Rule-based programming in Interactive Fiction

Or, How I Learned To Stop Worrying and Love them Declarative Languages*

* ...except for Prolog. That's still too weird.

Inform 7 (released April 30, 2006)

```
[...A fragment of Inform 7 source code...]

Check entering the cell-doors:
     instead say “You are not here to search the cells.”

Instead of climbing something when the location is the Guardroom:
     say “(that is, go up using [the noun])”;
     try moving the Hallway with the noun.

Check moving the Hallway with generic when the location is the
  Guardroom:
     if the closest enemy is not generic:
          instead say “The opening is above your unaided reach.
            But the guard is your more immediate problem.”;
     else:
          prepare move-with-tutorial;
          instead say “The opening is above your unaided reach.
            You will need to make use of something.”
```

Adventure (Colossal Cave)

Crowther, 1976; FORTRAN

```
67     FORMAT(' THERE ARE ',I2,'
   THREATENING LITTLE DWARVES IN THE
       1  ROOM WITH YOU.',/)
       GOTO 77
75     CALL SPEAK(4)
77     IF(ATTACK.EQ.0)GOTO 71
       IF(ATTACK.EQ.1)GOTO 79
       PRINT 78,ATTACK
78     FORMAT(' ',I2,' OF THEM THROW
   KNIVES AT YOU!',/)
       GOTO 81
79     CALL SPEAK(5)
       CALL SPEAK(52+STICK)
       GOTO(71,83)(STICK+1)
81     IF(STICK.EQ.0) GOTO 69
       IF(STICK.EQ.1)GOTO 82
       PRINT 68,STICK
68     FORMAT(' ',I2,' OF THEM GET
   YOU.',/)
       GOTO 83
82     CALL SPEAK(6)
83     PAUSE 'GAMES OVER'
```

Dungeon (Zork)

MIT, 1978; MDL

```
<DEFINE TRAP-DOOR ("AUX" (RM ,HERE))
    #DECL ((PRSA) VERB
           (RM) ROOM (DOOR) OBJECT)
    <COND (<AND <VERB?
                   "OPEN" "CLOSE">
                <==? .RM <SFIND-ROOM
                   "LROOM">>>
           <OPEN-CLOSE <PRSO>
"The door reluctantly opens to reveal
   a rickety staircase descending
   into darkness."
"The door swings shut and closes.">)
          (<==? .RM <SFIND-ROOM
                   "CELLA">>
           <COND (<VERB? "OPEN">
                  <TELL
"The door is locked from above.">)
                 (<TELL <PICK-ONE
                    ,DUMMY>>)>)>>
```

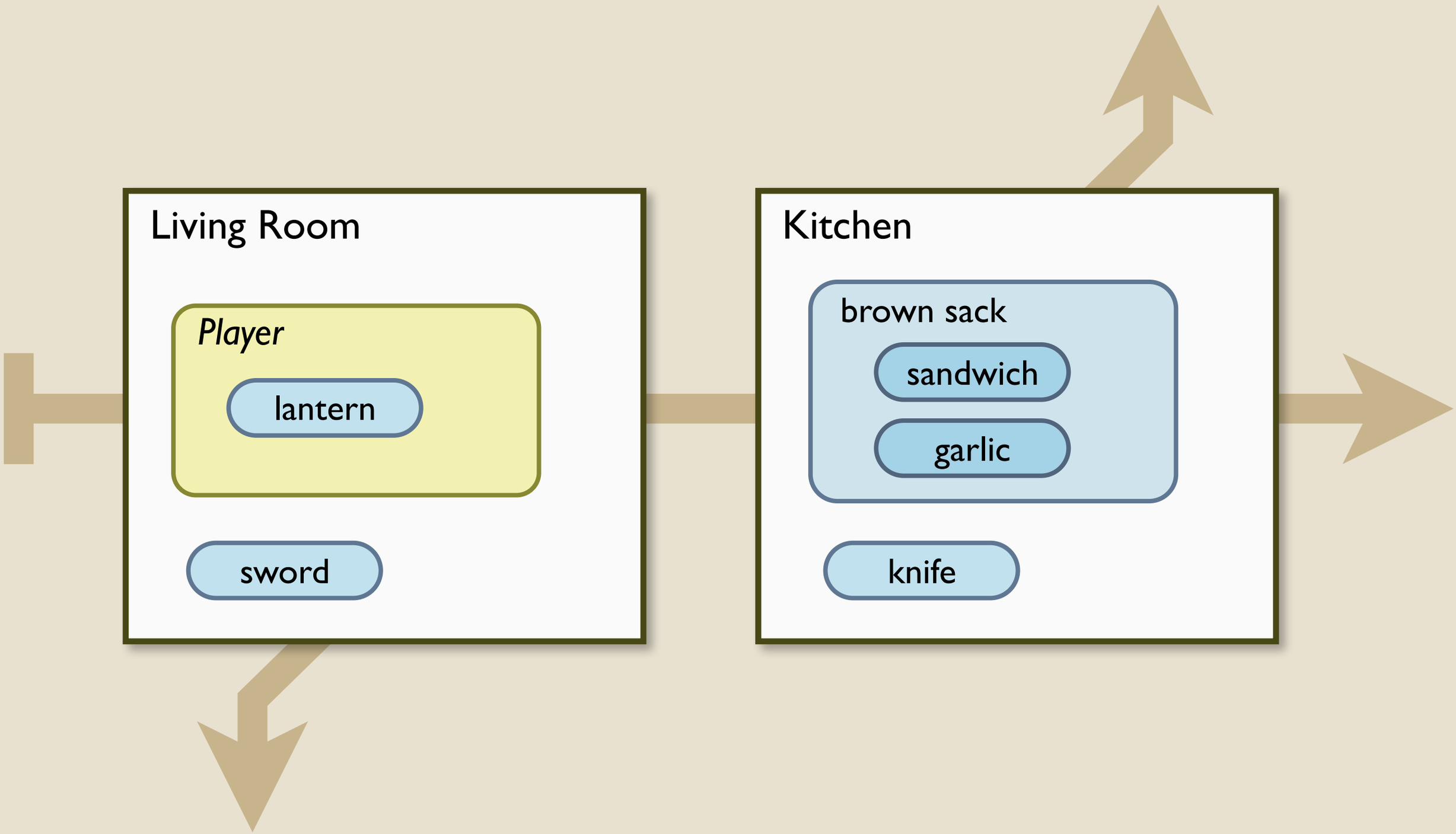
#3 *To stealthily survey the Temple of the Slumbering God, go to slide #4. To march up to the front gate, go to #5. To give up and leave, do neither.*

20 years of IF development systems

- 1981 – DDL
- 1983 – The Quill
- 1985 – GAGS
- 1986 – ADVSYS
- 1987 – AGT
- 1992 – ALAN
- 1992 – TADS 2
- 1994 – Inform 5
- 1995 – Hugo
- 1996 – Inform 6
- 1997 – ADRIFT

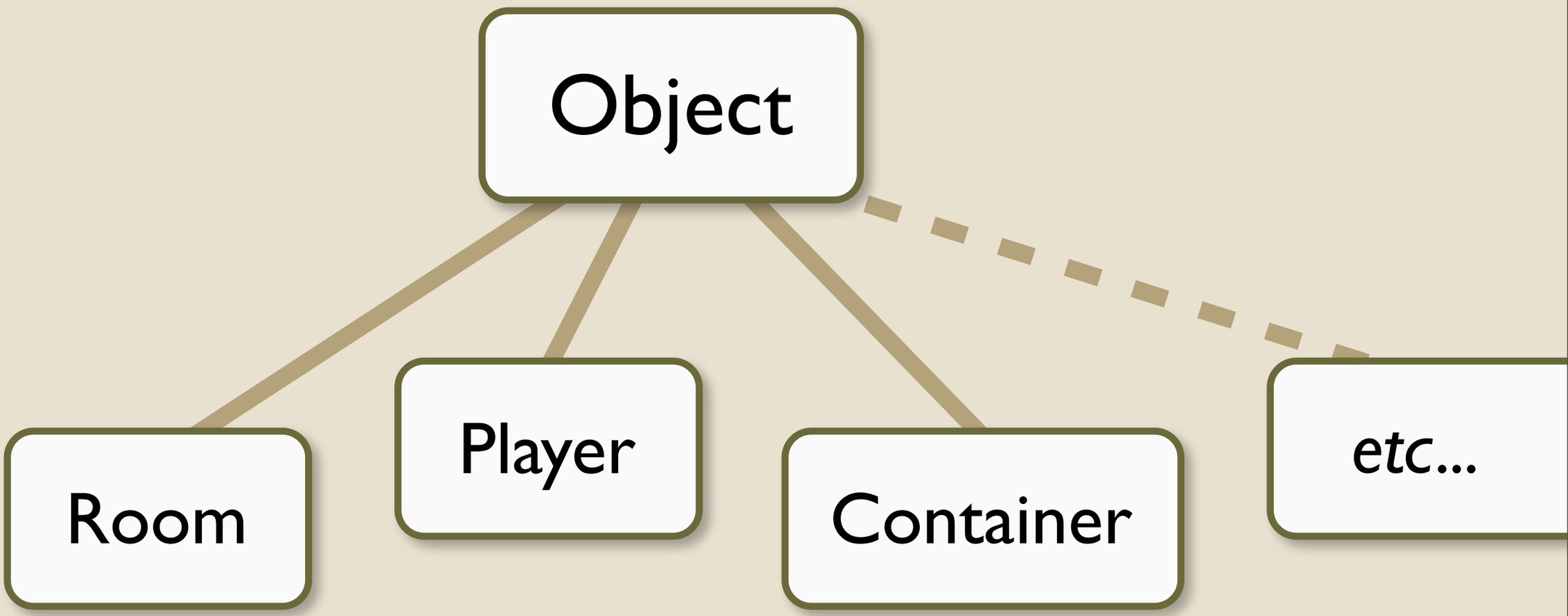
To crawl into the muddy hypocaust you discover behind the Temple, go to slide #7. To circle around to the front gate, go to slide #5.

What's in an IF game?



#5 *To carefully inspect the hinges of the immense iron gate, go to slide #8. To pound your sword-hilt on the gate until someone opens it, go to #10.*

What's in an IF game?



To be cursed by the Slumbering God for the thousandth time, go to slide #32.

The object-oriented approach

```
class Object:
    string description: "You see nothing special."

Object Bottle:
    string description: "A glass bottle."
```


To skewer the loathesome, slime-ridden lamprey-worm which inhabits the hypocaust, go to slide #9.

The object-oriented approach

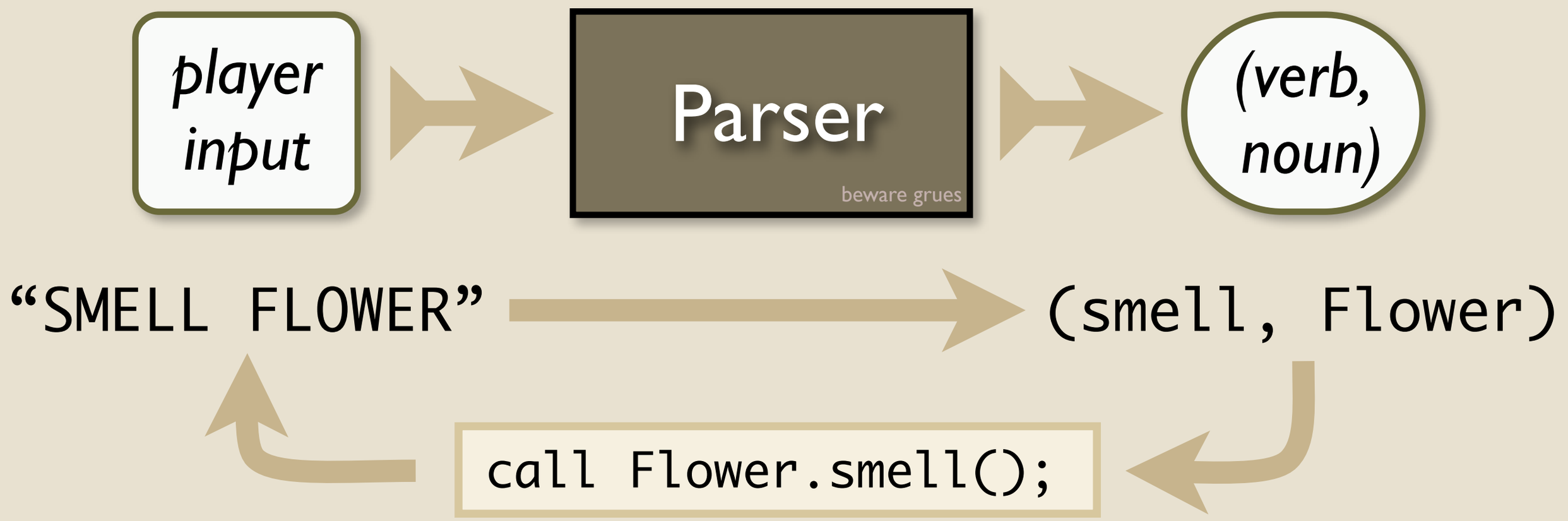
```
class Object:
  function description(): {
    print "You see nothing special.";
  }

Object Bottle:
  function description(): {
    if (self.empty)
      print "An empty glass bottle.";
    else
      print "A bottle of", self.contents, ".";
  }
```

#8 *To slip the catch which is so delicately hidden near the gate hinges, go to slide #12. To give up on the front gate and search the rear, go to #4.*

The OO IF model in action

```
Object Flower:
  method smell(): {
    print "Sweet as a rose.";
  }
```



To crawl out of the hypocaust into a disused temple hallway, go to slide #16. To descend into the labyrinth, go to slide #19.

Where to put more complex conditions?

"LOOK"	→	`Null.look()? Room.look?`
"EXAMINE SWORD" *(when room is dark)*	→	`Object.examine()?` `Darkness.examine()?`
"PUT MEAT IN BASKET"	→	`Meat.put(Basket)?` `Basket.put(Meat)?`
"TAKE PIE" *(when angry orc is in room)*	→	`Orc.takeNearby(Pie)?`
"PULL LEVER" *(when lever is electrified)*	→	`Lever.anyTouching()?`

Maybe if we put it all in one method...

```
Player.do(action, object) {
  if (isDark() && action is Examine/Read)
    { print "Too dark."; return; }
  if (Orc.nearby && action is Take) {
    { print "It stops you."; return; }
  switch (action) {
    case Look: ...
    case Take: ...
  }
}
```

...let's not.

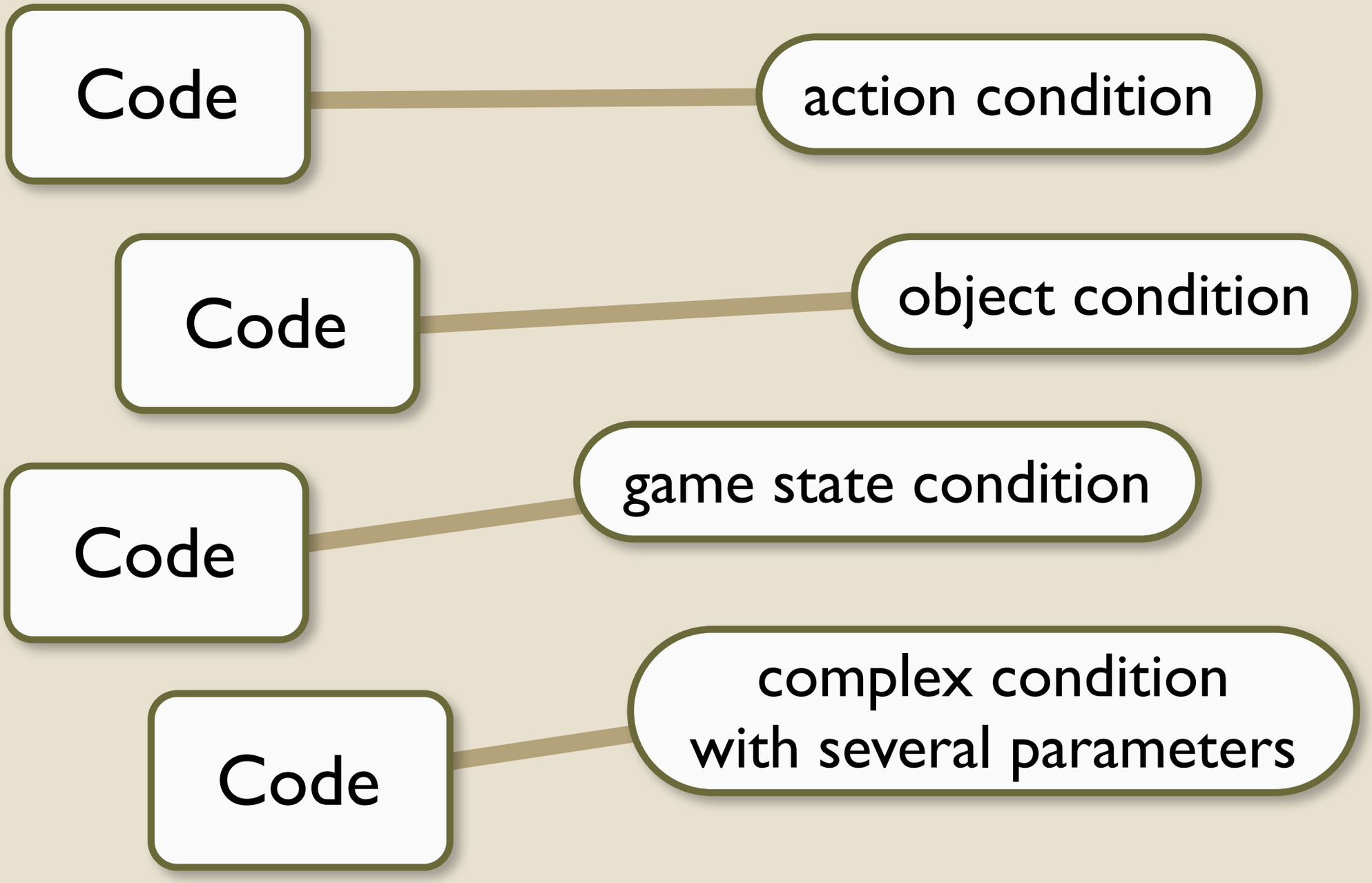
To descend into the demon's pit for the thousandth time, go to slide #6.

How Inform 6 does it

```
function PerformAction() {
  if player.does_action(...) return;
  foreach O in room:
    if O.nearby_action(...) return;
  if room.contains_action(...) return;
  if object.action(...) return;
  action_default(...);
}
```


To silently ease open the hidden sally-portal that you have discovered by the gate, go to slide #16. To smash it down, go to slide #10.

What model describes how I really work?



To be overwhelmed by the surprisingly well-trained guards, go to slide #14.

The rule: a general template

```
do ATOM as CODE if CONDITION
```

#14 *To be dragged to a vile prison cell by the guards, go to #17. To slip a wink to the masked hierophant who stands in the shadows, go to #18.*

The rule: a general template

```
do ATOM as CODE if CONDITION
```

```
do FuncName as {...code...} if true
```


To puzzle through the labyrinth for the thousandth time, go to slide #11.

The rule: a general template

```
do ATOM as CODE if CONDITION
```

```
do Description(obj) as "Nothing special."
  if true

do Description(obj) as "Looks valuable."
  if (obj ofclass Treasure)

do Description(obj) as "Huge and sparkly!"
  if (obj is HopeDiamond)
```


To seek the end of the secret passage, go to slide #23. To follow the masked figure who lays a hand on your arm, go to slide #18.

We could make atom just another parameter

```
do (atom, obj) as "Nothing special."
  if (atom is Description)

do (atom, obj) as "Looks valuable."
  if (atom is Description
      && obj ofclass Treasure)

do (atom, obj) as "Huge and sparkly!"
  if (atom is Description
      && obj is HopeDiamond)
```

...let's not.

To waken the filthy wretch who shares your cell, go to slide #21. To probe the ancient stonework for a hidden exit, go to slide #16.

More tricks with rules – constants

```
do Pi as 3.14159

do DarknessMessage as "It's very dark."
```


To attend the masked priest's babblings about the Eternal Ruby, go to slide #27. To search the hieratic chamber, go to slide #25.

More tricks with rules – methods

```
do Price(obj) as 0

do Price(obj) as 10
  if (obj ofclass Treasure)

do Price(obj) as {
  total = 0;
  foreach X in obj
    total += Price(X);
  return total;
} if (obj ofclass Container)
```

Assigning values at runtime

```
do Price(obj) as 10
  if (obj ofclass Treasure)
```

```
do Hit(MingVase) as {
  print "It cracks.";
  Price(MingVase) := 0;
}
```


To hide from the guards for the thousandth time, go to slide #15.

Assigning values at runtime

```
do Price(obj) as 10
  if (obj ofclass Treasure)

do Price(obj) as 0
  if (...the assignment below has occurred...)

do Hit(MingVase) as {
  print "It cracks.";
  Price(MingVase) := 0;
}
```


To take the key that the decrepit (but unpleasantly familiar) prisoner offers you, and slip out into the hallway, go to slide #22.

Assigning values at runtime

```
do Score as 0

do Slay(Dragon) as {
  print "The evil dragon is dead!";
  Score := Score + 10;
}

do Rescue(Princess) as {
  print "On the first date?";
  Score := Score + 20;
}
```


To flee the hoarse muttering of the guards behind you, go to slide #24. To hide in a gloomy chamber until they pass, go to slide #25.

Defining classes in terms of rules

```
do Treasure(obj) as false
do Treasure(HopeDiamond) as true
do Treasure(MingVase) as true

do Price(obj) as 0

do Price(obj) as 10
  if (Treasure(obj))
```

 To emerge from the hidden passage behind the foetorous idol, go to #24. To fall through a blood-rusted trap into the labyrinth, go to #19.

So why bother with any of this?

Or, let's back down the garden path a little

To seek the murderers of the pitiful wretches who died here, go to slide #25. To follow their time-lost spirits into the pit, go to #29.

Why rules?

- IF is made of exceptions.
- Handle complexity by being able to ignore most of it.
- A simple programming mechanism that can be made complicated later.
- You have to collaborate with the standard library (modify every part of it).
- Third-party libraries have to collaborate with the standard library, and with each other.
- You have to collaborate with yourself.

 To read in rotted scrolls of the dreams of the Slumbering God, which crawl up and down through time, until the priest finds you, go to #27.

Resolving rule collisions is the hard part

Some wrong ways to handle it:

- Logical precedence *(only works for the easy cases)*
- Source code order *(drives you insane)*
- Ornate table for which ops take precedence *(impossible to remember, and then usually wrong)*
- Throw an error, require fixing by hand *(billions and billions of errors)*

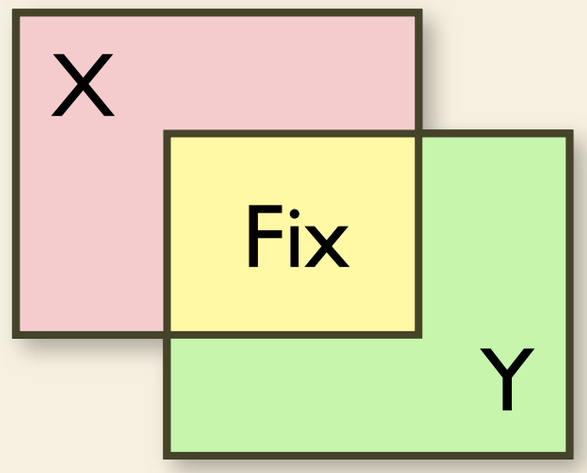
Fixing a rule conflict

```
Rule-X: do Description(person) as {...}
Rule-Y: do Description(obj) as {...}
  if IsDark
```

```
Rule-Fix: do Description(person) as {...}
  if IsDark
```

...or...

Rule-Y ⊃ Rule-X



 To unravel the secret of the labyrinth from the hierophant's raving, go to slide #26. To seek the pit of the Slumbering God, go to slide #29.

Resolving conflicts by groups of rules

```
Darkness-Extension ⊃ Standard-Library

        Game-Code ⊃ Library-Code

   My-Scenery-Rule ⊂ Darkness-Extension

Standard-Room-Rule ⊃ Game-Chapter-3

Standard-Room-Rule ⊃ My-Grue-Rule
```


To navigate the hypocaust for what seems like the thousandth time, go to slide #20.

Handling groups of rules with more rules

```
do GameGroup(rule) as true
  if (...rule is defined in game code...)

do LibraryGroup(rule) as true
  if (...rule is defined in library code...)

do Override(GameGroup, LibraryGroup)
  as true

do Override(LibraryRule27, Chapter3Group)
  as true
```

 To leap on the cyclopean form of the Slumbering God and stab it to the heart, go to #30. To lift the Ruby silently from the altar, go to #30.

Other difficult problems

- Complementary properties

```
Parent(X,Y) ⇔ Child(Y,X) ⇔ Nonempty(X)
```

- Multi-step operations

```
{ Perform(action); Report(action) }
```

- Adding steps

```
do Message as { "Prologue"; Message }
do Message as { Message; "Epilogue" }
```

- Retaining information across rules

 To snatch the Eternal Ruby from beneath the awakening demon, and then flee for your life, go to slide #31. To shatter the jewel, go to #3.

You can't demonstrate a complexity-management tool on toy problems

Or, I'm hosed

#31 *To be pierced by the Awakened God's gaze, scorched by the Ruby-light in your hands, and then dragged back into its riptide of time, go to #32.*

References

Dennis Jerz: *Somewhere Nearby is Colossal Cave*
http://www.digitalhumanities.org/dhq/vol/001/2/000009.html

Stephen Granade: *A Brief History of Interactive Fiction*
http://www.brasslantern.org/community/history/timeline-c.html

Graham Nelson: *Natural Language, Semantic Analysis and Interactive Fiction*
http://www.inform7.com/learn/documents/WhitePaper.pdf

The Interactive Fiction Wiki
http://ifwiki.org/

Andrew Plotkin: ongoing and completely unorganized notes on rule-based programming *(includes this presentation)*
http://www.eblong.com/zarf/rule-language.html

To enter the Temple for what seems like the thousandth time, go to slide #28.