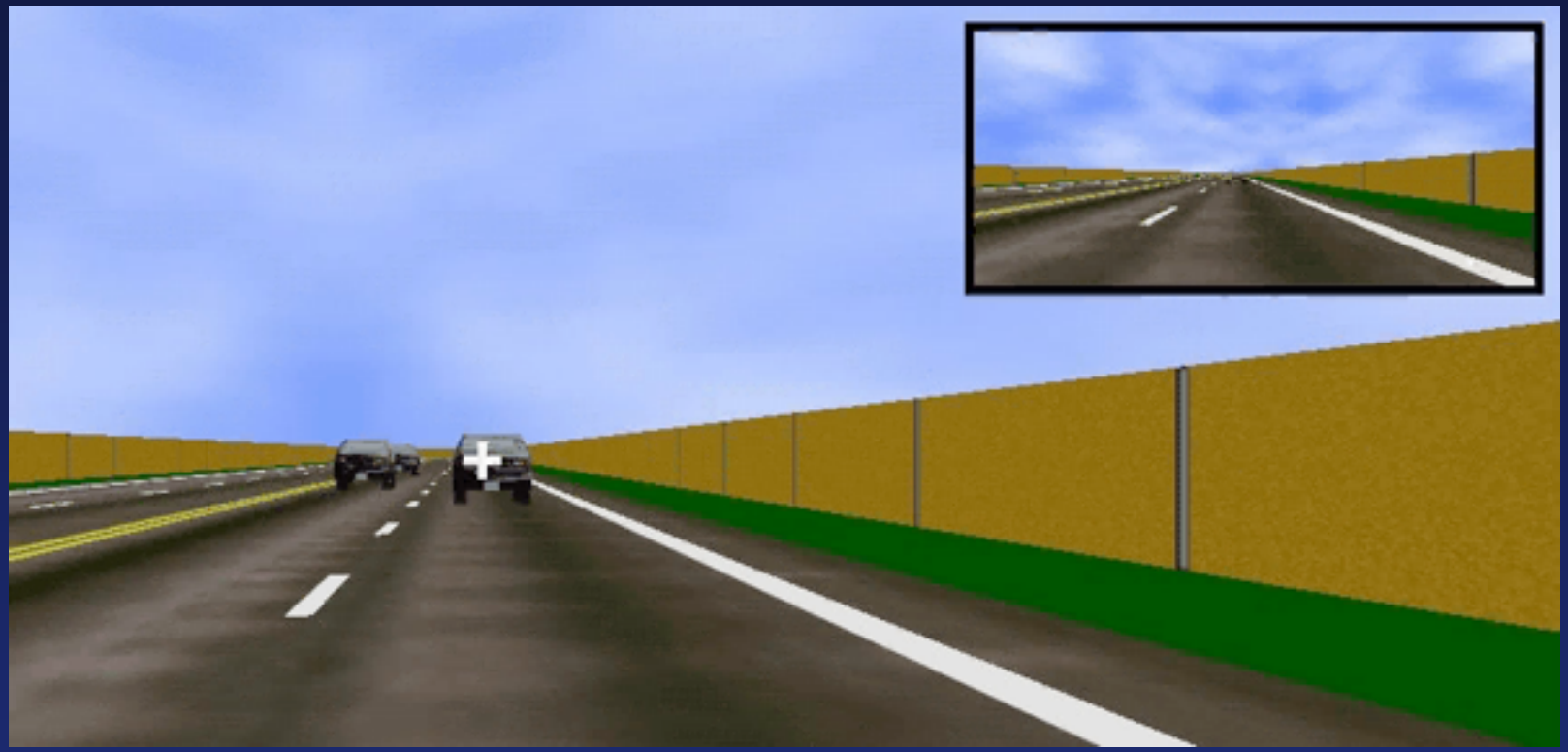
Eye Movements & Cognitive Models

Dario Salvucci

Drexel University

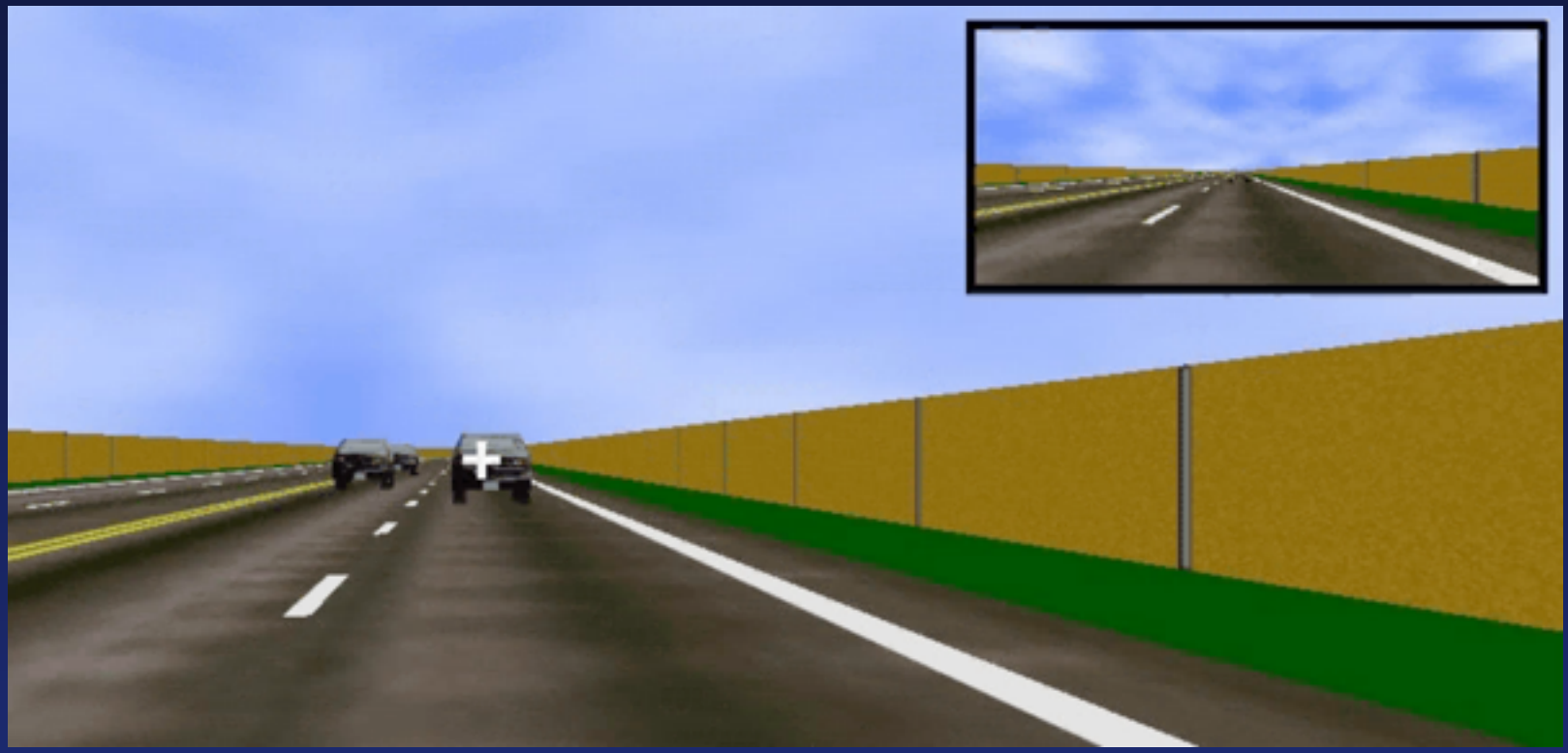
Eye Movements & Cognition

- Eye movements help us understand cognition
  - What is this driver thinking?
  - Could you guess the driver's intentions without them?



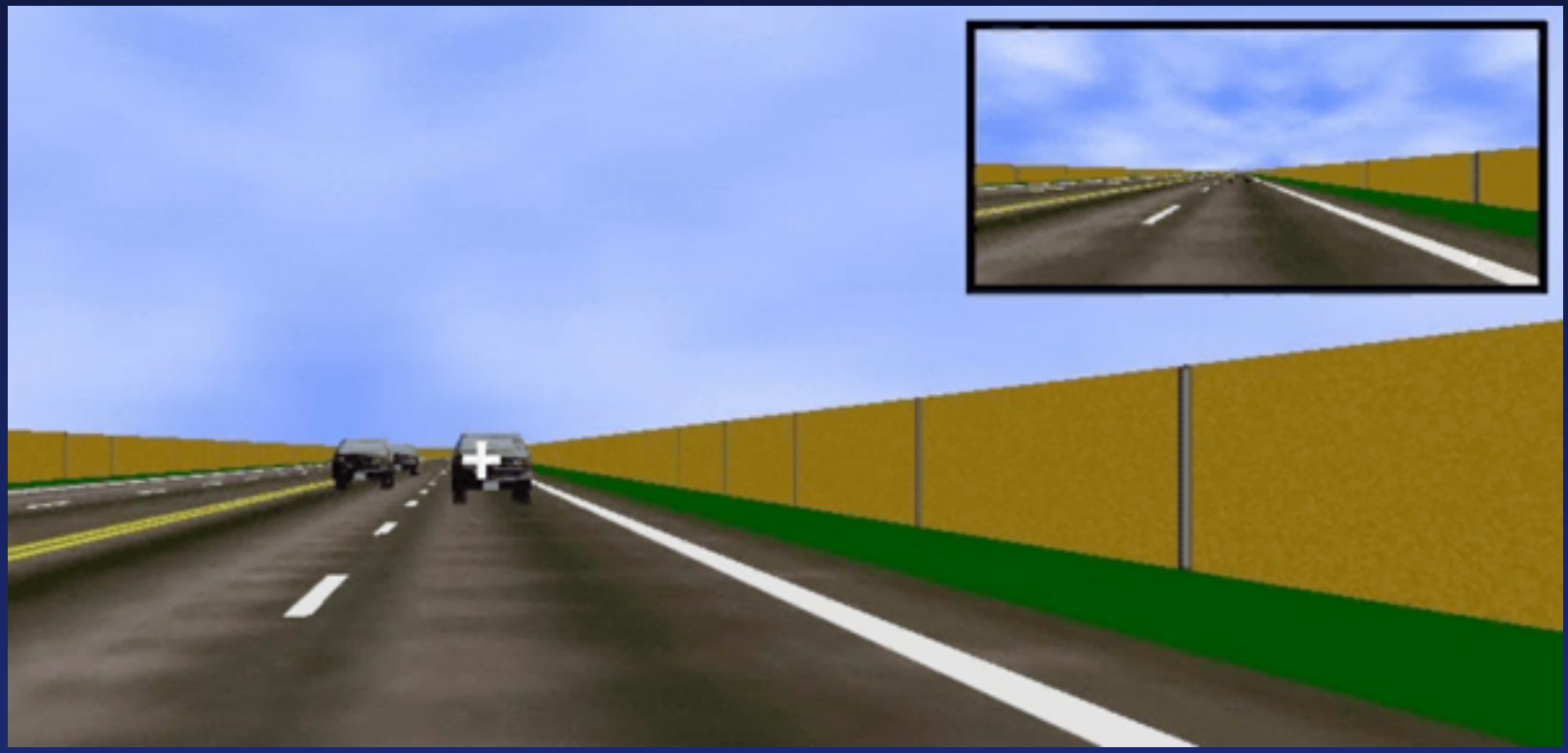
Eye Movements & Cognition

- Cognition helps us understand eye movements
  - Why does the driver look at the lead car?
  - Why doesn't the driver look at oncoming traffic?



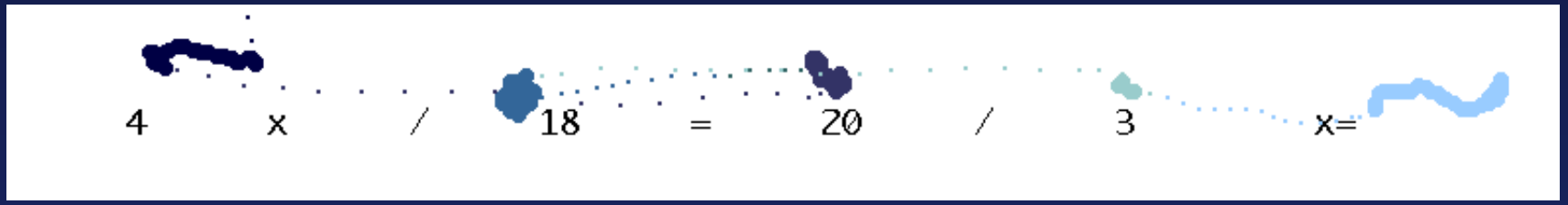
Eye Movements & Cognition

- Models help us understand both
  - How does is gaze information being acquired and used?
  - What information is being captured without gaze?



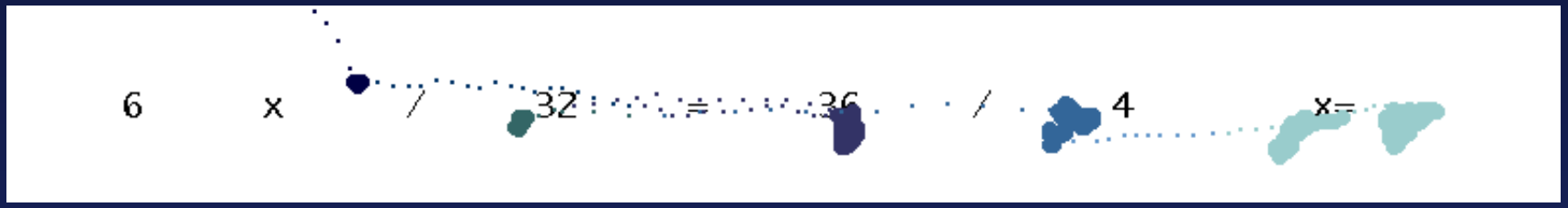
Eye Movements & Cognition

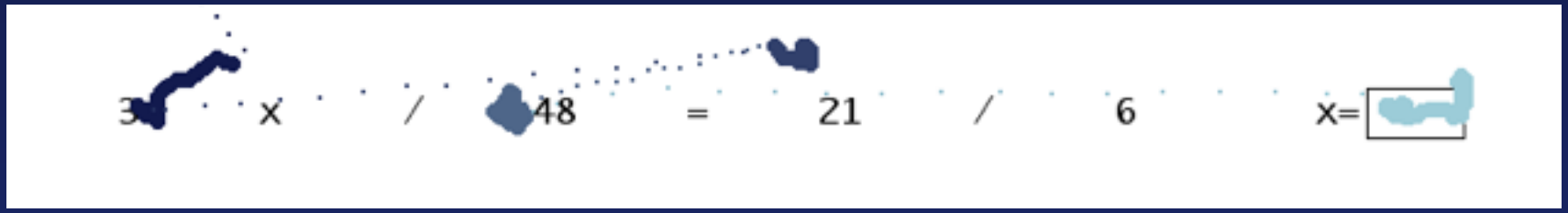
- Eye movements help us understand cognition
  - Without eye movements, we would see…
    - a long delay
    - a keyed response
  - What strategy is being used to solve the equation?



Eye Movements & Cognition

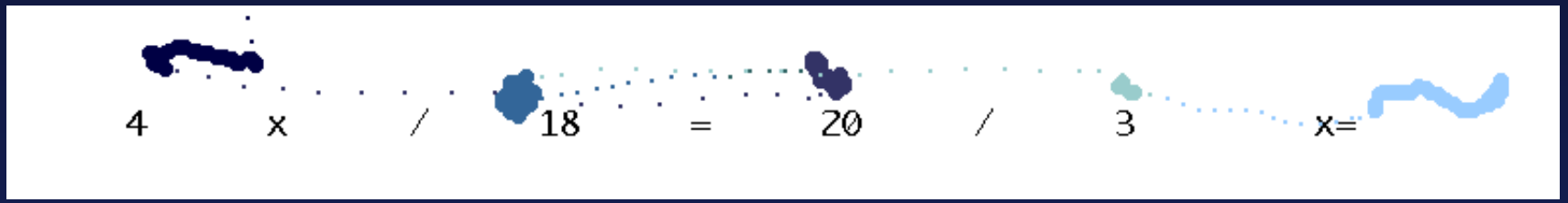
- Cognition helps us understand eye movements
  - Sometimes, eye movements don't behave as expected
  - How do we explain "misplaced" or missing fixations?

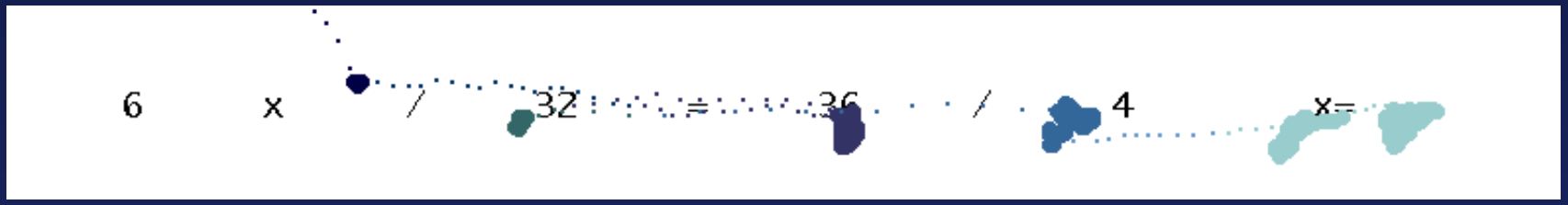


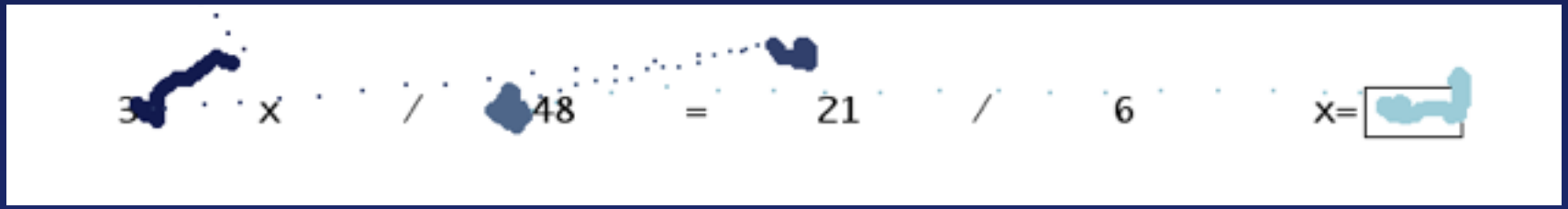


Eye Movements & Cognition

- Models help us understand both
  - They give us a way to tie the eyes and brain together

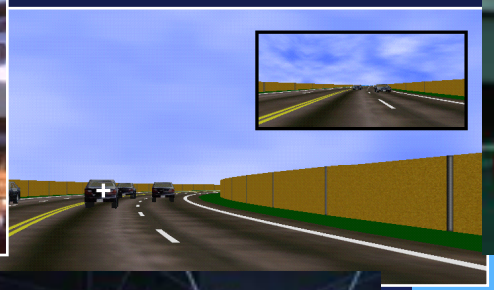


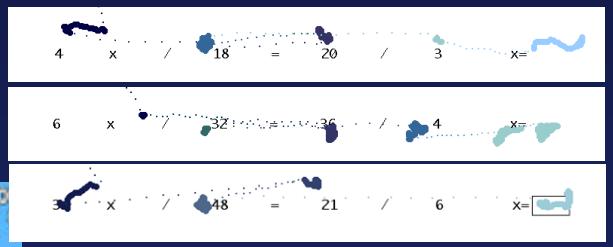


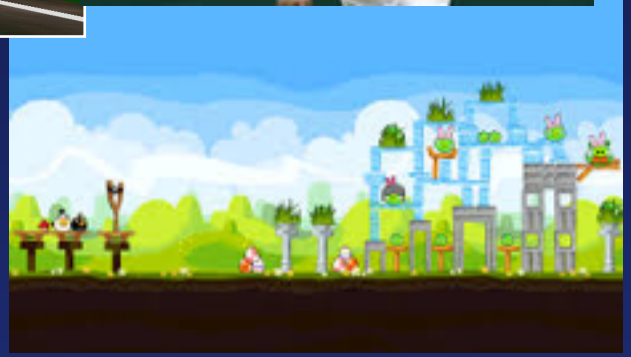


Eye Movements & Cognition

- Cognition, eye movements use the same processes and mechanisms in all these domains
- So we should explain them with a unified set of processes and mechanisms






Cognitive Architectures

- Cognitive architecture = psychological theory + computational framework
  - like a human-modeling programming language
  - built-in "functions"
    - e.g., memory store and recall, goal and subgoal setting, perceptual-motor behavior
  - built-in limitations
    - e.g., forgetting, errors, perceptual-motor parameters

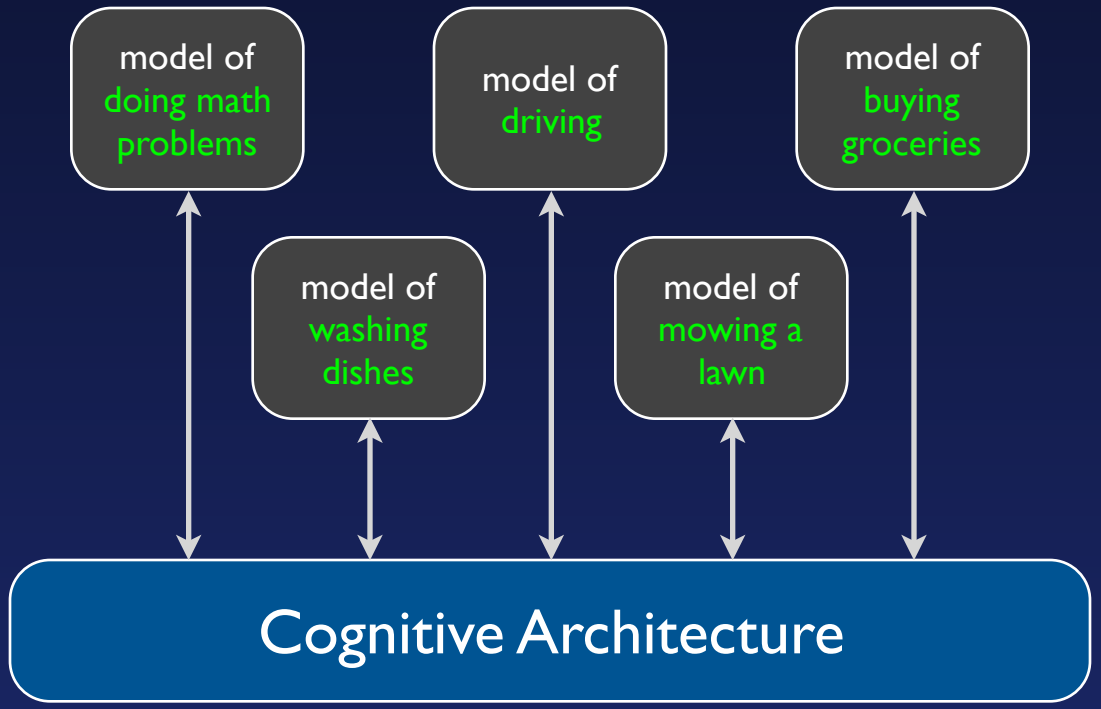
→ Models are constrained within larger theory

→ Models are psychologically plausible

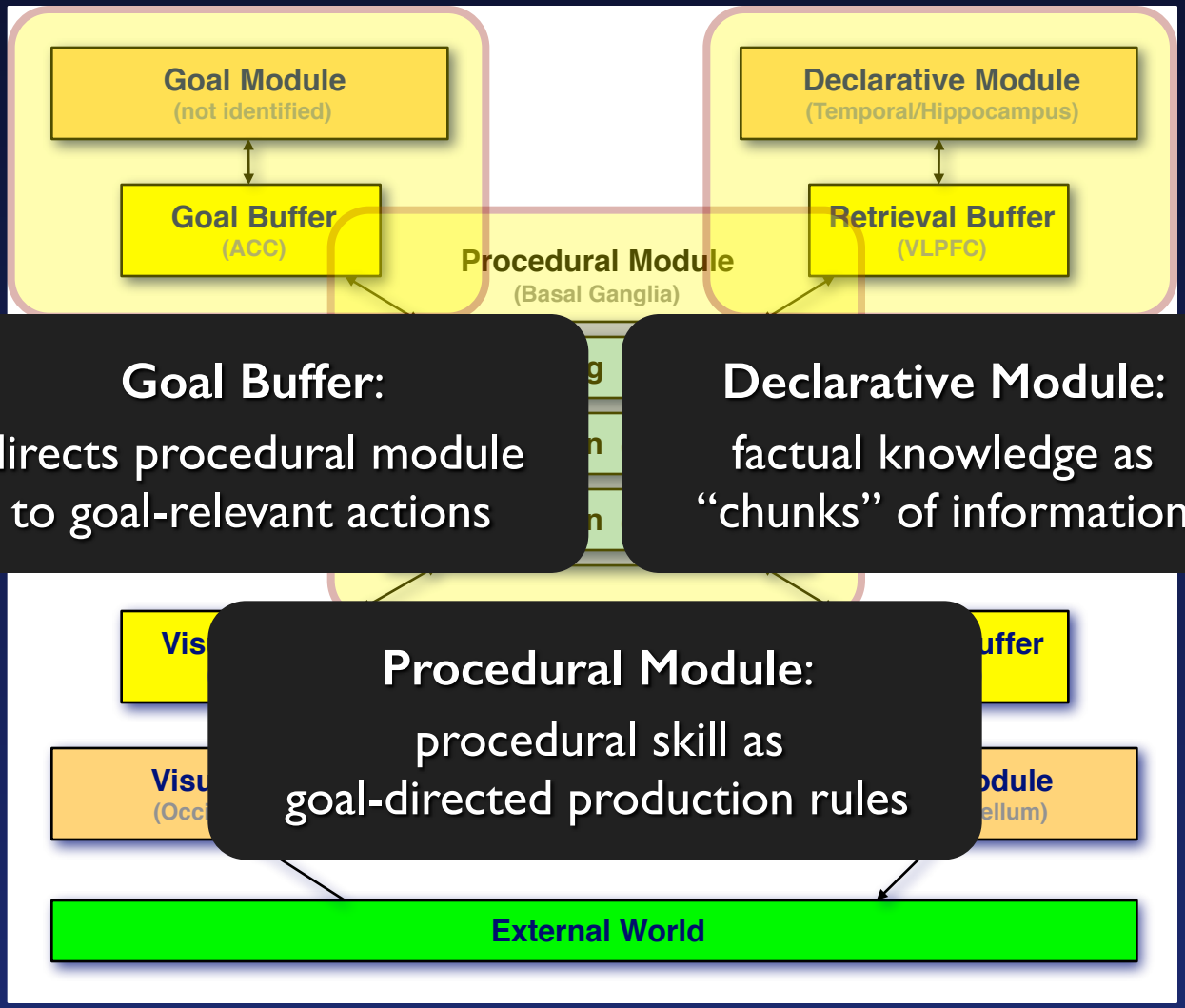
Cognitive Architectures

Domain "Software"
knowledge for
individual domains

Brain "Hardware"
core mechanisms
that all people have



ACT-R



Goal Buffer:
directs procedural module to goal-relevant actions

Declarative Module:
factual knowledge as "chunks" of information

Procedural Module:
procedural skill as goal-directed production rules

ACT-R Vision

- Spotlight theory of attention

what the world looks like

how we see the world

- limited fovea of high resolution
- large periphery of degraded resolution

ACT-R Vision

+visual-location>	Find a location in my visual field that satisfies a set of constraints
=visual-location> +visual>	Given that I have the location, move my attention to that location
=visual>	Use the encoded object

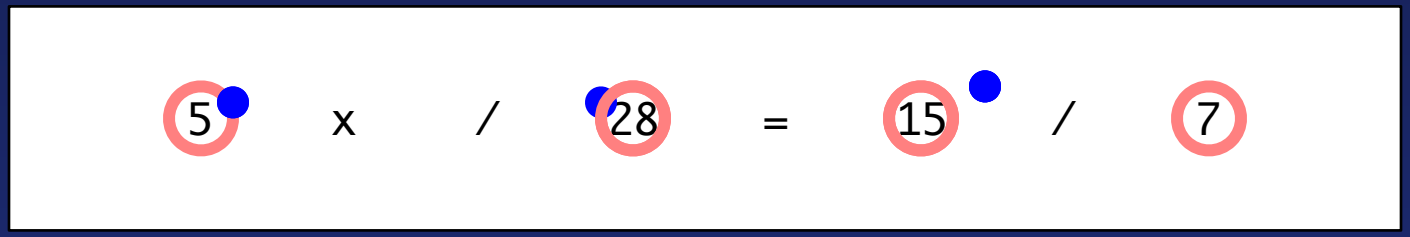
"Where"

"What"

- "Where" process is basically constant time
- "What" process depends on what's being encoded
  - e.g., *the* vs. *antidisestablishmentarianism*
  - e.g., *hän* vs. *peruspalveluliikelaitoskuntayhtymä*
  - (but general visual objects are a challenge: △? ⇨?)

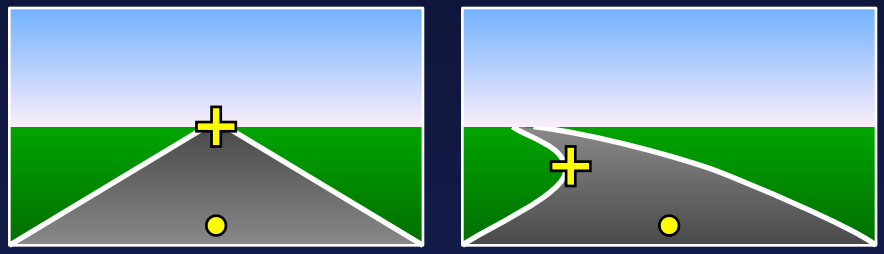
ACT-R Vision + EMMA

- EMMA dictates how visual attention ≠ fixation
  - ACT-R model generates a shift of attention
  - soon after (~200 ms) — maybe — the eyes move to the target of attention [based on E-Z Reader]
    - "labile" stage that can be cancelled if a new shift occurs
    - "non-labile" stage that cannot be cancelled
  - but…
    - movement can miss target, requiring a re-fixation
    - attention can shift quickly again, skipping the first target

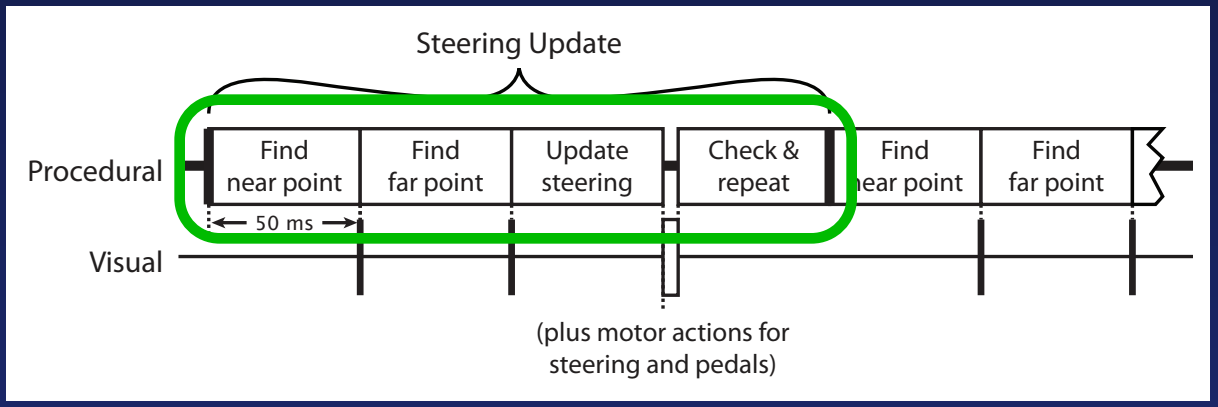


Driving as a Single Task

- Two-level steering with near & far points



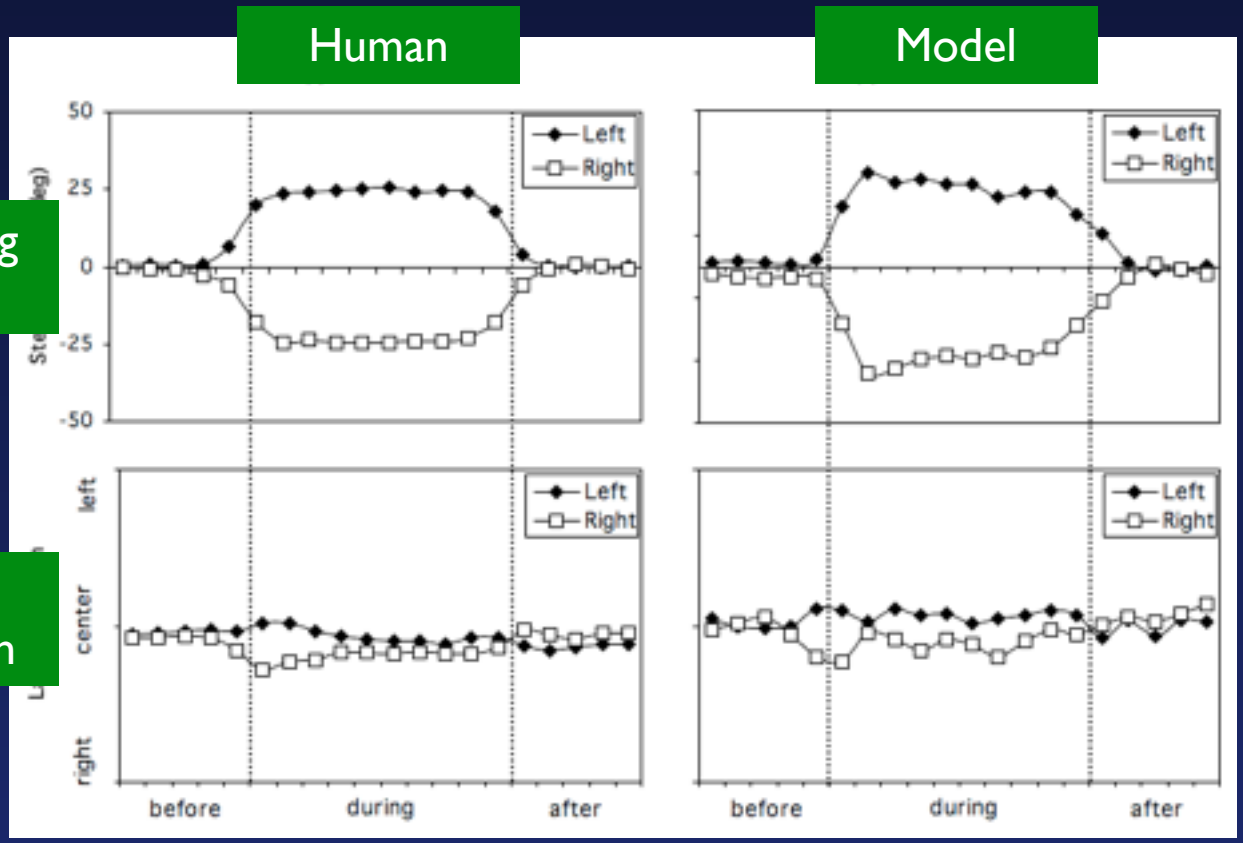
- ACT-R model procedural steps



Driving as a Single Task

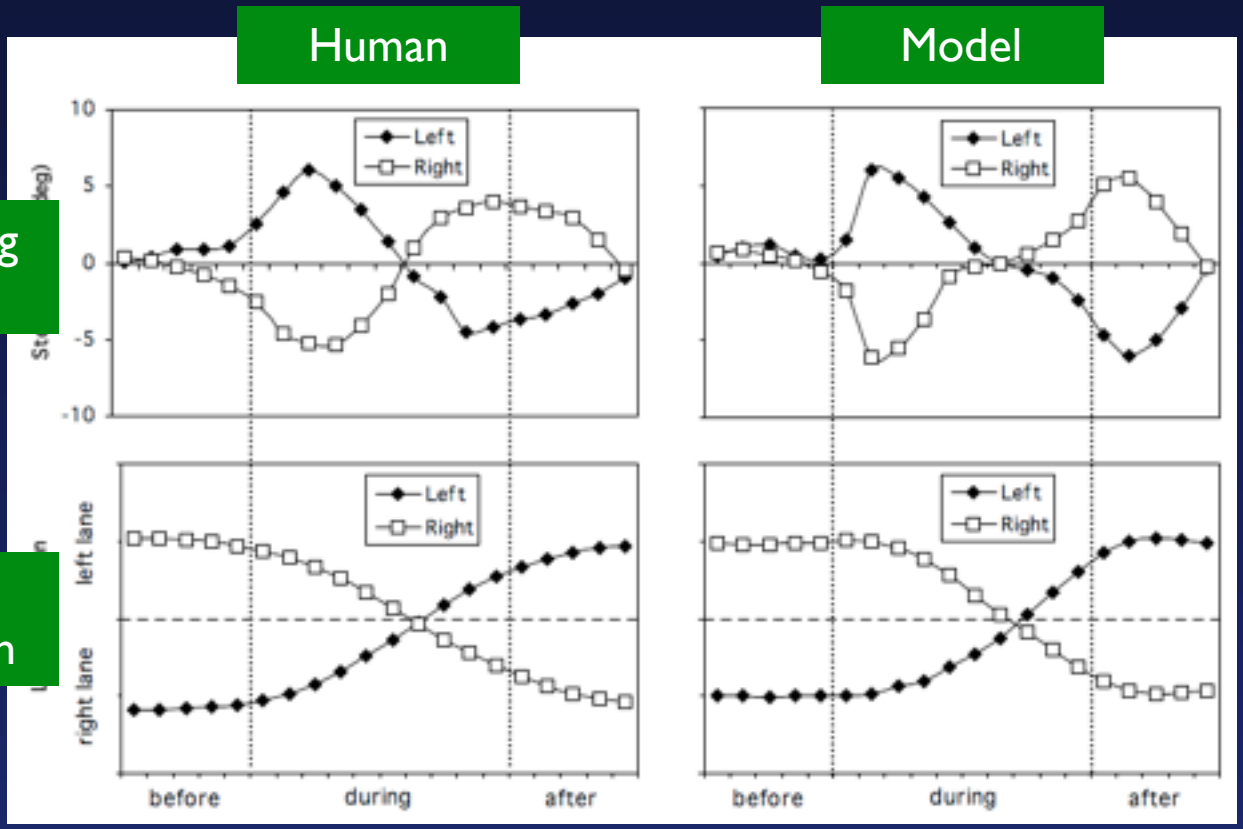
Driving as a Single Task

- Curve negotiation



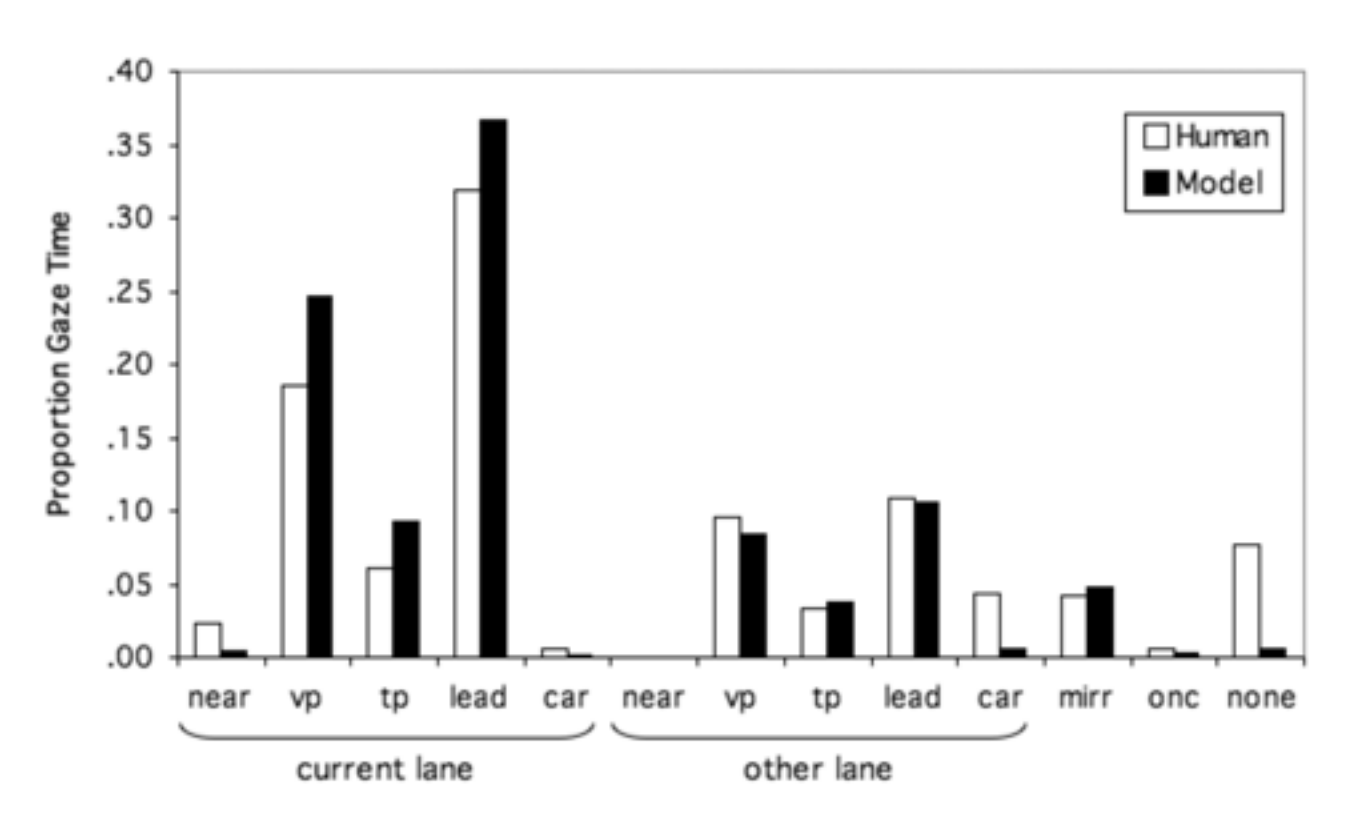
Driving as a Single Task

- Lane changing



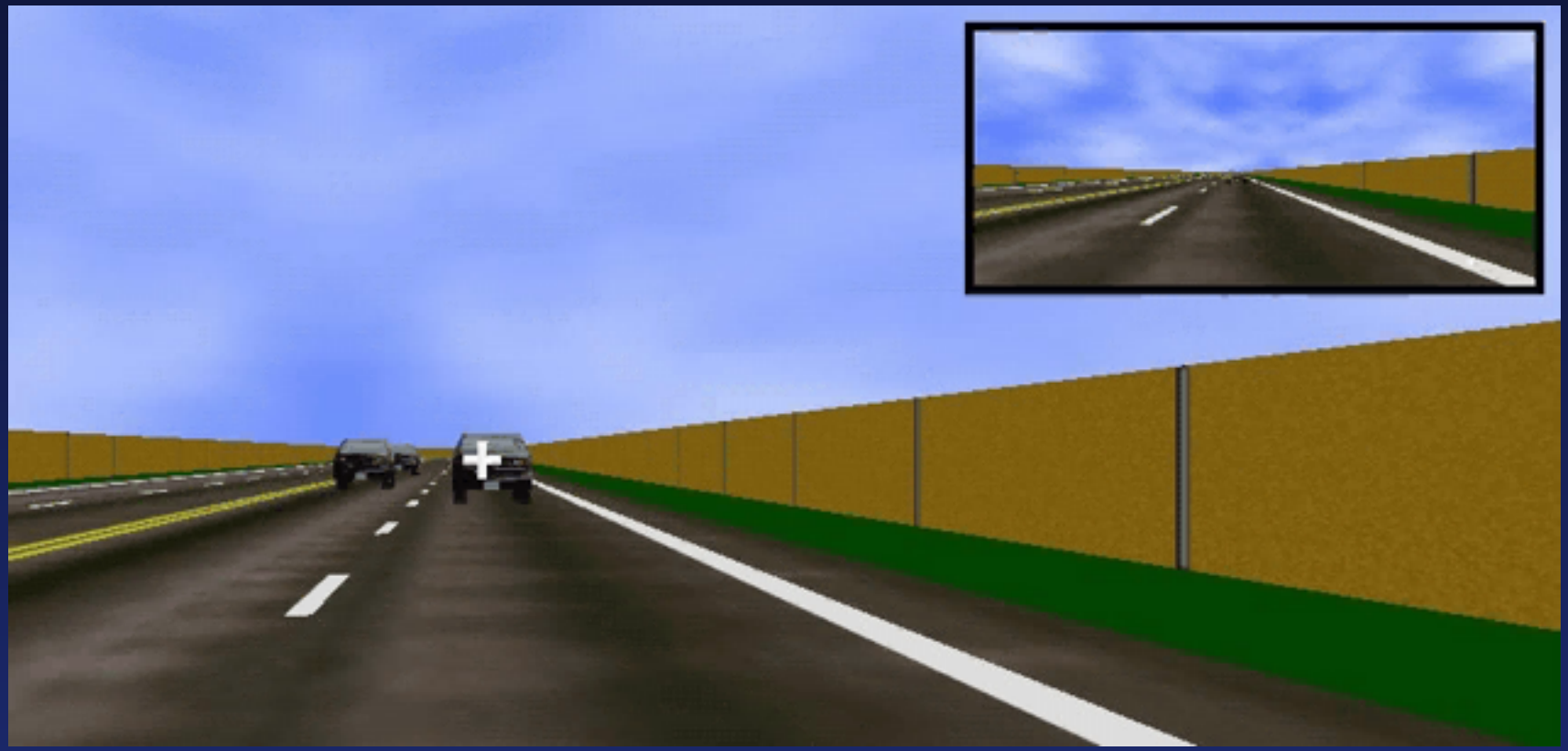
Driving as a Single Task

- Gaze time on regions of interest



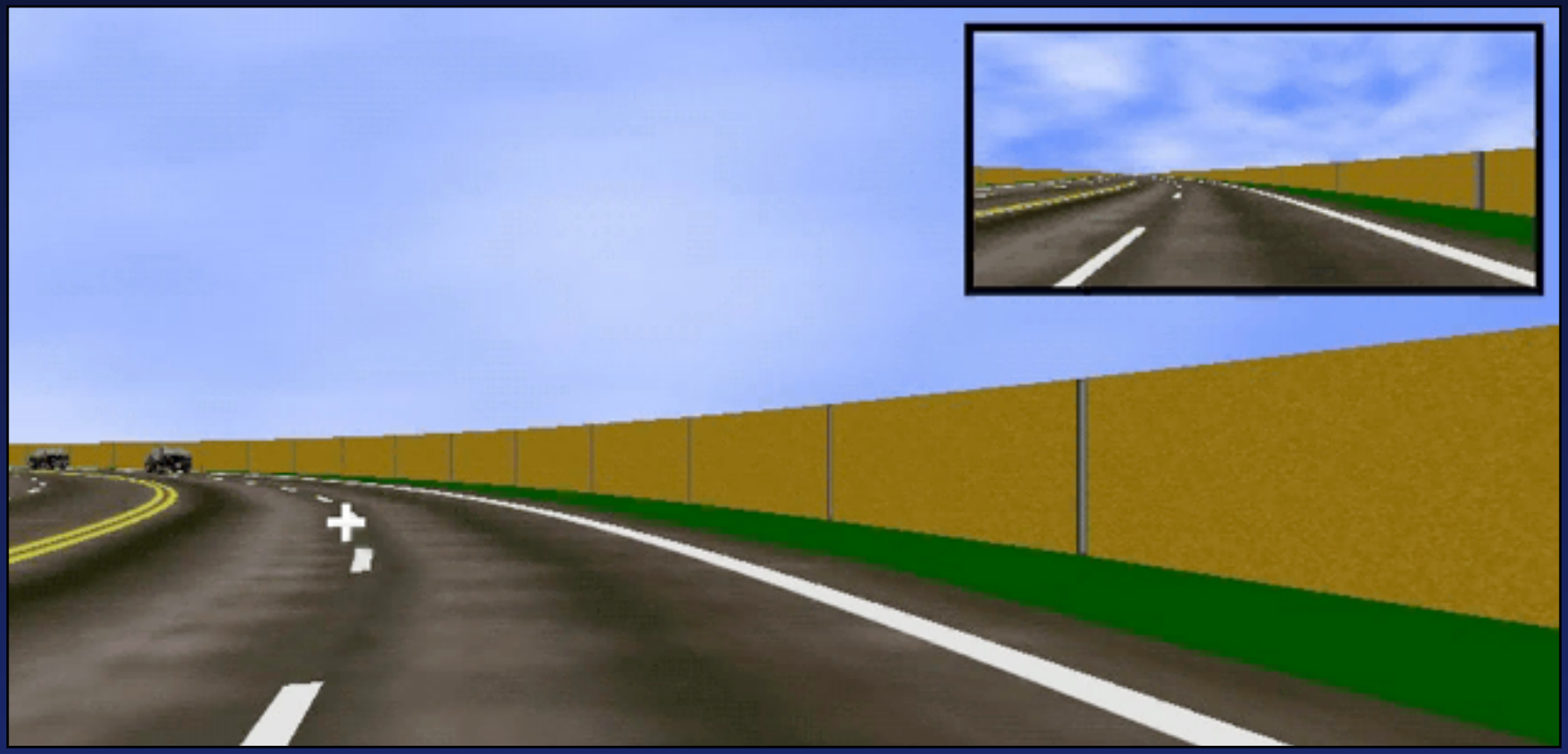
Driving as a Single Task

- Human driver...



Driving as a Single Task

- Model driver...



A Word about Abstraction

- At what "level of abstraction" should we model?
  - individual fixations and saccades
  - individual gazes
  - aggregate gaze time on regions
  - or higher levels? or lower levels?
- Many different levels may be valid, and useful

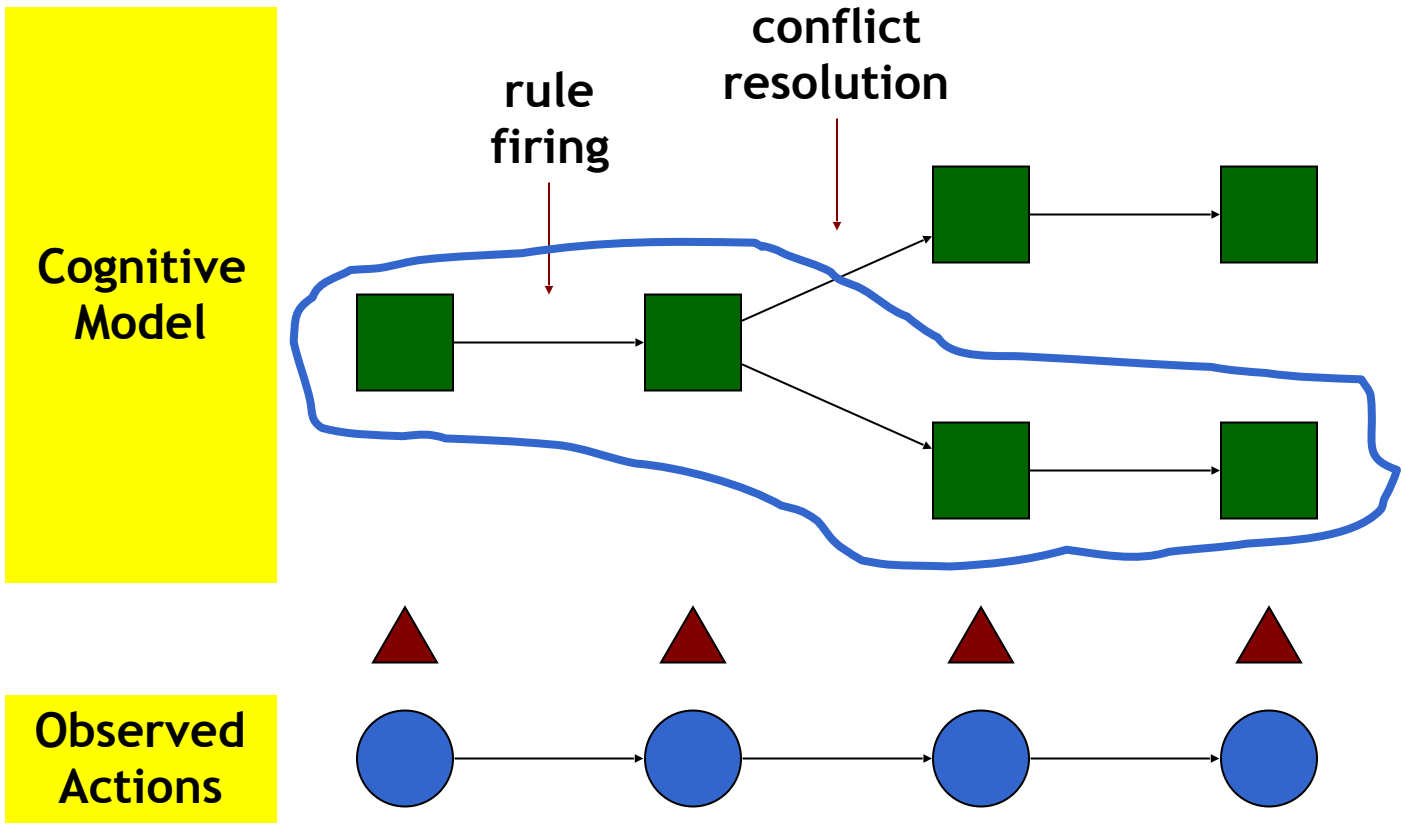
Modeling & Programming

- Lots of work in the 1980s-90s on modeling & programming within the ACT-R architecture
- Practically all of it was done in the context of Intelligent Tutoring Systems for programming
- Some of this work includes…
  - Model + tutor for programming recursion (Pirolli et al.)
    - "students can learn very different rules for recursive programming from the same example programs"
  - ACT Programming Tutor (Corbett et al.)
    - knowledge tracing: keeping a "skill-o-meter"
  - [Also: Eye movements in (math) tutoring systems]

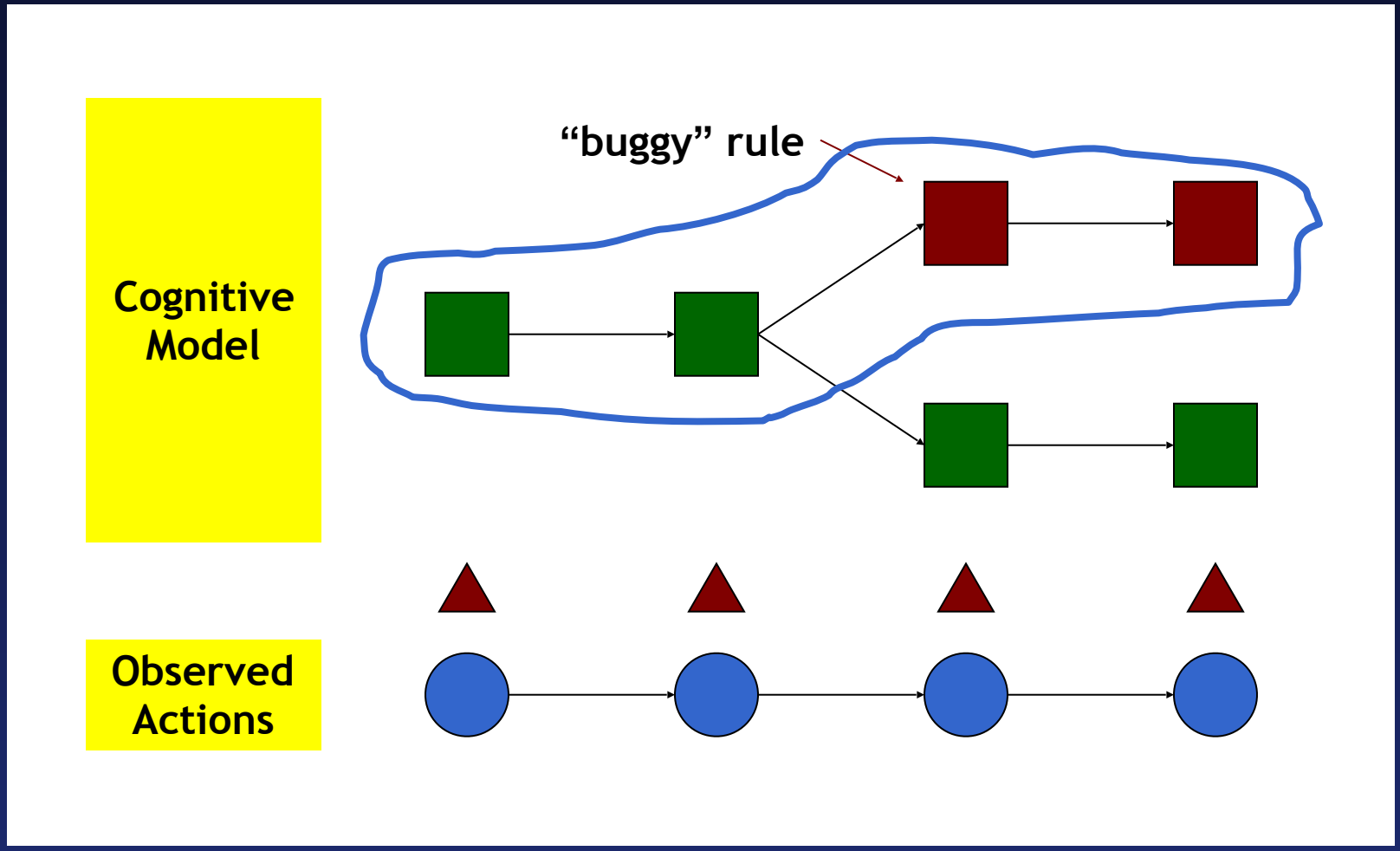
Intelligent Tutoring Systems

- ITS = system that tracks student cognition and teaches accordingly
- Central idea: Model tracing
  - that is, relating observed actions with unobservable cognitive states
  - in essence, "think" along with the student and keep track of cognitive state
    - simulate all possible "thought" sequences
    - find which model sequence matches human behavior
    - make the best matching sequence the current estimated cognitive state
    - "cognitive state" = state of the cognitive model

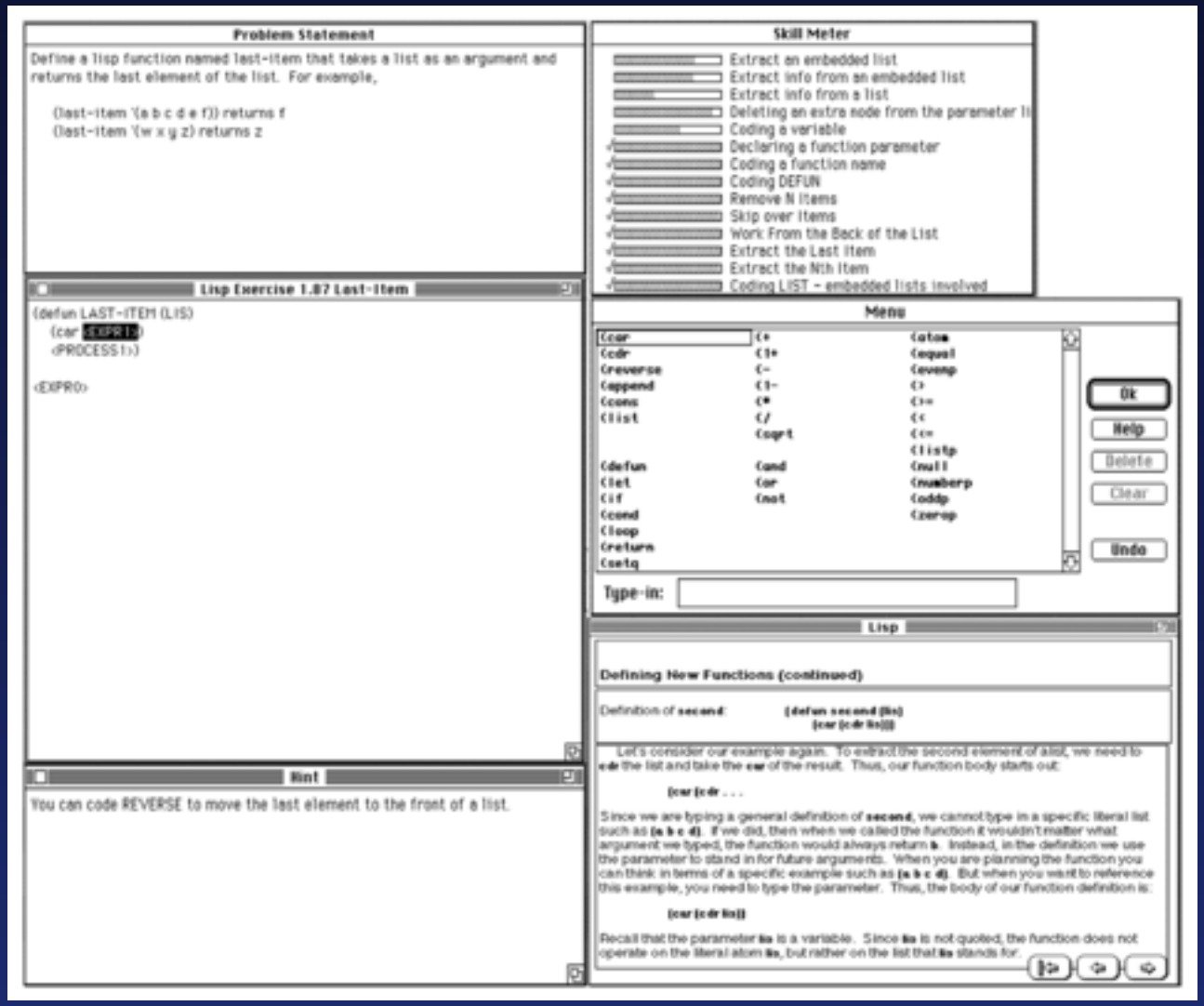
Model Tracing



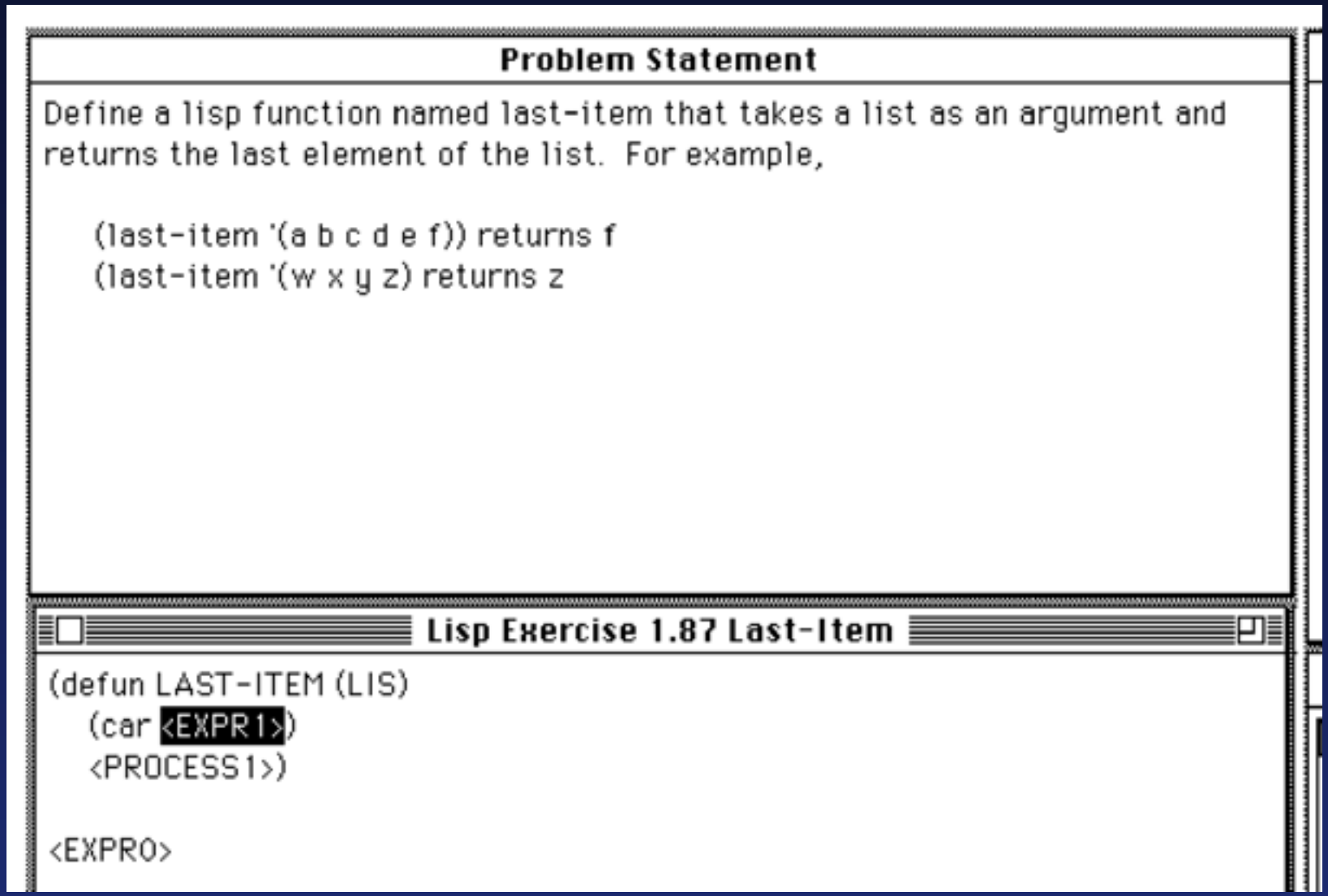
Model Tracing



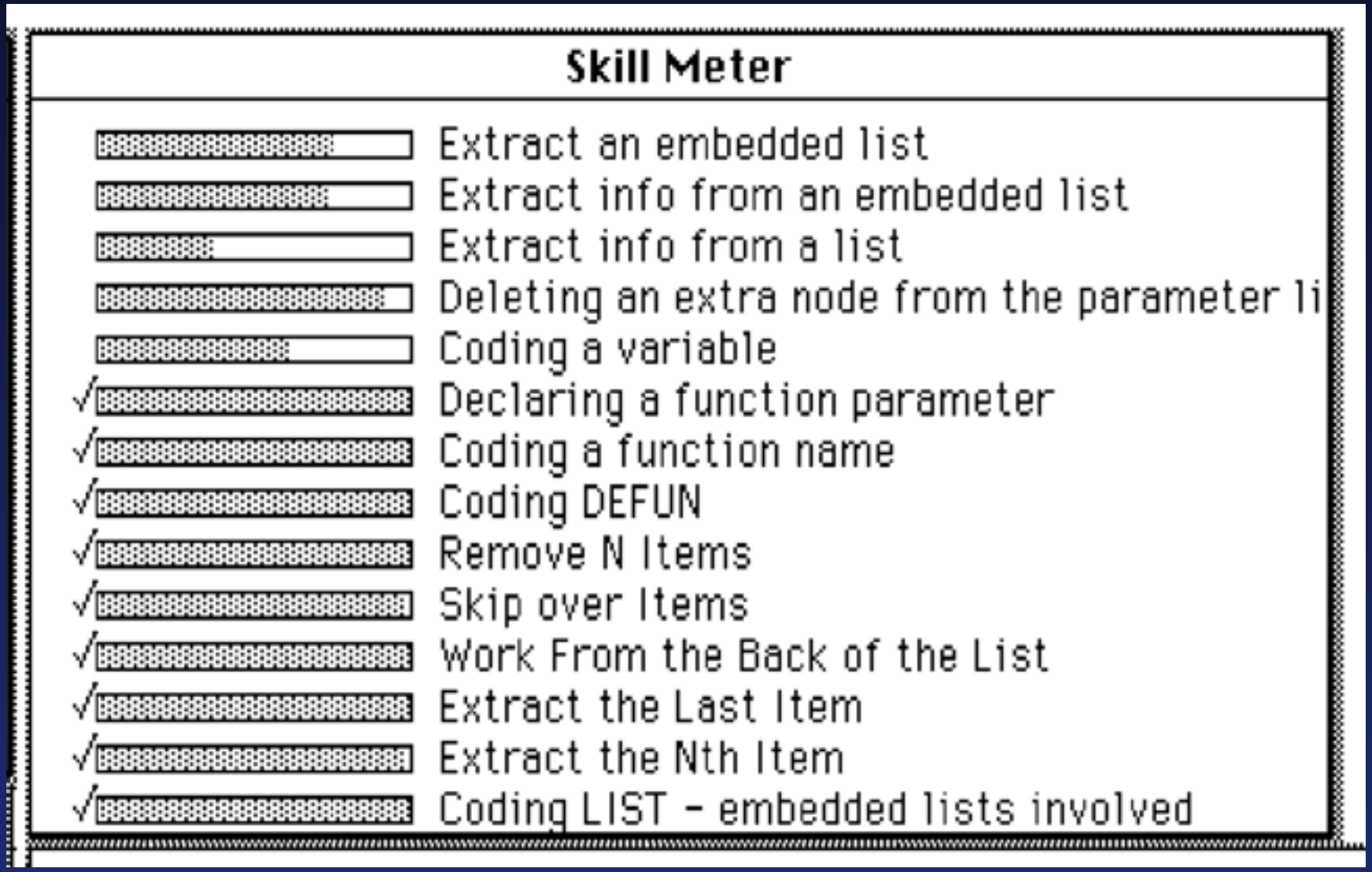
ACT Programming Tutor (Corbett et al.)



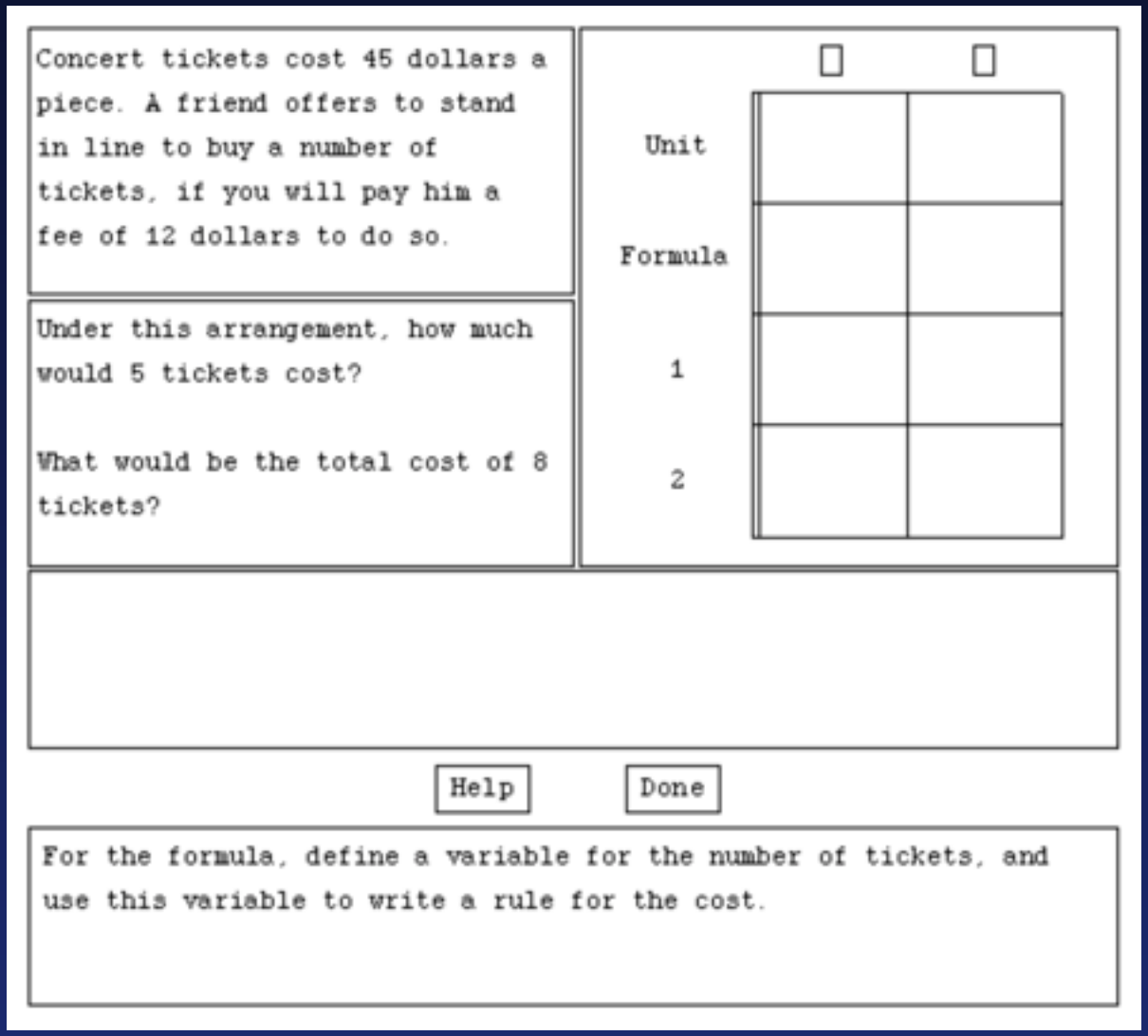
ACT Programming Tutor (Corbett et al.)



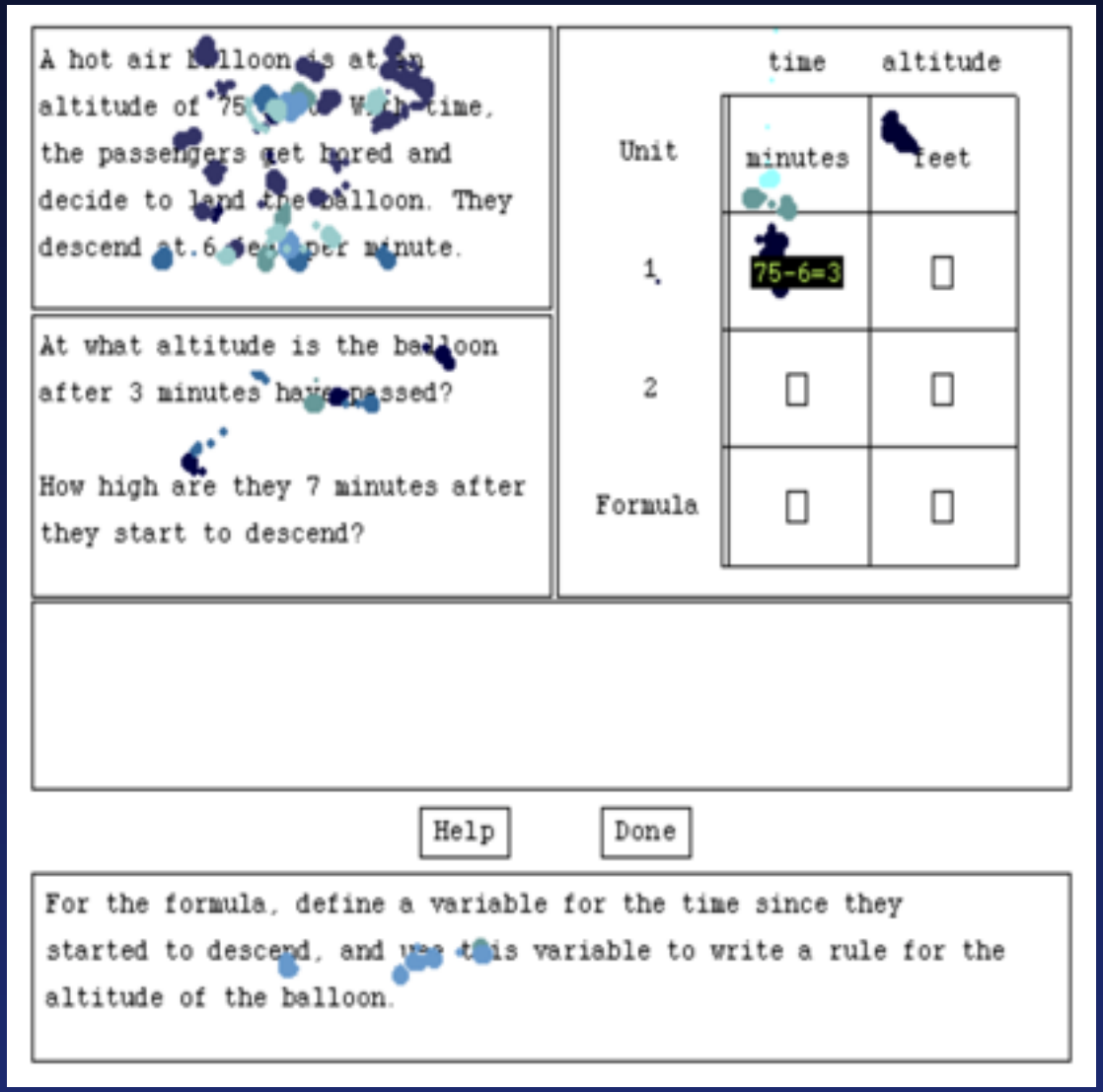
ACT Programming Tutor (Corbett et al.)



Eye Movements & Tutoring (Gluck)



Eye Movements & Tutoring (Gluck)



Eye Movements & Tutoring (Gluck)

- Eye movements helped to understand…
  - Failure to read bug messages
  - Disambiguation of an error
  - Time off-task
  - and more
- Key: Looking for "instructional opportunities" that would not be present without eye movements
  - Human tutors can have a huge impact (e.g., raise a student 2 standard deviations from mean!)
  - Computer tutors can manage about 1 standard deviation
    - maybe measures like eye movements can help close the gap

Intelligent Tutoring

- So, there were models of programming built into these intelligent tutors
- And there have been tutors that (sort of) use EMs
- But interestingly…
  - there are many tutors today (for 300,000+ students)
  - but no (ACT-R) programming models since 1990s!
  - tutors all based on 1990s cognitive architecture, with 1 rule ≈ 1-5 seconds of action
    - modeling behavior at a more strategic level
  - modern architectures: 1 rule ≈ 50-250 ms of action
    - good at predicting eye movements, but harder to build tutors at the strategic level

Modeling & Programming

- What lessons might we draw from ACT-R's history of modeling, tutoring, and eye movements?
  - Modeling the cognitive processes in programming is difficult, but can be done.
    - the cognitive architecture can help guide representations and skill sets to those that are psychologically plausible
  - There is sometimes a large leap from eye-movement patterns to cognitive strategies.
    - Model Tracing is very relevant and may help here
    - this is where the levels of abstraction come in… at what level are you most interested in learning about?
- So how do we map eye movements to strategies in computer programming?

Modeling & Programming

- EMIP 2014 coding scheme

Patterns (observable)

Flicking
JumpControl
JustPassingThrough
LinearHorizontal
LinearVertical
RetraceDeclaration
RetraceReference
Scan, Signatures
Thrashing
Word(Pattern)-Matching

Strategies (unobservable)

AttentionToDetail
DataFlow
Debugging
Deductive
DesignAtOnce
FlowCycle
Inductive
Interprocedural-ControlFlow
Intraprocedural-ControlFlow
StrayGlance
TestHypothesis
Touchstone
Trial&Error
Wandering

Modeling & Programming

- Possible patterns/ low-level strategies:
  - read everything (syntax checking?)
  - read words (not punctuation)
  - read method names, speak them out
- Models have…
  - ACT-R rules that specify the strategy
  - ACT-R architecture that drive the underlying processes (e.g., visual attention shifts leading to eye movements)

```
public class Rectangle {
        private int x1 , y1 , x2 , y2 ;
        public Rectangle ( int x1 , int y1 , int x2 , int y2 ) {
                this.x1 = x1 ;
                this.y1 = y1 ;
                this.x2 = x2 ;
                this.y2 = y2 ;
        }
        public int width ( ) { return this.x2 – this.x1 ; }
        public int height ( ) { return this.y2 – this.y1 ; }
        public double area ( ) { return this.width ( ) * this.height ( ) ; }
        public static void main ( String [ ] args ) {
                Rectangle rect1 = new Rectangle ( 0 , 0 , 10 , 10 ) ;
                System.out.println ( rect1.area ( ) ) ;
                Rectangle rect2 = new Rectangle ( 5 , 5 , 10 , 10 ) ;
                System.out.println ( rect2.area ( ) ) ;
        }
}
```

Conclusions

- Eye movements help us understand cognition
  - BUT they are not the only useful source of data, and are best viewed as complementary with other data
- Cognition helps us understand eye movements
  - BUT the hidden connection between the eyes and the mind will always make this a non-trivial process
- Models help us understand both
  - BUT there are many types of models — all of them are "wrong" :) but many of them are useful!
  - plus, models can be used for engineering development of model-based systems (like tutoring systems)

ACT-R Modeling

- If you'd like to see/run a working model, please download and extract…

http://cog.cs.drexel.edu/emip15.zip

- Launch "ACT-R-prog.jar"
  then open "Program1.actr"
  and click "Run":

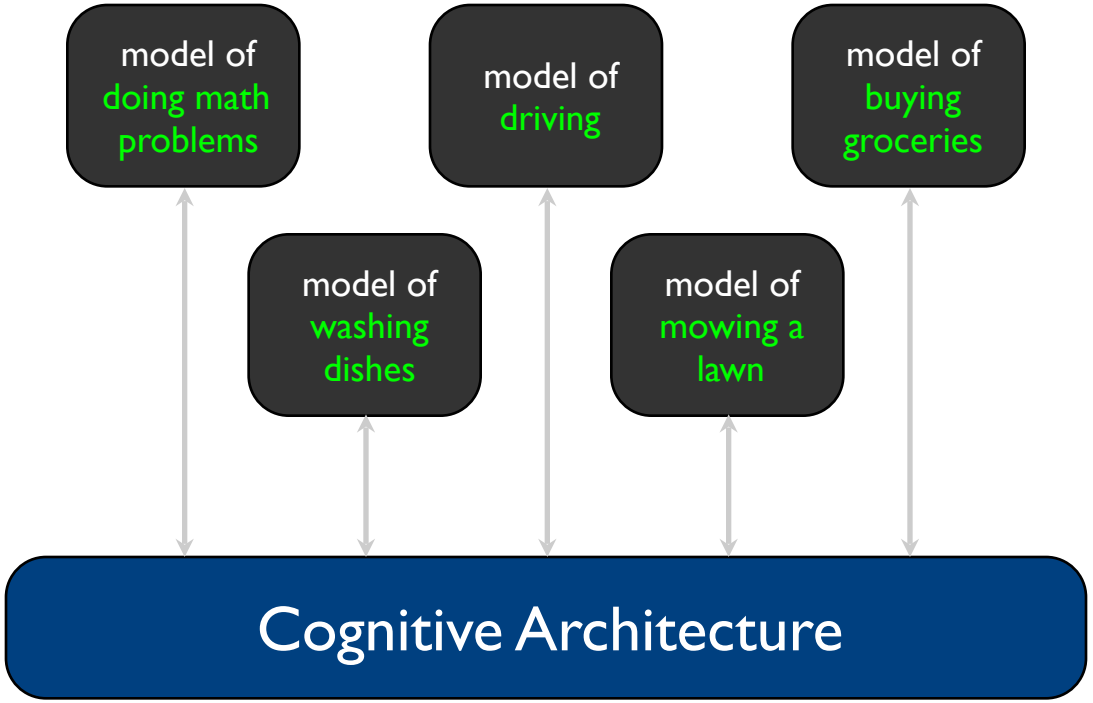
ACT-R Modeling

- The canonical implementation is in LISP:
  - http://act-r.psy.cmu.edu
  - like many (most?) systems developed since the early years of AI, Cognitive Science
- LISP is a cool language, but it's often inconvenient
  - environments are uneven across platforms
  - task/interface development is more difficult
- We will try an ACT-R system implemented in Java
  - actually, you don't need to know Java at all to use it (ACT-R has its own language)
    - though you do need Java to program new tasks
  - your "ACT-R-prog.jar" has a programming task built-in

ACT-R Modeling

Domain "Software"
knowledge for
individual domains

Brain "Hardware"
core mechanisms
that all people have



Model Overview

```
(set-task "prog.Programming")

(sgp
   :emma t
   :visual-num-finsts 300
   :visual-first-span 300
   :visual-movement-tolerance 1
   :v t
)

(add-dm
   (goal isa read)
)
(goal-focus goal)

(p read*find-line
...
)
```

Task Specification

Parameter Settings

Declarative Memory

Goal Specification

Production Rules

Task Code

```
public class Programming extends actr.task.Task {
    private String[] TEXT = {
        "public class Rectangle {“,
        "private int x1 , y1 , x2 , y2 ;”,
        …
    };

    public Programming() { … }

    @Override
    public void start() {
        // draw TEXT on the screen using labels
        processDisplay();  // to register items with visual system
    }
}
```

Model Overview

```
(p read*find-line
   =goal>
      isa read
      line nil
   ?visual-location>
      state free
      - buffer requested
   ?visual>
      state free
      buffer empty
==>
   +visual-location>
      isa visual-location
      screen-y lowest
      :attended nil
)
```

Define a production "read*find-line"
IF the goal…
is of type "read"
and the line is currently empty
and the visual-location status…
is free
with an empty buffer
and the visual status
is free
with an empty buffer
THEN
start a new visual-location process
for some visual location
at the lowest 'y' coordinate
that has not yet been attended

Model Overview

```
(p read*note-line
   =goal>
      isa read
      line nil
   =visual-location>
      isa visual-location
      screen-y =y
==>
   =goal>
      line =y
)
```

Define a production "read*note-line"
IF the goal…
is of type "read"
and the line is currently empty
and the visual-location contains a chunk…
of type visual-location
at a particular 'y' coordinate (=y)
THEN
change the goal
to note the found 'y' coordinate

Model Overview

```
(p read*find-token
   =goal>
      isa read
      line =y
   ?visual-location>
      state free
      - buffer requested
   ?visual>
      state free
      buffer empty
==>
   =goal>
      line =y
   +visual-location>
      isa visual-location
      screen-y =y
      screen-x lowest
      :attended nil
)
```

```
(p read*encode-token
   =goal>
      isa read
      line =y
   =visual-location>
      isa visual-location
   ?visual>
      state free
      buffer empty
==>
   +visual>
      isa move-attention
      screen-pos =visual-location
)
```

```
(p read*continue-line
   =goal>
      isa read
      line =y
   =visual>
      value =text
==>
   !output! (=text)
)
```

```
(p read*done-line
   =goal>
      isa read
      line =y
   ?visual-location>
      state error
==>
   =goal>
      line nil
   -visual-location>
)
```

Simulation Run

```
> (run)

     0.000   vision           unrequested [vision~63]
     0.000   procedural       start
     0.049   procedural       ** READ*NOTE-LINE **
     0.098   procedural       ** READ*FIND-TOKEN **
     0.098   vision           find-location [vision~66]
     0.147   procedural       ** READ*ENCODE-TOKEN **
     0.147   vision           move-attention
     0.168   vision           encoding-complete [word~69]
     0.217   procedural       ** READ*CONTINUE-LINE **
"public"
     0.266   procedural       ** READ*FIND-TOKEN **
     0.266   vision           find-location [vision~72]
     0.282   eye              preparation-complete [word~69]
     0.315   procedural       ** READ*ENCODE-TOKEN **
     0.315   vision           move-attention
     0.356   eye              execution-complete [word~69]
     0.370   vision           encoding-complete [word~75]
     0.419   procedural       ** READ*CONTINUE-LINE **
"class"
```