Planning

deciding what to do and what order to do it in

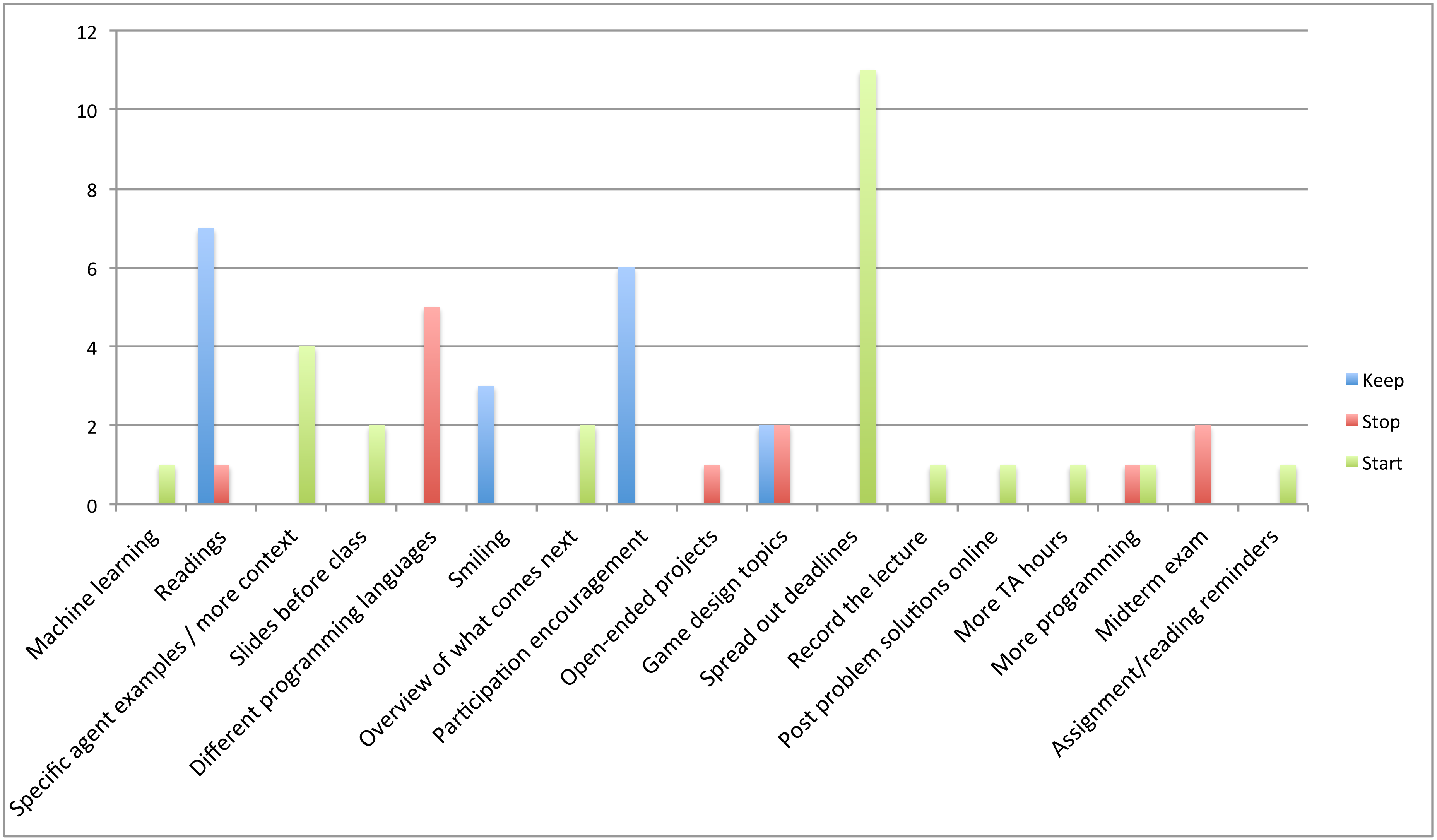
CS 4100/5100

Foundations of AI

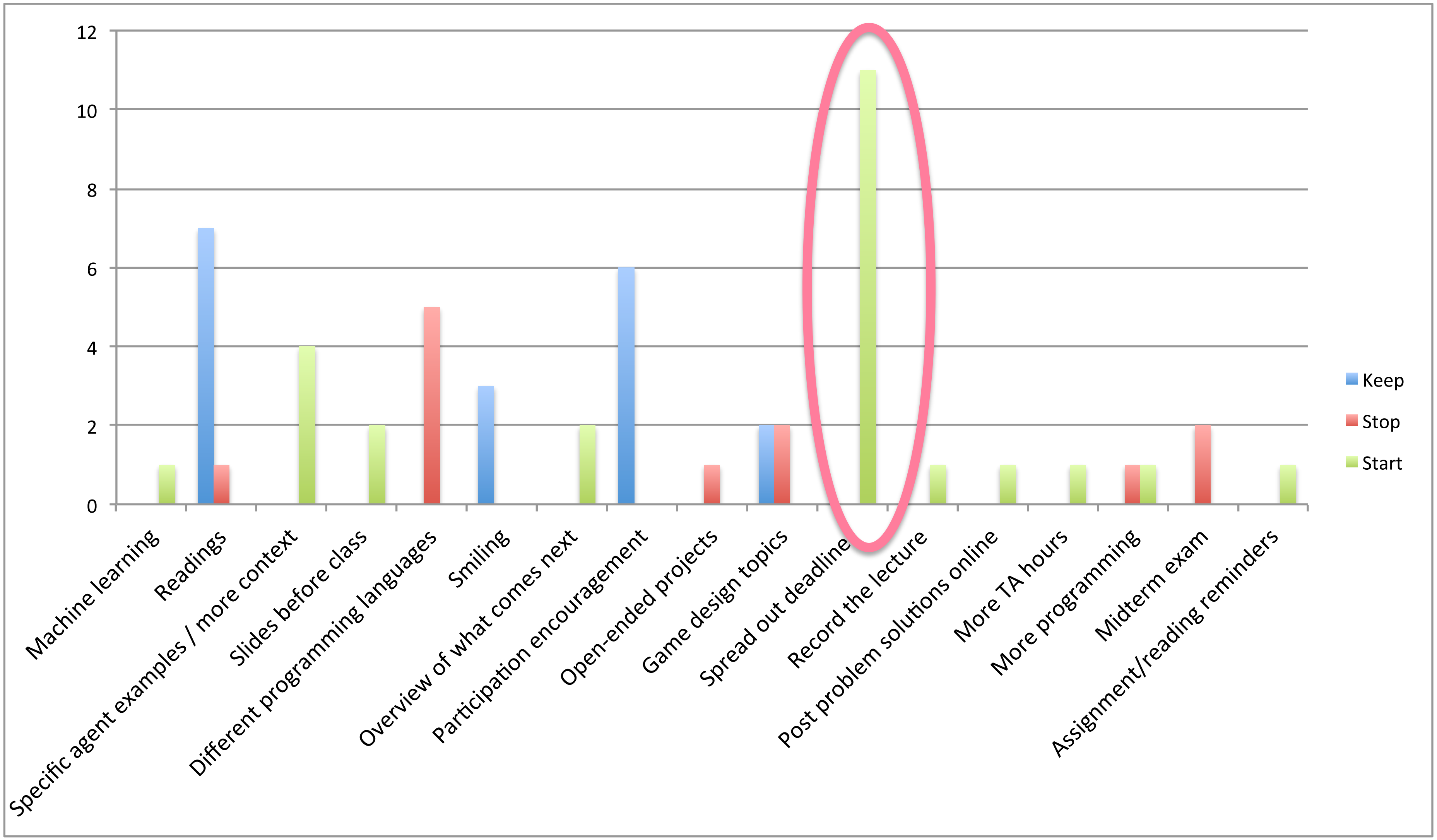
results!

STOP, KEEP, START

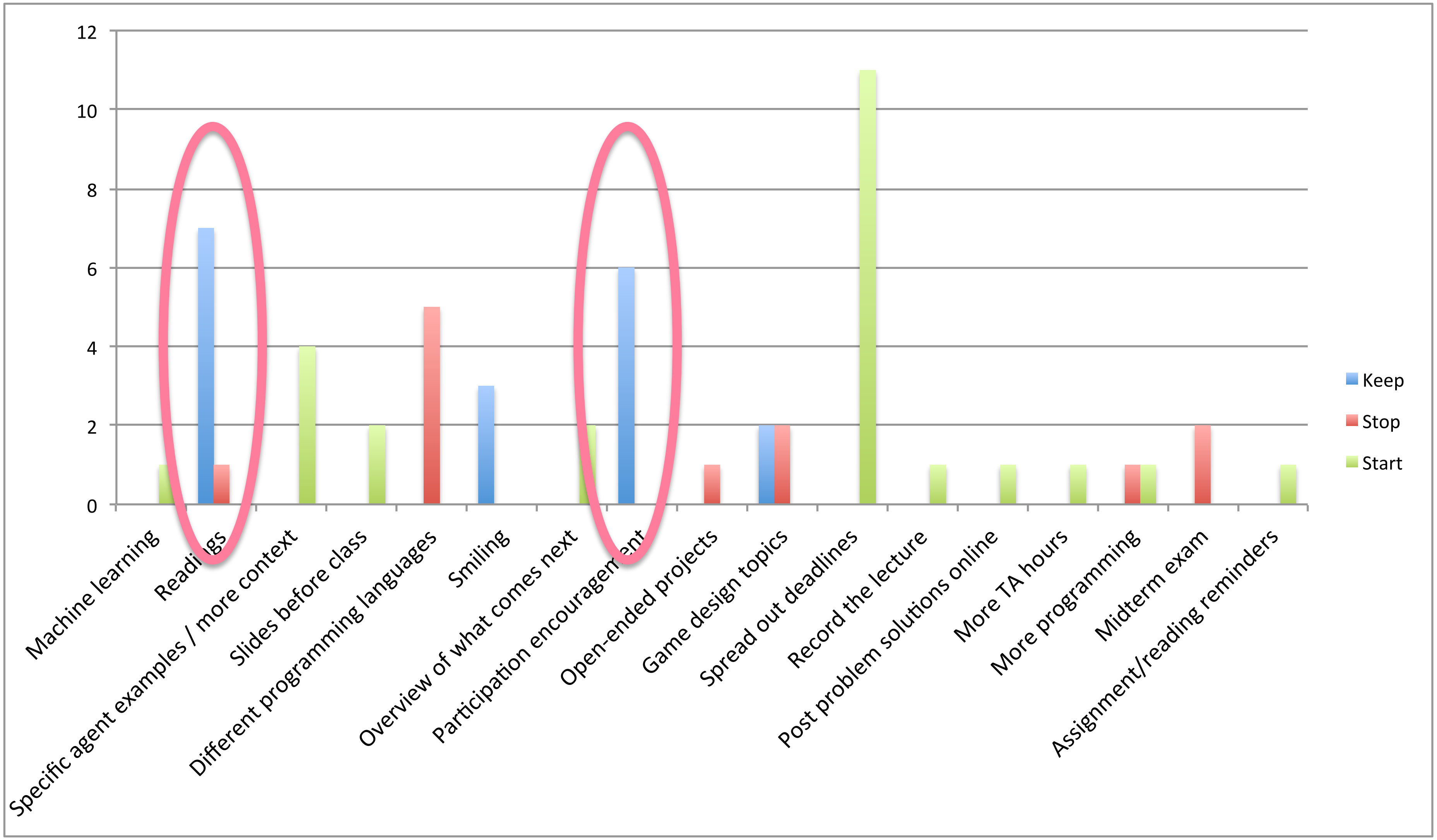
Stop, Keep, Start: Results



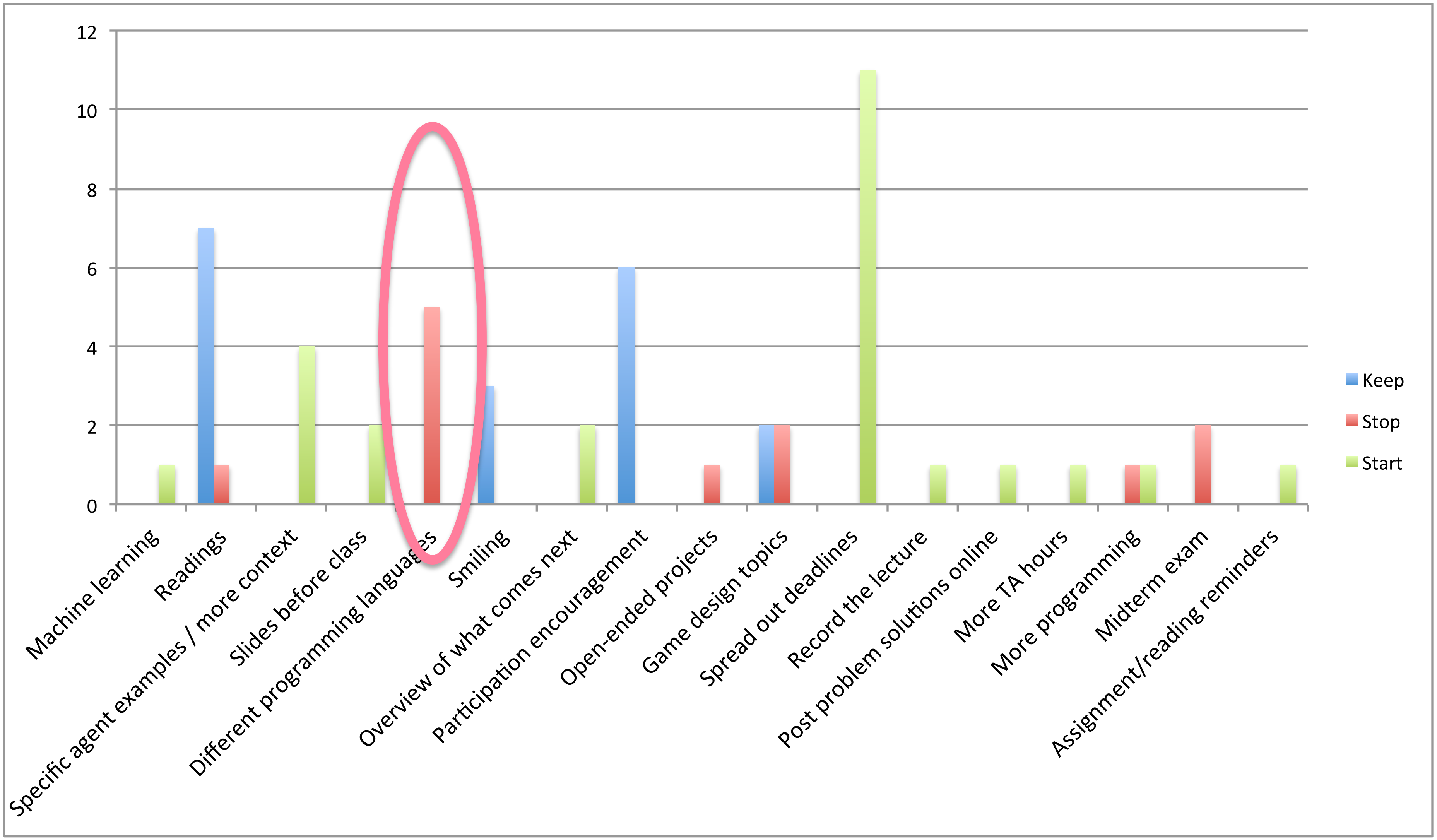
Spreading Out Deadlines



Readings/Participation

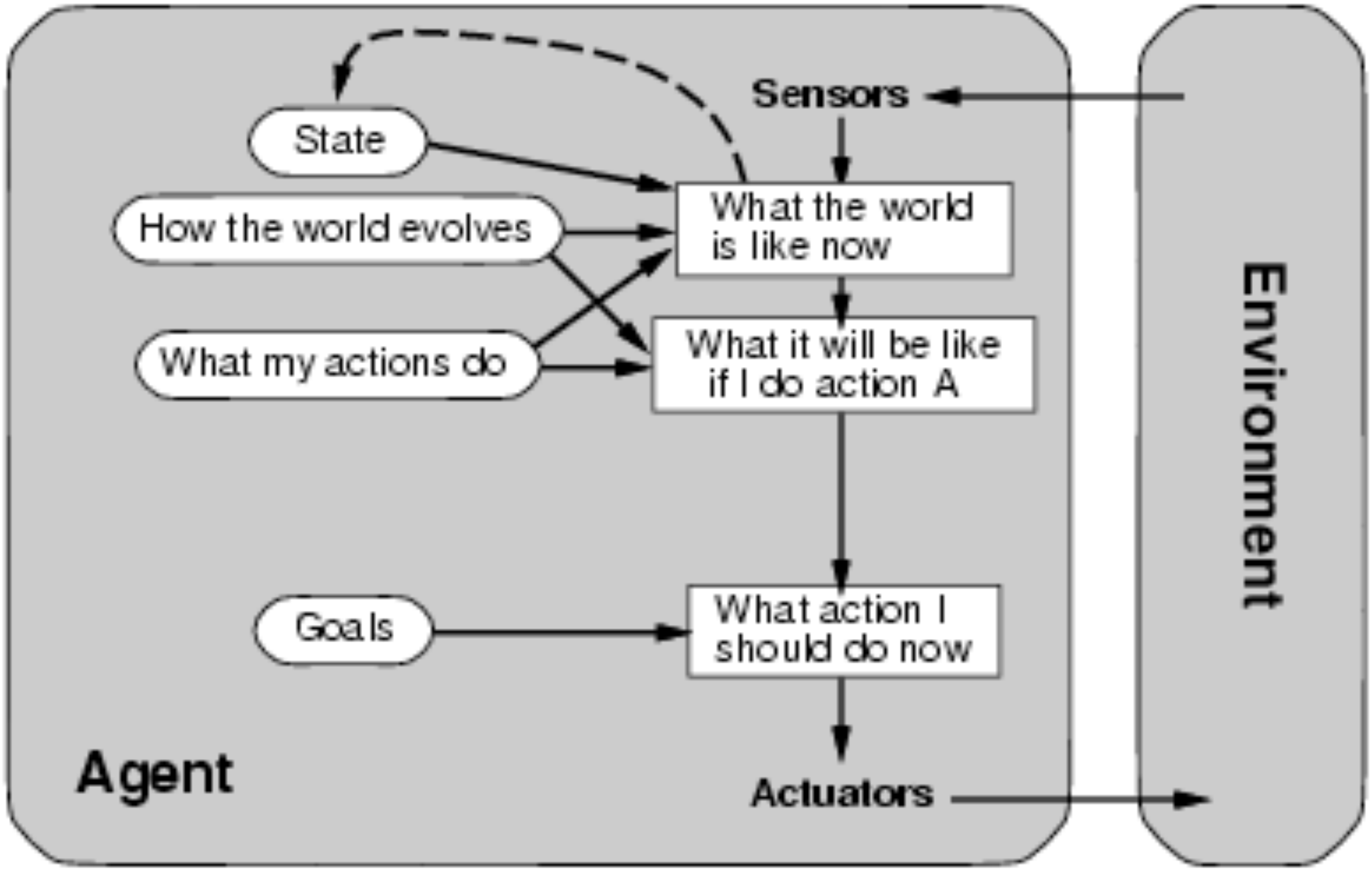


Different Programming Languages

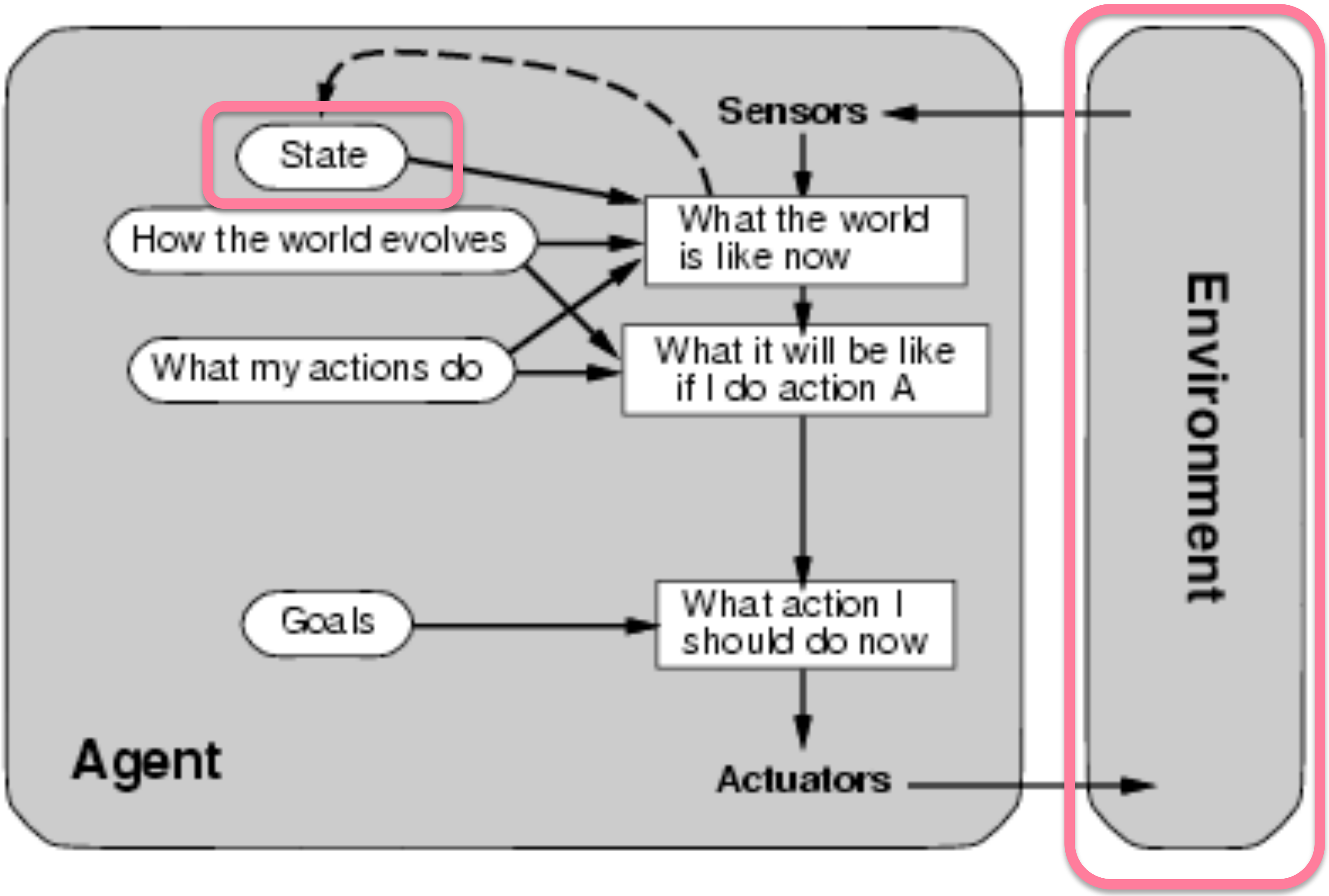


REVIEW

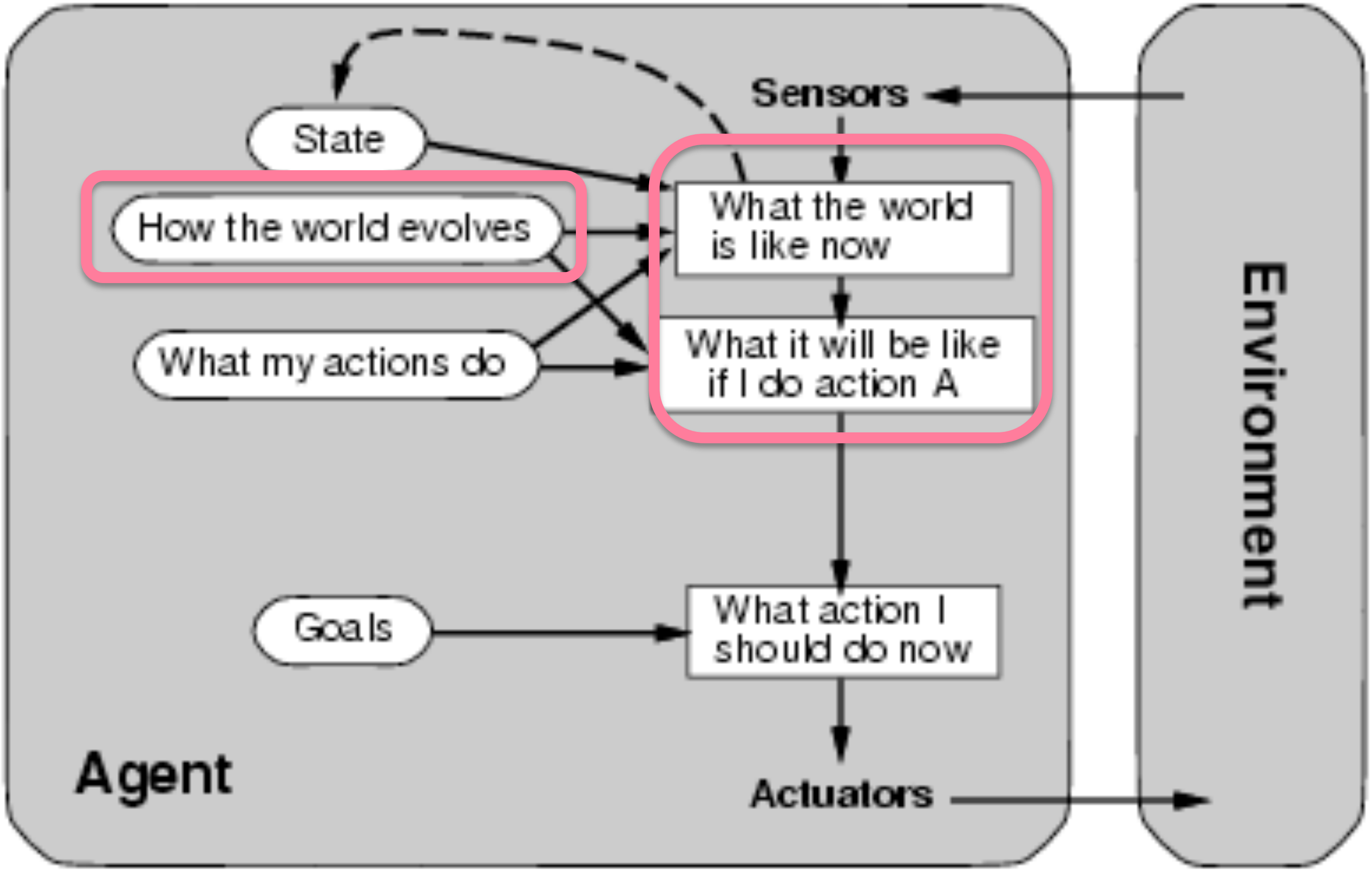
Goal-Based Agents


Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Goals
What action I should do now
Agent
Actuators
Environment

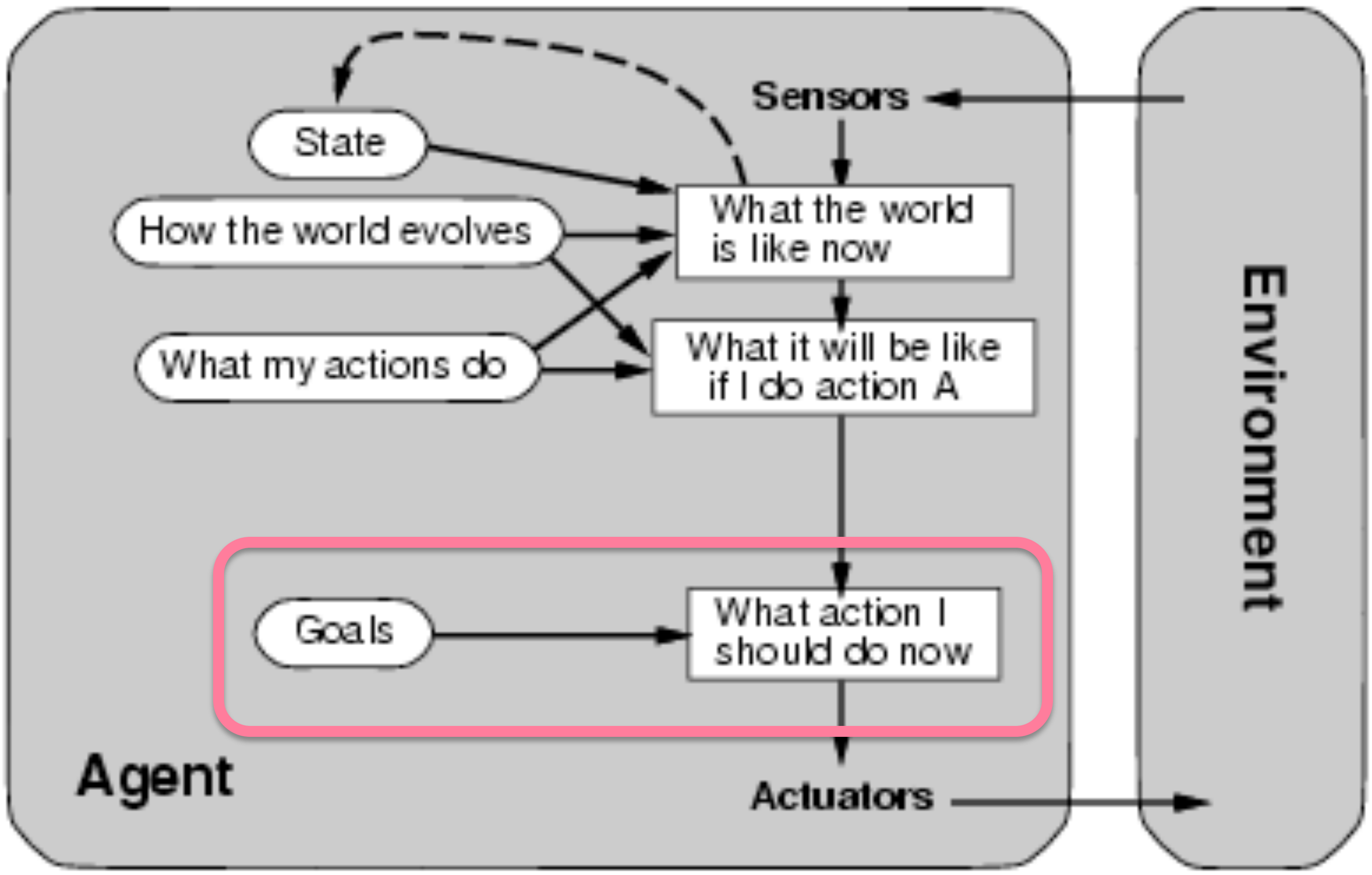
Knowledge Representation


State
How the world evolves
What my actions do
Goals
Sensors
What the world is like now
What it will be like if I do action A
What action I should do now
Actuators
Agent
Environment

Inference, Abduction, Constraint Solving


Sensors
State
How the world evolves
What the world is like now
What my actions do
What it will be like if I do action A
Goals
What action I should do now
Agent
Actuators
Environment

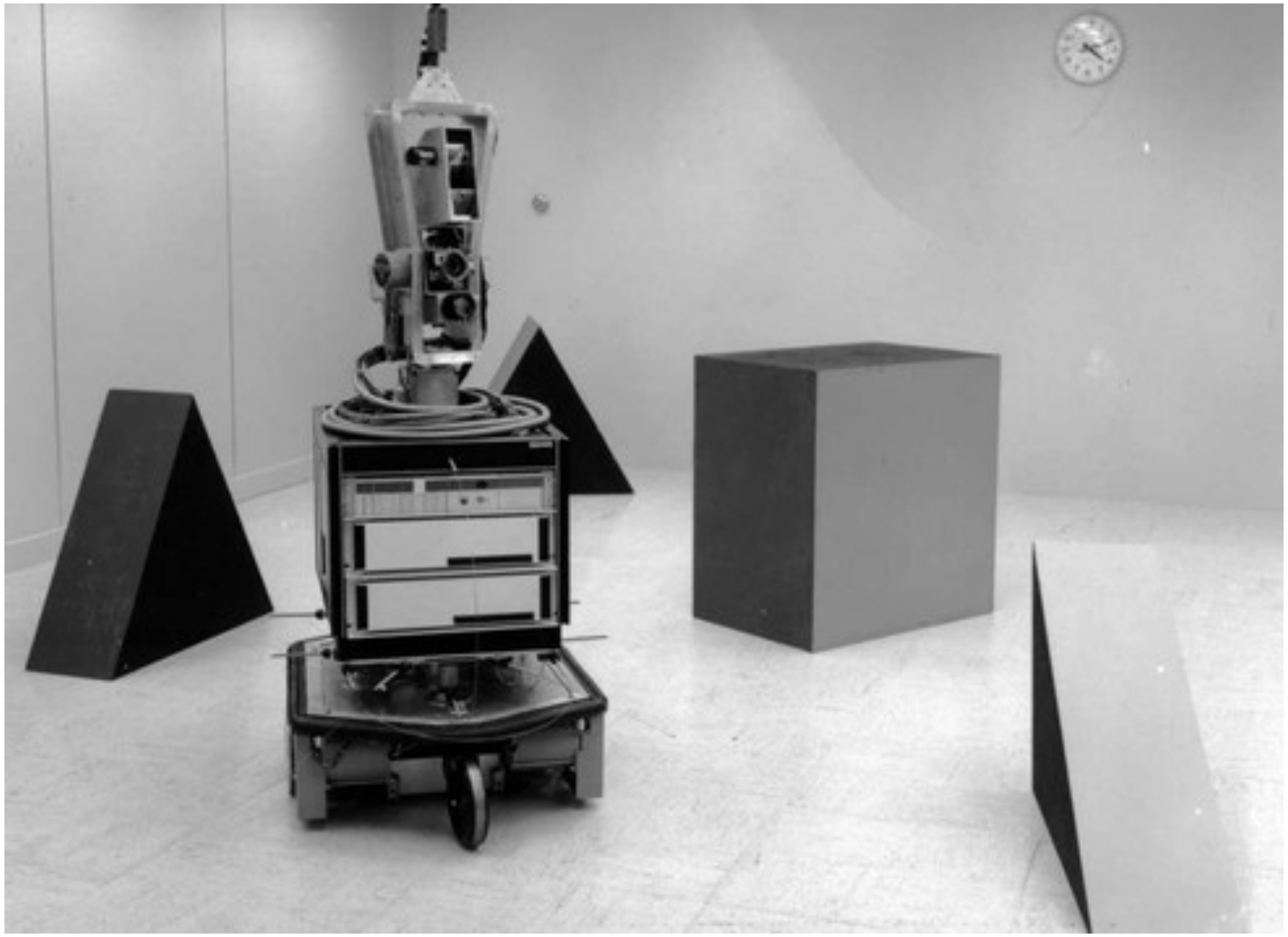
Inference, Abduction, Constraint Solving


State
How the world evolves
What my actions do
Sensors
What the world is like now
What it will be like if I do action A
Goals
What action I should do now
Agent
Actuators
Environment

AI Planning

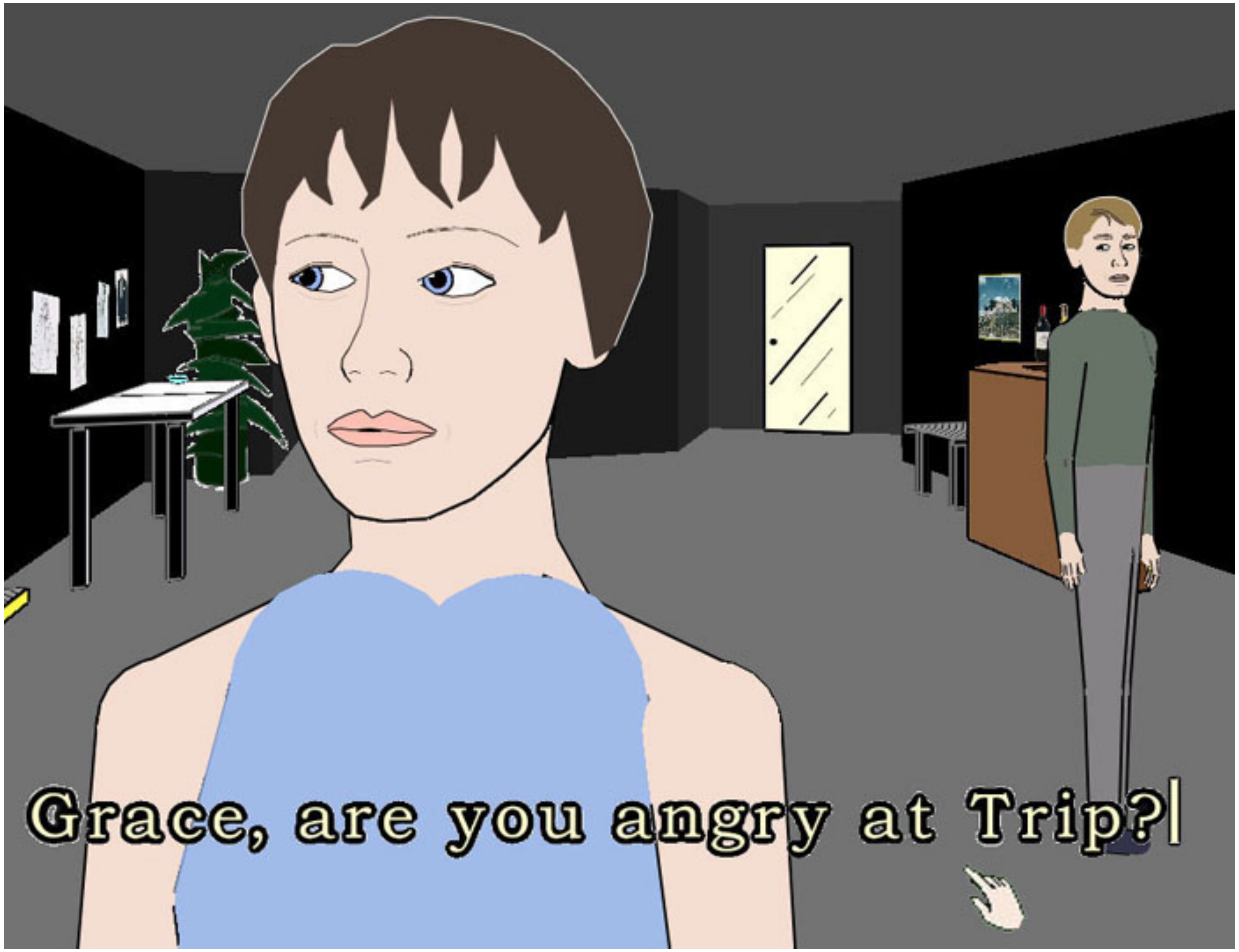
- What are my goals?
- What do I currently know?
- How do I plan my actions to reach my goals?
- What if the world changes?

Planning Applications



Planning Applications

Planning Applications



Planning Applications

…and many more!

- Forest fire management
- Playing games
- Factory automation
- Network security
- …

STRIPS PLANNING

AI Planning

- Description of a planning problem
  - Way to describe the world
  - Initial state
  - Goal description
  - Set of possible **actions** to change the world
- Result: generate a **plan**
  - Sequence of actions for changing the initial state into the goal state

The Frame Problem

- I go from home to the store – new situation S’

The Frame Problem

- I go from home to the store – new situation S’
- In S’
    - The store still sells chips
    - My age is still the same
    - Star Trek is still excellent
    - Santa Cruz is still a weird beach town
    - Boston is still cold
- How can we efficiently represent everything that doesn’t change?

The Ramification Problem

- I go from home to the store – new situation S’
- In S’
  - I am now in West Roxbury
  - The store has one more person in it
  - My head is now in the store
  - My legs are now in the store
  - The contents of my pockets are now in the store…
- That’s a lot of effects for an action!

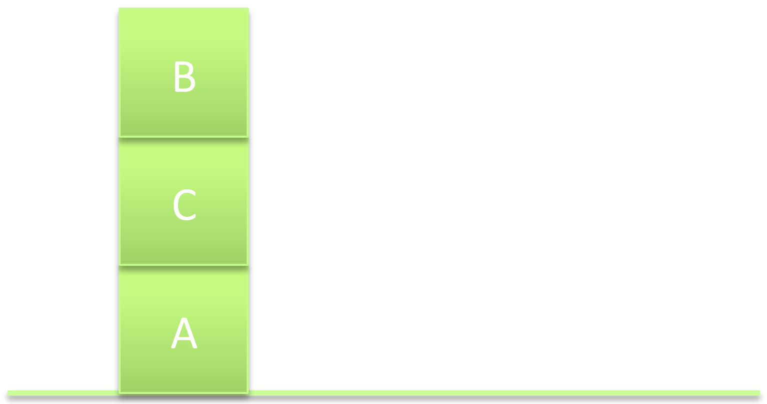
Solutions

- Ramification problem:
  - Some facts must be **inferred** based on world state
- Frame problem:
  - Facts are assumed to **persist** between states unless changed
- All comes down to knowledge engineering!

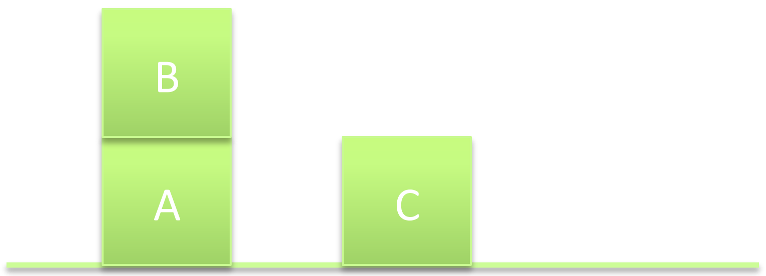
Actions

- What actions are available?
- How do they change the state of the world?

Example Problem: Blocks World

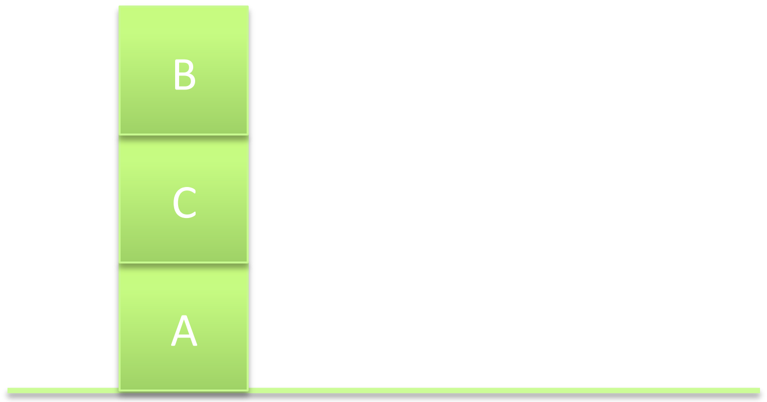


initial state

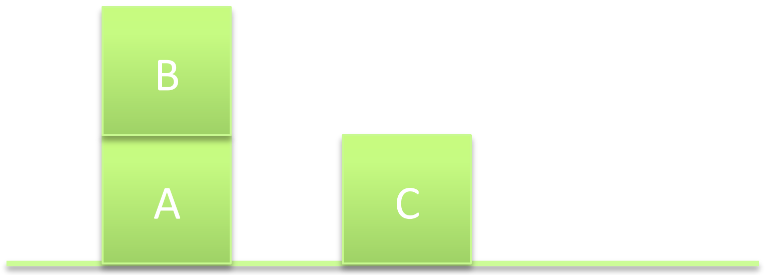


goal state

Example Problem: Blocks World



initial state



goal state

- Represent these states in first order logic

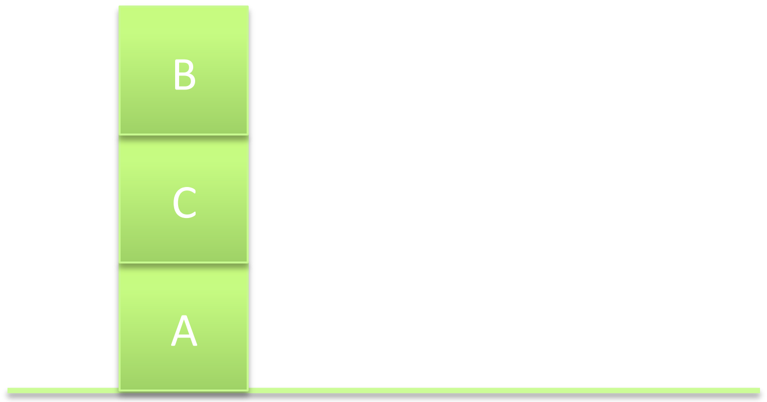
Actions

- What actions are available?
- **How do they change the state of the world?**

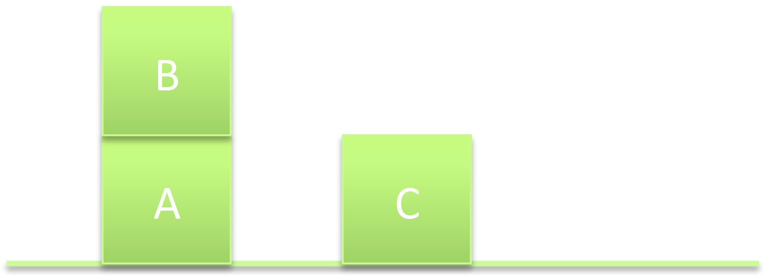
Language for Planning Representation

- PDDL – Planning Domain Description Language
- **Planning operators** to achieve goals
  - Preconditions
  - Effects
    - Add list
    - Delete list
- Subset of first-order logic

Example Problem: Blocks World



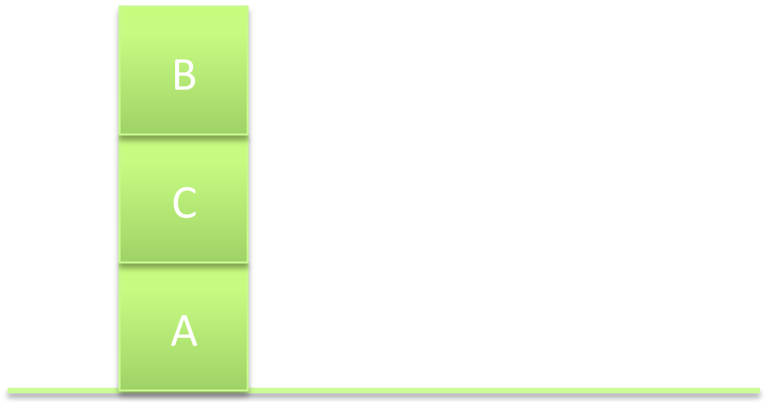
initial state



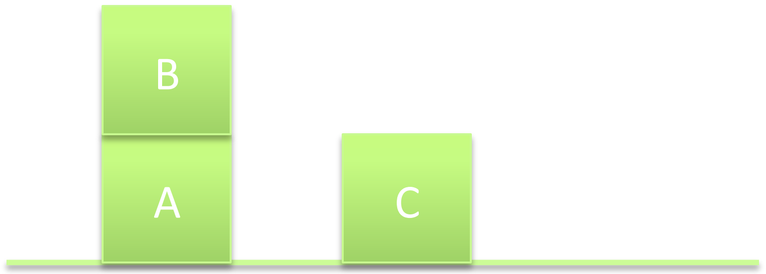
goal state

- What are the actions?

Example Problem: Blocks World



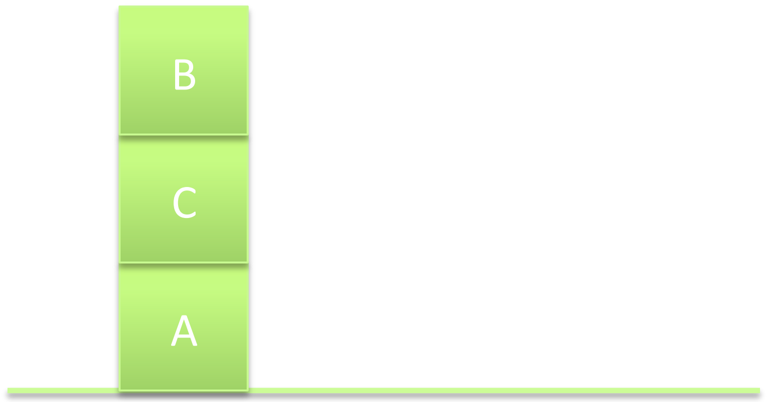
initial state



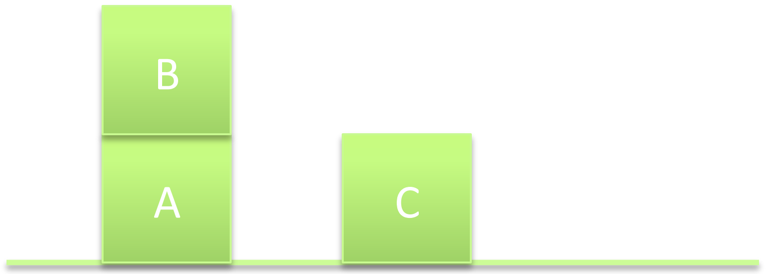
goal state

- And what are their preconditions?

Example Problem: Blocks World



initial state



goal state

- And what are their effects?

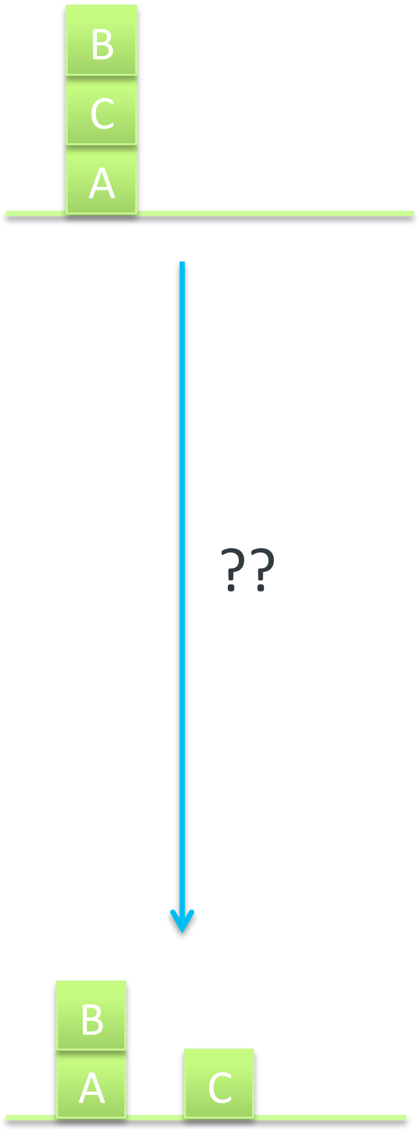
Finding a Plan: Searching Forwards

- Depth-first search
    - Better ways to do this? Stay tuned for next lecture!

- Starting from the initial state:
    - Pick an operator with satisfied preconditions
    - Apply it to update state
    - Expand tree to include updated state descriptions
    - Continue search with updated state as current
    - Stop when you find the goal within a state description
        - Fail when you've checked everything

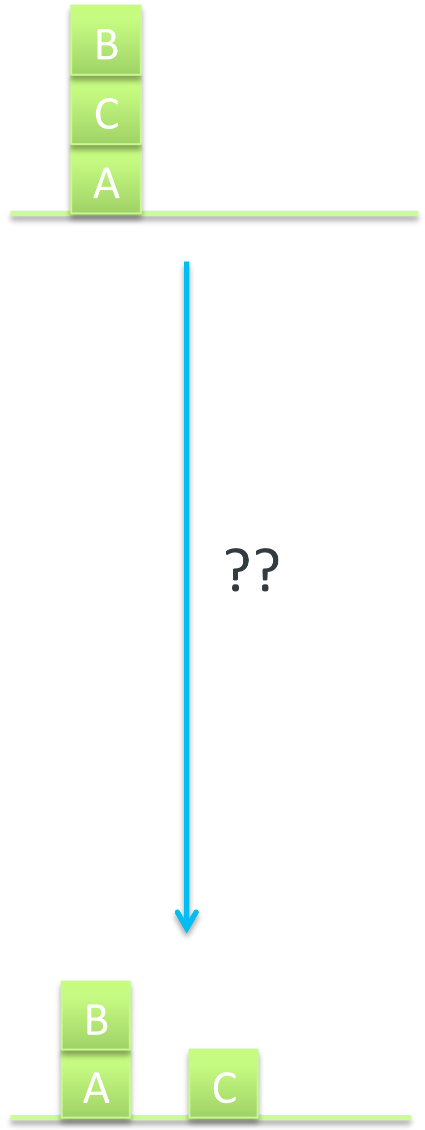
Finding a Plan: Searching Forwards

- Initial state:
  - on(C, A) ontable(A), on(B, C), clear(B), handempty
- Operator: unstack(B, C)
  - Precondition: handempty, clear(B)
  - Add: holding(B), clear(C)
  - Delete: clear(B), on(B, C), handempty
- Updated state:
  - on(C, A), ontable(A), …



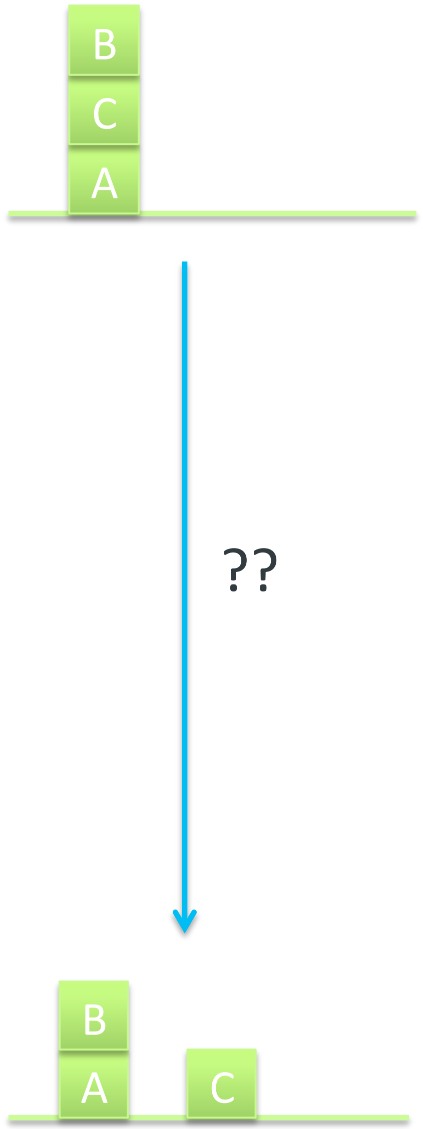
Finding a Plan: Searching Forwards

- Initial state:
  - on(C, A) ontable(A), **on(B, C), clear(B), handempty**
- Operator: unstack(B, C)
  - Precondition: handempty, clear(B)
  - Add: holding(B), clear(C)
  - Delete: **clear(B), on(B, C), handempty**
- Updated state:
  - on(C, A), ontable(A)

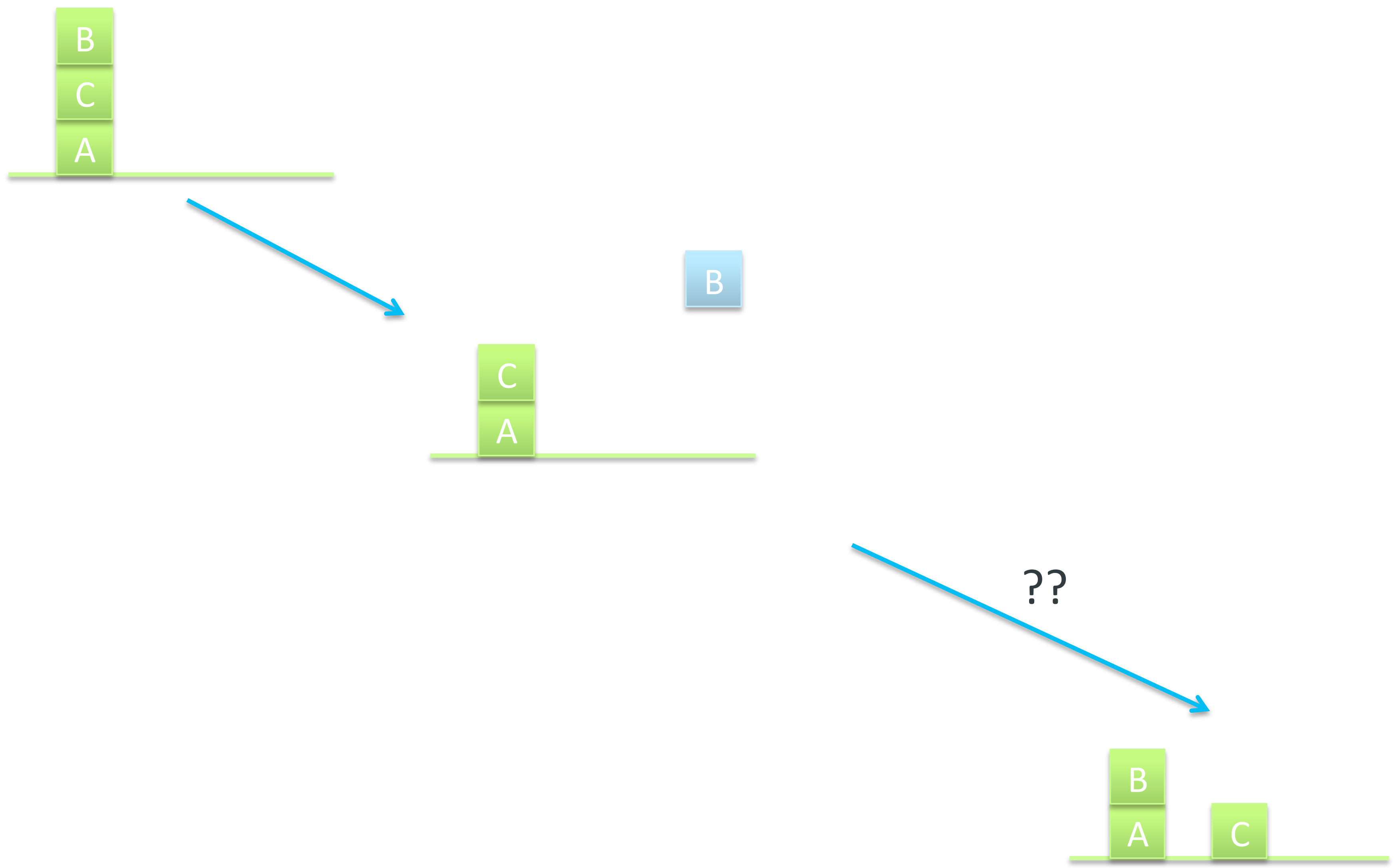


Finding a Plan: Searching Forwards

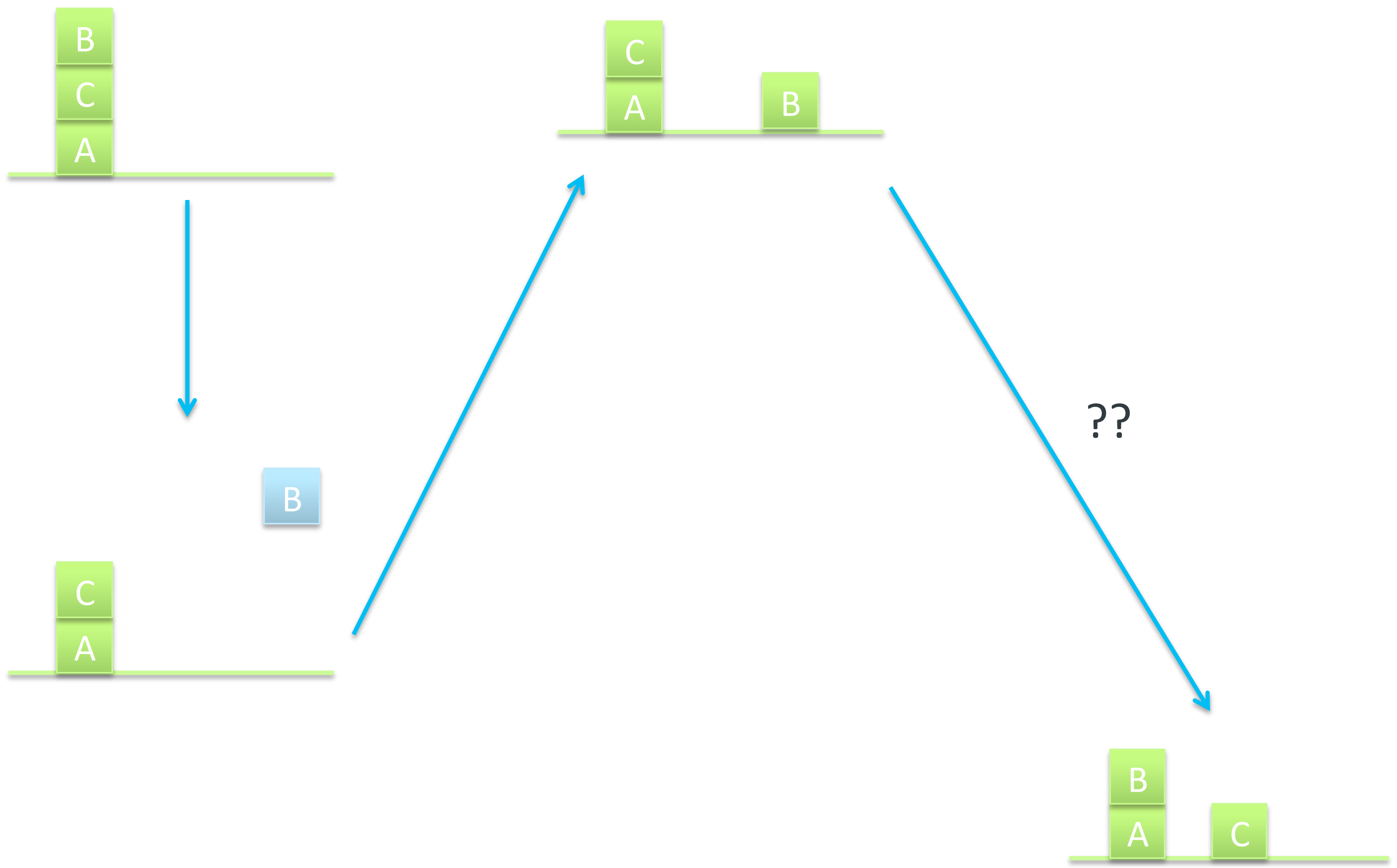
- Initial state:
  - on(C, A) ontable(A), on(B, C), clear(B), handempty
- Operator: unstack(B, C)
  - Precondition: handempty, clear(B)
  - Add: **holding(B), clear(C)**
  - Delete: clear(B), on(B, C), handempty
- Updated state:
  - on(C, A), ontable(A), **holding(B), clear(C)**



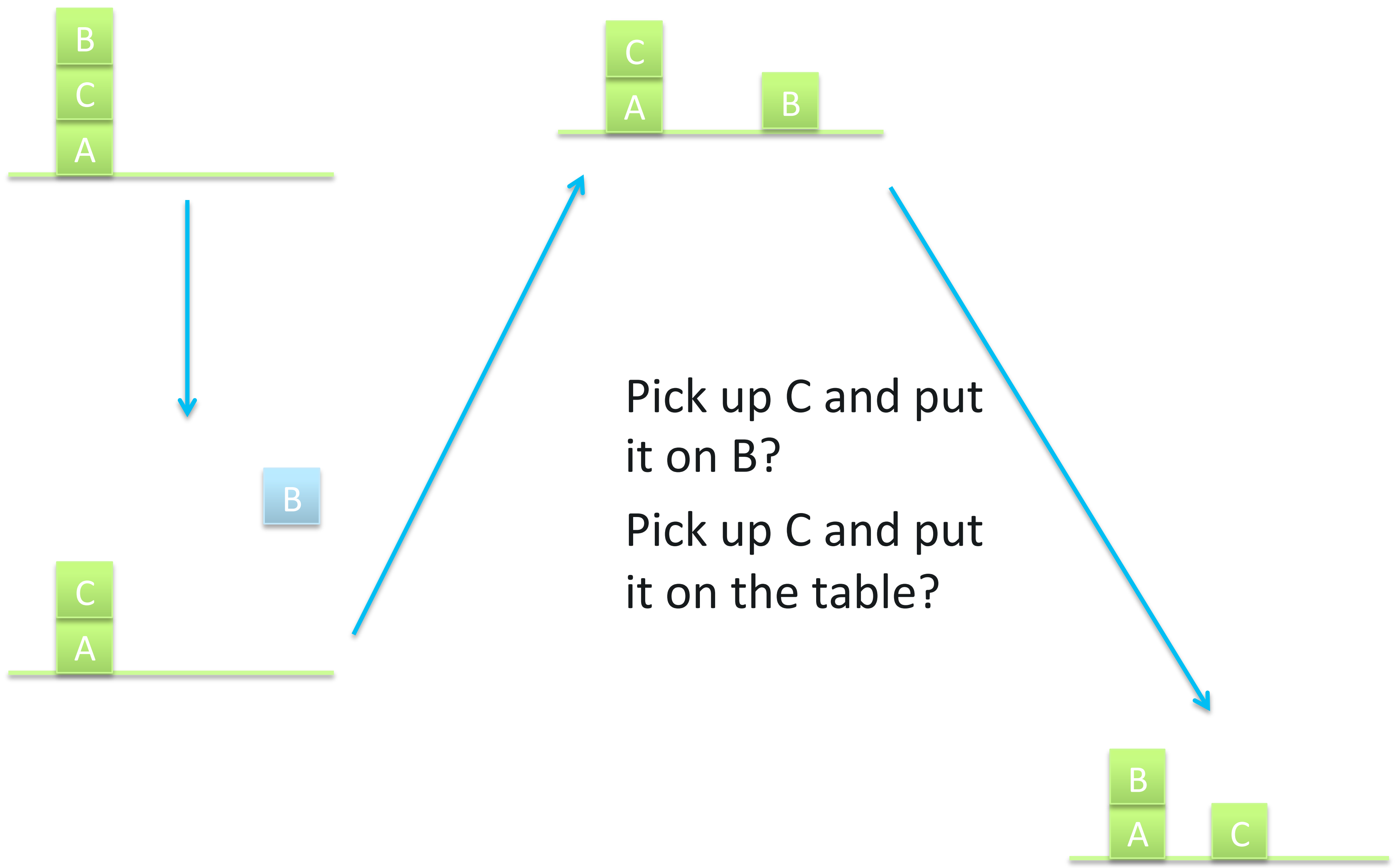
Finding a Plan: Searching Forwards



Finding a Plan: Searching Forwards



Finding a Plan: Searching Forwards



Pick up C and put it on B?

Pick up C and put it on the table?

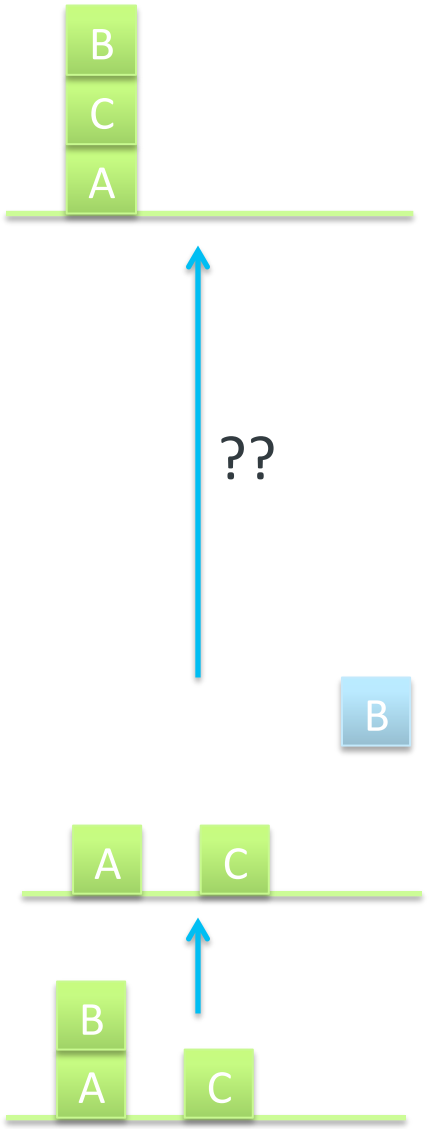
Finding a Plan: Searching Backwards

- Search **backwards** from goal to find a solution
    - Only looks at relevant actions!

- Starting from the **goal**:
    - If initial state satisfies the current goal, done!
    - Choose an operator with an add list that matches goals
        - Fail if no such operator, or if it has effects that contradict the goals
    - Update current goals by subtracting add list, adding preconditions
        - Fail if current goals are a subset of the new goals
    - Continue search with updated goals

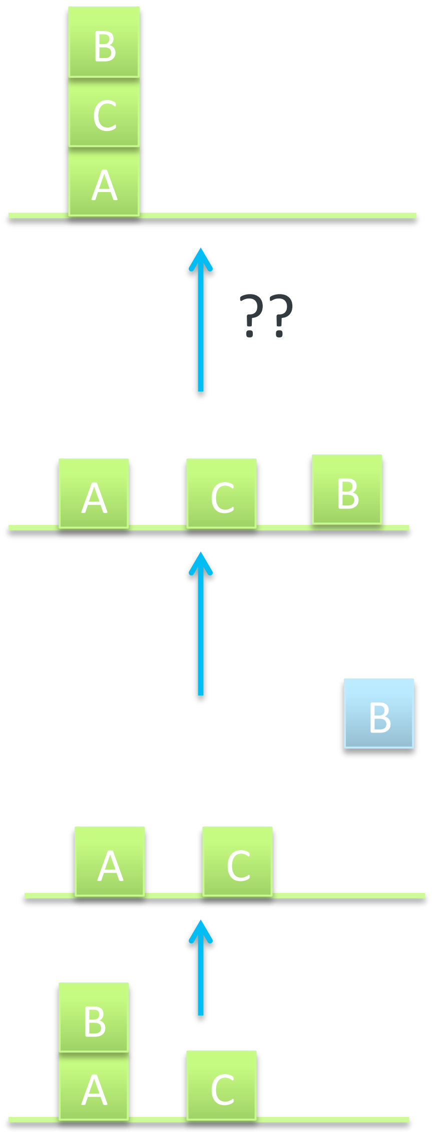
Finding a Plan: Searching Backwards

- Initial **goal**:
  - **on(B, A),** ontable(C), **handempty**, **clear(B)** clear(C), ontable(A)
- Operator: stack(B, A)
  - Add list: **on(B, A), clear(B), handempty**
  - Del list: holding(B)
  - Precondition: **holding(B), clear(A)**
- Updated **goal**:
  - ontable(C), clear(C), ontable(A) **holding(B), clear(A)**



Finding a Plan: Searching Backwards

- Goal:
  - ontable(C), clear(C), ontable(A), **holding(B)**, clear(A)
- Operator: pickup(B)
  - Add list: **holding(B)**
  - Del list: handempty, clear(B), ontable(B)
  - Precondition: **handempty, clear(B), ontable(B)**
- Updated **goal**:
  - ontable(C), clear(C), ontable(A), **handempty, clear(B), ontable(B),** clear(A)



In-Class Exercise

- A monkey is sitting at his desk, typing on his typewriter. There is a box next to the window. There are bananas hanging from the ceiling above the printer, but the monkey needs the box to reach them. The monkey's goal is to sit at his desk and eat a banana.
    - What is the initial state and goal state?

In-Class Exercise

- A monkey is sitting at his desk, typing on his typewriter. There is a box next to the window. There are bananas hanging from the ceiling above the printer, but the monkey needs the box to reach them. The monkey's goal is to sit at his desk and eat a banana.
  - What is the initial state and goal state?
  - What are the operators?

In-Class Exercise

- A monkey is sitting at his desk, typing on his typewriter. There is a box next to the window. There are bananas hanging from the ceiling above the printer, but the monkey needs the box to reach them. The monkey's goal is to sit at his desk and eat a banana.
  - What is the initial state and goal state?
  - What are the operators?
  - Run through the first few steps of forward and backward search.

HTN PLANNING

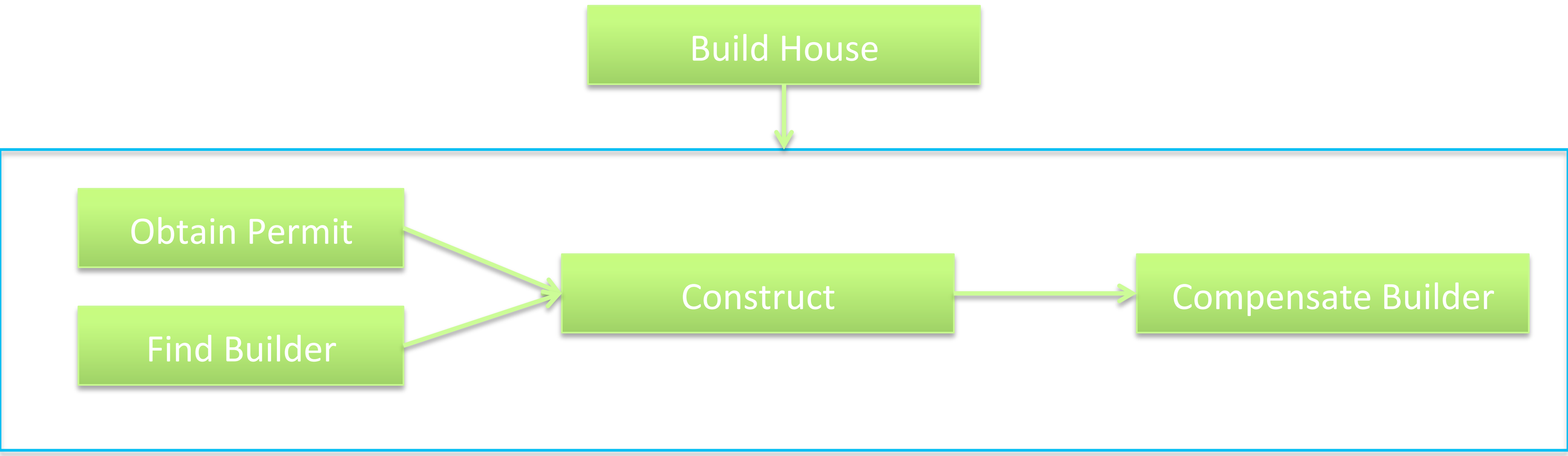
An Alternate Approach

- **H**ierarchical **T**ask **N**etwork planning
- Encoding domain-specific “recipes” for achieving goals
- **Tasks** instead of goals
- Hierarchical **decomposition** of task networks
  - Into other tasks
  - Into primitive operators

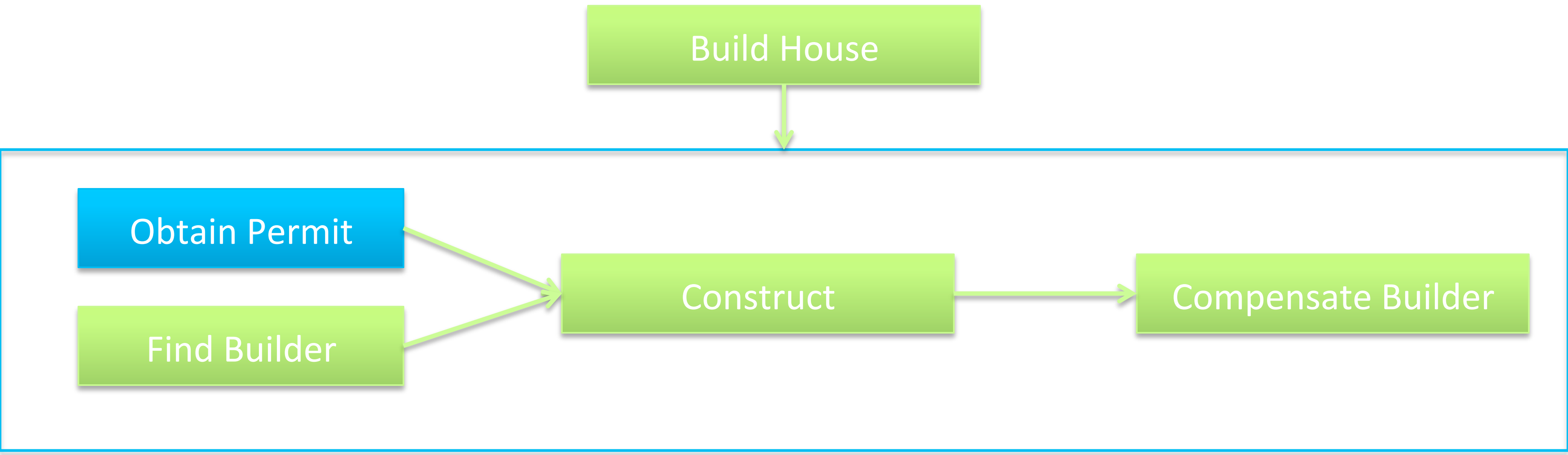
Example: Building a House

Build House

Example: Building a House

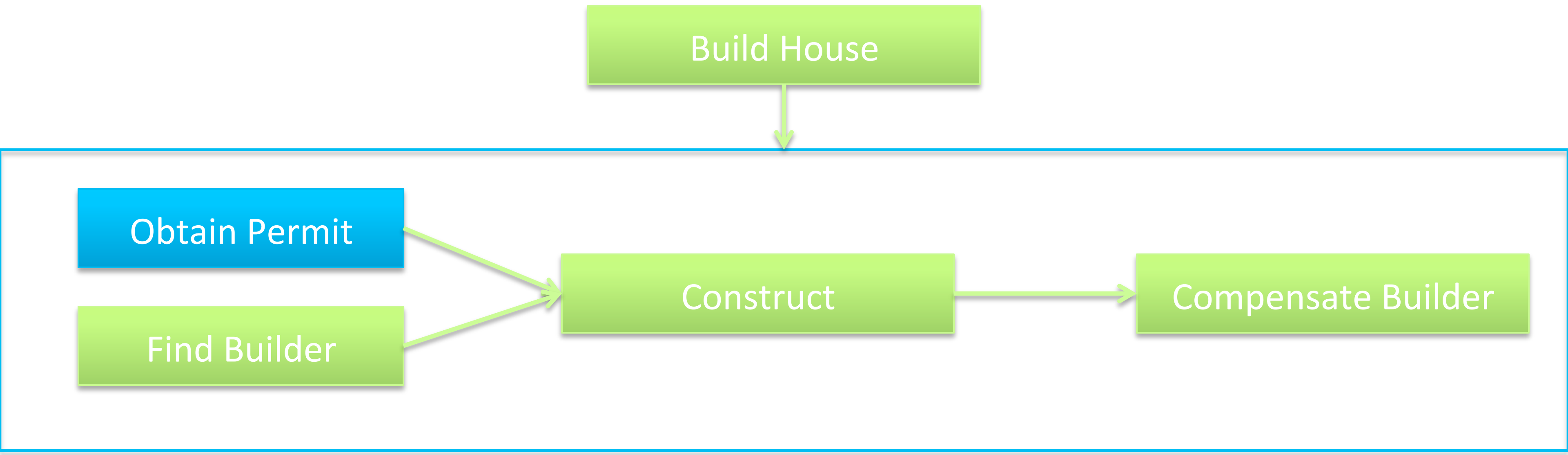


Example: Building a House



Visit city hall, take a number, wait to be called, walk to desk, fill out paperwork, ...

Example: Building a House



Visit city hall, take a number, wait to be called, walk to desk, fill out paperwork, ...

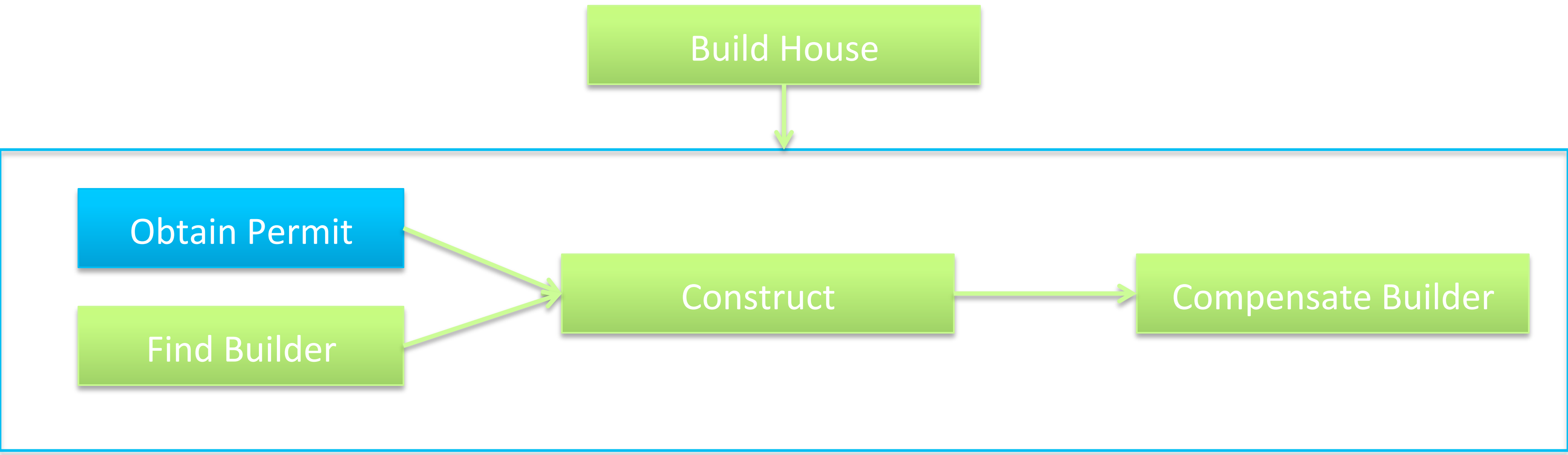
Walk to car

Drive car to city hall

Park

…

Example: Building a House



Visit city hall, take a number, wait to be called, walk to desk, fill out paperwork, ...

Walk to bus stop

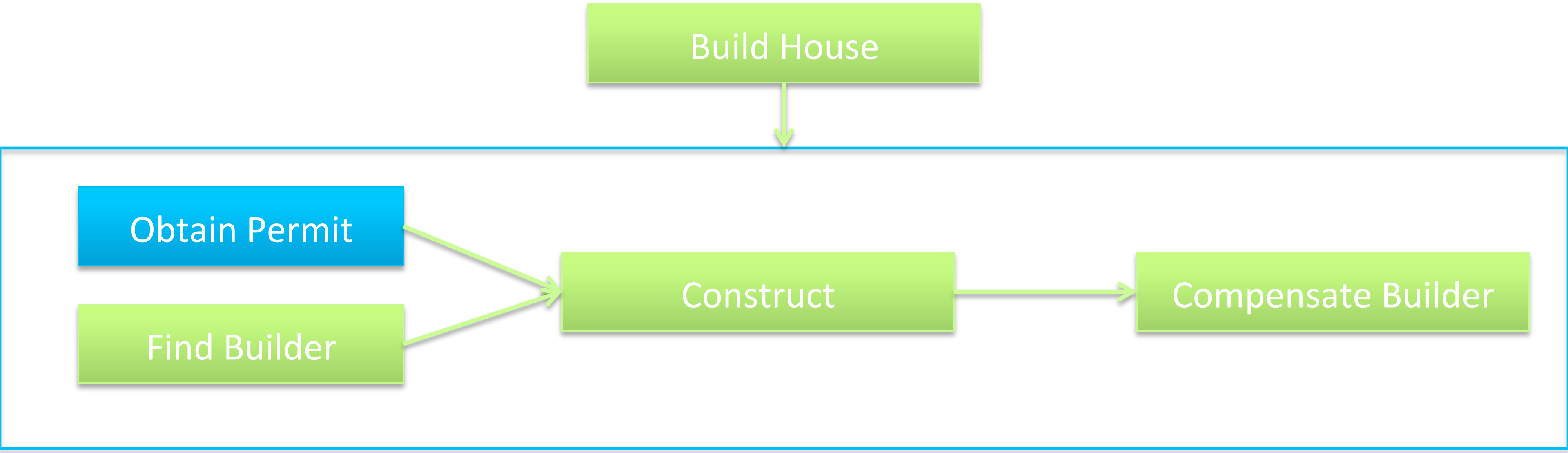
Take 37 bus to Forest Hills

Take Orange Line to South Station

…

These are the primitive operators!

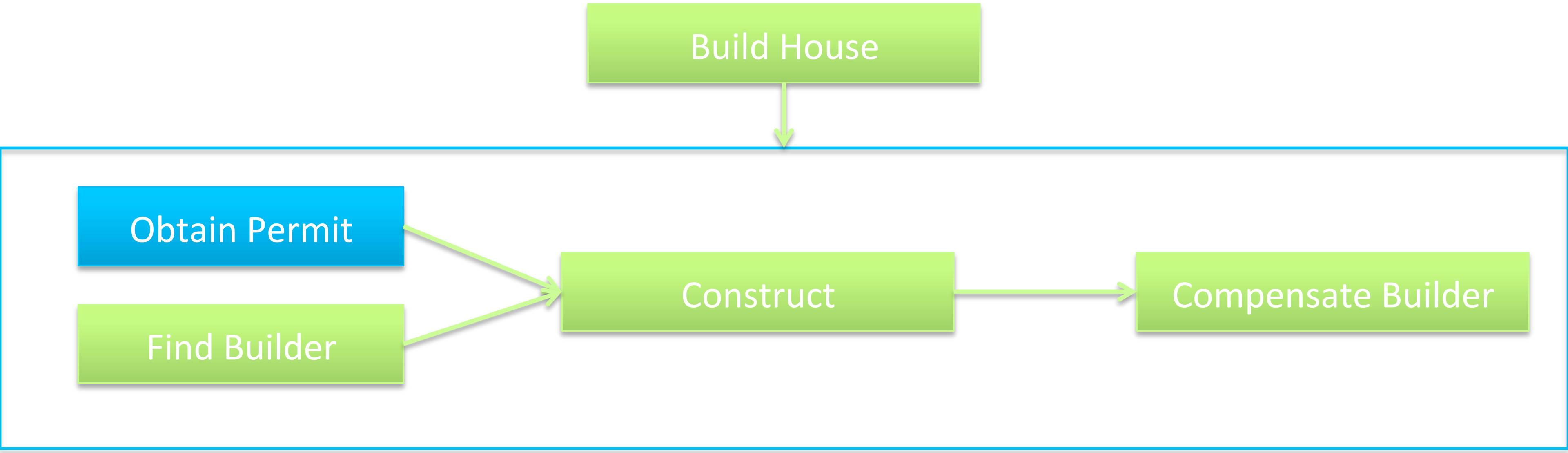
Example: Building a House



Visit city hall, take a number, wait to be called, walk to desk, fill out paperwork, ...

Find forger, meet forger in alley, pay $200, …

Example: Building a House



Visit city hall, take a number, wait to be called, walk to desk, fill out paperwork, ...

Precondition: unscrupulous

Find forger, meet forger in alley, pay $200, …

Class Exercise

- How can you express your morning routine as an HTN?

- Twists:
  - On Thursdays, you go to the gym
  - Your car is unreliable – some days it works, some days it does not
  - You like to take a bath instead of shower if you wake up before your alarm
  - You need caffeine in the morning, but sometimes your roommate accidentally prepares decaf coffee

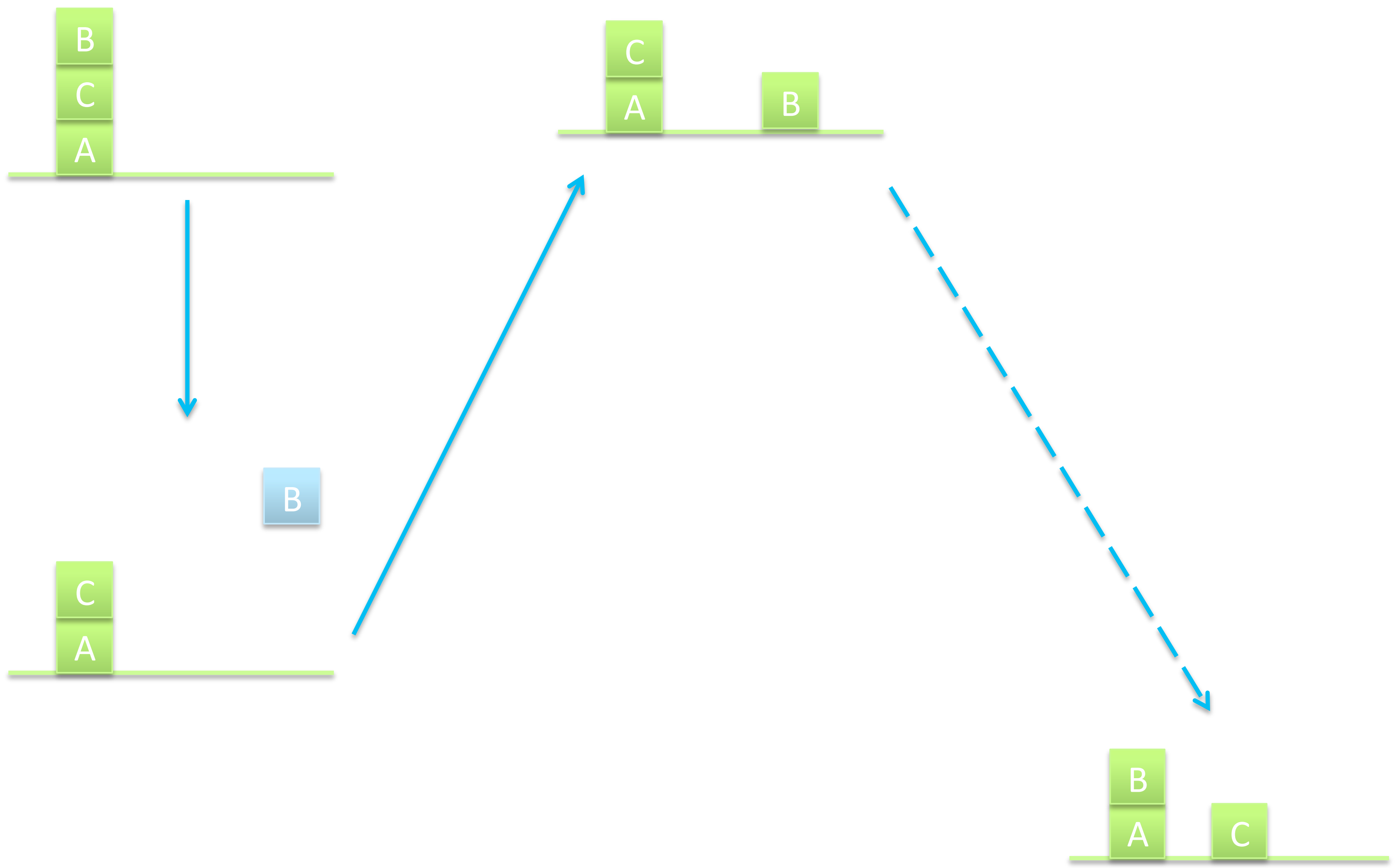
How do I decide which to use?

- HTN: good for planning problems that have a clear hierarchical decomposition

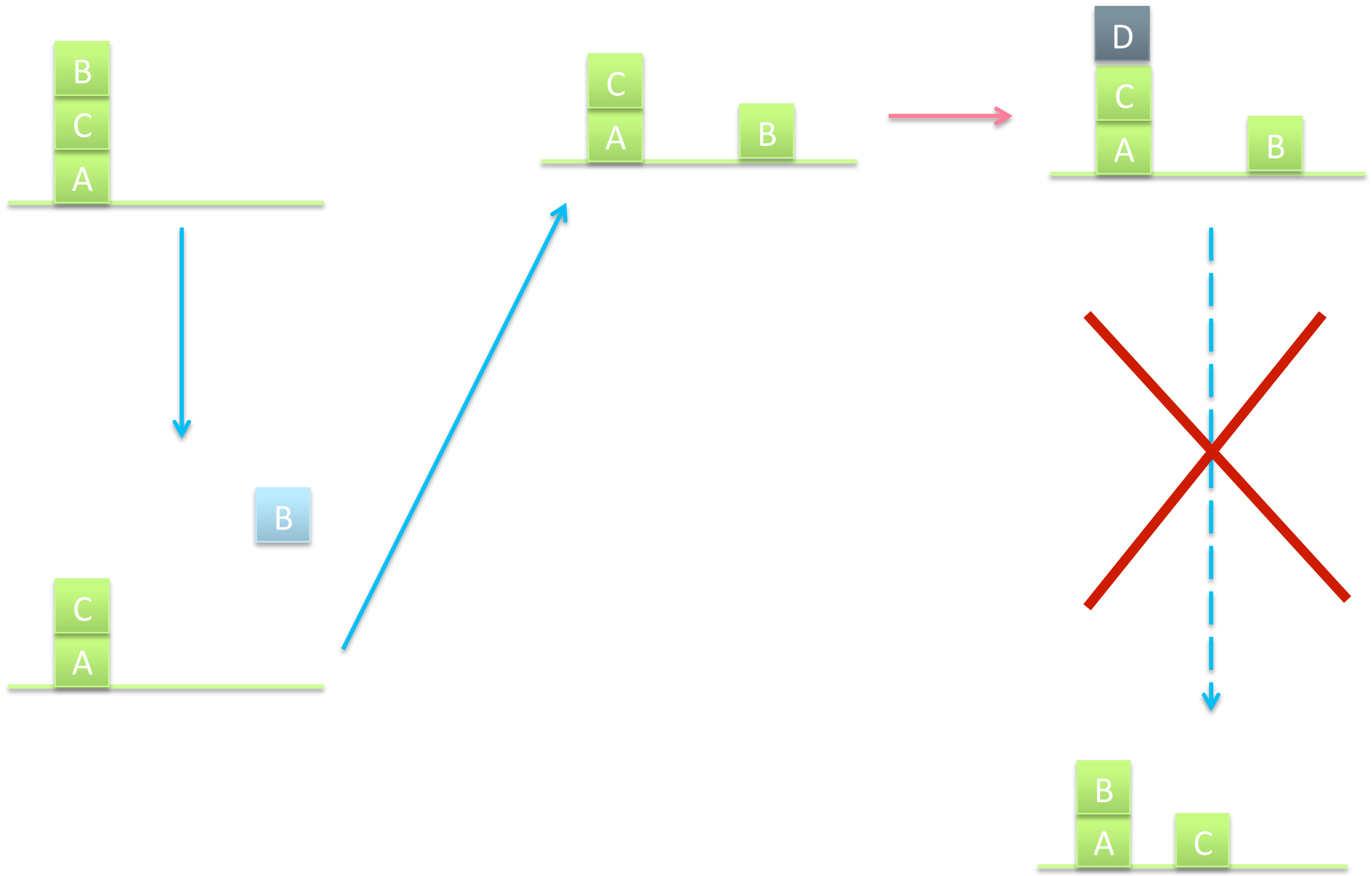
- STRIPS: backward search good for exploratory problems with no “recipes”

REACTIVE PLANNING

Executing Plans in a Changing World



Executing Plans in a Changing World



What do we do now?

- As soon as world changes or conflict is found where plan cannot execute, re-plan!

What do we do now?

- Reactive planning!
- Hierarchical task network where node expansion occurs in real-time
- Authoring problem: accounting for the unexpected

SITUATED ACTIONS