Lecture 1

Thinking like a Computer Scientist

Chinmay Nirkhe | CSE 421 Spring 2025

Thinking like a Computer Scientist

The computational lens

Nobel Physics Prize Awarded for Pioneering A.I. Research by 2 Scientists

With work on machine learning that uses artificial neural networks, John J. Hopfield and Geoffrey E. Hinton “showed a completely new way for us to use computers,” the committee said.

The New York Times, October 2024

Ideas & Trends; The Nobels: Dazzled By the Digital Light

By George Johnson

Oct. 15, 2000

The New York Times, October 2000

Nobel Prize in Chemistry Goes to 3 Scientists for Predicting and Creating Proteins

The Nobel, awarded to David Baker of the University of Washington and Demis Hassabis and John M. Jumper of Google DeepMind, is the second this week to involve artificial intelligence.

The New York Times, October 2024

> There is no Nobel Prize for computer science. But obliquely and perhaps unconsciously, the judges were using the tools at their disposal to recognize how formidable the notion of information has become, pervading not just the technologies we devise but the way we think about ourselves.
>
> – George Johnson

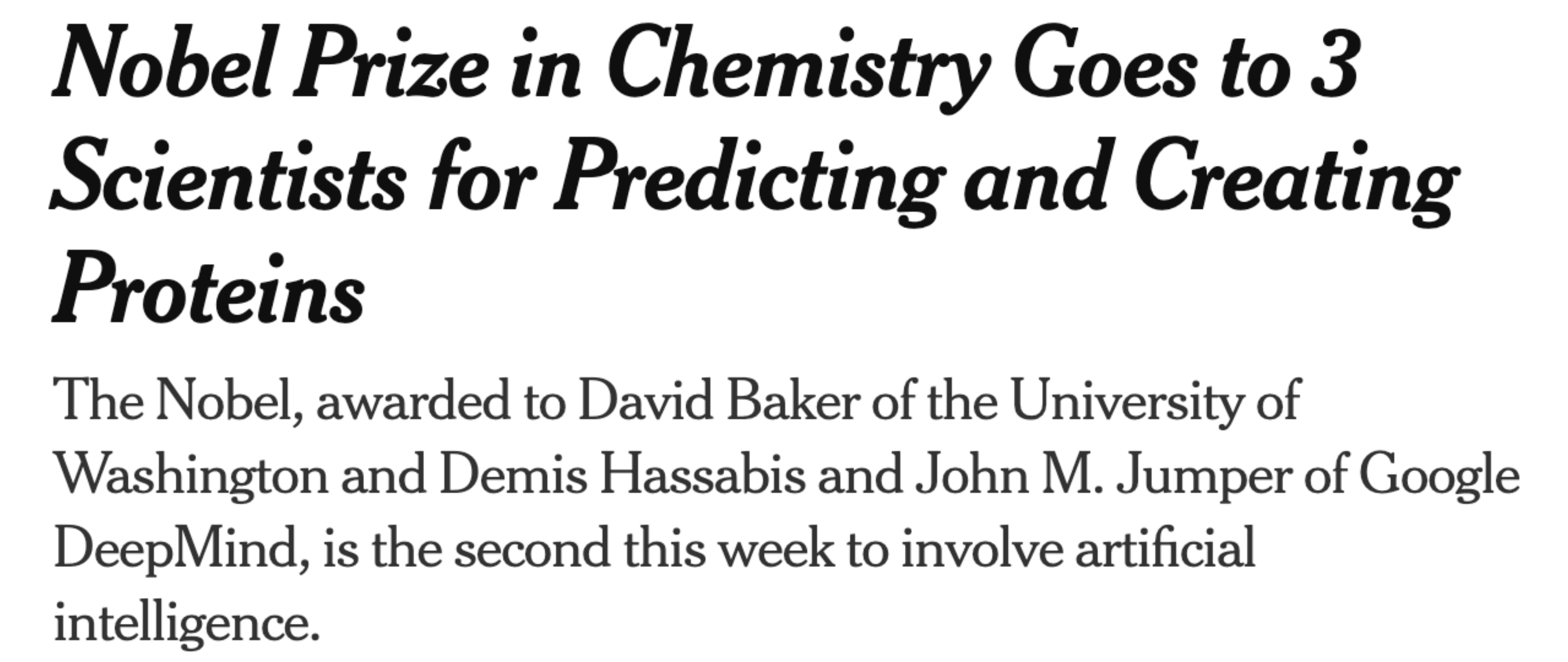
Thinking like a Computer Scientist

The computational lens

Nobel Physics Prize Awarded for Pioneering A.I. Research by 2 Scientists

With work on machine learning that uses artificial neural networks, John J. Hopfield and Geoffrey E. Hinton “showed a completely new way for us to use computers,” the committee said.

The New York Times, October 2024



Nobel Prize in Chemistry Goes to 3 Scientists for Predicting and Creating Proteins

The Nobel, awarded to David Baker of the University of Washington and Demis Hassabis and John M. Jumper of Google DeepMind, is the second this week to involve artificial intelligence.

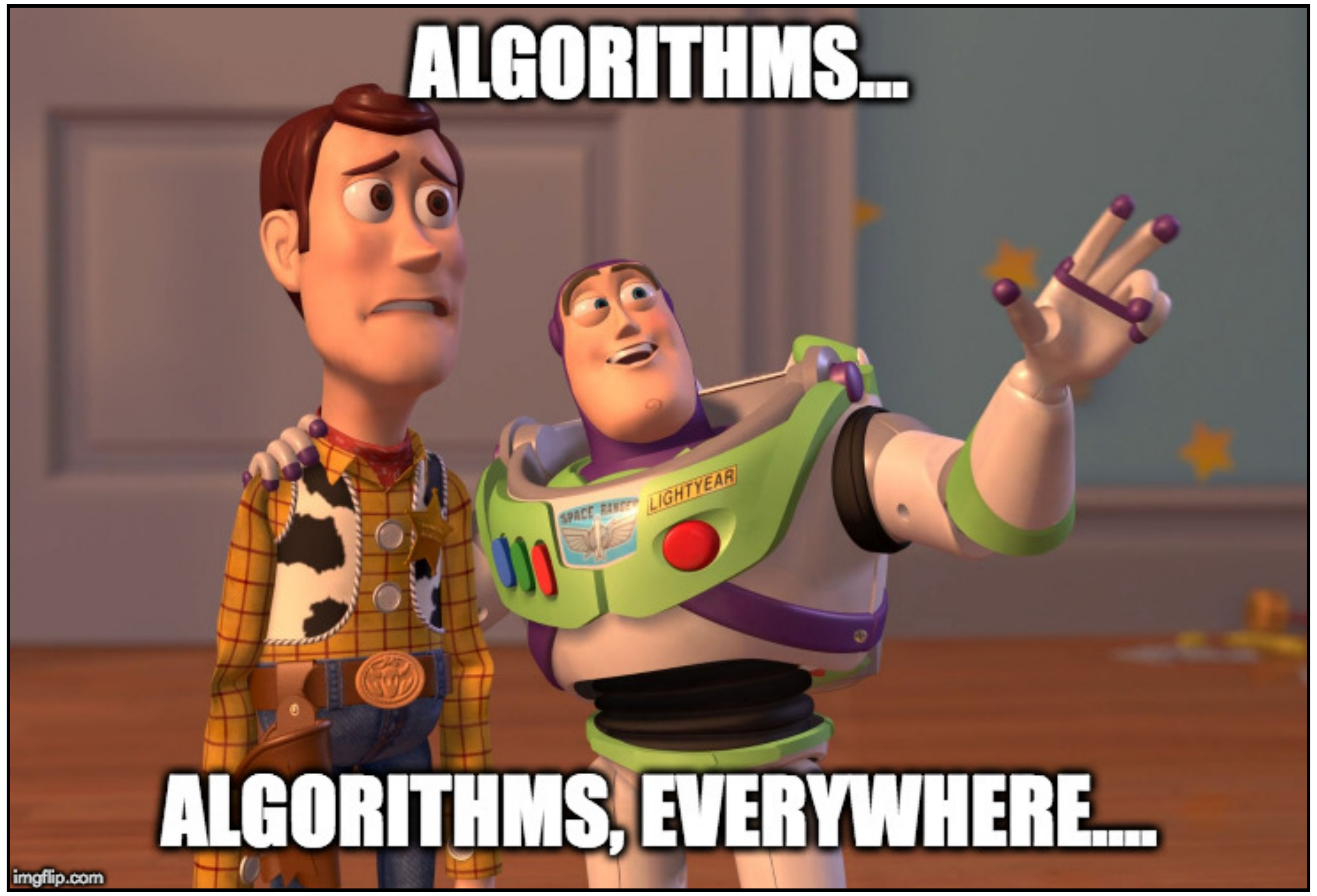
The New York Times, October 2024

- 2024 seemed like it was the year of “thinking like a computer scientist”. But really its the 2000s that is the century of thinking like a computer scientist.
- As the problems we wish to solve get bigger and bigger, understanding the *computational cost* associated with solving problems will gain an outsized importance.
- My goal is to train you how to think about the world through a *computational lens*.

The goal of this course

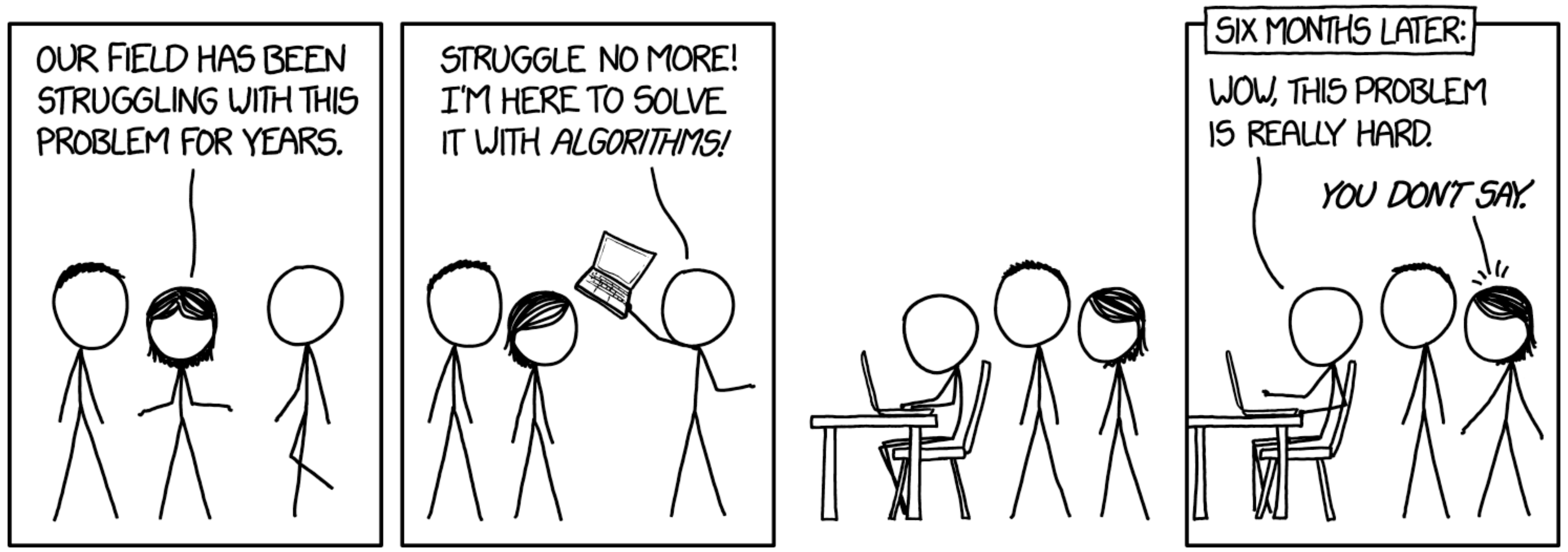
a.k.a. why I believe this course should be a required

- Help you learn to identify algorithmic problems
- Develop a toolkit for finding efficient algorithms
- Develop techniques for proving *correctness* and *analyzing* their properties
- Communicate your algorithms and their properties to others

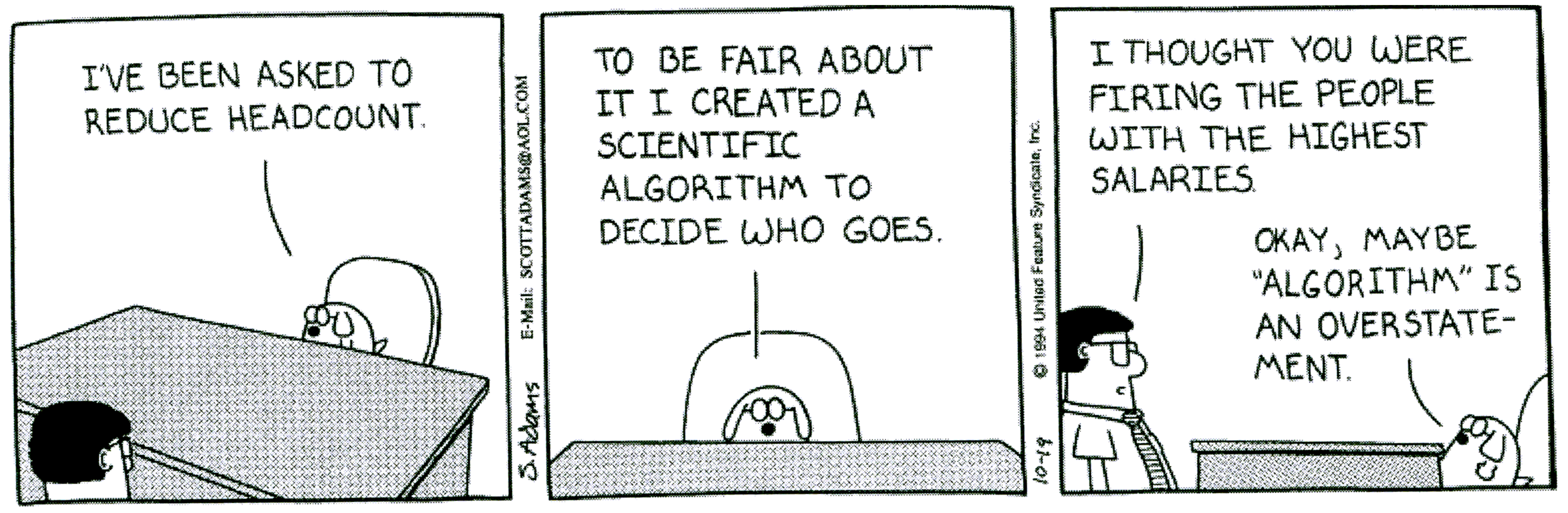


Properties of an algorithm

What should we be optimizing for?



- Computational efficiency
  - Speed, time, communication, etc.
  - Historically, important parameters because computers were slow and weak
  - Today, important parameters because computations are large
- Fairness
  - Does my algorithm have unintended consequences in its optimization?
  - Incredibly important as we apply algorithms in the real world
- In my research world of quantum computing
  - How much entanglement does my algorithm generate?
  - Is my algorithm error-robust? What kind of errors can it tolerate?



Dilbert by Scott Adams From the ClariNet electronic newspaper Redistribution prohibited info@clarinet.com

Course Logistics

Instructor

Chinmay Nirkhe **[he/him]**

`nirkhe@cs.washington.edu`

Speciality: **Complexity, Quantum**
Office: **CSE2 Room 217**
Office Hours: **Mondays 4:30 - 6:00pm**

The course staff

- **Head TAs:** Jay Dharmadhikari, Timothy Tran
  - Contact Head TAs with any questions, logistics, illnesses, etc.
- **TAs:** Siddharth Iyer, Oscar Sprumont, Yichuan Deng, Ajay Harilal, Jack Zhang, Owen Boseley, Shayla Huang

Day 1 checklist

- Find the course website. https://courses.cs.washington.edu/courses/cse421/
- **Read the syllabus**.
- Make sure you are on the following course resources: Gradescope, EdStem. Ask a TA for help, if you are not.
- **Attend your first section this week**.
- Problem set 1 is posted soon. You have the knowledge to start after section.
- Get all your credit: 4th credit available by signing up for 490D.

How to succeed in this class

- Attend lecture and ask questions/interact. Attendance is correlated strongly with final grade.
- I remember who asks questions, attends my office hours, and participates.
  - Can make the difference when I'm assigning grades if you are on the cusp.
  - It helps if I know who you are if you want me to write you a recommendation letter for a job or graduate school.
- Take lots of thinking walks and mull on these problems. *Being a computer scientist is different than being a programmer*. The same techniques may not apply.

Coursework

- 8 problem sets (roughly) due on Wednesdays at 11:59 PM
  - Typically 1 mechanical + 3 long-form problems
- See the homework guide on the website
- It is acceptable to talk to other students but write-ups must be done individually and *cannot* be shared/distributed
- Use of outside resources (including generative AI) is forbidden. **Read the syllabus and problem set guide**. Ask TAs if you have *any* questions.

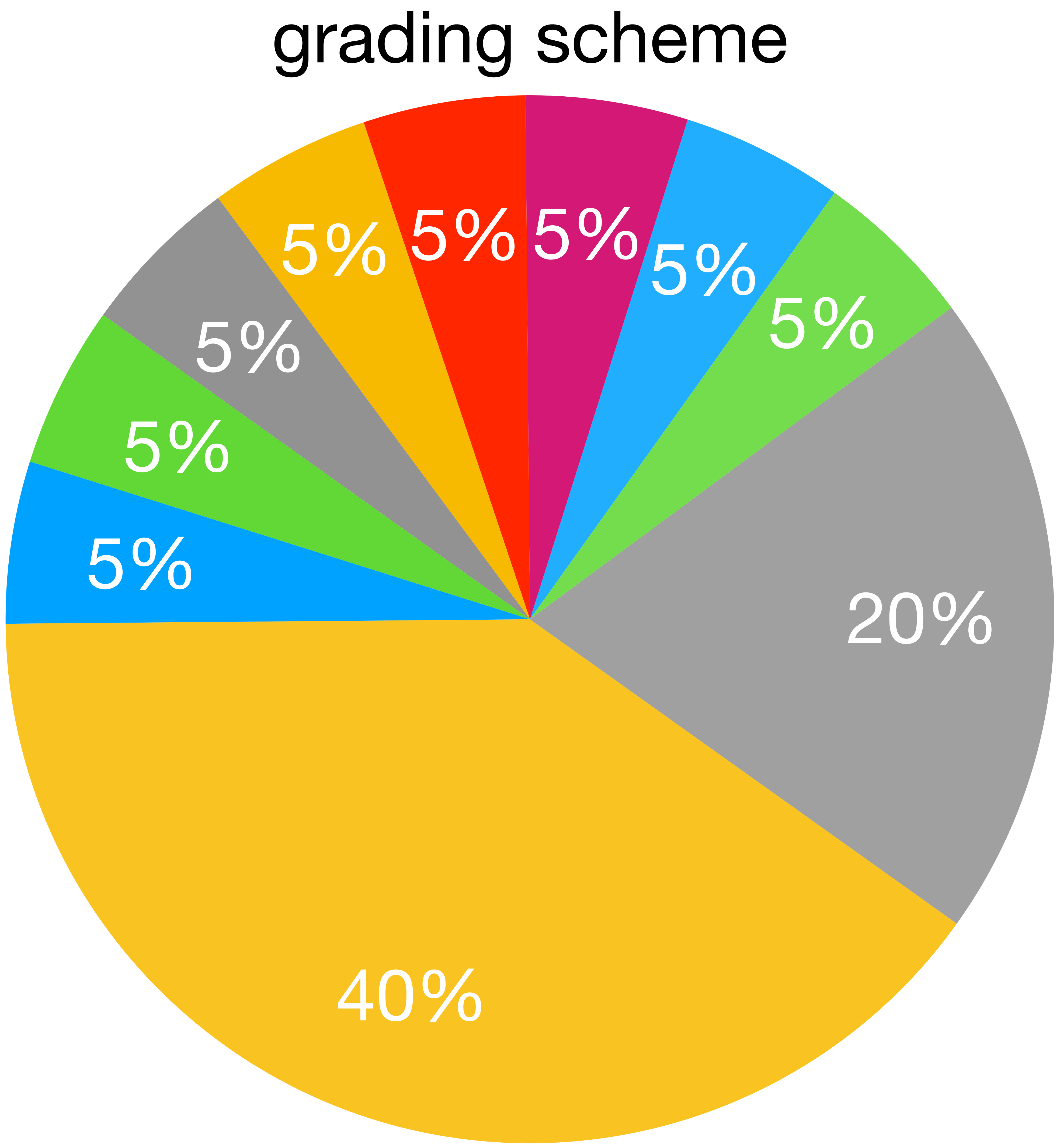
Late Problem Sets

You are allocated four 24-hour extensions on sets.

- No questions asked, no reason required.
- You may **not** use more than one extension on any problem set.
- After the four extensions are exhausted,
    - problem sets up to 24-hours late are graded with a 25% penalty.
    - problem sets up to 48-hours late are graded with a 50% penalty.
    - problem set after 48-hours are given automatic zeros.
- Extenuating circumstances can be discussed with Prof. Nirkhe, no TA can approve additional extensions.

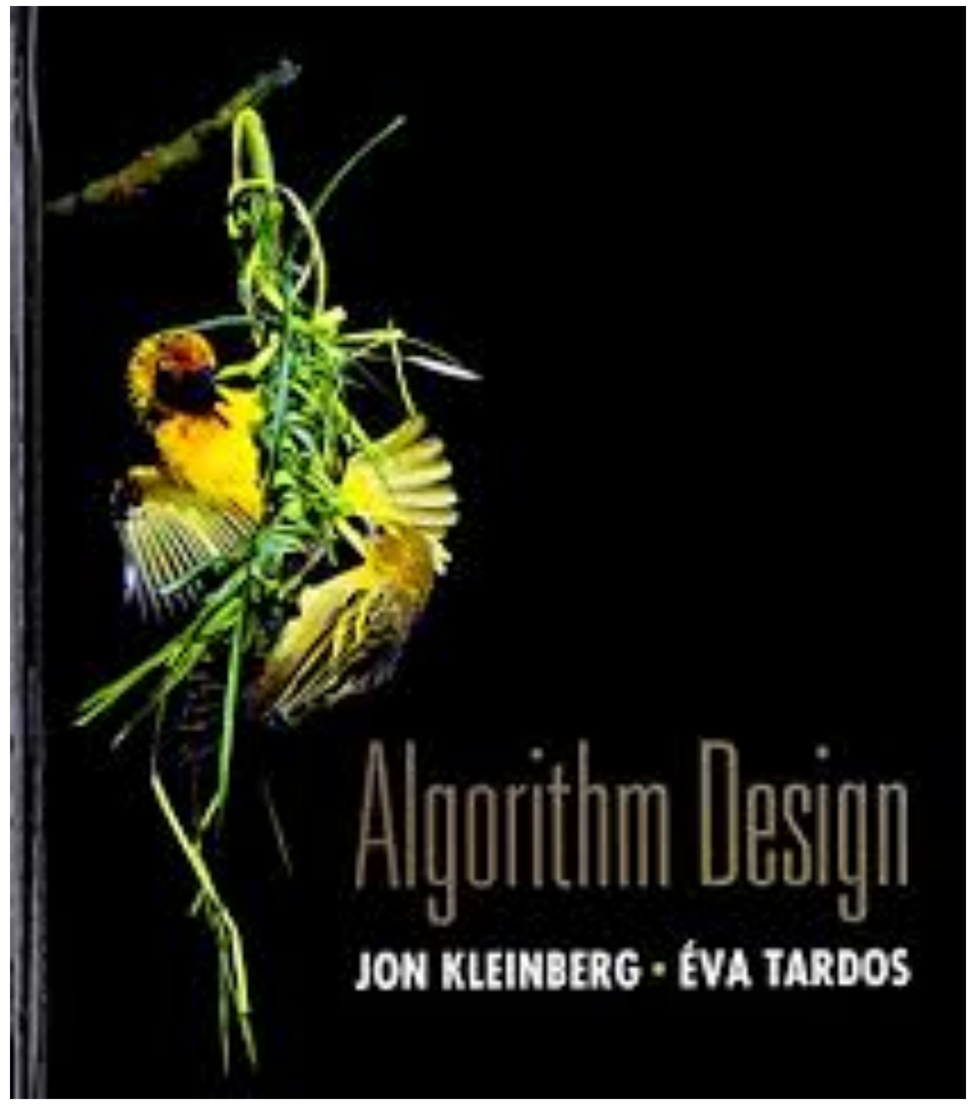
Exams and grading scheme

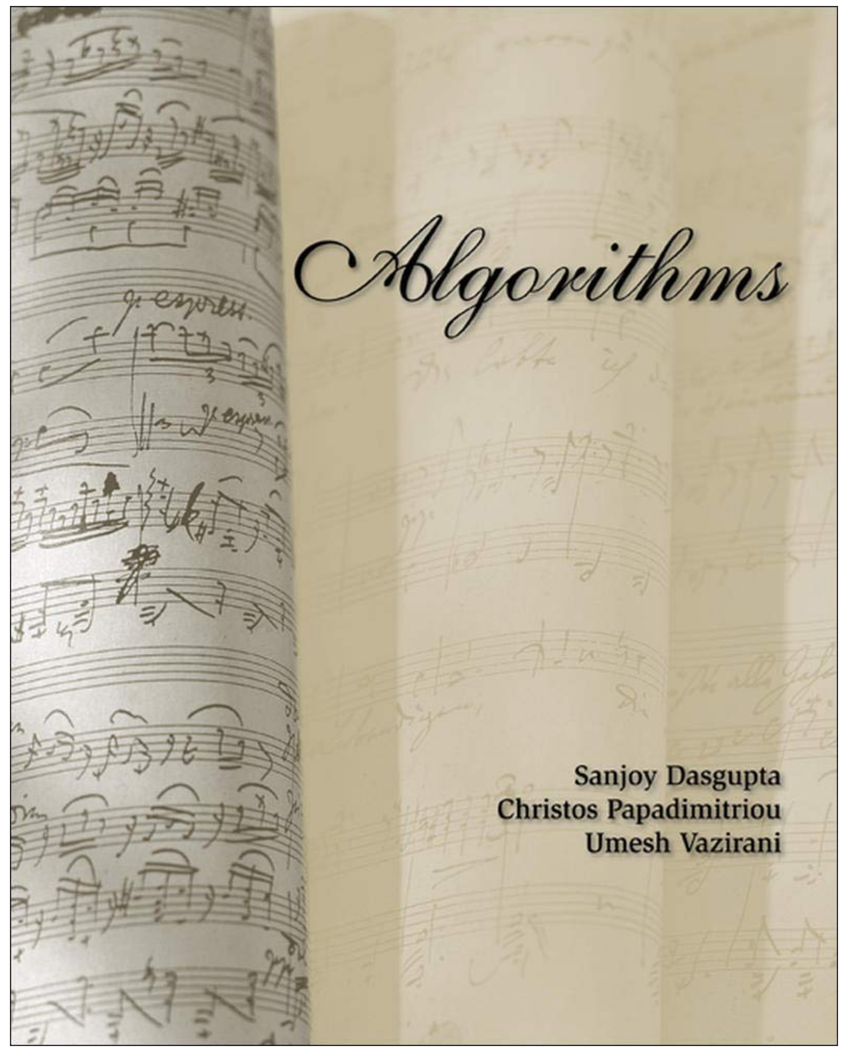
- The dates are set. Due to the size of the class, no late exam exceptions will be made. Extenuating circumstances should be discussed with Prof. Nirkhe only.
- Midterm: Monday May 5th 3:30-4:20pm
- Final: Thursday June 12th 2:30-4:20pm
- Grading scheme (approx.):
  - 8 problem sets: 40%
  - Midterm: 20%
  - Final: 40%



Textbook

- There are two suggested textbooks for the course.
- Both are page-turners and are great for learning how to think like an algorithm designer.
- They are also not required — all required content can be extracted from the lecture notes and quiz section materials.

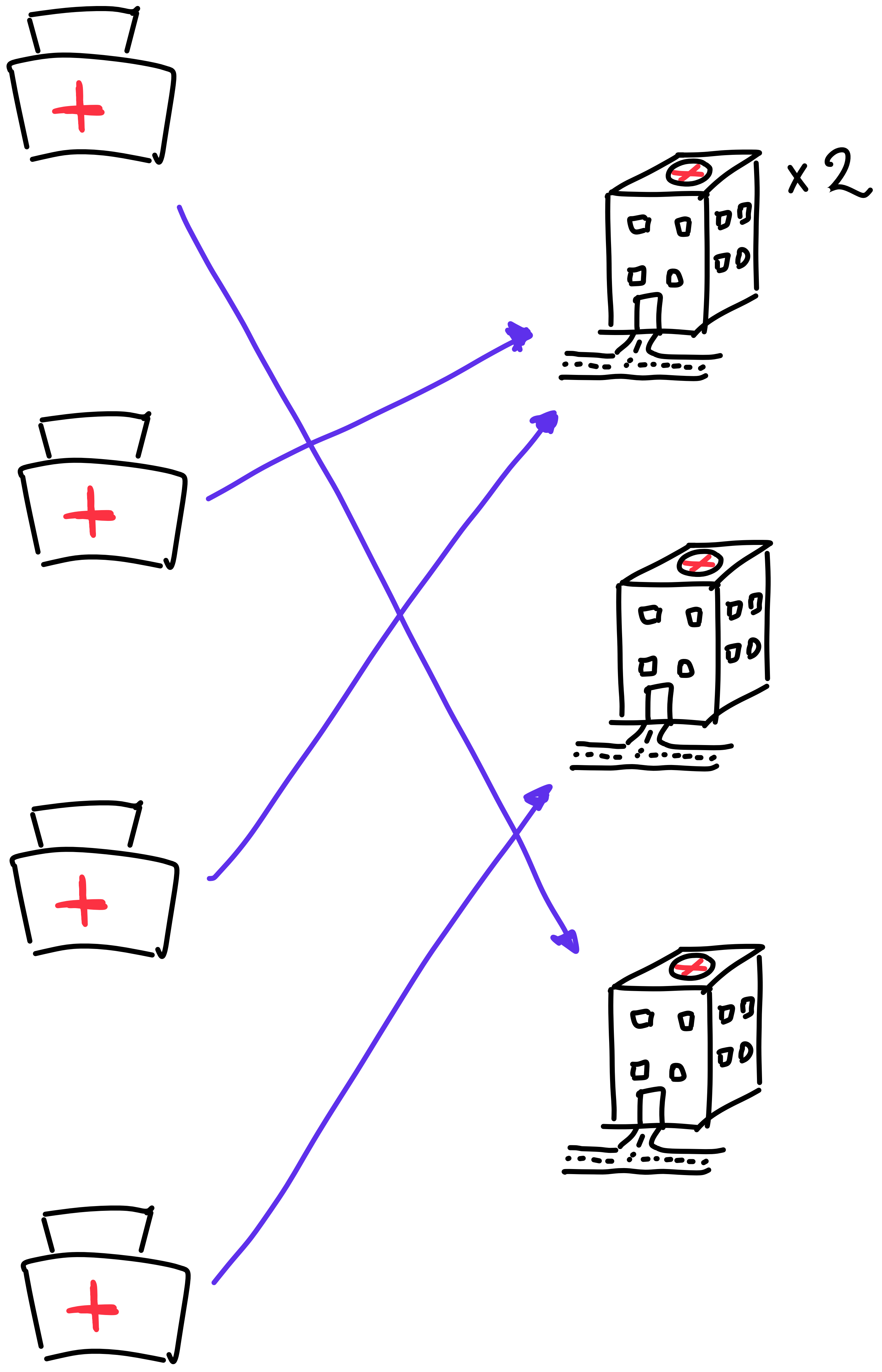




The Stable Matching Algorithm

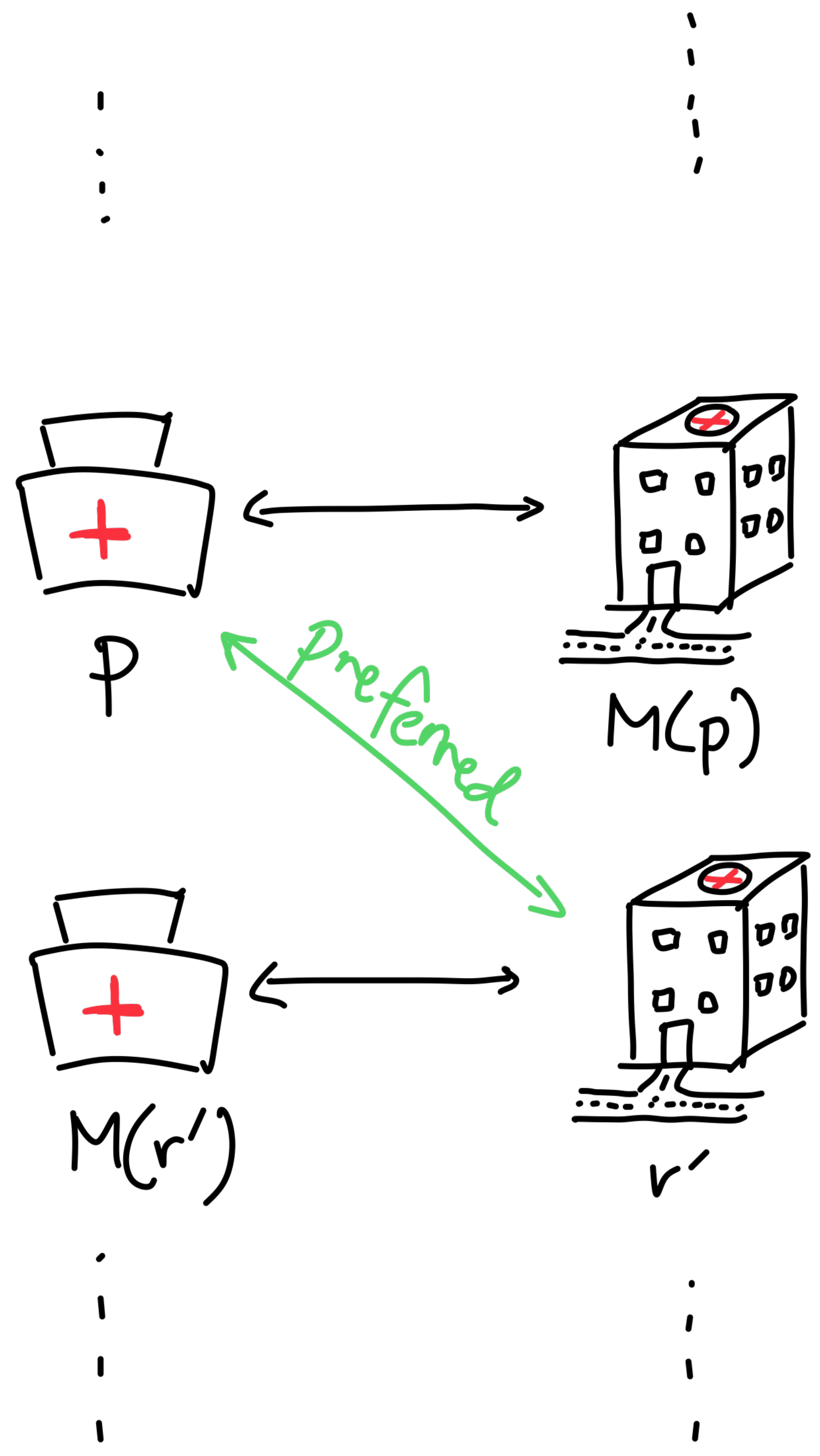
The matching problem

- **Goal**: Given a set of preferences amongst hospital and residents, design an admissions process to allocate residents to hospitals.
- What might we want to optimize for?
- When do we know we have achieved the optimal solution?
- What properties does our optimal solution have?



A notion of *stability*

- Lets assume there are n residents and n hospitals for now.
- A matching M is n disjoint pairs (p, r) assigning hospital r to resident p.
- A resident-hospital pair (resident p, hospital r') is **unstable** for M if both
  - resident p prefers hospital r' to their assigned hospital $M(p)$.
  - hospital r' prefers resident p to their assigned resident $M(r')$.
- A matching is **stable** if the matching has no **unstable** pairs.
  - Natural and desirable condition. *Self-interest* will prevent side-deals from being made.



Cartoon of unstable (p, r')

Can we design an algorithm to find a stable matching?

And does a stable matching necessarily exist?

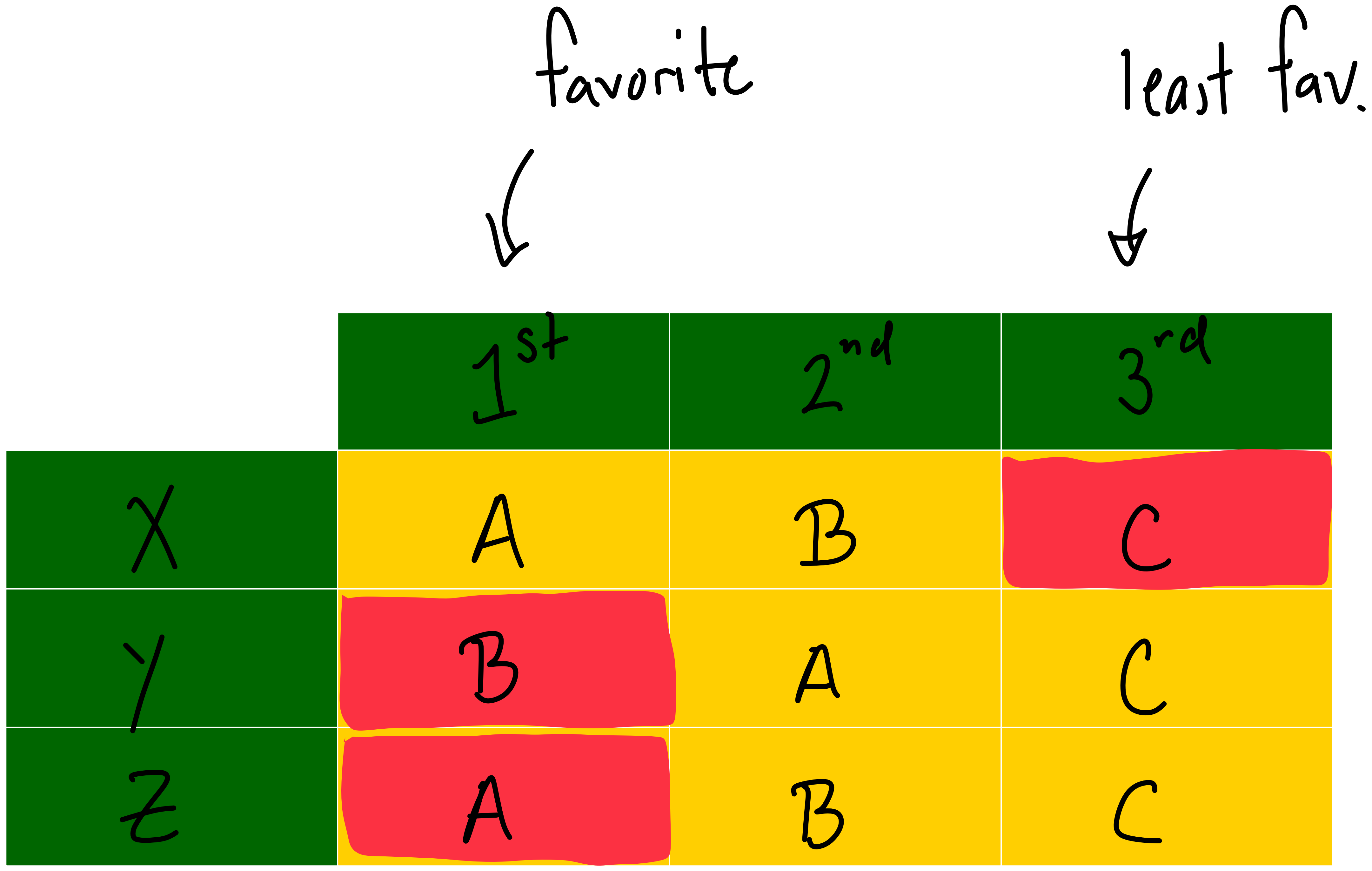
- **Input to the problem**:
    - Two groups of n people: one group P and the other group R.
    - For each $p \in P$, a ranking from 1 to n of the group R.
    - For each $r \in R$, a ranking from 1 to n of the group P.
- **Output of the problem**:
    - A list of n disjoint pairs M. The matching should be stable with respect to the input rankings.

favorite ↓ least fav. ↓

	1st	2nd	3rd
X	A	B	C
Y	B	A	C
Z	A	B	C

	1st	2nd	3rd
A	Y	X	Z
B	X	Y	Z
C	X	Y	Z

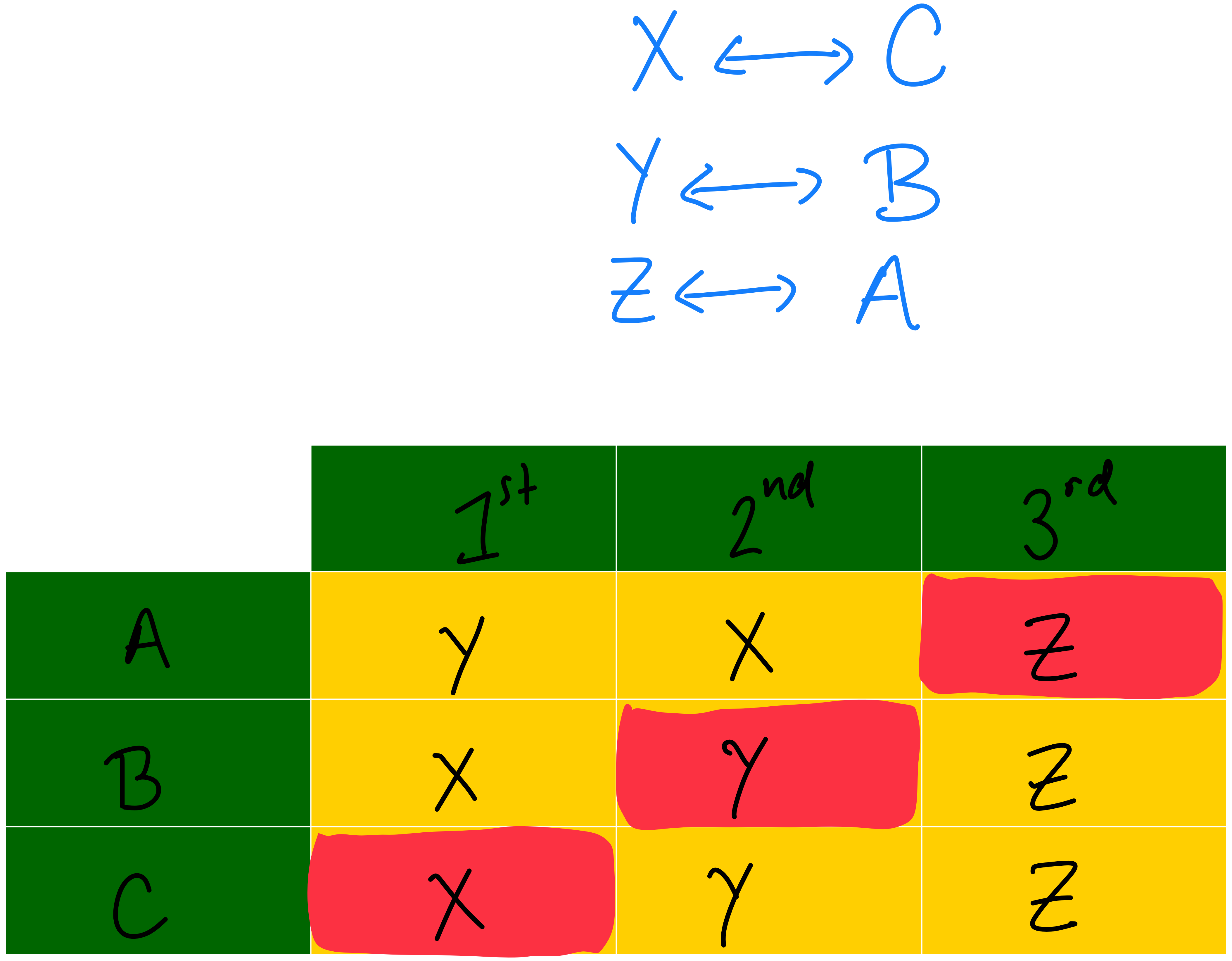
Example 1: Is the following matching stable?



favorite

least fav.

	1st	2nd	3rd
X	A	B	C
Y	B	A	C
Z	A	B	C



X ↔ C
Y ↔ B
Z ↔ A

	1st	2nd	3rd
A	Y	X	Z
B	X	Y	Z
C	X	Y	Z

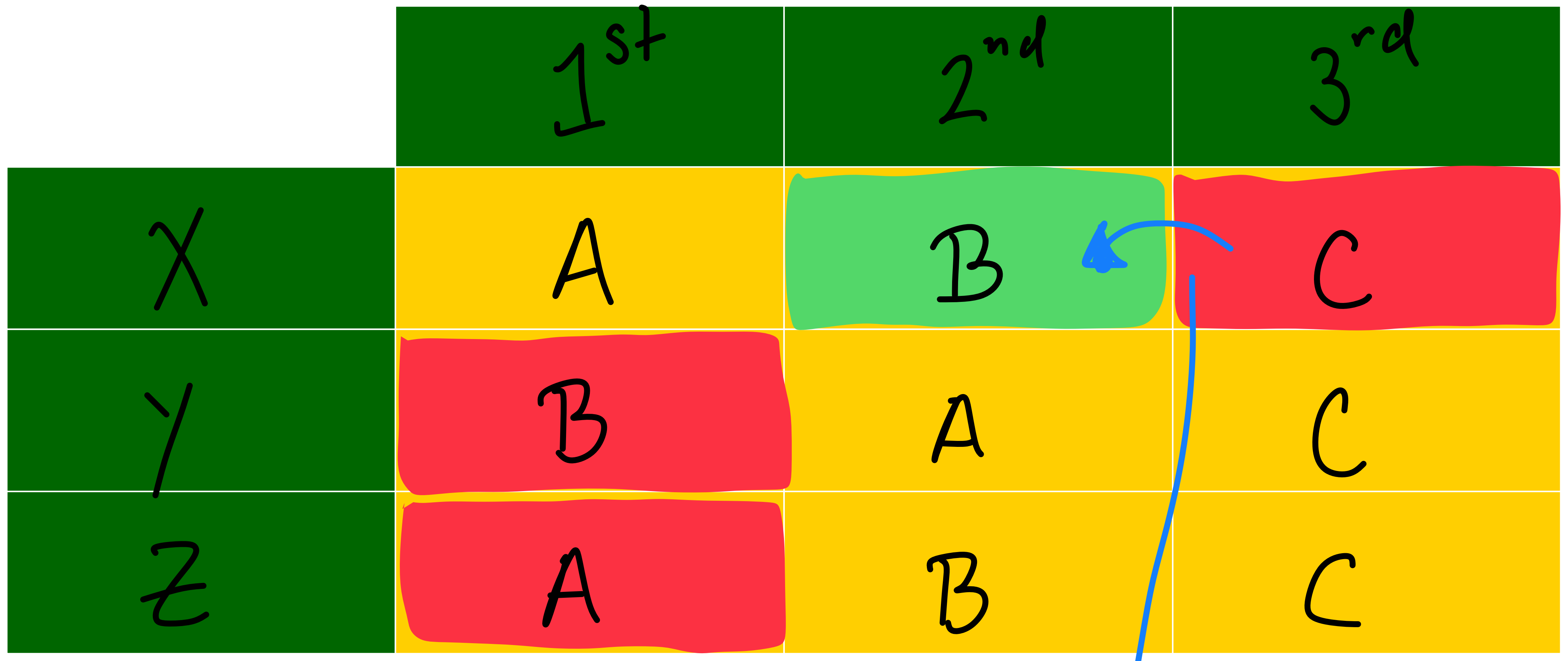
Example 1: Is the following matching stable?

No.

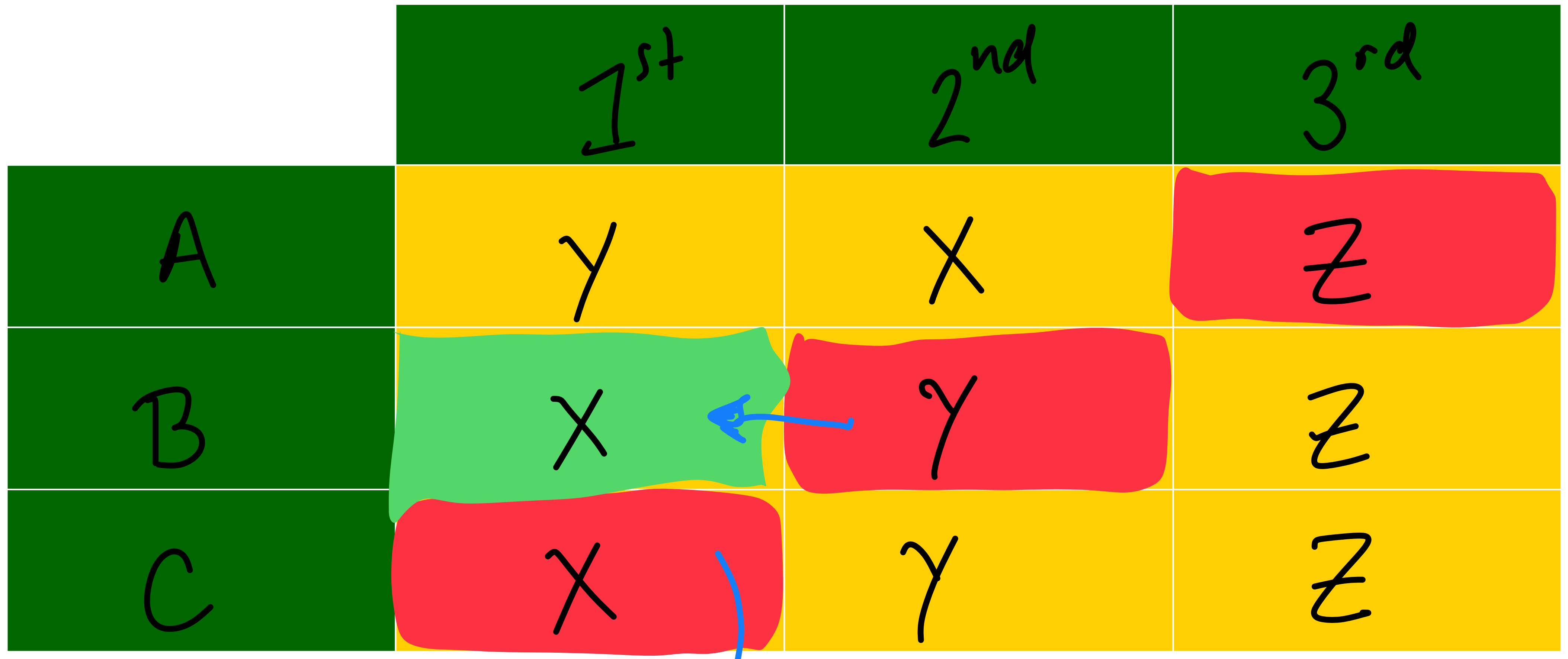
X ↔ C
Y ↔ B
Z ↔ A

favorite ↓

least fav. ↓



	1st	2nd	3rd
X	A	B	C
Y	B	A	C
Z	A	B	C



	1st	2nd	3rd
A	Y	X	Z
B	X	Y	Z
C	X	Y	Z

mutually preferred change

Example 2: Is the following matching stable?

X ⟷ A
Y ⟷ B
Z ⟷ C

favorite ↓

least fav. ↓

	1st	2nd	3rd
X	A	B	C
Y	B	A	C
Z	A	B	C

	1st	2nd	3rd
A	Y	X	Z
B	X	Y	Z
C	X	Y	Z

Example 2: Is the following matching stable?

YES.

X ↔ A
Y ↔ B
Z ↔ C

favorite ↓

least fav. ↓

	1st	2nd	3rd
X	A	B	C
Y	B	A	C
Z	A	B	C

	1st	2nd	3rd
A	Y	X	Z
B	X	Y	Z
C	X	Y	Z

The propose and reject algorithm

Gale & Shapley 1962

The group P proposes and the group R receives

```
Initialize each person to be free.
while (some p in P is free) {
    Choose some free p in P
    r = 1st person on p's preference list to whom p has not yet proposed
    if (r is free)
        tentatively match (p,r)   //p and r both engaged, no longer free
    else if (r prefers p to current tentative match p′)
        replace (p′,r) by (p,r)   //p now engaged, p′ now free
    else
        r rejects p
}
```

The propose and reject algorithm

Proof of termination

Observation 1: Every $p \in P$ proposes in decreases order of preference.

Observation 2: No proposal (p, r) is ever repeated.

Conclusion: Since there are only n^2 pairs (p, r), algorithm terminates after $\leq n^2$ iterations of the while loop.

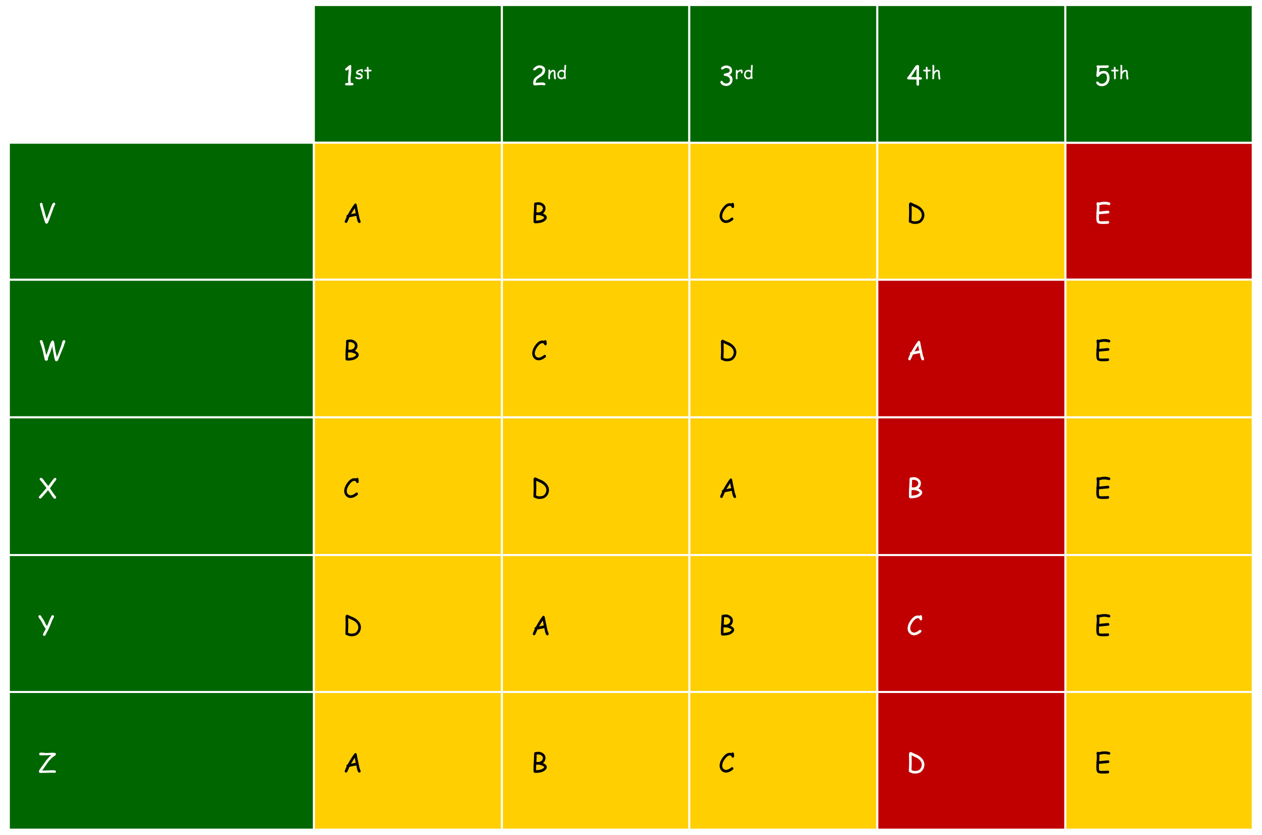
And indeed, it can take this long for many simple examples.

```
Initialize each person to be free.
while (some p in P is free) {
    Choose some free p in P
    r = 1st person on p's preference list to whom p has not yet proposed
    if (r is free)
        tentatively match (p,r)    //p and r both engaged, no longer free
    else if (r prefers p to current tentative match p')
        replace (p',r) by (p,r)    //p now engaged, p' now free
    else
        r rejects p
}
```

The propose and reject algorithm

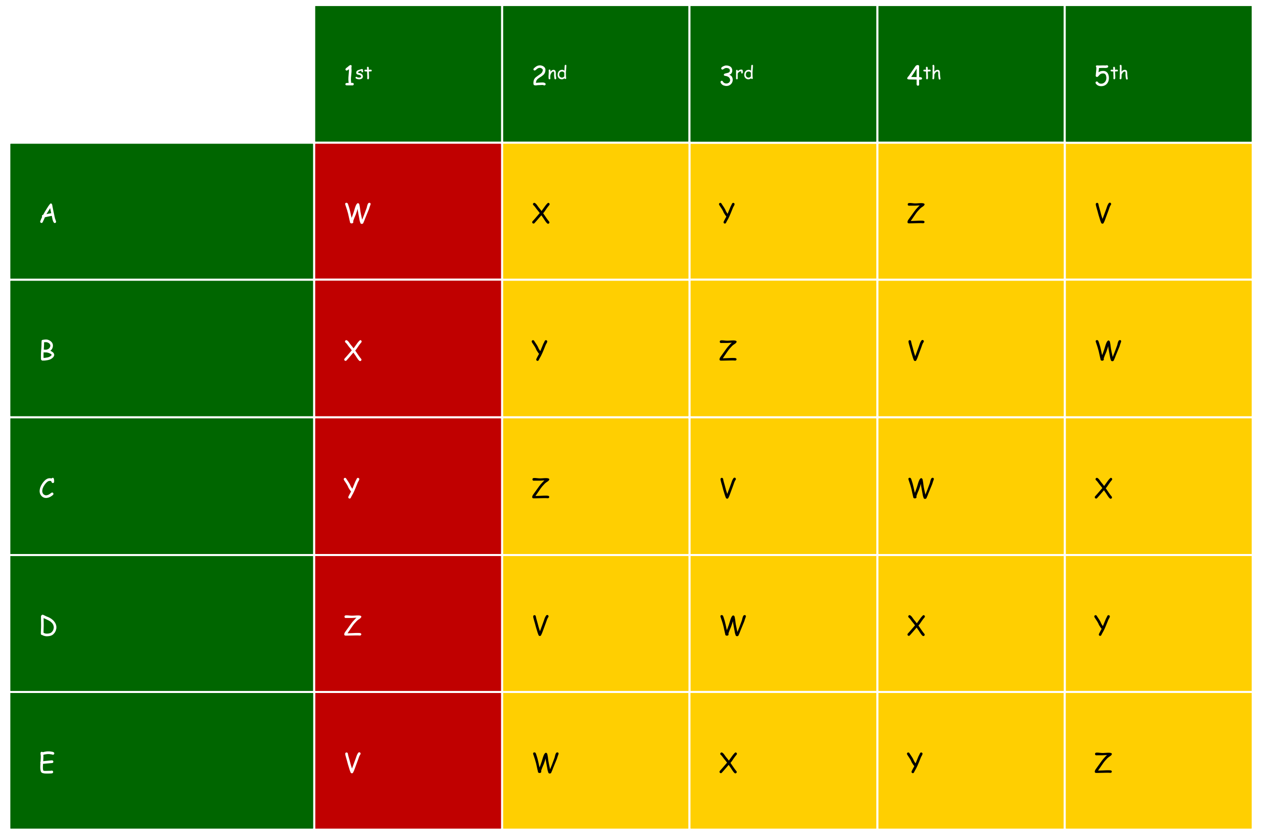
Proof of termination

This example takes
$n(n-1)+1$ iterations.



	1st	2nd	3rd	4th	5th
V	A	B	C	D	E
W	B	C	D	A	E
X	C	D	A	B	E
Y	D	A	B	C	E
Z	A	B	C	D	E

Preference Profile for P



	1st	2nd	3rd	4th	5th
A	W	X	Y	Z	V
B	X	Y	Z	V	W
C	Y	Z	V	W	X
D	Z	V	W	X	Y
E	V	W	X	Y	Z

Preference Profile for R

And indeed, it can take this long for many simple examples.

```
Initialize each person to be free.
while (some p in P is free) {
    Choose some free p in P
    r = 1st person on p's preference list to whom p has not yet proposed
    if (r is free)
        tentatively match (p,r)    //p and r both engaged, no longer free
    else if (r prefers p to current tentative match p')
        replace (p',r) by (p,r)    //p now engaged, p' now free
    else
        r rejects p
}
```

The propose and reject algorithm

Proof of perfection

Observation 3: One a receiver r is matched, they are never freed up. If anything, w.r.t. their preferences, they only ever trade up.

Claim: By the time the algorithm terminates, everyone gets matched.

Proof:

- Since $|P| = |R| = n$, if no receiver is free, then everyone is matched.
- If some $p \in P$ proposes to their last choice receiver r_n, then all previous receivers r must have already been matched. Then (p, r_n) matching is added and no receiver is free.

The propose and reject algorithm

Proof of stability

Claim: The final matching M of the algorithm does not have *unstable* pairs

Proof: Consider a pair (p, r) that is *not* matched by M: $M(p) \neq r$.

- Case 1: During the entire algorithm run, p *never* proposed to r.

- Case 2: Or at some time, p proposed to r.

The propose and reject algorithm

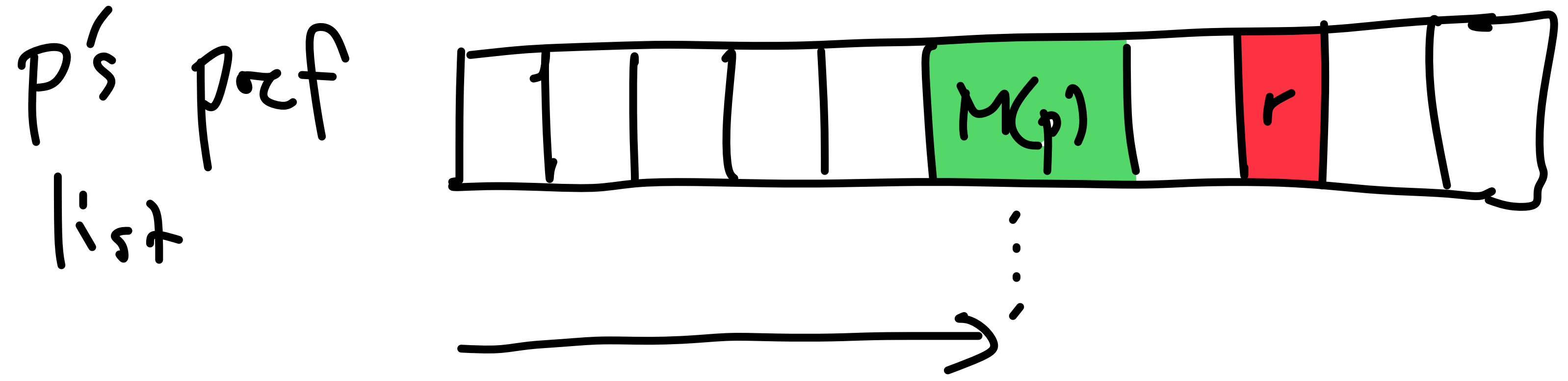
Proof of stability

Claim: The final matching M of the algorithm does not have *unstable* pairs

Proof: Consider a pair (p, r) that is *not* matched by M: $M(p) \neq r$.

- Case 1: During the entire algorithm run, p *never* proposed to r.
  - Therefore, p prefers $M(p)$ to r. So (p, r) is *not* unstable w.r.t. M.
- Case 2: Or at some time, p proposed to r.
  - Therefore, r prefers $M(r)$ to p. So (p, r) is *not* unstable w.r.t. M.

Case 1:



kept proposing until eventually terminated at M(p)

M(p) is preferred to r by p.

So not unstable.

The propose and reject algorithm

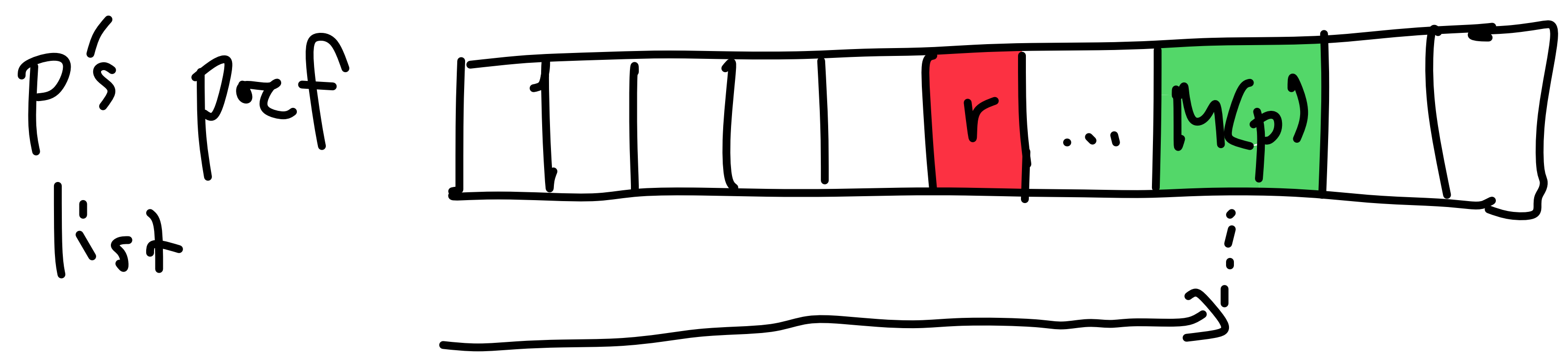
Proof of stability

Claim: The final matching M of the algorithm does not have *unstable* pairs

Proof: Consider a pair (p, r) that is *not* matched by M: $M(p) \neq r$.

- Case 1: During the entire algorithm run, p *never* proposed to r.
  - Therefore, p prefers $M(p)$ to r. So (p, r) is *not* unstable w.r.t. M.
- Case 2: Or at some time, p proposed to r.
  - Therefore, r prefers $M(r)$ to p. So (p, r) is *not* unstable w.r.t. M.

Case 2:



kept proposing until eventually terminated at M(p)

So, r was proposed to by p,

But r eventually rejected p meaning r has "traded up" to get M(r).

So r prefers M(r) to p.

The propose and reject algorithm

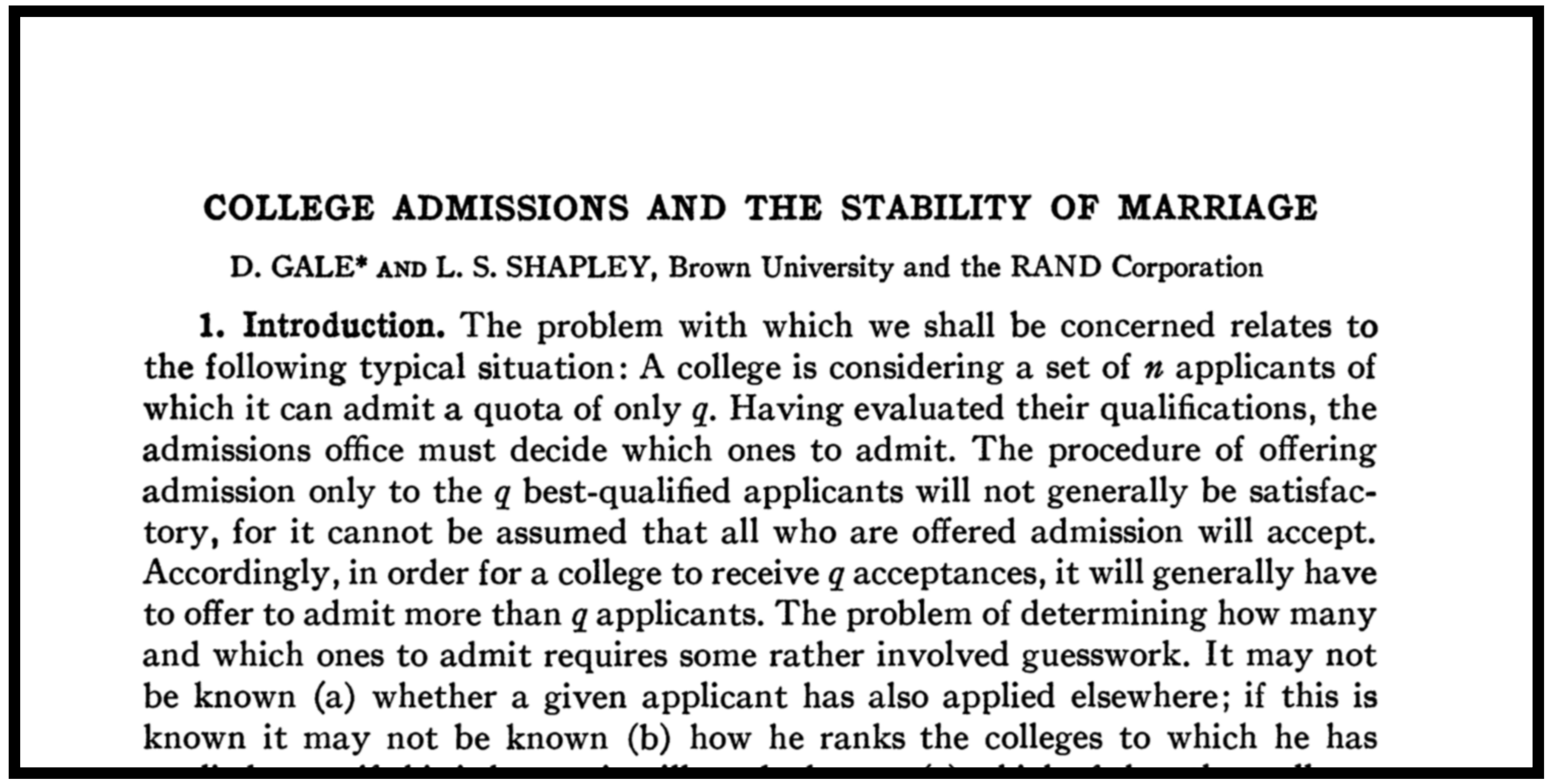
What have we learned?

- Proof of termination in n^2 iterations. ✓
- Proof of perfection: everyone gets matched. ✓
- Proof of stability: the output matching is stable for all pairs. ✓
- What have we not talked about?
  - Is it fair? Is it better to be a proposer or a receiver? Does the first proposer or the last proposer have it better?
  - Is there a faster algorithm?
  - How do we extend to n proposers and n' receivers?

The history of the propose and reject algorithm

Gale and Shapley 1962

- The original paper was about n men and n women and a heterosexual notion of marriage.
- Gale and Shapley's algorithm defined the proposers as the men and the receivers as the women.
    - We will see next week that the GS algorithm is **proposer**-optimal but not **receiver**-optimal.
    - For obvious reasons, we changed the notation.
    - As originally stated, the GS algorithm favored being a man. This social implication was not recognized for some time!
- Is fairness possible? In some cases, yes. But this is an active area of research!



COLLEGE ADMISSIONS AND THE STABILITY OF MARRIAGE

D. GALE* AND L. S. SHAPLEY, Brown University and the RAND Corporation

1. Introduction. The problem with which we shall be concerned relates to the following typical situation: A college is considering a set of n applicants of which it can admit a quota of only q. Having evaluated their qualifications, the admissions office must decide which ones to admit. The procedure of offering admission only to the q best-qualified applicants will not generally be satisfactory, for it cannot be assumed that all who are offered admission will accept. Accordingly, in order for a college to receive q acceptances, it will generally have to offer to admit more than q applicants. The problem of determining how many and which ones to admit requires some rather involved guesswork. It may not be known (a) whether a given applicant has also applied elsewhere; if this is known it may not be known (b) how he ranks the colleges to which he has

Shapley winning the 2012 Economics Nobel Prize (with Roth)