Verifying Distributed Adaptive Real-Time (DART) Systems

Sagar Chaki

Dionisio de Niz

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-0004128

DART: Motivation

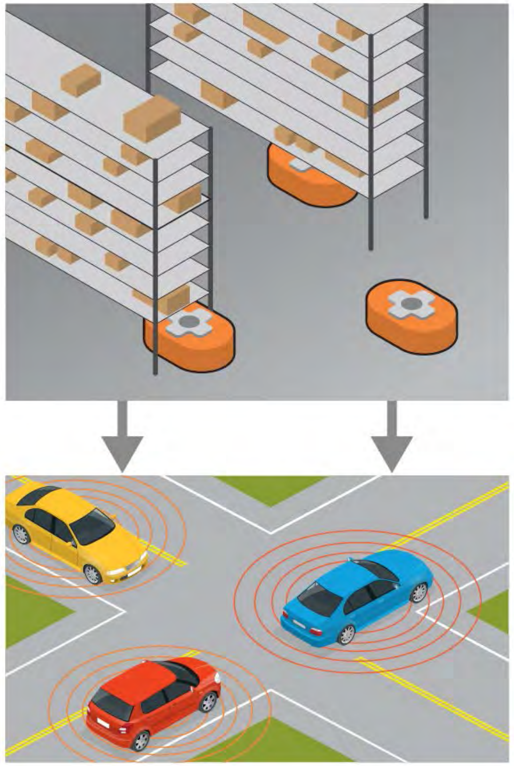
Distributed Adaptive Real-Time (DART) systems are key to many areas of DoD capability (e.g., autonomous multi-UAS missions) with civilian benefits.

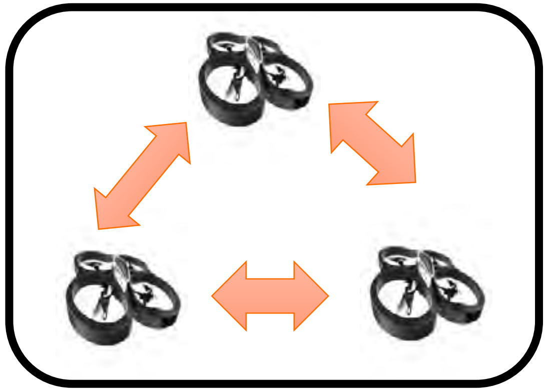
However, achieving high assurance DART software is very difficult

- 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 is unachievable via testing or ad-hoc analysis

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



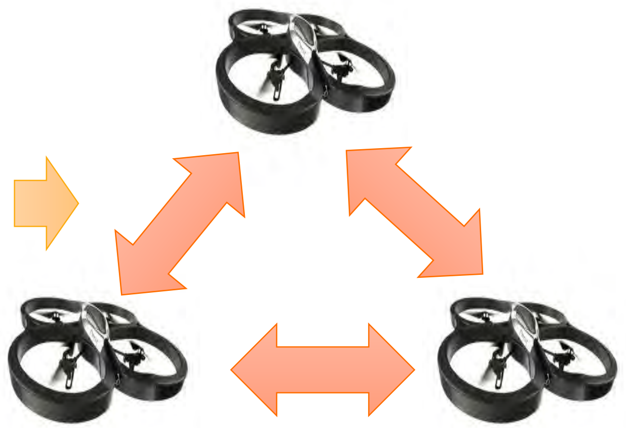


DART Approach

1. Use DMPL (a DSL we developed) and 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 and Properties → **Verification** → **Code Generation** →

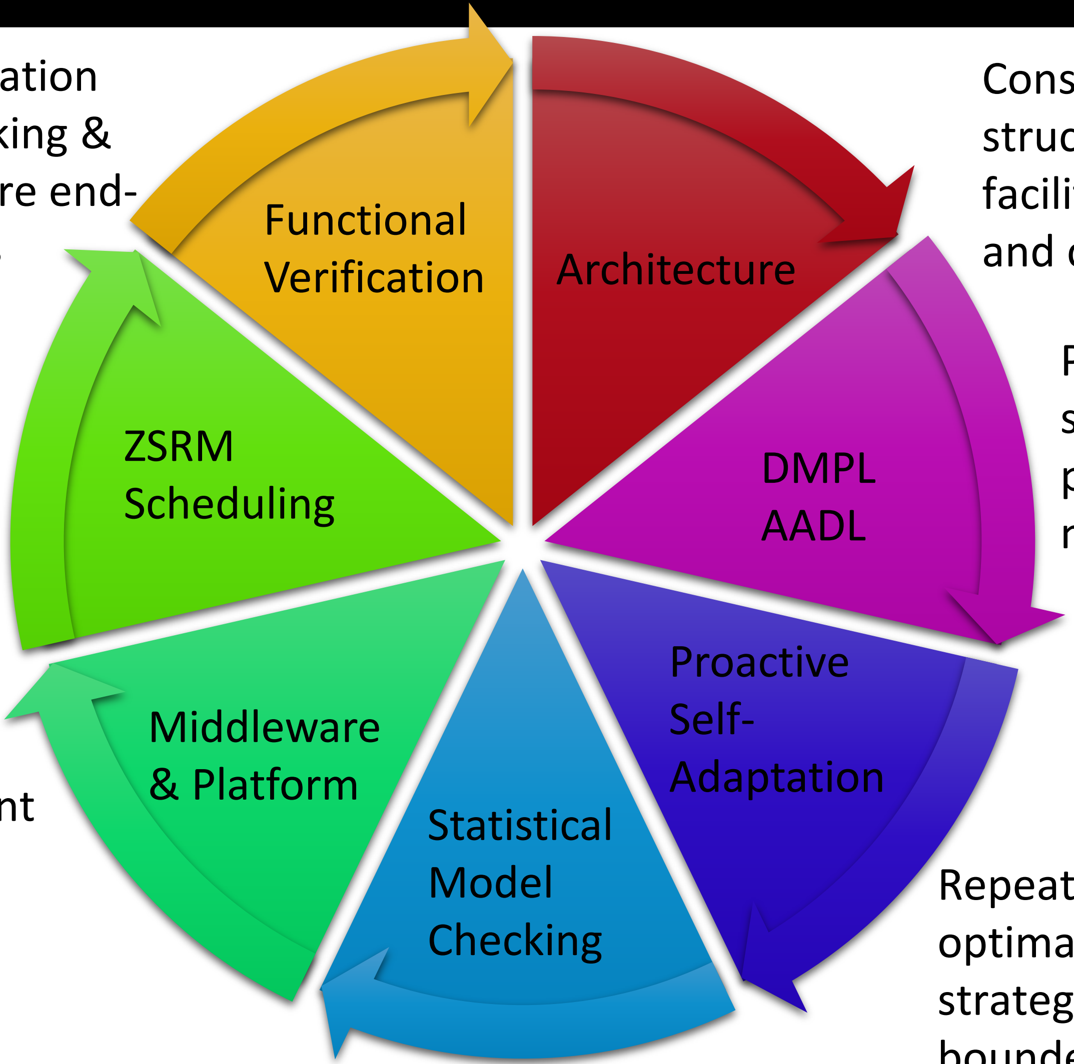


Brings Assurance to Code
1. Middleware for communication
2. Scheduler for ZSRM
3. Monitor for runtime assurance

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



Functional Verification — Sagar Chaki, Arie Gurfinkel

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

Architecture

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

DMPL AADL — David Kyle, Scott Hissam, Bud Hammons, Joseph Seibel

Program DART systems and specify properties in a precise manner

Proactive Self-Adaptation — Gabriel Moreno

Repeatedly compute optimal adaptation strategies with bounded lookahead

Statistical Model Checking — Jeffery Hansen

Evaluate adaptation strategy quality over mission lifetime

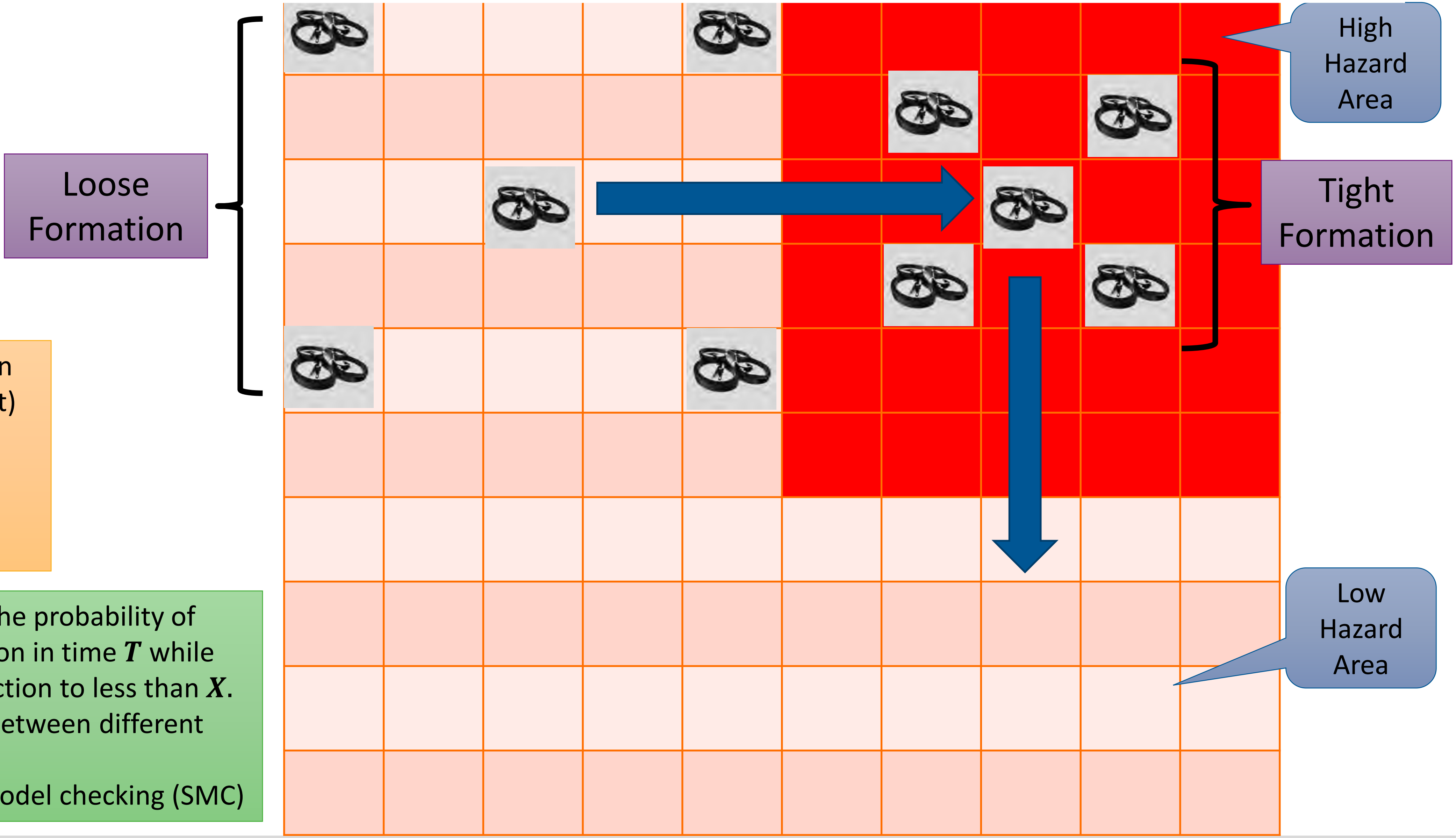
Middleware & Platform — James Edmondson

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

ZSRM Scheduling — Dionisio de Niz, Bjorn Andersson

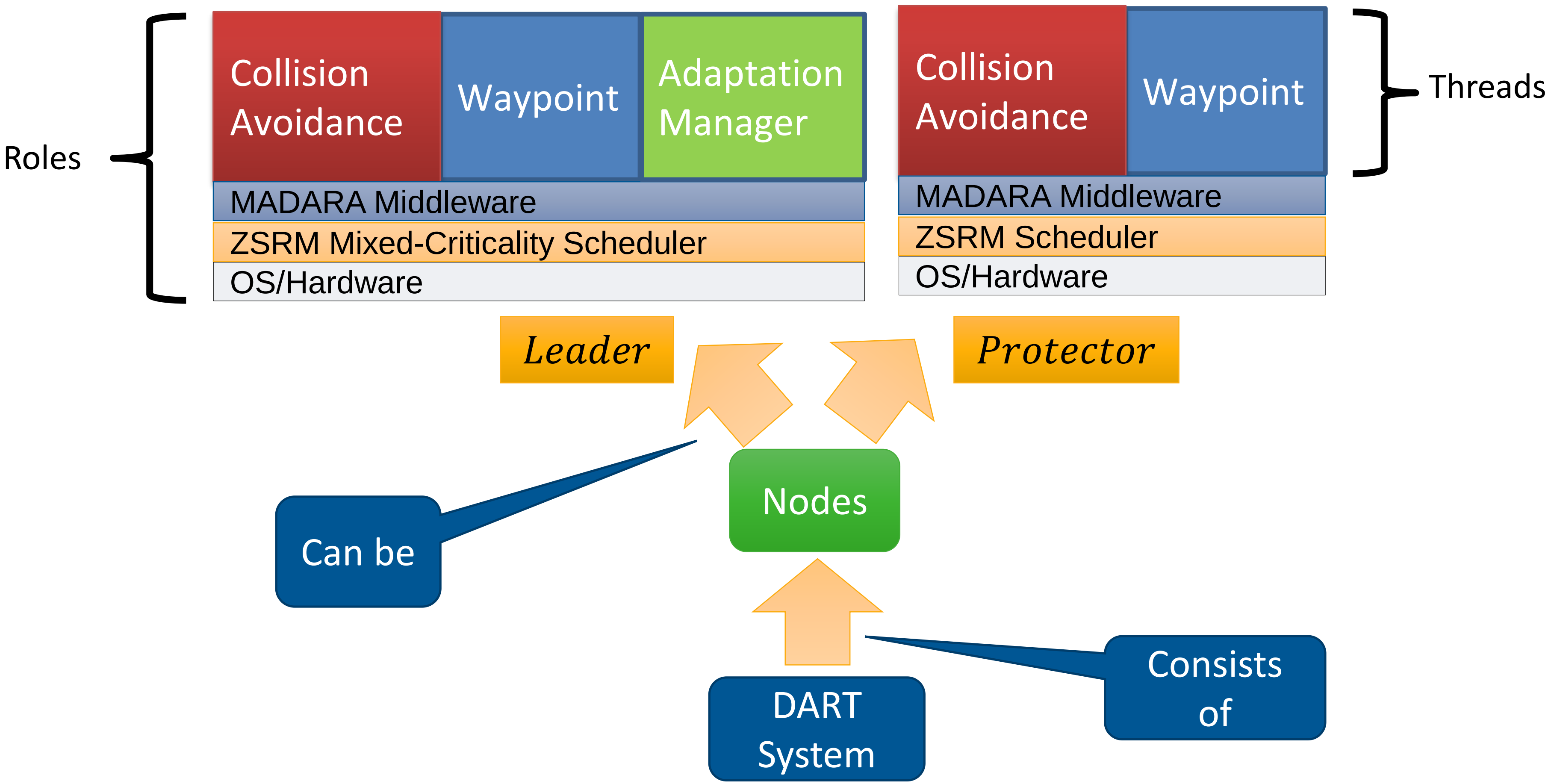
Ensures high-critical tasks meet their deadlines despite CPU overload

Example: Self-Adaptive and Coordinated UAS Protection



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

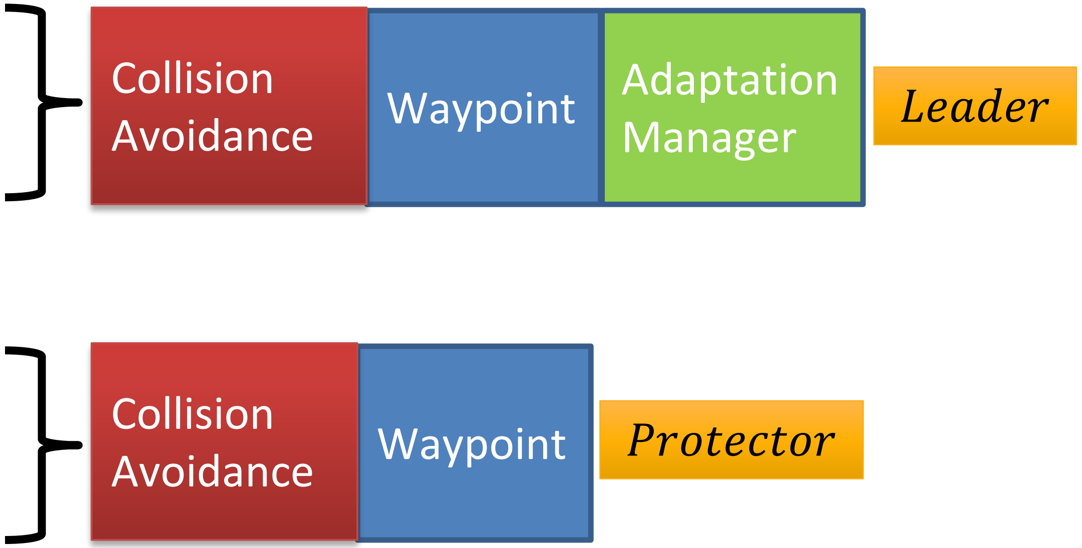
Challenge: compute the probability of reaching end of mission in time T while never reducing protection to less than X.
Challenge: compare between different adaptation strategies.
Solution: Statistical model checking (SMC)

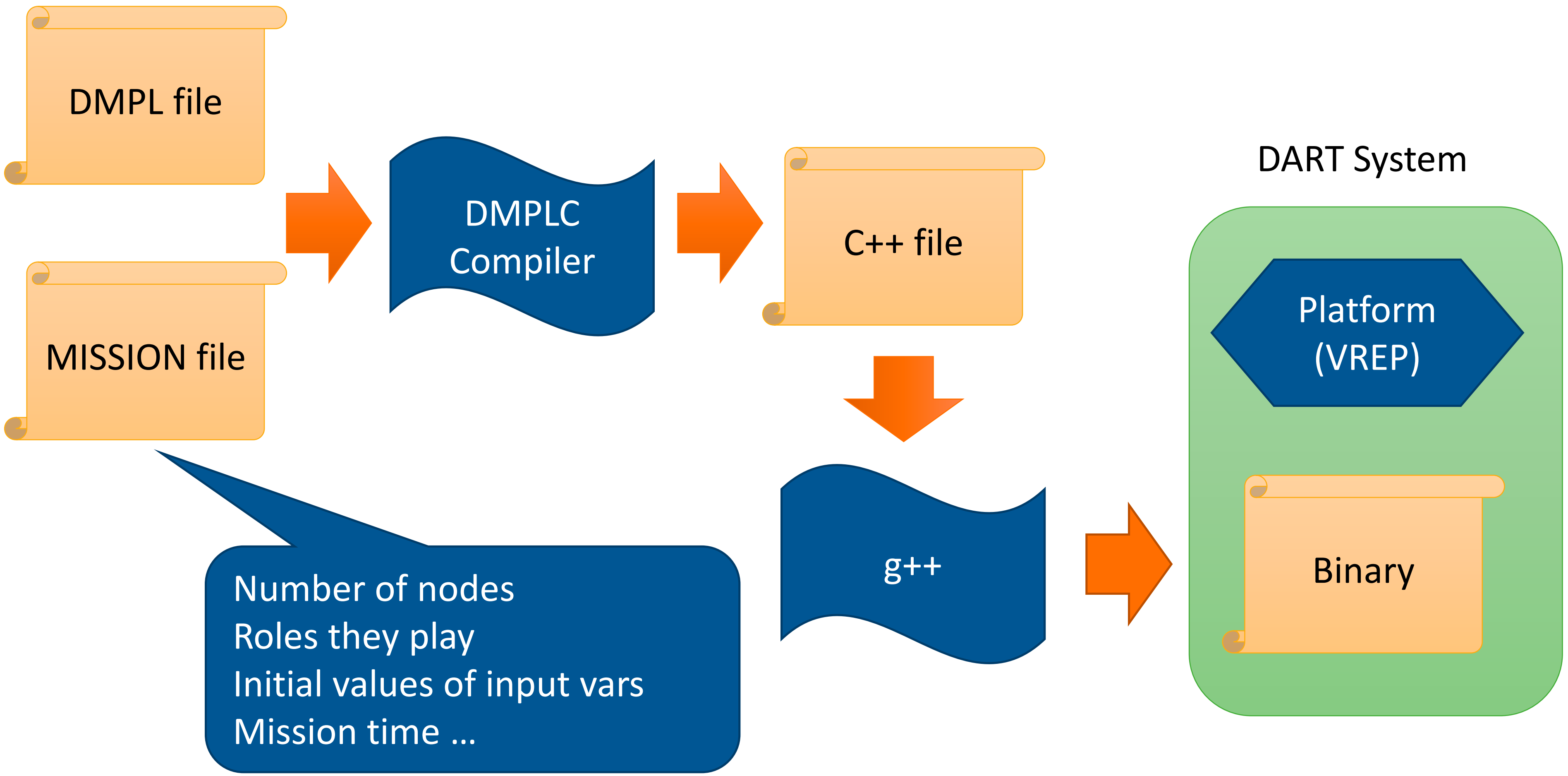

Roles
Collision Avoidance
Waypoint
Adaptation Manager
MADARA Middleware
ZSRM Mixed-Criticality Scheduler
OS/Hardware
Collision Avoidance
Waypoint
Threads
MADARA Middleware
ZSRM Scheduler
OS/Hardware
Leader
Protector
Can be
Nodes
Consists of
DART System

```
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 {…}
  }
}
```

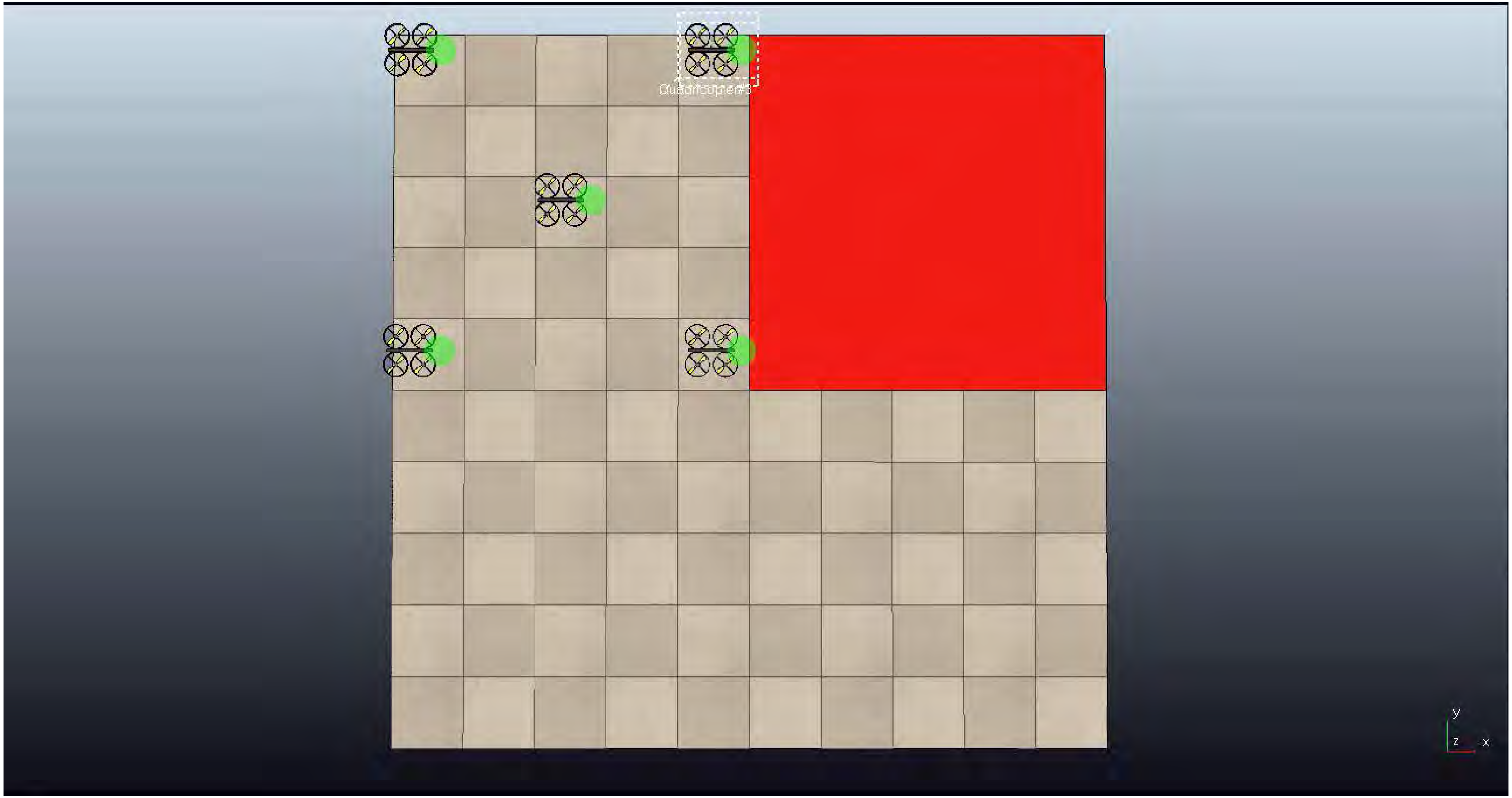
Shared between threads on the same node. Used to communicate next waypoint.

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



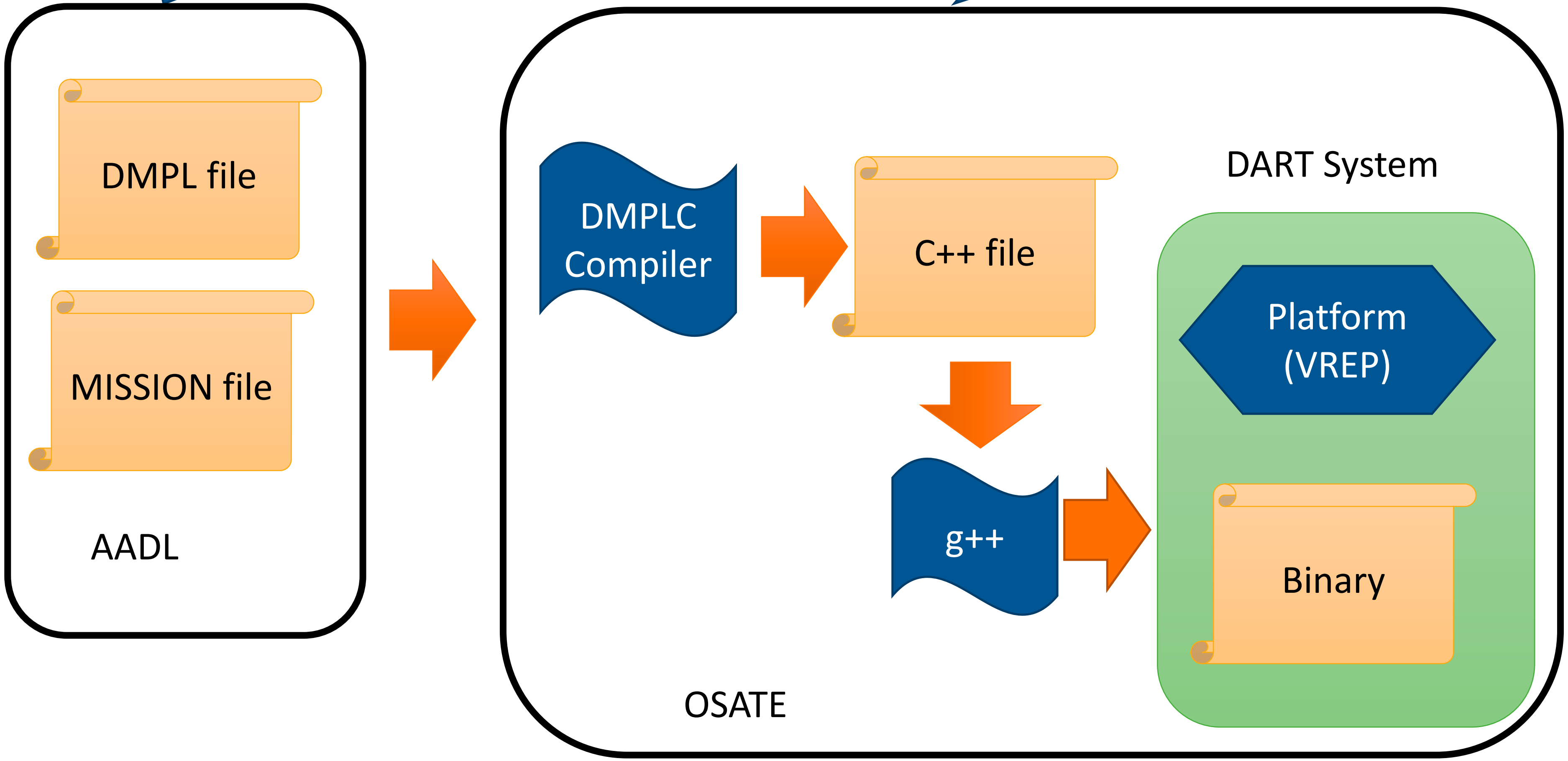

DMPL file
MISSION file
DMPLC Compiler
C++ file
g++
DART System
Platform (VREP)
Binary
Number of nodes
Roles they play
Initial values of input vars
Mission time …

Demo



DMPL and MISSION files expressed in AADL as a sub-language (a.k.a. "annex")

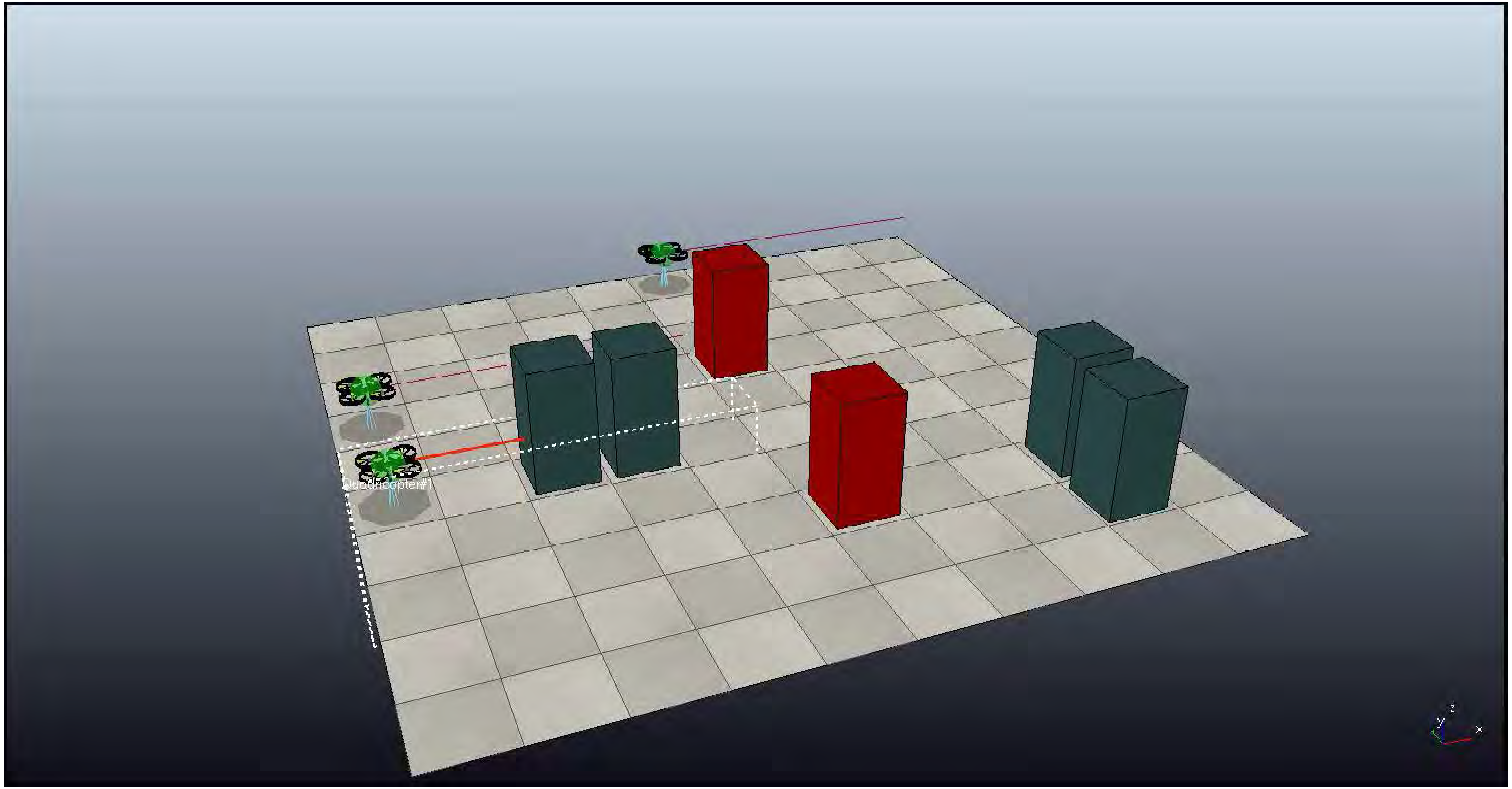
OSATE performs parsing, syntax checking, etc. and invokes the rest of the tool chain



Scenarios

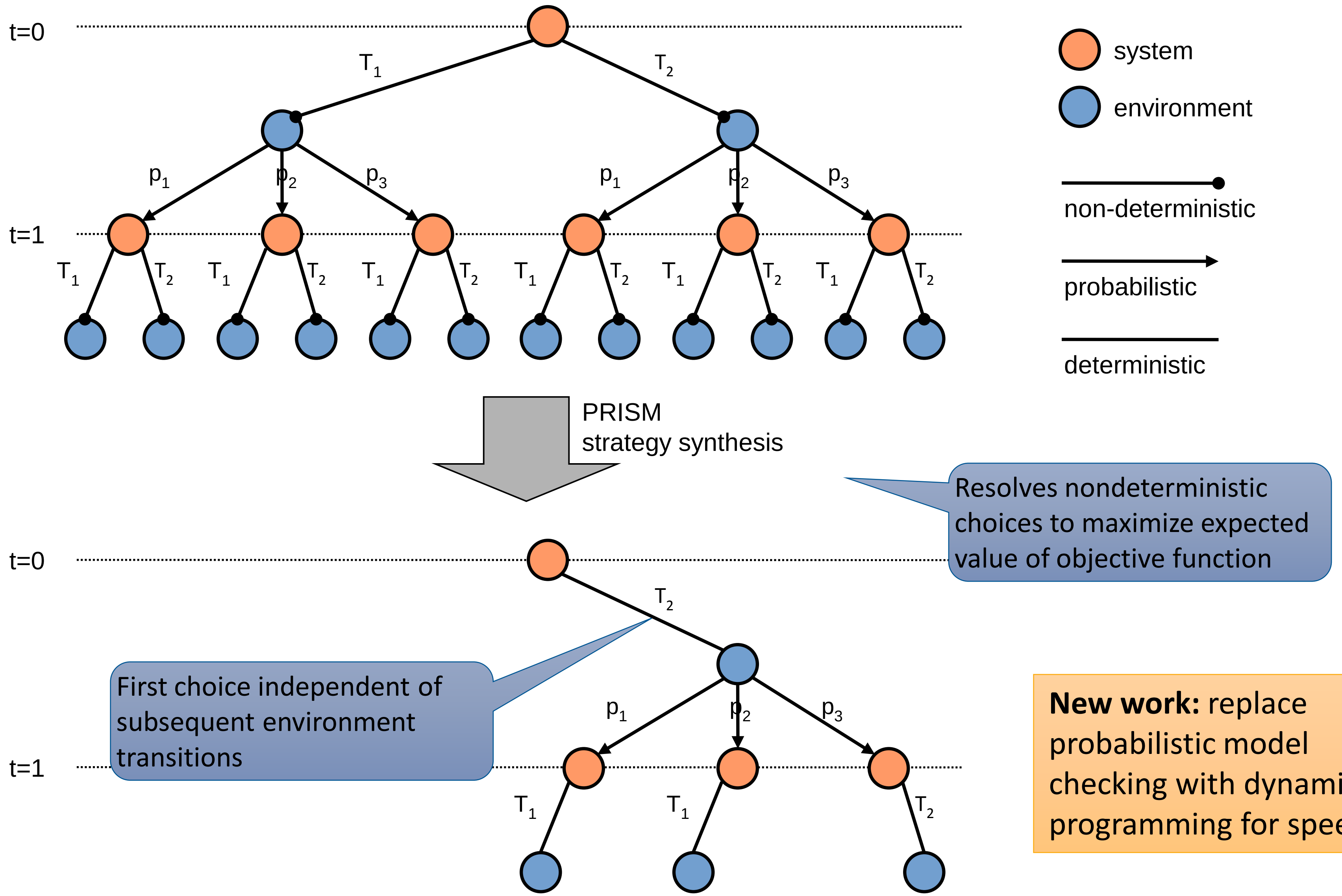
- ✓ Stage 0 – basic 3D collision avoidance
- ✓ Stage 1 – Navigation of “ensemble” from Point A to Point B
- ✓ Stage 2 – Navigation of “ensemble” from Point A to Point B through intermediate waypoints
- ✓ Stage 3: Add detection of solid objects, obstacles
  - Assume unobstructed path exists between Point 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

Demo

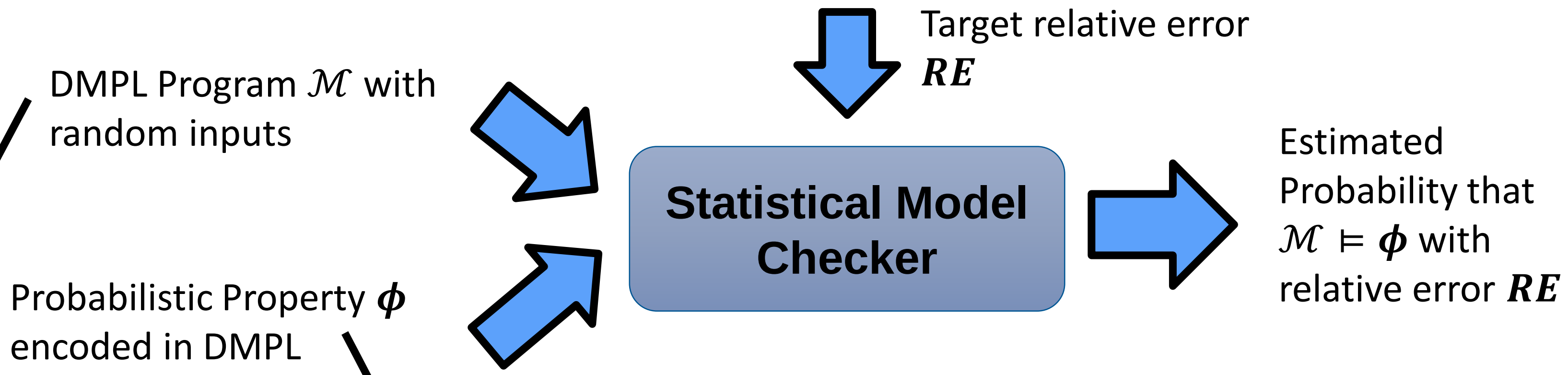


Gabriel A. Moreno, Javier Cámara, David Garlan, Bradley R. Schmerl: Proactive self-adaptation under uncertainty: a probabilistic model checking approach. ESEC/SIGSOFT FSE 2015: 1-12

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.



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




```
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);
}
```

Estimate probability for each property via "Bernoulli Trials"

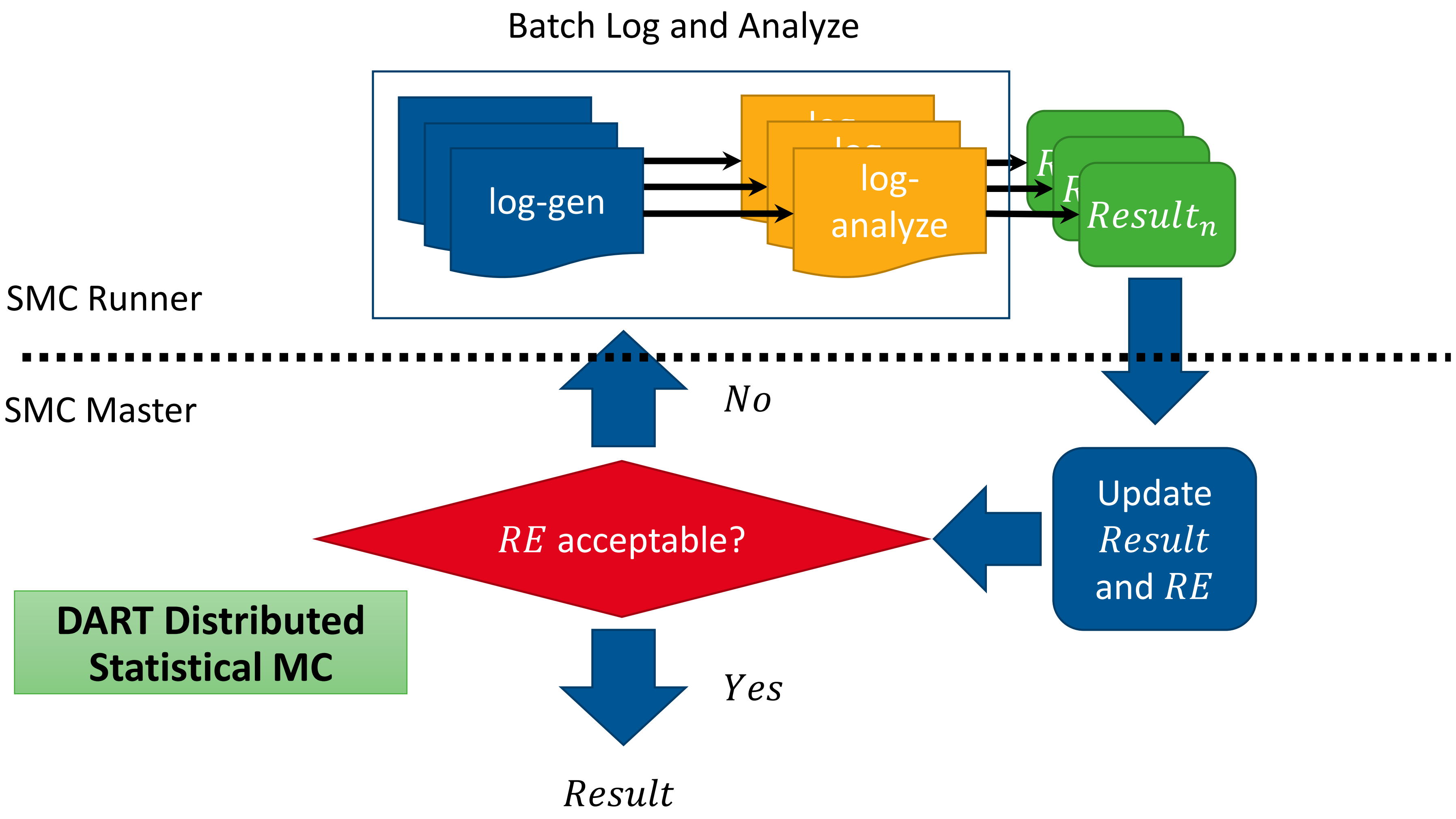
Number of trials depends on

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

Running trials in parallel reduces required simulation time.

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

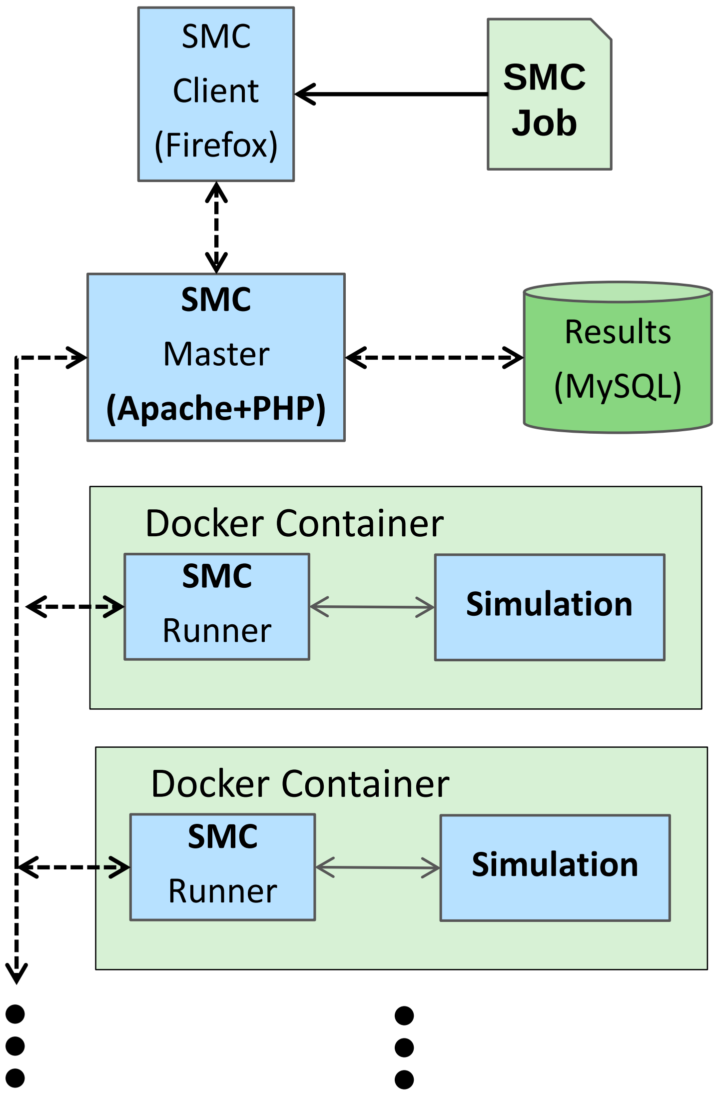
Importance sampling (focuses simulation effort on faults)


Batch Log and Analyze
log-gen
log-analyze
$Result_n$
SMC Runner
SMC Master
No
Update $Result$ and RE
RE acceptable?
Yes
$Result$
DART Distributed Statistical MC

Goal: Develop parallel infrastructure for SMC of DART systems

Accomplishments:

- Initial implementation with handwritten 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
- DEMETER: More robust infrastructure using "docker"



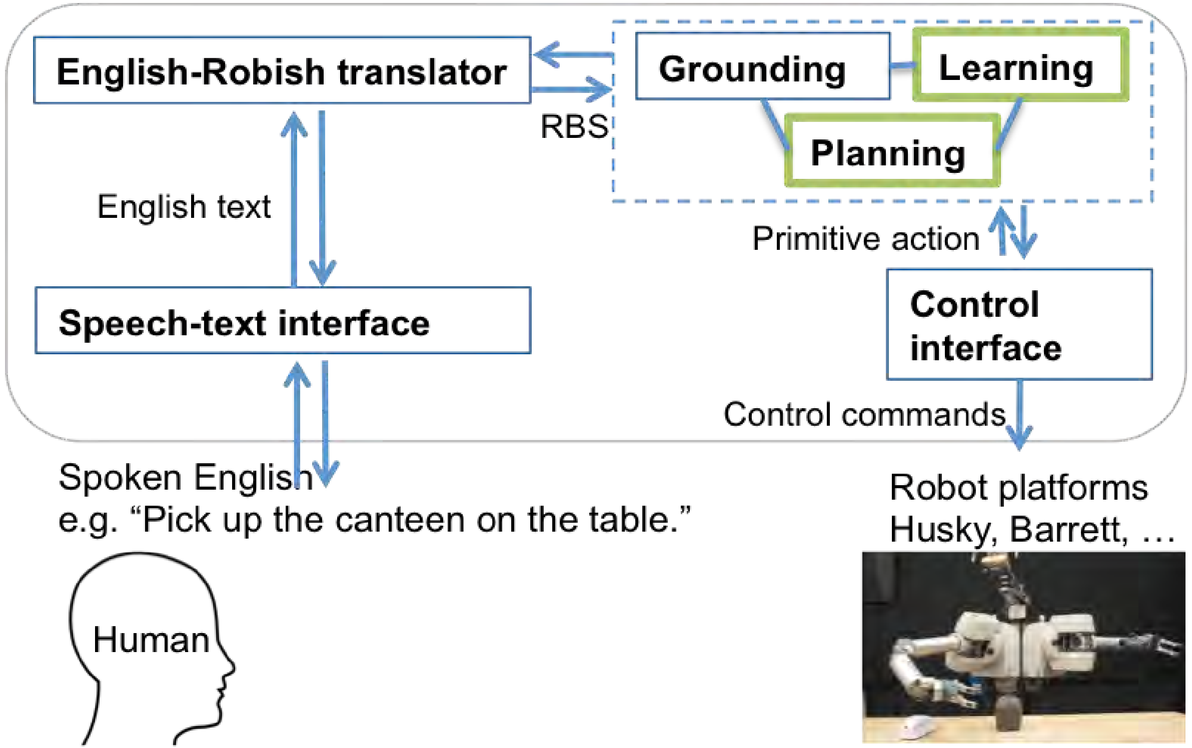
David Kyle, Jeffery P. Hansen, Sagar Chaki: Statistical Model Checking of Distributed Adaptive Real-Time Software. RV 2015: 269-274

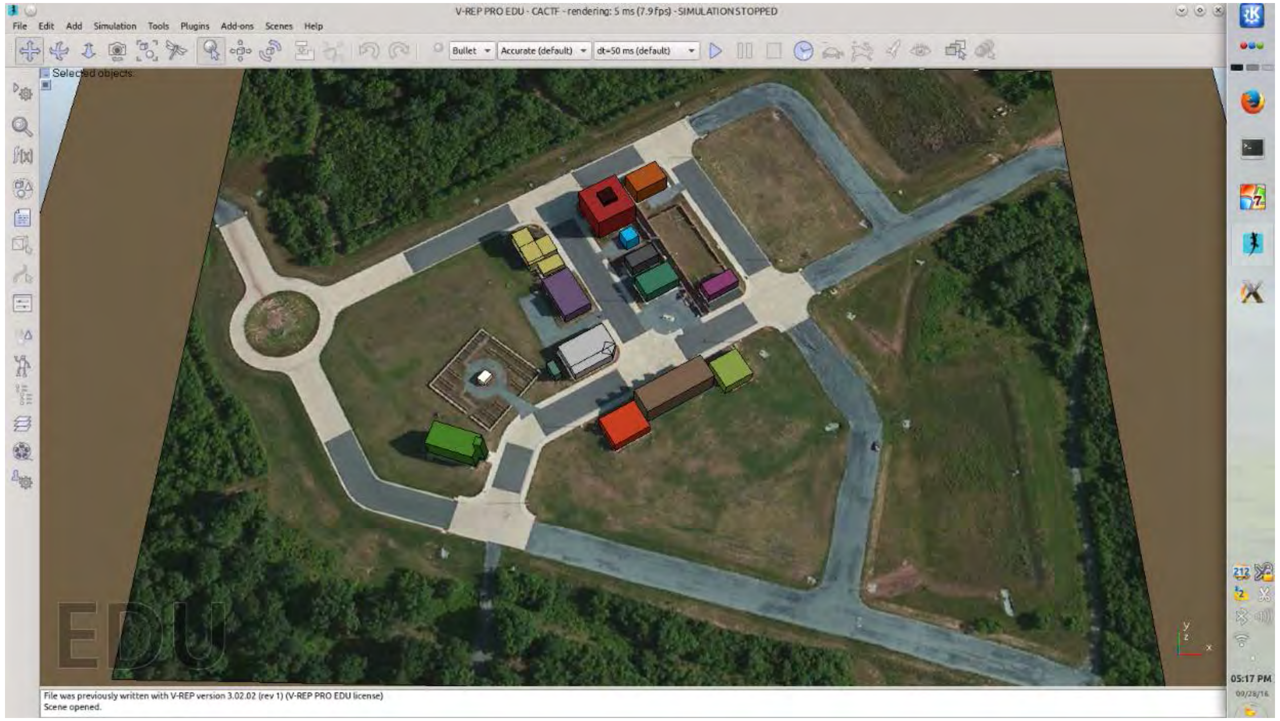
Jeffery P. Hansen, Sagar Chaki, Scott A. Hissam, James R. Edmondson, Gabriel A. Moreno, David Kyle: Input Attribution for Statistical Model Checking Using Logistic Regression. RV 2016: 185-200

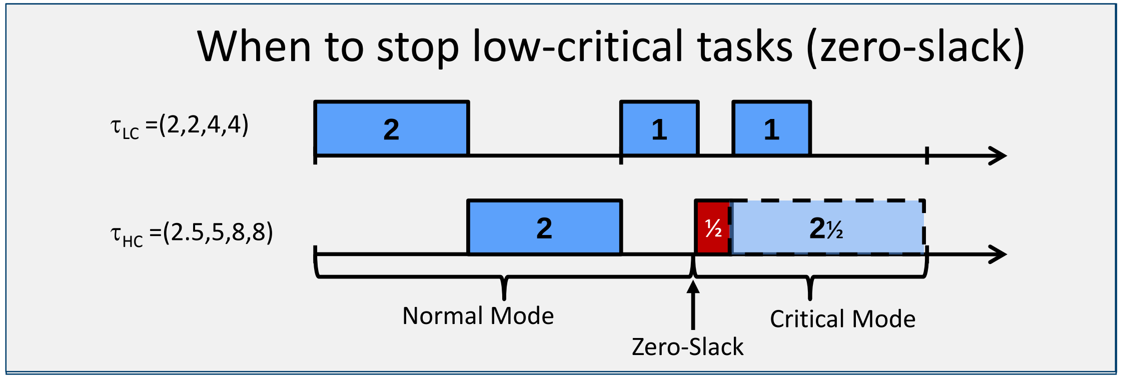
Evaluating quality of plans learned from verbal instructions by a robot using statistical model checking

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).

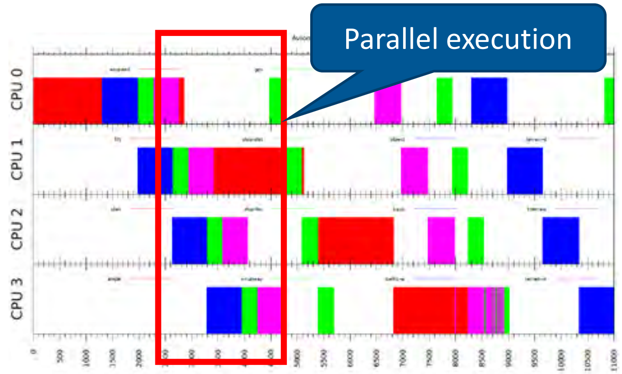
ZSRM: if no overload all task meet deadlines
if overload critical tasks meet deadlines
How: 1. when to stop low-critical tasks (Z)
2. stop them if not overload resume

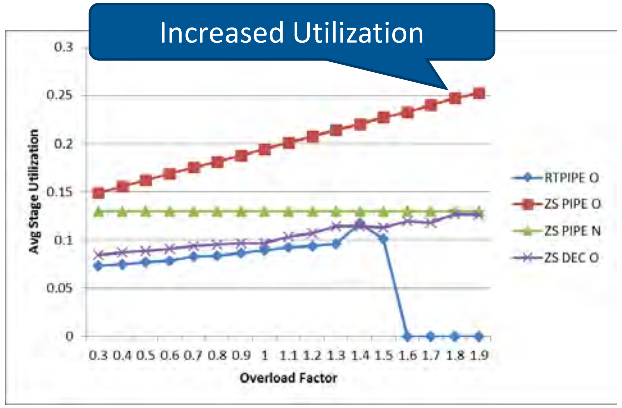
DART: requires distributed tasks

Accomplishments:

ZSRM Pipelines:

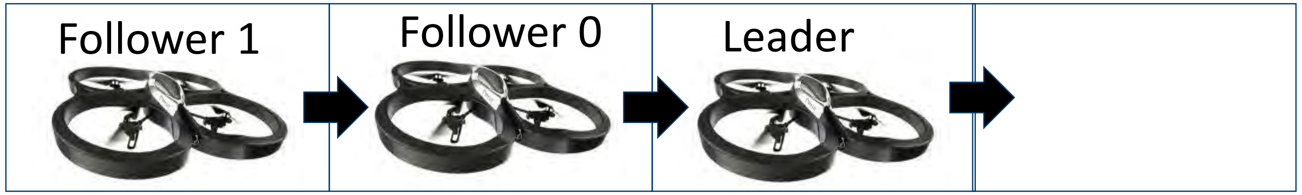
- Enforcement across processor
- Higher utilization

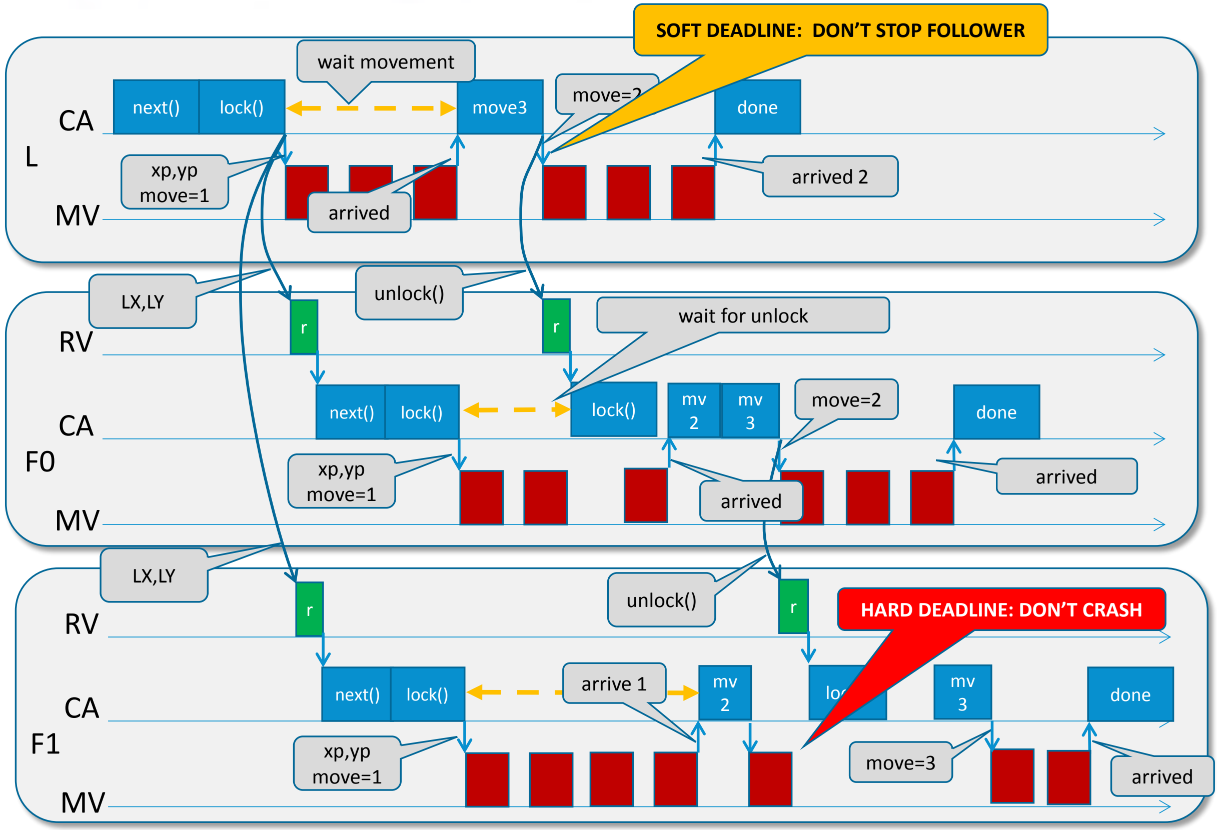


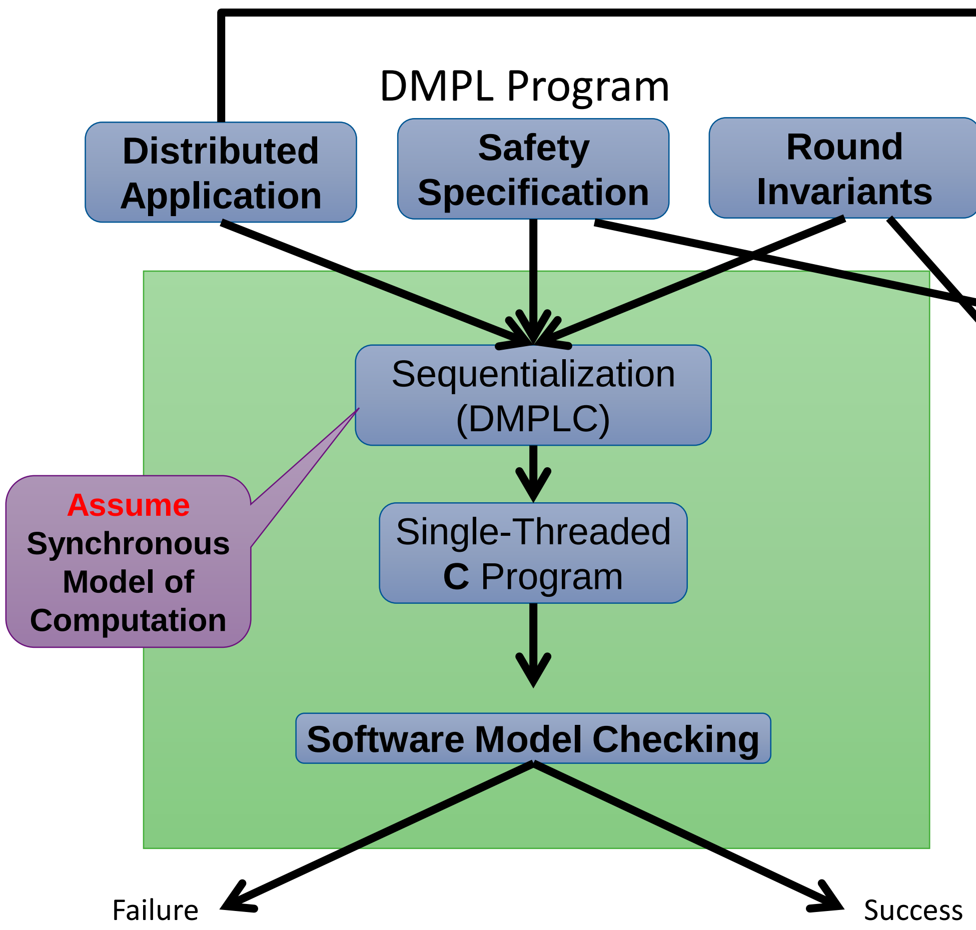


ZSRM Directed Acyclic Graph (DAG)

- Wait for movement
- Continuous movement:
  - Start moving before empty cell in front
  - Send early (half out) unlock to follower
  - Verify if no uncertainty meet deadline
- Guarantee no crashes
  - If drone in front delays hard stop
  - Guarantee no crash even if uncertainty








```
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. Stanley Bak, Sagar Chaki. International Conference on Embedded Software (EMSOFT), 2016

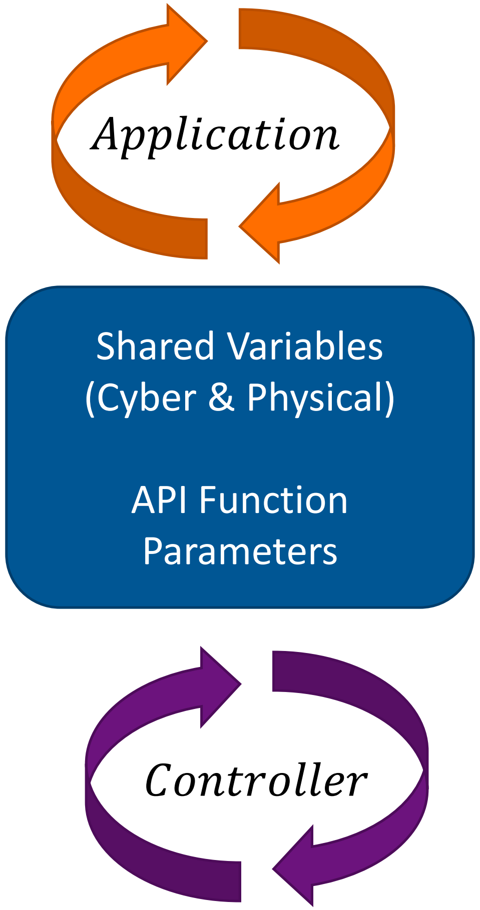
No existing tools to verify (source code + hybrid automata)

- 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 $A \parallel C \vDash \Phi$



Other FY16 Work

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

Future Work

Certifiable Distributed Runtime Assurance

QUESTIONS?

https://github.com/cps-sei/dart
http://cps-sei.github.io/dart
Please attend the poster session