DRONACHARYA
College of Engineering

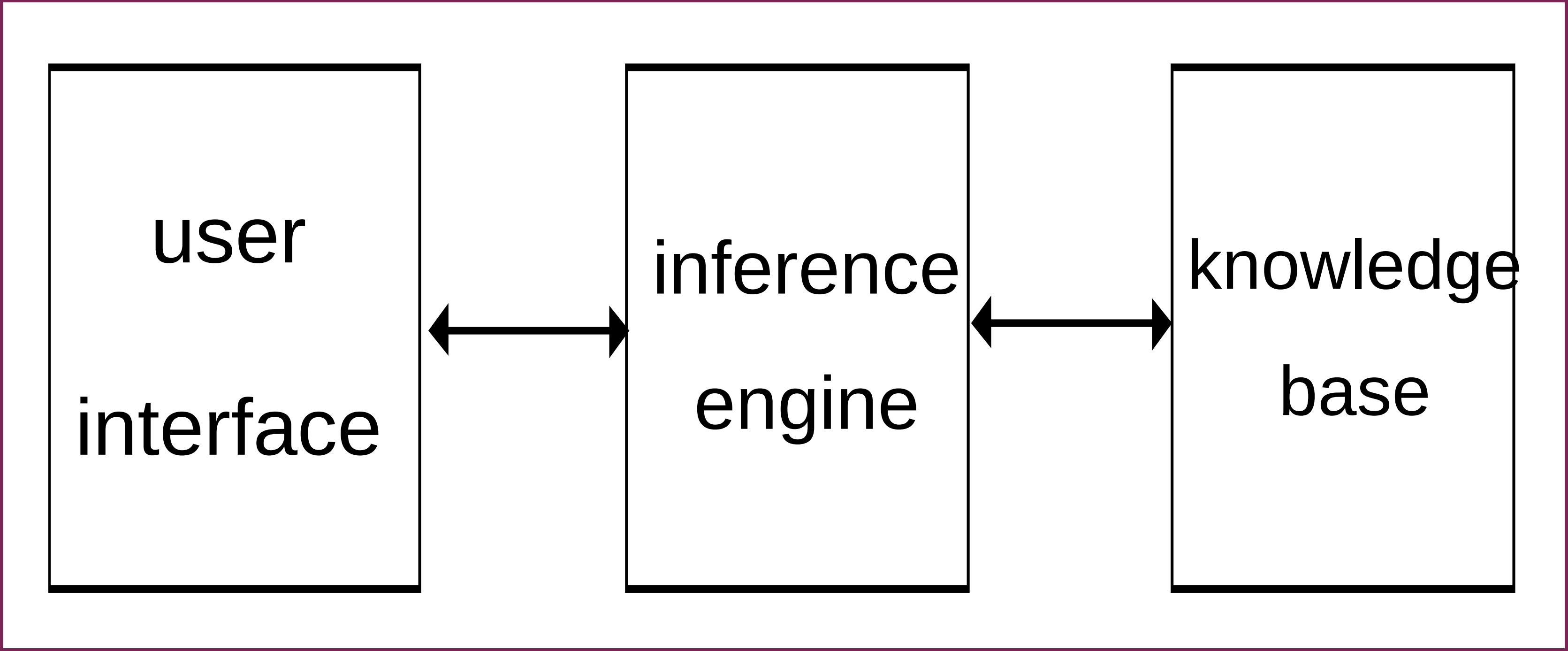
Section D

Rule-based Systems

KBS architecture

KBS architecture (1)

- The typical architecture of an KBS is often described as follows:



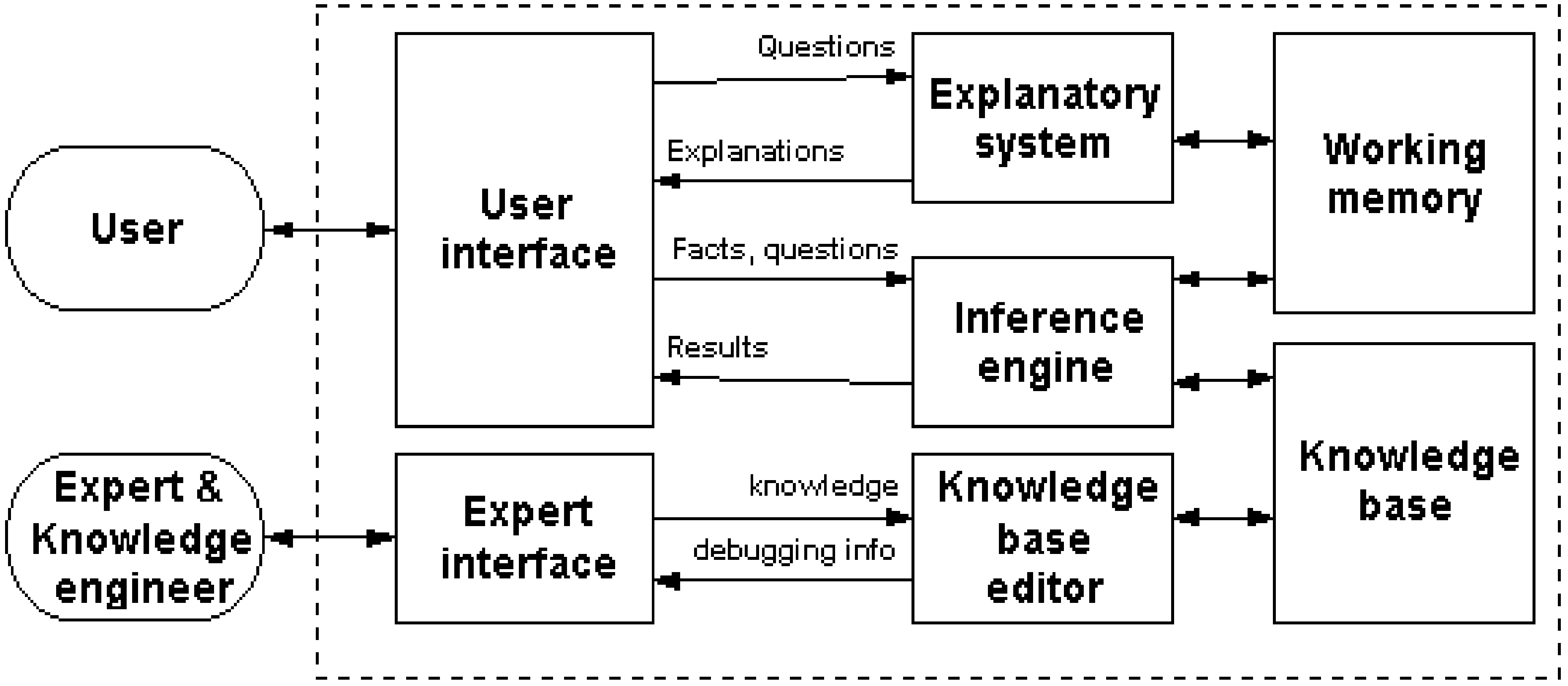
KBS architecture (1)

- The inference engine and knowledge base are separated because:
  - the reasoning mechanism needs to be as stable as possible;
  - the knowledge base must be able to grow and change, as knowledge is added;
  - this arrangement enables the system to be built from, or converted to, a shell.

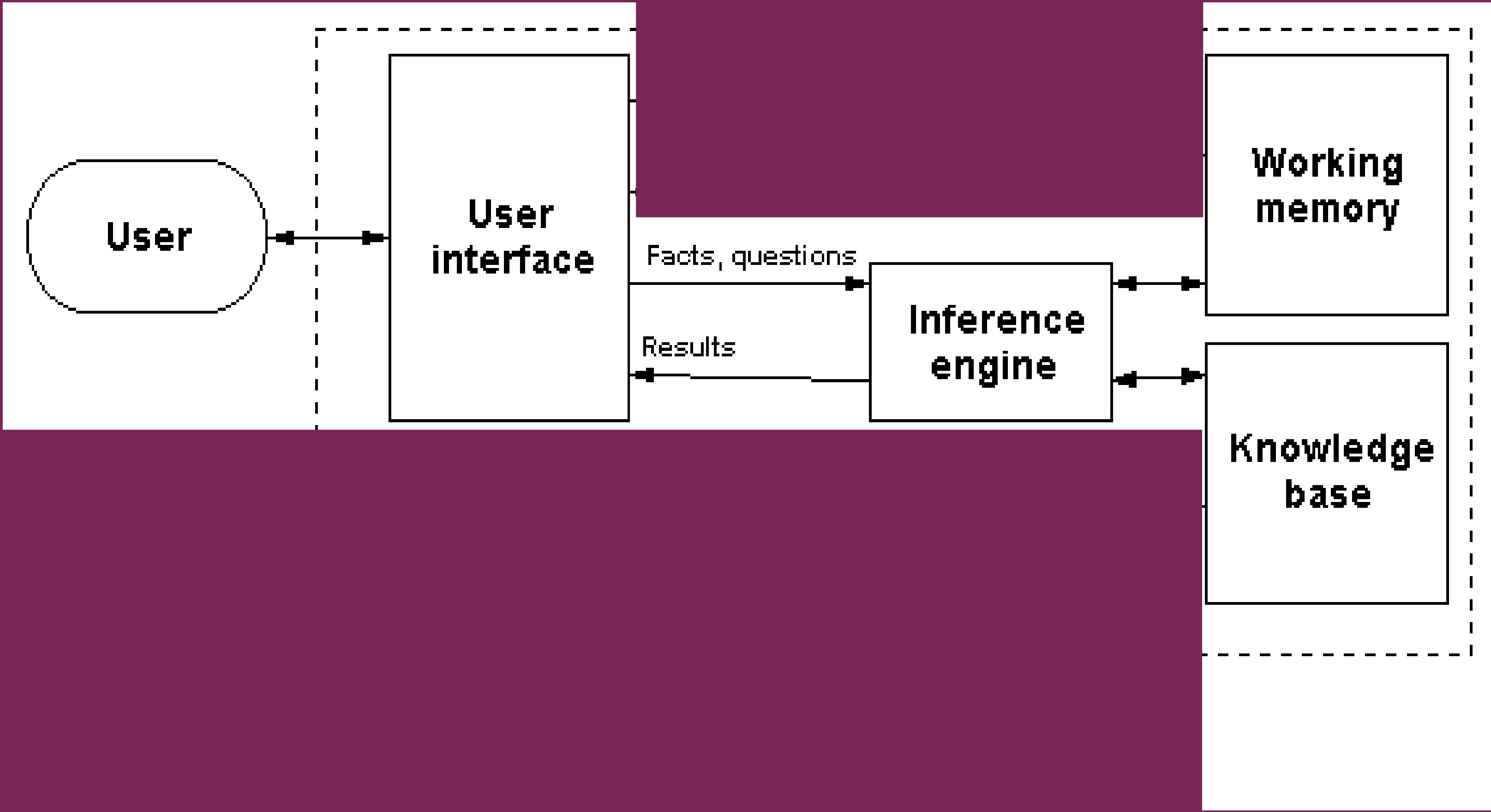
KBS architecture (2)

- It is reasonable to produce a richer, more elaborate, description of the typical KBS.
- A more elaborate description, which still includes the components that are to be found in almost any real-world system, would look like this:

KBS architecture (2)



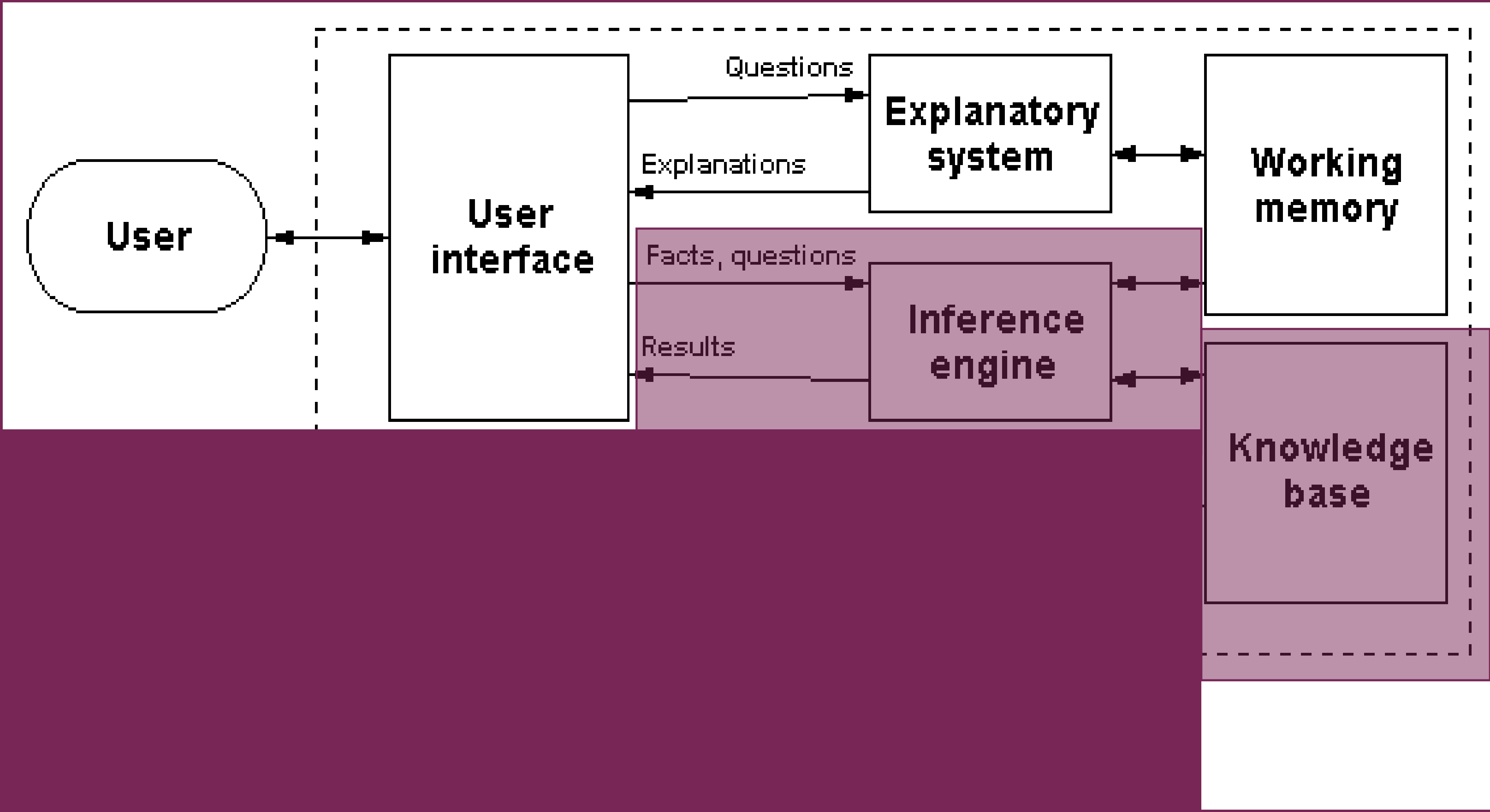
KBS architecture (2)



KBS architecture (2)

- The system holds a collection of *general principles* which can potentially be applied to any problem - these are stored in the *knowledge base*.
- The system also holds a collection of *specific details* that apply to the current problem (including details of how the current reasoning process is progressing) - these are held in *working memory*.
- Both these sorts of information are processed by the *inference engine*.

KBS architecture (2)



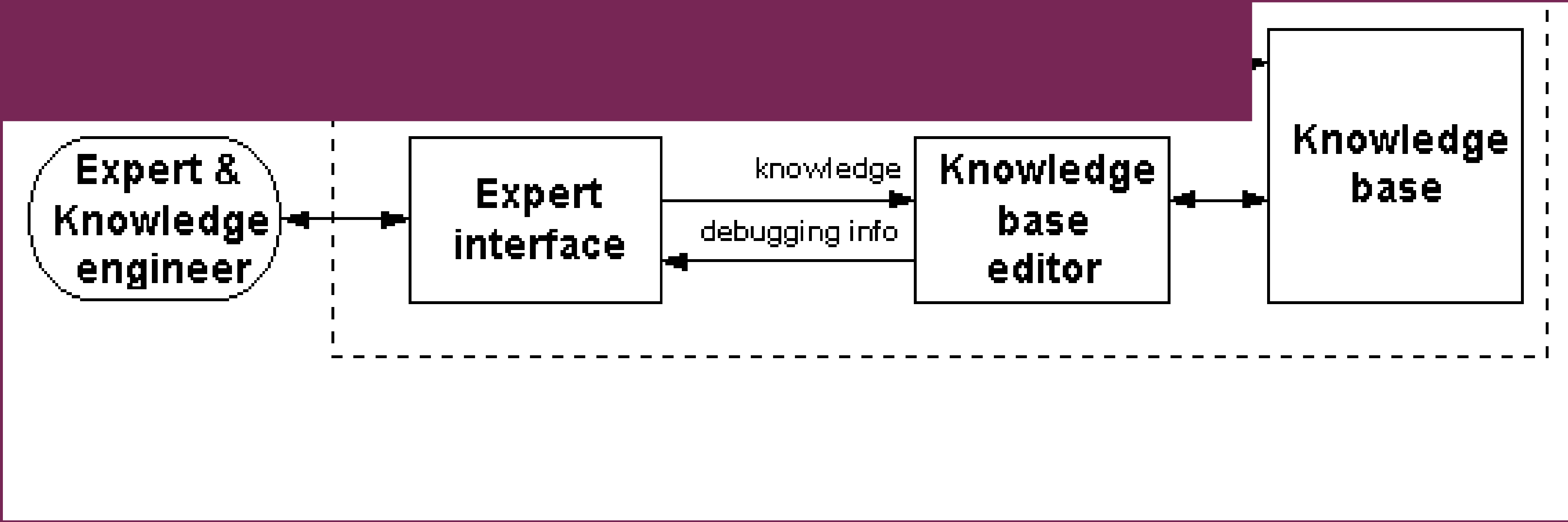
KBS architecture (2)

- Any practical expert system needs an explanatory facility. It is essential that an expert system should be able to explain its reasoning. This is because:

- it gives the user confidence in the system;

- it makes it easier to debug the system.

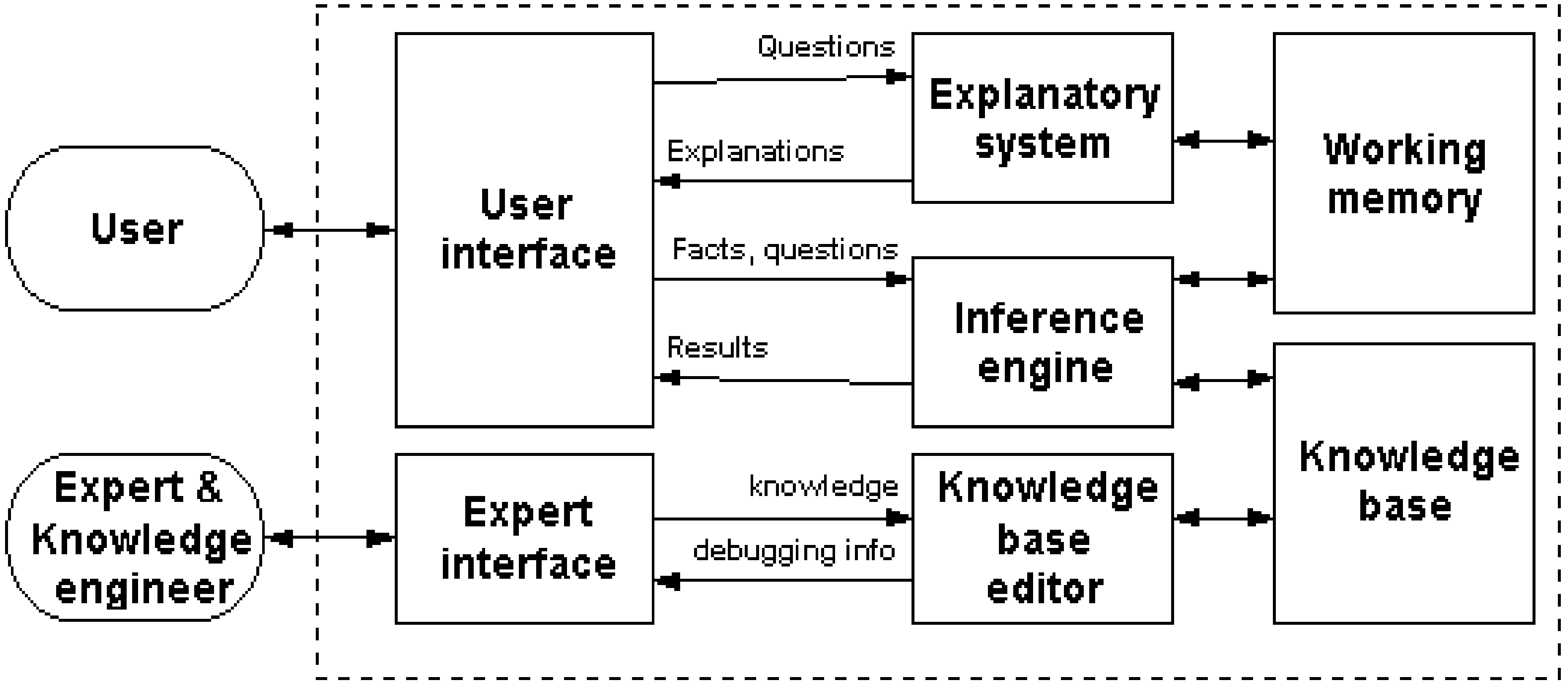
KBS architecture (2)



KBS architecture (2)

- It is not unreasonable to include an expert interface & a knowledge base editor, since any practical KBS is going to need a mechanism for efficiently building and modifying the knowledge base.

KBS architecture (2)



- As mentioned earlier, a reliable expert should be able to explain and justify his/her advice and actions.

Rule-based reasoning

Rule-based reasoning

- One can often represent the expertise that someone uses to do an expert task as rules.
- A rule means a structure which has an if component and a then component.
- This is actually a very old idea indeed -

The Edwin Smith papyrus

- The Edwin Smith papyrus is a 3700-year-old ancient Egyptian text.

ABCDEECDBBACDACDBCDECDADCADBADE

ECDBBACDACDBCDECDADCADBADCDBBACDA

BCDEECDBBACDACDBCDECDAD

BBACDACDBCDECDADCADBADEDCDBBA

DCDBBADCDBBABCDECDADCADBADEACDA

BACDACDBCDECDADBACDACDBCDECDAD

The Edwin Smith papyrus

- It contains medical descriptions of 48 different types of head wound.
- There is a fixed format for each problem description: Title - symptoms - diagnosis - prognosis - treatment.

The Edwin Smith papyrus

- There's a fixed style for the parts of each problem description. Thus, the prognosis always reads "It is an injury that I will cure", or "It is an injury that I will combat", or "It is an injury against which I am powerless".

- An example taken from the Edwin Smith papyrus:

The Edwin Smith papyrus

Title:

Instructions for treating a fracture of the cheekbone.

Symptoms:

If you examine a man with a fracture of the cheekbone, you will find a salient and red fluxion, bordering the wound.

The Edwin Smith papyrus

Diagnosis and prognosis:

Then you will tell your patient: "A fracture of the cheekbone. It is an injury that I will cure."

Treatment:

You shall tend him with fresh meat the first day. The treatment shall last until the fluxion resorbs. Next you shall treat him with raspberry, honey, and bandages to be renewed each day, until he is cured.

Rule-based reasoning: rules

- examples:

if - the leaves are dry, brittle and discoloured

then - the plant has been attacked by red spider mite

if - the customer closes the account

then - delete the customer from the database

Rule-based reasoning: rules

- The statement, or set of statements, after the word if represents some pattern which you may observe.

- The statement, or set of statements, after the word then represents some conclusion that you can draw, or some action that you should take.

Rule-based reasoning: rules

- A rule-based system, therefore, either
  - identifies a pattern and draws conclusions about what it means,

  or

  - identifies a pattern and advises what should be done about it,

  or

  - identifies a pattern and takes appropriate action.

Rule-based reasoning: rules

- The essence of a rule-based reasoning system is that it goes through a series of cycles.
- In each cycle, it attempts to pick an *appropriate* rule from its collection of rules, depending on the present circumstances, and to use it as described above.
- Because using a rule produces new information, it's possible for each new cycle to take the reasoning process further than the cycle before. This is rather like a human following a chain of ideas in order to come to a conclusion.

Terminology

- A rule as described above is often referred to as a *production rule*.
- A set of production rules, together with software that can reason with them, is known as a *production system*.

Terminology

- There are several different terms for the statements that come after the word if, and those that come after the word then.
  - The statements after if may be called the *conditions*, those after then may be called the *conclusions.*
  - The statements after if may be called the *premises*, those after then may be called the *actions.*
  - The statements after if may be called the *antecedents*, those after then may be called the *consequents.*

Terminology

- Some writers just talk about the *if-part* and the *then-part*.

Terminology

- If a production system chooses a particular rule, because the conditions match the current state of affairs, and puts the conclusions into effect, this is known as firing the rule.

Terminology

- In a production system, the rules are stored together, in an area called the rulebase.

Historical note

- Mathematicians, linguists, psychologists and artificial intelligence specialists explored the possibilities of production rules during the 40s, 50s and 60s.
- When the first expert systems were invented in the 70s, it seemed natural to use production rules as the knowledge representation formalism for the knowledge base.

Historical note

- Production rules have remained the most popular form of knowledge representation for expert systems ever since.

Conditional branching

- Is a production rule the same as a conditional branching statement?

- A production rule looks similar to the

if (statement to be evaluated) then (action)
pattern which is a familiar feature of all conventional programming languages.

Conditional branching

- e.g. The following fragment from a C program:

Conditional branching

```
{   int magic;
    int guess;
    magic = rand( );
    printf("guess the magic number: ");
    scanf("%d", &guess);
    if (guess == magic)  printf("** Right **");
    else  {
          printf("Wrong, ");
          if (guess > magic) printf("too high");
          else printf("too low");
    }
}
```

Conditional branching vs. production rules

- However, the similarity is misleading. There is a radical difference between a production system and a piece of conventional software.
  - In a conventional program, the if...then... structure is an integral part of the code, and represents a point where the execution can branch in one of two (or more) directions.

Conditional branching vs. production rules

- In a production system, the if...then... rules are gathered together in a rule base, and the controlling part of the system has some way of choosing a rule from this knowledge base which is appropriate to the current circumstances, and then using it.

Reasoning with production rules

- The statements forming the conditions, or the conclusions, in such rules, may be structures, following some syntactic convention (such as three items enclosed in brackets).

Reasoning with production rules

- Very often, these structures will include variables - such variables can, of course, be given a particular value, and variables with the same name in the same rule will share the same value.

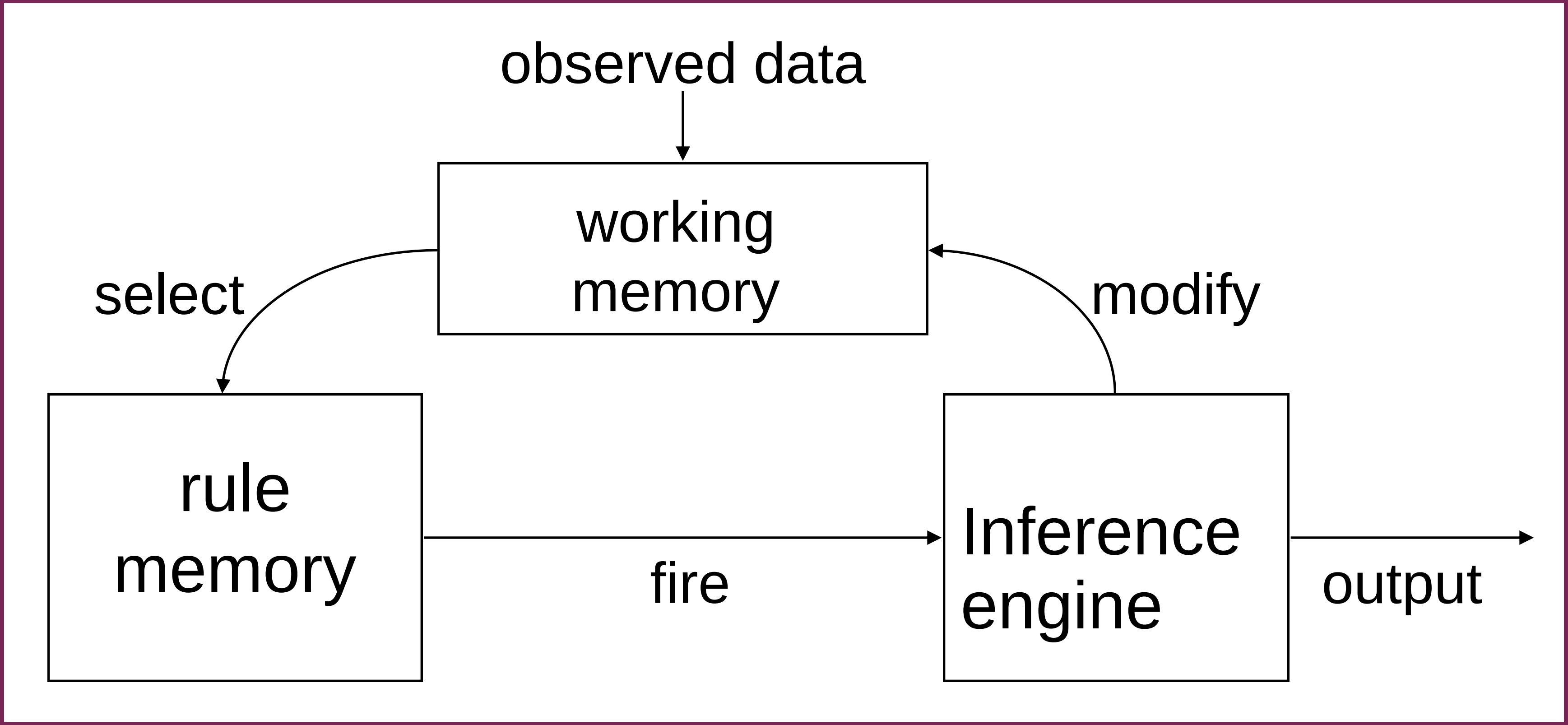
Reasoning with production rules

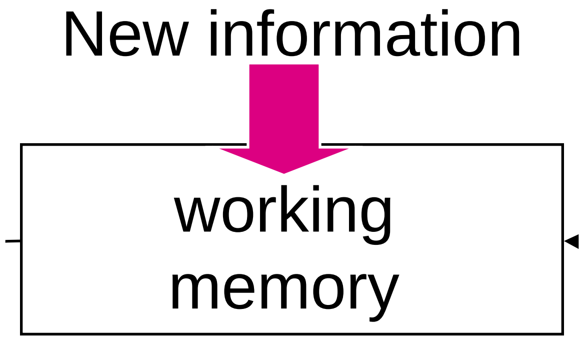
- For example (assuming words beginning with capital letters are variables, and other words are constants):

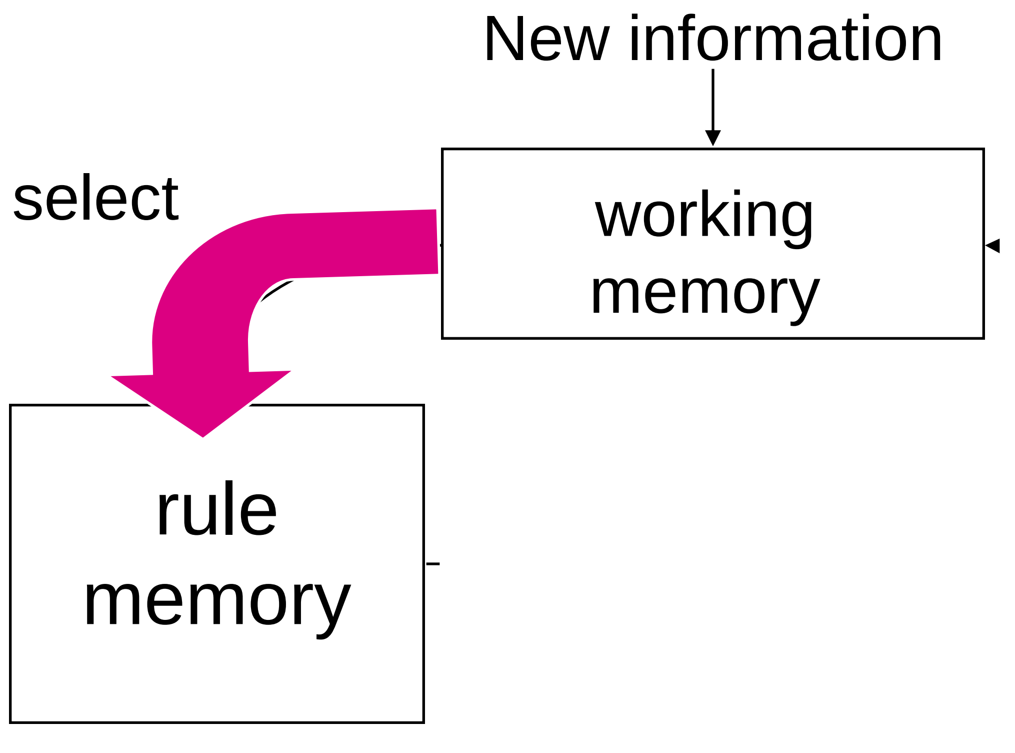
```
if      [Person, age, Number] &
        [Person, employment, none] &
        [Number, greater_than, 18] &
        [Number, less_than, 65]
then [Person, can_claim,
                unemployment_benefit].
```

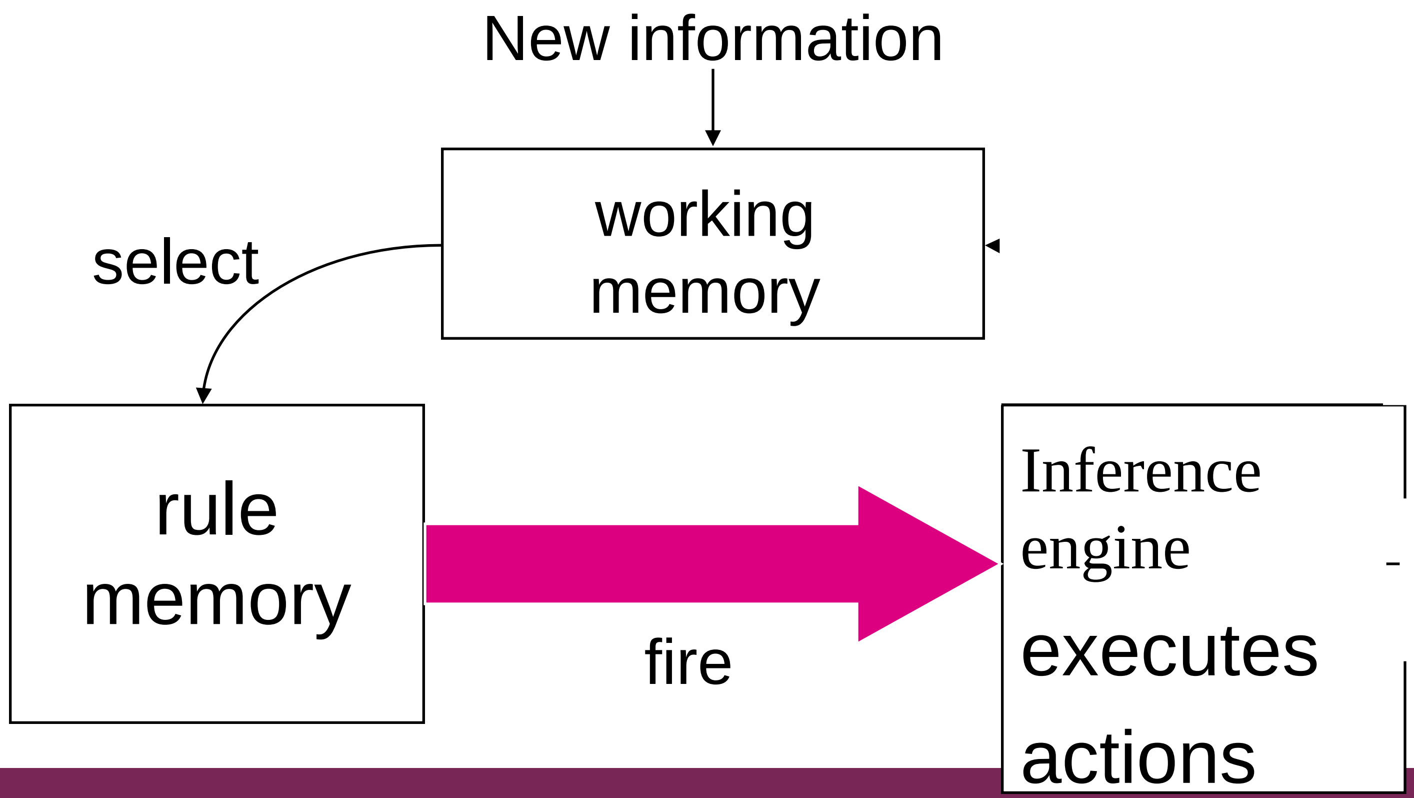
Reasoning with production rules

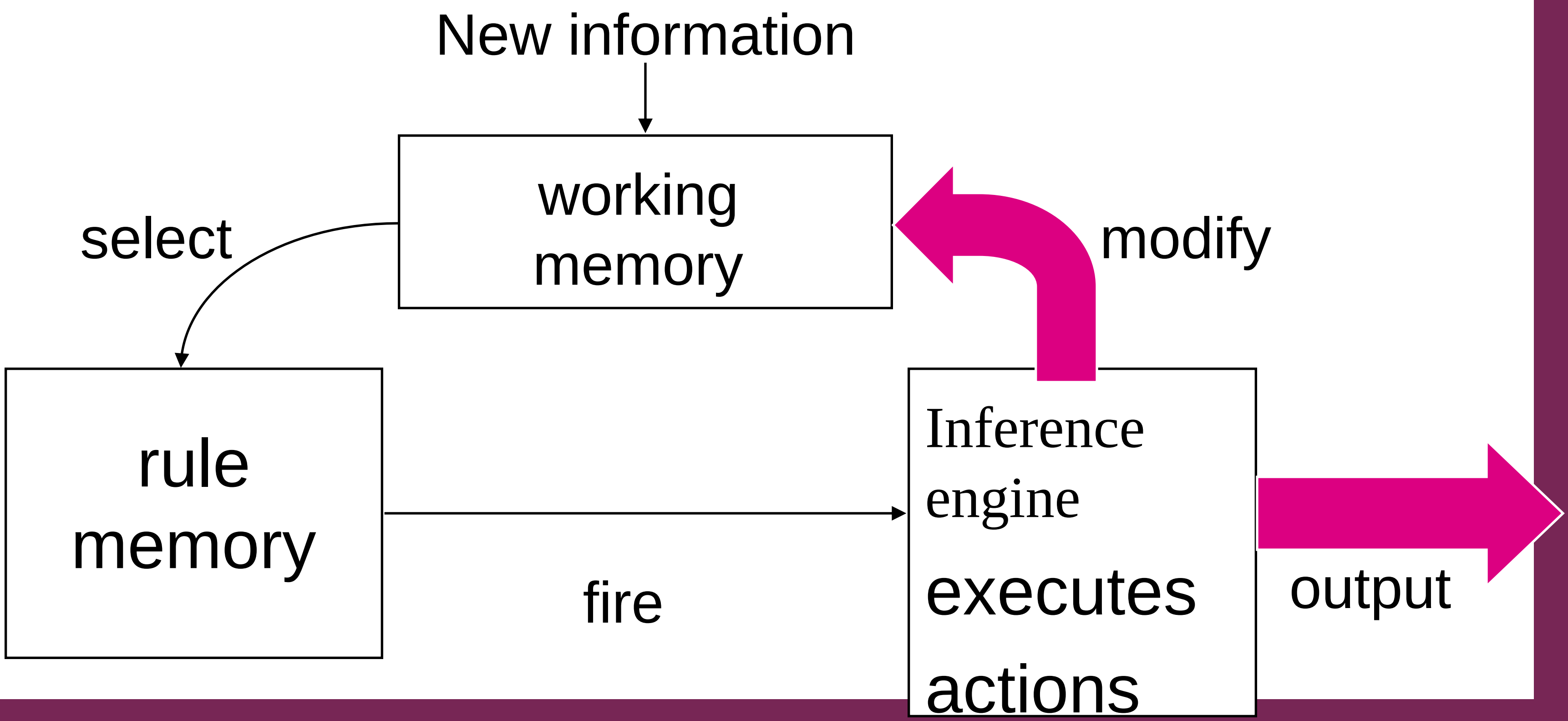
- Architecture of a typical production system:

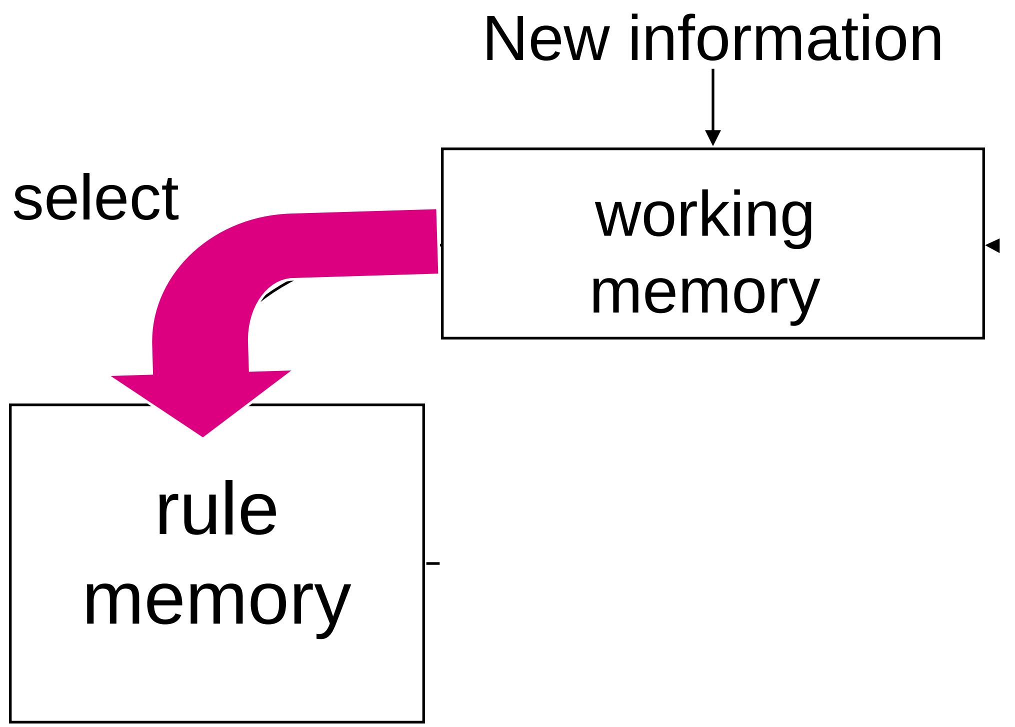


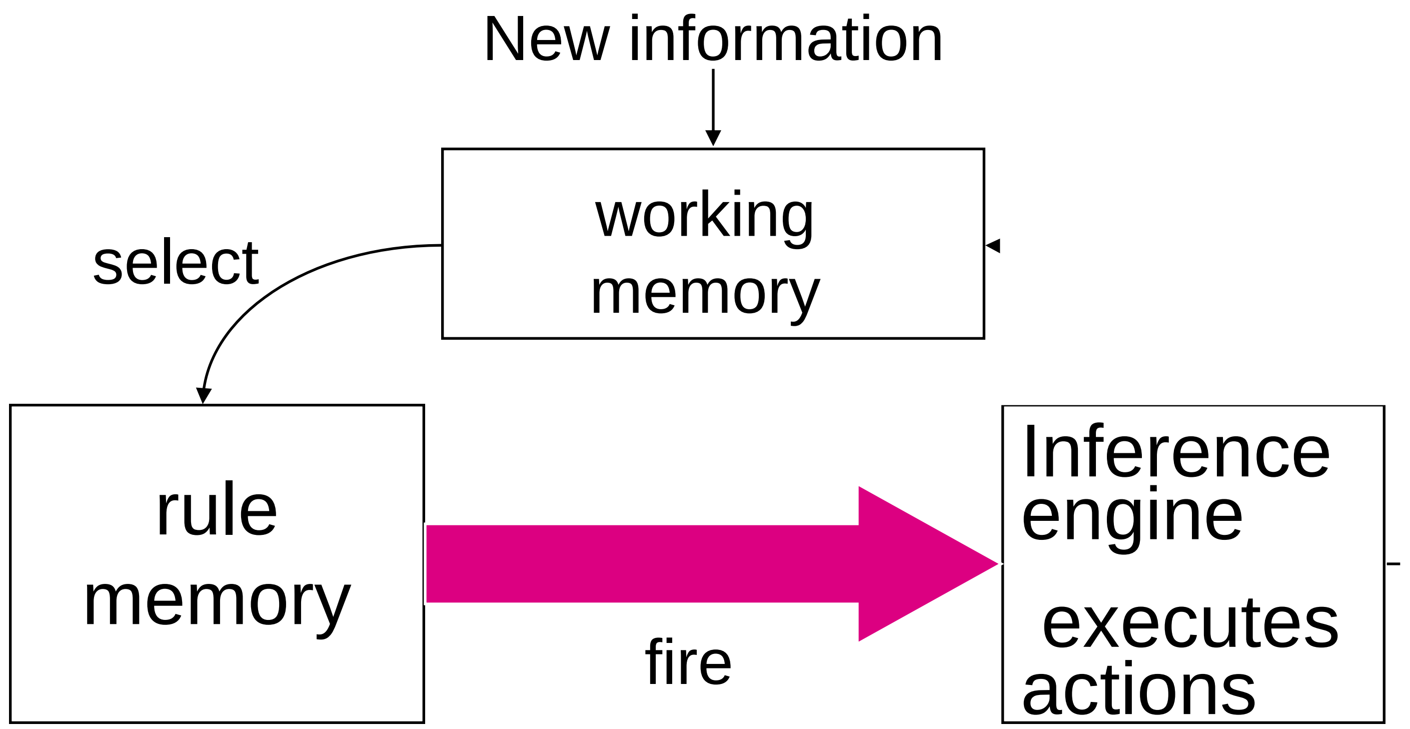

New information
working memory

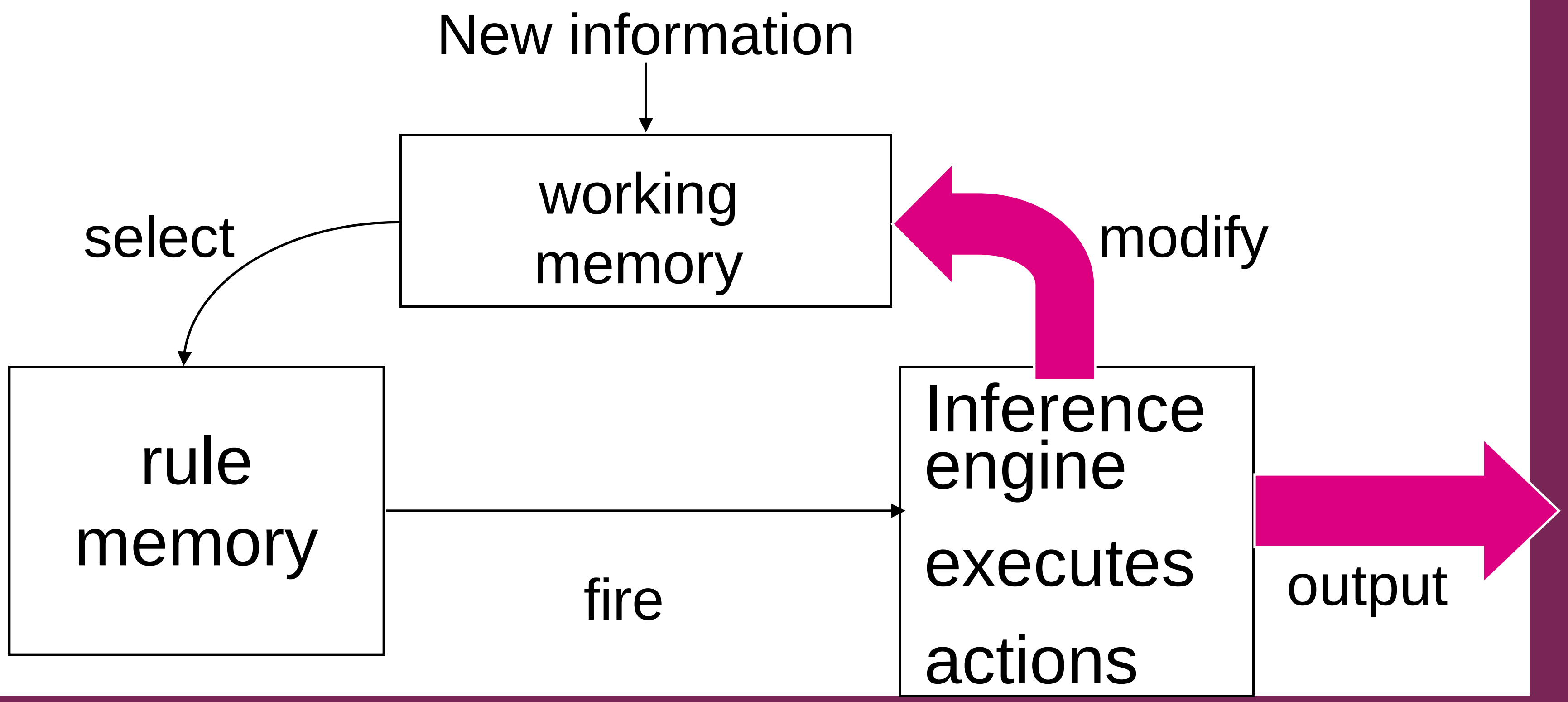

New information
select
working memory
rule memory


New information
working memory
select
rule memory
fire
Inference engine executes actions


New information
working memory
select
modify
rule memory
fire
Inference engine
executes actions
output


New information
select
working memory
rule memory


New information
working memory
select
rule memory
fire
Inference engine
executes actions


New information
working memory
select
modify
rule memory
fire
Inference engine executes actions
output

Architecture of a typical production system

- Has a working memory.
  - Holds items of data. Their presence, or their absence, causes the inference engine to trigger certain rules.
  - e.g. W.M. contains [john, age, 29] & [john, employment, none]
  - The system decides: does this match any rules in the rulebase? If so, choose the rule.

Architecture of a typical production system

- has an inference engine. Behaviour of the inference engine :
    - the system is started by putting a suitable data item into working memory.
    - *recognise-act cycle*: when data in the working memory matches the conditions of one of the rules in the system, the rule **fires** (i.e.is brought into action).

Advantages of production systems ... at first glance

- The principle advantage of production rules is notational convenience - it’s easy to express suitable pieces of knowledge in this way.

- The principle disadvantage of production rules is their restricted power of expression - many useful pieces of knowledge don’t fit this pattern.

Advantages of production systems ... at first glance

- This would seem to be a purely declarative form of knowledge representation. One gathers pieces of knowledge about a particular subject, and puts them into a rulebase. One doesn't bother about when or how or in which sequence the rules are used; the production system can deal with that.
- When one wishes to expand the knowledge, one just adds more rules at the end of the rulebase.

Advantages of production systems ... at first glance

- The rules themselves are very easy to understand, and for someone (who is expert in the specific subject the system is concerned with) to criticise and improve.

Advantages of production systems ... at first glance

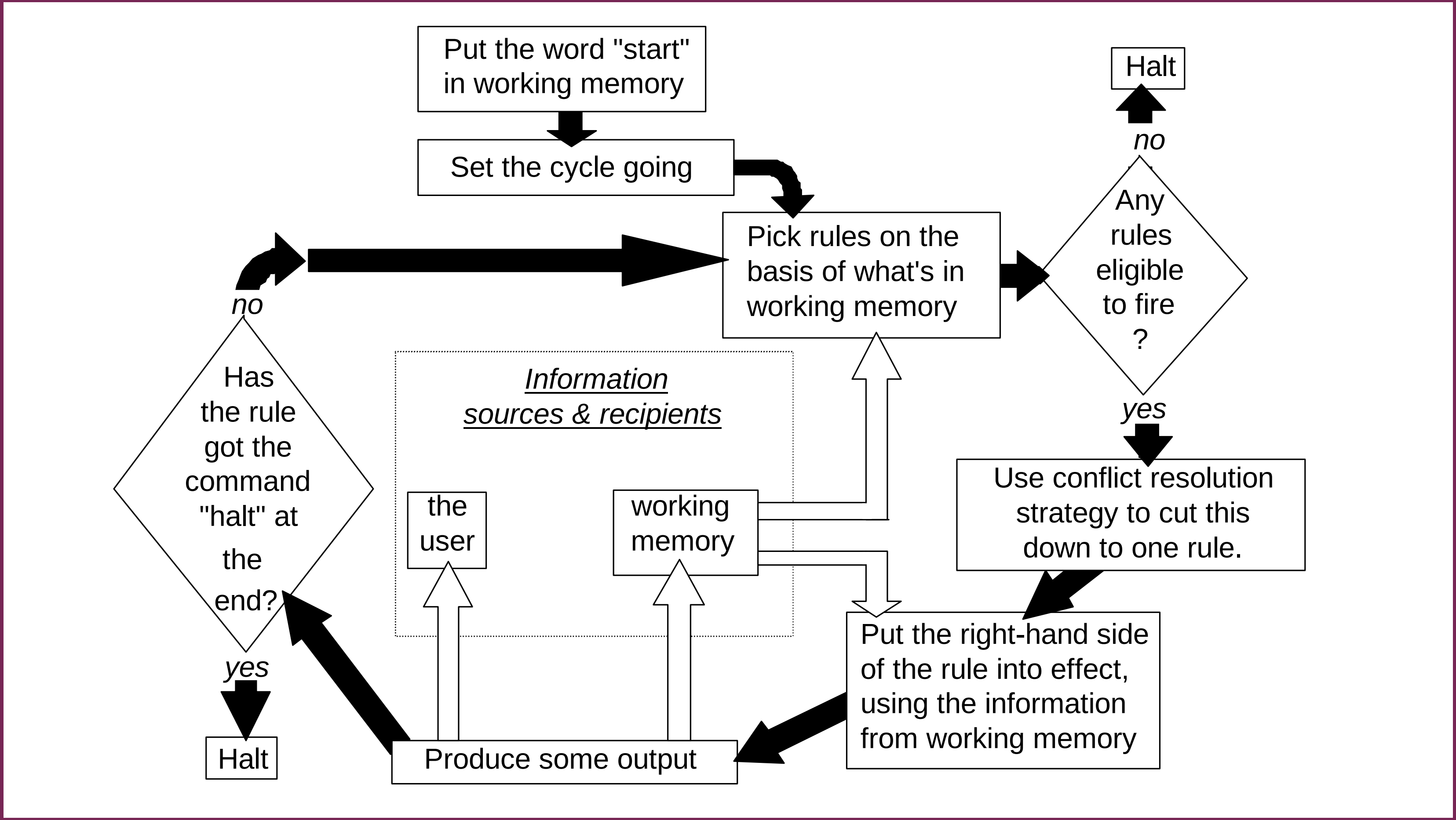
- It's fairly straightforward to implement a production system interpreter. Following the development of the Rete Matching Algorithm, and other improvements, quite efficient interpreters are now available.

Advantages of production systems ... at first glance

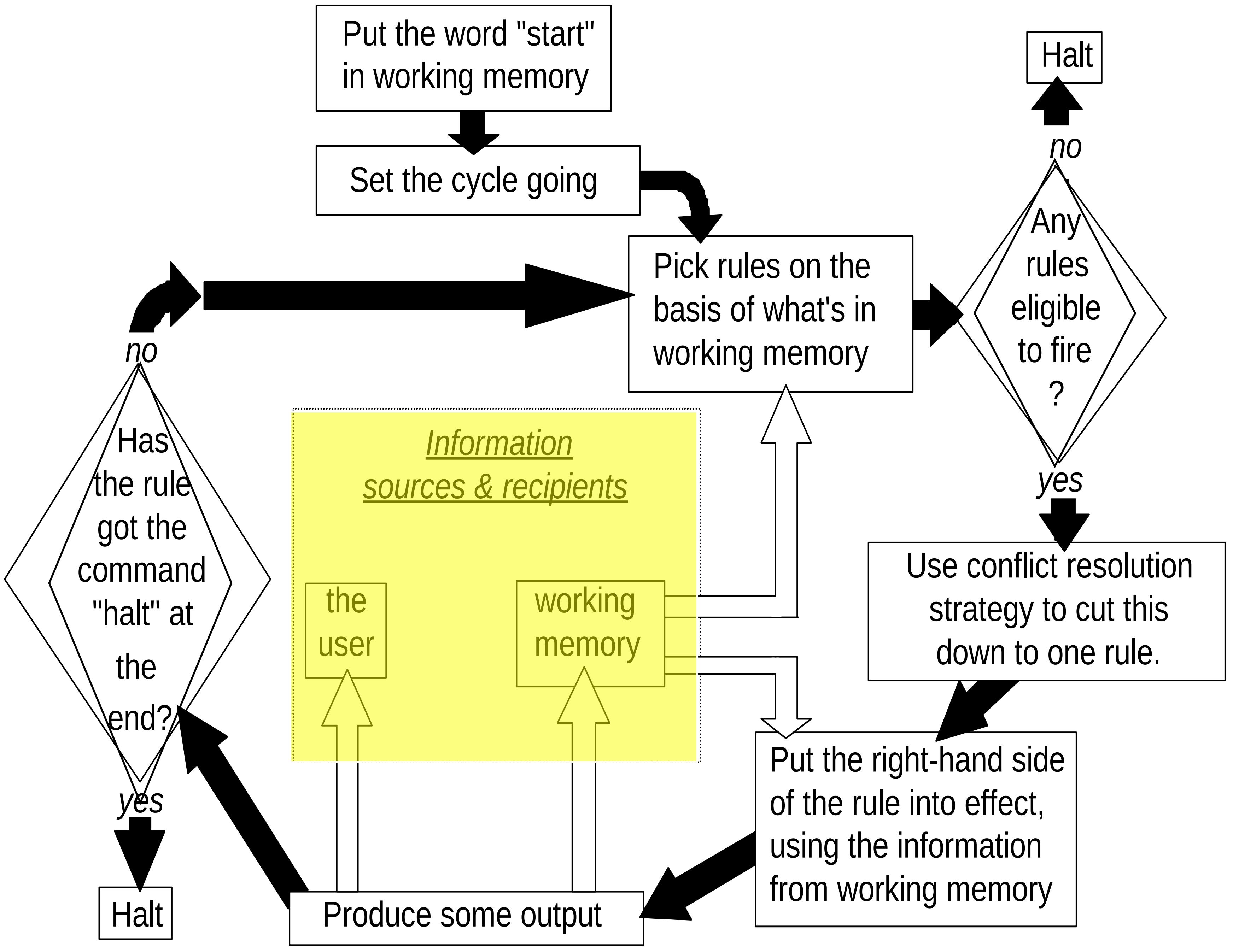
- However, it isn't that simple. See "advantages reconsidered" later on.

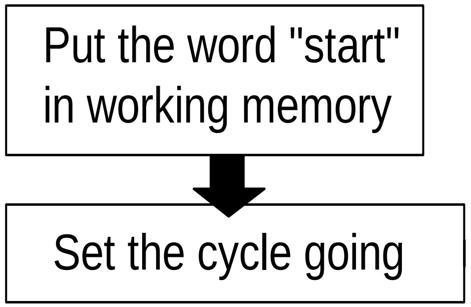
Operation of a production system in more detail

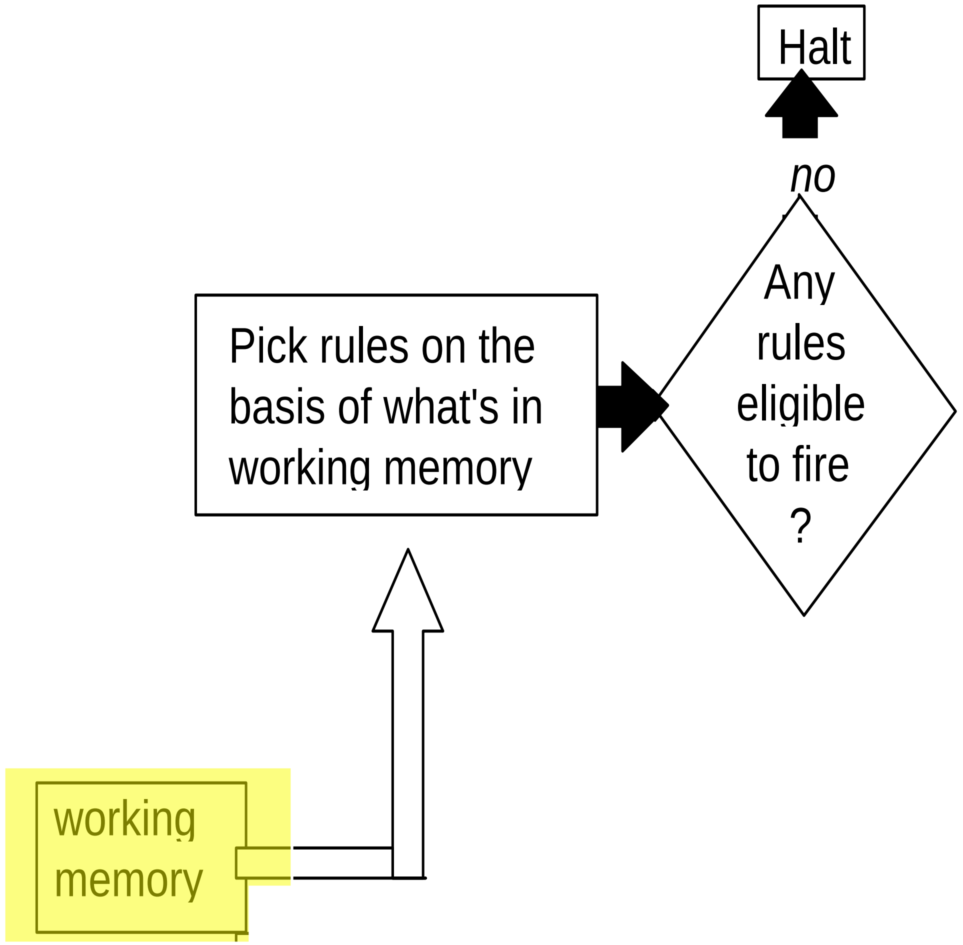
- The recognise-act cycle (forward-chaining):

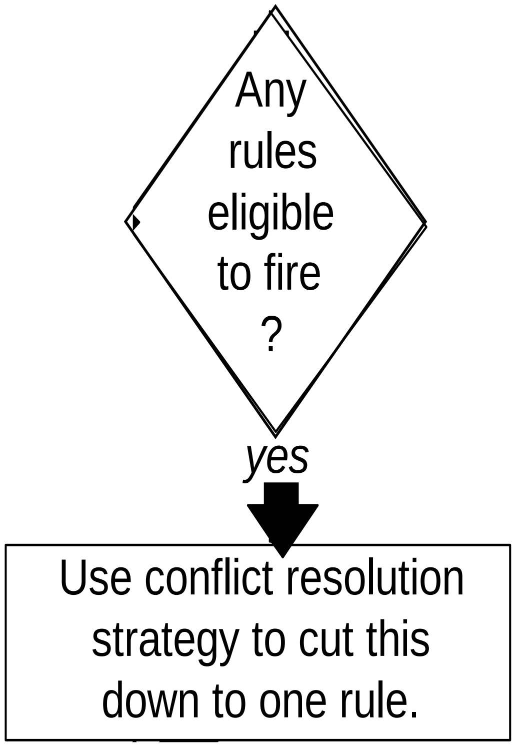


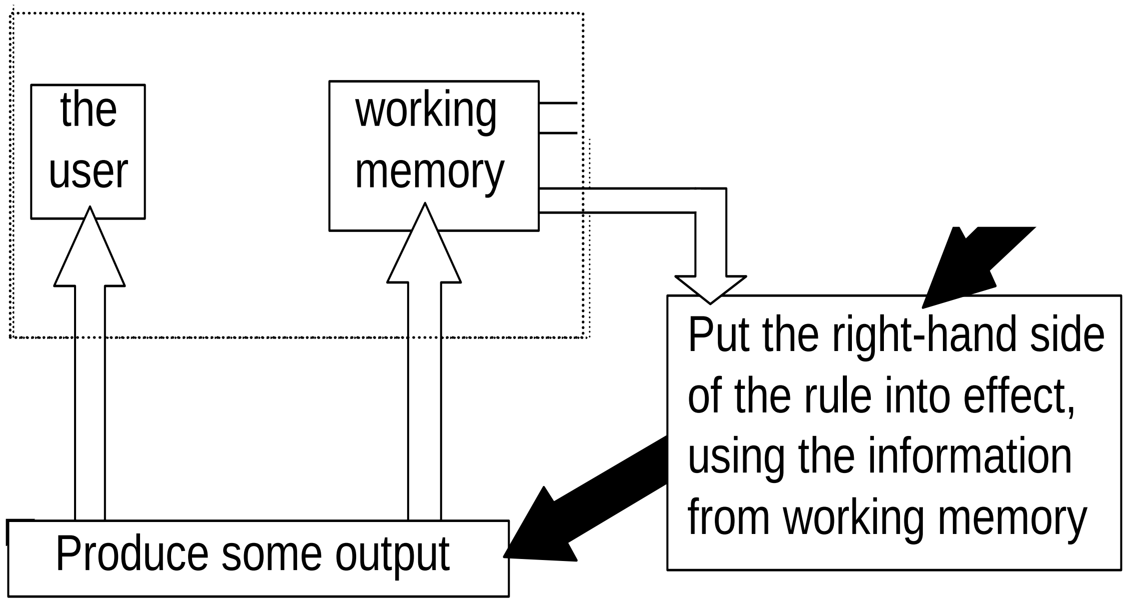
Put the word "start" in working memory
Set the cycle going
Pick rules on the basis of what's in working memory
Any rules eligible to fire ?
no
Halt
yes
Use conflict resolution strategy to cut this down to one rule.
Put the right-hand side of the rule into effect, using the information from working memory
Produce some output
Has the rule got the command "halt" at the end?
yes
Halt
no
Information sources & recipients
the user
working memory

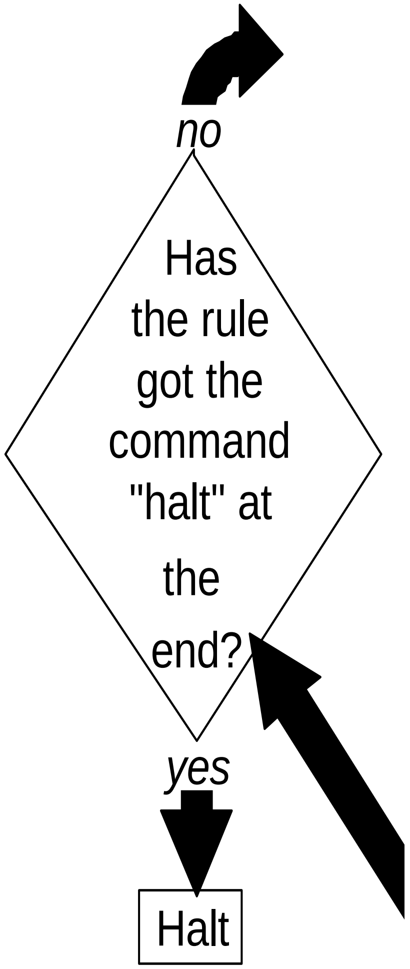

Put the word "start" in working memory
Set the cycle going
Pick rules on the basis of what's in working memory
Any rules eligible to fire ?
no
Halt
yes
Use conflict resolution strategy to cut this down to one rule.
Put the right-hand side of the rule into effect, using the information from working memory
Produce some output
Has the rule got the command "halt" at the end?
yes
Halt
no
Information sources & recipients
the user
working memory

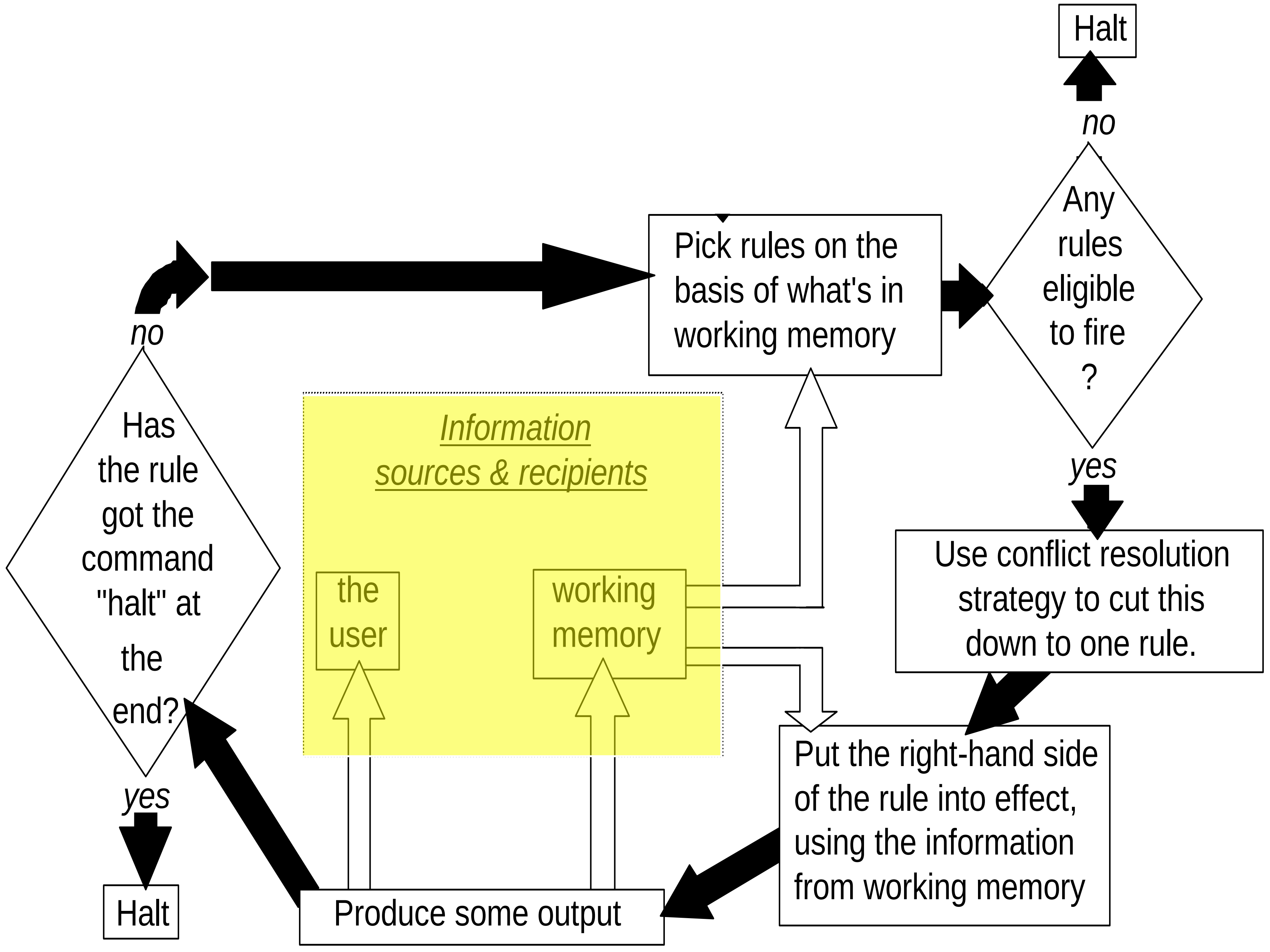

Put the word "start" in working memory
Set the cycle going


Halt
no
Any rules eligible to fire ?
Pick rules on the basis of what's in working memory
working memory


Any rules eligible to fire ?
yes
Use conflict resolution strategy to cut this down to one rule.


the user
working memory
Put the right-hand side of the rule into effect, using the information from working memory
Produce some output


no
Has the rule got the command "halt" at the end?
yes
Halt


Halt
no
Any rules eligible to fire ?
Pick rules on the basis of what's in working memory
no
Has the rule got the command "halt" at the end?
Information sources & recipients
the user
working memory
yes
Use conflict resolution strategy to cut this down to one rule.
Put the right-hand side of the rule into effect, using the information from working memory
yes
Halt
Produce some output

The recognise-act cycle

- N.B. "right-hand side of the rule" means the part after the word then.

The recognise-act cycle

- conflict resolution strategy: if **more than one** rule matches working memory contents, this decides which one is to fire. Alternatively, the rule base could be designed so there's never any conflict (but usually isn't).

The recognise-act cycle

- Applying the rule will probably modify the contents of working memory. Then the system continues with the recognise-act cycle.
- The system stops when
  - the rules stop firing, or
  - a rule fires which specifically tells the system to halt.

Conflict resolution strategies

- Choice of c.r.s. can make a big difference to system performance.
- Three favourite strategies:
  - Refractoriness: don't allow a rule to fire twice on same data.
  - Recency: take the data which arrived in working memory most recently, and find a rule that uses this data.
  - Specificity: use the most specific rule (the one with the most conditions attached).

Conflict resolution strategies

- However, in recent years the fashion (in expert system shells) has been for very simple CRSs, coupled with a reluctance to mention the problem to the potential system builder.
- Simple strategies:
  - Give each rule a priority number. If a choice has to be made, choose the rule with the highest number.
  - If a choice has to be made, choose the rule that comes first in the rule base.

Advantages of production systems reconsidered.

- Because of the effect of conflict resolution strategies, rules interact and the order of rules matters.
  - One must go beyond the declarative meaning of the rules and consider when (under which circumstances) they will fire.
  - One cannot properly understand a rule simply by reading it in isolation; one must consider the related rules, the meta-rules, and the conflict resolution strategy as well.

Advantages of production systems reconsidered.

- For the same reason, attempting to expand a production system by simply adding more rules at the end is dangerous.
  - Unexpected rule interactions are liable to happen.
  - The need to consider all these possible rule interactions makes large rule-based systems unwieldy and hard to update.

Advantages of production systems reconsidered.

- Although non-computer-specialists find it easy to grasp the meaning of individual rules, they don't find it easy to grasp these issues concerned with interactions.

Advantages of production systems reconsidered.

- Although efficient rule interpreters are available, one may still need to engage in meta-level programming in order to achieve a production system that shows acceptable performance on a large rulebase.