

Welcome to Advanced Algorithms

Lecturer: Shay Mozes (smozes@idc.ac.il)

Office Hours: Sunday 14:00-15:00
and by appointment

Fall 2014

Administrivia

- Most important resource: course web page
- Papers and sections from various books (mainly CLRS).
- Grading:
 - Homework - 25%
 - Final exam - 75% (must get at least 60).

About the Homework

- Every 2-3 weeks a homework exercise
- Some Hard questions - significant effort
- May use any resource: friends, papers, books; but **must write by yourself and acknowledge / specify working group.**
- Every student must hand in every exercise.
- Average of best $n-1$ from the HW component in the final grade. Probably $n=6$.
- 5 point bonus for HW written in LaTeX.

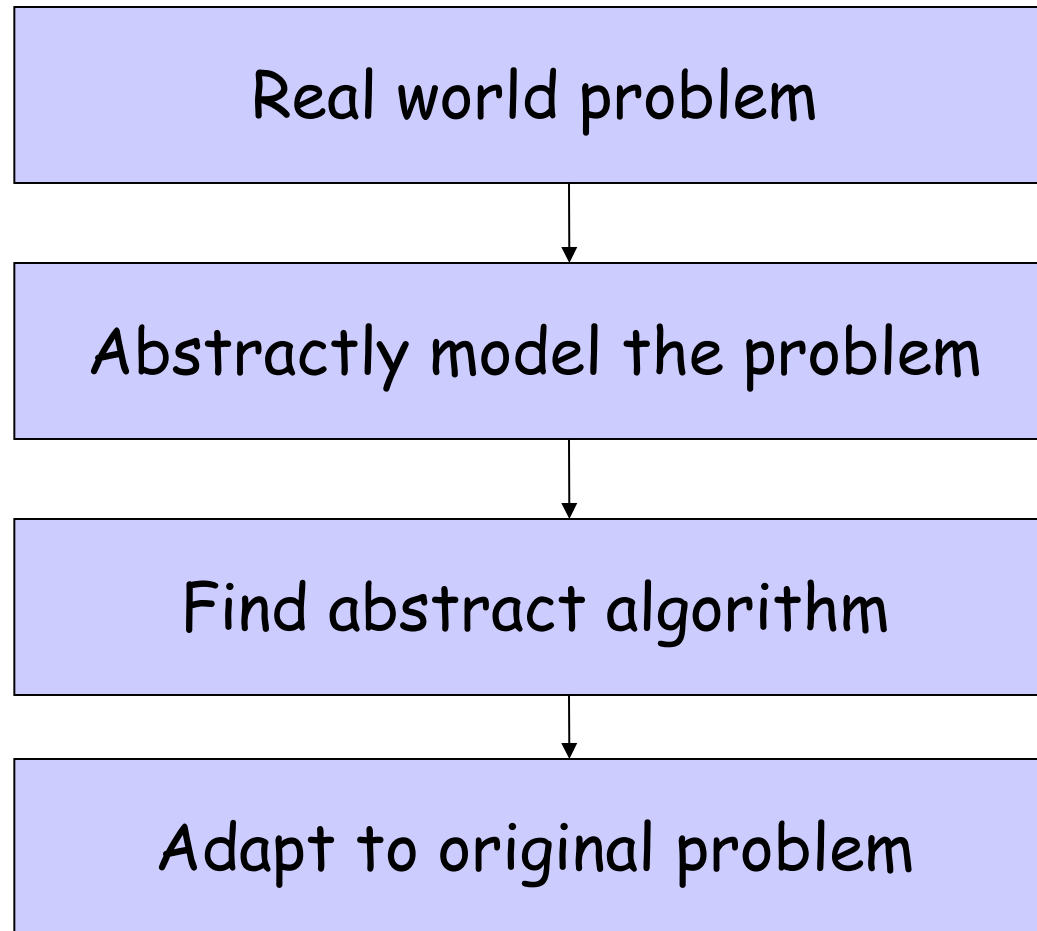
Course Goals

- A large algorithmic toolbox.
- A deeper understanding of the issues and tradeoffs involved in algorithm design.
- An appreciation for applications of algorithmic techniques in the real world.
- A better sense of how to model problems you encounter as well-known algorithmic problems.
- Fun! Elegance! Beauty!

Some Topics

- Classic techniques - greedy, divide and conquer, dynamic programming
- Approximation algorithms
- Solving NP hard problems on structured inputs
- Fixed parameter tractability
- Linear programming, LP duality and applications
- Online algorithms
- Streaming algorithms
- Randomized algorithms
- External memory / cache oblivious algorithms

Applied Algorithm Scenario



Modeling

- Formalize the goal
- What kind of algorithm is needed?
- Can I find an algorithm or do I have to invent one?
- Can I 'tune' an existing algorithm? Does it remind me a familiar problem?

Algorithm Design Goals

- Correctness
- Efficiency
- Simple, if possible.
- Ease of implementation

Evaluating an algorithm

Mike: My algorithm can sort 10^6 numbers in 3 seconds.

Bill: My algorithm can sort 10^6 numbers in 5 seconds.

Mike: I've just tested it on my new Intel core duo.

Bill: I remember my result from my undergraduate studies (1985).

Mike: My input is a random permutation of $1..10^6$.

Bill: My input is the sorted output, so I only need to verify that it is sorted.

Types of complexity

- Actual running time is not necessarily a good measure
- We need a 'stable' measure, independent of the implementation.
- * A complexity function is a function $T: \mathbb{N} \rightarrow \mathbb{N}$.
 $T(n)$ is the number of operations the algorithm does on an input of size n .
- * We can measure (at least) three different things.
 - Worst-case complexity
 - Best-case complexity
 - Average-case complexity

The RAM Model of Computation

- Each simple operation (e.g., C statement) takes 1 time step.
- Loops and subroutines are not simple operations.
- Each memory access takes one time step, and there is no shortage of memory.

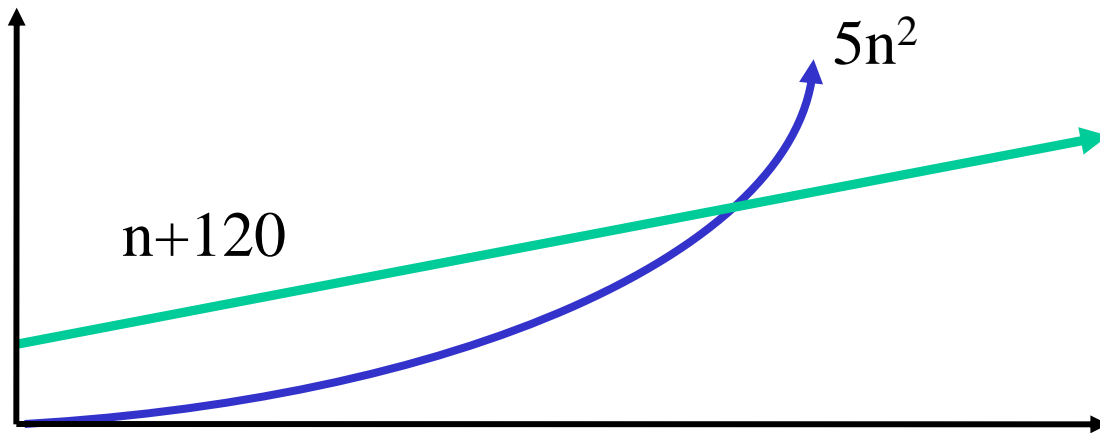
For a given problem instance:

- Running time of an algorithm = # of simple operations.
- Space used by an algorithm = # RAM memory cells

useful abstraction \Rightarrow captures the architecture of modern computers, but still allows us to analyze algorithms in a machine independent fashion.

Big O Notation

- **Goal :**
 - A stable measurement independent of the machine.
- **Way:**
 - ignore constant factors. Consider just the leading term.
- $f(n) = O(g(n))$ if $c \cdot g(n)$ is **upper bound** on $f(n)$
 - \Leftrightarrow There exist c, N , s.t. for any $n \geq N$, $f(n) \leq c \cdot g(n)$



$$\text{For all } n \geq 5 \quad n+120 \leq 5n^2$$

$$\Rightarrow n+120 = O(n^2).$$

$$\text{Also, for all } n \geq 60$$

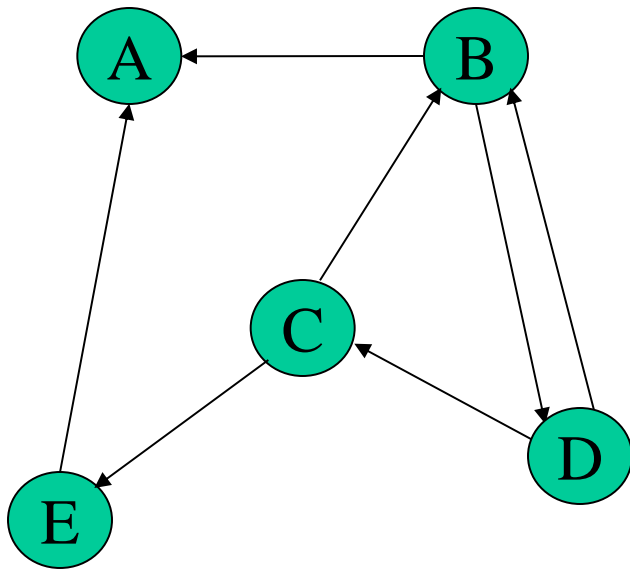
$$n+120 \leq 2n$$

$$\Rightarrow n+120 = O(n).$$

Growth Rates

- Even by ignoring constant factors, we can get an excellent idea of whether a given algorithm will be able to run in a reasonable amount of time on a problem of a given size.
- The "big O " notation and worst-case analysis are tools that greatly simplify our ability to compare the efficiency of algorithms.
- $O(n)$ $O(\log n)$ $O(2^n)$

Reminder : Graphs



- $G=(V,E)$
- $|V|=n, |E|=m$
- Directed/undirected
- Weighted/unweighted
- In/out-degree

Graph Search

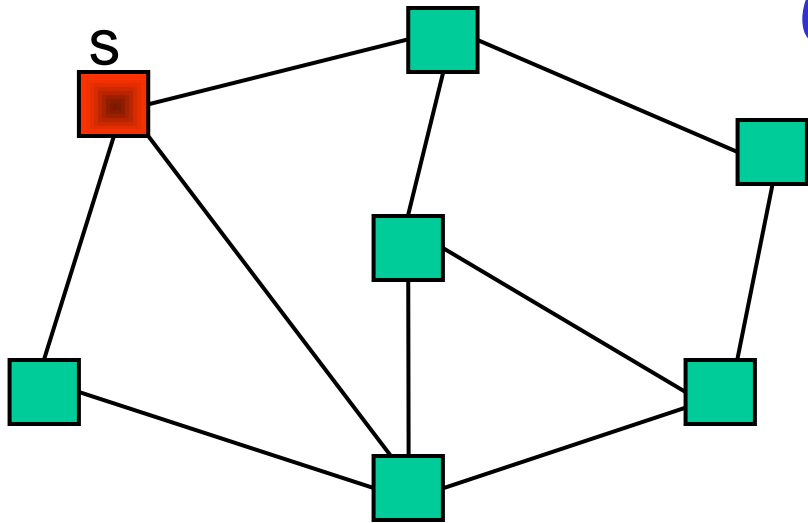
Input: Graph G , vertex s

Output: All vertices connected from s .

Two main approaches:

1. Breadth First Search
2. Depth First Search

Graph Search



DFS(G,s):

mark s

for all v neighbor of s

if v unmarked

DFS(G,v)

BFS(G,s):

mark s

F.enqueue(s)

Repeat

$u \leftarrow$ F.dequeue()

for all v neighbor of u

if v not marked

mark v

F.enqueue(v)

Until F is empty

Shortest-path Algorithms

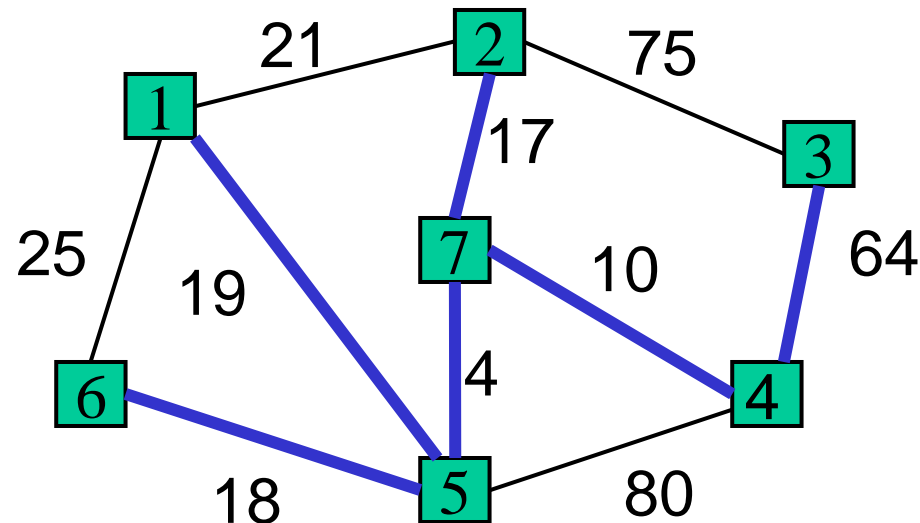
- **Single source**: given a vertex s , find the shortest path from s to any other vertex of G .
- **Variants**:
 - Different edges have different lengths (representing delay, cost, etc.)
 - Nonnegative/real weights. Negative cycles.
- **All-pair** shortest path problem: no specific source.

Shortest-path algorithms - Summary

- Single source, no weights:
BFS - $O(m)$
- Single source, non-negative weights:
Dijkstra $O(m + n \log n)$ or $O(n^2)$
- Single source, arbitrary weights:
Bellman-Ford: $O(nm)$
- All-pair shortest paths, arbitrary weights:
Floyd: $O(n^3)$, Johnson: $O(nm + n^2 \log n)$

Minimum Spanning Tree

- Each edge has a cost.
- Find a minimal-cost subset of edges that will keep the graph connected. (must be a ST).



Price of this tree = $18+19+4+10+17+64$

Minimum Spanning Tree Problem

- **Input:** Undirected connected graph $G = (V, E)$ and a cost function C from E to the reals. $C(e)$ is the cost of edge e .
- **Output:** A spanning tree T with minimum total cost. That is: T that minimizes

$$C(T) = \sum_{e \in T} C(e)$$

- **Another formulation:** Remove from G edges with maximal total cost, but keep G connected.

Two Popular algorithms:

- **Kruskal**

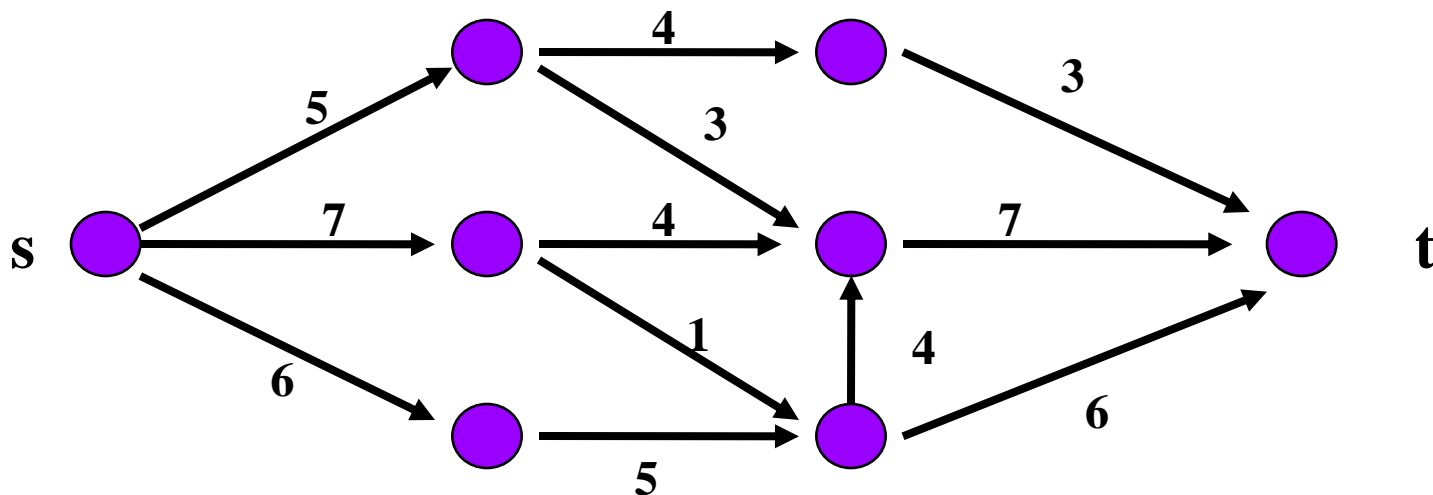
- Greedily add lightest edge that does not create a cycle
- Time complexity $O(m \log m)$ using basic sorting and Union-Find data structures

- **Prim**

- Grow a tree by greedily adding the lightest edge with one endpoint in the tree and one not in the tree
- Time complexity $O(m \log m)$ using binary heap, $O(m + n \log n)$ using Fibonacci heaps.

Maximum Flow

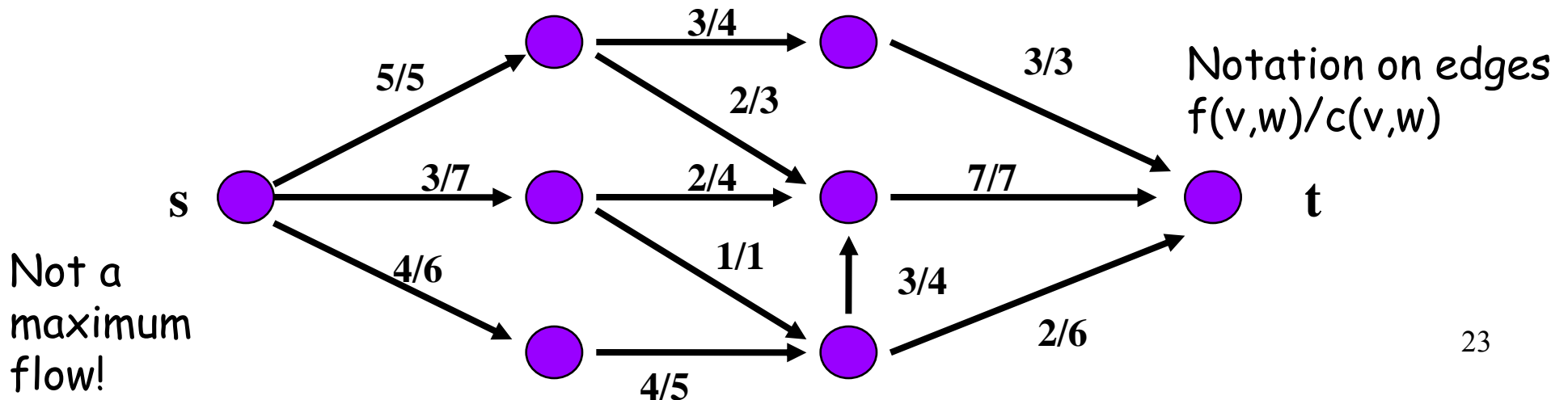
- **Input:** a directed graph (network) G
 - each edge (v,w) has associated capacity $c(v,w)$
 - a specified **source node** s and **target node** t
- **Problem:** What is the maximum flow you can route from s to t while respecting the capacity constraint of each edge?



Properties of Flow:

$f(v,w)$ - flow on edge (v,w)

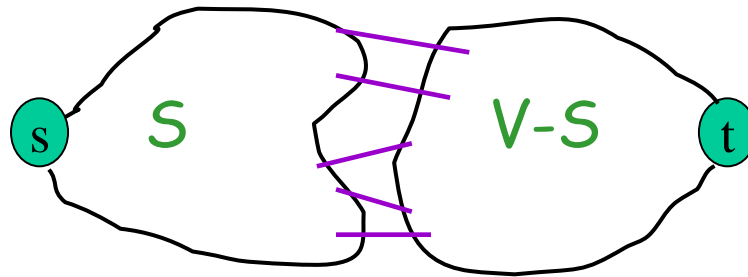
- **Edge condition (capacity):** $0 \leq f(v,w) \leq c(v,w)$: the flow through an edge cannot exceed the capacity of an edge.
- **Vertex condition (conservation):** for all v except s,t : $\sum_u f(u,v) = \sum_w f(v,w)$ the total flow entering a vertex is equal to total flow exiting this vertex.
- total flow **leaving s** = total flow **entering t** .



Cut

st-Cut - a set of edges that separates s from t .

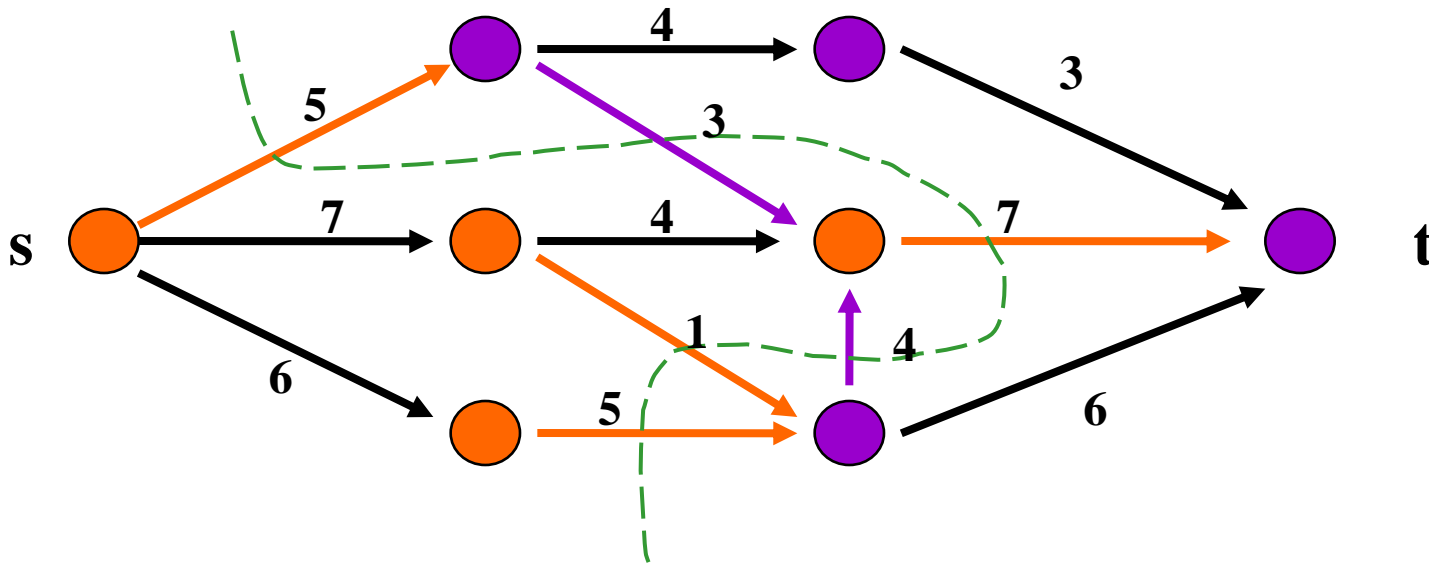
- A cut is defined by a set of vertices, S . This set includes s and maybe additional vertices. The sink t is not in S .
- The cut is the set of edges (u,v) such that $u \in S$ and $v \notin S$, or $v \in S$ and $u \notin S$.



out(S) - edges in the cut directed from S to $V-S$

in(S) - edges in the cut directed from $V-S$ to S

Cut - example



S - set of orange vertices.

$out(S)$ - orange edges

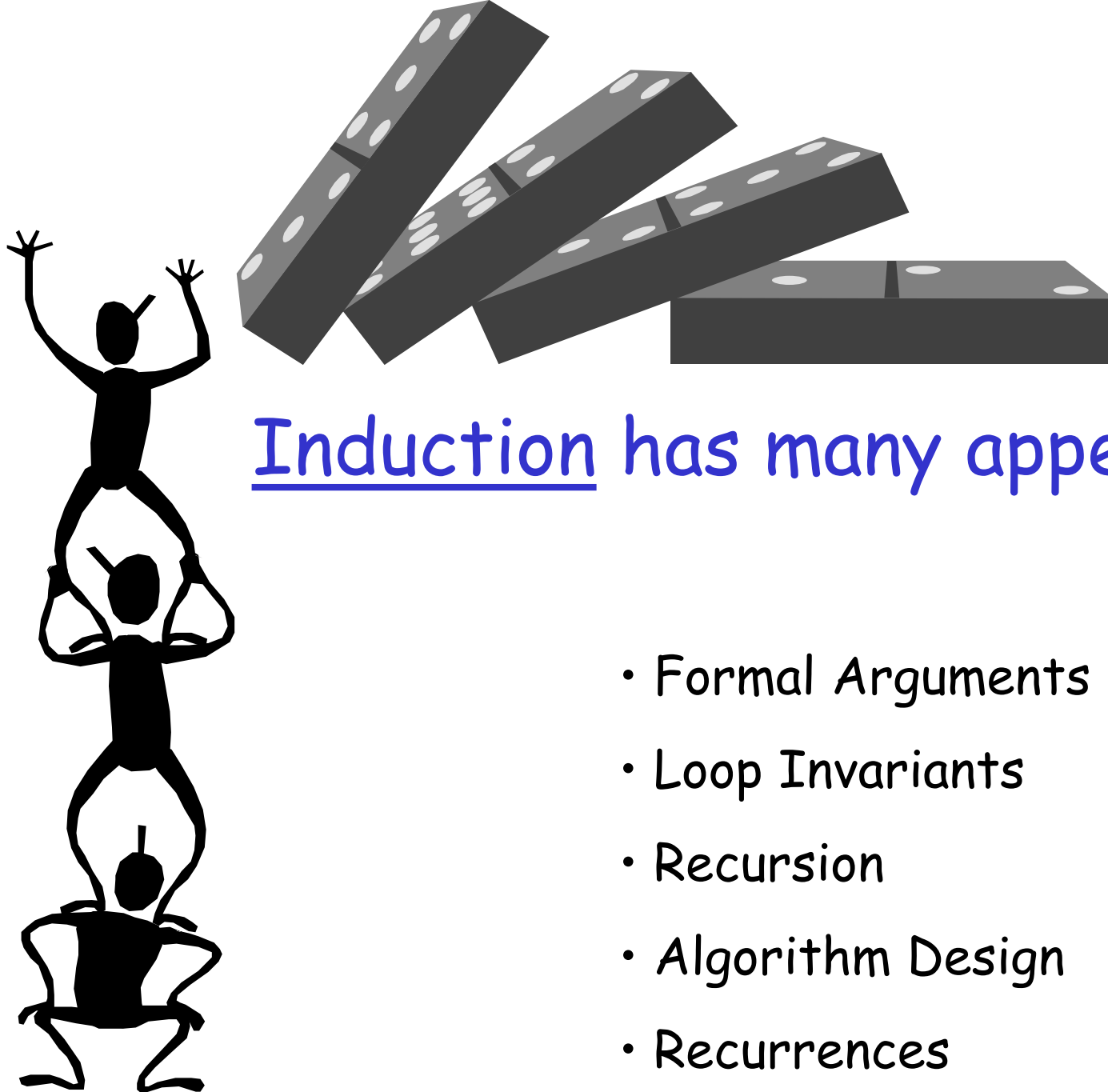
$in(S)$ - purple edges

The capacity of this cut = 18

For a cut S , the *capacity of S* is $c(S) = \sum_{e \in out(S)} c(e).$

Max-flow Min-Cut Theorem

The value of a maximum flow in a network is equal to the minimum capacity of a cut.



Induction has many appearances.

- Formal Arguments
- Loop Invariants
- Recursion
- Algorithm Design
- Recurrences

Review: Induction

- If
 - $P(k)$ is true for fixed constant k
 - Often $k = 0$
 - $P(n) \rightarrow P(n+1)$ for all $n \geq k$
- Then $P(n)$ is true for all $n \geq k$

Proof By Induction

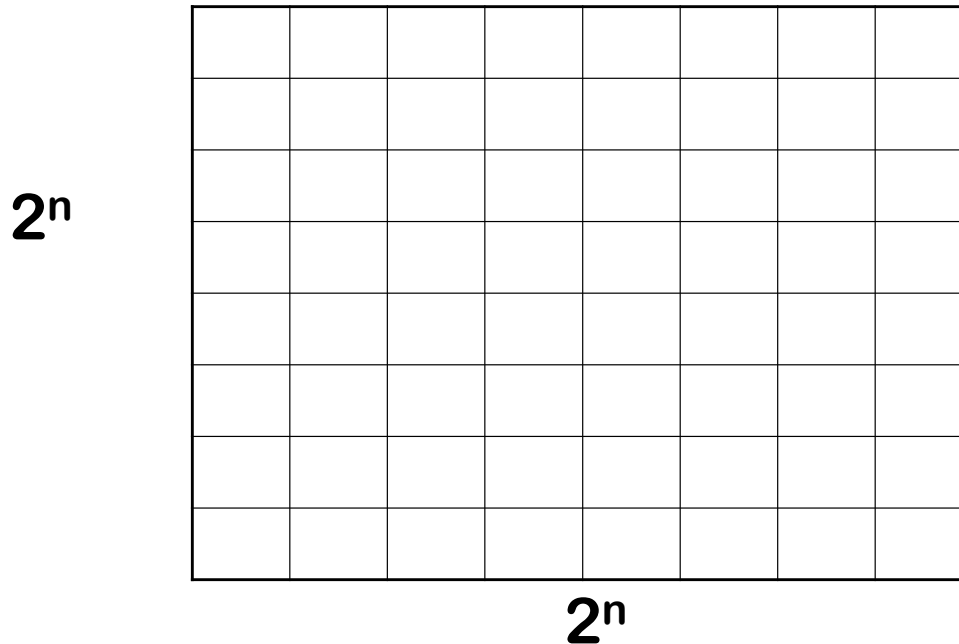
- **Claim:** $P(n)$ is true for all $n \geq k$
- **Base:**
 - Show $P(n)$ is true for $n = k$
- **Inductive hypothesis:**
 - Assume $P(n)$ is true for an arbitrary n
- **Step:**
 - Show that $P(n)$ is then true for $n+1$

Induction Example: Geometric sequence

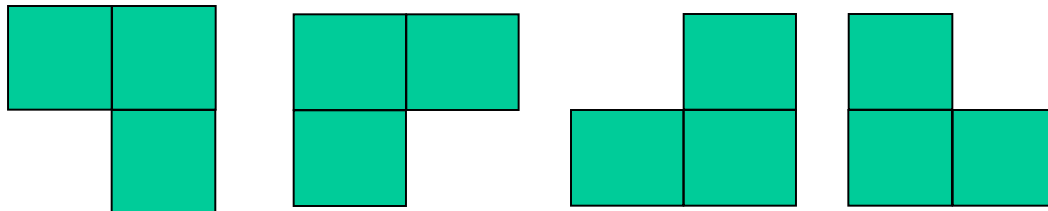
- Prove by induction on n : for all $a \neq 1$
- $a^0 + a^1 + \dots + a^n = (a^{n+1} - 1)/(a - 1)$
- Base: $n=0$. $a^0 = (a^{0+1} - 1)/(a - 1)$.
 $a^0 = 1 = (a^1 - 1)/(a - 1)$
- Inductive hypothesis:
 - Assume $a^0 + a^1 + \dots + a^n = (a^{n+1} - 1)/(a - 1)$
- Step (show true for $n+1$):
 $a^0 + a^1 + \dots + a^{n+1} = a^0 + a^1 + \dots + a^n + a^{n+1}$
 $= (a^{n+1} - 1)/(a - 1) + a^{n+1} = (a^{n+1+1} - 1)/(a - 1)$

Design by Induction Example: Tiling

Goal: tile a room of $2^n \times 2^n$ squares.



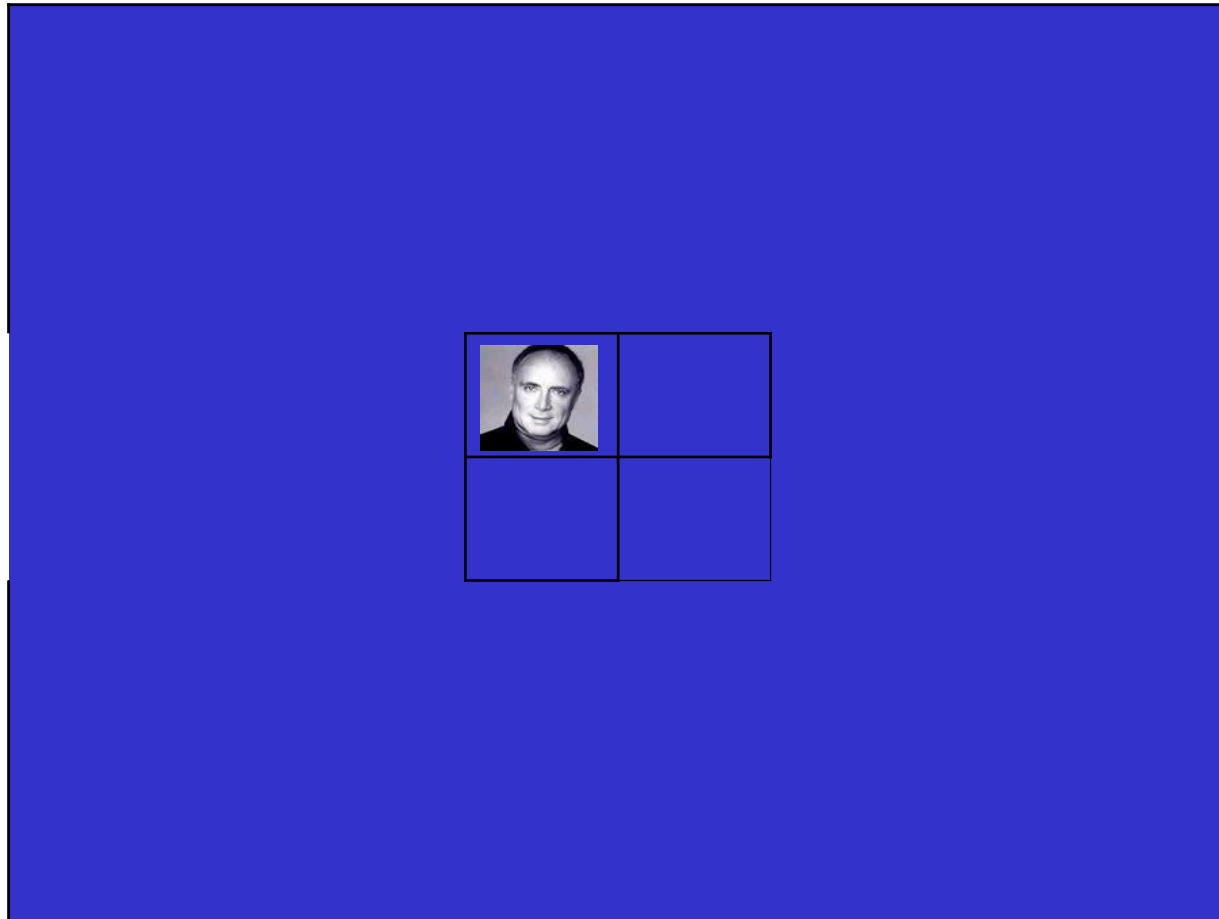
Architect allows only L-shaped tiles covering 3 squares



Tiling

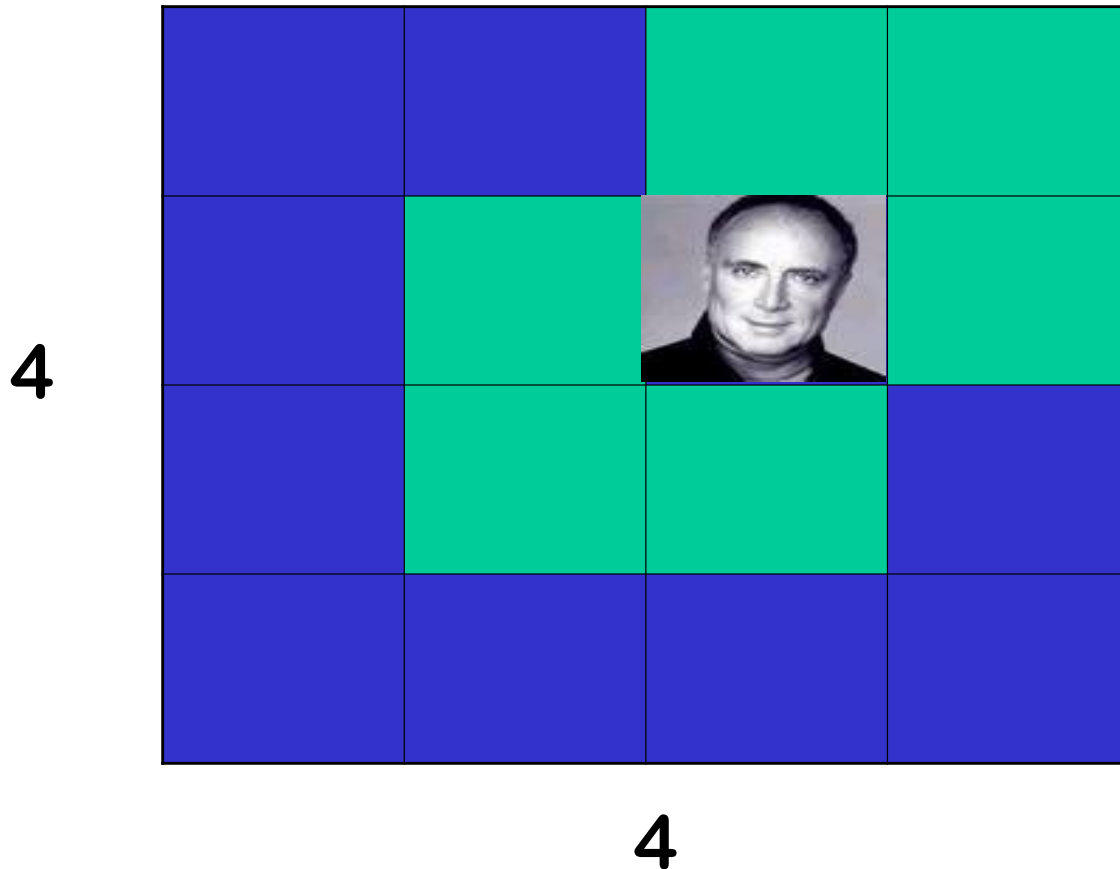
A tile in the middle is reserved to Efi Arazi statue.

Middle =
one of the
four
middle
squares.



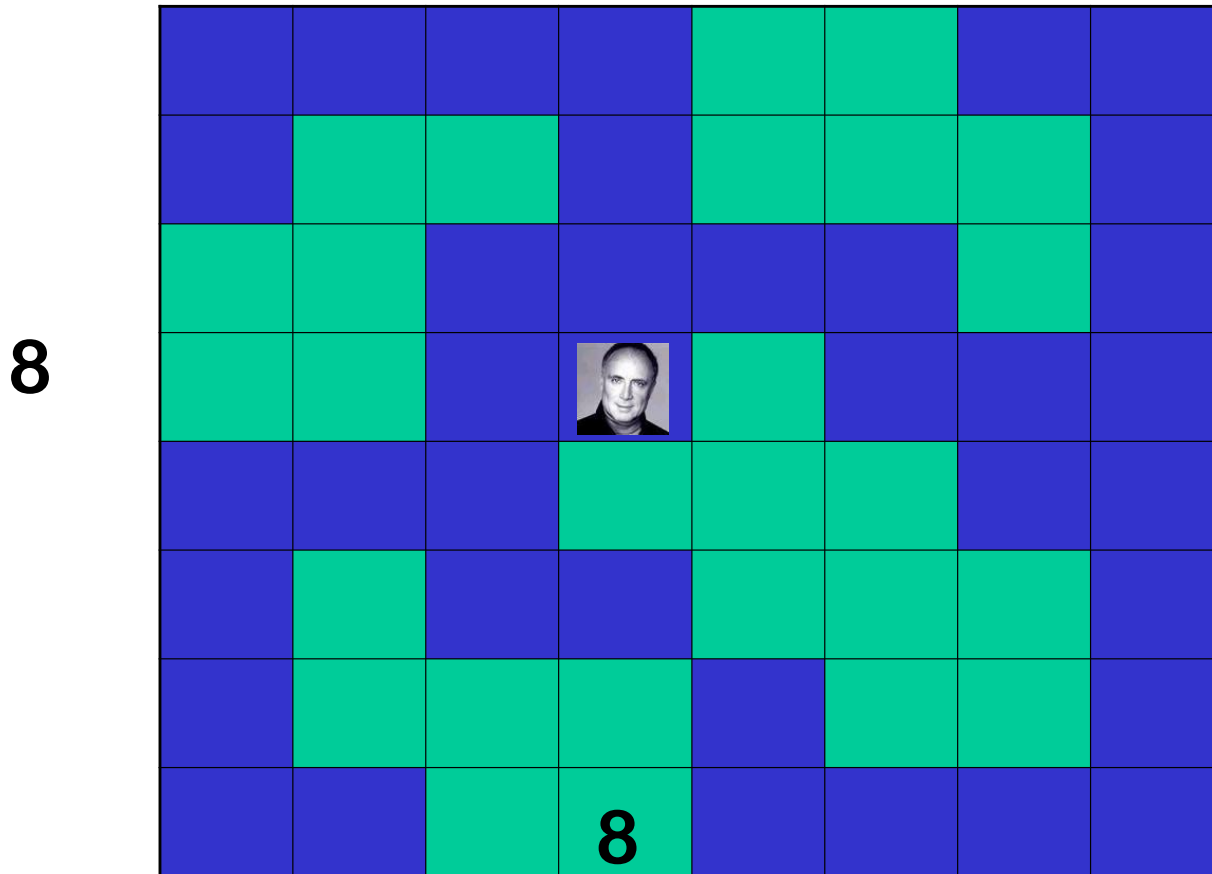
Tiling

For $n=2$, the 4×4 square can be tiled as follows:



Tiling

For $n=3$, the 8×8 square can be tiled as follows:



Tiling

Theorem: For all $n \in \mathbb{N}$ we can tile $2^n \times 2^n$ square so that both Efi and Architect are happy.

Proof: By induction on n . Let

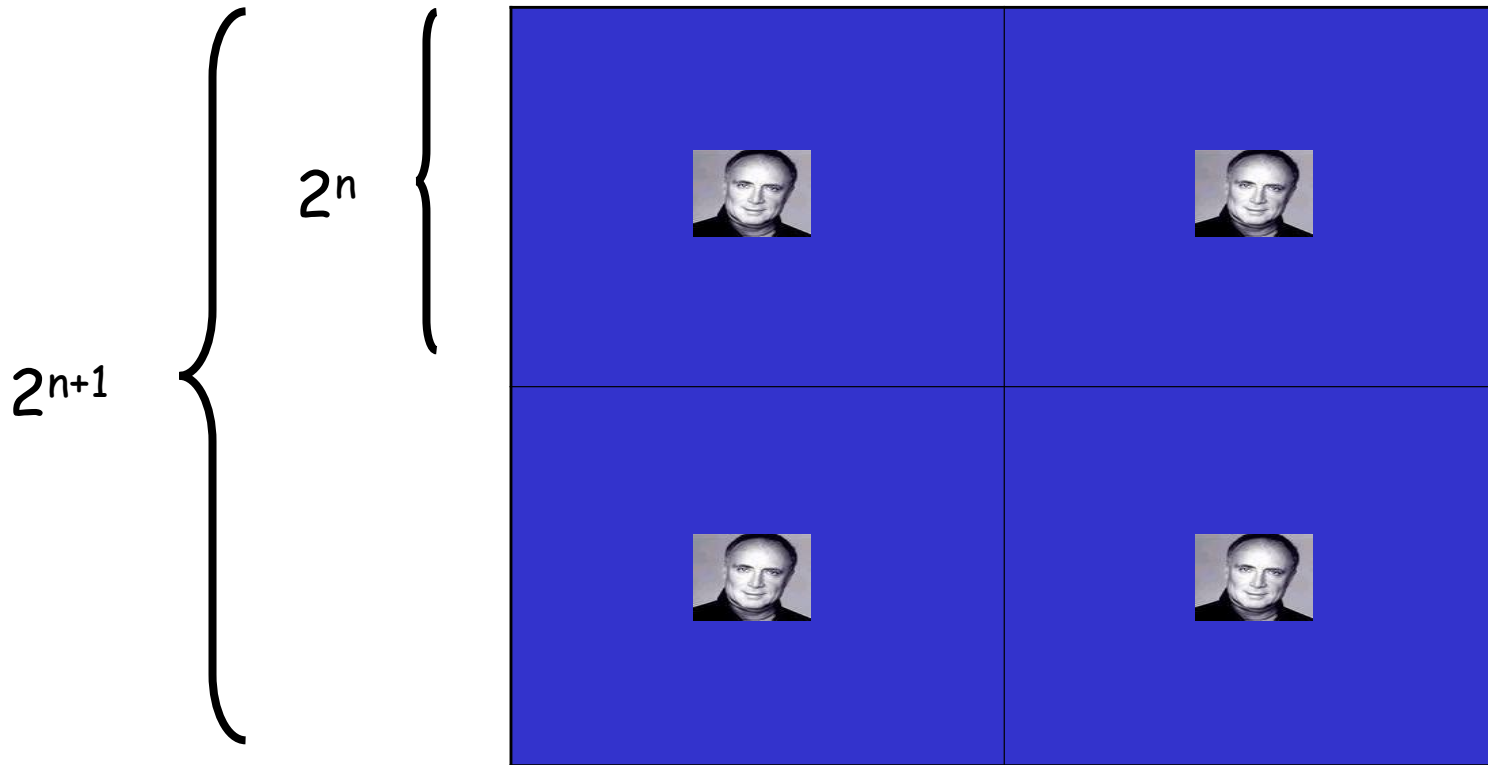
$P(n) :=$ [can tile $2^n \times 2^n$ square with Efi in middle]

Base case: True for $n = 0$ (no tiles are needed).



Tiling

Induction step: assume can tile $2^n \times 2^n$ square, prove that can tile $2^{n+1} \times 2^{n+1}$ square.



Now what??

Tiling

The idea: Use a **stronger** induction hypothesis.

1. Implies the original theorem.
2. Makes proving $P(n) \Rightarrow P(n+1)$ easier!

Proof (second attempt): By induction on n . Let

$P'(n) :=$ [can tile $2^n \times 2^n$ square with E_{fi} in any location]

Note: this implies

$P(n) :=$ [can tile $2^n \times 2^n$ square with E_{fi} in middle]

Tiling

$P'(n) :=$ [can tile $2^n \times 2^n$ square with Efi in any location]

Base case: Still true for $n = 0$ (no tiles are needed).

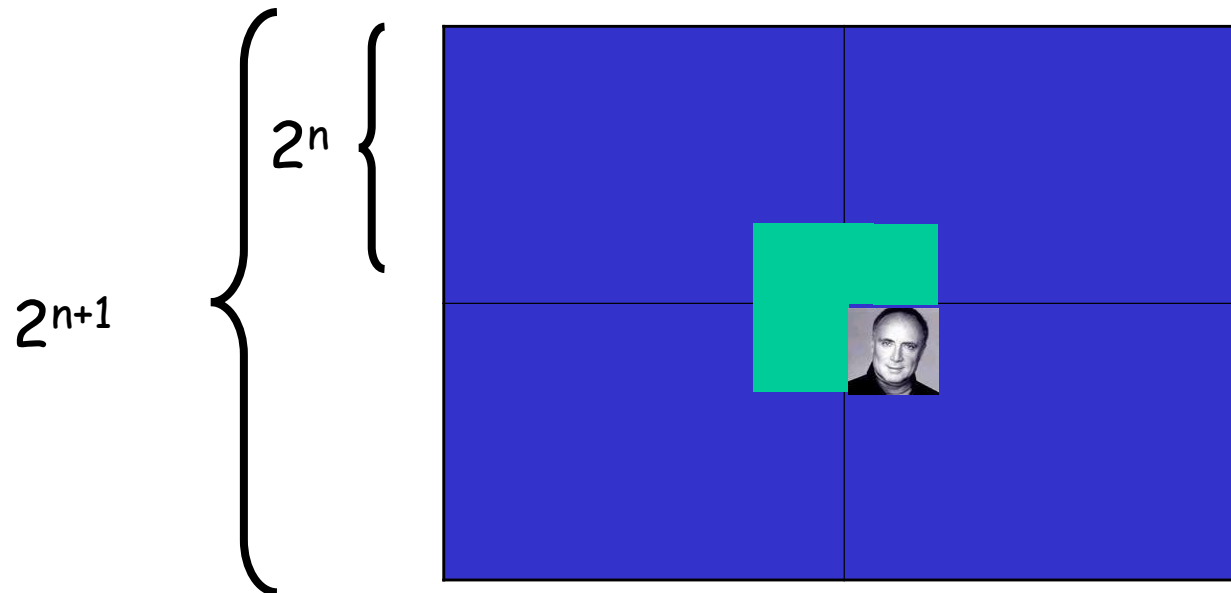


Induction step: Assume can tile $2^n \times 2^n$ square with Efi in any location.

Tiling

Given a $2^{n+1} \times 2^{n+1}$ square:

1. Ask the client to select Efi's location.
2. Locate the first tile in the middle, such that one block is missing from every quarter.
3. By the induction hypothesis the quarters can be legally tiled.



What are the Lessons?

Proof by induction can be constructive:

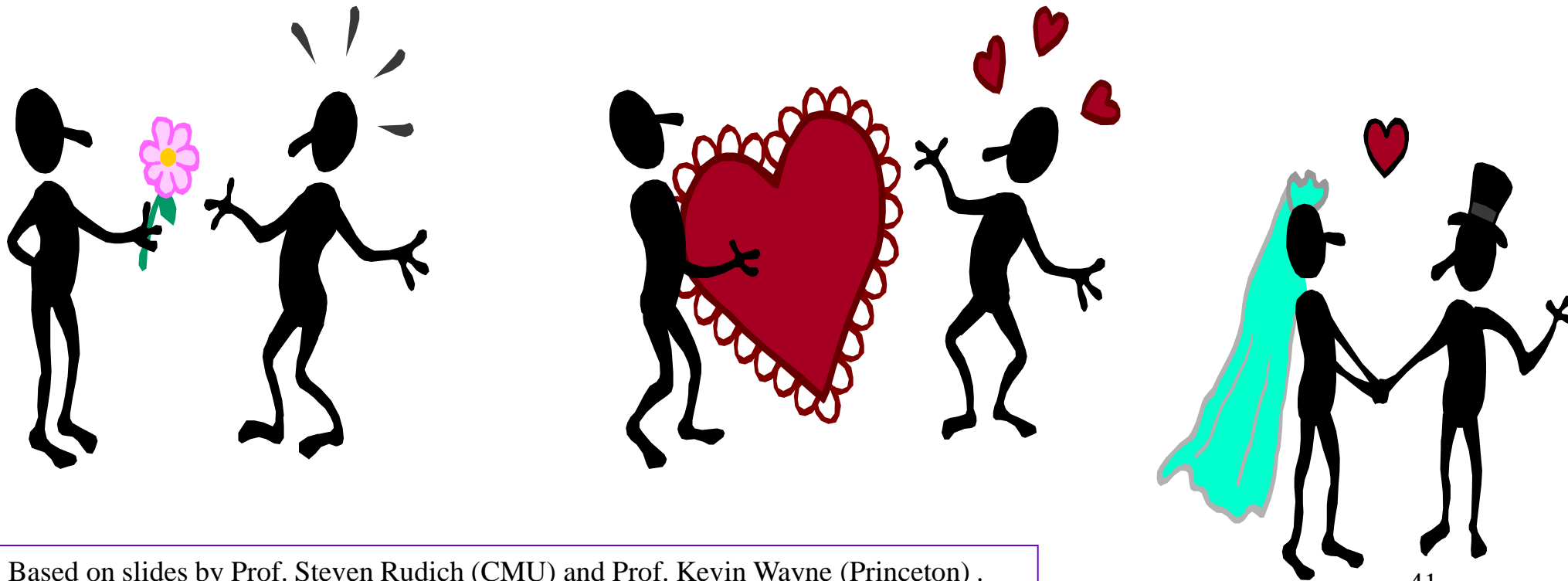
1. Sometimes yields an efficient **procedure/algorithm**.
2. Our proof implicitly defined a **recursive** procedure for tiling with Efi in the middle.

Choice of the induction hypothesis is crucial:

1. Assuming **stronger** hypothesis may make proof easier!
2. But need to ensure that $P(n) \Rightarrow P(n+1)$ is indeed true.

Great Theoretical Ideas In Computer Science

The Mathematics Of 1950's Dating: Who wins the battle of the sexes?



Based on slides by Prof. Steven Rudich (CMU) and Prof. Kevin Wayne (Princeton) .
Copyright © 2005 Pearson-Addison Wesley

3,2,5,1,4

Boys

1

1,2,5,3,4

2

4,3,2,1,5

3

1,3,4,2,5

4

1,2,4,5,3

5

Girls

3,2,5,1,4

1

5,2,1,4,3

2

4,3,5,1,2

3

1,2,3,4,5

4

2,3,4,1,5

5

Dating Scenario

- There are n boys and n girls
- Each girl has her own ranked preference list of all the boys
- Each boy has his own ranked preference list of all the girls
- The lists have no ties

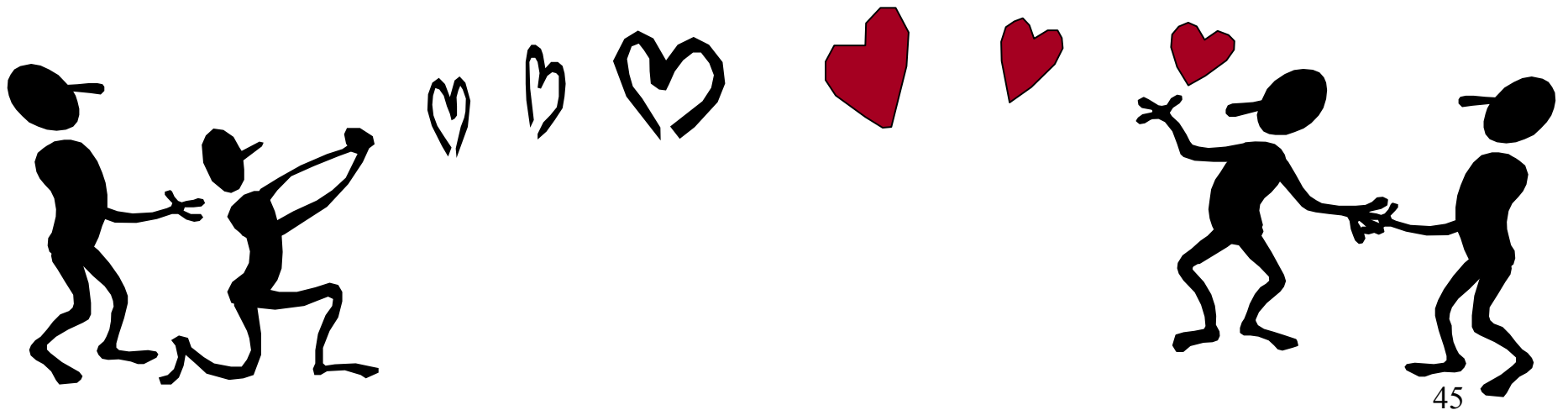
Question: How do we pair them off optimally?

What is considered a "good" pairing?

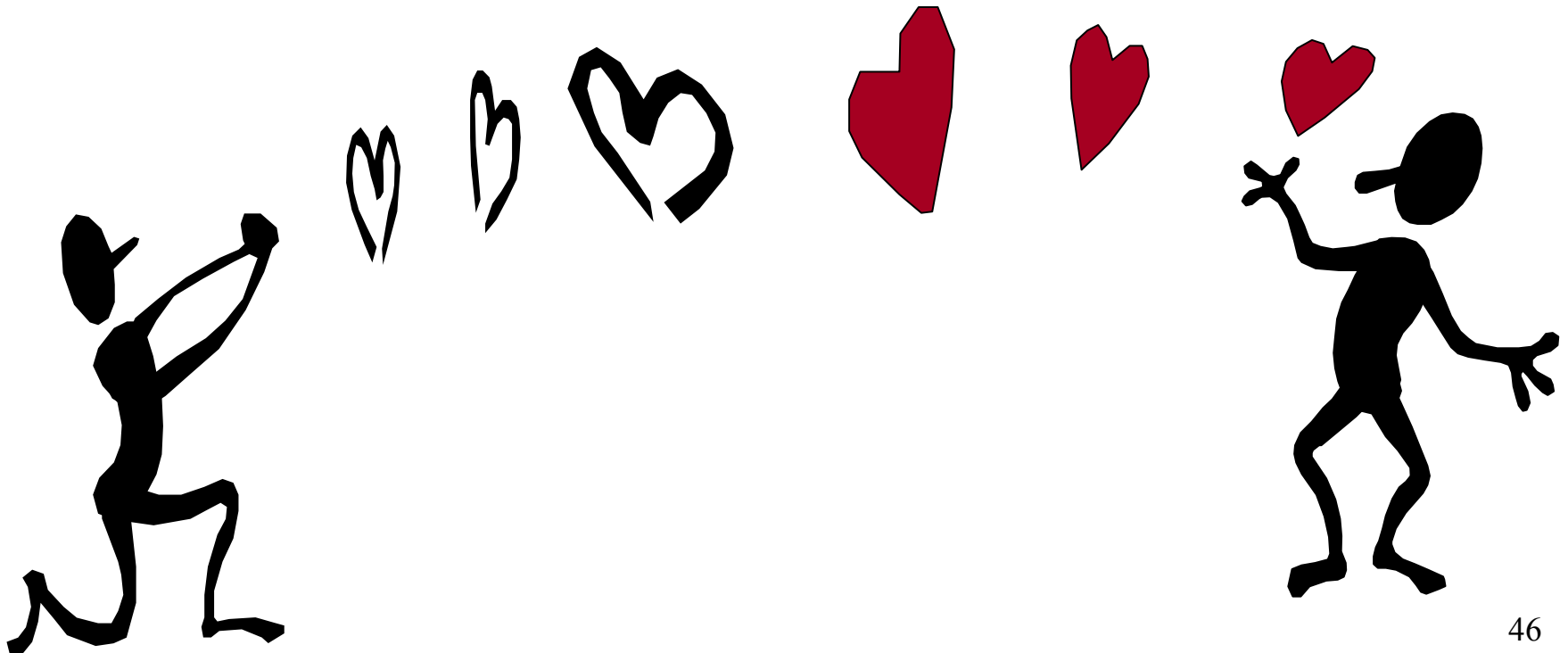
- Maximizing total satisfaction
 - What is the average rank of the partner in a person's ranking?
- Maximizing the minimum satisfaction
 - What is the rank of the partner in the most unsatisfied person's ranking?
- Minimizing the maximum difference in mate ranks
 - Everybody is more or less equally satisfied
- Maximizing the number of people who get their first choice

Rogue Couples

- Suppose we pair off all the boys and girls. Now suppose that some boy and some girl prefer each other to the people to whom they are paired. They will be called a rogue couple.



Why be with them when we can be
with each other?



Stable Pairings

- A pairing of boys and girls is called stable if it contains no rogue couples.

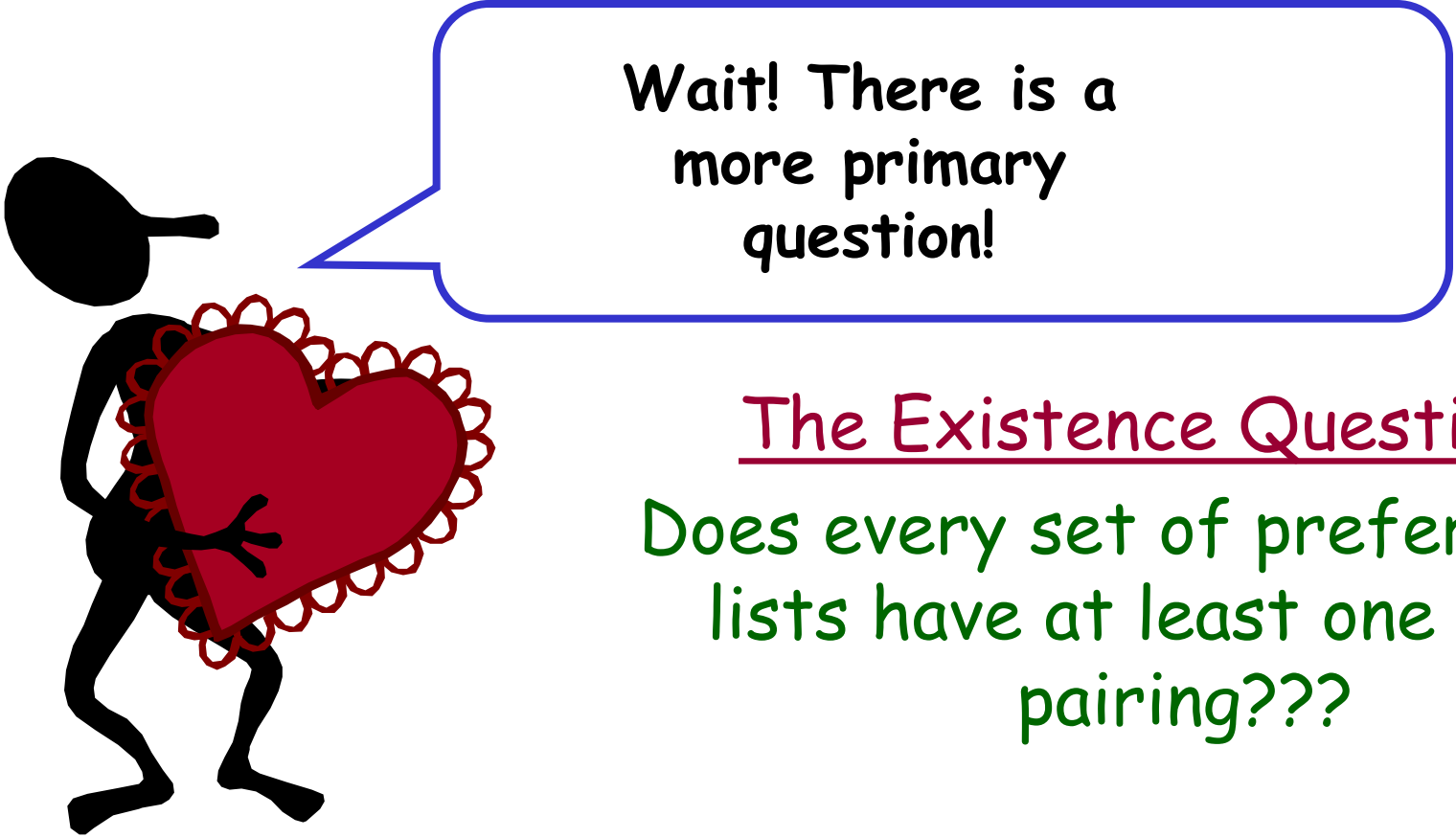
Stability is a Must.

- Any reasonable list of criteria for a good pairing must include **stability**. (A pairing is doomed if it contains a rogue couple.)

The study of stability will be the subject of the entire lecture.

- We will:
 - Analyze various mathematical properties of an algorithm that looks a lot like 1950's dating
 - Discover the **naked mathematical truth** about which sex has the romantic edge.
 - Learn how the world's largest, most successful dating service operates.

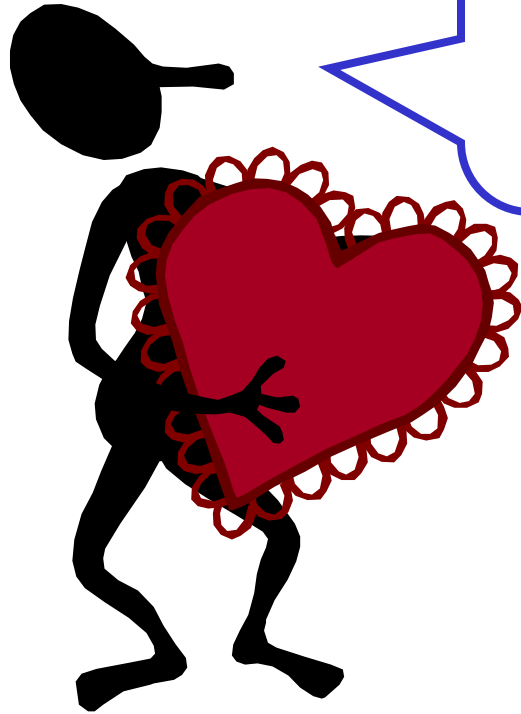
Given a set of preference lists, how do we find a stable pairing?



Wait! There is a more primary question!

The Existence Question:

Does every set of preferences lists have at least one stable pairing???



Can you argue that the couples will not continue breaking up and reforming forever?

An Instructive Variant: Roommate Problem

Stable roommate problem.

$2n$ people; each person ranks others from 1 to $2n-1$.

Assign roommate pairs so that no unstable pairs.

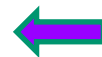
Observation: Stable matchings do not always exist for stable roommate problem.

	<i>1st</i>	<i>2nd</i>	<i>3rd</i>
<i>Adam</i>	B	C	D
<i>Bob</i>	C	A	D
<i>Chris</i>	A	B	D
<i>Dan</i>	A	B	C

A-B, C-D \Rightarrow B-C unstable

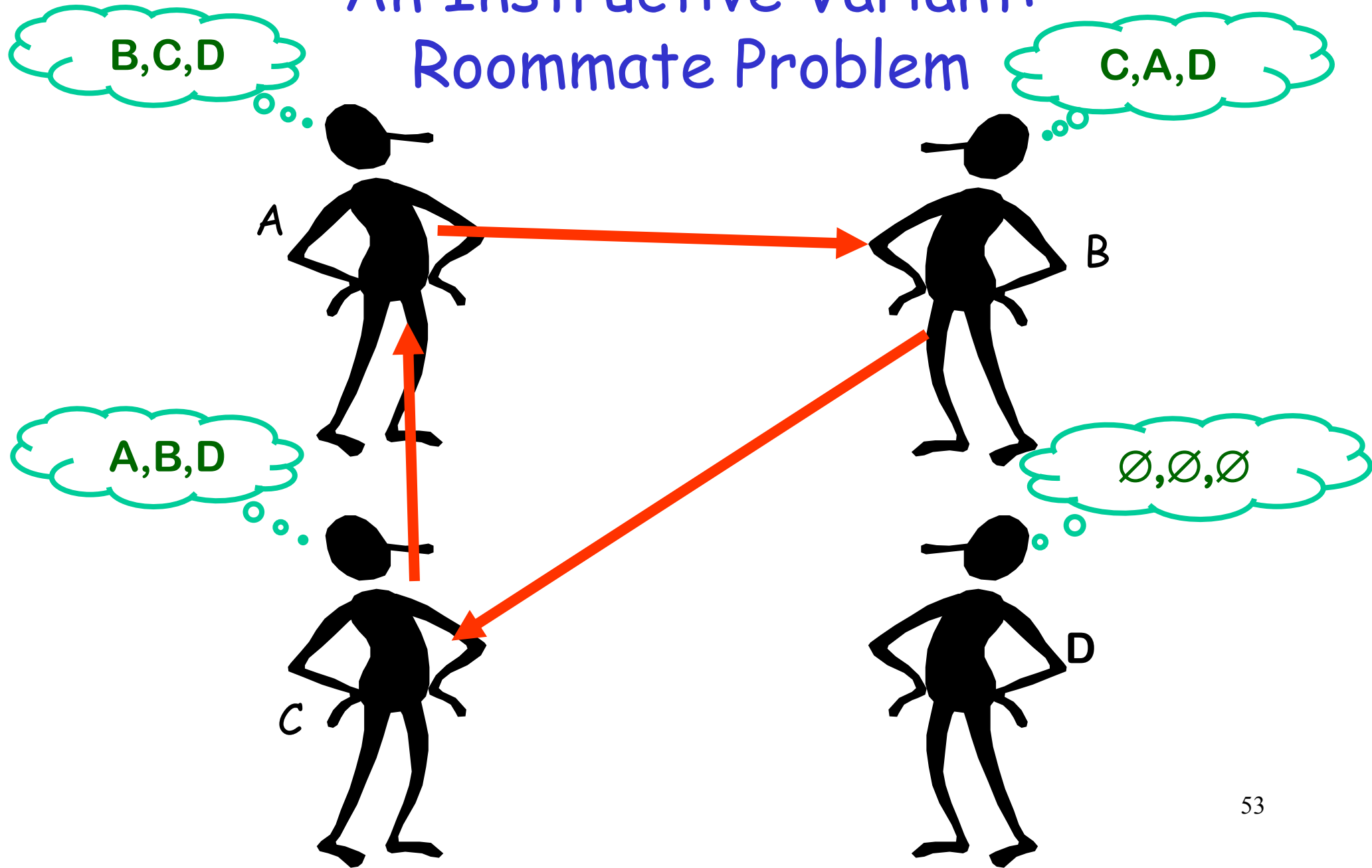
A-C, B-D \Rightarrow A-B unstable

A-D, B-C \Rightarrow A-C unstable



Can be any order

An Instructive Variant: Roommate Problem



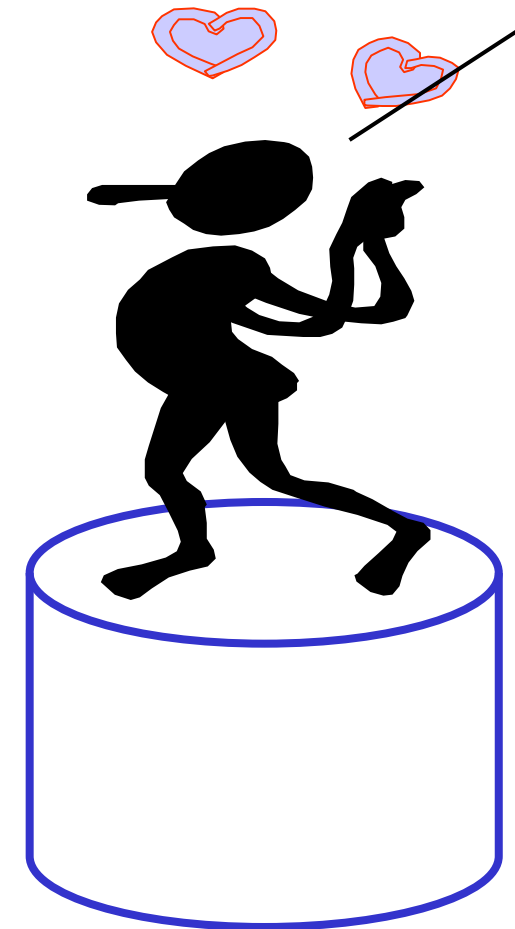
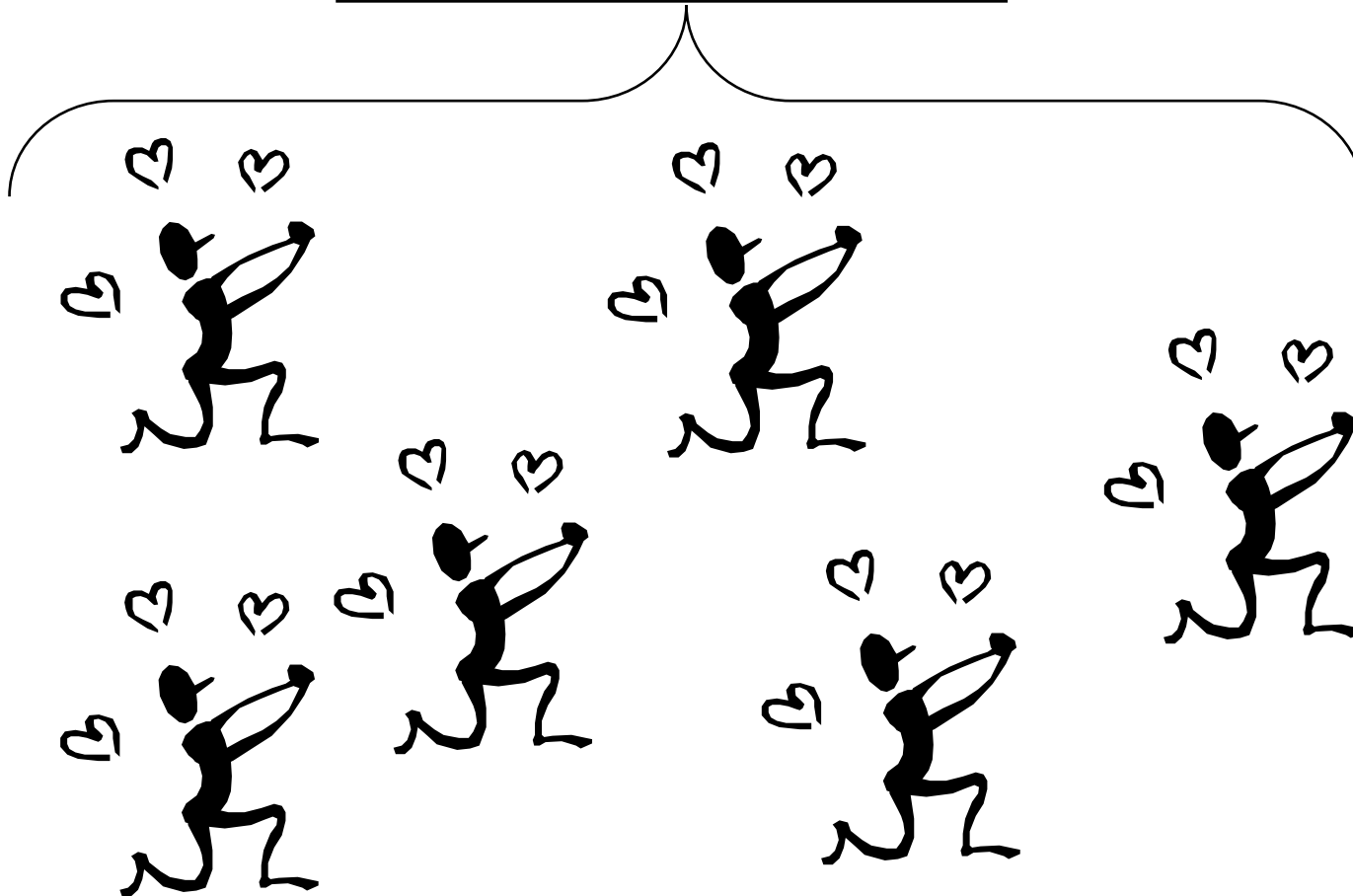
Insight

- Any proof that couples do not break up and reform forever must contain a step that fails in the case of the roommate problem.
- If you have a proof idea that works equally well in the marriage problem and the roommate problem, then your idea is not adequate to show the couples eventually stop.

The Traditional Marriage Algorithm

Female

Worshipping males



Traditional Marriage Algorithm

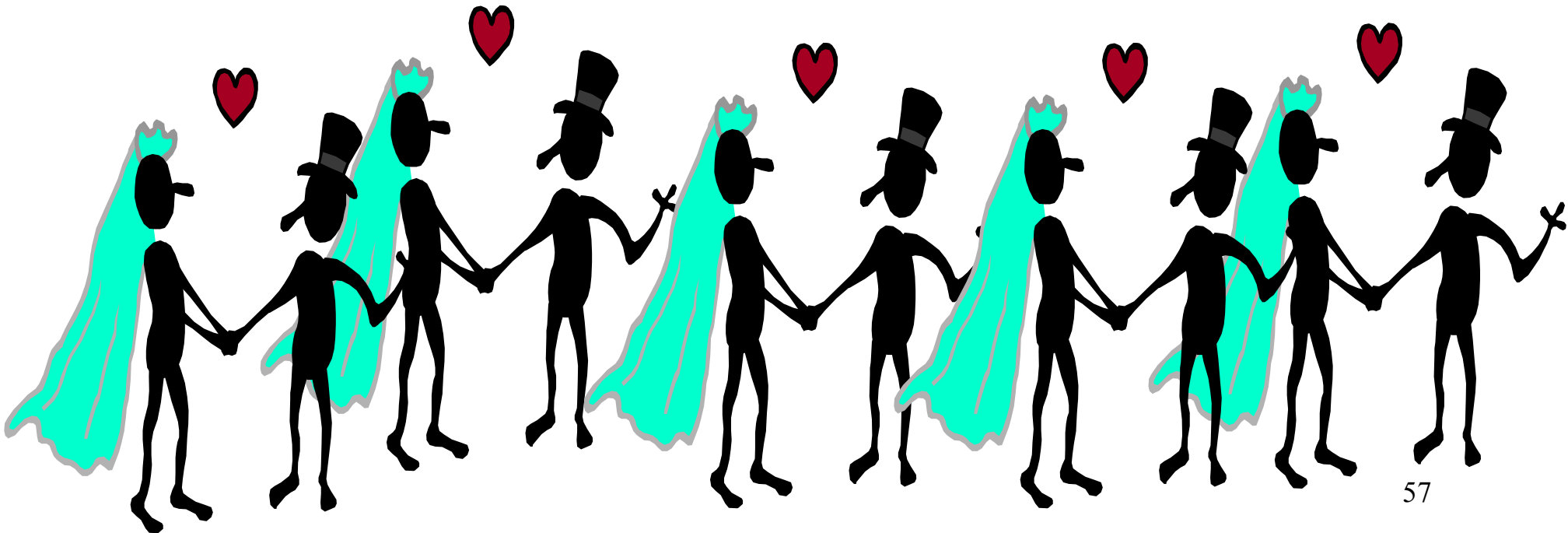
- Repeat:
 - Morning
 - Each girl stands on her balcony
 - Each boy proposes under the balcony of the best girl whom he has not yet crossed off
 - Afternoon (for those girls with at least one suitor)
 - To today's best suitor: **"Maybe, come back tomorrow"**
 - To any others: **"No, I will never marry you"**
 - Evening
 - Any rejected boy crosses the girl off his list.
- While some boy gets a **"No"** answer

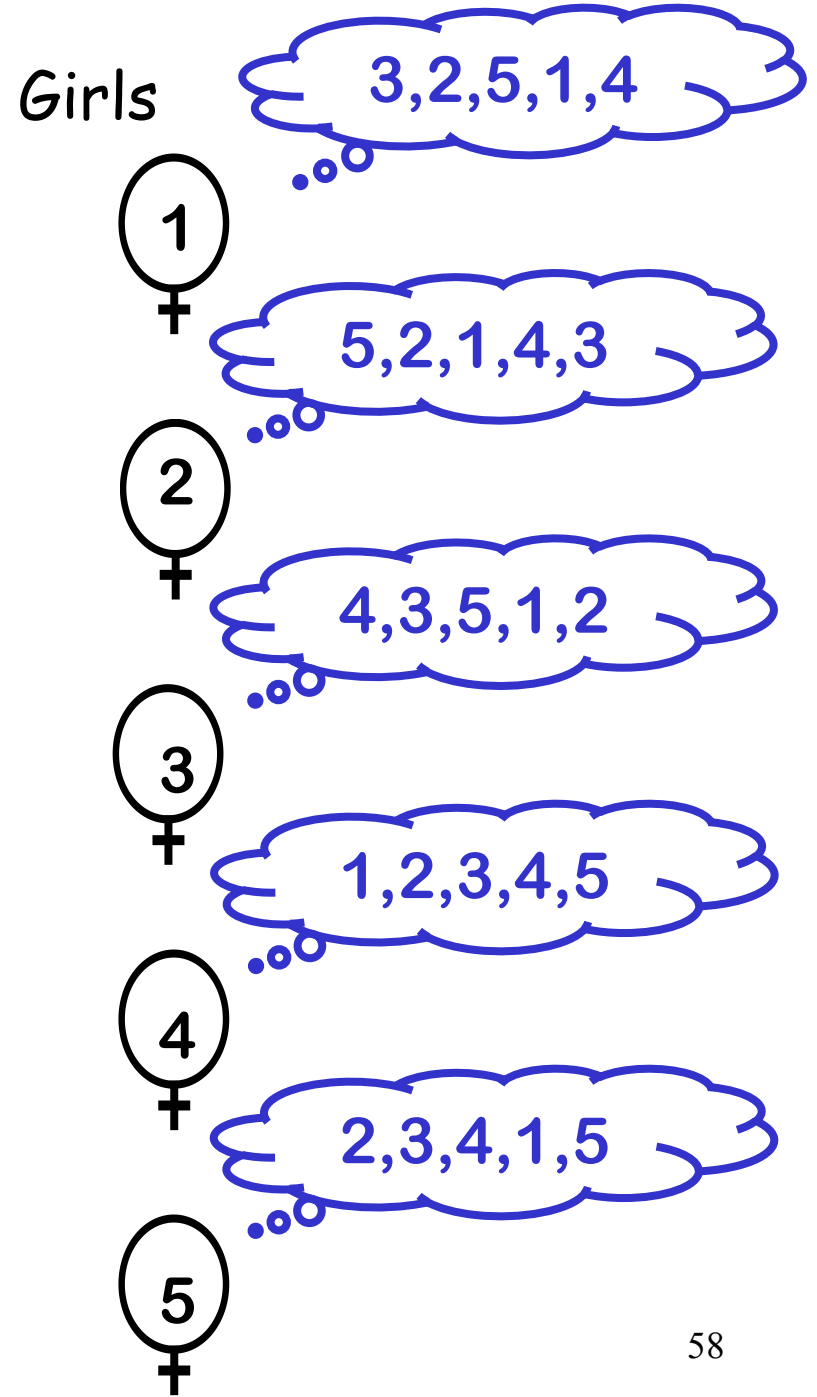
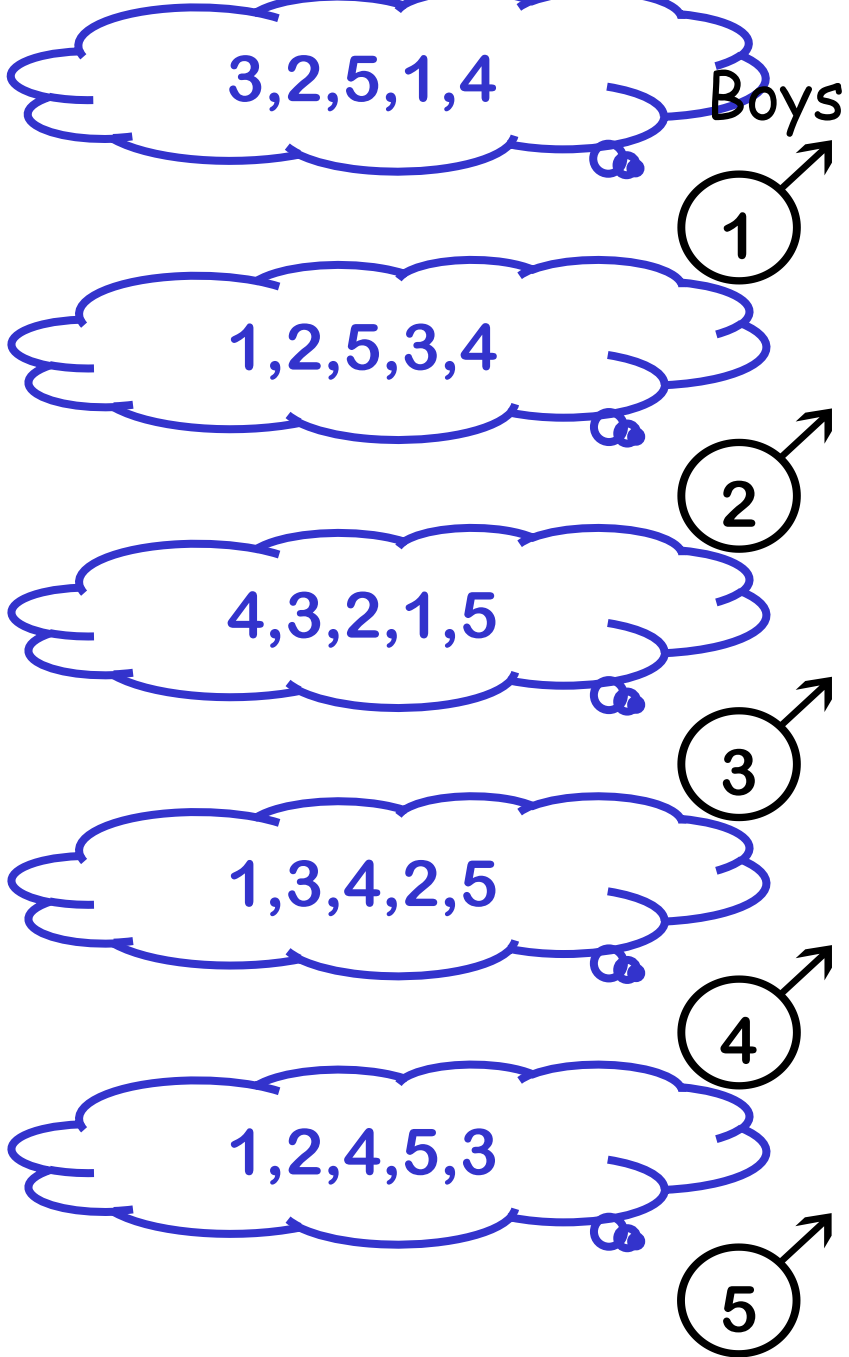
Each girl marries the boy to whom she just said "maybe"

Traditional Marriage Algorithm

Termination:

- When no boy gets a "No" (all were told "maybe"):
- Each girl marries the boy to whom she said "maybe".





Traditional Marriage Algorithm

- Example

girl	Day 1	Day 2
1	②,4,5	②
2		⑤
3	①	1,④
4	③	③
5		

	Boys	Girls
1	3,2,5,1,4	3,2,5,1,4
2	1,2,5,3,4	5,2,1,4,3
3	4,3,2,1,5	4,3,5,1,2
4	1,3,4,2,5	1,2,3,4,5
5	1,2,4,5,3	2,3,4,1,5

○ = come tomorrow

Traditional Marriage Algorithm

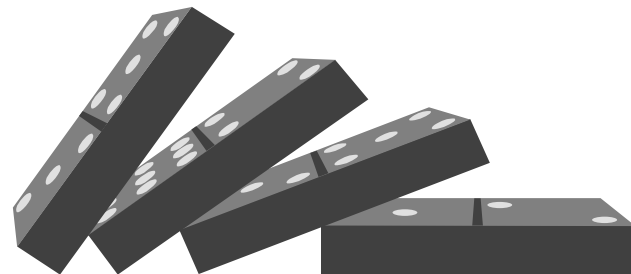
	Boys	Girls
1	3,2,5,1,4	3,2,5,1,4
2	1,2,5,3,4	5,2,1,4,3
3	4,3,2,1,5	4,3,5,1,2
4	1,3,4,2,5	1,2,3,4,5
5	1,2,4,5,3	2,3,4,1,5

girl	Day 1	Day 2	Day 3	Day 4
1	②,4,5	②	2	2
2		⑤	⑤, 1	5
3	①	1, ④	4	4
4	③	③	3	3
5				1

Trade-up lemma: In TMA, if on day i a girl says "maybe" to boy b , she is guaranteed to marry a husband that she likes at least as much as b .

- She would only let go of him in order to "maybe" someone better
- She would only let go of that guy for someone even better
- She would only let go of that guy for someone even better
- AND SO ON

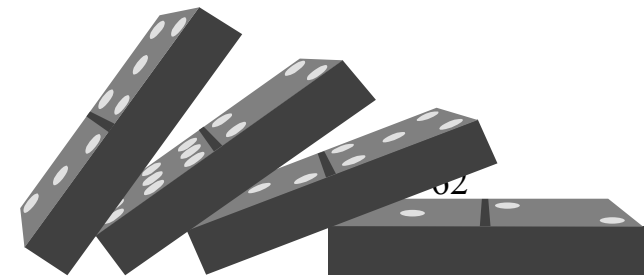
Informal Induction



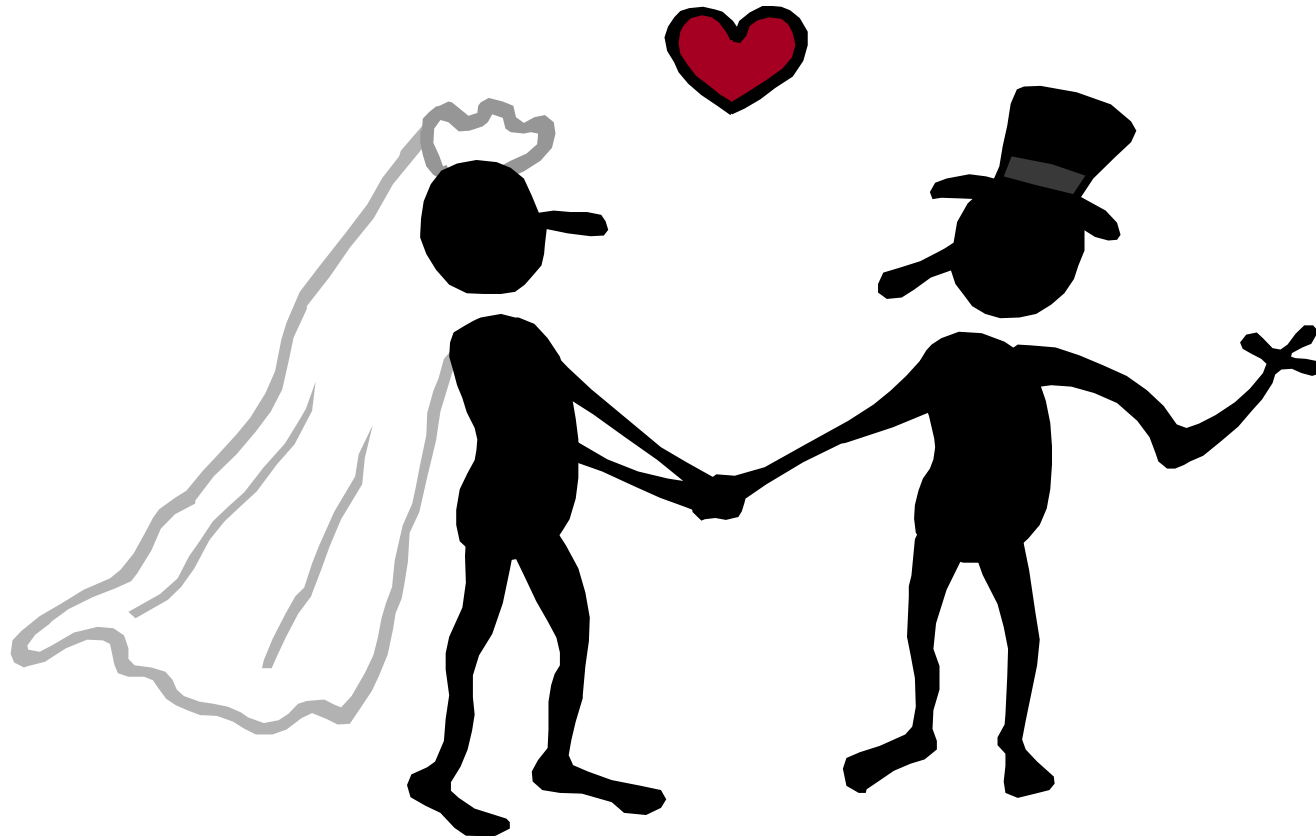
Trade-up Lemma: In TMA, if on day i a girl says "maybe" to boy b , she is guaranteed to marry a husband that she likes at least as much as b .

- (*) For all $k \geq 0$, on day $i+k$ the girl will say "maybe" to a boy she likes as much as b .
- Base: $k=0$ (true by assumption)
- Assume (*) is true for $k-1$. Thus she has a boy as good as b on day $i+k-1$. The next day she will either keep him or reject him for some better. Thus (*) is true for k .

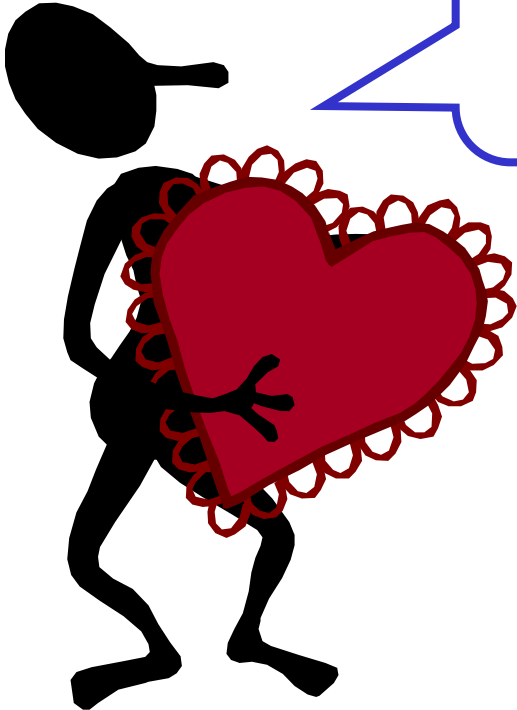
Formal Induction



Corollary: Each girl will marry her absolute favorite of the boys who visit her during the TMA.



Does the Traditional Marriage Algorithm always produce a stable pairing?



Wait! There is a more primary question!

Does TMA always terminate?

Does TMA always terminate?

- It might encounter a situation where the algorithm does not specify what to do next.
- It might keep on going for an infinite number of days.

Lemma: Everyone will be matched.

- We show that no boy can be rejected by all the girls. This would imply that there are n couples.
- Suppose by contradiction that Bob is rejected by all the girls.
- Then some woman, say Amy, is not matched when Bob marked out the last girl from his list.
 - By the trade-up lemma, Amy was never proposed to.
 - But Bob proposes to everyone, since he ends up unmatched. In particular to Amy

Contradiction

Theorem: The TMA always terminates
in at most n^2 days

- Consider the "master list" containing all the boy's preference lists of girls. There are n boys, and each list has n girls on it, so there are a total of $n \times n = n^2$ girls' names in the master list.
- Each day that at least one boy gets a "No", at least one girl gets crossed off the master list.
- Therefore, the number of days is bounded by the original size of the master list.

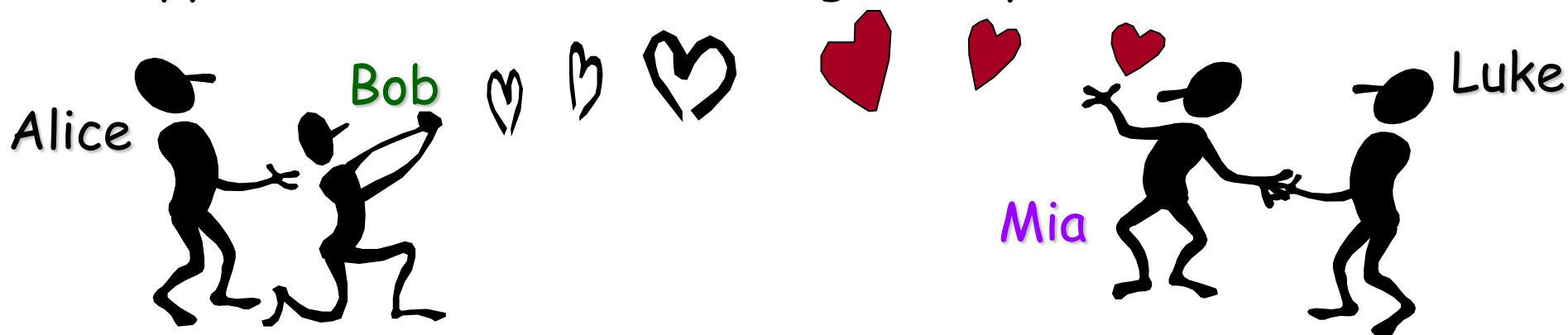
Great! We know that TMA will terminate and produce a pairing.

But is it stable?

Theorem: The pairing produced by TMA is stable.

- Proof by contradiction:

Suppose **Bob** and **Mia** are a rogue couple.

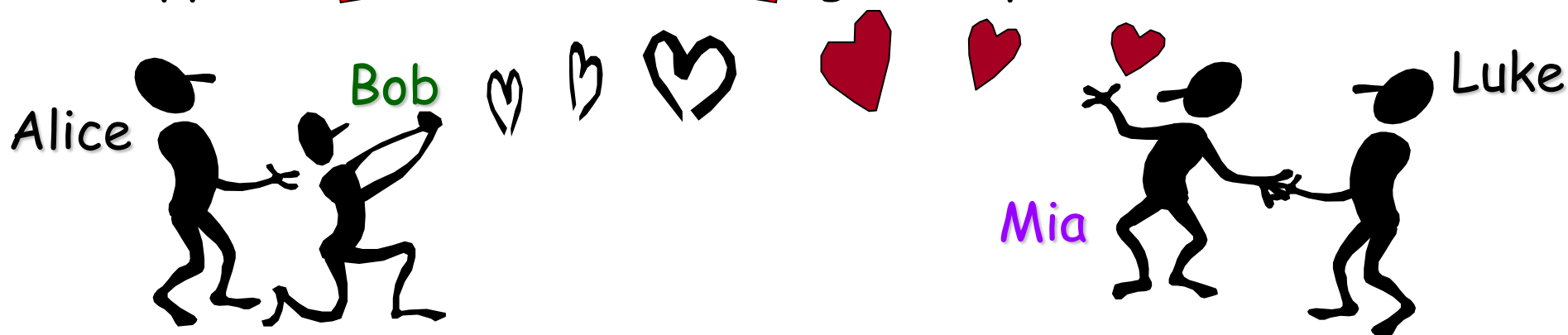


- This means **Bob** likes **Mia** more than his wife, Alice.
- Thus, **Bob** proposed to **Mia** before he proposed to Alice.
- **Mia** must have rejected **Bob** for someone she preferred.
- By the Tradeoff lemma, she must like her husband Luke more than **Bob**.

Contradiction!

Theorem: The pairing produced by TMA is stable.

- Proof by ~~contradiction~~:
Suppose ~~Bob and Mia~~ are a rogue couple.



- **Suppose** Bob likes Mia more than his wife, Alice.
- Thus, Bob proposed to Mia before he proposed to Alice.
- Mia must have rejected Bob for someone she preferred.
- By the Tradeoff lemma, she must like her husband Luke more than Bob.
- So no boy participates in a rogue couple.

Opinion Poll



Understanding the Solution

- **Question:** For a given problem instance, there may be several stable matchings. Do all executions of the algorithm yield the same stable matching? If so, which one?
- An instance with two stable matchings:
 - A-X, B-Y, C-Z.
 - A-Y, B-X, C-Z.

	1 st	2 nd	3 rd
Xavier	A	B	C
Yancey	B	A	C
Zeus	A	B	C

	1 st	2 nd	3 rd
Amy	Y	X	Z
Bertha	X	Y	Z
Clare	X	Y	Z

Forget TMA for a moment

- How should we define what we mean when we say "the optimal girl for Zeus"?

Flawed Attempt:
"The girl at the top of Zeus's list"

The Optimal Girl

- A boy's **optimal girl** is the highest ranked girl G for whom there is some stable pairing in which the boy marries G .
- She is the best girl he can conceivably get in a stable world. Presumably, she might be better than the girl he gets in the stable pairing output by TMA.

The Pessimal Girl

- A boy's **pessimal girl** is the lowest ranked girl G for whom there is some stable pairing in which the boy marries G .
- She is the worst girl he can conceivably get in a stable world.

Dating Heaven and Hell

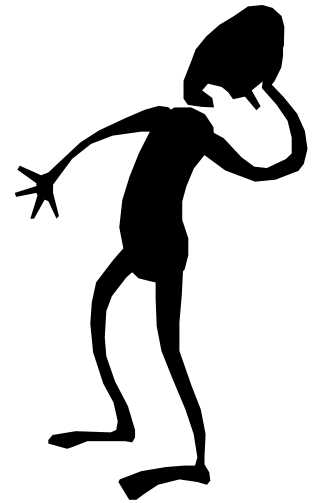
- A pairing is **male-optimal** if **every** boy gets his **optimal** girl. This is the best of all possible stable worlds for all the boys simultaneously.
- A pairing is **male-pessimal** if **every** boy gets his **pessimal** girl. This is the worst of all possible stable worlds for all the boys simultaneously.

Dating Heaven and Hell

- A pairing is **female-optimal** if **every** girl gets her **optimal** mate. This is the best of all possible stable worlds for every girl simultaneously.
- A pairing is **female-pessimal** if **every** girl gets her **pessimal** mate. This is the worst of all possible stable worlds for every girl simultaneously.

The Naked Mathematical Truth!

- The Traditional Marriage Algorithm always produces a **male-optimal**, **female-pessimal** pairing.



Theorem: TMA produces a male-optimal pairing

- Suppose not: i.e. that some boy gets rejected by his optimal girl during TMA.
- In particular, let's say **Bob** is the **first boy** to be rejected by his optimal girl **Mia**: Let's say she said "maybe" to **Luke**, whom she prefers.
- Since **Bob** was the only boy to be rejected by his optimal girl so far, **Luke** must like **Mia** at least as much as his optimal girl.

The lists:

Mia: Luke Bob.....

Luke: Mia... Optimal girl....

Bob: Mia (optimal girl)...

- We show that any pairing S in which Bob marries Mia cannot be stable (for a contradiction).
- Suppose S is stable:
 - Luke likes Mia more than his wife in S (At least as much as his optimal possible girl)
 - Mia likes Luke more than her husband Bob in S



Theorem: The TMA pairing, T , is female-pessimal.

- Suppose there is a stable pairing S where some girl $Alice$ does worse than in T .
- Let $Luke$ be her mate in T .
 - $Alice$ likes $Luke$ better than her mate in S .
 - T is male-optimal, so $Luke$ likes $Alice$ better than his mate in S .
 - Therefore, S is not stable.

A contradiction

Efficient Implementation

- We describe an $O(n^2)$ -time implementation.
- Men and women are denoted $1, \dots, n$ and $1', \dots, n'$ respectively.
- **Engagements:**
 - Maintain a list of rejected men.
 - Maintain two arrays `wife[m]`, and `husband[w]`.
 - set entry to 0 if unmatched
 - if w just said "maybe" to m then `wife[m]=w` and `husband[w]=m`
- **Men proposing:**
 - For each man, maintain a list of women, ordered by preference.
 - Maintain an array `count[m]` that counts the number of (different) proposals made by man m .

Efficient Implementation

- Women rejecting/accepting.
 - Does woman w prefer man m to man m' ?
 - For each woman, create **inverse** of preference list of men.
 - Constant time access for each query after $O(n)$ preprocessing.

Amy	1 st	2 nd	3 rd	4 th	5 th	6 th	7 th	8 th
prefBoy	8	3	7	1	4	5	6	2

Amy	1	2	3	4	5	6	7	8
inverse	4 th	8 th	2 nd	5 th	6 th	7 th	3 rd	1 st

Amy prefers man 3 to 6
since $\text{inverse}[3] < \text{inverse}[6]$
2 7

```
for i = 1 to n
    inverse[prefBoy[i]] = i
```

Efficient Implementation

- Initially, all men are rejected.
- Iteration: Each rejected man m proposes to `count[m]` (and `count[m]` increases)
- The women answer (each in $O(1)$)
- Arrays `husband/wife` updated. The queue is updated. $O(1)$ for each proposal made.
- Time complexity: Init: $O(n^2)$ (create `inverse[]`)
- n women, each woman is proposed to at most n times.
- n men, each man proposes at most n times
- Total of at most $O(n^2)$ proposals, each proposal in $O(1)$.

Game Theory @ Stable Pairing Problem

- **Question:** Can there be an incentive to misrepresent your preference profile?
 - Assume you know TMA algorithm will be used.
 - Assume that you know the preference profiles of all other participants.
 - **Answer:** No, for any man.
Yes, for some women.
- No mechanism can guarantee a stable matching and be cheatproof.

Game Theory @ Stable Pairing Problem

Example: Amy Lies and improves her match.

	1 st	2 nd	3 rd
Xavier	A	B	C
Yancey	B	A	C
Zeus	A	B	C

Men's Preference List

	1 st	2 nd	3 rd
Amy	Y	X	Z
Bertha	X	Y	Z
Clare	X	Y	Z

Women's True Preference Profile

	1 st	2 nd	3 rd
Amy	Y	Z	X
Bertha	X	Y	Z
Clare	X	Y	Z

Amy Lies

References

- D. Gale and L. S. Shapley, *College admissions and the stability of marriage*, American Mathematical Monthly 69 (1962), 9-15
- Dan Gusfield and Robert W. Irving, *The Stable Marriage Problem: Structures and Algorithms*, MIT Press, 1989

Application: Matching Residents to Hospitals

- Men \approx hospitals, Women \approx medical school residents.
 - Each medical school graduate submits a ranked list of hospitals where he/she wants to do a residency.
 - Each hospital submits a ranked list of newly minted doctors.
- Variant 1. Some participants declare others as unacceptable (resident A unwilling to work in Cleveland).
- Variant 2. Unequal number of men and women.
- Variant 3. Limited polygamy (hospital X wants to hire 3 residents).

Extensions: Matching Residents to Hospitals

Definition: Matching S is **unstable** if there is a hospital h and resident r such that:

- h and r are acceptable to each other; and
- either r is unmatched, or r prefers h to her assigned hospital; and
- either h does not have all its places filled, or h prefers r to at least one of its assigned residents.

How can the TMA algorithm be extended?

History

- 1900
 - Idea of hospitals having residents (“interns”)
- Over the next few decades
 - Intense competition among hospitals for an inadequate supply of residents
 - Each hospital makes offers independently
 - Process degenerates into a race. Hospitals steadily advancing date at which they finalize binding contracts
- 1944 Absurd Situation. Appointments being made 2 years ahead of time!
 - All parties were unhappy
 - Medical schools stop releasing any information about students before some reasonable date
 - **Offers were made at a more reasonable date, but new problems developed**

History

- 1945-1949
 - Hospitals started putting time limits on offers
 - Time limit gets down to 12 hours
 - Lots of unhappy people
 - Many instabilities resulting from lack of cooperation
- 1950 Centralized System
 - Each hospital ranks residents
 - Each resident ranks hospitals
 - National Resident Matching Program produces a pairing.
 - Sometimes unstable.
- By 1952 the algorithm was the TMA (hospital-optimal) and therefore stable.