

## Why PiELo?

PiELo explores designing a language from the ground up for swarms. To show the benefits, we will compare to ROS, a standard robotics programming framework.

### Limitations of ROS:

- Difficult to scale to multiple robots
- Reactivity and consensus require setting up multiple nodes and callback functions

This code example shows a synchronization behavior across a swarm. PiELo makes this a significantly easier task in terms of code quantity and complexity.

```
17 bool barrierCallback(barrier_test::Barrier::Request &req,
18                    barrier_test::Barrier::Response &res)
19 {
20     std::unique_lock<mutex> lock(barrier_mutex);
21
22     // Record arrival (ignore duplicates)
23     if(std::find(arrived_ids.begin(), arrived_ids.end(), req.robot_id) == arrived_ids.end()){
24         ROS_INFO("Robot %d arrived at the barrier.", req.robot_id);
25         arrived_ids.push_back(req.robot_id);
26     }
27     else {
28         ROS_WARN("Robot %d has already signaled the barrier.", req.robot_id);
29     }
30
31     // Wait until all arrived
32     while (arrived_ids.size() < robot_count) {
33         cv.wait(lock);
34     }
35
36     // When all arrived, release each waiting call
37     res.success = true;
38     res.message = "Barrier released for robot " + std::to_string(req.robot_id);
39     ROS_INFO("Barrier released for robot %d", req.robot_id);
40     return true;
41 }
42
43 int main(int argc, char **argv)
44 {
45     ros::init(argc, argv, "barrier_server");
46     ros::NodeHandle nh;
47     ros::ServiceServer s = nh.advertiseService("barrier", barrierCallback);
48     ROS_INFO("Barrier server is ready. Waiting for all robots to arrive.");
49     ros::AsyncSpinner spinner(5);
50     spinner.start();
51
52     // Monitor barrier condition, notify waiting callbacks.
53     ros::Rate rate(1);
54     while (ros::ok()) {
55         {
56             std::unique_lock<mutex> lock(barrier_mutex);
57             if (arrived_ids.size() >= expected_count) {
58                 cv.notify_all();
59             }
60         }
61         rate.sleep();
62     }
63     return 0;
64 }
65
66
67
68
69 int main(int argc, char **argv)
70 {
71     ros::init(argc, argv, "barrier_client");
72     if (argc < 2) {
73         ROS_ERROR("Usage: barrier_client <robot_id (1-5)>");
74         return 1;
75     }
76     int robot_id = std::stoi(argv[1]);
77     if (robot_id < 1 || robot_id > 5) {
78         ROS_ERROR("Robot ID must be between 1 and 5.");
79         return 1;
80     }
81
82     ros::NodeHandle nh;
83     ros::ServiceClient client = nh.serviceClient<barrier_test::Barrier>("barrier");
84
85     // Simulate different speeds
86     int sleep_time = 5 - robot_id;
87     ROS_INFO("Robot %d is moving toward the barrier. Sleeping for %d seconds to simulate speed.", robot_id, sleep_time);
88     ros::Duration(sleep_time).sleep();
89
90     // Arrives at barrier, call the service
91     barrier_test::Barrier srv;
92     srv.request.robot_id = robot_id;
93     ROS_INFO("Robot %d has arrived at the barrier and is now waiting for the others...", robot_id);
94     if (client.call(srv))
95     {
96         if (srv.response.success) {
97             ROS_INFO("Robot %d: %s", robot_id, srv.response.message.c_str());
98         }
99         else {
100             ROS_ERROR("Robot %d: Barrier was not released properly.", robot_id);
101         }
102     }
103     else {
104         ROS_ERROR("Robot %d: Failed to call service 'barrier'", robot_id);
105         return 1;
106     }
107     return 0;
108 }
109
110
111
112
```

ROS: 5 files with 200+ lines

```
1 (begin
2   (include functions.txt)
3   (var global inert barrierComplete)
4   (set barrierComplete 0)
5   (var global inert barrierSum)
6   (var map reactive barrier)
7
8   (fun reactive checkBarrier () (begin
9     (var local inert barrierSum)
10    (set barrierSum 0)
11    (foreach inert (in i barrier') (set barrierSum (+ barrierSum i)))
12    (if (> barrierSum 1) (set barrierComplete 1) 0)))
13
14   (var local inert total_distance)
15   (set total_distance 10)
16   (set distance_covered (get_distance_covered))
17   (set barrier (if (>= distance_covered total_distance) 1 0))
18   (set barrierChecker (checkBarrier))
19   (set speed (if (& (>= distance_covered total_distance) (! barrierComplete)) 0 (* (+ (% robotID 5) 1) 2)))
20   (set leftWheelVelocity speed')
21   (set rightWheelVelocity speed')
22   (spin)
23   (fun reactive step () (set distance_covered (get_distance_covered))))
```

PiELo: 22 lines of code

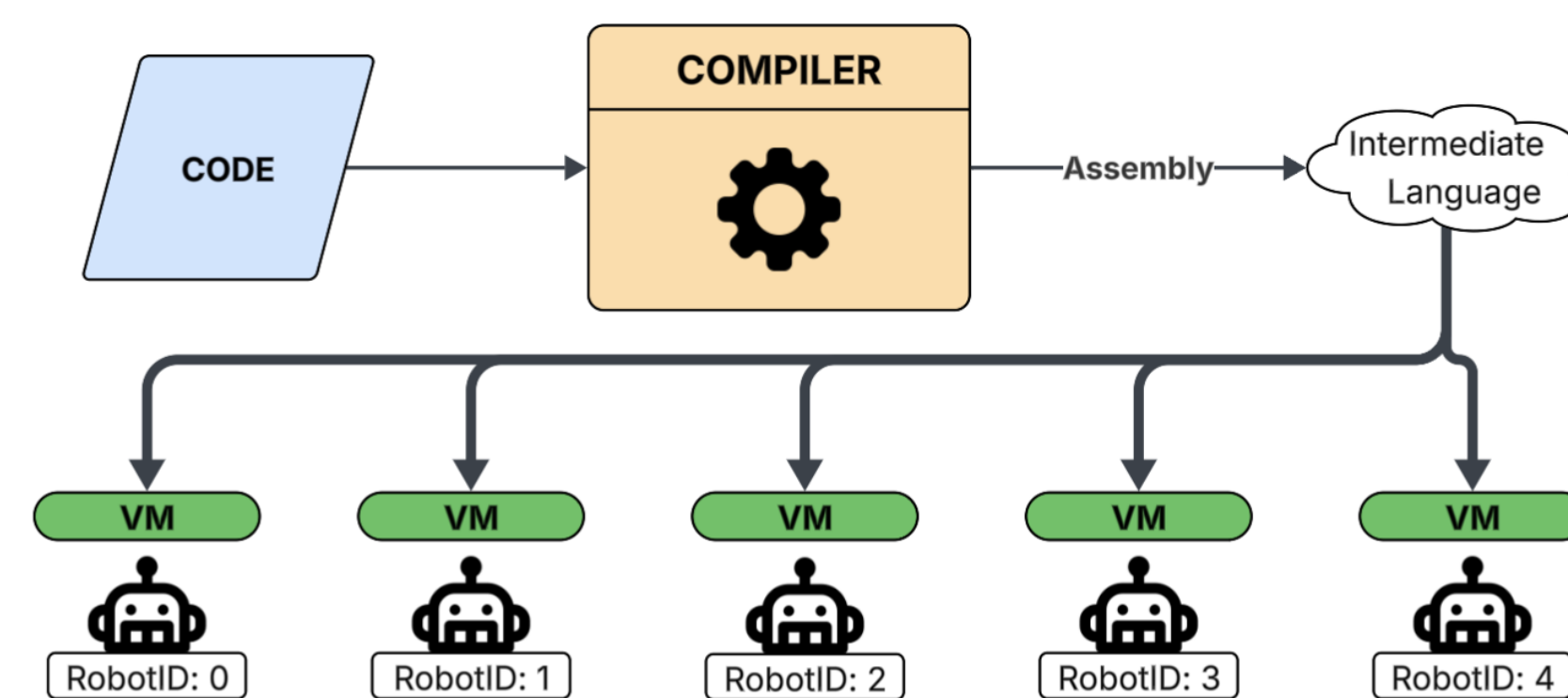
## The PiELo Programming Language

PiELo is a programming language for swarm robotics.

### Objectives:

- Trivial scaling for many robots
- Easily respond to events such as sensor updates or swarm communications
- Automatically achieve consensus across the swarm

## Code Compilation



Copies of the code are distributed to each robot in the swarm

## Practical Implementations



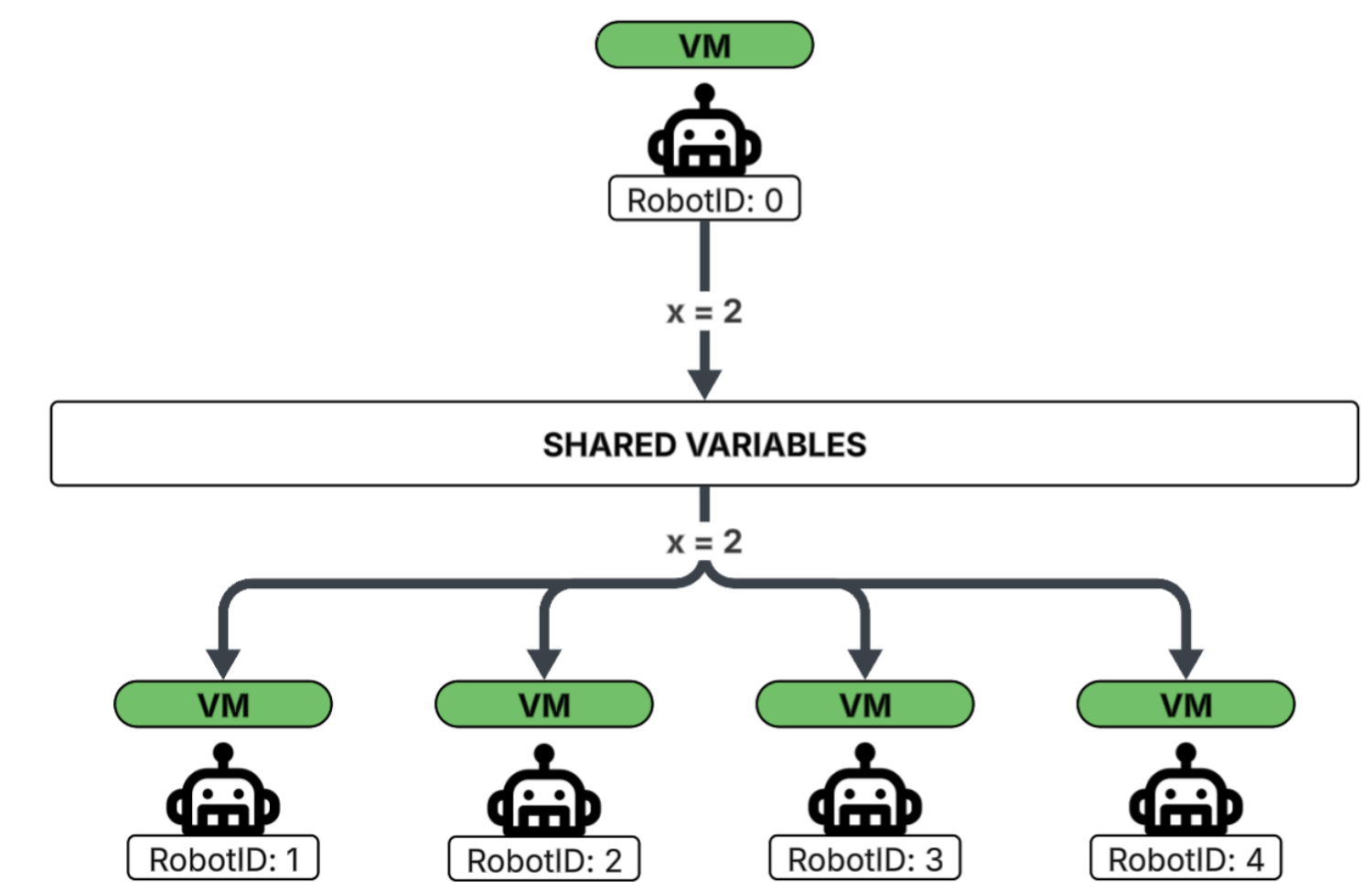
Barrier Test



Driving Robot

## Distributed Architecture

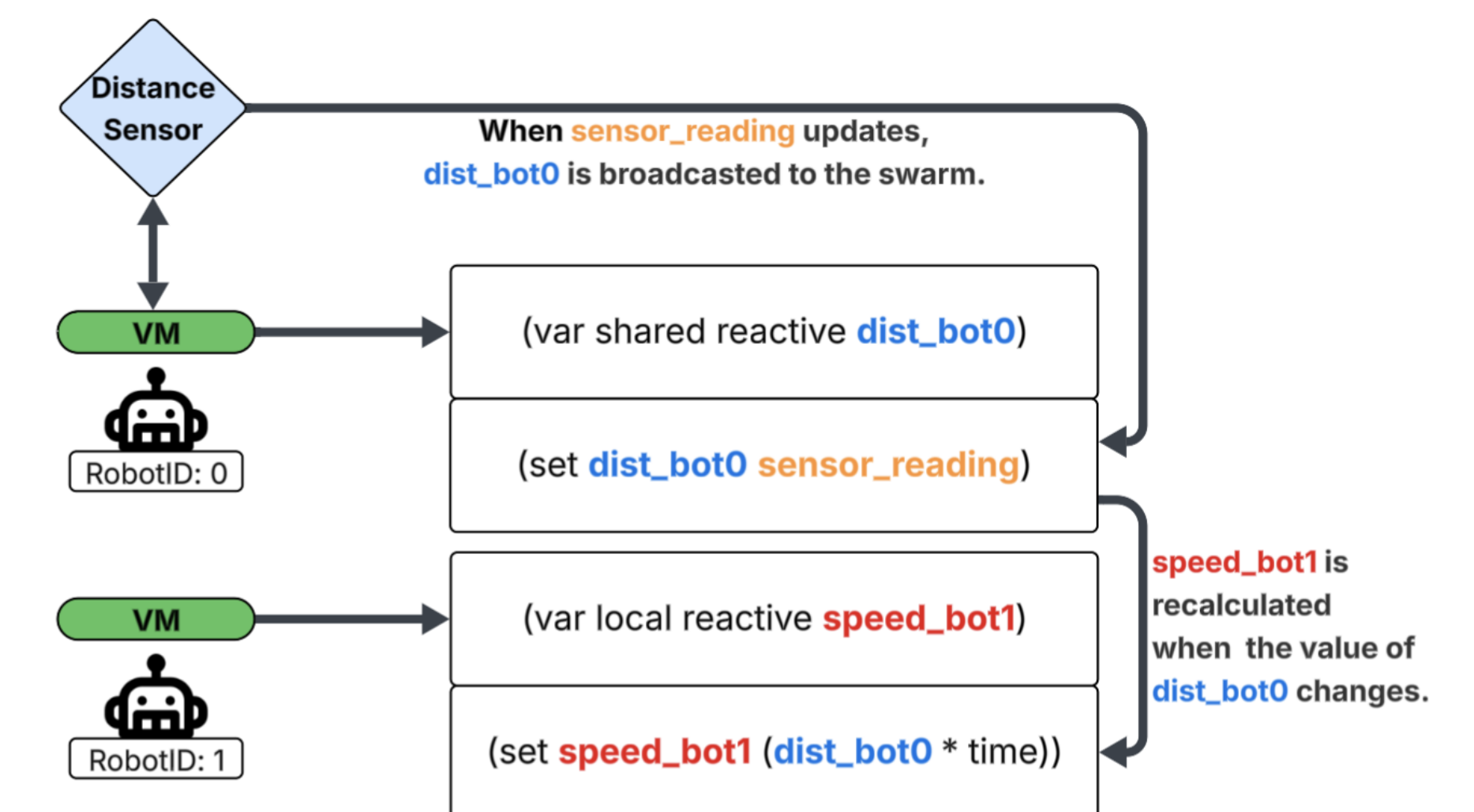
PiELo enables inter-robot communication using a network of Virtual Machines executing code on each robot. Shared variables are automatically updated with the rest of the swarm.



### Distributed Virtual Machine

The swarm achieves consensus with only two lines of code.

With reactivity built in, responding to updates across the swarm is trivial.



Reactivity & Consensus