CS218: design and analysis of algorithms

Greedy Algorithms

Yan Gu

Greedy Algorithms

- Among the commonly-used algorithm design strategies, greedy probably is the most intuitive and easiest to understand
- Once decision at a time
- When you need to make a decision, choose the “best” based on a certain criterion
- Not necessarily optimal, need to prove it

What do greedy choice and optimal substructure mean?

- **Greedy choice (intuitively):**
  - The element t you greedily choose is not a bad idea!
  - It appears in some optimal solution!
  - (for any optimal solution, if it doesn't contain t, we can modify it to contain t!)
  - So just choose it!

- **Optimal substructure (intuitively):**
  - After choosing some element t
  - The final optimal solution is just to find the optimal solution for the rest of the (compatible) elements!
  - Recursively solve it using the same approach

- **So we repeatedly choose the greedy choice!**

Well, sometimes greedy is not optimal, is it useless?

- They may still provide you with some nice features!

Approximation Algorithms and Competitive Analysis

Ski Rental (rent or buy problem)

- **You recently came to UCR and realized that you can go skiing at the Big Bear Lake Resort**
- **Buying the equipment costs about $500 and renting it for a weekend costs $50. Should your buy or rent?**
- **Clearly it depends on how many more times you go skiing in the future**
  - If you will go skiing a total of 11 times or more, then it is better to buy, and to do it now
  - If we will go 9 times or fewer, then it is better to rent
  - If we go 10 times it does not matter

A reasonable strategy

- **You will rent it first and buy it later**
- **You will first rent it for the first 10 times, and buy it if you go for the 11th time**
  - If you go 10 times or fewer time, then the strategy is optimal
  - If you go for the 11th time, you pay $10 \times \$50 + \$500 = \$1000$, while the optimal strategy (buying it on the first time) pays $\$500$
  - You will never pay more than twice of the optimal strategy

Approximation algorithms

- For a minimization problems instance I and an algorithm ALG, let $ALG(I)$ be the quality of the algorithm's output and $OPT(I)$ be the quality of the optimal solution

- For $c > 1$, ALG is a c-approximation algorithm if for every I, $ALG(I) \leq c \cdot OPT(I)$

- The abovementioned is a 2-approximation algorithm for the (online) ski rental problem

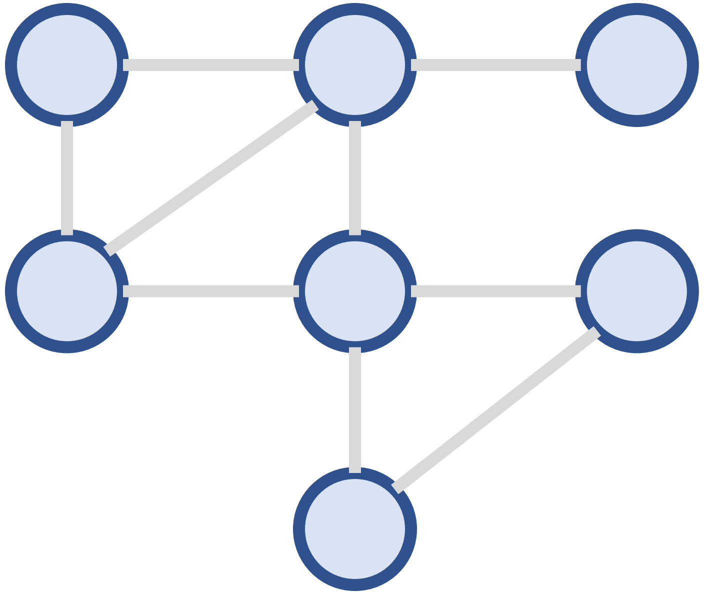
Approximation algorithms

- For a maximization problems instance I and an algorithm ALG, let $ALG(I)$ be the quality of the algorithm's output and $OPT(I)$ be the quality of the optimal solution

- For $c < 1$, ALG is a c-approximation algorithm if for every I, $ALG(I) \geq c \cdot OPT(I)$

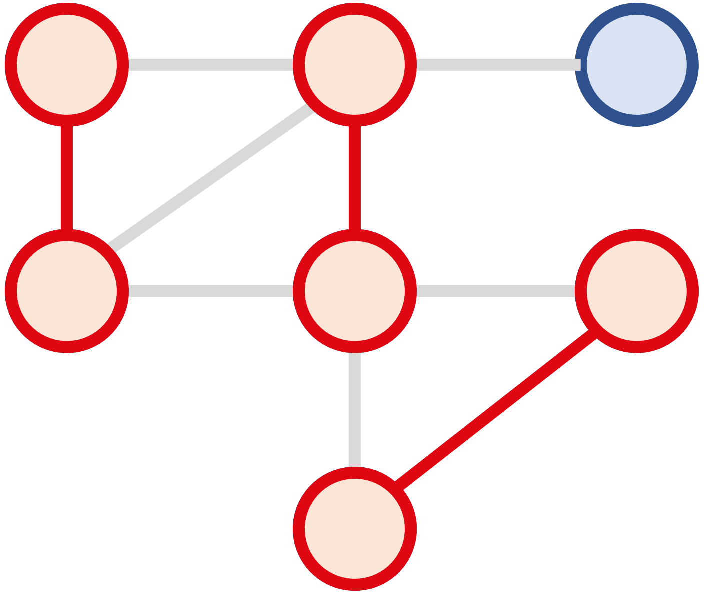
Holiday Celebration!

- You work for Riverside City Council and arranging the next Christmas celebration!
- You have the map of the celebration region, and want to have some police on some intersections, so they can help arranging and provide assistance
- You want to minimize the "police stops" but still have police on one of the two intersections of every (segment) of street
- Unfortunately, solving this problem (vertex cover) on a general graph is NP-Complete



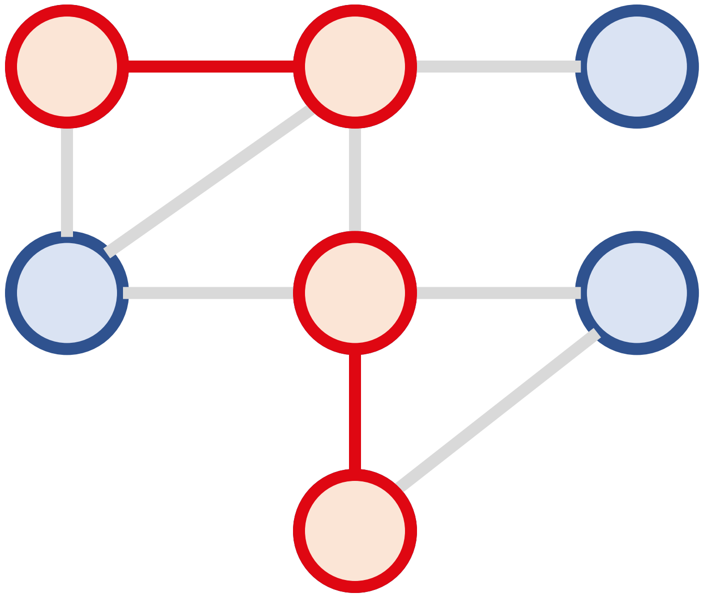
Vertex Cover Problem

- It is hard to know the exact number M, but …
- Consider a simple strategy, I enumerate every street, and if it is "uncovered", I just put police stops on both ends, and let S be the stops we put here
- Lemma: $|S| \geq |M|$
- Proof: S must cover at least one vertex for each edge in M, and this vertex covers no other edge in M



An approximate solution

- Repeatedly find "uncovered" streets and put stops on both ends
- Lemma: $|S| \leq 2 \cdot OPT(G)$
- Proof: For each $e \in E$, at least one endpoint is in M, so S is a valid solution, and $|S| = 2|M| \leq 2 \cdot OPT(G)$
- No need to know $OPT(G)$!



Fun fact: can we replace the constant 2 with $\alpha < 2$?

- No better constant-factor approximation algorithm is known
- The minimum vertex cover problem is APX-complete: it cannot be approximated arbitrarily well unless P = NP
- Using PCP theorem, one can show that $\alpha \geq \sqrt{2}$; if the unique games conjecture is true, then $\alpha = 2$
- More similar ideas will be covered by CS 219 by Amey
- Another course: https://www.cs.cmu.edu/~anupamg/adv-approx/

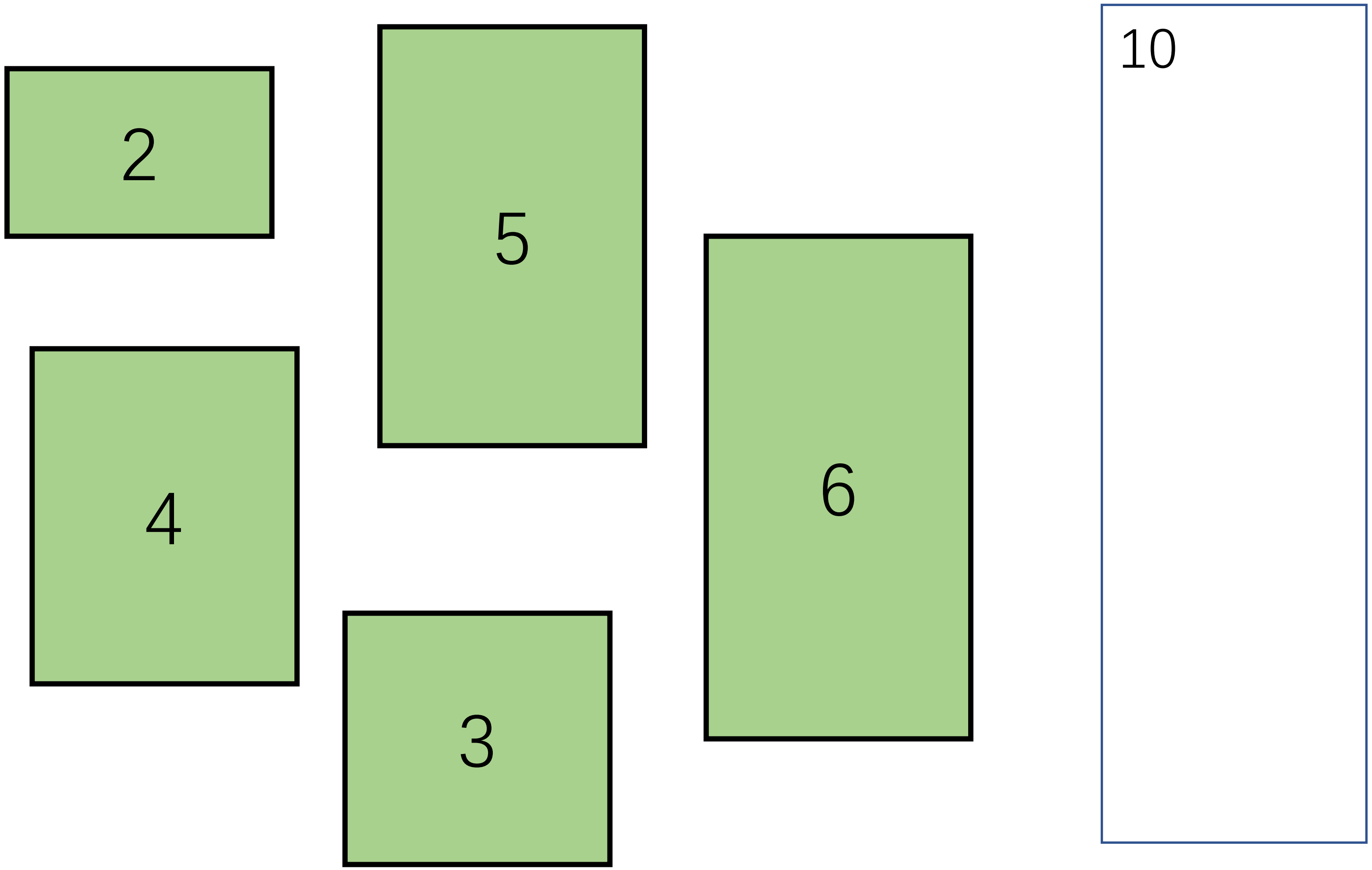
Bin Packing

- **Given n items of different weights and bins each of capacity c, assign each item to a bin such that number of total used bins is minimized. It may be assumed that all items have weights smaller than bin capacity.**

Bin Packing

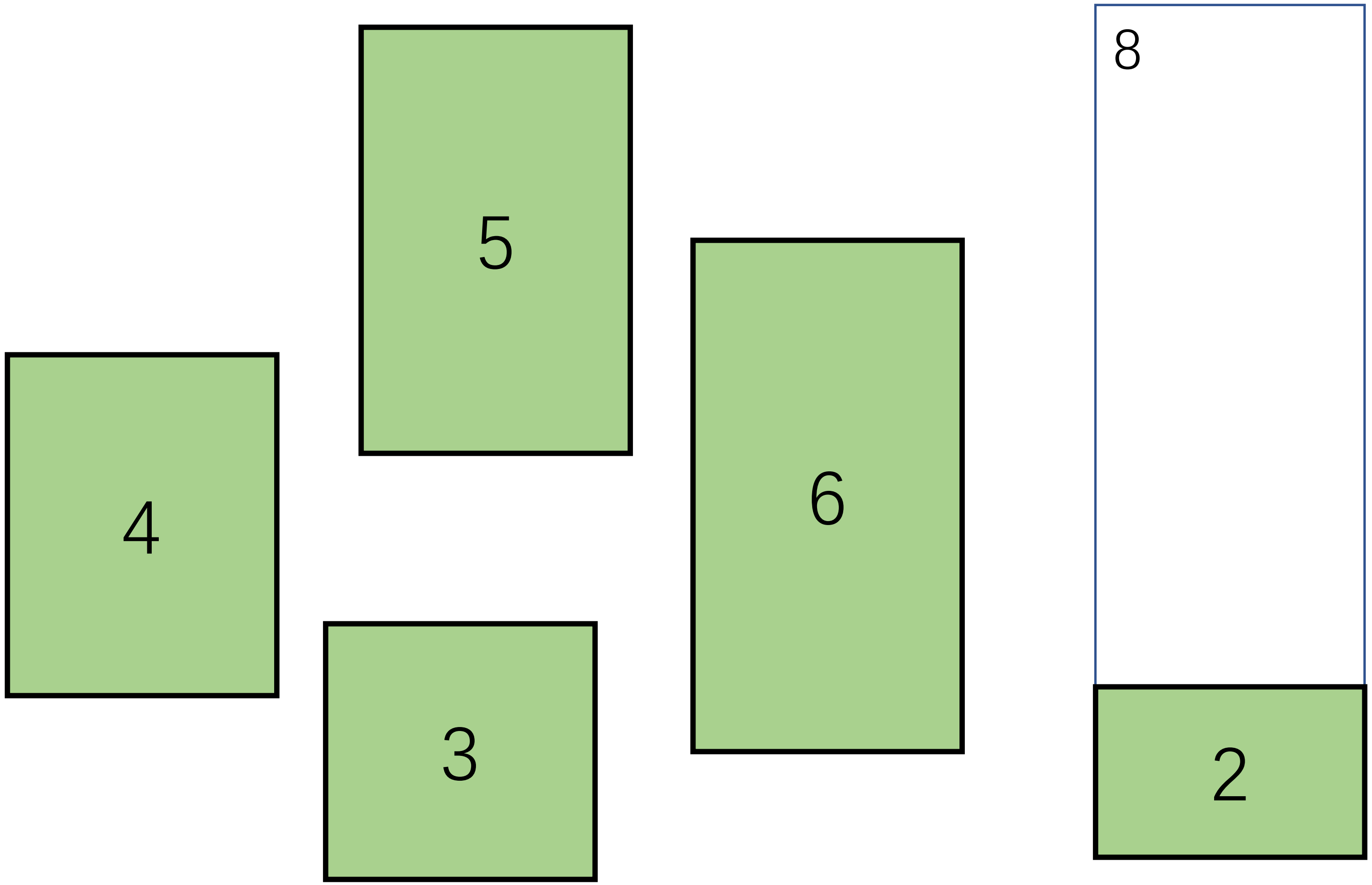
- **Given n items of different weights and bins each of capacity c, assign each item to a bin such that number of total used bins is minimized. It may be assumed that all items have weights smaller than bin capacity.**

Example: Bin Packing



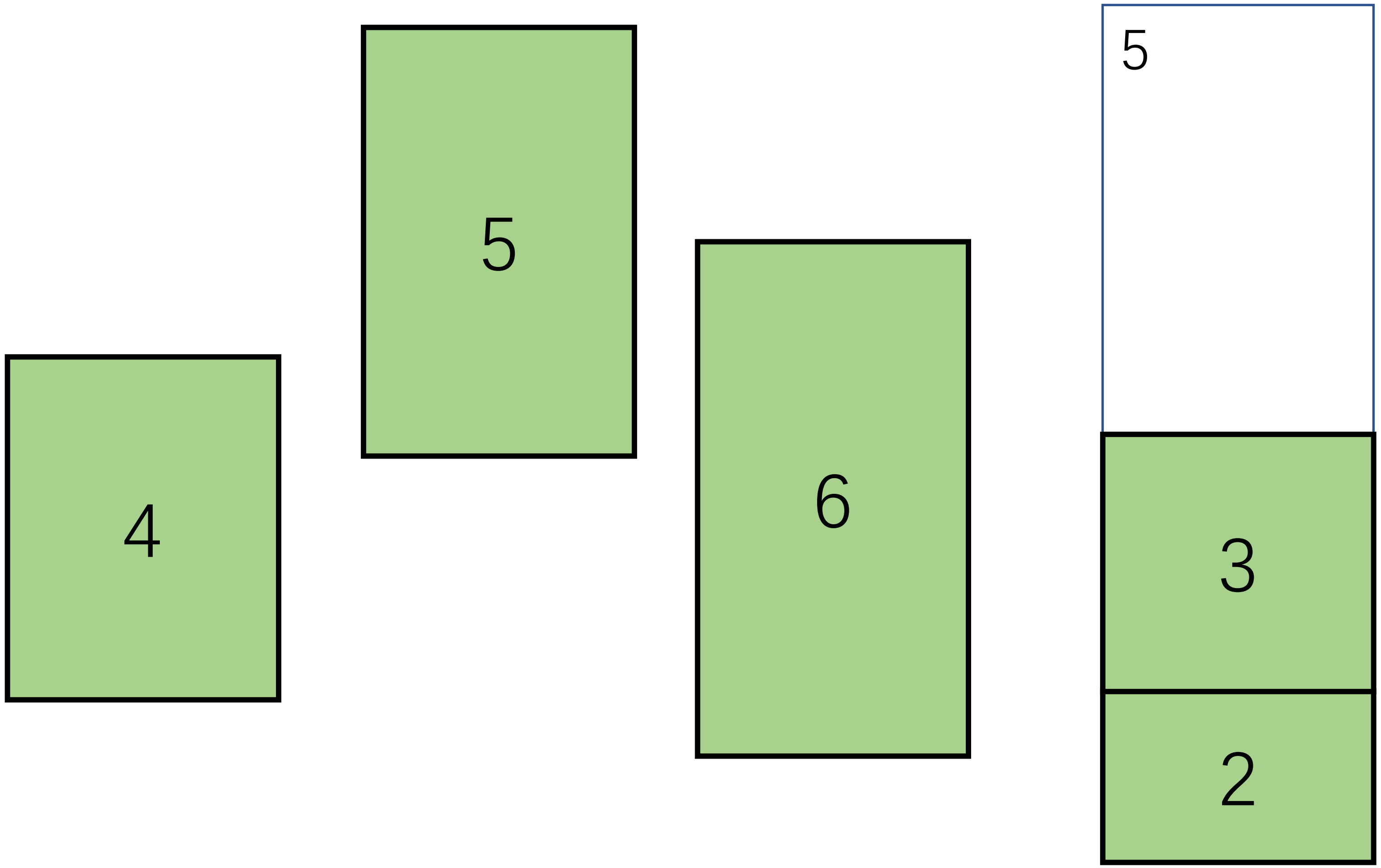
Using the fewest number of bins to pack all the objects

Example: Bin Packing



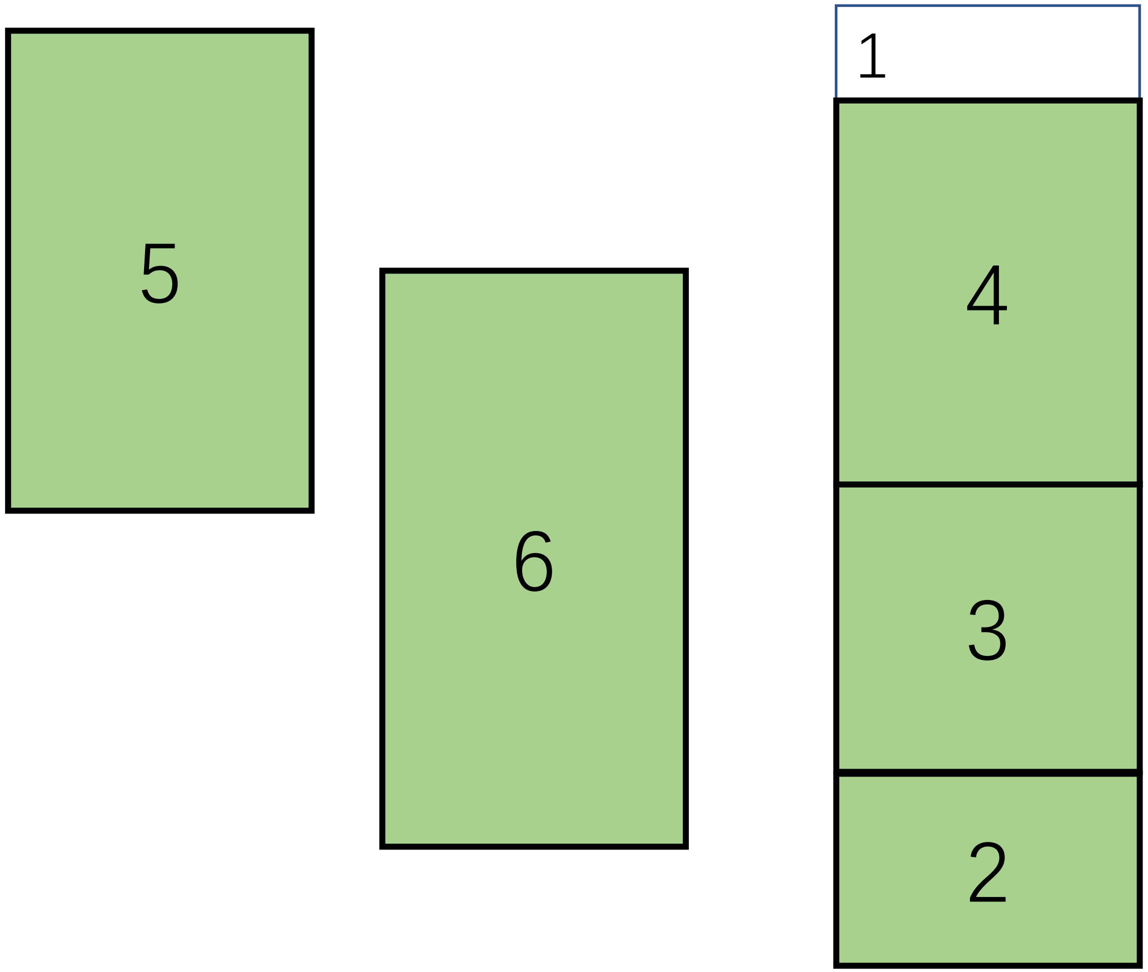
Using the fewest number of bins to pack all the objects

Example: Bin Packing



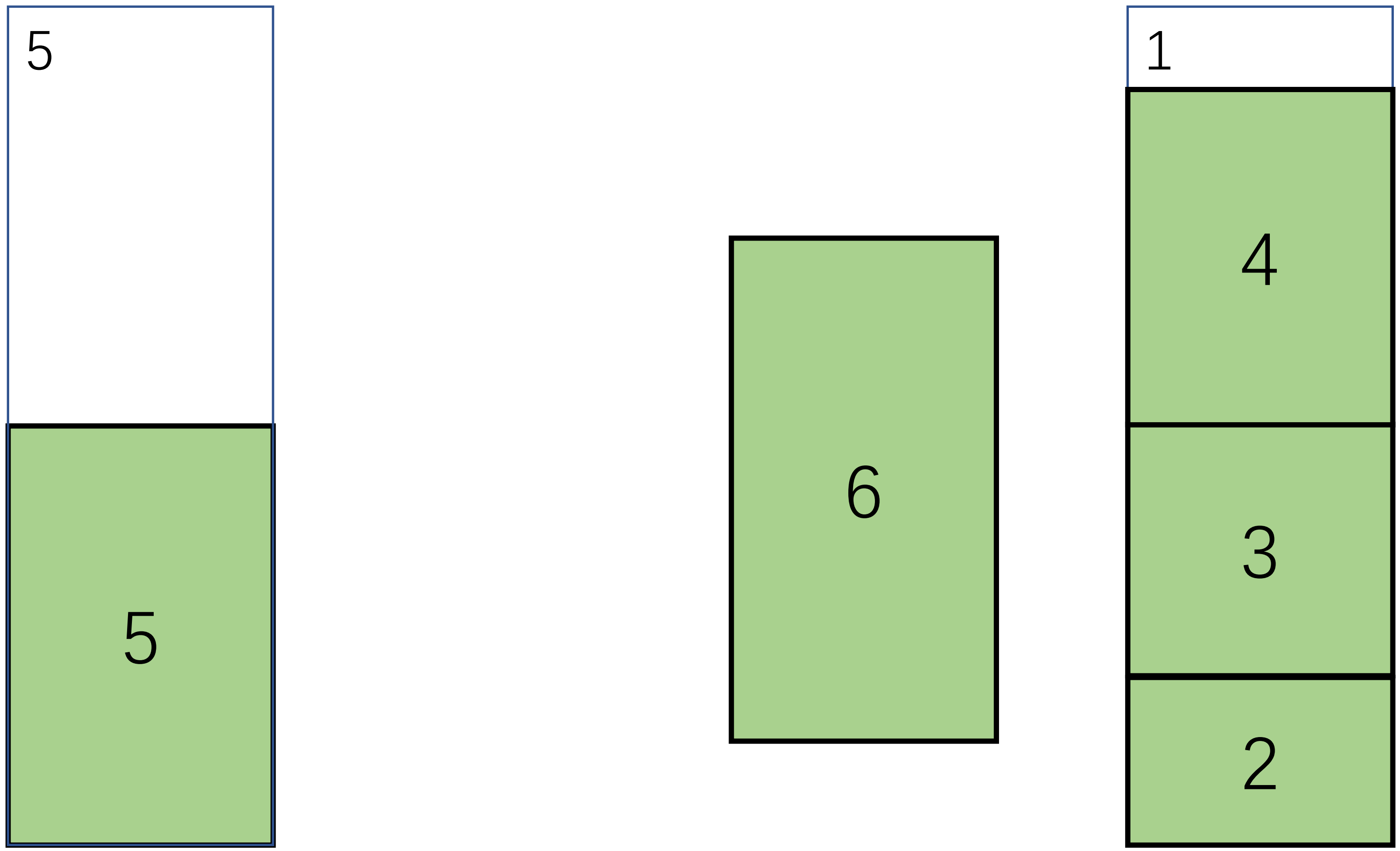
Using the fewest number of bins to pack all the objects

Example: Bin Packing



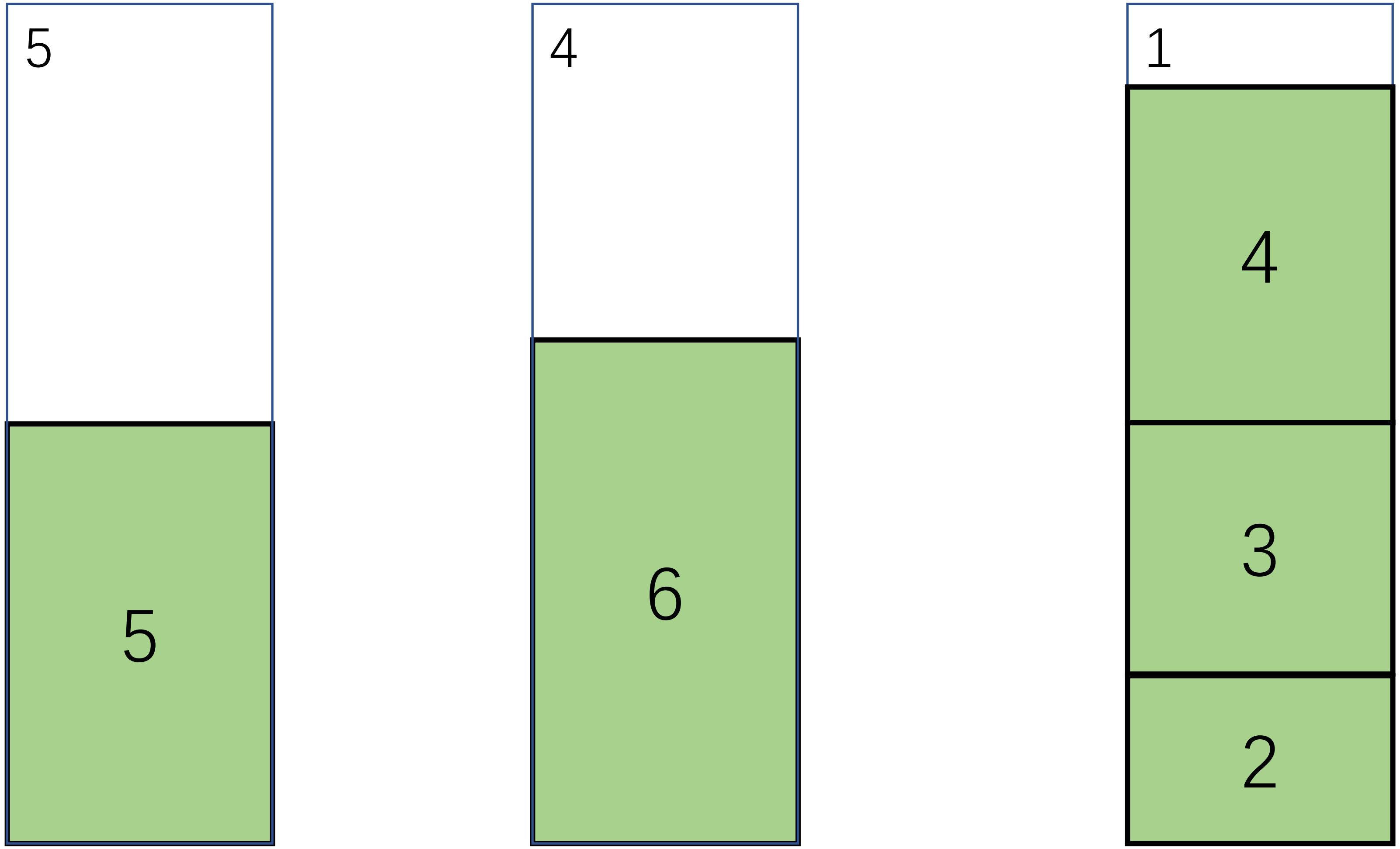
Using the fewest number of bins to pack all the objects

Example: Bin Packing



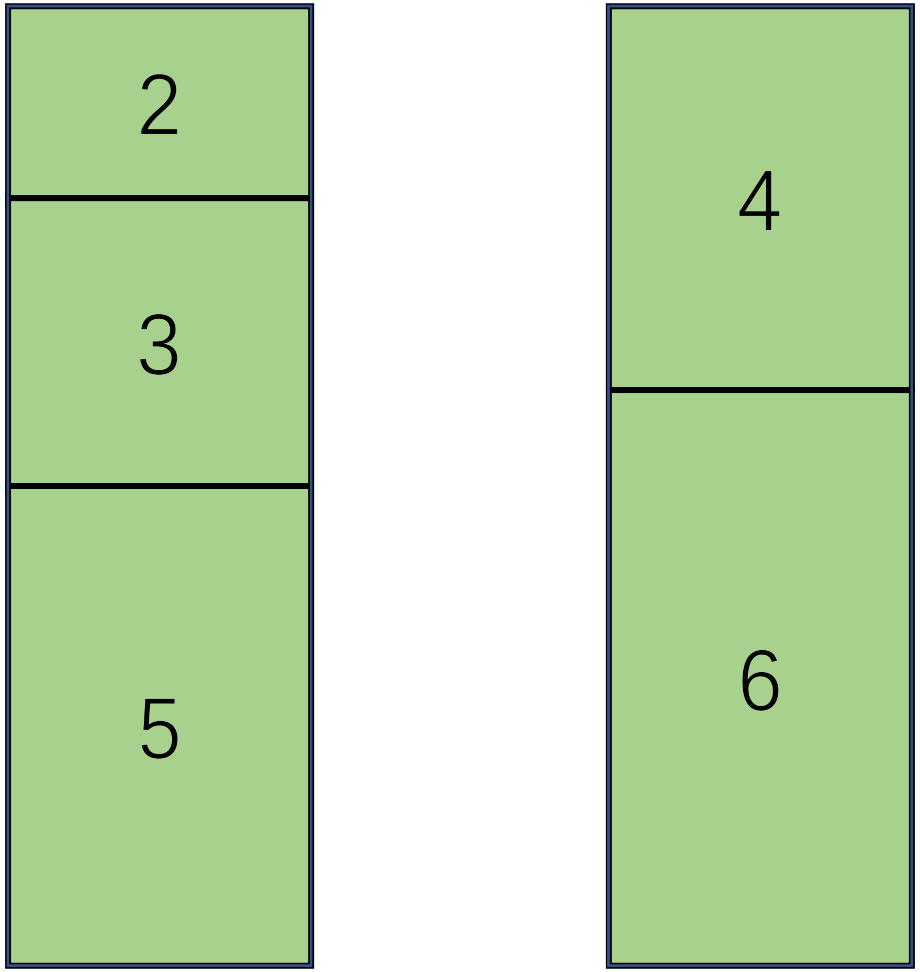
Using the fewest number of bins to pack all the objects

Example: Bin Packing



Number of bins = 3

Example: Bin Packing



(Optimal Solution) Number of bins = 2

Bin Packing

- Given n items of different weights and bins each of capacity c, assign each item to a bin such that number of total used bins is minimized. It may be assumed that all items have weights smaller than bin capacity.

- What strategies will you use?

Greedy algorithms and their bounds

- If the optimal solution is x

- Next fit (put in the current, start a new one if it doesn't fit the current): no more than $2x$ bins
- First fit (find the first one that fits): no more than $1.7x$ bins
- Best fit (put in the tightest spot): no more than $1.7x$ bins
- Worst fit (leave the largest space): no more than $2x - 2$ bins

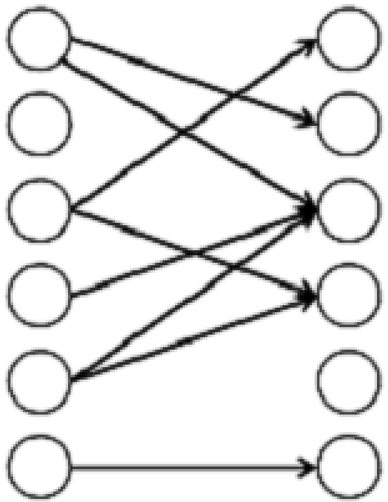
- First fit decreasing (sort and work on the largest first): no more than $(4x + 1)/3$ bins

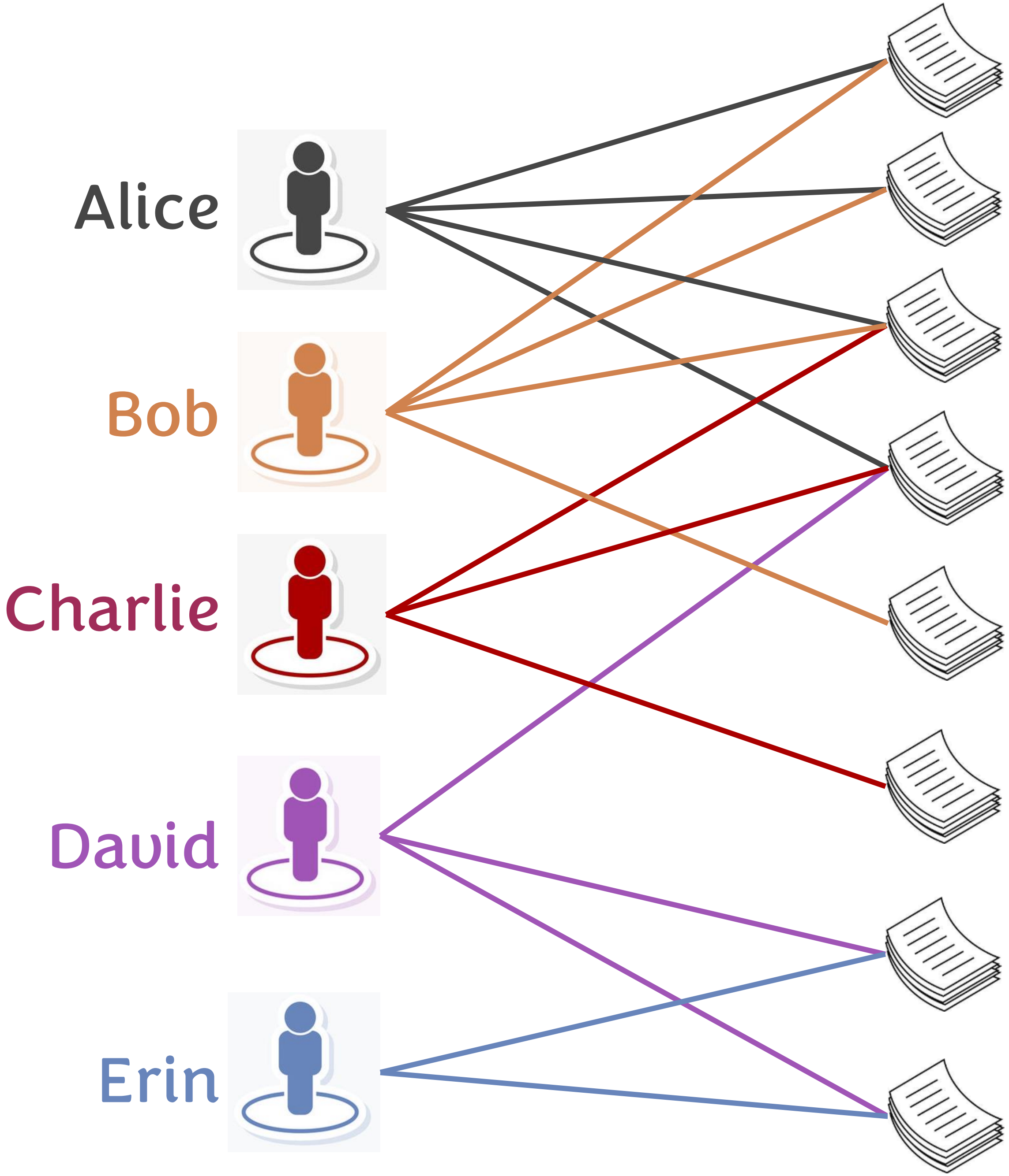
Maximum Coverage Problem

The maximum coverage problem

- You will organize a seminar at UCR!
- You want to invite some famous researchers, each of them will present their recent papers
- One paper may have multiple authors – anyone of them can give the talk
- A researcher may have multiple papers – they can present all of them
- Well, but you only have budget to invite k of them
- You want to maximize the number of papers that may be presented!

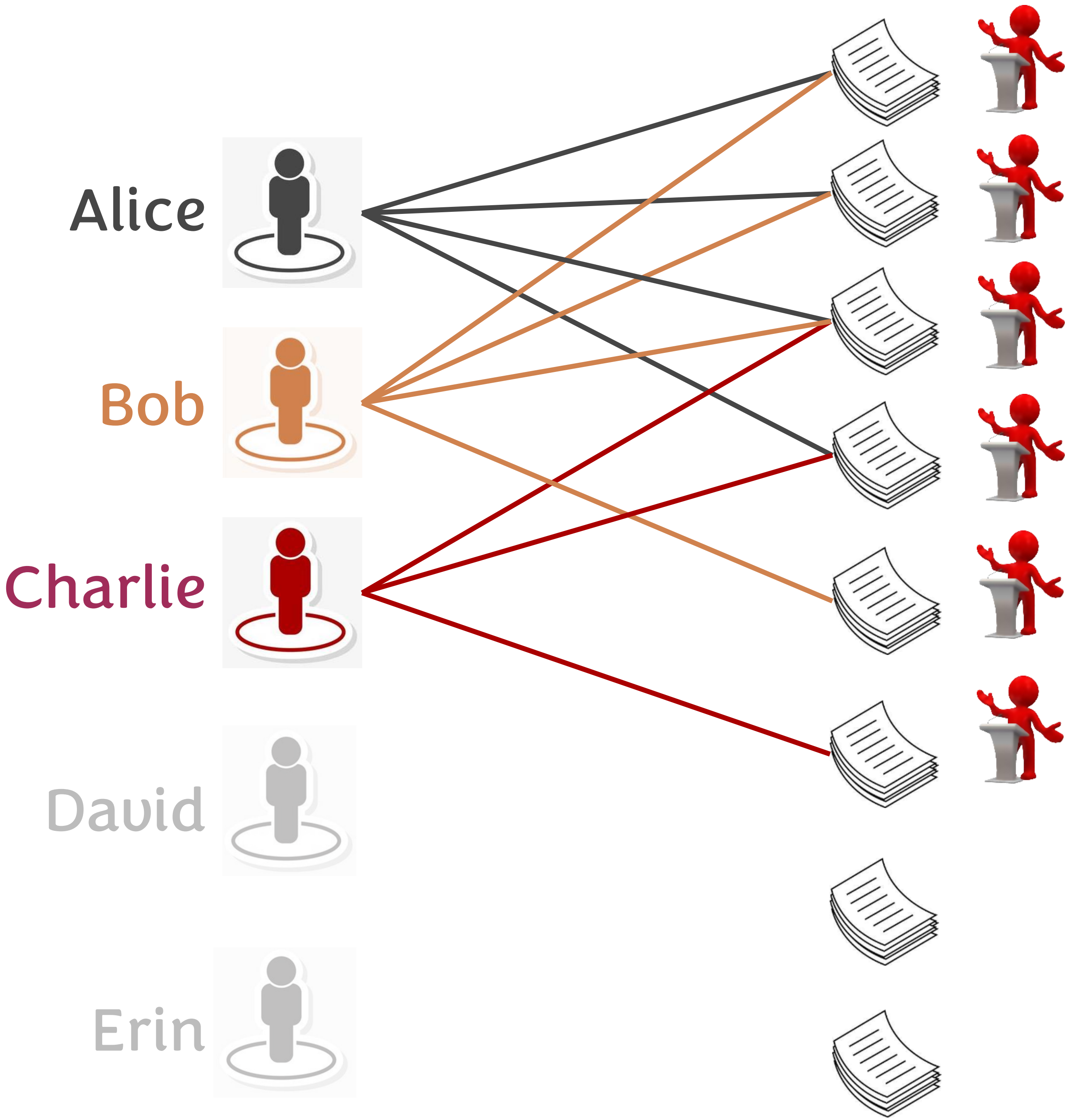
- (The general question: given a bipartite graph, choose k vertices on the left to cover as many vertices on the right as possible)





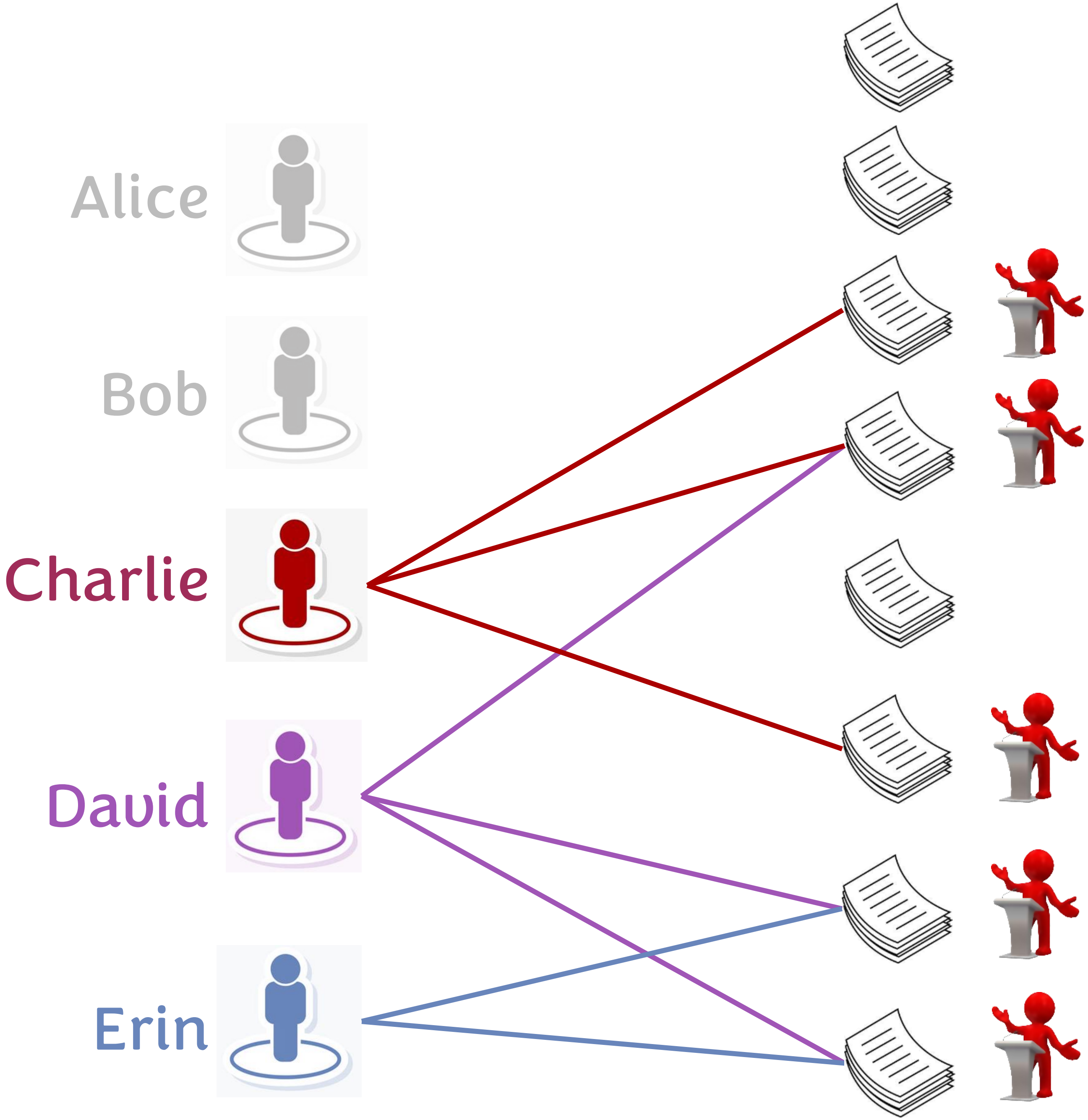
$$n = 5$$
$$m = 8$$
$$k = 3$$

How to maximize the number of presented papers?



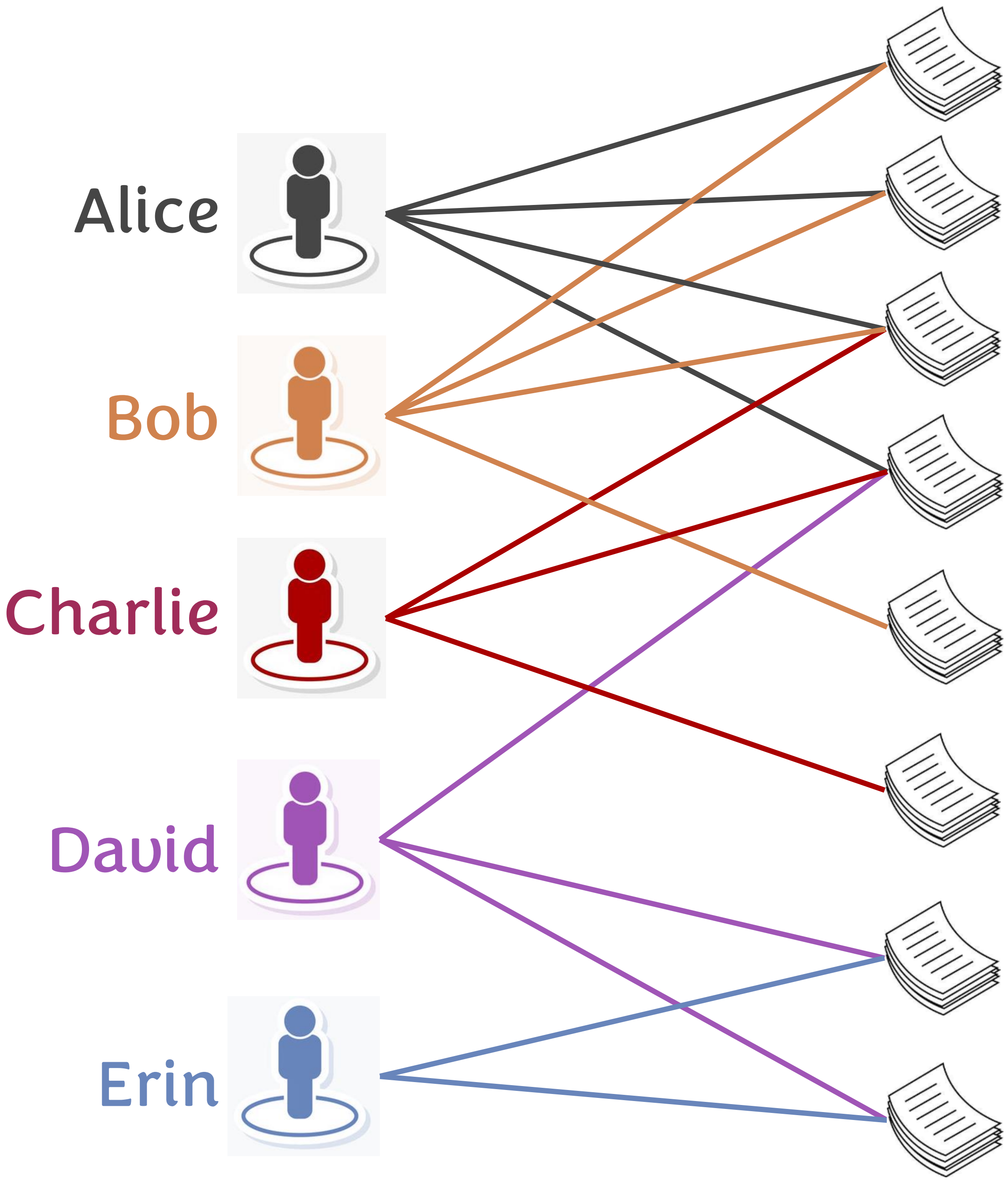
$n = 5$
$m = 8$
$k = 3$

How to maximize the number of presented papers?



$n = 5$
$m = 8$
$k = 3$

How to maximize the number of presented papers?



$n = 5$
$m = 8$
$k = 3$

- If $k = 1$,
  - Choose the one with the most number of papers!
  - Either Alice or Bob
- If $k = 2$, should we choose the top two with #papers?
  - Not necessarily!
  - Alice + Bob = 5
  - Alice + David (or Erin) = 6
- What is the criteria that we should consider??

How to maximize the number of presented papers?

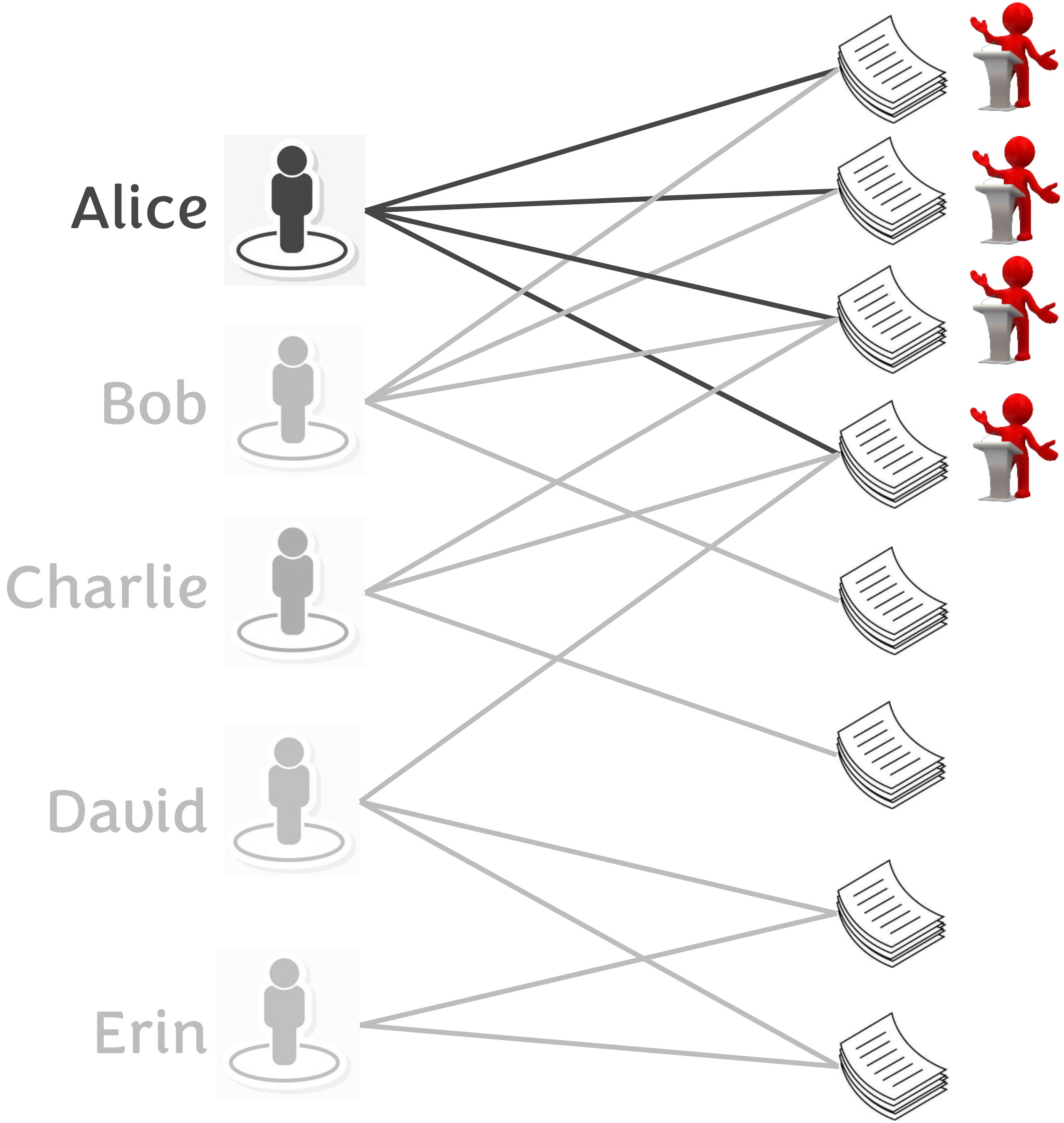
Marginal Increase

- **Given a set V, we want to select a subset $S \subseteq V$, in order to maximize some function $f(S)$**
  - $f(S)$ is a function on a set
- **When adding a new element x to the set, we say the "marginal gain" or "marginal increase" of adding x is**

$$f(S \cup \{x\}) - f(S)$$

- **Which is the extra benefit we can get by adding x**

- **For the researcher-paper problem, if we already decide to invite some researchers (set S), the marginal increase of inviting another researcher is the number of extra talks s/he will give**



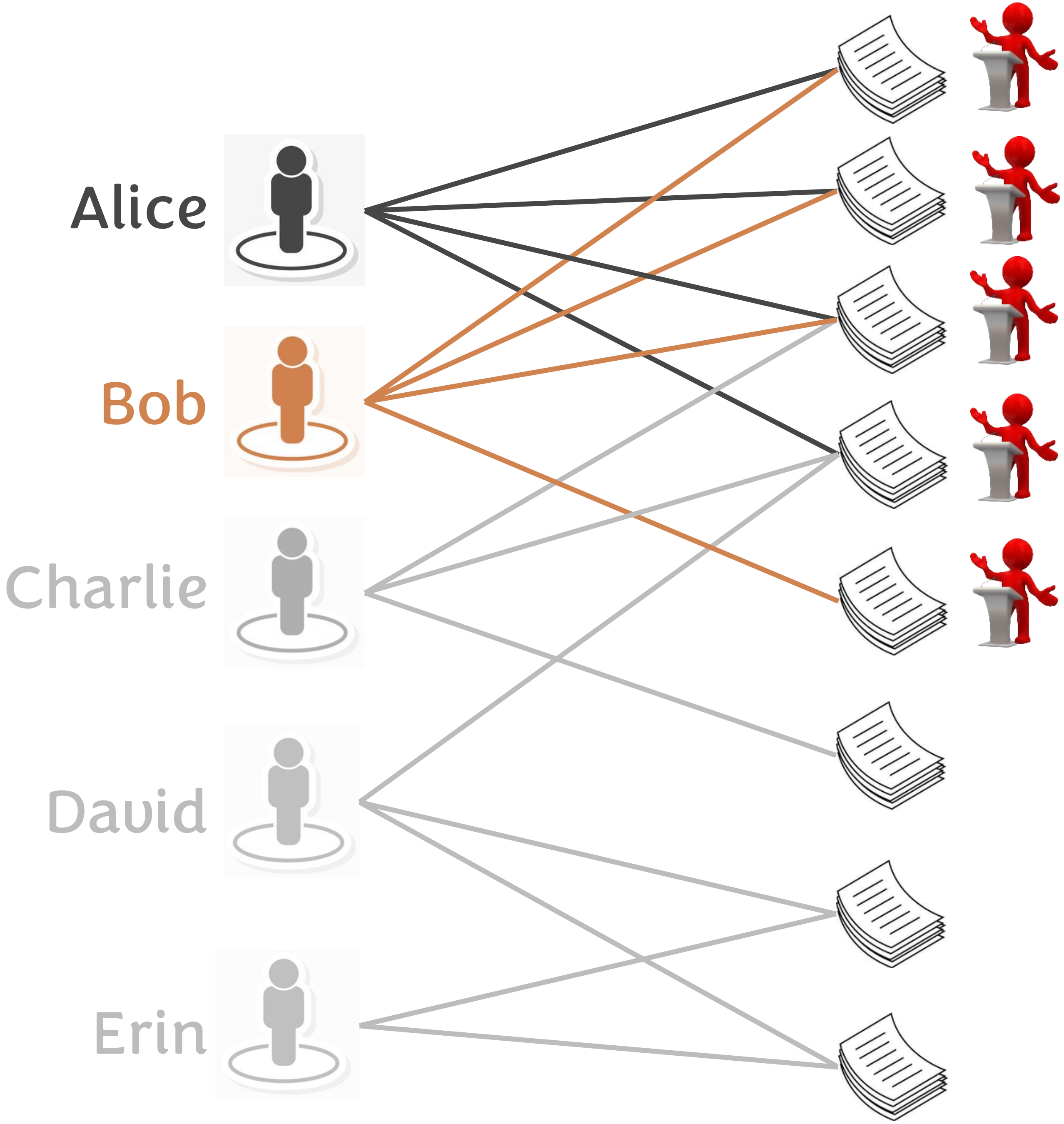
$n = 5$
$m = 8$
$k = 3$

- If we have invited Alice, who is the one with the most "marginal gain"?

How to maximize the number of presented papers?

Marginal gain

1



$n = 5$

$m = 8$

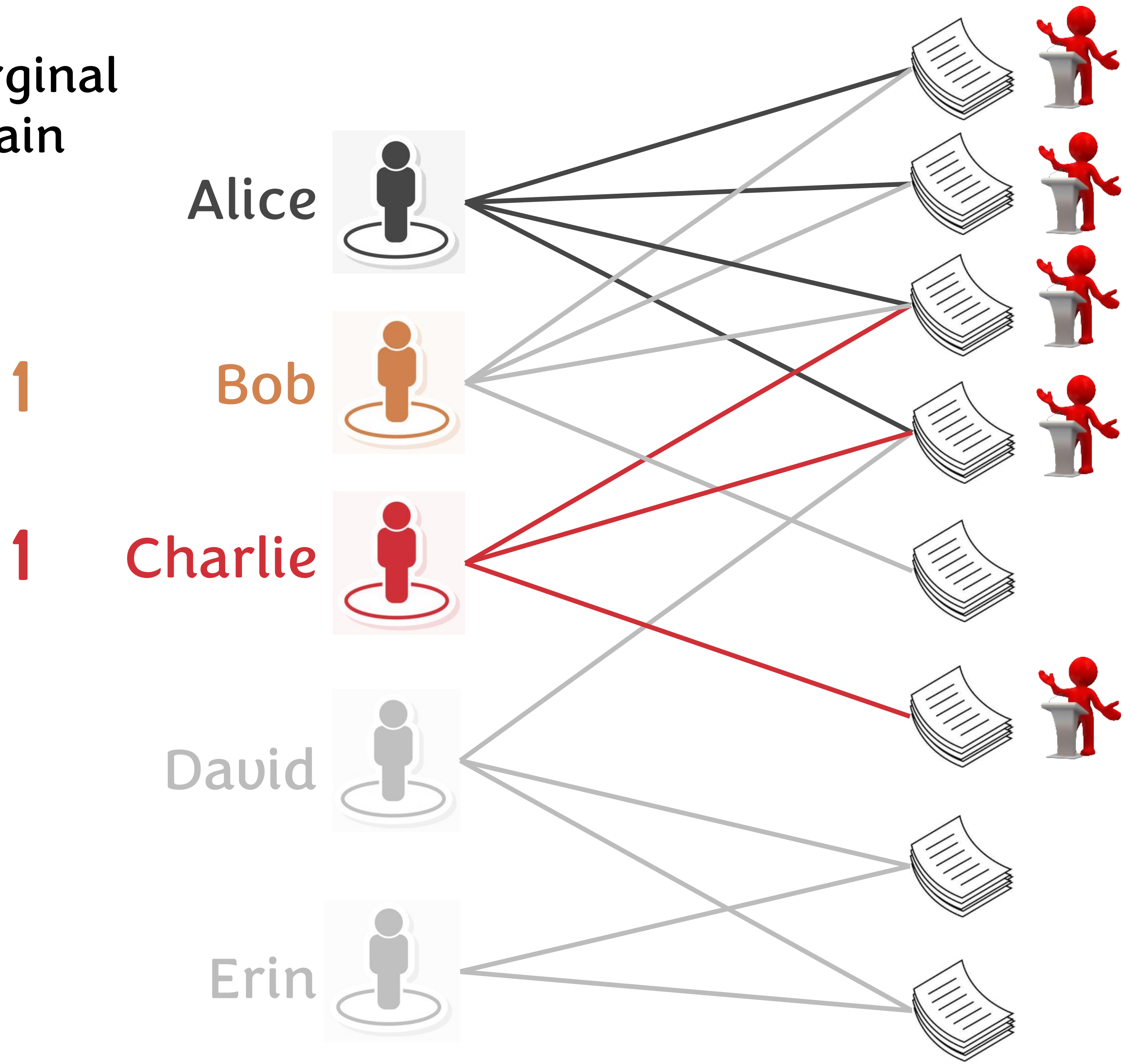
$k = 3$

- If we have invited Alice, who is the one with the most "marginal gain"?

- f({Alice, Bob}) = 5

How to maximize the number of presented papers?

Marginal gain



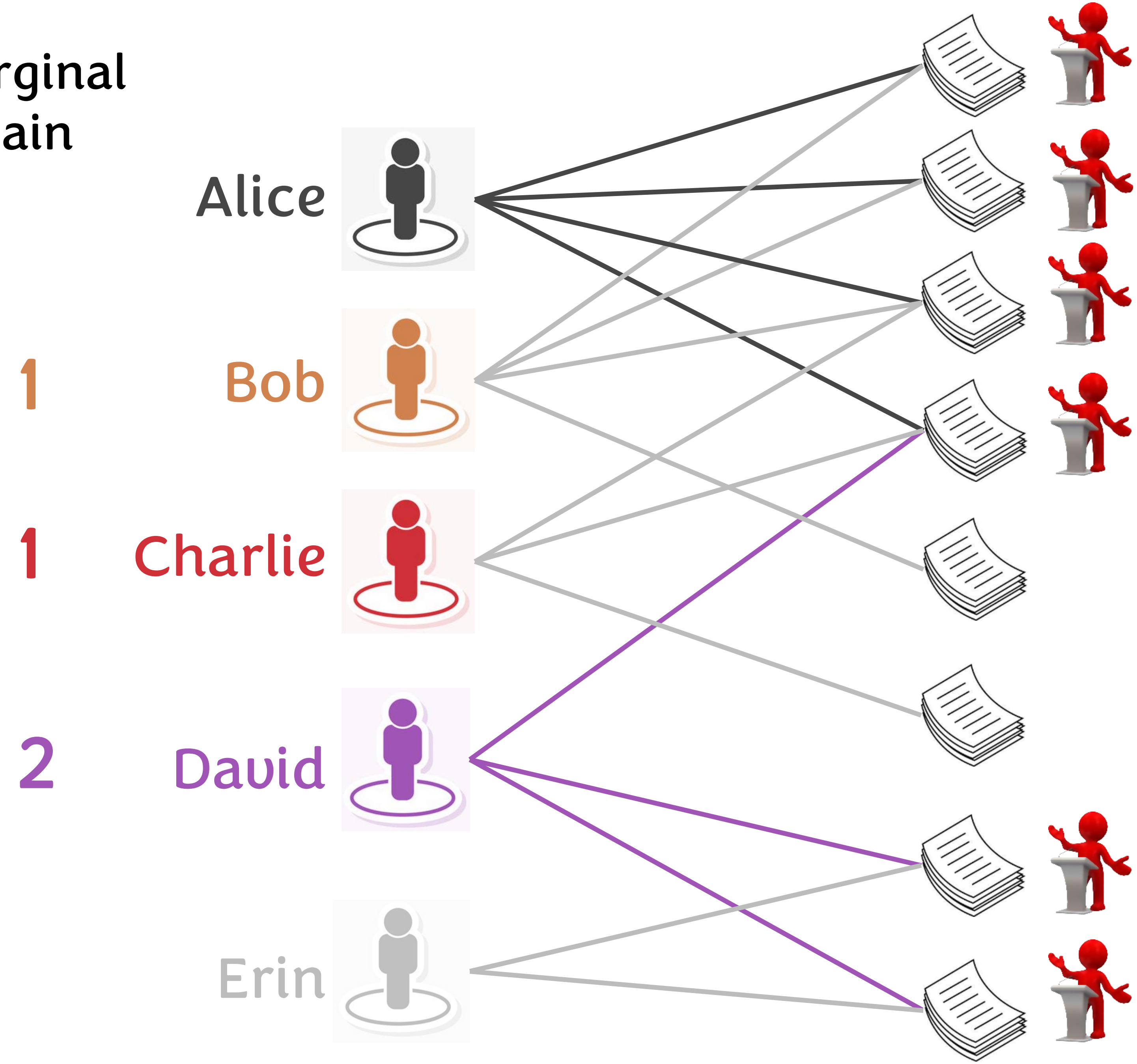
$n = 5$
$m = 8$
$k = 3$

- If we have invited Alice, who is the one with the most "marginal gain"?

- f({Alice, Bob}) = 5
- f({Alice, Charlie}) = 5

How to maximize the number of presented papers?

Marginal gain



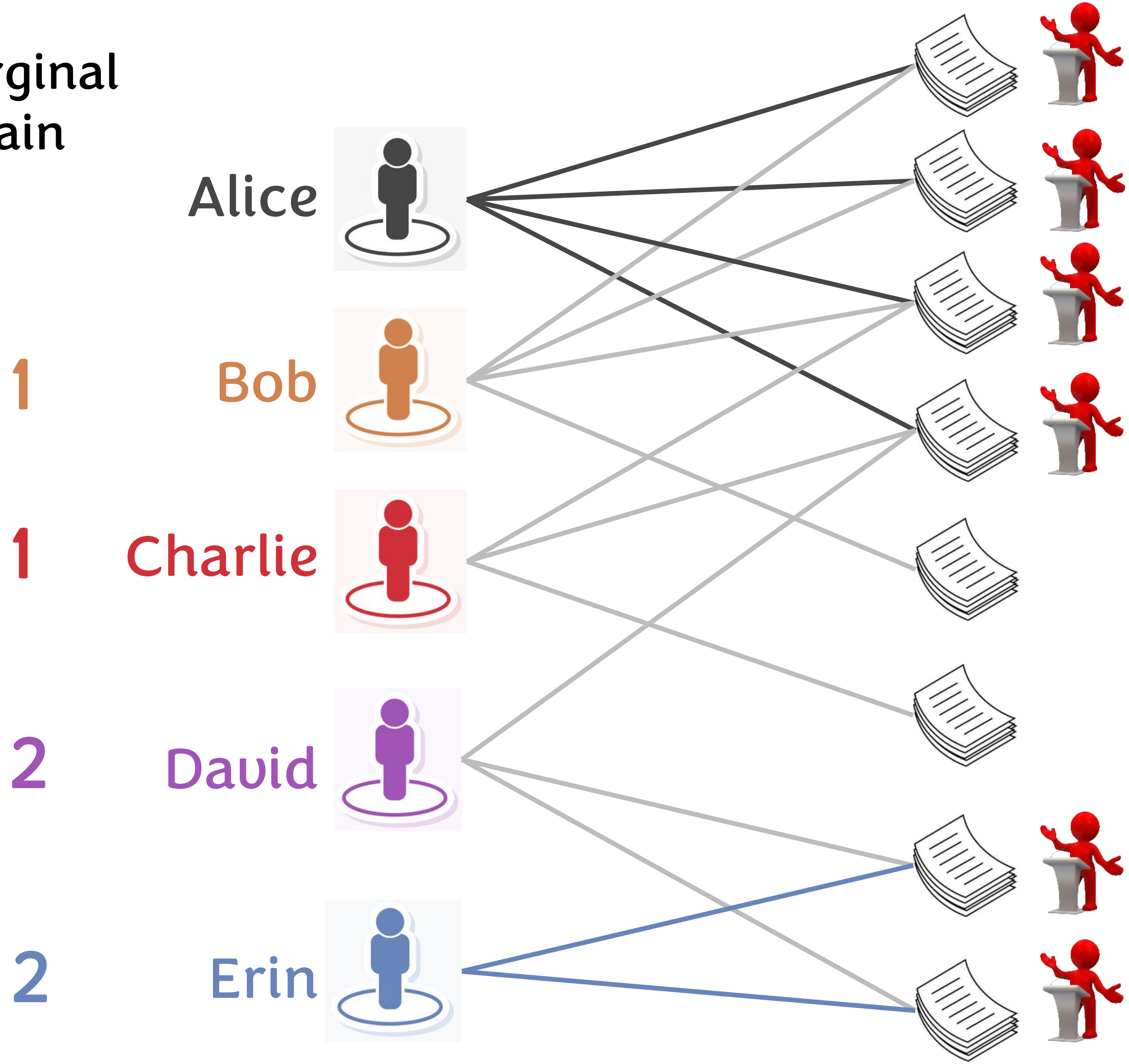
$n = 5$
$m = 8$
$k = 3$

- If we have invited Alice, who is the one with the most "marginal gain"?

- f({Alice, Bob}) = 5
- f({Alice, Charlie}) = 5
- f({Alice, David}) = 6

How to maximize the number of presented papers?

Marginal gain



$n = 5$
$m = 8$
$k = 3$

- If we have invited Alice, who is the one with the most “marginal gain”?

- f({Alice, Bob}) = 5
- f({Alice, Charlie}) = 5
- f({Alice, David}) = 6
- f({Alice, Erin}) = 6

How to maximize the number of presented papers?

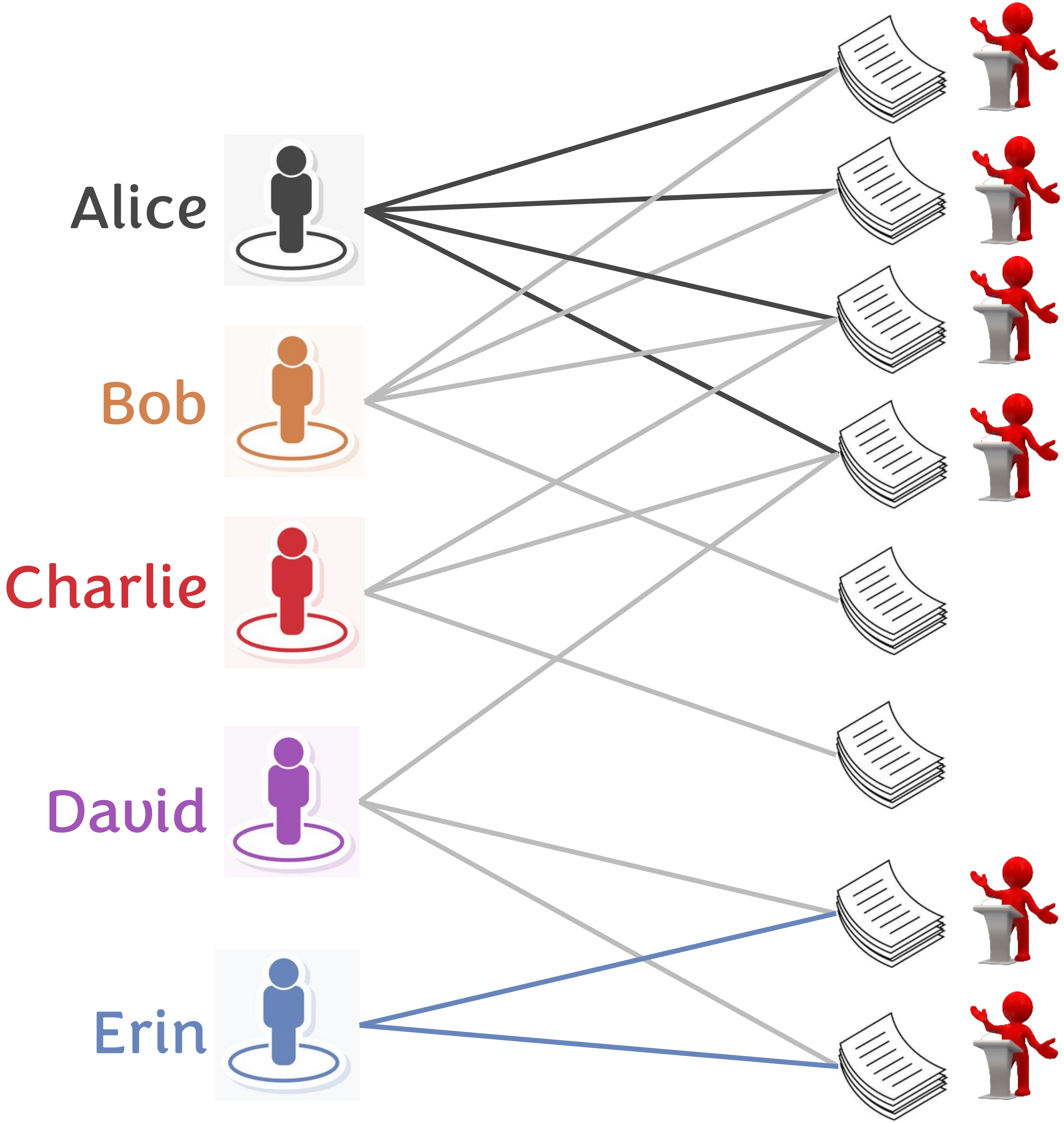
Marginal gain

1 (Bob)

1 (Charlie)

2 (David)

2 (Erin)

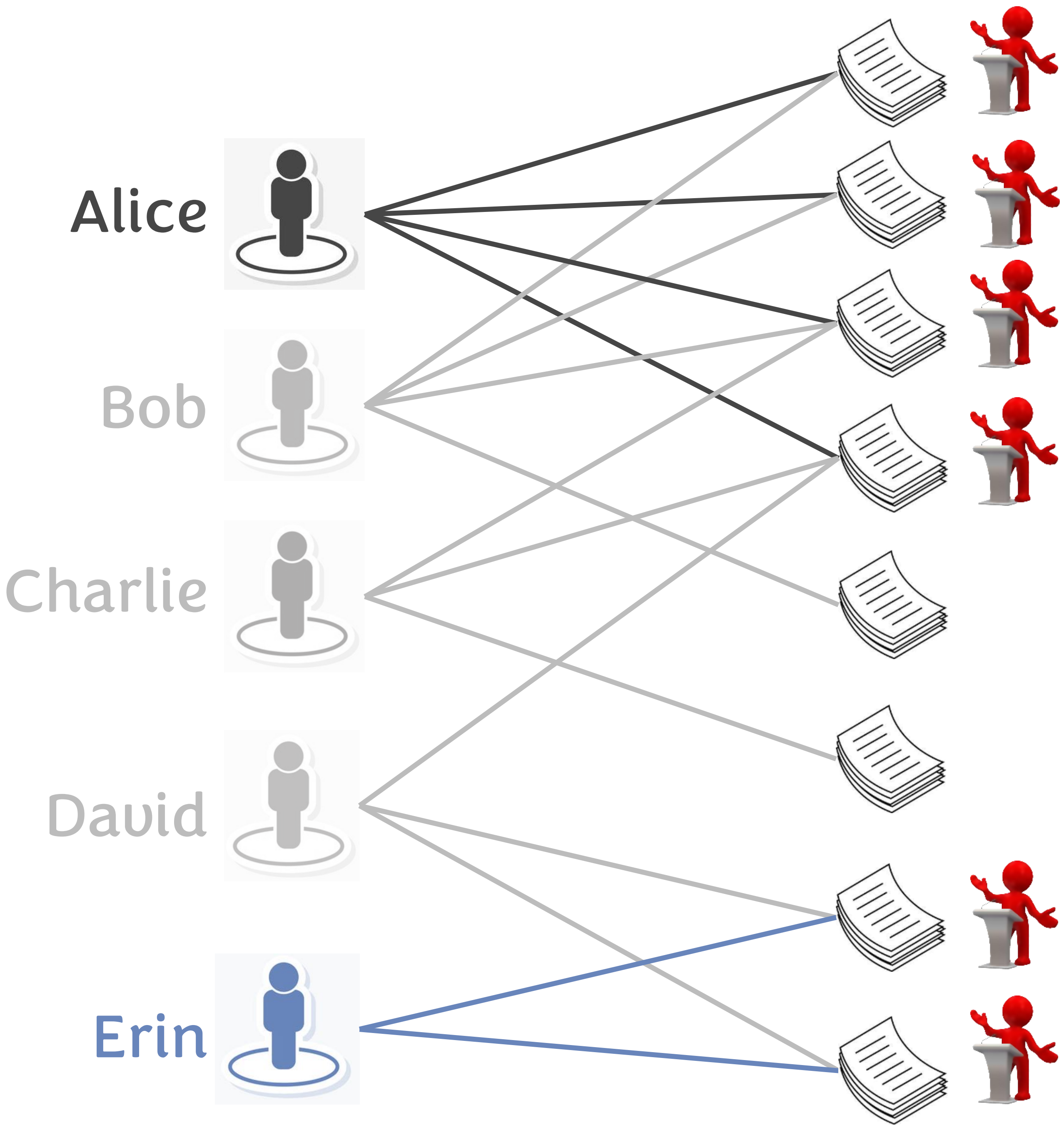


$n = 5$
$m = 8$
$k = 3$

- f({Alice, Erin}) = 6

How to maximize the number of presented papers?

Marginal gain



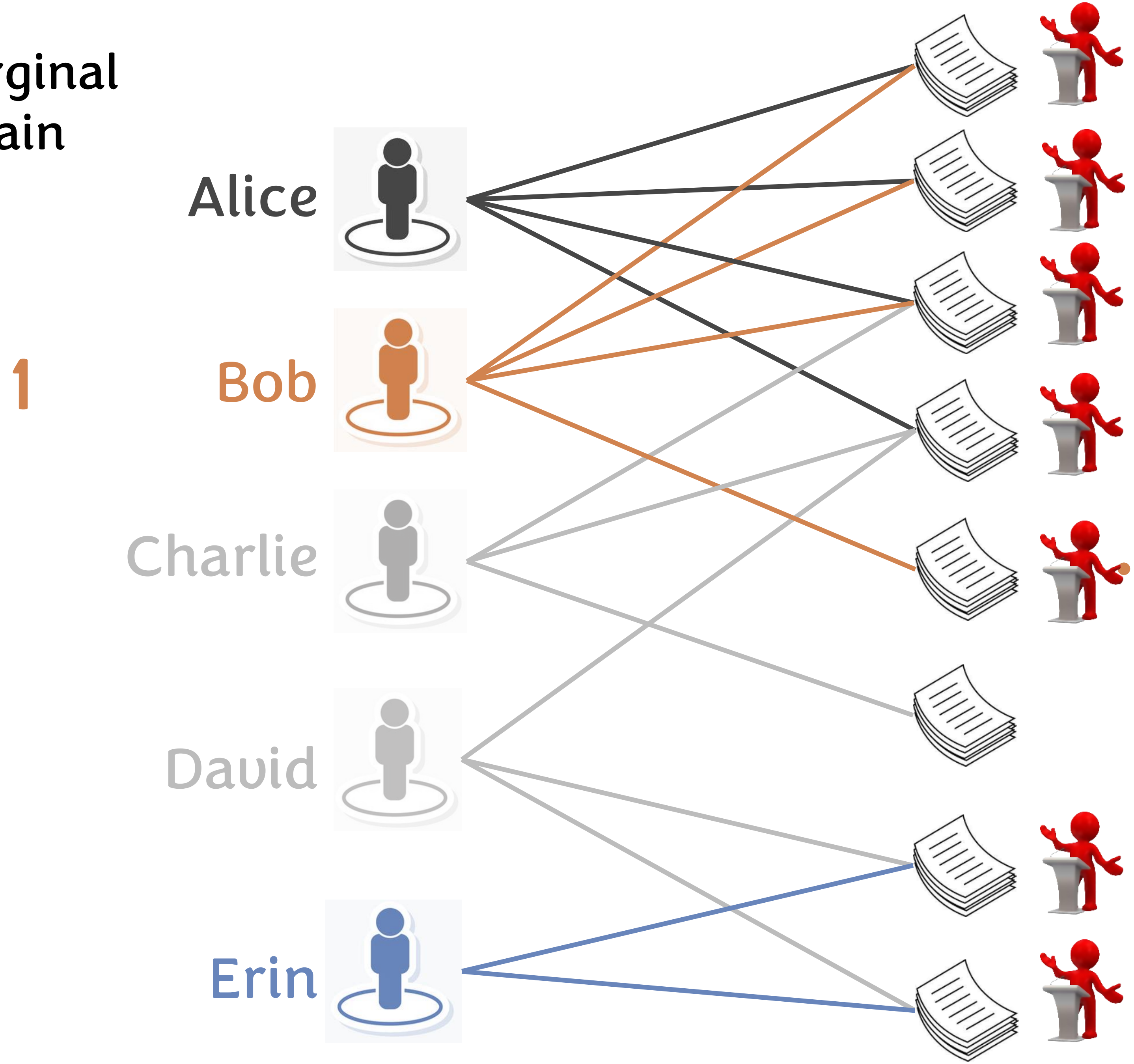
$n = 5$
$m = 8$
$k = 3$

- If we have invited Alice and Erin, who is the one with the most "marginal gain"?

- f({Alice, Erin}) = 6

How to maximize the number of presented papers?

Marginal gain

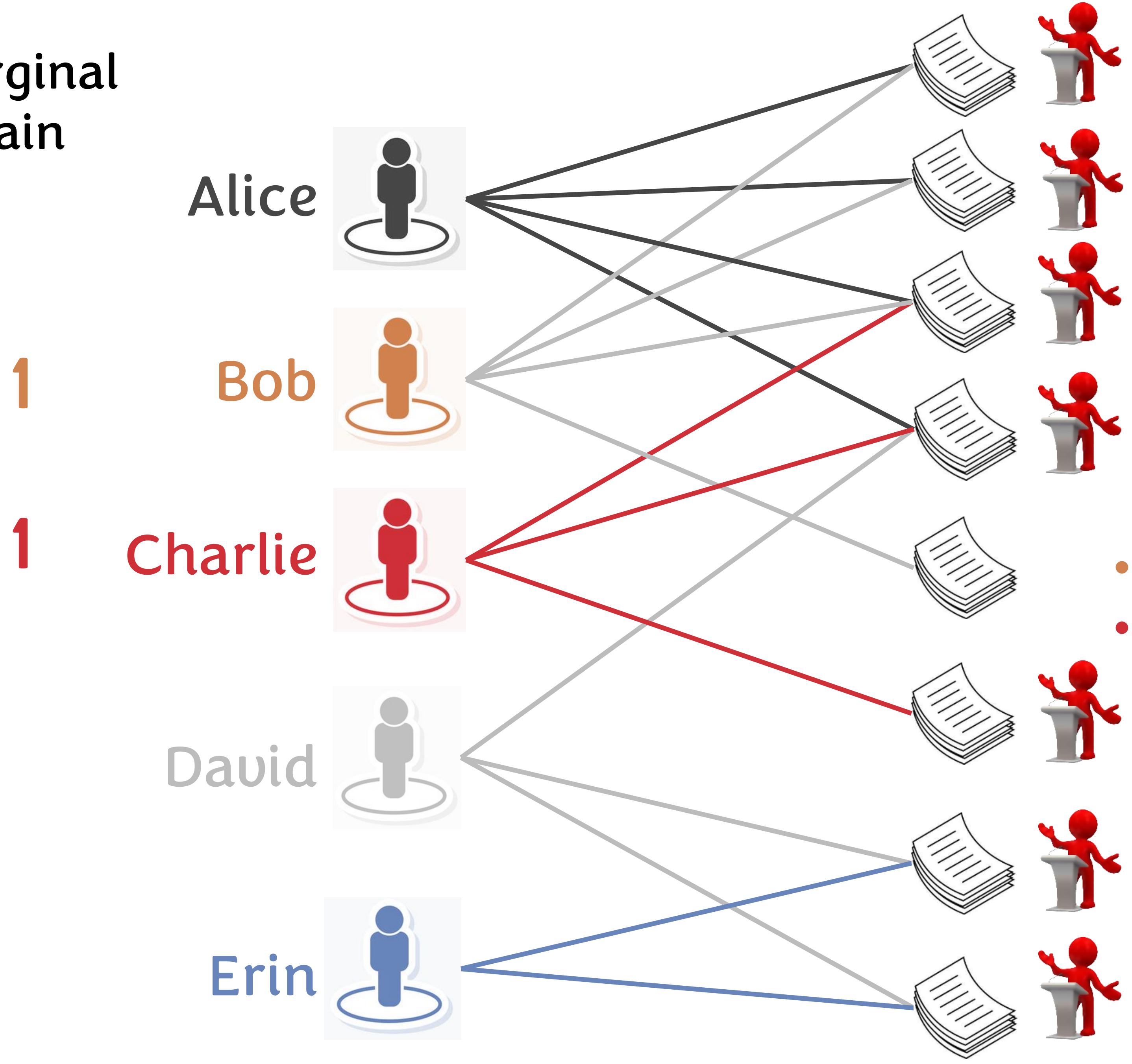


$n = 5$
$m = 8$
$k = 3$

- If we have invited Alice and Erin, who is the one with the most "marginal gain"?
- f({Alice, Erin, Bob}) = 7

How to maximize the number of presented papers?

Marginal gain



$n = 5$

$m = 8$

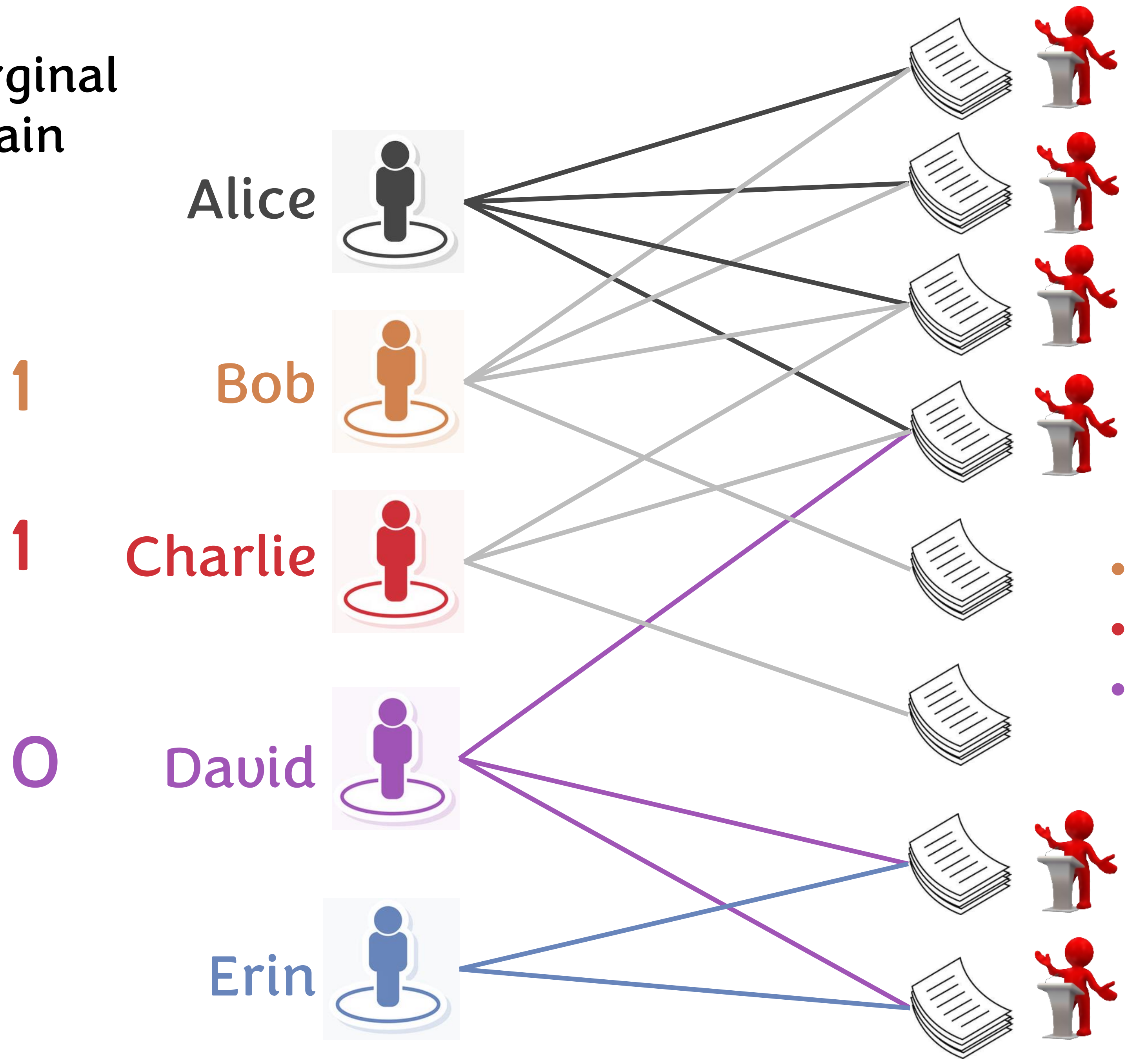
$k = 3$

- If we have invited Alice and Erin, who is the one with the most “marginal gain”?

- f({Alice, Erin, Bob}) = 7
- f({Alice, Erin, Charlie}) = 7

How to maximize the number of presented papers?

Marginal gain



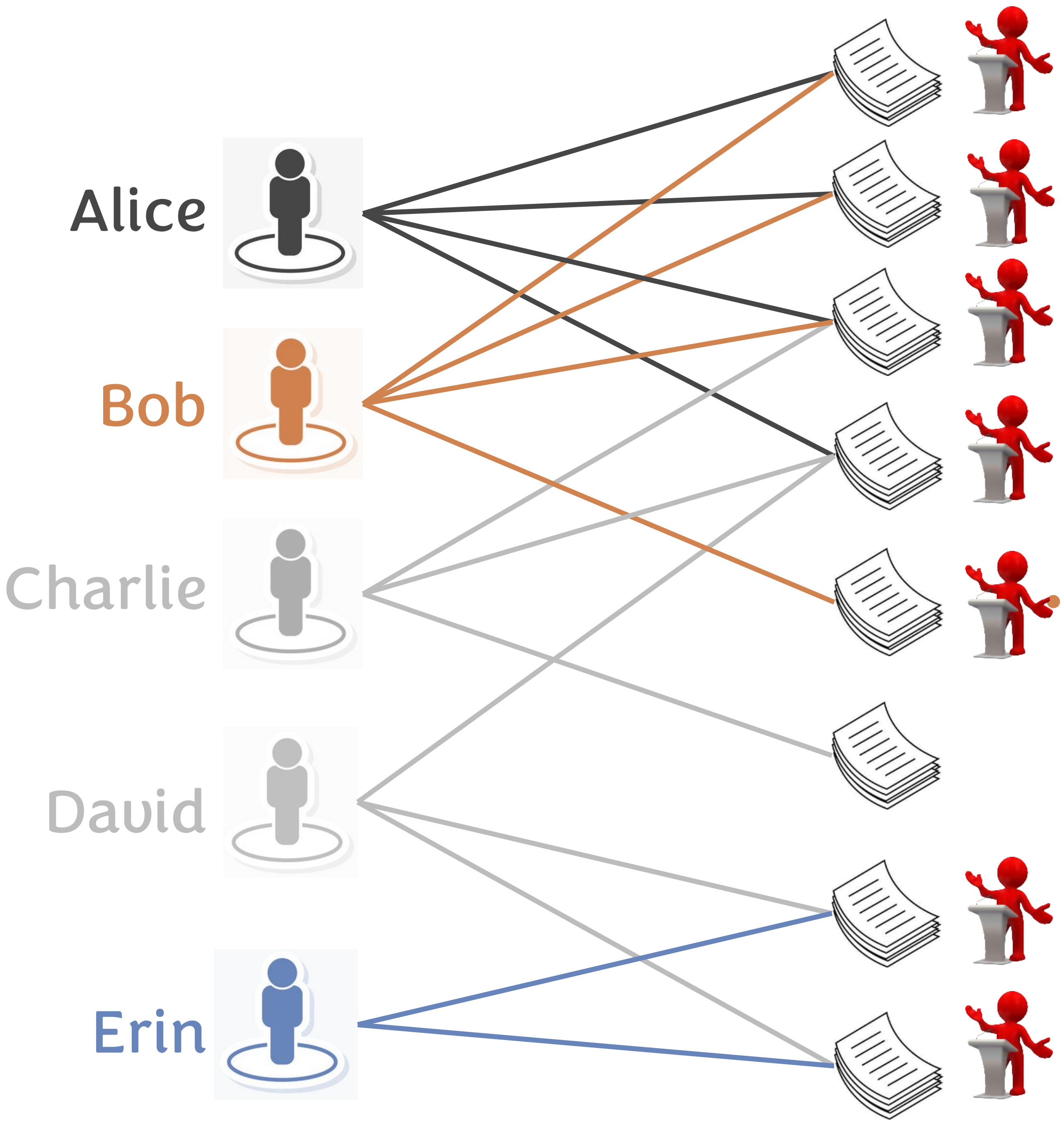
$n = 5$
$m = 8$
$k = 3$

- If we have invited Alice and Erin, who is the one with the most “marginal gain”?

- f({Alice, Erin, Bob}) = 7
- f({Alice, Erin, Charlie}) = 7
- f({Alice, Erin, David}) = 6

How to maximize the number of presented papers?

Marginal gain

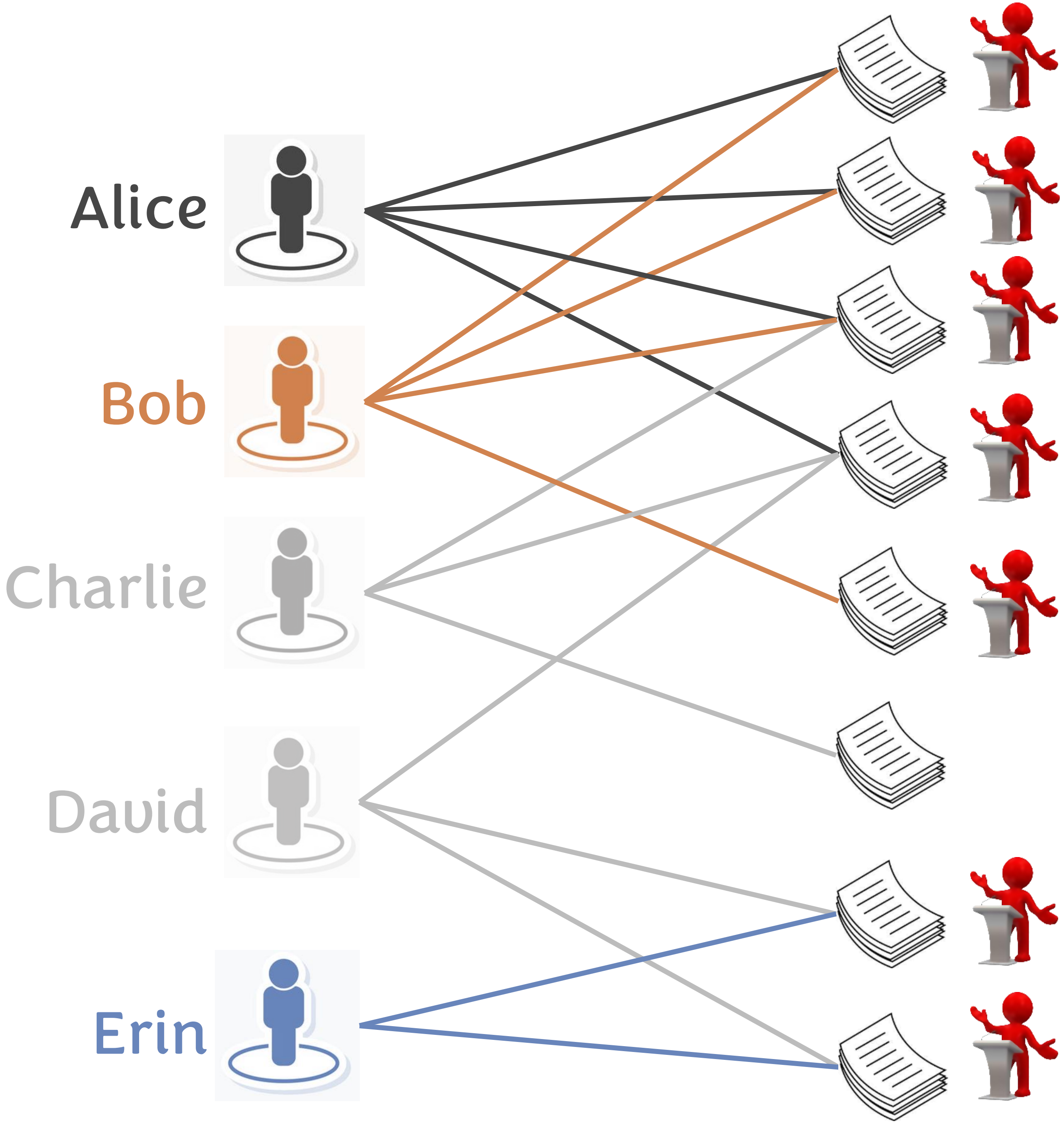


$n = 5$
$m = 8$
$k = 3$

- If we have invited Alice and Erin, who is the one with the most "marginal gain"?
- f({Alice, Erin, Bob}) = 7

How to maximize the number of presented papers?

Marginal gain



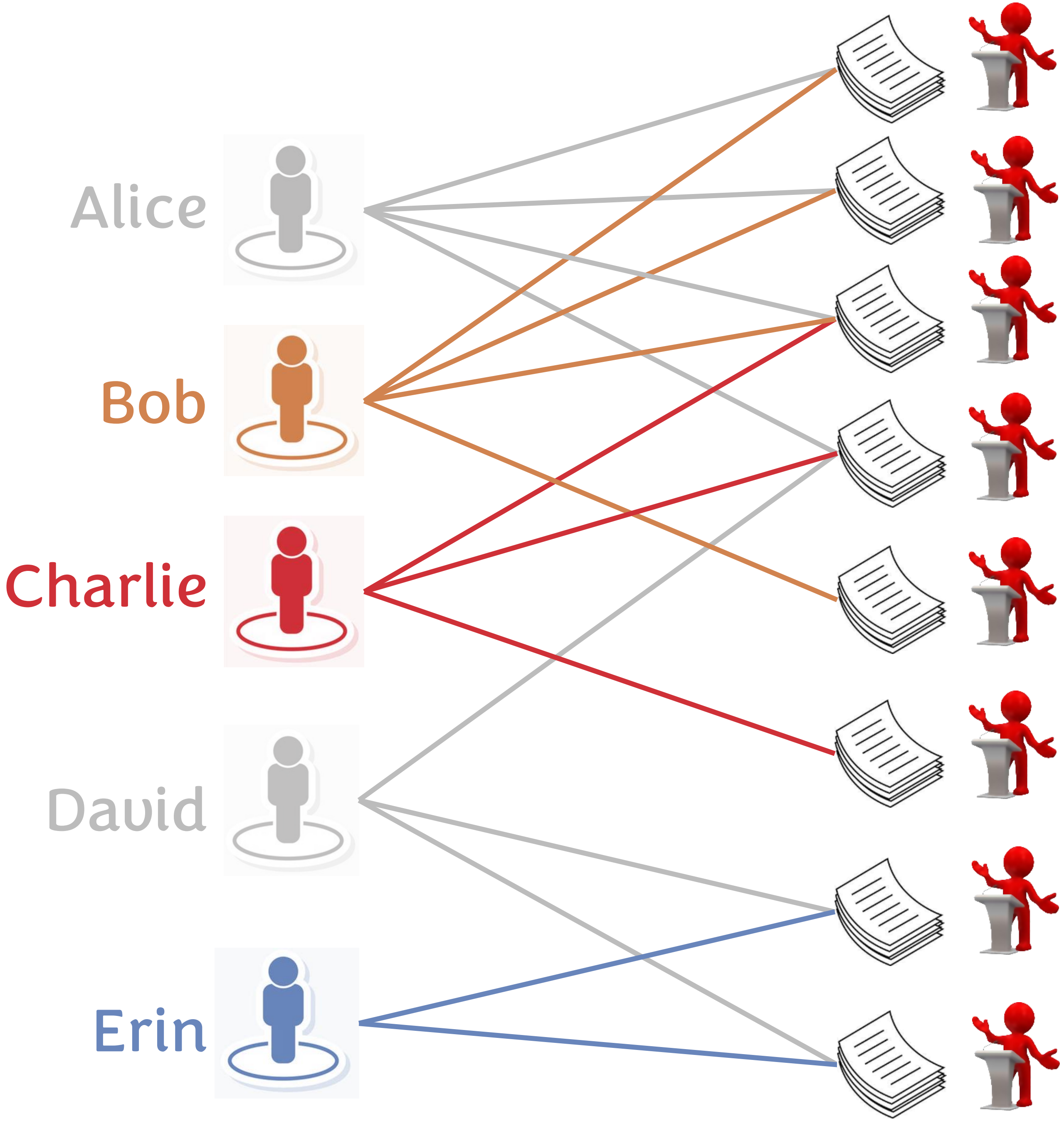
$n = 5$
$m = 8$
$k = 3$

Will "marginal gain first" give you the best solution?

- f({Alice, Erin, Bob}) = 7

How to maximize the number of presented papers?

Marginal gain



$n = 5$
$m = 8$
$k = 3$

Will "marginal gain first" give you the best solution?

Maximum Coverage Problem

- **Using the "marginal gain first" strategy, the solution is**
  - Not necessary optimal
  - But, can be **at least 63%** as good as the best solution!
  - I.e., the greedy algorithm gives you a 0.63-approximation
  - The greedy algorithm is essentially the best-possible polynomial time approximation algorithm for maximum coverage unless $P = NP$
  - This "63%" is actually $1 - \frac{1}{e}$

Submodularity

- **When your objective is a monotone submodular function, this "marginal gain first" strategy will provide you a solution that is $\left(1-\frac{1}{e}\right)$-competitive**
  - Monotone: For sets S and $T \subseteq S$, $f(T) \leq f(S)$
  - Submodular: For set S and $T \subseteq S$, and $x \notin S$,
$$f(T \cup \{x\}) - f(T) \geq f(S \cup \{x\}) - f(S)$$

  => adding an element to a smaller set will have a better marginal gain than adding it to a large set!
  - (You can try to verify this on the maximum coverage problem)
  - The greedy algorithm will find a solution that is at least (1-1/e)≈ .63 as good as optimal

Stable Matching Problem

Finding a good match

- n companies and n candidates
- Each candidate will go to one company, and each company will hire one candidate
- "preference list"

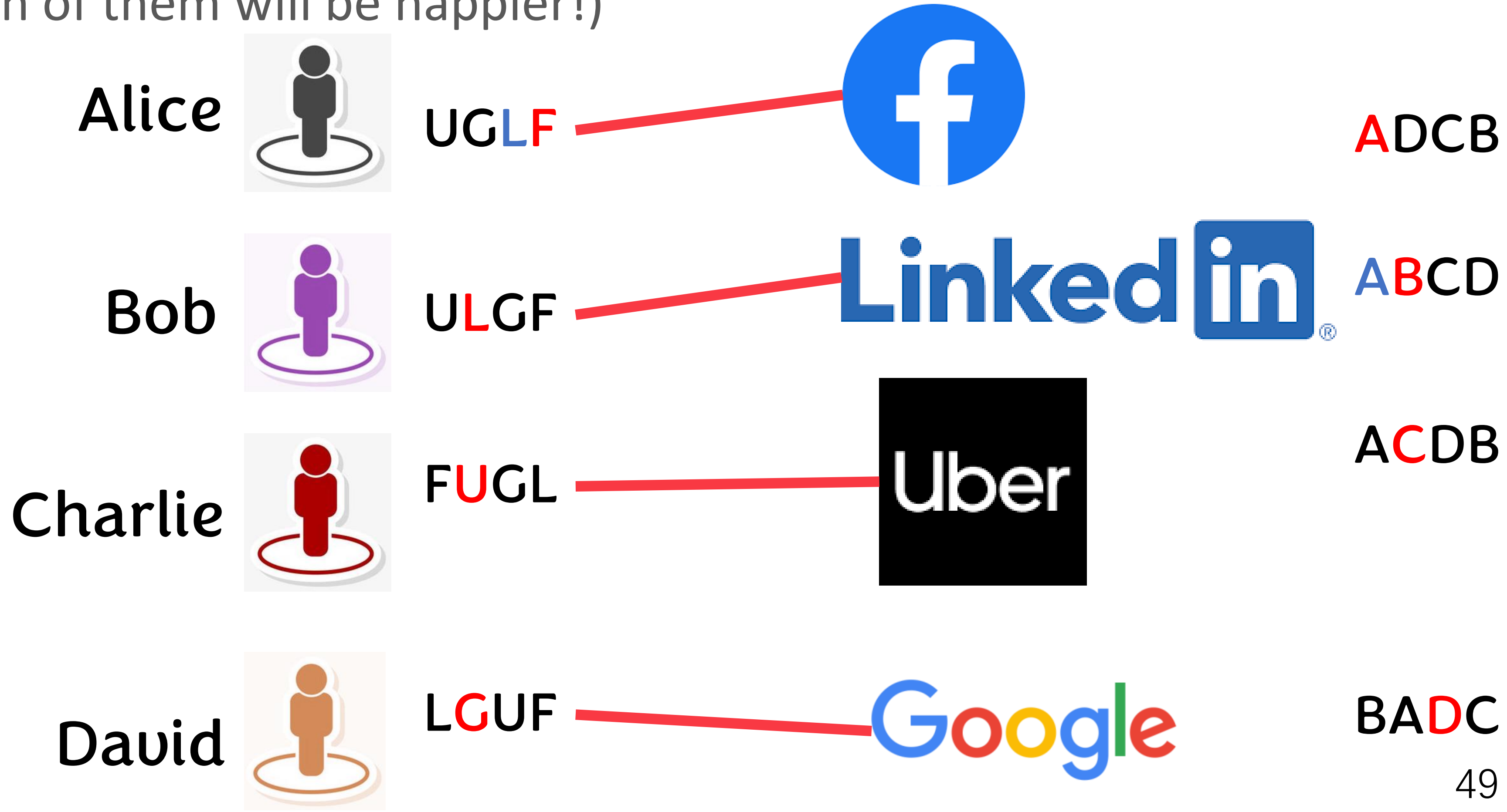
Alice	UGLF
Bob	ULGF
Charlie	FUGL
David	LGUF

Facebook	ADCB
LinkedIn	ABCD
Uber	ACDB
Google	BADC

Finding a good match

- **A stable match means that, there is no matching of:**
  - A ⇔ 1, B ⇔ 2, where A prefers 2 better than 1, and 2 prefers A better than B
  - (if A move to 2, both of them will be happier!)

An unstable matching:
- Alice prefers Linkedin better than Facebook
- Linked in prefers Alice better than Bob (current employee)
- So Alice should move to Linkedin, and Linked in should also accept Alice (replace Bob)

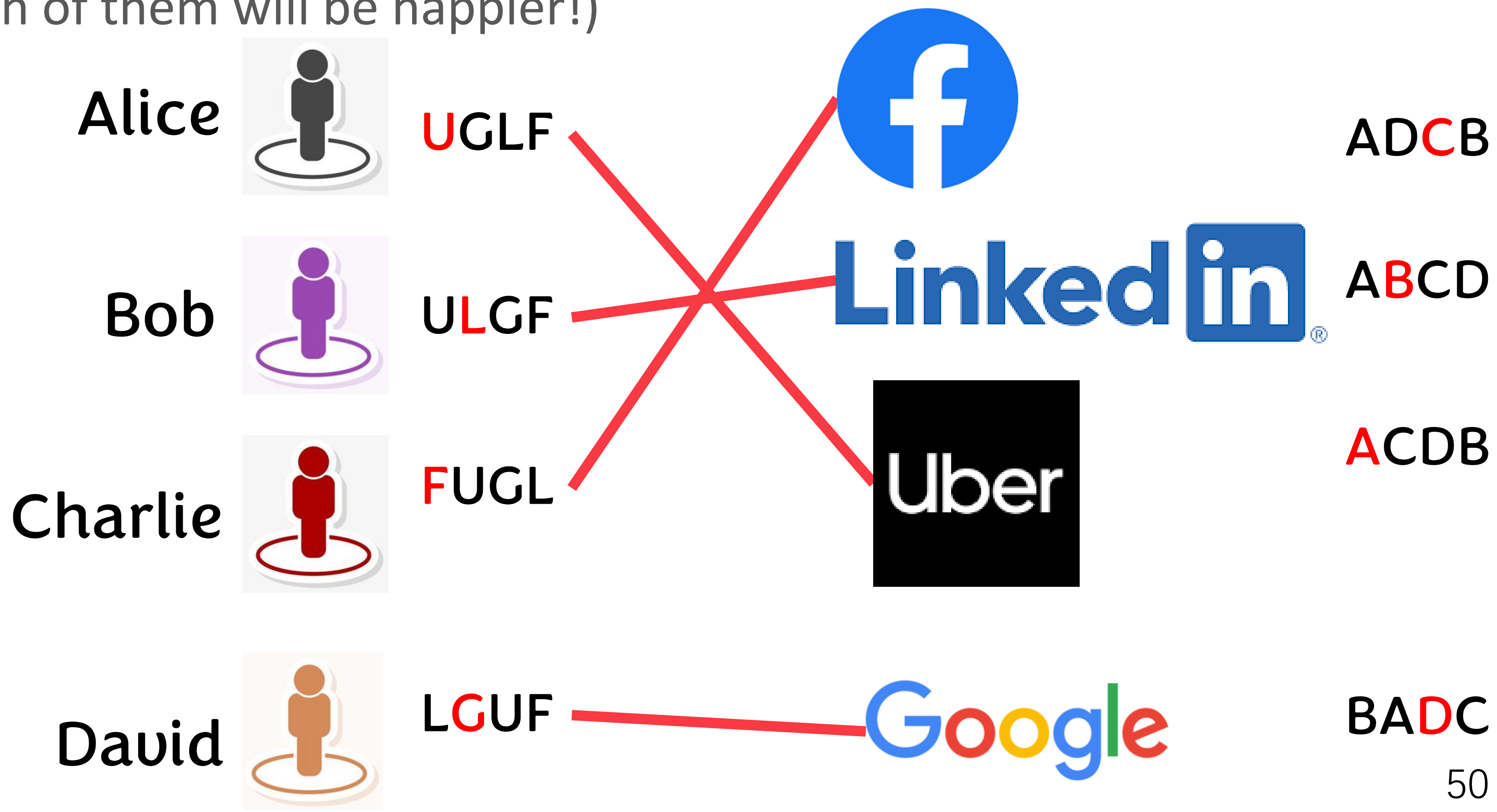


Finding a good match

- **A stable match means that, there is no matching of:**
  - A ⇔ 1, B ⇔ 2, where A prefers 2 better than 1, and 2 prefers A better than B
  - (if A move to 2, both of them will be happier!)

A stable matching:

- No one has a better choice
- E.g., Bob will work at Linkedin, he/she will be happier to be at Uber
- However, Uber got its best choice, so it will not accept Bob
- No one will move



Finding a good match (Gale-Shapley algorithm)

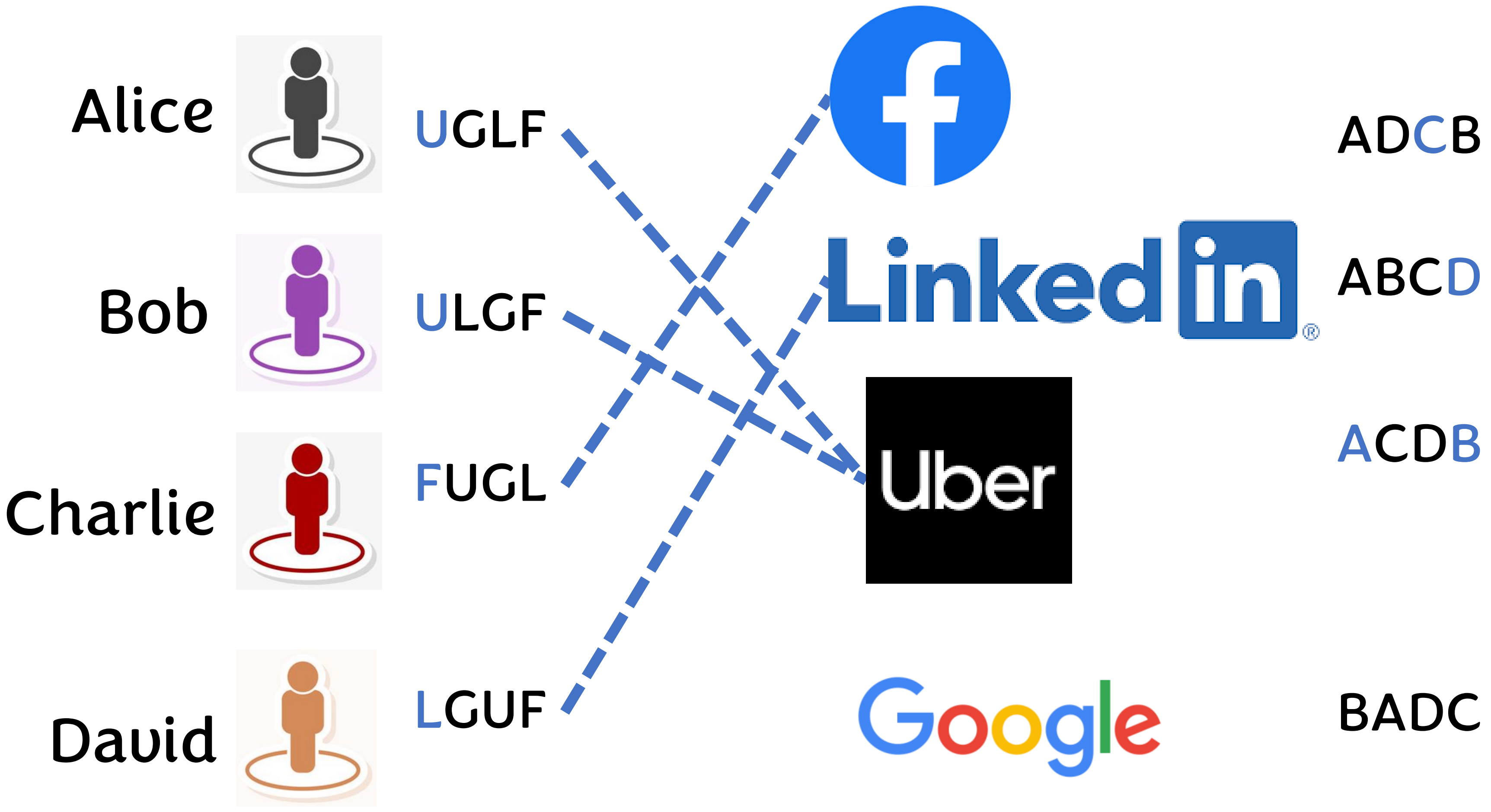
- **Round 1:**
  - All candidates go to the company on the top of its list for an interview
  - For any company:
    - No candidate comes: do nothing
    - 1 candidate: accept
    - Multiple candidate: choose the one based on its preference list. Reject all others.
  - Candidate: if got rejected, remove the company from the list.
- **Round 2 - ?**
  - All candidates goes to the company on the top of its list for an interview
  - For any company:
    - No candidate comes: do nothing
    - Given any candidate: look at the one x with the highest preference. If x is better than the current employee (or no current), replace (yes, cold-blooded employer!). Otherwise, keep the current.
  - Candidate: if got fired or rejected, remove the company from the list.

Finding a good match

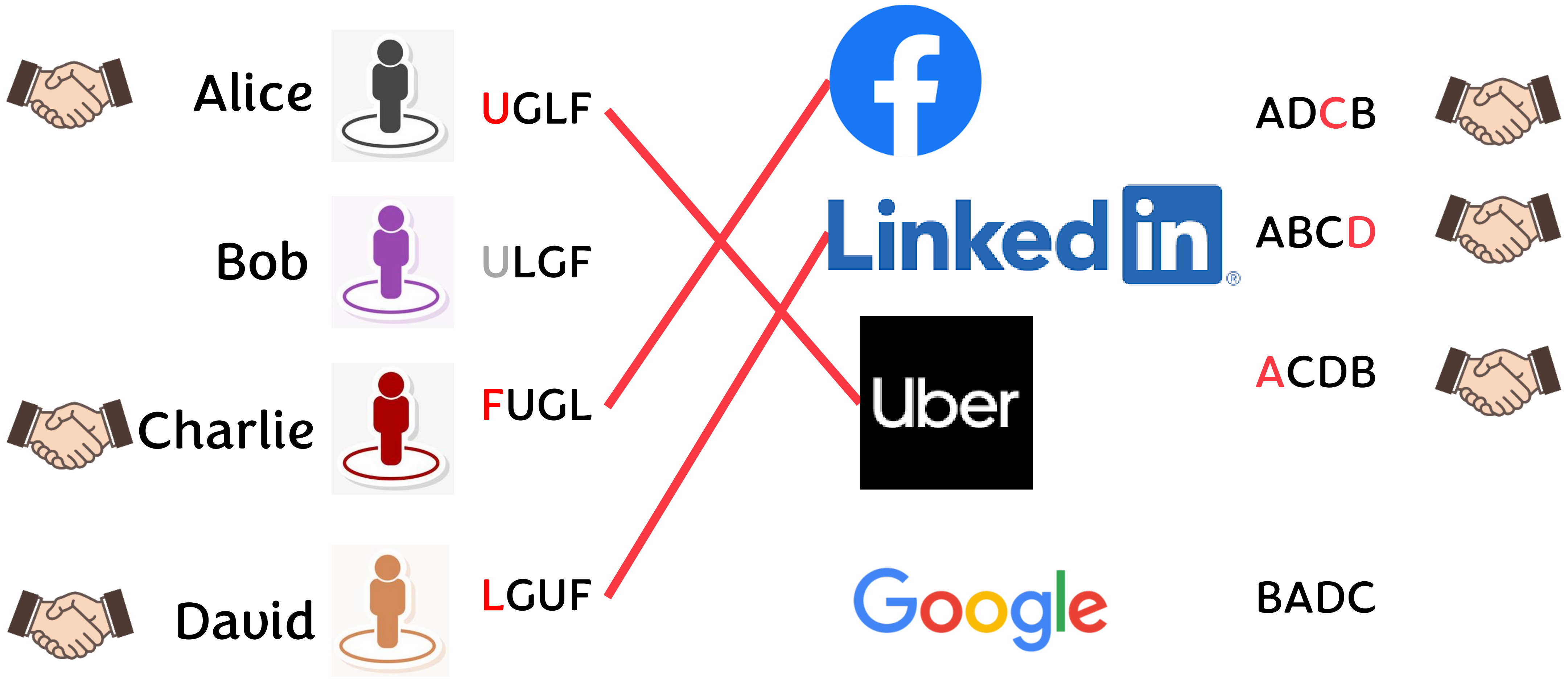
Alice	UGLF
Bob	ULGF
Charlie	FUGL
David	LGUF

Facebook	ADCB
LinkedIn	ABCD
Uber	ACDB
Google	BADC

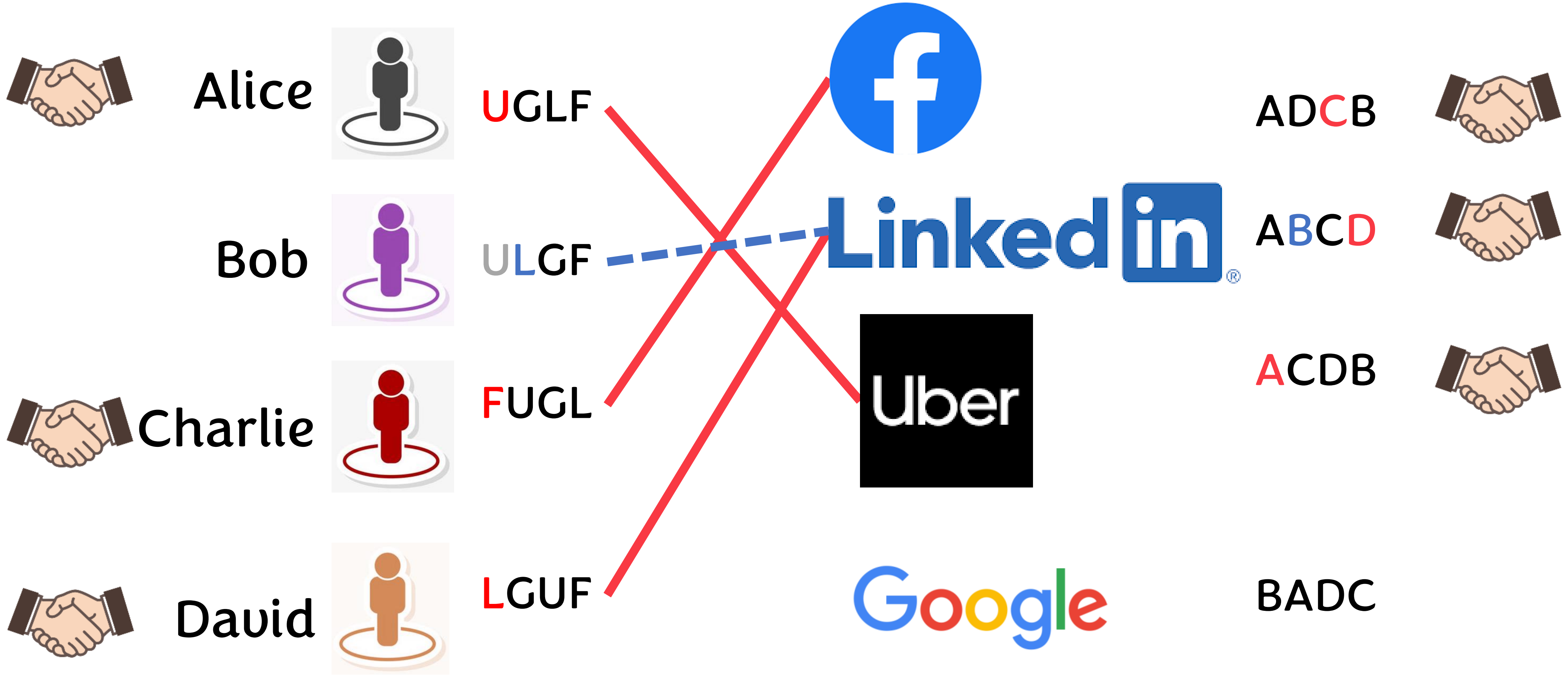
Finding a good match



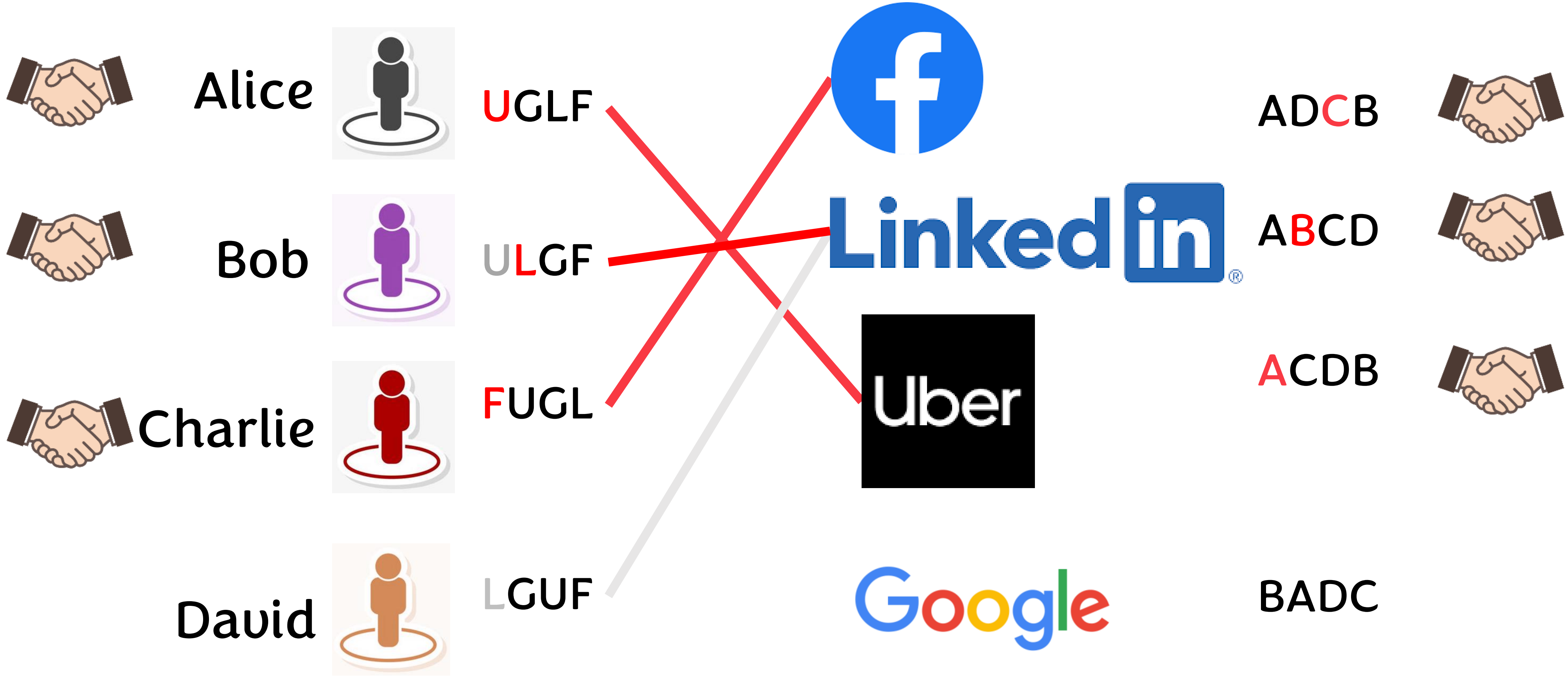
Finding a good match



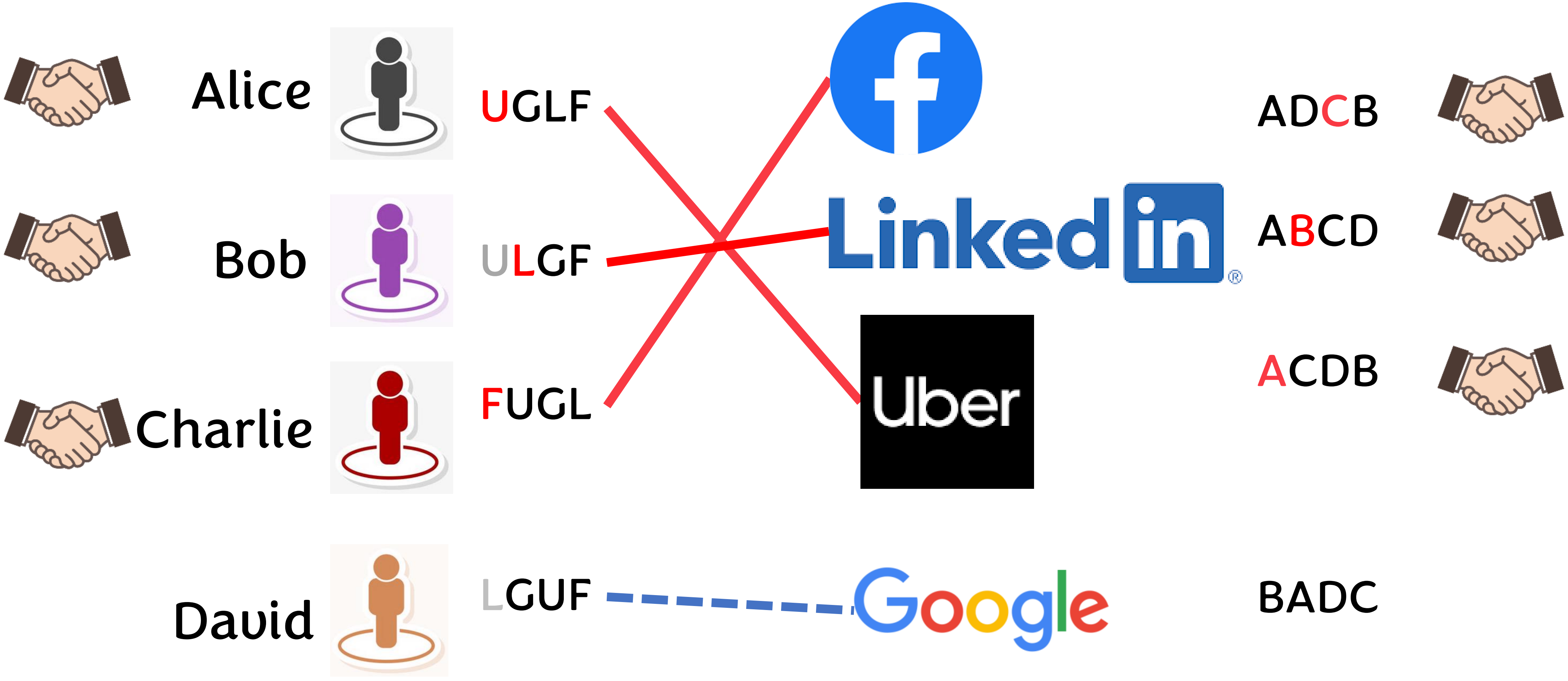
Finding a good match



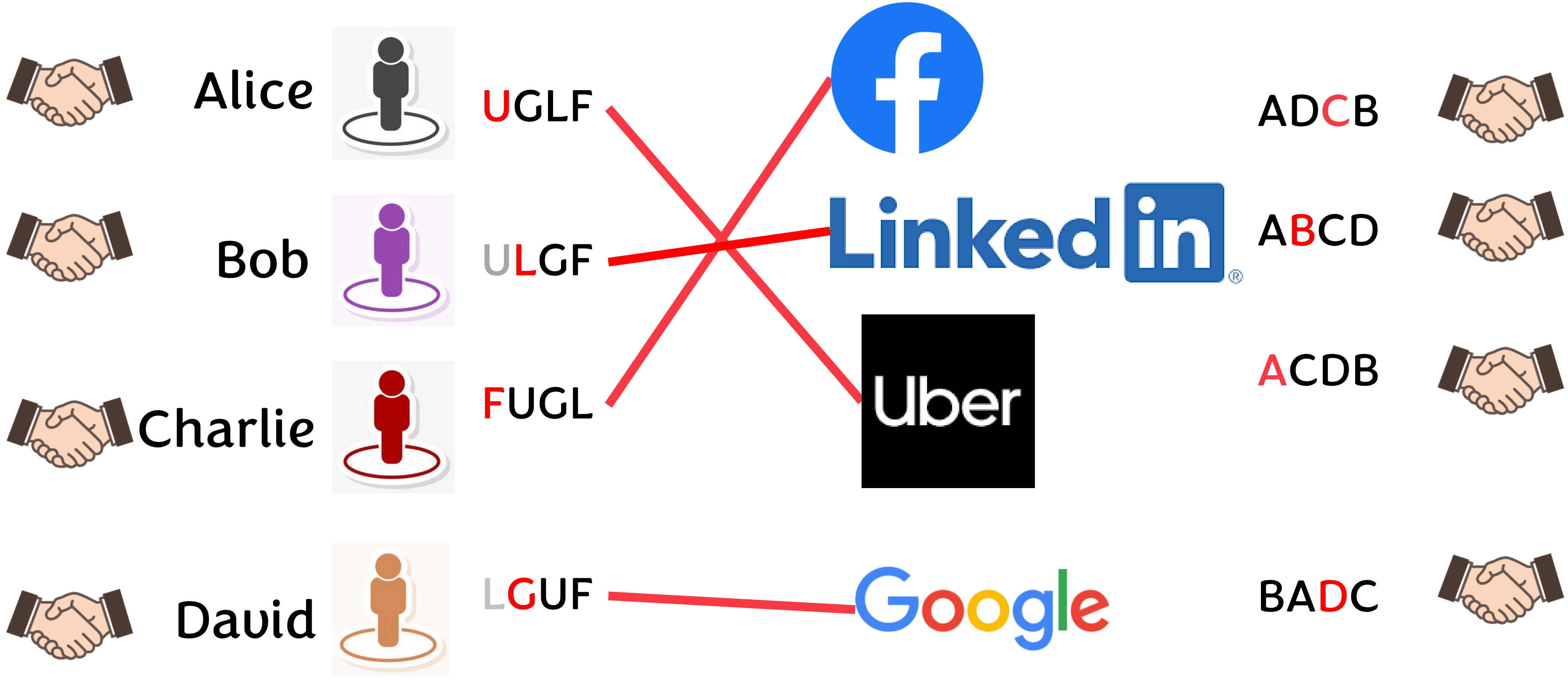
Finding a good match



Finding a good match



Finding a good match



Gale-Shapley Algorithm

David  LGUF

- **The algorithm must give you a stable matching**

First, it must stop

- In each round, either 1) all applicants have a job, which means the algorithm stops, or 2) at least one applicant will apply for a job
- We will show that, 2) can’t continue forever
  - An applicant will traverse all companies in the list, so one day s/he should have done interviews in all companies
  - For a company, as long as anyone applied for the job, the position will be taken
  - This means, when any applicant finish the list, all companies must have their employees hired
  - Contradiction!

Gale-Shapley Algorithm

Alice

Linkedin>Facebook

Bob

Alice>Bob

Second, it must be stable!

- Note that
  - A company’s employee is only getting better over time
  - An applicant’s job is only getting worse over time
- Assume ∃ Alice at Facebook and Bob at Linkedin, where Alice likes Linkedin more than Facebook, and Linkedin likes Alice more than Bob
  - So Alice must have applied for job at Linkedin before Facebook
  - If Linkedin didn’t make an offer to Alice, it must have a better choice than Alice
  - So Linkedin couldn’t end up offering the job to someone worse than Alice (Bob)
  - Contradiction!

Stable Matching Problem (Stable Marriage Problem)

- **This algorithm is usually described for matching a set of man and women, in order to find "stable marriage"**
  - There doesn't exist Alice and Bob, such that Alice like Bob more than her husband, and Bob like Alice more than his wife (otherwise? ……)
- **Gale–Shapley algorithm gives a solution to such problems**

Stable Matching Problem (Stable Marriage Problem)

- **So how does this model “marriage”?**
  - In the first round,
    - each single man proposes to the woman he prefers most, and
    - each woman accept the suitor she prefers most and say "no" to all other suitors.
    - “accept” mean “provisionally engaged” or “in relationship” – not marrying!
  - In later rounds,
    - each single man proposes to the most-preferred woman who hasn’t said “no” to him
    - each woman accept the most preferred suiter if she was single, or if her most preferred suitor is better than her current partner, update
  - Finally the process stops, that would be a stable marriage!

Stable Matching Problem (Stable Marriage Problem)

- **So how does this model "marriage"?**
  - It somehow indicates the following result for the algorithm
  - A man's partner will get worse and worse (but they have the chance to propose to their top choice)
  - A woman's partner will get better and better (but they can only choose among those that have proposed to them)
- **The algorithm is used in applicant-job matching (doctor-hospital) much earlier than it's formally studied!**
  - A simple algorithm and is easy to implement
  - Somehow people feel like it will work intuitively
  - but don't know why for a while

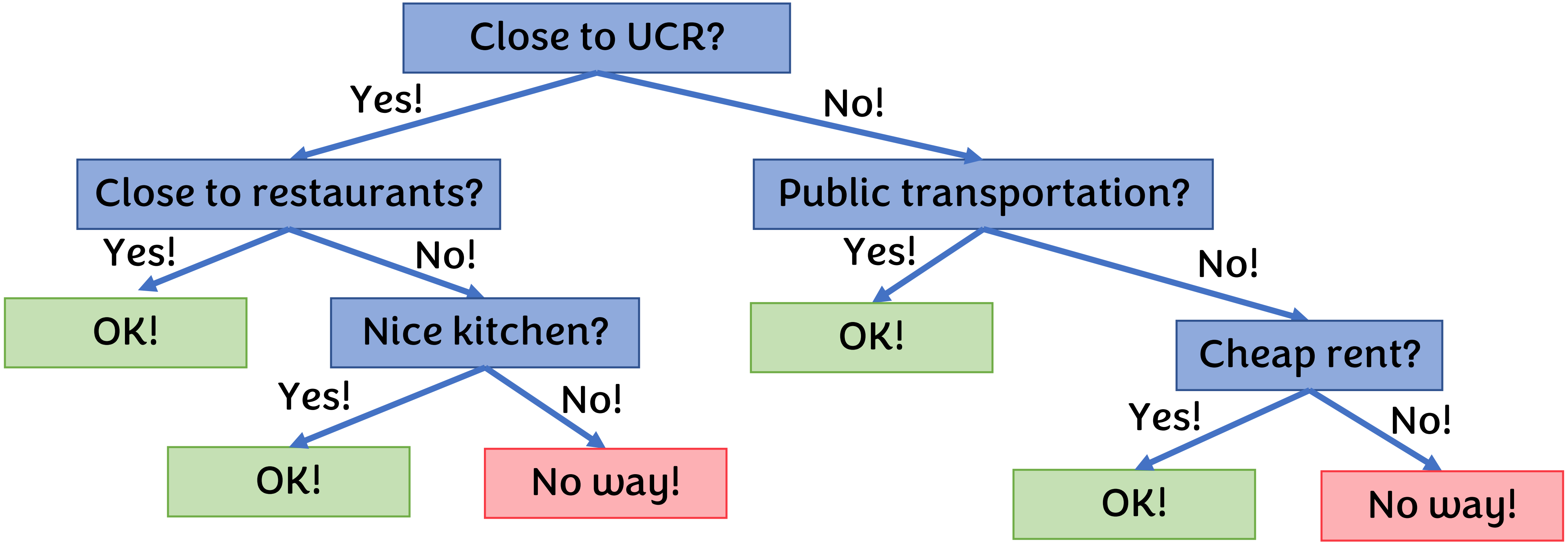
Stable Matching Problem (Stable Marriage Problem)

- **This algorithm is usually described for matching a set of man and women, in order to find "stable marriage"**
  - There doesn't exist Alice and Bob, such that Alice like Bob more than her husband, and Bob like Alice more than his wife (otherwise? ……)

- **Gale–Shapley algorithm gives a solution to such problems**
  - It does not need a centralized authority to "run" the algorithm
  - It only needs to tell each participant (or applicant/company) what the best strategy is. They will play the game automatically and get a stable solution!

Other interesting greedy algorithms

- **Decision tree: find the feature with the best “Information gain”**
  - The reduction in entropy or surprise by transforming a dataset
  - Intuitively: find a feature such that: one branch is mostly “yes” and the other is mostly “no”



Summary for today's lecture

- **How to handle NP-hard problems or other problems with their best solutions being very slow?**
- **Approximation algorithms can be useful in this case, and oftentimes we can prove the quality of the solutions**
- **Sometimes it is also theoretically interesting to understand the impossible results for approximation**
  - Take CS 215 and 219 (Spring 2023) for more details

Next lecture...

- Data structure I: tournament trees and augmented trees