# Lecture 17

Gale-Shapley (Deferred Acceptance) Algorithm

# Announcements

- HW8 due on Wednesday.

- This week's lectures (including this one) are NOT on the final exam.

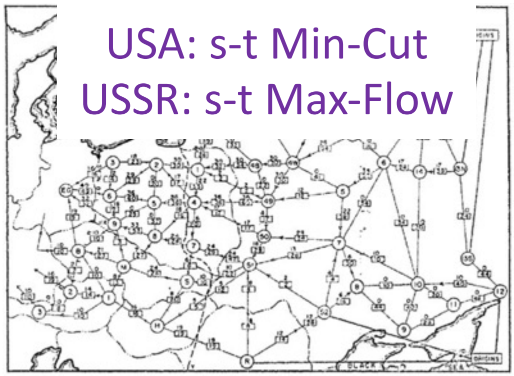
# Recap: One way to greedy algorithms

- Greedy algorithms
  - Make a series of choices.
    - Choose this activity, then that one, ..
    - Never backtrack.
  - Show (or hope) that your choice never rules out success.
    - At every step, there exists an optimal solution consistent with the choices we've made so far.
  - At the end of the day:
    - you've built only one solution,
    - never having ruled out success,
    - **so your solution must be correct.**

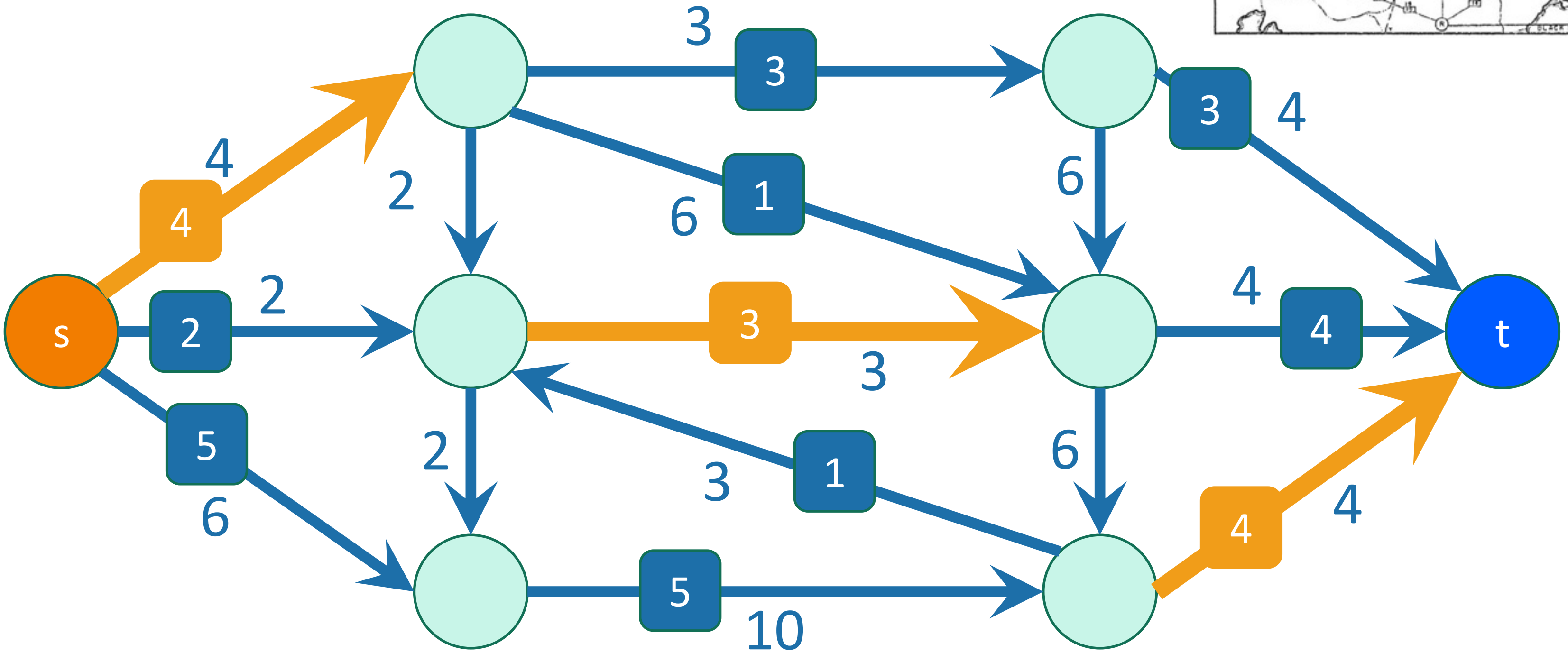
# Recap: A different approach to greedy

- Greedy algorithms
  - Make a series of choices.
    - Choose this activity, then that one, ..
    - ~~Never backtrack.~~
  - **Instead:** At each step, free to revert any of the choices we've already made – as long as the solution is improving!

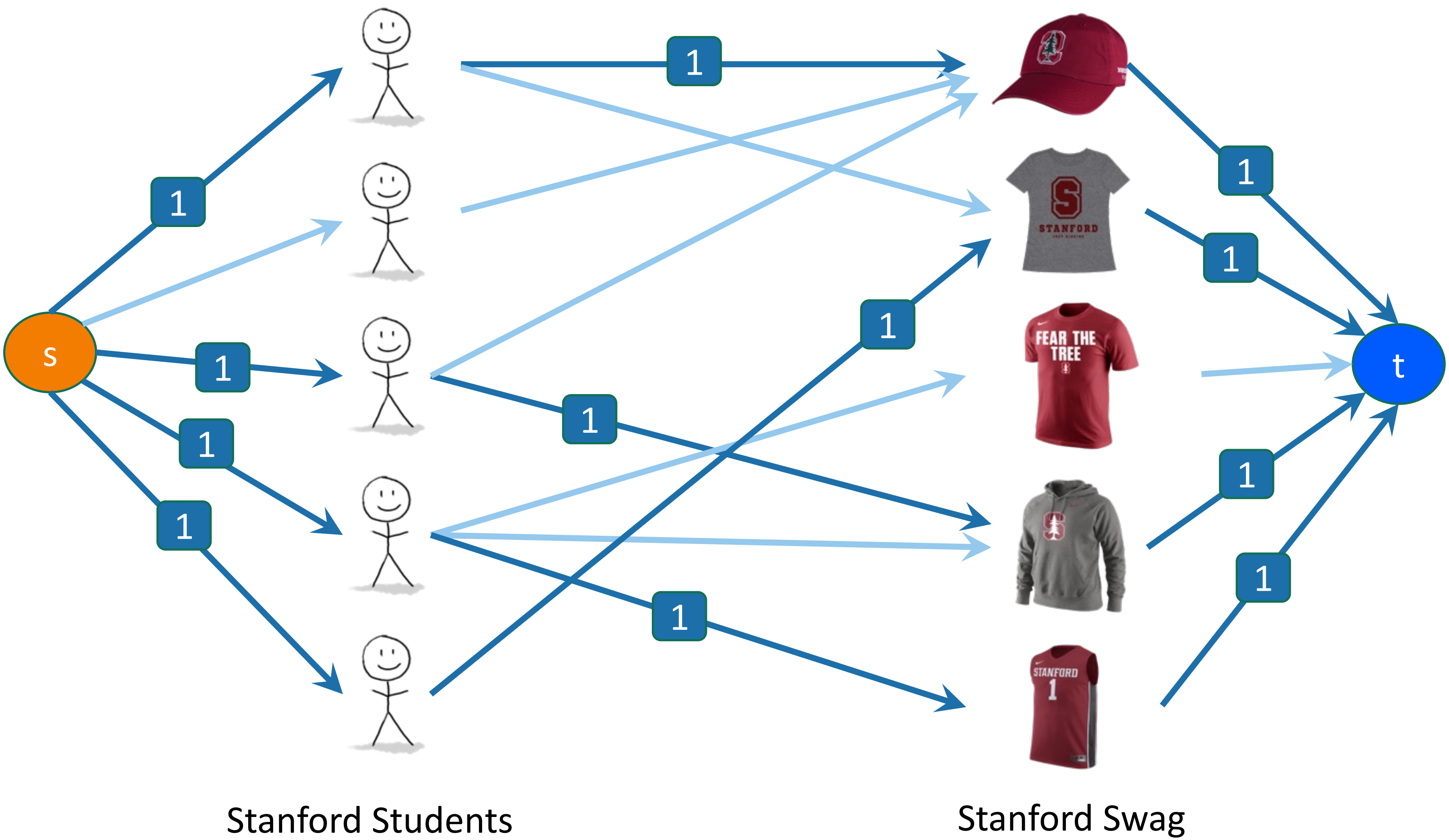
# Recap: Ford-Fulkerson algorithm for s-t min-cut / max-flow



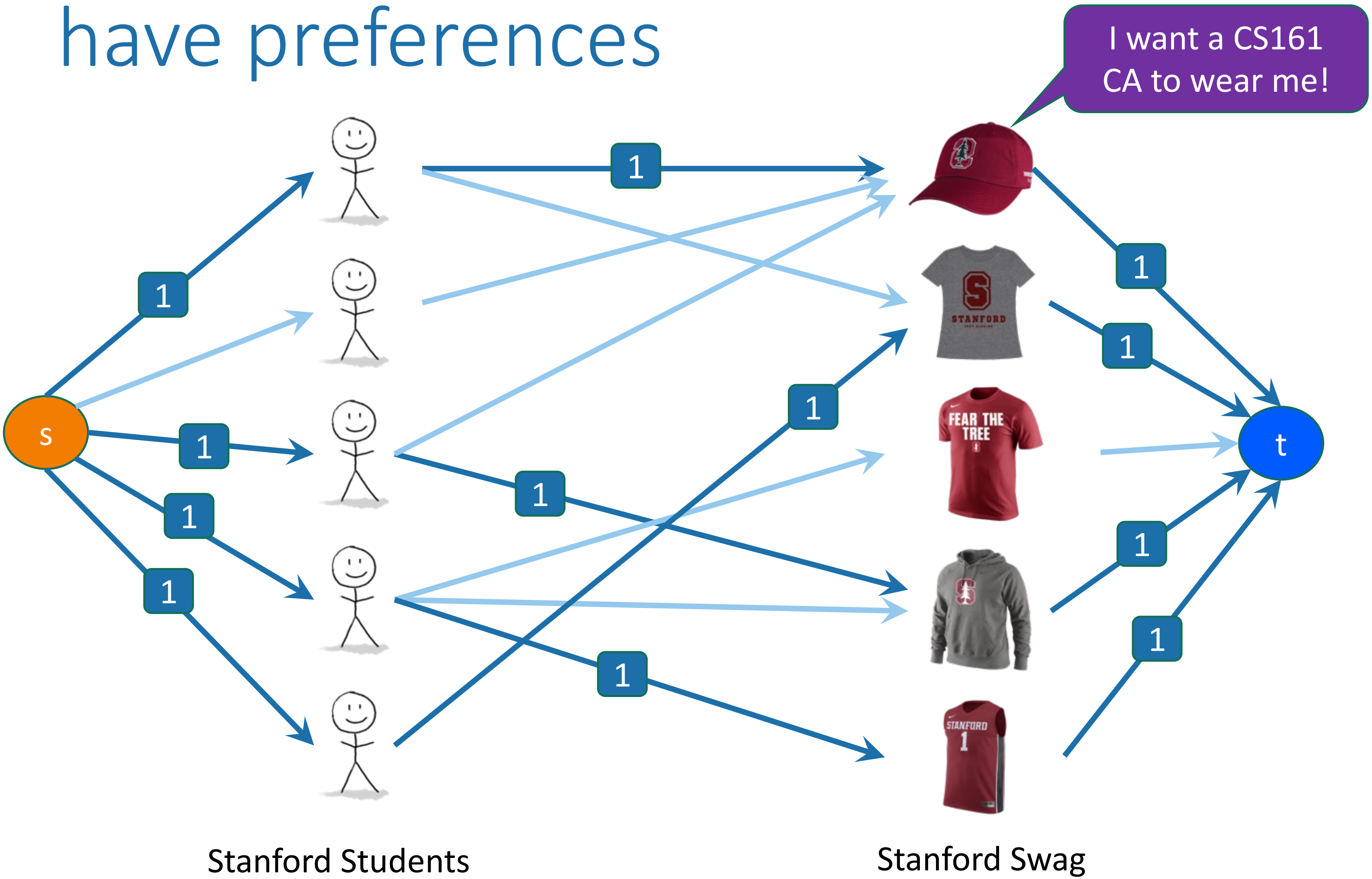

**The value of a max flow from s to t**
*is equal to*
**the cost of a min s-t cut.**



# Recap: used s-t max-flow to solve assignment problems



# Today: matching when both sides have preferences



# Today

- Hospitals/residents problem

- Stable matchings
  - Solve the hospitals/residents problem
  - But can we find them?

- Deferred Acceptance Algorithm
  - Find stable matchings!

- Discussion, applications and non-applications

# The hospital residency problem

- After completing medical school, students are finally ready to start their “residency” (similar to job internship):
  - In contrast, I’m told that many of you can get an internship after completing CS161…

- Each applicant has a preference over different residency programs.
- Each program has a preference over the applicants.

How should you match applicants to residencies?

Simplifying assumption today:
Each program has 1 slot

# The hospital residency problem

- After completing medical school, students are finally ready to start their "residency" (similar to job internship):
  - In contrast, I'm told that many of you can get an internship after completing CS161…

- Each **doctor** has a preference over different **hospitals**.
- Each **hospital** has a preference over the **doctors**.

How should you match doctors with hospitals?

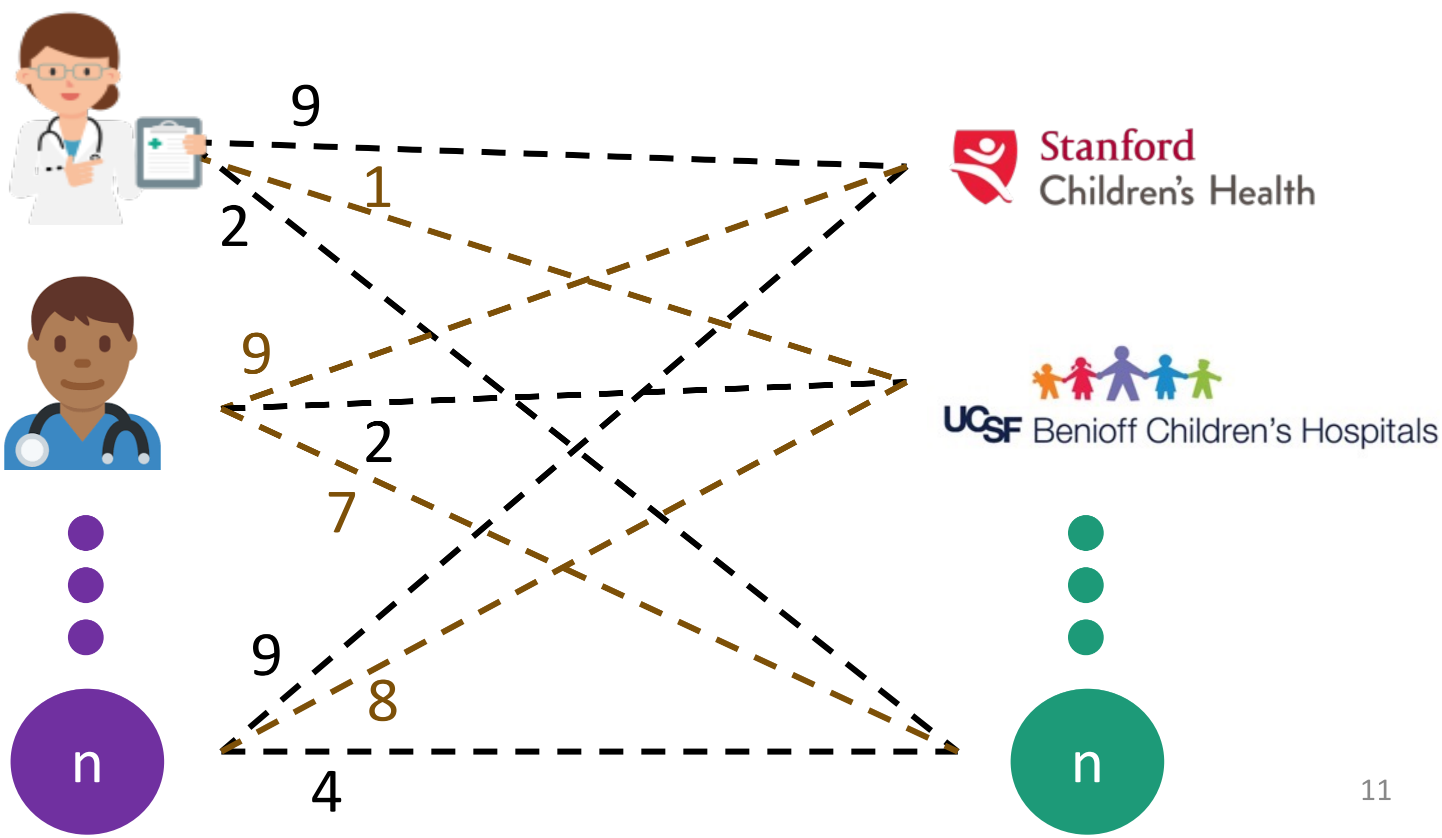
Simplifying assumption today:
Each hospital has 1 slot

This slide just for intuition!

# One way to model this problem

- Each doctor has a preference over hospitals
- Each hospital has a preference over the doctor

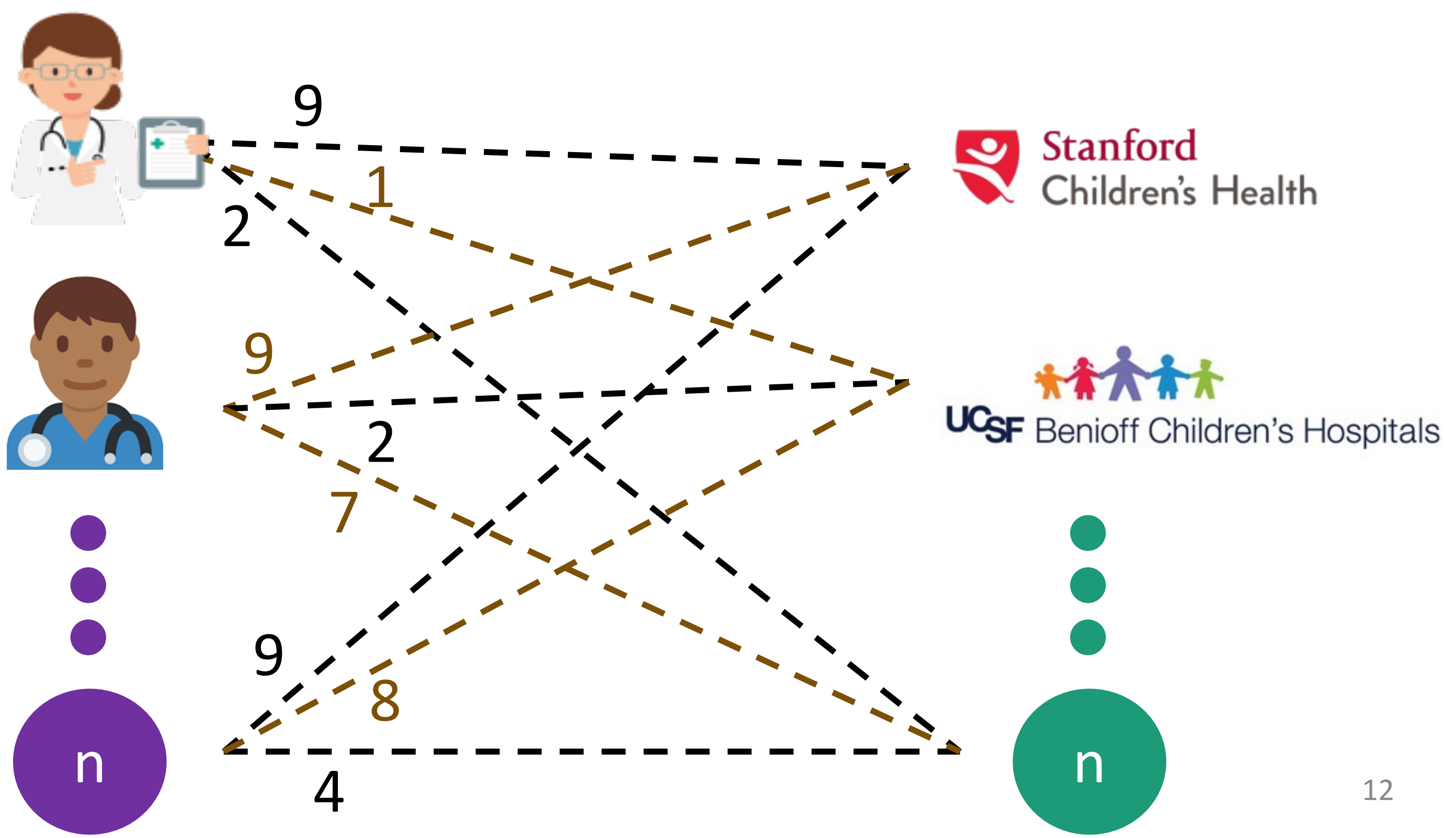
How should you match doctors with hospitals?



This slide just for intuition!

# One way to model this problem

- Bipartite graph between doctors and hospitals
- Weights on edges = some function of preferences (highest weight = most preferred)

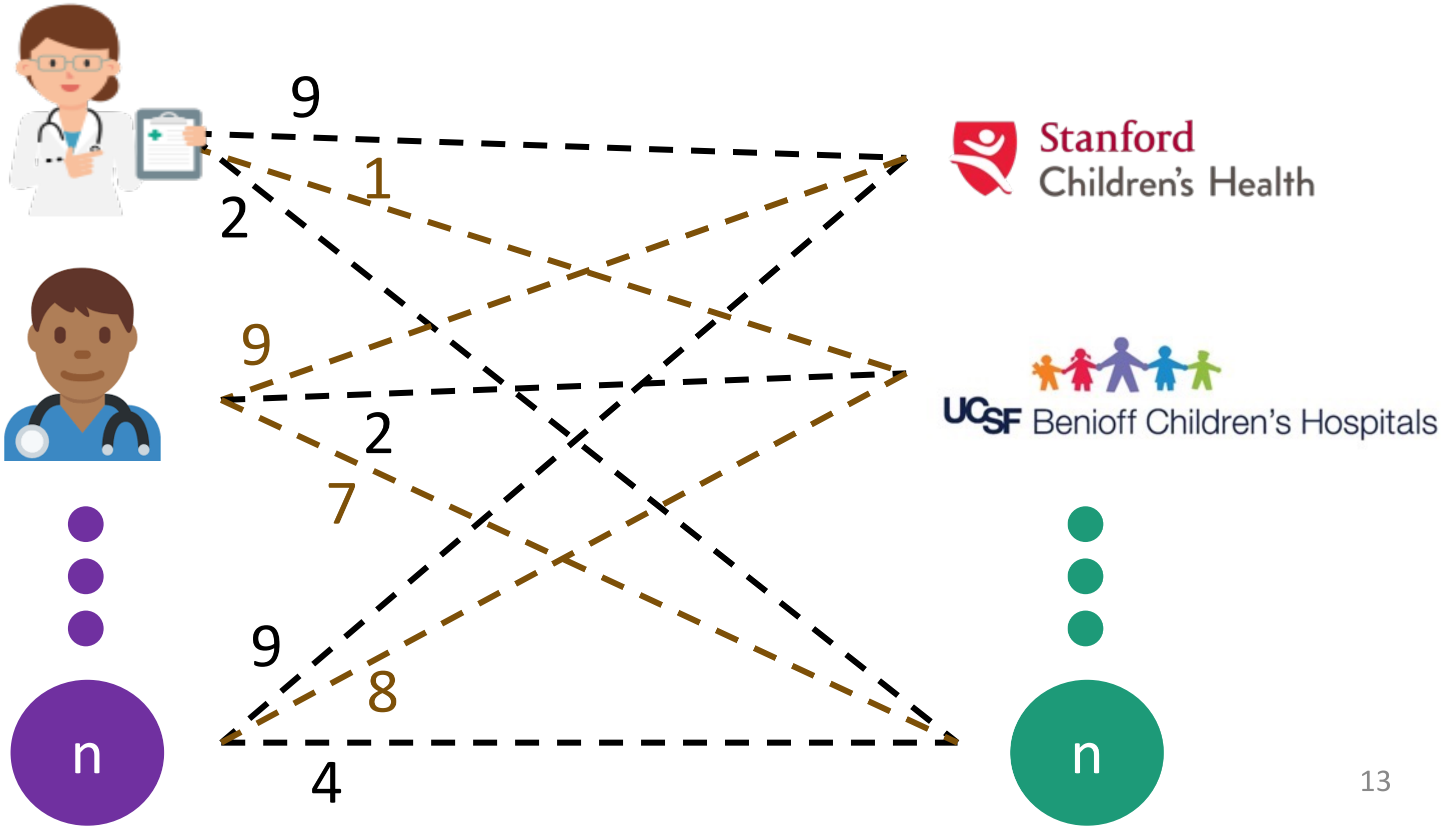


This slide just for intuition:
You don't need to know Hungarian Algorithm!

# One way to model this problem

- Bipartite graph between doctors and hospitals
- Weights on edges = some function of preferences

"Hungarian Algorithm" (CS261) finds a max weight matching

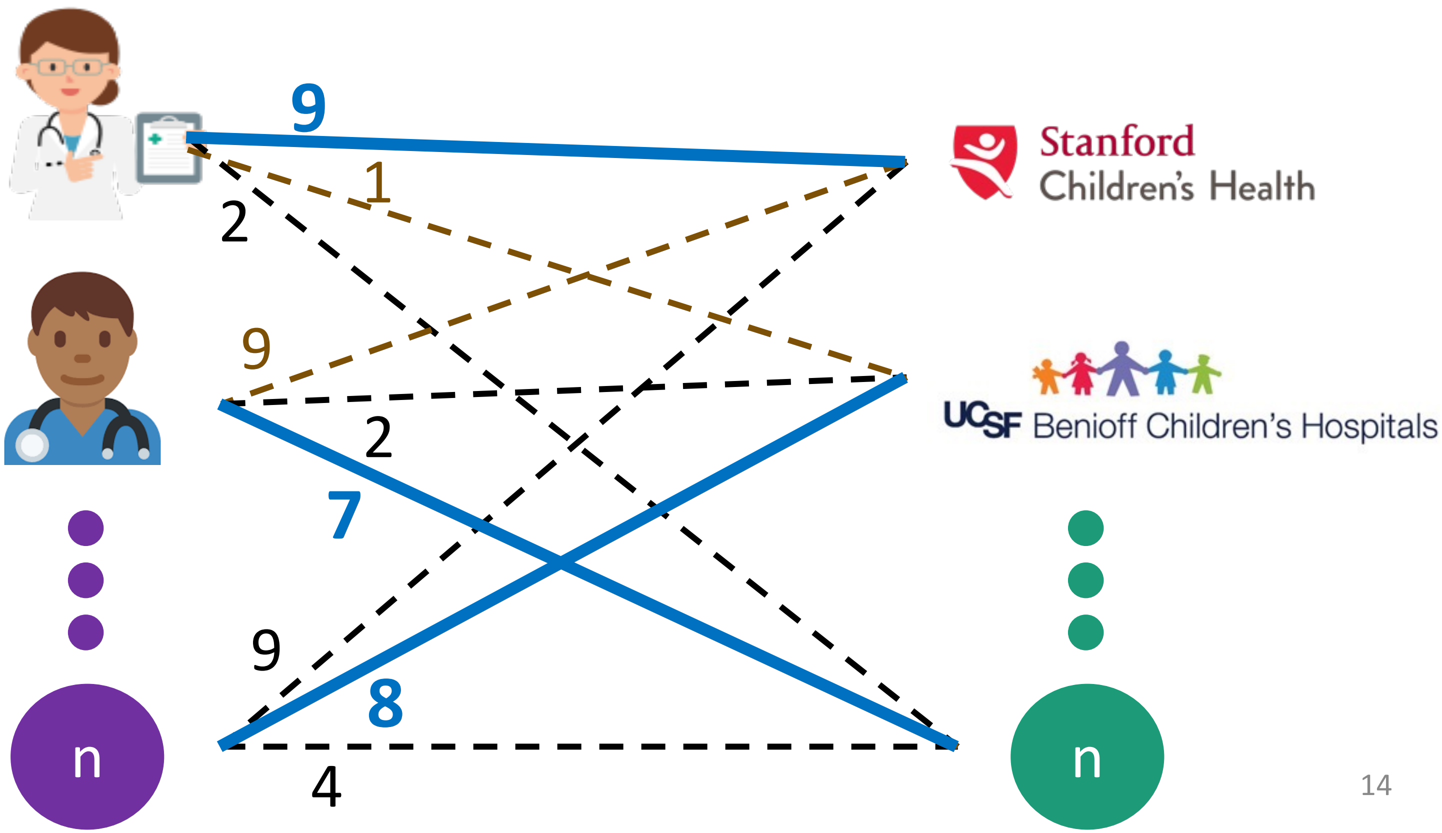


This slide just for intuition:
You don't need to know Hungarian Algorithm!

# One way to model this problem

- Bipartite graph between doctors and hospitals
- Weights on edges = some function of preferences

"Hungarian Algorithm" (CS261) finds a max weight matching



"Each hospital/doctor has a list of preferences"

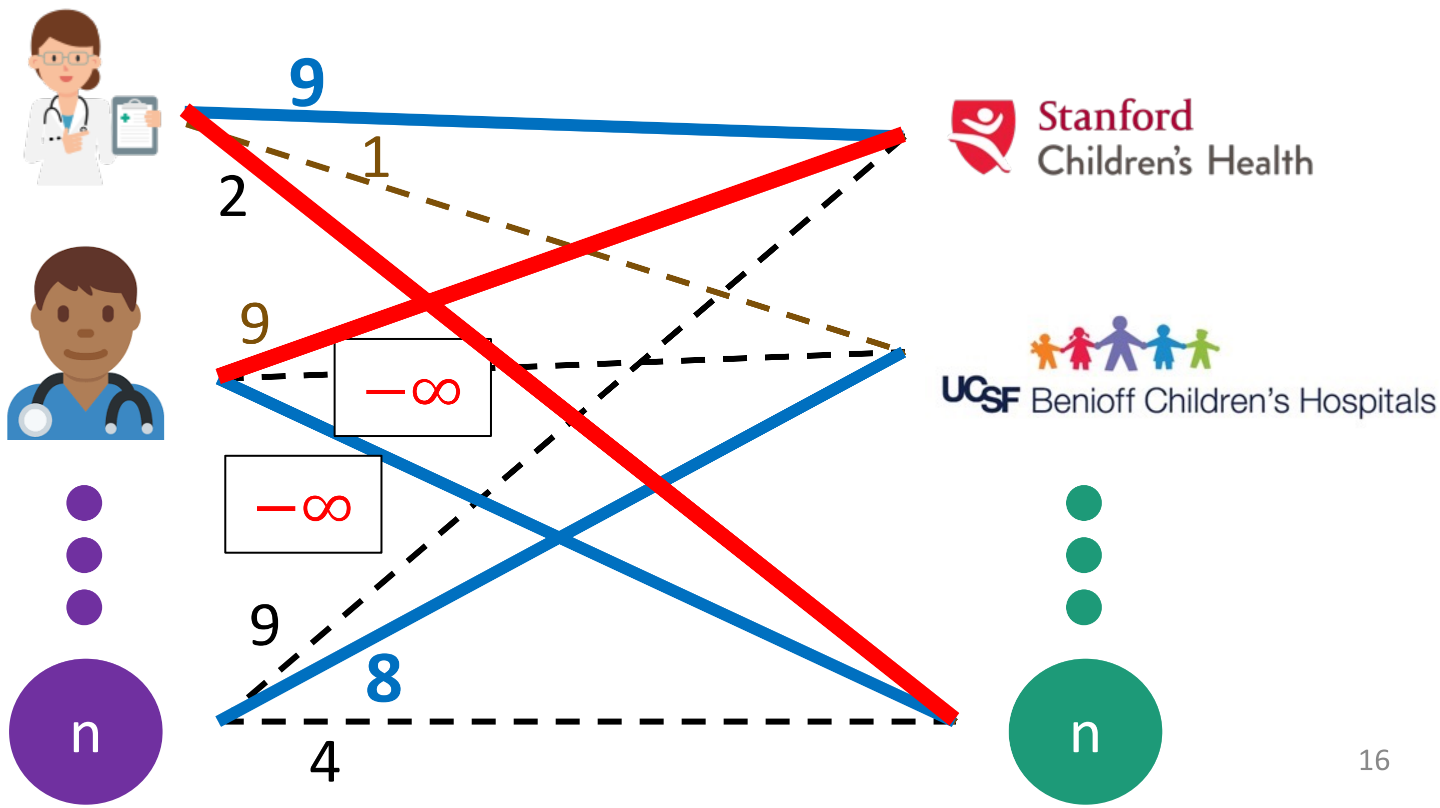
Missing step:

How does the *algorithm* get the preferences?

# Where does your input come from?

... and what can go wrong if we don't think about it carefully:

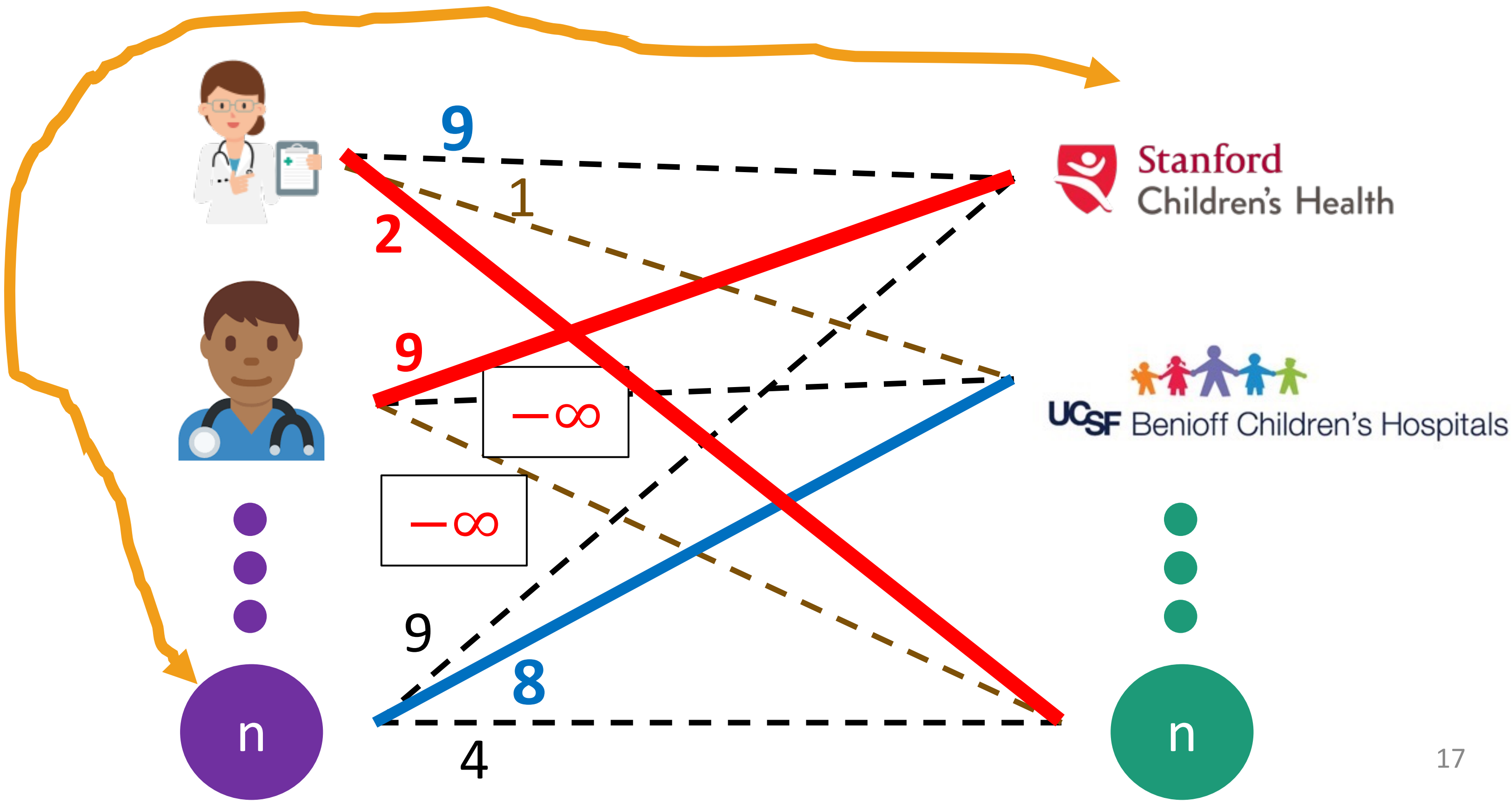
1. Some doctors may misreport their preferences



# Where does your input come from?

... and what can go wrong if we don't think about it carefully:

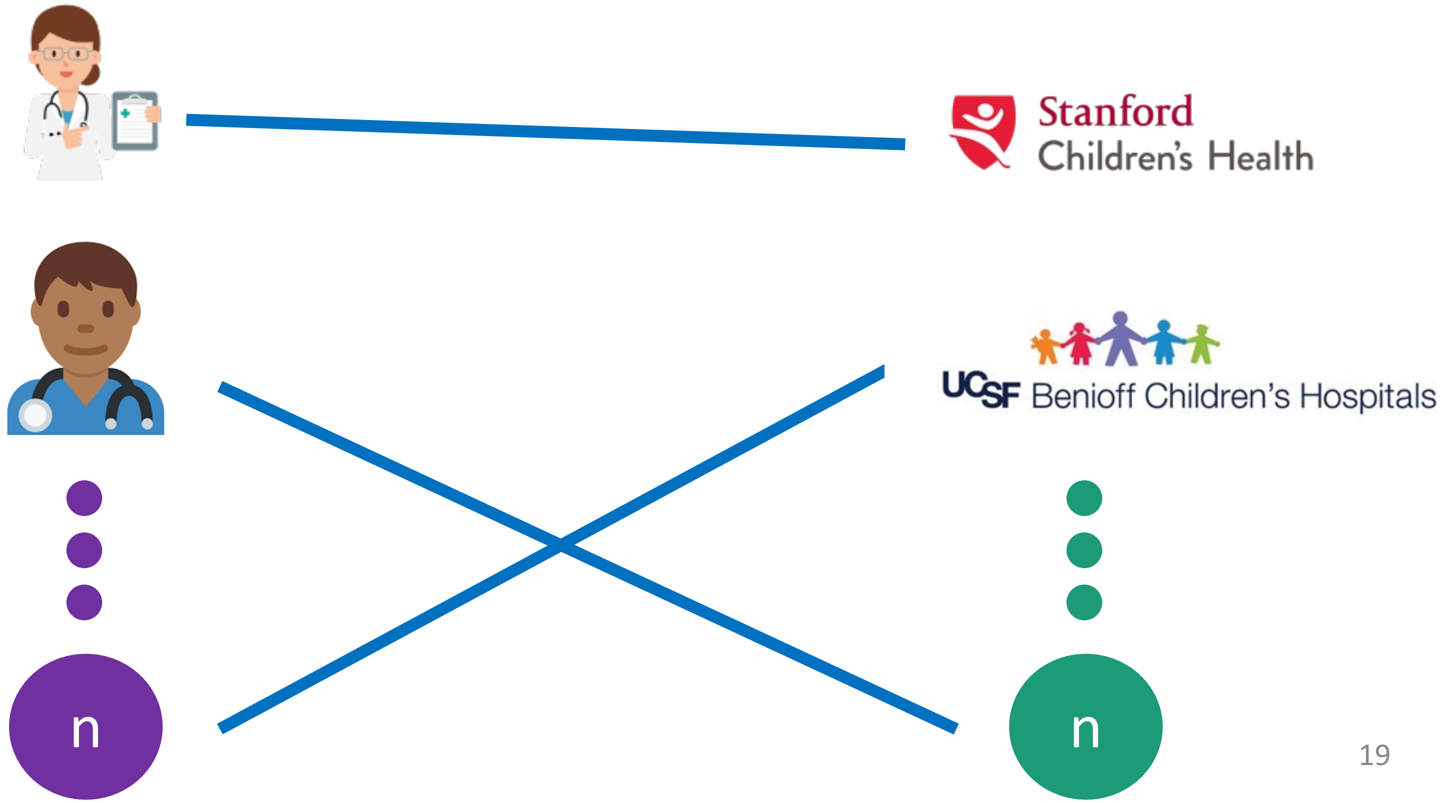
1. Some doctors may misreport their preferences
2. Some doc+hospital may match outside your algorithm



# Today

- Hospitals/residents problem
- Stable matchings
  - Solve the hospitals/residents problem
  - But can we find them?
- Deferred Acceptance Algorithm
  - Find stable matchings!
- Discussion, applications and non-applications

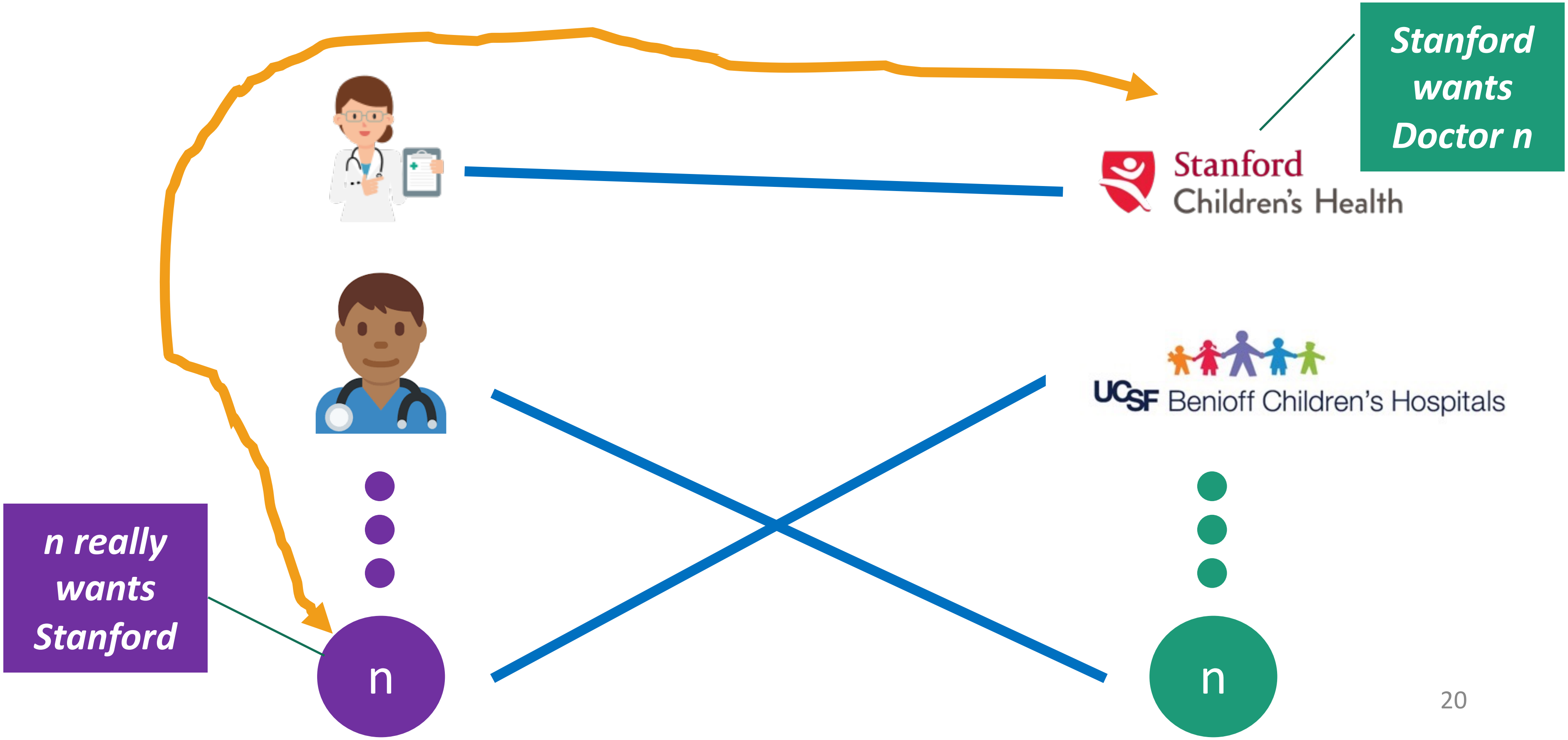
# Stable Matching



# Stable Matching

**Definition (blocking pair):**

Given Matching M, (Doctor i, Hospital j) are a ***blocking pair*** if they prefer each other to their assignment in M



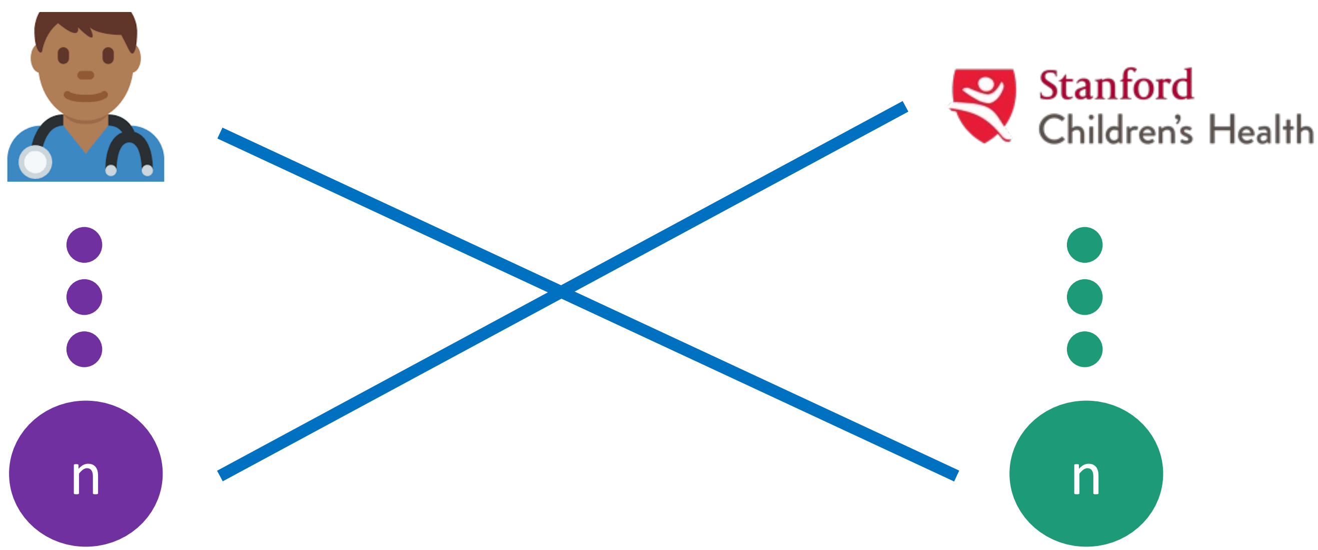
# Stable Matching

**Definition (blocking pair):**

Given Matching M, (Doctor i, Hospital j) are a ***blocking pair*** if they prefer each other to their assignment in M

**Definition (stable matching):**

M is a ***stable matching*** if there are no blocking pairs.



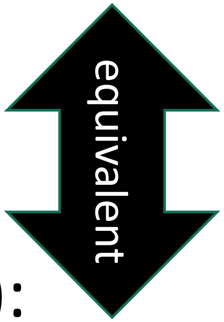
# Stable Matching

**Definition (blocking pair):**

Given Matching M, (Doctor i, Hospital j) are a ***blocking pair*** if they prefer each other to their assignment in M

**Definition (stable matching):**

M is a ***stable matching*** if there are no blocking pairs.



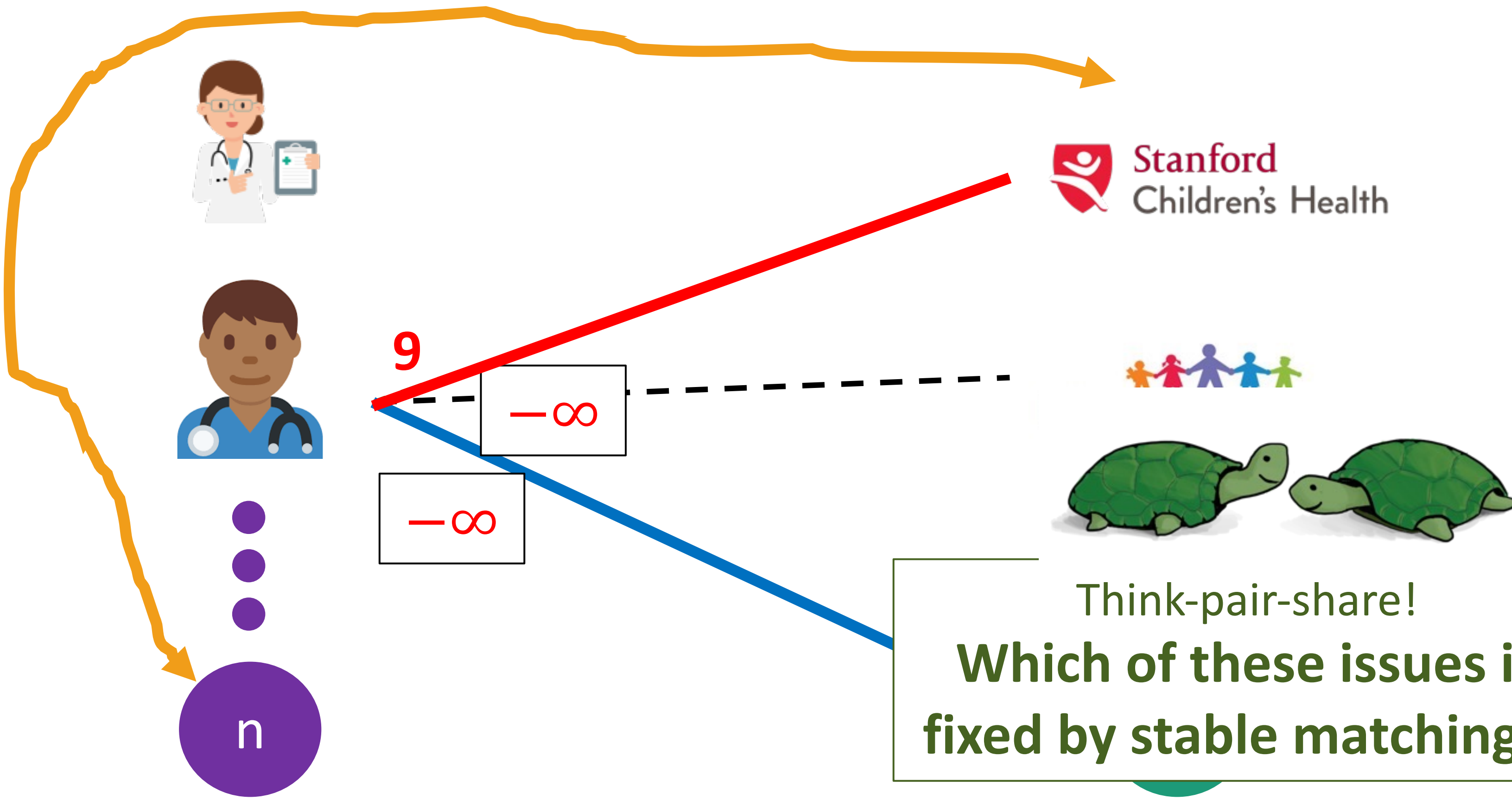

For every unmatched pair (i,j):

- Doctor i prefers Hospital M(i) over Hospital j, or;
- Hospital j prefers Doctor M(j) over Doctor i

# *Un*stable Matching and incentives

Problems we identified with unstable matchings:

1. Some doctors may misreport their preferences
2. Some doc+hospital may match outside your algorithm




Think-pair-share!

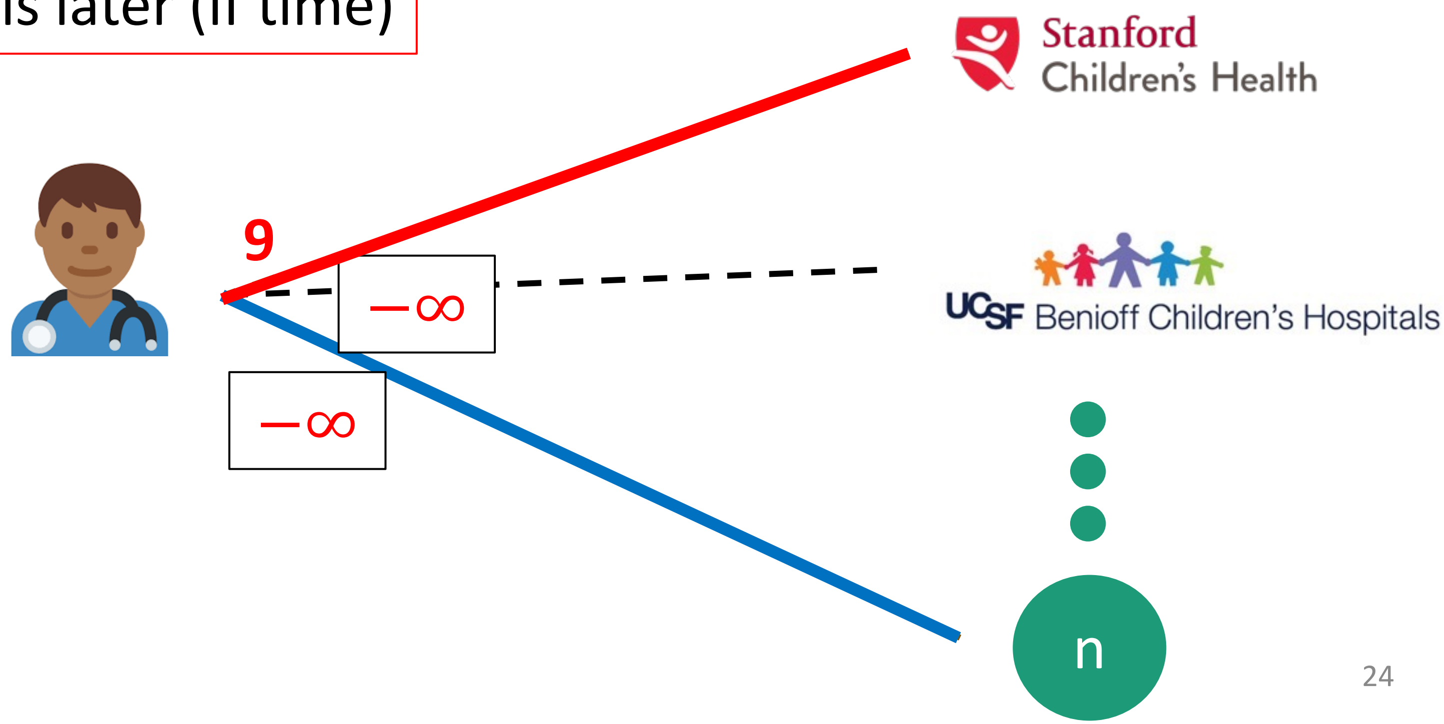
**Which of these issues is fixed by stable matchings?**

# Stable Matching and incentives

With **stable matching**:

1. Will doctors misreport their preferences?

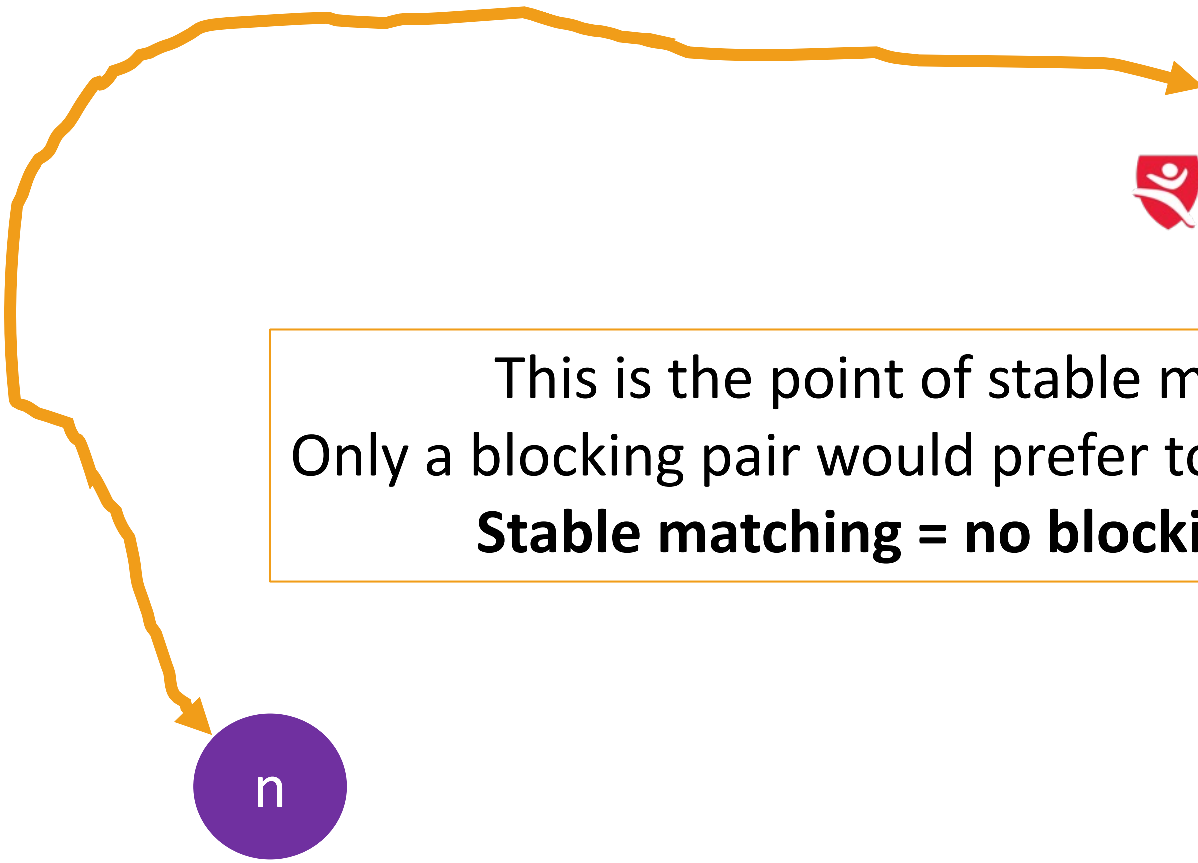
Not obvious!
We'll come back
to this later (if time)



# Stable Matching and incentives

With **stable matching**:

- Doctor+hospital ***never*** prefer to match outside algorithm!




Stanford Children's Health

This is the point of stable matching:
Only a blocking pair would prefer to match outside.
**Stable matching = no blocking pairs**!

# *Stable Matching Problem*

How to find stable matchings!
(do they even exist?)

# Stable Matching Problem

| Alice's preferences | |
|---|---|
| $1^{st}$ | Stanford |
| $2^{nd}$ | n |
| … | … |
| $n^{th}$ | UCSF |

| Stanford's preferences | |
|---|---|
| $1^{st}$ | Alice |
| $2^{nd}$ | n |
| … | … |
| $n^{th}$ | Bob |

**Stable Matching Problem**

**Input:** each doctor/hospital submits a ranking (permutation) of {1,…,n}

**Output:** a **stable matching**

**Definition (blocking pair):**

Given Matching M, (Doctor i, Hospital j) are a ***blocking pair*** if they prefer each other to their assignment in M

**Definition (stable matching):**

M is a ***stable matching*** if there are no blocking pairs.

# Naïve attempt #1

Greedy algorithm:

Step 1- match all the pairs (i,j) such that
j is i's top choice, and i is j's top choice

Step 2- hopefully recurse on the rest somehow…

- Observation: Step 1 never rules out any solution ☺

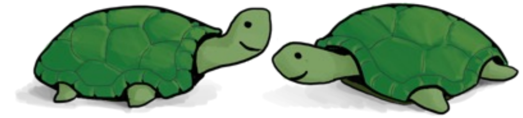
# A slightly more ambitious attempt

Greedy attempt #2:

Step 1- try to match every doctor to her favorite hospital

- Break ties by hospital preference

Step 2- hopefully recurse on the rest somehow…



# A slightly more an

Think-pair-share!

**Matching (C,y) was a bad idea…**

**How could we avoid it?**

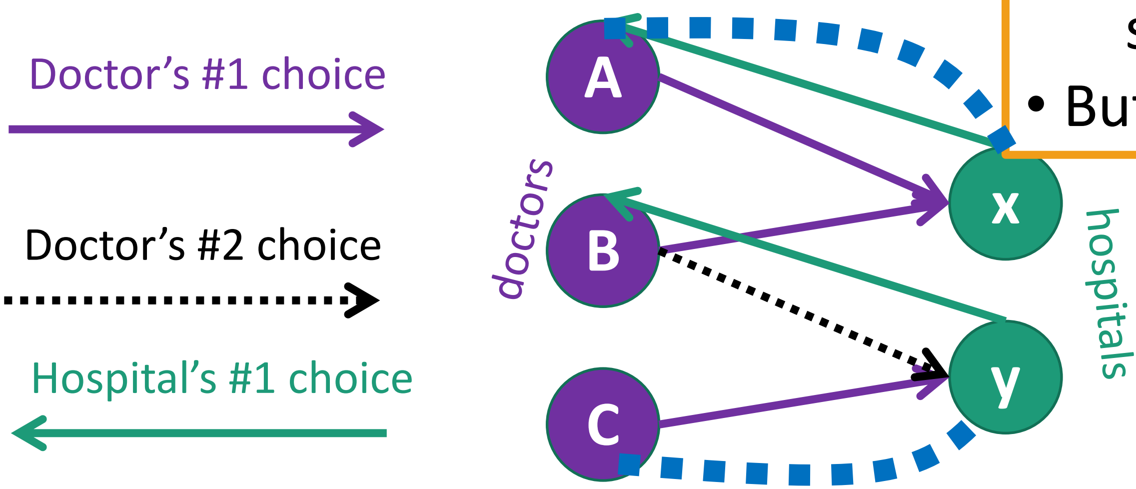
Greedy attempt #2:

Step 1- try to match every doctor to her favorite hospital

- Break ties by hospital preference

We're already wrong!

- Step 1: A,B want x, C wants y so we **match** (A,x) and (C,y)
- But now (B,y) is **blocking**!



# Today

- Hospitals/residents problem

- Stable matchings
  - Solve the hospitals/residents problem
  - But can we find them?

- Deferred Acceptance Algorithm
  - Find stable matchings!

- Discussion, applications and non-applications

# Questions?

**Definition (blocking pair):**

Given Matching M, (Doctor i, Hospital j) are a ***blocking pair*** if they prefer each other to their assignment in M

**Definition (stable matching):**

M is a ***stable matching*** if there are no blocking pairs.

⇕ equivalent

For every unmatched pair (i,j):

- Doctor i prefers Hospital M(i) over Hospital j, or;
- Hospital j prefers Doctor M(j) over Doctor i

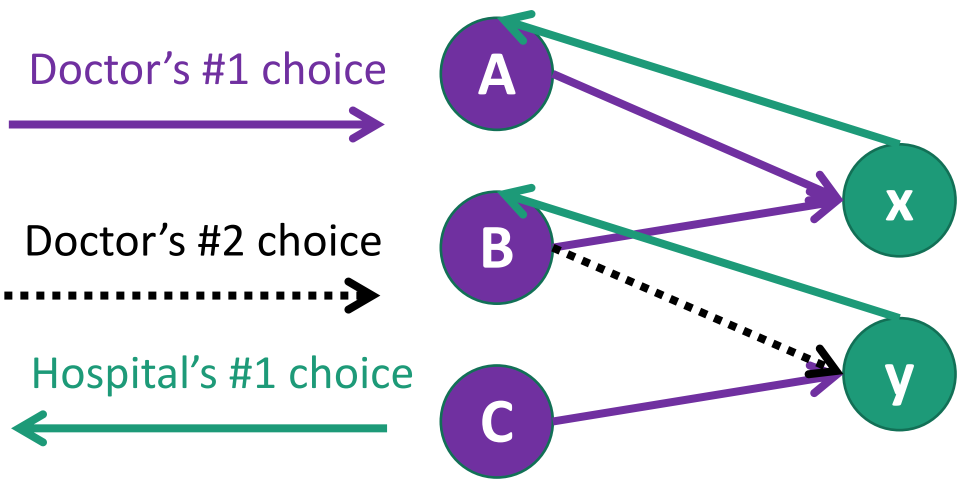
# Deferred Acceptance Algorithm

[Gale Shapley ‘62] -> 2012 Nobel Prize* in Econ!

*- Joint w/ Al Roth from Stanford

# Deferred Acceptance Algorithm

Main idea: ***try*** to match each doctor to top choice;
if you discover a blocking pair, just switch the matching!




The issue was:
A,B want x, C wants y
we tried to match (A,x) and (C,y)
but then (B,y) was **blocking**!

# Deferred Acceptance Algorithm

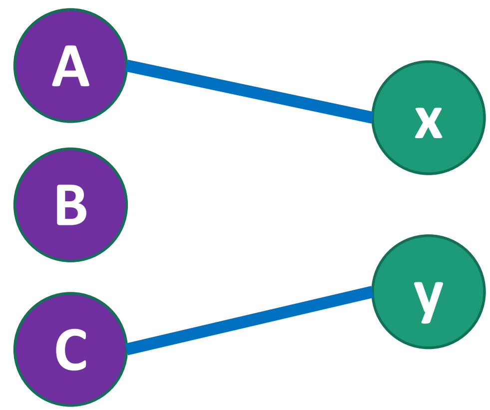
Main idea: ***try*** to match each doctor to top choice;

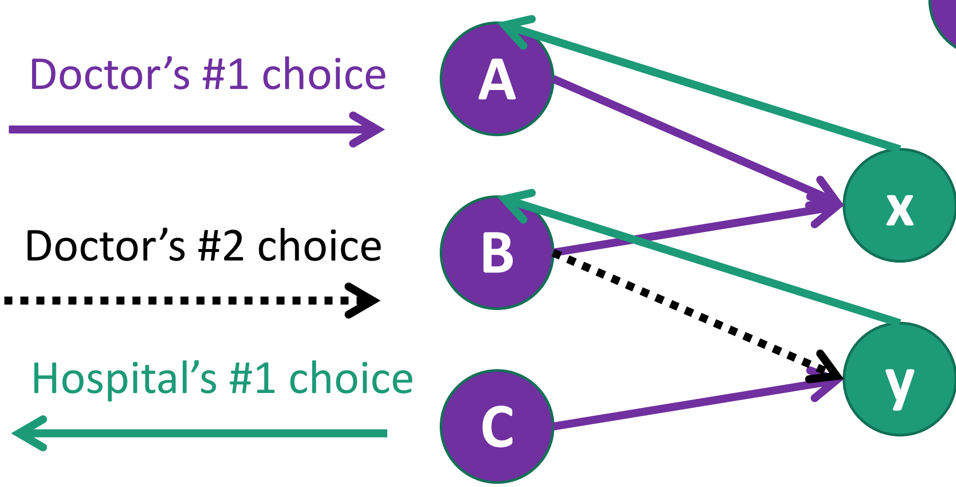
if you discover a blocking pair, just switch the matching!

Algorithm iteration 1:

A, B want x; C wants y

So we match (A,x) and (C,y)




The issue was:

A,B want x, C wants y

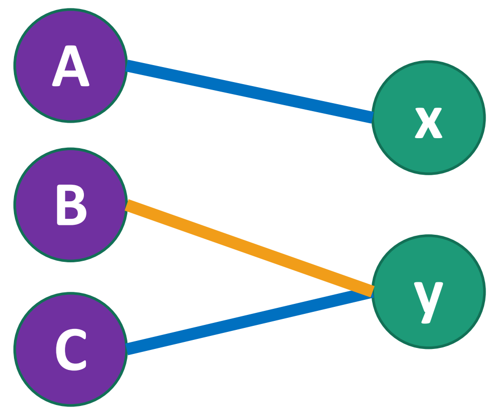
we tried to match (A,x) and (C,y)

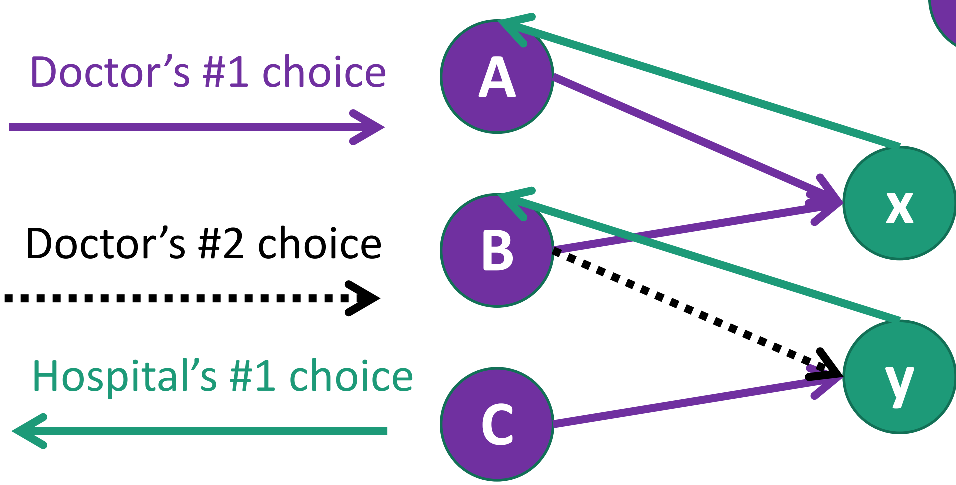
but then (B,y) was **blocking**!

# Deferred Acceptance Algorithm

Main idea: ***try*** to match each doctor to top choice;
if you discover a blocking pair, just switch the matching!

Algorithm iteration 2(a):
Now notice that (B,y) is **blocking** ☹




The issue was:
A,B want x, C wants y
we tried to match (A,x) and (C,y)
but then (B,y) was **blocking**!

# Deferred Acceptance Algorithm

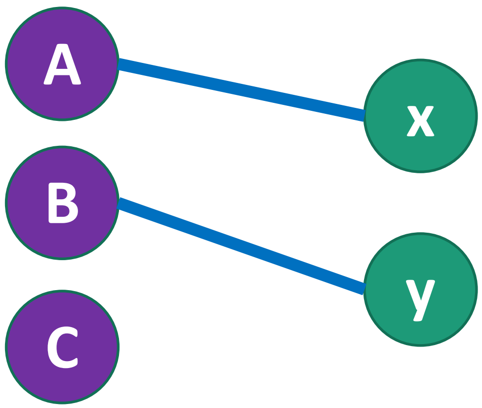
Main idea: ***try*** to match each doctor to top choice;

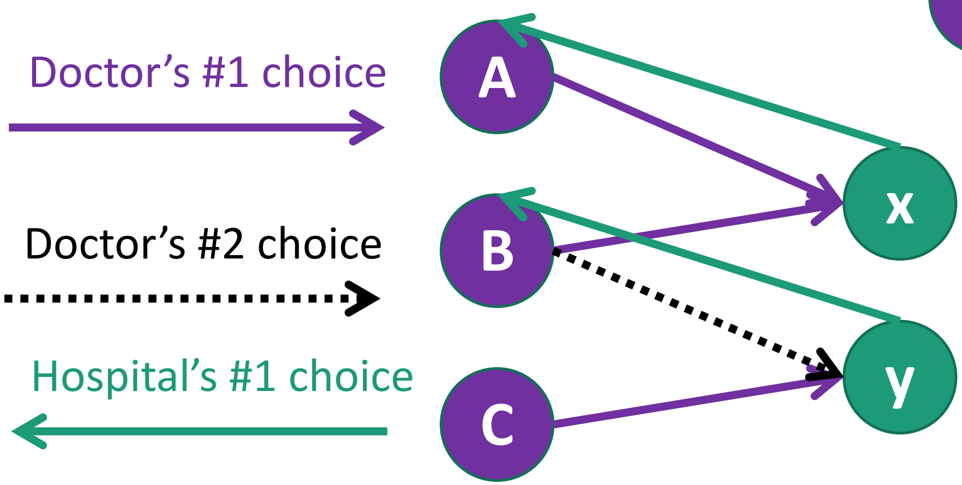
if you discover a blocking pair, just switch the matching!

Algorithm iteration 2(b):

Add (B,y) to the matching ☺

(and remove (C,y))




The issue was:

A,B want x, C wants y

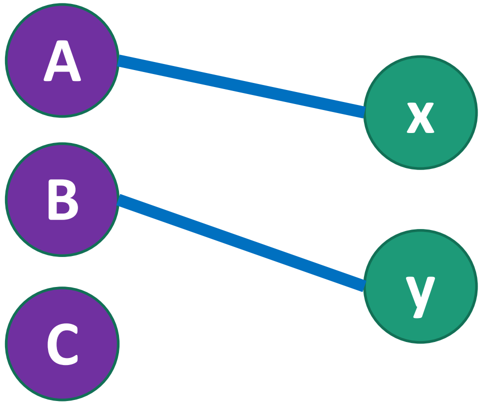
we tried to match (A,x) and (C,y)

but then (B,y) was **blocking**!

# Deferred Acceptance Algorithm

Main idea: ***try*** to match each doctor to top choice;
if you discover a blocking pair, just switch the matching!

Algorithm iteration 2(b):
Add (B,y) to the matching ☺
(and remove (C,y))




Don't worry
Just switch around
until no blocking pairs!

Lucky the Lackadaisical Lemur

The issue was:
A,B want x, C wants y
we tried to match (A,x) and (C,y)
but then (B,y) was **blocking**!

# Deferred Acceptance Algorithm

Main idea: ***try*** to match each doctor to top choice;
if you discover a blocking pair, just switch the matching!

<u>Almost-pseudo-code:</u>

**While** there is an unmatched doctor **i**:
    Try to match **i** to next-favorite hospital on her list;

    **If** this hospital doesn't have a doctor yet:
        Both Doctor **i** and hospital are happy with this new match ☺

    **Else-if** this hospital prefers its current match **i'** over **i**:
        Doctor **i** remains unmatched

    **Else-if** this hospital prefers **i** over **i'**:
        Unmatch **i'**; Match (**i**, hospital)

# Example run-through

# DA Example Run 1


Alice

X, Y, Z


Bob

Y, X, Z


Charlie

Y, Z, X

B, A, C

X

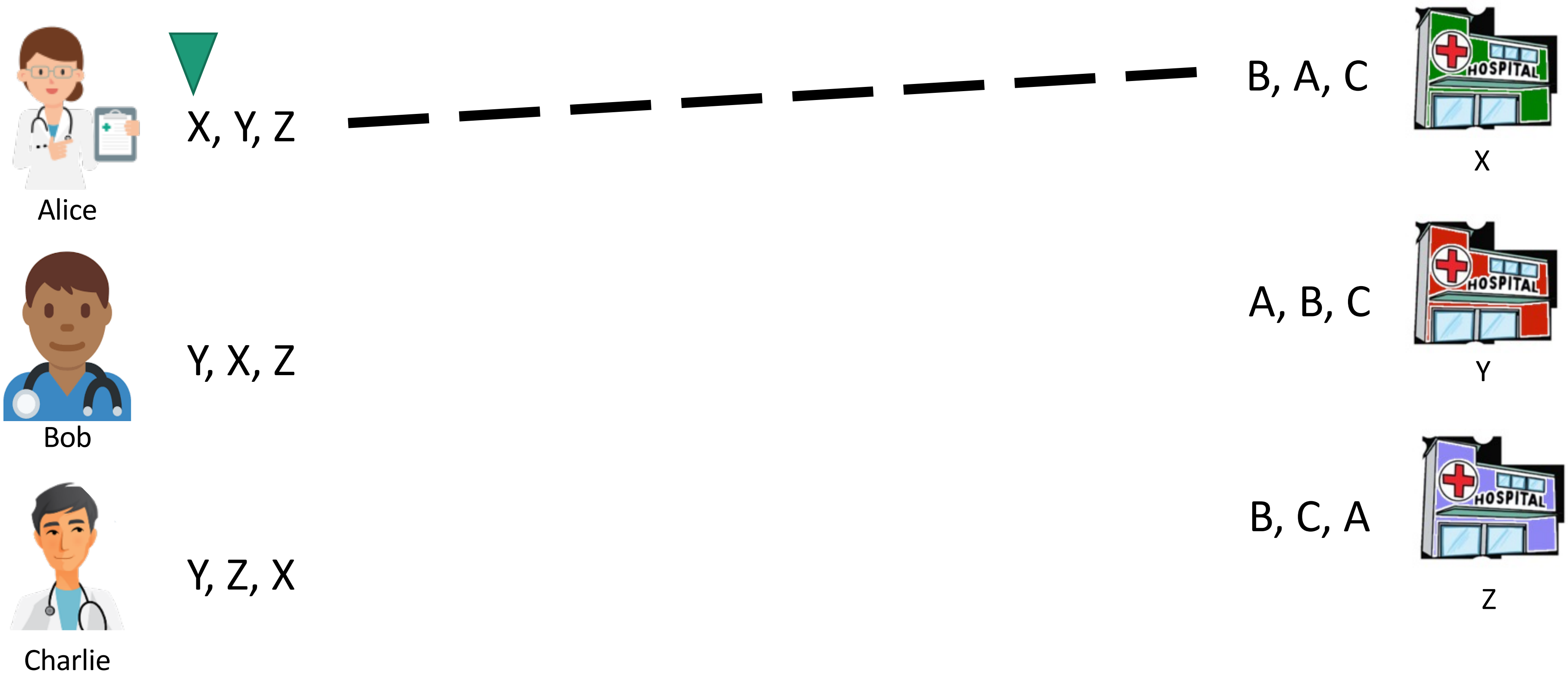
A, B, C

Y

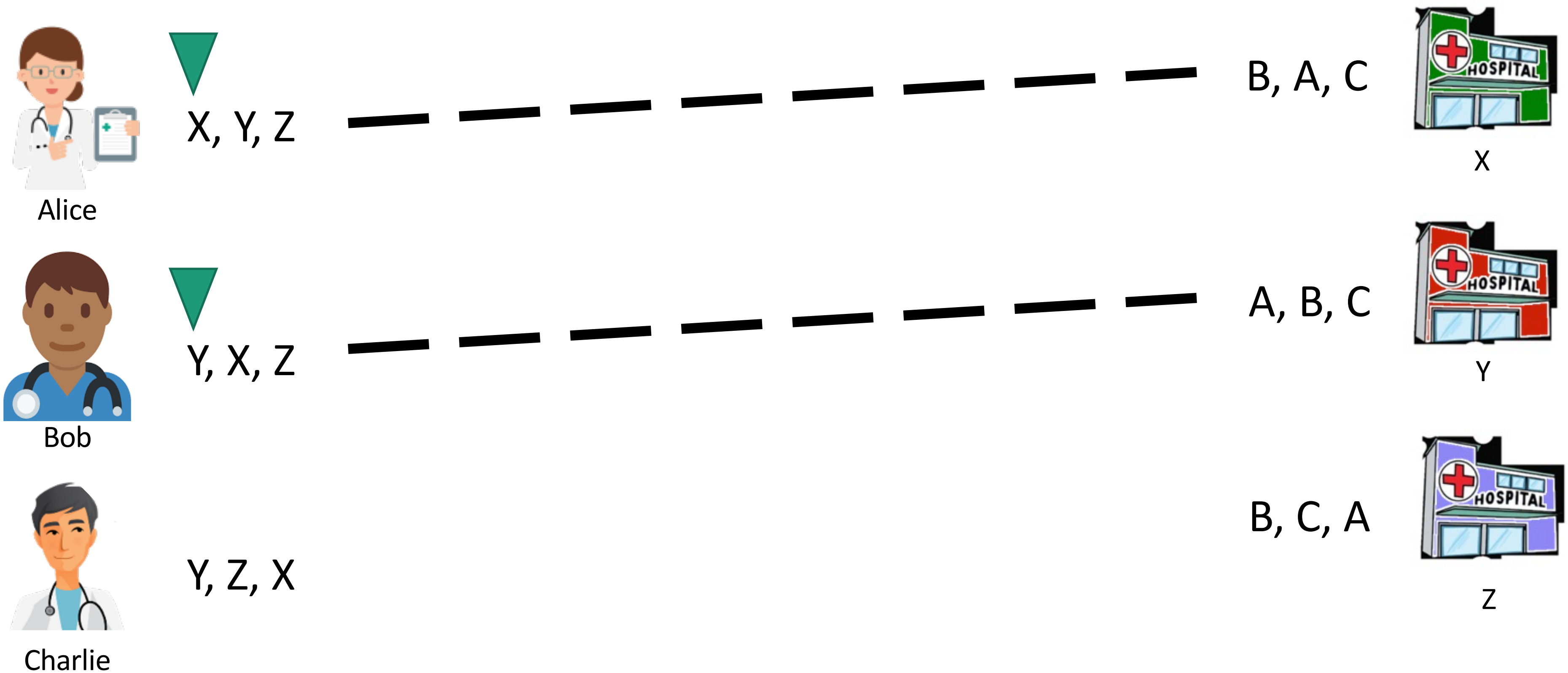
B, C, A

Z

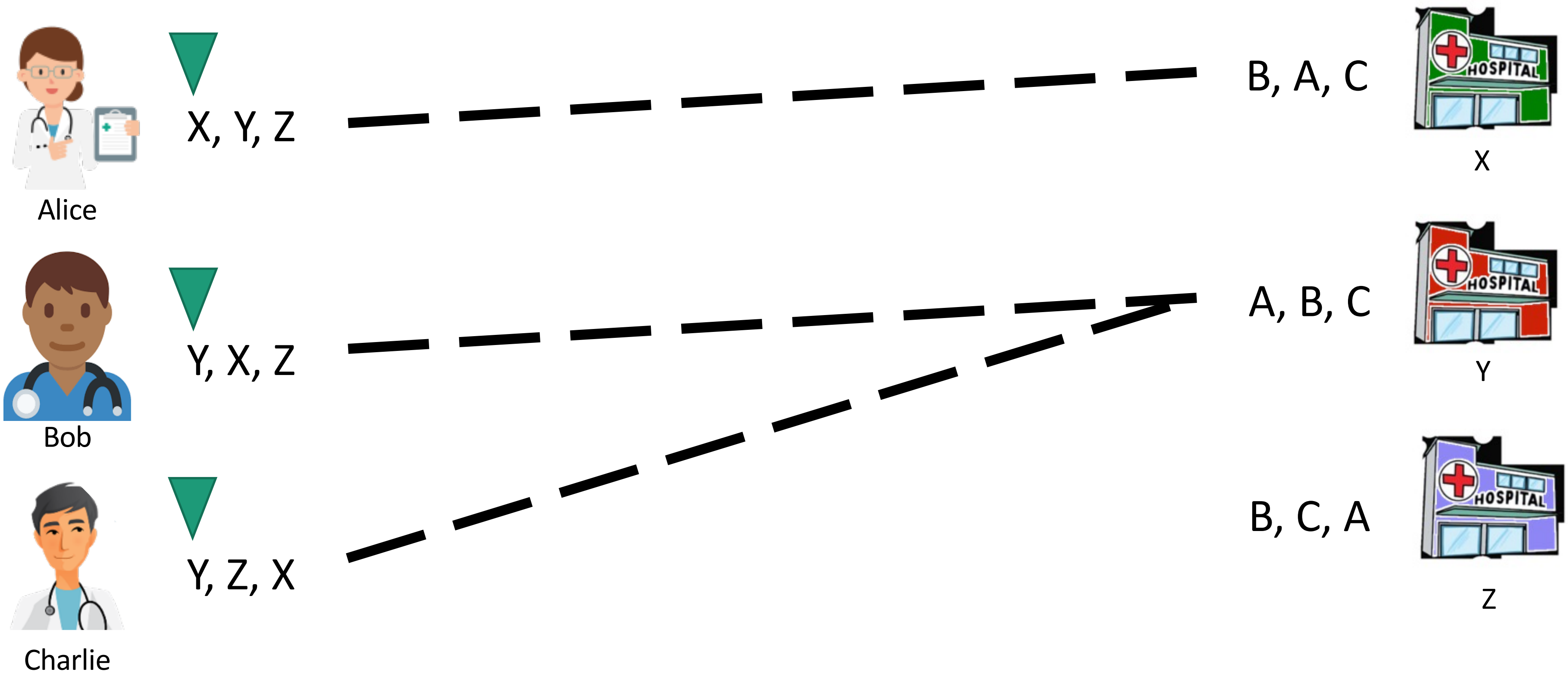
# DA Example Run 1



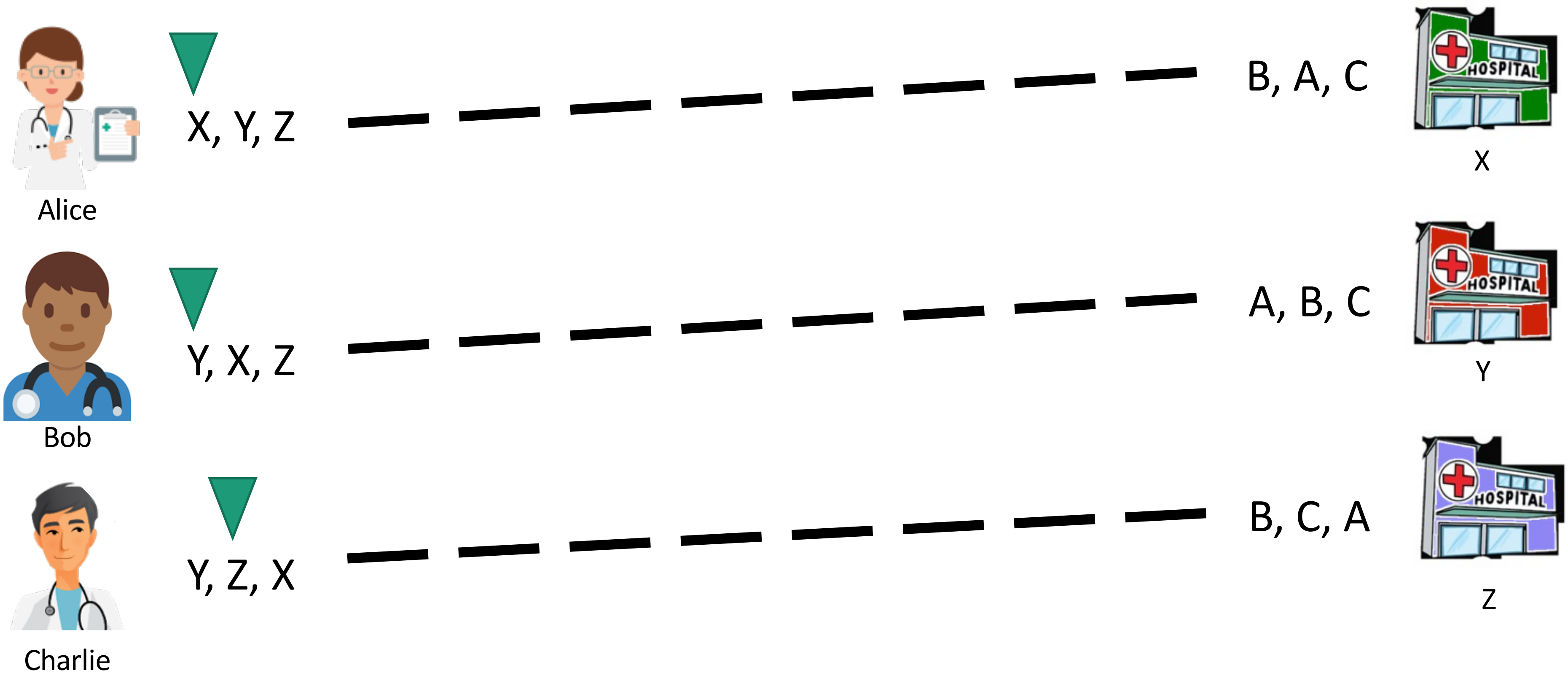
# DA Example Run 1



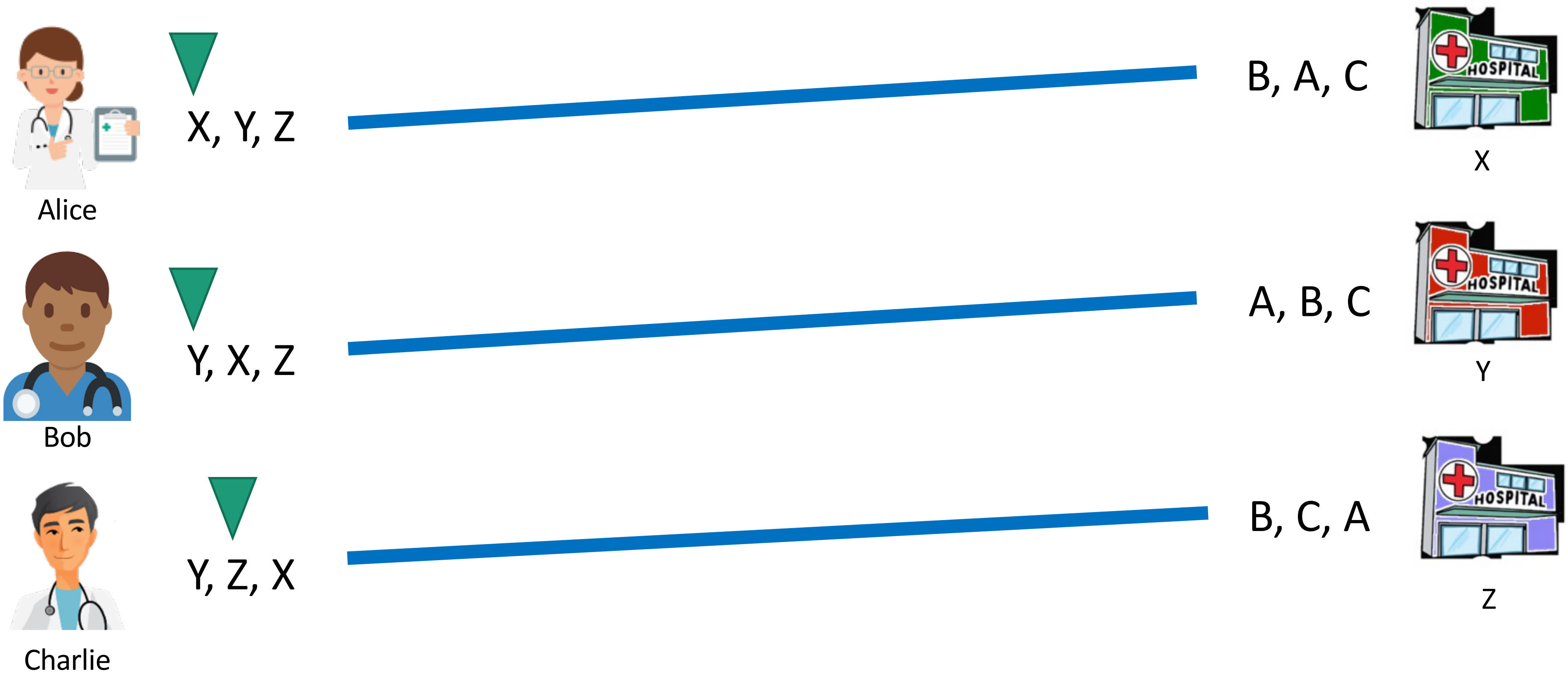
# DA Example Run 1



# DA Example Run 1



# DA Example Run 1



# Another example

# DA Example Run 2

Alice

X, Y, Z

Bob

Y, X, Z

Charlie

Y, Z, X

B, A, C


X

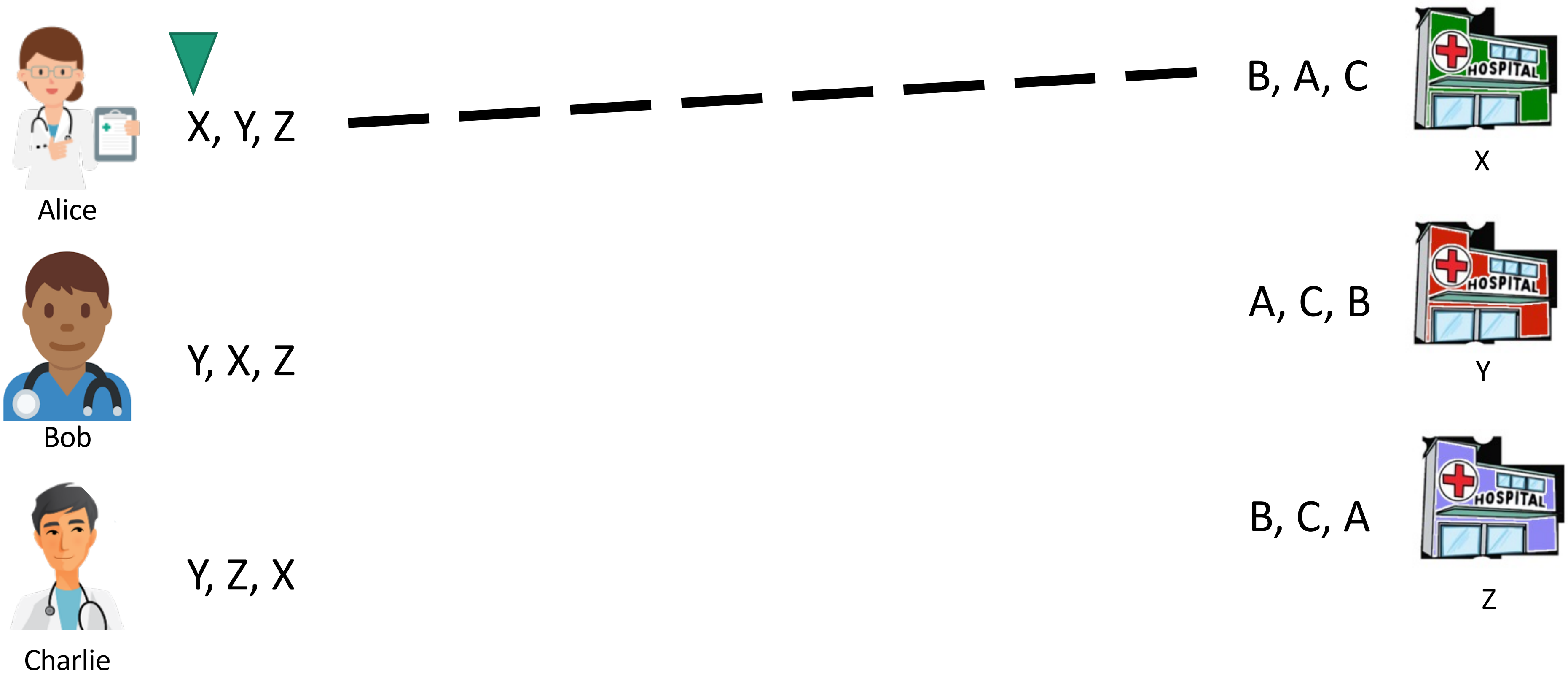
A, C, B


Y

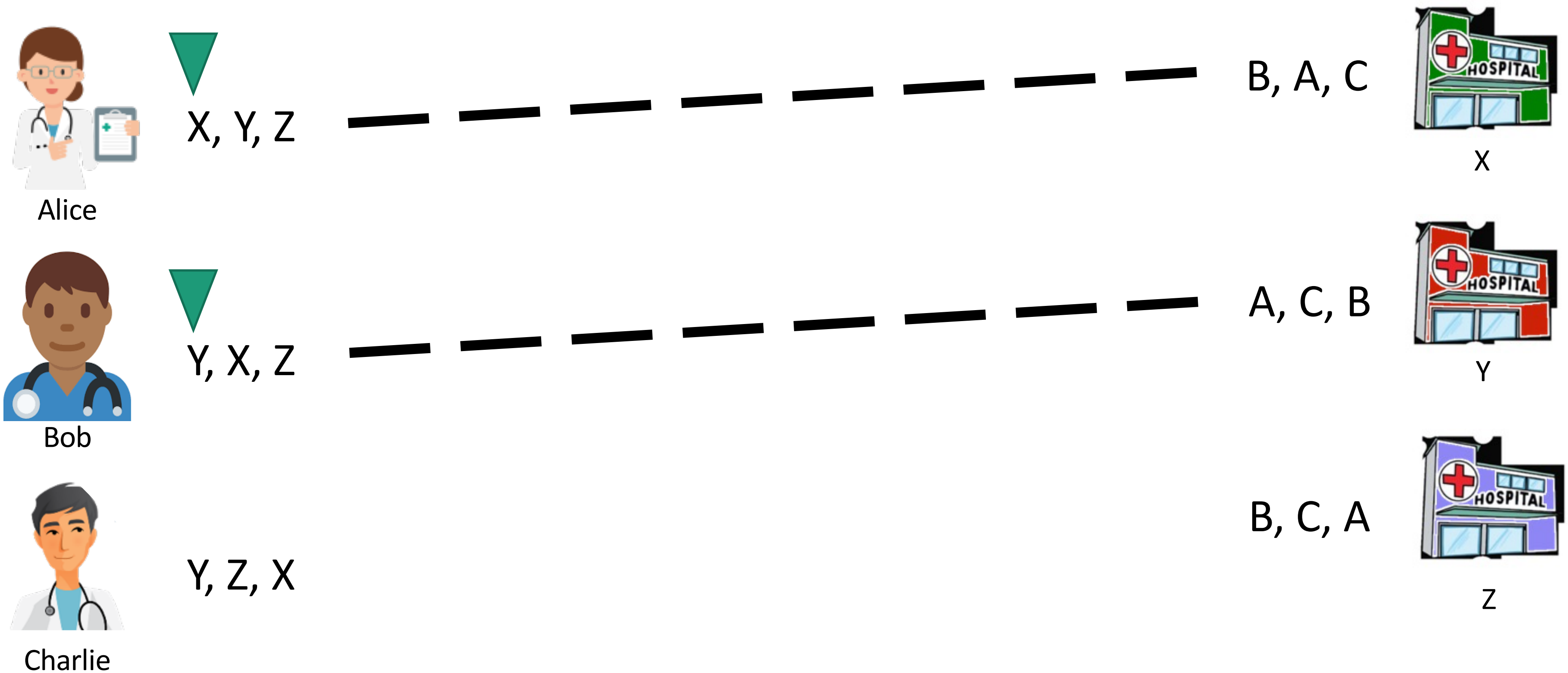
B, C, A


Z

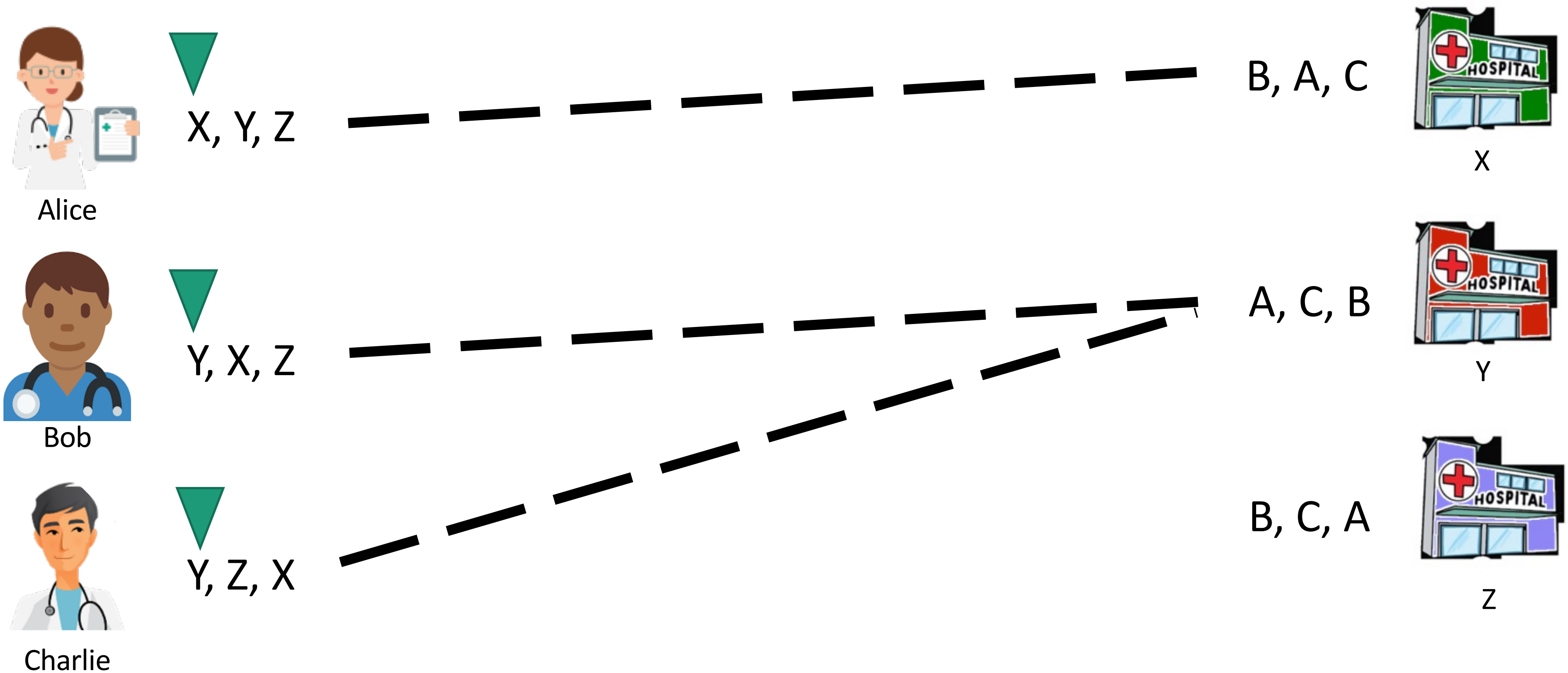
# DA Example Run 2



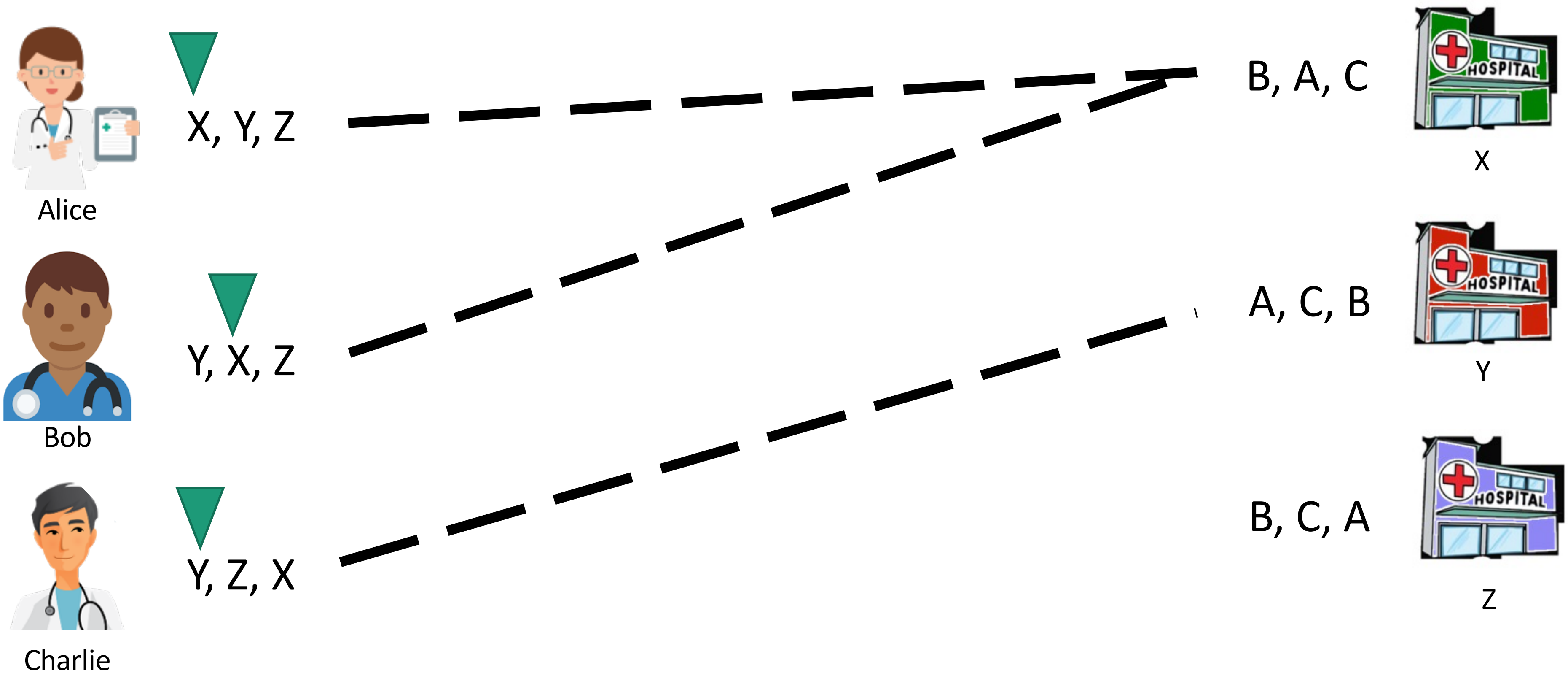
# DA Example Run 2



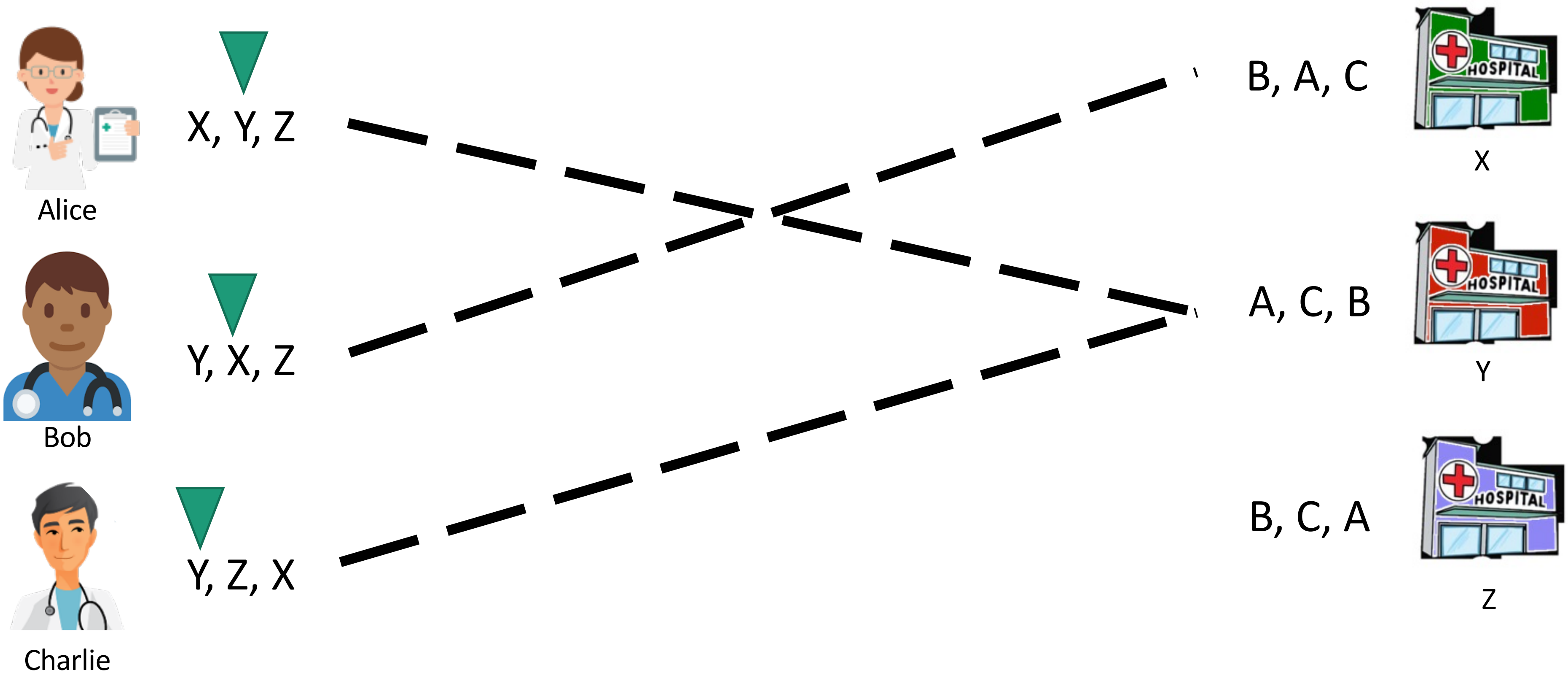
# DA Example Run 2



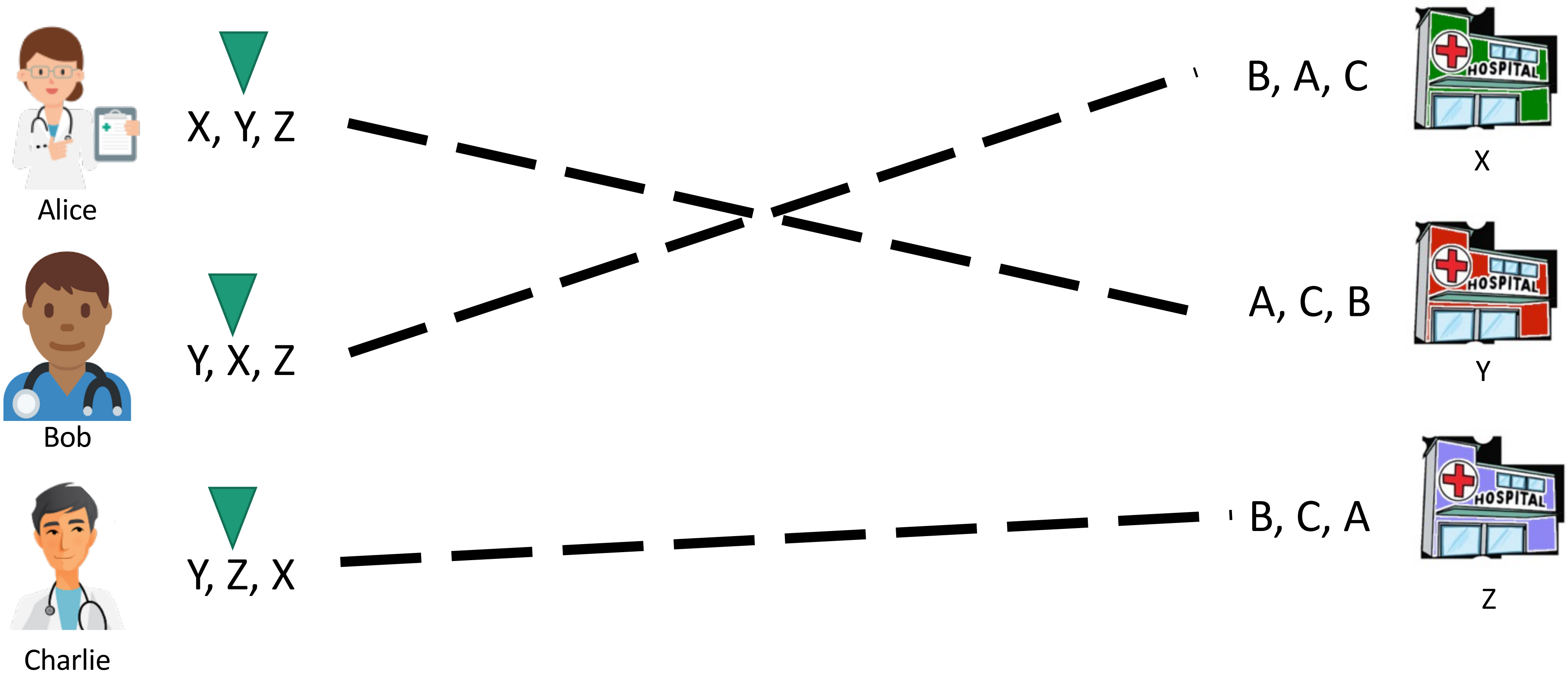
# DA Example Run 2



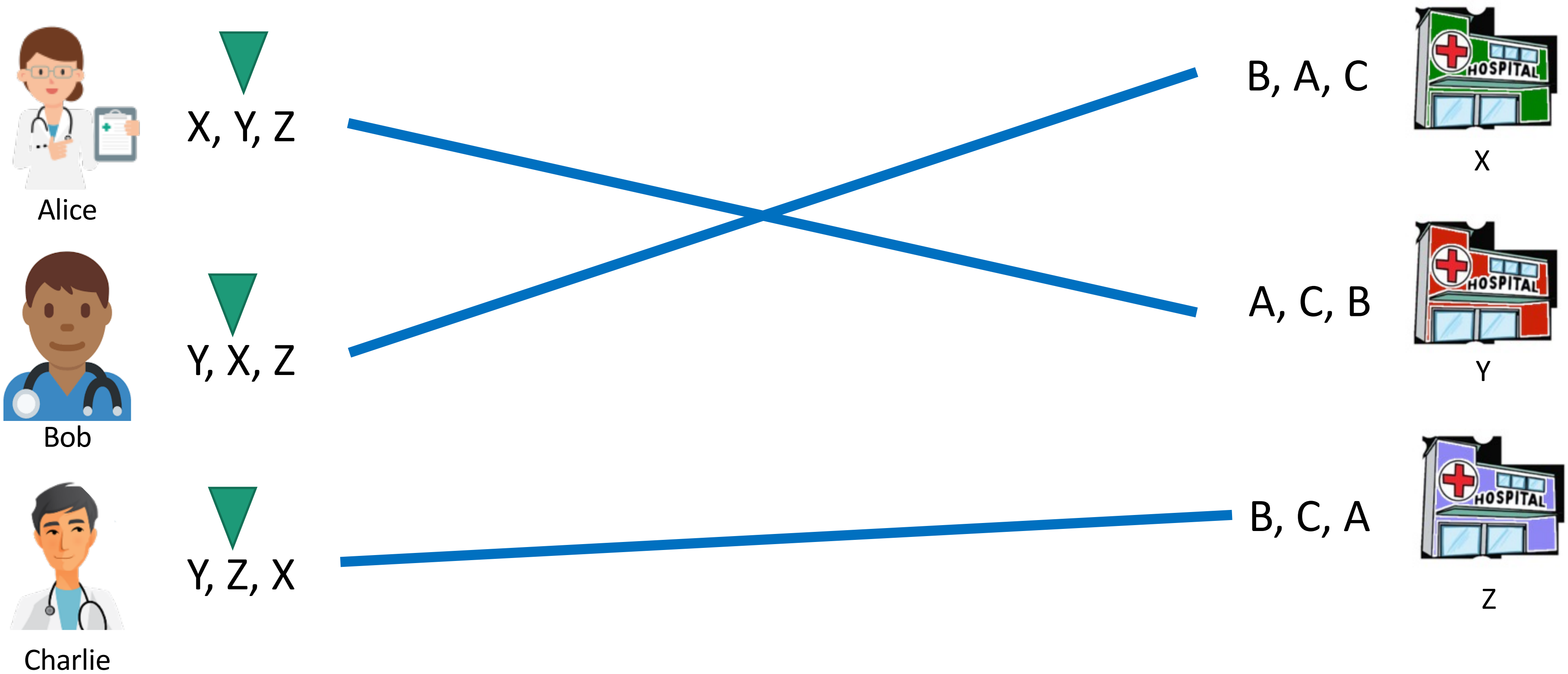
# DA Example Run 2



# DA Example Run 2



# DA Example Run 2



# Deferred Acceptance Algorithm

**Deferred-Acceptance(Doctors,Hospitals):**

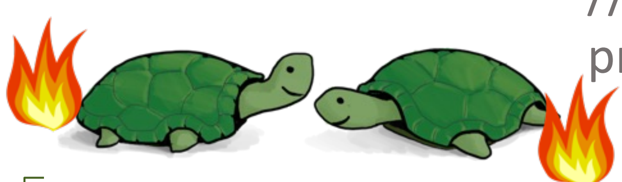
```
// initialize
freeDoctors ← Doctors
for all d in Doctors:
    d.current ← 0
for all h in Hospitals:
    h.doctor ← NIL
```

```
// main loop
while (exists d in freeDoctors)
    h ← d.ranking[d.current++]      // h is d's next favorite

    if (h is free)
        h.doctor ← d
        remove d from freeDoctors

    else-if (h.rank[d] < h.rank[h.doctor])      // h prefers d to previous match
        add h.doctor to freeDoctors
        h.doctor ← d
        remove d from freeDoctors

return (h,h.doctor) for all h in Hospitals
```



Think-pair-share!
**Running time?**

# Deferred Acceptance Algorithm

**Running time:**
Each iteration of while loop = O(1)

Each iteration:
We +1 d.current for some doctor

We always have:
d.current $\leq n$
for every doctor (There are $n$ doctors…)

Therefore, total run-time = $O(n^2)$

```
        // main loop
while (exists d in freeDoctors)
    h ← d.ranking[d.current++]  // h is d's next favorite

    if (h is free)
        h.doctor ← d
        remove d from freeDoctors

    else-if (h.rank[d] < h.rank[h.doctor])
        add h.doctor to freeDoctors
        h.doctor ← d
        remove d from freeDoctors

return (h,h.doctor) for all h in Hospitals
```

# DA algorithm

- Does it work?
  - Yes!

- Is it fast?
  - $O(n^2)$ - this is linear in the input size!

  At worst exhaust through every doctor's preference list

# Deferred Acceptance works!

**Theorem:** Given n doctors and n hospitals,
DA algorithm outputs a complete stable matching.

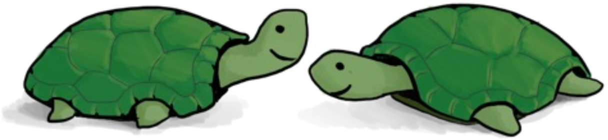
**Corollary:** A stable matching exists.

(This is not obvious!)

# Proof of Theorem

**Theorem:** Given n doctors and n hospitals,
DA algorithm outputs a complete stable matching.

**Proof:** Follows from Claims 1+3 below…



Think-pair-share:
**Prove these!**

**Claim 1:** At every iteration, current match is stable w.r.t. non-free doctors and hospitals.

**Claim 2:** Once a hospital is matched, it remains matched (possibly to a different doctor) until end of algorithm.

**Claim 3:** At the end of algorithm, every doctor/hospital is matched.

# Deferred Acceptance Algorithm

Main idea: ***try*** to match each doctor to top choice;
if you discover a blocking pair, just switch the matching!

<u>Almost-pseudo-code:</u>

**While** there is an unmatched doctor **i**:

Try to match **i** to next-favorite hospital on her list;

**If** this hospital doesn't have a doctor yet:

Both Doctor **i** and hospital are happy with this new match ☺

**Else-if** this hospital prefers its current match **i'** over **i**:

Doctor **i** remains unmatched

**Else-if** this hospital prefers **i** over **i'**:

Unmatch **i'**; Match (**i**, hospital)

# Proof of claims

**Claim 1:** At every iteration, current match is stable w.r.t. non-free doctors and hospitals.

**Proof by contradiction:** Suppose (*d*,*h*) blocking pair.

→ *d* is currently matched to worse hospital than *h*.

→ *d* already tried to match to *h*.

→ *h* either refused *d* or left *d* later. Why?

→ *h* must be matched to better doctor than *d* – contradiction!

**Claim 2:** Once a hospital is matched, it remains matched (possibly to a different doctor) until end of algorithm.

"**Proof**": obvious from algorithm

**Claim 3:** At the end of algorithm, every doctor/hospital is matched.

**Proof by contradiction:** Suppose (*d*,*h*) still free.

End of algorithm → *d* already tried to match to *h*.

→ after that step, *h* wasn't free → by Claim 2, contradiction!

# Deferred Acceptance works!

**Theorem:** Given n doctors and n hospitals,
DA algorithm outputs a complete stable matching.

**Corollary:** A stable matching exists.

**Claim 1:** At every iteration, current match is stable
w.r.t. non-free doctors and hospitals.

**Claim 2:** Once a hospital is matched, it remains matched
(possibly to a different doctor) until end of algorithm.

**Claim 3:** At the end of algorithm, every doctor/hospital is matched.

# What have we learned?

**Blocking Pair:** A doctor and hospital that prefer each other over their respective matches.

**Stable Matching:** A matching without blocking pairs!

**Deferred Acceptance Algorithm**

*"Tentatively match each free doctor to best interested hospital. Allow the hospital to leave match when a better doctor arrives."*

Runs in time $O(n^2)$ = linear in input size ☺

# Today

- Hospitals/residents problem

- Stable matchings
  - Solve the hospitals/residents problem
  - But can we find them?

- Deferred Acceptance Algorithm
  - Find stable matchings!

- Discussion, applications and non-applications

# The optimal stable matching?

DA algorithm found *a* stable matching…

- Is it *optimal*?
- What does optimality mean?

Theorem: The matching returned by DA is **doctor-optimal**, i.e. every doctor is matched to favorite hospital possible in any stable matching.

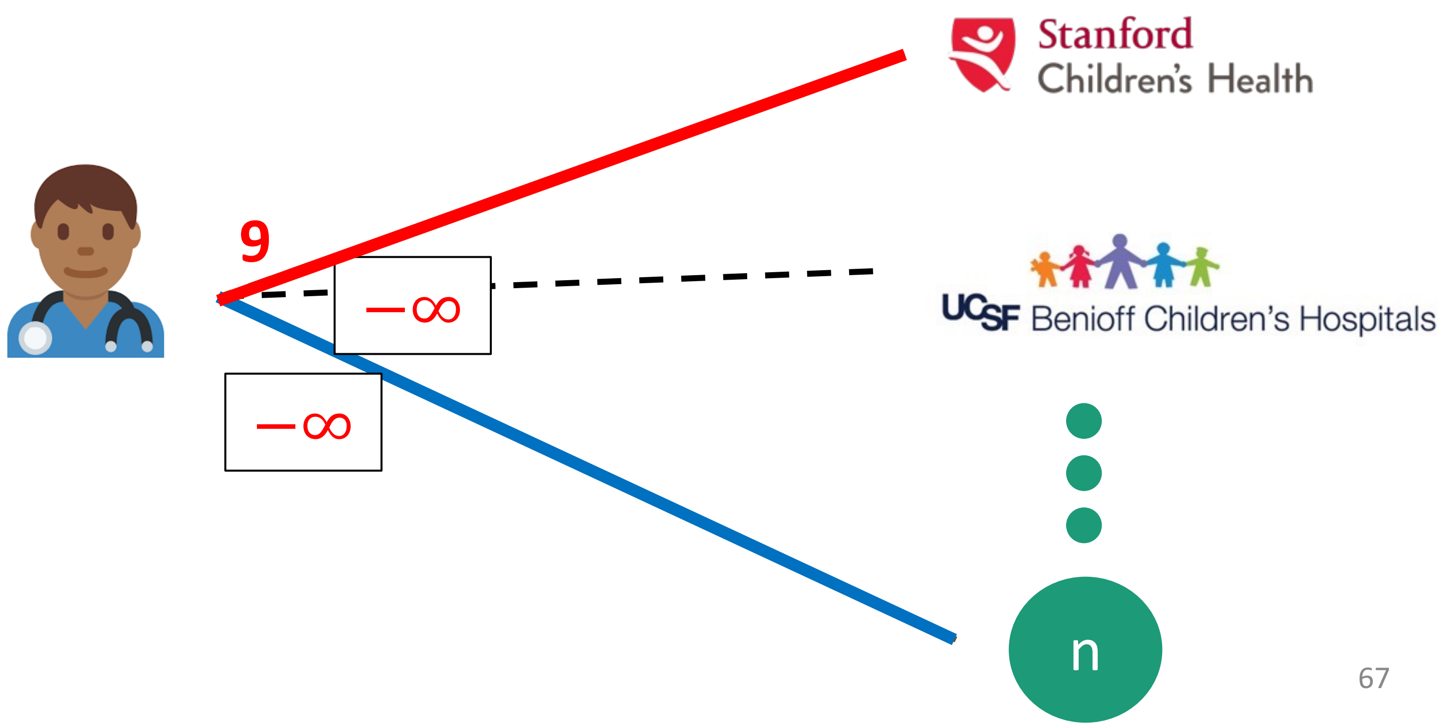
Corollary: Order of popping from freeDoctors does not change the output.

Theorem: Doctors cannot gain from misreporting their preferences.

# Stable Matching and Incentives

- Doctor 2 may tell you he only wants to go to Stanford, but…

Corollary: This won't help him if we find Stable Matching with DA!

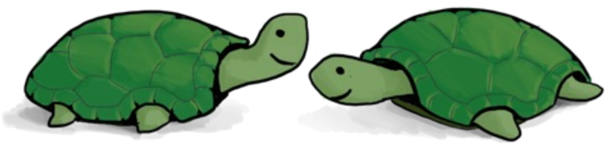


Prove this theorem too!

# The optimal stable matching?

<u>Theorem:</u> The matching returned by DA is **hospital-*worst***, i.e. every hospital is matched to *least*-favorite doctor possible in any stable matching.

<u>Caution:</u> Hospitals *can* gain from misreporting their preferences.



Think-pair-share:

**How would you find a hospital-optimal stable matching?**
**Should actual matching be doctor- or hospital-optimal?**

# What have we learned?

**Doctor-optimality**: The matching returned by DA is **doctor-optimal** (but hospital-*worst*)

**Truthful preferences corollary:** Doctors cannot gain from misreporting their preferences (but hospitals *can*).

Point:

It's important to **think** about how **our algorithms affect people.**

**Theorems** can help!

# Today

- Hospitals/residents problem

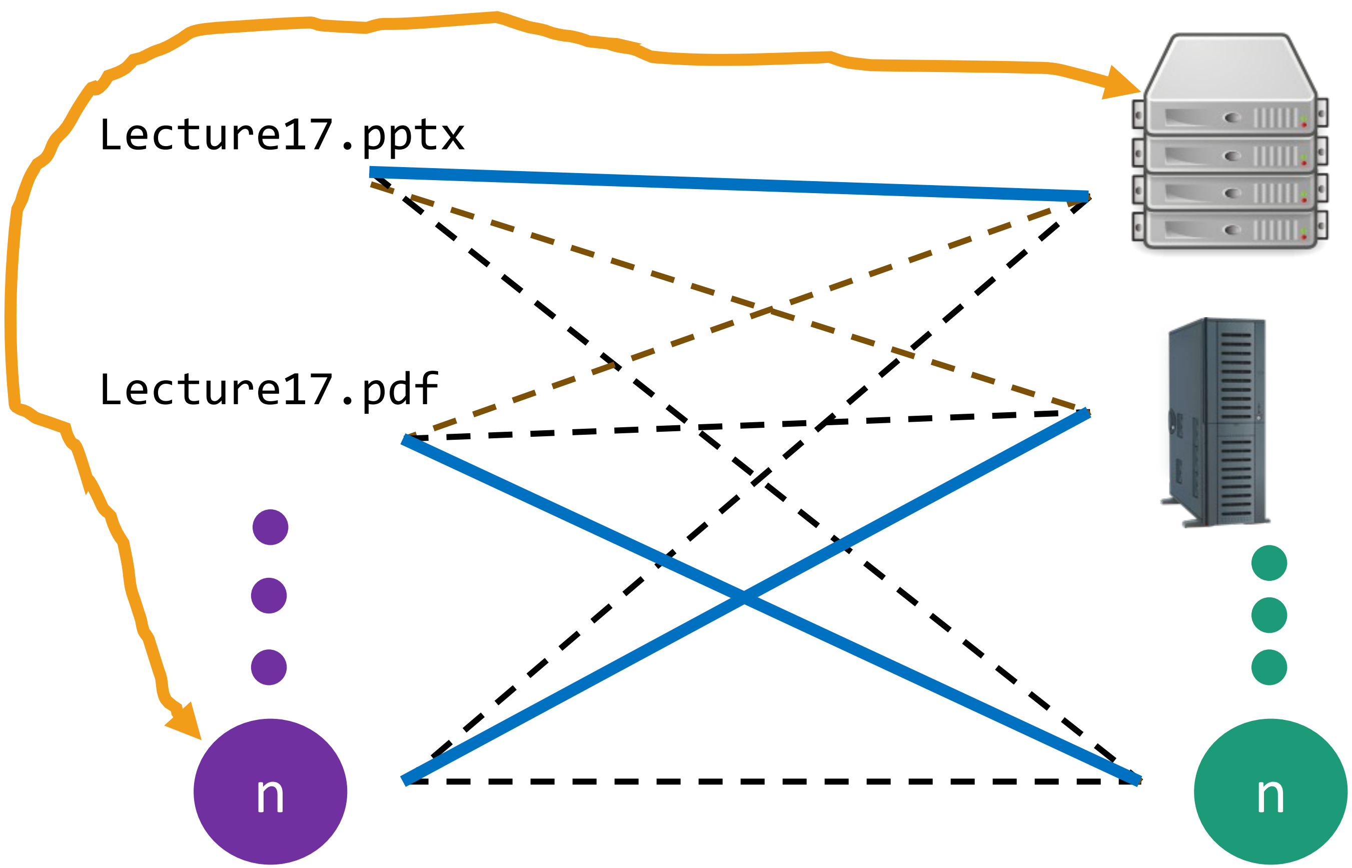
- Stable matchings
  - Solve the hospitals/residents problem
  - But can we find them?

- Deferred Acceptance Algorithm
  - Find stable matchings!

- Discussion, applications and non-applications

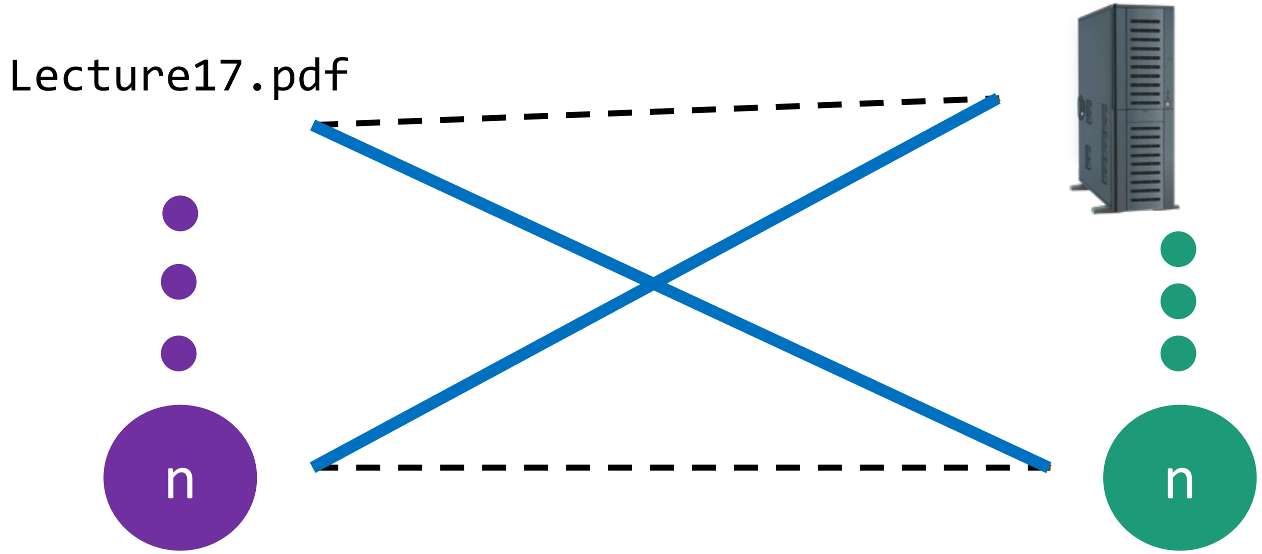
# Doctors vs Packets

- Suppose that instead of doctors and hospitals, you want to match packets to servers on the internet.



# Doctors vs Packets

- Suppose that instead of doctors and hospitals, you want to match packets to servers on the internet.
- When you *own all the servers*, you don't have to worry about them matching outside your algorithm...
- But it turns out that Deferred Acceptance is just very fast in practice ☺



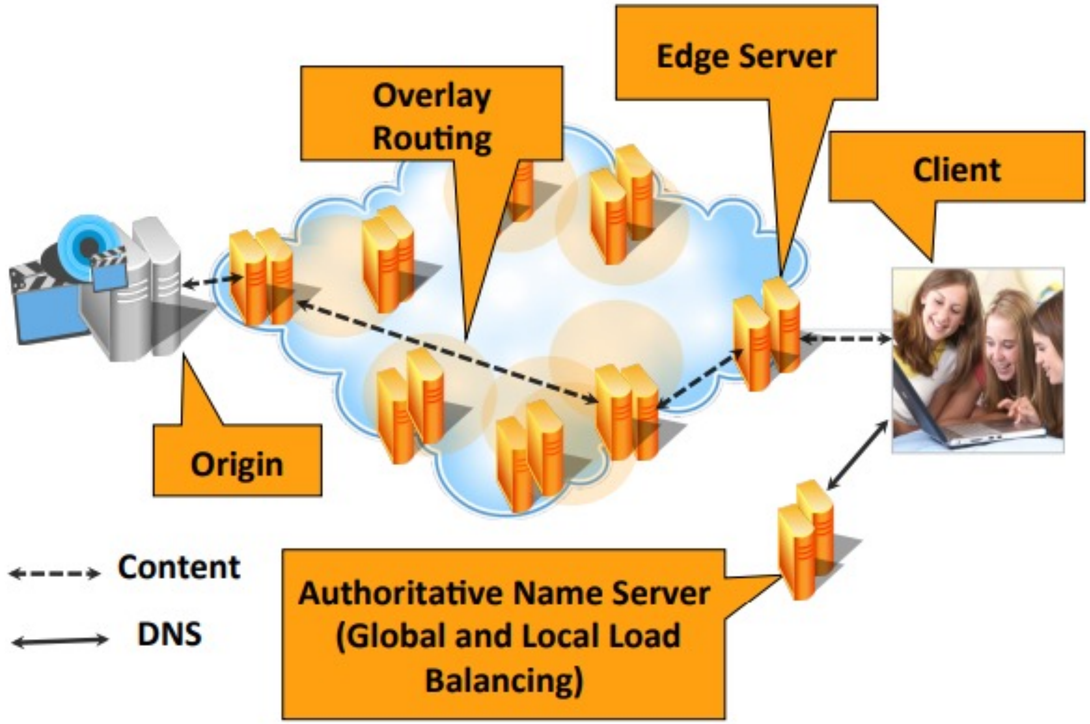
# Doctors vs Packets

- Suppose that instead of doctors and hospitals, you want to match packets to servers on the internet.
- When you *own all the servers*, you don't have to worry about them matching outside your algorithm...
- But it turns out that Deferred Acceptance is just very fast in practice ☺

**Truncated preference lists**
Packets typically get one of top servers
Total running time closer to $O(n)$!

Highly **distributed**:
Every packet looks for its own server!

# Doctors vs Packets




See "Algorithmic Nuggets in Content Delivery" (Maggs & Sitaraman, CCR'15)
for details on how Akamai uses Deferred Acceptance to match packets to servers

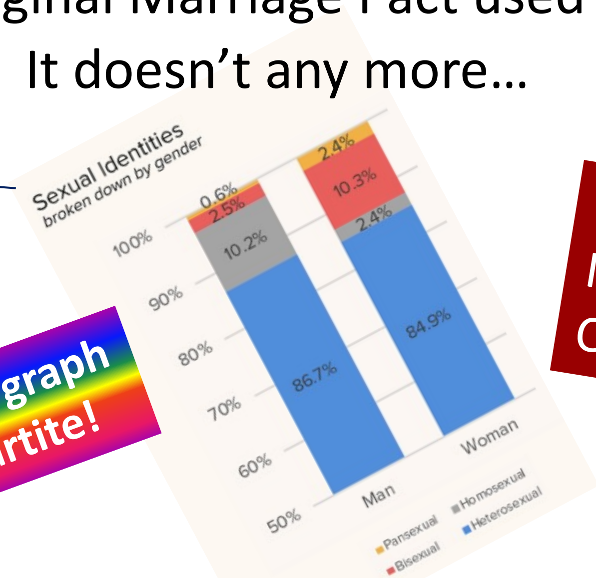
# Stanford Marriage Pact

# Stanford Marriage Pact

- Matches between Stanford students who want to make a *pact*: "If we don't get married by time X, we'll marry each other."

- Historically, Gale-Shapley's original paper talked about *Stable Marriage*
  - men = doctors; women = hospitals.

- Original Marriage Pact used variant of Deferred Acceptance
  - It doesn't any more…




**Preference graph is not bipartite!**

**Marriage Pact doesn't need stability:**
It is meant to be a back-up match-
Couples are *encouraged* to find outside matches!

# Recap

- Hospitals/residents problem
- Stable matchings
  - Solve the hospitals/residents problem
  - But can we find them?
- Deferred Acceptance Algorithm
  - Find stable matchings!
- Discussion, applications and non-applications

# Next time

- Quick and hand-wavey recap of past lectures.
- Algorithms beyond 161 …