Stable Matching

Iterative Algorithms

Outline

- Problem setup
  - Modeling: Define the stable matching problem
- Design
  - Try out some ideas
  - Specify an algorithm
- Analysis
  - Correctness
  - Runtime

Outline

- Problem setup
  - Modeling: Define the stable matching problem
- Design
  - Try out some ideas
  - Specify an algorithm
- Analysis
  - Correctness
  - Runtime

Learning objectives:
You are able to
- Describe the stable matching problem
- Model the stable matching problem as a graph problem.
- Analyze the correctness and the runtime of the proposal algorithm.

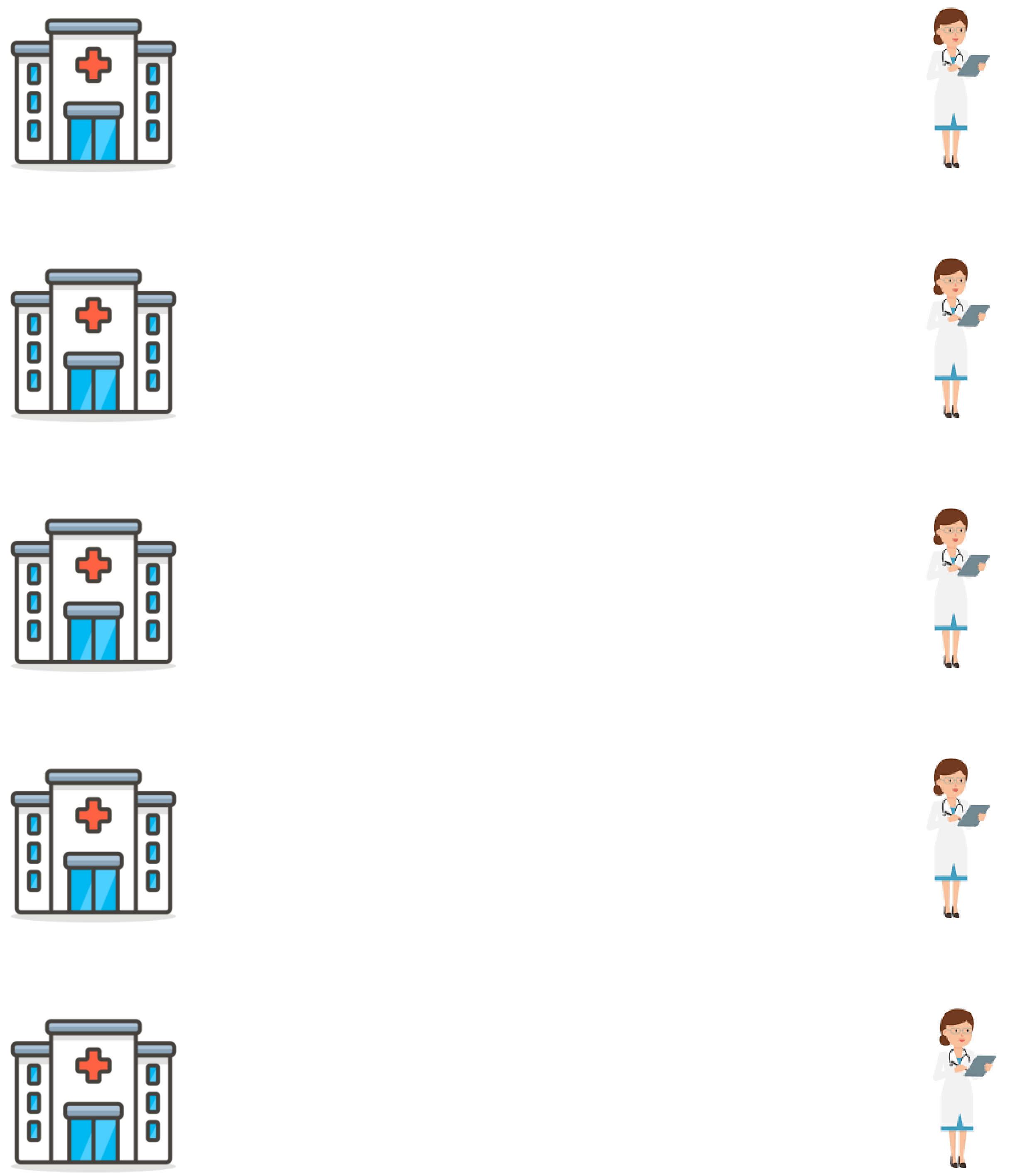
Stable Matching

Hospitals are constantly looking for new doctors.

Stable Matching

Hospitals are constantly looking for new doctors.

Every year, many doctors graduate and are looking for jobs.



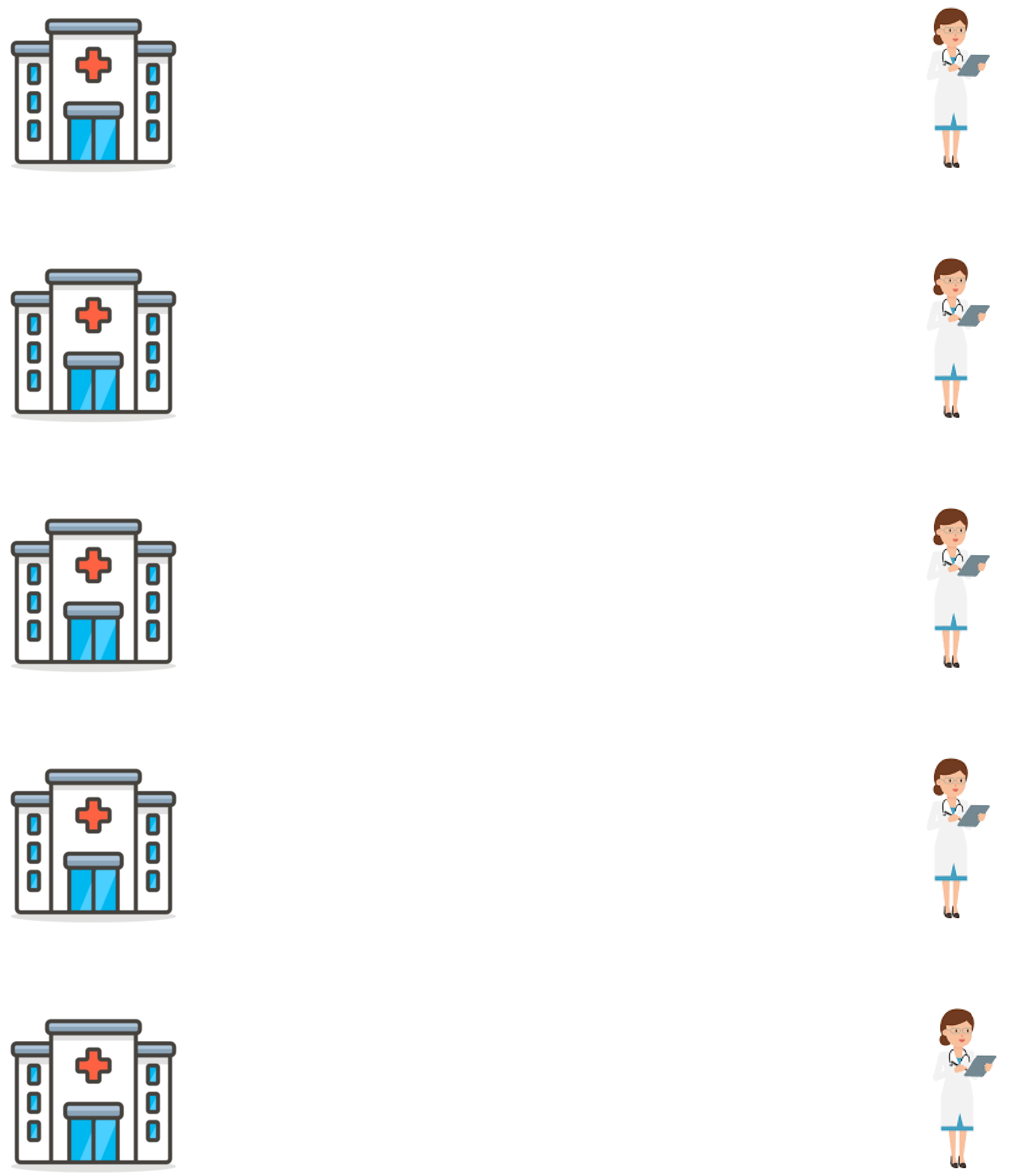
Stable Matching

Each hospital h has a preference ordering on the doctors.

$$h_1\colon\ d_1 < d_3 < d_2 < d_4 < d_5$$

Each doctor d has a preference ordering on the hospitals.

$$d_1\colon\ h_1 < h_3 < h_5 < h_4 < h_2$$



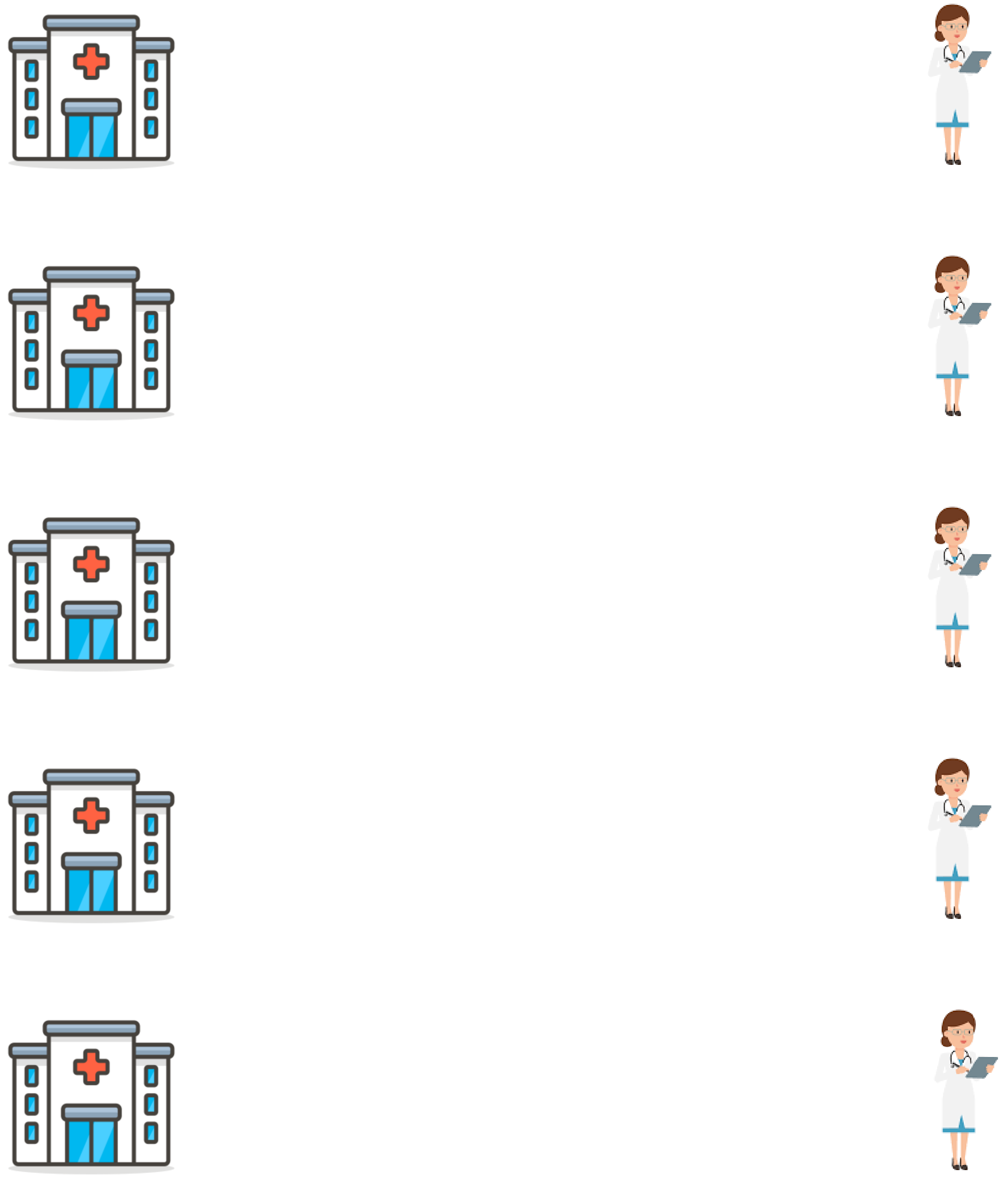
Stable Matching

Each hospital h has a preference ordering on the doctors.

$$h_1\colon\ d_1 < d_3 < d_2 < d_4 < d_5$$

Each doctor d has a preference ordering on the hospitals.

$$d_1\colon\ h_1 < h_3 < h_5 < h_4 < h_2$$



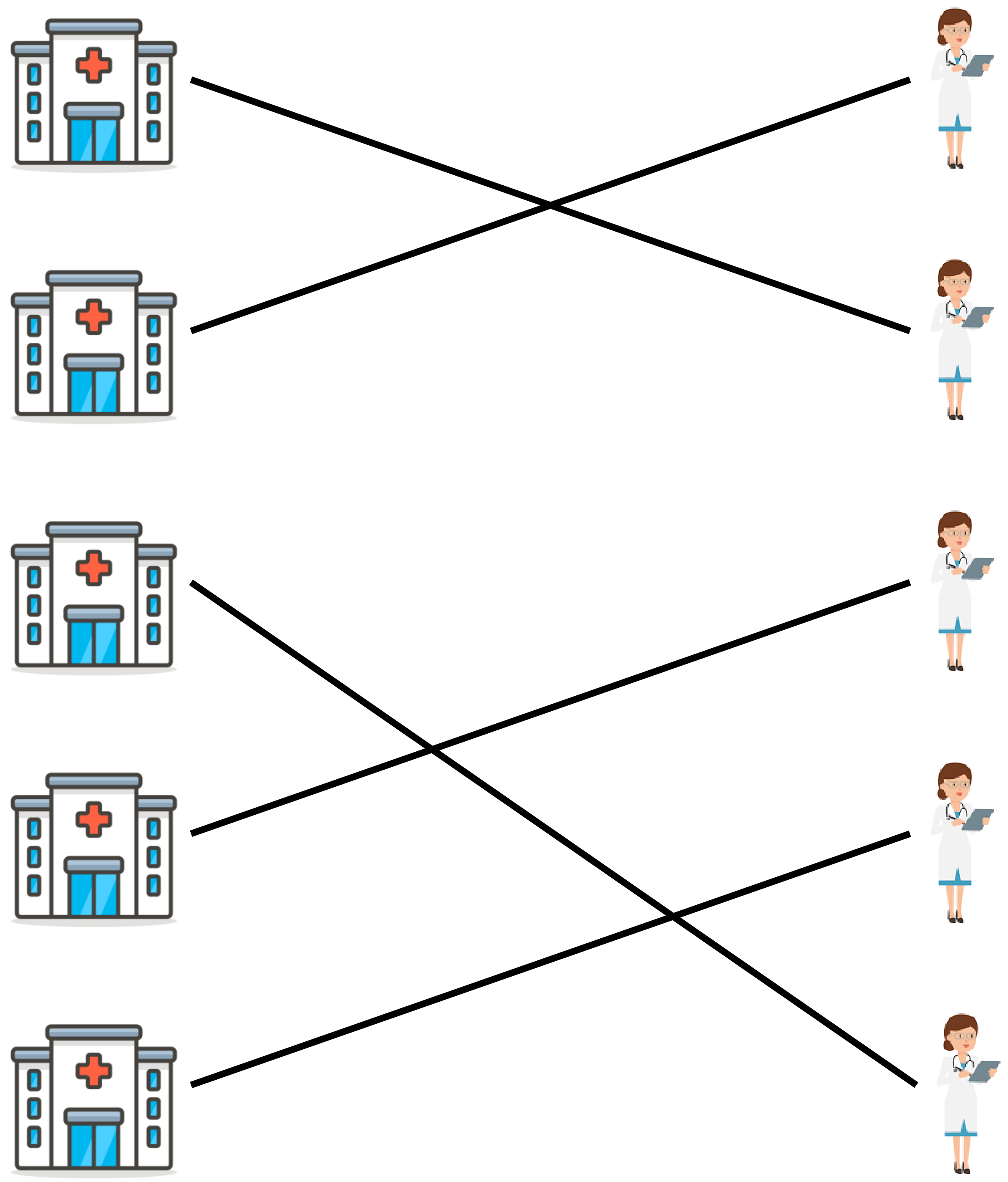
Stable Matching

Each hospital h has a preference ordering on the doctors.

$$h_1 \colon d_1 < d_3 < d_2 < d_4 < d_5$$

Each doctor d has a preference ordering on the hospitals.

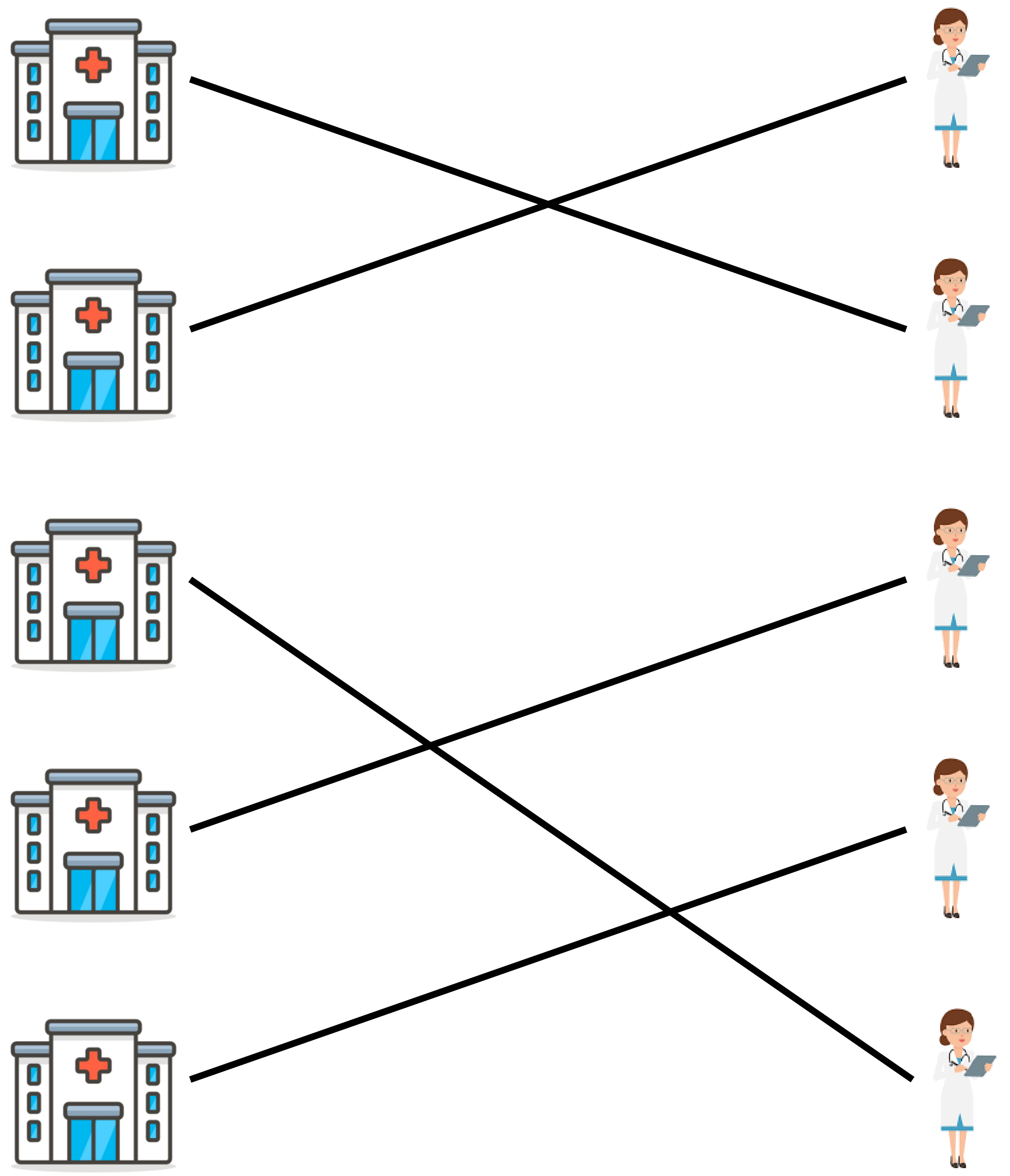
$$d_1 \colon h_1 < h_3 < h_5 < h_4 < h_2$$



Stable Matching

For simplicity:

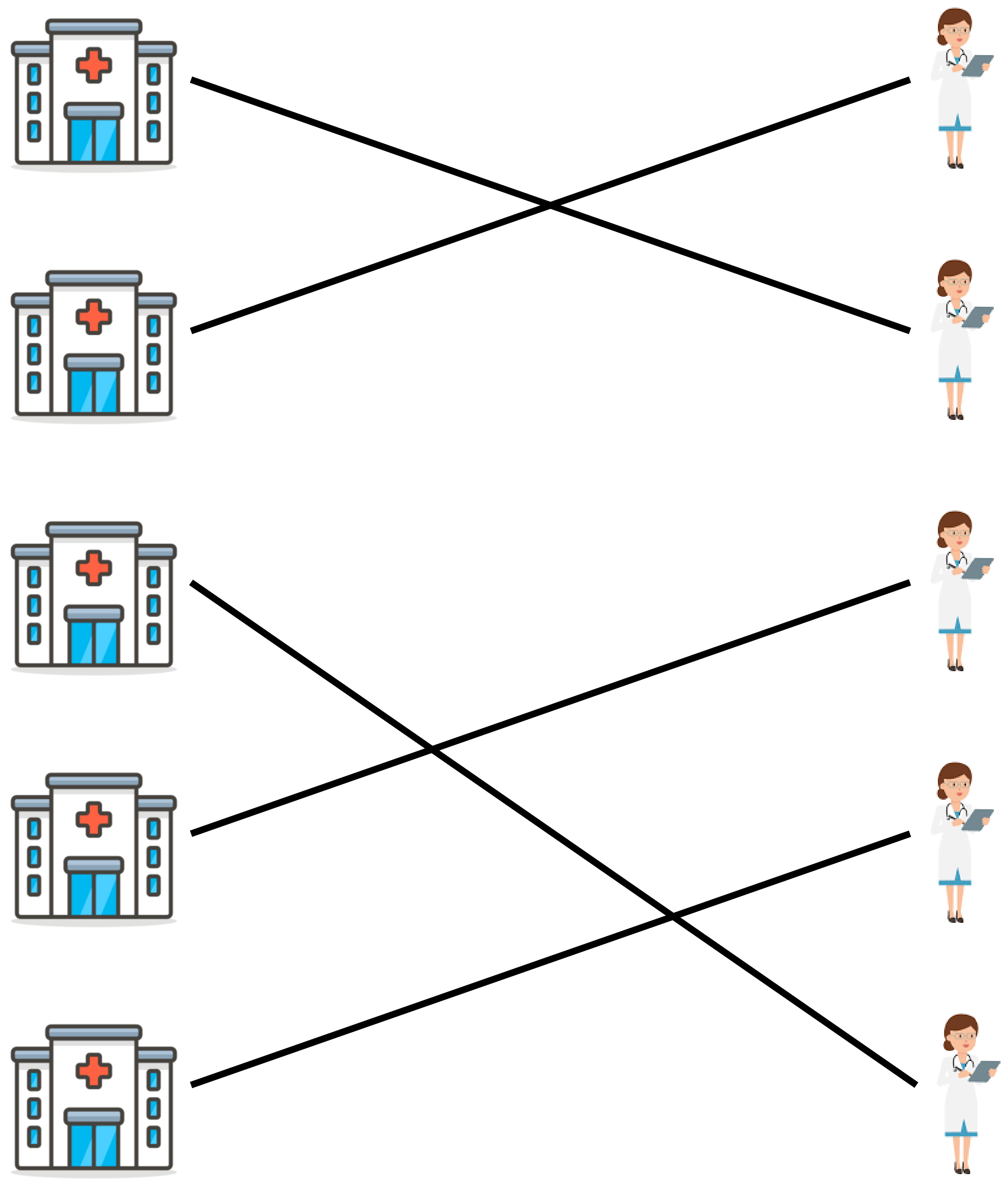
1) One doctor per hospital
2) Strict preferences



Stable Matching

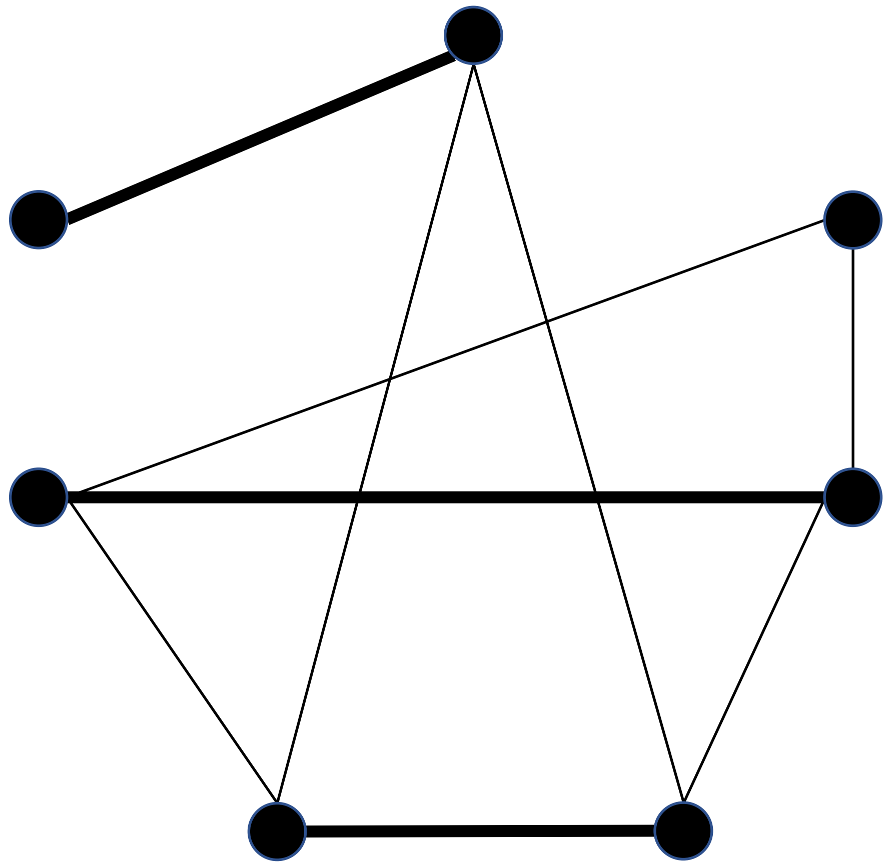
For simplicity:
1) One doctor per hospital
2) Strict preferences

Informally: Assign the doctors to the hospitals in a way that everyone is happy



Stable Matching

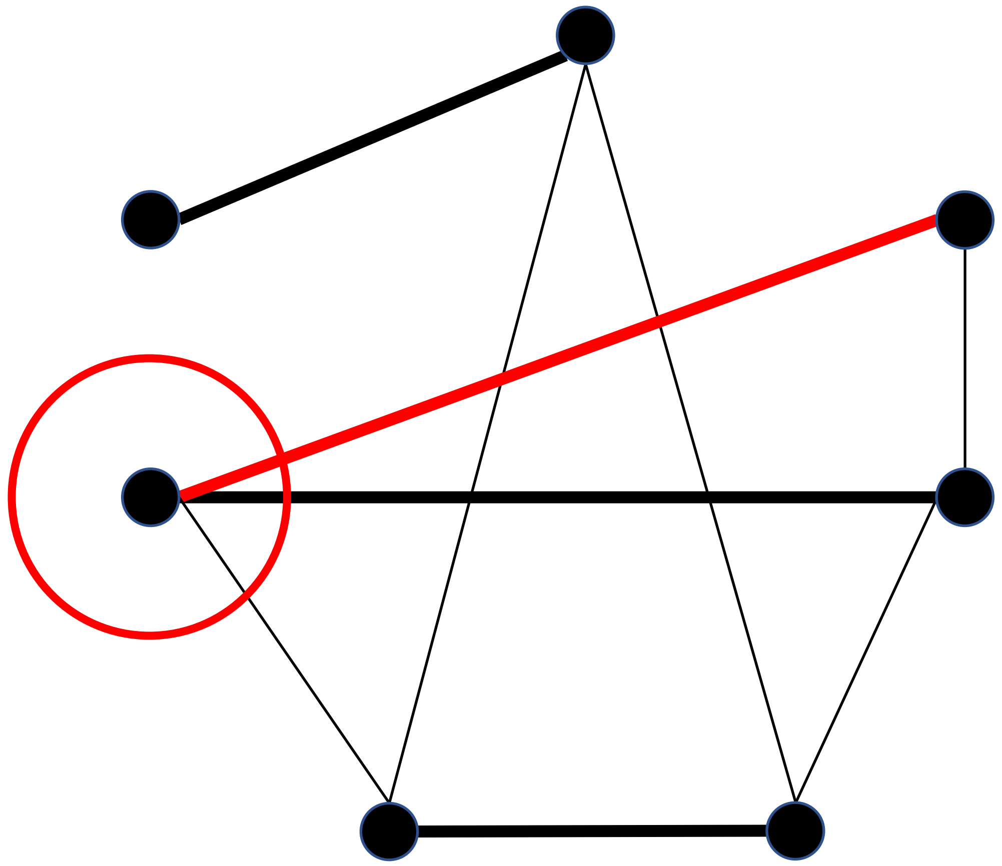
Matching:
A matching of a graph $G = (V, E)$ is a set of edges M such that no two edges in M share an endpoint.



Bold edges correspond to a matching

Stable Matching

Matching:
A matching of a graph $G = (V, E)$ is a set of edges M such that no two edges in M share an endpoint.

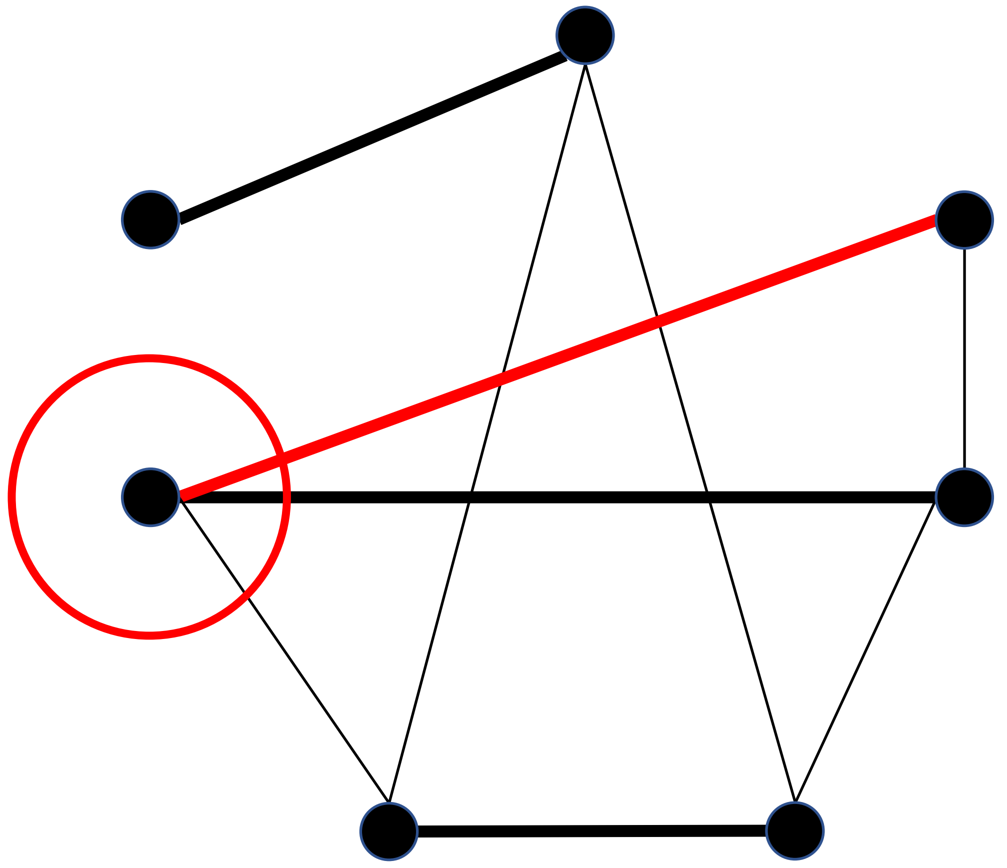


Matching property violated

Stable Matching

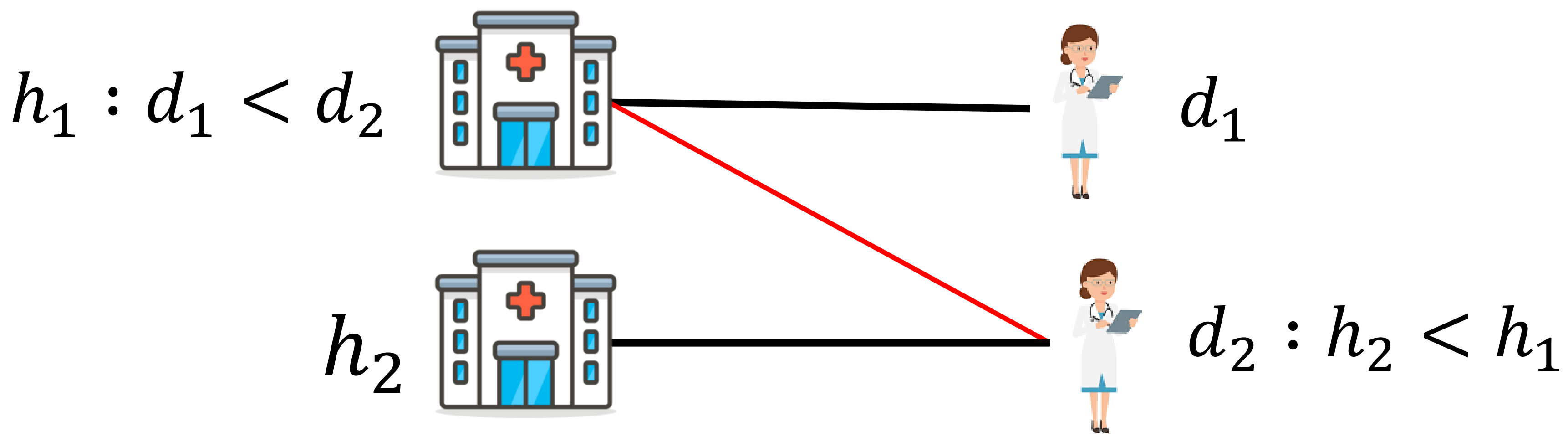
Matching:
A matching of a graph $G = (V, E)$ is a set of edges M such that no two edges in M share an endpoint.

An independent set of edges!

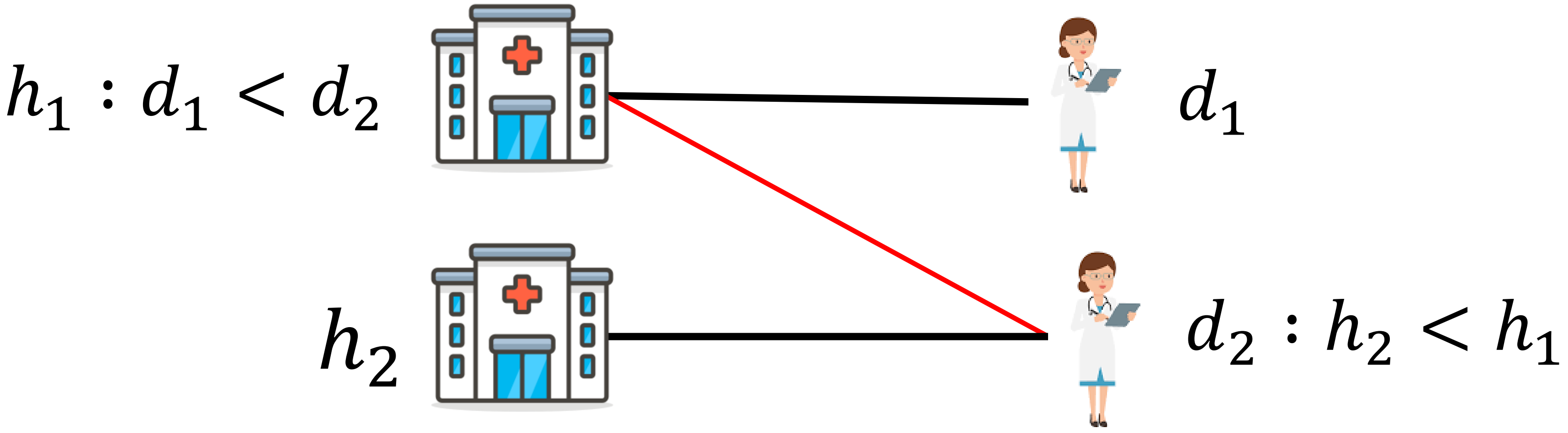


Matching property violated

Stable Matching



Stable Matching



Not matching over the red edge is bad, right?

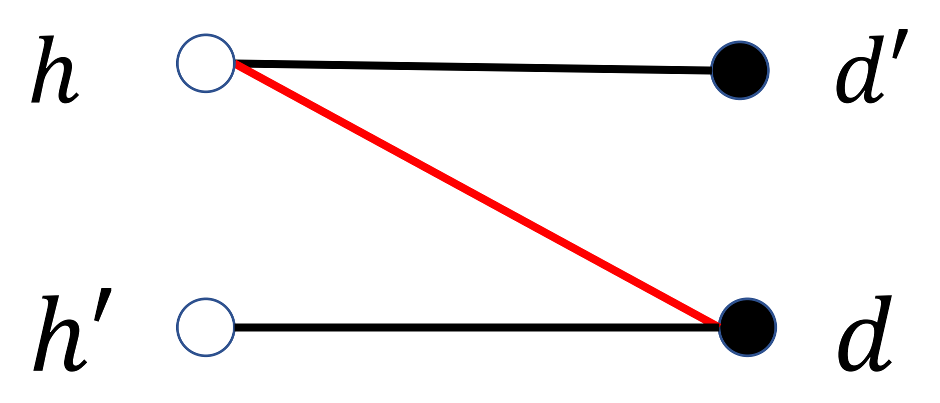
Stable Matching

A Stable Matching:
Consider a matching M of a bipartite graph $G = (B \cup R, E)$ with a bipartition to parts B and R.
Every node u is given a preference vector $<_u$ over its neighbors.

An edge $\{h, d\}$ is *unstable* (wrt M) if there exist nodes h' and d' such that

1. $\{h, d'\} \in M$ and $d' <_h d$
2. $\{h', d\} \in M$ and $h' <_d h$

A matching is *stable* if there are no unstable edges.



If $d' <_h d$ and $h' <_d h$
then the bold red edge is unstable

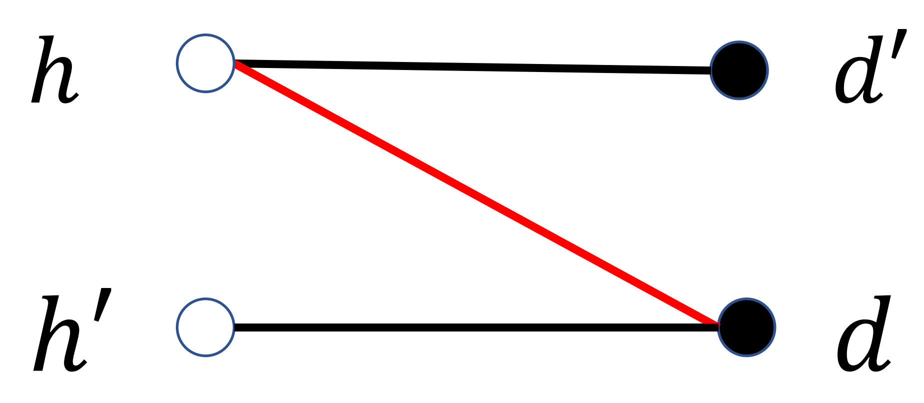
Stable Matching

A Stable Matching:
Consider a matching M of a bipartite graph $G = (B \cup R, E)$ with a bipartition to parts B and R.
Every node u is given a preference vector $<_u$ over its neighbors.

An edge $\{h, d\}$ is *unstable* (wrt M) if there exist nodes h' and d' such that

1. $\{h, d'\} \in M$ and $d' <_h d$
2. $\{h', d\} \in M$ and $h' <_d h$

A matching is *stable* if there are no unstable edges.



If $d' <_h d$ and $h' <_d h$
then the bold red edge is unstable

We assume that node u prefers any neighbor to being alone.

Stable Matching

Input:
A complete bipartite graph $G = (B \cup R, E)$ with a bipartition to parts B and R.

A preference vector $<_h$ over the entries of R for each node $\mathrm{h} \in B$

A preference vector $<_d$ over the entries of R for each node $\mathrm{d} \in R$

Output:
A stable matching

Outline

- Problem setup
  - Modeling: Define the stable matching problem
- Design
  - Try out some ideas
  - Specify an algorithm
- Analysis
  - Correctness
  - Runtime

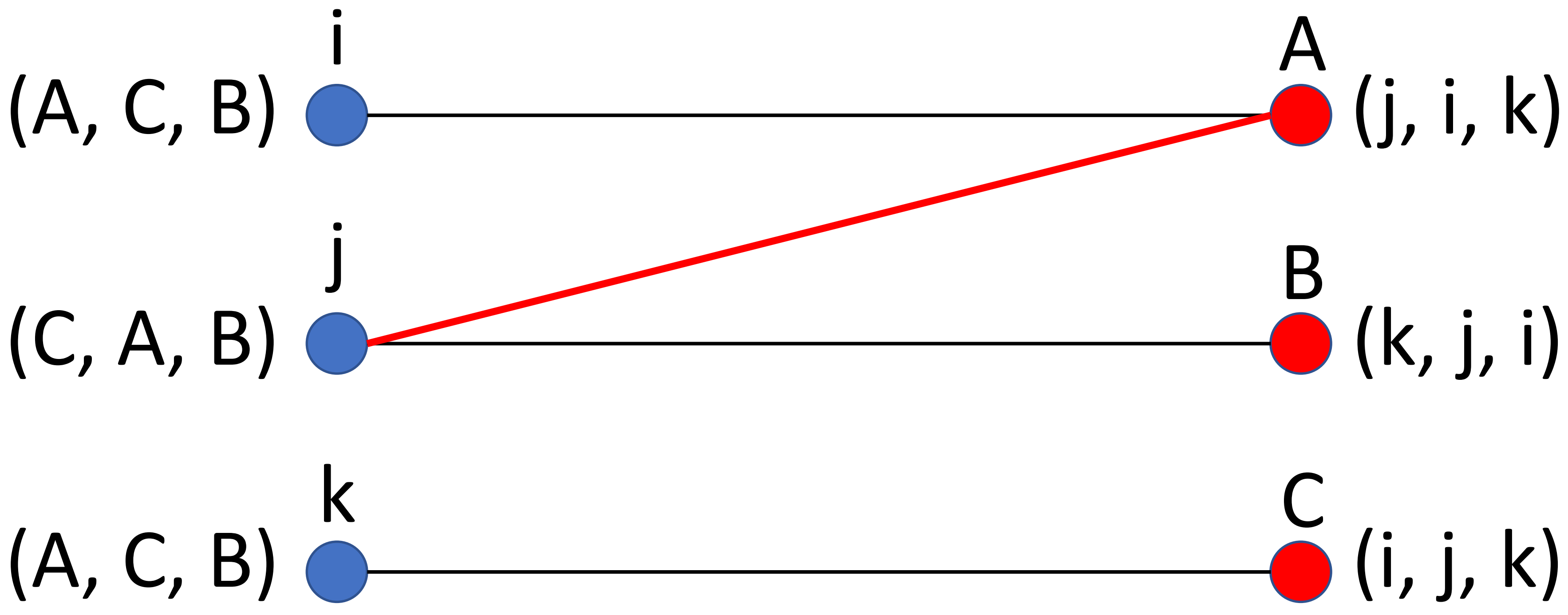
Stable Matching

Start with something very simple. Why does it fail?

Problem: Find a stable matching in a bipartite complete graph

Stable Matching

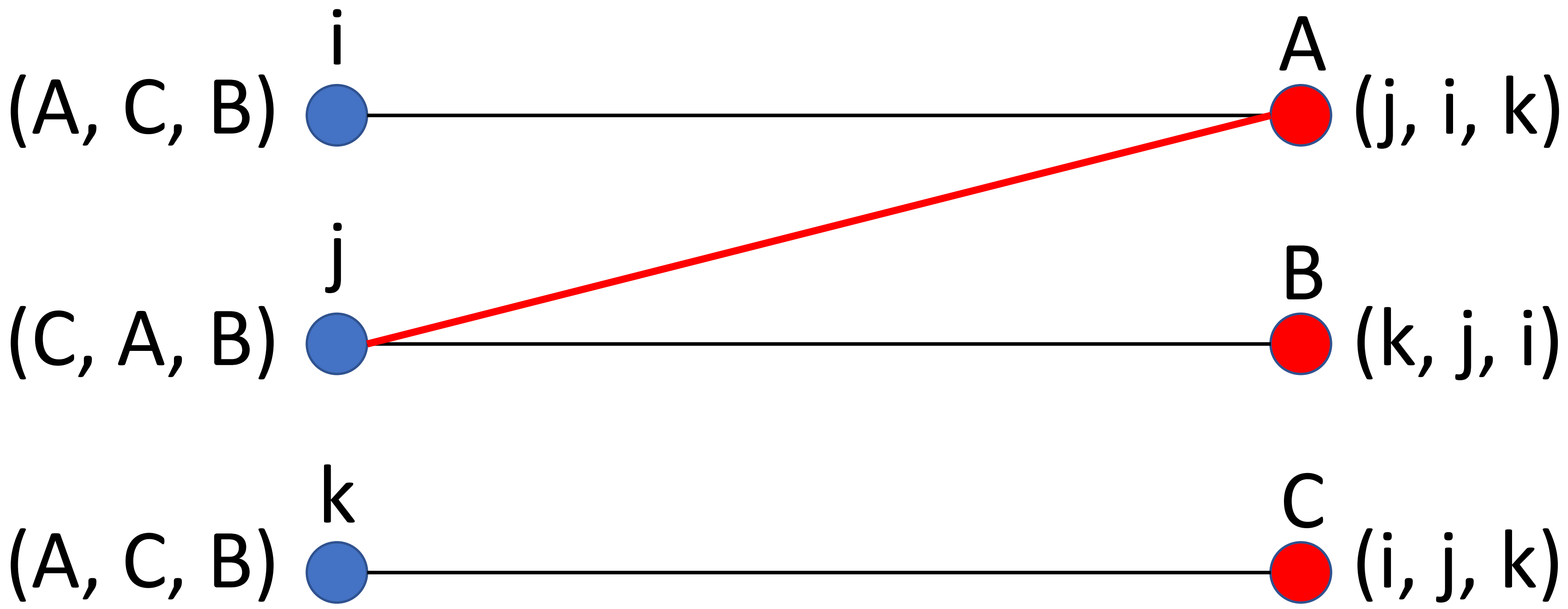
Start with something very simple. Why does it fail?



Not stable:
j prefers A over B and
A prefers j over i

Stable Matching

Start with something very simple. Why does it fail?



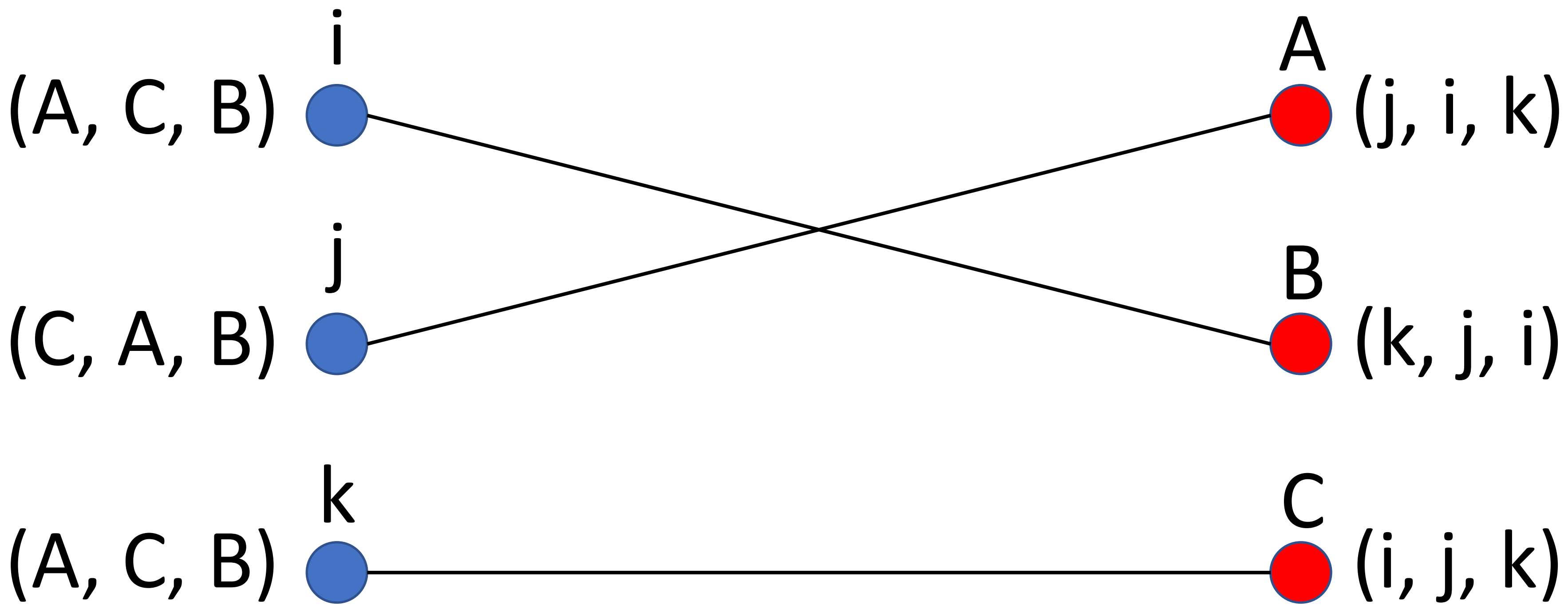
Solution (?):
Change the matching!

Not stable:
j prefers A over B and
A prefers j over i

Stable Matching

Start with something very simple. Why does it fail?

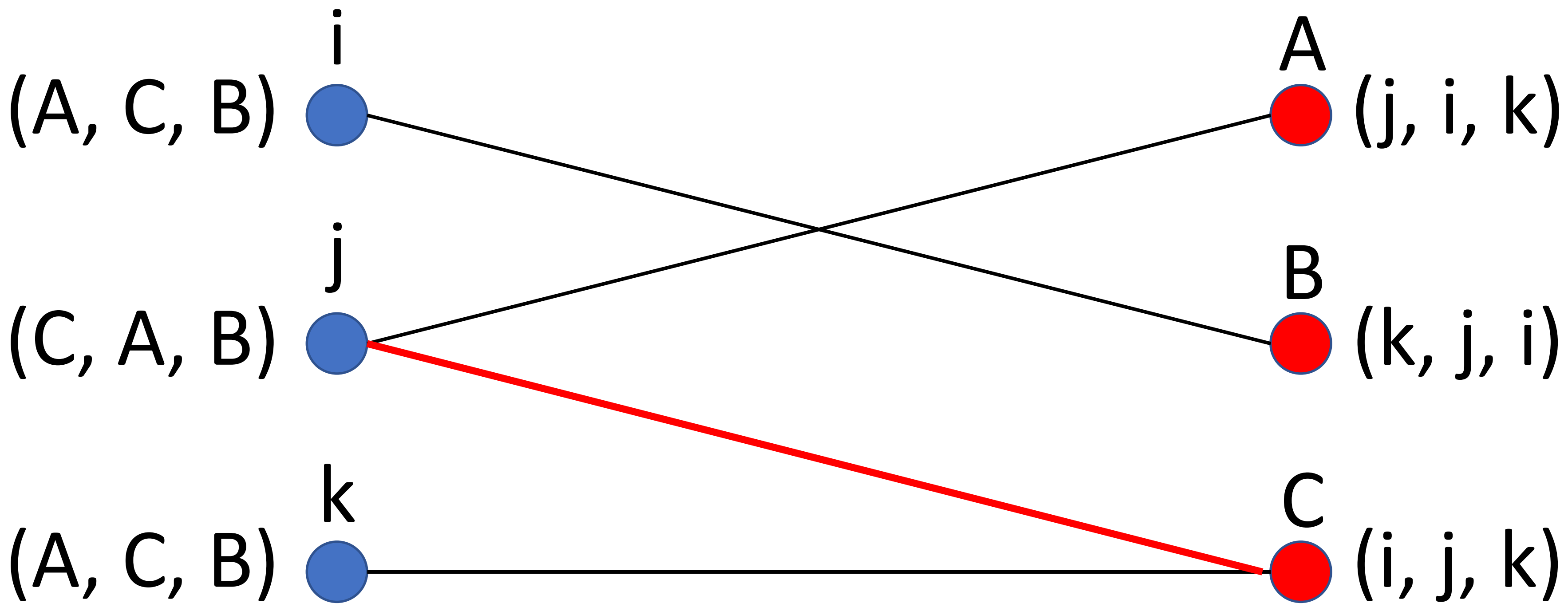
Solution (?):
Change the matching!



Not stable:
j prefers C over A and
C prefers j over k

Stable Matching

Start with something very simple. Why does it fail?

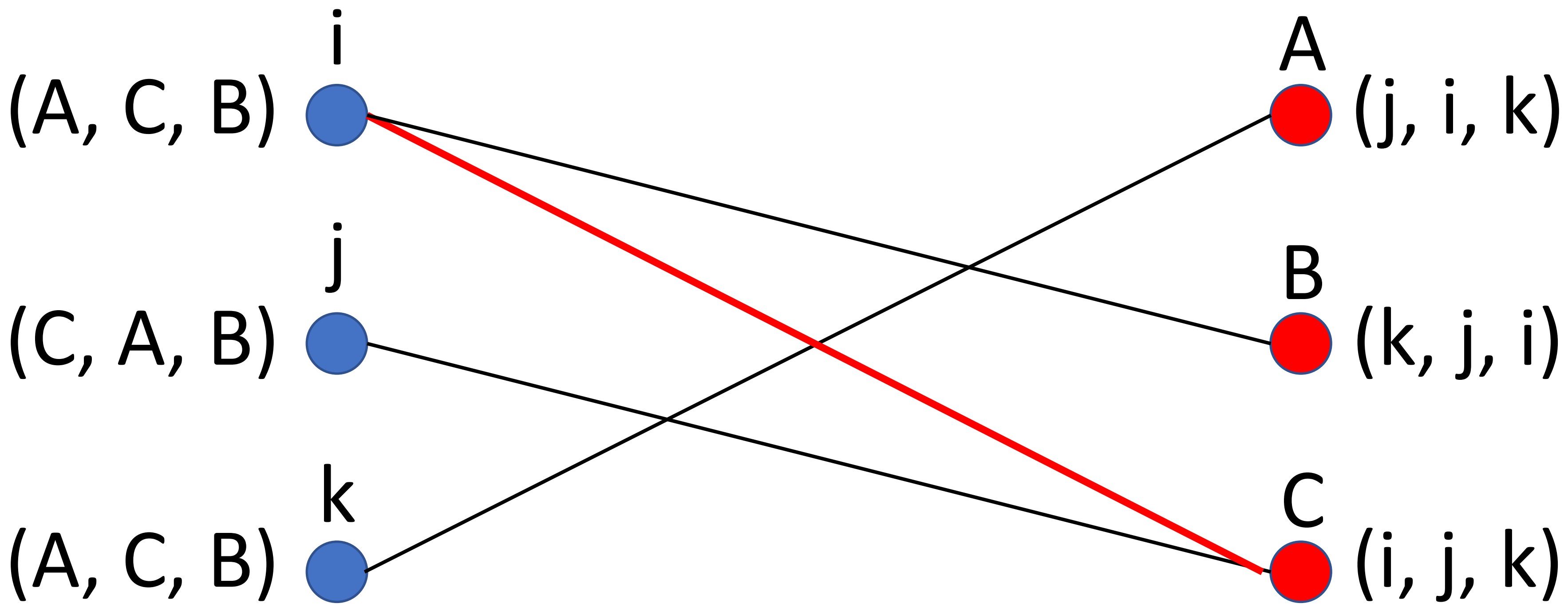


Solution (?):
Change the matching!

Not stable:
j prefers C over A and
C prefers j over k

Stable Matching

Start with something very simple. Why does it fail?

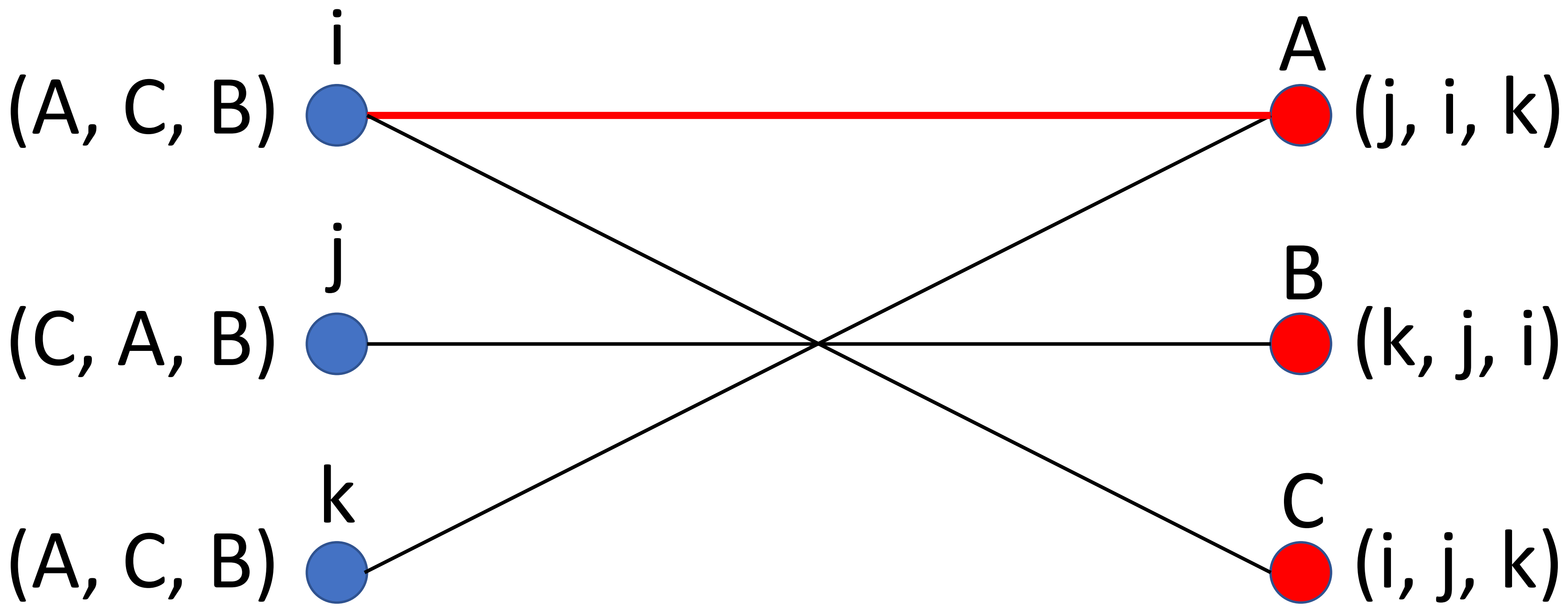


Solution (?):
Change the matching!

Not stable:
i prefers C over B and
C prefers i over j

Stable Matching

Start with something very simple. Why does it fail?

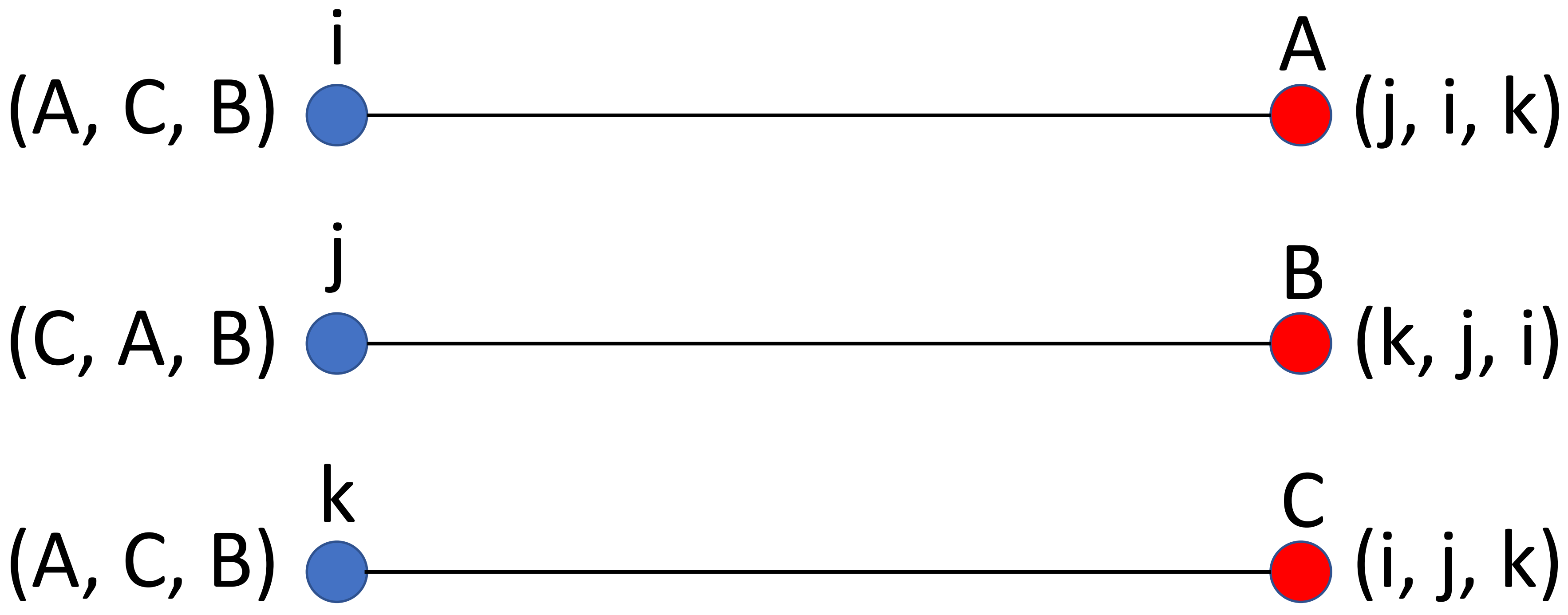


Solution (?):
Change the matching!

Not stable:
i prefers A over C and
A prefers i over k

Stable Matching

Start with something very simple. Why does it fail?



We are back to the original matching

Stable Matching

Start with something very simple. Why does it fail?

Problem: Find a stable matching in a bipartite complete graph

Sanity check: Does a solution always exist?

Stable Matching

Start with something very simple. Why does it fail?

Problem: Find a stable matching in a bipartite complete graph

Sanity check: Does a solution always exist?

Also not clear!

Stable Matching – The Proposal Algorithm

Gale-Shapley Algorithm:
Nobel prize in economics 2012

Stable Matching – The Proposal Algorithm

Does a solution always exist?

Design an algorithm that always finds a correct solution

A correct solution MUST exist. Otherwise, the algorithm must fail.

Outline

- Problem setup
  - Modeling: Define the stable matching problem
- Design
  - Try out some ideas
  - Specify an algorithm
- Analysis
  - Correctness
  - Runtime

Stable Matching – The Proposal Algorithm

Preference list $<_x$ for each blue node x
Preference list $<_y$ for each red node y
For(rounds $r = 1,2, \ldots$**)**

1. Each unmatched blue node u proposes to the most preferred (remaining) red node v.
2. If the proposal is better (i.e., …), node v accepts the proposal. If there is an edge $\{w, v\}$ currently in the matching, that is removed. Edge $\{u, v\}$ is added to the matching.

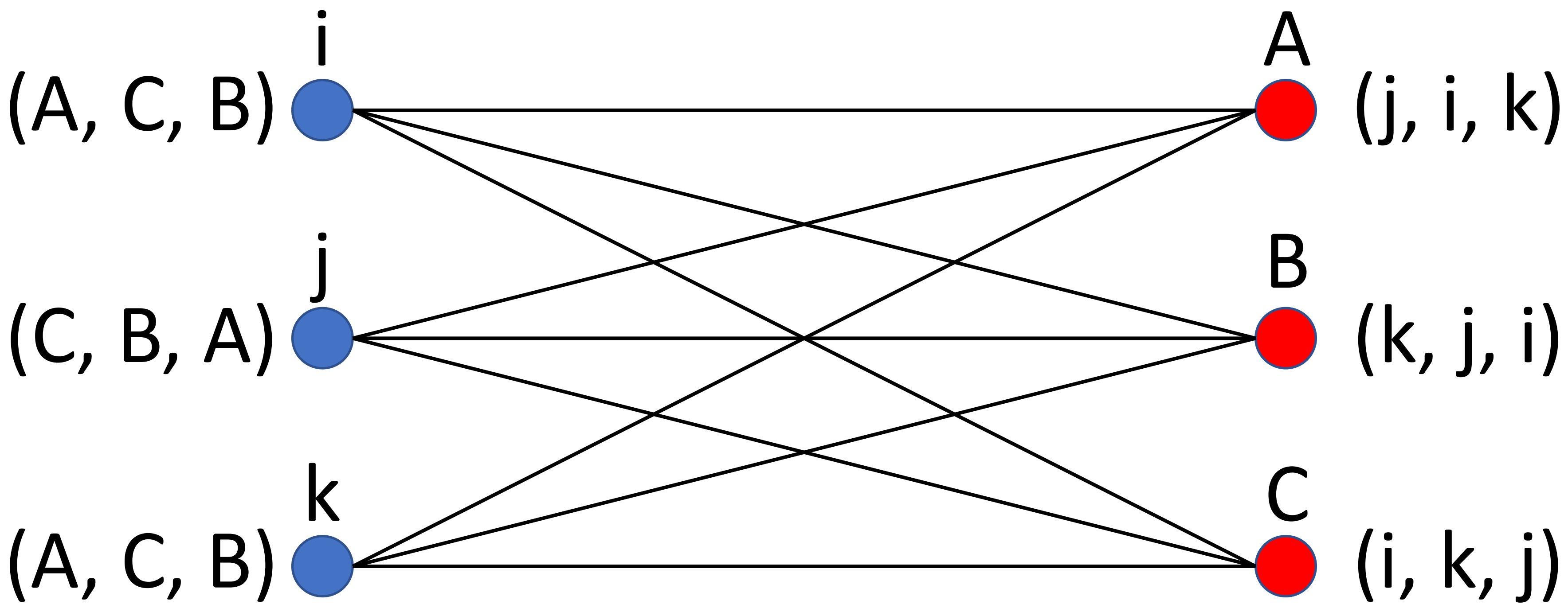
Stable Matching – The Proposal Algorithm

Preference list $<_x$ for each blue node x
Preference list $<_y$ for each red node y
For(rounds $r = 1,2, \ldots$**)**

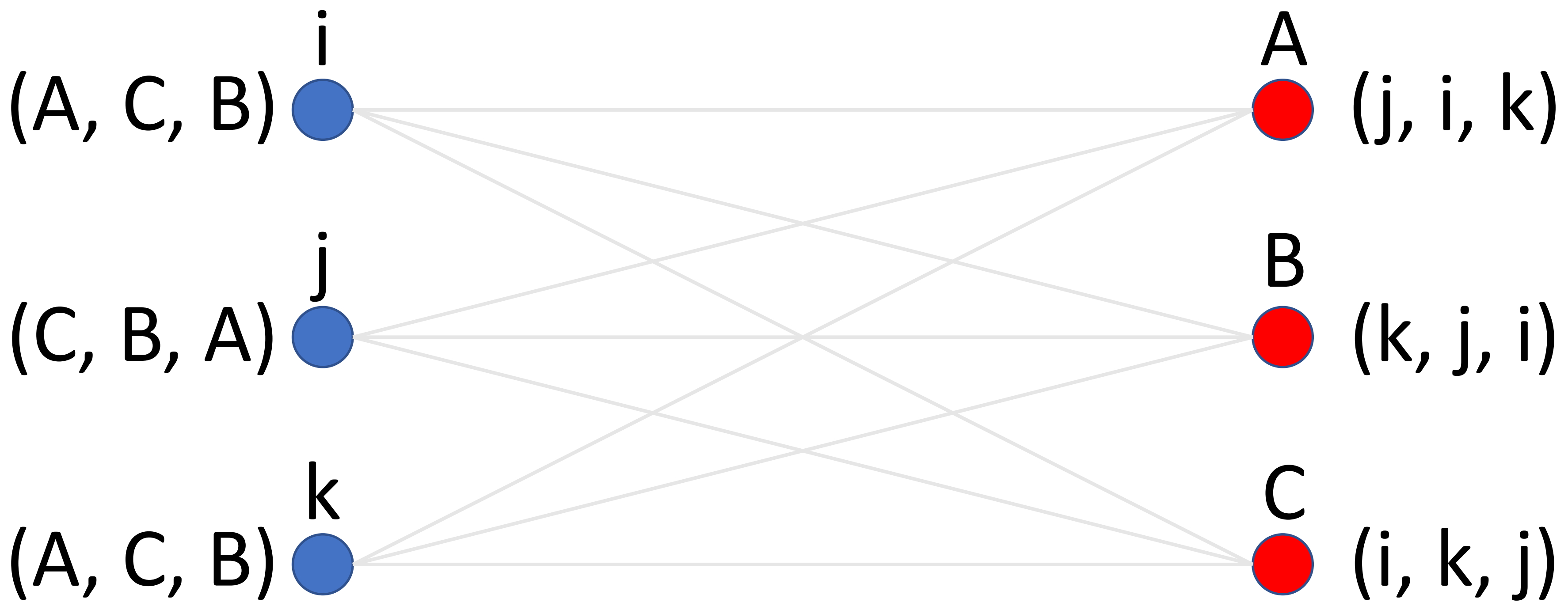
1. Each unmatched blue node u proposes to the most preferred (remaining) red node v.
2. If the proposal is better (i.e., …), node v accepts the proposal. If there is an edge $\{w, v\}$ currently in the matching, that is removed. Edge $\{u, v\}$ is added to the matching.

Iterative!

The Proposal Algorithm

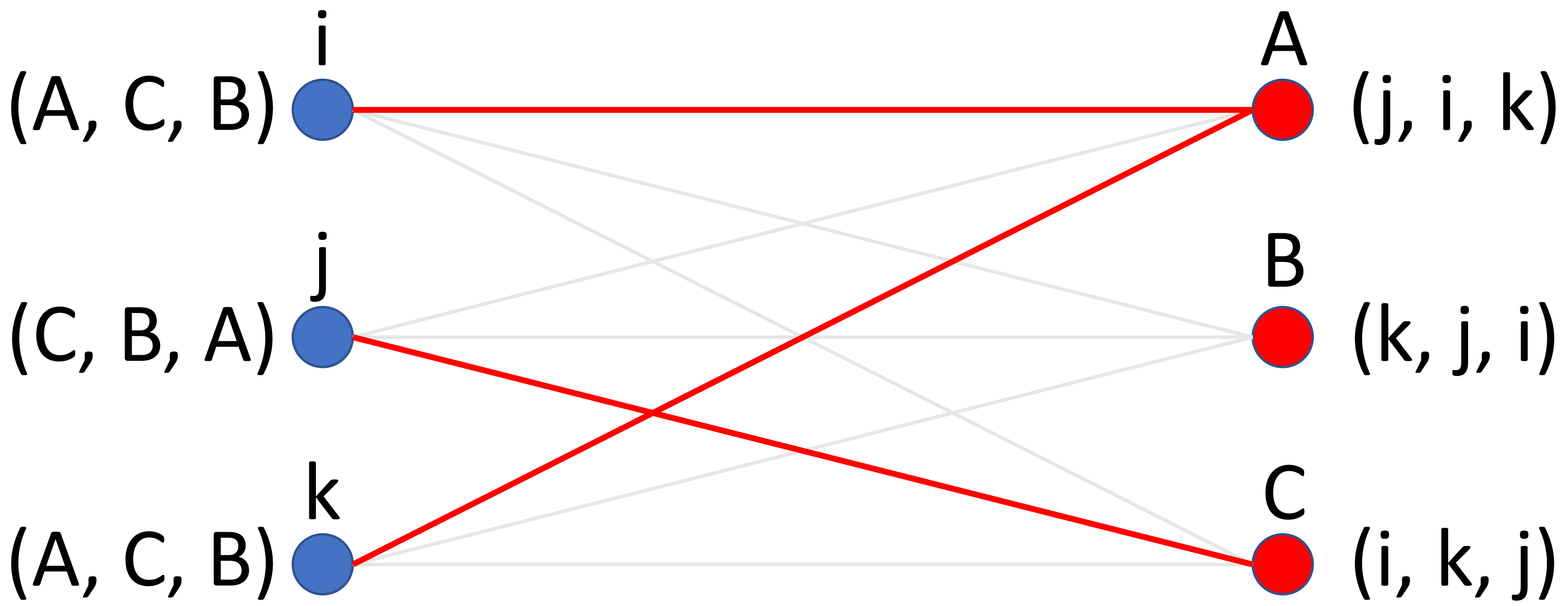


The Proposal Algorithm

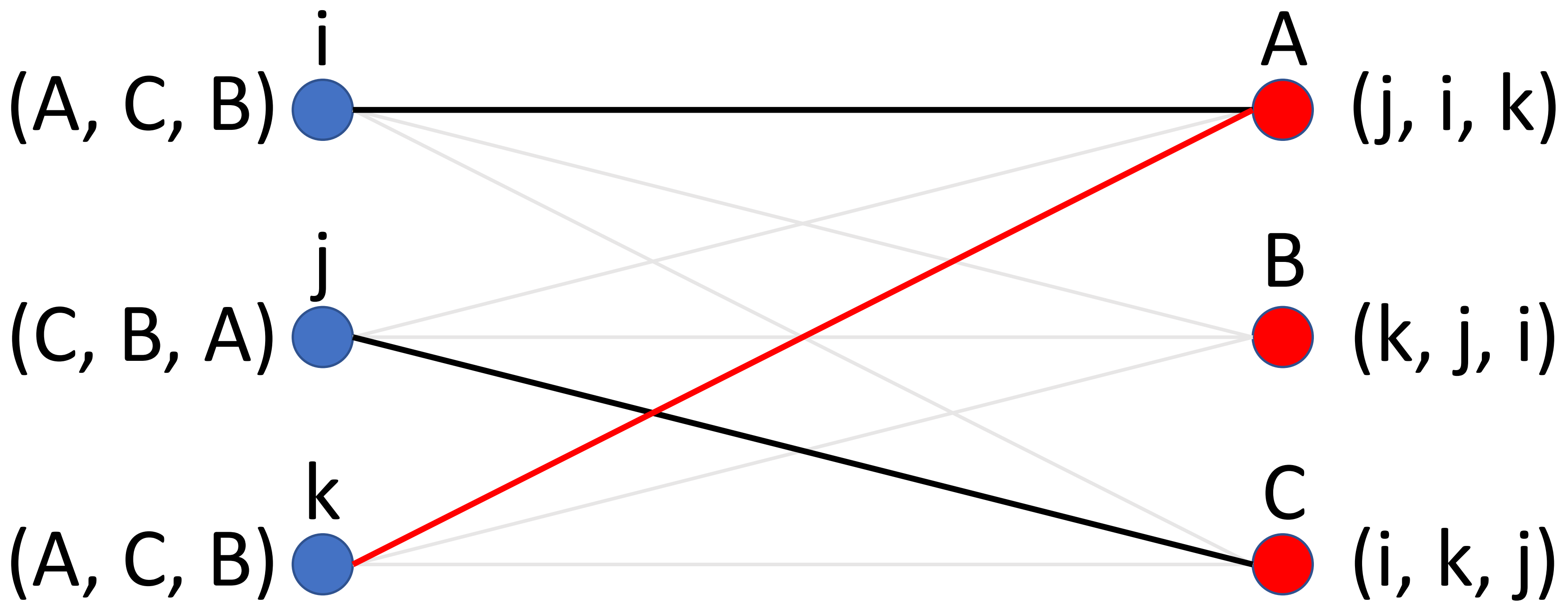


The Proposal Algorithm

Propose to
most preferred

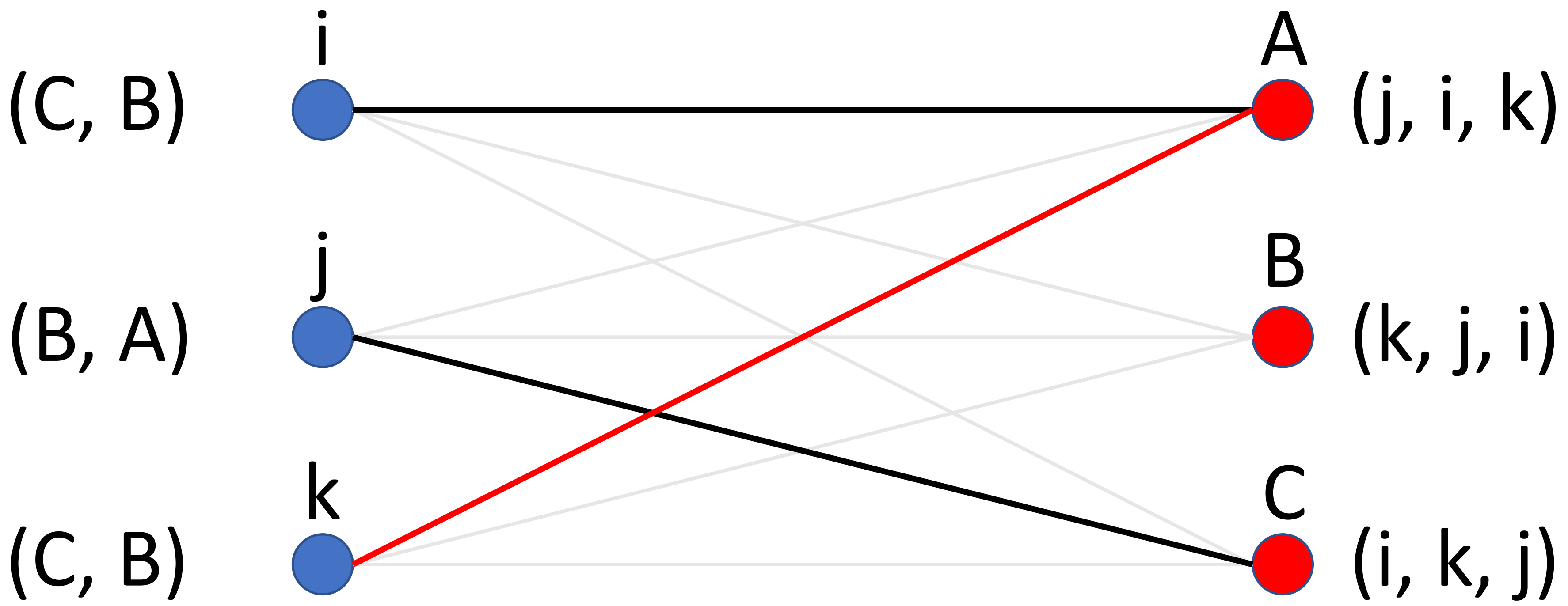


The Proposal Algorithm



Accept best

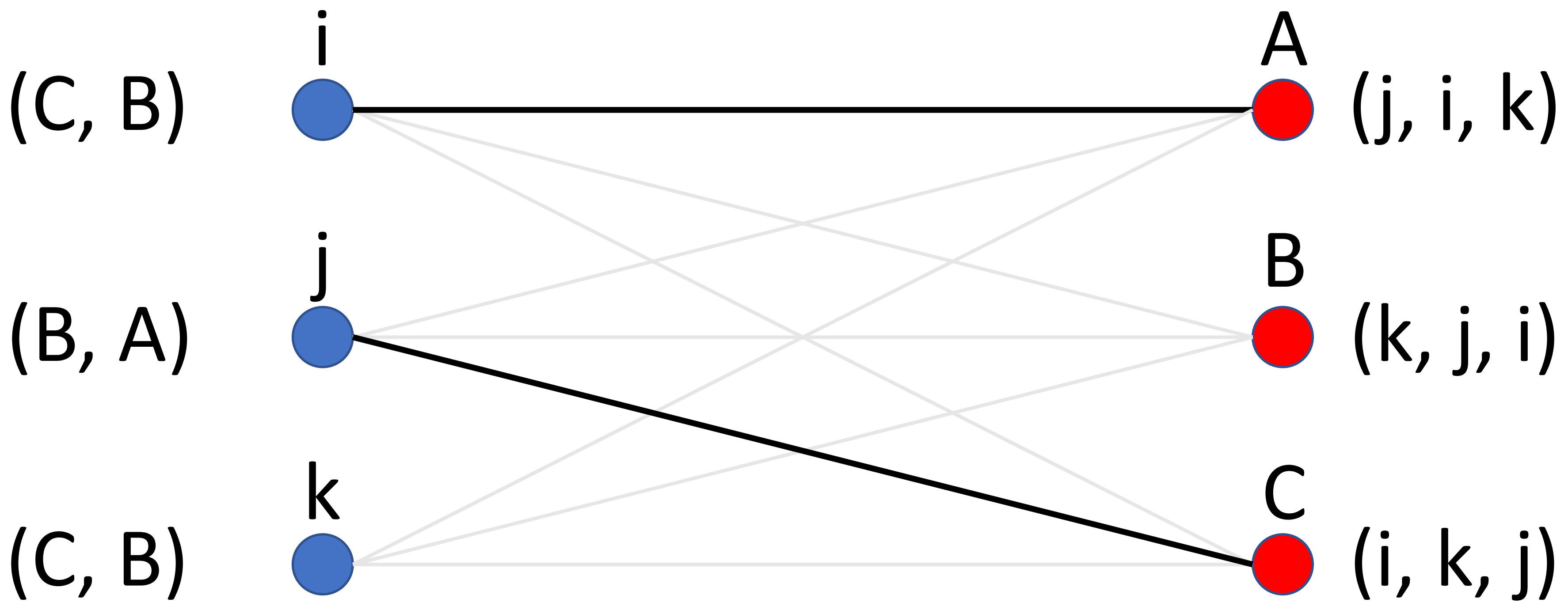
The Proposal Algorithm



Drop proposed nodes from the preference list

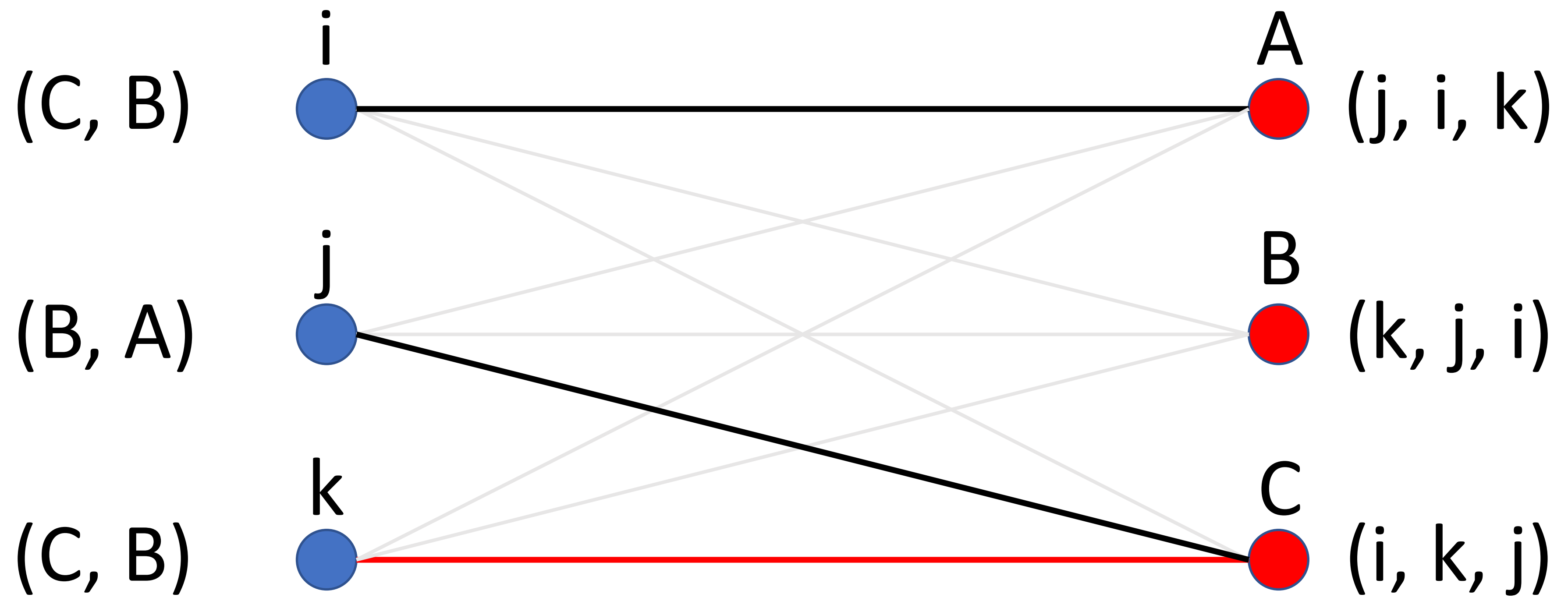
The Proposal Algorithm

Only k is unmatched in round 2

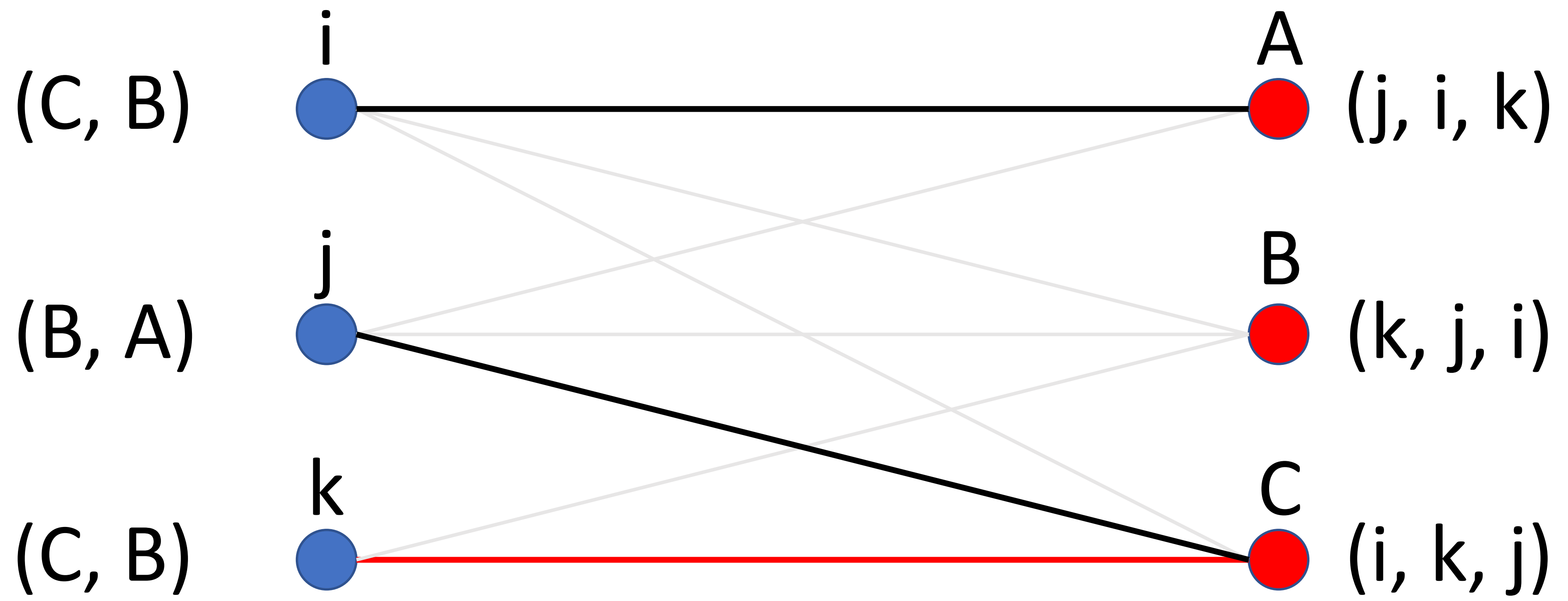


The Proposal Algorithm

Only k is unmatched in round 2

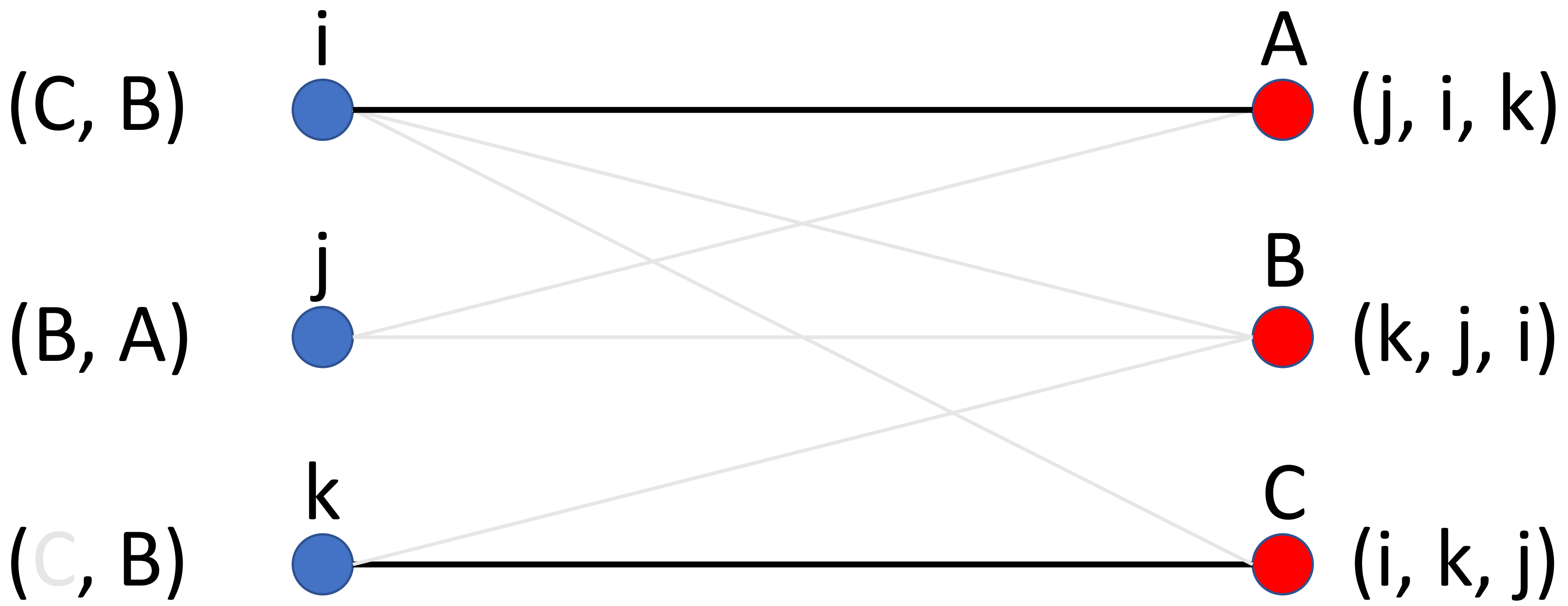


The Proposal Algorithm



New proposal is better

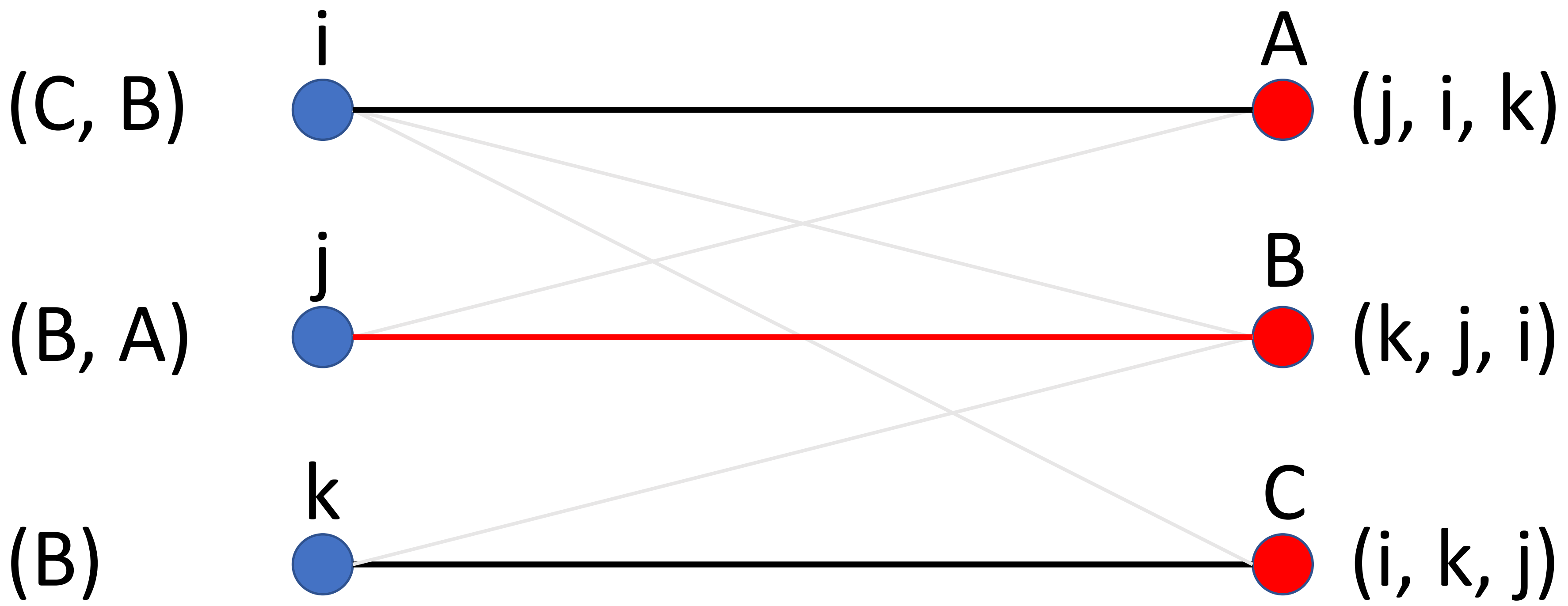
The Proposal Algorithm



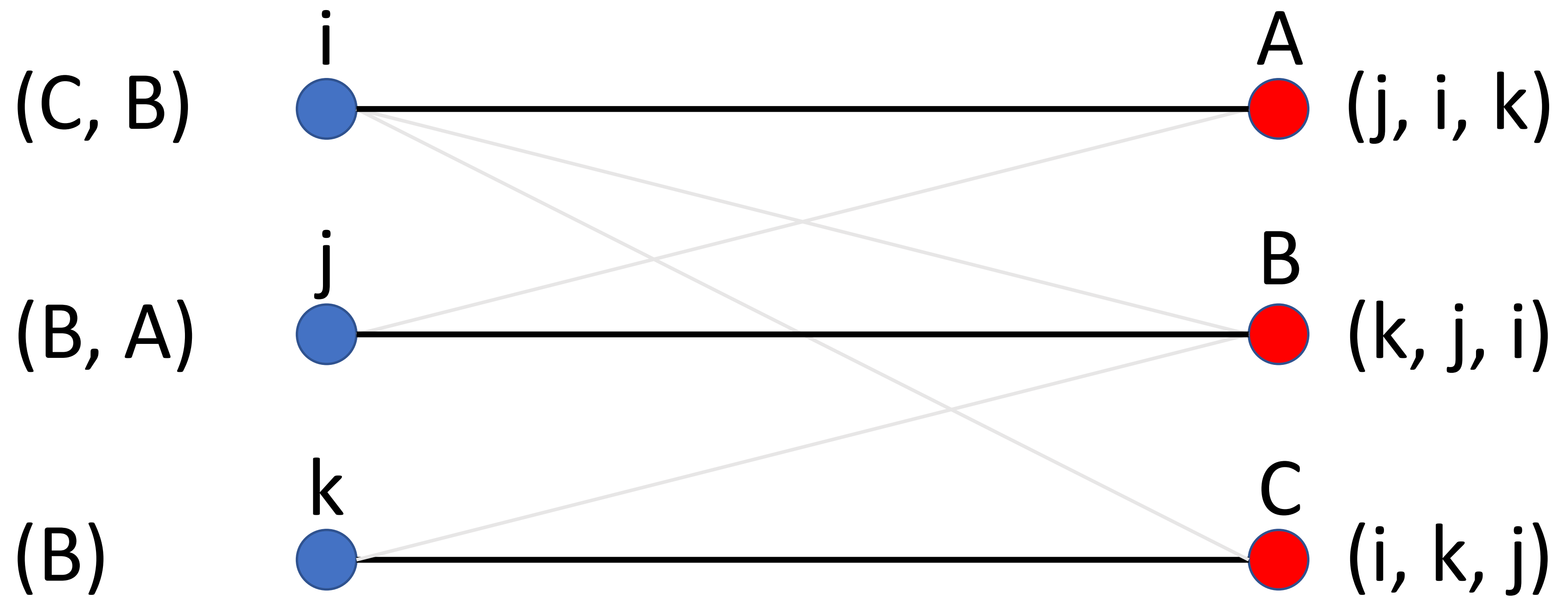
New proposal is better

The Proposal Algorithm

Round 3



The Proposal Algorithm



No one is unmatched anymore

Outline

- Problem setup
  - Modeling: Define the stable matching problem
- Design
  - Try out some ideas
  - Specify an algorithm
- Analysis
  - Correctness
  - Runtime

Stable Matching – The Proposal Algorithm

Correctness:
The proposal algorithm outputs a stable matching.

Runtime:
The proposal algorithm requires $O(n^2)$ proposals.

The Proposal Algorithm - Runtime

Correctness:
The proposal algorithm outputs a stable matching.

Runtime:
The proposal algorithm requires $O(n^2)$ proposals.

The Proposal Algorithm - Runtime

Runtime:
The proposal algorithm requires $O(n^2)$ proposals.

Idea:
Every time a blue node proposes, the preference list gets shorter

The Proposal Algorithm - Runtime

When blue node u proposes (in round i), we reduce $\Phi_i(u)$ by one, i.e.,$\Phi_{i+1}(u) \coloneqq \Phi_i(u) - 1$.

The Proposal Algorithm - Runtime

When blue node u proposes (in round i), we reduce $\Phi_i(u)$ by one, i.e.,$\Phi_{i+1}(u) \coloneqq \Phi_i(u) - 1$.

Let's drop the round index for clarity…

The Proposal Algorithm - Runtime

When blue node u proposes, we reduce $\Phi(u)$ by one.

The Proposal Algorithm - Runtime

When blue node u proposes, we reduce $\Phi(u)$ by one.

$\Phi(u)$ is the length of the preference list.

The Proposal Algorithm - Runtime

When blue node u proposes, we reduce $\Phi(u)$ by one.

$\Phi(u)$ is the length of the preference list.

When $\Phi = \Sigma_{u \in V} \Phi(u) = 0$, all proposal lists are empty and the algorithm has terminated.

The Proposal Algorithm - Runtime

When blue node u proposes, we reduce $\Phi(u)$ by one.

$\Phi(u)$ is the length of the preference list.

When $\Phi(\mathrm{V}) = \Sigma_{u \in V}\, \Phi(u) = 0$, all proposal lists are empty and the algorithm has terminated.

In total, we have $\Phi(\mathrm{V}) = \Sigma_{u \in V} \Phi(u) \leq n^2$

The Proposal Algorithm - Runtime

When blue node u proposes, we reduce $\Phi(u)$ by one.

$\Phi(u)$ is the length of the preference list.

When $\Phi(\mathrm{V}) = \Sigma_{u \in V}\, \Phi(u) = 0$, all proposal lists are empty and the algorithm has terminated.

In total, we have $\Phi(\mathrm{V}) = \Sigma_{u \in V} \Phi(u) \leq n^2$

Runtime:
The proposal algorithm requires $O(n^2)$ proposals.

A Remark about the Runtime

- Rounds are nice for intuition, but it can be that in one round, only one node proposes. In the worst case, we need $\Omega(n^2)$ rounds. With a naïve implementation, this could yield a $\Omega(n^3)$ runtime.
  - In a round, only unmatched nodes propose
  - For an efficient implementation, need efficient access to unmatched nodes.

The Proposal Algorithm

Correctness:
The proposal algorithm outputs a stable matching.

Runtime:
The proposal algorithm requires $O(n^2)$ proposals.

The Proposal Algorithm - Correctness

Correctness:
The proposal algorithm outputs a stable matching.

The Proposal Algorithm - Correctness

Roadmap:
Consider an unstable edge $\{u, v\}$.

Correctness:
The proposal algorithm outputs a stable matching.

The Proposal Algorithm - Correctness

Roadmap:
Consider an unstable edge $\{u, v\}$.

Correctness:
The proposal algorithm outputs a stable matching.

Blue node u proposed to v at some point.

For red node v, the situation can only get better.

The Proposal Algorithm - Correctness

Correctness:
The proposal algorithm outputs a stable matching.

Roadmap:
Consider an unstable edge $\{u, v\}$.

Blue node u proposed to v at some point.

For red node v, the situation can only get better.

Node v would have matched to u and not changed

The Proposal Algorithm - Correctness

A red node only improves

Claim:
Consider an edge $\{u, v\}$ in the matching in the end of round i, where u is blue and v is red.
For all rounds $j > i$, if v is matched to node w, then $w >_v u$.

The Proposal Algorithm - Correctness

A red node only improves

Claim:
Consider an edge $\{u, v\}$ in the matching in the end of round i, where u is blue and v is red.
For all rounds $j > i$, if v is matched to node w, then $w >_v u$.

Proof:
A blue node does not propose if it is matched and hence, the match of v can only change if v accepts another proposal.
According to the proposal algorithm, v only accepts a proposal from node w if $w >_v u$.

The Proposal Algorithm - Correctness

Blue node u has proposed over an unstable edge at some point

The Proposal Algorithm - Correctness

Blue node u has proposed over an unstable edge at some point

Claim:
Consider an unstable edge $\{u, v\}$ (after termination), where u is blue and v is red.
In some round i, node u proposed to v.

The Proposal Algorithm - Correctness

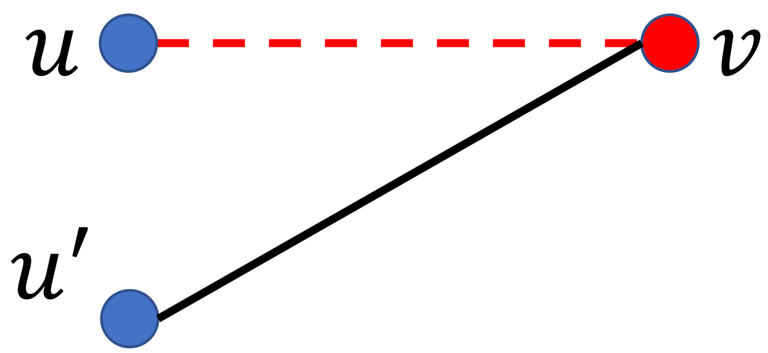
Blue node u has proposed over an unstable edge at some point

Claim:
Consider an unstable edge $\{u, v\}$ (after termination), where u is blue and v is red.
In some round i, node u proposed to v.

Proof:
1) If u is unmatched, then u must have proposed to all nodes before stopping.
2) If u is matched, consider the matched edge $\{u, w\}$. Since $\{u, v\}$ is unstable, it must be the case that $w <_u v$. Since the proposal algorithm proposes according to the preference ordering, it must be the case that u proposed to v.

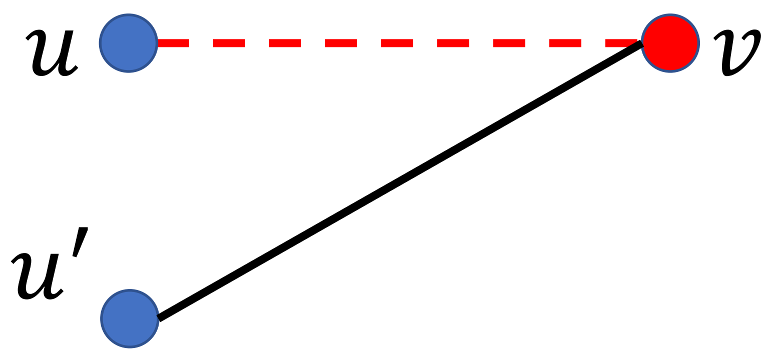
The Proposal Algorithm - Correctness

Suppose edge $\{u, v\}$ is unstable in the end.
We have that $u' <_v u$



The Proposal Algorithm - Correctness

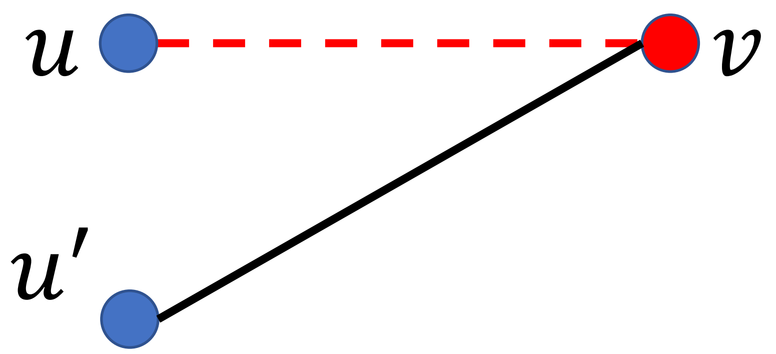
Suppose edge $\{u, v\}$ is unstable in the end.
We have that $u' <_v u$



Claim 1:
u proposed to v,
before proposing to v'

The Proposal Algorithm - Correctness

Suppose edge $\{u, v\}$ is unstable in the end.
We have that $u' <_v u$



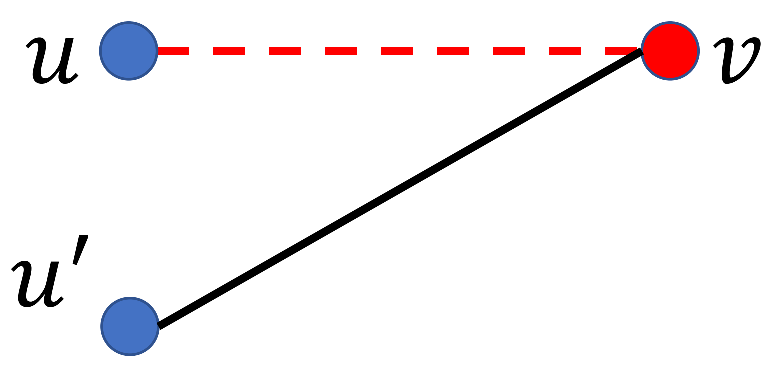
Claim 1:
u proposed to v,
before proposing to v'

$\longrightarrow$

Claim 2:
Node v has a match that is at least as good as u.

The Proposal Algorithm - Correctness

Suppose edge $\{u, v\}$ is unstable in the end.
We have that $u' <_v u$



Claim 1:
u proposed to v,
before proposing to v'

Claim 2:
Node v has a match that is at least as good as u.

Since preferences are strict:

$$u' >_v u$$

A contradiction!

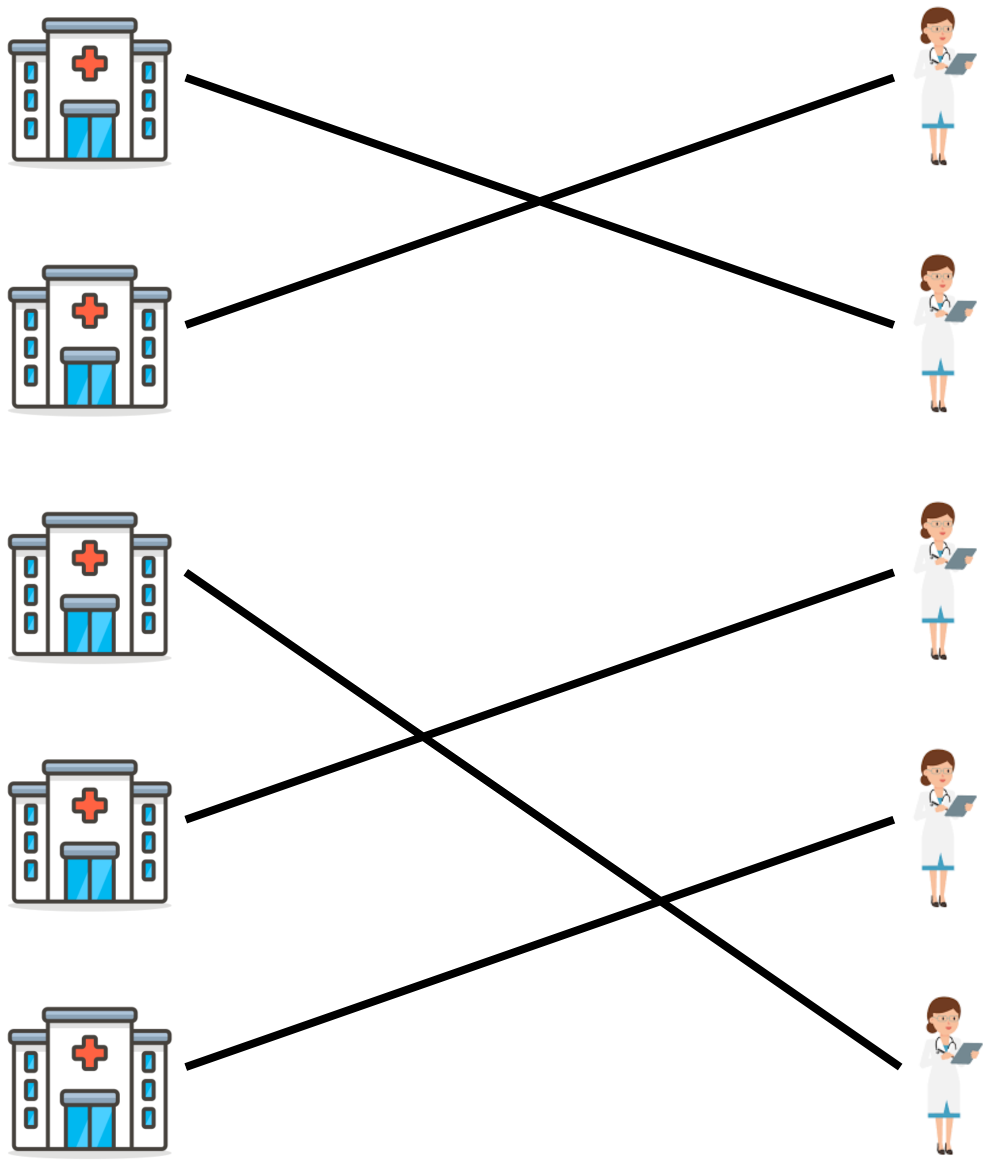
The Proposal Algorithm

Correctness:
The proposal algorithm outputs a stable matching.

Runtime:
The proposal algorithm requires $O(n^2)$ proposals.

Wrap-up

Stable Matching:
No pair unhappy



The proposal algorithm

Runtime

Correctness