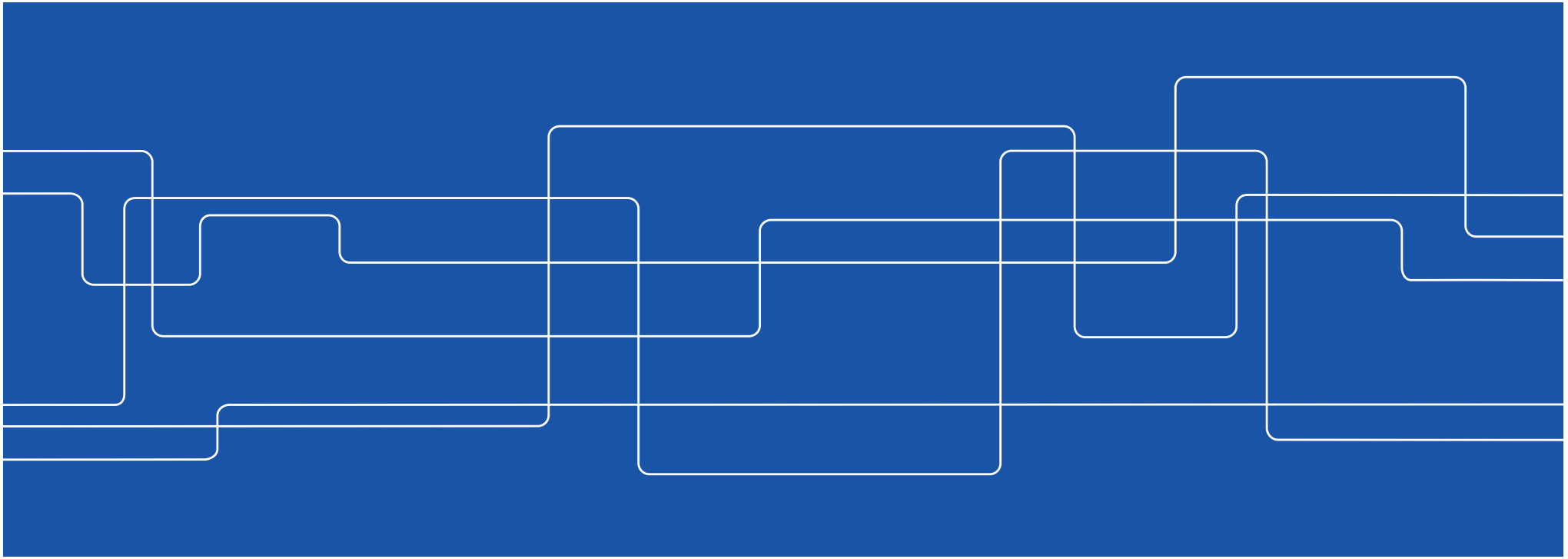
Task Switching and Behavior Trees in Robotics

Lecture 7

Petter Ögren and Michele Colledanchise



Outline

- **What do we mean by task switching?**
- Quotes on BTs
- Why BTs?
- How do BTs work?
- 5 Efficient Design Principles for BTs
- Alternatives to BTs
- The Big Picture
  - BTs and Planning
  - BTs and Control Theory
  - BTs and Genetic Algorithms
  - BTs and Deep Learning

What do we mean by Task Switching?

Robotics is about:

- Mobility, Sensing, Navigation, Mapping, Manipulation, Grasping…

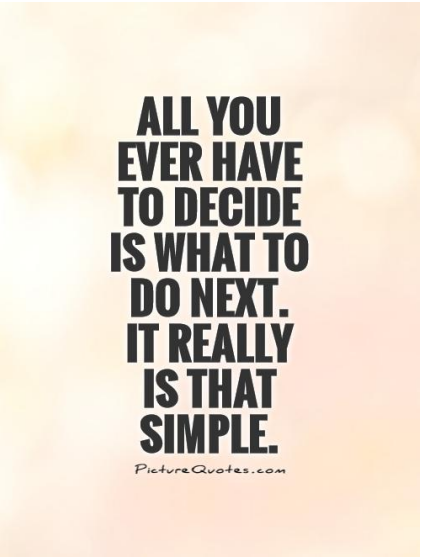
Robotics is also about:

- Unmanned cars
  - (parking, refueling, overtaking…)
- Robotic house cleaning
  - (vacuuming, laundry, sorting stuff…)
- Robotic factory workers
  - (assembling, painting, sorting…)

There is a need to switch between tasks!

- When to stop a task?
- What do do next?

(also challenging for humans: work/play/sleep, read/write/code)



Do we need to worry about Task Switching?

- Robotics researchers:
  - No, it is difficult enough to get grasping working…
  - Switching between 3 tasks is trivial using a FSM…
- Robotic industry:
  - Our Finite State Machines (FSMs) are a mess…
  - Rethink robotics and Jibo use Behavior Trees
- Game industry
  - Grasping, Mapping and Manipulation is trivial
  - Everyone uses Behavior Trees

Outline

- What do we mean by task switching?
- **Quotes on BTs**
- Why BTs?
- How do BTs work?
- 5 Efficient Design Principles for BTs
- Alternatives to BTs
- The Big Picture
  - BTs and Planning
  - BTs and Control Theory
  - BTs and Genetic Algorithms
  - BTs and Deep Learning

Quotes on Behavior Trees

”Over the last few years, various forms of behavior trees (BTs) have become the standard in industry.”

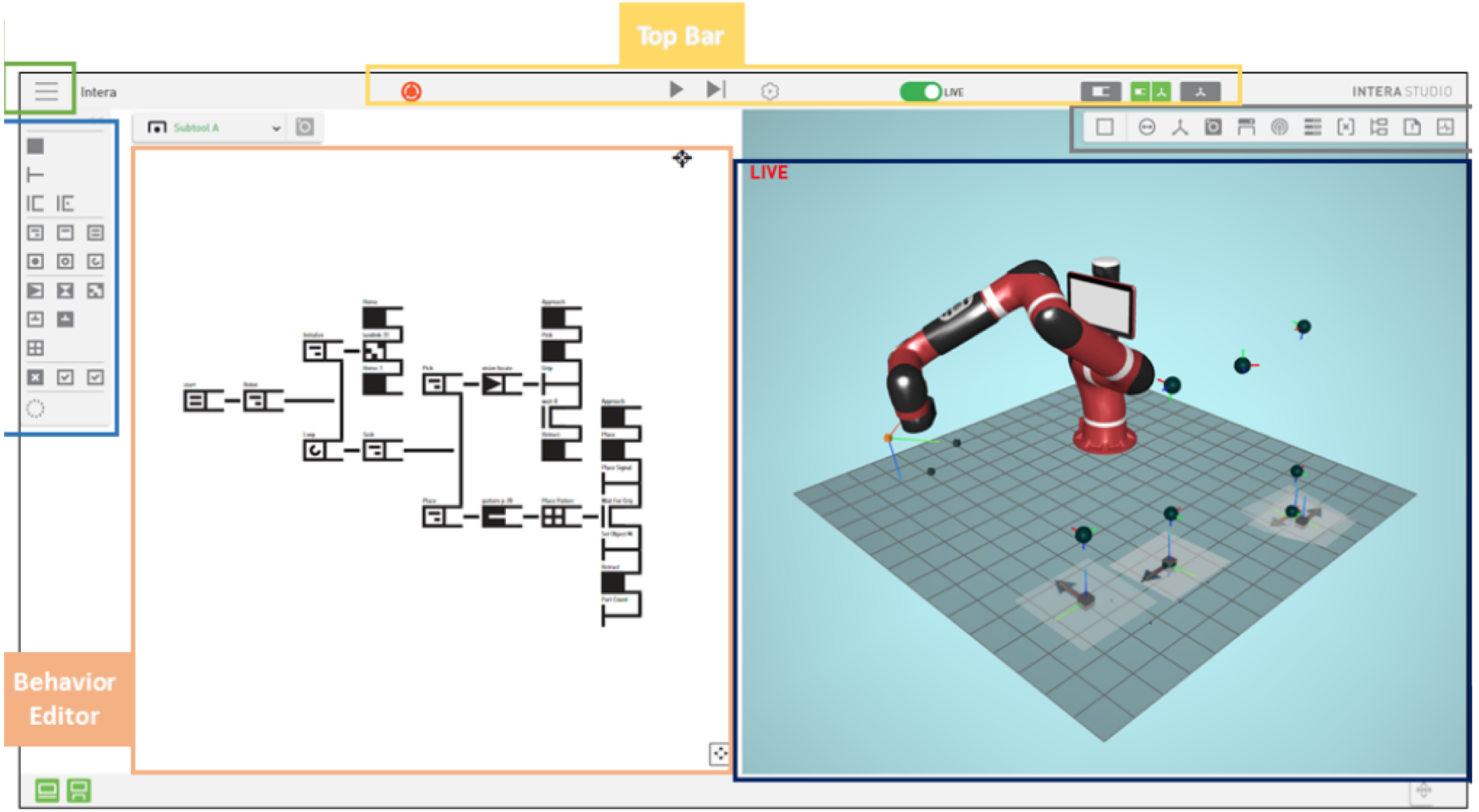
Dawe, Champandard, Hernandez, GDC 2010

Quotes on Behavior Trees

"[...]. Sure you could build the very same behaviors with a finite state machine (FSM). But anyone who has worked with this kind of technology in industry knows how fragile such logic gets as it grows. A finely tuned hierarchical FSM before a game ships is often a temperamental work of art not to be messed with!"

Alex Champandard
Senior AI Programmer
Rockstar Games

Rethink Robotics Intera 5 (2017)



Outline

- What do we mean by task switching?
- Quotes on BTs
- **Why BTs?**
- How do BTs work?
- 5 Efficient Design Principles for BTs
- Alternatives to BTs
- The Big Picture
  - BTs and Planning
  - BTs and Control Theory
  - BTs and Genetic Algorithms
  - BTs and Deep Learning

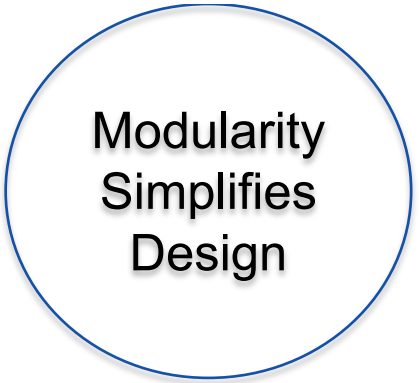
Who Invented BTs?

- The Computer Game Industry
- Valued at $93 billion
- Good at software
- Not afraid to try new stuff
- Choose Solutions that work

- Modularity is key!

BT-editors now available in Game Engines

Why BTs? Take home message:



Behavior Trees is a Control Policy Representation that

- increases **Modularity**
  - which **simplifies**
    - Human **design**
    - Machine **design** such as
      - Learning
      - Planning

Why Behavior Trees?

Problem 1:

“The GOTO statement should be abolished from all higher level programming languages”

- E. Dijkstra, 1968

Reason: GOTO harms **Modularity**

Solution: GOTO not used anymore


Letters to the Editor

Go To Statement Considered Harmful

Problem 2:

Finite State Machines are full of GOTO statements (transitions)

Solution candidate: Use Behavior Trees for Switching

Behavior Trees generalize other architectures

BTs generalize

- Finite State Machines
- Subsumption Architecture
- Teleo-Reactive Approach
- Decision Trees

BTs are simple!

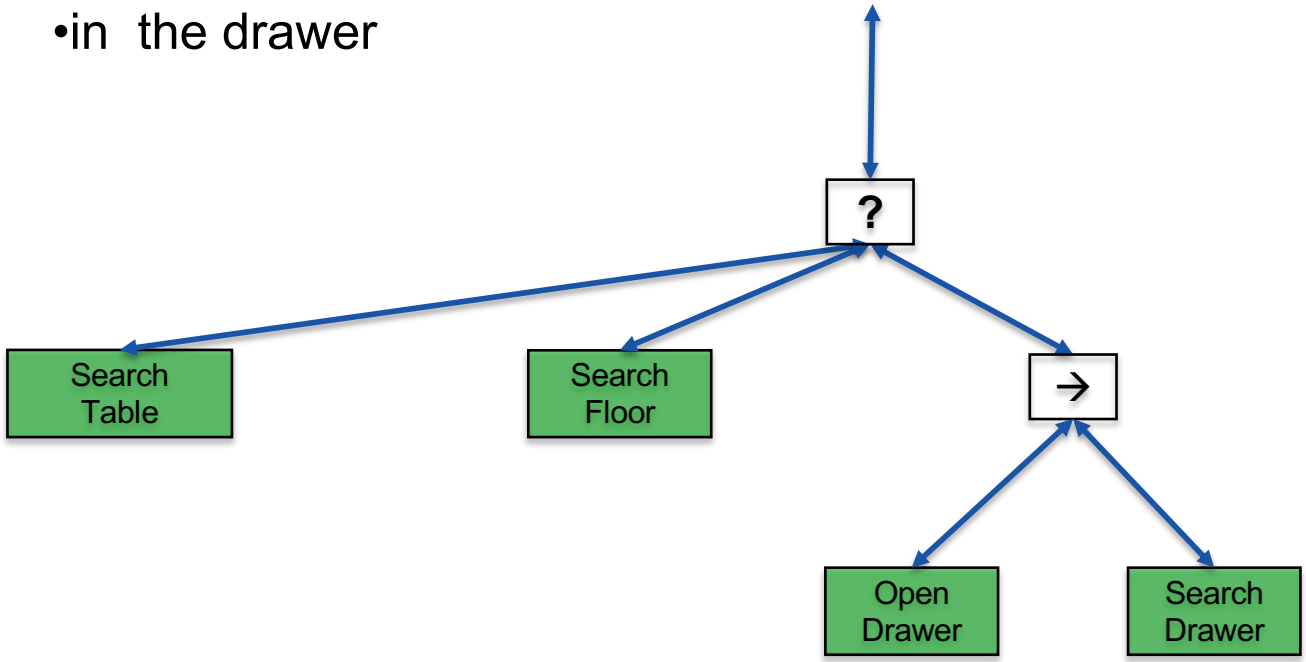
Complexity comes from composition...

Outline

- What do we mean by task switching?
- Quotes on BTs
- Why BTs?
- **How do BTs work?**
- 5 Efficient Design Principles for BTs
- Alternatives to BTs
- The Big Picture
  - BTs and Planning
  - BTs and Control Theory
  - BTs and Genetic Algorithms
  - BTs and Deep Learning

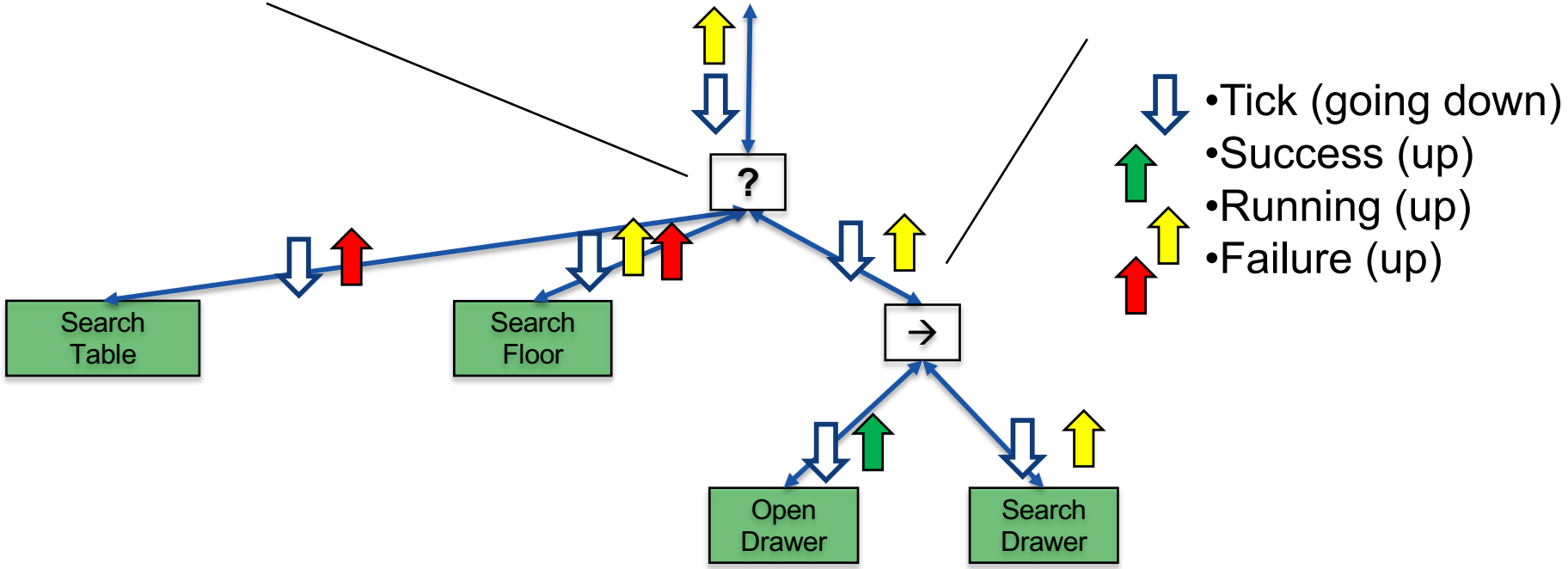
How does the BT work?

- Example: Look for keys
  - on the table
  - on the floor
  - in the drawer

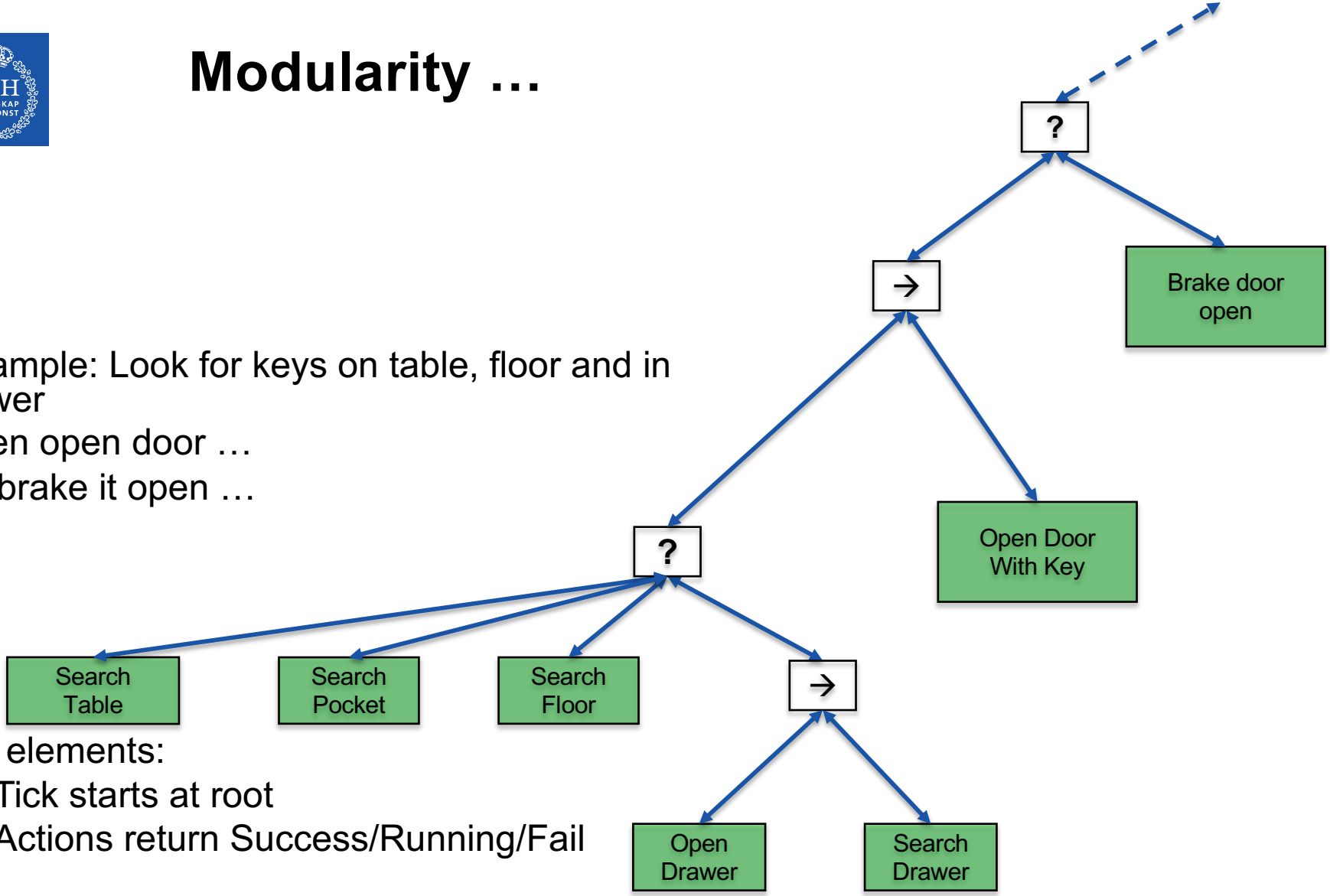


How does the BT work?

- Fallback
  - Stops at the first child that returns *success* or running
- Sequence
  - Stops at the first child that returns *failure* or running



Modularity …



•Example: Look for keys on table, floor and in drawer

•Then open door …

•Or brake it open …

Key elements:

- Tick starts at root
- Actions return Success/Running/Fail

•Leads to:

- Subtree needs no info about context

Additional Node types …

Parallell node

- Ticks all children
- Success/Fail at given %

Decorators

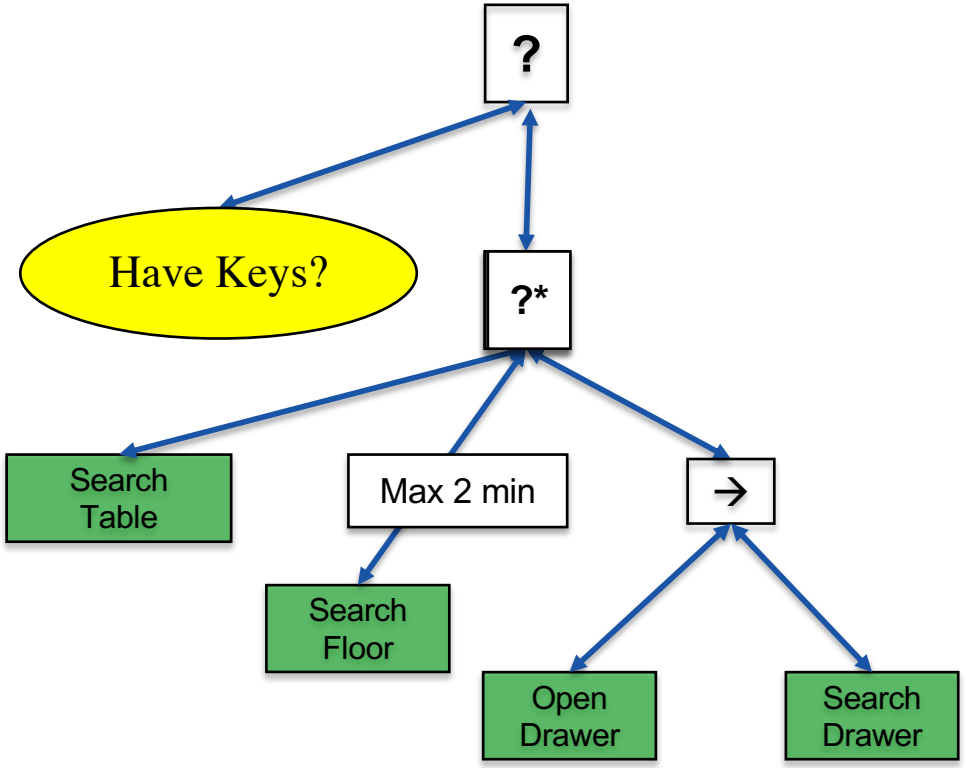
- Alters execution and/or return statement of child

Sequence/Fallback with memory

- Ticks the last running

Conditions

- Only Success/Fail



Outline

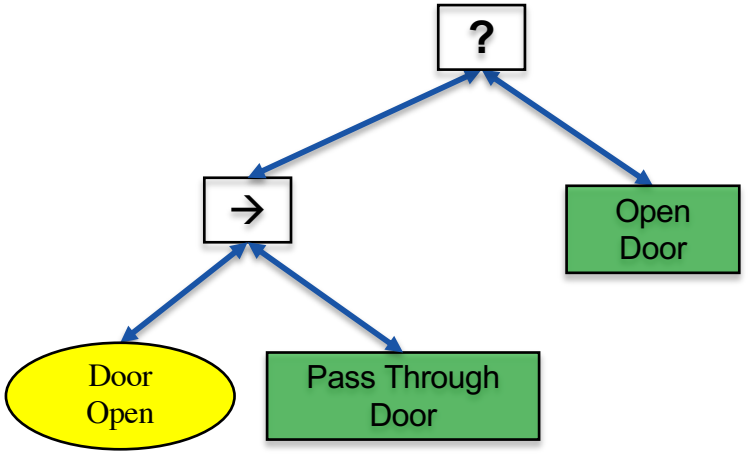
- What do we mean by task switching?
- Quotes on BTs
- Why BTs?
- How do BTs work?
- **5 Efficient Design Principles for BTs**
- Alternatives to BTs
- The Big Picture
  - BTs and Planning
  - BTs and Control Theory
  - BTs and Genetic Algorithms
  - BTs and Deep Learning

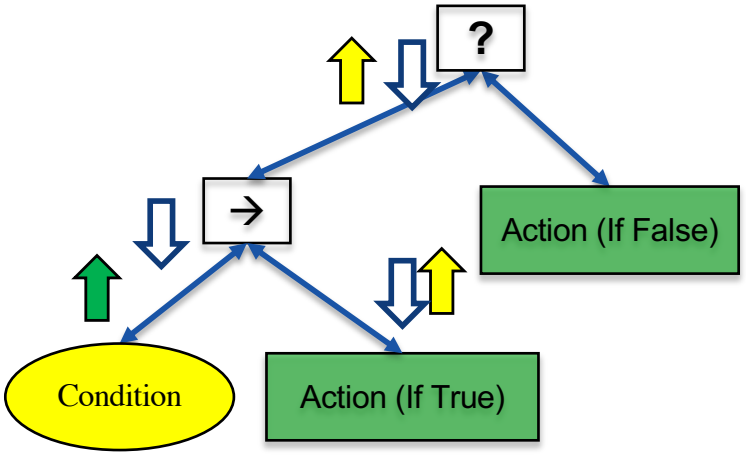
5 Efficient Design Principles

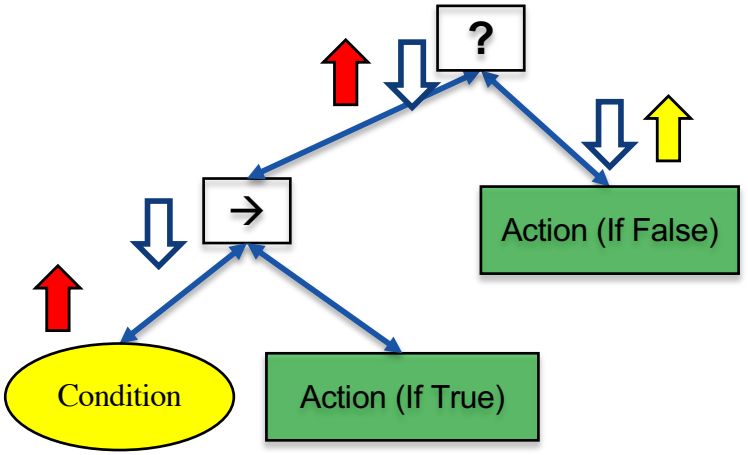
1. If-then-else
2. Explicit Success Conditions (→ Improves Readability)
3. Implicit Sequences (→ Improves Reactivity)
4. Decision Tree Principle (→ Divide and Conquer)
5. Sequences (→ Improve Safety)
6. Back-Chaining (→ Goal Directed)

If-then-else constructs

- How to do If-then-else?
- If True…
- If False …

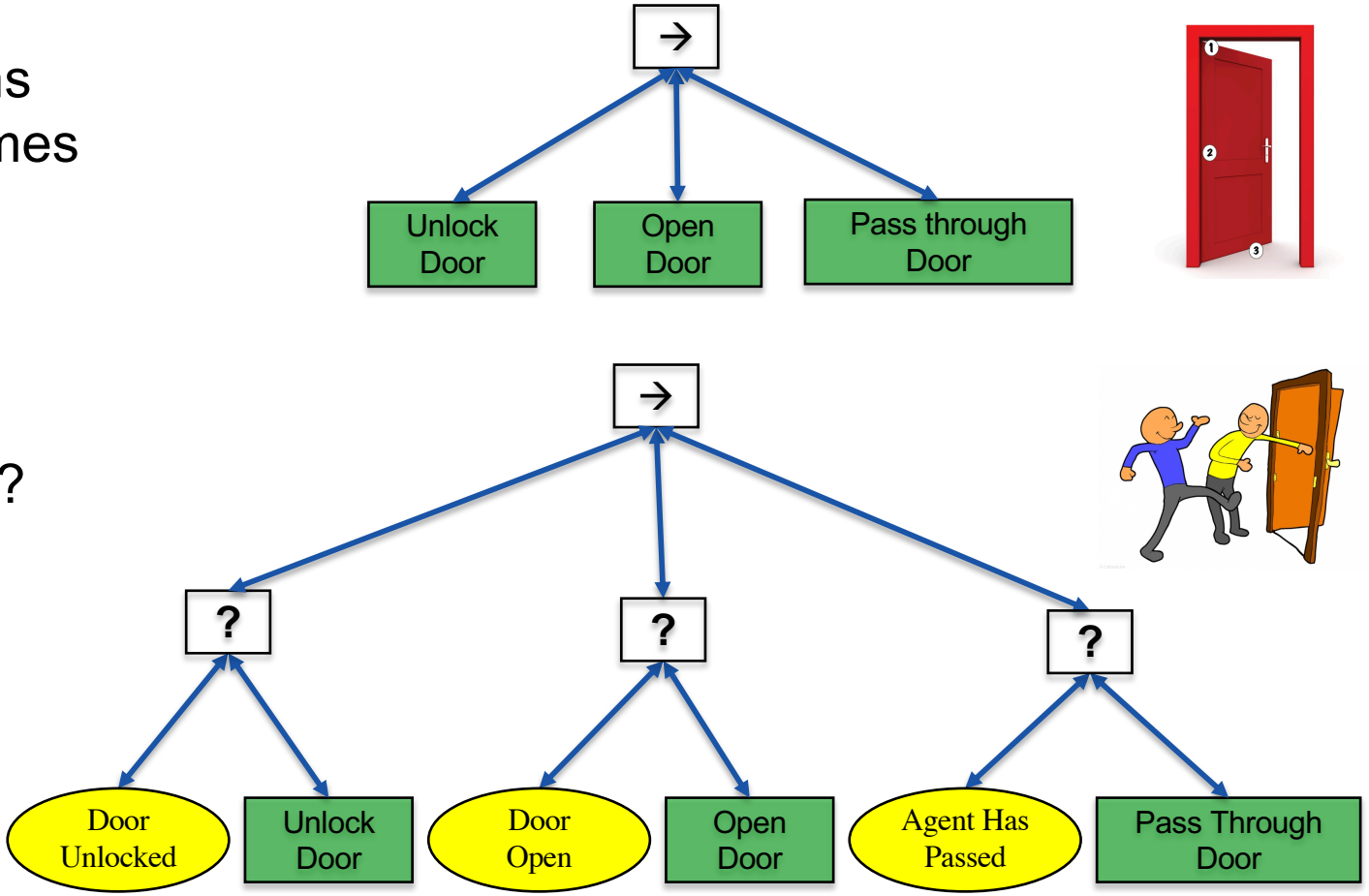






Explicit Success Conditions → Improves Readability

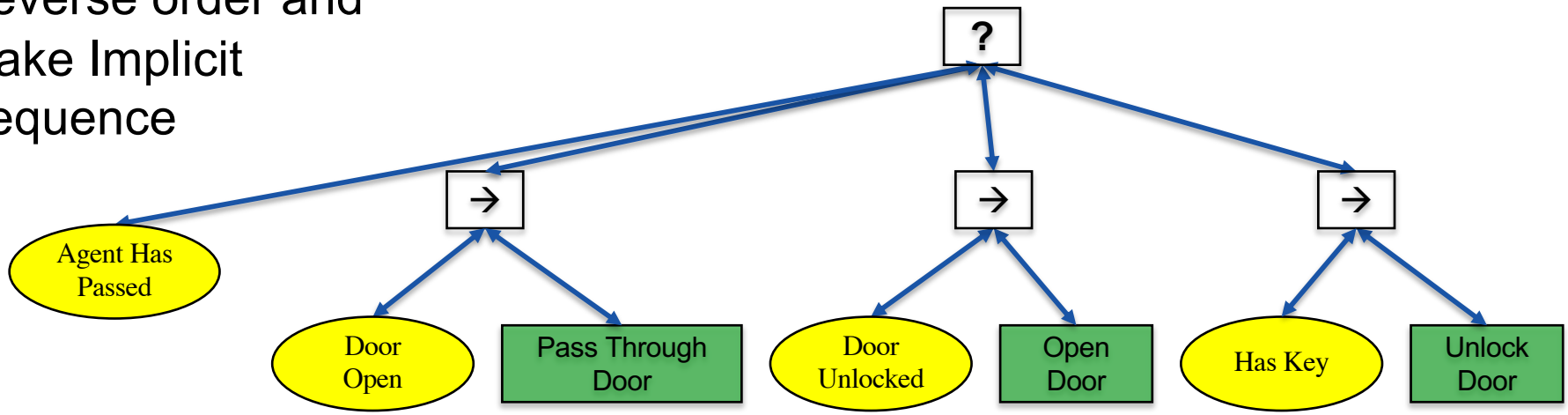
- Success conditions of Actions sometimes Unclear
  - For how long does unlock return success after unlocking?
- Explicit conditions Clarifies



Implicit Sequences → Improves Reactivity

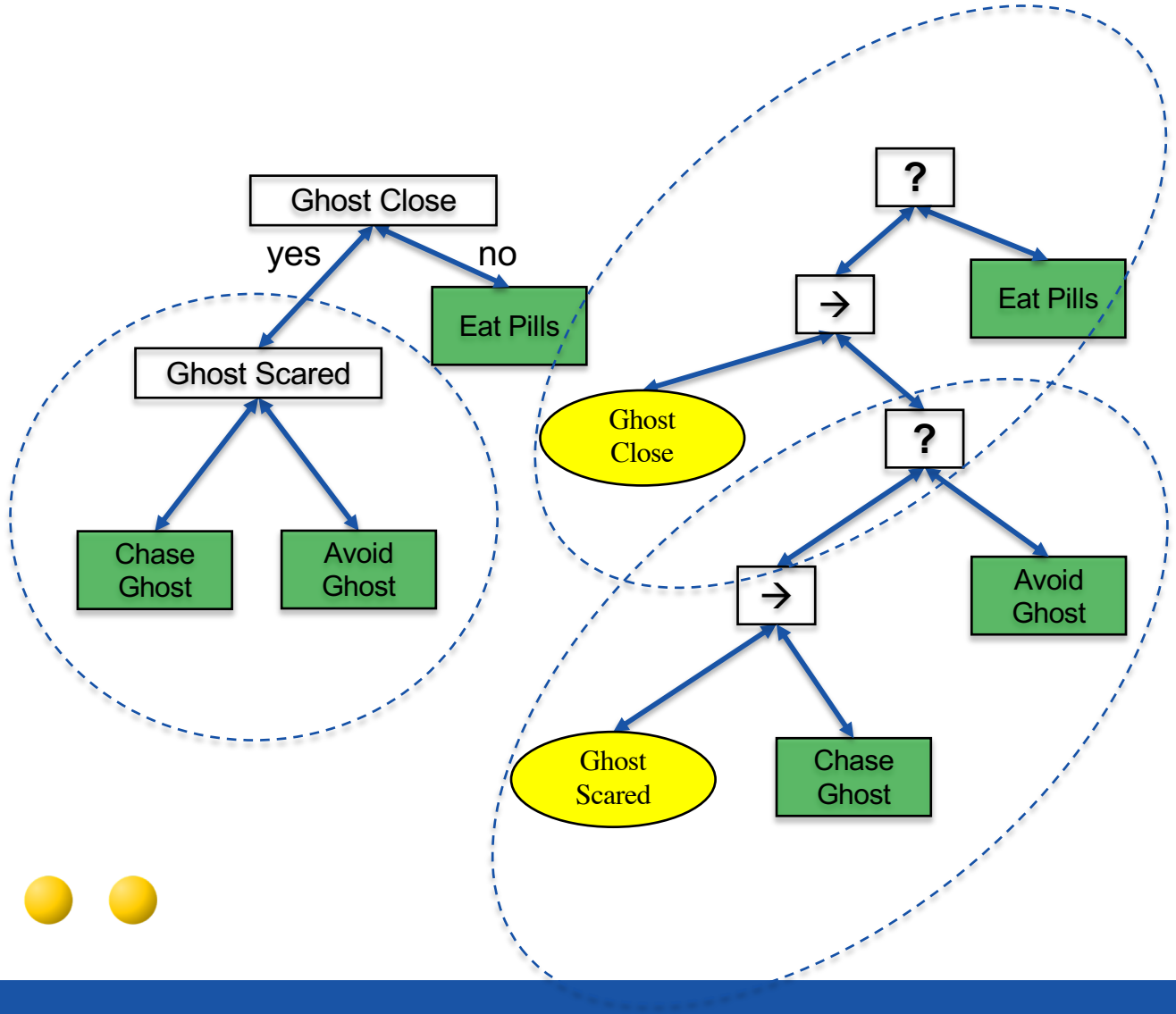
- What if Door is Closed, but Agent has already passed?

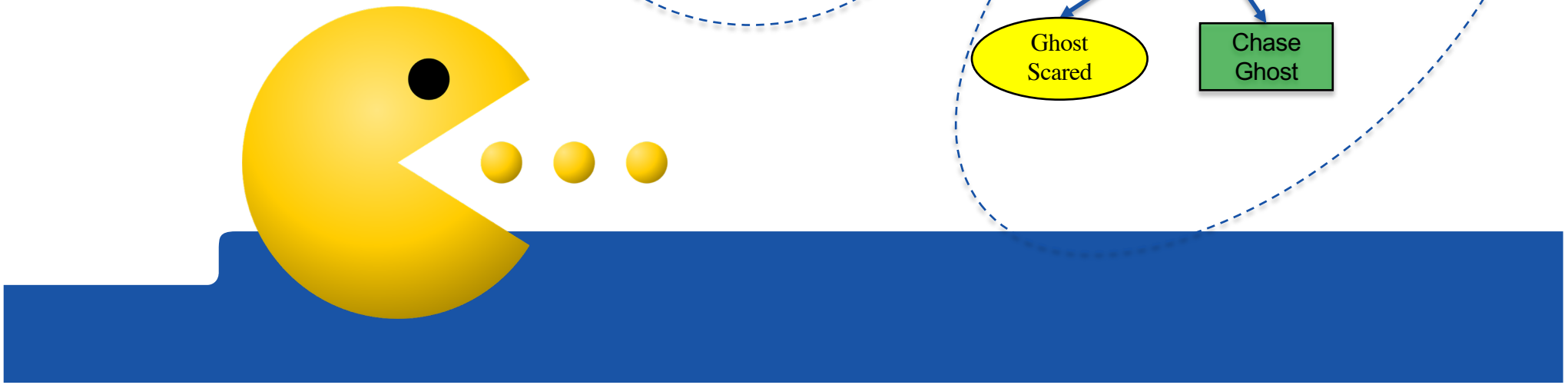
- Reverse order and make Implicit Sequence



Decision Tree Principle → Divide and Conquer

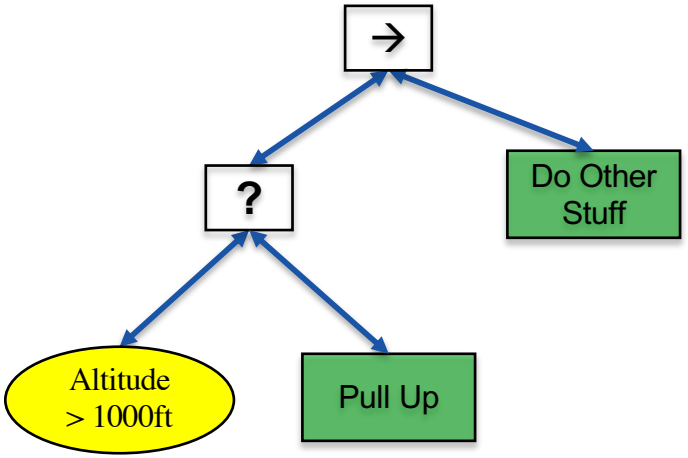
- Can Behavior be divided into Cases? (and sub-cases)
- Think “Decision Tree”
- Use If-then-else…





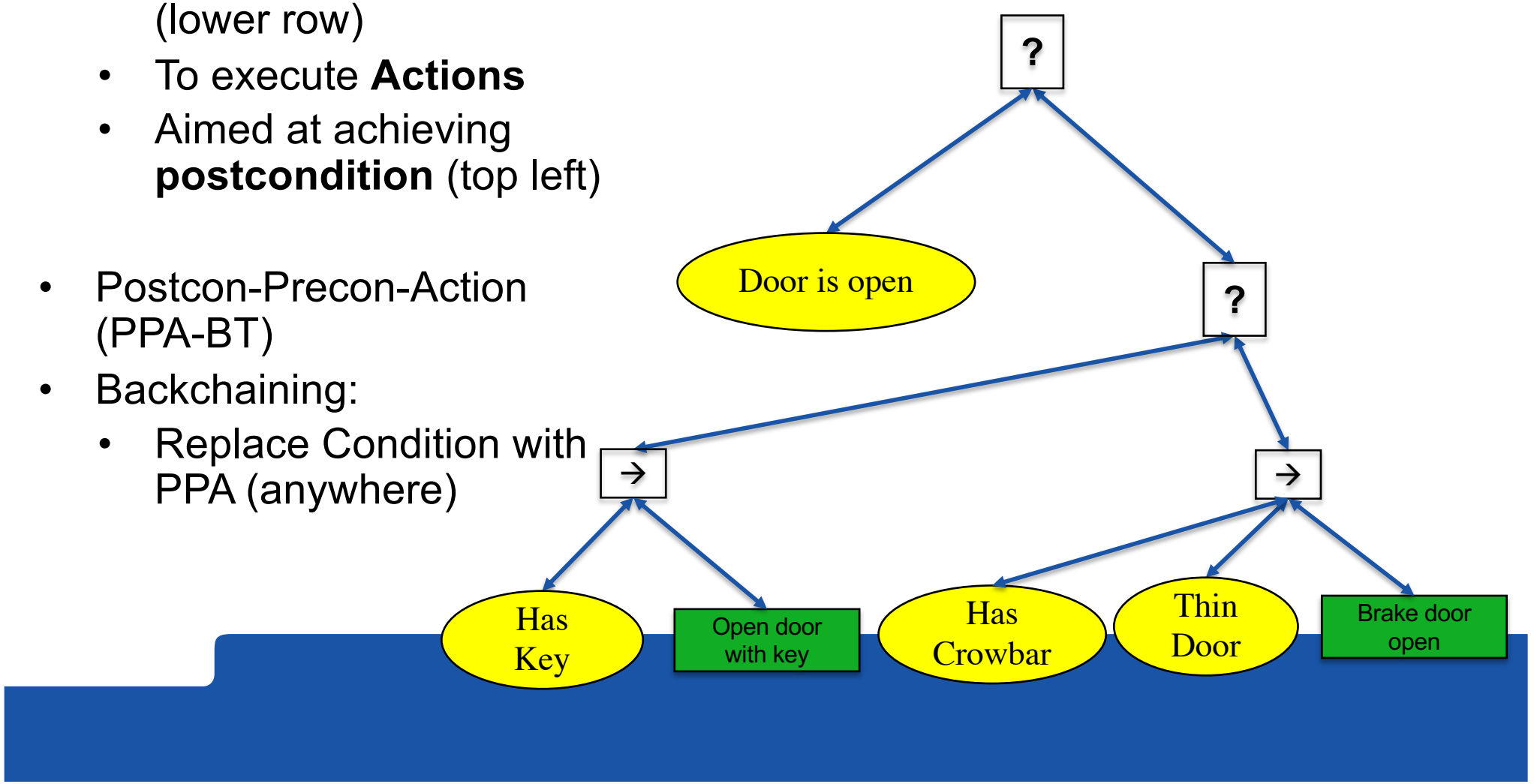
Sequences → Improve Safety

- BTs enable Safety-Guarantees using the following construction...
- If-not-then-else…
- Being checked every 0.1s (at 10Hz)



PPA Structure → Goal Directed BTs

- This BT
  - Checks **preconditions** (lower row)
  - To execute **Actions**
  - Aimed at achieving **postcondition** (top left)
- Postcon-Precon-Action (PPA-BT)
- Backchaining:
  - Replace Condition with PPA (anywhere)

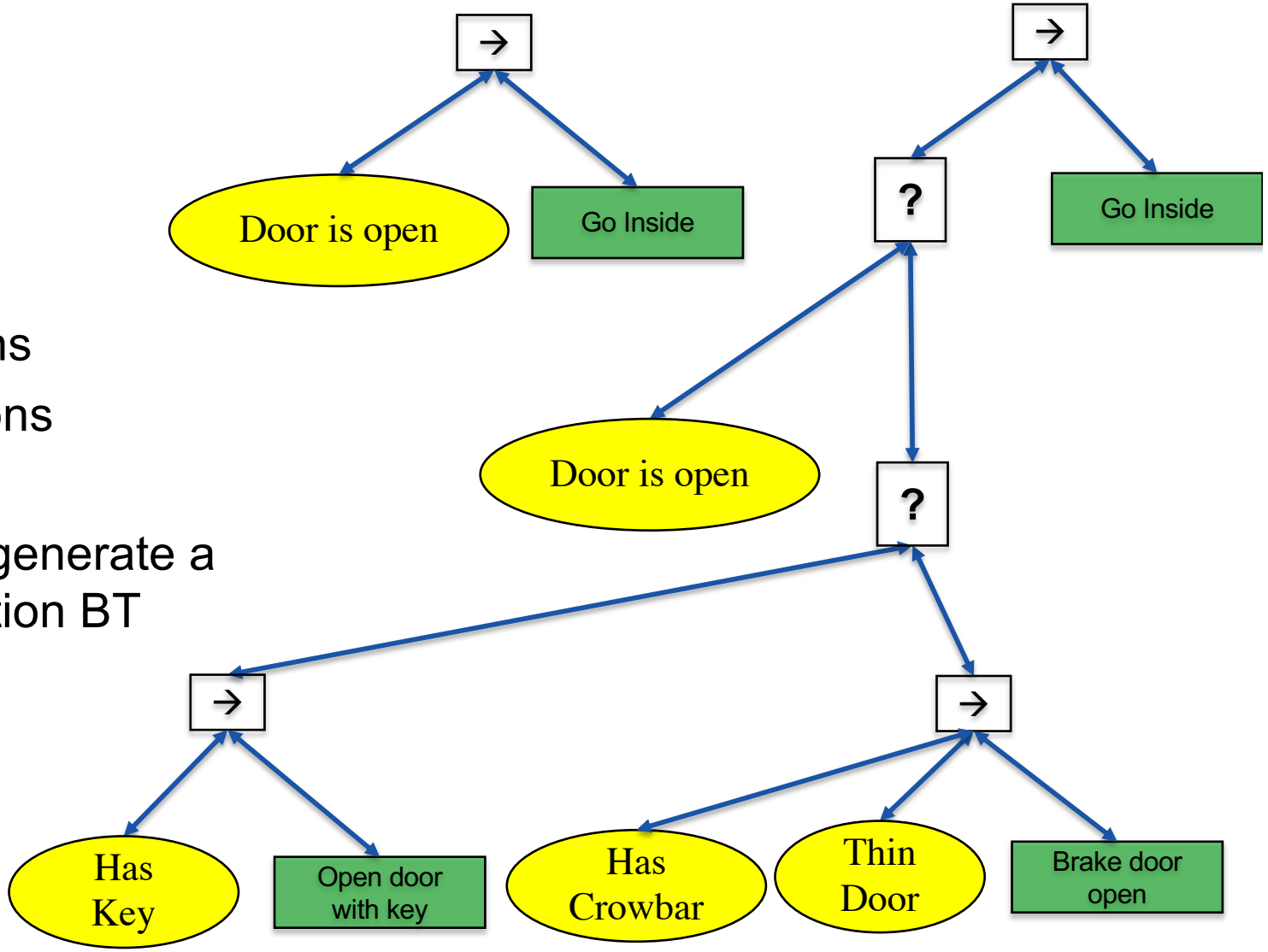


PPA Structure → Goal Directed BTs

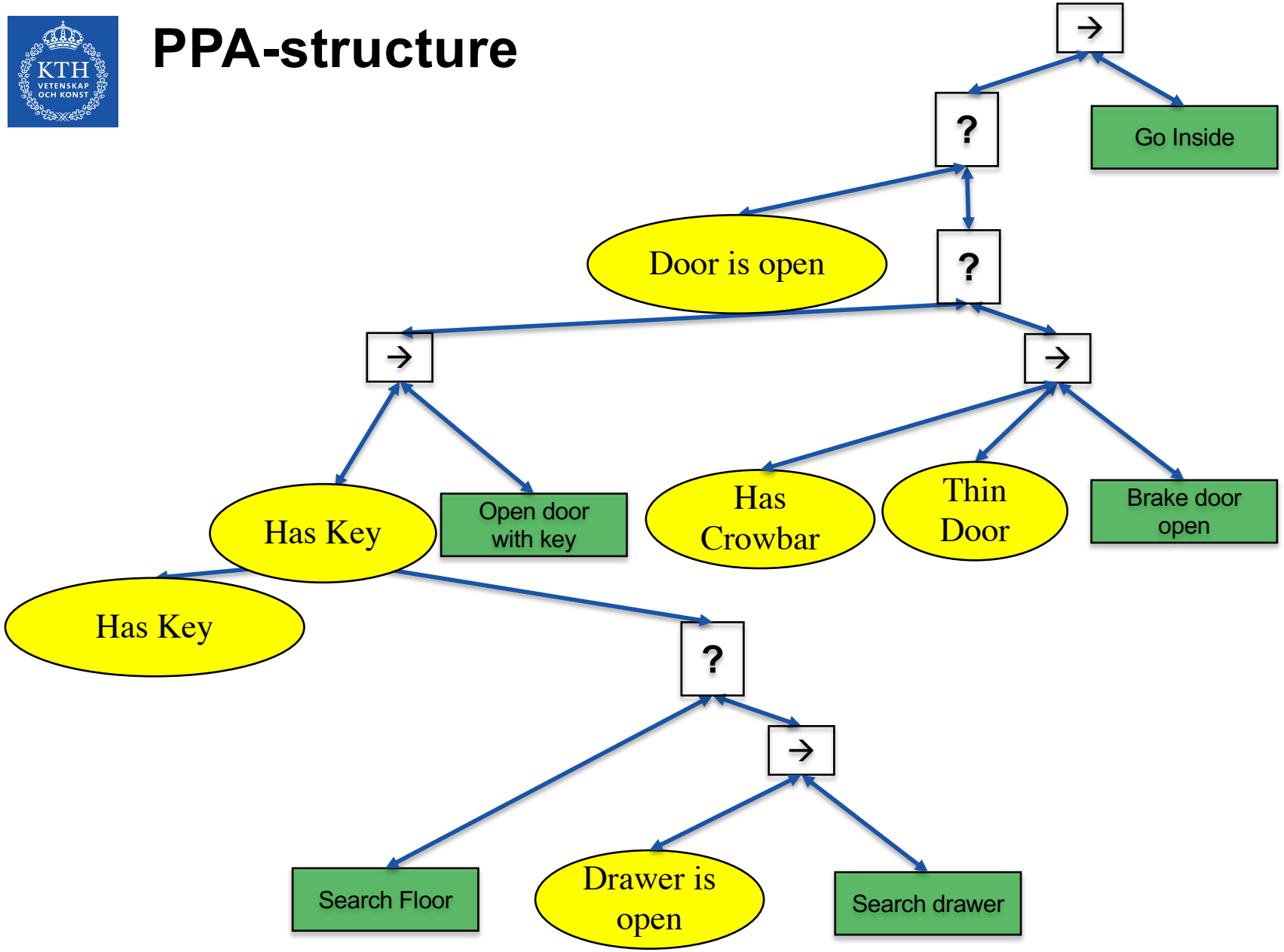
Given:

- Actions
  - preconditions
  - postconditions

- A planner can generate a reactive execution BT



PPA-structure

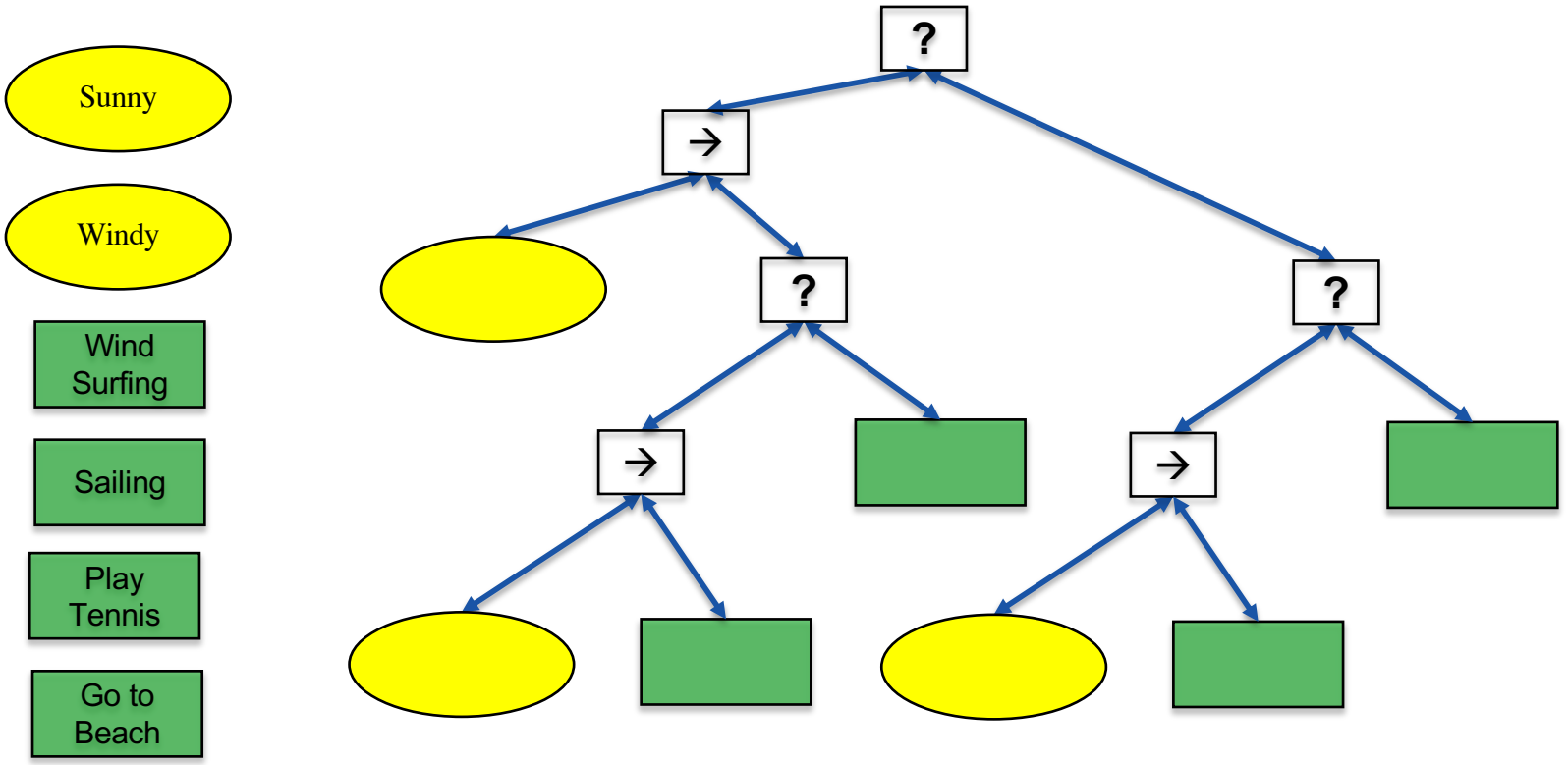


Design a BT

- We will create a BT including all these design principles
  - Decision tree design
  - Implicit Sequences
  - Safety
  - PPA (Backchaining, Postcondition-Precondition-Action)

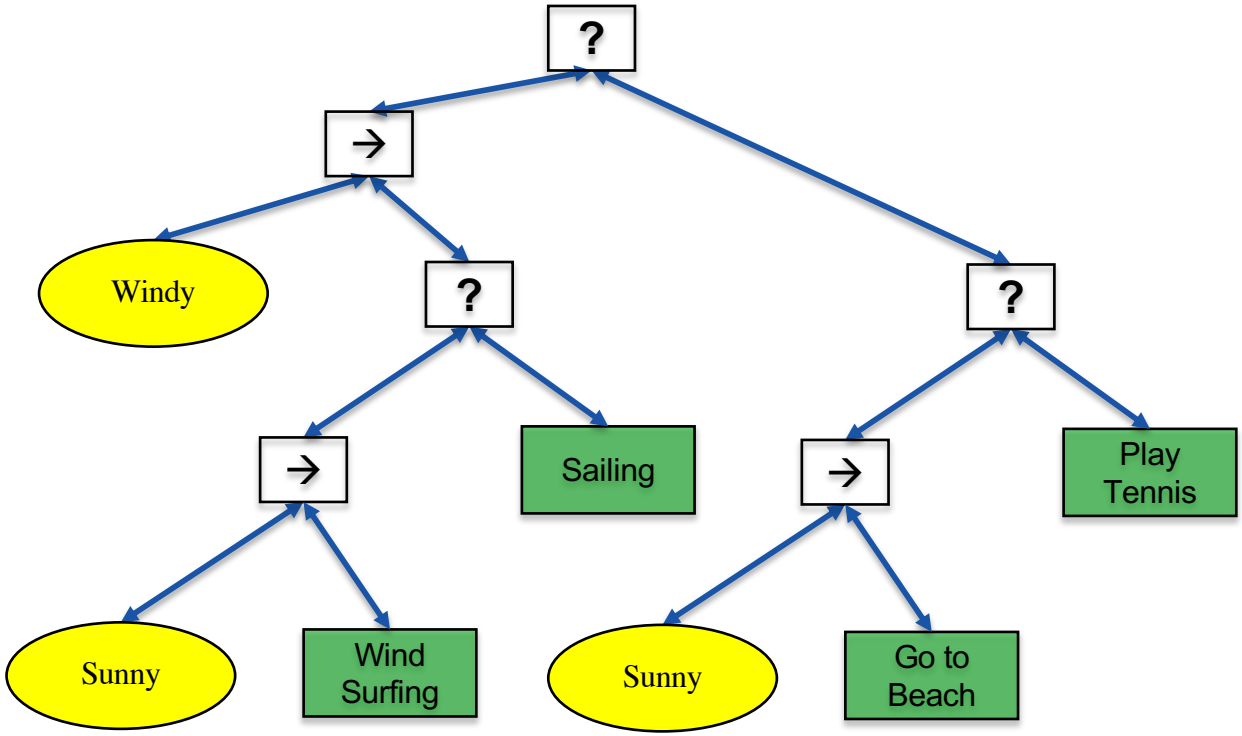
Decision Tree design

	Sunny	Not Sunny
Windy	Wind surfing	Sailing
Not Windy	Go to Beach	Tennis

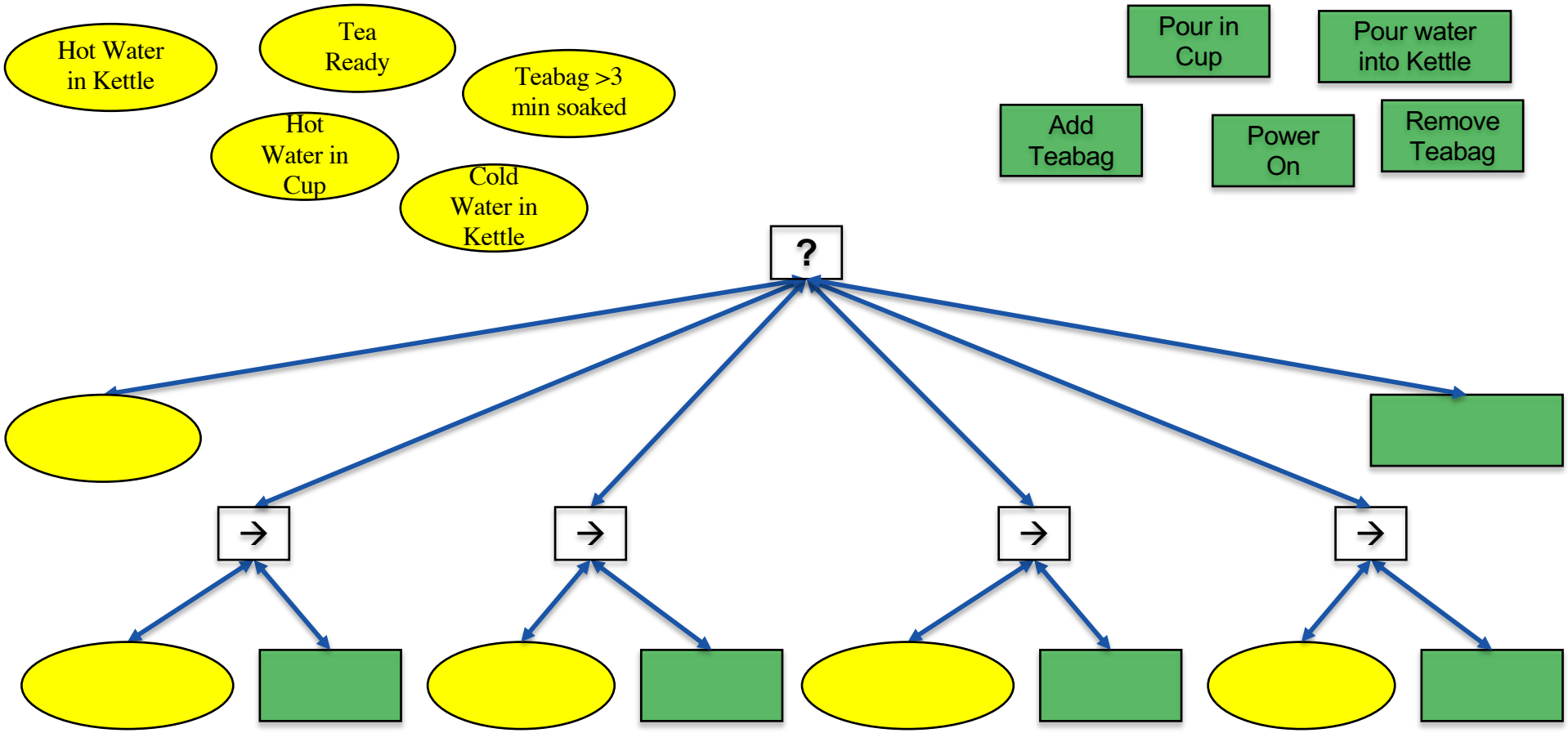


Decision Tree design

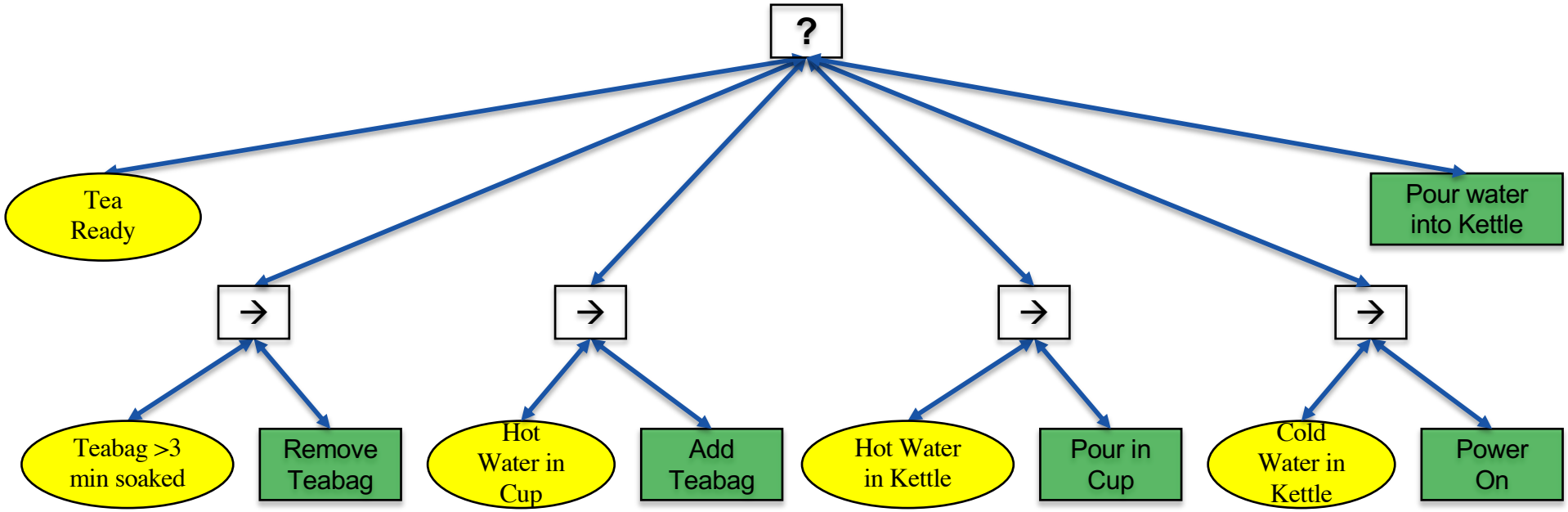
	Sunny	Not Sunny
Windy	Wind surfing	Sailing
Not Windy	Go to Beach	Tennis



Implicit Sequence Design: Making Tea

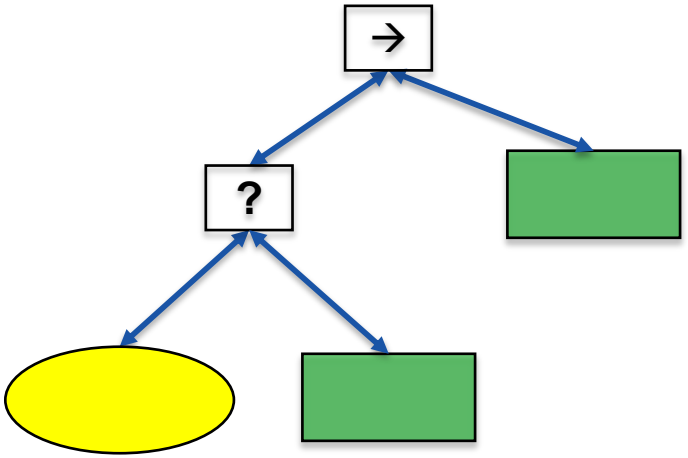


Implicit Sequence Design: Making Tea

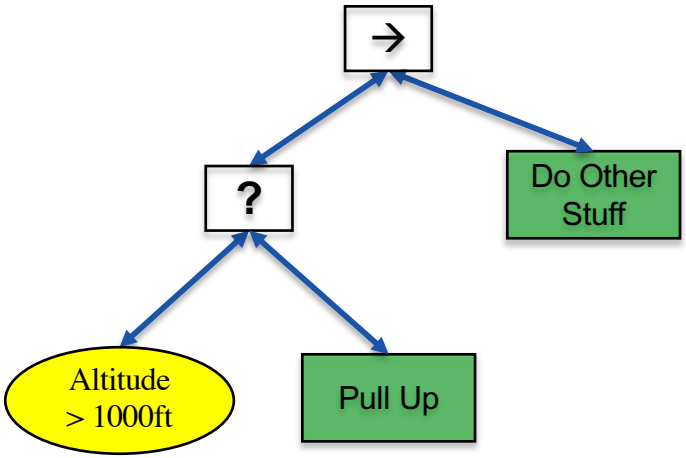


Sequences → Improve Safety

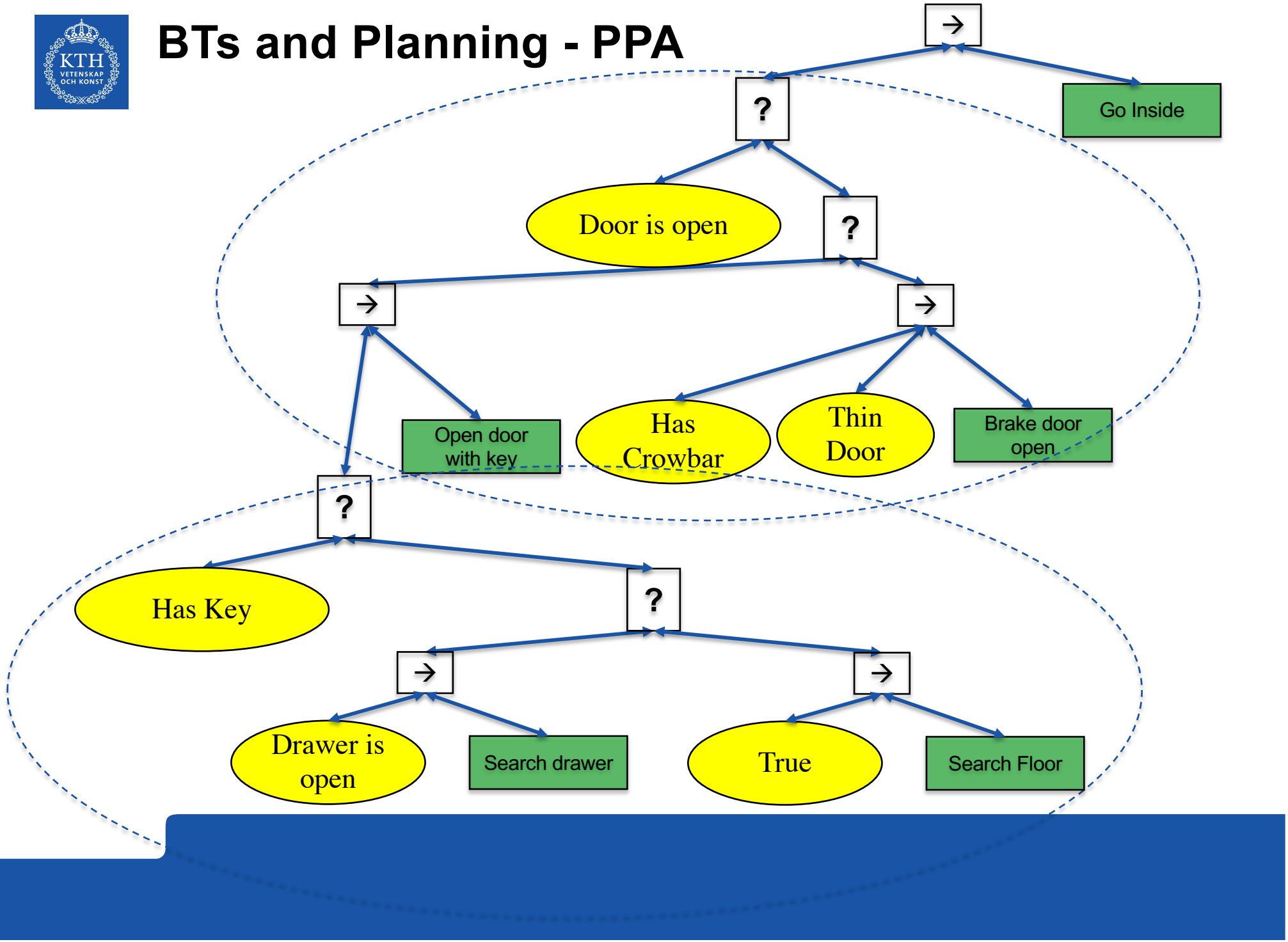
- Your safety example here…



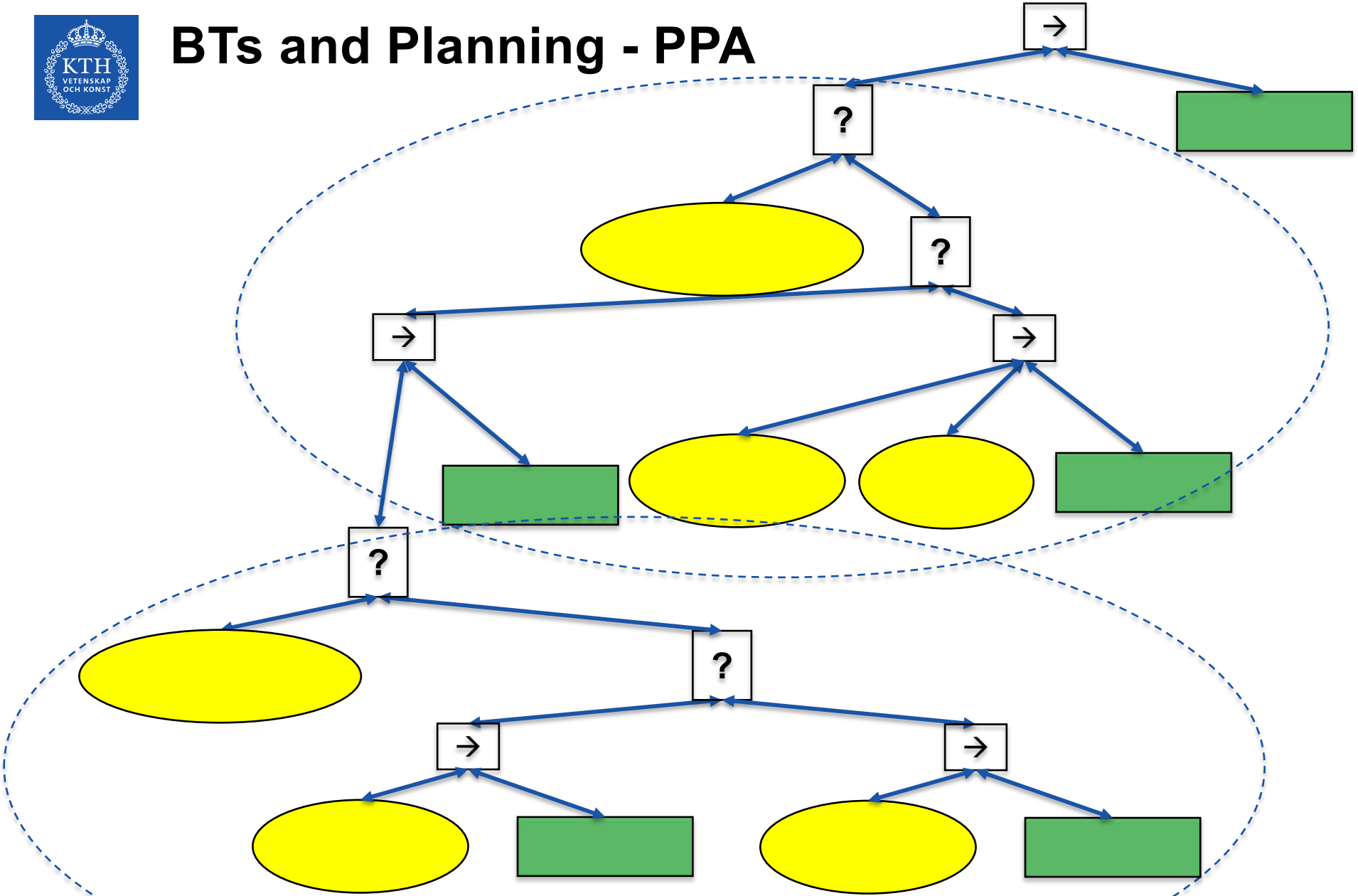
Sequences → Improve Safety



BTs and Planning - PPA



BTs and Planning - PPA



Outline

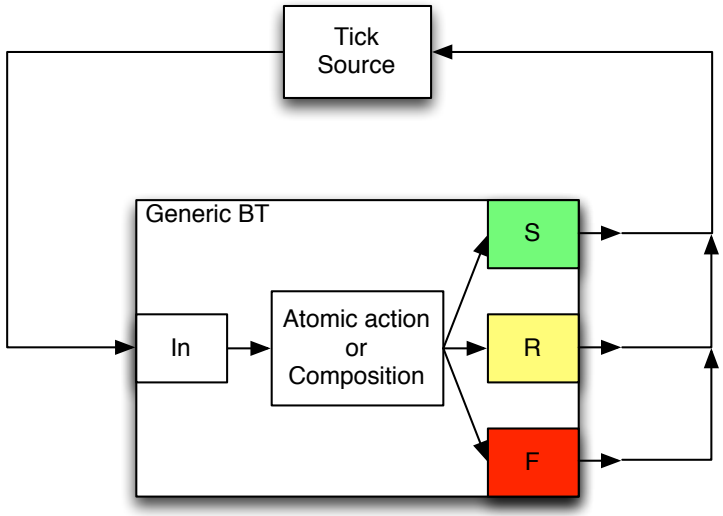
- What do we mean by task switching?
- Quotes on BTs
- Why BTs?
- How do BTs work?
- 5 Efficient Design Principles for BTs
- **Alternatives to BTs**
- The Big Picture
  - BTs and Planning
  - BTs and Control Theory
  - BTs and Genetic Algorithms
  - BTs and Deep Learning

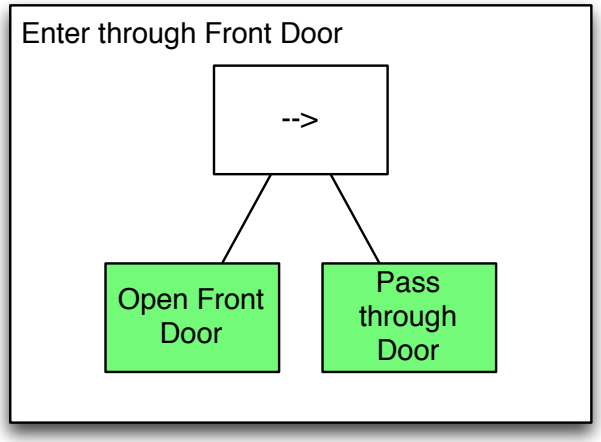
Alternative Approaches to Task Switching

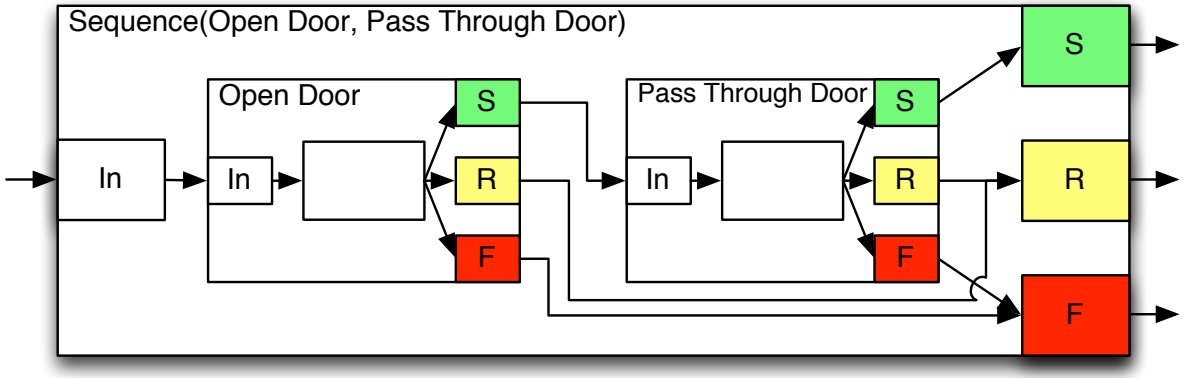
- Finite State Machines (FSM)
- Hierarchical FSMs
- Decision Trees
- Teleo-Reactive Approach
- Subsumption Architecture

Generalized by BTs

BTs as FSMs with special structure



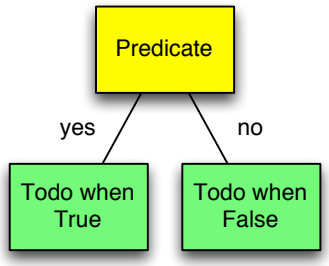


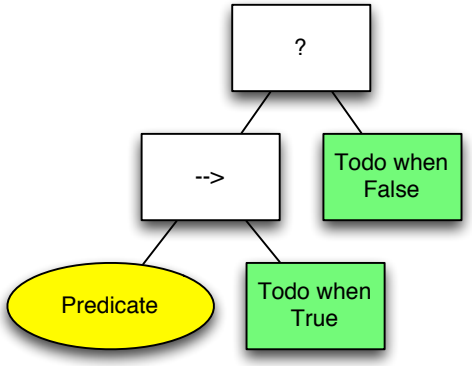


BTs generalizes Decision Trees

A decision tree (DT)

- actions at leaves
- predicates elsewhere



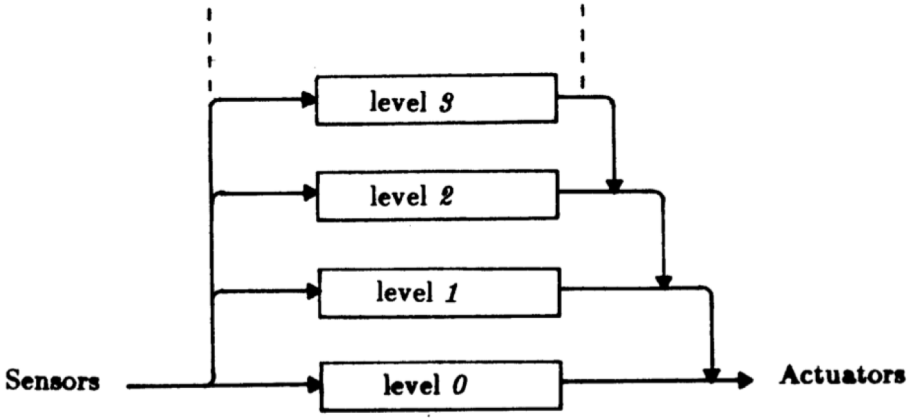


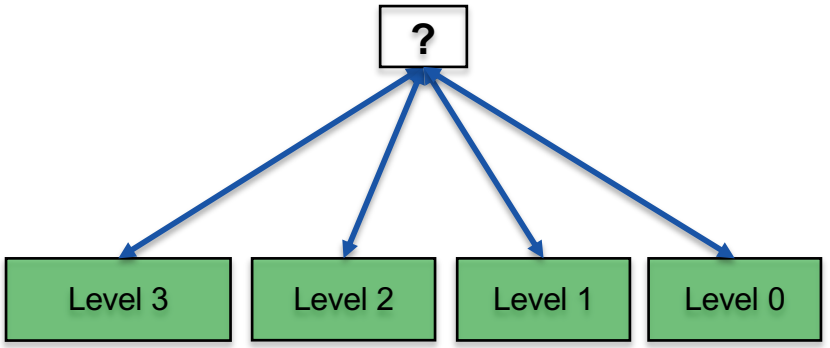
Each DT can be written as a BT by a simple mapping...

But **no return status** of actions, final decision has to be made at node

BTs generalize the Subsumption Architecture

Brooks, R. (1986). A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation*





But no Sequences and no Hierarchical compositions

BTs generalizes the Teleo-Reactive Approach of Nils Nilsson

Example:

“Goto goal in empty room”

Priority list of rules:

1. IF (at goal) run (stop)
2. IF (facing goal) run (go straight)
3. IF (true) run (turn on spot)
4. ....

Constantly evaluate conditions k_i

Execute corresponding a_i with highest priority

a_i can be sub-program

Set up such that a_j leads to satisfaction of k_i, $i<j$ (regression property)

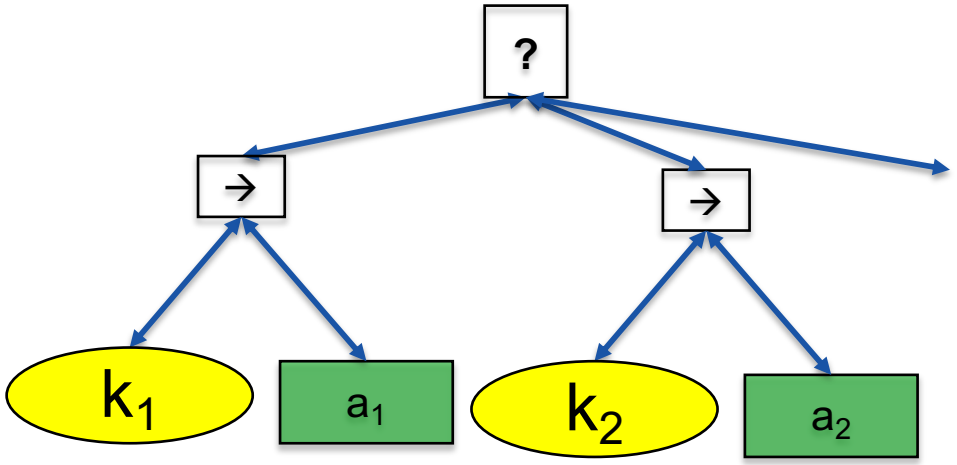
Note: IF all k_i cover entire statespace THEN k_1 will be achieved

Teleo-reactive Programs for agent control (Nilsson 1994), 369 citations

How BTs generalize Teleoreactive programs

Teleoreactive programs:

1. IF k_1 run a_1
2. IF k_2 run a_2
3. IF k_3 run a_3
4. ....

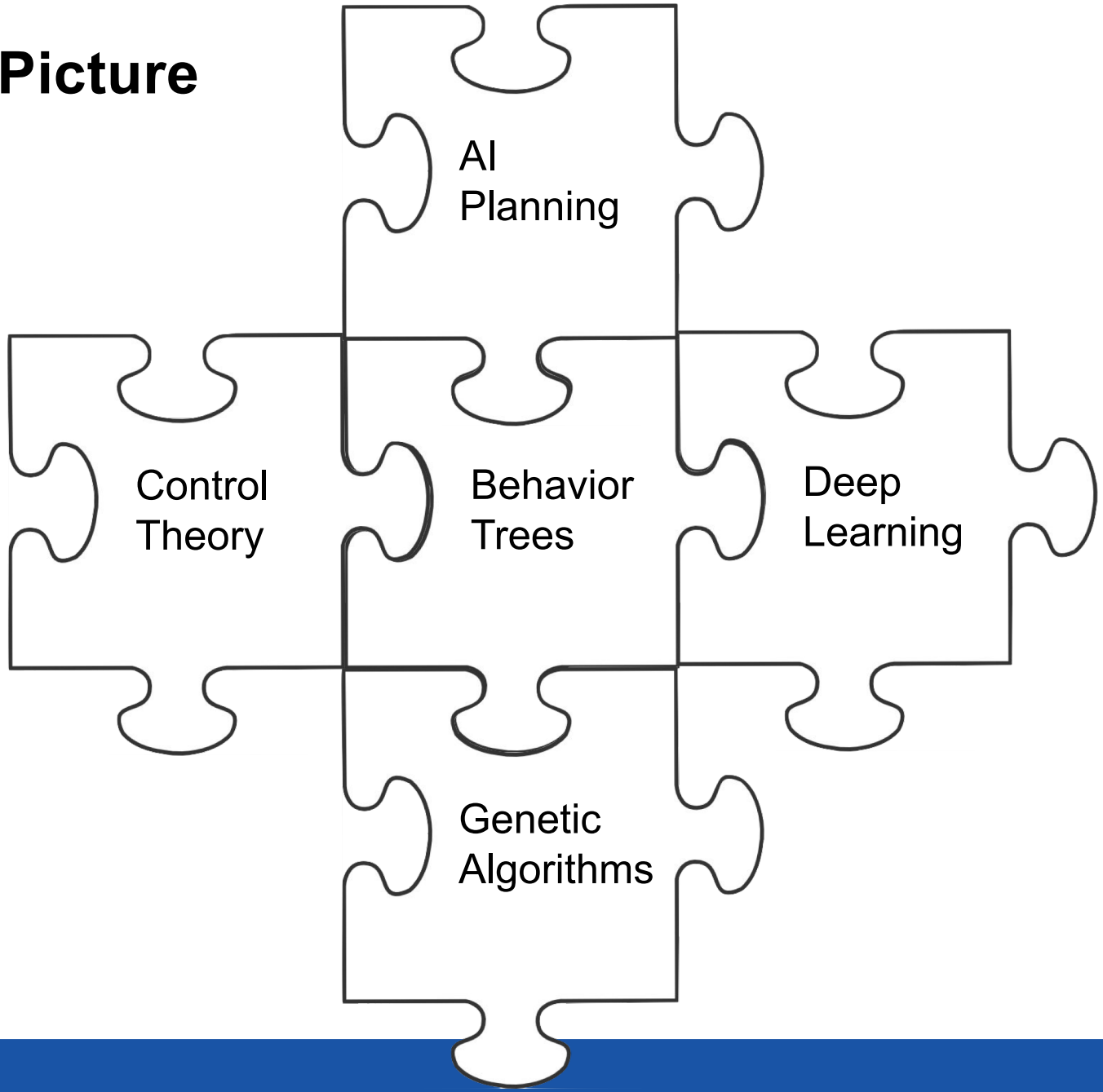


However, No return status

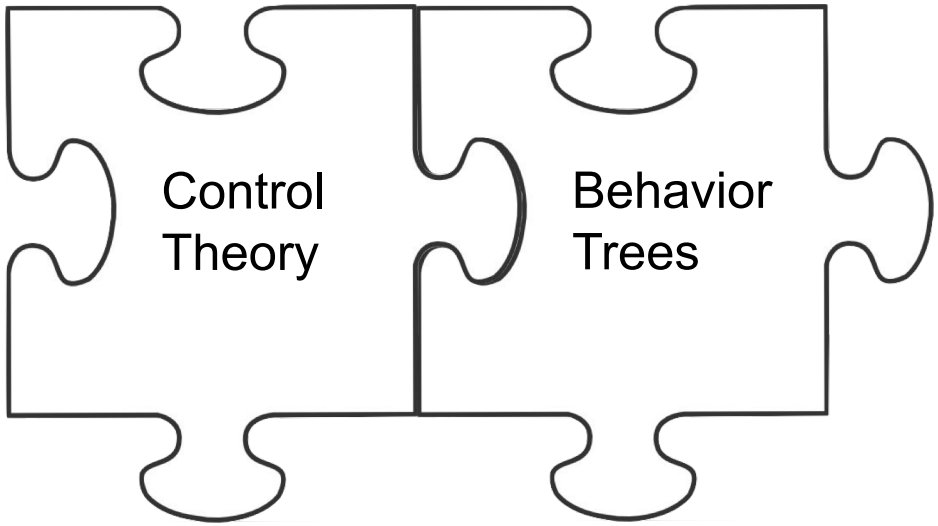
Outline

- What do we mean by task switching?
- Quotes on BTs
- Why BTs?
- How do BTs work?
- 5 Efficient Design Principles for BTs
- Alternatives to BTs
- **The Big Picture**
  - BTs and Control Theory
  - BTs and Planning
  - BTs and Genetic Algorithms
  - BTs and Deep Learning

The Big Picture



The Big Picture



- Hybrid Systems combine discrete and continuous dynamics
- BTs enable state space analysis of
  - Robustness
  - Safety
  - Efficiency

How Behavior Trees Modularize Hybrid Control Systems and Generalize Sequential Behavior Compositions, the Subsumption Architecture and Decision Trees

Michele Colledanchise, *Student Member, IEEE,* and Petter Ögren, *Member, IEEE*

***Abstract*—Behavior Trees (BTs) is a way of organizing the switching structure of a Hybrid Dynamical System (HDS), that was originally introduced in the computer game programming community. In this paper, we analyze how the BT representation increases the modularity of a HDS, and how key system properties are preserved over compositions of such systems, in terms of combining two BTs into a larger one. We also show how BTs can be seen as a generalization of Sequential Behavior Compositions, the Subsumption Architecture and Decisions Trees. These three tools are powerful, but quite different, and the fact that they are unified in a natural way in BTs might be a reason for their popularity in the gaming community. We conclude the paper by giving a set of examples illustrating how the proposed analysis tools can be applied to robot control BTs.**

***Index Terms*—Behavior Trees, Finite State Machines, Hybrid Dynamical Systems, Modularity, Subsumption Architecture, Sequential Behavior Compositions, Decision Trees**

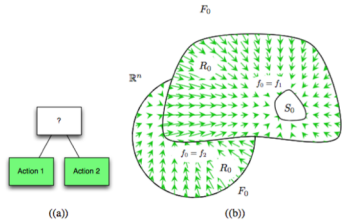


Fig. 1. A minimalist Behavior Tree composition (a) and the corresponding vector field (b). The second subtree increases the robustness of the composition by increasing the combined region of attraction.

I. Introduction

BEHAVIOR Trees (BTs) were developed in the computer gaming industry, as a tool to increase *modularity* in the control structures of in-game opponents [1]–[5]. In this billion dollar industry, modularity is a key property to enable reusability of code, incremental design of functionality and efficient testing of that functionality.

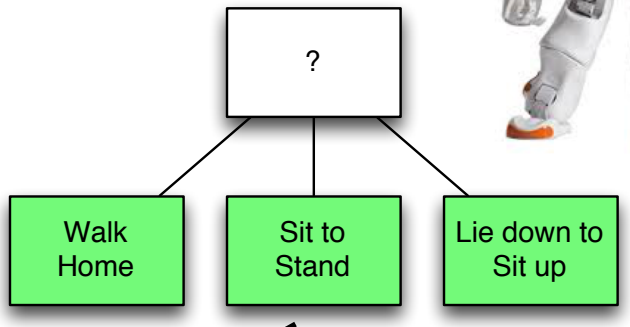
In games, the control structures of in-game opponents are naturally formulated in terms of Hybrid Dynamical Systems (HDSs), i.e. dynamical systems that have a continuous part, such as motion in a virtual environment, and a discrete part, such as decision making, in terms of switching between dif-

Following the development in industry, BTs have now also started to receive attention in academia, see e.g. [7]–[17]

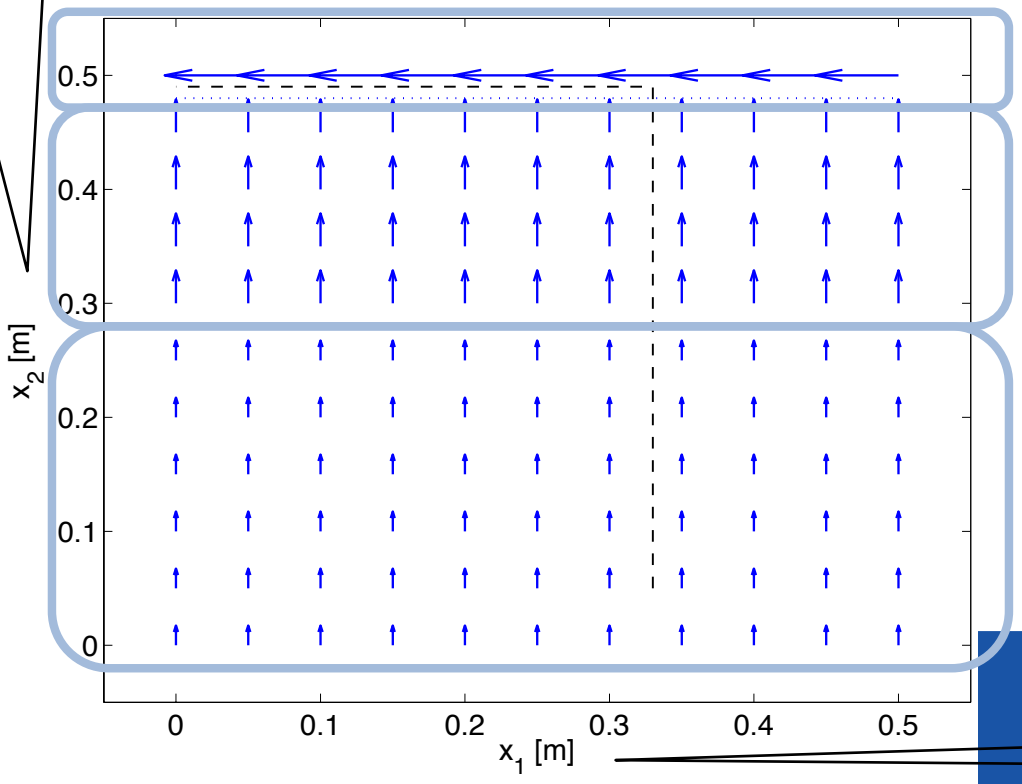
At Carnegie Mellon University, BTs have been used extensively to do robotic manipulation [12], [15]. The fact that modularity is the key reason for using BTs is clear from the following quote: "The main advantage is that individual behaviors can easily be reused in the context of another higher-level behavior, without needing to specify how they relate to subsequent behaviors." from [12].

How BTs can be used to extended the Region of attraction

Nao-robot “Get home” BT

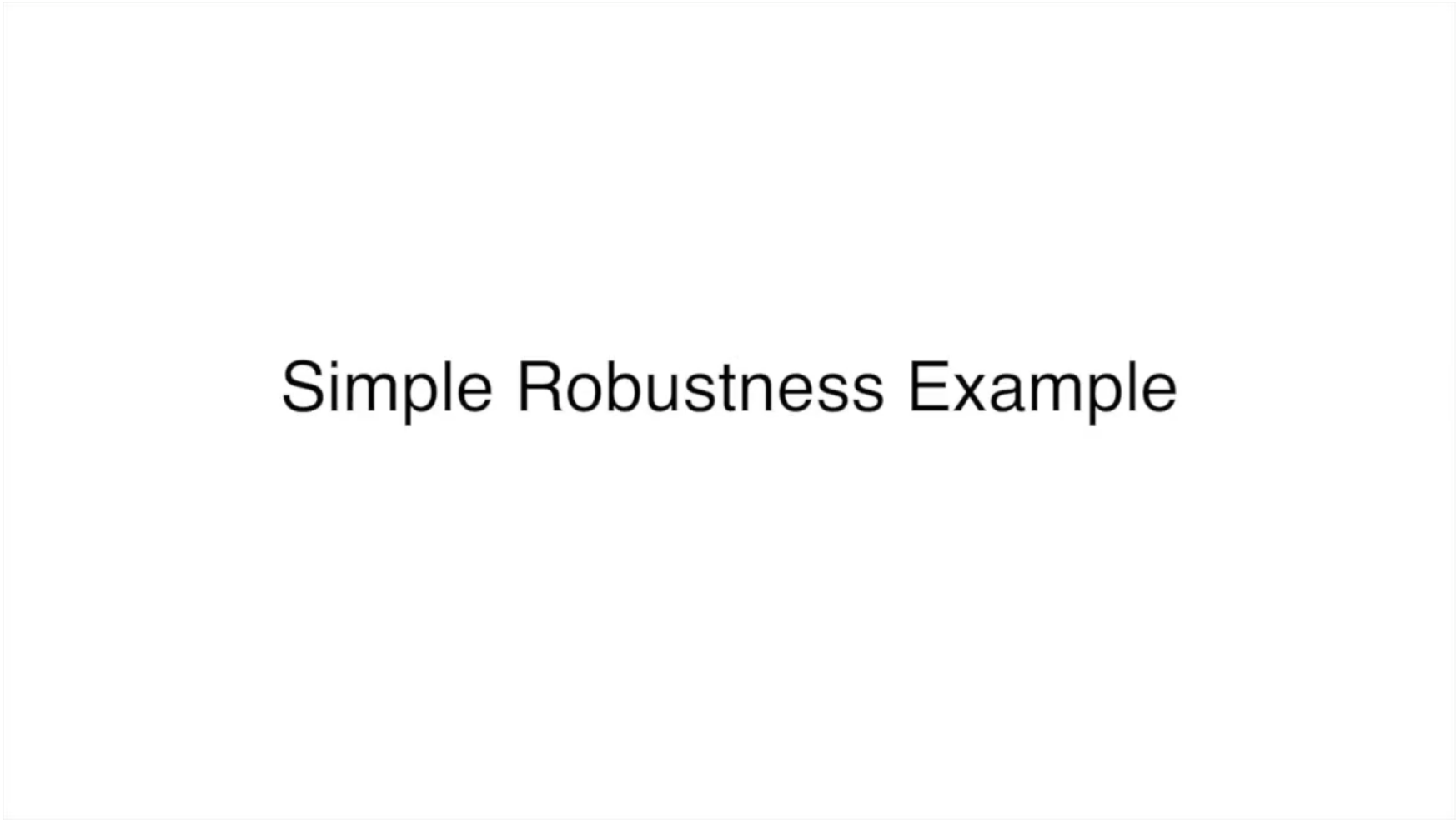


Height of head above the floor



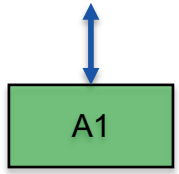
Pos

Example: Nao-robot “Move to”



How do we formalize this?

“Stability” of BTs



A typical problem in control theory:

- Prove Stability
- Design Stabilizing Controller

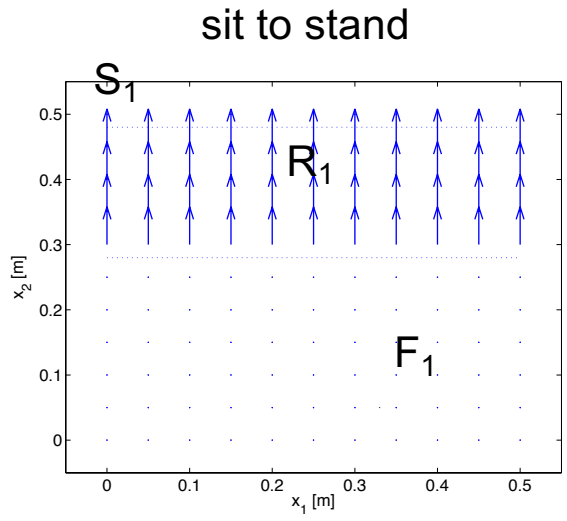
What does this correspond to in BTs?

Returning “Success”

… in Finite Time

- (good for analyzing compositions)

“Stability” of BTs



Given a BT A_1

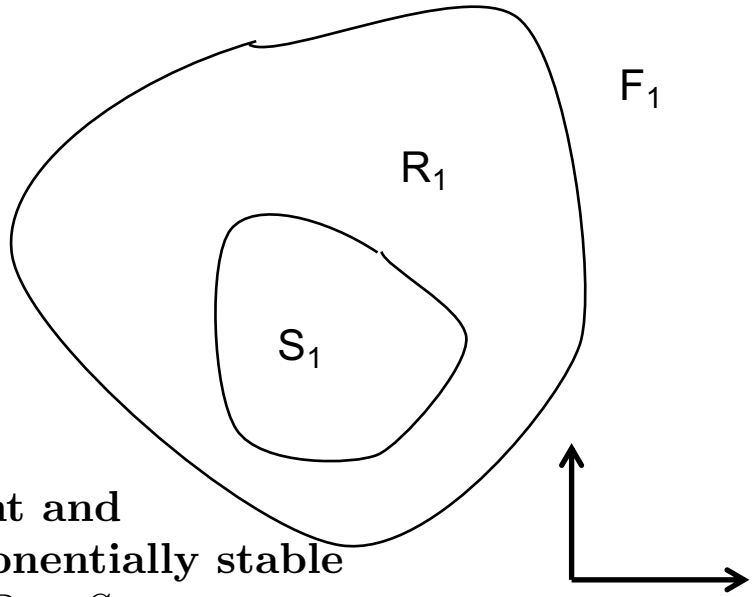
Partition the **Statespace** into regions corresponding to the return statuses

- S_1 (success)
- R_1 (running)
- F_1 (failure)

What is the design goal of a BT?

Def: **Finite Time Successful** (FTS)

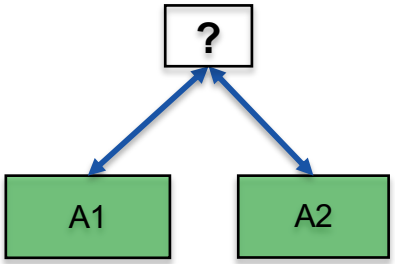
- If system starts in R_1
- Then it reaches S_1 before t=T
- Without entering F_1



Holds If $R_1 \cup S_1$ is invariant and equilibrium $x_1 \subset S_1$ is exponentially stable with region of attraction $R_1 \cup S_1$

“Stability” of BT compositions

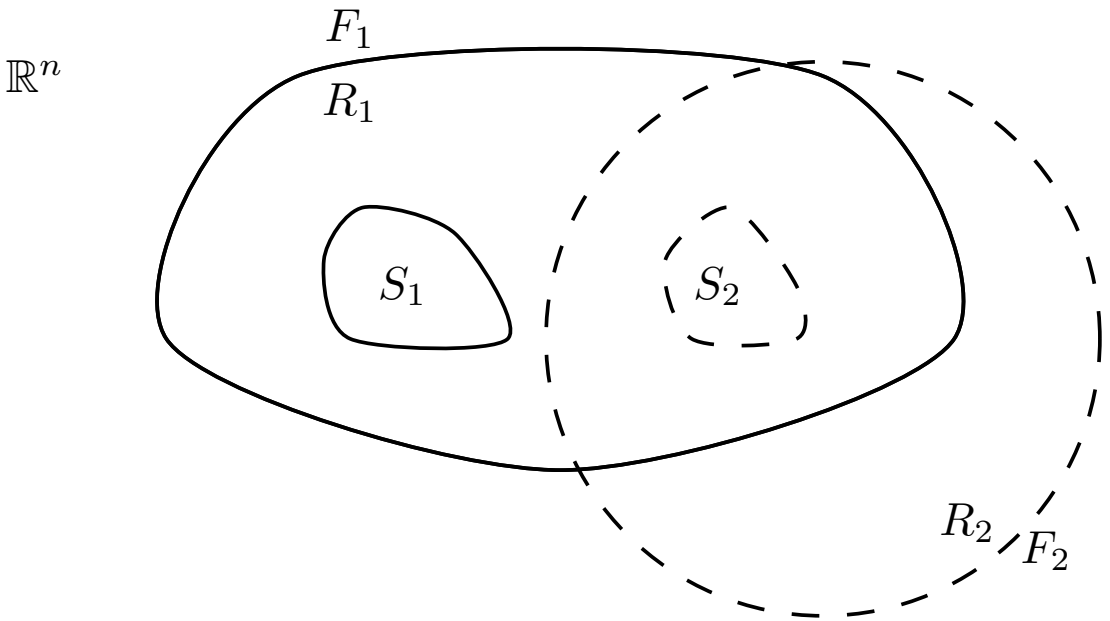
Teleoreactive programs are Fallback compositions...



Given a Fallback composition

Assume the following sets S/F/R

What can we say about the composition?



“Stability” of BT compositions

What can we say about the composition?

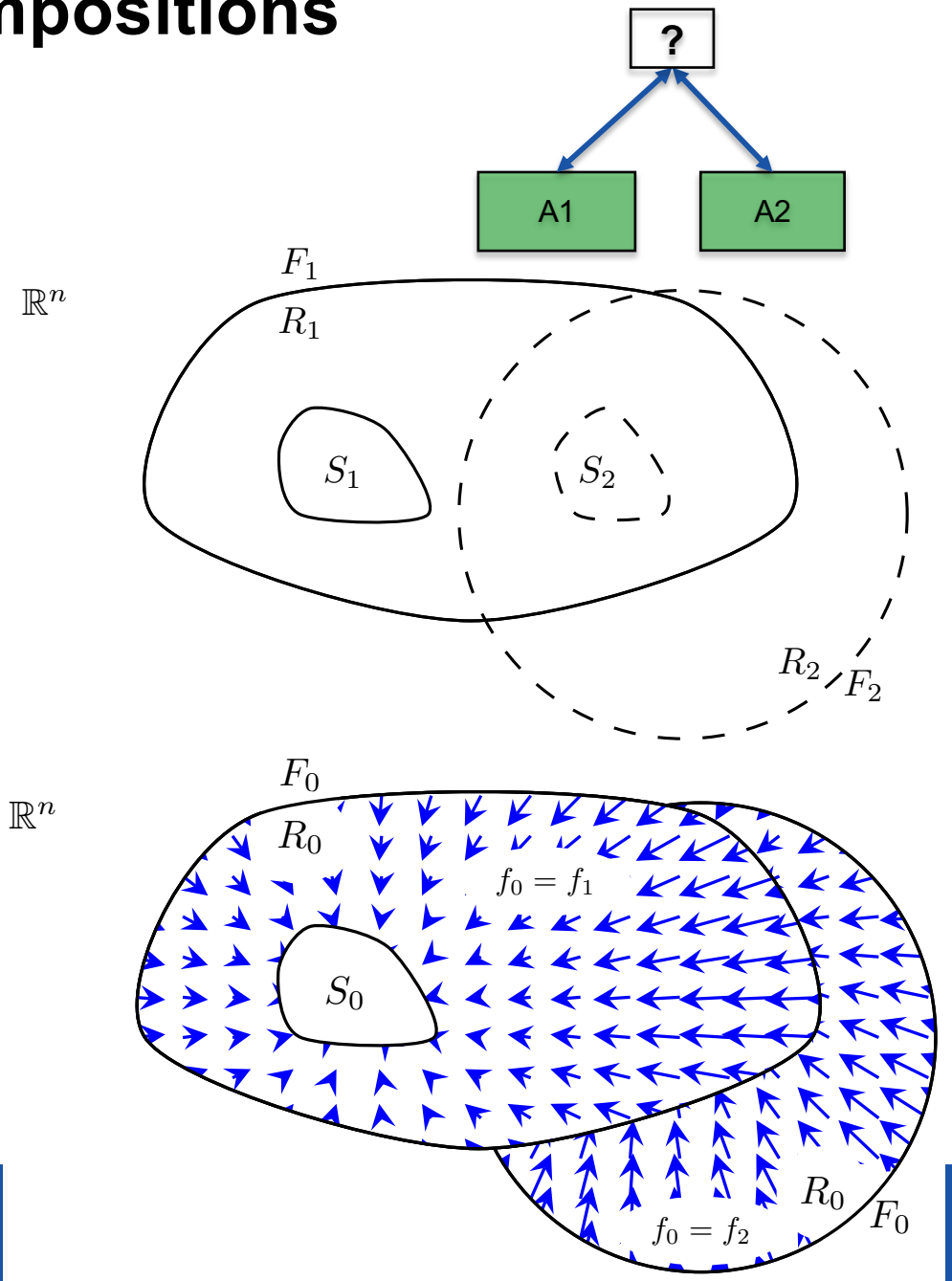
If both BTs are FTS

And (S2 subset R1)

Then the composition is FTS

Note that this property can be applied to arbitrarily large BTs

This is the BT version of the Nilsson result



Safety and BTs

Robot manufacturers need to guarantee Safety

- Manipulators
- Lawn Mower Robots
- (any other un-caged robot bigger than a toy)

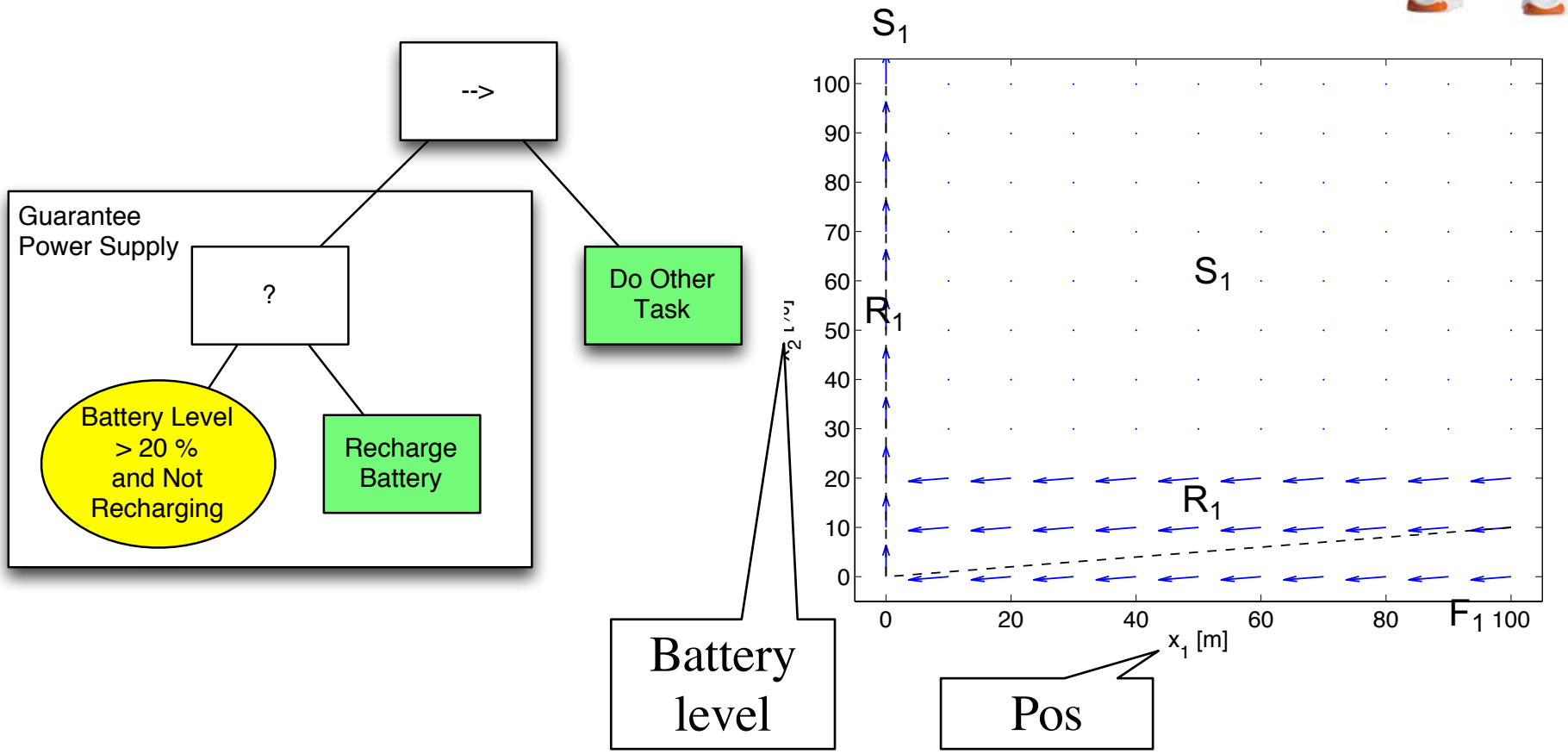
But

- New functionality → Increased Complexity → Increased Costs

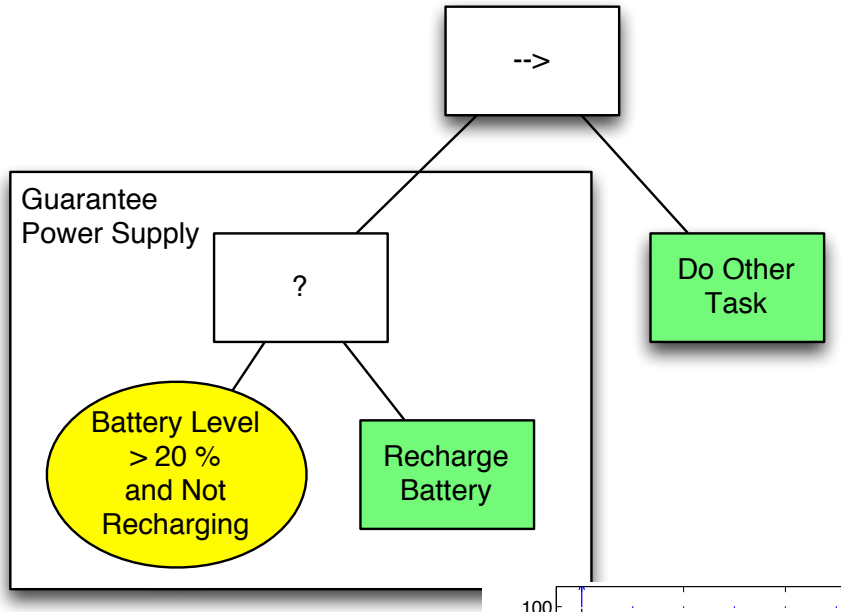
Question:

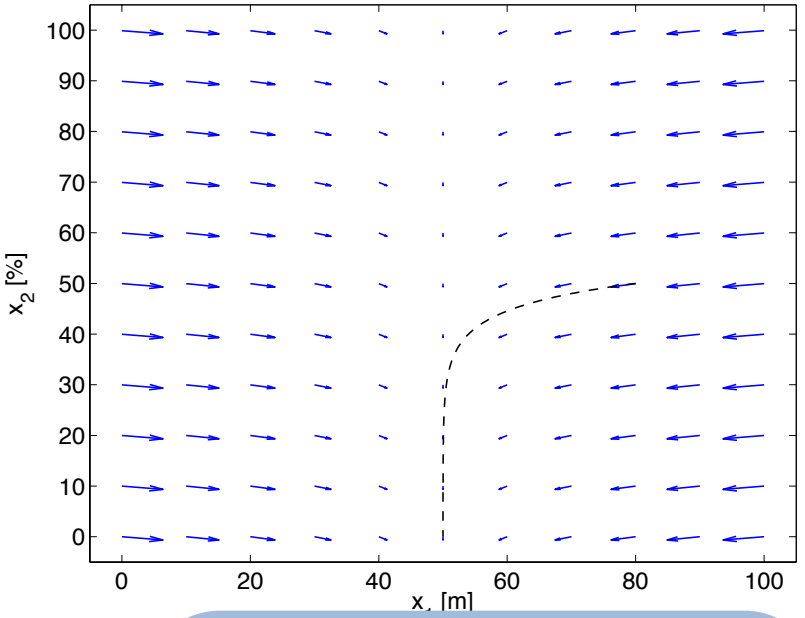
- Can we guarantee safety across BT compositions?

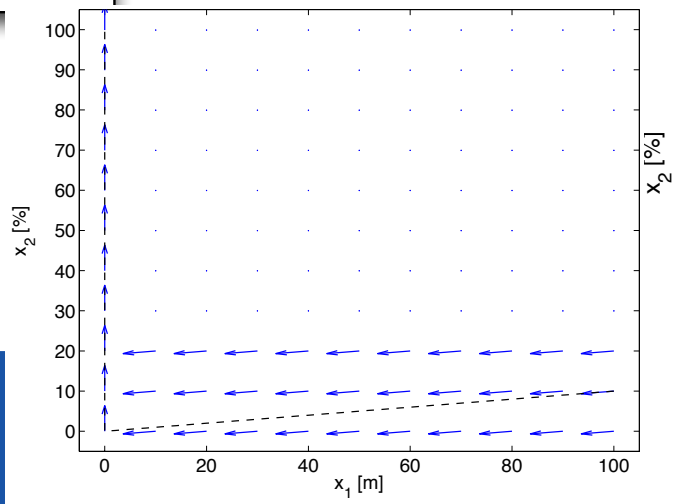
Example: Avoiding Empty Batteries

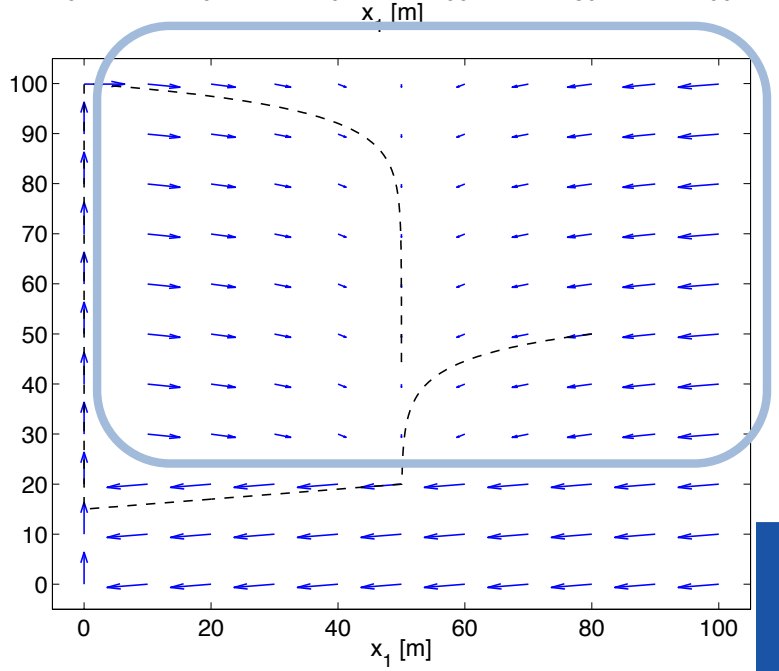


Avoiding Empty Batteries

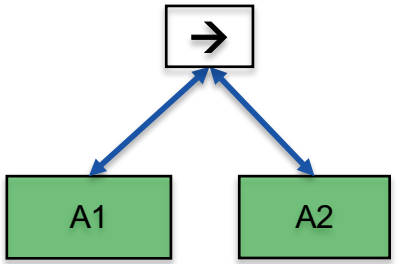








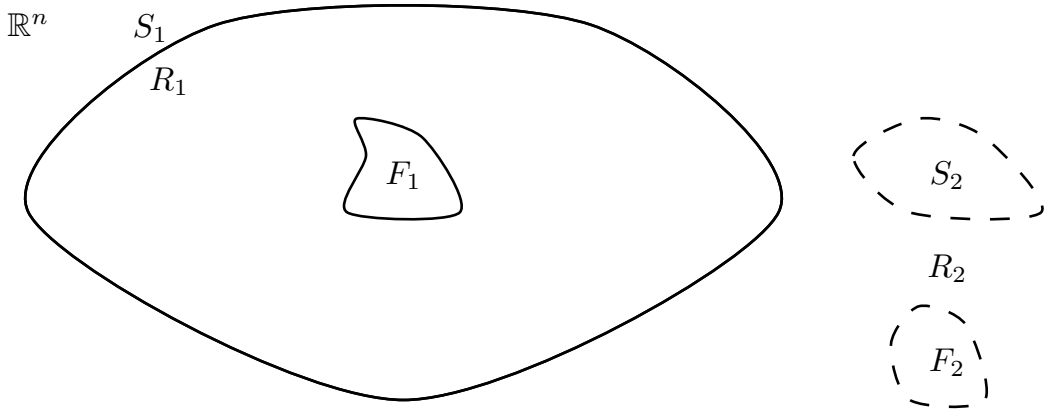
Safety in BTs

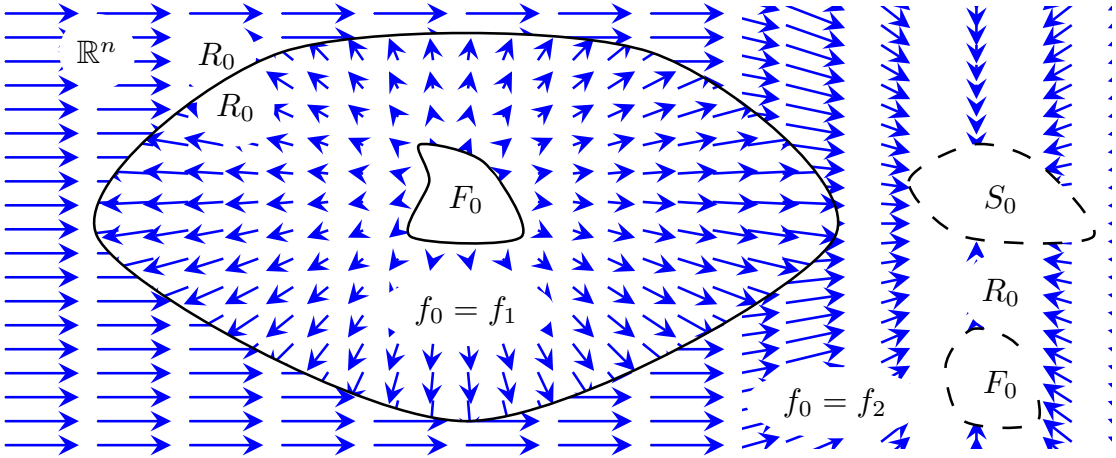


What can we say about the composition?

If A_1 is Safe

Then the composition is Safe wrt F_1

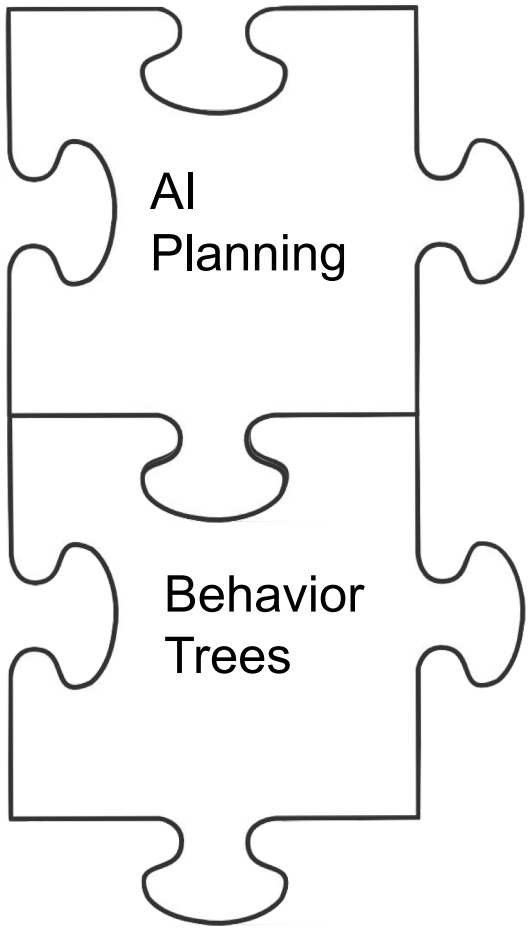




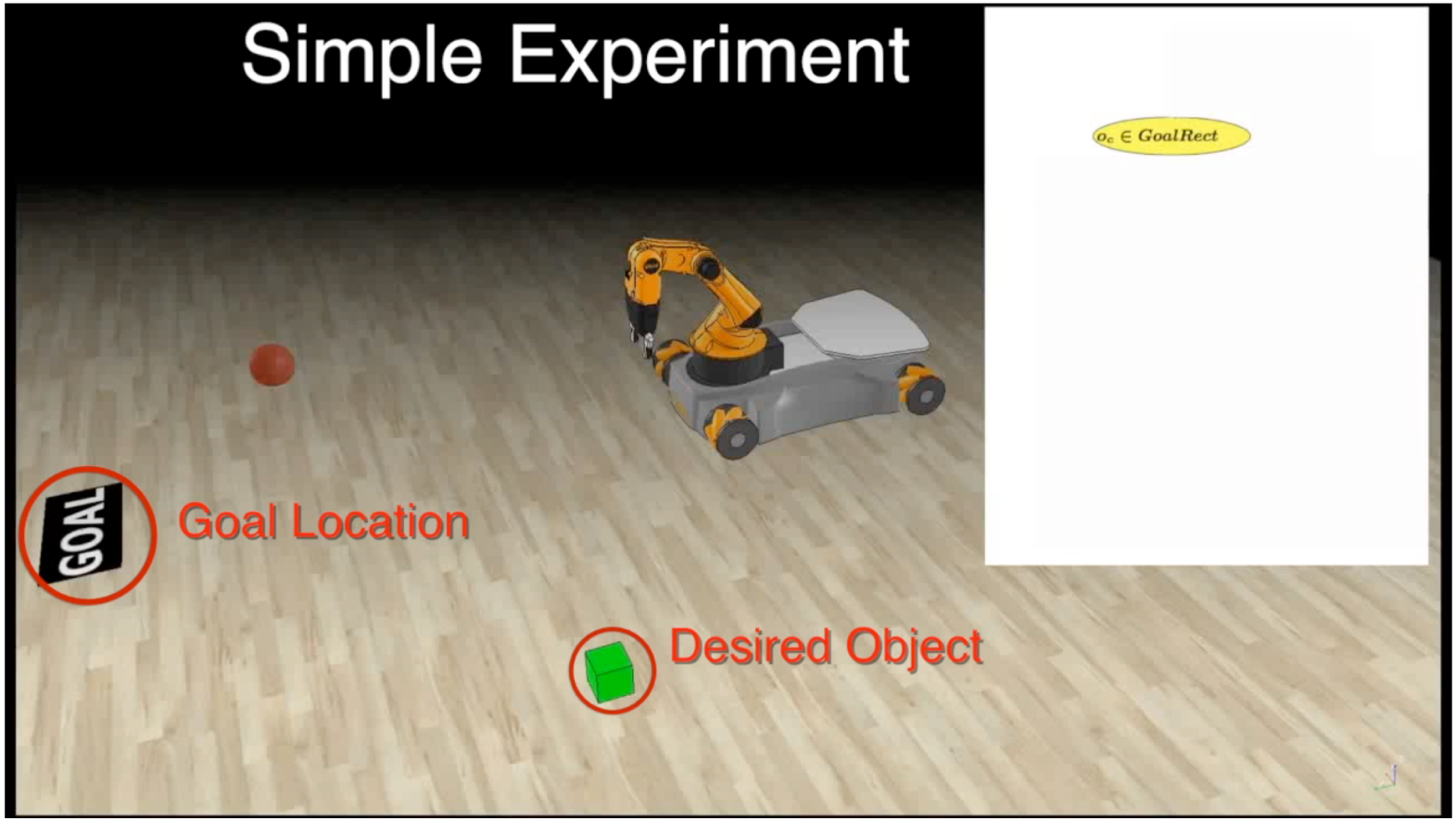
Note that we can have chattering…

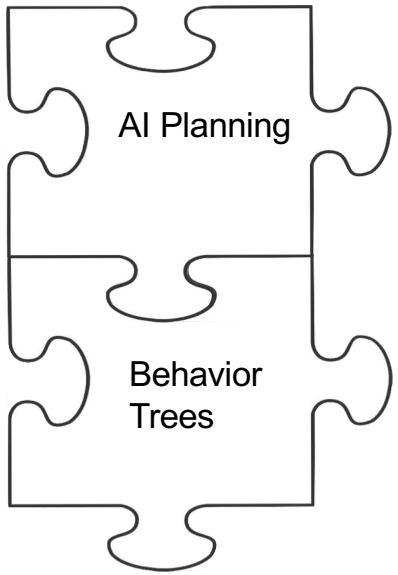
The Big Picture

- BT integrate very well with AI Planning
- Ex: Back-chaining
  - Need to handle Conflicts
    - one action disabling another
  - Need to handle Sampling
    - an object can be placed in an infinite number of positions...

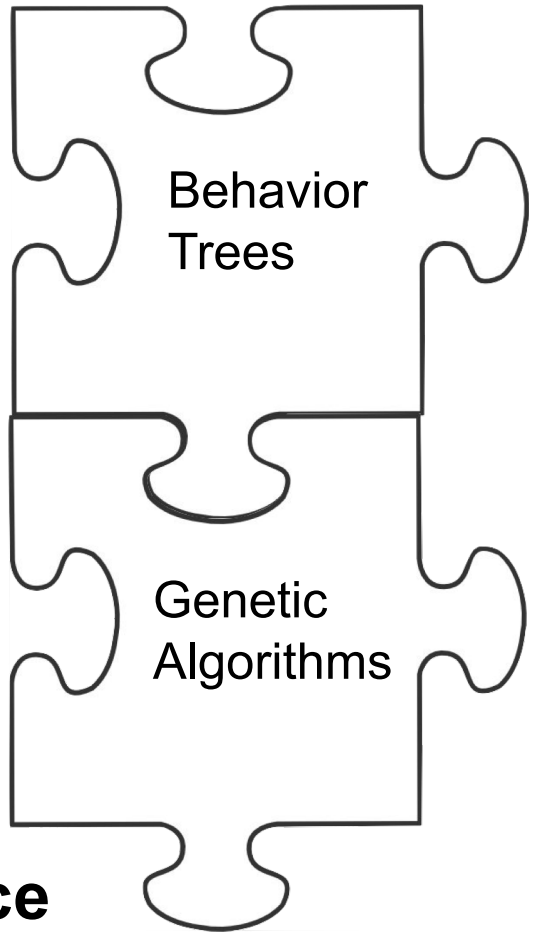


The Big Picture





The Big Picture



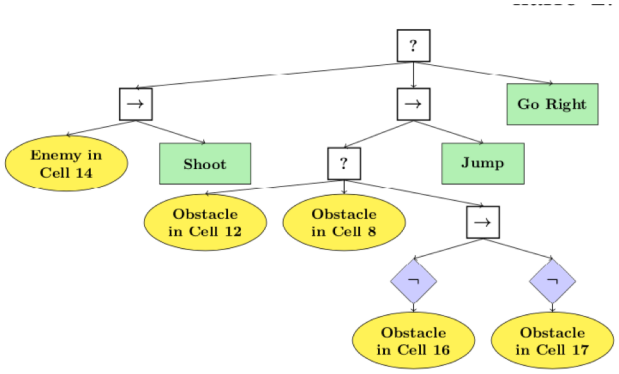
Actions

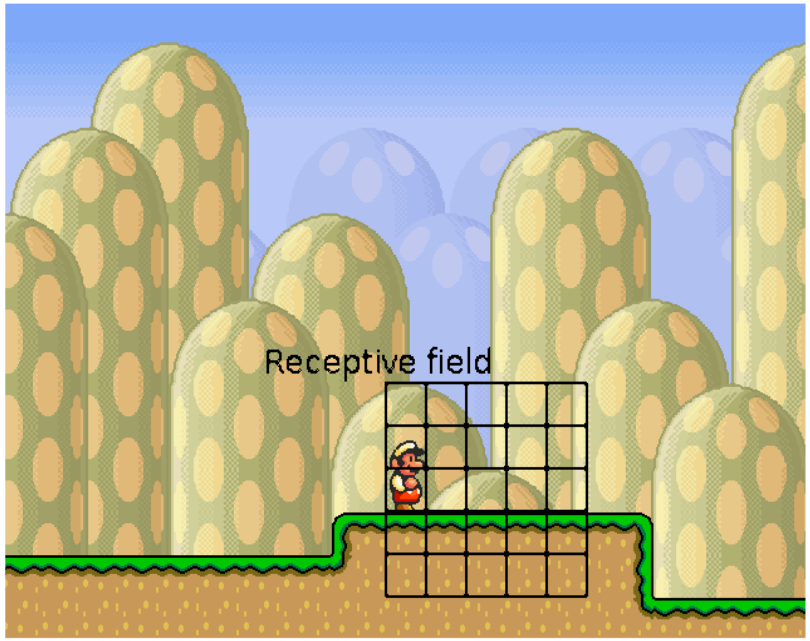
- Run
- Jump
- Shoot

Conditions

- 5x5 binary sensors

Example:

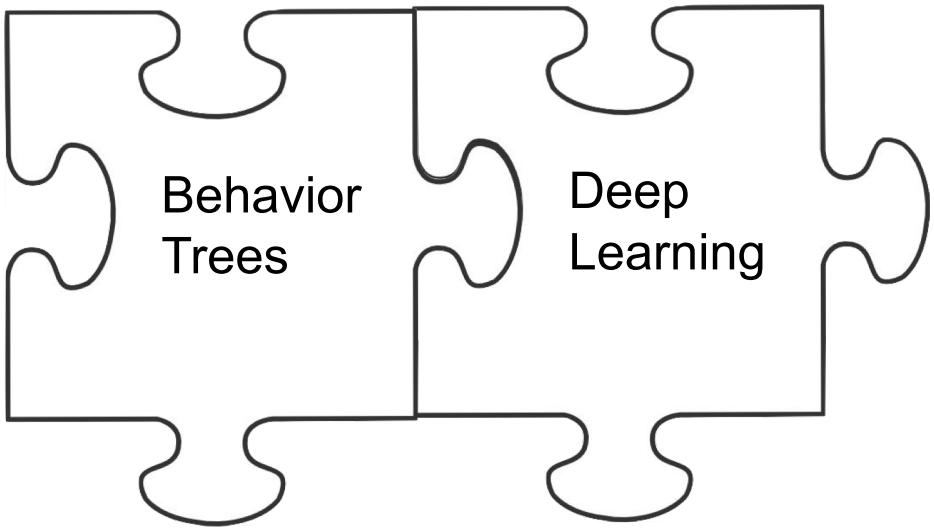




Learning from **Experience**

- Apply Genetic Programming
- BT trivially maps to Genes
- Mutation/Crossover easy

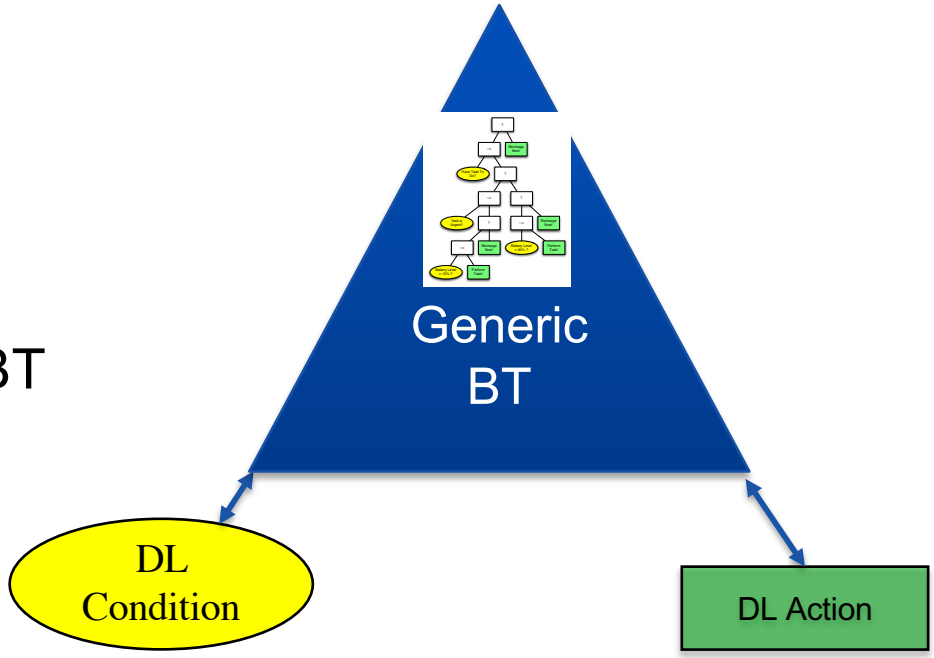
The Big Picture



- BTs enable a seamless transition to DL solutions
- Use DL for Conditions (image and speech processing)
- Use DL for Actions (Deep Q Learning)

- Use DL for larger sub-BTs

- If possible, replace entire BT with DL...



Outline

- What do we mean by task switching?
- Quotes on BTs
- Why BTs?
- How do BTs work?
- 5 Efficient Design Principles for BTs
- Alternatives to BTs
- The Big Picture
  - BTs and Planning
  - BTs and Control Theory
  - BTs and Genetic Algorithms
  - BTs and Deep Learning