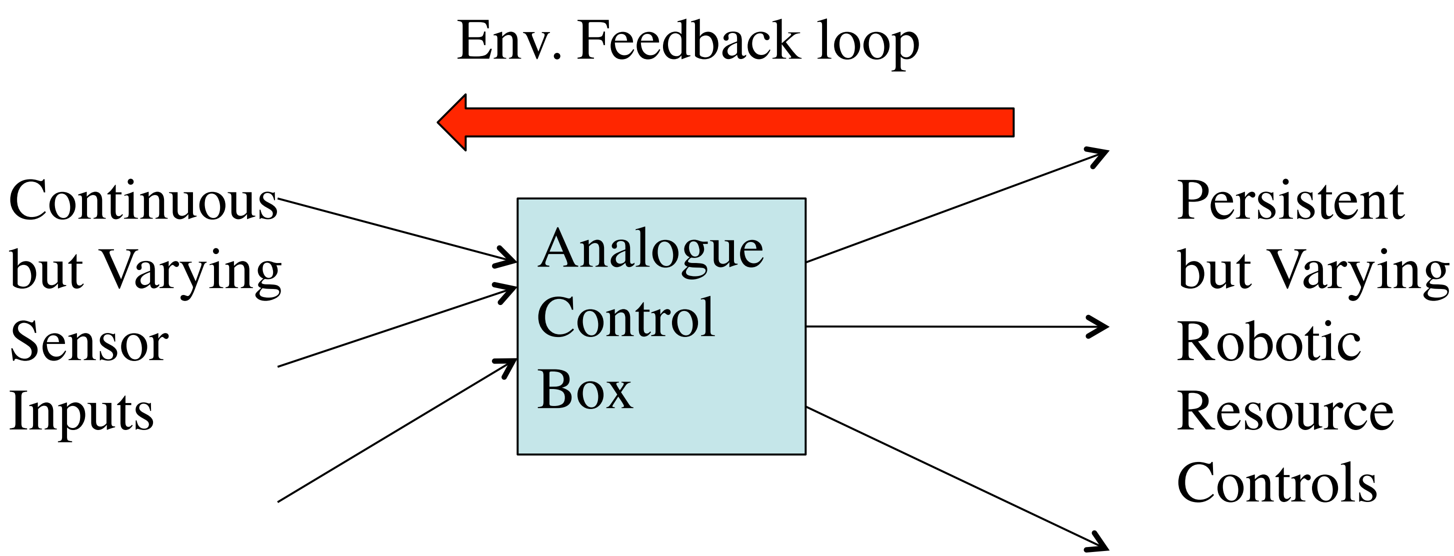
Programming Robotic Agents

A Multi-Tasking Teleo-Reactive Approach

Keith Clark
Imperial College London &
University of Queensland
joint work with
Peter Robinson
University of Queensland

Nilsson's Teleo-Reactive procedures - inspiration



1952 Ashby, *Design for a Brain*, homeostasis concept
1960 Miller et al, *Plans and the Structure of Behavior*, TOTE
1970's Logic based cognitive robotics – SRI's Shakey, STRIPS
1990 Brooks, Subsumption architecture

Nilsson's Teleo-Reactive procedures - realisation

- **Sensor readings** are **rapidly changing percept** facts in *BeliefStore*
  see(bottle,left,30) holding
- **Control procedures**: each a **sequence** of *Guard* ~> *Action* rules
  next_to(bottle,centre,0) ~> close_gripper   *robotic action*
  see(bottle,_) ~> approach(bottle)   *call to a TR proc.*
- **Earliest rule with inferable guard** is **fired** (in each called proc.)
- **Teleo** (goal seeking) aspect
  - *Action* should achieve *Guard* of an **earlier** rule in the proc.
- **Reactive** aspect
  - **All** rule firings reconsidered on percepts update (**key feature**)
- TR programmed tasks **robust** and **opportunistic**
- **Percept querying** via **rule defined (view) relations**
- *BeliefStore* can include **declarative model** of environment

From reason based to sensor reactive behaviour

p(….){

H_1 ~> ()   *% H_1 is goal of p, when achieved do nothing*

H_2 ~> A_1, A_2 *% parallel actions should achieve H_1*

H_3 ~> q(…) → q(….){

…

H_i ~> q'(…)

…}

q(….){

I_1 ~> ()

I_2 ~> r(…) → r(….){

…}

r(….){

…

P ~> A_3

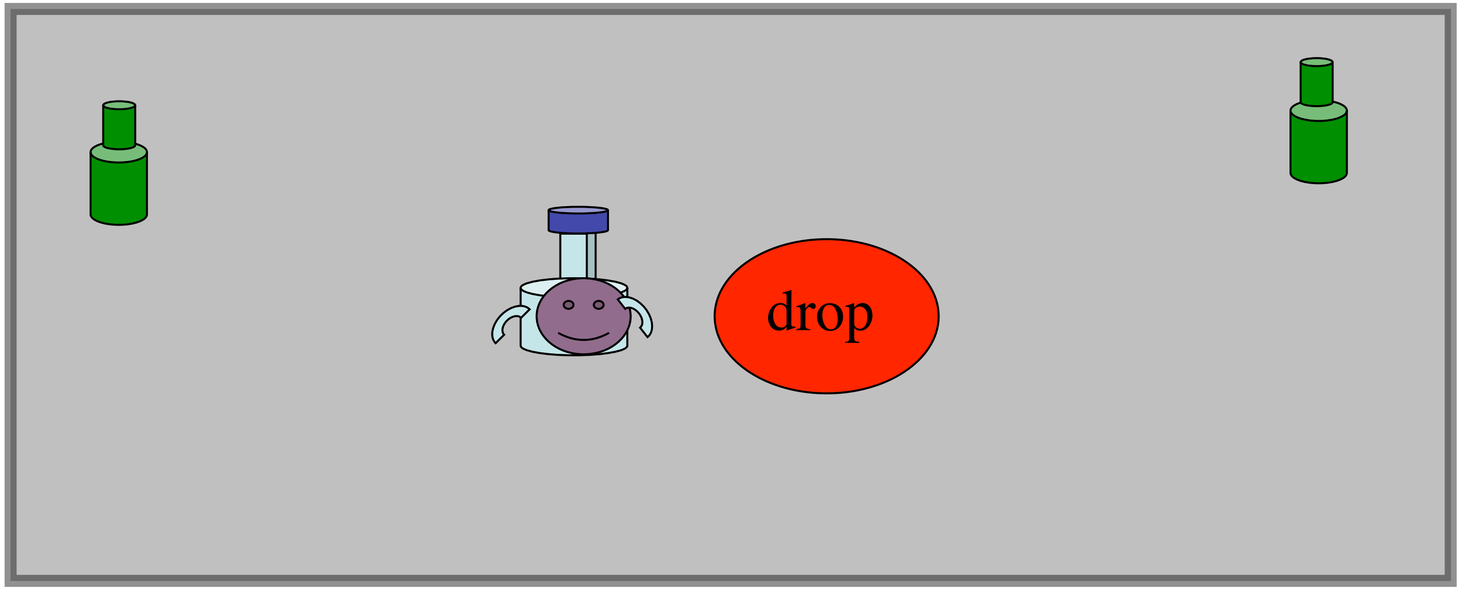
…}

Switches between diff. inter. level procs. as high level inferable beliefs change

Invokes a diff. low level proc. as inter. level inferable beliefs change

Introductory example: bottle collecting robotic agent

- Enclosed space with green bottles scattered about and a drop area painted red



- No other objects in the space
- Task is to find, grab and deliver a bottle to the red drop area
- Repeat behavior when bottle removed from drop

Top level control program

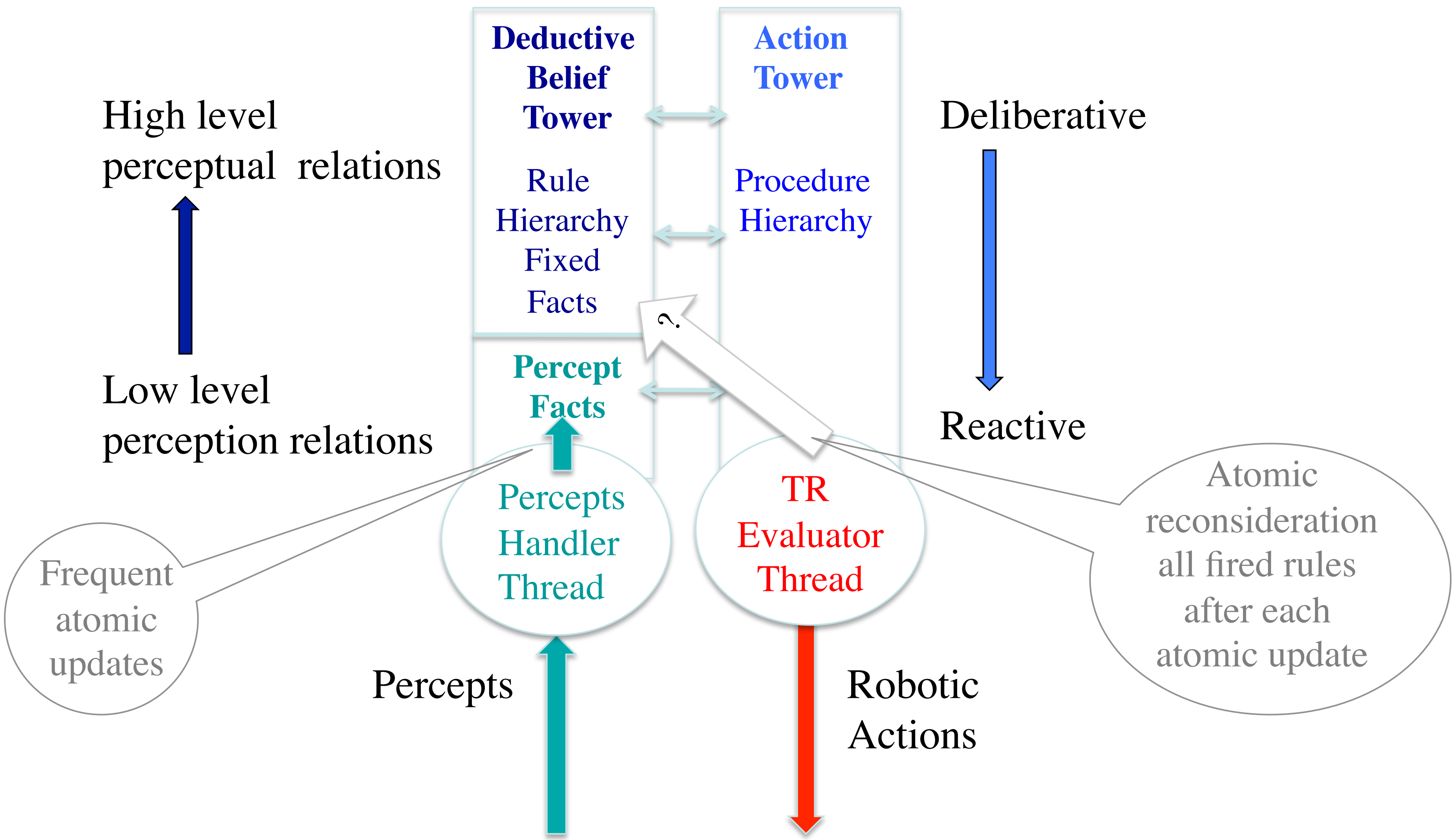
```
collect_bottle{        % goal to have delivered a bottle to drop area
   have_delivered                        ~> ()
   see(drop,_,0) & holding               ~> open_gripper
   holding                               ~> deliver_bottle
   true                                  ~> get_bottle
   }

have_delivered <= see(drop,_,0) & next_to_bottle(_) & gripper_open

next_to_bottle(Dir) <= see(bottle,Dir,Dist) & Dist≤2
```

Auxiliary procedures

```
deliver_bottle{                          % only active whilst holding true
  see(drop,_,0)                     ~> ()
  see(drop,_,Dist) & Dist>0         ~> approach(drop)
  true                              ~> turn(left, 0.5)
  }

approach(Th){  % goal to get closer to Th, calling proc. will terminate
   see(Th, center, Dist) ~> forward(calcSpeed(Th, Dist))
   see(Th, Dir, Dist) ~> forward(calcSpeed(Th, Dist)), turn(Dir,0.2)
   }
```

How can we be sure Sp will be a number and Dir an understood direction?

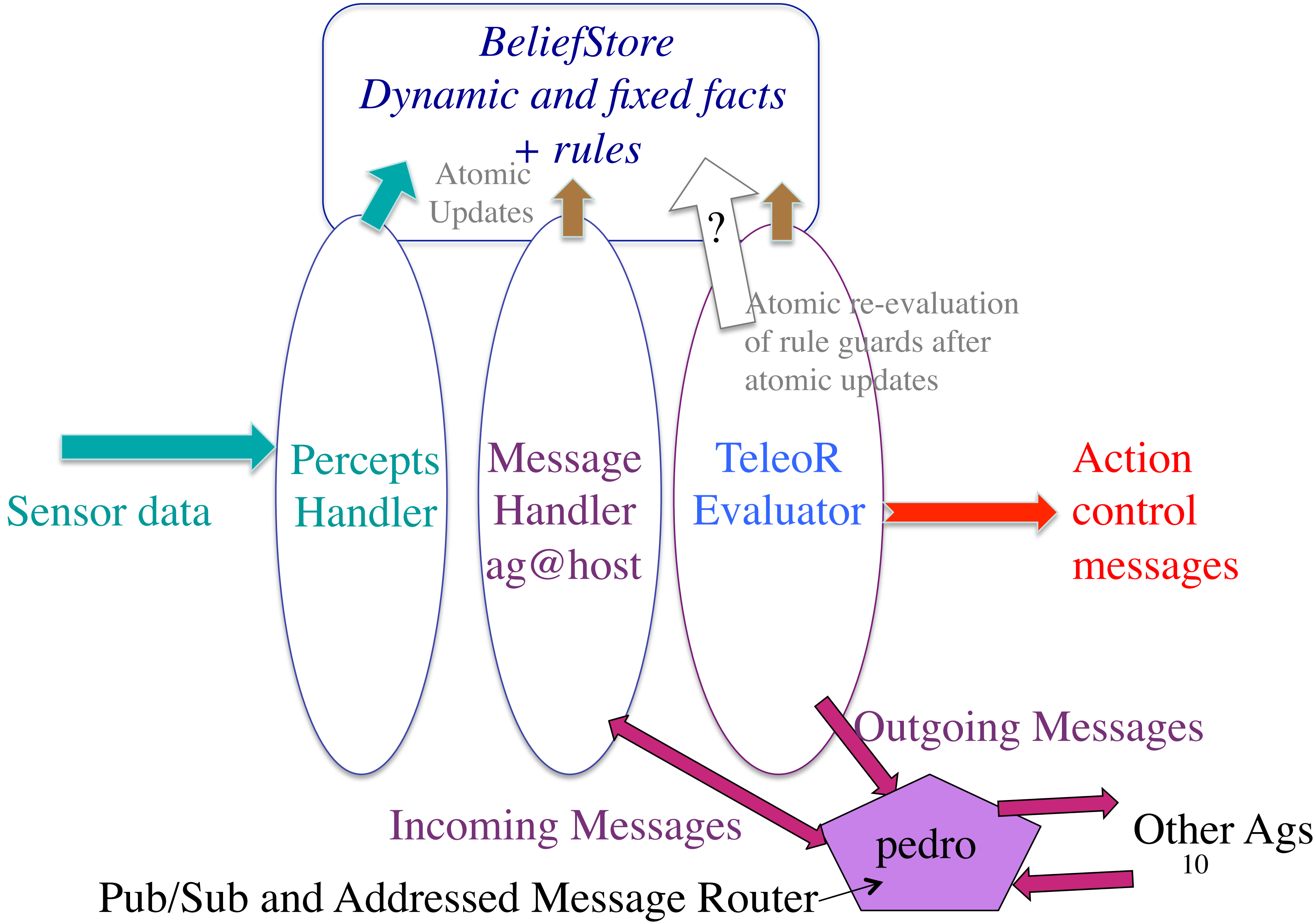
A TR Two Thread Agent Architecture



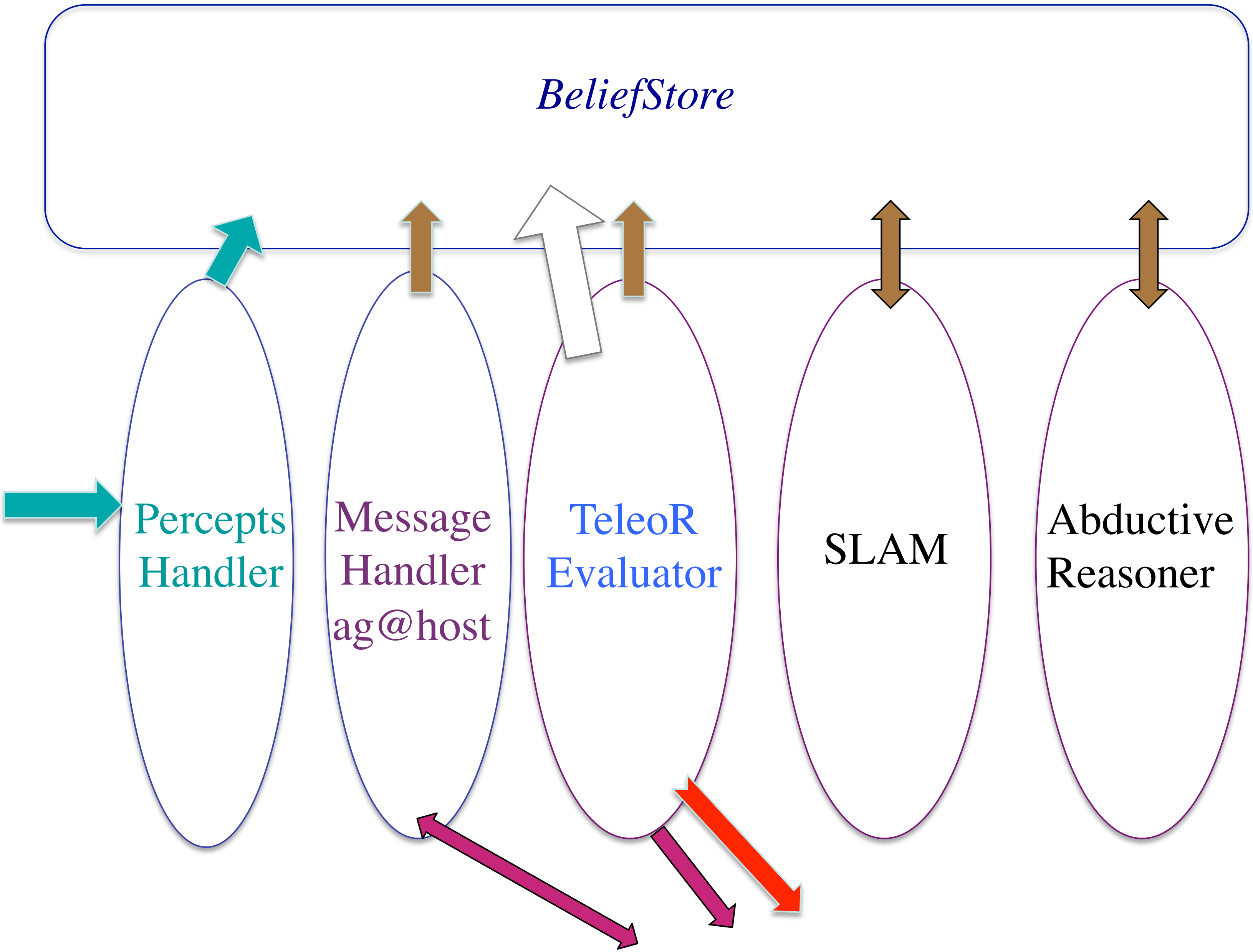
TeleoR: Extensions of TR for single task agents

- **Typed and moded** LP/FP *BeliefStore* language
  - Compile time abstract interpretation guarantees all actions will be fully instantiated and correctly typed
- **Extra forms of rules** inhibiting the firing of other rules *in the same proc*.
  - **while C** inhibits rules below
  - **until U** inhibits rules above and new instance re-firing
  - Echoes of Brook's behavior inhibition
- **Time sequence actions**
  - For when there is no sensor to determine when an action should terminate. Good for wander behavior.
- **Wait/repeat actions** for retrying actions that have failed to achieve intended effect by expected time
- ***BeliefStore* update + message send actions** as well as robotic actions

Three Thread **TeleoR** Agent Architecture



Extra threads for extra functionality



Example use of wait/repeat

```
collect_bottle{
    have_delivered                  ~> ()
    see(drop,_,0) & holding         ~> open_gripper wait 2 repeat 3
    ….
    }
```

- Suppose second rule is fired.
- If after 2 secs. no new rule is fired, open_gripper action restarted.
- 2 secs. after 3 such restart, system puts error fact in BS
- Stops trying restarts

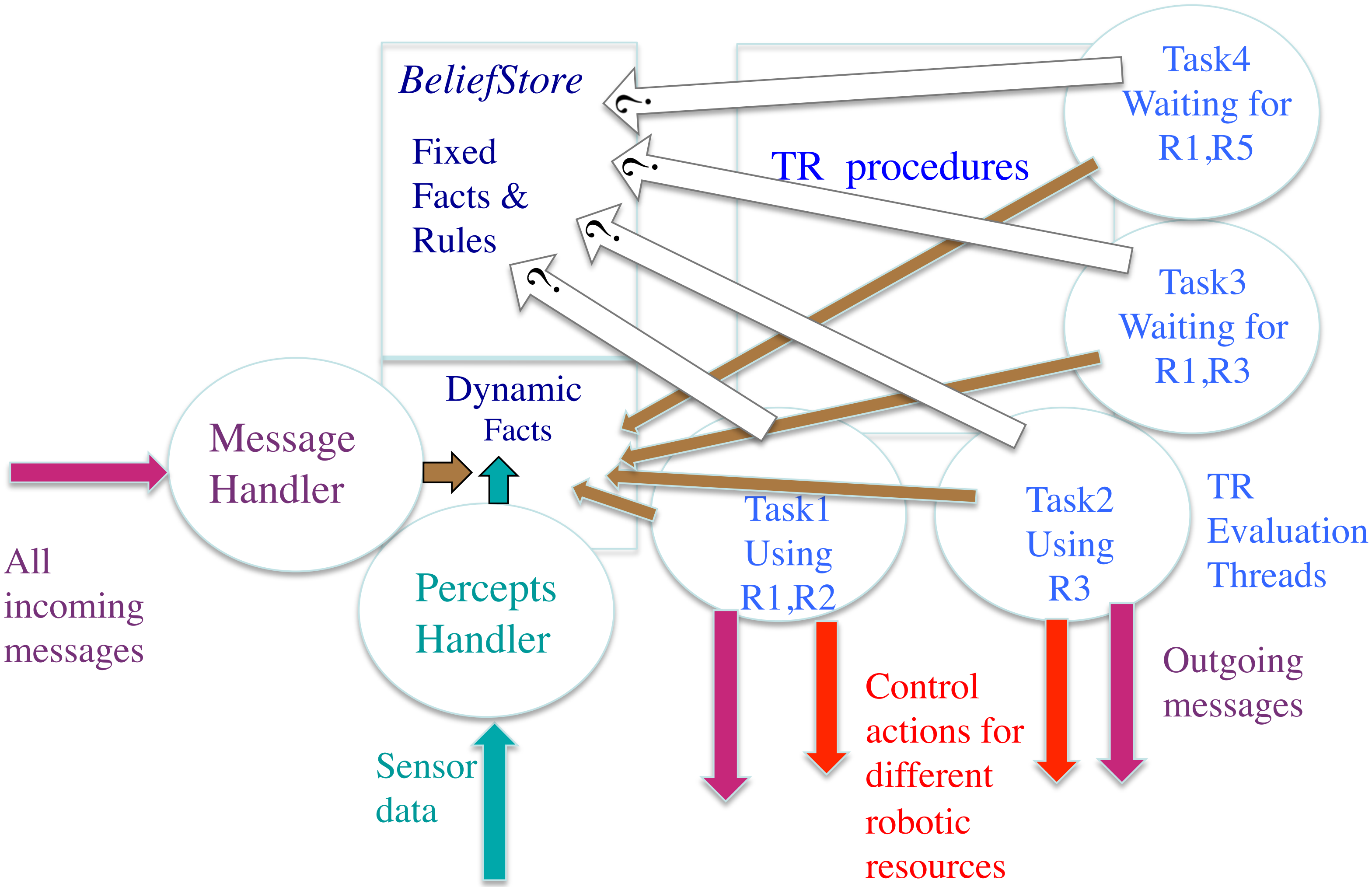
Catching and recovering from system added error beliefs

```
collect_bottle_wrapper: pedro_handle
collect_bottle_wrapperPersonAg){

        error(wait_repeat(open_gripper))  ~> () ++
                                   gripper_fault to PersonAg

        true ~> collect_bottle
        }
```

Multi-tasking using multiple robotic resources

- Resources now parameters of procedures and primitive actions e.g. move(robot1,2.5) collect_bottle(robot2)
- Resource use controlled using **task atomic** procedures that **achieve stable sub-goals** before releasing resources
  - Task must acquire **all** resources that may be needed before entry of the procedure
  - **Prevents deadlocks, allows the stable sub-goal achievement**
- Tasks **coordinate** use of resources using the *BeliefStore* as a **Linda** tuple store – invisible to TeleoR programmer
  - running_(Tsk, Resources), waiting_(Tsk, Since, Resources)

Multi-tasking architecture



Multi-resources tower building

arm1 arm2

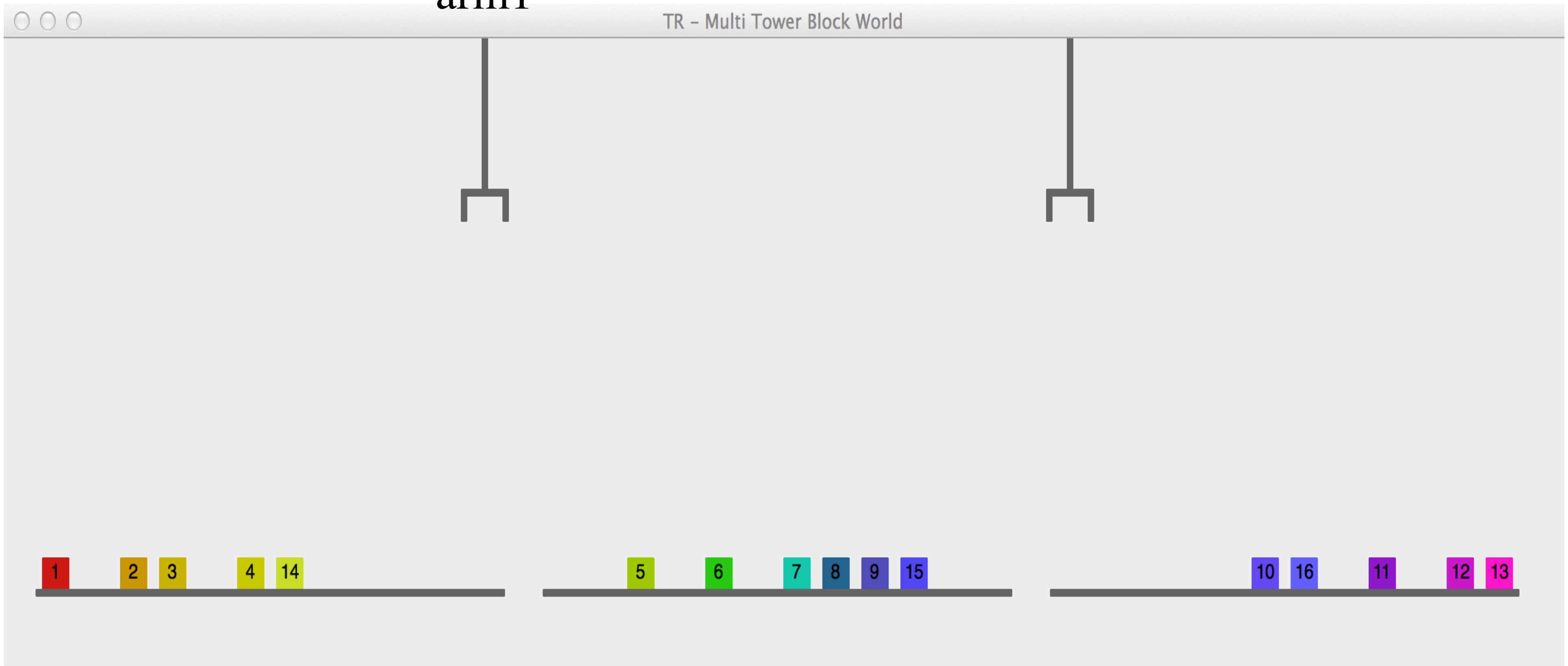


table1 shared table2

QuLog *BeliefStore*

```
block::= 1.. 15                          % range type of 15 integers
table::= table1 | shared | table2        % enumerated type
place::= table || block                  % union type
resource::= arm || table                 % resource reserved type name
```

percept on(block,place), holding(arm,block)

Example percept facts: on(1,table1) on(2,3) holding(arm1,5)

Plus rules for percept interrogation relations such as:

```
tower: [block],?table      % all adjacent on the table, with clear top block
tower([B,..Bs],Tbl) <= clear(B) & sub_tower([B,..Bs],Tbl)

sub_tower:[block],?table                          % recursive def
sub_tower([B],Tbl) <= on(B,Tbl)
sub_tower([B1,B2,..Bs],Tbl) <= on(B1,B2) & sub_tower([B2,..Bs],Tbl)

clear: place        % a table always clear, a block clear if nothing on top
clear(Tbl) <= isa(Tbl,table)
clear(B) <= isa(B,block) & not on(_,B)
```

makeTower procedure - uses inferable beliefs

```
durative pickup(arm,block,place),  putdown(arm,block,place)

task_start makeTower: arm, [block], table
makeTower(Arm,Blks,TwrTbl){   % TwrTbl is where tower must be
                  % built, always table1 or table2. Blks list of block labels
  tower(Blks,TwrTbl)  ~> ()
  sub_tower(Blks,TwrTbl) & Blks=[B,..]  ~>
                                  makeClear(Arm,B,TwrTbl)
  Blks=[B,B',.. Blks'] & tower([B',.. Blks'],TwrTbl)  ~>
                             possTwoArmMove(Arm,B, B',TwrTbl)
  Blks=[B]  ~>  possTwoArmMove(Arm,B,TwrTbl,TwrTbl)
   % Actions of above three rules should all achieve tower(Bs,TwrTbl)
  Blks=[B,.. Blks']  ~>  makeTower(Arm,Blks',TwrTbl)
 }                 % Last rule action should achieve guard of rule 3
```

Auxiliary procedures

possTwoArmMove: arm, block, place, table – 5 rules

% Will fetch the block *from wherever it is and put down on* place

```
task_atomic oneArmMove: arm, block, table, place, table
oneArmMove(Arm,Blk,BlkTab,ToPlace,ToPlaceTab){

    on(Blk,ToPlace)  ~>  ()

    clear(ToPlace) &  clear(Blk)  ~>
                    clearOneArmMove(Arm,Blk,BlkTbl,ToPlace,ToPlaceTab)

    clear(ToPlace) until not holding(Arm,_)  ~>  unpile(Arm,Blk,BlkTbl)

    true until not holding(Arm,_)  ~>  unpile(Arm,Place,ToPlaceTab)
    }
```

Entire control program just 20 TeleoR rules in 4 procedures.

TeleoR semantics and implementation

- Formal State Transition Semantics
- Optimised Reference Implementation
    - Translator to QuLog clauses
    - Runtime system, using QuLog action rules, that implements state transition semantics
    - Because of compile time analysis only rule guards that might have changed inference status are re-checked
- MQTT and ROS interfaces
- QuLog currently compiled to multi-threaded Qu-Prolog
- Will be compiled to QuLog Abstract Machine Code emulated in C by end of 2015

Future Extensions

- TeleoR plans with ?Goal (achieve a given goal) rule actions
- Use non-deterministic plan call selection rules

  ?Goal :: BSQuery ~~> PlanCallOption

  +Fact :: BSQuery ~~> PlanCallOption

  as in BDI architectures such as AgentSpeak and Jason
- Handle plan failures retrying most recent ?Goal action
- Task priorities and task deadlines
  - Application specific task scheduling
- Uncertain beliefs and probabilistic reasoning
- Constraint handling in QuLog

More info and software

Clark & Robinson, *Programming Robotic Agents*: *a Multi-tasking Teleo-reactive Approach*, Springer, early 2015,
Draft chapters from k.clark@imperial.ac.uk on request

Clark & Robinson, *QuLog: Engineering Agent Applications*, Springer, middle 2015,
Draft chapters from k.clark@imperial.ac.uk on request

TeleoR and QuLog Linux/OS X Software via
http://staff.itee.uq.edu.au/pjr/ Late February 2015

Collaboration and users welcomed