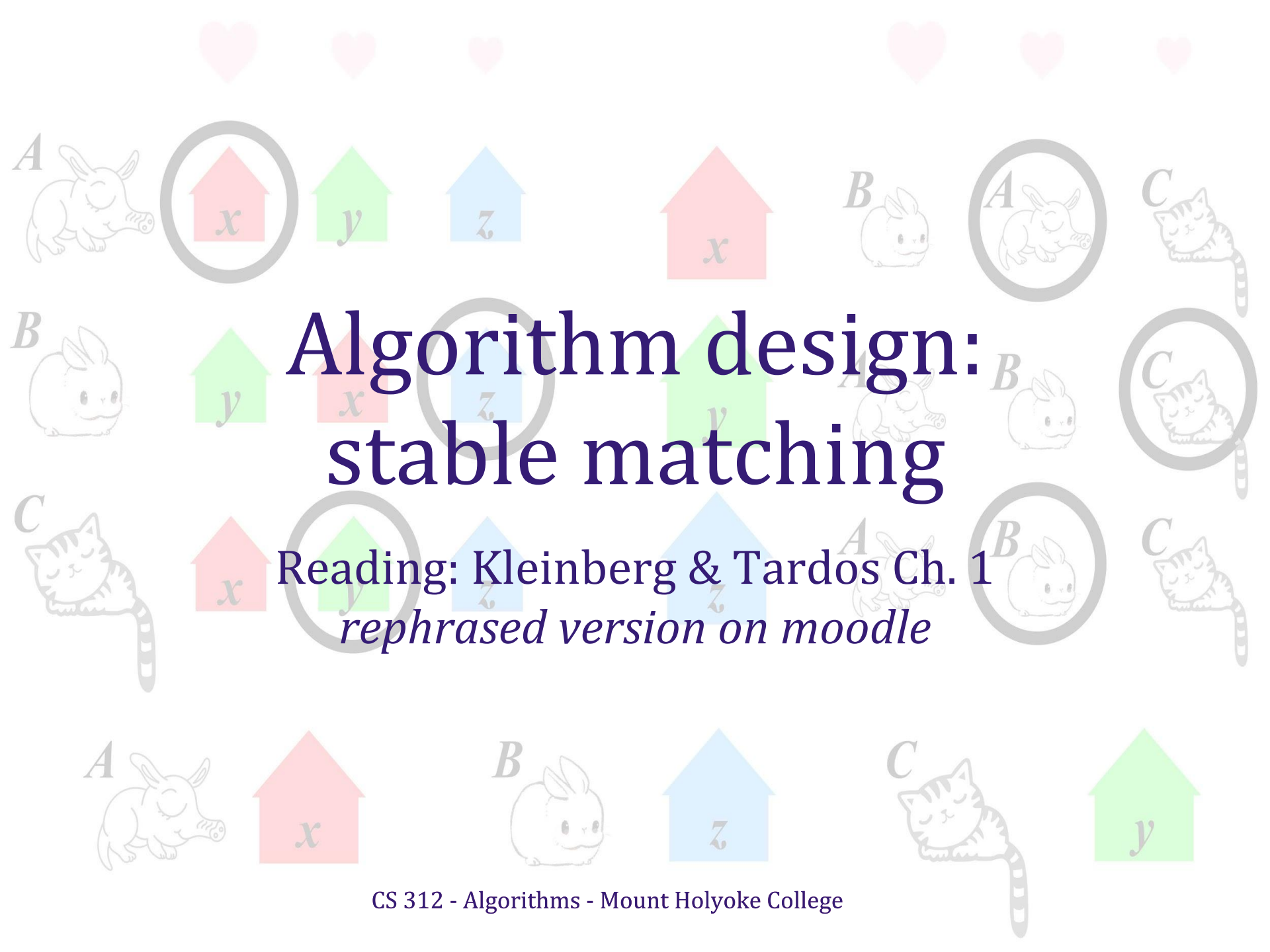


Algorithm design: stable matching

Reading: Kleinberg & Tardos Ch. 1

rephrased version on moodle

Algorithm design process

Algorithm:
computational procedure correctly solving a problem

1. pose the problem rigorously
   (formulate a *mathematically* clean definition)
2. propose a solution as a procedure
   (description can be *implemented* within specified computational model, e.g., a random-access machine)
3. prove that the procedure correctly solves the problem
4. analyze the procedure's running **time**

Algorithm design process

0. work examples to gain familiarity/intuition

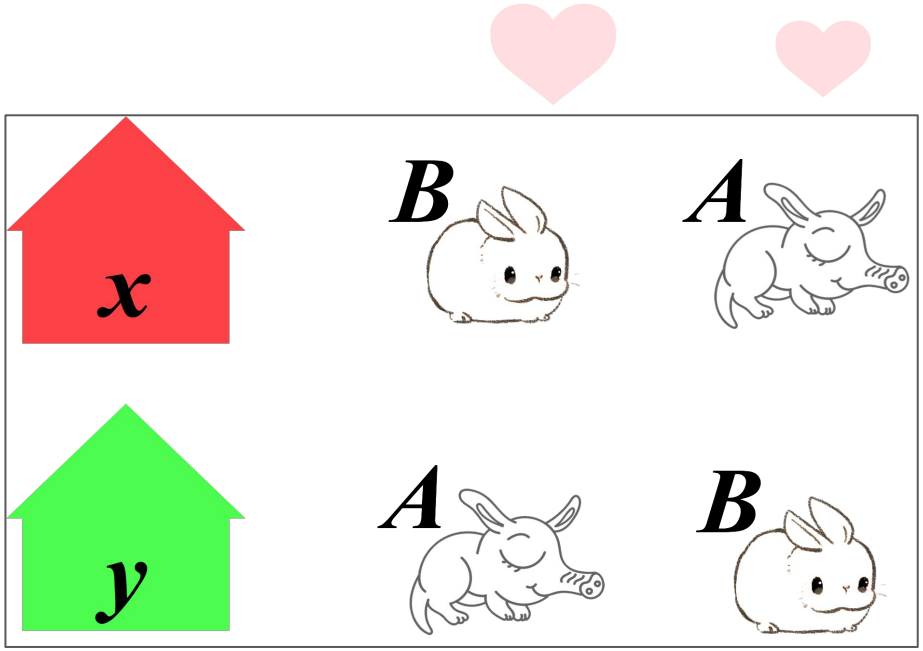
Algorithm:

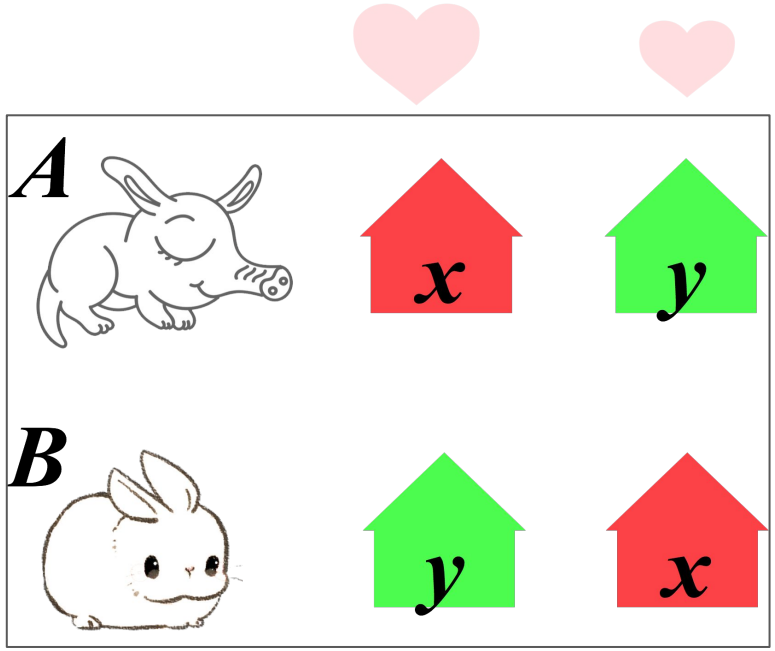
computational procedure correctly solving a problem

1. pose the problem rigorously
   (formulate a *mathematically* clean definition)
2. propose a solution as a procedure
   (description can be *implemented* within specified computational model, e.g., a random-access machine)
3. prove that the procedure correctly solves the problem
4. analyze the procedure's running **time**

Problem setup: pet adoption

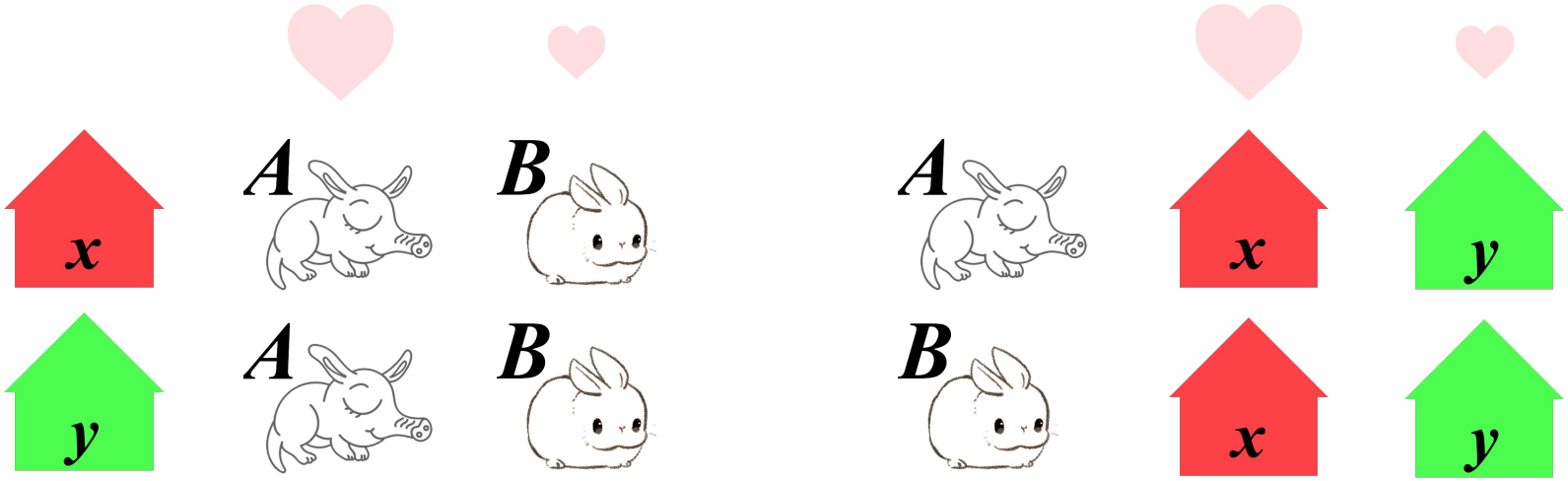
A shelter is looking to have its animals adopted into loving families. How do you decide which animals should become pets to which families?



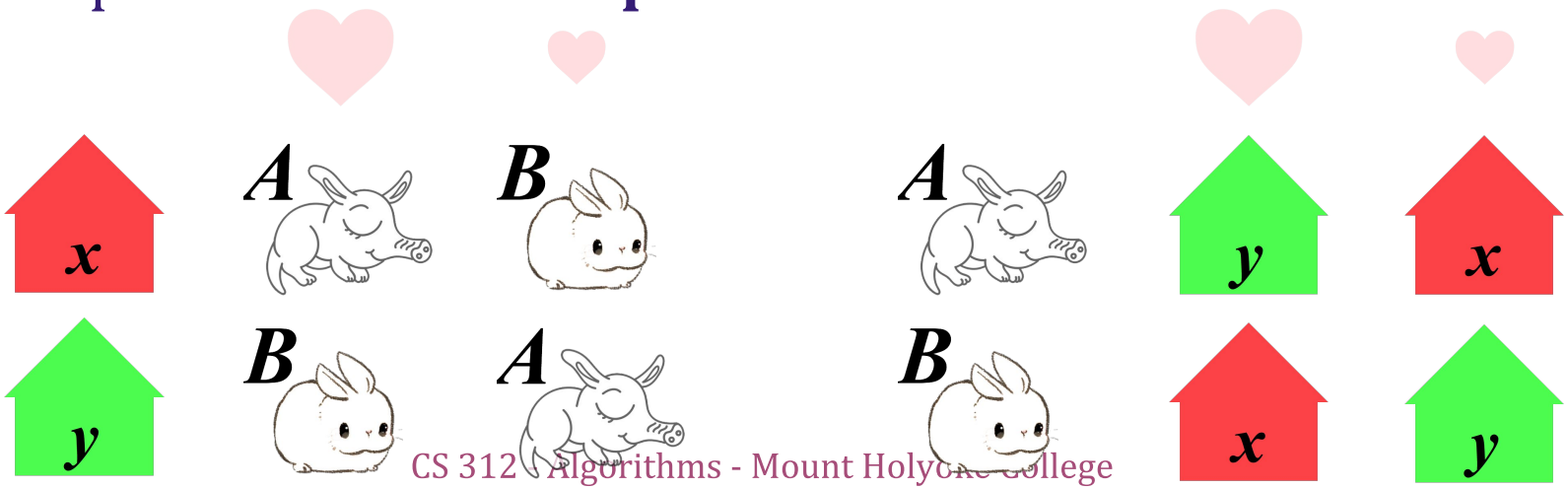


What adoptions would you suggest?

Example 1. **Universal prefs**



Example 2. **Inconsistent prefs**



Algorithm design process

Algorithm:
computational procedure correctly solving a problem

1. pose the problem rigorously
   (formulate a *mathematically* clean definition)
2. propose a solution as a procedure
   (description can be *implemented* within specified computational model, e.g., a random-access machine)
3. prove that the procedure correctly solves the problem
4. analyze the procedure's running **time**

Algorithm design process: step 1

Algorithm:
computational procedure correctly solving a problem

1. pose the problem rigorously
   (formulate a *mathematically* clean definition)

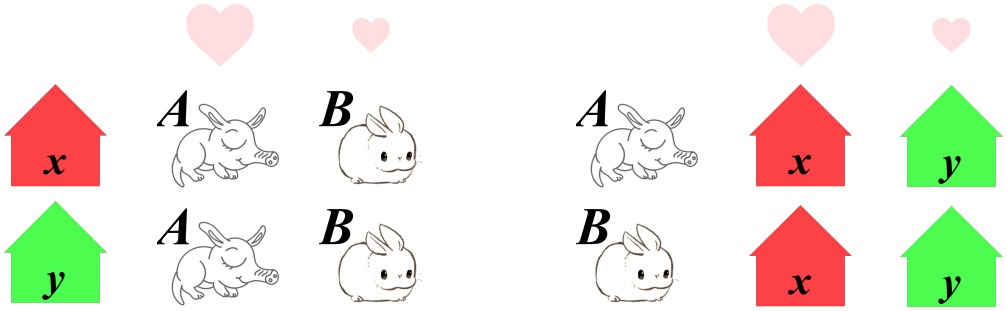
Formulating the problem of pet adoption

Given: *n* families looking to adopt *n* animals

- Each family ranks all animals
- Each animal ranks all families
- For simplicity, **assume each family can only adopt one animal**

Can we match animals to families s.t. everyone is **"happy"**?

- Not necessarily -- what if the aardvark were everyone's top choice?



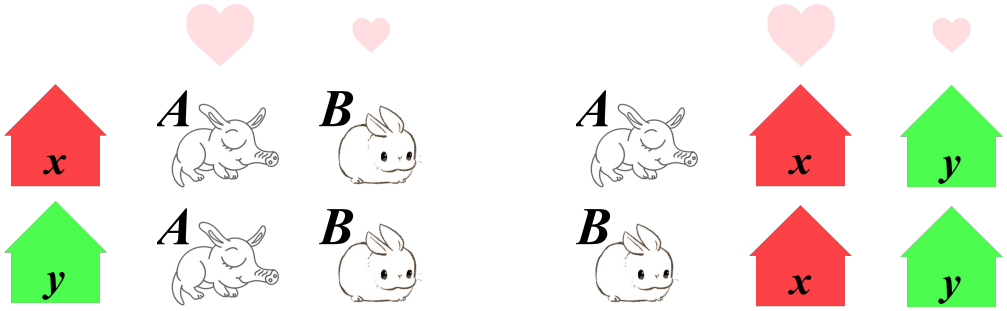
Stable matchings for pet adoption

Given: *n* families looking to adopt *n* animals

- Each family ranks all animals
- Each animal ranks all families
- For simplicity, assume each family can only adopt one animal

Can we match animals to families in a ***stable*** way?

- *Stable*: Don't match *(f, j')* and *(f', j)* if *f* and *j* would both prefer to be matched with each other



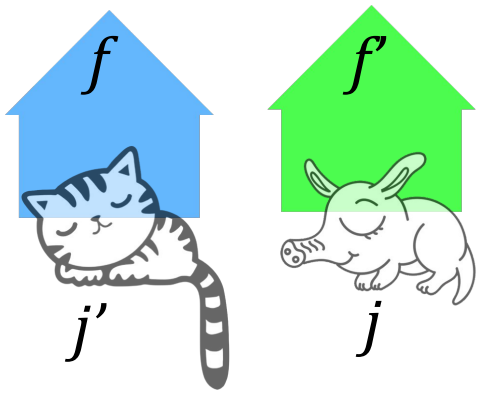
Step 1: precise problem formulation

Stable matching problem

Input: preference lists for n families and n animals

Output: ?? need definitions first

- **Matching**: set M of family-animal pairs
  each family/animal participates in *at most* one pair
- **Perfect matching**: each family/animal in *exactly* one pair
- **Instability** or unstable pair (with respect to matching M):
  a pair $(f, j) \notin M$ such that
  - $(f, j') \in M$ but f prefers j to j'
  - $(f', j) \in M$ but j prefers f to f'



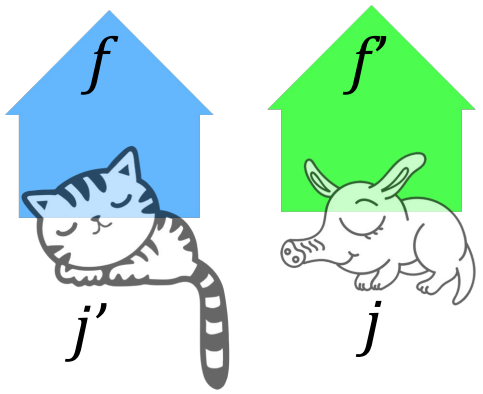
Step 1: precise problem formulation

Stable matching problem

Input: preference lists for n families and n animals

Output: ?? need definitions first

- **Matching**: set M of family-animal pairs
  each family/animal participates in *at most* one pair
- **Perfect matching**: each family/animal in *exactly* one pair
- **Instability** or unstable pair (with respect to matching M):
  a pair $(f, j) \notin M$ such that
  - $(f, j') \in M$ but f prefers j to j'
  - $(f', j) \in M$ but j prefers f to f'
- **Stable matching**:
  perfect matching with no instabilities



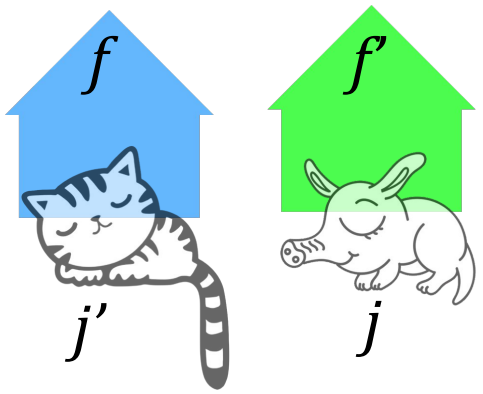
Step 1: precise problem formulation

Stable matching problem

Input: preference lists for n families and n animals

Output: a stable matching

- **Matching**: set M of family-animal pairs
  each family/animal participates in *at most* one pair
- **Perfect matching**: each family/animal in *exactly* one pair
- **Instability** or unstable pair (with respect to matching M):
  a pair $(f, j) \notin M$ such that
  - $(f, j') \in M$ but f prefers j to j'
  - $(f', j) \in M$ but j prefers f to f'
- **Stable matching**:
  perfect matching with no instabilities

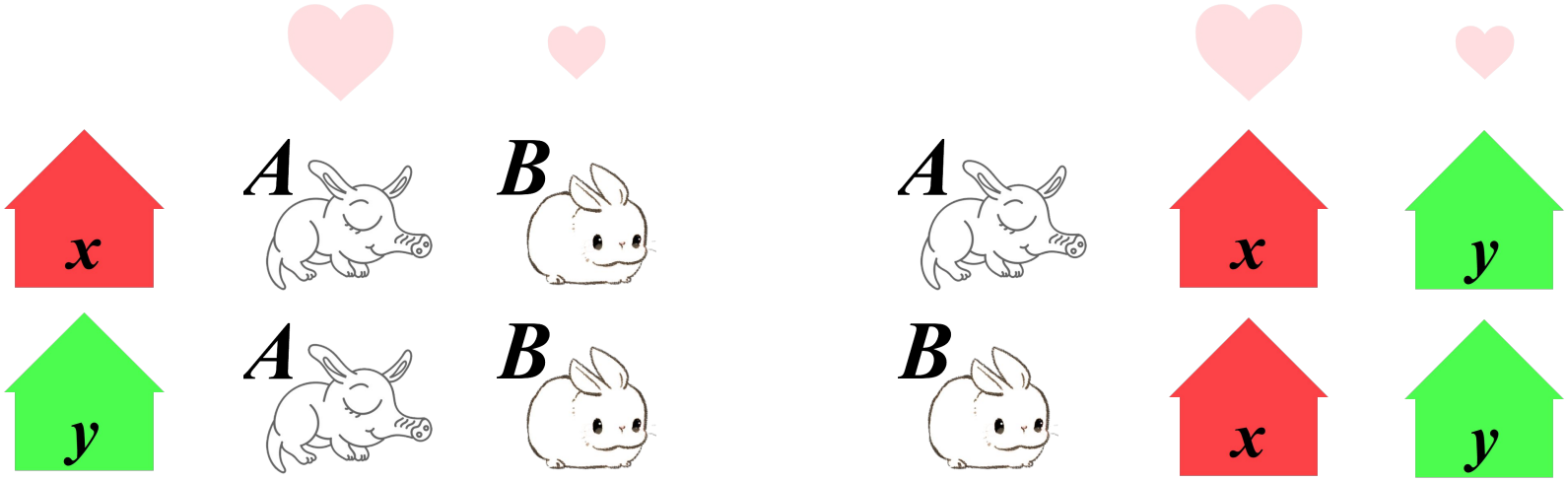


Stable matchings

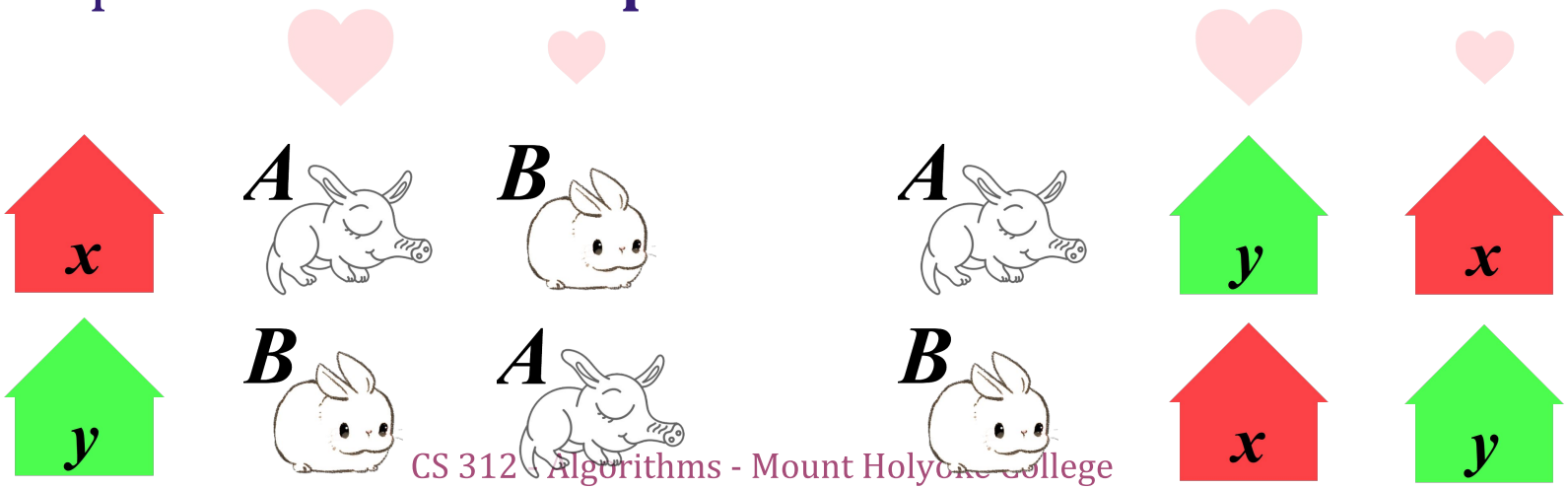
- Do they always exist?
- Are they unique?

What are the possible perfect matchings? Which are stable?

Example 1. **Universal prefs**



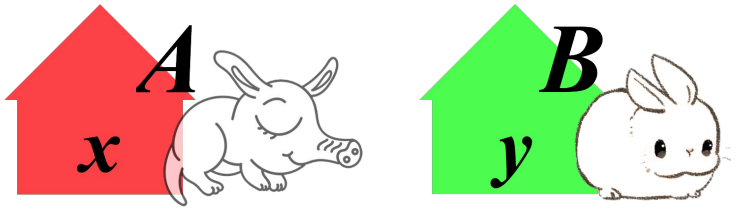
Example 2. **Inconsistent prefs**



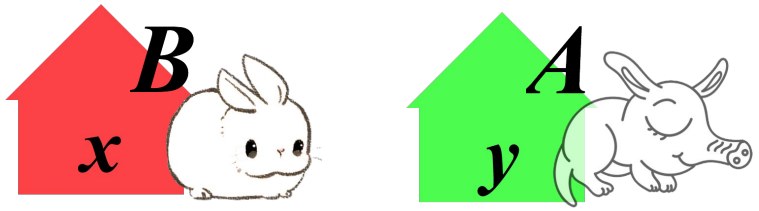
Perfect matchings

2 families, 2 animals

$M = \{(x,A),(y,B)\}$



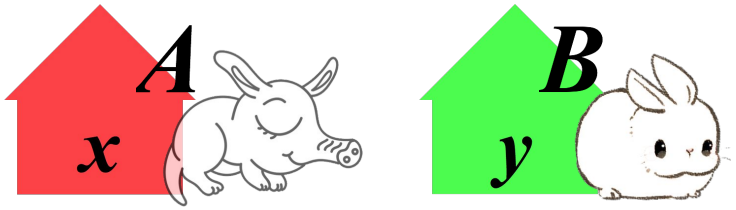
$M' = \{(x,B),(y,A)\}$



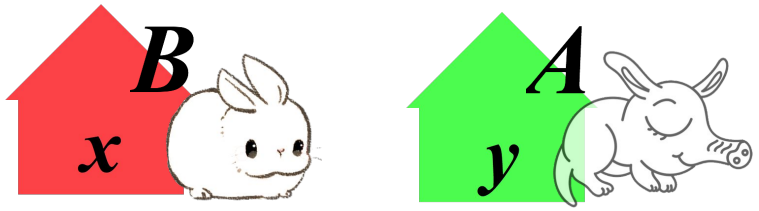
Stable?

2 families, 2 animals

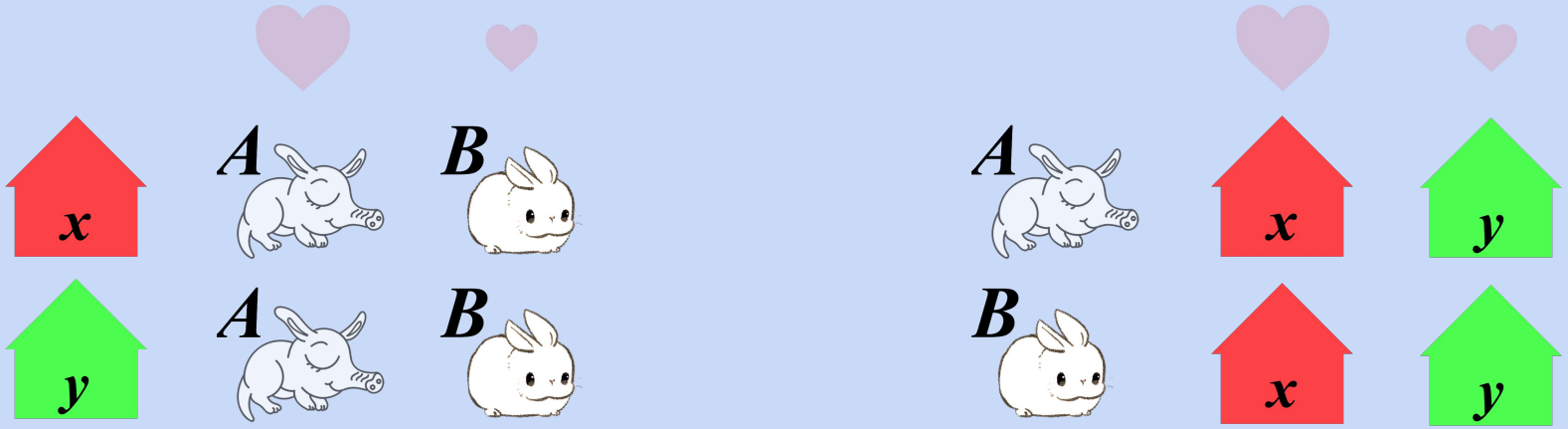
$M = \{(x,A),(y,B)\}$ ✔



$M' = \{(x,B),(y,A)\}$ ✘



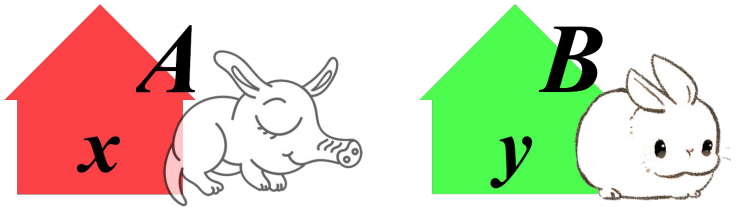
Universal preferences



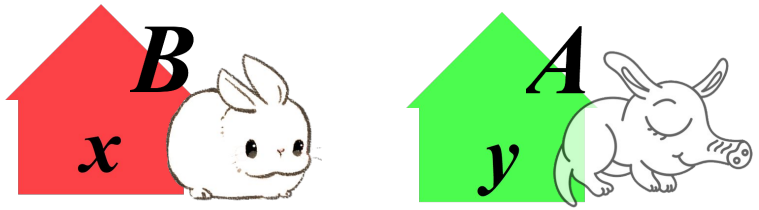
Stable?

2 families, 2 animals

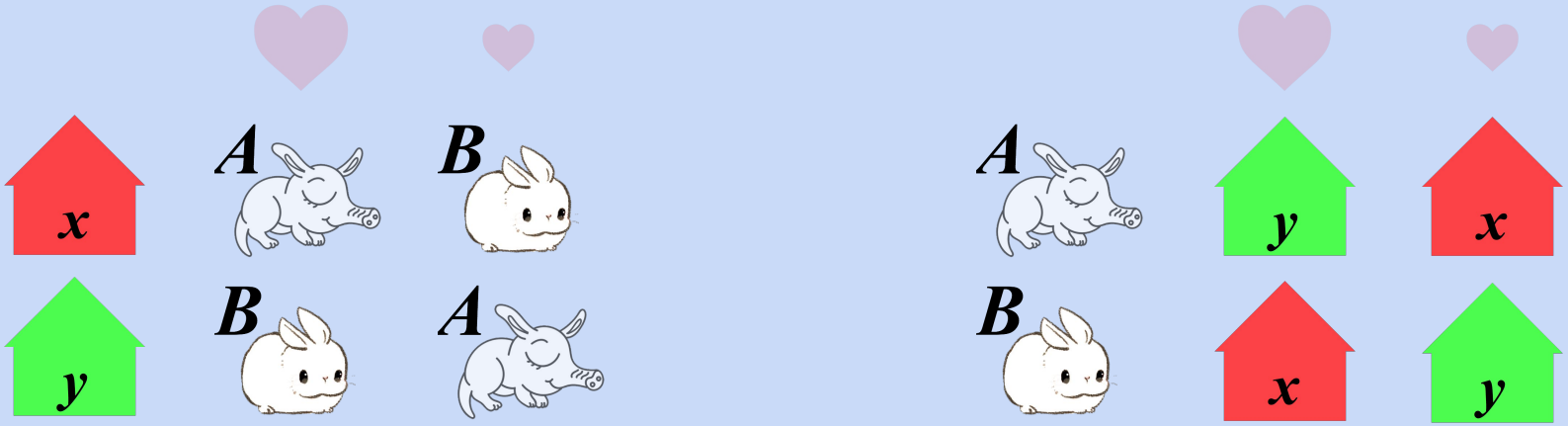
$M = \{(x,A),(y,B)\}$ ✓



$M' = \{(x,B),(y,A)\}$ ✓



Inconsistent preferences



Algorithm design process: step 2

Algorithm:
computational procedure correctly solving a problem

1. pose the problem rigorously
   (formulate a *mathematically* clean definition)
2. propose a solution as a procedure
   (description can be *implemented* within specified computational model, e.g., a random-access machine)

Algorithm design process: step 2

What would a "brute force" approach be?

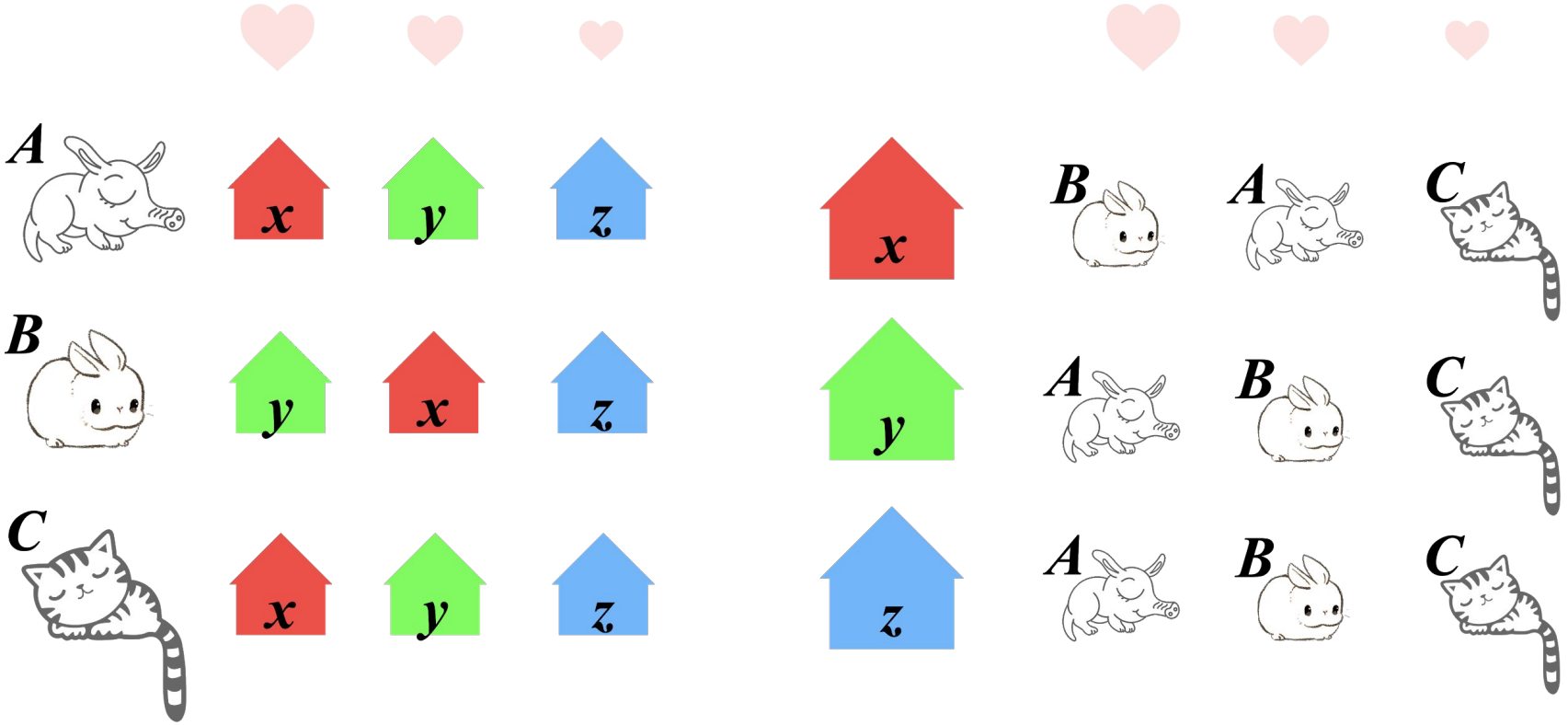
Algorithm:

computational procedure correctly solving a problem

1. pose the problem rigorously
   (formulate a *mathematically* clean definition)
2. propose a solution as a procedure
   (description can be *implemented* within specified computational model, e.g., a random-access machine)

Towards an algorithm: build M incrementally

A family walks into the shelter…



Propose-and-Reject (Gale-Shapley) Algorithm

Initialize all families and animals as **free**
`while` some family is **free** and hasn't proposed to every animal
    Choose such a family *f*
    Let *a* be the highest ranked animal to whom *f* has not yet proposed
    `if` *a* is **free**
        f and *a* become **committed**
    `else` // *a* is **committed** to *f'*
        `if` *a* prefers *f*
            f' becomes **free**
            f and *a* become **committed**
        `else`
            a rejects *f* and *f* remains **free**

Algorithm Design Process

Algorithm design process: analysis (steps 3 and 4)

Algorithm:
computational procedure correctly solving a problem

1. pose the problem rigorously
   (formulate a *mathematically* clean definition)
2. propose a solution as a procedure
   (description can be *implemented* within specified
   computational model, e.g., a random-access machine)
3. prove that the procedure correctly solves the problem
4. analyze the procedure's running **time**

Algorithm analysis: towards a proof

Some natural questions:

- Can we guarantee every family and animal gets a match?
- Can we guarantee the resulting adoptions are stable?
- Can we guarantee the algorithm terminates?

Some initial observations:

- (F1) Once matched, an animal stays matched and only “upgrades" during the algorithm.
- (F2) Each family proposes to animals in order of family’s preferences.

Algorithm design process: analysis (steps 3 and 4)

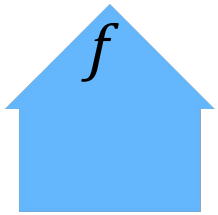
Algorithm:
computational procedure correctly solving a problem

1. pose the problem rigorously
   (formulate a *mathematically* clean definition)
2. propose a solution as a procedure
   (description can be *implemented* within specified
   computational model, e.g., a random-access machine)
3. prove that the procedure correctly solves the problem
4. analyze the procedure's running **time**

Can we guarantee a perfect matching?

Yes! Proof by contradiction...

- Suppose not all families and animals have matches
  - Then there exists unmatched family *f* and unmatched animal *a*
- *a* was never matched during the algorithm (by F1)
- But *f* proposed to every animal (by termination condition)
- When *f* proposed to *a*, it was unmatched and yet rejected *f*
- Contradiction!



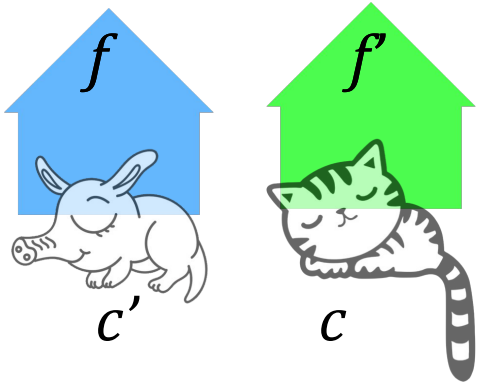
a

(F1) Once matched, an animal stays matched and only “upgrades"

Can we guarantee the resulting adoptions are stable?

Yes! Proof by contradiction.

- Suppose there is an instability *(f, c)*
  - *f* is matched to *c′* but prefers *c* to *c′*
  - *c* is matched to *f′* but prefers *f* to *f′*

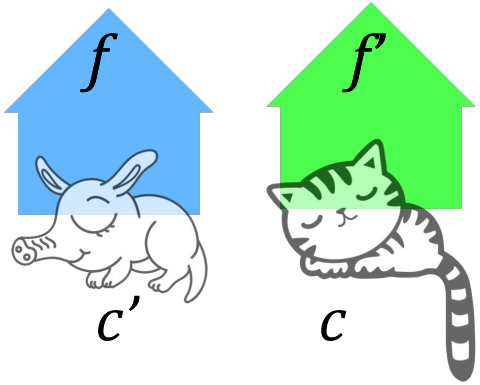


Can we guarantee the resulting adoptions are stable?

Yes! Proof by contradiction.

(F2) Each family proposes to animals in order of family's preferences.

- Suppose there is an instability *(f, c)*
  - *f* is matched to *c′* but prefers *c* to *c′*
  - *c* is matched to *f′* but prefers *f* to *f′*
- By (F2), *f* must have proposed to *c* before proposing and becoming committed to *c′*

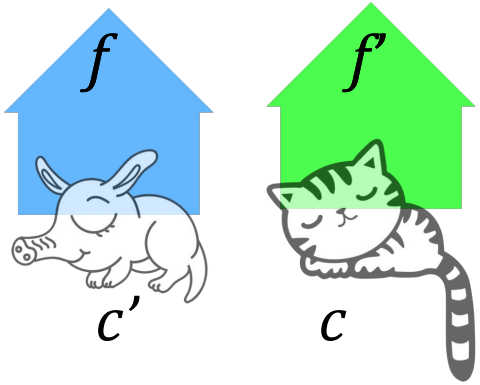


Can we guarantee the resulting adoptions are stable?

Yes! Proof by contradiction.

- Suppose there is an instability *(f, c)*
  - *f* is matched to *c′* but prefers *c* to *c′*
  - *c* is matched to *f′* but prefers *f* to *f′*
- By (F2), *f* must have proposed to *c* before proposing and becoming committed to *c′*
- Since *f* isn't matched to *c* at the end of the algorithm, it must have rejected *f* (either immediately or upon receiving a better proposal). By (F1), it prefers its final match *f′* to *f*.

(F1) Once matched, an animal stays matched and only "upgrades"



Propose-and-Reject (Gale-Shapley) Algorithm

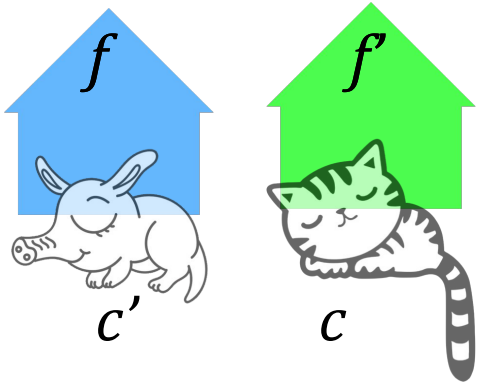
Initialize all families and animals as **free**
`while` some family is **free** and hasn't proposed to every animal
    Choose such a family *f*
    Let *a* be the highest ranked animal to whom *f* has not yet proposed
    `if` *a* is **free**
        f and *a* become **committed**
    `else` // *a* is **committed** to *f'*
        `if` *a* prefers *f*
            f' becomes **free**
            f and *a* become **committed**
        `else`
            a rejects *f* and *f* remains **free**

Can we guarantee the resulting adoptions are stable?

Yes! Proof by contradiction.

- Suppose there is an instability *(f, c)*
  - *f* is matched to *c′* but prefers *c* to *c′*
  - *c* is matched to *f′* but prefers *f* to *f′*
- By (F2), *f* must have proposed to *c* before proposing and becoming committed to *c′*
- Since *f* isn't matched to *c* at the end of the algorithm, it must have rejected *f* (either immediately or upon receiving a better proposal). By (F1), it prefers its final match *f′* to *f*.

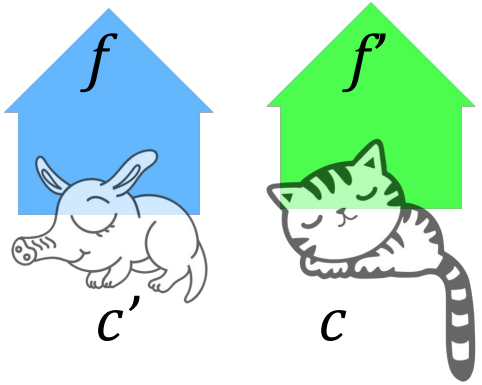
(F1) Once matched, an animal stays matched and only "upgrades"



Can we guarantee the resulting adoptions are stable?

Yes! Proof by contradiction.

- Suppose there is an instability *(f, c)*
  - *f* is matched to *c′* but prefers *c* to *c′*
  - *c* is matched to *f′* but prefers *f* to *f′*
- By (F2), *f* must have proposed to *c* before proposing and becoming committed to *c′*
- Since *f* isn't matched to *c* at the end of the algorithm, it must have rejected *f* (either immediately or upon receiving a better proposal). By (F1), it prefers its final match *f′* to *f*.
- Contradiction!



Algorithm design process: analysis (steps 3 and 4)

Algorithm:
computational procedure correctly solving a problem

1. pose the problem rigorously
   (formulate a *mathematically* clean definition)
2. propose a solution as a procedure
   (description can be *implemented* within specified computational model, e.g., a random-access machine)
3. prove that the procedure correctly solves the problem
4. analyze the procedure's running **time**

Can we guarantee the algorithm terminates?

Yes! Proof...

- In every round, some family proposes to some animal that they haven't already proposed to
- n families and n animals $\Rightarrow$ at most n^2 proposals

  $\Rightarrow$ at most n^2 rounds of the algorithm

But what about running time?

Yes! Proof...

- In every round, some family proposes to some animal that they haven't already proposed to
- n families and n animals $\Rightarrow$ at most n^2 proposals

  $\Rightarrow$ **at most n^2 rounds** of the algorithm

How do we formalize this in terms of **running time**?