
University of Glasgow

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

Frances Cooper

Supervisor: Dr David Manlove

# Outline

# Outline

- Matching problems

# Outline

- Matching problems
- Maximum sized stable matching

# Outline

- Matching problems
- Maximum sized stable matching
- Integer programming

# Outline

- Matching problems
- Maximum sized stable matching
- Integer programming
- Approximation algorithm

# Outline

- Matching problems
- Maximum sized stable matching
    - Integer programming
    - Approximation algorithm
- Future work

# Matching Problems

# Matching Problems

- Assign one set of entities to another set of entities

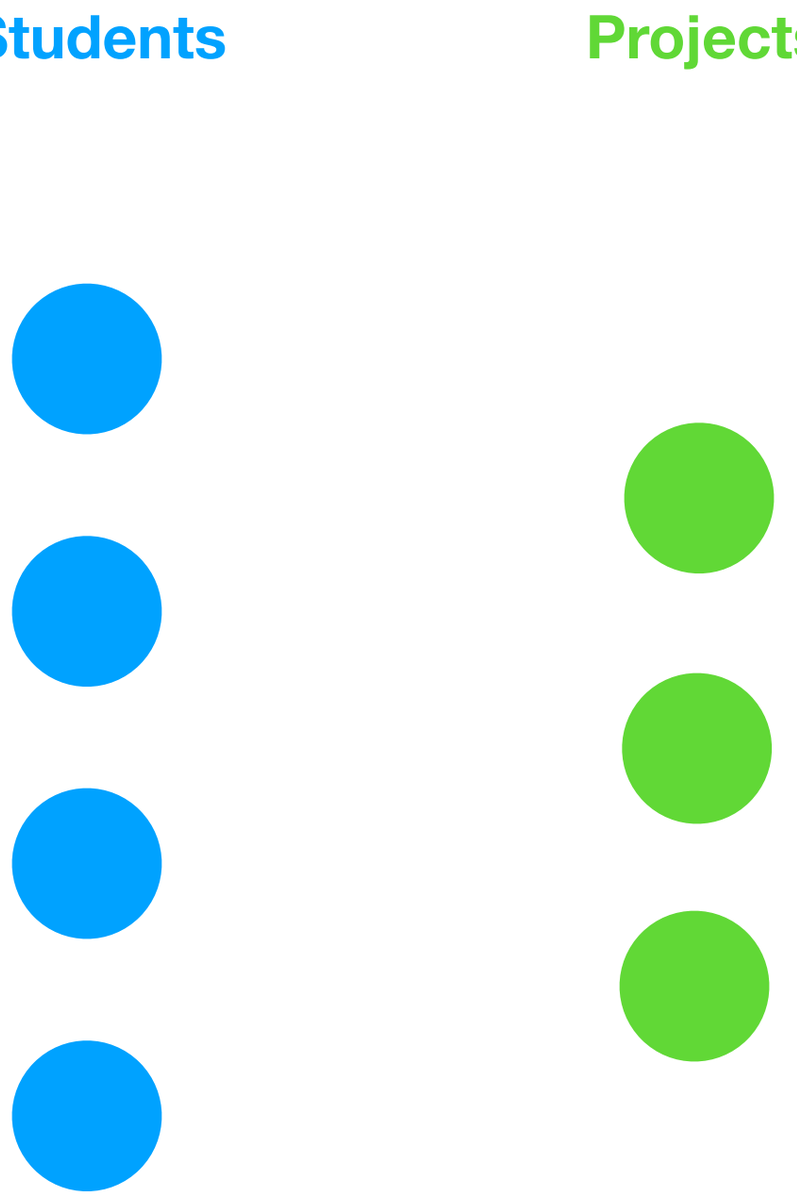

# Matching Problems

- Assign one set of entities to another set of entities
- Based on preferences and capacities

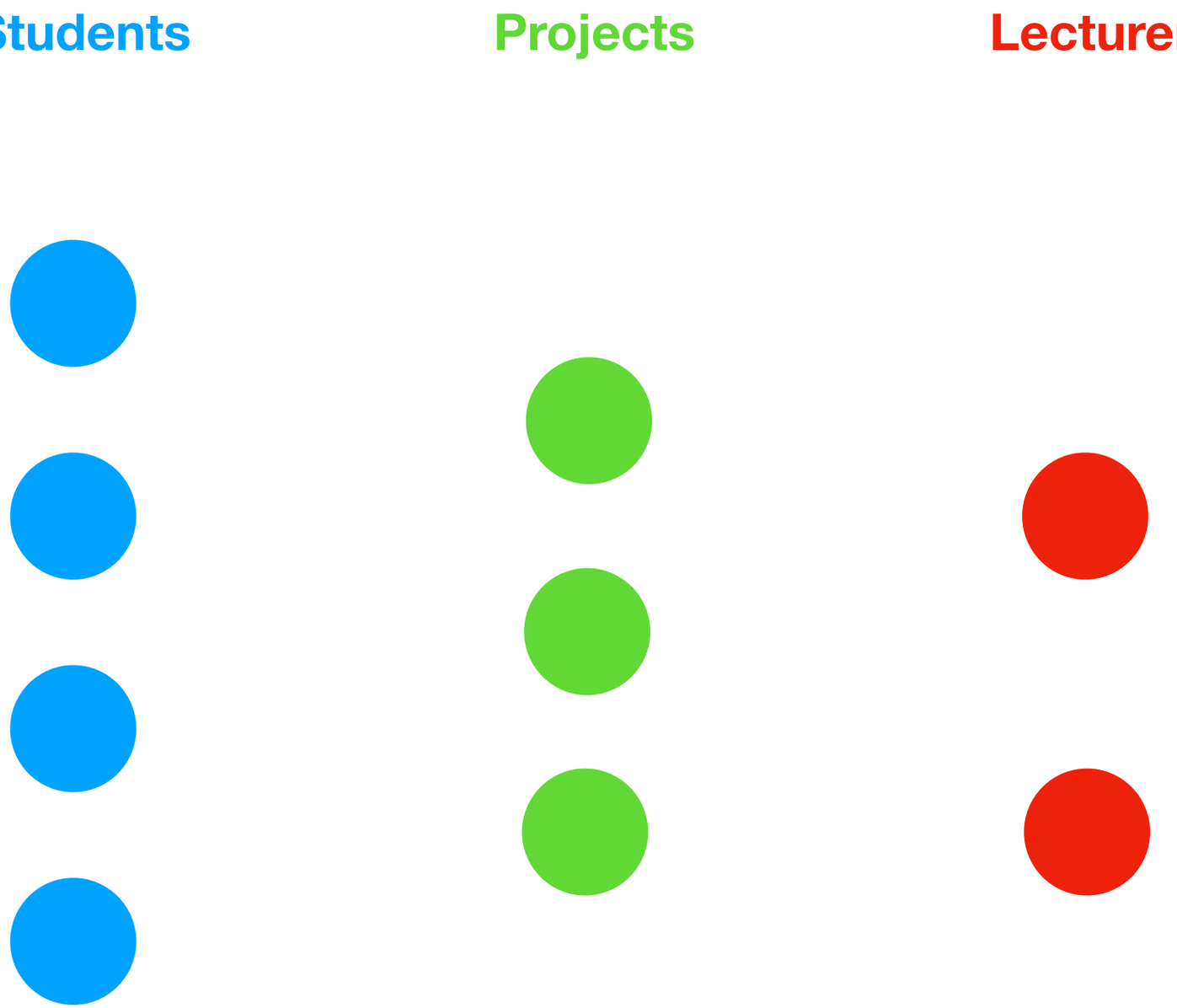
# Student-project allocation problem (SPA-ST)

# Student-project allocation problem (SPA-ST)

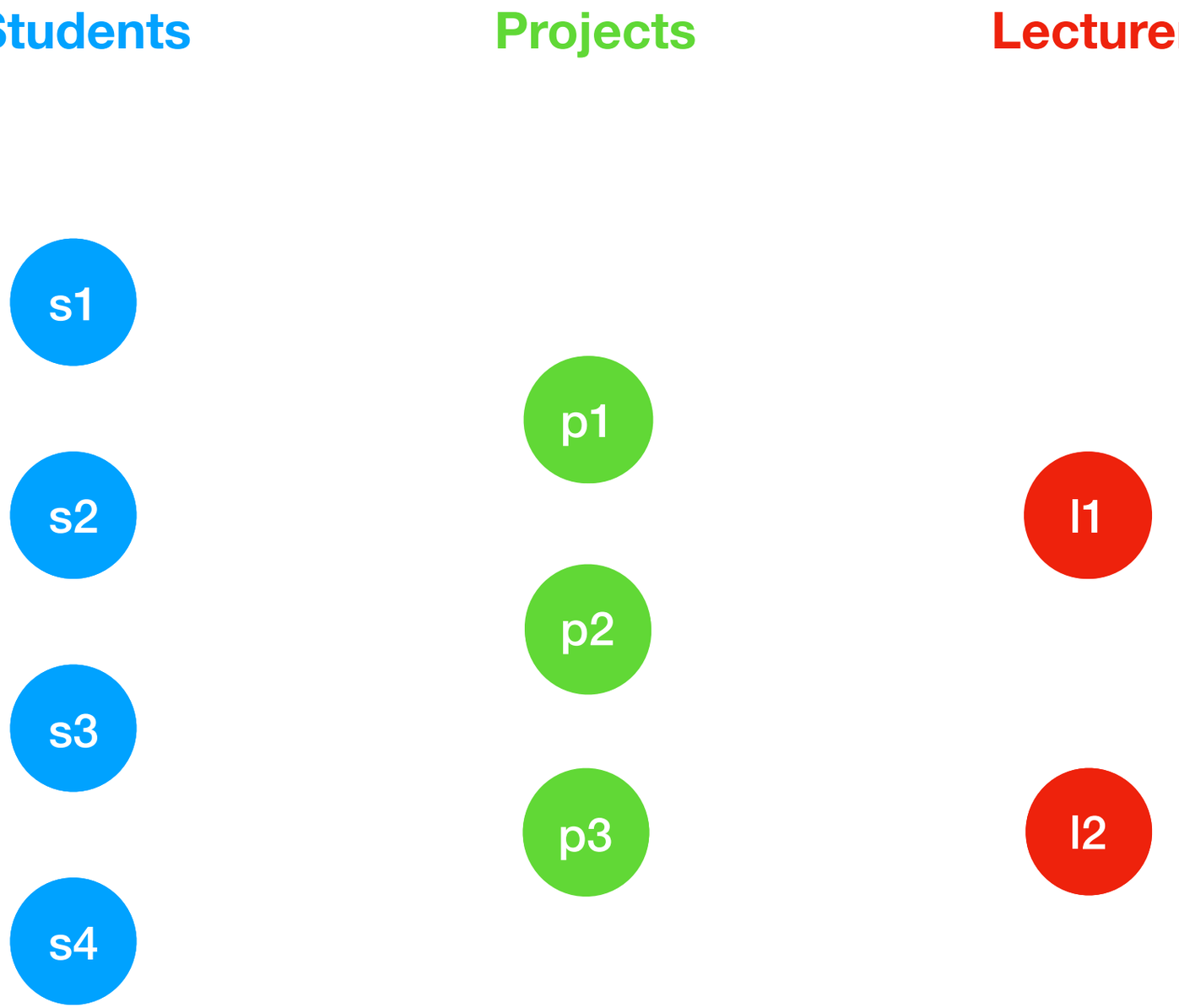
# Student-project allocation problem (SPA-ST)



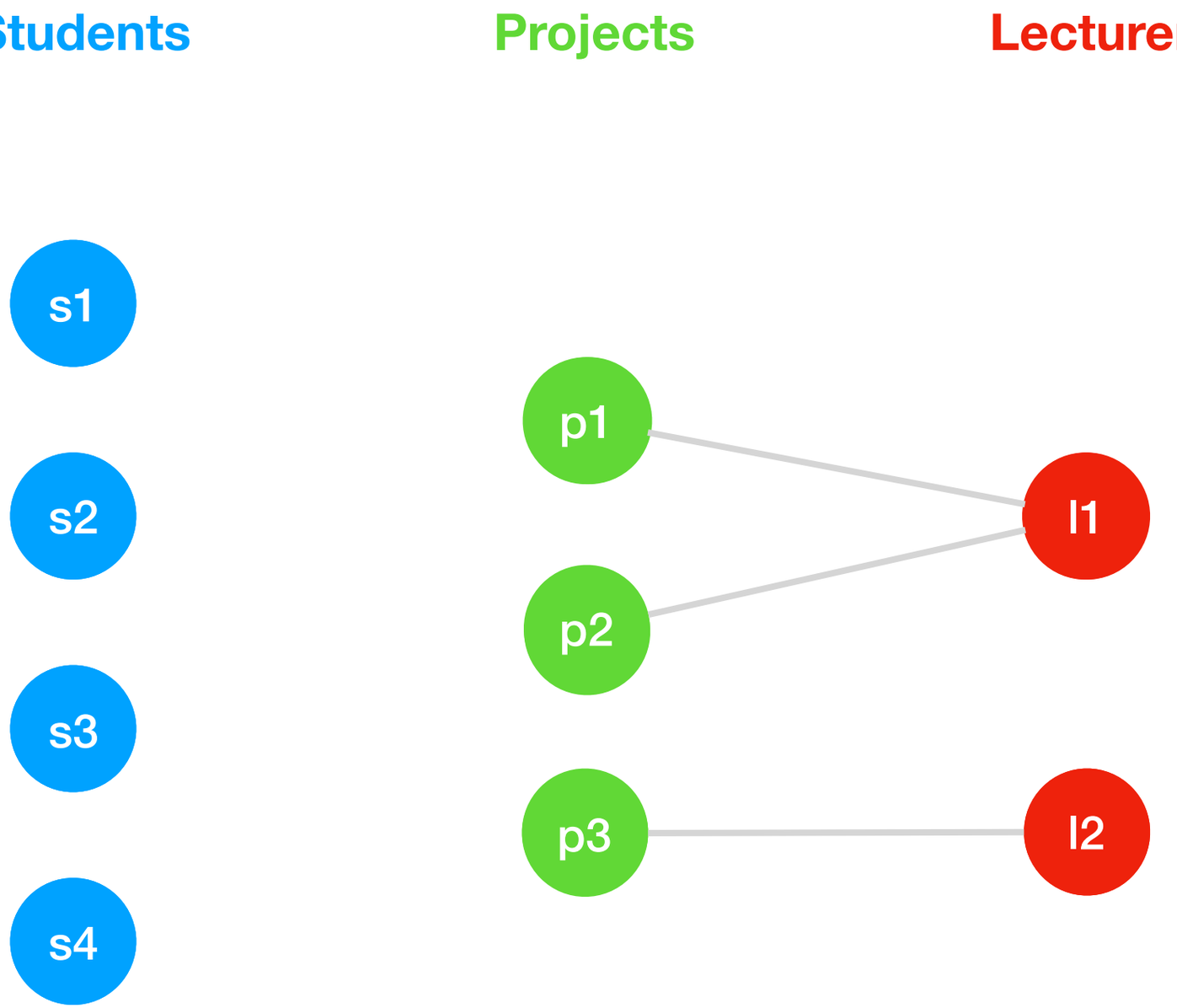
# Student-project allocation problem (SPA-ST)



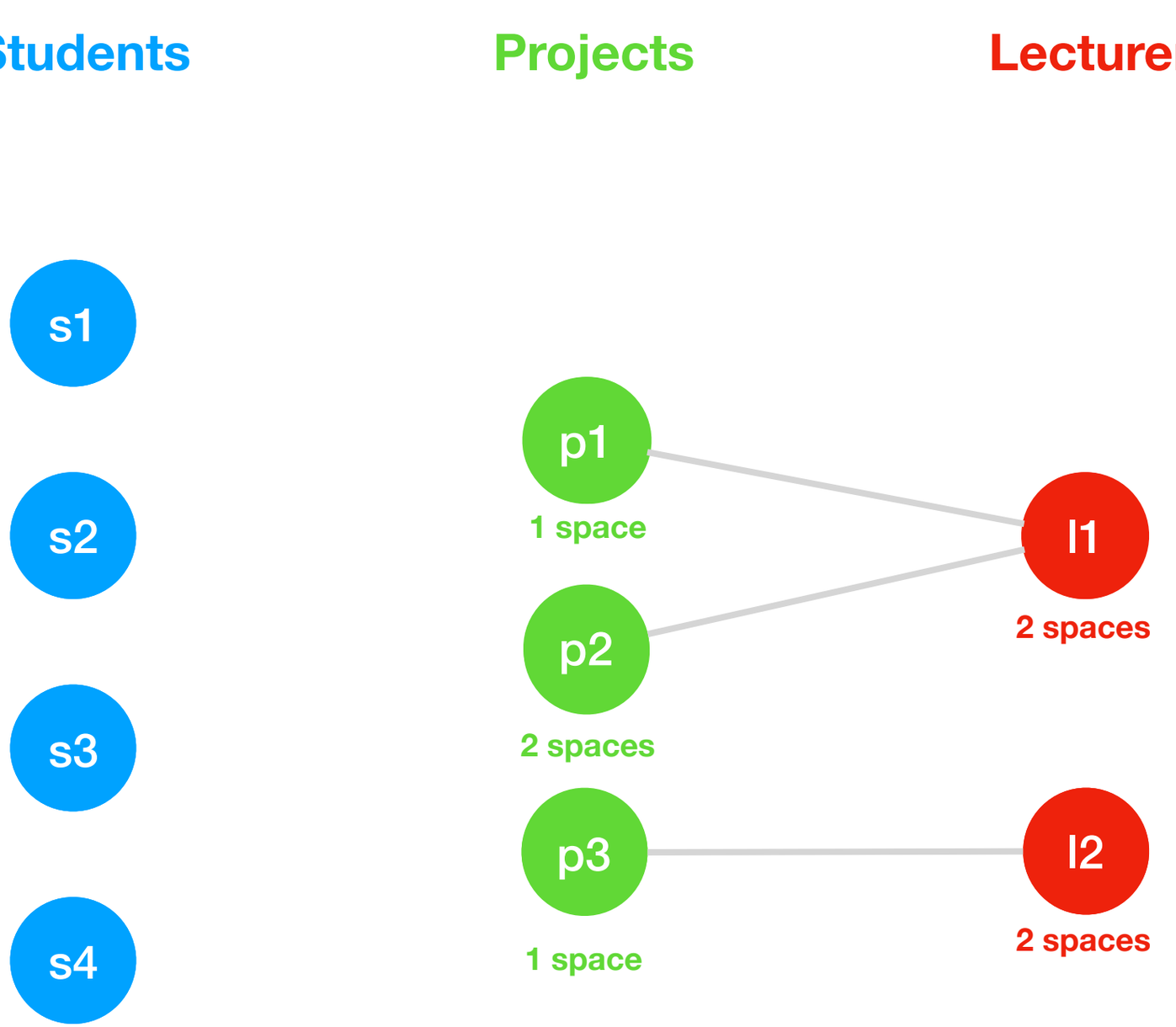
# Student-project allocation problem (SPA-ST)



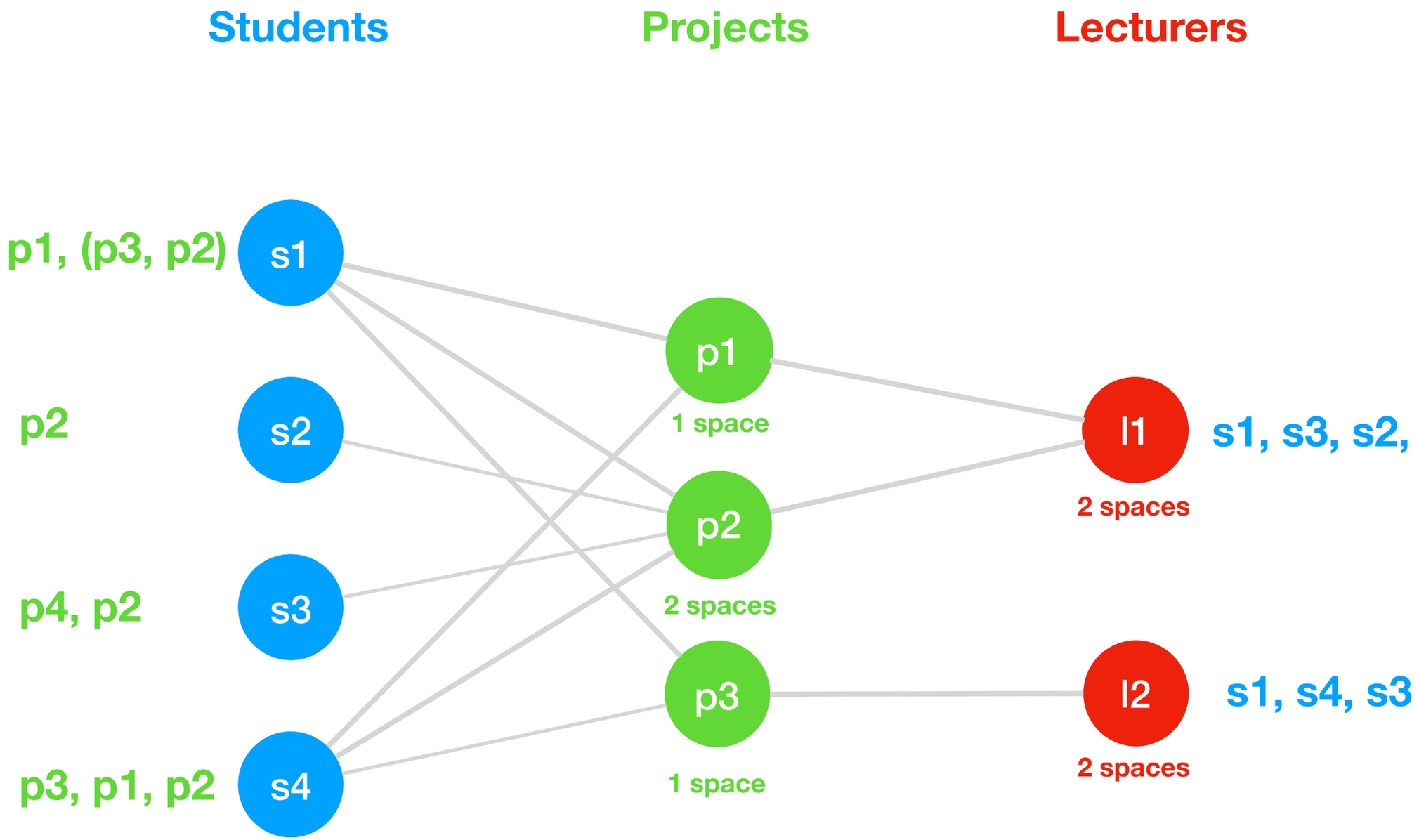
# Student-project allocation problem (SPA-ST)



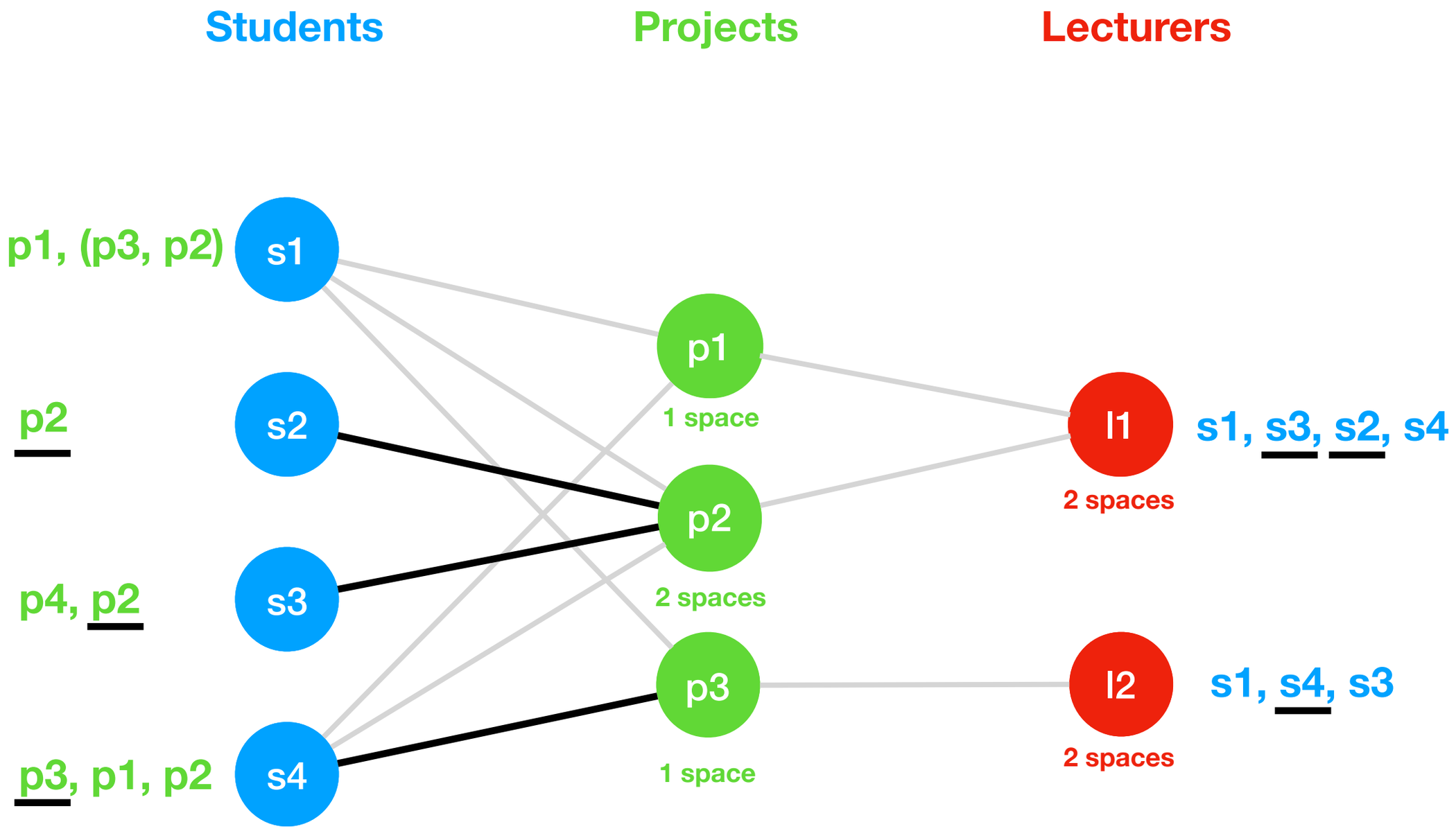
# Student-project allocation problem (SPA-ST)



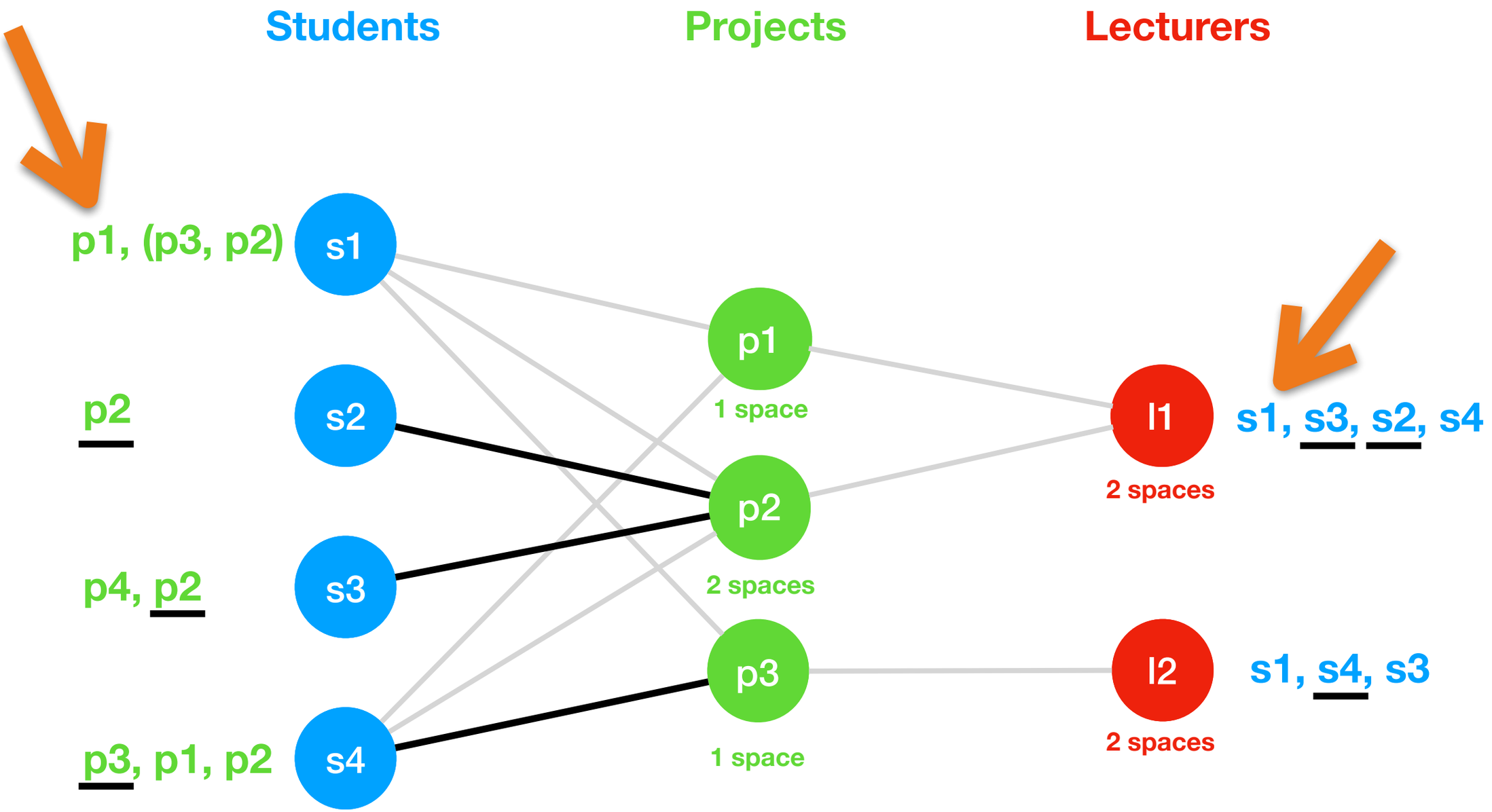
# Student-project allocation problem (SPA-ST)



# Student-project allocation problem (SPA-ST)

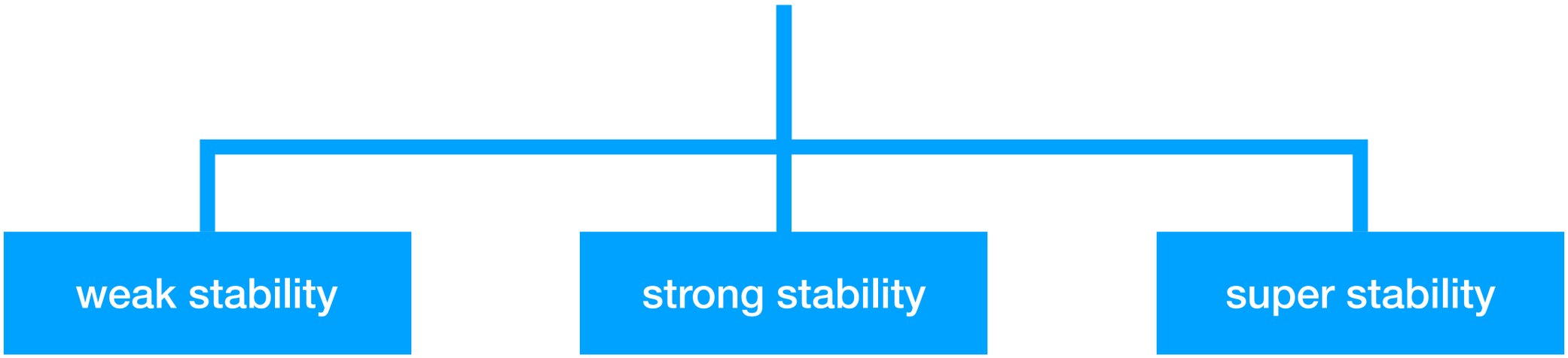


# Student-project allocation problem (SPA-ST)

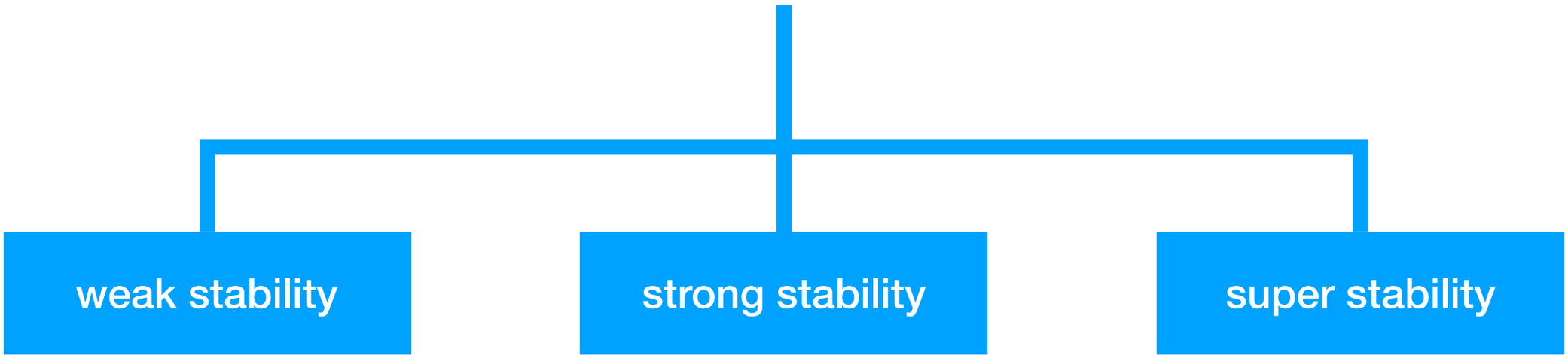


# Stable matching

# Stable matching

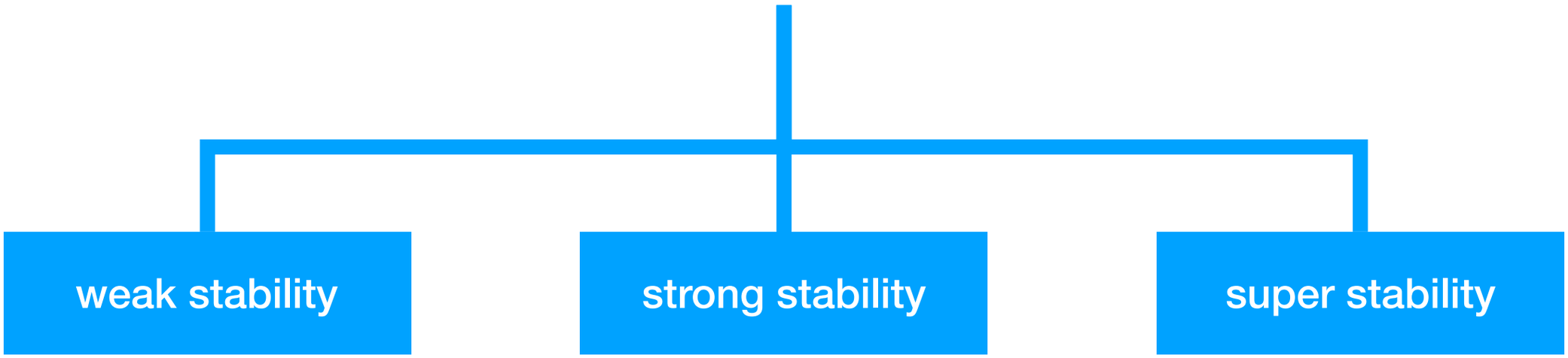


# Stable matching




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

# Stable matching



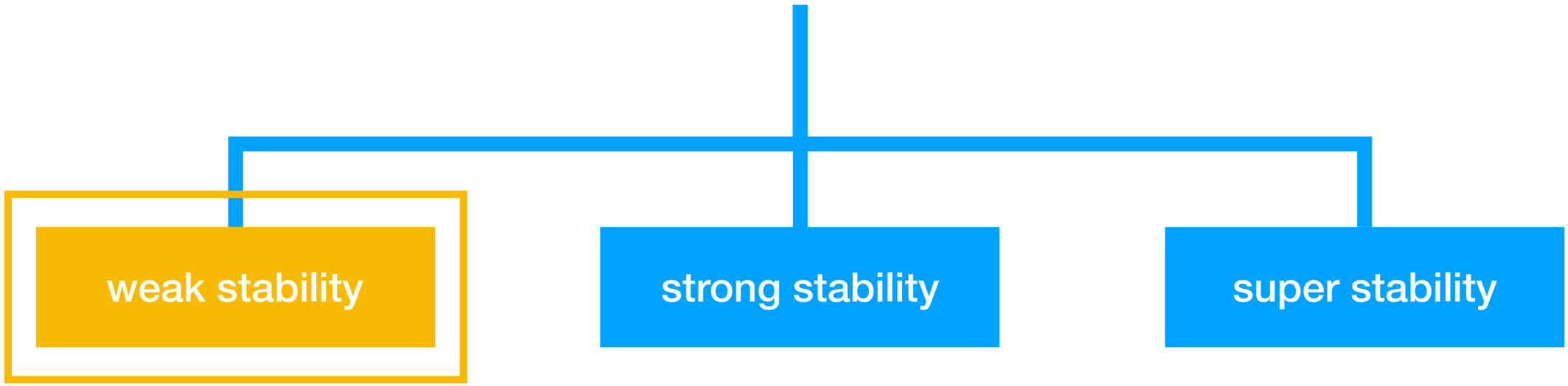

**No Blocking pairs: both agents are better off**

**No Blocking Pairs: one agent is better off, the other is no worse off**

**No Blocking Pairs: neither agent is worse off**

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

# Stable matching




**No Blocking pairs: both agents are better off**

**No Blocking Pairs: one agent is better off, the other is no worse off**

**No Blocking Pairs: neither agent is worse off**

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

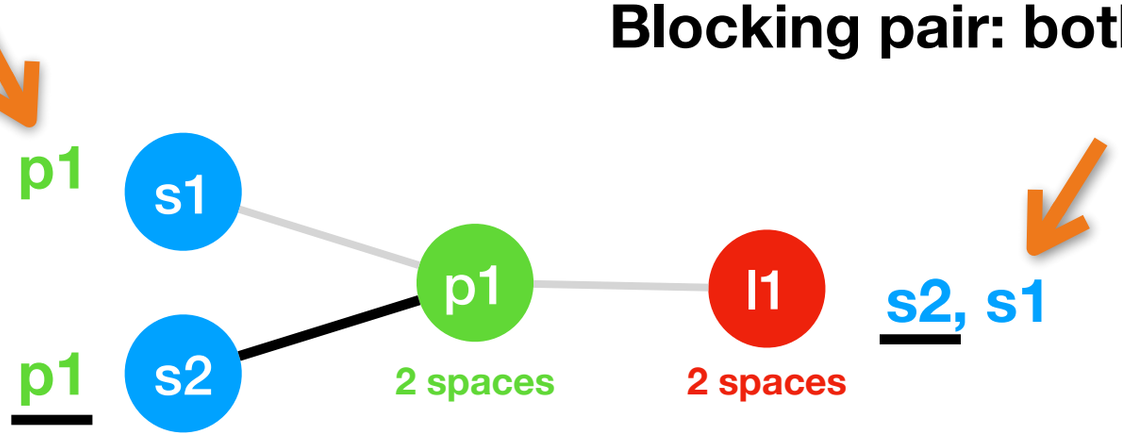
# weak stability

# weak stability

**Blocking pair: both agents are better off**

# weak stability

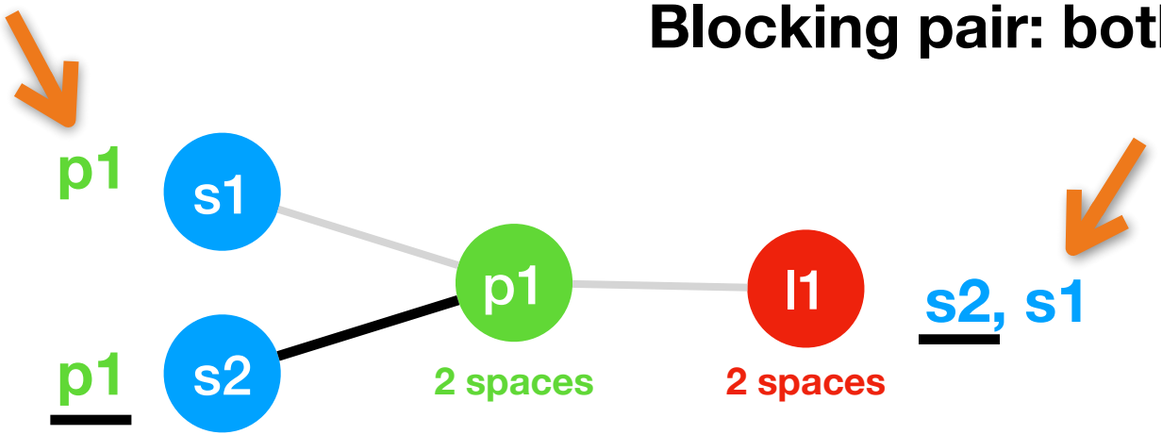
**Blocking pair: both agents are better off**



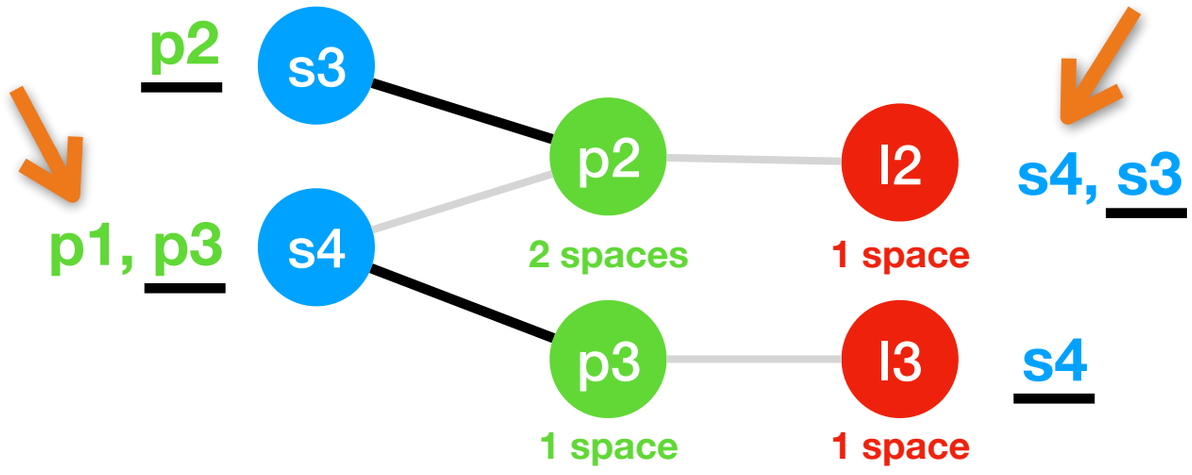

project and lecturer undersubscribed

# weak stability

**Blocking pair: both agents are better off**



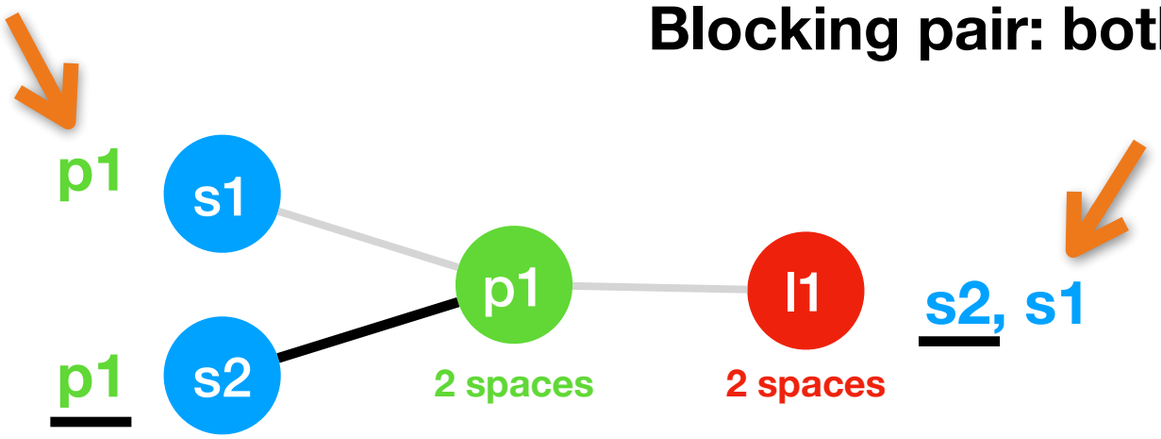

project and lecturer undersubscribed



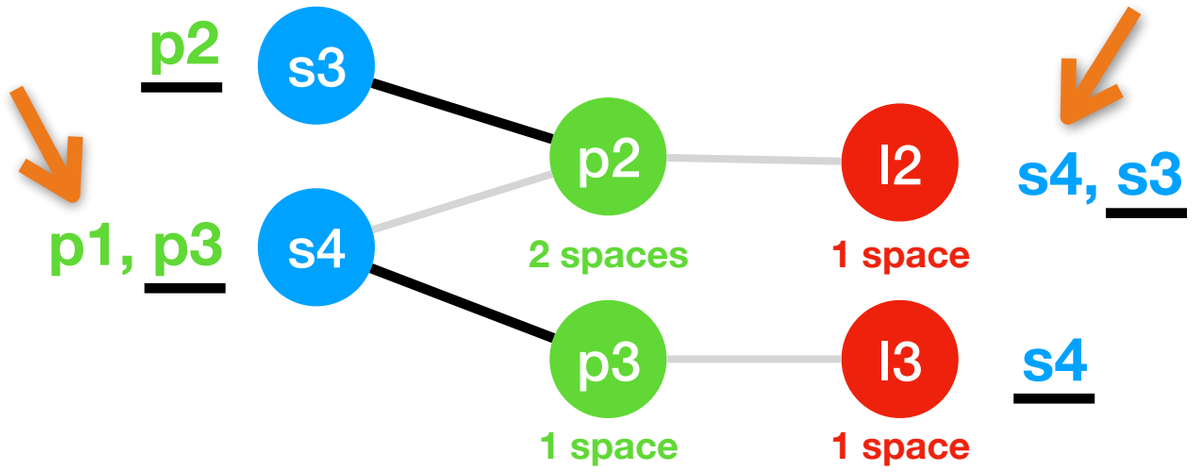

project undersubscribed, lecturer full

# weak stability

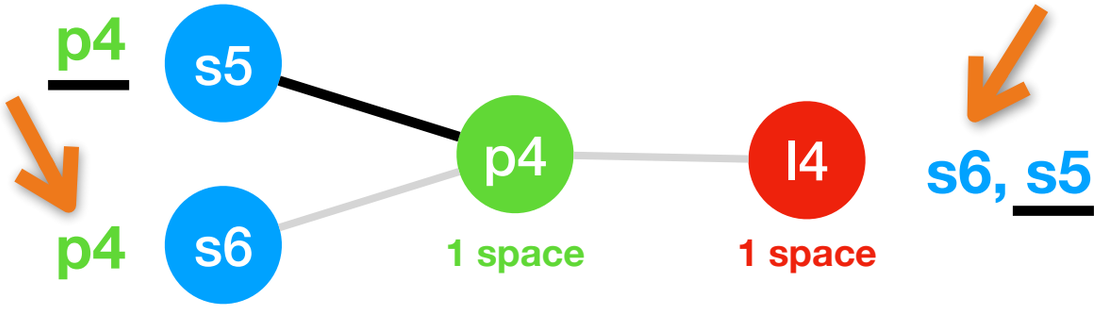
**Blocking pair: both agents are better off**




project and lecturer undersubscribed




project undersubscribed, lecturer full




project full, (lecturer full or undersubscribed)

# Maximum stable matchings

# Maximum stable matchings

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

# Maximum stable matchings

- A **stable matching** is a matching with no blocking pairs
- No ties in preference lists - find a stable matching in polynomial time - all same size

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

# Maximum stable matchings

- A **stable matching** is a matching with no blocking pairs
- No ties in preference lists - find a stable matching in polynomial time - all same size
- **Ties in preference lists** - find a stable matching in polynomial time - but stable matchings are **different sizes**

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

# Maximum stable matchings

- A **stable matching** is a matching with no blocking pairs
- No ties in preference lists - find a stable matching in polynomial time - all same size
- **Ties in preference lists** - find a stable matching in polynomial time - but stable matchings are **different sizes**
- Finding a maximum sized stable matching is **NP-hard**.

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

# Finding a maximum sized stable matching

# Finding a maximum sized stable matching

Two techniques:

# Finding a maximum sized stable matching

Two techniques:

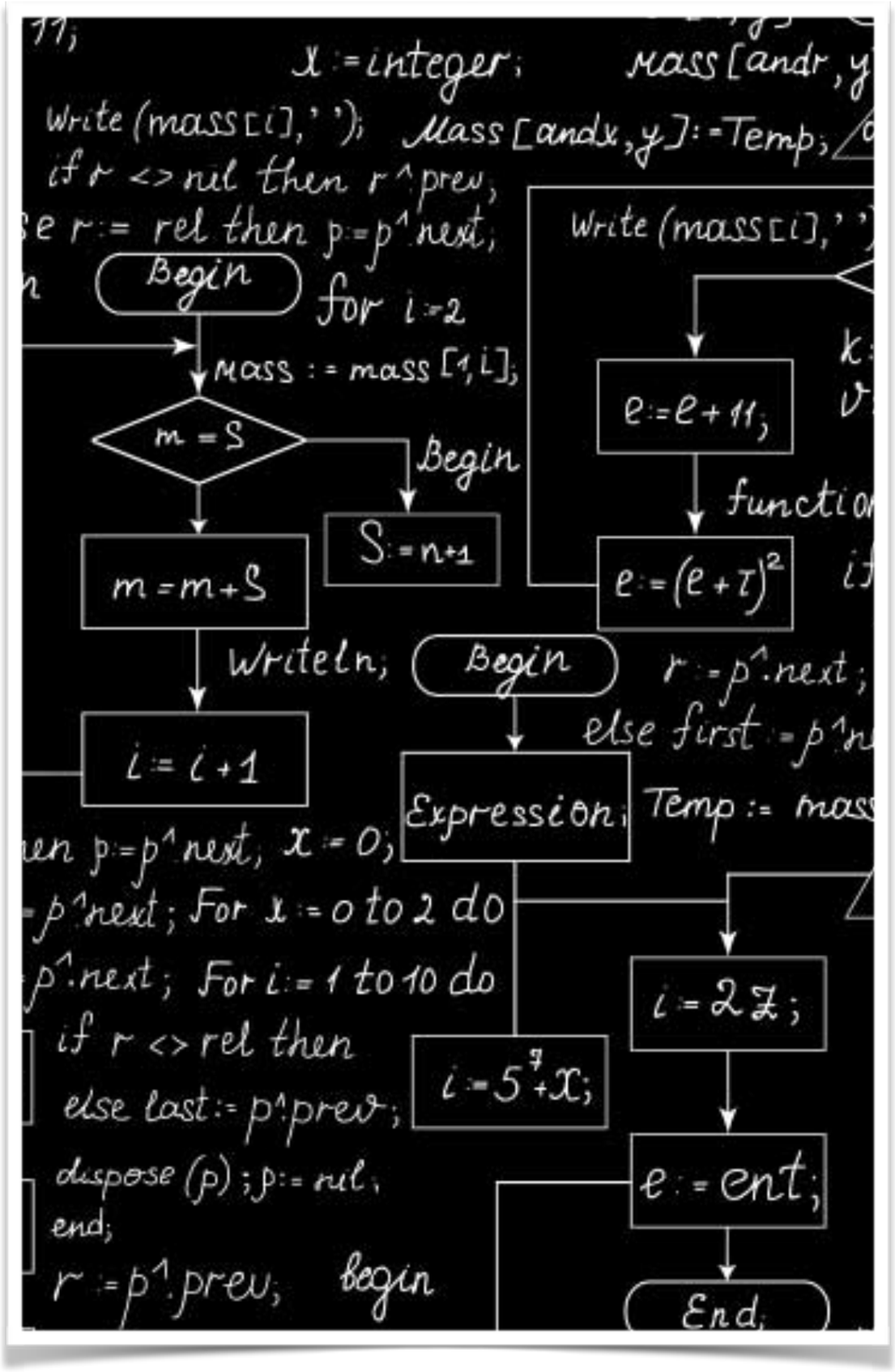
1. Approximation algorithm

# Finding a maximum sized stable matching

Two techniques:

1. Approximation algorithm
2. Integer Programming

# Approximation Algorithm



# Previous work

# Previous work

- Hospitals/Residents with Ties (HRT) - special case of SPA-ST, each lecturer offers one project and the capacity of each lecturer equals the capacity of their offered project

# Previous work

- Hospitals/Residents with Ties (HRT) - special case of SPA-ST, each lecturer offers one project and the capacity of each lecturer equals the capacity of their offered project
- A 3/2-approximation algorithm exists for HRT

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

# Previous work

- Hospitals/Residents with Ties (HRT) - special case of SPA-ST, each lecturer offers one project and the capacity of each lecturer equals the capacity of their offered project

- A 3/2-approximation algorithm exists for HRT

- Can I just convert my problem and use this conversion process?

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

# Previous work

- Hospitals/Residents with Ties (HRT) - special case of SPA-ST, each lecturer offers one project and the capacity of each lecturer equals the capacity of their offered project

- A 3/2-approximation algorithm exists for HRT

- Can I just convert my problem and use this conversion process?

- Not using a conversion process we tried.

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

# 3/2-approximation algorithm

# 3/2-approximation algorithm

- Created a new 3/2 approximation algorithm for SPA-ST, based on Kiraly's HRT algorithm.

# 3/2-approximation algorithm

- Created a new 3/2 approximation algorithm for SPA-ST, based on Kiraly's HRT algorithm.
- Moving from HRT to SPA-ST

# 3/2-approximation algorithm

- Created a new 3/2 approximation algorithm for SPA-ST, based on Kiraly's HRT algorithm.
- Moving from HRT to SPA-ST
    - Lecturers added a lot of complications

# 3/2-approximation algorithm

- Created a new 3/2 approximation algorithm for SPA-ST, based on Kiraly's HRT algorithm.
- Moving from HRT to SPA-ST
    - Lecturers added a lot of complications
    - Definition of a blocking pair is more complicated

# 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

# Proofs

# Proofs

Three proofs required:

# Proofs

Three proofs required:

- the resultant matching is stable

# Proofs

Three proofs required:

- the resultant matching is stable
- the algorithm runs in linear time

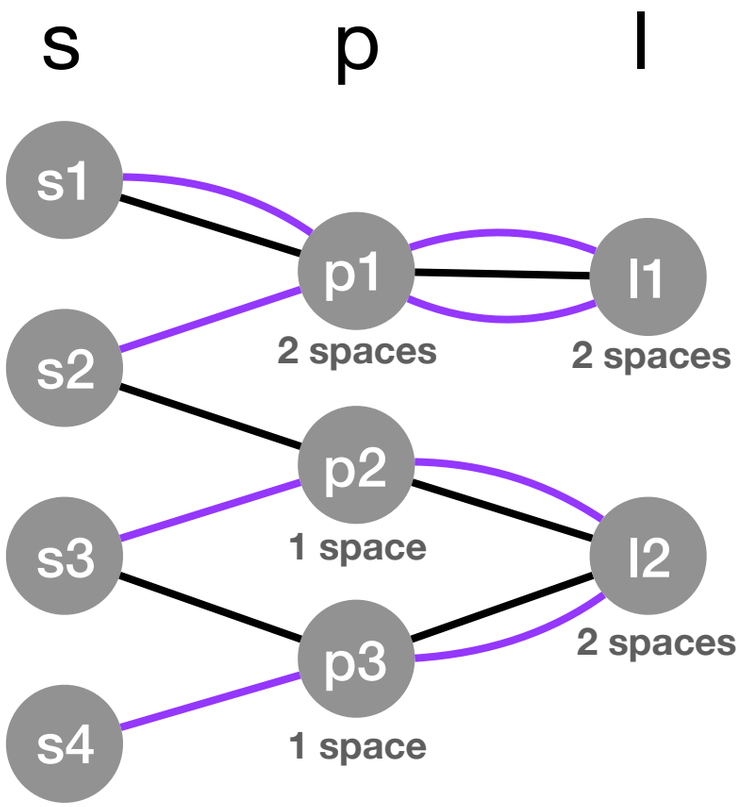
# Proofs

Three proofs required:

- the resultant matching is stable
- the algorithm runs in linear time
- the matching is at least 2/3 the size of optimal

# Performance guarantee - creating $G'$

# Performance guarantee - creating $G'$

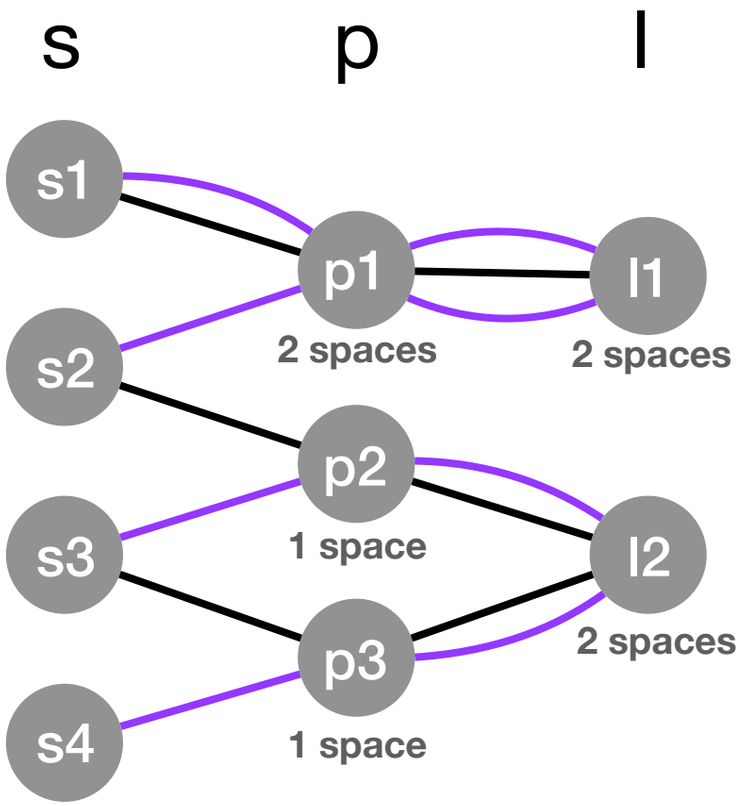



$G$

$M_{opt}$

$M$

# Performance guarantee - creating $G'$



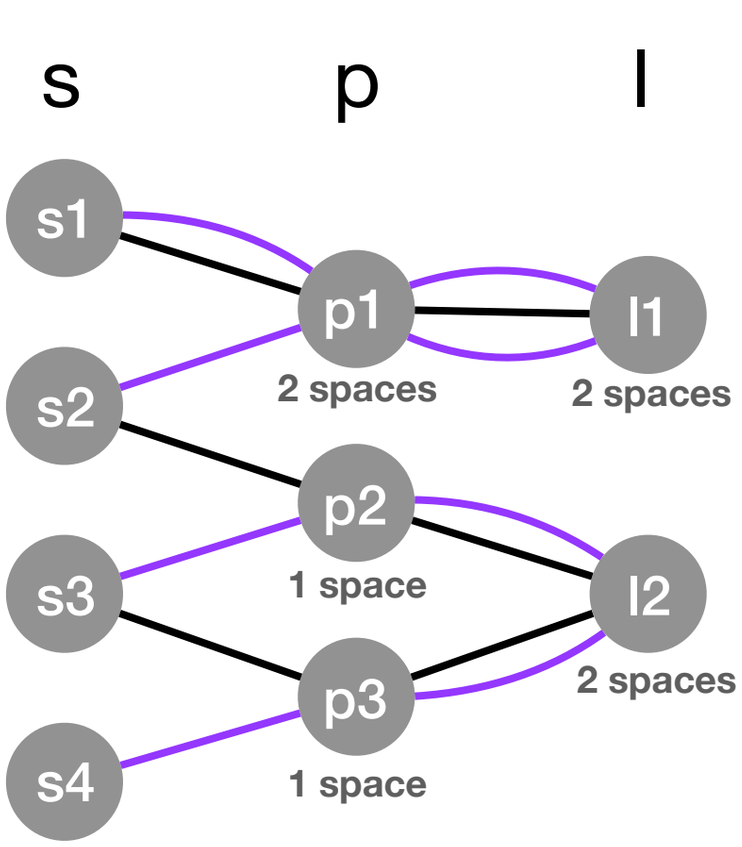

$G$

$G'$

$M_{opt}$

$M$

# Performance guarantee - creating $G'$



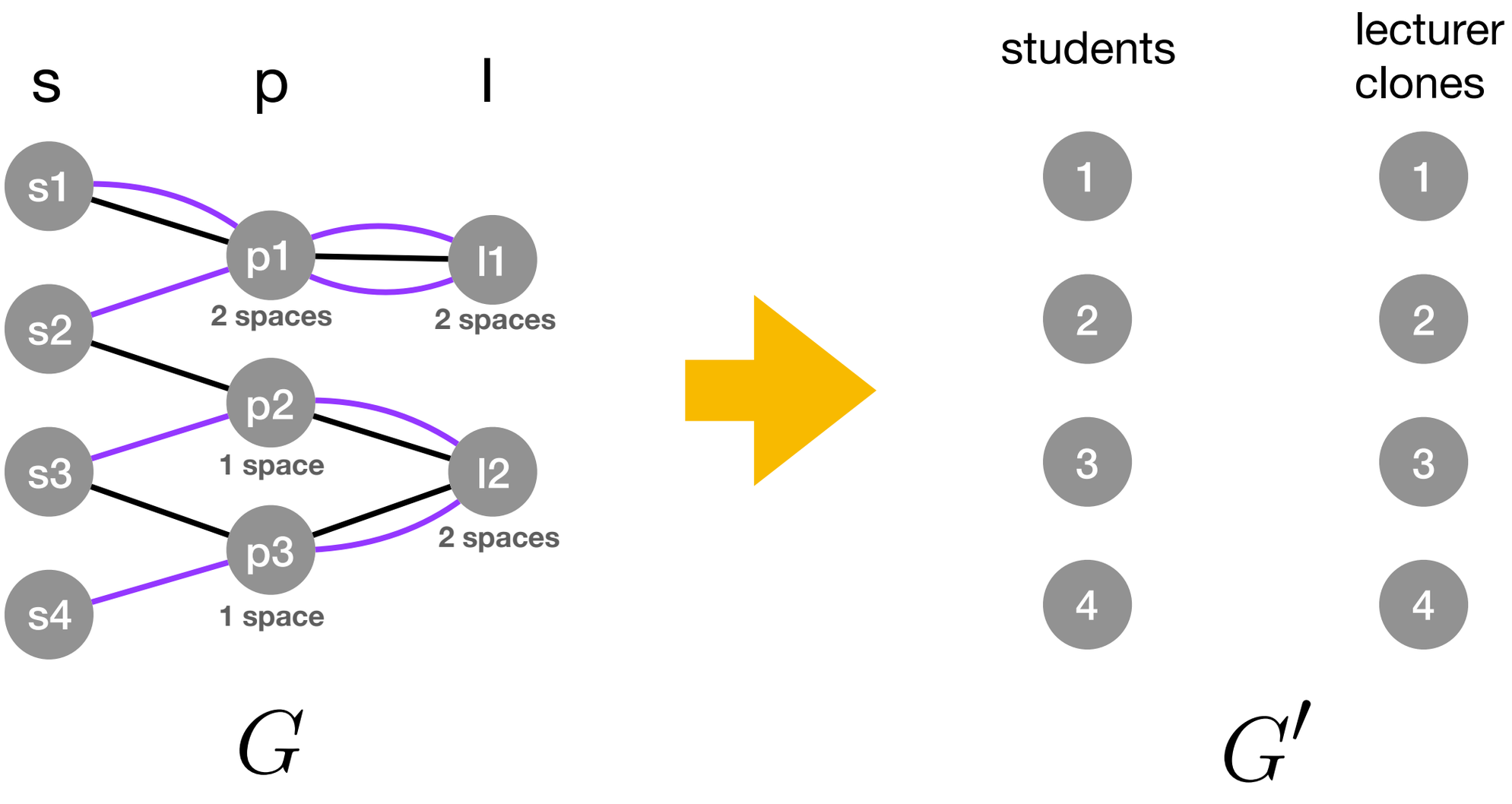

$G$


$G'$

$M_{opt}$

$M$

# Performance guarantee - creating $G'$

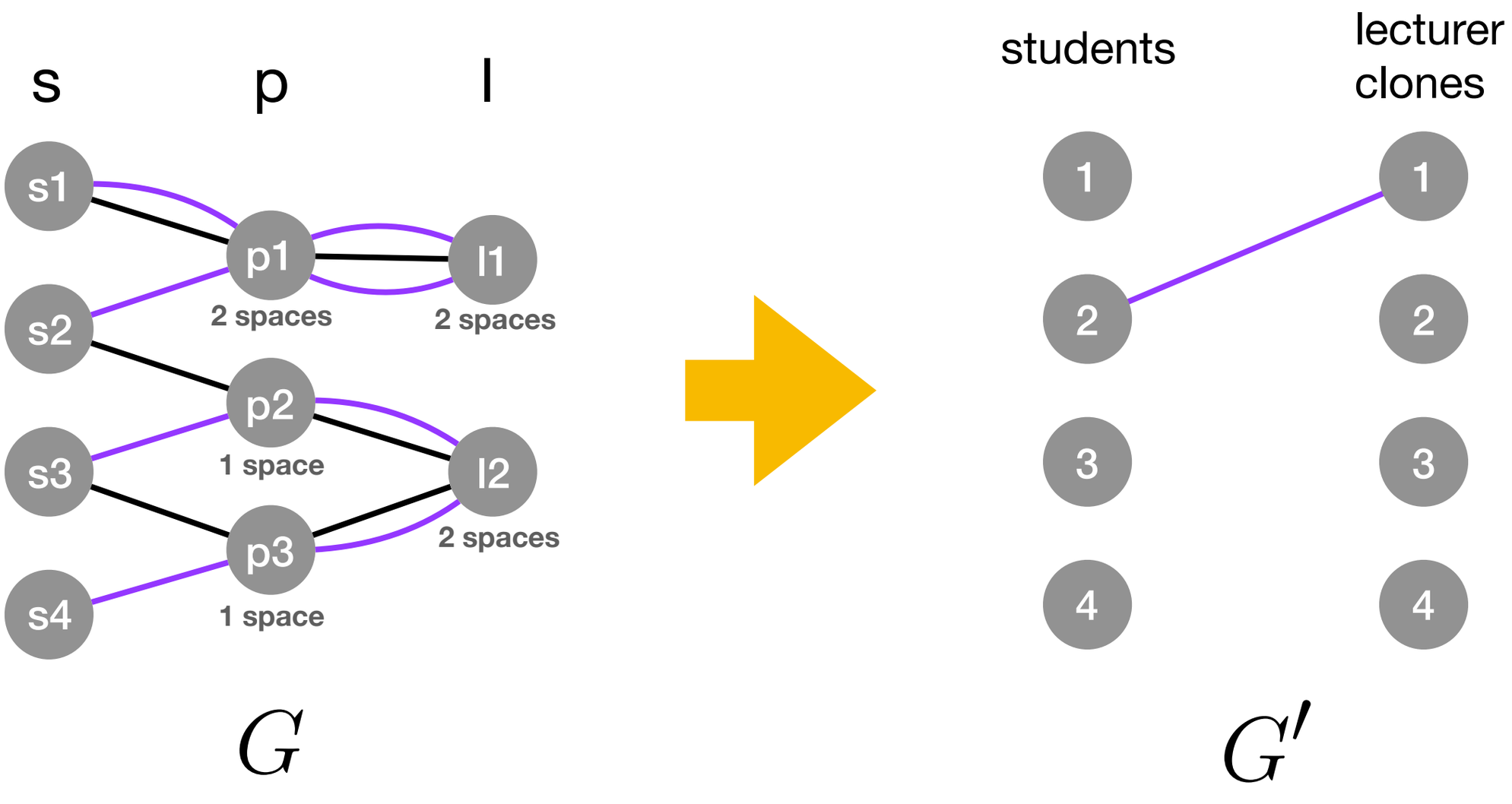



$G$ $G'$

$M_{opt}$

$M$

# Performance guarantee - creating $G'$

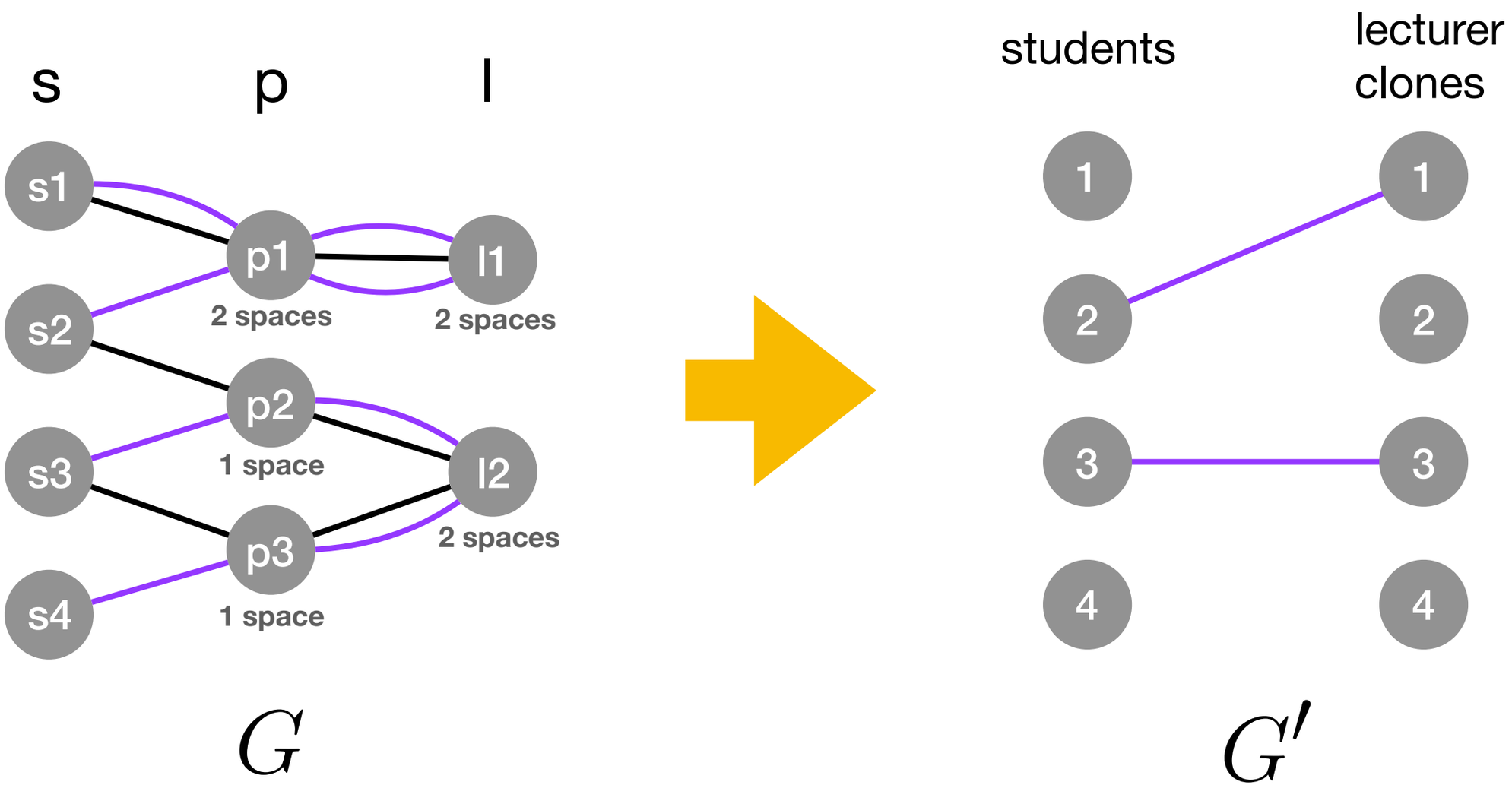



$G$ $G'$

$M_{opt}$

$M$

# Performance guarantee - creating $G'$

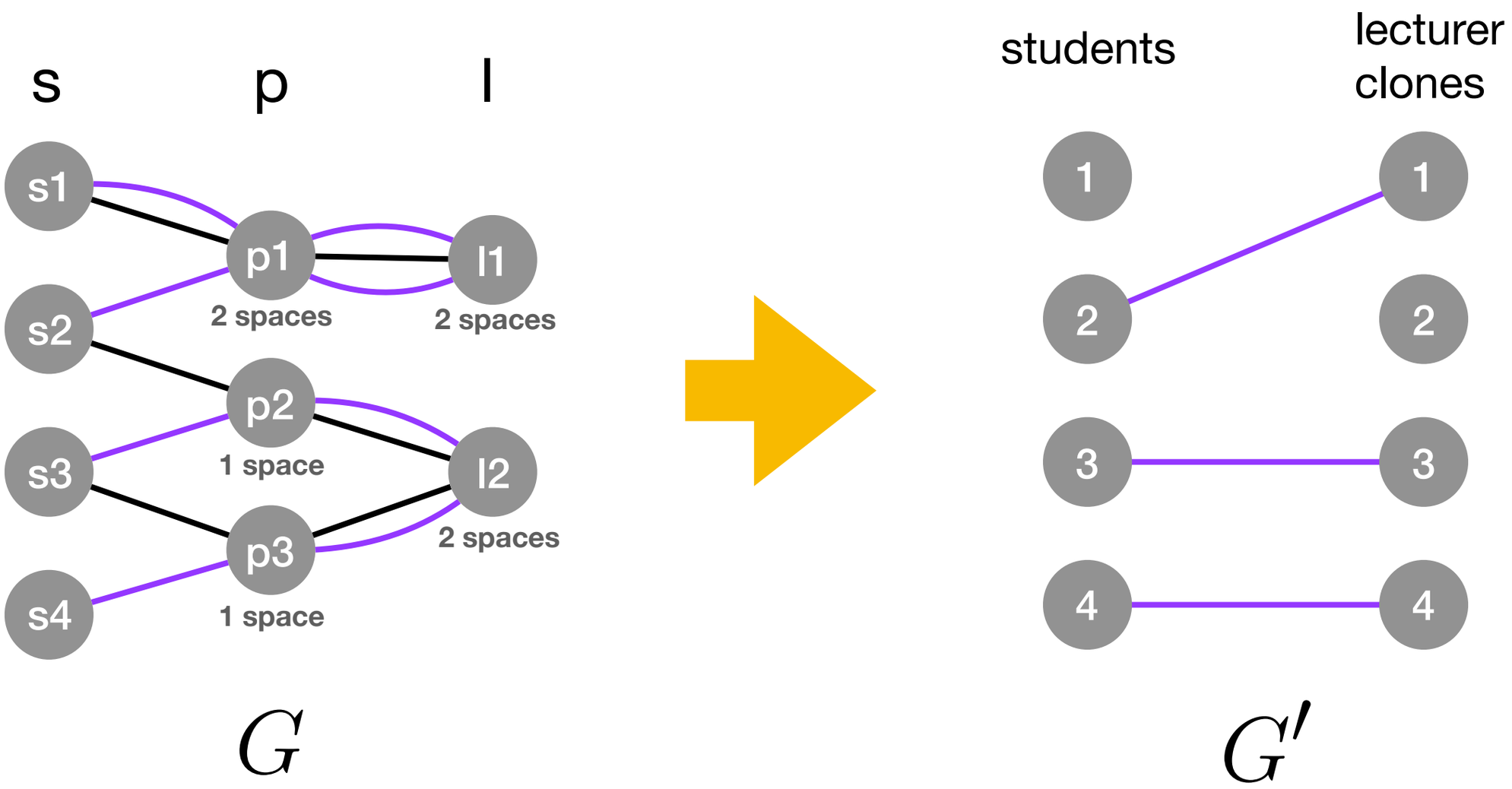



$G$ $G'$

$M_{opt}$

$M$

# Performance guarantee - creating $G'$

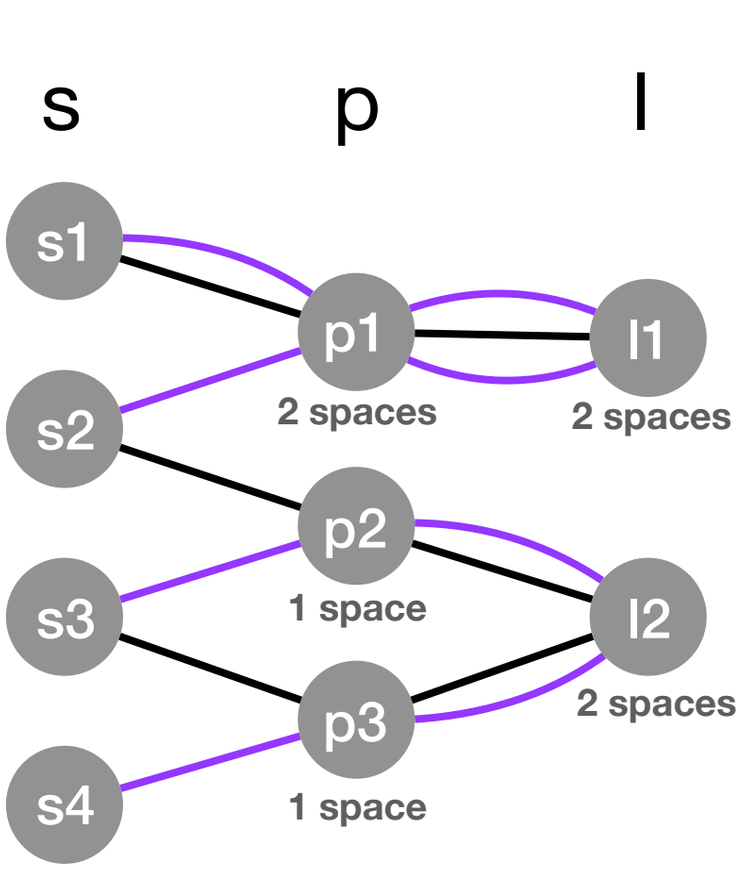



$G$ $G'$

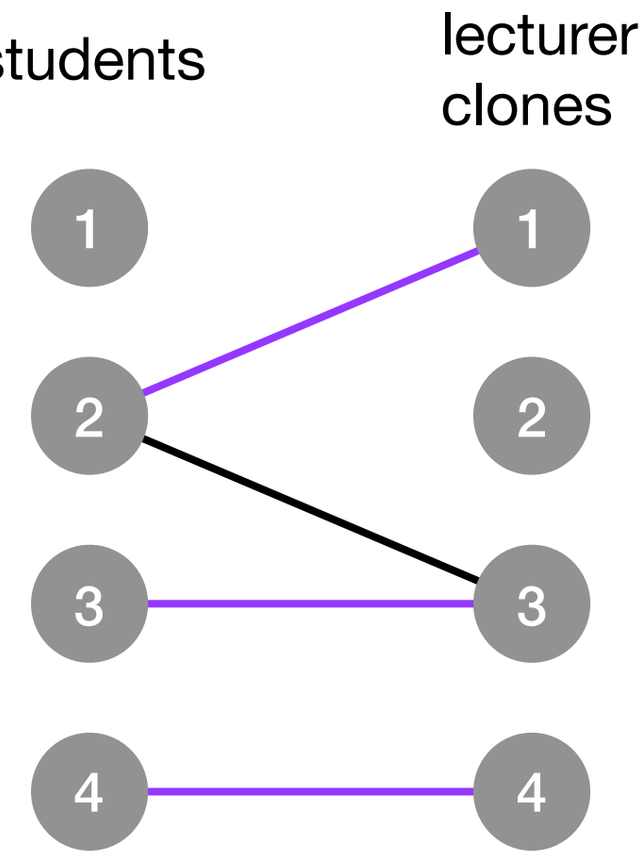
$M_{opt}$

$M$

# Performance guarantee - creating $G'$




$G$

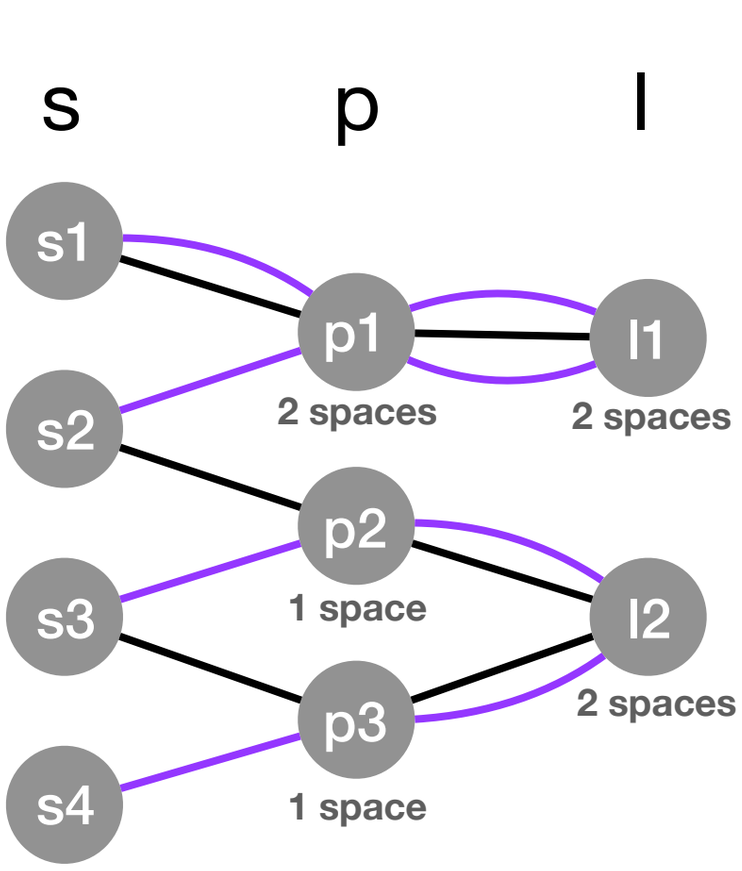



$G'$

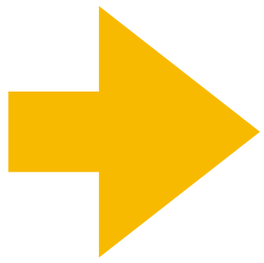
$M_{opt}$

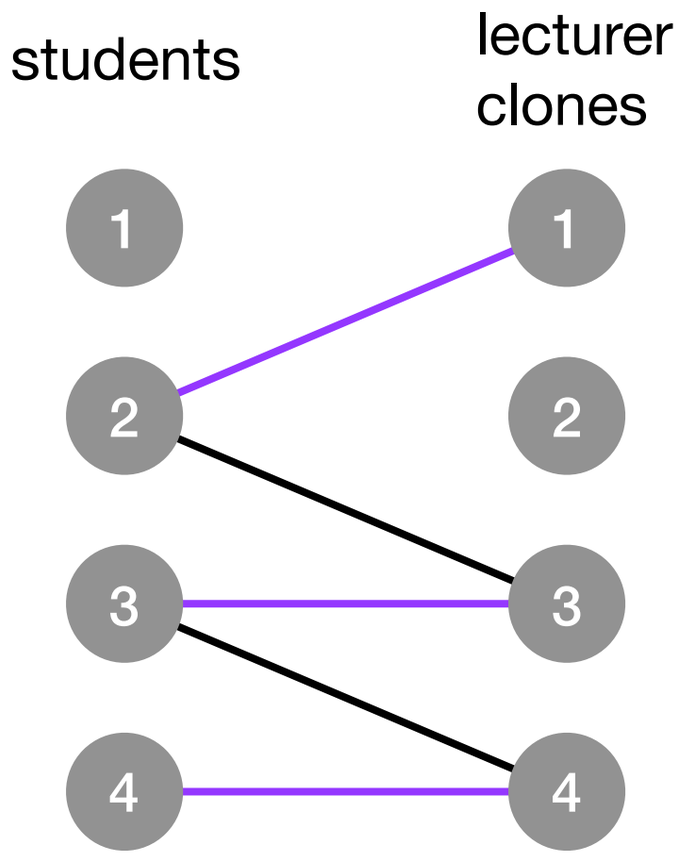
$M$

# Performance guarantee - creating $G'$




$G$



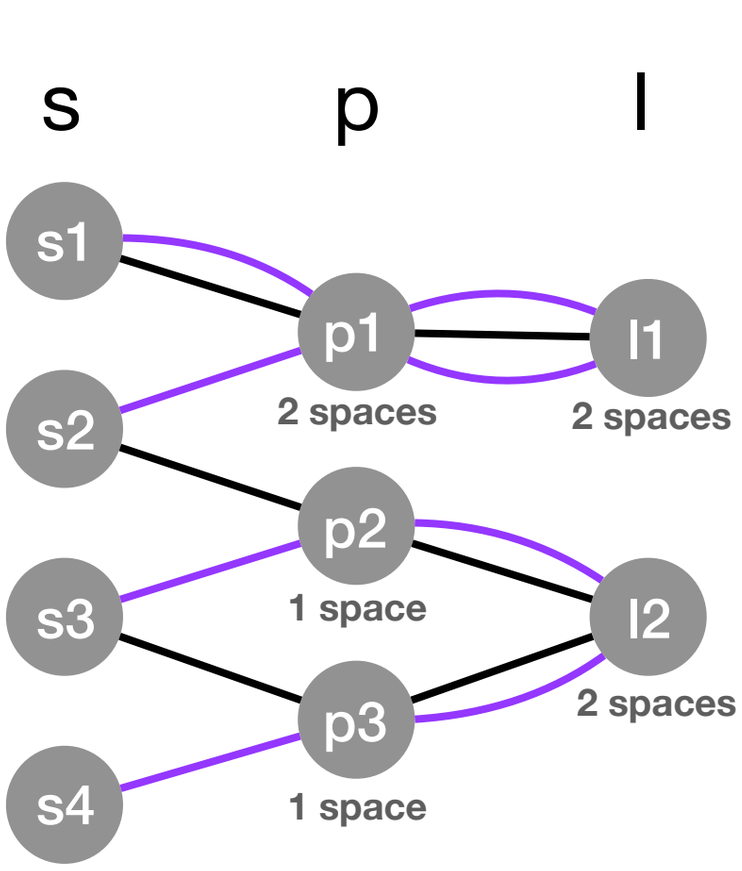



$G'$

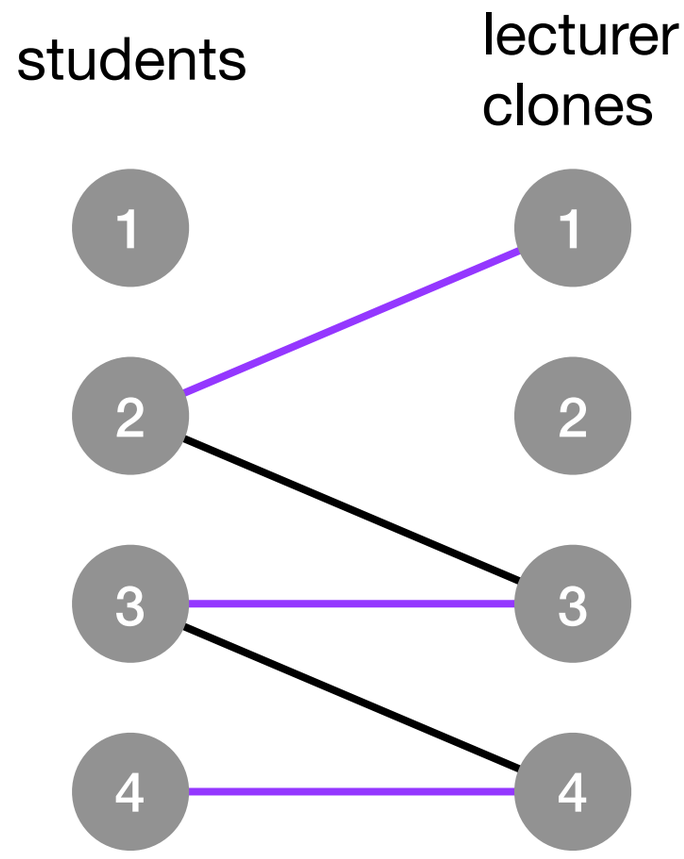
$M_{opt}$

$M$

# Performance guarantee - creating $G'$




$G$




$G'$

$M_{opt}$

$M$

$M'_{opt}$

$M'$

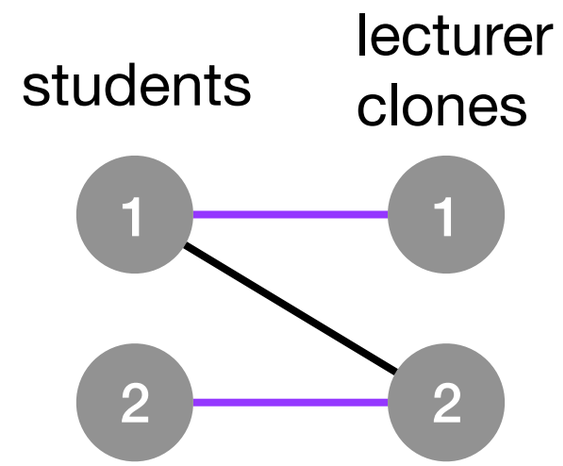
# Structures in $G'$

# Structures in $G'$

- odd length alternating path with end edges in $M'_{opt}$ (number of edges is 3)

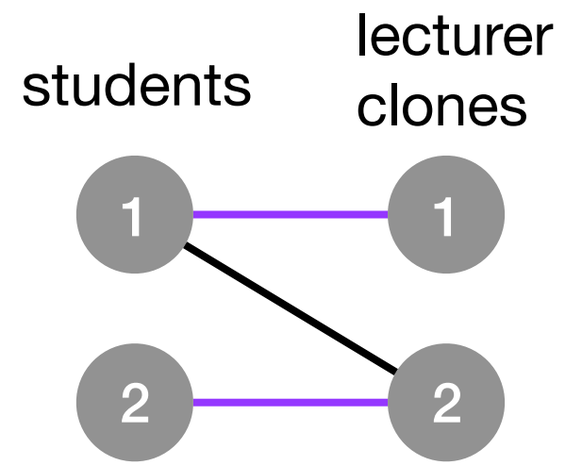
# Structures in $G'$

- odd length alternating path with end edges in $M'_{opt}$ (number of edges is 3)



# Structures in $G'$

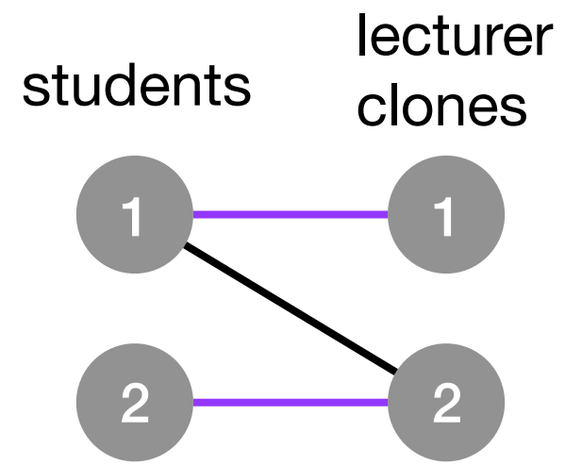
- odd length alternating path with end edges in $M'_{opt}$ (number of edges is 3)



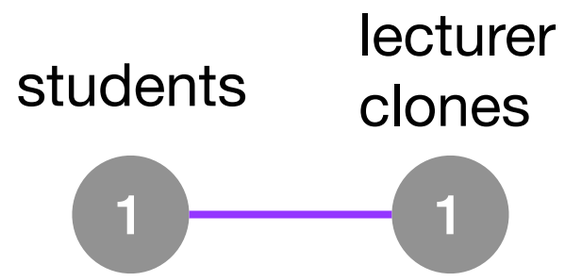

- odd length alternating path with end edges in $M'_{opt}$ (number of edges is 1)

# Structures in $G'$

- odd length alternating path with end edges in $M'_{opt}$ (number of edges is 3)

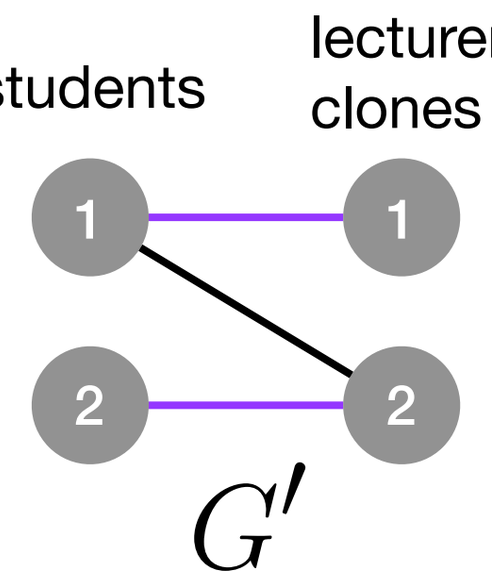



- odd length alternating path with end edges in $M'_{opt}$ (number of edges is 1)

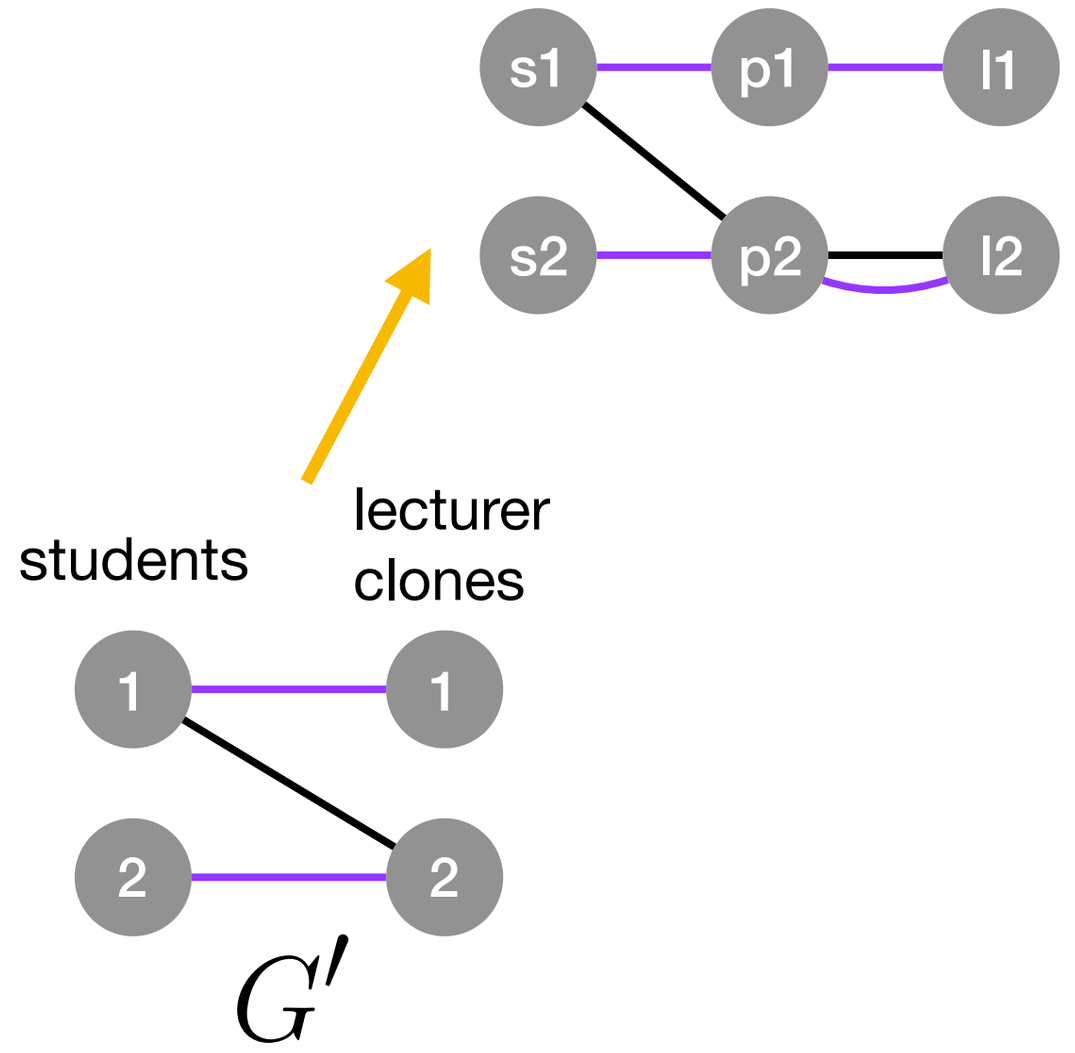


# Structures in $G'$

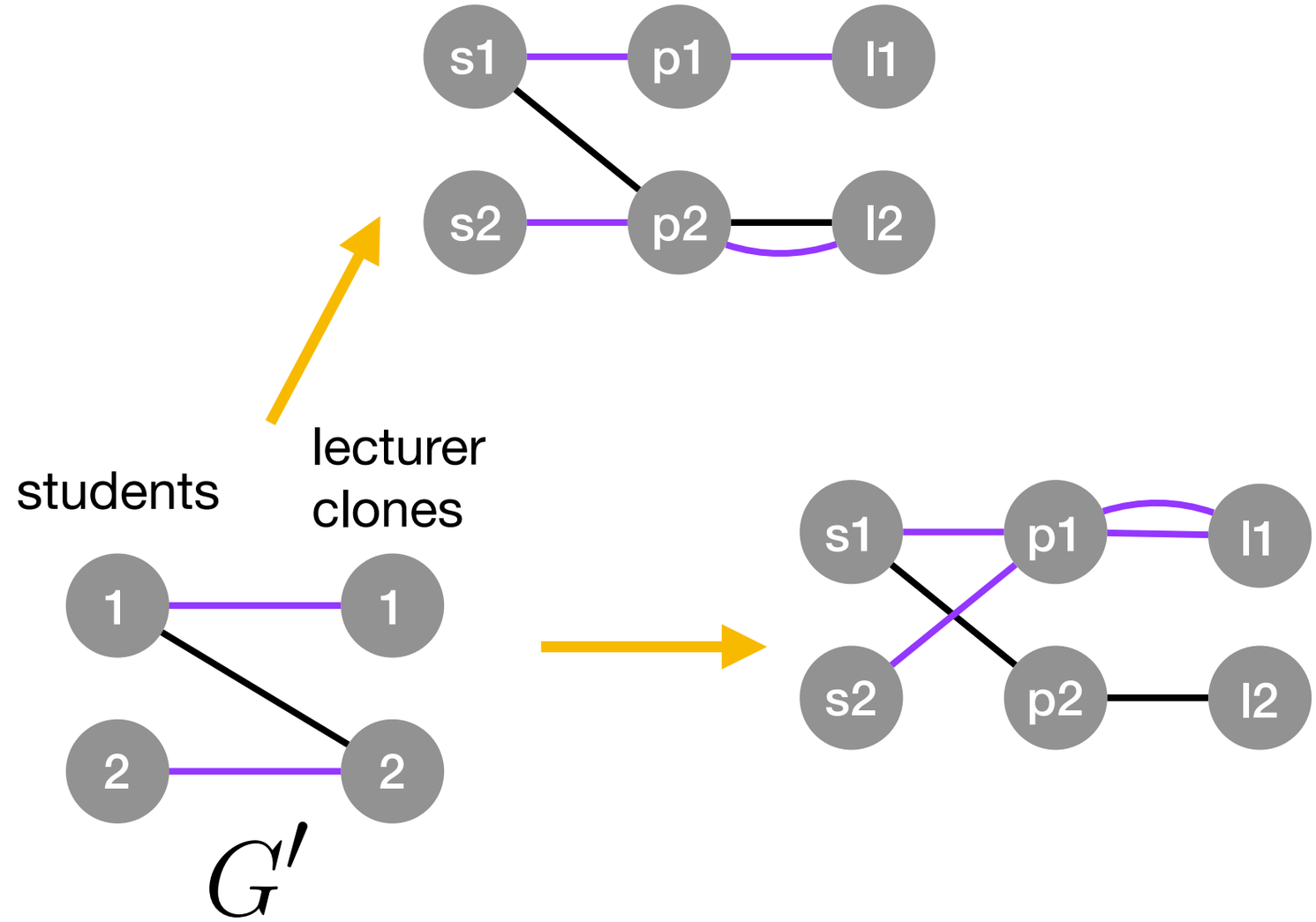
# Structures in $G'$



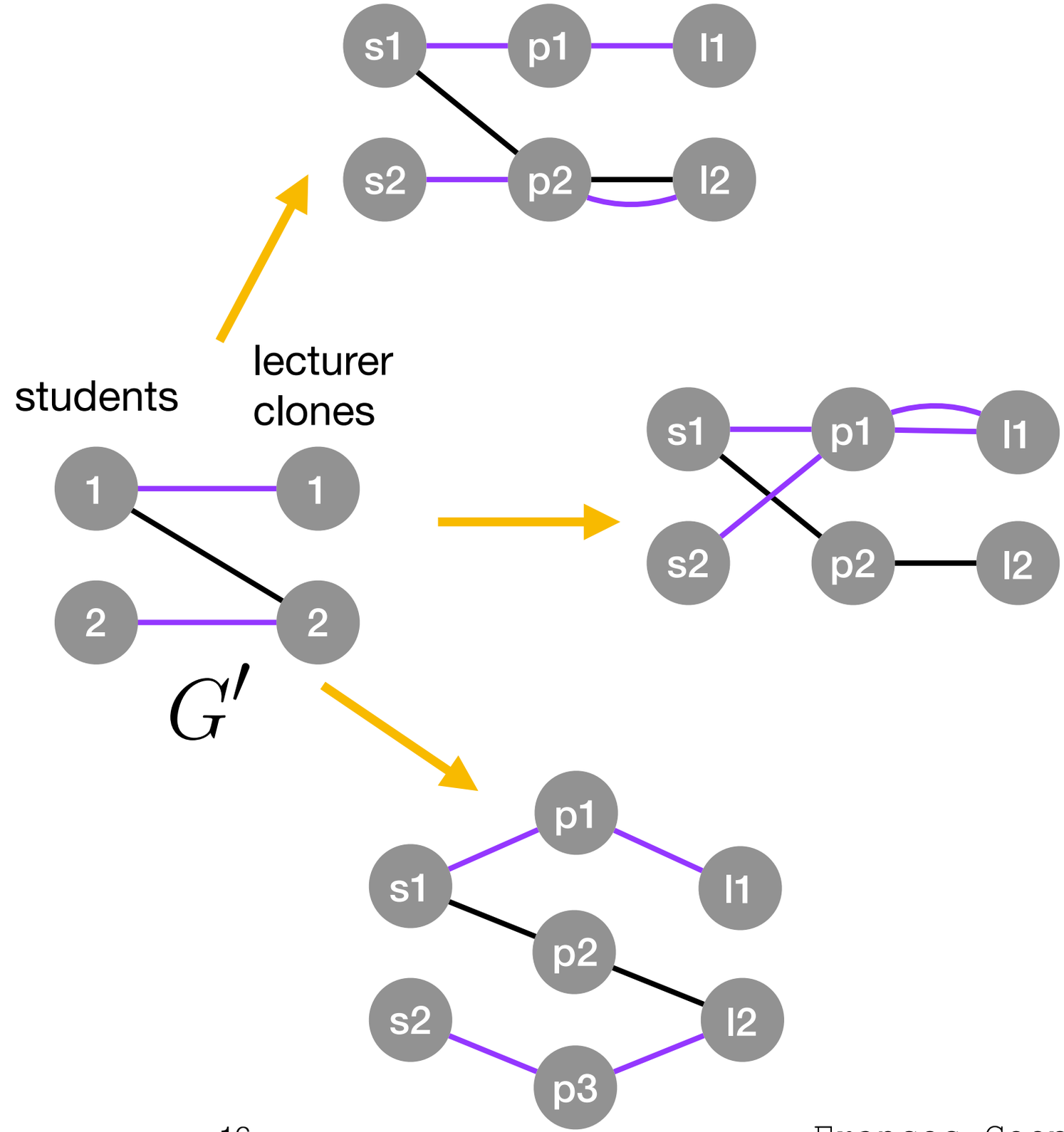
# Structures in $G'$



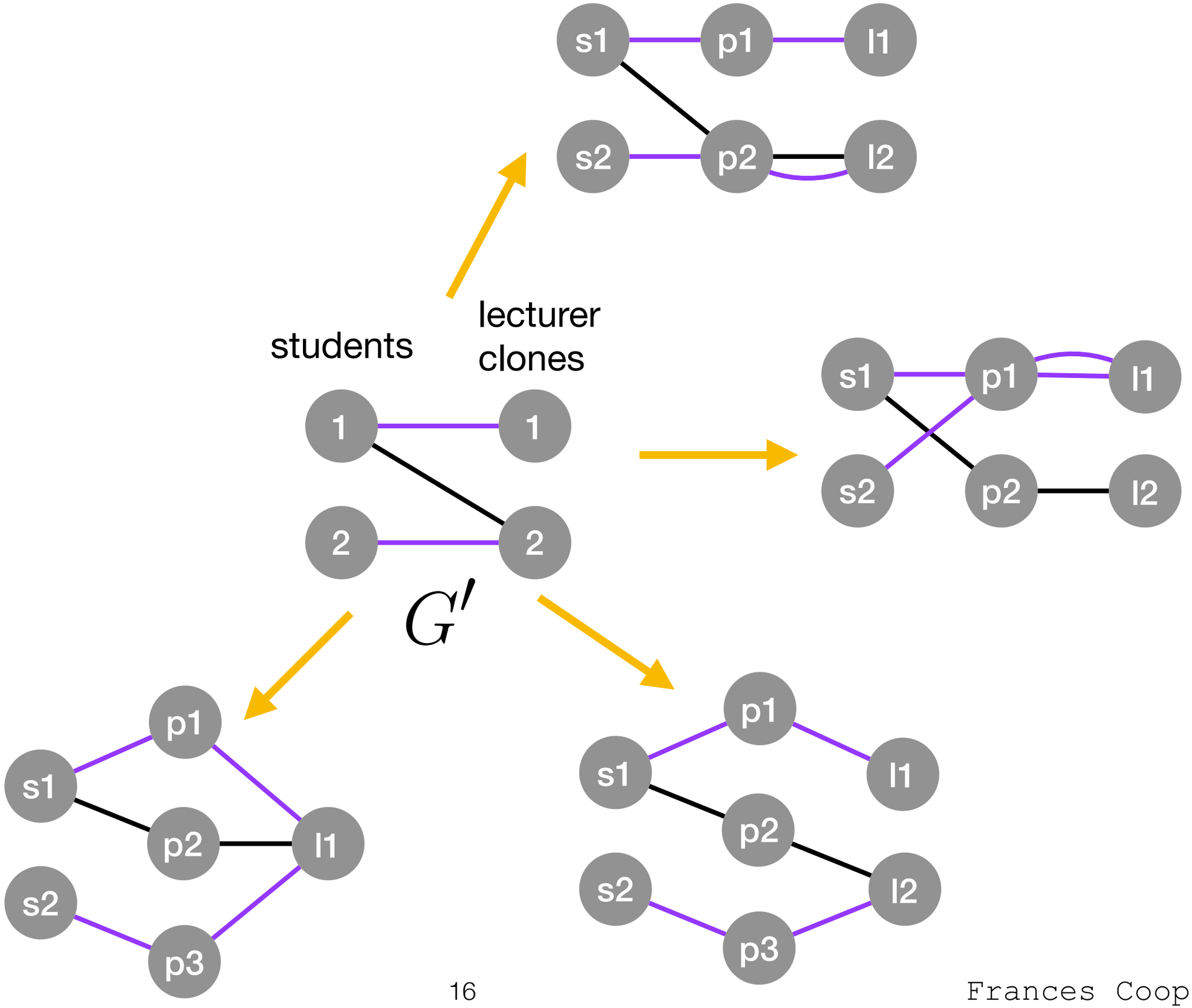
# Structures in $G'$



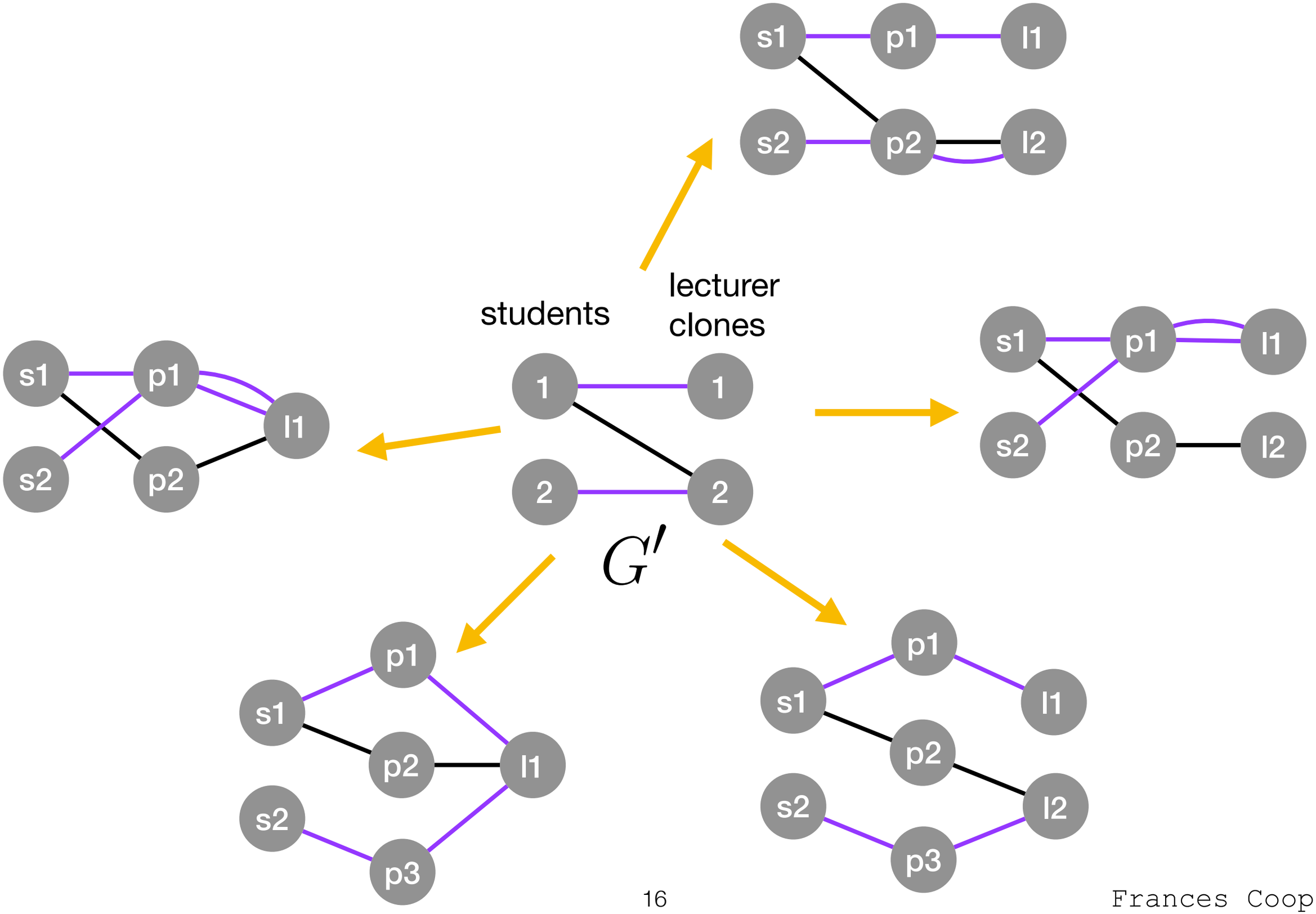
# Structures in $G'$



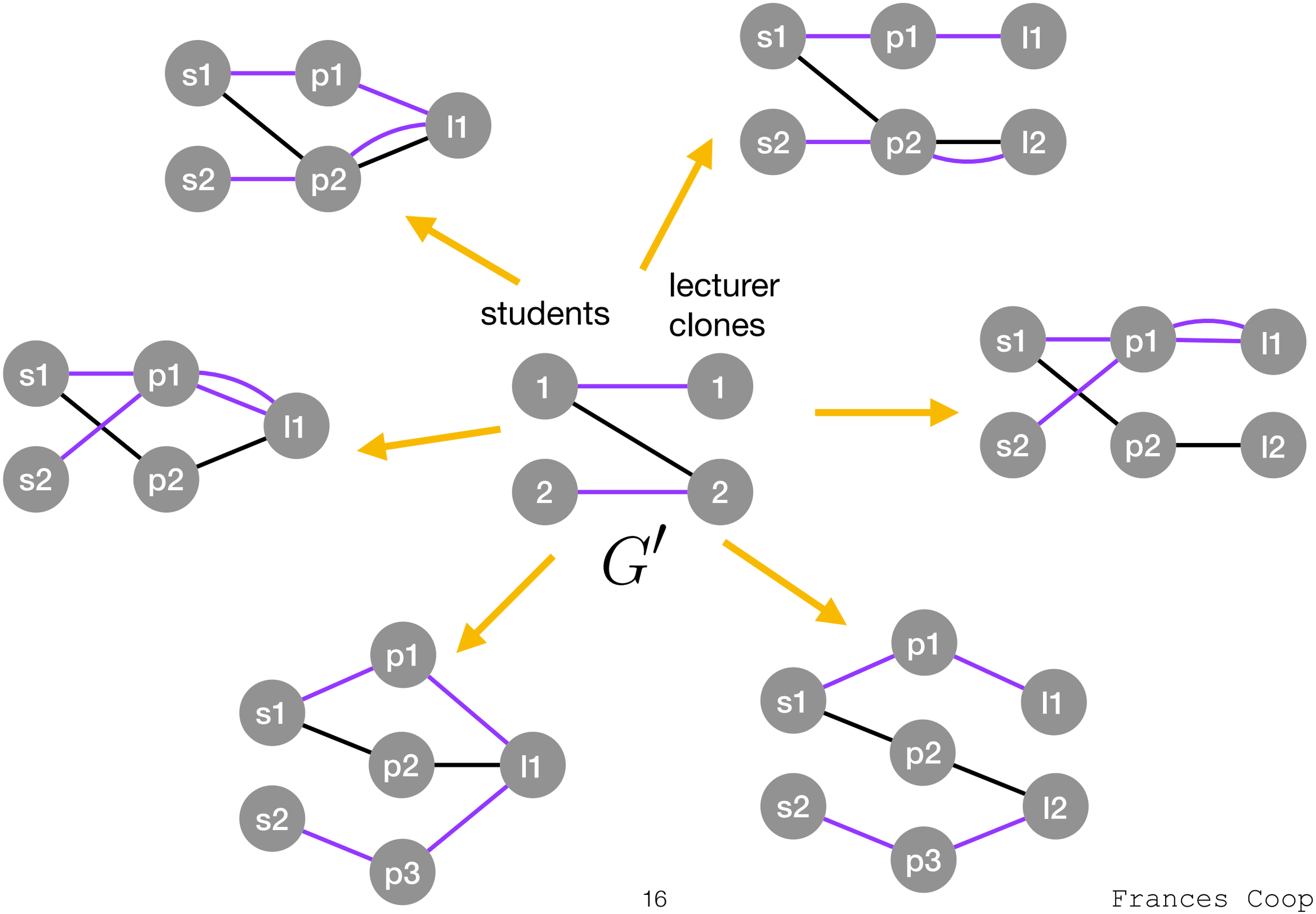
# Structures in $G'$



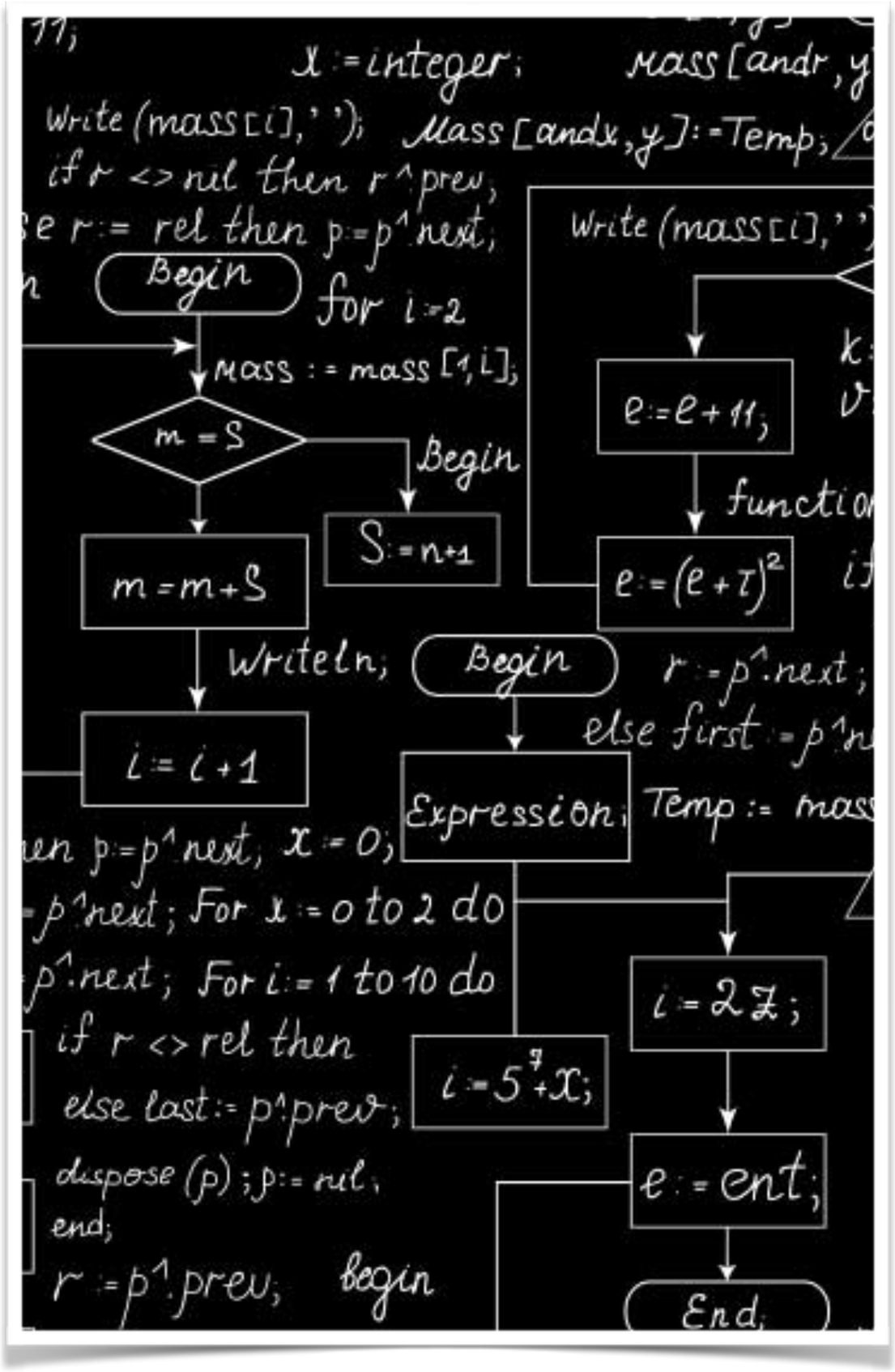
# Structures in $G'$



# Structures in $G'$



# Integer Program



# Integer Programming

# Integer Programming

# Integer Programming

- gives an optimal solution

# Integer Programming

- gives an optimal solution
- novel work: stability constraints

# Integer Programming

- gives an optimal solution
- novel work: stability constraints
- helped in correctness checking

# Integer Programming

- gives an optimal solution
- novel work: stability constraints
- helped in correctness checking
- Gives motivation for using approximation algorithm

# Experimental results

# Experimental Results

# Experimental Results

- Java (and Gurobi), 100s of thousands of instances with varying parameters. Ran on approximation algorithm and integer program.

# Experimental Results

- Java (and Gurobi), 100s of thousands of instances with varying parameters. Ran on approximation algorithm and integer program.

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

# Experimental Results

- Java (and Gurobi), 100s of thousands of instances with varying parameters. Ran on approximation algorithm and integer program.

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

| | minimum | average size | |
|---|---|---|---|
| Case | A/Max | A/Max | Min/Max |
| TIES1 | 1.0000 | 1.000 | 1.000 |
| TIES2 | 0.9792 | 0.997 | 0.987 |
| TIES3 | 0.9722 | 0.993 | 0.972 |
| TIES4 | 0.9655 | 0.990 | 0.958 |
| TIES5 | 0.9626 | 0.986 | 0.942 |
| TIES6 | 0.9558 | 0.984 | 0.927 |
| TIES7 | 0.9486 | 0.982 | 0.911 |
| TIES8 | 0.9527 | 0.980 | 0.896 |
| TIES9 | 0.9467 | 0.980 | 0.880 |
| TIES10 | 0.9529 | 0.982 | 0.866 |
| TIES11 | 0.9467 | 0.984 | 0.851 |

- TIES - 10,000 instances per set, 300 students, 250 projects (capacity 420), 120 lecturers (capacity 360), pref lists length 3 to 5.

# Experimental Results

- Java (and Gurobi), 100s of thousands of instances with varying parameters. Ran on approximation algorithm and integer program.

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

| | minimum | average size | |
|---|---|---|---|
| Case | A/Max | A/Max | Min/Max |
| TIES1 | 1.0000 | 1.000 | 1.000 |
| TIES2 | 0.9792 | 0.997 | 0.987 |
| TIES3 | 0.9722 | 0.993 | 0.972 |
| TIES4 | 0.9655 | 0.990 | 0.958 |
| TIES5 | 0.9626 | 0.986 | 0.942 |
| TIES6 | 0.9558 | 0.984 | 0.927 |
| TIES7 | 0.9486 | 0.982 | 0.911 |
| TIES8 | 0.9527 | 0.980 | 0.896 |
| TIES9 | 0.9467 | 0.980 | 0.880 |
| TIES10 | 0.9529 | 0.982 | 0.866 |
| TIES11 | 0.9467 | 0.984 | 0.851 |

- TIES - 10,000 instances per set, 300 students, 250 projects (capacity 420), 120 lecturers (capacity 360), pref lists length 3 to 5.

- increasing prob of student and lecturer ties from 0 to 0.5 in 0.05 steps

# Experimental Results

- Java (and Gurobi), 100s of thousands of instances with varying parameters. Ran on approximation algorithm and integer program.

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

| | minimum | average size | |
|---|---|---|---|
| Case | A/Max | A/Max | Min/Max |
| TIES1 | 1.0000 | 1.000 | 1.000 |
| TIES2 | 0.9792 | 0.997 | 0.987 |
| TIES3 | 0.9722 | 0.993 | 0.972 |
| TIES4 | 0.9655 | 0.990 | 0.958 |
| TIES5 | 0.9626 | 0.986 | 0.942 |
| TIES6 | 0.9558 | 0.984 | 0.927 |
| TIES7 | 0.9486 | 0.982 | 0.911 |
| TIES8 | 0.9527 | 0.980 | 0.896 |
| TIES9 | 0.9467 | 0.980 | 0.880 |
| TIES10 | 0.9529 | 0.982 | 0.866 |
| TIES11 | 0.9467 | 0.984 | 0.851 |

- TIES - 10,000 instances per set, 300 students, 250 projects (capacity 420), 120 lecturers (capacity 360), pref lists length 3 to 5.

- increasing prob of student and lecturer ties from 0 to 0.5 in 0.05 steps

- Average approx solution closer to optimal than minimum in all cases

# Experimental Results

# Experimental Results

## Scalability

| | instances completed | | average total time (ms) | |
|---|---|---|---|---|
| Case | A | Max | A | Max |
| SCALS1 | 10 | 10 | 1393.8 | 227764.3 |
| SCALS2 | 10 | 9 | 5356.7 | 1096045.6 |
| SCALS3 | 10 | 0 | 13095.3 | N/A |
| SCALS4 | 10 | 0 | 18883.5 | N/A |
| SCALS5 | 10 | 0 | 20993.0 | N/A |
| SCALP1 | 10 | 9 | 193.3 | 94242.9 |
| SCALP2 | 10 | 10 | 189.4 | 631225.2 |
| SCALP3 | 10 | 3 | 196.6 | 882251.0 |
| SCALP4 | 10 | 1 | 248.5 | 1594201.0 |
| SCALP5 | 10 | 0 | 283.7 | N/A |
| SCALP6 | 10 | 0 | 288.4 | N/A |

# Experimental Results

## Scalability

- SCALS - 10,000 students up to 50,000 students. Pref lists 3 to 5 and ties 0.2

| | instances completed | | average total time (ms) | |
|---|---|---|---|---|
| Case | A | Max | A | Max |
| SCALS1 | 10 | 10 | 1393.8 | 227764.3 |
| SCALS2 | 10 | 9 | 5356.7 | 1096045.6 |
| SCALS3 | 10 | 0 | 13095.3 | N/A |
| SCALS4 | 10 | 0 | 18883.5 | N/A |
| SCALS5 | 10 | 0 | 20993.0 | N/A |
| SCALP1 | 10 | 9 | 193.3 | 94242.9 |
| SCALP2 | 10 | 10 | 189.4 | 631225.2 |
| SCALP3 | 10 | 3 | 196.6 | 882251.0 |
| SCALP4 | 10 | 1 | 248.5 | 1594201.0 |
| SCALP5 | 10 | 0 | 283.7 | N/A |
| SCALP6 | 10 | 0 | 288.4 | N/A |

# Experimental Results

## Scalability

- SCALS - 10,000 students up to 50,000 students. Pref lists 3 to 5 and ties 0.2
- SCALP - 500 students, ties 0.4, Pref lists increased from 25 to 150 in steps of 25.

| | instances completed | | average total time (ms) | |
|---|---|---|---|---|
| Case | A | Max | A | Max |
| SCALS1 | 10 | 10 | 1393.8 | 227764.3 |
| SCALS2 | 10 | 9 | 5356.7 | 1096045.6 |
| SCALS3 | 10 | 0 | 13095.3 | N/A |
| SCALS4 | 10 | 0 | 18883.5 | N/A |
| SCALS5 | 10 | 0 | 20993.0 | N/A |
| SCALP1 | 10 | 9 | 193.3 | 94242.9 |
| SCALP2 | 10 | 10 | 189.4 | 631225.2 |
| SCALP3 | 10 | 3 | 196.6 | 882251.0 |
| SCALP4 | 10 | 1 | 248.5 | 1594201.0 |
| SCALP5 | 10 | 0 | 283.7 | N/A |
| SCALP6 | 10 | 0 | 288.4 | N/A |

# Experimental Results

## Scalability

- SCALS - 10,000 students up to 50,000 students. Pref lists 3 to 5 and ties 0.2

- SCALP - 500 students, ties 0.4, Pref lists increased from 25 to 150 in steps of 25.

- much faster than using the integer program

| | instances completed | | average total time (ms) | |
|---|---|---|---|---|
| Case | A | Max | A | Max |
| SCALS1 | 10 | 10 | 1393.8 | 227764.3 |
| SCALS2 | 10 | 9 | 5356.7 | 1096045.6 |
| SCALS3 | 10 | 0 | 13095.3 | N/A |
| SCALS4 | 10 | 0 | 18883.5 | N/A |
| SCALS5 | 10 | 0 | 20993.0 | N/A |
| SCALP1 | 10 | 9 | 193.3 | 94242.9 |
| SCALP2 | 10 | 10 | 189.4 | 631225.2 |
| SCALP3 | 10 | 3 | 196.6 | 882251.0 |
| SCALP4 | 10 | 1 | 248.5 | 1594201.0 |
| SCALP5 | 10 | 0 | 283.7 | N/A |
| SCALP6 | 10 | 0 | 288.4 | N/A |

# Experimental Results

# Experimental Results

- So is it worth using?

# Experimental Results

- So is it worth using?


- Coram - assigning adopted children to families. ~ 100's of agents. Preference lists long and probability of ties high

# Experimental Results

- So is it worth using?

coram | better chances for children since 1739

- Coram - assigning adopted children to families. ~ 100's of agents. Preference lists long and probability of ties high

- 21 instances, increasing difficulty. IP could only solve first 6 within 5 minutes, approximation algorithm took less than 2 seconds for each

# Future Work

# Future Work

- Finding an approximation algorithm with a better performance guarantee than 3/2

# Future Work

- Finding an approximation algorithm with a better performance guarantee than 3/2
- Finding a better inapproximability result than 33/29

Approximation Algorithms for Stable Matching Problems; PhD thesis; 2007; H. Yanagisawa

# Future Work

- Finding an approximation algorithm with a better performance guarantee than 3/2
- Finding a better inapproximability result than 33/29
- coalitions:

Approximation Algorithms for Stable Matching Problems; PhD thesis; 2007; H. Yanagisawa

# Future Work

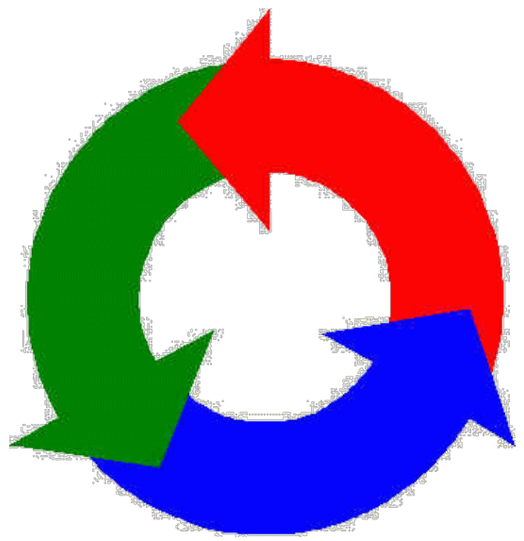
- Finding an approximation algorithm with a better performance guarantee than 3/2
- Finding a better inapproximability result than 33/29
- coalitions:
    - group of several students and lecturers

Approximation Algorithms for Stable Matching Problems; PhD thesis; 2007; H. Yanagisawa

# Future Work

- Finding an approximation algorithm with a better performance guarantee than 3/2
- Finding a better inapproximability result than 33/29
- coalitions:
  - group of several students and lecturers
  - permute their assignments

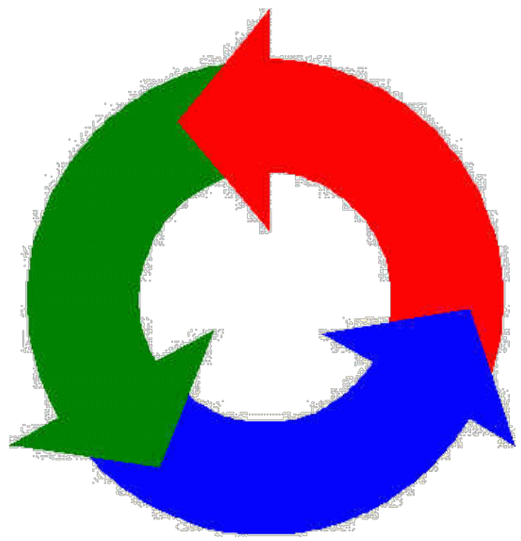
Approximation Algorithms for Stable Matching Problems; PhD thesis; 2007; H. Yanagisawa



# Future Work

- Finding an approximation algorithm with a better performance guarantee than 3/2
- Finding a better inapproximability result than 33/29
- coalitions:
  - group of several students and lecturers
  - permute their assignments
  - some or all get a better outcome

Approximation Algorithms for Stable Matching Problems; PhD thesis; 2007; H. Yanagisawa



# Thank you

## Summary

- Student-project allocation problem
- Finding a maximum stable matching
  - Integer programming
  - Approximation algorithm
- Future work: improved performance guarantee; improved inapproximability result; coalitions

University of Glasgow

f.cooper.1@research.gla.ac.uk
http://fmcooper.github.io

EPSRC
Engineering and Physical Sciences Research Council
EPSRC Doctoral Training Account