LINKING DATA & ACTIONS ON THE WEB

RESTfest 2012

INTRODUCTION

- **Stuart Charlton (@svrc)**
- Independent Consultant on cloud, REST, agile delivery, and IT strategy
- Weblog: **Stu Says Stuff** http://www.stucharlton.com/blog
- Many thanks to commenters and twitterers on this topic

DISCLAIMERS

- This is a sequel (of sorts) to my 2011 WS-REST Keynote
- Mostly this is my naïve and incomplete thinking on how to progress the state of the art.
- Heavily in debt to ideas from Miro Samek (Statecharts) & Alex Champandard (Game AI), and Jim Webber / Savas Parastadtitis / Ian Robinson (REST in Practice)
- I haven't shared code to back this stuff up so cannot be trusted to be correct or even sane ... but bear with me, I think it's at least going to be entertaining.

BACKGROUND

- The Web Architecture has been an immense success...
  - ... and yet, we can do better.

- There's a need to design the software for the ***write side*** of the web to **scale** and become nearly as **serendipitous** as the ***read side***
  - Is this even possible?

- *"There's no **write side of the web** really....*
  ... no, in fact, it's all GETs"

MY CLAIMS

- The Web architecture's core strength is in encouraging small pieces of independent **agreement** to be **linked** together and **shared**; we're **missing** some agreements for writes

- The Web architecture encourages clients to be designed as **agents** in a **dynamic** information space

- There are **practical** approaches to programming **agents** in a **dynamic** environment

- It should be possible to create a **general purpose** media type for **systems** to manipulate state on the web, in lieu of more specific media types.

WHY?

WHY?

- **Troublesome Growth in Centralized Services**
  - e.g. Facebook, Twitter, Google+
- Centralization occurs in part because of a **limitation** of the current web architecture
  - This is why we have 4 or 5 incompatible "like" buttons everywhere

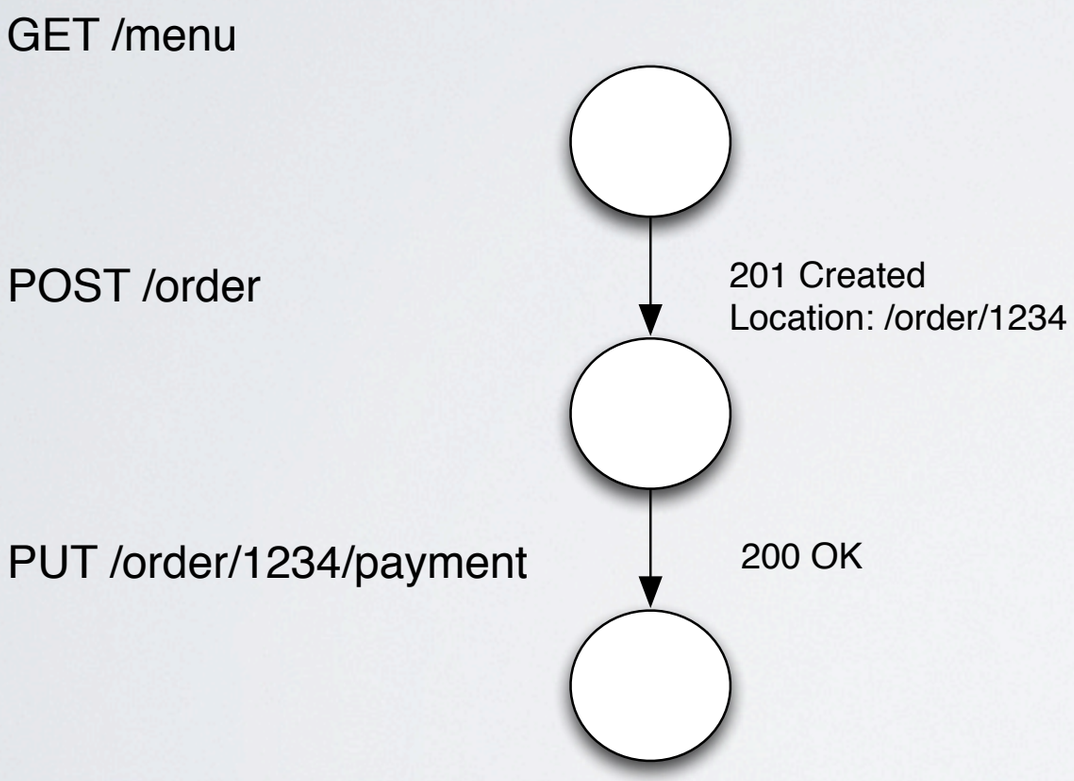
WHY?

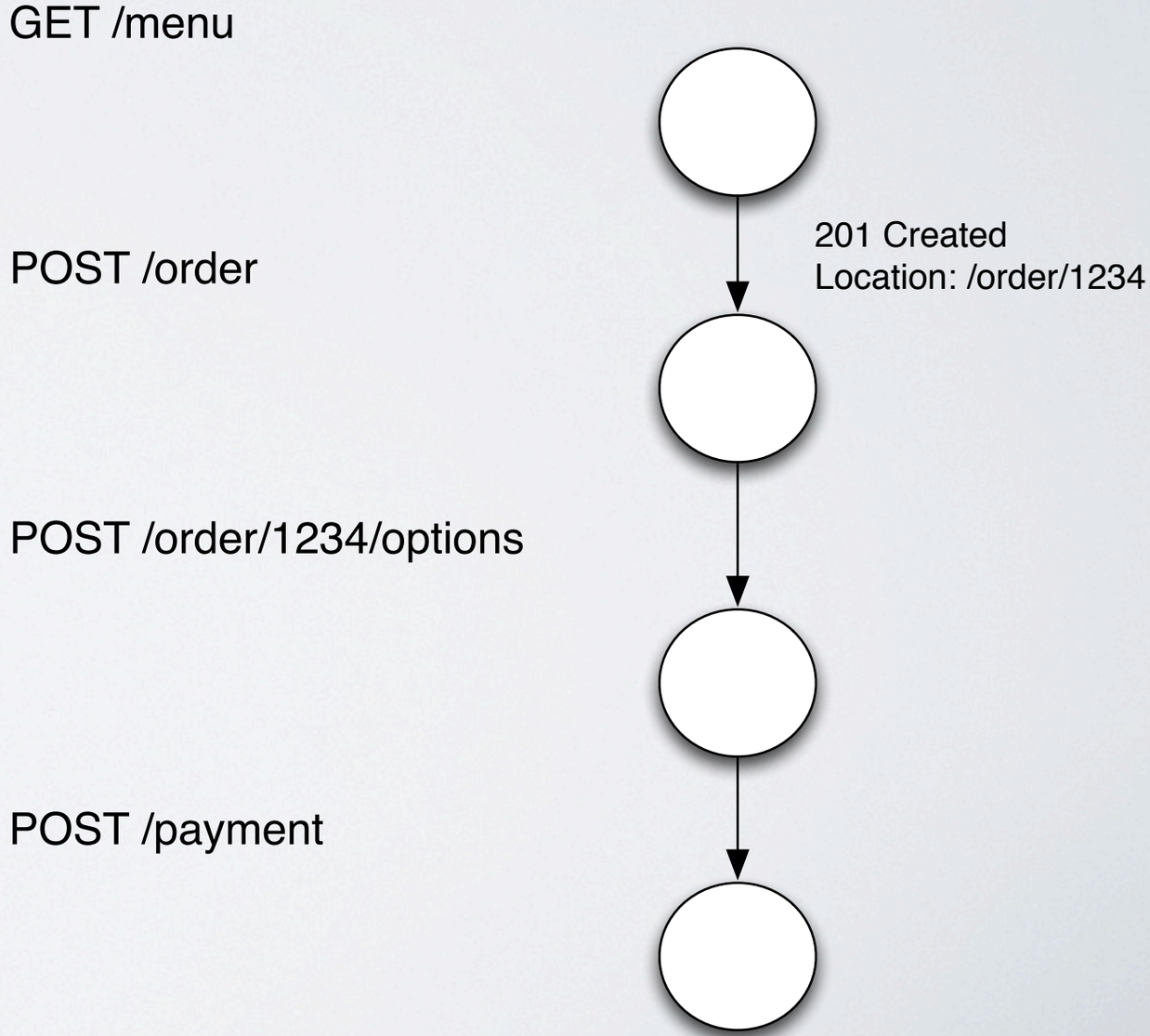
- **REST APIs and Integration**
  - Custom media types are the current approach...
  - ... but this generally is a transitory solution
  - "RESTful" design thrashing due to lack of prescriptive guidance
    - Would be reduced with more generic media types (e.g. as with HTML, AtomPub)

WHY?

- **Interoperable Domain Application Protocols**

- *RESTbucks vs RESTual Roasters: "All I wanted was a coffee - but I had to rewrite my client!"*





WHY?

- **Programming models <u>matter</u>**

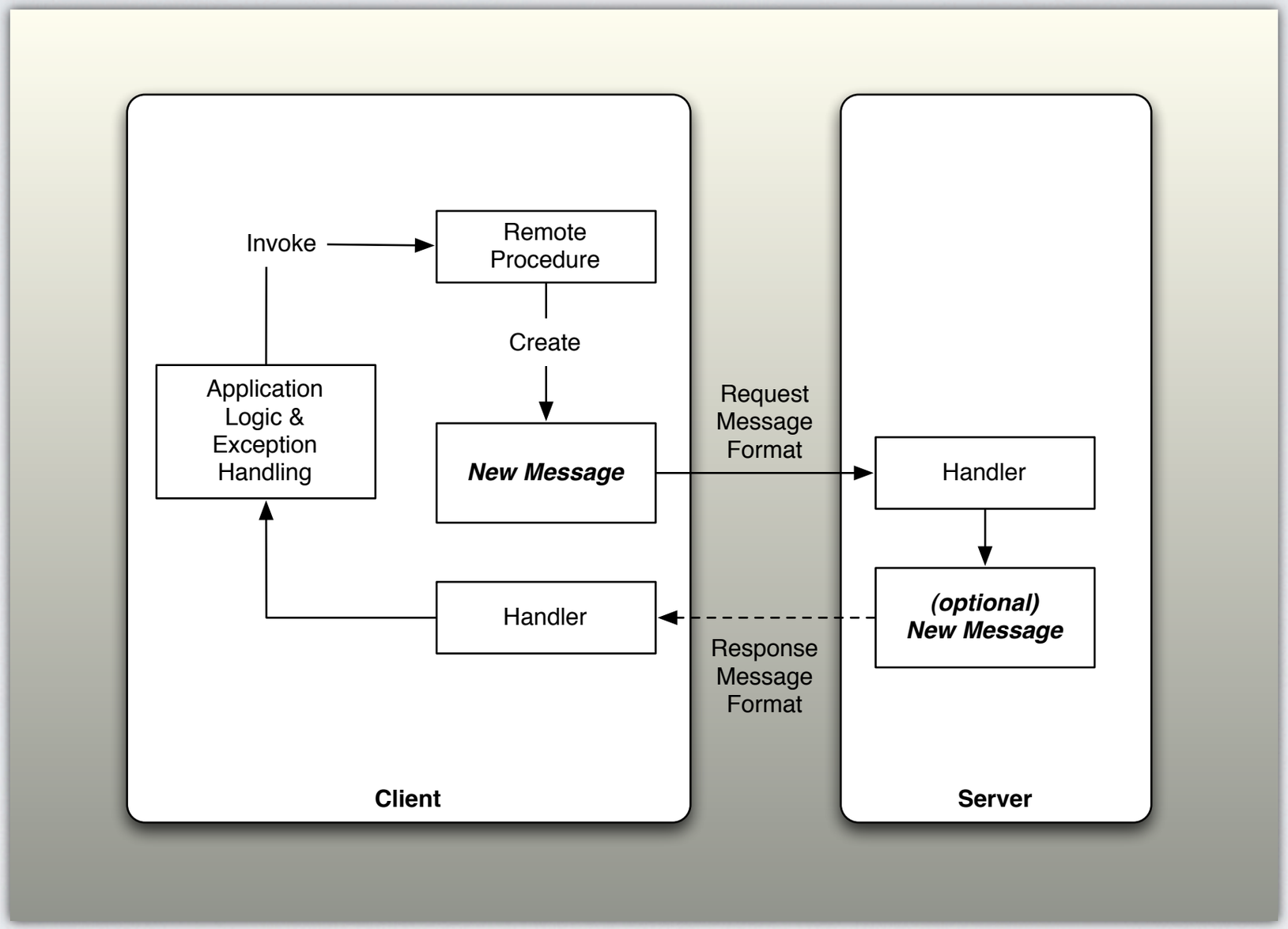
- In particular, the **client's** model of how it interacts with the server

- We're still *often* stuck on
  client method = network method (dynamic RPC)

PROGRAMMING

Or Scripts vs. Agents

CLIENT/SERVER PROGRAMMING



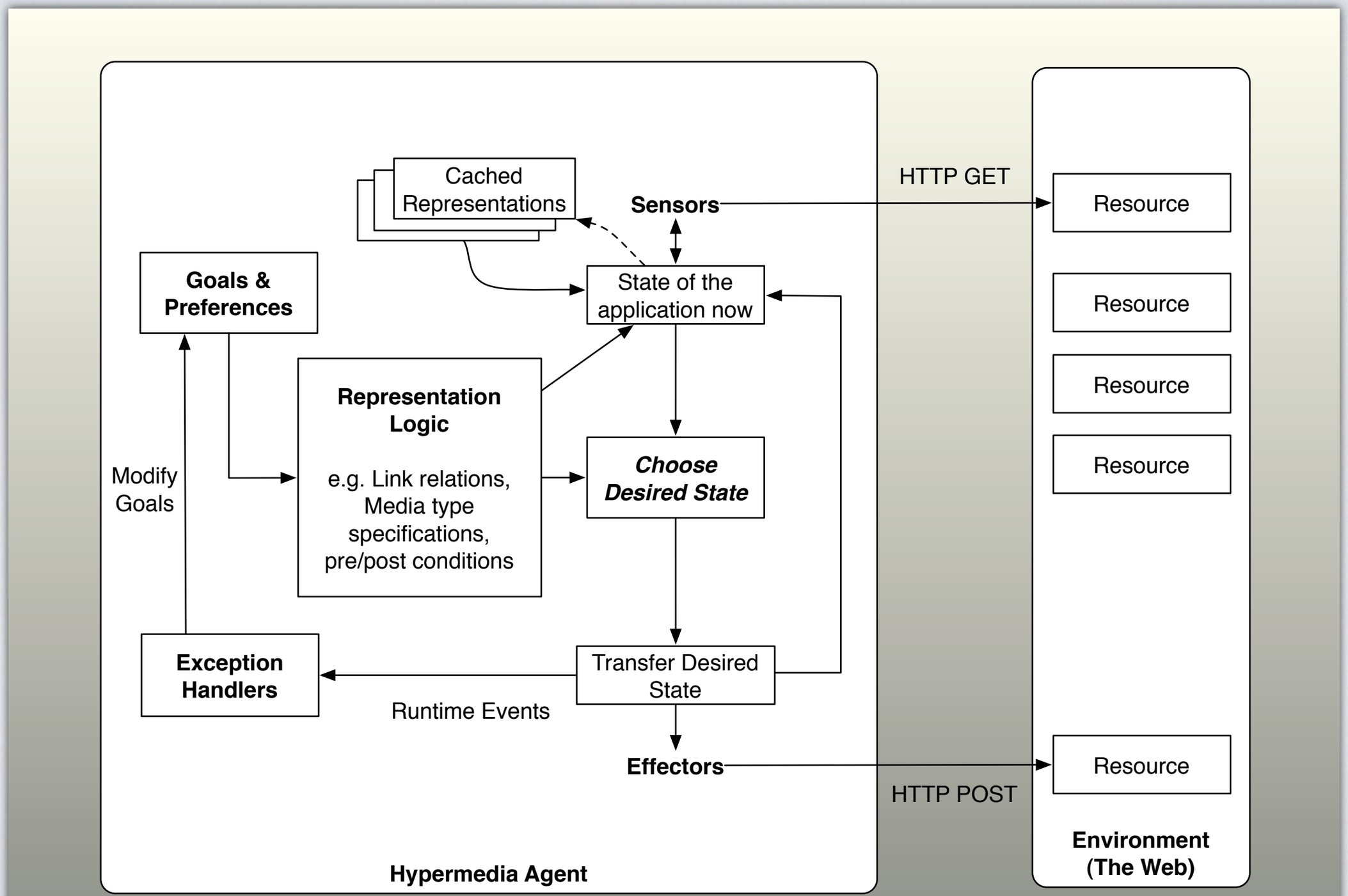
REST RAISES THE LEVEL OF ABSTRACTION

- The **message** vs. the **resource/representation**

- Traditional Client/Server:
  Client is a **script** sending/receiving **messages**

- REST:
  Client is an **agent** acting in an **information space**

HYPERMEDIA PROGRAMMING



QUALITIES OF THE AGENT

- Goal-Directed
- Reactive
- Hypermedia workspace
  (affordances, cached representations)
- Sensing can be done to pick up on effects

TWO POPULAR WAYS TO PROGRAM ON THE WEB

- **Linked Data**
  aka. "CRUD"

- **Scripting against a REST API**
  aka. "Dynamic RPCs across a pile of Link Relations"

A THIRD WAY?

Linked Data

Great for interoperable analysis
Not so great for doing things

REST APIs

Everyone understands it
Mostly works well
Not often interoperable across sites

REQUIREMENTS

- Need to be able to describe ***actions***
  - An action is a desired side-effect ; not an HTTP method
  - Build higher level behaviors while maintaining uniformity of protocol
- Preferably describe these **serendipitously**
  - Searchable, linkable, follow-your-nose
- Preferably **without tight coupling** to HTTP methods
  - One server's action may require a single POST,
    another's may require PUT-POST-POST
- Present and process data in a **uniform** way (ala. linked data)
  - Express **requirements**, **assertions** and **conditions**

A THIRD WAY

Interoperable actions and data

An alternative (not replacement) to custom media types

BEHAVIOR TREE

- A **constrained** hierarchical finite state machine (HFSM)
- Approach to programming Game Agent AI (Halo 2, 3, GTA)
- Constrained, how?
  - *Goal-oriented* HFSM
  - *Reusable* sub-states to enable "programming by difference"
  - *Patterns* for common programming constructs

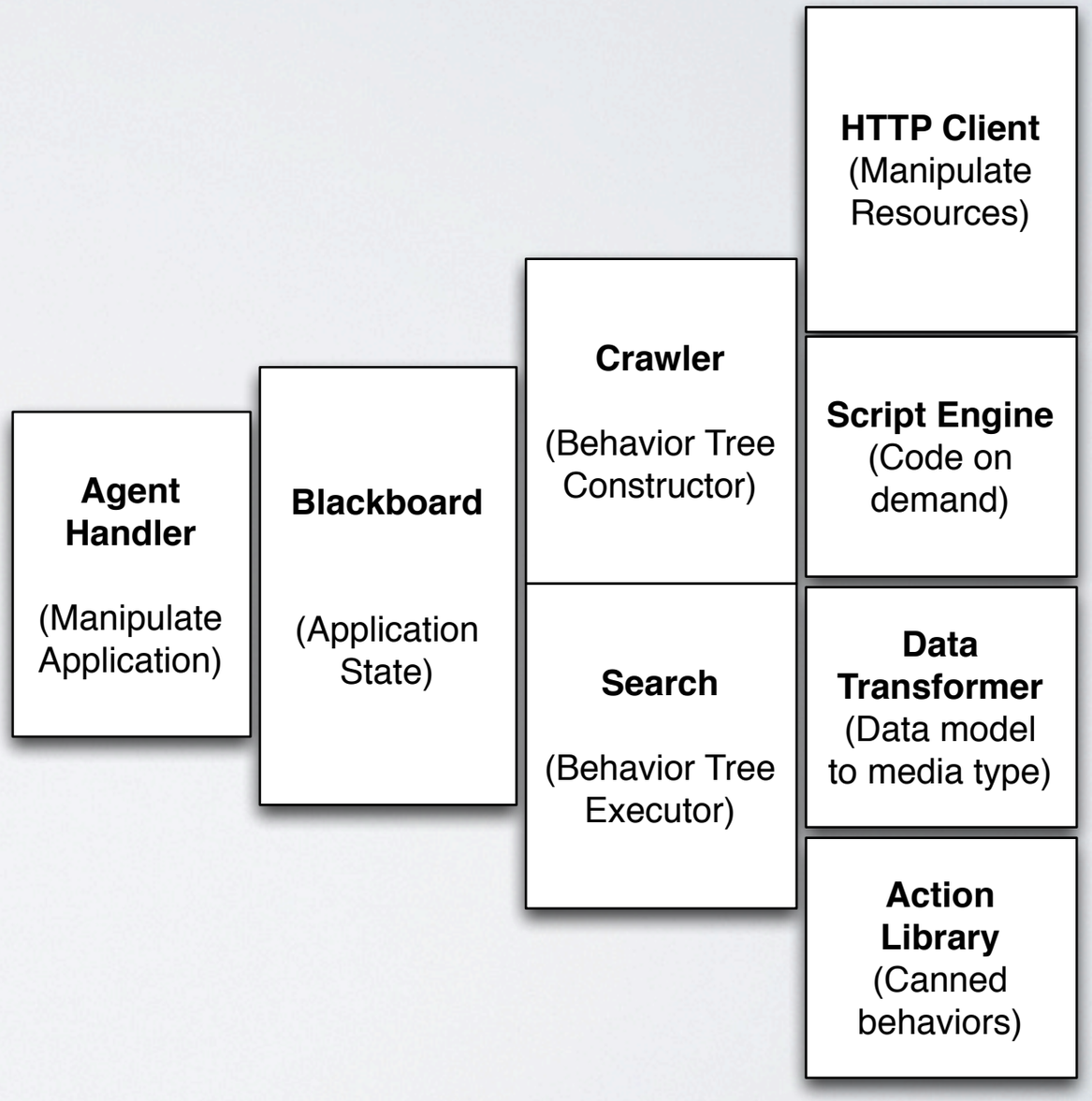
BEHAVIOR TREES ON THE WEB

- Specifying dynamic behavior as a search problem

- A behavior tree is effectively a **process flow** that's **assembled** by a **crawler** and executed by **depth first search**

- More complexity, potentially evolvability & less programming

AGENT DESIGN



AGENT DESIGN

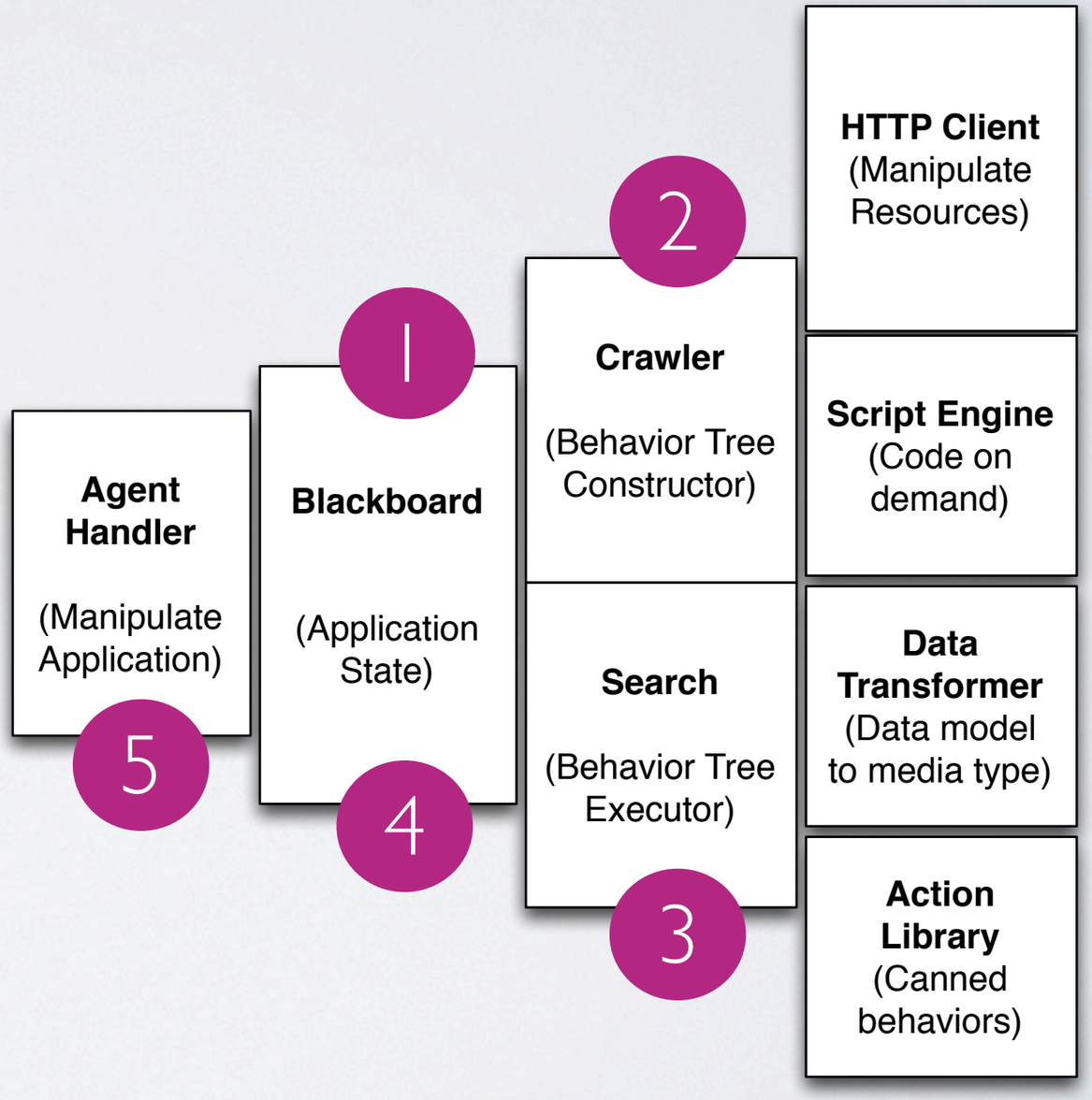
1. Start with your basic app information in the blackboard (personal info, credit card, etc.)

2. Crawl the behavior tree, starting with your goal tree

3. Execute the tree, picking actions, scripts, doing data transforms

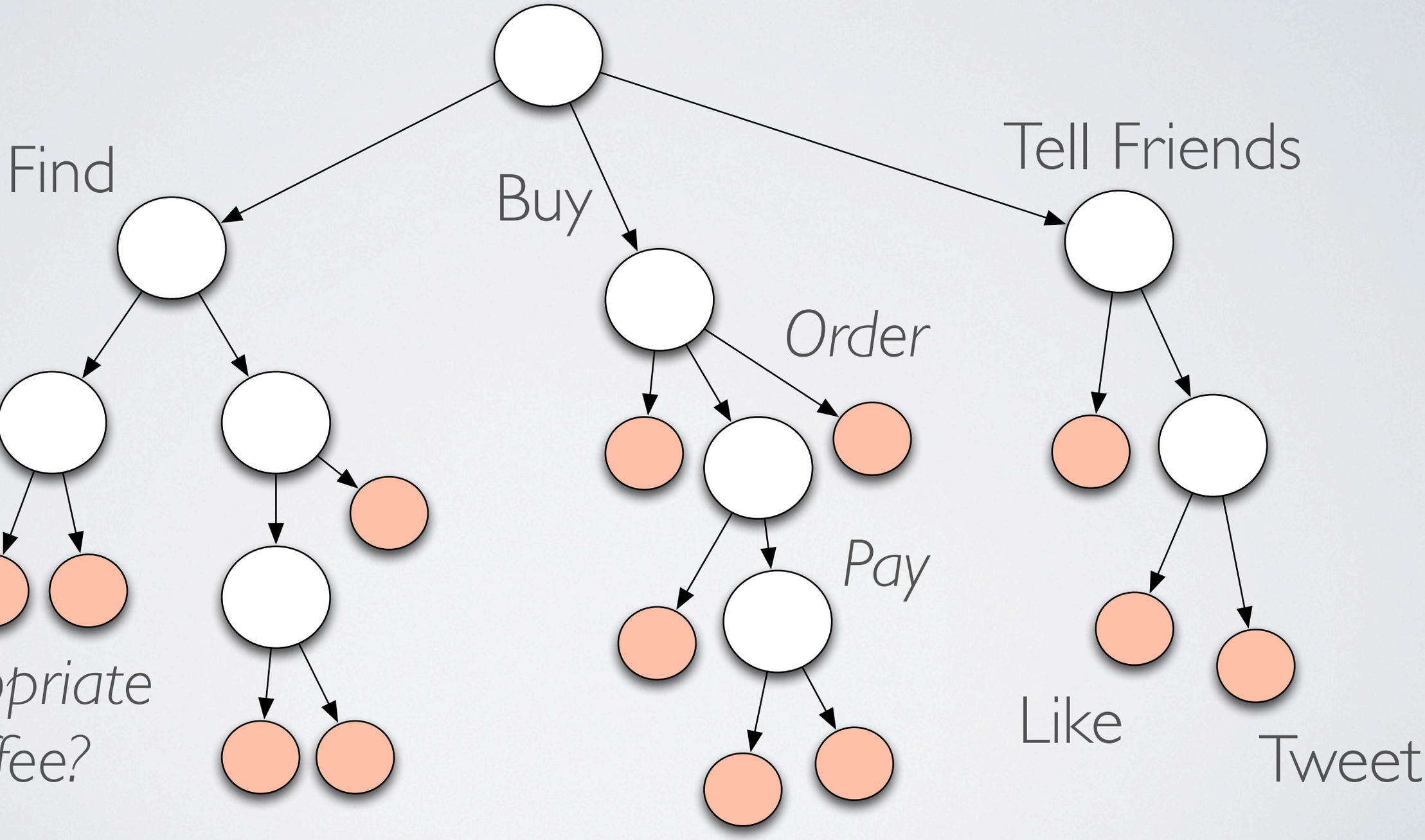
4. Update the blackboard with sensed state (via GETs)

5. Agent handler is notified when certain blackboard values change (i.e. as goal is reached or exceptions happen)

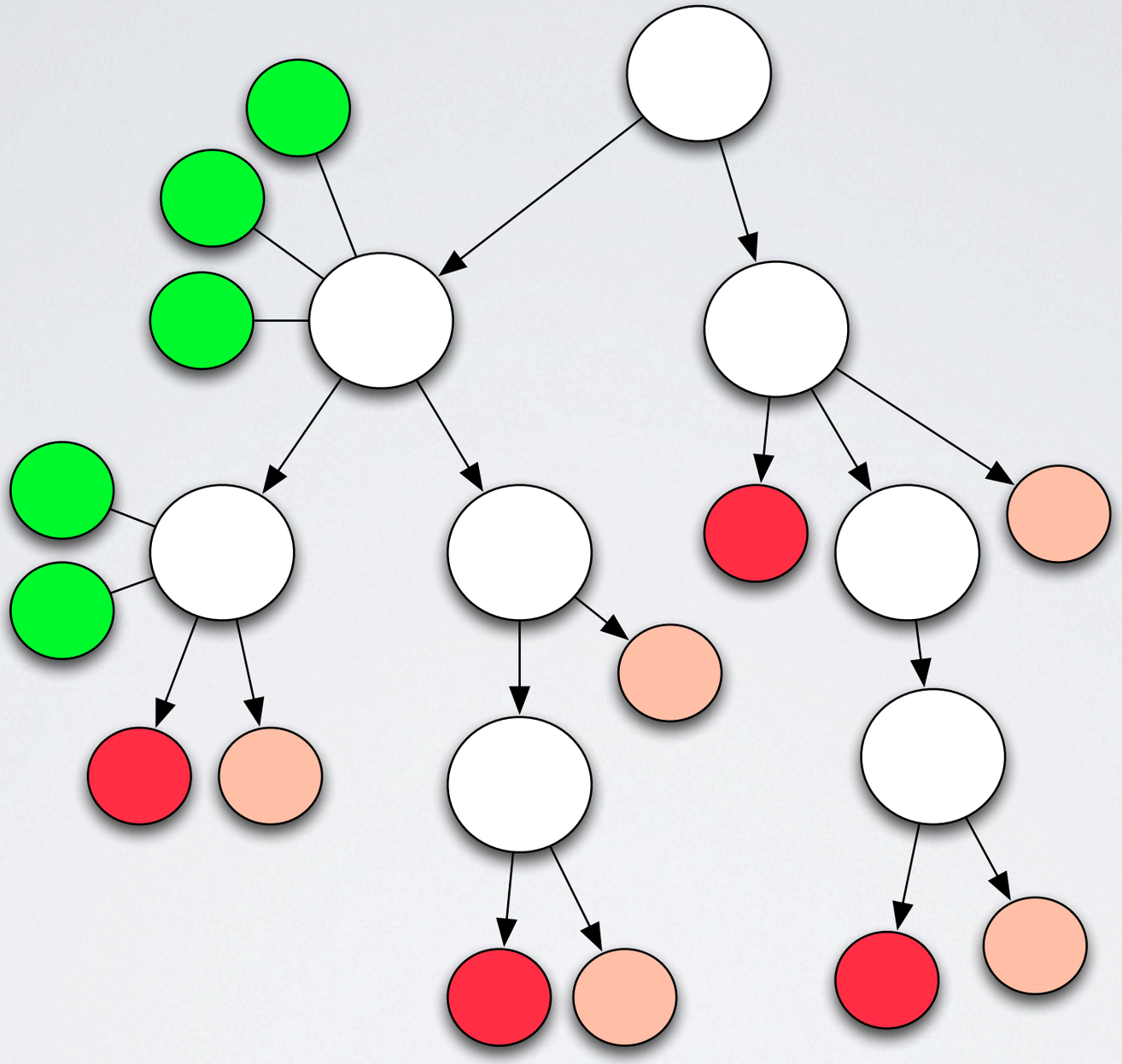


A BEHAVIOR TREE

Goal: Buy a coffee & tell my friends

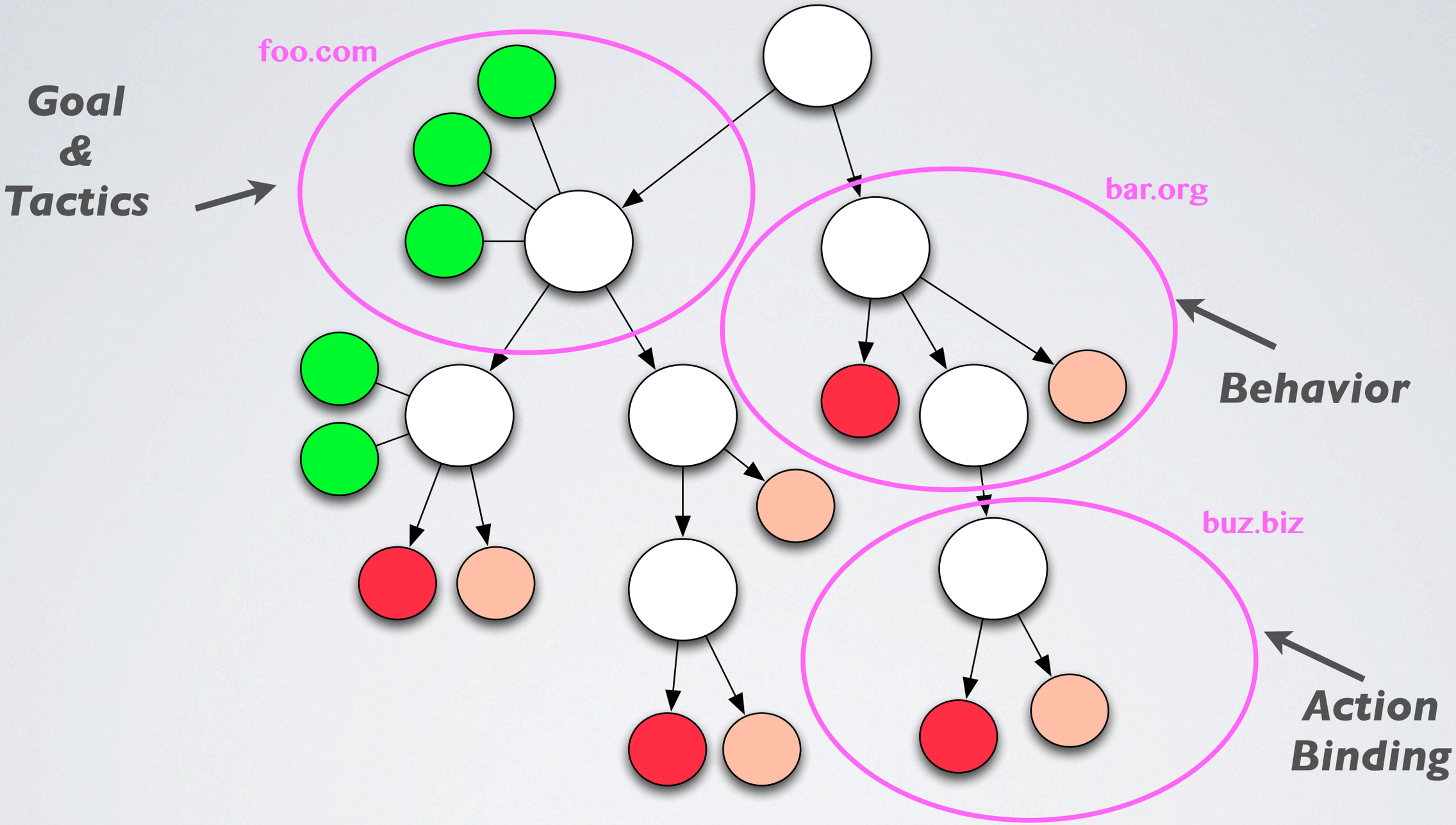


A BEHAVIOR TREE



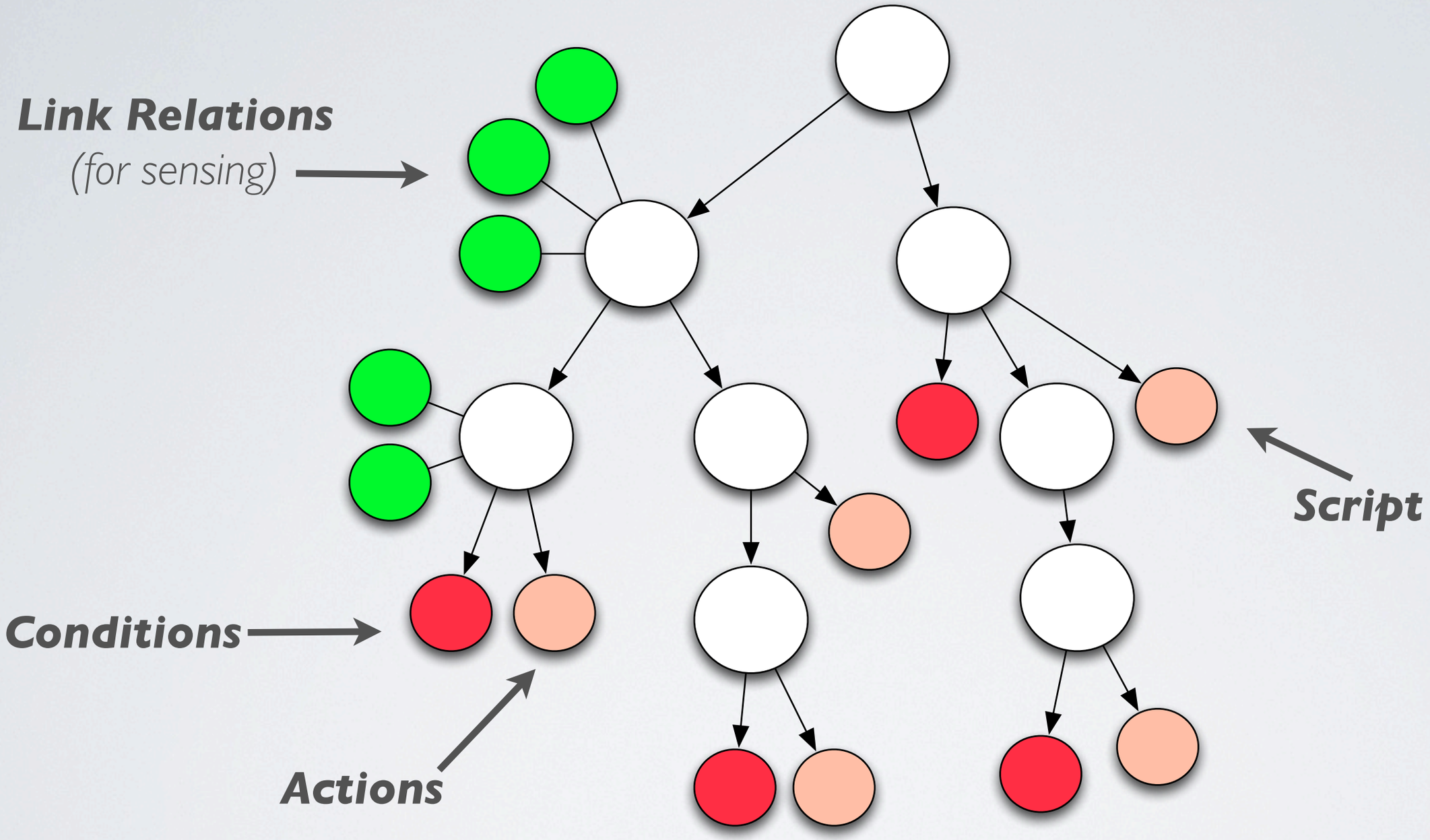
- Each node is a hyperlink (may be an #anchor within a doc)
- Leaves are tasks: the interface with the Web and/or your Agent

DISTRIBUTED & LINKED



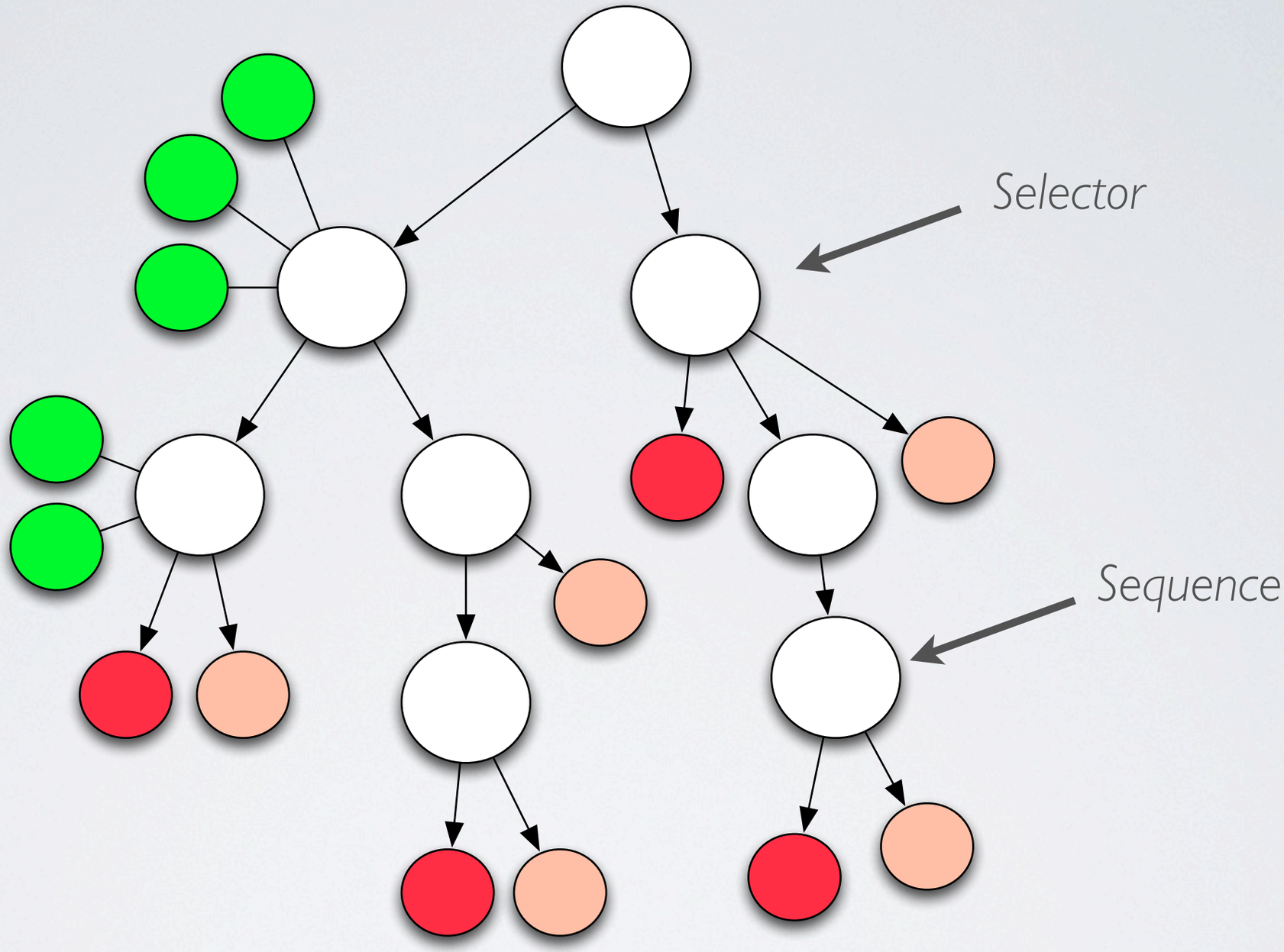
- Different domains & origin servers can describe pieces of the tree
- Goals lead to behaviors, which trickle down into concrete actions

ATOMIC TASKS



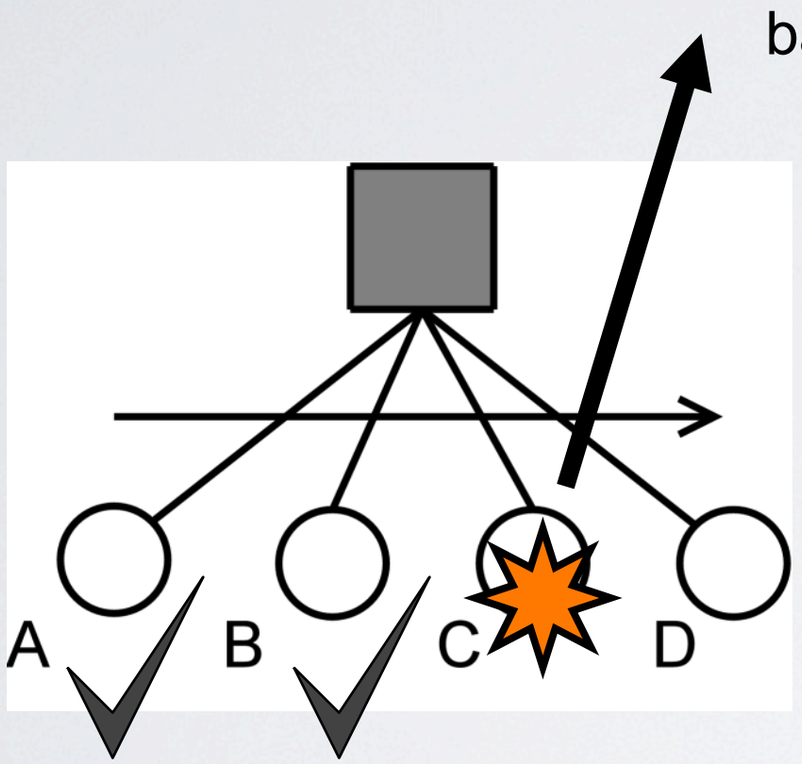
- **Relation** links are GETted where found to change application state
  Conditions sense your application state
  Actions update (POST, PUT, DELETE) resource state
  Scripts enable code on demand to change application/resource state opaquely

COMPOSITE TASKS

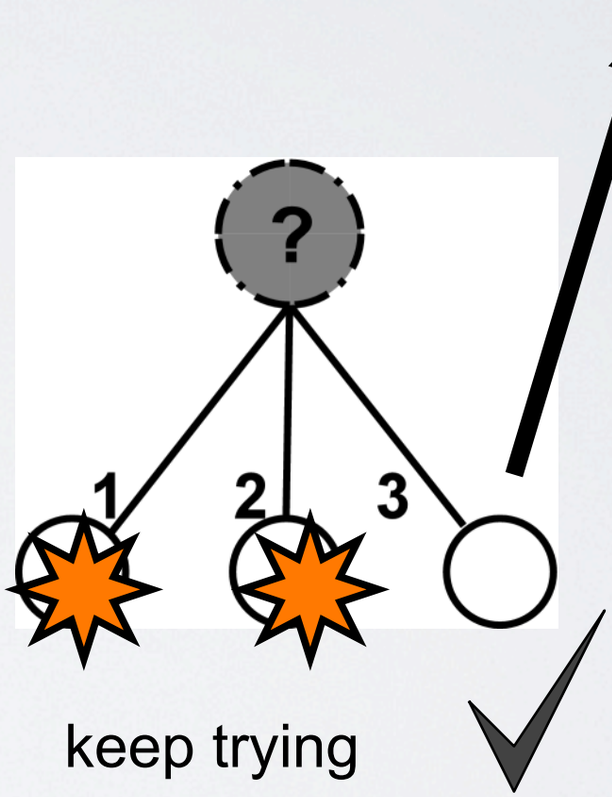


- **Branches** control complexity by enabling higher-level logic
  Examples: selectors, sequences, decorators, filters, parallel, etc.

SELECTOR VS. SEQUENCE



keep going

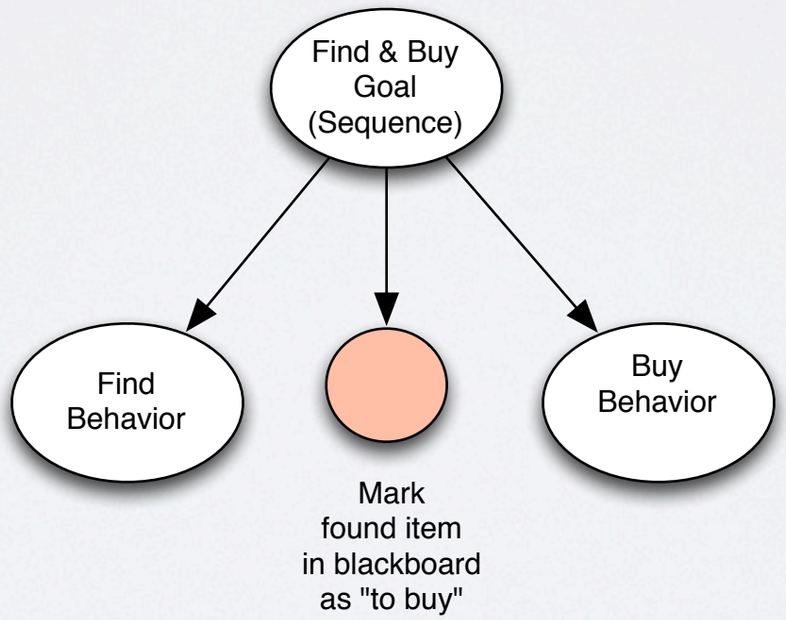


keep trying

EXAMPLE

Find & Buy a Coffee

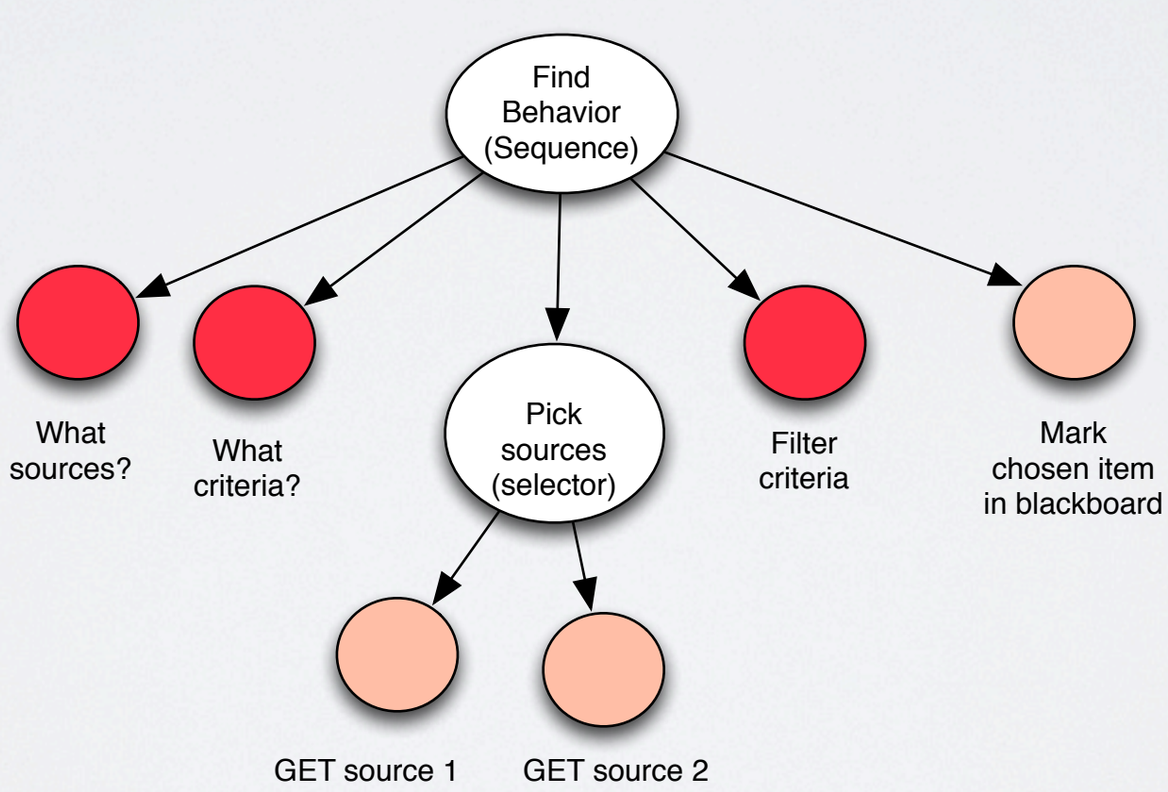
- Tree provided by me to my agent
- Find and Buy are effectively link relations that others can reference that they implement with their own trees



EXAMPLE

Find & Buy a Coffee

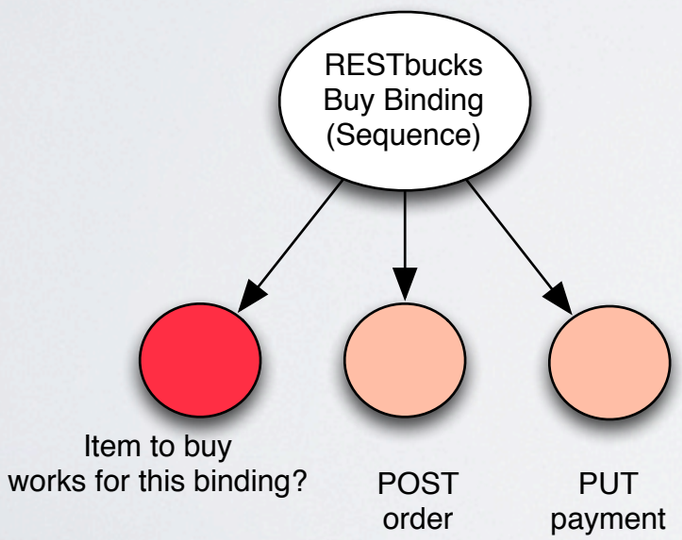
- Tree provided someone else for generic "find something in a document by a criteria"

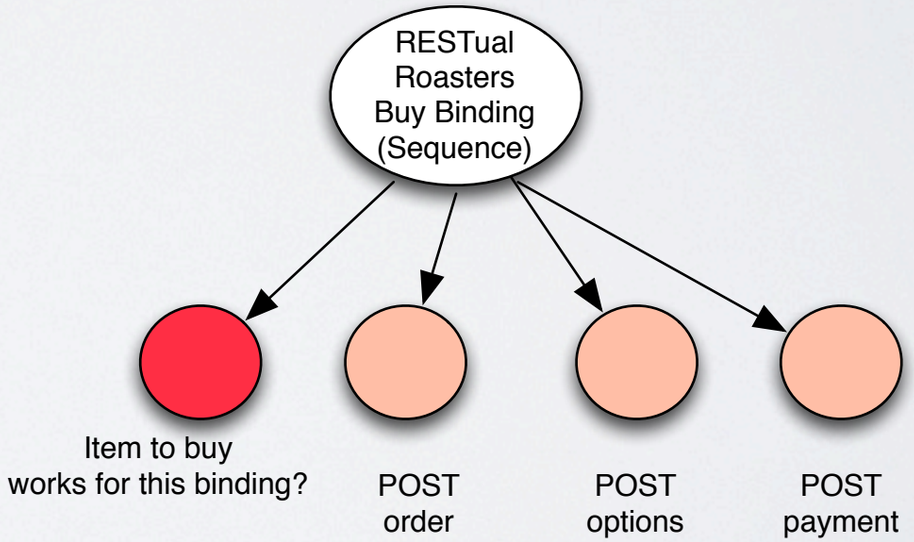


EXAMPLE

Find & Buy a Coffee

- Bindings to the "buy" link relation
- Provided by their respective origin servers (or some good samaritan)

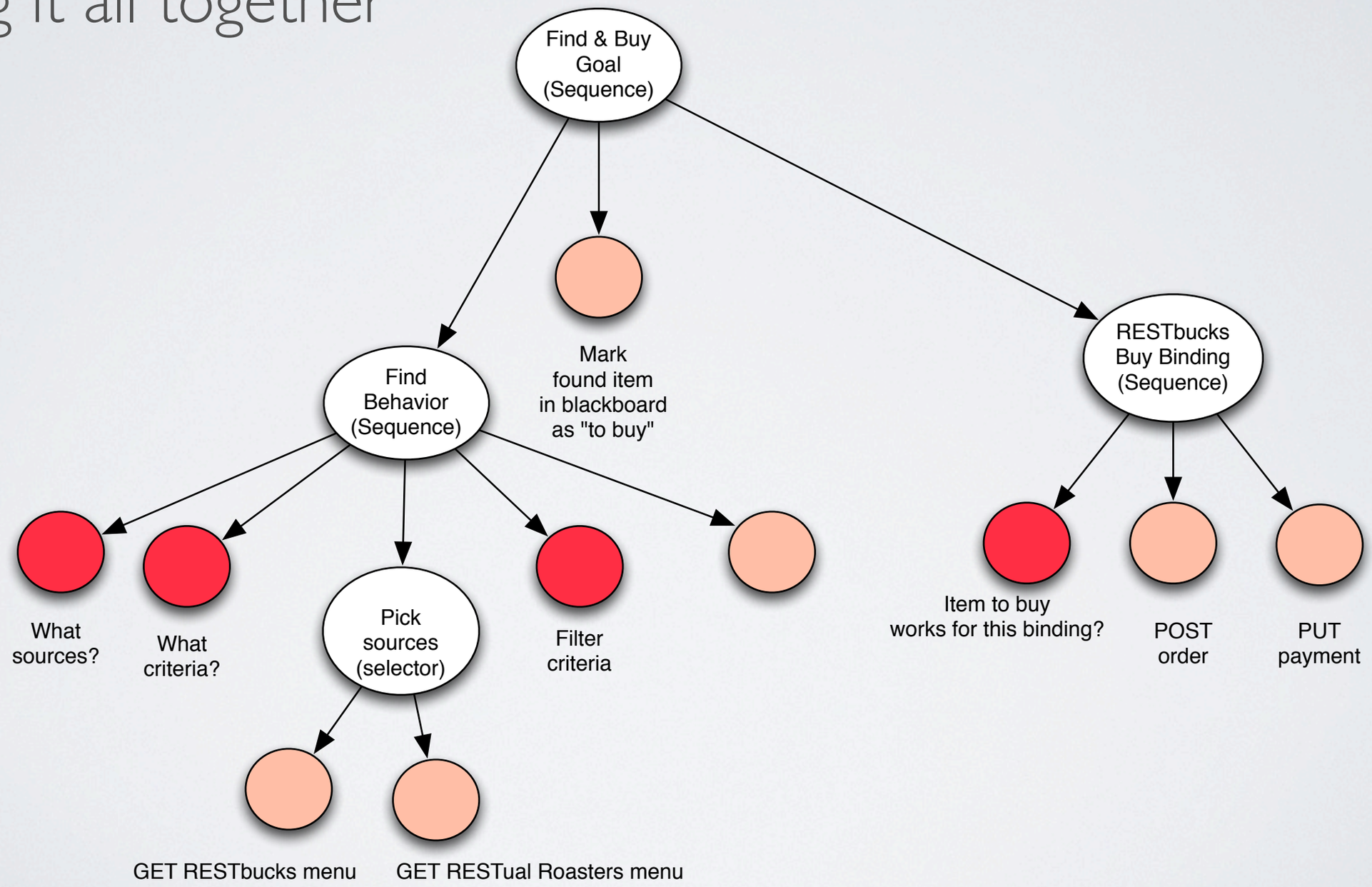




EXAMPLE

Find & Buy a Coffee

- Putting it all together



SUMMARY

- The Web architecture encourages clients to be designed as **agents** in a **dynamic** information space

- There are **practical** approaches to programming **agents** in a **dynamic** environment (blackboard, search, crawler)

- **Linked behavior trees** may be a **general purpose** media type for **systems** to manipulate state on the web, in lieu of more specific media types