Lecture 02
Systems Thinking 1

Andrew Sonta

CIVIL 534: Computational systems thinking for sustainable engineering

28 February 2024

Learning goals

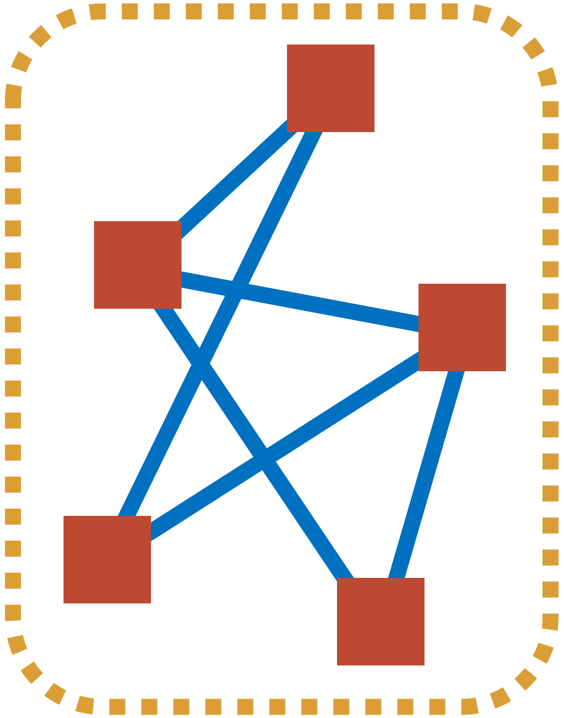
- Introduction to systems thinking and system dynamics
  - Stocks and flows
  - Behavior and system dynamics
  - Feedback loops
- Implementing systems models in Python
  - Object-oriented programming
  - In-class exercises

Our definition of a system

A system is…

a set of elements | interconnected coherently organized | in a way that achieves something

- Multiple parts without interdependencies are just collections
- The structure helps to drive the system toward its purpose

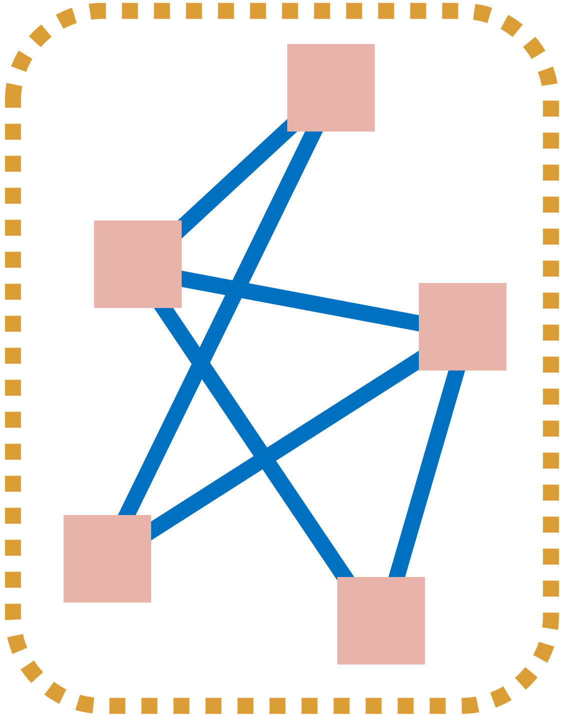


Systems definition: Meadows, 2008

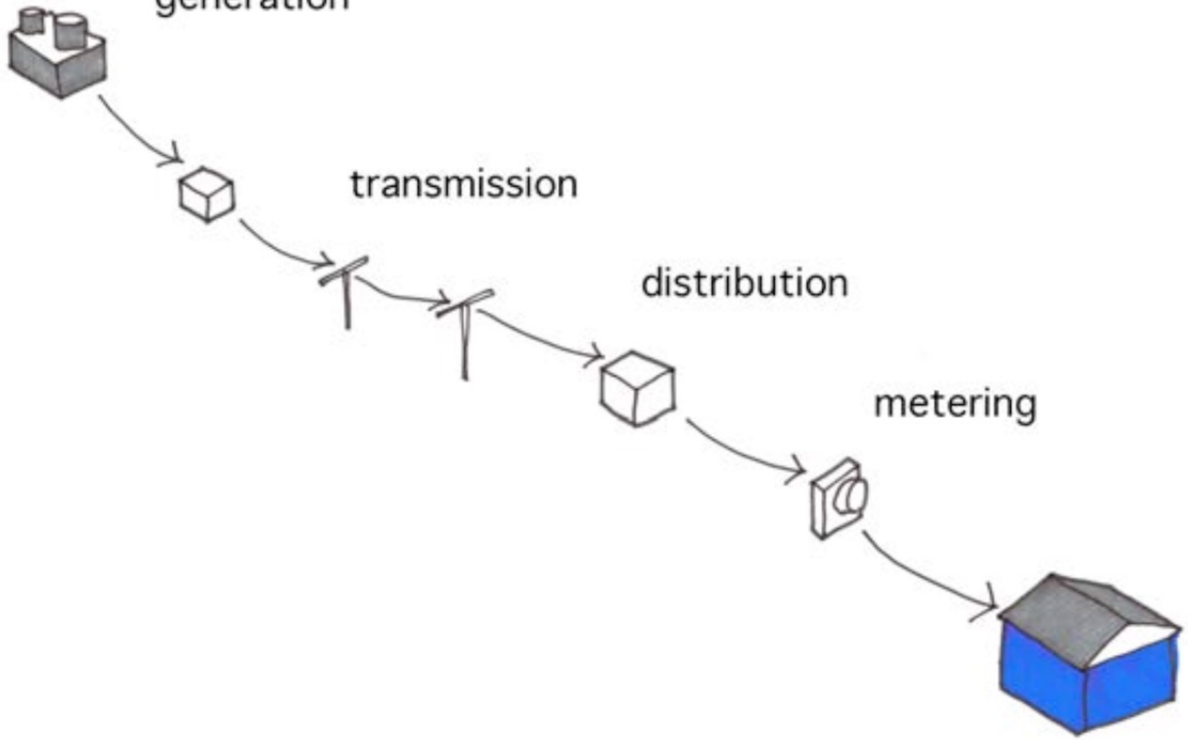
Systems thinking

In systems thinking, we:

- Take a **top-down perspective** rather than bottom-up
  - **Holistic** rather than compartmental
- Focus on **relationships** rather than the individual parts
  - Individual components only have **meaning** in the context of the whole system



Example from last time: Urban energy grid



Parts?
Impact each other?
Aggregate impact?
Persist?

Source: David Hsu, MIT

System or collection?

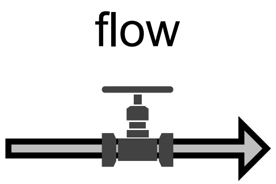
- Closet full of clothes
- The human body
- Football team
- Toaster
- Database of customer information
- Tools in a toolbox

Some comments on "Thinking in Systems"

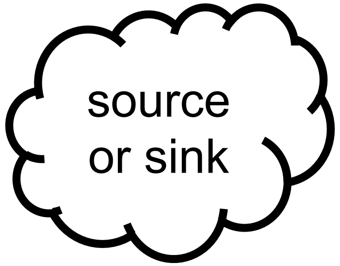
- The math is easy
- The way of thinking is not
  - Generating your own models of complex systems is a difficult task
- Our collective challenge:
  - Fully grasping the lessons in how to think systemically
  - Applying the framework to engineered urban systems
  - Generating and analyzing useful computational models via this framework

Systems nomenclature

The quantities we care most about modeling

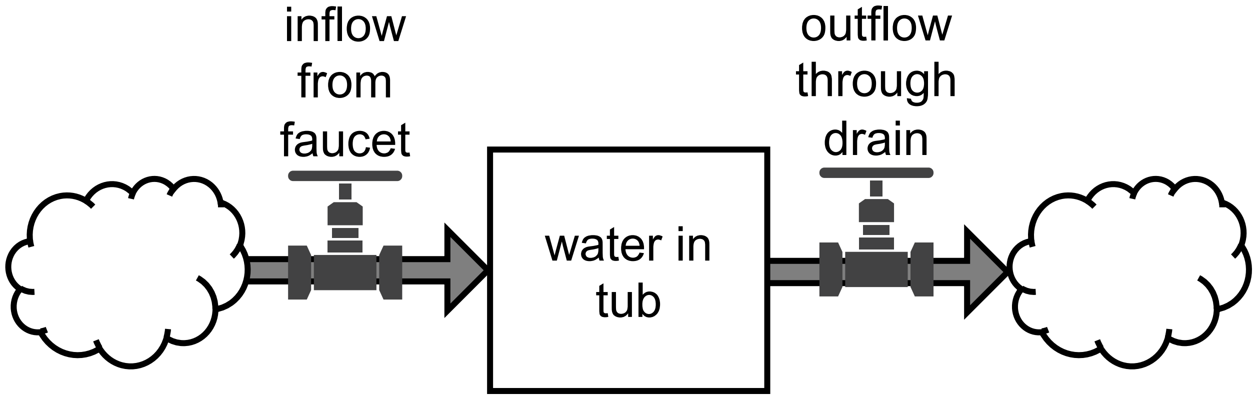


Flows cause changes in quantities of stocks

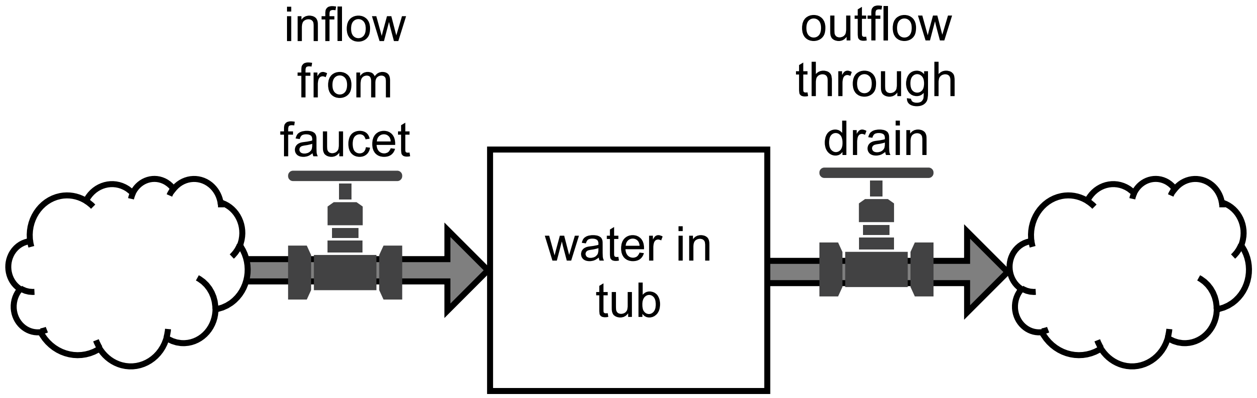


Quantities flow in and out of system boundaries,
from sources and *to* sinks

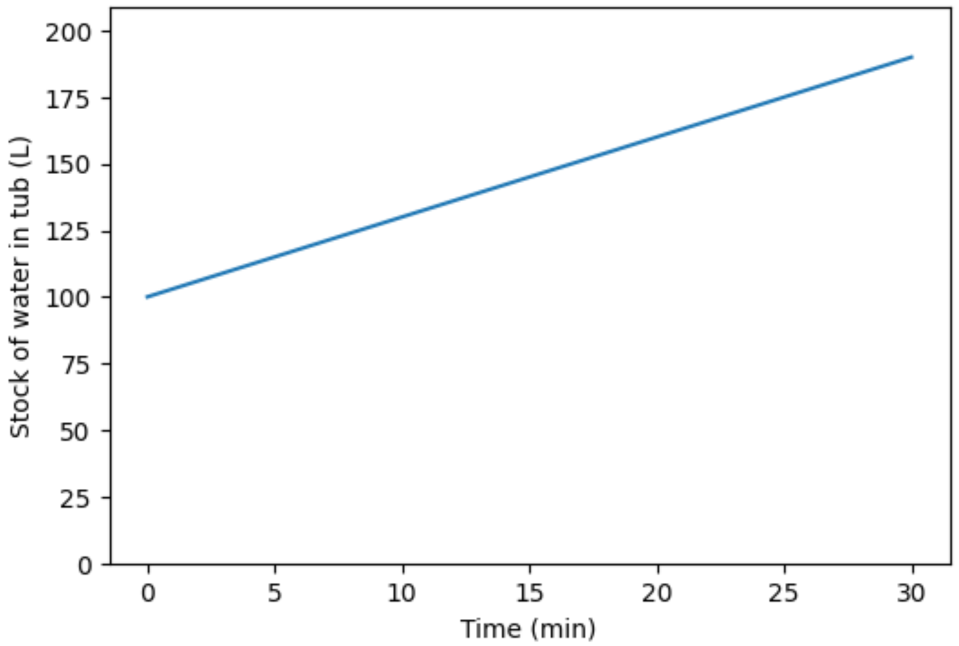
A simple system



From systems models to system dynamics

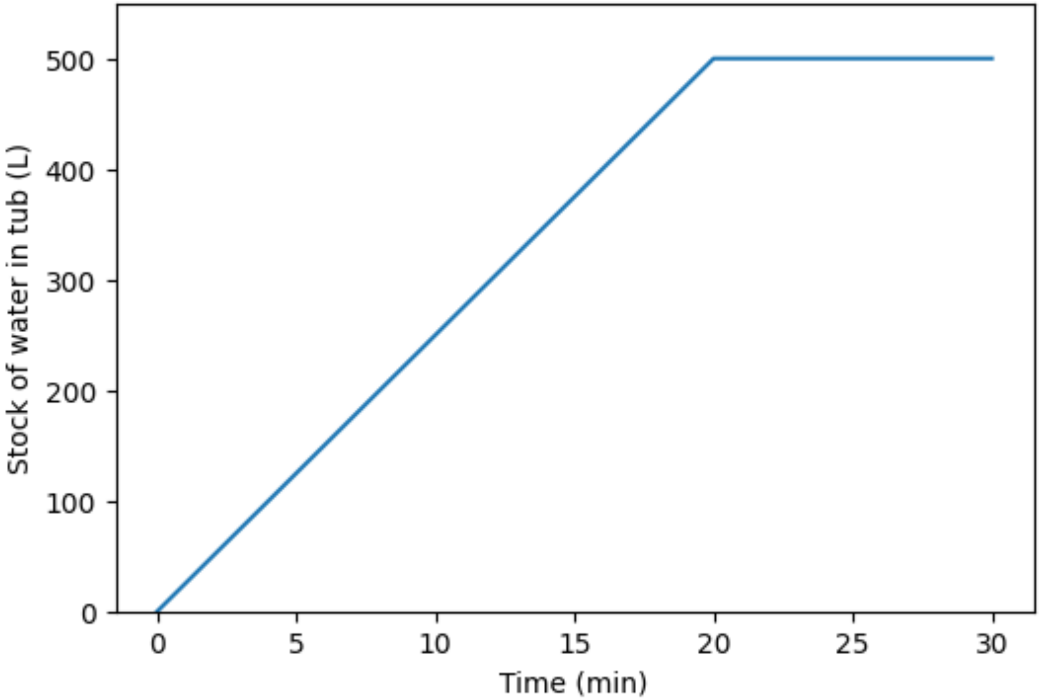


stock of water in tub: 100L
inflow: 5 L/min
outflow: 0 L/min



Dynamic equilibrium

stock = 0 (assume it starts empty)
inflow = 25 L/min
outflow = 0 if stock < 500 L
25 L/min if stock ≥ 500 L



Computing system dynamics

- The goal: model the system dynamics in Python
- The plan: use object-oriented programming (OOP) for implementation
- We will work through this example together

Python fundamentals: Object-oriented programming (OOP)

- Easier to maintain
- Reduces code repetition
- Enables you to easily add new functionality to existing code

Class creation

Class attribute

Class method

Object creation

```python
class Dog:
    species = "Canis familiaris"
    def __init__(self, breed, name):
        self.breed = breed
        self.name = name

buddy = Dog("poodle", "buddy")
```

`self` refers to the specific instance of the class

Code for course project is implemented with OOP principles

```
class World2:
    """
    World2 class contains helpers to configure and run a simulation. Defaults
    parameters leads to a standard run.

    Examples
    --------
    >>> w2 = World2()                 # possibly modify time limits and step
    >>> w2.set_state_variables()      # possibly modify the model constants
    >>> w2.set_initial_state()        # possibly modify the condition constants
    >>> w2.set_table_functions()      # possibly do your own tables in a json file
    >>> w2.set_switch_functions()     # possibly choose switches in a json file
    >>> w2.run()                      # run the simulation
```

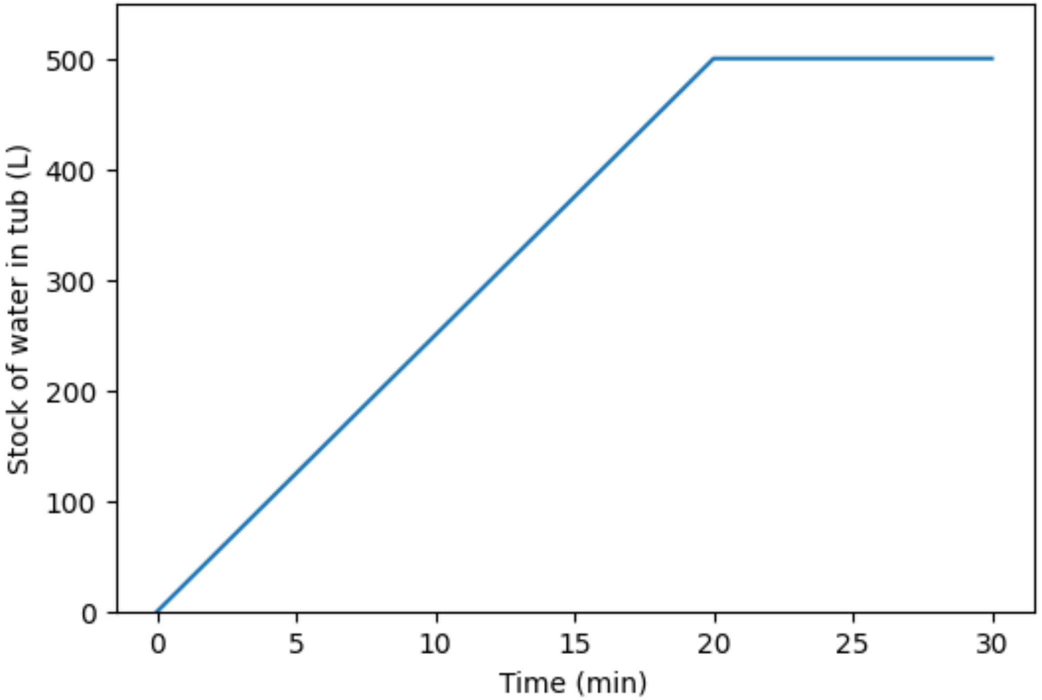
```
def __init__(self, year_min=1900, year_max=2100, dt=0.2):
    """
    __init__ of class World2.

    Parameters
    ----------
    year_min : int, optional
        starting year of the simulation. The default is 1900.
    year_max : int, optional
        end year of the simulation. The default is 2100.
    dt : float, optional
        time step of the numerical integration [year]. The default is 0.2.

    """
    self.year_min = year_min
    self.year_max = year_max
    self.dt = dt
    self.time = np.arange(self.year_min, self.year_max + self.dt, self.dt)
    self.n = self.time.size
```

Dynamic equilibrium

stock = 0 (assume it starts empty)
inflow = 25 L/min
outflow = 0 if stock < 500 L
25 L/min if stock ≥ 500 L



Stock and flow implications

- It is easier to **quickly change a flow** than it is to **quickly change a stock.**
- Stocks are constant when: $\frac{df_{in}}{dt} = \frac{df_{out}}{dt}$
- Stocks can act as **buffers** to changes in flows when:
  - Flows are not drastically different: $\frac{df_{in}}{dt} \approx \frac{df_{out}}{dt}$
  - Stocks are large relative to flows $\frac{df}{dt} \ll s(t)$

➢ Stocks allow inflows and outflows to be decoupled and to be independent and temporarily out of balance with each other.

Buffers

Trees in forest in logging system

Carbon in atmosphere

Buffers: our electricity grid example

Feedback loops

- Monitoring stocks allows decisions to be made that adjust inflows and outflows.
- Feedback loop: **When a change in a stock affects the flows into or out of that stock**

Feedback drives a system toward its purpose by relaying information on the state of the system

Two kinds of feedback loops

Balancing feedback loop

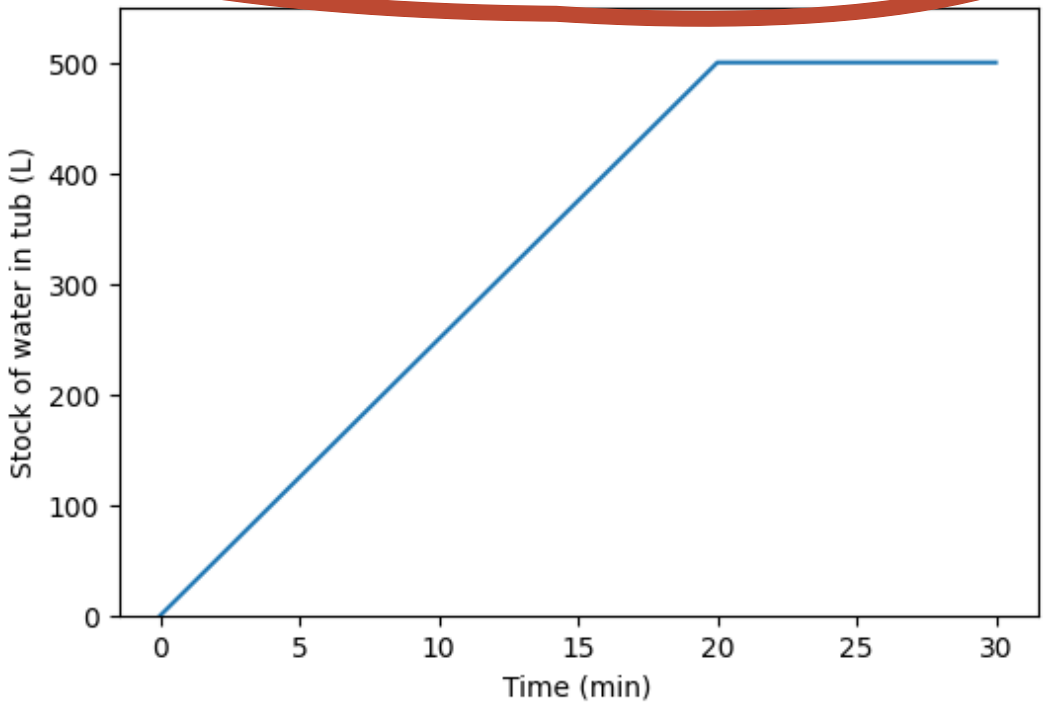
- Stabilizes the level of the stock
- Example:
  - Swiss water fountains
  - Coffee drinker

Reinforcing feedback loop

- Enhances whatever direction of change is imposed on the stock
- Example:
  - Neighborhood population
  - Albedo of the earth

Simple feedback

stock = 0 (assume it starts empty)
inflow = 25 L/min
outflow = 0 if stock < 500 L
25 L/min if stock ≥ 500 L



Feedback loops in diagrams

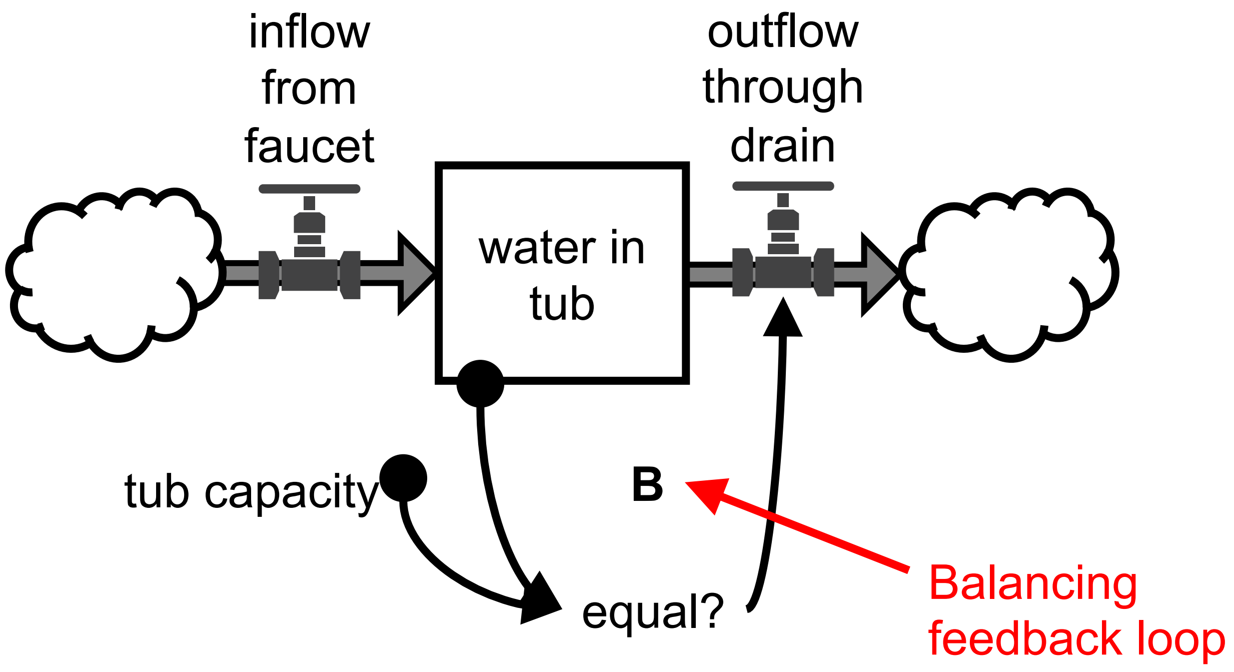
Variables

Components of the system that are not stocks or flows

Influence arrows

Demonstrate the influence of variables

Note: Variables cannot directly influence stocks, but they can influence flows



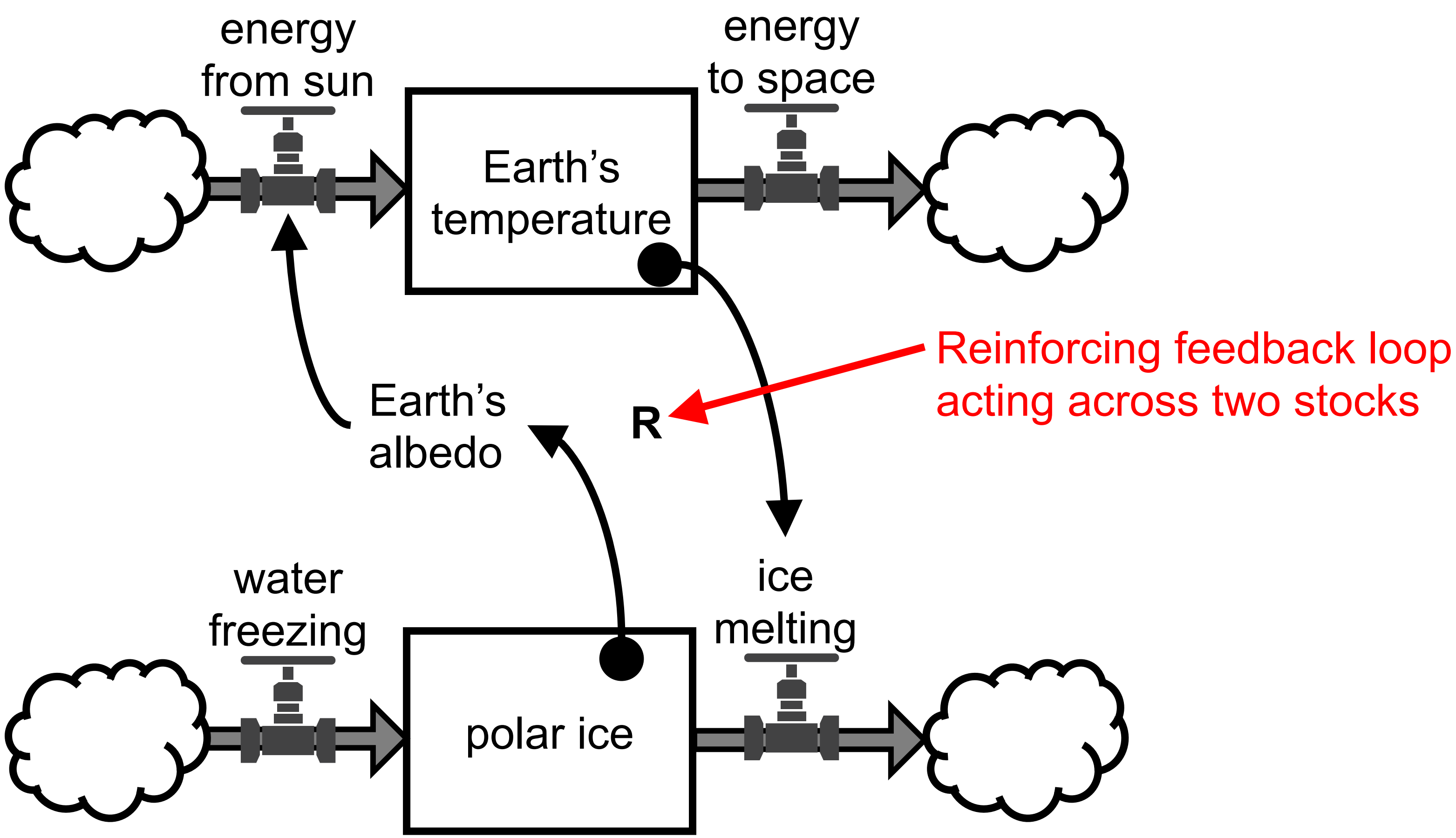
Balancing feedback example: Coffee

Balancing feedback example: Coffee

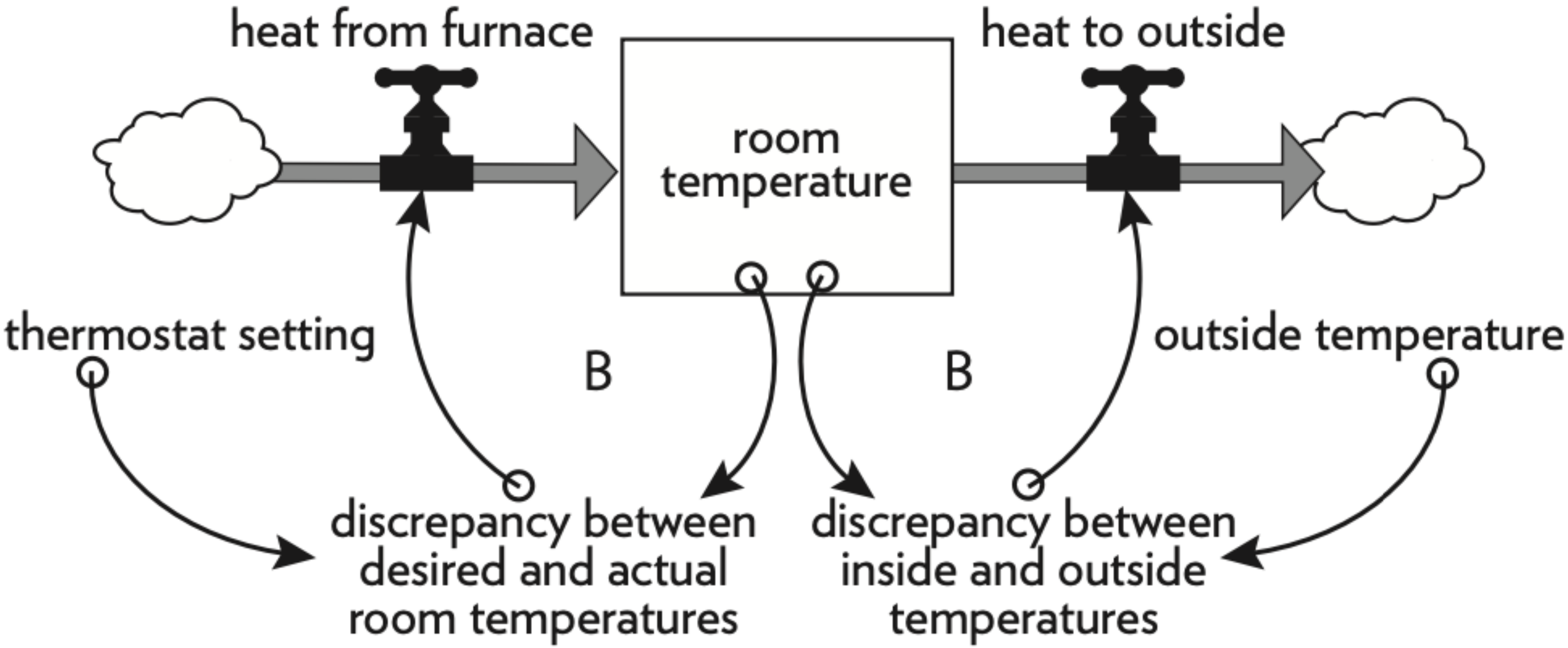
Reinforcing feedback example: Neighborhood population

Reinforcing feedback example: Neighborhood population

Reinforcing feedback example: ice and albedo



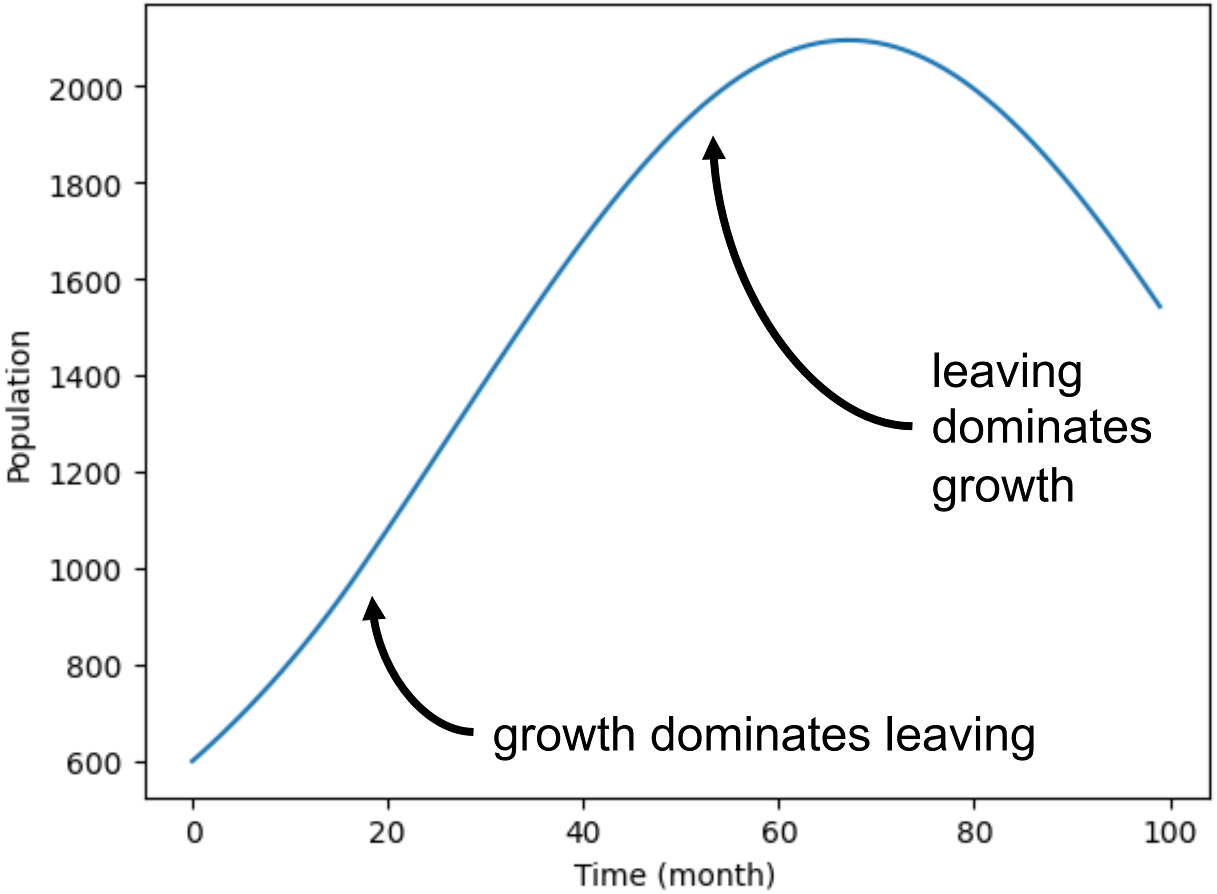
Competing feedback loops



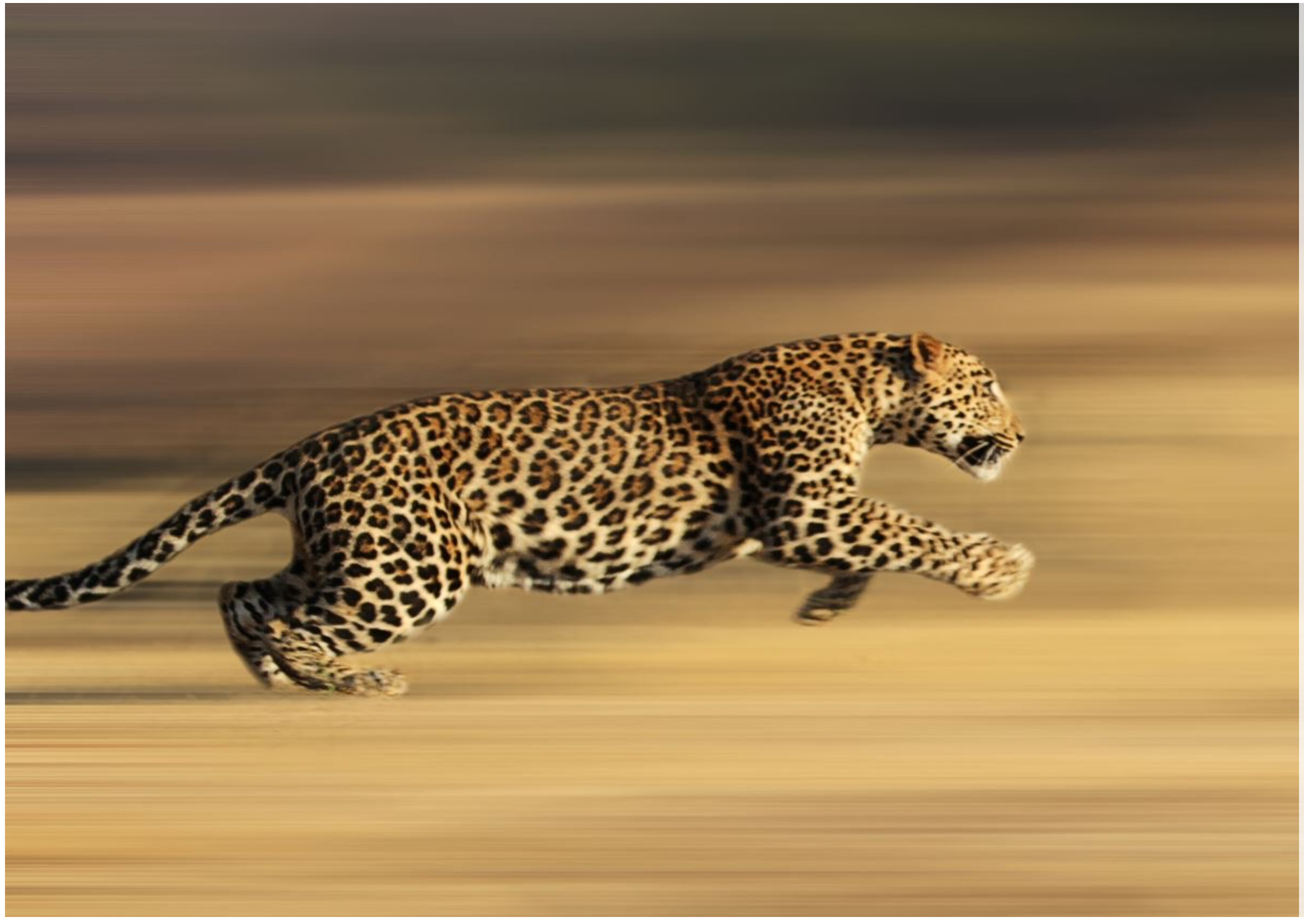
1 reinforcing and 1 balancing loop

Neighborhood population example

Shifting dominance: when the dominating feedback loops changes



Your turn
Predator-prey cycle



Stocks:
- Number of predators
- Number of prey

Feedback loops:
- 4 simple
- 1 complex

Next time

- Read Meadows Chapter 2
- Two-stock systems, delays and oscillations, etc.
- Short intervention by ENAC representative for organizational culture, Ingrid Le Duc