
# Research in the Verification of Flight-Critical Systems

**Alwyn Goodloe**

**NASA Langley Research Center**


# Overview

- Background
- Verification and Certification
- Aircraft self-separation
- Runtime verification
- Formal-Methods for numerical software
- Experimental research

# NASA Langley

- LaRC created in 1917 as the first National Advisory Committee for Aeronautics (NACA) research facility
  - Located in Hampton, Virginia
  - Primarily R&D focus
- LaRC became a NASA lab in 1958
  - The Mercury program began at LaRC
- Research areas of focus: Aeronautics, Atmospheric Sciences, and Exploration
  - Aerospace engineers dominate the research culture
- I belong to Safety-Critical Avionics Systems Branch

# NASA R&D in Formal Methods

- NASA Langley Research Center (LARC) - Safety Critical Avionics Systems Branch
  - Fault-tolerance
  - Air Traffic management
  - Theorem proving
- NASA Ames Research Center (ARC) – Robust Software Engineering Group
  - Model Checking
  - Static analysis
- Jet Propulsion Laboratory (JPL) – Laboratory for Reliable Software
  - Mission oriented
  - Mars rover software

# VERIFICATION & CERTIFICATION

# So, You Want to Build a Commercial Aircraft?

- Form a startup and start hacking - just like silicon valley right?
  - Not so fast!
- Process starts off with a notification of intent to the FAA
  - A minuet begins between the company and the regulators
  - For a Part 25 aircraft they will tell you over 1500 safety criteria you must meet
    - Autos and medical devices are easy in comparison
    - DoD aircraft not subject to these regulations
- The Federal Aviation Administration (FAA) must certify the aircraft
  - Designated Engineering Representative (DER)
- The cyber-physical component is one of the largest risk factors
- Verification ≠ Certification
  - Safety is a systems level property

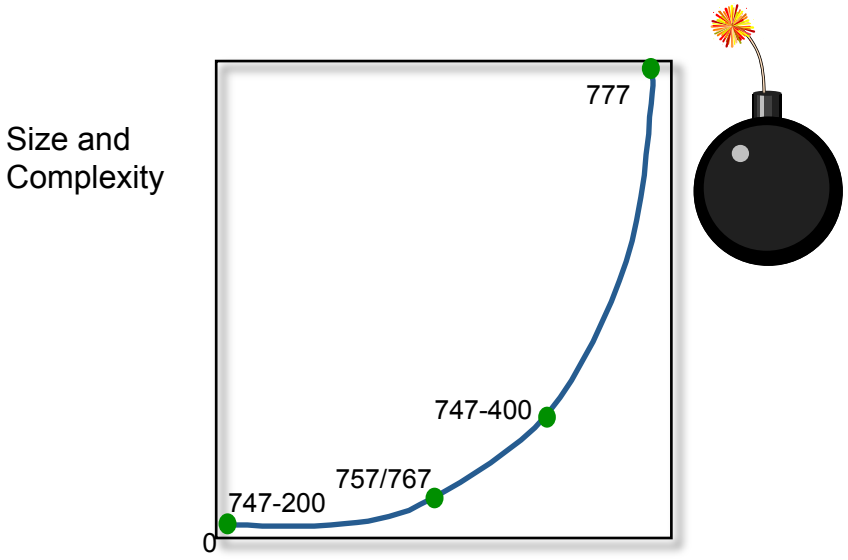
# Ultra-Reliability is Hard

We are very good at building complex software systems that work 95% of the time---but, we do not know how to build complex software systems that are ultra-reliable and safe.

What, then has saved us in the past?

- –minimal amount of software that is safety critical
- –simple designs
- –Enormously-expensive verification and certification processes
- –backups that are not software, e.g.
  - ° hardware interlocks
  - ° human intervention

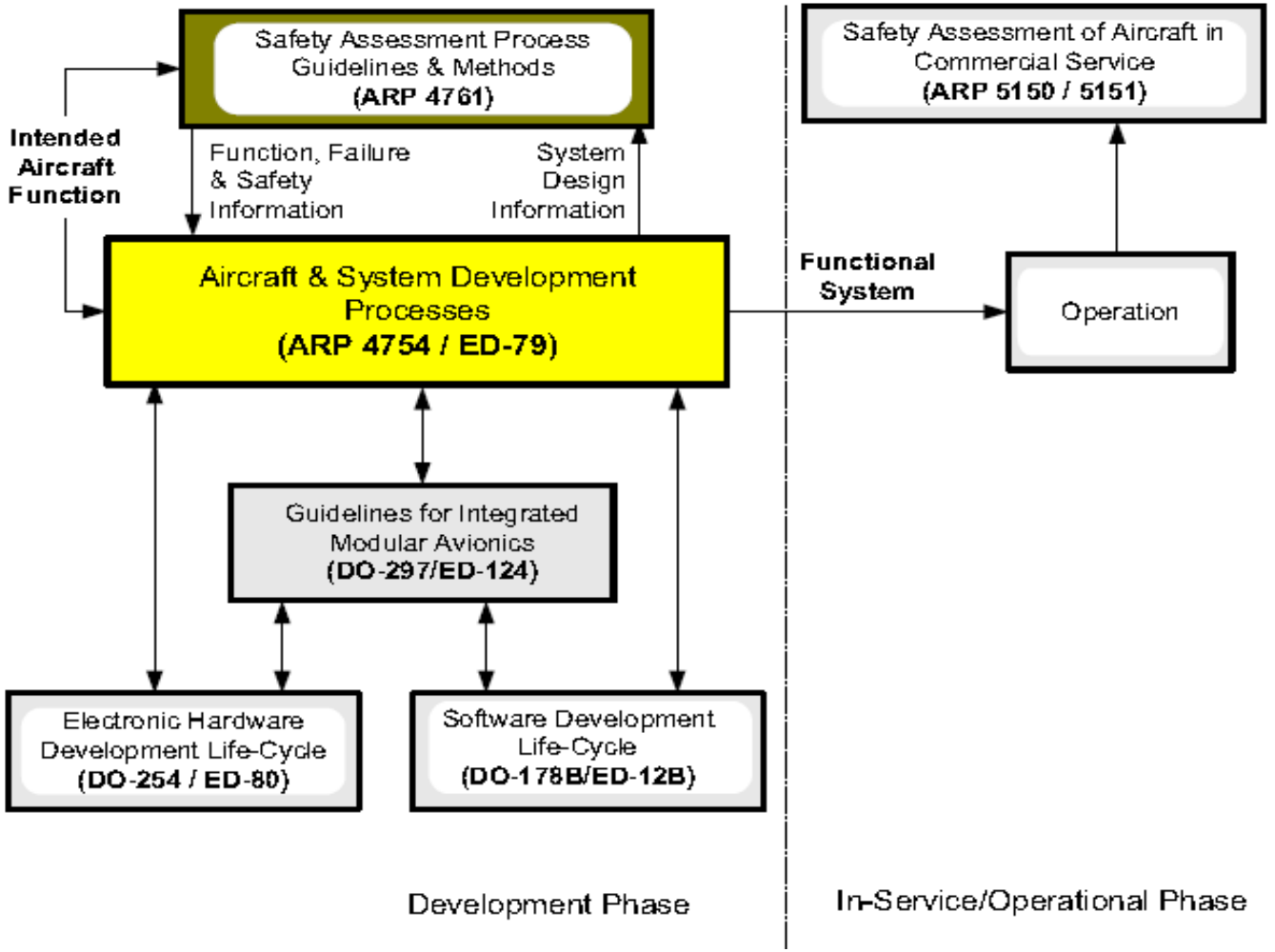
*All sectors of aerospace are increasingly relying on software to perform safety-critical functions*




# Software and Safety

- Critical avionics software is typically controlling some aspect of the aircraft
  - Control surfaces, fuel, …
- System must continue to operate safely in the presence of hardware failures
  - Byzantine faults are a reality in this environment
- Systems must be engineered to be be safe
  - Human is a critical component
- Burden to handle the added complexity to ensure safety usually falls on the software and the humans in the cockpit

# Guideline Documents



# Eliminating Common Mode Errors

- Independence – A concept to minimize the likelihood of common mode and cascade errors
- Diversity
  - HW, SW,
- Redundancy
  - Triple redundancy
  - Com/Mon
- Can mix techniques
  - Dissimilar com/mon


# Formal Methods I

- Aerospace industry has used formal methods to analyze architectural properties of designs
  - TTE protocols
- Existing certification regime very process/test oriented
- DO-333 is an RTCA standard allowing formal methods to replace unit test for evidence
  - Standard explicitly mentions: syntax, semantics, soundness
  - Still resolving tool qualification questions


# Formal Methods II

- Need much more work on languages and tools for specifying and analyzing architectures and designs of complex distributed hard real-time systems
  - Industry typically ignores the semantics until too late
- Avionics software much more restrictive than most commercial software
  - No recursion
  - No dynamic memory allocation allowed
  - Real-time scheduling issues always an issue
  - Lots of numerical code
- Most existing static analysis efforts not focused on this class of code
  - Very small market


# Certification I

- Certification authorities certify an aircraft as a whole
  - You build everything in conformance to standards and processes
  - DERs sign off every step of the way
  - No provision for certification by composition of certified modules
- Why?
- Hint: Certification is about safety

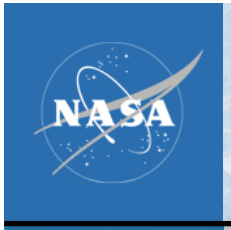

# Certification II

- Safety is not a compositional property
- Can assume/guarantee reasoning help?
  - Assumptions must include a fault model
  - How does system behave when assumptions fail
- Little work has been done in identifying the hurdles for modular certification
  - Rushby “Modular Certification”
- Challenge lies in the intersection of formal verification, fault tolerance, and safety-engineering

# SELF SEPARATION CONCEPT

# Aircraft Separation

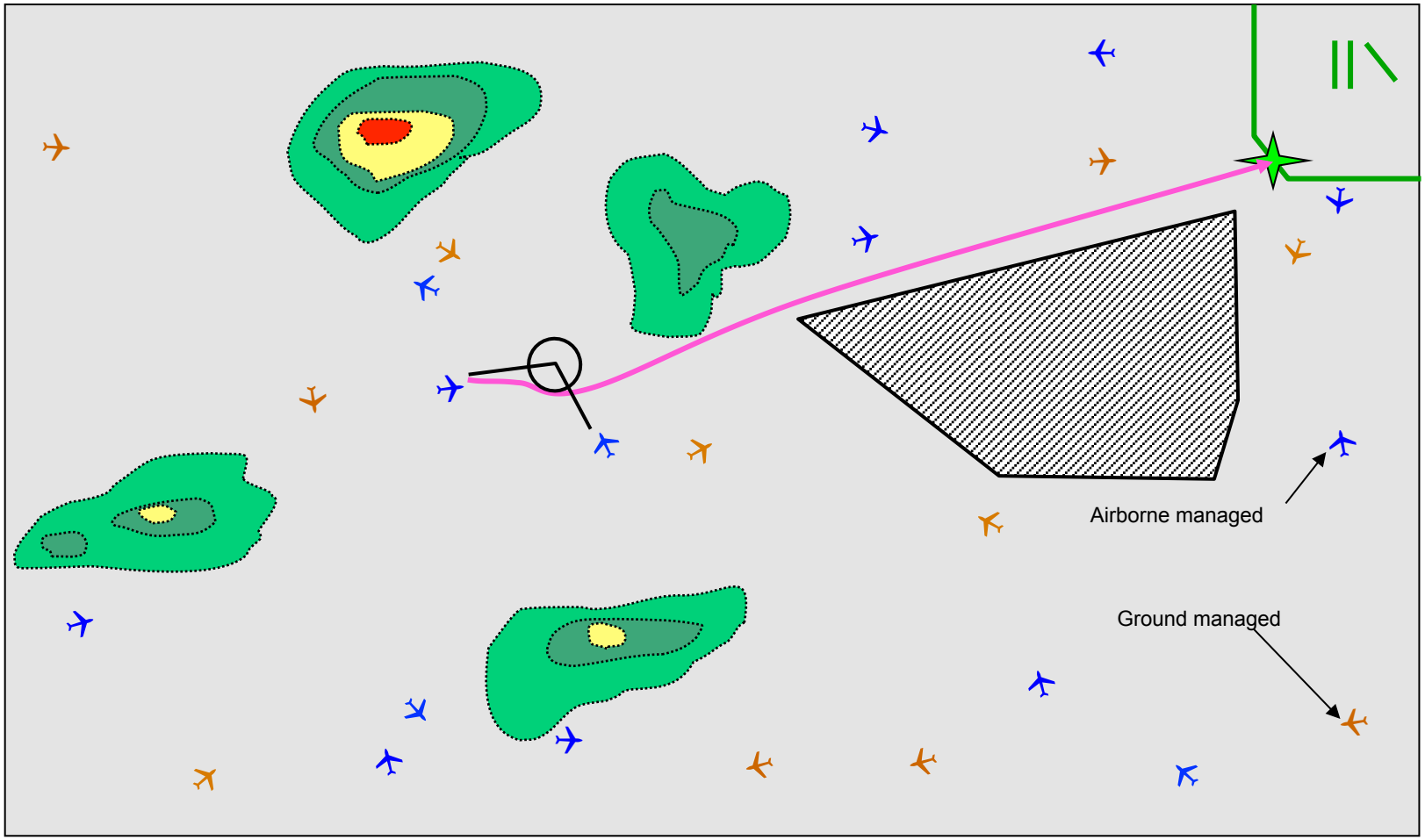
- NASA is investigating a variety of air traffic management concepts to look at increasing capacity, efficiency, flexibility, etc.
- Adding more controllers will not achieve gains in these parameters
- Enabled by Automatic Dependent Surveillance Broadcast (ADS-B)
- Idea is to place more information in the hands of the pilots and trust them to make the right decision




# Safe Self Separation

- More automation doesn't remove safety issues, but simply shifts the risk from people to automation
- Simulation helps find bugs
- Formal methods help show correctness
- Automated tools such as model checking and SMT solvers of little use due to lots of continuous math
- Interactive theorem proving is required

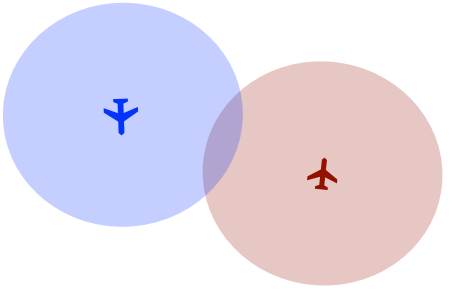
# Self Separation Concept



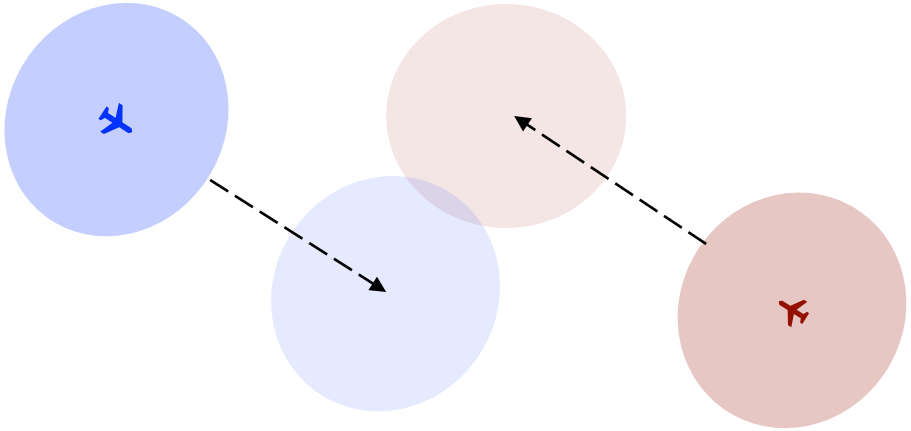
# Separation and Automation

- Collision
  - Scrape paint
  - Avoid through pilot, controller, and TCAS

- Loss of Separation
  - Separation standards are violated (5nmi, 1000ft)
  - Avoid through human and/or automation decisions



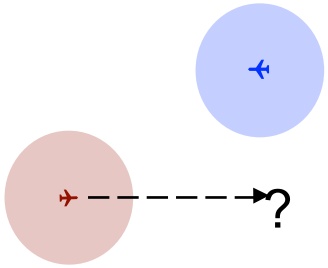
- Conflict
  - Predicted loss of separation



# Separation Algorithms

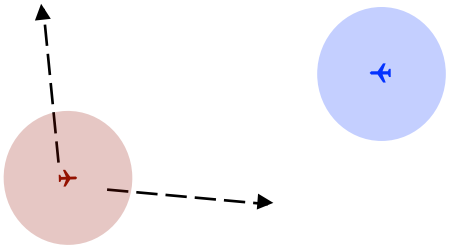
## Conflict Detection

- Detect future loss of separation



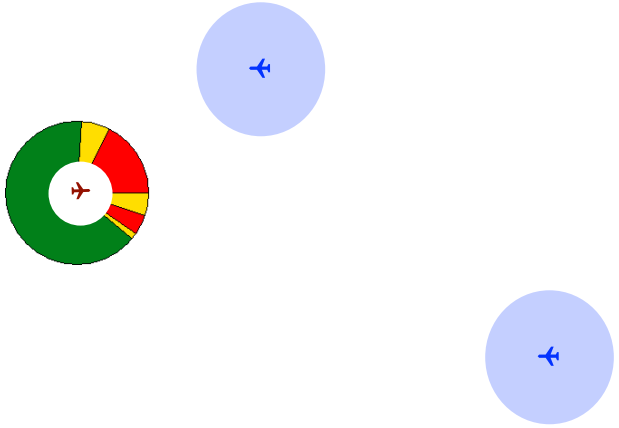

## Conflict Resolution

- Suggest maneuvers to resolve a conflict



## Conflict Prevention

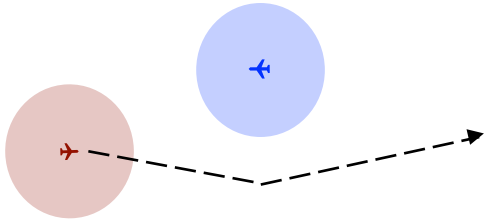
- Provides ranges of conflict-free and conflict prone maneuvers



# Recovery Algorithms

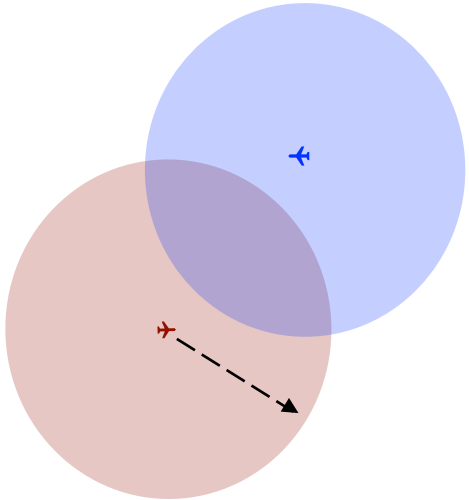
## Conflict Recovery

- Suggest maneuvers to regain desired path



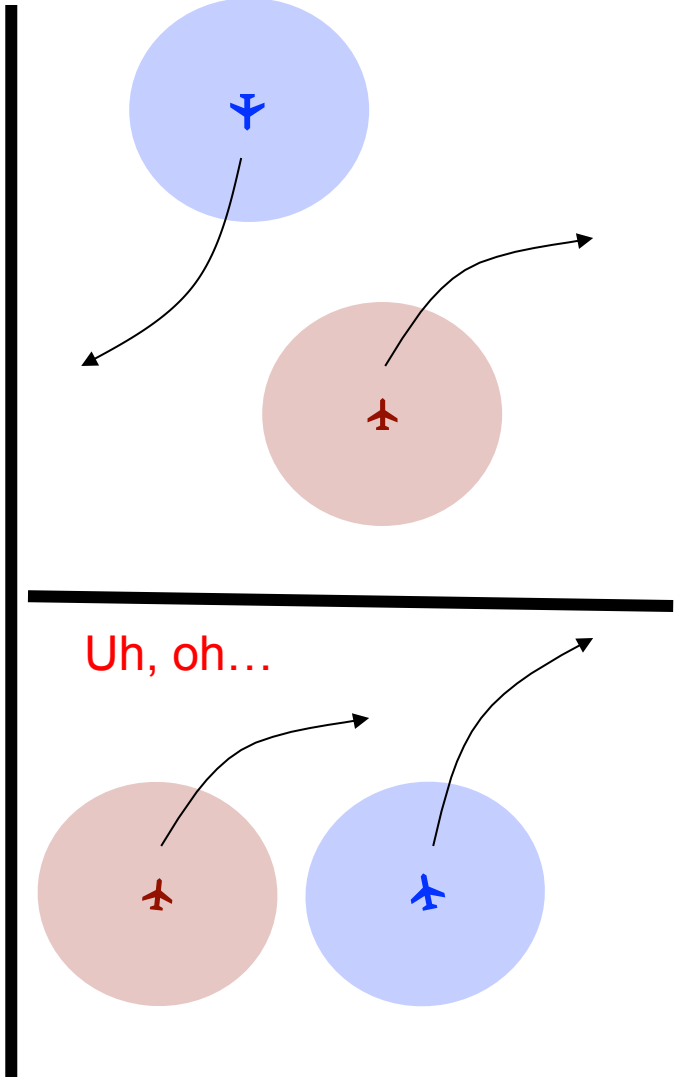
## Loss of Separation Recovery

- For a variety of reasons separation may be lost
- Suggest a maneuver to regain separation



# Conflict Resolution

- Each aircraft determines its own maneuvers through a combination of:
  - Go right/left, Speed up/slow down, Go up/down
- Properties
  - Independence: free of conflicts if one aircraft maneuvers
  - Coordination: free of conflicts if both aircraft maneuver
- Requirements
  - No specific comm between aircraft
  - No unfair rules: lower aircraft ID goes first, etc.



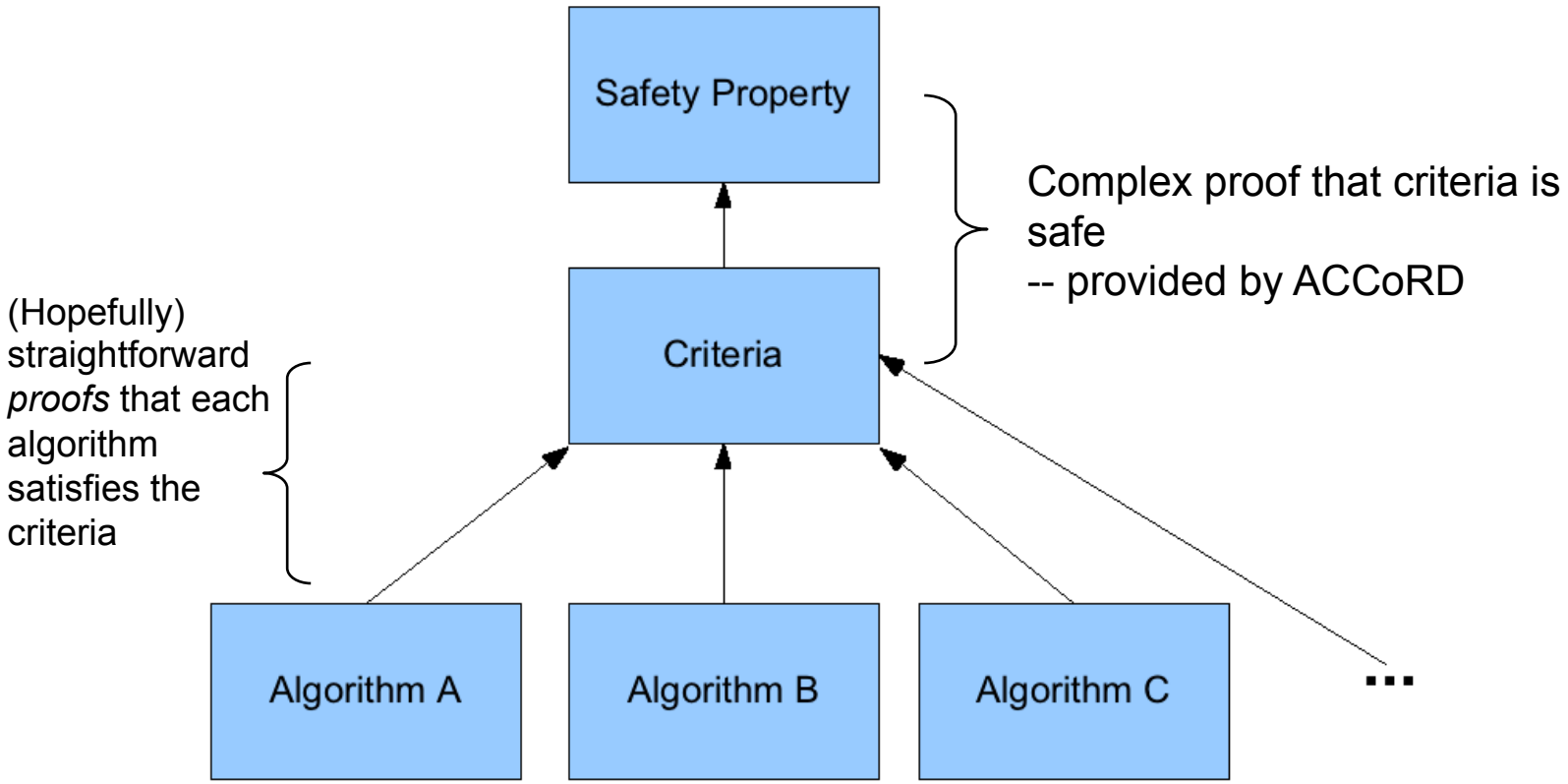
# Formal Statement of Properties

```
independent: THEOREM
    precondition_ind?(s(a), s(b), v(a), v(b)) AND
    (nva = cr3d_vertical_speed(a,b) OR
     nva = cr3d_ground_speed(a,b) OR
     nva = cr3d_heading(a,b)) AND
   IMPLIES
      NOT conflict?(s(a),s(b),nva-v(b))

coordinated: THEOREM
    precondition_coord?(s(a), s(b), v(a), v(b)) AND
    (nva = cr3d_vertical_speed(a,b) OR
     nva = cr3d_ground_speed(a,b) OR
     nva = cr3d_heading(a,b)) AND
    (nvb = cr3d_vertical_speed(b,a) OR
     nvb = cr3d_ground_speed(b,a) OR
     nvb = cr3d_heading(b,a))
    IMPLIES
      NOT conflict?(s(a),s(b),nva-nvb)
```

# ACCoRD Framework

- Airborne Coordinated Conflict Resolution and Detection (ACCoRD) – a verification framework for *classes* of separation algorithms

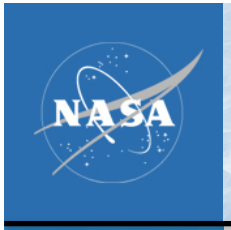


# Criteria is Very General

- The criteria was developed to aid the verification process
- Criteria allows combination of horizontal and vertical maneuvers
- But even more, if different algorithms satisfy the criteria, then they will be coordinated with each other
- Self separation does not rely on everyone running the same algorithm!

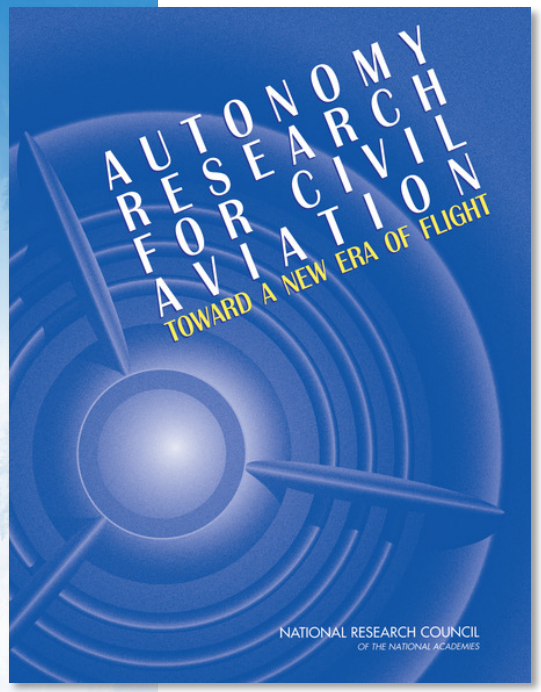
# Criteria

| | in Conflict | in Loss of Separation |
|---|---|---|
| horizontal | $(\mathbf{s} \bullet \mathbf{v'}) \geq \varepsilon \; R \, (\mathbf{s}^{\perp} \bullet \mathbf{v'})$ | $\mathbf{s} \bullet \mathbf{v'} > \mathbf{s} \bullet \mathbf{v}$ AND<br>$\mathbf{s} \bullet \mathbf{v'} \geq \|\|s\|\| \, (D - \|\|S\|\|)/T_h$ |
| vertical | $\Delta > 0$ AND $t > 0$ AND<br>$\delta = 1$ AND $s_z \, v_z \geq 0$<br>OR<br>$\| s_z + t \, v_z \| \geq H$ AND<br>$\delta \, (s_z + t \, v_z) \, v_z \leq 0$ | $v_z' \neq 0$ AND $s_z \, v_z' \geq 0$ AND<br>$s_z \, v_z \geq 0$ ==><br>if $v_z = 0$ then<br>break_symm(s)($v_z'$) > 0<br>else<br>sign($v_z$) $v_z' \geq 0$ |




# RUNTIME VERIFICATION

# Autonomy Research for Civil Aviation: Toward a New Era of Flight




National Research Council *Autonomy Research for Civil Aviation: Toward a New Era of Flight*. Washington, DC: The National Academies Press, 2014

The committee did not individually prioritize these barriers. However, there is one critical, crosscutting challenge that must be overcome to unleash the full potential of advanced increasingly autonomous (IA) systems in civil aviation. This challenge may be described in terms of the question "How can we assure that advanced IA systems—especially those systems that rely on adaptive/nondeterministic software—will enhance rather than diminish the safety and reliability of the NAS?" There are four particularly challenging barriers that stand in the way of meeting this key challenge:

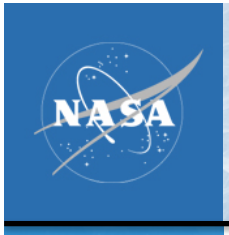
- *Certification process*
- *Decision making by adaptive/nondeterministic systems*
- *Trust in adaptive/nondeterministic IA systems*
- *Verification and validation*


# Runtime Verification Motivation

- Given the current state-of-the-art not all code can be formally verified
  - Code base too large
  - Complex logic
- Learning algorithms are a particular challenge
- How do we ensure that assumptions that underline safety are actually correct
- Runtime Verification part of the solution


# Runtime Verification

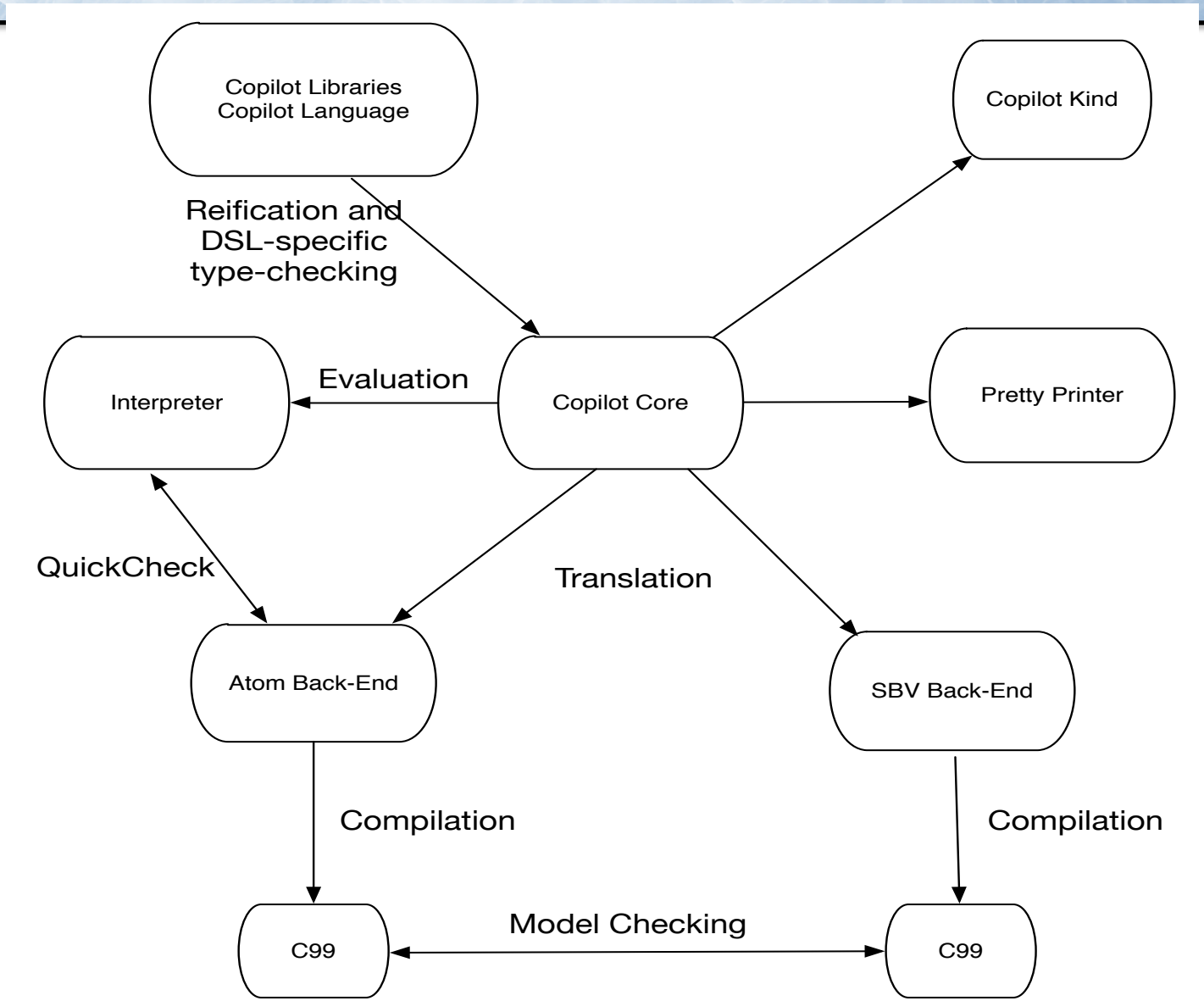
- System under observation (SUO)
- Correctness property φ
  - Past-time temporal logic
  - Regular languages
- Monitors observe SUO and detect violations of φ
- Accept all traces admitting φ




# Runtime Verification in Copilot

- Copilot is a runtime verification framework aimed at hard real-time systems
  - Co-developed by Lee Pike at Galois and myself
  - Many Haskell hackers: Robin Moriset, Nis Wegmann, Sebastian Niller, Jonathan Laurent
- Written as a Haskell EDSL
- Composed of approximately 3000 lines of Haskell
- Copilot type system is embedded in Haskell's
  - Hindley-Milner extended with type classes
- Translates into C99 through Atom or SBV

# Copilot Architecture




# Copilot Language Operators

- Stream language – constants and arithmetic operations are lifted to stream level
- (++) :: [a] -> Stream a -> Stream a
- xs ++ s - prepends list xs to stream s

- drop :: Int -> Stream a -> Stream a
- drop n s - skips the first n values in the stream

- Copilot specs must be causal – stream values cannot depend on future values


# Sample Copilot Specification

Haskell

```
fib :: [Word32]
fib = [0,1] ++ zipWith (+)  (drop 1 fib)
```

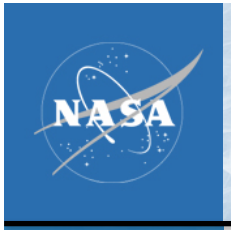
Copilot

```
fib :: Stream Word32
fib = [0,1] ++ (fib + drop 1 fib)
```

Special constructs for input (sampling) and output (triggers)


# Triggers

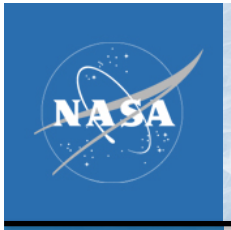
- Triggers provide a mechanism for Copilot streams to affect the outside world
- trigger:: String -> Steam Bool -> [TriggerArg] -> Spec
  - Name of external function
  - Guard determining when trigger is executed
  - List of arguments passed to the trigger



# Trigger Example I

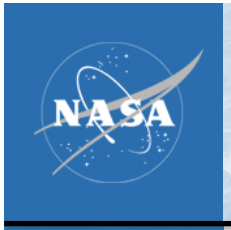
- If the temperature rises more than 2.3 degrees within 0.2 seconds, then the fuel injector should not be running
- Assuming that the global samplerate is 0.1 seconds

```
propTempRiseShutOff :: Spec
 propTempRiseShutOff =
      trigger "over_temp_rise”
        (overTempRise && running) []
```




# Trigger Example II

```
where
    max = 500 -- maximum engine temperature
    temps :: Stream Float
     temps = [max, max, max] ++ temp
     temp = extern "temp" Nothing
     overTempRise :: Stream Bool
     overTempRise = drop 2 temps > (2.3 + temps)
     running :: Stream Bool
     running = extern "running" Nothing
```




# Watching the Watchers

- Lightweight verification techniques applied to ensure the generated code is safe and correct
  - Model checking
  - Quick check
- SMT-based model checking applied to verify monitor properties
  - U of Iowa's Kind2 new IC3 based model checker employed

# Copilot Experiments

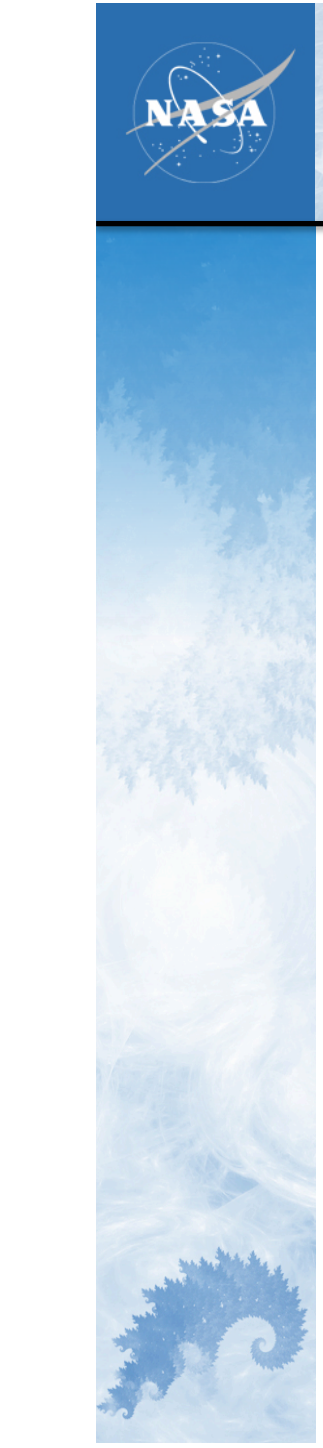
## Tasks

- Use monitors with custom HW to bolt on fault tolerance
- Flight tests to validate concept

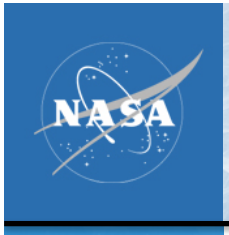
## Edge


# Future Research

- Flight Control software running on RTOS
  - Explore scheduling and architectures
  - Reachability analysis to derive monitors
- Simplex architectures that we can certify
- Monitor assumptions that underline safety cases
- Runtime verification applied to geofencing
- Using Copilot to verify autonomous software




# VERIFICATION OF NUMERICAL SOFTWARE




# From Models to Programs

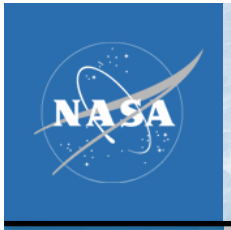
- Our models and proofs on aircraft separation are carried out on real numbers in PVS
- Computers running on aircraft do not execute exact real arithmetic!
- A theorem that depends on exact mathematical analysis may fail to hold in the inexact realm of floating point numbers
- Recent effort to focus on these issues
  - A different perspective than our scientific computing and numerical analysis experts


# Frama-C

- Frama-C is an OCAML-based platform for the static analysis of C programs developed at CEA
- Frama-C uses external decision procedures to prove ACSL annotations of C functions
- WP is a plug-in named for Dijkstra's Weakest Precondition calculus for deductive verification of programs
- Epsilon is an new project to add special functionality to WP to analyze numerical programs


# Separation of Concerns

- Verify the functional correctness of air traffic management algorithms  in PVS
  - Higer-Order Logic
  - Reasoning over ***R***
- Implement the algorithm in C
- Annotate the code using ACSL
  - First-Order logic
- Use Frama-C deductive verification tool to prove numerical error bounds on operations preserve safety argument




# Frama-C Specification + Code

```
/*@ logic real tauR(real ux, real uy, real vx, real vy, real b, real t) =
  @   dmin(dmax(b*sqvR(vx,vy), - dotR(ux,uy,     vx,vy)),   t*sqvR(vx, vy)) ; @*/
/*@ requires -185200. <= s_x <= 185200. && -185200. <= s_y <= 185200. &&
  @                -308. <= v_x <= 308. && -308. <= v_y <= 308. ;
  @ ensures  \abs(\result - tauR(s_x, s_y, v_x, v_y, Br, Tr)) <= 0x1.p-8;  @*/


double tau_vv(double s_x, double s_y, double v_x, double v_y)
       { return min(max((Br-epb)*sqv(v_x,v_y),
                - dot(s_x,s_y, v_x, v_y)), (Tr+ept)*sqv(v_x, v_y));}
```


# Epsilon

- Represent floating point values as intervals
  - Applied interval analysis
- Represent numbers as finite sum of powers of 2
- Keep track of relative error and absolute error at each operation
- We want to use static analysis techniques to estimate the error
- Apply techniques from abstract interpretation to the logical specification


# EXPERIMENTS

# Edge 540T UAS Test Bed

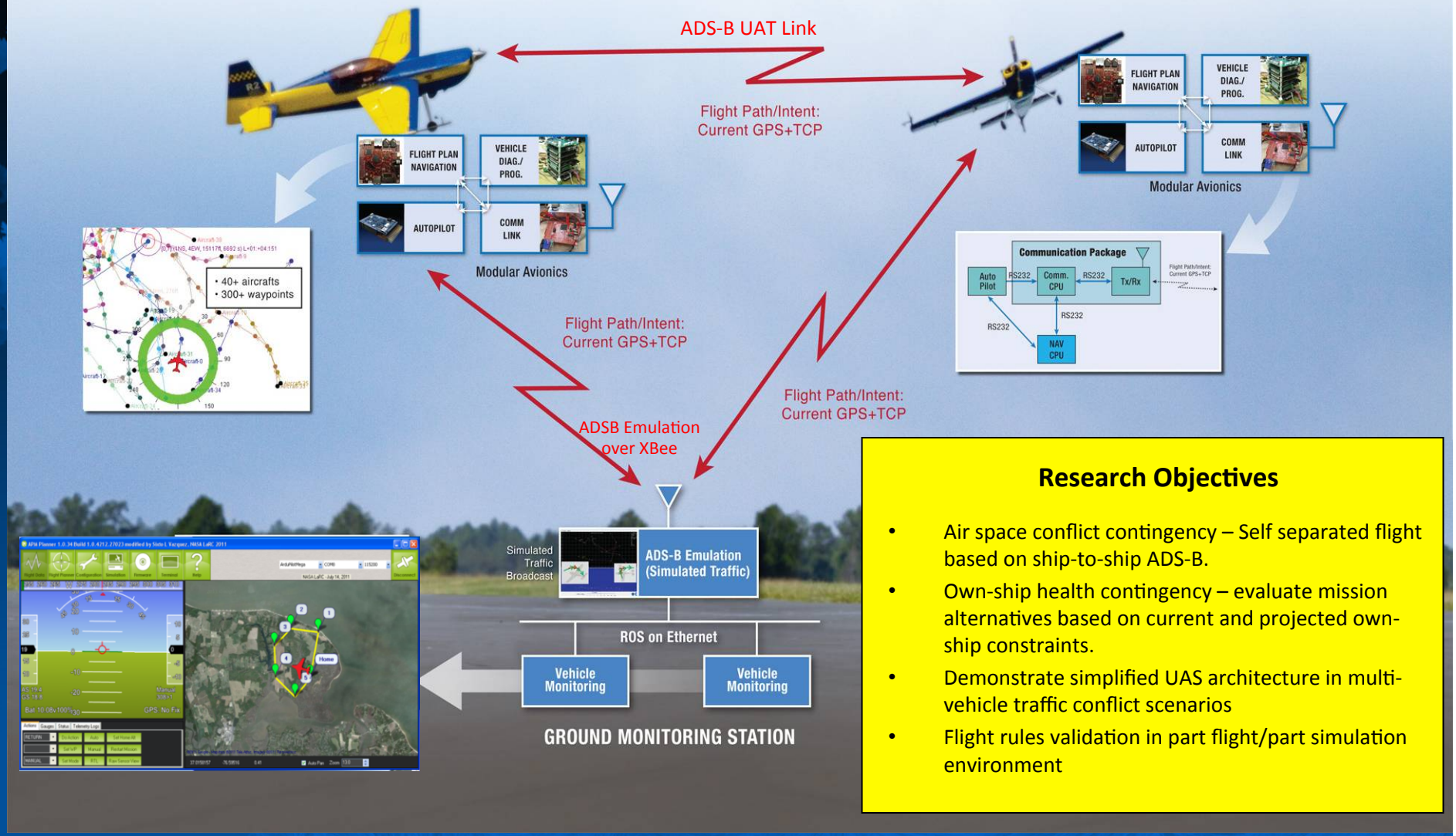
- Initial development funded by ARMD/IVHM now ARMD/SSAT
  - Battery health prognosis
  - Software health
  - Autonomous decision system algorithms
- Current Capability:
  - Autopilot – flight plan following
  - Battery prognostics algorithms
  - ADS-B IN/OUT
  - Air traffic conflict detection and resolution
- Available Data:
  - Airframe parameters
  - Power plant parameters
  - INS/GPS
  - ADSB IN traffic table
- Advantages:
  - Low cost vehicle suitable for high risk avionics algorithm and hardware testing.
  - Low cost air traffic scenario flight testing.
  - Minimal ground support hardware requirements.
  - Environment for testing decision algorithms for space applications.

**FAA registered LaRC 8' UAS: "N802RE"**

**Soon to be registered: "N803RE"**

**~ $20k per copy**

# Test Concept




## Research Objectives

- Air space conflict contingency – Self separated flight based on ship-to-ship ADS-B.
- Own-ship health contingency – evaluate mission alternatives based on current and projected own-ship constraints.
- Demonstrate simplified UAS architecture in multi-vehicle traffic conflict scenarios
- Flight rules validation in part flight/part simulation environment

# Questions?