Verifying Distributed Adaptive Real (DART) Systems

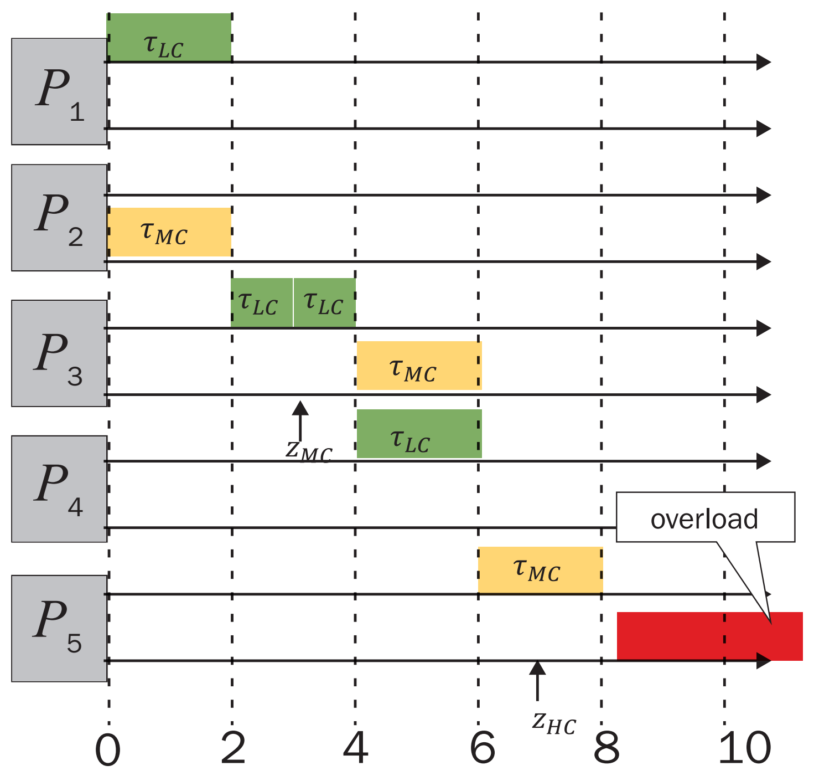
Research Review 2016

Pipelined ZSRM Scheduling

- Reduces pipeline to single-resource scheduling
- Avoids assuming worst alignment in all stages

But need to deal with transitive interferences due to zero-slack

Ongoing work: theory worked out, implementing scheduler in Linux



Functional Verification

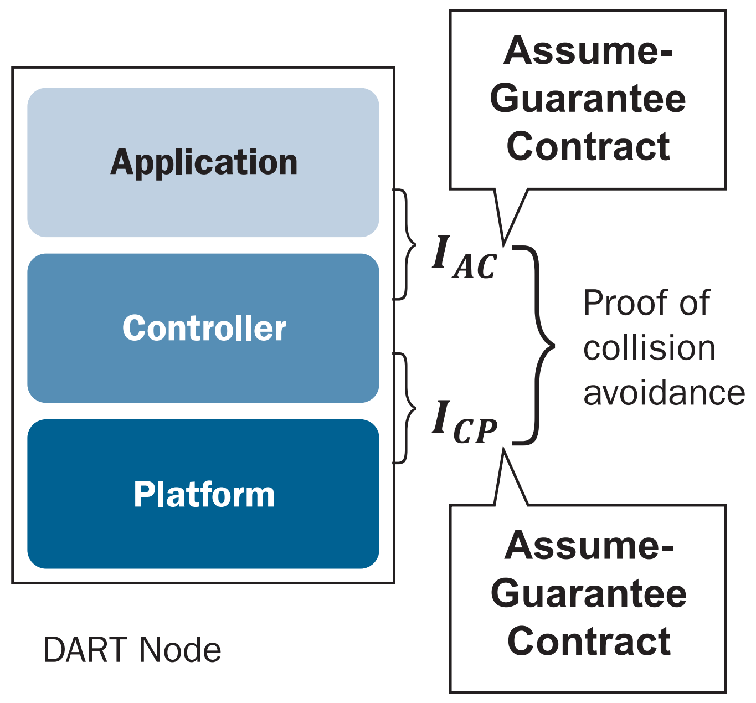
Prove application-controller contract for unbounded time

- Previously limited to bounded verification only

Prove controller-platform contract via hybrid reachability analysis

- Done by AFRL

Working on automation and asynchronous model of computation

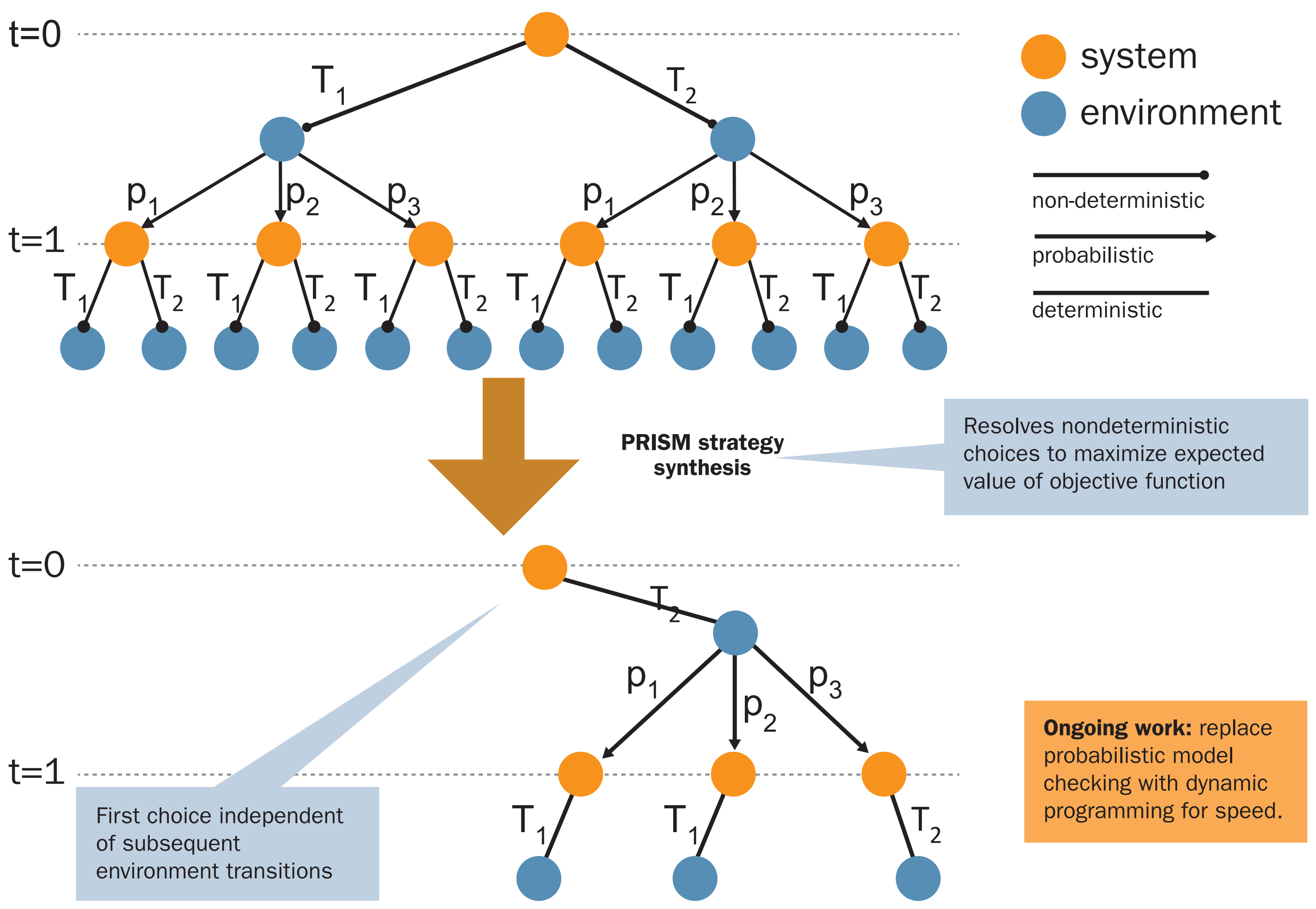


Application
Controller
Platform
I_{AC}
I_{CP}
Assume-Guarantee Contract
Proof of collision avoidance
Assume-Guarantee Contract

DART Node

End-to-End Functional Verification of CPS

Proactive Self-Adaptation Using Probabilistic Model Checking



Resolves nondeterministic choices to maximize expected value of objective function

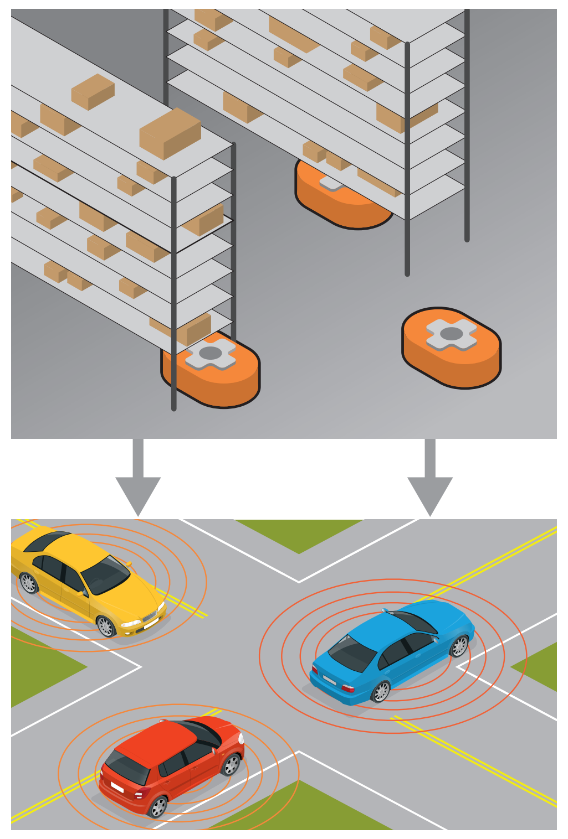
First choice independent of subsequent environment transitions

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

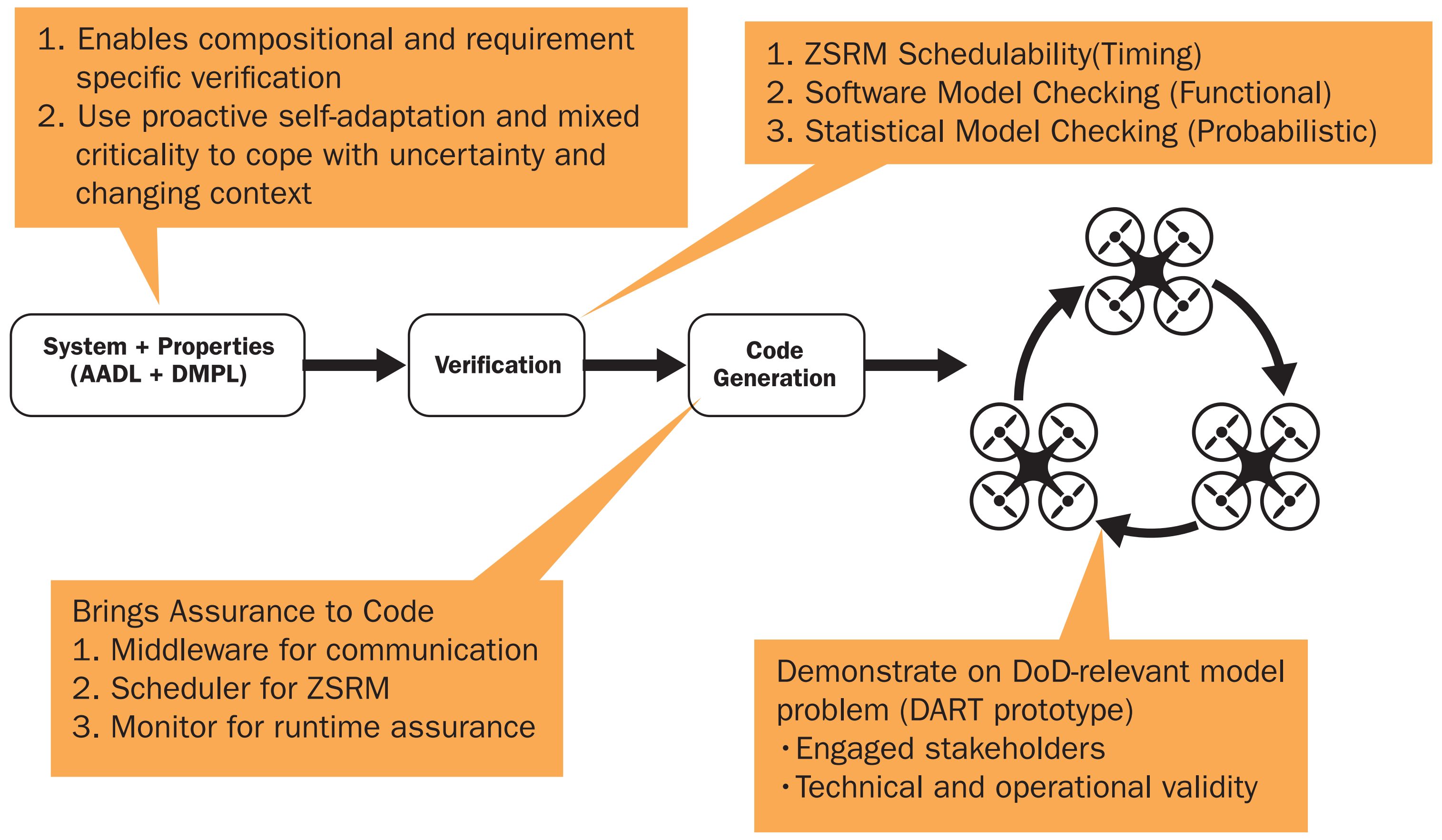
DART Vision

A sound engineering approach based on the judicious use of precise semantics, formal analysis and design constraints leads to assured behavior of (DART) systems while accounting for

- critical requirements
- probabilistic requirements
- uncertain environments
- necessary coordination
- assurance at source code level



DART Process



1. Enables compositional and requirement specific verification
2. 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)

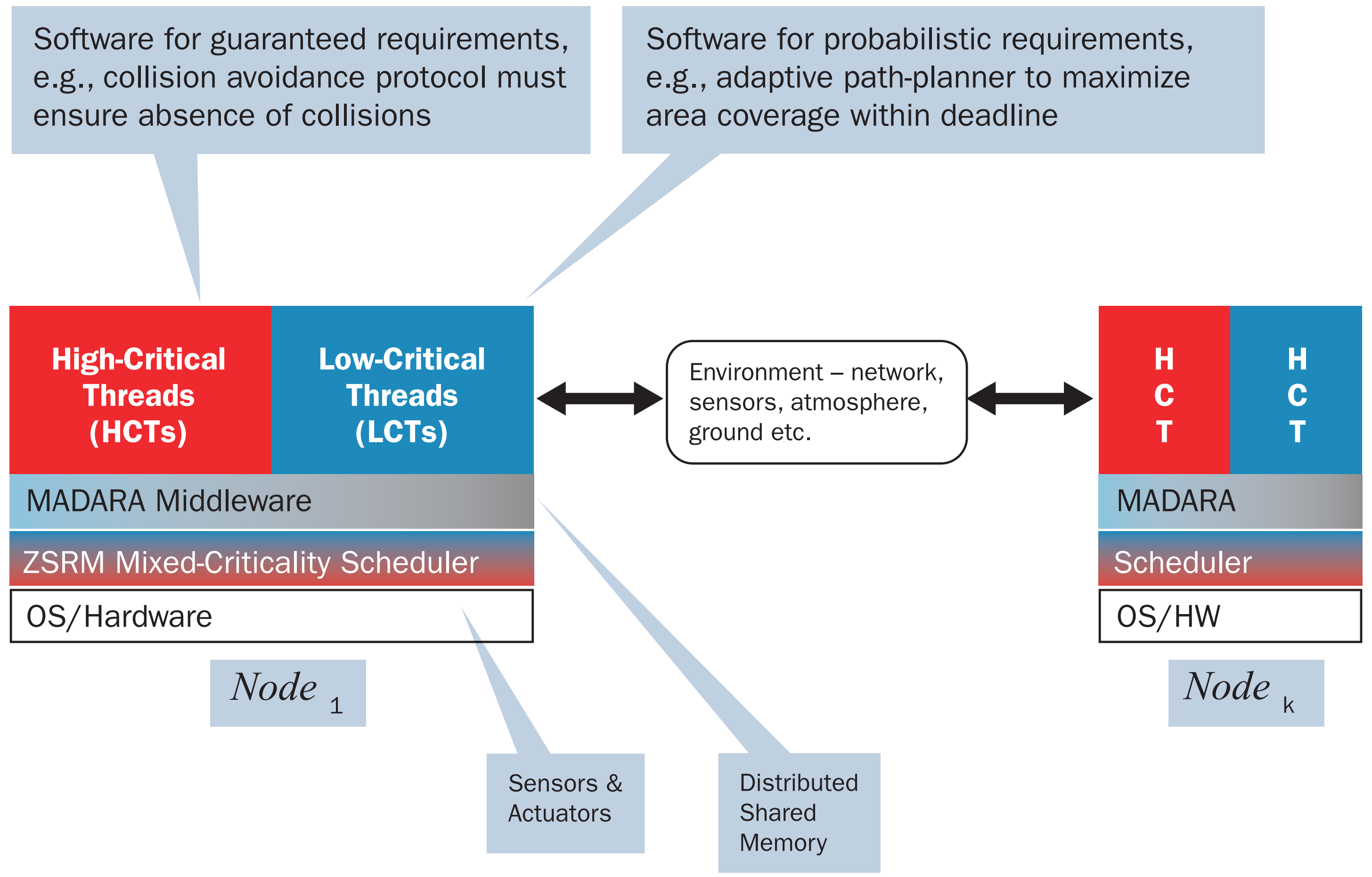
Brings Assurance to Code

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

DART Architecture



High-Critical Threads (HCTs)
Low-Critical Threads (LCTs)
MADARA Middleware
ZSRM Mixed-Criticality Scheduler
OS/Hardware
Node 1
Environment – network, sensors, atmosphere, ground etc.
H C T
H C T
MADARA
Scheduler
OS/HW
Node k

Software for guaranteed requirements, e.g., collision avoidance protocol must ensure absence of collisions

Software for probabilistic requirements, e.g., adaptive path-planner to maximize area coverage within deadline

Sensors & Actuators

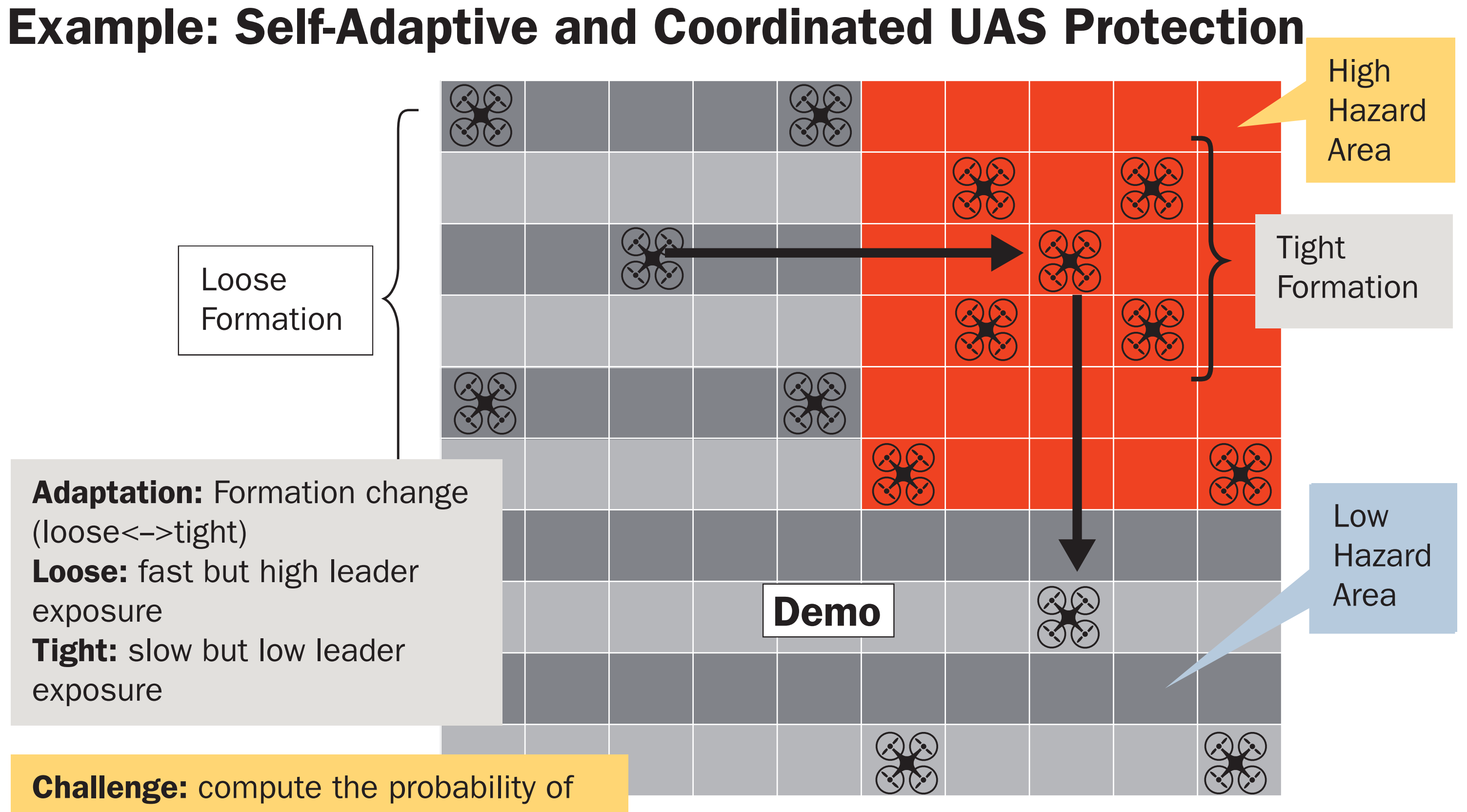
Distributed Shared Memory

DMPL: DART Modeling and Programming Language

- C-like language that can express distributed, real-time systems
- Semantics are precise
- Supports formal assertions usable for model checking and probabilistic model checking
- Physical and logical concurrency can be expressed in sufficient detail to perform timing analysis
- Can call external libraries
- Generates compilable C++
- Developed syntax, semantics, and compiler (dmplc)

DMPL supports the right level of abstraction. github.com/cps-sei/dart

Example: Self-Adaptive and Coordinated UAS Protection

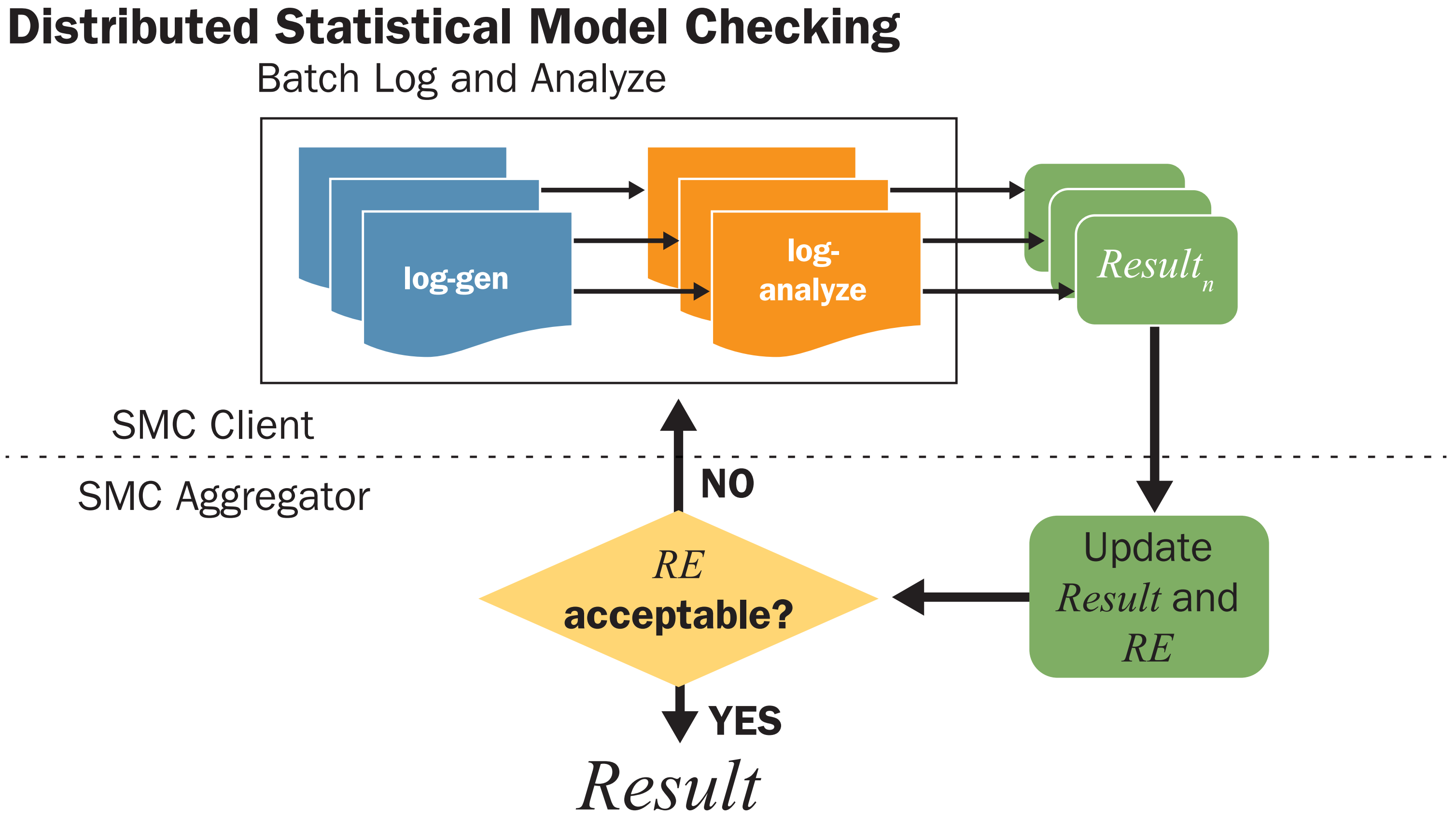


Adaptation: Formation change (loose<–>tight)
Loose: fast but high leader exposure
Tight: slow but low leader exposure

Challenge: compute the probability of mission success & compare between different adaptation strategies.
Solution: Statistical Model Checking

Statistical Model Checking of Distributed Adaptive Real-Time Software. David Kyle, Jeffery Hansen, Sagar Chaki.In Proc. of Runtime Verifcation 2015

Distributed Statistical Model Checking



Future Work: Importance Sampling to reduce number of simulations needed for "rare" events.

Each run of log-generator and log-analyzer occurs on a VM. Multiple VMs run in parallel on HPC platform. Clients added and removed on-the-fly.

Software Engineering Institute | Carnegie Mellon University

Distribution Statement A: Approved for Public Release;
Distribution is Unlimited

Contact: Sagar Chaki | chaki@sei.cmu.edu