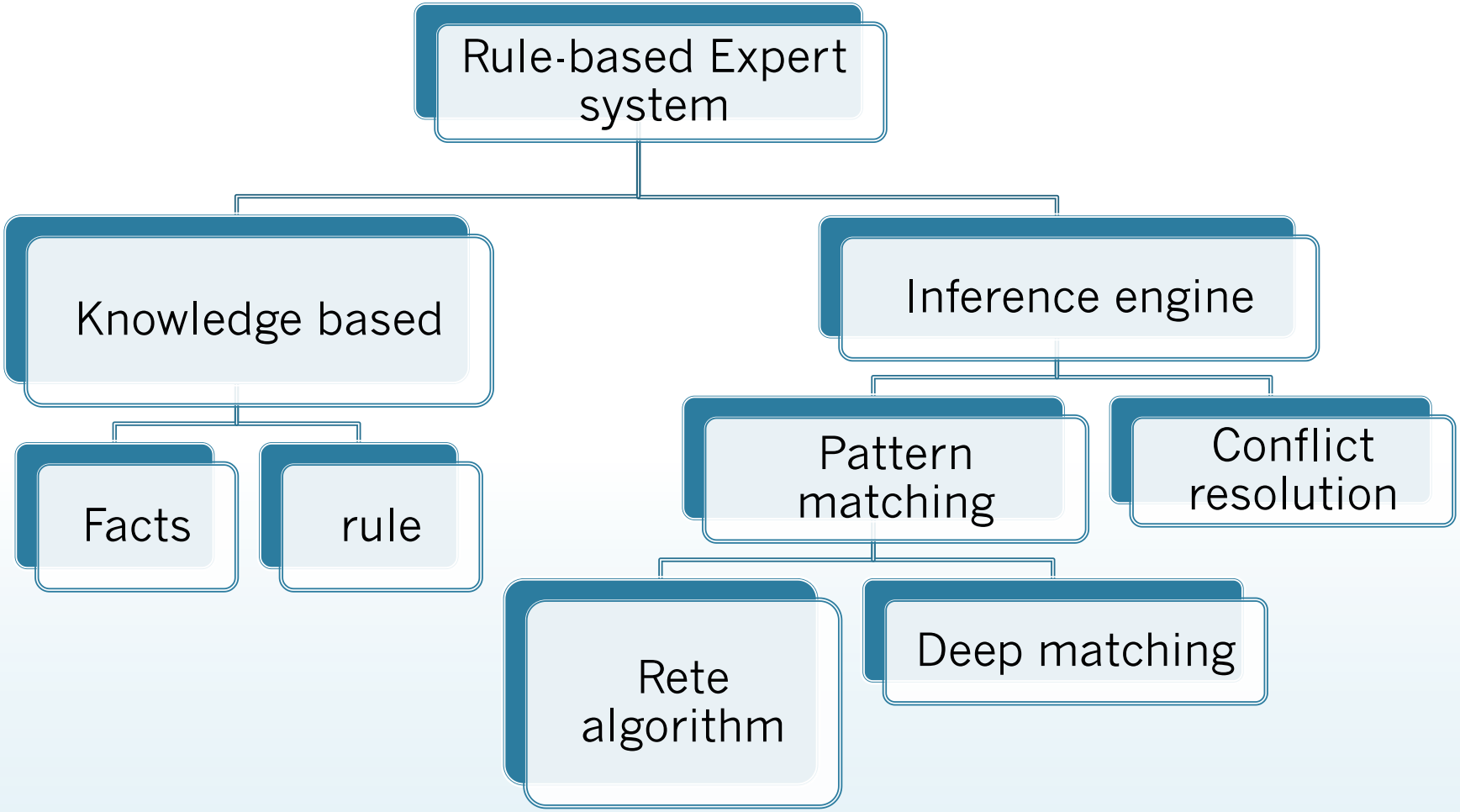
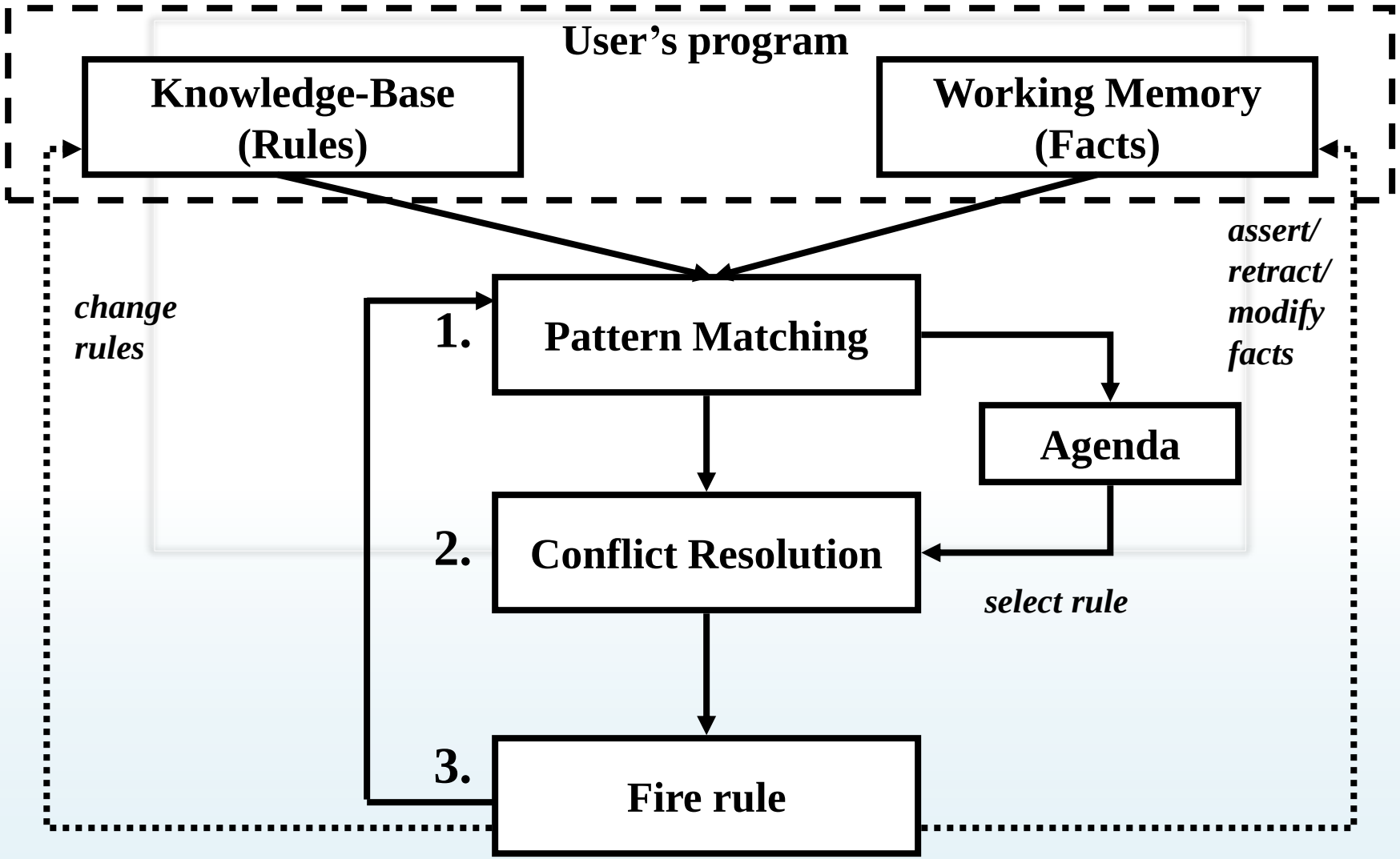
Foundations of Expert Systems



Rule-Based Reasoning

- Proceeds through matching, of either the antecedents or the consequents of rules, with the current situation
- Applying a rule changes the current situation so reasoning builds up chains of dependencies between reasoning steps
- There are two main styles
  - Forward chaining
  - Backward chaining

A Production System Cycle



Inference engine

- RBS Inference Cycle
  - Match ⇒ Conflict Set (CS)
  - Select
  - Act (Execute)
- RBS strategy
  - Rule chaining direction: forward, backward
  - Rule selection from CS
- Computational effort
  - Match – 80-90%

Fire the rule which was found…

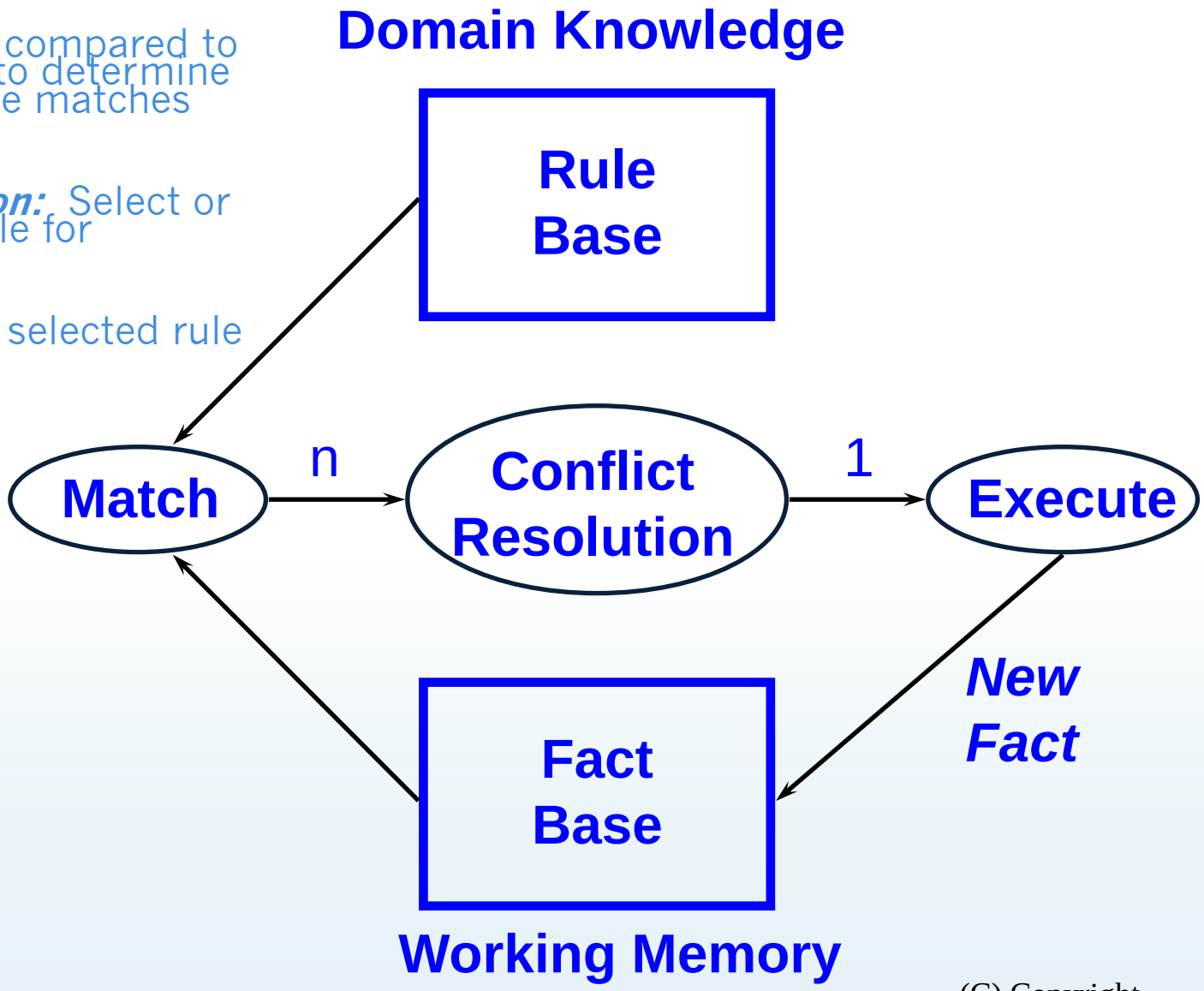
- May locate several rules
- Recognize-Act cycle:
  1. Scan the rules looking for ones whose premises match the contents of the working memory.
  1'. Choose one rule to fire.
  2. Fire the rule which was found.
  3. Place its conclusion in the working memory.
  4. Until no additional rule fires, go to 1.

Recognize-Act Cycle

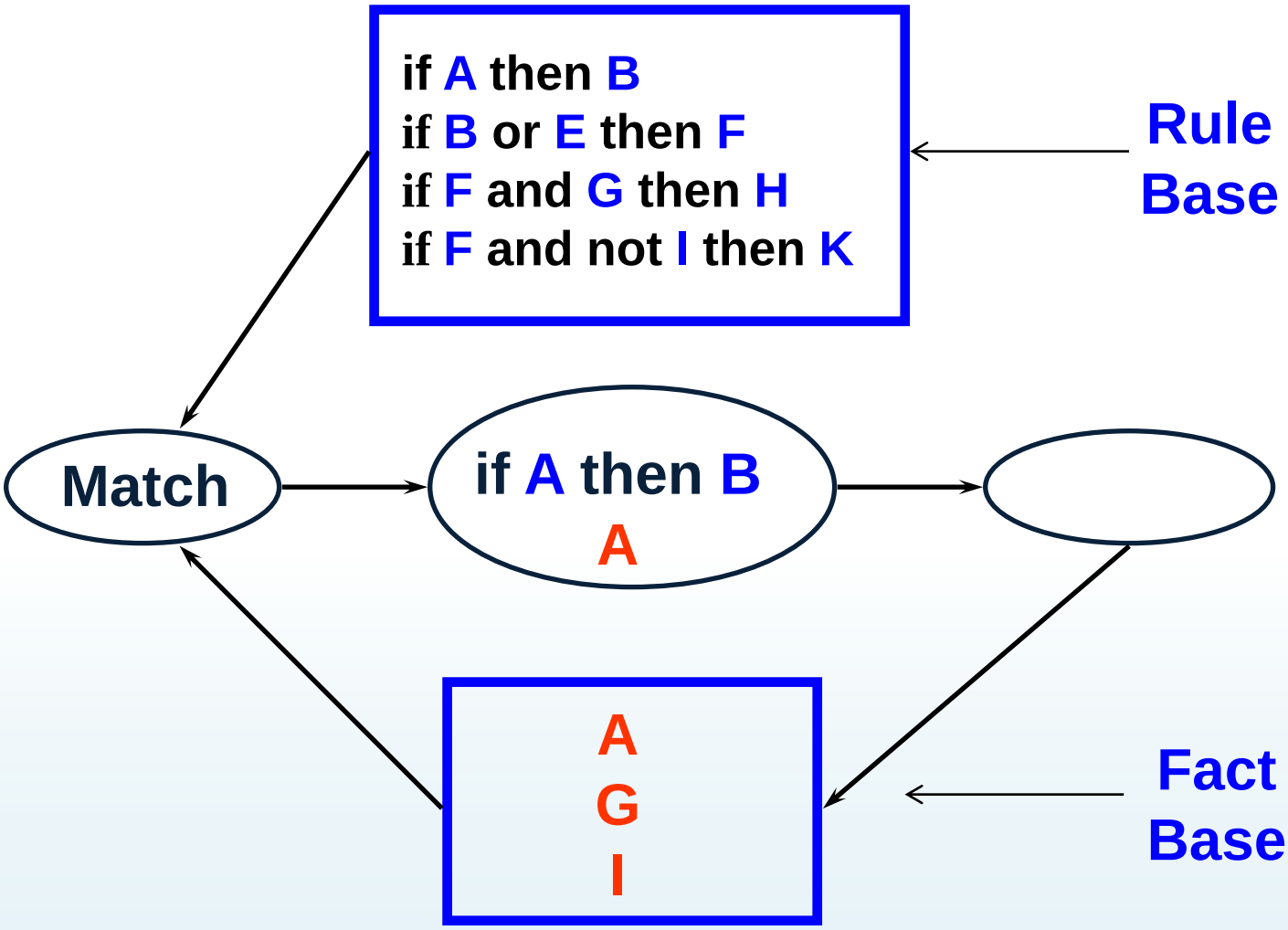
loop

1. ***Match:*** Rules are compared to working memory to determine matches. if no rule matches then *stop*
2. ***Conflict Resolution:*** Select or enable a single rule for execution
3. ***Execute:*** Fire the selected rule
   - Add new fact

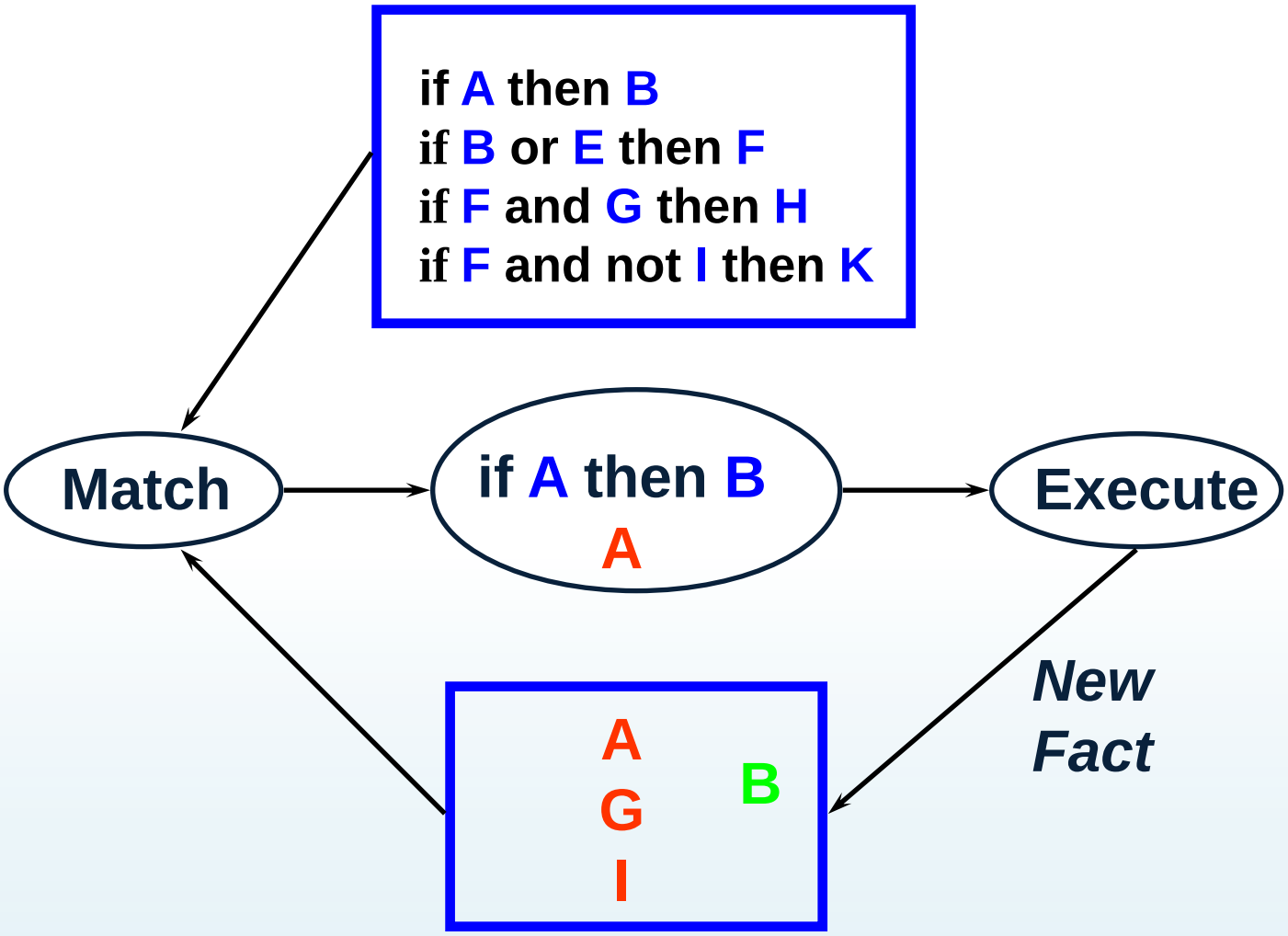
end loop



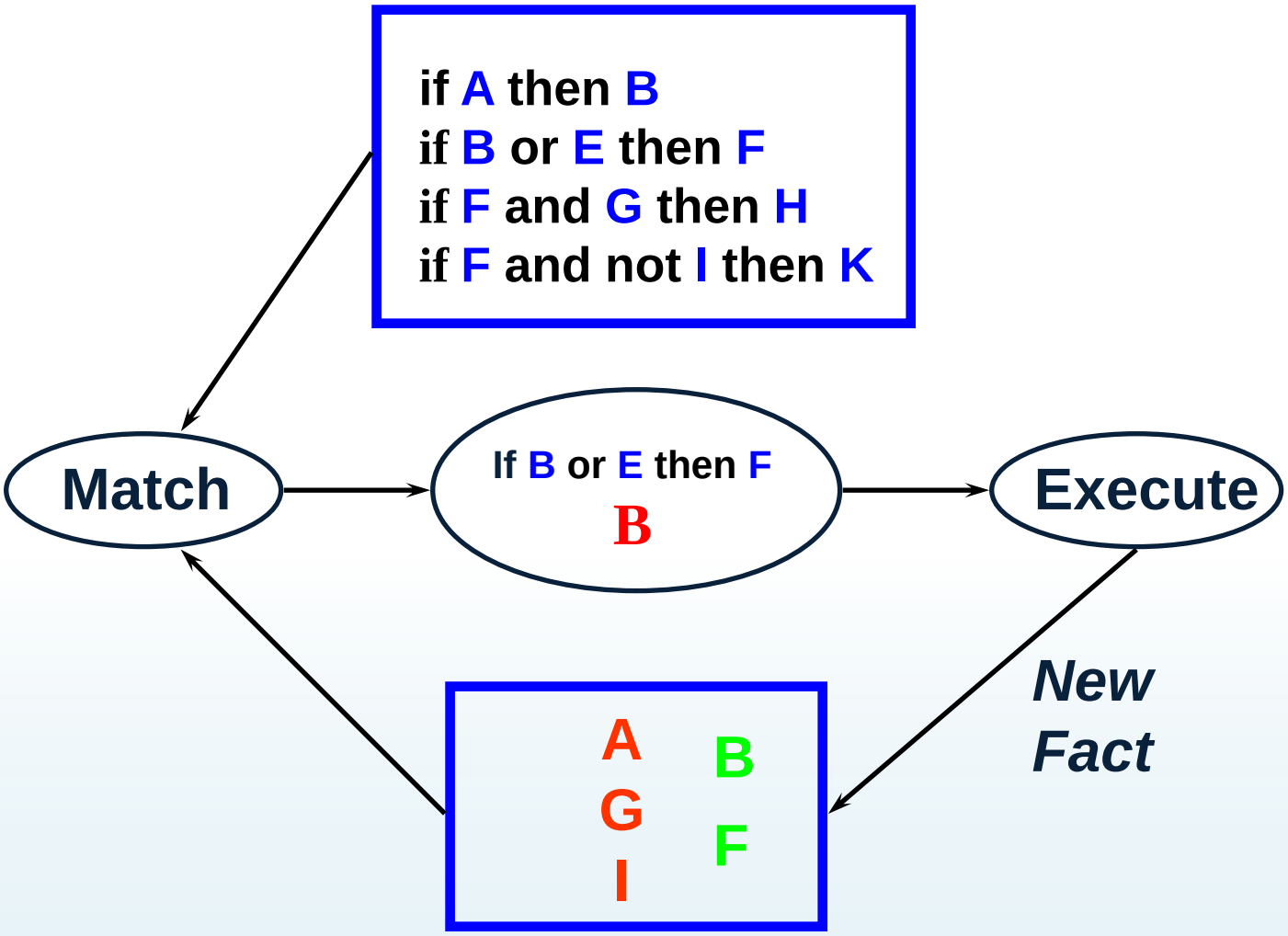
Recognize-Act Cycle



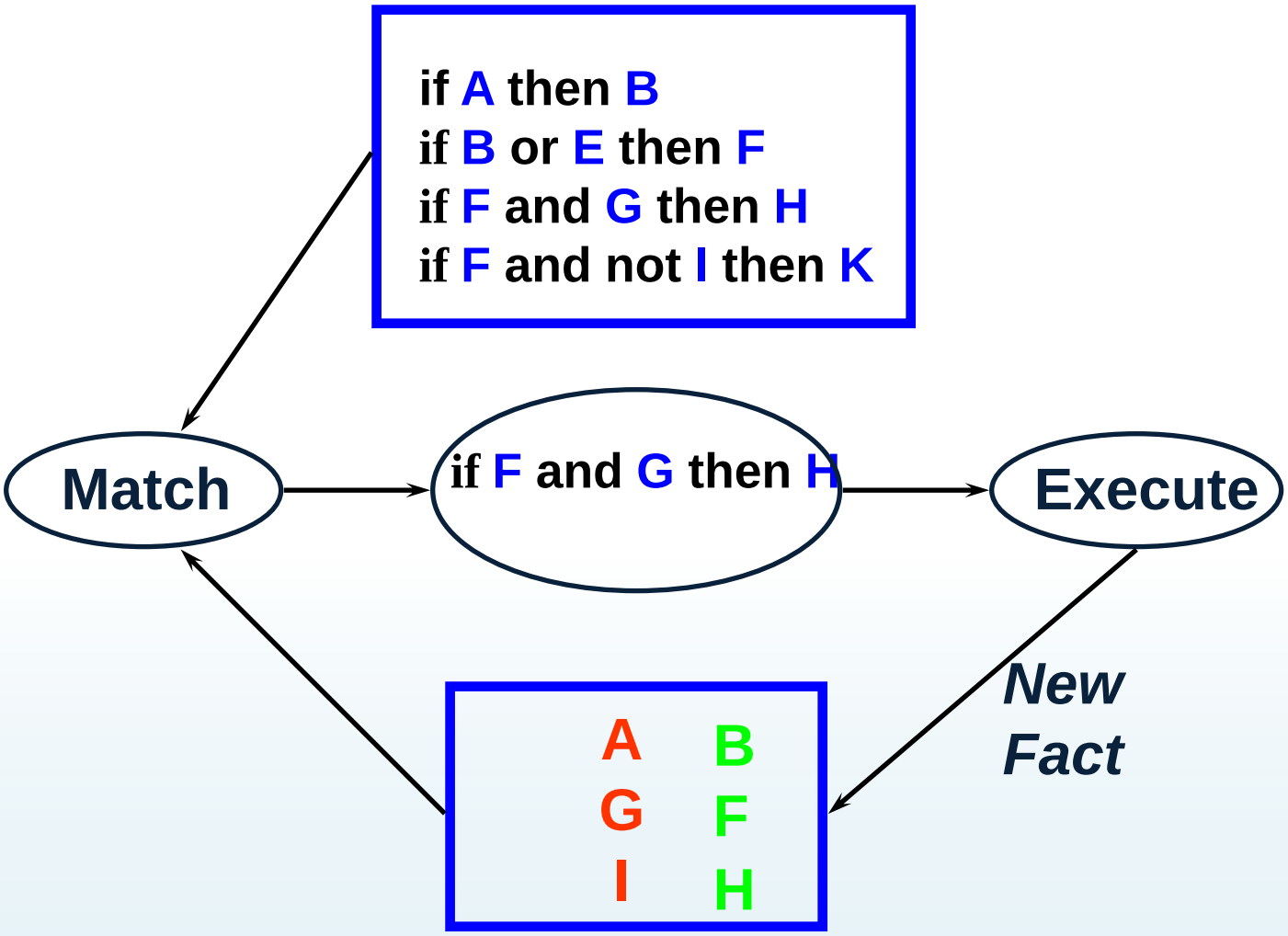
Recognize-Act Cycle



Recognize-Act Cycle



Recognize-Act Cycle



Action Systems

- The if-parts specify the conditions that have to be satisfied and the then-parts specify actions that need to be undertaken
- Actions
  - Add an assertion
  - Delete an assertion
  - Execute some procedure that has nothing to do with working memory or rule base

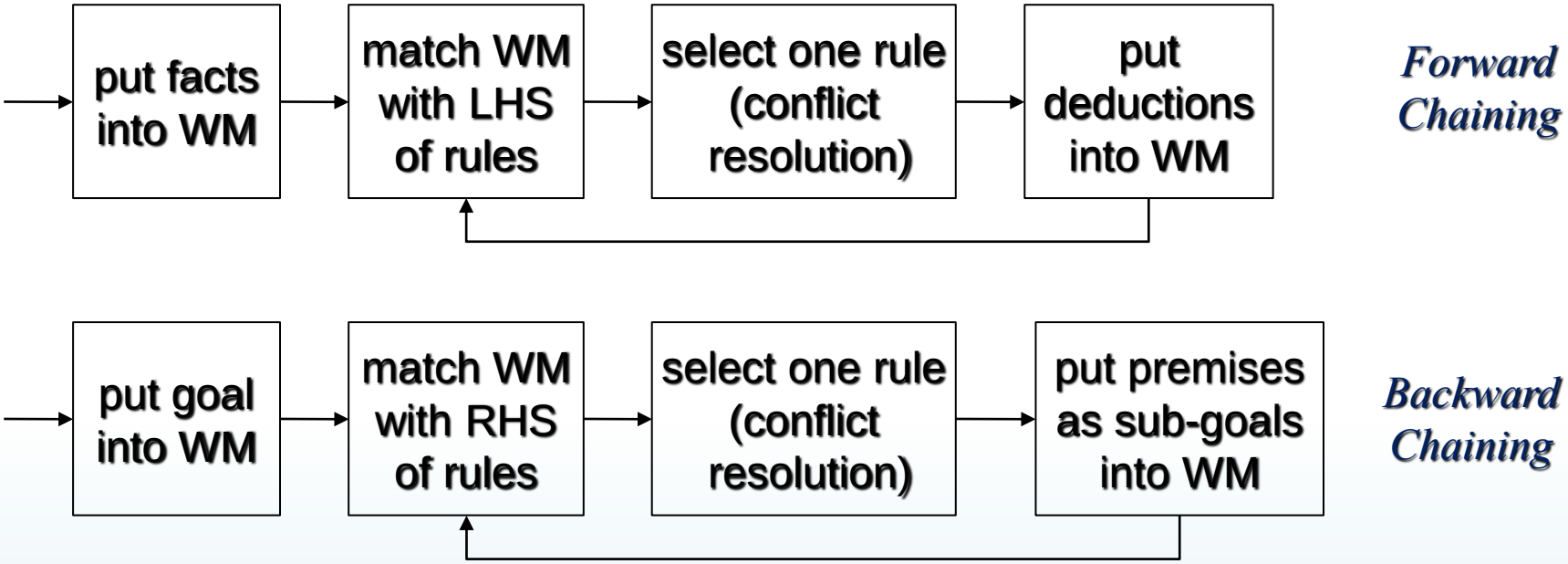
Conflict resolution

Let us have the following Example

- *Rule* 1:
  IF the 'traffic light' is green
  THEN the action is go
- *Rule* 2:
  IF the 'traffic light' is red
  THEN the action is stop
- *Rule* 3:
  IF the 'traffic light' is red
  THEN the action is go

- We have two rules, *Rule* 2 and *Rule* 3, with the same IF part.
- Thus both of them can be set to fire when the condition part is satisfied.
- These rules represent a conflict set.
- The inference engine must determine which rule to fire from such a set.
- A method for choosing a rule to fire when more than one rule can be fired in a given cycle is called **conflict resolution**.

Conflict Resolution



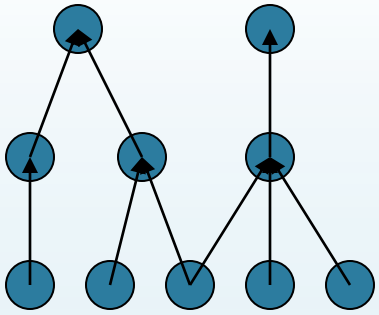
- what to do if there is more than 1 matching rule in each inference cycle?

Conflict Resolution

- ***conflict set***
  - the set of rules which match the WM content in each cycle
    - not just rules, but rules with ***variable instantiations***
  - ***deterministic*** rule set
    - at most 1 rule matched at each cycle (i.e. 0 or 1)
  - ***non-deterministic*** rule set
    - more than 1 matching rule
    - inference engine (interpreter) must select one rule to apply using its ***conflict resolution strategy***

Choice of Conflict Resolution Strategy

- Determine how the ES search in the problem space
  - how ***quick*** to find the solution
  - if the system will find a solution
    - and if found, which solution
- Should be chosen to fit your application
- Desirable properties
  - ***stability***
    - follow a line of reasoning
  - ***sensitivity***
    - react quickly to any change in WM



Conflict Resolution Strategy

- A common problem in rule-based system is if several rules have been activated at the same time, what rule should be fired first???

1. Rule ordering
2. Recency

   The most recently used data has highest priority(time ordering)
3. Most Specific

   This strategy is based on the number of conditions of the rules. (rule with most condition clauses)
4. User-defined priority

   Rule with highest priority (according to user-defined priority setting) applies
5. Refraction

   A rule should not be allowed to fire more than once on the same data (facts).

Step-1: Order by Salience(priority)

- In which order are rules placed on the AGENDA?
  - SORT by Salience: Newly activated rules are placed above all rules with lower salience and below all rules with higher salience.
  - Assume the following rules are on the AGENDA:

    Rule-1 salience = 100;

    Rule-2 salience = 50;

    & a new rule is activated:

    Rule-3 salience = 75;

    Then the order will be as follows:

    Rule-1 salience =100;

    Rule-3 salience = 75;

    Rule-2 salience = 50;

1-Priority (salience)

- in some systems, rules can have a ***priority*** (***salience***) attached, indicating how likely the rule is to be a good solution
- fire rule with ***highest*** priority
- ***priority*** should be used to:
  - emphasize the importance of a rule
  - delay firing of a non-promising rule
  - ***NOT*** to establish the order in which all the rules should fire

Step-2: Order by Strategy

- **What if two rules have equal salience?**
  - **SORT by Strategy: The conflict resolution strategy is used to determine placement of a rule that has the same salience as another.**
  - **Assume the previous Rule-1 and Rule-2 are on the AGENDA & the newly activated rule has the same salience as Rule-1:**

    Rule-3 salience = 100 ;

    Will the order be?

    Rule-1 salience =100;
    Rule-3 salience = 100;
    Rule-2 salience = 50;

    Or?

    Rule-3 salience =100;
    Rule-1 salience = 100;
    Rule-2 salience = 50;

1- Priority: Example

```
(defrule rule-1
(atest 1)
(atest 2) =>
(assert (rule 1 fires)))

(defrule rule-2
(atest 1)
(atest 3) =>
(assert (rule 2 fires)))

(defrule rule-3
(atest 3) =>
(assert (rule 3 fires)))

(defrule rule-4
(declare (salience 1))
(atest 1) =>
(assert (rule 4 fires)))
```

```
(deffacts try-CR
(atest 1)
(atest 2)
(atest 3))
```

```
(assert (rule 4 fires))
(assert (rule 2 fires))
(assert (rule 3 fires))
(assert (rule 1 fires))
```

2-Specificity (complexity)

- more ***specific*** rules are preferred to more general rules
  - rules with greater number of conditions
  - rules with fewer variables are more specific
  - more instantiated by the WM are more specific
- more ***specific*** rules should be applied earlier because they use more data and so can be used for special cases or exceptions to ***general*** rules

Specificity: Example

```
(defrule rule-1
(atest 1)
(atest 2) =>
(assert (rule 1 fires)))

(defrule rule-2
(atest 3) =>
(assert (rule 2 fires)))

(defrule rule-3
(atest 1)
(atest 2)
(atest 3) =>
(assert (rule 3 fires)))
```

```
(deffacts try-CR
(atest 1)
(atest 2)
(atest 3))

(assert (rule 1 fires))
(assert (rule 2 fires))

(assert (rule 3 fires))
(assert (rule 1 fires))
(assert (rule 2 fires))
```

3- Conflict Resolution Recency

- The rule is fired which uses the data added most recently to the working memory (a common mistake is to use the rule added most recently to the knowledge base - wrong).
- E.g.
- Rule 1: IF A AND B THEN C
  Rule 2: IF D AND E THEN F
  Rule 3: IF G AND H THEN I
- If the contents of working memory are B, A, H, G, E, D

rule 2 will fire, as D and E are the most recent additions. (the 'freshest' evidence in the trail).
Then rule 3
Then rule1

3-Recency

- Rules that use ***recently added data*** are favored (***depth-first***)
- if more than one rule use the same (most) recent data, the next most recent data used in each of the rules is used to determine which one is preferred

Conflict Resolution Strategies (1)

- Based on either one of the following:
  - Refraction-based:
    - A rule is not allowed to fire more than once on the same set of facts.
  - Specificity-based: Simplicity and Complexity
    - Rules with a greater number of conditions are harder to satisfy and are preferred because they take more facts into account.
  - Recency-based:
    - Every fact promoted to the Fact-Base gets a time stamp. This allows us to know which facts are more recent than others.

- 1- IF (lecturing X) AND (marking- practicals X)

THEN ADD (overworked X)

- 2- IF (month February)

THEN ADD (marking- practicals Alison)

- 3- IF (overworked X) OR (slept-badly X)

THEN ADD (bad-mood X)

- 4- IF (bad-mood X)

THEN DELETE (happy X)

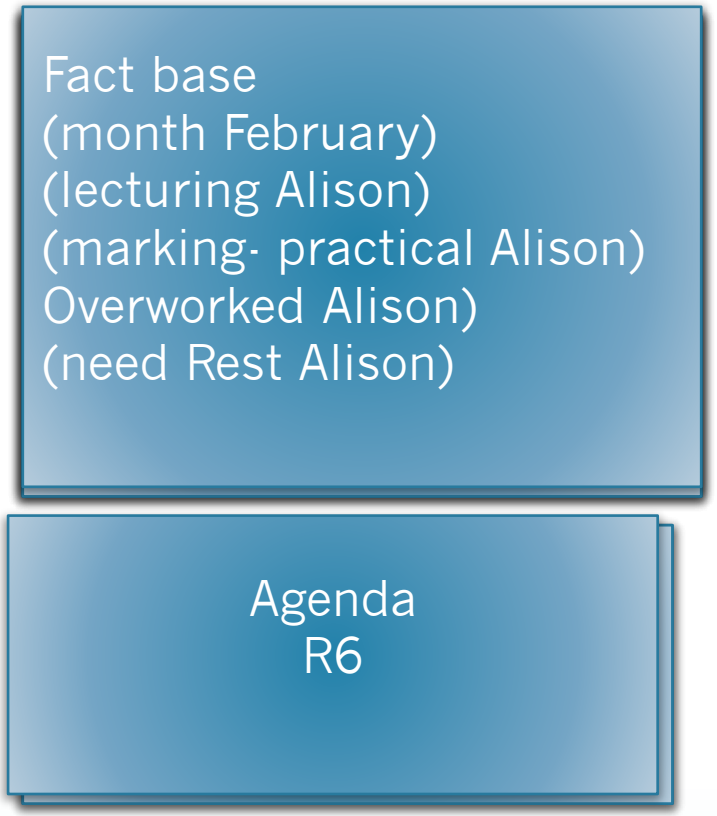
- 5- IF (lecturing X)

THEN DELETE (researching X)

- 6- IF (marking – praticals X)

THEN ADD(Needsrest X)

X is used to represent variable



A Simple Example

```
(defrule rule1
        (declare (salience 1))
        (f1)
        (f2)
        (f3)
=>
        (assert (action1))
)

(defrule rule2
        (declare (salience 10))
        (f2)
=>
        (assert (action2))
)

(defrule rule3
        (declare (salience 5))
        (f1)
        (f2)
=>
        (assert (action3))
)
```

```
(defrule rule4
        (declare (salience 5))
        (f1)
        (f2)
        (f3)
=>
        (assert (action4))
)


(deffacts the-facts
        (f1)
        (f2)
)

(deffacts recent-facts
        (f3)
)
```

Depth

- Depth is the default strategy. You only need it to specify it if you have been using another strategy and would like to revert back to it.
- How it works: Within the same salience class, place the new rules above all rules of same salience:

  R1.

  R2, R1.

  R2, R3, R1.

  R2, R4, R3, R1.

Breadth

- How it works: Within the same salience class, place the new rules below all rules of same salience:

  R1.

  R2, R1.

  R2, R3, R1.

  R2, R3, R4, R1.

Specificity

- How it works: Within the same salience class, place newly activated rules above all rules with equal or lower specificity i.e. priority over rules of same salience and less premises to satisfy:

  R1.

  R2, R1.

  R2, R3, R1.

  R2, R4, R3, R1.

Conflict Resolution in CLIPS

- CLIPS fires rules from the agenda in order of their *salience*.
- Salience is normally defined explicitly
- CLIPS offers a variety of CR strategies.
  - Recency (depth), breadth, LEX, MEA, complexity (Specificity) , simplicity & random strategies.
  - **default is recency**.
- In the CLIPS IDE you can set the CR strategy:
  - Commands -> Execution -> Option -> Salience Evaluation
- You can also set it at the prompt. It will return the previous strategy.
  - `CLIPS> (set-strategy breadth)`
  - `depth`
  - `CLIPS>`