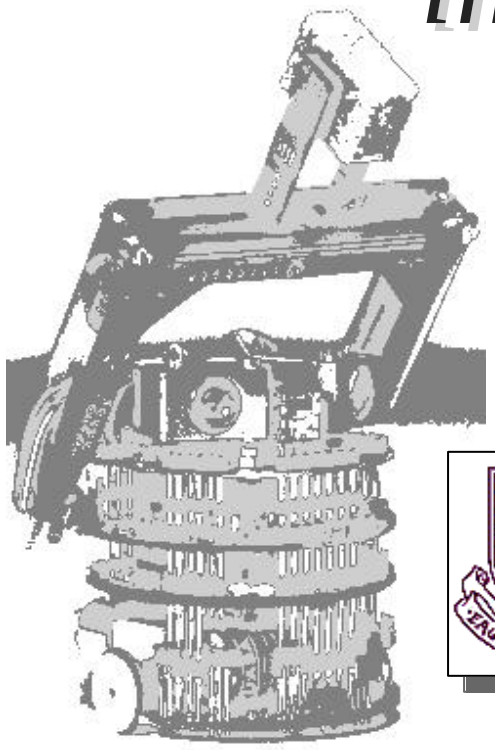
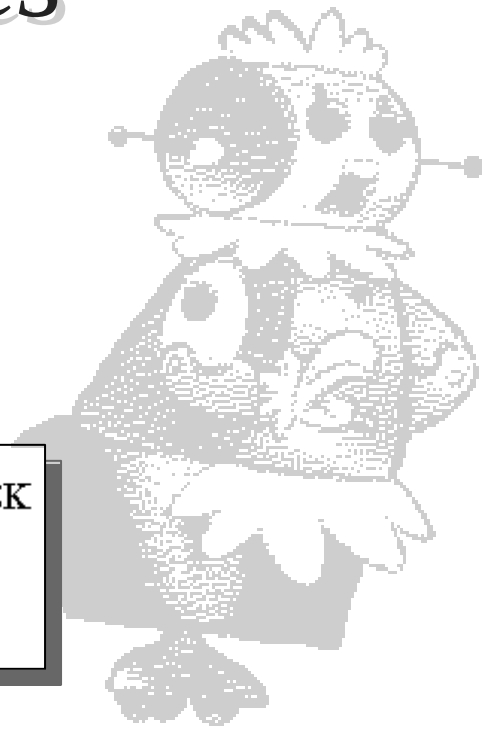
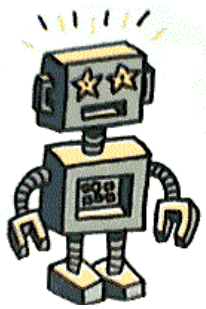
Behaviour-Based Control in Mobile Robotics

Paul Fitzpatrick

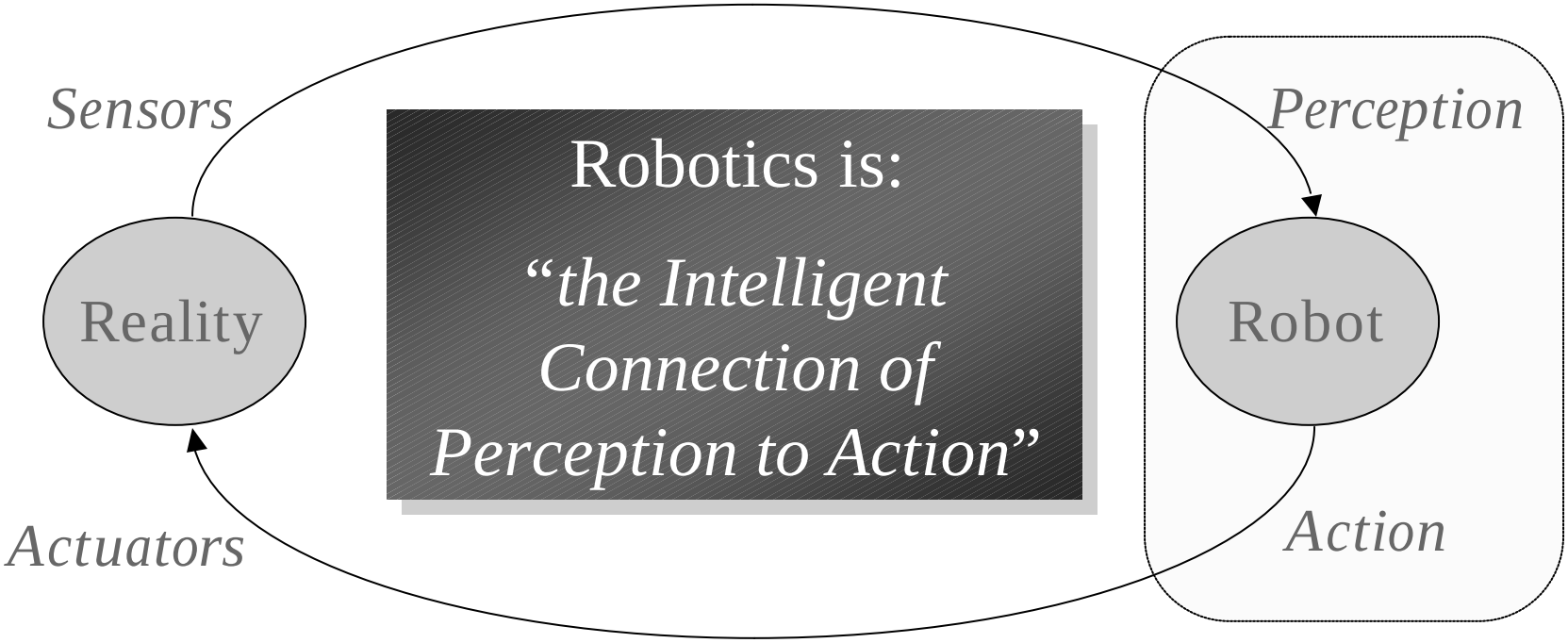


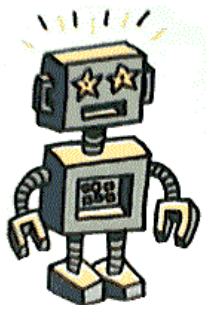
UNIVERSITY *of* LIMERICK
OLLSCOIL LUIMNIGH
LIMERICK , IRELAND





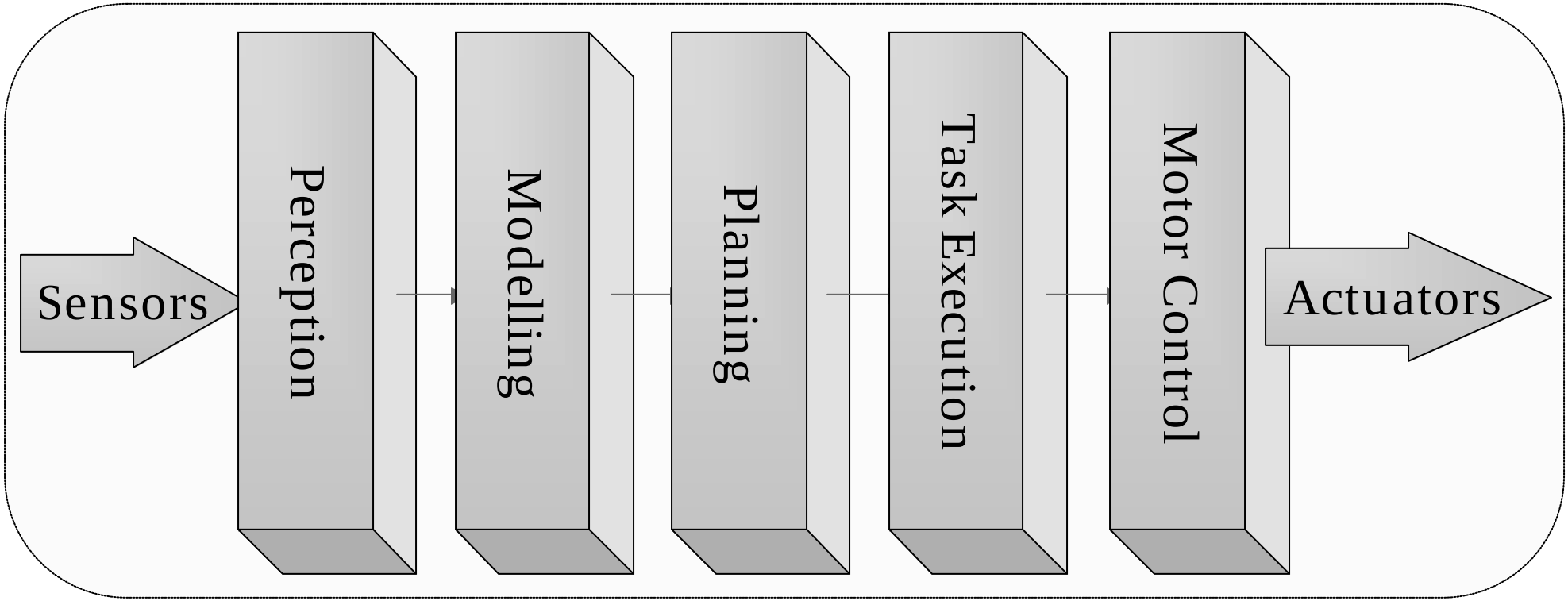
Definition of Robotics

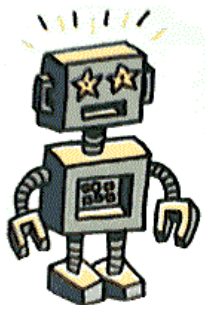




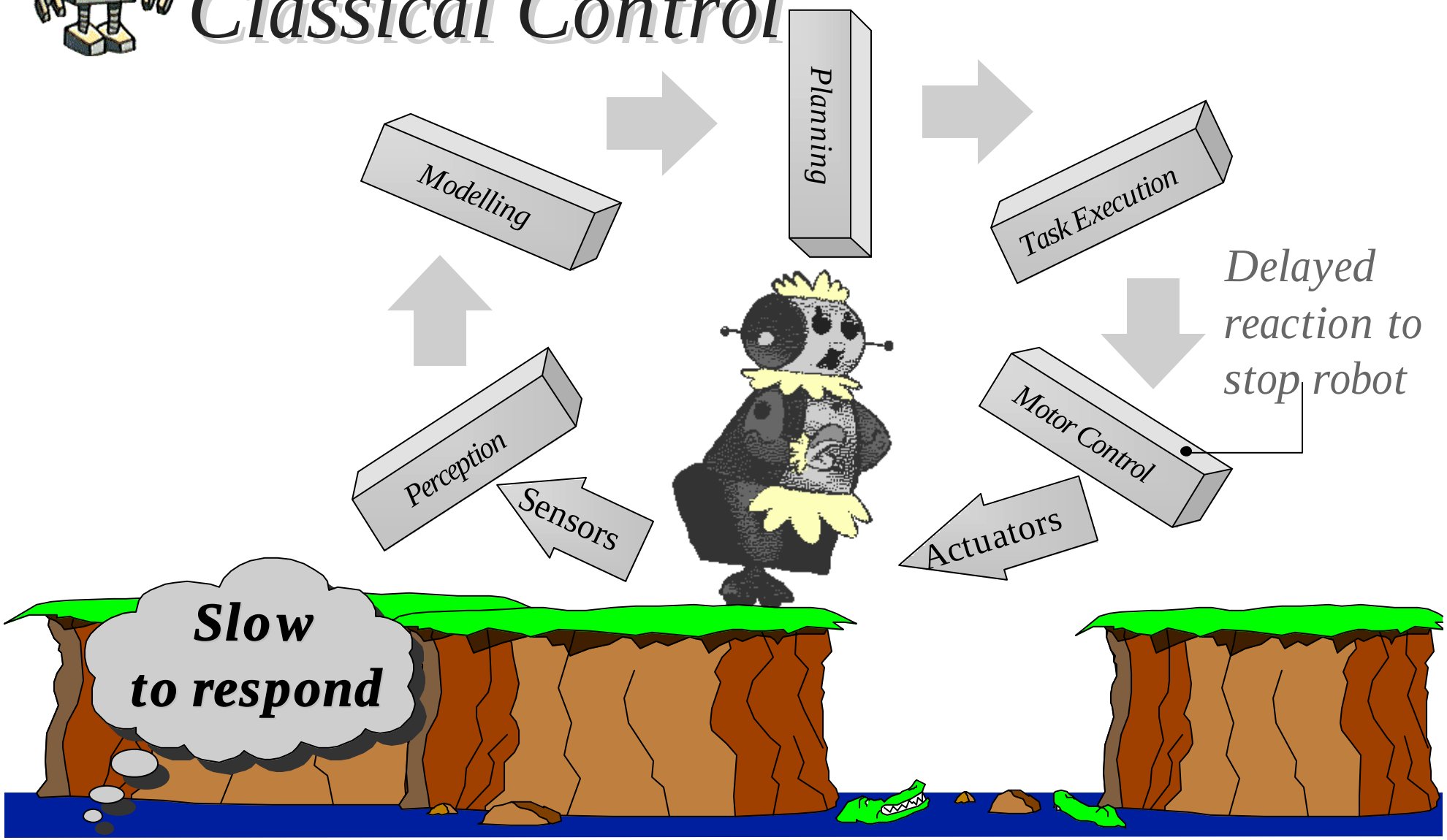
Classical Robotic Control

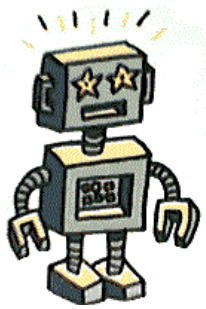
"Pipelined" approach





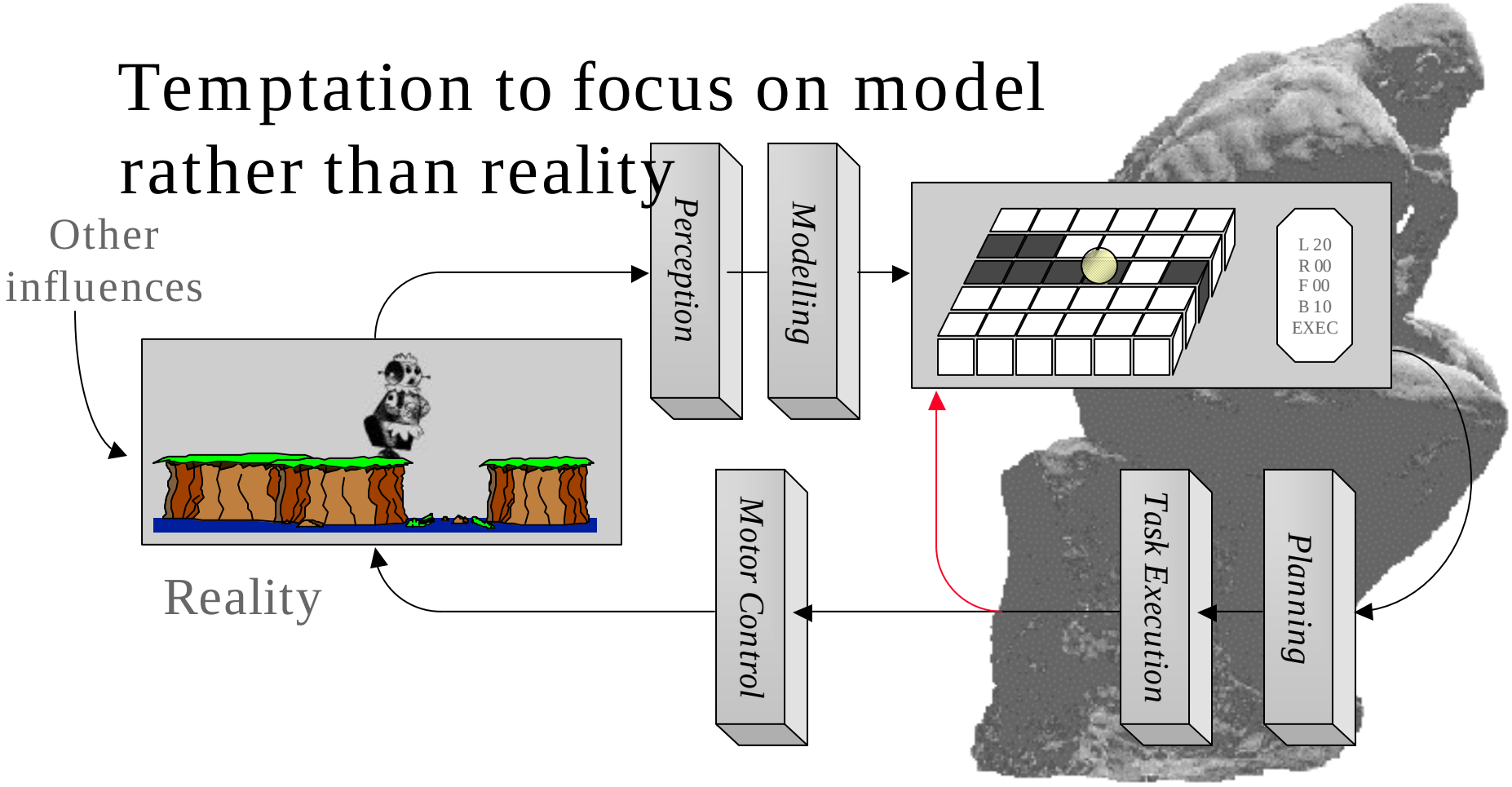
Practical Problems with Classical Control

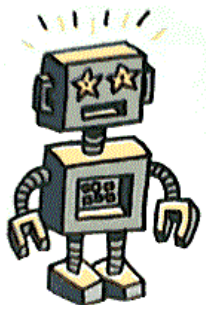




Research Problems with Classical Control

Temptation to focus on model rather than reality

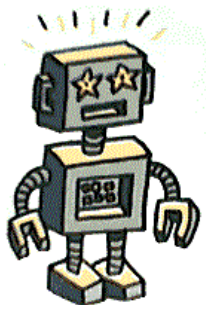




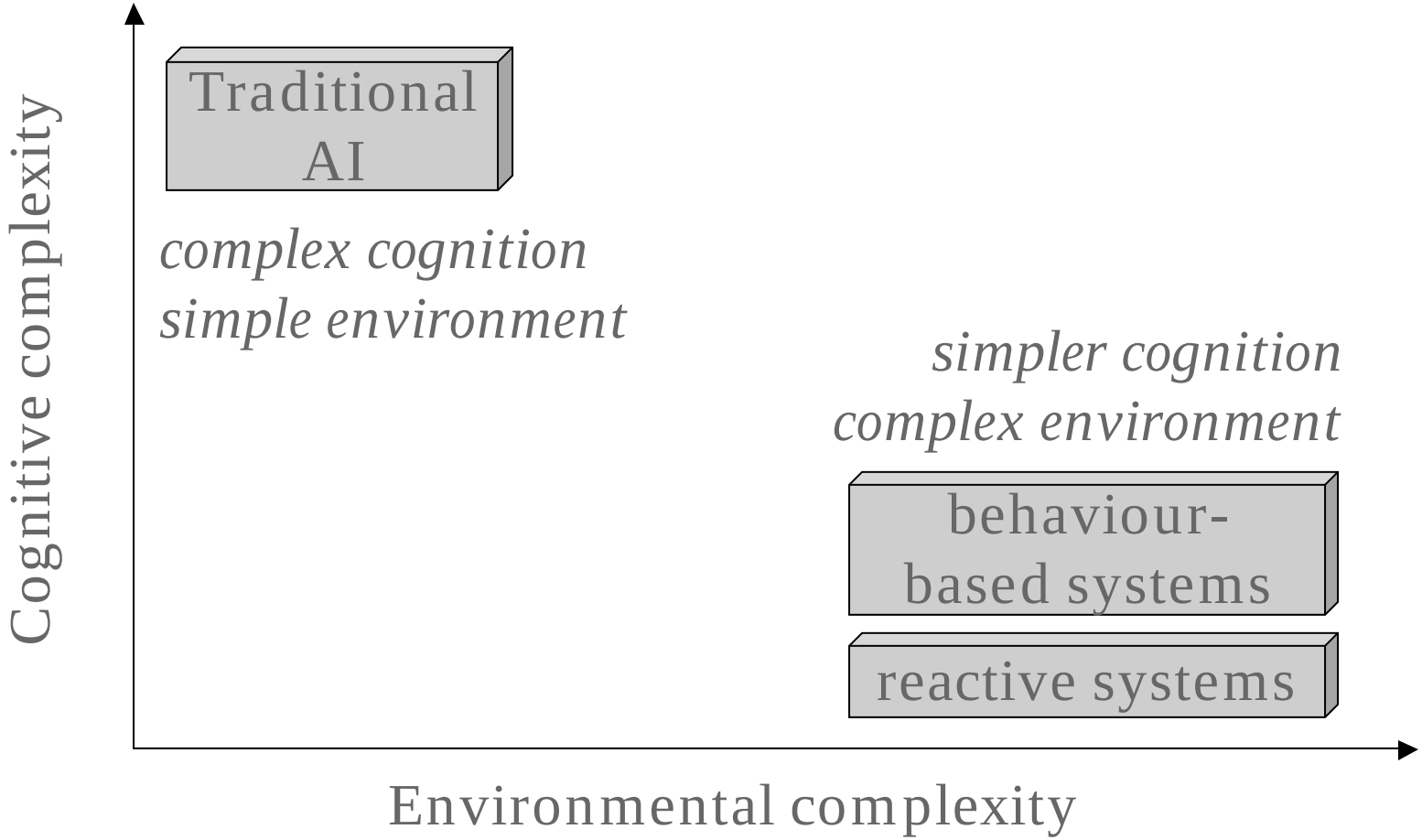
Classical Control: Summary

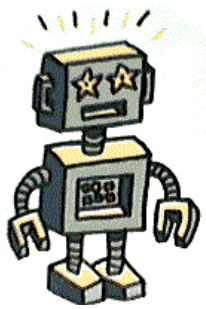
Assumes a complete internal world model can be built, then manipulated

- Slow. Modelling must occur before robot can react to changes in environment
- Emphasis on using model is misleading-
  - Makes complex tasks seem solvable by directing attention away from perception
  - Makes potentially simple tasks complicated
- Requires crippling simplications



Alternatives to Classical Control

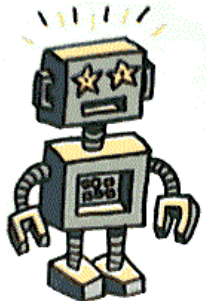




Behaviour-Based Control

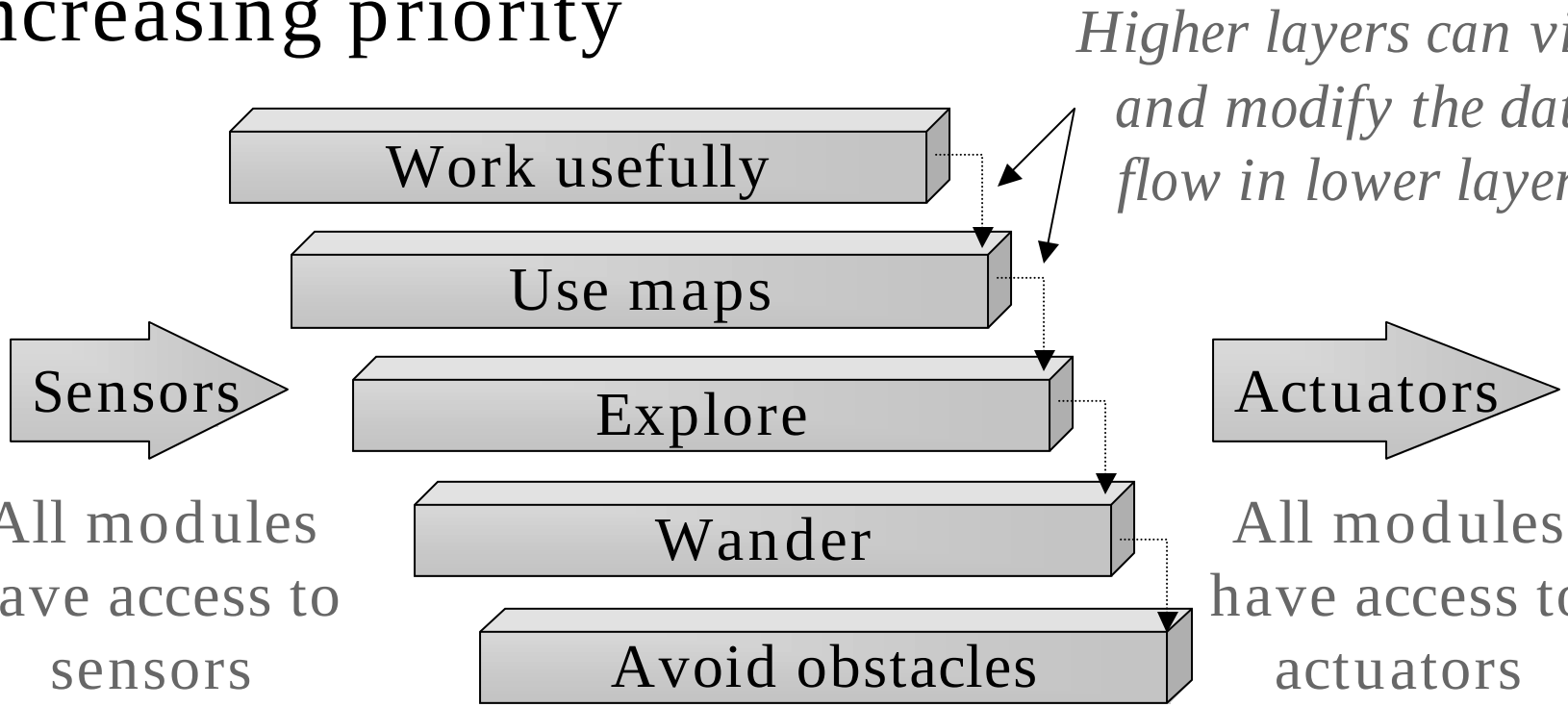
Rather than functional decomposition, use “Behaviours”

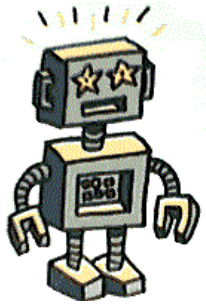
- As much as possible, *behaviours interact with each other through the environment, not the system*
- The world is its own best model, so consult it directly whenever practical
- Use distributed representations tailored to the particular behaviours using them



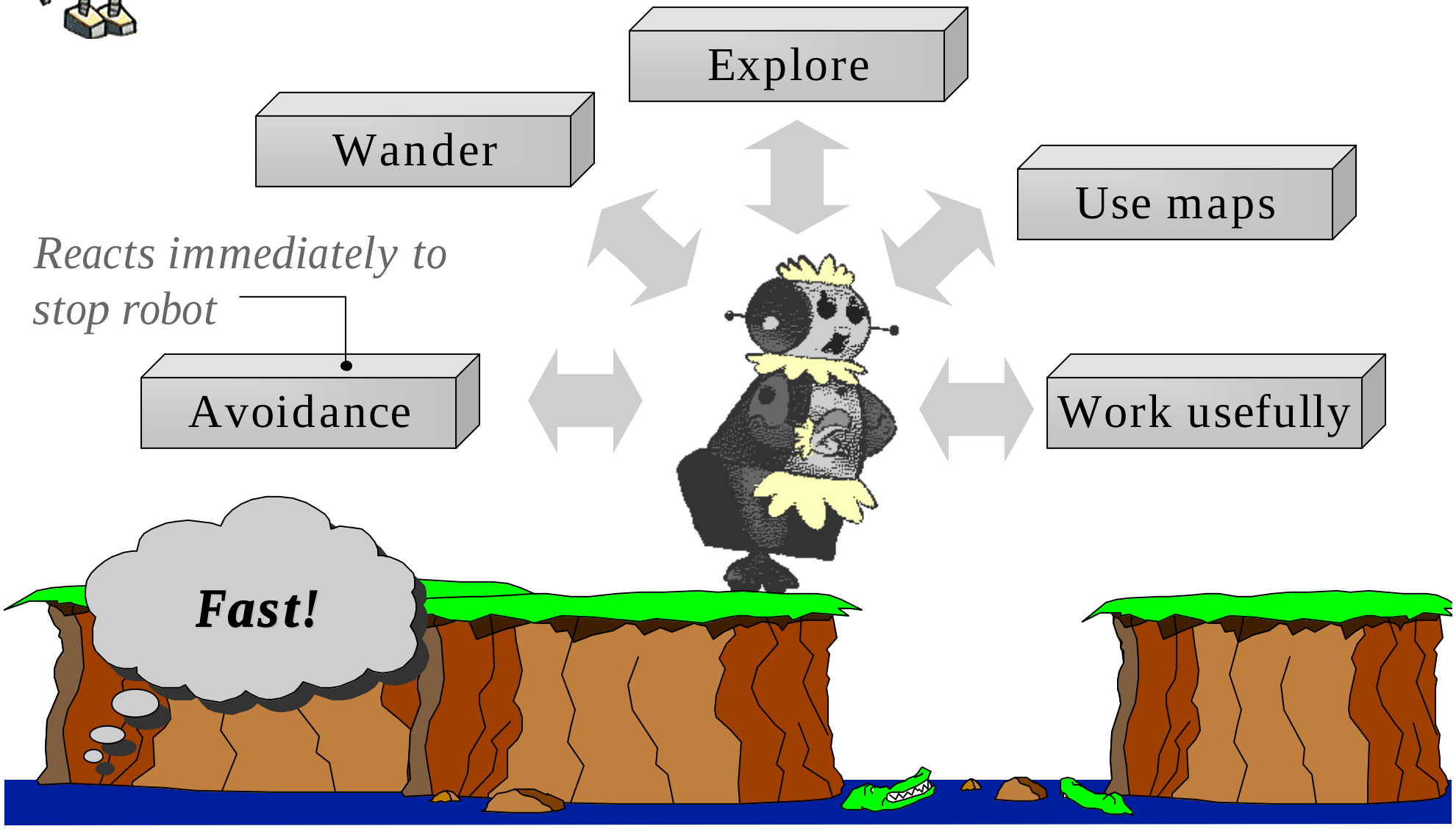
Behaviour-Based Control: Subsumption Architecture

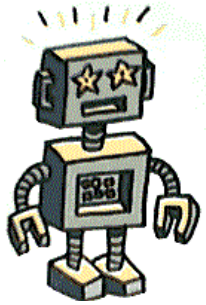
Modules act in parallel, in layers of increasing priority





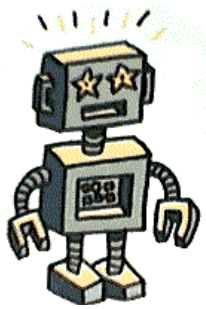
Advantages of Subsumption





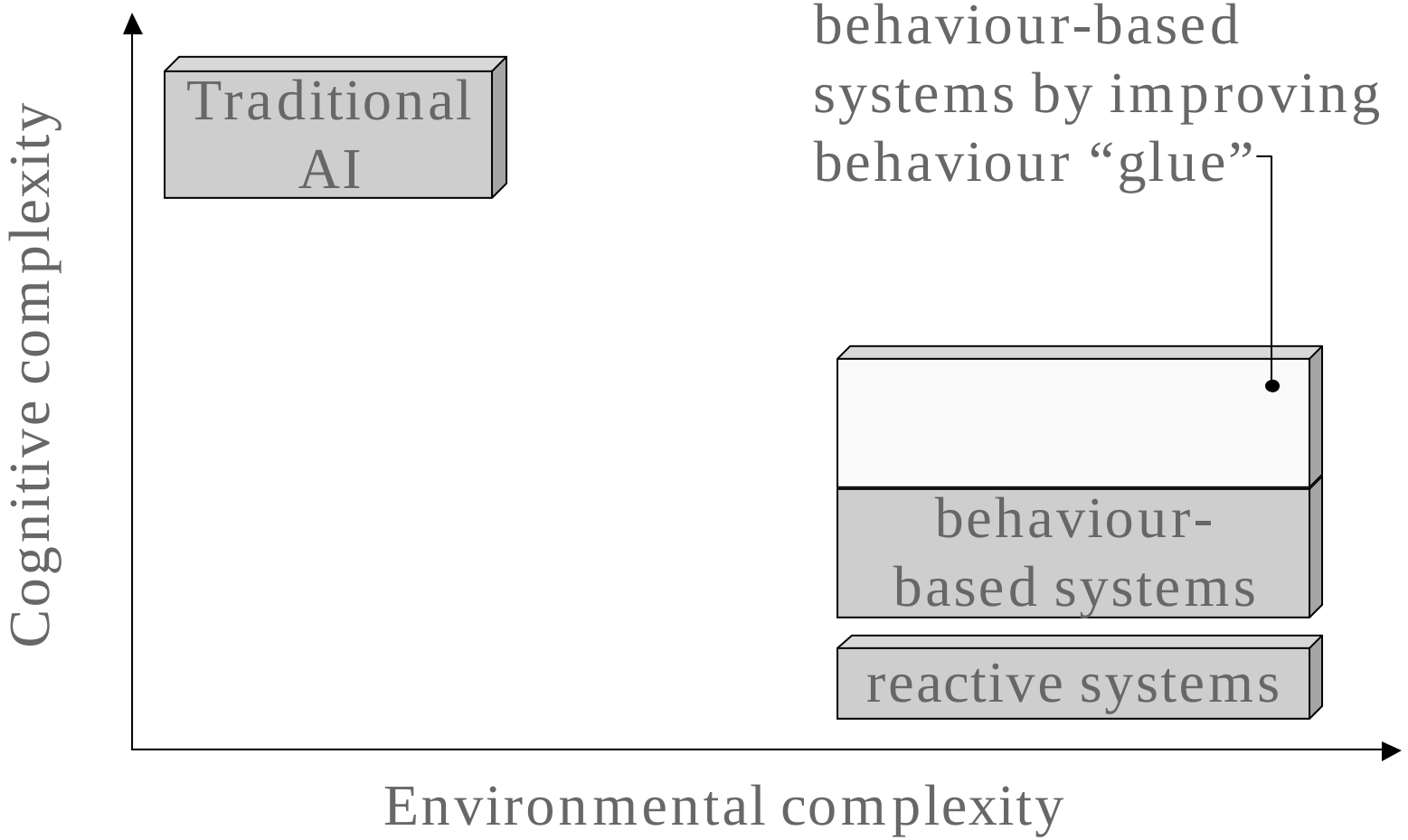
Disadvantages of Subsumption

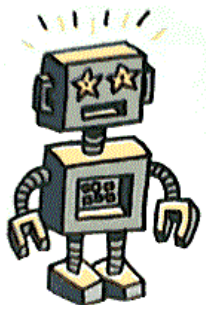
- Rigid priority scheme and strict layering are limiting -
  - Priorities must be evaluated and hardwired at design time
  - Requires behaviours to fit into a simple single-inheritance hierarchy
  - Behaviours cannot be combined, only enhanced linearly



Alternatives?

Extend scope of behaviour-based systems by improving behaviour “glue”





"Lateral" Architecture

Priority flows between behaviours as they make use of each other

Work usefully

Use maps

Sensors

Explore

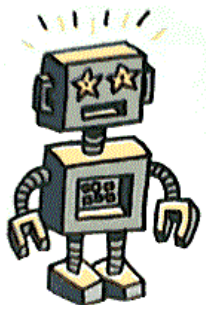
Actuators

Wander

Avoid obstacles

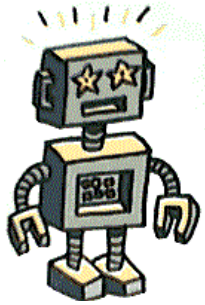
All modules have access to sensors

All modules have access to actuators

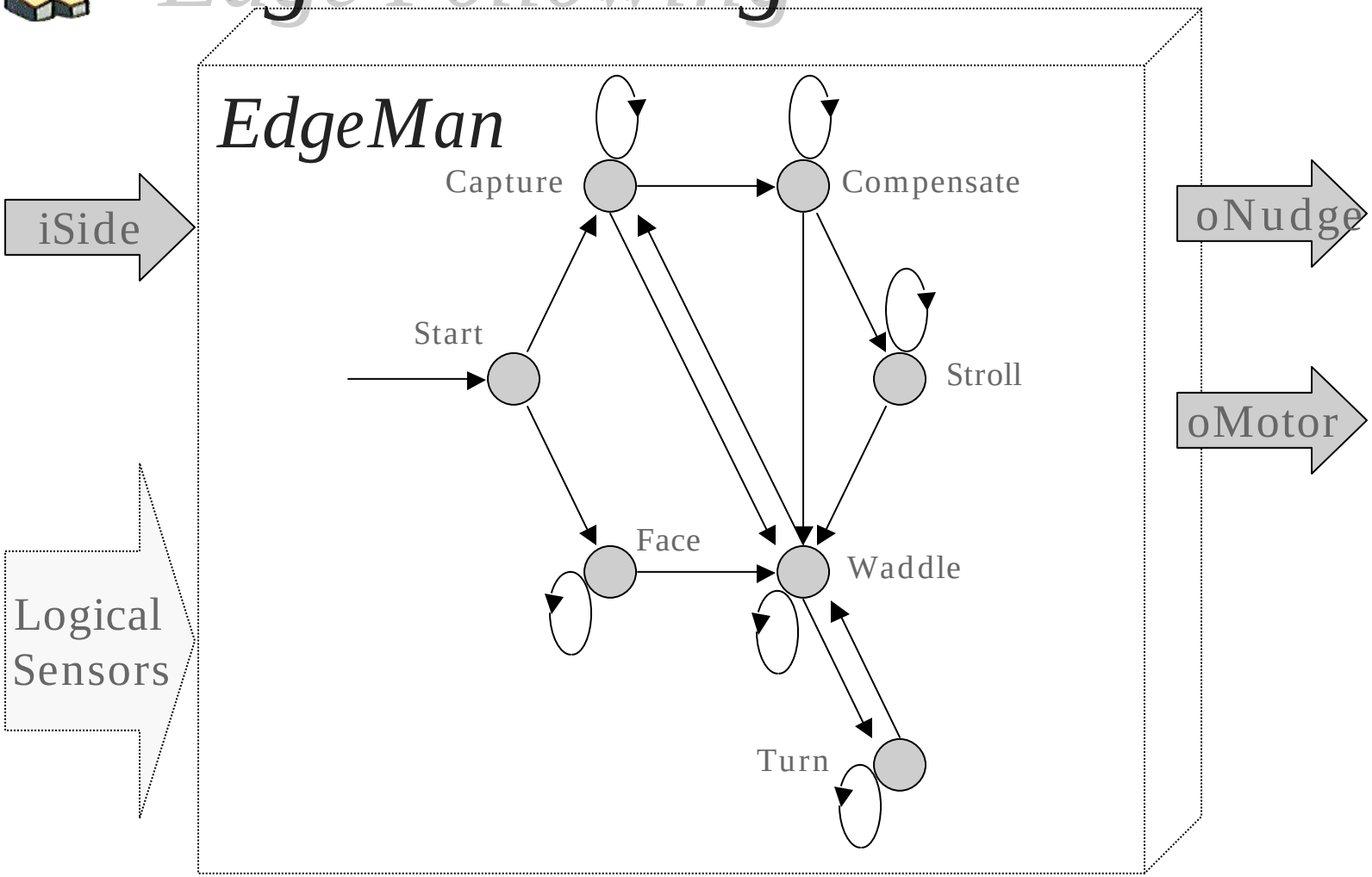


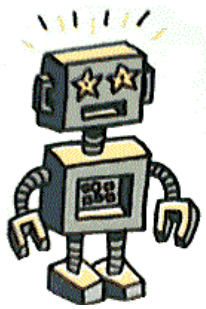
Features of Lateral

- Lateral has a dynamic priority system designed to make it easy to build behaviours using other behaviours
- A behaviour is given the priority its highest priority user at a given time thinks it should have (“sponsorship”)
- Behaviours expose a limited public interface to users, rather than allowing access to their internal data flows.



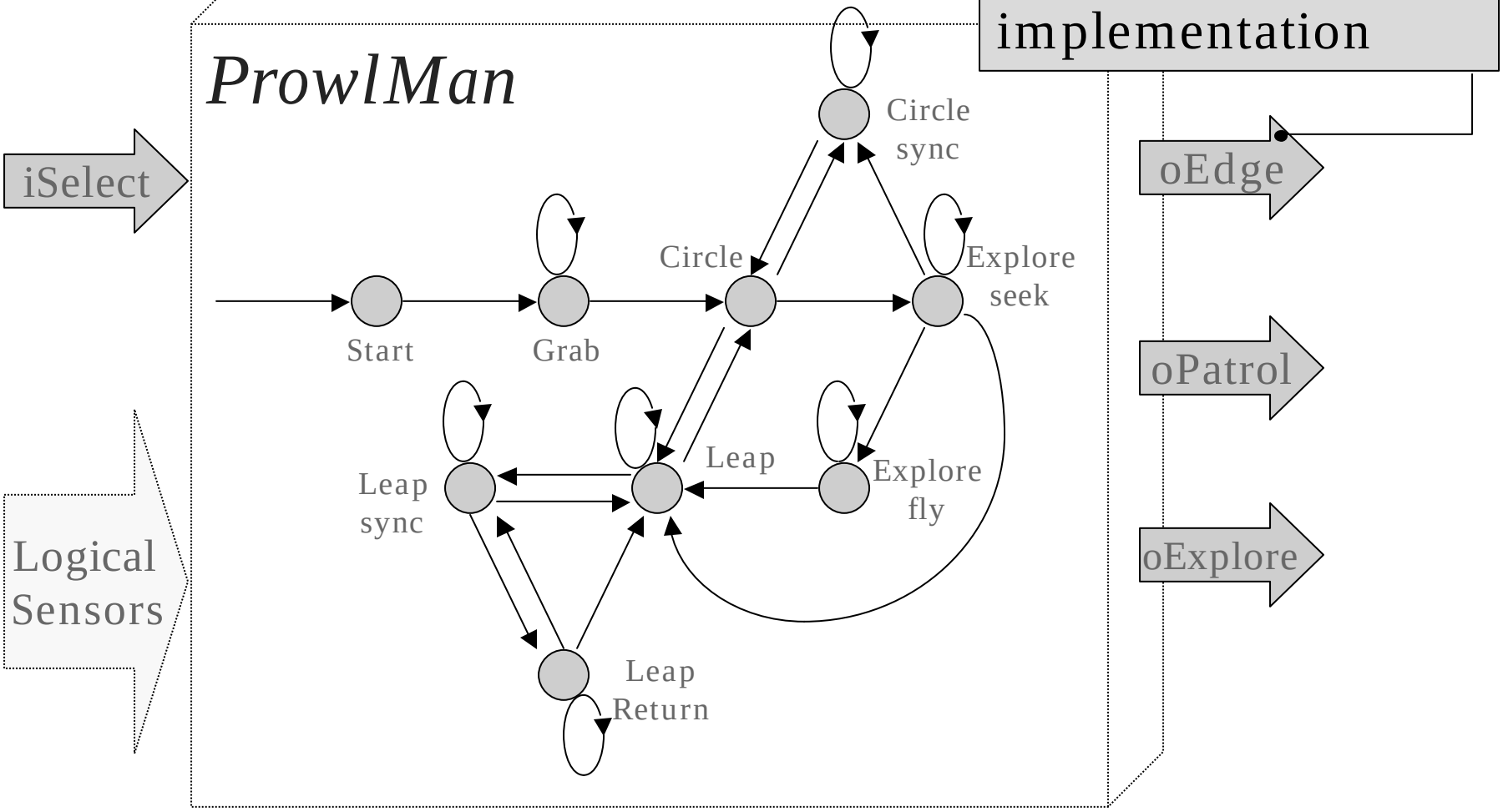
Example Behaviour: Edge Following

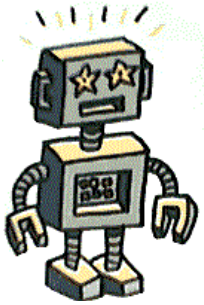




Prowling Behaviour

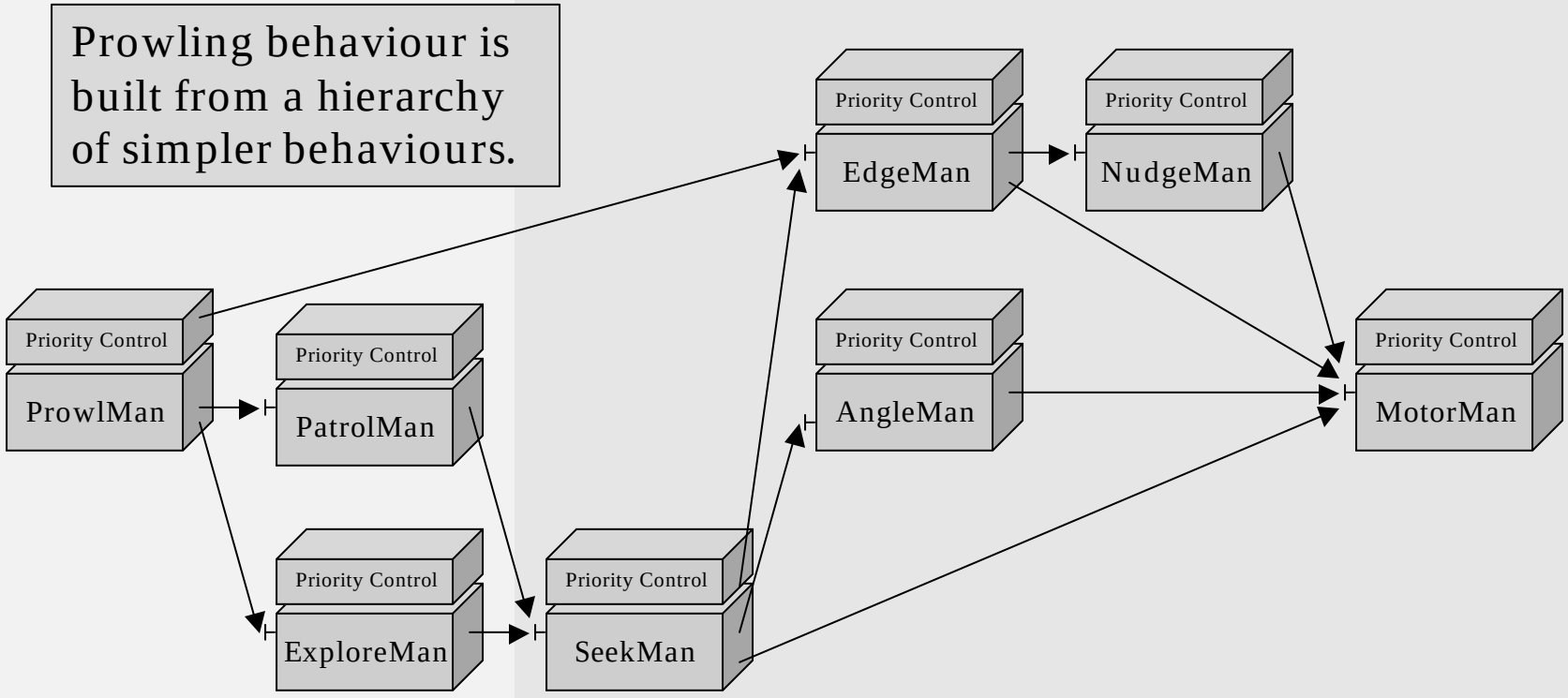
Prowling uses edge following in its implementation

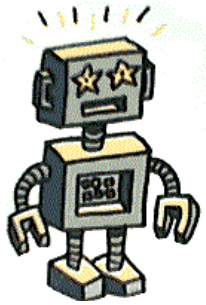




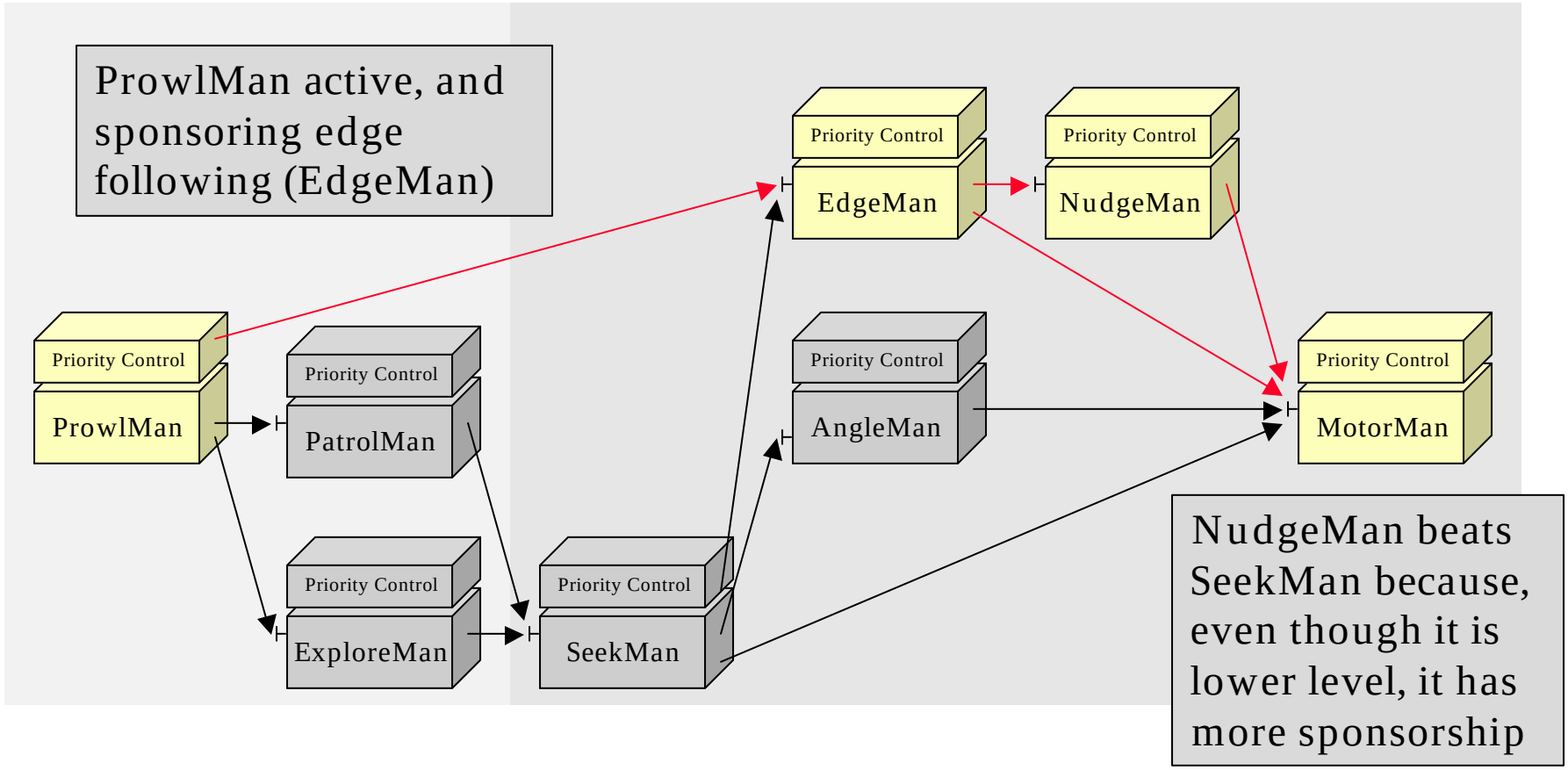
Prowling - Behaviours used

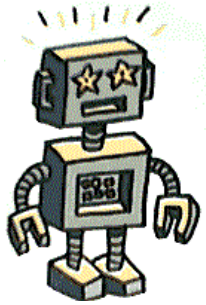
Prowling behaviour is built from a hierarchy of simpler behaviours.



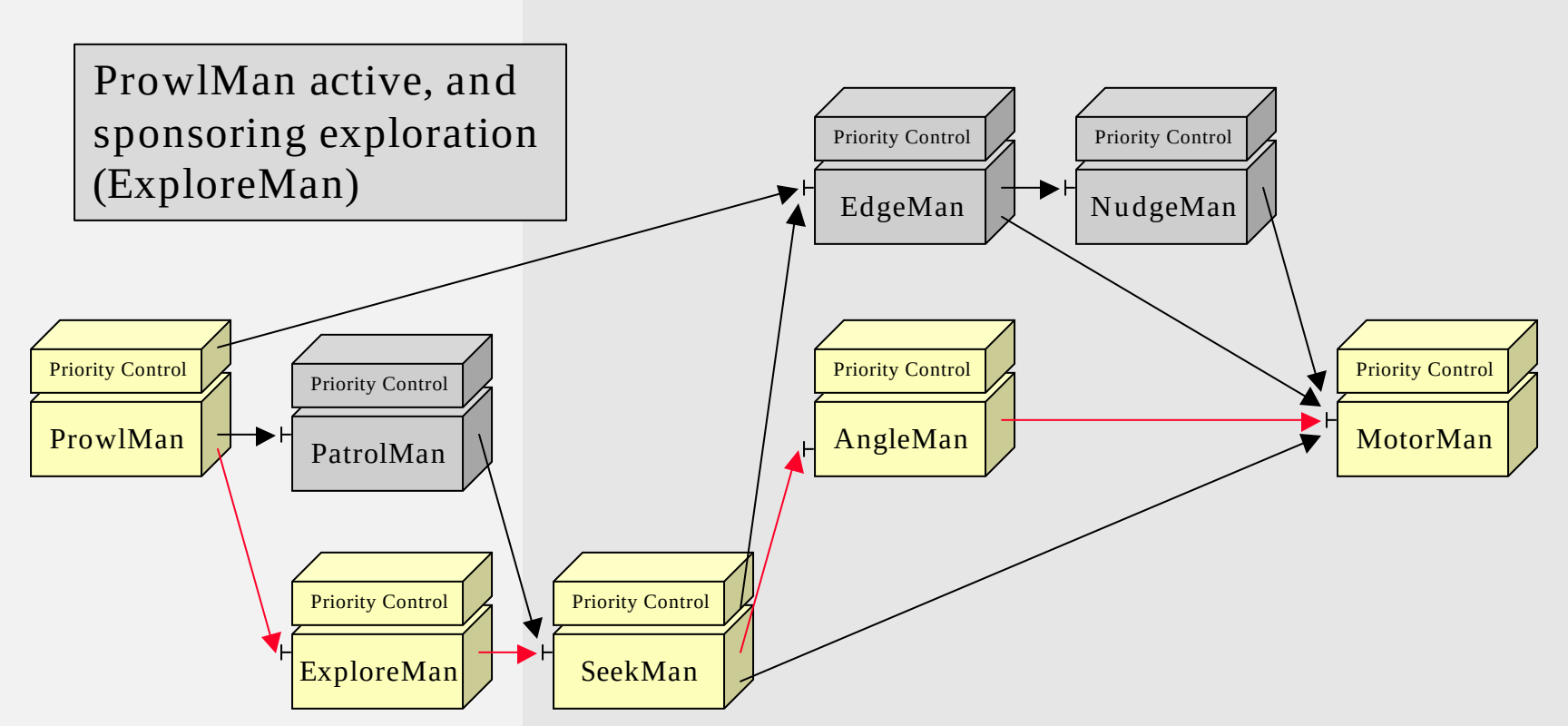


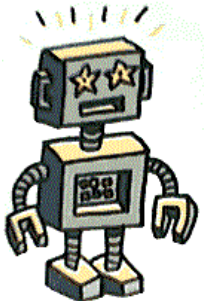
Prowling - Edge Following



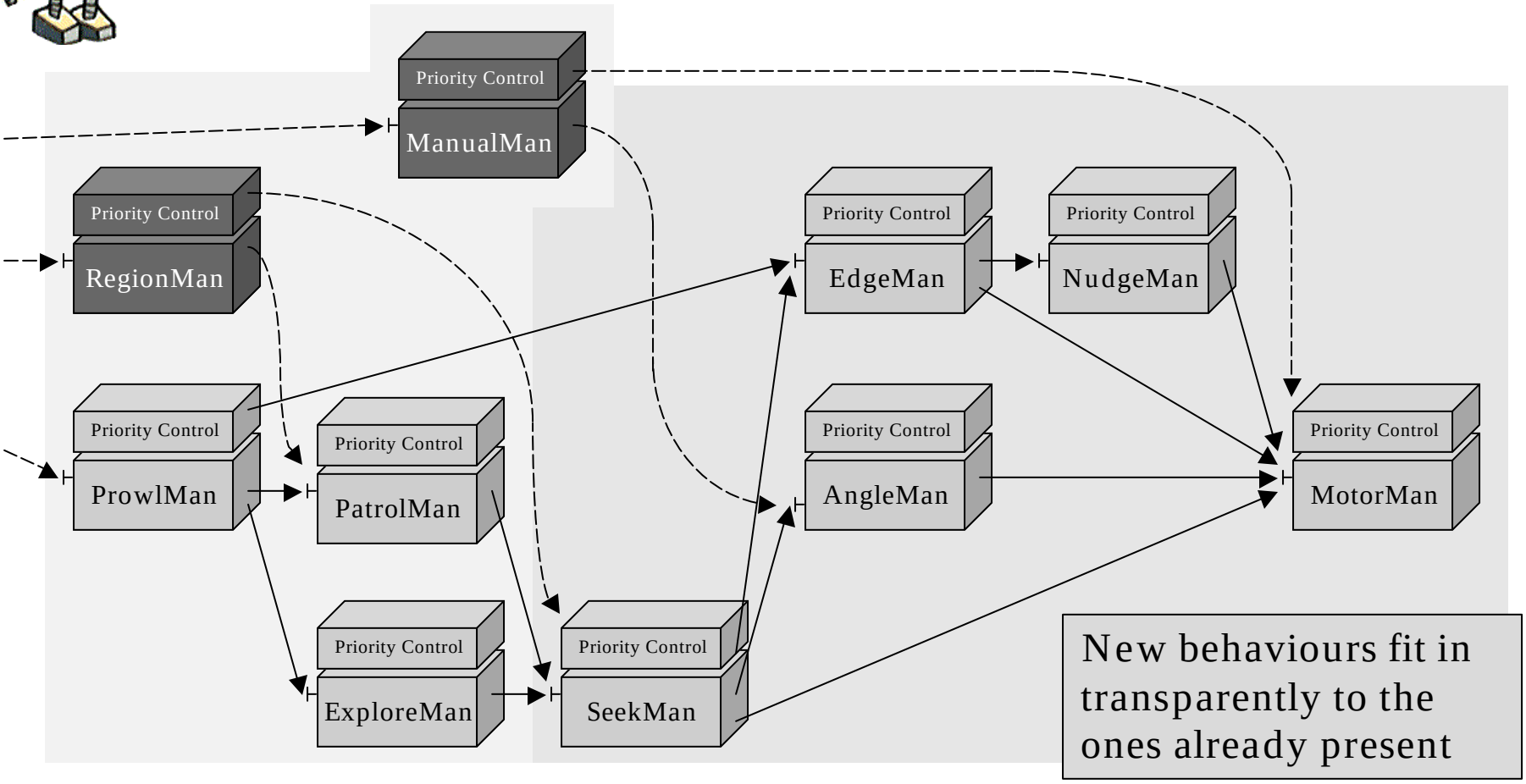


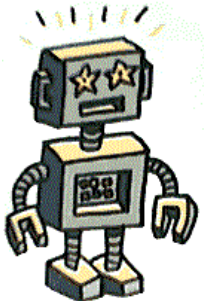
Prowling - Exploring



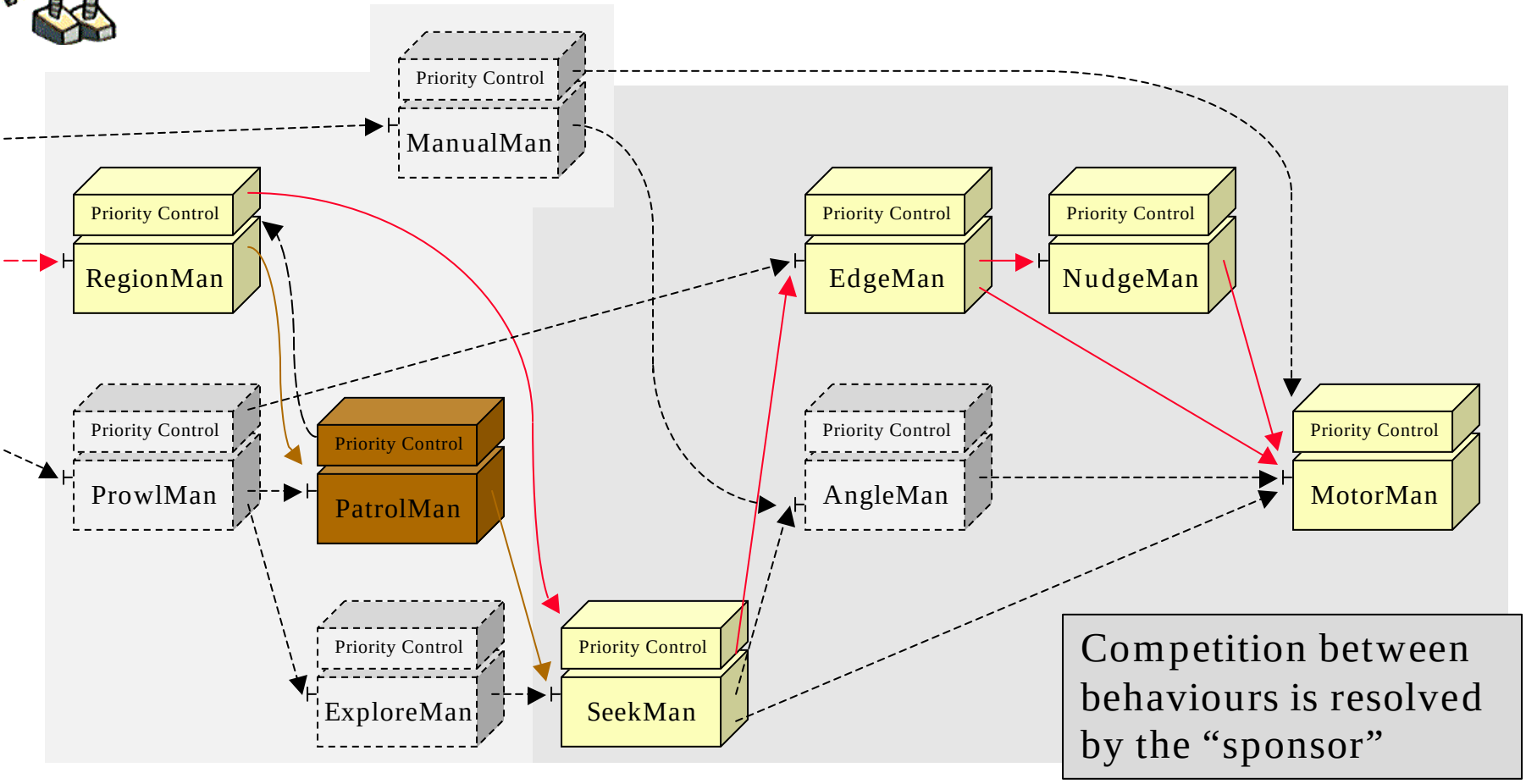


More Behaviours

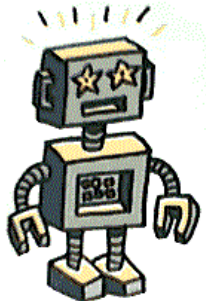




Competition

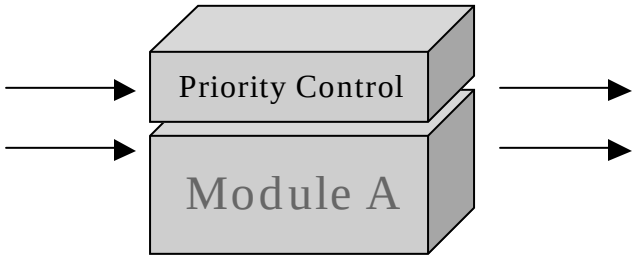


Competition between behaviours is resolved by the “sponsor”



Lateral - Implementation

Extended syntax for expressing parallel processes (as state machines) and data flow between them

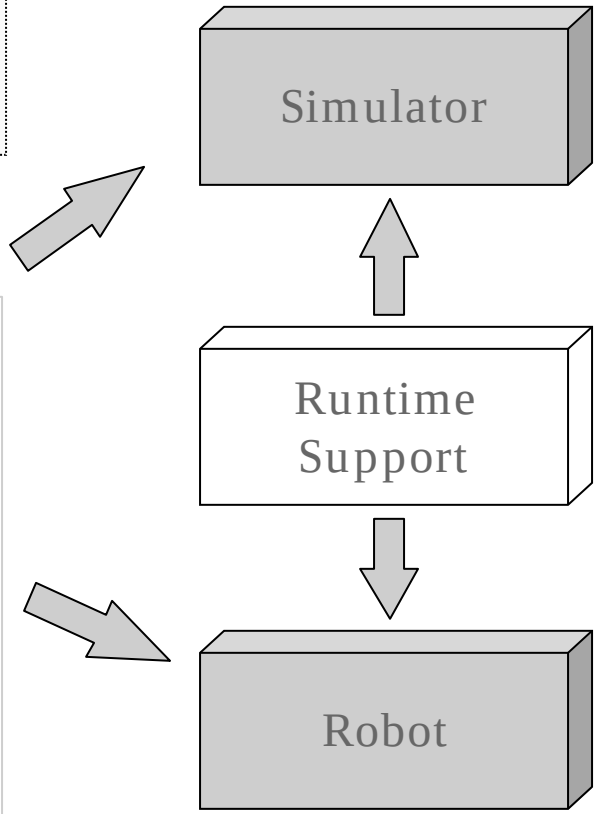


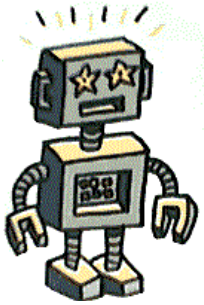

```
PROCESS ModuleA
{
  int localVariable;
  INPUT(int, boredom);
  INPUT(Point, distance);
  OUTPUT(MotorCtrl, motor);
  OUTPUT(Message, msg);
    @CONTROL
      // Priority Control
    @state1
      DoSomething();
      NEXT state2;
    @state2
      DoSomethingElse();
      NEXT @;
};
```

Lateral

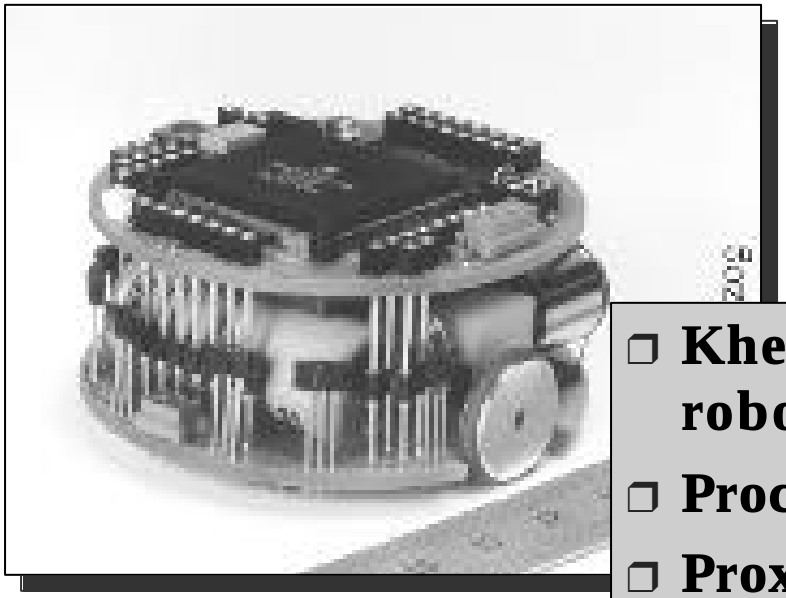
```
class ModuleA : public ZACProcess
{
private:
  int localVariable;
public:
  ZACInput<int> boredom;
  ZACInput<Point> distance;
  ZACOutput<MotorCtrl> motor;
  ZACOutput<Message> msg;

  int ZAC_Run ( int ZAC_state );
  ...
};

...
```

C++

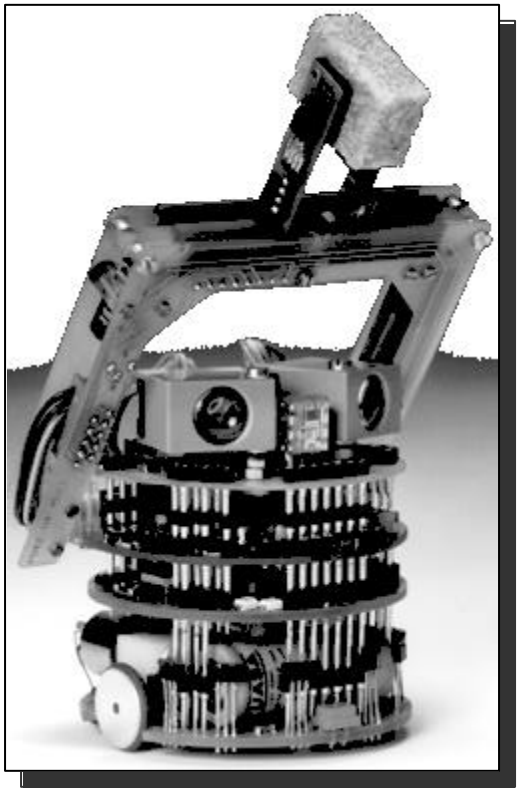


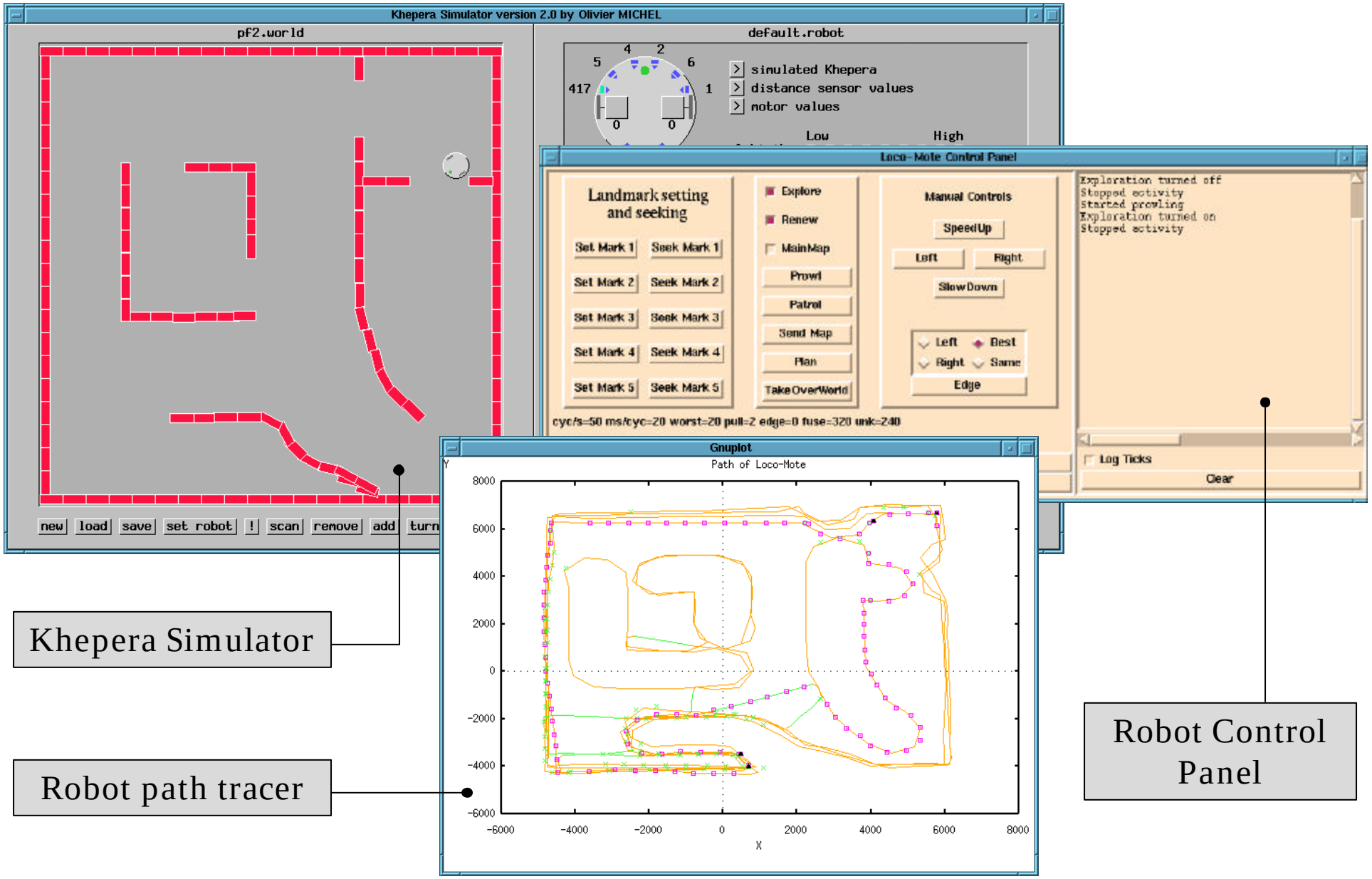


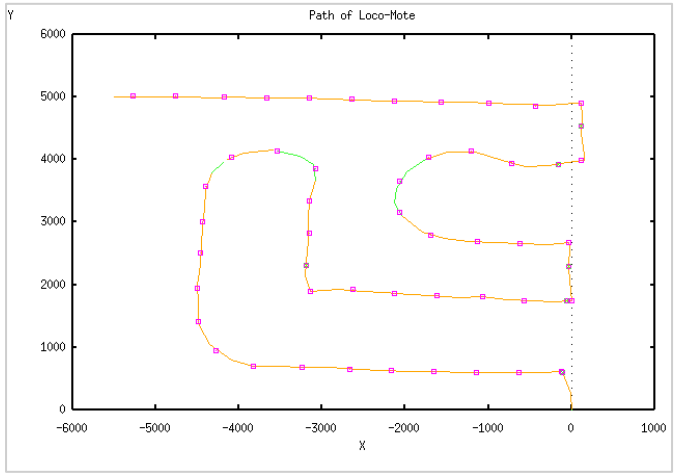
Hardware: “ Khepera® ”

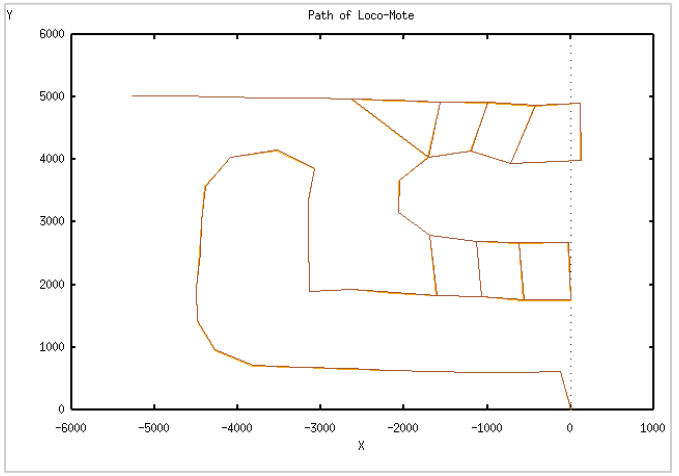


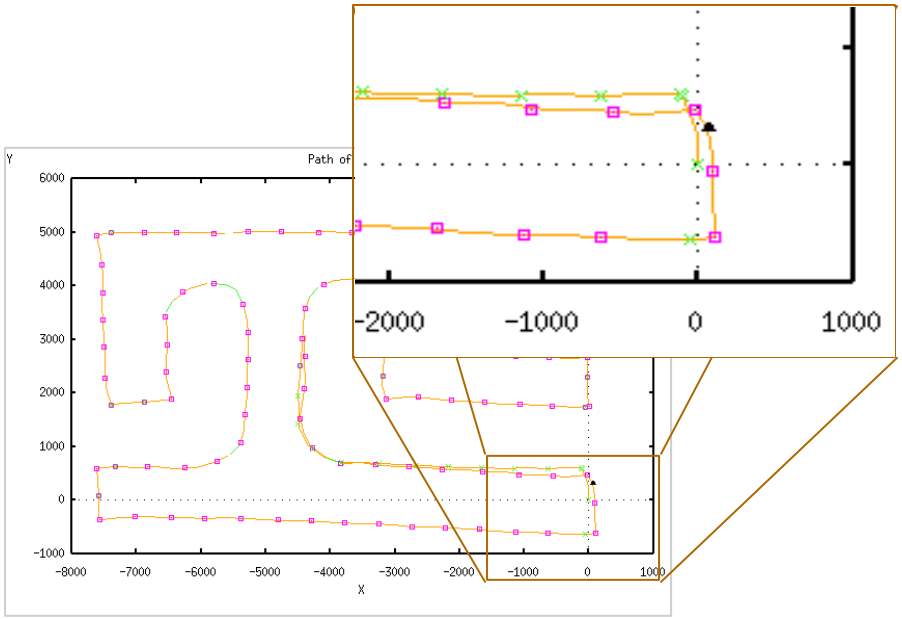
- Khepera miniature robot
- Processor: 68332
- Proximity sensors
- Light sensors
- Stepper motors

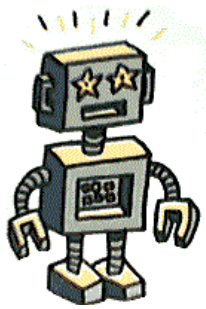



Khepera Simulator version 2.0 by Olivier MICHEL
pf2.world
default.robot
simulated Khepera
distance sensor values
motor values
Low
High
new
load
save
set robot
!
scan
remove
add
Loco-Mote Control Panel
Landmark setting and seeking
Set Mark 1
Seek Mark 1
Set Mark 2
Seek Mark 2
Set Mark 3
Seek Mark 3
Set Mark 4
Seek Mark 4
Set Mark 5
Seek Mark 5
Explore
Renew
MainMap
Prowl
Patrol
Send Map
Plan
TakeOverWorld
Manual Controls
SpeedUp
Left
Right
SlowDown
Left
Best
Right
Same
Edge
Exploration turned off
Stopped activity
Started prowling
Exploration turned on
Stopped activity
Log Ticks
Clear
cyc/s=50 ms/cyc=20 worst=20 pull=2 edge=0 fuse=320 unk=240
Gnuplot
Path of Loco-Mote
Y
X
8000
6000
4000
2000
0
-2000
-4000
-6000
-6000
-4000
-2000
0
2000
4000
6000
8000
Khepera Simulator
Robot path tracer
Robot Control Panel


Path of Loco-Mote
Y
X
6000
5000
4000
3000
2000
1000
0
-6000
-5000
-4000
-3000
-2000
-1000
0
1000


Path of Loco-Mote
Y
X
6000
5000
4000
3000
2000
1000
0
-6000
-5000
-4000
-3000
-2000
-1000
0
1000


Path of
Y
X
6000
5000
4000
3000
2000
1000
0
-1000
-8000
-7000
-6000
-5000
-4000
-3000
-2000
-1000
0
1000
-2000
-1000
0
1000



Priority Implementation: Connections