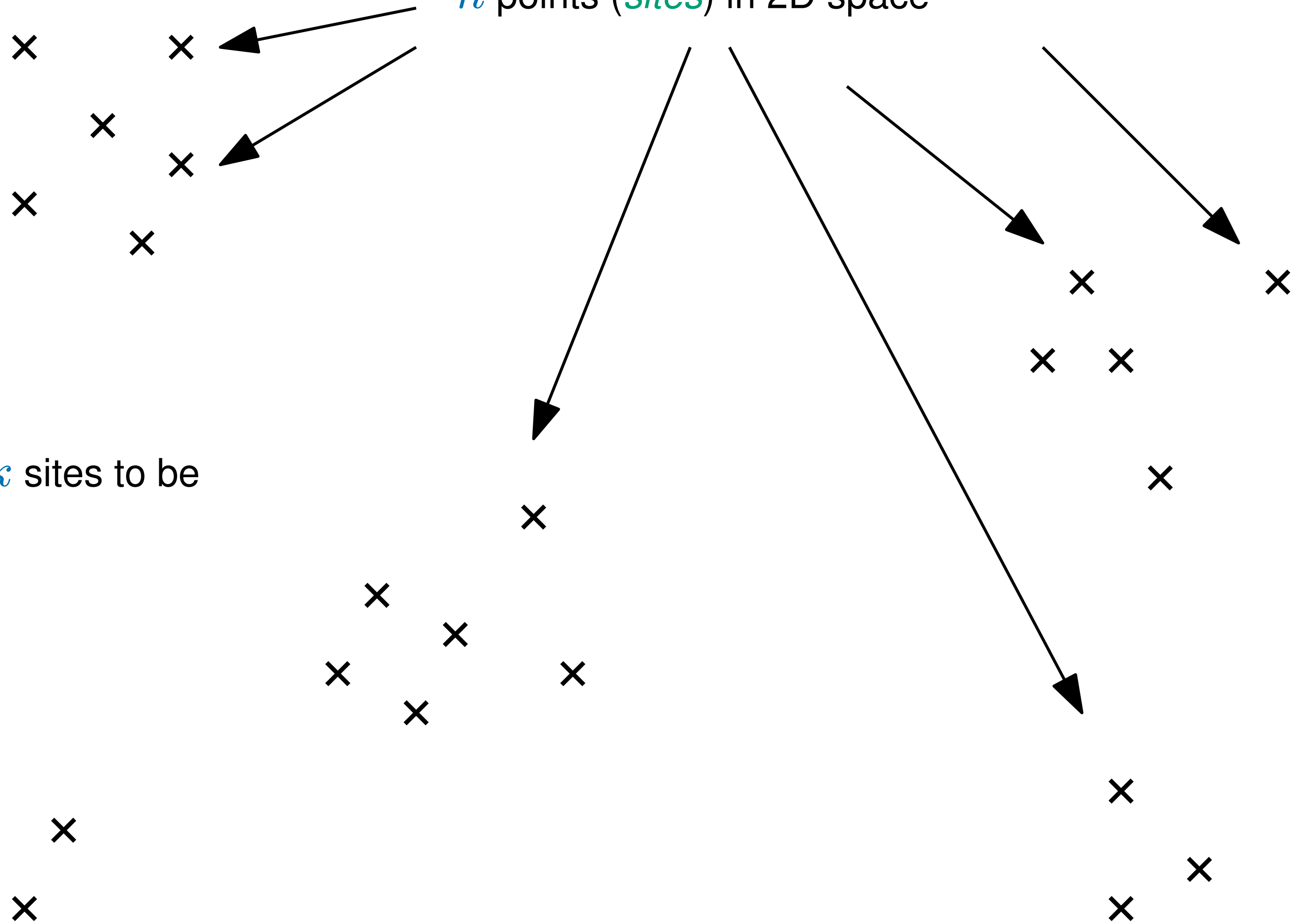


The distance between points s_i, s_j is $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

(i.e. ‘normal’ euclidean distance)

k-centers

n points (*sites*) in 2D space

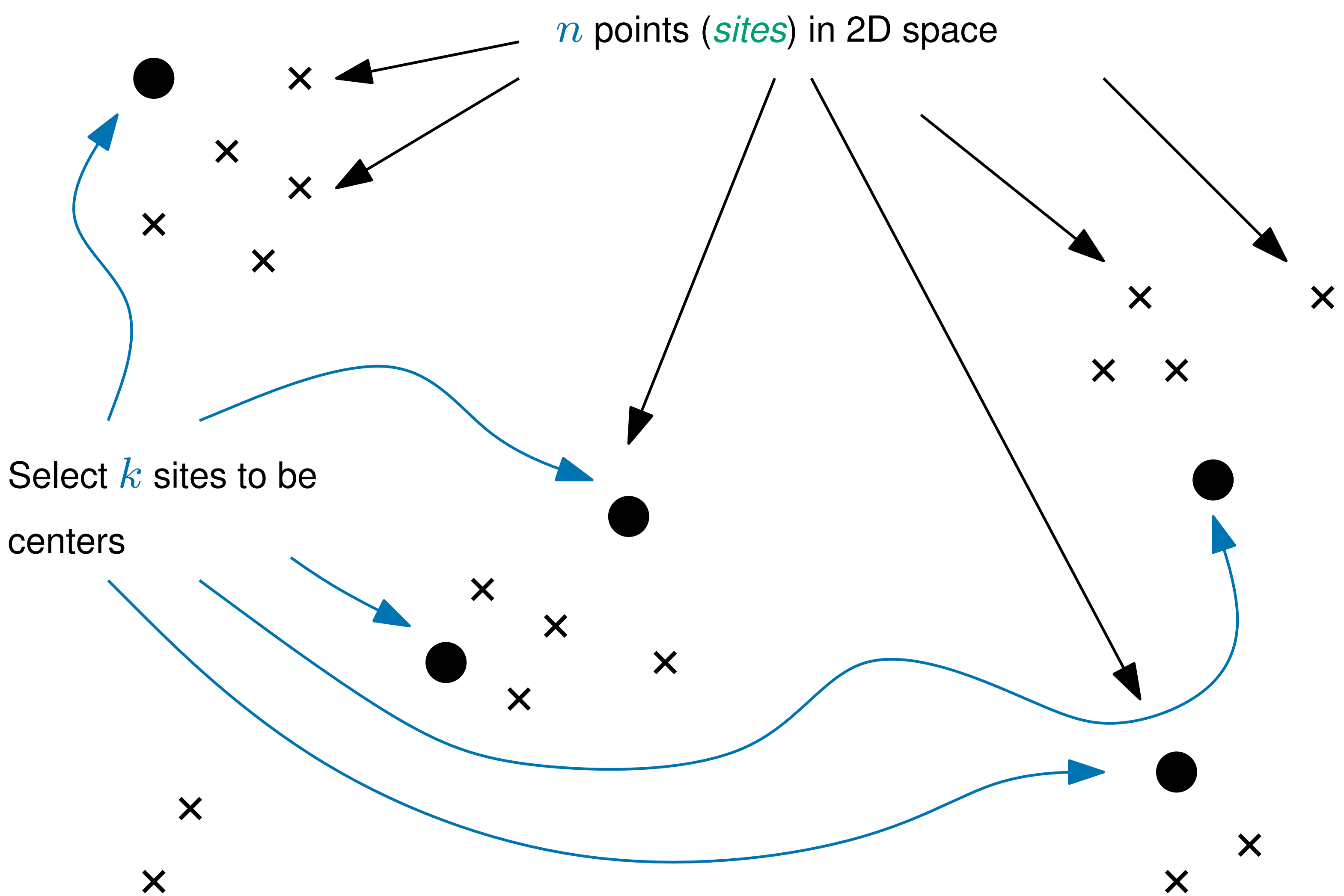


Select k sites to be centers

The distance between points s_i, s_j is $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

(i.e. 'normal' euclidean distance)

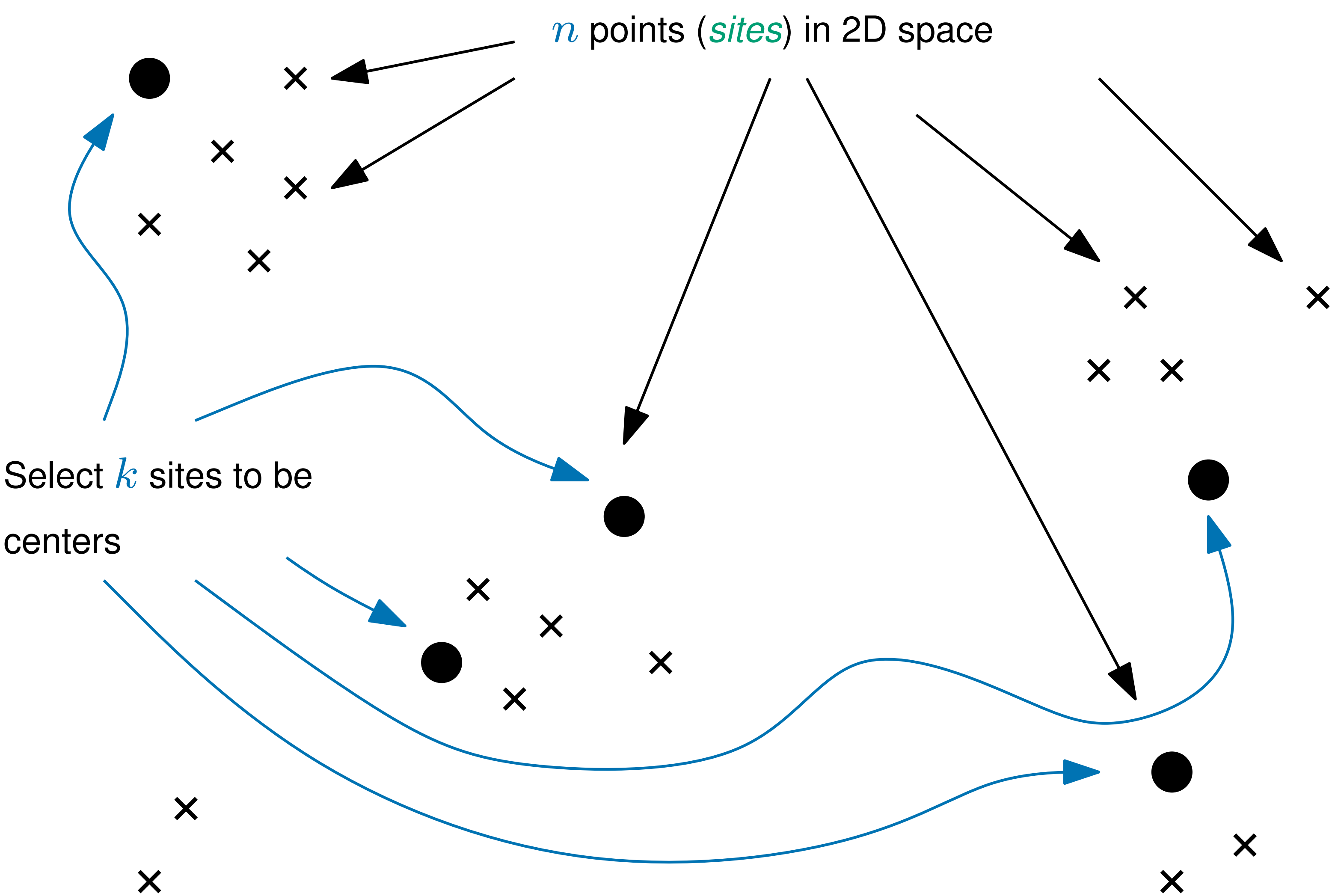
k-centers



The distance between points s_i, s_j is $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

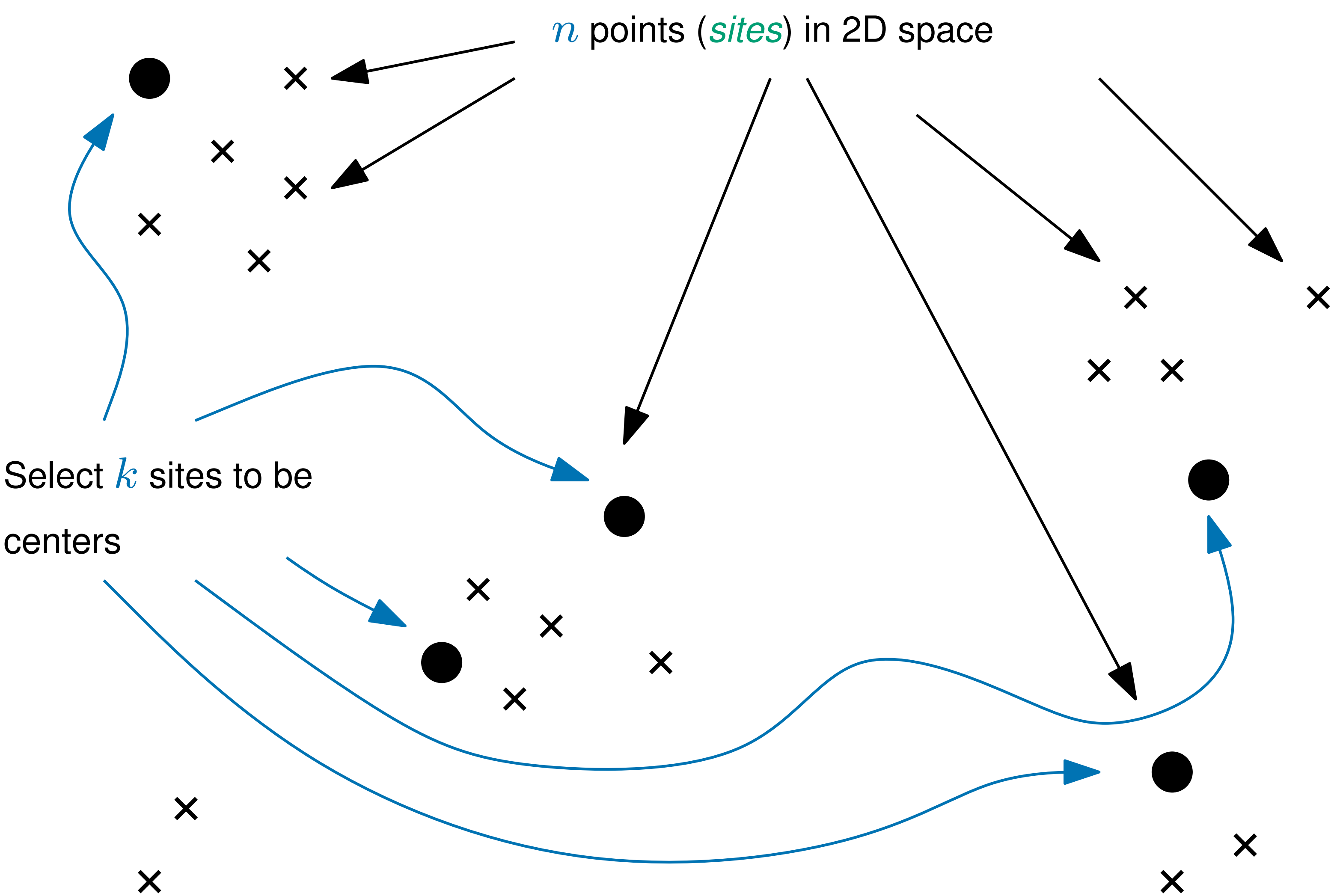
(i.e. 'normal' euclidean distance)

k-centers



The distance between points s_i, s_j is $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

k-centers

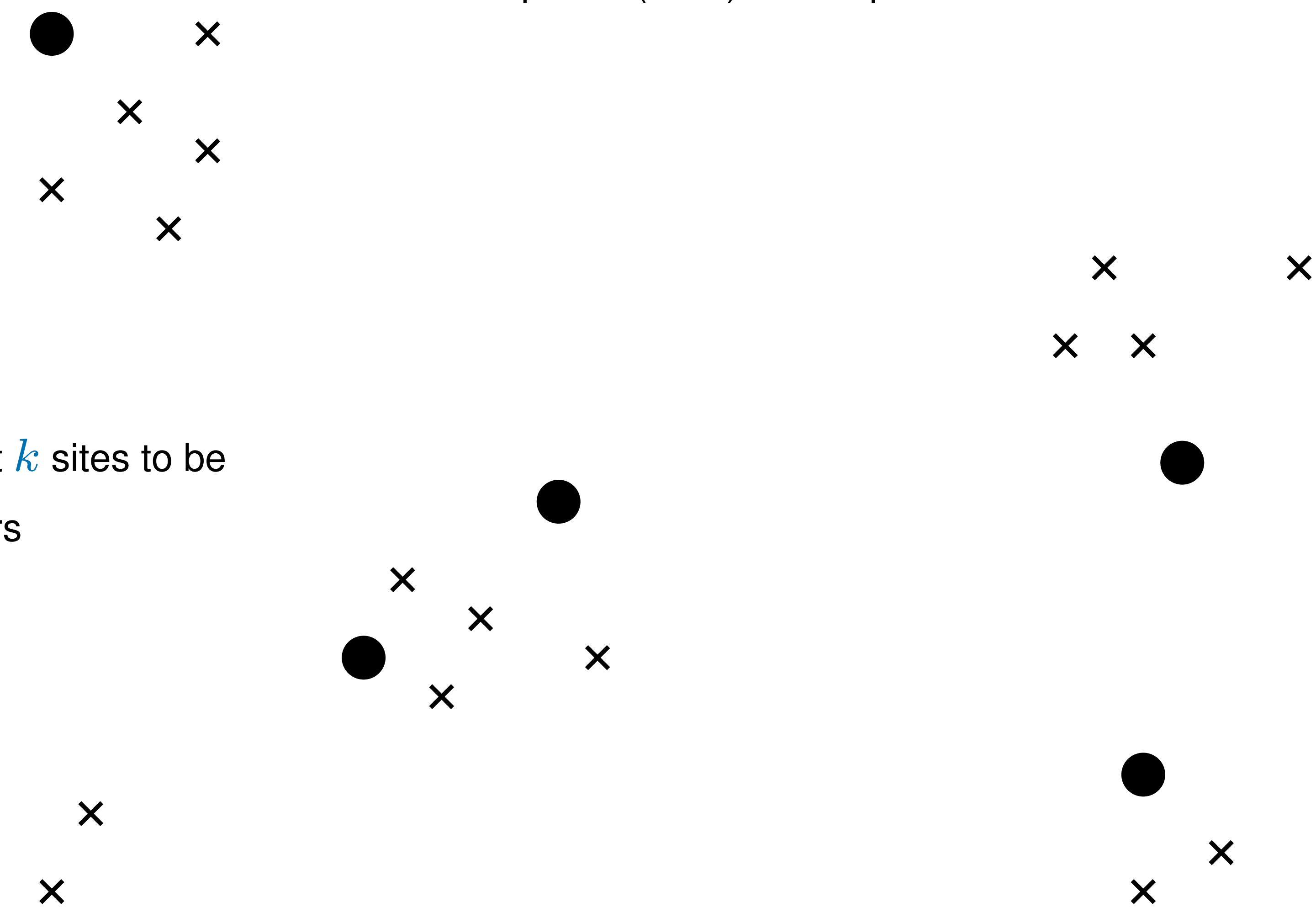


The distance between points s_i, s_j is $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

Goal Minimise the largest distance from any site to the closest center

k-centers

n points (*sites*) in 2D space



Select k sites to be centers

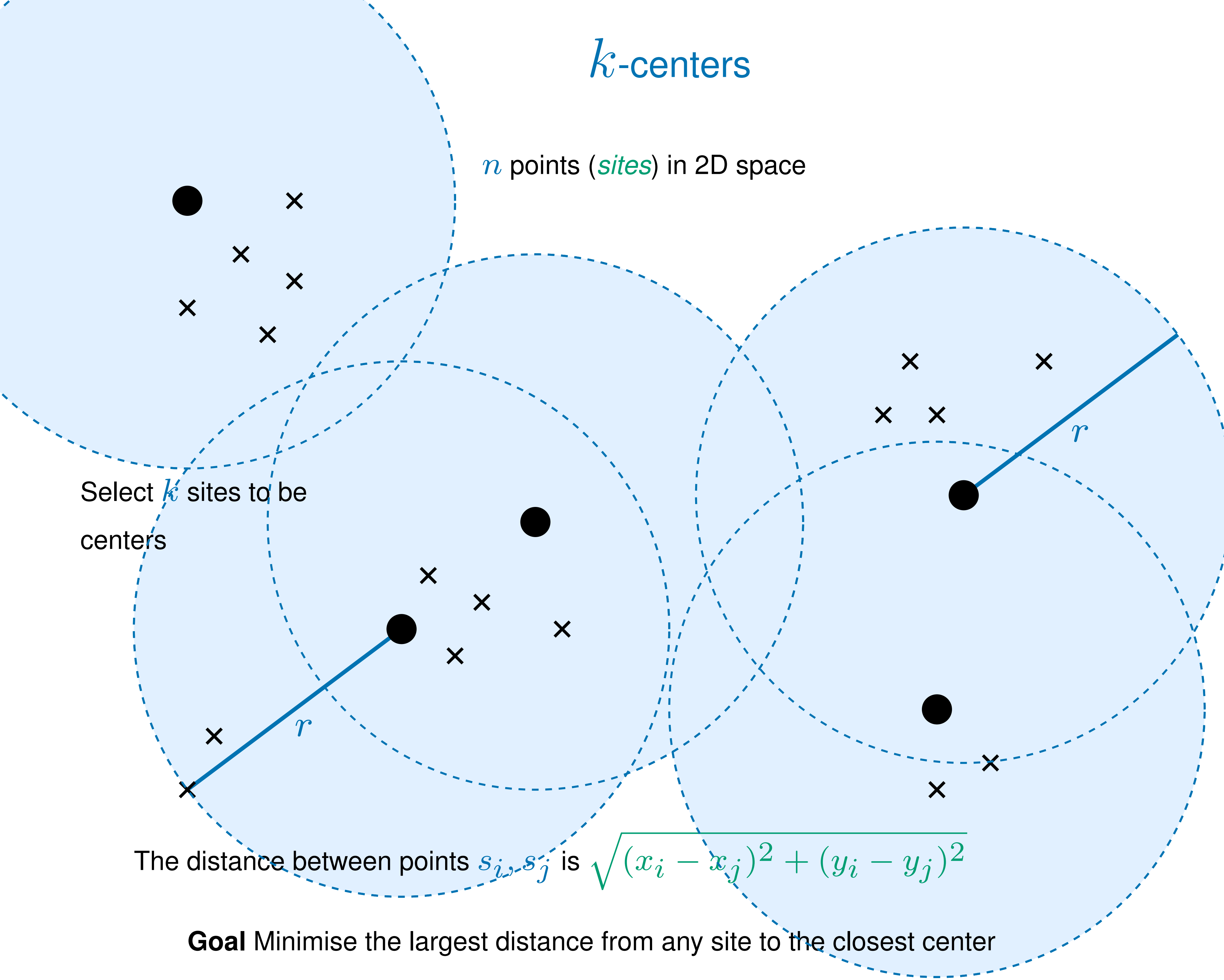
The distance between points s_i, s_j is $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

Goal Minimise the largest distance from any site to the closest center

k-centers

n points (*sites*) in 2D space

Select k sites to be centers

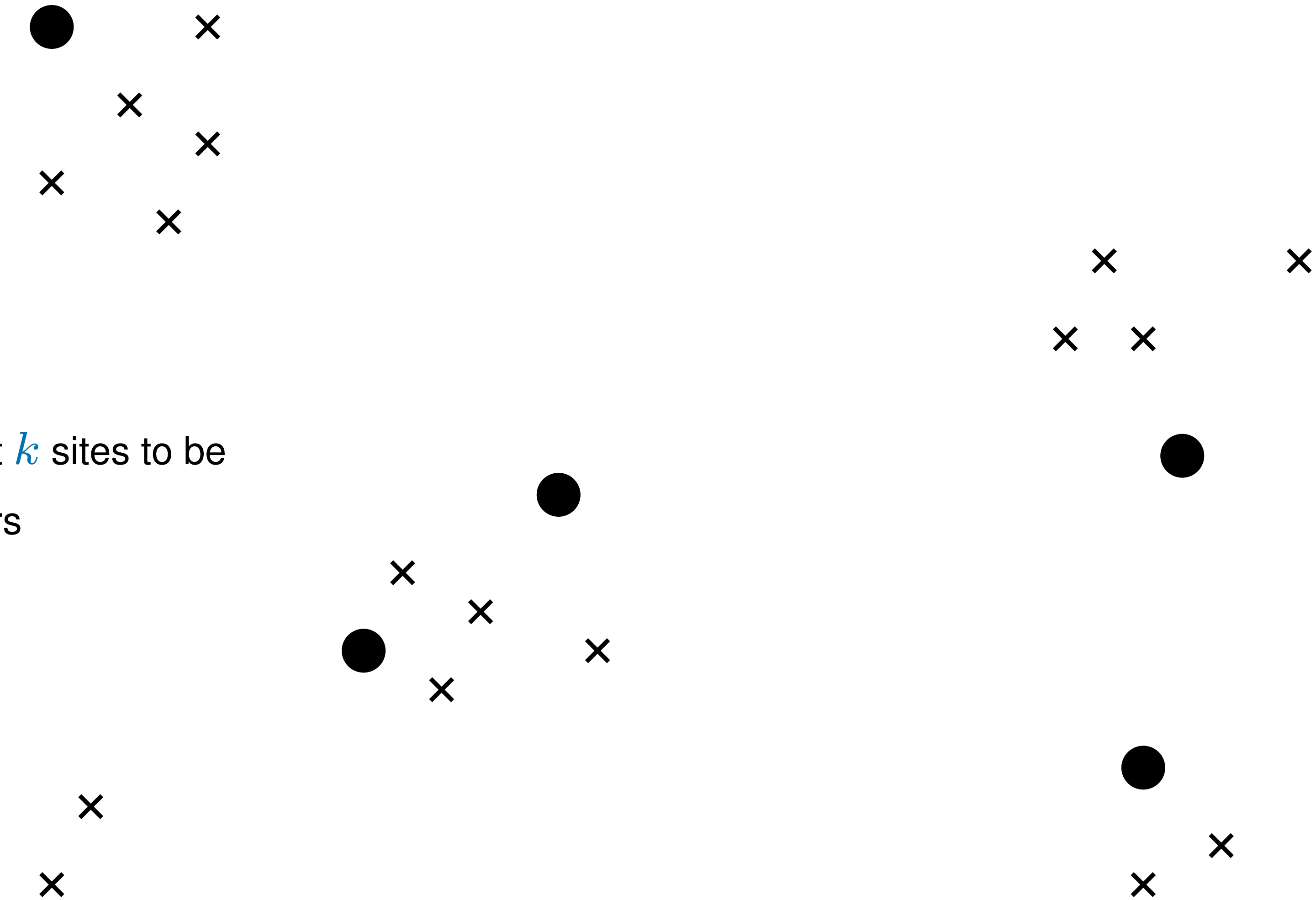


The distance between points s_i, s_j is $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

Goal Minimise the largest distance from any site to the closest center

k-centers

n points (*sites*) in 2D space



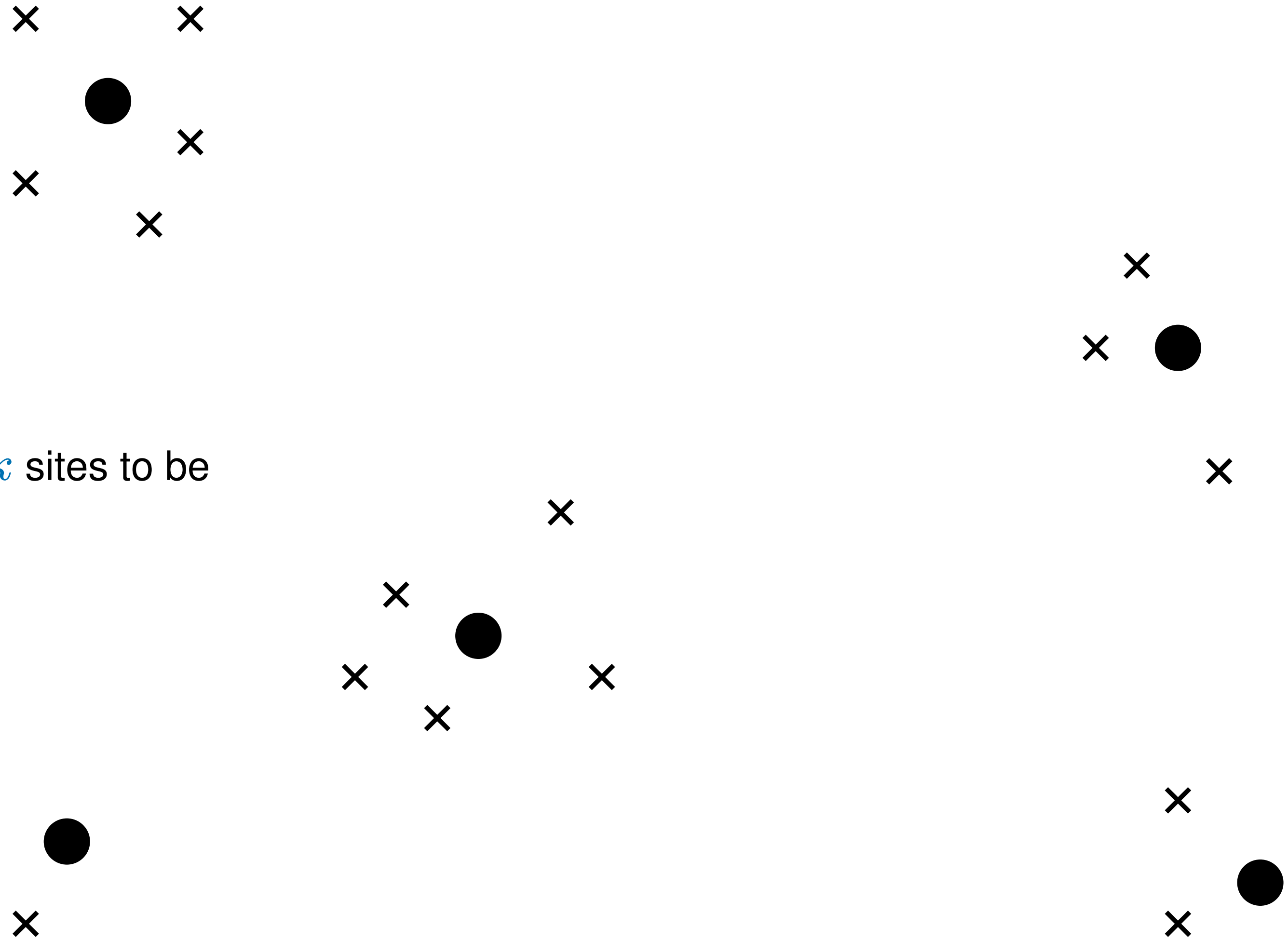
Select k sites to be centers

The distance between points s_i, s_j is $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

Goal Minimise the largest distance from any site to the closest center

k-centers

n points (*sites*) in 2D space



Select k sites to be centers

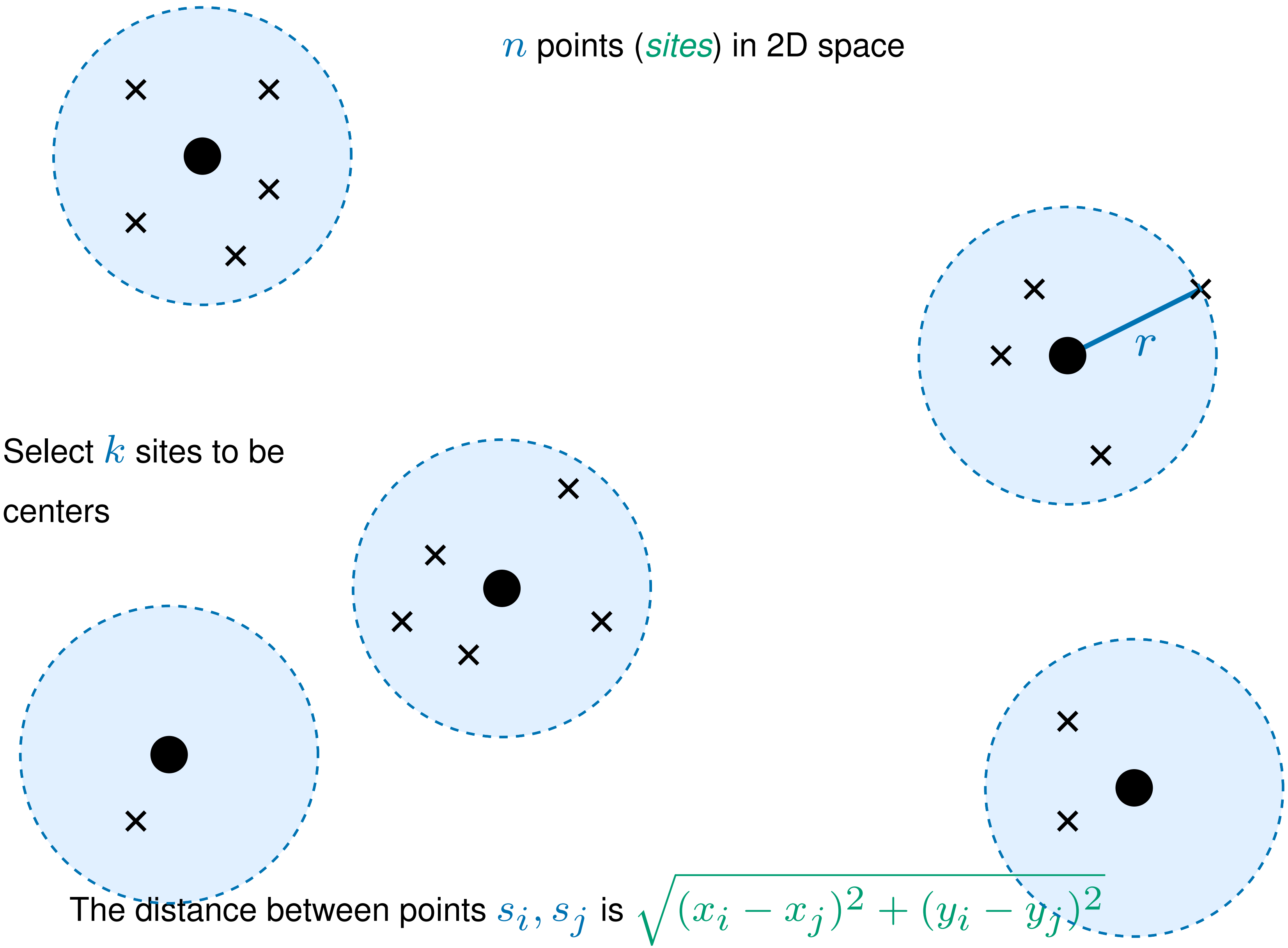
The distance between points s_i, s_j is $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

Goal Minimise the largest distance from any site to the closest center

k-centers

n points (*sites*) in 2D space

Select k sites to be centers

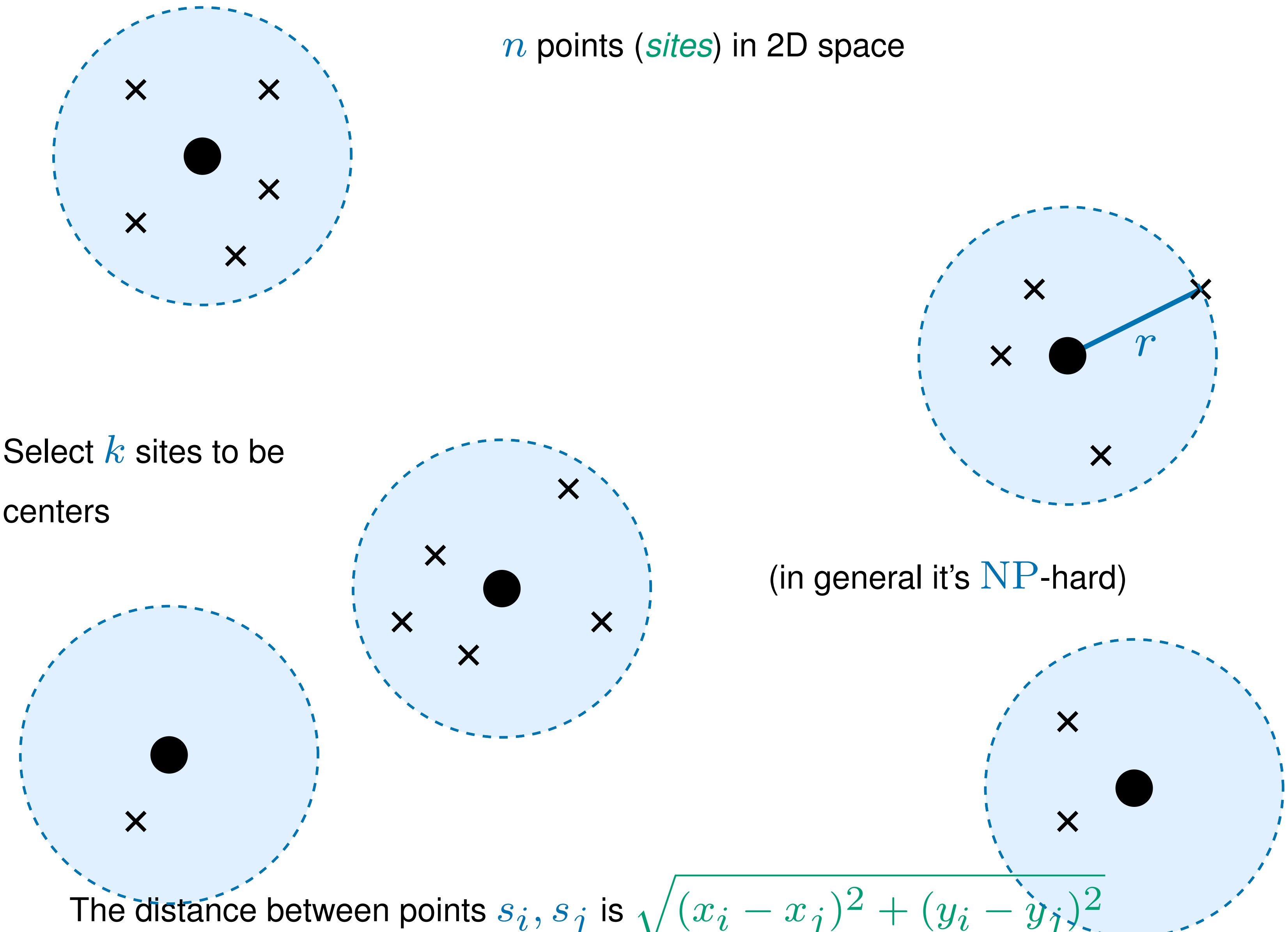


The distance between points s_i, s_j is $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

Goal Minimise the largest distance from any site to the closest center

k-centers

n points (*sites*) in 2D space



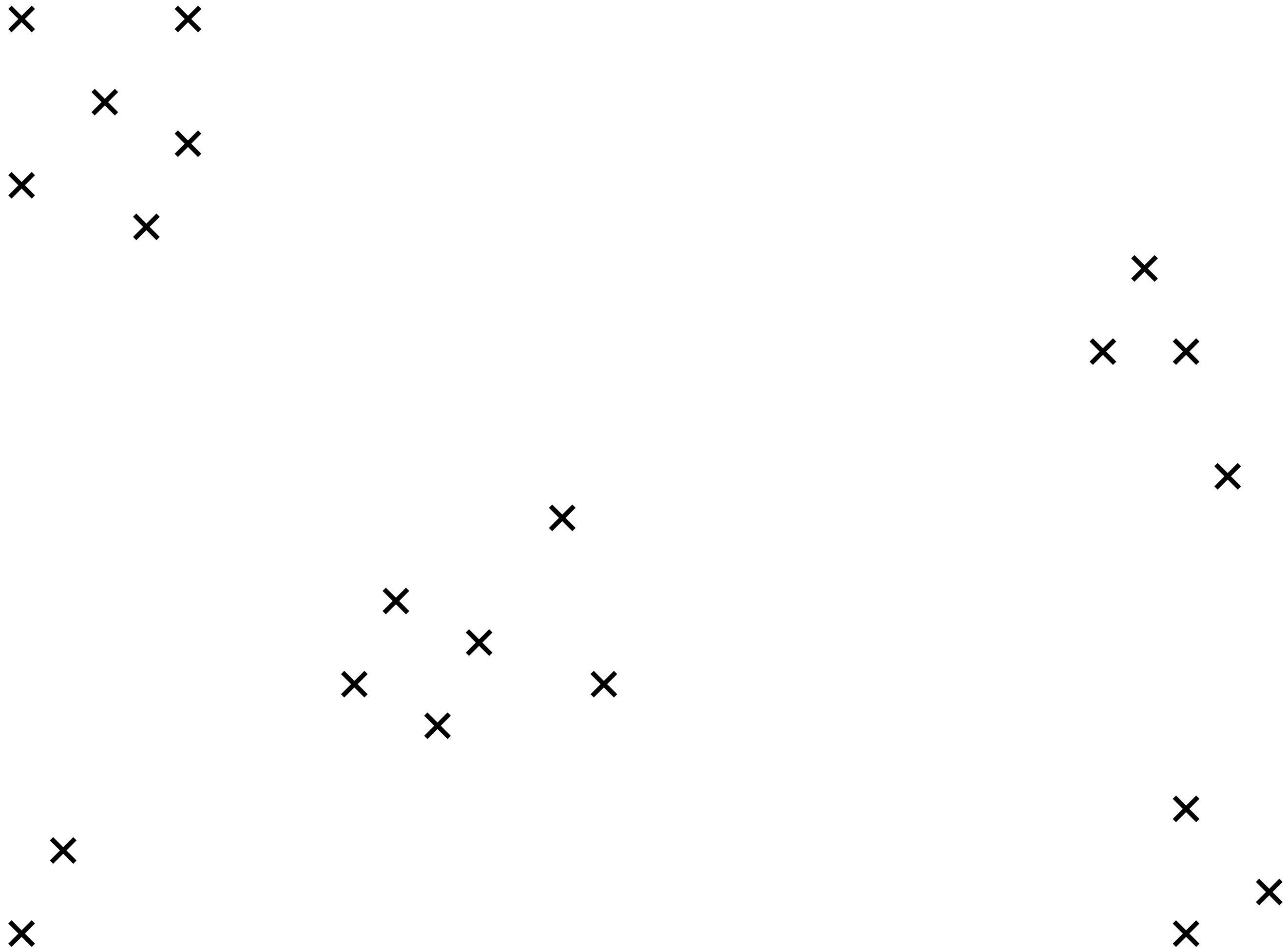
Select k sites to be centers

(in general it's NP-hard)

The distance between points s_i, s_j is $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

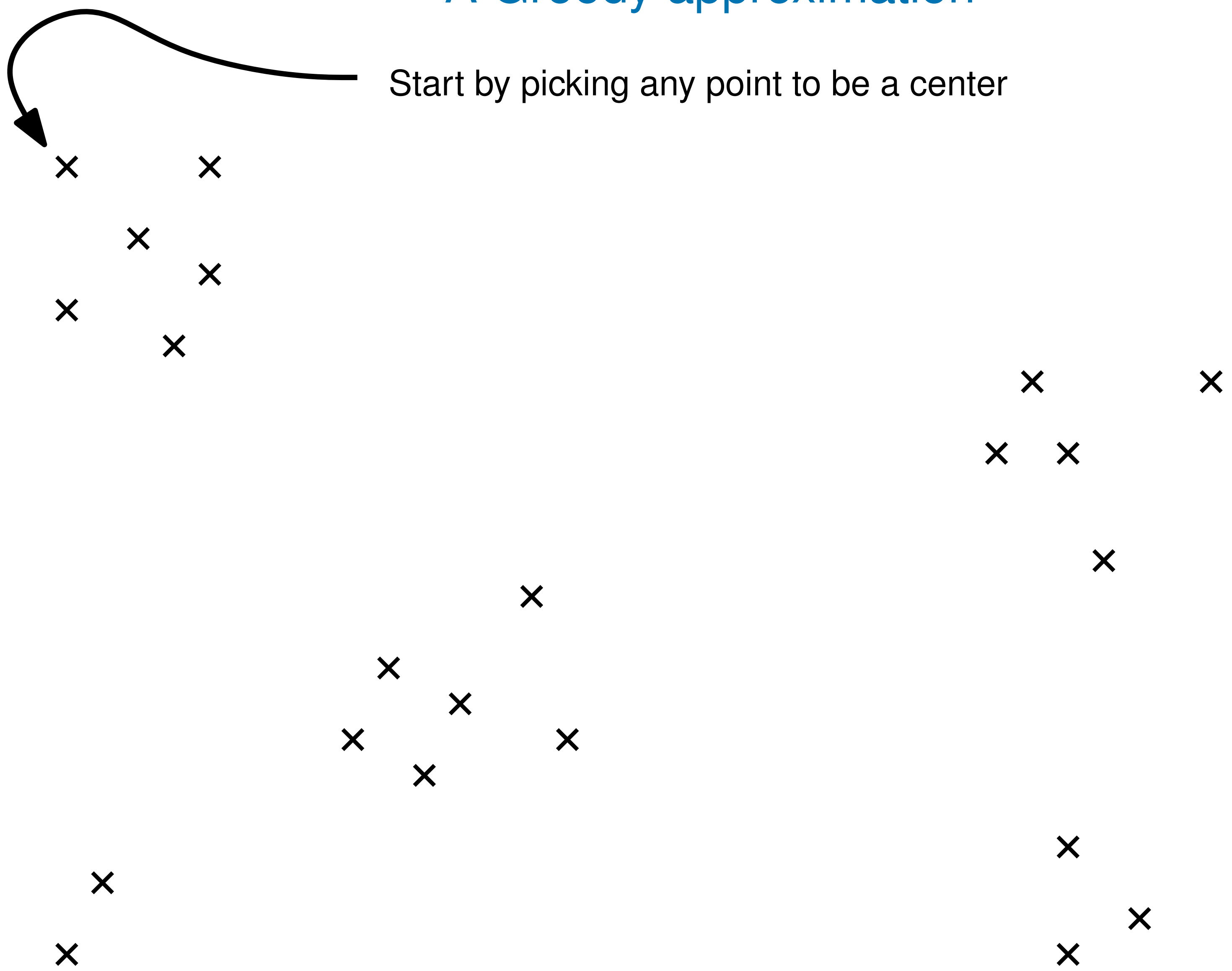
Goal Minimise the largest distance from any site to the closest center

A Greedy approximation



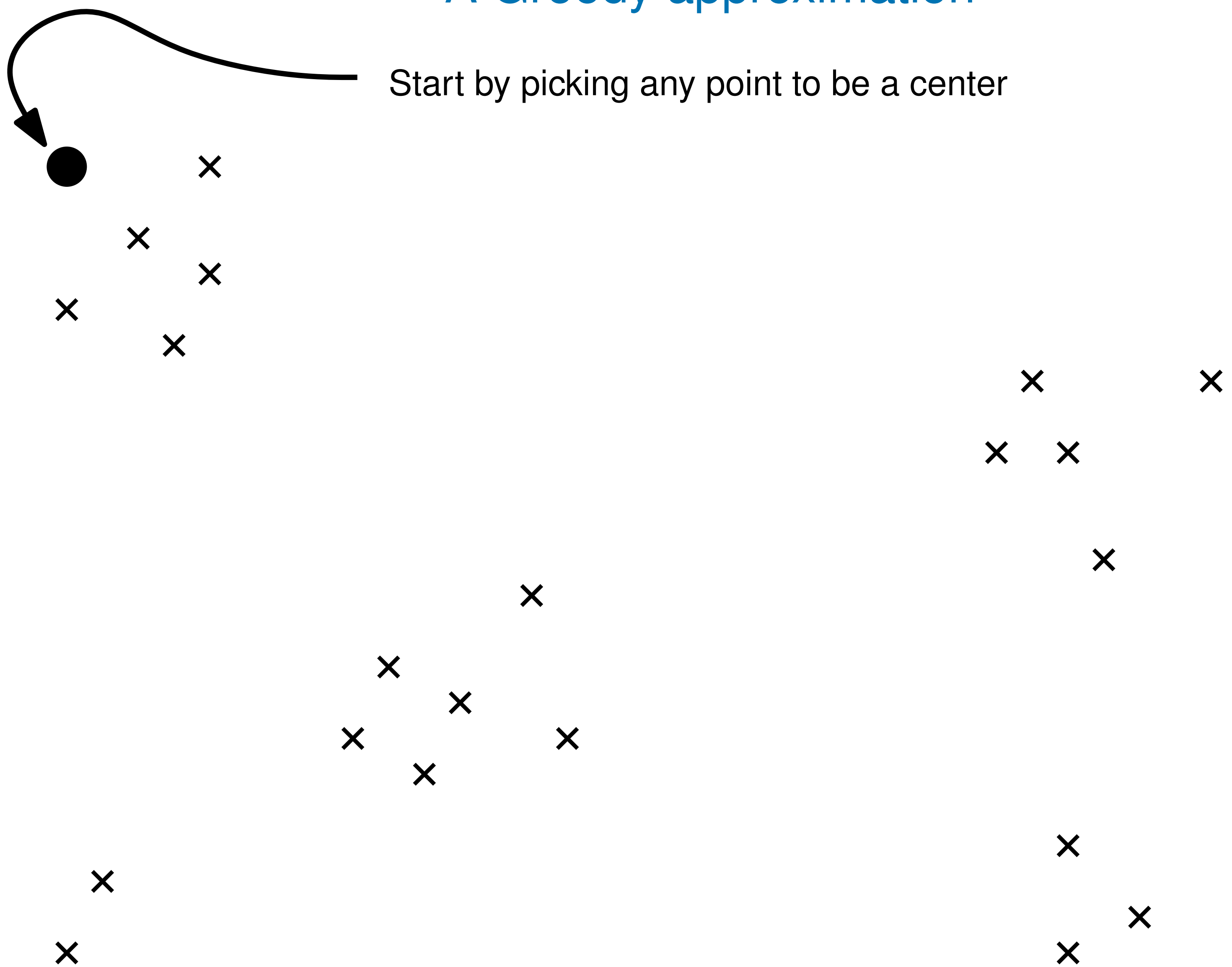
A Greedy approximation

Start by picking any point to be a center

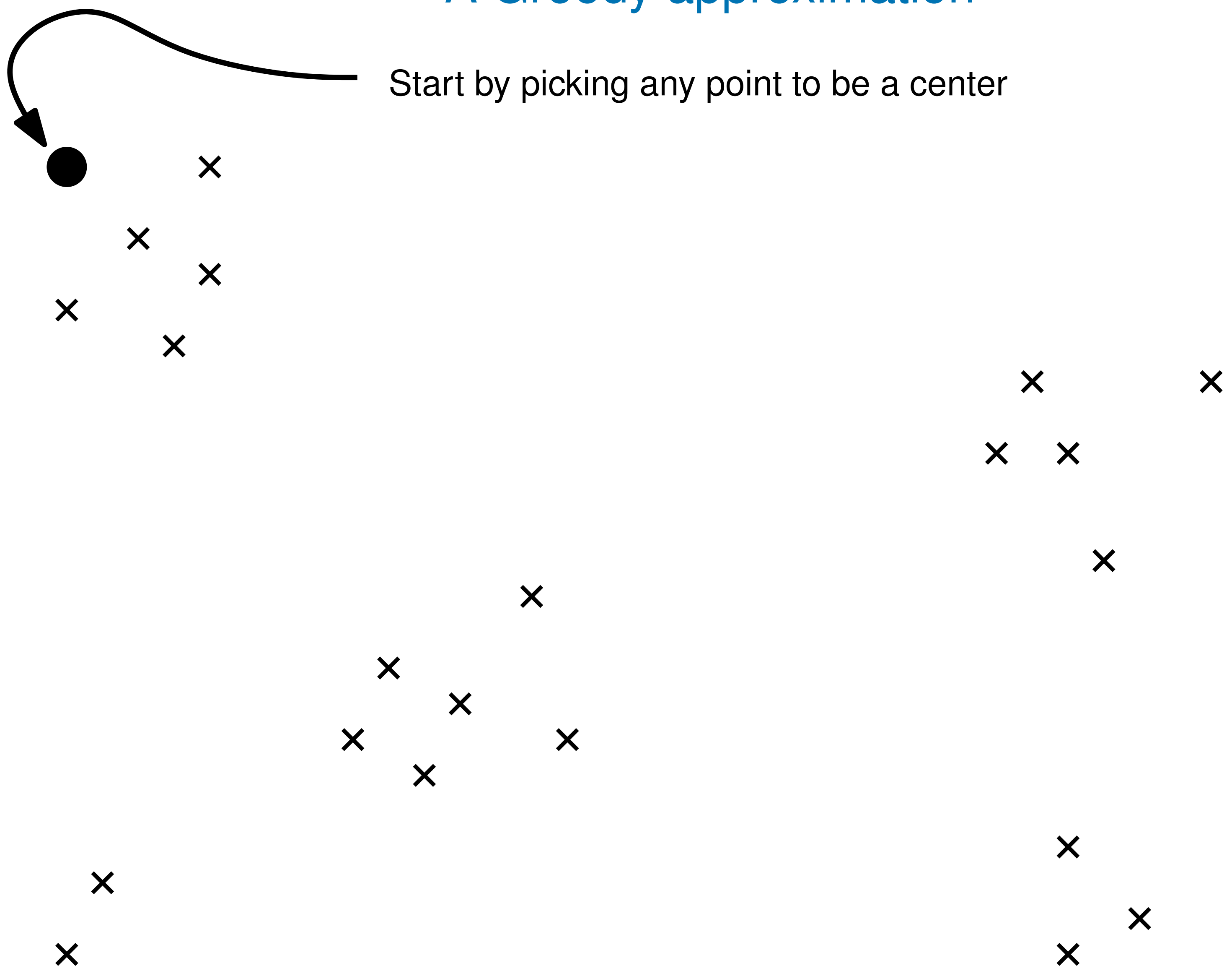


A Greedy approximation

Start by picking any point to be a center



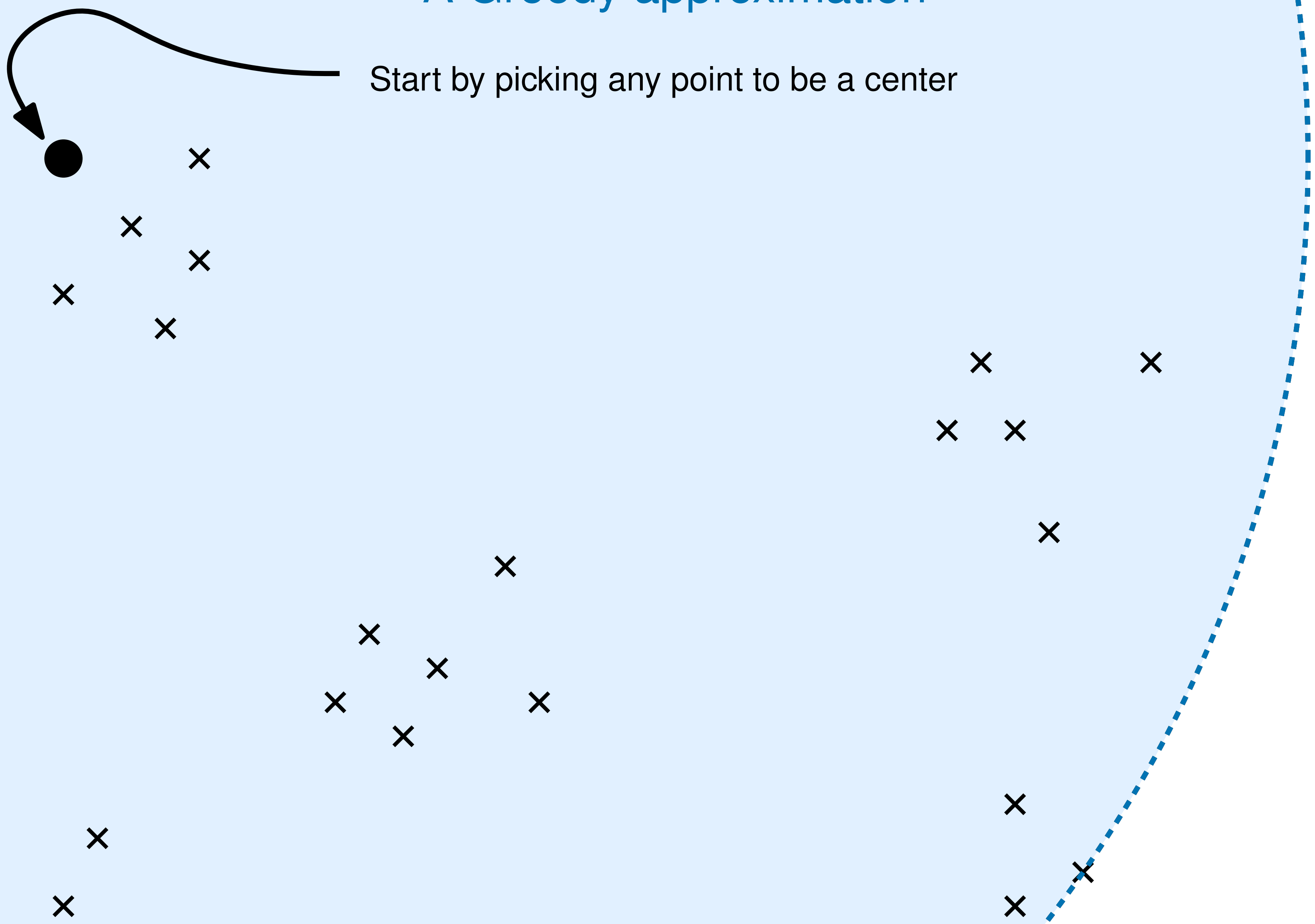
A Greedy approximation



Repeatedly pick the site which is furthest from any existing center

A Greedy approximation

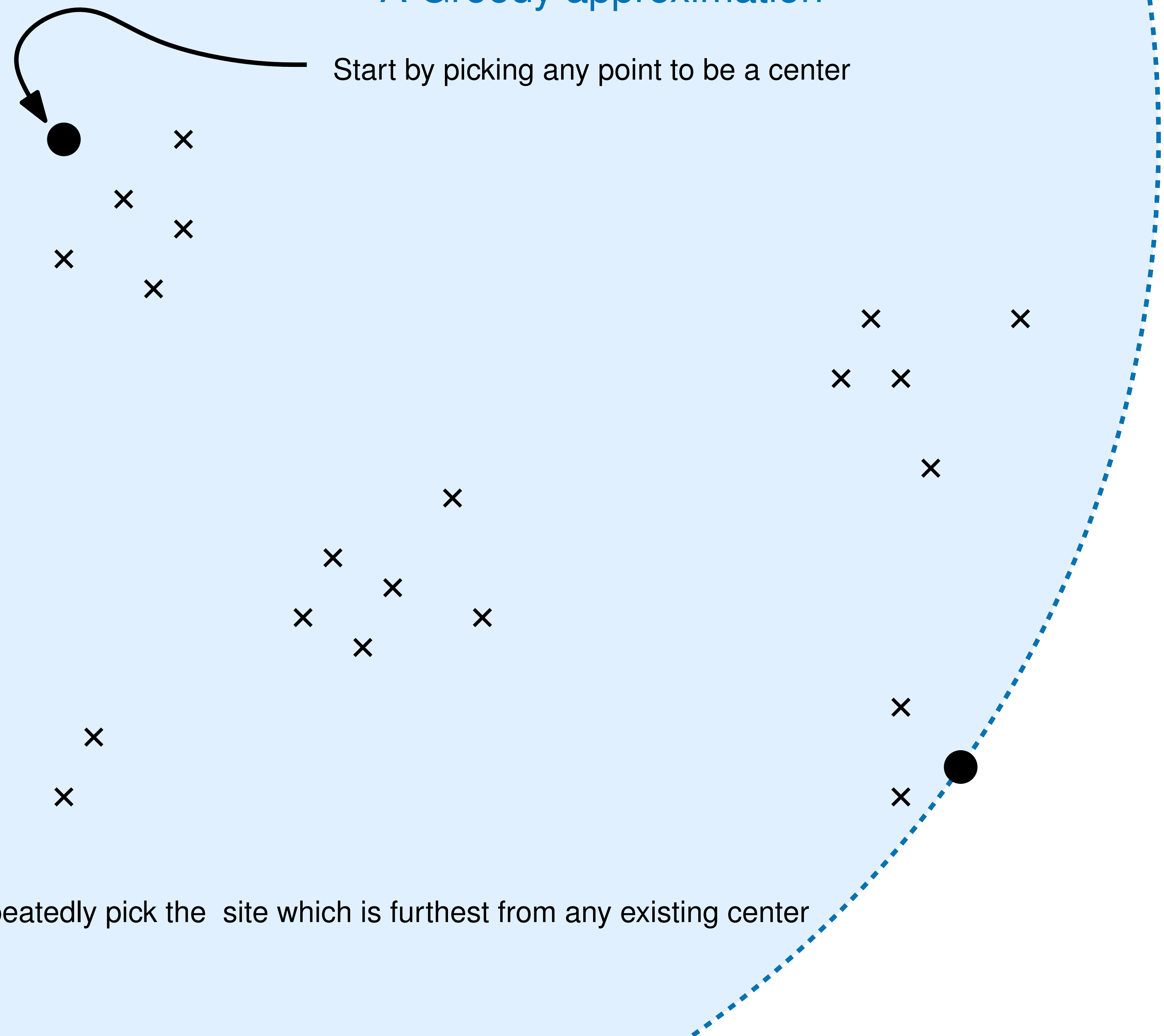
Start by picking any point to be a center



Repeatedly pick the site which is furthest from any existing center

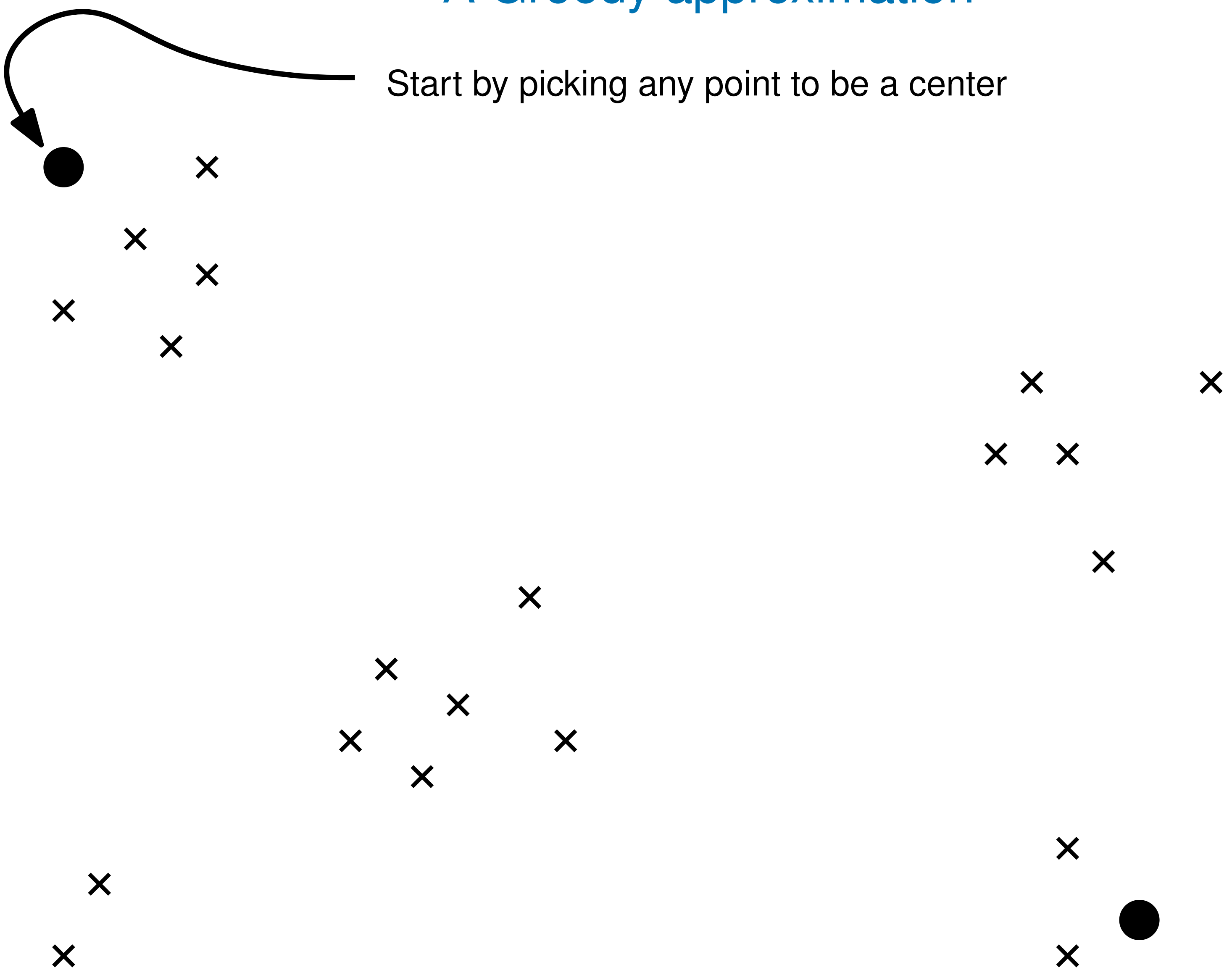
A Greedy approximation

Start by picking any point to be a center



Repeatedly pick the site which is furthest from any existing center

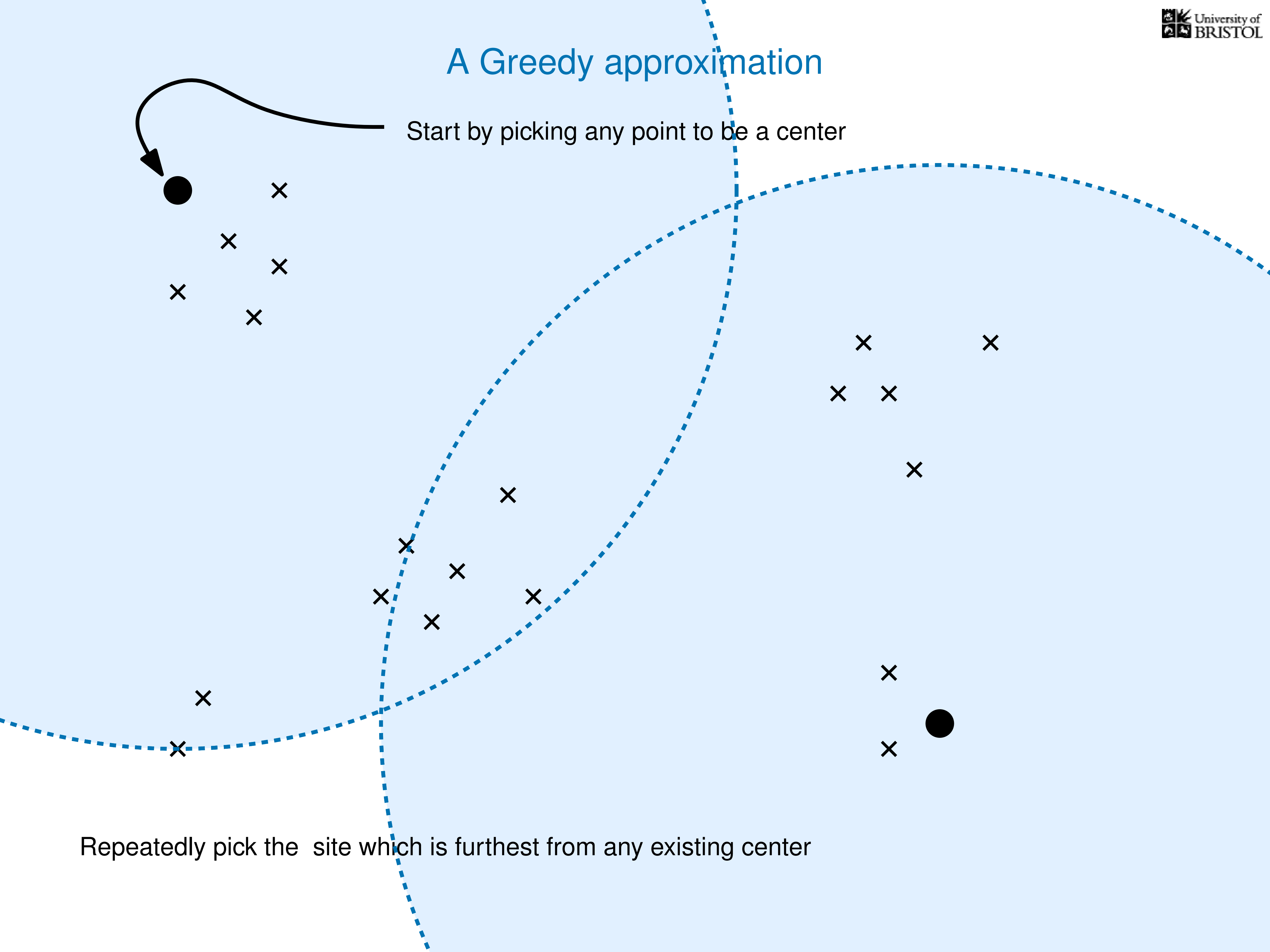
A Greedy approximation



Repeatedly pick the site which is furthest from any existing center

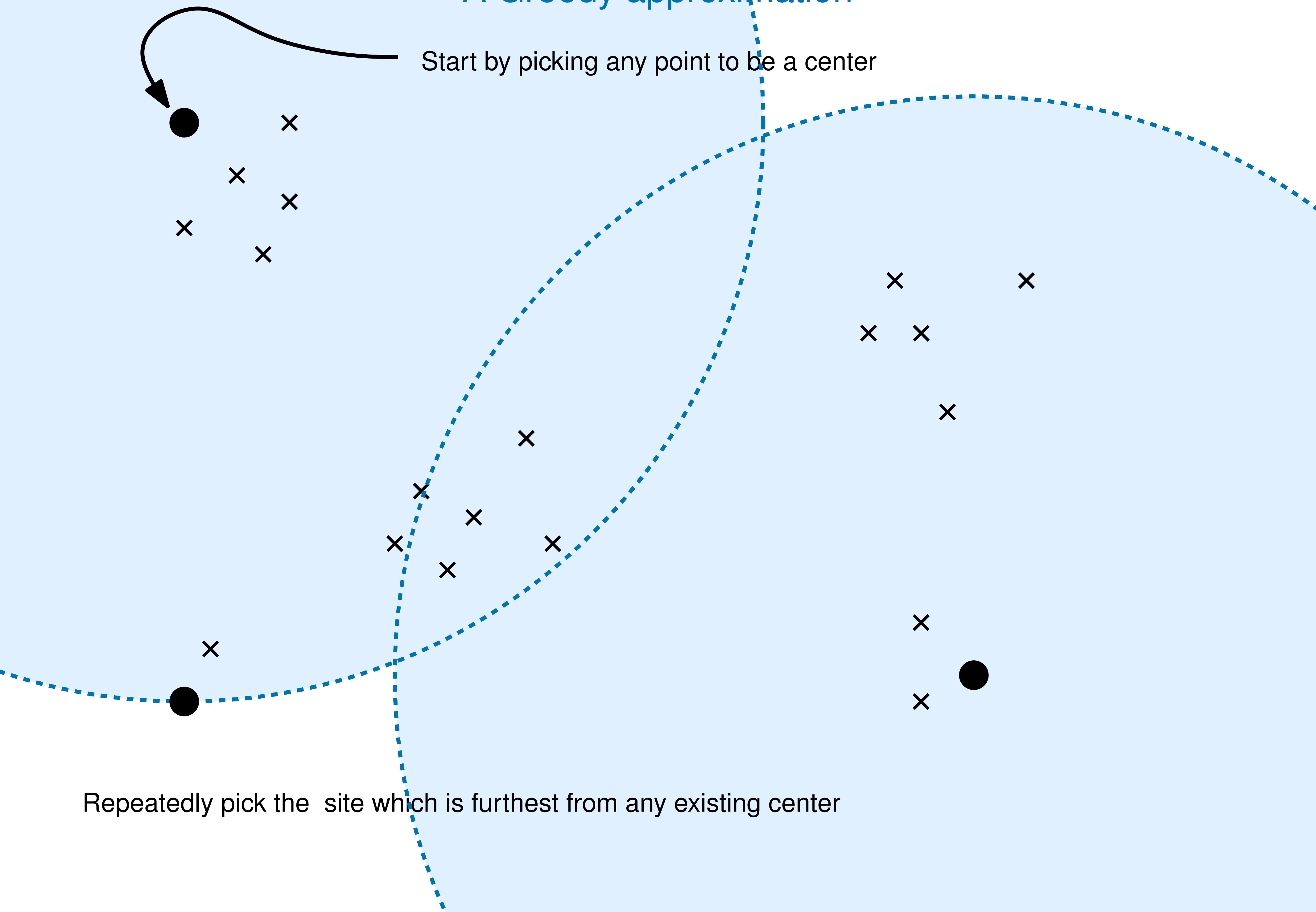
A Greedy approximation

Start by picking any point to be a center



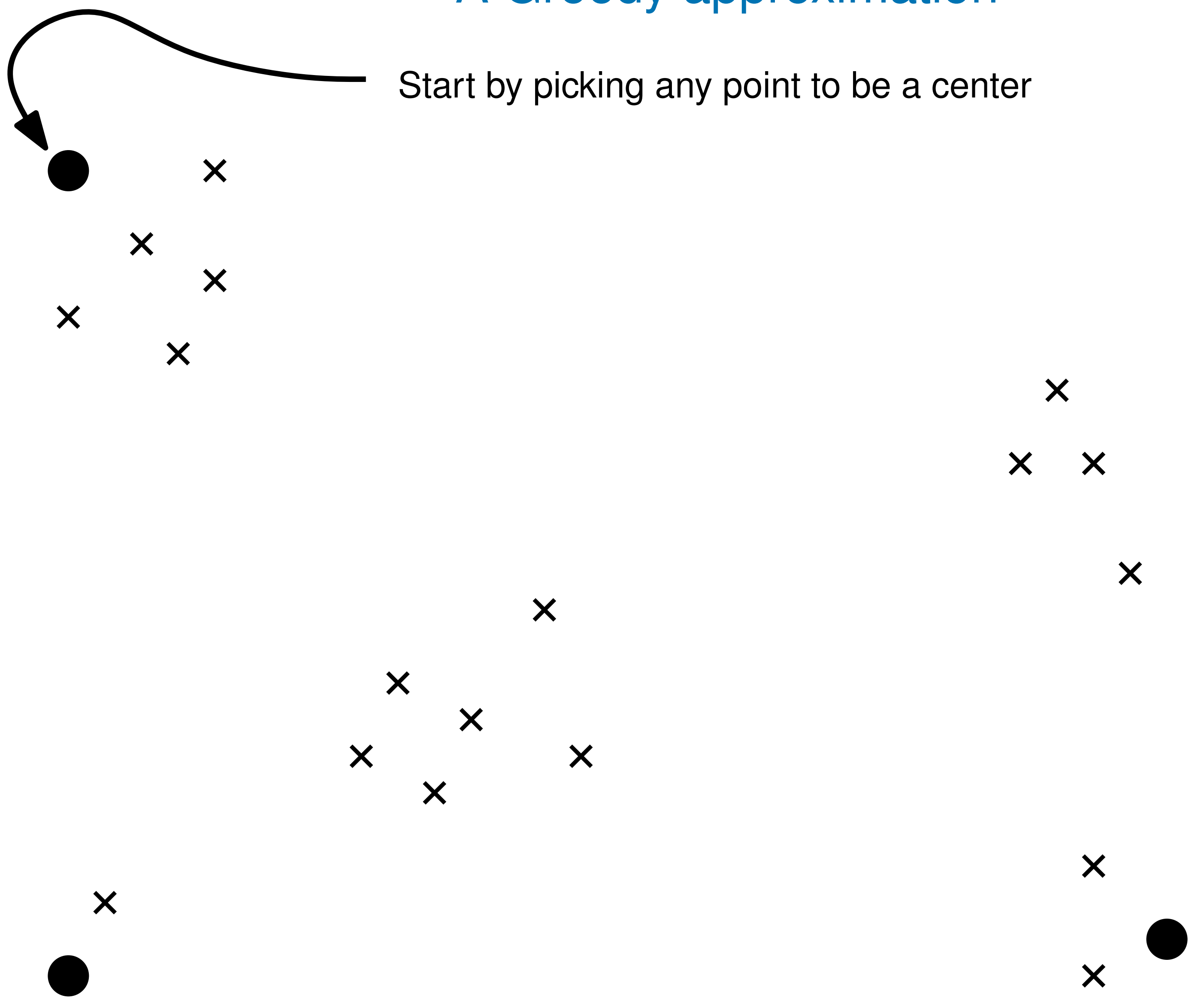
Repeatedly pick the site which is furthest from any existing center

A Greedy approximation



A Greedy approximation

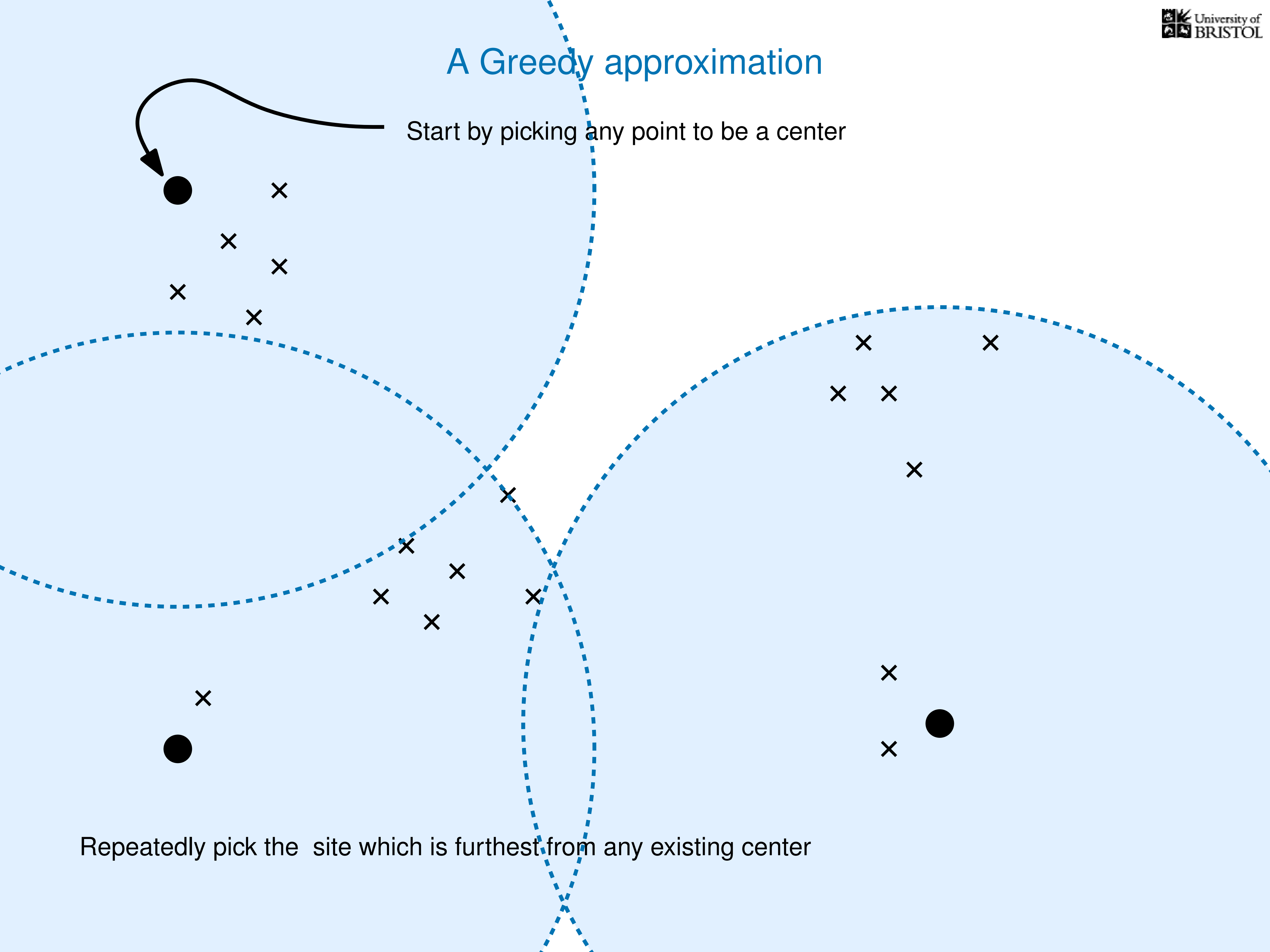
Start by picking any point to be a center



Repeatedly pick the site which is furthest from any existing center

A Greedy approximation

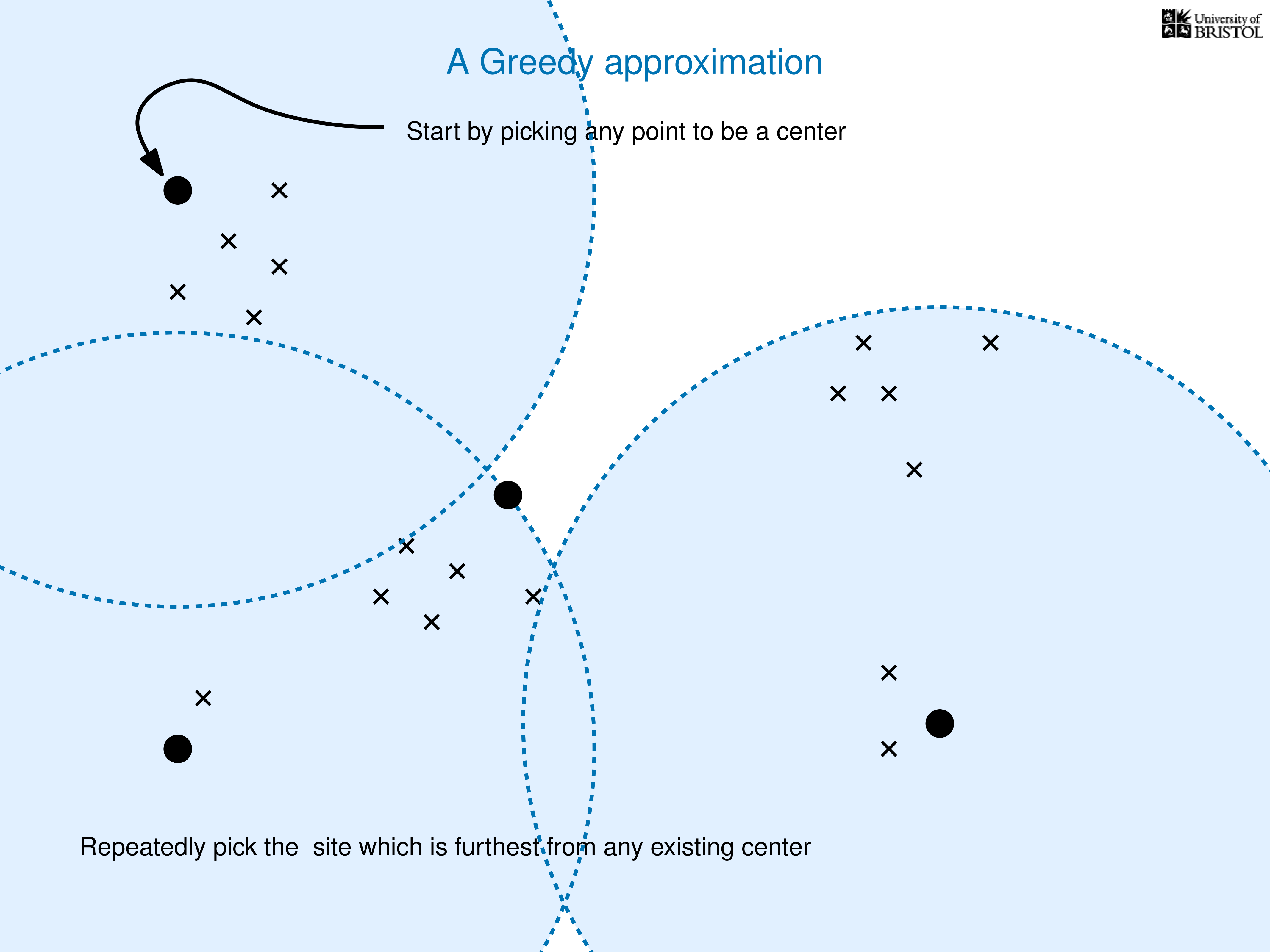
Start by picking any point to be a center



Repeatedly pick the site which is furthest from any existing center

A Greedy approximation

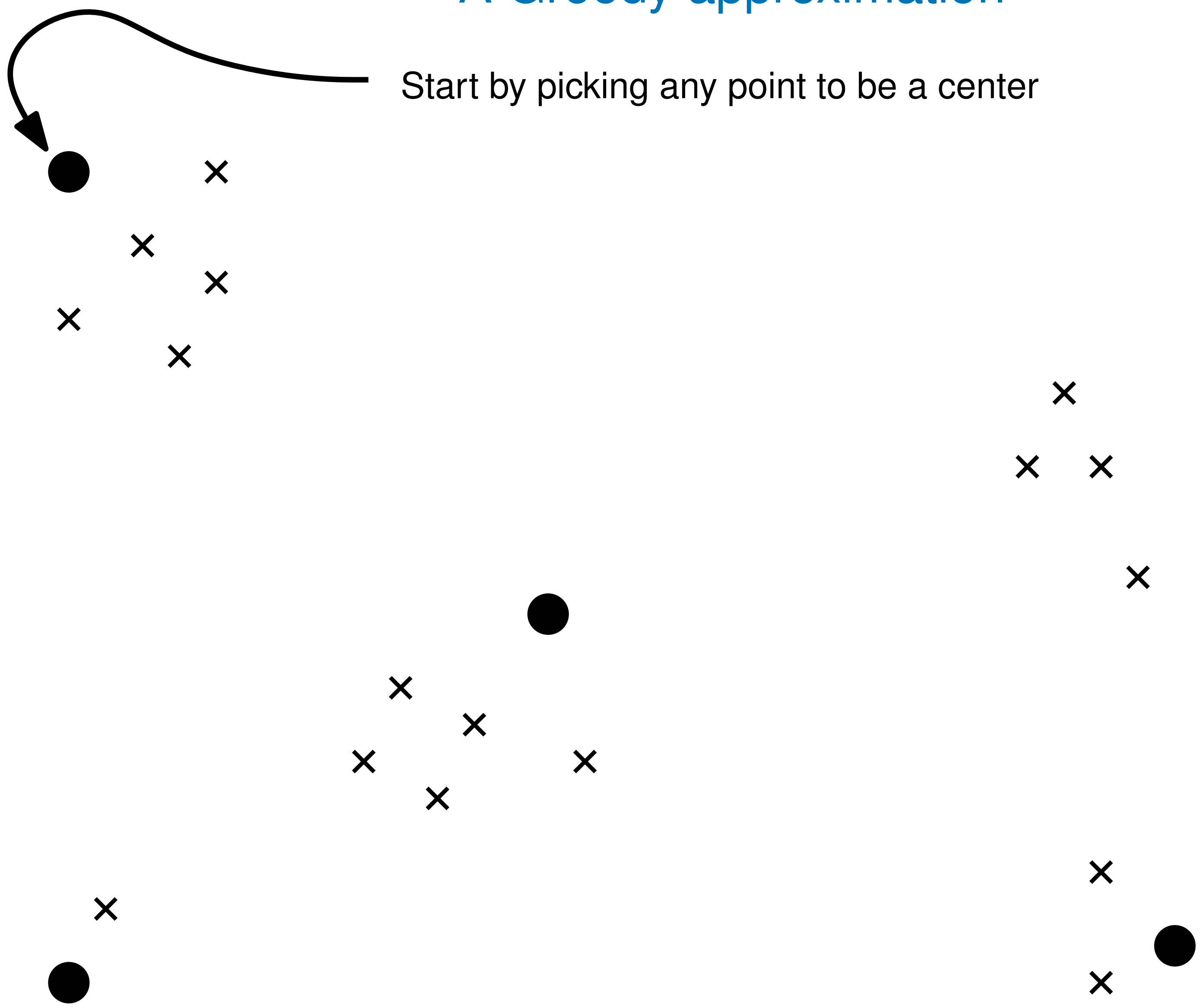
Start by picking any point to be a center



Repeatedly pick the site which is furthest from any existing center

A Greedy approximation

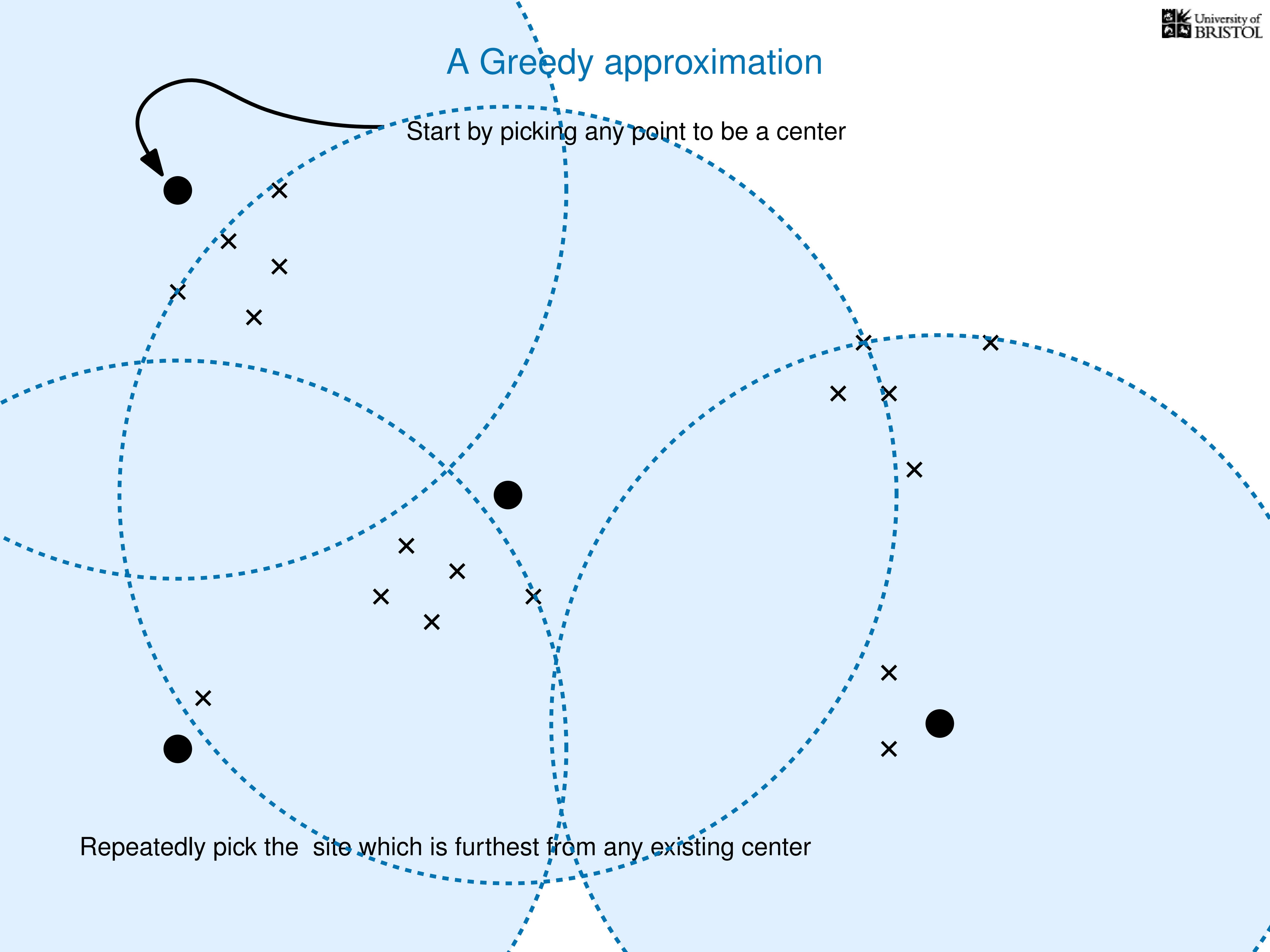
Start by picking any point to be a center



Repeatedly pick the site which is furthest from any existing center

A Greedy approximation

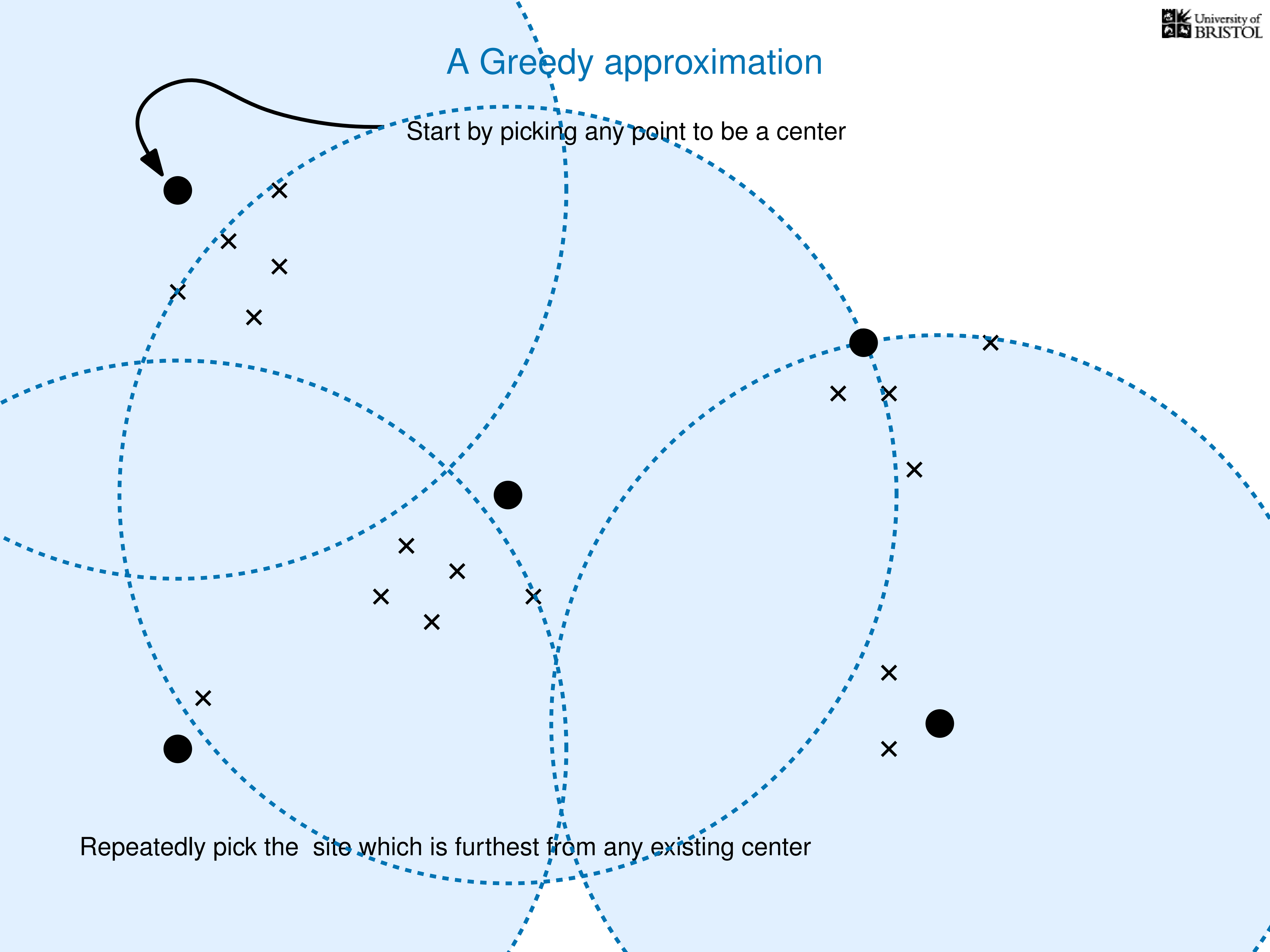
Start by picking any point to be a center



Repeatedly pick the site which is furthest from any existing center

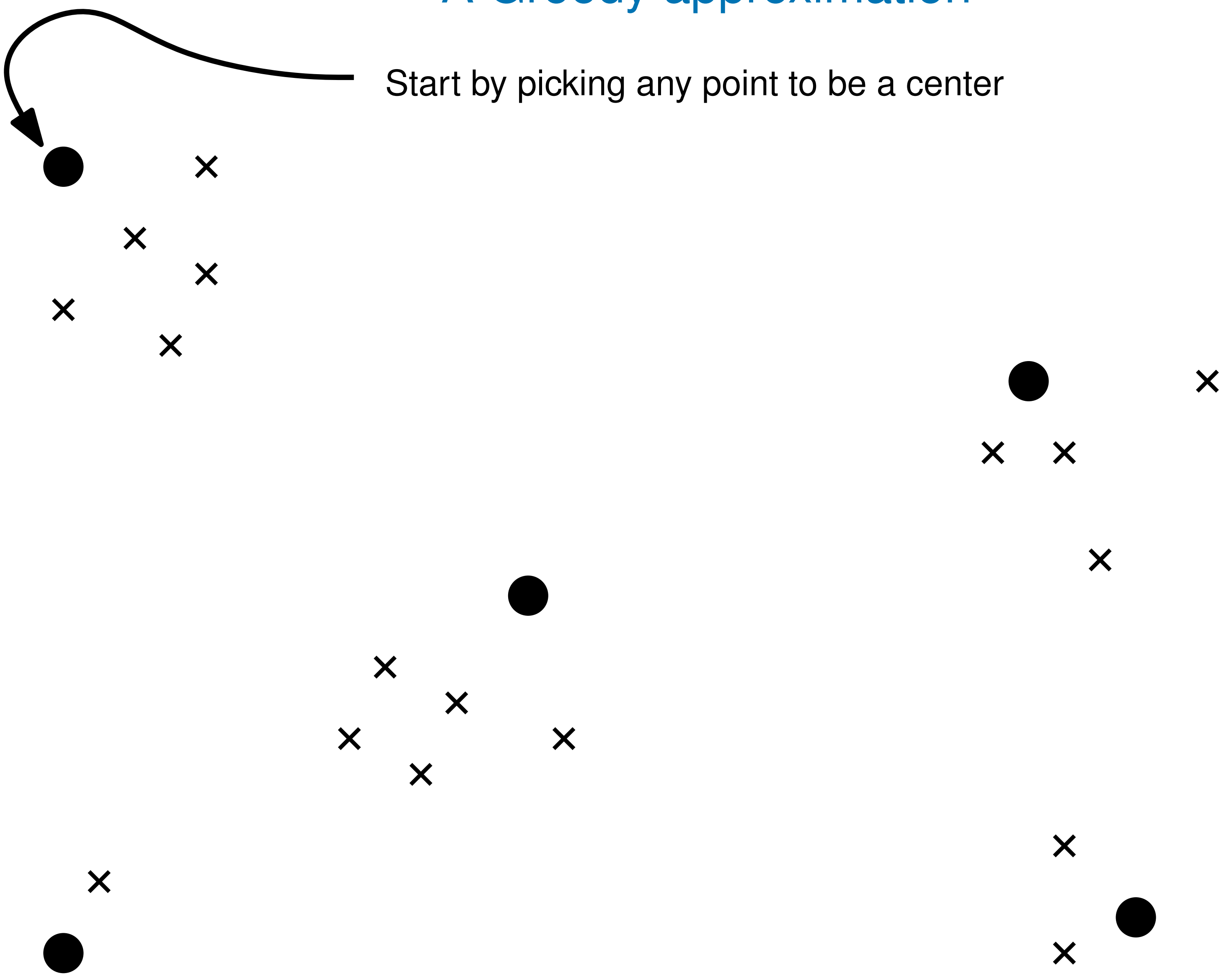
A Greedy approximation

Start by picking any point to be a center



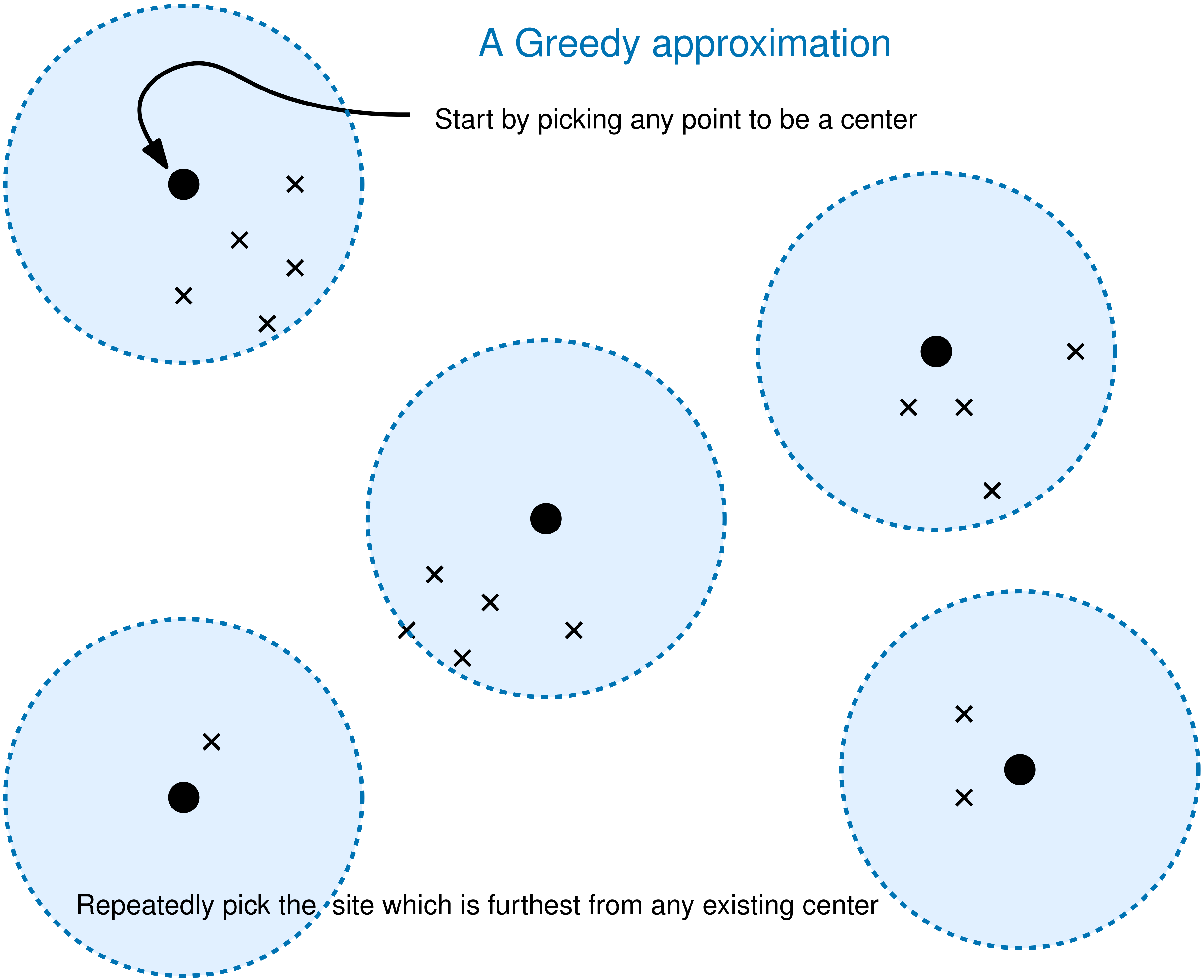
Repeatedly pick the site which is furthest from any existing center

A Greedy approximation



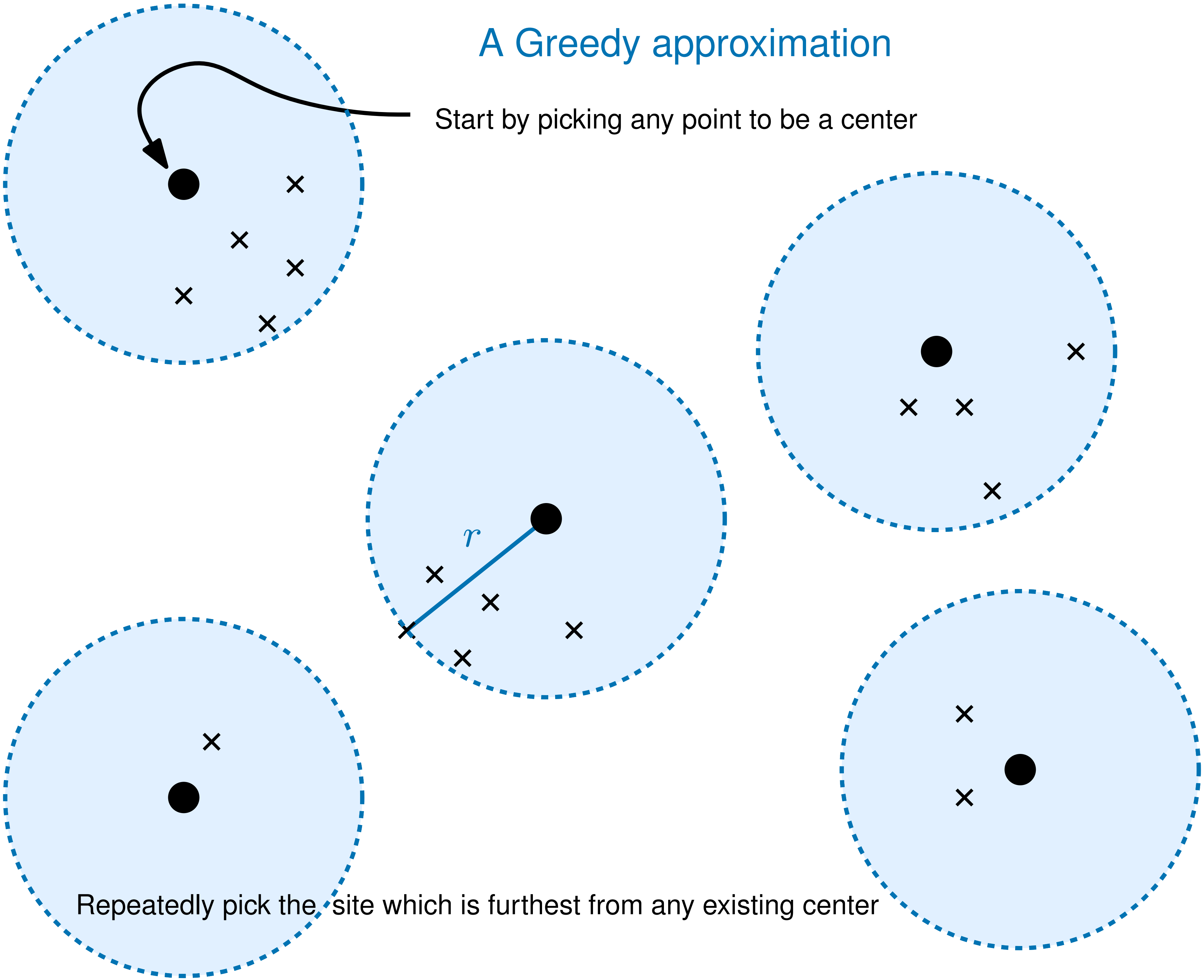
Repeatedly pick the site which is furthest from any existing center

A Greedy approximation



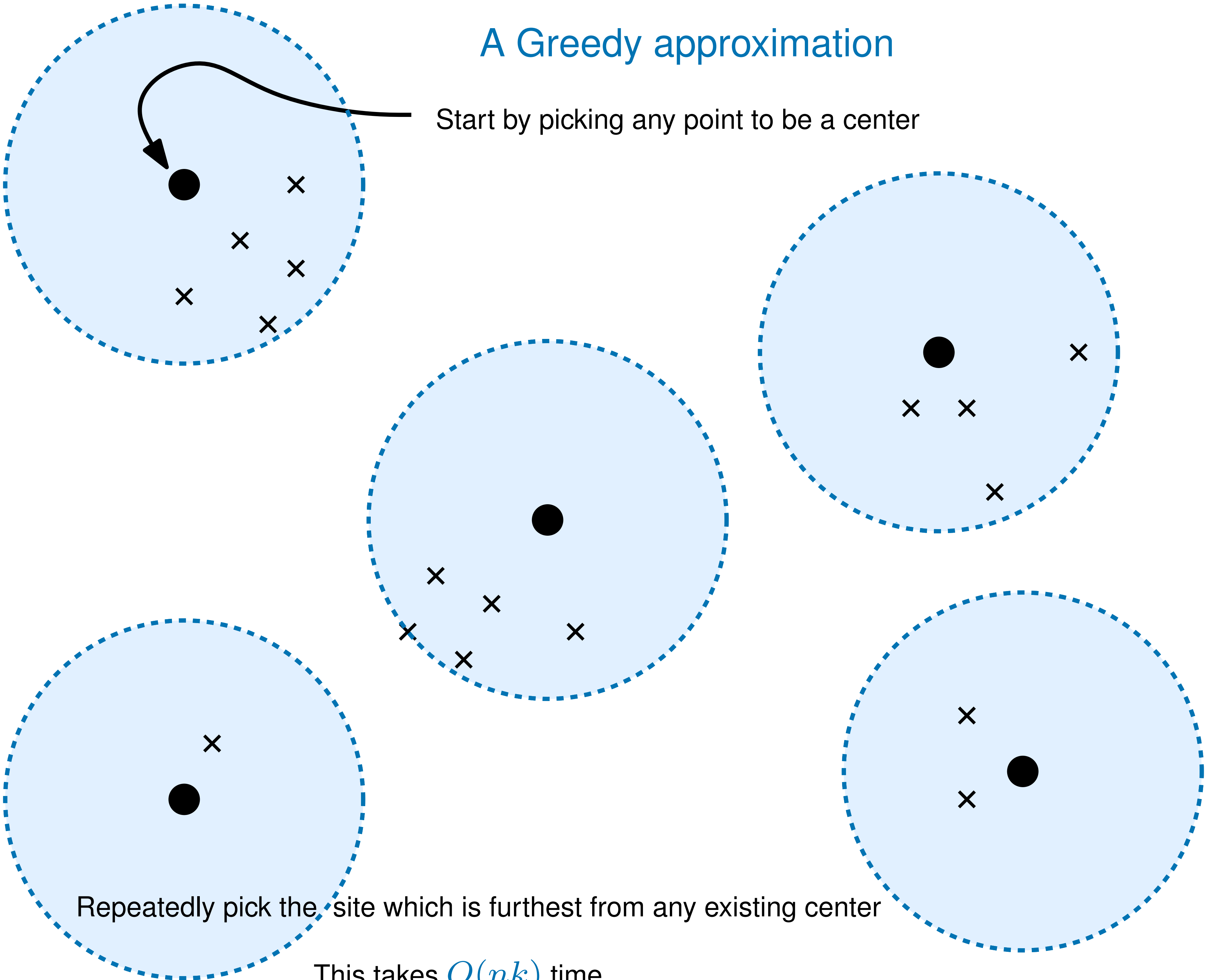
A Greedy approximation

Start by picking any point to be a center



Repeatedly pick the site which is furthest from any existing center

A Greedy approximation



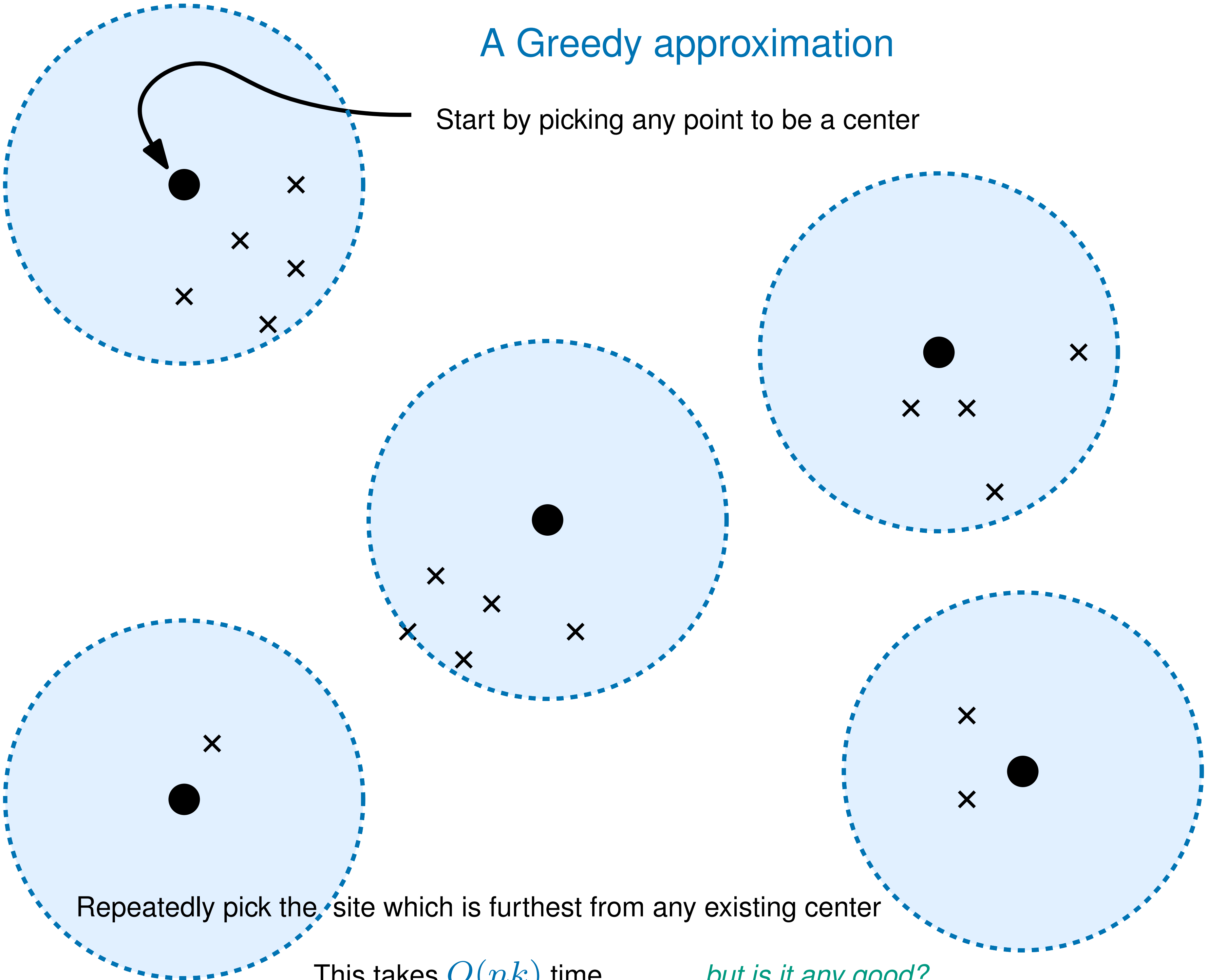
Start by picking any point to be a center

Repeatedly pick the site which is furthest from any existing center

This takes $O(nk)$ time

A Greedy approximation

Start by picking any point to be a center



Repeatedly pick the site which is furthest from any existing center

This takes $O(nk)$ time *but is it any good?*

The Greedy approximation

Theorem The Greedy algorithm for k-center is a 2-approximation algorithm

Proof

Let C_g (resp. C_{Opt}) denote the set of centers selected by Greedy (resp. Optimal)

Let r_g (resp. Opt) denote largest site-center distance using Greedy (resp. Optimal)

The Greedy approximation

Theorem The Greedy algorithm for k-center is a 2-approximation algorithm

Proof

Let C_g (resp. C_{Opt}) denote the set of centers selected by Greedy (resp. Optimal)

Let r_g (resp. Opt) denote largest site-center distance using Greedy (resp. Optimal)

Case 1: No $s_i, s_{i'} \in C_g$ are closest to the same $s_j \in C_{\mathrm{Opt}}$

The Greedy approximation

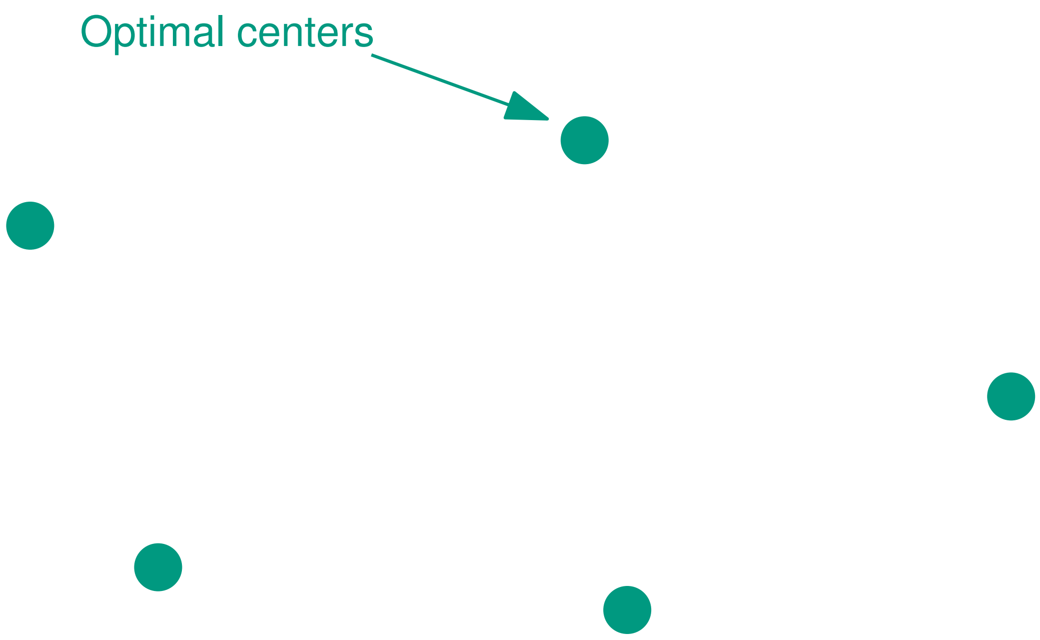
Theorem The Greedy algorithm for k-center is a 2-approximation algorithm

Proof

Let C_g (resp. C_{Opt}) denote the set of centers selected by Greedy (resp. Optimal)

Let r_g (resp. Opt) denote largest site-center distance using Greedy (resp. Optimal)

Case 1: No $s_i, s_{i'} \in C_g$ are closest to the same $s_j \in C_{\mathrm{Opt}}$



The Greedy approximation

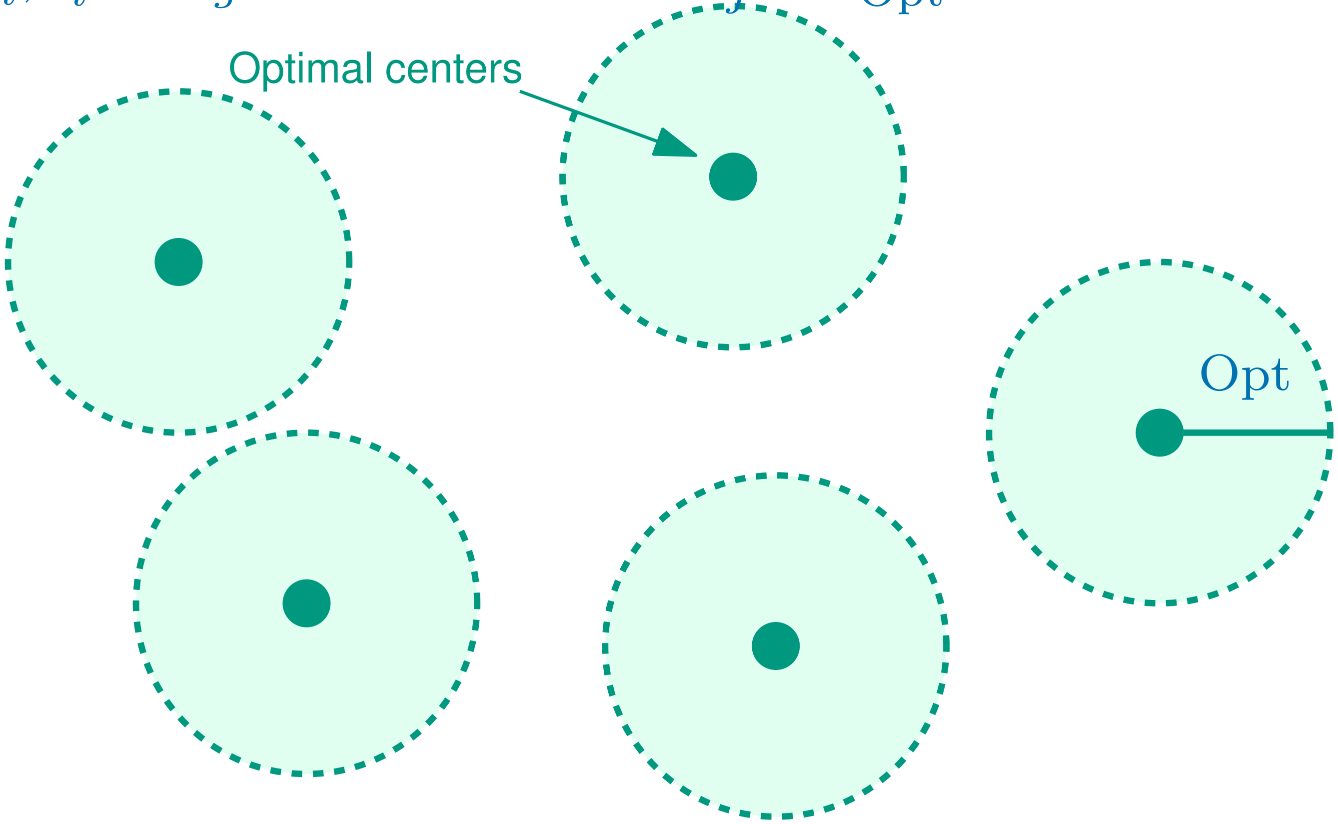
Theorem The Greedy algorithm for k-center is a 2-approximation algorithm

Proof

Let C_g (resp. C_{Opt}) denote the set of centers selected by Greedy (resp. Optimal)

Let r_g (resp. Opt) denote largest site-center distance using Greedy (resp. Optimal)

Case 1: No $s_i, s_{i'} \in C_g$ are closest to the same $s_j \in C_{\mathrm{Opt}}$



The Greedy approximation

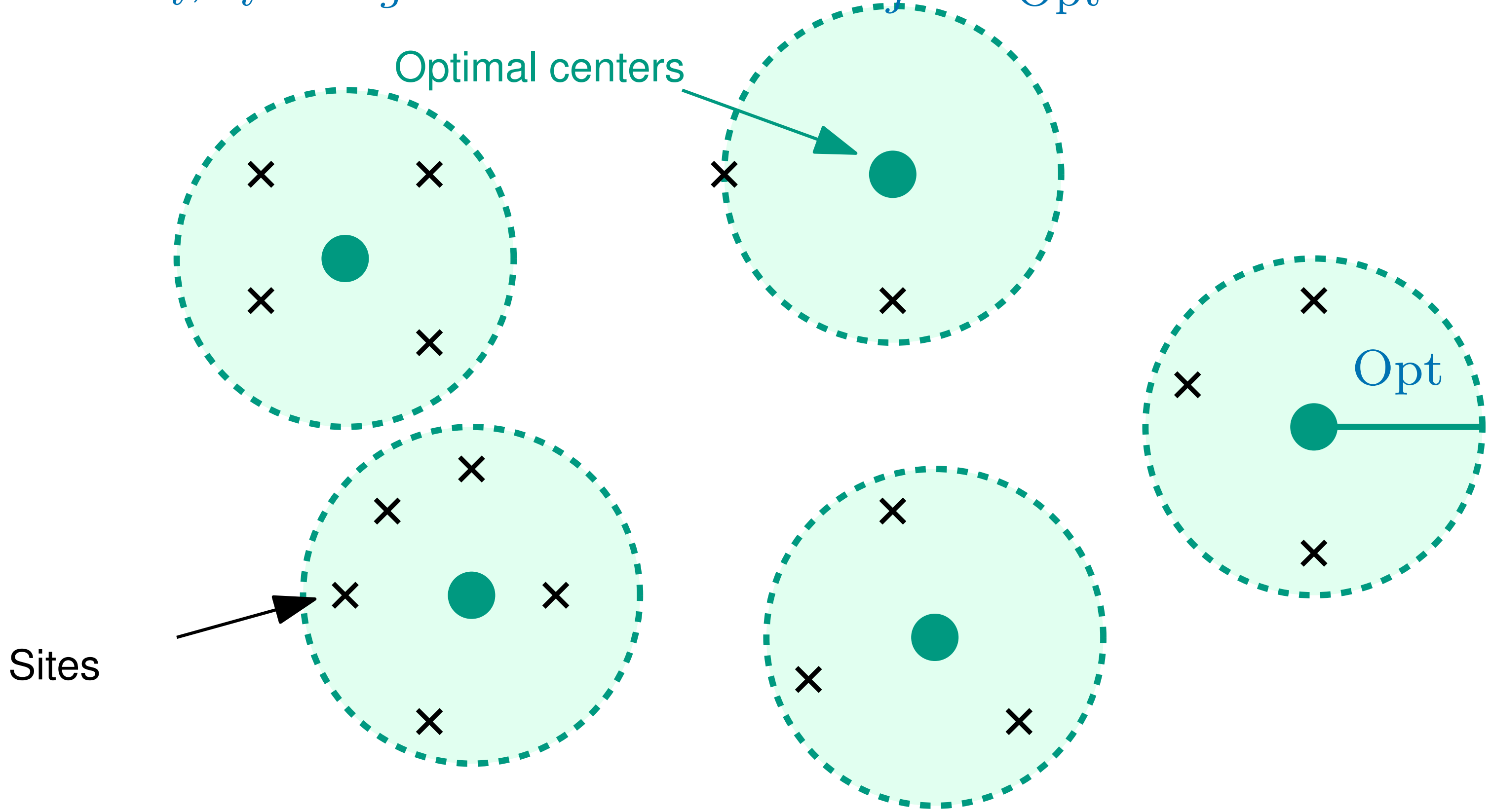
Theorem The Greedy algorithm for k-center is a 2-approximation algorithm

Proof

Let C_g (resp. C_{Opt}) denote the set of centers selected by Greedy (resp. Optimal)

Let r_g (resp. Opt) denote largest site-center distance using Greedy (resp. Optimal)

Case 1: No $s_i, s_{i'} \in C_g$ are closest to the same $s_j \in C_{\mathrm{Opt}}$



The Greedy approximation

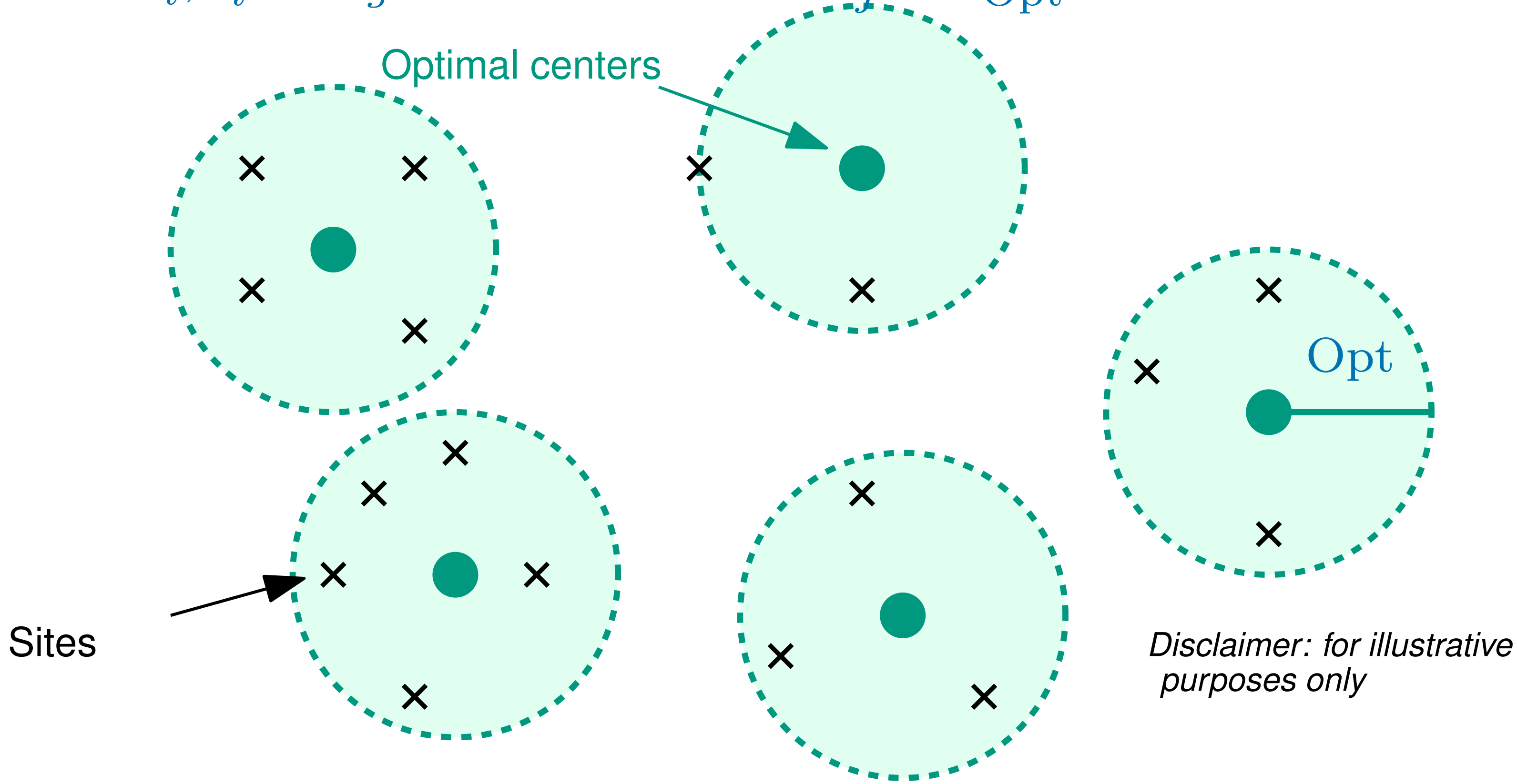
Theorem The Greedy algorithm for k-center is a 2-approximation algorithm

Proof

Let C_g (resp. C_{Opt}) denote the set of centers selected by Greedy (resp. Optimal)

Let r_g (resp. Opt) denote largest site-center distance using Greedy (resp. Optimal)

Case 1: No $s_i, s_{i'} \in C_g$ are closest to the same $s_j \in C_{\mathrm{Opt}}$



The Greedy approximation

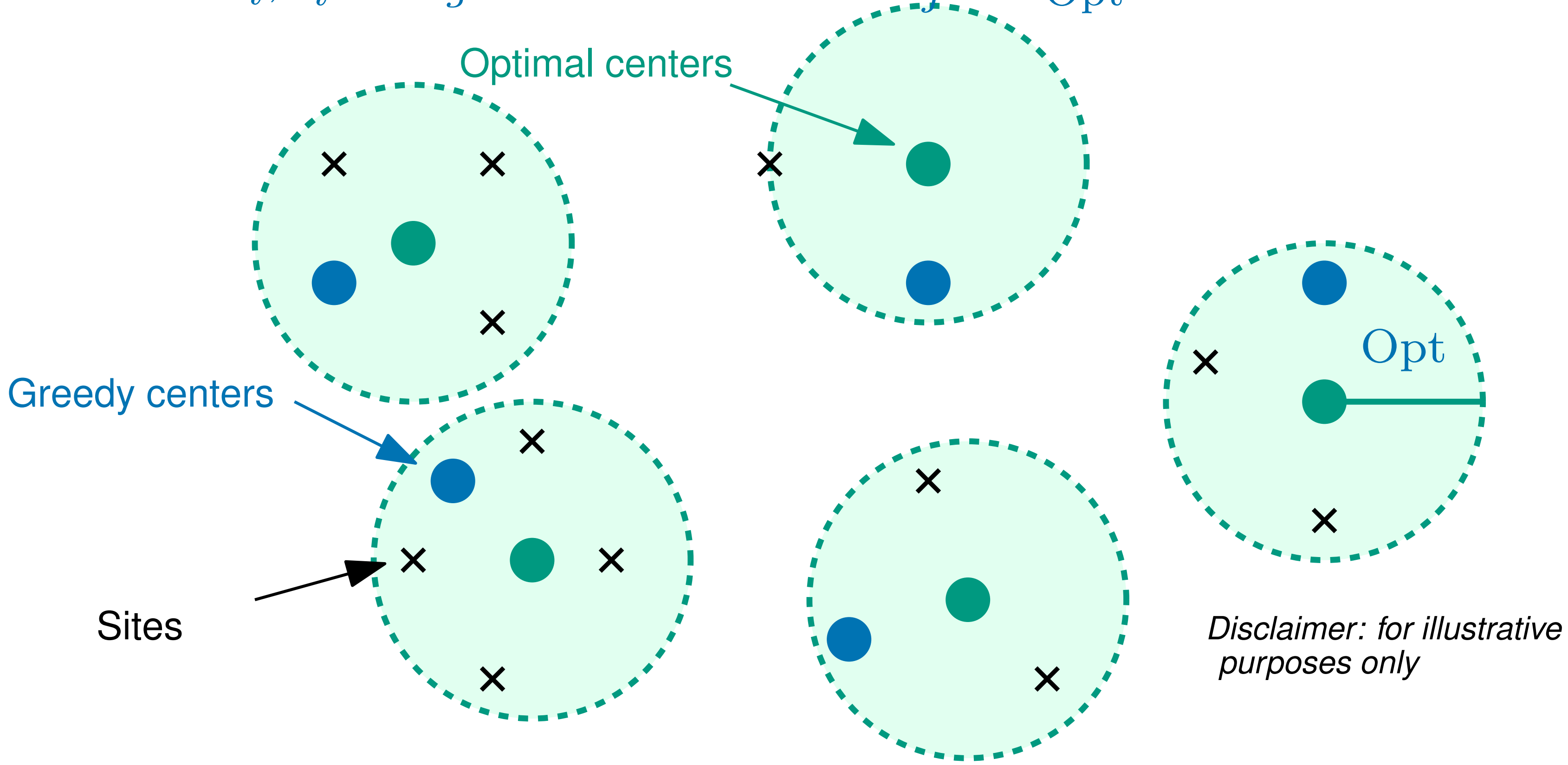
Theorem The Greedy algorithm for k-center is a 2-approximation algorithm

Proof

Let C_g (resp. C_{Opt}) denote the set of centers selected by Greedy (resp. Optimal)

Let r_g (resp. Opt) denote largest site-center distance using Greedy (resp. Optimal)

Case 1: No $s_i, s_{i'} \in C_g$ are closest to the same $s_j \in C_{\mathrm{Opt}}$



The Greedy approximation

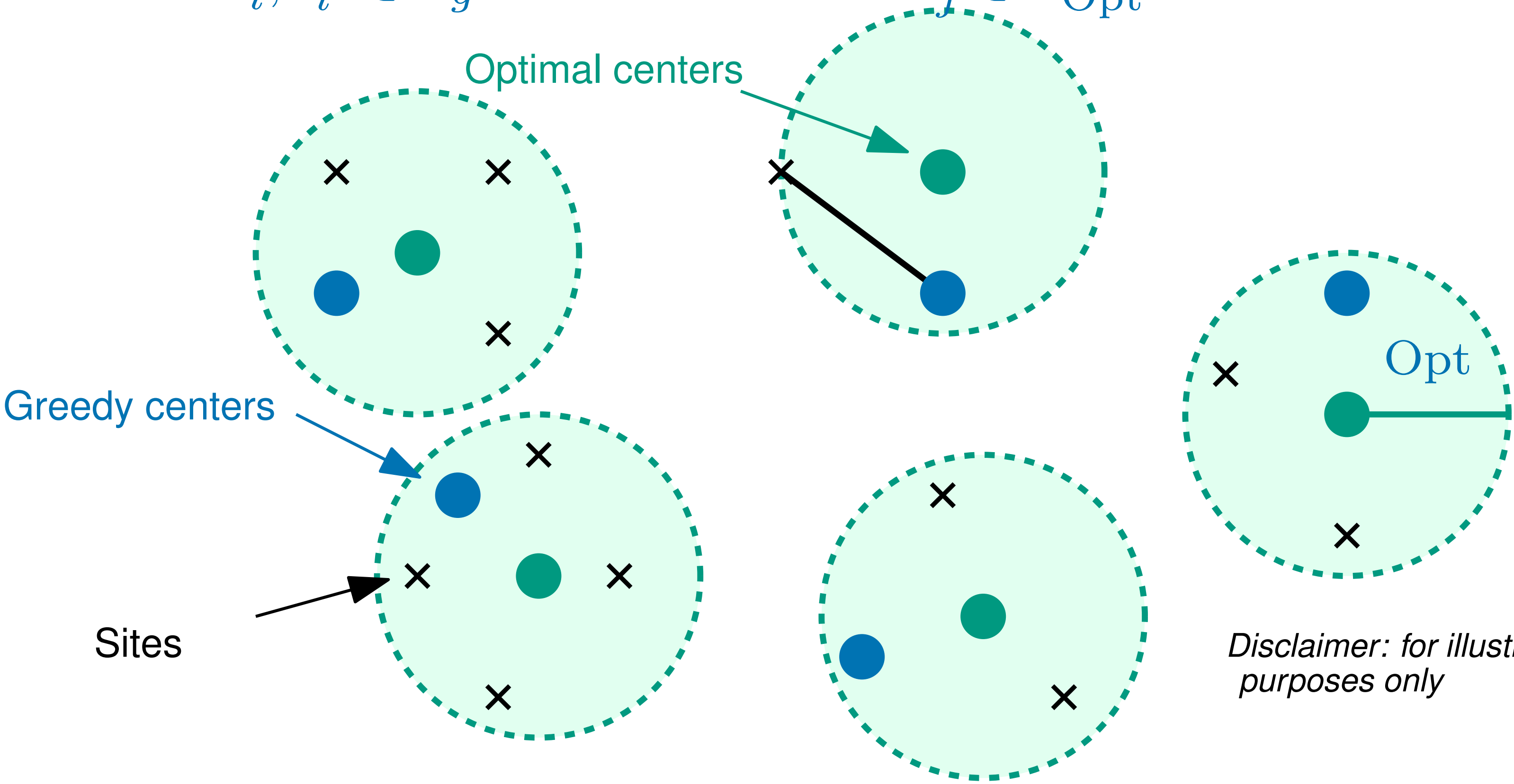
Theorem The Greedy algorithm for k-center is a 2-approximation algorithm

Proof

Let C_g (resp. C_{Opt}) denote the set of centers selected by Greedy (resp. Optimal)

Let r_g (resp. Opt) denote largest site-center distance using Greedy (resp. Optimal)

Case 1: No $s_i, s_{i'} \in C_g$ are closest to the same $s_j \in C_{\mathrm{Opt}}$



Disclaimer: for illustrative purposes only

The Greedy approximation

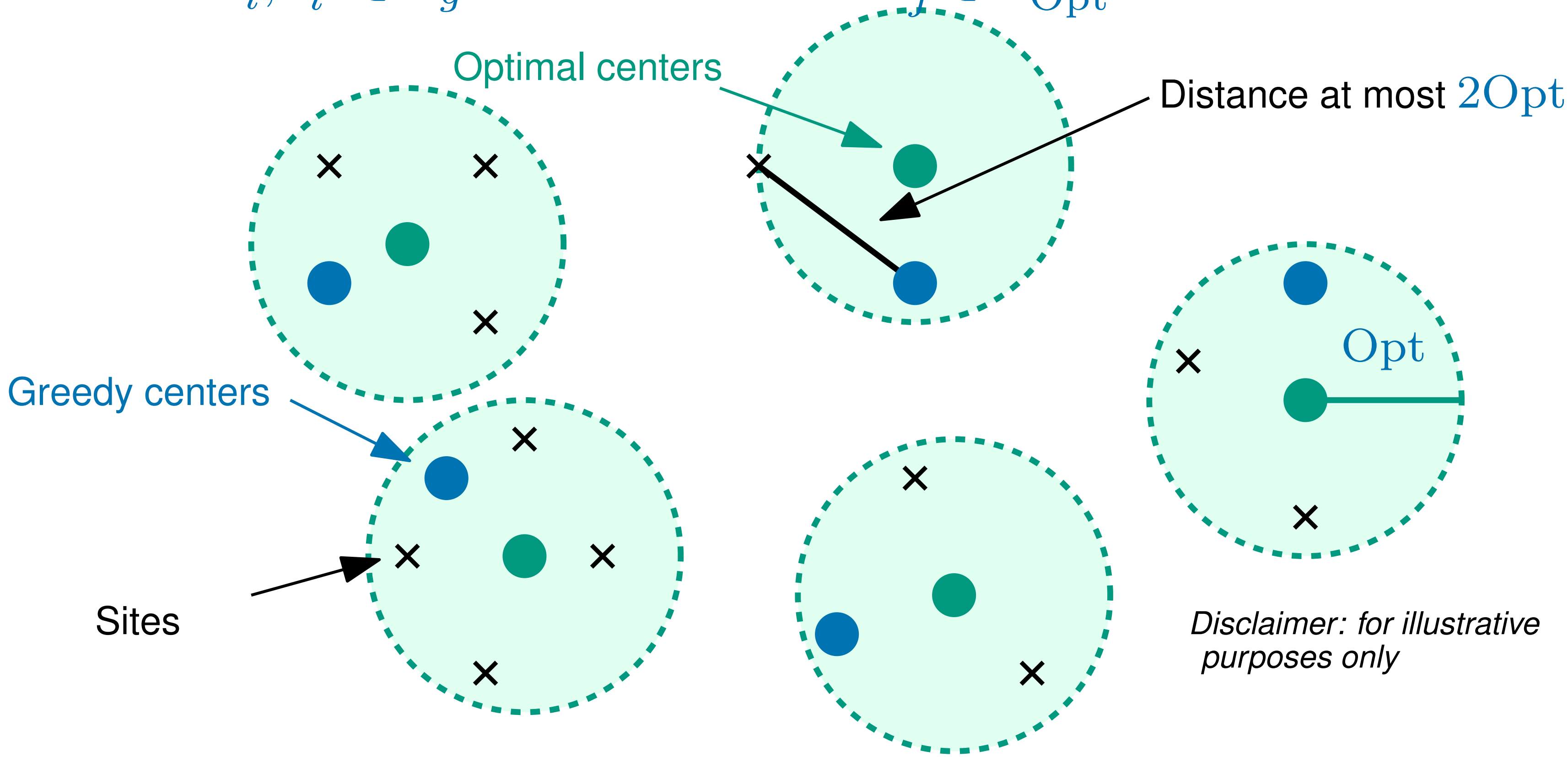
Theorem The Greedy algorithm for k-center is a 2-approximation algorithm

Proof

Let C_g (resp. C_{Opt}) denote the set of centers selected by Greedy (resp. Optimal)

Let r_g (resp. Opt) denote largest site-center distance using Greedy (resp. Optimal)

Case 1: No $s_i, s_{i'} \in C_g$ are closest to the same $s_j \in C_{\mathrm{Opt}}$



Disclaimer: for illustrative purposes only

The Greedy approximation

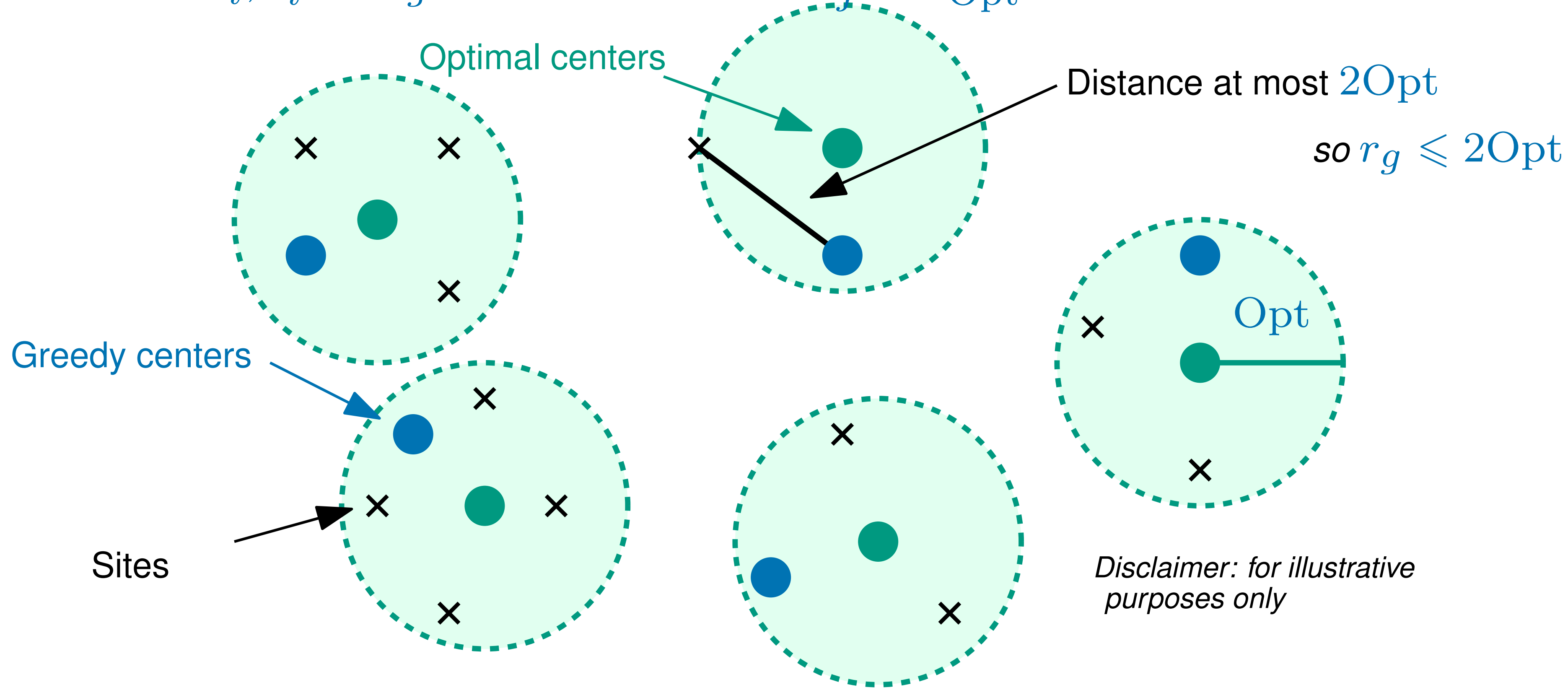
Theorem The Greedy algorithm for k-center is a 2-approximation algorithm

Proof

Let C_g (resp. C_{Opt}) denote the set of centers selected by Greedy (resp. Optimal)

Let r_g (resp. Opt) denote largest site-center distance using Greedy (resp. Optimal)

Case 1: No $s_i, s_{i'} \in C_g$ are closest to the same $s_j \in C_{\mathrm{Opt}}$



The Greedy approximation

Theorem The Greedy algorithm for k-center is a 2-approximation algorithm

Proof

Let C_g (resp. C_{Opt}) denote the set of centers selected by Greedy (resp. Optimal)

Let r_g (resp. Opt) denote largest site-center distance using Greedy (resp. Optimal)

Case 2: Some $s_i, s_{i'} \in C_g$ are closest to the same $s_j \in C_{\mathrm{Opt}}$

The Greedy approximation

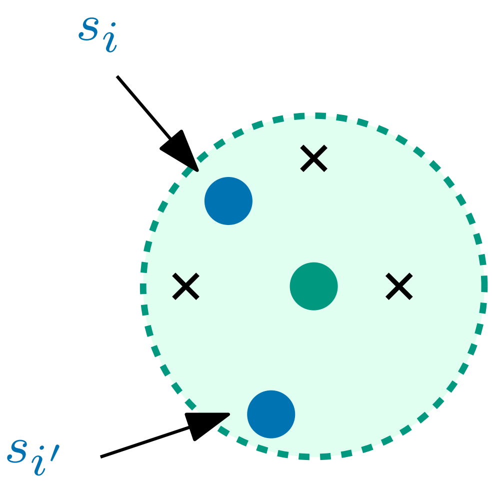
Theorem The Greedy algorithm for k-center is a 2-approximation algorithm

Proof

Let C_g (resp. C_{Opt}) denote the set of centers selected by Greedy (resp. Optimal)

Let r_g (resp. Opt) denote largest site-center distance using Greedy (resp. Optimal)

Case 2: Some $s_i, s_{i'} \in C_g$ are closest to the same $s_j \in C_{\mathrm{Opt}}$



The Greedy approximation

Theorem The Greedy algorithm for k-center is a 2-approximation algorithm

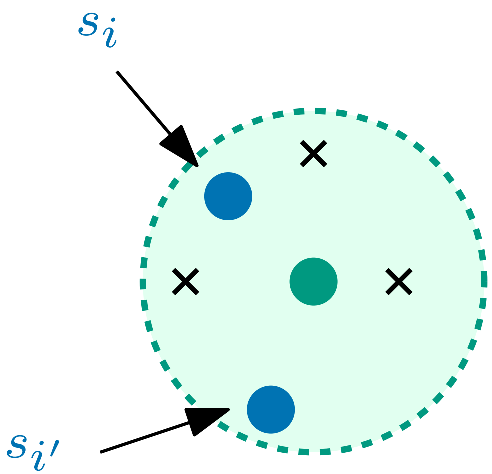
Proof

Let C_g (resp. C_{Opt}) denote the set of centers selected by Greedy (resp. Optimal)

Let r_g (resp. Opt) denote largest site-center distance using Greedy (resp. Optimal)

Case 2: Some $s_i, s_{i'} \in C_g$ are closest to the same $s_j \in C_{\mathrm{Opt}}$

Assume wlog. that Greedy made s_i a center after $s_{i'}$



The Greedy approximation

Theorem The Greedy algorithm for k-center is a 2-approximation algorithm

Proof

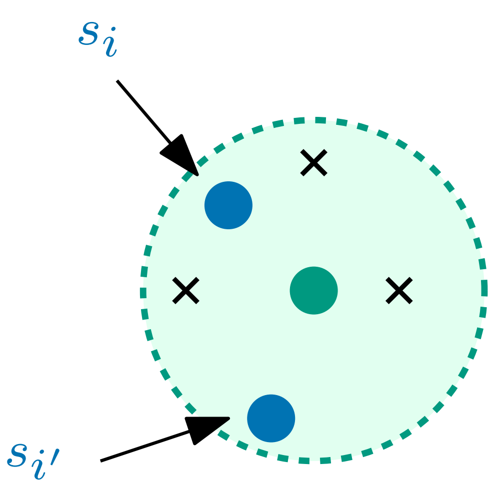
Let C_g (resp. C_{Opt}) denote the set of centers selected by Greedy (resp. Optimal)

Let r_g (resp. Opt) denote largest site-center distance using Greedy (resp. Optimal)

Case 2: Some $s_i, s_{i'} \in C_g$ are closest to the same $s_j \in C_{\mathrm{Opt}}$

Assume wlog. that Greedy made s_i a center after $s_{i'}$

s_i was added as a center because it was



The Greedy approximation

Theorem The Greedy algorithm for k-center is a 2-approximation algorithm

Proof

Let C_g (resp. C_{Opt}) denote the set of centers selected by Greedy (resp. Optimal)

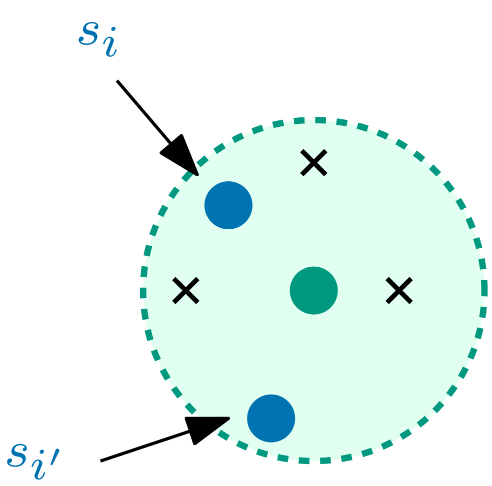
Let r_g (resp. Opt) denote largest site-center distance using Greedy (resp. Optimal)

Case 2: Some $s_i, s_{i'} \in C_g$ are closest to the same $s_j \in C_{\mathrm{Opt}}$

Assume wlog. that Greedy made s_i a center after $s_{i'}$

s_i was added as a center because it was

the furthest from any existing Greedy center



The Greedy approximation

Theorem The Greedy algorithm for k-center is a 2-approximation algorithm

Proof

Let C_g (resp. C_{Opt}) denote the set of centers selected by Greedy (resp. Optimal)

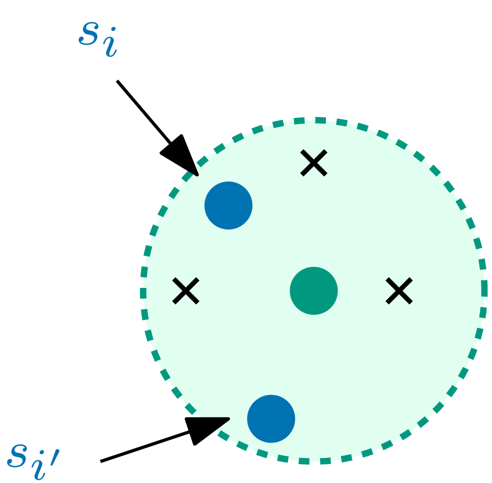
Let r_g (resp. Opt) denote largest site-center distance using Greedy (resp. Optimal)

Case 2: Some $s_i, s_{i'} \in C_g$ are closest to the same $s_j \in C_{\mathrm{Opt}}$

Assume wlog. that Greedy made s_i a center after $s_{i'}$

s_i was added as a center because it was
the furthest from any existing Greedy center

However, s_i is at most $2\mathrm{Opt}$ away from $s_{i'}$



The Greedy approximation

Theorem The Greedy algorithm for k-center is a 2-approximation algorithm

Proof

Let C_g (resp. C_{Opt}) denote the set of centers selected by Greedy (resp. Optimal)

Let r_g (resp. Opt) denote largest site-center distance using Greedy (resp. Optimal)

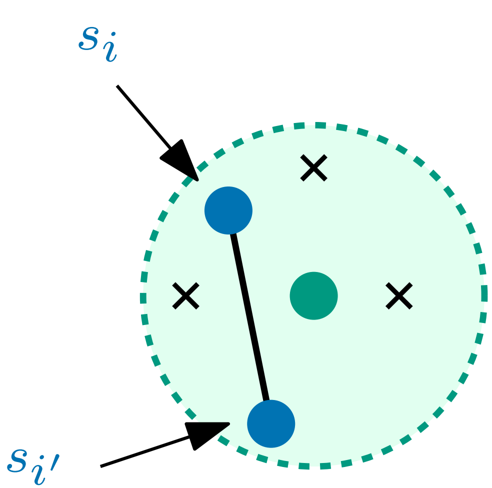
Case 2: Some $s_i, s_{i'} \in C_g$ are closest to the same $s_j \in C_{\mathrm{Opt}}$

Assume wlog. that Greedy made s_i a center after $s_{i'}$

s_i was added as a center because it was

the furthest from any existing Greedy center

However, s_i is at most $2\mathrm{Opt}$ away from $s_{i'}$



The Greedy approximation

Theorem The Greedy algorithm for k-center is a 2-approximation algorithm

Proof

Let C_g (resp. C_{Opt}) denote the set of centers selected by Greedy (resp. Optimal)

Let r_g (resp. Opt) denote largest site-center distance using Greedy (resp. Optimal)

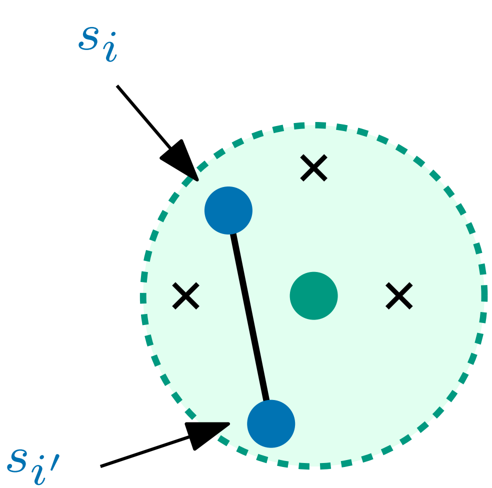
Case 2: Some $s_i, s_{i'} \in C_g$ are closest to the same $s_j \in C_{\mathrm{Opt}}$

Assume wlog. that Greedy made s_i a center after $s_{i'}$

s_i was added as a center because it was the furthest from any existing Greedy center

However, s_i is at most $2\mathrm{Opt}$ away from $s_{i'}$

So even before adding s_i as a center, all sites were $\leqslant 2\mathrm{Opt}$ away from a Greedy center



The Greedy approximation

Theorem The Greedy algorithm for k-center is a 2-approximation algorithm

Proof

Let C_g (resp. C_{Opt}) denote the set of centers selected by Greedy (resp. Optimal)

Let r_g (resp. Opt) denote largest site-center distance using Greedy (resp. Optimal)

Case 2: Some $s_i, s_{i'} \in C_g$ are closest to the same $s_j \in C_{\text{Opt}}$

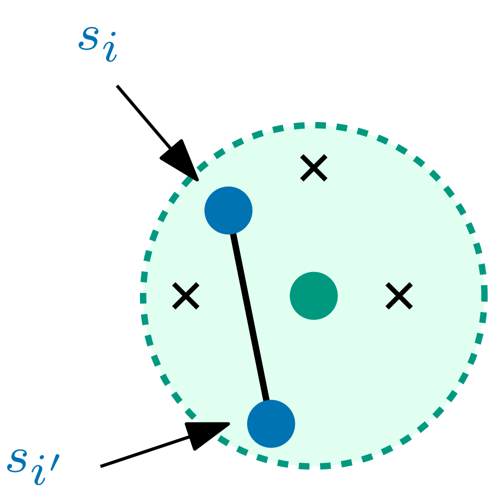
Assume wlog. that Greedy made s_i a center after $s_{i'}$

s_i was added as a center because it was
the furthest from any existing Greedy center

However, s_i is at most 2Opt away from $s_{i'}$

So even before adding s_i as a center, all sites
were $\leqslant 2\text{Opt}$ away from a Greedy center

Therefore, $r_g \leqslant 2\text{Opt}$



k-center Conclusions

Theorem The Greedy algorithm for k-center is a 2-approximation algorithm

which runs in $O(nk)$ time

k-center Conclusions

Theorem The Greedy algorithm for k-center is a 2-approximation algorithm which runs in $O(nk)$ time

- The approximation works for any (metric) distance function,

k-center Conclusions

Theorem The Greedy algorithm for k-center is a 2-approximation algorithm

which runs in $O(nk)$ time

- The approximation works for any (metric) distance function,

  $d(s_i, s_j) = L_1$ or L_∞ for example

k-center Conclusions

Theorem The Greedy algorithm for k-center is a 2-approximation algorithm which runs in $O(nk)$ time

- The approximation works for any (metric) distance function,

  $d(s_i, s_j) = L_1$ or L_∞ for example

- Distance function d is a metric iff

  $d(x, y) = d(y, x)$, $d(x, y) \geqslant 0$

  ($d(x, y) = 0$ iff $x = y$) and $d(x, z) \leqslant d(x, y) + d(y, z)$

k-center Conclusions

Theorem The Greedy algorithm for k-center is a 2-approximation algorithm which runs in $O(nk)$ time

- The approximation works for any (metric) distance function,

  $d(s_i, s_j) = L_1$ or L_∞ for example

- Distance function d is a metric iff

  $d(x, y) = d(y, x)$, $d(x, y) \geqslant 0$

  ($d(x, y) = 0$ iff $x = y$) and $d(x, z) \leqslant d(x, y) + d(y, z)$

- For a general (metric) d, the problem is not α-approximable with $\alpha < 2$

k-center Conclusions

Theorem The Greedy algorithm for k-center is a 2-approximation algorithm which runs in $O(nk)$ time

- The approximation works for any (metric) distance function,

  $d(s_i, s_j) = L_1$ or L_∞ for example

- Distance function d is a metric iff

  $d(x, y) = d(y, x)$, $d(x, y) \geqslant 0$

  ($d(x, y) = 0$ iff $x = y$) and $d(x, z) \leqslant d(x, y) + d(y, z)$

- For a general (metric) d, the problem is not α-approximable with $\alpha < 2$
- For $d = L_2$, the problem is not α-approximable with $\alpha < \sqrt{3} \approx 1.73$

k-center Conclusions

Theorem The Greedy algorithm for k-center is a 2-approximation algorithm which runs in $O(nk)$ time

- The approximation works for any (metric) distance function,

  $d(s_i, s_j) = L_1$ or L_∞ for example

- Distance function d is a metric iff

  $d(x, y) = d(y, x)$, $d(x, y) \geqslant 0$

  ($d(x, y) = 0$ iff $x = y$) and $d(x, z) \leqslant d(x, y) + d(y, z)$

- For a general (metric) d, the problem is not α-approximable with $\alpha < 2$
- For $d = L_2$, the problem is not α-approximable with $\alpha < \sqrt{3} \approx 1.73$
- For $d = L_1$ or $d = L_\infty$, the problem is not α-approximable with $\alpha < 2$