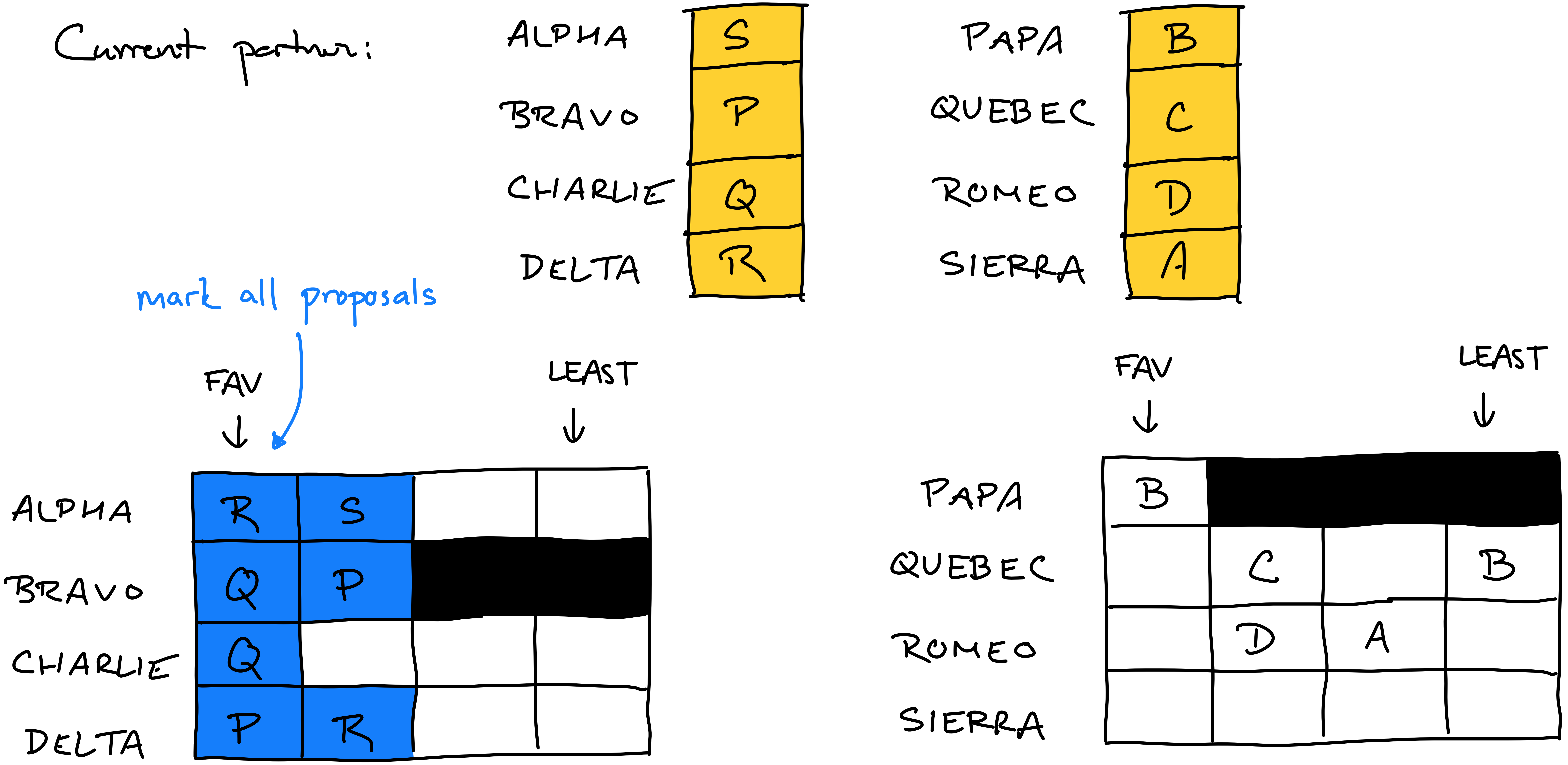
Lecture 11

Stable matching II and midterm review

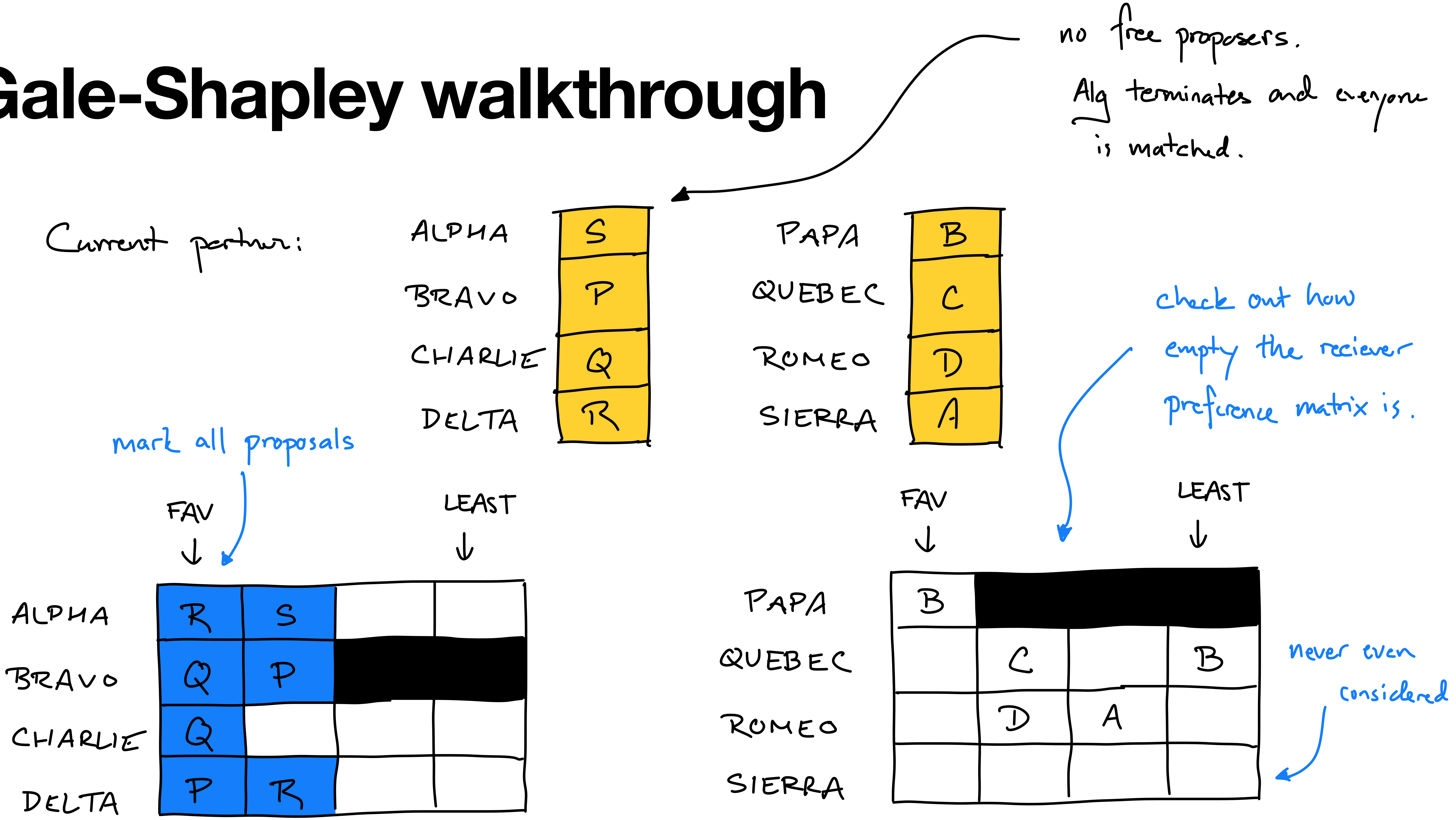
Chinmay Nirkhe | CSE 421 Winter 2026

Gale-Shapley walkthrough

```
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
}
```



Gale-Shapley walkthrough



Gale-Shapley walkthrough

no free proposers.
Alg terminates and everyone is matched.

Current partner:

ALPHA	S
BRAVO	P
CHARLIE	Q
DELTA	R

PAPA	B
QUEBEC	C
ROMEO	D
SIERRA	A

mark all proposals

	FAV			LEAST
ALPHA	R	S		
BRAVO	Q	P	[redacted]	
CHARLIE	Q			
DELTA	P	R		

	FAV			LEAST
PAPA	B	[redacted]		
QUEBEC		C		B
ROMEO		D	A	
SIERRA				

Gale-Shapley walkthrough

```
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
}
```

Current partner:

Is (A,R) stable?

ALPHA	S
BRAVO	P
CHARLIE	Q
DELTA	R

PAPA	B
QUEBEC	C
ROMEO	D
SIERRA	A

FAV ↓ … LEAST ↓

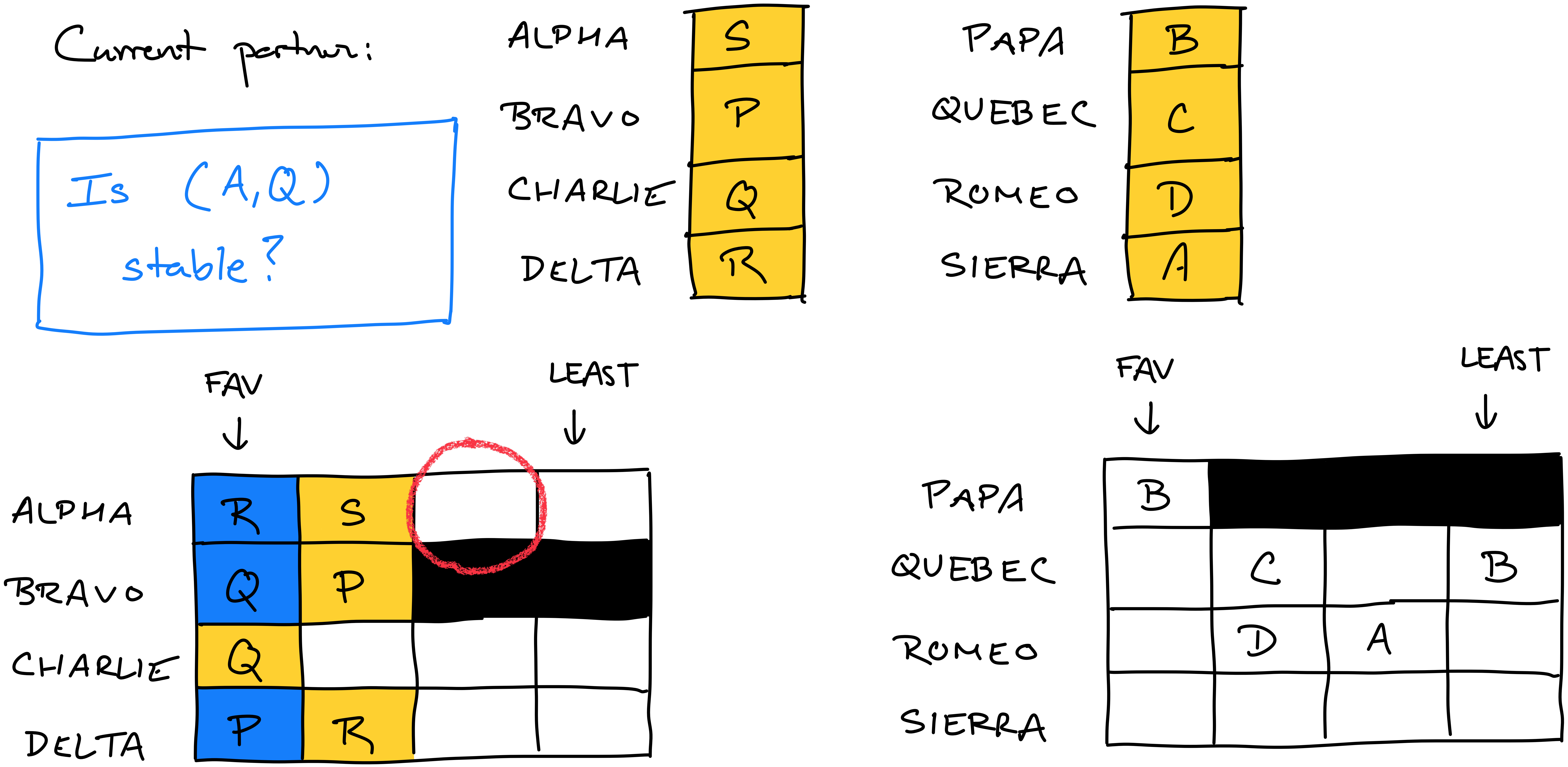
ALPHA	R	S		
BRAVO	Q	P	[redacted]	
CHARLIE	Q			
DELTA	P	R		

FAV ↓ … LEAST ↓

PAPA	B	[redacted]		
QUEBEC		C		B
ROMEO		D	A	
SIERRA				

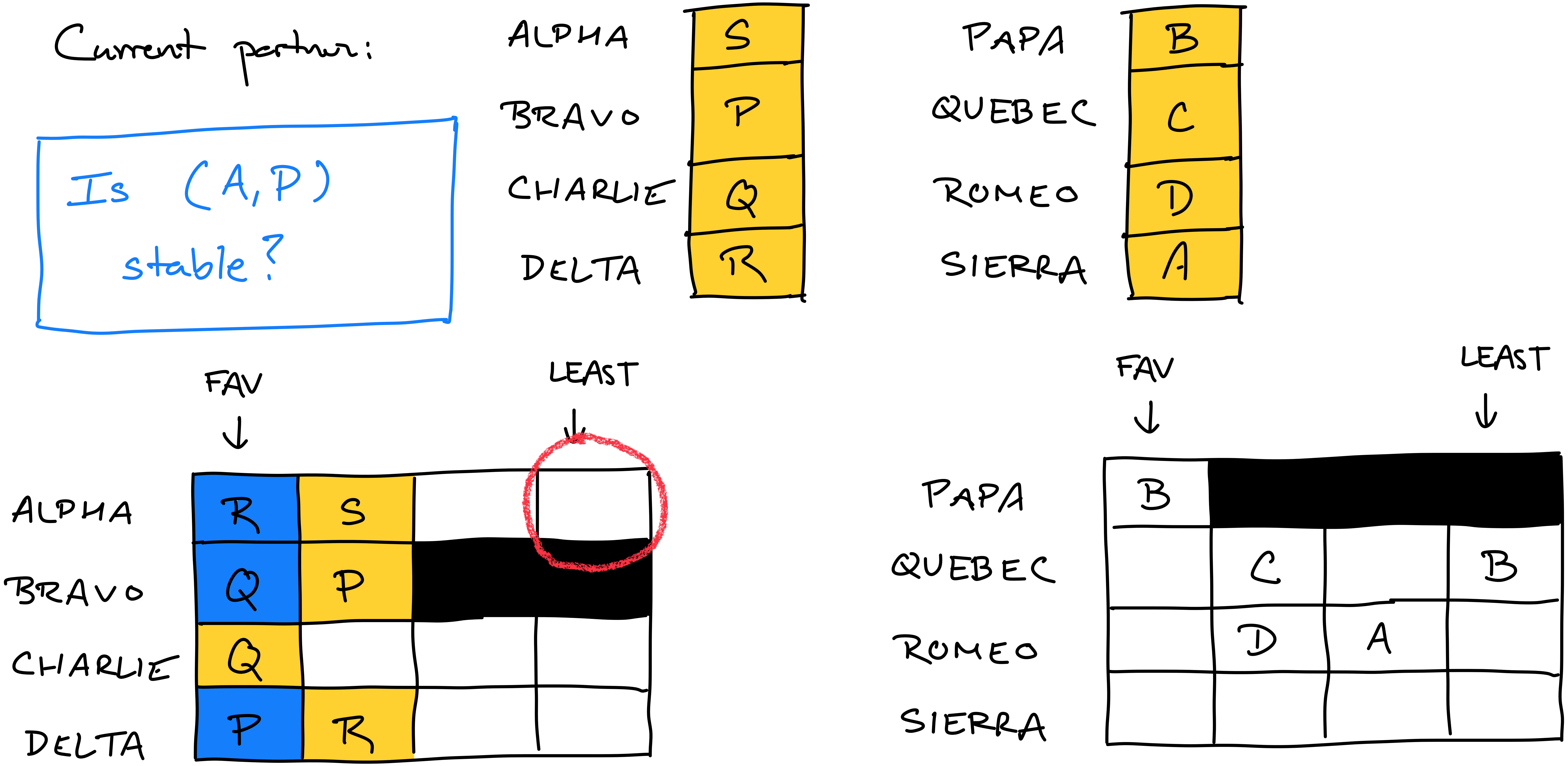
Gale-Shapley walkthrough

```
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
}
```



Gale-Shapley walkthrough

```
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

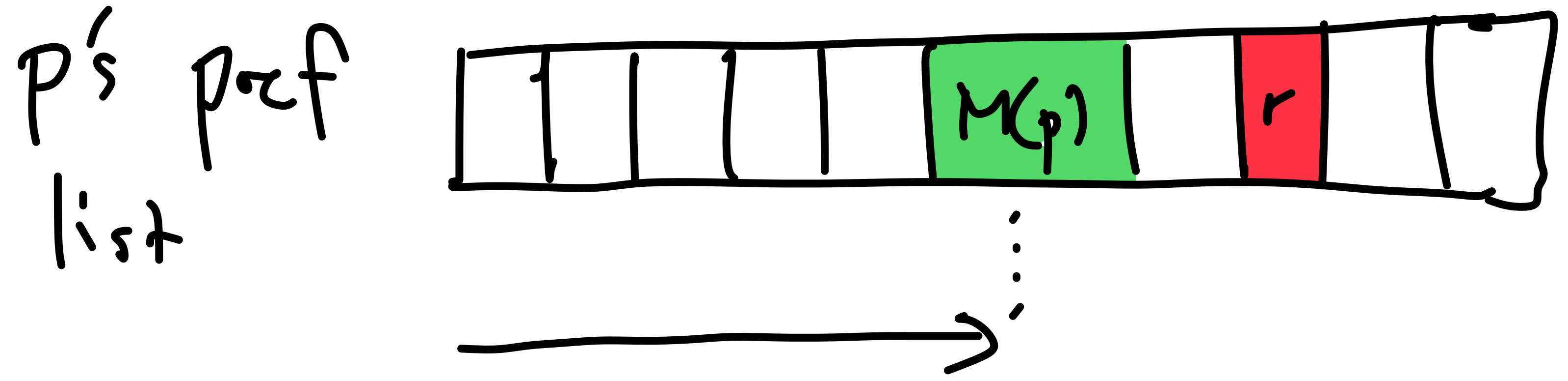
Proof of stability

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

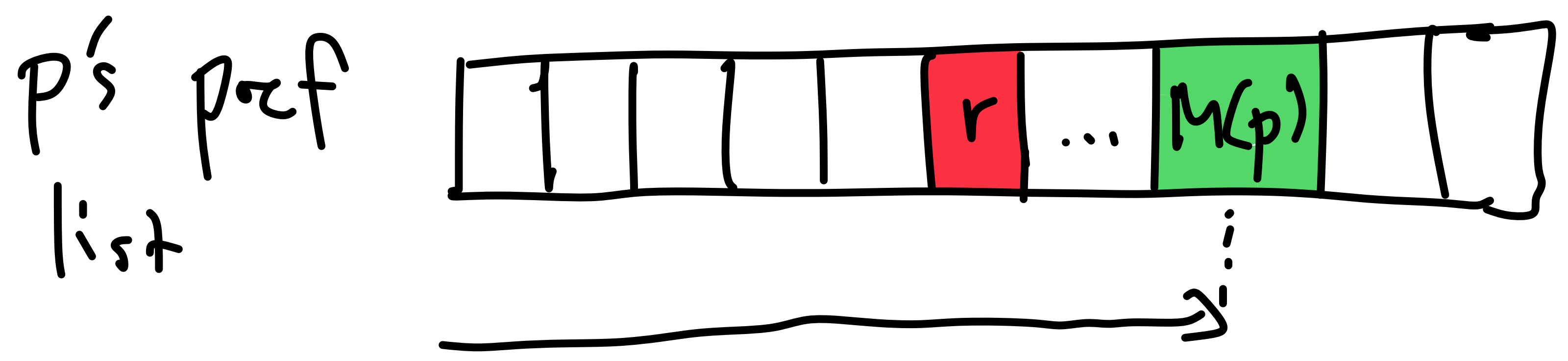
Proof of stability

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

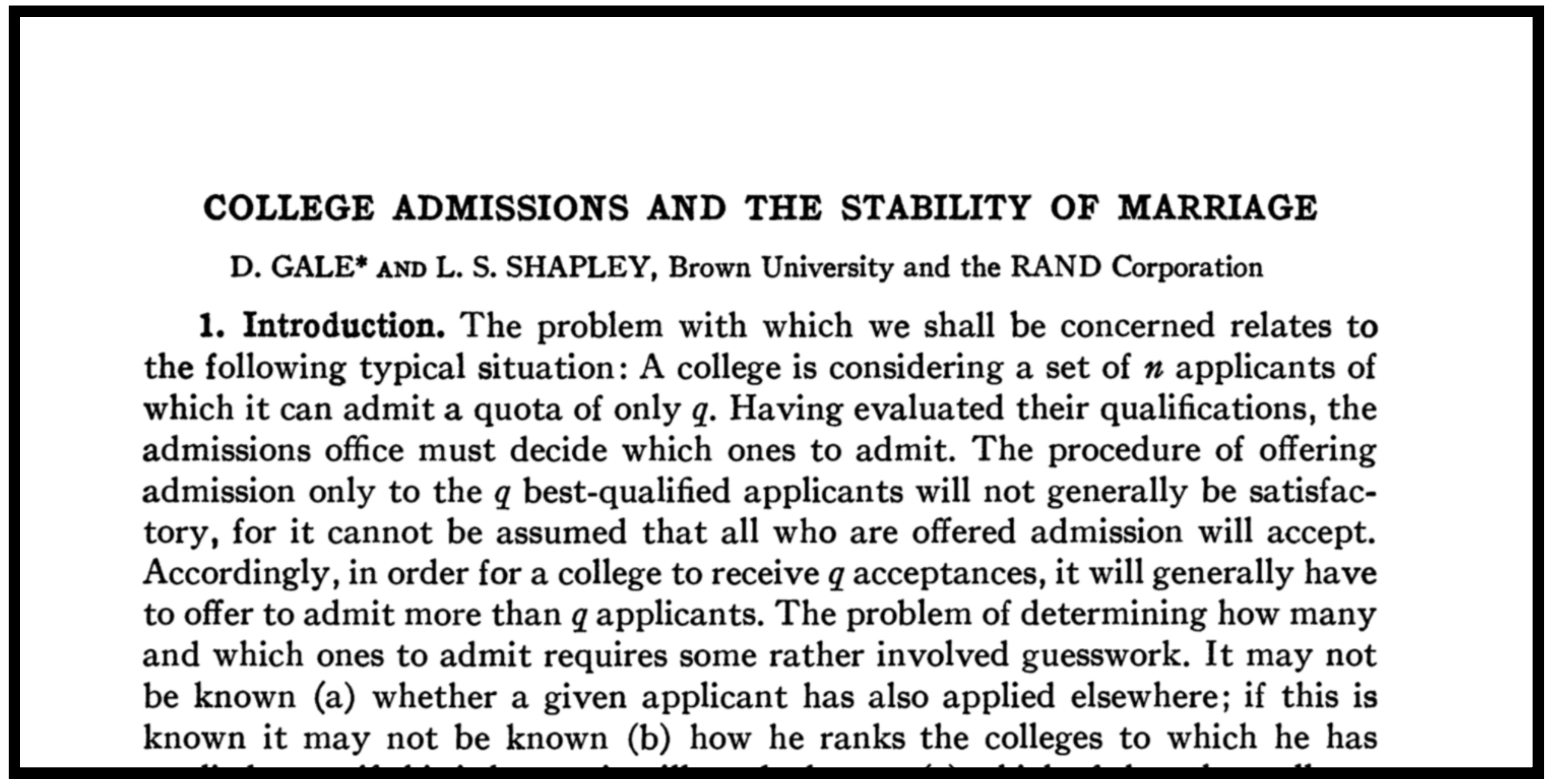
What have we learned?

- 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 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)

Implementing stable matching

- Input length
  - $N := 2n^2$ words in length because $2n$ people × preference list of length n.
  - A "word" here is a number $\in [n] = \{1,2,\ldots,n\}$. Takes $\lceil \log_2 n \rceil$ bits to represent.
  - Input length of $2n^2 \lceil \log_2 n \rceil$ bits.
- **Brute force algorithm**: Try all $n!$ possible matchings. Testing if a matching is stable requires testing if each of the n^2 pairs (p, r) is stable.
- **Gale-Shapley algorithm**: takes $\leq n^2$ iterations. How long does each iteration take to run?

Implementing Gale-Shapley in $O(n^2)$ time
Comparing

- **Input**: 2 $n \times n$ representing the preferences of P and R:
  - $\text{pref}_P[p][j], \text{pref}_R[r][j]$
  - Assume the proposers and receivers are numbers $1,2,\ldots,n$
  - Each preference array is a *permutation* of $\{1,2,\ldots,n\}$
- **Data structure for the matching**:
  - Maintain two arrays $M_P[p]$ and $M_R[r]$ denoting match of p and r
  - Initialize both arrays to all $\perp$, a symbol denoting that the match isn't set
  - If during the algorithm, (p, r) is matched, set $M_P[p] \leftarrow r, M_R[r] \leftarrow p$
- **Making proposals**:
  - Maintain a queue Q of all the free proposers. Initially Q contains all n proposers.
  - Maintain an array $\text{count}[p]$ which counts how many proposals p has made so far. Initially all entries are 0.

```
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
}
```

Implementing Gale-Shapley in $O(n^2)$ time

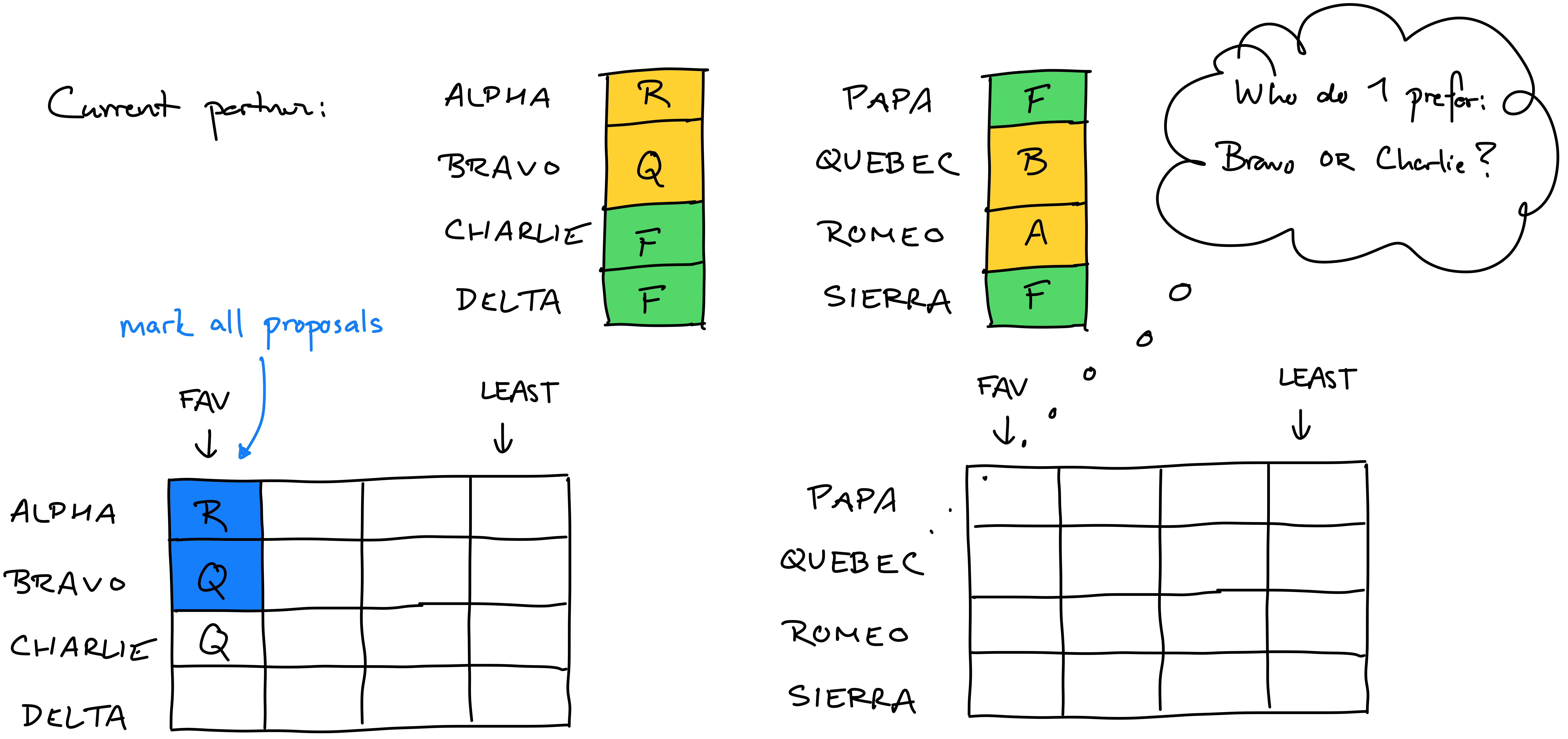
Rejecting & accepting proposals

- How do we decide efficiently if receiver r prefers proposer p to proposer p'?

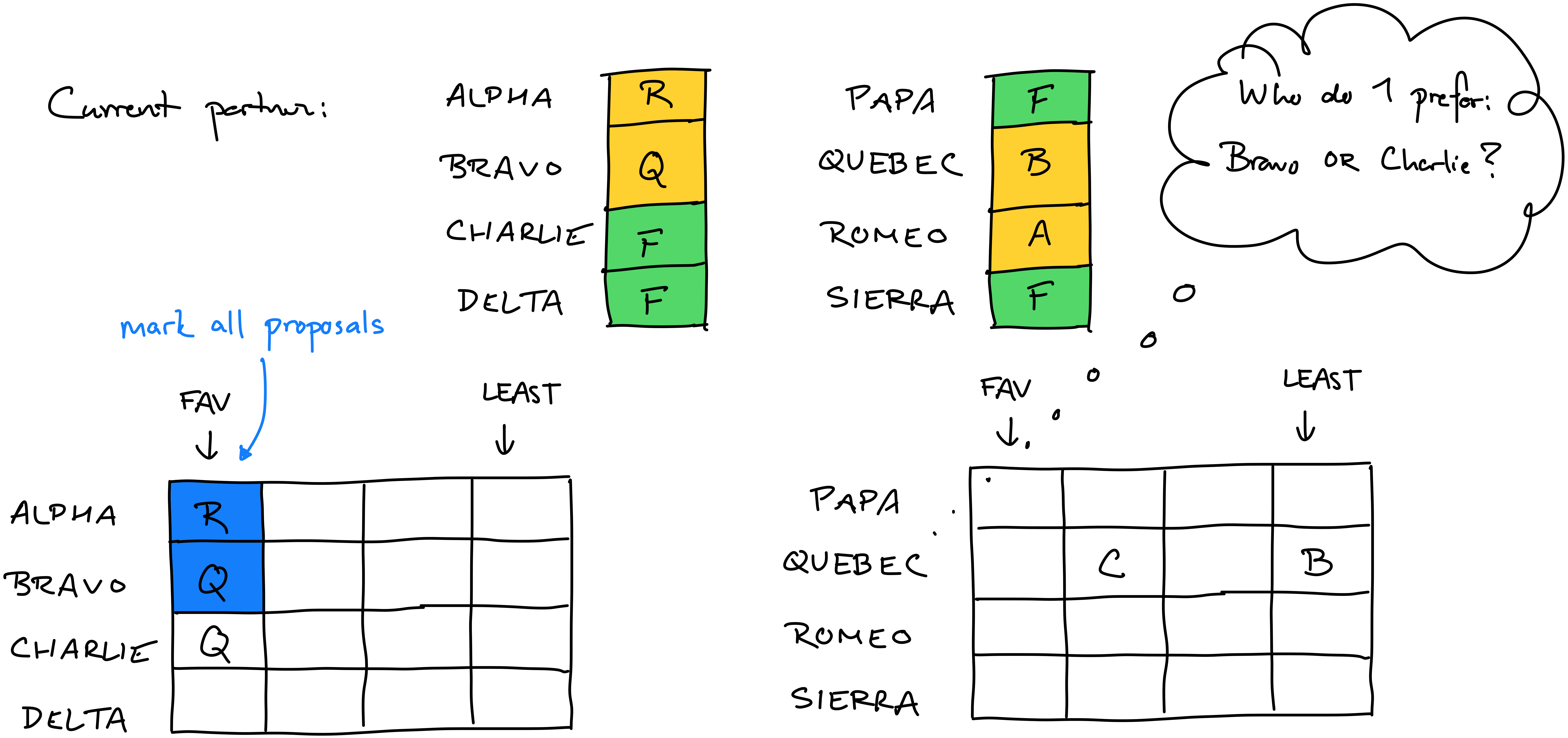
- Naïvely would take $O(n)$ queries to read through $\text{pref}_R[r][\,\cdot\,]$ to find both p and p'

```
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
}
```

Gale-Shapley walkthrough



Gale-Shapley walkthrough



Implementing Gale-Shapley in $O(n^2)$ time

Rejecting & accepting proposals

- How do we decide efficiently if receiver r prefers proposer p to proposer p'?
- Naïvely would take $O(n)$ queries to read through $\text{pref}_R[r][\cdot]$ to find both p and p'
- Instead, *precompute* the inverse list of preferences: $\text{invpref}_R[r][p]$.
- Property: $j = \text{invpref}_R[r][p]$ if and only if $p = \text{pref}_R[r][j]$.
- Takes $O(n^2)$ time to precompute inverse list. Once computed, each comparison takes time $O(1)$.

```
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
}
```

r	1st	2nd	3rd	4th	5th	6th	7th	8th
pref	8	3	7	1	4	5	6	2

r	1	2	3	4	5	6	7	8
inverse	4th	8th	2nd	5th	6th	7th	3rd	1st

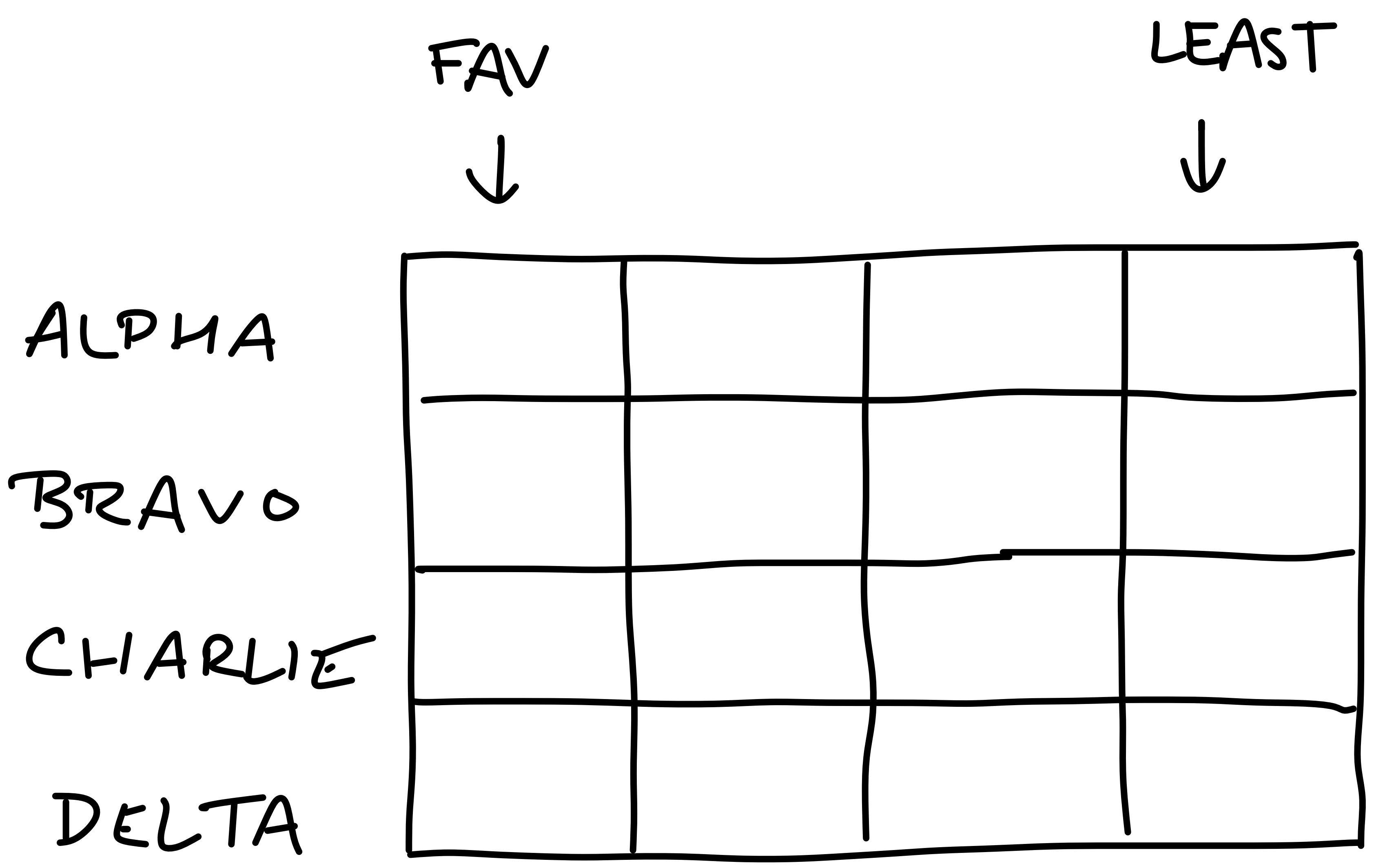
```
for each i
   invpref[r][pref[r][i]] = i
```

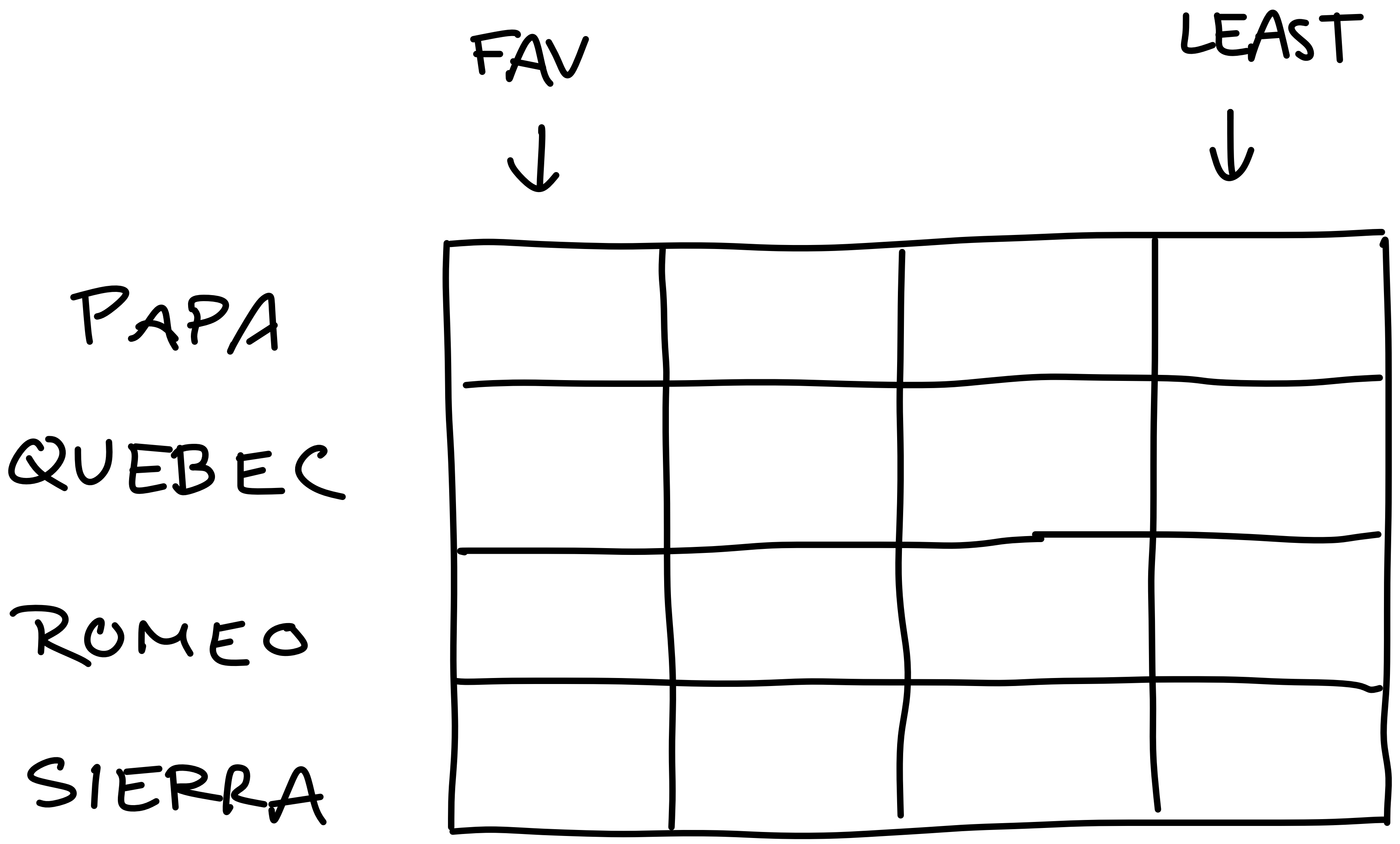
Implementing Gale-Shapley in $O(n^2)$ time

- When a proposer p becomes free, p starts proposing to new receivers starting from $\mathrm{count}[p]$. All previous receivers have been proposed to in previous steps of the algorithm. Update $\mathrm{count}[p]$ as rejections occur.
- Combined with the inverse list pre computation, we achieve that every proposer-receiver pair (p, r) is considered in $O(1)$ computational steps and there are a total n^2 possible pairs.
- This completes the entire time complexity argument of $O(n^2)$.

Does the ordering of the people matter?

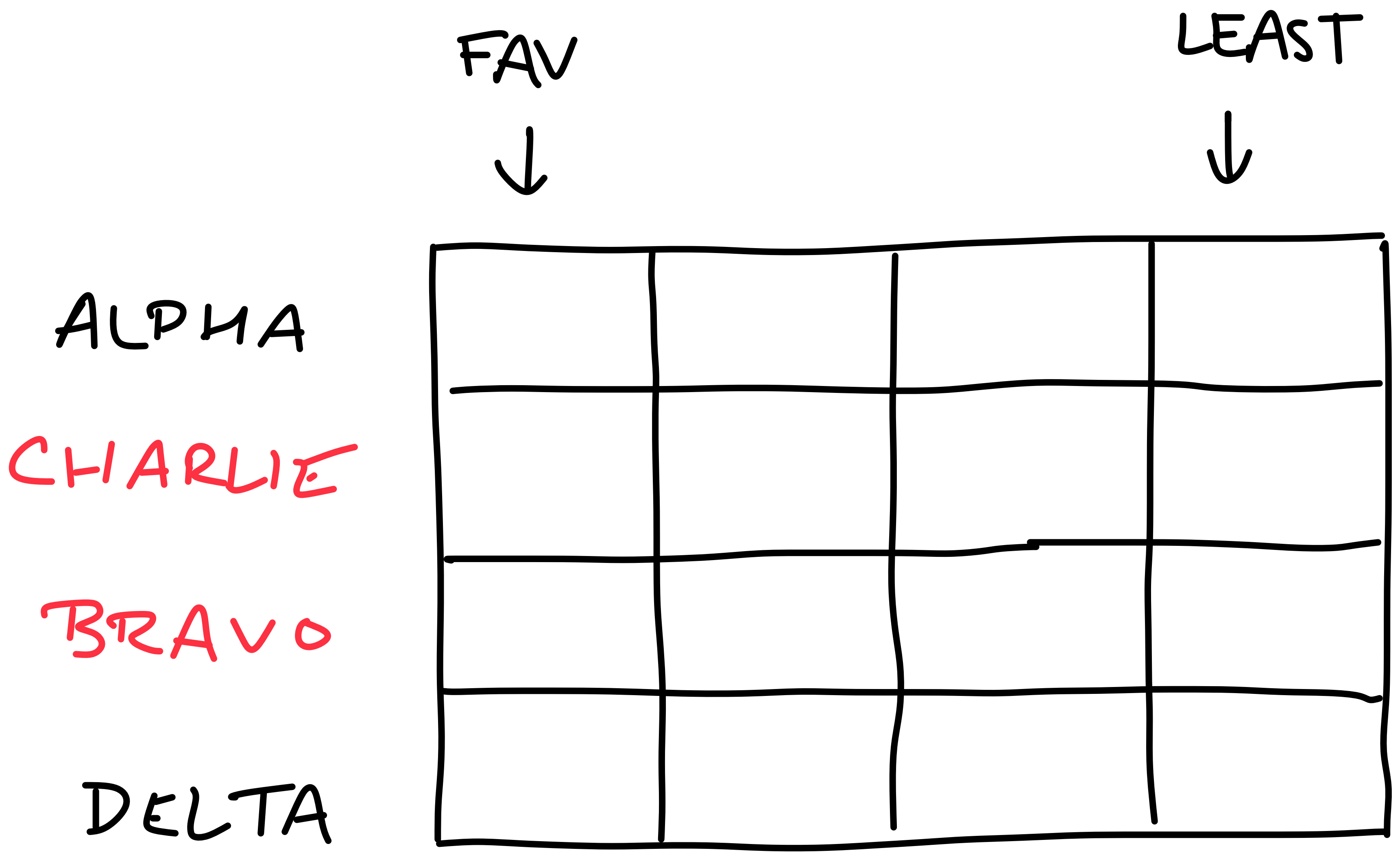
- We arbitrarily assigned the proposers and receivers indexes $1 \ldots n$.
- Would a different assignment have occurred under a different ordering?
- Multiple stable matchings can exist!

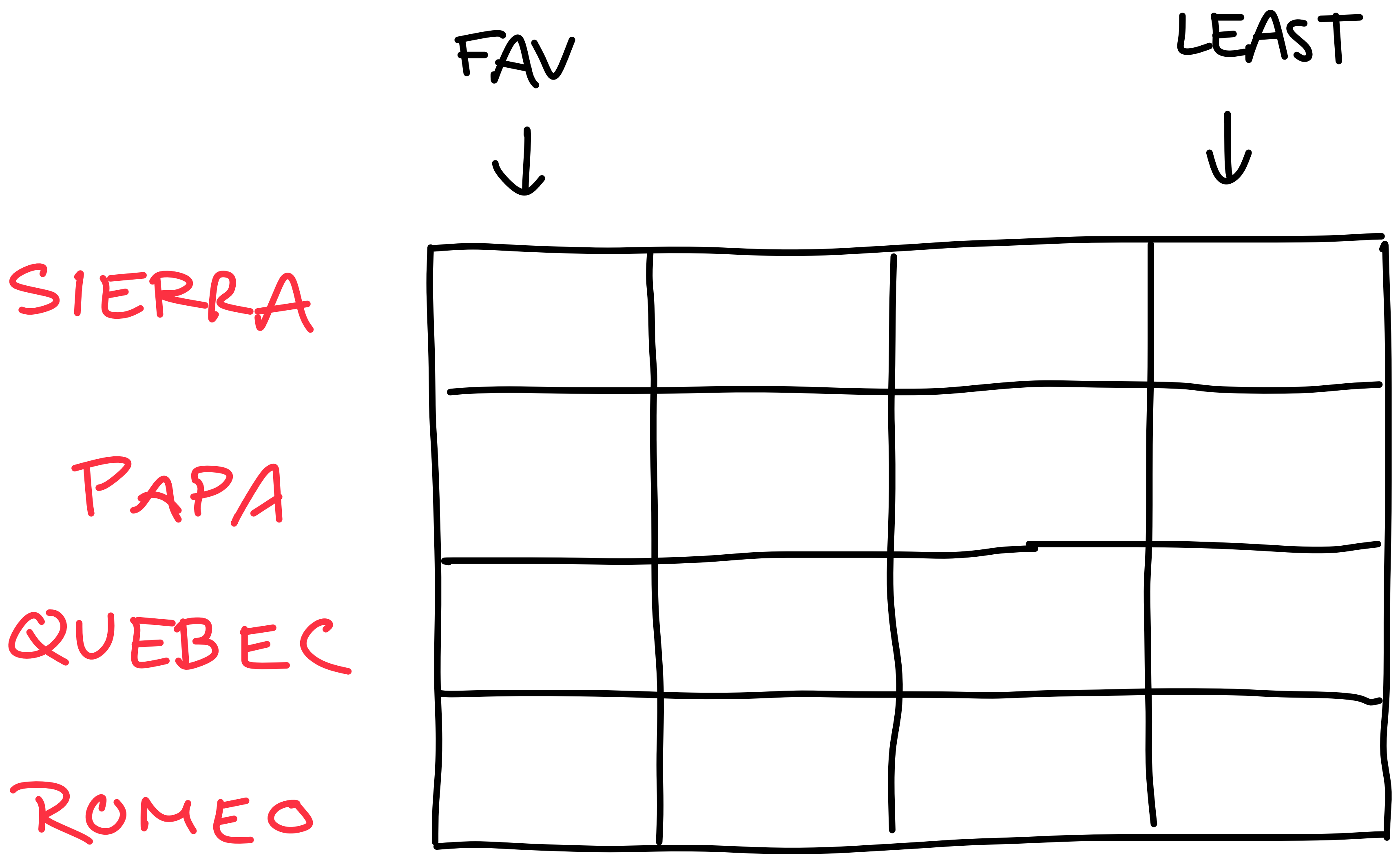




Does the ordering of the people matter?

- We arbitrarily assigned the proposers and receivers indexes $1 \ldots n$.
- Would a different assignment have occurred under a different ordering?
- Multiple stable matchings can exist!





It’s good to be a proposer

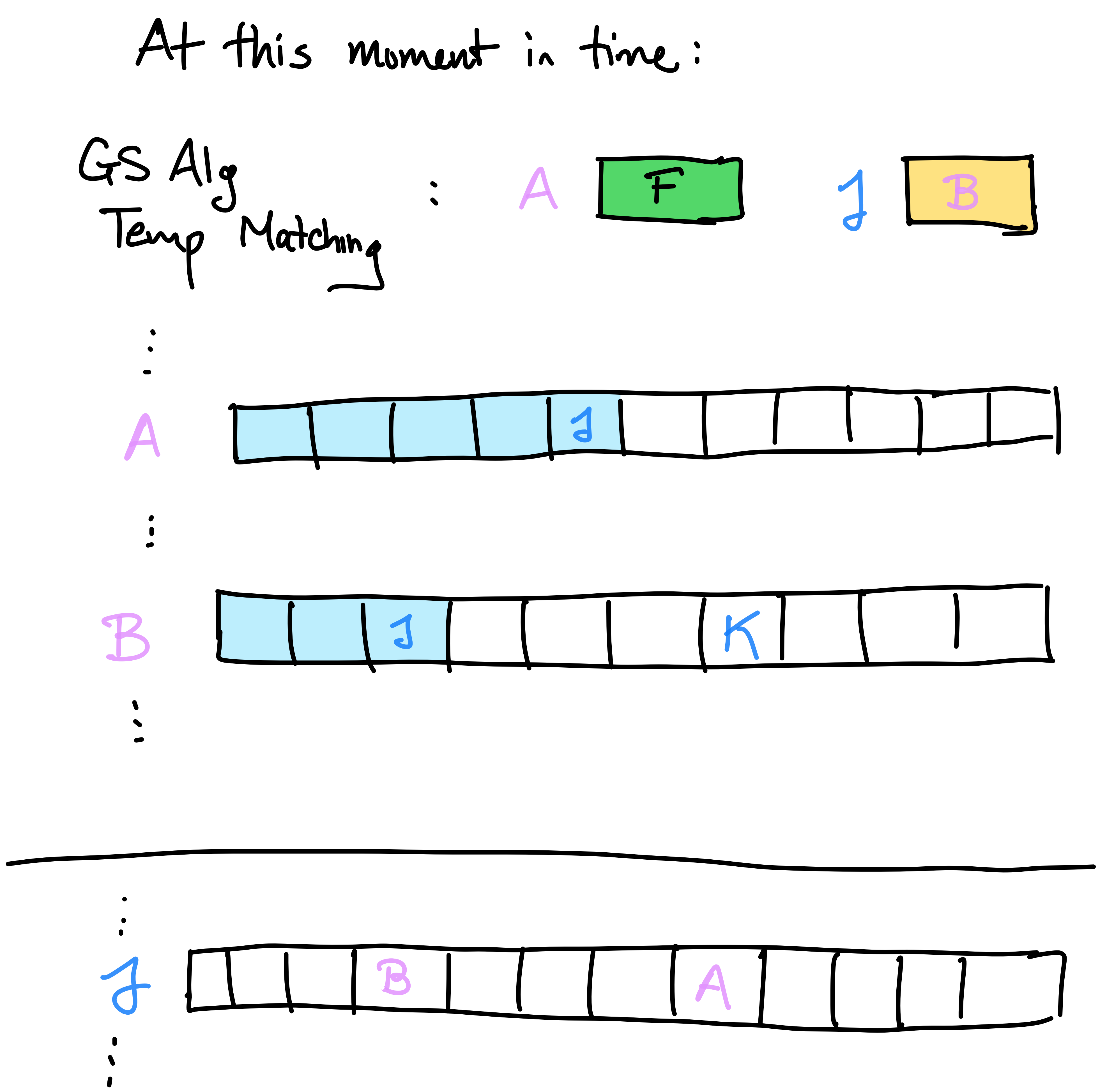
Proposer-optimality of Gale-Shapley

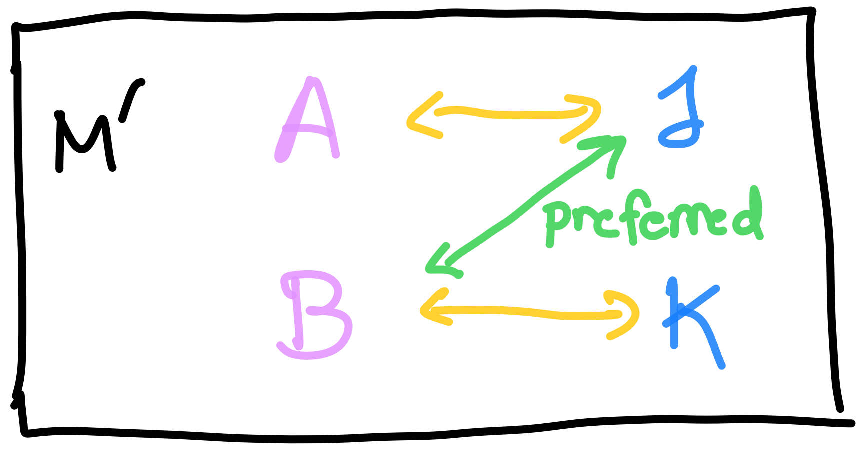
- **Proposer-optimal:** The proposer-optimal assignment is one in which *every* proposer p is matched with their best *valid* partner
- **Valid partnership:** p and r is a valid partnership if there exists some stable matching containing (p, r)
- **Lemma:** Gale-Shapley always produces a proposer-optimal stable matching.
  - **Corollary:** Gale-Shapley always produces the same assignment. I.e. ordering does not matter!
  - **Proof:** There is at most one proposer-optimal stable matching. Since Gale-Shapley always outputs a proposer-optimal stable matching, it always outputs the same assignment irrespective of permutation of players.

Proof of proposer-optimality

there is some stable matching M' containing (Alice, Jake).

- A proof by contradiction. Assume M is not proposer-optimal then there is some ***first time in running GS*** that a proposer Alice is rejected by a valid partner Jake since proposers propose in order of preference.
- Since Jake rejected Alice, let Bob be the partner Jake prefers: either (Bob was engaged to Jake) or (Bob replaced Alice). And in M', let Kevin be the partner of Bob: (Bob, Kevin) is stable.
- Since Jake rejecting Alice is the ***first*** rejection by a valid partner, at that moment in the algorithm, Kevin cannot have rejected Bob. Only possibility, Bob hasn't proposed to Kevin yet.
  - So Bob prefers Jake to Kevin.
  - And, we said that Jake prefers Bob to Alice.
  - So (Bob, Jake) is unstable for M'. A contradiction to its stability of M'.





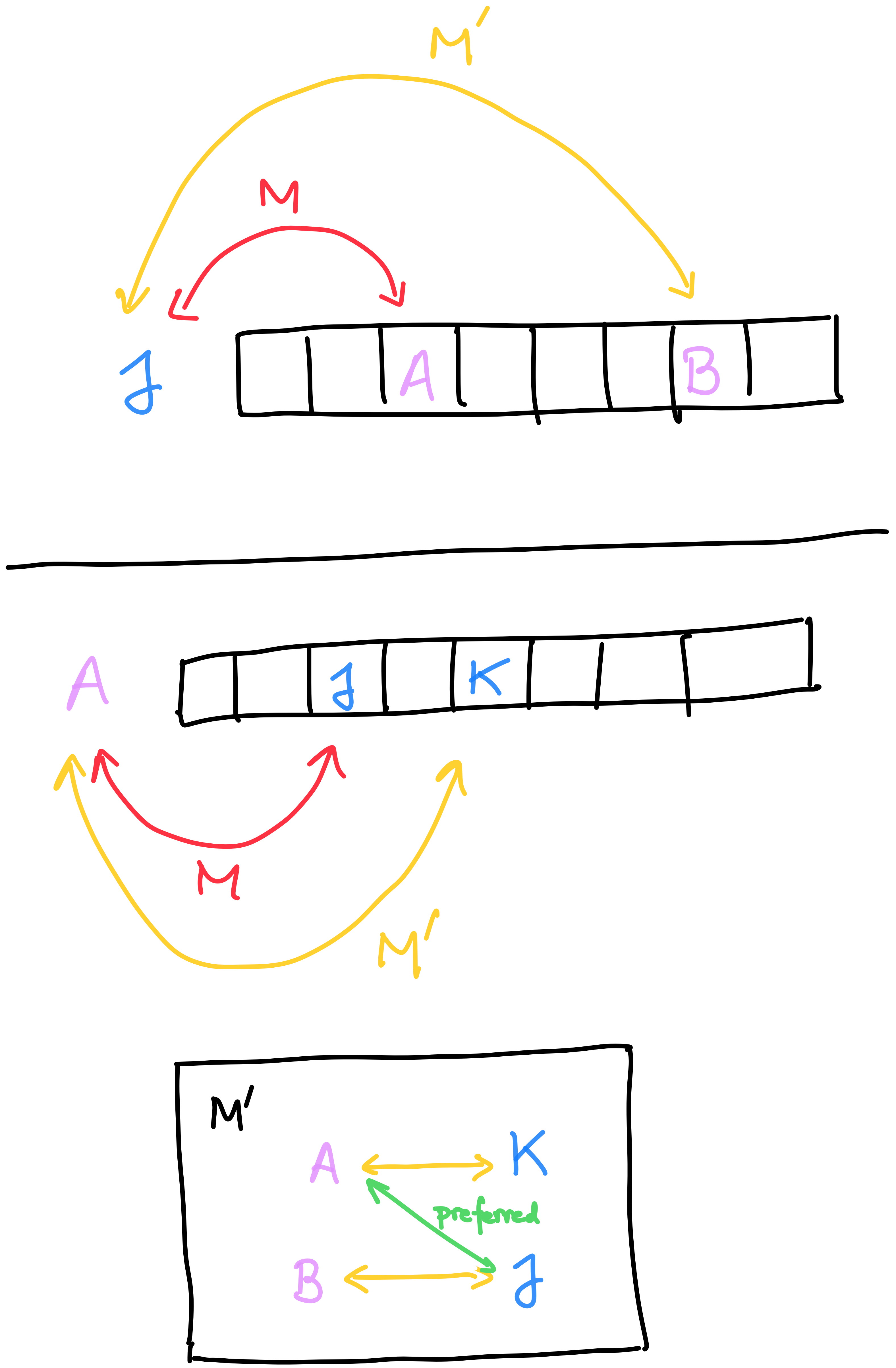
It’s bad to be a receiver

Receiver-pessimality of Gale-Shapley

- **Receiver-pessimal:** The receiver-pessimal assignment is one in which *every* receiver r is matched with their worst *valid* partner
- **Valid partnership:** p and r is a valid partnership if there exists some stable matching containing (p, r)
- **Lemma:** Gale-Shapley always produces a receiver-pessimal stable matching.

Proof of receiver-pessimality

- A proof by contradiction. Assume M is not receiver-pessimal i.e. some receiver Jake is matched to Alice but Alice is not the worst valid partner
  - There exists a M' stable matching in which Jake is matched to Bob but Bob is lower ranked by Jake in M'
  - Let Kevin be the match of Alice in M'
- Proposer-optimality of M gives that Alice prefers Jake to Kevin
  - (Alice, Jake) is unstable for M', a contradiction.



Natural extensions

Example: Matching residents to hospitals

- Original form: proposers are hospitals and receivers are med. school residents
- Variations that make the problem different:
  - Some participants could declare some partners as unacceptable. (Rank = ∞).
  - Unequal number of proposers and receivers.
  - Participants can participate in more than one matching.
  - A different notion of “stability”.
  - Residents may want to perform “couples matching”.
- Many natural variants turn out to be NP-complete! A topic we will discuss in depth later in the course.

Actual implementation

- NRMP (National Resident Matching Program)
  - 23,000+ residents legally bound by the outcome
  - Pre-1995 NRMP had the hospitals as proposers (recall, proposer optimality)
  - Post-1995 has the hospitals as receivers (recall, receiver pessimality)
- Rural hospital dilemma
  - How to get residents to unpopular (often rural hospitals)?
  - Rural hospitals were often undersubscribed in matchings.

Meta-lessons from stable matching

- To design and analyze algorithms, isolate the underlying structure of the problem.
- Algorithms can have deep social ramifications that need to be understood. Algorithm design can have unintended consequences.
- Technique for study algorithms: Find the first time the “bad event” might happen in the running of the algorithm and prove it doesn’t occur.
    - Variant of proof by contradiction.

Are you incentivized to lie?

- Should stable matching players lie about their preferences to get better outcomes?
  - By proposer optimality, a proposer has no incentive to lie.
  - Receivers are incentivized to lie.
- No mechanism can guarantee stable matchings and incentivize honesty. (Not proven in this class).

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

Group P Preference List

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

Group R True Preference List

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

A pretends to prefer Z to X