Engineering High-Assurance Software for Distributed Adaptive Real-Time Systems

Sagar Chaki

Midwest Verification Day

October 22, 2016

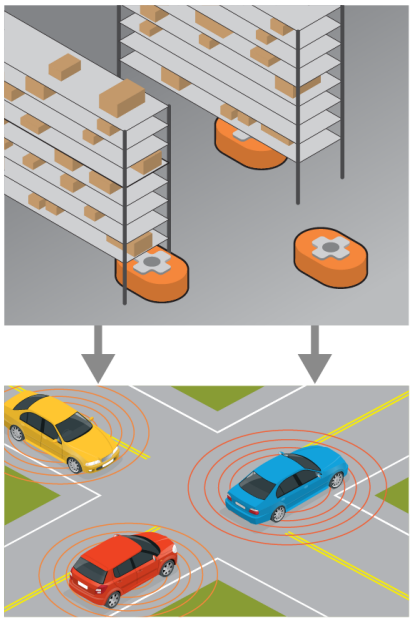
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM-0004136

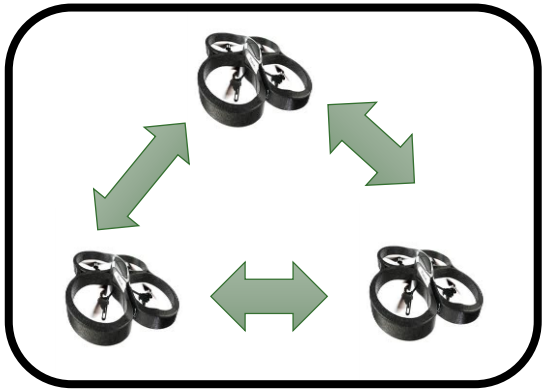
Motivation

Distributed Adaptive Real-Time (DART) systems are key to many areas of DoD capability (e.g., autonomous multi-UAS missions) with civilian benefits.



However achieving high assurance DART software is very difficult

- Concurrency is inherently difficult to reason about.
- Uncertainty in the physical environment.
- Autonomous capability leads to unpredictable behavior.
- Assure both guaranteed and probabilistic properties.
- Verification results on models must be carried over to source code.



High assurance unachievable via testing or ad-hoc formal verification

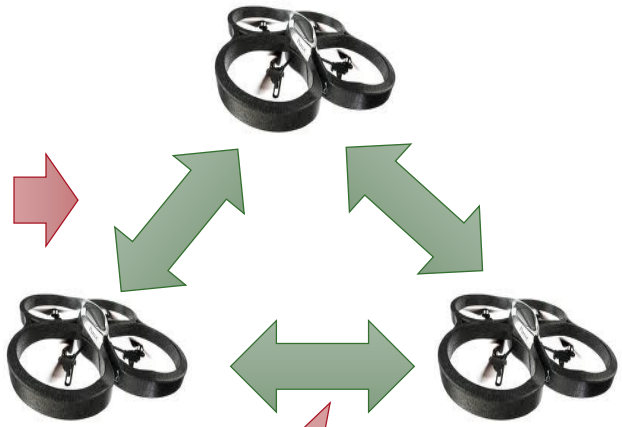
Goal: Create a <u>sound</u> engineering approach for producing high-assurance software for Distributed Adaptive Real-Time (DART)

DART Approach

1. Use DMPL (a DSL we developed) & AADL
2. Enables compositional and requirement specific verification
3. Use proactive self-adaptation and mixed criticality to cope with uncertainty and changing context

1. ZSRM Schedulability (Timing)
2. Software Model Checking (Functional)
3. Statistical Model Checking (Probabilistic)

Formal Description of System & Properties → **Verification** → **Code Generation** →

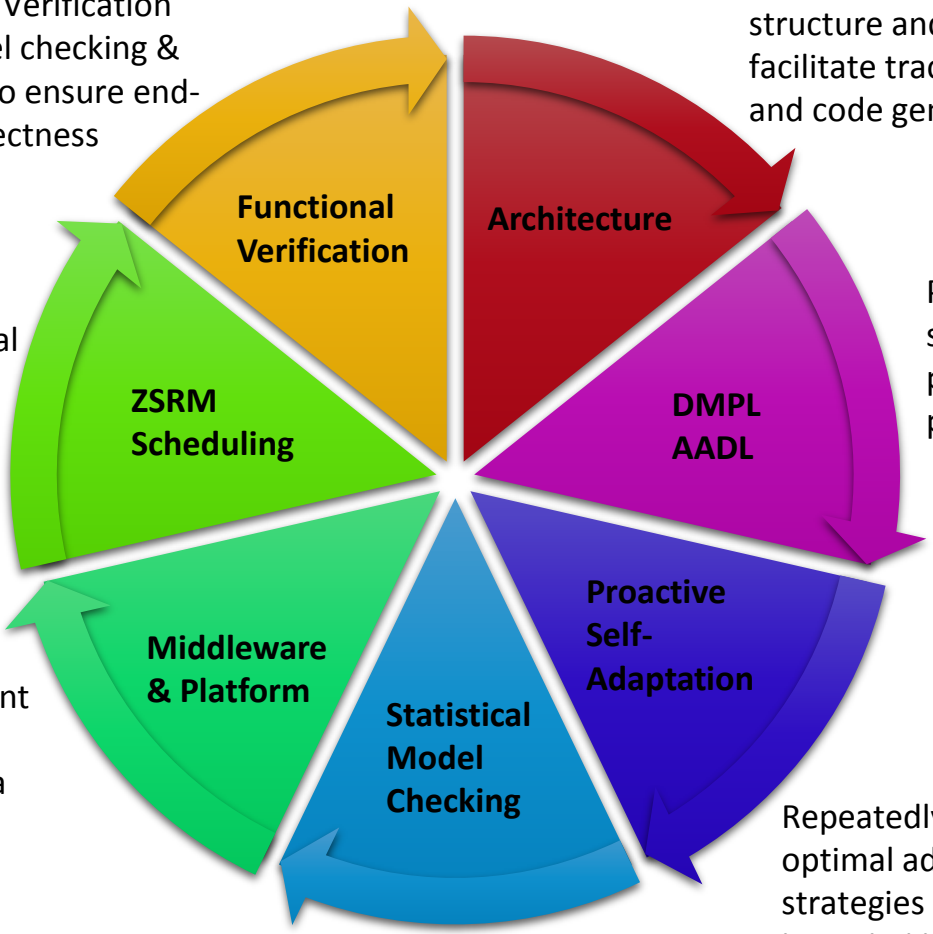


Brings Assurance to Code
1. Middleware for communication
2. Scheduler for ZSRM
3. Monitor for runtime assurance

Demonstrate on DoD-relevant model problem (DART prototype)
- Engaged stakeholders
- Technical and operational validity

https://github.com/cps-sei/dart
http://cps-sei.github.io/dart

Key Elements of DART



Sagar Chaki
Arie Gurfinkel

- Parameterized Verification
- Combine model checking & hybrid analysis to ensure end-to-end CPS correctness

Constrain the system structure and behavior to facilitate tractable analysis and code generation

Program DART systems and specify properties in a precise manner

David Kyle
Scott Hissam
Bud Hammons
Joseph Seibel

Repeatedly compute optimal adaptation strategies with bounded lookahead

Gabriel Moreno

Evaluate adaptation strategy quality over mission lifetime

Jeffery Hansen

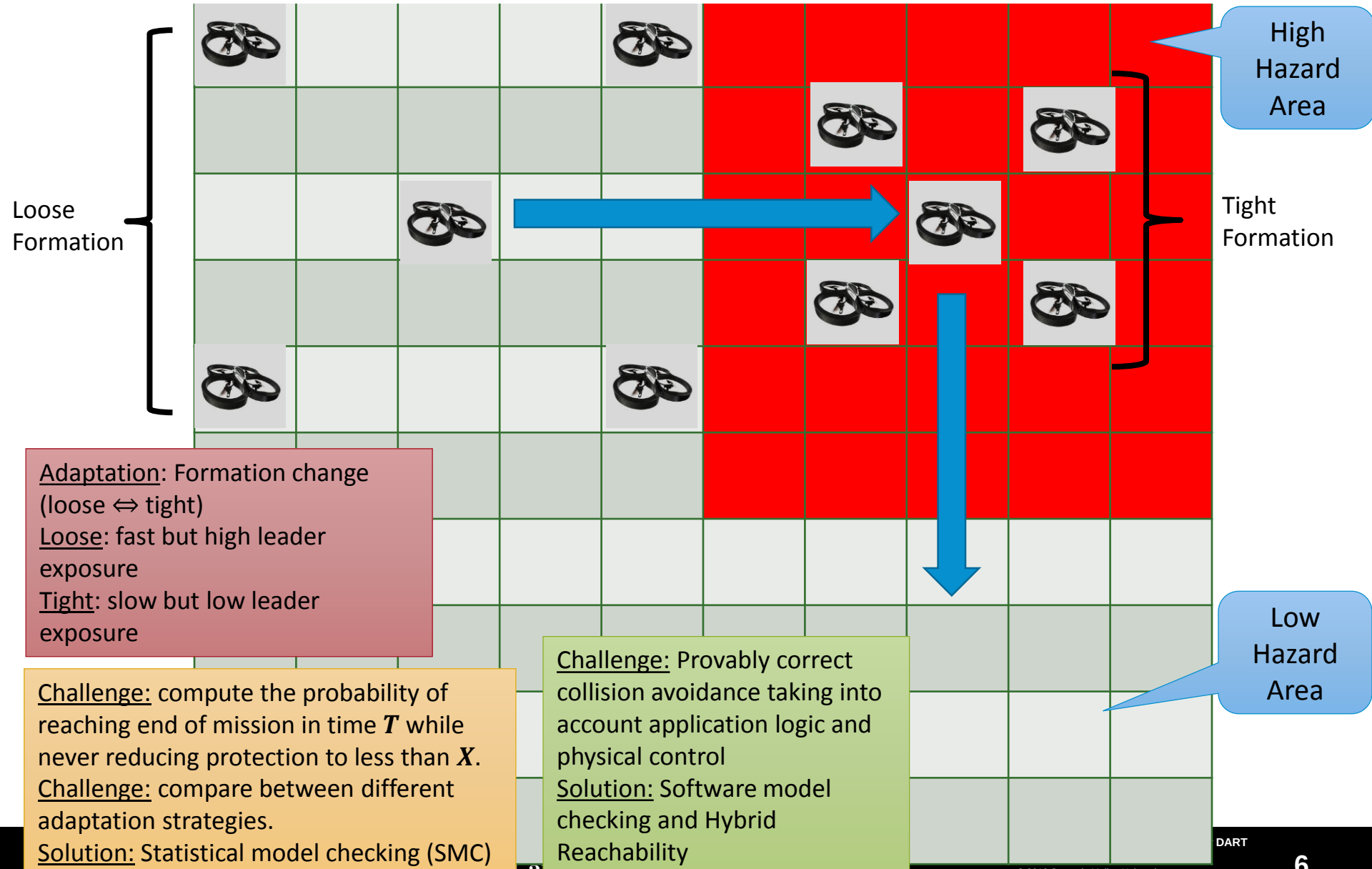
MADARA → efficient distributed shared variables with data consistency and quality of service. GAMS →Platform Interaction.

James Edmondson

Ensures high-critical tasks meet their deadlines despite CPU overload

Dionisio de Niz
Bjorn Andersson

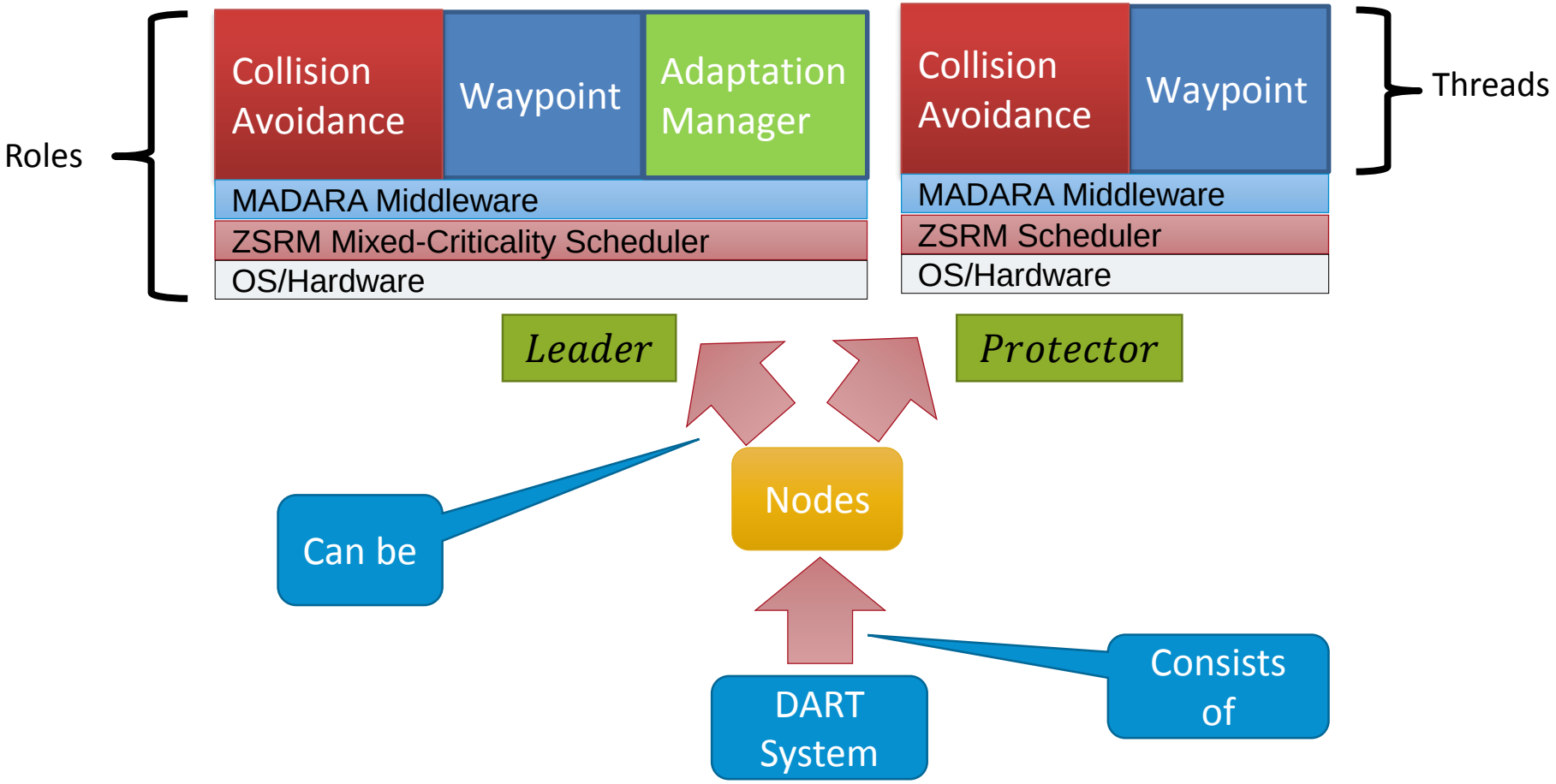
Example: Self-Adaptive and Coordinated UAS Protection



Adaptation: Formation change (loose ⇔ tight)
Loose: fast but high leader exposure
Tight: slow but low leader exposure

Challenge: compute the probability of reaching end of mission in time T while never reducing protection to less than X.
Challenge: compare between different adaptation strategies.
Solution: Statistical model checking (SMC)

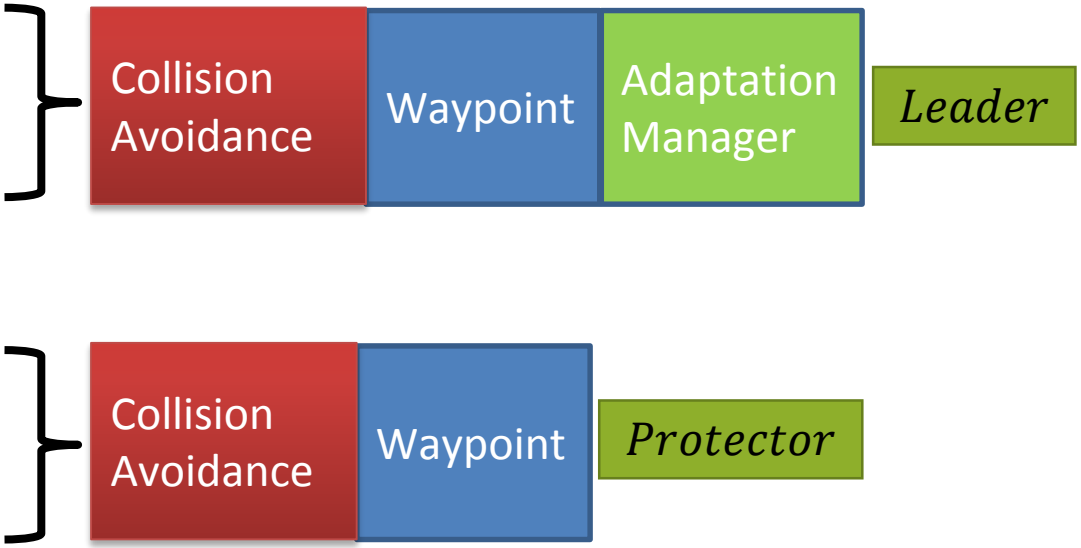
Challenge: Provably correct collision avoidance taking into account application logic and physical control
Solution: Software model checking and Hybrid Reachability

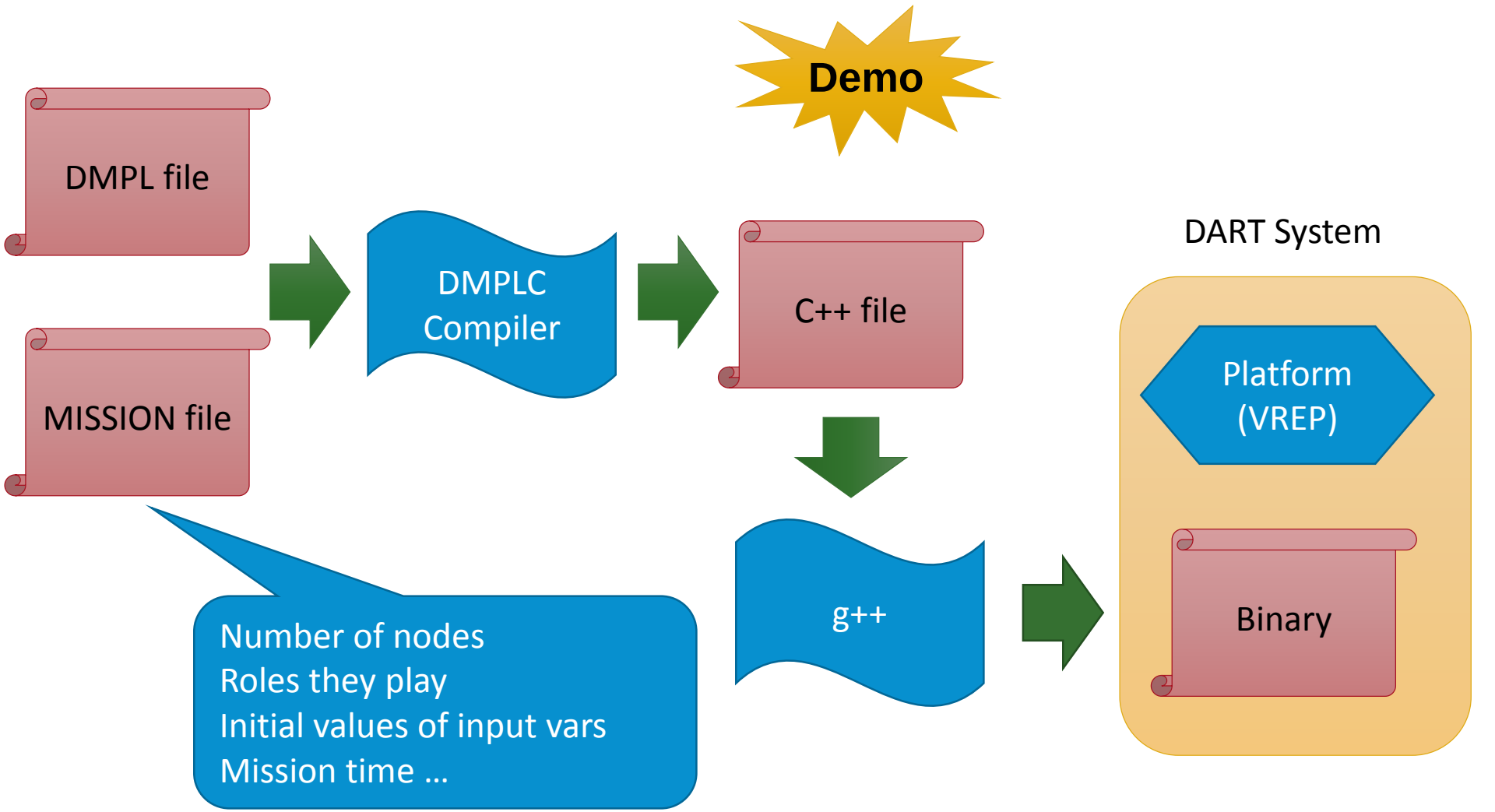


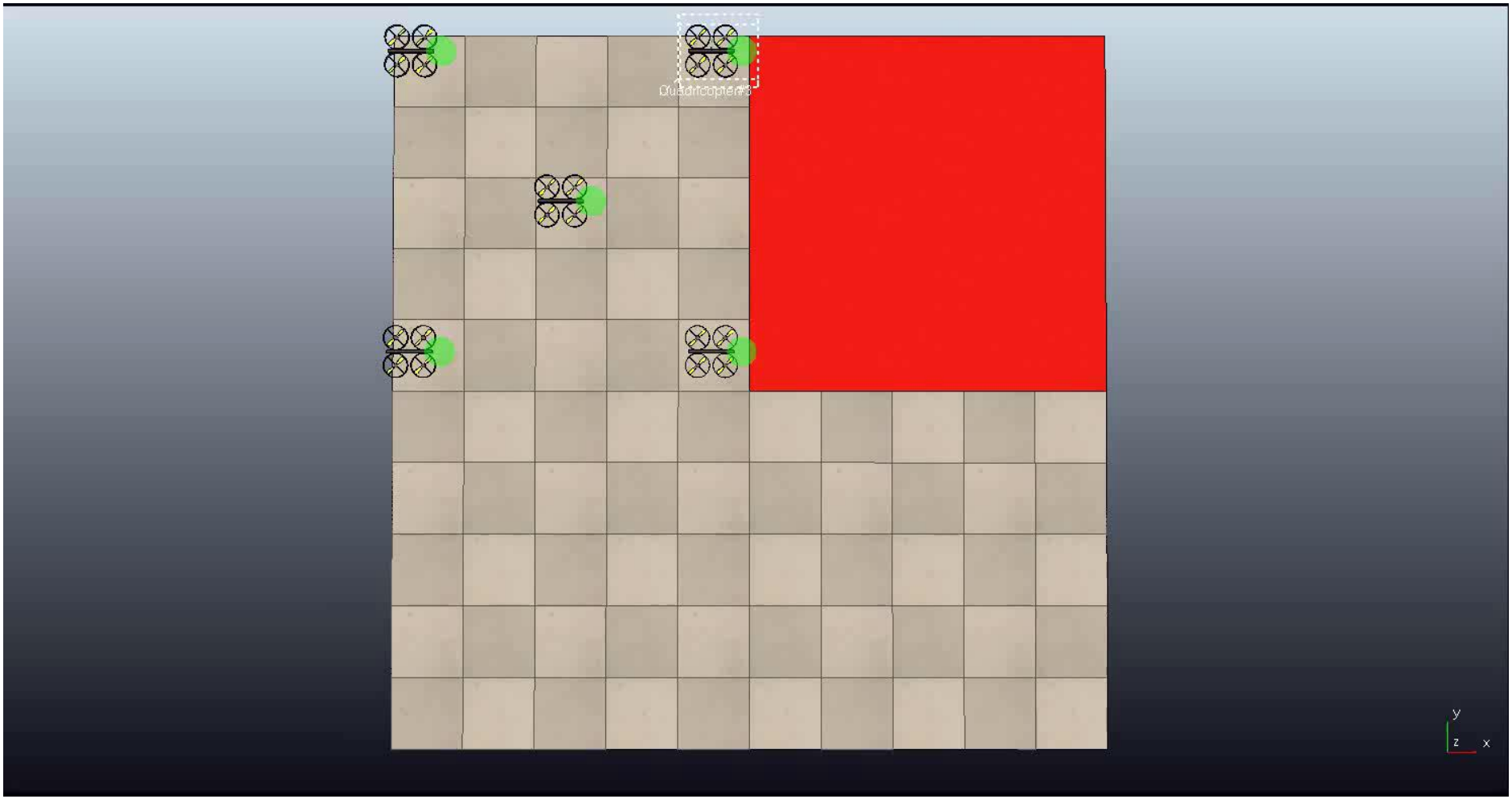
```
node uav {
  local input int x,y;
  local int xp=x, yp=y;
  global lock[X][Y] = {…}

  role Leader {
    thread COLLISION_AVOIDACE {…}
    thread WAYPOINT {…}
    thread ADAPTATION_MANAGER {…}
  }

  role Protector {
    thread COLLISION_AVOIDACE {…}
    thread WAYPOINT {…}
  }
}
```

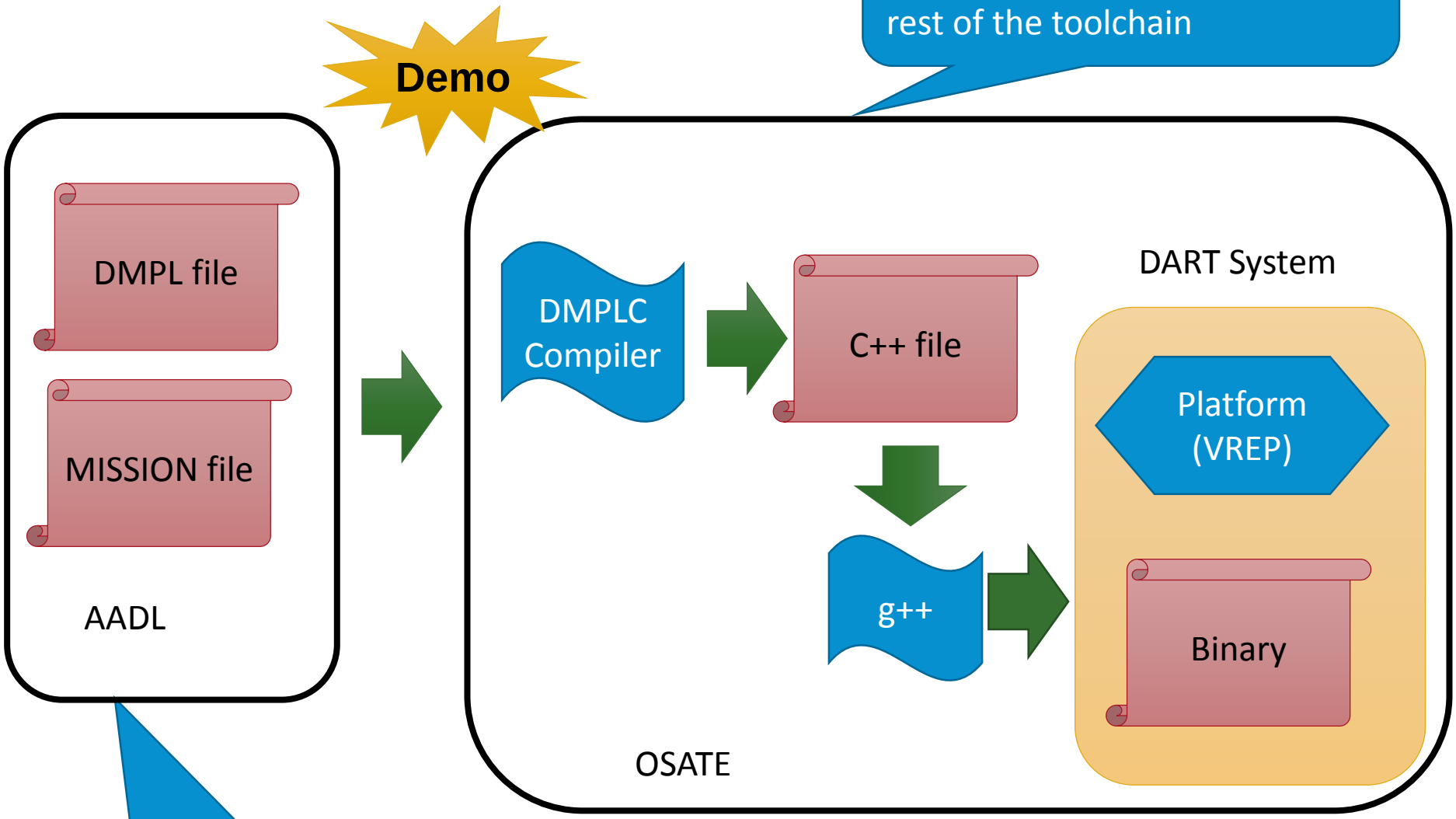
Shared between threads on the same node. Used to communicate next waypoint.

Shared between threads on different nodes. Used for collision avoidance,



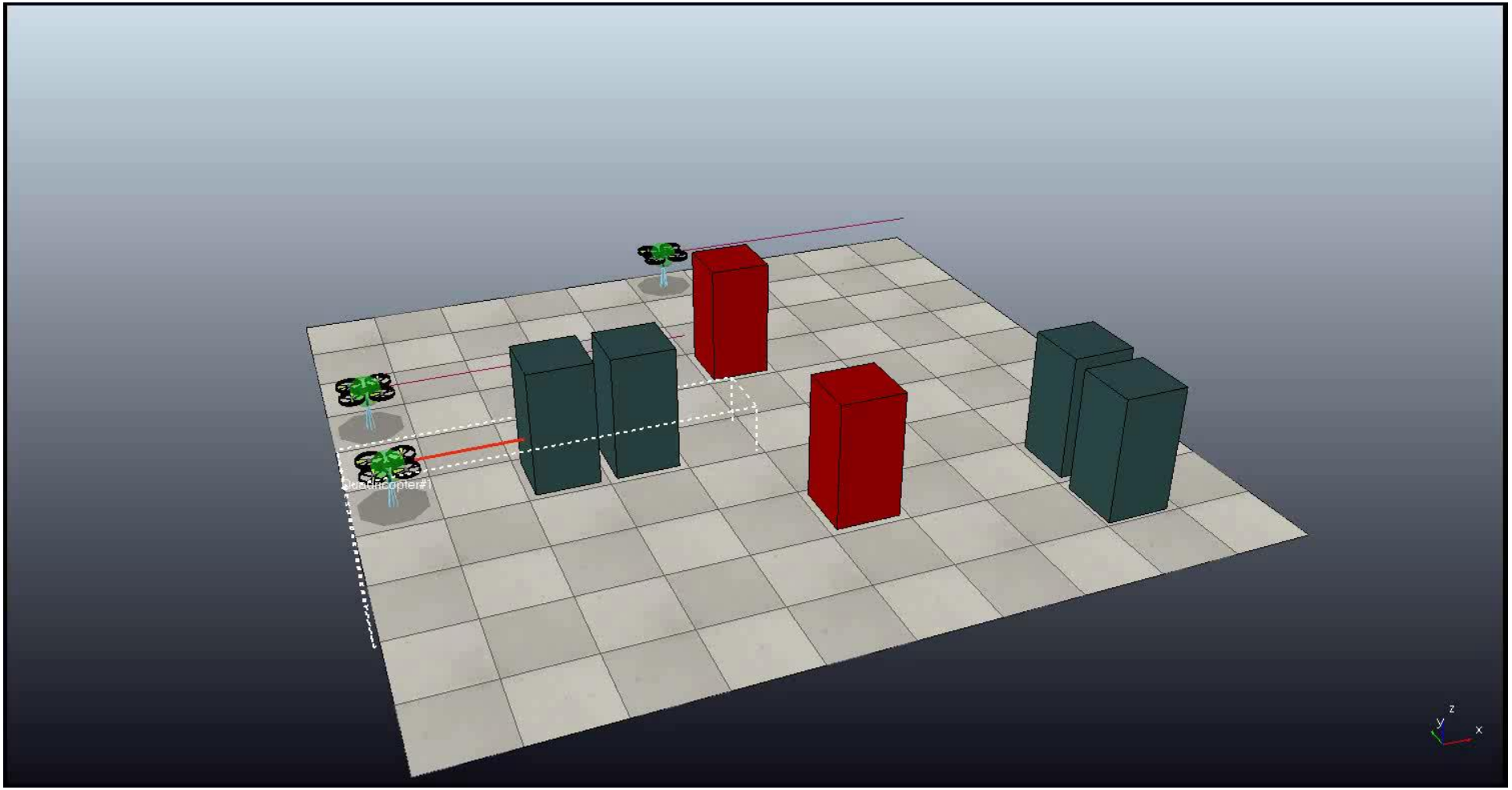


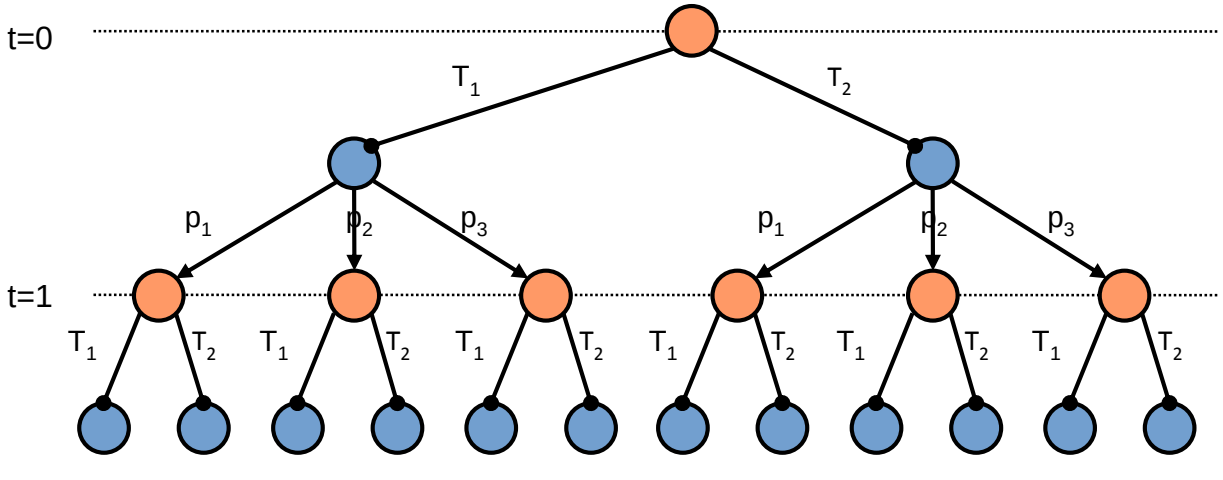


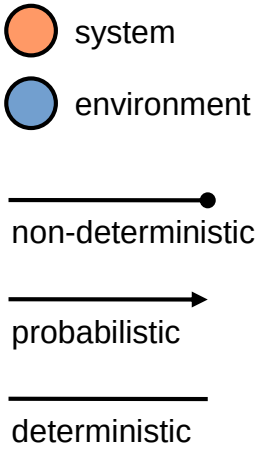


Scenarios

- ✓ Stage 0 – basic 3D collision avoidance
- ✓ Stage 1 – Navigation of "ensemble" from Point A to Point B
- ✓ Stage 2 – Navigation of "ensemble" from Point A to Point B through intermediate waypoints
- ✓ Stage 3: Add detection of solid objects, obstacles
  - Assume unobstructed path exists between Point A and Point B
  - Navigation of "ensemble" from Point A to Point B
- ✓ Stage 4: "Map" obstructions in a 3D region
- ✓ Stage 5
  - Add ability to detect location of potential "threats" (analogous to identifying IFF transponders)
  - "Map" threats and obstructions in 3D region
- Stage 6
  - Add mobility to "threats"
  - Maintain overwatch of region and keep track of location of "threats" that move in the environment

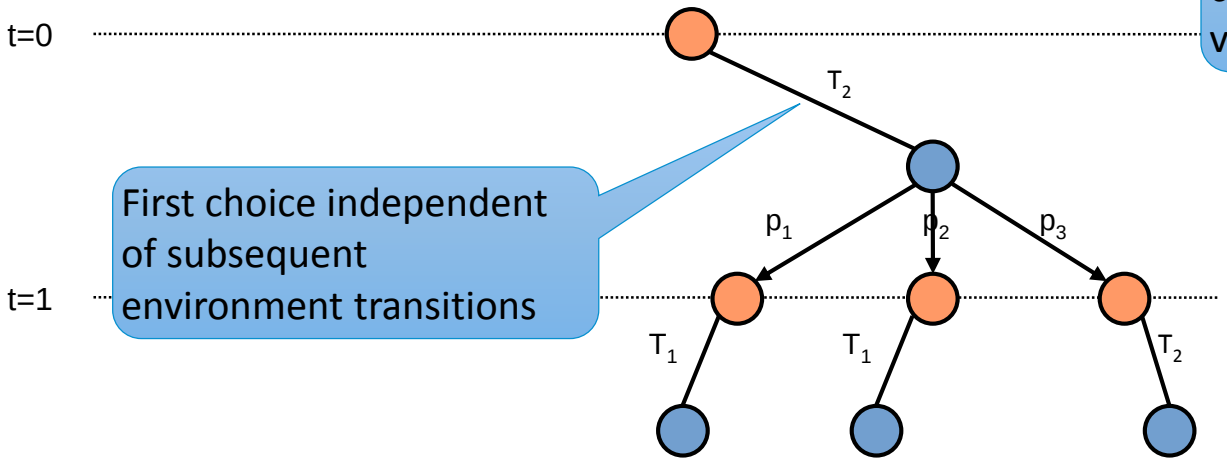






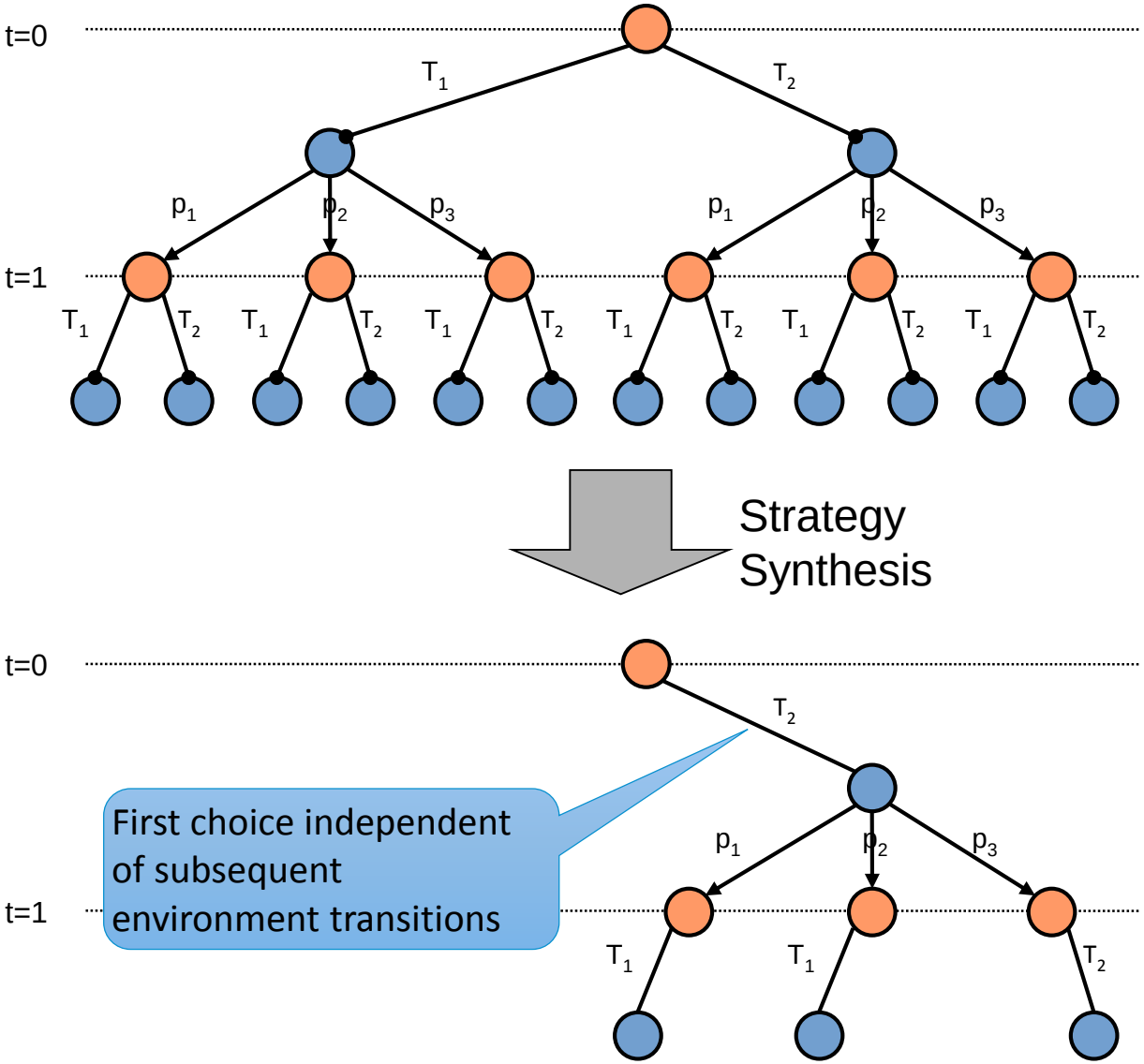
Strategy Synthesis

Resolves nondeterministic choices to maximize expected value of objective function



First choice independent of subsequent environment transitions

Original Work: Use PRISM model checker to synthesize optimal strategy
New work: replace probabilistic model checking with dynamic programming for speed.



system

environment

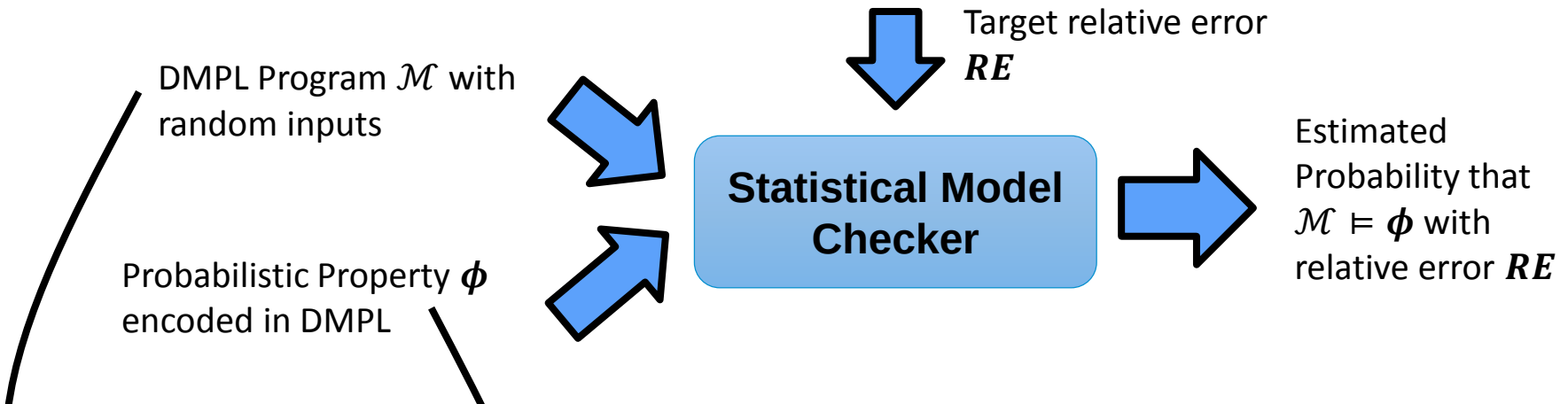
Proactive self-adaptation under uncertainty: a probabilistic model checking approach. Gabriel A. Moreno, Javier Cámara, David Garlan, Bradley R. Schmerl: ESEC/SIGSOFT FSE 2015: 1-12

Efficient Decision-Making under Uncertainty for Proactive Self-Adaptation. Gabriel A. Moreno, Javier Camara, David Garlan, Bradley Schmerl. In proceedings of the 13th IEEE International Conference on Autonomic Computing, 2016.

Original Work: Use PRISM model checker to synthesize optimal strategy

New work: replace probabilistic model checking with dynamic programming for speed.

Oct 21, 2016




```
node uav {
  local input int x,y;
  local int xp=x, yp=y;
  role Leader {…}
  role Protector {…}
  double coverage() {…}
  expect at_end (coverage() > 0.8);
}
```

Etimate probability for each property via "Bernoulli Trials"

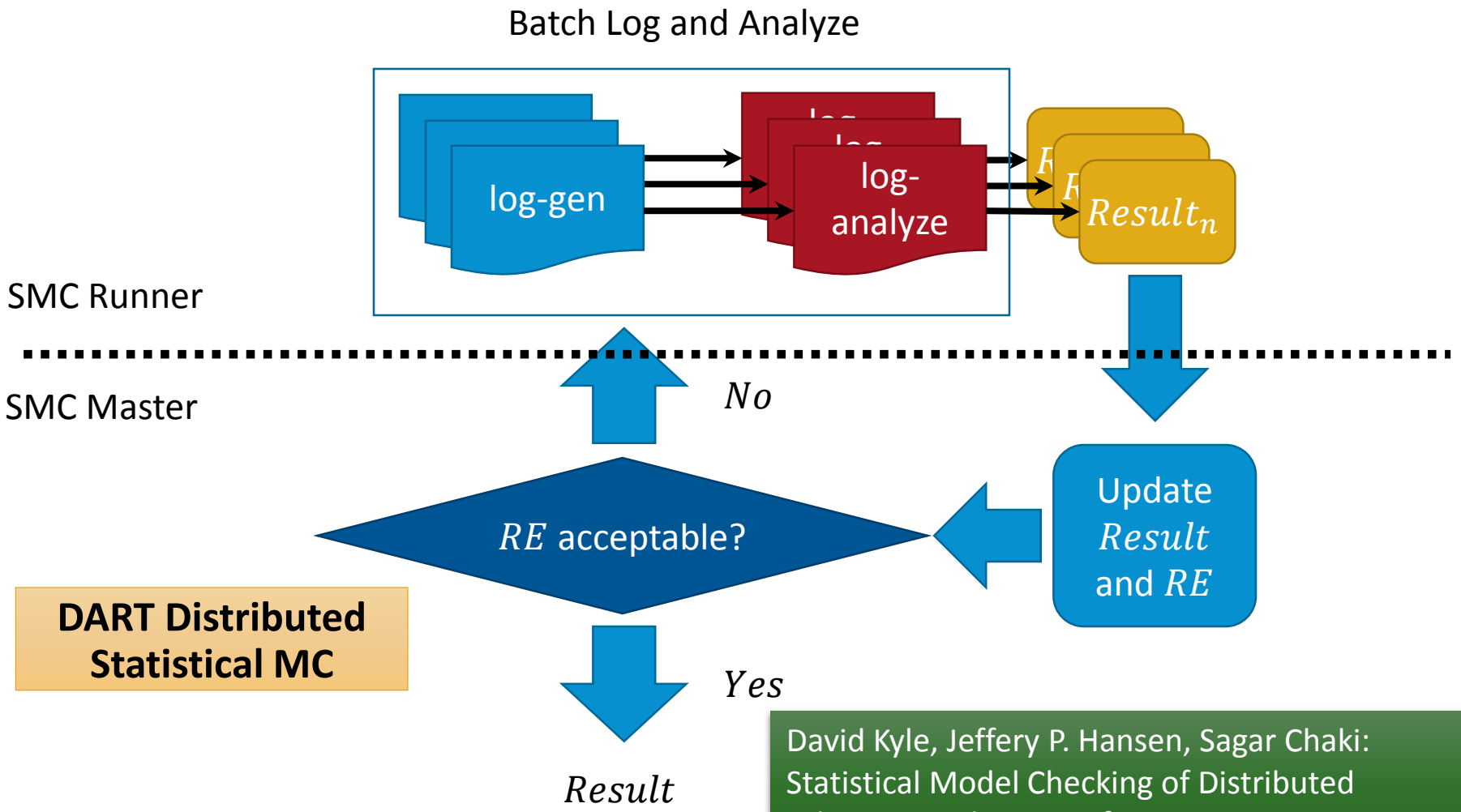
Number of trials depends on

- desired "relative error" (st.dev. / mean)
- true probability of the property

Running trials in parallel reduces required simulation time.

- *SMC Runner* invokes Vrep simulation on each node.
- *SMC Master* collects results and determines if precision is met.
- Simulations run in "batches" to prevent simulation time bias.

Importance sampling (focuses simulation effort on faults)



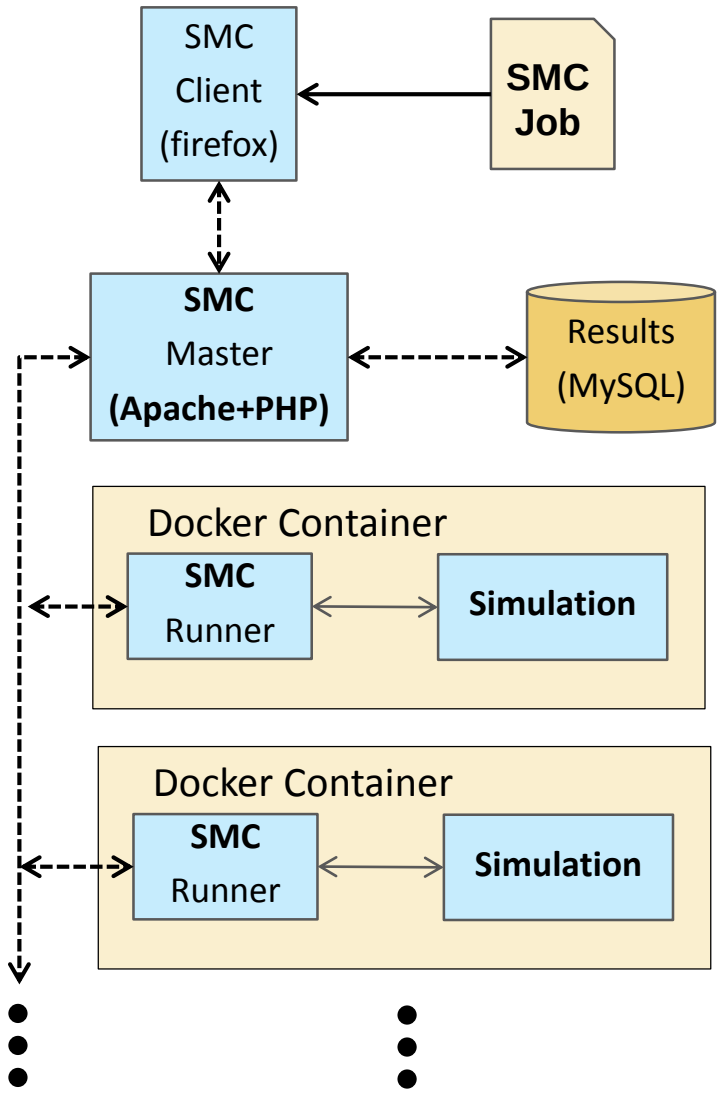
DART Distributed Statistical MC

David Kyle, Jeffery P. Hansen, Sagar Chaki: Statistical Model Checking of Distributed Adaptive Real-Time Software. RV 2015: 269-274

Goal: Develop parallel infrastructure for SMC of DART systems

Accomplishments:

- Initial implementation with hand-written scripts for managing multiple virtual machines
- Created master-client SMC architecture with web-based control
  - Each client runs a simulation managed by master
  - Results stored in mysql database.
- Update SMC code generation to new DART/DMPL syntax
- More robust infrastructure using "docker"
- **Input Attribution** – the "Why?" of SMC



Jeffery P. Hansen, Sagar Chaki, Scott A. Hissam, James R. Edmondson, Gabriel A. Moreno, David Kyle: Input Attribution for Statistical Model Checking Using Logistic Regression. RV 2016: 185-200

Logistic Regression Model:

$$L(x) = \frac{1}{1 + e^{-(\beta_0+\beta_1 x_1+\beta_2 x_2+\cdots+\beta_N x_N)}}$$

$L(x)$ represents predicted probability that input x will satisfy the predicate.

Positive/negative values represent increase/decrease of predicate probability.

- Measure of statistical significance
- Probability that $\beta = 0$
- >0.05 → not statistically significant

Name	$\hat{\beta}$	$se(\hat{\beta})$	p-Value
–	-4.28	0.874	$< 10^{-4}$
x_1	0.154	0.0138	$< 10^{-4}$
x_2	-1.91	0.3551	$< 10^{-4}$
x_3	0.0635	0.0277	0.0219
x_4	5.05	2.77	0.0685

Constant Term

Predictors (input variables)

Error in estimation of β.

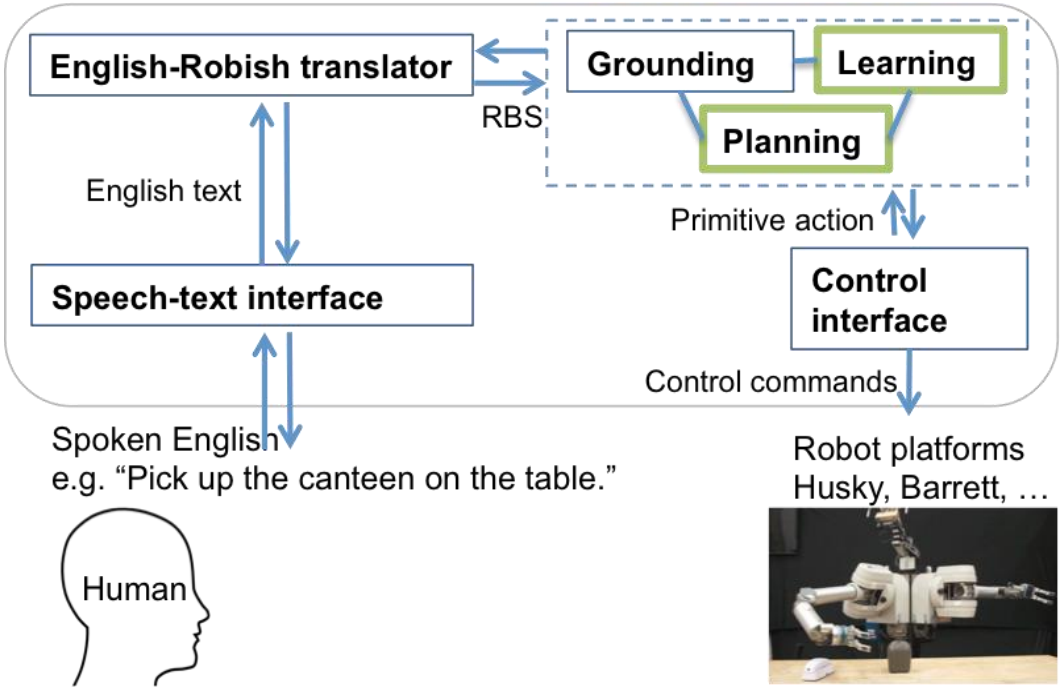
This predictor is not statistically significant since its p-value is greater than 0.05.

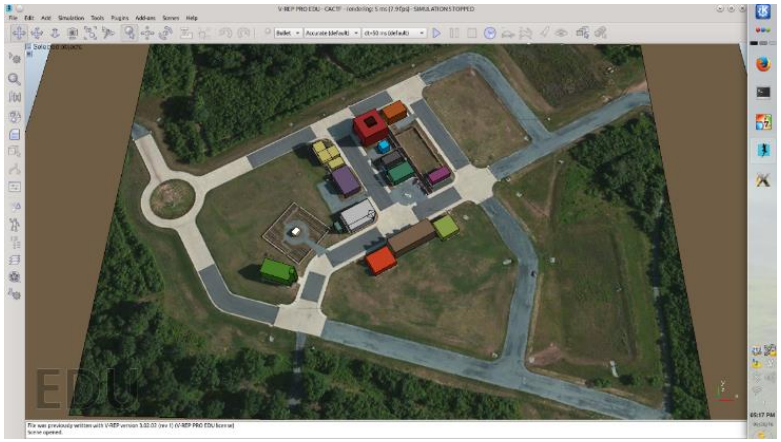
Jeffery P. Hansen, Sagar Chaki, Scott A. Hissam, James R. Edmondson, Gabriel A. Moreno, David Kyle: Input Attribution for Statistical Model Checking Using Logistic Regression. RV 2016: 185-200

Evaluating quality of plans learned from verbal instructions by a robot using statistical model checking

Collaborative work with NREC

- **Part of ARL sponsored Robotics Collaborative Technology Alliance (RCTA)**





WCET may be uncertain in autonomous systems (e.g. more obstacles larger WCET).

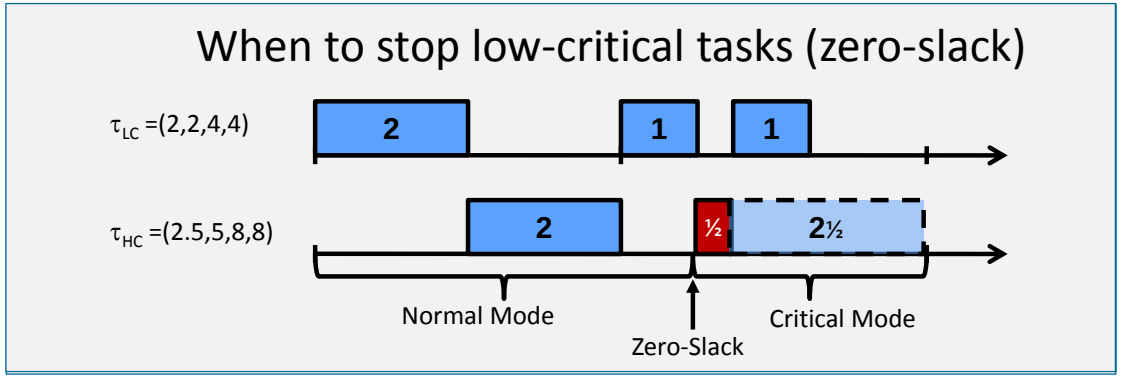
ZSRM: if no overload all task meet deadlines
if overload critical tasks meet deadlines
How: 1. when to stop low-critical tasks (Z)
2. stop them if not overload resume

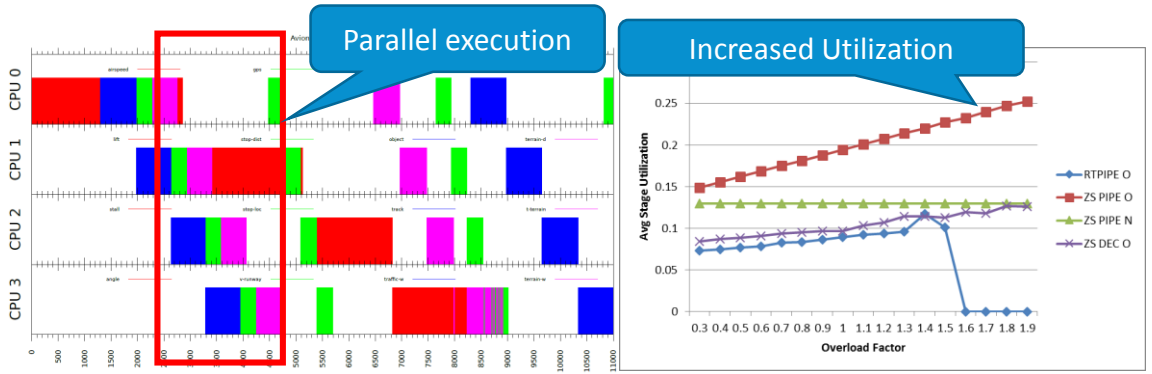
DART: requires distributed tasks

Accomplishments:

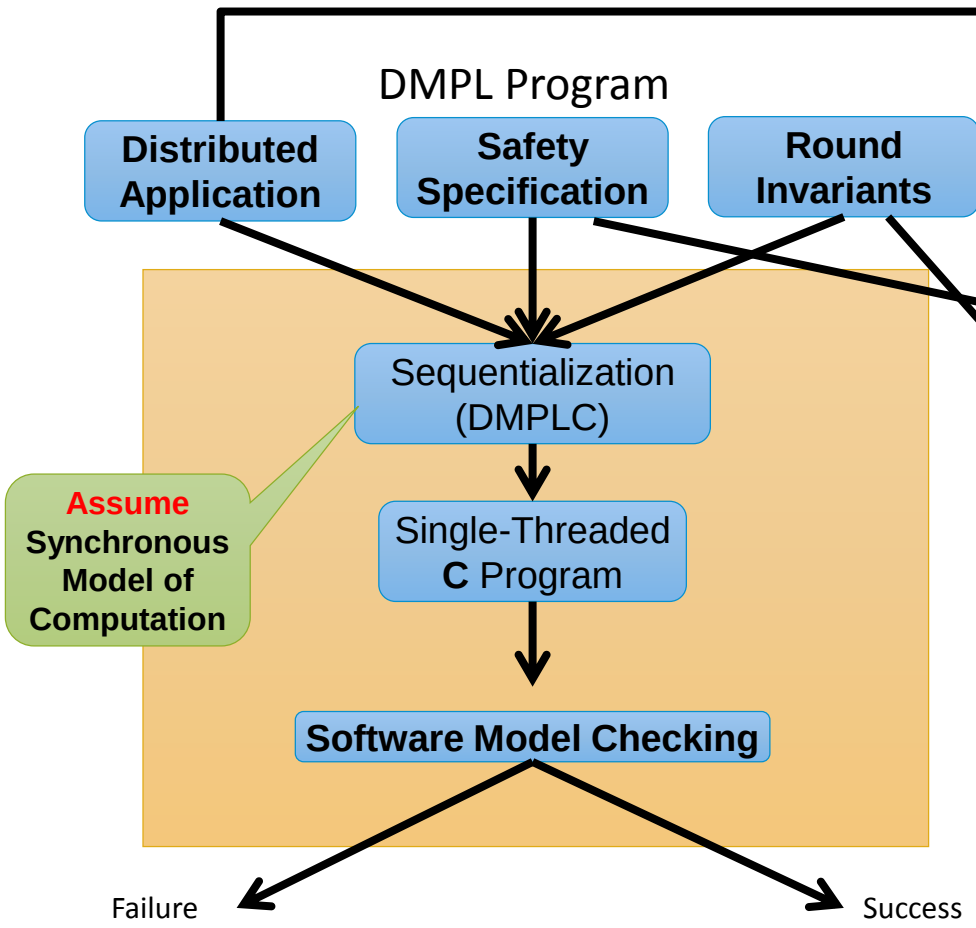
ZSRM Pipelines:

- Enforcement across processor
- Higher utilization





Verifying Safety Properties




```
node uav {
  local input int x,y;
  local int xp=x, yp=y;
  role Leader {…}
  role Protector {…}

  forall_distinct_nodes(i1,i2)
     (x@i1 != x@i2 || y@i1 != y@i2);

  forall_nodes(i)
     (x@i == xp@i || y@i == yp@i);
}
```

Bounded Model Checking can prove correct behavior up to **a finite number of execution steps** (e.g., rounds of synchronous computation.

Useful to find bugs.

But incomplete. Can miss bugs if we do not check up to sufficient depth.

Unbounded Model Checking can prove correct behavior up to a **arbitrary number of execution steps.**

Useful for complete verification. Will never miss bugs.

But can be expensive to synthesize inductive invariants. Cost can be managed by supplying invariants manually and checking that they are inductive. We have experimented with both approaches.

Parameterized Model Checking can prove correct behavior up to a arbitrary number of execution steps and an **arbitrary number of nodes.**

Useful for complete verification. Will never miss bugs even if you have very large number of nodes.

Very hard in general but we have developed a sound and complete procedure that works for programs written in a restricted style and for a restricted class of properties. This was sufficient to verify our collision avoidance protocol.

Verifying Cyber-Physical Systems by Combining Software Model Checking with Hybrid Systems Reachability

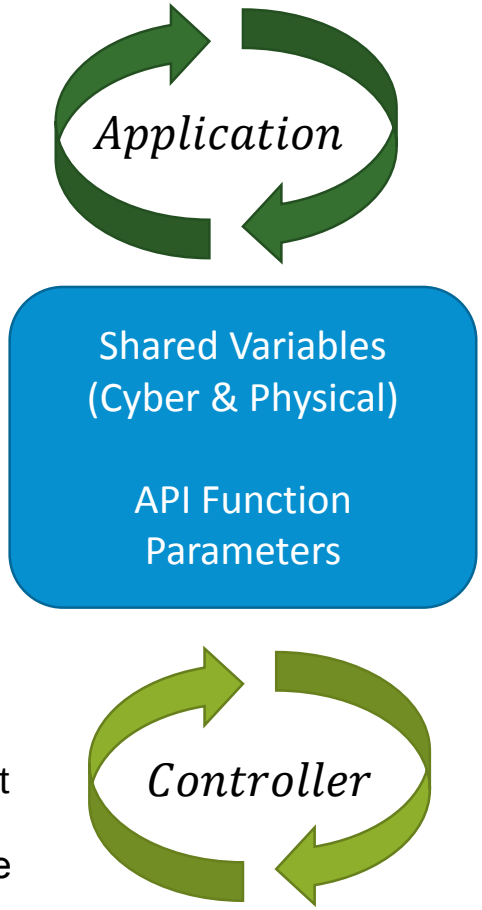
No existing tools to verify (source code + hybrid automata)

- But each domain has its own specialized tools: software model checkers and hybrid reachability checkers
- Developing such a tool that combines the statespace A and C in a brute-force way will not scale

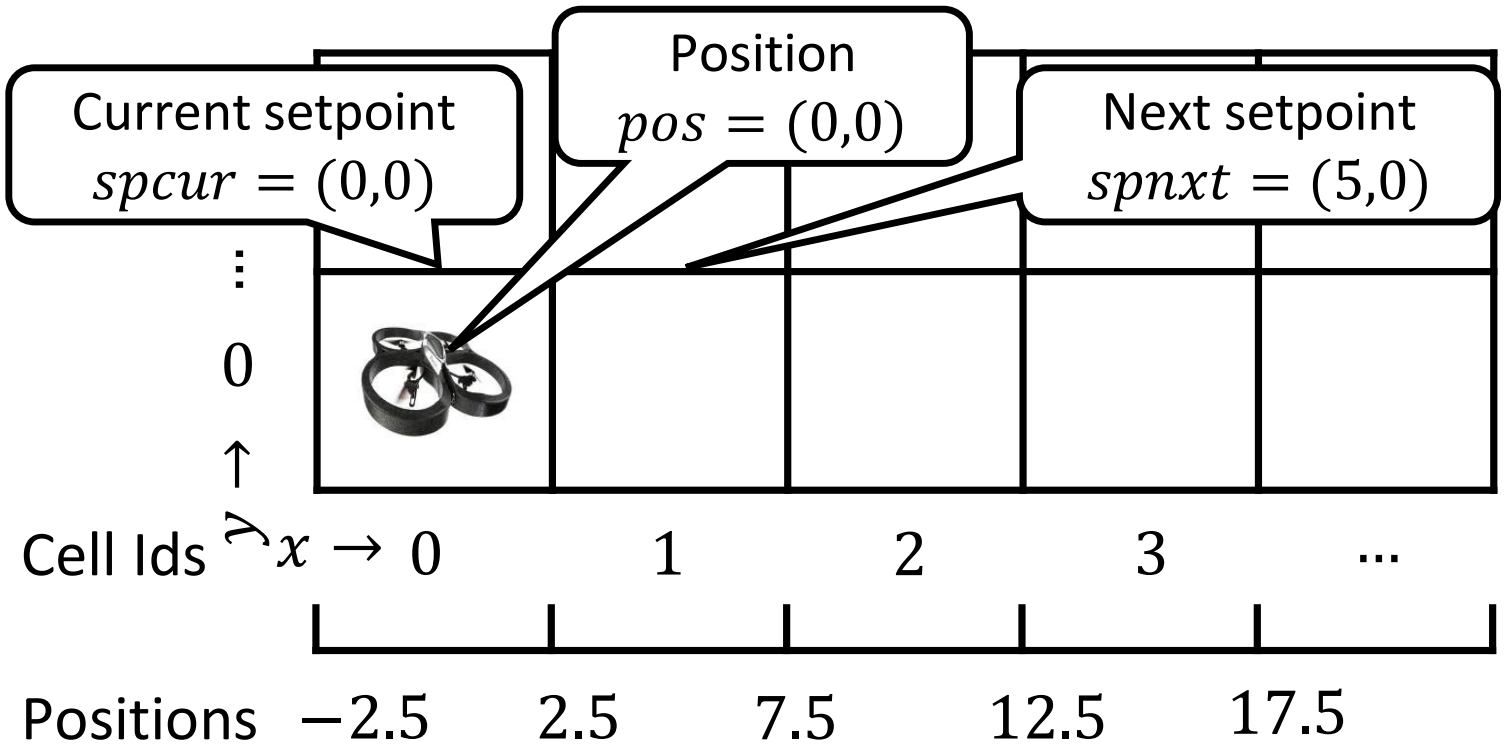
Insight: application and controller make assumptions about each other to achieve overall safe behavior

Approach:

- Use "contract automaton" to express inter-dependency between A and C
- Separately verify that A and C implement desired behavior under the assumption that the other party does so as well
- Use an "assume-guarantee" style proof rule to show the $A \parallel C \vDash \Phi$



Verifying Cyber-Physical Systems by Combining Software Model Checking with Hybrid Systems Reachability. Stanley Bak, Sagar Chaki. International Conference on Embedded Software (EMSOFT), 2016



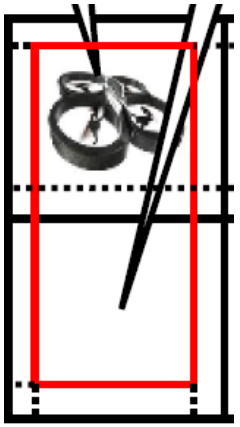
(C1) The application always calls $update\_setpoint(x, y)$, with arguments that satisfy the condition $|(x, y) - spcur| = (5, 0) \vee |(x, y) - spcur| = (0, 5)$.

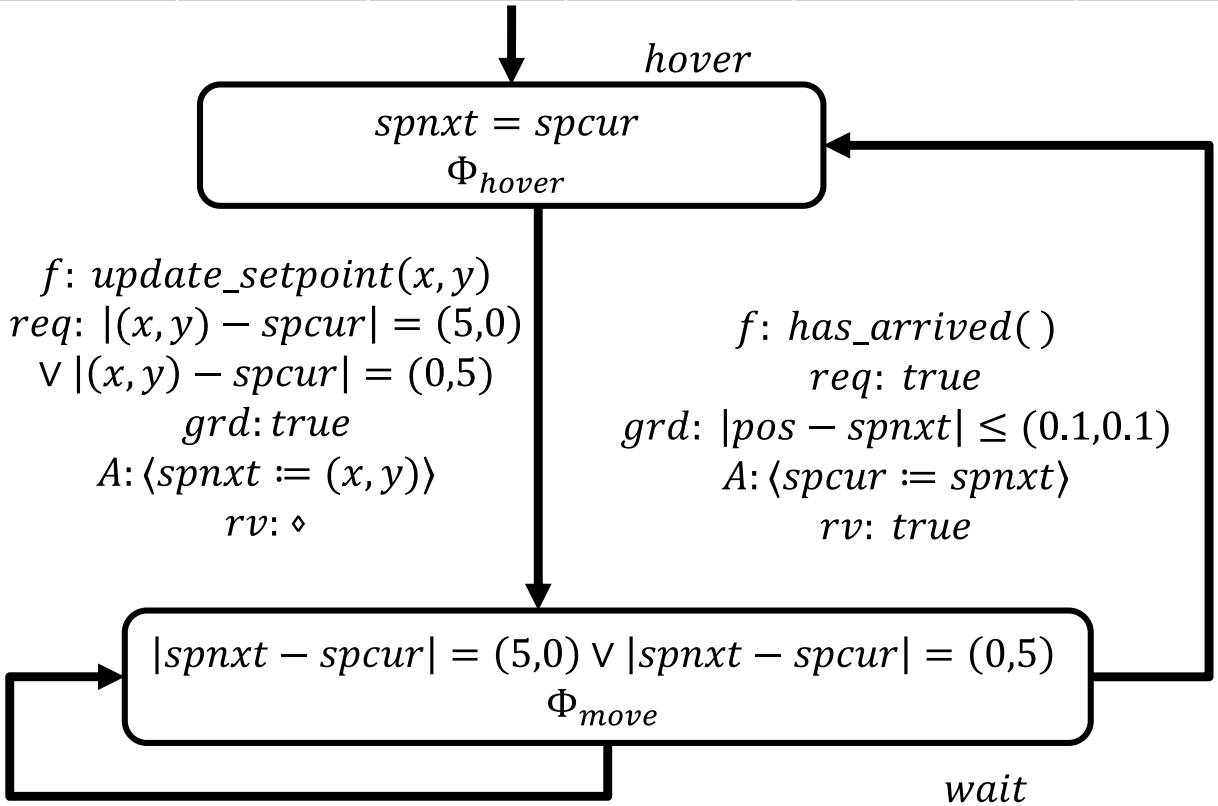
(C2) Once the application calls $update\_setpoint(x, y)$, it can keep calling $has\_arrived()$ until it gets a return value of TRUE; once $has\_arrived()$ returns TRUE, the application can only then start to call $update\_setpoint(x, y)$ again.

(C3) When the quadcopter is hovering (i.e., $spnxt = spcur$), the controller must maintain the following invariant: $\Phi_{hover} \equiv |pos - spcur| \leq (1.5, 1.5)$.

(C4) When the quadcopter is moving (i.e., $|spnxt - spcur| = (5, 0) \vee |spnxt - spcur| = (0, 5)$), the controller must maintain the following invariant:

$$\begin{aligned} \Phi_{move} \equiv\ & min(spcur_x, spnxt_x) - 1.5 \leq pos_x \\ & \leq max(spcur_x, spnxt_x) + 1.5 \\ \wedge\ & min(spcur_y, spnxt_y) - 1.5 \leq pos_y \\ & \leq max(spcur_y, spnxt_y) + 1.5 \end{aligned}$$





Target Property

CA Invariant = disjunction of state invariants

$$(\Phi_{hover} \wedge spnxt = spcur) \vee (\Phi_{move} \wedge \begin{pmatrix} |spnxt - spcur| = (5,0) \vee \\ |spnxt - spcur| = (0,5) \end{pmatrix})$$

Premise1: Application A refines the contract automaton M (calls API functions in the right order and with proper arguments)

Premise2: Controller C refines the contract automaton M (keeps the physical state within required bounds)

THEOREM 1 (COMPOSITIONAL REFINEMENT).

$$\frac{A \preceq M \qquad C \preceq M}{A \parallel C \preceq M}$$

Conclusion: System satisfies all invariants of the contract automaton M = target safety property

$$(\Phi_{hover} \wedge spnxt = spcur) \vee (\Phi_{move} \wedge (|spnxt - spcur| = (5,0) \vee |spnxt - spcur| = (0,5)))$$

Future Work

Related Ongoing Work

Verification of Software with Timers and Clocks (Real Time Schedulers and Enforcers, Distributed Timed Protocols, etc.)

Contract-Based Verification of Timing Enforcers. Sagar Chaki, Dionisio de Niz, ACM SIGAda's High Integrity Language Technology International Workshop on Model-Based Development and Contract-Based Programming (HILT), October 6-7, 2016.

Future Work

Certifiable Distributed Runtime Assurance