Industrial Scale Proof Engineering for Critical Trustworthy Applications (INSPECTA)

DARPA PROVERS : Pipelined Reasoning of Verifiers Enabling Robust Systems

High Confidence Software and Systems | HCSS 2024
Darren Cofer | Principal Investigator

PROVERS PROGRAM OBJECTIVES

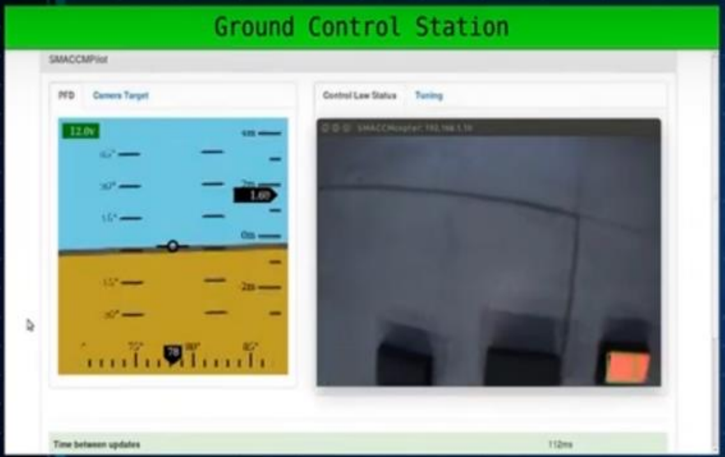
Develop automated, scalable formal methods tools that are integrated into traditional software development pipelines.

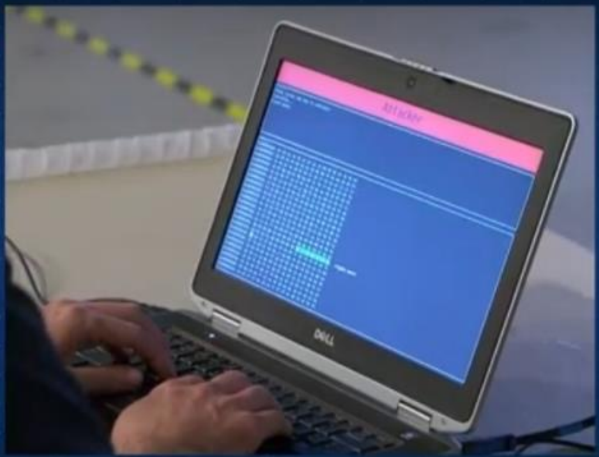
DEFENSE & AEROSPACE

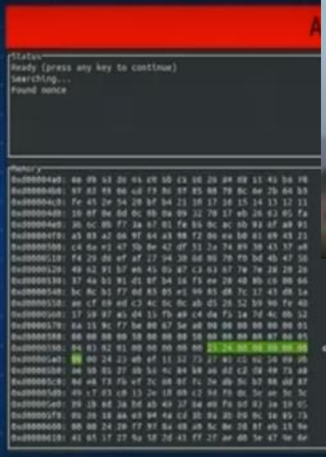
Enable traditional software developers to incrementally produce and maintain high-assurance national security systems.

Adoption of formal methods in Defense Industrial Base development processes

HIGH ASSURANCE CYBER MILITARY SYSTEMS (HACMS)






```
Status
Ready (press any key to continue)
Searching...
Found nonce
```
```
Memory
[illegible]
```

DEFCON AEROSPACE VILLAGE

Loonwerks.com/projects/hacms

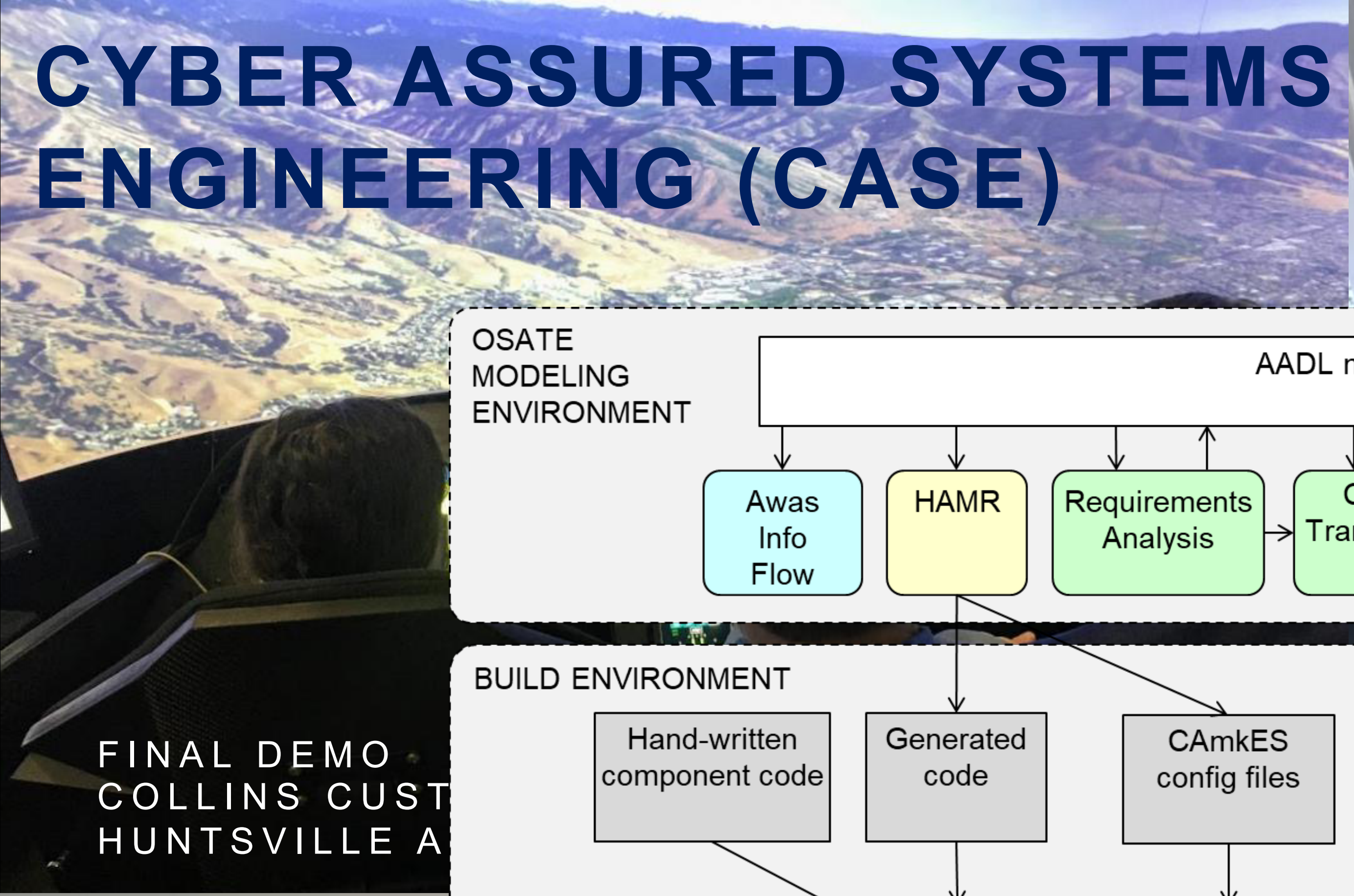
DARPA @DARPA

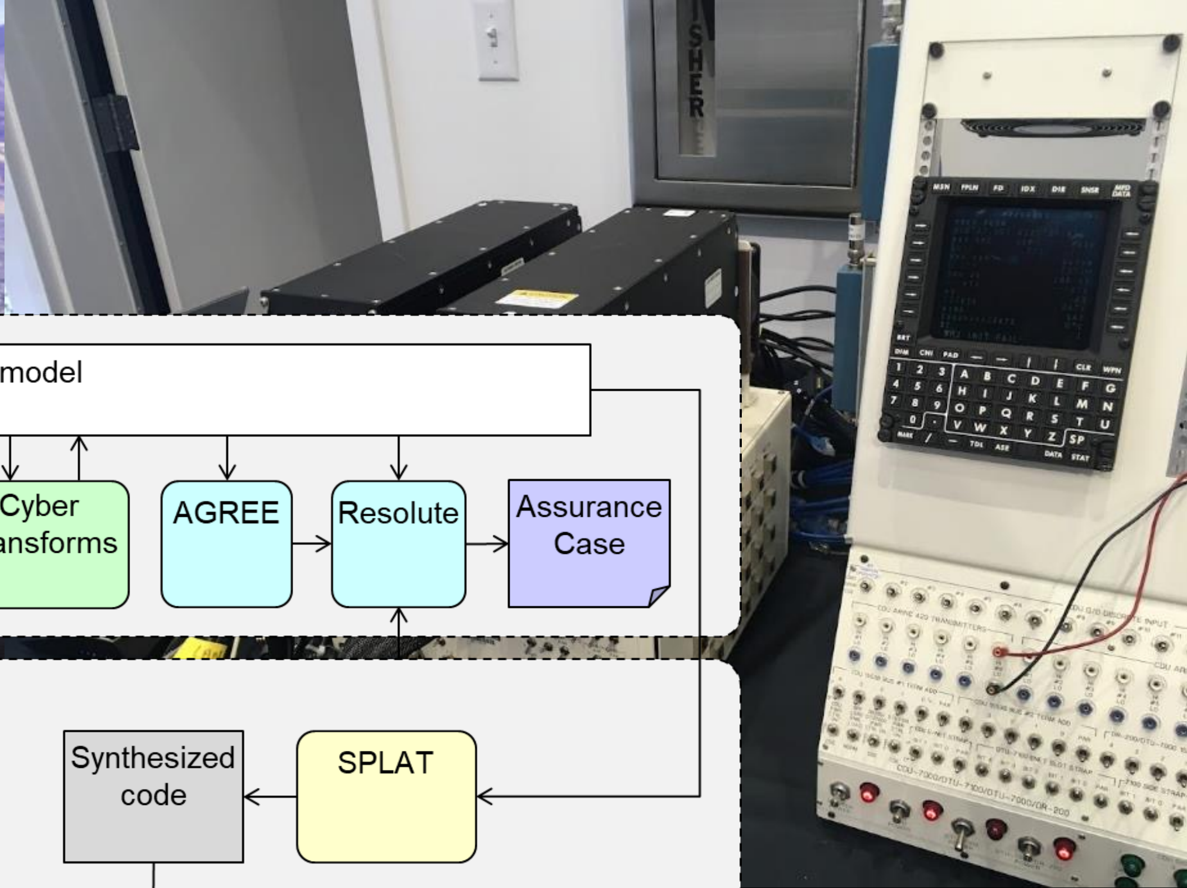
We brought a hackable quadcopter with defenses built on our HACMS program to @defcon #AerospaceVillage. As program manager @raymondrichards reports, many attempts to breakthrough were made but none were successful. Formal methods FTW!

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

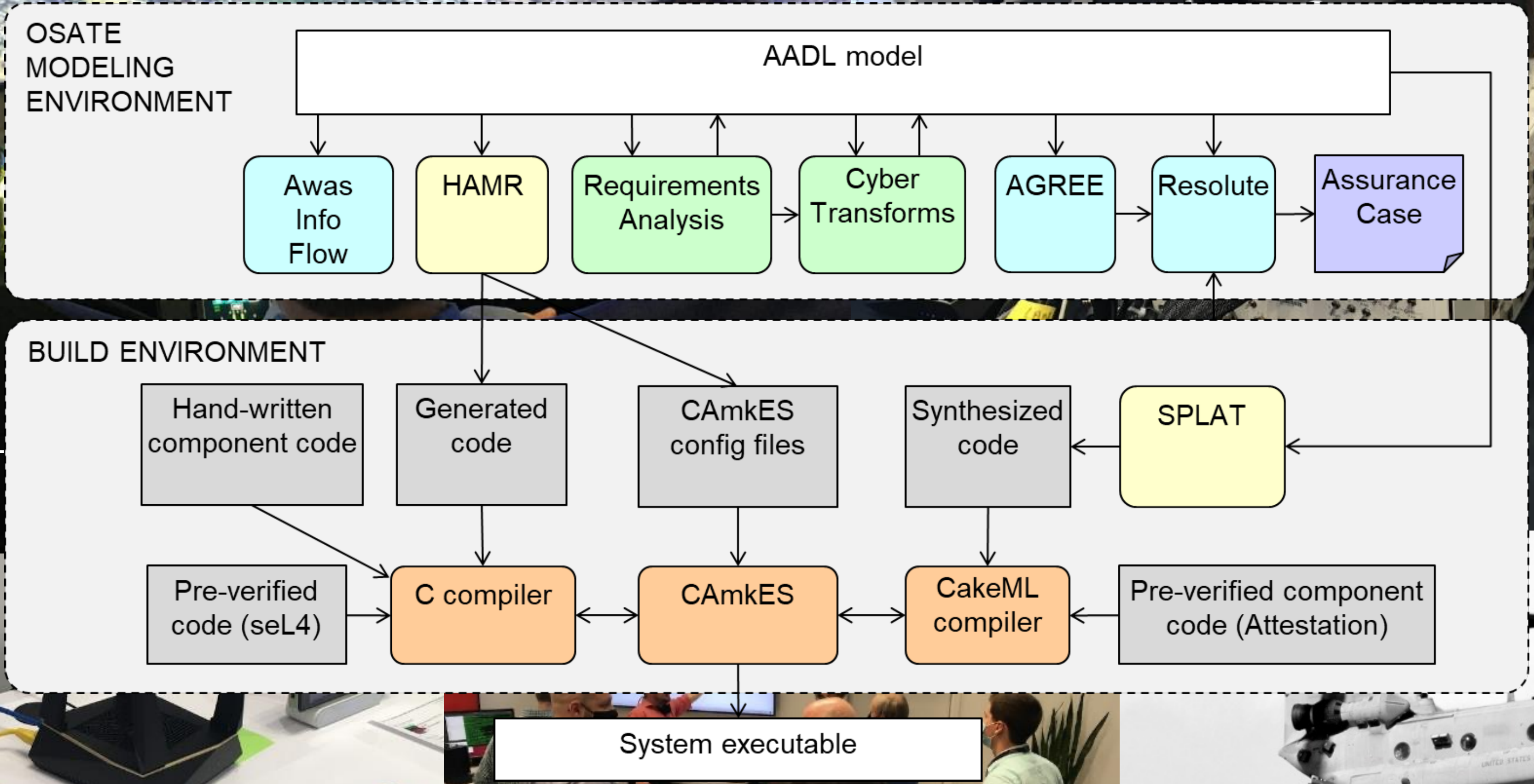
Collins Aerospace

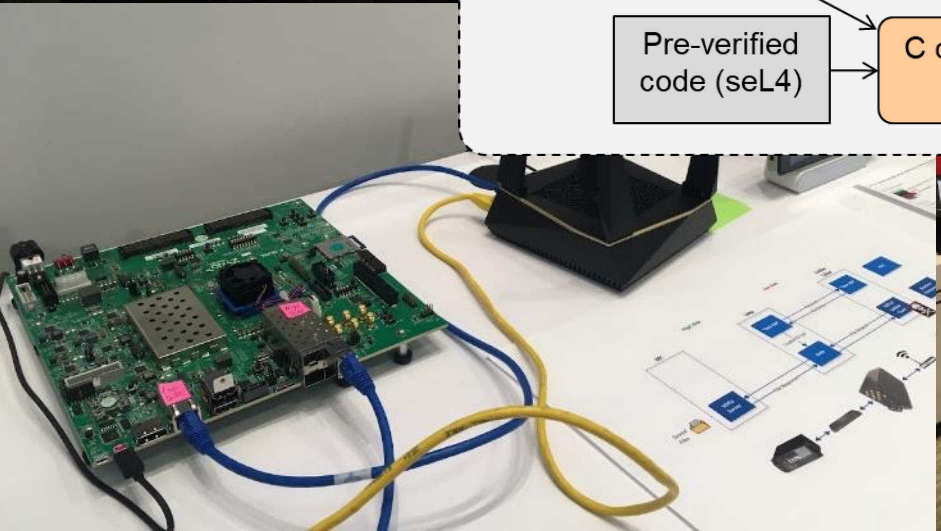
CYBER ASSURED SYSTEMS ENGINEERING (CASE)

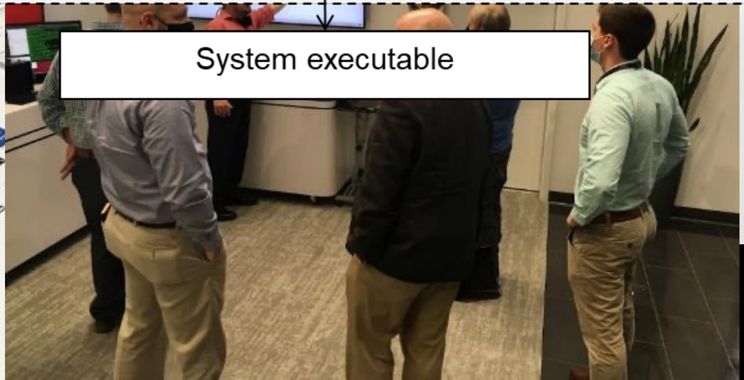


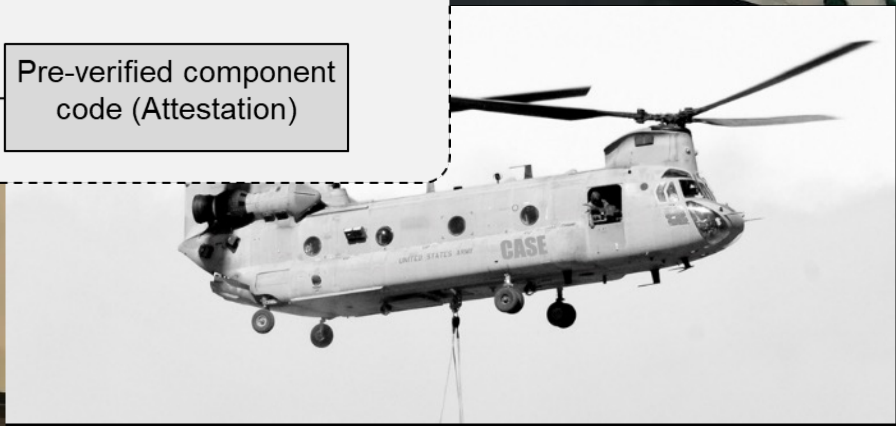


FINAL DEMO
COLLINS CUST
HUNTSVILLE A









INSPECTA : HIGHLIGHTS

- Our workflow and tools will address the **entire software development stack** from requirements and system models, to component source code, through build and deployment on the seL4 secure microkernel, linked by formal verification at each level.
- We will achieve scalability for complex defense systems through **compositional reasoning** at the system level and automated analysis of components based on powerful, cloud-based solvers.
- We will achieve the **highest levels of assurance** by building upon the best available technologies and leveraging our experience from recent research programs as a starting point.
- Our tools will be **integrated with current Collins workflow** automation processes and applied to defense and aerospace products currently under development to demonstrate their usability, practicality, and effectiveness.
- **Formal verification will be made accessible to non-formal methods experts** through automated analysis with streamlined user feedback and generalized proofs that are robust to changes, augmented by automated repair tools.
- Our **framework is adaptable and extendable** to allow incorporation of results from other researchers, including other specification languages, other source code languages, and other operating system targets.
- Our **access to critical defense and aerospace products** in both commercial and military domains served by Collins will serve as the basis for compelling demonstrations of INSPECTA technologies.

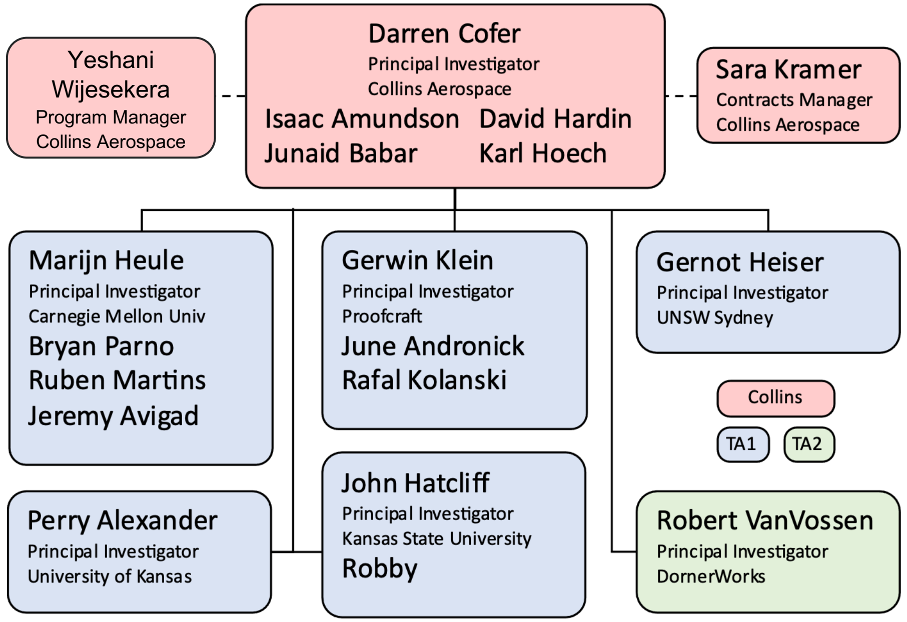
INSPECTA TEAM

Technology Area 1 – Proof Engineering

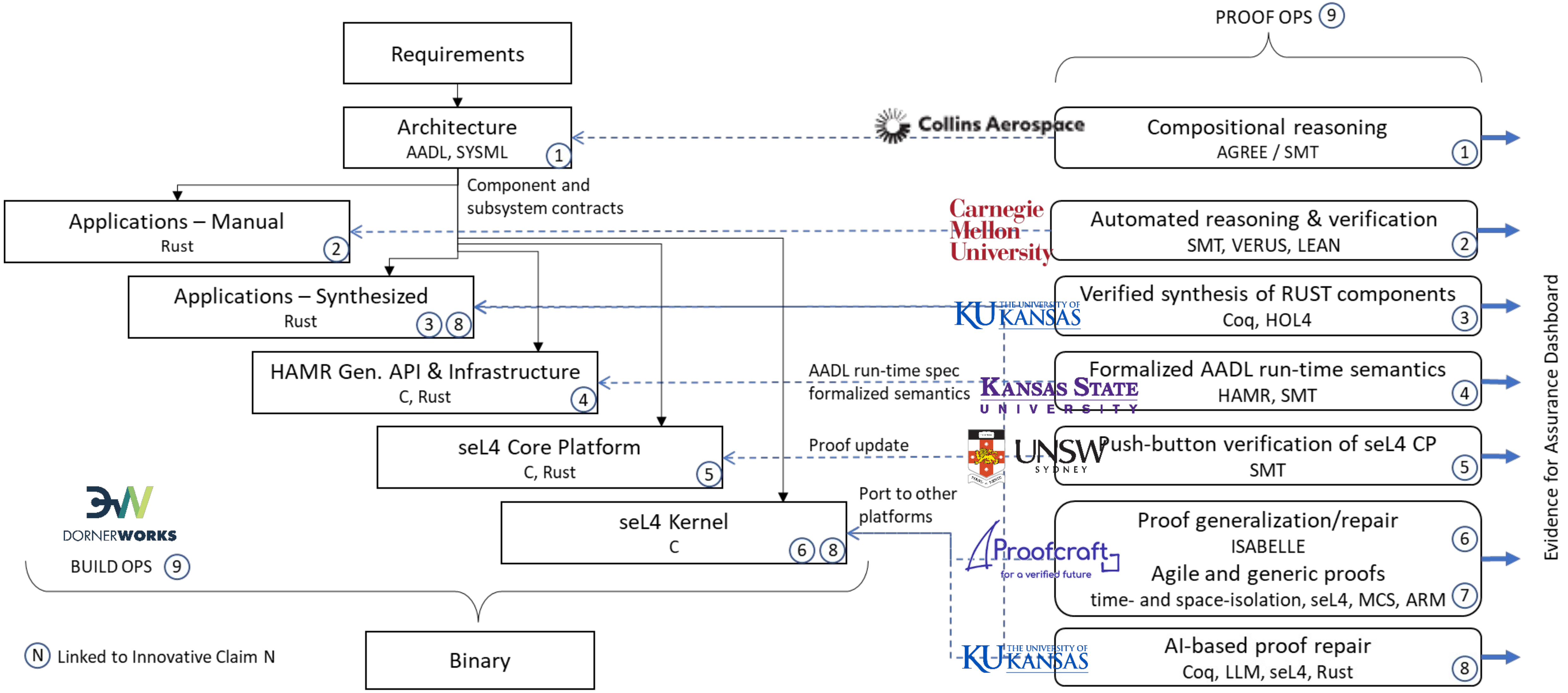
- **Collins**: System requirements, model-based compositional reasoning, workflow integration and assurance gathering, user feedback and measurement
- **CMU**: Software component analysis, scalable SAT/SMT analysis, Rust software verification environment
- **KS State Univ**: Model-based build framework, formal model of AADL, code generation for seL4 and other OS
- **Proofcraft**: Robust and generalized proofs, seL4 verification
- **UNSW Sydney**: Push-button verification of seL4 microkit, seL4 OS components
- **Univ of KS**: Component software synthesis, AI-Enhanced proof repair, lifecycle attestation for workflow

Technology Area 2 – Platform Development

- **Collins**: Provide platforms for demonstration of TA1 tools, requirements changes to evaluate tool effectiveness, including US Army (Air) Launched Effects
- **DornerWorks**: Develop open demonstration platform based on Army SBIR with Collins, UAS mission software running on seL4



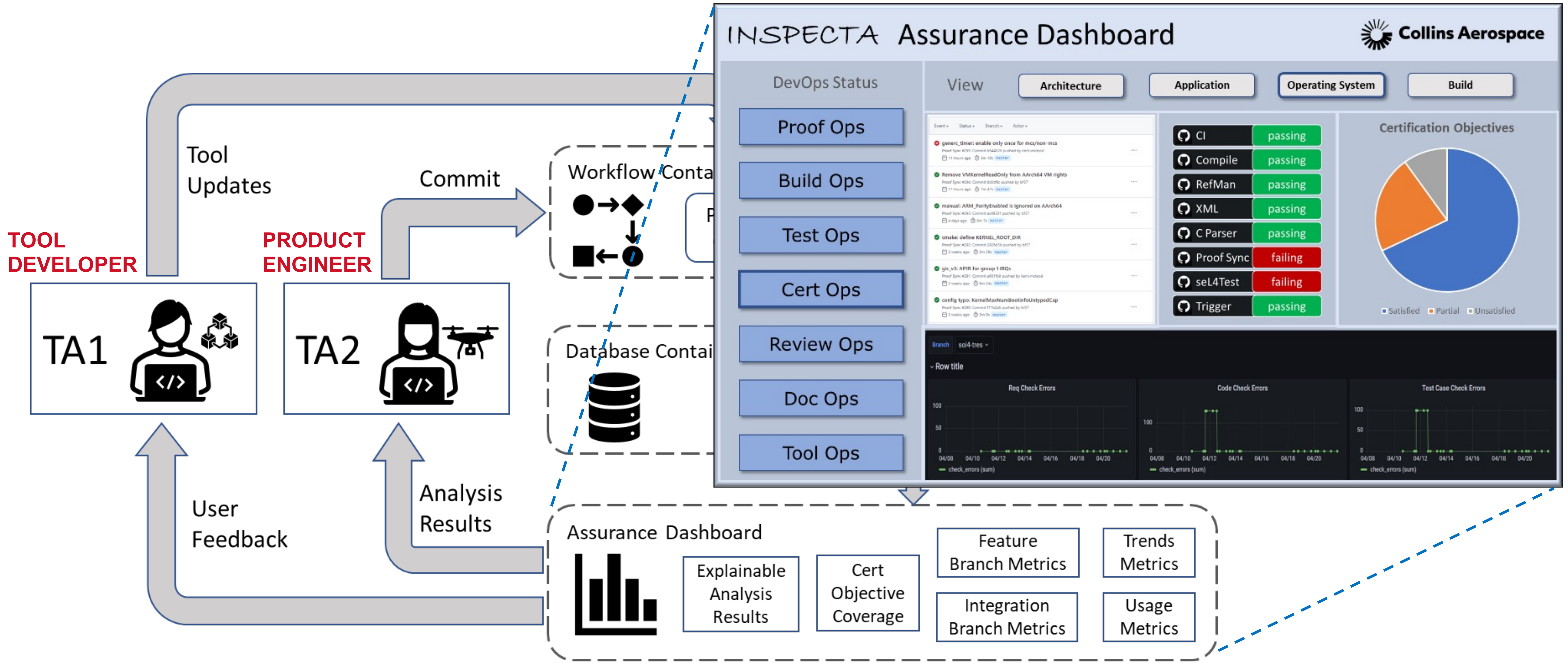
INSPECTA : TA1 TOOLS



Ⓝ Linked to Innovative Claim N

Collins Aerospace

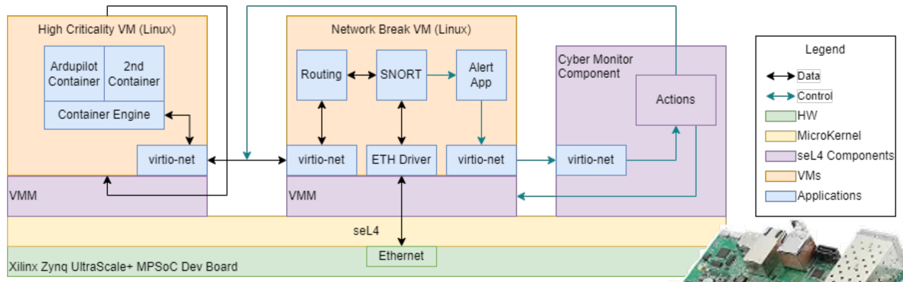
INSPECTA : TA1 WORKFLOW

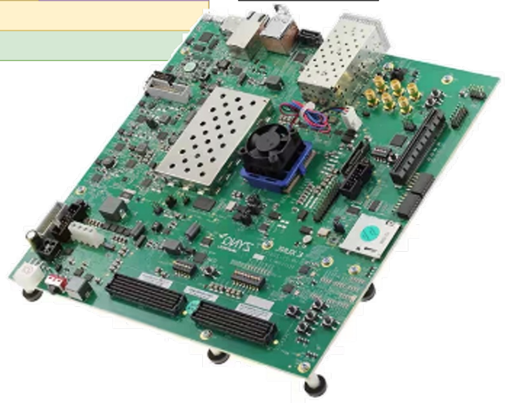


TA2 PLATFORM DEVELOPMENT

Open Platform

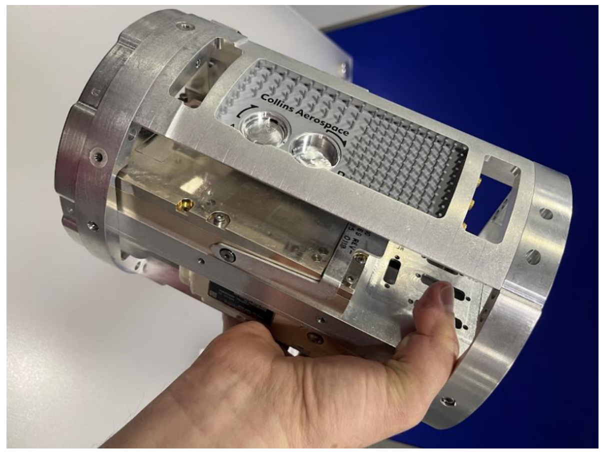
- Developed and supported by DornerWorks
- Unrestricted UAV mission software, system model with formal properties, multiple VMs, Rust software components, seL4 kernel
- Xilinx Zynq UltraScale+ MPSoC-based development board (equivalent to RapidEdge)





Restricted Platform

- Collins Air Launched Effects (ALE) Mission Computer
- Tube-launched air vehicle, payload(s), & mission system applications for autonomously delivery of kinetic or non-kinetic effects
- RapidEdge provides mission computing for ALE, supporting autonomous functionality, and includes radios for communication and handling multiple levels of classified data
- Based on same computer hardware family as Open Platform



Collins Aerospace

(AIR) LAUNCHED EFFECTS

Distribution A. Approved for public release: distribution unlimited

https://youtu.be/SpnGE2CCx2w

https://youtu.be/OosofUsbaRc

Both air and ground launched options supporting a wide variety of missions

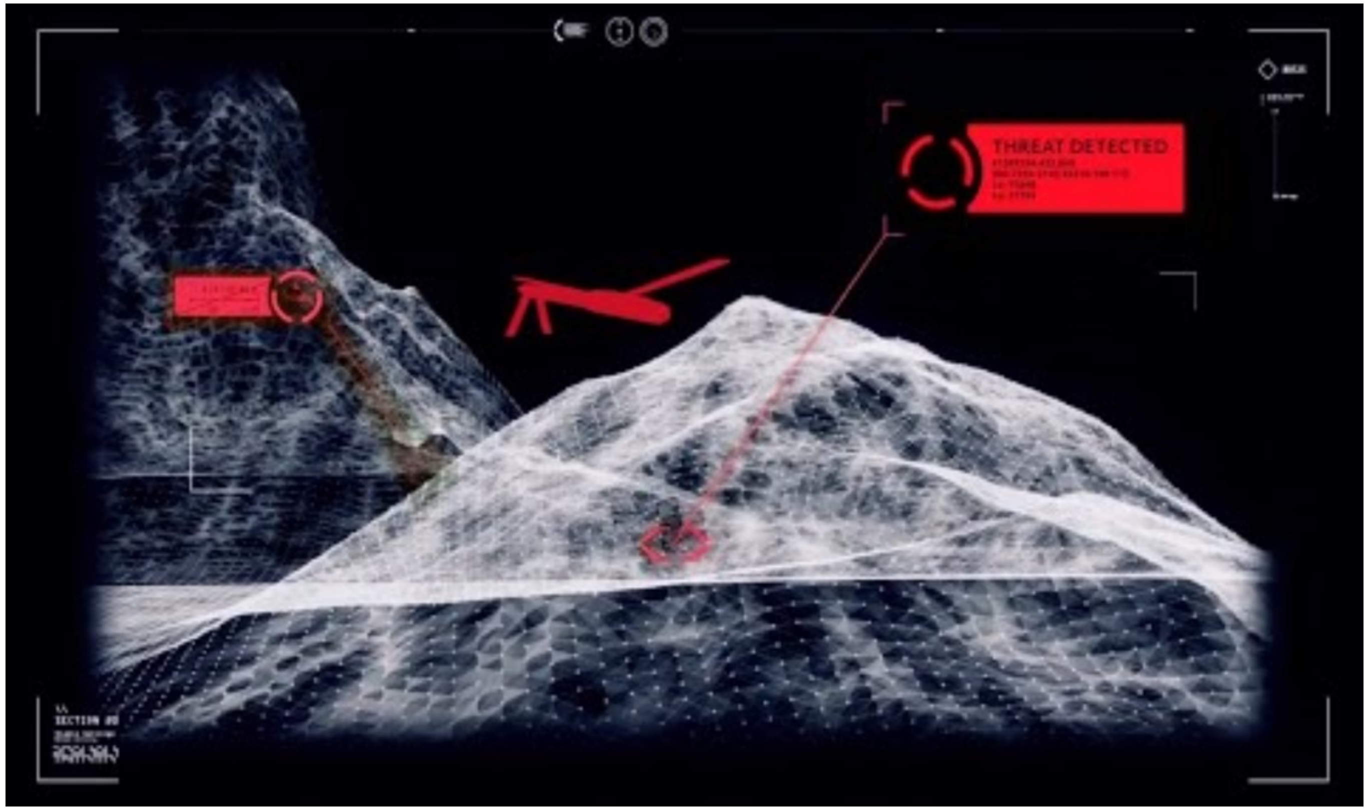
Collins Aerospace

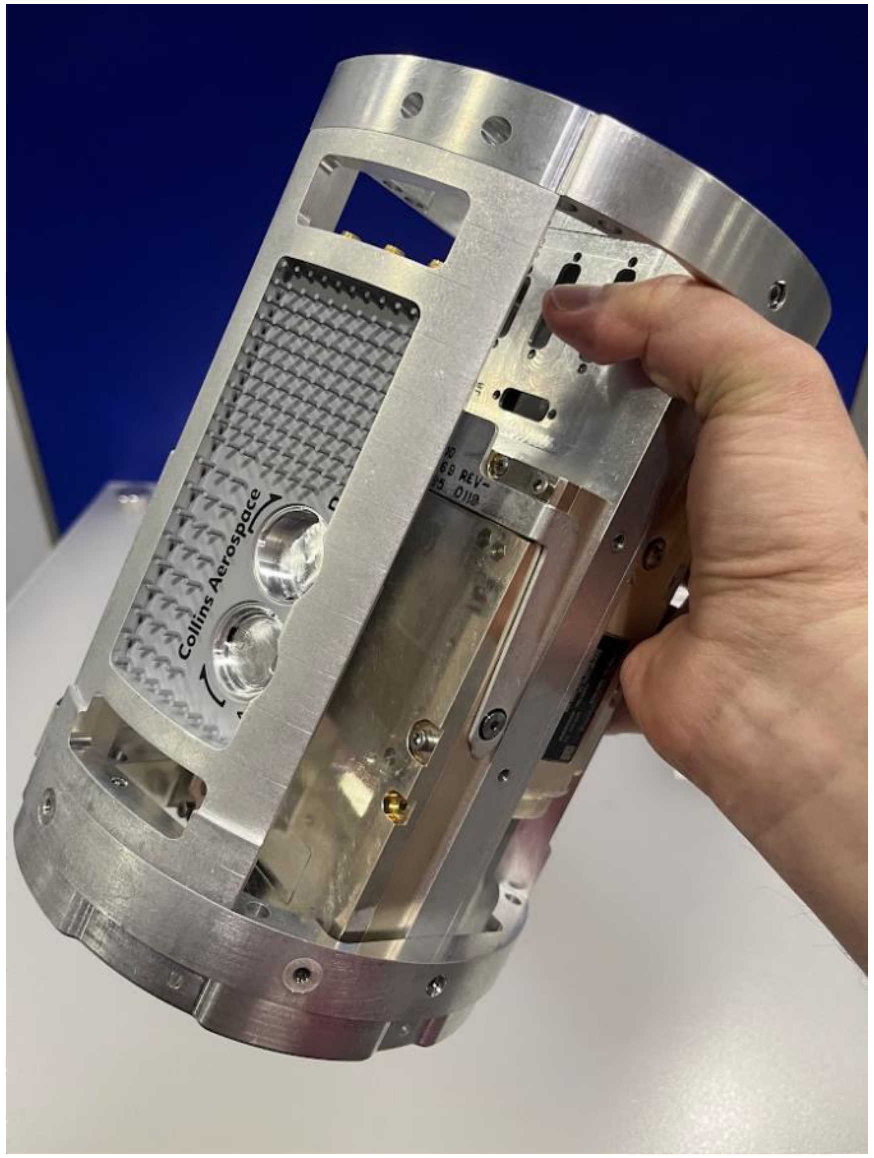
RESTRICTED DEMO PLATFORM

Collins ALE Mission Computer

- RapidEdge provides mission computing for ALE, supporting autonomous functionality, and includes radios for communication and handling multiple levels of classified data

https://youtu.be/SwPJHmZQMaM



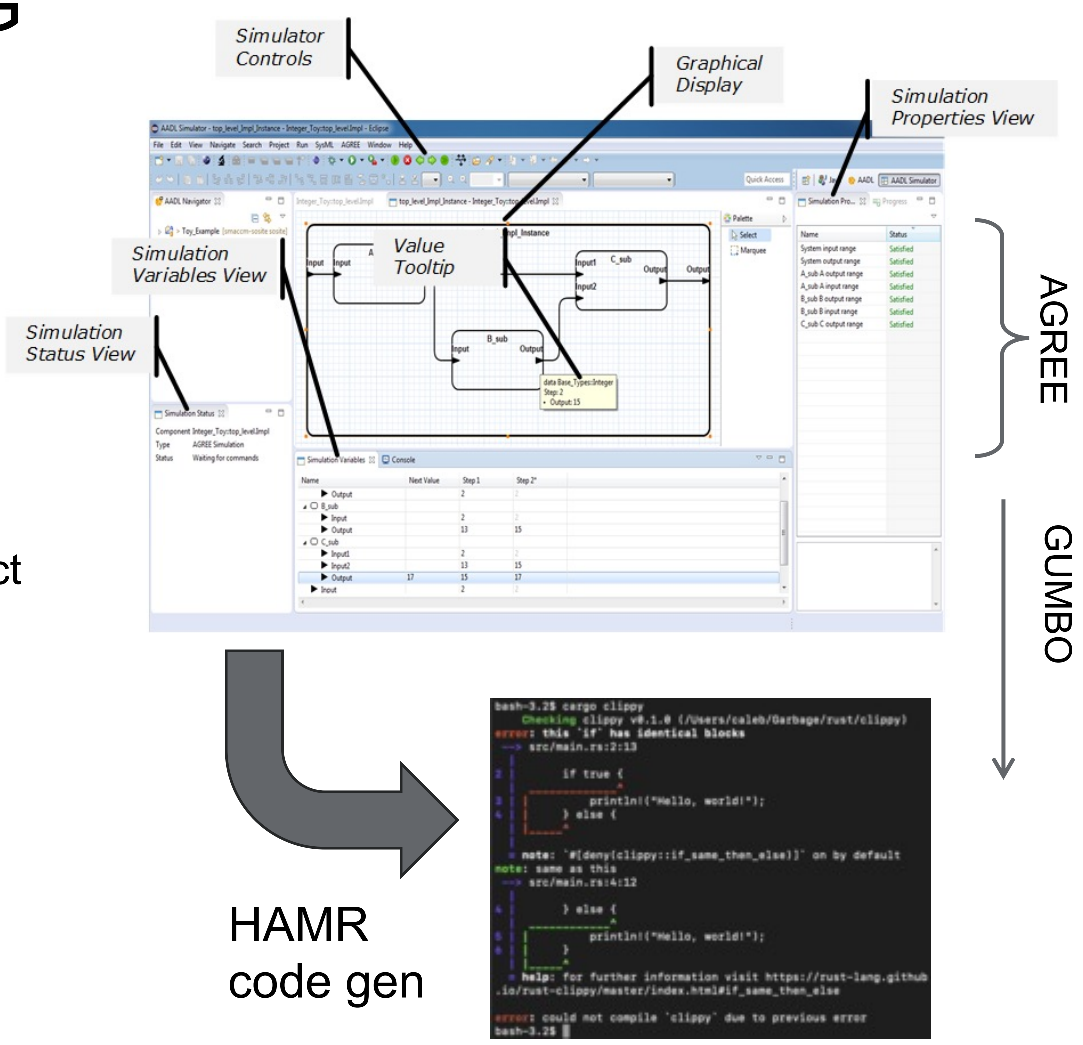


Collins RapidEdge^{TM} Mission System
https://ale.collins.ixperiential.com

Collins Aerospace

COLLINS : ARCHITECTURE MODEL AND COMPOSITIONAL REASONING

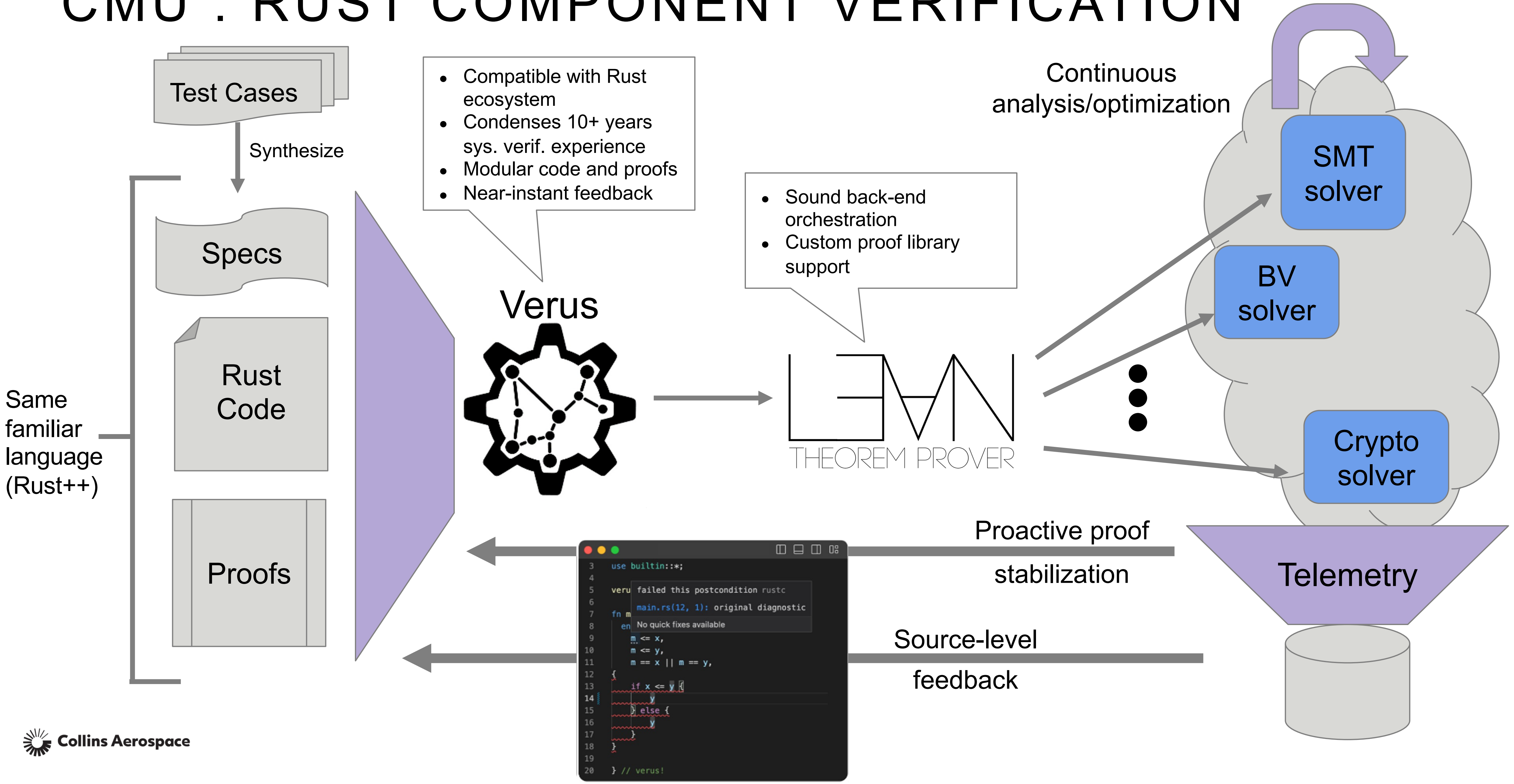
- Develop language abstractions to simplify contract specification in AGREE
- Enhance graphical interface for AGREE that enables engineers to walk through generated counterexamples more intuitively
- Establish traceability to proof obligations at the source code level
  - Achieved through tighter coupling with KSU's GUMBO contract language
- Integrate AGREE into DevOps workflow
- Compositional reasoning for SysMLv2
  - OMG Real-Time Embedded Safety Critical Systems working group



```
bash-3.2$ cargo clippy
    Checking clippy v0.1.0 (/Users/caleb/Garbage/rust/clippy)
error: this `if` has identical blocks
 --> src/main.rs:2:13
  |
2 |         if true {
  |                 ^
  |  _____________|
3 | |           println!("Hello, world!");
4 | |       } else {
  | |_______^
  |
  = note: `#[deny(clippy::if_same_then_else)]` on by default
note: same as this
 --> src/main.rs:4:12
  |
4 |       } else {
  |  ____________^
5 | |           println!("Hello, world!");
6 | |       }
  | |_______^
  = help: for further information visit https://rust-lang.github
.io/rust-clippy/master/index.html#if_same_then_else

error: could not compile `clippy` due to previous error
bash-3.2$
```

Collins Aerospace

CMU : RUST COMPONENT VERIFICATION


Test Cases
Synthesize
Specs
Rust Code
Proofs
Same familiar language (Rust++)
Compatible with Rust ecosystem
Condenses 10+ years sys. verif. experience
Modular code and proofs
Near-instant feedback
Verus
Sound back-end orchestration
Custom proof library support
LEAN THEOREM PROVER
Continuous analysis/optimization
SMT solver
BV solver
Crypto solver
Telemetry
Proactive proof stabilization
Source-level feedback

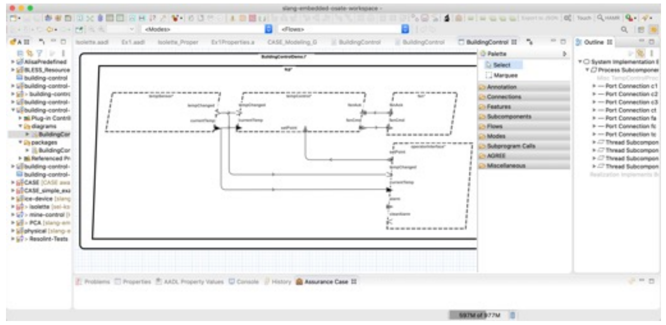
```
use builtin::*;

veru  failed this postcondition rustc
      main.rs(12, 1): original diagnostic
fn m
  en  No quick fixes available
    m <= x,
    m <= y,
    m == x || m == y,
{
    if x <= y {
        y
    } else {
        y
    }
}

} // verus!
```

Collins Aerospace

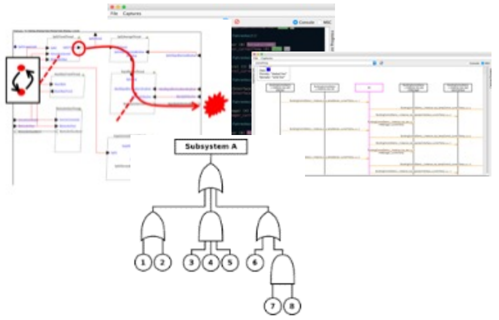
KSU : HAMR

HAMR – tool chain for [H]igh [A]ssurance [M]odeling and [R]apid engineering for embedded systems (developed by Kansas State University and Galois)

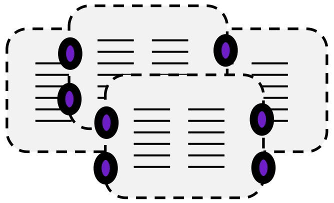
Modeling, analysis, and verification in the **AADL** modeling language



Leveraging analyses from AADL community



Component development and verification in multiple languages



- C
- Slang (developed at Kansas State)
  - high integrity subset of Scala
  - contract verification framework
  - translates to C

Deployments aligned with AADL run-time on multiple platforms

JVM Deployment

Linux Deployment

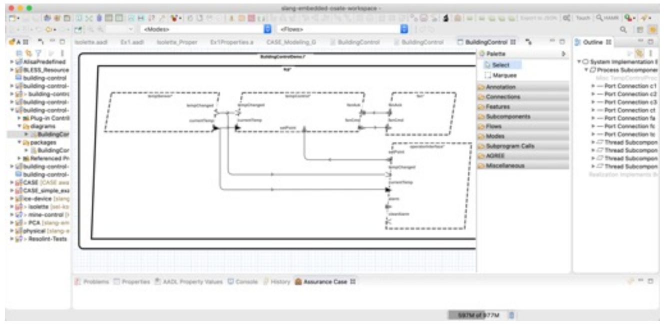
seL4 Deployment

verified micro-kernel

KSU : HAMR

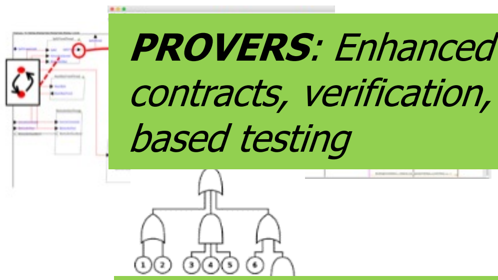
HAMR – tool chain for [H]igh [A]ssurance [M]odeling and [R]apid engineering for embedded systems (developed by Kansas State University and Galois)

Modeling, analysis, and verification in the **AADL** modeling language



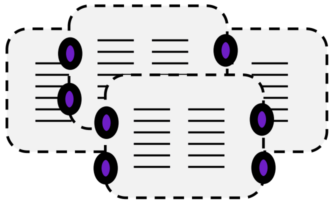
Levera
commu

***PROVERS**: Add SysMLv2 prototype*



***PROVERS**: Enhanced support for contracts, verification, property-based testing*

Component development and verification in multiple languages



- C
- Slang (developed
  - high integrit
  - contract verification framework
  - translates to C

***PROVERS**: Add code- and contract-generation, and property-based testing for Rust*

Deployments aligned with AADL run-time on multiple platforms

JVM Deployment

JVM (Java Virtual Machine)

Linux Deployment

Linux

seL4 Deploymen

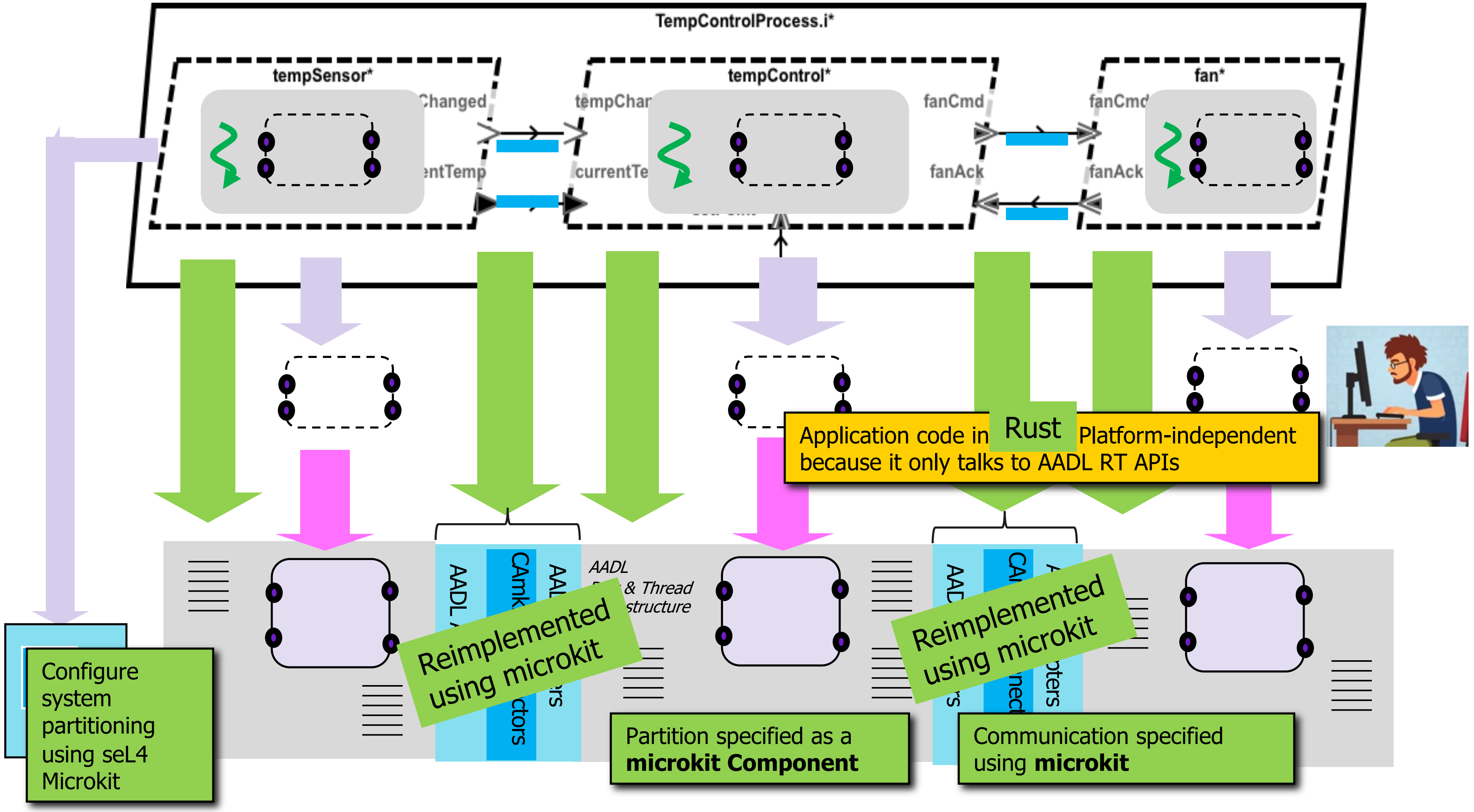
***PROVERS**: Retarget to seL4 micro-kit (Core Platform)*

seL4 Security. Performance. Proof.

verified micro-kernel

HAMR Code Generation

HAMR instantiation for Rust based development on **seL4 microkernel** using seL4 microkit



Application code in Rust Platform-independent because it only talks to AADL RT APIs

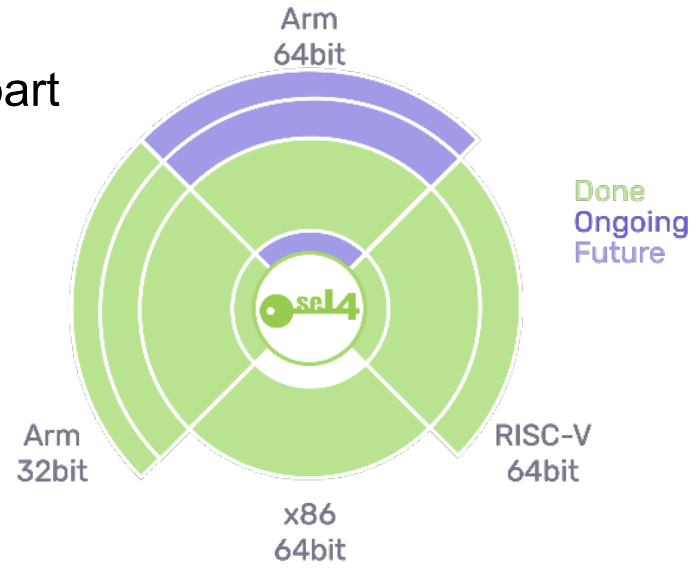
Configure system partitioning using seL4 Microkit

Partition specified as a **microkit Component**

Communication specified using **microkit**

PROOFCRAFT : SEL4 KERNEL PROOF GENERALIZATION AND REPAIR

- Goal: make seL4 proofs less dependent on experts for maintenance and extensions
  - Automated Verification for Platform Ports
    - → Proof parametrization, proved once against sufficient conditions
    - → For each new platform: automatic extraction of configuration parameters & check against conditions
  - More Agile and Generic Proofs
    - → Split generic architecture-independent part from architecture-dependent part
    - → Extend verification to latest major feature: MCS seL4
- Impact: scalable access to formal methods
    - → Reduced cost and reliance on experts for maintenance and extensions
    - → Increased assurance robustness against anticipated change
    - → Increased features for verified foundation

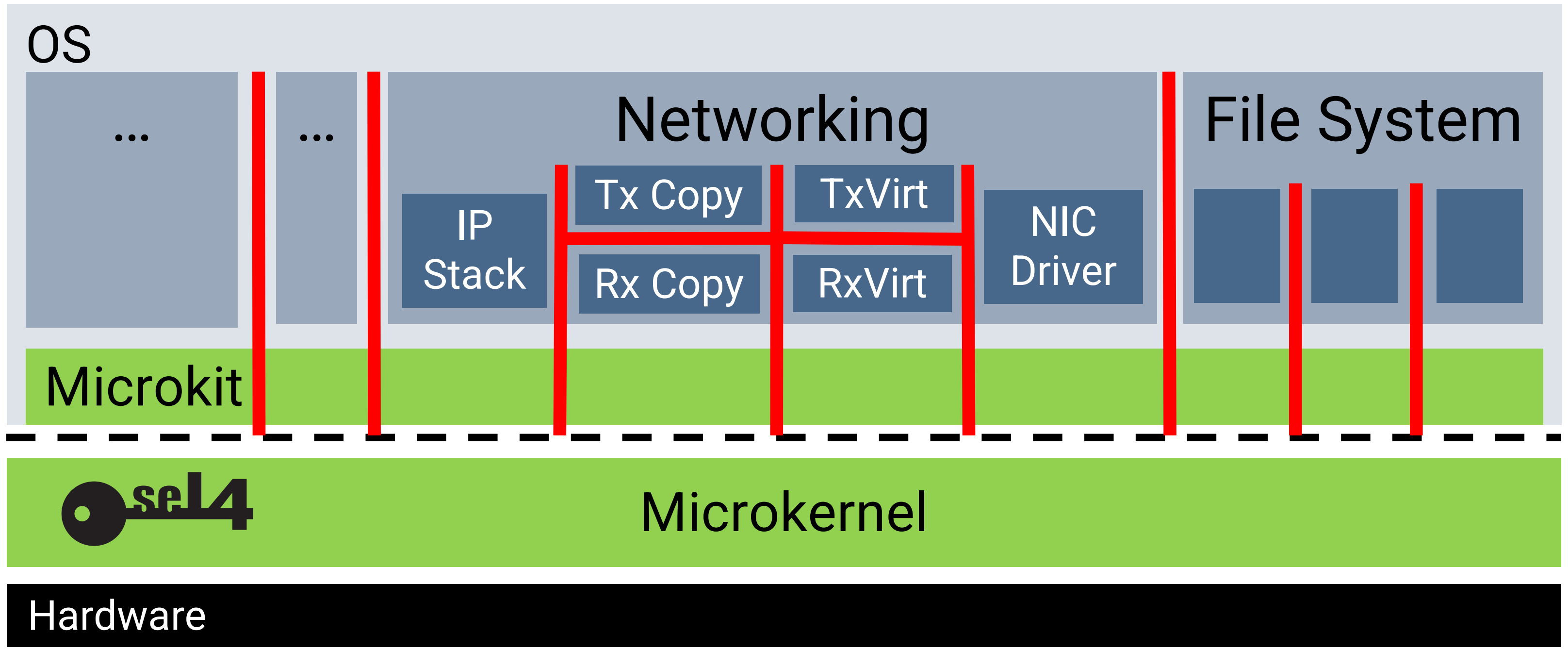


seL4 verified on more platforms, with more features, for less cost and less expertise

Collins Aerospace

UNSW SYDNEY : LIONS OPERATING SYSTEM & COMPONENT VERIFICATION

- Lions OS: new seL4-based OS developed from scratch at UNSW
- Highly-componentized, verification-friendly, yet high performance
- Simplicity & adaptability by use-case-specific, swappable policies



Collins Aerospace

KANSAS (KU) : AI-BASED PROOF REPAIR AND COMPONENT SYNTHESIS

- ML-Enhanced Proof Repair
  - Maintain evidence over design, requirements and environmental changes
  - Update and replay proofs, retake measurements, replay testing
- Evidence Protocols
  - Update and generalize Copland attestation protocols for general-purpose evidence gathering
  - Develop canonical techniques for parametric adaptation, refinement and abstraction, protocol synthesis
  - Reuse MAESTRO attestation environment for general evidence gathering
- Verified Synthesis
  - Enable working at the requirements level
  - Synthesis of Rust from requirements language retargeting Coq to CakeML synthesis

SUMMARY

- Workflow and tools address the entire software development stack
- Building upon the best available technologies and leveraging our experience from recent research programs as a starting point
- Integrate new formal methods tools with Collins workflow automation processes
- Applied to ALE mission computer to demonstrate usability, practicality, and effectiveness
- Formal verification will be made accessible to non-formal methods experts through automated analysis with streamlined user feedback and generalized proofs that are robust to changes, augmented by automated repair tools

END

Collins Aerospace