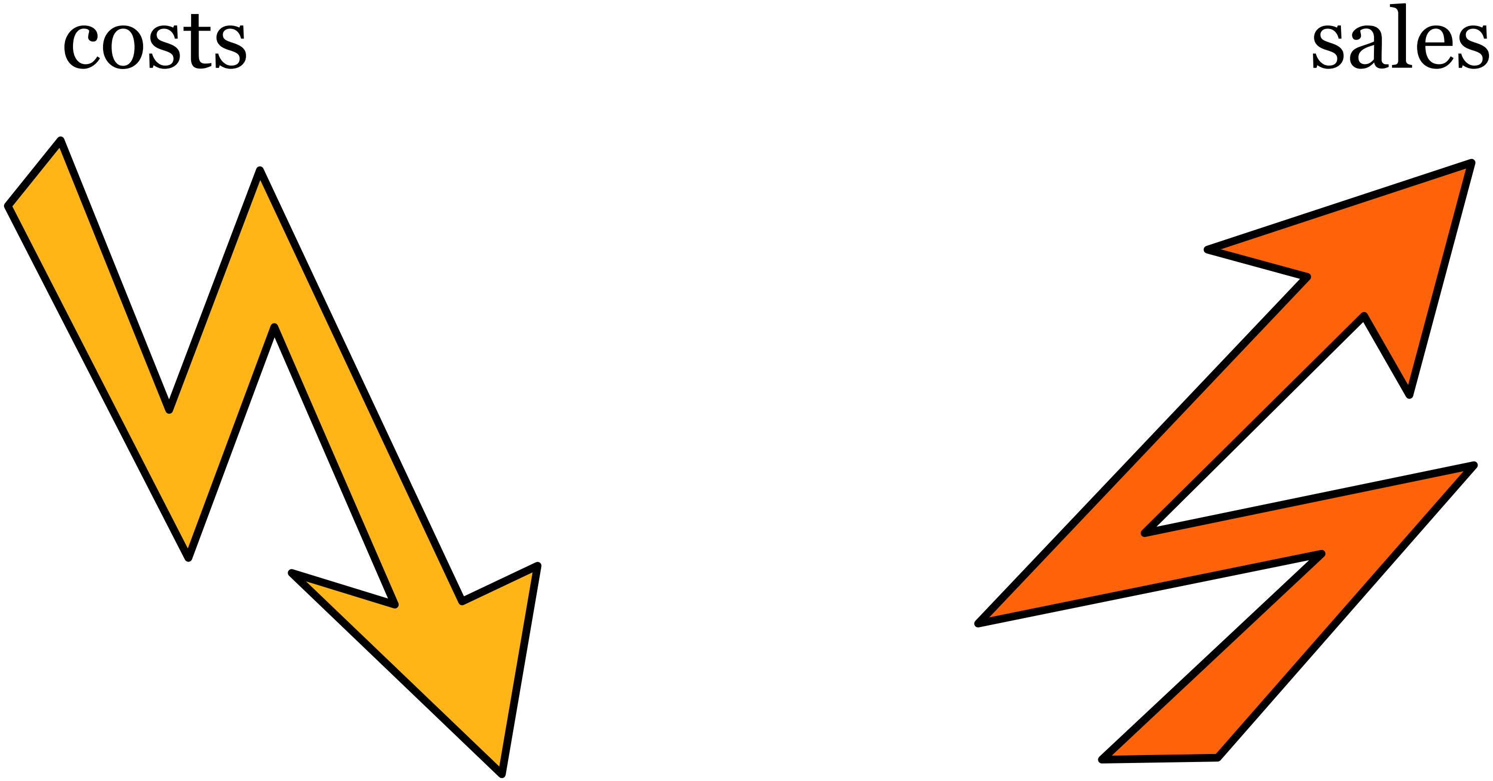
Behavior Trees

for

Next-Gen AI

alexjc@AiGameDev.com

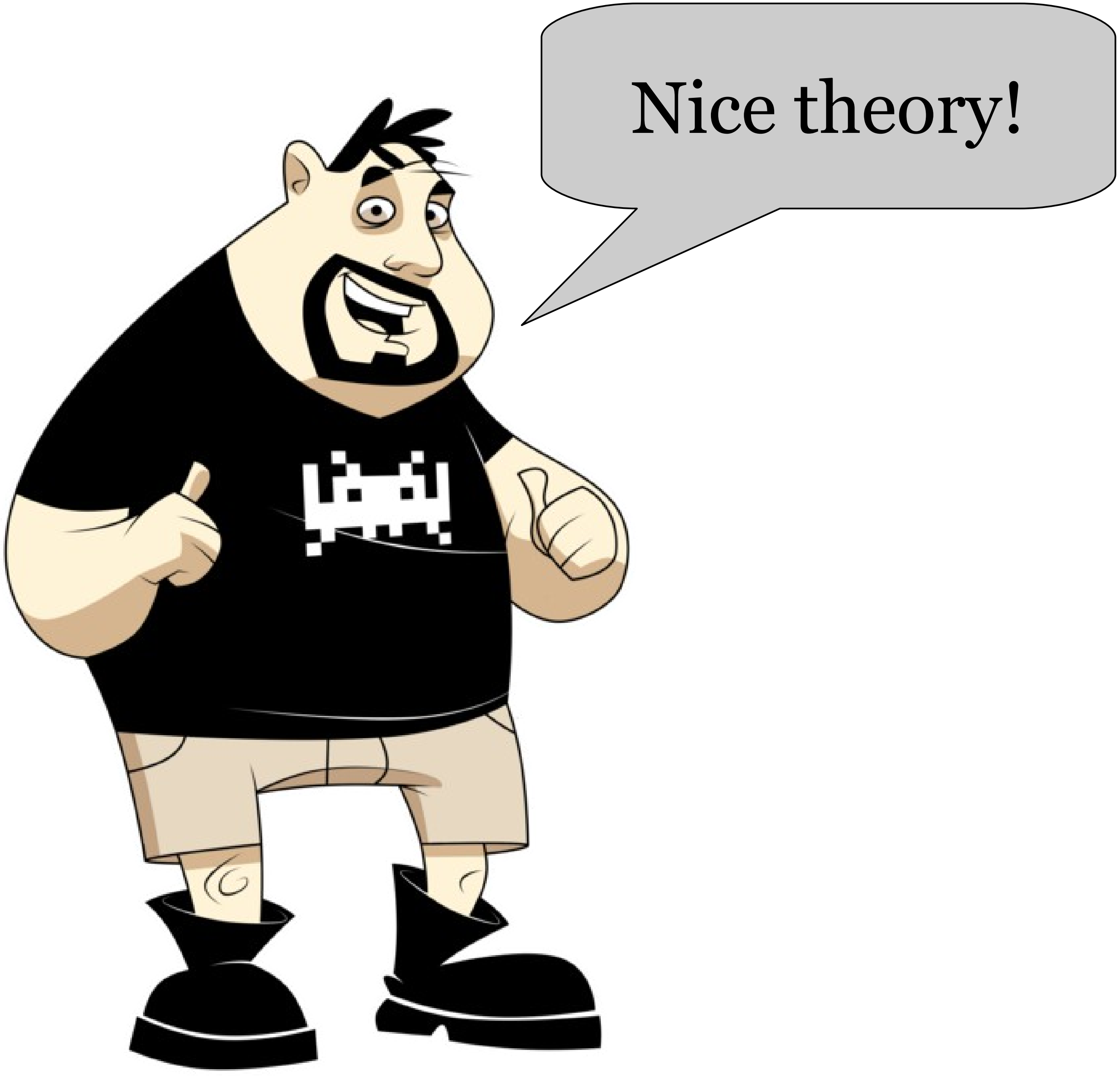


costs

sales

Next-Gen AI

Scripting could actually be fun?
Innovative gameplay!


Nice theory!

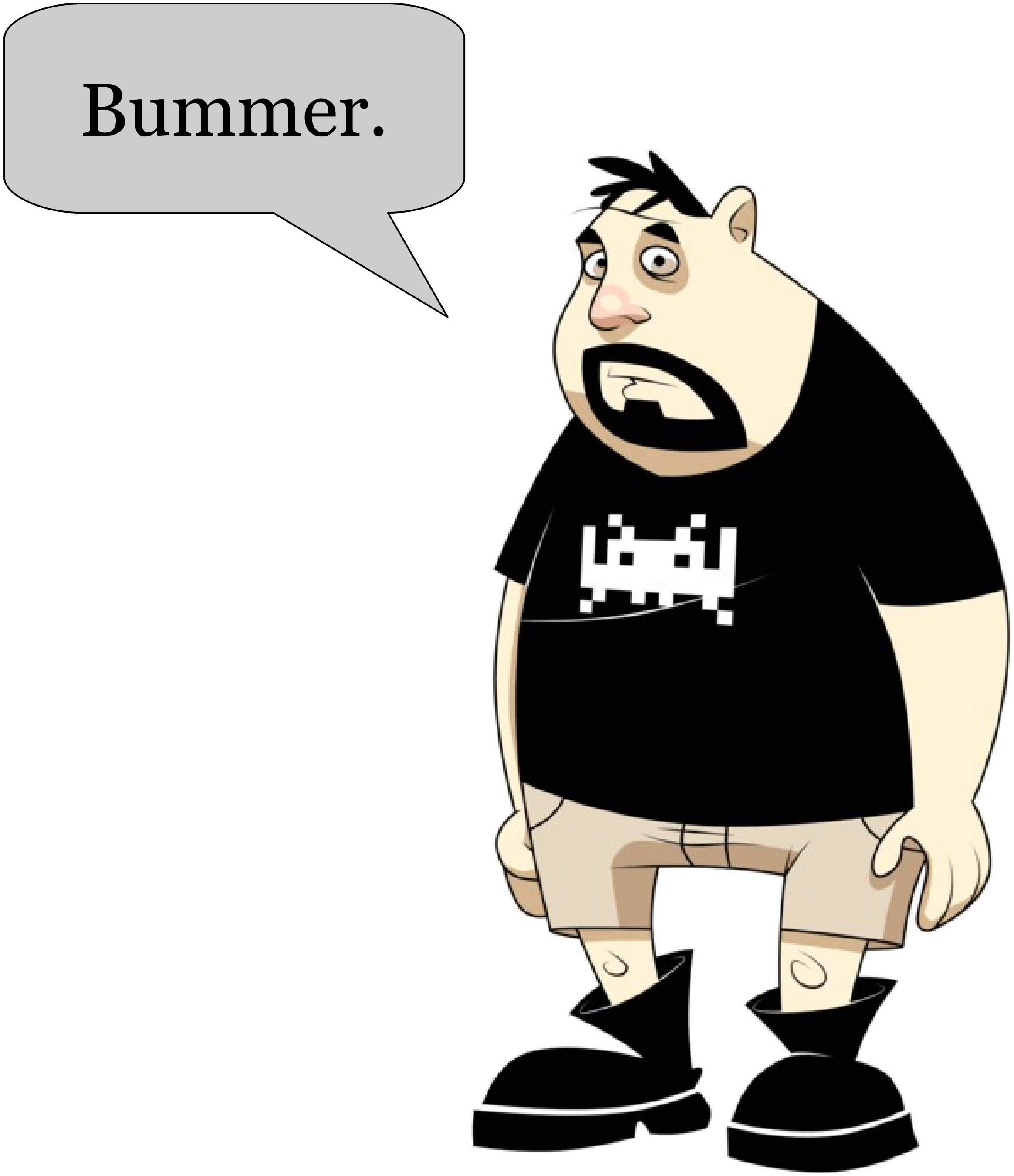
Challenges

☑ Bigger environments

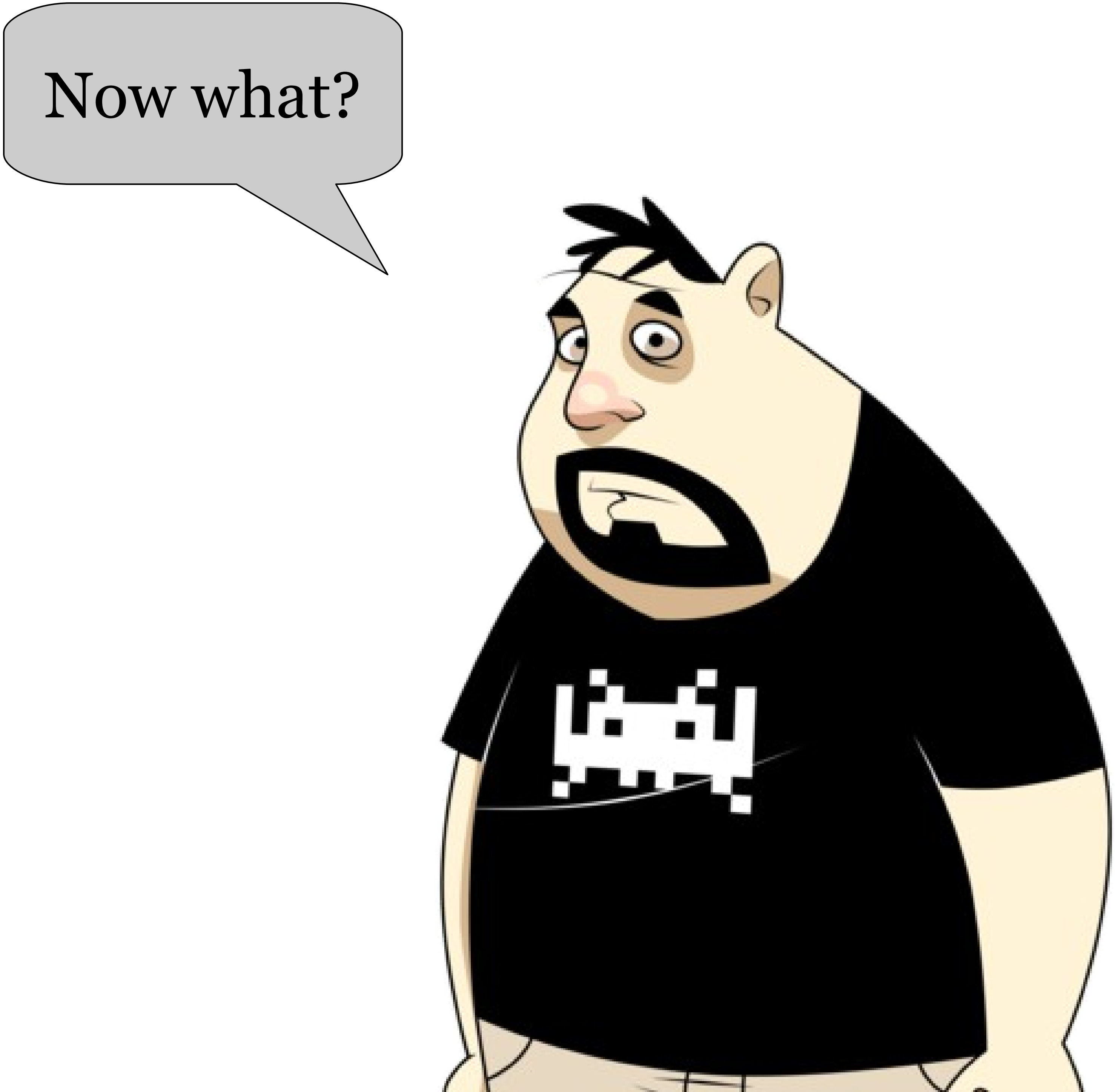
☑ Greater numbers of entities

☑ More accurate physics

☑ Advanced animation


Bummer.

Next-Gen AI


Now what?

Requirements

Wish List

☑ Perfect designer supervision

☑ AI behaves autonomously too

Wish List

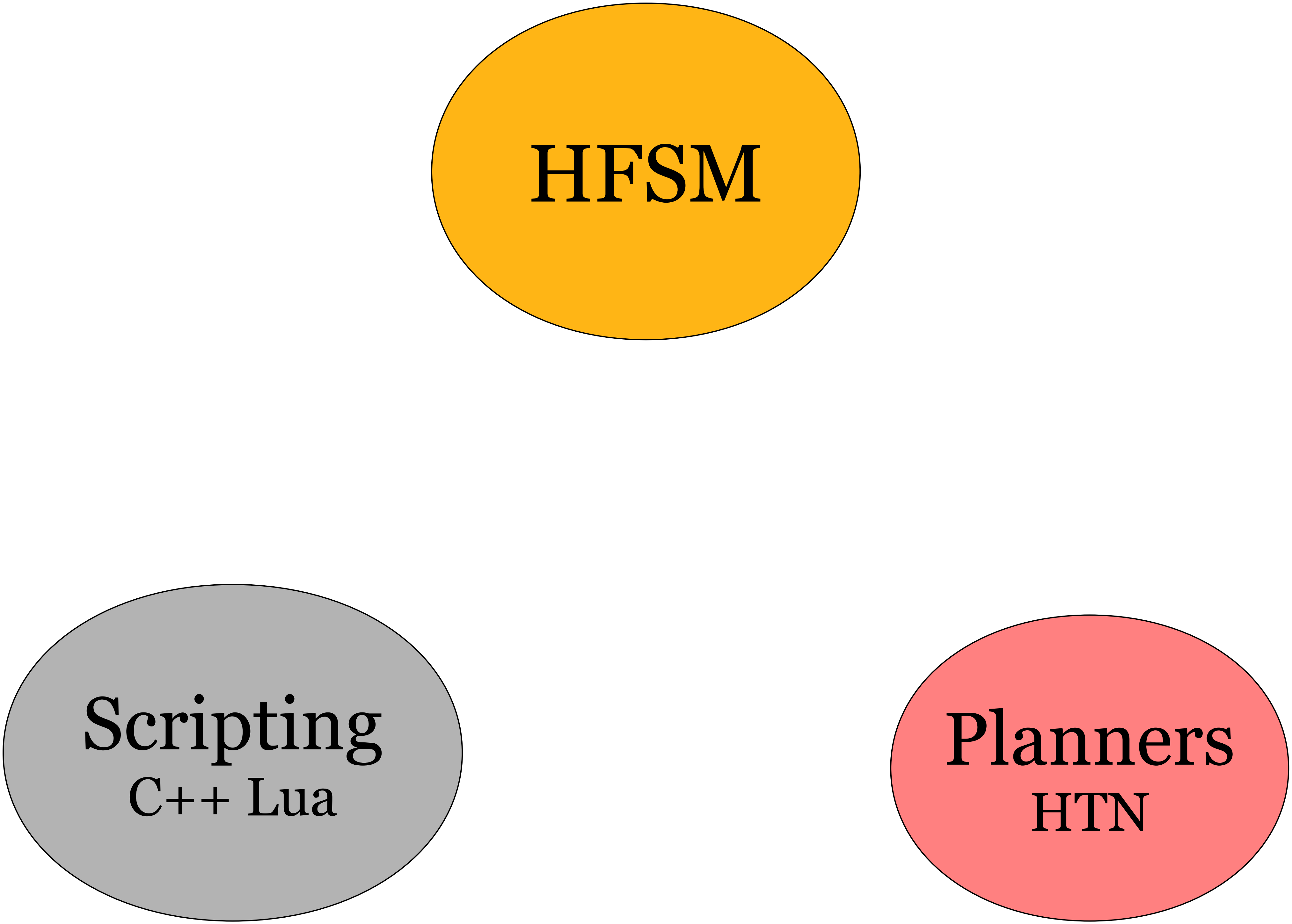
The AI should
be goal directed

...yet react to
sudden changes!

☑ Purposeful behaviors

☑ Responds to events

Hierarchical Logic



HFSM

Scripting
C++ Lua

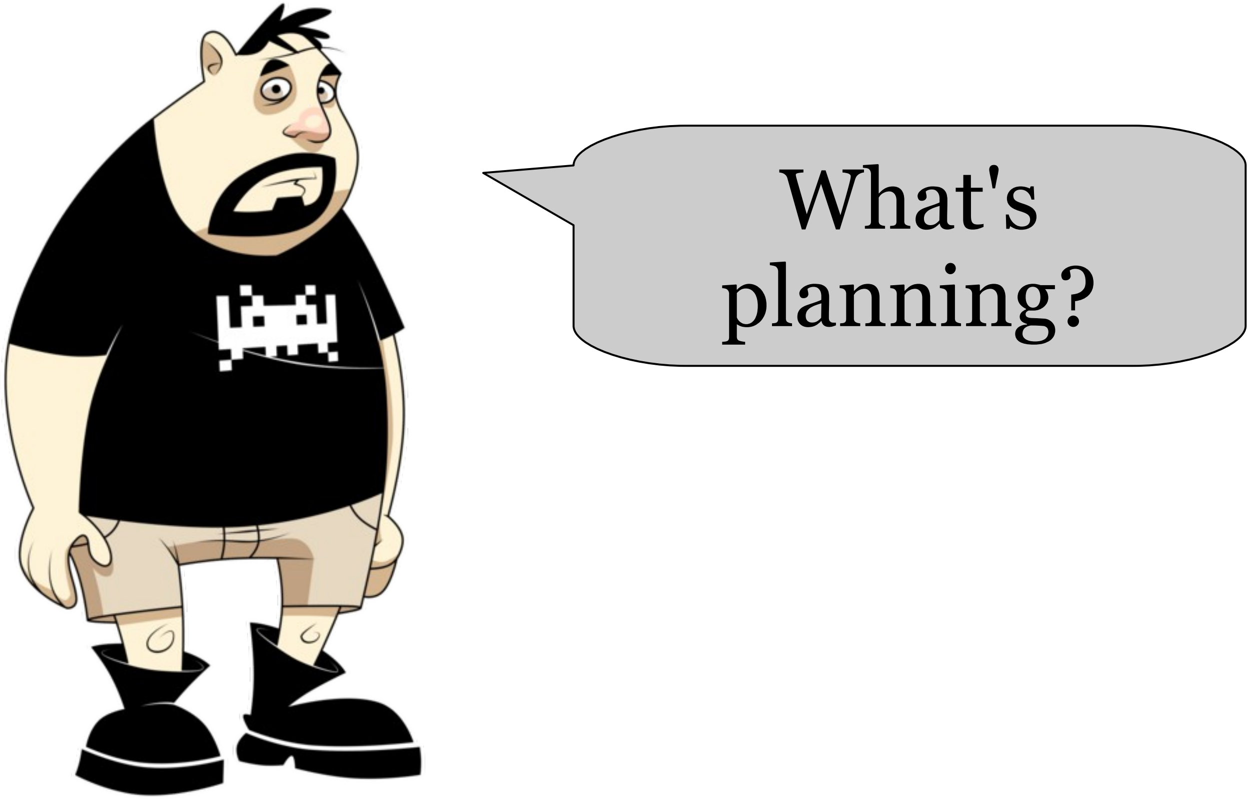
Planners
HTN

Scripting: The Good

☑ Any computation possible

☑ Widespread experience

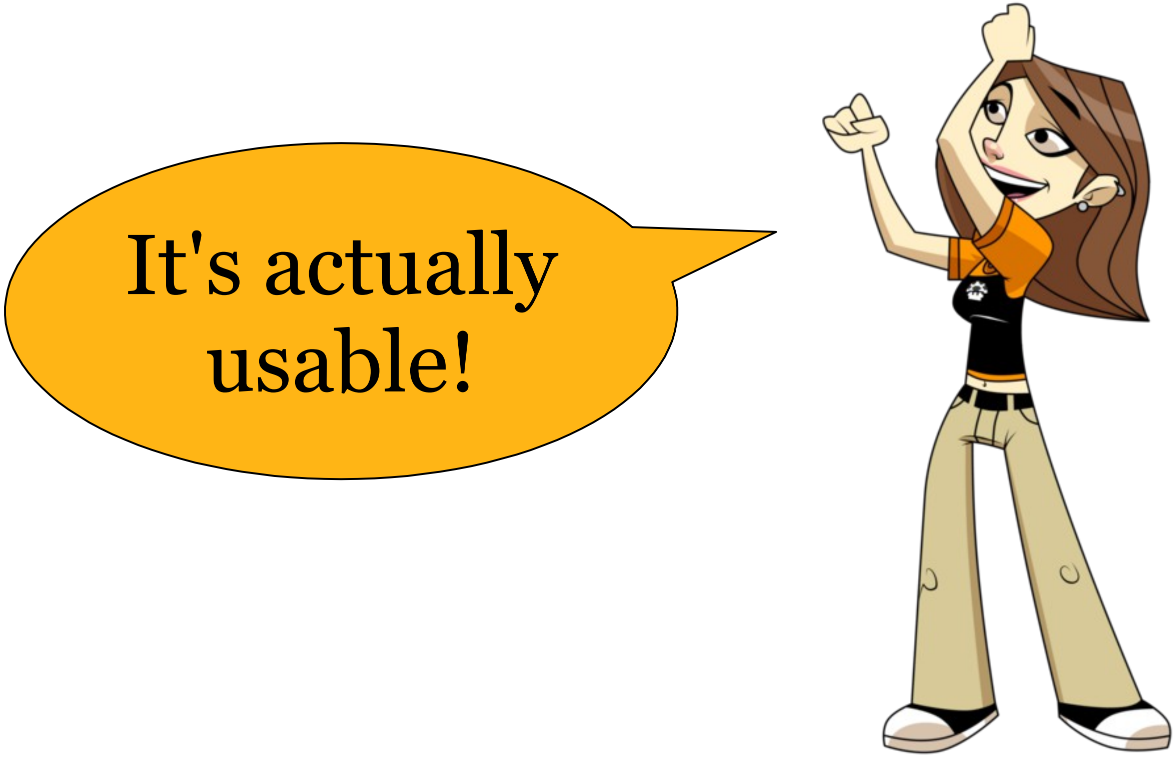
Scripting: The Bad



☑ Difficult to introspect, analyze

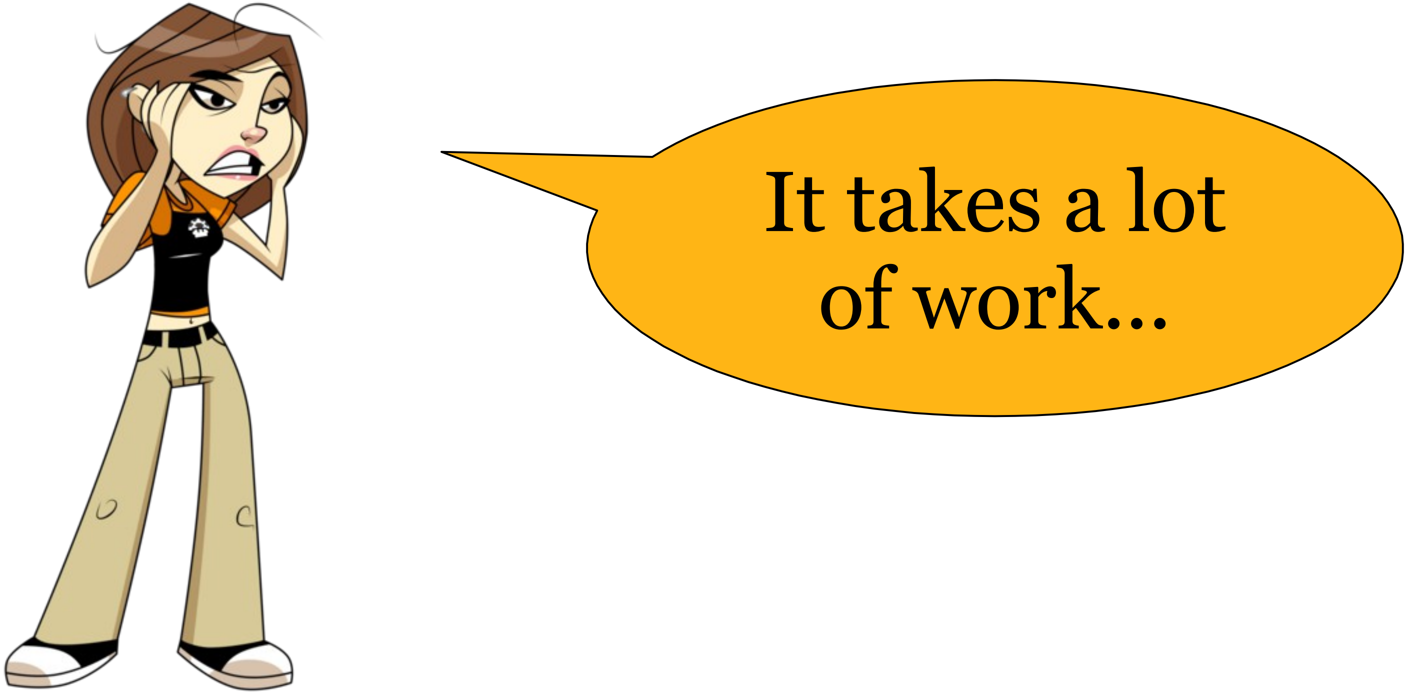
☑ Not accessible to many designers

HFSM: The Good



- ☑ Simple and intuitive
- ☑ Full low-level reactive control

HFSM: The Bad



☑ Generally labor intensive

☑ Not easy to build goal directed

Planning: The Good

AI can solve many logic problems.

☑ Uses search for automation

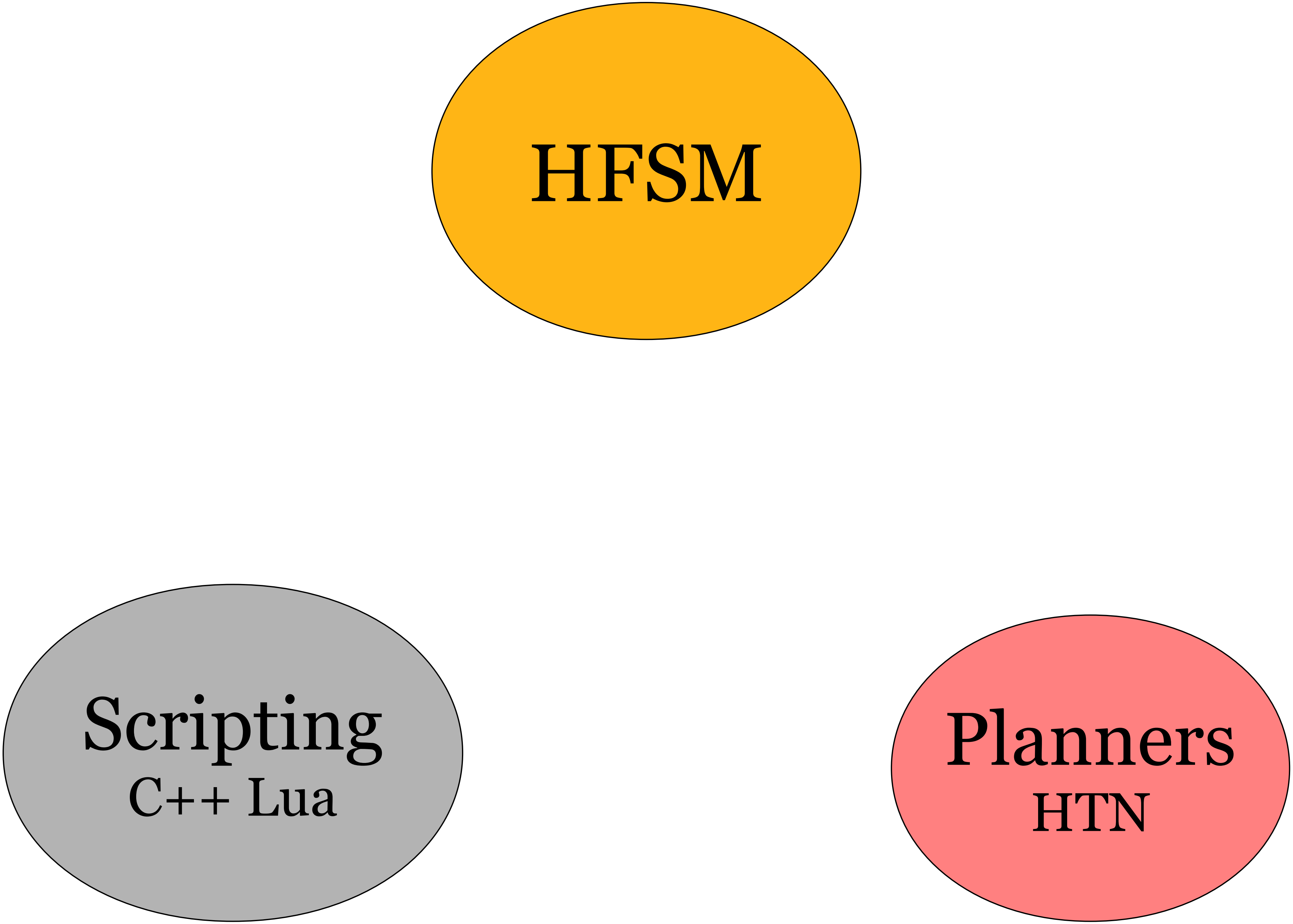
☑ Goal directed by default!

Planning: The Bad

It's disconnected from the real world?

☑ Integration of procedural code

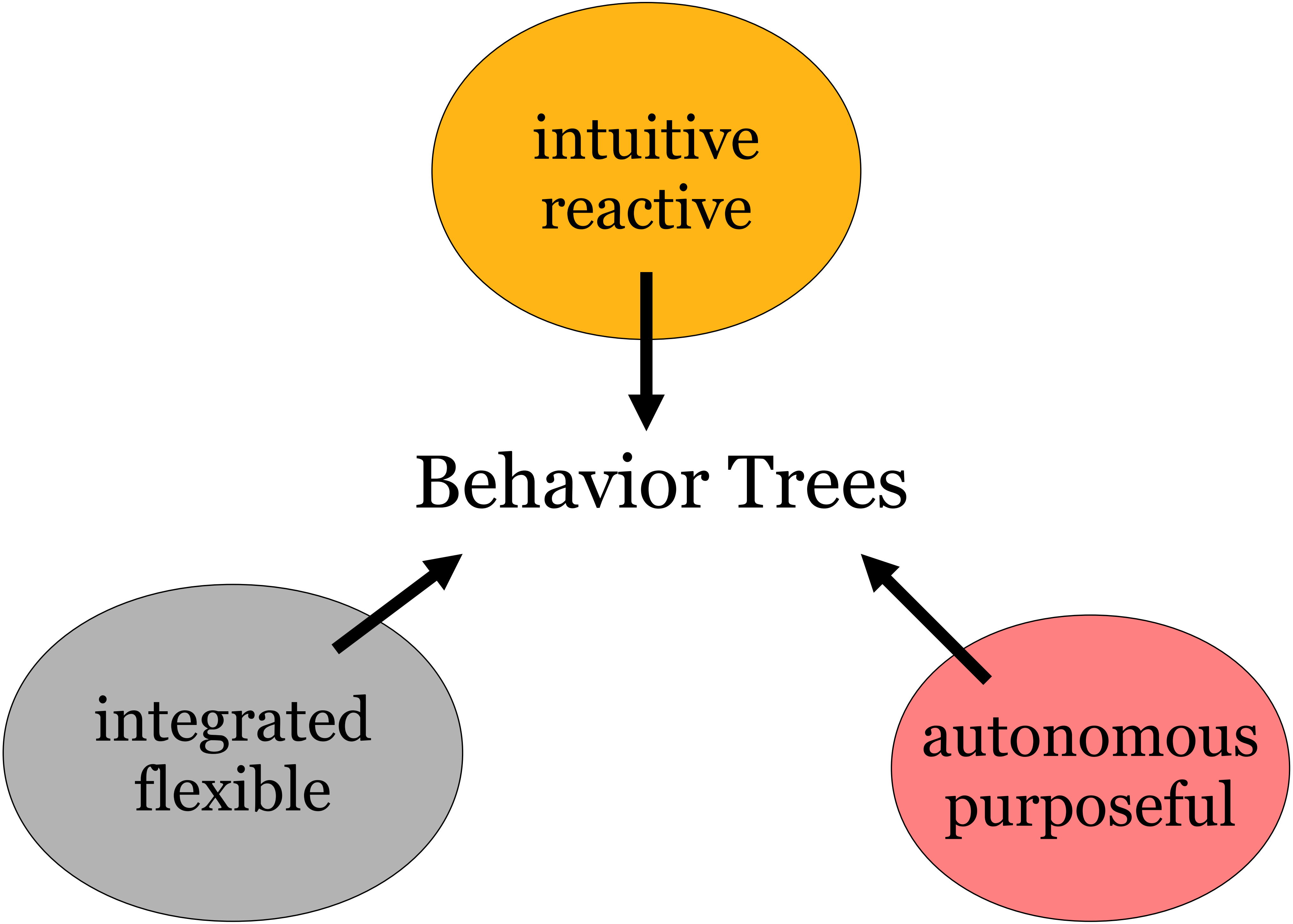
☑ Ignores control & execution



HFSM

Scripting
C++ Lua

Planners
HTN



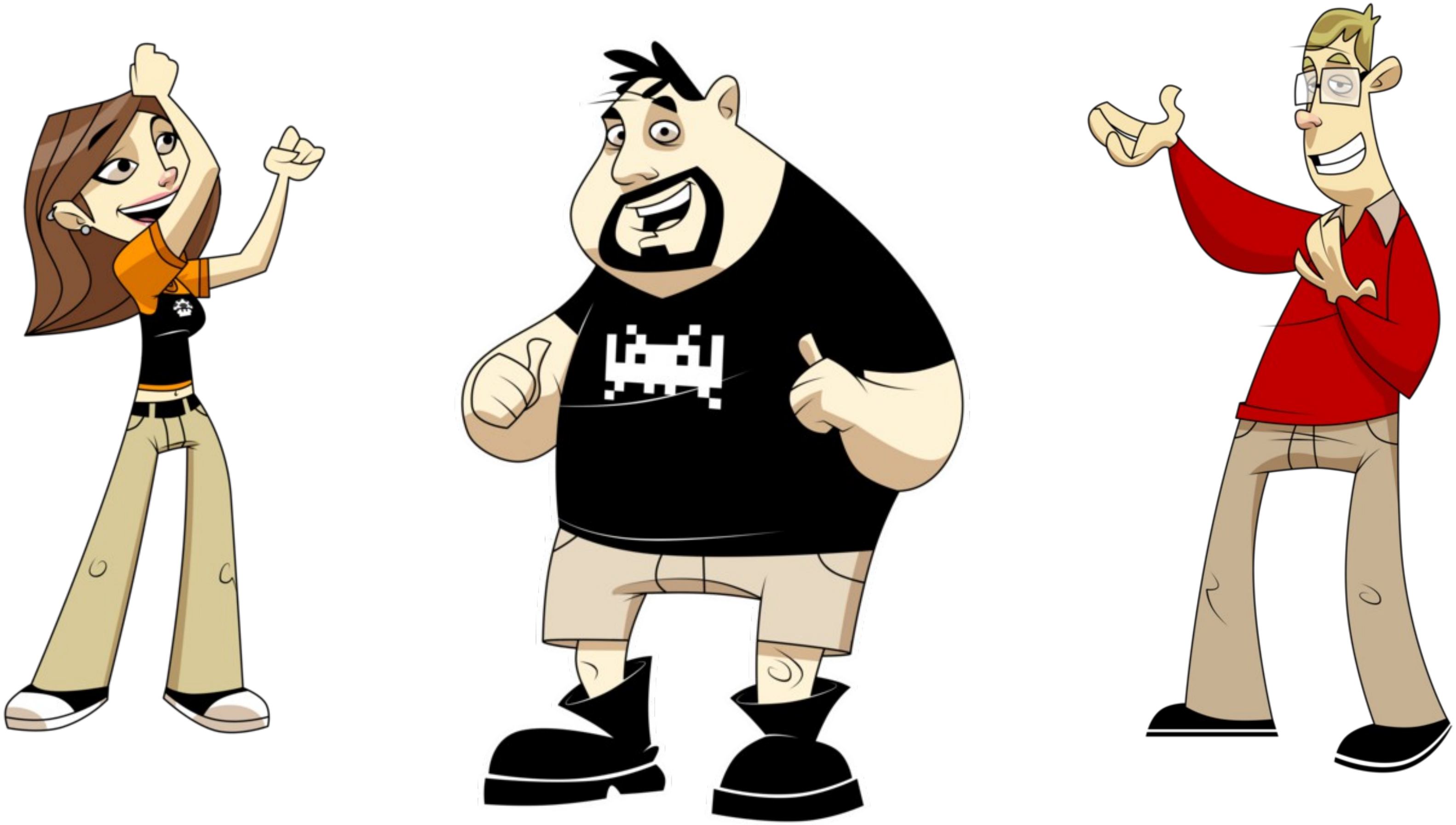
intuitive
reactive

Behavior Trees

integrated
flexible

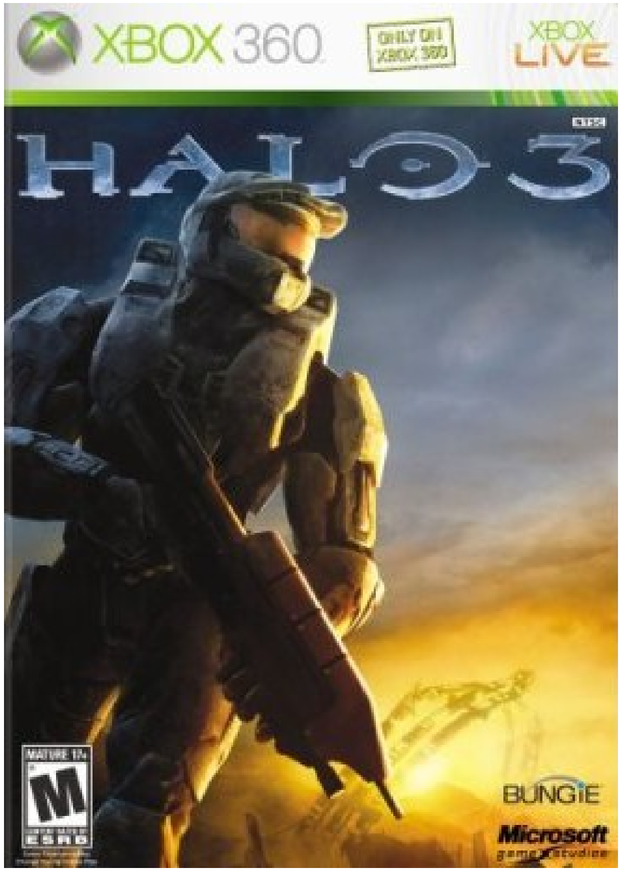
autonomous
purposeful

Bang for Buck!



☑ Decision making over time

☑ Control & monitoring execution



Managing Complexity in the Halo 2 AI System
Damian Isla
GDC 2005

Three Approaches to Behavior Tree AI

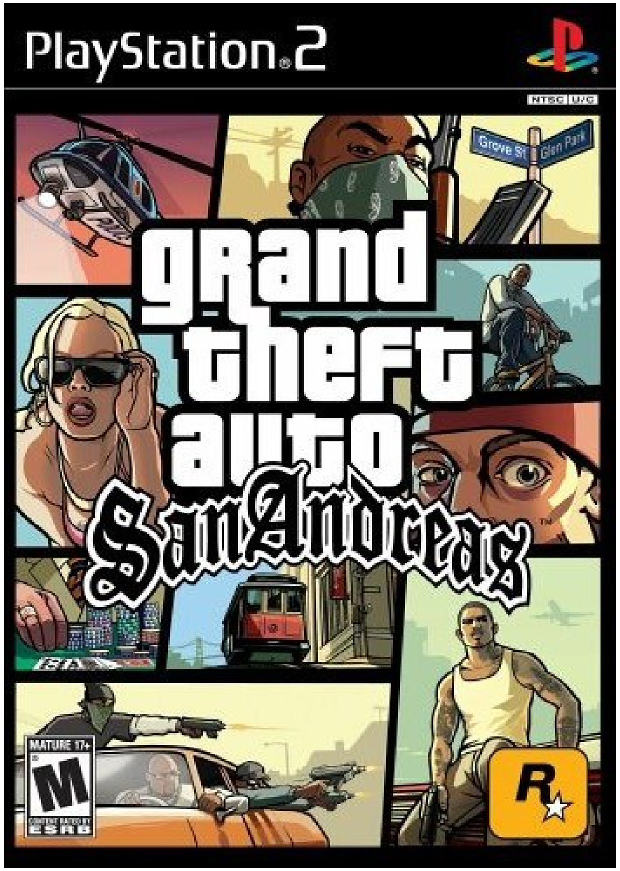
Lauren McHugh

GDC 2007

Managing Intermixing Behavior Hierarchies

Michael Mateas, Andrew Stern

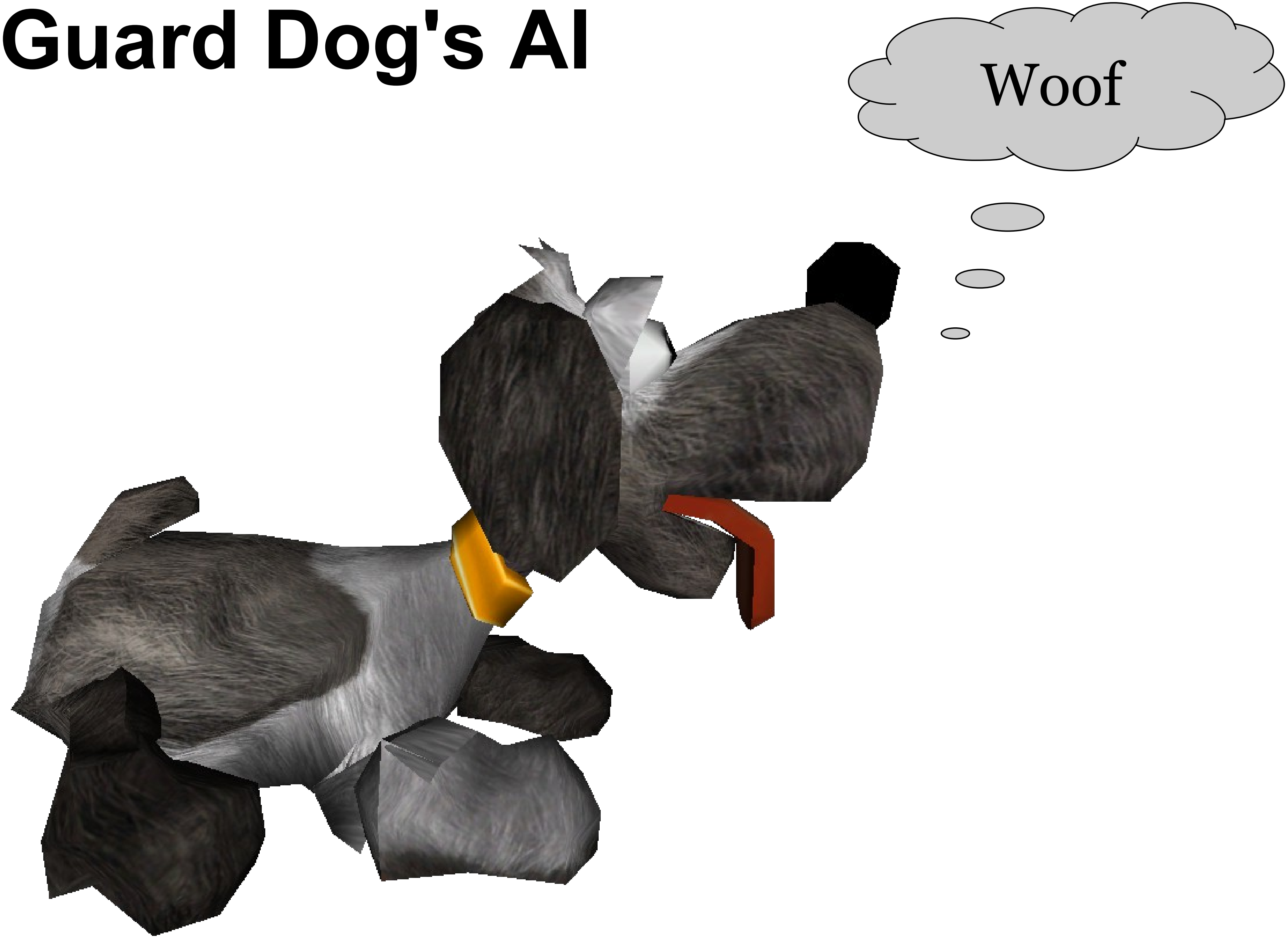
GDC 2004



Top Secret
Classified
Never

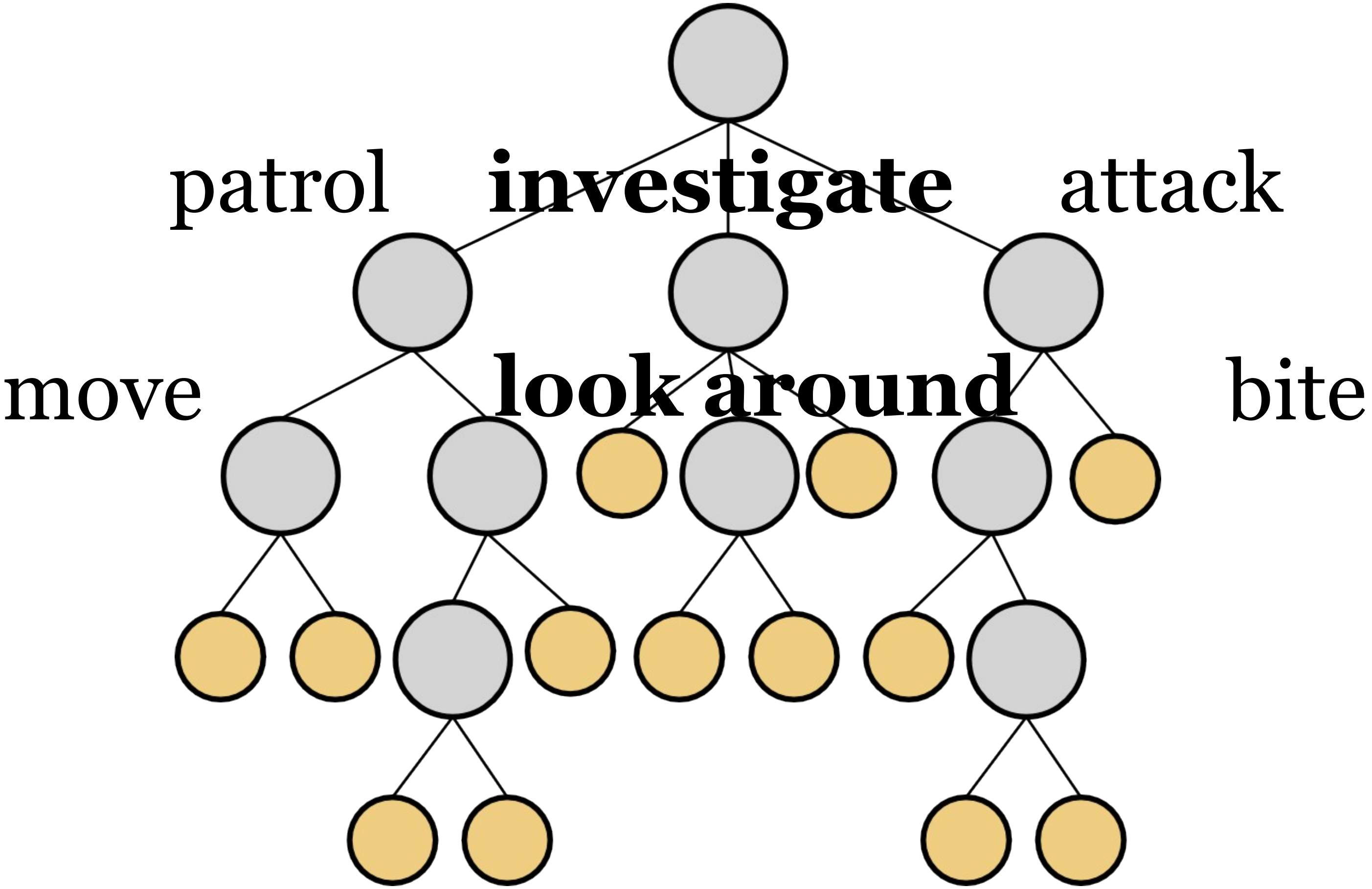
AiGameDev.com

A Guard Dog's AI



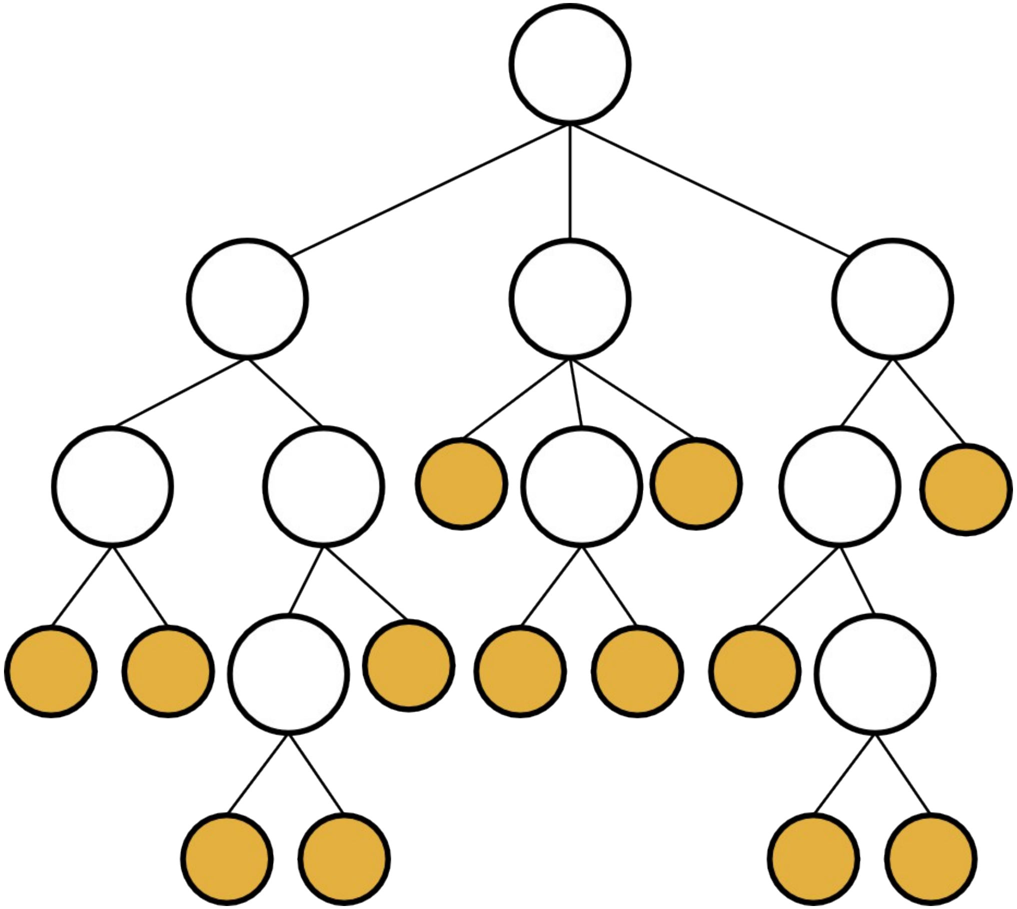
Behavior Trees

Example



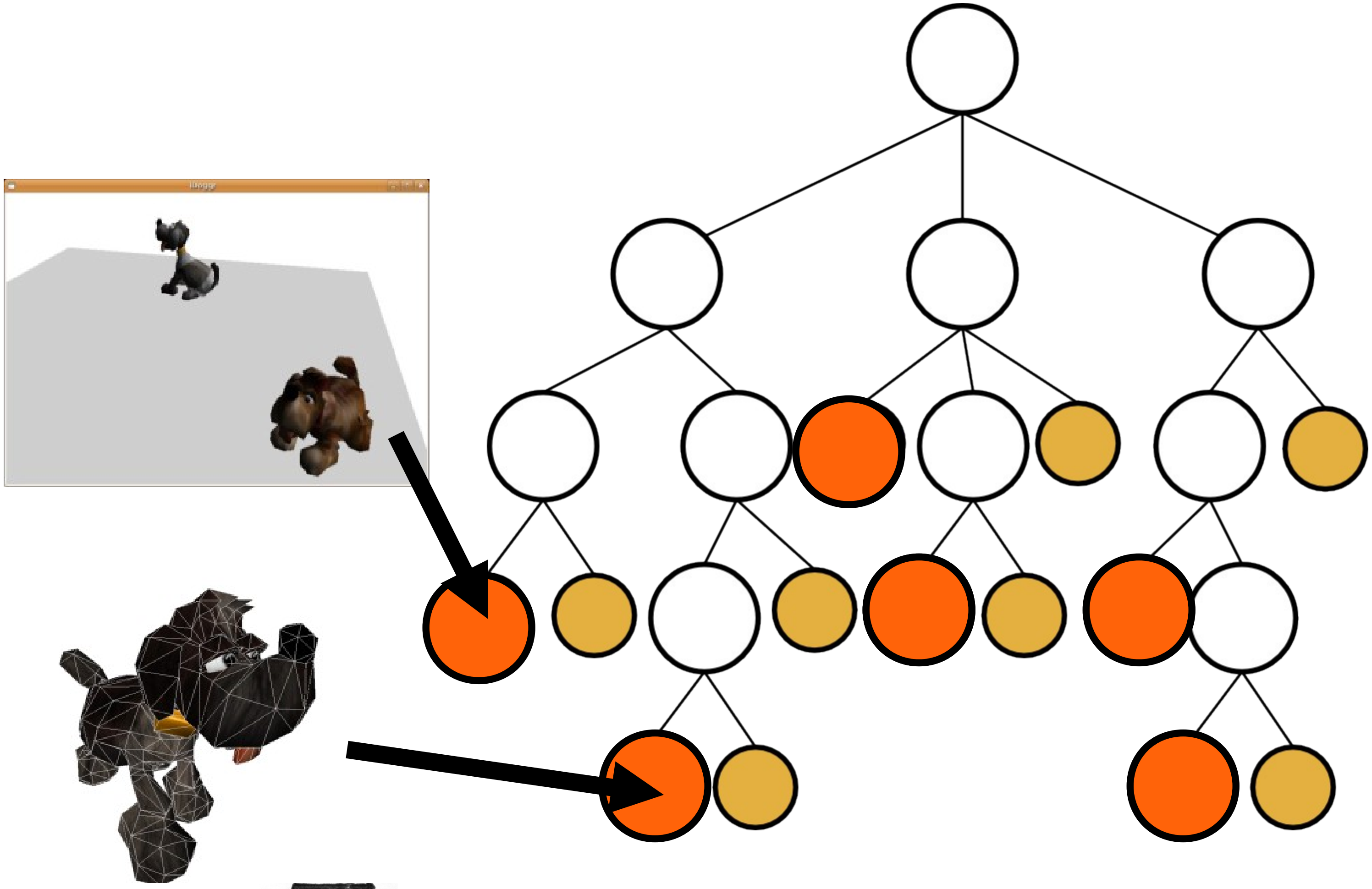
☑ recursive decomposition

Leaves



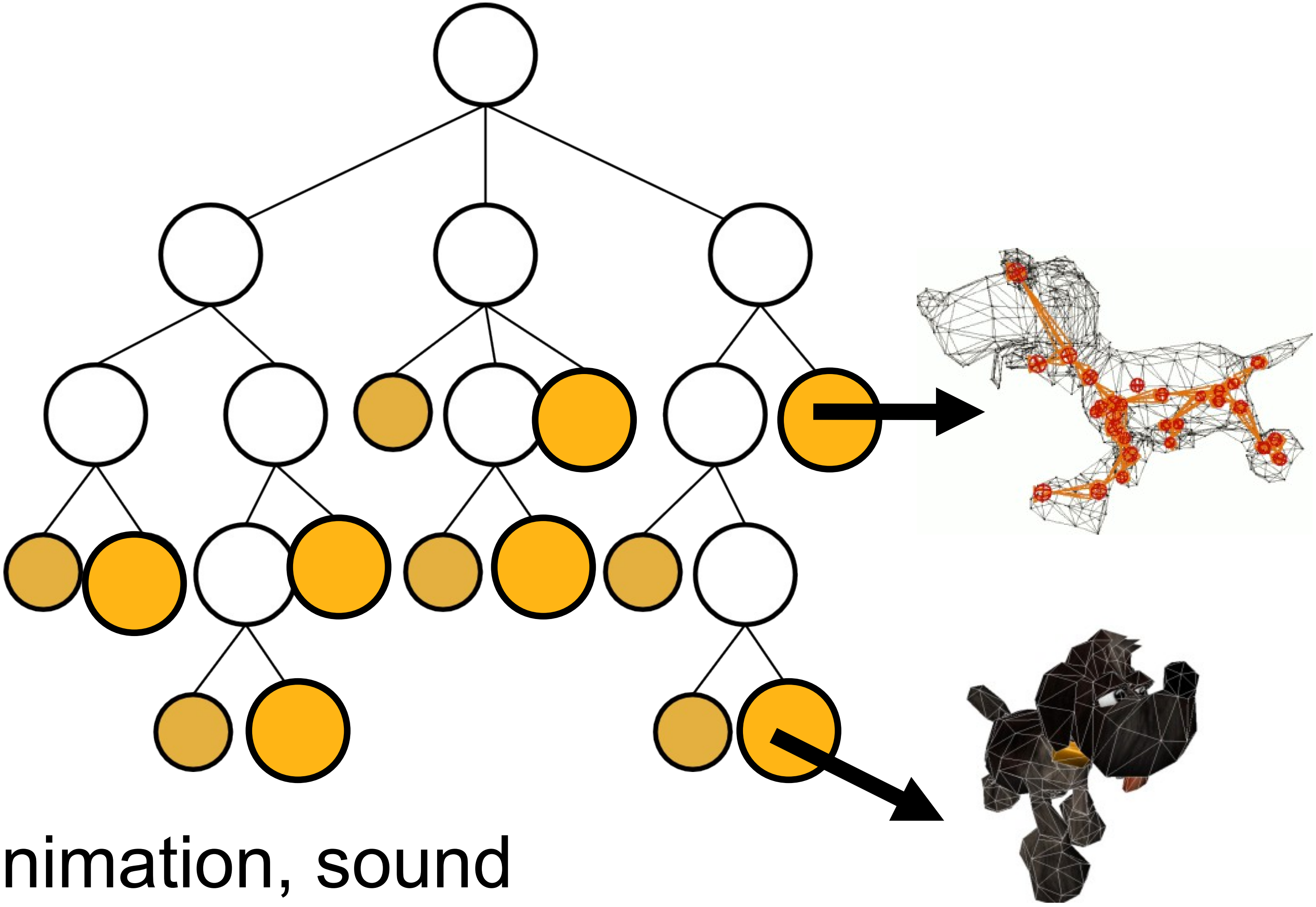
☑ interface between AI and engine

Conditions



- ☑ actor state, meta-checks
- ☑ collision, entity queries

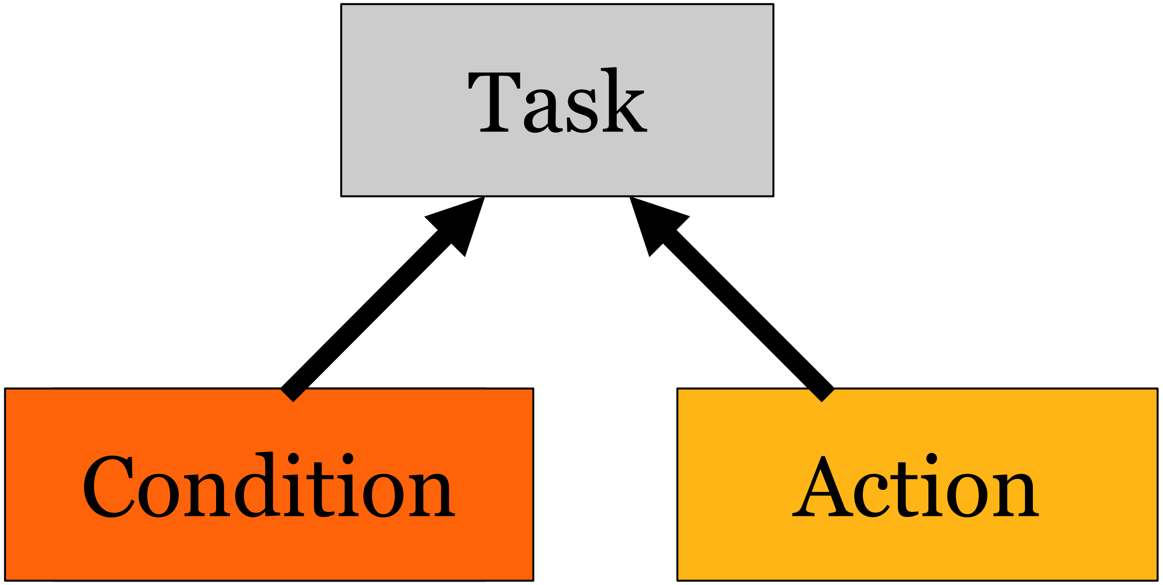
Actions



☑ animation, sound

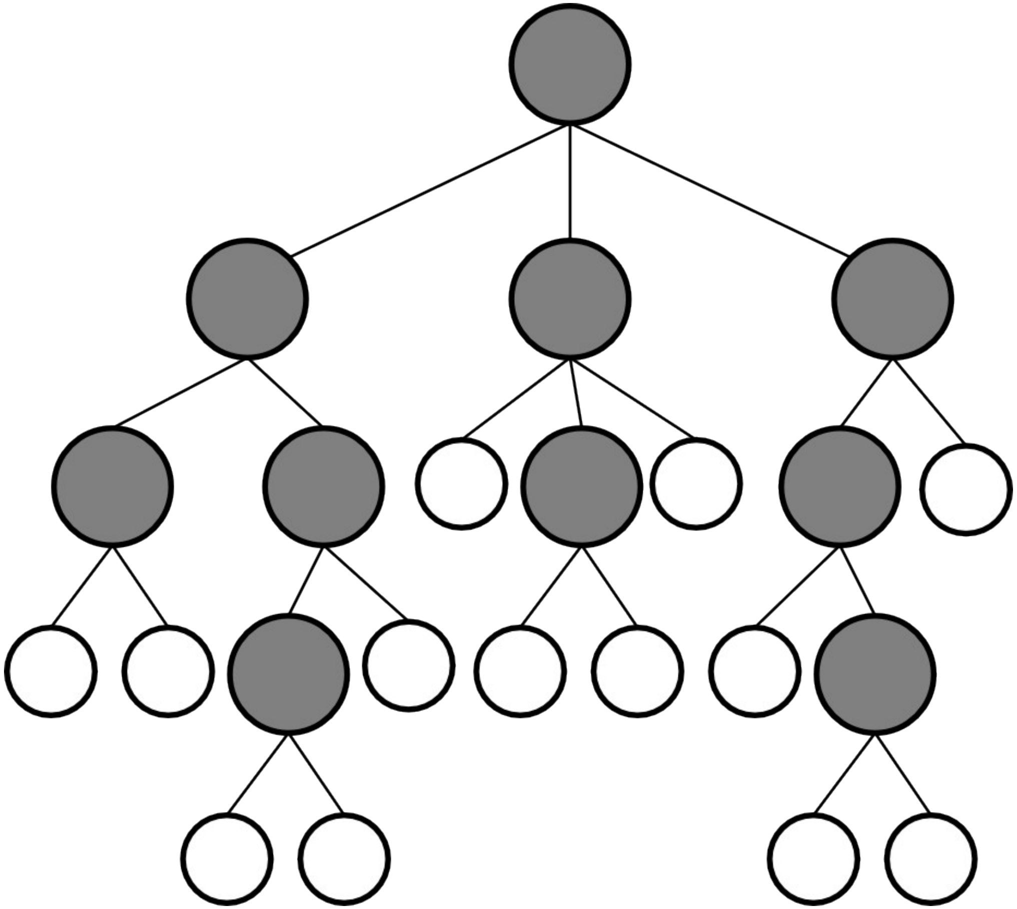
☑ using objects, game logic

It's All About Tasks



- ☑ Latent computation
- ☑ Succeed or Fail

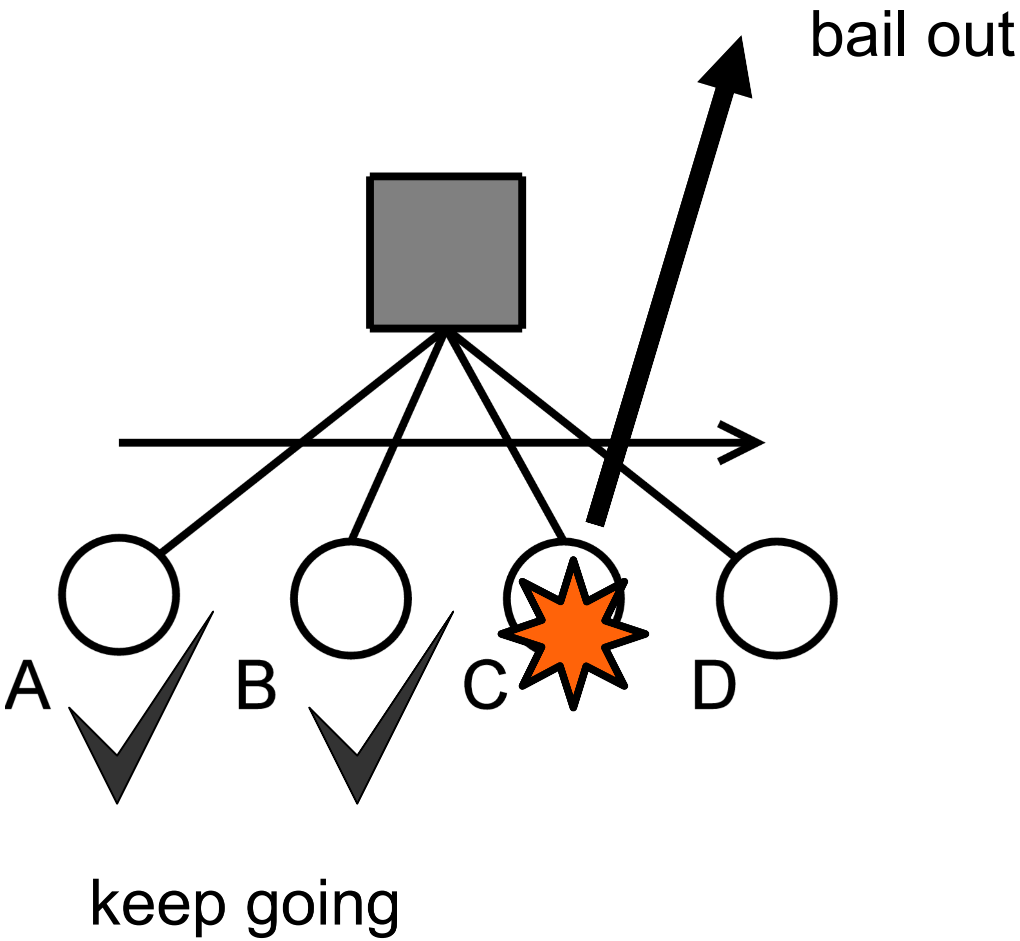
Building Complexity



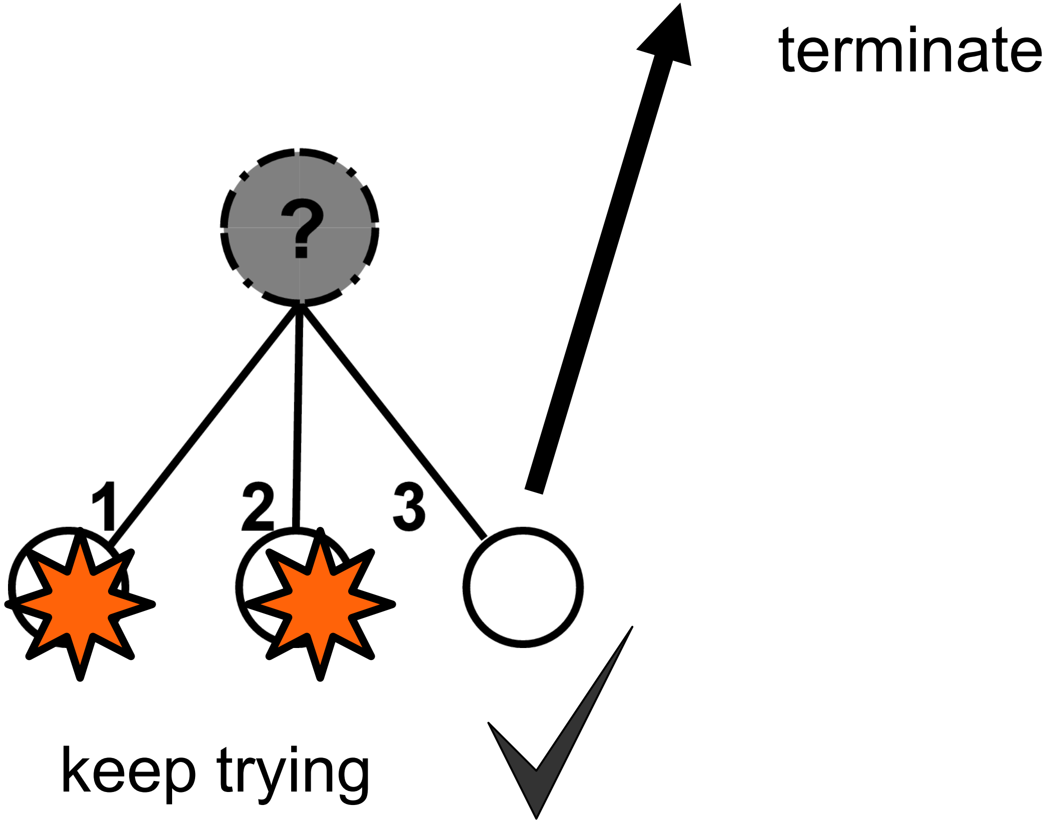
☑ branches manage the leaves

☑ done using composite tasks

Sequences



Selectors



A Powerful Model

☑ programming language basics

☑ statements and conditionals

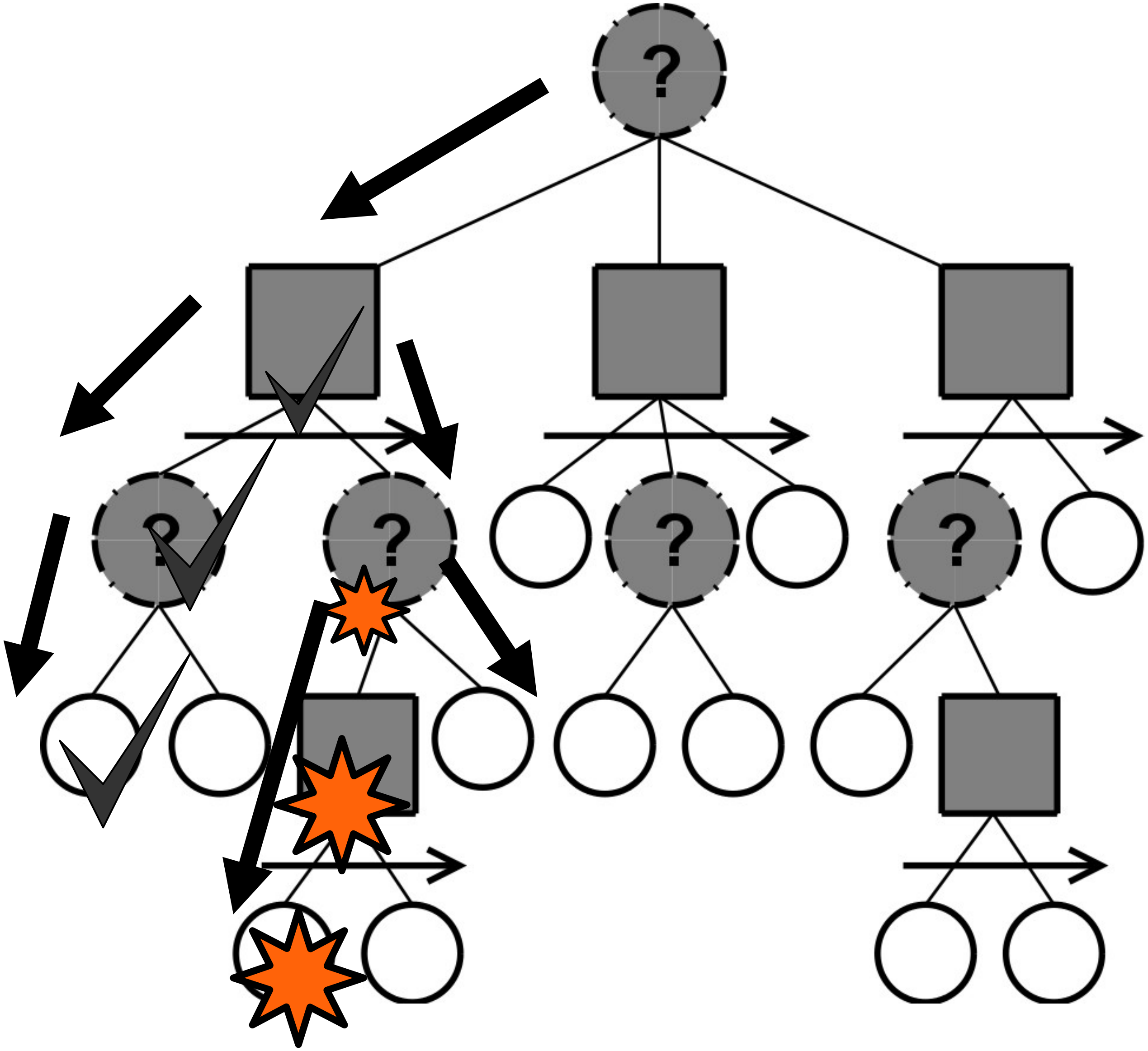
☑ essence of HTN planners

☑ and-or tree nodes

The Backbone of the Tree

With the right actions and conditions, I can build **all my logic** like this.

Dynamic Behavior as Search



Behavior Language

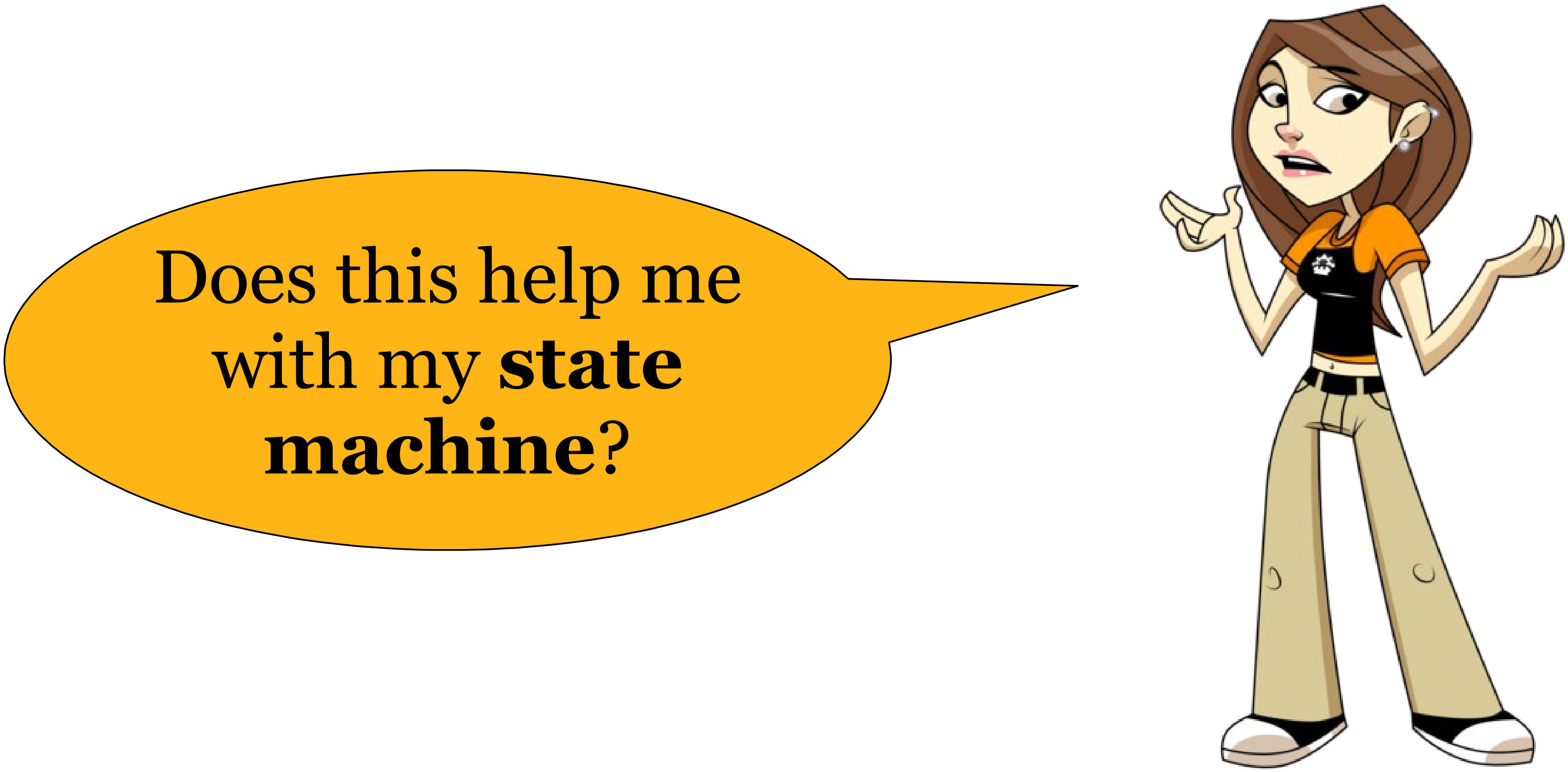
So I implement concepts **once**, and **reuse** them everywhere?

☑ **Standard** & **high-level** composite tasks

☑ Rather than **custom low-level** logic

In Practice

Improving Your HFSM



Improving Your HFSM

- ☑ make it easy to build sequences
- ☑ no need to (re)wire transitions
- ☑ easier to build purposeful behaviors

Design Principles

That gives me some **guidelines** to follow for editing all those transitions!

Improving Your Scripts

How does that help me with my **scripts**?

Improving Your Scripts

- ☑ provide better dynamic error handling
- ☑ by making it easy to build selectors

Software Patterns

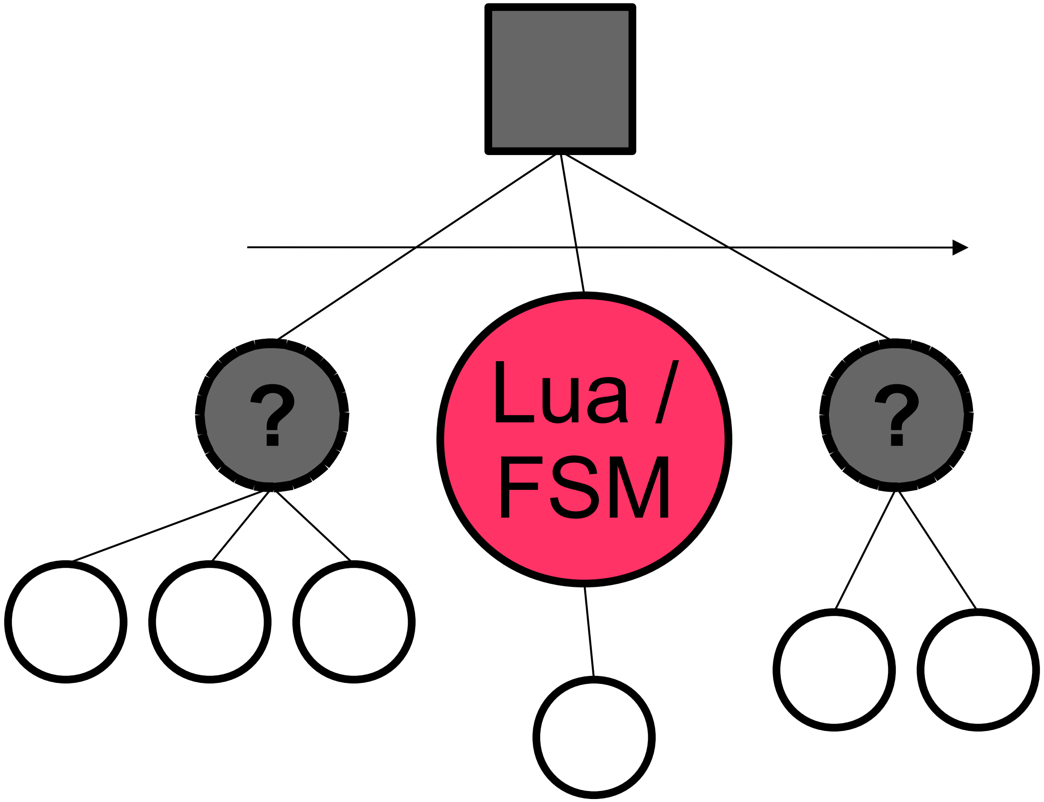
It'll help me think about my scripts on a **higher-level**.

Side Note: Efficiency

A behavior tree is no less efficient than a **well-written** script.

It can be **faster** if you build your AI engine as a behavior tree.

The Next Step



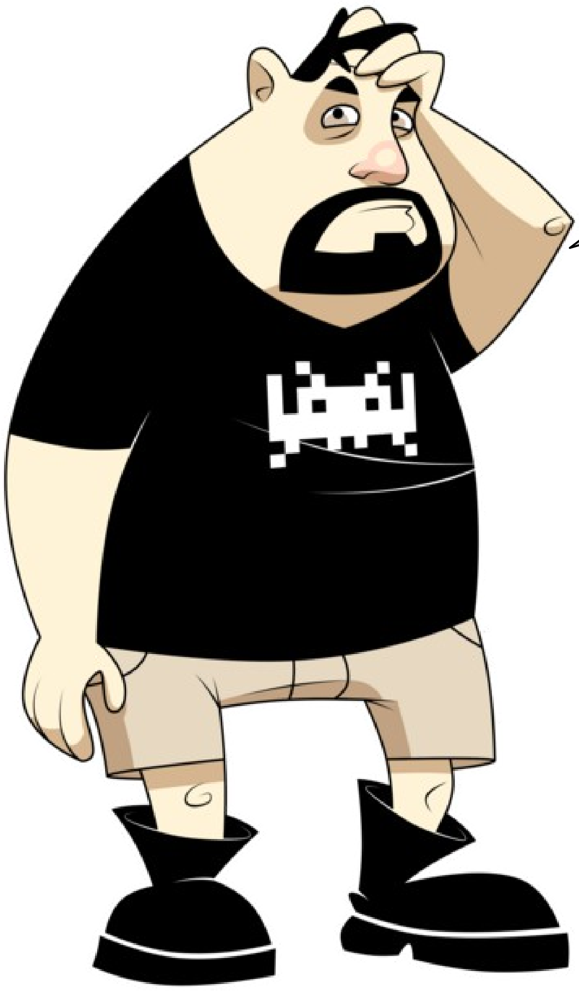
API **in**

API **out**

☑ Implement the AI as a behavior tree

☑ Support your current AI logic as a task

Taking It Further



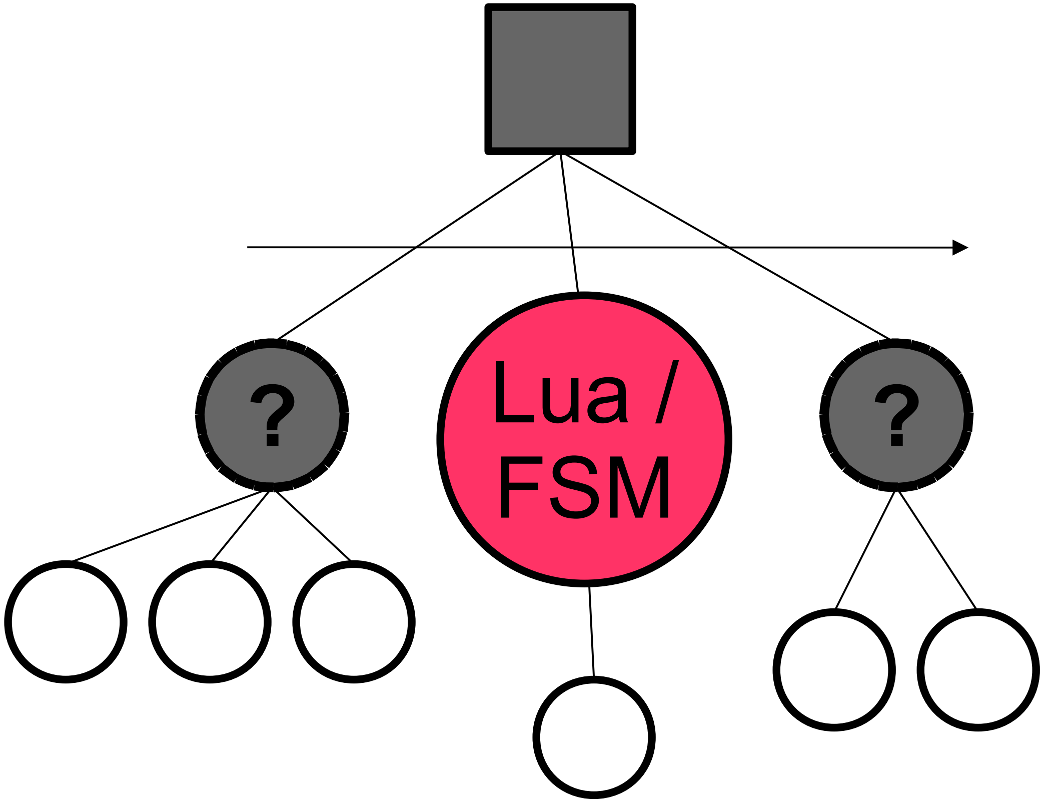
Ok. So how do you make a behavior tree **next-gen**?

It's all about **size** right?

Sure! But it helps if you **know how** to use it...

Scalability

Remove Bottlenecks



☑ Custom logic takes time to code up...

☑ Also much more likely to cause bugs

Embrace Design Patterns

☑ find common patterns

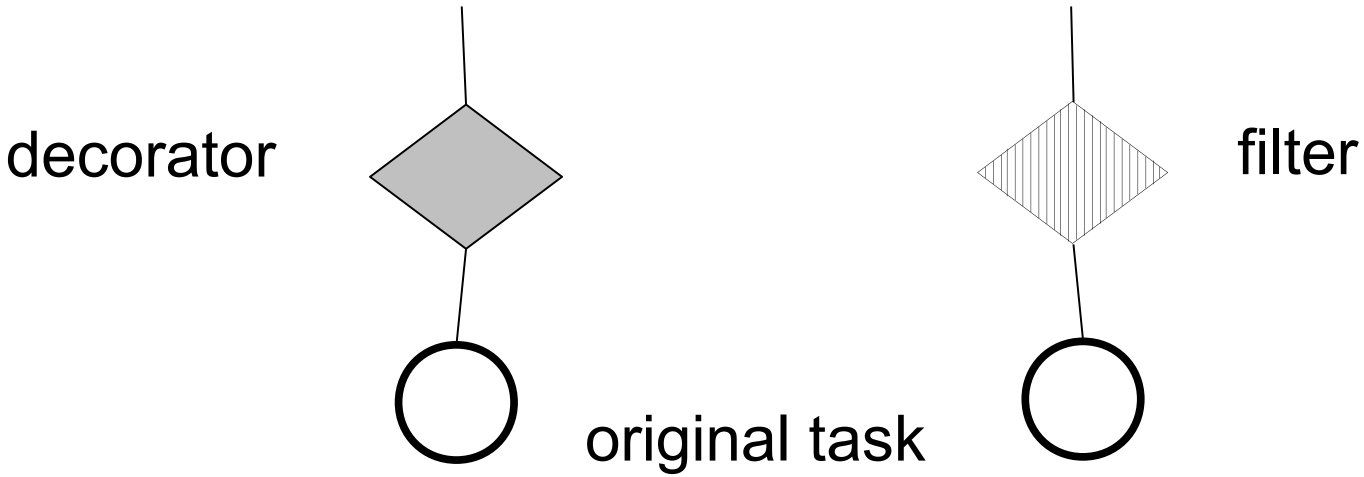
☑ implement them as high-level tasks

☑ it's much simpler and intuitive

☑ helps designers mix and match

Decorator Tasks

> “In object-oriented programming, the decorator pattern allows **additional behavior** to be added to an existing method of an object **without modifying** the original code.”



Decorating a Behavior

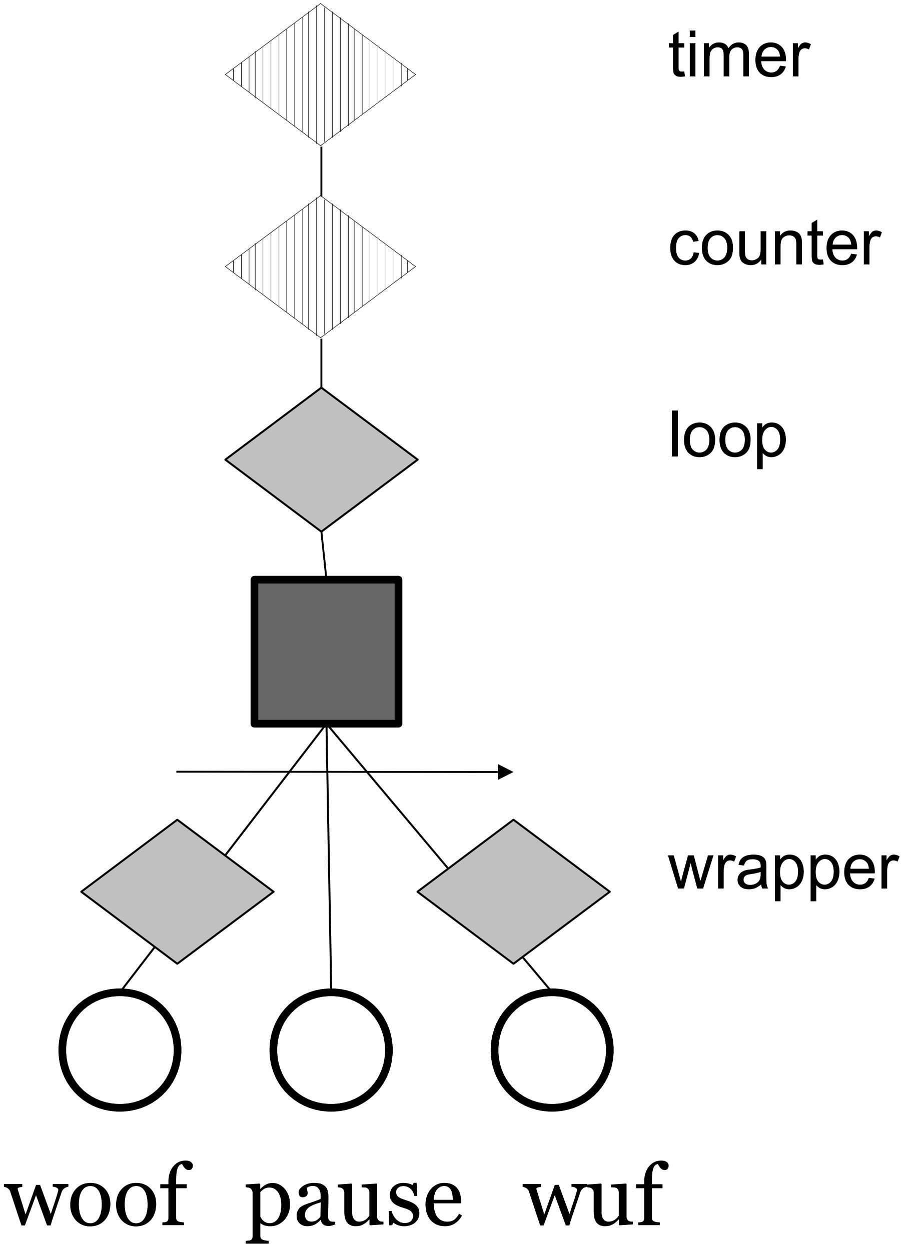
Bark,

multiple times,

ignoring voice failures,

at most *n* times in total,

no more often than every *x* seconds.



Incremental Development

These decorators can be implemented as they are required.

☑ it's a modular script interpreter

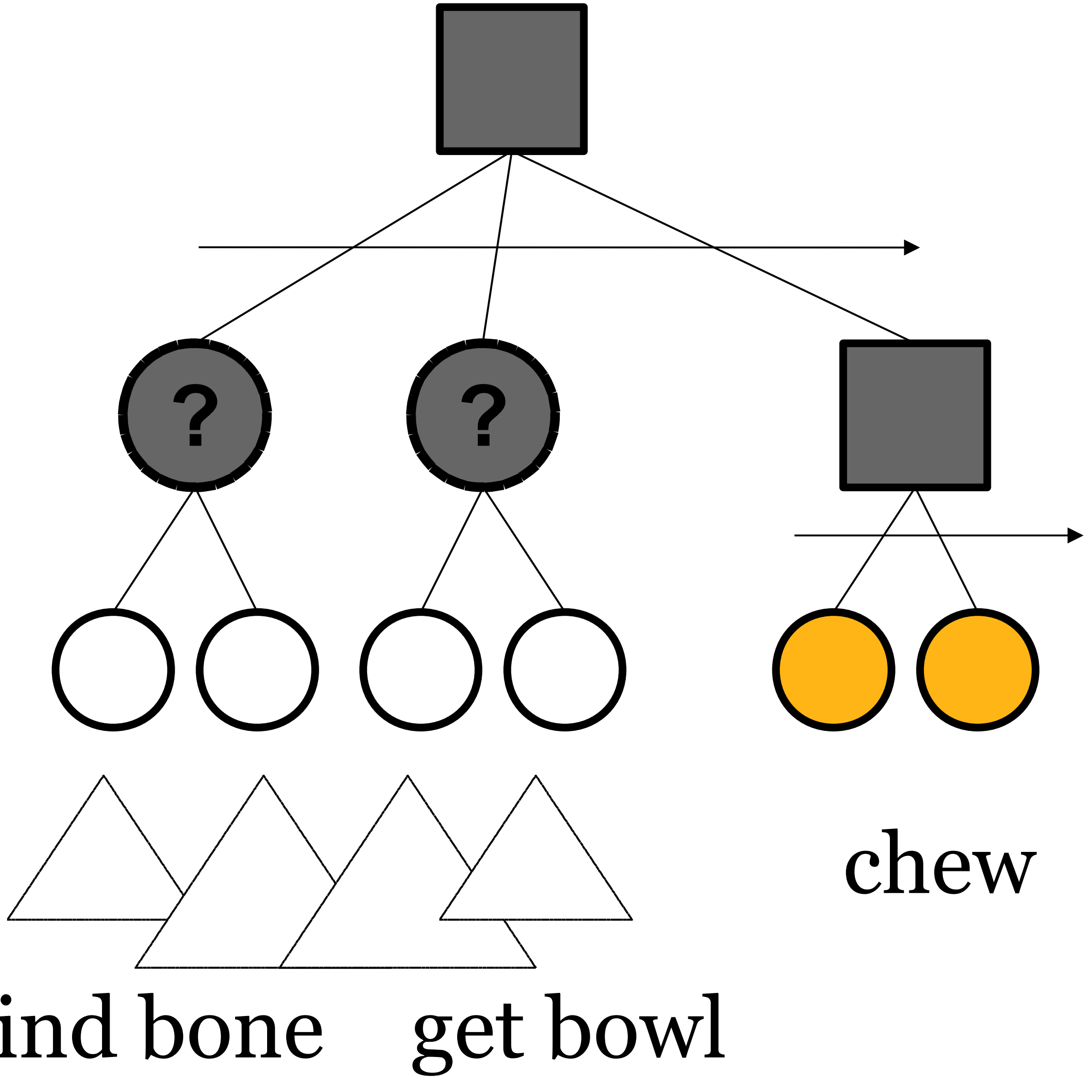
Goal Architecture

Goal Directed Behaviors

No large FSMs to control resources? Sounds nice!

bark, eat bone, walk to location, bite, jump, sit down, hide, chase, growl

Example: Eating a Bone



monitor them
for problems

finite
execution
time

depend on
other tasks

use minimal
resources

? ?

find bone get bowl

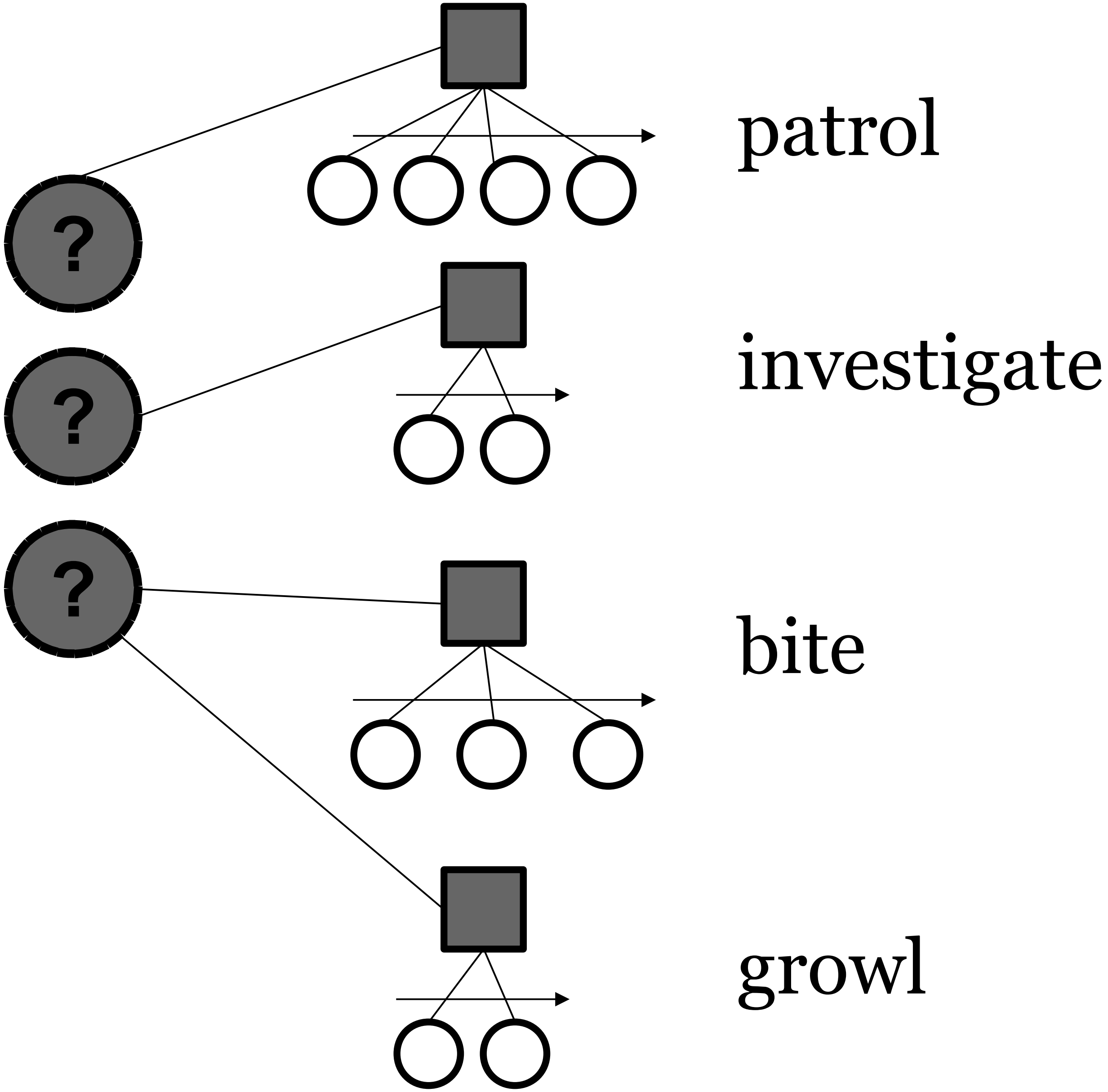
chew

A Little More Abstraction

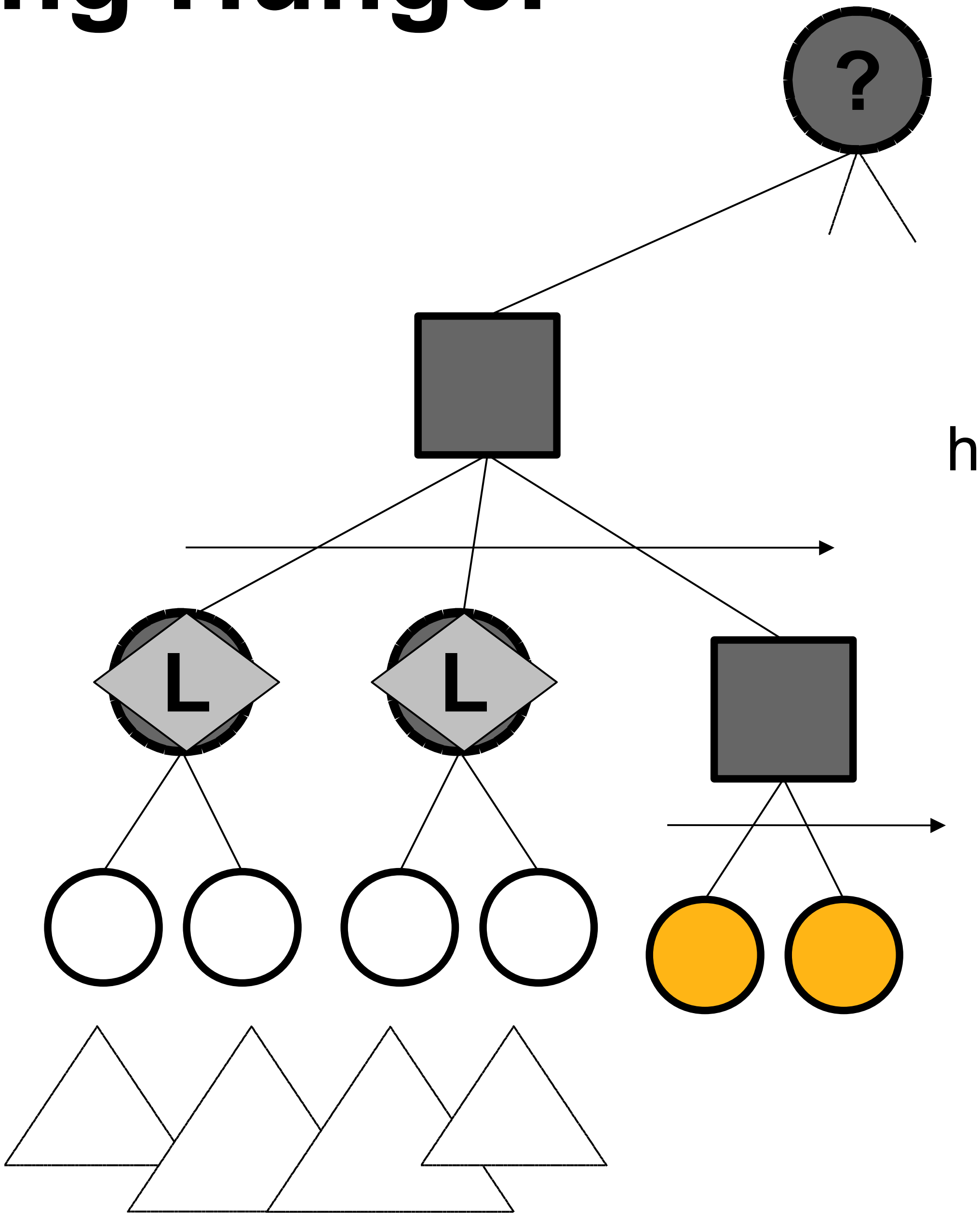
- ☑ separate WHAT: the **goals**
- ☑ from HOW: the **behaviors**
- ☑ easier to combine trees together

It's Just a Lookup Table!

Idle
Suspicious
Alert



Satisfying Hunger



satisfy
hunger

register with a
higher-level goal

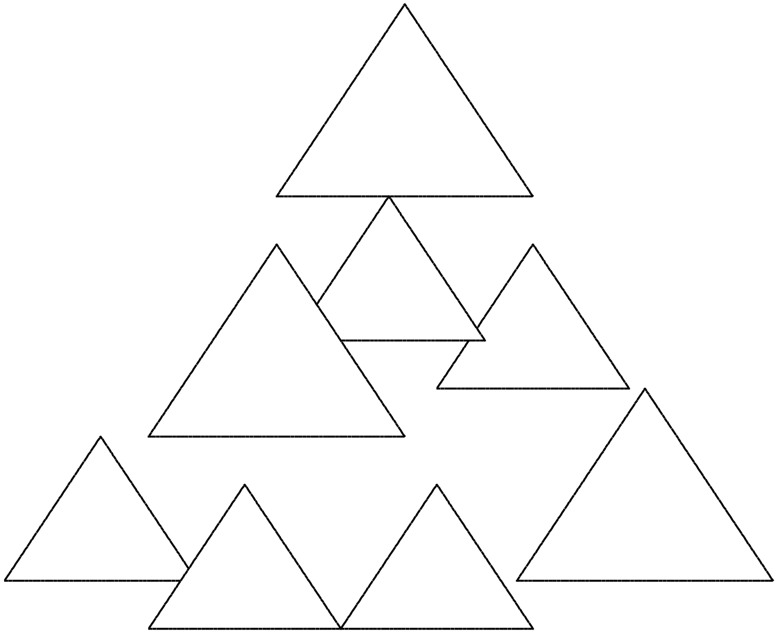
lookup
dependencies

Workflow

☑ build lots of small trees

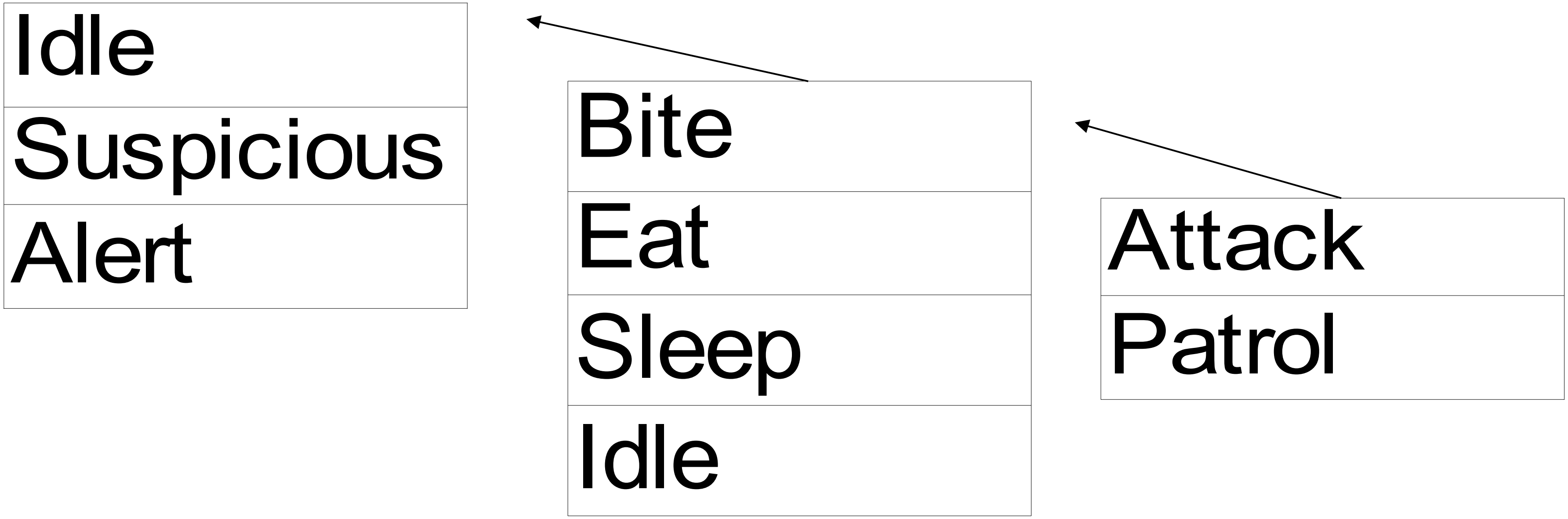
☑ connect trees via lookup decorator

☑ “search tree” assembled automatically



Customization

- ☑ use lookup table to customize AI
- ☑ per-character, per-group, per-type
- ☑ use simple inheritance of tables



Idle
Suspicious
Alert

Bite
Eat
Sleep
Idle

Attack
Patrol

Planning

Preventing Problems

Aren't planners supposed to **search ahead** in time?

Reactive Planning

☑ BTs use depth-first search

☑ simple to implement

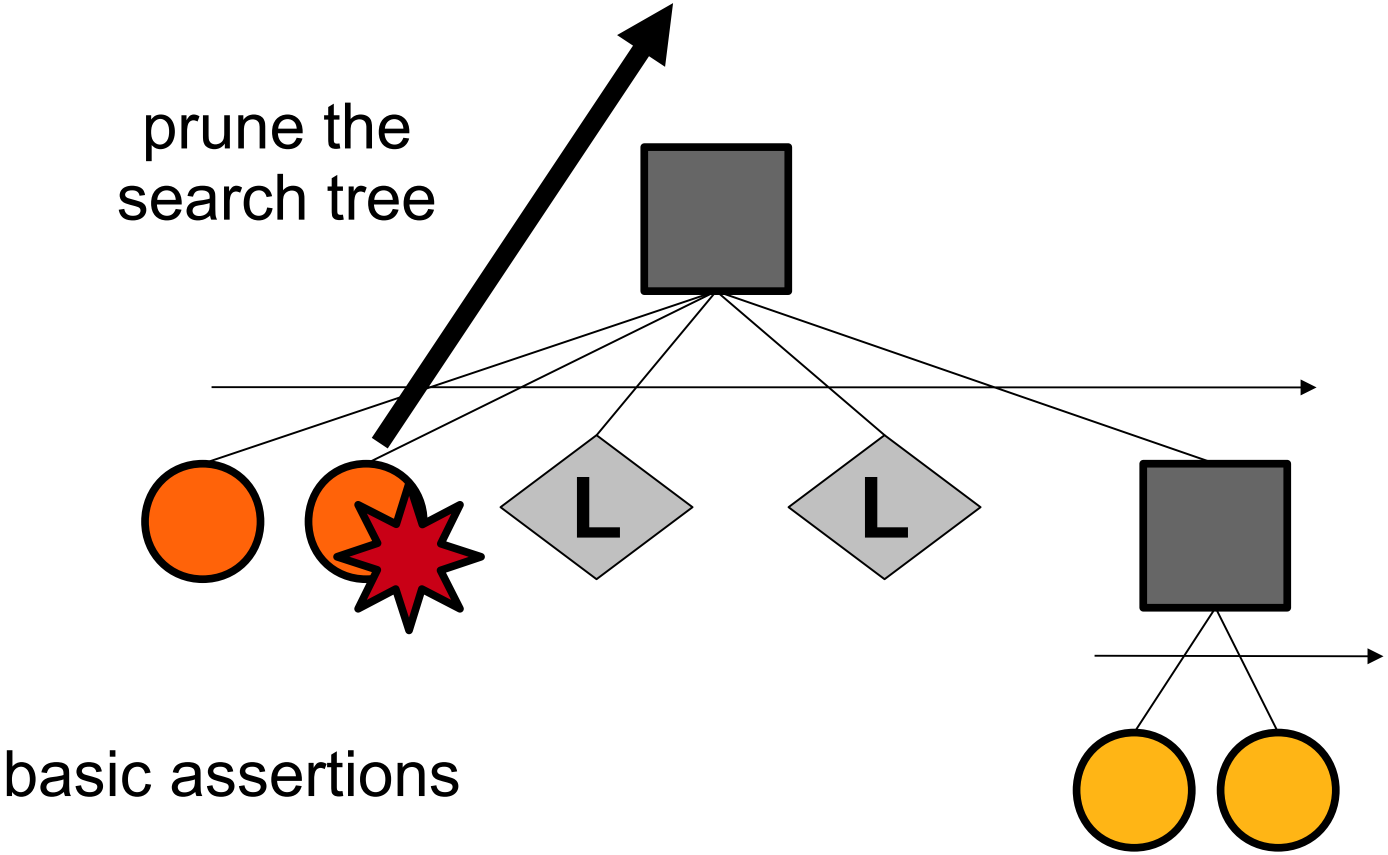
☑ but without lookahead

☑ can't prevent certain problems

Planning Tricks

But you can help reactive planners deal with **most situations** with these tips.

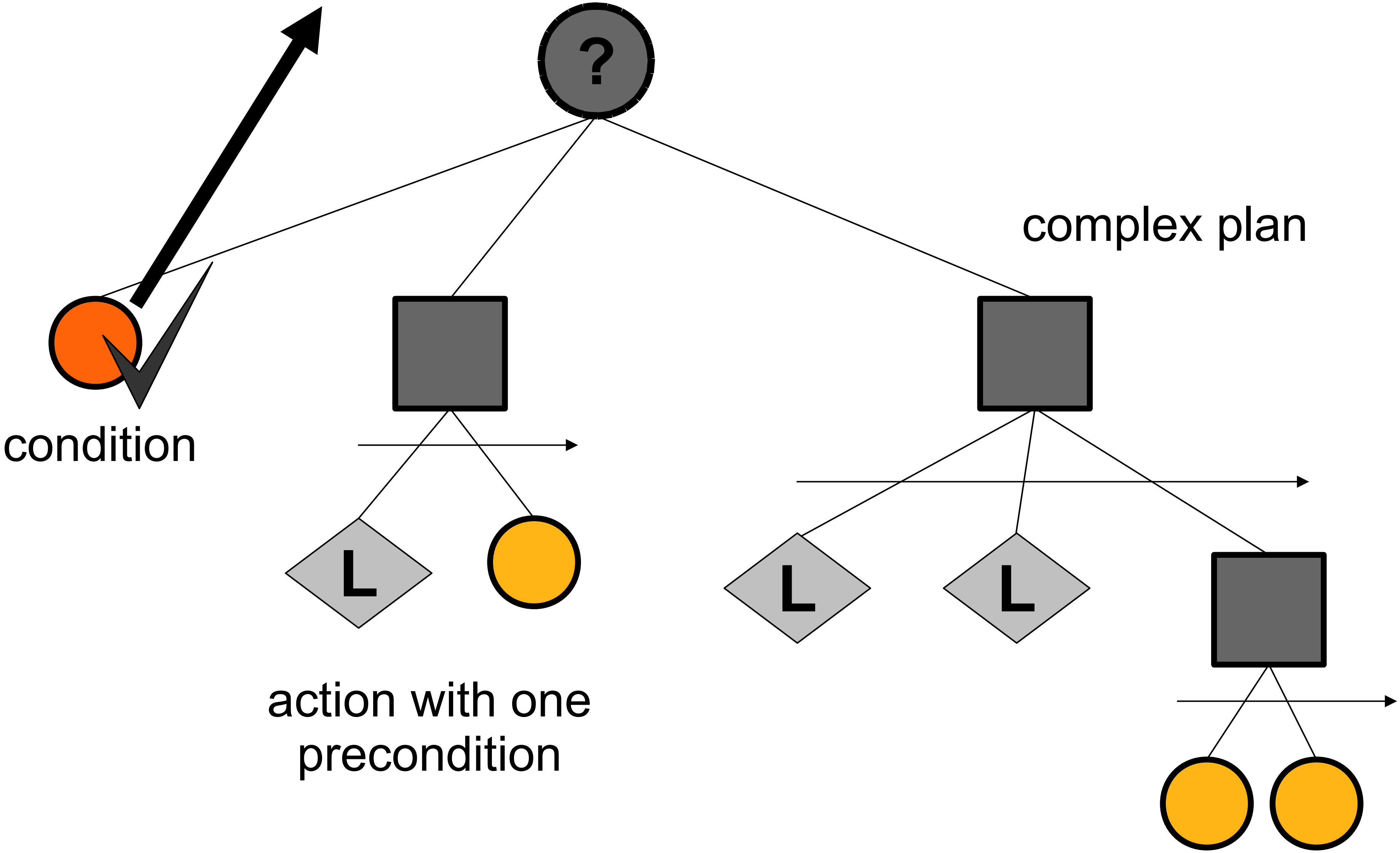
1, Use Assertions



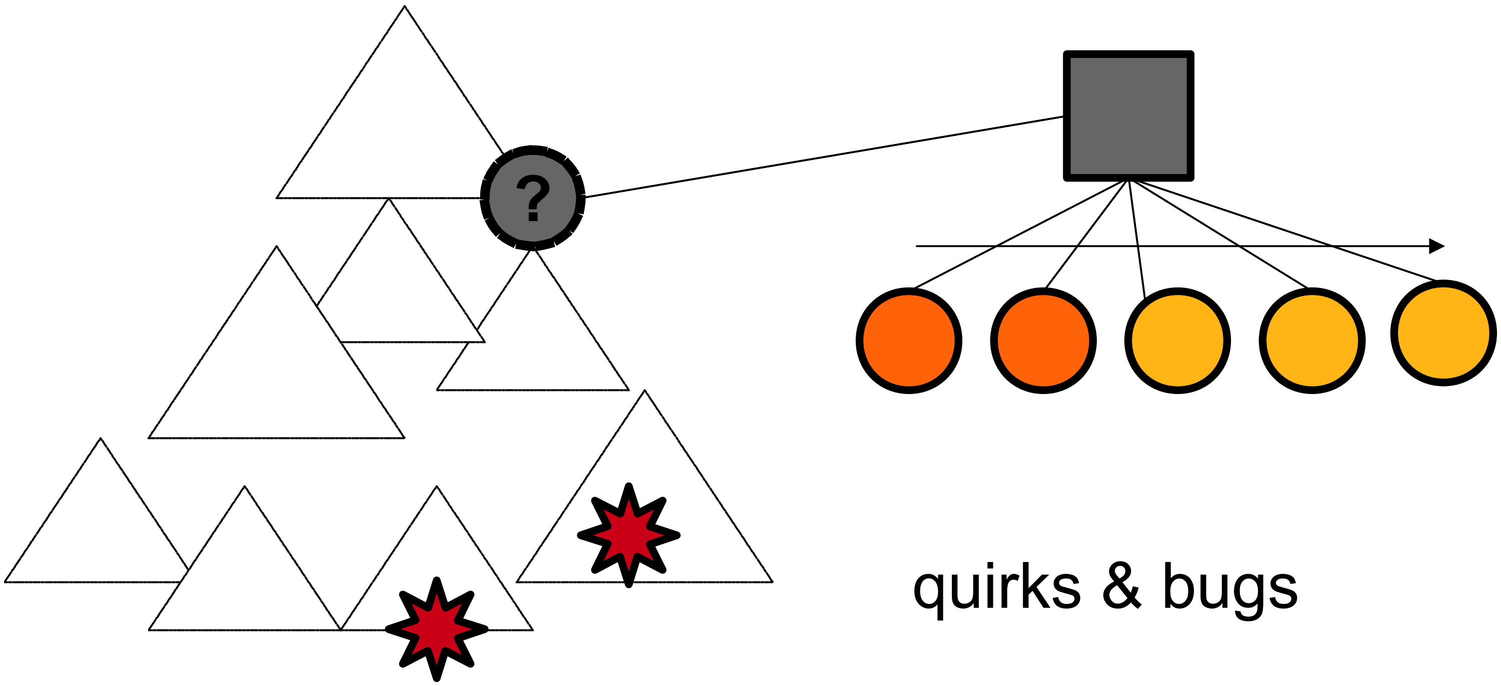
prune the search tree

basic assertions

2. Order Selectors



3. Build Specific Plans



☑ insert canned plans into the tree

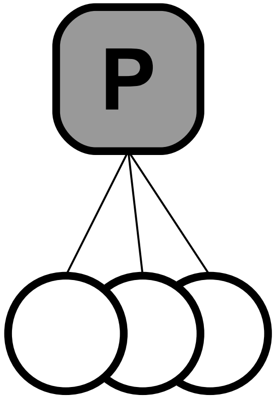
☑ it overrides the lower-level search

Better Efficiency!

These tricks should help **speed-up** my **lookahead** planner too!

Concurrency

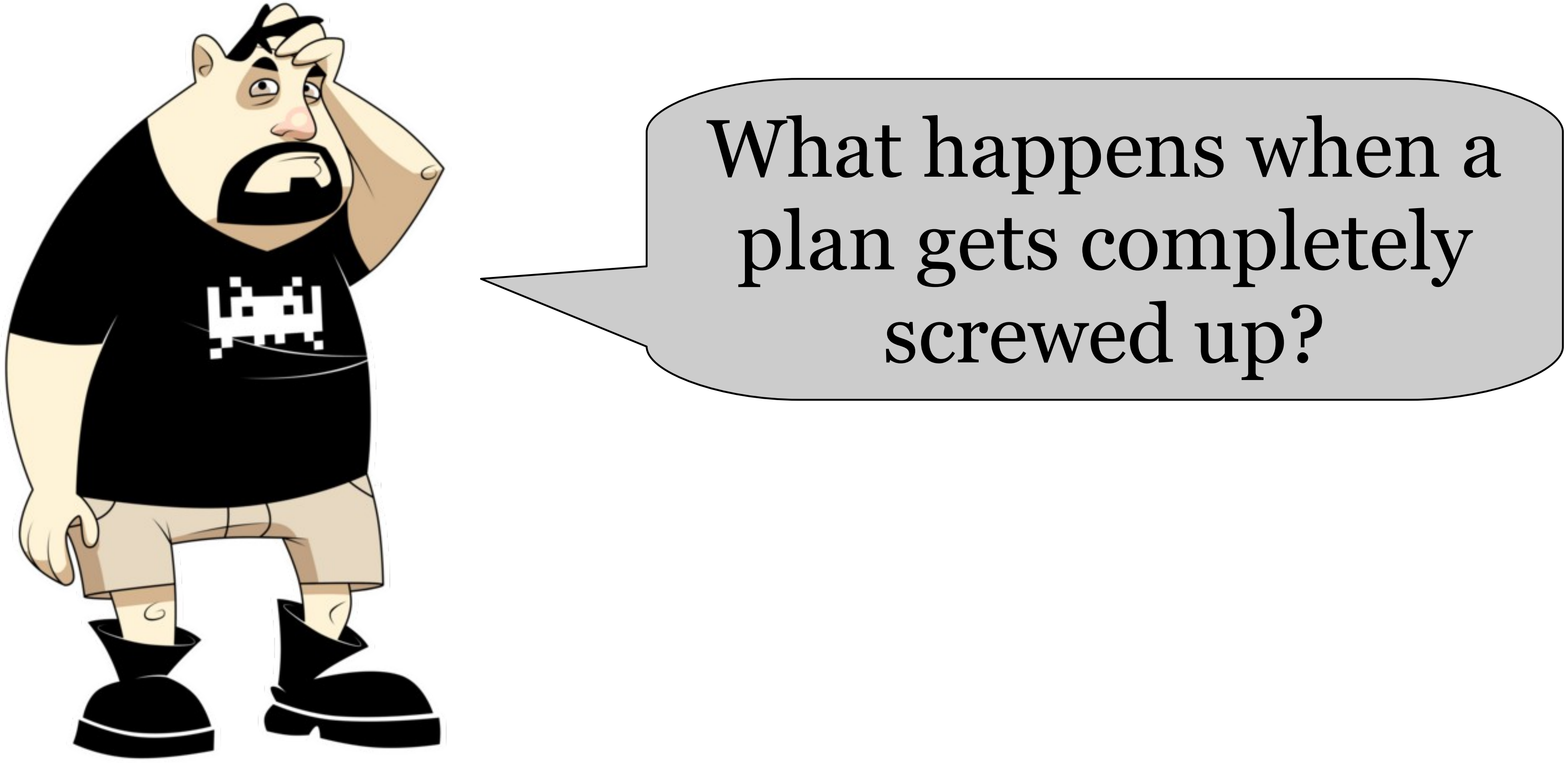
Parallel Composite



☑ options for when to succeed or fail

☑ based on number of child tasks

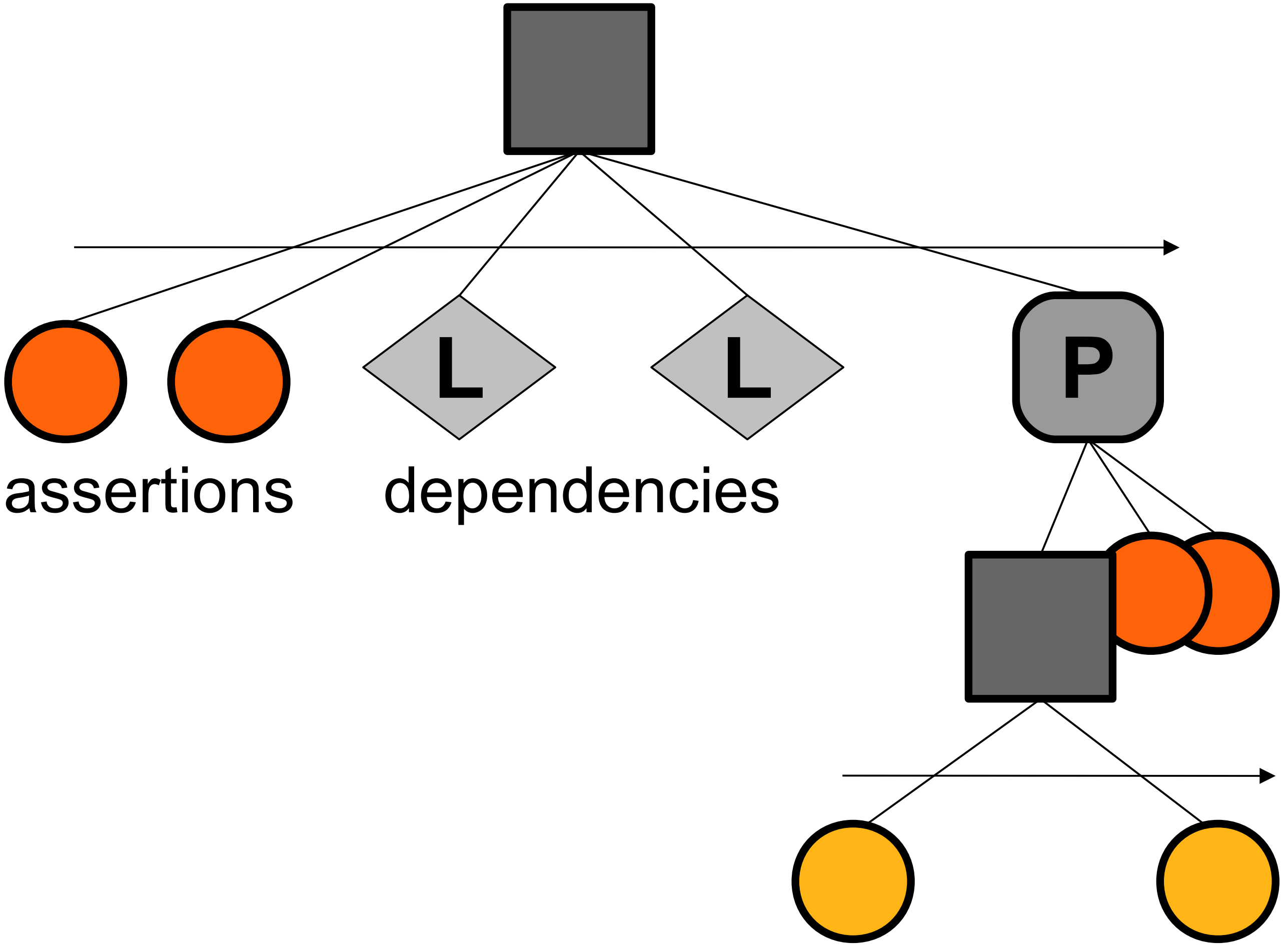
Dynamic Exceptions



☑ selectors only deal with local failures

☑ unexpected errors invalidate whole trees

Read-Only Concurrency



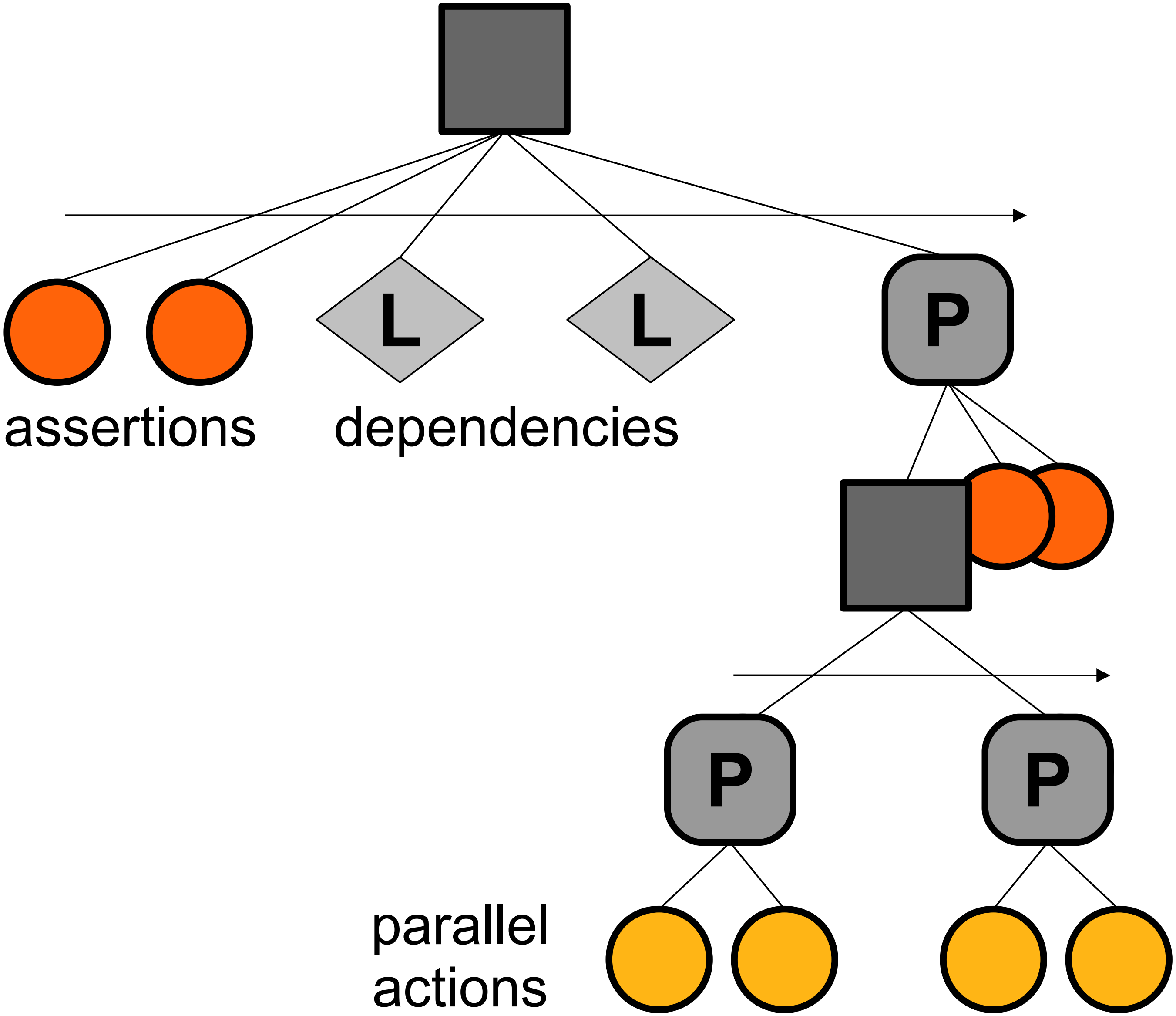
assertions

dependencies

monitoring assumptions

parallel conditions

Low-Level Concurrency



assertions

dependencies

parallel actions

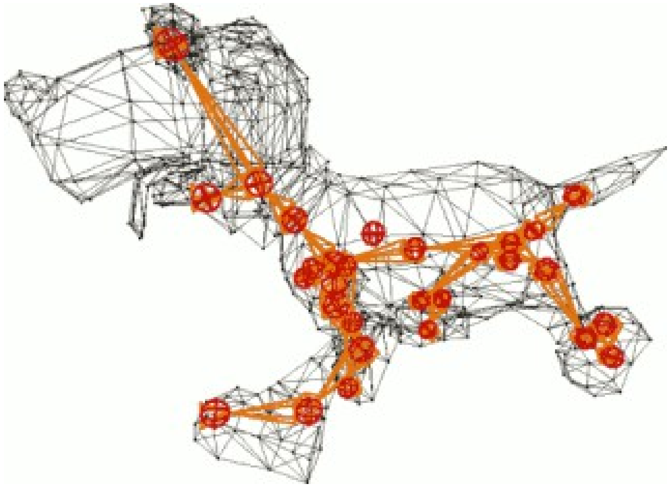
What about Full Concurrency?

So how can I run goal-directed behaviors at the same time?

Resource Allocation Conflicts

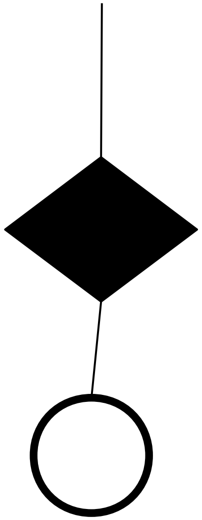
☑ playing one animation

☑ one vocal sound at a time

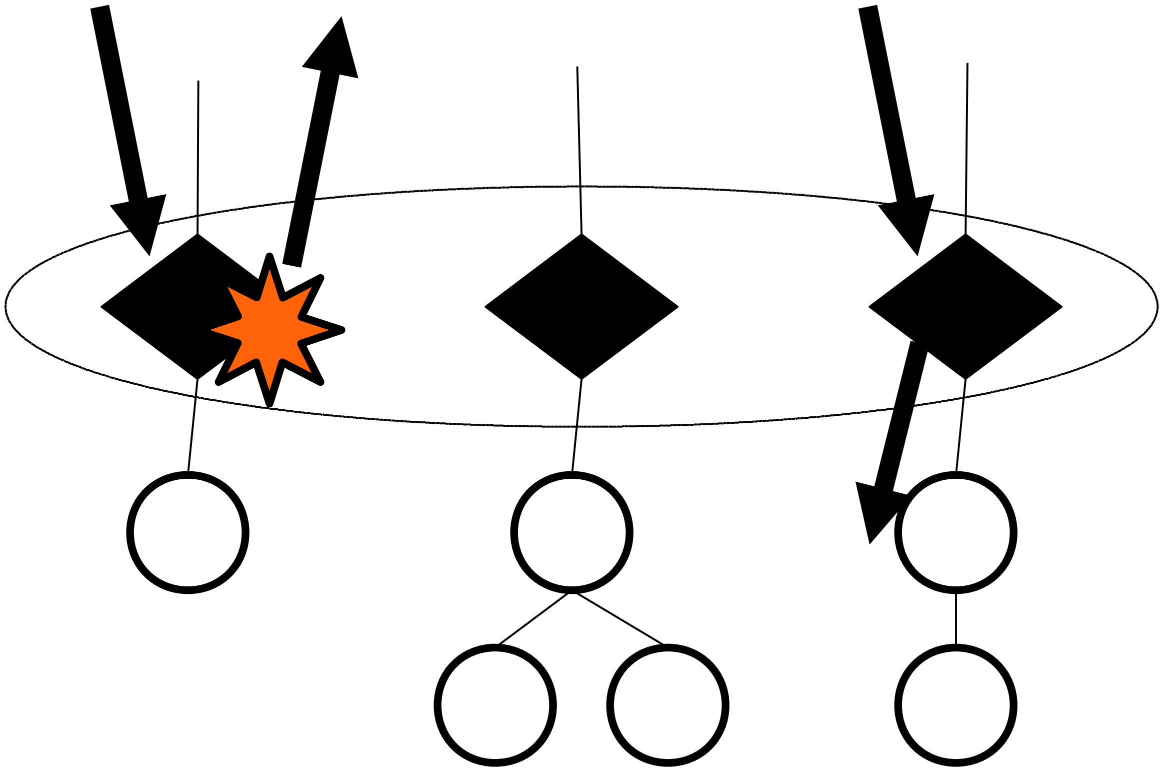


Using Semaphores...

“A **semaphore** is a protected variable for **restricting access** to shared resources in a multiprogramming environment, typically implemented as a counter for a set of available resources.”

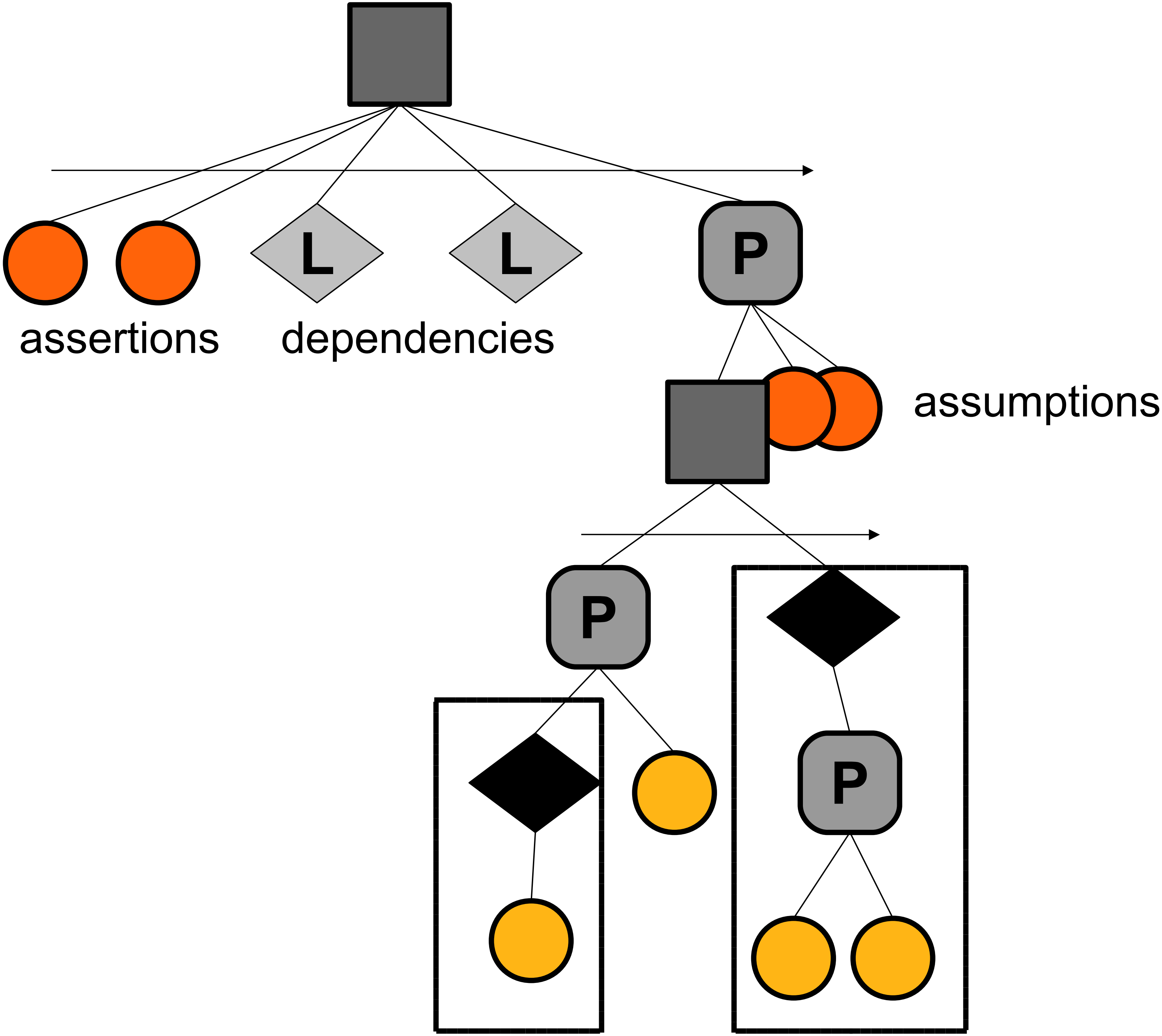


Resource Allocation



automatic
fallback

resource
locked



assertions

dependencies

assumptions

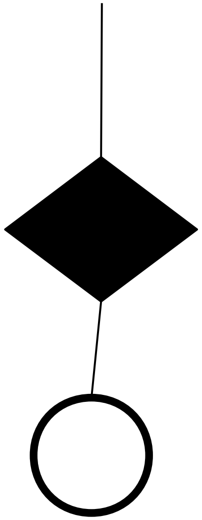
Example: Mouse Reaction

- ☑ patrol behavior locks the body
- ☑ ideally use a full body reaction
- ☑ head and voice not locked
- ☑ instead fall back to growl and stare

Multiple Applications

restricting
enemy fire

controlling
squad leapfrog



managing
group
behaviors

limiting
high-level
orders

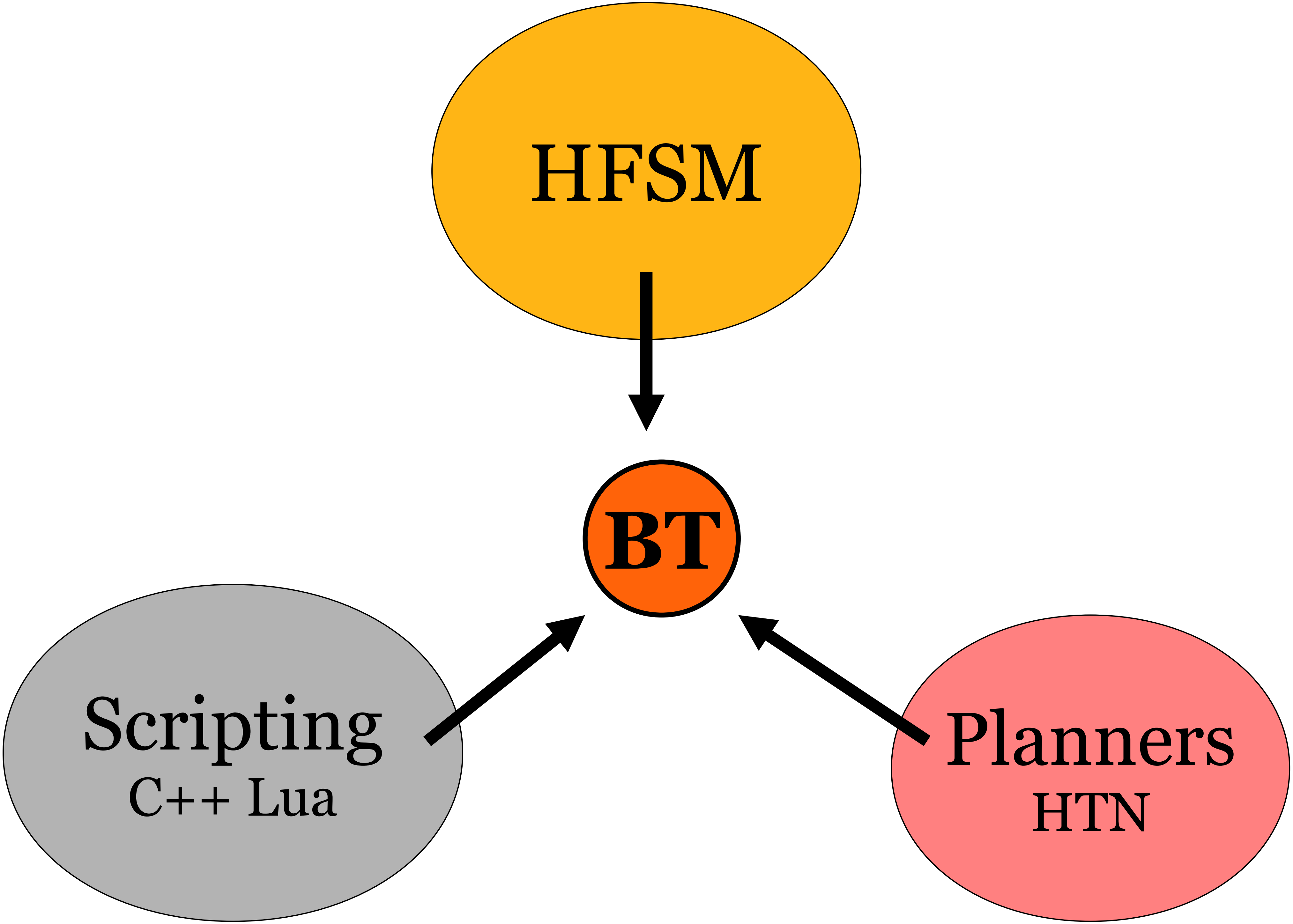
A Low Risk Solution

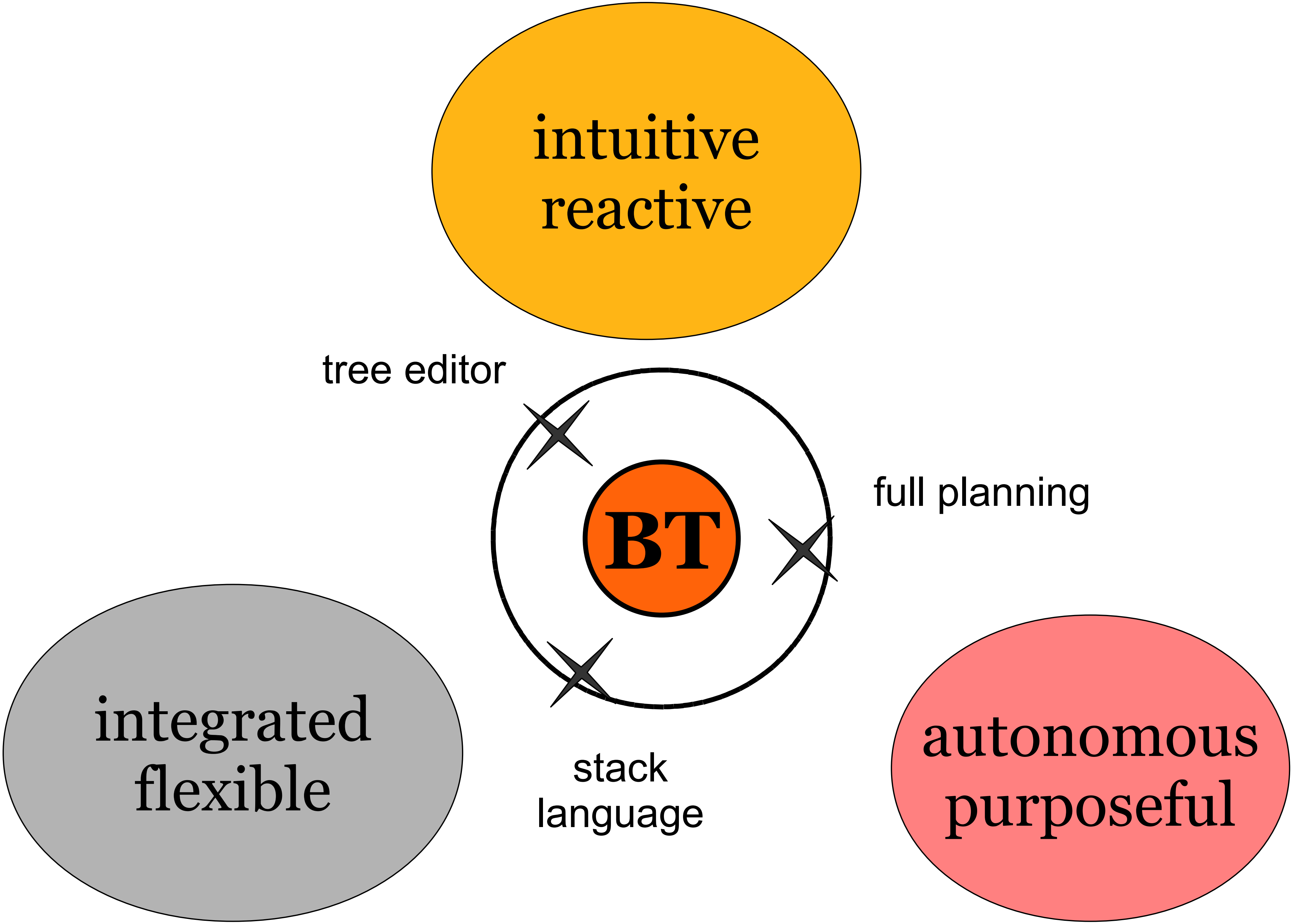
That seems **simple** to implement. What's the catch?

Advanced Concurrency

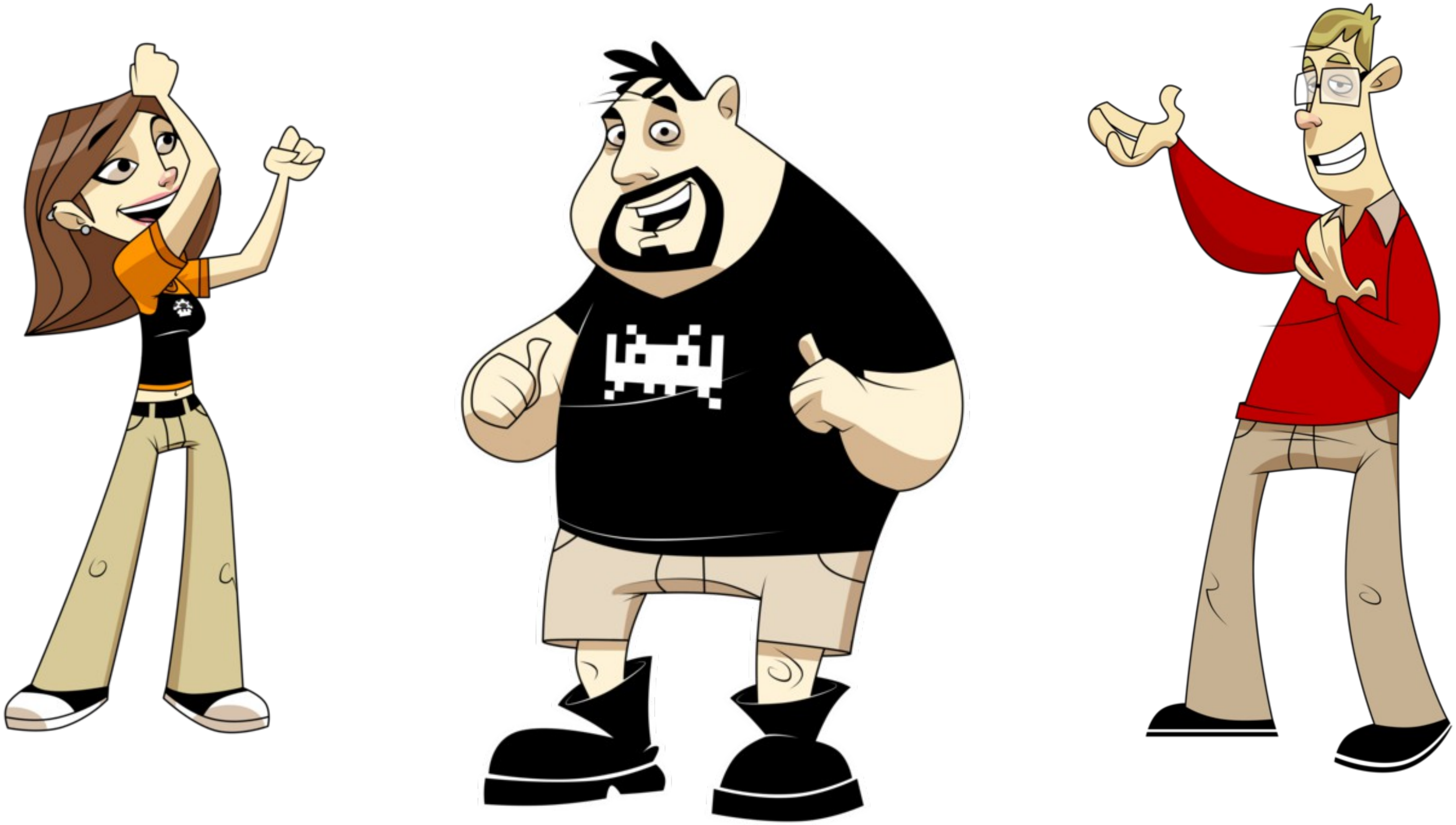
- ☑ Behavior priorities
- ☑ Queuing up behaviors
- ☑ Quality of service
- ☑ Interrupting behaviors

Summary


HFSM
BT
Scripting
C++ Lua
Planners
HTN


intuitive
reactive
tree editor
full planning
BT
integrated
flexible
stack
language
autonomous
purposeful

That's All Folks!



Behavior Trees

for

Next-Gen AI

alexjc@AiGameDev.com