Parallel Programming

Introduction & Course Overview

SS 2021

Dr. Barbara Solenthaler

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

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecturers

Dr. Barbara Solenthaler
CNB G 106.1
solenthaler@inf.ethz.ch

Teaches Part I
Office hours: per email request

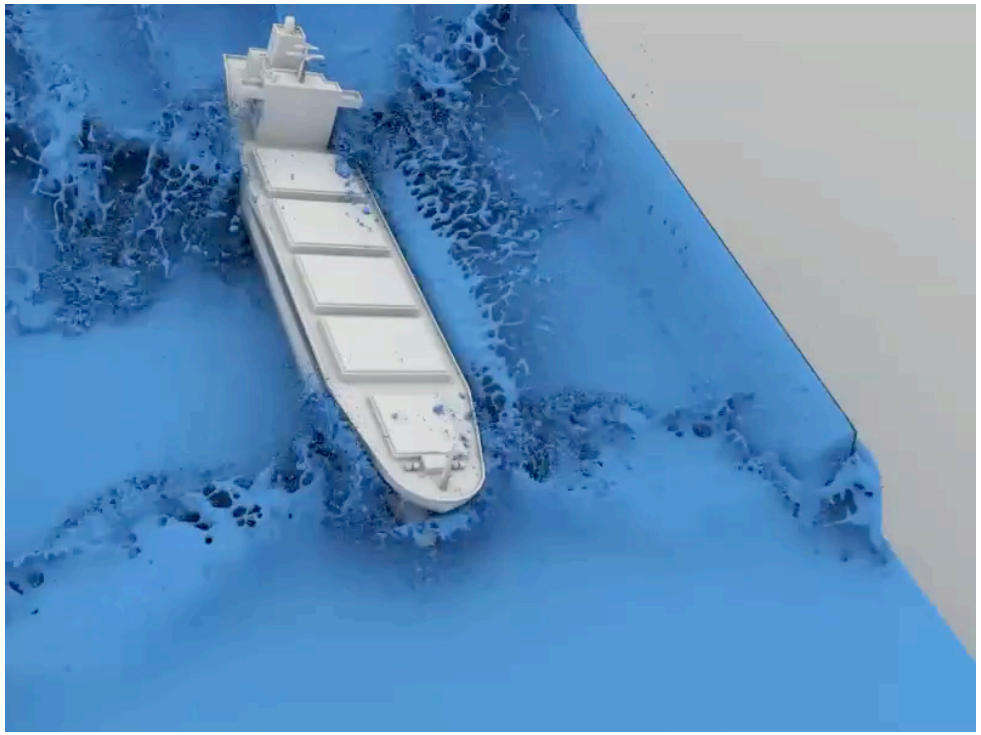
Prof. Torsten Hoefler
CAB F 75
torsten.hoefler@inf.ethz.ch

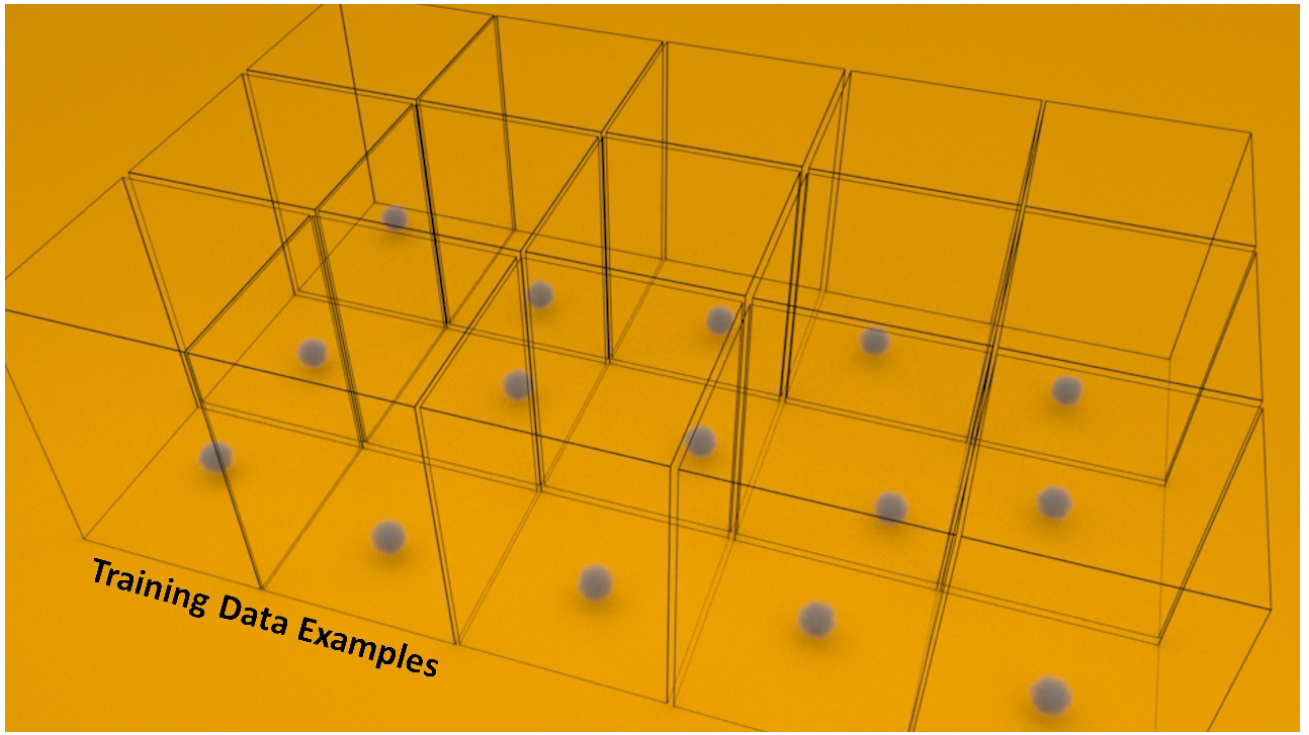
Teaches Part II
Office hours: per email request

https://spcl.inf.ethz.ch/Teaching/2021-pp/

Computer Graphics Lab

- Senior researcher at ETH since 2010 / Head of the simulation & animation research
  https://cgl.ethz.ch / https://people.inf.ethz.ch/~sobarbar/

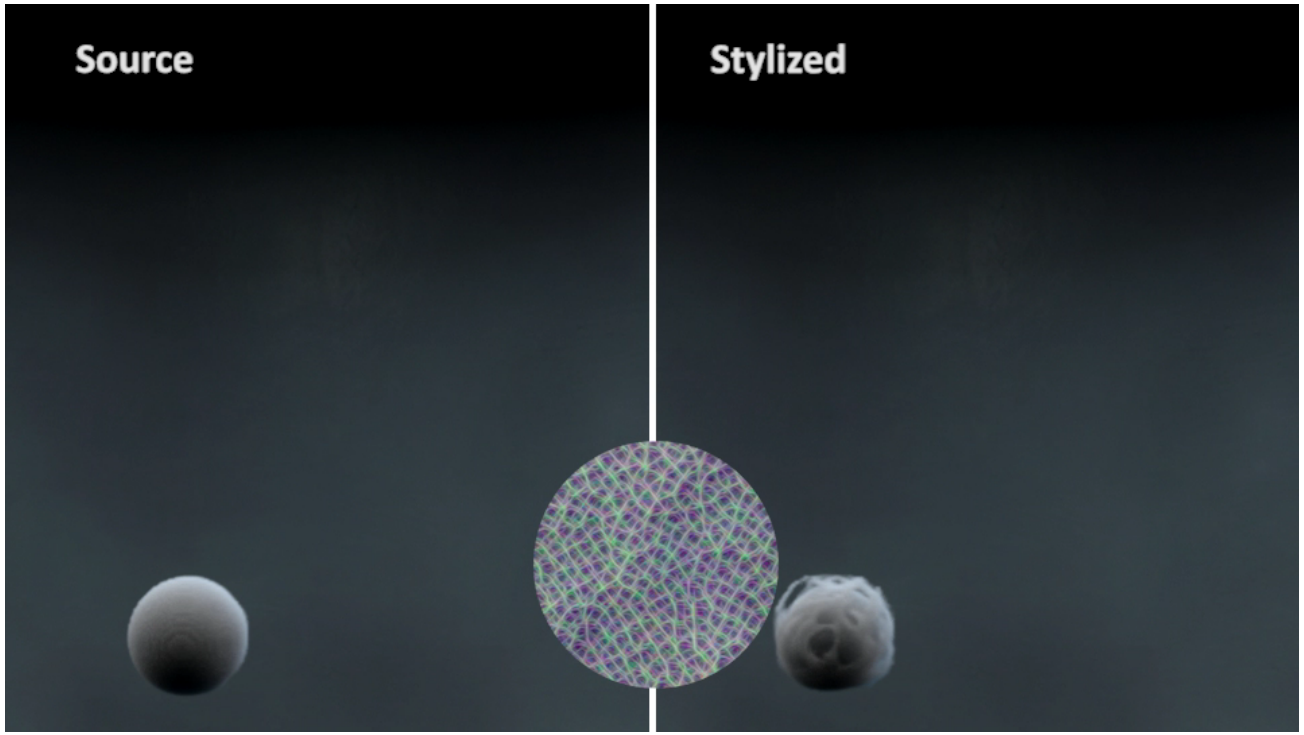


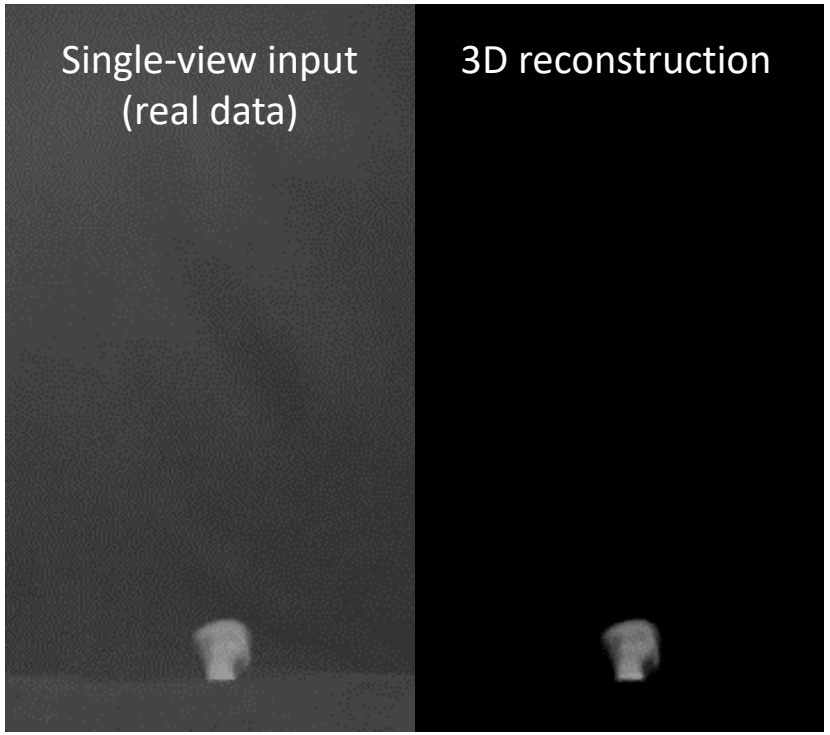


physics simulations for graphics, simulation & deep learning, artistic control, medical applications

Computer Graphics Lab

- Senior researcher at ETH since 2010 / Head of the simulation & animation research
  https://cgl.ethz.ch / https://people.inf.ethz.ch/~sobarbar/

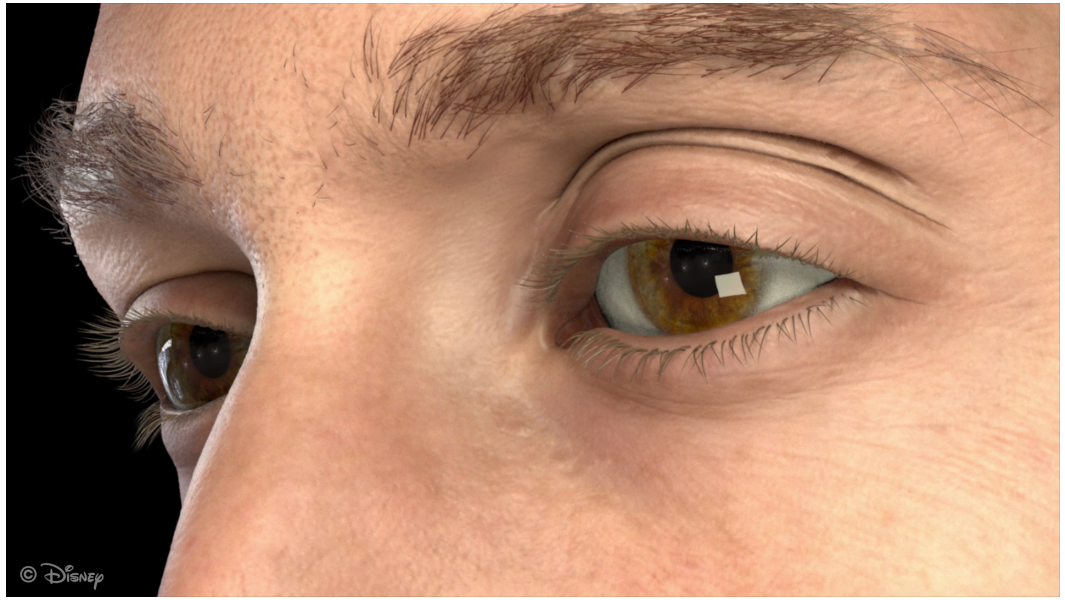


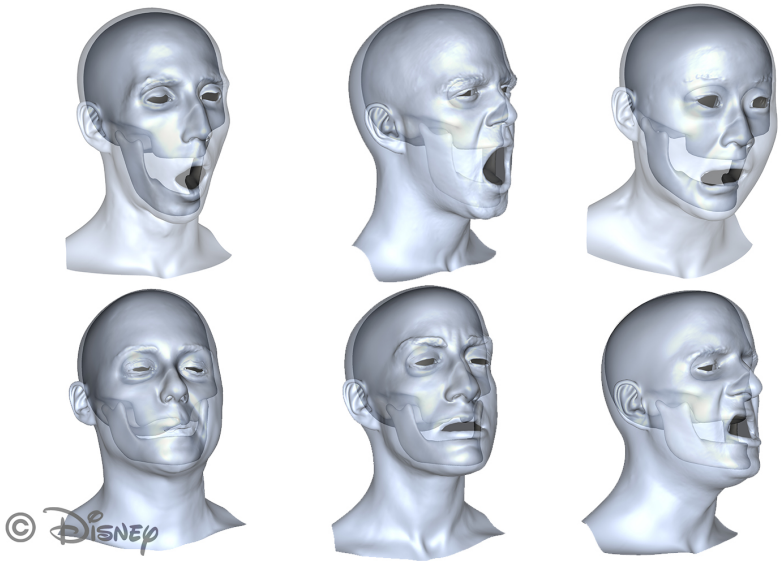


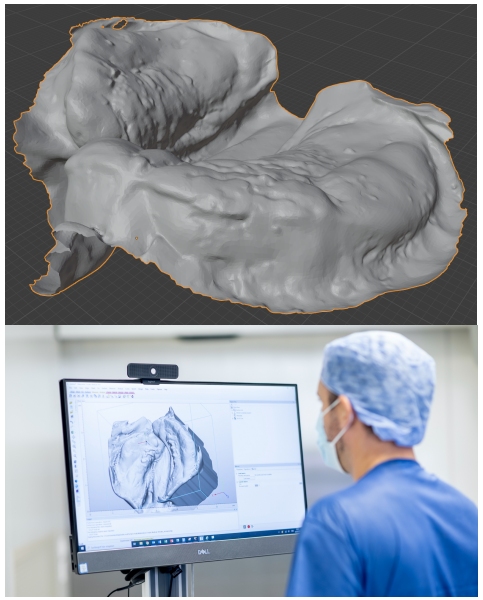
physics simulations for graphics, simulation & deep learning, artistic control, medical applications

Computer Graphics Lab

- Senior researcher at ETH since 2010 / Head of the simulation & animation research
  https://cgl.ethz.ch / https://people.inf.ethz.ch/~sobarbar/



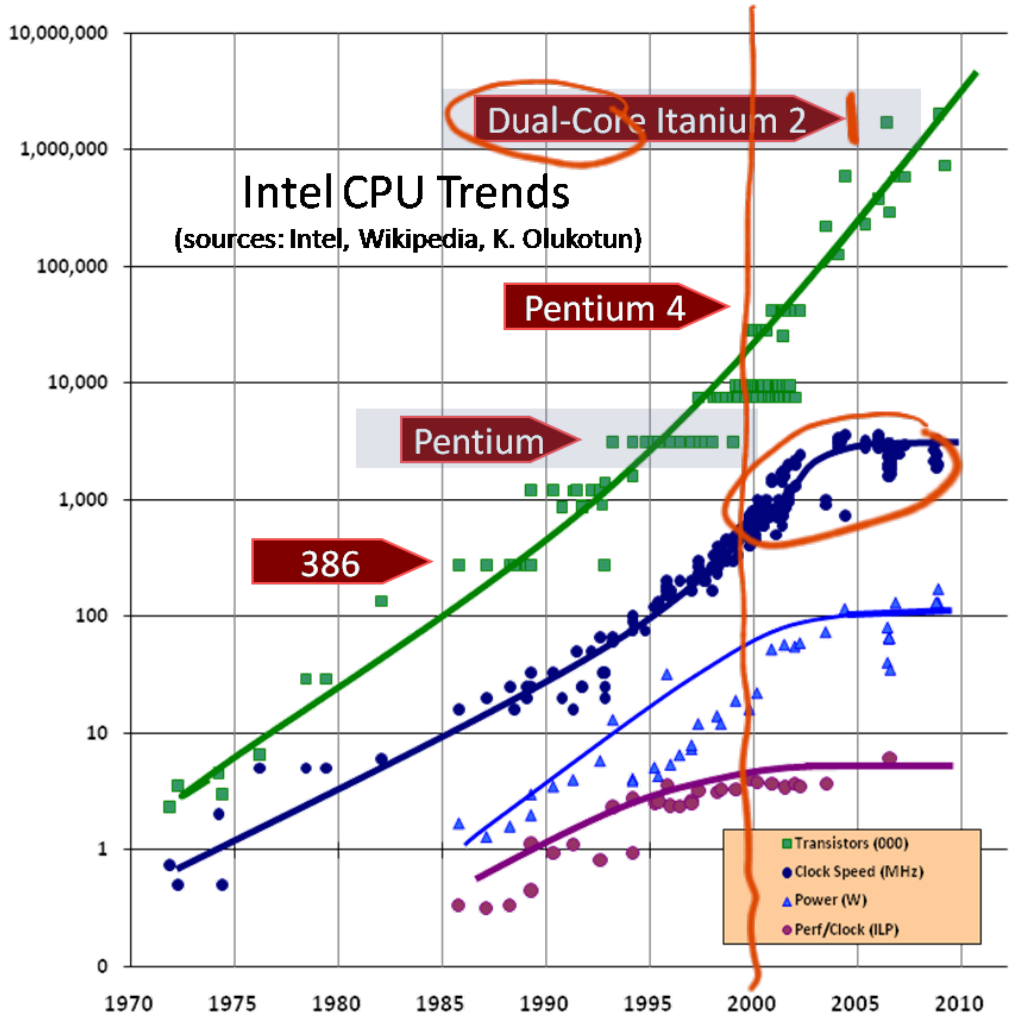




physics simulations for graphics, simulation & deep learning, artistic control, medical applications

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



Herb Sutter for Dr. Dobb's Journal

Course Overview

Parallel Programming (252-0029-00L)

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

- Lecture Language: Denglisch

Course Coordination

- Lectures 2 x week:
  - Tuesday 10-12 Part I Online (Zoom), Part II ?
  - Wednesday 14-16 Part I Online (Zoom), Part II ?

- Weekly Exercise Sessions
  - Enroll via myStudies
  - Wednesday 16-18 or Friday 10-12

Course Material and Communication

Course website:

https://spcl.inf.ethz.ch/Teaching/2021-pp/

Moodle

https://moodle-app2.let.ethz.ch/course/view.php?id=14267

Lecture slides, exercises, recordings, forum

About This Course

Head TAs:

- Rafael Wampfler (Part I)
- Timo Schneider (Part II)

Teaching Assistants (Part I):

- Prashant Chandran
- Daniel Dorda
- Robin Renggli
- Cliff Hodel
- Ricardo Heinzmann
- Jonas Maier
- Lukas Möller
- Michael Heider

Communication:

Your TA ▶ Head TA ▶ Lecturer

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

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:

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

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

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

Schedule (Part I)

Lecture

Feb.23	Introduction & Course Overview
Feb.24	Java Recap and JVM Overview
Mar.02	Introduction to Threads and Synchronization (Part I)
Mar.03	Introduction to Threads and Synchronization (Part II)
Mar.09	Introduction to Threads and Synchronization (Part II)
Mar.10	Parallel Architectures: Parallelism on the Hardware Level (Part I)
Mar.16	Parallel Architectures: Parallelism on the Hardware Level (Part II)
Mar.17	Basic Concepts in Parallelism
Mar.23	Divide and Conquer, Cilk-style bounds
Mar.24	Divide and Conquer, Cilk-style bounds
Mar.30	ForkJoin Framework and Task Parallel Algorithms
Mar.31	ForkJoin Framework and Task Parallel Algorithms / Shared Memory Concurrency, Locks and Data Races
Apr.06	Easter break
Apr.07	Easter break
Apr.13	Shared Memory Concurrency, Locks and Data Races (High-level data races)
Apr.14	Fast Forward

Exercises

Ex 1	Introduction
Ex 2	Introduction to Multi-threading
Ex 3	Multi-threading
Ex 4	Parallel Models
Ex 5	Divide and Conquer
Ex 6	Task Parallelism
Ex 7	Synchronization and Resource Sharing

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.

Schedule (Part II)

Lecture

Apr.20	Data Races - Implementing locks with Atomic Registers
Apr.21	Data Races - Implementing locks with Atomic Registers II
Apr.27	Beyond Locks I: Spinlocks, Deadlocks, Semaphores
Apr.28	Beyond Locks II: Semaphore, Barrier, Producer-/Consumer, Monitors
May.04	Readers/Writers Lock, Lock Granularity: Coarse Grained, Fine Grained, Optimal, and Lazy Synchronization
May.05	Lock tricks, skip lists, and without Locks I
May.11	Without Locks II
May.12	ABA Problem, Concurrency Theory
May.18	Sequential Consistency, Consensus, Transactional Memory
May.19	Consensus Hierarchy + Transactional Memory
May.25	Transactional Memory + Message Passing
May.26	Message Passing
Jun.01	Consensus Proof and Reductions
Jun.02	Parallel Sorting

Exercises

Ex 8	Synchronization II
Ex 9	Reasoning about Locks / Java Memory Model Basics
Ex 10	Advanced Synchronization Mechanisms
Ex 11	Advance Synchronization Mechanisms
Ex 12	Linearizability
Ex 13	Software Transactional Memory
Ex 14	MPI + Reductions

Terminology

- https://cgl.ethz.ch/teaching/parallelprog21/pages/terminology.html

Parallel Programming (1st half): Terminology

atomic
A statement or instruction is (truly) atomic if it is executed by the CPU in a single, non-interruptible step.

abstractly atomic
A statement or instruction that, at a certain level of abstraction, appears to be executed atomically. E.g. from a caller's perspective, a method `synchronized append(x)` of a queue appears to append element `x` in one step, but from the queue's perspective, this might take several steps.

Amdahl's law
Specifies the maximum amount of speedup that can be achieved for a program with a given sequential part. The pessimistic view on scalability.

bad interleaving
An interleaving that yields a problematic or otherwise undesirable computation. E.g. an incorrect result, a deadlock or non-deterministic output.

busy waiting
Occurs when a thead busily (actively) waits, e.g. by spinning in a loop, for a condition to become true. In the opposite scenario, the thread sleeps (i.e. is blocked; in Java: `join()`, `wait()`) until the condition becomes true. Trade-off: busy waiting uses up CPU time, whereas blocking may cause additional context switches.

cache coherence protocols
Hardware protocols that ensure consistency across caches, typically by tracking which locations are cached, and synchronising them if necessary.

cilk-style programming
Parallel programming idiom: To compute a program, execute code and spawn new tasks if required. Before returning, wait for all spawned tasks to complete. The system manages the eventual execution of the spawned tasks potentially in parallel. spawning and waiting on tasks creates a task graph which is a DAG.

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

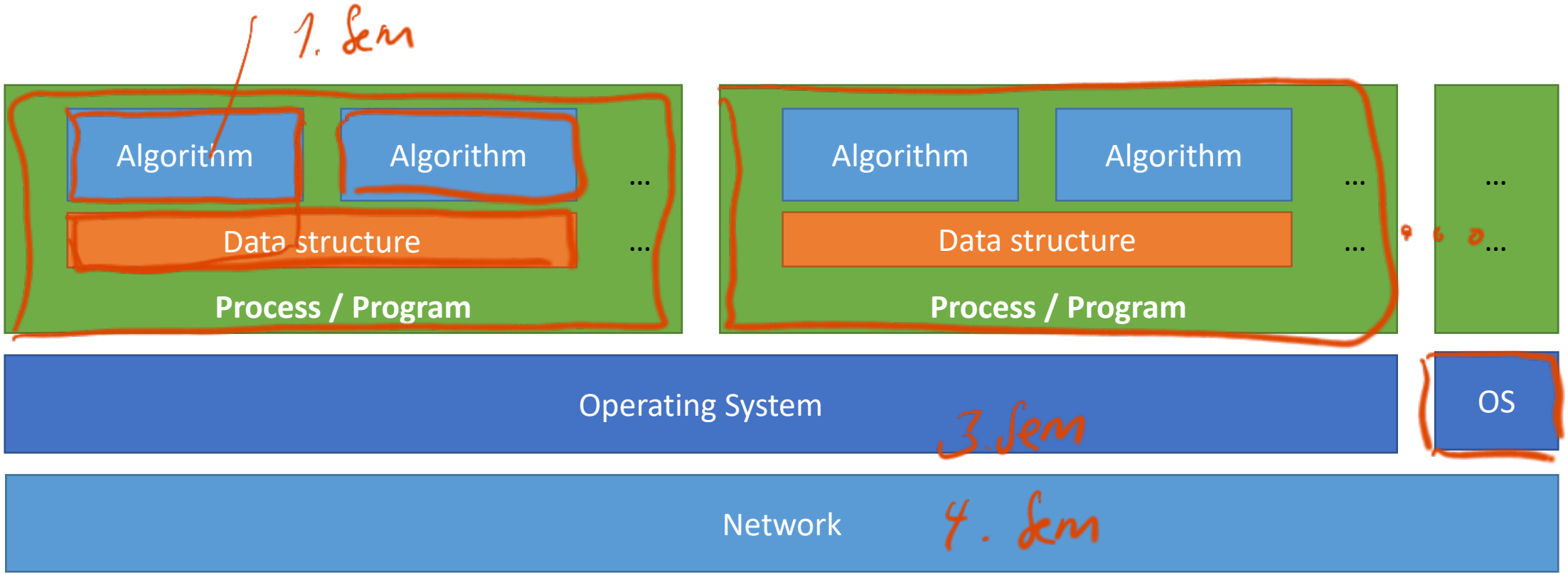
System = Processes + Objects + Communication

This class

This class

Intro to Programming

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

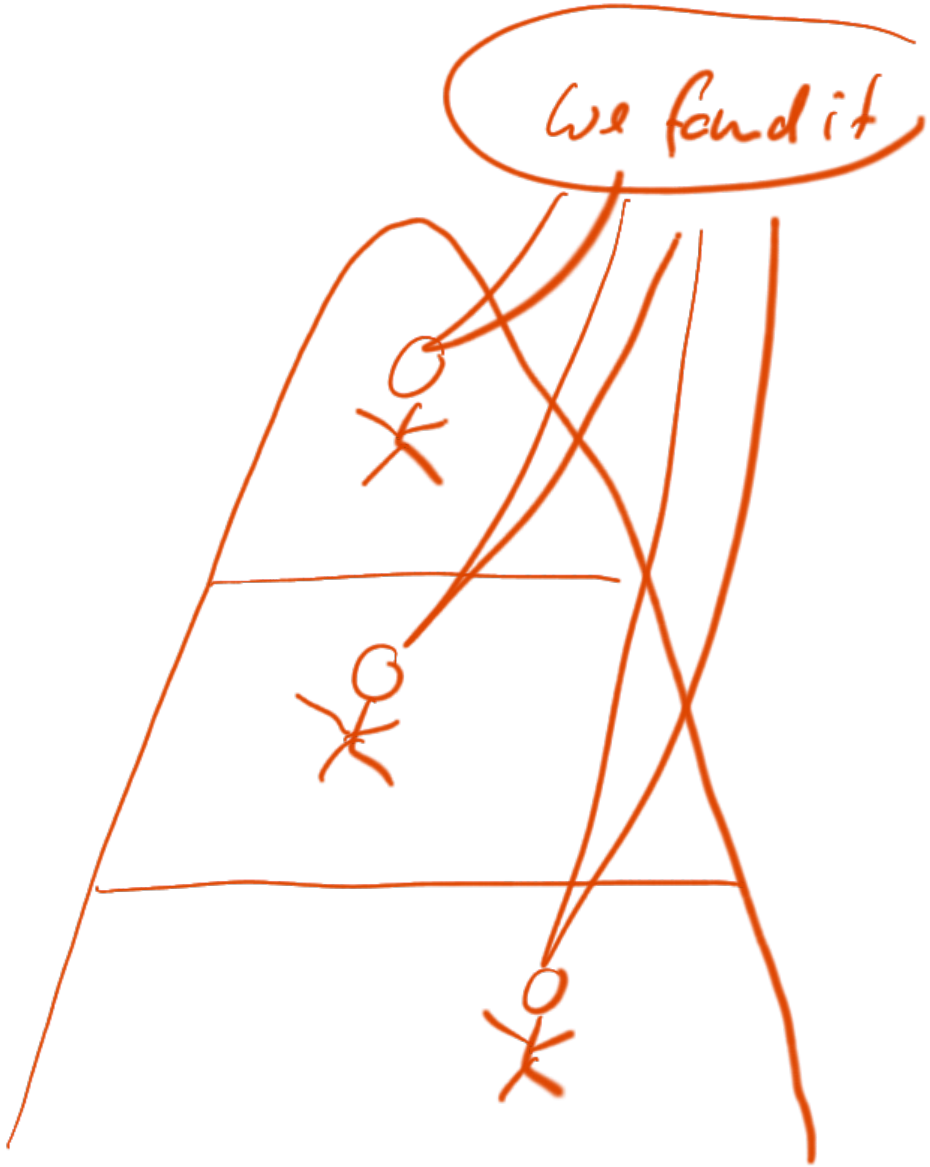
Basic knowledge of programming concepts:

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

Basic understanding of computer architectures:

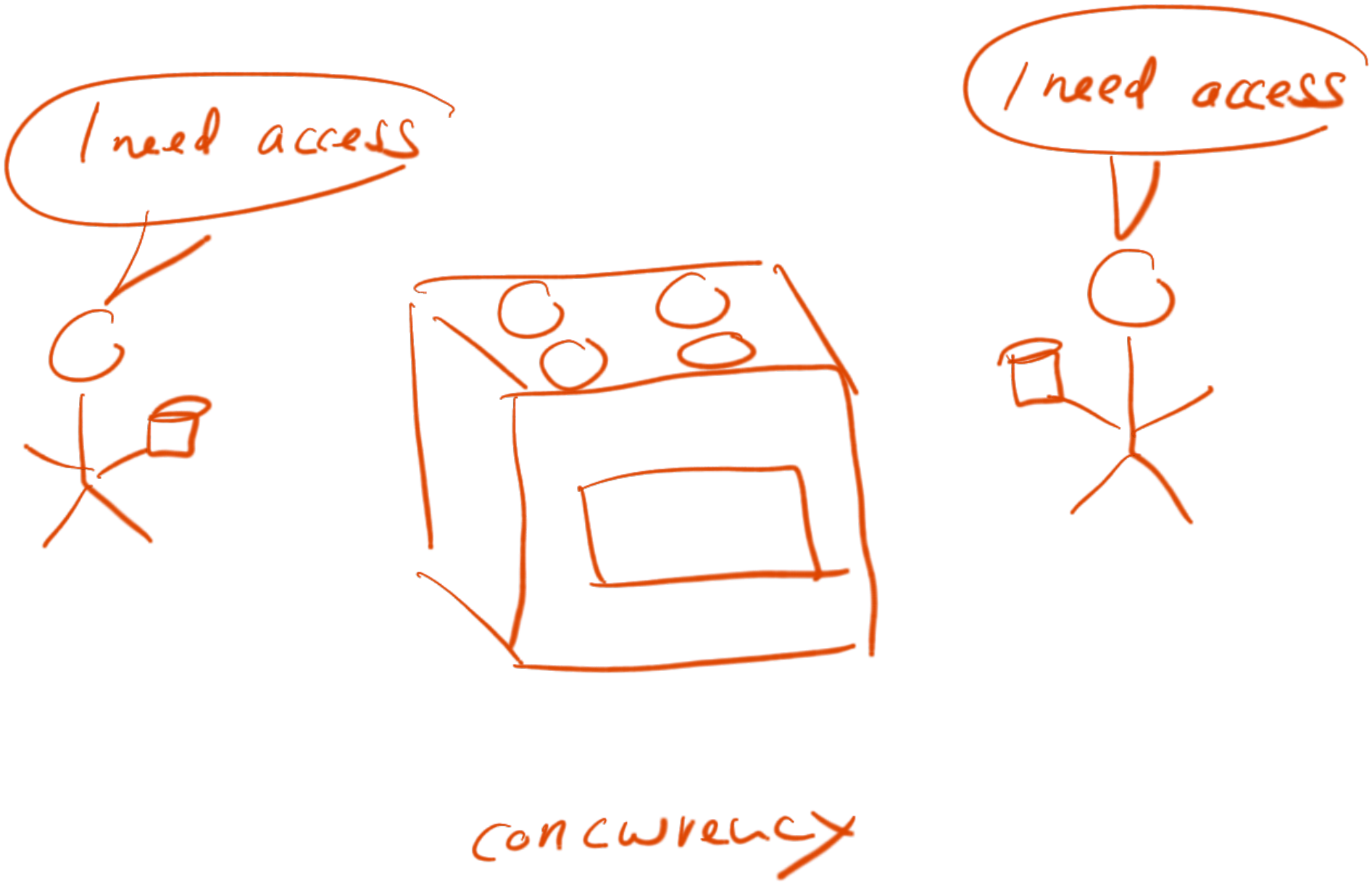
No detailed knowledge necessary (we will cover some)

sequential

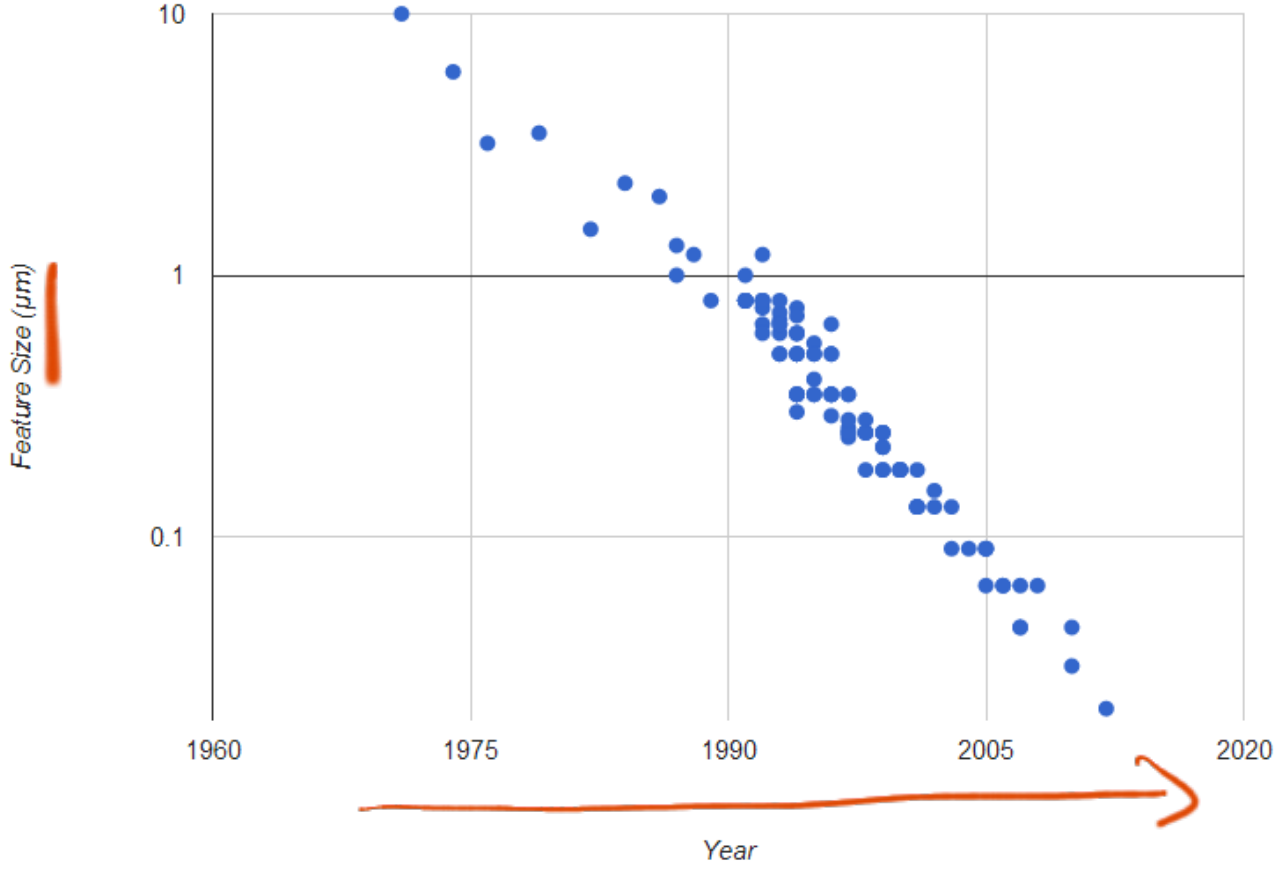


parallel

Image source: http://jolyon.co.uk


I need access
I need access
concurrency

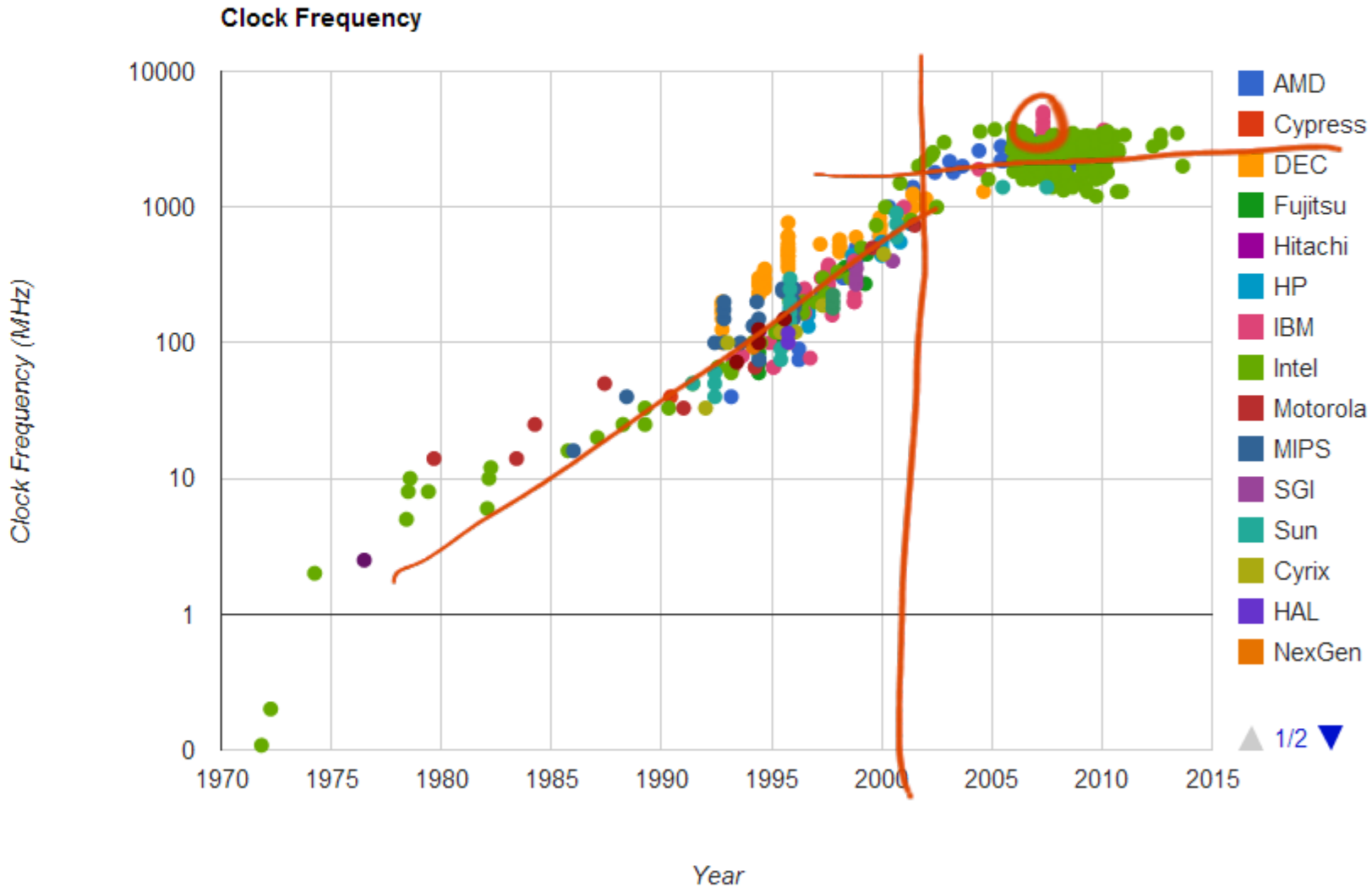
Motivation – Why Parallelism?



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)

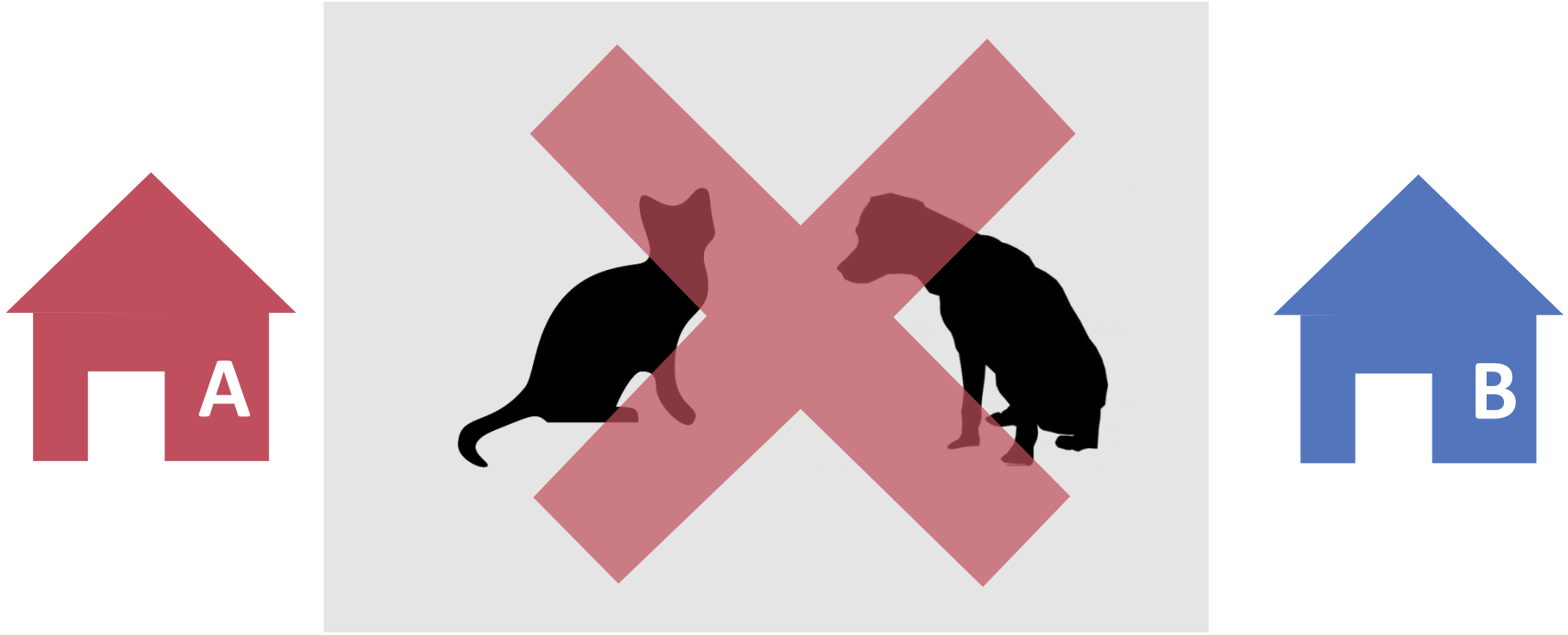
- Many-Core/GPUs etc:
  - Simpler Control Hardware
  - Pro: Potentially more power efficient (Ops / Watt)
  - Con: More restrictive / complex programming models [but useful in many domains, e.g. deep learning].

Three stories

1. MUTUAL EXCLUSION

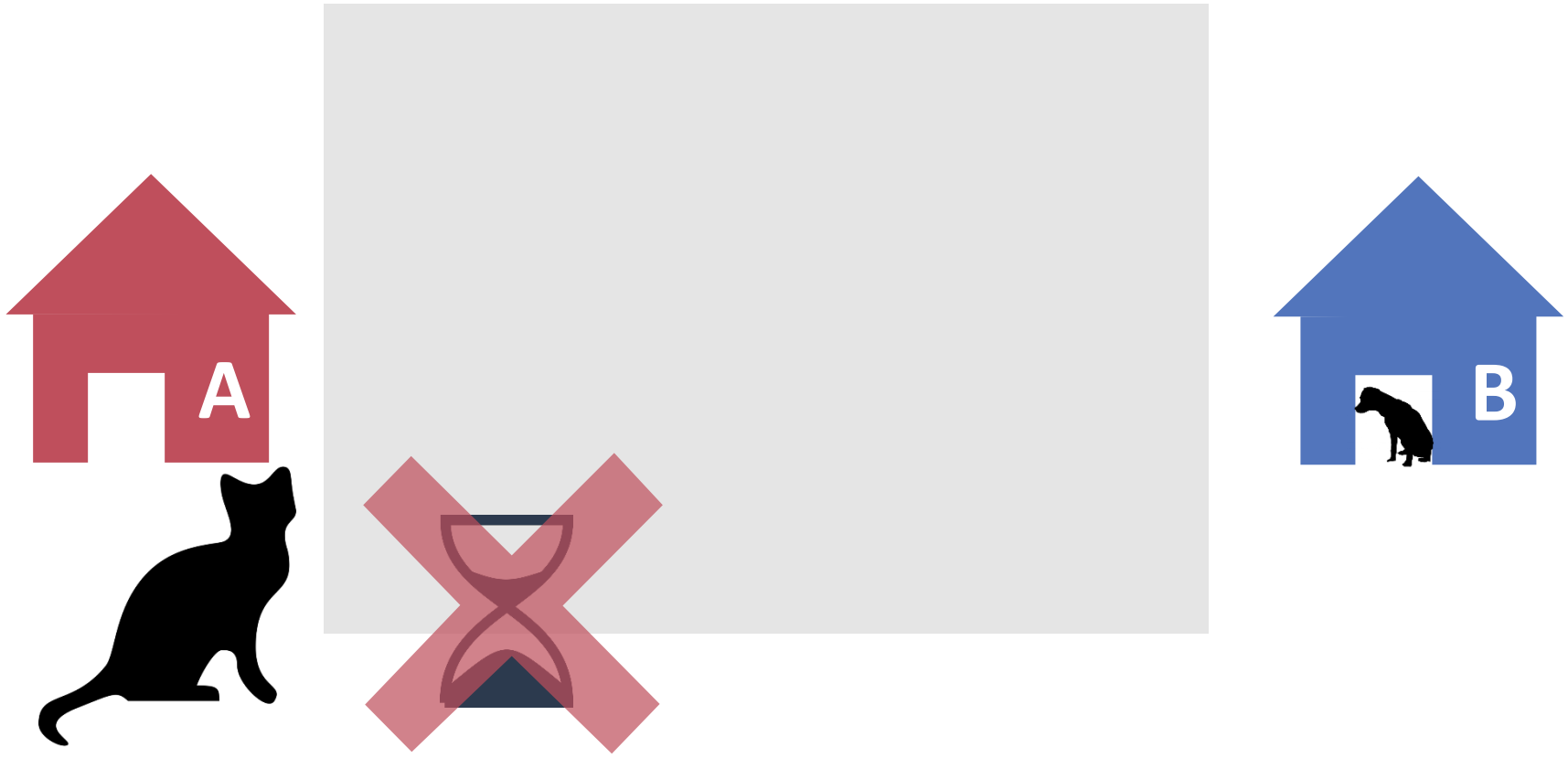
Alice’s Cat vs. Bob’s Dog

Requirement I: Mutual Exclusion !

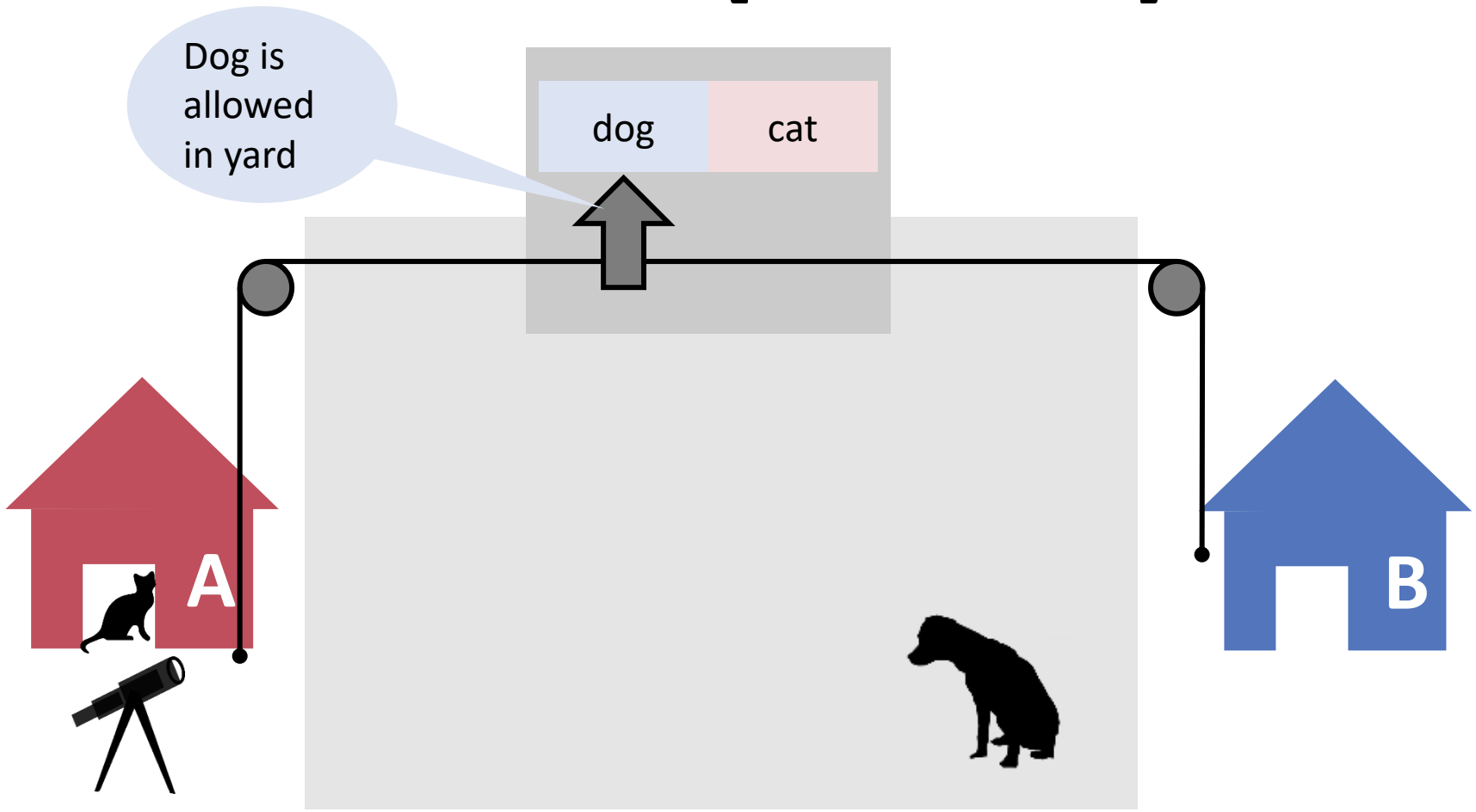



A
B

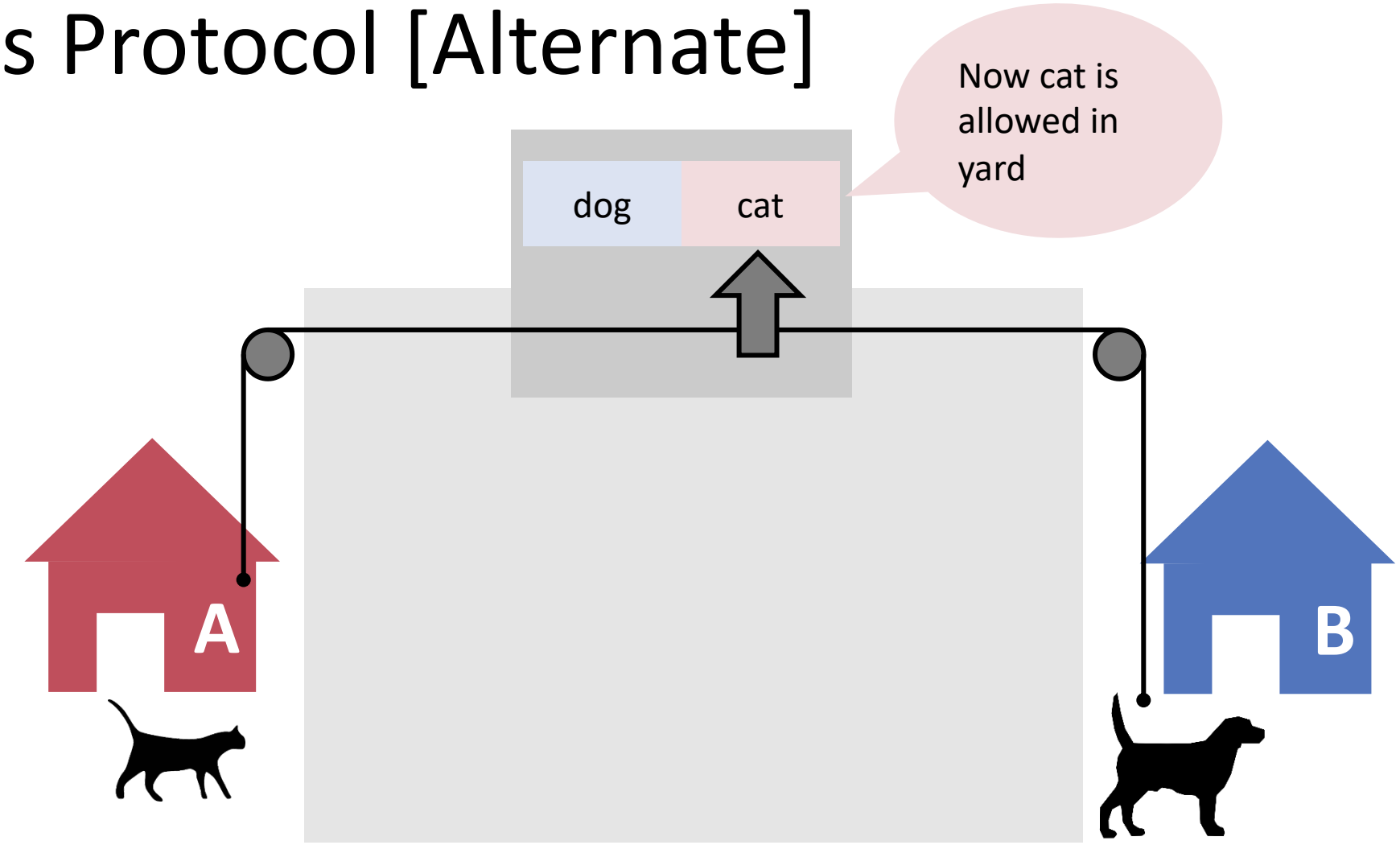
Requirement II: No Lockout when free



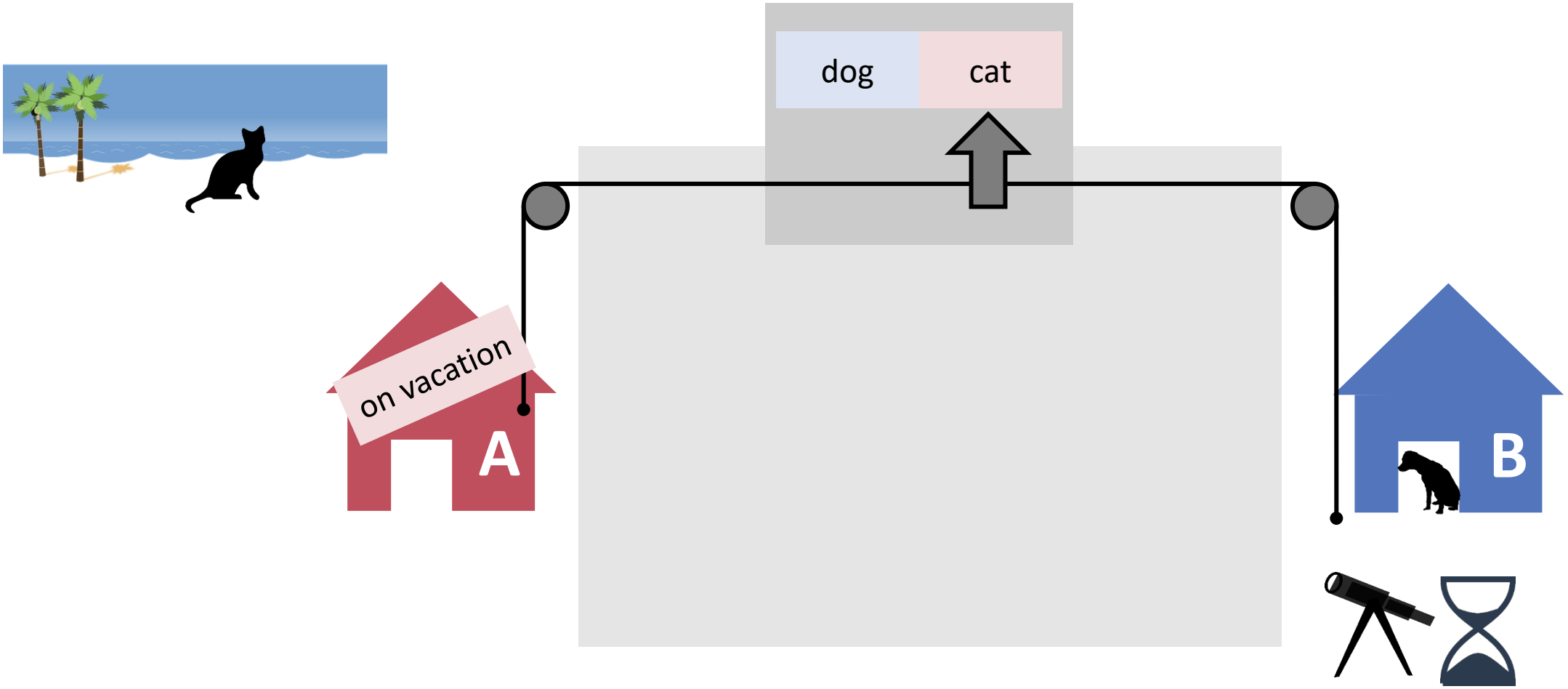
Communication: Idea 1 [Alternate]



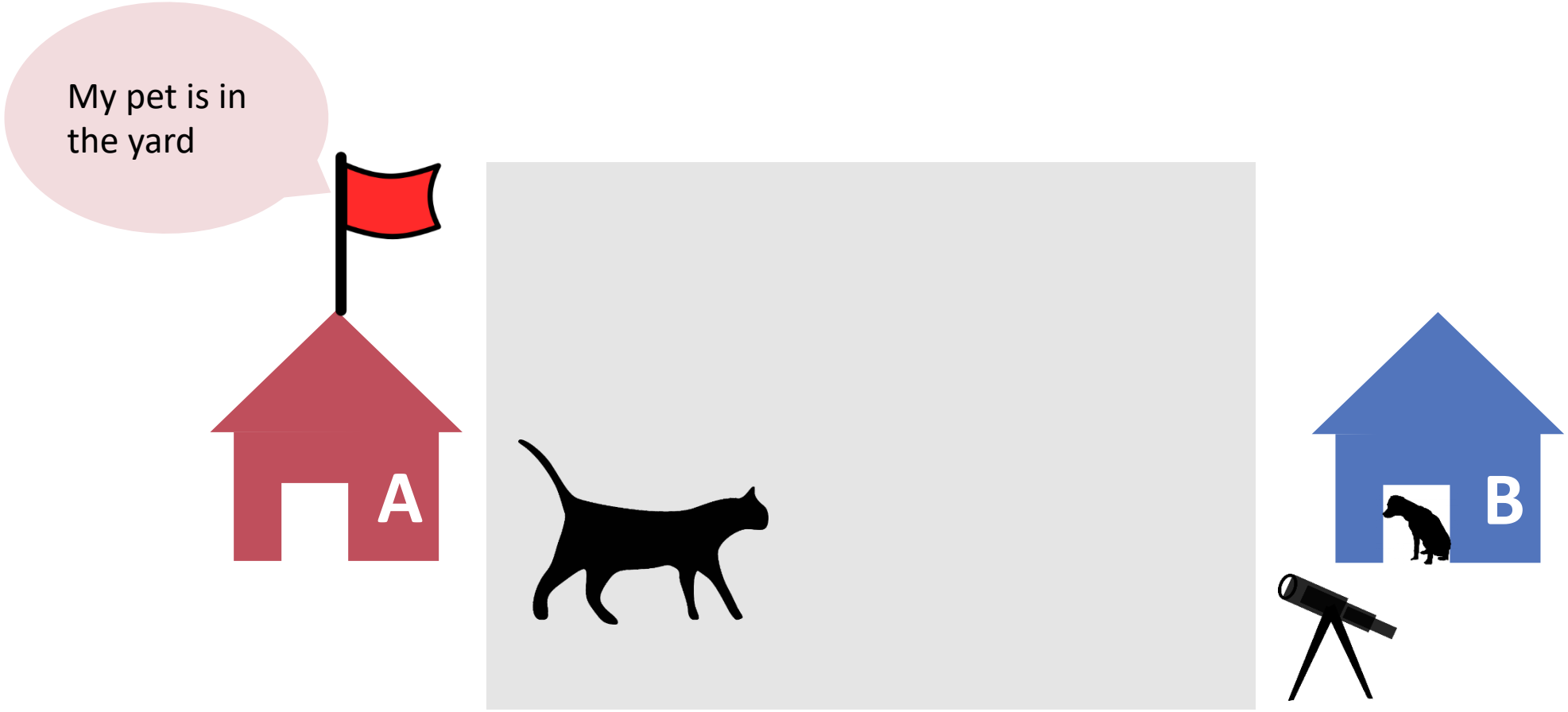
Access Protocol [Alternate]



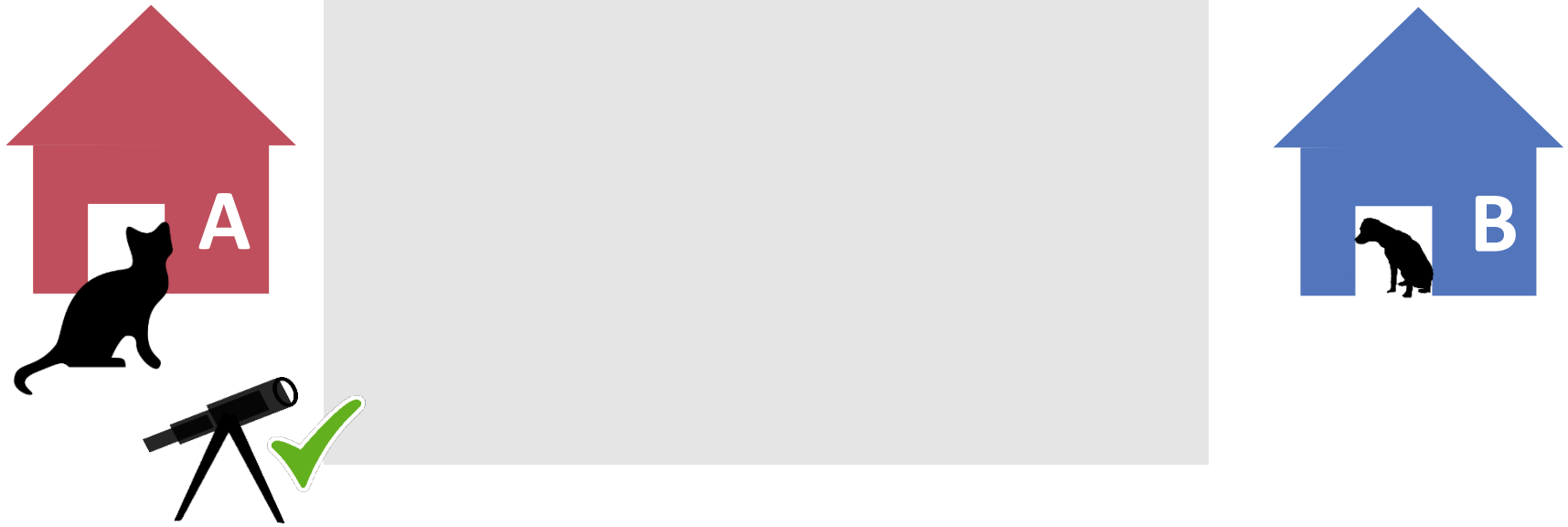
Problem: Starvation!

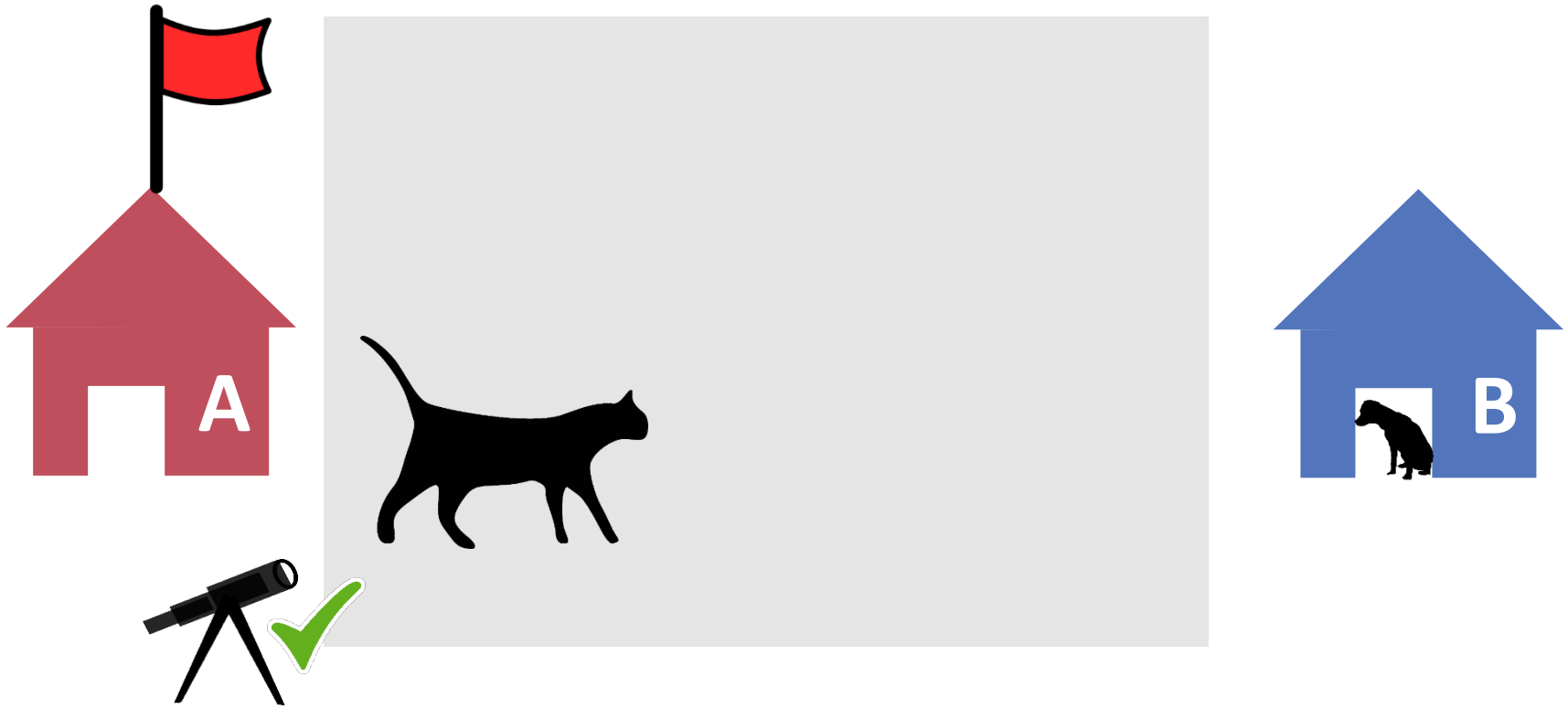


Communication: Idea 2 [Notification]

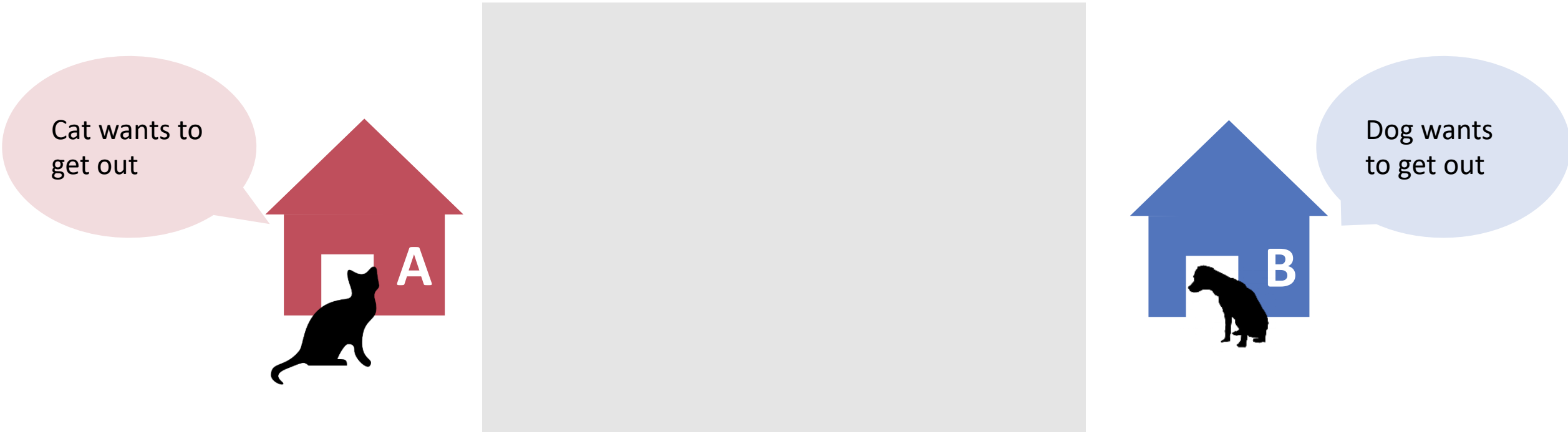


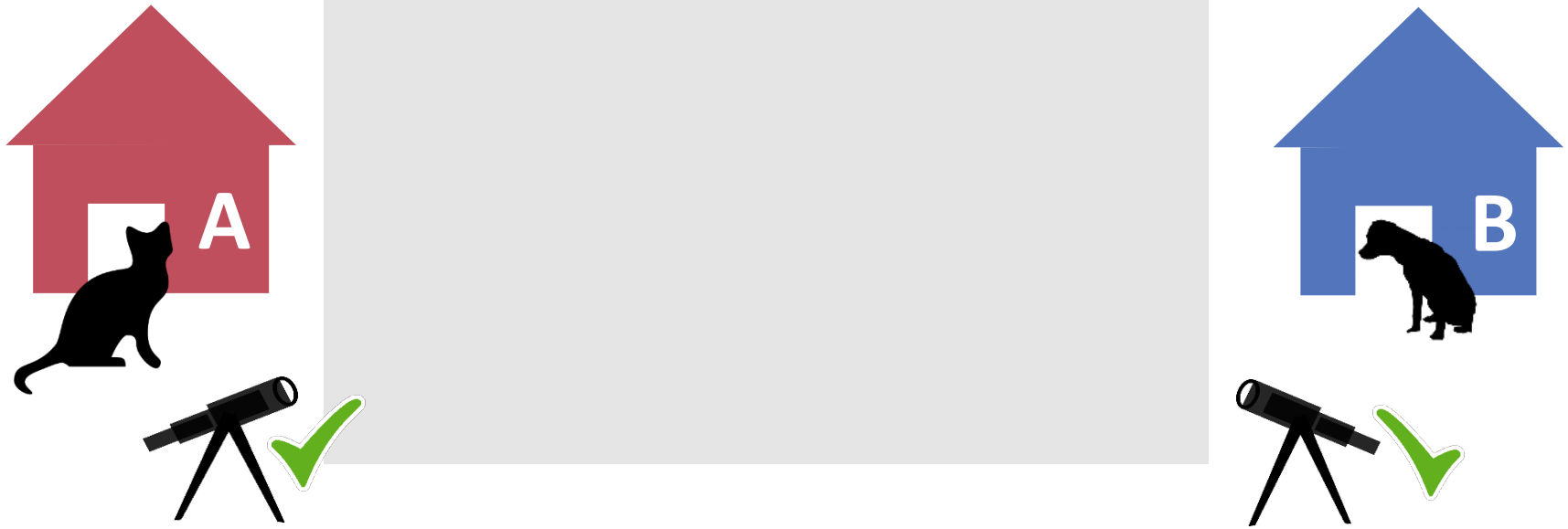
Access Protocol 2.1: Idea

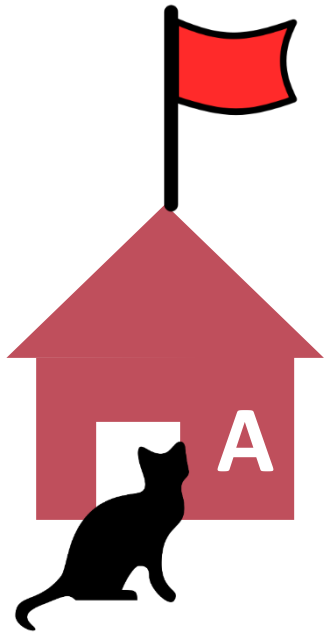

A
B

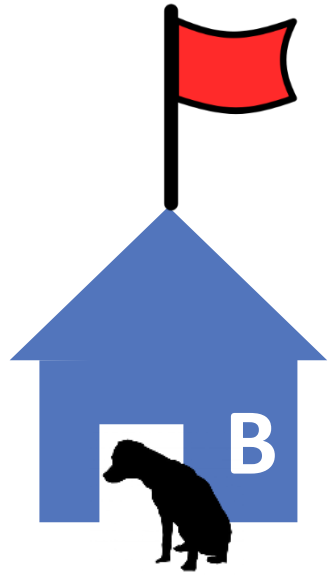

A
B

Another Scenario

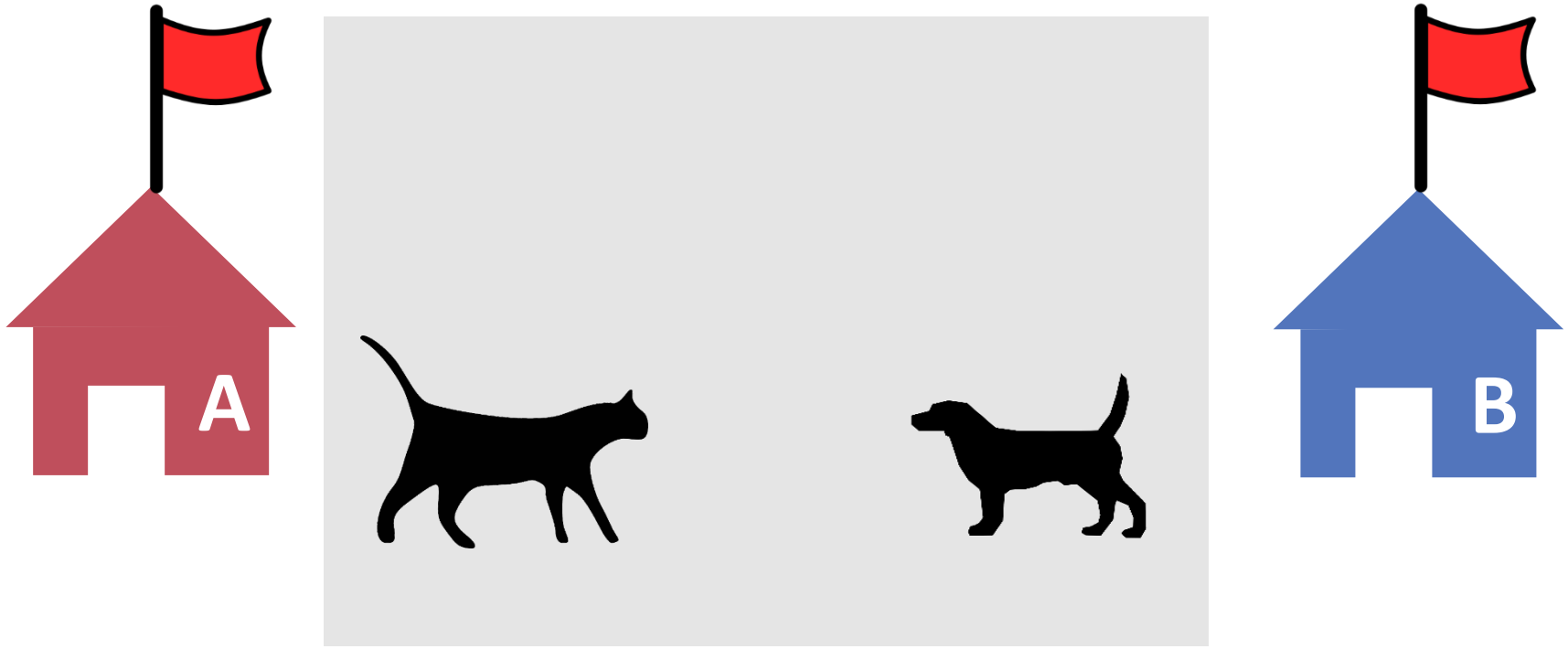



A
B


A

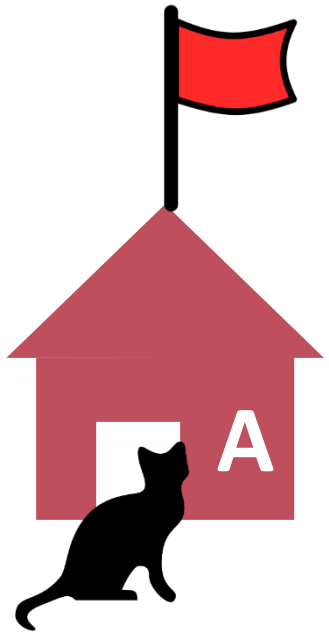

B

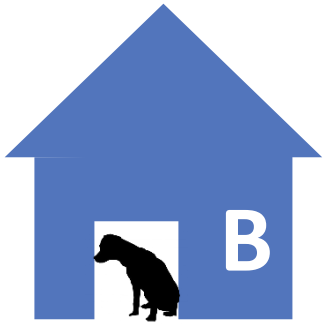
Problem: No Mutual Exclusion!

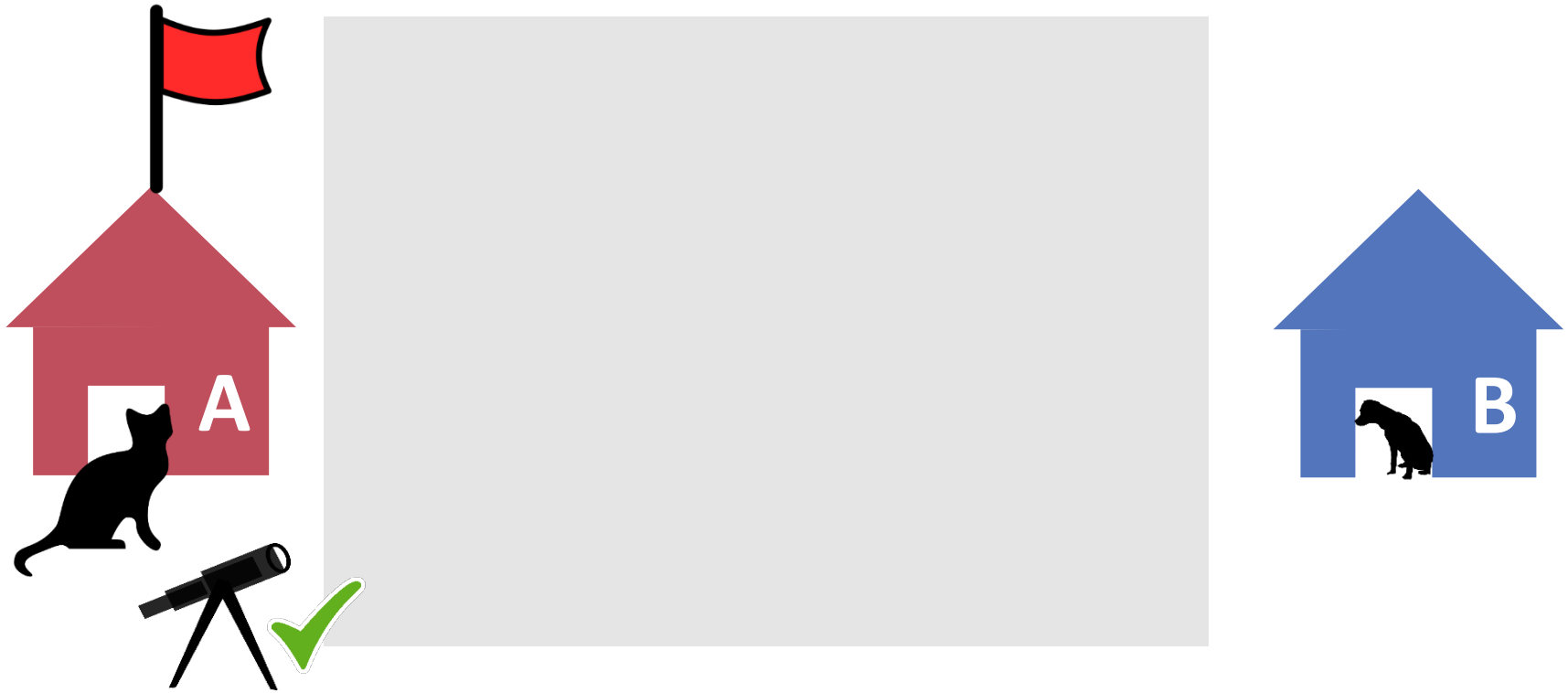


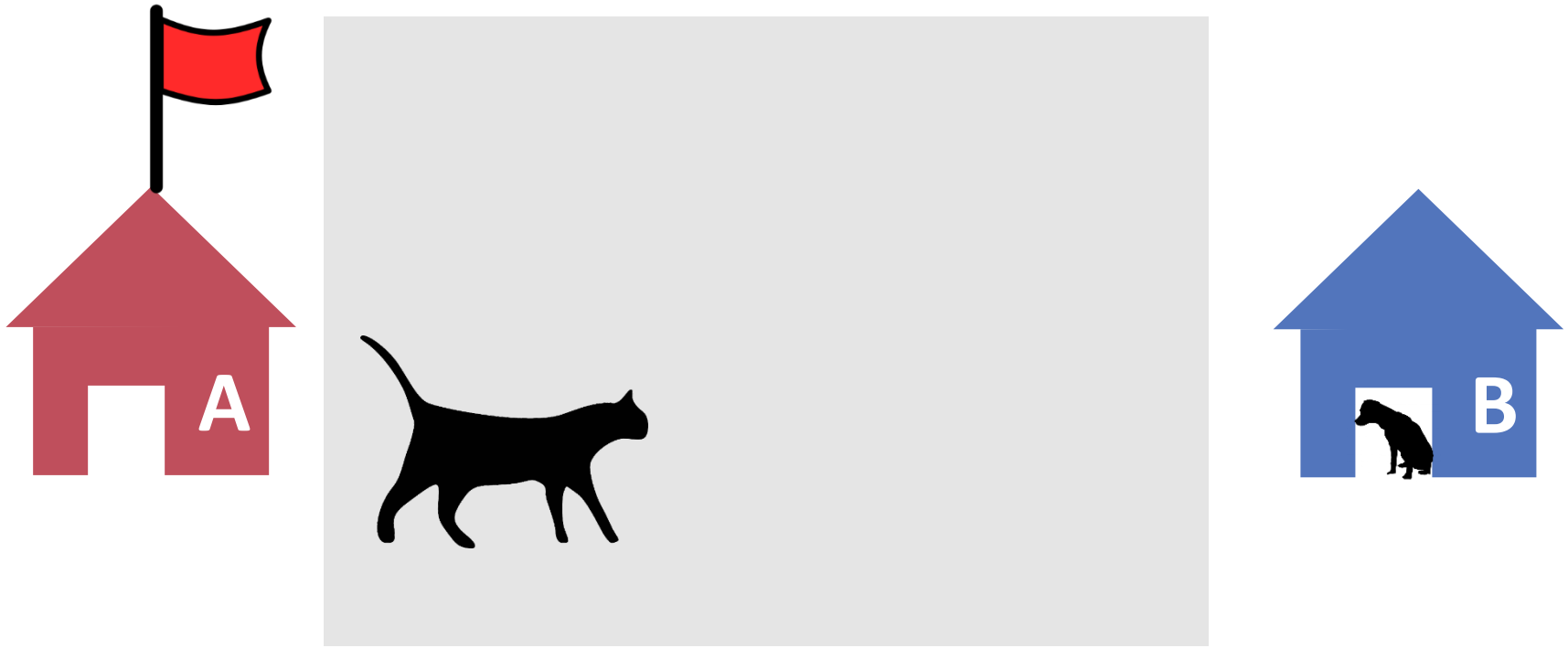
Checking Flags Twice Does Not Help: Deadlock!

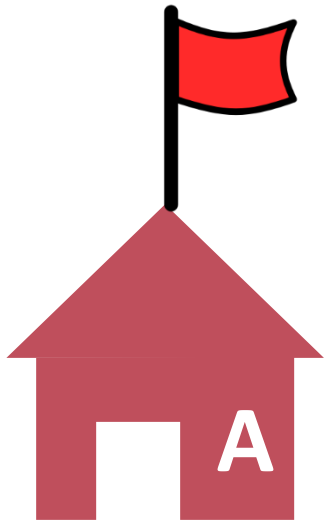
Access Protocol 2.2

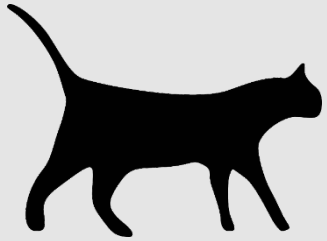

A

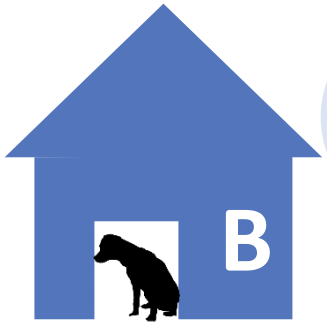

B


A
B

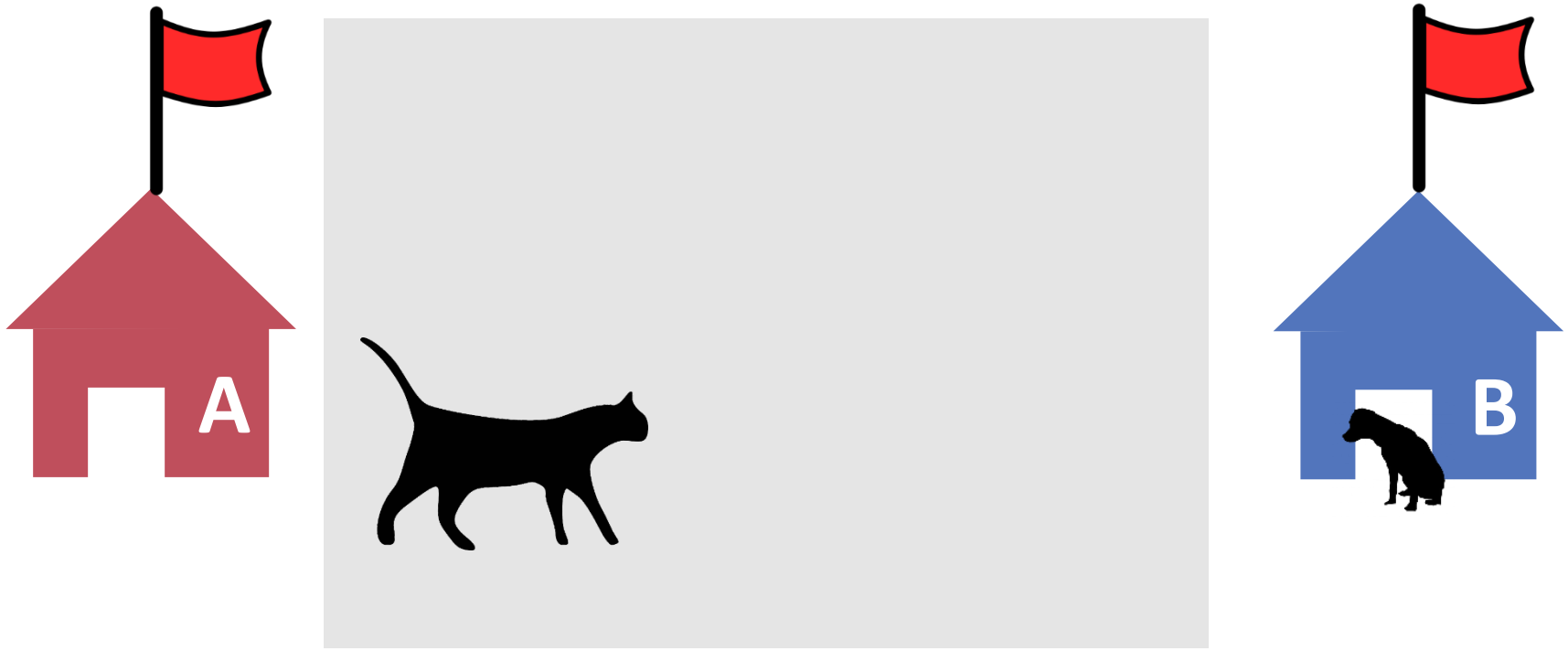

A
B

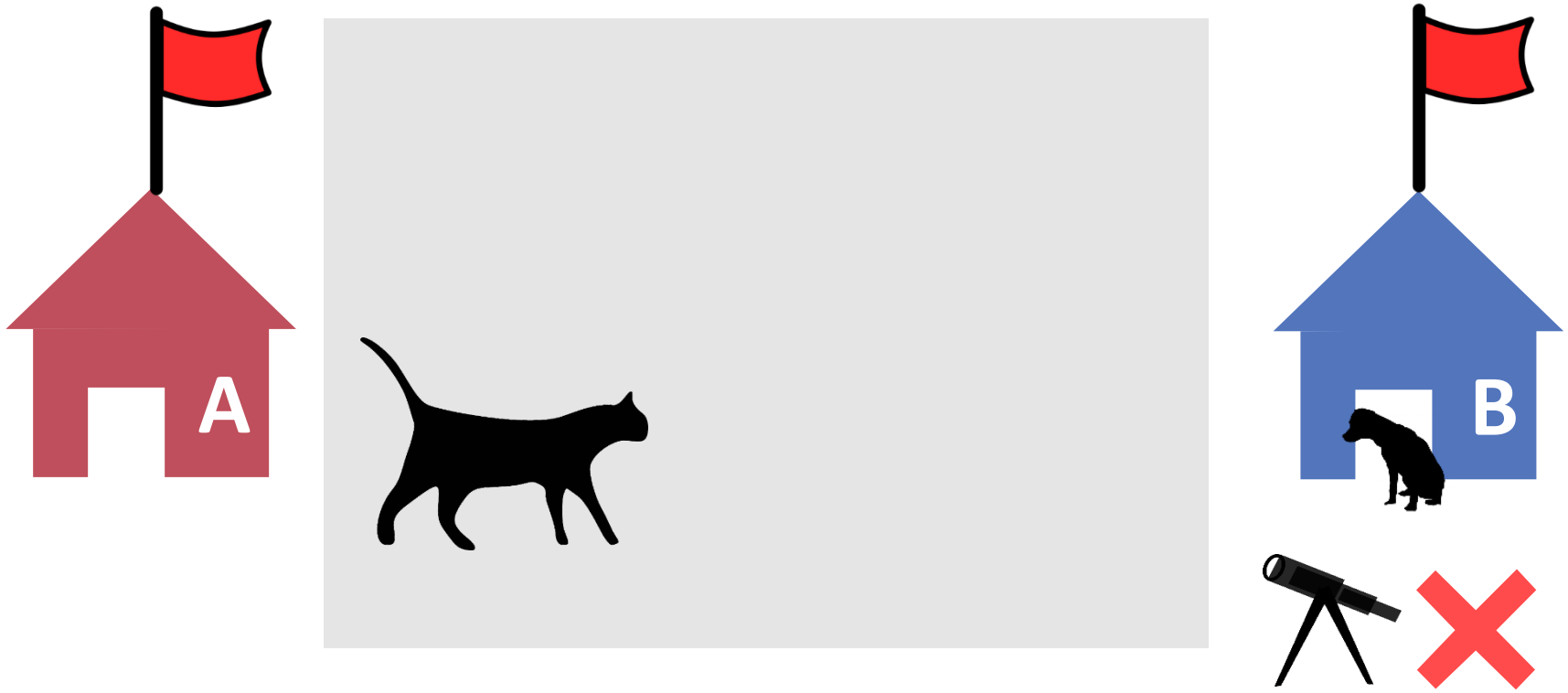




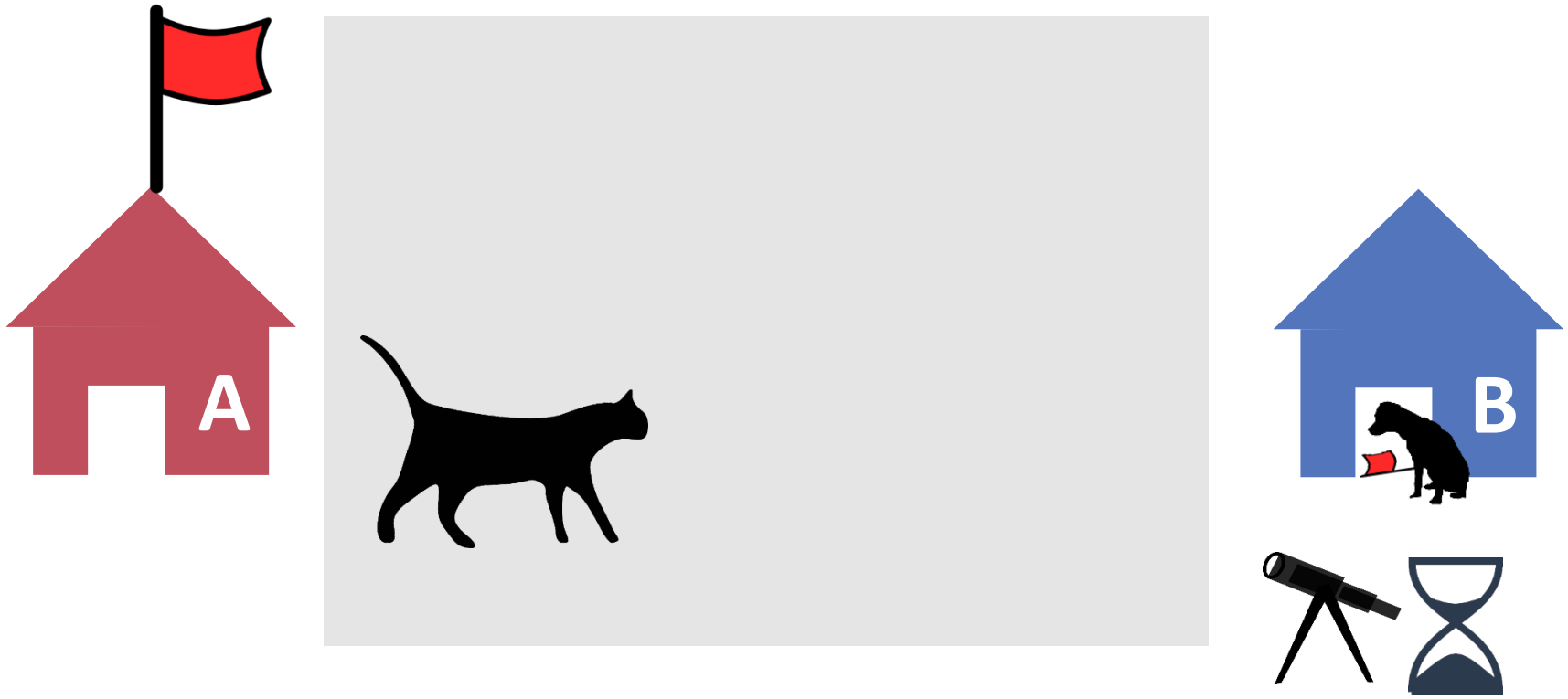


Dog wants
to get out


A
B

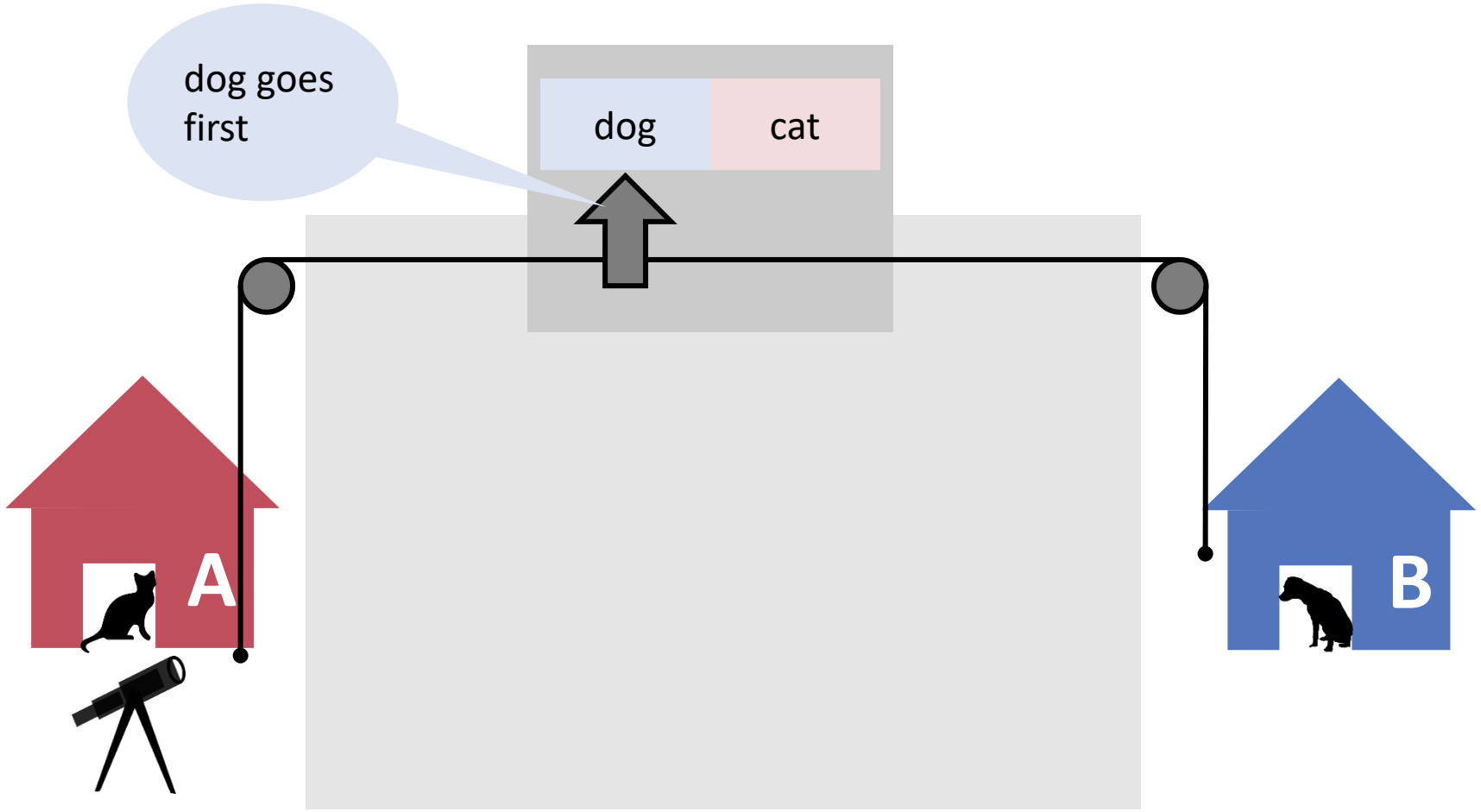

A
B

Access Protocol 2.2 is provably correct

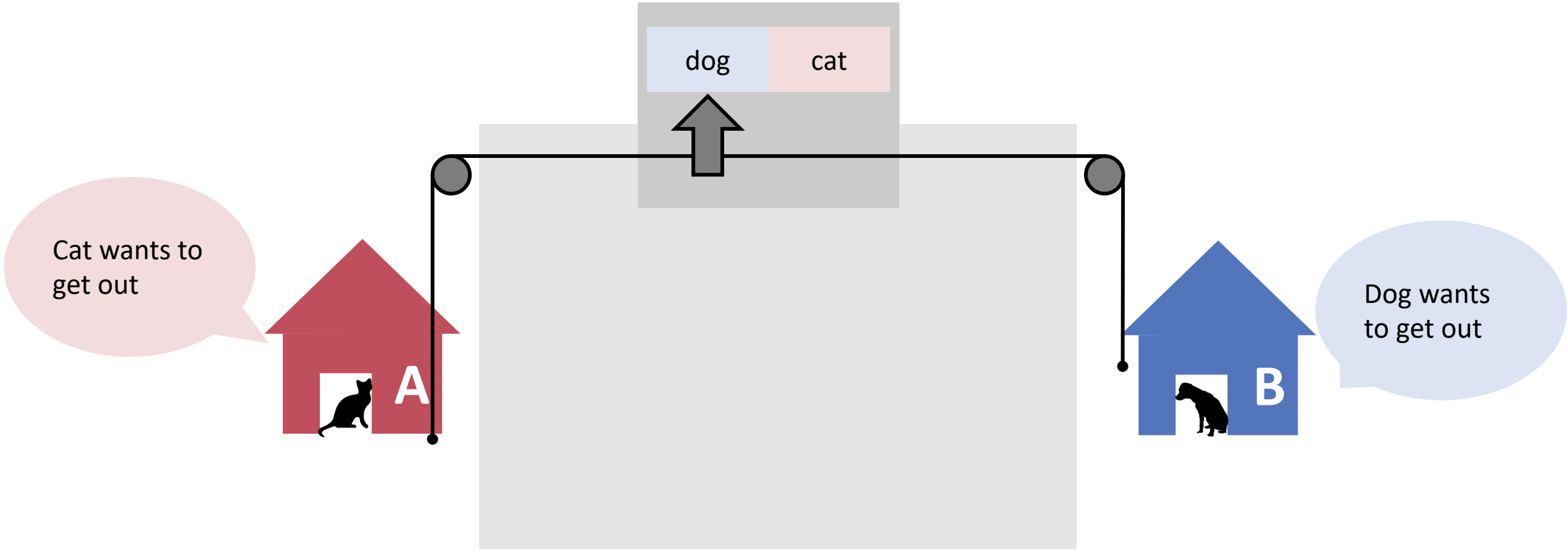


Minor (?) Problems: Livelock, Starvation

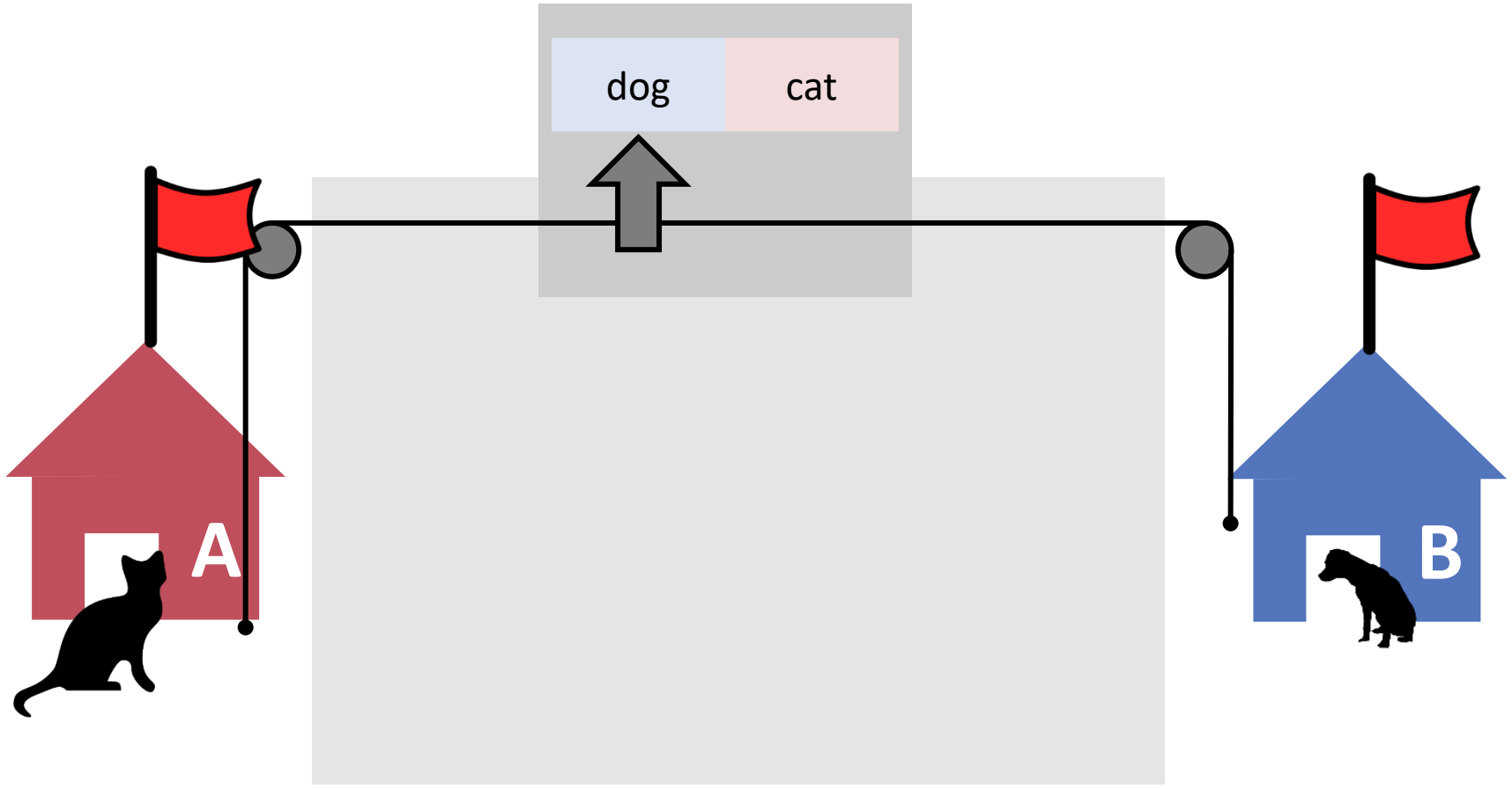
Final Solution



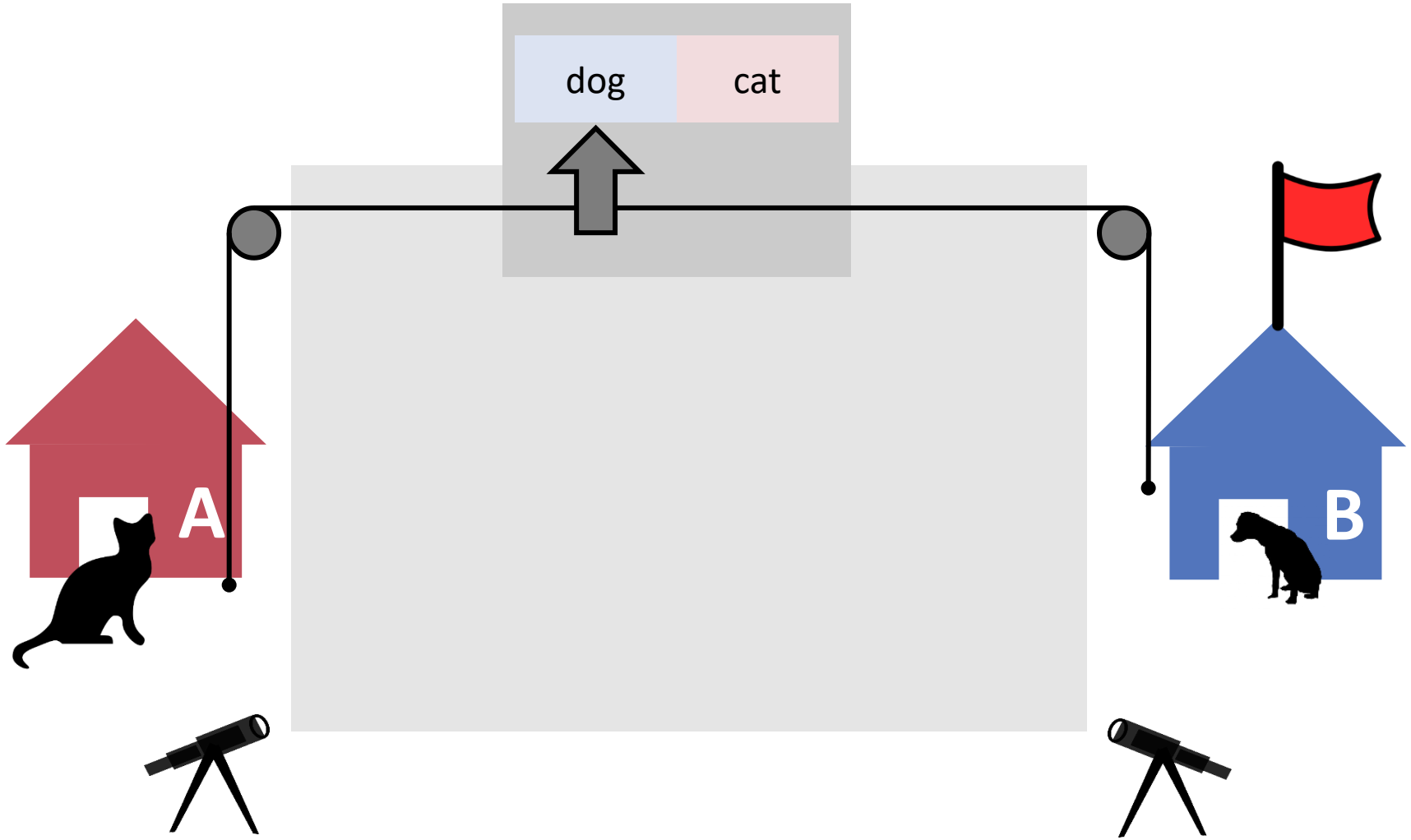
Final Solution



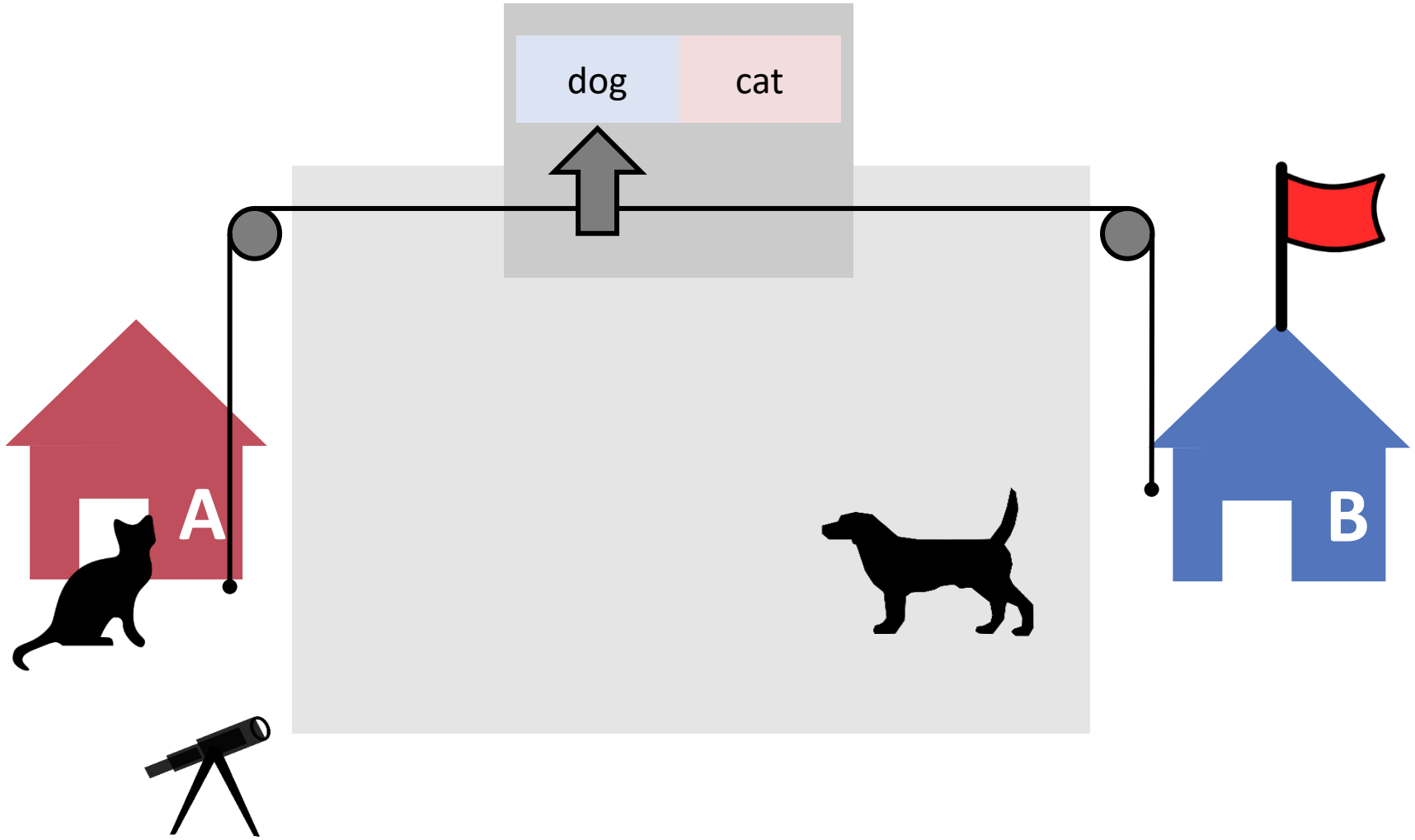
Final Solution



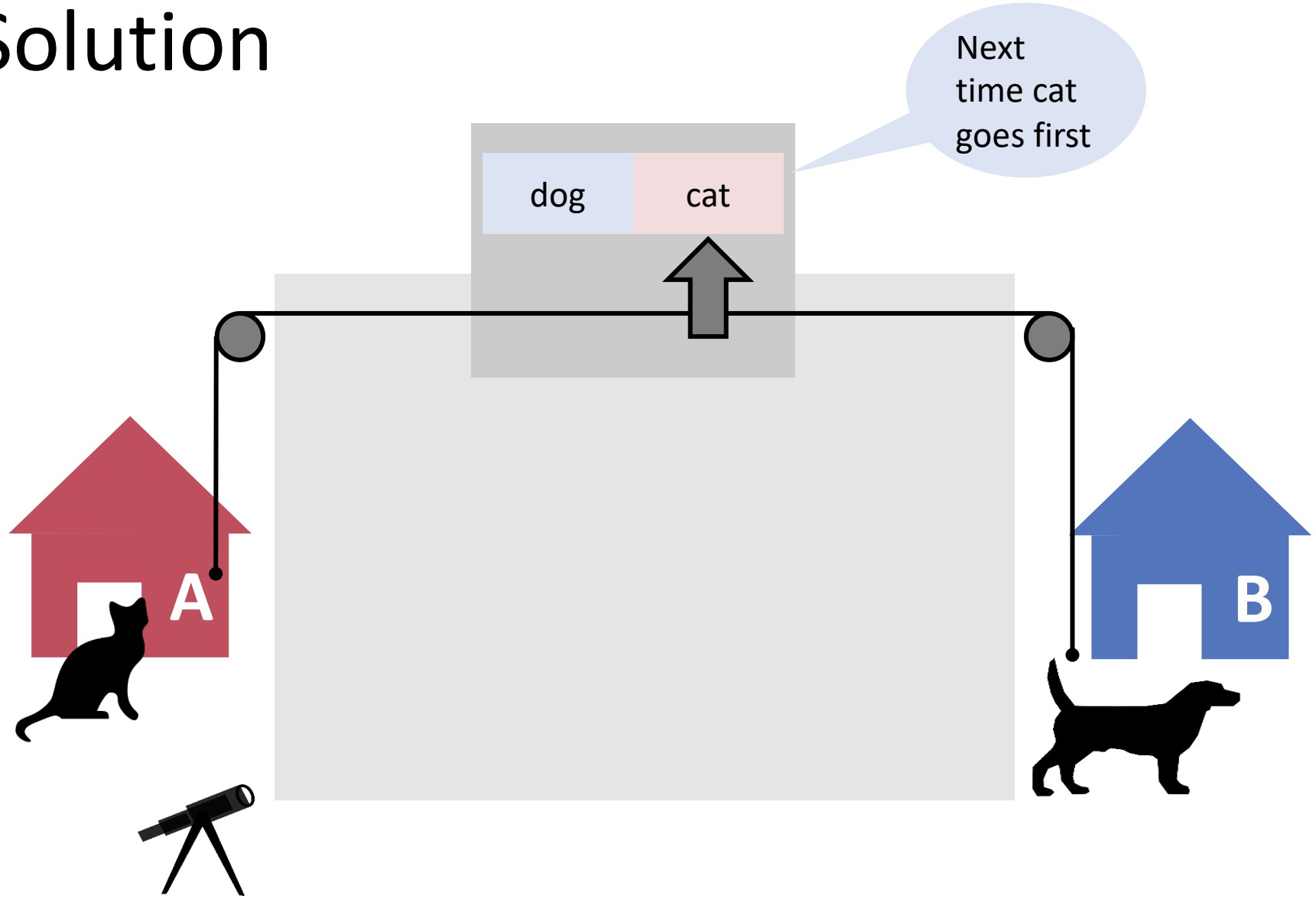
Final Solution



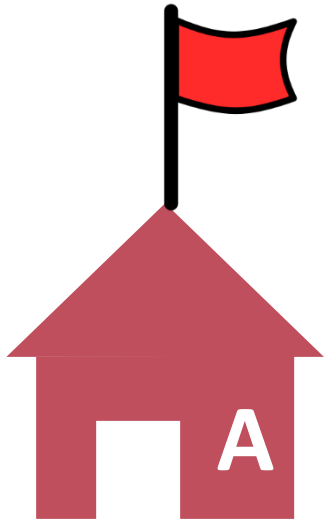
Final Solution

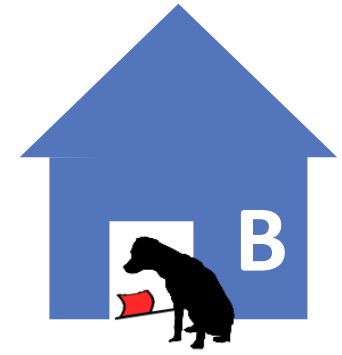


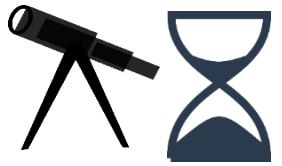
Final Solution



Still: General Problem of Waiting ...



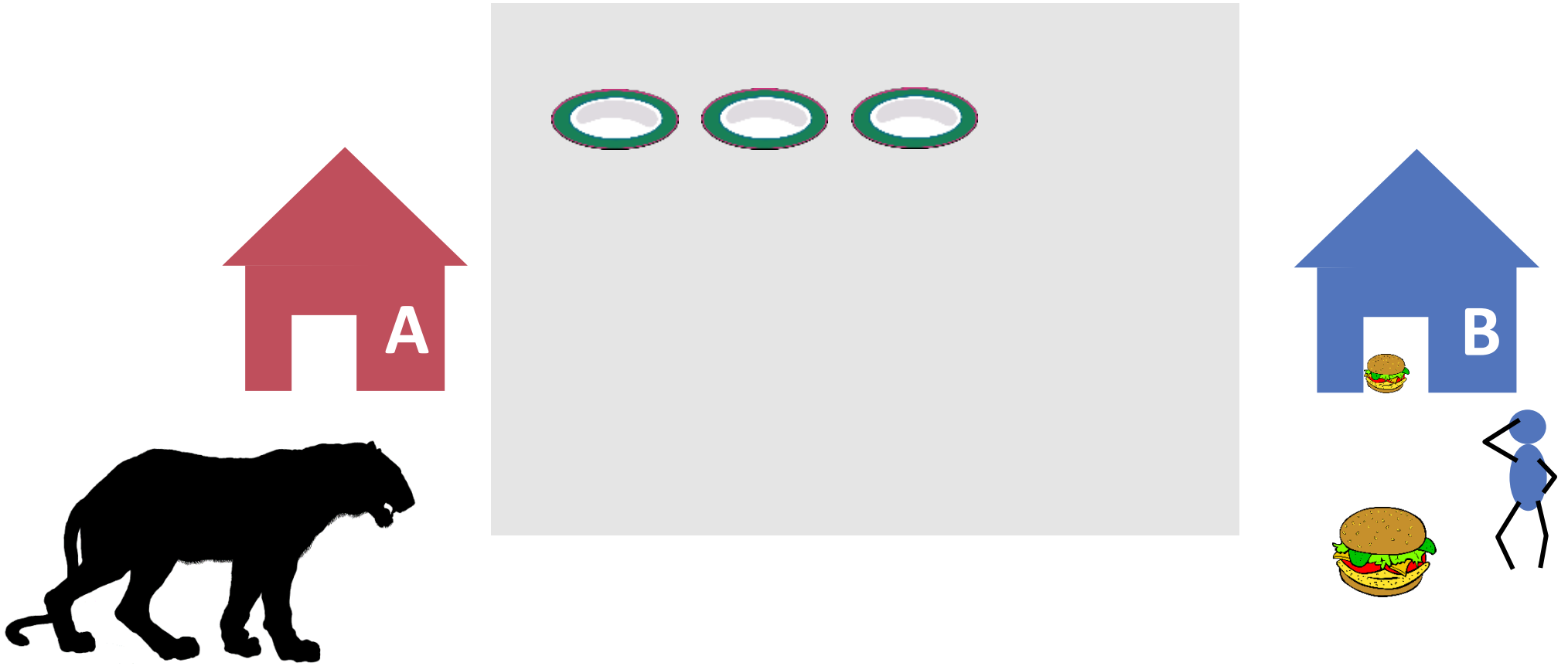




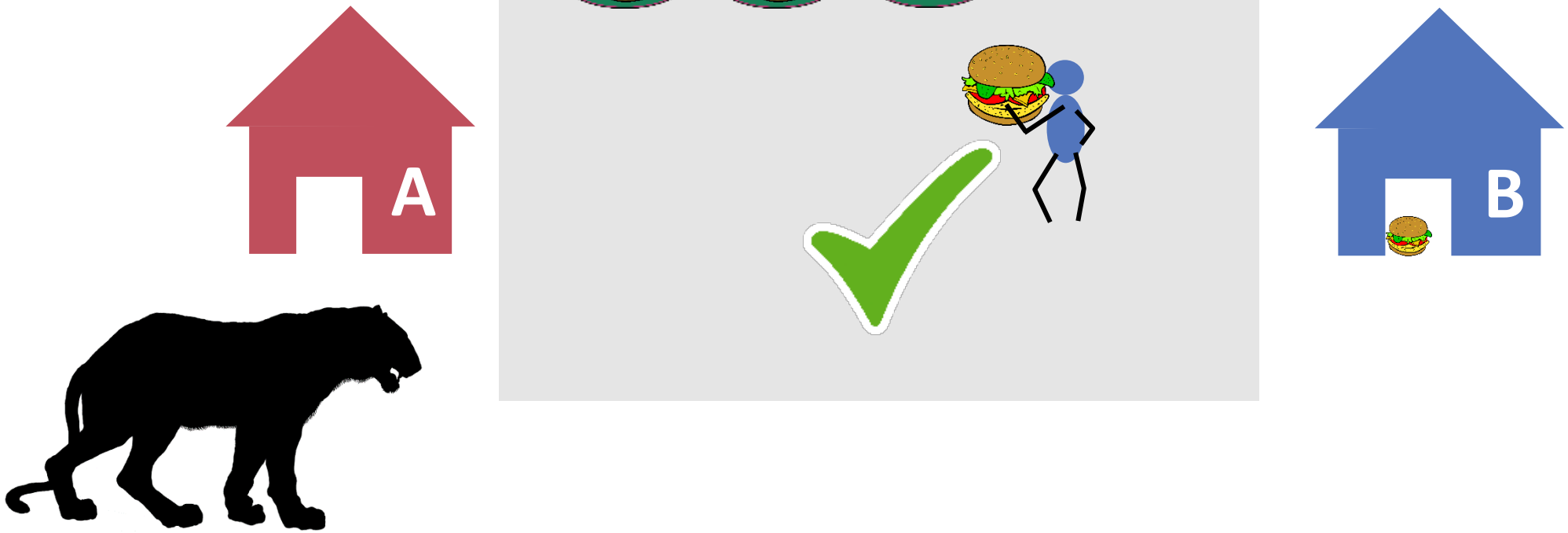
Three stories

2. PRODUCER-CONSUMER

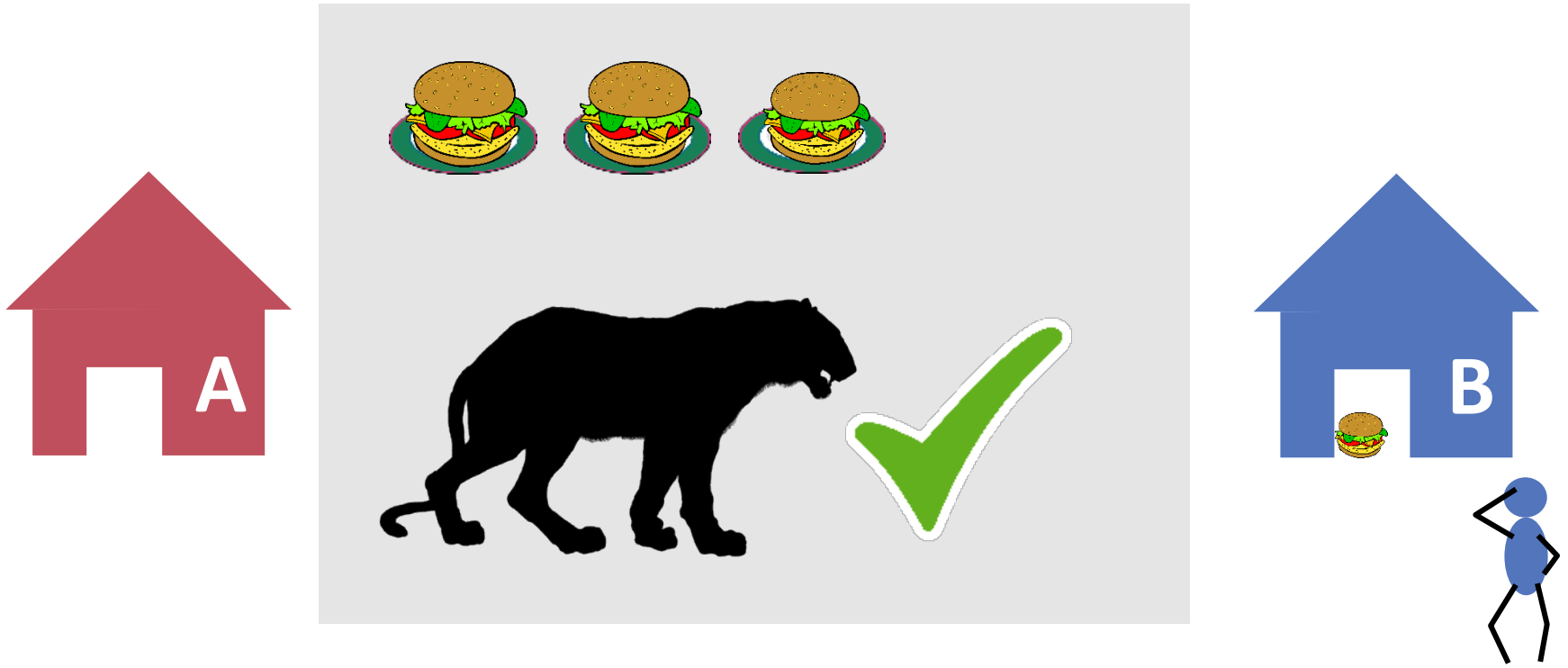
Producer-Consumer

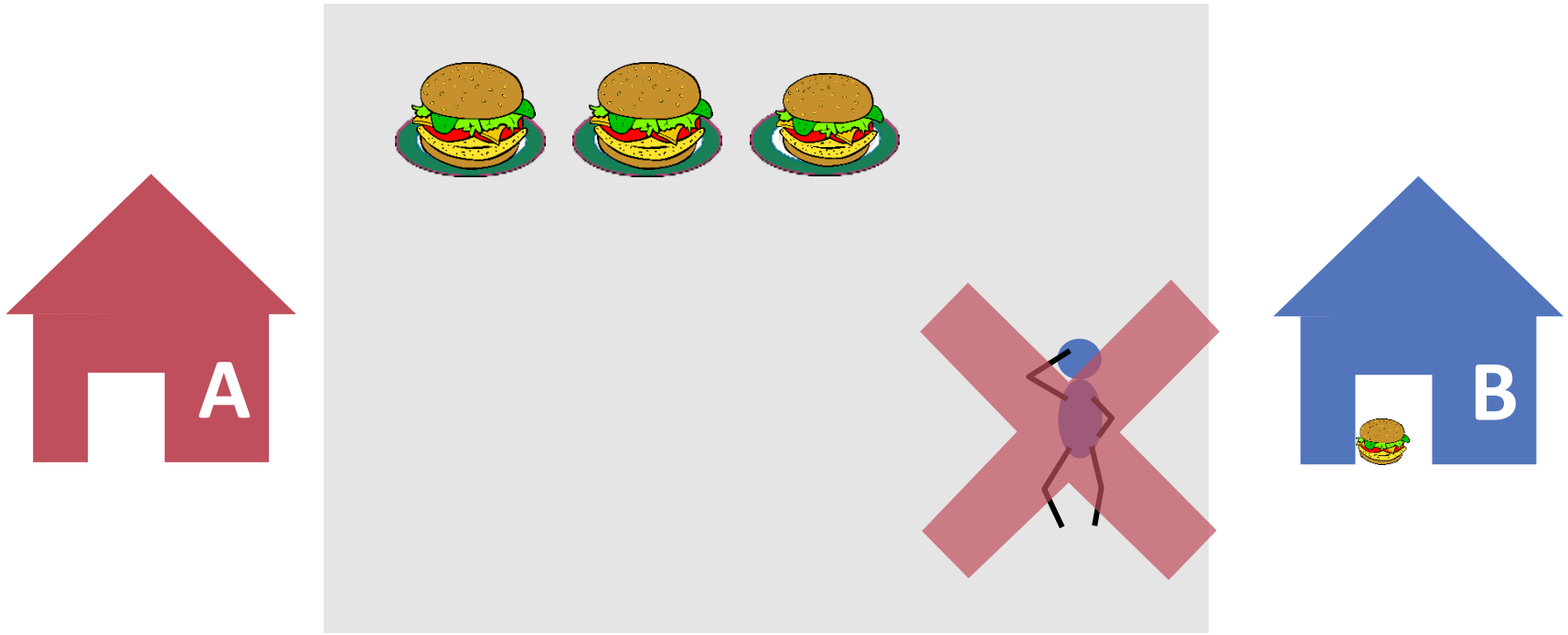


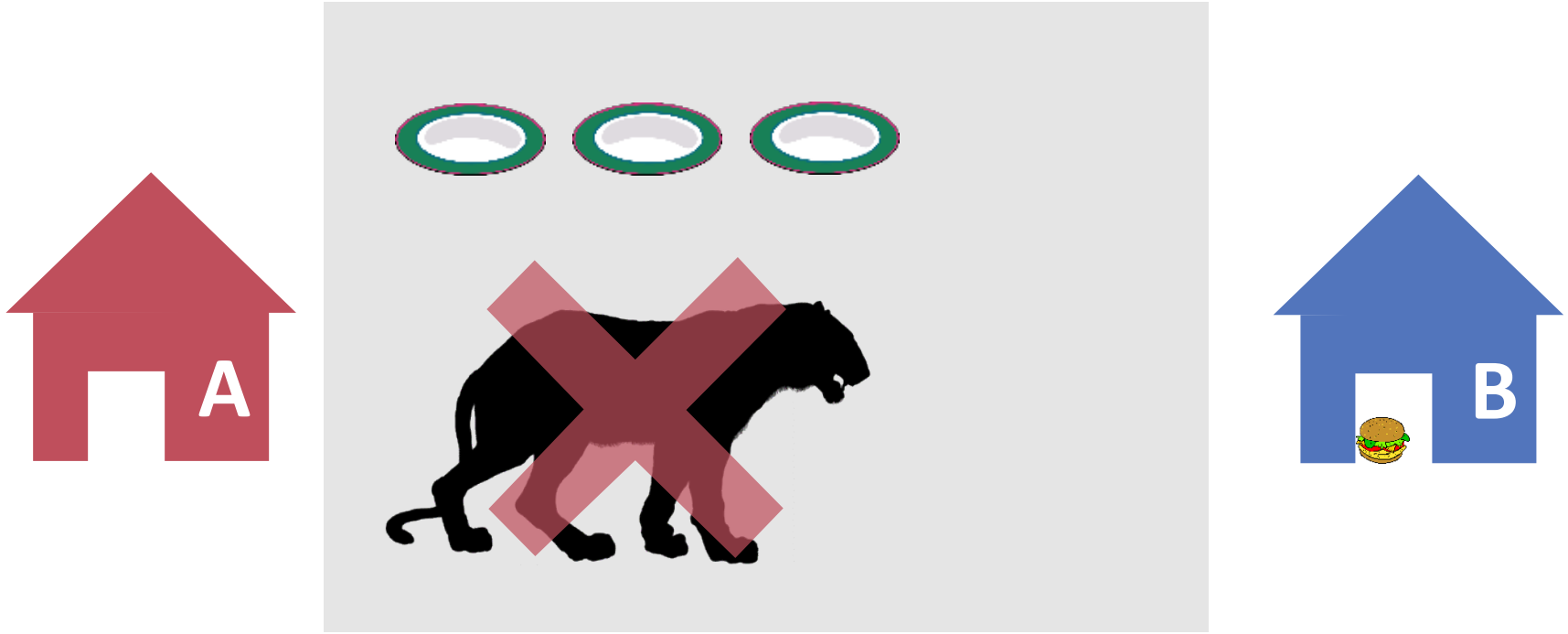
Producer-Consumer



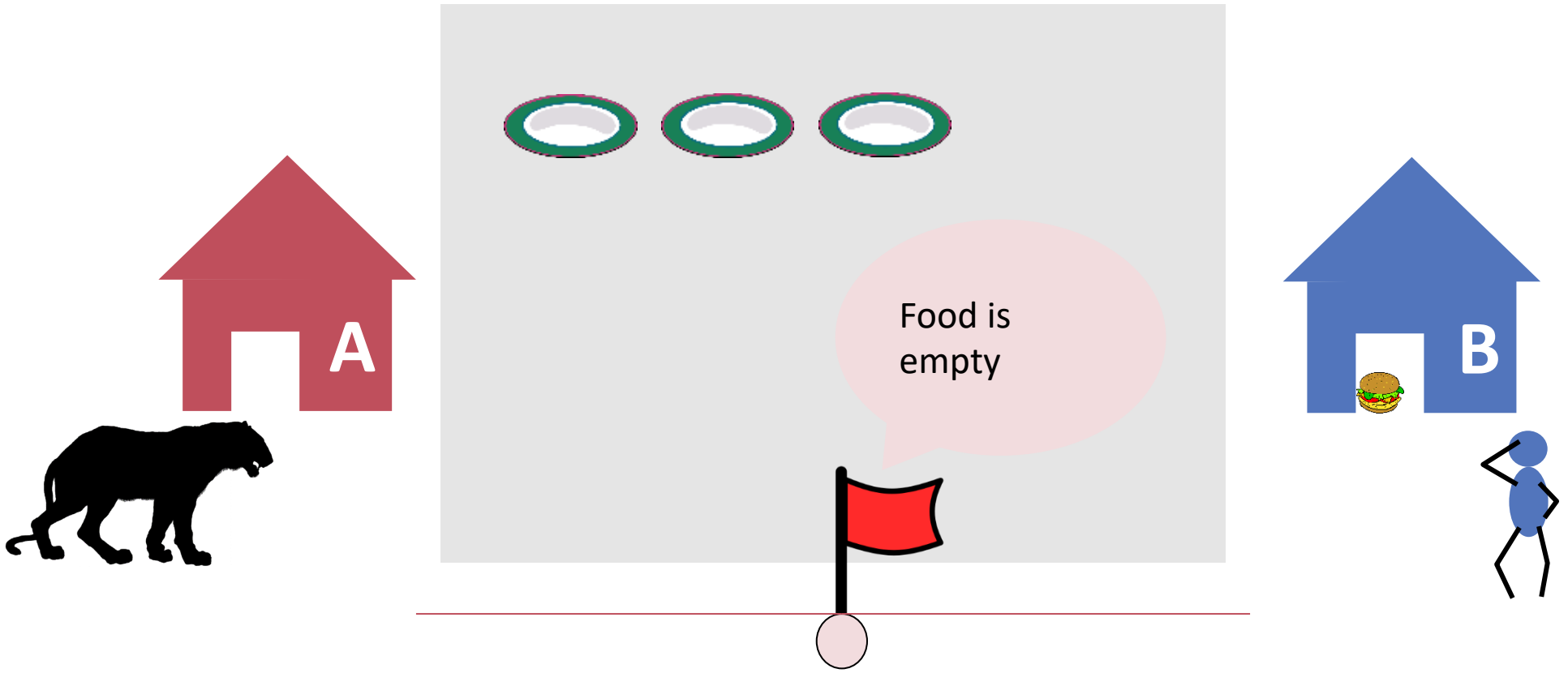
Rules



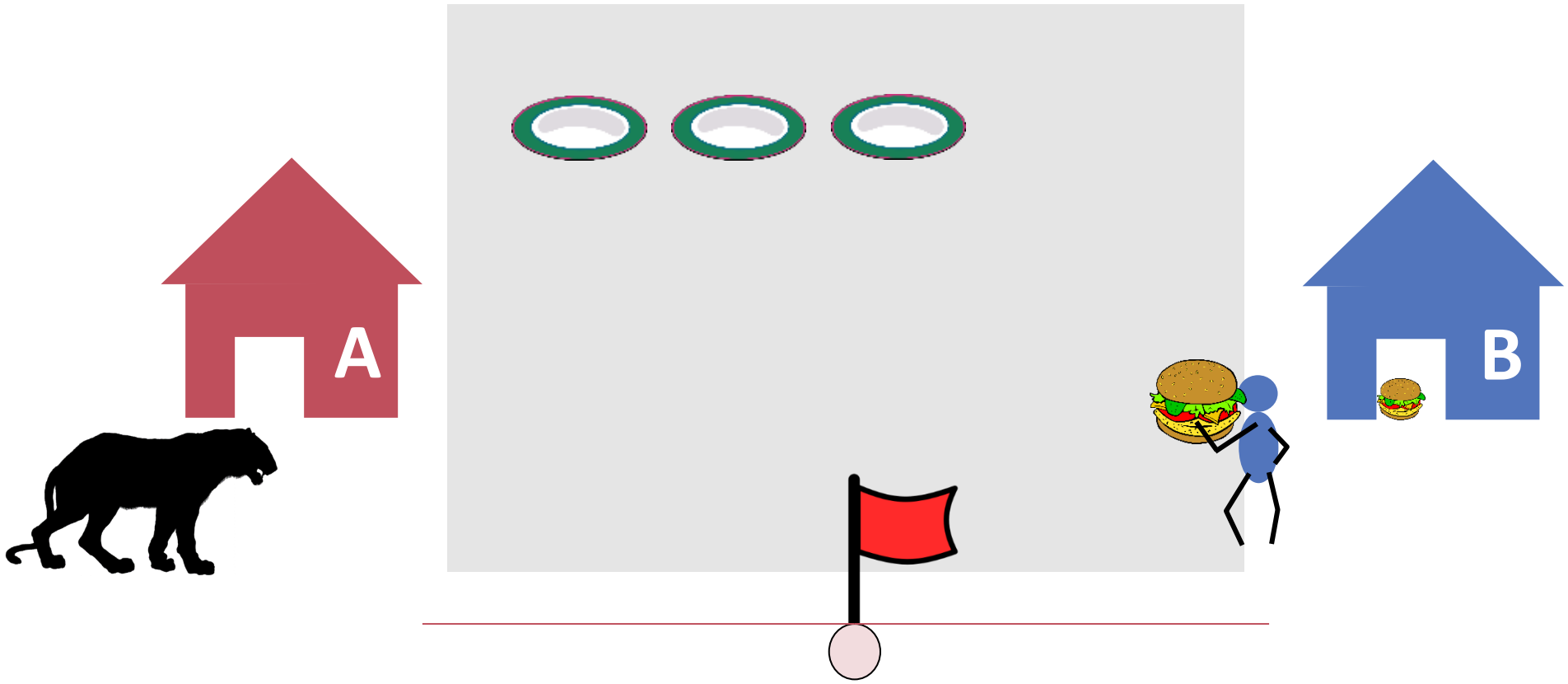

A
B


A
B

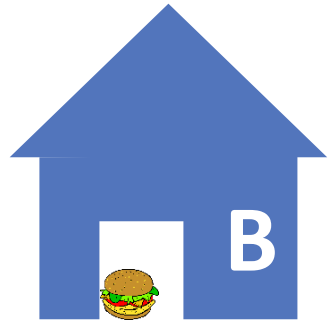
Communication

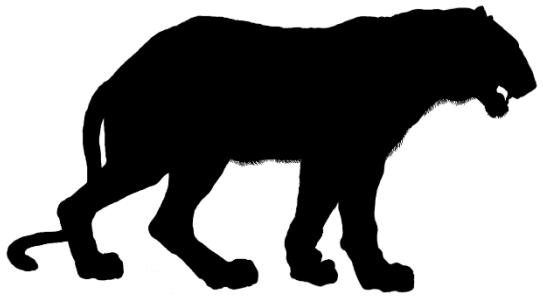


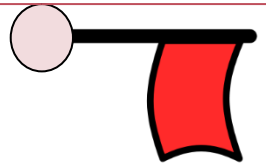
Protocol

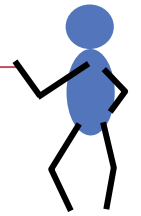


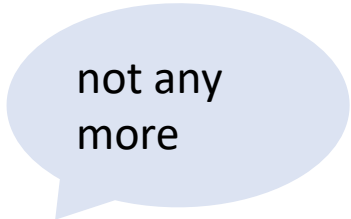

A

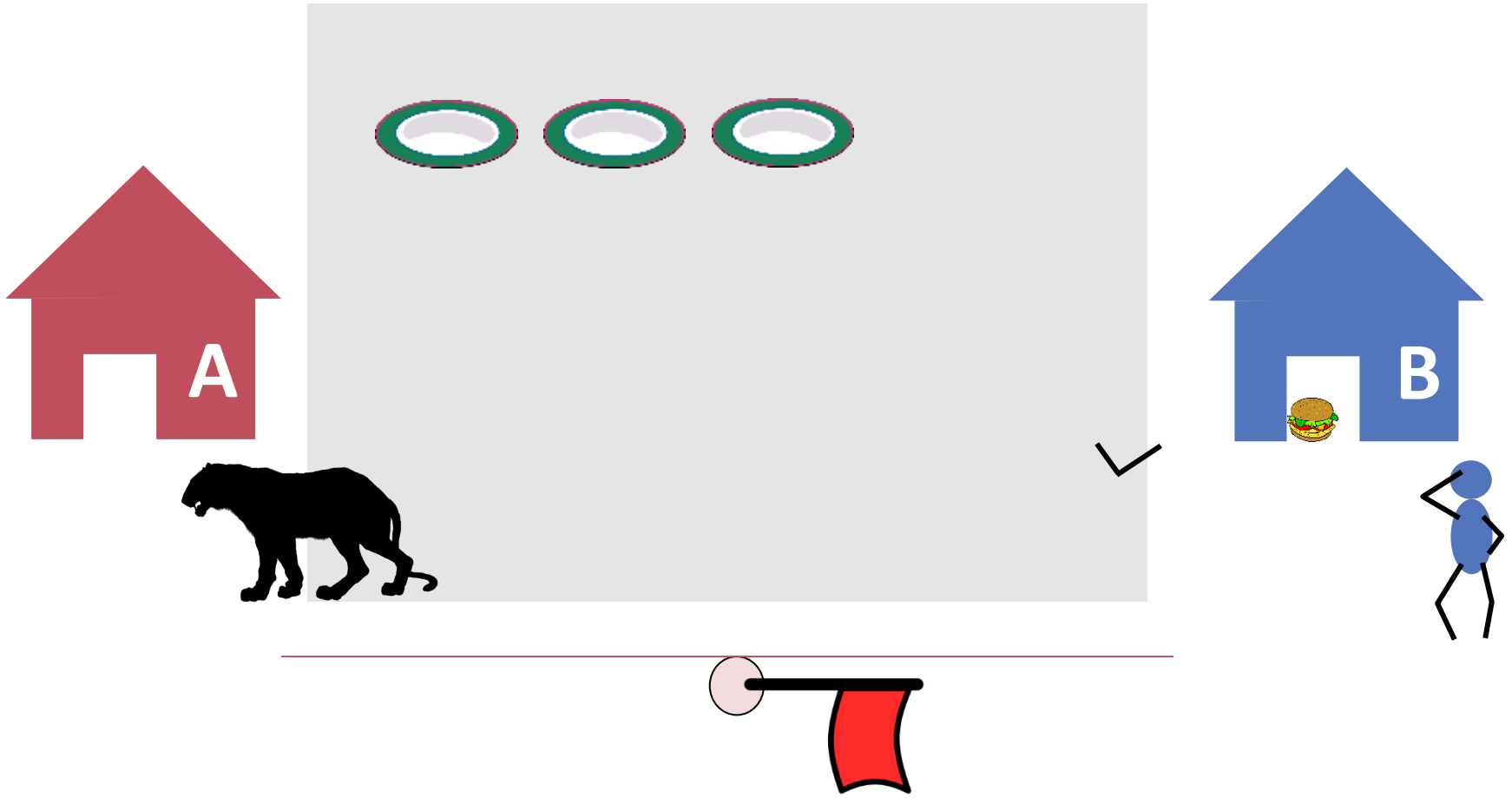

B

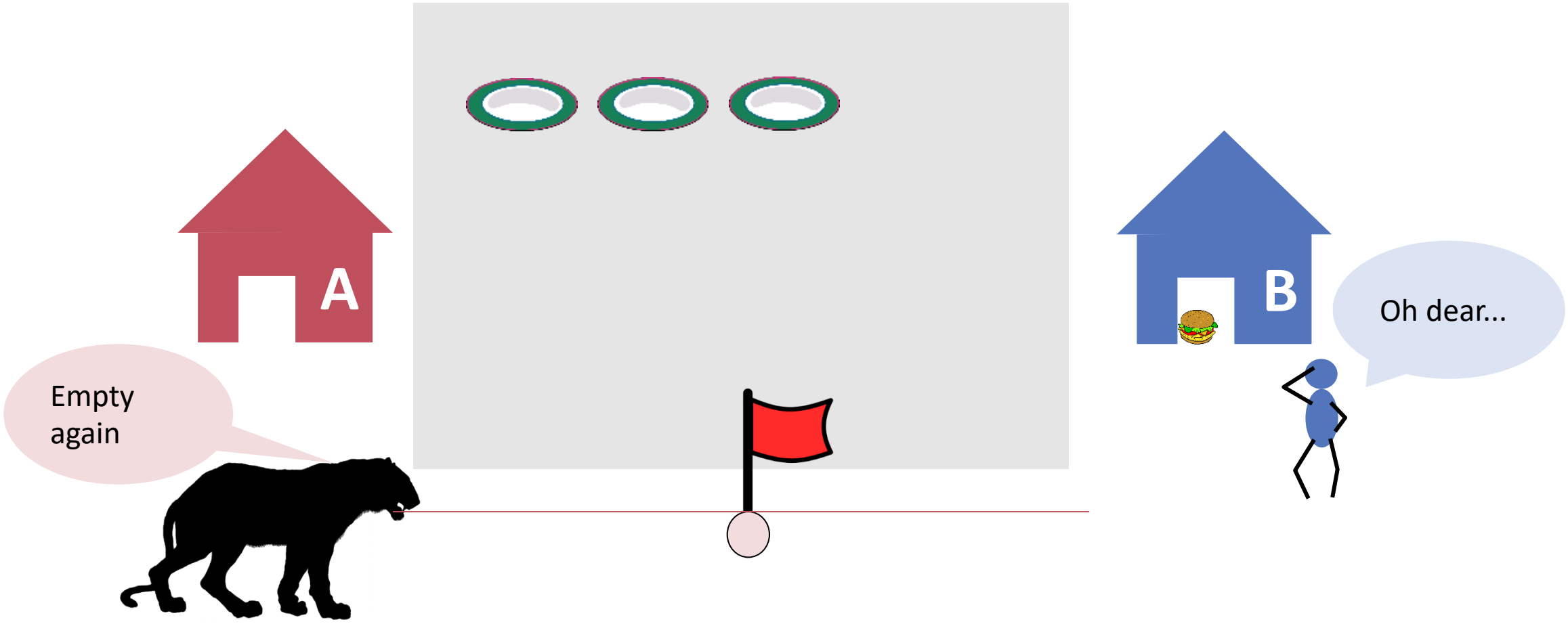






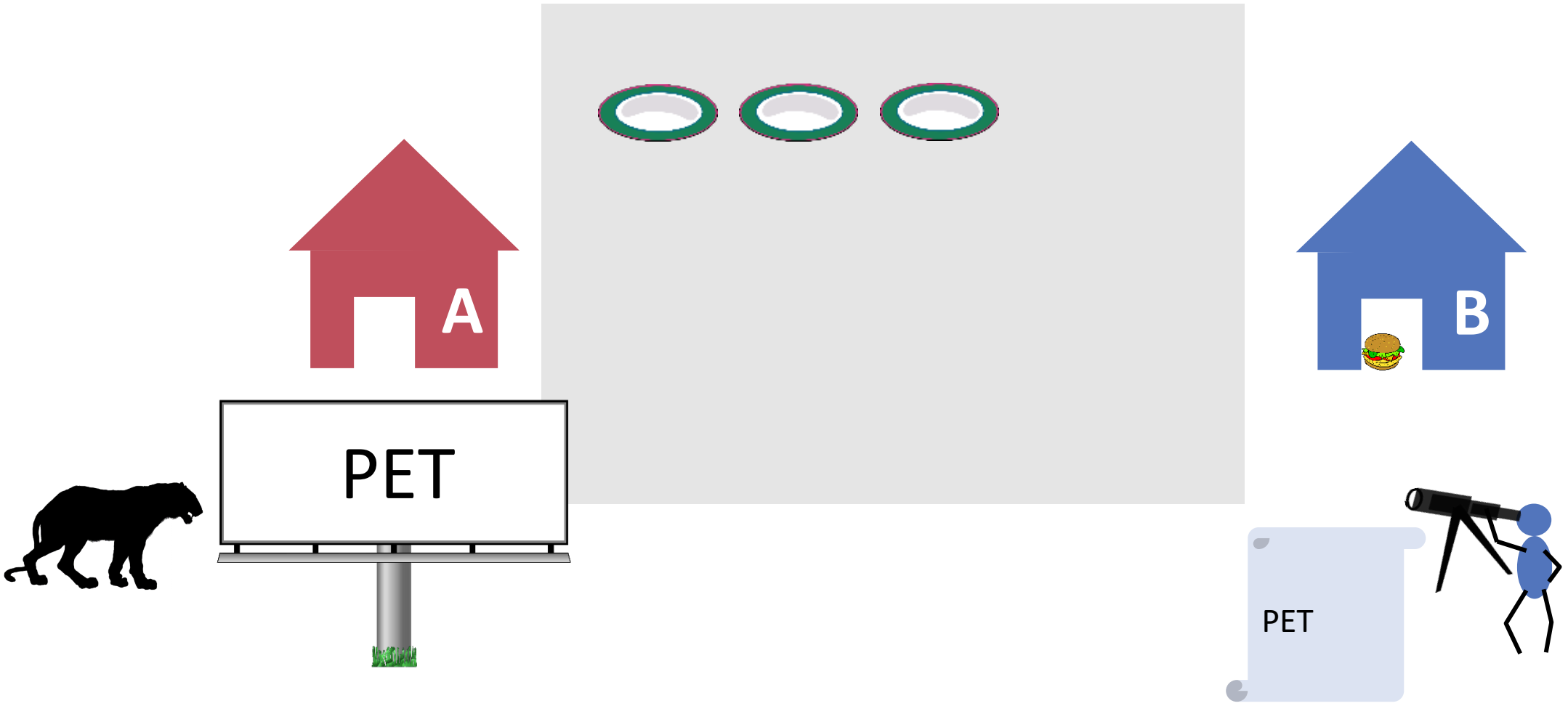

not any
more

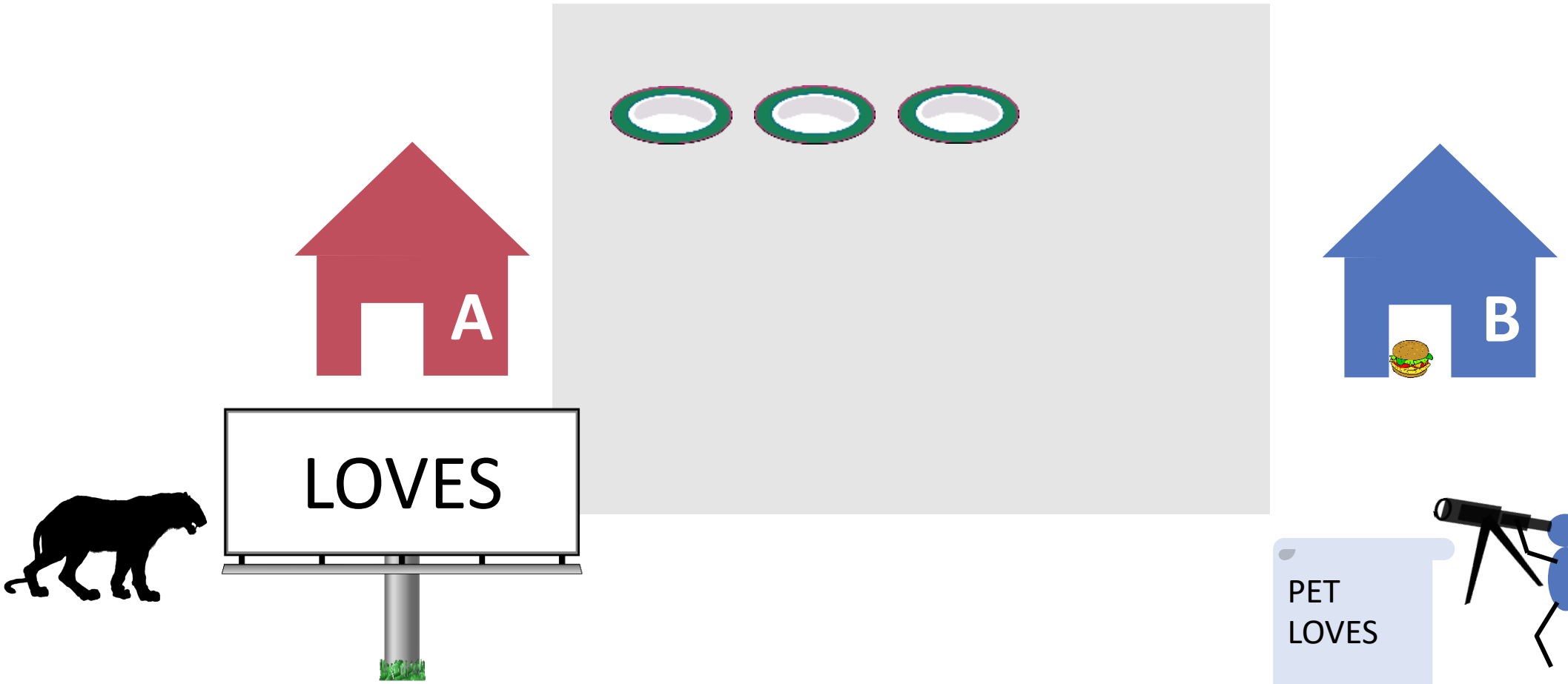

A
B

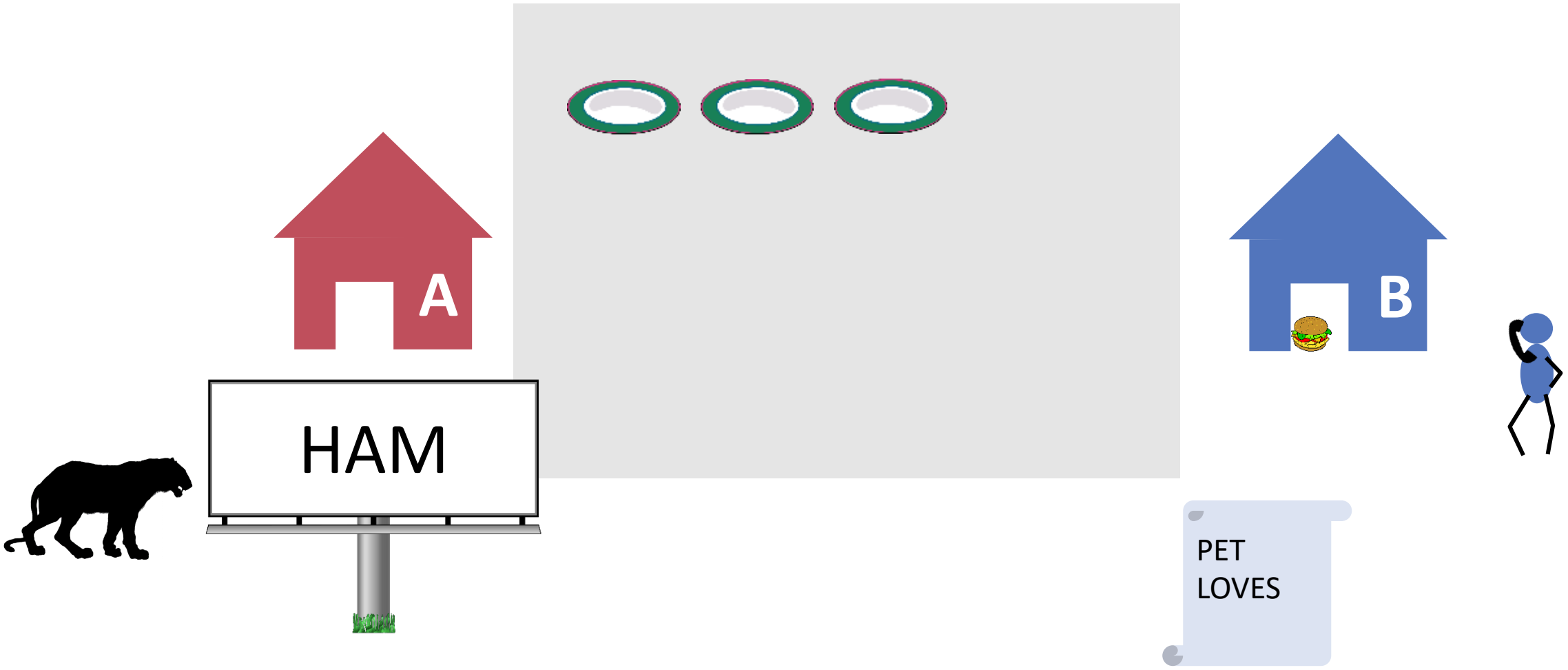

A
B
Empty again
Oh dear...

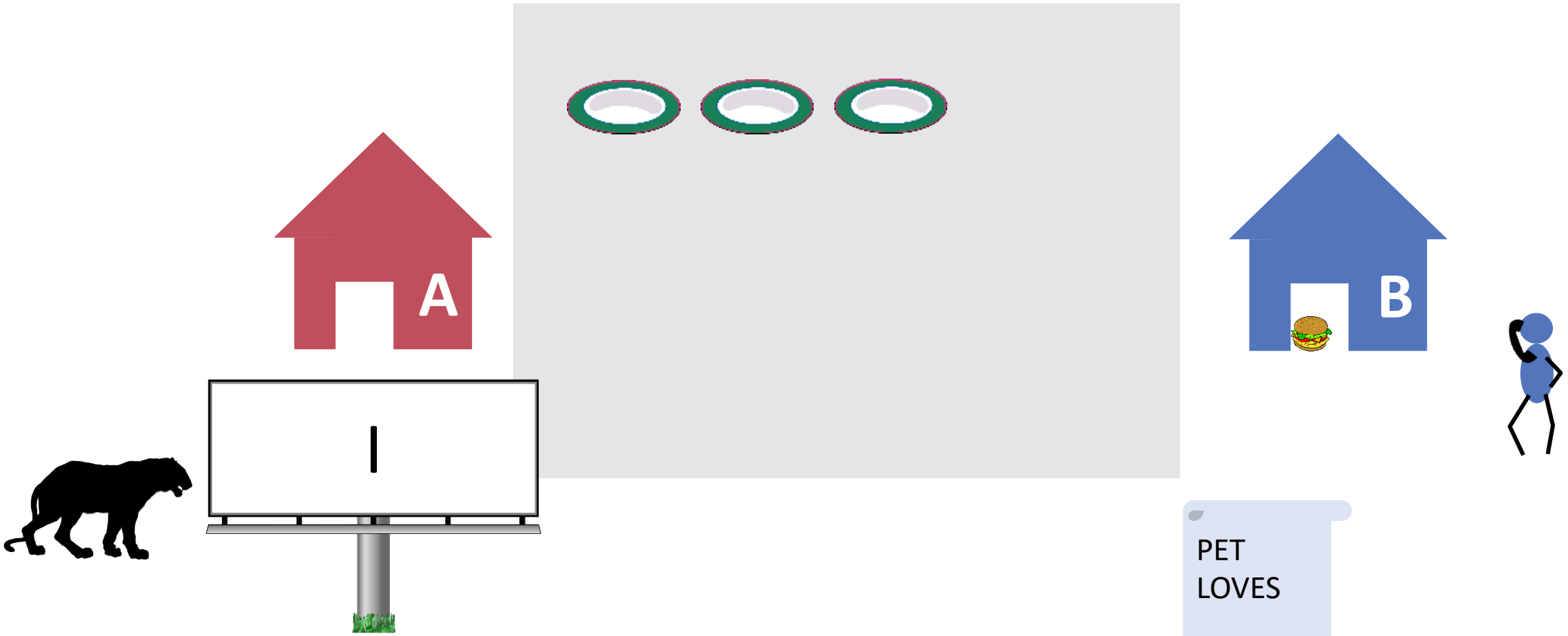
Three stories

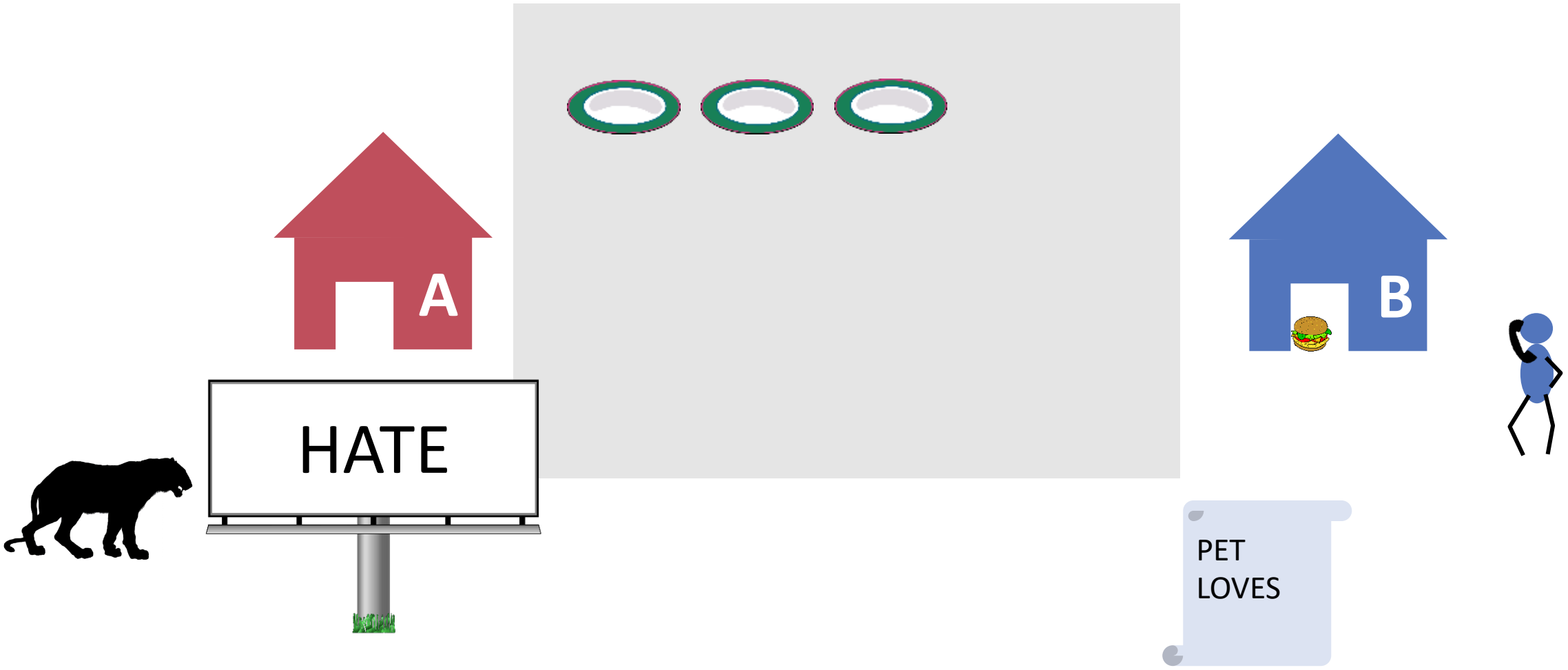
3. READERS-WRITERS

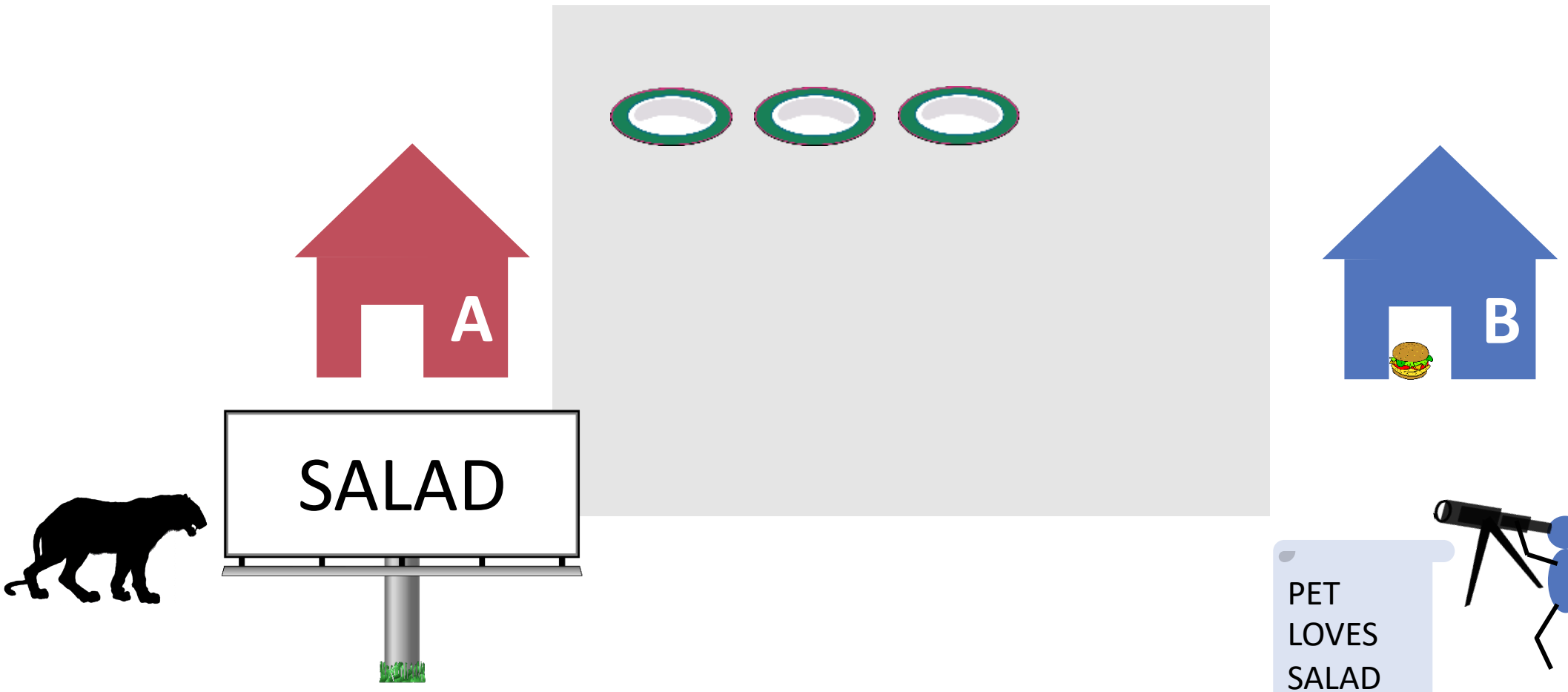

A
B
PET
PET


A
B
LOVES
PET
LOVES


A
B
HAM
PET
LOVES


A
B
I
PET
LOVES


A
B
HATE
PET
LOVES


A
B
SALAD
PET
LOVES
SALAD

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 or via forum
- 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

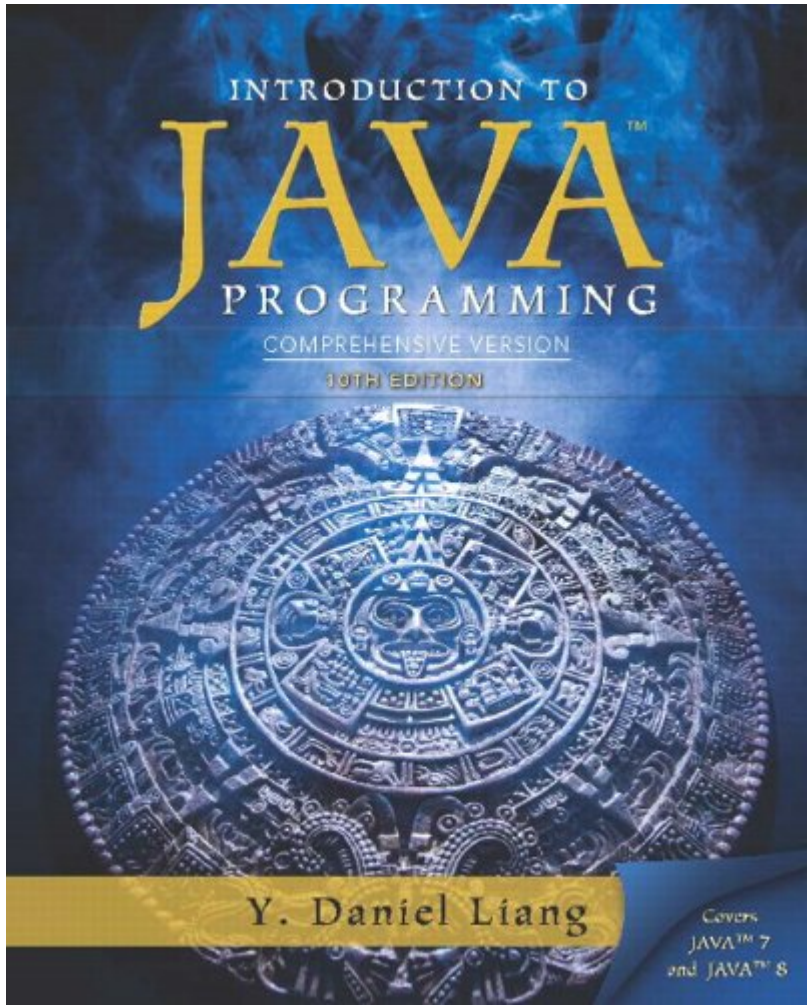
Hence, exercise sessions are

- 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

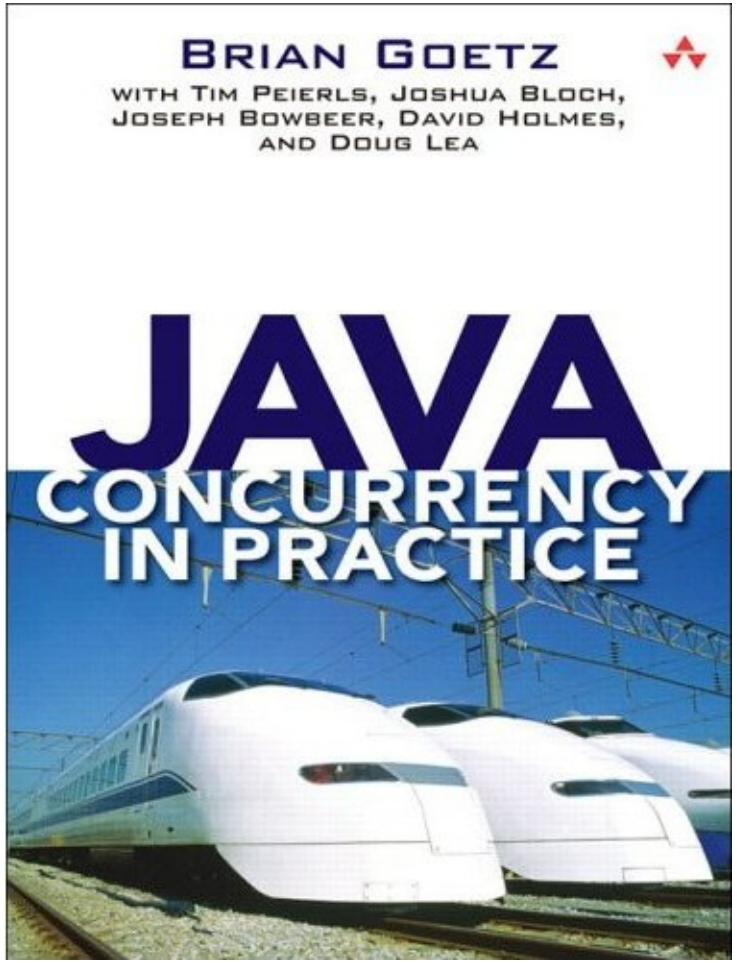
- Chapters 1-13 (with some omissions)
- Week 1-3



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