Introduction to CLIPS
(C Language Integrated Production System)

Prof. Khaled Shaalan

Khaled.shaalan@buid.ac.ae

CLIPS and Knowledge Management

- CLIPS is an expert systems building tool
  - In development it is used to capture Knowledge
  - In usage it is used to apply knowledge

CLIPS: An expert system tool

- Tool
  - Complete environment for developing expert systems, including integrated editor and debugger
- Shell
  - Performs inferences and reasoning, i.e. uses the created rule and fact system
- freeware expert system tool
- The link for the CLIPS web site
  - *http://clipsrules.sourceforge.net/*
- Download
  - *http://sourceforge.net/projects/clipsrules/files/CLIPS/*

The CLIPS Programming Tool

- History of CLIPS
  - Influenced by OPS5 (early OOP language)
  - Implemented (source code) in C for efficiency and portability
  - Developed by NASA, distributed & supported by COSMIC
  - Runs on PC, Mac, also under UNIX and VAX VMS
- CLIPS provides mechanisms for expert systems
  - A top-level interpreter
  - Production rule interpreter
  - Object oriented programming language
  - LISP-like procedural language

Components of CLIPS

- Rule-Based Language
  - Can create a fact list
  - Can create a rule set
  - An inference engine matches facts against rules
- Object-Oriented Language
  - Can define classes
  - Can create different sets of instances
  - Special forms allow you to interface rules and objects

Example. Expert system for diagnosing car problems.

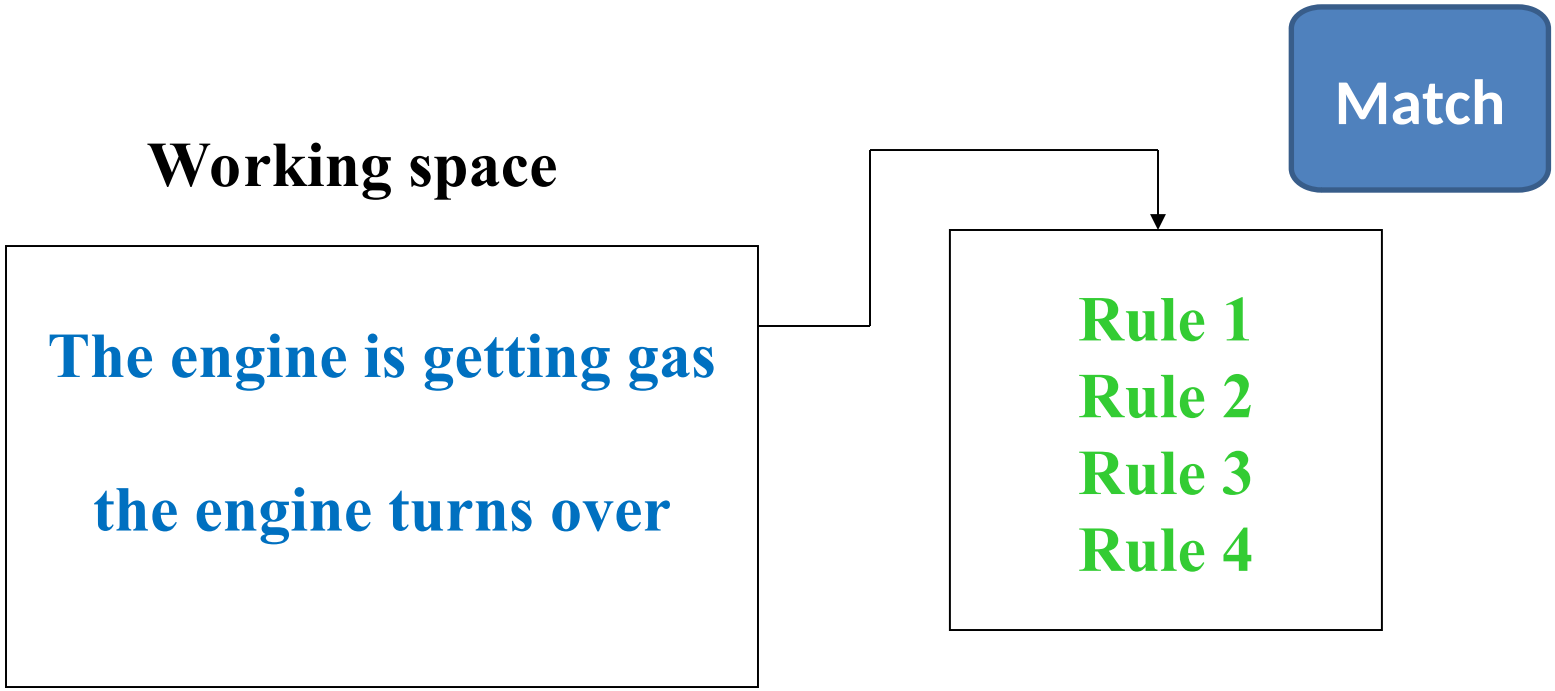
Rule 1: IF the engine is getting gas
AND the engine will turn over
THEN the problem is spark plugs

Rule 2: IF the engine does not turn over
AND the lights do not come on
THEN the problem is battery or cables.

Rule 3: IF the engine does not turn over
AND the lights do come on
THEN the problem is the starter motor.

Rule 4: IF there is gas in the fuel tank
AND there is gas in the carburettor
THEN the engine is getting gas

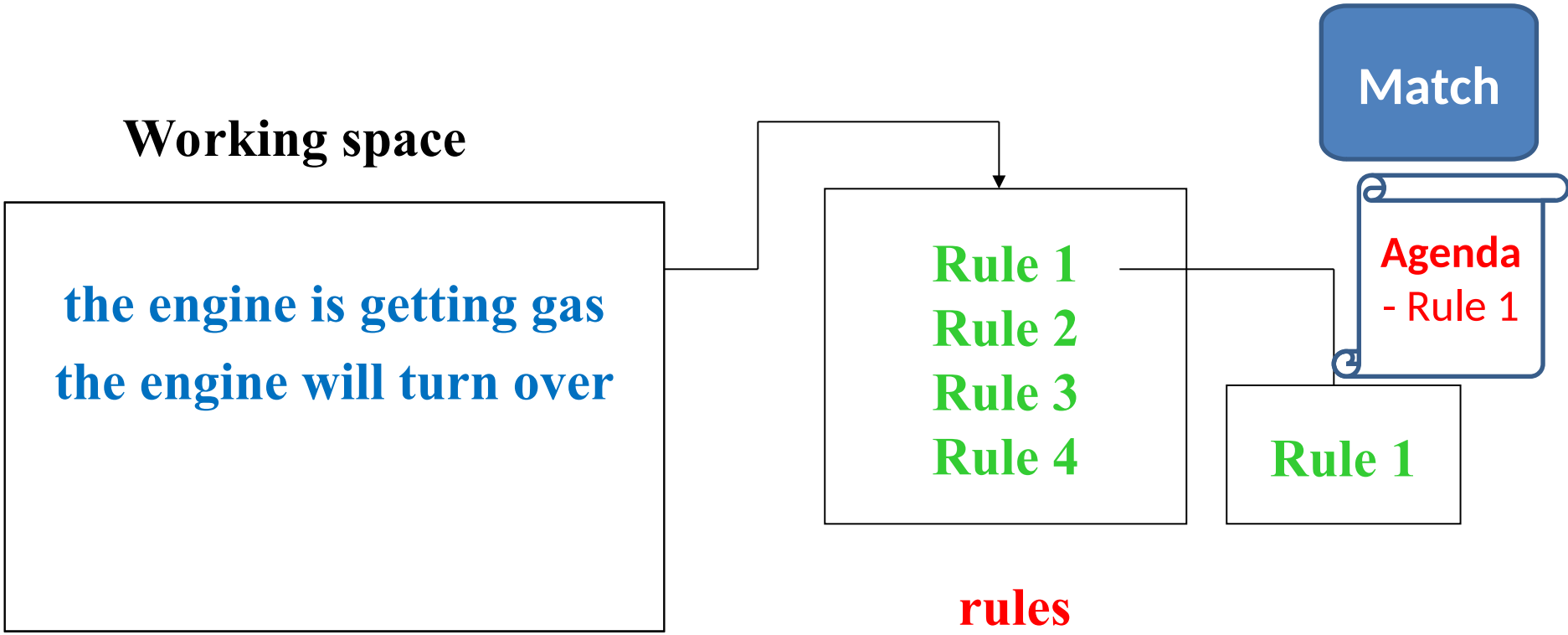
<u>Example</u>. Expert system for diagnosing car problems.



Facts
(knowledge we know)

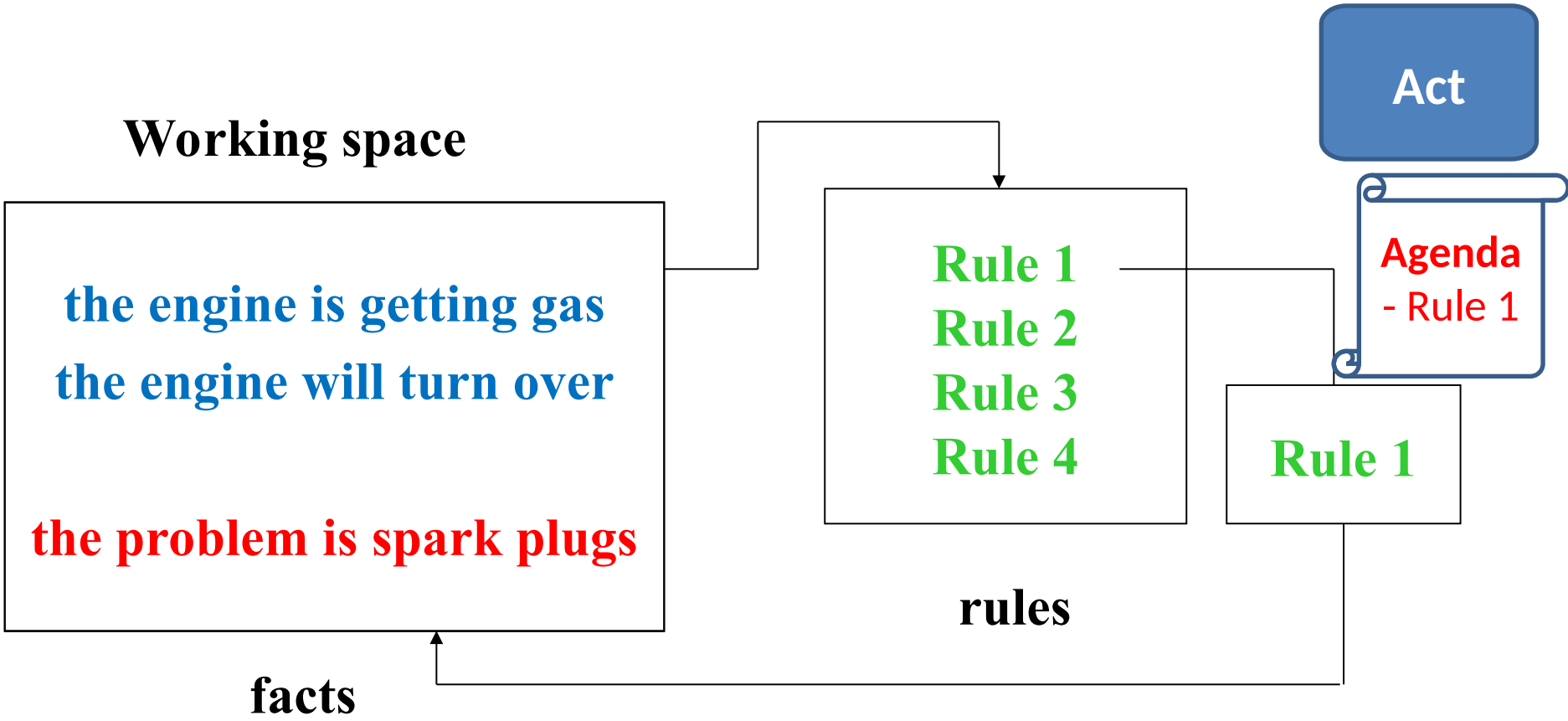
rules

<u>Example</u>. Expert system for diagnosing car problems.



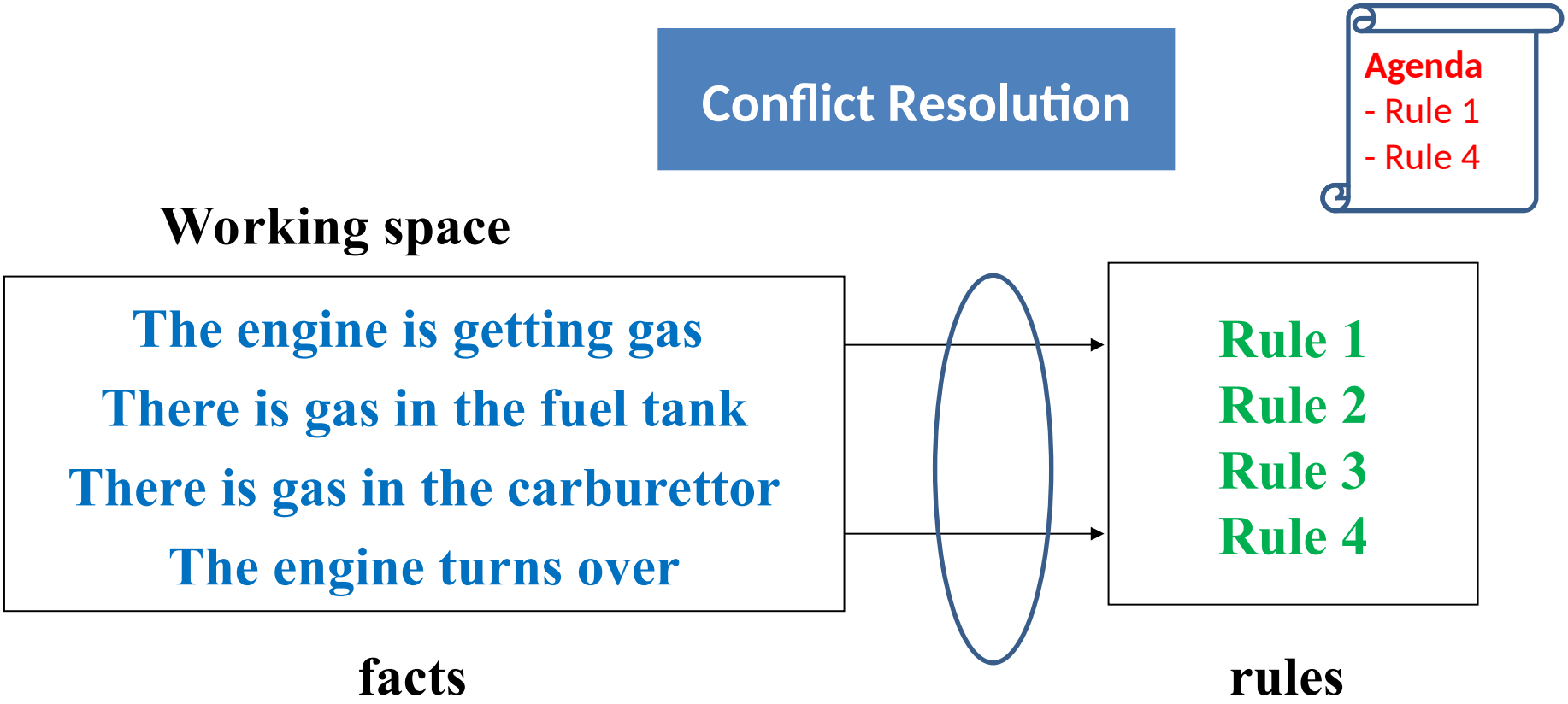
Rule 1: IF the engine is getting gas
AND the engine will turn over
THEN the problem is spark plugs

<u>Example</u>. Expert system for diagnosing car problems.



Rule 1: IF the engine is getting gas
AND the engine will turn over
THEN the problem is spark plugs

<u>Example</u>. Expert system for diagnosing car problems.



Rule 1:

IF the engine is getting gas
AND the engine will turn over
THEN the problem is spark plugs

Rule 4:

IF there is gas in the fuel tank
AND there is gas in the carburettor
THEN the engine is getting gas

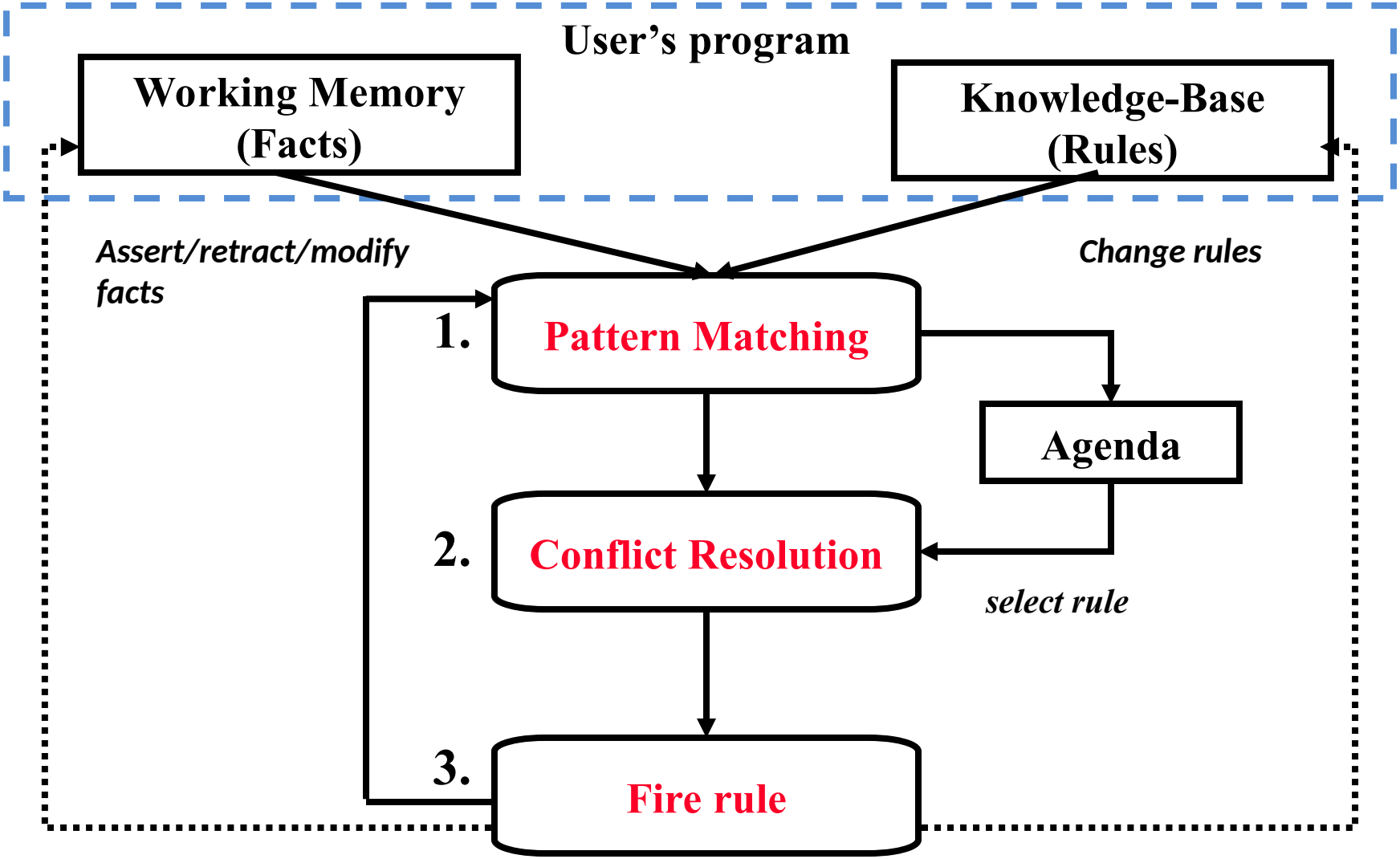
CLIPS

(C Language Integrated Production System)

Interpreter uses *recognize-act* **Cycle**:

1. *Match* - find all rules with matched antecedents
   a. **each combination of facts that satisfies a rule is called an *instantiation***
   b. **each matching rule is added to the *agenda***
2. *Conflict Resolution* - select a rule from the agenda to execute. If none, halt.
3. *Act* - execute rule performing specified actions
4. *Repeat* - go back to step 1.

A Production System Cycle



CLIPS – Programming Systems

- CLIPS Editor
  - load, save and edit CLIPS program files
- CLIPS Interpreter
  - enter commands, execute CLIPS programs
- Execution-Menu
  - set execution parameters (e.g. watch)
- Browse-Menu
  - manage constructs

CLIPS Data Base

Consists of a list of facts

Each fact:

- consists of one or more fields enclosed by parentheses

  (x y z)

  (> 3 1)

  (pen color red)

- Three facts with three fields each.
- represents a piece of information
- used to represent the current state of the problem
- is declarative knowledge
- First field in a fact should express a relationship

CLIPS Data Base (cont.)

In Clips program:

- Initial state of a problem (or the problem domain) is defined by a *deffacts*
- New facts are added by *assert*
- *Current facts are displayed by facts*
- Re-asserted facts are ignored- called *Refraction*
- Old facts are removed by *retract*

Fields of a Fact

Words (symbols)

- A sequence of alphabetic, numeric, underscores, or dash symbols

foo

foo-bar

foo_bar

foo-12

- Note that CLIPS is case sensitive

foo is not the same as *FOO* or *Foo*

Fields of a Fact (cont.)

Strings

- A collection of characters within double quotation marks
- Strings and words are not equivalent

"cat" is not the same as *cat*

Numbers

- Are stored in a single precision, floating point representation

237 237.0 2.37E+2

are all the same!

Example Facts

(foo 1286 “this is field 3”)
(this is a fact with 7 fields)
(“This is a facts with 1 field - a string”)

(animal-is walrus)
(animal-is duck)
(animals-are duck horse cow)

(said duck “quack”)

(address 1000 main st)
(address 1000 “main st”)
(address “1000 main st”)

Address with diff. no. of fields

ordered Facts

Use the first field to describe relationship between subsequent fields:

(<*relation*> <*field-1*> <*field-2*> …)

Use *object-attribute-value* and *attribute-value* formats:

(person l-name smith
f-name john
ssn 123457689
dept engineering)

Relation

attribute

value

Asserting and Retracting Facts

Actions:

```
clips> (reset) ; you may use Ctrl-E
clips> (assert (plays ivan tennis))
clips> (assert (plays martina tennis))
clips> (retract 1)
clips> (assert (plays martina
tennis))
FALSE
clips>
```

The data base: use (facts) to show data base)

```
f-0 (initial-fact)
f-1
f-2 (plays martina tennis)
```

No Change!

Refraction

Asserting multipe Facts

Actions:

```
clips> (reset)
clips> (assert  (a) (b) (c))
clips>
```

The data base: use (facts) to show data base)

```
f-0  (initial-fact)
f-1  (a)
f-2  (b)
f-3  (c)
```

Linking Facts through Common Fields

Sometimes it is advantageous to create multiple facts logically linked together by a common field:

(person ssn 123456789 l-name smith
f-name john dept engineering)

(personal ssn 123456789 age 31
height 71 weight 175
sex male m-status single)

(financial ssn 123456789 salary 45000
title senior-engineer)

Same Person

Why Link Facts through Common Fields?

(machine id m-1 status idle cur-order none)
(machine id m-2 status idle cur-order none)
…
(machine id m-10 status idle cur-order none)

(order id o-1 status waiting requires m-1)
…
(order id o-9 status waiting requires m-8)

Now envision rules:

IF order waits for a specific machine *and*
the machine is idle
THEN assign order to machine *and*
change status of order to assigned *and*
change status of machine to busy

CLIPS Notation

- Symbols other than those delimited by < >, [ ], or { } should be typed exactly as shown.
  - E.g degfacts
- [ ] mean the contents are optional
  - E.g [comment]
- and < > mean that a replacement is to be made.
  - E.g deffacts <anyname>

CLIPS Notation

- A Description followed by * means that the description can be replaced by zero or more occurrences of the specified value.
  - E.g. <facts> * zero or more facts
- A Description followed by + means that the description can be replaced by one or more occurrences of the specified value.
  - E.g <file name>+ one or more file name
- A vertical bar | indicates a choice among one or more of the items separated by the bars.
  - E.g. +|plus indicates plus sign or the word plus

Control of Fact Base: Deffacts Construct

- The *deffacts* construct can be used to assert a group of facts.

```
(deffacts <deffacts name>  [optional comment]
        <facts> * )
```

- The *reset* command is used to assert the facts in a deffacts statement.

Control of Fact Base

Undeffacts

removes specified deffacts statement from memory

(undeffacts <*deffacts-name*>)

Note that CLIPS provides a special deffacts statement:

(deffacts initial-fact

(initial-fact))

Load

used to load file containing knowledge base into memory. It does *not* execute the deffacts statement(s)

(load "myfile-name")

How Do These Work

Load places information from a file into memory storage

Reset takes information in memory storage and creates material needed for program

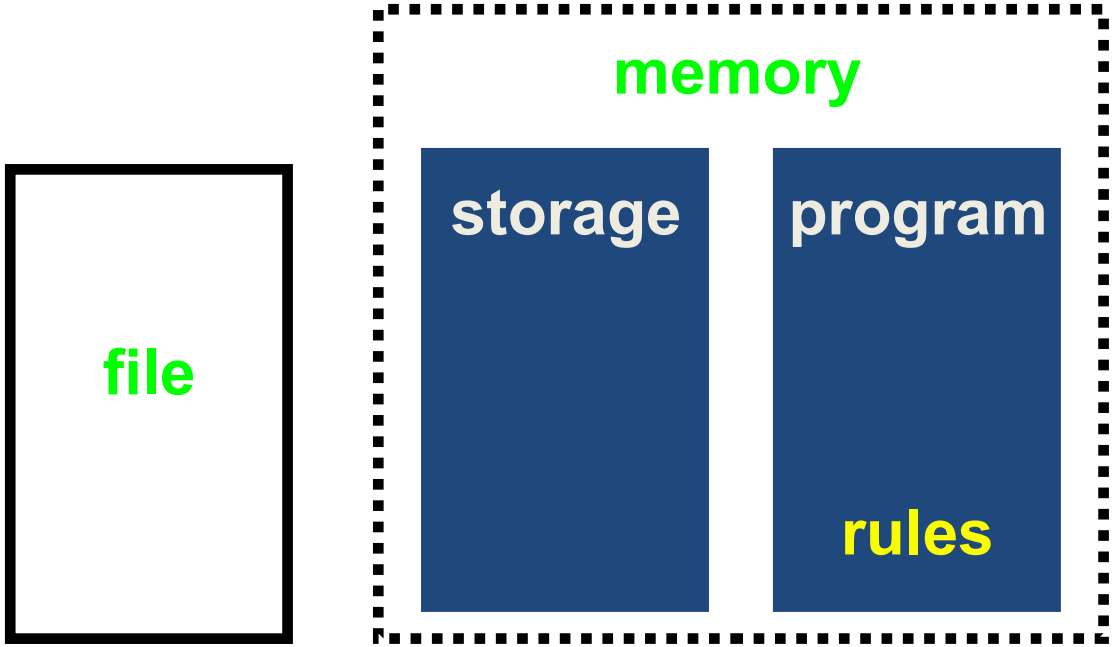
Undeffacts removes information from memory storage

(load "file")

(reset)

(undeffacts "X")

(reset)



Entering / Exiting CLIPS

- The CLIPS prompt is: CLIPS>
- This is the top-level mode where commands can be entered.
- To exit CLIPS, one types: CLIPS> (exit) ↵
- CLIPS will accept input from the user / evaluate it / return an appropriate response:

  CLIPS> (+ 3 4) ↵ → value 7 would be returned.

Example

Assume the following is in the file "mfile":

```
(deffacts   initial-machine-configs
            (machine  id  m-1  status  idle)
            (machine  id  m-2  status  idle)
            (machine  id  m-3  status  idle))
```

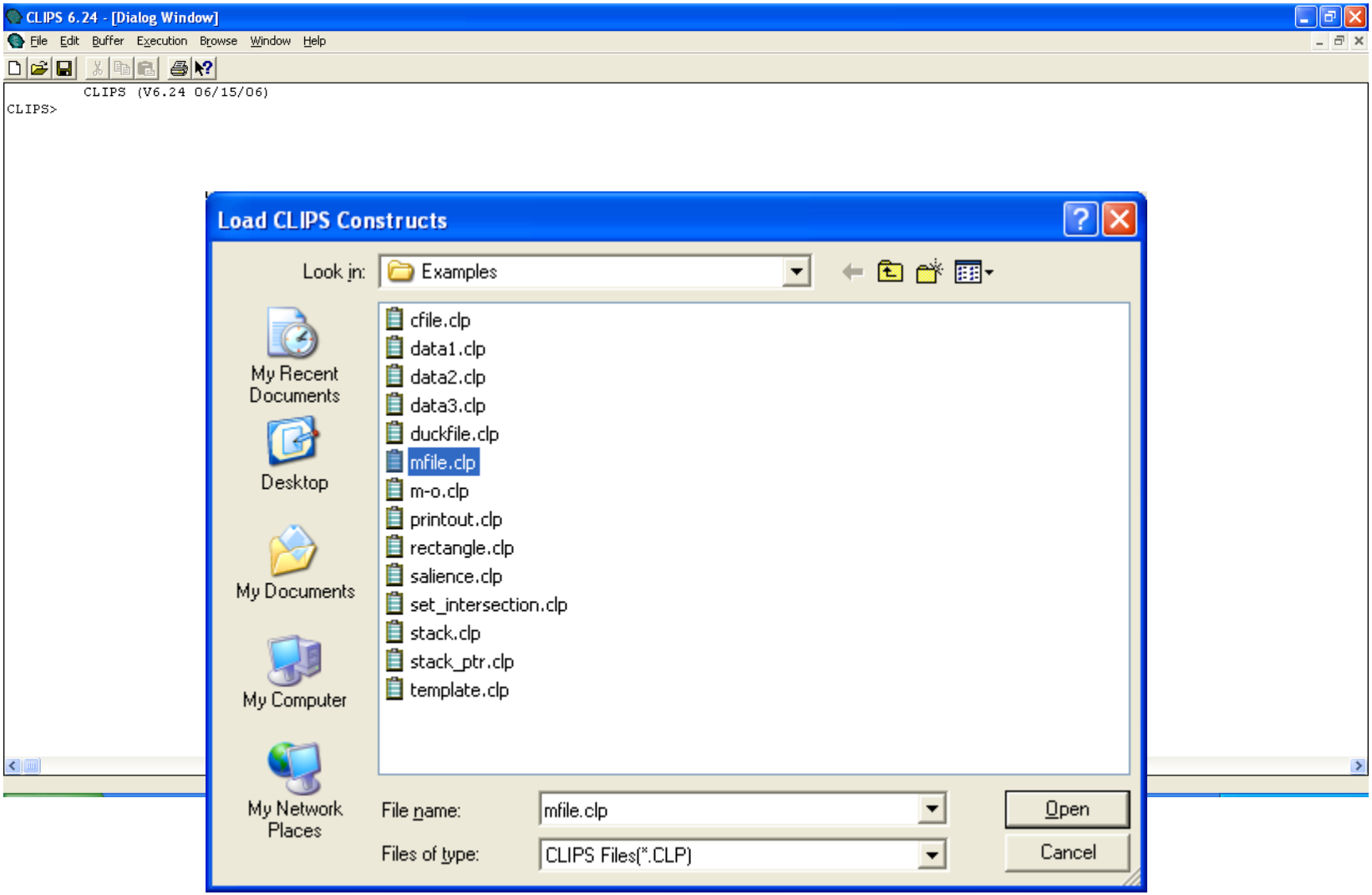
We now enter the following commands:

```
clips>  (load "mfile")
clips>  (reset)
clips>  (facts)
```

What's in memory?

```
f-0  (initial-fact)
f-1  (machine id m-1 status idle)
f-2  (machine id m-2 status idle)
f-3  (machine id m-3 status idle)
```

clips> (load “mfile”)



```
clips>  (reset)
```

clips> (facts)

CLIPS 6.24 - [Dialog Window]

File Edit Buffer Execution Browse Window Help

```
         CLIPS (V6.24 06/15/06)
CLIPS> (load "E:/512/handout/Examples/mfile.clp")
Defining deffacts: initial-machine-configs
TRUE
CLIPS> (reset)
CLIPS> (facts)
f-0     (initial-fact)
f-1     (machine id m-1 status idle)
f-2     (machine id m-2 status idle)
f-3     (machine id m-3 status idle)
For a total of 4 facts.
CLIPS>
```

Example (cont.)

Now we enter:

```
clips> (undeffacts initial-machine-configs)
clips> (facts)
```

What's in memory?

```
f-0  (initial-fact)
f-1  (machine id m-1 status idle)
f-2  (machine id m-2 status idle)
f-3  (machine id m-3 status idle)
```

What about now entering:

```
clips> (reset)
clips> (facts)
```

What's in memory?

```
f-0  (initial-fact)
f-1
f-2
f-3
```

Deftemplate

- Before facts can be constructed, CLIPS must be informed of the list of valid slots for a given relation name.
- A **deftemplate** is used to describe groups of facts sharing the same relation name and contain common information.

Deftemplate General Format

```
(deftemplate <relation-name> [<optional-comment>]
    <slot-definition>*)
```

↓

```
    (slot <slot-name>) | (multislot <slot-name>)
```

```
(deftemplate person "An example deftemplate"
    (slot name)
    (slot age)
    (slot eye-color)
    (slot hair-color))
```

Deftemplate vs. Ordered Facts

- Facts with a relation name defined using deftemplate are called *deftemplate facts*.
- Facts with a relation name that does not have a corresponding deftemplate are called *ordered facts* – have a single implied multifield slot for storing all the values of the relation name.

Adding (asserting) Facts

- CLIPS store all facts known to it in a fact list.
- To add a fact to the list, we use the *assert* command.

```
(deftemplate student
    (slot  name)
    (slot  age)
    (slot major))

(assert  (student (name  "John Summers")
                  (age  19)
                  (major  "Information Technology")))
```

Modifying Facts

- Slot values of deftemplate facts can be modified using the *modify* command:

Slot values of deftemplate facts can be modified using the *modify* command:

```
(modify <fact-index> <slot-modifier>+)
```

where <slot-modifier> is:

```
(<slot-name>  <slot-value>)
```

For example, we could make the following modification:

```
(modify 0 (age 21))
```

and then request to see the facts again:

```
(facts)↵

f-4 (student    (name  "John Summers")
                (age  21)
                (major  "Information Technology")))

For a total of 1 fact.
```

Results of Modification

- A new fact index is generated because when a fact is modified:
  - The original fact is retracted
  - The modified fact is asserted
- The *duplicate* command is similar to the *modify* command, except it does not retract the original fact.

Watch Command

- The *watch* command is useful for debugging purposes.
- If facts are “watched”, CLIPS will automatically print a message indicating an update has been made to the fact list whenever either of the following has been made:
  - Assertion
  - Retraction

Watch example (give it a try!)

```
CLIPS> (watch facts)
CLIPS> (reset)
==> f-0 (initial facts)
CLIPS> (deftemplate student (slot name) (slot age) (slot
   major))
CLIPS> (assert (student (name "John Summers")
(age 19) (major "Information Technology")))
==> f-1 (student (name "John Summers")
(age 19) (major "Information Technology"))
<Fact-1>
```

Watch example

```
CLIPS> (modify 1 (age 21))
<== f-1 (student (name "John Summers")
(age 19) (major "Information Technology"))
==> f-2 (student (name "John Summers")
(age 21) (major "Information Technology"))
<Fact-2>
```