15-251
Great Theoretical Ideas in Computer Science

Graphs Algorithms II: Stable and Maximum Matchings

February 19th, 2015

Finding internship

Bob

Macrosoft

Moogle
Google

UMBRELLA
CORPORATION
"Our business is life itself..."

Finding internship

1.
2.
3.
4.

Bob

1. Alice
2. Bob
3. Charlie
4. David

•

•

•

1. Bob
2. David
3. Alice
4. Charlie

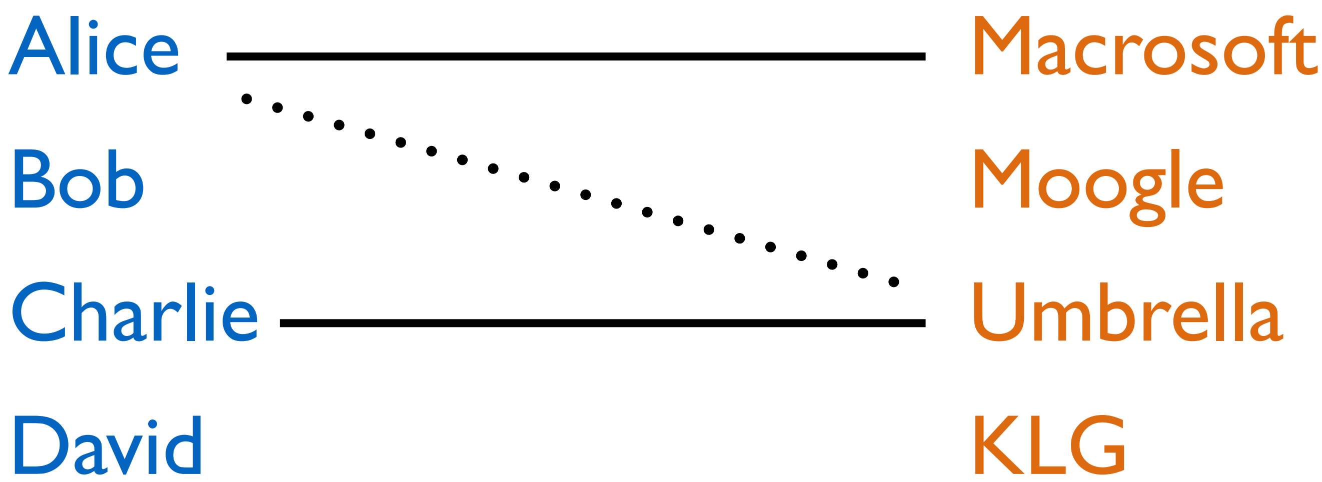
Other examples:

medical residents - hospitals

students - colleges

Finding internship

What can go wrong?



Suppose Alice gets "matched" with Macrosoft.
Charlie gets "matched" with Umbrella.

But, say, Alice prefers Umbrella over Macrosoft
and Umbrella prefers Alice over Charlie.

Steps we'll follow

1. Formulate the problem

2. Is there a trivial algorithm?

3. Is there a better algorithm?

4. Analyze the algorithm

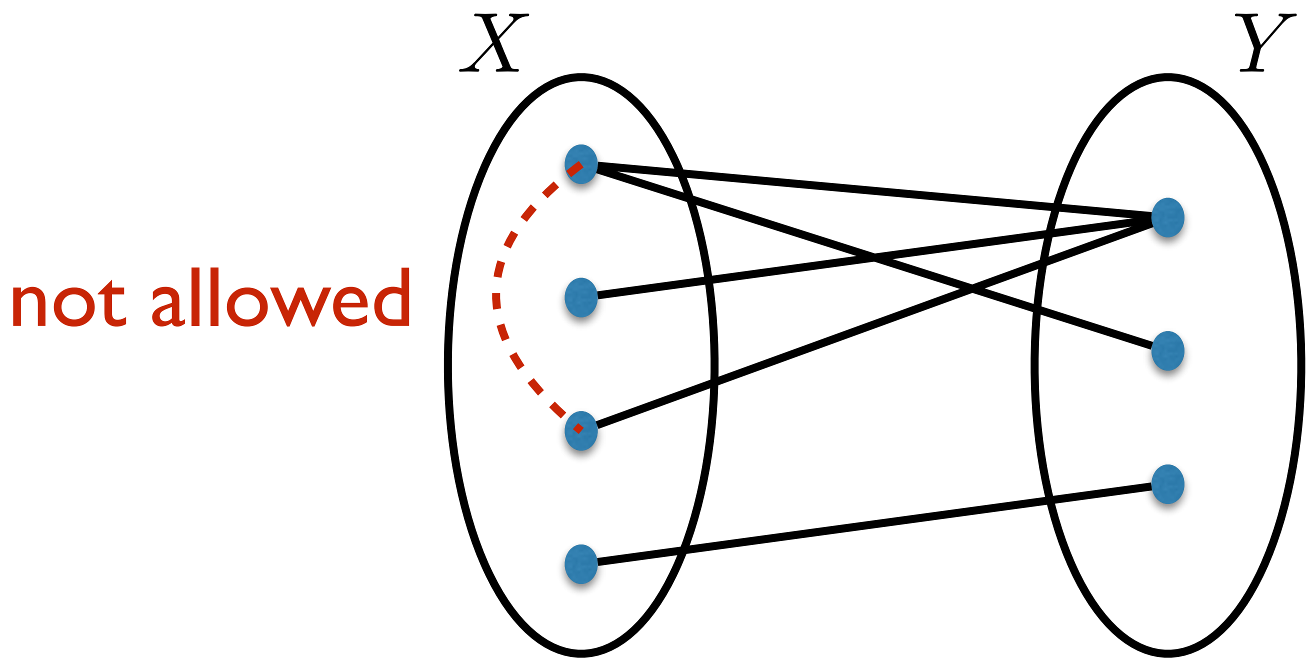
Step 1: Formulate the problem

Formulate the problem mathematically,
and focus on a meaningful simplification.

The purpose:

- Get rid of all the distractions
- Identify the crux of the problem
- Get a clean mathematical model that is easy to reason about.

Bipartite Graphs



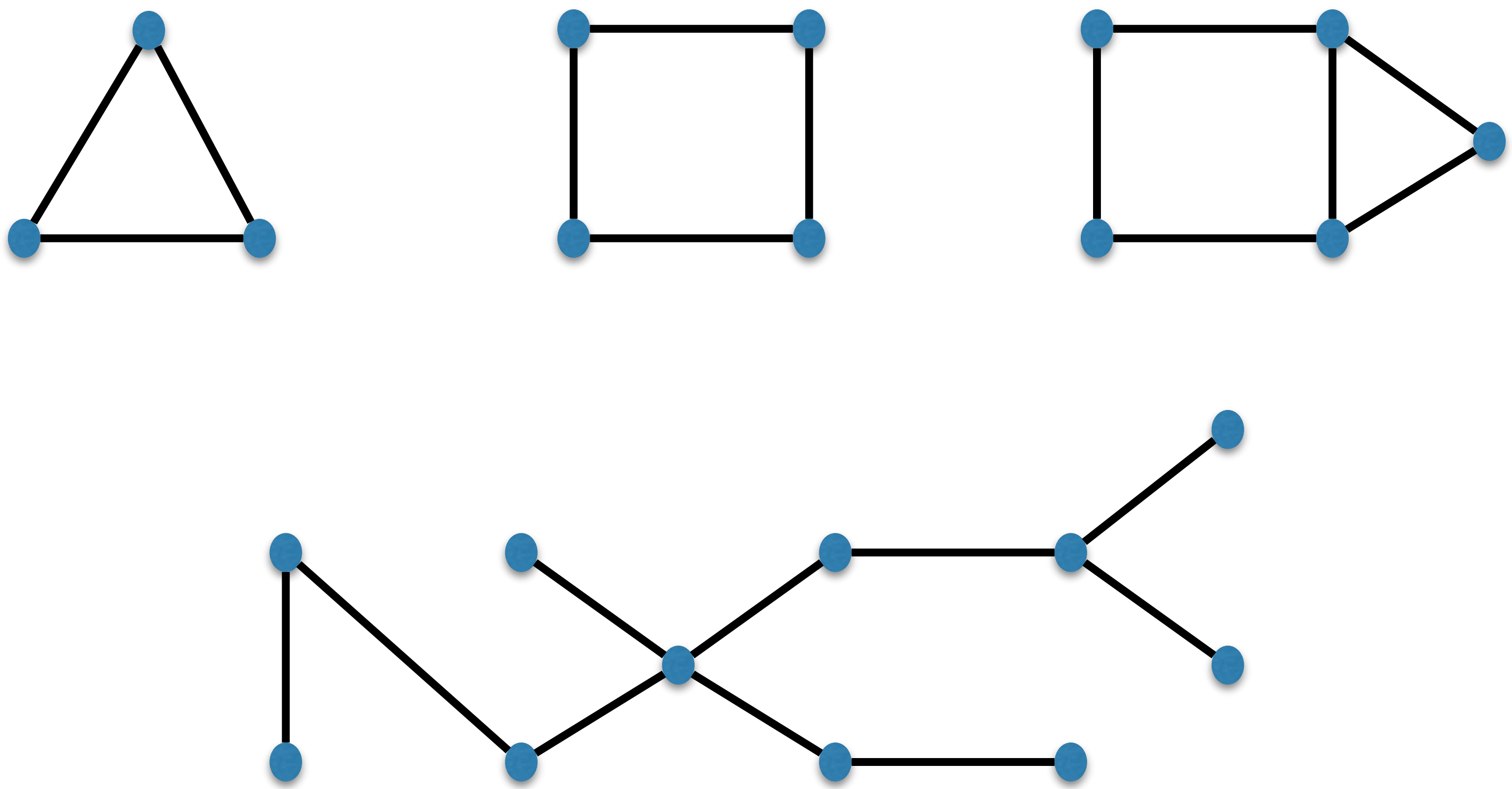
$G = (V, E)$ is **bipartite** if:

- there exists a bipartition X and Y of V
- each edge connects a vertex in X to a vertex in Y

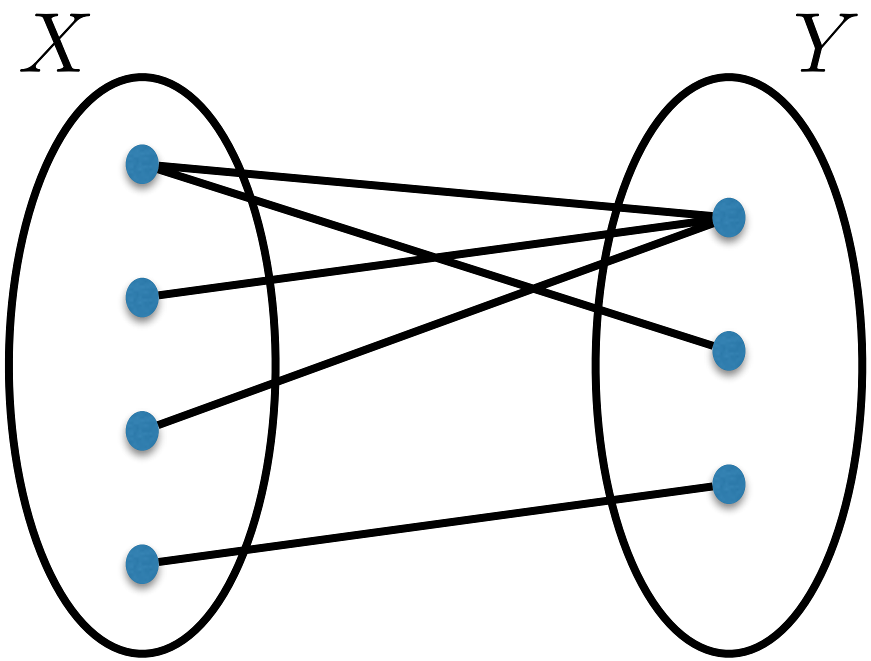
Given a graph $G = (V, E)$, we could ask, is it bipartite?

Bipartite Graphs

Given a graph $G = (V, E)$, we could ask, is it bipartite?



Bipartite Graphs

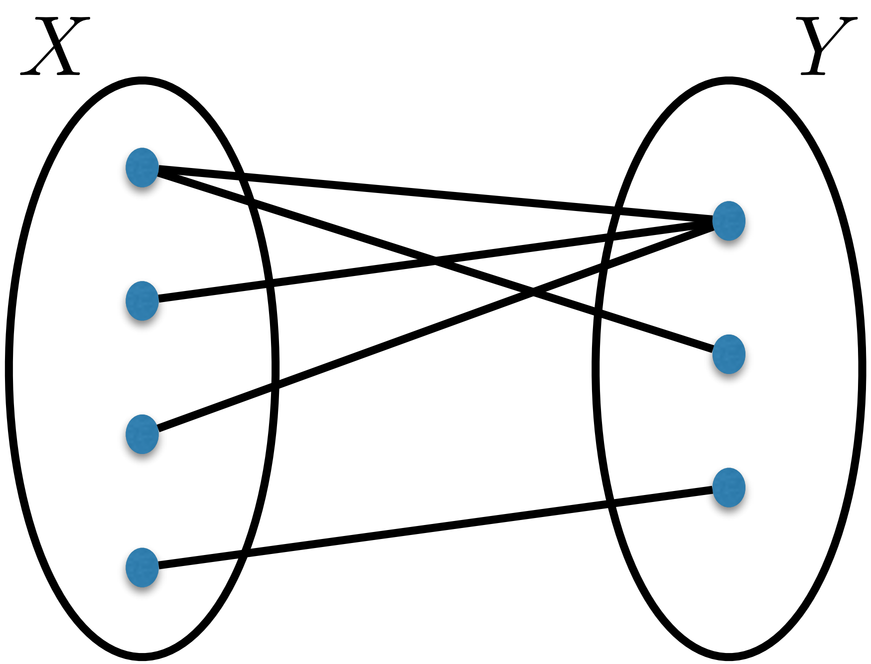


Sometimes we write the bipartition explicitly:

$$G = (X, Y, E)$$

Matchings in bipartite graphs

Often, we are interested in find a **matching** in a bipartite graph

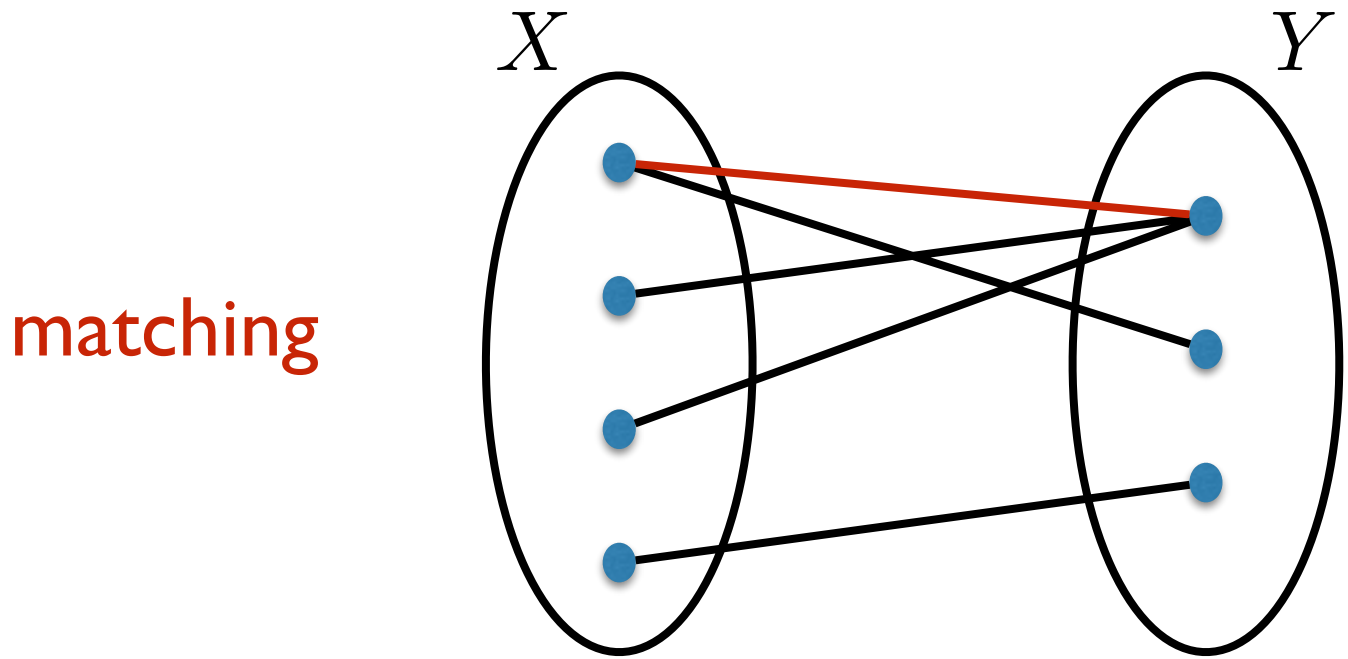


A **matching** :

A subset of the edges that do not share an endpoint.

Matchings in bipartite graphs

Often, we are interested in find a **matching** in a bipartite graph

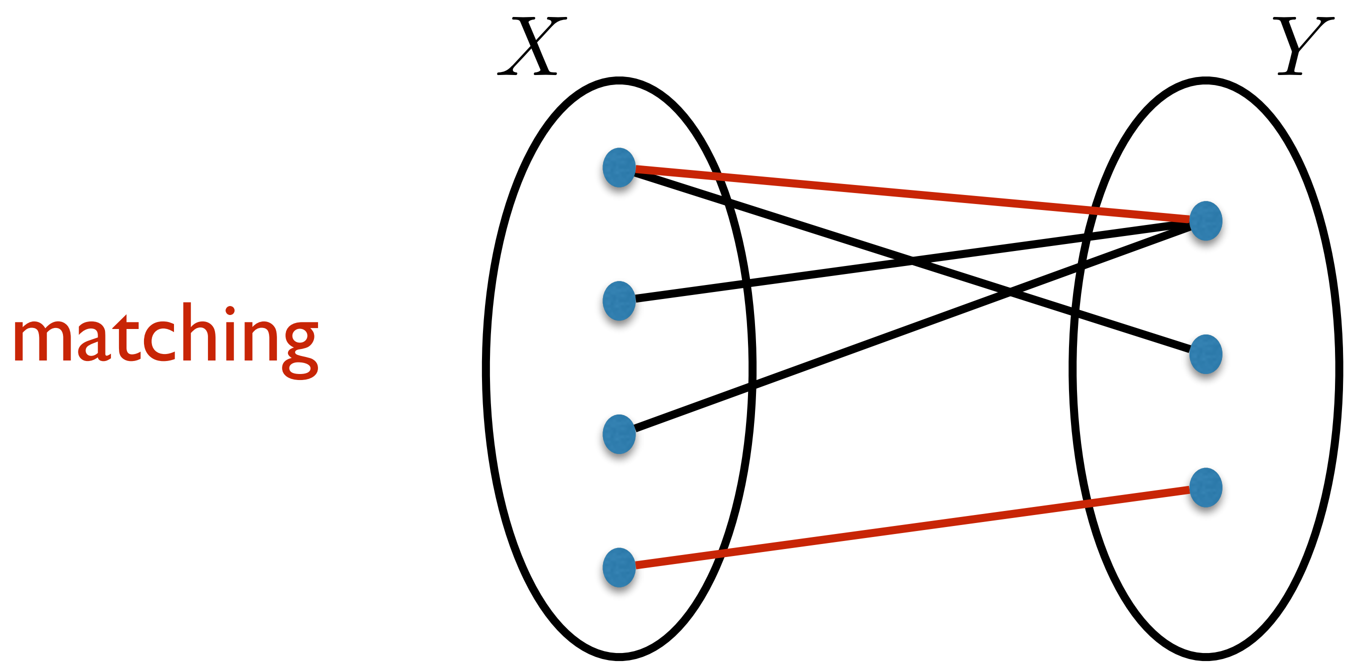


A **matching** :

A subset of the edges that do not share an endpoint.

Matchings in bipartite graphs

Often, we are interested in find a **matching** in a bipartite graph

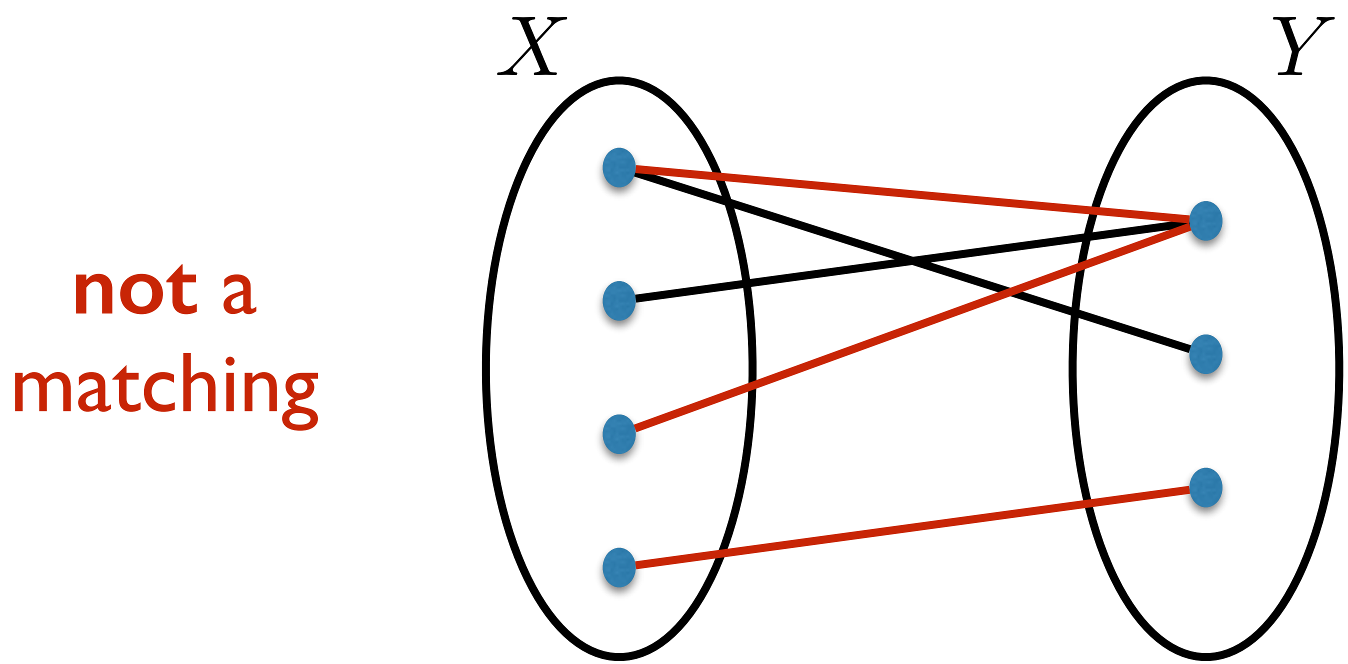


A **matching** :

A subset of the edges that do not share an endpoint.

Matchings in bipartite graphs

Often, we are interested in find a **matching** in a bipartite graph



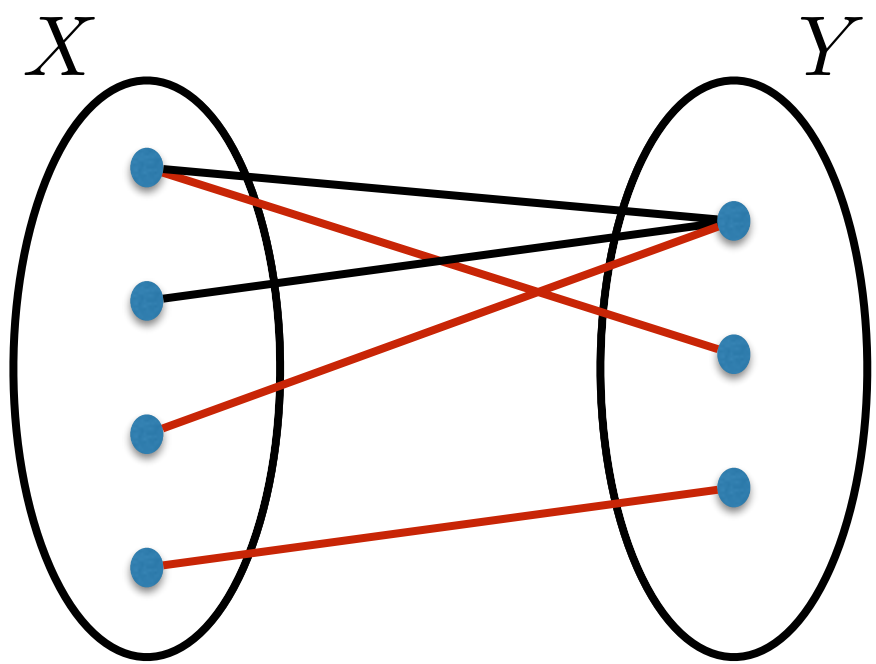
A **matching** :

A subset of the edges that do not share an endpoint.

Matchings in bipartite graphs

Often, we are interested in find a **matching** in a bipartite graph

maximum matching



A **matching** :

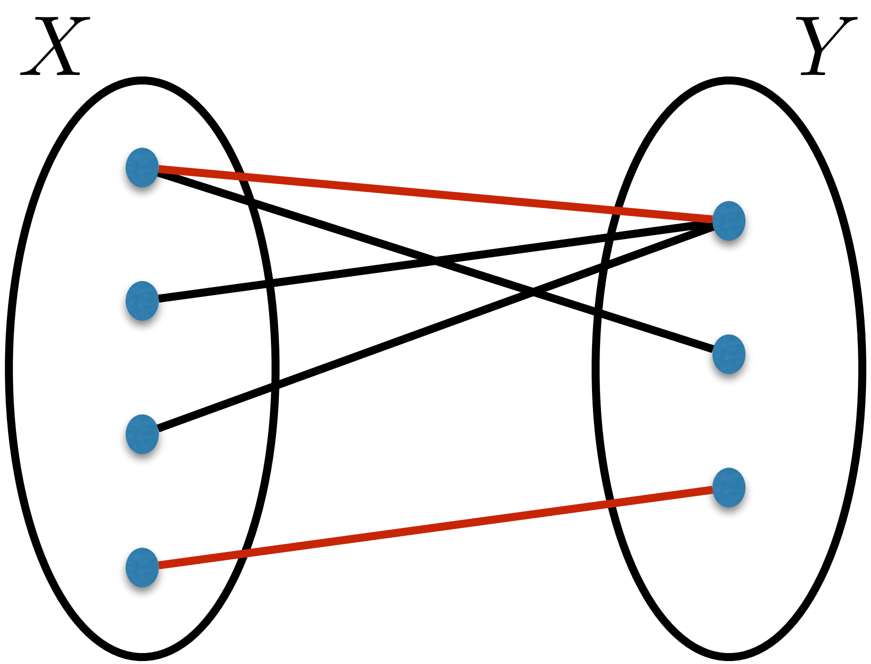
A subset of the edges that do not share an endpoint.

Maximum matching: a matching with largest number of edges (among all possible matchings).

Matchings in bipartite graphs

Often, we are interested in find a **matching** in a bipartite graph

maximal matching



Cannot add more edges.

"Local optimum"

A **matching** :

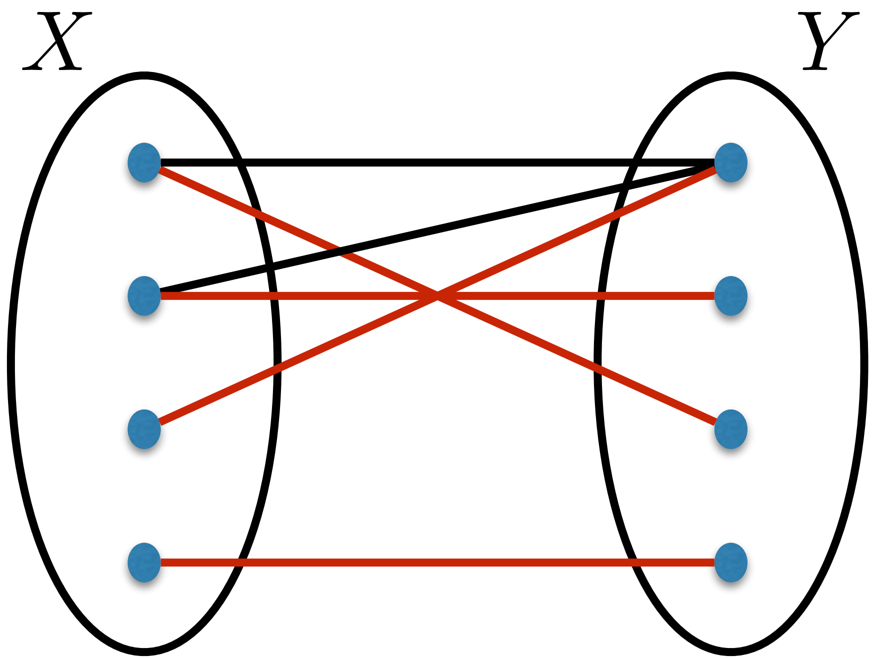
A subset of the edges that do not share an endpoint.

Maximal matching: a matching which cannot contain any more edges.

Matchings in bipartite graphs

Often, we are interested in find a **matching** in a bipartite graph

perfect matching



a necessary condition for perfect matching:
$|X| = |Y|$

A **matching** :

A subset of the edges that do not share an endpoint.

Perfect matching: a matching that covers all vertices.

Bipartite Graphs

Great for modeling relations between two classes of objects.

Examples:

X = machines, Y = jobs

An edge (x, y) means x is capable of doing y.

X = professors, Y = courses

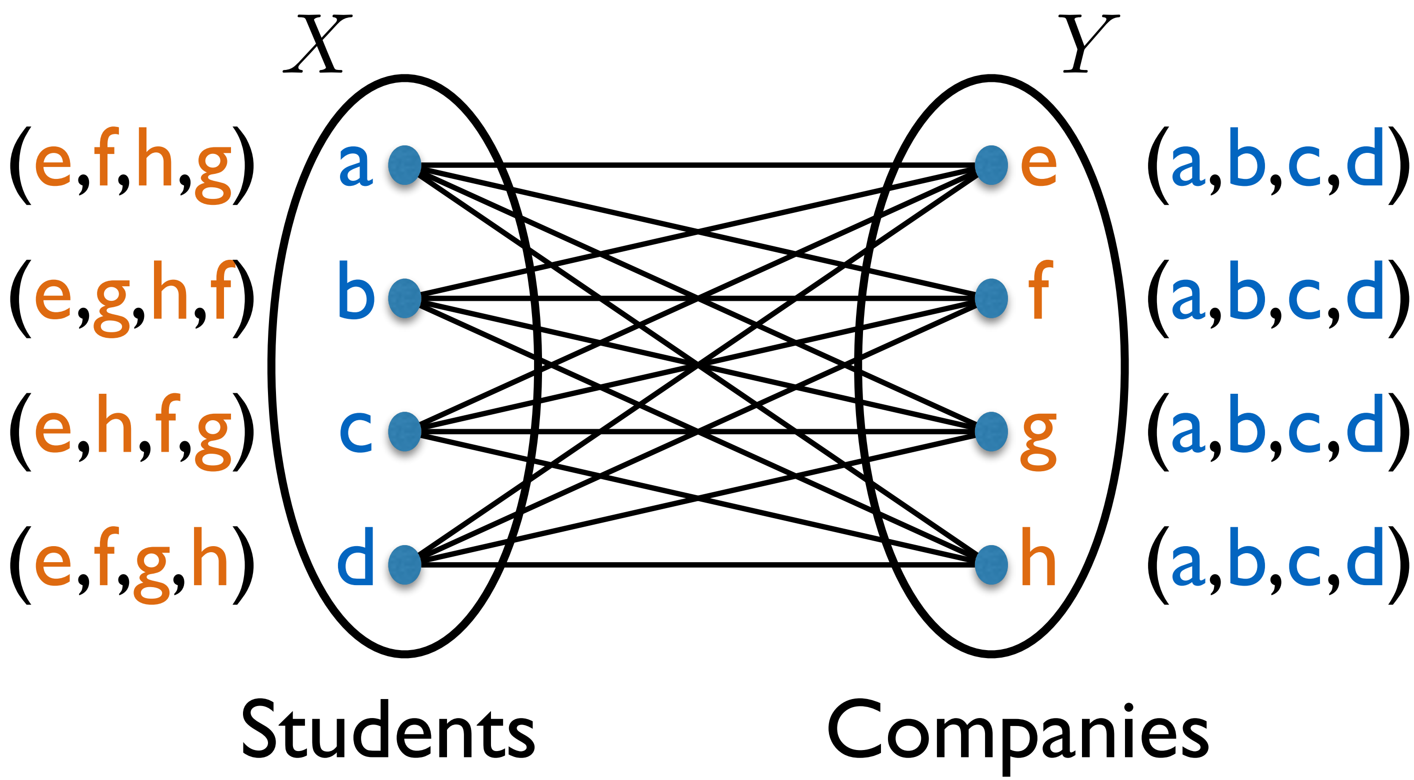
An edge (x, y) means x can teach y.

X = students, Y = internship jobs

An edge (x, y) means x and y are interested in each other.

Back to the internship problem

An instance of the problem can be represented as a complete bipartite graph + preference list of each node.

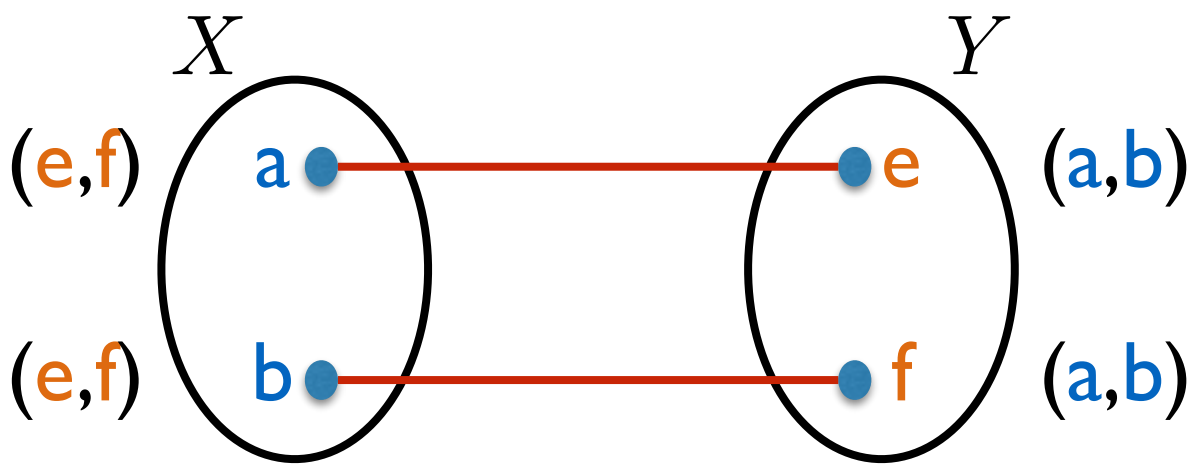


$$|X| = |Y| = n$$

Goal: Find a stable matching.

Back to the internship problem

What is a stable matching?

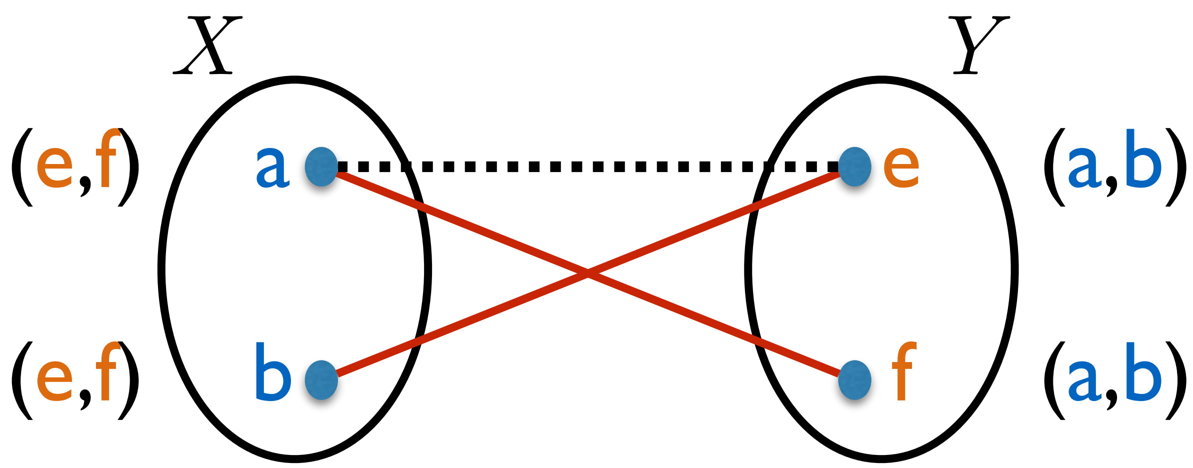


1. It has to be a perfect matching.
2. Cannot contain an unstable pair:

   A pair of vertices u and v which are not matched to each other, BUT they prefer each other to their current partners.

Back to the internship problem

What is a stable matching?



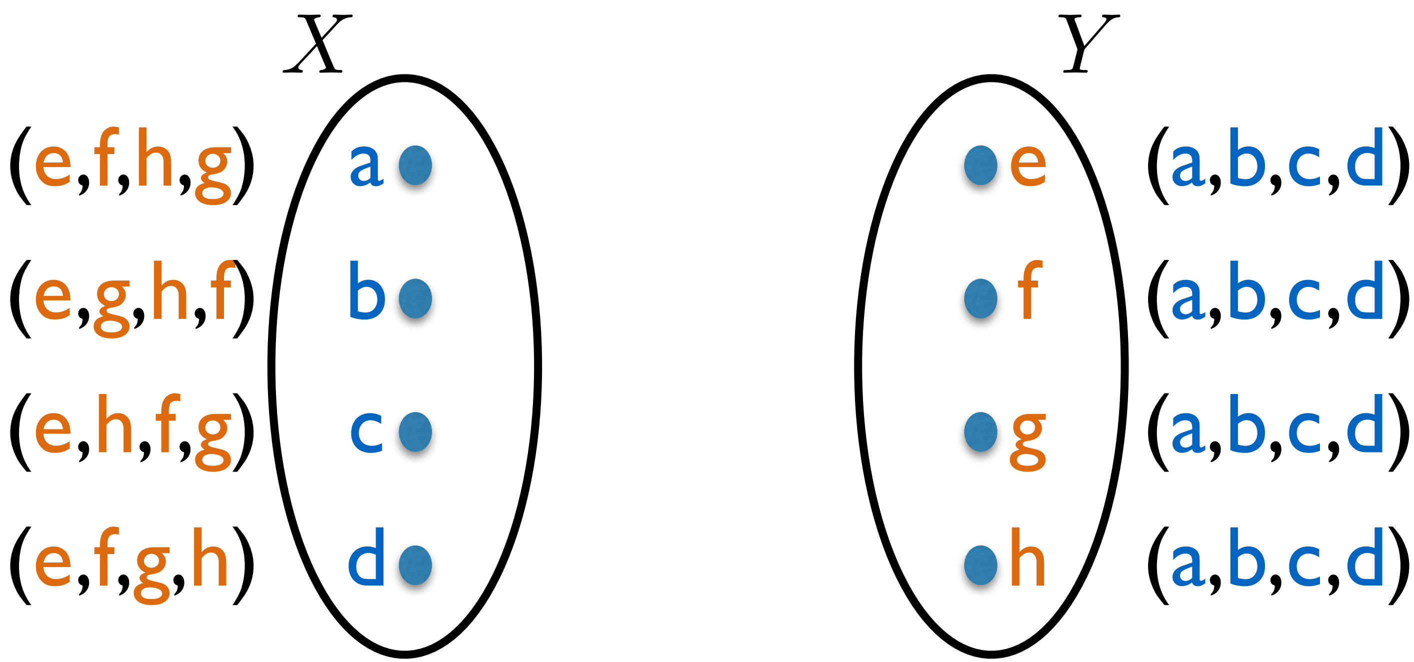
(a, e) is an unstable pair.

1. It has to be a perfect matching.
2. Cannot contain an unstable pair:

   A pair of vertices u and v which are not matched to each other, BUT they prefer each other to their current partners.

Back to the internship problem

An instance of the problem can be represented as a complete bipartite graph + preference list of each node.

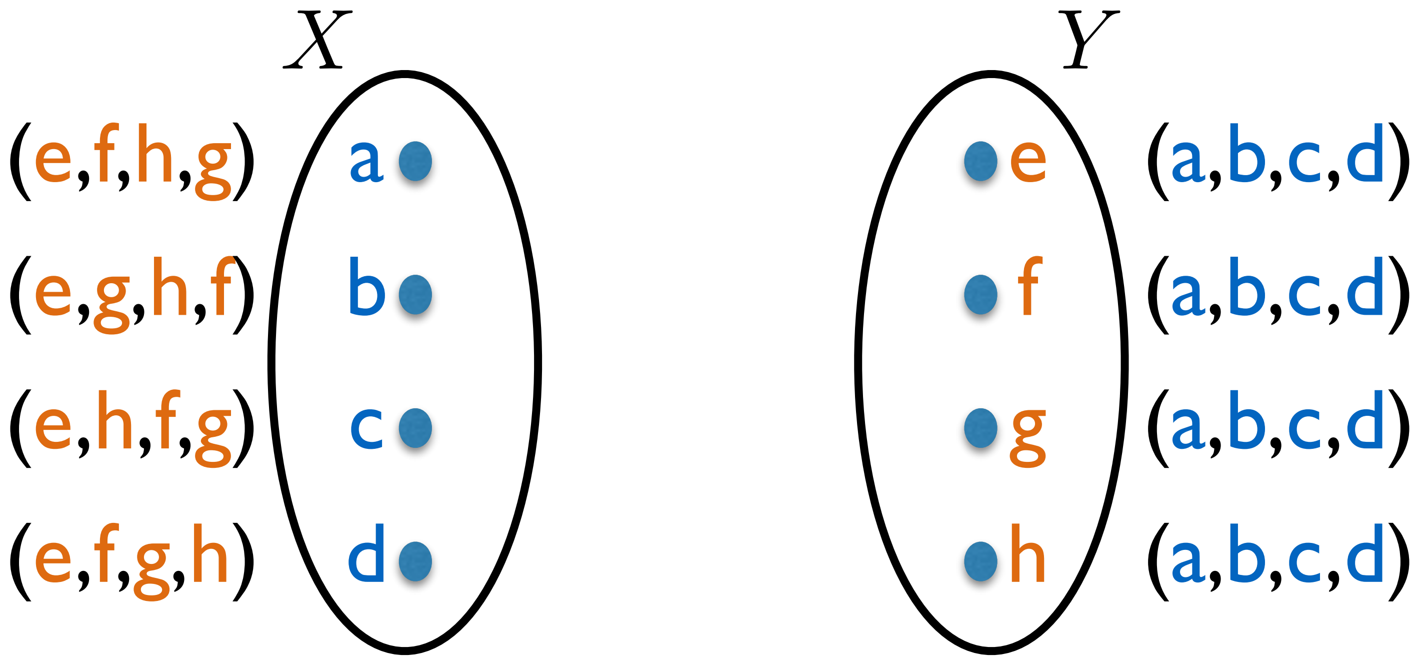


$$|X| = |Y| = n$$

Goal: Find a stable matching.

Is it guaranteed to always exist?

Step 2: Is there a trivial algorithm?



Try all possible perfect matchings,
and check if it is stable.

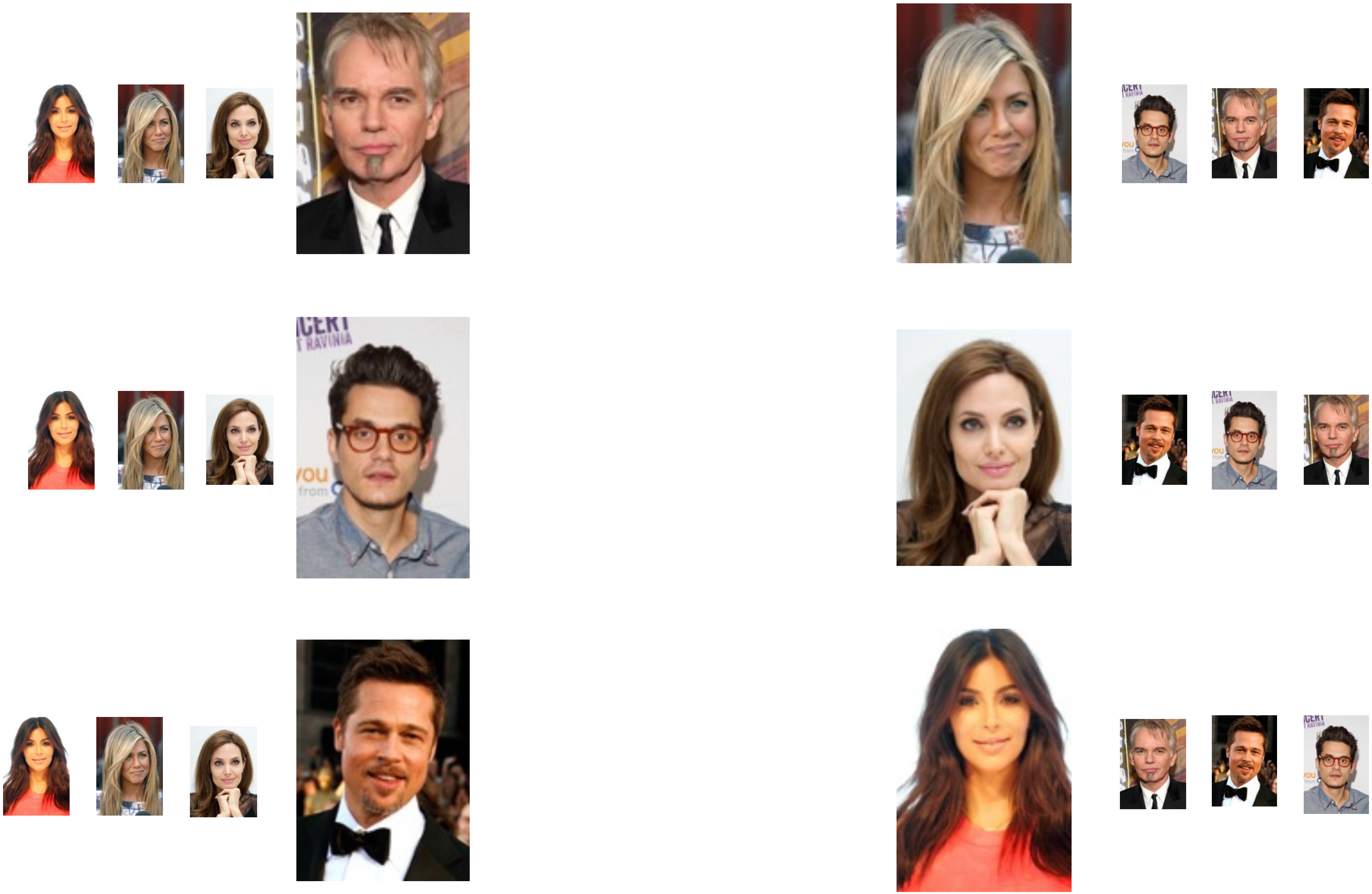
perfect matchings: $n!$ where $n = |X|$.

Step 3: Can we do better?

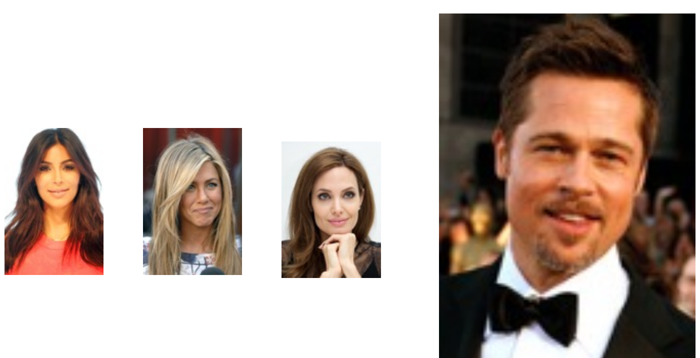
The Gale-Shapley Proposal Algorithm (1962)

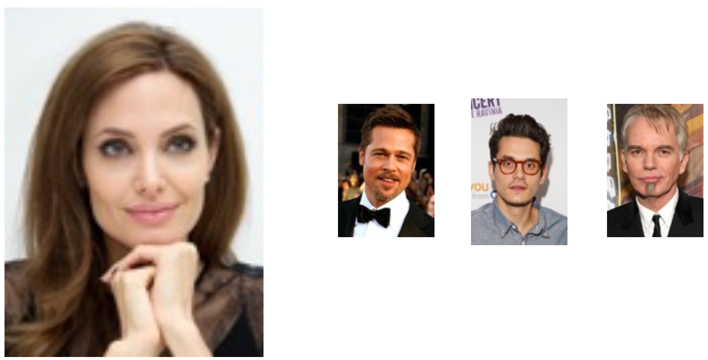
Nobel Prize in Economics
2012

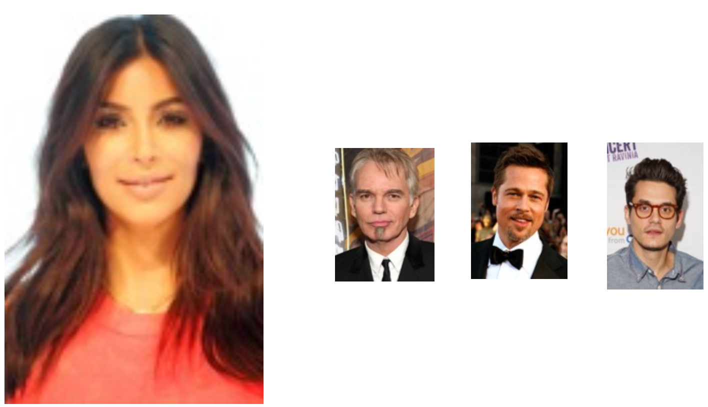
The Gale-Shapley Proposal Algorithm



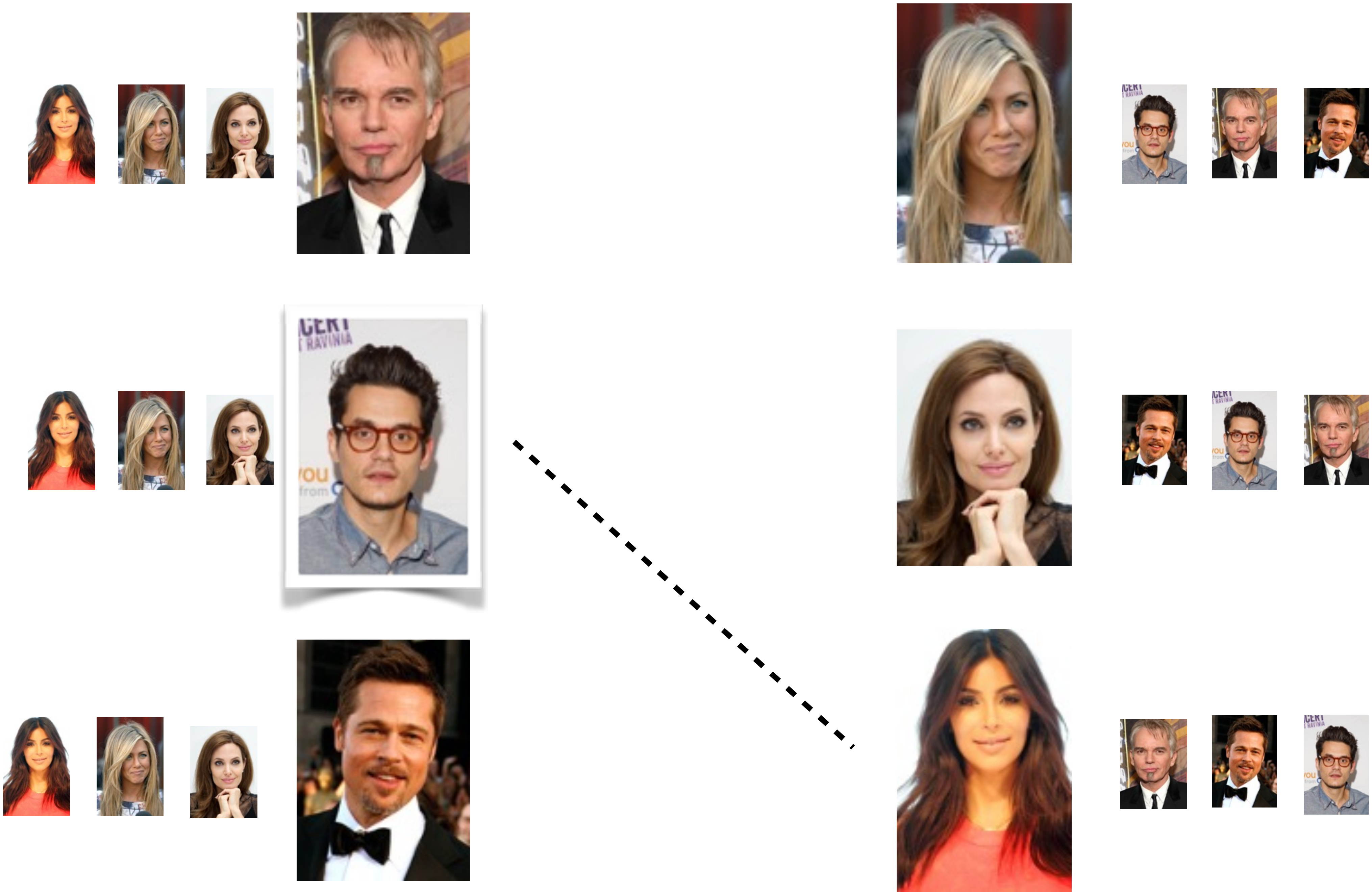
The Gale-Shapley Proposal Algorithm



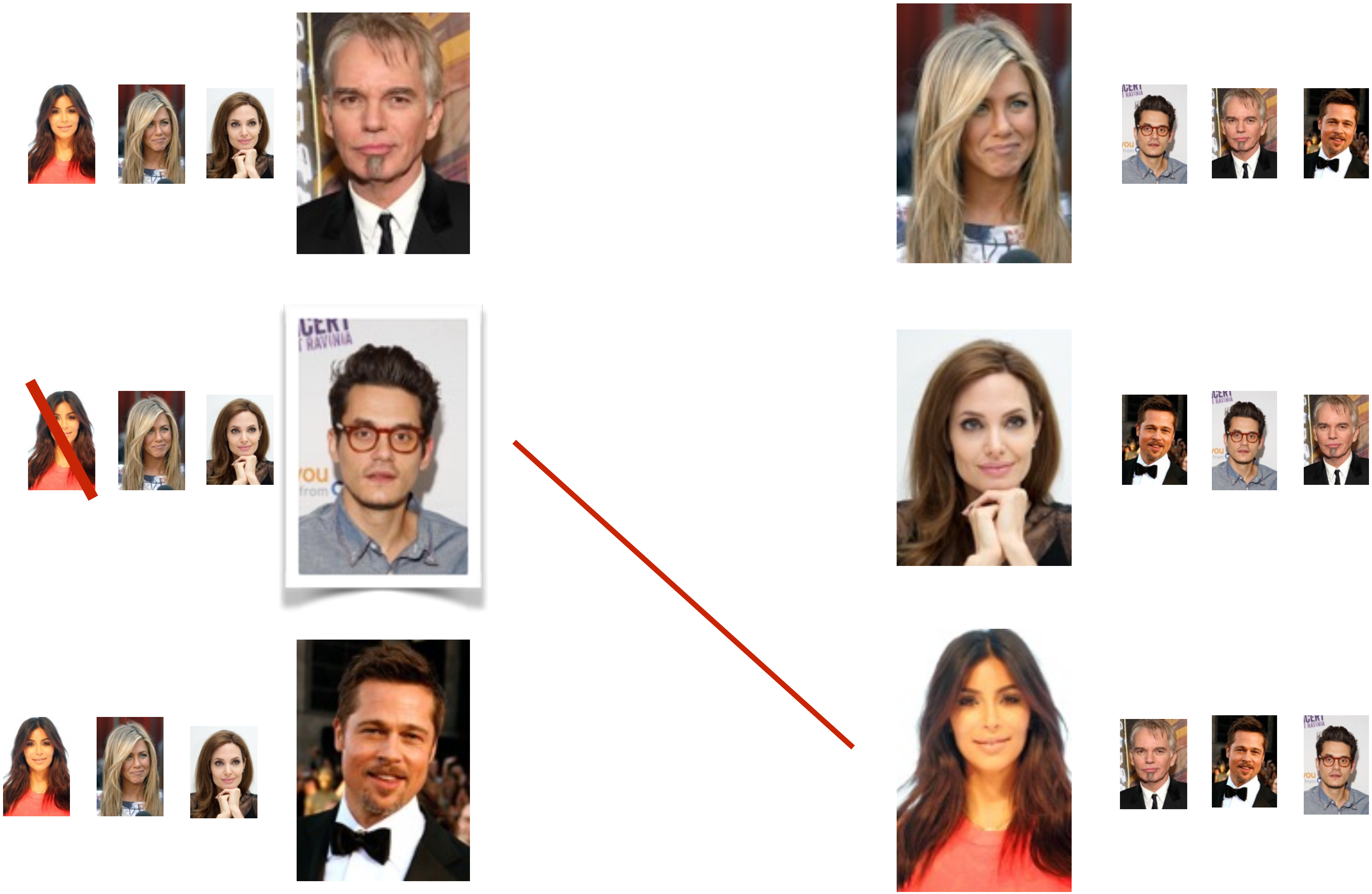




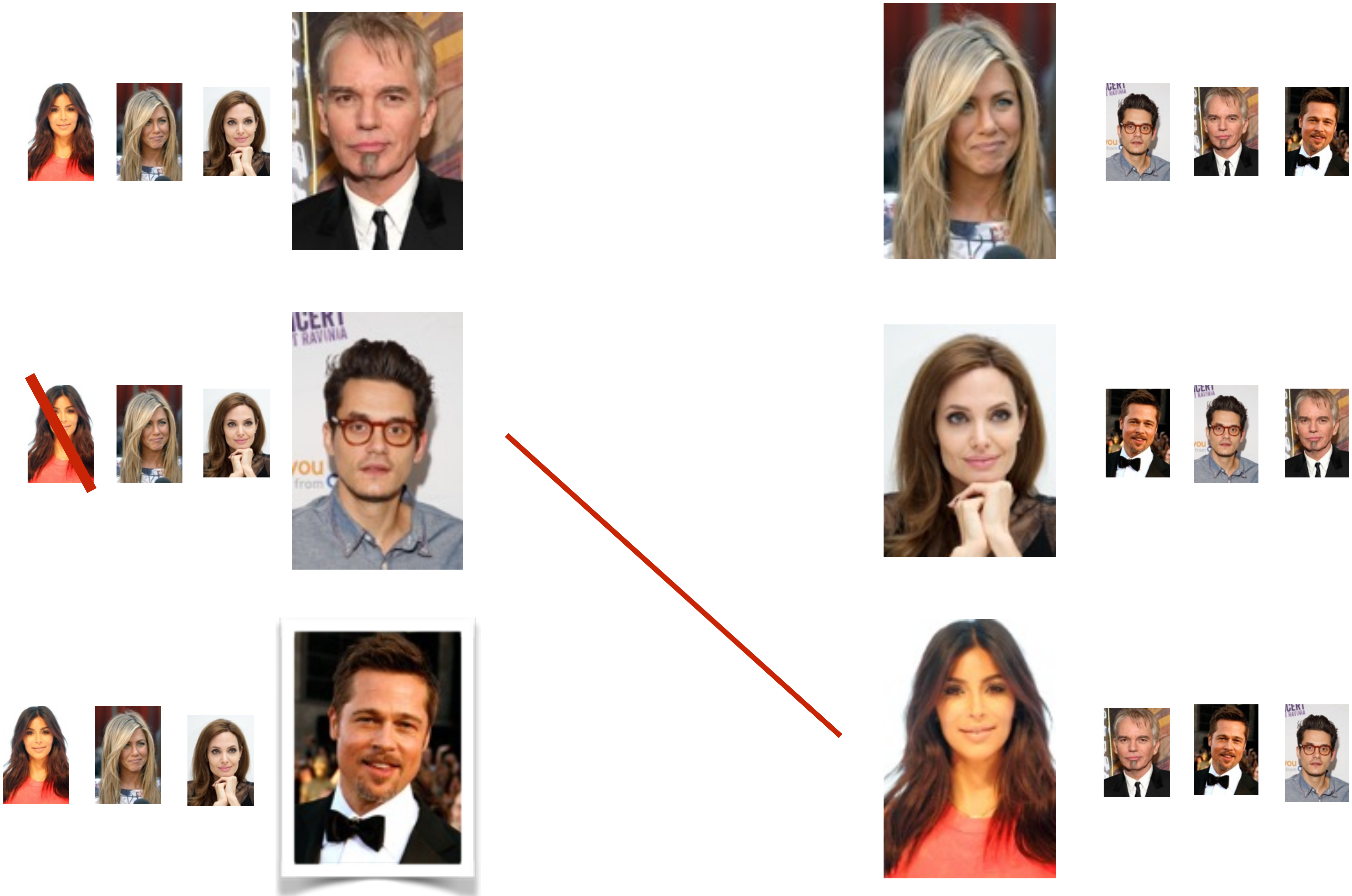
The Gale-Shapley Proposal Algorithm



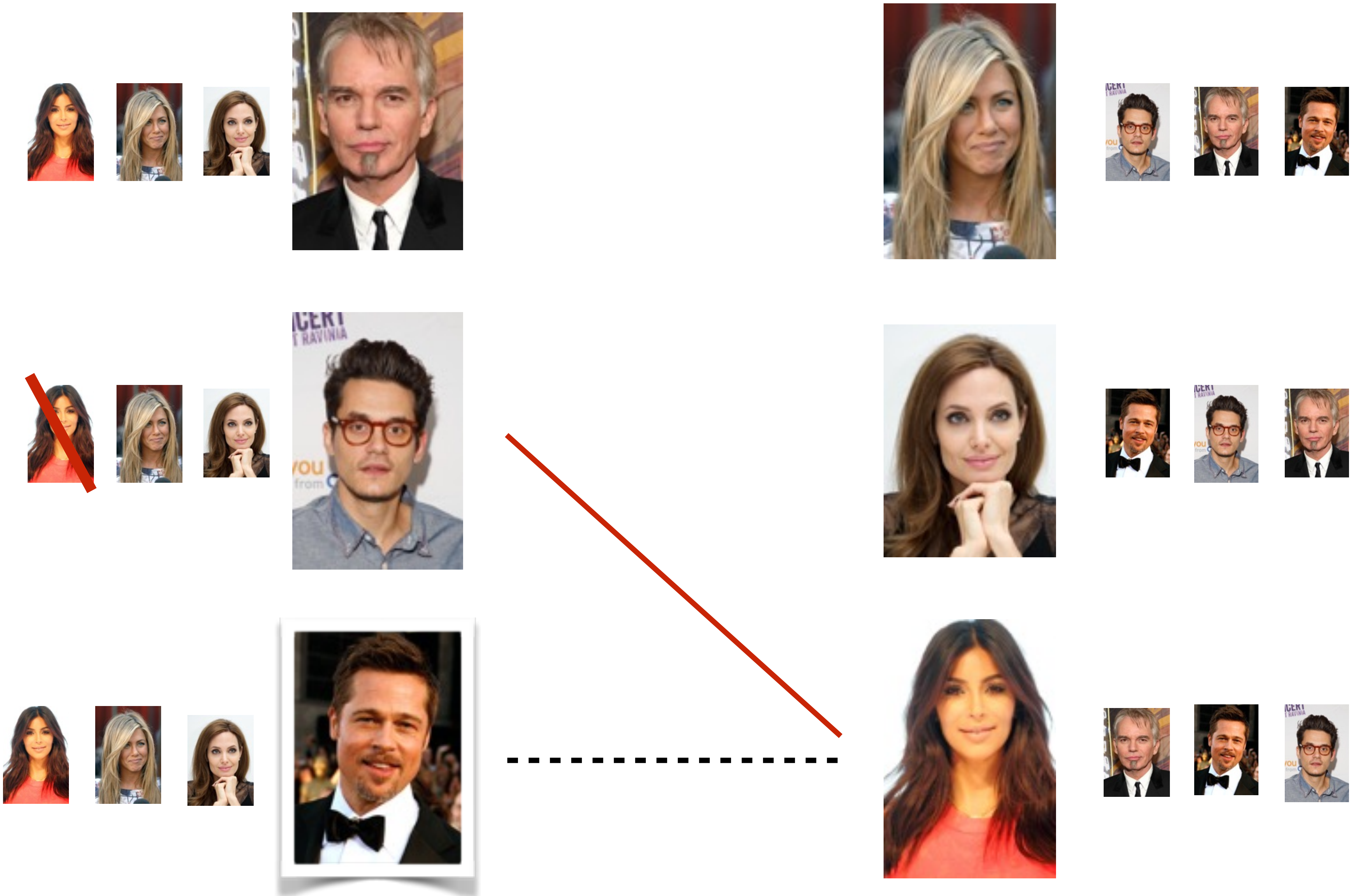
The Gale-Shapley Proposal Algorithm



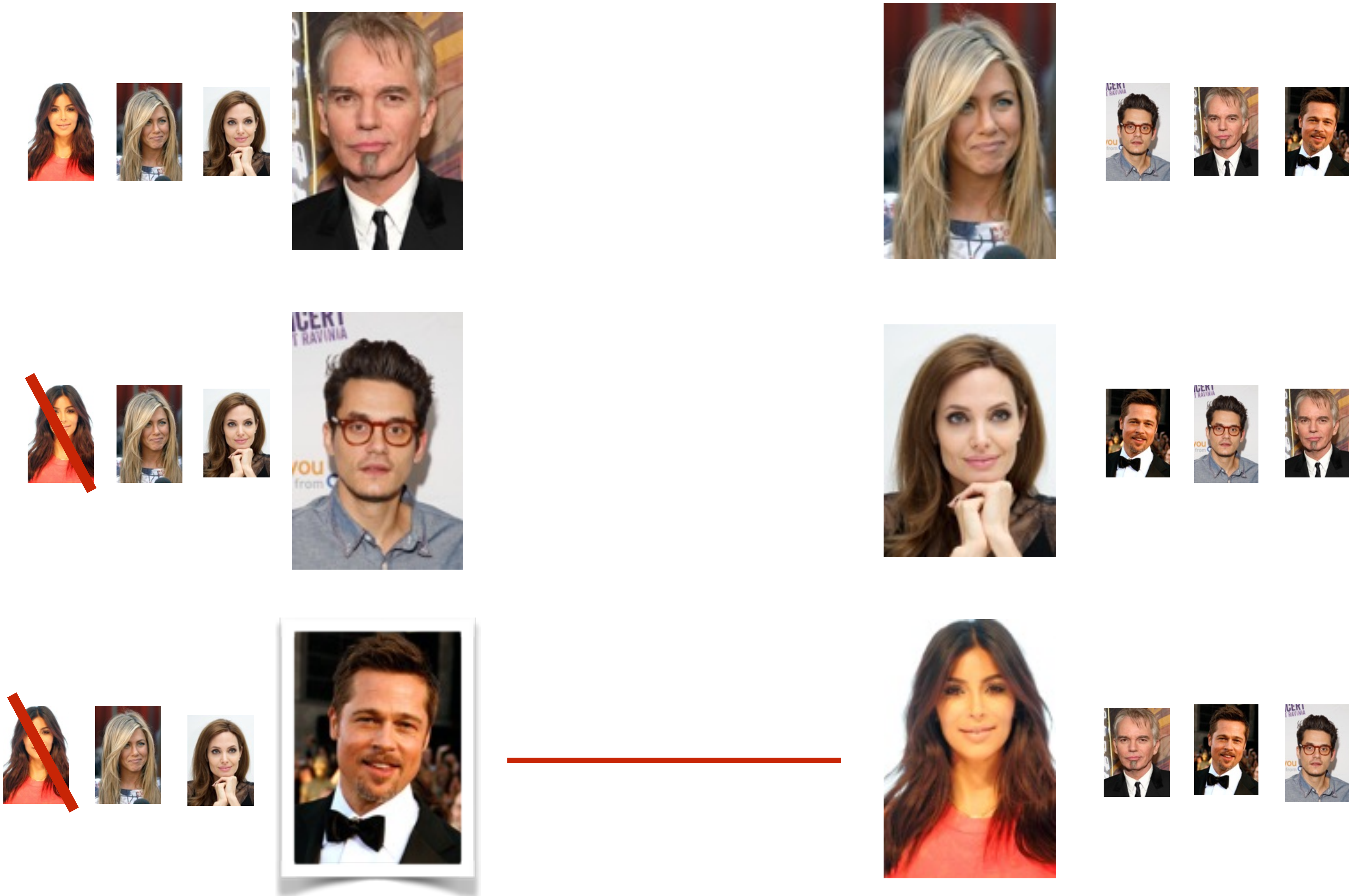
The Gale-Shapley Proposal Algorithm



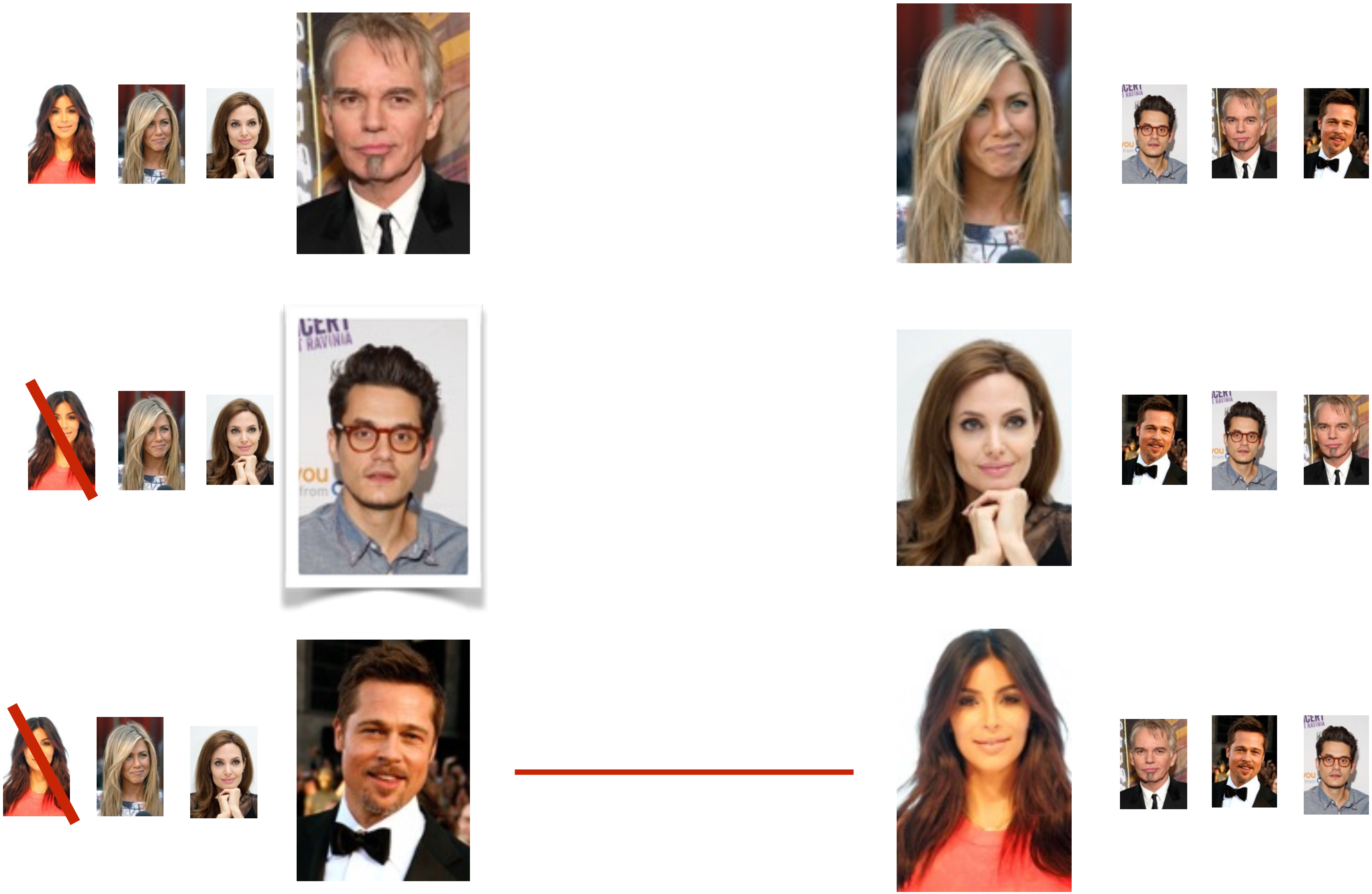
The Gale-Shapley Proposal Algorithm



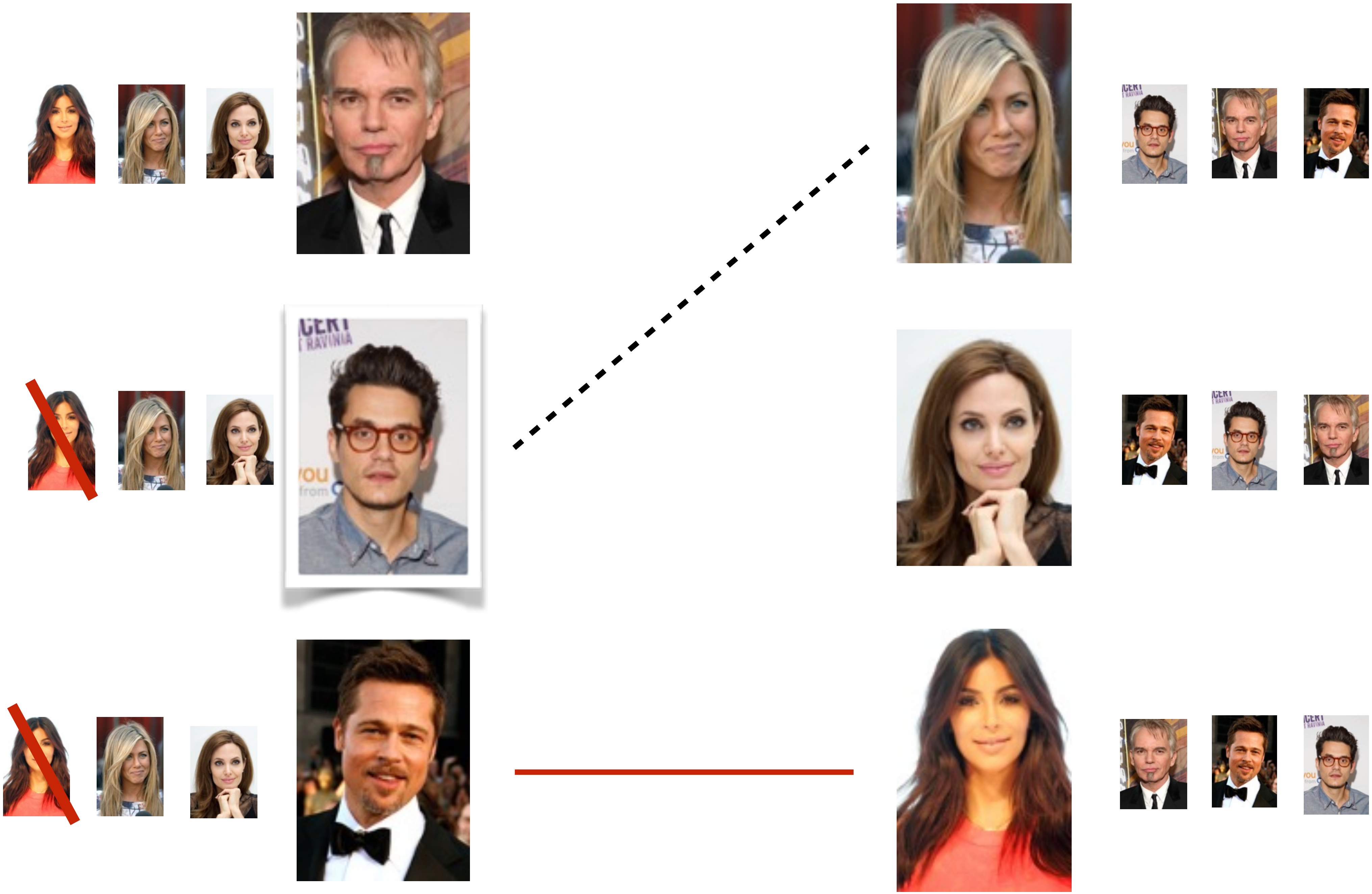
The Gale-Shapley Proposal Algorithm



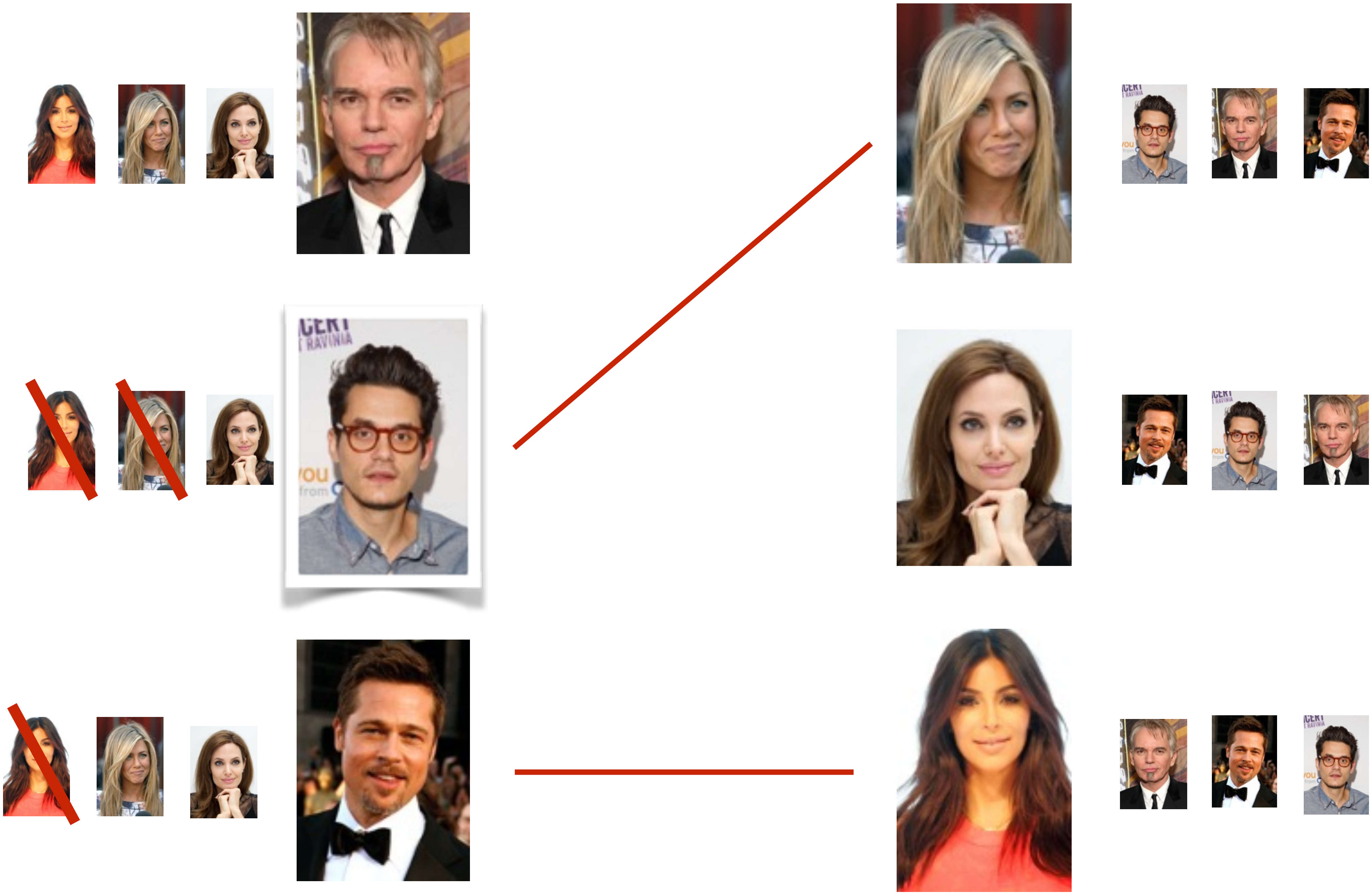
The Gale-Shapley Proposal Algorithm



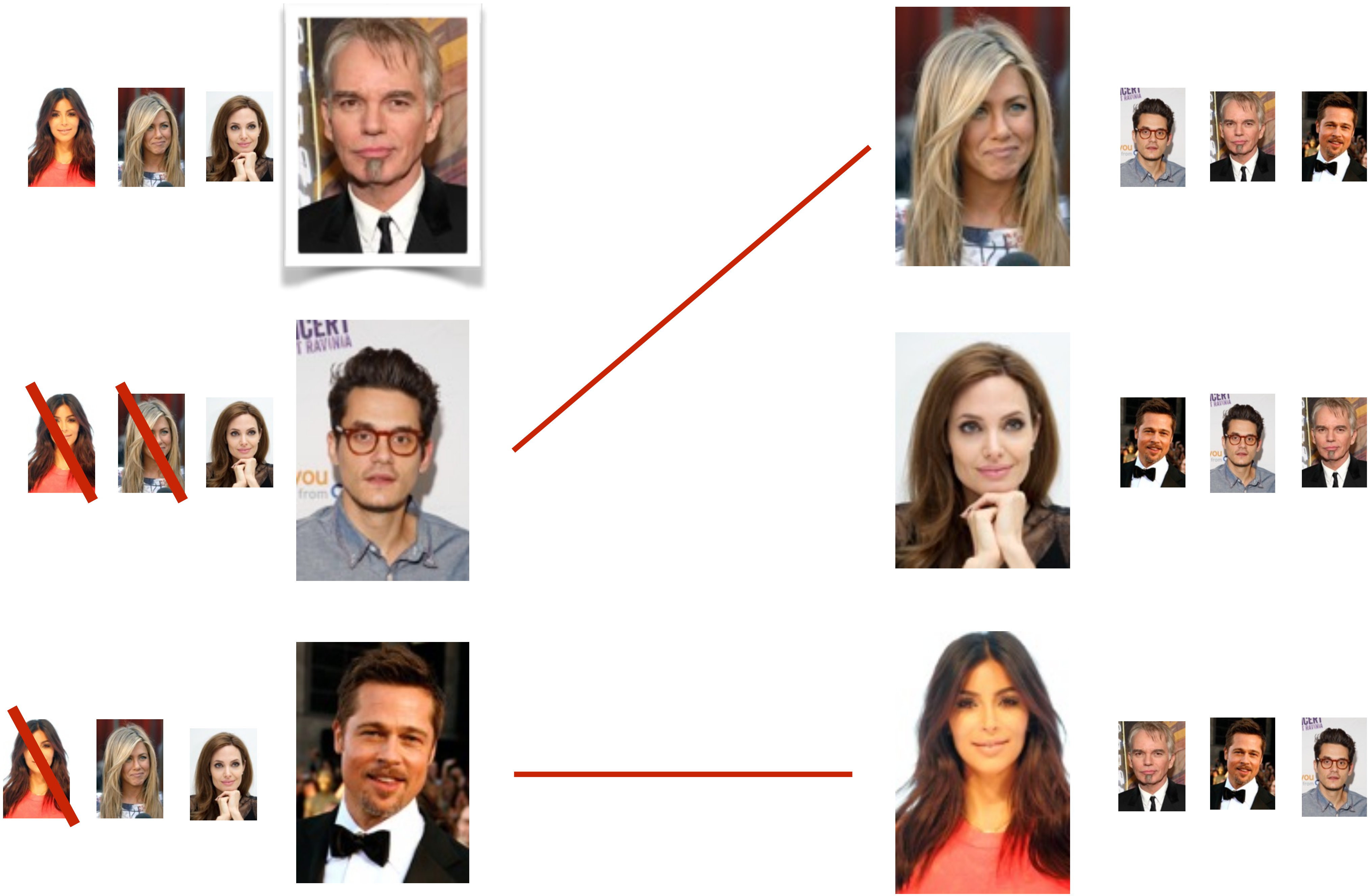
The Gale-Shapley Proposal Algorithm



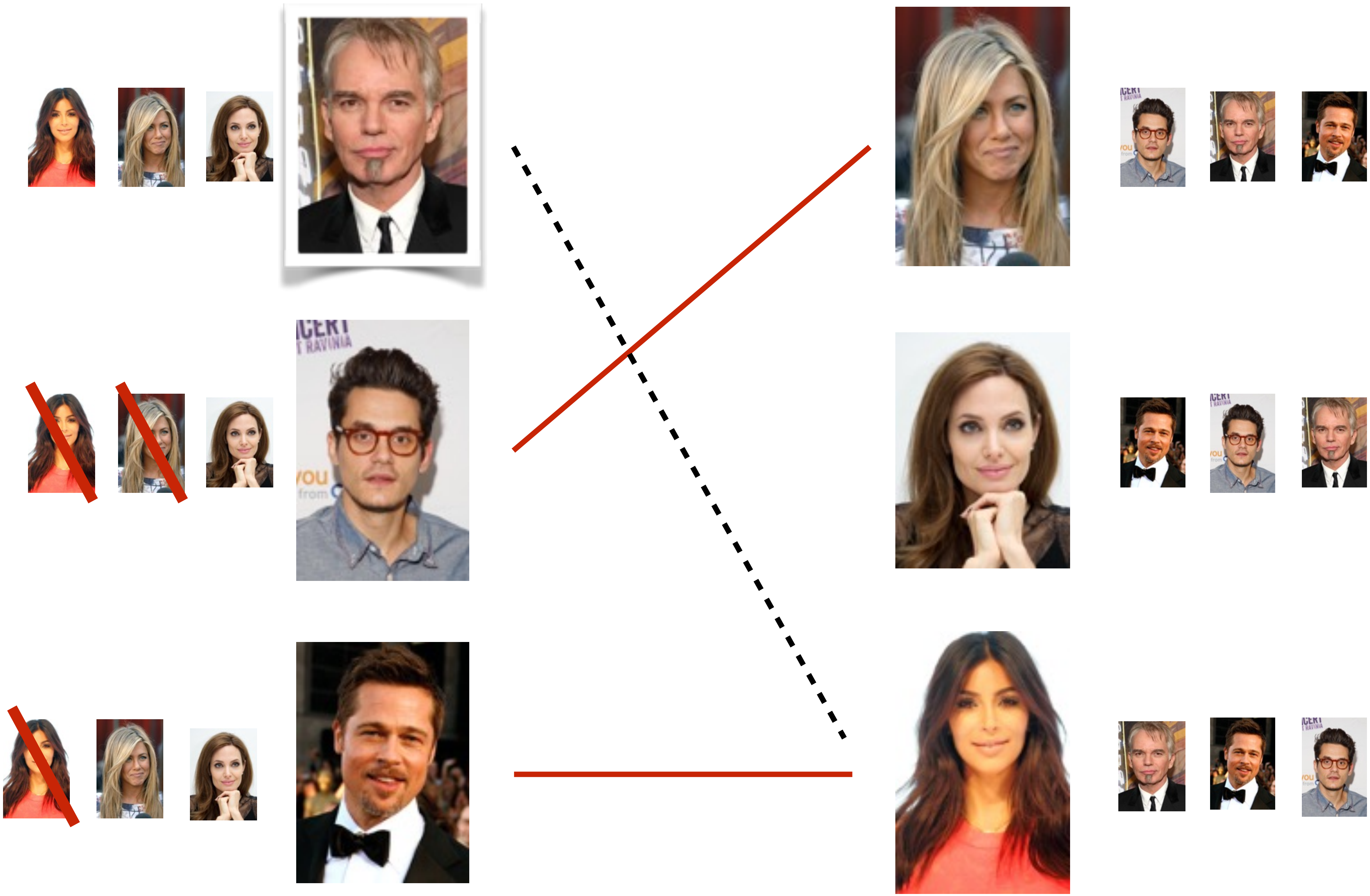
The Gale-Shapley Proposal Algorithm



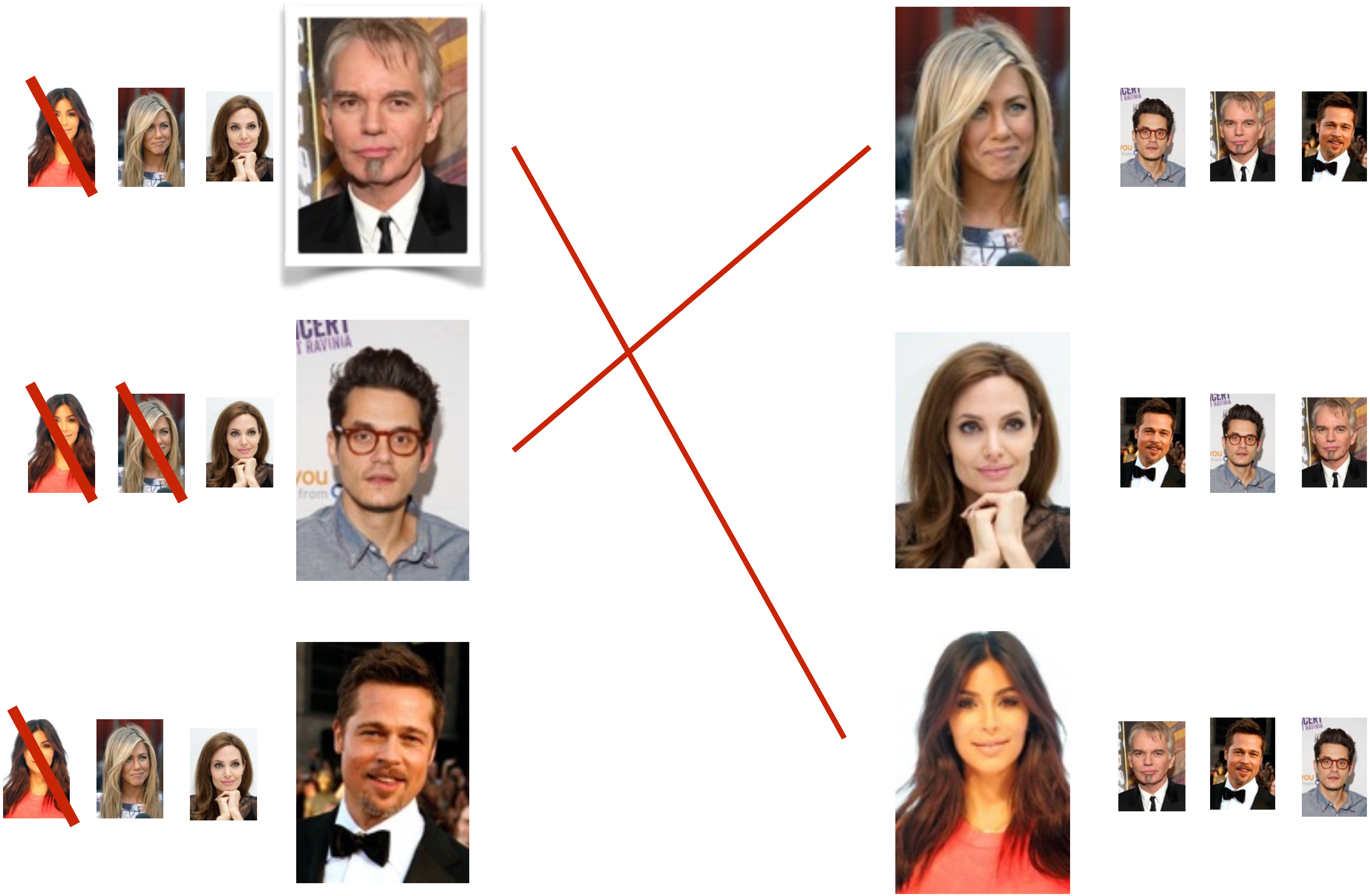
The Gale-Shapley Proposal Algorithm



The Gale-Shapley Proposal Algorithm

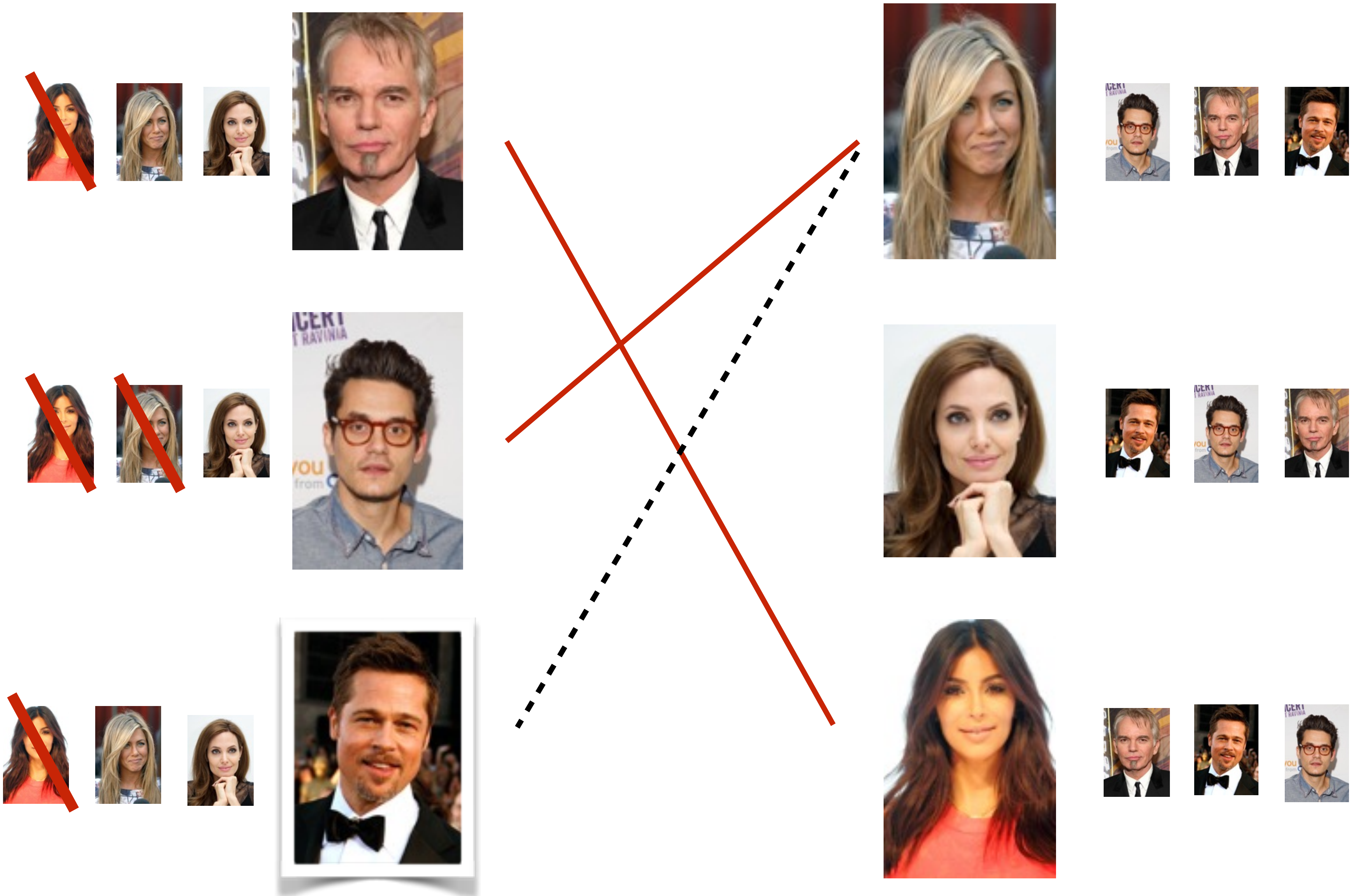


The Gale-Shapley Proposal Algorithm



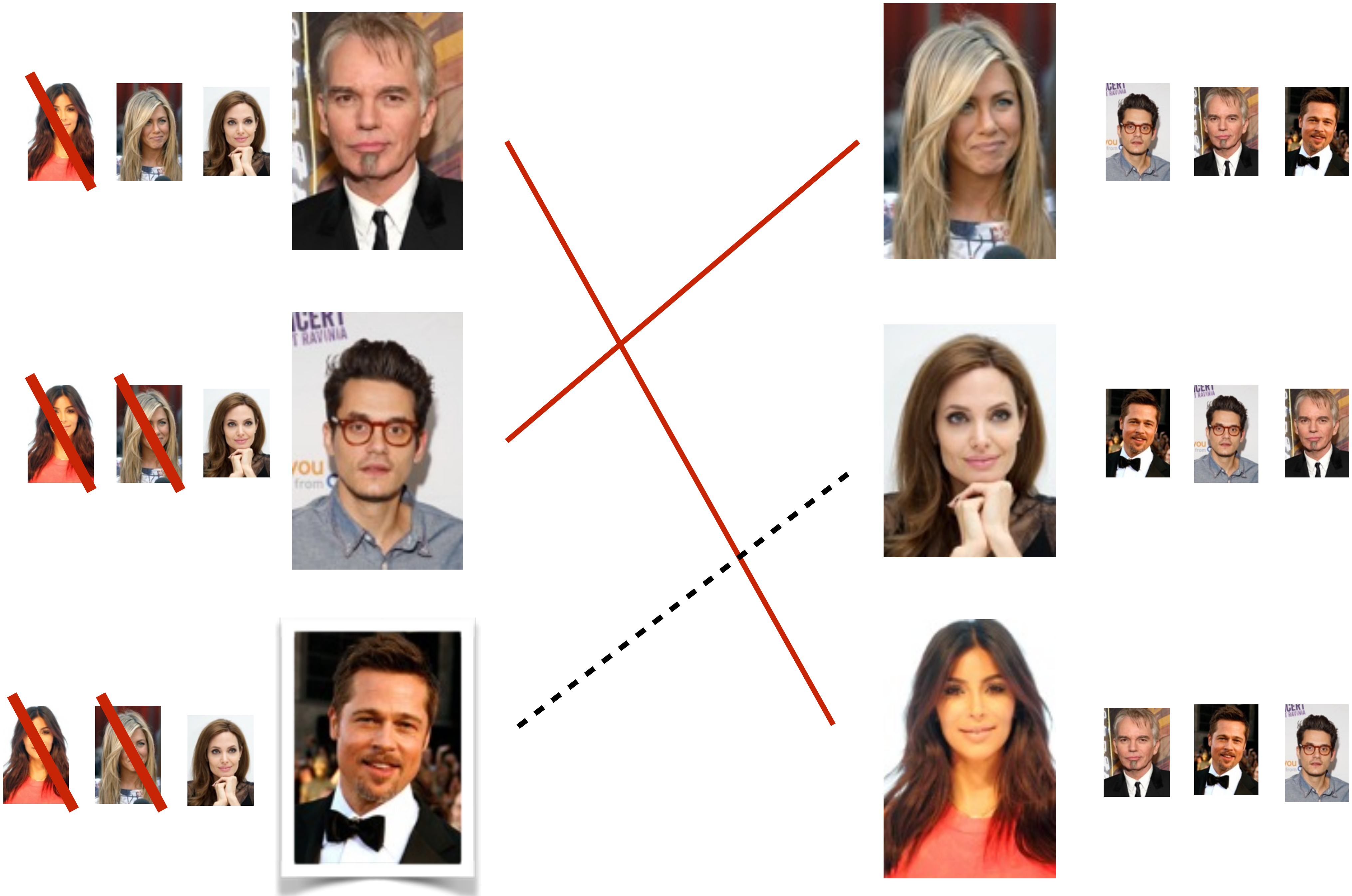
The Gale-Shapley Proposal Algorithm

The Gale-Shapley Proposal Algorithm



The Gale-Shapley Proposal Algorithm

The Gale-Shapley Proposal Algorithm



The Gale-Shapley Proposal Algorithm

The Gale-Shapley Proposal Algorithm

While there is a man who is not matched:

- Let m be such a man
- Let w be the highest ranked woman in m's list to whom m has not proposed yet.
- If w is unmatched, or w prefers m over her current partner:
  - Match m and w.
    (The previous match of w is now unmatched.)

Cool, but does it work correctly?

- Does it always terminate?
- Does it always find a stable matching?
  (Does a stable matching always exist?)

Step 4: Analyze the algorithm

Theorem:

The Gale-Shapley proposal algorithm always terminates with a stable matching after at most n^2 iterations.

A *constructive* proof that a stable matching always exists.

3 things to show:

1. Number of iterations is at most n^2.
2. The algorithm terminates with a perfect matching.
3. The matching has <u>no</u> unstable pairs.

Step 4: Analyze the algorithm

1. Number of iterations is at most n^2.

iterations = # proposals

No man proposes to a woman more than once.

So each man makes at most n proposals.

There are n men in total.

$$\Longrightarrow \quad \text{\# proposals} \leq n^2.$$

$$\Longrightarrow \quad \text{\# iterations} \leq n^2.$$

Step 4: Analyze the algorithm

2. The algorithm terminates with a perfect matching.

Suppose not.

This means some man is not matched to any woman.

i.e., the man got rejected by all the woman.

All the women are engaged
when the man proposes to them.
(A women prefers to be engaged than be single.)

If a woman is engaged, she stays engaged.

$\Longrightarrow$ All men are engaged.

Step 4: Analyze the algorithm

2. The algorithm terminates with a perfect matching.

A man is not engaged

$\implies$ All women must be engaged

$\implies$ All men are engaged.

Contradiction

Step 4: Analyze the algorithm

3. The matching has no unstable pairs.

Unstable pair: A pair of vertices m and w which are not matched to each other, BUT they prefer each other to their current partners.

Observations:

A man can only go down in his preference list.

A woman can only go up in her preference list.

Step 4: Analyze the algorithm

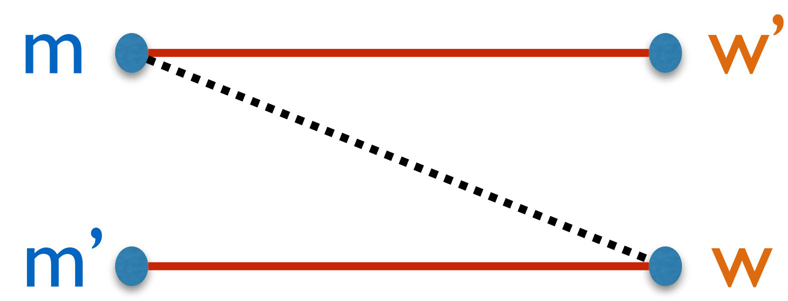
3. The matching has no unstable pairs.

Observations:

A man can only go down in his preference list.

A woman can only go up in her preference list.

Consider any pair (m, w)



Case 1: m never proposed to w

w' must be higher in the preference list of m than w

Case 2: m proposed to w

w rejected m $\implies$ w prefers her current partner $\square$

Further questions

Does the order of how we pick men matter?
Would it lead to different matchings?

Does this algorithm favor men or women or neither?

Answering further questions

w is a **valid partner** of m if there is some stable matching in which m and w are matched.

w is the **best valid partner** of m if w is the highest ranked valid partner of m.

Theorem:

The Gale-Shapley algorithm always matches m with its best valid partner.

Answering further questions

Theorem:

The Gale-Shapley algorithm always matches m with its best valid partner.

best(m) = best valid partner of m

Gale-Shapley returns {(m, best(m)) : m is in X }

Not at all obvious this would be a matching,
let alone a stable matching!

Proof of man optimality

Proof:

Suppose in the G-S algorithm, there is some man not matched to his best valid partner.

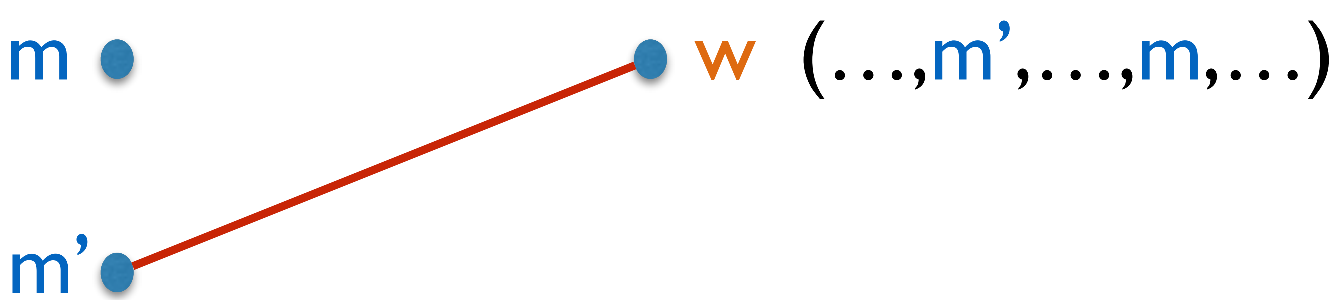
Some man got rejected by his best valid partner.

Some man got rejected by some valid partner.

Consider the first time this happens in the algorithm.
i.e., first time a man m gets rejected by a valid partner w.

During G-S algorithm:

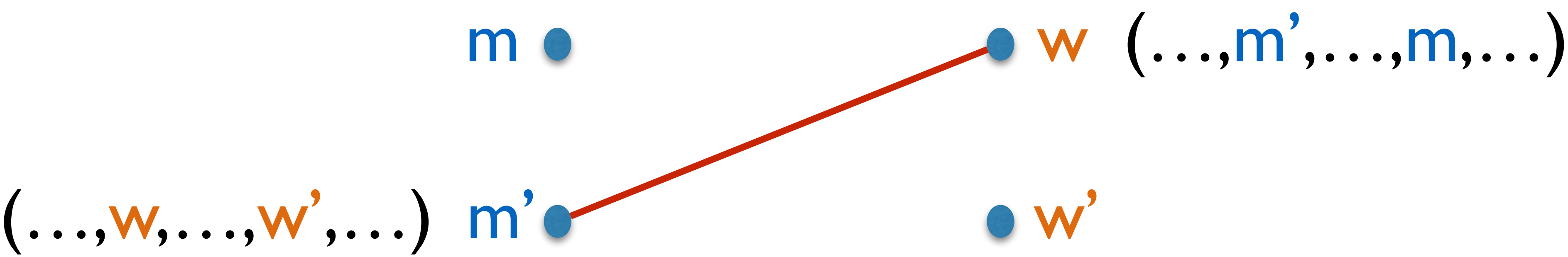
(right after m got rejected by w)



Proof of man optimality

Proof:

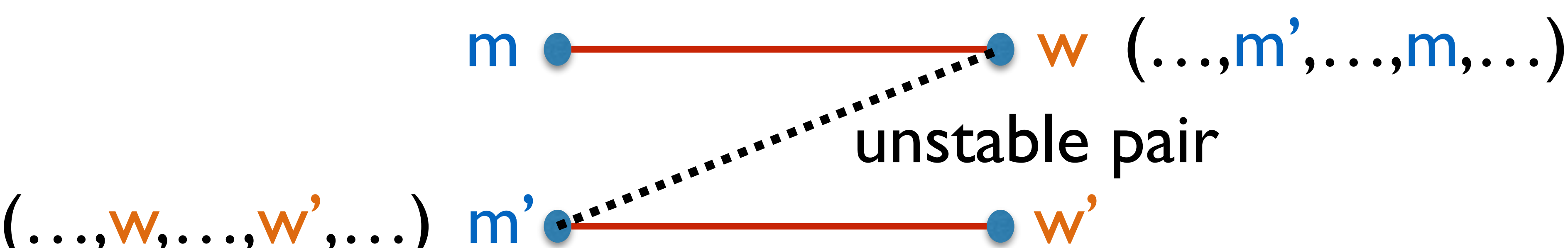
During G-S algorithm:



At this point, m’ could not have been rejected by w’.
(because m is the first men to be rejected by a valid partner.)

Some other stable matching:

(where m and w are matched)



□

Answering further questions

m is a **valid partner** of w if there is some stable matching in which m and w are matched.

m is the **best valid partner** of w if m is the highest ranked valid partner of w.

m is the **worst valid partner** of w if m is the lowest ranked valid partner of w.

Theorem:

The Gale-Shapley algorithm always matches w with its worst valid partner.

Proof of woman pessimality

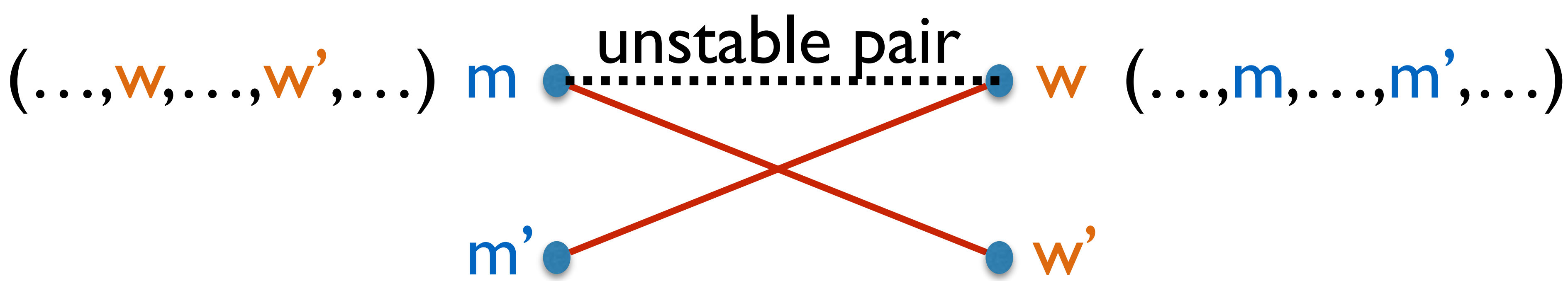
Proof:

Suppose some w is matched with m, but m is not the worst valid partner of w.

After G-S algorithm:

Some other stable matching:

(where w is matched with a worse partner m')



□

Stable matching variants

The original “finding internship” problem.

(the women can accept more than one proposal)

The original “finding internship” problem with “couples”.

(couples must be assigned together)

Stable roommate problem.

(all participants in one pool)

…

Maximum Matching

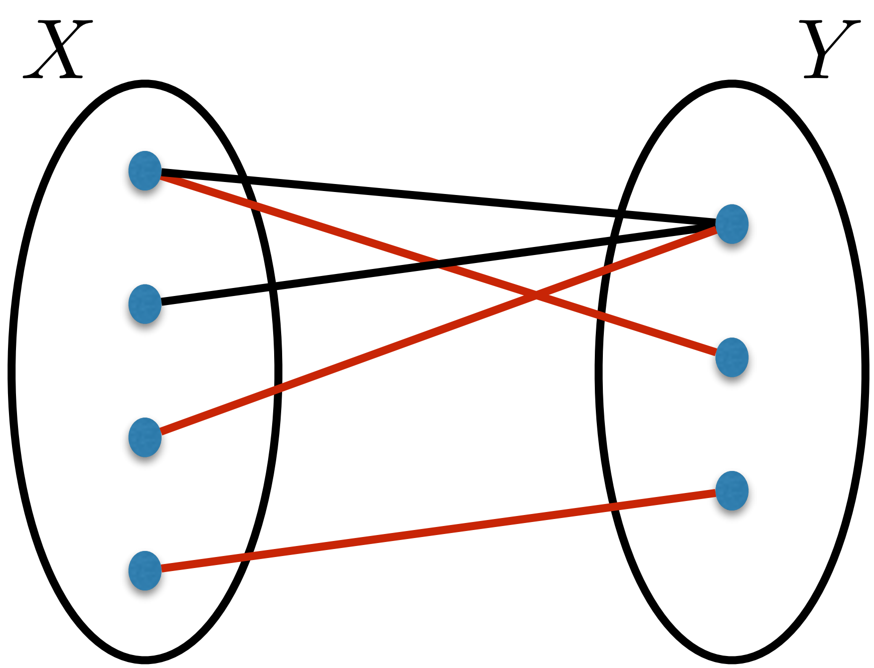
Matching problems

Matching :

A subset of the edges that do not share an endpoint.

Maximum matching:

A matching with largest number of edges (among all possible matchings).



e.g.: machines jobs

Matching problems

Can also define a matching in non-bipartite graphs.

It is just a subset of edges that don't share an endpoint.

Input: A graph $G = (V, E)$.

Output: A maximum matching in G.

Input: A graph $G = (V, E)$.

Output: Yes if G contains perfect matching. No otherwise.

(touches every vertex)

The restriction where G is bipartite is already interesting!

Bipartite maximum matching problem

Input: A bipartite graph $G = (X, Y, E)$.

Output: A maximum matching in G.

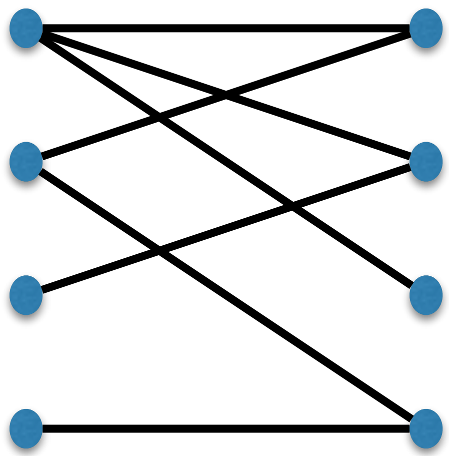
Is there an algorithm to solve this problem?

Try all possible subsets of the edges.
Check if it is a matching.
Keep track of the maximum one found.

Can we do better?

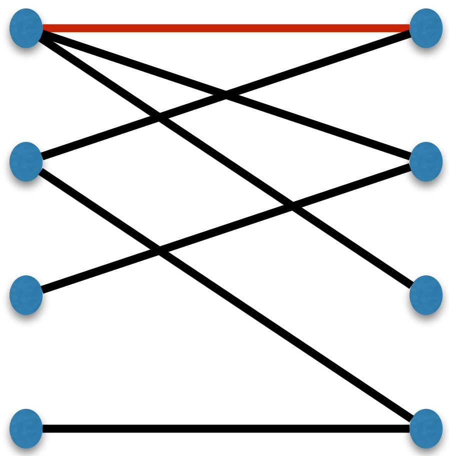
Bipartite maximum matching problem

What if we picked edges greedily?



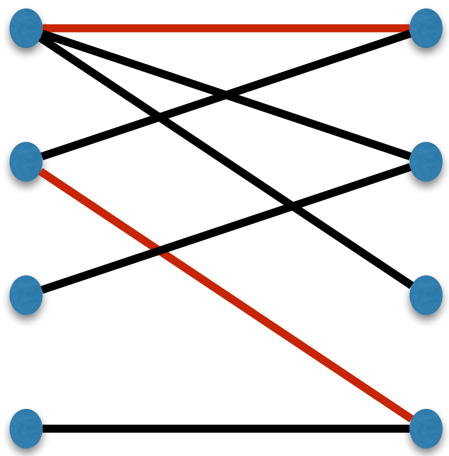
Bipartite maximum matching problem

What if we picked edges greedily?



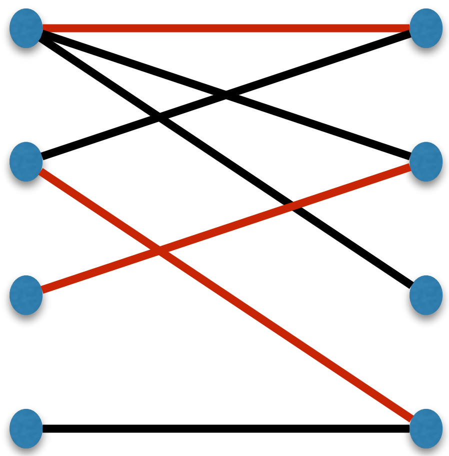
Bipartite maximum matching problem

What if we picked edges greedily?



Bipartite maximum matching problem

What if we picked edges greedily?



maximal matching

but not maximum

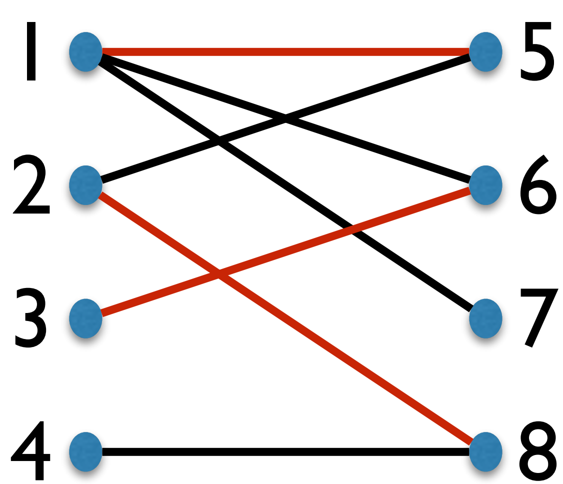
Is there a way to get out of this local optimum?

Augmenting paths

Let M be some matching.

An **augmenting path** with respect to M is a path in G such that:

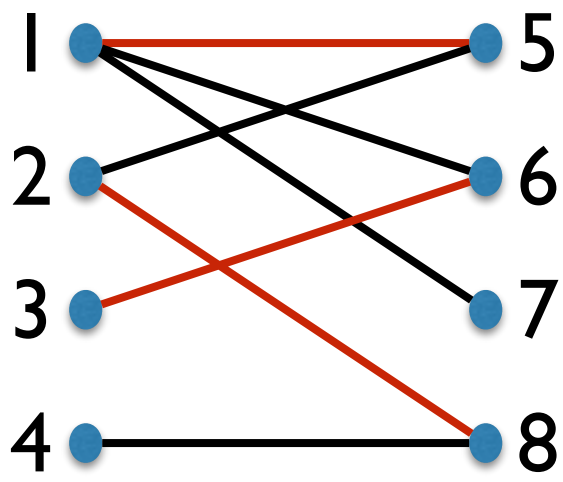
- the edges in the path alternate between being in M and not being in M
- the first and last vertices are not matched by M



Augmenting path:

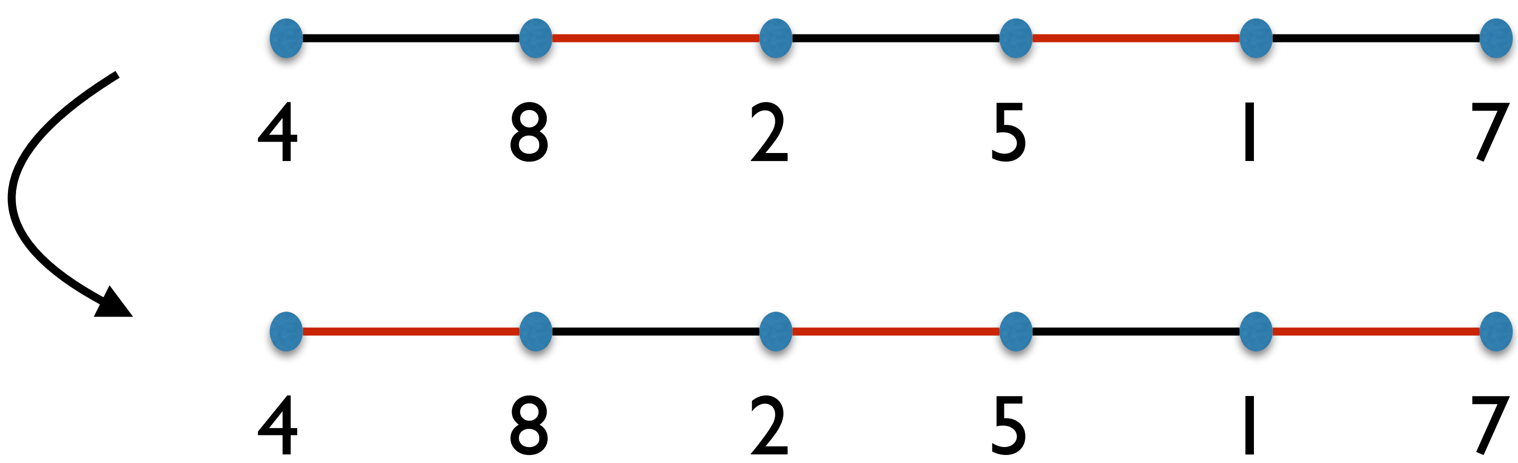
4-8-2-5-1-7

Augmenting paths



Augmenting path:

4-8-2-5-1-7



augmenting path $\Longrightarrow$ can obtain a bigger matching.

Augmenting paths and maximum matchings

augmenting path $\implies$ can obtain a bigger matching.

In fact:

no augmenting path $\implies$ maximum matching.

Theorem:

A matching M is maximum if and only if
there is no augmenting path with respect to it.

Augmenting paths and maximum matchings

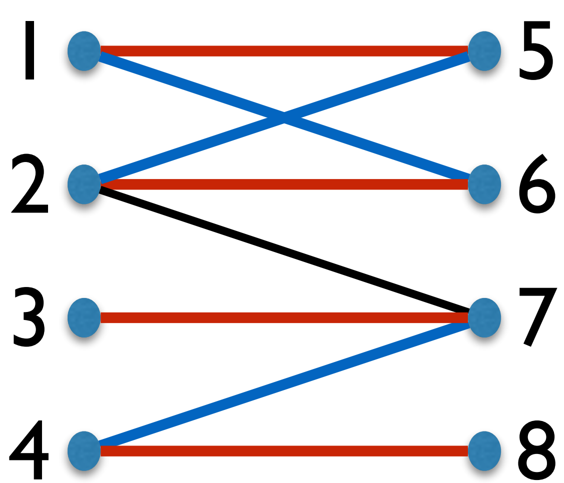
Proof:

If there is an augmenting path with respect to M, we saw that M is not maximum.

Want to show:

If M is not maximum, then there is an augmenting path.

Let M* be the maximum matching. $|M^*| > |M|$

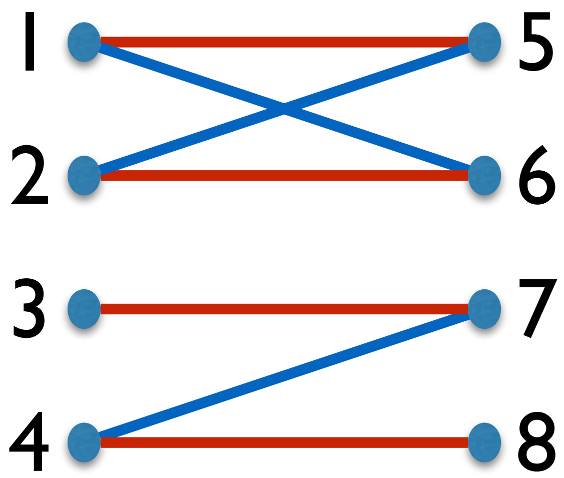


Let S be the set of edges contained in M* or M but not both.

$S = (M^* \cup M) - (M \cap M^*)$

Augmenting paths and maximum matchings

Proof:



Let S be the set of edges contained in M^* or M but not both.

$$S = (M^* \cup M) - (M \cap M^*)$$

What does S look like?

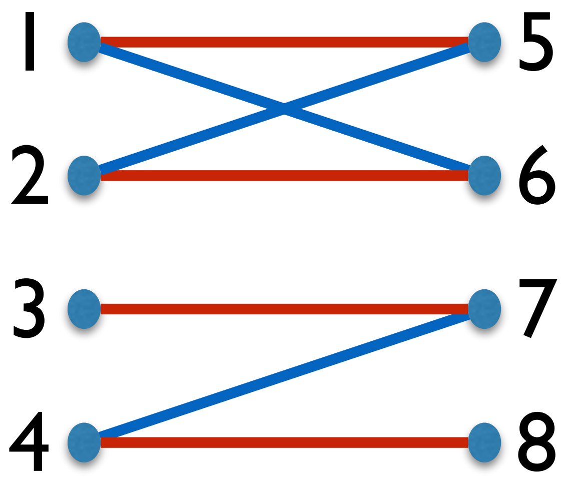
A vertex is incident to at most one edge in M^* and one edge in M.

(each vertex has degree at most 2)

So S is a collection of cycles and paths, and the edges alternate red and blue.

Augmenting paths and maximum matchings

Proof:



Let S be the set of edges contained in M* or M but not both.

$S = (M^* \cup M) - (M \cap M^*)$

So S is a collection of cycles and paths,
and the edges alternate red and blue.

In S, # red > # blue

Cycles must have even length. So # red = # blue in cycles.

Then there must be a path with # red > # blue.

This is an augmenting path with respect to M. □

Algorithm to find maximum matching

Theorem:

A matching M is maximum if and only if
there is no augmenting path with respect to it.

Algorithm:

Start with a single edge as your matching M.

Repeat until there is no augmenting path w.r.t. M:

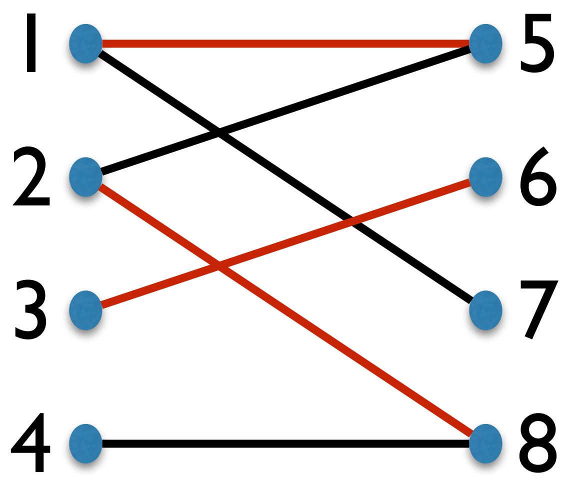
Find an augmenting path with respect to M.

Update M according to the augmenting path.

OK, but how do you find an augmenting path?

Algorithm to find maximum matching

OK, but how do you find an augmenting path?



If an edge is in M

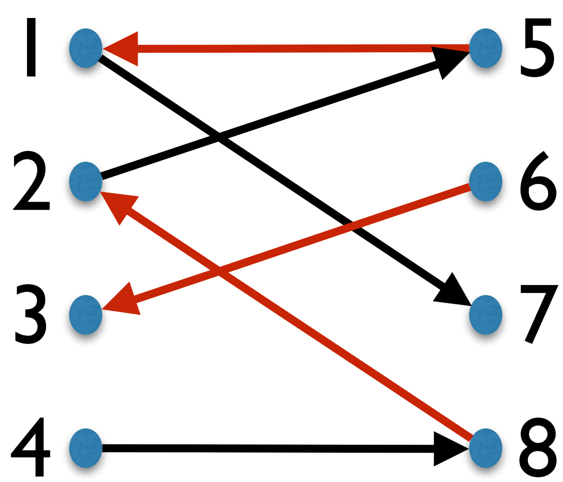
direct it from left to right.

If an edge is **not** in M

direct it from right to left.

Algorithm to find maximum matching

OK, but how do you find an augmenting path?



If an edge is in M

direct it from right to left.

If an edge is **not** in M

direct it from left to right.

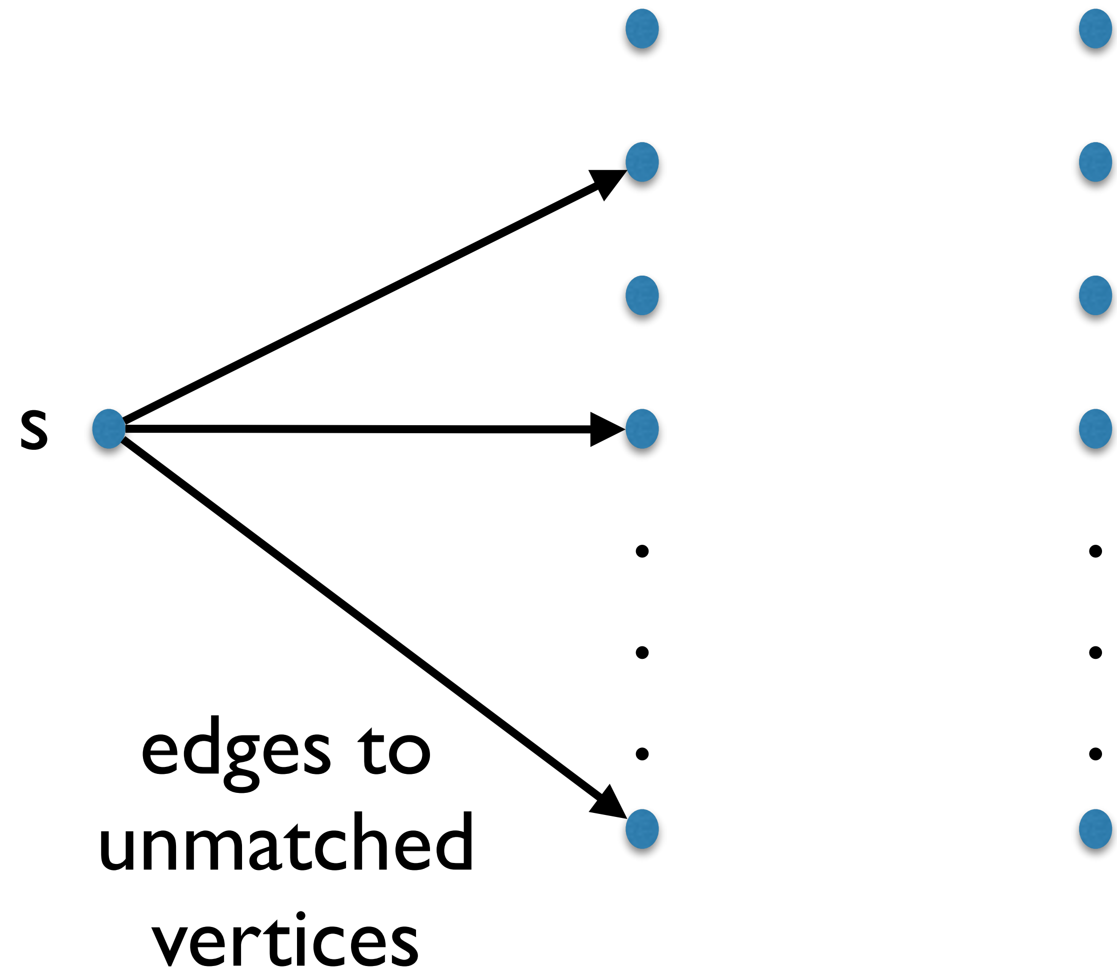
There is an augmenting path with respect to M

if and only if

There is a path from an unmatched vertex on the left to an unmatched vertex on the right.

Exercise

Algorithm to find maximum matching



Do a DFS starting at s.
Stop when you reach an unmatched vertex on the right.

Algorithm to find maximum matching

Algorithm:

Start with a single edge as your matching M.

Repeat until there is no augmenting path w.r.t. M:

Find an augmenting path with respect to M.

Update M according to the augmenting path.

$O(m \cdot n)$ time

Summary

Bipartite graphs and matchings arise very naturally in many areas.

They are extremely well-studied in math and CS.

- Gale Shapley Proposal algorithm

  Finds a stable matching.

- Augmenting path algorithm

  Finds a maximum matching.