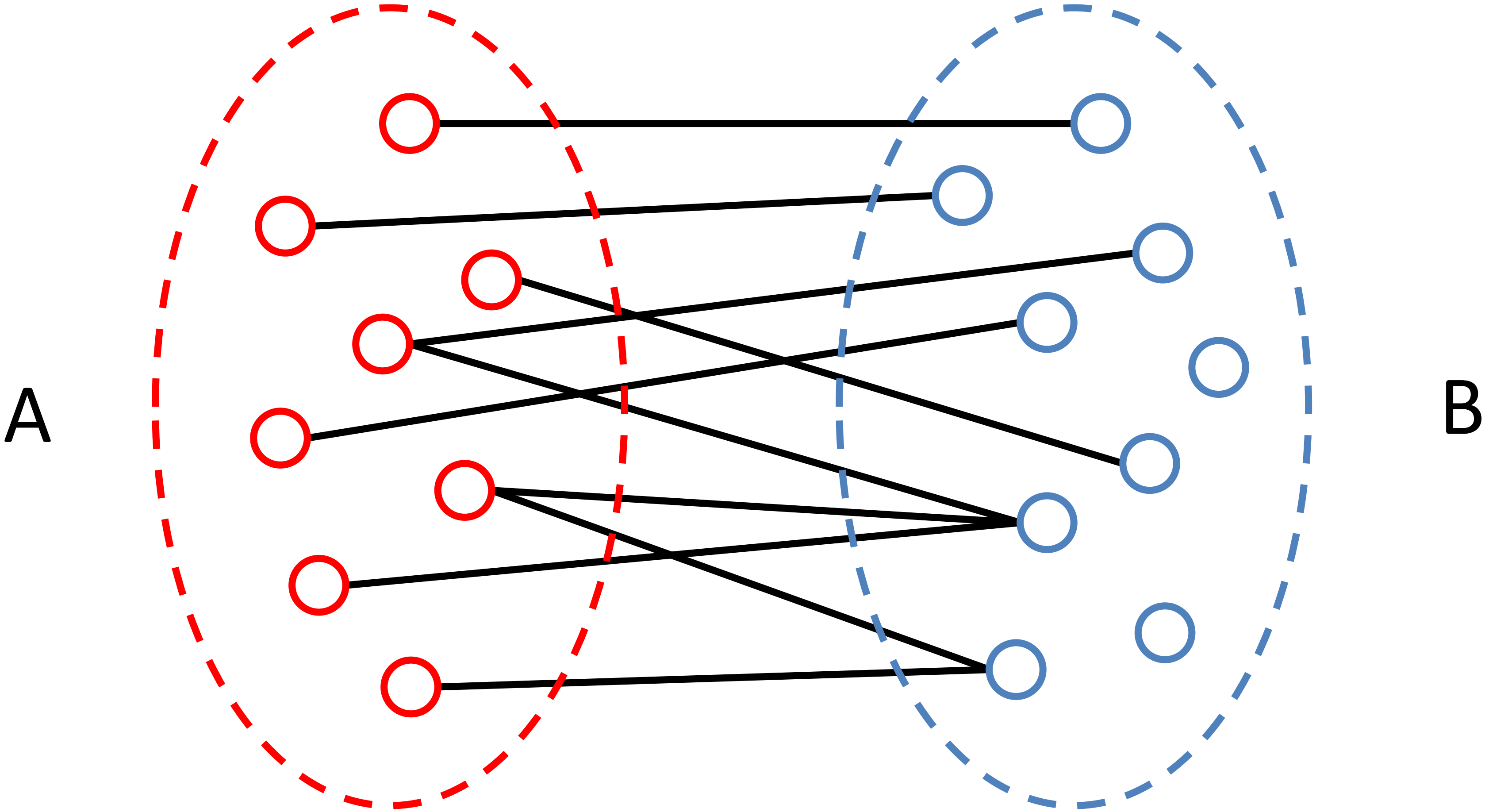
# COMP251: Bipartite graphs

Giulia Alberini & Jérôme Waldispühl
School of Computer Science
McGill University

Based on slides fom M. Langer (McGill) & P. Beame (UofW) & K. Wayne (Princeton)

# Bipartite graphs

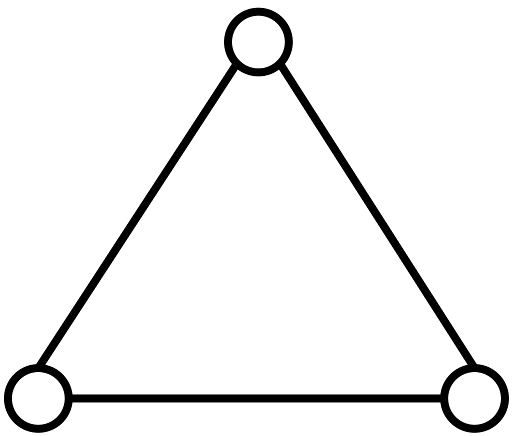



Vertices are partitioned into 2 sets.
All edges cross the sets.

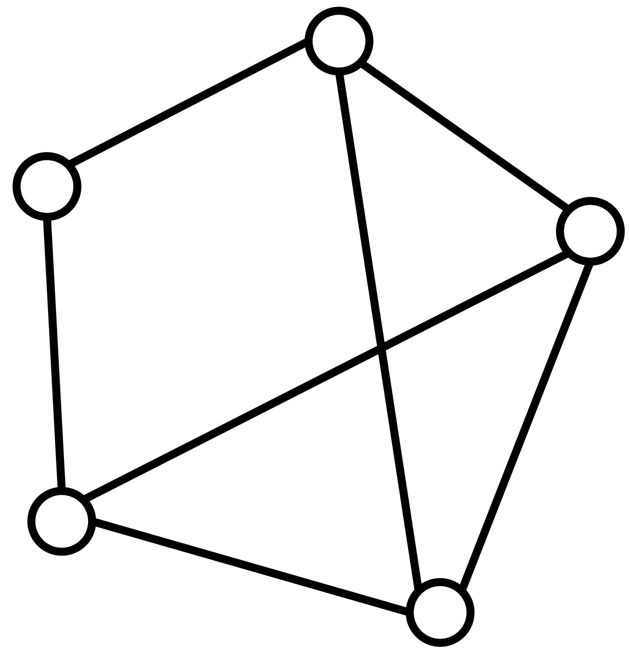
# Examples

| A | | B |
|---|---|---|
| Courses | registration | Students |
| Candidates | employment | Companies |
| People | Have read/seen | Books/Movies |

# Counter-examples



Easy to identify.



But not always...

# Cycles

**Core property of bipartite graph:**
No two vertices of the same set can be adjacent in the graph.

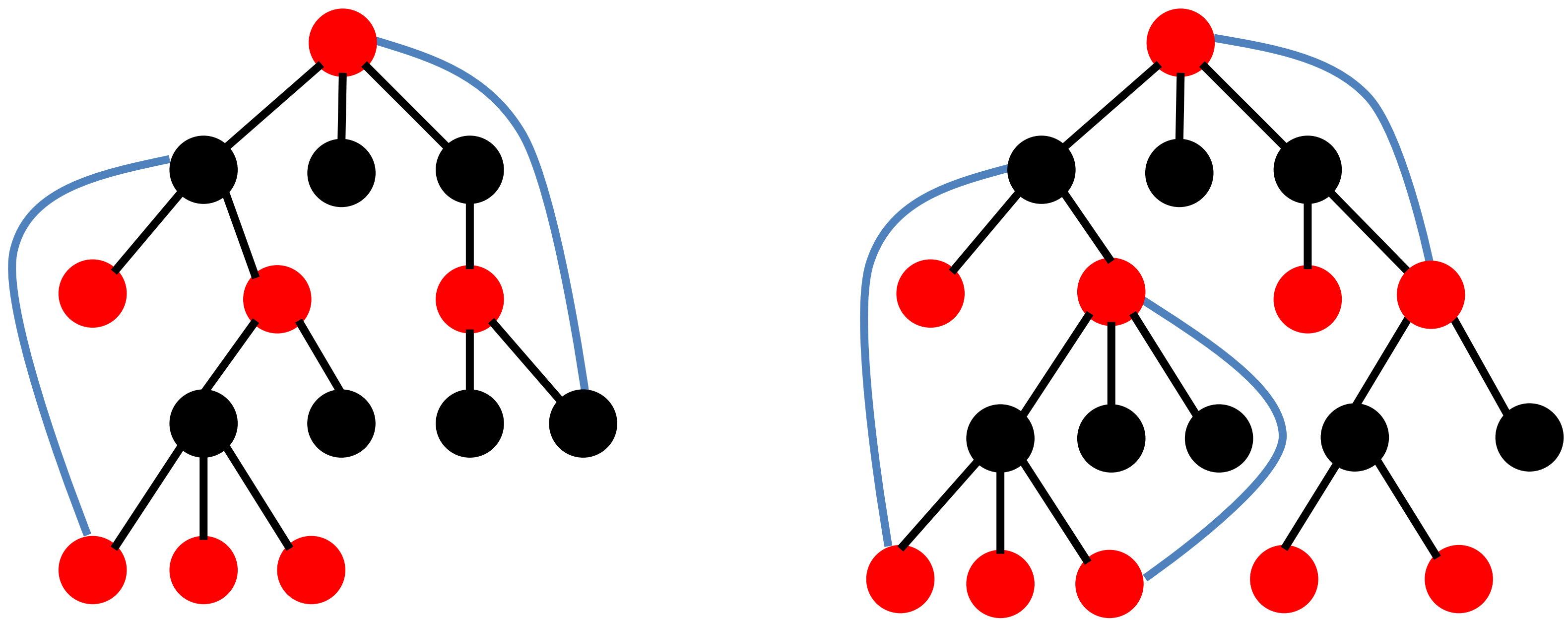
**Claim:** A graph is bipartite *if and only if* it does not contain an odd cycle.

**Proof:** Exercise.

# Is it a bipartite graph?

Assuming G=(V,E) is an undirected connected graph.

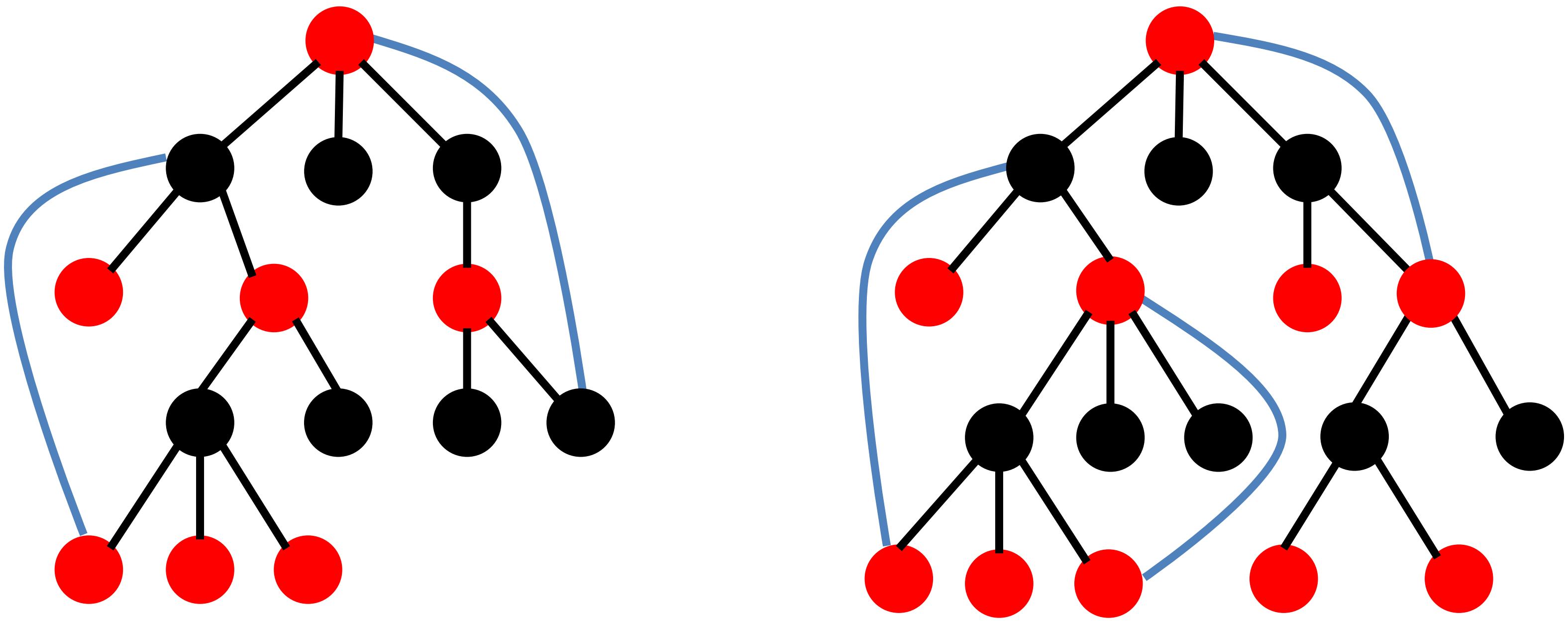
1. Run DFS and use it to build a DFS tree.
2. Color vertices by layers (e.g. red & black)
3. If all non-tree edges join vertices of different color, then the graph is bipartite. (guarantees only even cycles)



Non-tree edges in DFS tree cross 2 or more levels. Why?

# Is it a bipartite graph?

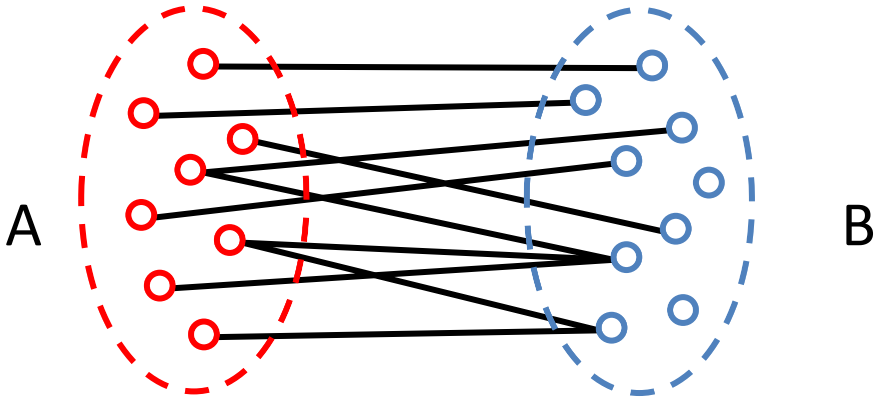
Non-tree edges in DFS tree cross 2 or more levels. Why?



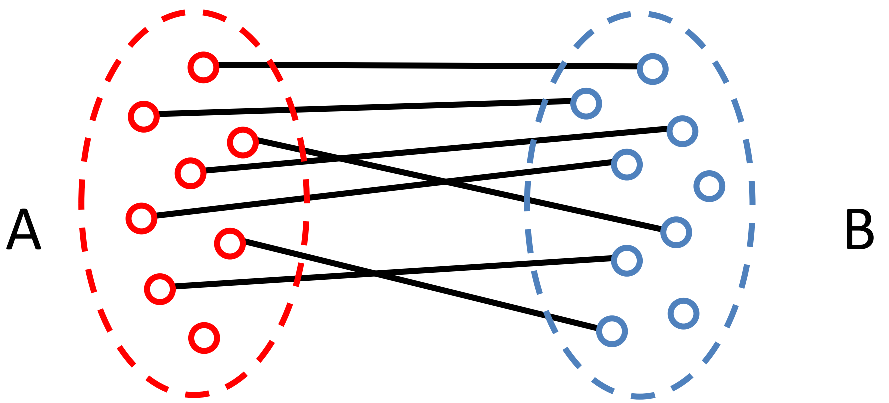
If there was a non-tree edge connecting a node with another on the same level or just one level above, then while discovering that node DFS would have not backtracked without exploring that edge (making it a tree edge)

# Bipartite matching

Consider an undirected bipartite graph.

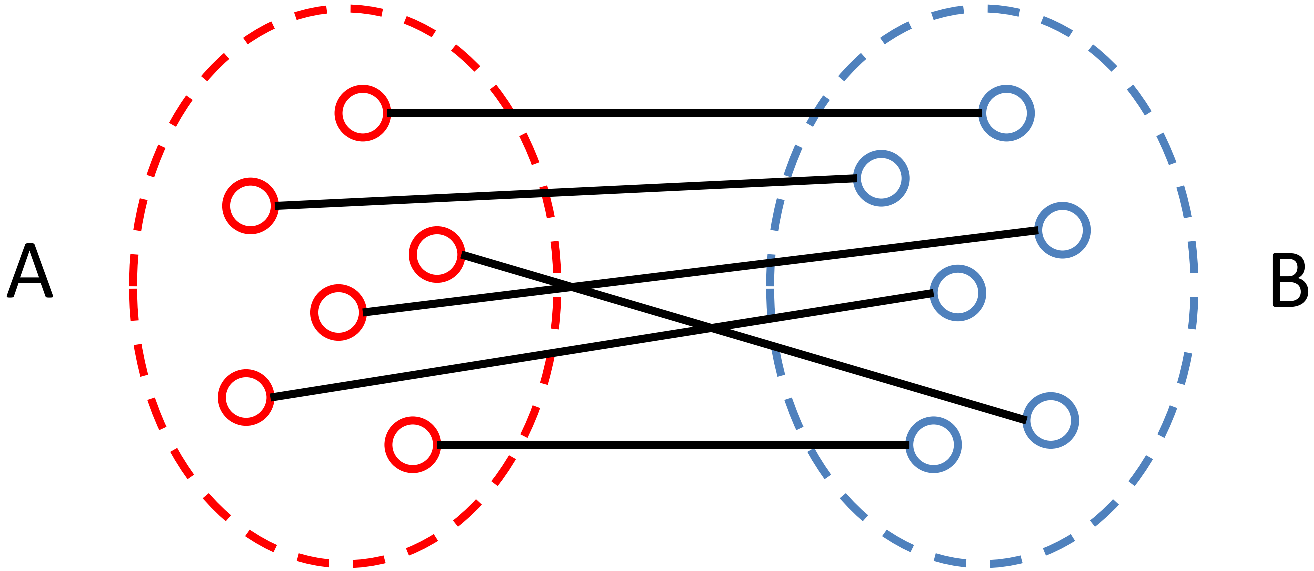



A matching is a subset of the edges $\{(\alpha, \beta)\}$ such that no two edges share a vertex.




Note: some vertices may not have an edge

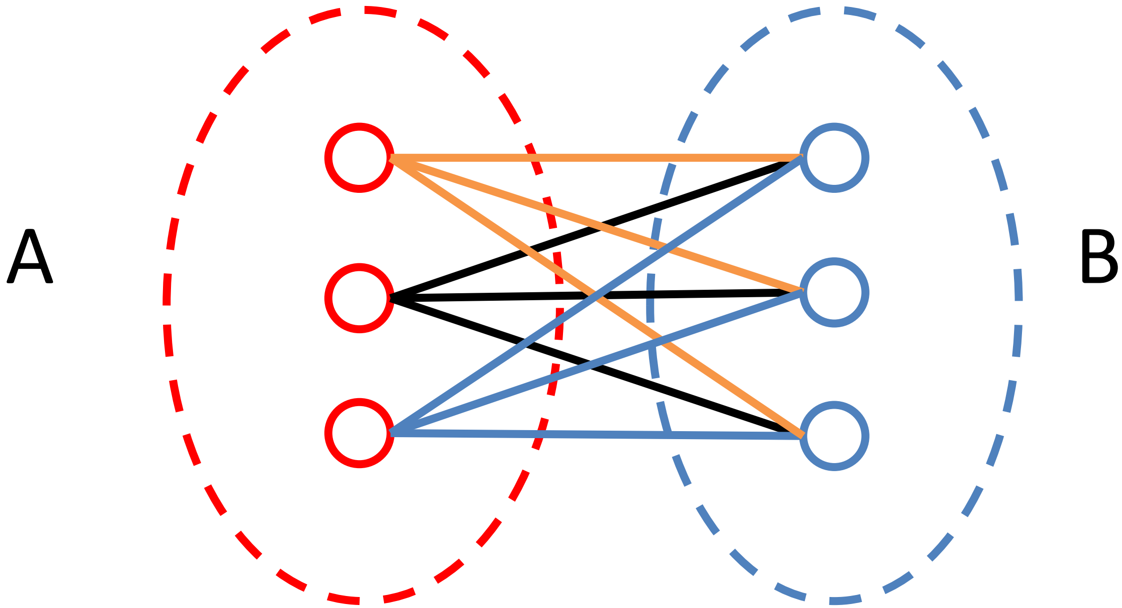
# Perfect matching




Suppose we have a bipartite graph with $n$ vertices in each A and B.
A **perfect matching** is a matching that has $n$ edges.

Note: It is not always possible to find a perfect matching.

# Complete bipartite graph




A complete bipartite graph is a bipartite graph that has an edge for every pair of vertices $(\alpha, \beta)$ such that $\alpha \in A, \beta \in B$.

# The algorithm of happiness

# Resident matching program

- **Goal:** Given a set of preferences among hospitals and medical school students, design a self-reinforcing admissions process.
- **Unstable pair:** applicant $x$ and hospital $y$ are unstable if:
  - $x$ prefers $y$ to their assigned hospital.
  - $y$ prefers $x$ to one of its admitted students.
- **Stable assignment:** Assignment with no unstable pairs.
  - Natural and desirable condition.
  - Individual self-interest will prevent any applicant/hospital deal from being made.

# Stable matching problem

**Goal:** Given $\boldsymbol{n}$ elements of $\boldsymbol{A}$ and $\boldsymbol{n}$ elements of $\boldsymbol{B}$, find a "suitable" matching. Participants rate members of opposite set:

- Each element of $A$ lists elements of $B$ in order of preference from best to worst.
- Each element of $B$ lists elements of $A$ in order of preference from best to worst.

**A**'s preferences

| | $1^{st}$ | $2^{nd}$ | $3^{rd}$ |
|---|---|---|---|
| Xavier | Alphabet | Baidu | Campbell |
| Yulia | Baidu | Alphabet | Campbell |
| Zoran | Alphabet | Baidu | Campbell |

**B**'s preferences

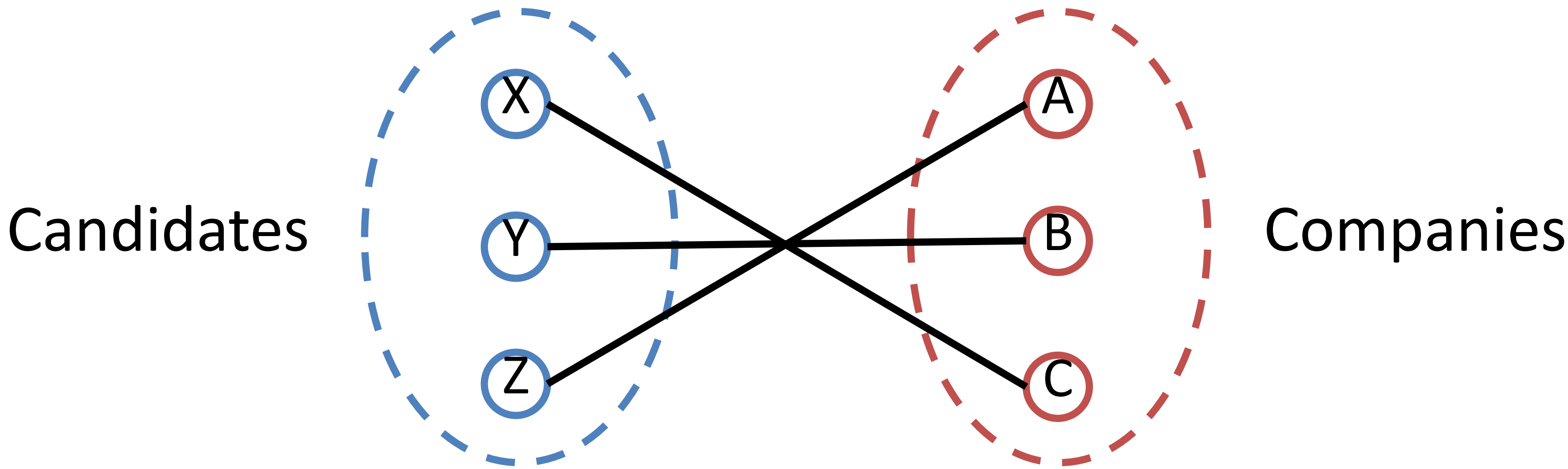
| | $1^{st}$ | $2^{nd}$ | $3^{rd}$ |
|---|---|---|---|
| Alphabet | Yulia | Xavier | Zoran |
| Baidu | Xavier | Yulia | Zoran |
| Campbell | Xavier | Yulia | Zoran |

# Stable matching problem

- **Context:** Candidates apply to companies.
- **Perfect matching:** everyone is matched with a single company.
  - Each candidate gets exactly one company.
  - Each company gets exactly one candidate.
- **Stability:** no incentive for some pair of participants to undermine assignment by joint action.
  - In matching **M**, an unmatched pair $\boldsymbol{\alpha}$-$\boldsymbol{\beta}$ is unstable if candidate $\boldsymbol{\alpha}$ and company $\boldsymbol{\beta}$ prefer each other to current match.
  - Unstable pair $\boldsymbol{\alpha}$-$\boldsymbol{\beta}$ could each improve by "escaping".
- **Stable matching:** perfect matching with no unstable pairs.
- **Stable matching problem:** Given the preference lists of **n** candidates and **n** companies, find a stable matching (if one exists).

# Example

Q: Is X-C, Y-B, Z-A a good assignment?




Candidates' preferences

| | 1st | 2nd | 3rd |
|---|---|---|---|
| Xavier | Alphabet | Baidu | Campbell |
| Yulia | Baidu | Alphabet | Campbell |
| Zoran | Alphabet | Baidu | Campbell |

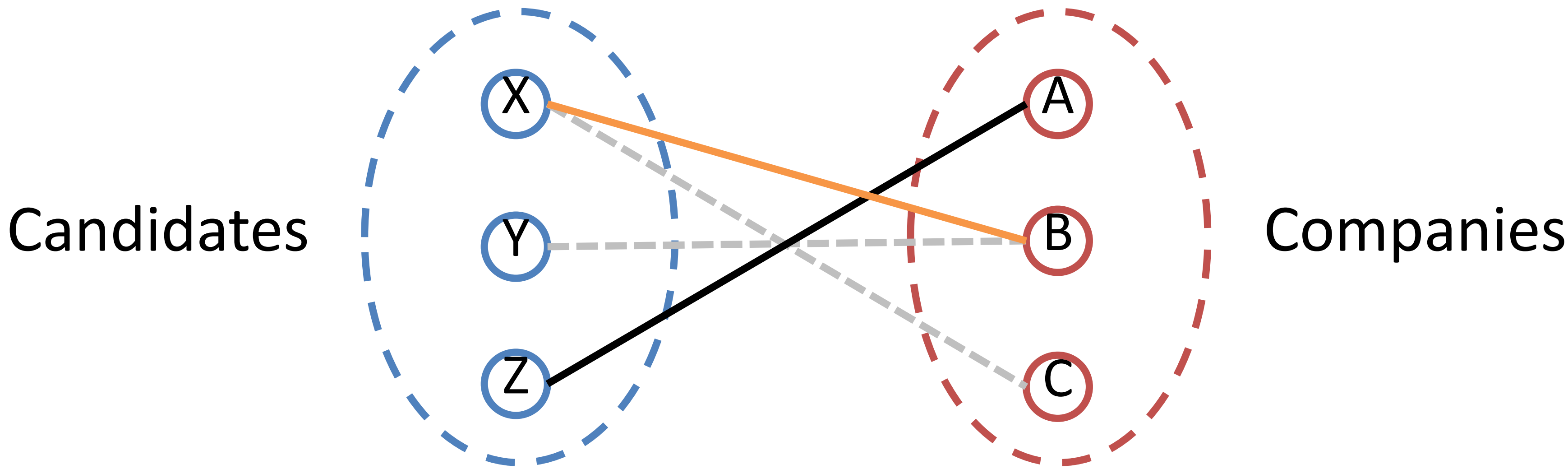
Companies' preferences

| | 1st | 2nd | 3rd |
|---|---|---|---|
| Alphabet | Yulia | Xavier | Zoran |
| Baidu | Xavier | Yulia | Zoran |
| Campbell | Xavier | Yulia | Zoran |

# Example

Q: Is X-C, Y-B, Z-A a good assignment?

A: No! Xavier and Baidu will hook up…




Candidates' preferences

| | 1st | 2nd | 3rd |
|---|---|---|---|
| Xavier | Alphabet | Baidu | Campbell |
| Yulia | Baidu | Alphabet | Campbell |
| Zoran | Alphabet | Baidu | Campbell |

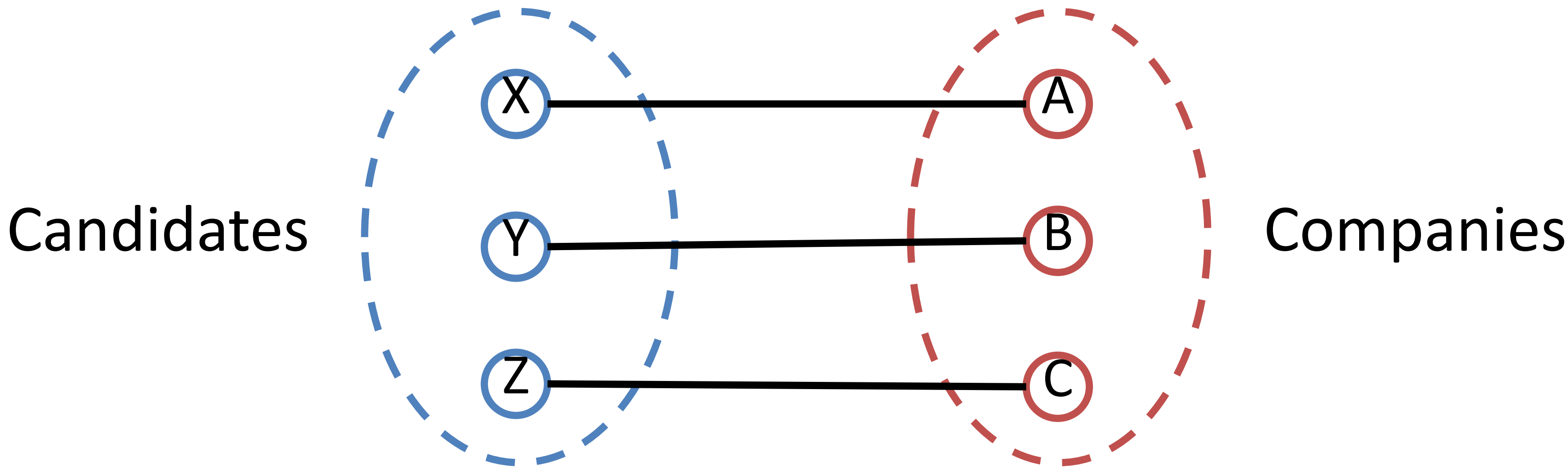
Companies' preferences

| | 1st | 2nd | 3rd |
|---|---|---|---|
| Alphabet | Yulia | Xavier | Zoran |
| Baidu | Xavier | Yulia | Zoran |
| Campbell | Xavier | Yulia | Zoran |

# Example

Q: Is X-A, Y-B, Z-C a good assignment?

A: Yes!




Candidates' preferences

| | 1st | 2nd | 3rd |
|---|---|---|---|
| Xavier | Alphabet | Baidu | Campbell |
| Yulia | Baidu | Alphabet | Campbell |
| Zoran | Alphabet | Baidu | Campbell |

Companies' preferences

| | 1st | 2nd | 3rd |
|---|---|---|---|
| Alphabet | Yulia | Xavier | Zoran |
| Baidu | Xavier | Yulia | Zoran |
| Campbell | Xavier | Yulia | Zoran |

# Stable matching problem

Consider a complete bipartite graph such that $|A| = |B| = n$.

- Each member of $A$ has a preference ordering of members of $B$.
- Each member of $B$ has a preference ordering of members of $A$.

<u>Algorithm for finding a matching.</u>

Until there's an unmatched member in $A$:

- Each $A$ member makes an offer to a $B$ member, in order of preference.
- Each $B$ member accepts the first offer from an $A$, but then rejects that offer if/when it receives an offer from an $A$ that it prefers more.

In our example: Candidates applies to companies. Companies accept the first offer they receive, but companies will drop their applicant when/if a preferred candidate applies after.

Note the asymmetry between A and B.

# Gale-Shapley algorithm

For each $\alpha \in A$, let `pref[α]` be the ordering of its preferences in $B$.
For each $\beta \in B$, let `pref[β]` be the ordering of its preferences in $A$.

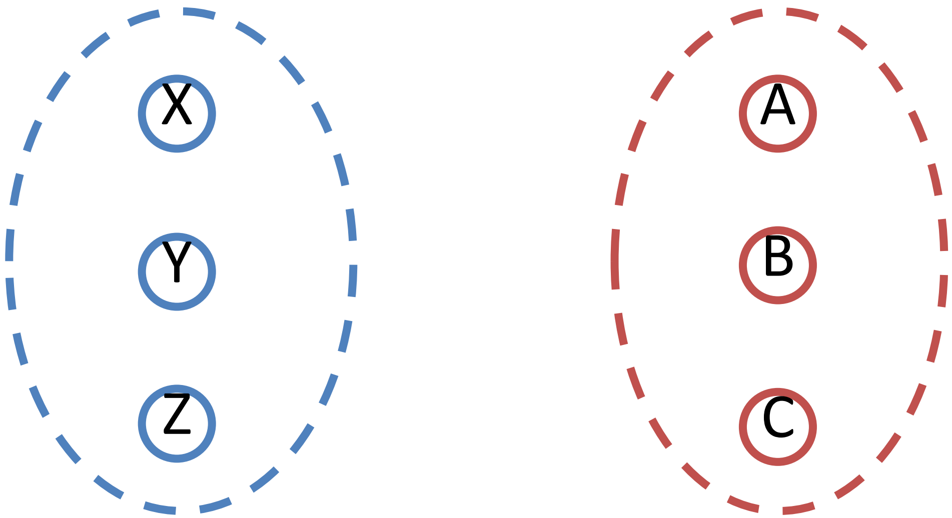
Let matching be a set of crossing edges between $A$ and $B$

`matching` $\leftarrow \emptyset$
**while** there is $\alpha \in A$ not yet matched **do**
  $\beta \leftarrow$ `pref[α].removeFirst()`  $\beta$ is $\alpha$'s first remaining choice
  **if** $\beta$ not yet matched **then**
    `matching` $\leftarrow$ `matching` $\cup \{(\alpha, \beta)\}$  If B has no match, accept
  **else**
    $\gamma \leftarrow \beta$'s current match
    **if** $\beta$ prefers $\alpha$ over $\gamma$ **then**
      `matching` $\leftarrow$ `matching` $- \{(\gamma, \beta)\} \cup \{(\alpha, \beta)\}$
**return** `matching`

If B has a match, check if they would prefer this new match, if yes, dump the old one

# Example

Candidates




Companies

Candidates' preferences

| | 1st | 2nd | 3rd |
|---|---|---|---|
| Xavier | Baidu | Alphabet | Campbell |
| Yulia | Baidu | Campbell | Alphabet |
| Zoran | Alphabet | Campbell | Baidu |

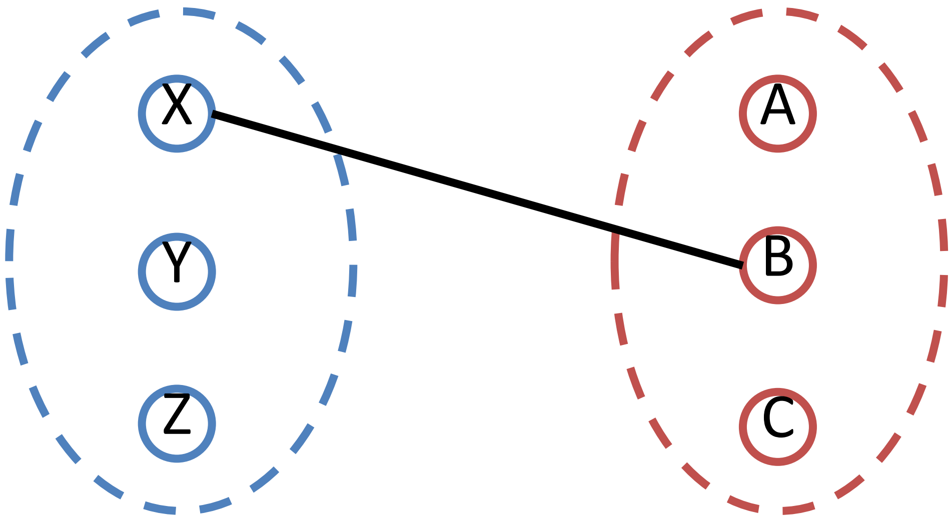
Companies' preferences

| | 1st | 2nd | 3rd |
|---|---|---|---|
| Alphabet | Zoran | Xavier | Yulia |
| Baidu | Yulia | Zoran | Xavier |
| Campbell | Xavier | Yulia | Zoran |

Note: In practice, we inverse the roles. Companies makes offers…

# Example

Candidates




Companies

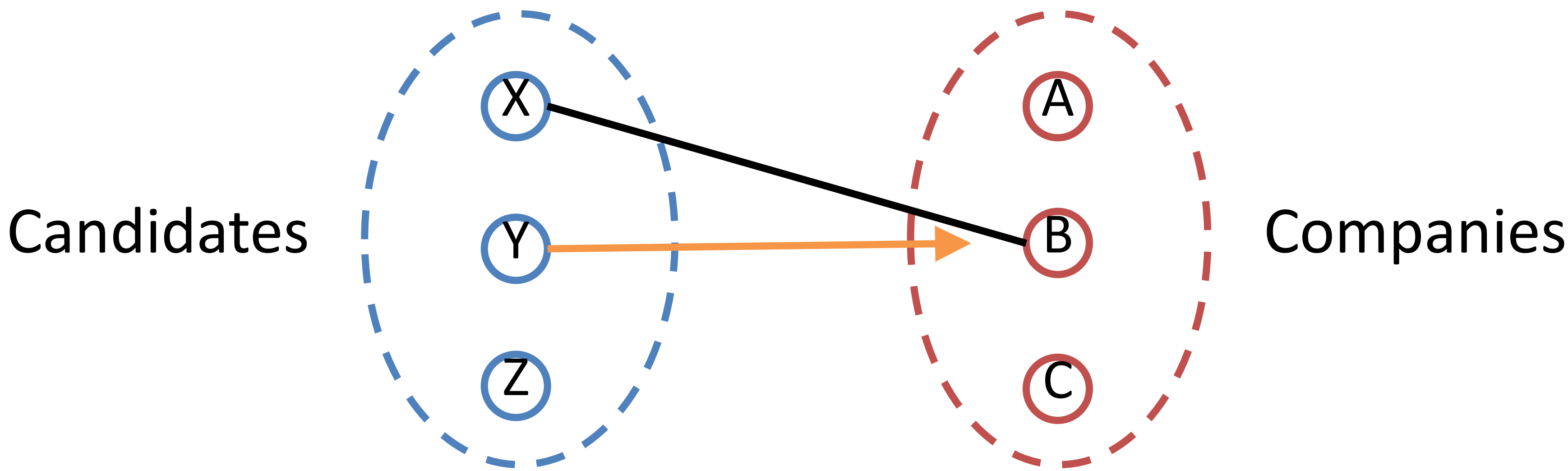
Candidates' preferences

| | 1st | 2nd | 3rd |
|---|---|---|---|
| Xavier | Baidu | Alphabet | Campbell |
| Yulia | Baidu | Campbell | Alphabet |
| Zoran | Alphabet | Campbell | Baidu |

Companies' preferences

| | 1st | 2nd | 3rd |
|---|---|---|---|
| Alphabet | Zoran | Xavier | Yulia |
| Baidu | Yulia | Zoran | Xavier |
| Campbell | Xavier | Yulia | Zoran |

# Example



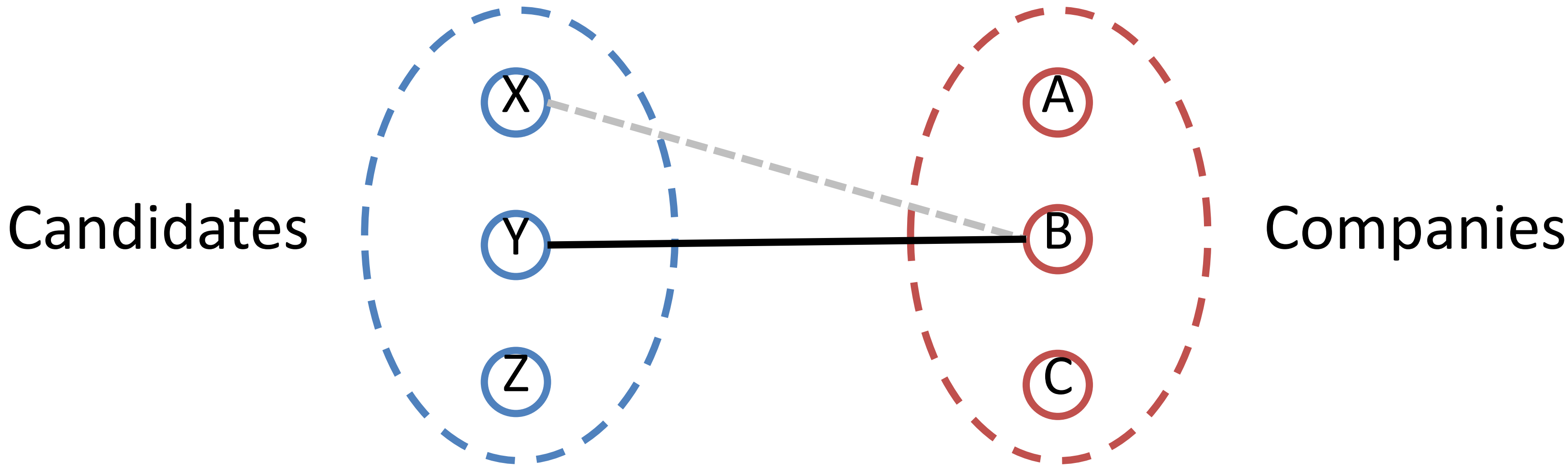

Candidates' preferences

| | 1st | 2nd | 3rd |
|---|---|---|---|
| Xavier | Baidu | Alphabet | Campbell |
| Yulia | Baidu | Campbell | Alphabet |
| Zoran | Alphabet | Campbell | Baidu |

Companies' preferences

| | 1st | 2nd | 3rd |
|---|---|---|---|
| Alphabet | Zoran | Xavier | Yulia |
| Baidu | Yulia | Zoran | Xavier |
| Campbell | Xavier | Yulia | Zoran |

# Example



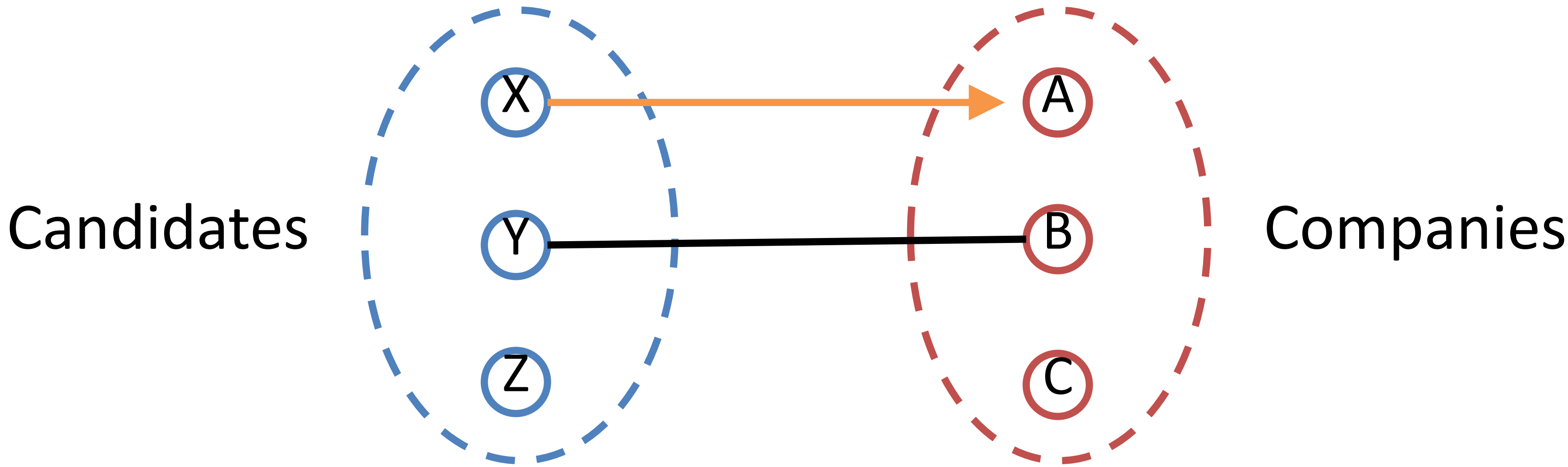

Candidates' preferences

| | 1st | 2nd | 3rd |
|---|---|---|---|
| Xavier | Baidu | Alphabet | Campbell |
| Yulia | Baidu | Campbell | Alphabet |
| Zoran | Alphabet | Campbell | Baidu |

Companies' preferences

| | 1st | 2nd | 3rd |
|---|---|---|---|
| Alphabet | Zoran | Xavier | Yulia |
| Baidu | Yulia | Zoran | Xavier |
| Campbell | Xavier | Yulia | Zoran |

# Example



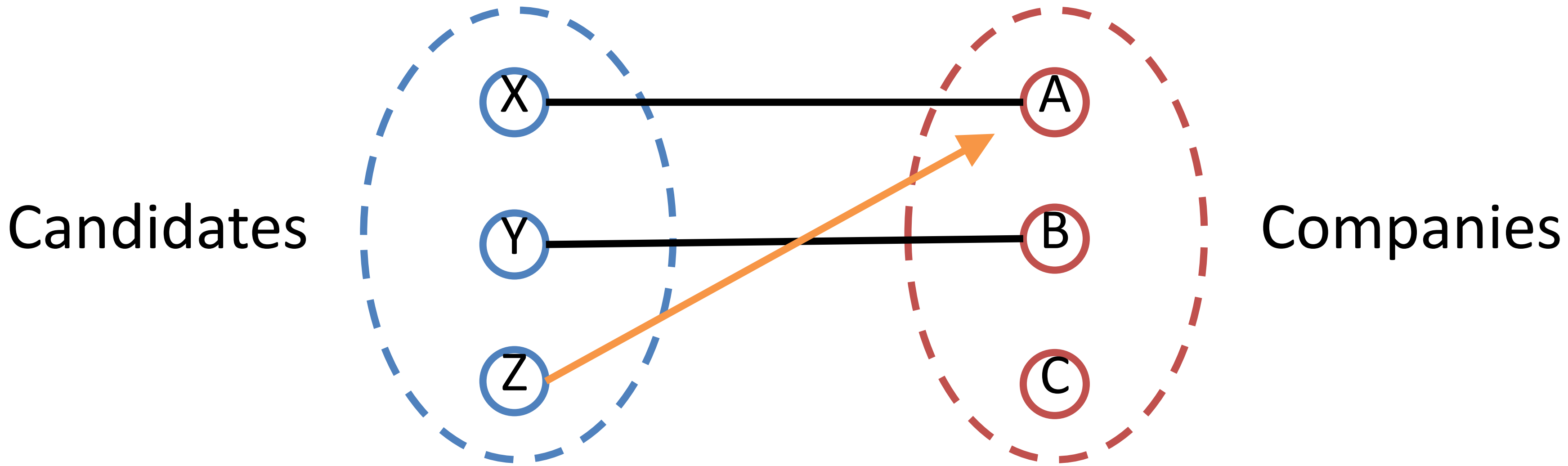

Candidates' preferences

| | 1st | 2nd | 3rd |
|---|---|---|---|
| Xavier | Baidu | Alphabet | Campbell |
| Yulia | Baidu | Campbell | Alphabet |
| Zoran | Alphabet | Campbell | Baidu |

Companies' preferences

| | 1st | 2nd | 3rd |
|---|---|---|---|
| Alphabet | Zoran | Xavier | Yulia |
| Baidu | Yulia | Zoran | Xavier |
| Campbell | Xavier | Yulia | Zoran |

# Example



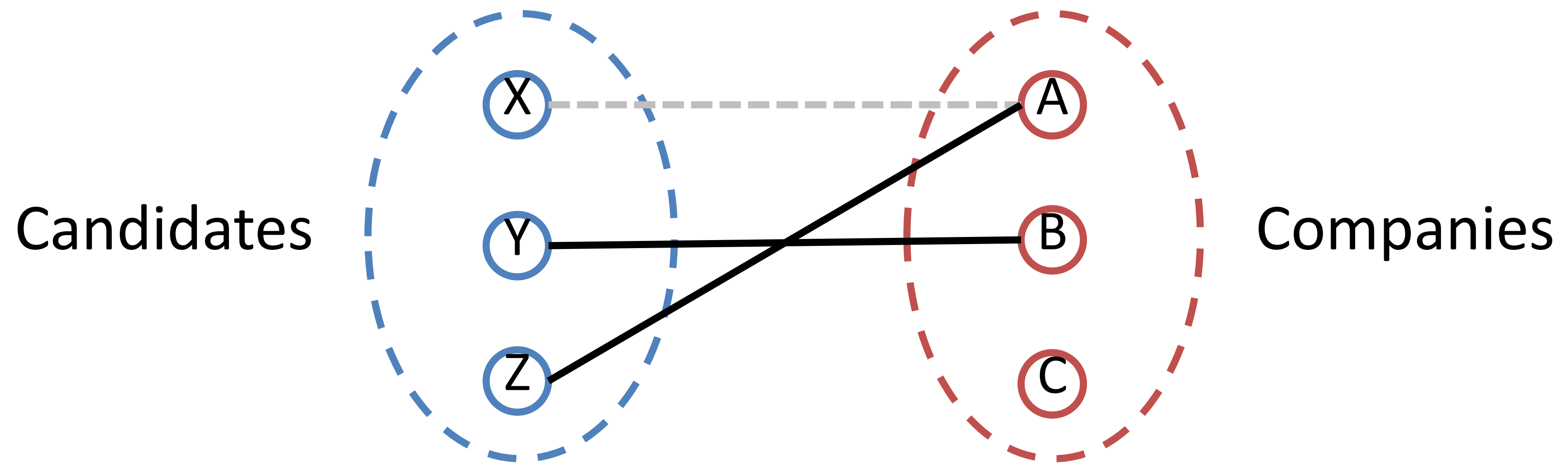

Candidates' preferences

| | 1st | 2nd | 3rd |
|---|---|---|---|
| Xavier | Baidu | Alphabet | Campbell |
| Yulia | Baidu | Campbell | Alphabet |
| Zoran | Alphabet | Campbell | Baidu |

Companies' preferences

| | 1st | 2nd | 3rd |
|---|---|---|---|
| Alphabet | Zoran | Xavier | Yulia |
| Baidu | Yulia | Zoran | Xavier |
| Campbell | Xavier | Yulia | Zoran |

# Example



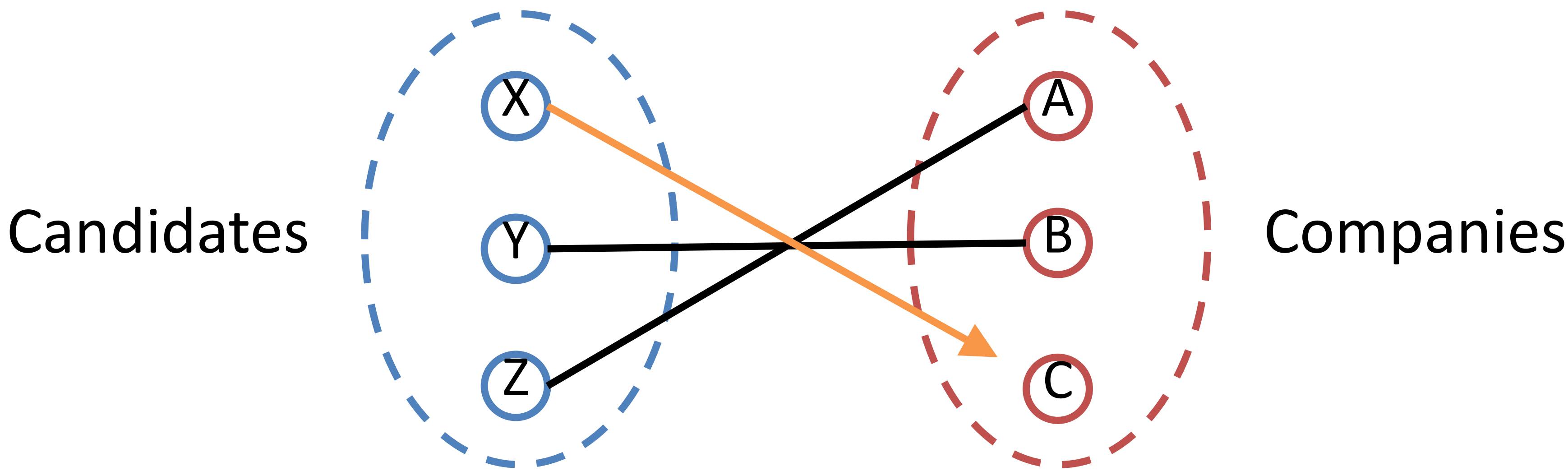

Men's preferences

| | 1st | 2nd | 3rd |
|---|---|---|---|
| Xavier | Baidu | Alphabet | Campbell |
| Yulia | Baidu | Campbell | Alphabet |
| Zoran | Alphabet | Campbell | Baidu |

Companies' preferences

| | 1st | 2nd | 3rd |
|---|---|---|---|
| Alphabet | Zoran | Xavier | Yulia |
| Baidu | Yulia | Zoran | Xavier |
| Campbell | Xavier | Yulia | Zoran |

# Example



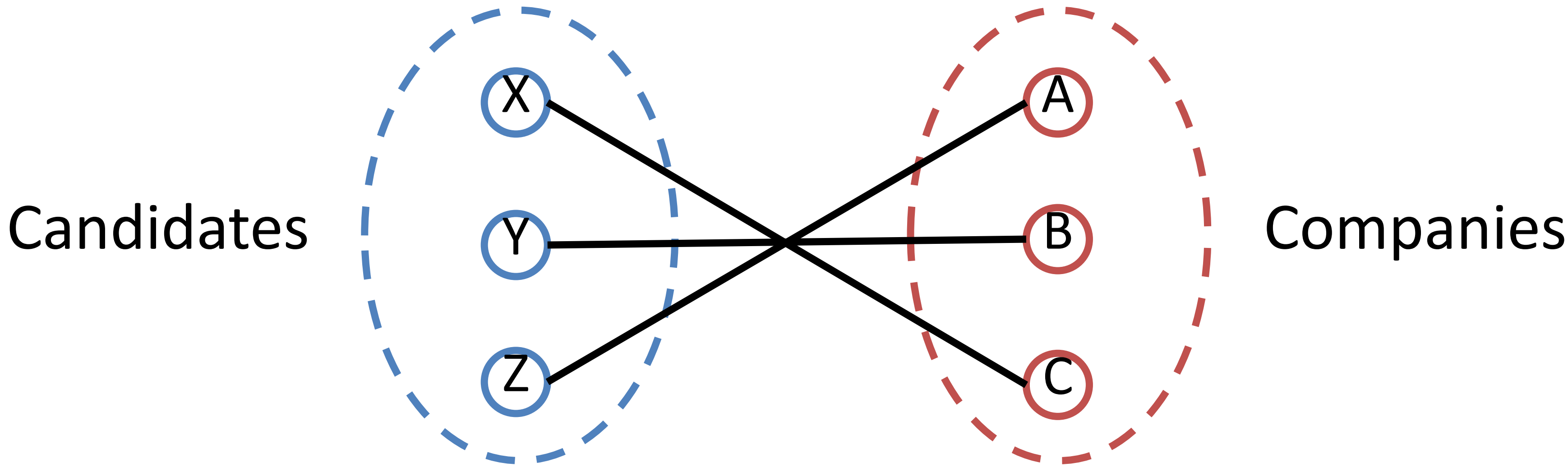

Candidates' preferences

| | 1st | 2nd | 3rd |
|---|---|---|---|
| Xavier | Baidu | Alphabet | Campbell |
| Yulia | Baidu | Campbell | Alphabet |
| Zoran | Alphabet | Campbell | Baidu |

Companies' preferences

| | 1st | 2nd | 3rd |
|---|---|---|---|
| Alphabet | Zoran | Xavier | Yulia |
| Baidu | Yulia | Zoran | Xavier |
| Campbell | Xavier | Yulia | Zoran |

# Example




Candidates' preferences

| | 1st | 2nd | 3rd |
|---|---|---|---|
| Xavier | Baidu | Alphabet | Campbell |
| Yulia | Baidu | Campbell | Alphabet |
| Zoran | Alphabet | Campbell | Baidu |

Companies' preferences

| | 1st | 2nd | 3rd |
|---|---|---|---|
| Alphabet | Zoran | Xavier | Yulia |
| Baidu | Yulia | Zoran | Xavier |
| Campbell | Xavier | Yulia | Zoran |

# Correctness (termination)

**Observations:**

1. Candidates apply to companies in decreasing order of preference.
2. Once a company is matched, it never becomes unmatched; it only "trades up."

**Claim:** Algorithm terminates after at most $n^2$ iterations of while loop (i.e. $O(n^2)$ running time).

**Proof:** Each time through the while loop a candidate applies to a new company. There are only $n^2$ possible matches. ■

# Correctness (perfection)

**Claim:** All candidates and companies get matched.

**Proof:** (by contradiction)

- Suppose, for sake of contradiction, that Zoran is not matched upon termination of algorithm.
- Then some company, say Alphabet, is not matched upon termination.
- By Observation 2 (only trading up, never becoming unmatched), Alphabet never received any application.
- But, Zoran applies everywhere. Contradiction. ■

# Correctness (stability)

**Claim:** No unstable pairs.

**Proof:** (by contradiction)

- Suppose **Z-A** is an unstable pair: they prefer each other to the association made in Gale-Shapley `matching`.
- Case 1: **Z** never applied to **A**.
  $\Rightarrow$ **Z** prefers his GS match to **A**.
  $\Rightarrow$ **Z-A** is stable.

Z would have applied to A before applying to its current match if it preferred A

- Case 2: **Z** applied to **A**.
  $\Rightarrow$ **A** rejected **Z** (right away or later)
  $\Rightarrow$ **A** prefers its GS match to **Z**.
  $\Rightarrow$ **Z-A** is stable.

If A rejected Z, it means it prefers its current match

- In either case **Z-A** is stable. Contradiction. ■

# Optimality

**Definition:** Candidate $\boldsymbol{\alpha}$ is a valid partner of company $\boldsymbol{\beta}$ if there exists some stable matching in which they are matched.

**Applicant-optimal assignment:** Each candidate receives **best** valid match (according to his preferences).

**Claim**: All executions of GS yield an **applicant-optimal** assignment, which is a stable matching!

Note: the notation “Applicant-optimal” refers to $\boldsymbol{\alpha}$-optimality

# Example

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

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

Two stable matchings: S = { X-A, Y-B, Z-C } and S’ = { Y-A, X-B, Z-C }

Then:

- Both X and Y are valid partners for A.
- Both X and Y are valid partners for B.
- Z is the only valid partner for C.
- In S’, X Y Z match their best valid partner.

# Applicant-Optimality

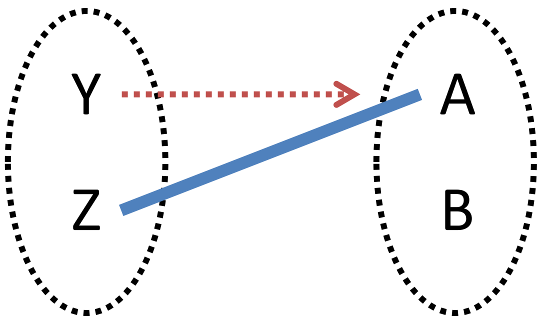
**Claim:** GS matching **S*** is applicant-optimal.

**Proof:** (by contradiction)

- Suppose some candidate is paired with a company other than his/her best option. Candidates apply in decreasing order of preference ⇒ some candidate is rejected by a valid match.
- Let **Y** be first such candidate, and let **A** be the first valid company that rejects him (i.e. **Y-A** is optimal).
- Let **S** be a stable matching (not from GS) where **Y** and **A** are matched.
- **[In GS]** when **Y** is rejected, **A** forms (or reaffirms) engagement with a candidate, say **Z**, whom it prefers to **Y** ⟹ **A** prefers **Z** to **Y**.
- Let **B** be **Z**'s match in **S**.
- **[In GS]** **Z** is not rejected by any valid match (including B) at the point when **Y** is rejected by **A** (because Y is the first valid rejection). Thus, Z has not proposed to B (a valid match) when Z proposed to A ⟹ **Z** prefers **A** to **B**.
- Thus **A**-**Z** would be preferred in **GS** (i.e. **Y-A** and **Z-B** are unstable) and **S** is not a stable matching. Contradiction. ■

# Why does Z prefer A to B?

In Gale-Shapley



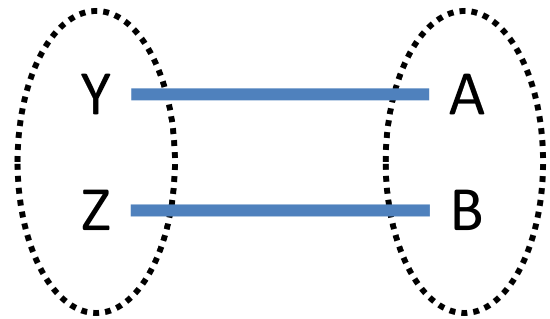

- Y is the first rejection of a valid pair.
- Y-A rejected because of Z

$\Longrightarrow$ if Z had proposed to B before it would need to break the **valid pair** Z-B first

$\Longrightarrow$ impossible (Y first reject)

$\Longrightarrow$ Z did not propose to B

S




We started from the assumption that there's a better valid pair for Y than the one found by GS.
$\Longrightarrow$ There's a stable matching S with Y-A and Z-B as pairs.

But Z prefers A to B, and A prefers Z to Y $\Longrightarrow$ Z-A is unstable $\Longrightarrow$ S is not a stable matching.

# Company($\beta$)-pessimality

Each $\beta$ receive the worst valid partner

Claim: GS find the finds a company-pessimal stable matching.

Proof: Exercise… (by contradiction)