Unit 3 Attention and Actions

Dan Bothell

Department of Psychology

Carnegie Mellon University

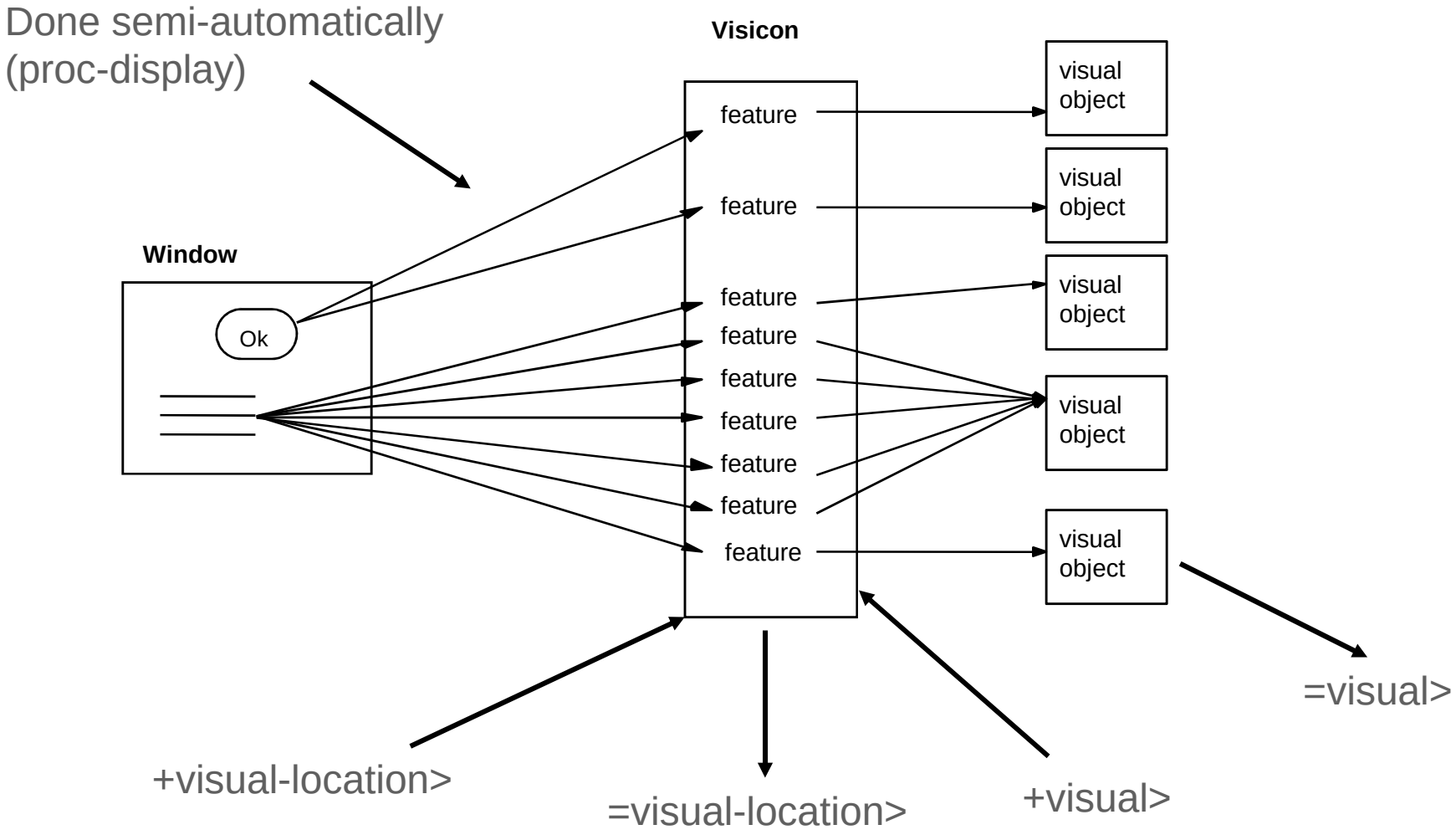
Perceptual-Motor Modules & Buffers

- Encode the world (input)
  - Vision Module
    - visual
    - visual-location
  - Audition Module
    - aural
    - aural-location
- Perform actions (output)
  - Motor Module
    - manual
  - Speech Module
    - vocal

Vision Module

- "Where" system
  - Provides information about visual locations
  - Ask for locations with certain properties
    - X and Y location, size, color, etc.
  - Very fast, places results in visual-location buffer
- "What" system
  - Model of visual attention
  - Shift attention to an object
    - A chunk is created representing that object
    - Placed in the visual buffer
  - Takes time
  - Only one visual attention shift at a time

Visual Icon


Done semi-automatically
(proc-display)
Visicon
Window
Ok
feature
feature
feature
feature
feature
feature
feature
feature
visual object
visual object
visual object
visual object
visual object
+visual-location>
=visual-location>
+visual>
=visual>

Cognition-Vision Interface

- Requests to the “where” system (visual-location buffer):
  - Specify a set of constraints
    - Visual properties of the object (e.g. “color red”)
    - Spatial location of the object (e.g. “screen-x less-than 153”)
  - Returns a result immediately
  - Supports things like “pop-out” search

- A request to the “what” system (visual buffer)
  - Only need to specify a visual location
  - Attention shifts to what is there
  - Default timing is a fixed cost of 85 ms

Find-shift-harvest Paradigm

- Three production sequence typically used to access visual information

- Find
  - Request a chunk of “where” information be put into the visual-location buffer
  - Produces a FIND-LOCATION event in the trace
  - This step can occasionally be skipped

Example Find

IF

the goal is to find a red letter

and the visual-location buffer is empty

THEN

request a visual-location specifying
color and object type

```
(P example-find
 "find a red text item"
   =goal>
      ISA         report-red
   ?visual-location>
      buffer      empty
==>
   +visual-location>
      ISA         visual-location
      color       red
      kind        text
)
```

Find-shift-harvest Paradigm

- Shift
  - Harvest the visual-location chunk
  - Initiate an attention shift with a request to the visual buffer
    - Remember to check that the module is free
  - Produces a MOVE-ATTENTION event

Example Shift

IF
 the goal is to find a red letter

and a visual location was found

and the vision module is not busy

THEN
 shift attention to that location

```
(P example-shift
   "Shifts attention to the object"
   =goal>
      ISA         report-red
   =visual-location>
      ISA         visual-location
   ?visual>
      state    free
==>
   +visual>
      ISA         move-attention
      screen-pos  =visual-location
)
```

Find-shift-harvest Paradigm

- Harvest
  - After attention shifts and the item is encoded for the visual buffer
    - takes time to move attention
    - The ENCODING-COMPLETE event in the trace signals that vision is done
  - Harvest the chunk from the visual buffer

Example Harvest

IF

the goal is to find a red letter

and a text visual object with some value is available

THEN

press the corresponding key

```
(P example-harvest
   "report value of the object"
   =goal>
      ISA         report-red
   =visual>
      ISA         text
      value       =letter
==>
   +manual>
      ISA         press-key
      key         =letter
)
```

Visual-location Requests

- Return a chunk with features that match
  - If more than one matches it picks the “newest”
  - If still multiple matches then randomly from those
- Can use any visual-location slots in the request any number of times with any of the modifiers
  - -,<,>,<=,>=
  - The inequality tests only work for numbers
- Three special values that can be used
  - Lowest/Highest
    - Only works for numbers
    - Match fixed specifications first
    - From those pick the one with the least/greatest value in the slot
    - If multiple relative constraints process them “left to right”
  - Current
    - Matching the value of the last visual-location chunk attended

```
VISUAL-LOCATION
   SCREEN-X
   SCREEN-Y
   DISTANCE
   KIND
   COLOR
   VALUE
   HEIGHT
   WIDTH
   SIZE
```

```
+VISUAL-LOCATION>
   isa visual-location
   kind  text
   color current
   size  highest
 > screen-x 50
 <= screen-x 100
   screen-x lowest
 <= screen-y current
```

:Attended

- One of the most common things to test for in a visual-location request is the attended state
  - Not an actual slot of the visual-location chunk-type
    - A request parameter :attended
- Three possible values
  - T, meaning the location is marked as attended
  - NIL, meaning it is not marked as attended
  - NEW, meaning it is not marked as attended
    AND the item is a new entry into the visicon (last 500 ms)
- Shifting attention to a location will automatically mark it as attended
  - even if the result is never explicitly harvested

Attention markers

- The attended markers for visual-locations are called Finsts (fingers of instantiation)
- They are limited both in number and duration
- Defaults to 4 finsts that last 3 seconds
  - Can be changed with parameters
- When limit exceeded the oldest one is reused
  - The feature that was marked as attended t then reverts to attended nil

Audition

- Works a lot like vision
- Find happens via aural-location request
  - Very often not needed
- Shift auditory attention with an aural request for sound
- Harvest result from the aural buffer
- The timing is different from vision
  - Events take time to enter the audicon
  - Attentional latency is longer than vision
  - Detection and encoding depend on type of sound
    - Tones, words, numeric digits, user defined

```
+aural-location>
          isa       audio-event

+aural>
          isa       sound
          event     =aural-location

=aural>
          isa       sound
```

Buffer Stuffing

- Previously mentioned not needing a visual-location or aural-location request
- Modules have limited bottom-up control
- When there is a change in the world the perceptual modules may put an item into the *-location buffer
  - If the buffer is currently empty and an appropriate item exists
- What gets selected can be controlled
  - Default for visual-location is left-most unattended item
  - Default for aural-location is any sound

Motor Module

- Movement is divided into two phases: preparation and execution
- Can only prepare one movement at a time, and can only execute one movement at a time
  - But, can prepare one while executing another
- Manual buffer queries allow testing detailed state
  - preparation free/busy
  - execution free/busy

Motor Module actions

- Movement styles (types of movement):
  - **Punch**: simple downstroke followed by an upstroke of a finger, for pressing a key that is already directly below a given finger
  - **Peck**: directed movement of a finger to a location followed by a keystroke, all as one movement
  - **Peck-recoil**: same as "peck" above, but the finger moves back to the location at which it started
  - **Ply**: moves a device (e.g. mouse) to a location in space
- Complex movements typically built on these movements
  - move-cursor is a ply with a visual-location or visual-object destination
  - press-key is either a punch or peck-recoil depending on the key

Example

- Simple typing can be done with the PRESS-KEY “macro”:
  - +manual>
    ISA press-key
    key “C”
  - which is really just a macro for...
  - +manual>
    ISA peck-recoil
    hand left
    finger middle
    r 1
    theta 1.57

Movement Style Components

- “Punch” the key beneath a finger
  - 3 features:
    - Movement style itself “punch”
    - Hand
    - Finger
- “Peck-recoil” – hit a key not below the finger
  - Adds 2 more features:
    - Distance
    - Direction
- Move the mouse to a new location
  - Has 4 features:
    - Movement style itself “ply”
    - Hand
    - Distance
    - Direction

Preparation Time

- Time to prepare a movement is determined by the number of features that need to be prepared
  - 50 ms per feature
- The Motor Module remembers the last movement that was prepared and will re-use features when it can
- Features are hierarchical:
  - A new style will always result in no savings
  - If only finger changes, only that feature needs to be prepared
  - If hand changes, both hand and finger need prep
  - Style > hand > finger

Execution Time

- Depends entirely on the movement being made
- Some movements (e.g. “punch”) have a fixed, short duration
- Other movements (e.g. mouse moves) have a variable duration that is determined by the properties of the movement
  - In the case of an aimed movement, Fitts’ law is used

Preparation + Execution Example

- Press same key twice:

0.050	PROCEDURAL	PRODUCTION-FIRED PRESS-KE	
0.050	PROCEDURAL	MODULE-REQUEST MANUAL	
0.050	MOTOR	PRESS-KEY r	
0.300	MOTOR	PREPARATION-COMPLETE	400 ms
0.350	MOTOR	INITIATION-COMPLETE	
0.450	MOTOR	OUTPUT-KEY #(4 3)	
0.600	MOTOR	FINISH-MOVEMENT	
0.650	PROCEDURAL	PRODUCTION-FIRED PRESS-KE	
0.650	PROCEDURAL	MODULE-REQUEST MANUAL	
0.650	MOTOR	PRESS-KEY r	
0.650	MOTOR	PREPARATION-COMPLETE	150 ms
0.700	MOTOR	INITIATION-COMPLETE	
0.800	MOTOR	OUTPUT-KEY #(4 3)	
0.950	MOTOR	FINISH-MOVEMENT	

Speech Module

- Rough approximation for short utterances
  - Execution time is a linear function of the number of characters to be spoken
- Same kind of prepare-execute system as Motor Module
- Only two commands:
  - SPEAK
    - Output speech which the model and others may hear
  - SUBVOCALIZE
    - Output speech which only the model can hear

```
+vocal>
   Isa speak
   String "hello"
```

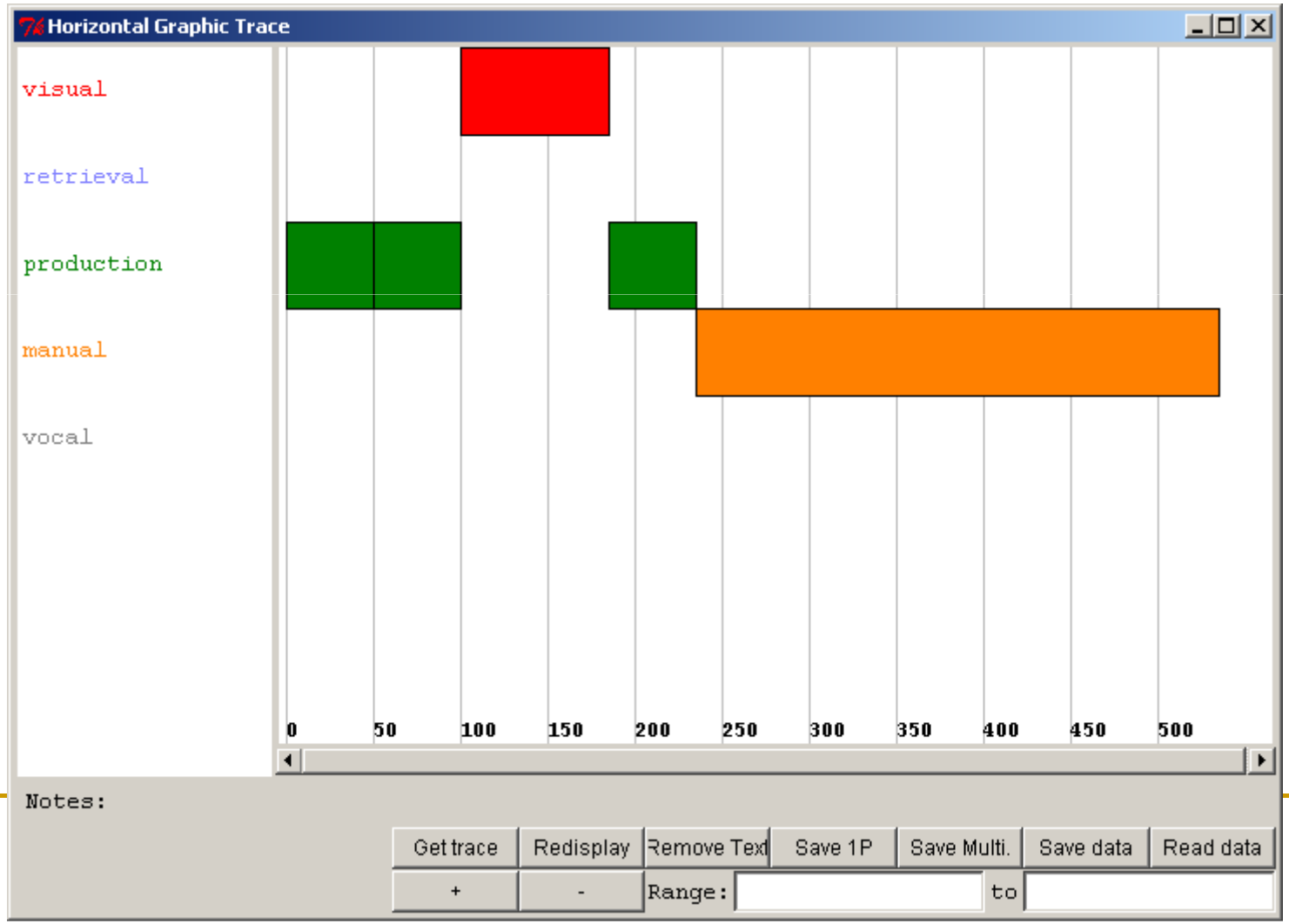
```
+vocal>
   Isa subvocalize
   String "cheese"
```

Serial & Parallel Processing

- Serial
  - Procedural module
    - fires one production at a time
  - Other modules
    - Only perform one request at a time
    - Put a single chunk into their buffers
- Parallel
  - The modules can all operate in parallel
  - Internal mechanisms of a module could be highly parallel
    - Conflict resolution
    - Making a retrieval
    - Finding a visual-location

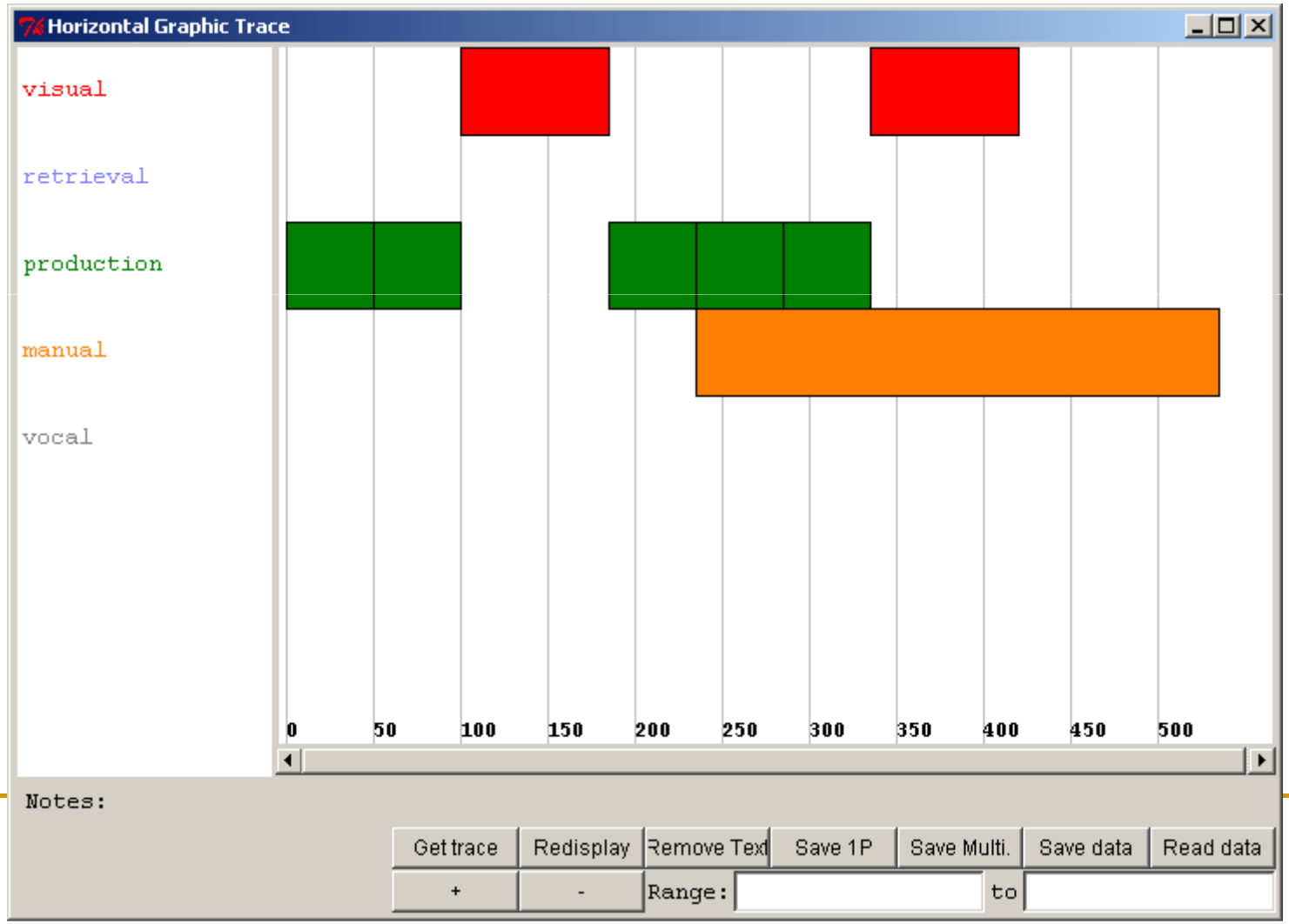
ACT-R and Parallelism

- See something, type it



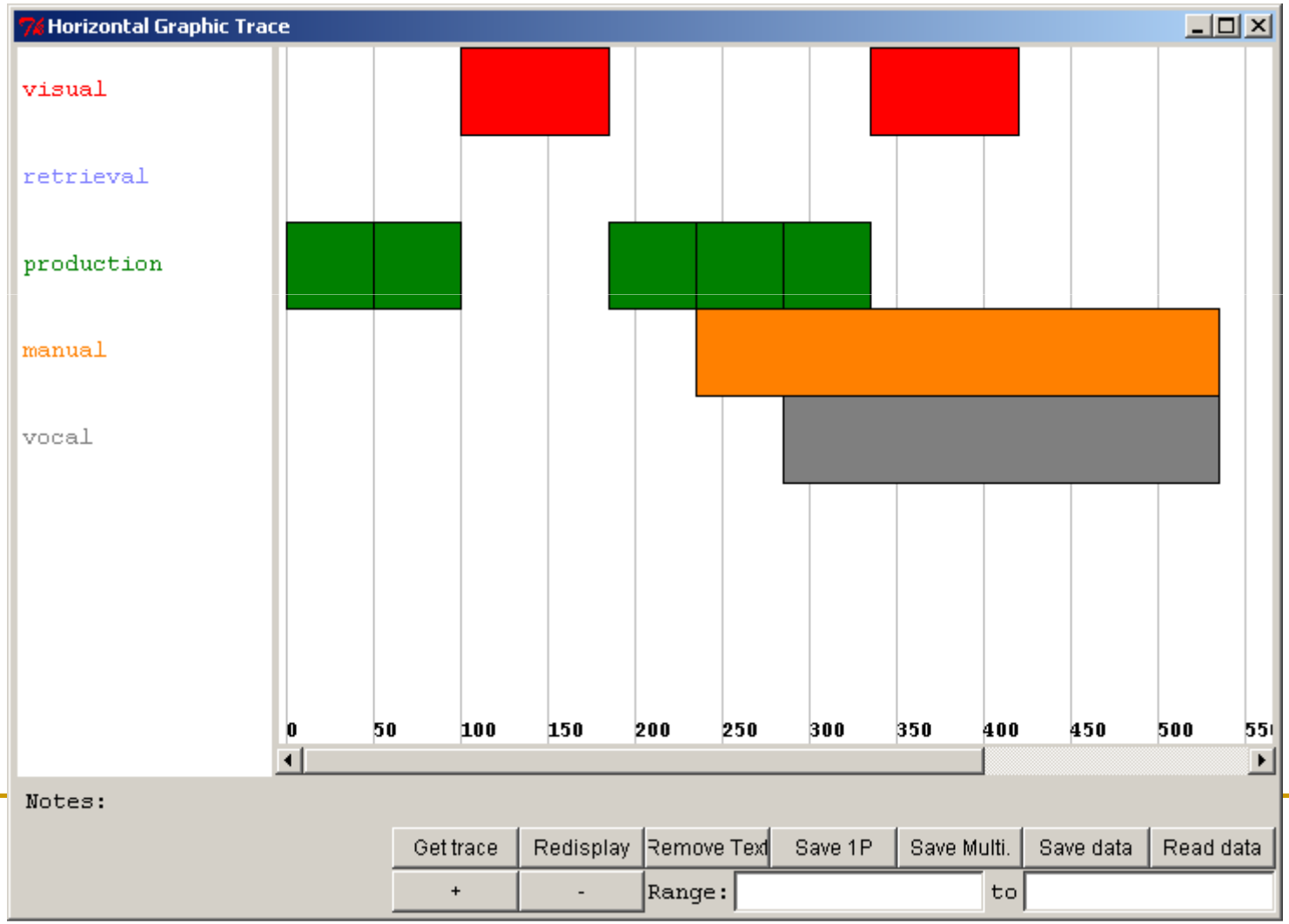
ACT-R and Parallelism

- See something, type it, look at next



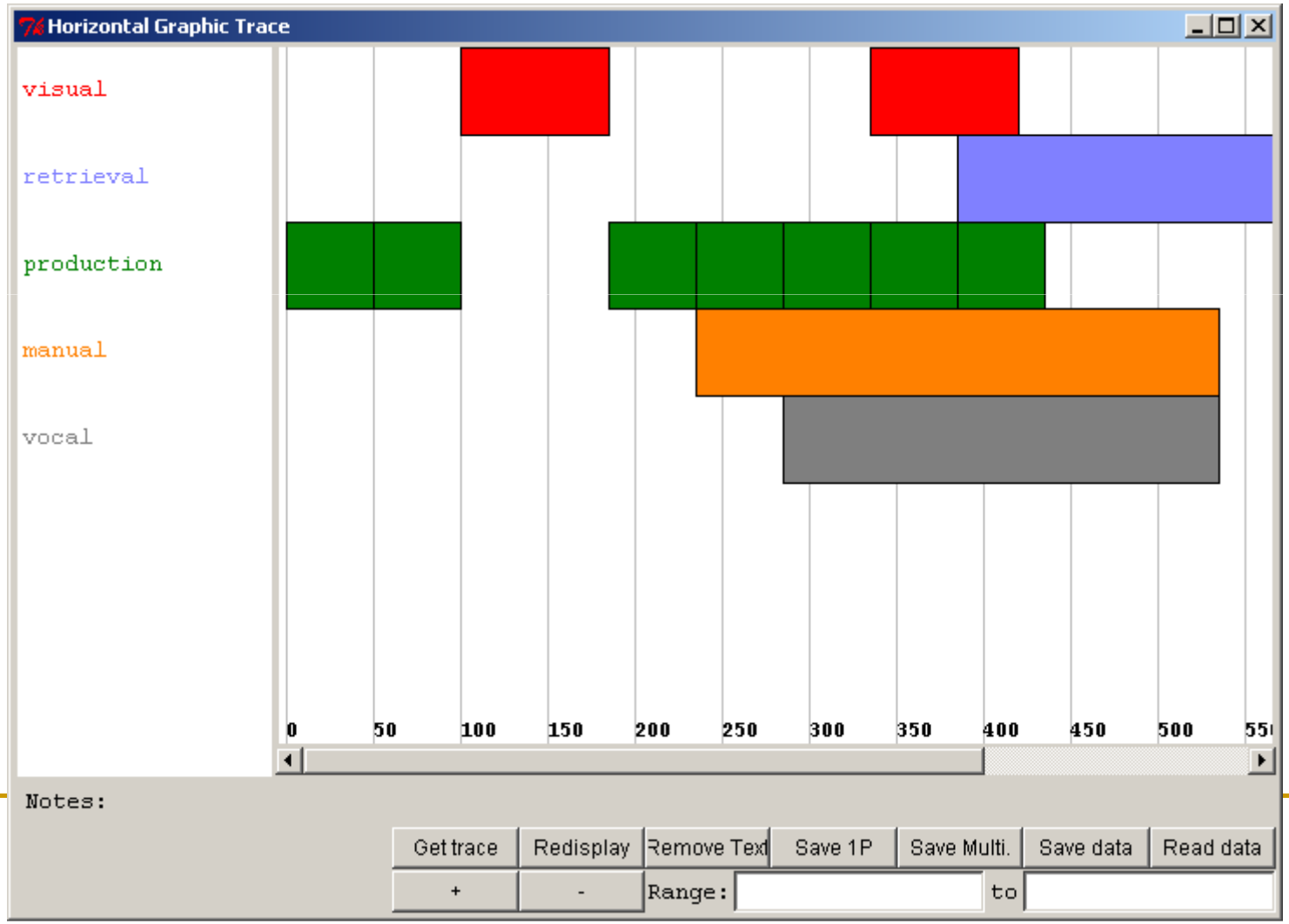
ACT-R and Parallelism

- See something, type it, say it, look at next



ACT-R and Parallelism

- See something, type it, say it, look at next, remember something



Data Fitting

- From now on the assignment models will be compared to human performance
  - Mostly Response time
    - Correlation and Mean deviation
- Provides a way to compare and judge the models
- Not the only way
  - Plausibility
  - Generality
  - Simplicity
- Make sure the model does the right thing before trying to tune it with parameters!

Example: Sperling

- Classic visual icon experiment
  - Present three rows of letters very briefly
  - Report back as many as you can
  - Row cued by tone (high, middle, low)
- The ACT-R model
  - Uses slightly different timing for reasons explained in the unit
  - Starts encoding items at random until tone is processed
  - Once tone is processed, encode only from the cued row
  - When everything disappears, report
- Good idea to make sure you understand this model before you start the assignment

Assignment: Subitizing

- Simple task: A bunch of objects appear on the display, report the number
  - Model must respond by speaking the answer
- Model starts with the counting facts from 0-11
- Will need to manage visual attention
  - Make sure the model gets to every item
  - Needs to know when its done
  - Given 10 finsts with a long duration to start
    - Do not have to use that if you do not want to
- Should not need to adjust parameters to get a reasonable fit to the data

Debugging Tips

- Check the visicon
  - Call (print-visicon) at the prompt
  - Visicon button in the environment
  - Will show you where things are, what they are, and their attended state
- Check the state of the buffers and their modules with the (buffer-status) command or environment button
- Always have the stepper and Why not? tools.

Questions?