Matching Problems

A 3/2-approximation algorithm for the student-project allocation problem with ties

Frances Cooper

Supervisor: Dr David Manlove

Outline

Outline

- Algorithms

Outline

- Algorithms
- Matching problems

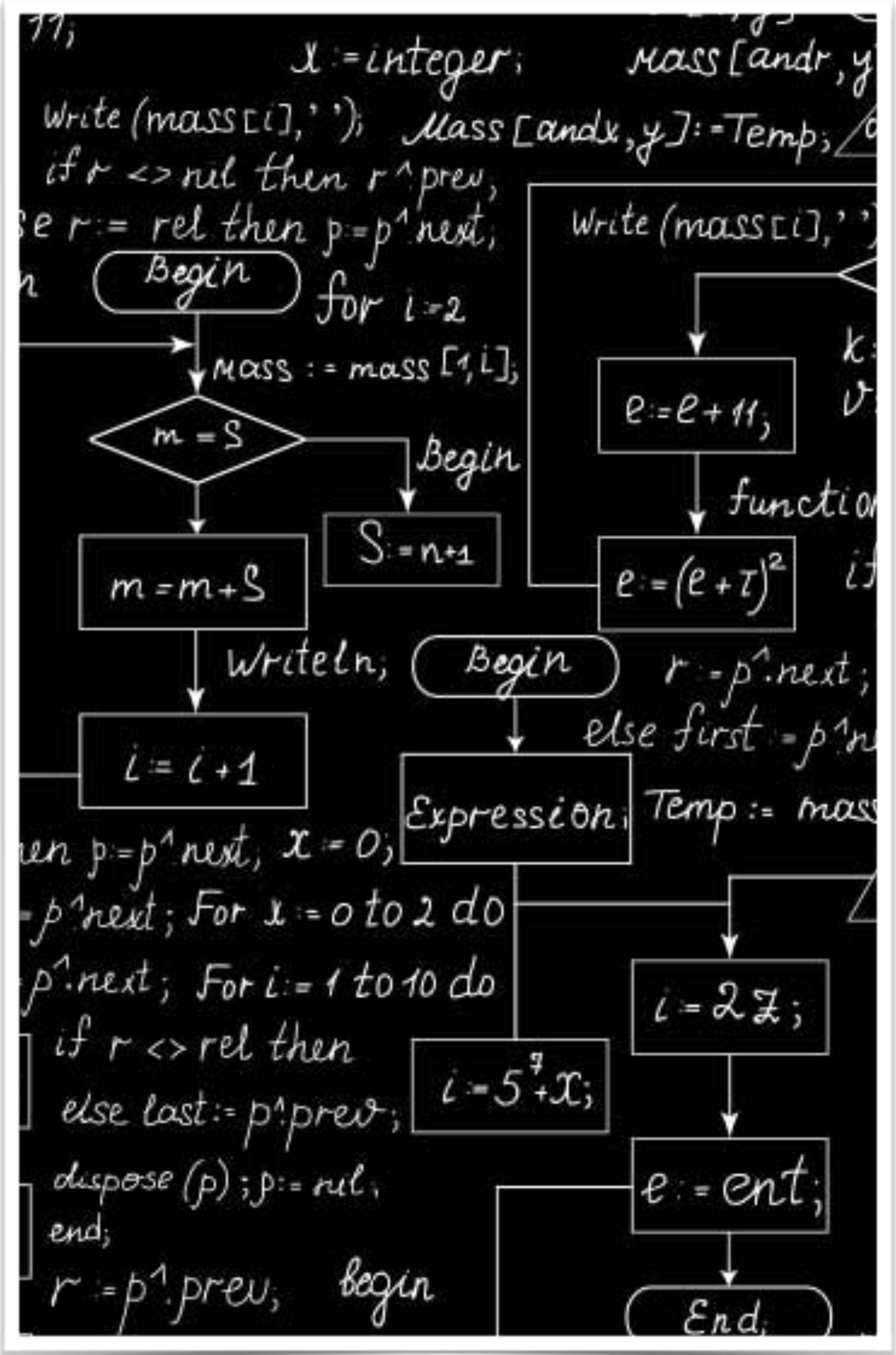
Outline

- Algorithms
- Matching problems
- Stable matchings

Outline

- Algorithms
- Matching problems
- Stable matchings
- Finding maximum stable matchings

Algorithms



x := integer;
mass [andr, y
Write (mass[i], ' ');
Mass [andx, y] := Temp;
if r <> nil then r^prev;
e r := rel then p := p^next;
Write (mass[i], ' '
Begin
for i := 2
mass : = mass [1, i];
m = S
Begin
S := n+1
e := e + 11;
function
m = m+S
e := (e + τ)²
Writeln;
Begin
r := p^next;
else first := p^n
i = i + 1
Expression;
Temp := mass
en p := p^next; x := 0;
p^next; For x := 0 to 2 do
p^next; For i := 1 to 10 do
if r <> rel then
i := 2Z;
else last := p^prev;
i := 5⁷+x;
dispose (p); p := nil;
end;
e := ent;
r := p^prev;
begin
End;

Algorithms

Algorithms

- What is an algorithm? - a list of instructions given to a computer in order to solve a problem

Algorithms

- What is an algorithm? - a list of instructions given to a computer in order to solve a problem

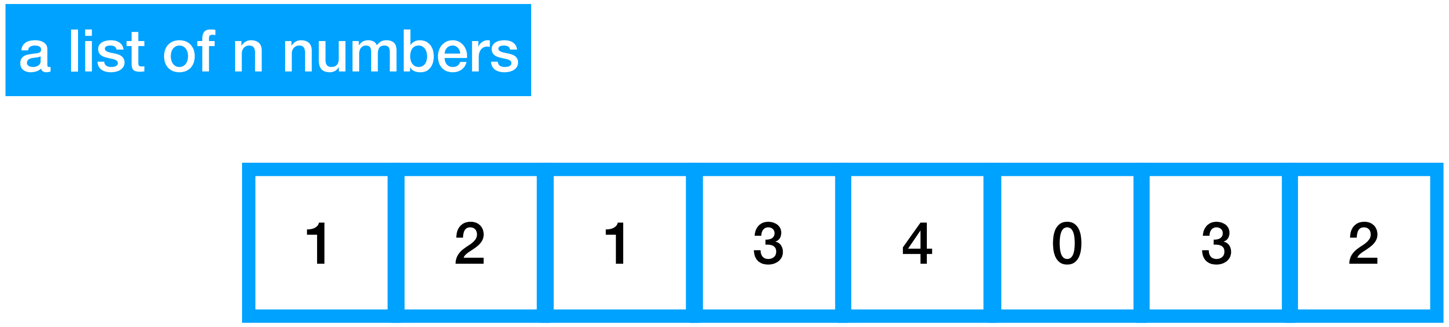
- The **time complexity** of an algorithm is a measure of how the number of operations grows with respect to input size

Algorithms

- What is an algorithm? - a list of instructions given to a computer in order to solve a problem
- The **time complexity** of an algorithm is a measure of how the number of operations grows with respect to input size
- Example:

Algorithms

- What is an algorithm? - a list of instructions given to a computer in order to solve a problem
- The **time complexity** of an algorithm is a measure of how the number of operations grows with respect to input size
- Example:



a list of n numbers

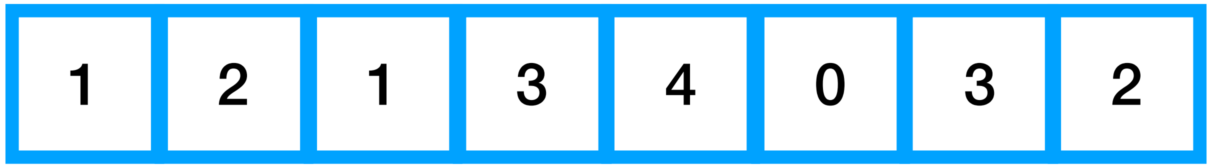
1	2	1	3	4	0	3	2

Algorithms

- What is an algorithm? - a list of instructions given to a computer in order to solve a problem
- The **time complexity** of an algorithm is a measure of how the number of operations grows with respect to input size
- Example:

a list of n numbers

task: double each number



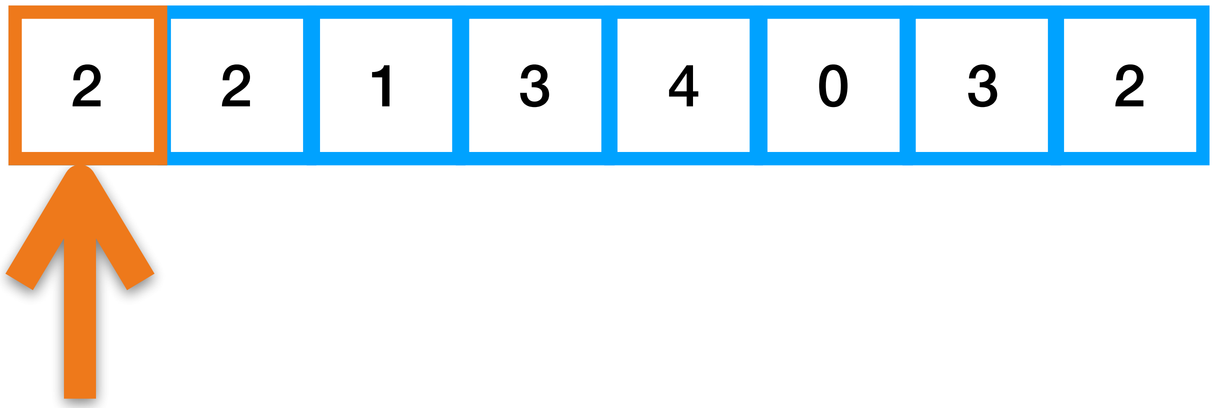
1	2	1	3	4	0	3	2

Algorithms

- What is an algorithm? - a list of instructions given to a computer in order to solve a problem
- The **time complexity** of an algorithm is a measure of how the number of operations grows with respect to input size
- Example:

a list of n numbers

task: double each number



Algorithms

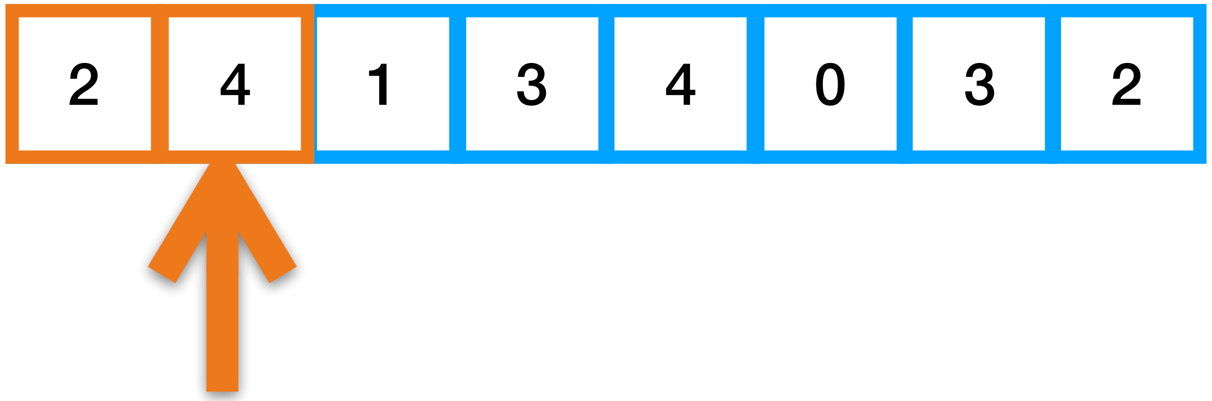
- What is an algorithm? - a list of instructions given to a computer in order to solve a problem

- The **time complexity** of an algorithm is a measure of how the number of operations grows with respect to input size

- Example:

a list of n numbers

task: double each number



2	4	1	3	4	0	3	2

Algorithms

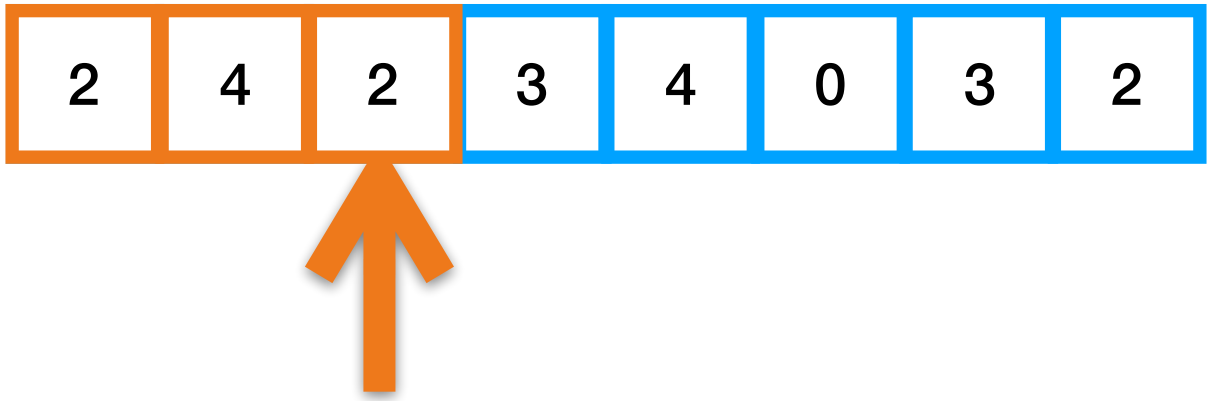
- What is an algorithm? - a list of instructions given to a computer in order to solve a problem

- The **time complexity** of an algorithm is a measure of how the number of operations grows with respect to input size

- Example:

a list of n numbers

task: double each number



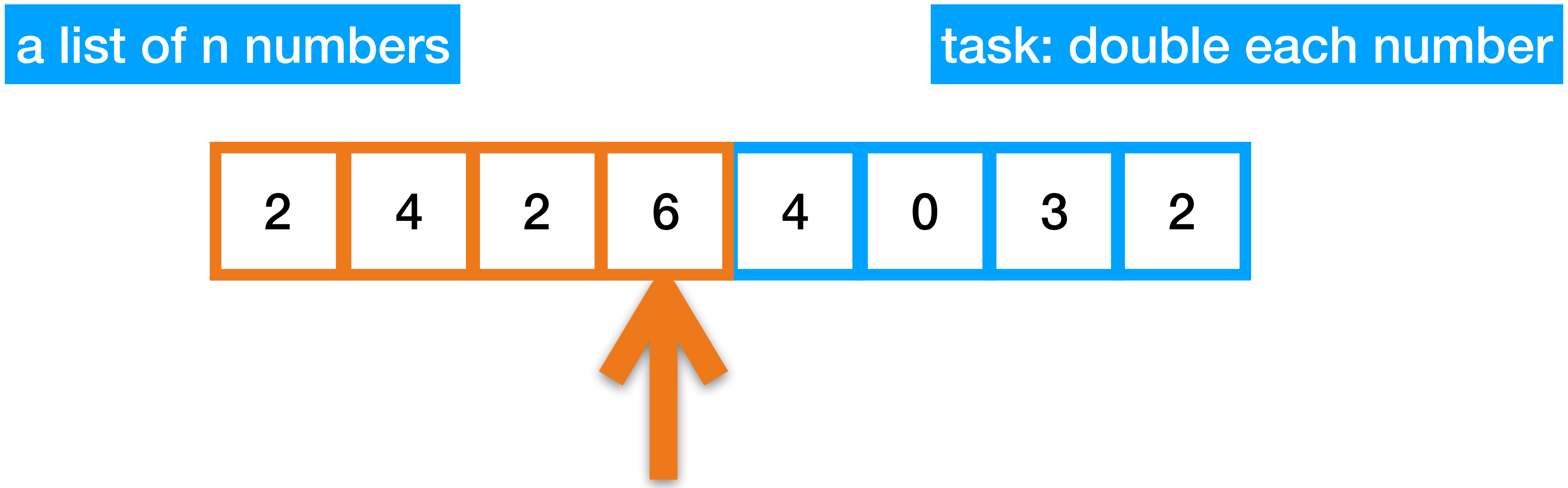
2	4	2	3	4	0	3	2

Algorithms

- What is an algorithm? - a list of instructions given to a computer in order to solve a problem

- The **time complexity** of an algorithm is a measure of how the number of operations grows with respect to input size

- Example:



a list of n numbers

task: double each number

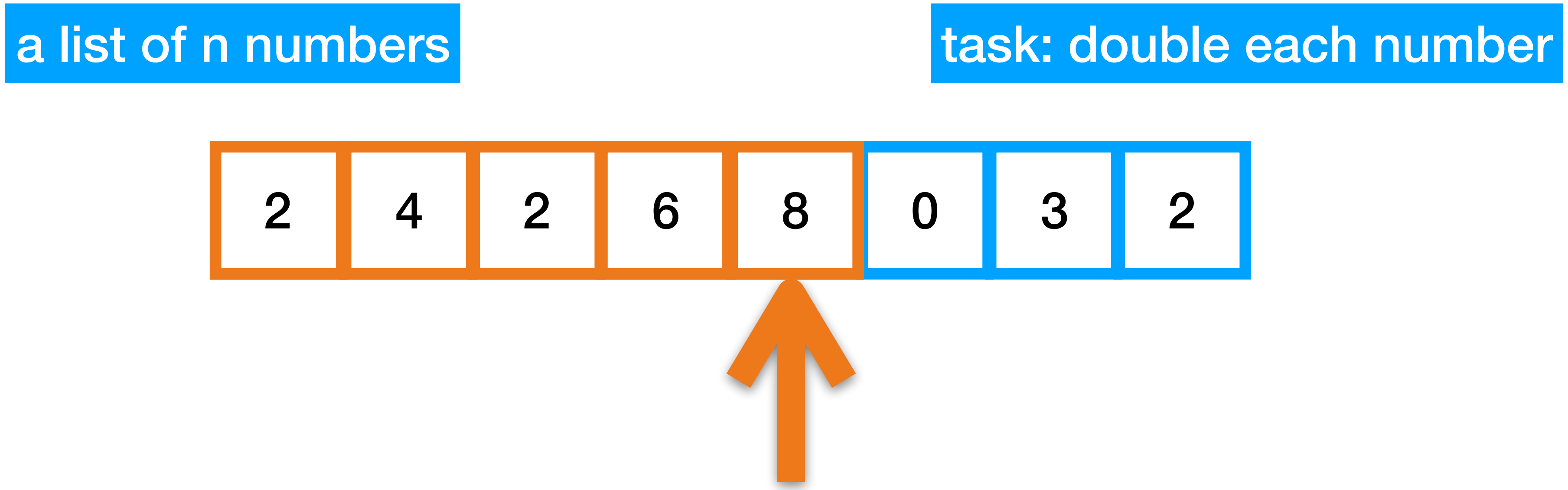
2	4	2	6	4	0	3	2

Algorithms

- What is an algorithm? - a list of instructions given to a computer in order to solve a problem

- The **time complexity** of an algorithm is a measure of how the number of operations grows with respect to input size

- Example:



a list of n numbers

task: double each number

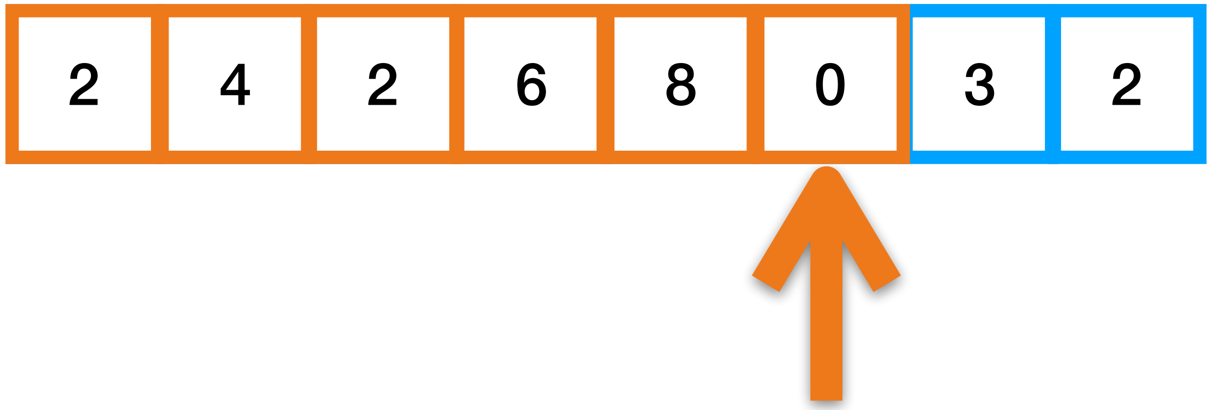
2	4	2	6	8	0	3	2

Algorithms

- What is an algorithm? - a list of instructions given to a computer in order to solve a problem
- The **time complexity** of an algorithm is a measure of how the number of operations grows with respect to input size
- Example:

a list of n numbers

task: double each number



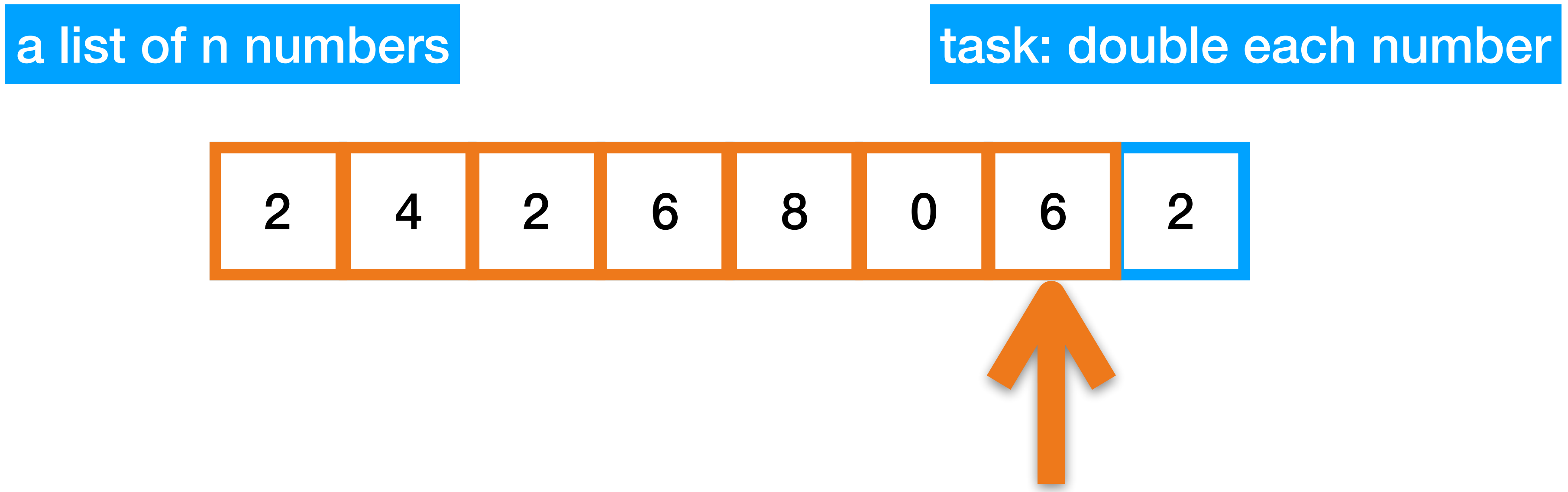
2	4	2	6	8	0	3	2

Algorithms

- What is an algorithm? - a list of instructions given to a computer in order to solve a problem

- The **time complexity** of an algorithm is a measure of how the number of operations grows with respect to input size

- Example:



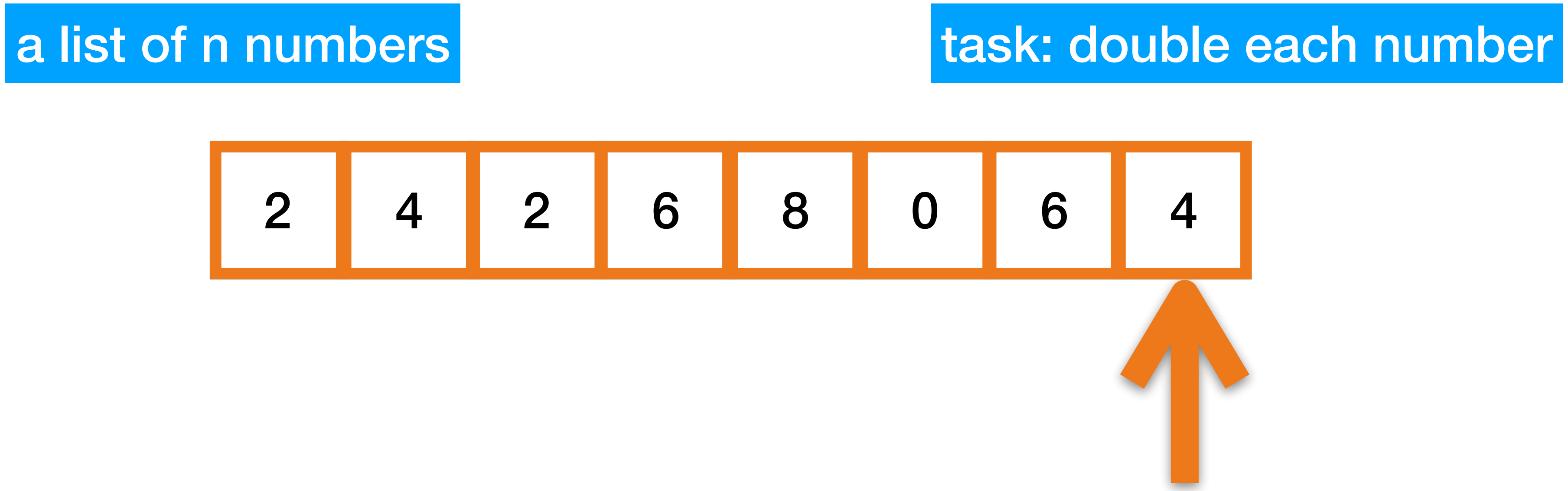
a list of n numbers

task: double each number

2	4	2	6	8	0	6	2

Algorithms

- What is an algorithm? - a list of instructions given to a computer in order to solve a problem
- The **time complexity** of an algorithm is a measure of how the number of operations grows with respect to input size
- Example:



a list of n numbers

task: double each number

2	4	2	6	8	0	6	4

Algorithms

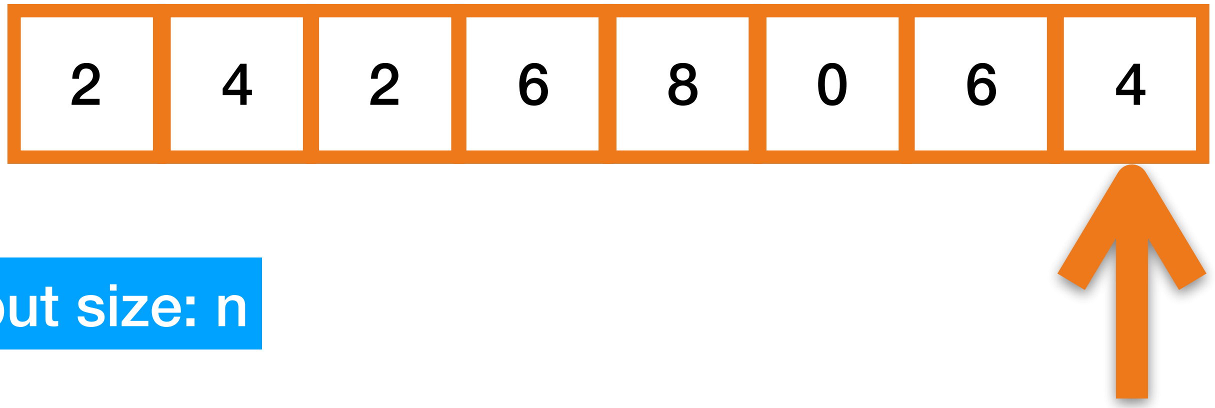
- What is an algorithm? - a list of instructions given to a computer in order to solve a problem

- The **time complexity** of an algorithm is a measure of how the number of operations grows with respect to input size

- Example:

a list of n numbers

task: double each number



2	4	2	6	8	0	6	4

input size: n

Algorithms

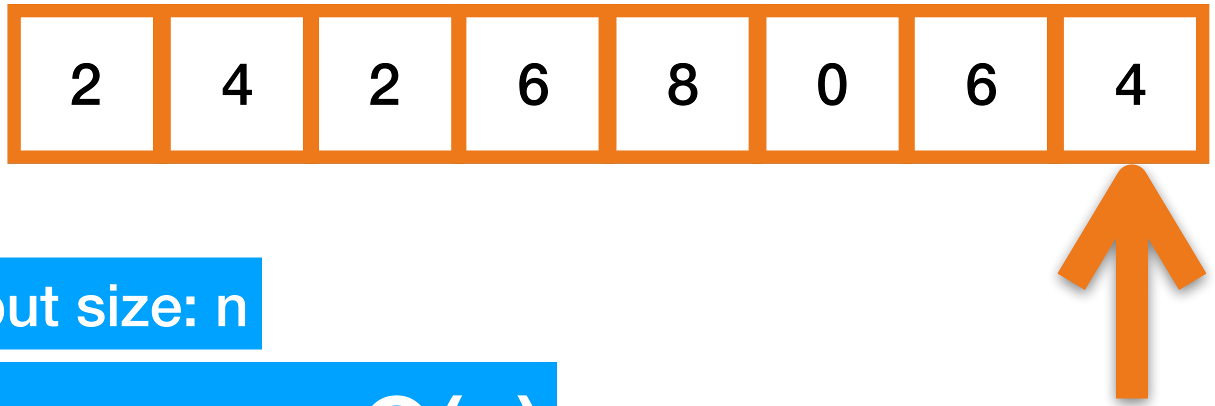
- What is an algorithm? - a list of instructions given to a computer in order to solve a problem

- The **time complexity** of an algorithm is a measure of how the number of operations grows with respect to input size

- Example:

a list of n numbers

task: double each number



2	4	2	6	8	0	6	4

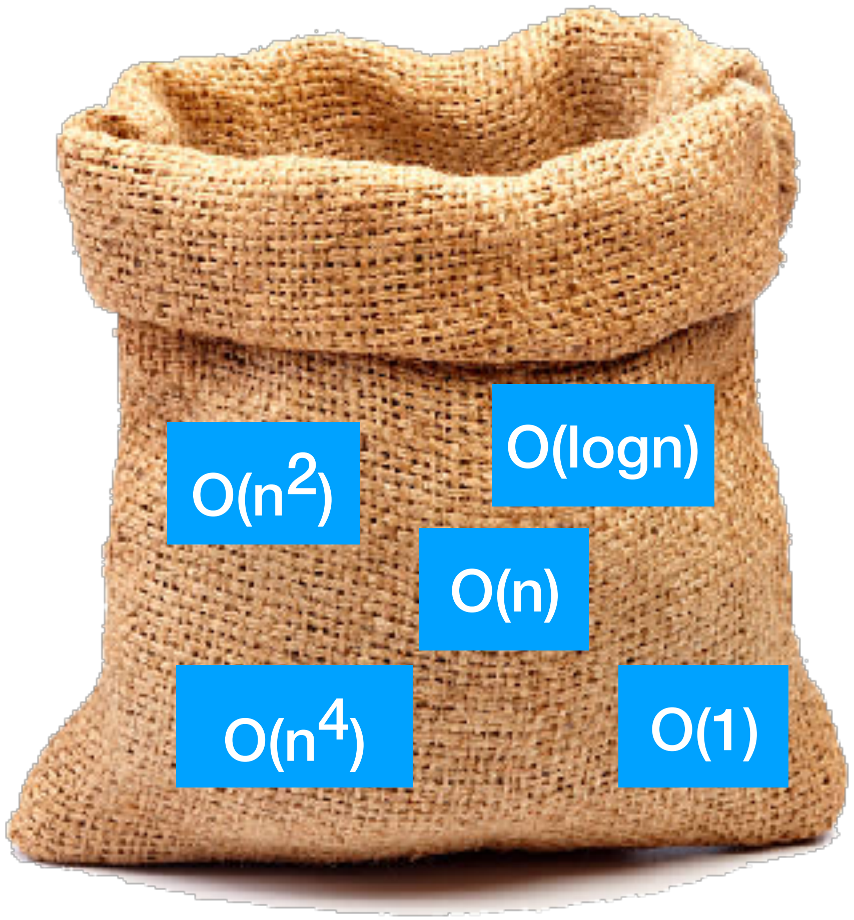
input size: n

algorithm time complexity: O(n)

Efficient algorithms

Efficient algorithms

efficient



$O(n^2)$

$O(\log n)$

$O(n)$

$O(n^4)$

$O(1)$

inefficient

$O(2^n)$

$O(n!)$

$O(n^n)$

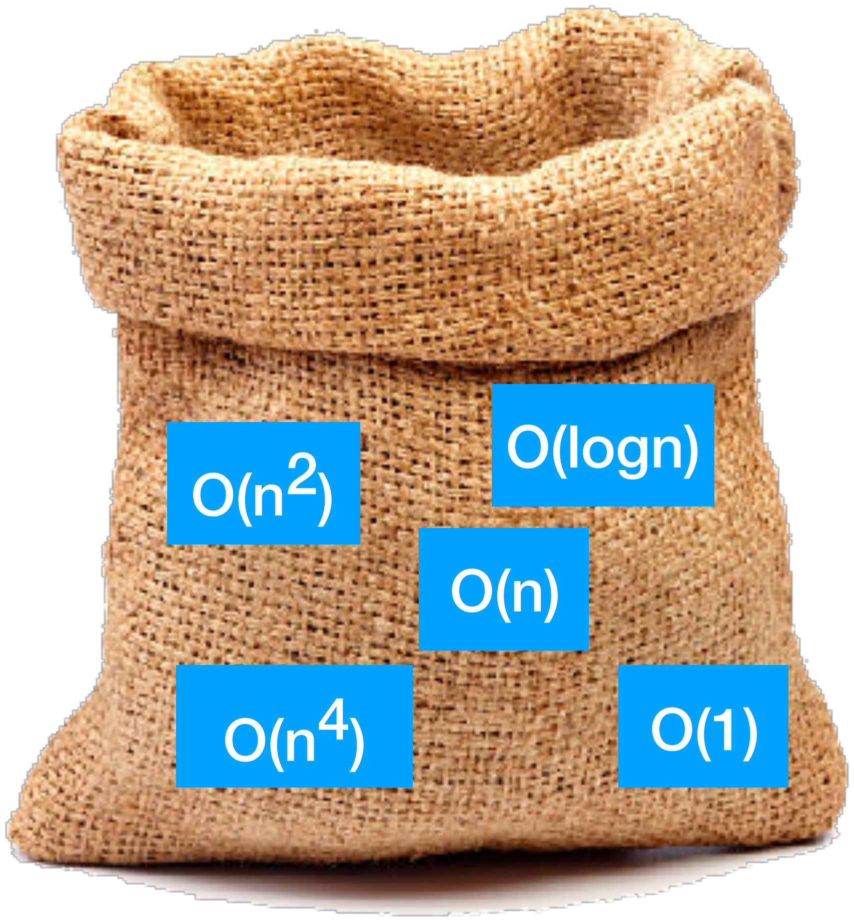
Efficient algorithms

efficient

polynomial time

inefficient

exponential time



$O(n^2)$

$O(\log n)$

$O(n)$

$O(n^4)$

$O(1)$

$O(2^n)$

$O(n!)$

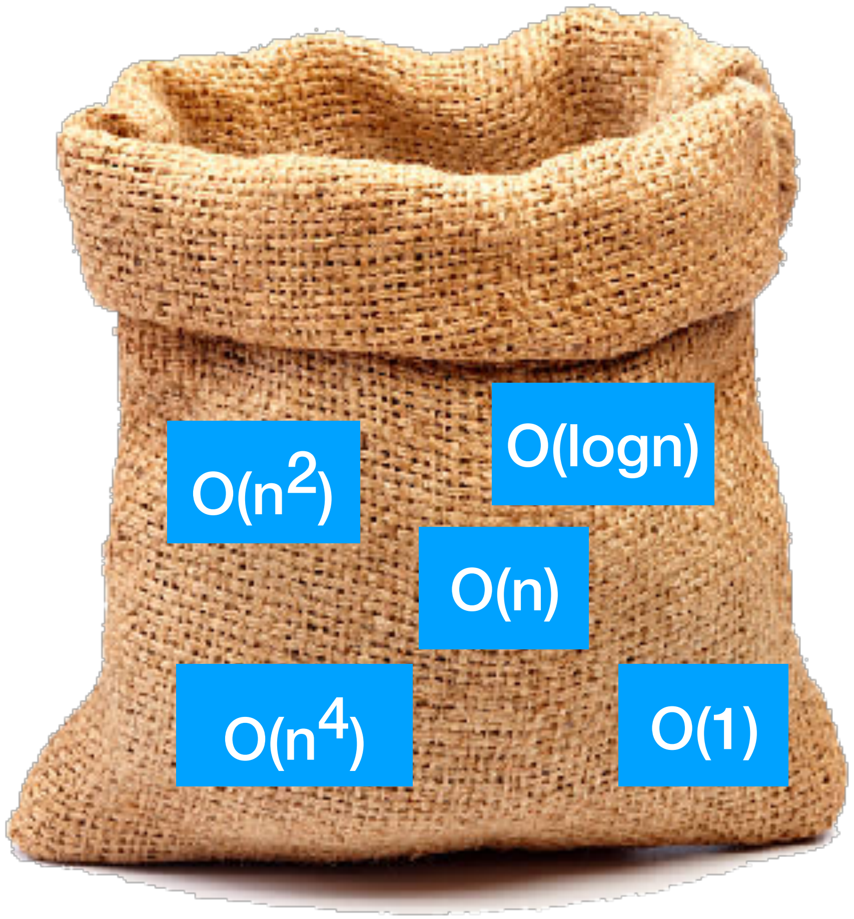
$O(n^n)$

Efficient algorithms

efficient

polynomial time

fast in general



$O(n^2)$

$O(\log n)$

$O(n)$

$O(n^4)$

$O(1)$

inefficient

exponential time

slow in general

$O(2^n)$

$O(n!)$

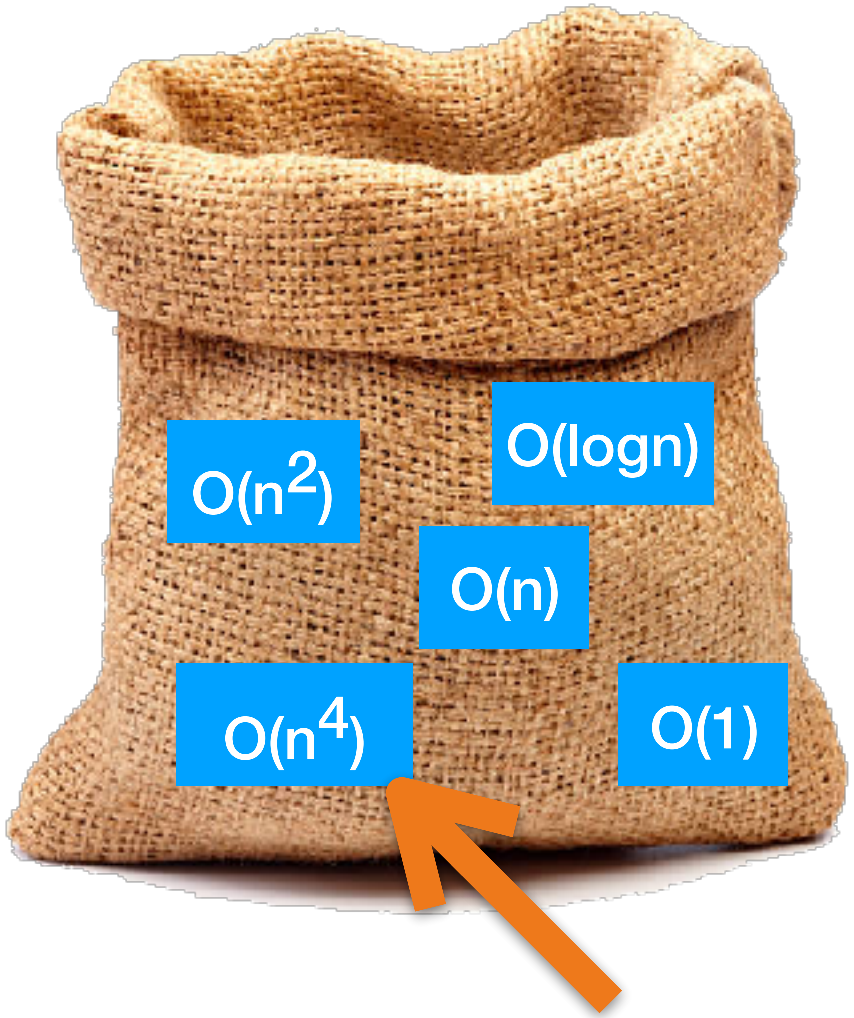
$O(n^n)$

Efficient algorithms

efficient

polynomial time

fast in general



$O(n^2)$

$O(\log n)$

$O(n)$

$O(n^4)$

$O(1)$

inefficient

exponential time

slow in general

$O(2^n)$

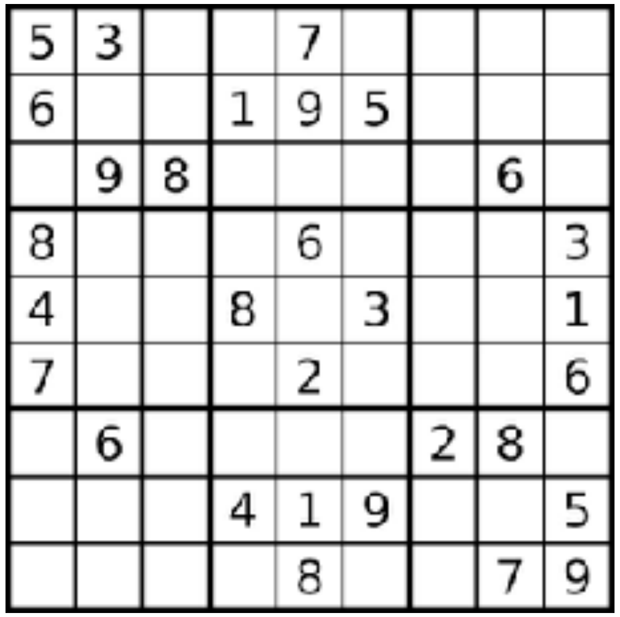
$O(n!)$

$O(n^n)$

NP hardness

NP hardness

- If a problem is **NP hard**, then there is no known efficient algorithm that can solve it

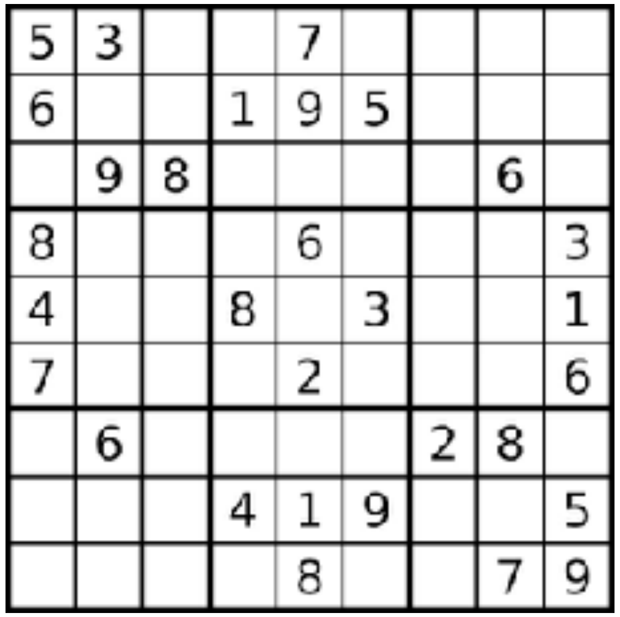


5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

NP hardness

- If a problem is **NP hard**, then there is no known efficient algorithm that can solve it

- Is it possible for an efficient algorithm to solve an **NP hard** problem?

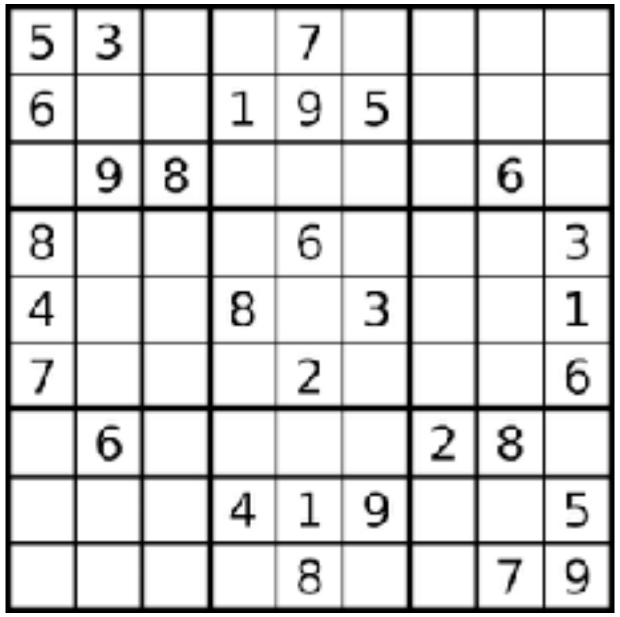


5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

NP hardness

- If a problem is **NP hard**, then there is no known efficient algorithm that can solve it

- Is it possible for an efficient algorithm to solve an **NP hard** problem?



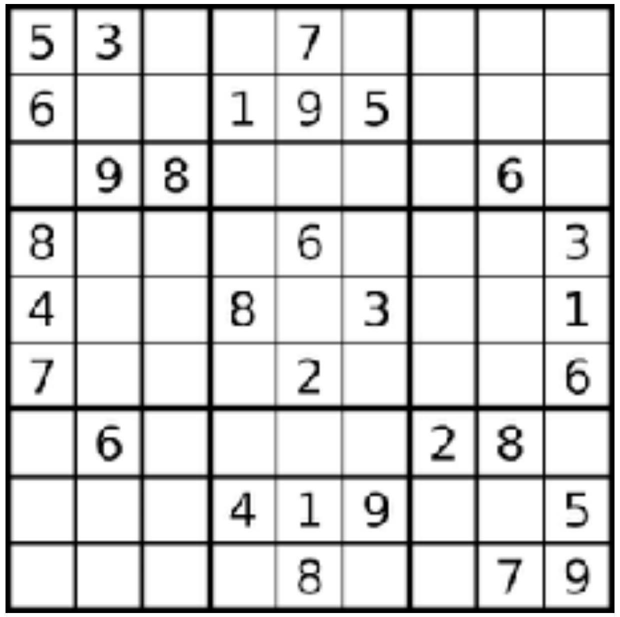
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

This is one of the biggest open questions in computing science

NP hardness

- If a problem is **NP hard**, then there is no known efficient algorithm that can solve it

- Is it possible for an efficient algorithm to solve an **NP hard** problem?



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

This is one of the biggest open questions in computing science

http://www.claymath.org/millennium-problems

$1,000,000 prize

Matching Problems

Matching problems

Matching problems

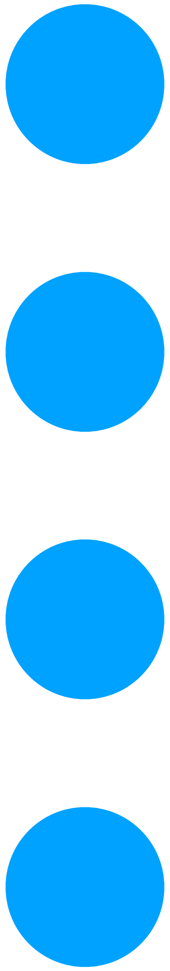
- Assign one group of things to another group of things

Matching problems

- Assign one group of things to another group of things
- Based on preferences

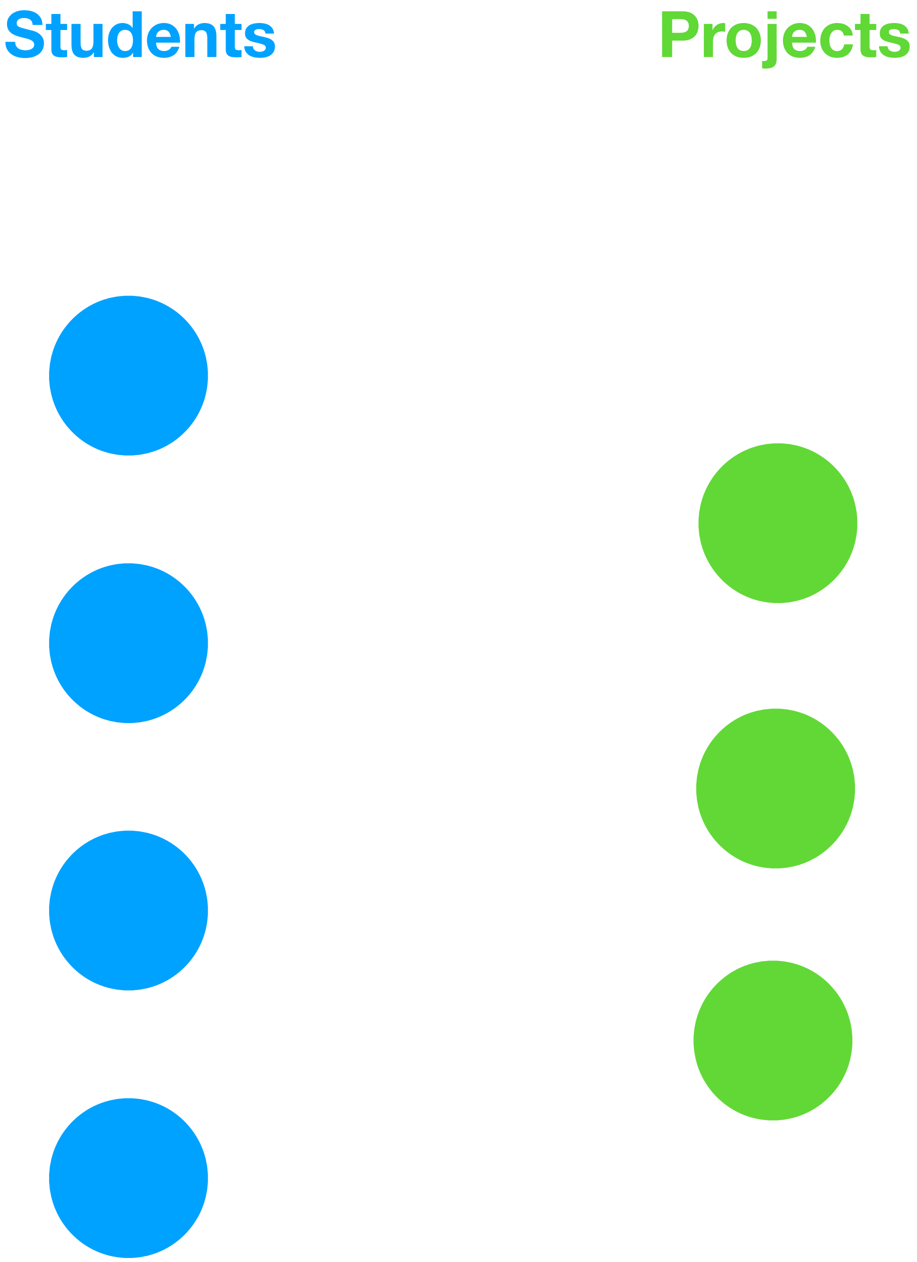
Student-project allocation problem

Student-project allocation problem

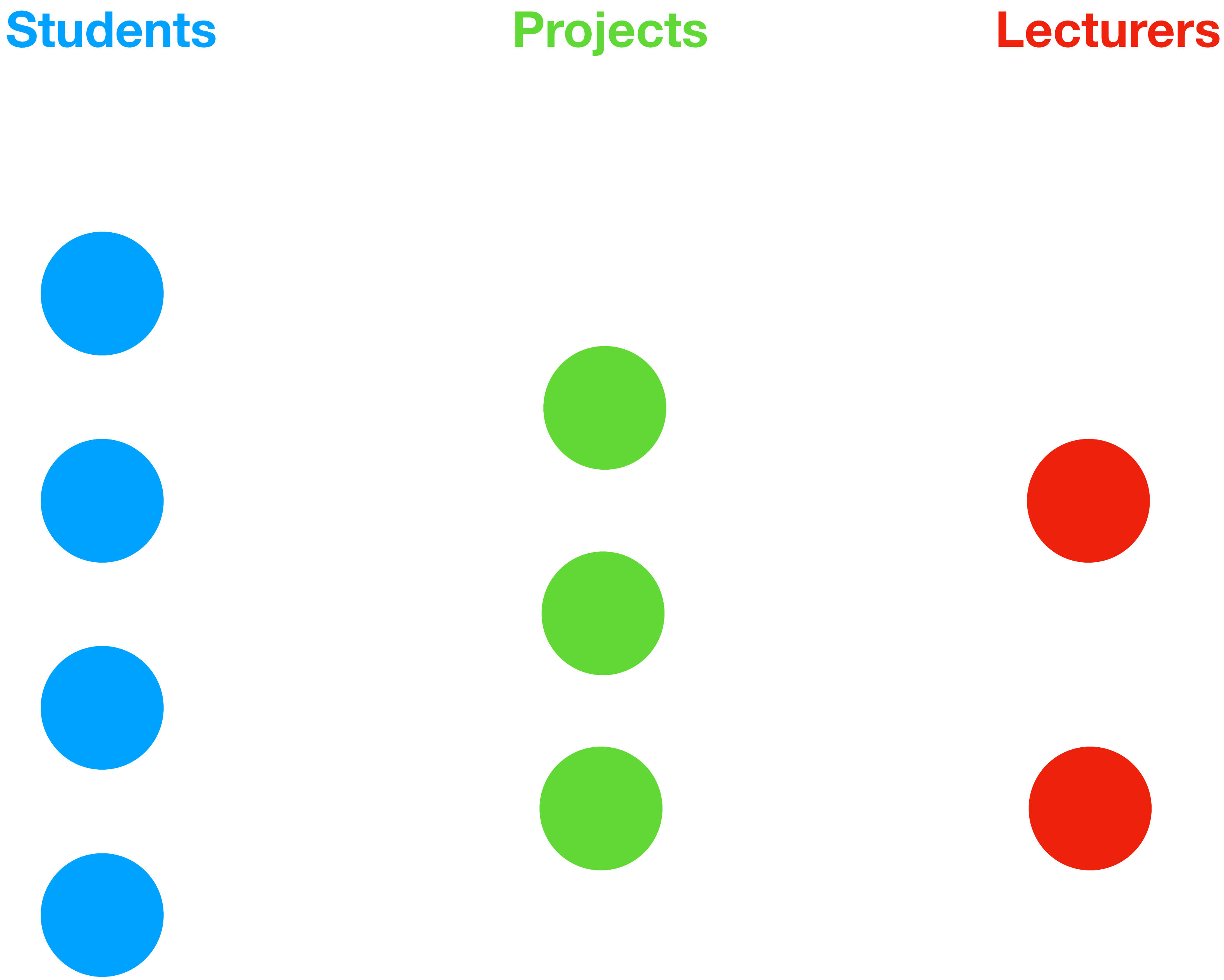


Students

Student-project allocation problem



Student-project allocation problem

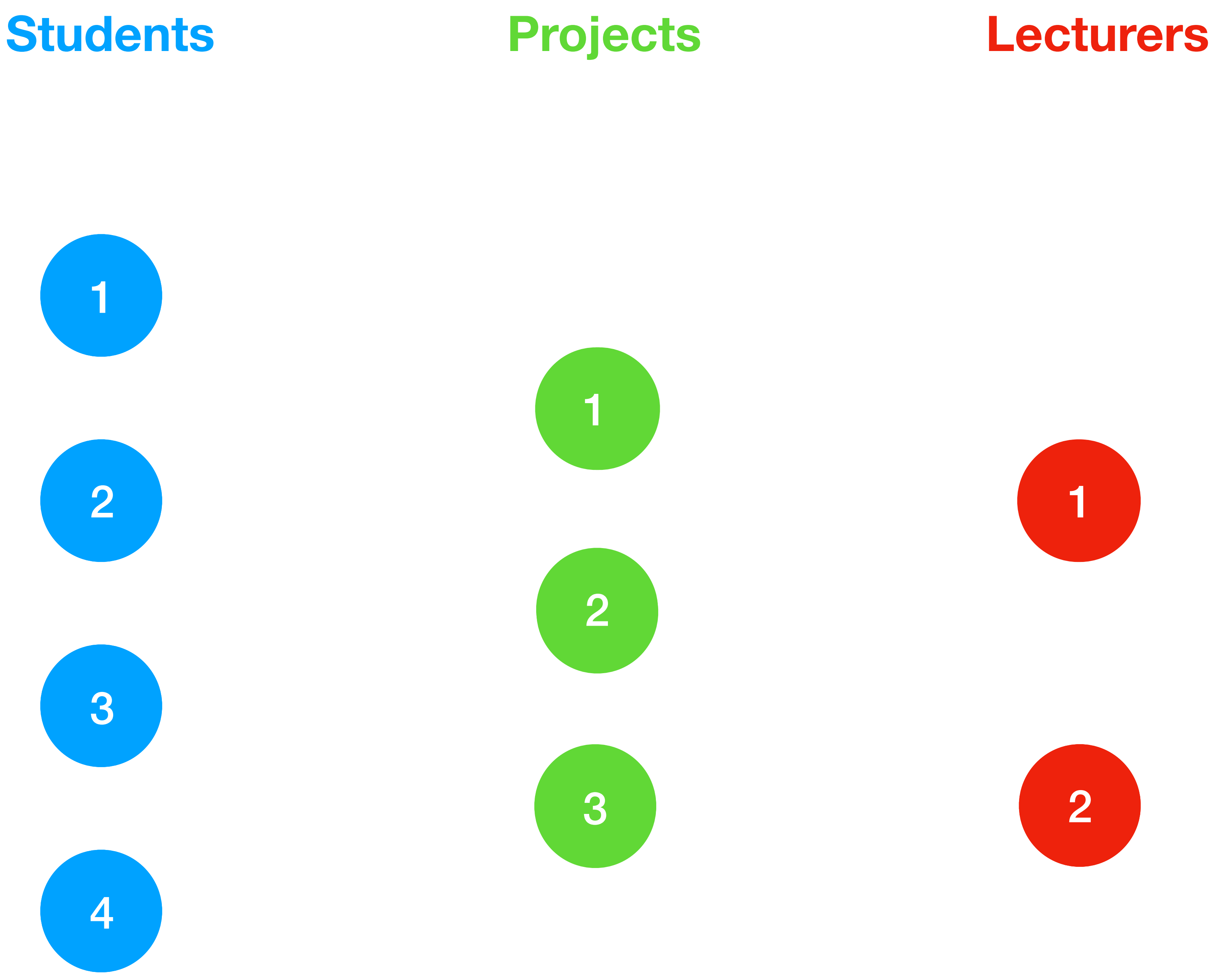


Students

Projects

Lecturers

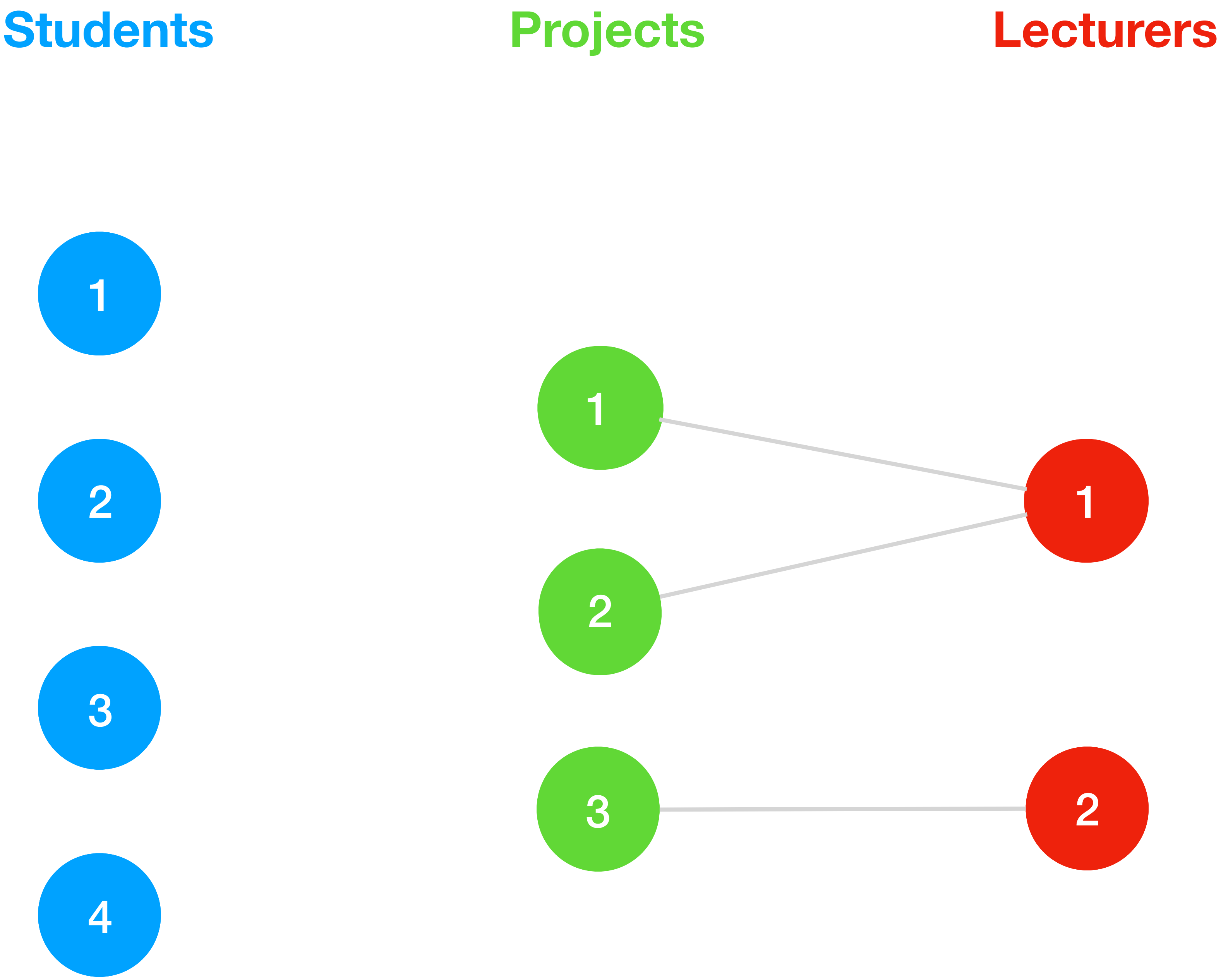
Student-project allocation problem


Students

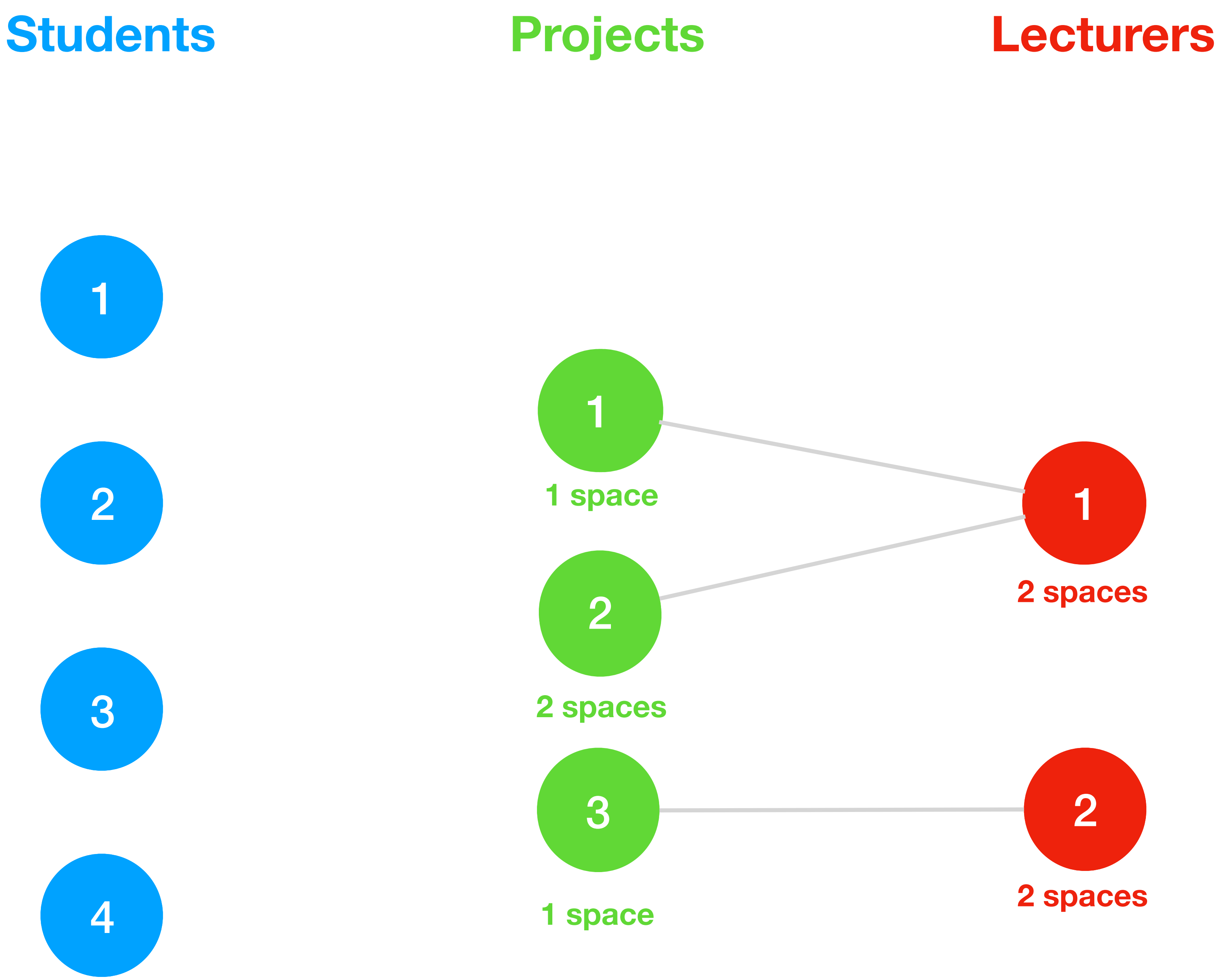
Projects

Lecturers

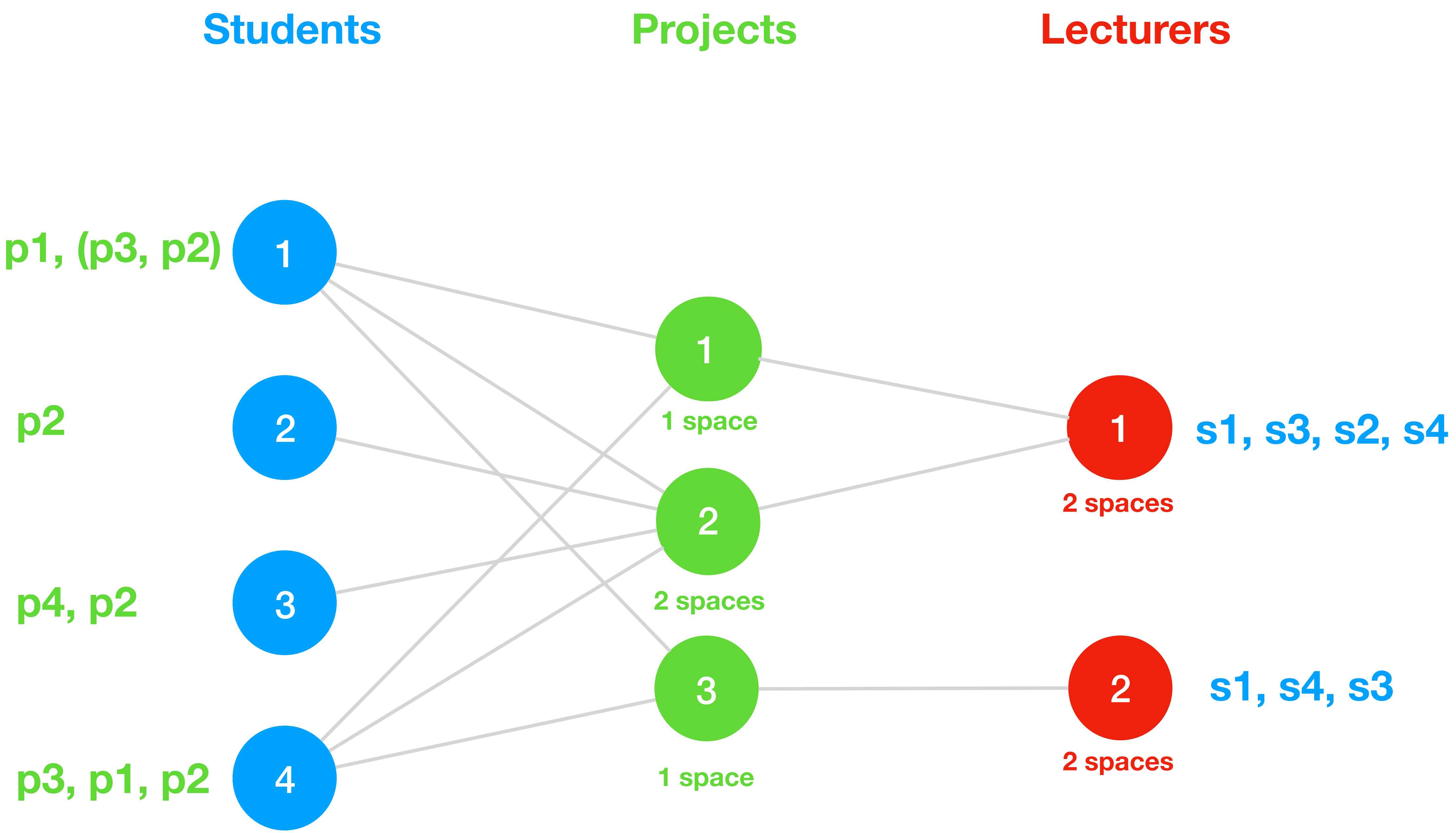
Student-project allocation problem



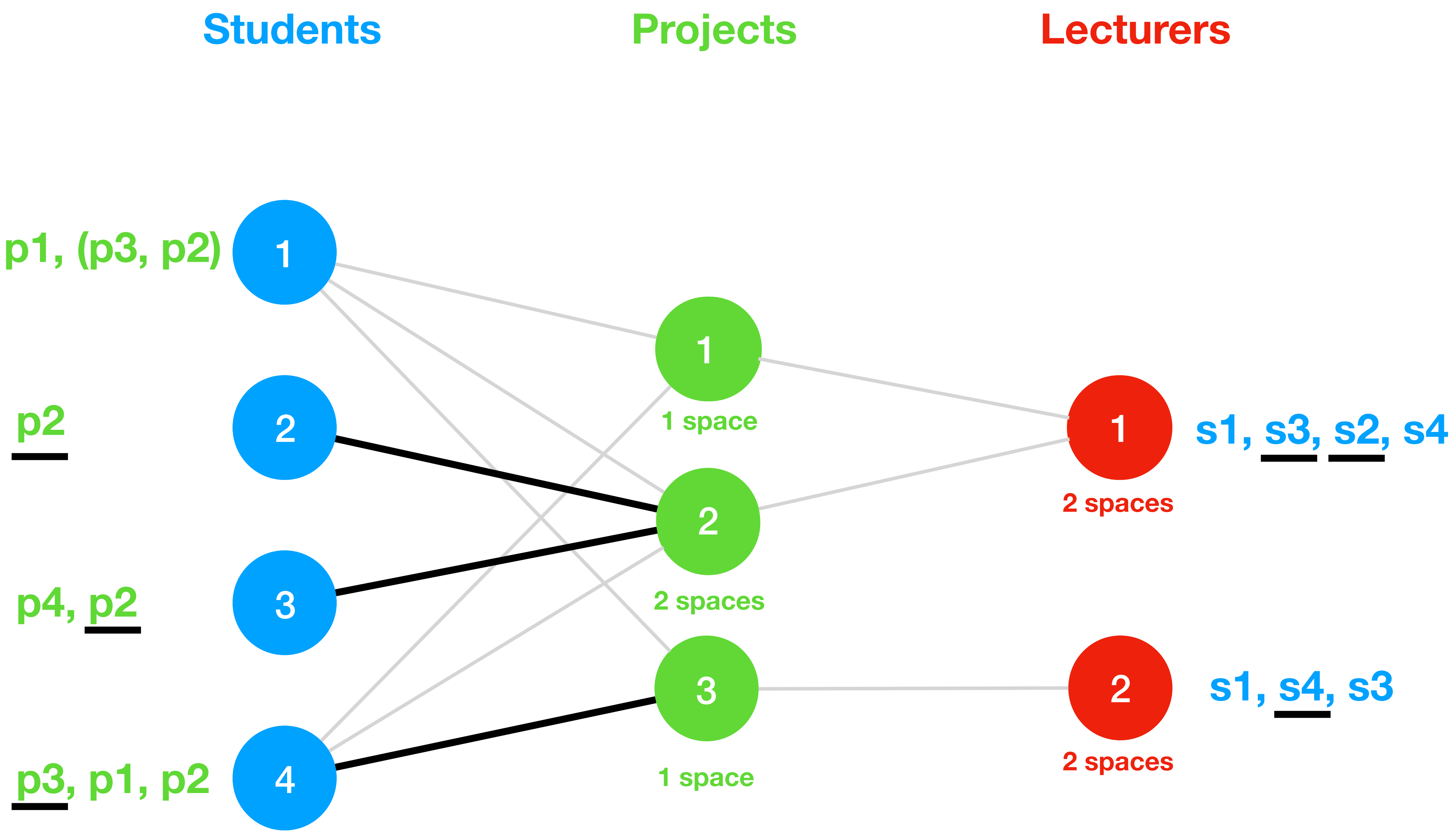
Student-project allocation problem



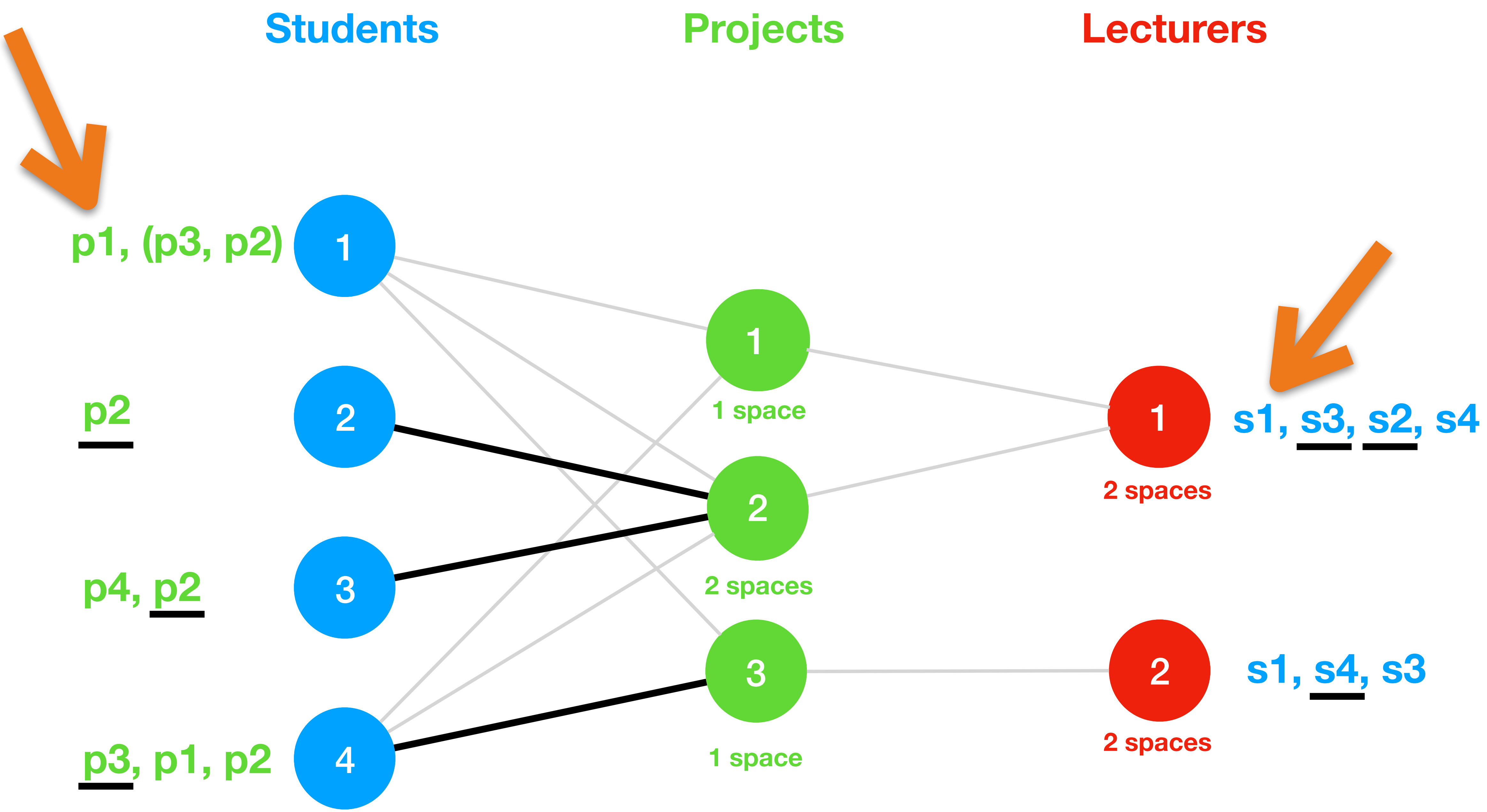
Student-project allocation problem



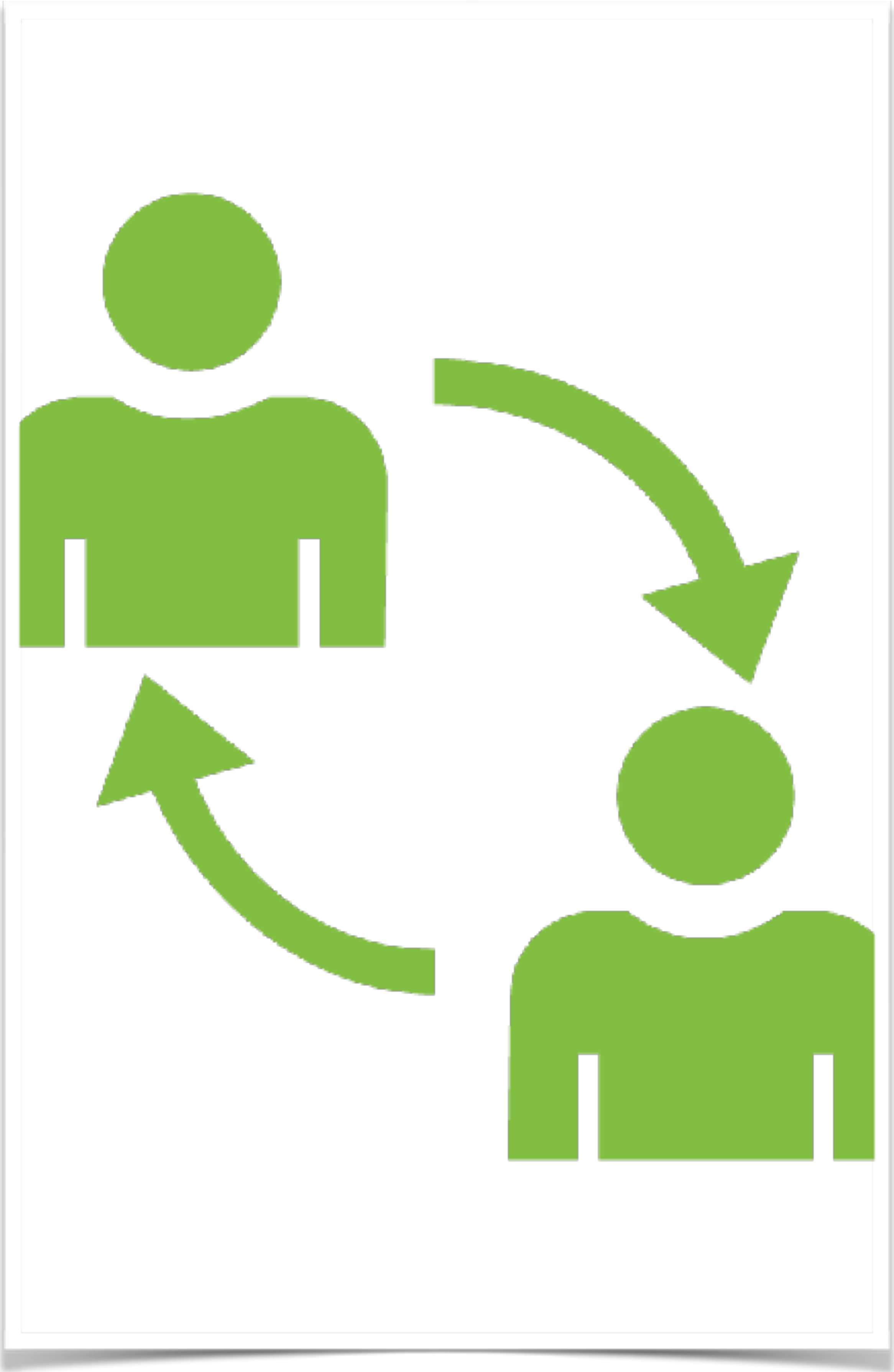
Student-project allocation problem



Student-project allocation problem

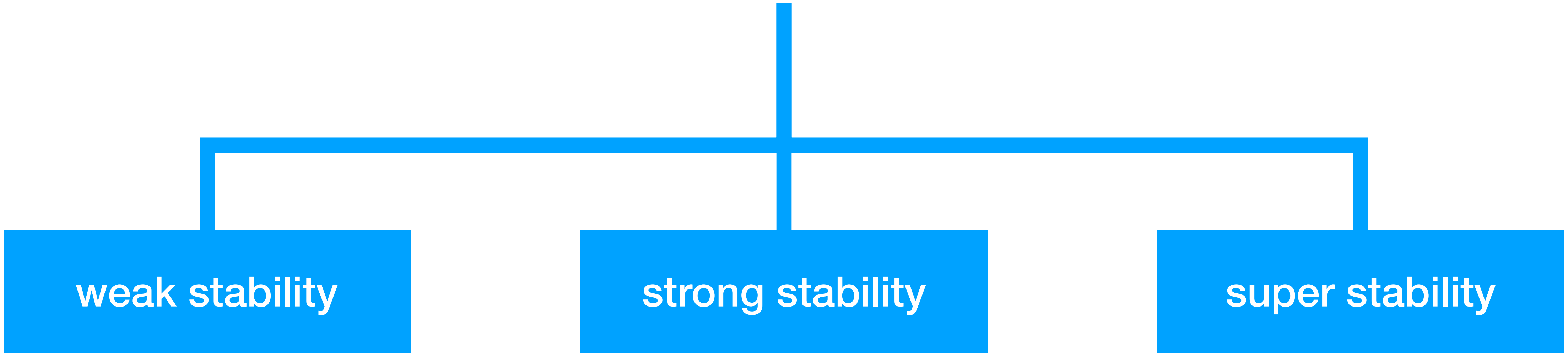


Stable matching



Stable matching

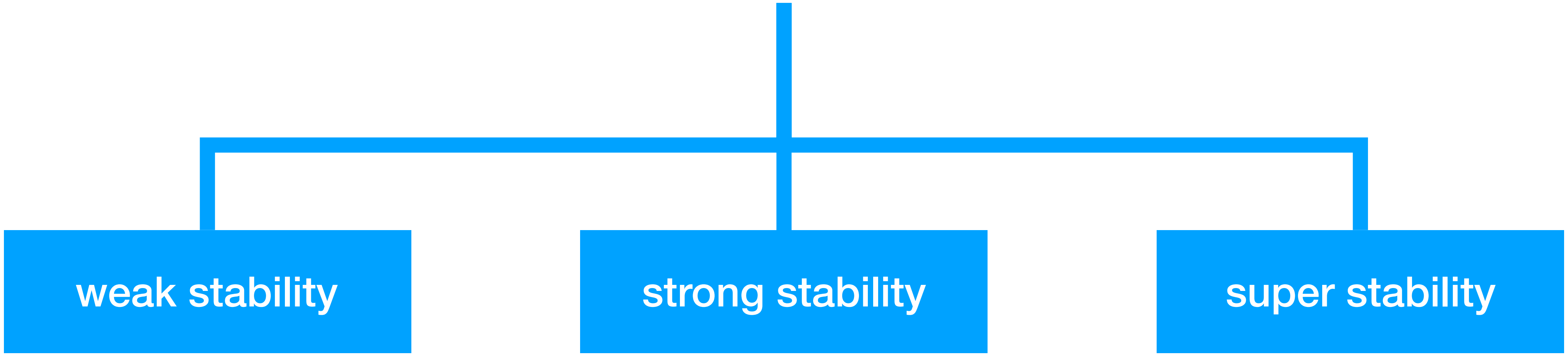
Stable matching


weak stability

strong stability

super stability

Stable matching



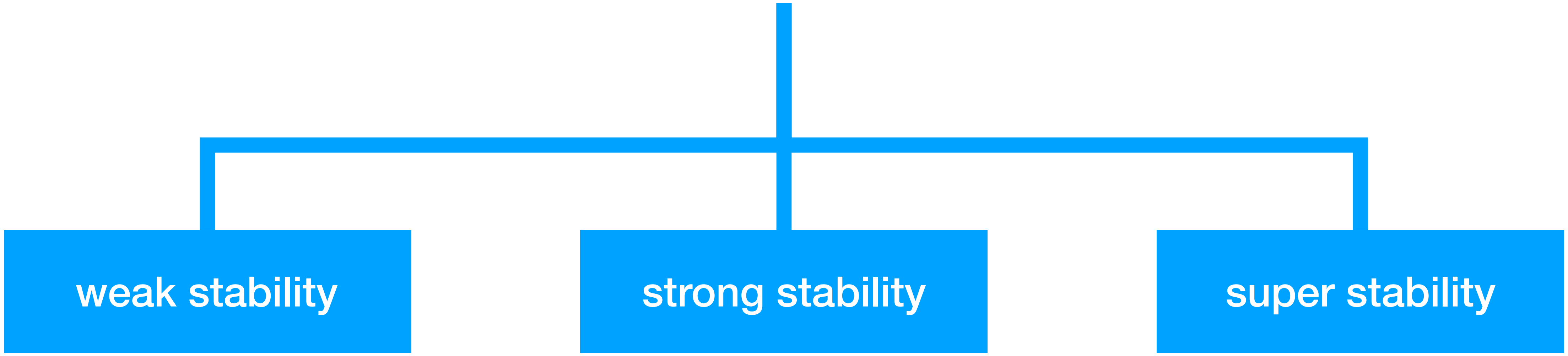
weak stability

strong stability

super stability

- A **stable matching** is a matching with no blocking pairs

Stable matching



weak stability

strong stability

super stability

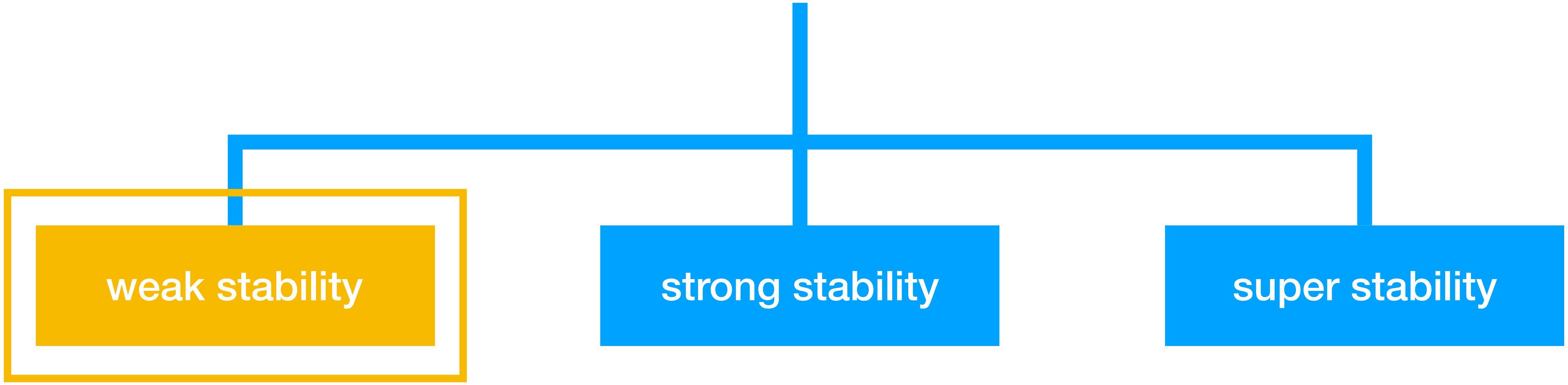
No Blocking pairs: both agents benefit

No Blocking Pairs: one agent benefits, the other is indifferent

No Blocking Pairs: neither agent is disadvantaged

- A **stable matching** is a matching with no blocking pairs

Stable matching



weak stability

strong stability

super stability

No Blocking pairs: both agents benefit

No Blocking Pairs: one agent benefits, the other is indifferent

No Blocking Pairs: neither agent is disadvantaged

- A **stable matching** is a matching with no blocking pairs

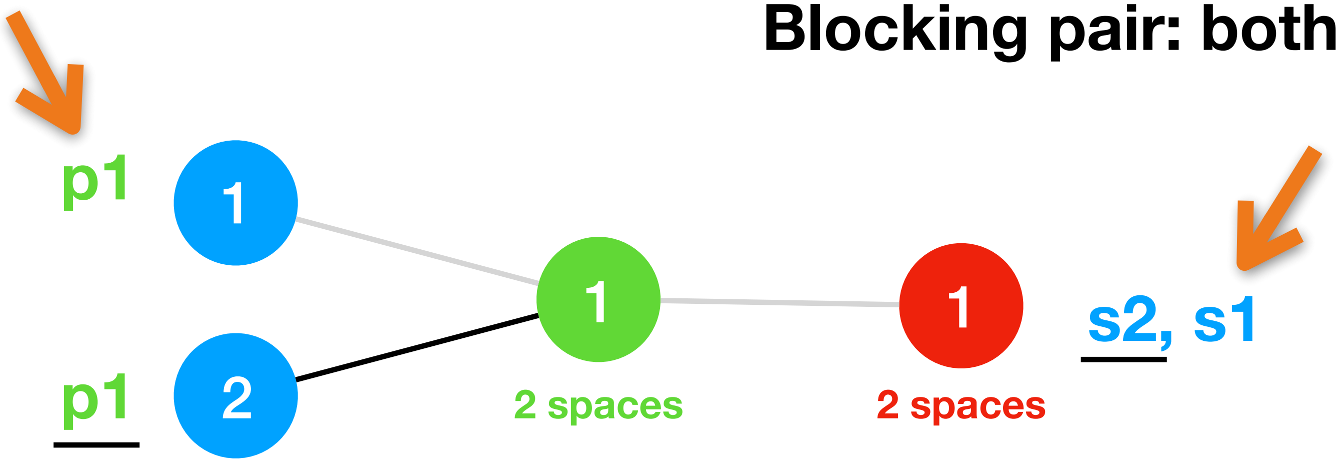
weak stability

weak stability

Blocking pair: both agents benefit

weak stability

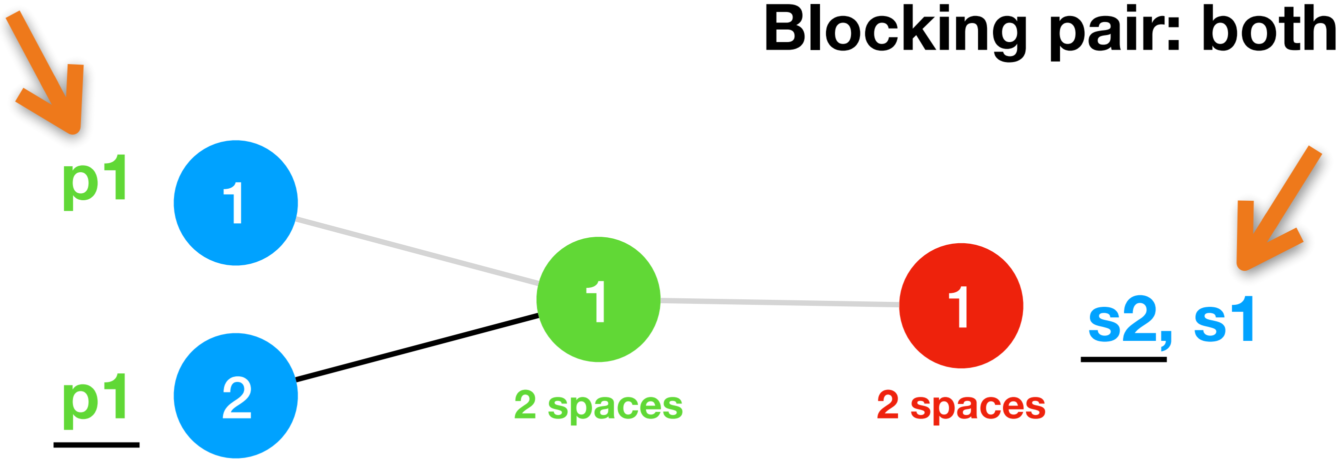
Blocking pair: both agents benefit



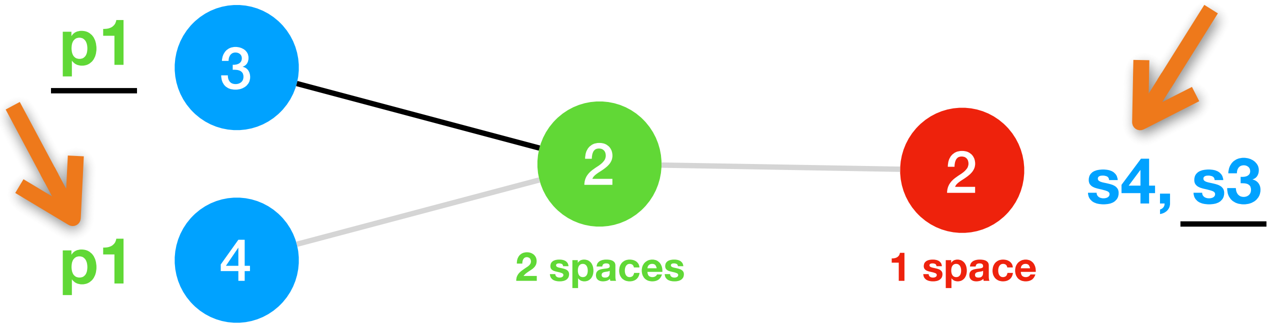
project and lecturer undersubscribed

weak stability

Blocking pair: both agents benefit



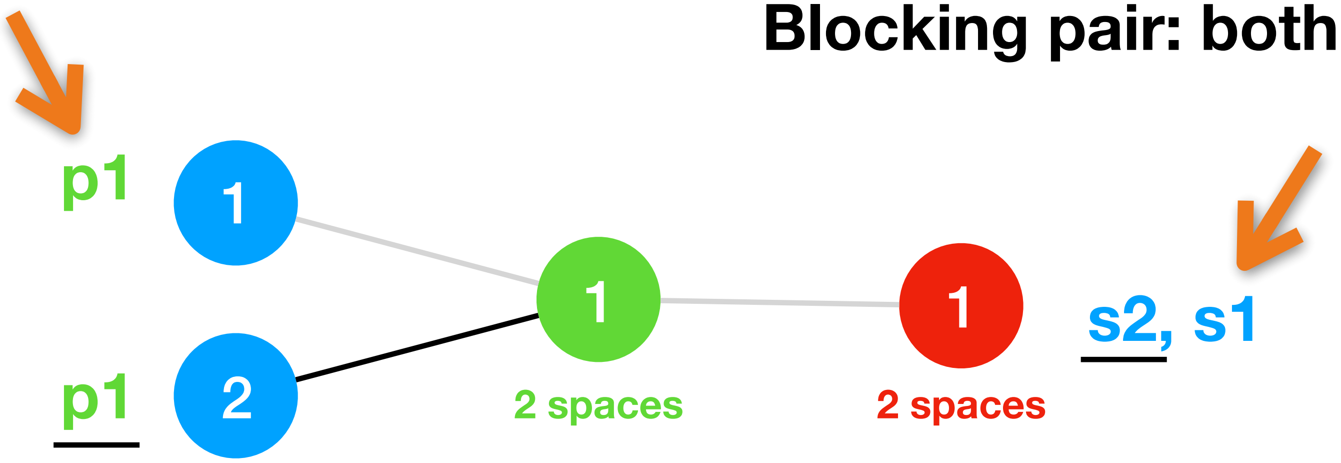
project and lecturer undersubscribed



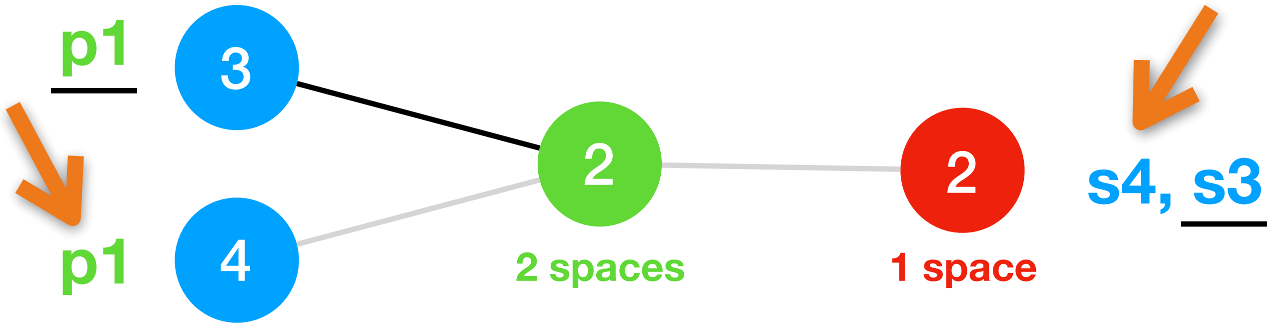
project undersubscribed, lecturer full

weak stability

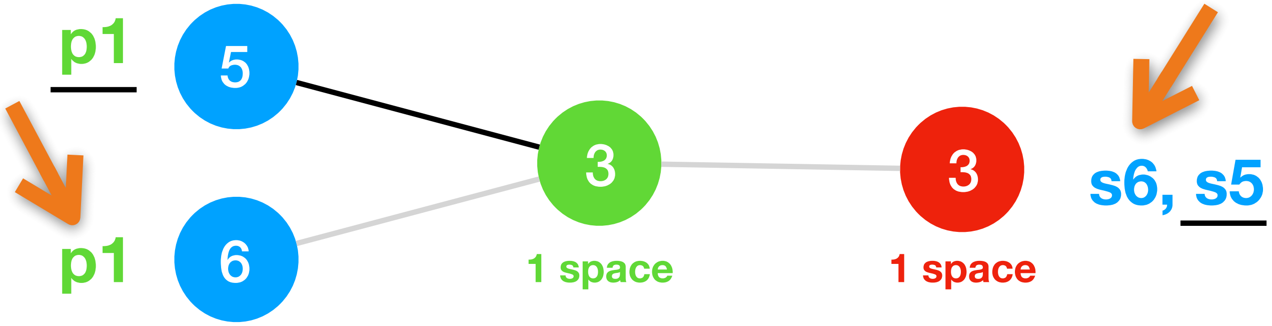
Blocking pair: both agents benefit



project and lecturer undersubscribed

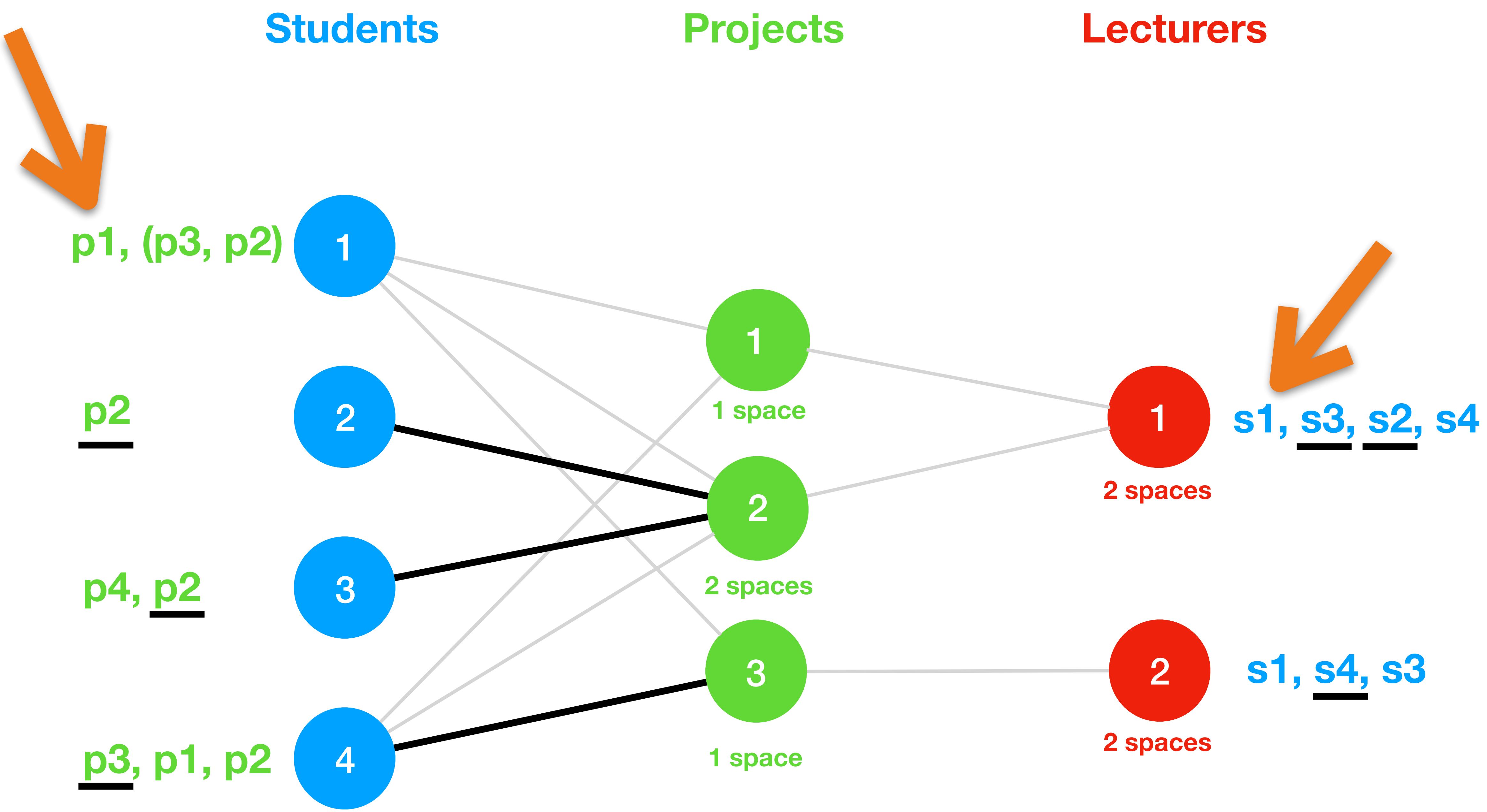


project undersubscribed, lecturer full



project full

Student-project allocation problem



Stable matchings

Stable matchings

- A **stable matching** is a matching with no blocking pairs

Stable matchings

- A **stable matching** is a matching with no blocking pairs
- When there are no ties in preference lists we can find a stable matching using an **efficient algorithm**. Also all stable matchings are the **same size**.

Two Algorithms for the Student Project Allocation Problem;
Journal of Discrete Algorithms; 2007; Abraham, Irving, Manlove

Stable matchings

- A **stable matching** is a matching with no blocking pairs
- When there are no ties in preference lists we can find a stable matching using an **efficient algorithm**. Also all stable matchings are the **same size**.
- When there are **ties in preference lists** we can find a stable matching using an **efficient algorithm** - but stable matchings are **different sizes**

Two Algorithms for the Student Project Allocation Problem;
Journal of Discrete Algorithms; 2007; Abraham, Irving, Manlove

Stable matchings

- A **stable matching** is a matching with no blocking pairs

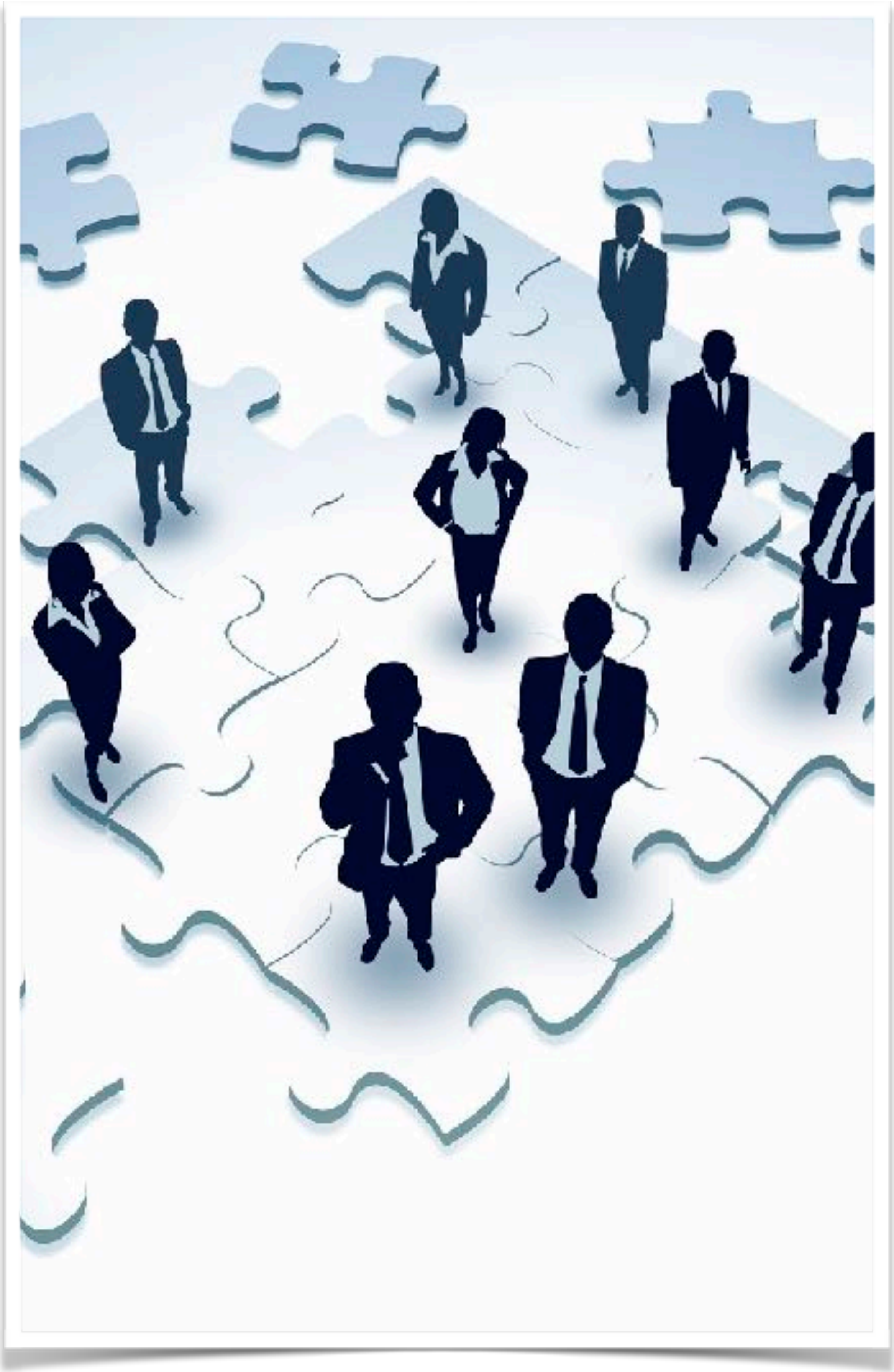
- When there are no ties in preference lists we can find a stable matching using an **efficient algorithm**. Also all stable matchings are the **same size**.

- When there are **ties in preference lists** we can find a stable matching using an **efficient algorithm** - but stable matchings are **different sizes**

Two Algorithms for the Student Project Allocation Problem;
Journal of Discrete Algorithms; 2007; Abraham, Irving, Manlove

Gives us new questions

Maximum sized stable matching



Maximum sized stable matchings

Maximum sized stable matchings

Finding a maximum sized stable matching is **NP-hard**. What can we do?

Maximum sized stable matchings

Finding a maximum sized stable matching is **NP-hard**. What can we do?

- Build an exhaustive search algorithm inefficient

Maximum sized stable matchings

Finding a maximum sized stable matching is **NP-hard**. What can we do?

- Build an exhaustive search algorithm **inefficient**
- Find an efficient algorithm for an NP-hard problem **tricky** **$1,000,000 prize**

Maximum sized stable matchings

Finding a maximum sized stable matching is **NP-hard**. What can we do?

- Build an exhaustive search algorithm inefficient
- Find an efficient algorithm for an NP-hard problem tricky $1,000,000 prize
- Integer programming (uses inefficient algorithms)

Maximum sized stable matchings

Finding a maximum sized stable matching is **NP-hard**. What can we do?

- Build an exhaustive search algorithm inefficient
- Find an efficient algorithm for an NP-hard problem tricky
  $1,000,000 prize
- Integer programming (uses inefficient algorithms)

Maximum sized stable matchings

Finding a maximum sized stable matching is **NP-hard**. What can we do?

- Build an exhaustive search algorithm inefficient
- Find an efficient algorithm for an NP-hard problem tricky $1,000,000 prize
- Integer programming (uses inefficient algorithms)
- Approximation algorithms (poly time)

Integer Programming

Integer Programming

Integer Programming

- Uses software that can optimise or solve a problem when it is given an Integer Programming model as input

Integer Programming

- Uses software that can optimise or solve a problem when it is given an Integer Programming model as input
- Uses exponential-time algorithms

Integer Programming

- Uses software that can optimise or solve a problem when it is given an Integer Programming model as input
- Uses exponential-time algorithms
- But this is only in a worst-case scenario. Integer Programming algorithms are optimised to work quickly in many cases

Integer Programming

- Uses software that can optimise or solve a problem when it is given an Integer Programming model as input

- Uses exponential-time algorithms

- But this is only in a worst-case scenario. Integer Programming algorithms are optimised to work quickly in many cases

- An Integer programming model has been built to find a **maximum stable matching**

Approximation algorithm

Approximation algorithm

- Instead of finding a **maximum stable matching**

Approximation algorithm

- Instead of finding a **maximum stable matching**
- Look for a stable matching that is at least x% of the size of the **maximum stable matching**

Approximation algorithm

- Instead of finding a **maximum stable matching**
- Look for a stable matching that is at least x% of the size of the **maximum stable matching**
- A trade off

Approximation algorithm

- Instead of finding a **maximum stable matching**
- Look for a stable matching that is at least x% of the size of the **maximum stable matching**
- A trade off

negative: aren't solving to optimality

Approximation algorithm

- Instead of finding a **maximum stable matching**
- Look for a stable matching that is at least x% of the size of the **maximum stable matching**
- A trade off

negative: aren't solving to optimality

positive: **efficient** algorithm

Creating approximation algorithm

Creating approximation algorithm

- An approximation algorithm exists for a simpler problem where lecturers aren't involved

Linear Time Local Approximation Algorithm for Maximum Stable Marriage; Algorithms; 2013; Kiraly

Creating approximation algorithm

- An approximation algorithm exists for a simpler problem where lecturers aren't involved

Linear Time Local Approximation Algorithm for Maximum Stable Marriage; Algorithms; 2013; Kiraly

  - Can I just convert my problem and use this? No!

Creating approximation algorithm

- An approximation algorithm exists for a simpler problem where lecturers aren't involved

Linear Time Local Approximation Algorithm for Maximum Stable Marriage; Algorithms; 2013; Kiraly

  - Can I just convert my problem and use this? No!

- Had to create a new 3/2 approximation algorithm

Creating approximation algorithm

- An approximation algorithm exists for a simpler problem where lecturers aren't involved

Linear Time Local Approximation Algorithm for Maximum Stable Marriage; Algorithms; 2013; Kiraly

  - Can I just convert my problem and use this? No!

- Had to create a new 3/2 approximation algorithm

  - Lecturers added a lot of complications

Creating approximation algorithm

- An approximation algorithm exists for a simpler problem where lecturers aren't involved

Linear Time Local Approximation Algorithm for Maximum Stable Marriage; Algorithms; 2013; Kiraly

    - Can I just convert my problem and use this? No!

- Had to create a new 3/2 approximation algorithm
    - Lecturers added a lot of complications
    - Proved that this algorithm is efficient (polynomial-time) and correct (results in a stable matching at least 2/3 the size of a **maximum stable matching**)

Approximation algorithm high-level look

Approximation algorithm high-level look

Students (who are not already assigned) apply in turn to their favourite project on their preference list. Assume student **s** applies to project **p**.

Approximation algorithm high-level look

Students (who are not already assigned) apply in turn to their favourite project on their preference list. Assume student **s** applies to project **p**.

- if **p** and **l** (the lecturer of **p**) are undersubscribed then we add (**s**,**p**) to our matching

Approximation algorithm high-level look

Students (who are not already assigned) apply in turn to their favourite project on their preference list. Assume student **s** applies to project **p**.

- if **p** and **l** (the lecturer of **p**) are undersubscribed then we add (**s**,**p**) to our matching

- if either **p** or **l** are full then we need to check whether (**s**,**p**) should replace an *existing* pair in the matching

Approximation algorithm high-level look

Students (who are not already assigned) apply in turn to their favourite project on their preference list. Assume student **s** applies to project **p**.

- if **p** and **l** (the lecturer of **p**) are undersubscribed then we add (**s**,**p**) to our matching

- if either **p** or **l** are full then we need to check whether (**s**,**p**) should replace an *existing* pair in the matching

- if there is no chance for **s** to assign to **p** then **s** will remove **p** from their preference list (and will now apply to their next favourite)

Approximation algorithm high-level look

Students (who are not already assigned) apply in turn to their favourite project on their preference list. Assume student **s** applies to project **p**.

- if **p** and **l** (the lecturer of **p**) are undersubscribed then we add (**s**,**p**) to our matching
- if either **p** or **l** are full then we need to check whether (**s**,**p**) should replace an *existing* pair in the matching
- if there is no chance for **s** to assign to **p** then **s** will remove **p** from their preference list (and will now apply to their next favourite)
- Students iterate twice through their preference list

How good is the approximation algorithm?

How good is the approximation algorithm?

- Experiments! 100s of thousands of instances with varying parameters. Ran on approximation algorithm and integer program.

How good is the approximation algorithm?

- Experiments! 100s of thousands of instances with varying parameters. Ran on approximation algorithm and integer program.
- Correctness testing

How good is the approximation algorithm?

- Experiments! 100s of thousands of instances with varying parameters. Ran on approximation algorithm and integer program.
- Correctness testing
- Does the approximation algorithm stick to 2/3 the size of optimal? Or do we get close to maximum?

How good is the approximation algorithm?

- Experiments! 100s of thousands of instances with varying parameters. Ran on approximation algorithm and integer program.

- Correctness testing

- Does the approximation algorithm stick to 2/3 the size of optimal? Or do we get close to maximum?

- Somewhere in between, but closer to maximum

How good is the approximation algorithm?

- Experiments! 100s of thousands of instances with varying parameters. Ran on approximation algorithm and integer program.

- Correctness testing

- Does the approximation algorithm stick to 2/3 the size of optimal? Or do we get close to maximum?

- Somewhere in between, but closer to maximum

- Much faster than using the integer program

How good is the approximation algorithm?

- Experiments! 100s of thousands of instances with varying parameters. Ran on approximation algorithm and integer program.
- Correctness testing
- Does the approximation algorithm stick to 2/3 the size of optimal? Or do we get close to maximum?
- Somewhere in between, but closer to maximum
- Much faster than using the integer program
- So is it worth using?

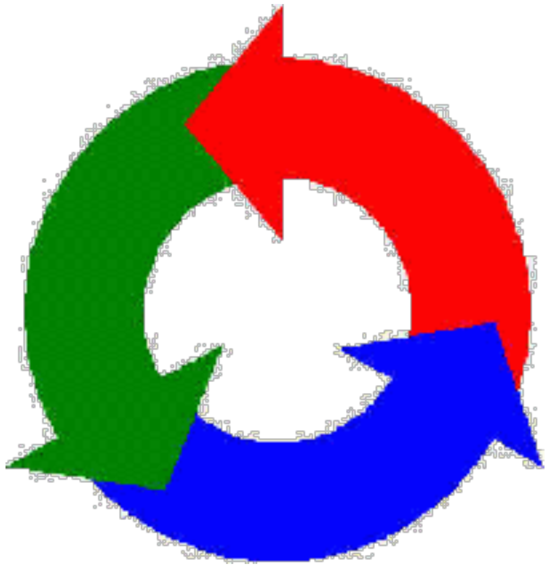
Future Work: coalitions

Future Work: coalitions

- group of several students and lecturers

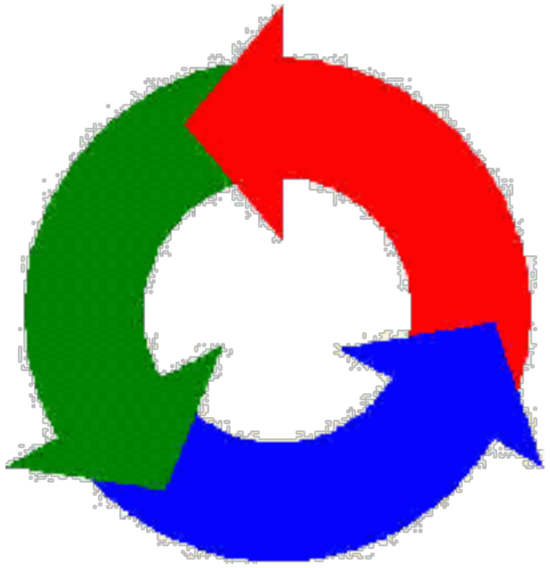
Future Work: coalitions

- group of several students and lecturers
- permute their assignments



Future Work: coalitions

- group of several students and lecturers
- permute their assignments
- some or all get a better outcome



Summary

Summary

- Algorithms

Summary

- Algorithms
- Student-project allocation problem

Summary

- Algorithms
- Student-project allocation problem
- Finding a maximum stable matching

Summary

- Algorithms
- Student-project allocation problem
- Finding a maximum stable matching
    - Integer programming

Summary

- Algorithms
- Student-project allocation problem
- Finding a maximum stable matching
    - Integer programming
    - Approximation algorithm

University of Glasgow

VIA VERITAS VITA

Thank you

f.cooper.1@research.gla.ac.uk

http://www.dcs.gla.ac.uk/~francesc/