ACT-R 6 Proposals

Dan Bothell
db30@andrew.cmu.edu
July 9, 2004

Overview

- General implementation details
  - More specifics and an API at the post-ICCM meeting
- Conceptual changes/Theory updates

Overall Design

- Divided into two sections
  - Framework
    - Organization
    - Common components
    - Constant components
  - Modules
    - Instantiate the system
    - Implementation can vary greatly

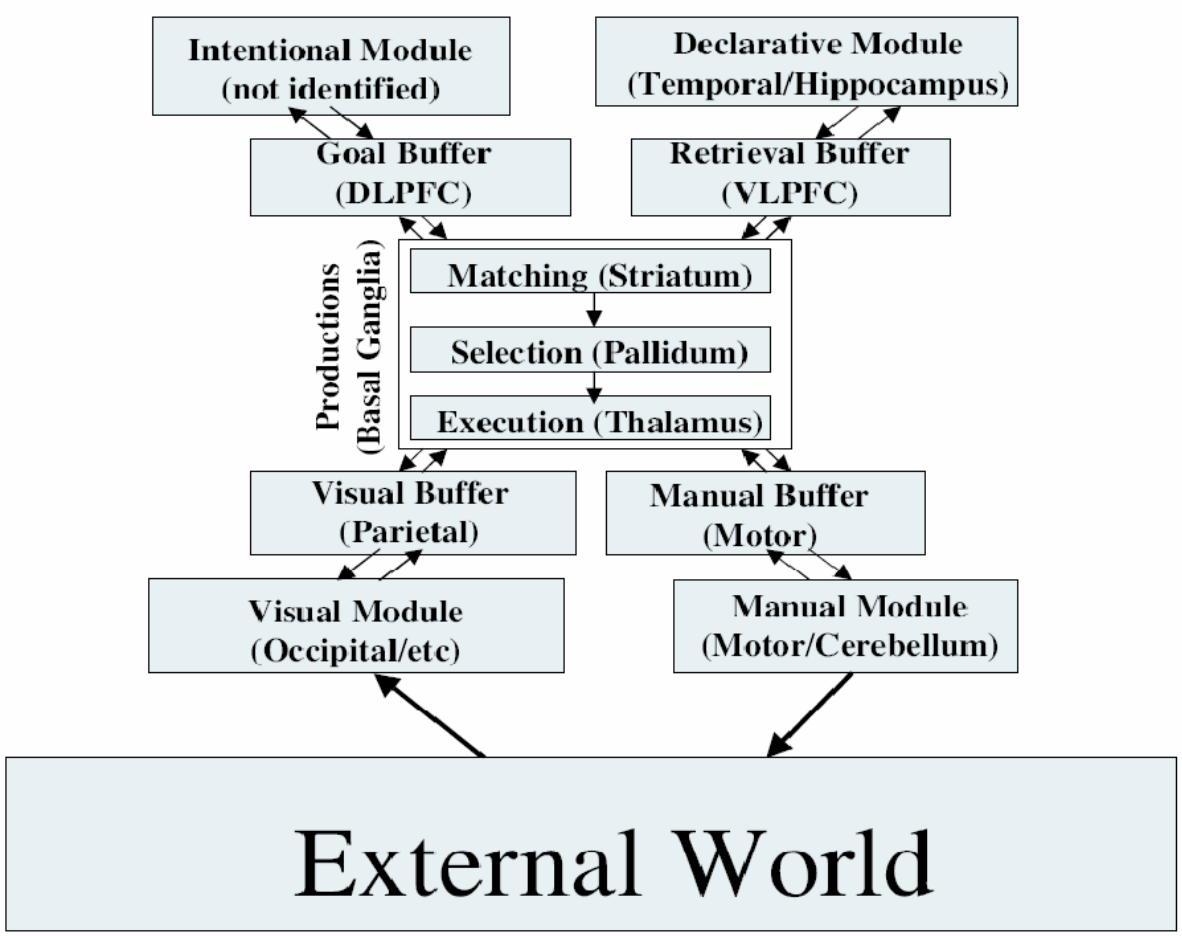
Framework

- Mostly independent of the theory
  - Scheduler
    - Clock
    - Event management
    - Running
  - Data structures
    - Models
    - Modules
    - Buffers
    - Chunks
  - Communication mechanisms
    - Everything goes through the scheduler

Modules

- Theory components
  - Procedural, declarative, visual, etc.
- Support components
  - Naming, random numbers
- Should be easy to add new ones
  - Easily use the work of others
  - Pick and choose components

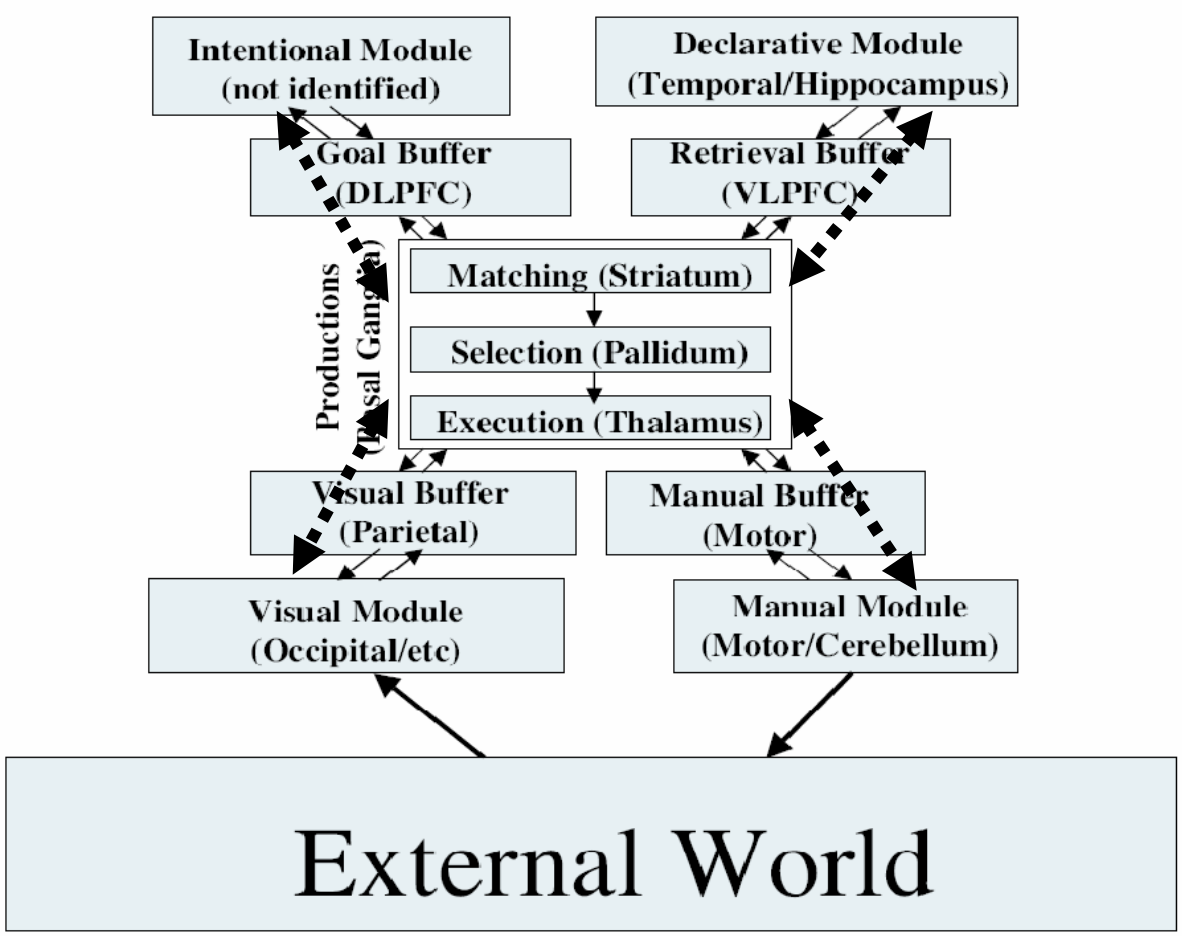
ACT-R 5 Architecture



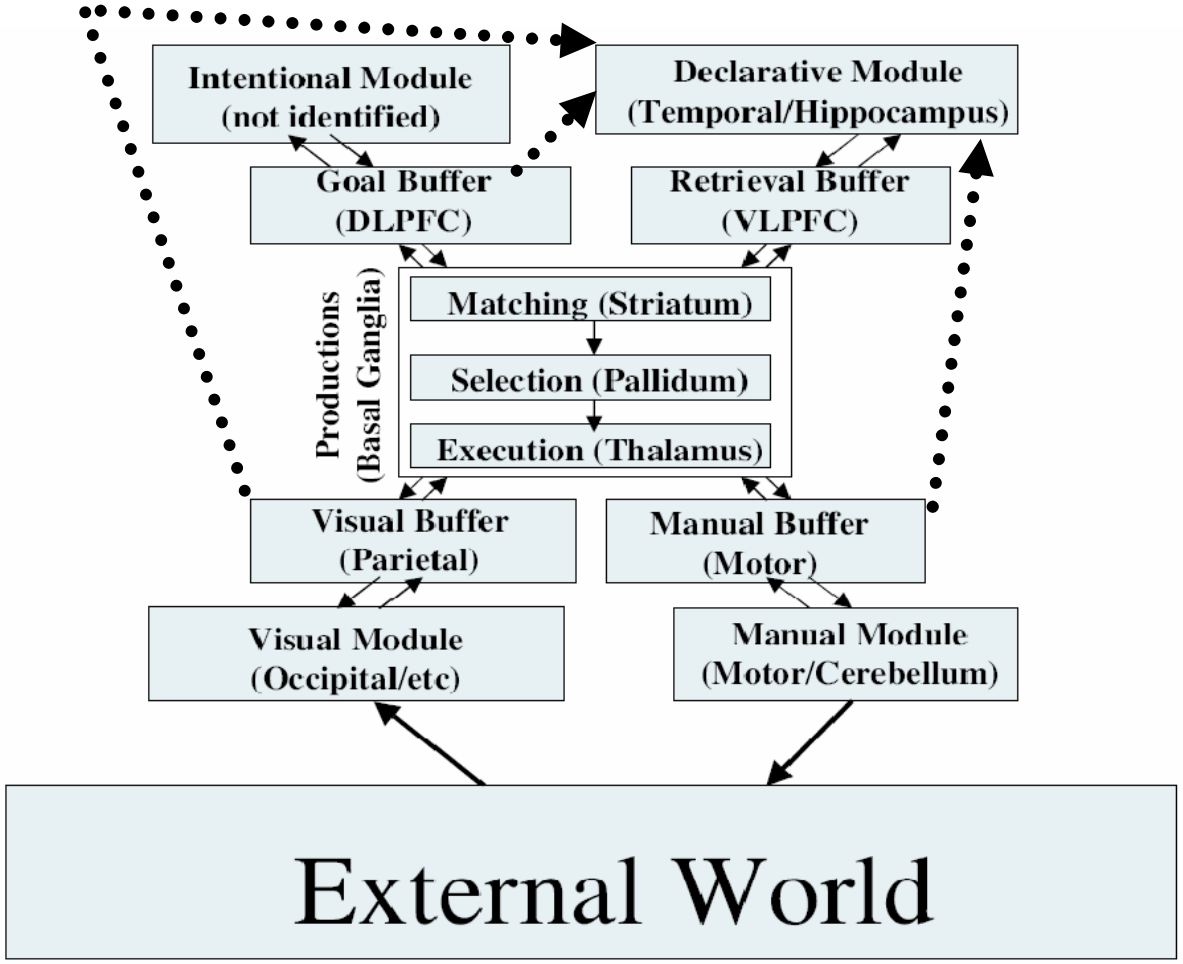
ACT-R 6

- The picture for ACT-R 6 is similar
- Make explicit some of the implicit connections
  - Procedural module ←→ other modules
  - All buffers and Declarative module
  - *Maybe still don't want them in the "official" picture*
- Specify the API and guidelines for adding a module

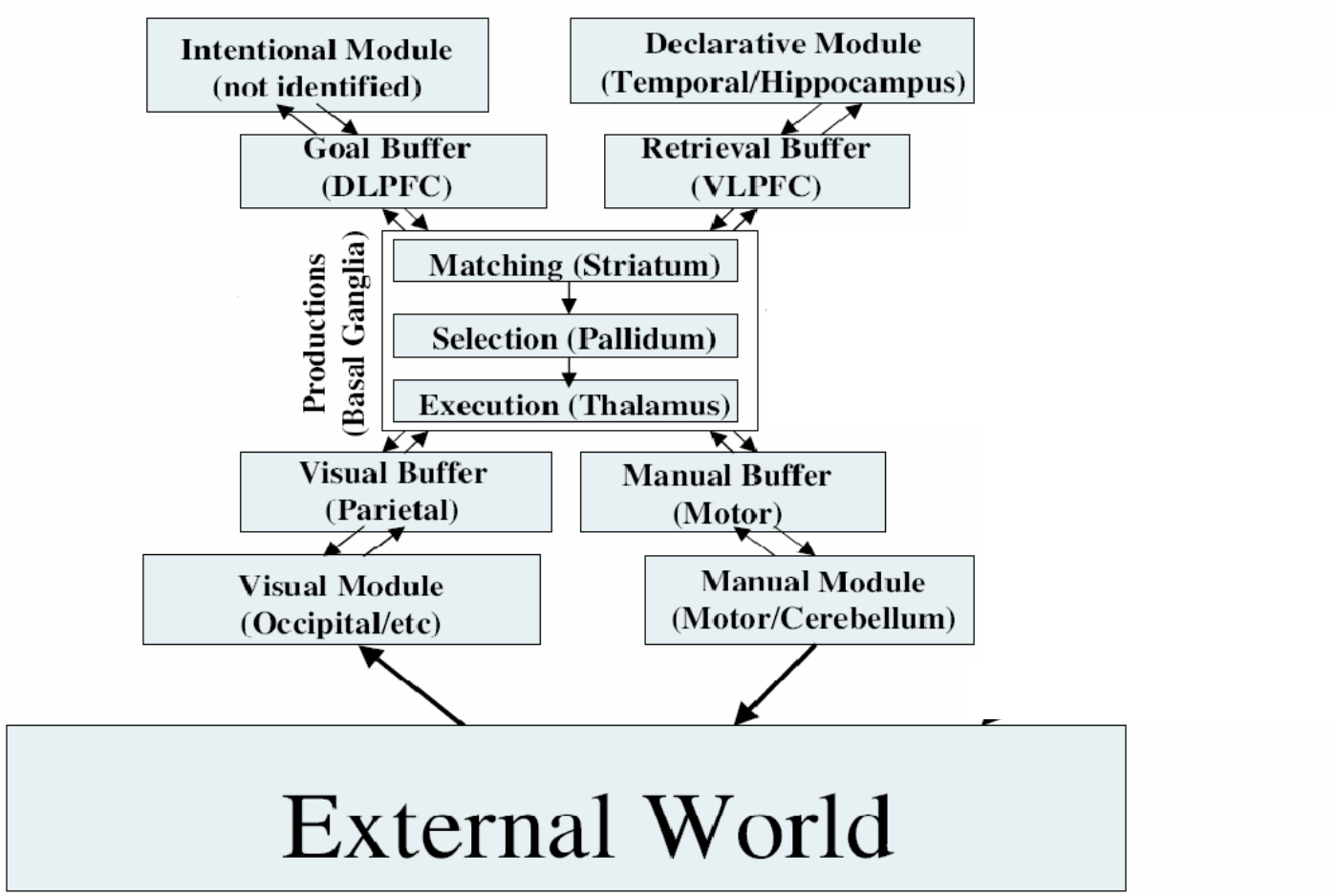
Procedural to Modules “Through” buffers



Chunks go from the buffers to Declarative Memory



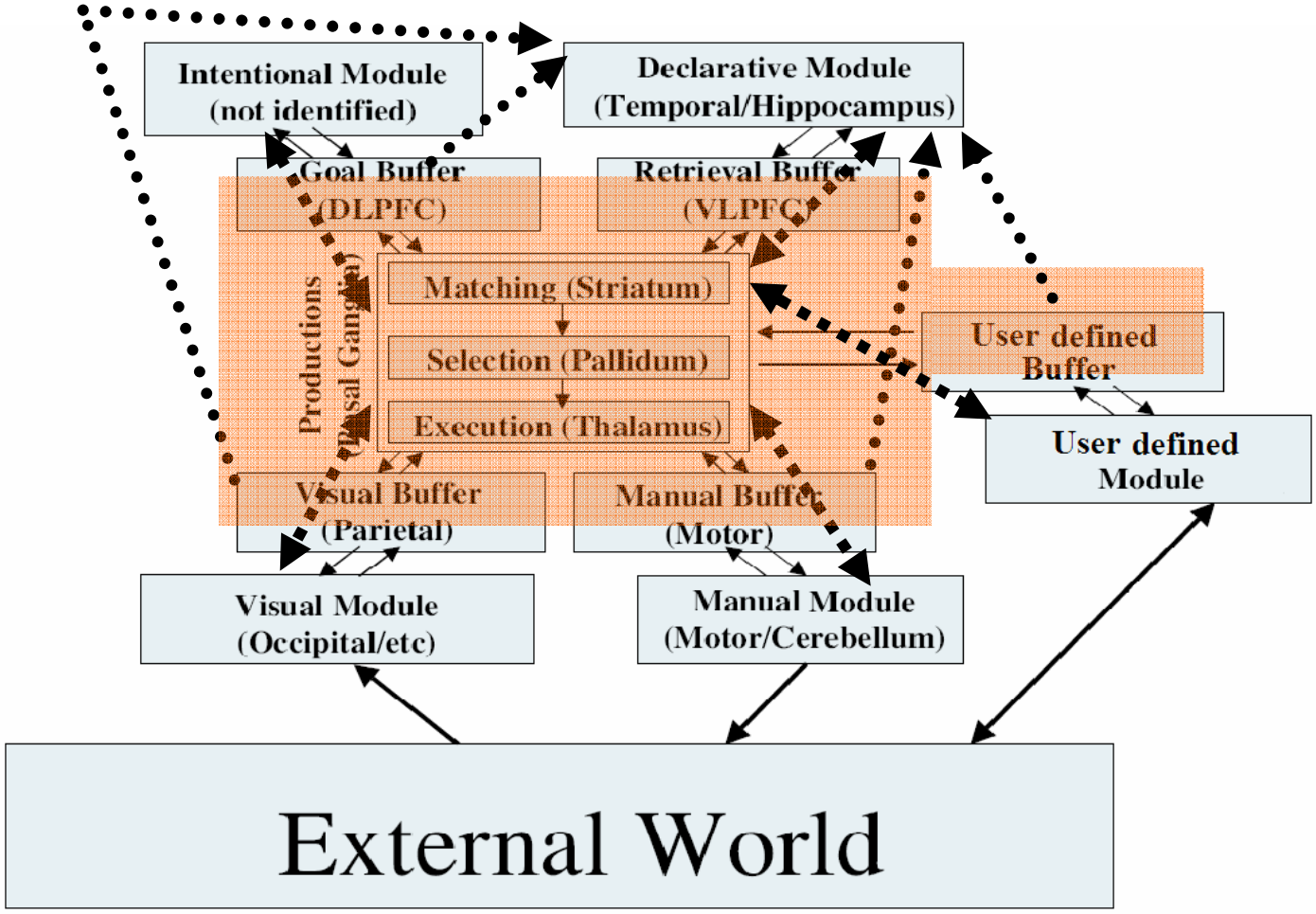
Allow additions to be made easily



ACT-R 6 Buffers

- Unify the operation/implementation
  - Procedural module treats them all the same
  - User added buffers work like the default ones
- Support the claim of ACT-R 5 that buffers are the “source” of the DM chunks

Unchangeable protocol and operation



Buffer Implementation

- Part of the framework
  - Uniform representation
- Every module's buffer(s) act the same with respect to the Procedural module

Differences from ACT-R 5

- Augment the description of the buffer to be not only the module interface but chunk generation “scratch pads”

- The chunks become part of DM when they leave a buffer
  - That’s the claim of ACT-R 5 now, but it’s not actually true!
  - Implies that chunks in buffers exist outside of DM
    - Do we want a new name to differentiate chunks in DM from those outside of DM?

- Proposed operation: When a buffer is cleared the chunk there is merged into DM as happens with goal chunks now

Differences from ACT-R 5 cont.

- Unlike the ACT-R 5 “modular buffer” mechanism for adding new buffers users cannot configure what happens on an =buffer> or –buffer>
- The operations “on the buffers” by productions are constant across all buffers/modules
- Only the +’s provide the communication between the procedural module and other modules
- Should be functionally equivalent
  - (see the RHS + actions later)

Buffer operation

- The buffers don't hold a specific chunk or a "pointer" to a specific chunk
- They hold a copy of a chunk which can be modified freely without changing the original
  - Even for the retrieval buffer!
    - All buffers are treated equally
  - The old claim that DM can't be modified is now enforced from within productions

Why do people modify DM chunks?

- Marking for exhaustive retrieval
  - Seems like a place to extend the theory
  - Don’t have a real mechanism at this point
- Any other reasons ???

Temporary Alternative: Declarative Finsts

- Just like the visual finsts
  - Memory is like perceiving the past
- A parameterized number of markers for recently retrieved chunks with a parameterized decay time
- Retrieval requests could specify it something like

  +retrieval>

  isa some-chunk-type

  slot value

  recently-retrieved *nil*
- Some more specification necessary…

Productions

- No changes to the basic operation
  - Conflict resolution picks one
  - The one selected fires
  - repeat
- No LHS retrievals
  - Not backward compatible
- Unify the buffer/module mechanics
  - A few syntax changes

Production LHS: =buffer

=buffer>

{chunk description}

Tests the contents of the buffer

Same as ACT-R 5

One new suggestion:

=buffer> =variable

Test that the chunk in buffer matches on all the slots of =variable (note the name is not a slot)

Production LHS: -buffer

-buffer>

New LHS test

Tests that the buffer is empty

Production LHS: +buffer

+buffer>

{chunk description}

A test of the module's state

Replaces the ACT-R 5 *-state buffers

More general

- A module may respond to queries other than module-state tests
- Must respond with t/nil immediately
- Should not affect the chunk in the buffer

Production LHS: +buffer cont.

- All modules would be required to respond to certain requests
- Additional requests would be up to the module creator

+buffer> isa module-state	• Same as it is now
+buffer> isa free +buffer> isa busy	• Simplified tests that are equivalent to just testing the modality slot
+buffer> isa error	• Replaces the explicit failure chunk (the buffer should probably be left empty)
+buffer> isa changed/unchanged	• Not quite sure what really wanted or needed – ???

Production LHS: the !'s

!eval! and !bind!

Same as always

The eval buffer idea doesn't quite work

Production RHS: =buffer

=buffer>
{chunk description}*

An immediate modification of the contents of the buffer
Same as ACT-R 5

Two new proposals:

=buffer> =variable
copies the chunk =variable into the buffer
(similar to how +buffer> =variable works now)

=buffer>
more on this later

Production RHS: -buffer

-buffer>

Clears the buffer

That’s all it does – no communication with the module

Production RHS: +buffer

+buffer>

{chunk description}

Sends a request to a module

Similar to what happens now but with some minor additions/updates

Production RHS: +buffer cont.

+buffer> isa request

Just like now

Sends the request to the module and implicitly clears the buffer

+buffer> {slot value}$^{+}$

Without an isa test it's a request for the module to modify the chunk in the buffer instead of a request to replace the chunk in the buffer

Production RHS: the !'s

!eval!, !bind!, !output!, and !stop!

Same as now

Except that !stop! will actually stop the whole system

Gone are !push!, !pop!, !focus-on!, !move-attention!, !delete!, !copy!, !move-mouse!, !click-mouse!, !press-key!, !retrieve!, !send-command!, and !restart!

RHS order of operations

1. Execute all !bind! and !eval! in the order provided
2. All = modifications (no constraint on ordering)
3. Send all + requests (again no guarantee on ordering)
4. all explicit and implicit buffer clearing
5. Any !output!s get printed
6. If there's a !stop! halt the operation after all events at the current time complete

Open production questions

- Extend the variables in productions to allow them in place of
  - Chunk-types
  - Slot names
  - Buffer references (?!)
- Adds flexibility that isn't there now
- Introduces need for more "run-time" checking
- Does anybody have a reason why they need such a thing?

- Current proposal is to add none of that

Strict Harvesting

If a production matches a chunk in a buffer on the LHS then unless it is used on the RHS that chunk will be automatically cleared from the buffer after the production fires

Why?

- “When” to clear a buffer a confusing issue for students (mostly visual/visual-location)
  - Eliminates the need for most –buffer> RHS calls
- Goes well with the BLL suggestion (later)
- Solves a problem with Production compilation
- Doesn’t really break any existing mechanism
  - “Empty” RHS modification keeps it around
    - Adding an “=buffer>” on the RHS of an ACT-R 5 production all that’s necessary for updating

- Current proposal is to add it

Sources of Activation

- Should all buffers be sources or only the Goal buffer?
- Current proposal is that all buffers have a parameter (like :ga/W currently) which the declarative memory system can use
  - If they default to 0 it's no different than now
  - Interesting side note: if the retrieval buffer's W were negative, then it would work to inhibit re-retrieval of a chunk matching the current chunk in the buffer

Base-Level Learning

- Current mechanism is specific to the retrieval buffer
- Also differs from the ACT-R 4 concept of what constitutes a reference
  - Each LHS =retrieval credits a reference
  - ACT-R 4 required an actual retrieval for a reference
  - Are people using “multiple references”???

BLL proposals

- Generalize it to all buffers
- Make explicit the distinction between the Procedural and Declarative modules

Simplify what constitutes a reference

- The merging of the chunk into DM is the only source of a reference
  - Covers the current cases of creation and merging
  - Makes the separation of Procedural and Declarative
  - Works for all buffers

- Closer to the ACT-R 4 mechanism
- Coupled with Strict Harvesting it is almost identical to the current ACT-R 5 if one doesn't "reuse" the retrieval

ACT Equations 4.3 + 4.4

- Remove Equation 4.4 (production strength)
  - Basically removed in ACT-R 5 already
- What about Eq 4.3 (posterior strength equation)
- When, which chunk, what buffer(s)?

- Remove Equation 4.3 as well

Production Compilation

- Strict Harvesting provides a way to avoid a serious current problem
  P1 → P2 → P3 sharing a retrieval

- Extend it to more buffers
  - Not just goal and retrieval
  - Still want a “safe” mechanism
- Develop a general mechanism applicable to all buffers
- Buffers would have a parameter that specifies whether it is safe to compile across or not

Equation 4.3

$$R_{ji} = \frac{assoc * R^{*}_{ji} + F(C_j)E_{ji}}{assoc + F(C_j)}$$

$$S_{ji} = \ln(R_{ji})$$

Back