CS354R

DR SARAH ABRAHAM

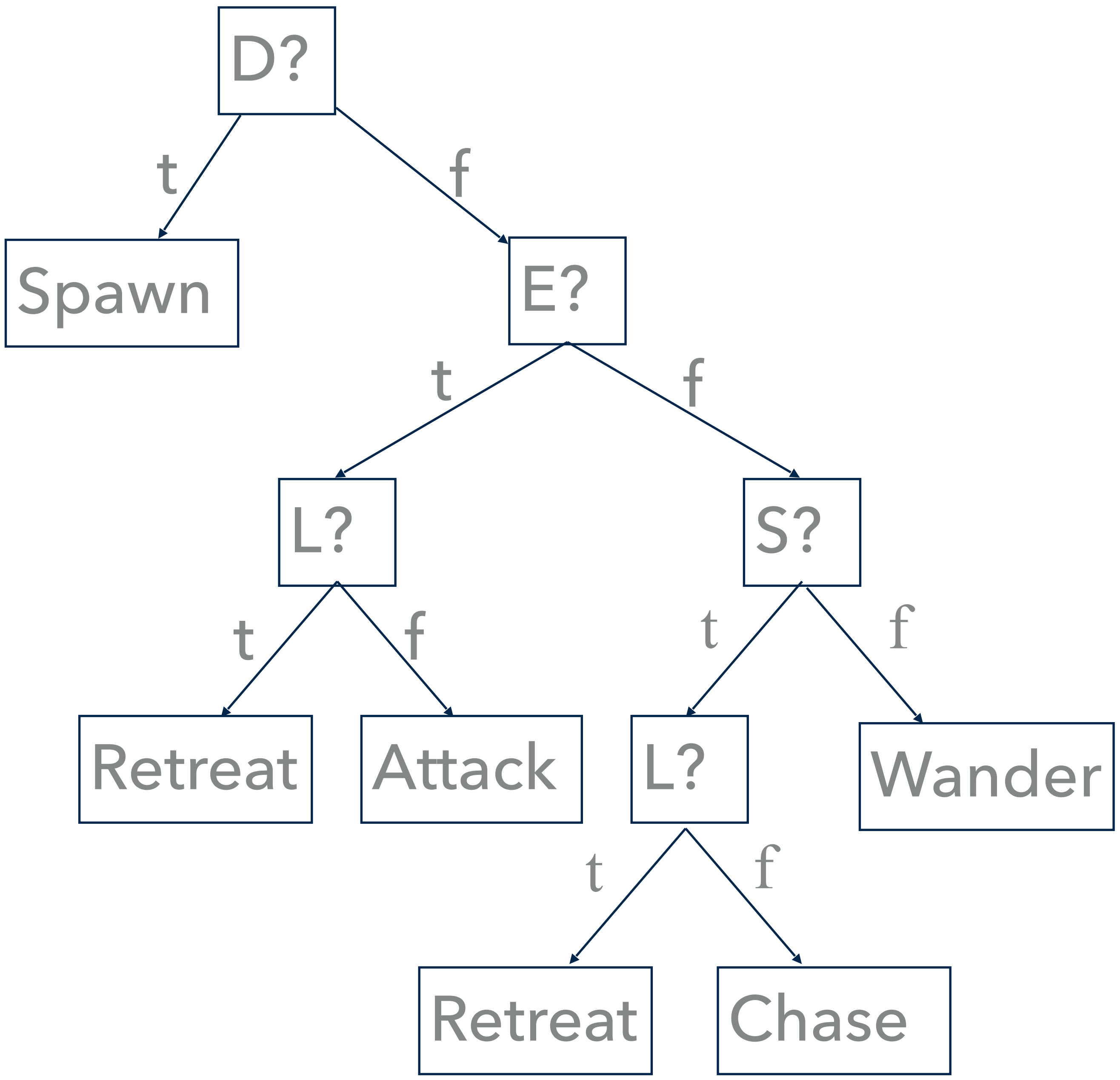
AI DECISION TREES AND RULE SYSTEMS

DECISION TREES

- Nodes represent attribute tests
  - One child for each outcome
- Leaves represent classifications
  - Can have same classification across leaves
- Classify by descending from root to a leaf
  - Perform test and descend
  - Return leaf's classification (action)
- Decision tree is a "disjunction of conjunctions of constraints on the attribute values of an instance"
  - Action if (A and B and C) or (A and ~B and D) or (…) …
  - Retreat if (low health and see enemy) or (low health and hear enemy) or (…) …

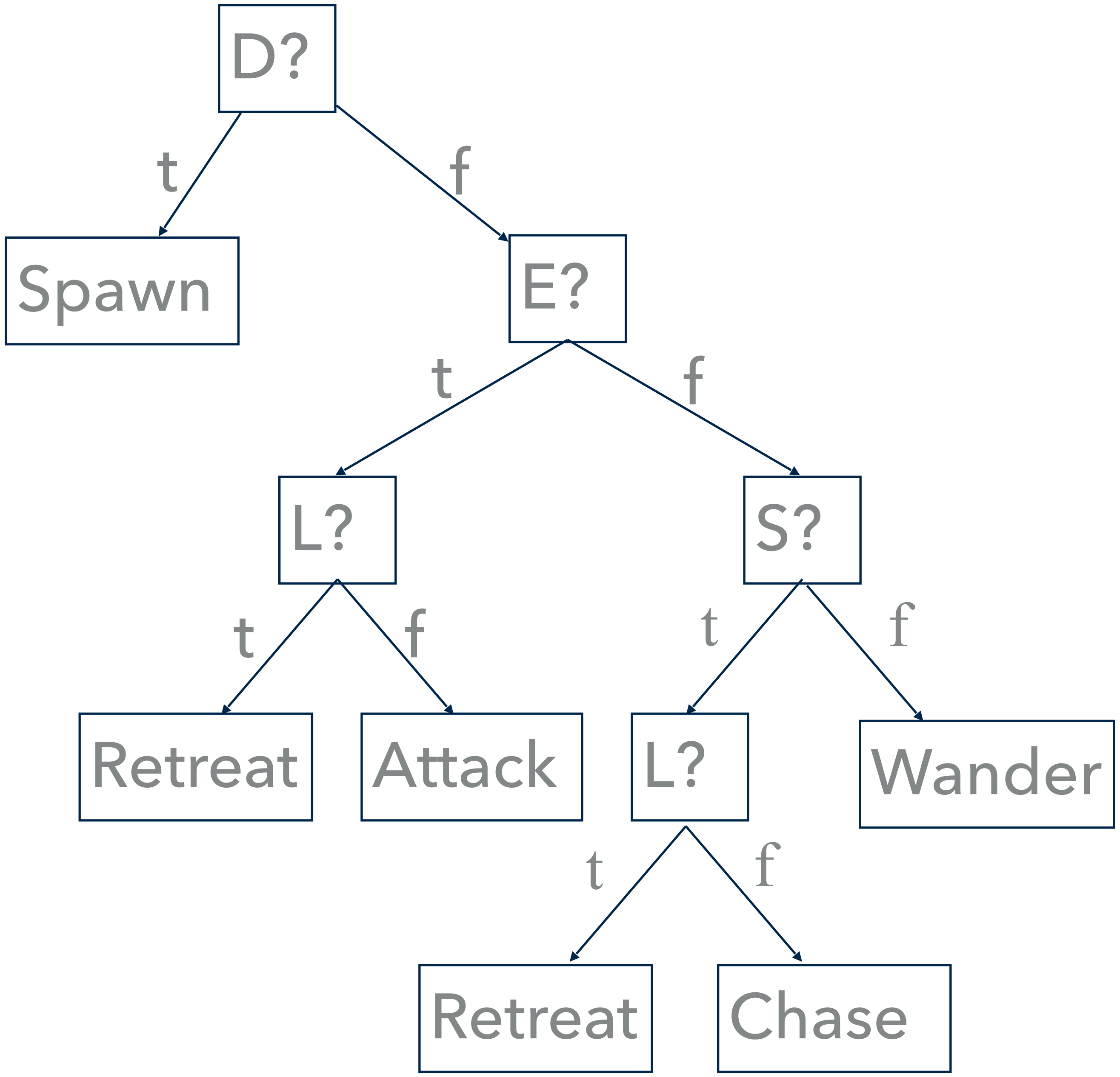
DECISION TREE FOR QUAKE

- Just one tree
- Attributes:
  Enemy=<t,f>
  Low=<t,f>
  Sound=<t,f>
  Death=<t,f>
- Actions: Attack, Retreat, Chase, Spawn, Wander

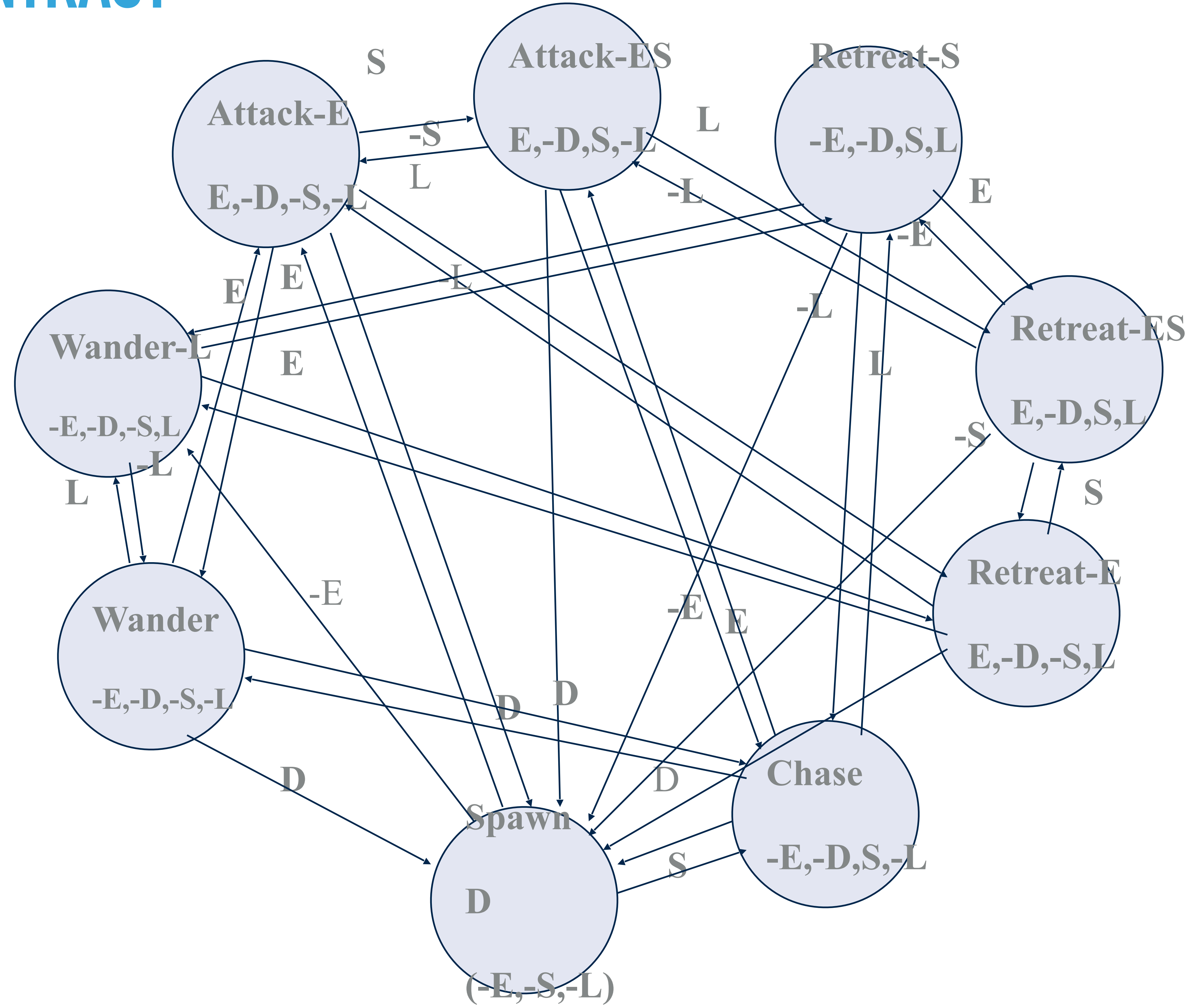


DECISION TREE FOR QUAKE

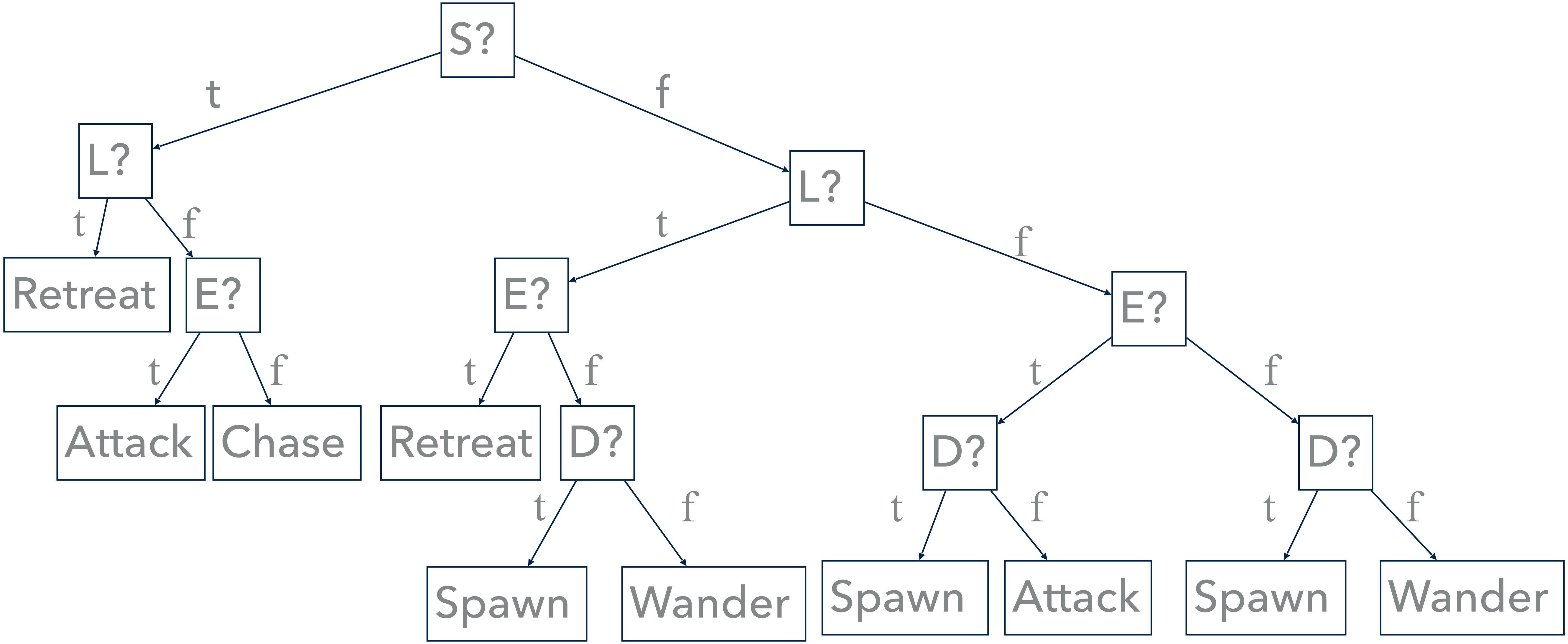
- Could add additional trees
  - If I'm attacking, which weapon should I use?
  - If I'm wandering, which way should I go?
  - Can be thought of as just extending given tree
  - Or, can share pieces of tree, such as a Retreat sub-tree



COMPARE AND CONTRAST



DIFFERENT TREES – SAME DECISION



HANDLING SIMULTANEOUS ACTIONS

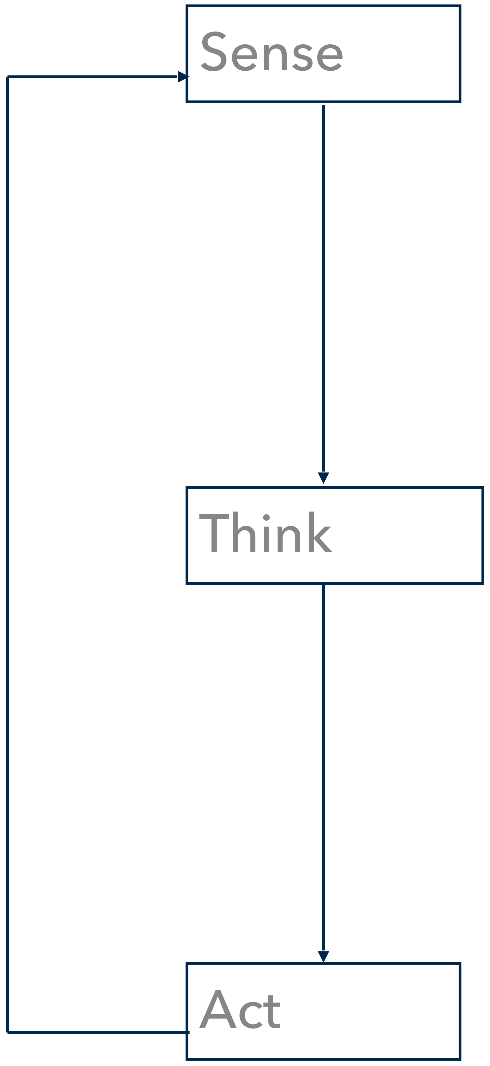
- Treat each output command as a separate classification problem
  - Given inputs should walk => <forward, backward, stop>
  - Given inputs should turn => <left, right, none>
  - Given inputs should run => <yes, no>
  - Given inputs should weapon => <blaster, shotgun…>
  - Given inputs should fire => <yes, no>
- Have a separate tree for each command
- If commands are not independent, two options:
  - Have a general conflict resolution strategy
  - Put dependent actions in one tree

DECIDING ON ACTIONS

- Each time the AI is called:
  - Poll each decision tree for current output
  - Event driven - only call when state changes
- Need current value of each input attribute
  - All sensor inputs describe the state of the world
- Store the state of the environment
  - Most recent values for all sensor inputs
  - Change state upon receipt of a message
  - Or, check validity when AI is updated
  - Or, a mix of both (polling and event driven)

SENSE, THINK, ACT CYCLE

- Sense
  - Gather input sensor changes
  - Update state with new values
- Think
  - Poll each decision tree
- Act
  - Execute any changes to actions



BUILDING DECISION TREES

- Decision trees can be constructed by hand
  - Think of the questions you would ask to decide what to do
  - For example: Tonight I can study, play games or sleep. How do I make my decision?
- But, decision trees in AI are typically *learned*:
  - Provide examples: many sets of attribute values and resulting actions
  - Algorithm then constructs a tree from the examples
  - Reasoning: We don’t know how to decide on an action, so let the computer do the work

LEARNING DECISION TREES

- Decision trees are usually learned by induction
  - Generalize from examples
  - Induction doesn’t guarantee correct decision trees
- Bias towards smaller decision trees
  - Occam’s Razor: Prefer simplest theory that fits the data
  - Too expensive to find the very smallest decision tree
- Learning is non-incremental
  - Need to store all the examples
- ID3 is the basic learning algorithm
  - C4.5 is an updated and extended version

INDUCTION

- If X is true in every example that results in action A, then X must always be true for action A
  - More examples are better
  - Errors in examples cause difficulty
    - If X is true in most examples X must always be true
    - D3 does a good job of handling errors (noise) in examples
  - Note that induction can result in errors
    - It may just be coincidence that X is true in all the examples
- Typical decision tree learning determines what tests are always true for each action
  - Assumes that if those things are true again, then the same action should result

LEARNING ALGORITHMS

- Recursive algorithms
  - Find an attribute test that separates the actions
  - Divide the examples based on the test
  - Recurse on the subsets
- What does it mean to separate?
- Separation:
  - Ideally, there are no actions that have examples in both sets
  - Failing that, most actions have most examples in one set
  - The thing to measure is entropy - the degree of homogeneity (or lack of it) in a set
    - Entropy is also important for compression

WHERE TO GET EXAMPLES?

- Generating examples:
  - Programmer/designer provides examples
  - Capture an expert player's actions, and the game state, while they play
- Number of examples needed depends on difficulty of concept
  - Difficulty: Number of tests needed to determine the action
  - More is always better
- Training set vs. Testing set
  - Train on most (75%) of the examples
  - Use the rest to validate the learned decision trees by estimating how well the tree does on examples it hasn't seen

DECISION TREE ADVANTAGES

- Simpler, more compact representation
- State is recorded in a memory
    - Create “internal sensors” - Enemy-Recently-Sensed
- Easy to create and understand
- Decision trees can be learned

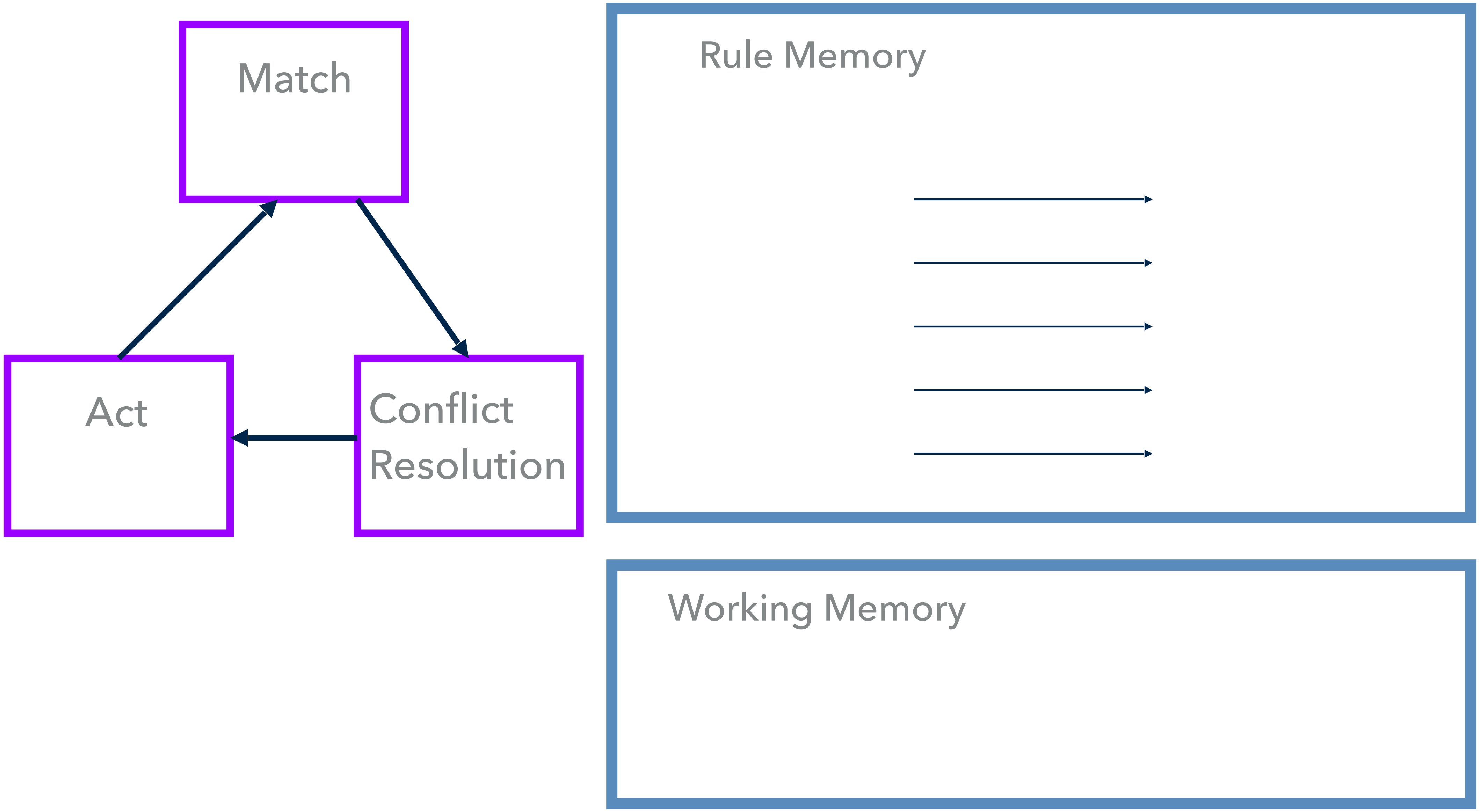
DECISION TREE DISADVANTAGES

- Decision tree engine requires more coding than FSM
- Need as many examples as possible
- Higher CPU cost (but not much higher)
- Learned decision trees may contain errors

RULE-BASED SYSTEMS

- Rule-based systems let you write the rules
  - Decision trees can be converted into rules
- System consists of:
  - A rule set - the rules to evaluate
  - A working memory - stores state
  - A matching scheme - decides which rules are applicable
  - A conflict resolution scheme - if more than one rule is applicable, decides how to proceed
- What types of games make the most extensive use of rules?

RULE-BASED SYSTEMS STRUCTURE



Match

Conflict Resolution

Act

Rule Memory

Working Memory

Program

Procedural Knowledge

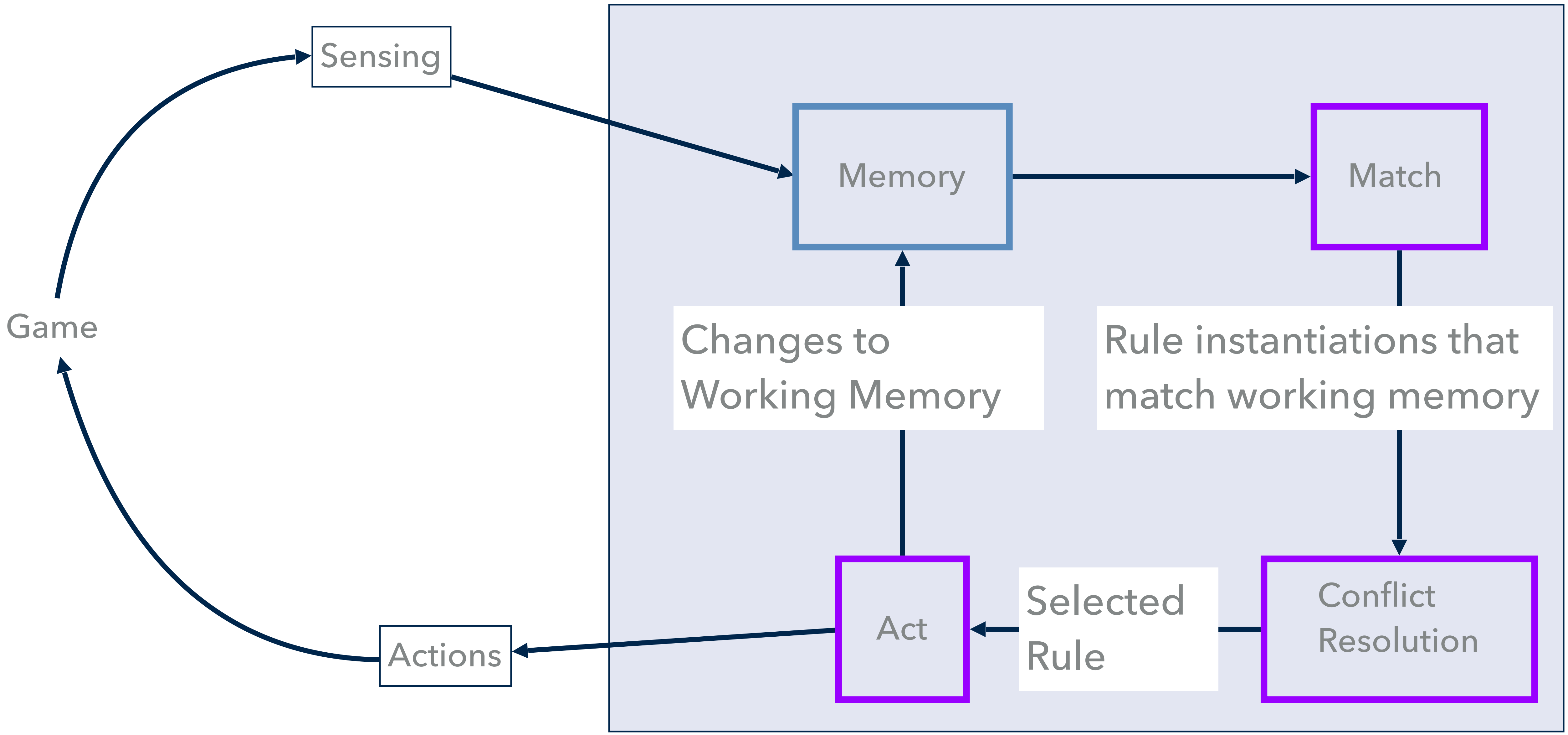
Long-term Knowledge

Data

Declarative Knowledge

Short-term Knowledge

AI CYCLE


Sensing
Memory
Match
Game
Changes to Working Memory
Rule instantiations that match working memory
Act
Selected Rule
Conflict Resolution
Actions

AGE OF KINGS

```
; The AI will attack once at 1100 seconds and then again
; every 1400 sec, provided it has enough defense soldiers.

(defrule
      (game-time > 1100)
=>
      (attack-now)
      (enable-timer 7 1400))

(defrule
      (timer-triggered 7)
      (defend-soldier-count >= 12)
=>
      (attack-now)
      (disable-timer 7)
      (enable-timer 7 1400))
```

Rule

Action

AGE OF KINGS

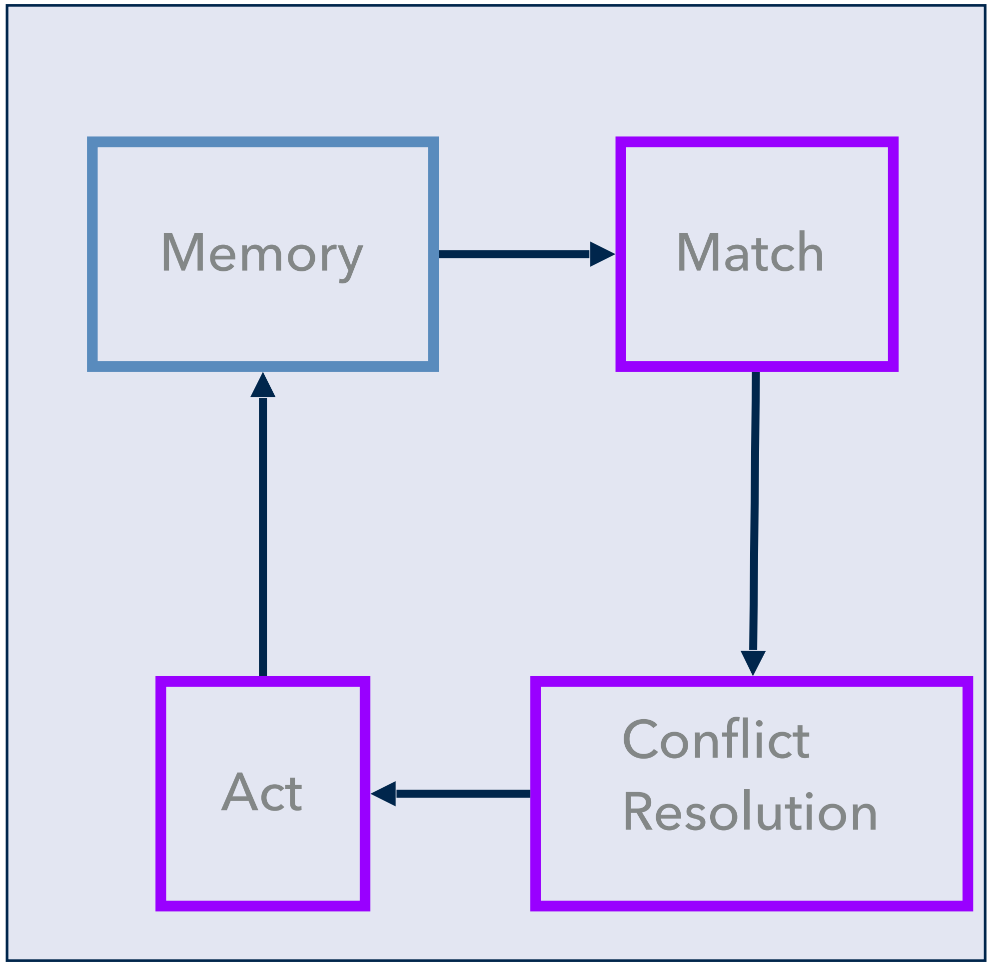
- What is it doing?

```
(defrule
	(true)
=>
	(enable-timer 4 3600)
	(disable-self))

(defrule
	(timer-triggered 4)
=>
	(cc-add-resource food 700)
	(cc-add-resource wood 700)
	(cc-add-resource gold 700)
	(disable-timer 4)
	(enable-timer 4 2700)
	(disable-self))
```

IMPLEMENTING RULE-BASED SYSTEMS

- Where does the time go?
- 90-95% goes to Match
  - Matching all rules against all of working memory each cycle is way too slow
- Key observation
  - # of changes to working memory each cycle is small
  - If conditions, and hence rules, can be associated with changes, then we can make things fast (event-driven)



GENERAL CASE

- Rules can be arbitrarily complex
  - In particular: function calls in conditions and actions
- If we have arbitrary function calls in conditions:
  - Run through rules one at a time and test conditions
  - Pick the first one that matches (or do something else)
- Time to match depends on:
  - Number of rules
  - Complexity of conditions
  - Number of rules that don't match

RESOLVING MULTIPLE MATCHES?

- Rule order – pick the first rule that matches
  - Makes order of loading important – not good for big systems
- Rule specificity - pick the most specific rule
- Rule importance – pick rule with highest priority
  - When a rule is defined, give it a priority number
  - Forces a total order on the rules – is right 80% of the time
  - Decide Rule 4 [80] is better than Rule 7 [70]
  - Decide Rule 6 [85] is better than Rule 5 [75]
  - Enforces ordering between all of them

REDUCING COST OF MATCHING

- Save intermediate match information (RETE)
  - Memory intensive
  - Fast search
  - DAGs that represent high-level rule sets
  - Tuples of facts matched against hierarchy of rules
  - Relevant facts asserted in working memory
- Recompute match for rules affected by change (TREAT)
  - Memory efficient
  - May be faster than RETE
- Make extensive use of hashing (mapping between memory and tests/rules)

RULE-BASED SYSTEM: ADVANTAGES

- Corresponds to way people often think of knowledge
- Very expressive
- Modular knowledge
  - Easier to write and debug compared to decision trees
  - More concise than FSMs

RULE-BASED SYSTEM: DISADVANTAGES

- Can be memory intensive
- Can be computationally intensive
- Can be difficult to debug

FURTHER READING

- RETE:
  - Forgy, C. L. Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence, 19(1) 1982, pp. 17-37
- TREAT:
  - Miranker, D. TREAT: A new and efficient match algorithm for AI production systems. Pittman/Morgan Kaufman, 1989