The Engineer’s Guide to Soar

Course 01: Soar Essentials

A Project-Based Tutorial Series

By Dr. Bryan Stearns, 2024

Course Design

This course has one main objective:

- To equip you to be capable of programming and maintaining a Soar application

In this course, you will build a basic *Supplier Sorting* agent

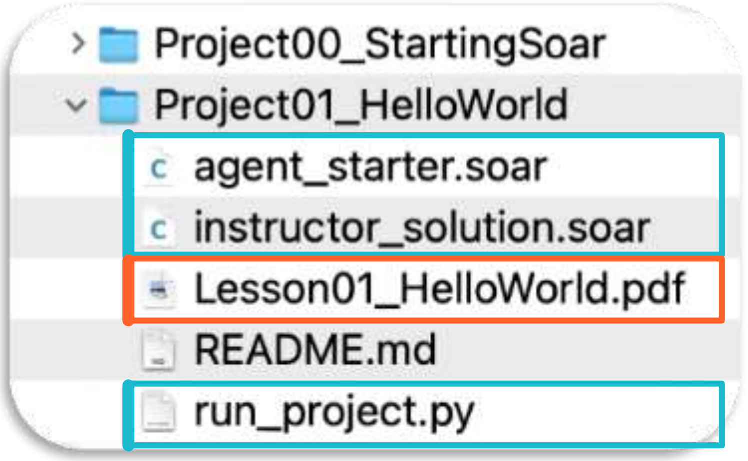
- See the *ProblemOverview.pdf* document for a description of the problem
- Each lesson in this course builds on the previous lesson to gradually build up the agent from scratch into production-ready code.

After completing this course, you will be able to build and maintain Soar agents on your own!

This course only covers the basics.

- It won't teach advanced capabilities such as learning (Chunking, RL, SMem, EpMem, etc.)
- But once you are comfortable with Soar basics, check the Soar Manual to learn more!

Lesson Design


Project00_StartingSoar
Project01_HelloWorld
agent_starter.soar
instructor_solution.soar
Lesson01_HelloWorld.pdf
README.md
run_project.py

Each lesson has two components:

1. **Instruction**: A pdf that describes the concepts and goals for the lesson
   - (You are here)
2. **Application**: A project assignment that applies the concepts to develop your agent
   - Each project folder comes with some starter Soar code that you must modify.
   - Walk through the pdf and edit the code as you go according to its instructions.
     - This is a self-guided learning and development experience.
     - By the end of the lesson pdf, you should have completed the assignment.
   - Each folder also comes with instructor solution code for the assignment.
     - (Don’t look at the solution unless you are stuck after watching the video.)
   - The provided “`run_project.py`” Python script lets you test your code at any time.

Lesson 00 – Outline

This first lesson answers two questions:

1. What is Soar?
2. How do I install and start Soar on my machine?

What is the Soar Cognitive Architecture?

A High-level Overview

What is Soar?

For the *academic*, Soar is an implemented *theory of cognition*

- Used for experiments that help us test our understanding of human intelligence

For the *engineer*, Soar is a *software engine*

- Used for making and running Artificial Intelligence (AI) agents.

Soar is both.

- Soar is a **Cognitive Architecture**

What is an Architecture?

An **architecture** defines how different components connect to provide an emergent capability….

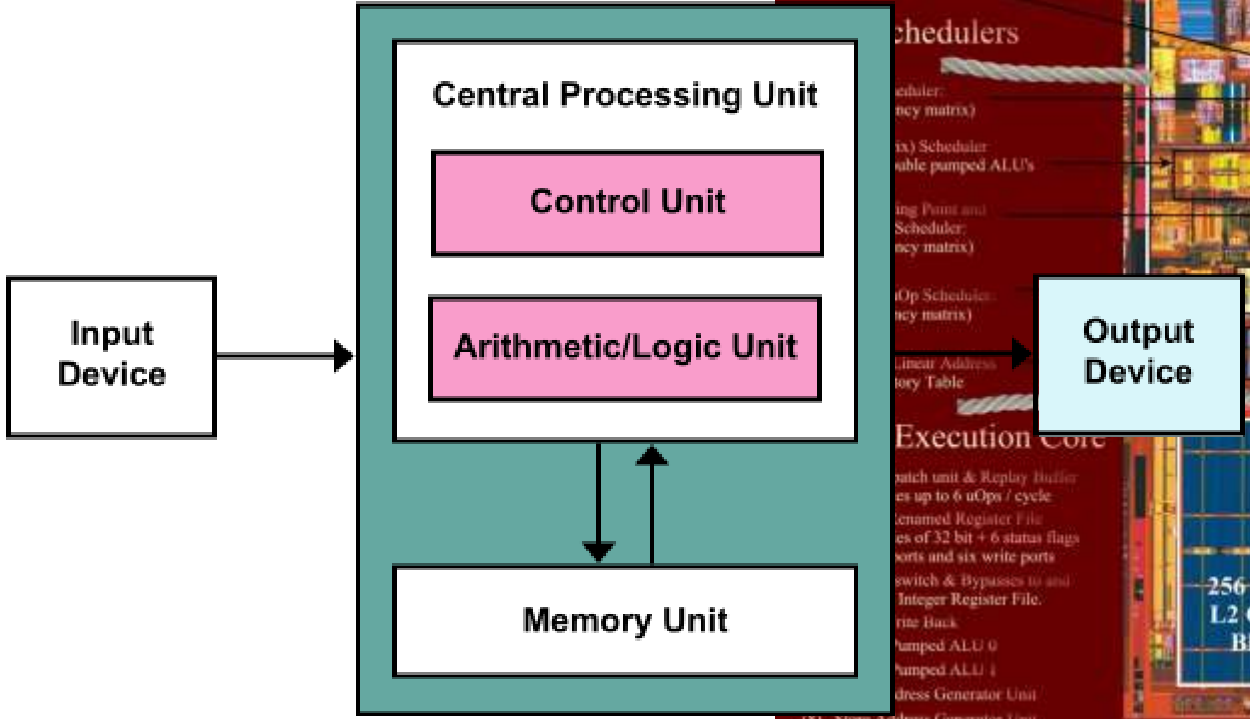
Something greater than the sum of its parts.

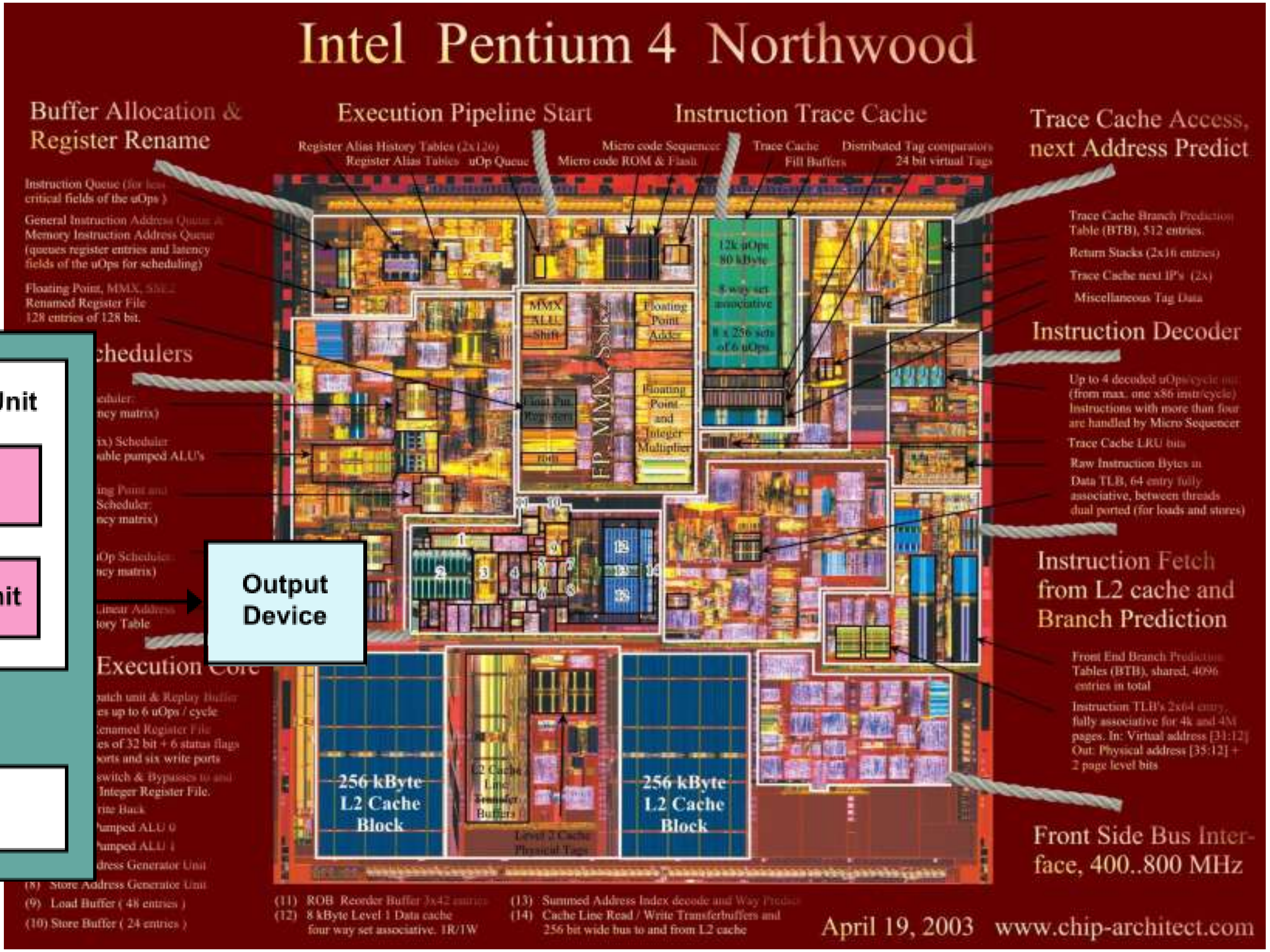
No one component is sufficient.

Removing any one component usually breaks the system.

Computer Architecture

A computer uses a complex architecture to support general computation….





Human Cognitive Architecture

The human brain has an architecture

Human general processing *emerges* from the sum of all parts working together

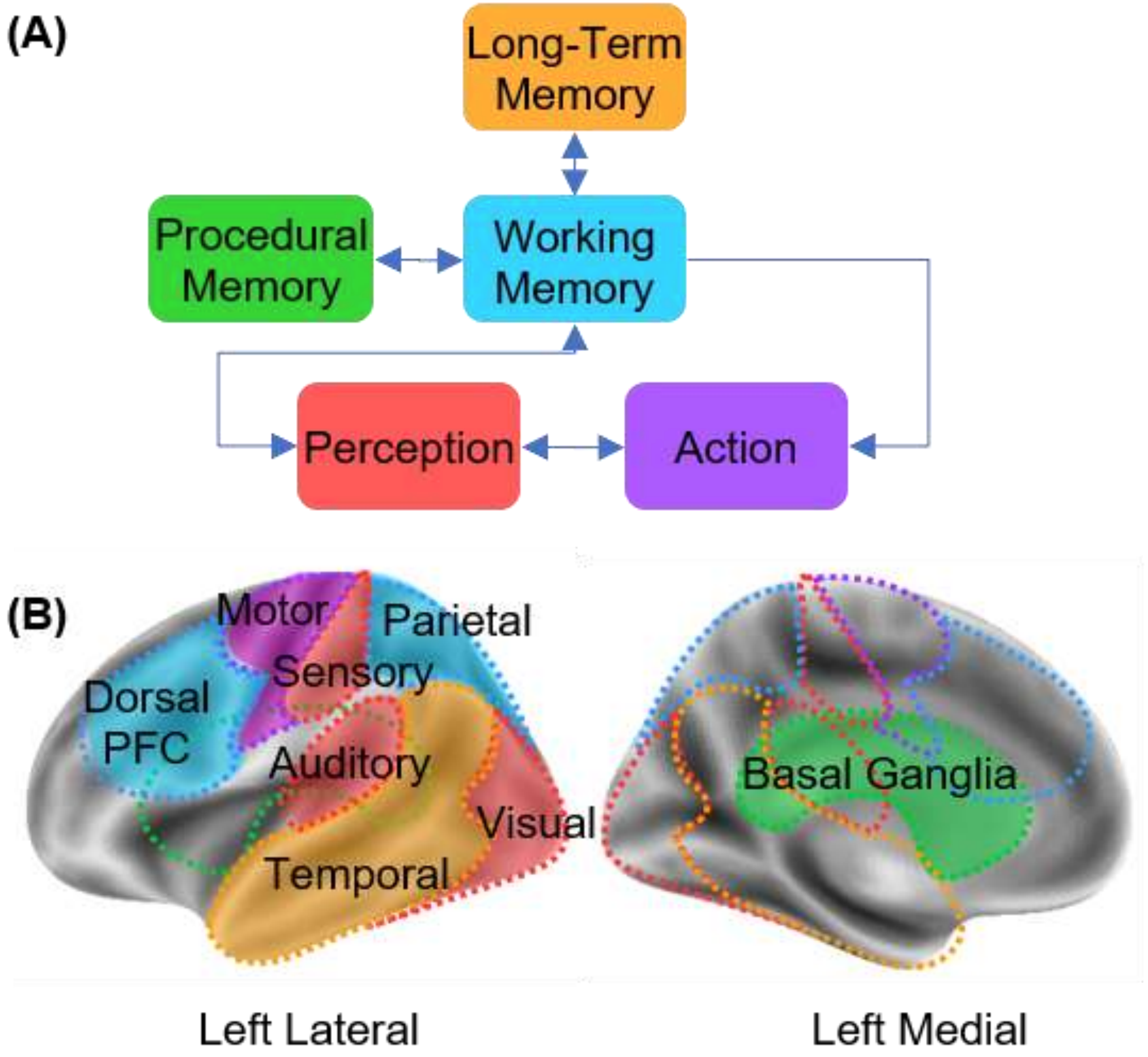
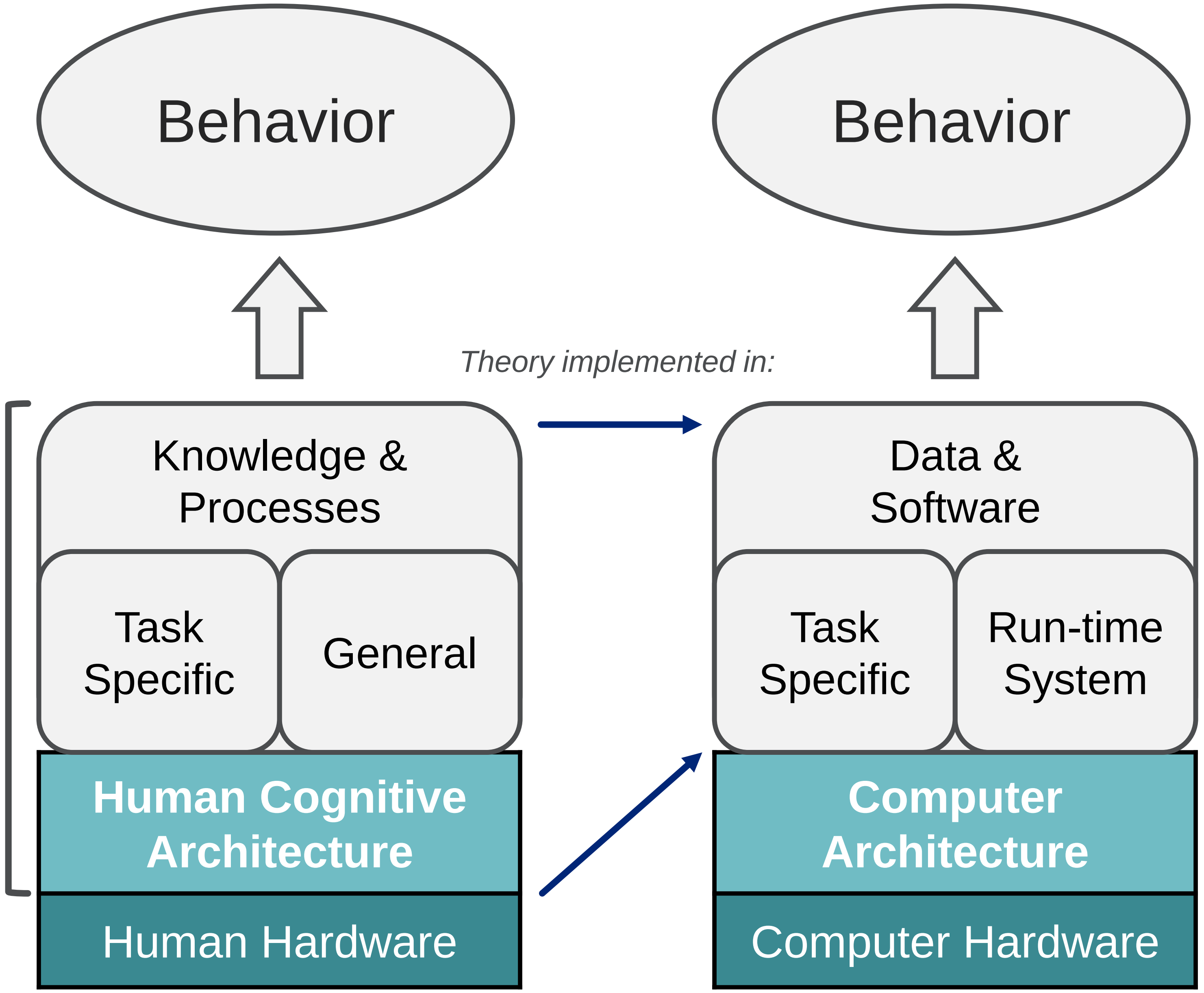


Image source: Wapstra et al. (2022). Increased Basal Ganglia Modulatory Effective Connectivity Observed in Resting-State fMRI in Individuals With Parkinson’s Disease. *Frontiers in Aging Neuroscience*. 14. 719089. 10.3389/fnagi.2022.719089.

Artificial Cognitive Architecture

Cognitive Architectures are:

- **Computational theories** about architectures that supply human-like cognition
- **Software engines** that implement those theories



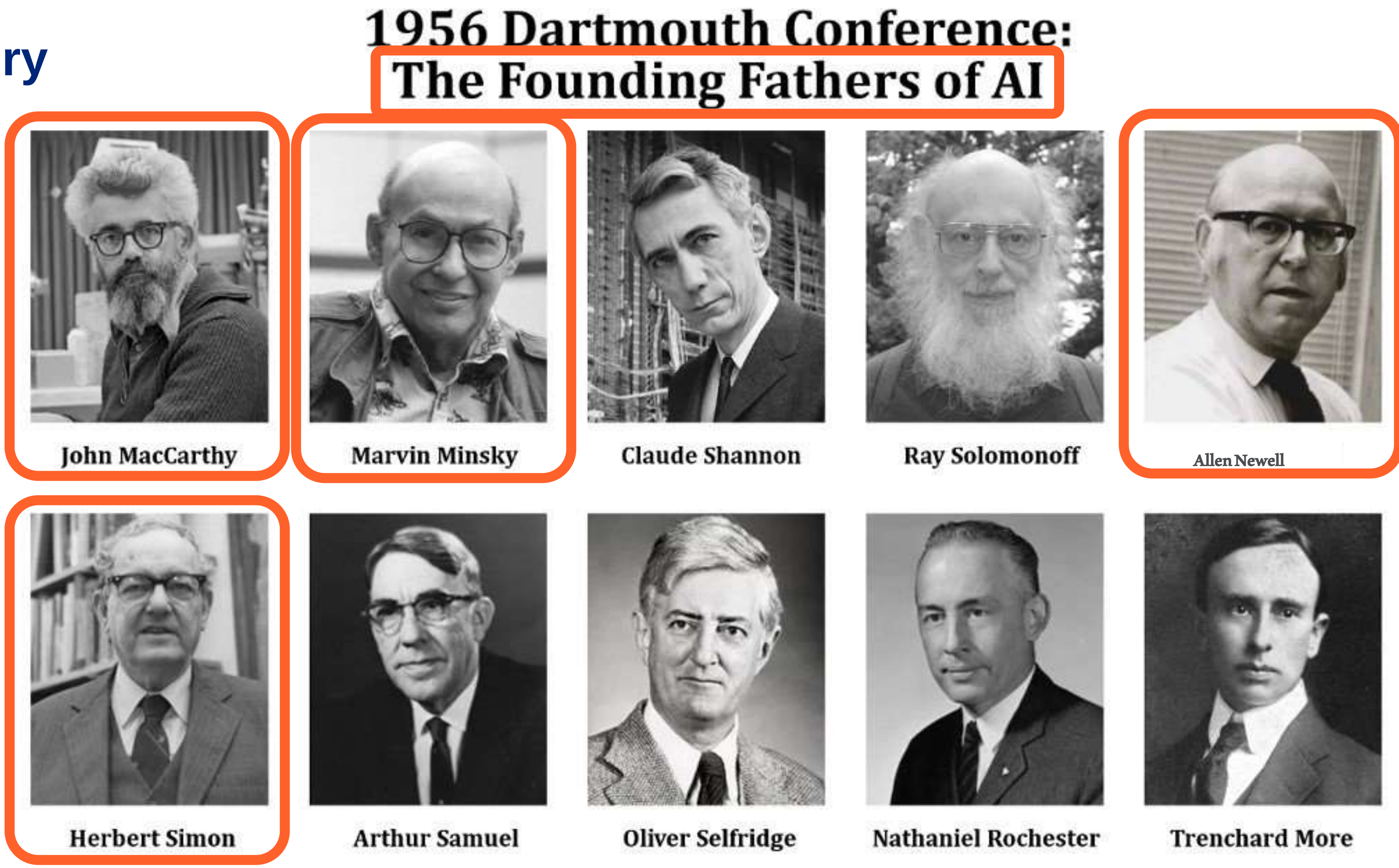
Cognitive Architecture History

Rooted in the birth of AI

- Pursued by Allen Newell with Herbert Simon
- Grew from work with the *Logic Theorist* presented at Dartmouth Conference

Field founded by **Allen Newell**

- To improve theories of human cognition by implementing them in software
- To improve software by learning from human cognition



Many Cognitive Architectures Today

Each focuses on different aspects of human processing

Most are for academic experimentation in psychology or cognitive science

- (Not built for AI or business)

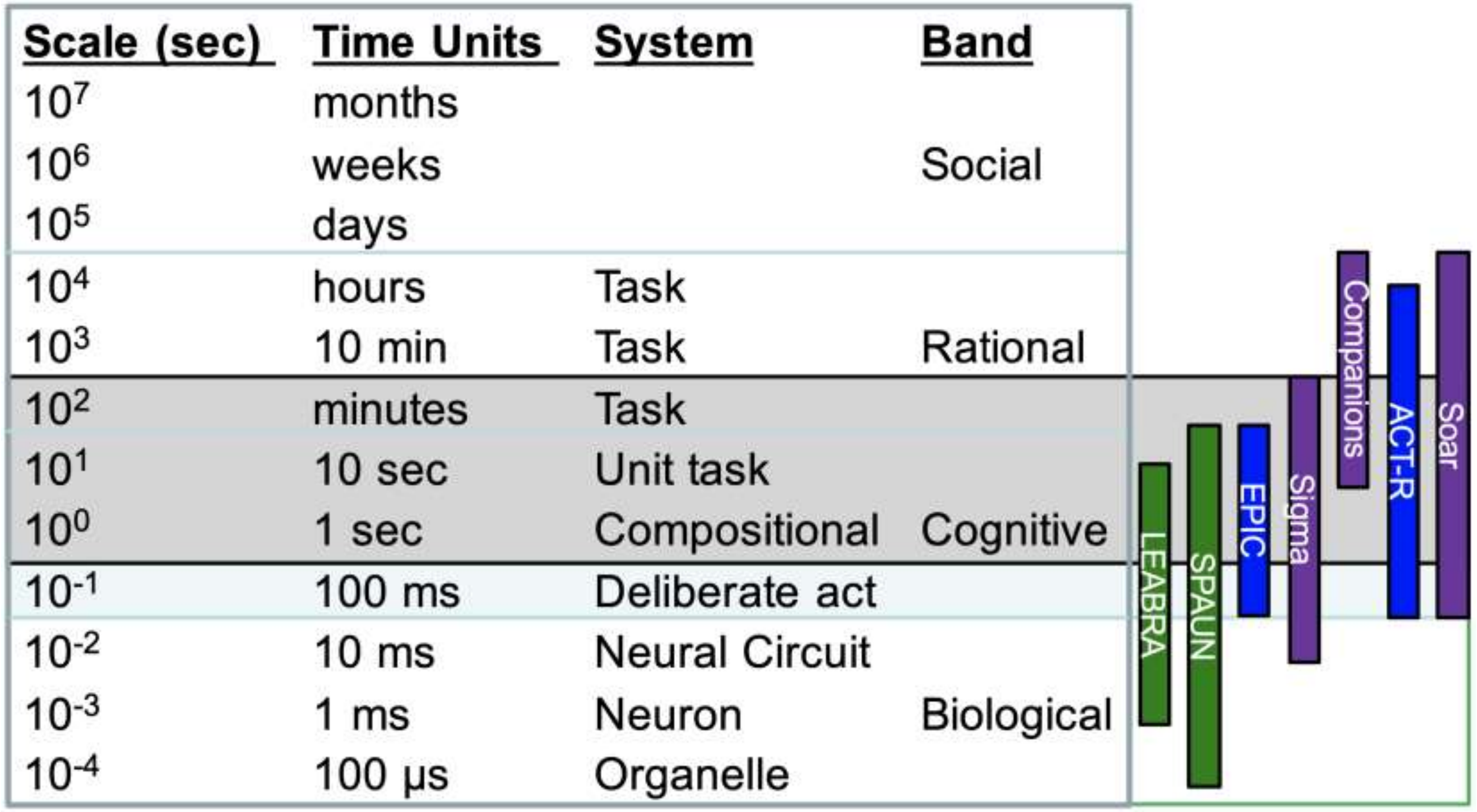
MicroPsi
SNePS
4CAPS
CogPrime
Global Workspace Theory
ICARUS
PRS
HTM
ERE
ACT-R
SAL
AIS
IBCA
OpenCog
Shruti
GLAIR
ART
Polyscheme
Soar
PRODIGY
NARS
DUAL
LIDA
EPIC
CHREST
NOMAD
DeSTIN
ADAPT
CLARION
SAL
Copycat/Metacat
4D/RCS

Popular cognitive architectures today

What is Soar's Focus?

- Built for practical AI that is *inspired* by humans
- Design elements:
  - Hierarchical context-based decision making
  - End-to-end interactivity
  - Composing complex behaviors from common elements
  - Unified symbolic knowledge representations across architecture components
  - 50 msec decision cycle

Newell's Time Scale of Human Action



Scale (sec)	Time Units	System	Band
10^{7}	months		
10^{6}	weeks		Social
10^{5}	days		
10^{4}	hours	Task	
10^{3}	10 min	Task	Rational
10^{2}	minutes	Task	
10^{1}	10 sec	Unit task	
10^{0}	1 sec	Compositional	Cognitive
10^{-1}	100 ms	Deliberate act	
10^{-2}	10 ms	Neural Circuit	
10^{-3}	1 ms	Neuron	Biological
10^{-4}	100 μs	Organelle	

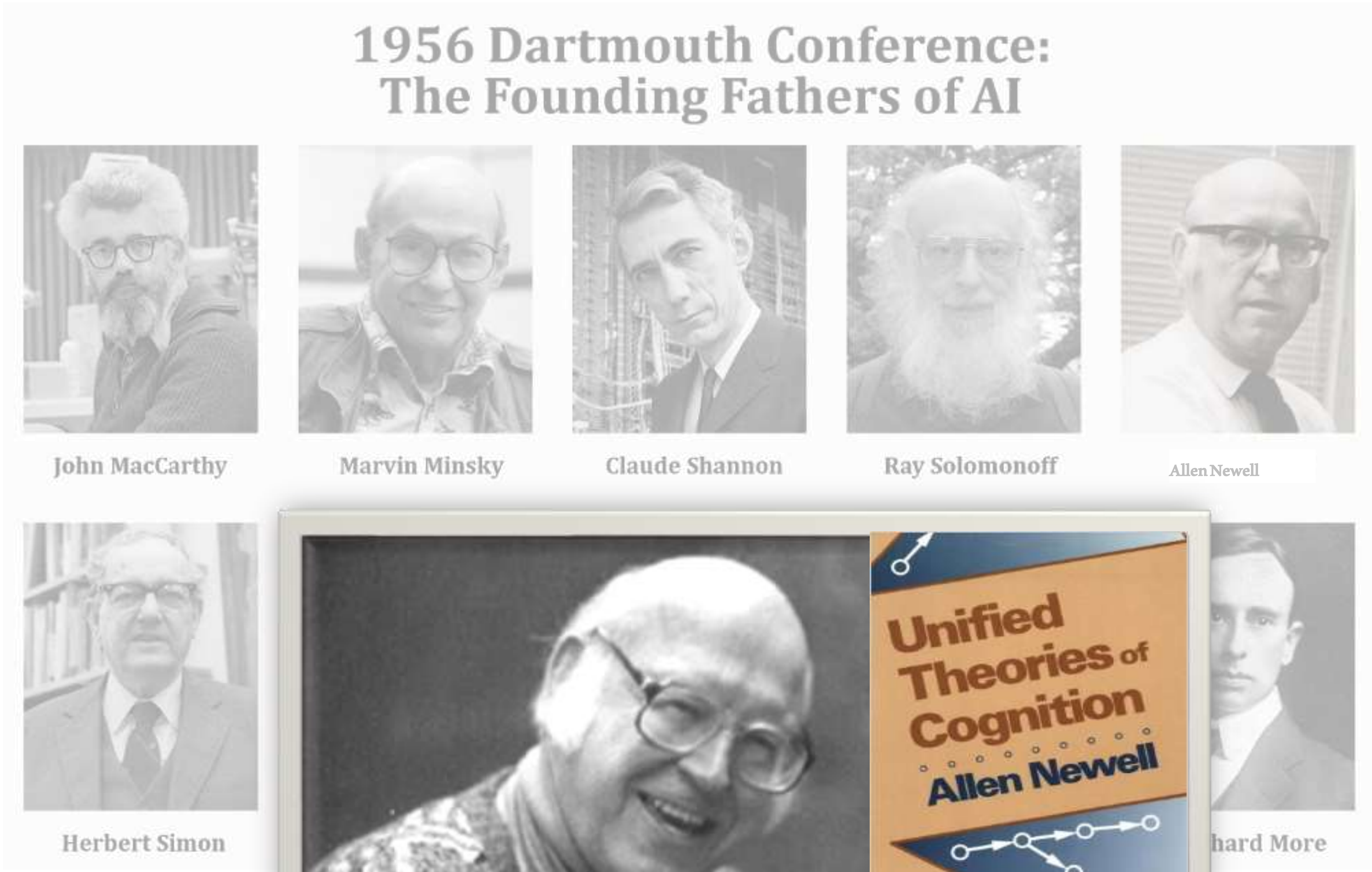
Soar’s Origins

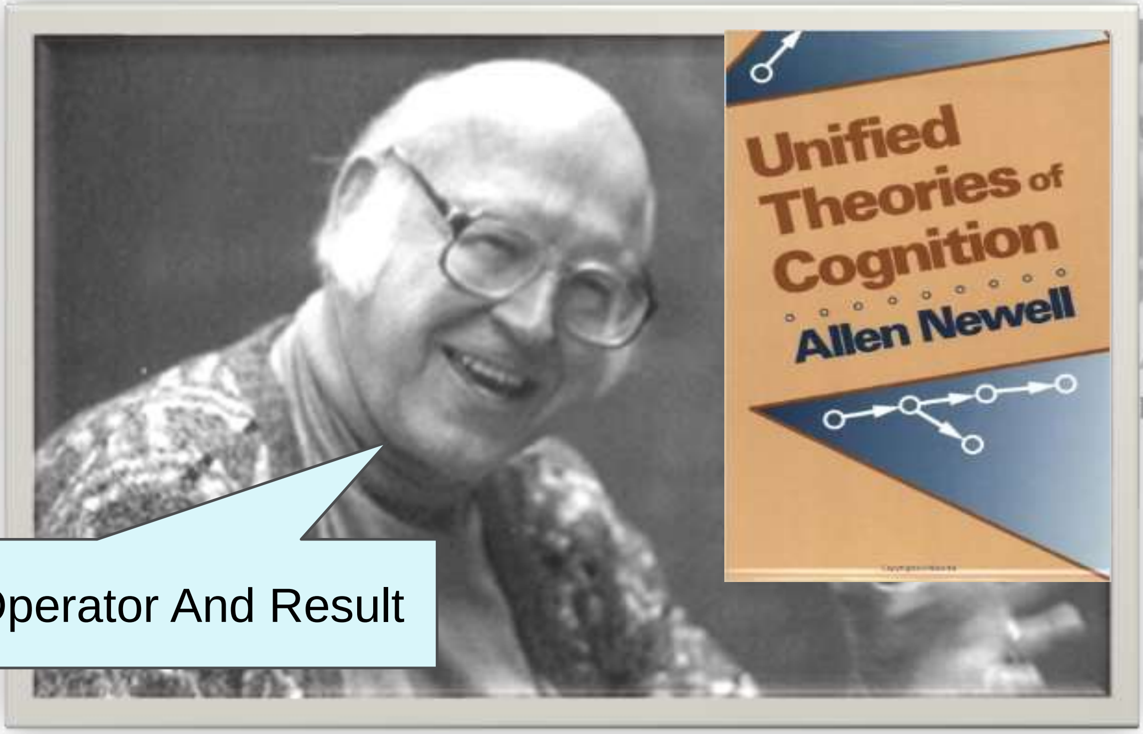
Soar was Newell’s instantiation of Cognitive Architecture

- Co-created with Newell’s students John Laird and Paul Rosenbloom in the 1980s
- Intention for it to evolve over time with continued research

Soar continues to evolve

- Led by John Laird and others
- Now in version 9.6





SOAR = State Operator And Result

The Modern Soar Cognitive Architecture

Soar is today the most popular cognitive architecture for use in AI applications.

- It is 35+ years mature in the academic community and has spun off to a successful military contractor Enterprise ($25M+ AR) serving DARPA, Navy, and the Air Force.
- It is free and open-source (BSD License)
- Large enterprises (such as Optum) have begun to adopt it in recent years
  - Typically used for cases where rules engines are useful in concept but not adaptive enough in practice
  - Soar allows rule-like behavior that is more adaptive and generative while permitting full transparency and governance that one cannot get with black-box ML approaches

What Do People Use Soar For?

Autonomous, real-time, interactive agents

- Ex: UAVs, personal assistants, game AI

Contextual conflict resolution in decision making

- A>B, B>C, C>A… autonomously solve and continue
  - Ex: Policy1 says “recommend brain surgery”. Policy2 says “require continuous blood thinners”. Policy3 says “brain surgery requires stopping blood thinners”….
- Use general problem solving to resolve conflicts
  - Can include asking humans for input if needed

Learning rules from experience (the results of past problem solving)

- Ex: After finding the desired behavior to the blood thinner problem, autonomously learn it for the next time that scenario is encountered

What is Soar's Architecture?

Production Rules

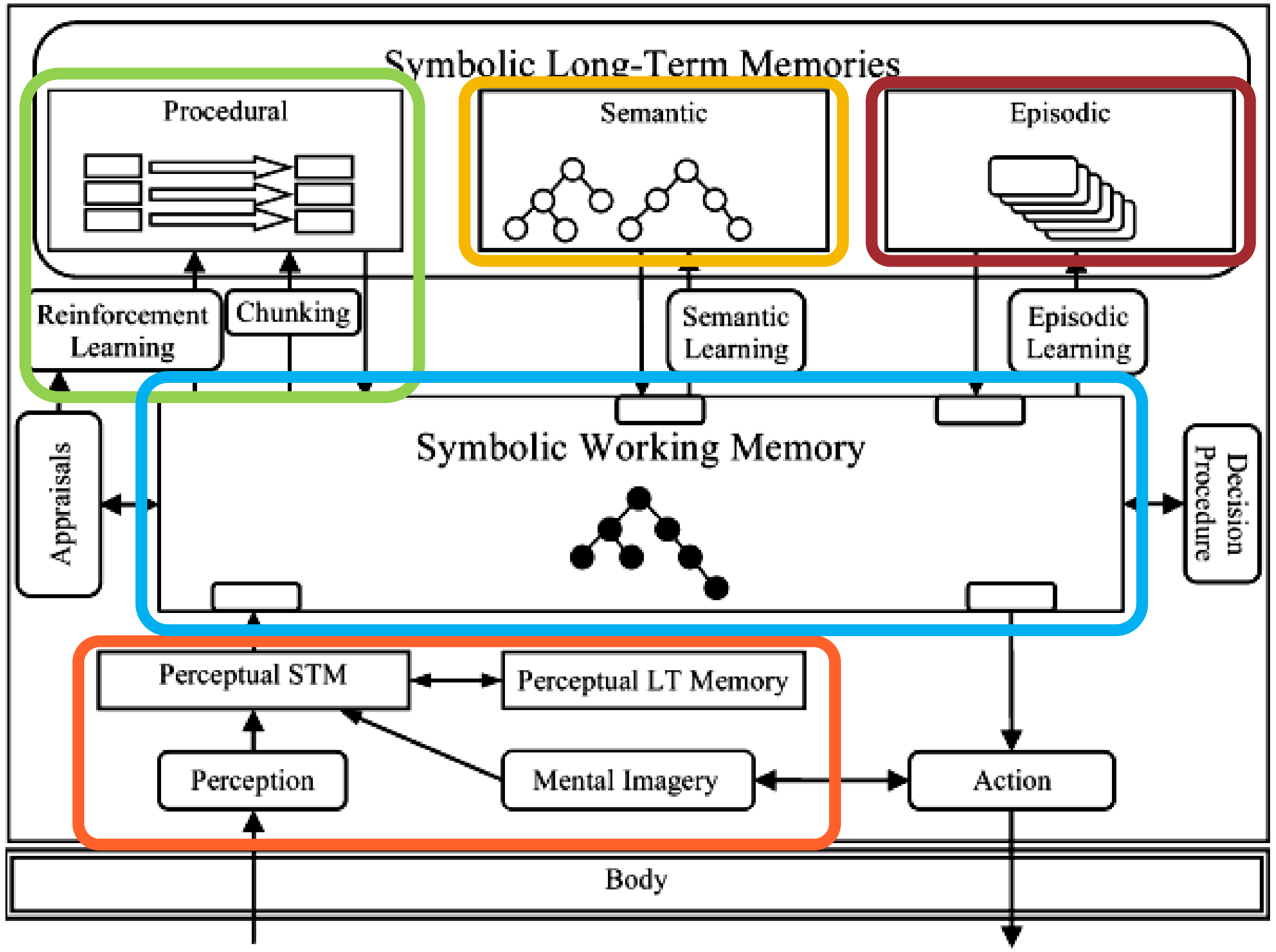
- **Procedural Memory**
- Rules created by programmer, or learned

Symbolic Graph-based Memory

- **Working Memory**
  - The central hub, a single memory graph
  - Rule conditions and actions happen here
- **Semantic Memory** (**optional**)
  - Long-term graph storage
- **Episodic Memory** (**optional**)
  - Snapshots of Working Memory over time

Other

- **Semantic Visual System** (**optional**)
  - Converts 3D input into symbolic graph input
  - Allows agent to control focus of perception



What is Soar's Architecture?

Production Rules

- **Procedural Memory**
- Rules created by programmer, or learned

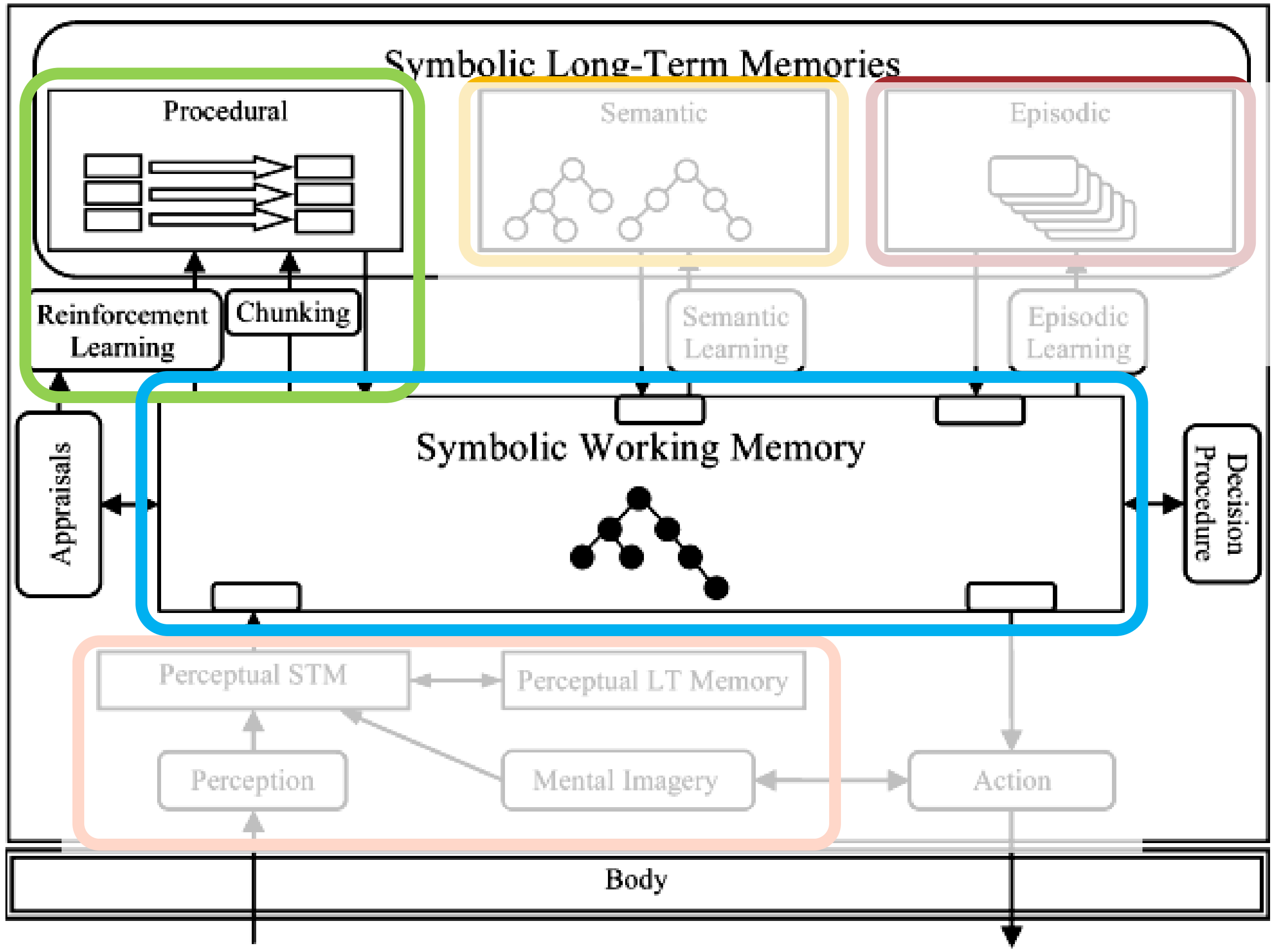
Symbolic Graph-based Memory

- **Working Memory**
  - The central hub, a single memory graph
  - Rule conditions and actions happen here
- **Semantic Memory (optional)**
  - Long-term graph storage
- **Episodic Memory (optional)**
  - Snapshots of Working Memory over time

Other

- **Spatial Visual System (optional)**
  - Converts 3D input into symbolic graph input
  - Allows agent to control focus of perception

This course will focus on Procedural Memory and Working Memory



Modern Soar as an Application

Single C++ executable

- Lightweight – can run on old smartphone, can run on large cluster
  - (more complex processing possible with greater resources)
  - Soft limit of 50 msec per I/O cycle (usually much less than that)
- Single-threaded

Has API libraries for other common languages (built using SWIG)

- Java
- Python
- C#
- Tcl

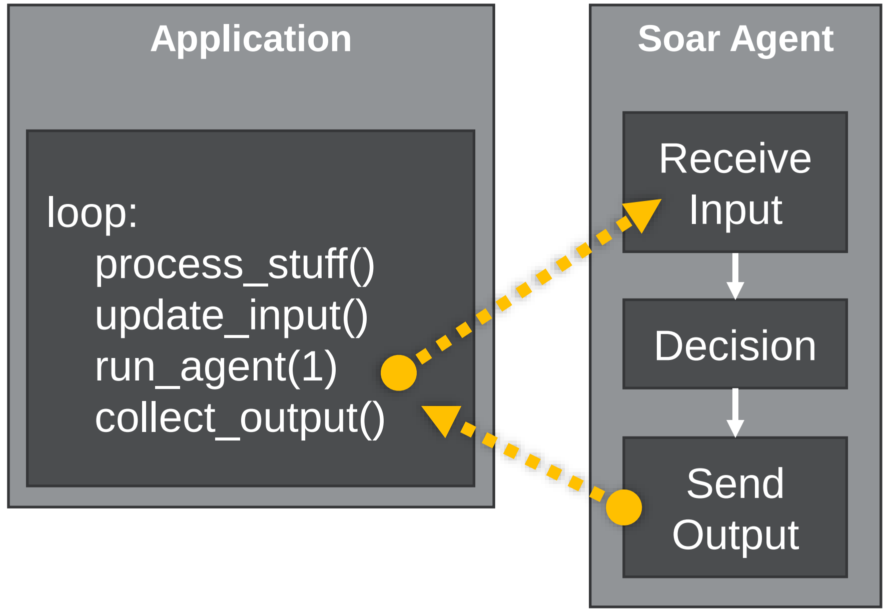
How Does Soar Fit Into Software?

Call Soar run methods from interfacing application

Can use event listeners

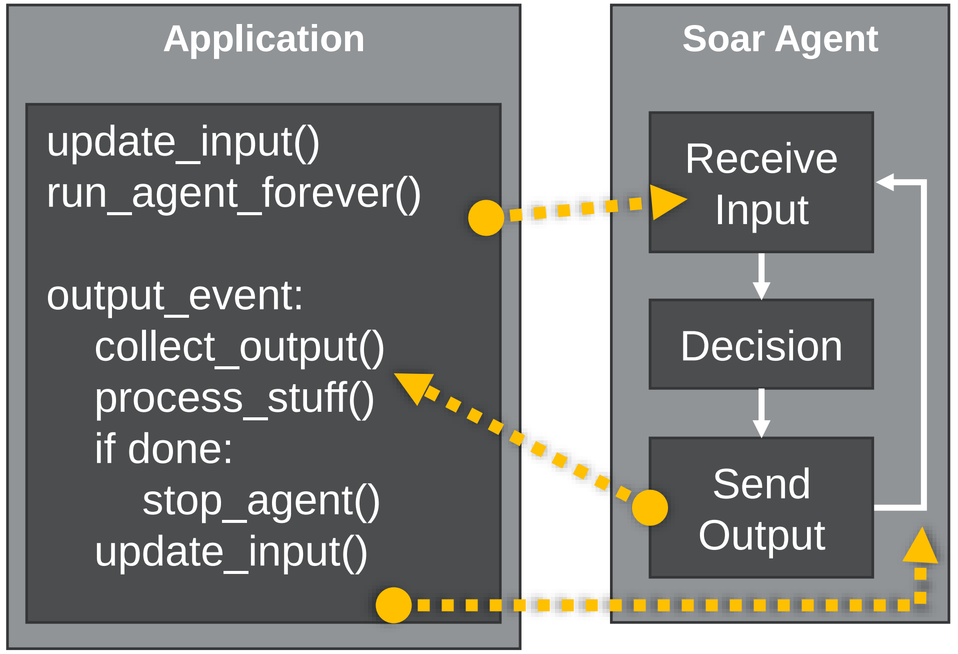
(We'll cover this in more detail later in the course.)

Using Direct Control



```
loop:
    process_stuff()
    update_input()
    run_agent(1)
    collect_output()
```

Using Event Listeners



```
update_input()
run_agent_forever()

output_event:
    collect_output()
    process_stuff()
    if done:
        stop_agent()
    update_input()
```

Soar as an Engine

Does **not provide** task functionality out of the box

Provides the *processing cycle* with features like autonomous decision-making and learning

- Like a game engine is for making computer games
- Like a cloud platform is for running hosted applications

The Soar programmer must define (using production rules):

- The choices used in decision making
- The graph data structures used in I/O and internal processing / learning
- The If/Then associations and sequences of logic

Soar's Learning Capabilities

Soar provides several advanced learning capabilities:

- **Chunking**: Autonomously learn new production rules
- **Reinforcement Learning**: Autonomously tune decision making and rule selection
- **Episodic Memory**: Autonomously record past experiences for rapid search and recall

Soar also provides a **Semantic Memory** store that allows long-term knowledge updates with more custom algorithms.

This course does not teach the learning features.
It teaches core Soar processing and programming.

Once you understand the core, the rest is easy.
Read the Soar Manual once you're ready to learn how to learn with Soar!

Soar Use Case: Rosie

A particular Soar agent

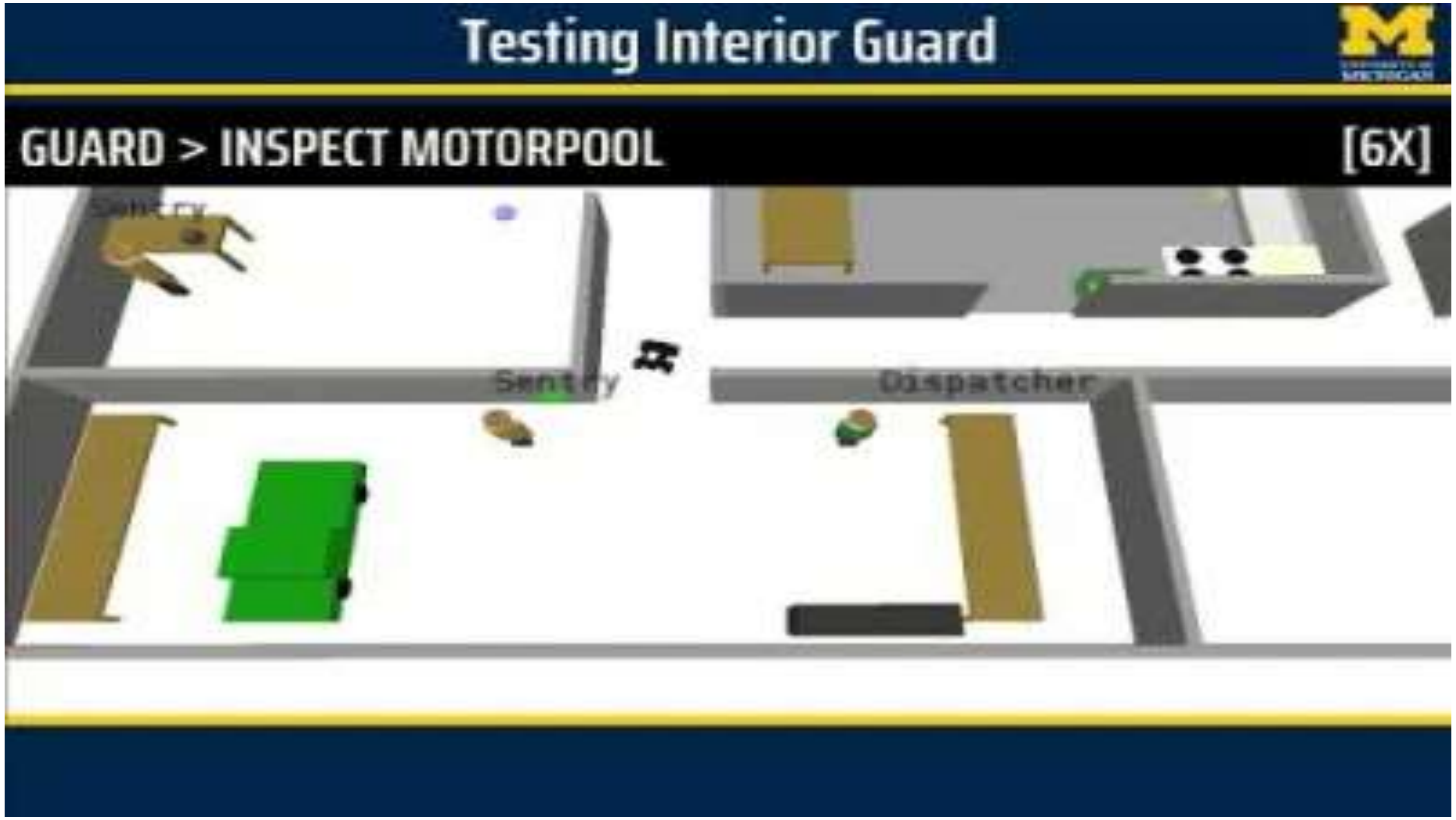
- http://soargroup.github.io/rosie

Learns tasks by instruction

- Language interaction
- Asks for details when it doesn't understand
- Remembers definitions given in the past
- Builds up a vocabulary of concepts and skills over time.

Video:

- https://www.youtube.com/watch?v=_KsqaUavksk



Setting Up Soar

On Your Local Machine

The Soar Executable

The course materials include a copy of Soar 9.6.2 in the folder, “SoarSuite_9.6.2-Multiplatform”

- You can also download Soar from https://soar.eecs.umich.edu

Files of interest inside the Soar folder:

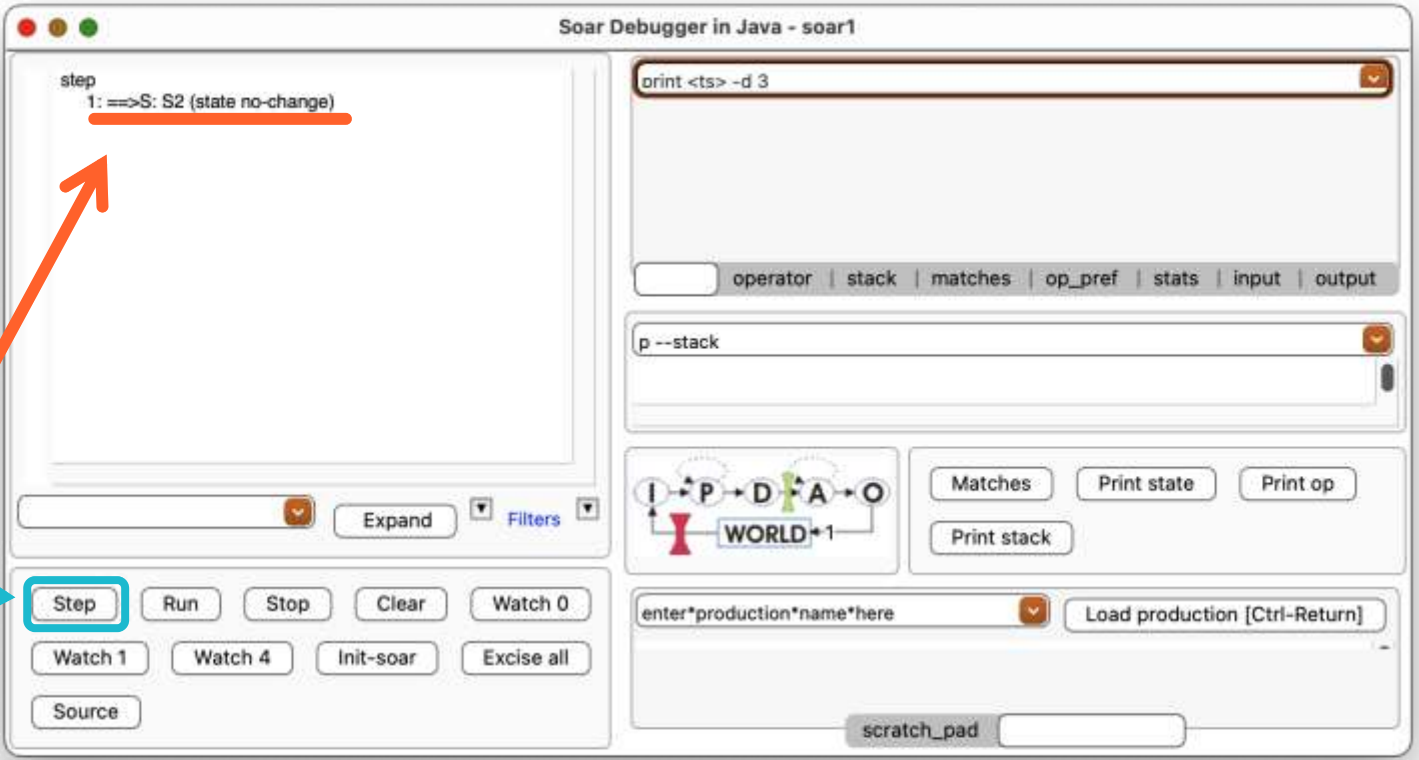
File	Name	Description
bin/soar (.exe)	Soar Executable	The core Soar application
SoarJavaDebugger.sh (.bat)	Debugger Run Script	A useful tool for running/testing Soar agents
bin/Python_sml_ClientInterface.py	Python interface module	Python interface code imports this. (Other languages use similar files.)
bin/_Python_sml_ClientInterface.so (.pyd)	Python <-> C++ interpreter	The Python interface module uses this to interact with the Soar executable (Other languages use similar files.)
SoarManual.pdf	The Soar Manual	The definitive guide for all your unanswered Soar questions

Running Soar Tutorial Projects

The course materials provide some Python scripts that automatically start Soar with the code you write, using a SoarJavaDebugger window.

1. Run the ”**run_project.py**” script located in the *Project00_StartingSoar* folder.
   - This window should open ➔
   - This is the **SoarJavaDebugger**
   - It is a Java-based tool for interacting with and debugging Soar
2. Click the “Step” button once
   - If Soar is running, you should see “1: ==>S: S2 (state no-change)” appear in the main text area.

Each project will provide a similar ”run_project.py” script you can use to quickly test your code.



```
step
   1: ==>S: S2 (state no-change)
```

print <ts> -d 3

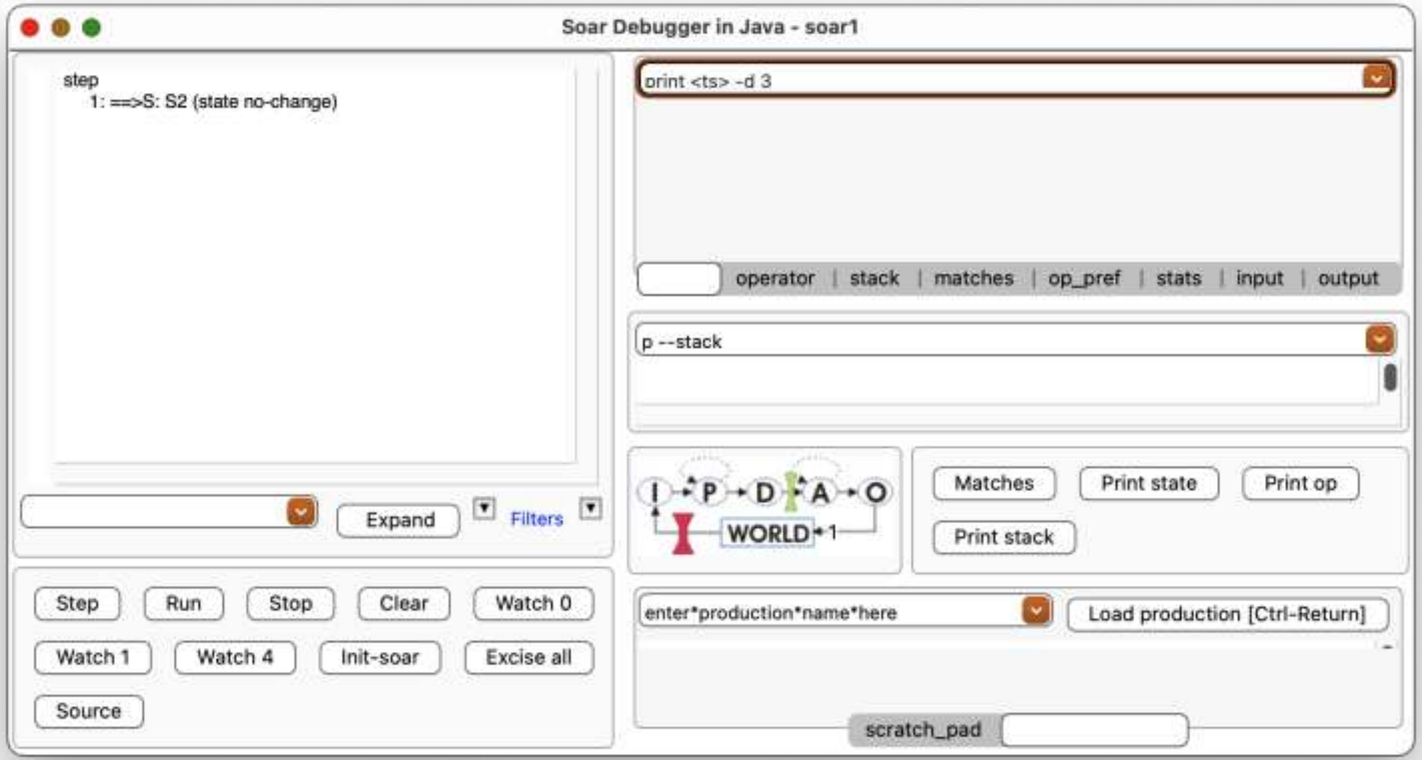
p --stack

enter*production*name*here

Opening the Java Debugger Directly

The direct way to open the Soar Java Debugger is to run the SoarJavaDebugger.sh/.bat file.

- (If using the .sh file, search online for how to set your computer to run the script when you double-click the file so you don't need to use your terminal.)
- The first time you run this file it will check your operating system and copy the appropriate Soar build files into the bin directory.
  - If running the SoarJavaDebugger file fails, you may need to add permissions on your machine to allow the script to run.
- The Python script mentioned in the previous slide calls this same SoarJavaDebugger file, so it will also automatically perform this file copy the first time you run that script.



```
step
  1: ==>S: S2 (state no-change)
```

```
print <ts> -d 3
```

```
p --stack
```

```
enter*production*name*here
```

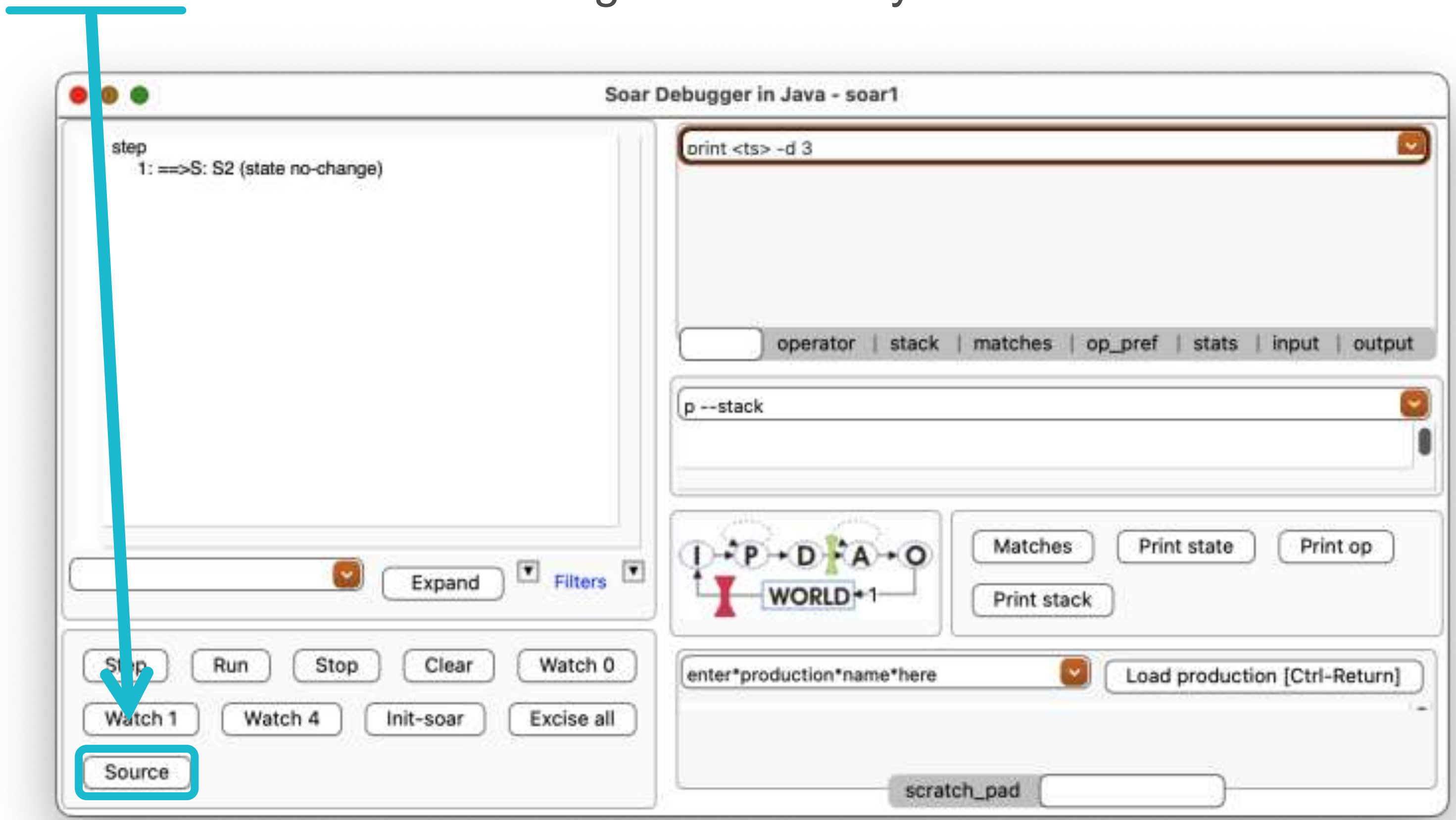
Loading Soar Agents Manually

The *run_project.py* script automatically loads your agent code into Soar.

Running the *SoarJavaDebugger* directly does not!

- After opening the debugger, click the "Source" button to load an agent code file you have written into this Soar instance.

(You will not need to do this for projects in this course if you use the run_project.py script.)



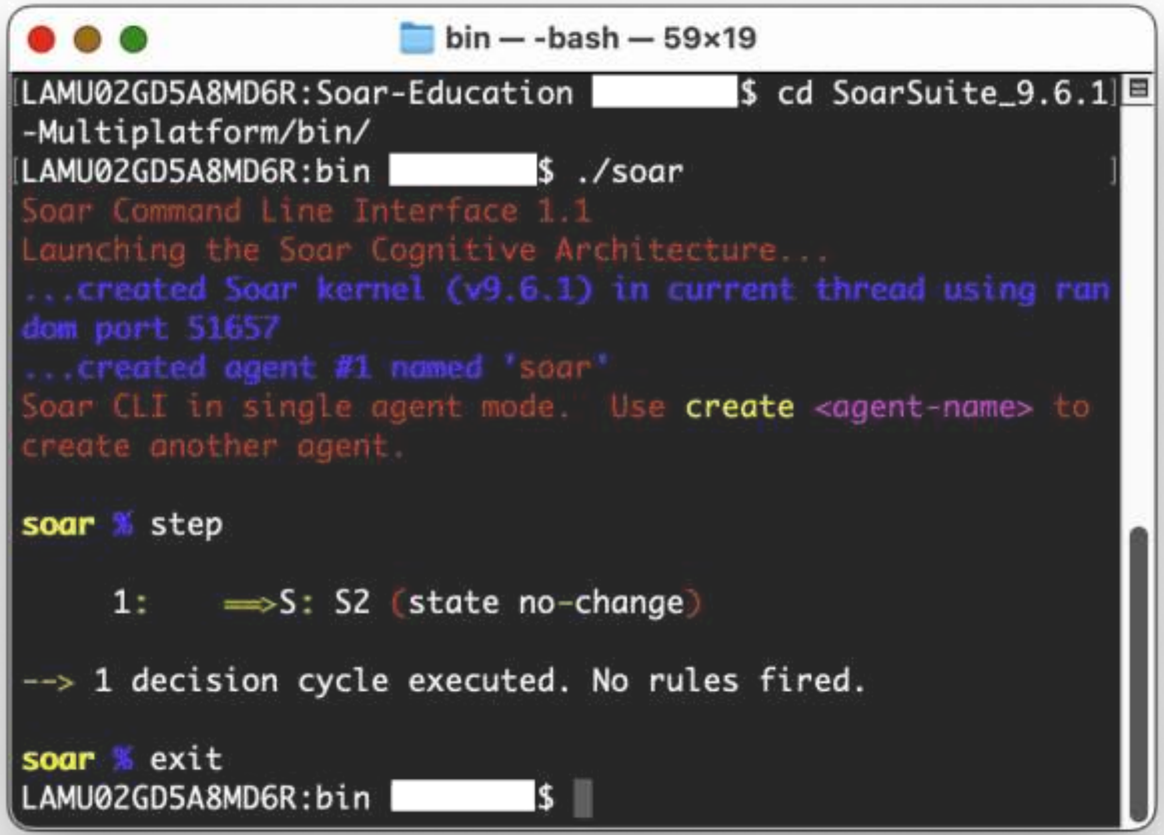
Starting the Soar Command Line Interface

Soar doesn't need a debugger window.

There are three easy ways to run Soar direct from your terminal / command line:

1. From your terminal, run the **`soar`** (Mac/Linux) or **`soar.exe`** (Windows) executable
   - Initially this will be under the platform-specific directory within the `bin` folder.
2. Run the "**`SoarCLI.sh/.bat`**" script
   - This automatically runs the soar executable file appropriate for your OS.
3. Run the **`run_soar_cli()`** Python function in the *`run_soar.py`* file provided within this tutorial.

(The "exit" command ends the Soar process.)



```
LAMU02GD5A8MD6R:Soar-Education [redacted]$ cd SoarSuite_9.6.1
-Multiplatform/bin/
LAMU02GD5A8MD6R:bin [redacted]$ ./soar
Soar Command Line Interface 1.1
Launching the Soar Cognitive Architecture...
...created Soar kernel (v9.6.1) in current thread using ran
dom port 51657
...created agent #1 named 'soar'
Soar CLI in single agent mode.  Use create <agent-name> to
create another agent.

soar % step

     1:     ==>S: S2 (state no-change)

--> 1 decision cycle executed. No rules fired.

soar % exit
LAMU02GD5A8MD6R:bin [redacted]$
```

Installing Soar for Other Applications

This step is not needed for projects in this course.

Your project files will call the Soar executable using relative paths in the tutorial directory.

If you want to install Soar such that you can use it in other applications without knowing the relative path, use **environment variables** so that those applications can find your Soar bin directory.

Set these environment variables on your machine to include the “bin” subfolder of your Soar installation:

• `SOAR_HOME`	*(Soar libraries reference this; all systems)*
• `DYLD_LIBRARY_PATH`	*(library search path; Windows/Linux systems)*
• `PATH`	*(dll search path; Windows systems)*
• `PYTHONPATH`	*(if you intend to run Soar from Python code)*
• CLASSPATH	*(if you intend to run Soar from Java code)*

(Search online for how to set up environment variables on your machine if you need help.

Read this article for more info about search paths in Soar:

- https://soar.eecs.umich.edu/articles/articles/building-soar/86-how-library-search-paths-work