CYBER ASSURED SYSTEMS ENGINEERING

SEL4 SUMMIT
11 OCTOBER 2022

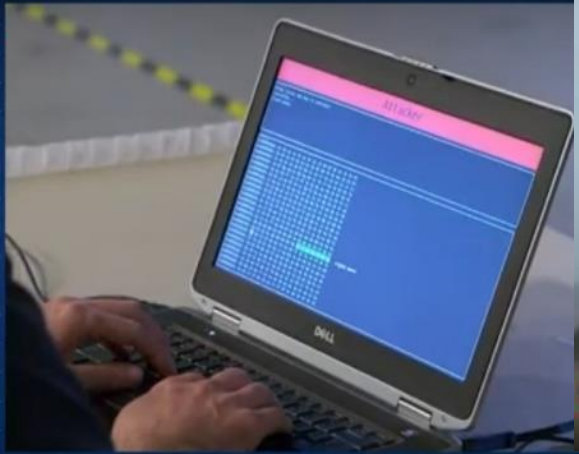
DARREN COFER

UNSW SYDNEY · THE UNIVERSITY OF KANSAS · Adventium LABS · KANSAS STATE UNIVERSITY

Collins Aerospace

DARPA HACMS

HIGH ASSURANCE CYBER MILITARY SYSTEMS



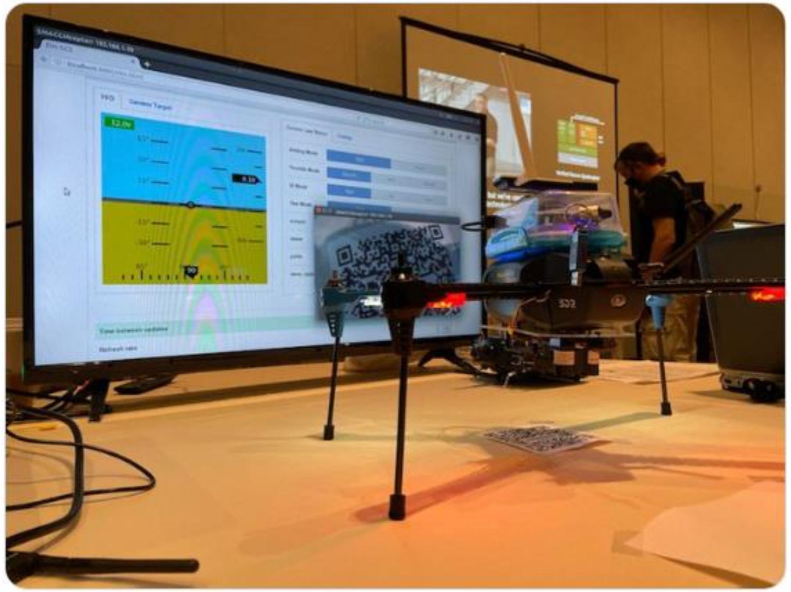
Loonwerks.com/projects/hacms
April 2017

9:59 / 25:07

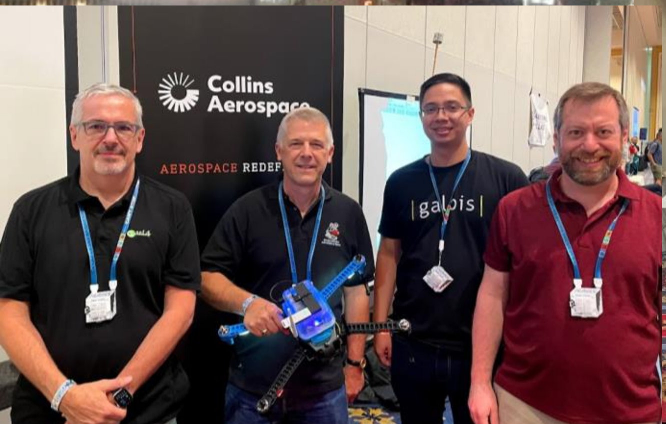
Collins Aerospace
seL4
galois
BOEING
University of Minnesota

DARPA
@DARPA

We brought a hackable quadcopter with defenses built on our HACMS program to @defcon #AerospaceVillage. As program manager @raymondrichards reports, many attempts to breakthrough were made but none were successful. Formal methods FTW!



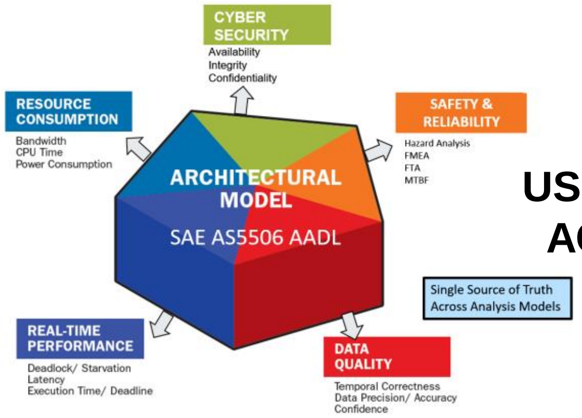
10:20 AM · Aug 9, 2021 · Hootsuite Inc.



CYBER ASSURED SYSTEMS ENGINEERING (CASE)

Develop model-based systems engineering tools and workflow to make the HACMS approach repeatable, scalable, more incremental

- **Design-in cyber-resiliency**
  - Automated architecture transforms for threat mitigation
  - High assurance components generated from specifications
  - Techniques to deal with legacy code (“cyber retrofit” using virtual machines)
- **Build what you model**
  - Build system directly from detailed, verified AADL model
  - Make the security guarantees of seL4 accessible to system developers
  - Ability to target different platforms to facilitate incremental debugging/development
- **Provide evidence**
  - Formal verification of functional and cyber-resiliency properties, information flow properties, component proofs
  - Code generation equivalence to model, seL4 build preserves properties
  - Integrate evidence as an assurance case demonstrating how/why requirements are satisfied

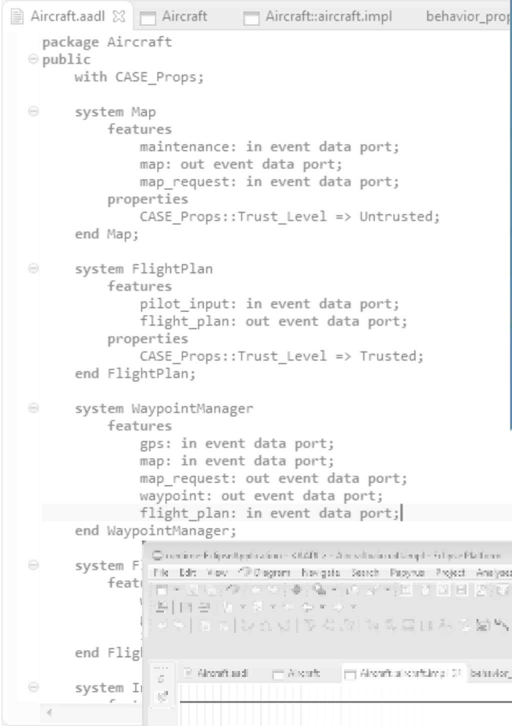


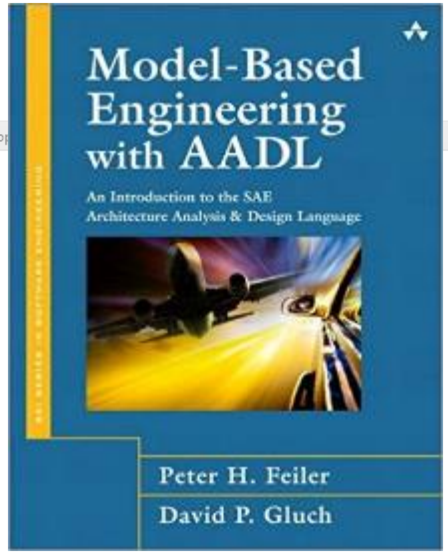
US Army ACVIP

BRIEFCASE INTEGRATED WORKFLOW

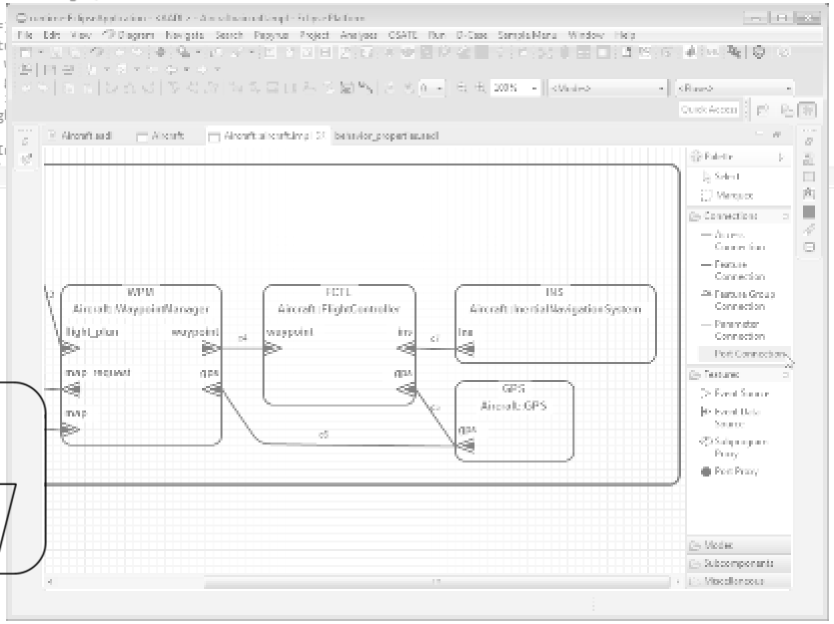
WITH INTEGRATED ASSURANCE

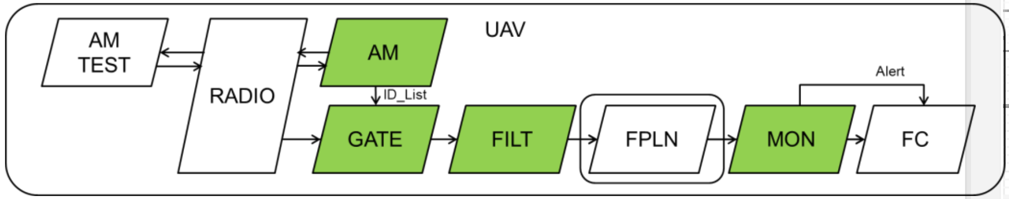
1. Capture/import **cyber-resiliency requirements** based on initial AADL model analysis (GearCASE and DCRYPPS)
2. **Transform system architecture** model to satisfy cyber-resiliency requirements
3. Generate new **high-assurance components** from formal specifications (SPLAT) or pre-verified libraries
4. Verify system design using **formal methods** (AGREE) and information flow analysis (Awas)
5. Checks **model conformance** to standards (Resolint)
6. Generate **software integration code** (HAMR) directly from verified architecture models, targeting multiple operating systems (including seL4)
7. Document evidence/compliance with **assurance case** (Resolute)





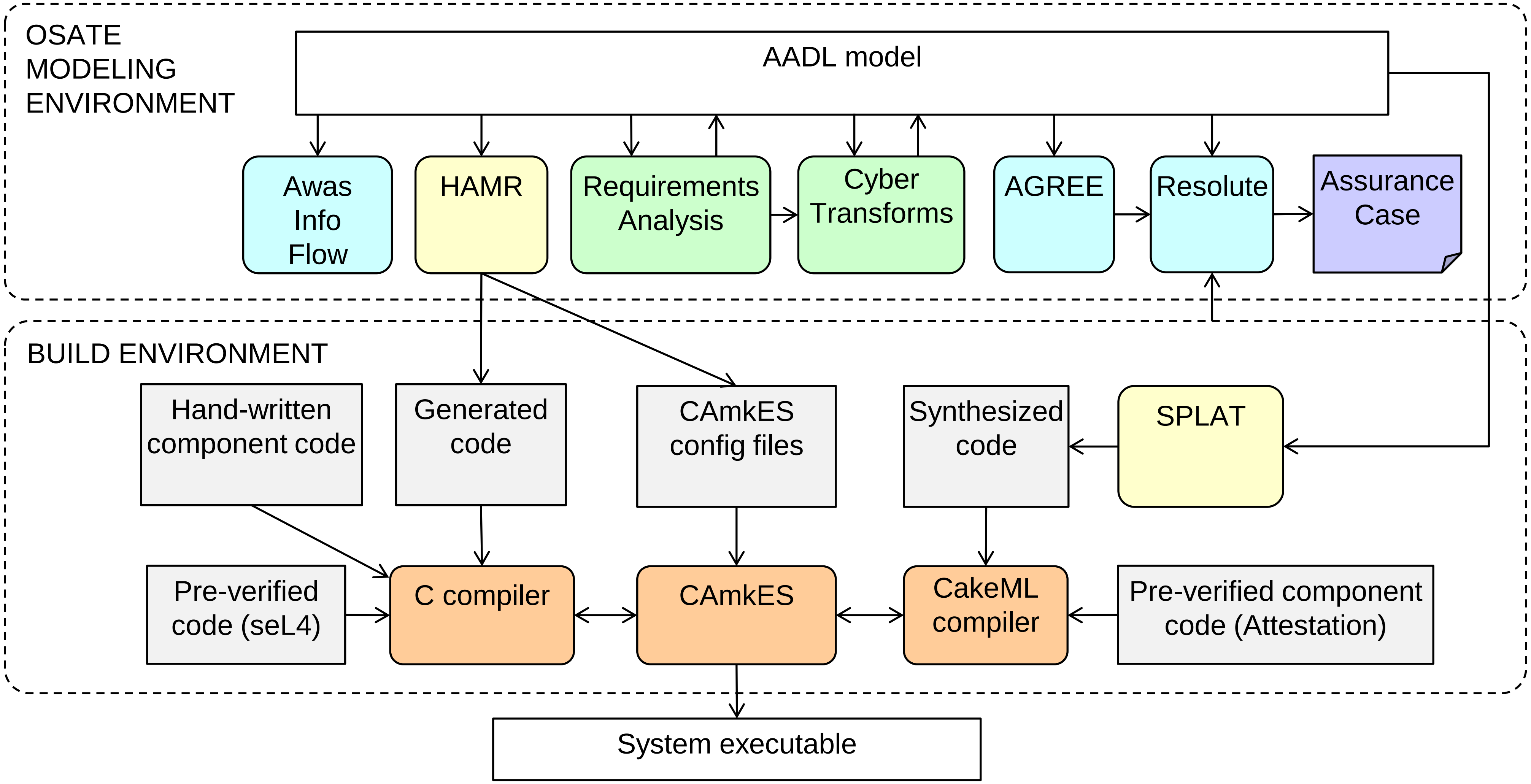
SAE AS5506 STANDARD





Collins Aerospace

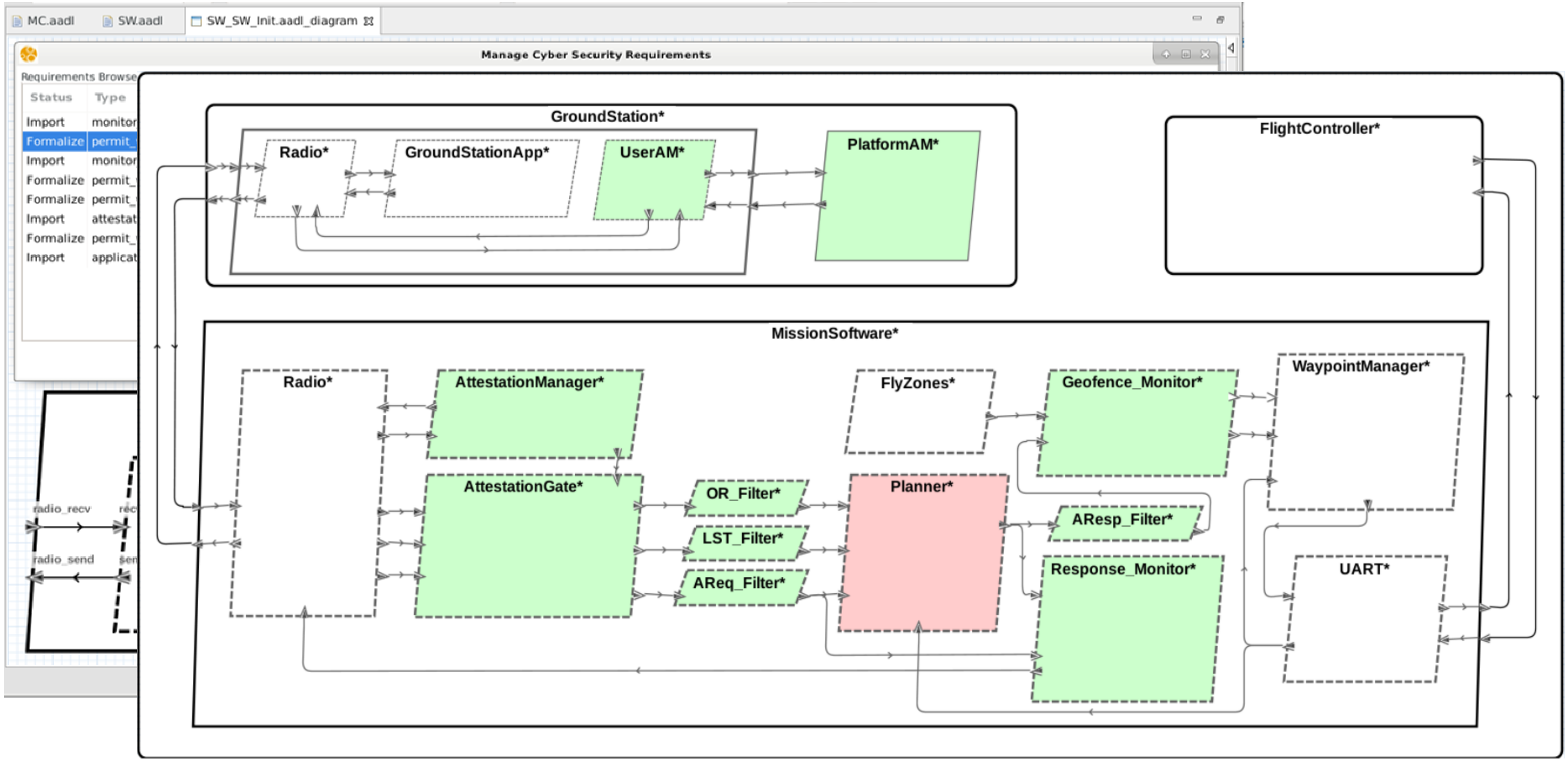
BRIEFCASE TOOL ARCHITECTURE



Collins Aerospace

SYSTEM ARCHITECTURE TRANSFORMATION

EXPERIMENTAL PLATFORM : SMALL UAV



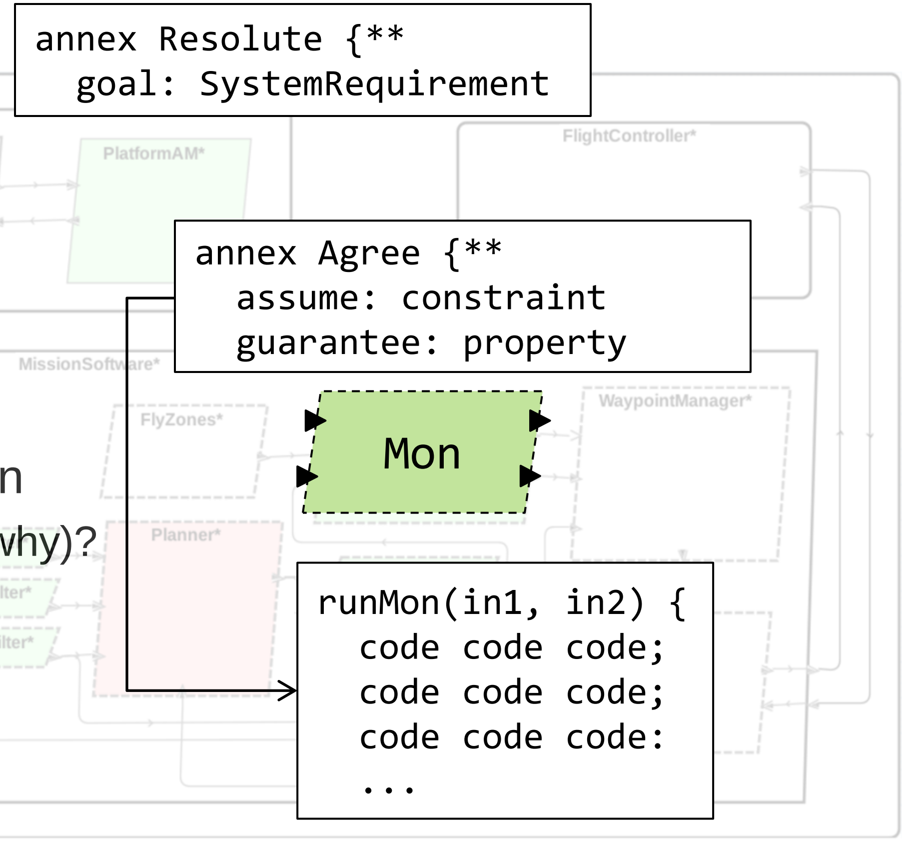
Collins Aerospace

SYSTEM ARCHITECTURE TRANSFORMS

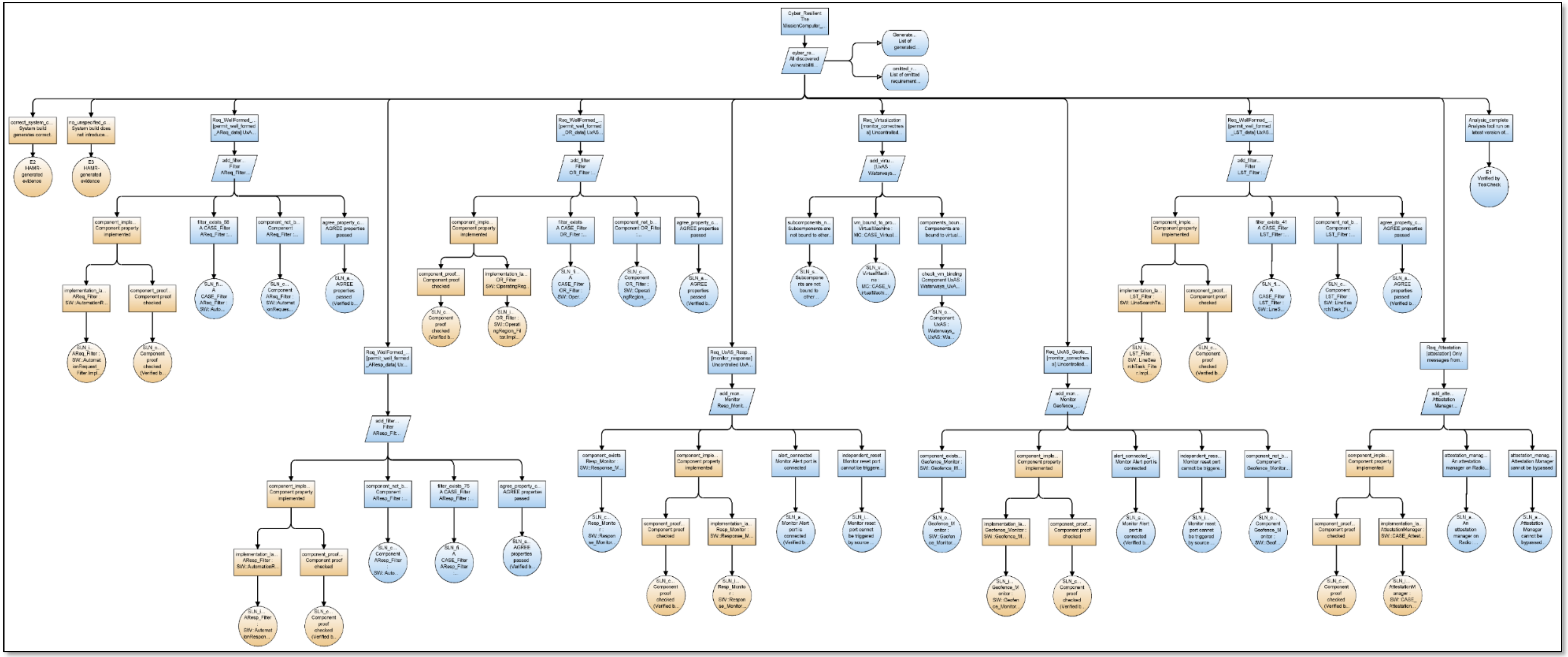
ASSURANCE CASE BUILT AUTOMATICALLY

Each model transformation accomplishes the following:

- Address some system requirement
  - Goal to be met in assurance case
- Modify model, specifying any new connections and interfaces
- Formal contract describing new component behavior
- Implementation of that behavior
  - Synthesized from specification or pre-verified library
- Automatically update assurance case to document this information
  - What has been done to the model to address this requirement (and why)?
  - Linked to evidence supporting the requirement (goal)
  - Why is the requirement also true for the implementation?
- Example transforms
  - Filter, Monitor, Gate (controlled by monitor)
  - Attestation (remote computer trustworthy?)
  - Virtualization, seL4 build prep



RESOLUTE ASSURANCE CASE

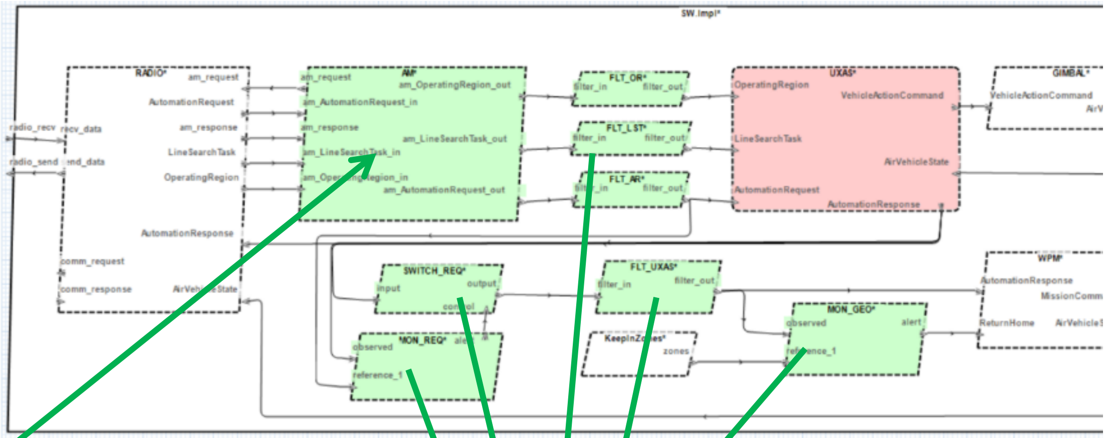


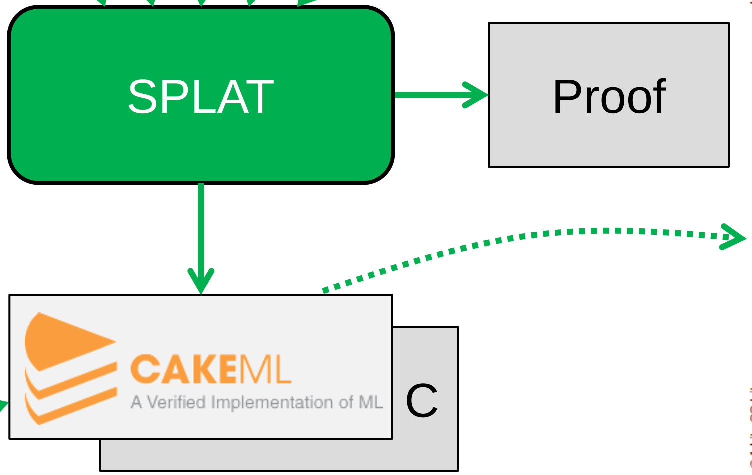
GENERATE HIGH ASSURANCE COMPONENTS

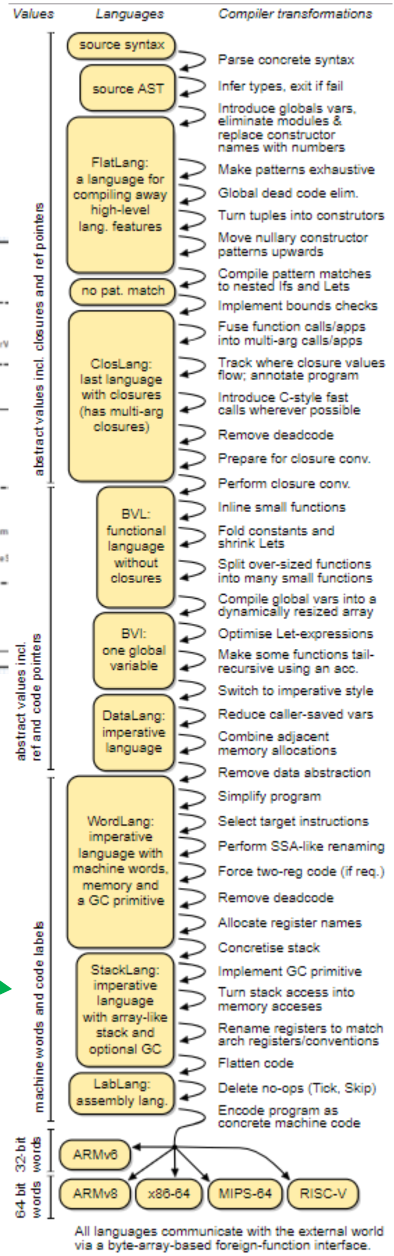
- Some of the cyber transforms insert new high-assurance components into the model
- The behavior of the component (its contract) is specified in AGREE
- **SPLAT generates component implementations from their specifications**
- SPLAT also generates a proof showing that the component implements its specification

- Other components (e.g., **Attestation Manager**) are pre-built pre-verified libraries
- Their implementations are essentially library functions that are added to the build, possibly with some configuration data from the model

- Code can be generated in the CakeML language which has a verified compiler





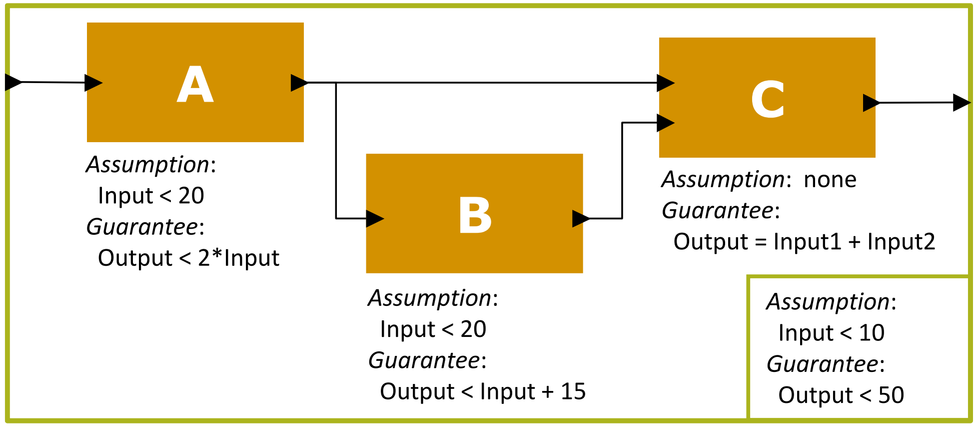


ANALYZE SYSTEM BEHAVIOR

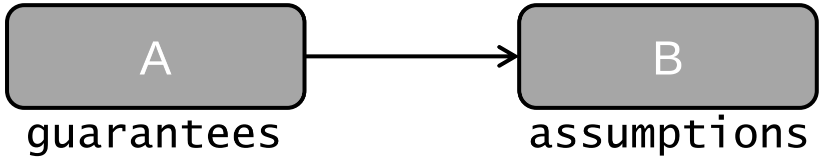
ASSUME GUARANTEE REASONING ENVIRONMENT (AGREE)

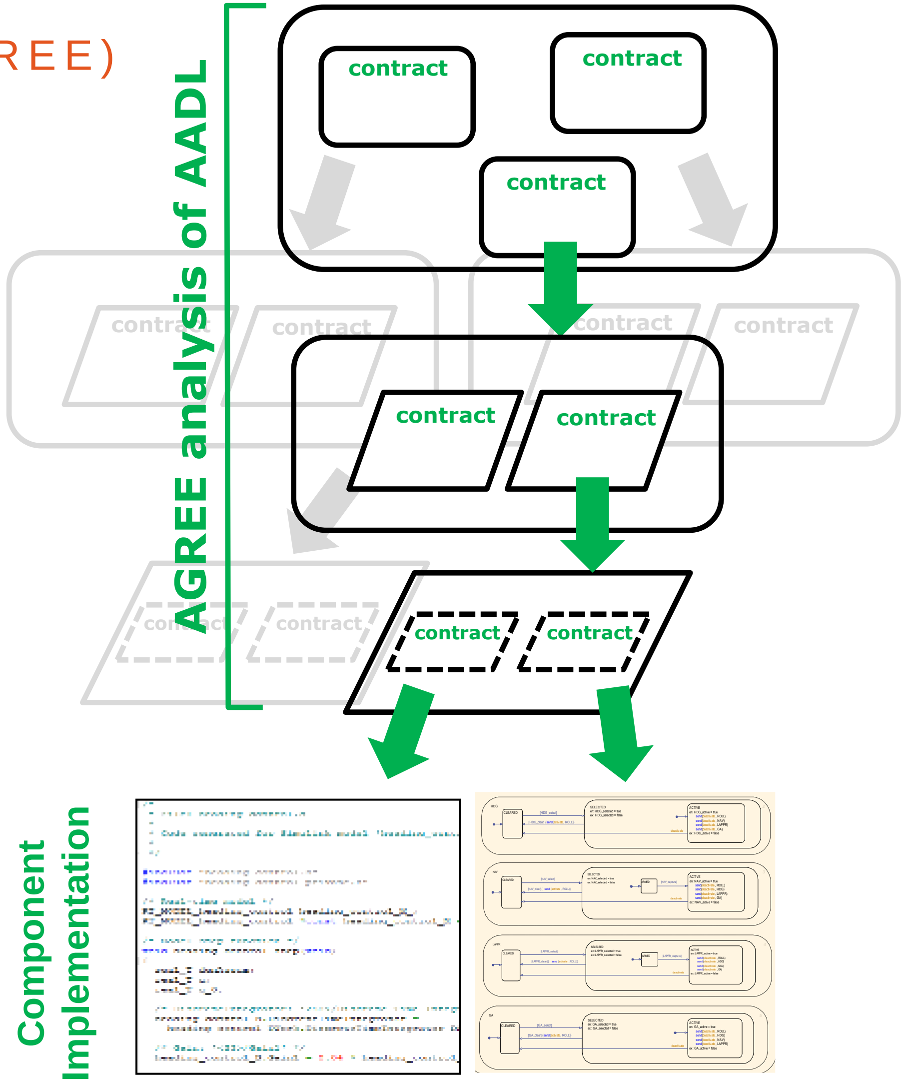
- Contract-based *compositional reasoning* provides **scalability**
- Each component has a *contract* consisting of assumptions and guarantees
- The contract of a component abstracts the behavior of its implementation
- Contracts at each layer must be satisfied by contracts of its subcomponents
- Leaf component contracts must be satisfied by implementation

Composition



Modularity



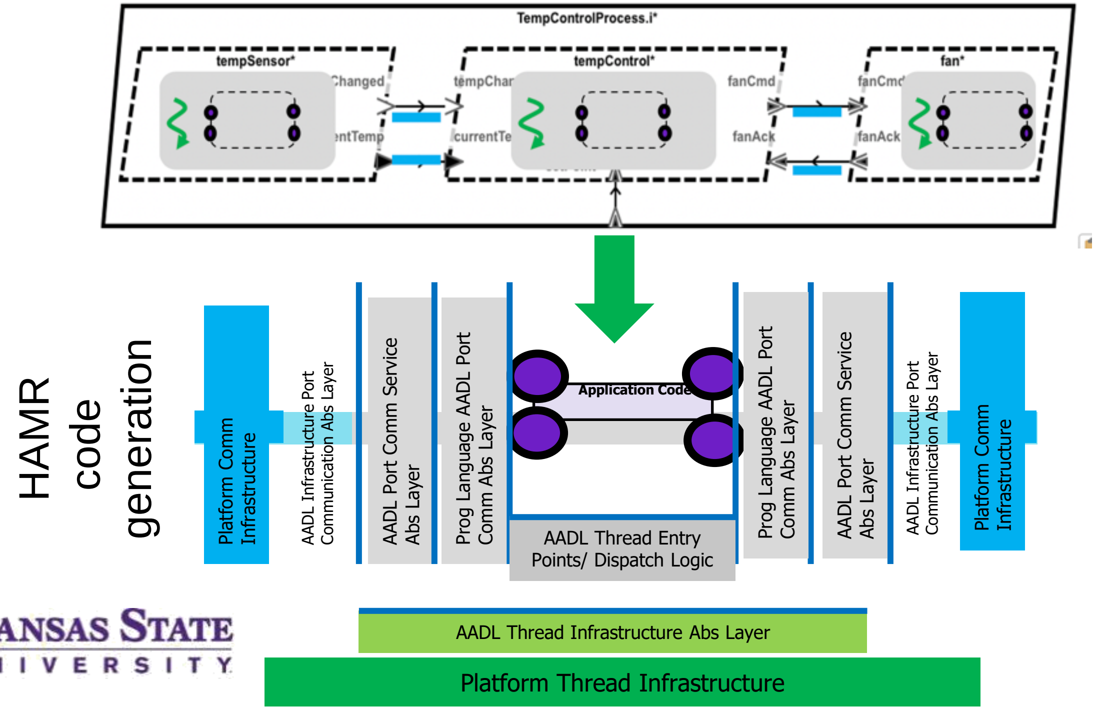


SOFTWARE INFRASTRUCTURE

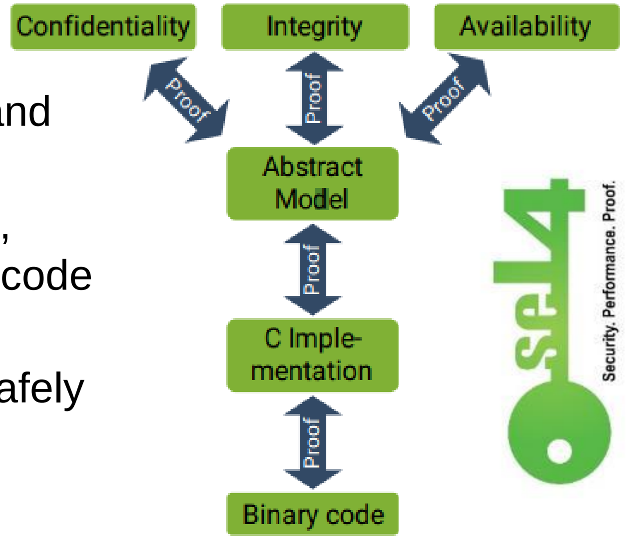
HAMR AND SEL4

- HAMR is a multi-stage translation architecture to address CASE goals of component migration between platforms and information flow control
- Semantic consistency from model to execution
- Ensures model-level analysis applies to deployed code
- Same computational model across different platforms
- Build for multiple target platforms:
  - seL4 / Linux / Virtual Machine
  - Build for workstation / emulator / embedded platform

- seL4 microkernel guarantees partitioning of components and communication, backed by computer-checked proofs
- seL4 guarantees no infiltration, exfiltration, eavesdropping, interference, and provides fault containment for untrusted code
- Ensures soundness of the MBSE design process – components can be analyzed separately and composed safely



Kansas State University

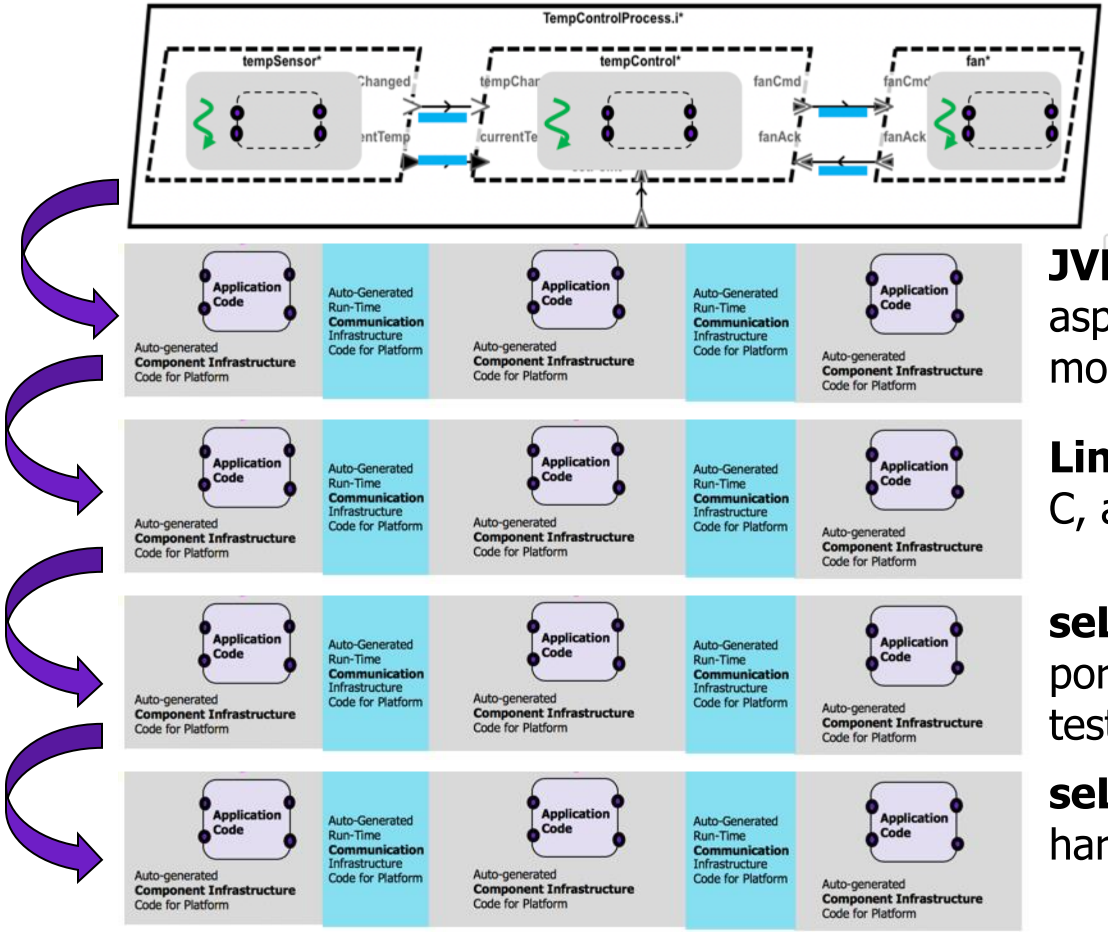


seL4 is…
- An operating system microkernel
- A hypervisor
- Proved correct
- Provably secure
- Fast

Collins Aerospace

HAMR SUPPORTS MULTIPLE LANGUAGE/ PLATFORM COMBINATIONS

The flexibility of being able to easily shift between different platforms was quite useful as the team experimented with building the Phase 2 Experimental Platform assessment deliverable.



AADL / OSATE – design model, types, perform analyses

JVM/Slang – data types, port constraints, basic aspects of application logic, initial unit testing – some mocked up components, many useful visualizations

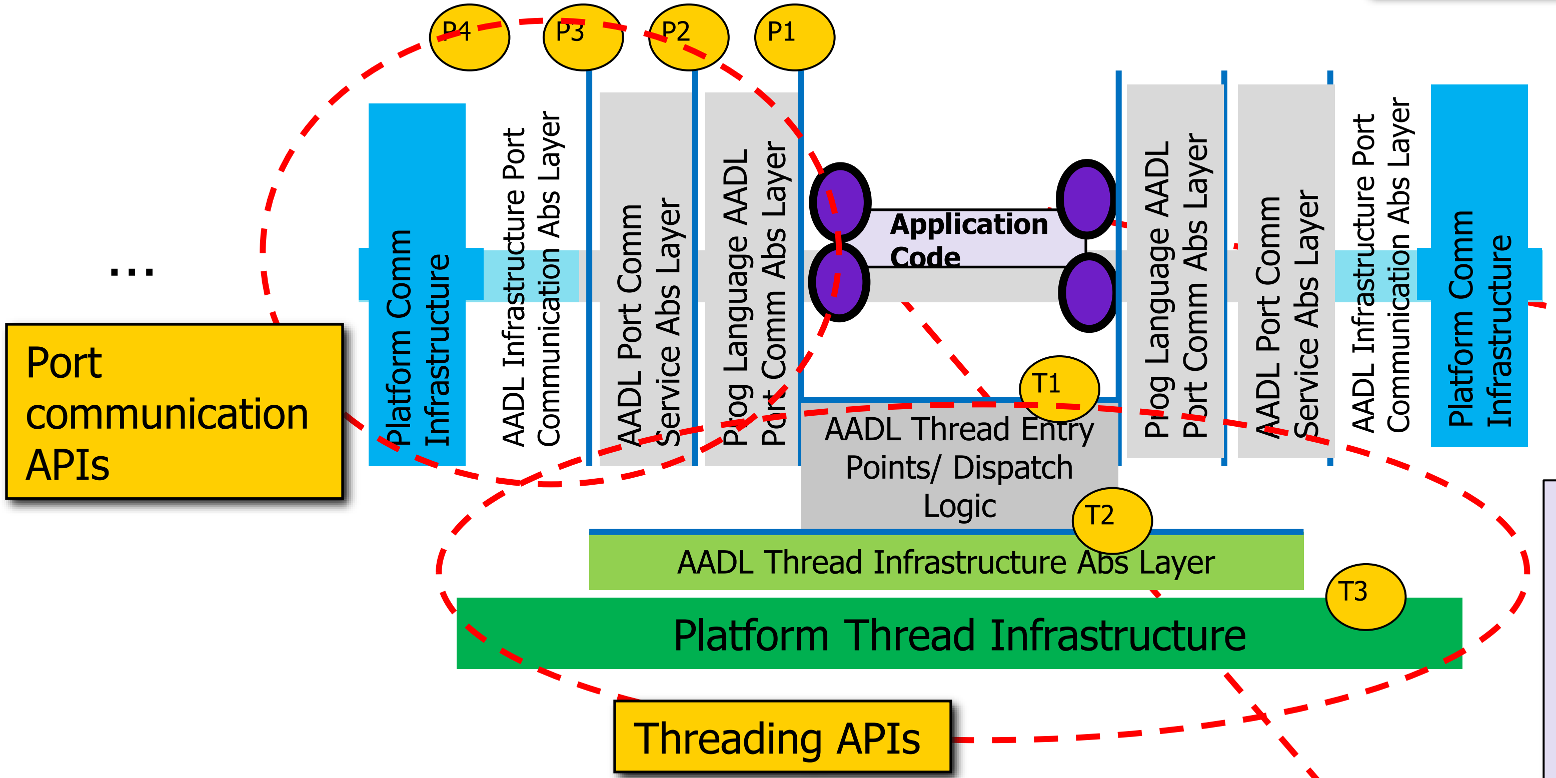
Linux C – compile Slang to C, or manually code C, and debug C implementation, VMs mocked up

seL4 C / Qemu – C application code easily ports to seL4 native components, add in VMs, test/simulate/debug in Qemu

seL4 C / board – seL4 build shifted to actual hardware for final testing and deployment

HAMR ABSTRACTION LAYERS

*By changing the implementation of these layers, we can easily switch to **different platforms** or **different programming languages***



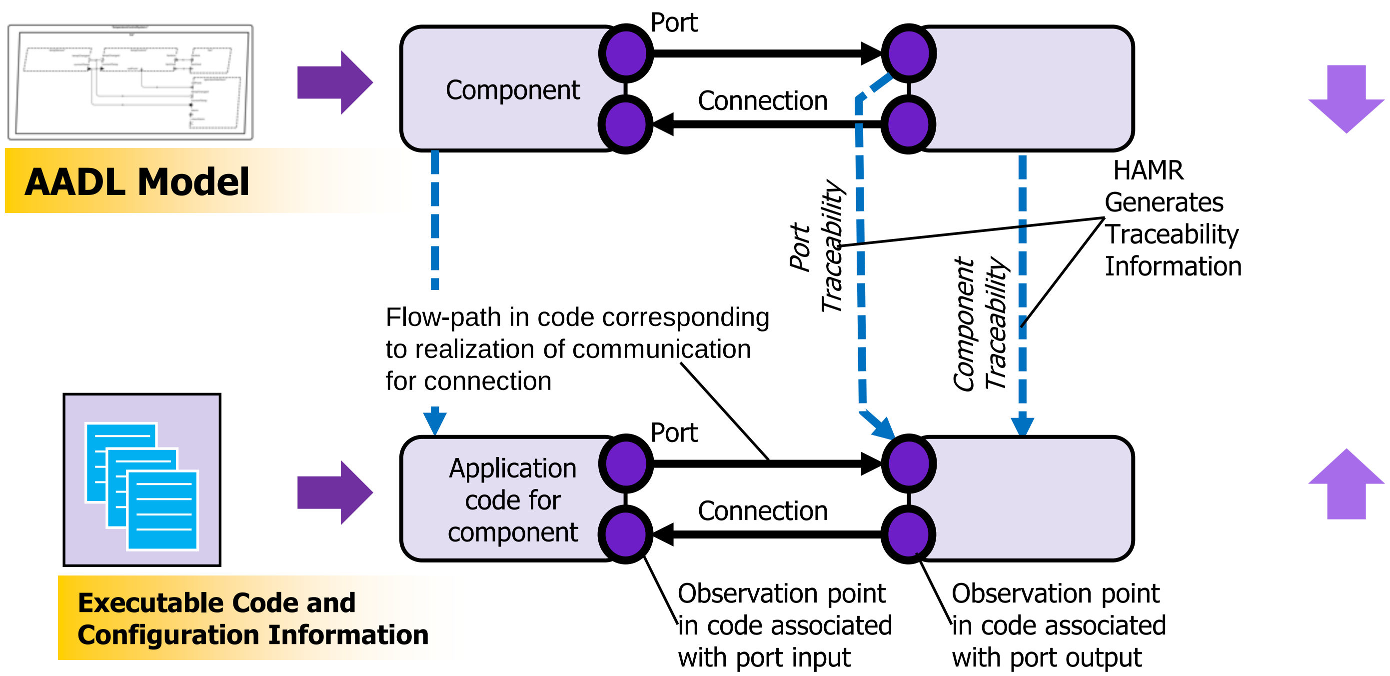

```
Unit hamr_SW_WaypointPlanManagerService_thr_Impl_Impl_timeTriggered_(
  STACK_FRAME
  hamr_SW_WaypointPlanManagerService_thr_Impl_Impl this) {
    bool dataReceived = false;
    size_t numBits = 0;
    uint8_t payload[MAX_PAYLOAD];

    dataReceived =
api_get_ReturnHome__hamr_SW_WaypointPlanManagerService_thr_Impl_Impl(this);
    if (dataReceived) {
      returnHome = true;
    }

    if (returnHome || automationResponse != NULL) {
      dataReceived =
api_get_AirVehicleState__hamr_SW_WaypointPlanManagerService_thr_Impl_Impl(
        this, &numBits, payload);
      if (dataReceived) {
        air_vehicle_state_in_event_data_receive_handler(this, payload);
      }
    }
}
```

HAMR CORRESPONDENCE PROOF

- All information flows in AADL model are accurately preserved in HAMR generated code
- Connects AADL information flow analysis to seL4 security proofs

HAMR generates a topological structure (formally specified)

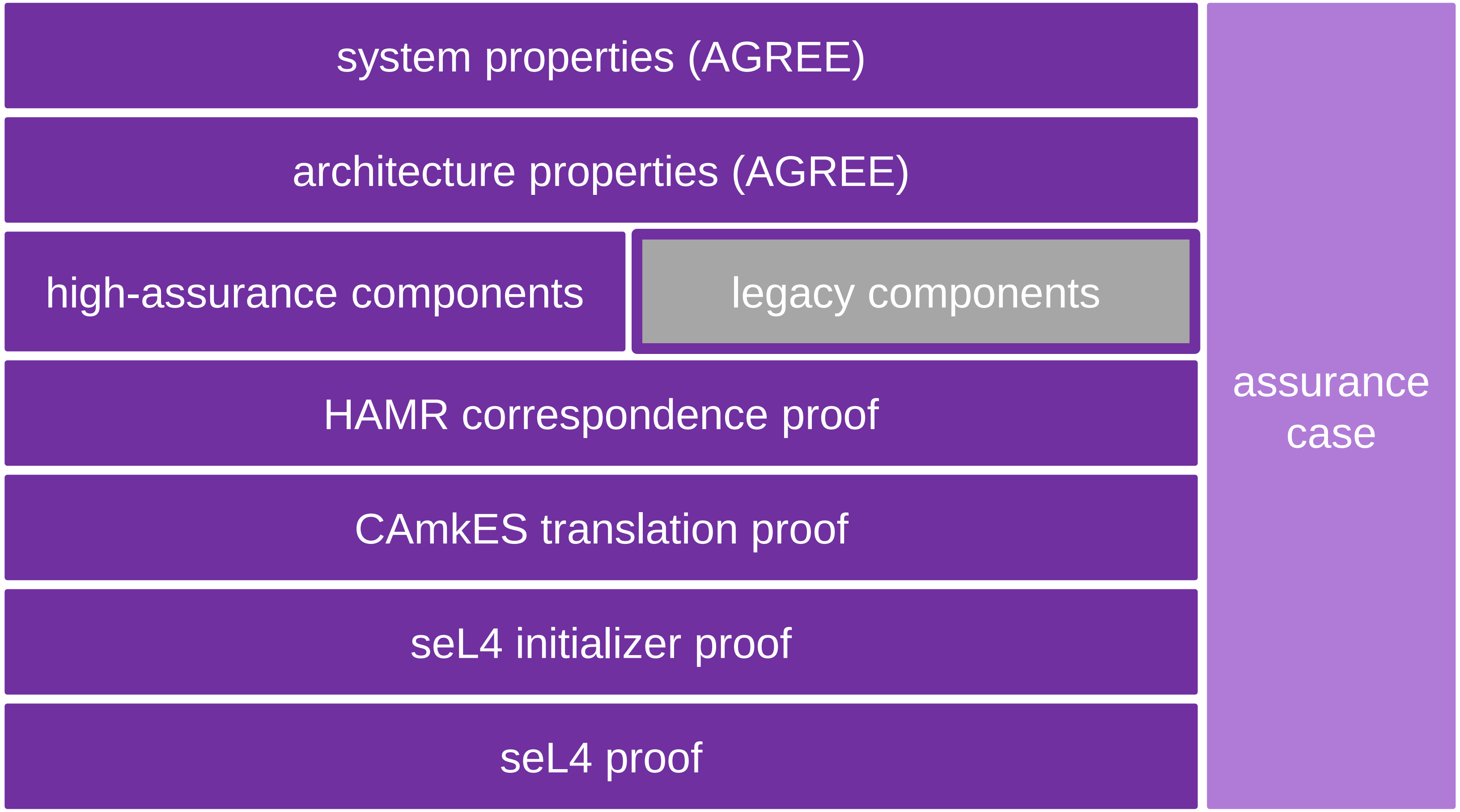
HAMR generates a topological structure (formally specified)



FlowPreservation (formal SMT spec): For every connection between two components in AADL, there is a flow path in the source code between code artifacts associated with the ports.

NoNewFlows (formal SMT spec): For every flow path between two components in the source code, there is a connection in the AADL model between corresponding ports.

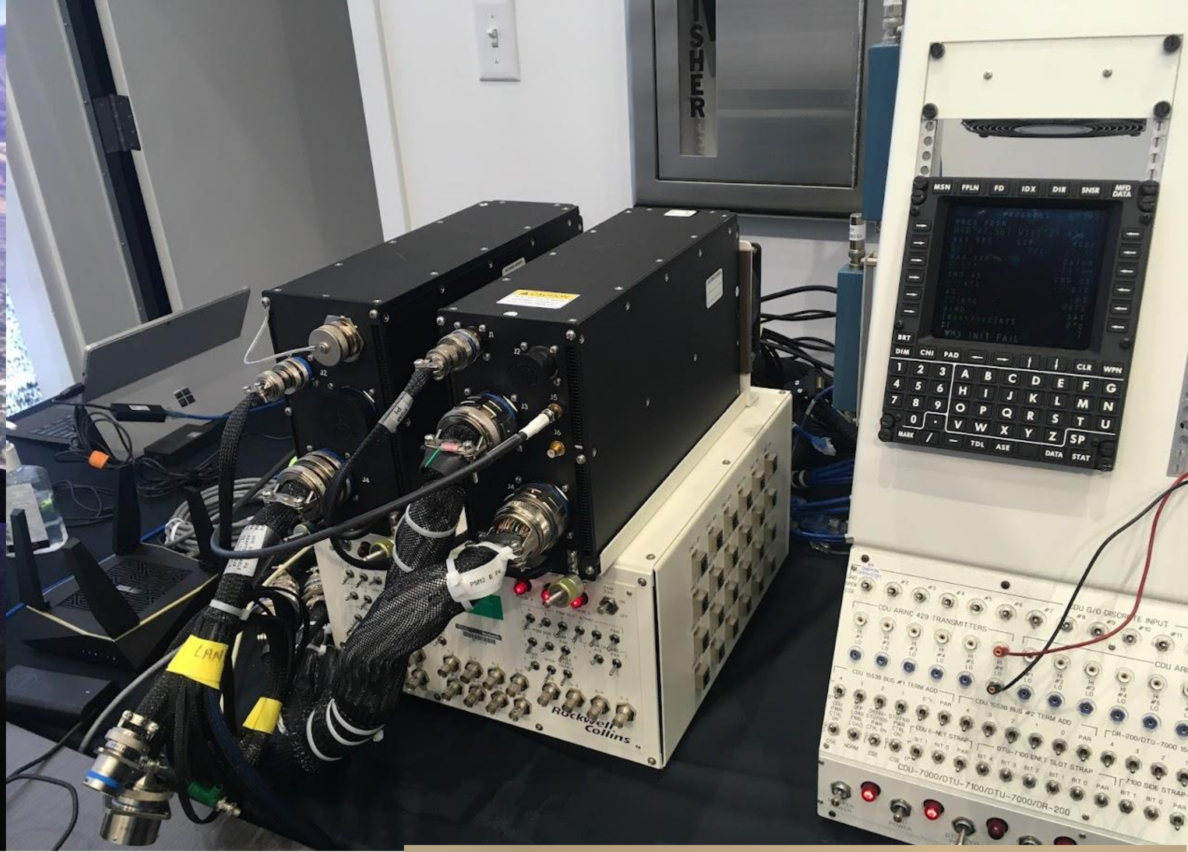
END-TO-END INTEGRATED FORMAL VERIFICATION



Collins Aerospace

CASE FINAL DEMO

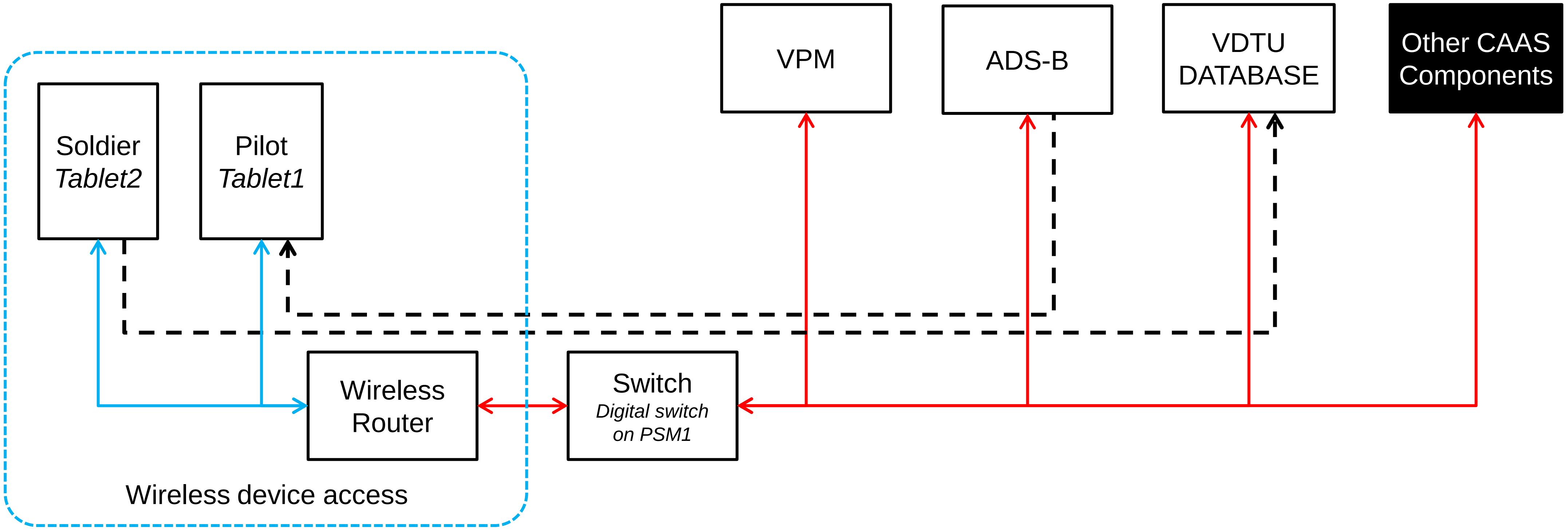
COLLINS CUSTOMER EXPERIENCE CENTER
HUNTSVILLE AL
DECEMBER 2021



FINAL DEMO PLATFORM : BASELINE

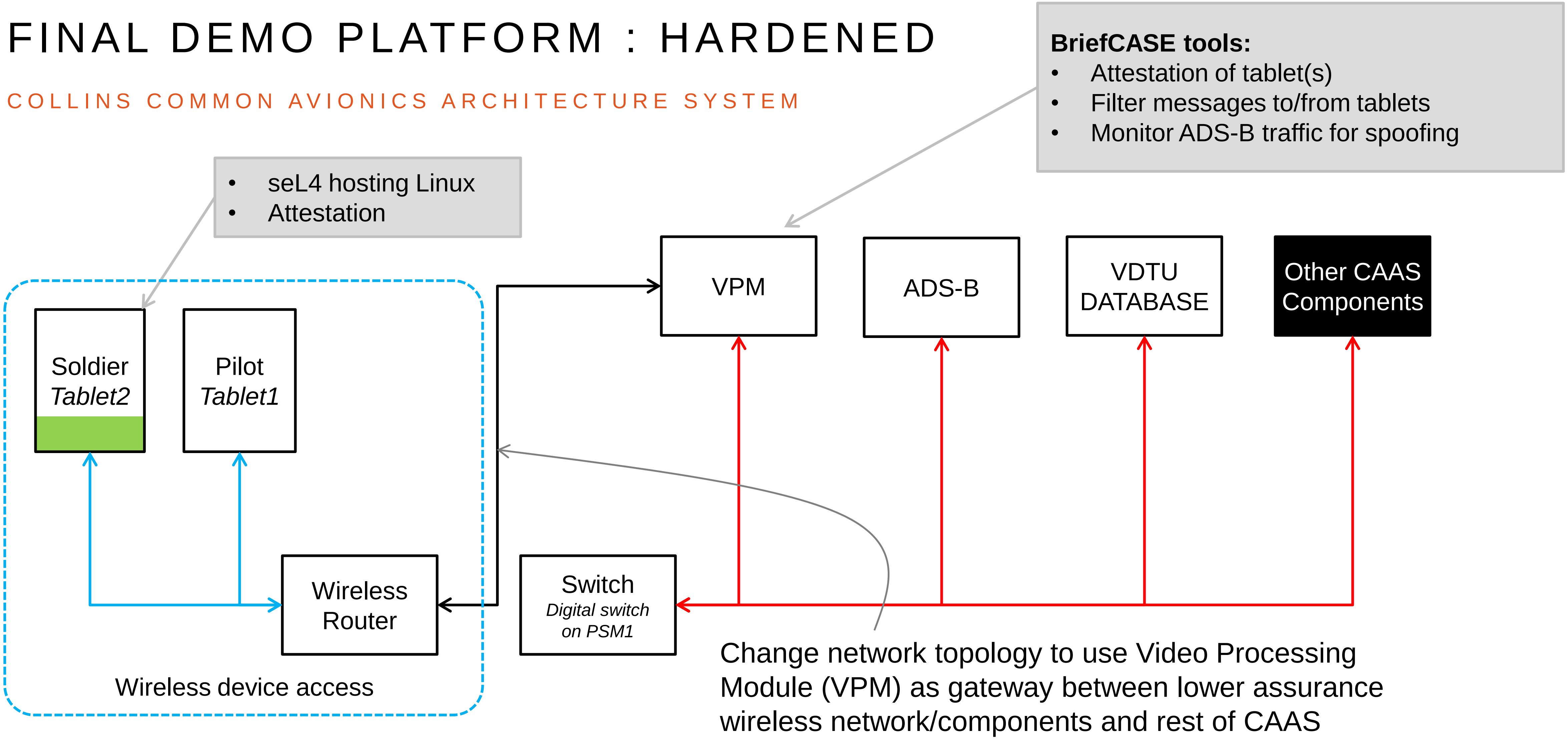
COLLINS COMMON AVIONICS ARCHITECTURE SYSTEM (CAAS)

- Goal : Extend (securely) to add wireless connectivity



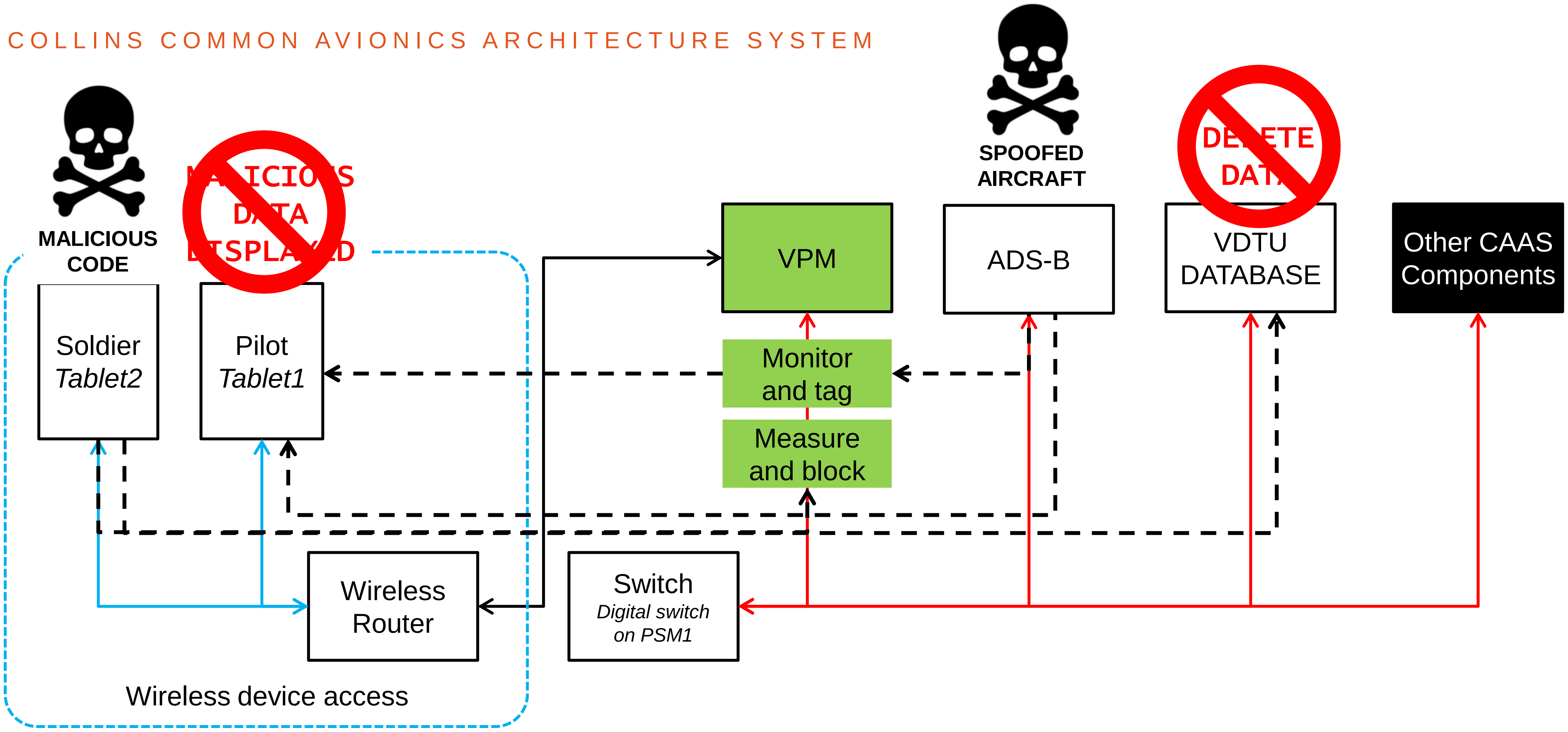
FINAL DEMO PLATFORM : HARDENED

COLLINS COMMON AVIONICS ARCHITECTURE SYSTEM



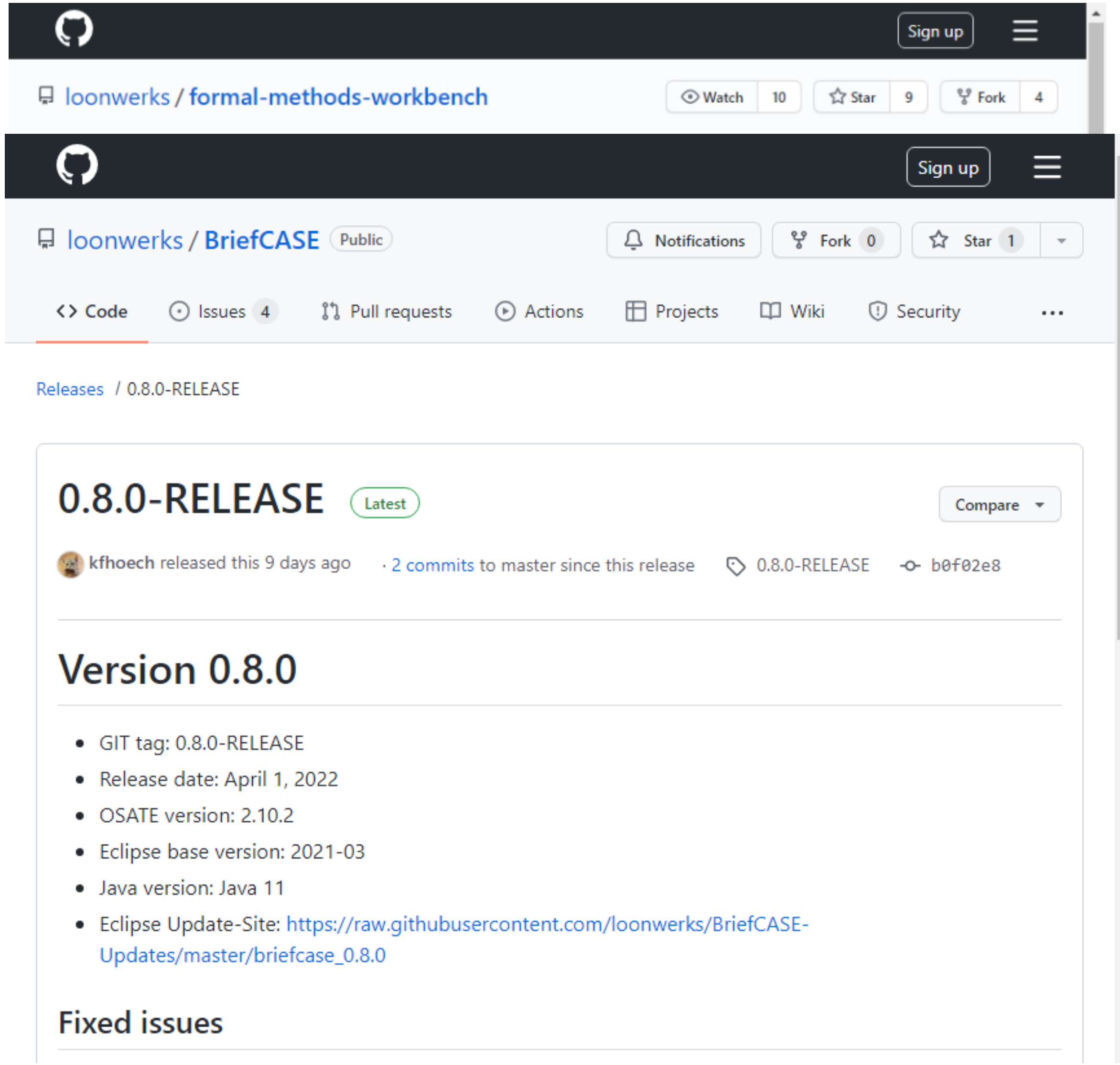
DEMO PLATFORM : ATTACKS

COLLINS COMMON AVIONICS ARCHITECTURE SYSTEM



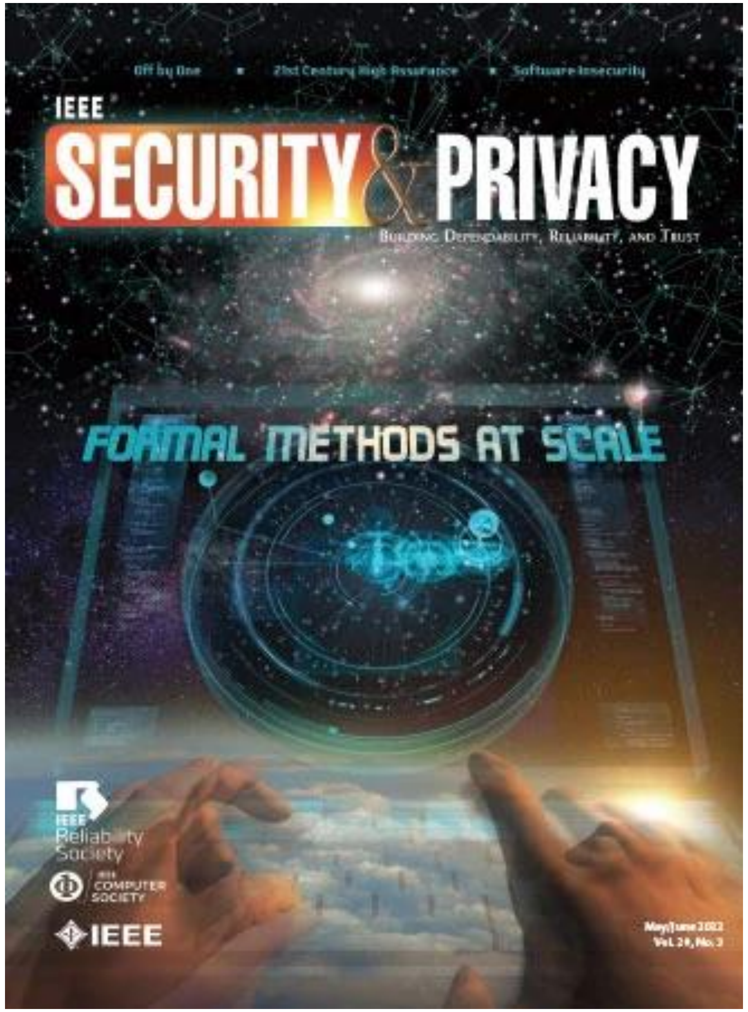
OPEN-SOURCE SOFTWARE TOOL DISTRIBUTION

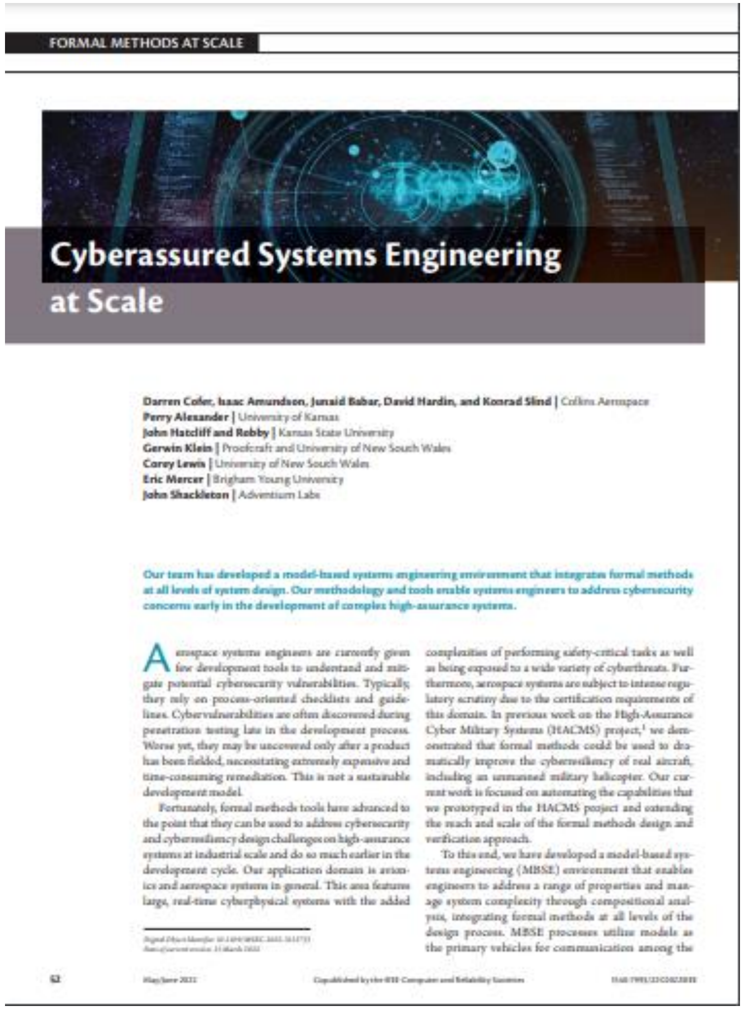
- Tool source code resides in several public GitHub repositories
  - `https://github.com/loonwerks/CASE-Final`
    `also {/BriefCASE, /splat, /AGREE, /Resolute, /jkind}`
  - `https://github.com/ku-sldg`
  - `https://github.com/seL4`
  - `https://github.com/CakeML/cakeml`
  - `https://github.com/sireum`
- Integrated OSATE/AADL tools and plugins
- Vagrant VM
  - Provides automatic, consistent, and reproducible provisioning of VM and native environments for developing and testing all CASE tools
- Documentation
  - Workflow example tutorial and models
  - User Guide
  - Videos, publications
- Overview
  - `http://loonwerks.com/projects/case.html`



Collins Aerospace

CYBER-ASSURED SYSTEMS ENGINEERING AT SCALE





FORMAL METHODS AT SCALE

Cyberassured Systems Engineering at Scale

Darren Cofer, Isaac Amundson, Junaid Babar, David Hardin, and Konrad Slind | Collins Aerospace
Perry Alexander | University of Kansas
John Hatcliff and Robby | Kansas State University
Gerwin Klein | Proofcraft and University of New South Wales
Corey Lewis | University of New South Wales
Eric Mercer | Brigham Young University
John Shackleton | Adventium Labs

Our team has developed a model-based systems engineering environment that integrates formal methods at all levels of system design. Our methodology and tools enable systems engineers to address cybersecurity concerns early in the development of complex high-assurance systems.

Aerospace systems engineers are currently given few development tools to understand and mitigate potential cybersecurity vulnerabilities. Typically, they rely on process-oriented checklists and guidelines. Cybervulnerabilities are often discovered during penetration testing late in the development process. Worse yet, they may be uncovered only after a product has been fielded, necessitating extremely expensive and time-consuming remediation. This is not a sustainable development model.

Fortunately, formal methods tools have advanced to the point that they can be used to address cybersecurity and cyberresiliency design challenges on high-assurance systems at industrial scale and do so much earlier in the development cycle. Our application domain is avionics and aerospace systems in general. This area features large, real-time cyberphysical systems with the added complexities of performing safety-critical tasks as well as being exposed to a wide variety of cyberthreats. Furthermore, aerospace systems are subject to intense regulatory scrutiny due to the certification requirements of this domain. In previous work on the High-Assurance Cyber Military Systems (HACMS) project,[1] we demonstrated that formal methods could be used to dramatically improve the cyberresiliency of real aircraft, including an unmanned military helicopter. Our current work is focused on automating the capabilities that we prototyped in the HACMS project and extending the reach and scale of the formal methods design and verification approach.

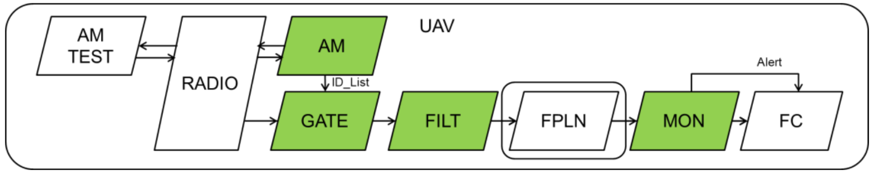
To this end, we have developed a model-based systems engineering (MBSE) environment that enables engineers to address a range of properties and manage system complexity through compositional analysis, integrating formal methods at all levels of the design process. MBSE processes utilize models as the primary vehicles for communication among the

Also available at: https://loonwerks.com/publications/cofer2022secpriv.html

BRIEFCASE TUTORIAL

Come to our Bootcamp session!

- Learn to use the BriefCASE tools
- Address cyber-resiliency requirements on a small example, analyze properties, generate code, create assurance argument, build and run system on seL4 (in QEMU)
- VM with all tools, models, and instructions
- Get it from Darren or Isaac, or download from github before the session



Day 4 (Bootcamp, on-site participants only) 13 October 2022		
9:00 - 10:00	Bootcamp	seL4: from zero to hello world Ihor Kuz, Kry10
Break		
10:15 - 10:45	Bootcamp	CAmkES Sebastian Eckl, HENSOLDT Cyber
10:45 - 12:00	Bootcamp	TRENTOS Sebastian Eckl, HENSOLDT Cyber
Break		
13:00 - 14:30	Bootcamp	The seL4 Core Platform (seL4CP) Ivan Velickovic & Peter Chubb, UNSW
14:30 - 15:15	Bootcamp	DornerWorks' VM Composer: the easy button for virtualized seL4-based systems Chris Guikema & Robbie VanVossen, Dornerworks
Break		
15:30 - 17:00	Bootcamp	BriefCASE tutorial Isaac Amundson & Darren Cofer, Collins Aerospace
17:00 - 17:05	Plenary	*Concluding remarks*

QUESTIONS?

Collins Aerospace