# 智能体体系结构（Agent Architecture）

毛文吉

中国科学院自动化研究所

# Agent Architecture

An intelligent agent is a computer system capable of *flexible autonomous action* …

- Reactive

  Responsive to changes occurred in its environment

- Pro-active/Deliberative

  Take initiative and act purposefully toward achieving goals

Issues need to address to build agent-based systems …

- Three types of agent architecture:
  - Deliberative
  - Reactive
  - Hybrid

# What is an Agent Architecture?

- Agent "Shell", separates *structure* from variable *content*
- The principles/structure to be reused from task to task
  - *Why rediscover these from scratch each time?*
- Some well-known agent architectures:
  - *IRMA* (*Intelligent Resource-bounded Machine Arch.*)
  - *PRS, dMARS, JACK* (*started from PRS*)
  - *TouringMachines*, *InteRPaP*
  - *Soar* (*Cognitive Arch.*)

# A Brief History

- Before ~1985, almost all agents designed within AI were symbolic reasoning agents
  - Develop a *deliberative* agent or agent architecture that contains explicit *symbolic representation* of the world, and makes decisions (e.g. about what actions to perform) via *symbolic reasoning*
- Around 1985~1990, problems with symbolic reasoning led to a reaction against this – the so-called reactive agents
- Since late 90s, a number of alternatives proposed: hybrid architectures, which attempt to combine the *best* of reasoning and reactive architectures

# Outline

- Reactive agent architectures
- Deliberative agent architectures
- Hybrid agent architectures
- Soar agent architecture

# Outline

- Reactive agent architectures
- Deliberative agent architectures
- Hybrid agent architectures
- Soar agent architecture

# Reactive（反应式）Architectures

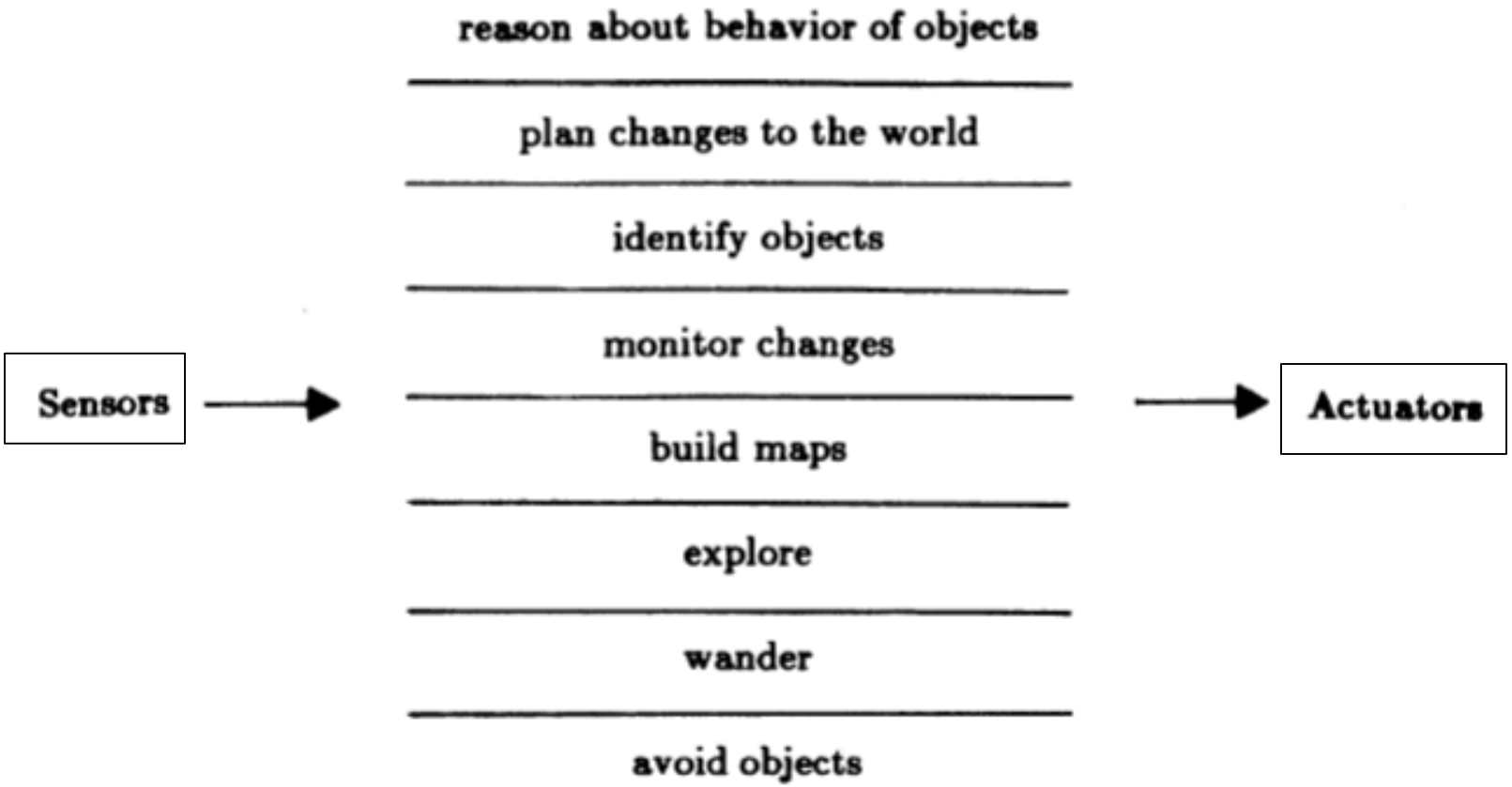
- The unsolved problems associated with symbolic AI have led to the development of reactive architectures
- Rodney Brooks (1986) is the central representative of this view, who has put forward three theses:
  - Intelligent behavior can be generated without explicit representations of the kind that symbolic AI proposes
  - Intelligent behavior can be generated without explicit abstract reasoning of the kind that symbolic AI proposes
  - Intelligence is an emergent property of certain complex systems

# Brooks: Subsumption（包容式）Architecture

- A subsumption architecture is a hierarchy of task-accomplishing *behaviors*
- Each behavior is a *rather simple* rule-like structure
- Each behavior '*competes*' with others to exercise control over the agent
- Lower layers represent more primitive kinds of behavior (such as avoiding obstacles), and have *precedence*（优先权）over layers further up the hierarchy
- The resulting systems are *extremely* simple in terms of the amount of computation cost

# Layered Control in Subsumption Architecture

- Task decomposition of mobile robot control system:




R. A. Brooks. A Robust Layered Control System for a Mobile Robot. IEEE Journal of Robotics and Automation, RA-2(1):14-23, 1986

# Reactive Architectures: Features

Advantages:

- Simplicity (fast response, real-time efficiency)
- Economy
- Computational tractability
- Robustness against failure

Limitations:

- "*Short-term*" view（短视）for decision making
- Difficult to make reactive agents learn from experience
- Hard to engineer agent with large numbers of behaviors
  - Dynamics of the interactions between different behaviors become too complex to be understood

# Outline

- Reactive agent architectures
- Deliberative agent architectures
- Hybrid agent architectures
- Soar agent architecture

# Deliberative（慎思式）Architectures

- BDI-based agent architectures
  - *IRMA* (*Intelligent Resource-bounded Machine Arch*.) [Bratman 88]
  - *PRS-type* architecture: *PRS* [87], *JACK* [01], *dMARS* [04]
  - *RAP* (*Reactive Action Packages*)
  - *TCA* (*Task Control Architecture*)
- BDI-based teamwork models
  - *GRATE** [Jennings 95]
  - *COLLAGEN* [Rich & Sindner 97]
  - *STEAM* [Tambe 97], TEAMCORE [Tambe *et al* 00]

  … …

# Outline

- Reactive agent architectures
- Deliberative agent architectures
- ➢ Hybrid agent architectures
- Soar agent architecture

# Hybrid（混合式）Architectures

- Many researchers have argued that *neither* a completely deliberative *nor* completely reactive architecture is suitable for building agents
- A hybrid architecture attempts to build an agent system out of *two* (or *more*) subsystems by marrying them:
  - Deliberative sub-structure

    Using a symbolic model to develop plans and make decisions
  - Reactive sub-structure

    Capable of reacting to events without complex reasoning

# Hybrid Architectures

- The reactive component is often given the precedence over the deliberative one
- Naturally lead to the idea of the layered architecture, of which *TouringMachines* (1992) and *InteRRaP* (1997) are representative examples
- An agent's control subsystems are arranged into a hierarchy, with higher layers dealing with information at increasing levels of abstraction

# Hybrid Architectures

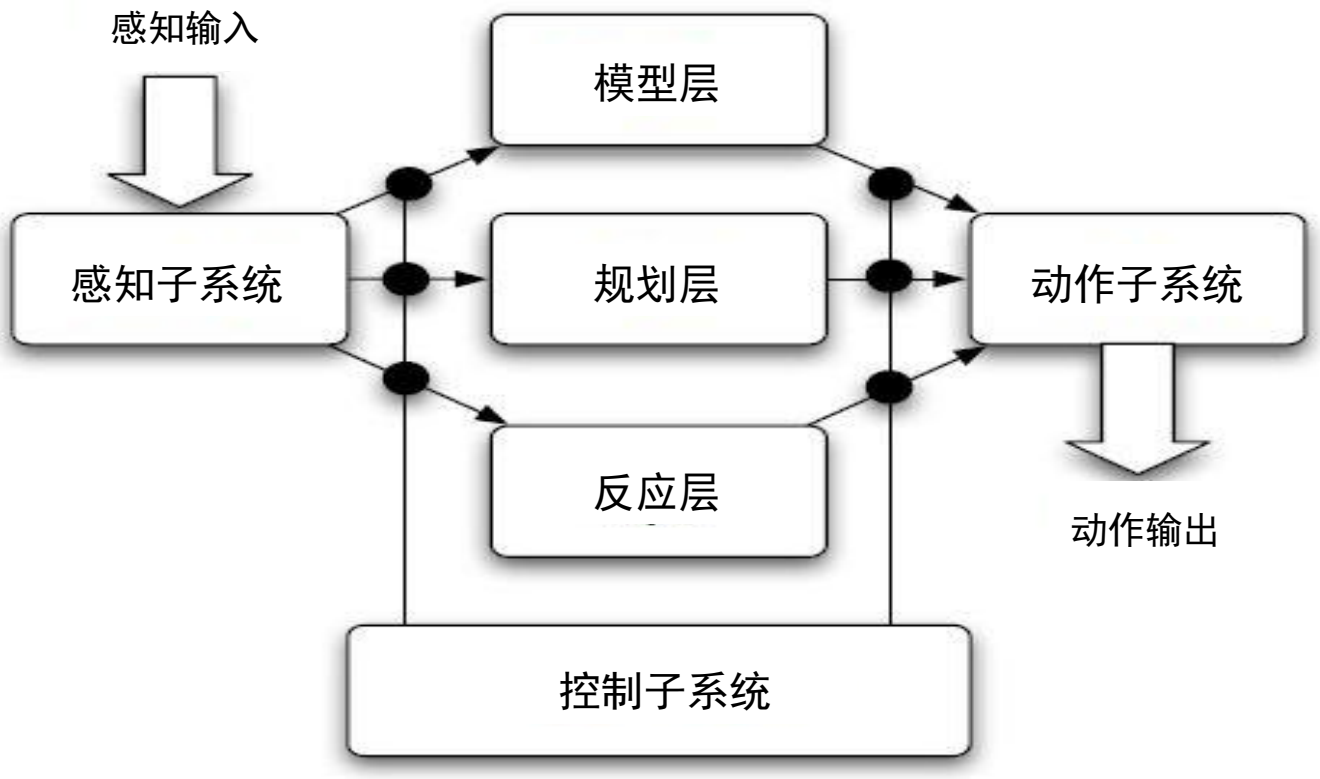
- Key problem: what kind of *control* framework to embed the agent's subsystems in, so as to manage *interactions* between layers
- Horizontal layering
  - Layers are each directly connected to the sensory input and action output
  - In effect, each layer itself acts like an agent, producing suggestions as to what action to perform
- Vertical layering
  - The sensory input and action output are each dealt with by at most one layer

# Ferguson: TouringMachines

- The *TouringMachines* architecture consists of perception and action subsystems, which interface directly with the agent's environment, and three control layers:
  - Reactive layer
    A set of situation-action rules using *subsumption architecture*
  - Planning layer
    Construct *plans* and execute actions in order to achieve *goals*
  - Modeling layer
    Symbolic representations of the '*cognitive state*' of other entities

  *Communicate* with each other and are embedded in a control framework, which mediates between the layers using control rules

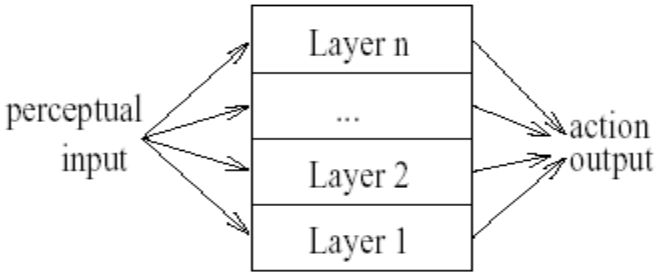
# TouringMachines Architecture



# Hybrid Architectures

- Suppose totally *n* layers with *m* possible actions suggested by each layer

**Interactions：** $m^n$

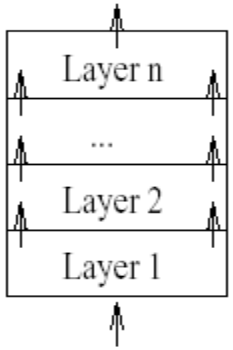


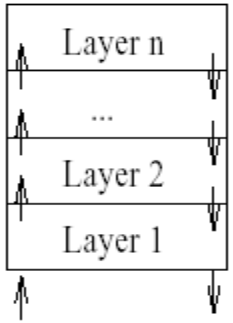

**水平层次结构**

(a) Horizontal layering

**Introduce bottleneck in central control system**

**Interactions：** $m^2(n-1)$




**垂直层次结构**

(b) Vertical layering (One pass control)

(c) Vertical layering (Two pass control)

**Not fault tolerant to layer failure**

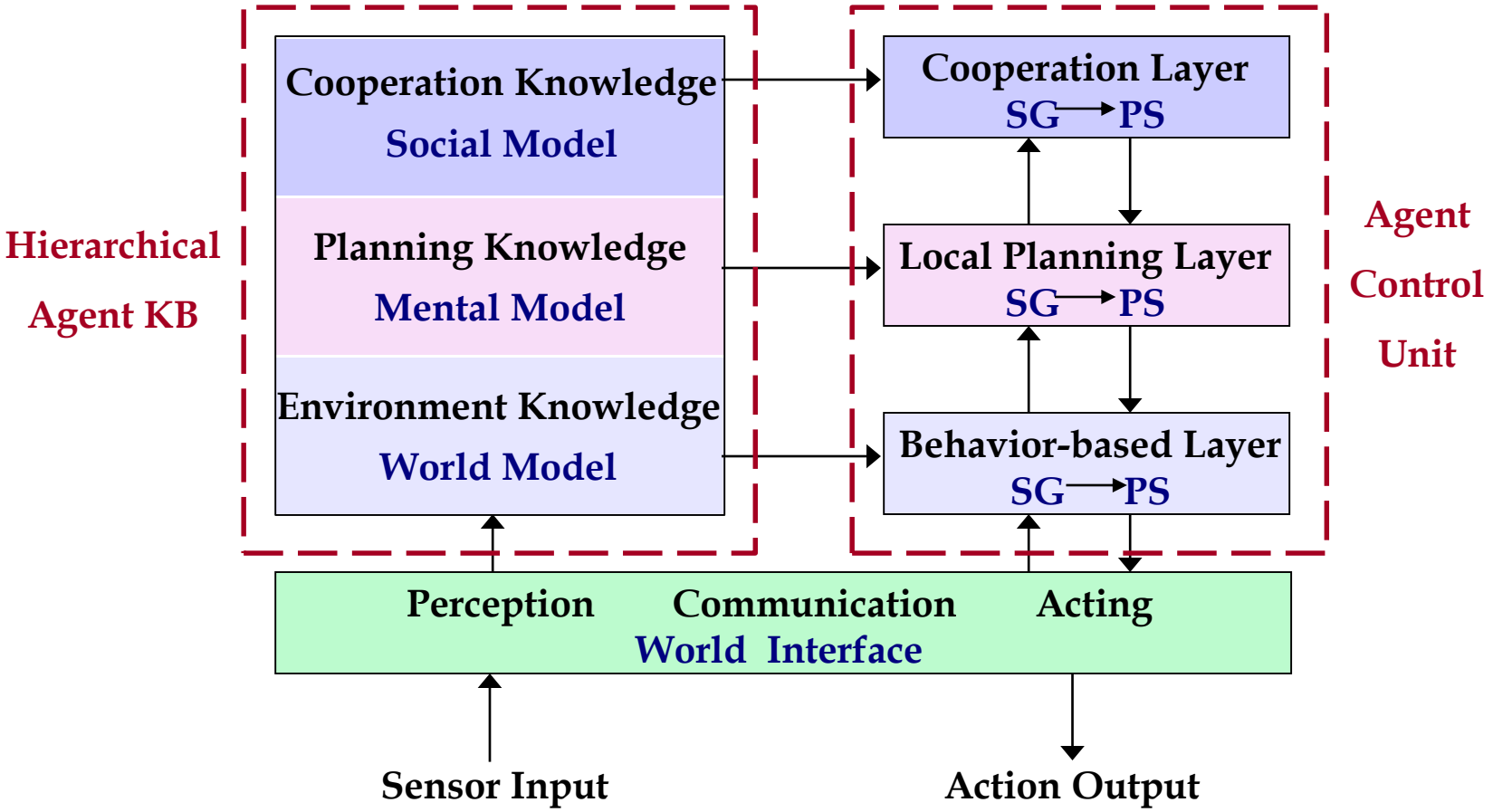
# Müller: InteRRaP

- *InteRRaP* is a vertically layered two-pass architecture
- Agent control components
  - Each control layer consists of two subprocesses:

    *SG*: situation recognition / goal activation
    *PS*: planning, scheduling and execution
  - Lowest layer represents *reactive* component and higher layers *deliberative* components
- Hierarchical knowledge bases
  - World model: Information about the environment
  - Mental model: Plan knowledge and actions of agent itself
  - Social model: Maintaining models of other agents (i.e. plans and actions of other agents)

# InteRRaP Architecture

- *InteRRaP* is a vertically layered two-pass architecture



# InteRRaP Control Function and KB

| Control Component | Corresponding Knowledge base | Function |
|---|---|---|
| Cooperation layer | Cooperation knowledge<br>Plans of other agents | Generate joint plans that satisfy the goals of a number of agents, in response to request from local planning component |
| Local planning layer | Local planning knowledge<br>Plan library of agent | Generate single-agent plans in response to requests from the behavior-based component |
| Behavior-based layer | Raw environment knowledge<br>Beliefs of agent | Implement and control the basic reactive capability of the agent |
| World interface | World model<br>Agent's perception | Manage the interface between the agent and its environment |

# Hybrid Architectures: Features

- Have been the most popular agent architecture available
- Interactions between layers are partly alleviated in InteRRaP

Main problem with layered architectures:

- *Lack* the conceptual and semantic *clarity*
- Arguably a pragmatic solution

# Comparison: Different Approaches

| | Reactive Architectures | Deliberative Architectures | Hybrid Architectures |
|---|---|---|---|
| Technology consensus | *No* consensus | *Yes* | *No clear* consensus, but lots of *similarity* |
| Methodology and theory | *No* methodology Only isolated islands of theory | *Clear* methodology A lot of theory | *No real* methodology and theory, an ad hoc one |
| Advantage/ Disadvantage | Work best using *simple* behavior in *unknown* environments | *Cannot* work well in *highly dynamic* domains | A *pragmatic* solution for real-world applications |

# Outline

- Reactive agent architectures
- Deliberative agent architectures
- Hybrid agent architectures
- ➢ Soar agent architecture

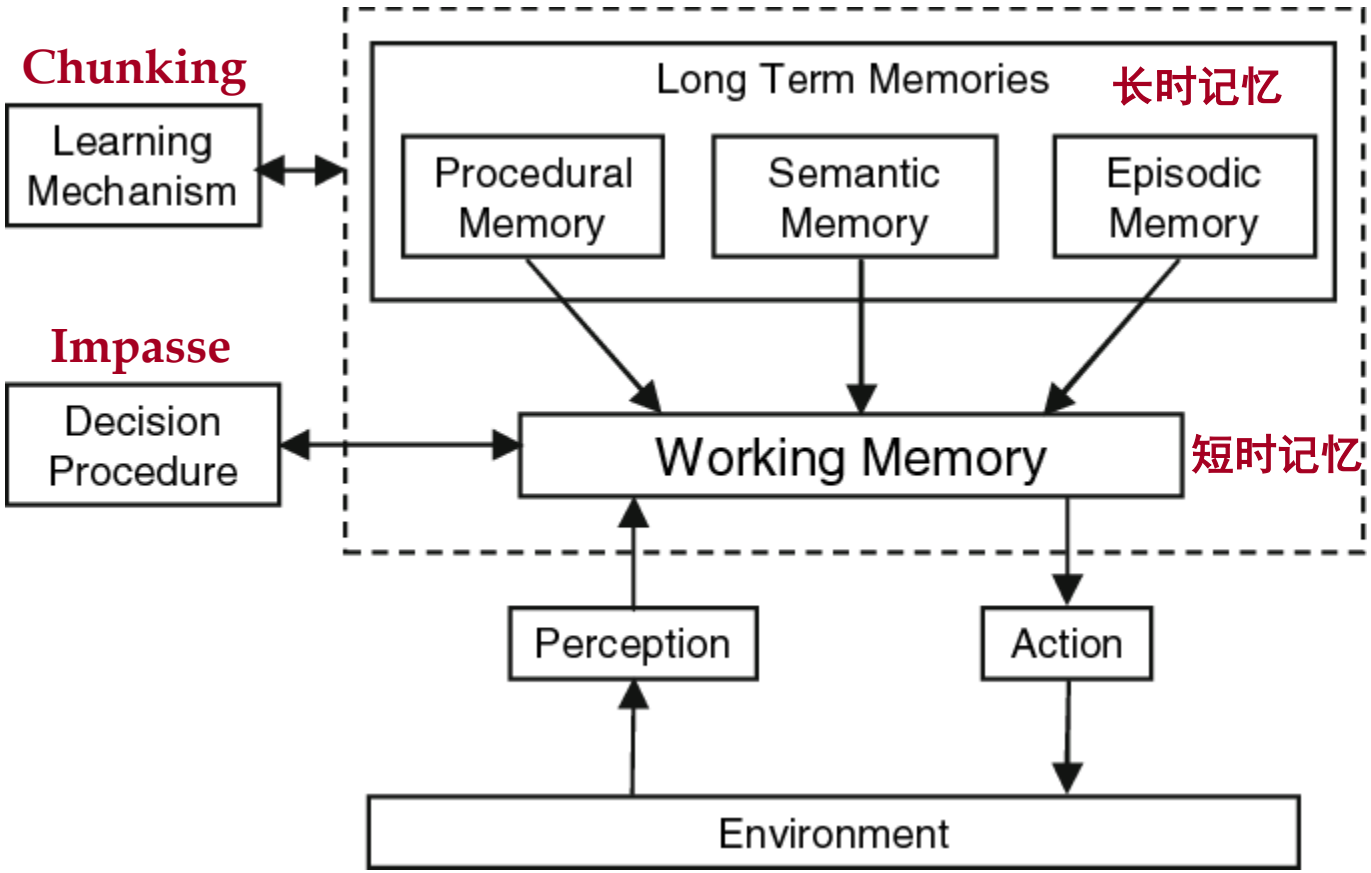
# Soar Cognitive/Agent Architecture

- A production system that can be regarded as
  - A candidate Unified Theory of Cognition (*UTC*, Allen 90)
  - An architecture for intelligent behavior
- *Soar* (*State*, *Operator And Result*) provides an integration of
  - Knowledge, planning, reaction, search and learning
  - Within one efficient cognitive architecture
- Links
  - Learning *Soar*:
    https://soar.eecs.umich.edu
  - Frequently ask questions:
    http://acs.ist.psu.edu/projects/soar-faq

# A Brief History of Soar

- First version operational in *1982*
  - Written by Allen Newell, John Laird, Paul Rosenbloom at CMU
- Versions 1-5 written in *Lisp*
- Version 6 written in *C*
- Version 7 written in *C* with *Tcl/Tk*
- Version 7.0.4 *most commonly* used
- Version 7.2 true *multi-platform* version
- Version 8.3/4 are the latest multi-platform versions
- Version 9/9.1 has improved interfaces and more types of memories
- Version 9.6.1 is the latest current version

# Original Soar Architecture



# Major Design Principles

- A single framework for all tasks and subtasks (i.e. *problem spaces*)
- A single representation of permanent knowledge (i.e. *productions*)
- A single representation of temporary knowledge (i.e. *attribute/value* pairs)
- A single mechanism for generating goals (i.e. *subgoaling*)
- A single learning mechanism (i.e. *chunking*)
- All decisions are made at *run-time*, based on current sensory data and any relevant long-term knowledge

# Some Definitions

Soar aims at supporting a full range of tasks and problem-solving methods, representing and using appropriate knowledge forms:

- A *goal* – is a desired situation
- A *state* – is a representation of a problem solving situation
- A *problem space* – is a set of states and operators for task
- An *operator* – transforms the state by some action

# Knowledge Level

- Soar can be thought of as an engine for applying knowledge to situations to yield behavior
- Knowledge is encoded in production rules. A rule has conditions on its *LHS*, and actions on the *RHS*: C —> A

Example:

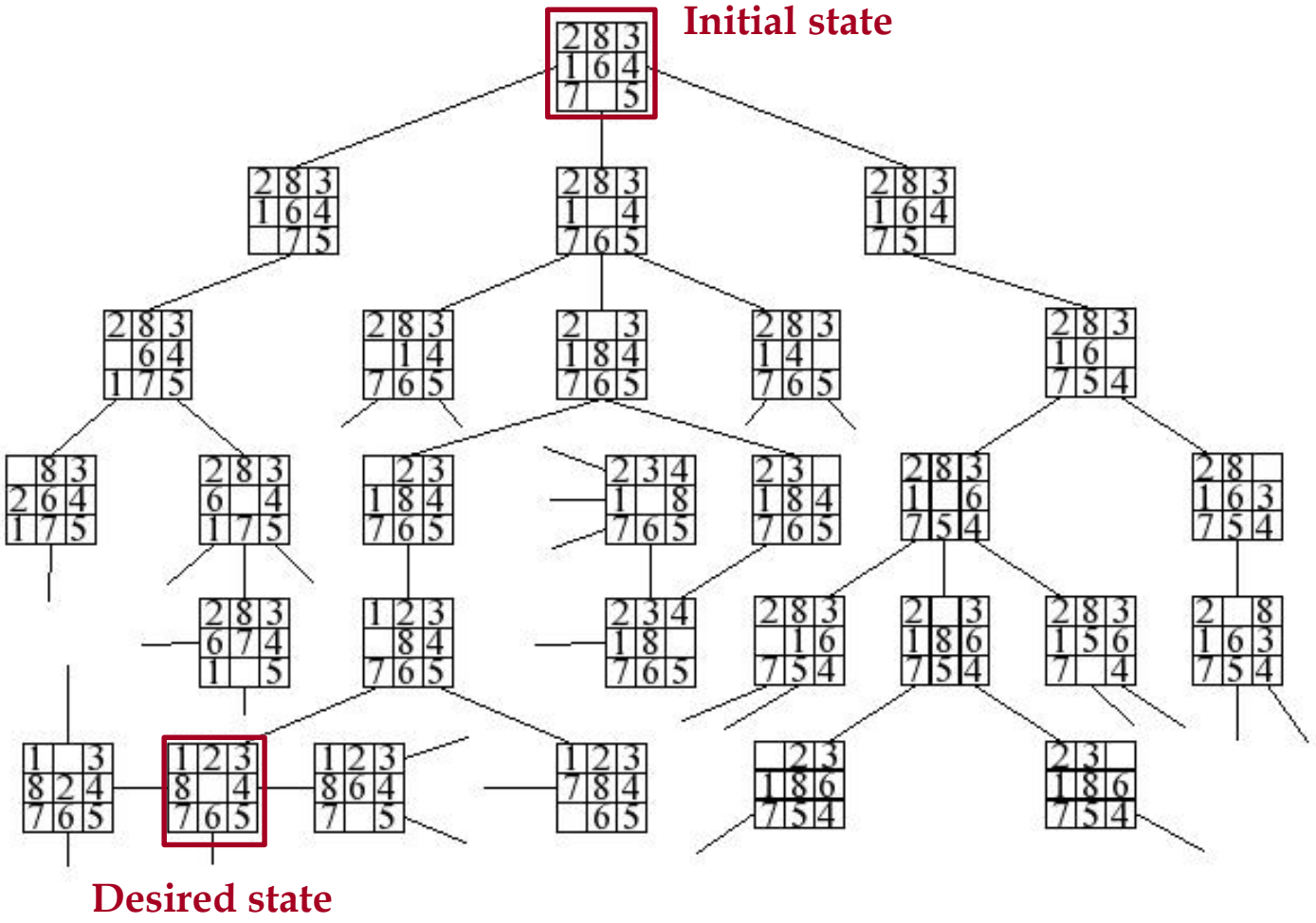
IF we are in the *hungry-thirsty* problem space, AND
current state is *thirsty*

THEN propose an operator to *apply* to current state
and call this operator "*drink*"

```
## Propose drink.
    sp {(state <s> ^problem-space <p> ^thirsty yes)
          (<p> ^name hungry-thirsty)
    –>  (<s> ^operator <o>)
          (<o> ^name drink)}
```

# Problem Space Level

- Structure of the problem space for Eight Puzzle:



# Adding Knowledge to Problem Space

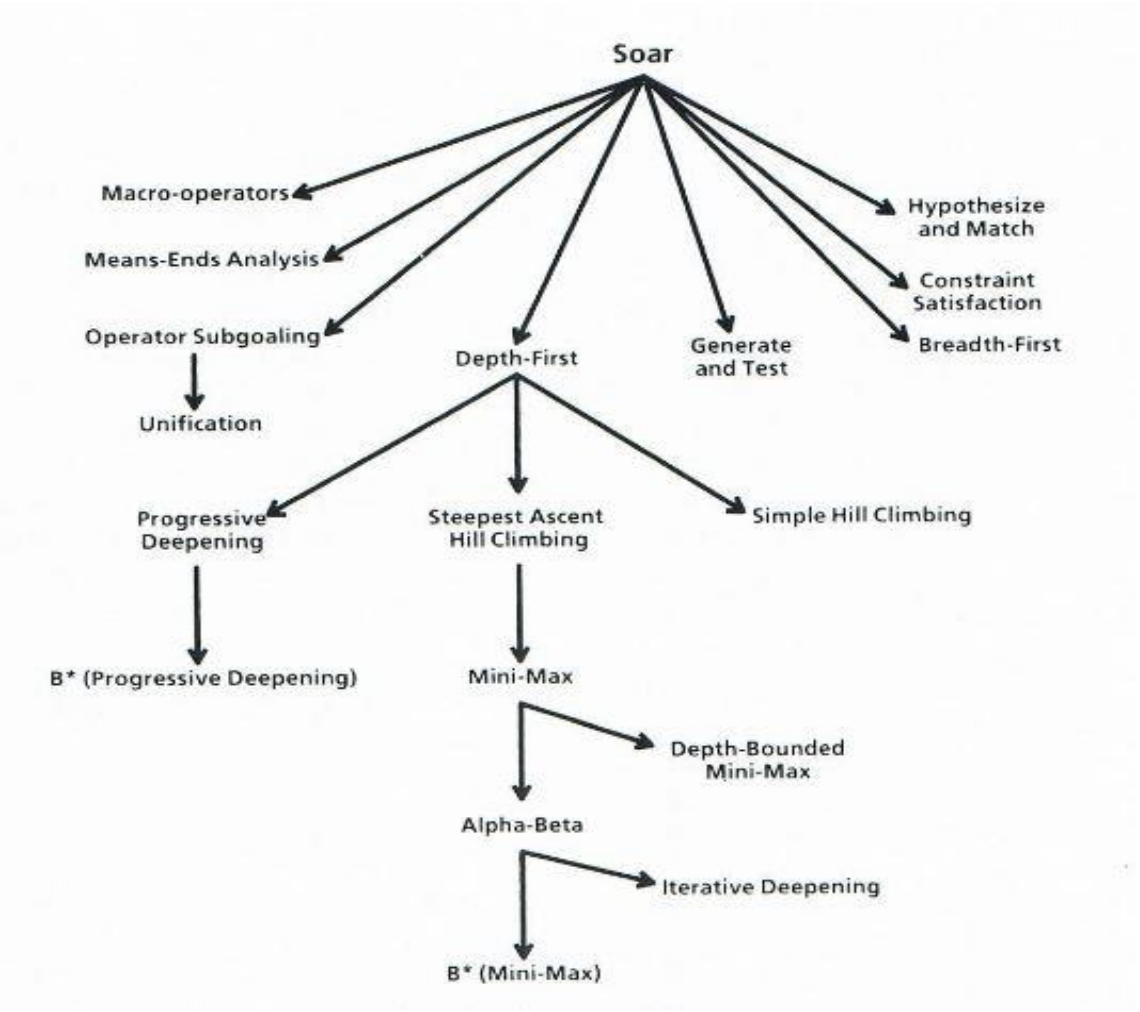
- In order to act, Soar must have knowledge of that domain (either given to it or learned)
- Domain knowledge can be divided into two categories:
  - Basic problem space knowledge:

    Definitions of the state representation, the “legal move” operators, their applicability conditions, and their effects
  - Control knowledge:

    Give guidance on choosing what to do, such as heuristics for solving problems in the domain

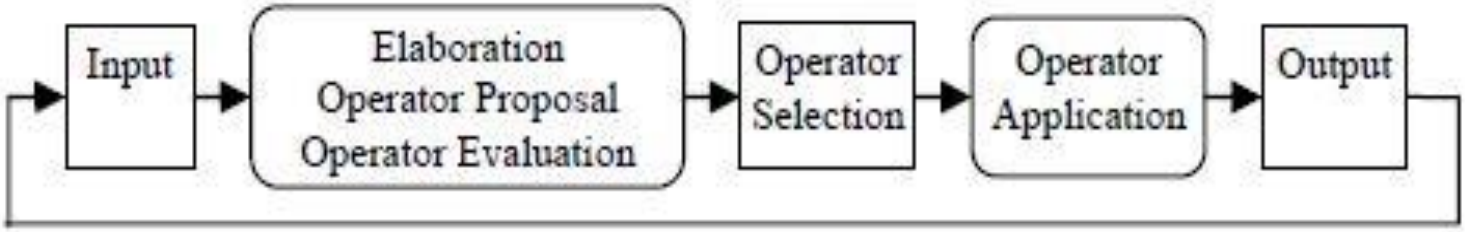
# Problem Solving Methods

- Universal *weak methods* realized in Soar
- Weak methods require relatively *little knowledge* about the task
- All the standard weak methods are built on *heuristic search*



# Processing Cycle and Impasse

- Once the situation is set up and running, the main activity is the repeated *selection* and then *application* of one operator after another




- If something prevents this processing cycle, e.g.
  - Soar knows of no operators to apply to that state (i.e. *no-change impasse*)
  - It knows of several, but has no knowledge of how to choose? (i.e. *tie impasse*)

In such cases, an *impasse*（僵局）occurs

# Learning Mechanism: Chunking

- In Soar, *subgoals* are created *only* in response to impasse
- Resolving an *impasse* leads to learning
- Soar includes a simple, uniform learning mechanism, called *chunking*（组块）
- A *chunk* is a new production rule that summarizes the processing that was needed to resolve the *impasse*
- Whenever a result is returned from an *impasse*, a new rule is learned connecting the relevant parts of the pre-impasse situation with the result — next time a sufficiently similar situation occurs, the impasse is *avoided*

# Many Applications

Work successfully in real-world agent-based applications:

- Computer games @ UoM
- Modeling social processes
- Learning by observation
- Virtual agents/humans – Immersive training
- Simulating emotions
- Electronic elves project – Teamwork @ USC
- Development tools
- Making Soar articulate / High-level language

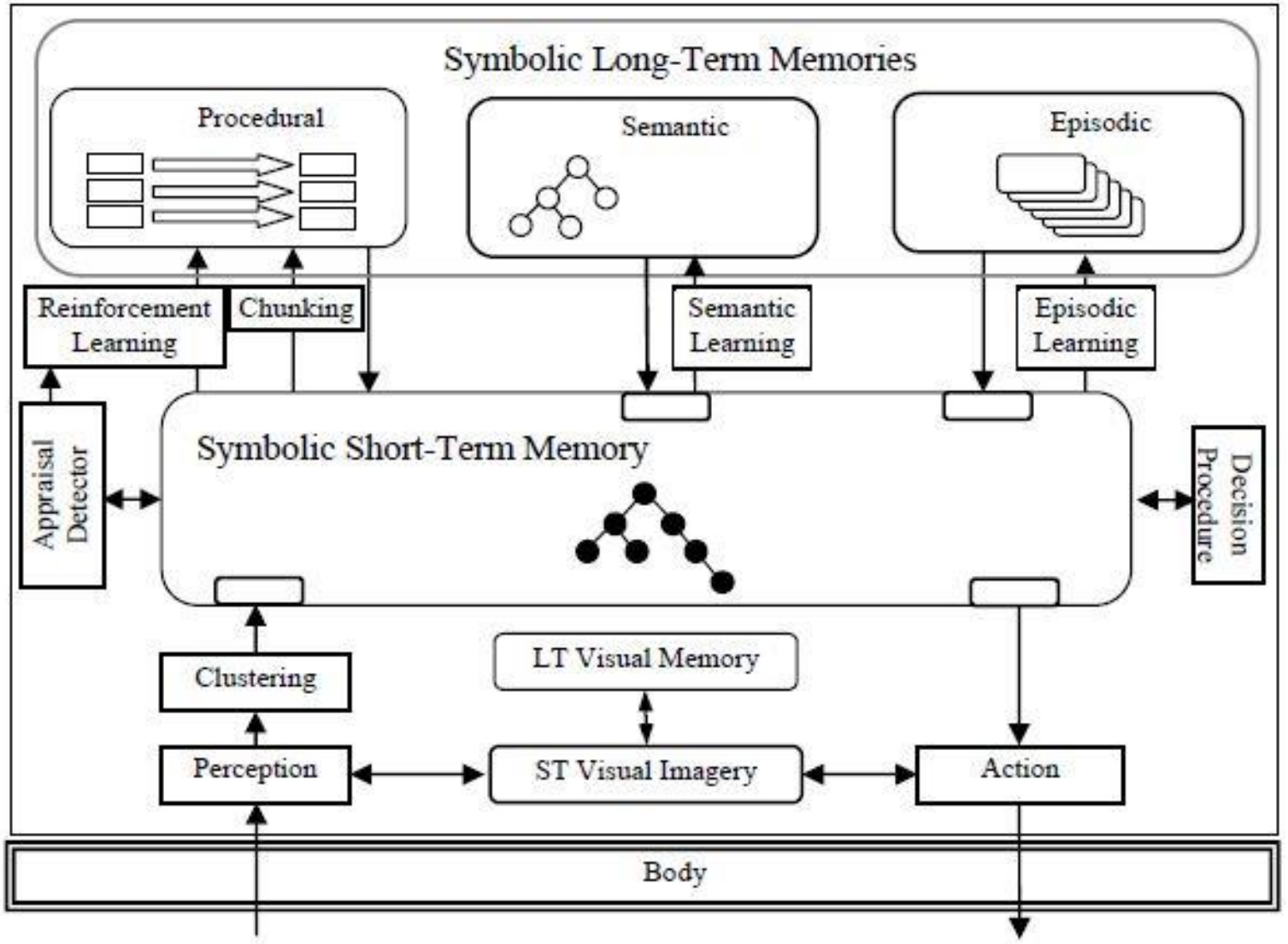
➢ Active research community with *yearly* workshop

*43*nd Soar Workshop in June, 2023

# Soar: Pros and Cons

- Soar remains the exemplar as a candidate unified theory of cognition
- Problem solving and chunking mechanisms are tightly *intertwined*
- Soar comes with access to Java
  adding user-written functions easier; debugging; rapid prototyping of GUIs
- Learning has some problems
  *overgeneralization* of chunks; *expensive* chunks
- Multiple goals not naturally supported
- Non-symbolic I/O not naturally supported

# Soar 9 Extension



# Course Summary

Cover two important levels of agents:

- Autonomous agents (micro level)
  - BDI model
  - Planning and practical reasoning
  - Learning
  - Cognitive modeling
- Multi-agent systems (macro level)
  - Modeling other agents (plan recognition)
  - Agent communication
  - Game theory / multi-agent learning
  - Agents work together (teamwork model, cooperation, coordination and negotiation)
  - Social modeling and reasoning

# 其它软件和工具

- NetLogo (Northwestern University)
  https://ccl.northwestern.edu/netlogo/
- Repast (Argonne National Lab)
  https://repast.github.io/
- Jade (Telecom Italia Lab)
  https://jade.tilab.com/
- GAMA
  https://gama-platform.github.io/
- Jason
  https://jason.sourceforge.net/wp/
- Swarm (Santa Fe Institute)
  http://www.swarm.org/wiki/Swarm_main_page

# References


- R. A. Brooks. *A Robust Layered Control System for a Mobile Robot*. IEEE Journal of Robotics and Automation, RA-2(1):14-23, 1986
- I. A. Ferguson. *Integrated Control and Coordinated Behaviour: A Case for Agent Models*. In: M. Wooldridge and N. R. Jennings (Eds.), Intelligent Agents: Theories, Architectures and Languages (LNAI 890), pp.203-218. Springer, 1995
- J. P. Müller. *A Cooperation Model for Autonomous Agents*. In: J. P. Müller, M. Wooldridge and N. R. Jennings (Eds.), Intelligent Agents III (LNAI 1193), pp.245-260. Springer, 1997
- J. E. Laird, P. S. Rosenbloom and A. Newell. *Soar: An Architecture for General Intelligence*. In: P. S. Rosenbloom, J. E. Laird and A. Newell (Eds.), The Soar Papers: Research on Integrated Intelligence. The MIT Press, 1993
- J. E. Laird. *Extending the Soar Cognitive Architecture*. Proceedings of the First Conference on Artificial General Intelligence, 2008

End.