

Introduction & Course Overview

SS 2020— Parallel Programming

Dr. Malte Schwerhoff, Dr. Hermann Lehner

Slides (mainly) from Prof. Martin Vechev, Prof. Otmar Hilliges, Dr. Felix Friedrich



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

About this course

Lecturers:



Dr. Malte Schwerhoff

Dr. Hermann Lehner

UNG F 15/16

{firstname.lastname}@inf.ethz.ch

Teach Part I

Office hours: per email request



Prof. Torsten Hoefler

CAB F 75

torsten.hoefler@inf.ethz.ch







Teaches Part II

Office hours: per email request

<https://spcl.inf.ethz.ch/Teaching/2020-pp/>

About us



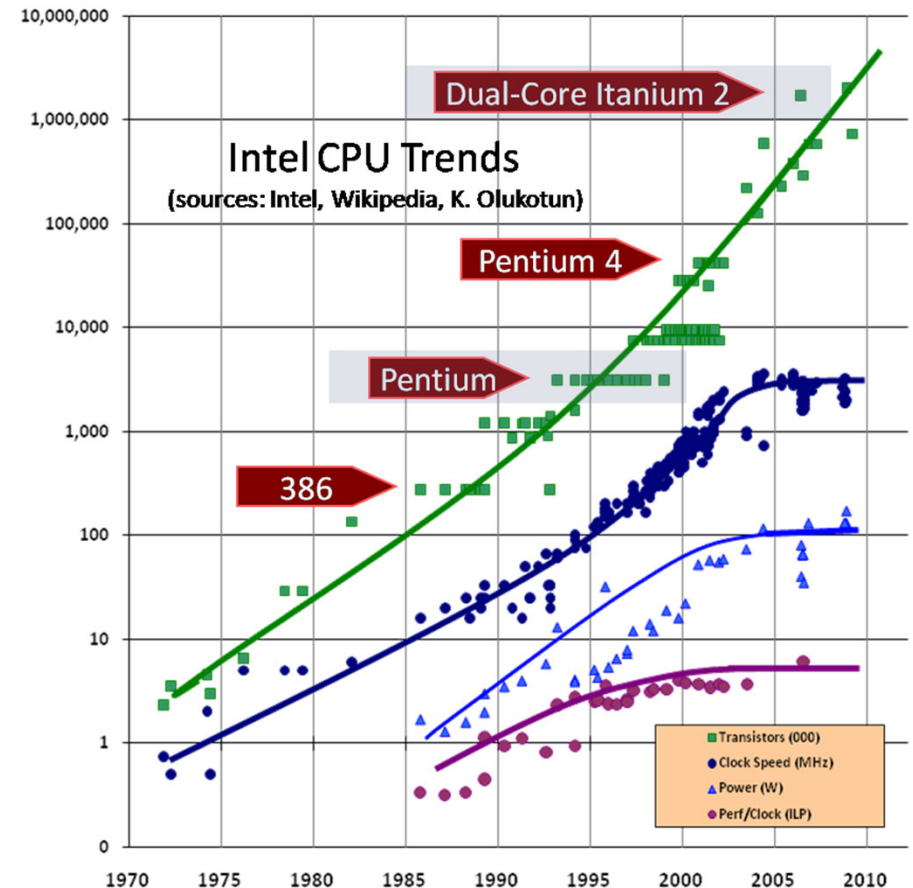
- BSc at FH Gelsenkirchen 
- MSc at ETH 
- PhD at ETH: Language & tools for verifying parallel programs  
- Semesters abroad in Tomsk (Russia), Leuven (Belgium)  
- ETH lecturer since 2017



- Dipl. Informatik-Ingenieur at ETH
- PhD at ETH: Program semantics, formal verification
- 5 years in industry as software engineer and team lead
- ETH lecturer since 2016

Why this course?

1. Parallel programming is a necessity – since 2000-ish
2. A different way of computational thinking – who said everything needs a total order?
3. Generally fun (since always) – if you like to challenge your brain



Course Overview

Parallel Programming (252-0029-00L)

- 4L + 2U
- 7 ECTS Credits
- Audience: Computer Science Bachelor
 - Part of Basisprüfung
- Lecture Language: Denglisch

Course Coordination

Communication via course website:

<https://spcl.inf.ethz.ch/Teaching/2020-pp/>

- Lectures 2 x week:
 - Tuesday 10-12 HG F 7 (video transmission HG F 5)
 - Wednesday 13-15 HG F 7 (video transmission HG F 5)
- Weekly Exercise Sessions
 - Enroll via myStudies
 - Wednesday 15-17 or Friday 10-12

About this Course

Head TAs:

- Pavol Bielik (first part)
- Timo Schneider (second part)

Teaching Assistants:

- | | |
|--------------------------|-----------------------|
| • Yishai Oltchik | • Victor Cornillere |
| • Salvatore Di Girolamo | • Marcin Copik |
| • Andrei Ivanov | • Neville Walo |
| • Grzegorz Kwasniewski | • Felix Stöger |
| • Johannes de Fine Licht | • Lasse Meinen |
| • Nikoli Dryden | • Robin Renggli |
| • Alexandros Ziogas | • Soel Micheletti |
| • Marc Ficher | • Patrick Wicki |
| • Velko Vechev | • Andreas Bergmeister |
| • Enis Ulqinaku | |

Grades:

- Class is part of Basisprüfung: written, centralized exam after the term
- 100% of grade determined by final exam
- Exercises not graded but essential

Academic Integrity

- Zero tolerance cheating policy (cheat = fail + being reported)
- Homework
 - Don't look at other students code
 - Don't copy code from anywhere
 - Ok to discuss things – but then you have to do it alone
 - Code may be checked with tools
- Don't copy-paste
 - Code
 - Text
 - Images

We might check this with automated tools!

Course Overview

aka why should you care?

How does this course fit into the CS curriculum?

- Programming-in-the-small => Data Structures and Algorithms



Program = Algorithms + Data Structures

- Programming-in-the-large

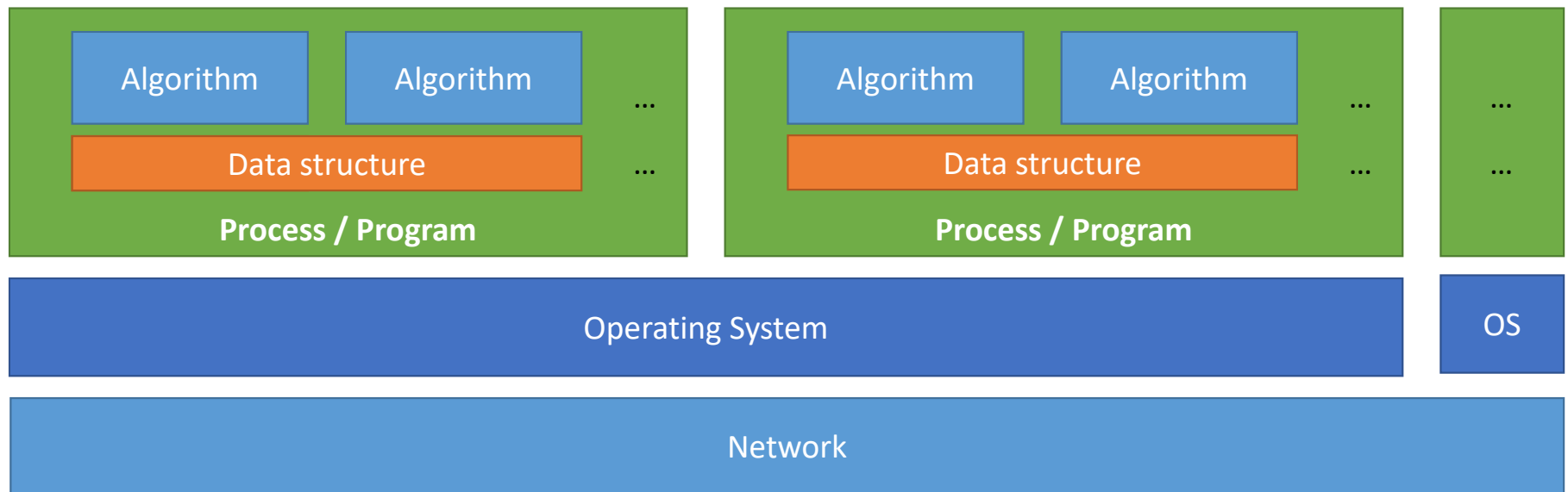
System = Processes + Objects + Communication

↑
This class

↑
Intro to Programming

↑
This class

How does this course fit into the CS curriculum?



Learning Objectives

By the end of the course you should

1. have mastered **fundamental concepts in parallelism**
2. know how to **construct parallel algorithms** using different parallel programming paradigms (e.g., task parallelism, data parallelism) and mechanisms (e.g., threads, tasks, locks, communication channels).
3. be qualified to **reason about correctness** and **performance** of parallel algorithms
4. be ready to **implement parallel programs** for **real-world application tasks** (e.g. searching large data sets)

Requirements

Basic understanding of Computer Science concepts

Basic knowledge of programming concepts:

We will do a *quick* review of Java and briefly discuss JVMs

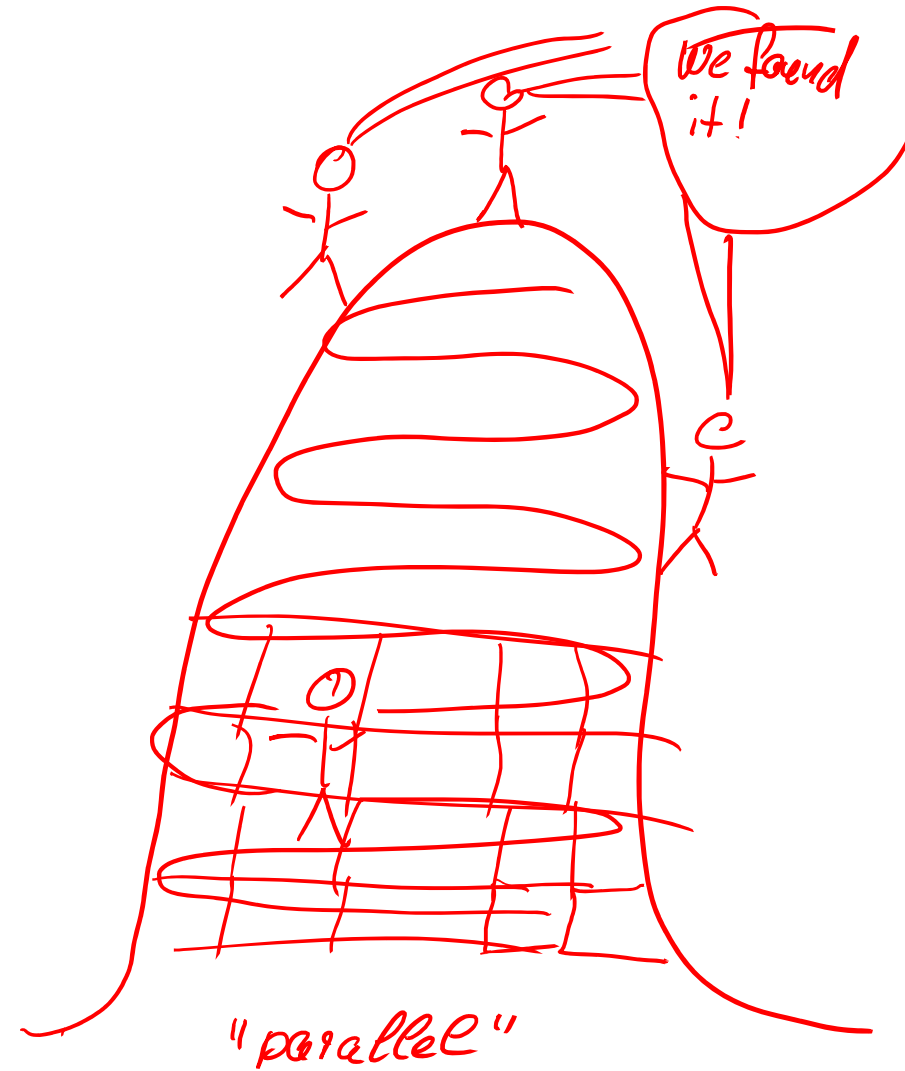
Basic understanding of computer architectures:

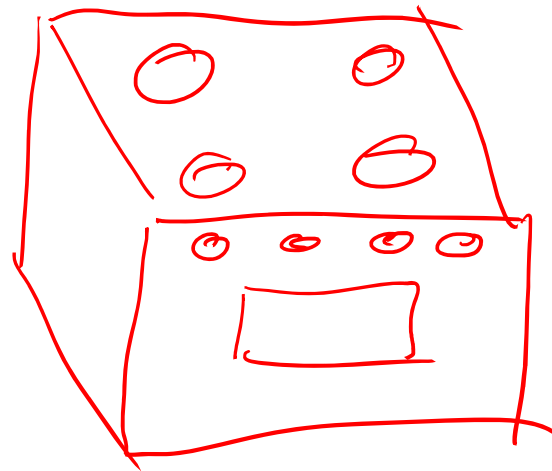
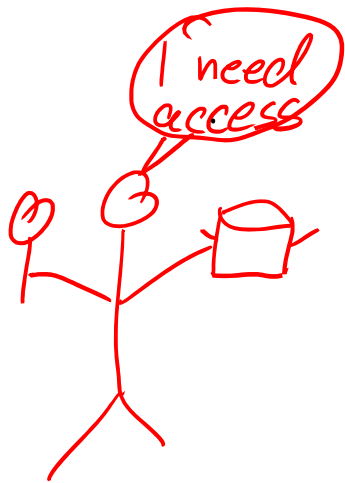
No detailed knowledge necessary (we will cover some)



www.jolyon.co.uk

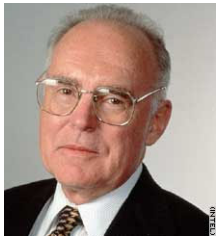
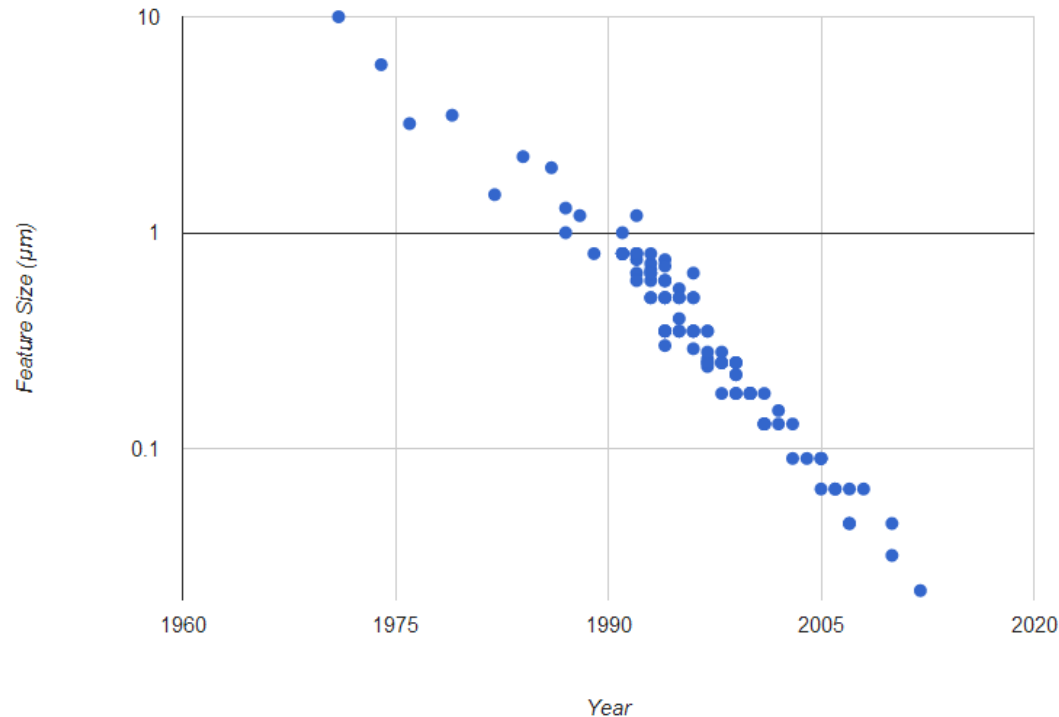
"sequential"





"concurrency"

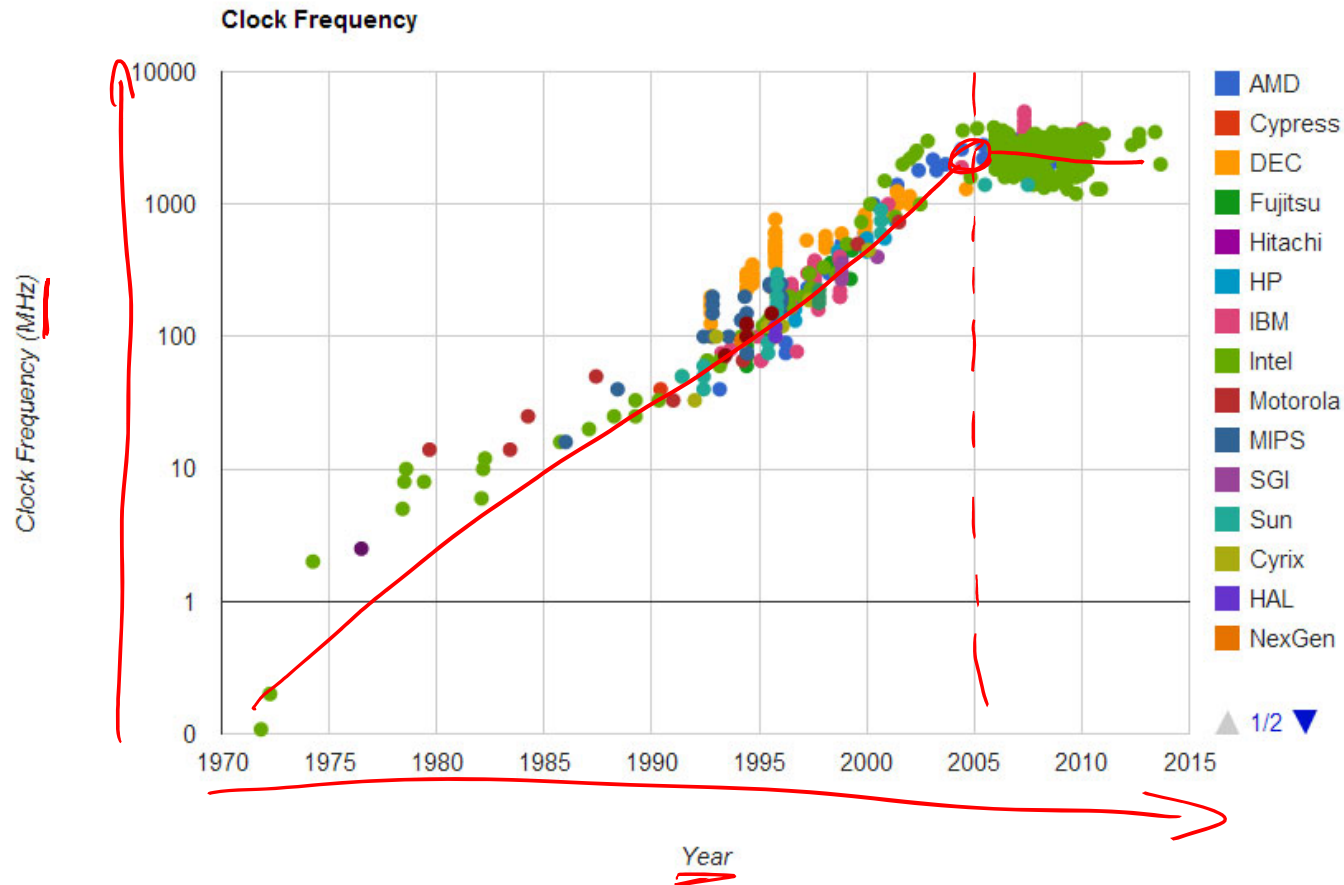
Motivation – Why Parallelism?



Moore's Law Recap: Transistor counts double every two years

- Means: Smaller transistors => can put more on chip => computational power grows exponentially => your sequential program automatically gets faster.
- Also applies to RAM size and pixel densities

Motivation – Why Parallelism?



Why don't we keep increasing clock speeds?

Transistors have *not* stopped getting smaller + faster (Moore lives)

Heat and power have become the primary concern in modern computer architecture!

Consequence:

- Smaller, more efficient Processors
- More processors – often in one package

What kind of processors do we build then?

Main design constraint today is power

- Single-Core CPUs:

- Complex Control Hardware
- **Pro:** Flexibility + Performance!
- **Con:** Expensive in terms of power (Ops / Watt)

- Many-Core/GPUs etc:

- Simpler Control Hardware
- **Pro:** Potentially more power efficient (Ops / Watt)
- **Con:** More restrictive / complex programming models [but useful in many domains, e.g. deep learning].

Class Overview

(Parallel) Programming

- Recap: Programming in Java + a bit of JVM
- Parallelism in Java (Threads)

Parallelism

- Understanding and detecting parallelism
- Intro to PC Architectures
- Formalizing parallelism
- Programming models for parallelism

Concurrency

- Shared data
- Race Conditions
- Locks, Semaphores, etc.
- Lock-free programming
- Communication across tasks and processes

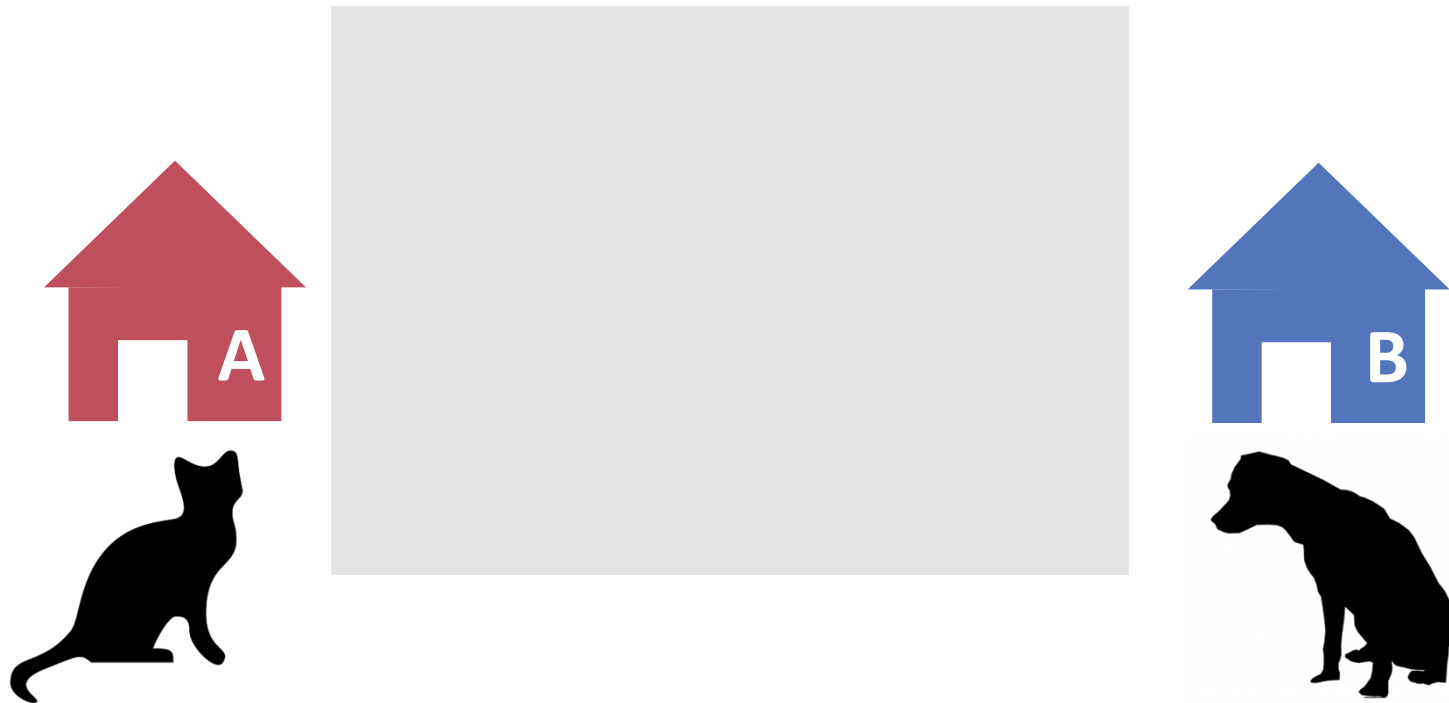
Parallel Algorithms

- Useful & common algorithms in parallel
- Data structures for parallelism
- Sorting & Searching, etc.

Three stories

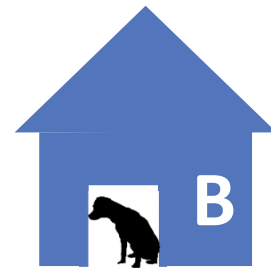
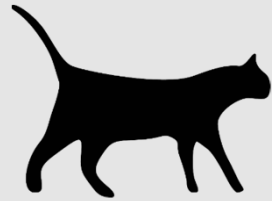
1. MUTUAL EXCLUSION

Alice's Cat vs. Bob's Dog

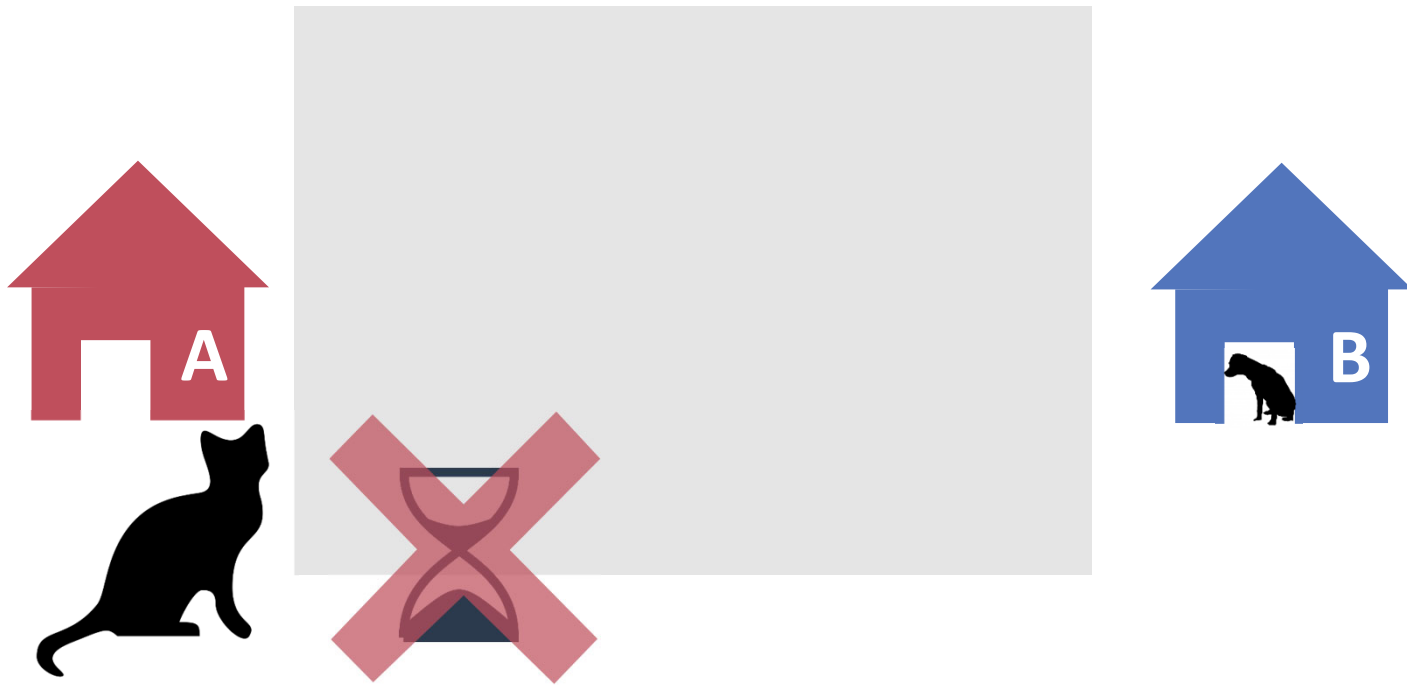


Requirement I: Mutual Exclusion !

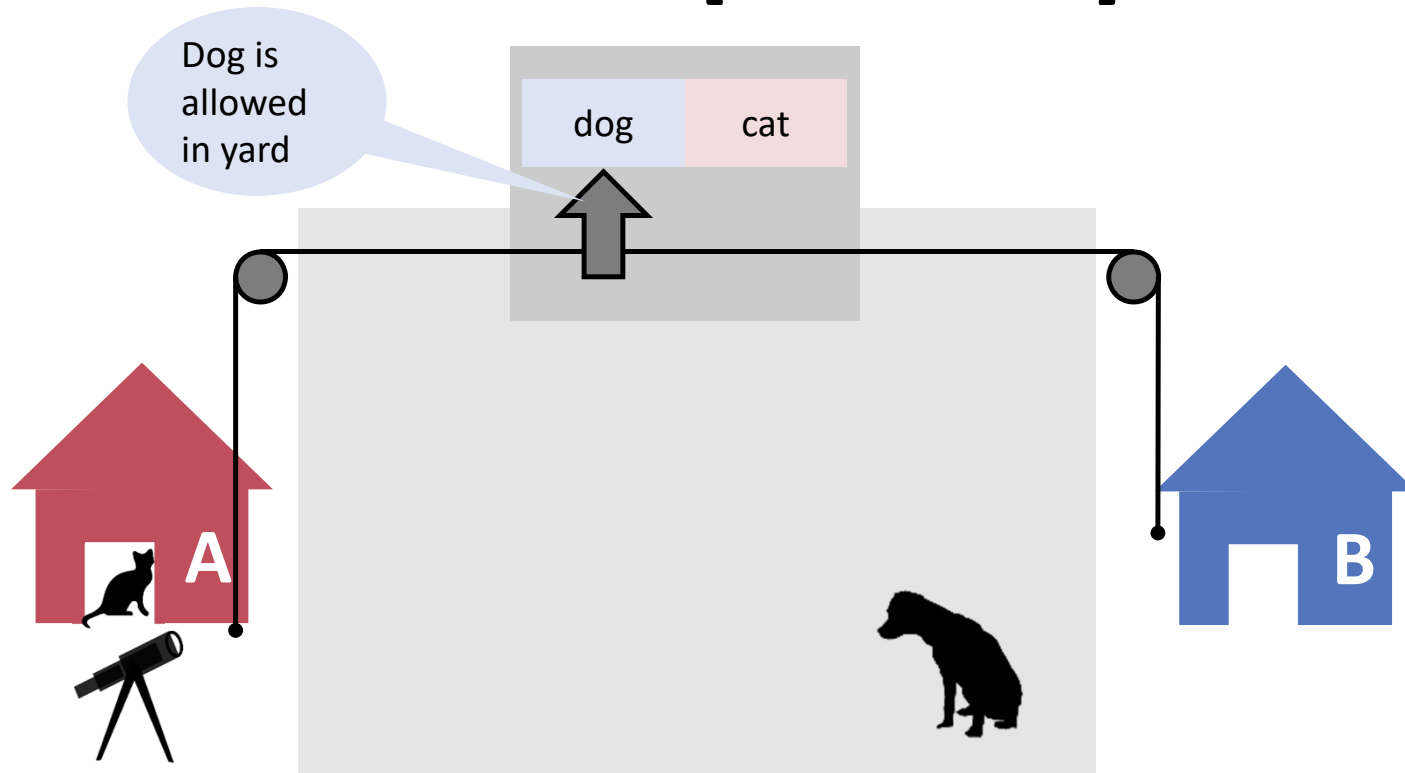




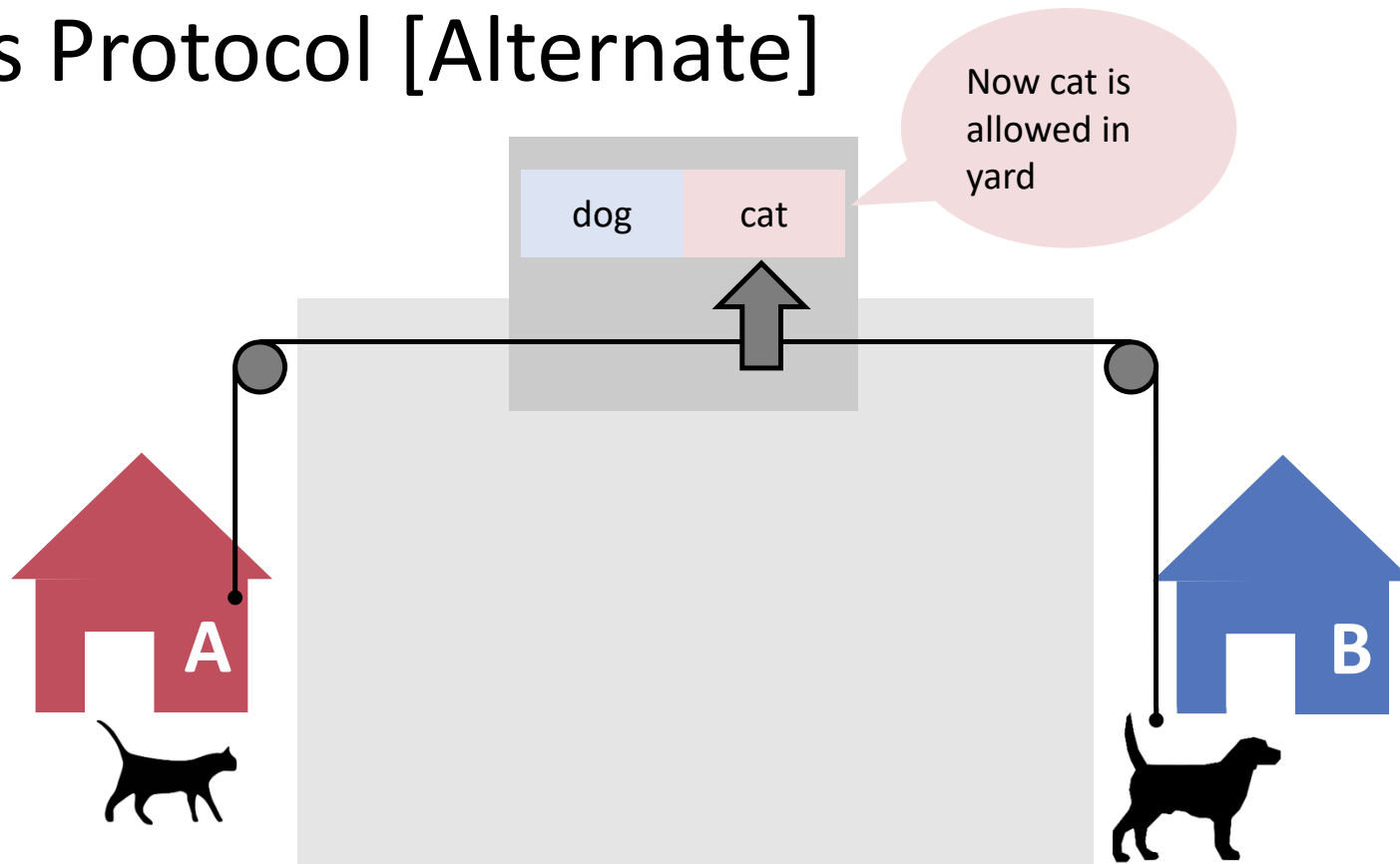
Requirement II: No Lockout when free



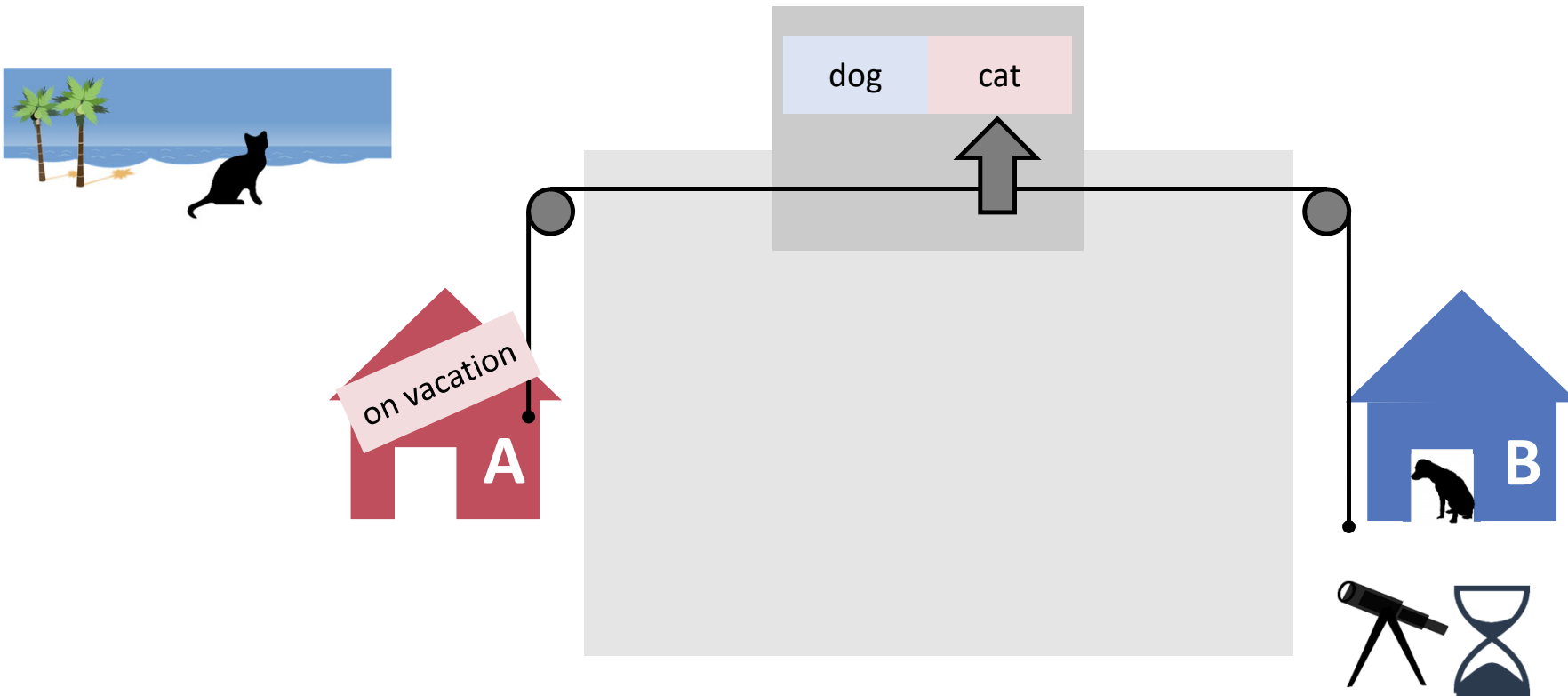
Communication: Idea 1 [Alternate]



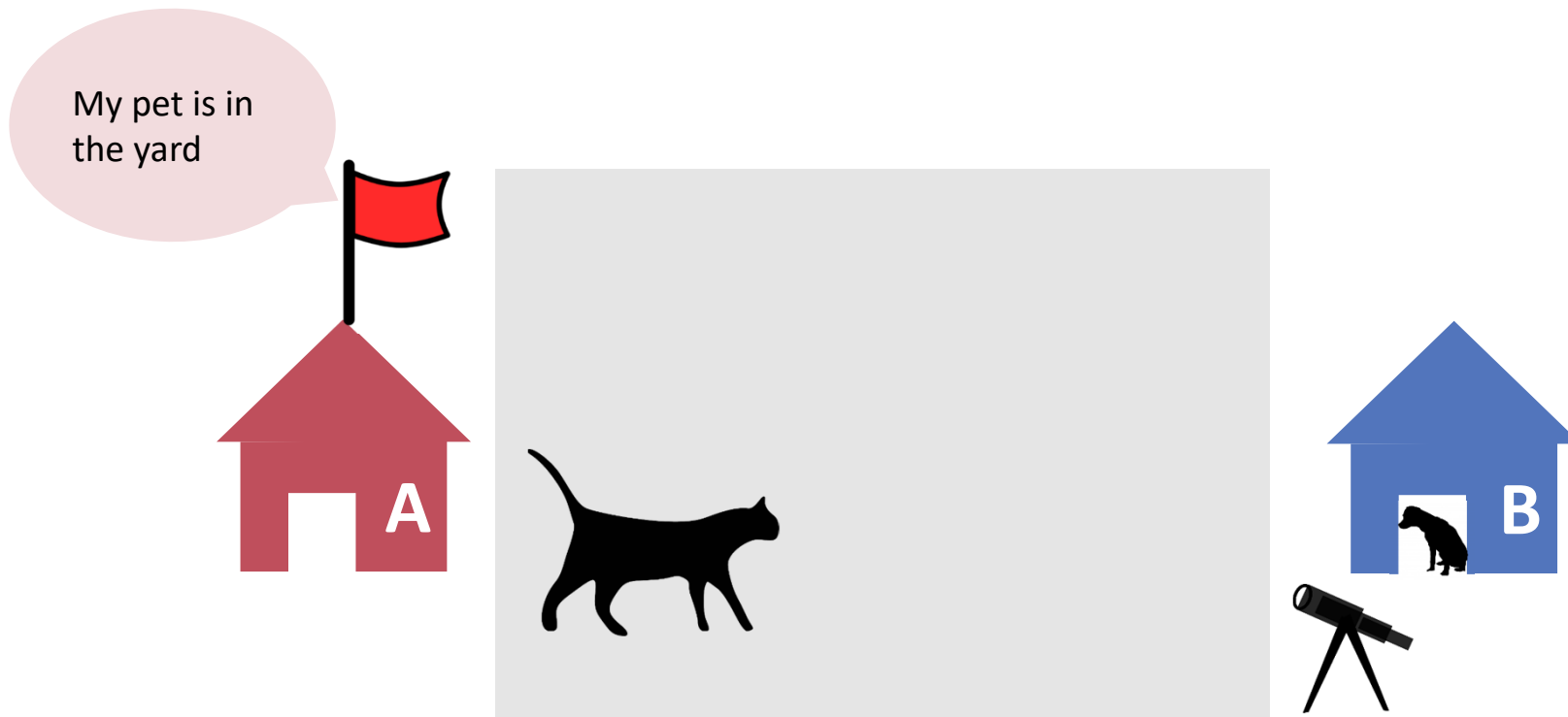
Access Protocol [Alternate]



Problem: starvation!

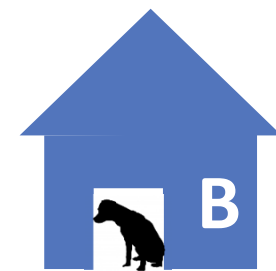
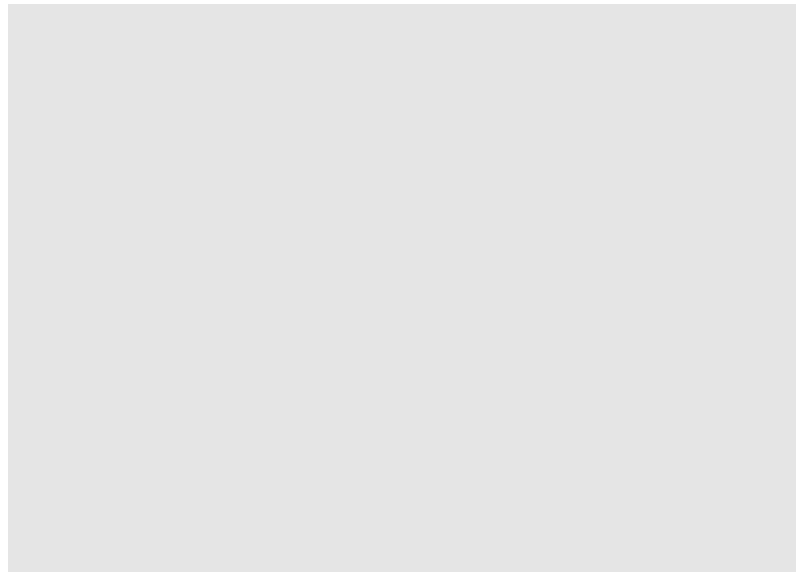
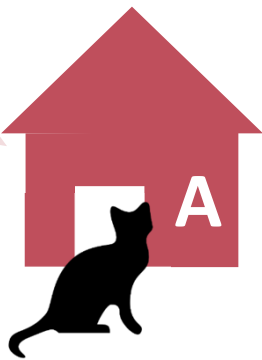


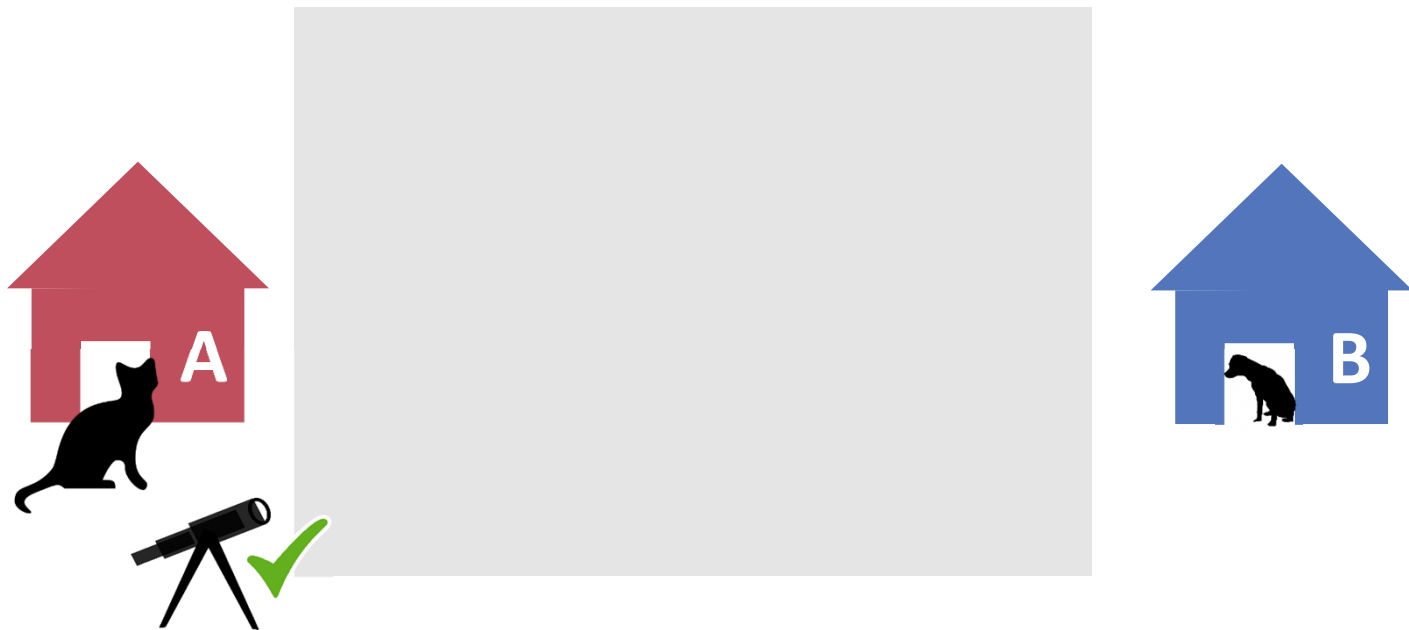
Communication: Idea 2[Notification]

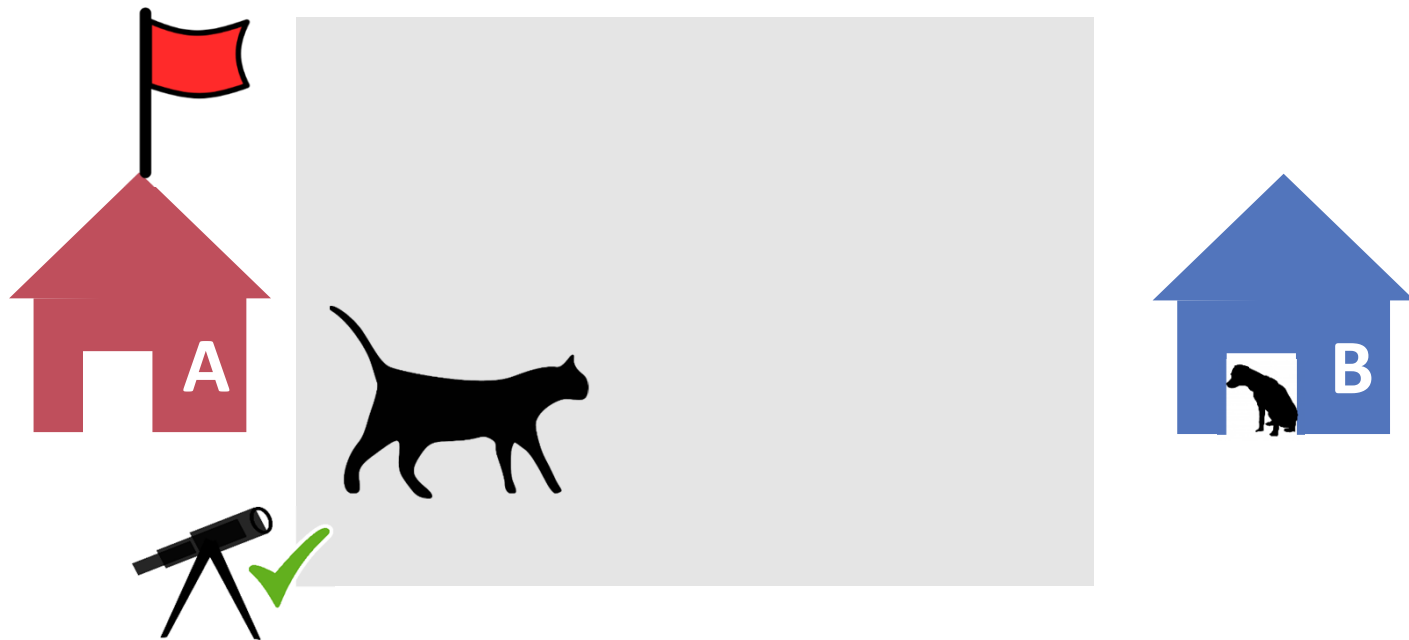


Access Protocol 2.1: Idea

Cat wants to
get out

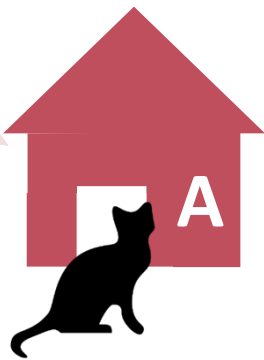




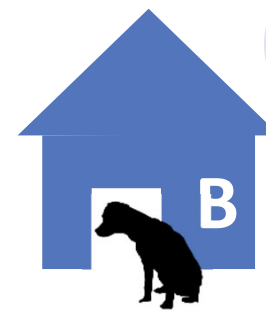


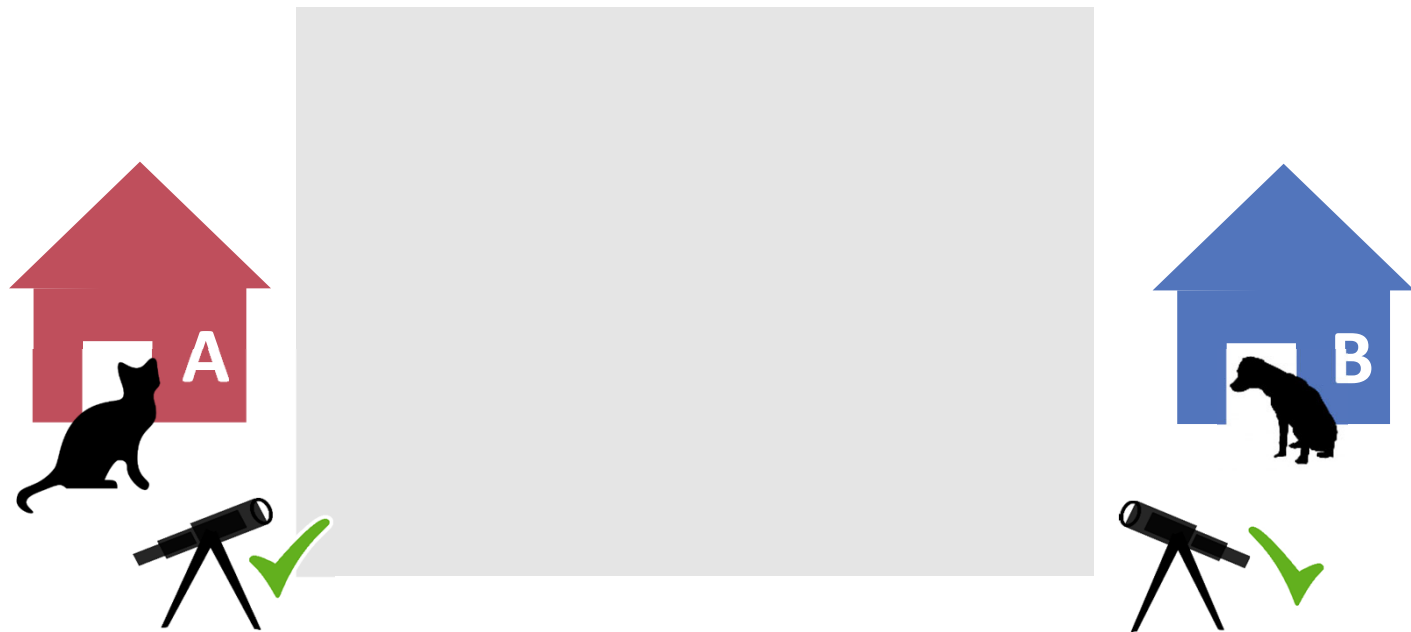
Another Scenario

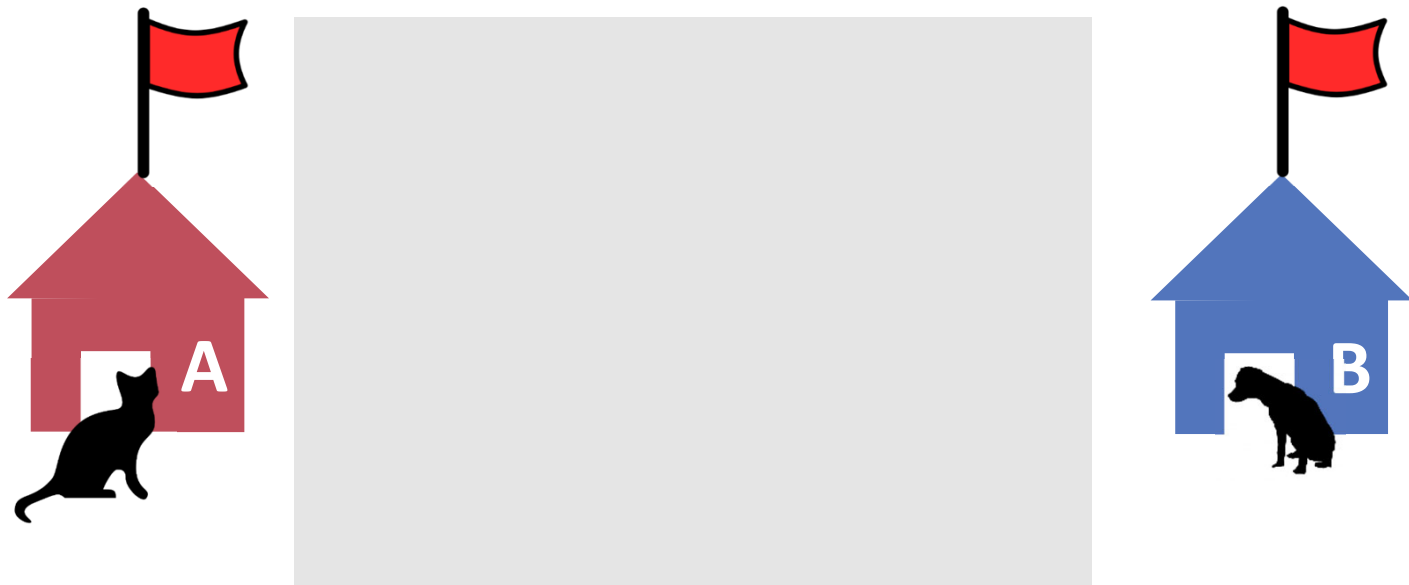
Cat wants to get out



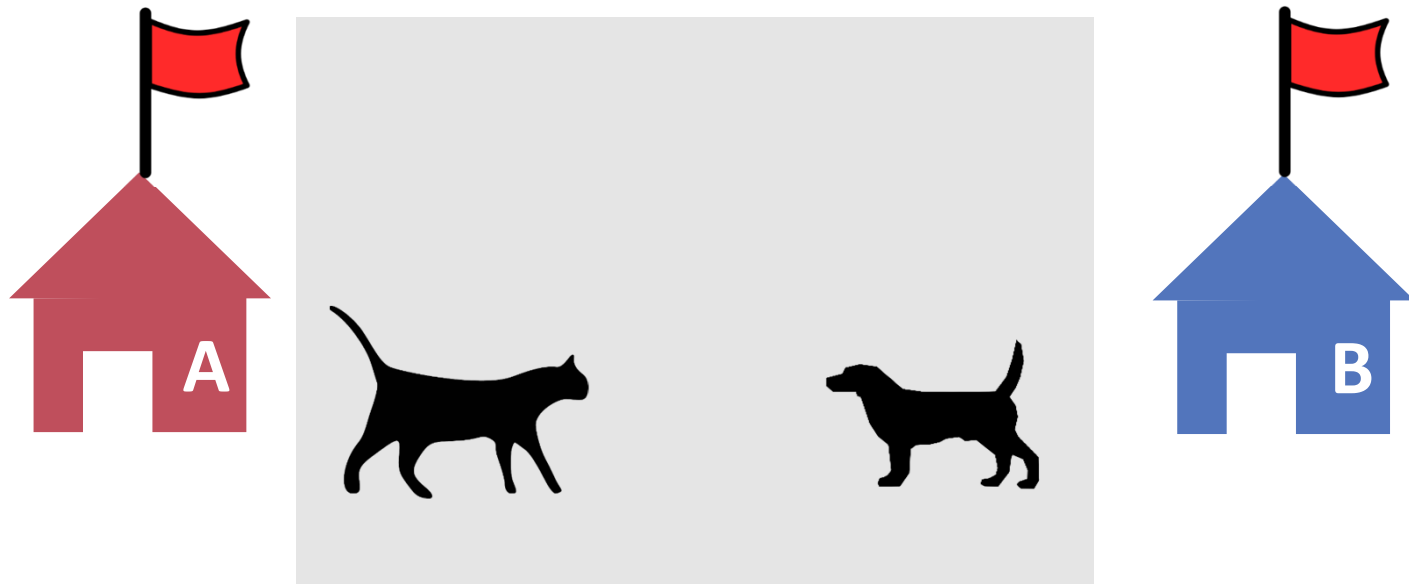
Dog wants to get out



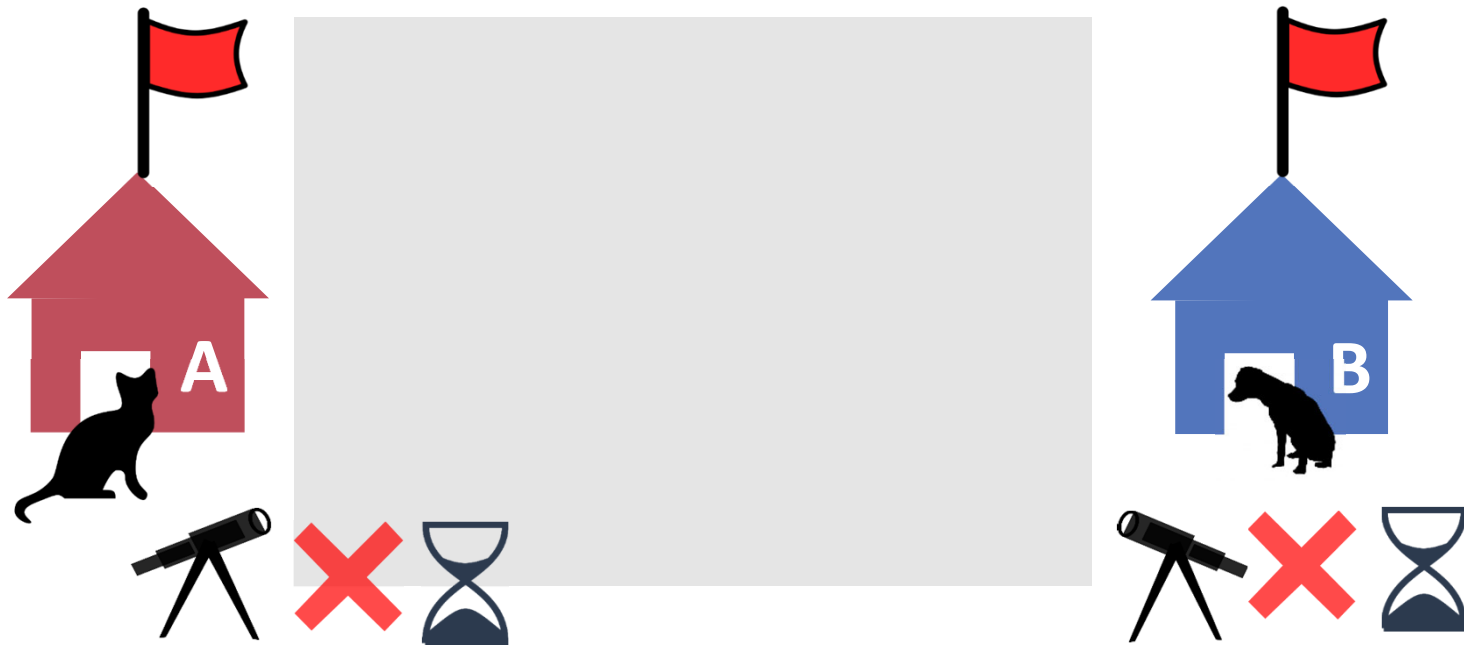




Problem: no Mutual Exclusion!

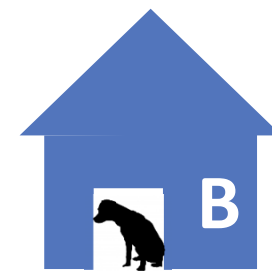
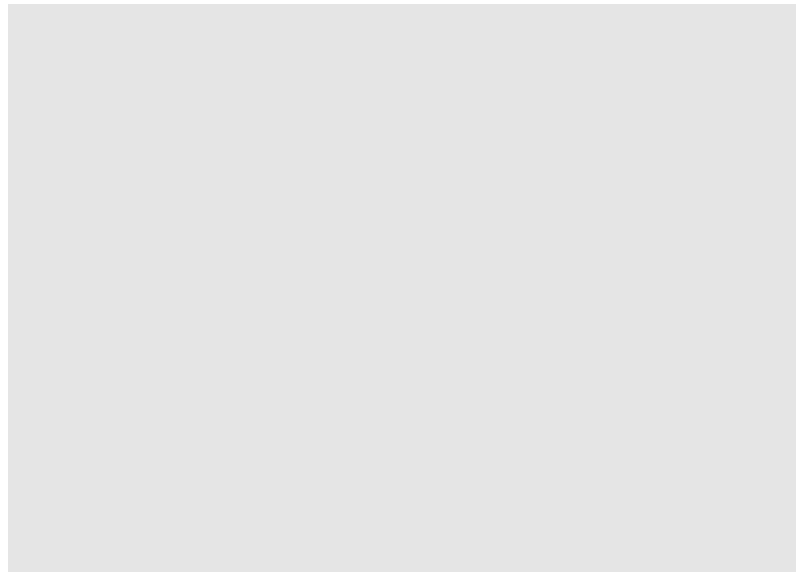
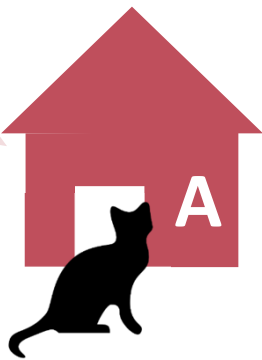


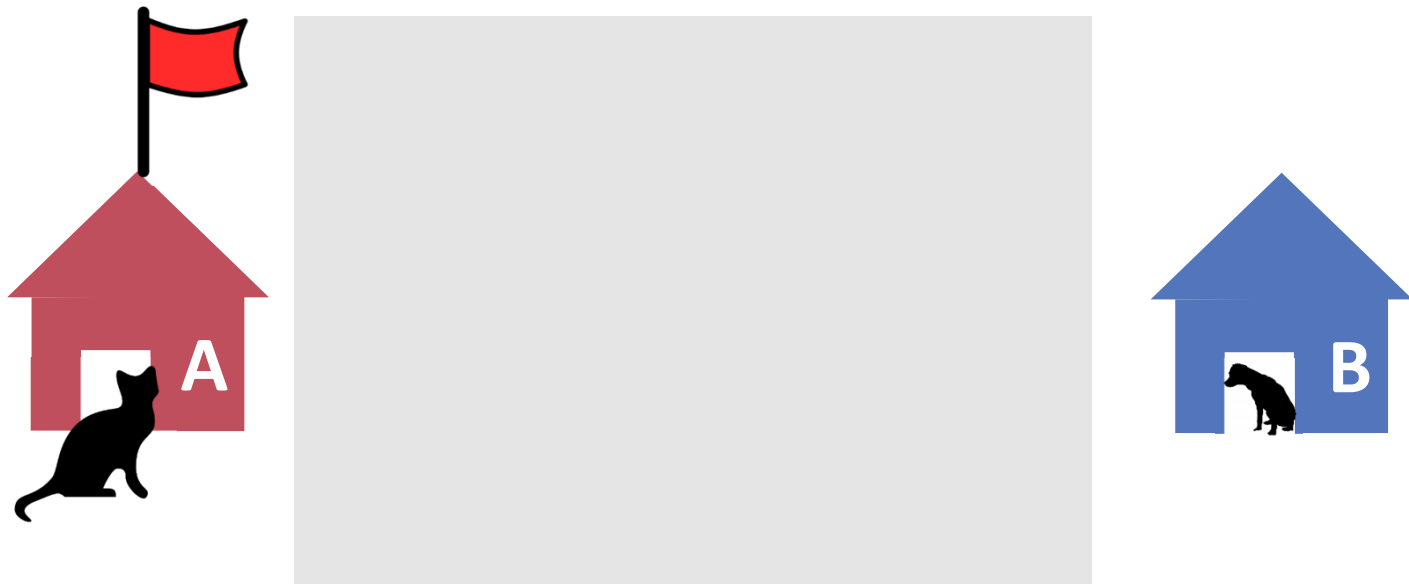
Checking flags twice does not help: deadlock!

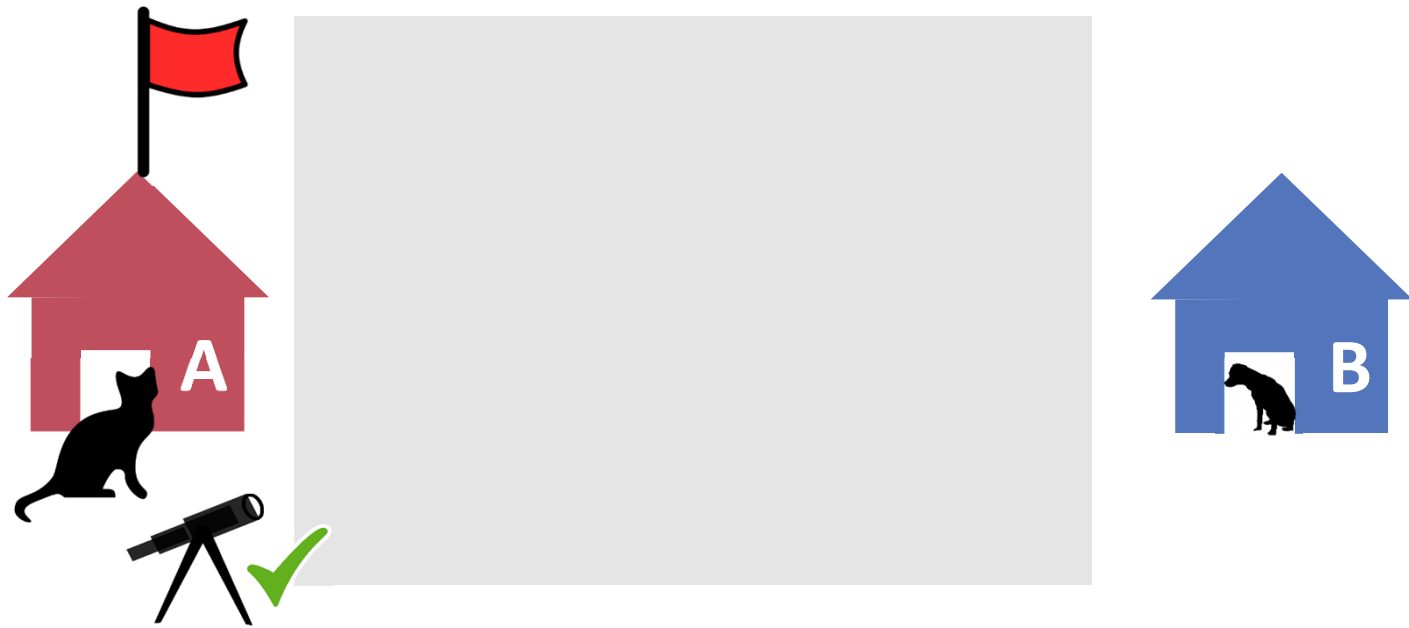


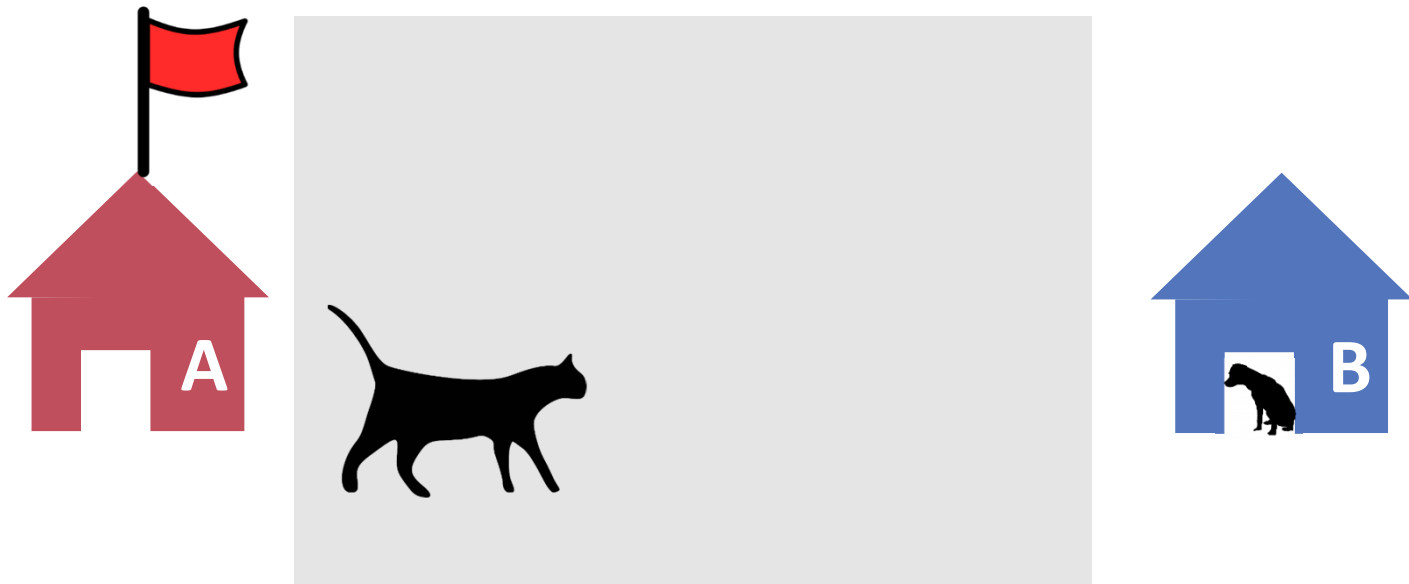
Access Protocol 2.2

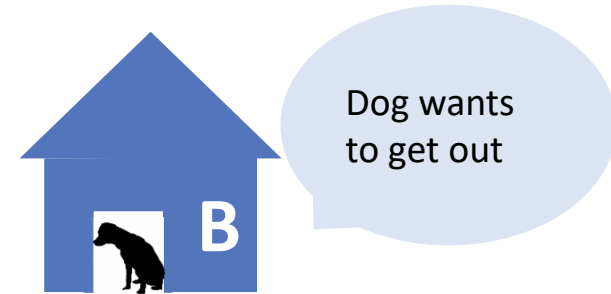
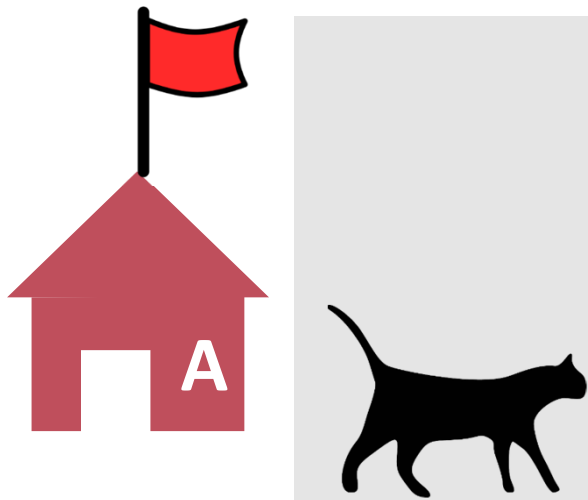
Cat wants to
get out

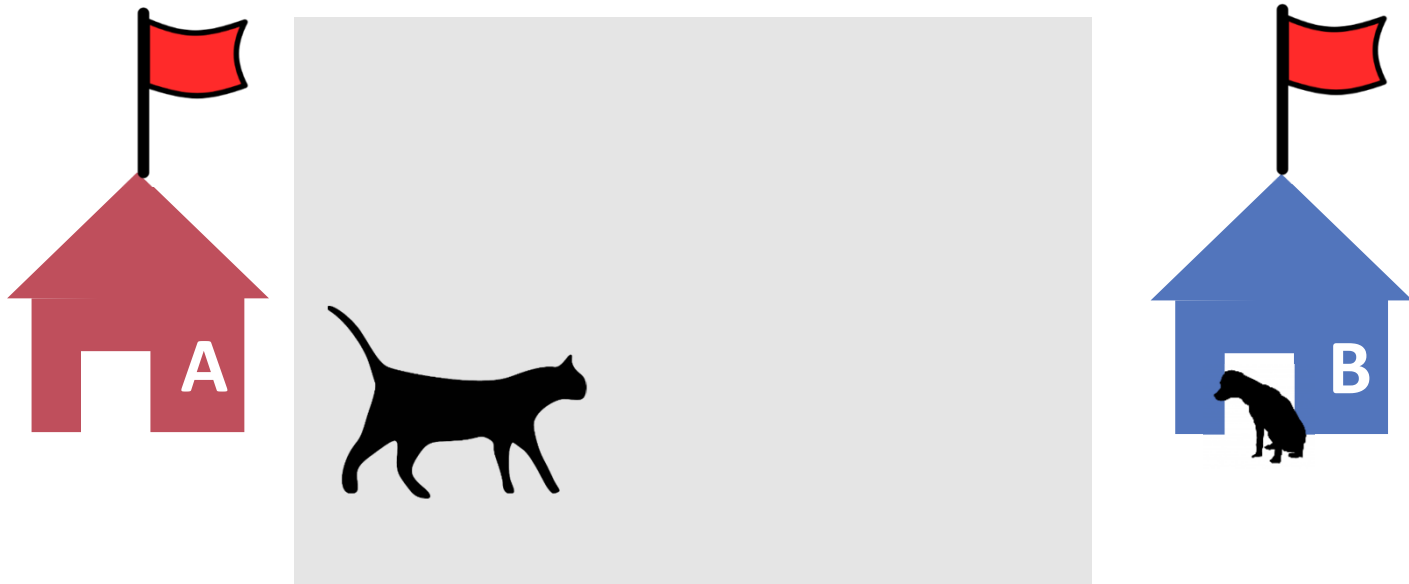


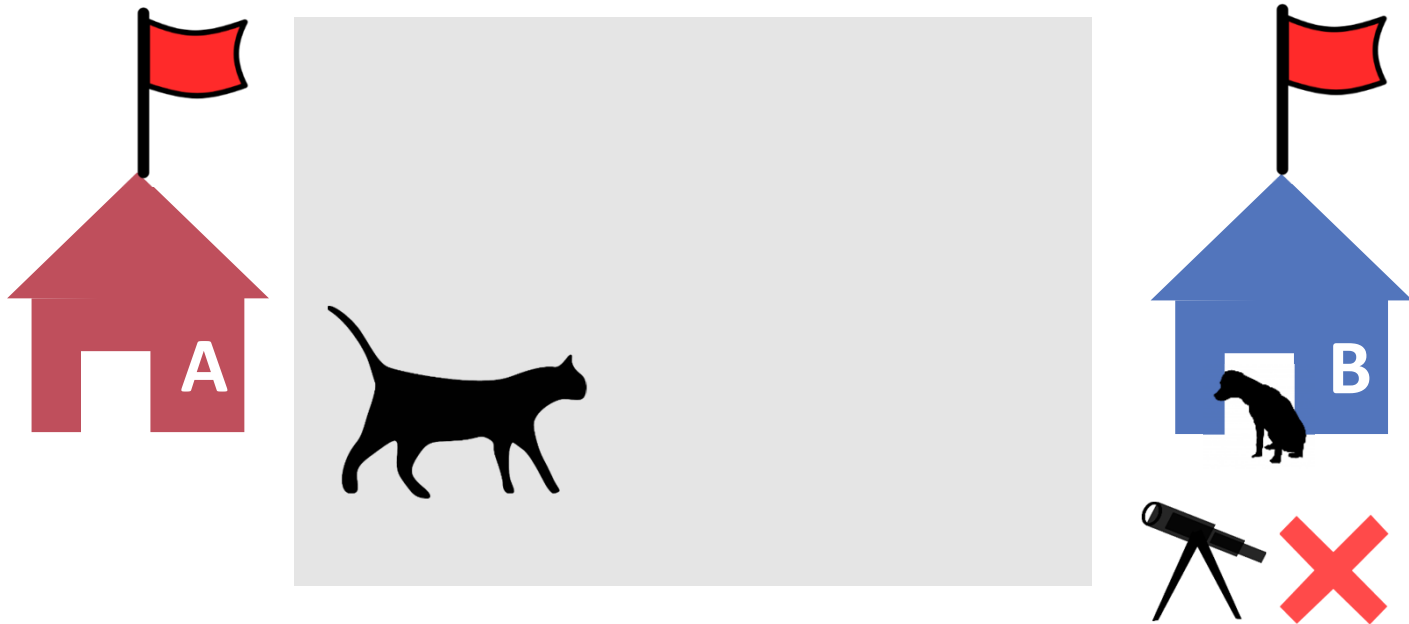




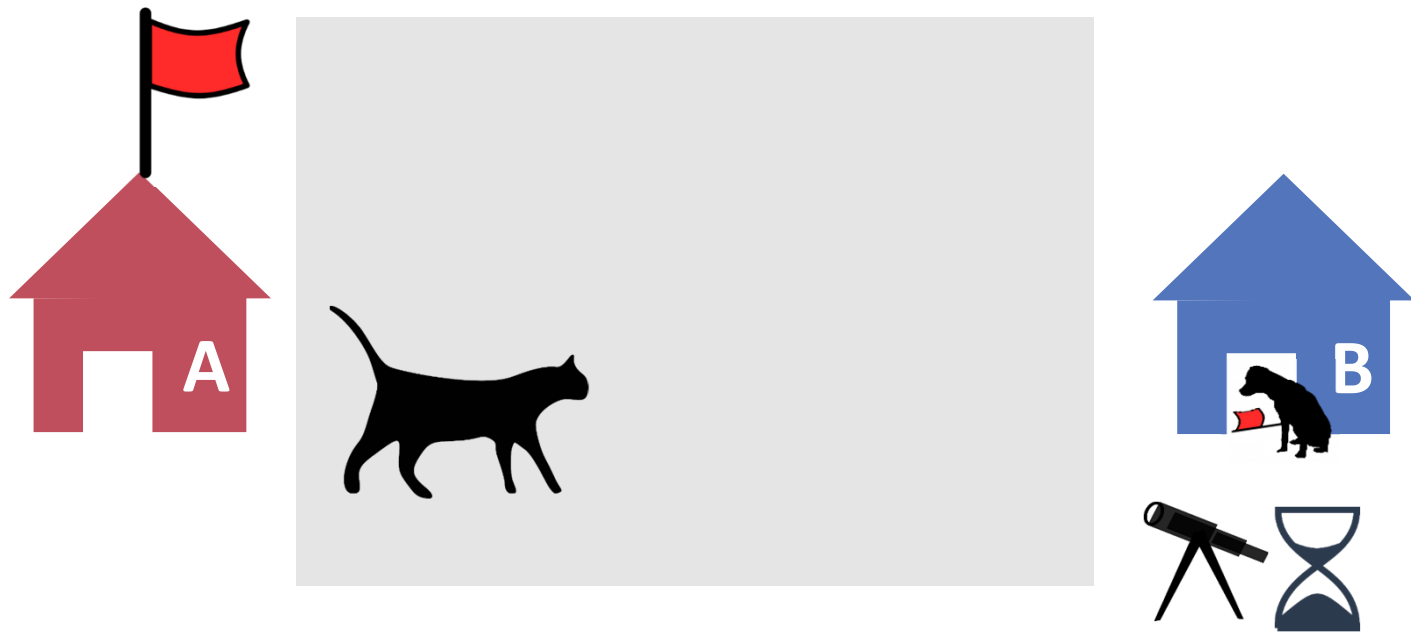








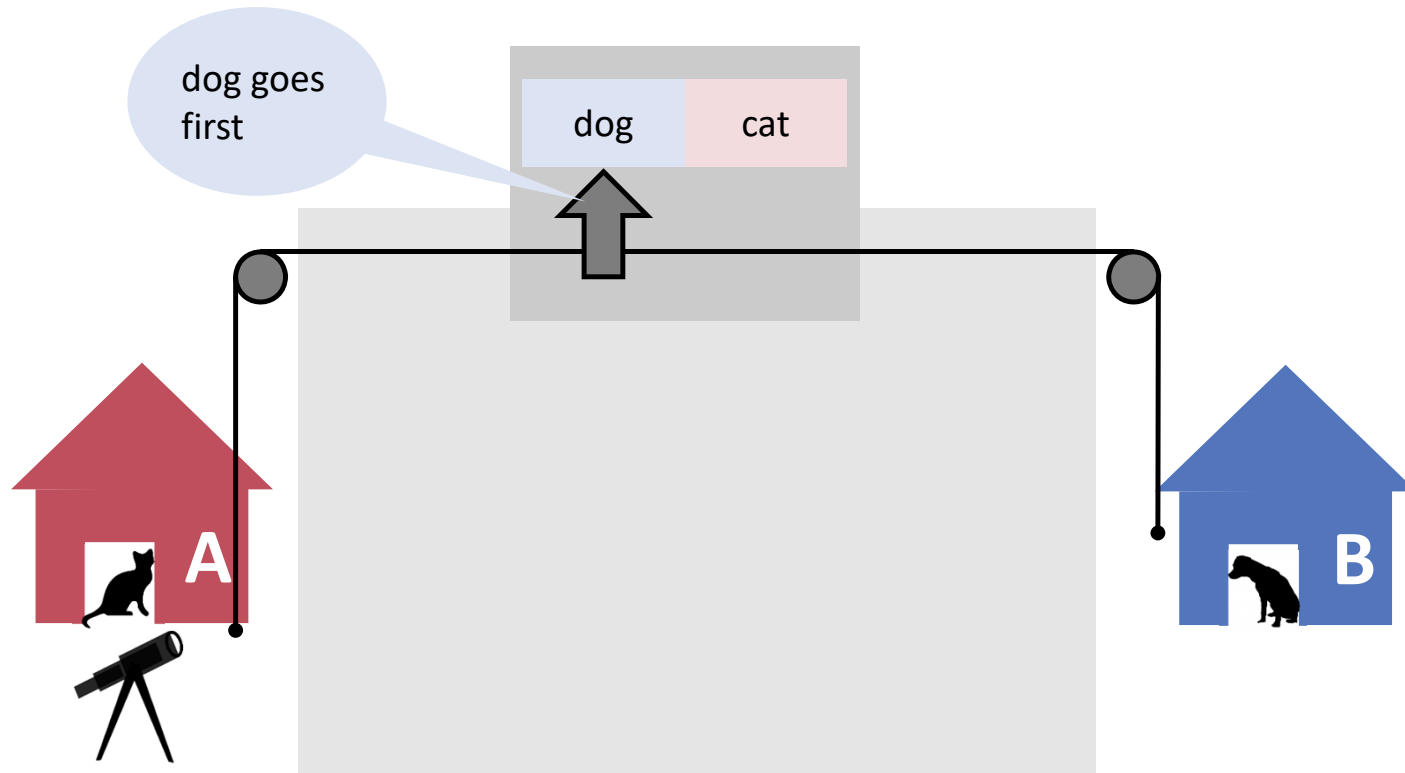
Access Protocol 2.2 is provably correct



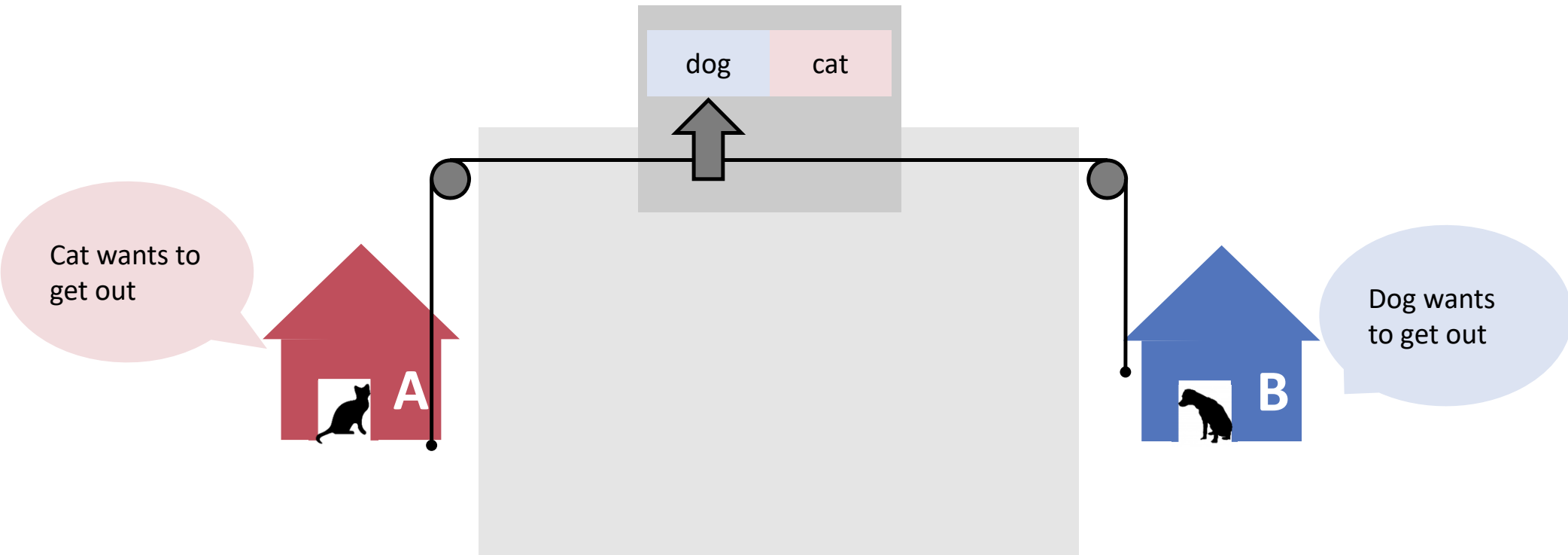
Minor (?) Problems: Livelock, Starvation



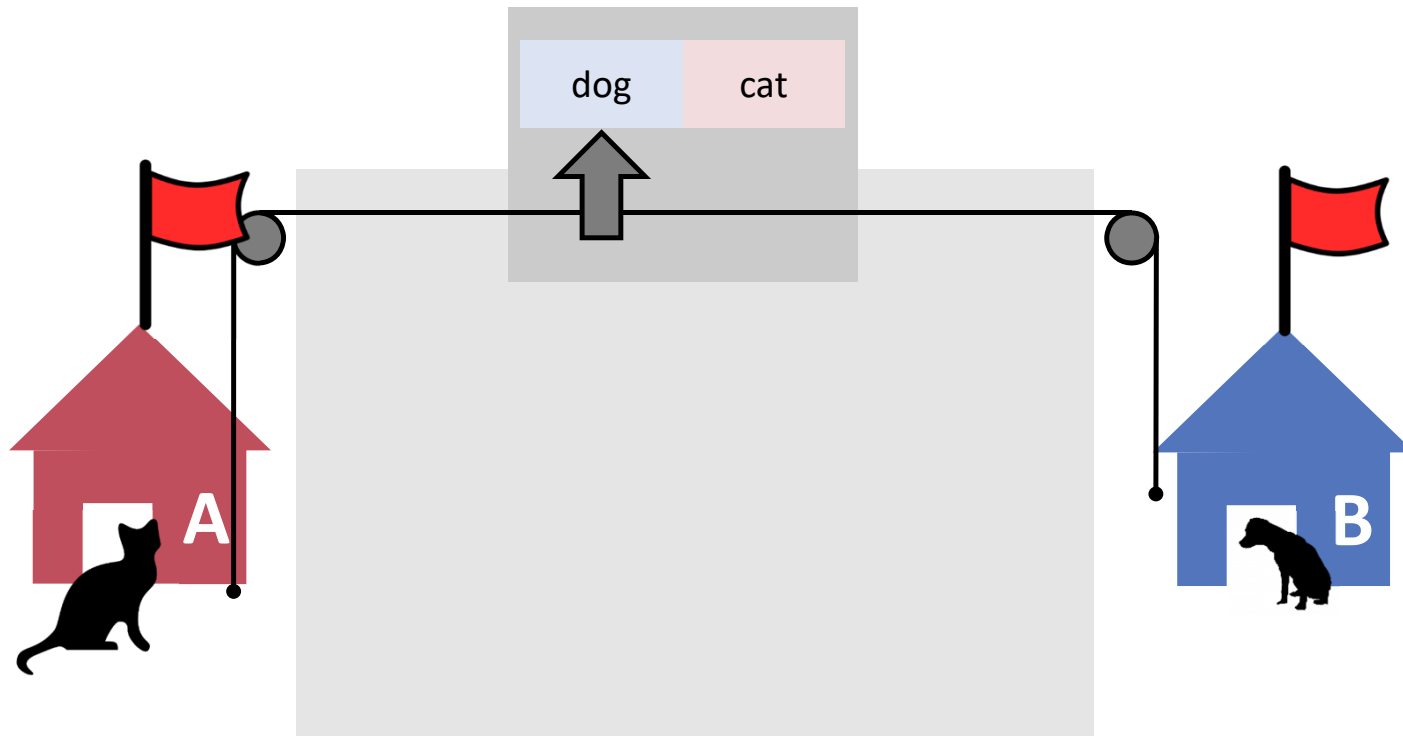
Final Solution



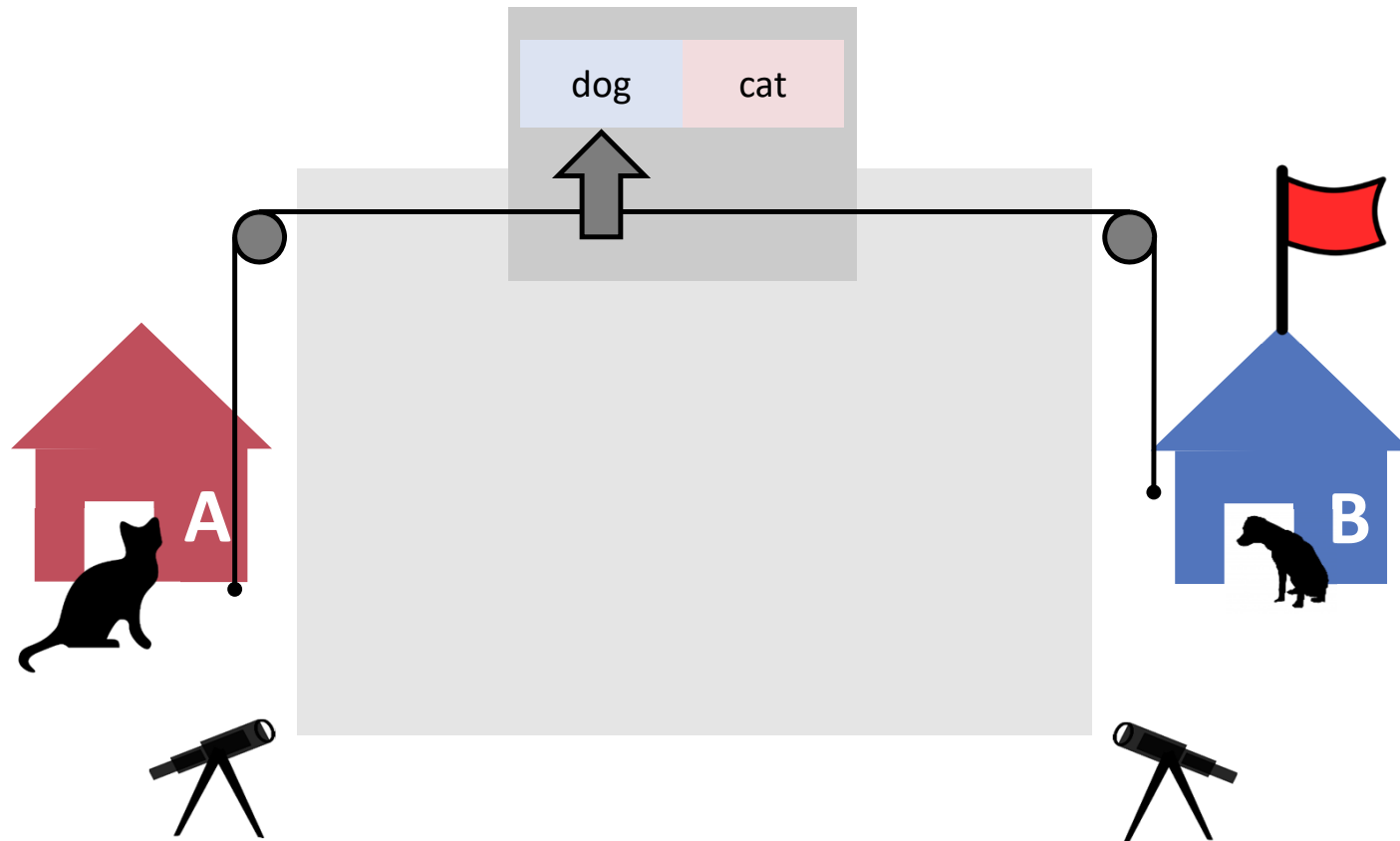
Final Solution



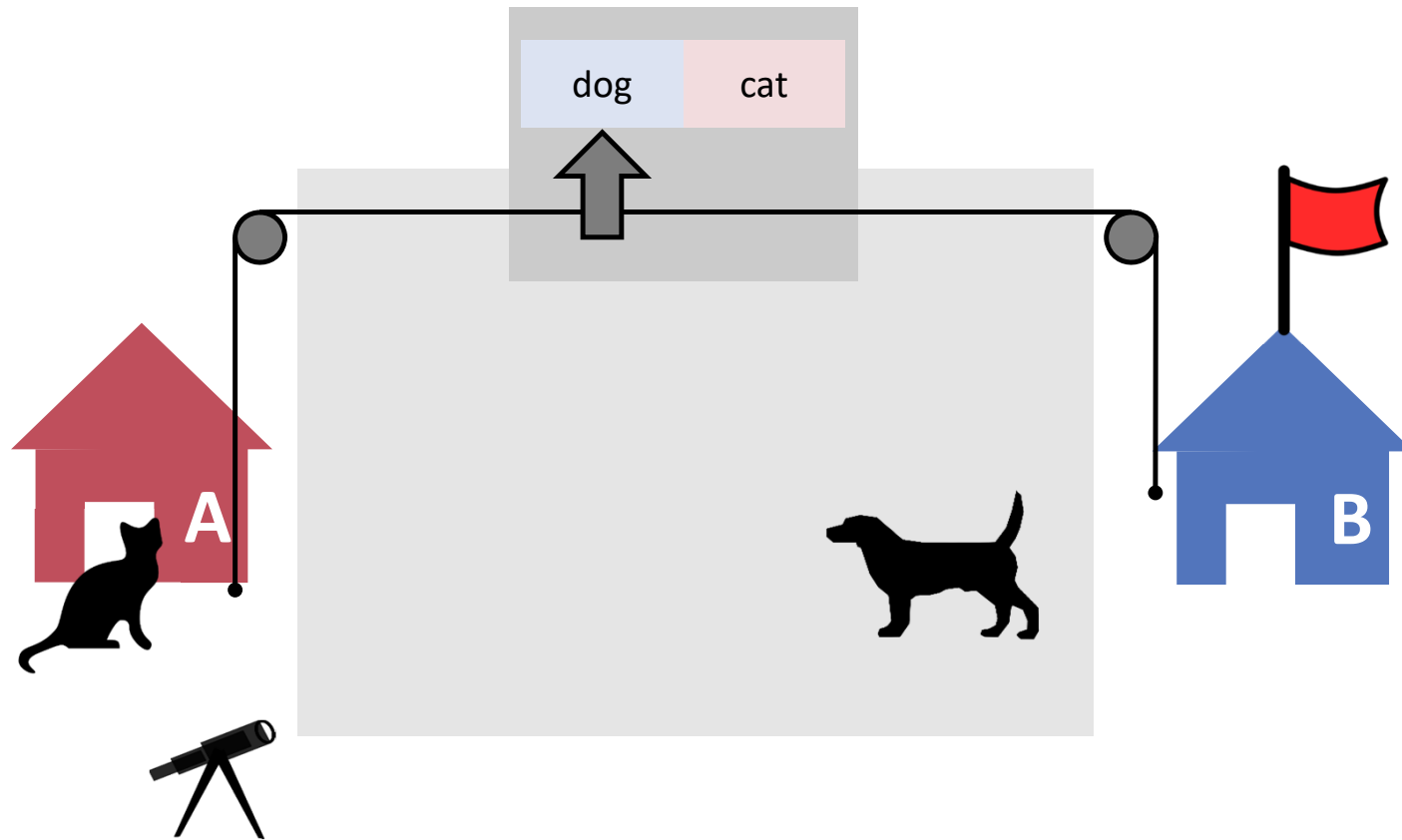
Final Solution



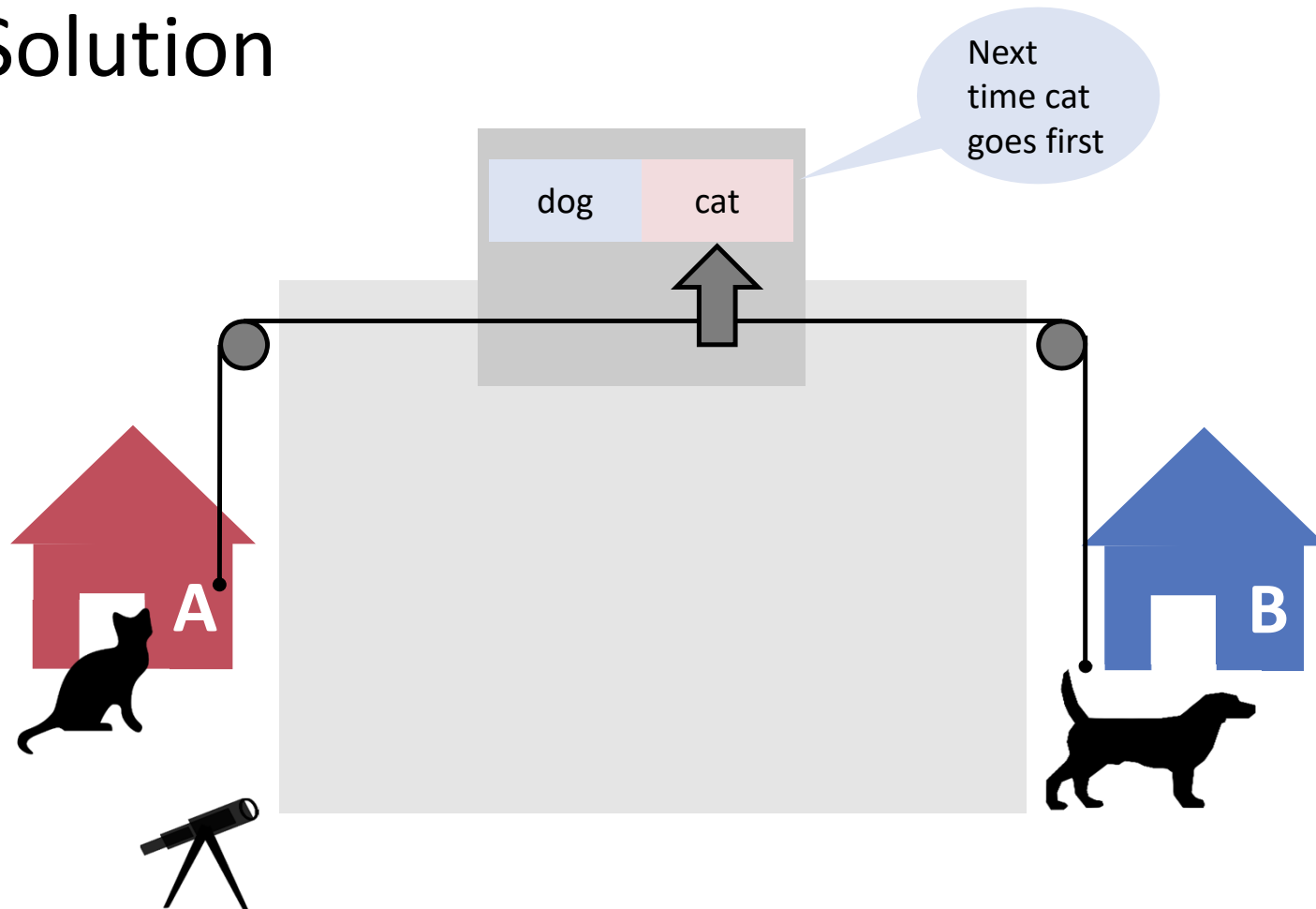
Final Solution



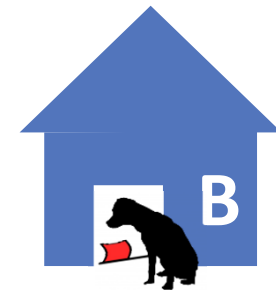
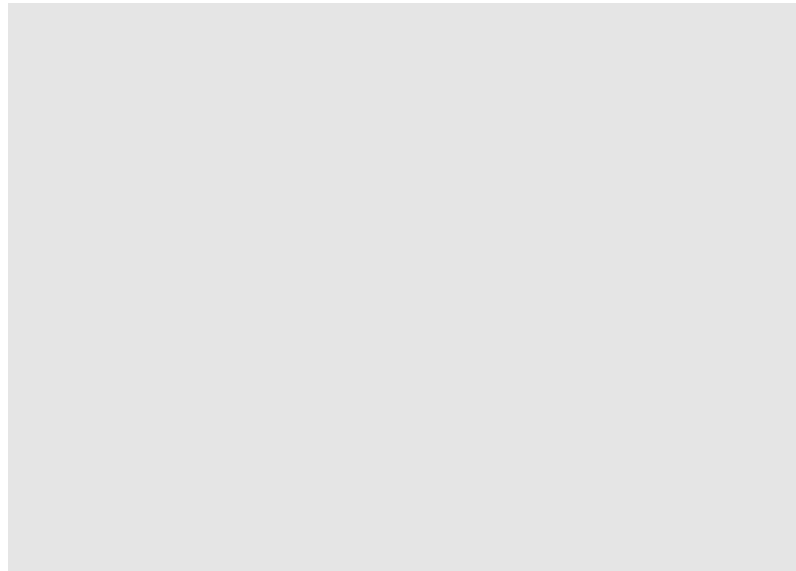
Final Solution



Final Solution



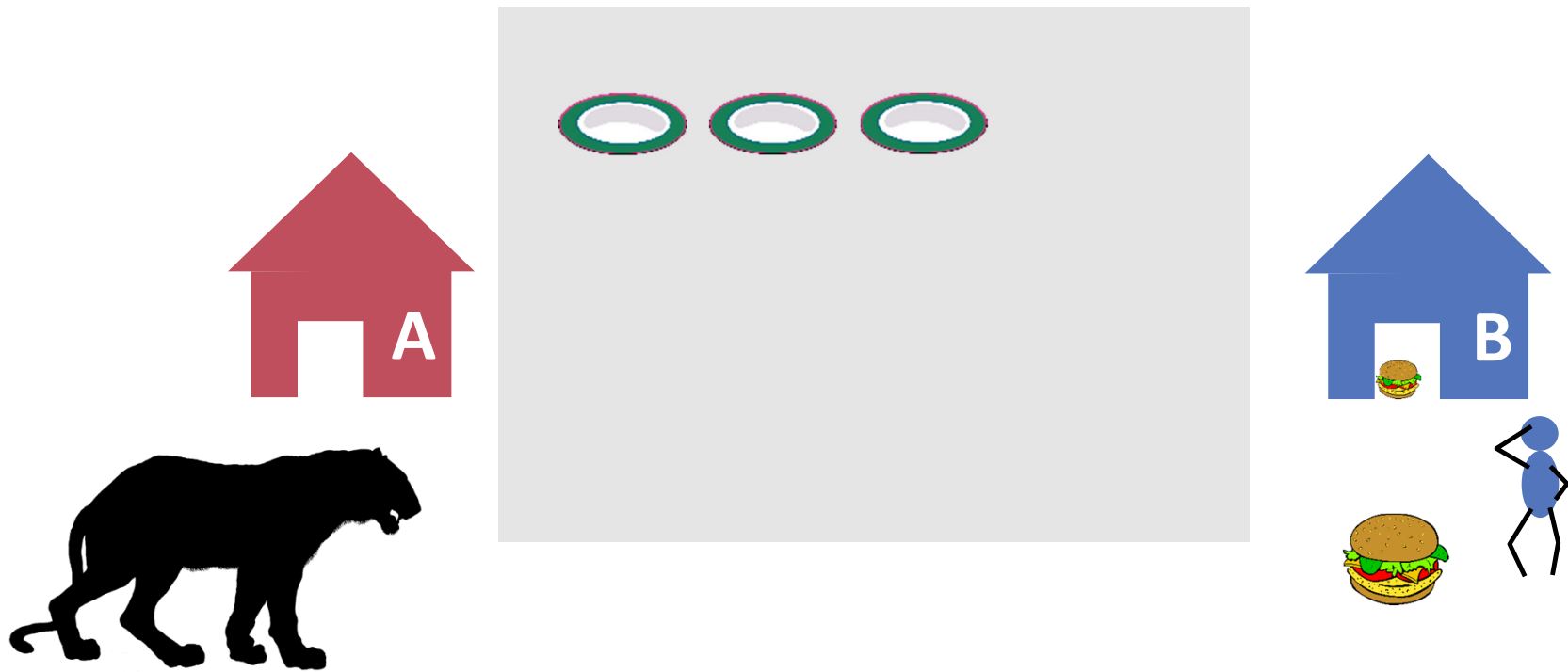
Still: General Problem of Waiting ...



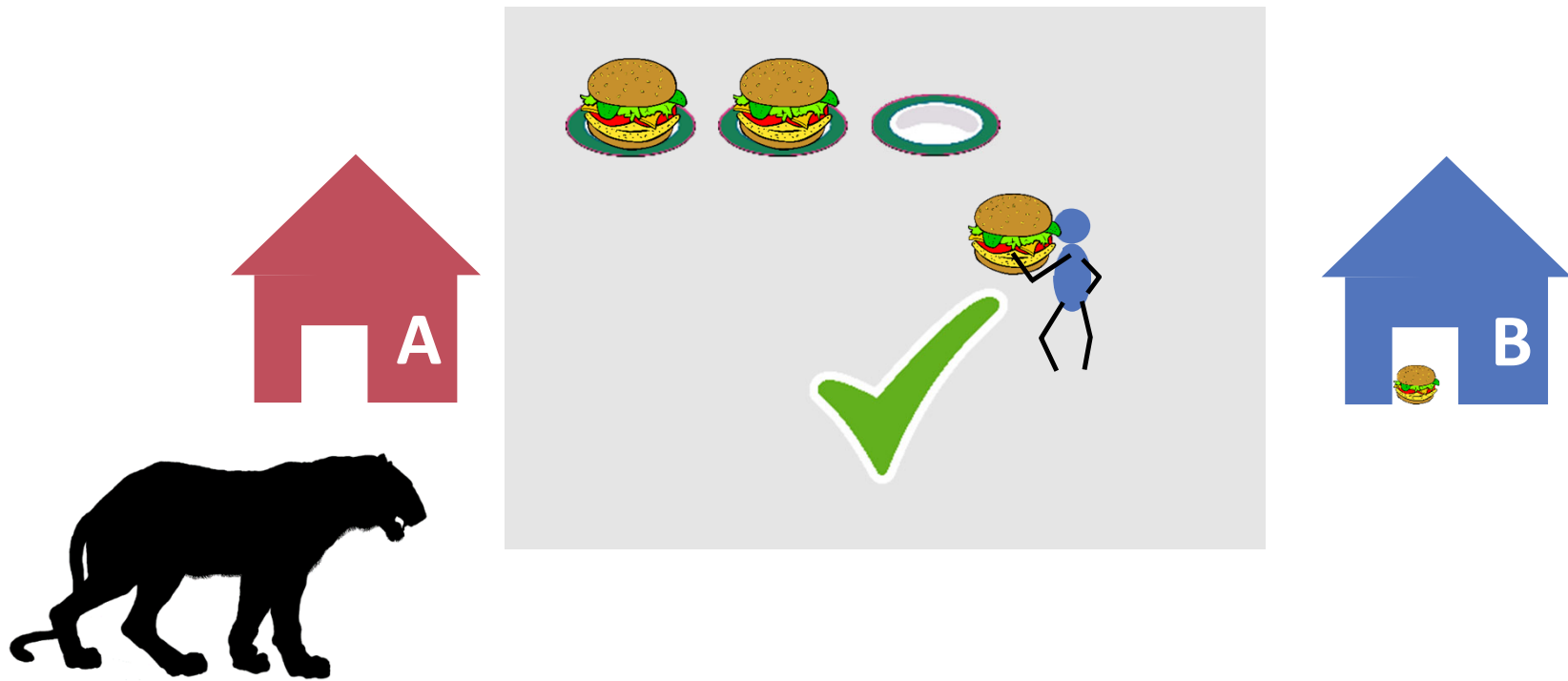
Three stories

2. PRODUCER-CONSUMER

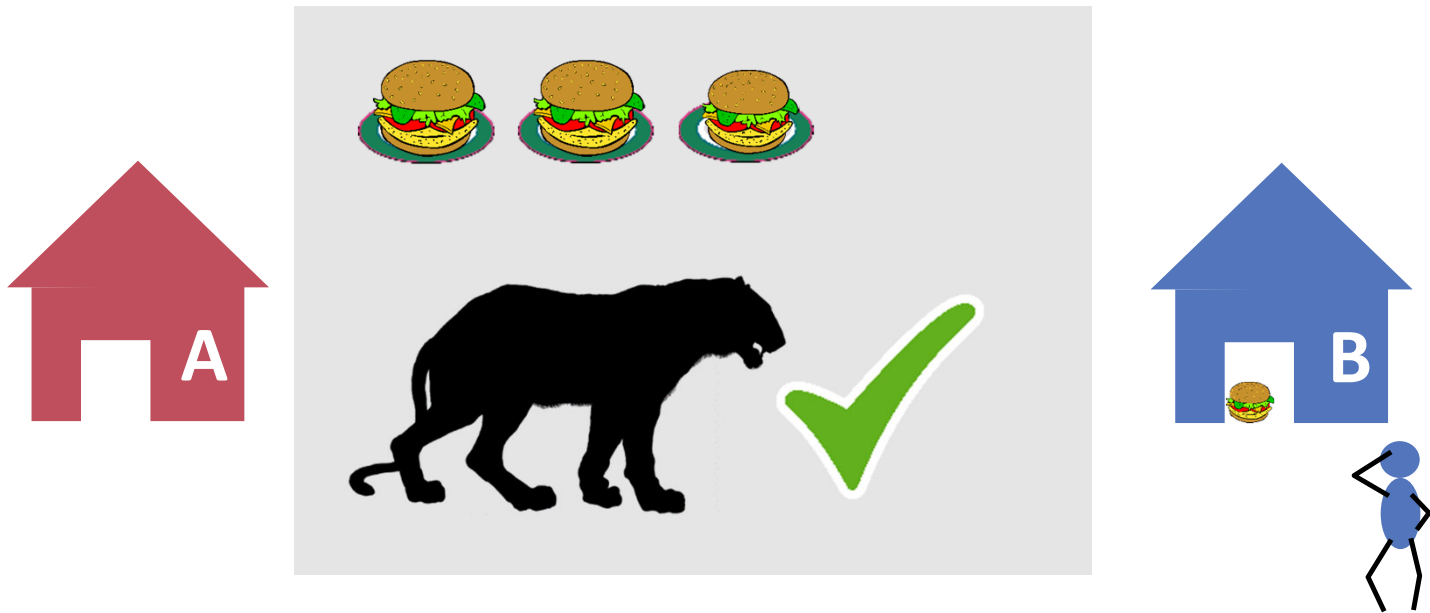
Producer-Consumer

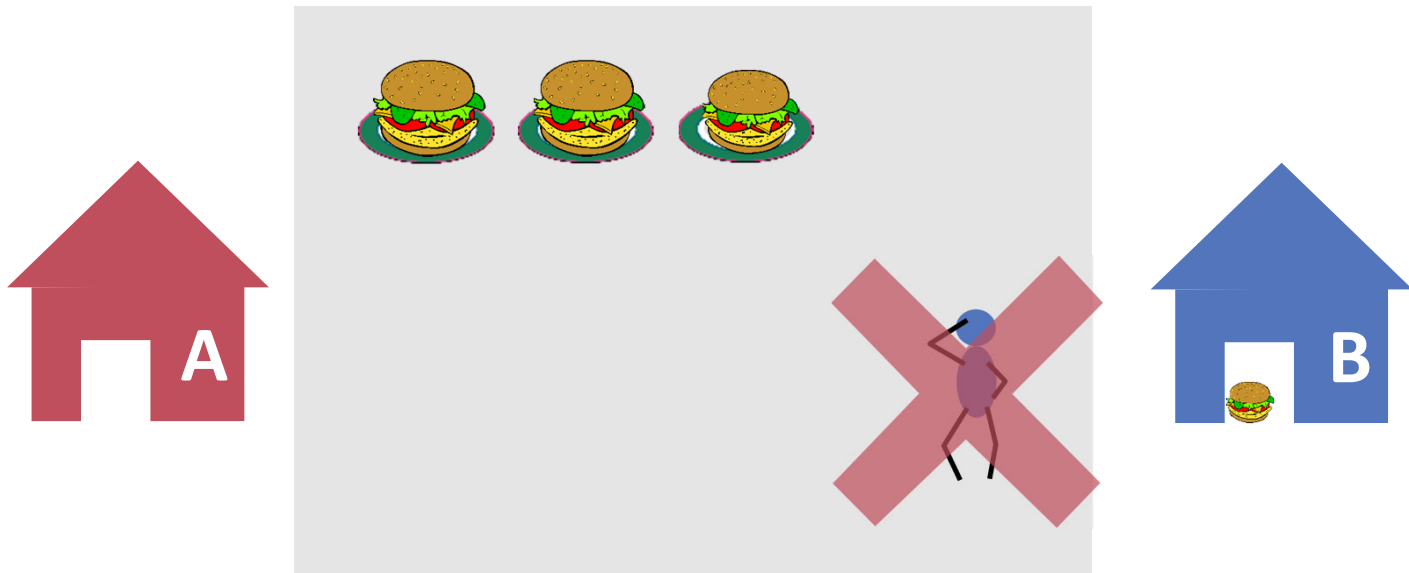


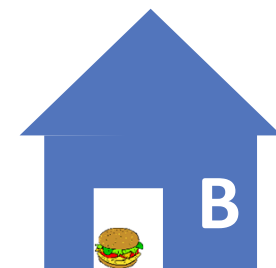
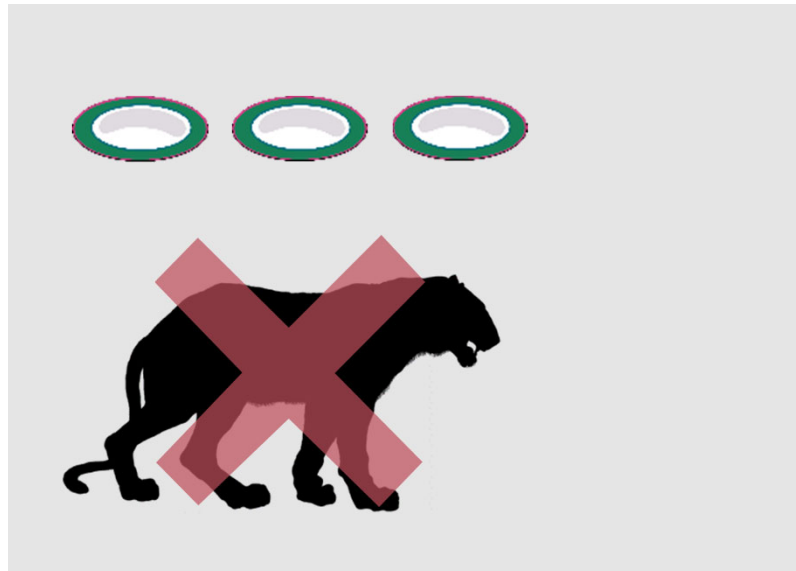
Producer-Consumer



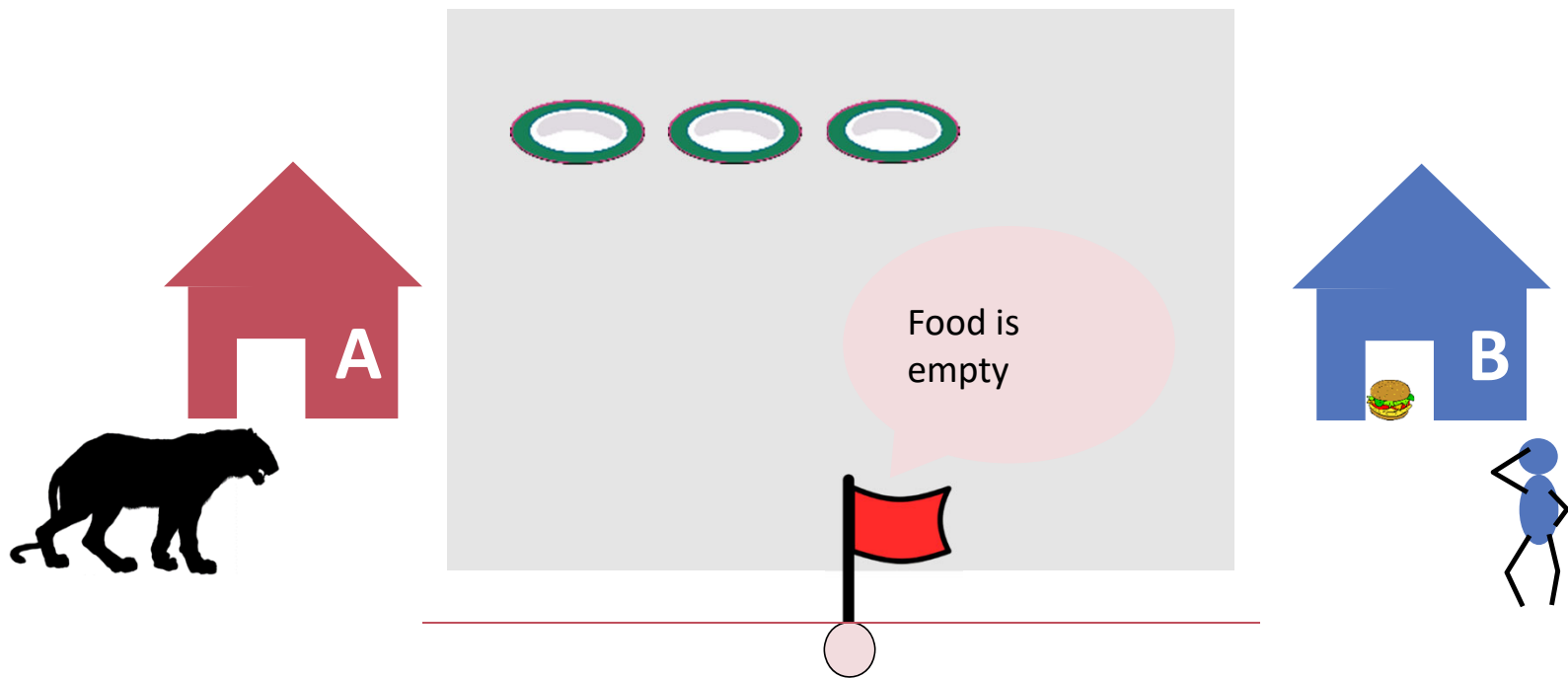
Rules



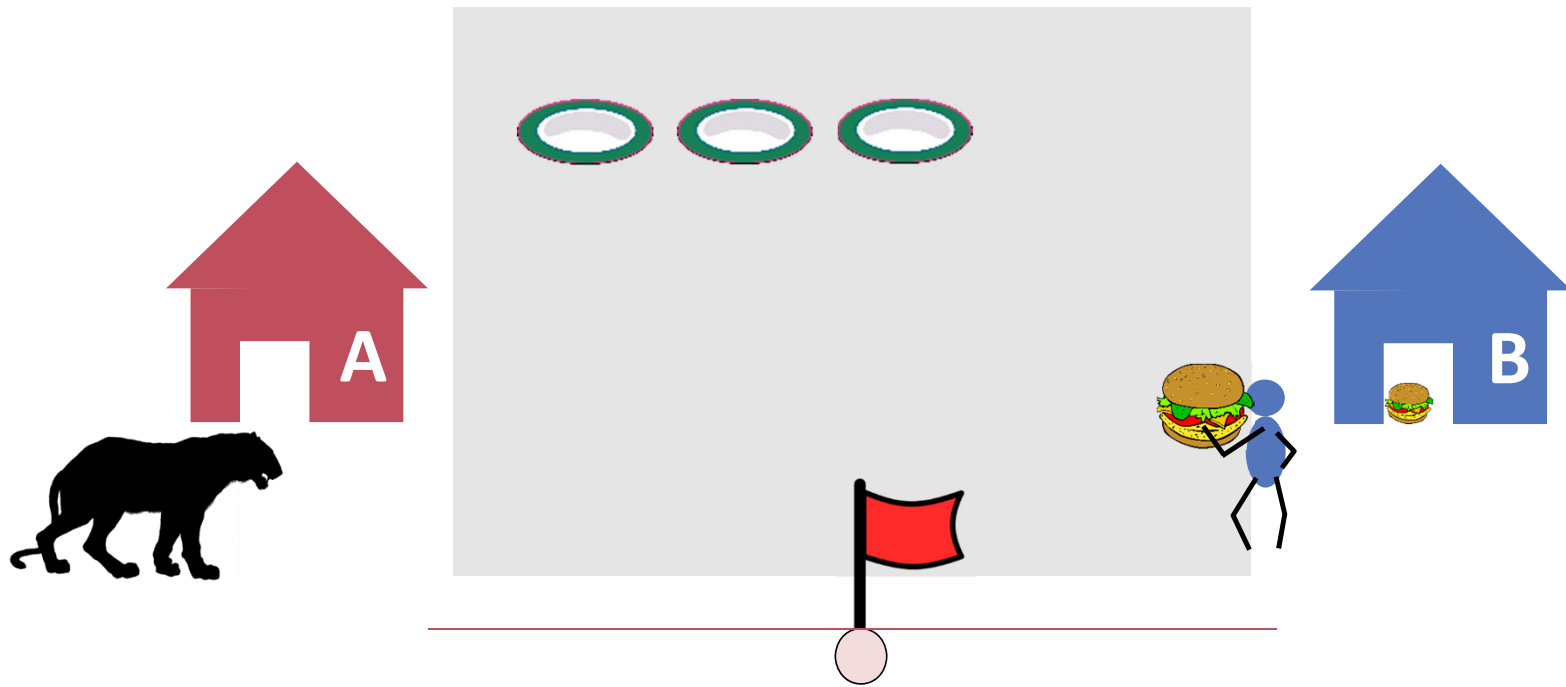


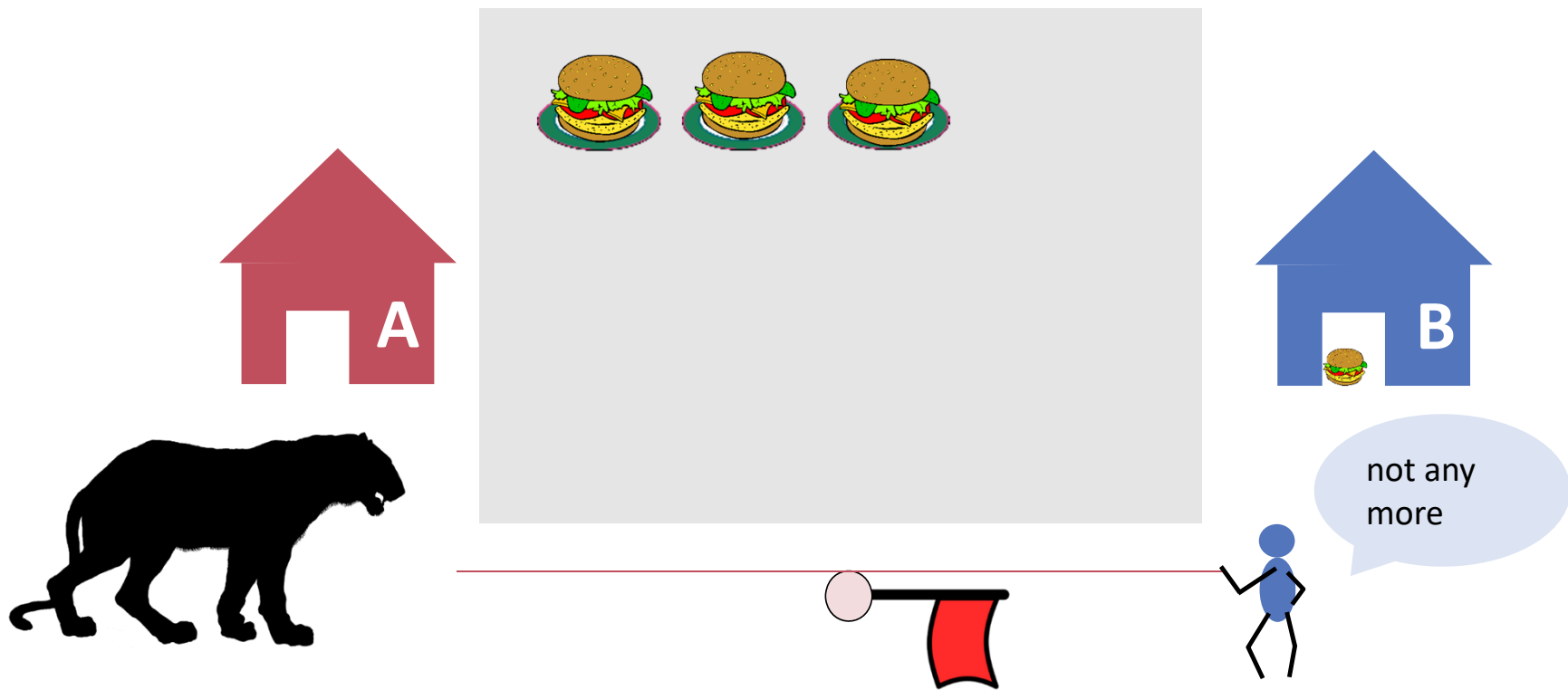


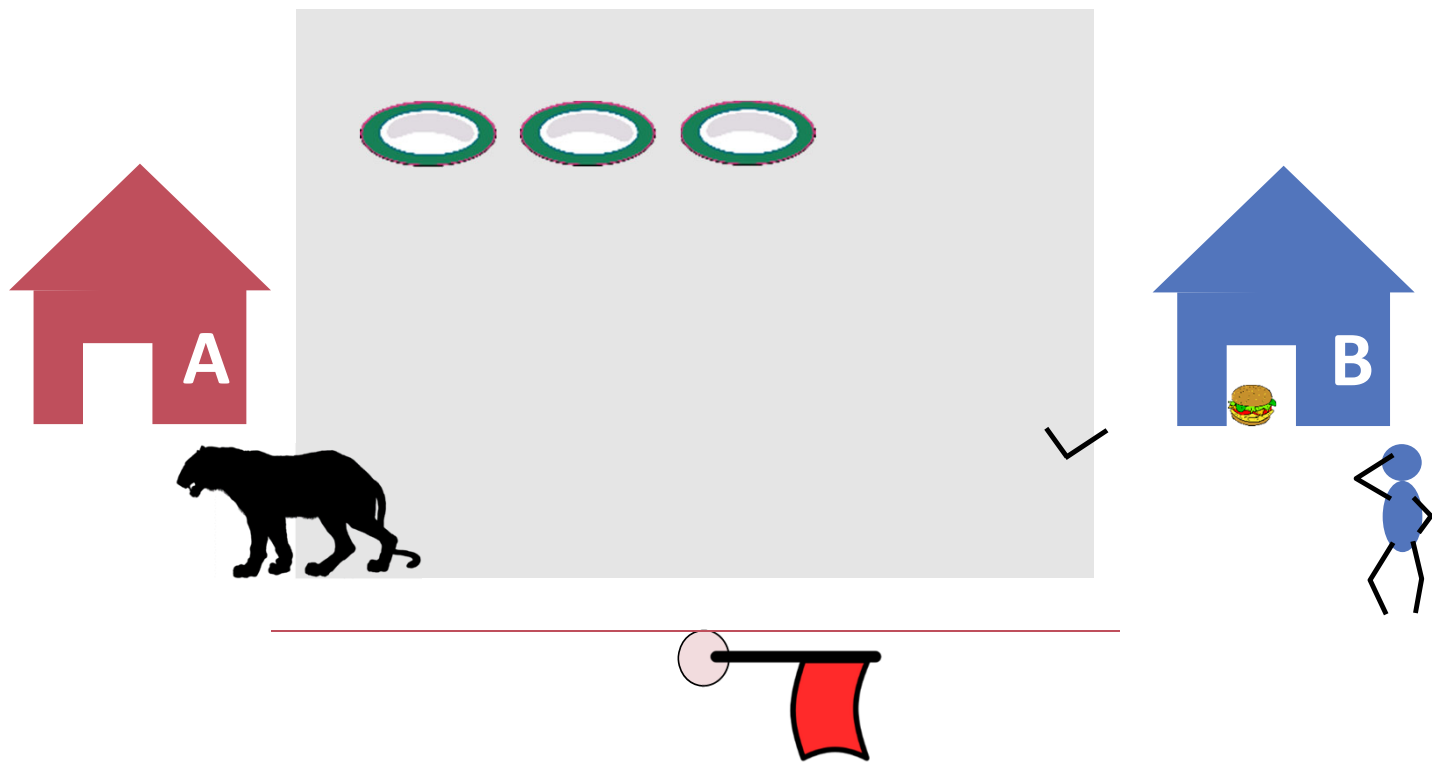
Communication

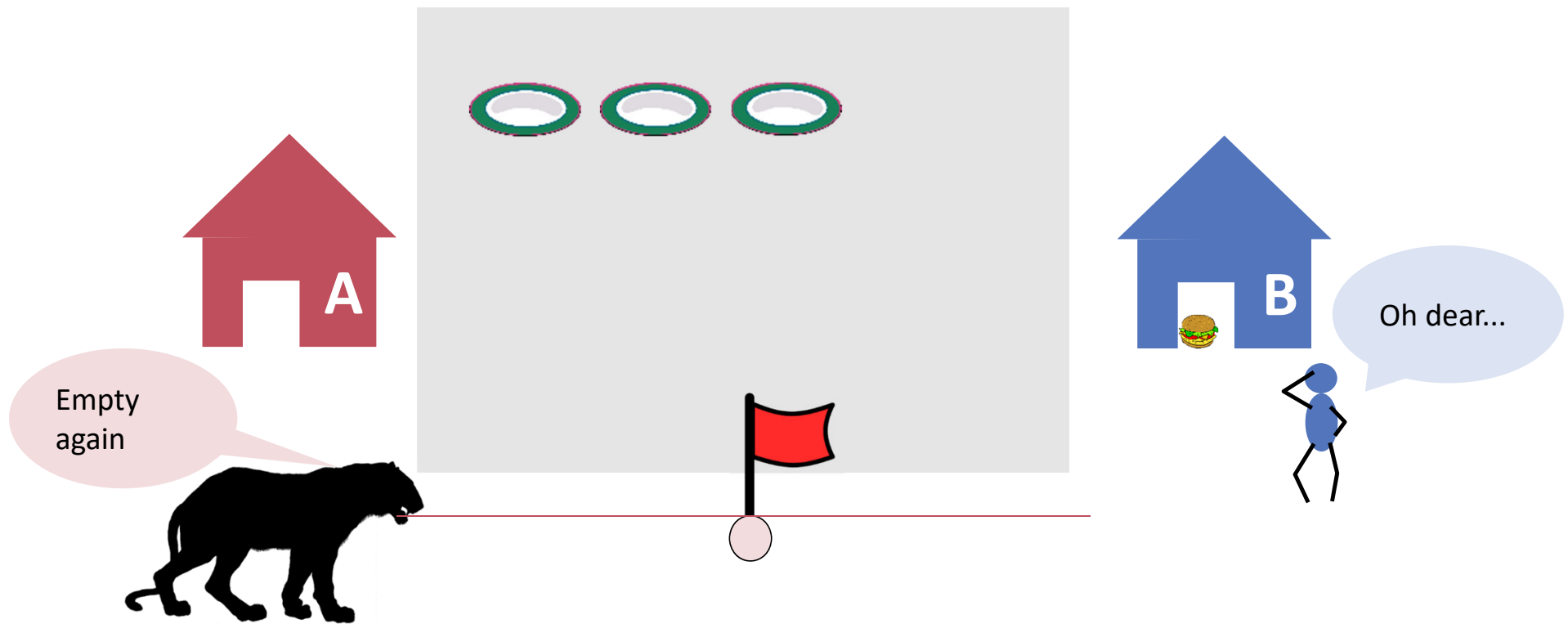


Protocol



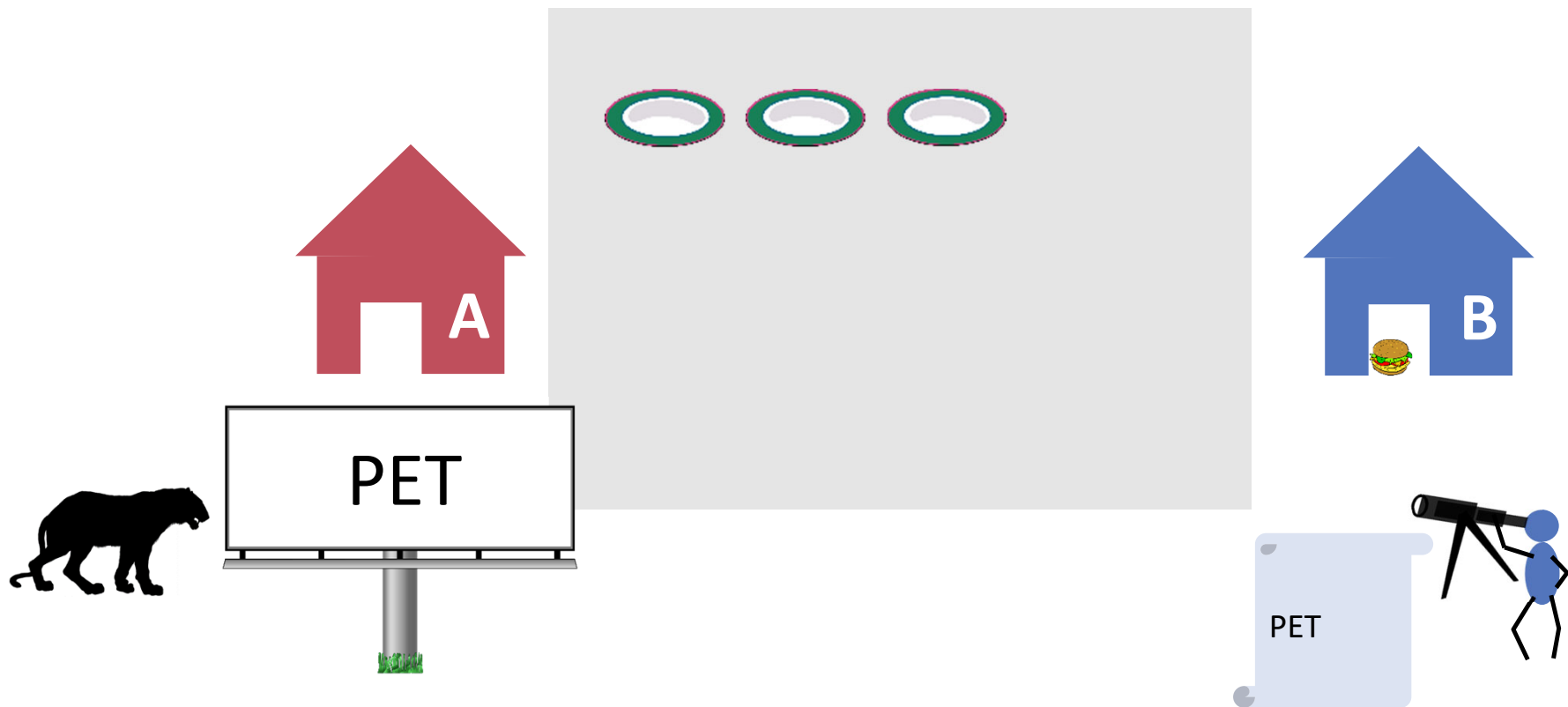


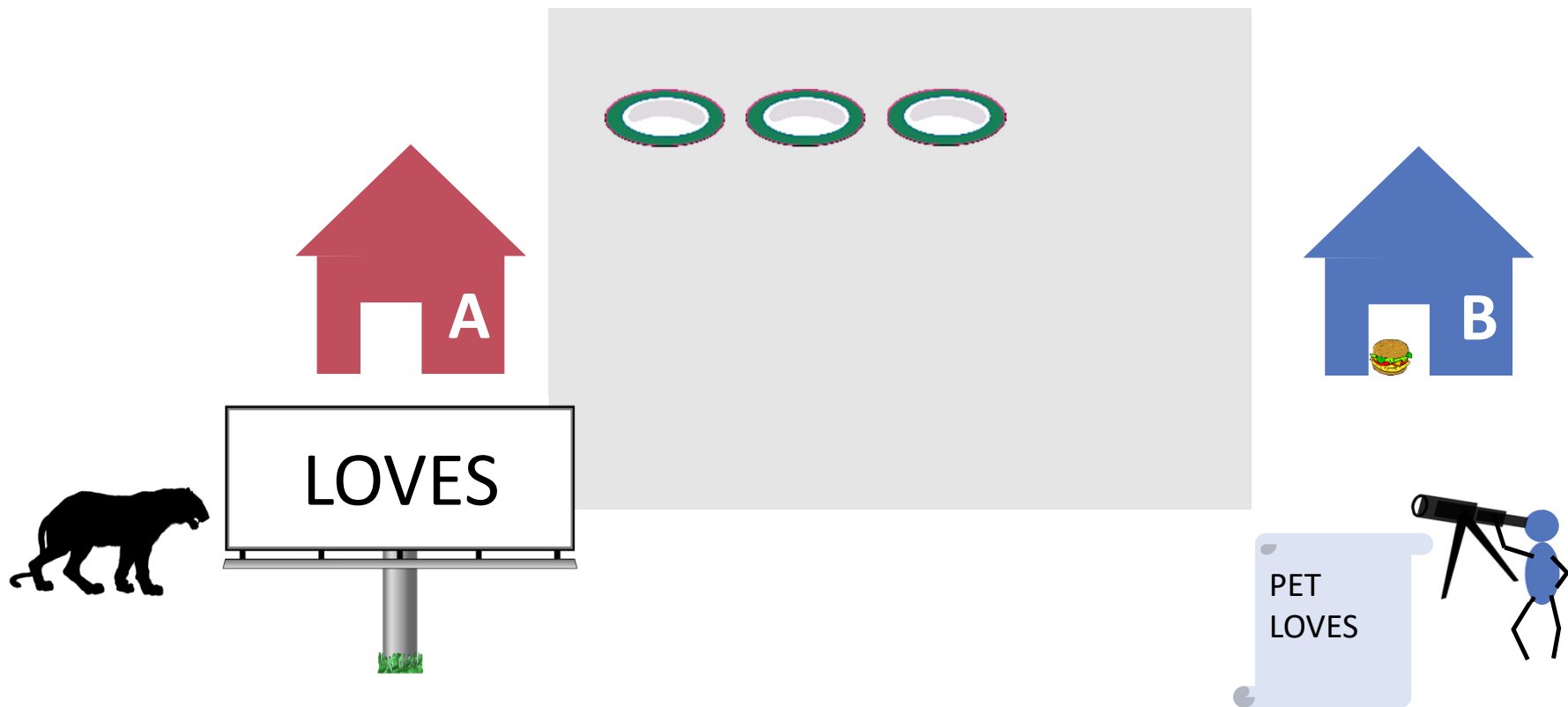


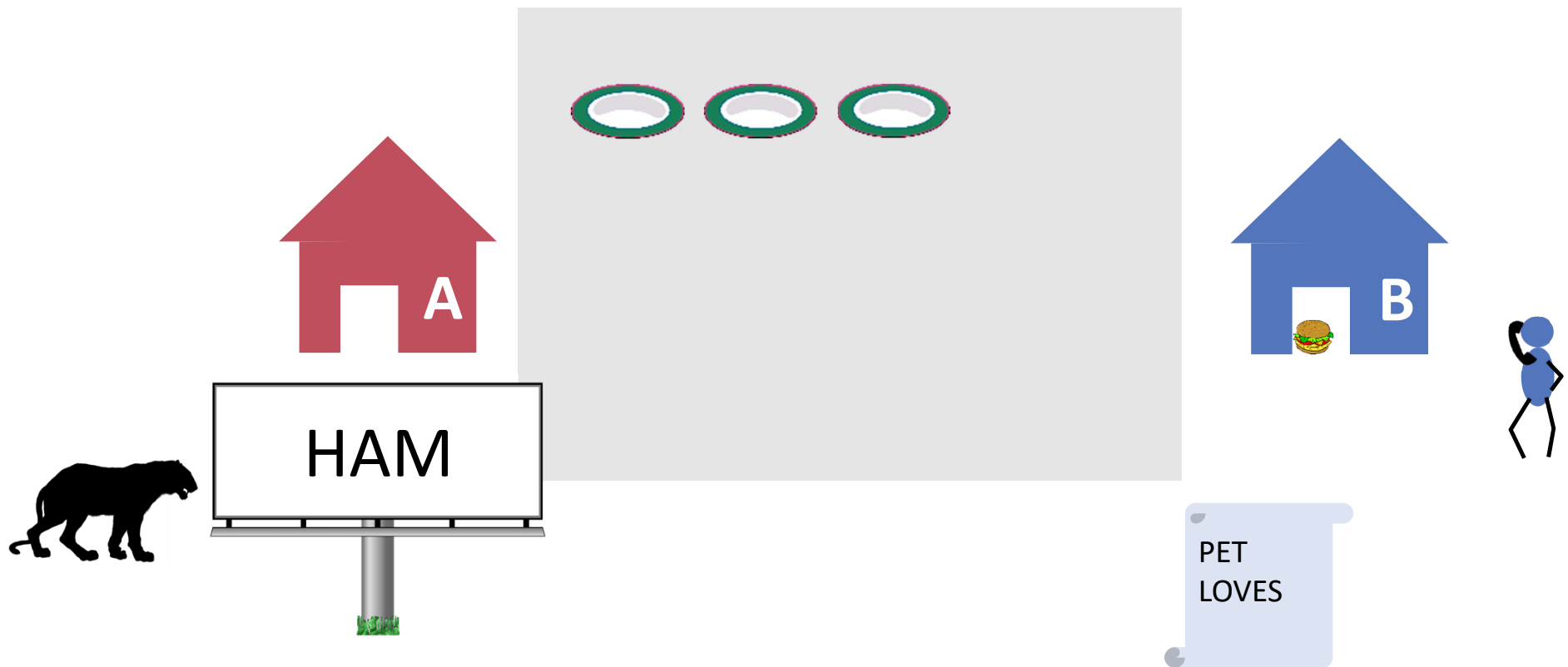


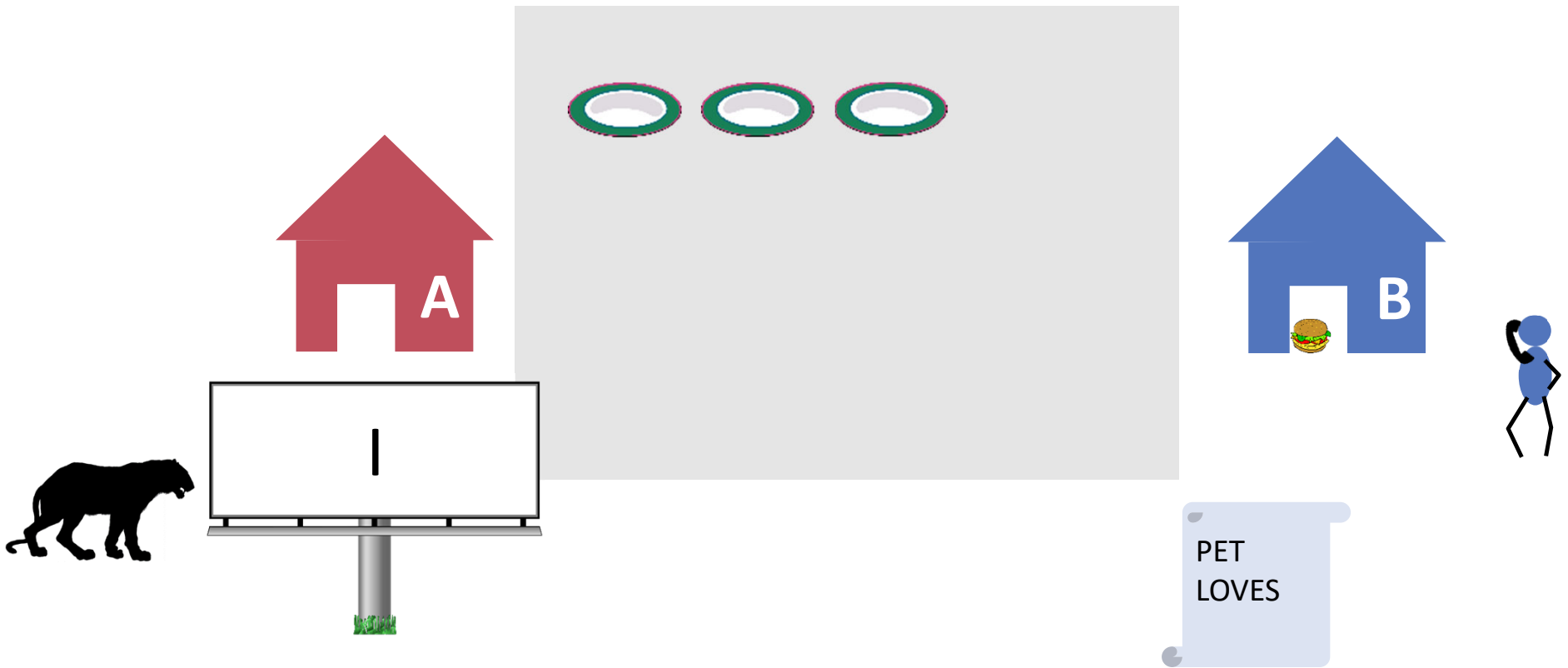
Three stories

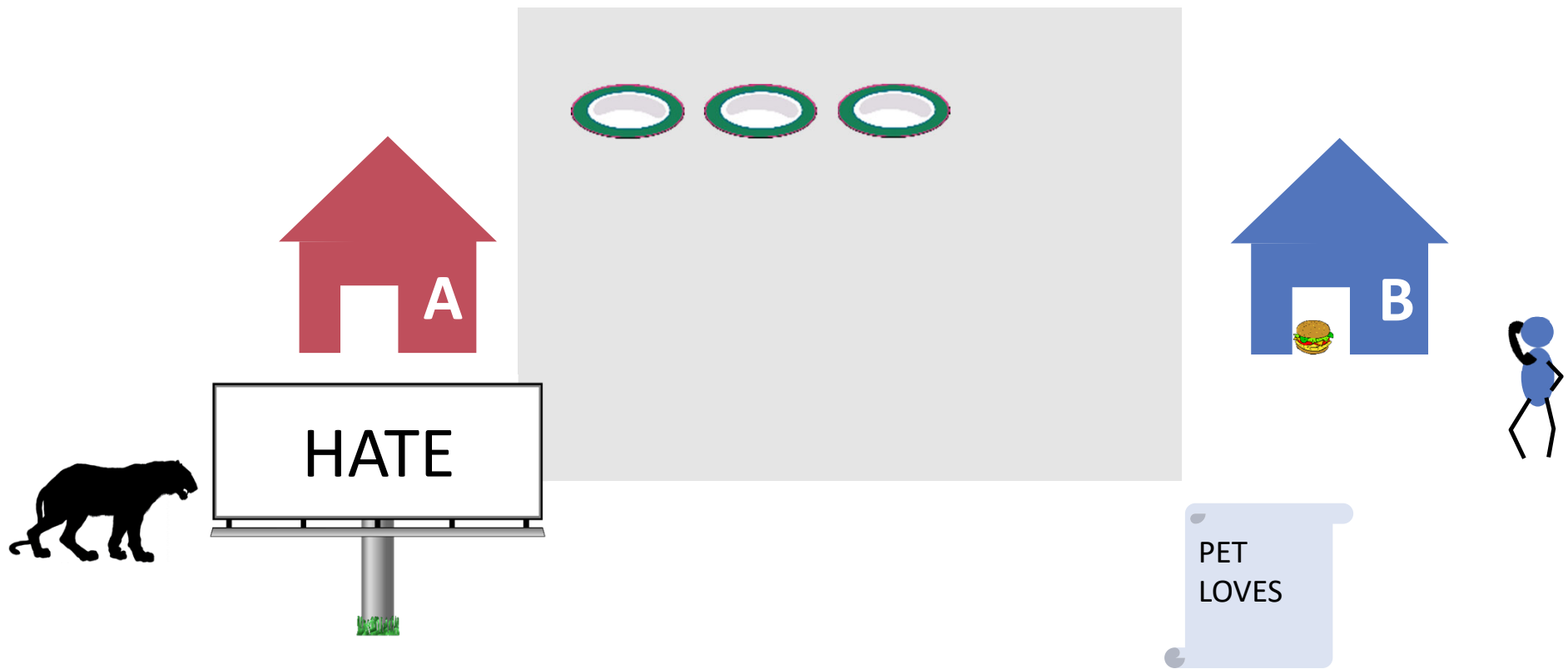
3. READERS-WRITERS

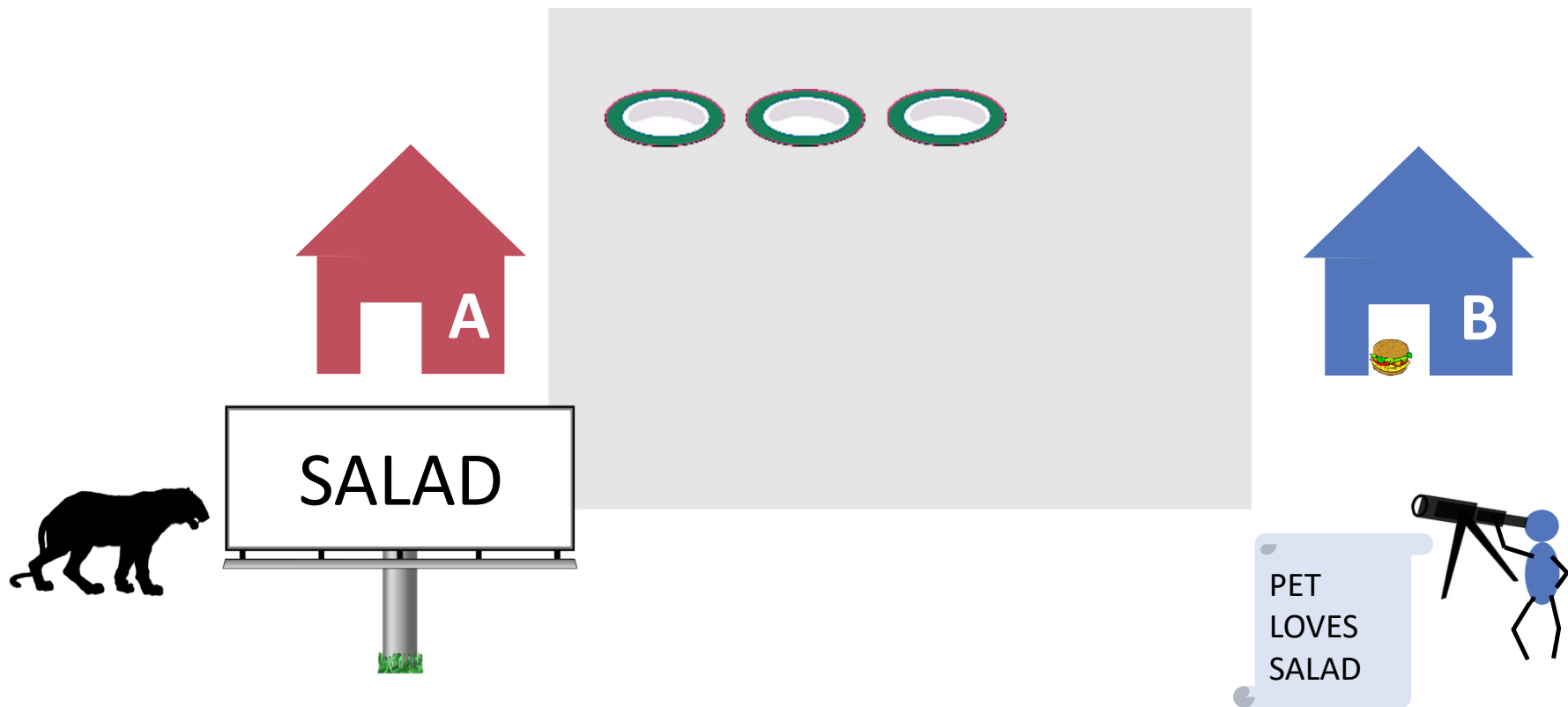












The bad news

- Reality of parallel computing is **much more complicated** than this.
- The results of one action, such as the lifting of a flag by one thread, can become visible by other threads delayed or even in different order, making the aforementioned protocols even more tricky.
- Precise reasons will become clear much later in your studies. But we will understand consequences in the lectures later.

The good news

- On parallel hardware we will find an interesting tool to deal with low level concurrency issues.
- There is sufficient **abstraction** in the **programming models** of different programming languages.
- Later on, we will not really have to deal with such low level concurrency issues. But we should have understood them once.

Language Landscape

C, C++

Java

Python, Ruby, Perl

Scala, Clojure, Groovy

Erlang, Go, Rust

Haskell, OCaml

JavaScript

...



Why use Java?

Is ubiquitous (see oracle installer)

- Many (very useful) libraries
- Excellent online tutorials & books

Parallelism is well supported

- In the language and via frameworks

Interoperable with modern JVM languages

- E.g., Akka framework

Yet, not perfect

- Tends to be verbose, lots of boilerplate code



Concepts and Practice

Our goal is twofold:

- Learn how to write parallel programs in practice
 - Using Java for the most part
 - And showing how it works in C
- Understand the underlying fundamental concepts
 - Generic concepts outlive specific tools
 - There are other approaches than Java's

You are Encouraged to:

- Ask questions:
 - helps us keep a good pace
 - helps you understand the material
 - let's make the course interactive
 - class or via e-mail
- Use the web to find additional information
 - Javadocs
 - Stack Overflow
- Write Code & Experiment

*If there is a problem,
let us know as **early**
as **possible!***

What are Exercises for?

Learning tool

Seeing a correct solution is not enough

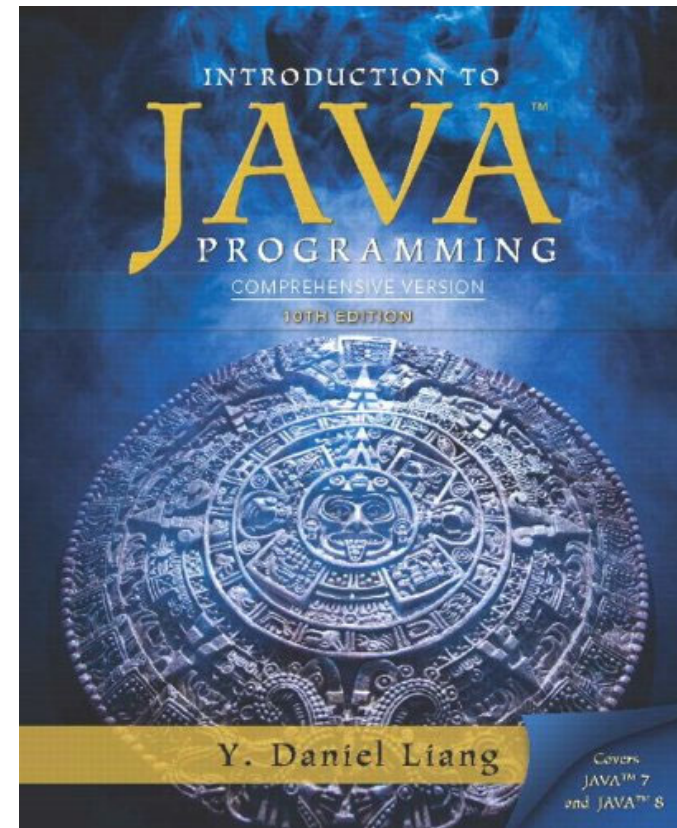
You should try to solve the problem yourselves

Hence, exercise sessions are

- for guiding you to solve the problem
- not for spoon-feeding you solutions

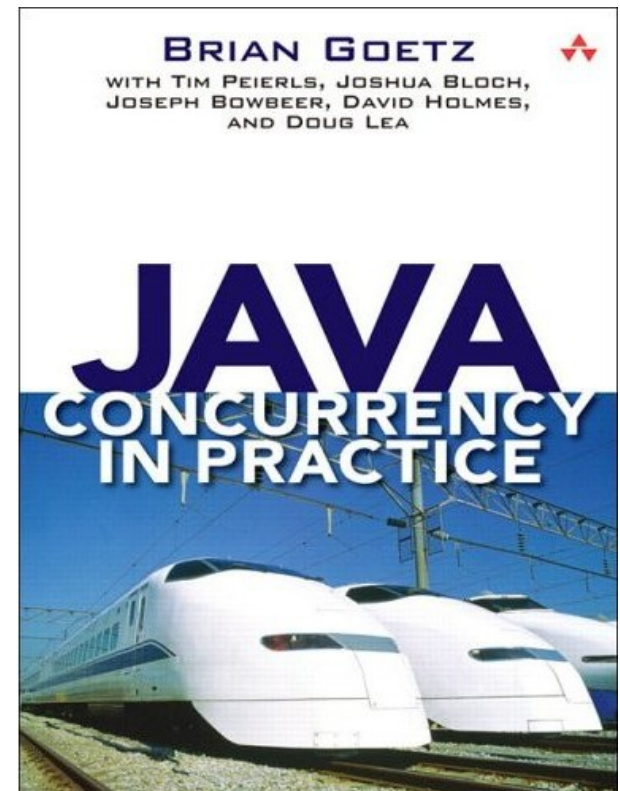
Introduction to Java Programming

- Introduction to Java Programming, 2014.
- Daniel Liang.
- ISBN-13: 9780133813463
- Chapters 1-13 (with some omissions)
- Week 1-3



Java Concurrency in Practice

- Java Concurrency in Practice, 2006.
- Brian Goetz, Tim Peierls, Joshua Bloch, Joseph Bowbeer, David Holmes, Doug Lea.
- ISBN-13: 9780321349606
- Week 4-9



Theory and beyond

- Fundamental treatment of concurrency
- In particular the "Principles" part is unique
- Not easy
- In this course
 - Theory of concurrency
 - Behind locks
 - Lock-free programming

