Autocatalytic Endogenous Reflective Architecture

Kristinn R. Thórisson

Associate Professor, School of Computer Science, Reykjavik University
Member, Center for Analysis and Design of Intelligent Agents, Reykjavik U.
Managing Director, Icelandic Institute for Intelligent Machines, Reykjavik, Iceland

AGI Systems

should be able to:

- learn to be able to perform a host of unanticipated vastly different tasks
- adapt to vastly new circumstances
- actively acquire new knowledge for the above, as needed, of their own accord

These Systems

should be able to:

- learn to be able to perform a host of *unanticipated* vastly different tasks
- adapt to vastly new circumstances
- actively acquire new knowledge for the above, as needed, of their own accord

These Systems

should be able to:

- learn to be able to perform a host of *unanticipated vastly different* tasks

- adapt to vastly new circumstances

- actively acquire new knowledge for the above, as needed, of their own accord

These Systems

should be able to:

- learn to be able to perform a host of *unanticipated vastly different* tasks
- adapt to vastly new circumstances
- actively acquire new knowledge for the above, as needed, of their own accord

These Systems

should be able to:

- learn to be able to perform a host of *unanticipated vastly different* tasks
- adapt to vastly new circumstances
- actively acquire new knowledge for the above, as needed, of their own accord

These Systems

should be able to:

- learn to be able to perform a host of *unanticipated vastly different* tasks
- adapt to vastly new circumstances
- actively acquire new knowledge for the above, as needed, of their own accord

What AGIs Need:

- Powerful adaptation mechanisms
  - require: Cognitive growth capabilities
    - which calls for: Self-inspection capabilities
      - which requires: a semantically and operationally closed programming language

Self-Programming: The Key to Cognitive Growth

- If a system could inspect its own operation, it could possibly use experience to find better ways to operate
  - i.e. improve its own operation in light of acquired experience
- But this would mean: architecture and algorithms must be manipulatable by the system itself
- This calls for ***reflection***: the ability of the system to self-inspect

Why is Self-Programming Needed?

- Cognitive growth = most powerful path to achieve adaptation
- Self-programming: a way to achieve cognitive growth, and hence **autonomy**
- Reduces or eliminates human control and intervention
  - Less operational cost
  - Highly autonomous systems more reusable
- Building learning, flexible, self-adaptive systems that can operate without complete pre-specification of tasks
- Adaptation must be “allways on” in AGIs

What Programming Language?

- *All present programming languages designed to be used by human-level intelligences*

→ Current programming paradigms cannot support cognitive development in artificial systems

Autocatalytic Endogeneous Reflective Architecture

- Autocatalytic
  - Operation exclusively event-driven
  - Attention (resource control), learning, and planning catalyze each other
  - Operation is model-based: chains of models invoking models control the events in the system

Autocatalytic Endogeneous Reflective Architecture

- Endogenous
  - All knowledge generated from a tiny seed
  - All sub-goals auto-generated from top-level goal (provided in seed)
  - All knowledge acquisition driven endogenously (but does not preclude exogeneous knowledge provision)

Autocatalytic Endogeneous Reflective Architecture

- Reflective
  - Explicit traces of system operation allows building models of it
  - The system's code is parsable and readable by the system due to special programming language (Replicode)
  - Models of self-operation enables self-control (aka meta-control).

Methodology

Constructivist Methodology

Con - struct - ivist A.I.: Self-constructive artificial intelligence systems with general knowledge acquisition skills; systems develop from a seed specification; capable of learning to perceive and act in a wide range of novel tasks, situations, and domains

HUMANOBS Project

- AERA: Broad-scope AGI-aspiring architecture
  - general-purpose learning in dynamic worlds domain independence

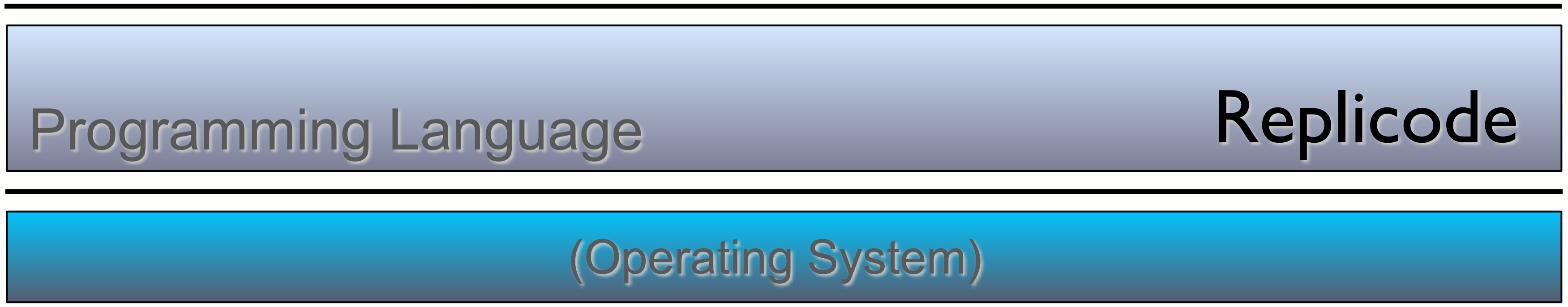
- Demonstrating transversal cognitive skills
  - at multiple levels of granularity and abstraction
    - system-wide learning
    - temporal grounding
    - observation and imitation of complex realtime events
    - attention
    - inference, abstraction ... and more

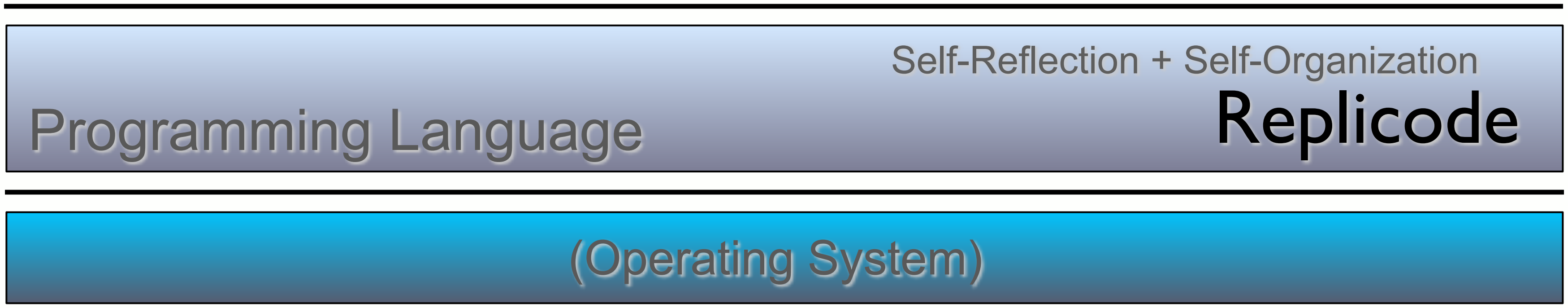
A New Kind of A.I.

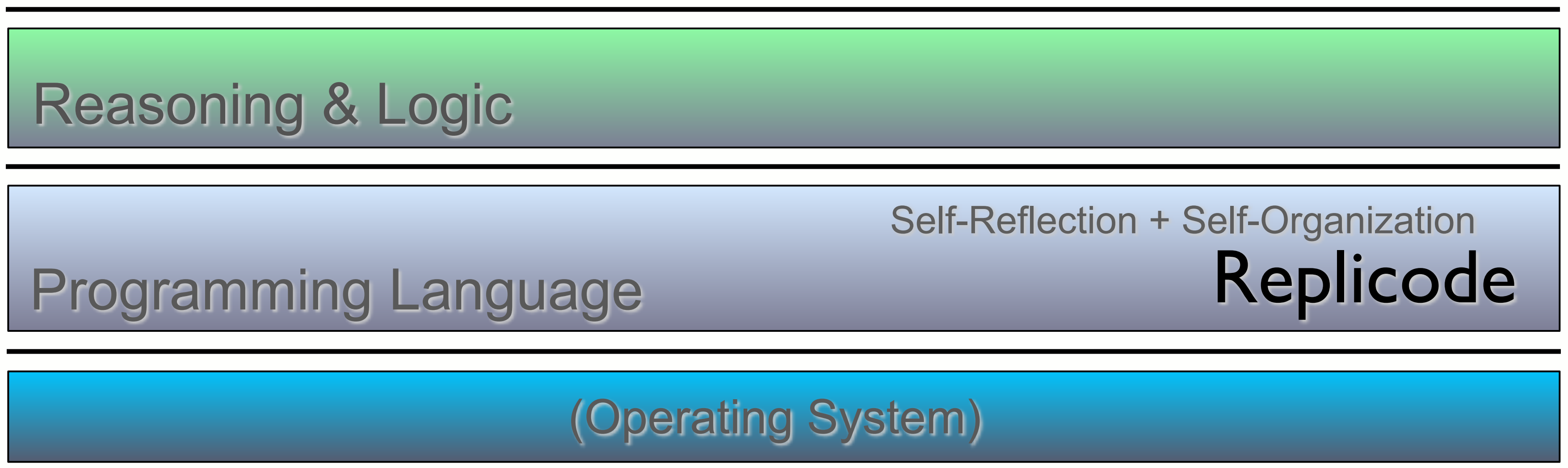
- Acquires knowledge autonomously
    - starting with observation
    - honed through abduction and induction
- Self-modeling inherent in system operation
- Continuous learning
- Domain-independent learning

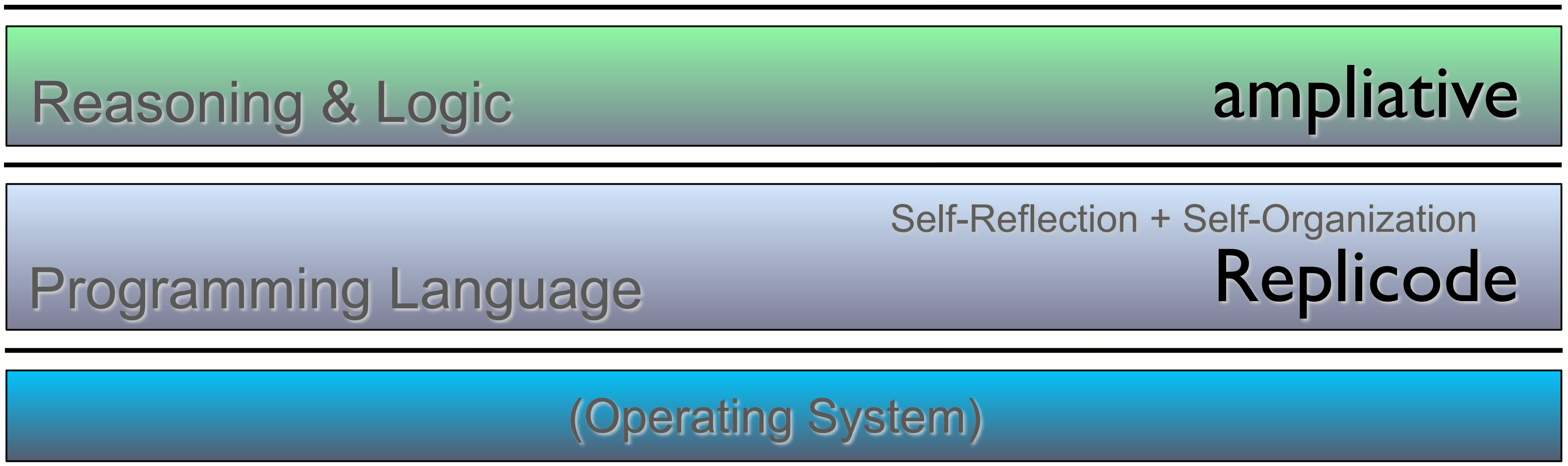
Programming Language

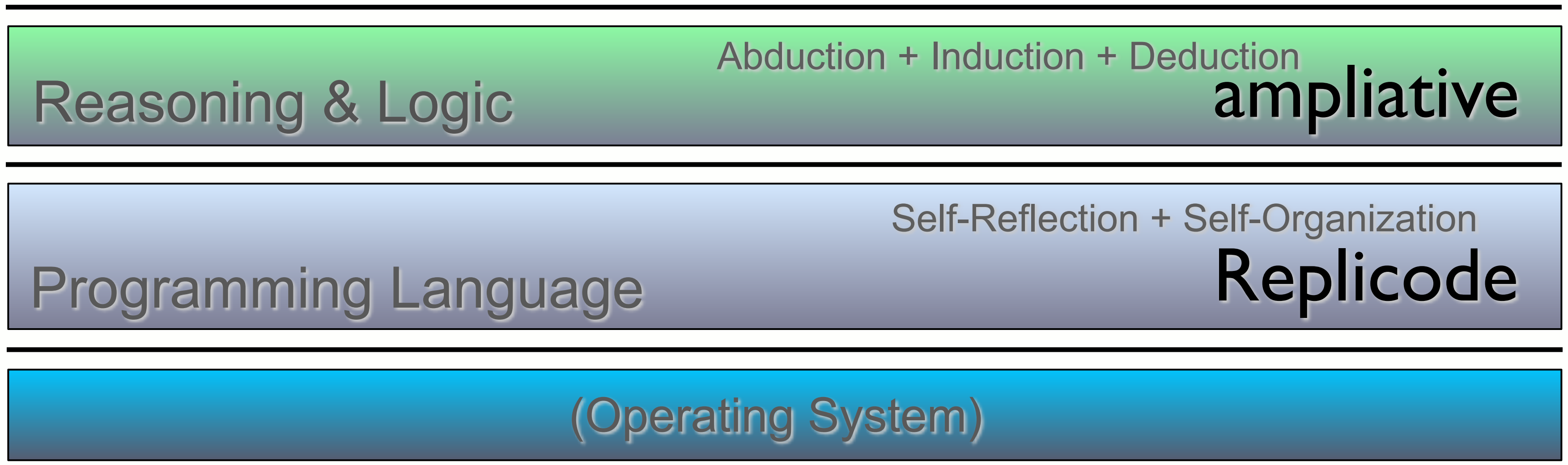
(Operating System)

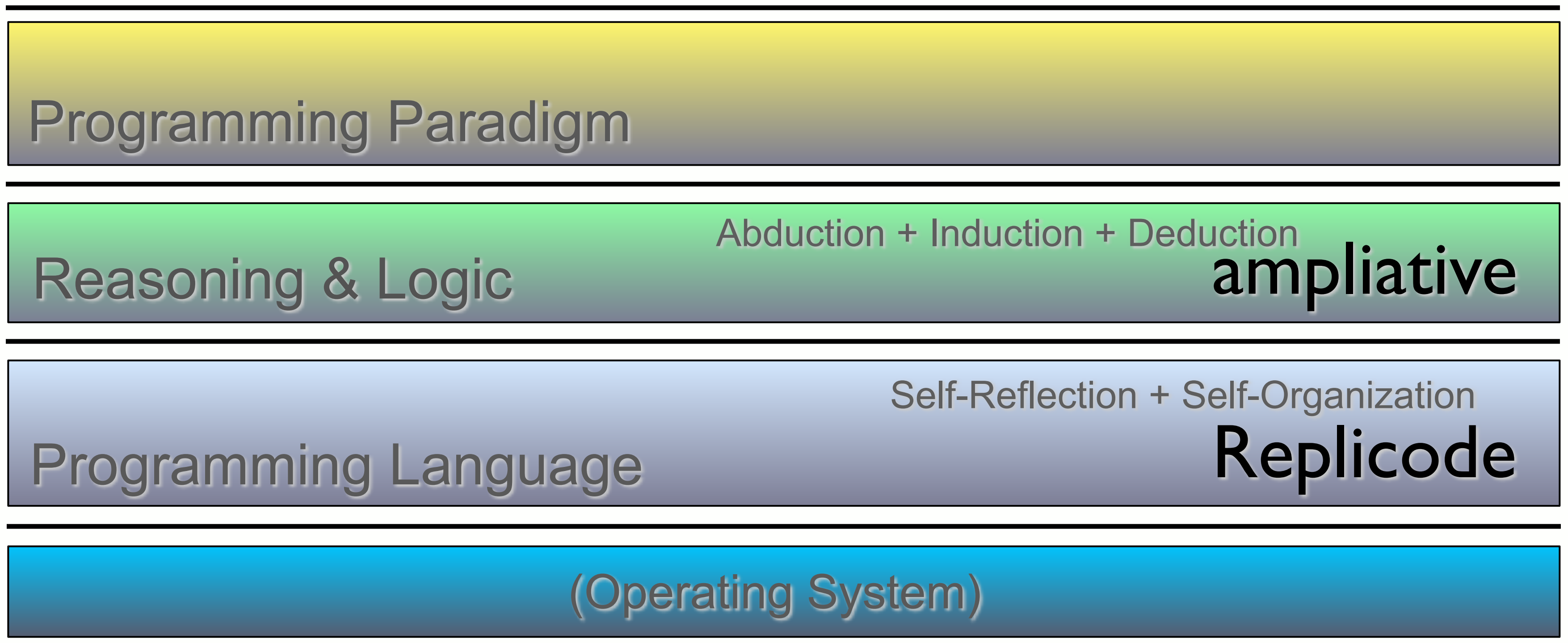

Programming Language
Replicode
(Operating System)

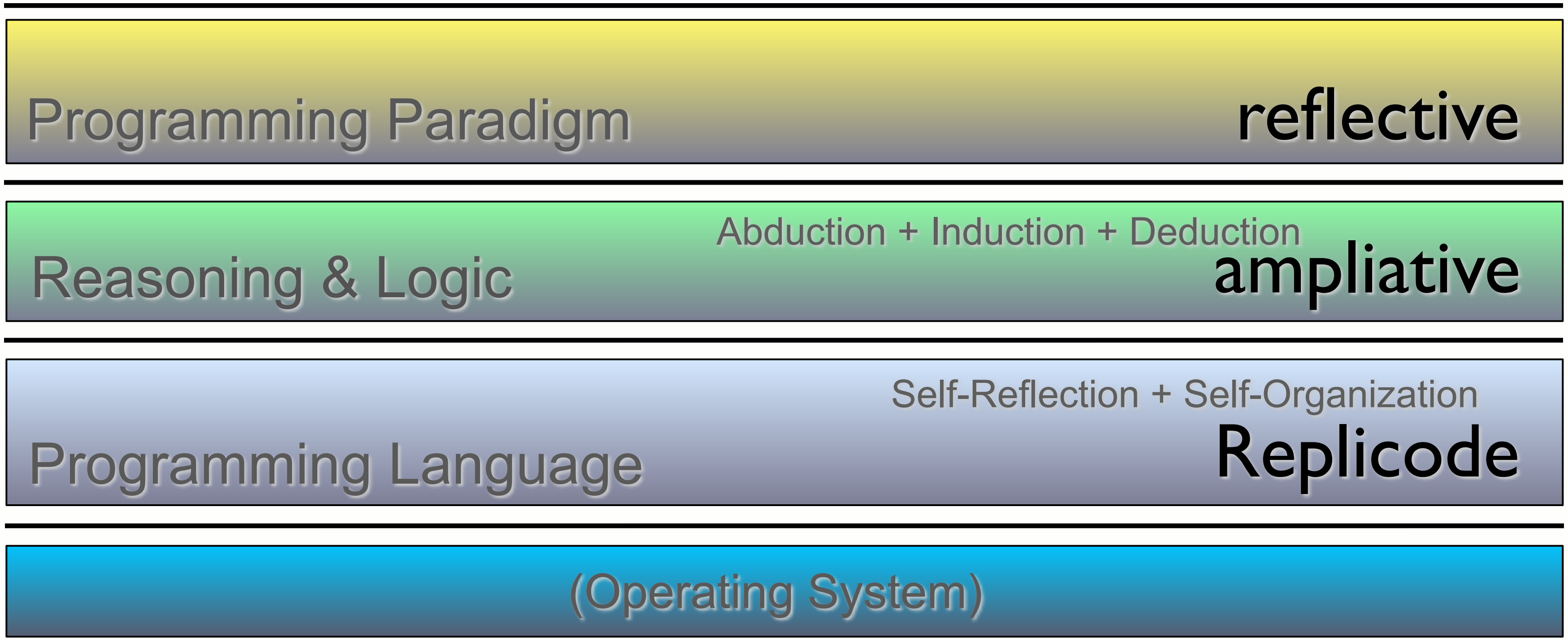

Self-Reflection + Self-Organization
Replicode
Programming Language
(Operating System)

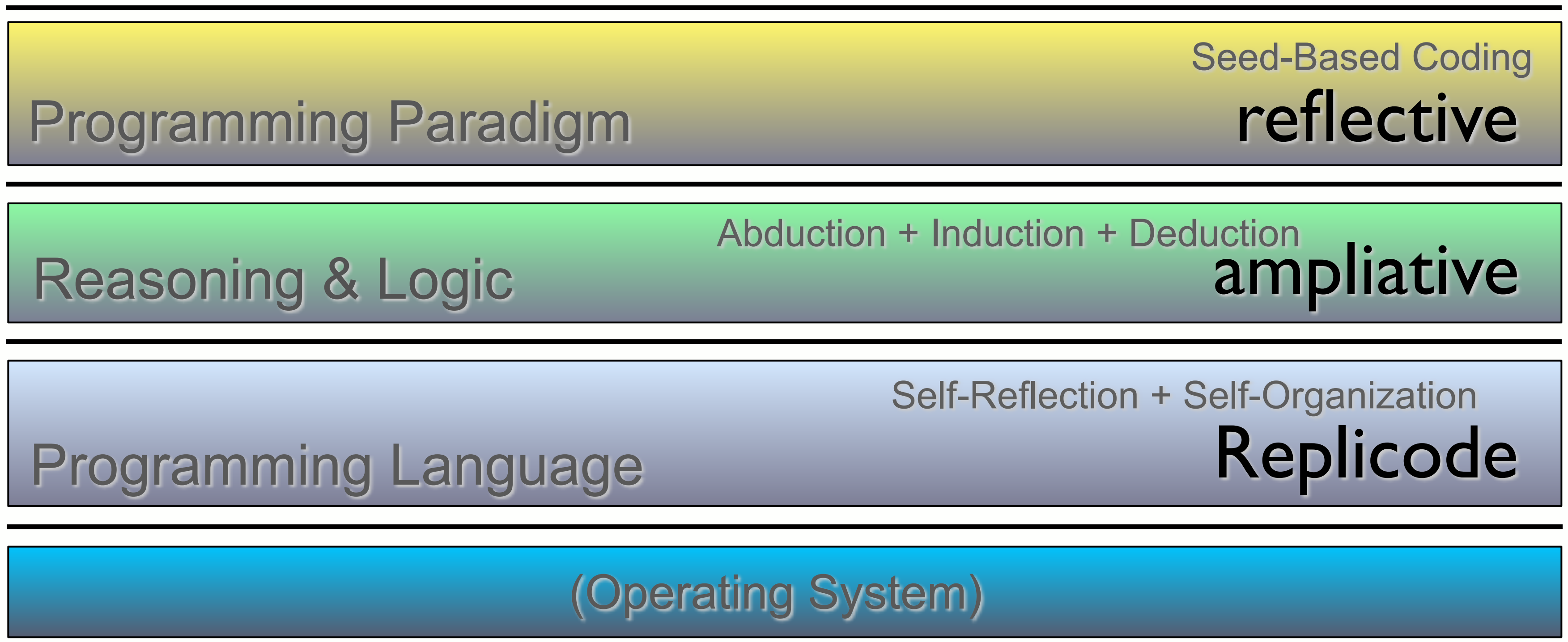

Reasoning & Logic
Self-Reflection + Self-Organization
Replicode
Programming Language
(Operating System)

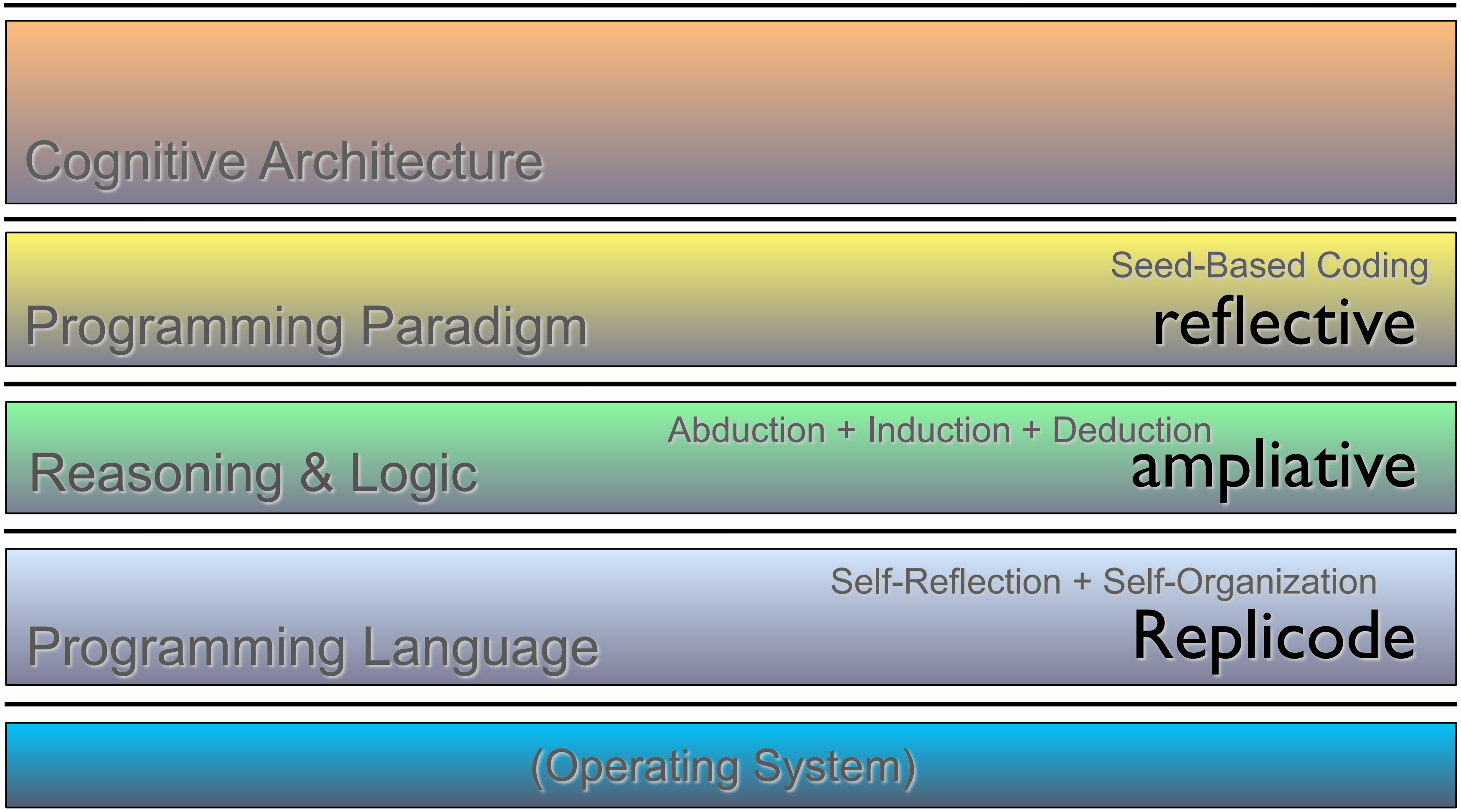

Reasoning & Logic
ampliative
Self-Reflection + Self-Organization
Programming Language
Replicode
(Operating System)

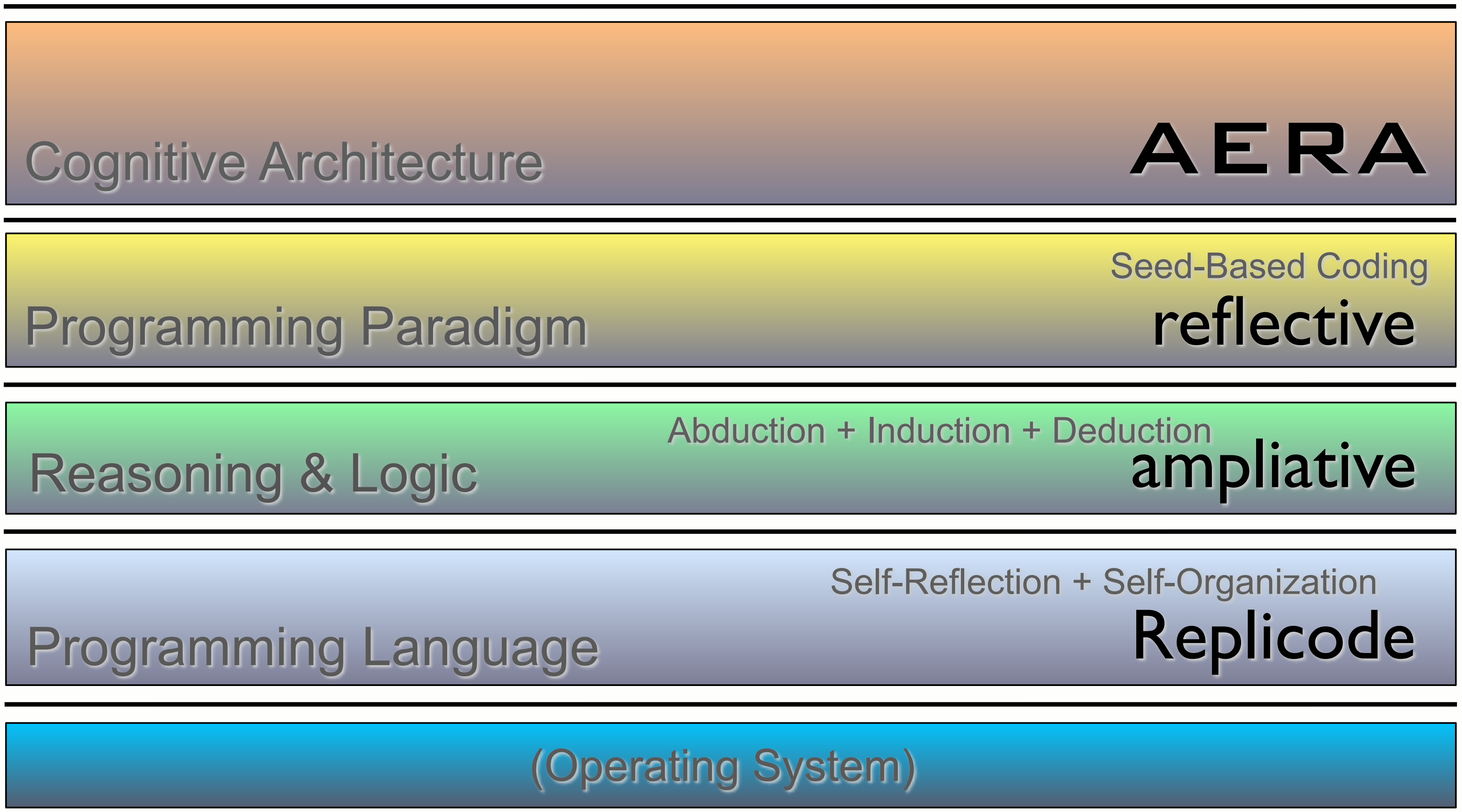

Reasoning & Logic
Abduction + Induction + Deduction
ampliative
Programming Language
Self-Reflection + Self-Organization
Replicode
(Operating System)

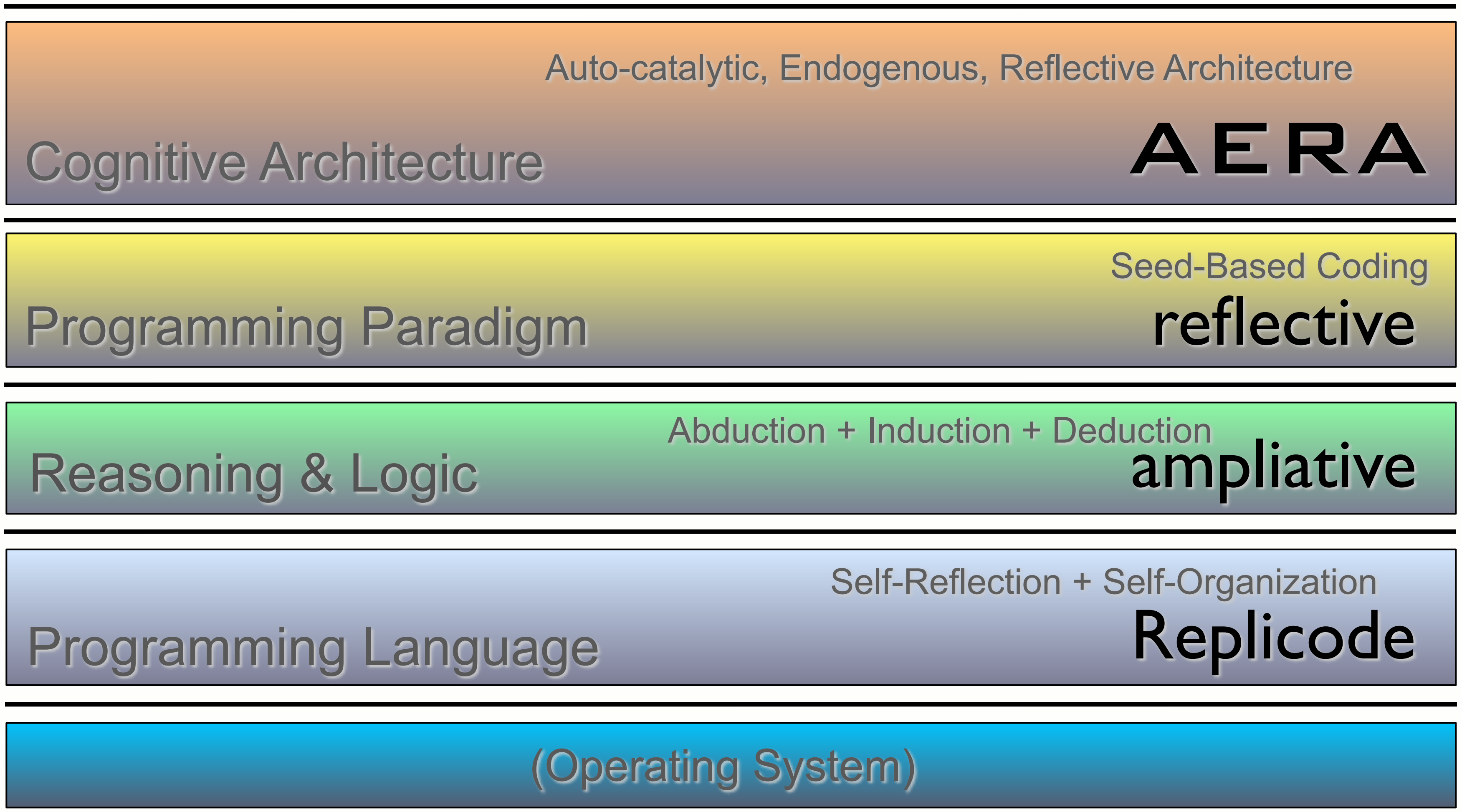

Programming Paradigm
Reasoning & Logic
Abduction + Induction + Deduction
ampliative
Programming Language
Self-Reflection + Self-Organization
Replicode
(Operating System)


Programming Paradigm
reflective
Reasoning & Logic
Abduction + Induction + Deduction
ampliative
Programming Language
Self-Reflection + Self-Organization
Replicode
(Operating System)


Seed-Based Coding
Programming Paradigm
reflective
Abduction + Induction + Deduction
Reasoning & Logic
ampliative
Self-Reflection + Self-Organization
Programming Language
Replicode
(Operating System)


Cognitive Architecture
Programming Paradigm
Seed-Based Coding
reflective
Reasoning & Logic
Abduction + Induction + Deduction
ampliative
Programming Language
Self-Reflection + Self-Organization
Replicode
(Operating System)


Cognitive Architecture
AERA
Seed-Based Coding
Programming Paradigm
reflective
Abduction + Induction + Deduction
Reasoning & Logic
ampliative
Self-Reflection + Self-Organization
Programming Language
Replicode
(Operating System)


Auto-catalytic, Endogenous, Reflective Architecture
Cognitive Architecture
AERA
Seed-Based Coding
Programming Paradigm
reflective
Abduction + Induction + Deduction
Reasoning & Logic
ampliative
Self-Reflection + Self-Organization
Programming Language
Replicode
(Operating System)

Methodology

Cognitive Architecture

Auto-catalytic, Endogenous, Reflective Architecture

AERA

Programming Paradigm

Seed-Based Coding

reflective

Reasoning & Logic

Abduction + Induction + Deduction

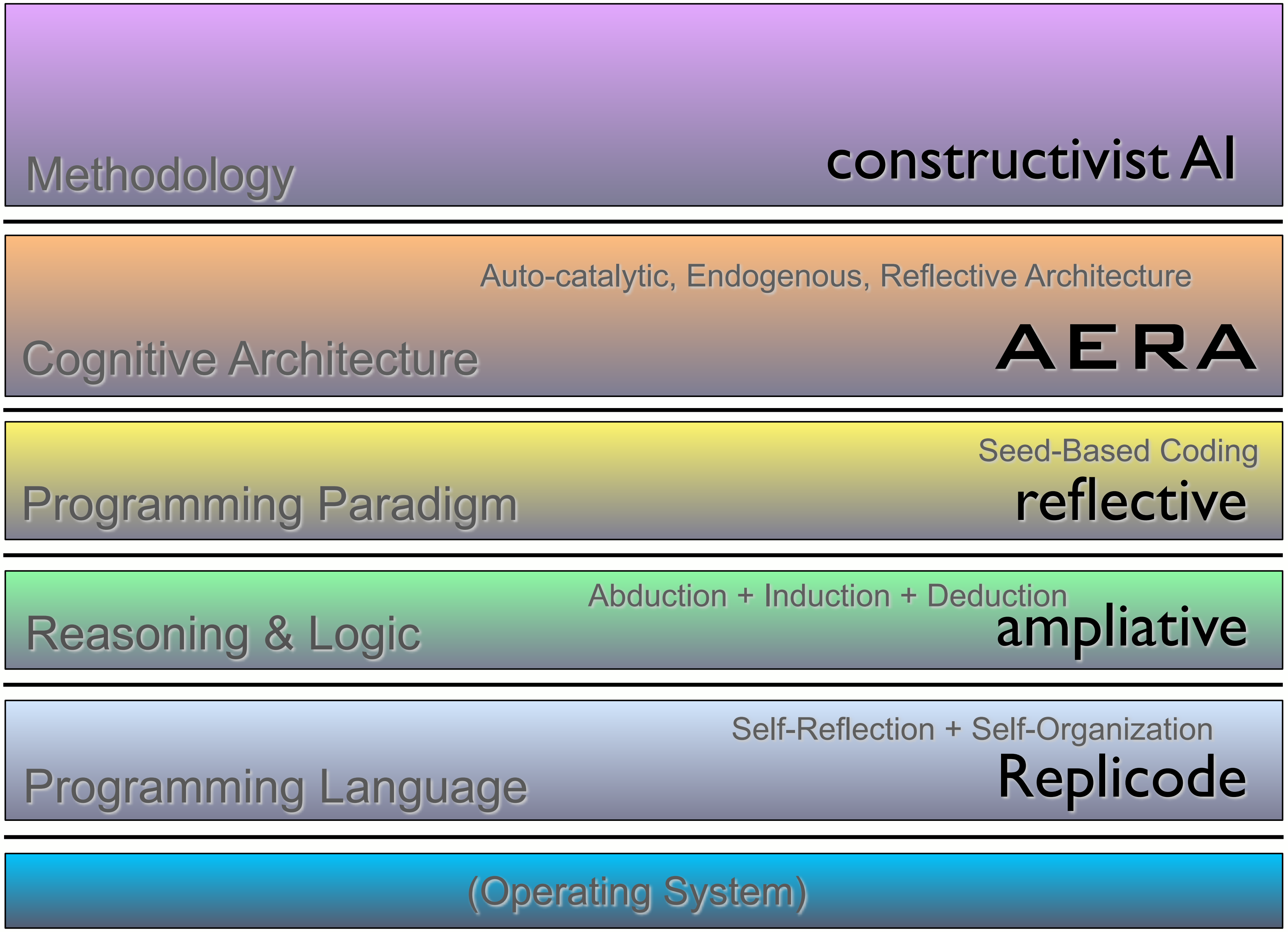
ampliative

Programming Language

Self-Reflection + Self-Organization

Replicode

(Operating System)


Methodology
constructivist AI
Auto-catalytic, Endogenous, Reflective Architecture
Cognitive Architecture
AERA
Seed-Based Coding
Programming Paradigm
reflective
Abduction + Induction + Deduction
Reasoning & Logic
ampliative
Self-Reflection + Self-Organization
Programming Language
Replicode
(Operating System)

Methodology

Inspired by Piaget's theory of cognitive development

constructivist AI

Cognitive Architecture

Auto-catalytic, Endogenous, Reflective Architecture

AERA

Programming Paradigm

Seed-Based Coding

reflective

Reasoning & Logic

Abduction + Induction + Deduction

ampliative

Programming Language

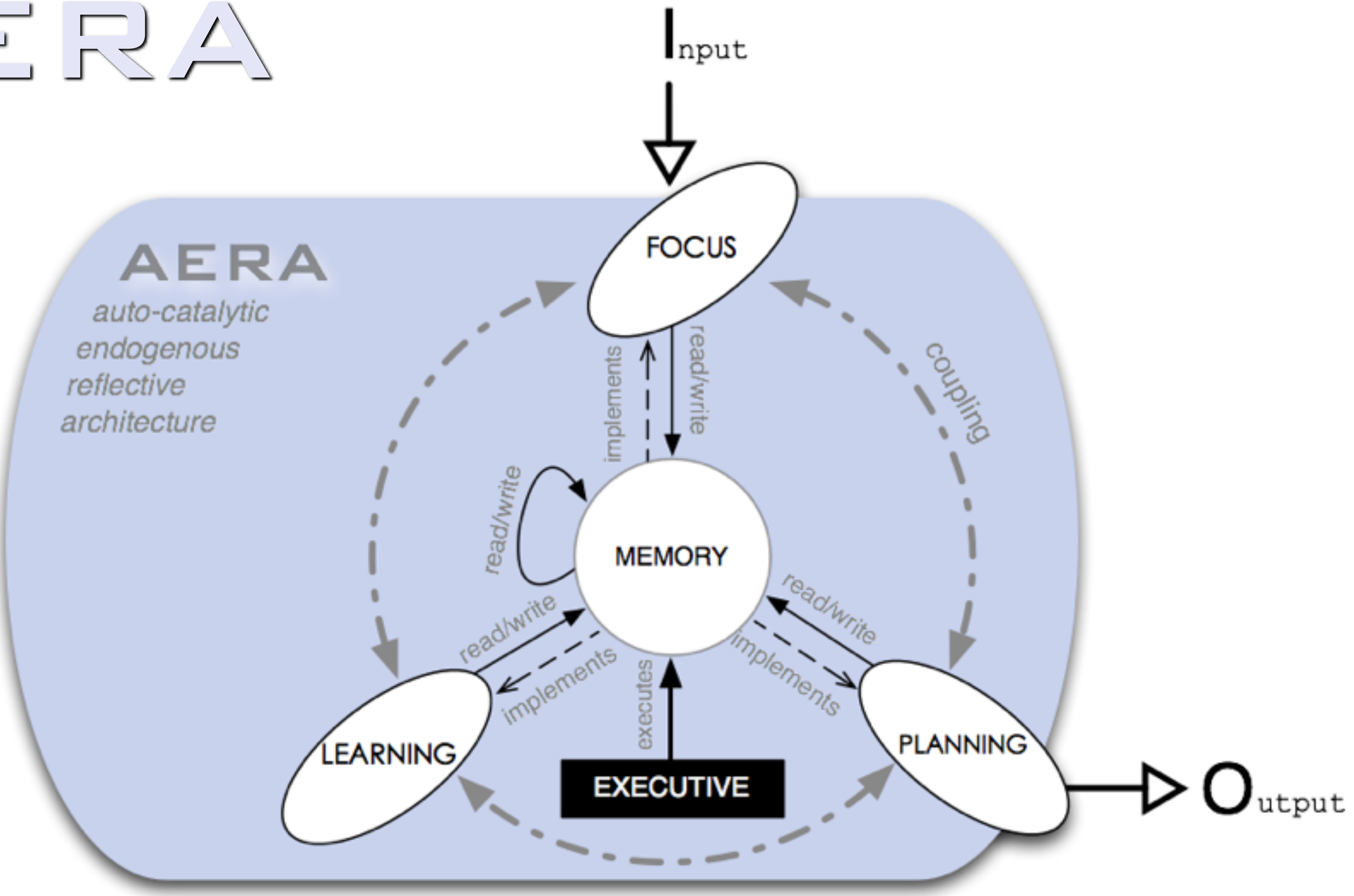
Self-Reflection + Self-Organization

Replicode

(Operating System)

AERA

AERA


Input
AERA
auto-catalytic
endogenous
reflective
architecture
FOCUS
implements
read/write
coupling
read/write
MEMORY
read/write
implements
executes
read/write
implements
LEARNING
EXECUTIVE
PLANNING
Output

Holistic Design

Desired operation of the system results indirectly from the inter-operation of a multitude of general-purpose underlying processes

No component called “learning” or “planner” and so on. Instead, learning and planning are emergent processes that result from the same set of system-wide functions

High-level processes (like planning and learning) influence each other (positively and negatively): they are dynamically coupled, as they both result from the execution of the same knowledge - the very core of the system, its models

Reflectivity

- A system must know **what** it is doing, **when,** and **at what cost**. Enforcing explicit traces of the system's operation allows the building of models of said operation, which is needed for self-control (i.e. meta-control)
- The architecture shall be applicable to itself, i.e. a control system for the system shall be implementable **in the same way the system is in the domain**. This principle must be followed if one wants to implement "integrated cognitive control"

Uniform Operations

- All operations are controllable in a uniform way, using mechanisms as simple as possible

- More elaborate control schemes are learned by a control system
  - Enables the system to incrementally move towards higher efficiency, more capabilities, and more targeted and focused operation

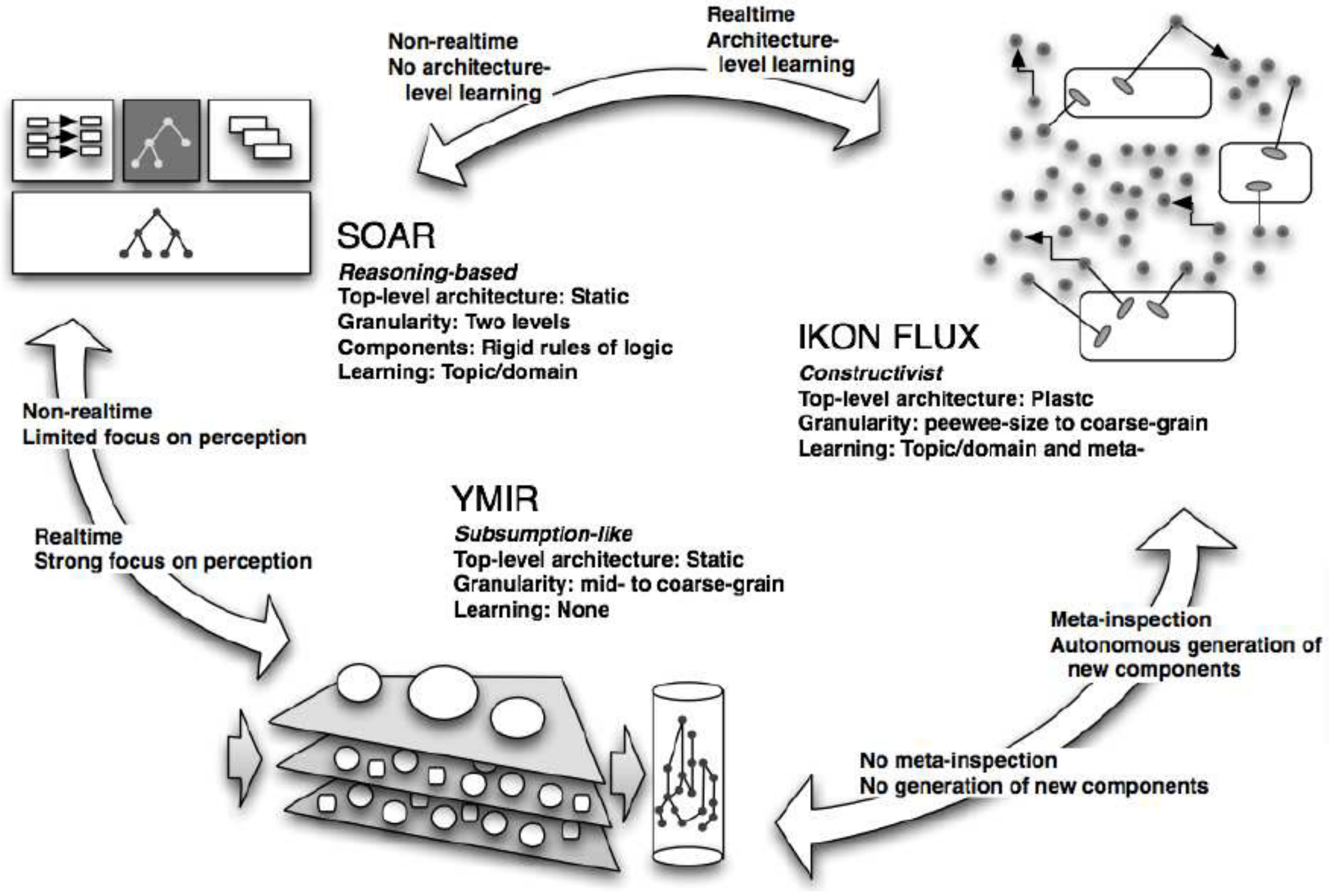
Low Granularity of Knowledge Representation

- The encoding of knowledge shall be short and concise
- All primitive operations of the system shall focus on one task and take as little time as possible, keeping in mind that higher-level operations result from the coupling of a multitude of said primitive operations
- This principle aims at preserving plasticity – the capability of implementing small, incremental changes in the system
  - this is one of the key requirements for the architecture and underpins our entire research avenue

Deep Handling of Time

- Accounts for time at **all stages** of computation and **at all scales** - from the scale of an individual operation (e.g. performing a reduction) to the scale of a collective operation (e.g. achieving a goal)
- An essential requirement for a system that (a) has to perform in the real world and, (b) has to model its own operation with regards to its expenditure of resources
- Time values considered as **intervals** to encode the variable precisions and accuracies to be expected in the real world
    - for example, sensors are not always performing at fixed frame rates and therefore the ability to model their operation is critical to ensure the reliable operation of their controllers and the models that depends on their input
- The precision for goals and predictions may vary considerably depending on both their time horizons and semantics

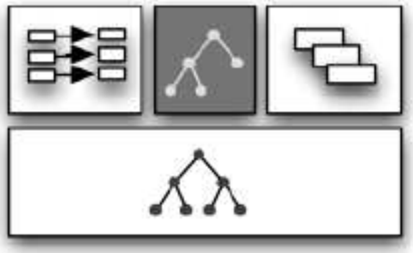
Qualitative comparison


Non-realtime
No architecture-level learning
Realtime
Architecture-level learning
SOAR
Reasoning-based
Top-level architecture: Static
Granularity: Two levels
Components: Rigid rules of logic
Learning: Topic/domain
IKON FLUX
Constructivist
Top-level architecture: Plastc
Granularity: peewee-size to coarse-grain
Learning: Topic/domain and meta-
Non-realtime
Limited focus on perception
Realtime
Strong focus on perception
YMIR
Subsumption-like
Top-level architecture: Static
Granularity: mid- to coarse-grain
Learning: None
Meta-inspection
Autonomous generation of new components
No meta-inspection
No generation of new components

Qualitative comparison

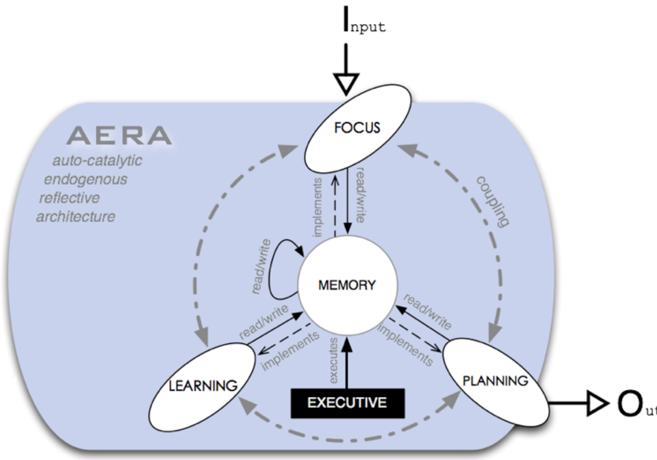
Non-realtime
No architecture-level learning

Realtime
Architecture-level learning



SOAR

Reasoning-based
Top-level architecture: Static
Granularity: Two levels
Components: Rigid rules of logic
Learning: Topic/domain



AERA

Constructivist
Top-level architecture: Plastc
Granularity: peewee-size to coarse-grain
Learning: Topic/domain and meta-

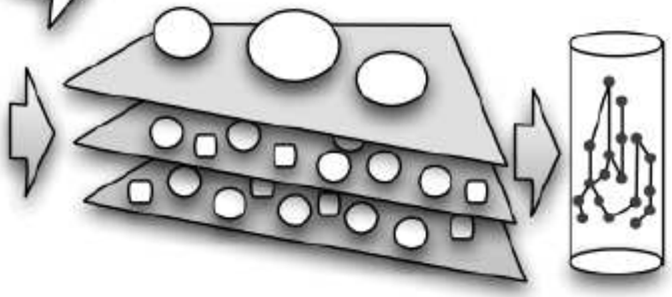
Non-realtime
Limited focus on perception

Realtime
Strong focus on perception

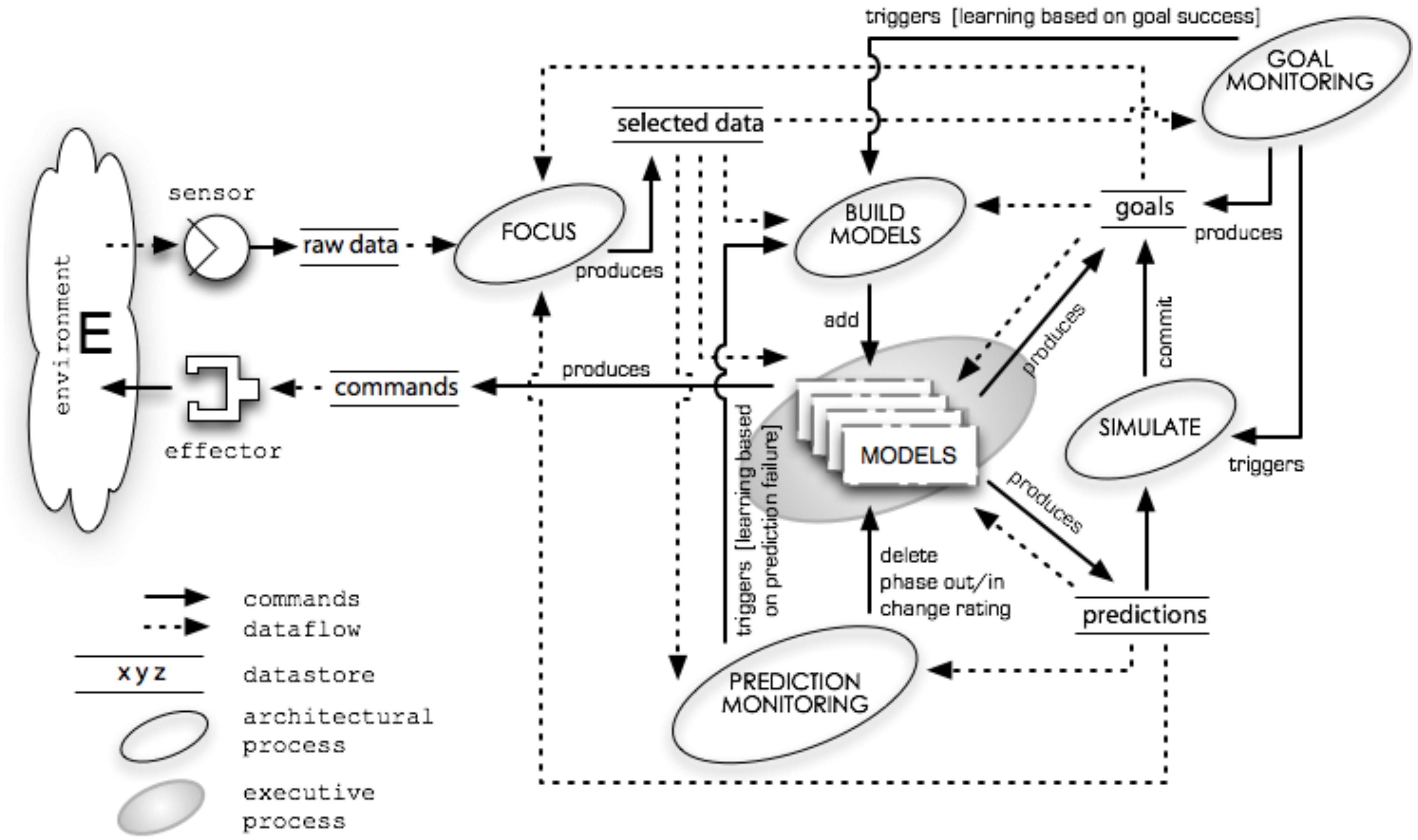
YMIR

Subsumption-like
Top-level architecture: Static
Granularity: mid- to coarse-grain
Learning: None

Meta-inspection
Autonomous generation of new components



No meta-inspection
No generation of new components



triggers [learning based on goal success]
GOAL MONITORING
selected data
sensor
raw data
FOCUS
produces
BUILD MODELS
goals
produces
environment
E
add
produces
commit
produces
commands
effector
MODELS
SIMULATE
triggers
triggers [learning based on prediction failure]
produces
delete
phase out/in
change rating
predictions
PREDICTION MONITORING
commands
dataflow
xyz
datastore
architectural process
executive process

How Models Are Built

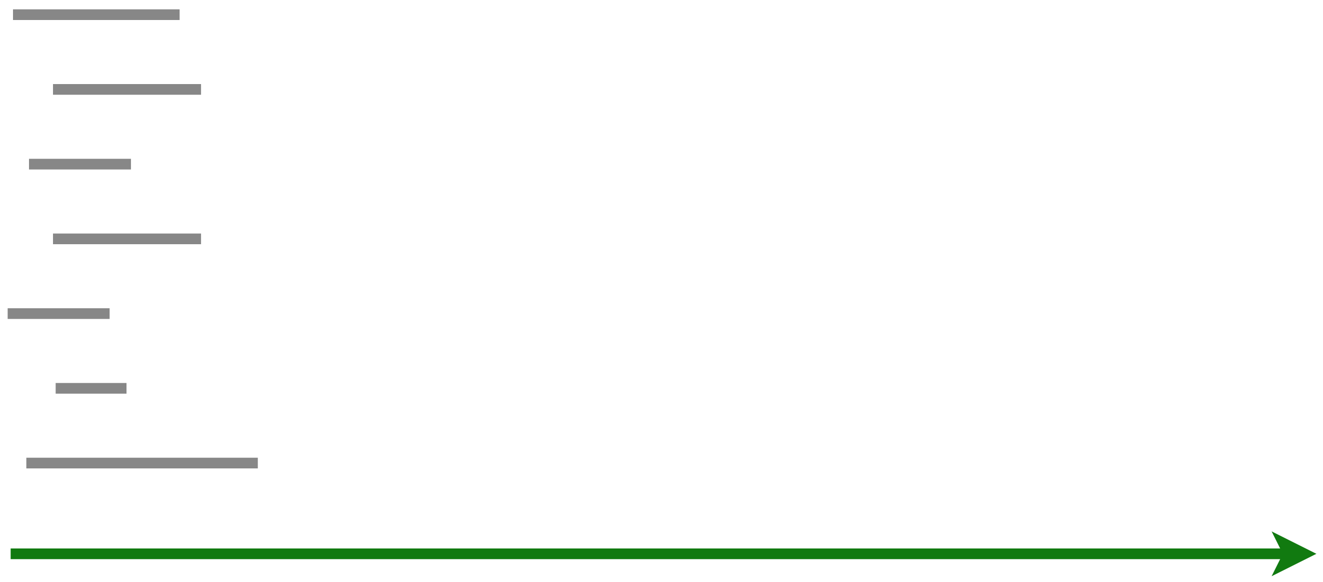
- Models are built by Targeted Pattern Extractors (TPX)
    - by observing events (external and internal)
    - proposing causal relationships between them
    - and representing these as cause-effect models

Learning About the World

Events produced in the real world

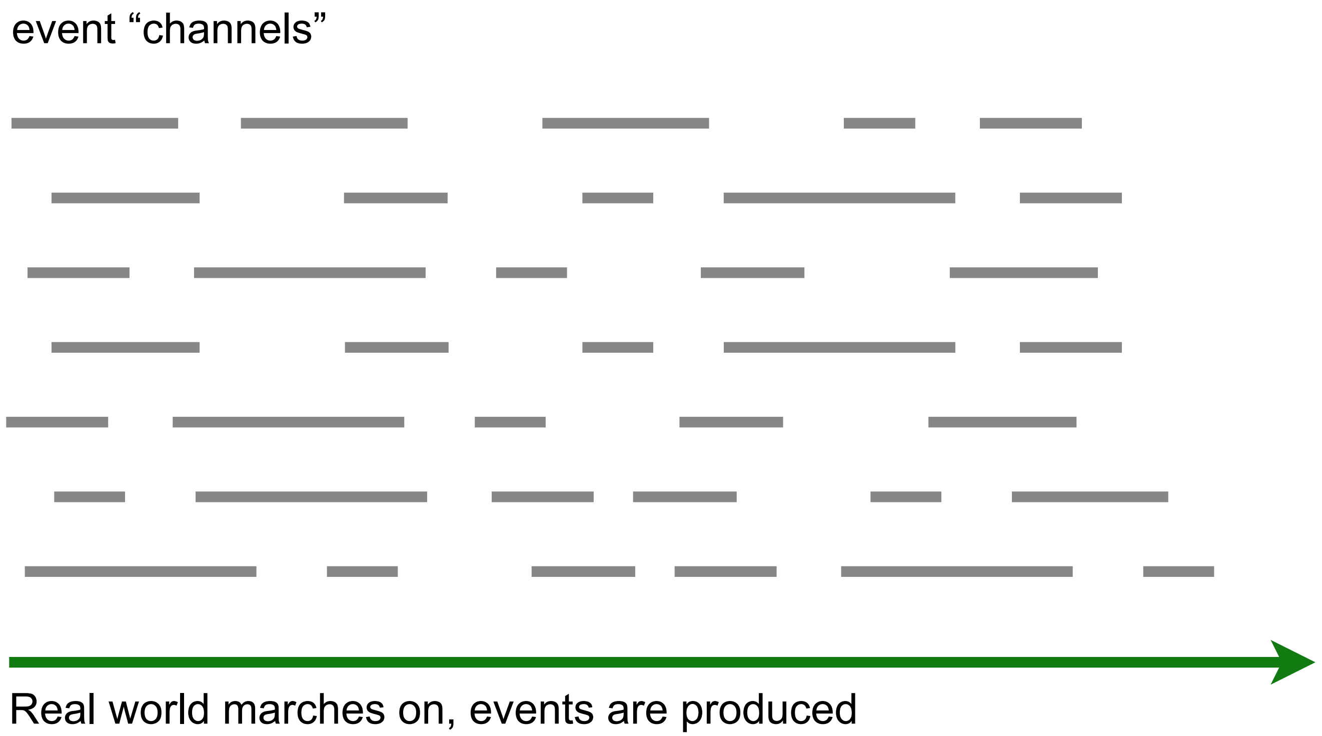
Learning About the World

event “channels”



Events produced in the real world

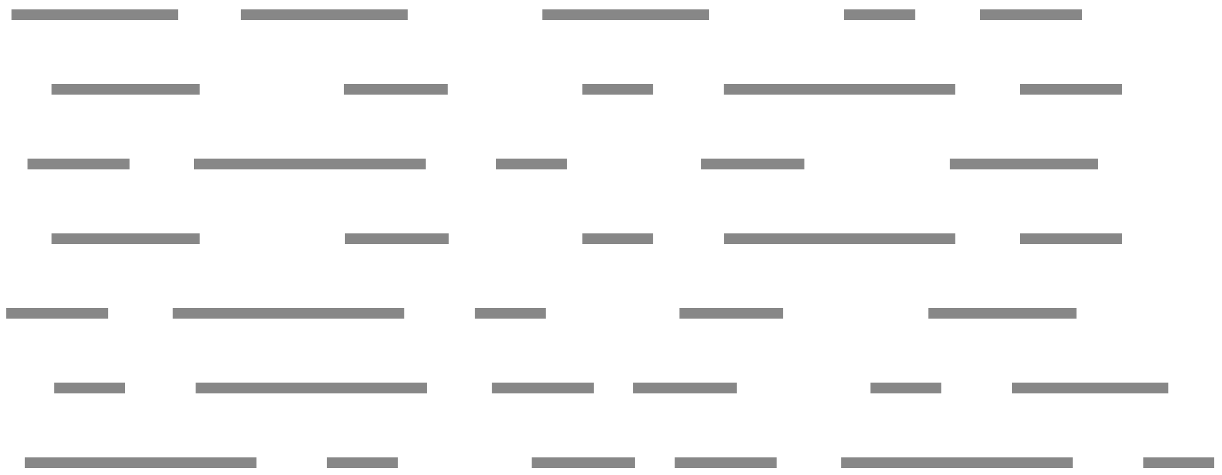
Learning About the World



Learning About the World

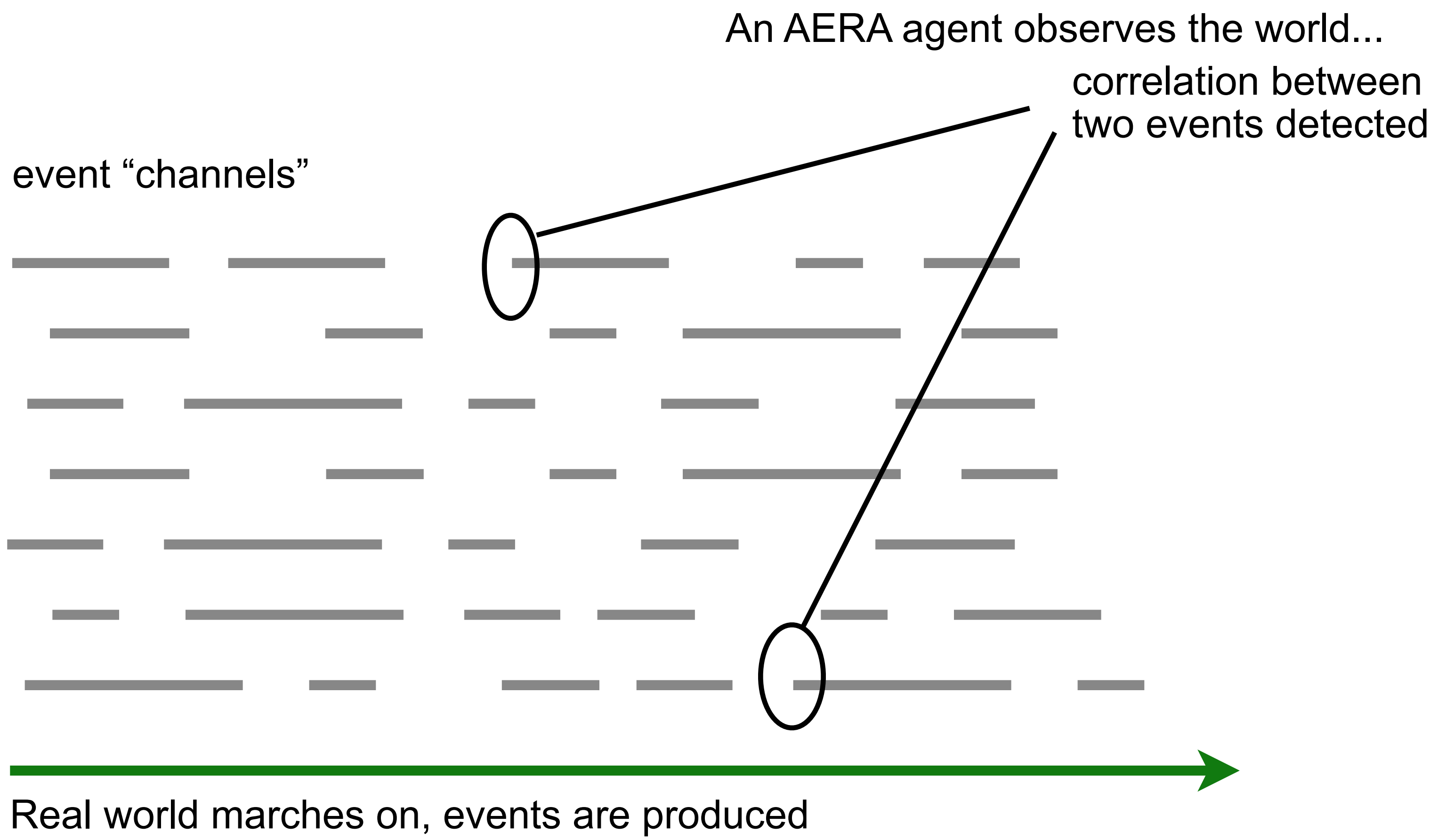
An AERA agent observes the world...

event “channels”

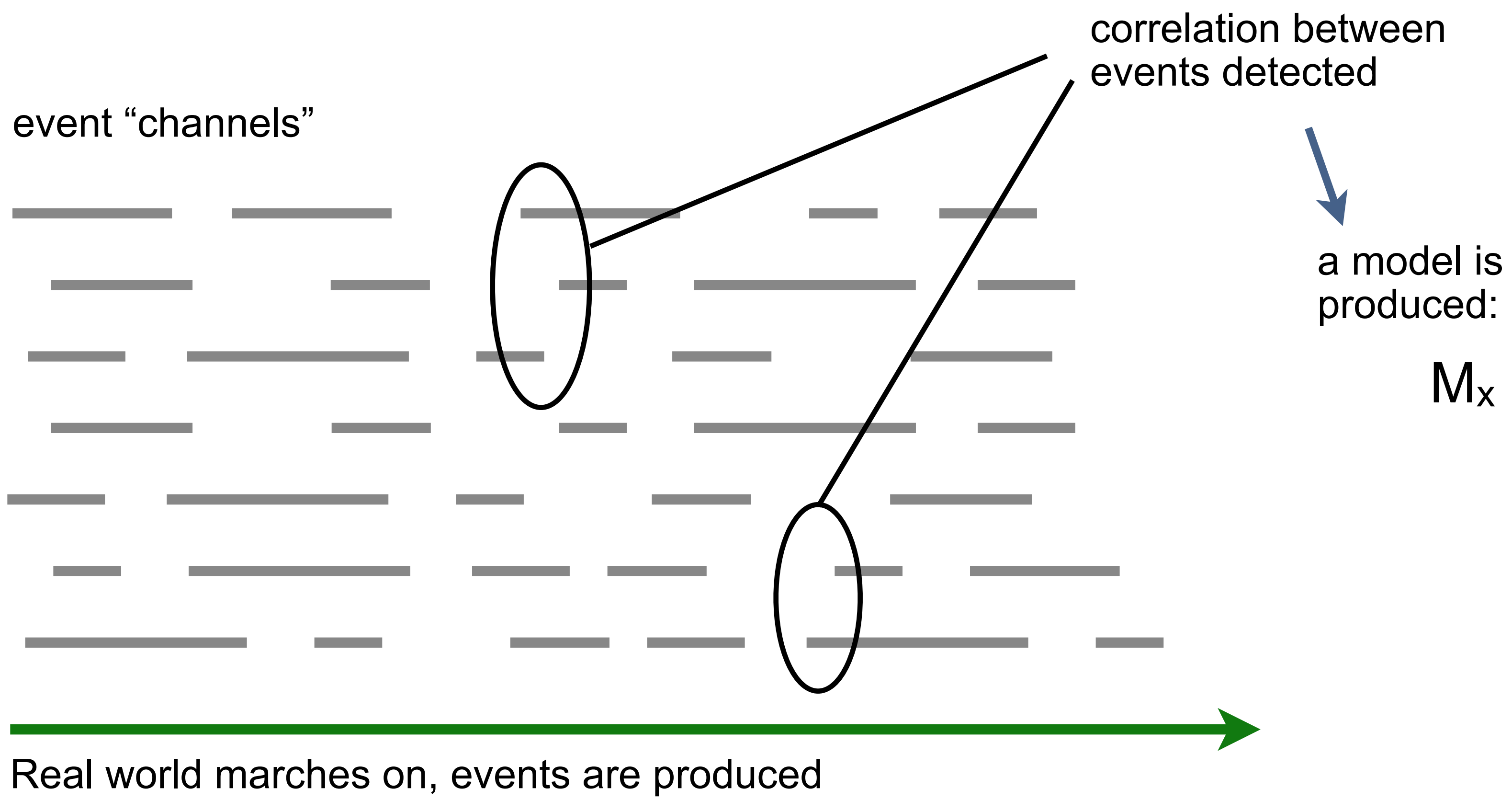


Real world marches on, events are produced

Learning About the World


An AERA agent observes the world...
correlation between two events detected
event "channels"
Real world marches on, events are produced

Learning About the World


correlation between events detected
event "channels"
a model is produced:
M_x
Real world marches on, events are produced

Models

- Models are bi-directional descriptions of the relationship between observed events (inside the system and outside)
  - events are encoded as patterns
  - each model has one pattern on the left-hand side (L), another on the right-hand side (R)
  - patterns represent a kind of bi-directional causation: “if you get an L, M_x predicts you will have an R; if you want an R, M_x predicts it would help to get an L”
- AERA models are designed to be non-axiomatic
  - Their semantics are given by the relationship between the system’s goals, its predicted way of achieving them, and the relation of this to the present state of the world

Models

- Example

Models

- Example

$$M_1 \text{: } [A \rightarrow B]_{[t0\ t1[}$$

Models

- Example

M_1: $[A \to B]$

Models

- Example

$$M_1: [A \rightarrow B]$$

$$\underset{a}{\rightsquigarrow}$$

Models

- Example

M_1: $[A \rightarrow B]$

$$\underset{a}{\rightsquigarrow}\ b \text{ @ } [t_0\ t_1[$$

Models

- Example

M_1: $[A \rightarrow B]$

$\underset{a}{\rightsquigarrow}$ b @ $[t_0\ t_1[$

- A and B are **terms**

Models

- Example

$$M_1: [A \rightarrow B]$$

$$\underset{a}{\rightsquigarrow} \; b \;@\; [t_0\ t_1[$$

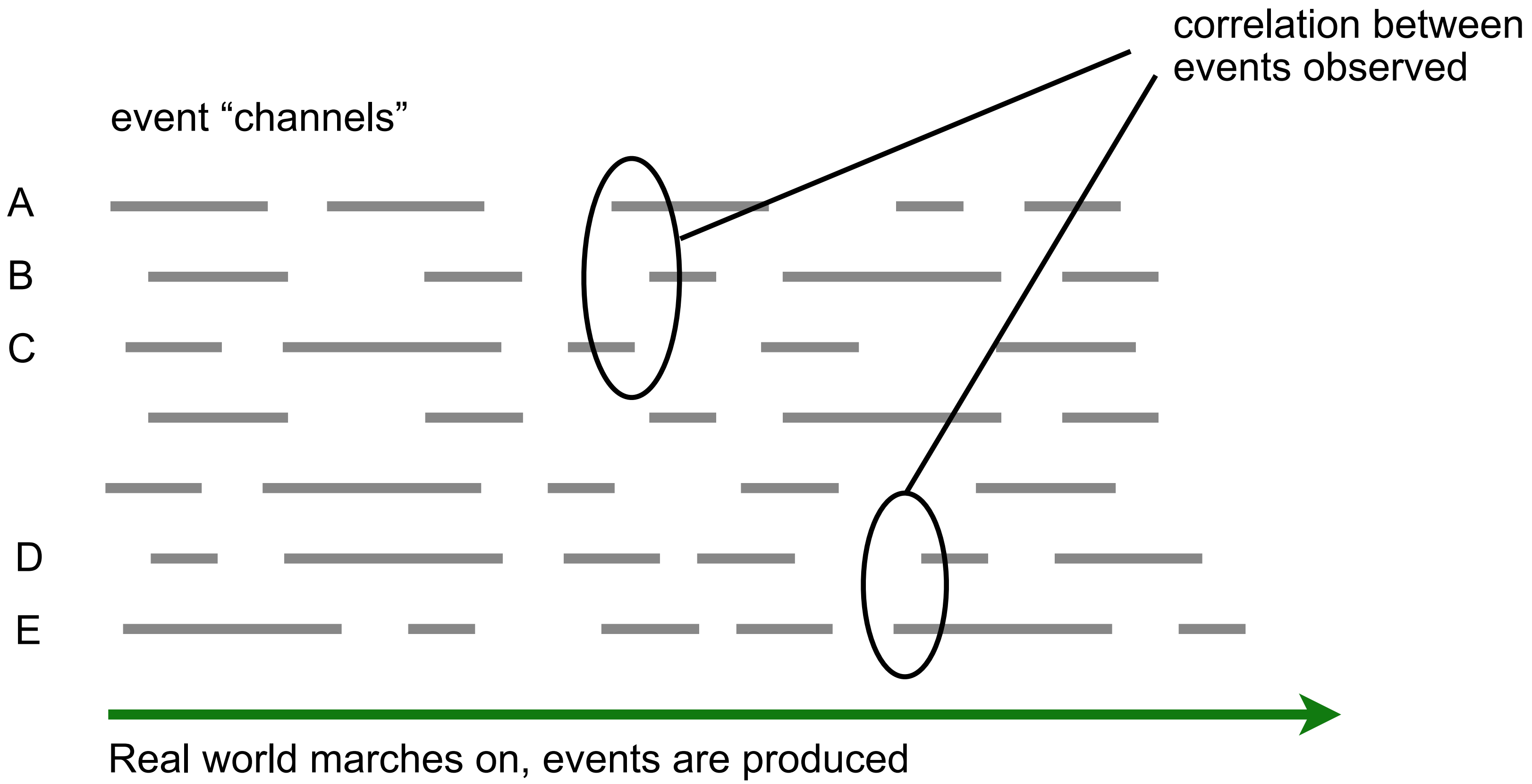
- A and B are terms
  - A term is a **fact** that points to a marker
    - mk.color X red Y

Models

M_1: [A → B]

A: (fact (mk.val e red 12) (conf 0.9) (100 msec 120 msec))

Learning About the World: Composite States


correlation between events observed
event "channels"
A
B
C
D
E
Real world marches on, events are produced

Learning About the World: Composite States

S_1: [A B C D E] $_{[t0\ t1[}$

Example of Model Use

- Let's assume the repository contains these (innate) models:

  M1: pick-up (obj) → hand-attached-to (obj)

  M2: hand-free → M1

  M3: move-hand (dx) → hand-at-position (dx)

Example of Model Use

- Let's assume the repository contains these (innate) models:

  M1: pick-up (obj) → hand-attached-to (obj)

  M2: hand-free → M1

  M3: move-hand (dx) → hand-at-position (dx)

where *move-hand* is an external command linked to the robot's arm, *dx* is amount of displacement (as a vector in 3D), and *hand-at-position* is an observable state of the world

Goal1: hand-attached-to (obj)

Goal2: hand-at-position (dx)

Example of Model Use

Analyzing the activity of the system's manipulator in achieving these goals:

- the system has acquired the following simple model:

M4: move-hand (dx) → object-at-position (dx)

This model has the following precondition extracted from data:

M5: hand-attached-to (obj) → M4

Example of Model Use

- Given:

Goal: Displace obj-1 by dx

Using abduction (backward chaining) the system can find that to satisfy this goal it must activate the model M4.

However, having this model is a precondition expressed in the model M5, which is not satisfied (the hand is not attached to the object), so it must first

satisfy the goal hand-attached-to

System commits to this new subgoal, which requires the activation of model M1. This time, the prerequisite of the model M1 is satisfied (indeed, the hand is free – see model M2) which implies the actual activation of the model M1 and the execution of command pick-up.

The environment will output the fact hand-attached-to which satisfies both the current subgoal and the prerequisite of the model M4. The model is activated which implies the execution of the command move-hand and, as the consequence, the actual displacement of the object at the desired position.

Example of Model Use

- Let's assume the repository contains these (innate) models:

  M1: pick-up (obj) → hand-attached-to (obj)

  M2: hand-free → M1

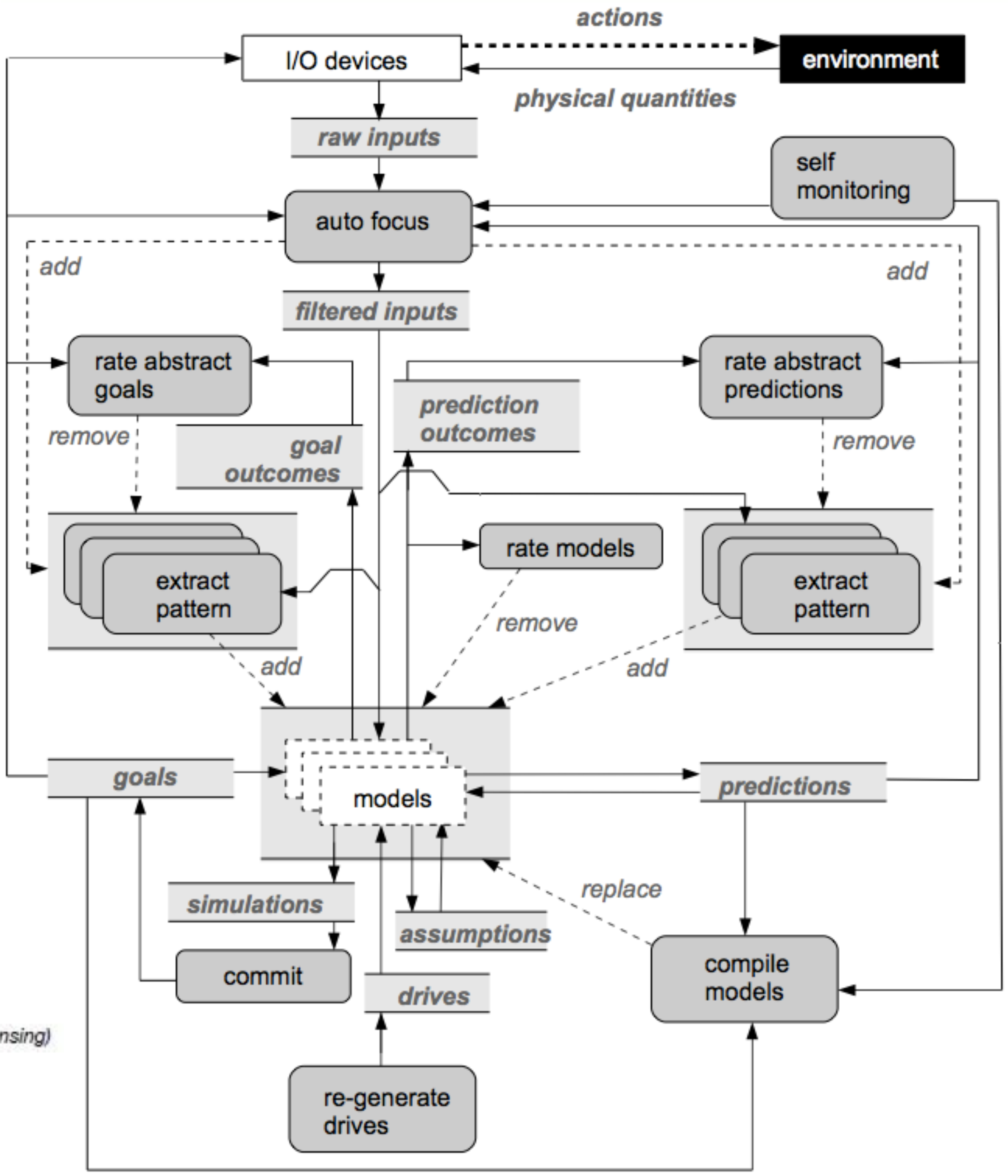
  M3: move-hand (dx) → hand-at-position (dx)

  M4: move-hand (dx) → object-at-position (dx)

  M5: hand-attached-to (obj) → M4

Goal3: Displace obj-1 by dx

AERA



Key:

▭	*process*
▨	*code/data storage*
→	*code/data flow*
⇢	*operation on storage*
▪▪▶	*operation in the domain (this includes sensing)*

Autonomy dimensions

- Learning
  - Domain
    - Enables system to handle novel situations and task varations
  - Meta-
    - System improves own operation, increasing its capacity to solve complex tasks
- Realtime
  - Failure to keep up with the environment reduces autonomy and overall operation
    - May introduce artificial pauses etc.
- Resource management
  - Autonomous operation involving multiple simultaneous cognitive processes, complex environments, limited resources and time constraints requires sophisticated management of resources

Replicode

- The programming language of AERA, **Replicode**, was created to meet several shortcomings plaguing ***all prior programming languages:***
  - Syntax and semantics meant for humans
  - Inefficiencies in execution
  - Lack of support for induction and abduction as first-class logical operations
  - Lack of ability to model own behavior, at a high level of detail
  - Lack of ways to handle passage and representation of (external and internal) time
  - Lack of support for self-generated code

Replicode

- Parallel programming language consisting of and supporting:
  - a new programming paradigm
    - reflectivity
  - unified ampliative reasoning
    - abduction, induction, deduction
  - unified knowledge representation
    - goals, sequentiality, association
  - low granularity supporting temporal grounding
  - model-based self-organization

Replicode: Semantically Closed

- A language is semantically closed iff it is able to talk about its own semantics
  - The meanings of the terms of the language can be given within the language

[Manuel Bremer]

- In our work: A **system** is **semantically closed** iff it can make sense of its own meaning (read: mission) and steer its evolution accordingly

[see e.g. Rocha 2000]

Replicode Features

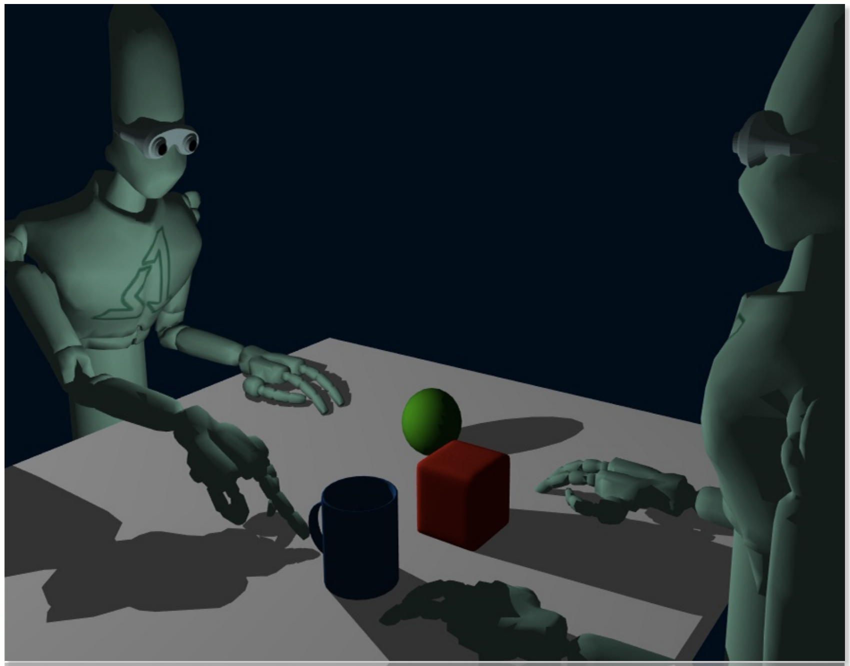
- Machine-readable operational semantics
- Simple syntax
- Simple semantics: e.g. no `if-then` constructs, no `loop` constructs
- Methods for distributed partial consolidation and coordination of knowledge (distributed asynchronous knowledge updating management)
- Direct support for unified logical operations: induction, deduction, abduction
- Extremely fast logical and distributed operations execution

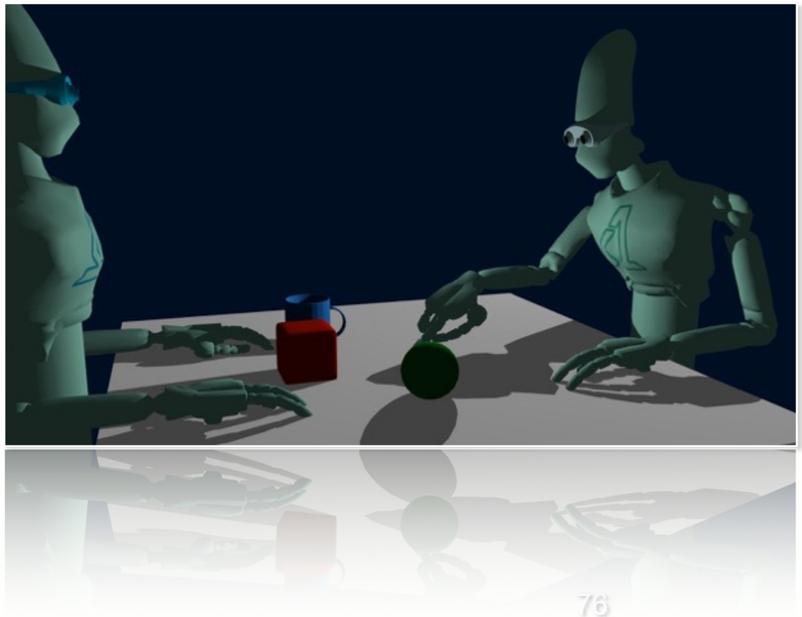
AERA / S1

in action

Evaluation

- Result to be evaluated in a complex domain
  - Human-human face-to-face communication, more specifically, a simulated TV interview
- Interview conduced in cyberspace





The (short-term) Goal

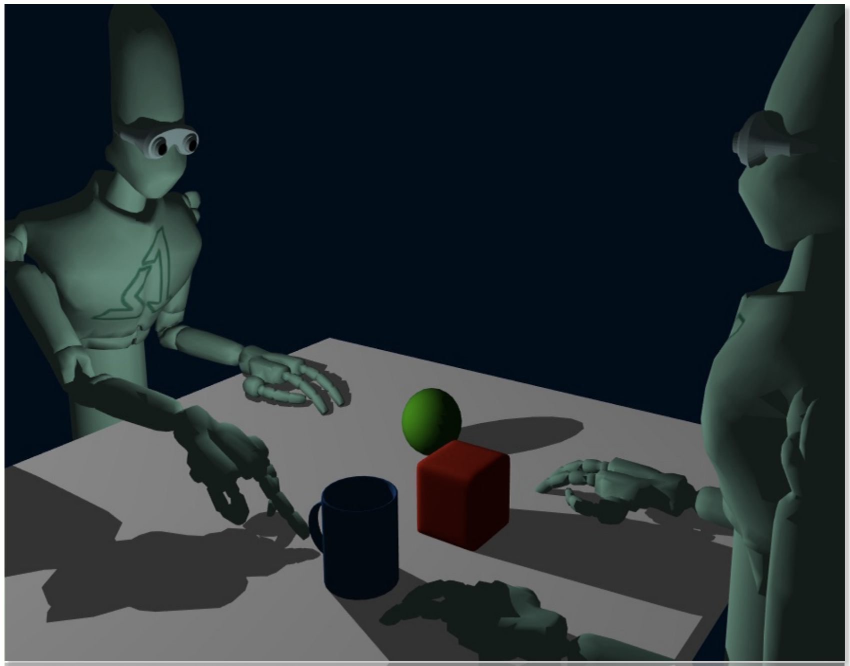
To develop a system that can

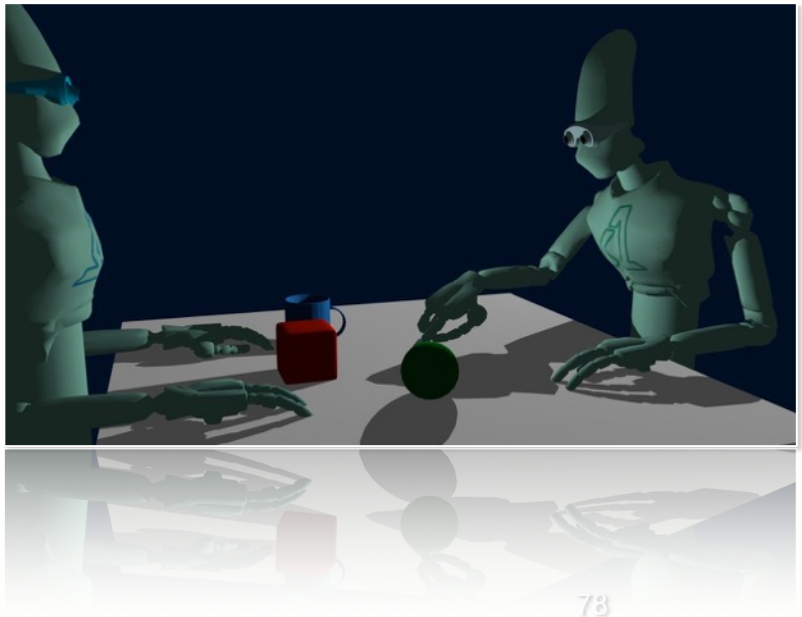
learn a highly complex task

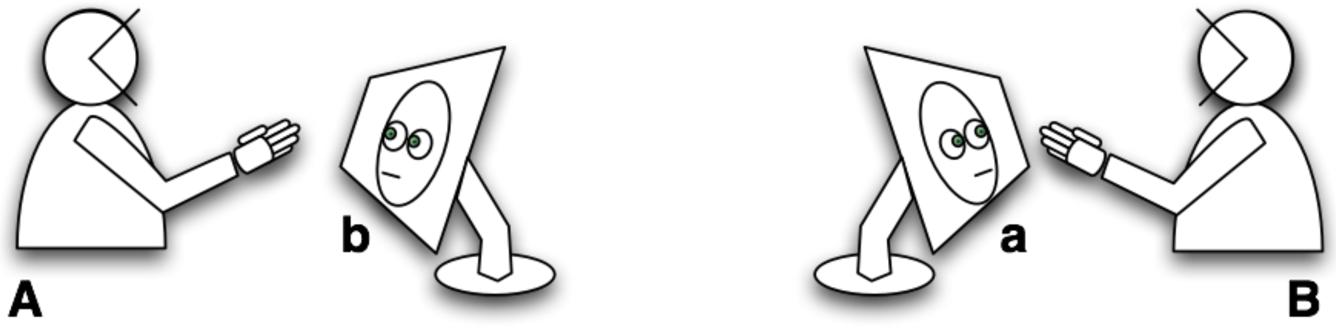
from observation and goal imitation

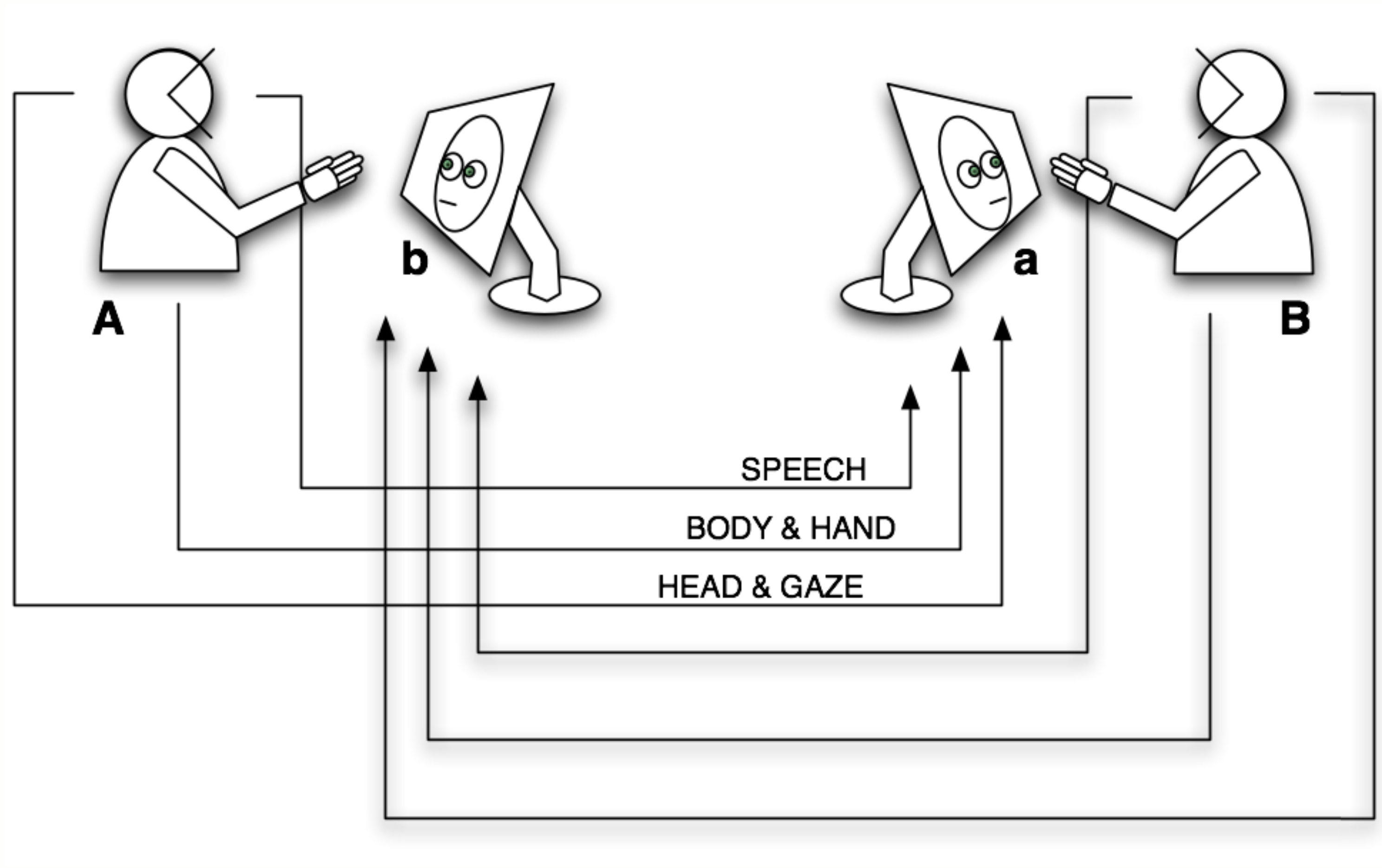
AERA Evaluation

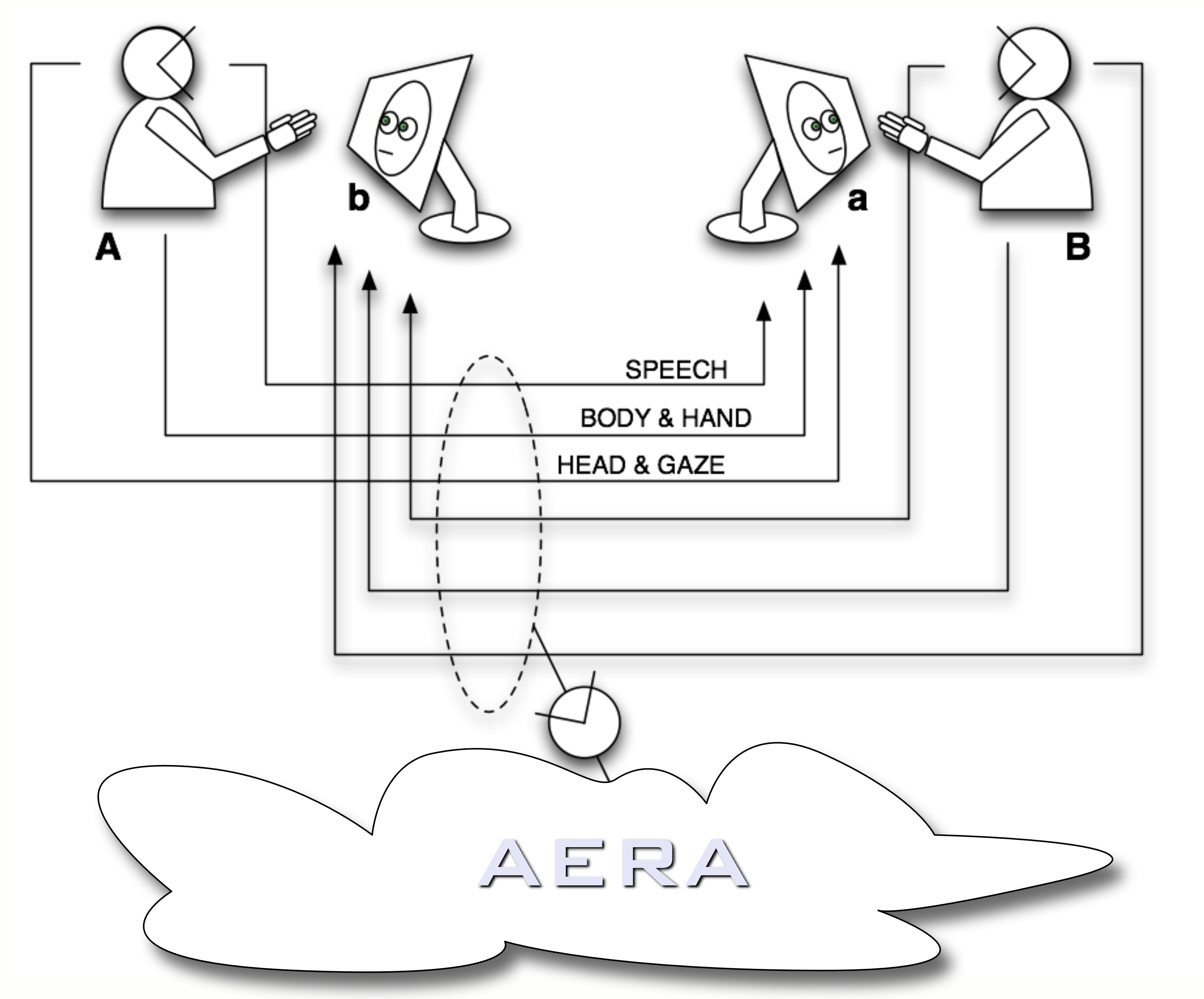
- AERA agent S1:
  - Observes two humans in an interview
  - Learns how to conduct an interview
    - Taking either role: interviewer and interviewee
  - Demonstrates skills in an actual interview with a human






A
b
a
B


b
a
A
B
SPEECH
BODY & HAND
HEAD & GAZE


b
a
A
B
SPEECH
BODY & HAND
HEAD & GAZE
AERA

What is Given

- words (but no grammar)
- actions: grab, release, point-at, look-at
- stopping the interview clock ends the session
- interviewee-role
- interviewer-role
- objects: glass-bottle, plastic-bottle, cardboard-box, wodden-cube, newspaper, wooden-cube
- objects have properties (e.g. made-of)
- in interruption case: a time limit is imposed

What is Given

Top-level model of interviewer:

- interviewer wants to prompt interviewee to talk (interviewer's role is pressure interviewee to speak)

Top-level models of interviewee:

- interviewee wants to talk
- never talk unless prompted
- tell about properties of objects being asked about, for as long as there still are properties available
- never talk about properties that have already been mentinoned

What is Given

1. words (but no grammar)
2. actions: grab, release, point-at, look-at (defined as event types constrained by geometric relationships)
3. stopping the interview clock ends the session
4. objects: glass-bottle, plastic-bottle, cardboard-box, wodden-cube, newspaper, wooden-cube
5. objects have properties (e.g. made-of)
6. interviewee-role
7. interviewer-role
8. Model for interviewer
   I. top-level goal of interviewer: prompt interviewee to speak
   II. in interruption case: an imposed interview duration time limit
9. Models for interviewee
   I. top-level goal of interviewee: to talk
   II. never talk unless prompted
   III. tell about properties of objects being asked about, for as long as there still are properties available
   IV. don't talk about properties that have already been mentioned

What is Given

+ Human interaction sessions examples
    - about 20 hours

What is Learned

MULTIMODAL COORDINATION & JOINT ACTION

- take turns speaking
- a silence from the interviewer means "go on"
- a nod from the interviewer means "go on"
- co-verbal deictic reference
  - manipulation as deictic reference
  - looking as deictic reference
  - pointing as decitic reference

What is Learned

INTERVIEW

- an interview involves a series of Qs and As
- role of interviewer and interviewee
  - interviewer: to ask a series of questions
  - interviewee: a property is not spoken of if it is not asked for
- interruption condition: using "hold on, let's go to the next question" as a way to keep interview within time limits
- "thank you" stops the interview clock

What is Learned

LANGUAGE

- interviewee: **what/how to answer** based on **what is asked**
- word order (“grammar”)

What is Learned

After 20 hours of watching two humans in a simulated TV interview like the one above, S1 has learned the following via **goal-level imitation**:

- GENERAL INTERVIEW PRINCIPLES
  1. word order in sentences (with no a-priori grammar)
  2. disambiguation via co-verbal deictic references
  3. role of interviewer and interviewee
  4. interview involves a series of Qs and As

- MULTIMODAL COORDINATION & JOINT ACTION
  5. take turns speaking
  6. co-verbal deictic reference
     I. manipulation as deictic reference
     II. looking as deictic reference
     III. pointing as decitic reference

- INTERVIEWER
  7. to ask a series of questions
  8. “thank you” stops the interview clock
  9. interruption condition: using “hold on, let's go to the next question” can be used to keep interview within time limits

- INTERVIEWEE
  10. what to answer based on what is asked
  11. an object property is not spoken of if it is not asked for
  12. a silence from the interviewer means “go on”
  13. a nod from the interviewer means “go on”