COMP251: Bipartite graphs

Jérôme Waldispühl
School of Computer Science
McGill University

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

Recap: Dijkstra's algorithm

```
DIJKSTRA(V, E,w,s)
INIT-SINGLE-SOURCE(V,s)
S ← ∅
Q ← V
while Q ≠ ∅ do
   u ← EXTRACT-MIN(Q)
   S ← S ∪ {u}
   for each vertex v ∈ Adj[u] do
      RELAX(u,v,w)
```

Recap: Correctness of Dijkstra

Loop invariant:
At the start of each iteration of the while loop, $d[v] = \delta(s,v)$ for all $v \in S$.

Initialization:
Initially, $S = \emptyset$, so trivially true.

Termination:
At end, $Q=\emptyset \Rightarrow S = V \Rightarrow d[v] = \delta(s,v)$ for all $v \in V$.

Maintenance:
Show that $d[u] = \delta(s,u)$ when u is added to S in each Iteration.

Recap: Correctness of Dijkstra

Show that $d[u] = \delta(s,u)$ when u is added to S in each iteration.

Suppose there exists u such that $d[u] \neq \delta(s,u)$.

Let u be the first vertex for which $d[u] \neq \delta(s, u)$ when u is added to S.

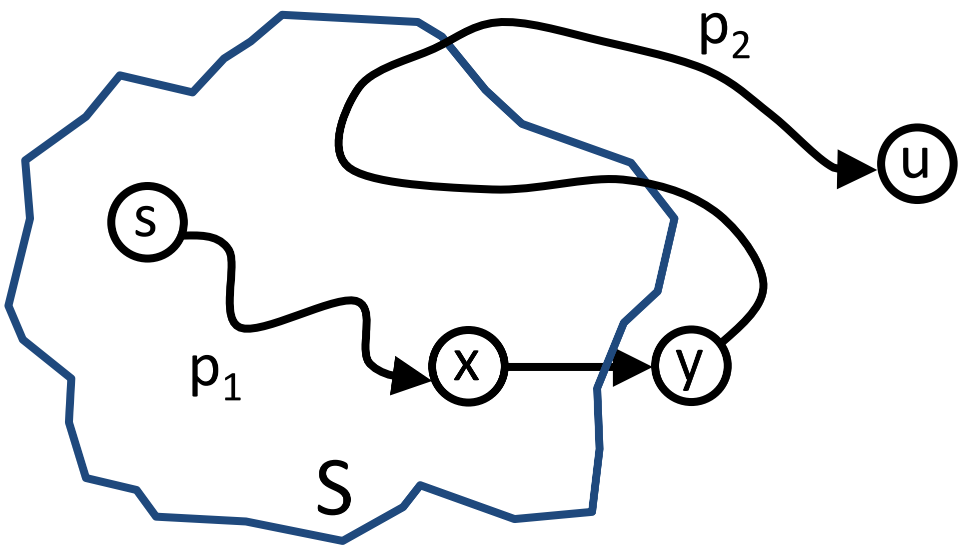
- $u \neq s$, since $d[s] = \delta(s,s) = 0$.
- Therefore, $s \in S$, so $S \neq \emptyset$.
- There must be some path $s \rightsquigarrow u$. Otherwise $d[u] = \delta(s,u) = \infty$ by no-path property.
- So, there is a path $s \rightsquigarrow u$. Thus, there is a shortest path p $s \overset{p}{\rightsquigarrow} u$.

Recap: Correctness of Dijkstra

Show that $d[u] = \delta(s,u)$ when u is added to S in each iteration.

Just before u is added to S, shortest path p connects a vertex in S (i.e., s) to a vertex in $V - S$ (i.e., u).

Let y be first vertex along p that is in $V - S$, and let $x \in S$ be y is predecessor.



Decompose p into $s \overset{p_1}{\leadsto} x \to y \overset{p_2}{\leadsto} u$.

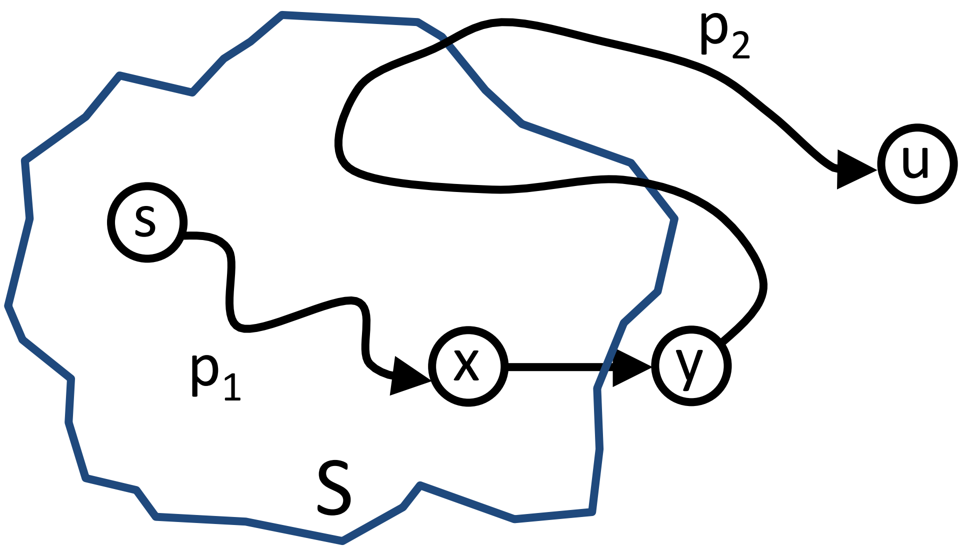
Recap: Correctness of Dijkstra

Claim: d[y] = $\delta(s, y)$ when u is added to S.

Proof:

x $\in$ S and u is the first vertex such that d[u] $\neq$ $\delta(s, u)$ when u is added to S $\Rightarrow$ d[x] = $\delta(s, x)$ when x is added to S.

Relaxed (x, y) at that time, so by the convergence property, d[y] = $\delta(s, y)$.



Recap: Correctness of Dijkstra

Show that $d[u] = \delta(s,u)$ when u is added to S in each iteration.

Now can get a contradiction to $d[u] \neq \delta(s, u)$:

y is on shortest path $s \overset{p}{\leadsto} u$, and all edge weights are nonnegative.

$\Rightarrow d[y] = \delta(s,y)$ (from previous claim)

$\leq \delta(s,u)$ (by sub-optimal paths property)

$\leq d[u]$ (upper-bound property)

In addition, since y and u were in Q when we chose u: $d[u] \leq d[y]$

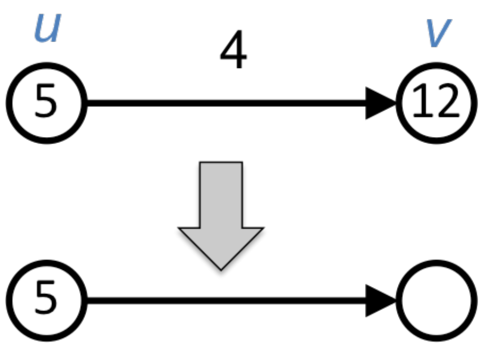
$\Rightarrow d[u] = d[y]$.

But $d[y] \leq \delta(s,u) \leq d[u] \Rightarrow d[y] = \delta(s, u) = d[u]$.

Contradicts assumption that $d[u] \neq \delta(s,u)$. ■

what will be the value of d[v] after relaxation of the edge (u,v)?

- 12
- 9 ✓
- 17
- 7
- None of the values proposed



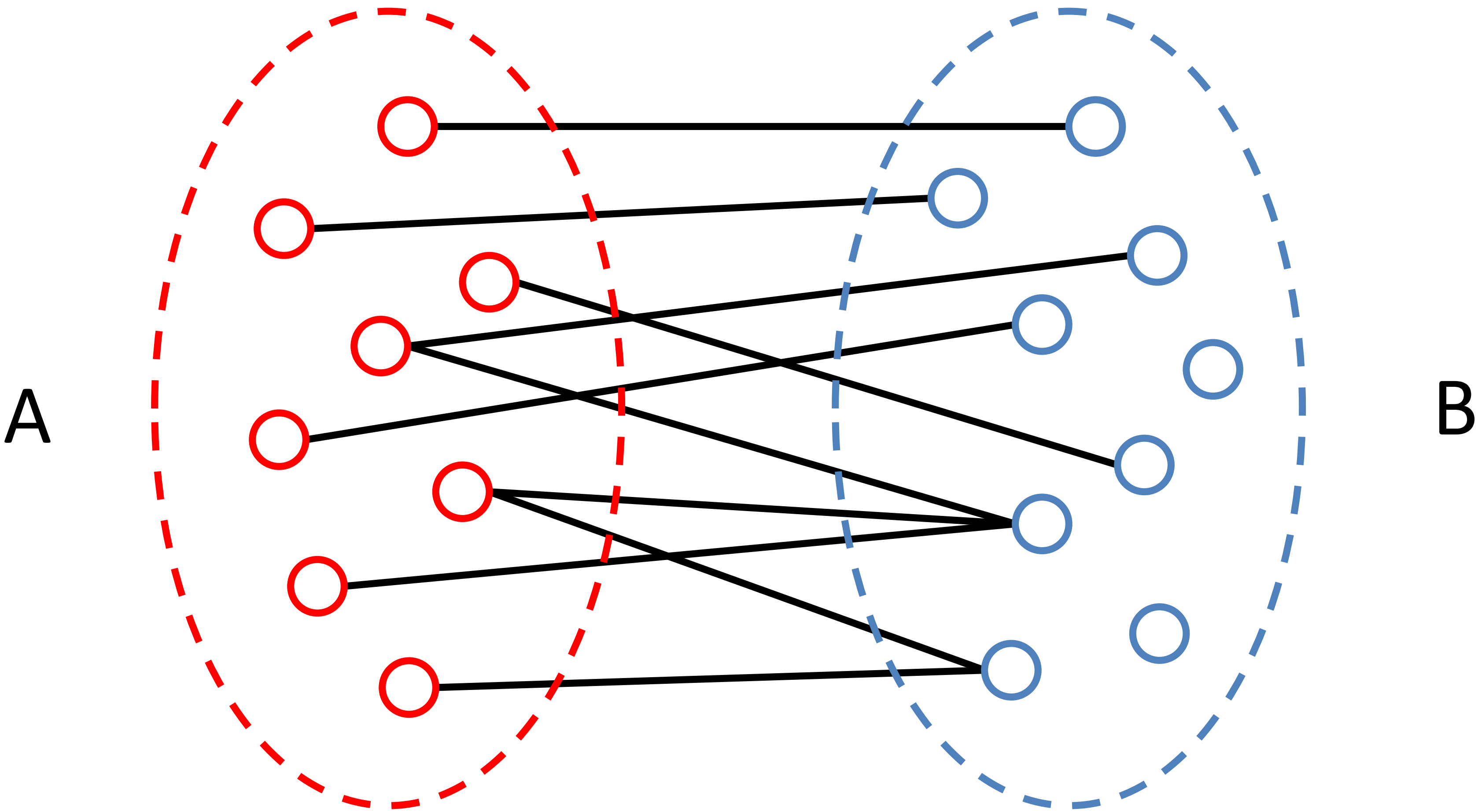
We want to calculate the shortest paths from s in a DAG with negative weight edges. Is it ok?

- Yes because there are no negative weight cycles. ✓
- Not a problem. Negative weight edges cannot be reached from the source.
- This is wrong! Negative weight edges are forbidden even in case of a DAG.

Let u be a vertex extracted from the queue during the execution of the Dijkstra's algorithm. What would happen if we use a First-In-First-Out queue instead of a min priority queue?

- We cannot guarantee that the shortest-path estimate of u is the shortest path from s to u. ✓
- Relaxing the outgoing edges of u is useless (i.e. it will not change the shortest path estimates).
- It does not matter. We can use a FIFO queue.

Bipartite graphs

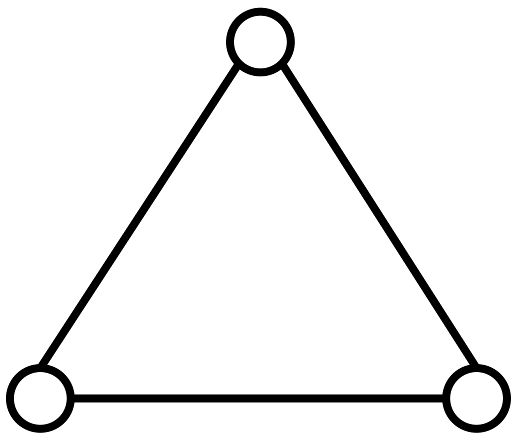


Vertices are partitioned into 2 sets.
All edges cross the sets.

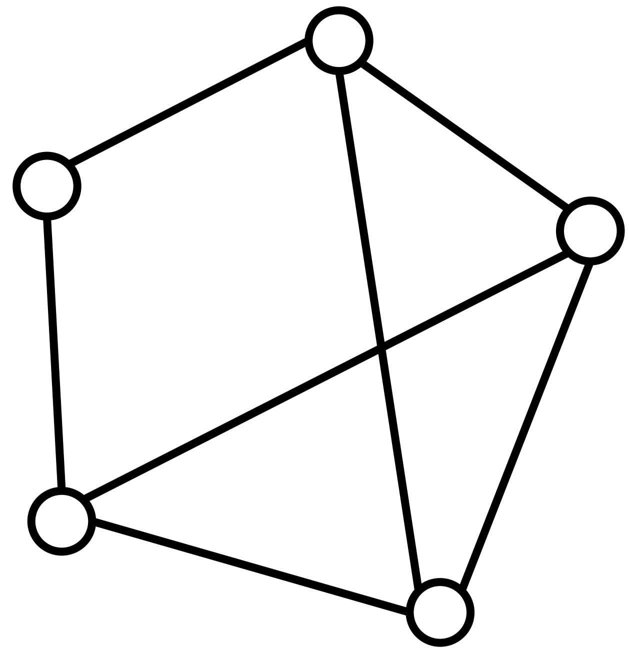
Examples

A		B
Women	Traditional marriage	Men
Students	registration	Courses
People	employment	Companies
People	Have read/seen	Books/Movies

Counter-examples



Easy to identify.



But not always...

Cycles

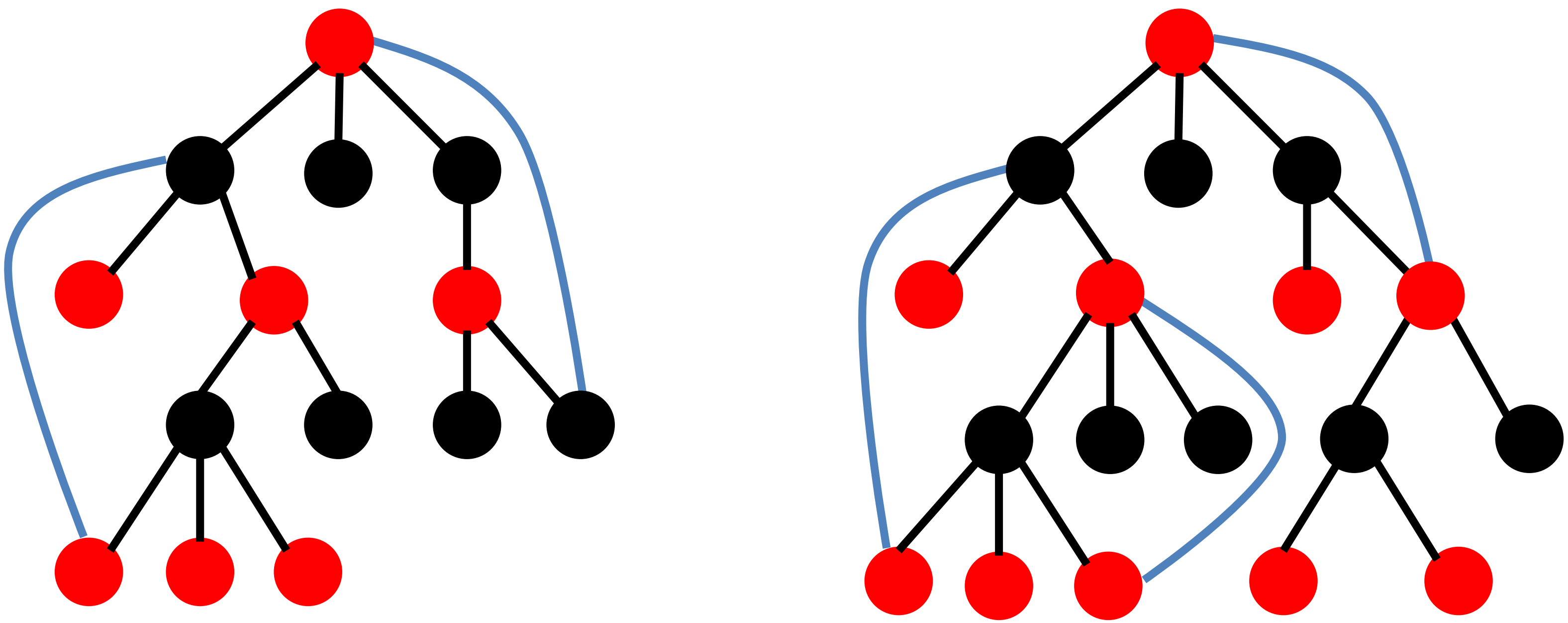
Claim: If a graph is bipartite if and only if does not contain an odd cycle.

Proof: Q5 of assignment 2.

Is it a bipartite graph?

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

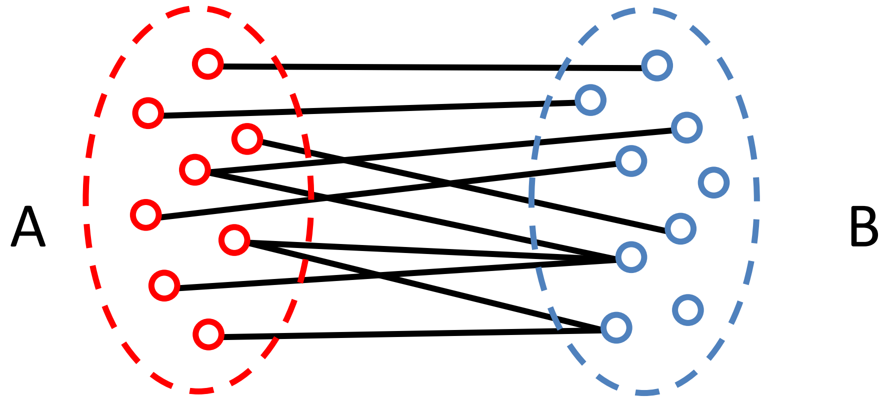
1. Run DFS and use it to build a DFS tree.
2. Color vertices by layers (e.g. red & black)
3. If all non-tree edges join vertices of different color then the graph is bipartite.



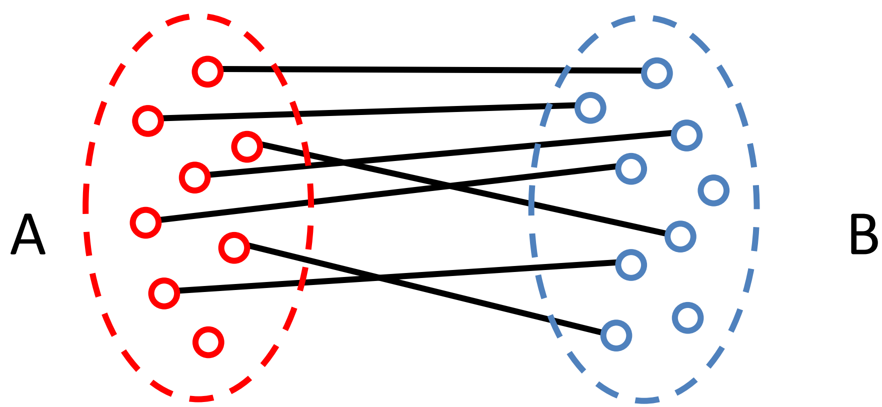
Non-tree edges in DFS tree cross 2 or more levels. Why?

Bipartite matching

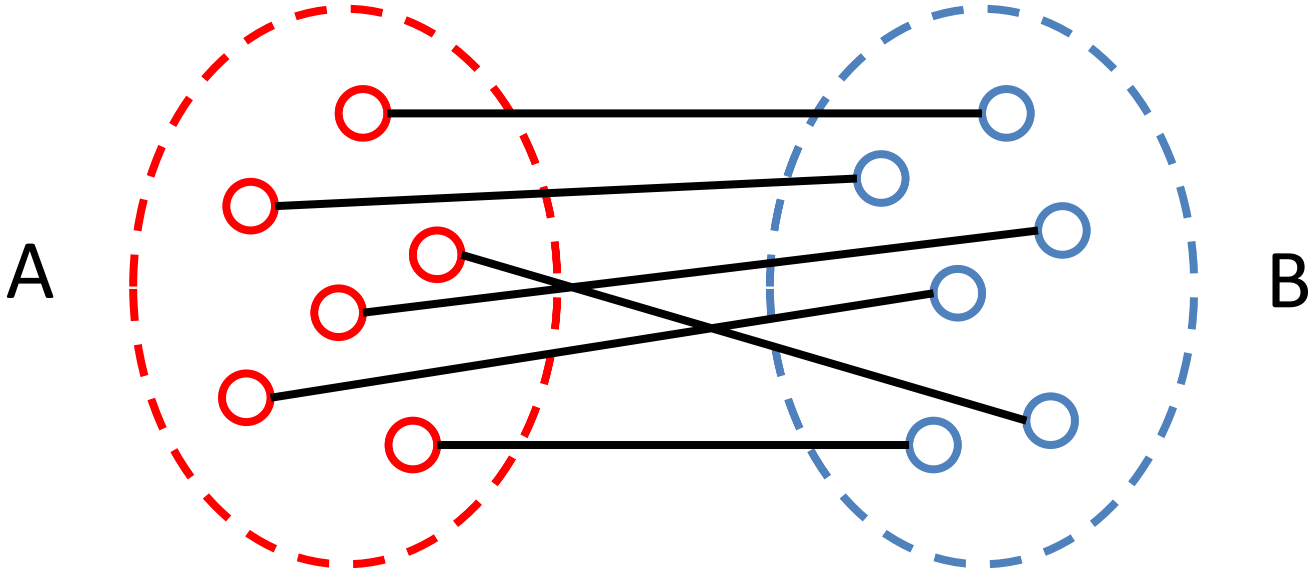
Consider an undirected bipartite graph.



A matching is a subset of the edges { (α, β) } such that no two edges share a vertex.



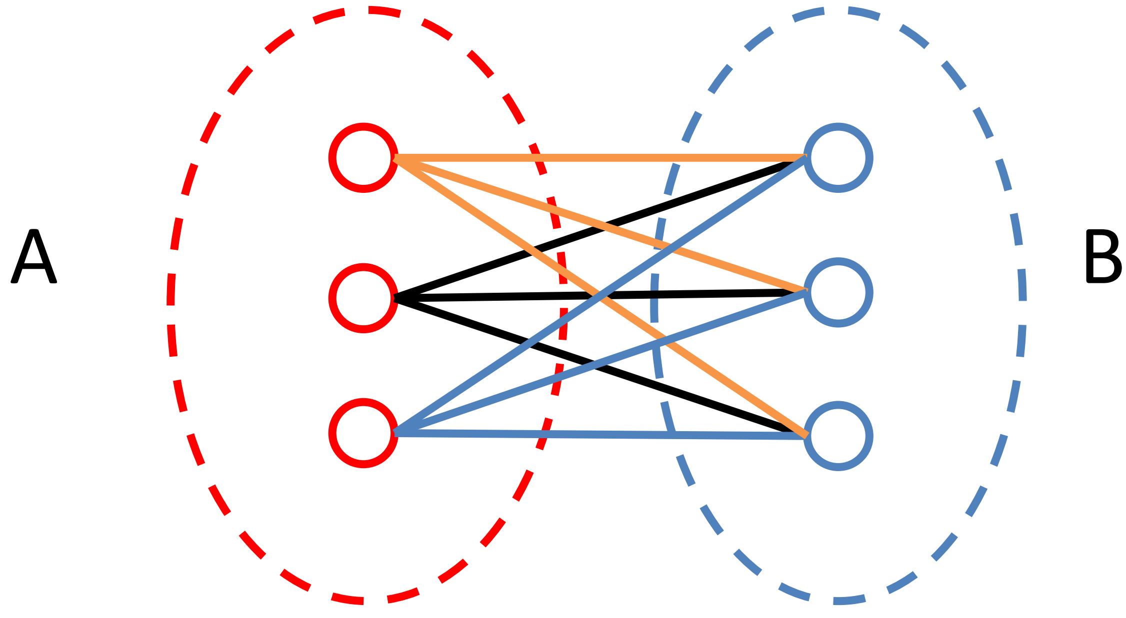
Perfect matching



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 (α, β) 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 marriage problem

Goal: Given **n** elements of **A** and **n** elements of **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

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

B's preferences

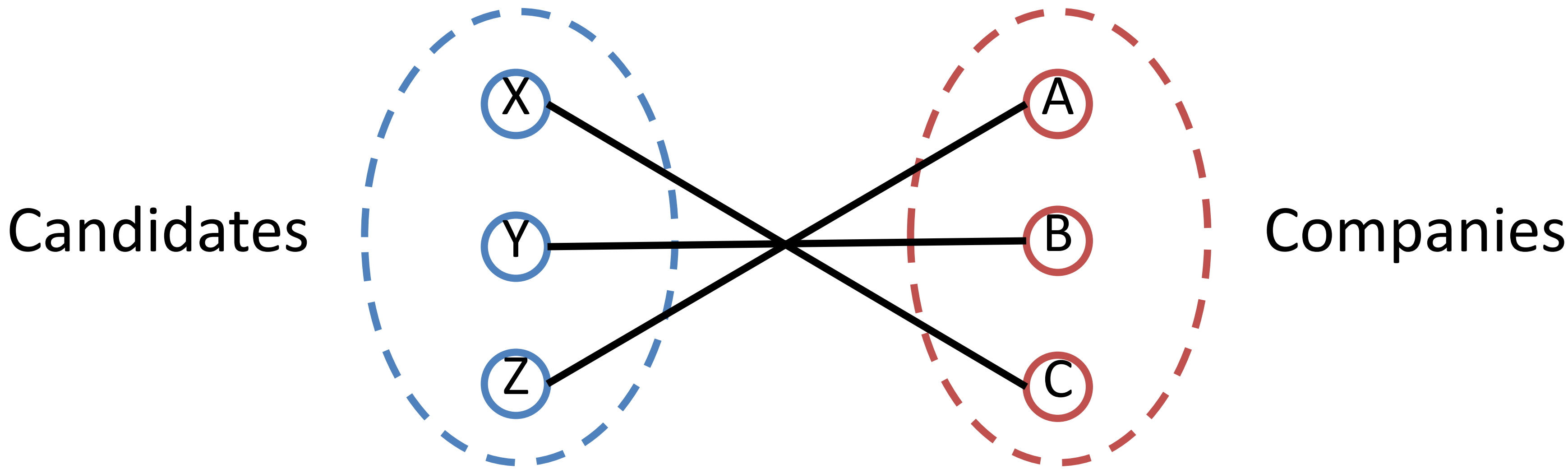
	1st	2nd	3rd
Alphabet	Yulia	Xavier	Zoran
Baidu	Xavier	Yulia	Zoran
Campbell	Xavier	Yulia	Zoran

Stable marriage 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

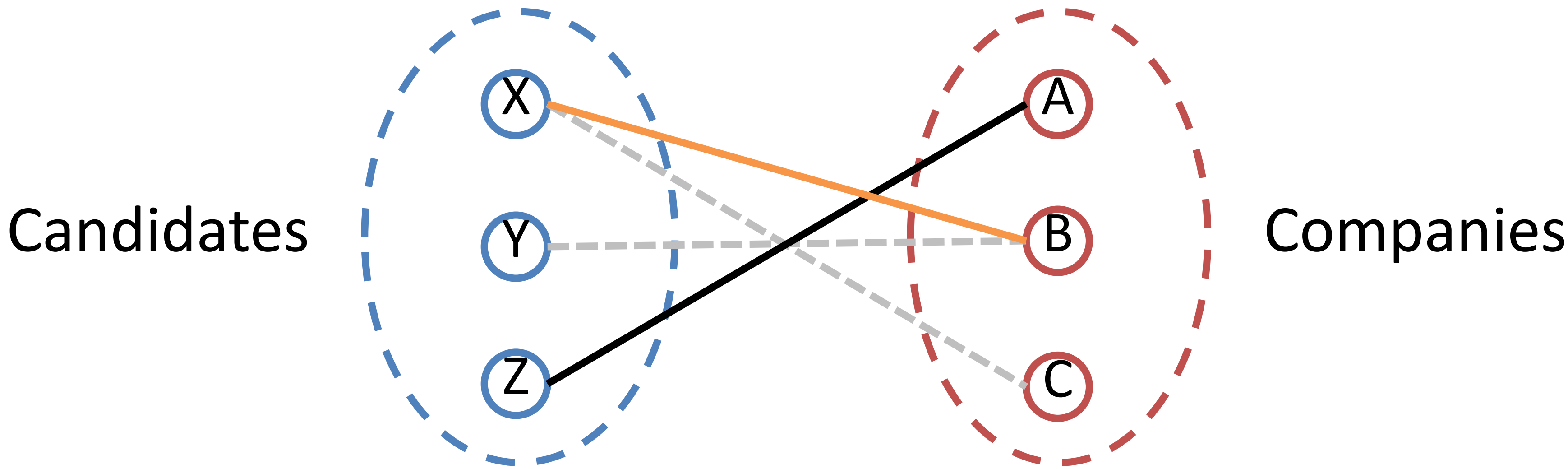
Companies' preferences

	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

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

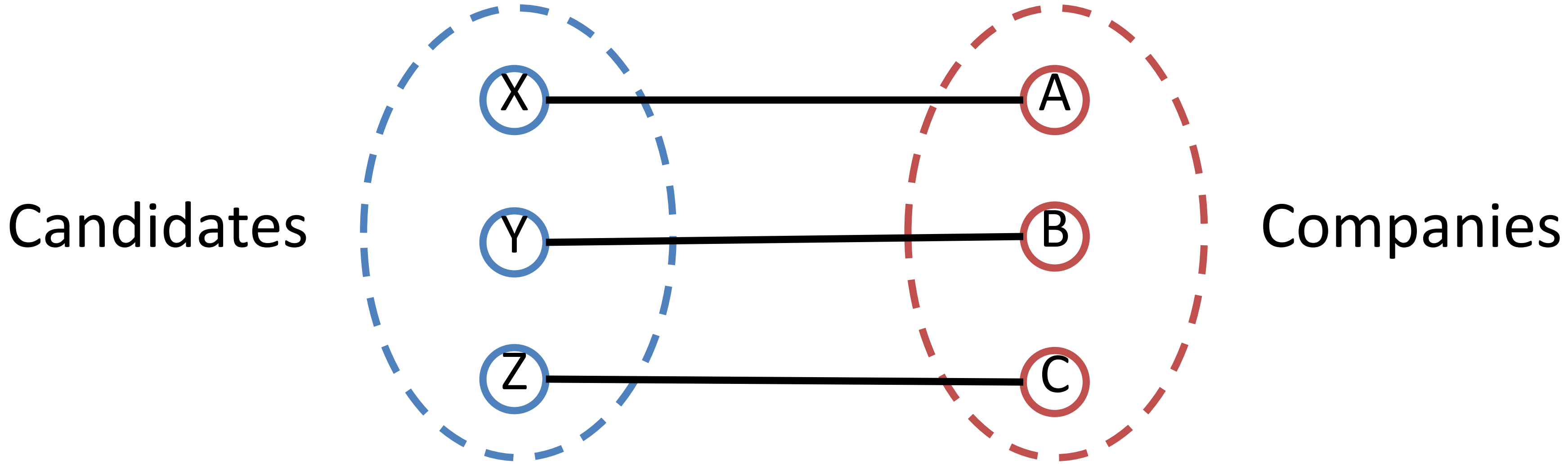
Companies' preferences

	1^{st}	2^{nd}	3^{rd}
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 “marriage” 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.

Algorithm for finding a matching:

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

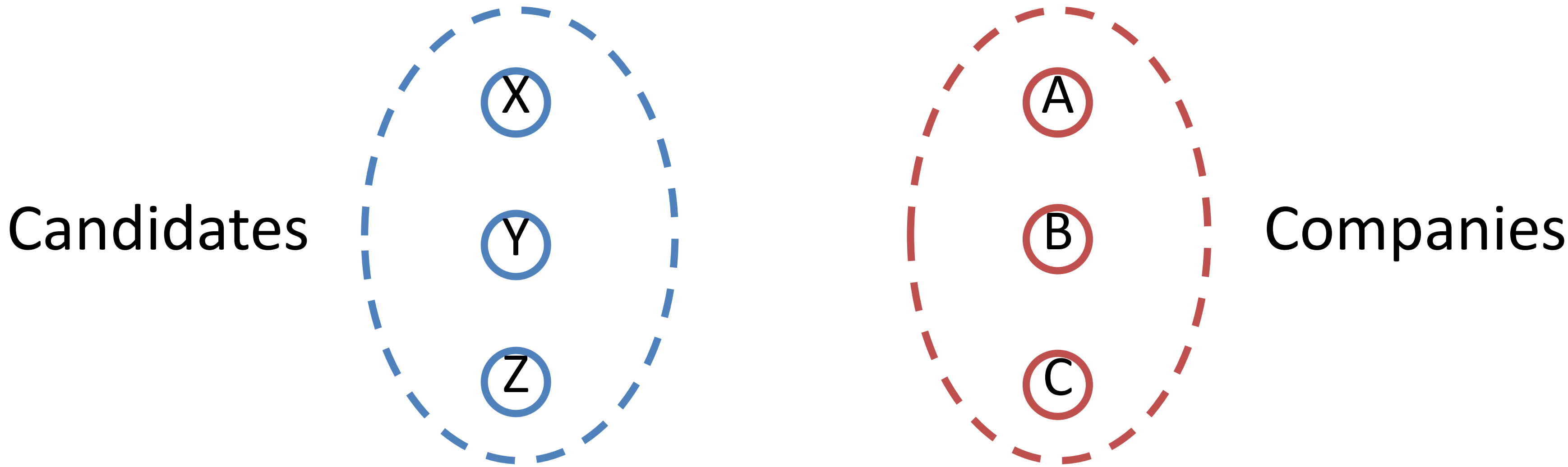
For each $\alpha \in A$, let pref[α] be the ordering of its preferences in B

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`$\leftarrow \varnothing$
while there is $\alpha \in A$ not yet matched **do**
    $\beta \leftarrow$pref[α].removeFirst()
    if β not yet matched **then**
        `matching`$\leftarrow$`matching`$\cup\{(\alpha,\beta)\}$
    else
        $\gamma \leftarrow \beta$'s current match
        if β prefers α over γ **then**
            `matching`$\leftarrow$`matching`$-\{(\gamma,\beta)\}\cup\{(\alpha,\beta)\}$
return `matching`

Example



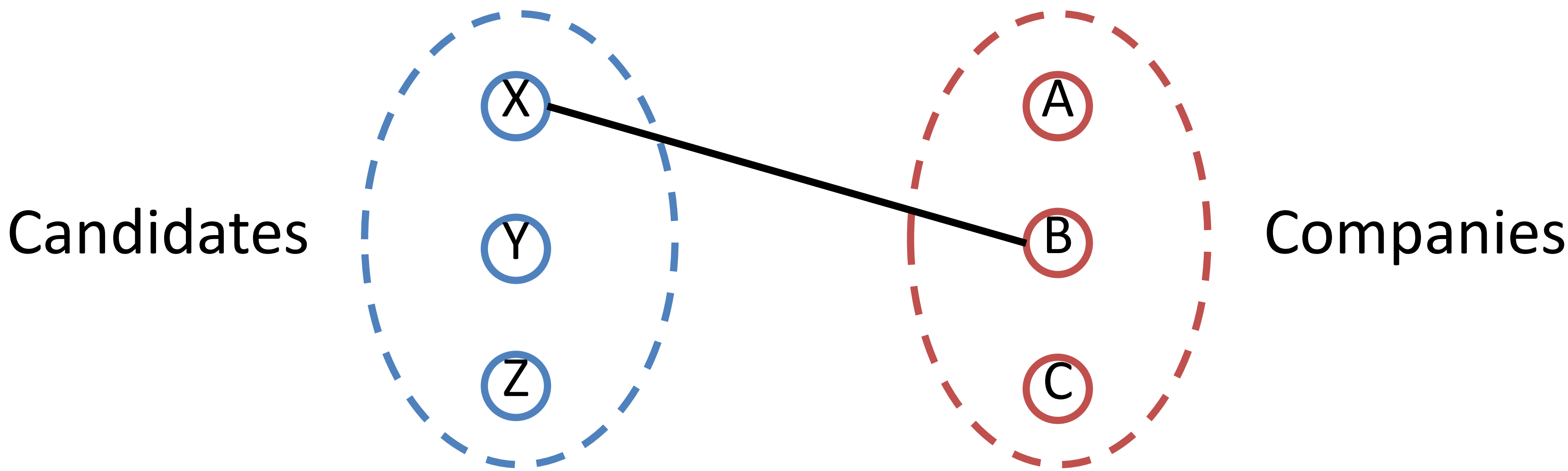
Candidates' preferences

	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



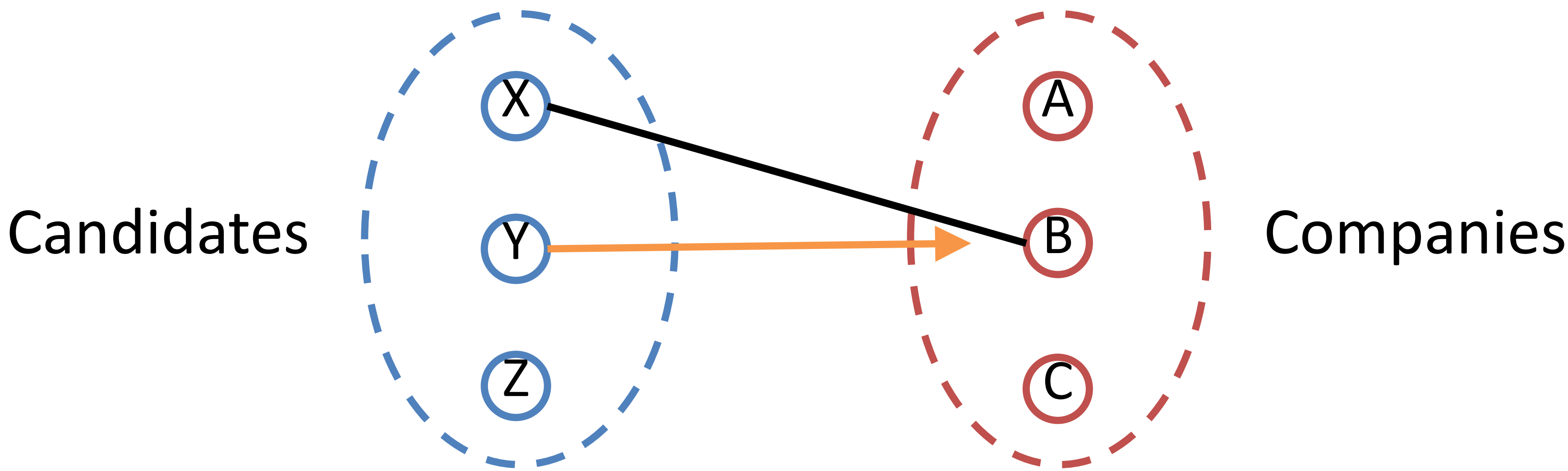
Candidates' preferences

	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



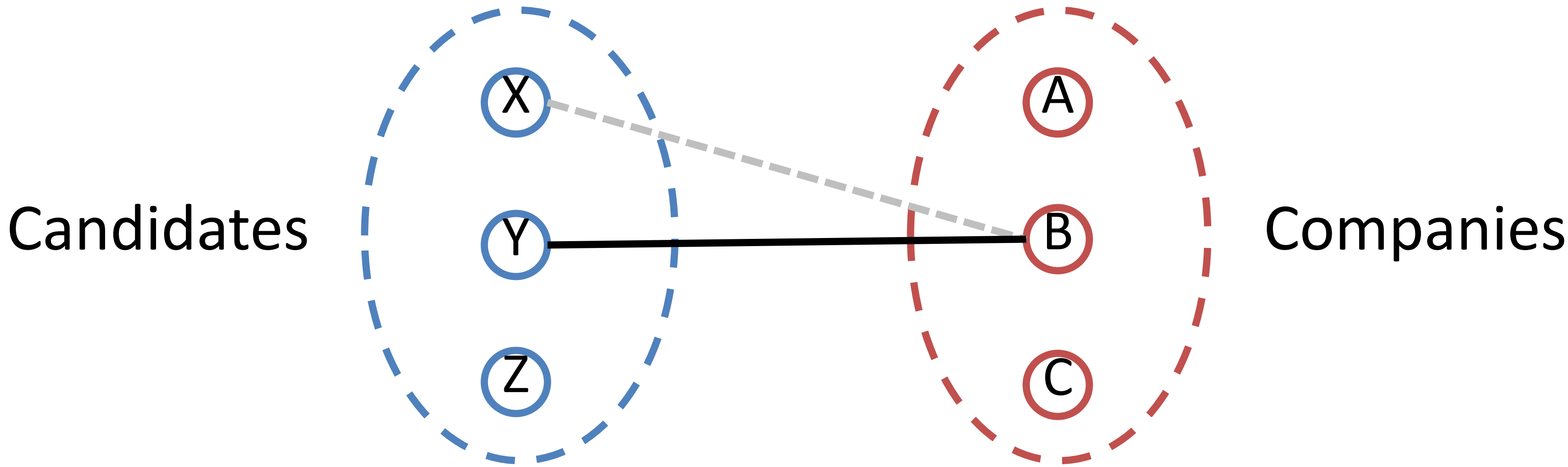
Candidates' preferences

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

Companies' preferences

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

Example



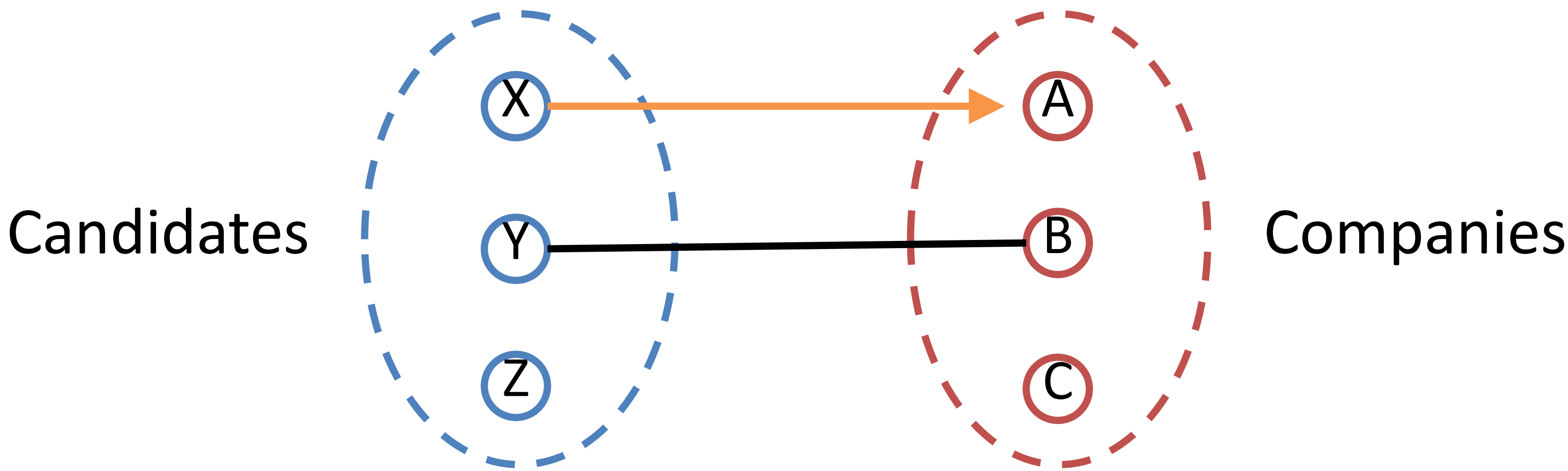
Candidates' preferences

	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



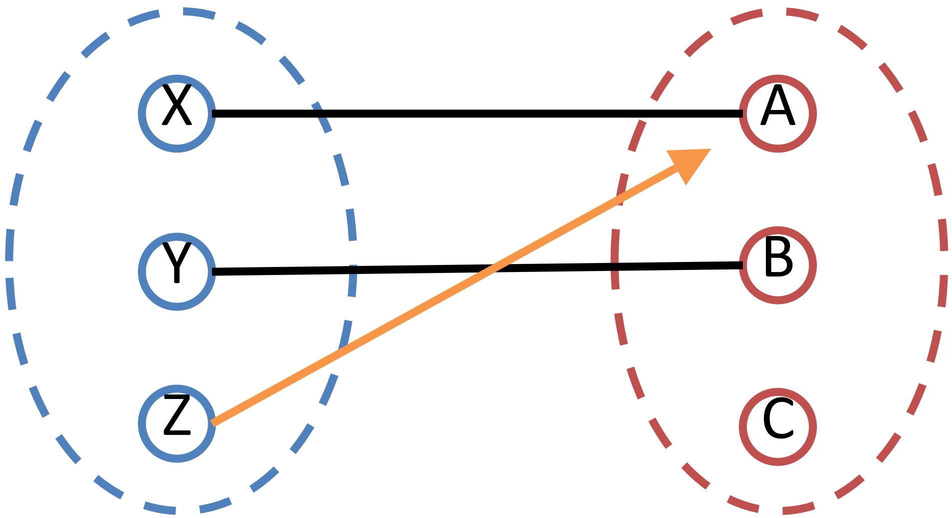
Candidates' preferences

	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

Companies

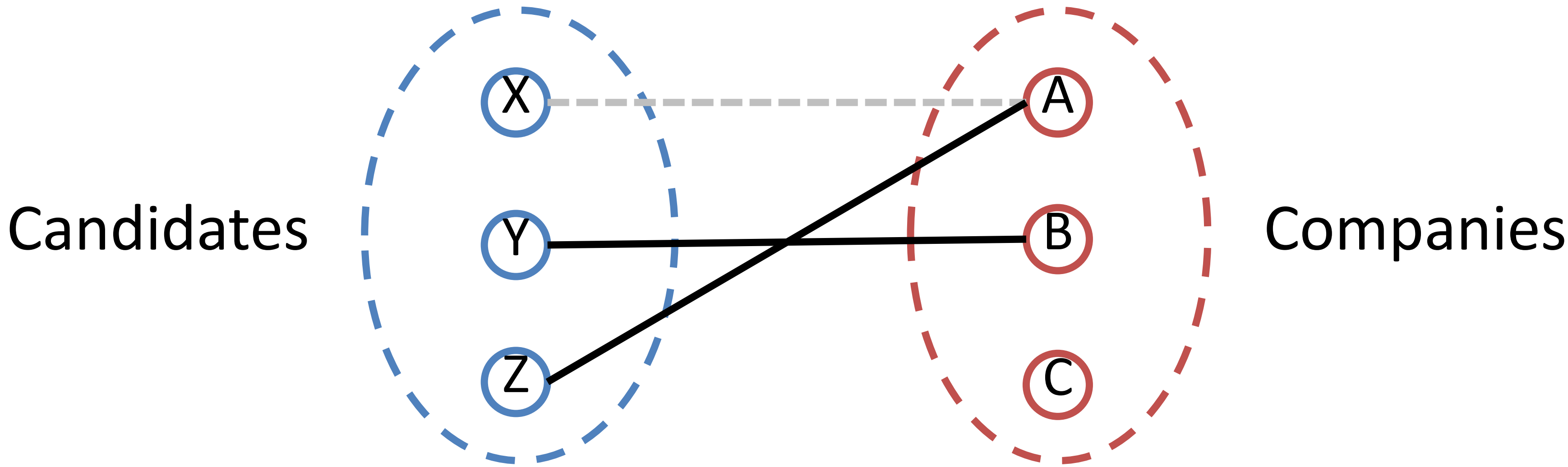
Candidates' preferences

	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



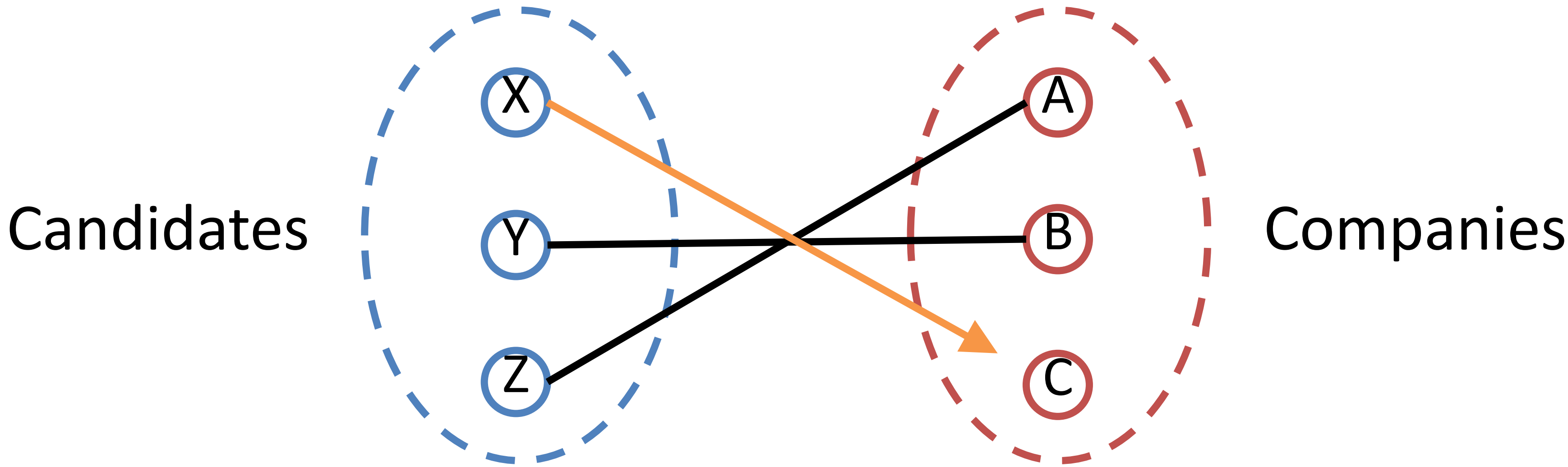
Men’s preferences

	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



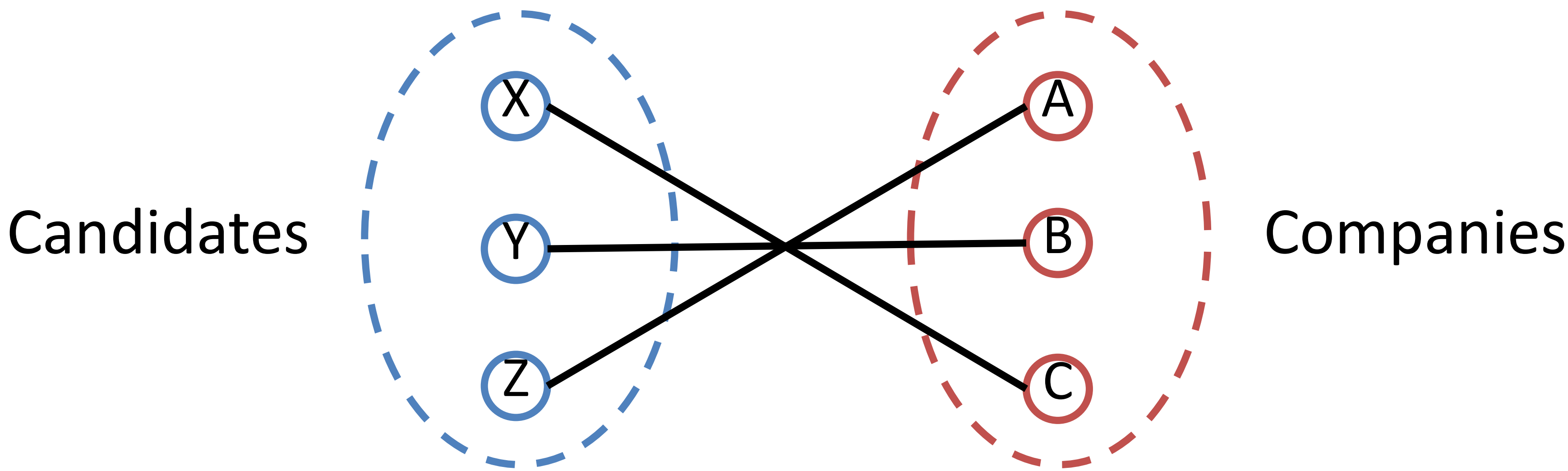
Candidates' preferences

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

Optimality

Definition: Candidate **α** is a valid partner of company **β** 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 a **applicant-optimal** assignment, which is a stable matching!

Applicant-Optimality

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

Proof: (by contradiction)

- Suppose some candidate is paired with someone other than his best option. Candidates apply in decreasing order of preference $\Rightarrow$ 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. **A-Y** is optimal).
- Let **S** be a stable matching (not from GS) where **A** and **Y** are matched.
- In GS**,** when **Y** is rejected, **A** forms (or reaffirms) engagement with a candidate, say **Z**, whom it prefers to **Y** (i.e. **A** prefers **Z** to **Y)**
- Let **B** be **Z**'s match in **S** (i.e. **B-Z** is optimal).
- In GS**, Z** is not rejected by any valid match at the point when **Y** is rejected by **A**.
- Thus, **Z** prefers **A** to **B** (because **B-Z** is optimal).
- But **A** prefers **Z** to **Y**.
- Thus **A**-**Z** would be preferred in **S** (i.e. **A-Y** and **B-Z** are unstable).