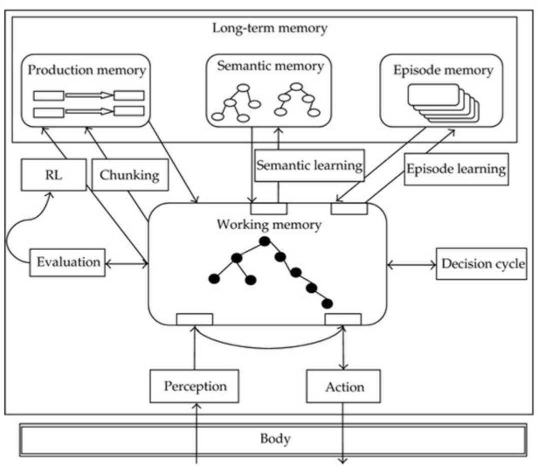
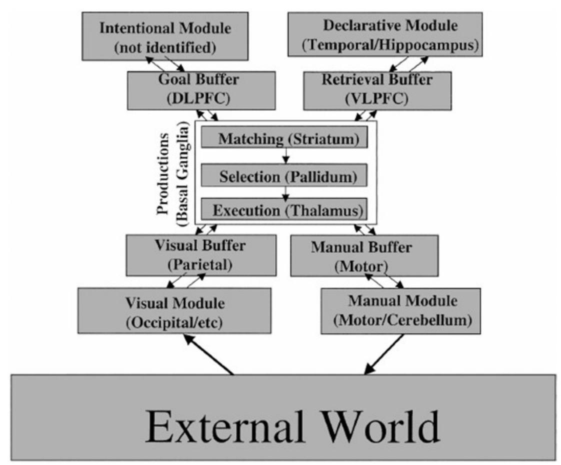
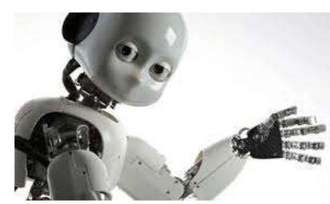
Cognitive Robotics and Cognitive Architectures

Cognitive Robotics

- Cognitive Robotics
  - Embodied AI/Embodied CS
    - Robot capable of perception, reasoning, learning, deliberate, planning, acting, interacting, etc.
  - Cognitive Architectures
    - Unified Theory of Cognition
    - Cognitive Models







What is Cognitive Architecture?

Blueprint for Intelligent Agents

It proposes (artificial) computational processes that act like cognitive systems (human)

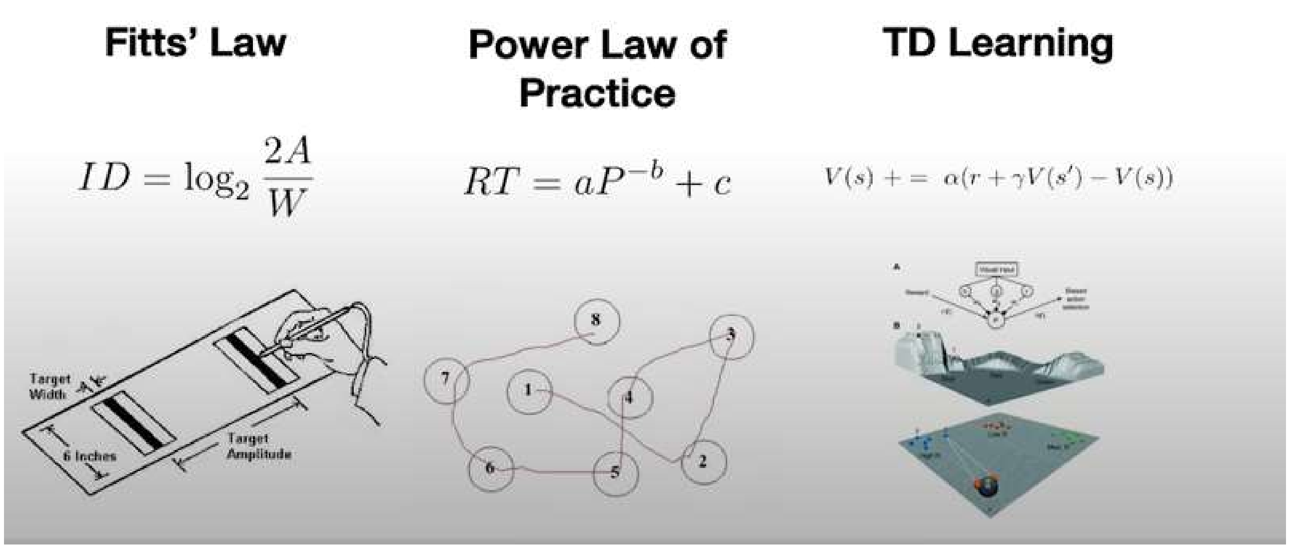
An approach that attempts to model behavioral as well as structural properties of the modeled system.

Aim:

- to summarize the various results of cognitive psychology in a comprehensive computer model
- to model systems that accounts for the whole of cognition.

What is Cognitive Architecture?

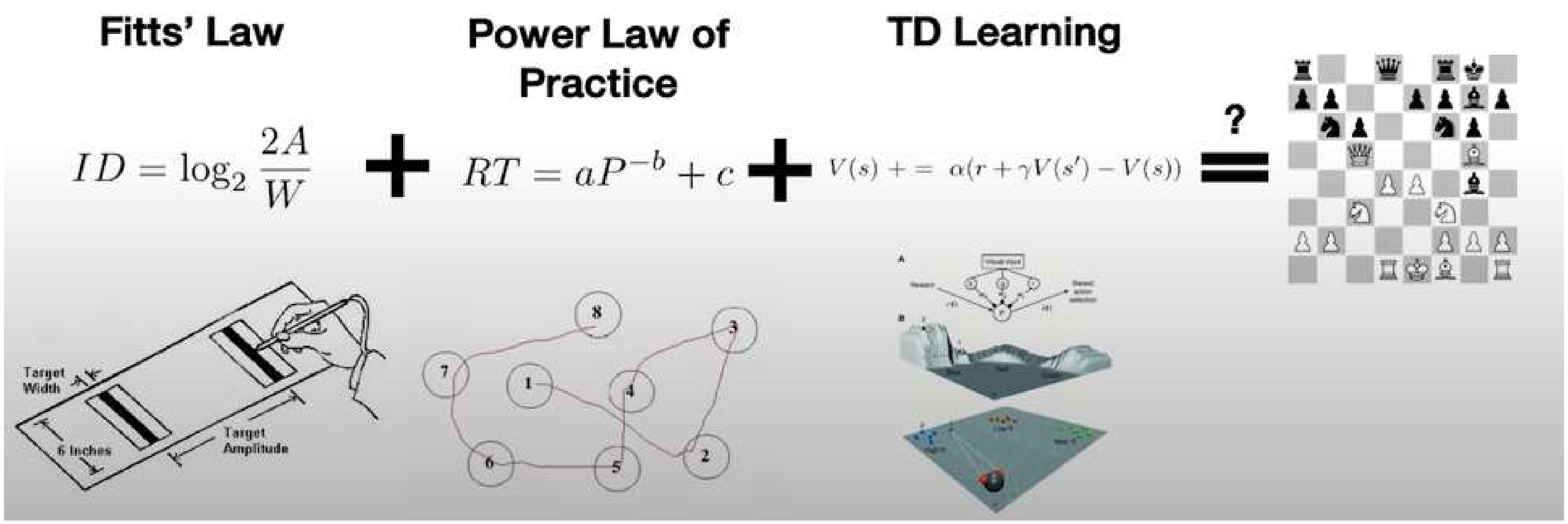
Integrate and generalize different findings on intelligent behaviour



Debrinsky – MIT AGI

What is Cognitive Architecture?

Integrate and generalize different findings on intelligent behaviour



Debrinsky – MIT AGI

Unified Theory of Cognition

Book by Allen Newell

Newell's aim:

To define the architecture of human cognition, which is the way that humans process information. This architecture must explain how we react to stimuli, exhibit goal directed behavior, acquire rational goals, represent knowledge, and learn.

Unified Theory of Cognition

Cognitive Architecture specifies aspects of cognition that remain constant across lifetime of an agent:

- Memory systems of beliefs, goals, experience
- Knowledge Representation
- Processes: perception, execution, cognition
- Learning mechanisms

Goal: understand and exhibit intelligence across several tasks and domains

Artificial General Intelligence (AGI)

Unified Theory of Cognition

[Newell 1990] Regularities at multiple scales and abstraction layers:
- Biological, Cognitive, Rational, Social, etc.

Scale (sec)	Time Units	System	World (theory)
10^7	Months		
10^6	Weeks		Social Band
10^5	Days		
10^4	Hours	Task	
10^3	10 min	Task	Rational Band
10^2	Minutes	Task	
10^1	10 sec	Unit Task	
10^0	1 sec	Operations	Cognitive Band
10^{-1}	100 ms	Deliberate act	
10^{-2}	10 ms	Neural circuit	
10^{-3}	1 ms	Neuron	Biological Band
10^{-4}	100 μs	Organelle	

Cognitive Band

Time Units	System	Cog. Capabilities
10 sec	Unit Task	Complex reasoning Planning, Theory of Mind
1 sec	Composition	Simple Reasoning, Language
100 ms	Deliberation	Reactive decisions Skilled behavior, Access LTM

Regularities at 100ms [Newell 1990], architecture at this level

Bounded Rationality

Agent rationality is limited [Simon 1957]

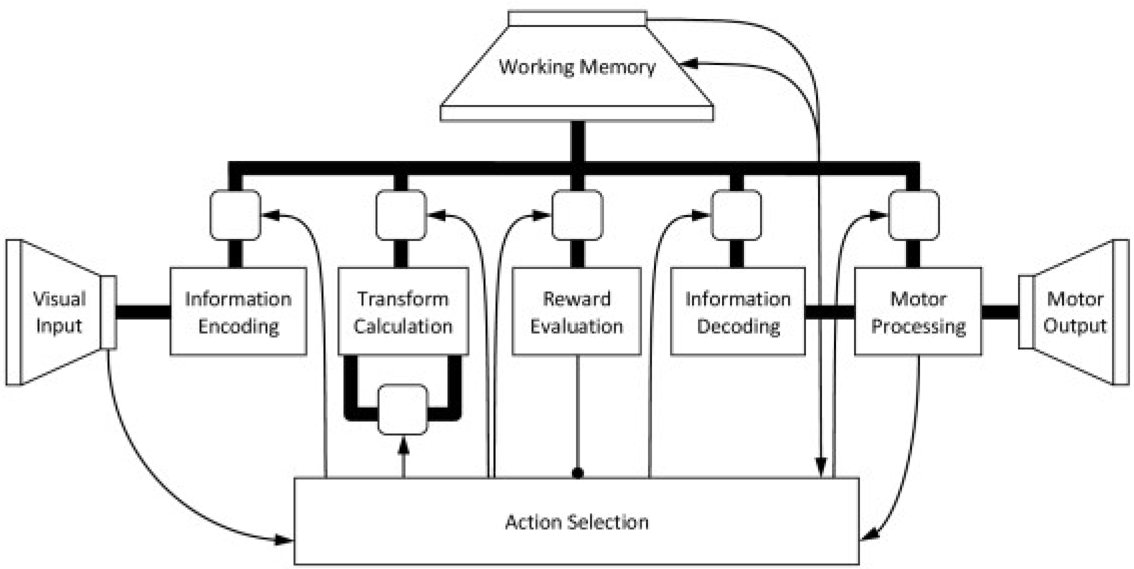
- Tractability of the problem
- Cognitive/Computational limitations
- Time available

Different Research Goals

- Biological Plausibility
  - Leabra [bio and cog band]
  - SPAUN [bio and cog band]
- Psycological Plausibiliy
  - ACT-R, CLARION, EPIC [Cog and Rational band]
- <u>Agent Functionality</u>
  - Soar, Sigma, ICARUS, LIDA [Cog and Rational band]

Different Research Goals

- SPAUN:
  - Semantic Pointer Architecture
    - Semantic pointers are neural representations that carry partial semantic content and are composable into the representational structures necessary to support complex cognition
    - cognitive and non-cognitive tasks integrated in a single large-scale, spiking neuron model



Another Taxonomy

- <u>Cognitive:</u>
  - SOAR, ACT-R, ICARUS, ADAPT, EPIC, etc.
- <u>Emergent:</u>
  - SASE, DARWIN, SPAUN, Global Workspace, etc.
- <u>Hybrid:</u>
  - CLARION, HUMANOID, Cog: Theory of Mind, Kismet, LIDA, etc.

- <u>Robotics</u> (embodied agent):
  - ACT-RE, ADAPT, HUMANOID, Kismet, Cog, ICARUS, etc.

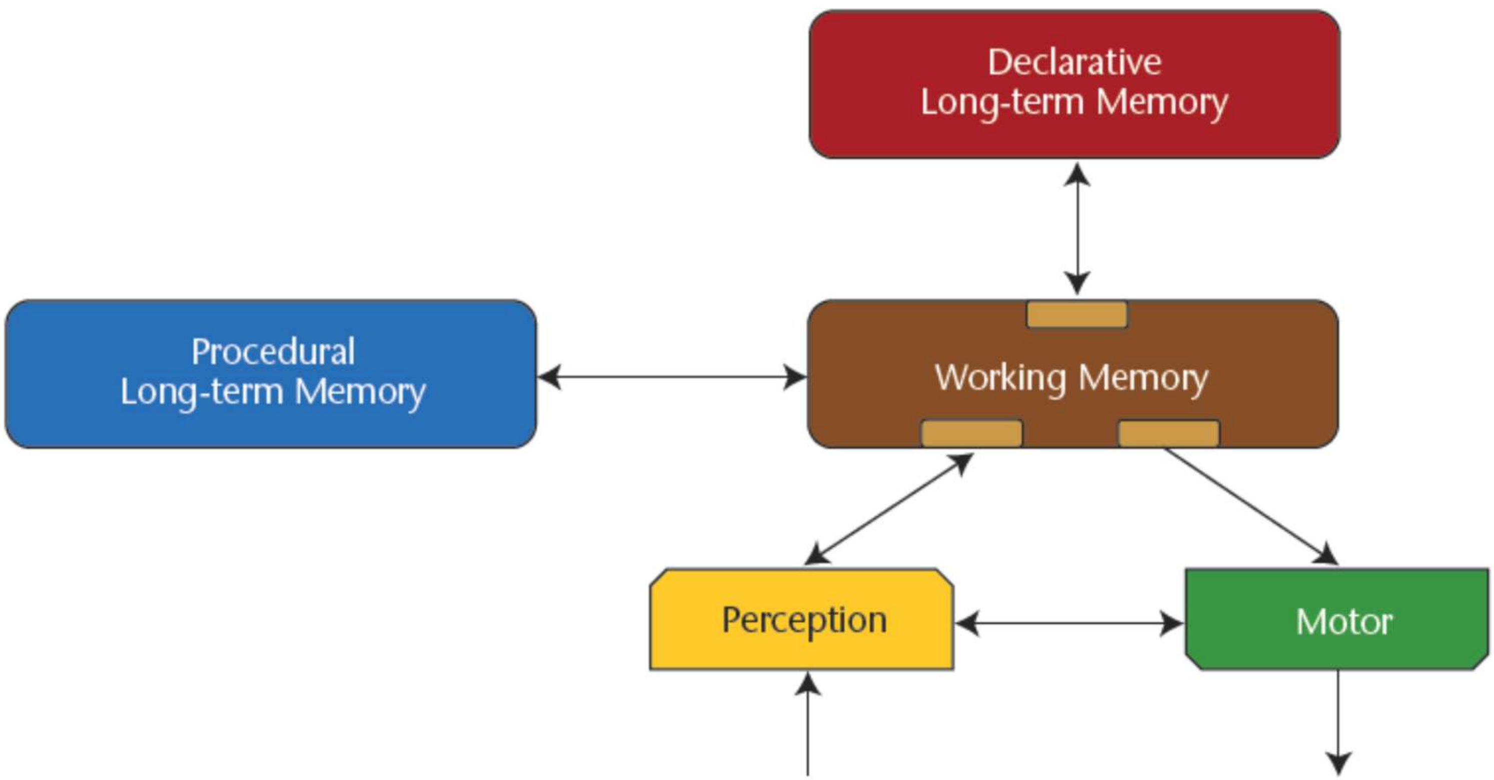
1970	• GPS (Ernst & Newell, 1969) Means-ends analysis, recursive subgoals **• ACT (Anderson, 1976) Human semantic memory** • CAPS (Thibadeau, Just, Carpenter) Production system for modeling reading
1975	**• Soar (Laird, & Newell, 1983) Multi-method problem solving, production systems, and problem spaces** • Theo (Mitchell et al., 1985) Frames, backward chaining, and EBL
1980	**• PRS (Georgeff & Lansky, 1986) Procedural reasoning & problem solving** **• BB1/AIS (Hayes-Roth & Hewitt 1988) Blackboard architecture, meta-level control**
1985	• Prodigy (Minton et al., 1989) Means-ends analysis, planning and EBL • MAX (Kuokka, 1991) Meta-level reasoning for planning and learning
1990	• Icarus (Langley, McKusick, & Allen,1991) Concept learning, planning, and learning • 3T (Gat, 1991) Integrated reactivity, deliberation, and planning
1995	• CIRCA (Musliner, Durfee, & Shin, 1993) Real-time performance integrated with planning **• AIS (Hayes-Roth 1995) Blackboard architecture, dynamic environment**
2000	• EPIC (Kieras & Meyer, 1997) Models of human perception, action, and reasoning

Cognitive Architecture

- Architecture:
  - Modules, processes, communication, data/knowledge
  - Cognitive Cycle:
    - Complex behavior usually obtained from primitive processes and decisions generated/monitored through cycles
    - Regularities at 100ms

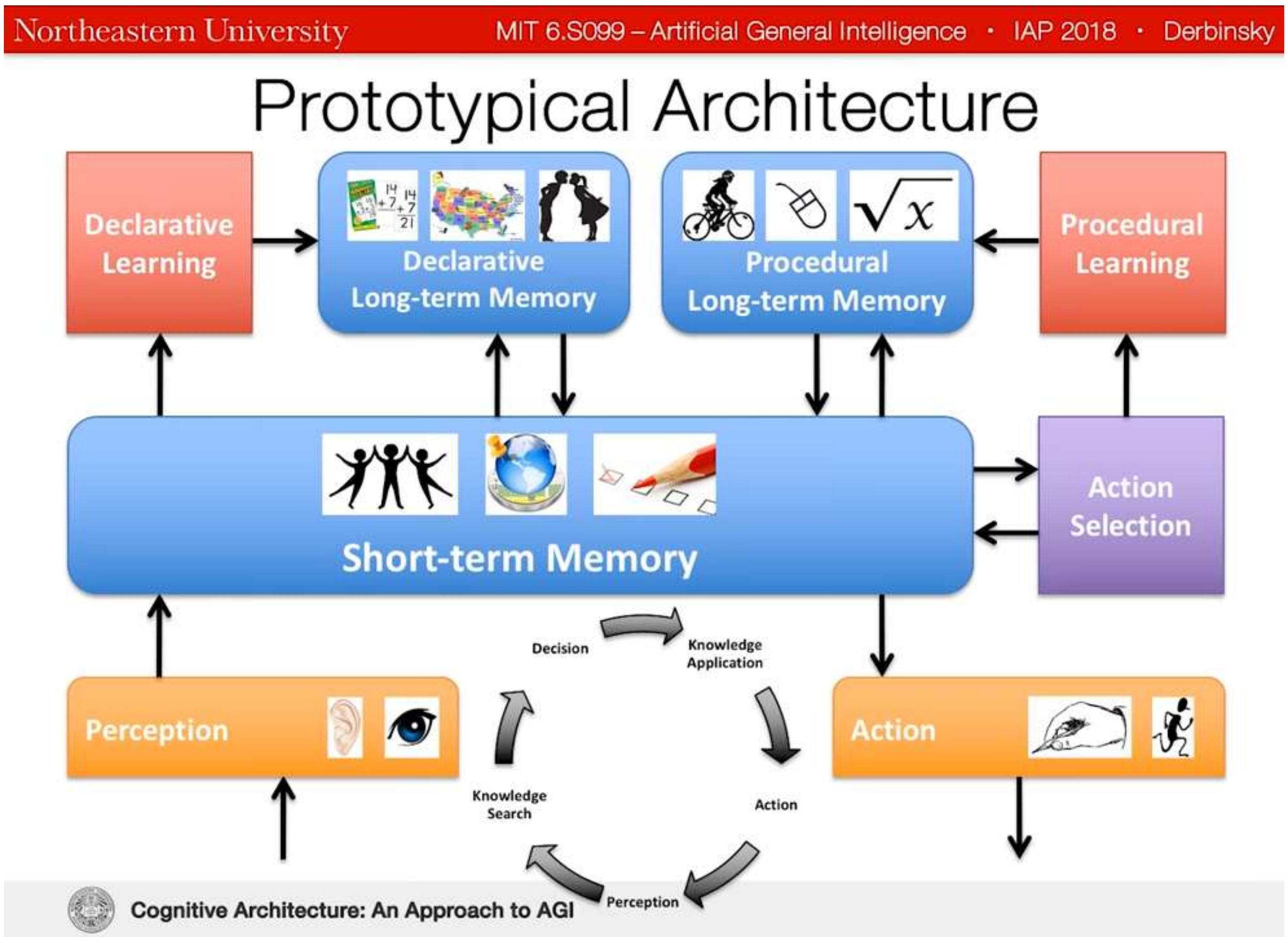
Cognitive Architecture

- Standard Model [J. Laird et al. 2017]



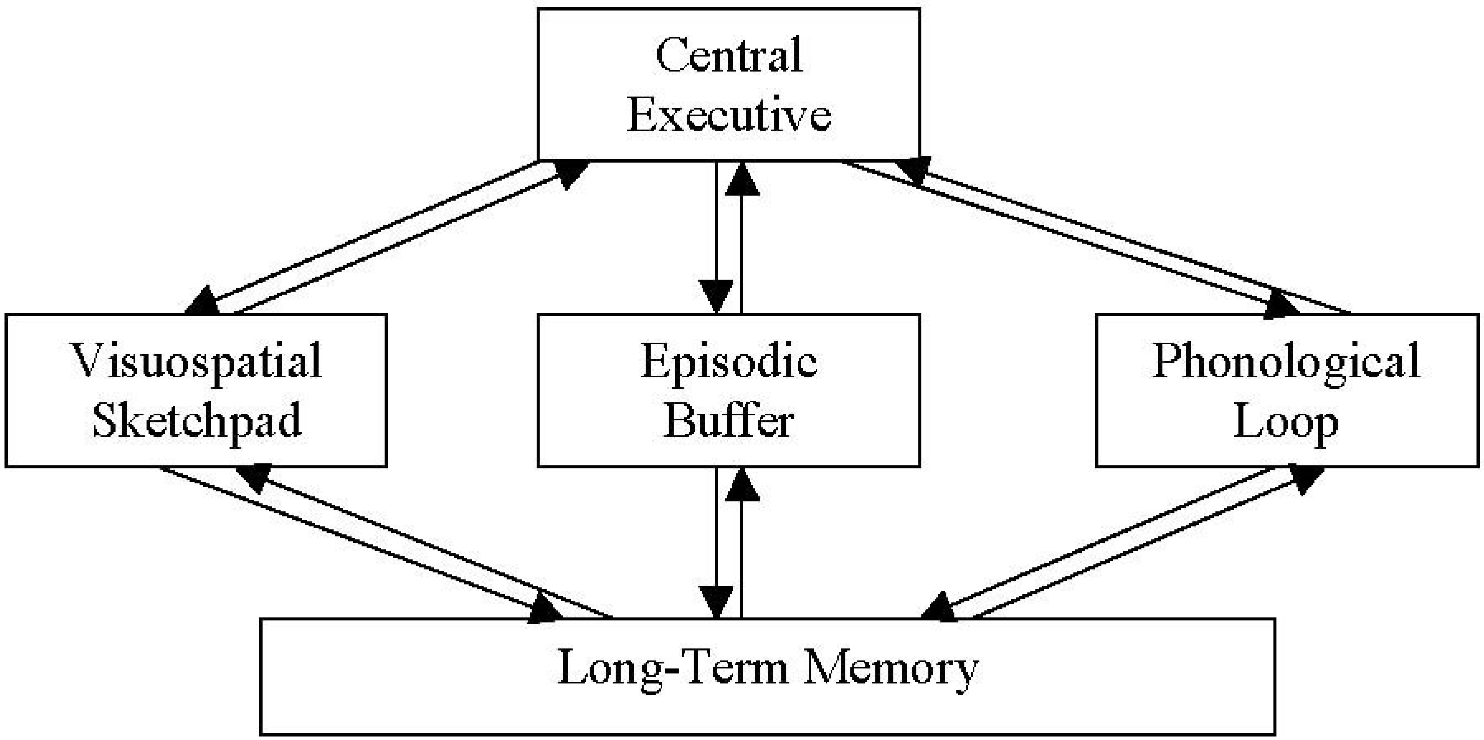
Cognitive Architecture

- Standard Model [J. Laird et al. 2017]



Baddeley Model

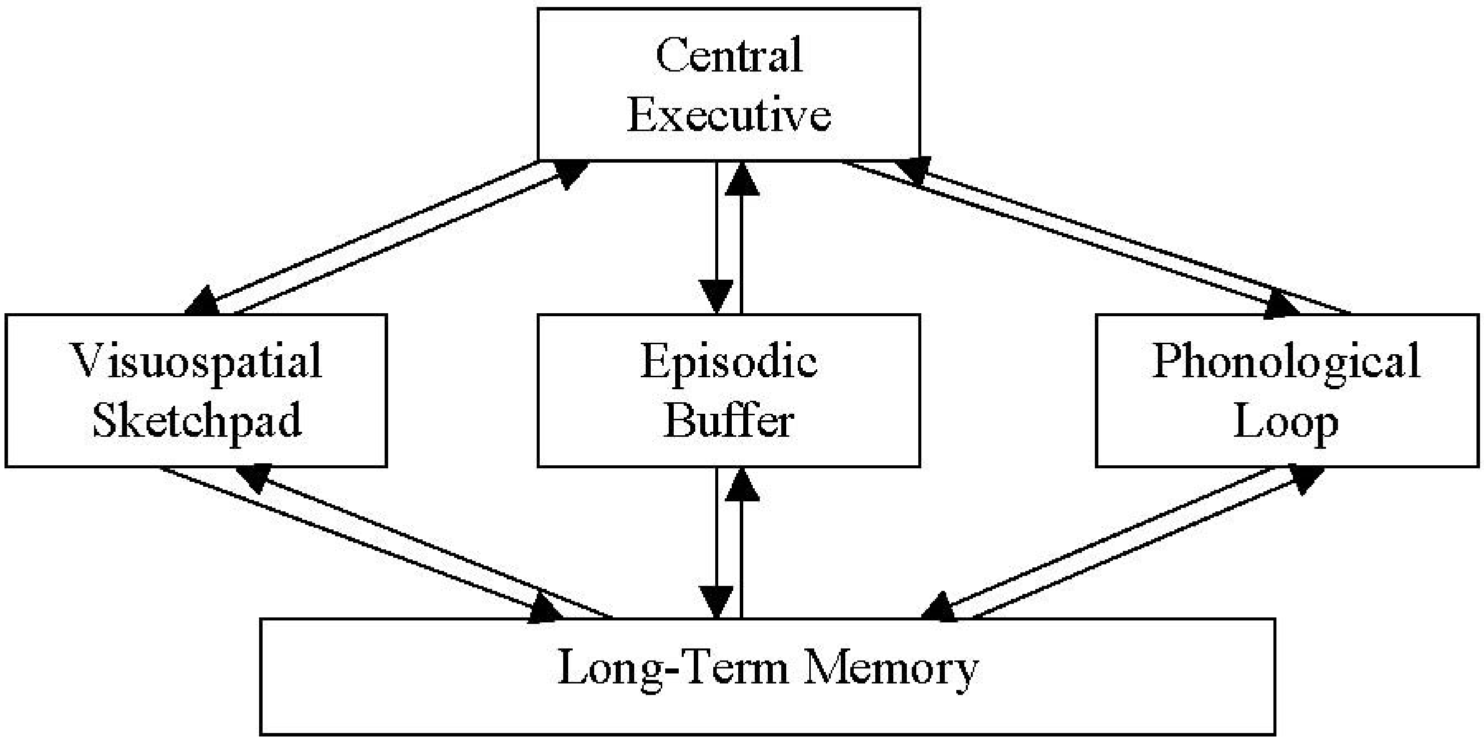
Central Executive model [Baddeley & Hitch 1976, Baddeley 1986, Baddeley 2000] of Working Memory



Central Executive drives the whole system and allocates data to the subsystems

Baddeley Model

Central Executive model [Baddeley & Hitch 1976, Baddeley 1986, Baddeley 2000] of Working Memory



- Visuospatial sketchpad stores and processes information in a visual or spatial form
- Phonological loop deals with spoken and written material
- Episodic buffer 'backup' store which communicates with both LTM and WM

Newell's Cognitive Model

Newell introduces Soar: architecture for general cognition.

Soar is a problem solver that creates its own sub-goals and learns from its own experience.

Soar operates with real-time constraints:

- immediate-response, item-recognition tasks, etc..

What is Soar?

Soar is a symbolic cognitive architecture:

- AI programming framework
- Cognitive architectural framework to define and exploit cognitive models
- Architecture for knowledge-based problem solving, learning, and interaction with external environments
- Physical symbol system hypothesis:
  - a symbolic system is necessary for general intelligence

Newell's Cognitive Model

Created by John Laird, Allen Newell, and Paul Rosenbloom at Carnegie Mellon University in 1983

John Laird

Allen Newell

Paul Rosenbloom

Soar

Historically, Soar was for **State, Operator And Result**, because problem solving in Soar is a search through a problem space in which you apply an operator to a state to get results

Over time, the community no longer regarded Soar as an acronym: this is why it is no longer written in upper case

Problem Solving

- Soar is based upon a theory of human problem solving (symbolic):
  - problem solving activity is formulated as the selection and application of operators to a state, to achieve some goal.
  - Problem Space Hypothesis:
    - all behavior, even planning, is decomposable into a sequence of selection and application of primitive operators, which take about ~50ms
    - A single operator selected at each step (serial bottleneck), but selection and application associated with parallel rule firings (context-dependent retrieval of procedural knowledge).
  - Universal sub-goaling:
    - Impasses generates sub-states

Problem Solving

Newell introduces the *problem space principle* as follows.

"The rational activity in which people engage to solve a problem can be described in terms of (1) a set of states of knowledge, (2) operators for changing one state into another, (3) constraints on applying operators and (4) control knowledge for deciding which operator to apply next.“

Problem spaces (e.g. STRIPS domain) are commonly composed of a set of goals, a state or set of states, and a set of valid operators which contain the constraints under which the operator can be applied.

The state consists of a set of literals that describe the knowledge of the agent and the present model of the world.

Problem Spaces

Soar represents all tasks as collections of <u>problem spaces</u>

- Problem spaces are made up of a set of states and operators that manipulate the states

- Soar begins work on a task by choosing a <u>problem space</u>, then an <u>initial state</u> in the space

- Soar represents the goal of the task as some final state in the problem space

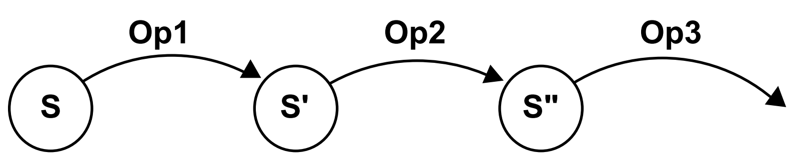
Soar

- G*oal*: is a desired situation.
- S*tate*: representation of a problem solving situation.
- P*roblem space:* set of states and operators for the task.
- O*perator*: transforms the state by some action.

Problem Space Level

- Behaviour in a problem spaces:
  - made up of States (S) and Operators (Op)

- Fluent behaviour:
  - an *operator* is selected and applied to the current state to give a new current state

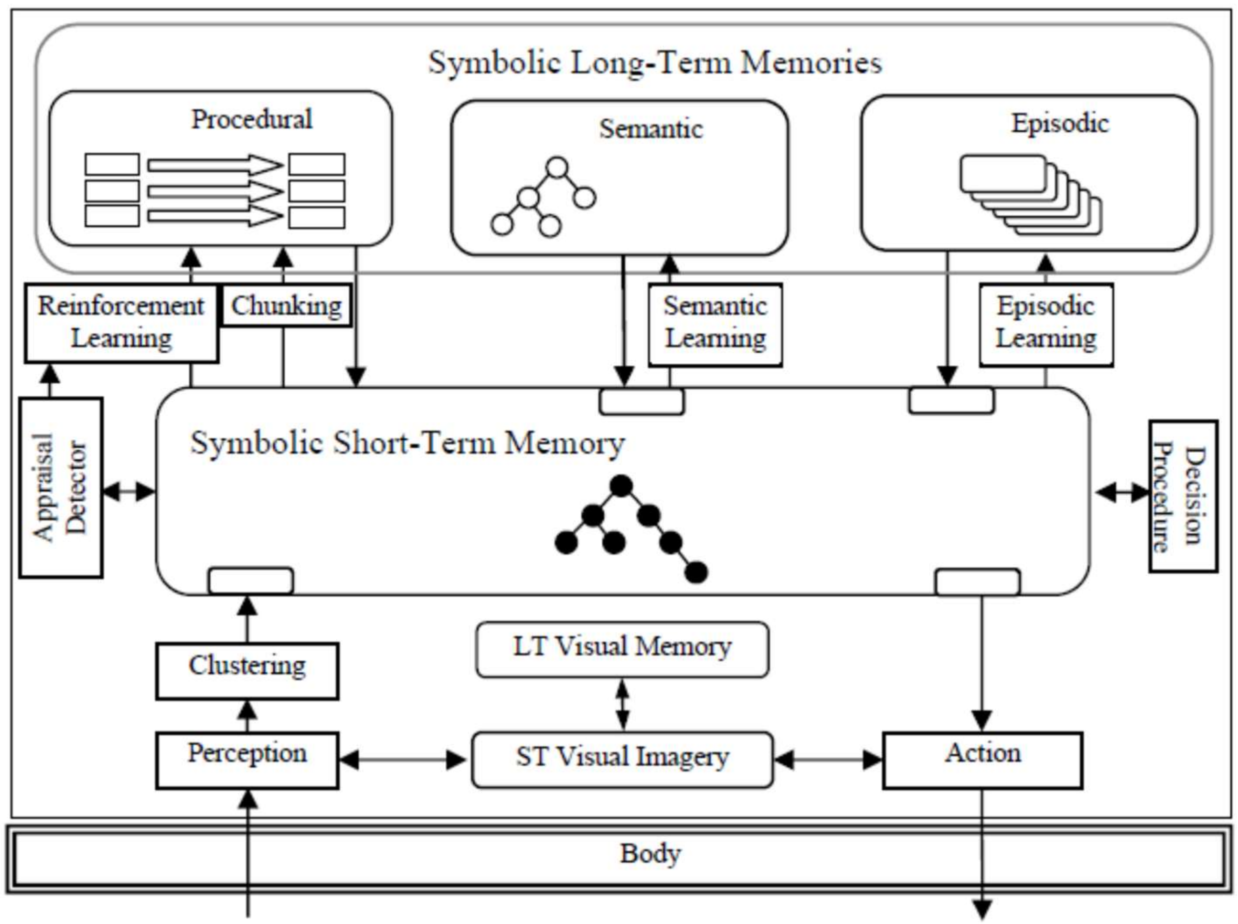


Problem Space Level

- Main cycle:
  - repeated *selection* and then *application* of one operator after another

- Impasse:
  - If something prevents that process from continuing (e.g., no operators to apply to *that* state, or no knowledge of how to choose) an ***impasse occurs***

Soar Architecture



Soar Cycle

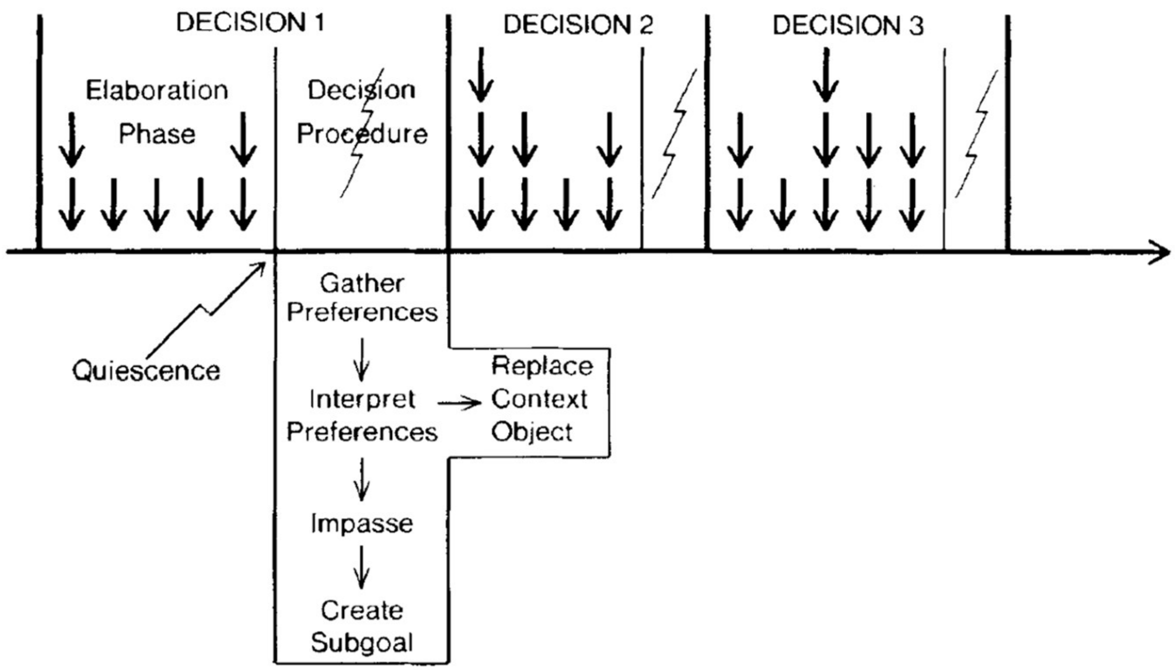
Soar main processing cycle:

- Interaction between Procedural Memory (knowledge about how to do things) and Working Memory (representation of the current situation):
    - WM is represented as a symbolic graph structure, rooted in a *state.*
    - PM is represented as if-then rules (sets of conditions and actions), which are continually matched against the contents of working memory,
- if the conditions of a rule matches the working memory it *fires* and performs its actions
- All rules match in parallel
- Operators are selected exploiting preferences
- Rules that match the operator changes the WM
- These changes induce other changes in the other modules

Soar Cycle

Soar main processing cycle:

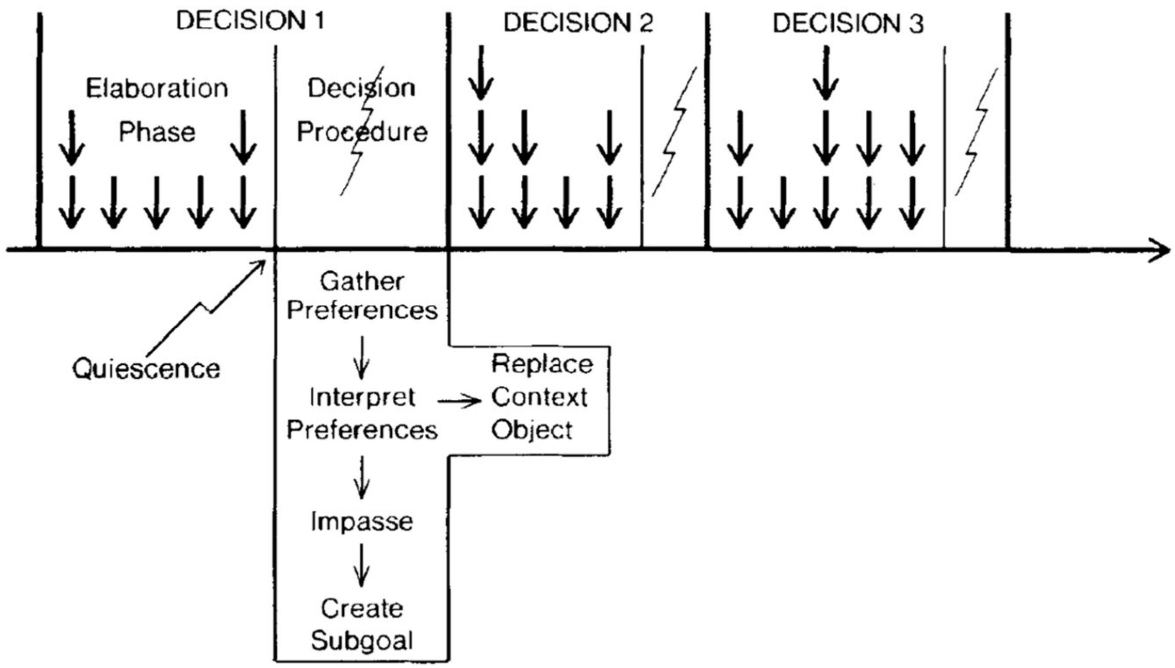
- During the elaboration phase, all *directly* available knowledge relevant to the current situation is brought to bear
- The contexts of the goal hierarchy and their augmentations serve as the working memory for these productions
- *preferences* can be created that specify the desirability of an object



Soar Cycle

Soar main processing cycle:

- *Soar* responds to an impasse by creating a subgoal (and an associated context)
- Once a subgoal is created, a problem space must be selected, along with an initial state and an operator
- the goals (contexts) in working memory are structured as a stack, referred to as the *context stack*



Structure of Soar

Soar can be layered into 3 levels :

1. Memory Level
2. Decision Level
3. Goal Level

Memory Level

A general intelligence requires a memory with a large capacity for the storage of knowledge.

A variety of types of knowledge must be stored, including :

- Declarative knowledge
- Procedural knowledge
- Episodic knowledge

Long-term Production Memory

All of Soar's long-term knowledge is stored in a single production memory.

Each production is a condition-action structure that performs its actions when its conditions are met.

Memory access consists of the execution of these productions.

During the execution of a production, variables in its actions are instantiated with value.

Long-term Production Memory

All of Soar's long-term knowledge is stored in a single production memory.

sp = Soar production

```
sp {water-jug*apply*fill
     (state <s> ^name water-jug
                ^operator <o>
                ^jug <j>)
     (<o> ^name fill
          ^fill-jug <j>)
     (<j> ^volume <volume>
          ^contents <contents>)
     -->
     (<j> ^contents <volume>)
     (<j> ^contents <contents> -)}
```

Working Memory

The result of memory access is the retrieval of information into a global Working Memory.

It is the temporary memory that contains all of Soar's short-term processing context. It has 3 components :

- The context stack specifies the hierarchy of active goals, problem spaces, states and operators

- Objects, such as goals and states (and their sub-objects)

- Preferences that encode the procedural search-control knowledge

Working Memory

The result of memory access is the retrieval of information into a global Working Memory.

(State space of Soar $\neq$ state space in which the problem lives!)
Working memory is where most of the action happens
Set of "augmentations" (key-value table/dict)

```
(s1 ^name water-jug
    ^jug  j1
    ^jug  j2 )

(j1 ^volume   5
    ^contents 0 )

(j2 ^volume   3
    ^contents 0 )
```

idle $\rightarrow$ state augmentation $\sim$ goal/activity $\rightarrow$ behavior

Preferences

- There is one special type of working memory structure - “the preference”

- Preferences encode control knowledge about the acceptability and desirability of actions:

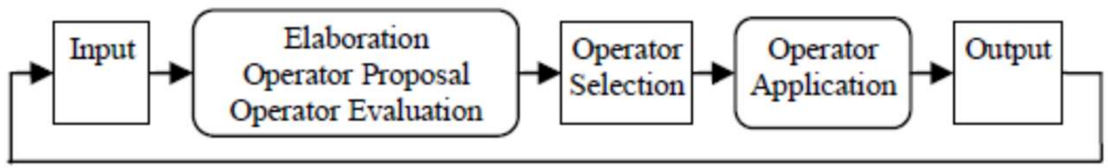
  - **Acceptability** preferences determine which actions should be considered as candidates

  - **Desirability** preferences define a partial ordering on the candidate actions.

Decision Level

Two phase decision cycle: elaboration and decision. The two phases are repeated until the goal of the current task is reached:

- A typical Soar decision cycle, takes much less than 50 milliseconds (humans' level, what humans expect), usually less than 1ms



- Elaboration phase:
  - all productions which match the current working memory fire. All productions fire in parallel.
  - The elaboration phase runs to quiescence (until no more productions fire).

- Decision phase:
  - examines any preferences put into preference memory (either in this phase, or previous ones), and chooses the next problem space, state, operator or goal to place in the context stack

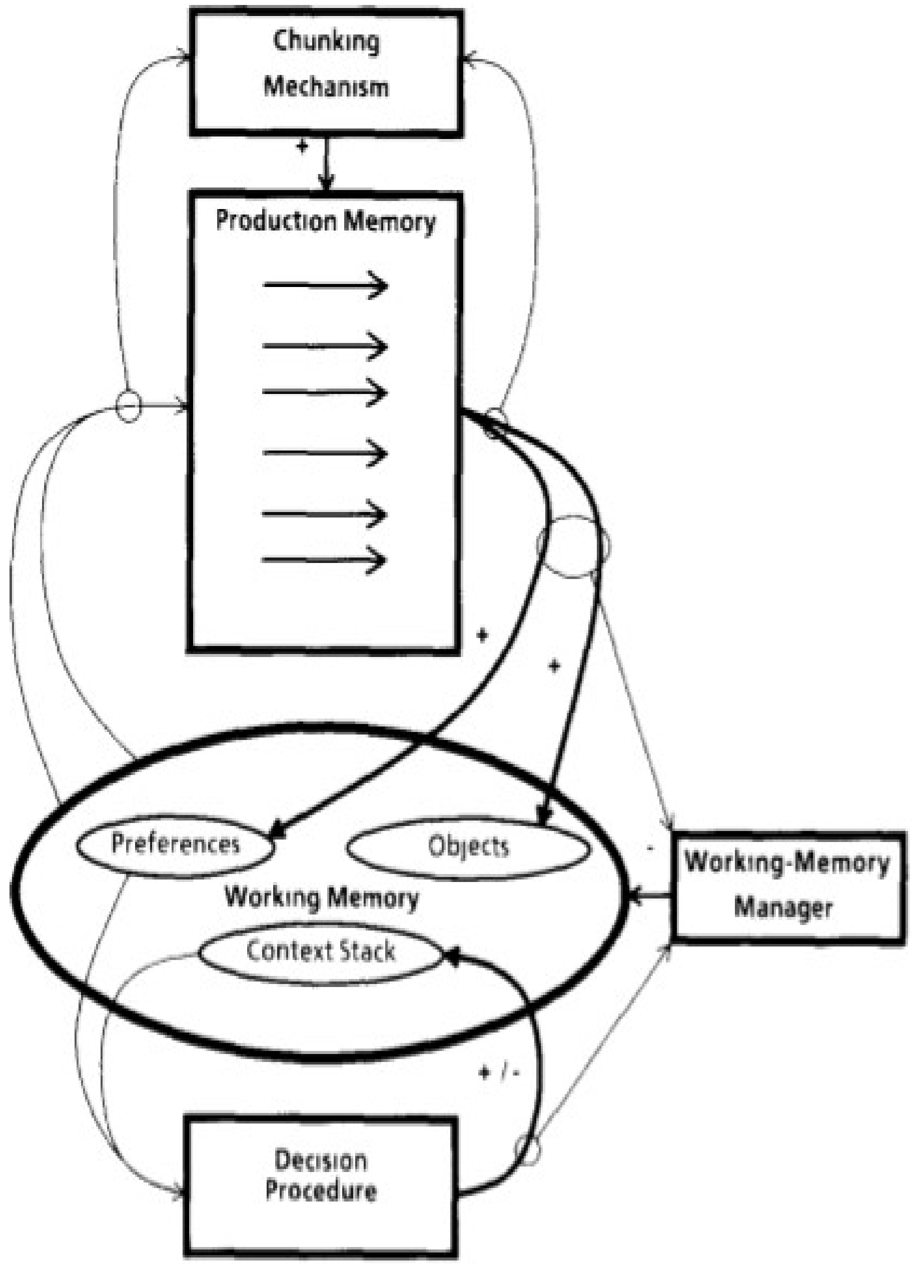


FIG 5 Architectural structure of SOAR

Decision Level

If there is not enough information (or contradictory) for the decision phase to choose the next value, then an impasse results.

- There are four types of impasses:
  - When two are more elements have equal preference, then there is a "tie impasse".
  - When no preferences are in working memory, this causes a "no-change impasse"
  - When the only preferences in working memory are rejected by other preferences, then there is a "reject impasse".
  - A "conflict impasse" results when preferences claim that two or more elements are each better choices then the others.
- When Soar reaches an impasse, it chooses a new problem space in an attempt to resolve the impasse.

Goal Level

- A general intelligence must be able to set and work towards goals. This level is based on the decision level.

- Goals are set whenever a decision cannot be made; that is, when the decision procedure reaches an impasse.

- Impasses occur when there are no alternatives that can be selected (no-change and rejection impasses) or when there are multiple alternatives that can be selected, but insufficient discriminating preferences exist to allow a choice to be made among them (tie and conflict impasses).

Impasse Resolution

- Whenever an impasse occurs, the architecture generates the goal of resolving the impasse which becomes the sub-goal.

- Along with this goal, a new performance context is created.

- The creation of a new context allows decisions to continue to be made in the service of achieving the goal of “resolving the impasse”.

- A stack of impasses is possible.

- The original goal is resumed after all the impasse stack is cleared.

Learning

- **Chunking**: new chunks to overcome impasses
- **Reinforcement Learning**: better operator selection
- **Episodic and Semantic Learning**: working memory re-organization

Learning via Chunking

- Learning occurs by the acquisition of chunks--productions that summarize the problem solving that occurs in subgoals, a mechanism called “Chunking”

- The actions of a chunk represent the knowledge generated during the sub-goal; that is, the results of the subgoal

- Three steps in chunk creation:
  - (1) the collection of conditions and actions,
  - (2) the variabilization of identifiers,
  - (3) chunk optimization

Learning via Chunking

- Learning occurs by the acquisition of chunks--productions that summarize the problem solving that occurs in subgoals, a mechanism called “Chunking”

- The actions of a chunk represent the knowledge generated during the sub-goal; that is, the results of the subgoal

- When Soar detects are useful sequence of firings, it creates chunks:
  - A chunk is essentially a large production that does the work of an entire sequence of smaller ones.
  - Chunks may be generalised before storing.

Soar 9

- Unifying Cognitive Functions and Emotional Appraisal

- The functional and computational role of emotion is open to debate.

- Appraisal theory is the idea that emotions are extracted from our evaluations (appraisals) of events that cause specific reactions in different people.

- The main controversy surrounding these theories argues that emotions cannot happen without physiological arousal.

Appraisal's Detector

This theory proposes that an agent continually evaluates a situation and that evaluation leads to emotion.

The evaluation is hypothesized to take place along multiple dimensions, such as

- goal relevance
- goal conduciveness
- causality and control

These dimensions are exactly what an intelligent agent needs to compute as it pursues its goals, while interacting with an environment.

Soar

- Non-symbolic processing

Non-symbolic Processing	Function
Numeric Preferences	Control operator selection
Reinforcement Learning	Learn operator selection control knowledge
Working memory activation	Aid long-term memory retrieval
Visual Imagery	Represent images and spatial data for reasoning
Appraisals: Emotions & Feelings	Summarize intrinsic value of situation – aid RL
Clustering	Create symbols that capture statistical regularities

- Memory & Learning

Memory/Learning System	Source of Knowledge	Representation of knowledge	Retrieval of knowledge
Chunking	Traces of rule firings in subgoals	Rules	Exact match of rule conditions, retrieve actions
Clustering	Perception	Classification networks	Winner take all
Semantic Memory	Working memory existence	Mirror of working memory object structures	Partial match, retrieve object
Episodic Memory	Working memory co-occurrence	Episodes: Snapshots of working memory	Partial match, retrieve episode
Reinforcement Learning	Reward and numeric preferences	Numeric preferences	Exact match of rule conditions, retrieve preference
Image Memory	Image short-term memory	Image	Deliberate recall based on symbolic referent

Soar Applications

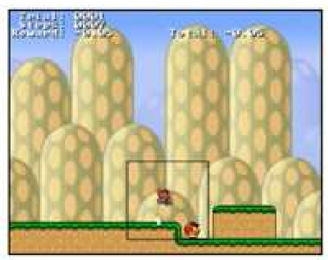
MOUTbot
Team Tactics & Unpredictable Behavior

SORTS
Spatial Reasoning & Real-time Strategy

Simulated Scout
Mental Imagery

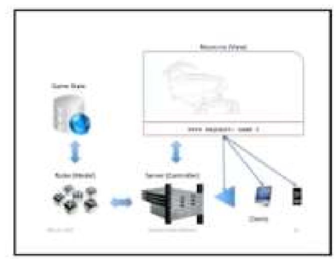
Splinter-Soar
Robot Control

ReLAI
Mental Imagery & Reinforcement Learning



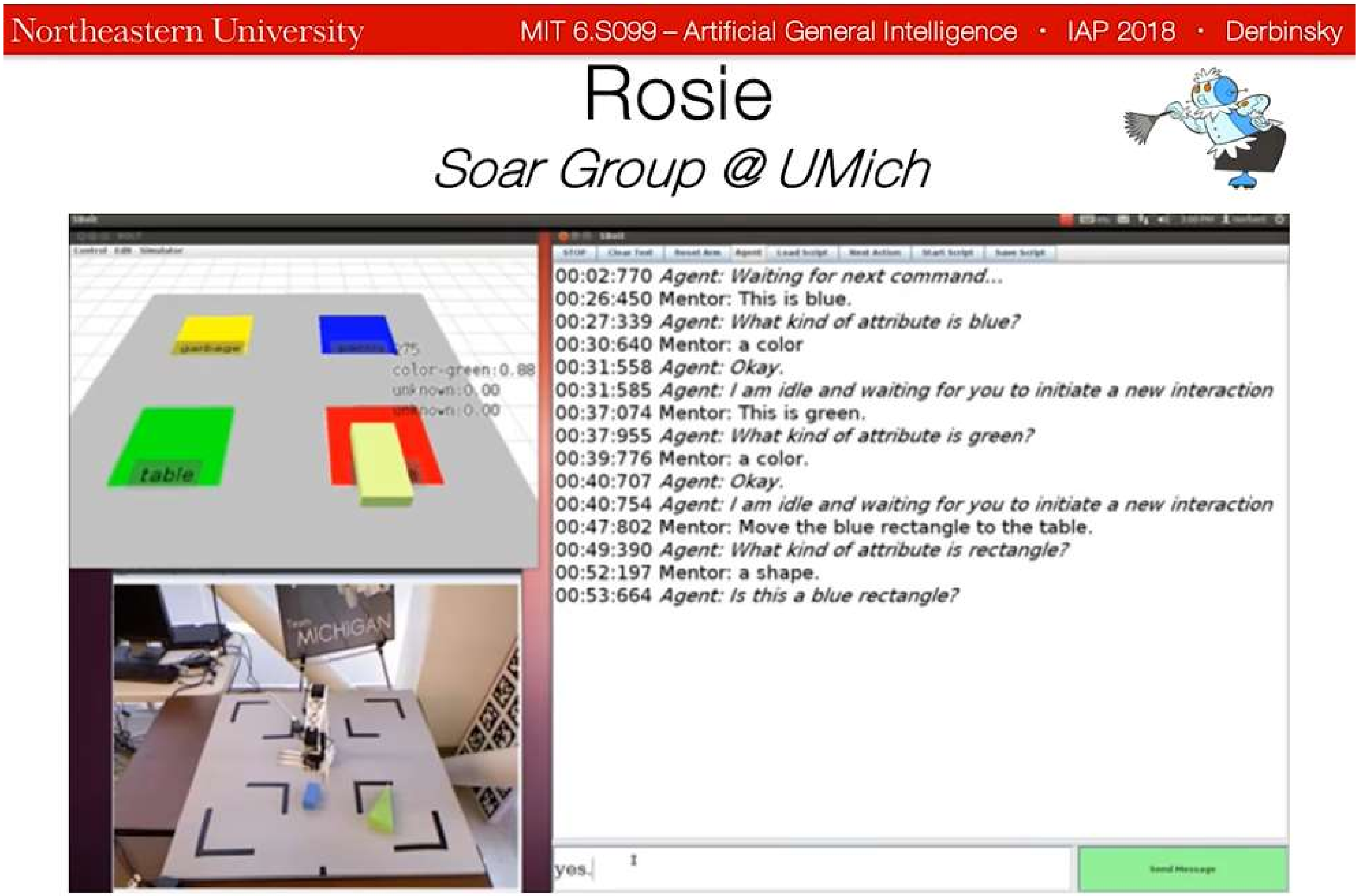
Infinite Mario
Hierarchical Reinforcement Learning

iSoar
Mobile Reinforcement Learning



RESTful Soar
Web-based Gameplay, Probabilistic Learning

Soar Applications



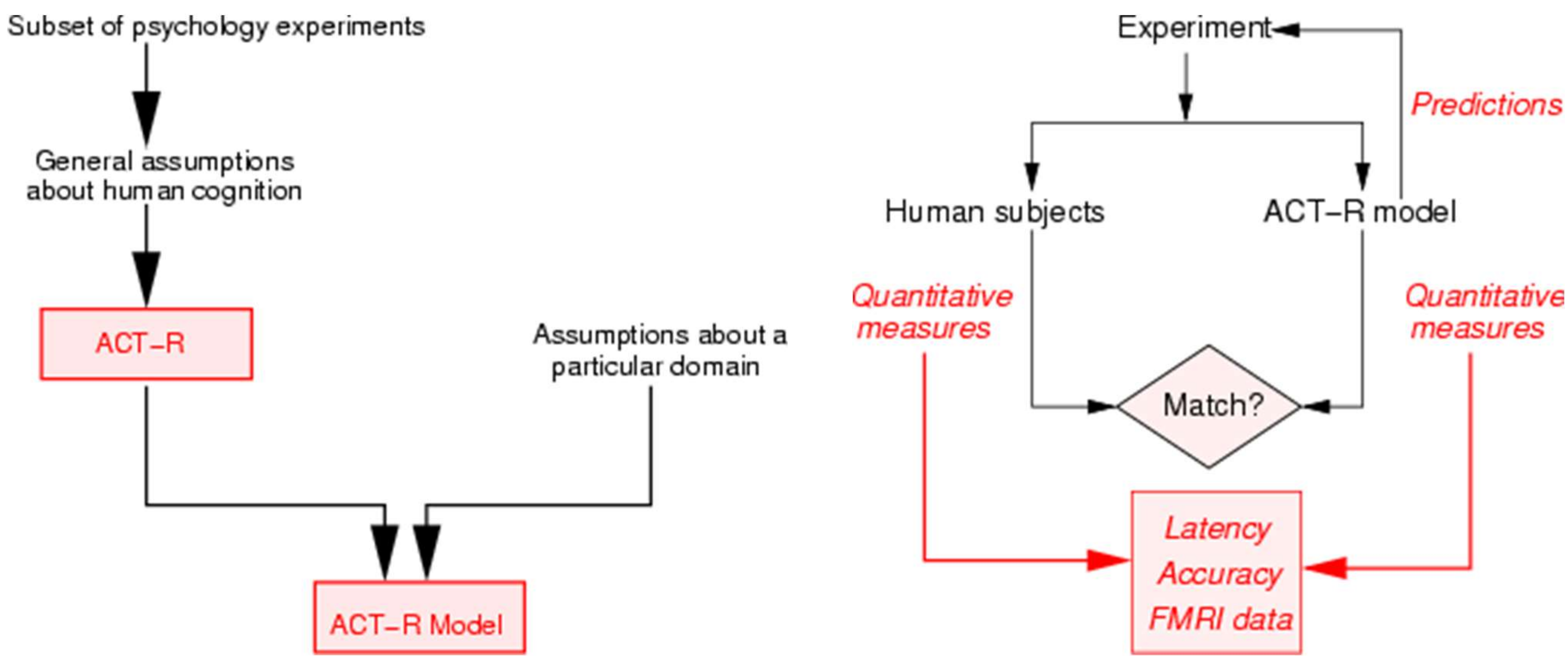
ACT-R
Adaptive Control of Thought-Rational

- ACT-R [Lebiere Anderson 93] is a cognitive architecture, a theory about how human cognition works
  - Looks like a (procedural) programming language.
  - Constructs based on assumptions about human cognitions
  - Cognitive Models
  - Psychological Plausibility
  - Hybrid Cognitive Architecture (symbolic and sub-symbolic)

ACT-R

- ACT-R is a framework
  - Researchers can create **models** that are written in ACT-R including
    - ACT-R's assumptions about cognition.
    - The researcher's assumptions about the task.
  - The assumptions are tested against data.
    - Reaction time
    - Accuracy
    - Neurological data (fMRI)

ACT-R


Subset of psychology experiments
General assumptions about human cognition
ACT–R
Assumptions about a particular domain
ACT–R Model
Experiment
Predictions
Human subjects
ACT–R model
Quantitative measures
Quantitative measures
Match?
Latency
Accuracy
FMRI data

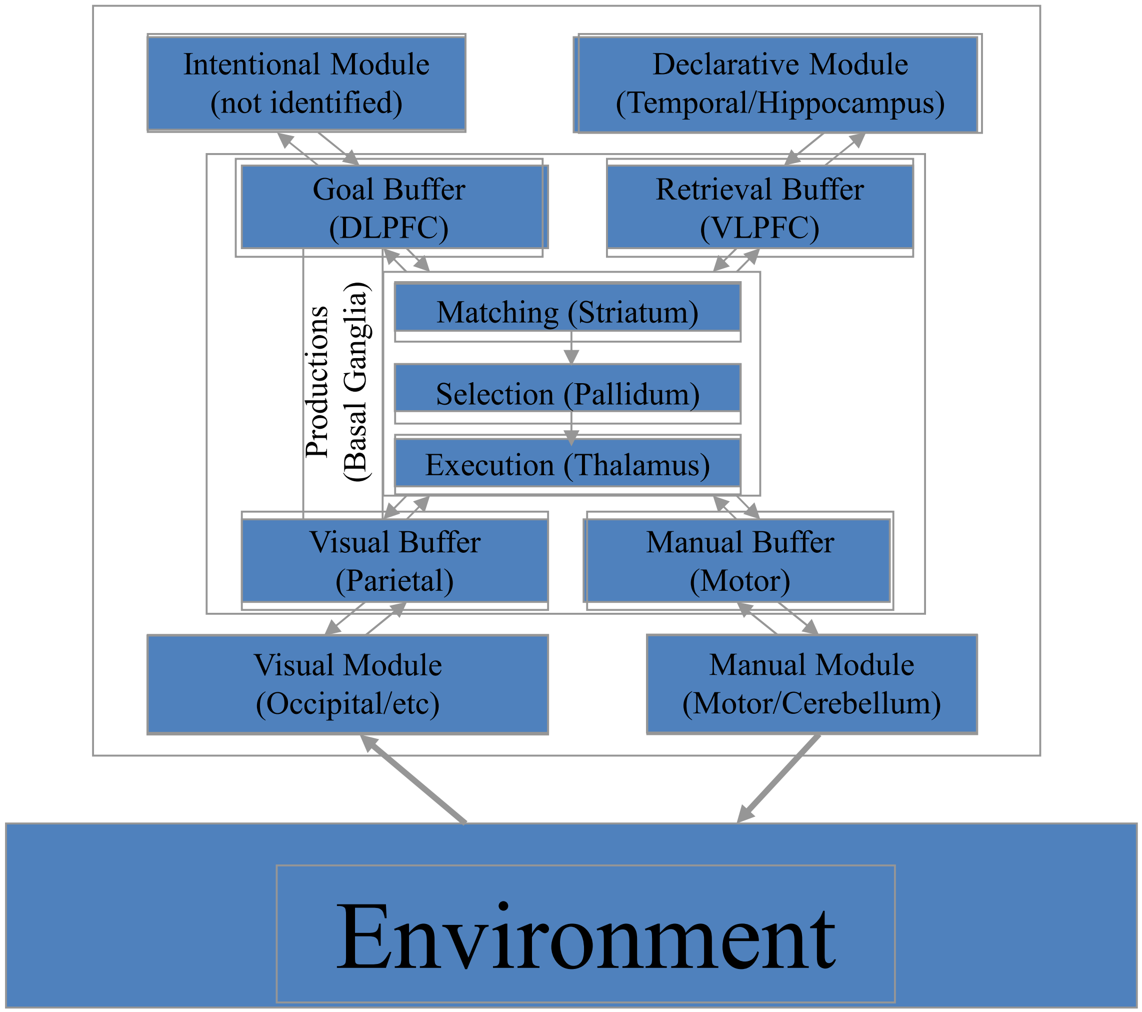
ACT-R

- ACT-R is an integrated cognitive architecture.
  - Brings together not just different aspects of cognition, but of
    - Cognition
    - Perception
    - Action
  - Runs in real time.
  - Learns.
  - Robust behavior in the face of error, the unexpected, and the unknown.

Overview of ACT-R

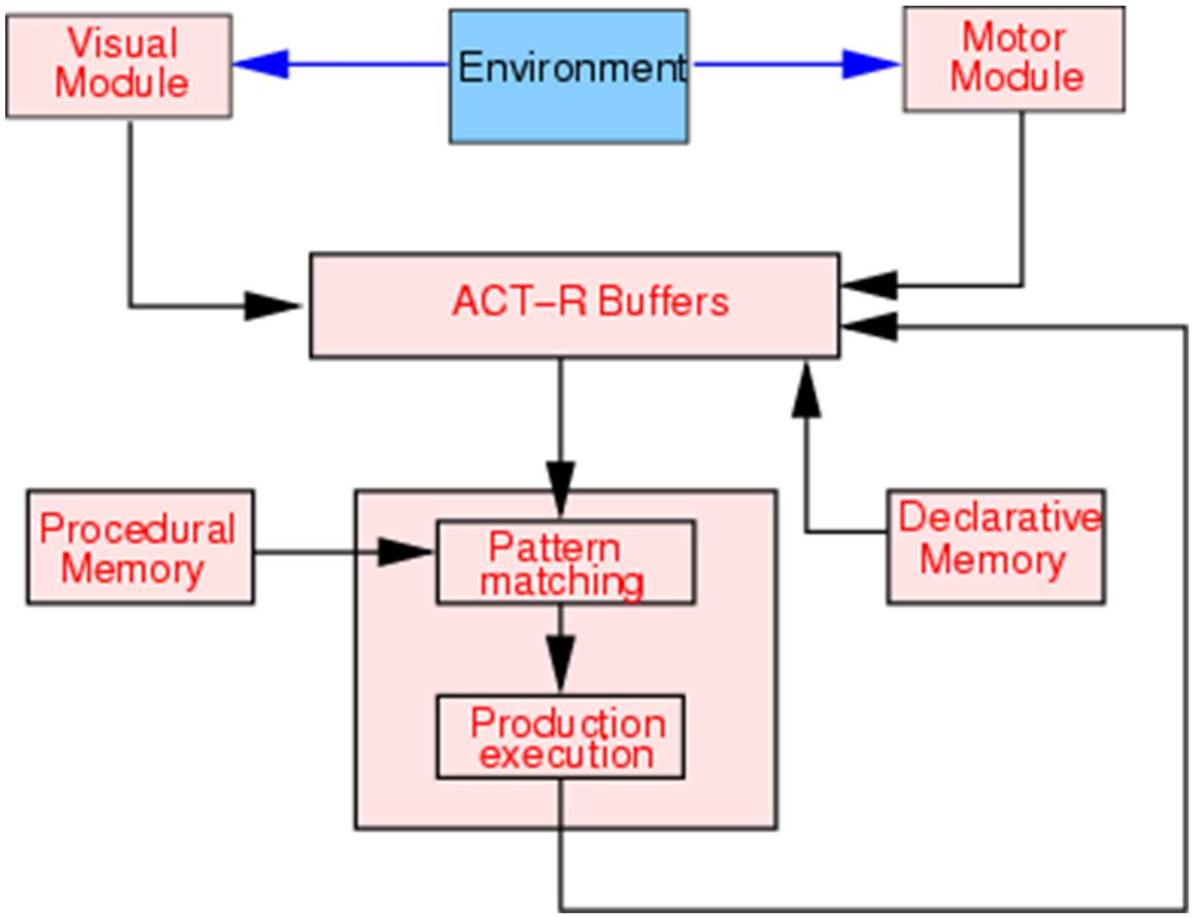
- ACT-R is made up of
  - Modules:
    - Perceptual/motor
    - Memory:
      - Declarative: facts
      - Procedural: productions
  - Buffers
  - A sub-symbolic level

ACT-R: Architecture



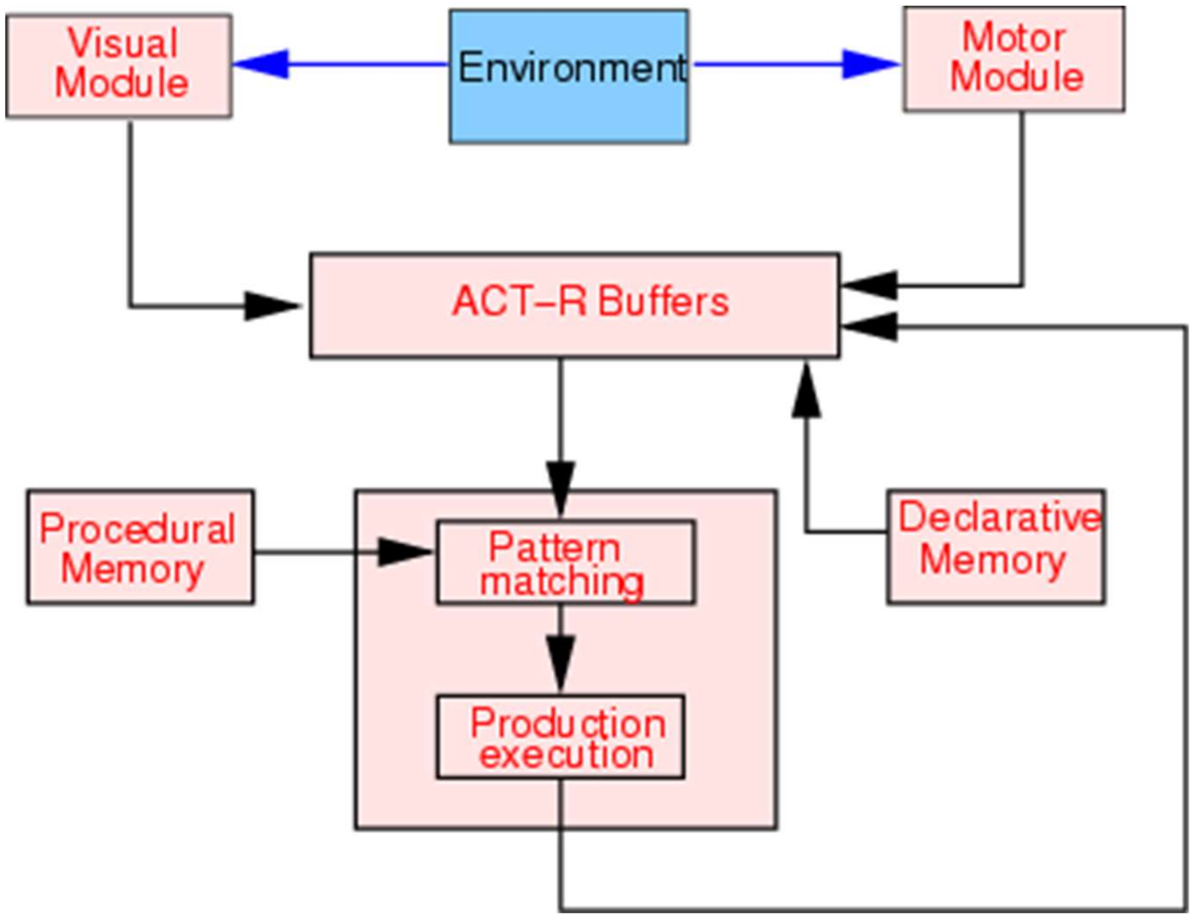
ACT-R: Cycle

ACT-R accesses its modules (except for the procedural-memory module) through buffers. For each module, a dedicated buffer serves as the interface with that module. The contents of the buffers at a given moment in time represents the state of ACT-R at that moment.



ACT-R: Cycle

At each cycle period, a pattern matcher searches for a production that matches the current state of the buffers. Only one such production is executed at a given cycle. A production that fires can modify the buffers changing the state of the system



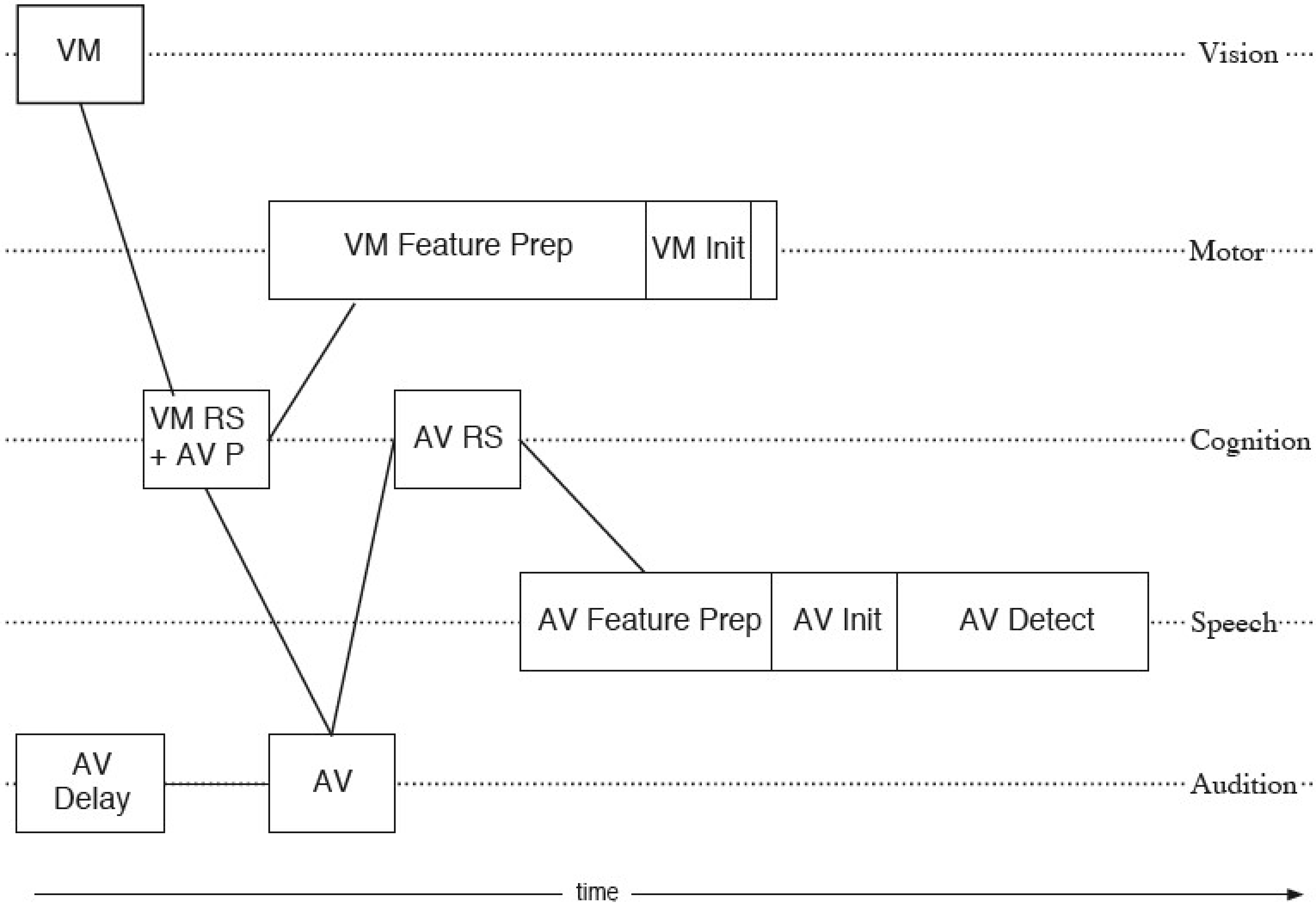
Perceptual-Motor Modules

- Takes care of the interface with the “real” world:
  - Visual module
  - Auditory module
  - Motor module
  - etc

Perceptual-Motor Modules

- 3 tones: low, med, high
  - 445ms
- 3 positions: left, middle, right
  - 279ms
- Tones and positions
  - 456ms
  - 283ms

Perceptual-Motor Modules



Declarative Module

- Declarative memory:
  - Facts
    - Washington, D.C. is the capital of the U.S.
    - 2+3=5.
  - Knowledge a person might be expected to have to solve a problem
  - Called chunks

Declarative Module

(CHUNK-TYPE NAME SLOT1 SLOT2 SLOTN)

(b

isa count-order

first 1

second 2

)

Procedural Module

- Procedural memory:
  - Knowledge about how to do something:
    - How to type the letter “Q”
    - How to drive
    - How to perform addition

Procedural Module

- Made of condition-action data structures called production rules
- Each production rule takes 50ms to fire
- Serial bottleneck in this parallel system

	(p name
condition part	Specification of Buffer Tests
delimiter	==>
action part	Specification of Buffer Transformations
	)

Procedural Module

```
( p     example-counting
        =goal>
           isa count
           state counting
           number =num1
        =retrieval>
           isa count-order
           first =num1
           second =num2
  ==>
        =goal>
           number =num2
        +retrieval>
           isa count-order
           first =num2
 )
```

IF the goal is
    to count
    the current state is counting
    there is a number called =num1
    and a chunk has been retrieved
    of type count-order
    where the first number is =num1
    and it is followed by =num2
THEN
    change the goal
    to continue counting from =num2
    and request a retrieval
    of a count-order fact
    for the number that follows =num2

Buffers

- The procedural module accesses the other modules through buffers
- For each module (visual, declarative, etc.), a dedicated buffer serves as the interface with that module
- The contents of the buffers at any given time represent the state of ACT-R at that time

Buffers

1. Goal Buffer (=goal, +goal)
   -represents where one is in the task
   -preserves information across production cycles

2. Retrieval Buffer (=retrieval, +retrieval)
   -holds information retrieval from declarative memory
   -seat of activation computations

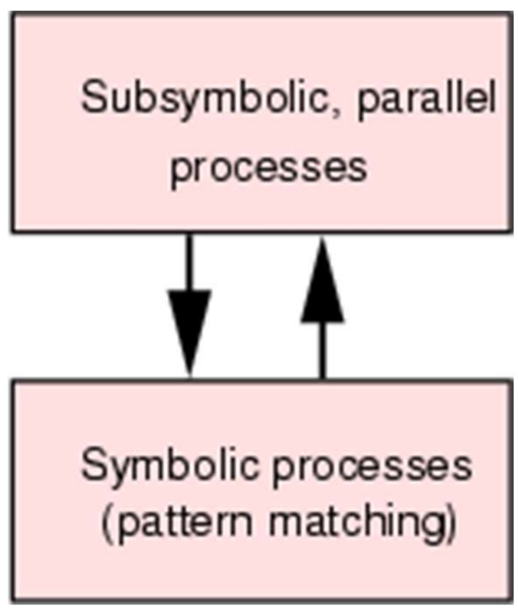
3. Visual Buffers
   -location (=visual-location, +visual-location)
   -visual objects (=visual, +visual)
   -attention switch corresponds to buffer transformation

4. Auditory Buffers (=aural, +aural)
   -analogous to visual

5. Manual Buffers (=manual, +manual)
   -elaborate theory of manual movement include feature preparation, Fitts law, and device properties

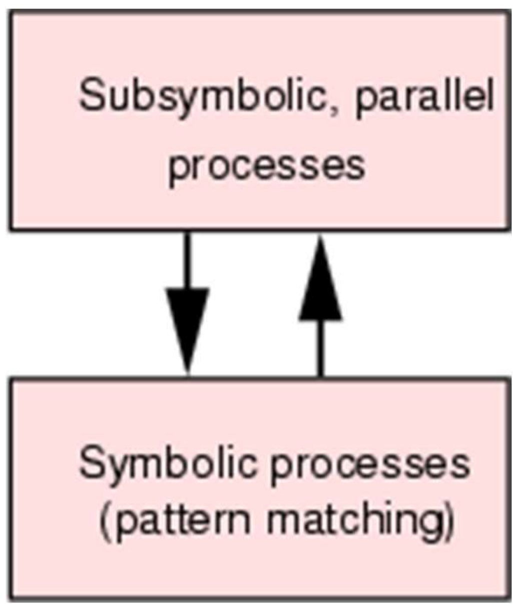
Sub-symbolic Level

- The production system is symbolic.
- The sub-symbolic structure is a set of parallel processes that can be summarized by a number of mathematical equations.
- The sub-symbolic equations control many of the symbolic processes.



Sub-symbolic Level

- Sub-symbolic equations control many symbolic processes
- If several productions match the state of the buffers, a sub-symbolic utility equation estimates the relative cost and benefit associated with each production and decides to select for the production with the highest utility
- Facts retrieved from declarative memory depend on sub-symbolic retrieval equations, which take into account context and history of usage of that fact
- Sub-symbolic mechanisms are also responsible for most learning processes



Production Utility

- When several productions match the state of the buffers:
  - a sub-symbolic utility equation estimates the relative cost and benefit associated with each production and
  - selects the production with the highest utility

Production Utility

Expected Gain = E = PG - C

P expected probability of success
G value of goal
C expected cost

Probability of choosing i = $\dfrac{e^{E_i/t}}{\sum_j e^{E_j/t}}$

t noise in evaluation (temperature in the Bolztman equation)

$$P = \frac{\text{Successes}}{\text{Successes + Failures}}$$

Successes = α + m
Failures = β + n

α prior successes
m experienced successes
β prior failures
n experienced failures

Retrieved Facts

- Whether and how fast a chunk can be retrieved from declarative memory:
  - depends on the sub-symbolic retrieval equations, which take into account the context and the history of usage of that fact
- <u>Chunk activations</u>:
  - The activation of a chunk is a sum of base-level activation, reflecting its general usefulness in the past, and an associative activation, reflecting it's relevance in the current context

Chunk Activation

Activation of Chunk i

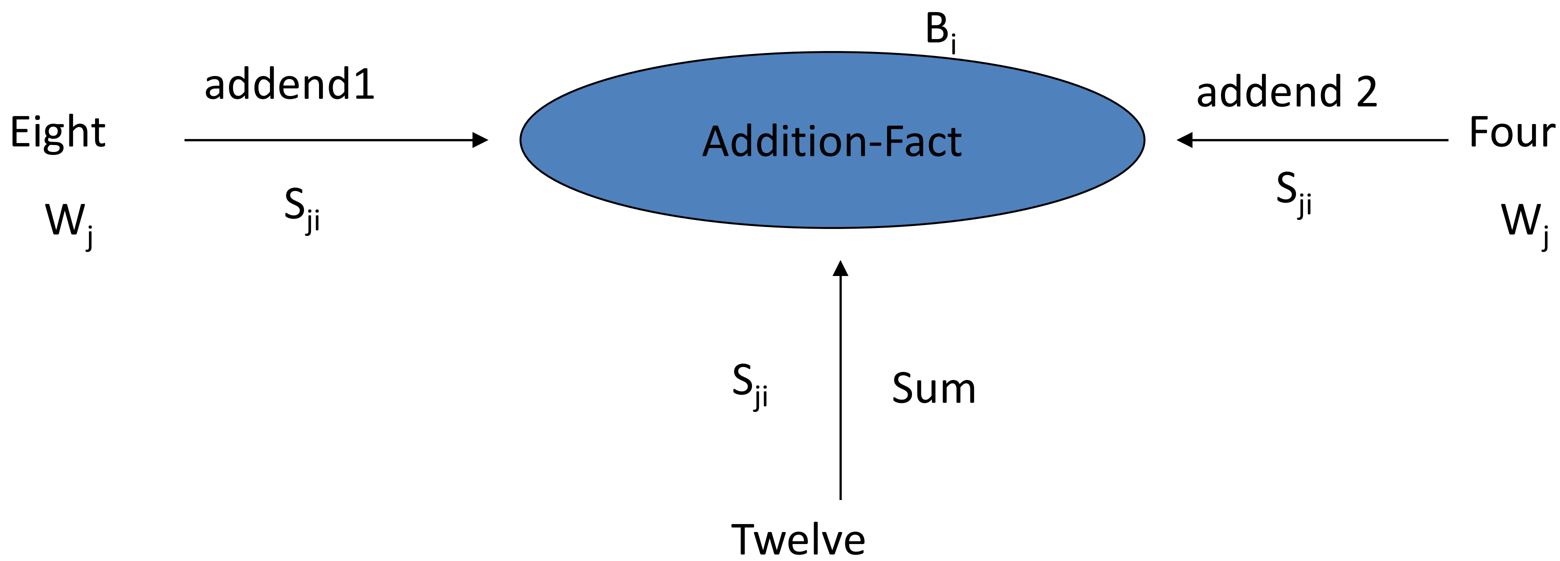
Attentional weighting of
Element j of Chunk i

$$A_i = B_i + \sum_j W_j S_{ji}$$

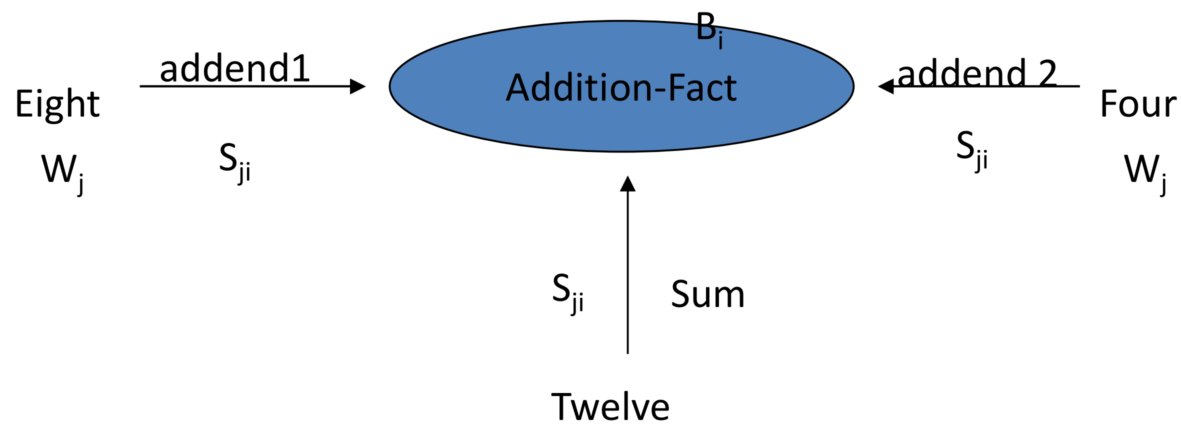
Base-level activation
(Higher if used recently)

Strength of association
of Element j to Chunk i

Chunk Activation



Chunk Activation



W_j decreases with the number of elements associated with Chunk i.

S_{ji} decreases with the number of chunks associated with the element.

Probability/Time Retrieval

- The <u>probability of retrieving</u> a chunk is given by:

$$P_i = 1 / (1 + \exp(-(A_i - \tau)/s))$$

Here τ is the activation threshold, s controls the sensitivity of recall to changes in activation

- The <u>time to retrieve</u> a chunk is given by

$$T_i = F \exp(-A_i)$$

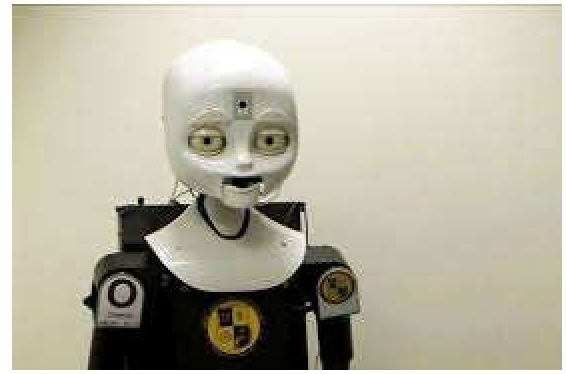
F: The latency factor parameter

Sub-symbolic Level

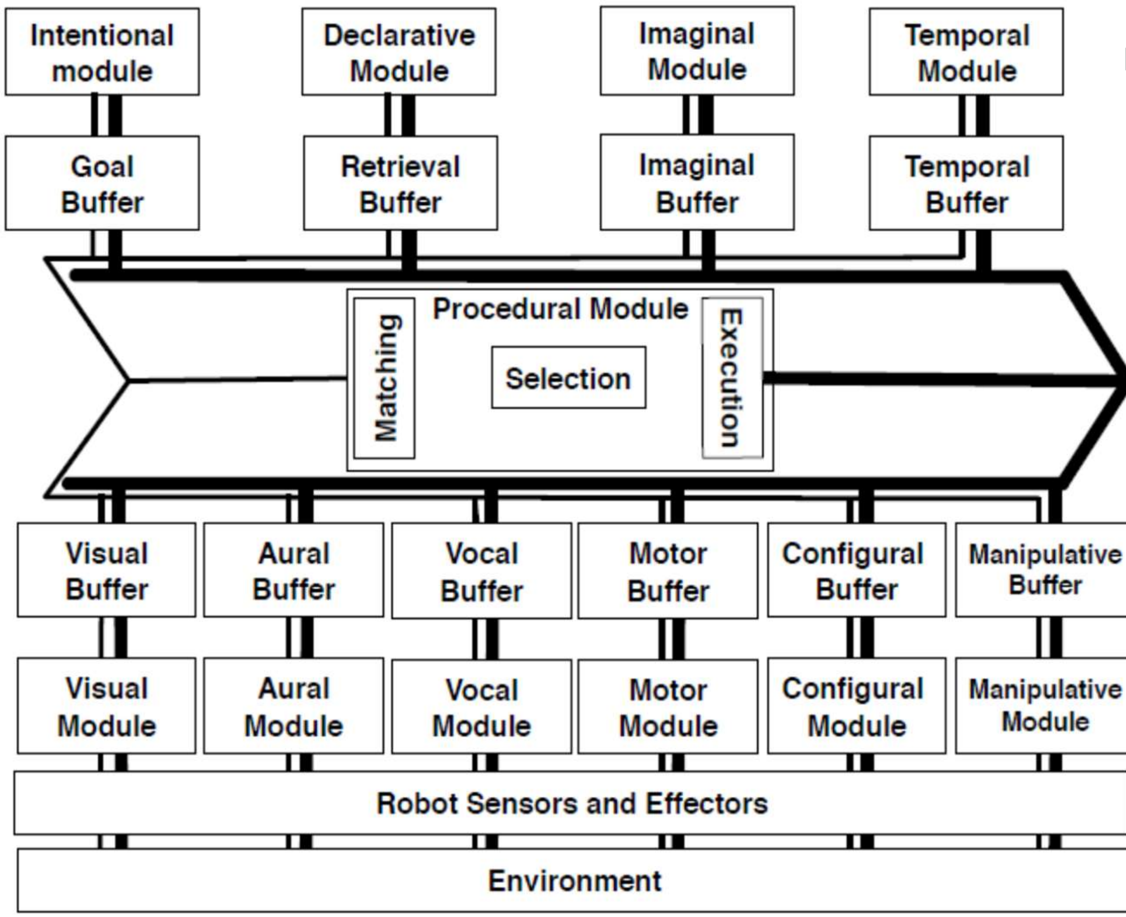
- The equations that make up the sub-symbolic level are not static and change with experience.
- The sub-symbolic learning allows the system to adapt to the statistical structure of the environment.

ACT-R/E

- Embodied: spatial reasoning



Octavia (MDS Platform MIT) at Navy Center for Applied Research in Artificial Intelligence




```
Production FIND-NEXT-ROOM
conditions:
=goal>       isa patrol-goal
             slot cur-room: A
             slot next-room: B
=retrieval>  isa room
             slot name: A
             slot location: building-1
actions:
+retrieval>  isa room
             slot name: B
             slot location: ?
```

$$P(m) = \frac{e^{E_m/t}}{\sum_n e^{E_n/t}}$$

$$P(i) = \frac{e^{A_i/t}}{\sum_j e^{A_j/t}}$$

ACT-R/E

- HRI tasks

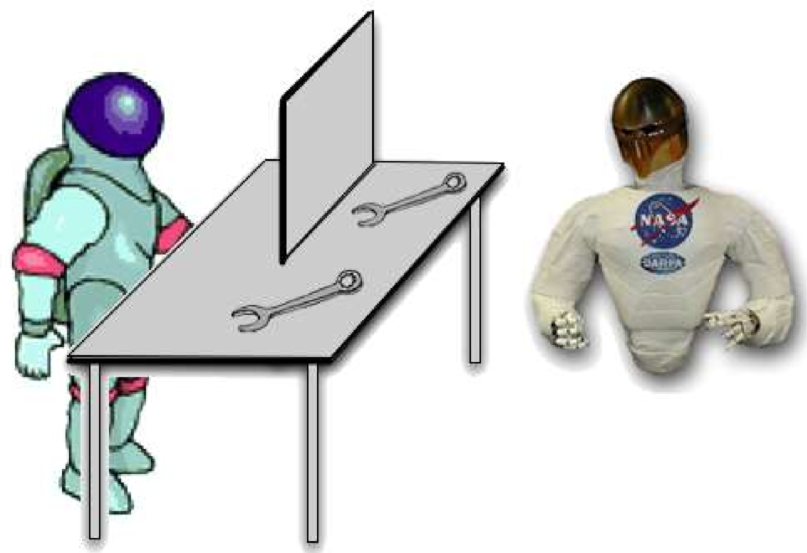
Task	Components of ACT-R/E	Dataset
Gaze following	Manipulative module Configural module Utility learning	Corkum & Moore (1998) Moll & Tomasello (2006)
Hide and seek	Imaginal module Visual module Vocal module	Trafton, Schultz, Perzanowski, et al. (2006)
Interruption and resumption	Declarative module Intentional module Imaginal module Procedural module	Trafton et al. (2012)
Theory of mind	Declarative module Architecture as a whole	Leslie, German, & Polizzi (2005) Wellman, Cross, & Watson (2001)

(1) test and evaluate each component separately, to validate it against human subject data;
(2) test different sets of the components as they interact;
(3) show how models increase the ability, breadth, and parsimony of cognitive models.

Perspective Taking

- Perspective taking is critical for collaboration.
- How do we model it? (ACT-R, Polyscheme…)
- Scenario:

"Please hand me the wrench"



Perspective Taking and Changing Frames of Reference

Bob, if you come straight down from where you are, uh, and uh kind of peek down under the rail on the nadir side, by your right hand, almost straight nadir, you should see the uh…

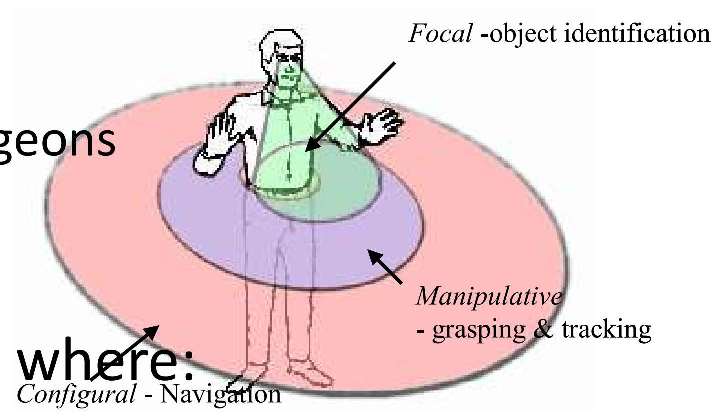
- Notice the mixing of perspectives: exocentric (down), object-centered (down under the rail), addressee-centered (right hand), and exocentric again (nadir) all in one instruction!
- Notice the “new” term developed collaboratively: mystery hand rail

Perspective taking in human interactions

- How do people usually resolve ambiguous references that involve different spatial perspectives? (Clark, 96)
  - Principle of least effort (which implies least joint effort)
    - All things being equal, agents try to minimize their effort
  - Principle of joint salience
    - The ideal solution to a coordination problem among two or more agents is the solution that is the most salient, prominent, or conspicuous with respect to their current common ground.
    - In less simple contexts, agents may have to work harder to resolve ambiguous references

Perspective Taking

- ACT-R/S (Schunn & Harrison, 2001)
  - Perspective-taking system using ACT-R/S is described in Hiatt et al. 2003
    - Three Integrated Visuo Spatial buffers
    - Focal: Object ID; non-metric geon parts
    - Manipulative: grasping/tracking; metric geons
    - Configural: navigation; bounding boxes



- Polyscheme (Cassimatis)
  - Computational Cognitive Architecture where:
    - Mental Simulation is the primitive
    - Many AI methods are integrated
  - Perspective-taking using Polyscheme is described in Trafton et al., 2005

Robot Perspective Taking

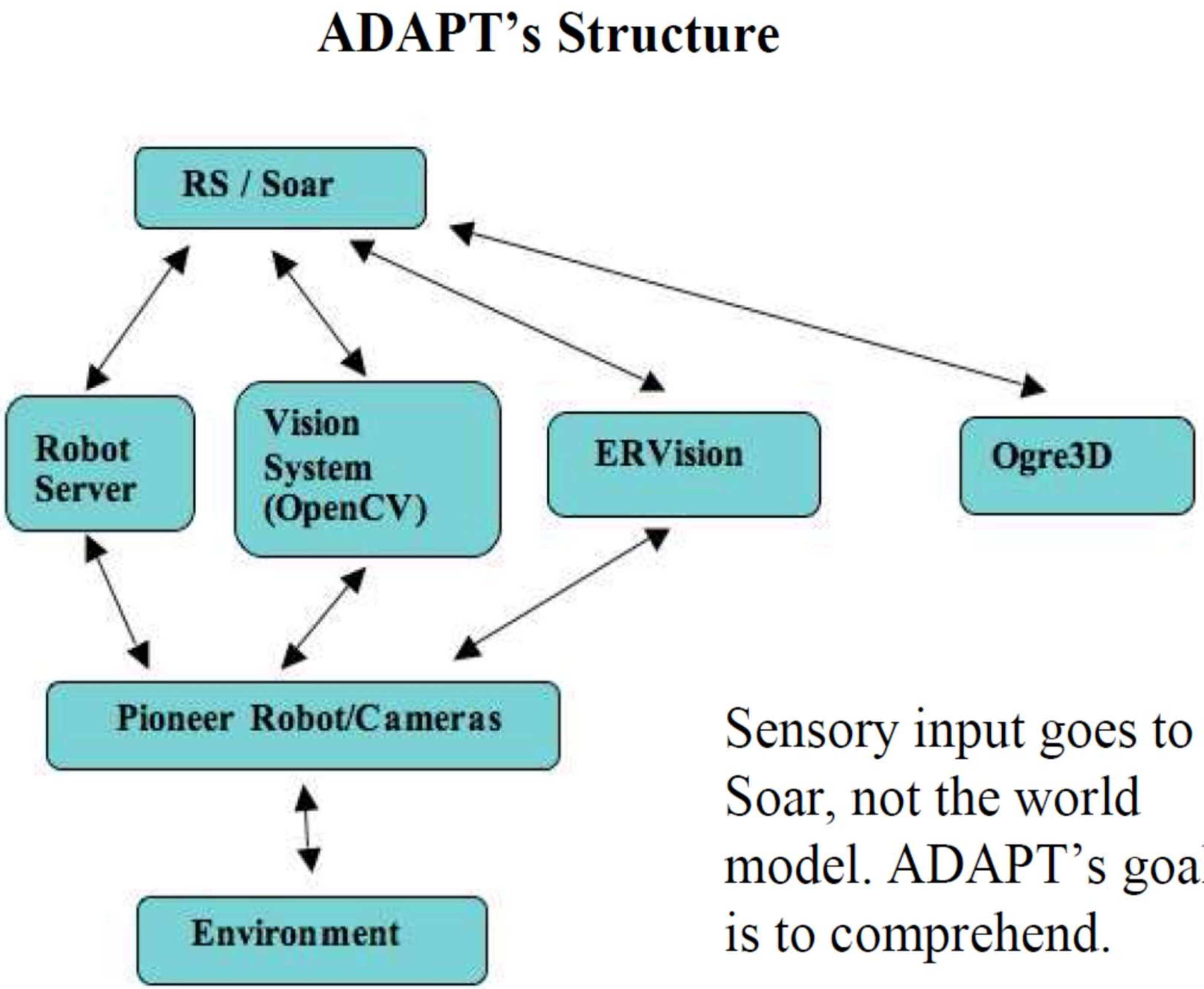
Human can see one cone
Robot can sense two cones

(Fong et al., 06)

ADAPT

- ADAPT is an architecture for Robots that integrates features of Soar and ACT-R
- Aims:
  - Embodied agent (robot) that implements sophisticated behaviors managing vision, natural language, problem solving, and learning
  - Two principles:
    - Active perception: perception is context-related and goal-oriented, therefore enhanced perception of related input
    - Real-time reasoning about parallel processes and multiple actions

ADAPT



Sensory input goes to Soar, not the world model. ADAPT's goal is to comprehend.

ADAPT vs. Soar and ACT-R

Soar has a single buffer for each goal, hence a single operator is selected

ACT-R allows one firing for each cycle, depending on the context

Both Soar and ACT-R impose a bottleneck to parallel processing

ADAPT vs. Soar and ACT-R

ADAPT continuously updates the WM with respect to the rules as in Soar

Schemata are stored in LTM, as in ACT-R

Schema similar to operator of Soar or chunk in ACT-R:

- Schema theory representation (perception and action schema)
- Integrates procedural and declarative knowledge (reasoning about plans)

ADAPT

The RS (Robot Schemas) language is the basis of the robotics capabilities of ADAPT. RS is precise and mature.

RS is a CSP-type programming language for robotics, that controls a hierarchy of concurrently executing schemas.

$$\mathrm{Joint}_i(s)() = [\mathrm{Jpos}_i()(x),\ \mathrm{Jset}_i(s, x)(u),\ \mathrm{Jmot}_i(u)()\ ]^{c0}$$

c0: $(\mathrm{Jpos}_i, x)\ (\mathrm{Jset}, x)$ $\qquad$ $(\mathrm{Jset}, u)\ (\mathrm{Jmot}_i, u)$

$\mathrm{Jpos}_i()(x)$ continuously reports the position of joint i on port x
$\mathrm{Jmot}_i(u)()$ accepts a signal on port u and applies it to the actuator of joint i
$\mathrm{Jset}_i(s, x)(u)$ accepts a setpoint on port s and iteratively inputs a joint position on port x and outputs a motor signal on port u to drive the joint position to the setpoint

ADAPT

$P = (Q, L, X\ \delta, \beta, \tau)$ where

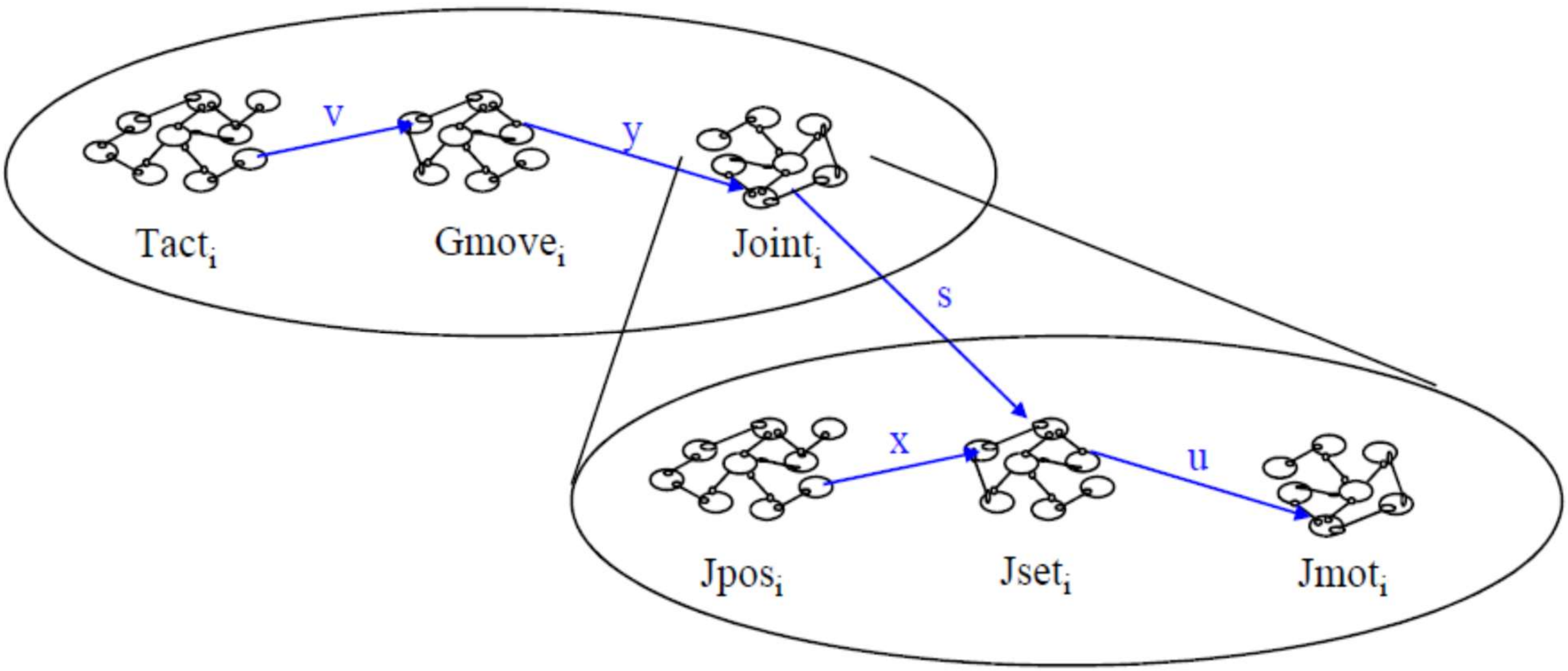
Q is the set of states
L is the set of ports
$X = (X_i \mid i \in L)$ is the event alphabet for each port

$XL = \{ (i, X_i) \mid i \in L \}$ i.e., a disjoint union of L and X

$\delta : Q \times XL \rightarrow 2^Q$ is the transition function
$\beta = (\beta_i \mid i \in L)\ \beta_i : Q \rightarrow X_i$ is the output map for port i
$\tau \in 2^Q$ is the set of start states

ADAPT

The behavior of every RS schema is defined using port automata. This provides precision to the semantics and also a constructive means of reasoning about the behavior and meaning of schemas.



ADAPT

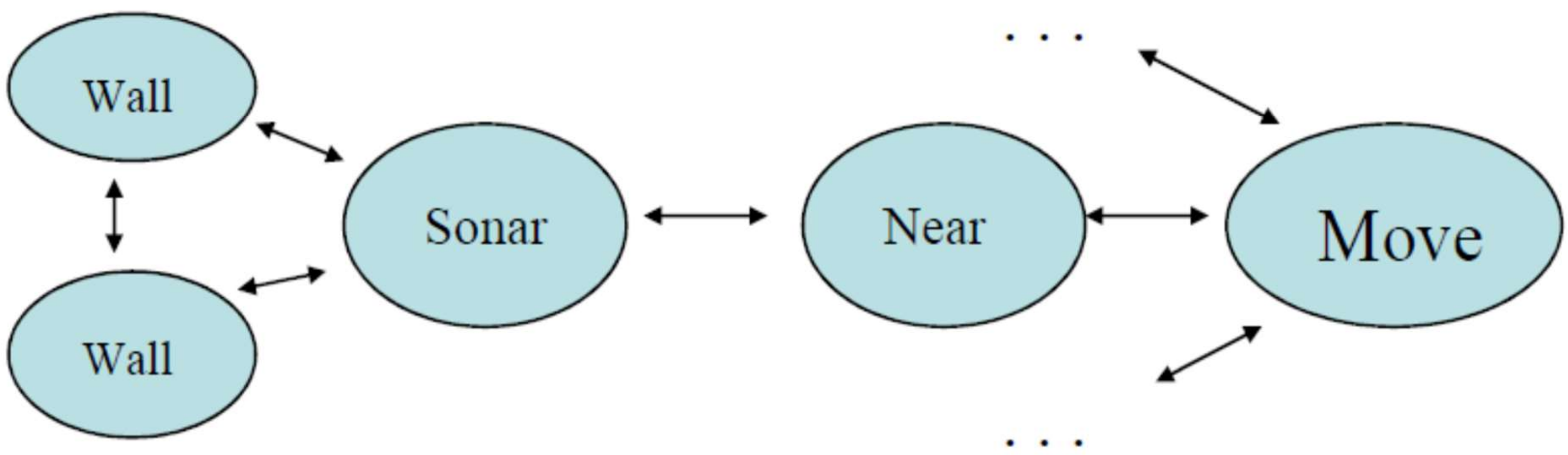
1. Sequential Composition: $\mathtt{T} = \mathtt{P};\mathtt{Q}$. The process $\mathtt{T}$ behaves like the process $\mathtt{P}$ until that terminates, and then behaves like the process $\mathtt{Q}$ (regardless of $\mathtt{P}$'s termination status).
2. Concurrent Composition: $\mathtt{T} = (\mathtt{P} \mid \mathtt{Q})^c$. The process $\mathtt{T}$ behaves like $\mathtt{P}$ and $\mathtt{Q}$ running in parallel and with the input ports of one connected to the output ports of the other as indicated by the port-to-port connection map c. This can also be written as $\mathtt{T} = (\big|_{i \in I} \mathtt{P}i)^c$ for a set of processes indexed by I.
3. Conditional Composition: $\mathtt{T} = \mathtt{P}\langle v \rangle : \mathtt{Q}_v$. The process $\mathtt{T}$ behaves like the process $\mathtt{P}$ until that terminates. If $\mathtt{P}$ aborts, then $\mathtt{T}$ aborts. If $\mathtt{P}$ terminates normally, then the value v calculated by $\mathtt{P}$ is used to intialize the process $\mathtt{Q}$, and $\mathtt{T}$ then behaves like $\mathtt{Q}_v$.
4. Disabling Composition: $\mathtt{T} = \mathtt{P}\#\mathtt{Q}$. The process $\mathtt{T}$ behaves like the concurrent composition of $\mathtt{P}$ and $\mathtt{Q}$ until either terminates, then the other is aborted and $\mathtt{T}$ terminates. At most one process can stop; the remainder are aborted.
5. Synchronous Recurrent Composition: $\mathtt{T} = \mathtt{P}\langle v \rangle :; \mathtt{Q}_v$. This is a recursively defined as follows: $\mathtt{P} :; \mathtt{Q} = \mathtt{P} : (\mathtt{Q}; \mathtt{P} :; \mathtt{Q})$.
6. Asynchronous Recurrent Composition: $\mathtt{T} = \mathtt{P}\langle v \rangle :: \mathtt{Q}_v$. This is recursively defined as follows: $\mathtt{P} :: \mathtt{Q} = \mathtt{P} : (\mathtt{Q} \mid (\mathtt{P} :: \mathtt{Q}))$.

Operator Precedence: The operator precedence from loosest to tightest is as follows: Concurrent; Disabling; Sequential; Conditional; Synchronous Recurrent; Asynchronous Recurrent.

ADAPT

Schemas, facts, and hypotheses are nodes in a graph.
Links implement the composition operations, as well as otl
relations, including deductive and evidential inference.

Automata that implement a schema are built as needed.



ADAPT

ADAPT plans by transforming a hierarchy of schemas

At each step, ADAPT can perform one of the following steps:

- refine a schema into subschemas,
- instantiate variables in a schema (this includes connecting two or more schemas by binding their variables together),
- start execution of a schema, suspend execution of a schema, or terminate and remove a schema.

ADAPT operators work at the executive level rather than at the task level, continually modifying the schema hierarchy.

Task-level actions are executed by the motor parts of the schemas

ADAPT

The basic loop of ADAPT is:

1 - check Soar's output link to see if there are any commands, which may be either motion commands for the robot or modeling commands for the World Model,

2 - blend the motion commands that are to be sent to the robot,

3 - send all robot commands both to the robot and to the virtual robot in the World Model,

4 - send all other commands to the World Model,

5 – periodically (every tenth of a second) fetch data from the robot to be put into Soar's working memory,

6 - periodically fetch data from the Vision System, compare it to visual data from the World Model, and put any significant differences into Soar's working memory.

ADAPT

Two different methods of learning in ADAPT:

- procedural learning of search control (from Soar)
- inductive inference of schemas

ADAPT generates procedural “chunks” when goals are satisfied:

- chunks are productions as in Soar:
  - left-hand sides contain all the working memory elements that were referenced in making the search-control decision
  - right-hand side is the decision

A search-control chunk that ADAPT learns may use:

- Bayesian estimate to make the choice of action
  - the chunk performs in one step the same choice that ACT-R would make.
- The chunk may compile the results of a search of alternatives
  - the chunk performs just as a Soar chunk does

ADAPT can perform inductive inference on schemas:

- examine the execution history and hypothesize more general schemas that are added to the declarative memory
  - by replacing a constant with a variable or by enlarging an interval of permitted numeric values

ICARUS

Icarus [Shapiro & Langley 1999] designed as an integrated architecture for controlling an agent that exists in a complicated physical environment.

Features in common with Soar, ACT-R and other production-system architectures.

The design is modular, using separate modules for planning, perception, execution and long-term memory.

ICARUS

Designs for ICARUS have been guided by six principles:

1. Cognitive reality of physical objects
2. Cognitive separation of categories and skills
3. Primacy of categorization and skill execution
4. Hierarchical organization of long-term memory
5. Correspondence of long-term/short-term structures
6. Modulation of symbolic structures with utility functions

These ideas distinguish ICARUS from most other architectures.

ICARUS

Designs for ICARUS

1. ARGUS – perception

   an attention mechanism to determine which of these changes is worthy of attention

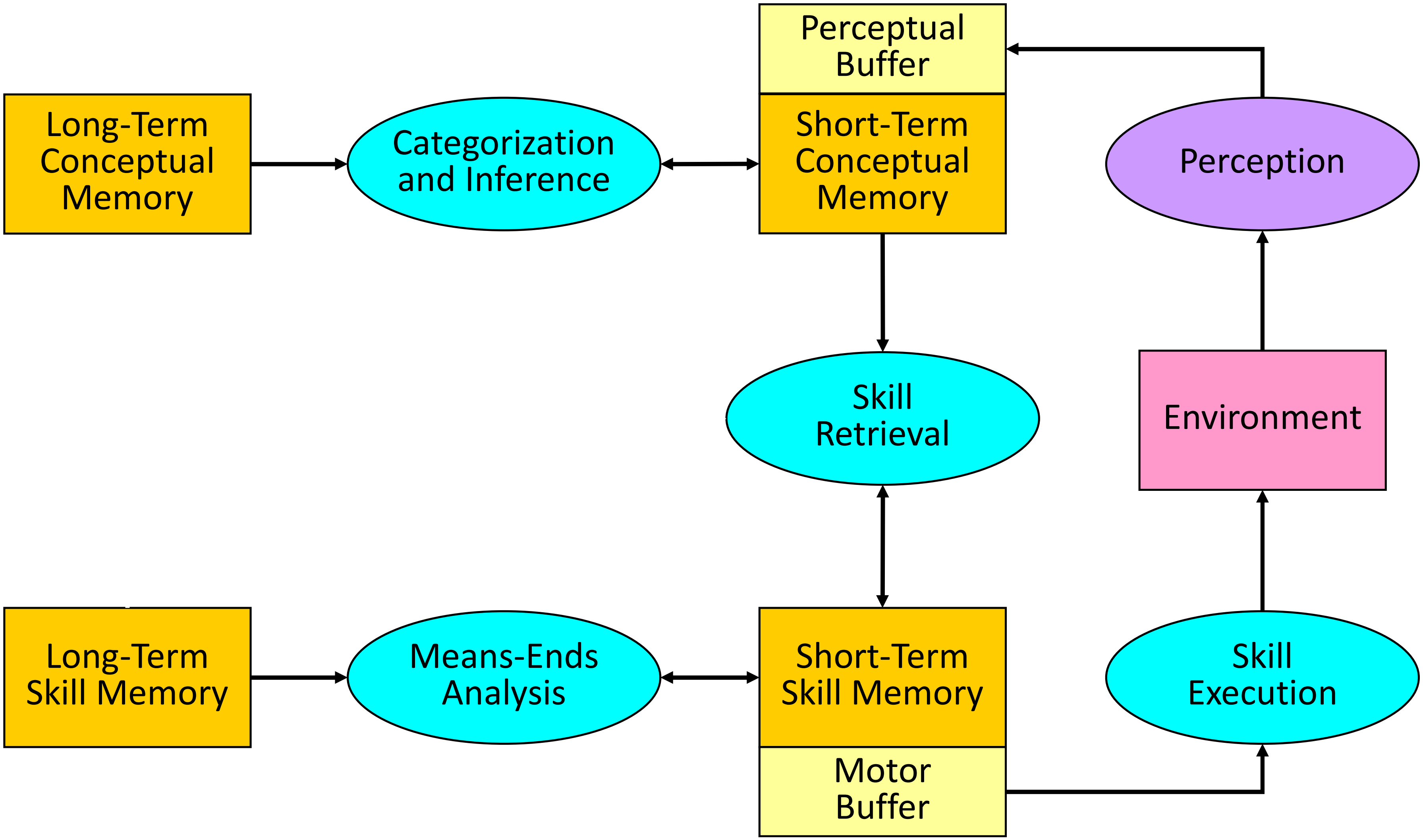
2. DAEDALUS – planning

   heuristic best-first search through the problem space.

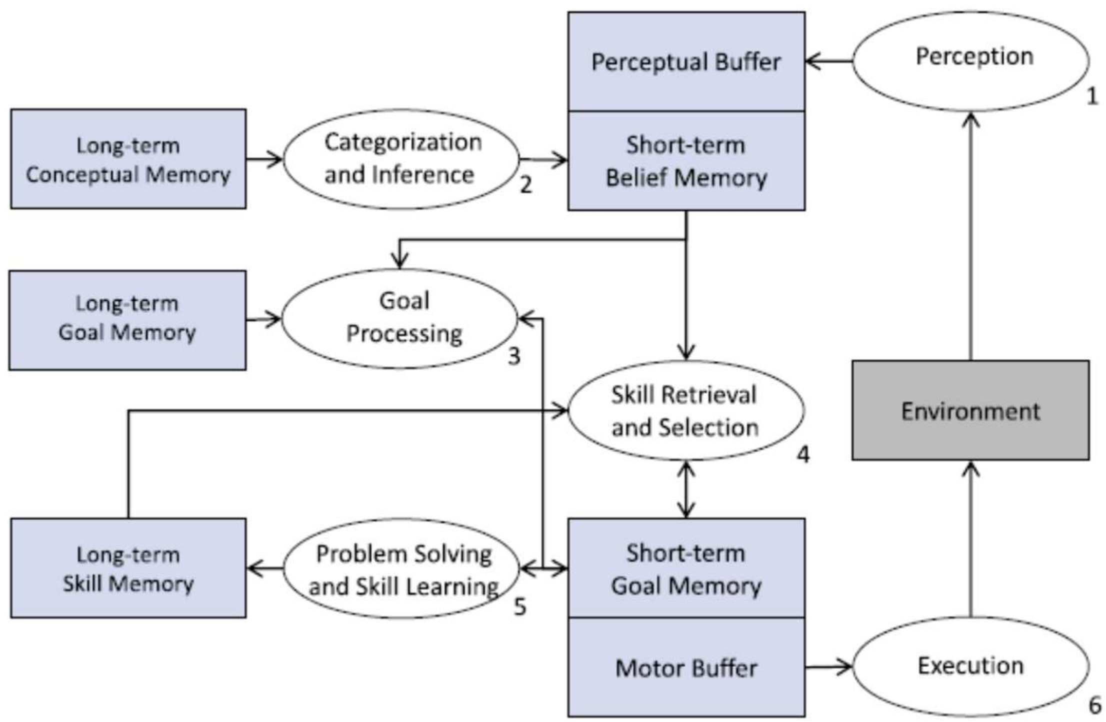
3. MAENDER – execution
   executes all the primitive actions

4. LABYRINTH – memory
   probabilistic hierarchy to store the knowledge

Overview of the ICARUS Architecture



Overview of the ICARUS Architecture



Some Concepts from the Blocks World

```
(on  (?block1 ?block2)
 :percepts     ((block ?block1 xpos ?x1 ypos ?y1)
                (block ?block2 xpos ?x2 ypos ?y2 height ?h2))
 :tests        ((equal ?x1 ?x2)
                (>= ?y1 ?y2)
                (<= ?y1 (+ ?y2 ?h2))) )

(clear  (?block)
 :percepts     ((block ?block))
 :negatives    ((on ?other ?block)) )

(unstackable  (?block ?from)
 :percepts     ((block ?block) (block ?from))
 :positives    ((on ?block ?from)
                (clear ?block)
                (hand-empty)) )
```

Primitive Skills from the Blocks World

```
(pickup  (?block ?from)
 :percepts     ((block ?block xpos ?x)
                (table ?from height ?h))
 :start        ((pickupable ?block ?from))
 :requires     ( )
 :actions      ((∗ move ?block ?x (+ ?h 10)))
 :effects      ((holding ?block))
 :value        1.0 )

(stack  (?block ?to)
 :percepts     ((block ?block)
                (block ?to xpos ?x ypos ?y height ?h))
 :start        ((stackable ?block ?to))
 :requires     ( )
 :actions      ((∗ move ?block ?x (+ ?y ?h)))
 :effects      ((on ?block ?to)
                (hand-empty))
 :value        1.0 )
```

A Nonprimitive Skill from the Blocks World

```
(puton  (?block ?from ?to)
 :percepts     ((block ?block) (block ?from) (table ?to))
 :start        ((ontable ?block ?from) (clear ?block)
                (hand-empty) (clear ?to))
 :requires     ( )
 :ordered      ((pickup ?block ?from) (stack ?block ?to))
 :effects      ((on ?block ?to))
 :value        1.0 )

(puton  (?block ?from ?to)
 :percepts     ((block ?block) (block ?from) (block ?to))
 :start        ((on ?block ?from) (clear ?block)
                (hand-empty) (clear ?to))
 :requires     ( )
 :ordered      ((unstack ?block ?from) (stack ?block ?to))
 :effects      ((on ?block ?to))
 :value        1.0 )
```

Hierarchical Organization of Memory

ICARUS' long-term memories are organized into hierarchies:

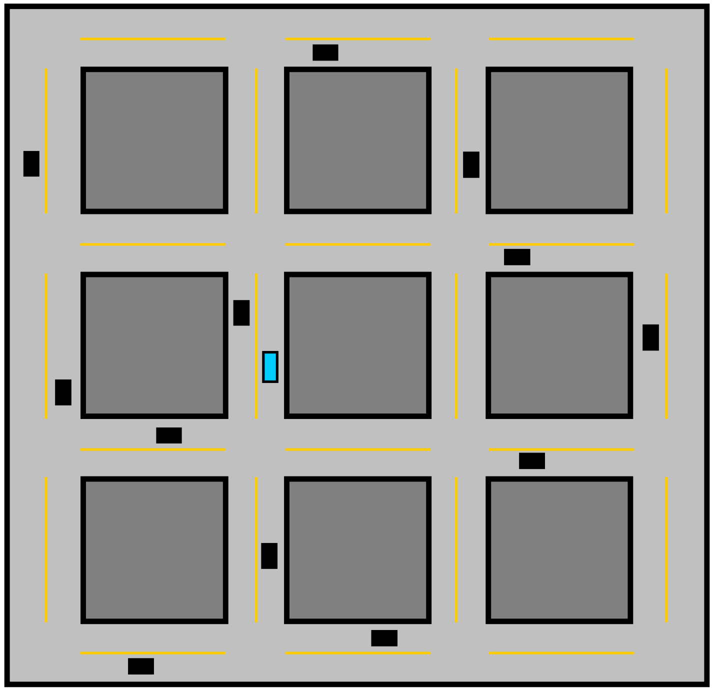
- concepts can refer to percepts and to other concepts;
- skills refer to percepts, to concepts, and to other skills.

Conceptual memory is similar to a network, but each node represents a meaningful category.

Different expansions for skills and concepts also make them similar to Horn clause programs.

These hierarchies are encoded by direct reference, rather than through working-memory elements, as in ACT and Soar.

ICARUS' Short-Term Memories



short-term skill memory

(deliver-package g029)
(avoid-collisions g001)

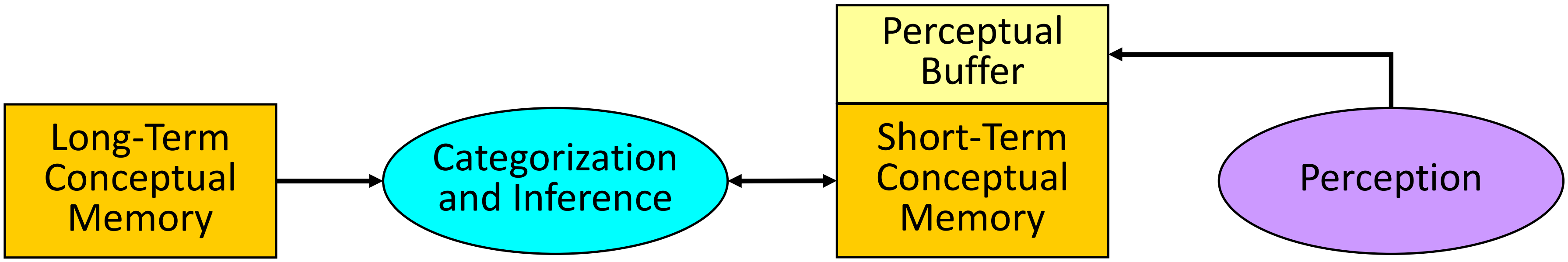
short-term concept memory

(ahead-right-corner g008)
(ahead-left-corner g011)
(behind-right-corner g017)
(approaching g001 g023)
(opposite-direction g001 g023)
(parallel-to-line g001 g019)
(on-cross-street g001 g029)

perceptual buffer

(self g001 speed 32 wheel-angle -0.2 fuel-level 0.4)
(corner g008 r 15.3 theta 0.25 street-dist 12.7)
(corner g011 r 18.4 theta -0.34 street-dist 12.7)
(corner g017 r 7.9 theta 1.08 street-dist 5.2)
(lane-line g019 dist 1.63 angle -0.07)
(street g025 name campus address 1423)
(package g029 street panama cross campus address 2134)

Categorization and Inference



On each cycle, perception deposits object descriptions into the perceptual buffer.

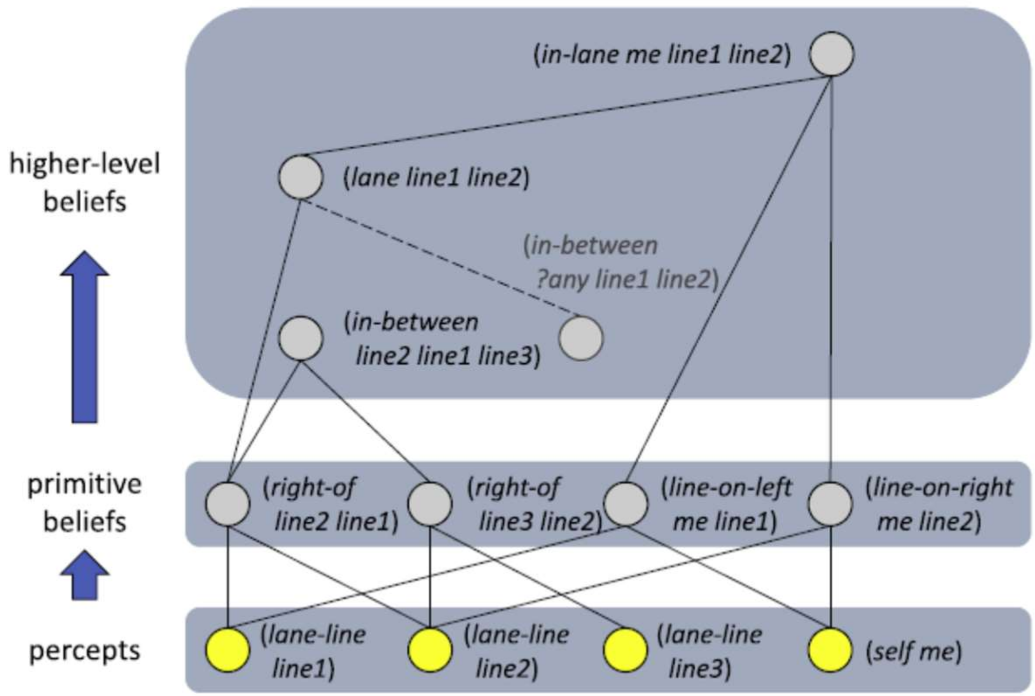
ICARUS matches its concepts against the contents of this buffer.

Categorization proceeds in an automatic, bottom-up manner, much as in a Rete matcher.

This process can be viewed as a form of monotonic inference that adds concept instances to short-term memory.

Beliefs

- Inference of beliefs
  - For each cognitive cycle, a perceptual process deposits a set of visible objects, each with a type (e.g., lane-line and self) and associated attributes, into the architecture's perceptual buffer
  - Inference mechanism finds all ways that primitive conceptual rules matches against these objects
  - Nonprimitive rules matches to infer high-level beliefs



[Choi, P. Langley. Cognitive Systems Research 2018]

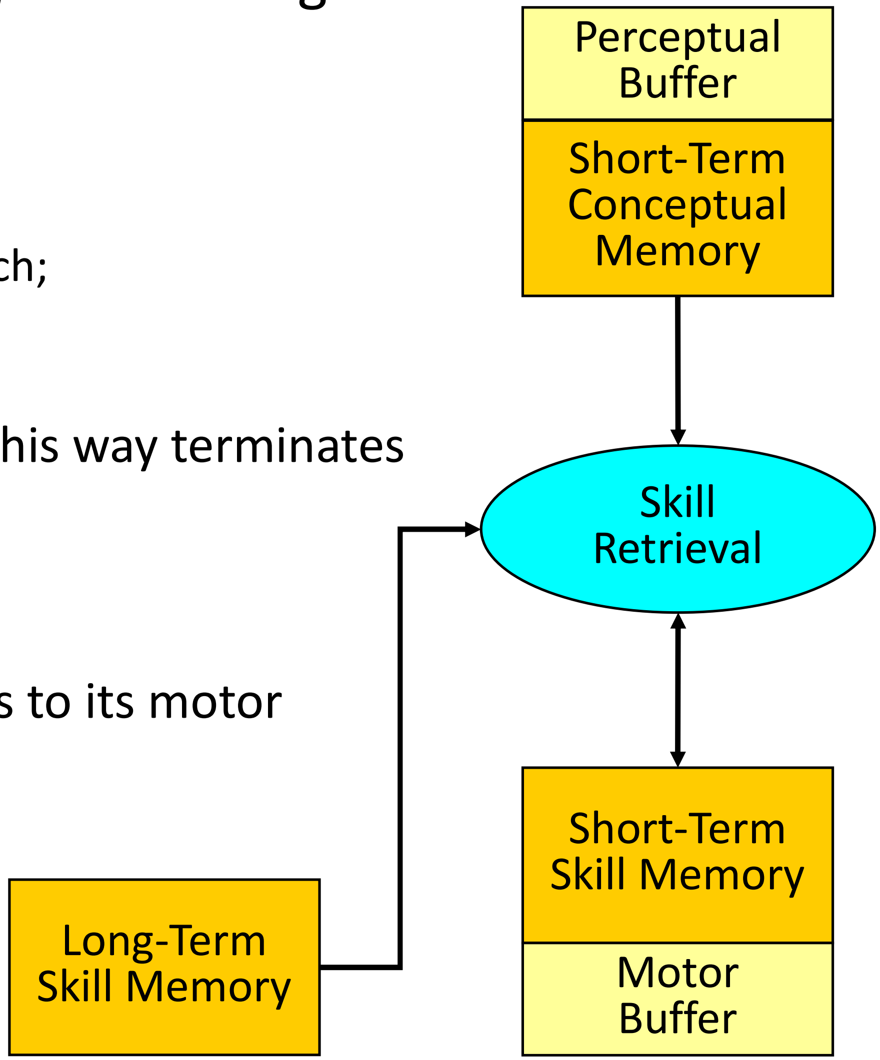
Retrieving and Matching Skill Paths

On each cycle, ICARUS finds all *paths* through its skill hierarchy which:

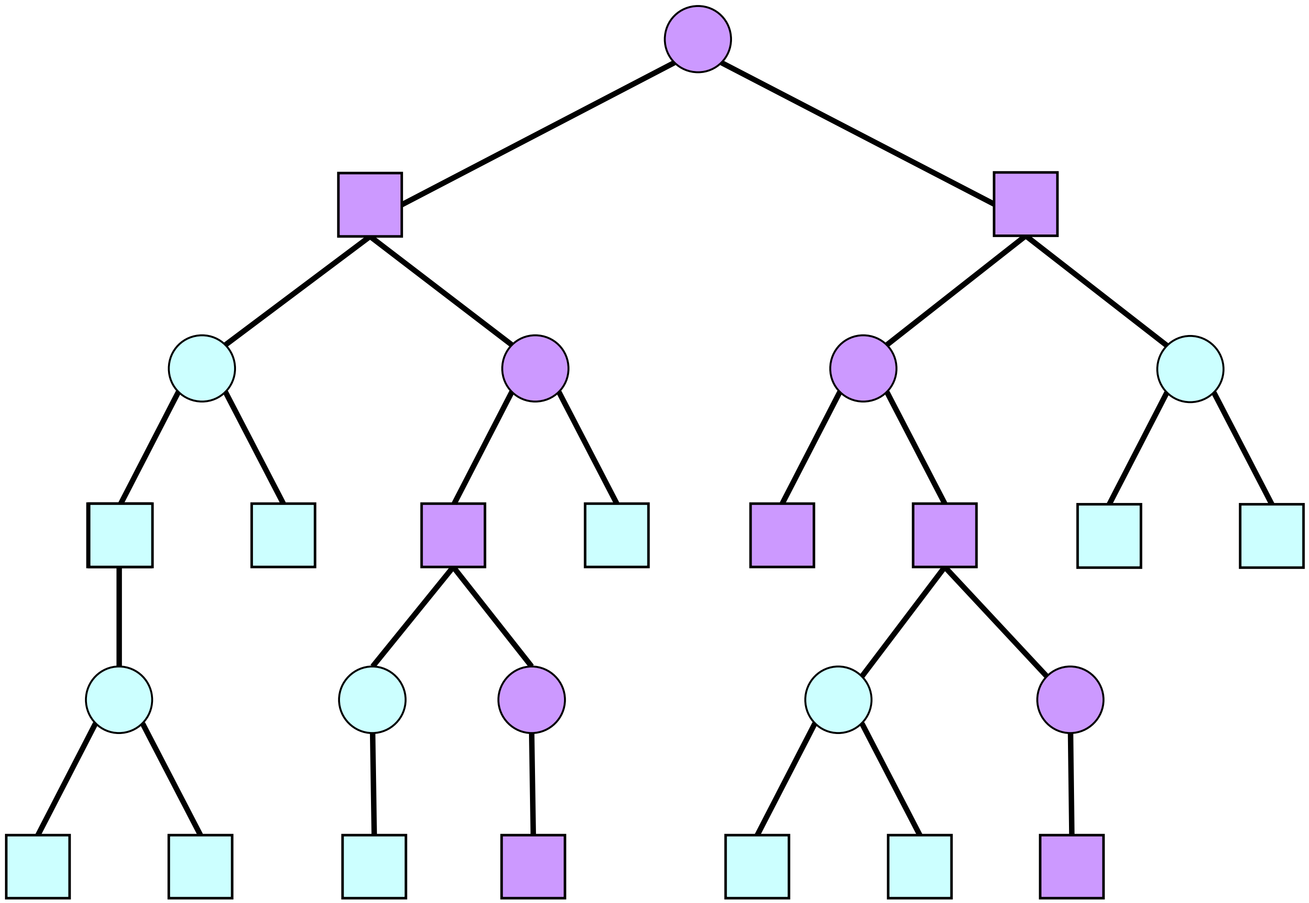
- begin with an instance in skill STM;
- have start and requires fields that match;
- have effects fields that do not match.

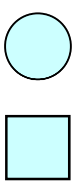
Each instantiated path produced in this way terminates in an executable action.

ICARUS adds these candidate actions to its motor buffer for possible execution.



Retrieving and Matching Skill Paths



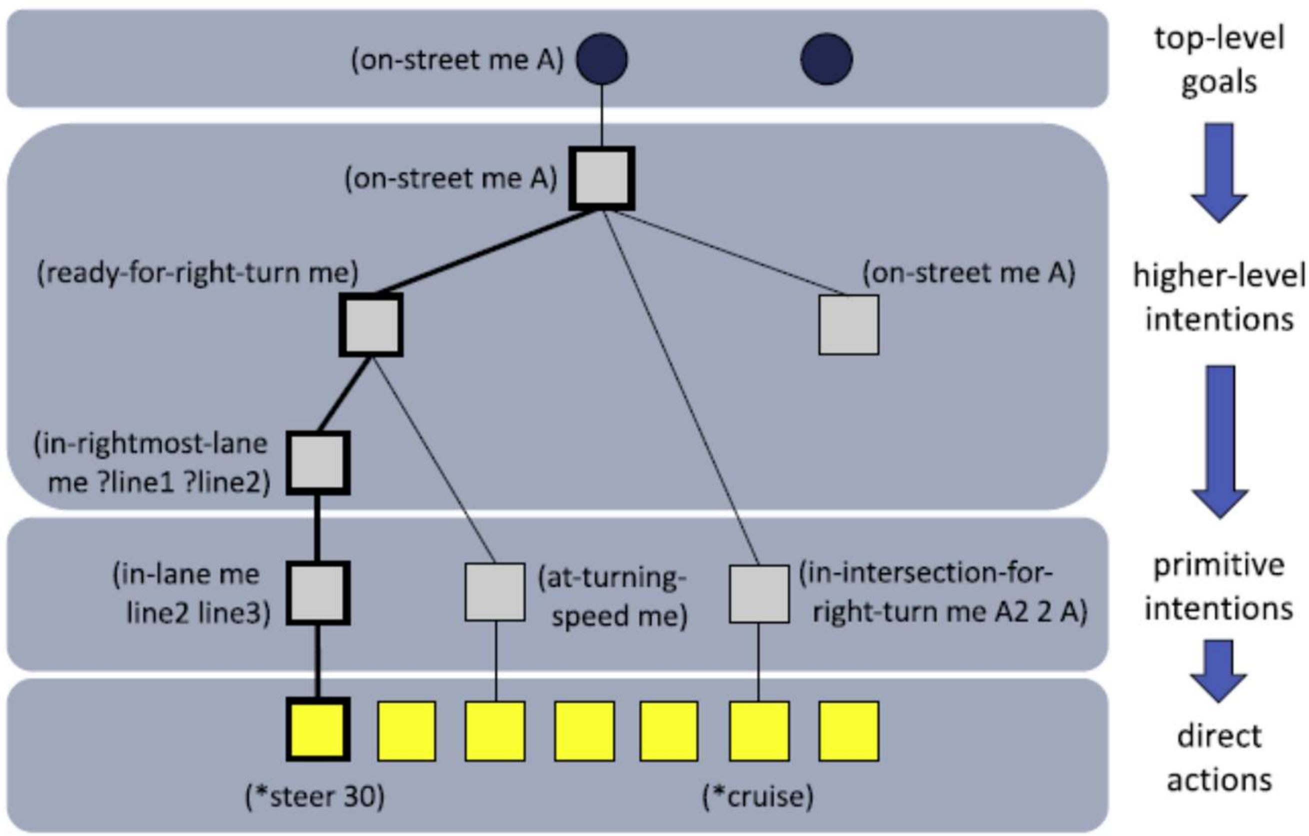


skills

skill expansions

Each path through the skill hierarchy starts at an intention and ends at a primitive skill instance.

Retrieving and Matching Skill Paths



Top-down selection of subgoals (boxes) and assocated intentions (not shown) by ICARUS' execution module on a single cognitive cyle [Choi, P. Langley. Cognitive Systems Research 2018]

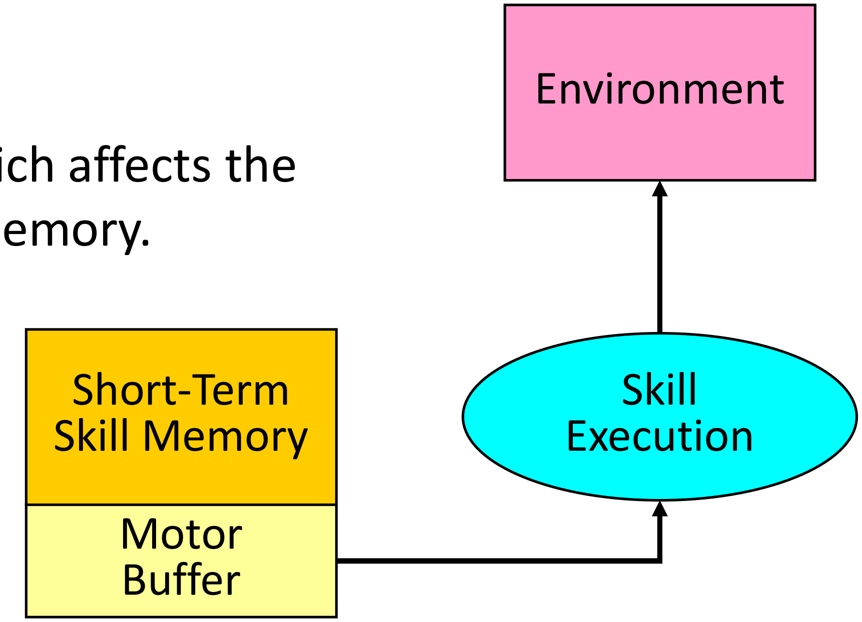
Evaluating and Executing Skills

For each selected path, ICARUS computes a utility by summing the values of each skill along that path.

For each path, in order of decreasing utility:

- If required resources are available, execute actions;
- If executed, commit the resources for this cycle.

These actions alter the environment, which affects the perceptual buffer and thus conceptual memory.



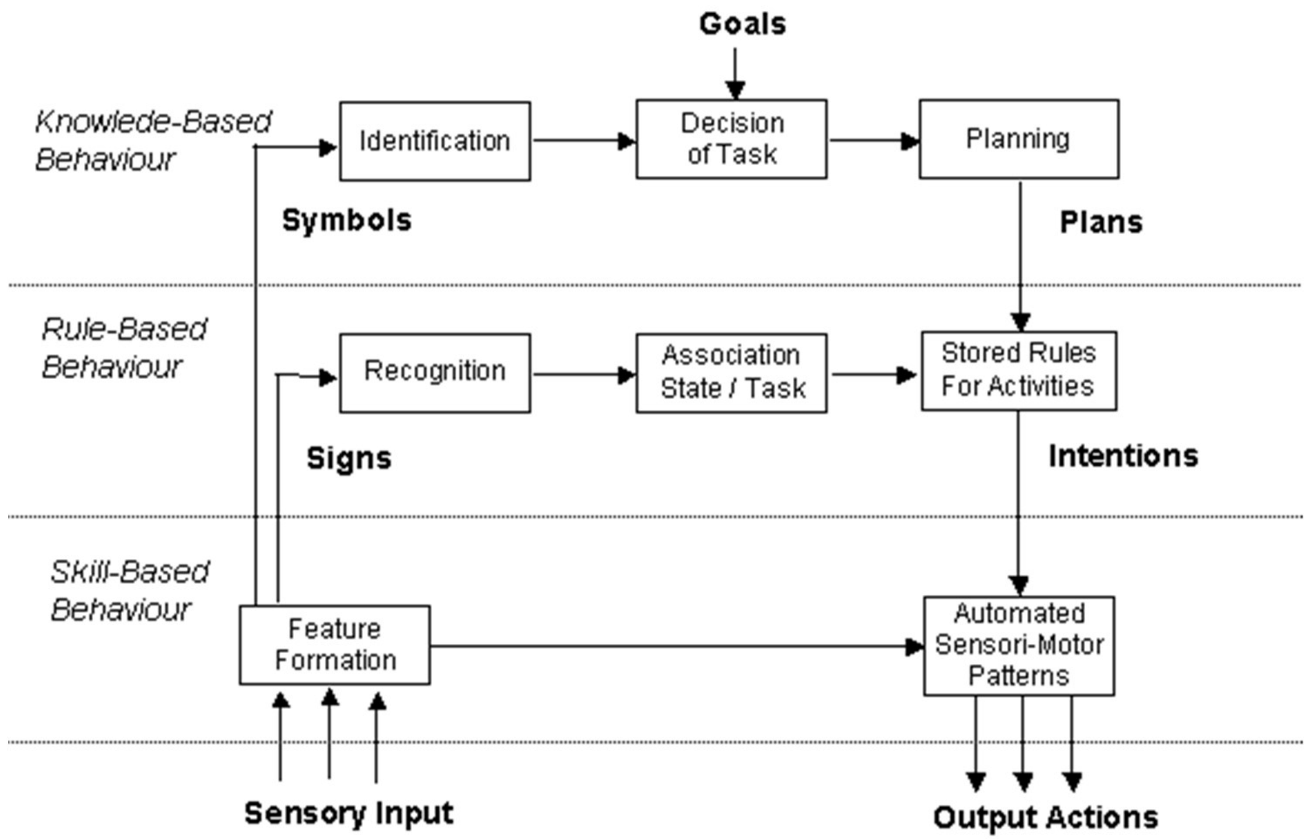
Learning

Incremental learning is central to most cognitivist cognitive architectures:

- new cognitive structures are created by problem solving when an impasse is encountered
- ICARUS adopts a similar approach:
    - when an execution module cannot find an applicable skill that is relevant to the current goal, it resolves the impasse by backward chaining

Cognitive Control

- Cognitive Control:
  - SRK (Skill, Rule, Knowledge) [Rasmussen83,86,87]
    - Human factors (error classification)



Cognitive Control

- Cognitive Control:
  - SRK (Skill, Rule, Knowledge) [Rasmussen83,86,87]
    - Different types of information processing involved in industrial tasks
    - Framework for identifying the types of error likely to occur in:
      - different operational situations,
      - different aspects of the same task with different types of information processing demands

    - Knowledge based mode,
      - Human carries out a task in a conscious manner
    - Skill based mode,
      - Execution of highly practiced actions without conscious monitoring
    - Rule-based mode,
      - Learned rules (interacting with the plant, formal training, working with experienced process workers)
      - Situation assessment leads to recognition of which procedures apply to particular familiar situations

Cognitive Control

- Cognitive Control:
  - SRK (Skill, Rule, Knowledge) [Rasmussen83,86,87]

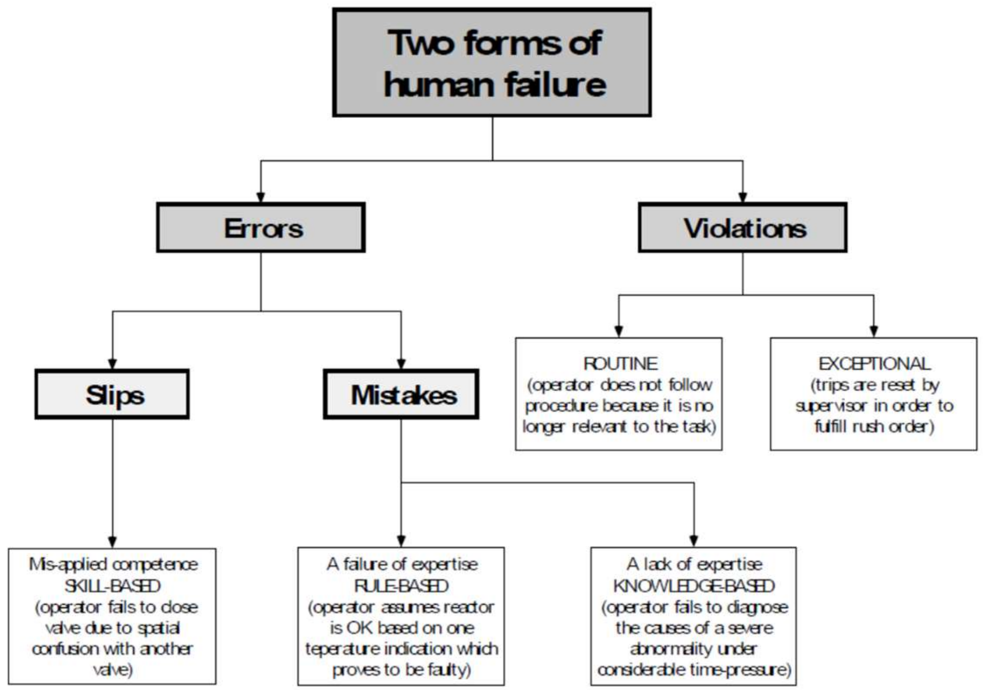
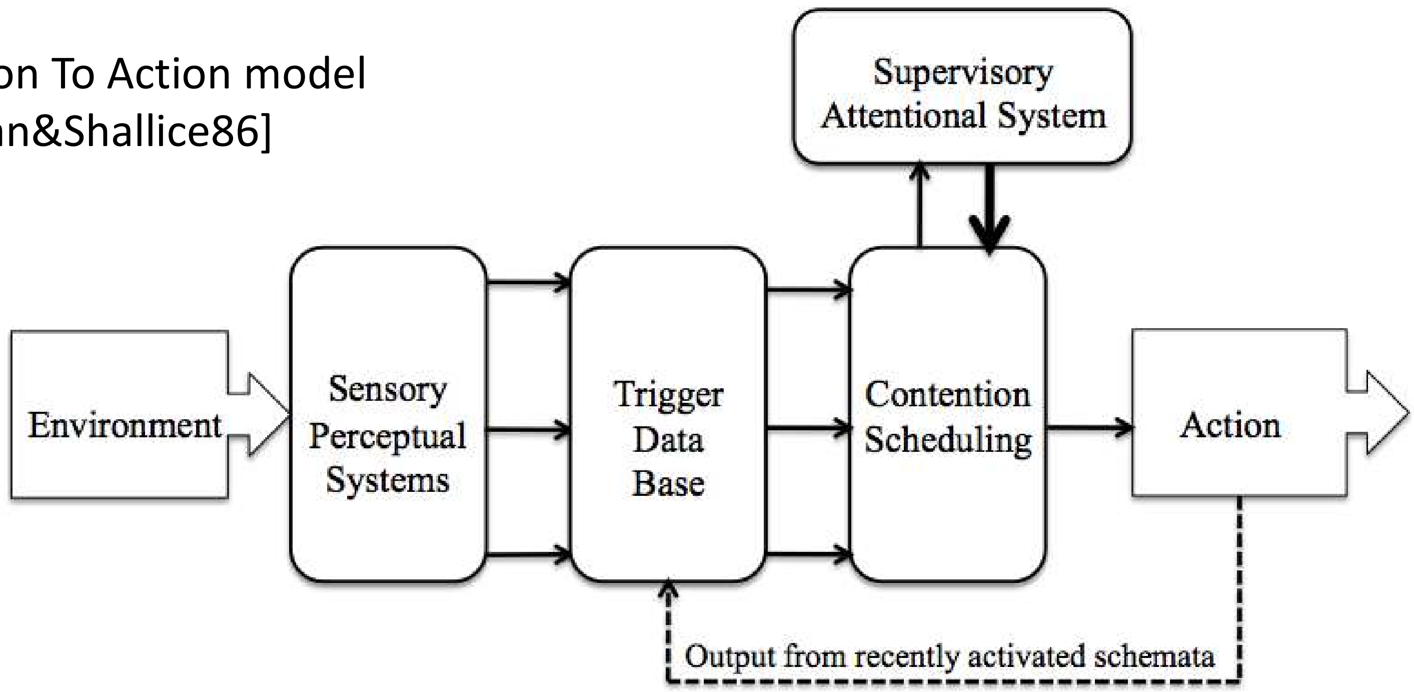


Figure 4: Classification of Human Errors (adapted from Reason, 1990)

Cognitive Control

- Cognitive Control:
  - Orchestration of cognitive and reactive processes for flexible execution of complex tasks

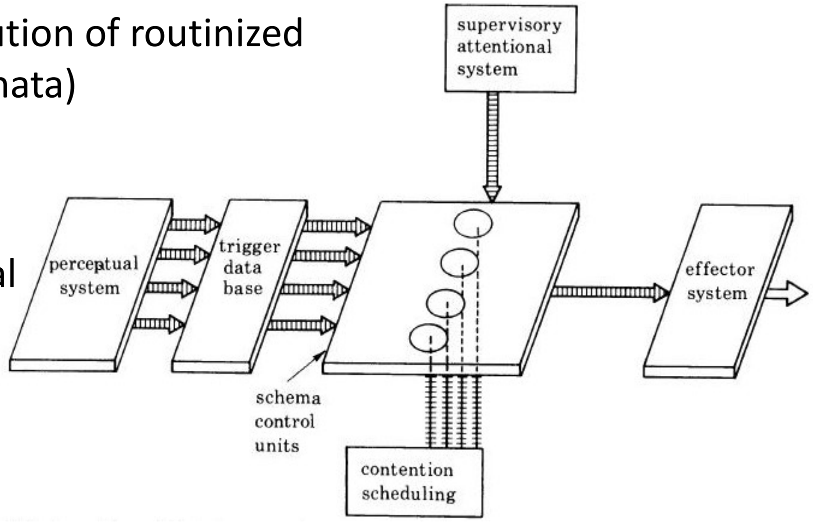
Attention To Action model
[Norman&Shallice86]



Cognitive Control and Attention

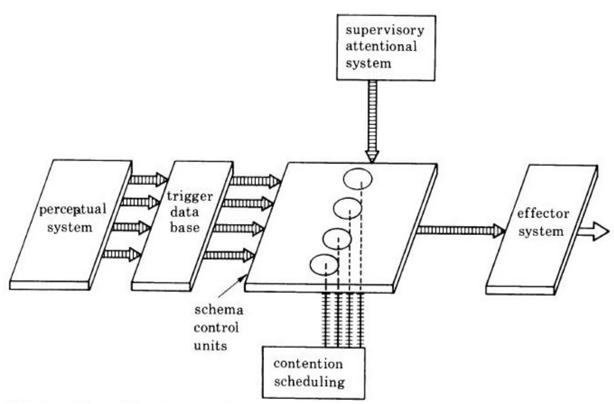
- Cognitive Control:
  - Ability of flexibly orchestrating structured goal-oriented activities and reactive actions [Posner & Snyder '75, Botvinick et al. '01]
  - Attentional mechanisms play a crucial role

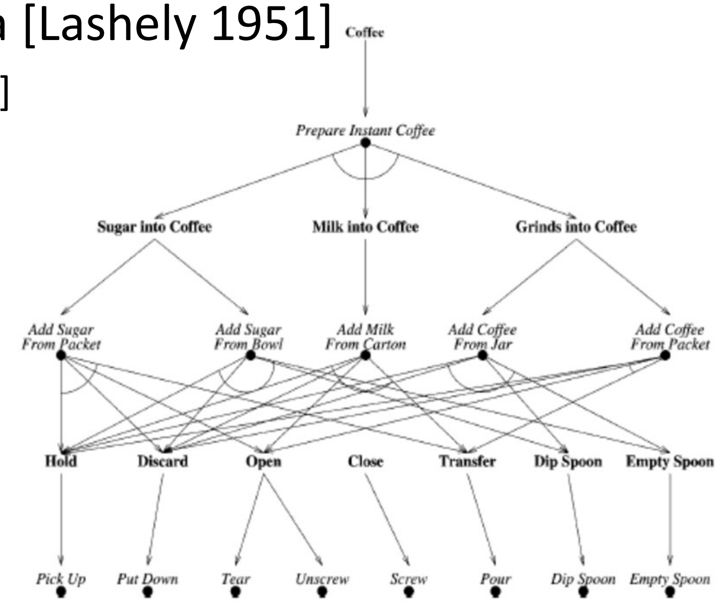
- Supervisory Attentional System [Noman Shallice '86]:
  - **Contention scheduling:**
    - low-level process that manages the execution of routinized activities (competing sensorimotor schemata)
  - **Supervisory attention:**
    - higher level mechanism that affects contention scheduling through attentional modulation (inhibition, stimulation).



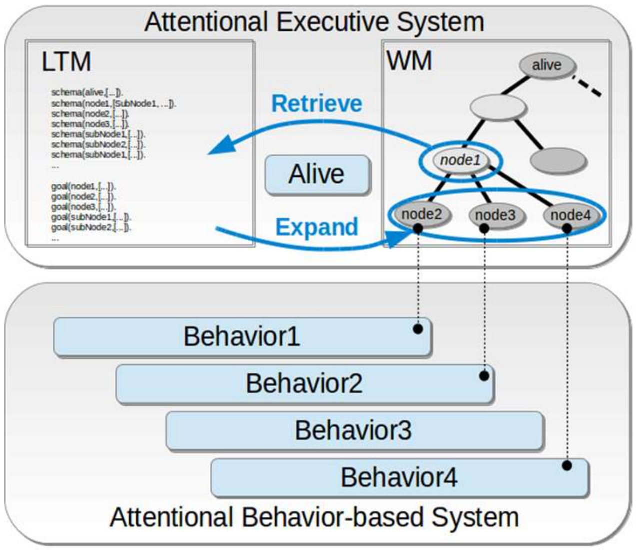
Cognitive Control and Attention

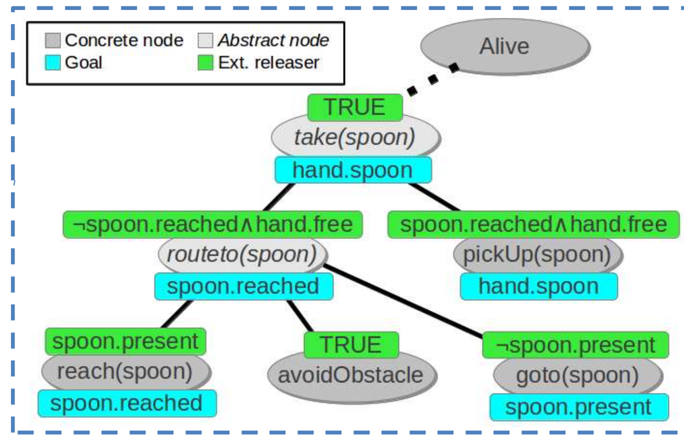
- Cognitive Control:
  - Ability of flexibly orchestrating structured goal-oriented activities and reactive actions [Posner & Snyder '75, Botvinick et al. '01]
  - Attentional mechanisms play a crucial role
- Supervisory Attentional System [Noman Shallice '86]:
  - Hierarchically organized action schemata [Lashely 1951]
    - Goal-oriented methods [Cooper Shallice 2000]
    - Activation values [Norman Shallice '86]

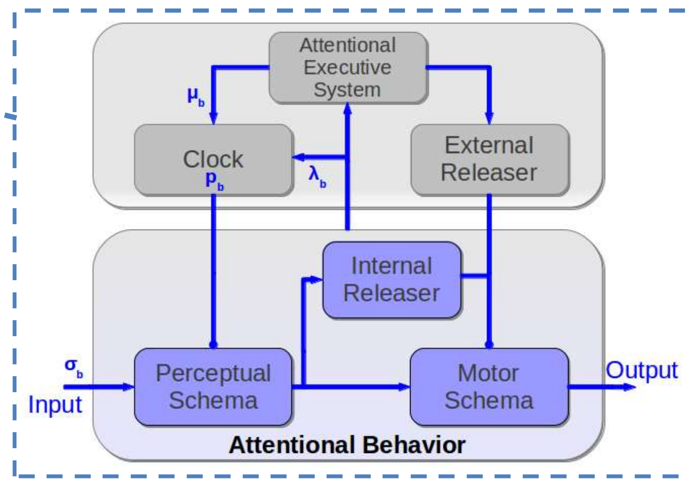




SEED: Attentional Executive System



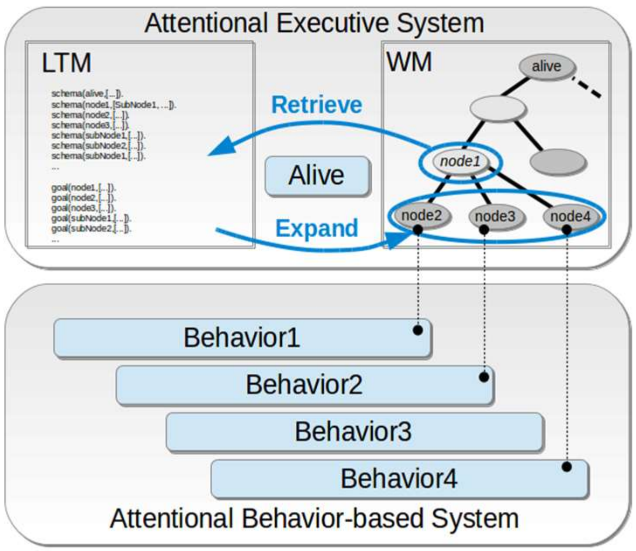


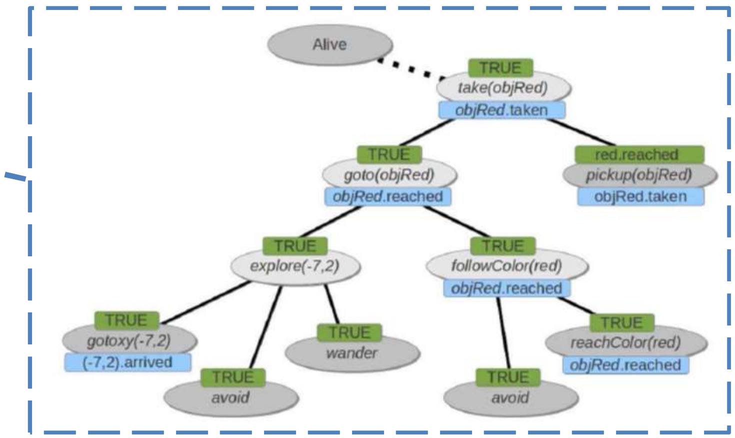


[SMC-2015, IEEE-TCDS-2016, AURO-2019]

- **Long Term Memory**: repository of the tasks/schemata available to the system
- **Working Memory**: maintains the executive state and the structure of the tasks/schemata in the attentional focus of the system (tasks to be executed)
- **Attentional Behaviors**: concrete sensorimotor processes associated with an activation level

SEED: Attentional Executive System





- **Long Term Memory**: repository of the tasks/schemata available to the system
  - HTN-like methods:

$$schema(m, l, e)$$
$$l = \langle (m_1, r_1), \dots, (m_n, r_n) \rangle$$

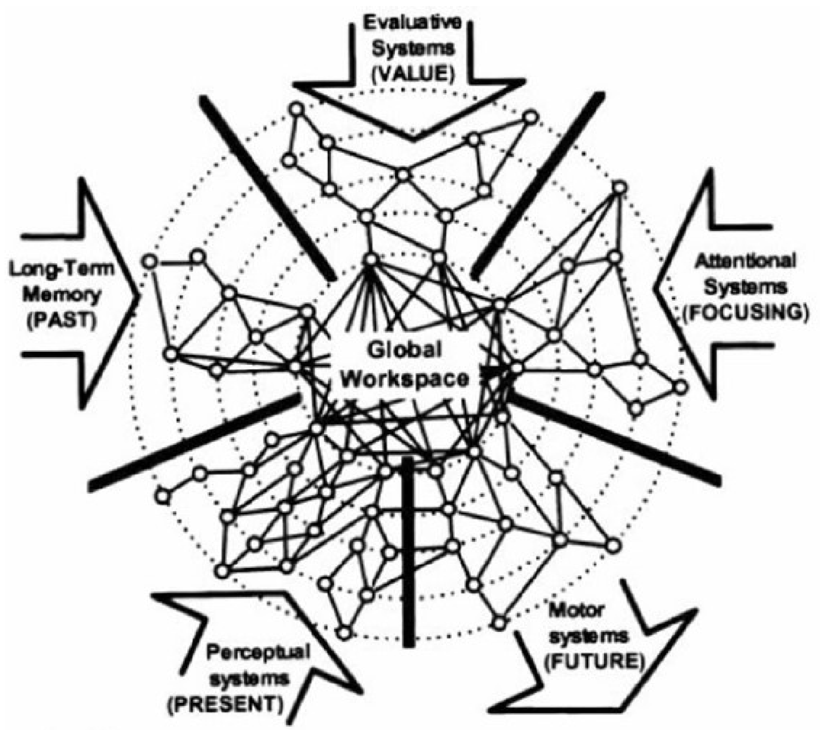
  - STRIPS-like primitive operators: $a \in A$

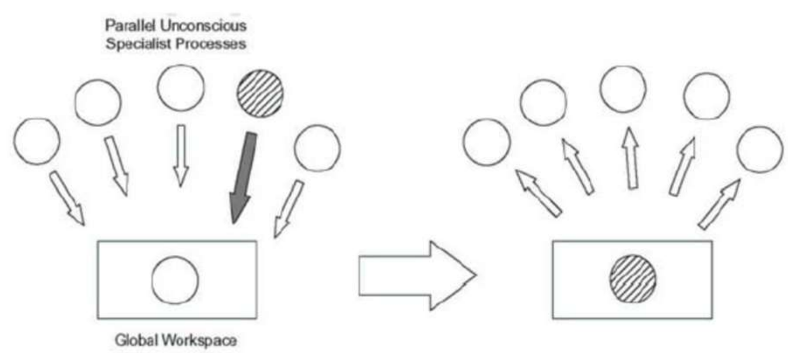
schema(take(Obj)
⟨(goto(Obj), true), (pickup(Obj), Obj.reached)⟩
Obj.taken)
schema(goto(Obj)
⟨(explore(X, Y), true), (followColor(Obj), true)⟩
Obj.reached).

- **Execution vs Planning:**
  - Executive Model extends an associated HTN Planning Domain

Cognitive Control

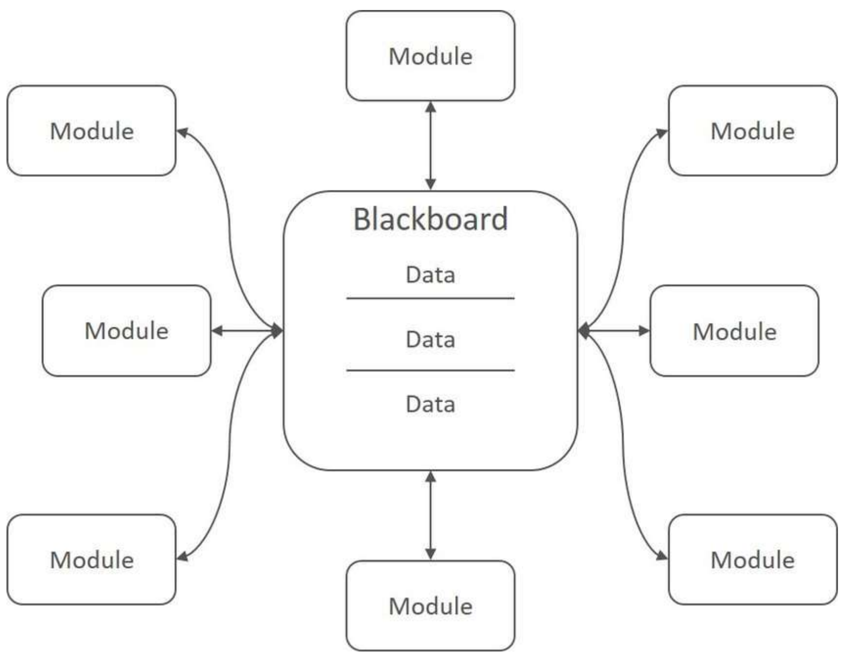
- Global Workspace Theory [Baars97]
  - conscious experience:
    - global workspace, set of specialized unconscious processors, and a set of unconscious contexts that serve to select, evoke, and define conscious contents (Baars, 1988).





Black Board Architecture

- Black Board Systems [Erman et al. 1980]
  - Group of specialists with a large blackboard using the blackboard as the workplace for cooperatively developing the solution
    - Problem specifications written onto the blackboard
    - knowledge sources (KSs) can apply their expertises
    - Control shell, which controls the flow of problem-solving activity in the system



LIDA

Learning - Intelligent Distribution Agent [Frankling ed at. 2006]

Hybrid Architecture
- http://ccrg.cs.memphis.edu/framework.html

Assumptions:
- Cognitive cycles (~10 Hz) as building blocks of cognitive processing
    - Memory access and action selection
    - Higher-level cognitive processes are based on them

Cognitive Cycle:
- Understanding phase:
    - From low-level features to episodic and declarative memory (situation model)
- Attention (consciousness) phase:
    - Coalitions of salient portions of the situation model (competition to global ws)
- Action selection and learning phase:
    - Schemas are instantiated and compete for the execution

LIDA

Learning - Intelligent Distribution Agent [Frankling ed at. 2006]

Hybrid Architecture
- http://ccrg.cs.memphis.edu/framework.html

IDA: "What do I do next?"

LIDA, the learning IDA adds three modes of learning to IDA's design:

- perceptual learning,
- episodic learning,
- procedural learning

LIDA

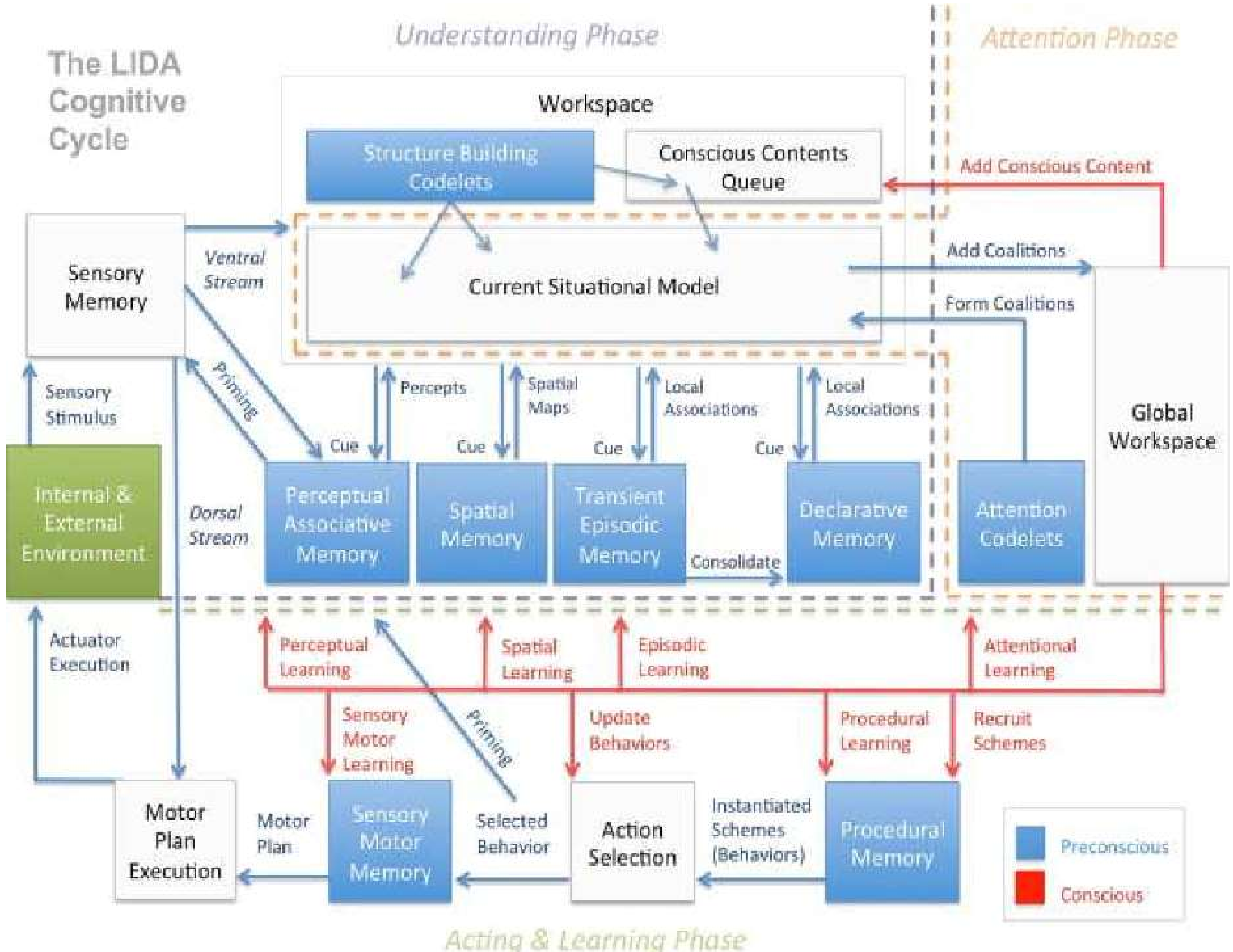


Figure 1 **LIDA Cognitive Cycle Diagram**

LIDA

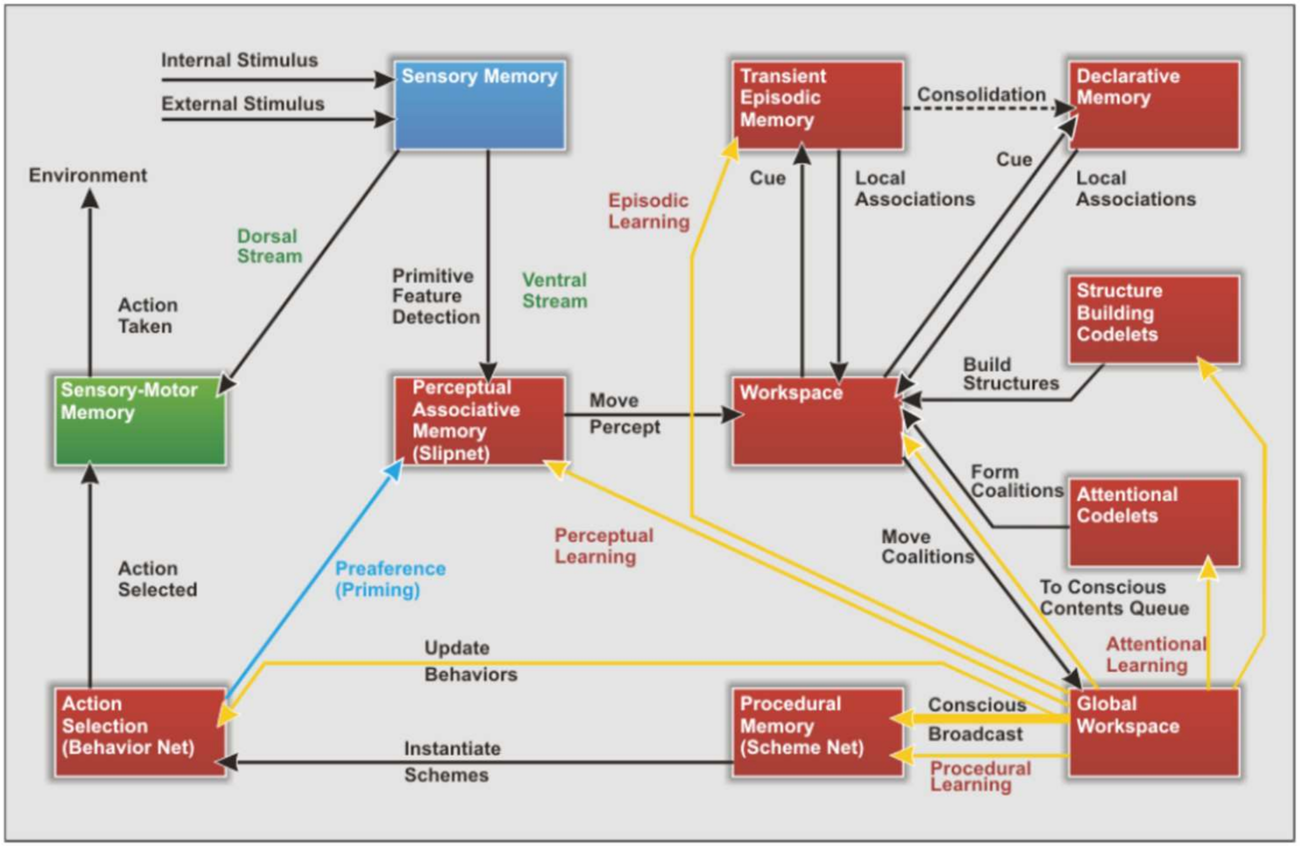


Figure 1. The LIDA Model Diagram

LIDA

Learning - Intelligent Distribution Agent [Frankling ed at. 2006]

Default implementations of the following LIDA modules:

- Environment
- Sensory Memory
- Perceptual Associative Memory
- Transient Episodic Memory
- Declarative Memory
- Workspace
- Structure-Building Codelets

  tasks maintaining the current situation in the Current Situational Model
- Attention Codelets

  Type of task that tries to bring Workspace content in the Situational Model to the Global WS.
  Upon finding such content it creates a coalition containing the content and adds it to the Global WS
- Global Workspace

  The content of the winning coalition is the current contents to broadcast throughout the system
- Procedural Memory
- Action Selection

  Inspired by Maes' (1989) Behavior Net: selects a behavior to execute for each cognitive cycle
- Sensory-Motor Memory