Planning, Execution & Learning: Execution Architectures

Reid Simmons

Architectural Design Principles

- ***Modularity***
  - Reduces complexity
  - Algorithms and representations tuned to particular roles
- ***Hierarchy***
  - Layers of increasingly complex behaviors
  - Promotes reactivity
  - *Disagreements on how to create hierarchy*
- ***Concurrency***
  - Monitor environment while carrying out plans
  - Concurrent planning and execution

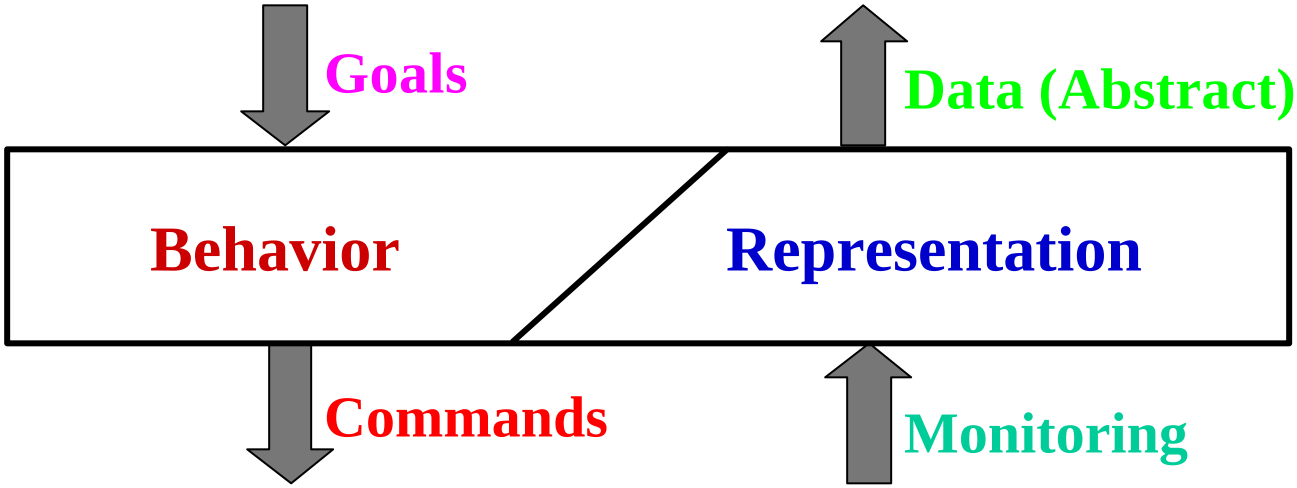
Layered Architectures

- Upper layers utilize functionality of lower layers to implement more complex tasks
- Upper layers typically operate at lower ***temporal*** and ***spatial*** resolutions

Xavier Architecture (1995)

Task Planning (Prodigy)
Path Planning (Decision-Theoretic)
Map-Based Navigation (POMDPs)
Local Obstacle Avoidance (Curvature Velocity Method)
Servo-Control (Commercial)

Anatomy of a Layer



- Each Layer Provides "Guidance" to Next Lower Level
- Each Layer has Relative Autonomy to Achieve Tasks Robustly, in Face of Uncertainty
- Each Layer Abstracts Data for Higher Levels
  - Each layer must monitor progress of lower level

Three-Tiered Architectures

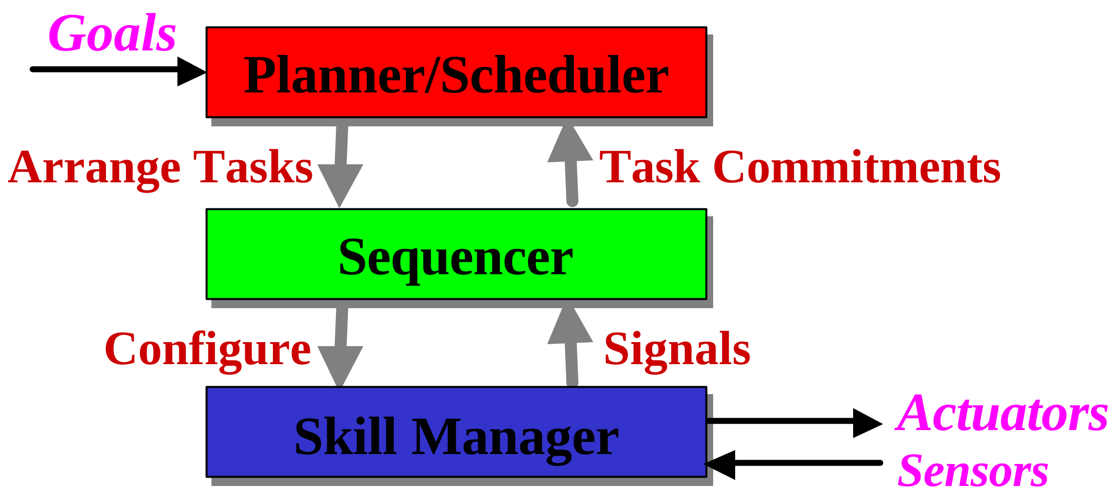
Deals with goals and resource interactions

Task decomposition; Task synchronization; Monitoring; Exception handling; Resource management

Deals with sensors and actuators

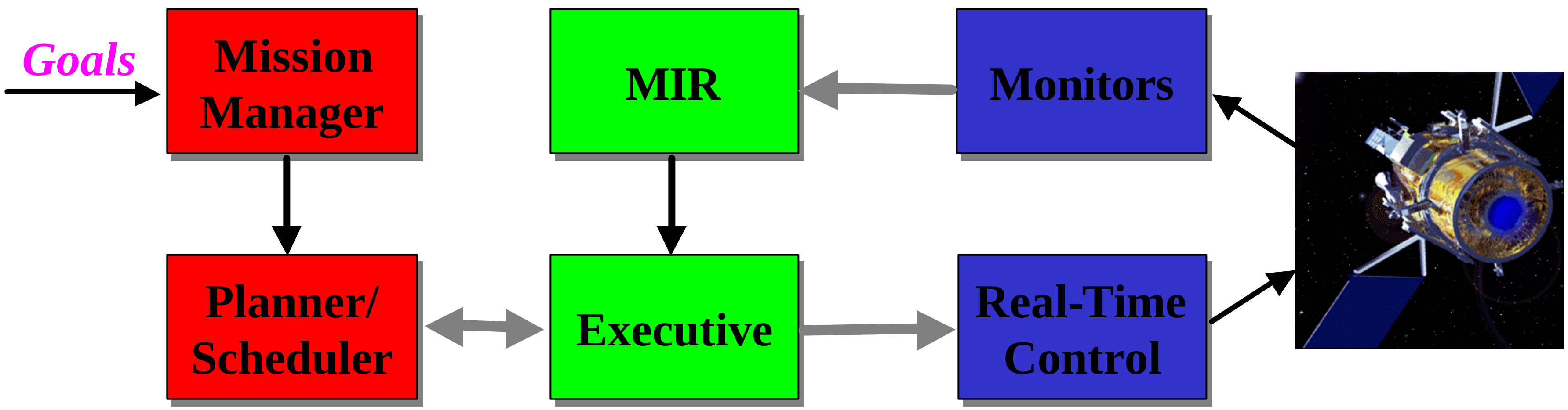
3T (Bonasso & Kortenkamp, 1996)

- Explicit Separation of Planning, Sequencing, and Control
  - Upper layers provide ***control flow*** for lower layers
  - Lower layers provide ***status*** (state change) and ***synchronization*** (success/failure) for upper layers
- Heterogeneous Architecture
  - Each layer utilizes algorithms tuned for its particular role
  - Each layer has a representation to support its reasoning



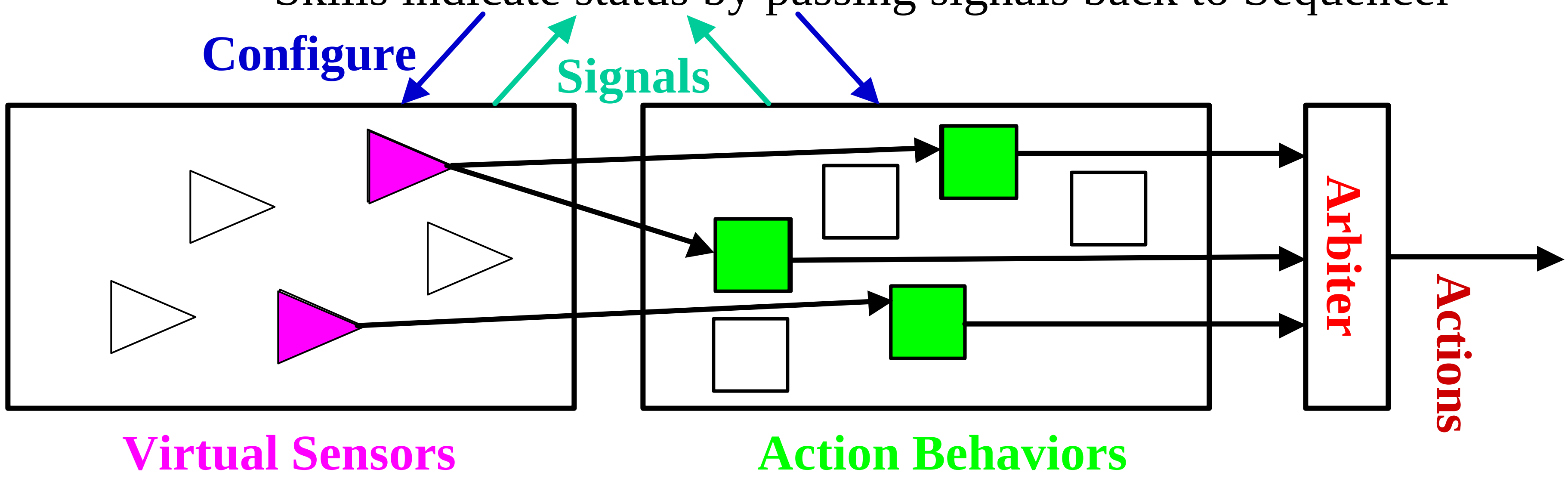
Remote Agent (1998)

- First Truly Autonomous System in Space
  - Controlled DS1 spacecraft for several days in 1999
  - Closed-loop, goal-based commanding
  - Model-based programming
  - Real-time inference
  - Integrated declarative/procedural paradigms



Managing Sets of Behaviors

- 3T "Skill Manager"
  - ***Skills*** are concurrent behaviors, including perceptual behaviors
  - Dynamic creation of real-time feedback loops
    - Higher tier ("Sequencer") connects sensing and action modules and ***enables*** subsets of skills
    - Skills indicate status by passing signals back to Sequencer



Sequencer / Executive

- Forms a Bridge Between Planning and Behaviors
  - Discrete vs. continuous control
  - Symbolic vs. numeric representations
  - Real-time considerations
- Basic Roles
  - Decompose task into subtasks and dispatch tasks
  - Monitor execution for contingencies and opportunities
  - Reschedule tasks (or schedule new tasks) upon failure
- Differences Between Approaches
  - Methods for distributing functionality
  - Representation of domain and control knowledge
  - **RAP** (Firby); **TCA/TDL** (Simmons); ESL (Gat); PRS (Georgeoff)

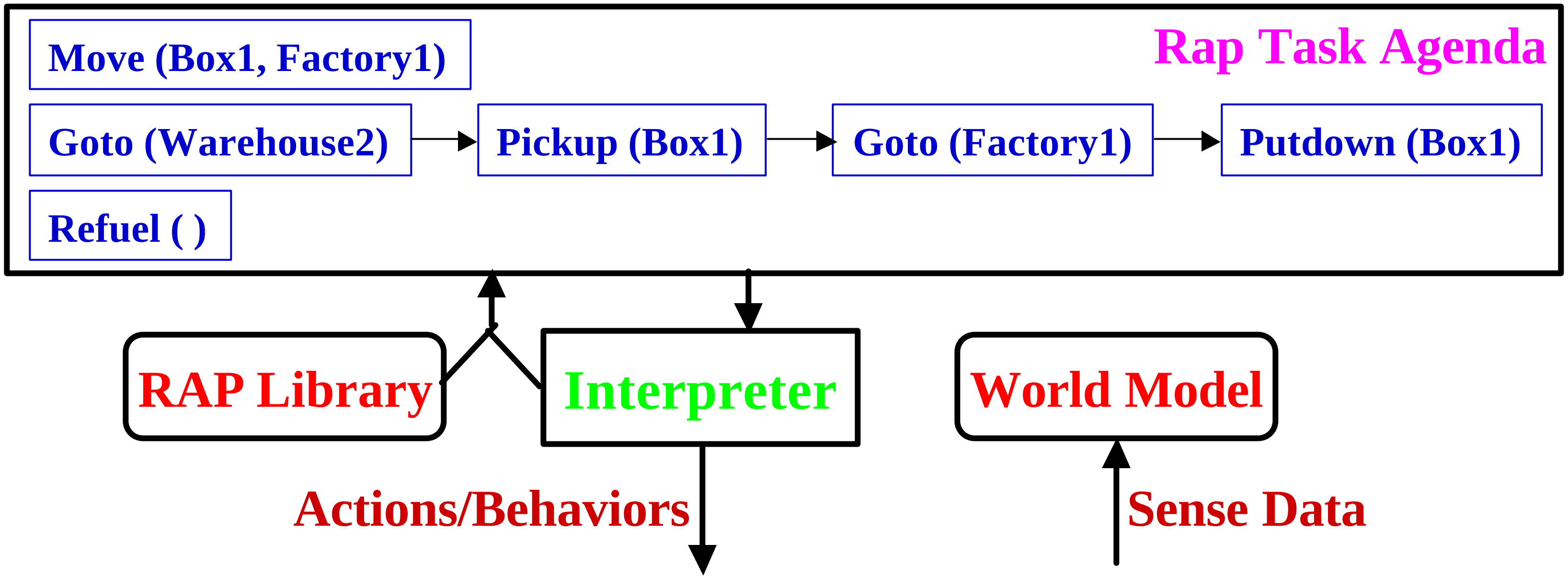
Reactive Action Packages (RAPs) (Firby 1987)

- Reactive Action Package
  - Autonomous process that pursues a planning goal
  - Sensing (monitoring) intrinsic part
  - Goal satisfaction always verified
  - Multiple methods to achieve goals

```
(define-rap
    (index (move ?thing ?place))
    (succeed (location ?thing ?place))
    (method (context (and (location ?thing ?loc) (not (= ?loc UNKNOWN))
        (task-net
            (t0 (goto ?loc)        ((truck-location ?loc) for t1))
            (t1 (pickup ?thing)  ((truck-holding ?thing) for t2)
                                  (truck-holding ?thing) for t3)))
            (t2 (goto ?place)     ((truck-location ?place) for t3))
            (t3 (putdown ?thing))))
    (method (context (location ?thing UNKNOWN))
            (t0 (goto WAREHOUSE)))
```

RAP Interpreter

- Methods are Chosen Based on Current Situation
- If a Method Fails, Another is Tried Instead
- Tasks do not Complete Until Satisfied
- Methods can Include Monitoring Subtasks to Deal with Unexpected Contingencies and Opportunities

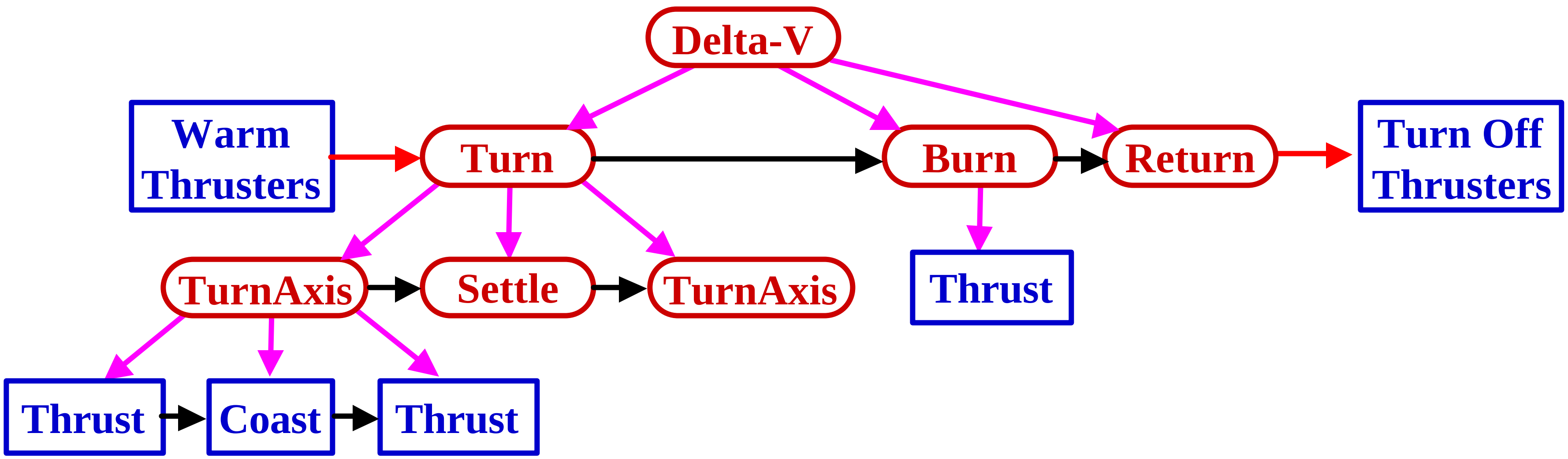


Task Control Architecture (TCA) *(Simmons 1994)*

- Provides Commonly Needed Control Constructs
  - Task decomposition
  - Task coordination and synchronization
  - Execution monitoring and exception handling
  - Resource management (simple)
- Integrates ***Deliberative*** and ***Reactive*** Behaviors
- Facilitates Incremental Development
  - Adding new tasks
  - Adding new reactive behaviors
- Used in Over a Dozen Autonomous Systems

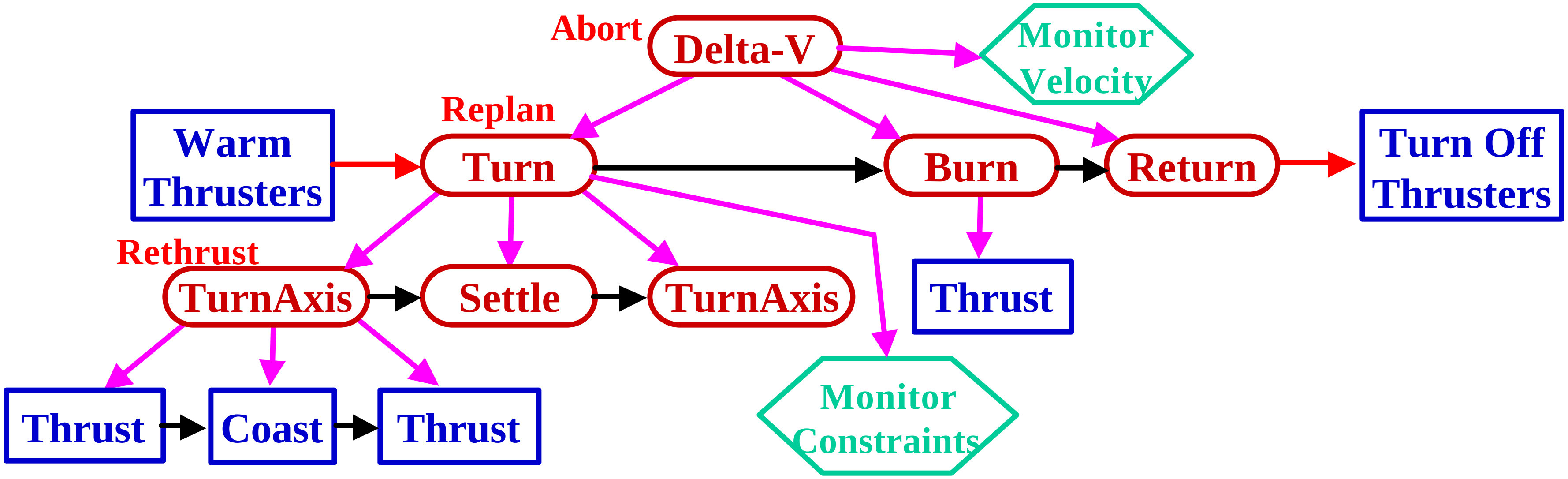
Planning and Execution

- TCA Maintains and Coordinates ***Task Trees***
  - ***Execution trace*** of hierarchical plans
    - Created dynamically at run time
    - Can be conditional and recursive
  - Temporal constraints (partially) order task execution
  - Planning and sensing treated as schedulable activities; Concurrent planning, sensing, and execution



Monitoring and Exception Handling

- Task Trees Augmented with ***Reactive*** Elements
  - Task-specific execution monitors
  - Context-dependent, hierarchical exception handlers



- Replan by Analyzing and Manipulating Task Trees
  - Terminate subtrees
  - Add new nodes and/or temporal constraints

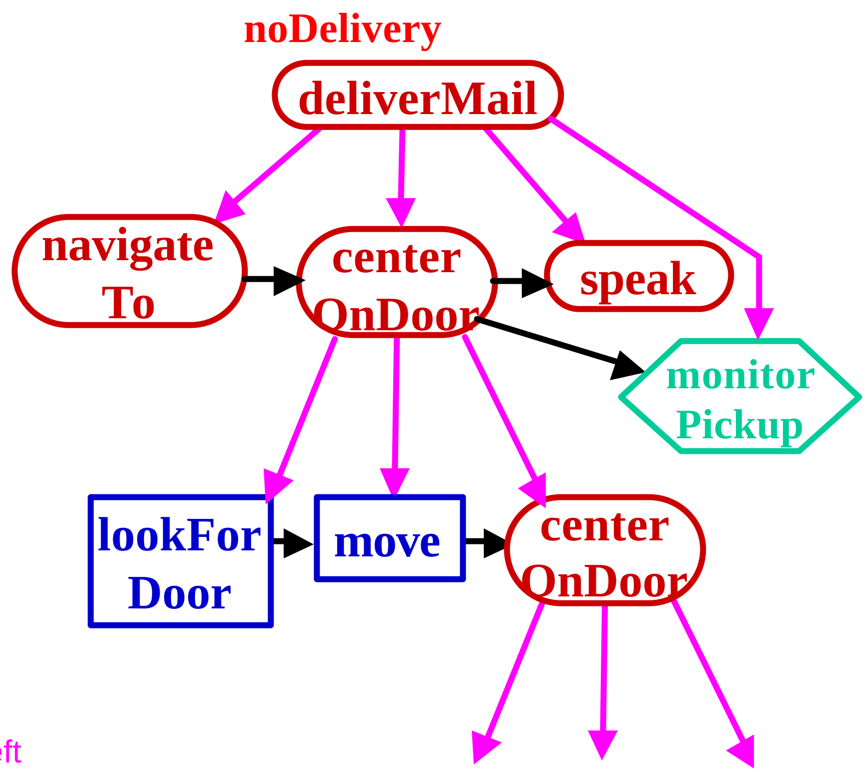
Task Definition Language (TDL)

- High-Level Language Tailored to Task-Level Control
  - Extension of C++ with explicit syntax for task-level control constructs
  - Compiles into pure C++ with calls to task management library
  - Extension of functionality provided by TCA
  - Threaded
- Requirements
  - ***Simple concepts*** should be expressible in ***simple terms***
  - Do not ***preclude*** expression of complex control constructs
  - ***Natural integration*** with existing code

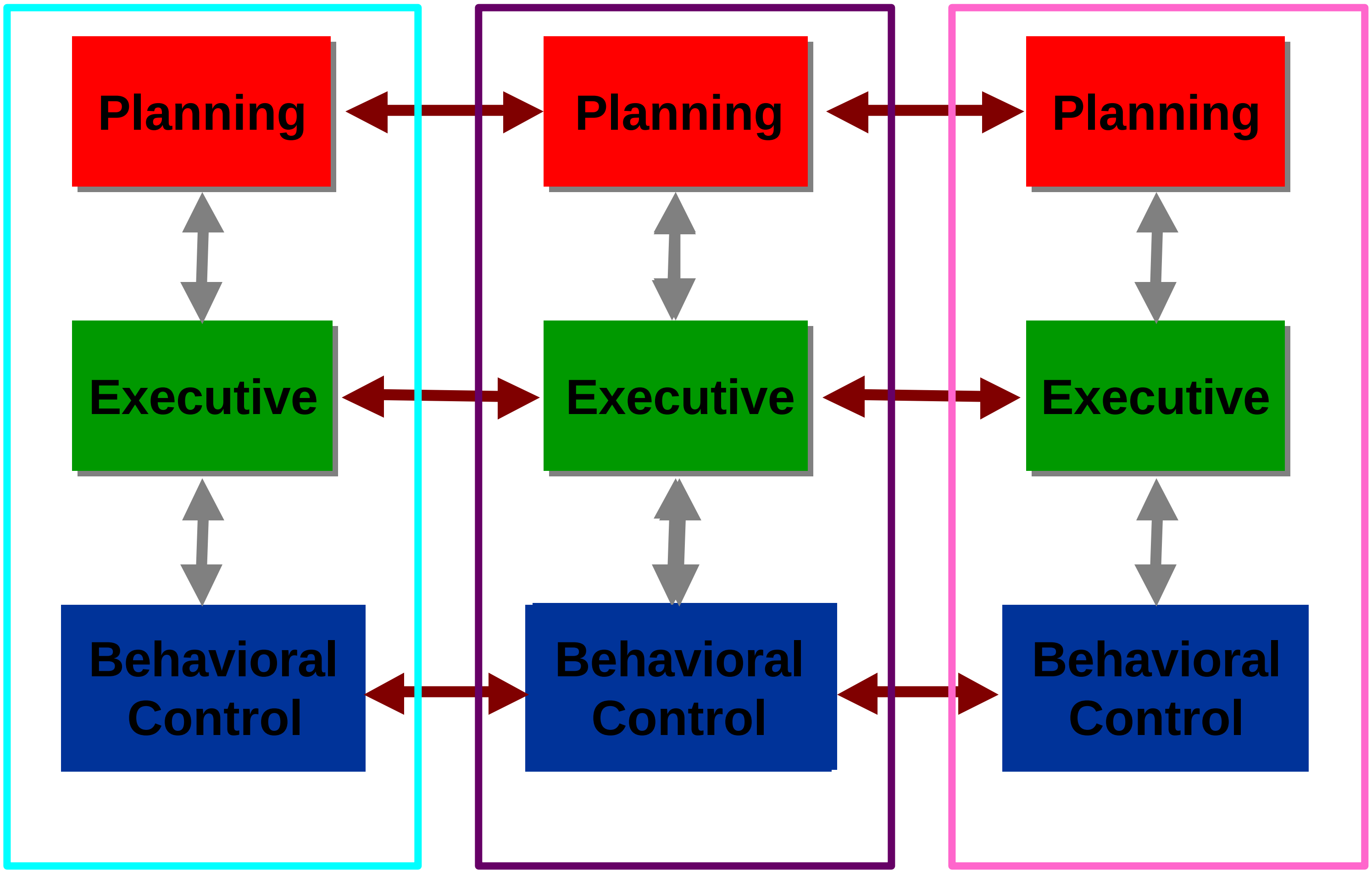
TDL Example

```
Goal deliverMail (int room)
        Exception Handler noDelivery
{
        double x, y;
        getRoomCoordinates(room, &x, &y);
        spawn navigateTo(x, y);
         spawn centerOnDoor(x, y)
           with sequential execution previous,
                   terminate in 30.0;
         spawn speak("Xavier here with your mail")
           with sequential execution centerOnDoor,
                   terminate at monitorPickup completed;
         spawn monitorPickup()
           with sequential execution centerOnDoor;
}
```

```
Goal centerOnDoor(double x, double y)
{
        int whichSide;
         spawn lookForDoor(&whichSide) with wait;
        if (whichSide != 0) {
           if (whichSide < 0) spawn move(-10); // move left
           else spawn move(10);  // move right
        }
         spawn centerOnDoor(x, y)
           with disable execution until
                   previous execution completed;
}
```

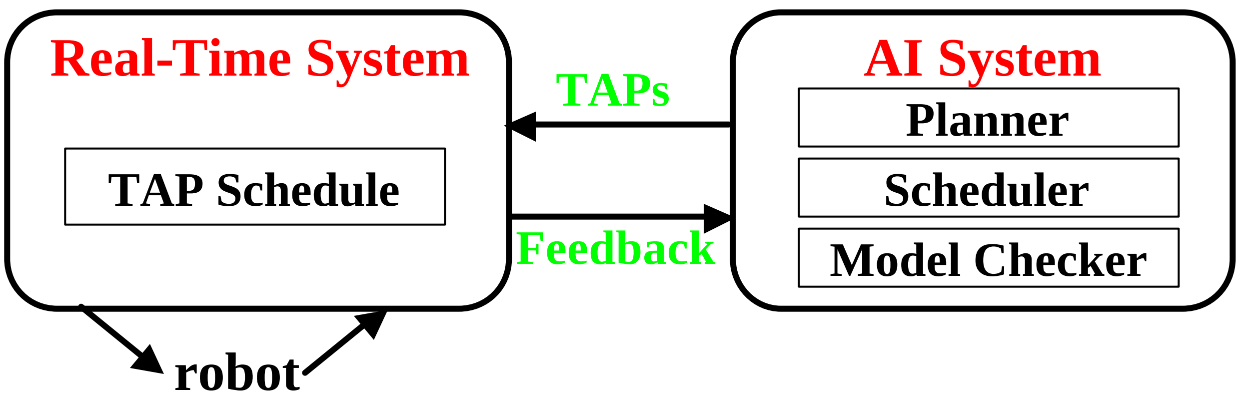


Multi-Robot Coordination



CIRCA (Musliner 1993)

- Provide Both ***Bounded Rationality*** and ***Bounded Reactivity***
  - Distinguishes control-level and task-level goals
  - Guarantee achievement of control-level goals
    - AI system creates provably (probabilistically) feasible schedules that prevent failure
  - Trades off ***performance*** for ***reliability***
    - Reduce set of task-level goals
    - Change task parameters (e.g., move slower)

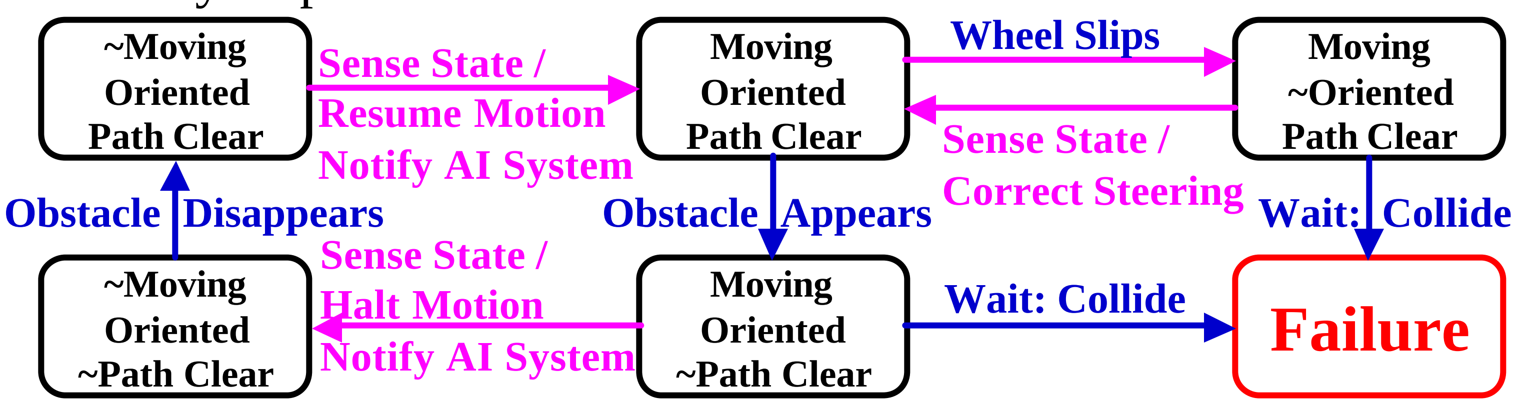


CIRCA Representations

- ***Test Action Pair*** (TAP)
  - Interface between real-time and AI system
  - Simple production rule with resource bounds

```
TAP stop-if-object-ahead
 TEST: [0.15] (< (sonar-forward) *safety-distance*)
 ACTION: [0.05] (progn (halt) (notify-AIS 'halted))
 RESOURCES: (sonar base-motors)
 MAX-PERIOD: 0.7
```

- Model of Dynamics
  - State diagram with event, action and temporal transitions
  - May be probabilistic



CLARAty (Volpe & Nessnas, 2000)

- Two-Tiered Architecture
  - Functional layer: Object oriented, reusable
  - Decision layer: Tightly integrates planner (Aspen) and executive (TDL)
- Developed at NASA for Next-Generation Mars Rovers
  - Still very much under development