SAE AEROSPACE SYSTEMS AND TECHNOLOGY CONFERENCE

Formal, Architecture-Driven Assurance with AADL and Trusted Build

Mike Whalen
University of Minnesota

Outline

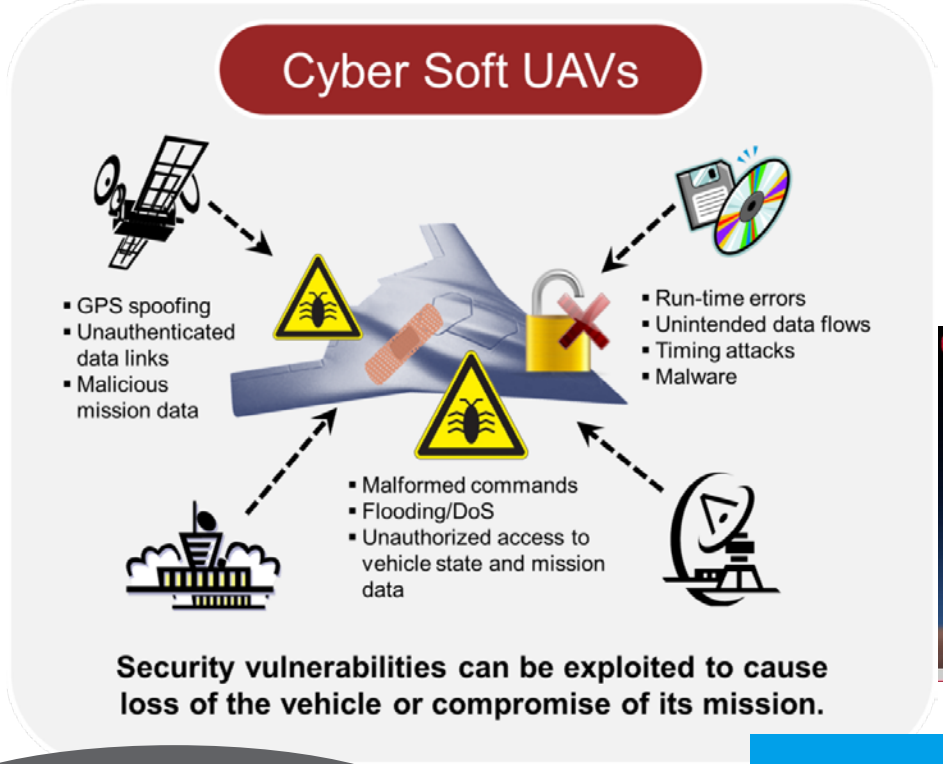
Motivation

Technologies

Results

Motivation: hacks on embedded systems

Aircraft may be susceptible to cyber attack



Security vulnerabilities that can lead to safety hazards

Growing awareness among customers, regulators, and public

Secure Mathematically Assured Composition of Control Models (SMACCM)

Cyber Hard UAVs

Secure Analyzable Architecture

- Proof tools for verification of information flow, timing, and safety
- Ensure correctness of system architecture properties

Secure Software Components

- Embedded code synthesized from specifications – no C run-time errors
- Run-time monitoring shuts down suspicious activity

Secure Operating Systems

- Formally verified operating systems
- Provide robust infrastructure and separation of critical/non-critical functions

HACMS Red Team

"No unexpected behaviors or crashes were possible via exposed interfaces"

In Phase 1 of the SMACCM project we designed, built, and performed flight demonstrations of a Cyber Hard UAV.

SMACCM

High Assurance Cyber Military Systems (HACMS)

HACMS Program Architecture

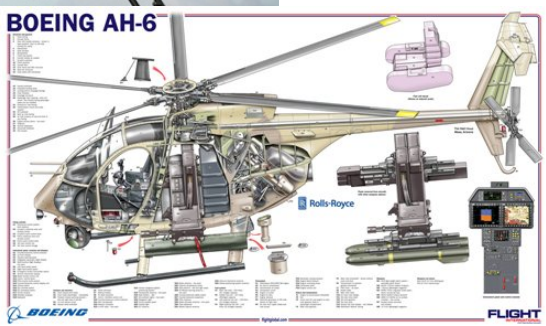
	1. Military Vehicle Experts	2. Operating Systems	3. Control Systems	4. Research Integration	5. Red Team
Air	**Boeing** Pilot-able Unmanned Little Bird Helicopter	**NICTA** Synthesize OS components; Verified sel4 kernel; Verified RTOS	**Galois** Embedded DSL; Synthesize and verify control system code	**RC*/U. Minn** Compositional verification; Integrated workbench	**DRAPER*/AIS/ U. Oxford** Traditional penetration testing; novel formal methods approach
Ground	**HRL*/GM** Cadillac SRX	**SRI*/UIUC** EF-SMT solvers; Synthesize monitors and wrappers	**SRI*** Synthetic sensors; Synthesis for controllers of hybrid systems	**SRI*** Evidential Tool Bus; Kernel of Truth; Lazy Composition; Vehicle Integration	
	 Army TARDEC	**Princeton*/Yale/MIT** Build and verify in CoQ vehicle-control OS; compCERT: concurrent; compositional integration of verified components	**CMU*/Drexel/ SpiralGen/UIUC** Map high-level spec into low-level C code; Extend Spiral for hybrid systems		
		Kestrel* Synthesize protocols: refinement of high-level spec to low-level implementations	**UPenn*/UCLA** Synthesize attack-resilient control systems		

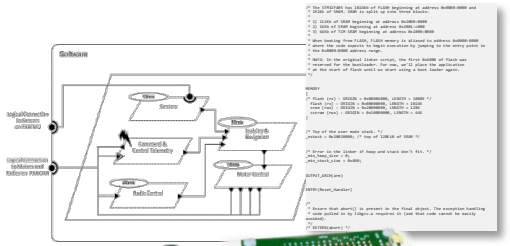
- Three 18-month phases
- Red Team assessment at the end of each phase

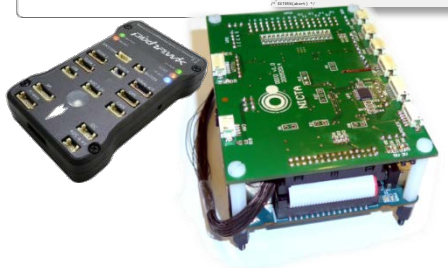
Air Team Platforms

3DR IRIS+ quadcopter
(SMACCMcopter)

Boeing Unmanned
Little Bird (ULB)
AH-6 derivative







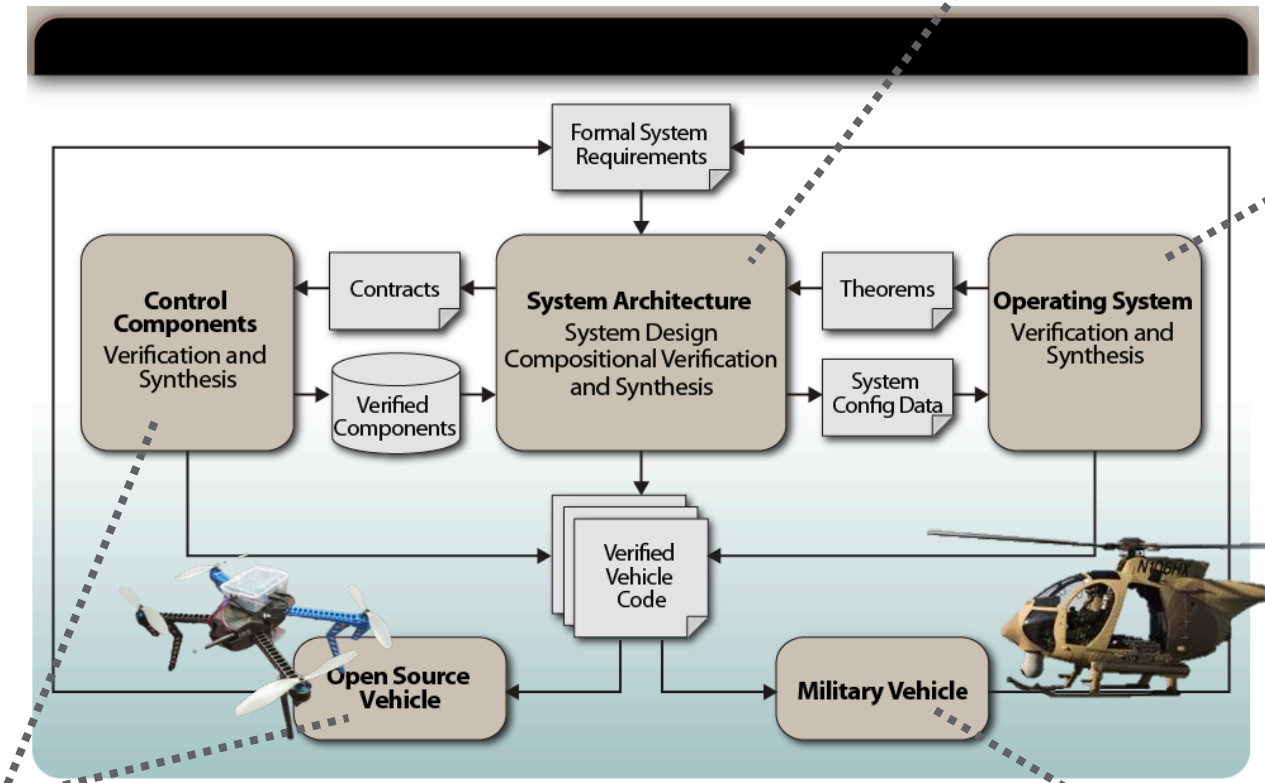
New electronics to host
provably secure software

ULB Phase 2 Flight Demo – 24 July, Mesa

Architecture-Driven Assurance

Rockwell Collins

University of Minnesota



DATA 61

galois

HACMS_002_SWEng_12

BOEING

Why focus on system architecture?

Scalability

Reasoning about large systems through (de)composition

Security

Understand information flows

Interfaces are sources of vulnerabilities

Confidence

Structure for reasoning about system

Transition

System-level modeling and verification is immature

Excellent technology transition target

Can we trust the architecture model?

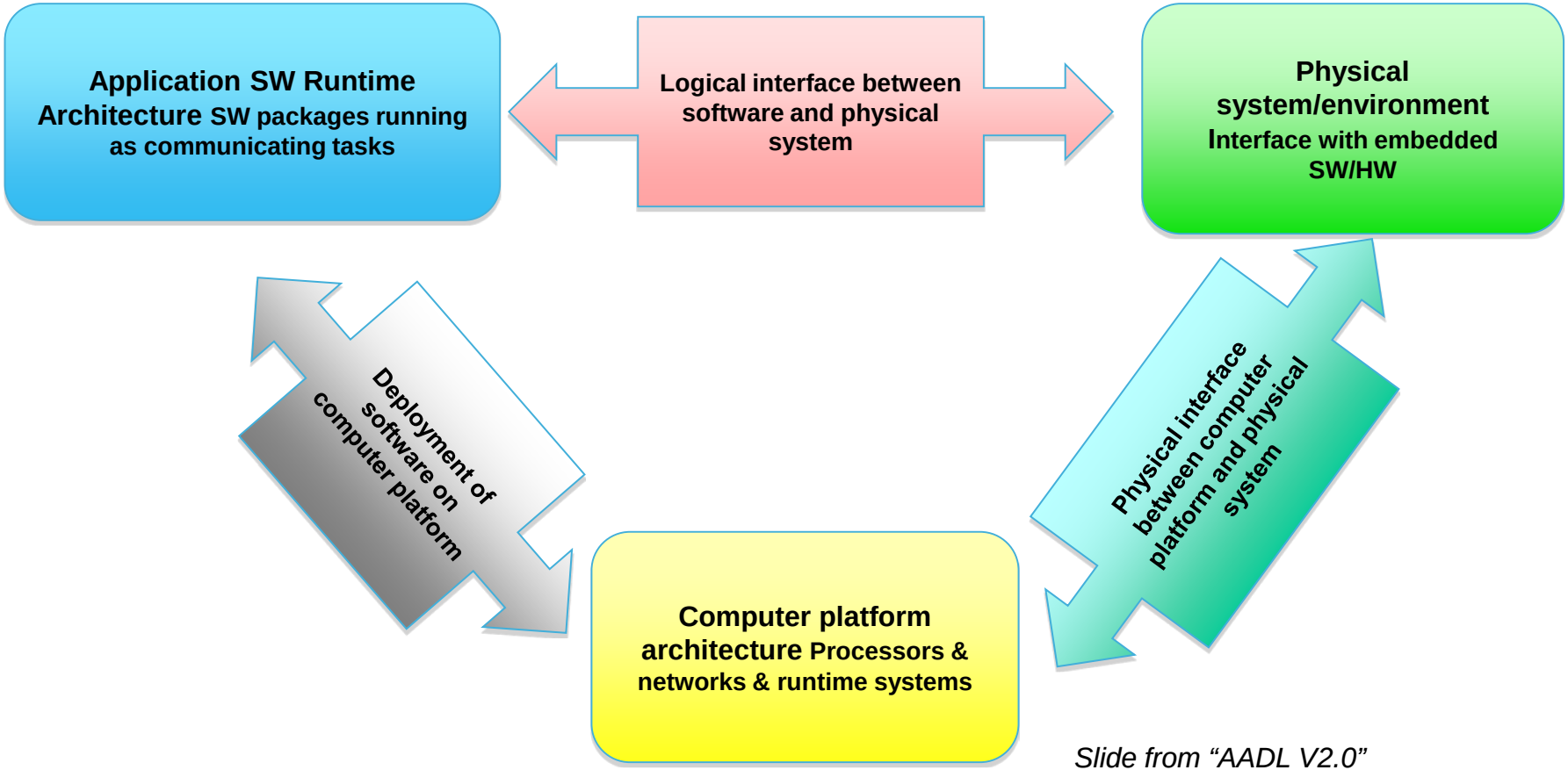
Architecture model is correct
- Properties, structure, behavior, interaction of components, interfaces, contracts
- Analyzable

Components are correct
- Realizable contracts
- Components verified to implement contracts

System does what the model says
- No other information flows (memory safety / isolation)
- OS executes model correctly

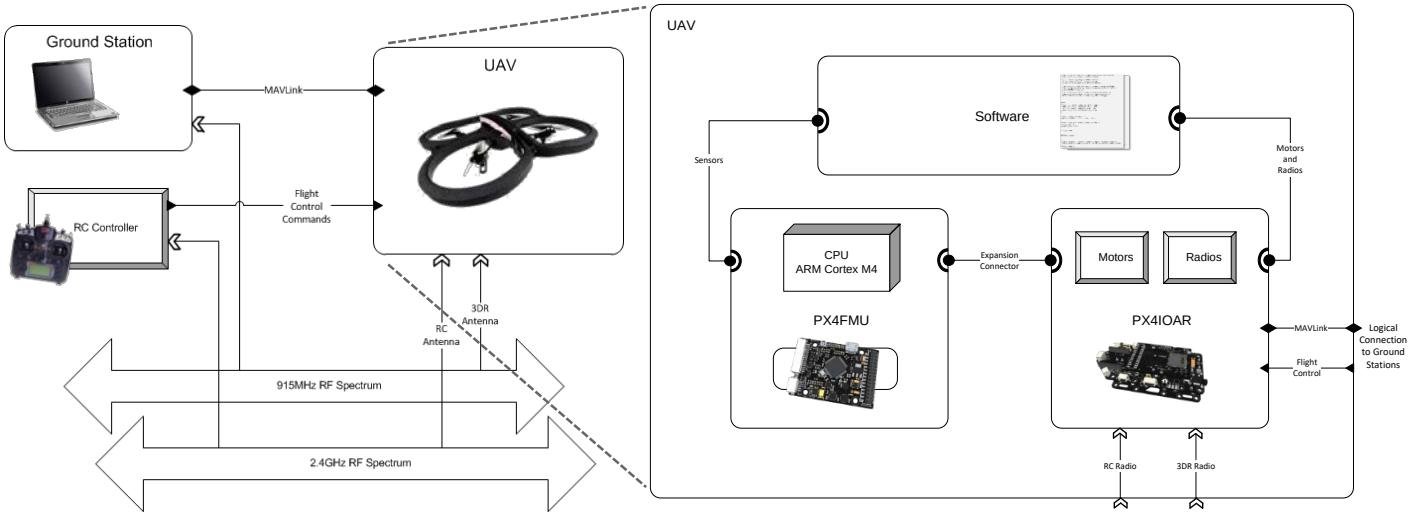
System implementation corresponds to model
- Automatic build from component and architecture models

Embedded Software System Architecture



Slide from "AADL V2.0" presentation

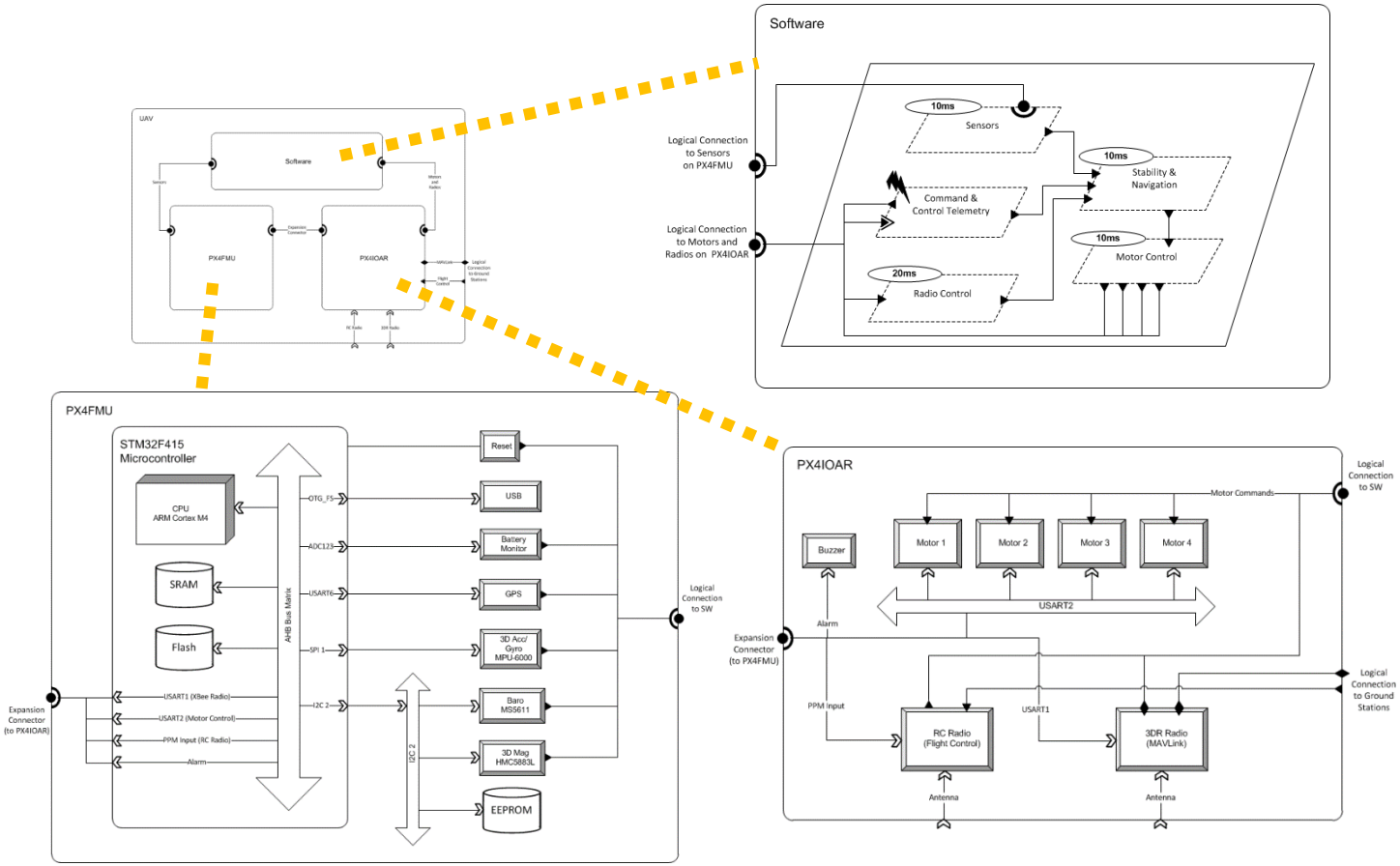
Architecture Analysis and Design Language (AADL)



AADL = SAE AS5506 standard

- Target: Embedded, real-time, distributed systems
- Describes both hardware and software
- Extensible syntax
- Open source tools, supported by SEI

AADL Model of Phase 1 SMACCMcopter



Vulnerabilities: Baseline Assessment

Research Vehicle (quadcopter)

- **3DR Radios have no security; injection and sniffing are trivial**
- **APM (Ground Station) Mission Planner DoS**
- **3DR firmware retrieved from unsecure server by Mission Planner**
- **3DR radios allow remote reboot into firmware update mode**
- **MavLink channel operates near saturation, trivial to overload channel causing effects on Mission Planner**
- **MavLink protocol allows read/write of internal APM memory**

Unmanned Little Bird

- **L3 Mini-TCDL multicast network routing failure**
- **ESR-904 Ethernet to serial converter crash**
- **STANAG 4586 message injection**
- **VSM status message saturation can cause link saturation and failure**
- **Unauthenticated control of Wescam EO/IR payload**
- **VSM waypoint processing DoS**
- **L3 Mini-TCDL communications can be hijacked**

Out of scope for HACMS

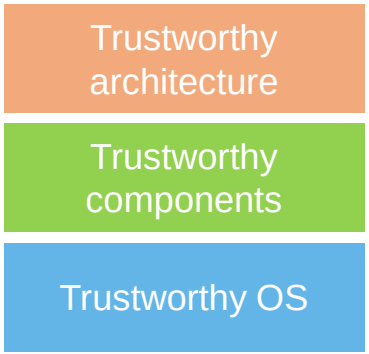
Secure comms

Fixed component

System design

Other Possible Vulnerabilities

- **Secure dataflows? (can authentication be bypassed?)**
- **Memory safety? (unintended flows)**
- **C runtime errors? (crash)**
- **Task blocking? (CPU utilization)**
- **Response to DoS attacks?**



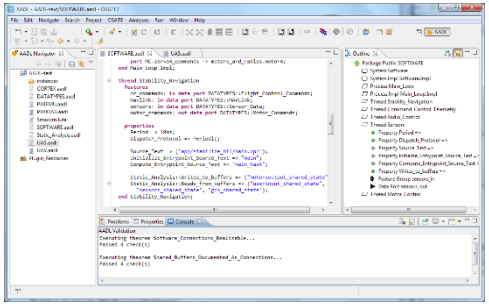
Weakness/Attack Sources:

Common Weakness Enumeration, http://cwe.mitre.org

Common Attack Pattern Enumeration and Classification, http://capec.mitre.org

SMACCM milestone 6: Security requirements document

AADL Tools: Formal Methods Workbench



OSATE

Trusted Build

Resolute

Assurance Case

AGREE

Behavioral Analysis

Lute

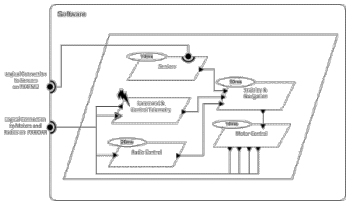
Structural Analysis

Architecture Models

Architecture Translation

Architecture Analysis

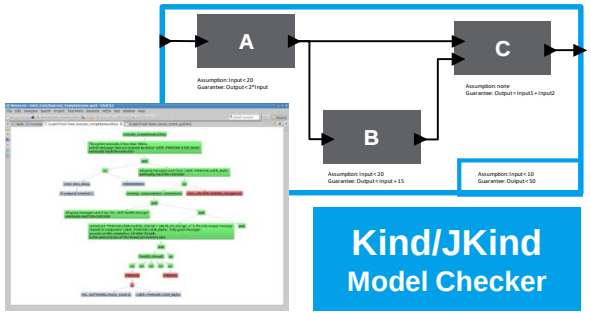
AADL



seL4

eChronos

VxWorks



Kind/JKind

Model Checker

Assurance Cases

A reasoned and compelling argument, supported by a body of evidence, that a system, service or organization will operate as intended for a defined application in a defined environment

GSN community standard V1

A graphical representation of an argument supported by evidence

May address different system aspects

- Safety
- Security
- Correctness

Evidence about the model

How can we make high-level claims about correctness?

Combine heterogeneous evidence from multiple sources

Galois: IVORY DSL

NICTA: Sel4 Microkernel

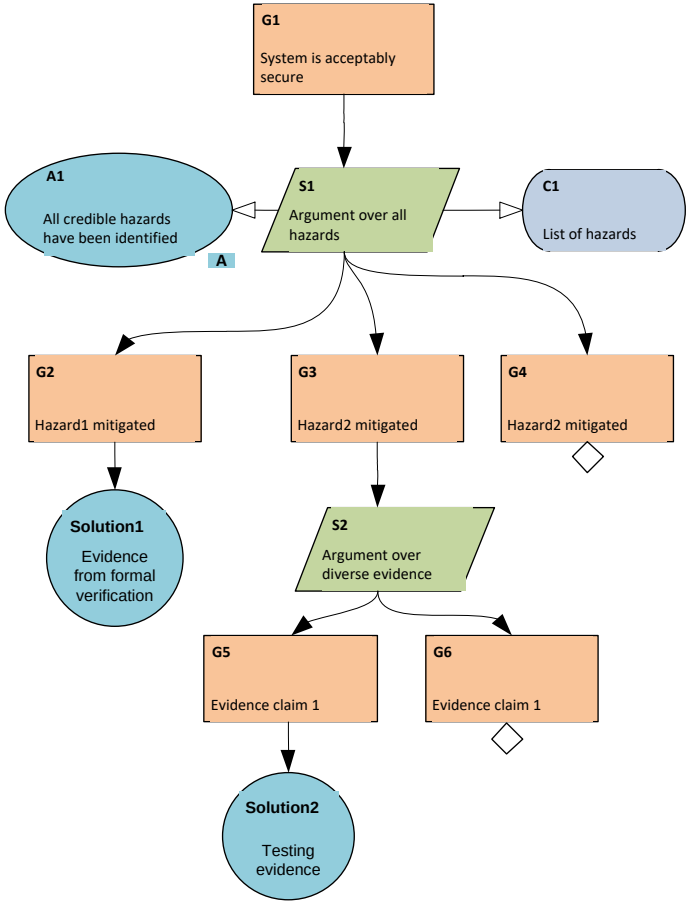
Boeing: Internal Processes and Best Practices

Generate system image from the model

Assurance Cases

Goal Structuring Notation (GSN) is used in several tools

- *Goals*: claims about the system
- *Strategy*: argues why a goal is true
- *Assumptions*
- *Solution*: leaf level evidence



Assurance Cases

Positives

- Informal
- Can include many different sources of evidence
- Understandable by domain experts
- Captures structure of argument

Negatives

- Informal
- Not strongly tied to the system design
- Semantics are loose (English is ambiguous)

Assurance Cases

Negatives

Informal

Not strongly tied to the system design

Semantics are loose (English is ambiguous)

Resolute: An Assurance Case Language for Architecture Models

Use a logic to generate an assurance case

Make the structure of the system architecture help form the structure of the assurance case

Use an architectural design language with defined semantics (AADL)

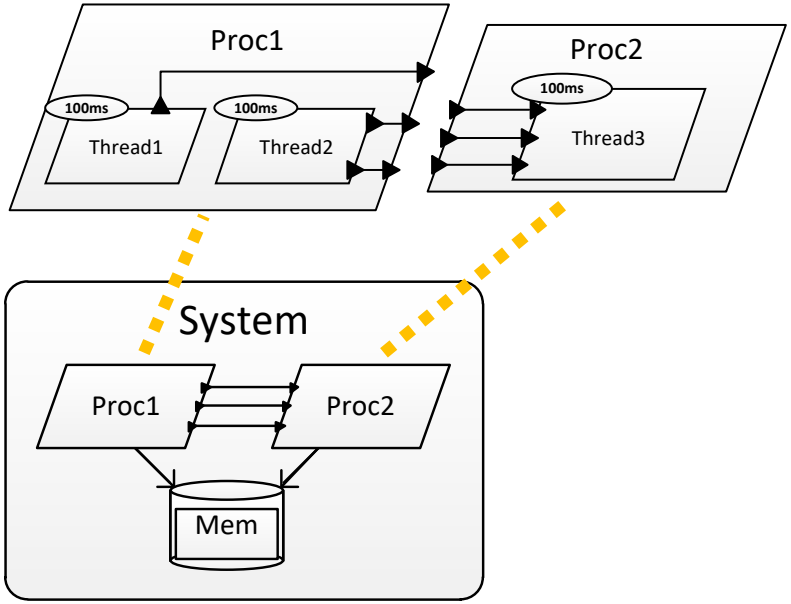
Resolute Language

Claims and rules for satisfying those claims

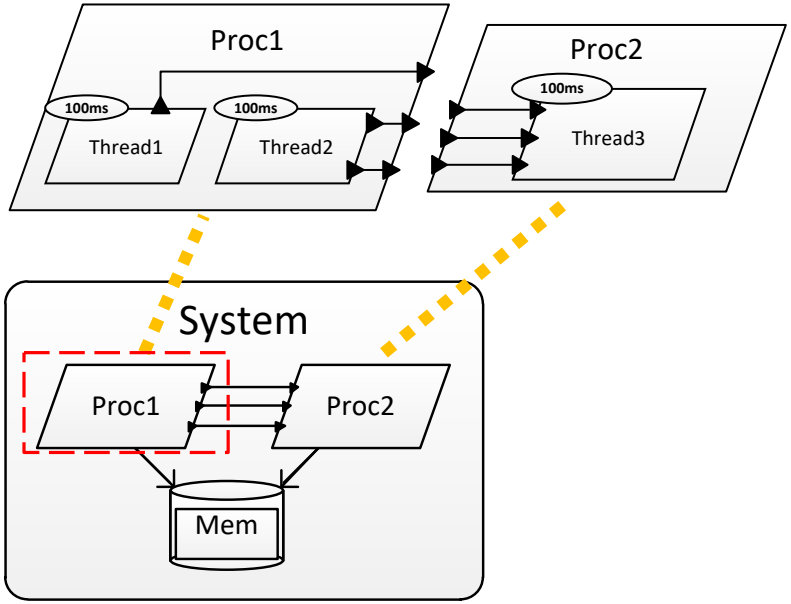
Rules and claims parameterized by AADL types

Assurance cases instantiated with elements from AADL model

```
memory_protected(p : process) <=
  ** "The memory of process " p " is protected from alterations by other processes" **
  property(p, SMACCM::OS) = "SeL4" or
  (property(p, SMACCM::OS) = "eChronos" and
   forall (mem : memory). bound(p, mem) =>
     forall (q : process). bound(q, mem) => memory_safe_process(q))
```

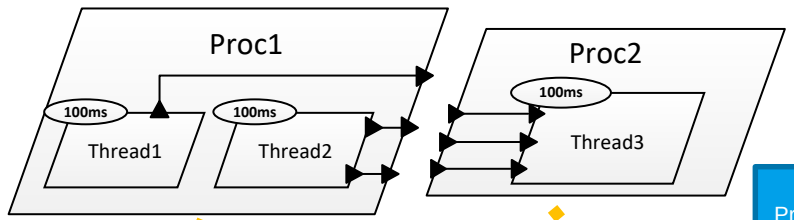



```
memory_protected(p : process) <=
  ** "The memory of process " p " is protected
from alterations by other processes" **
  property(p, SMACCM::OS) = "SeL4" or
  (property(p, SMACCM::OS) = "eChronos" and
   forall (mem : memory). bound(p, mem) =>
     forall (q : process). bound(q, mem) => memory_safe_process(q))

memory_safe_process(p : process) <=
  ** "The process " p " only writes to its own memory space" **
  forall (t : thread). contained(t, p) => memory_safe_thread(t)

memory_safe_thread(t : thread) <=
  ** "The thread " t " only writes to its own memory space" **
  ivory_thread(t)

ivory_thread(t : thread) <=
  ** "The thread " t " is generated from Ivory" **
  property(t, SMACCM::Language) = "Ivory"
```



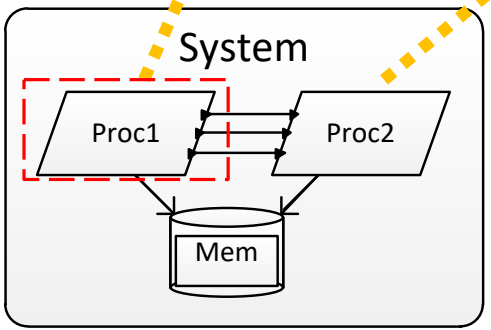
Process “System.Proc1” protected from other processes

```
memory_protected(p : process) <=
  ** "The memory of process " p " is protected
from alterations by other processes" **
   property(p, SMACCM::OS) = "SeL4" or
   (property(p, SMACCM::OS) = "eChronos" and
    forall (mem : memory). bound(p, mem) =>
      forall (q : process). bound(q, mem) => memory_safe_process(q))

memory_safe_process(p : process) <=
   ** "The process " p " only writes to its own memory space" **
   forall (t : thread). contained(t, p) => memory_safe_thread(t)

memory_safe_thread(t : thread) <=
   ** "The thread " t " only writes to its own memory space" **
   ivory_thread(t)

ivory_thread(t : thread) <=
   ** "The thread " t " is generated from Ivory" **
   property(t, SMACCM::Language) = "Ivory"
```

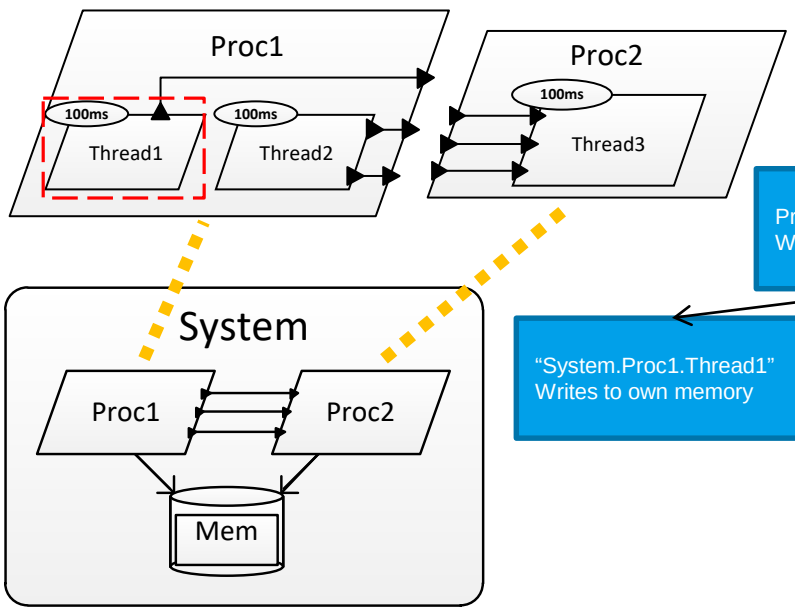


Process “System.Proc1” protected from other processes

Process “System.Proc1” Writes to own memory




```
memory_protected(p : process) <=
  ** "The memory of process " p " is protected
from alterations by other processes" **
  property(p, SMACCM::OS) = "SeL4" or
  (property(p, SMACCM::OS) = "eChronos" and
   forall (mem : memory). bound(p, mem) =>
     forall (q : process). bound(q, mem) => memory_safe_process(q))

memory_safe_process(p : process) <=
  ** "The process " p " only writes to its own memory space" **
  forall (t : thread). contained(t, p) => memory_safe_thread(t)

memory_safe_thread(t : thread) <=
  ** "The thread " t " only writes to its own memory space" **
  ivory_thread(t)

ivory_thread(t : thread) <=
  ** "The thread " t " is generated from Ivory" **
  property(t, SMACCM::Language) = "Ivory"
```

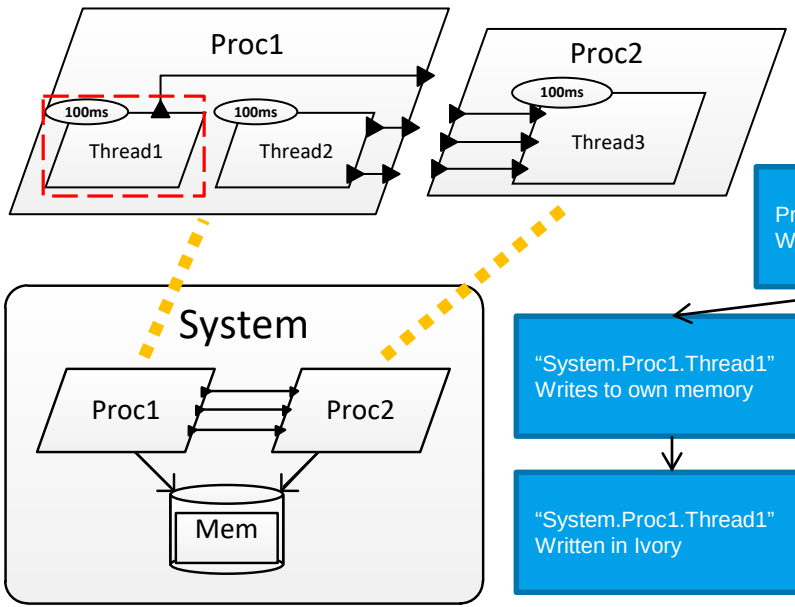


Process “System.Proc1” protected from other processes

Process “System.Proc1” Writes to own memory

“System.Proc1.Thread1” Writes to own memory

```
memory_protected(p : process) <=
  ** "The memory of process " p " is protected
from alterations by other processes" **
  property(p, SMACCM::OS) = "SeL4" or
  (property(p, SMACCM::OS) = "eChronos" and
   forall (mem : memory). bound(p, mem) =>
     forall (q : process). bound(q, mem) => memory_safe_process(q))

memory_safe_process(p : process) <=
  ** "The process " p " only writes to its own memory space" **
  forall (t : thread). contained(t, p) => memory_safe_thread(t)

memory_safe_thread(t : thread) <=
  ** "The thread " t " only writes to its own memory space" **
  ivory_thread(t)

ivory_thread(t : thread) <=
  ** "The thread " t " is generated from Ivory" **
  property(t, SMACCM::Language) = "Ivory"
```



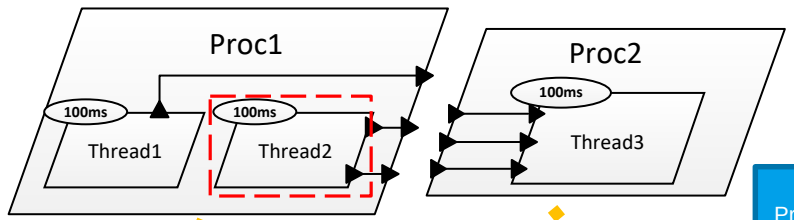
Process “System.Proc1” protected from other processes

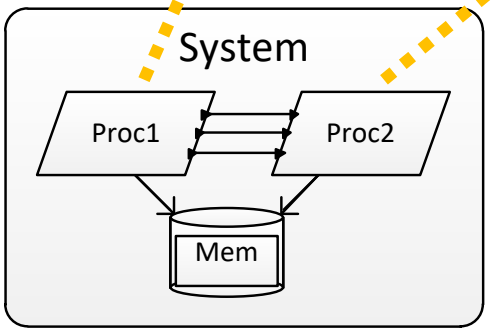
Process “System.Proc1” Writes to own memory

“System.Proc1.Thread1” Writes to own memory

“System.Proc1.Thread1” Written in Ivory

```
memory_protected(p : process) <=
  ** "The memory of process " p " is protected
from alterations by other processes" **
  property(p, SMACCM::OS) = "SeL4" or
  (property(p, SMACCM::OS) = "eChronos" and
   forall (mem : memory). bound(p, mem) =>
     forall (q : process). bound(q, mem) => memory_safe_process(q))

memory_safe_process(p : process) <=
  ** "The process " p " only writes to its own memory space" **
  forall (t : thread). contained(t, p) => memory_safe_thread(t)

memory_safe_thread(t : thread) <=
  ** "The thread " t " only writes to its own memory space" **
  ivory_thread(t)

ivory_thread(t : thread) <=
  ** "The thread " t " is generated from Ivory" **
  property(t, SMACCM::Language) = "Ivory"
```





Process “System.Proc1” protected from other processes

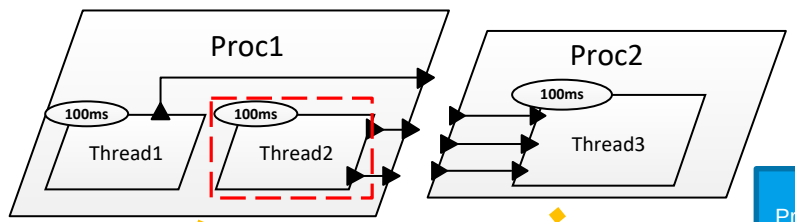
Process “System.Proc1” Writes to own memory

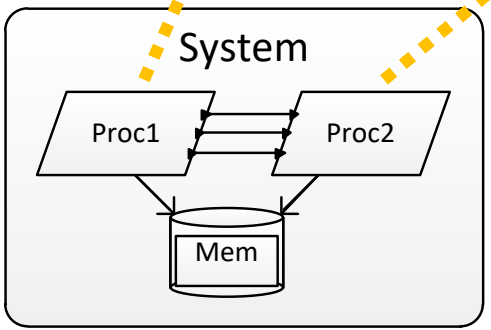
“System.Proc1.Thread1” Writes to own memory

“System.Proc1.Thread2” Writes to own memory

“System.Proc1.Thread1” Written in Ivory

```
memory_protected(p : process) <=
  ** "The memory of process " p " is protected
from alterations by other processes" **
  property(p, SMACCM::OS) = "SeL4" or
  (property(p, SMACCM::OS) = "eChronos" and
   forall (mem : memory). bound(p, mem) =>
     forall (q : process). bound(q, mem) => memory_safe_process(q))

memory_safe_process(p : process) <=
  ** "The process " p " only writes to its own memory space" **
  forall (t : thread). contained(t, p) => memory_safe_thread(t)

memory_safe_thread(t : thread) <=
  ** "The thread " t " only writes to its own memory space" **
  ivory_thread(t)

ivory_thread(t : thread) <=
  ** "The thread " t " is generated from Ivory" **
  property(t, SMACCM::Language) = "Ivory"
```





Process “System.Proc1” protected from other processes

Process “System.Proc1” Writes to own memory

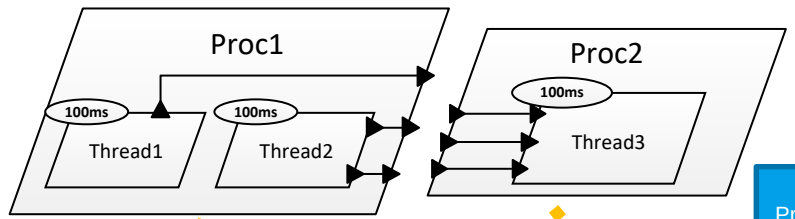
“System.Proc1.Thread1” Writes to own memory

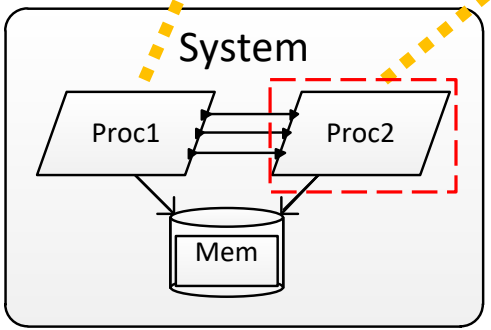
“System.Proc1.Thread2” Writes to own memory

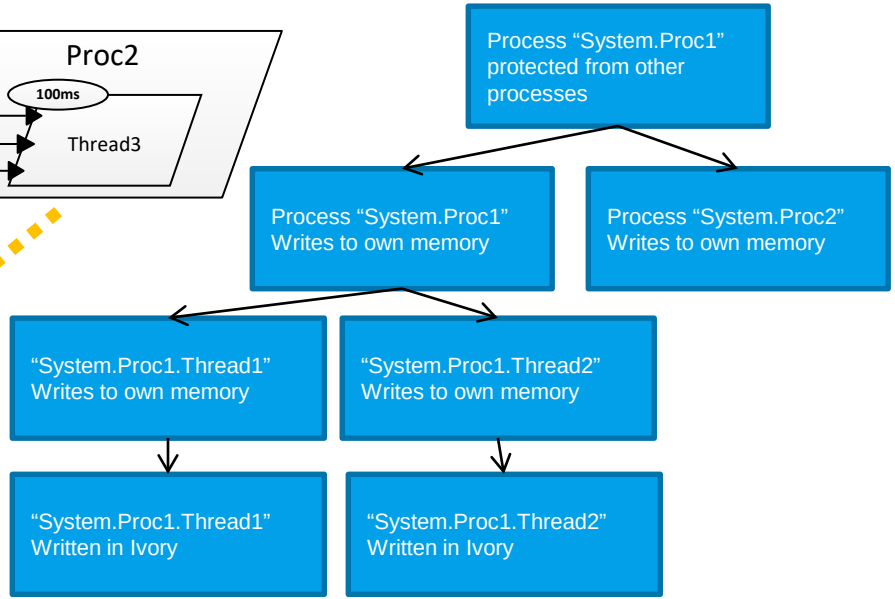
“System.Proc1.Thread1” Written in Ivory

“System.Proc1.Thread2” Written in Ivory

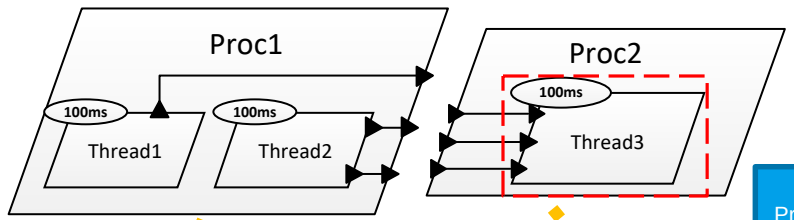
```
memory_protected(p : process) <=
  ** "The memory of process " p " is protected
from alterations by other processes" **
  property(p, SMACCM::OS) = "SeL4" or
  (property(p, SMACCM::OS) = "eChronos" and
   forall (mem : memory). bound(p, mem) =>
     forall (q : process). bound(q, mem) => memory_safe_process(q))

memory_safe_process(p : process) <=
  ** "The process " p " only writes to its own memory space" **
  forall (t : thread). contained(t, p) => memory_safe_thread(t)

memory_safe_thread(t : thread) <=
  ** "The thread " t " only writes to its own memory space" **
  ivory_thread(t)

ivory_thread(t : thread) <=
  ** "The thread " t " is generated from Ivory" **
  property(t, SMACCM::Language) = "Ivory"
```

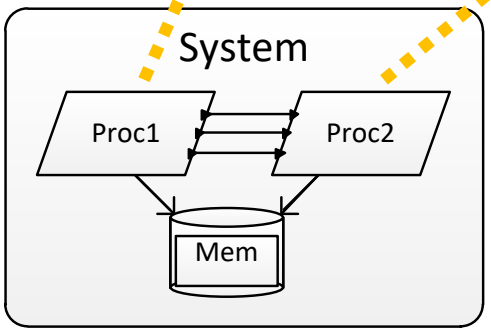




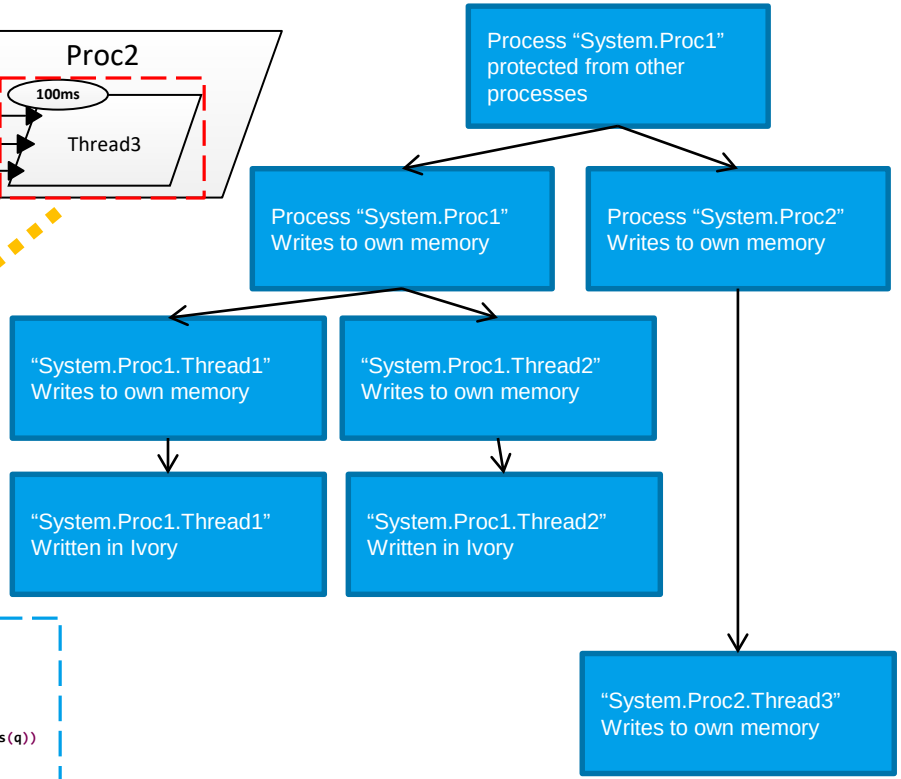


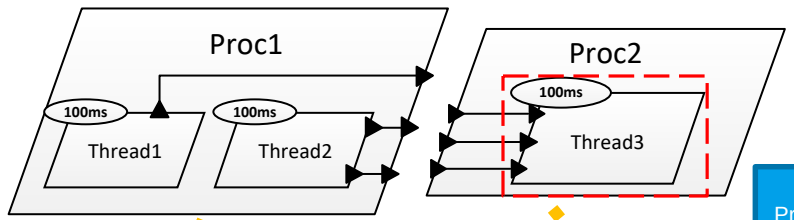

```
memory_protected(p : process) <=
  ** "The memory of process " p " is protected
from alterations by other processes" **
  property(p, SMACCM::OS) = "SeL4" or
  (property(p, SMACCM::OS) = "eChronos" and
   forall (mem : memory). bound(p, mem) =>
     forall (q : process). bound(q, mem) => memory_safe_process(q))

memory_safe_process(p : process) <=
  ** "The process " p " only writes to its own memory space" **
  forall (t : thread). contained(t, p) => memory_safe_thread(t)

memory_safe_thread(t : thread) <=
  ** "The thread " t " only writes to its own memory space" **
  ivory_thread(t)

ivory_thread(t : thread) <=
  ** "The thread " t " is generated from Ivory" **
  property(t, SMACCM::Language) = "Ivory"
```

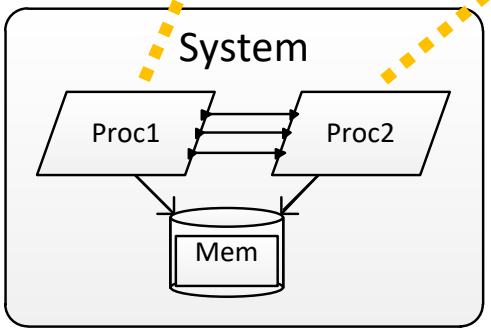




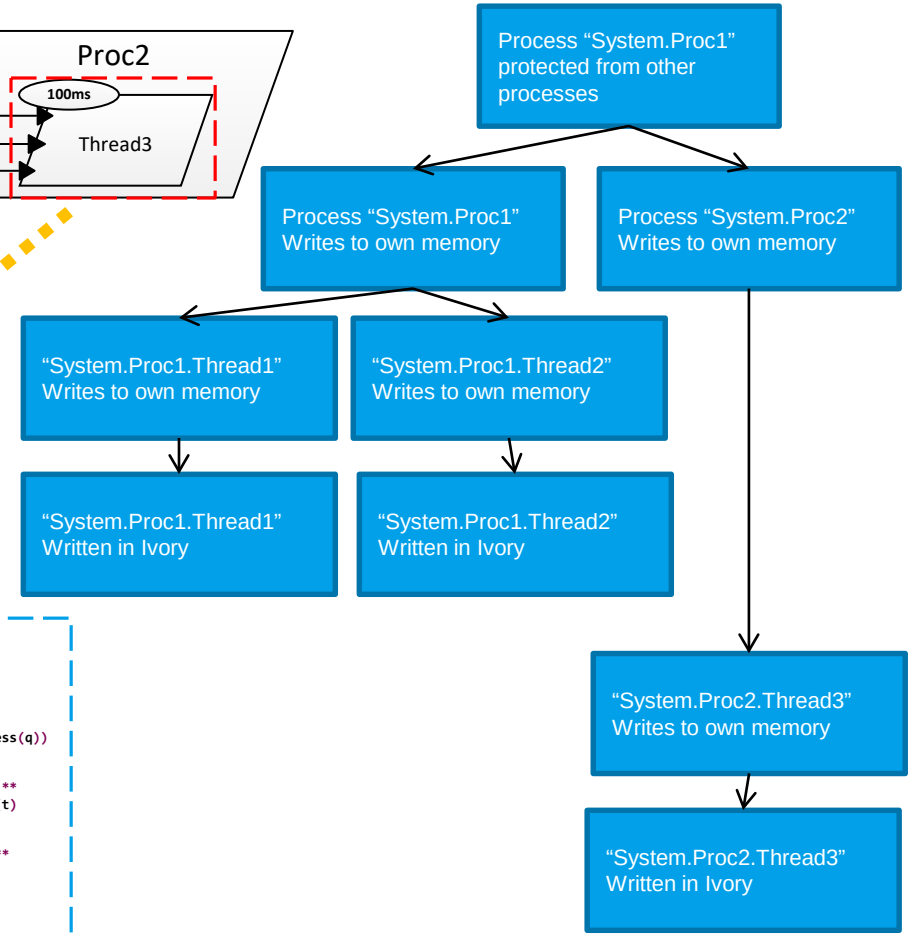

```
memory_protected(p : process) <=
   ** "The memory of process " p " is protected
from alterations by other processes" **
   property(p, SMACCM::OS) = "SeL4" or
   (property(p, SMACCM::OS) = "eChronos" and
    forall (mem : memory). bound(p, mem) =>
      forall (q : process). bound(q, mem) => memory_safe_process(q))

memory_safe_process(p : process) <=
   ** "The process " p " only writes to its own memory space" **
   forall (t : thread). contained(t, p) => memory_safe_thread(t)

memory_safe_thread(t : thread) <=
   ** "The thread " t " only writes to its own memory space" **
   ivory_thread(t)

ivory_thread(t : thread) <=
   ** "The thread " t " is generated from Ivory" **
   property(t, SMACCM::Language) = "Ivory"
```








```
memory_protected(p : process) <=
   ** "The memory of process " p " is protected
from alterations by other processes" **
   property(p, SMACCM::OS) = "SeL4" or
   (property(p, SMACCM::OS) = "eChronos" and
    forall (mem : memory). bound(p, mem) =>
      forall (q : process). bound(q, mem) => memory_safe_process(q))

memory_safe_process(p : process) <=
   ** "The process " p " only writes to its own memory space" **
   forall (t : thread). contained(t, p) => memory_safe_thread(t)

memory_safe_thread(t : thread) <=
   ** "The thread " t " only writes to its own memory space" **
   ivory_thread(t)

ivory_thread(t : thread) <=
   ** "The thread " t " is generated from Ivory" **
   property(t, SMACCM::Language) = "Ivory"
```

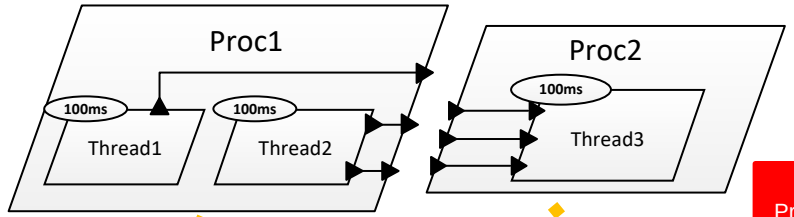


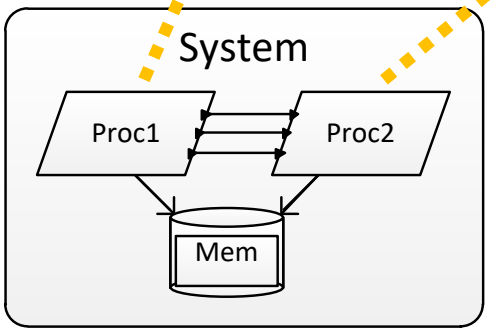
Failed Assurance Cases

Unlike traditional assurance cases, Resolute can produce a failed assurance case

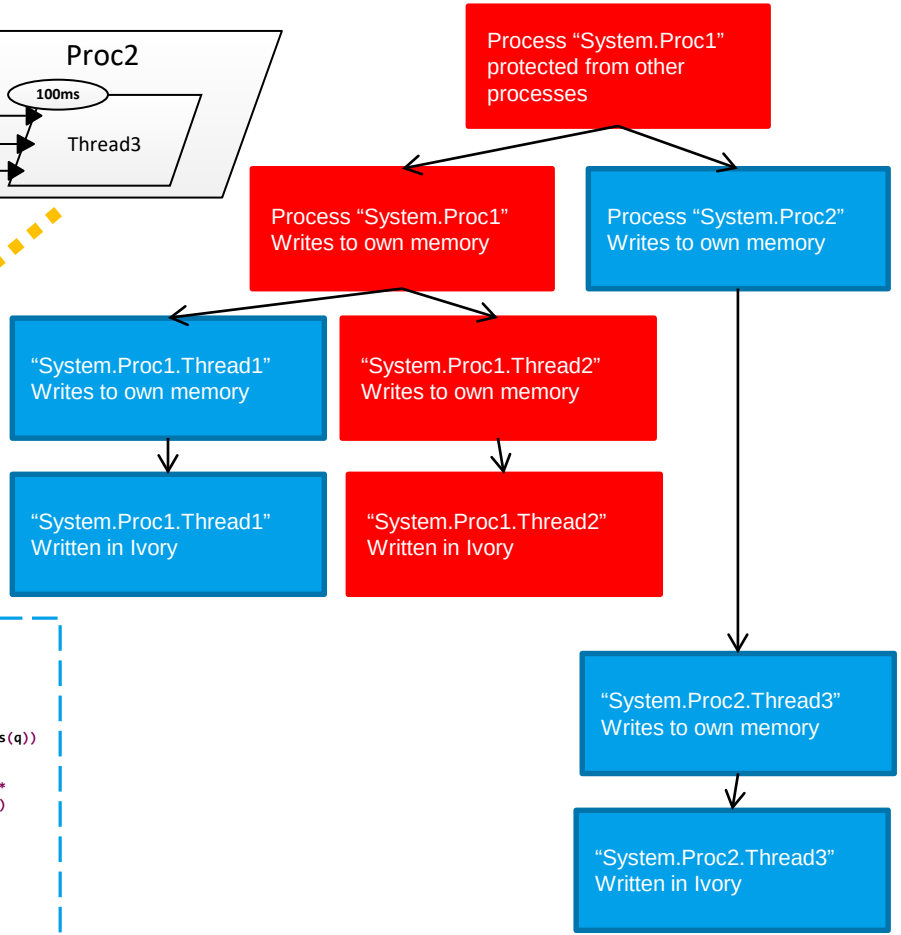
Claims that are false are shown in red so the assurance case can be debugged

Failures may occur if the architecture changes, or if external analysis fails

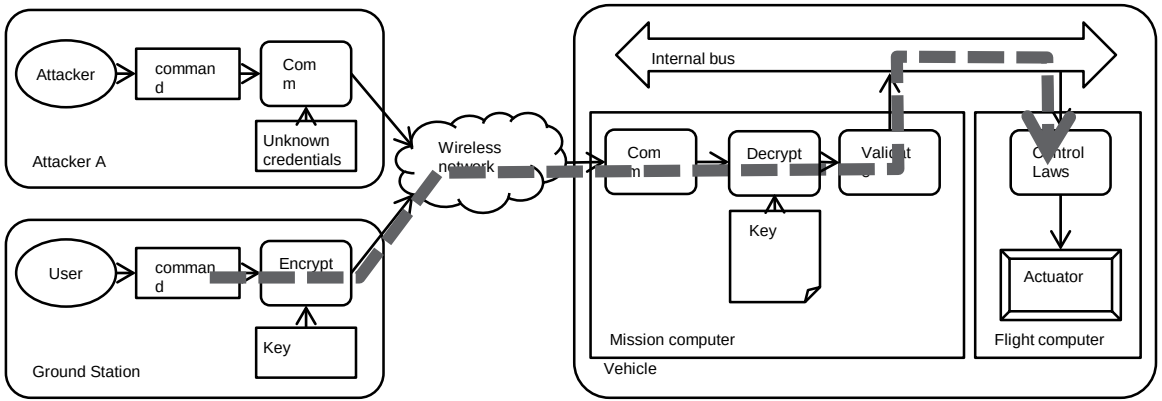





```
memory_protected(p : process) <=
  ** "The memory of process " p " is protected
from alterations by other processes" **
  property(p, SMACCM::OS) = "SeL4" or
  (property(p, SMACCM::OS) = "eChronos" and
   forall (mem : memory). bound(p, mem) =>
     forall (q : process). bound(q, mem) => memory_safe_process(q))

memory_safe_process(p : process) <=
  ** "The process " p " only writes to its own memory space" **
  forall (t : thread). contained(t, p) => memory_safe_thread(t)

memory_safe_thread(t : thread) <=
  ** "The thread " t " only writes to its own memory space" **
  ivory_thread(t)

ivory_thread(t : thread) <=
  ** "The thread " t " is generated from Ivory" **
  property(t, SMACCM::Language) = "Ivory"
```

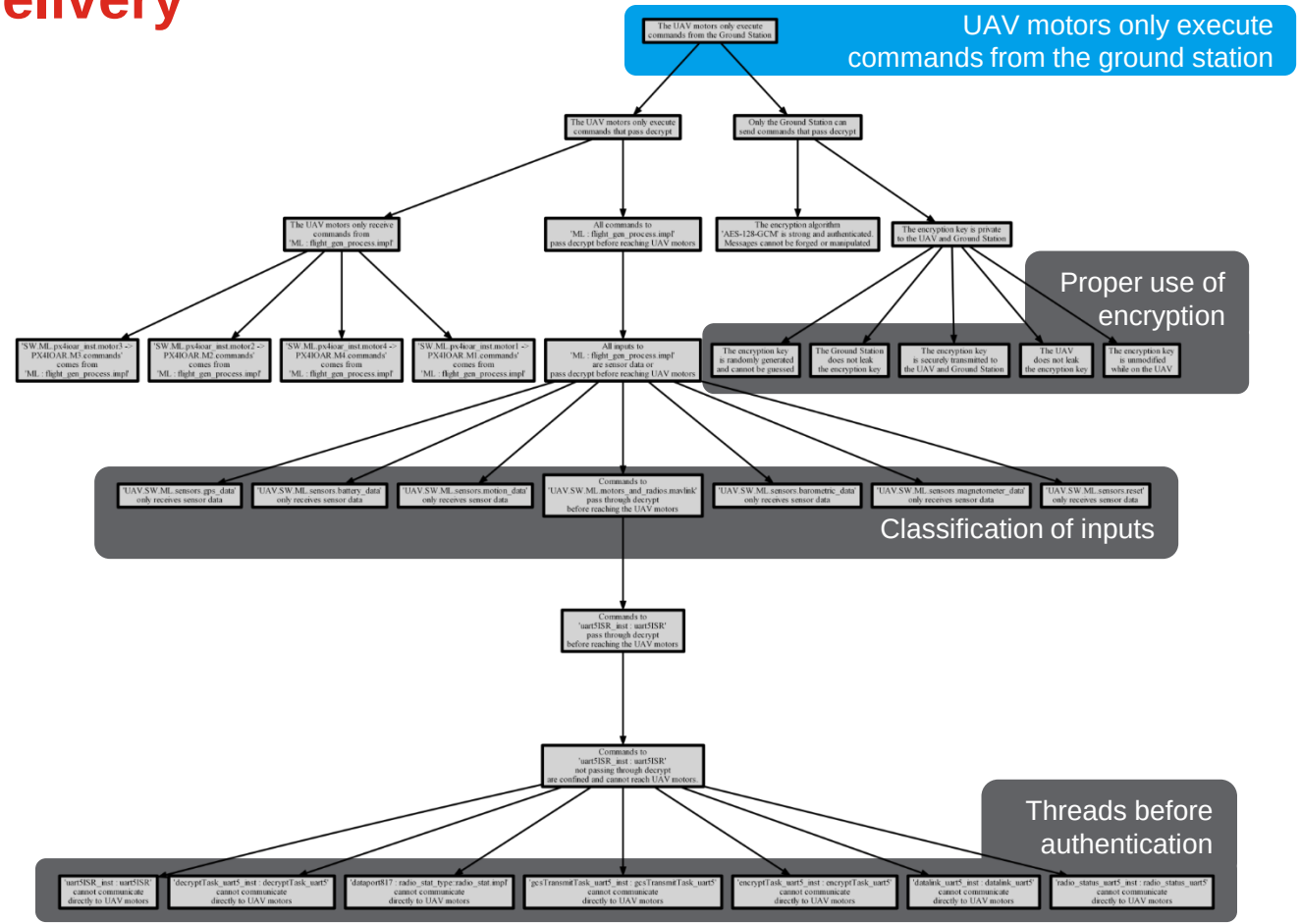


Properties from SMACCM Project



- The motor controller only receives messages from the trusted ground station.
- All messages received by the radio reach the motor controller.
- All connections are accurate/non-bypassable.
  - Requires memory-safety

Secure Delivery



Analysis Results

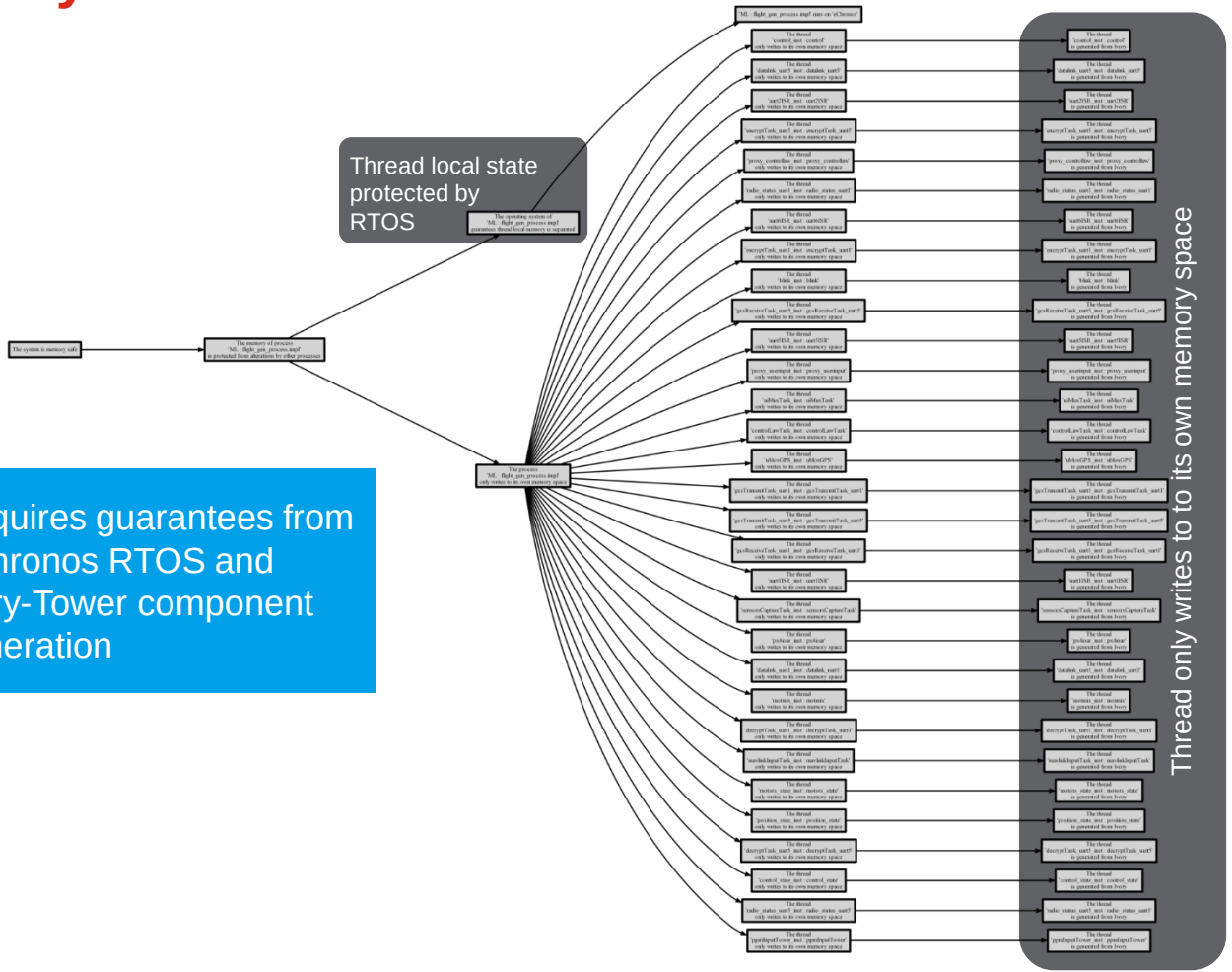
- only_receive_gs(ML : SOFTWARE::Main_Loop.Impl)
  - 'MC : SOFTWARE::Motor_Control' only receives messages from the Ground Station
    - Only the Ground Station can send messages that pass Decrypt
    - The component 'MC : SOFTWARE::Motor_Control' only receives messages that pass Decrypt
      - The connection 'SN.motor_commands -> MC.motor_commands' only carries messages that pass Decrypt
        - The connection 'SN.motor_commands -> MC.motor_commands' delivers data without alteration
        - The component 'SN : SOFTWARE::Stability_Navigation' only receives messages that pass Decrypt
          - The connection 'CCT.mavlink_out -> SN.mavlink' only carries messages that pass Decrypt
          - The connection 'RC.commands_out -> SN.rc_commands' only carries messages that pass Decrypt
            - The connection 'RC.commands_out -> SN.rc_commands' delivers data without alteration
            - The component 'RC : SOFTWARE::Radio_Control' only receives messages that pass Decrypt

Failed assurance case

- only_receive_gs(ML : SOFTWARE::Main_Loop.Impl)
  - 'MC : SOFTWARE::Motor_Control' only receives messages from the Ground Station
    - Only the Ground Station can send messages that pass Decrypt
    - The component 'MC : SOFTWARE::Motor_Control' only receives messages that pass Decrypt
      - The connection 'SN.motor_commands -> MC.motor_commands' only carries messages that pass Decrypt
        - The connection 'SN.motor_commands -> MC.motor_commands' delivers data without alteration
        - The component 'SN : SOFTWARE::Stability_Navigation' only receives messages that pass Decrypt
          - The connection 'CCT.mavlink_out -> SN.mavlink' only carries messages that pass Decrypt
            - The connection 'CCT.mavlink_out -> SN.mavlink' delivers data without alteration
            - The component 'CCT : SOFTWARE::Command_Control_Telemetry' only receives messages that pass Decrypt
              - The connection 'DC.decrypt_out -> CCT.mavlink_in' only carries messages that pass Decrypt
          - The connection 'SS.sensors_out -> SN.sensors' only carries sensor data

Successful assurance case

Memory Safety



Requires guarantees from eChronos RTOS and Ivory-Tower component generation

Galois: Domain-Specific Languages (DSLs)
The *Ivory-Tower* DSL

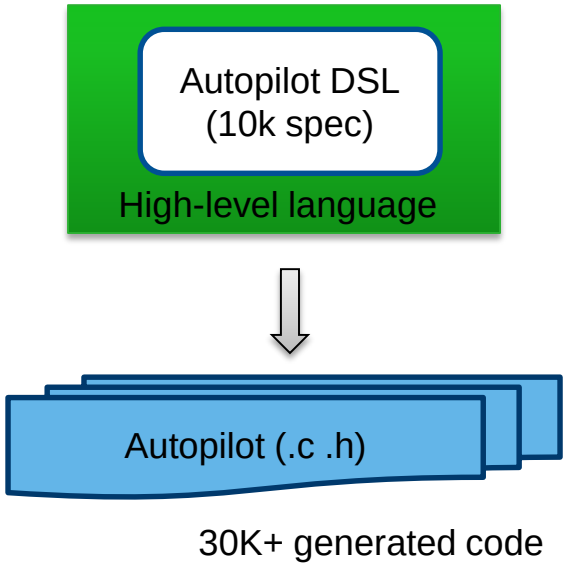
Trustworthy components

Embed the DSL in a high-level language:

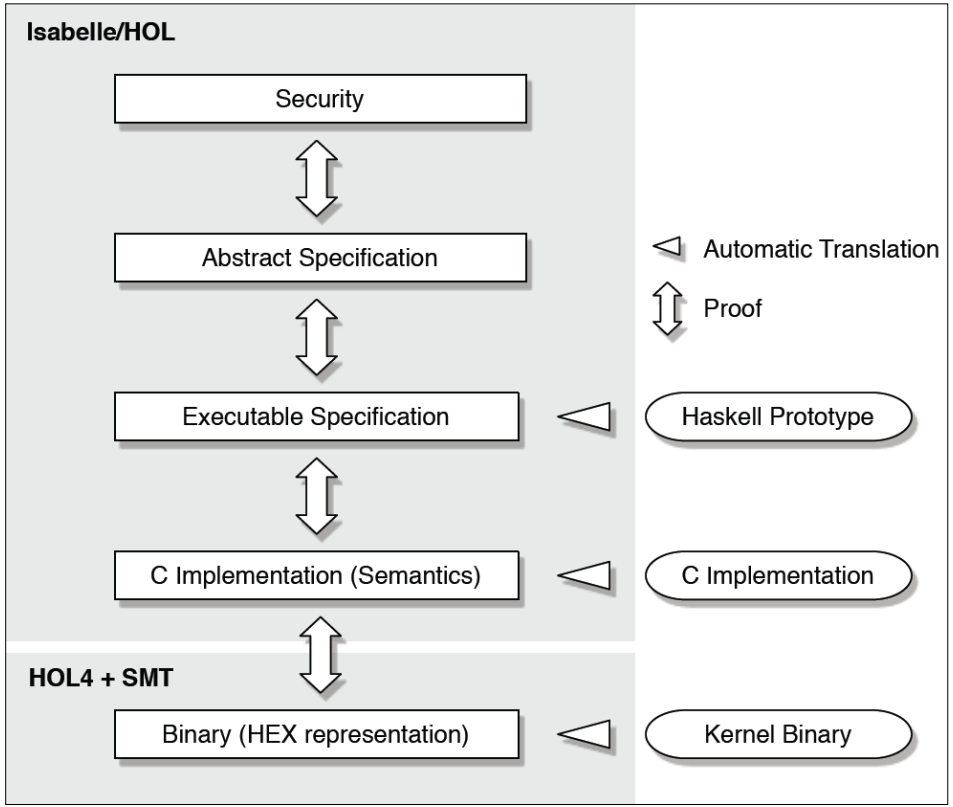
- DSL compiler is small (8k locs)
- Language prevents user from making common C errors
- Type-safe macros in the host language allow fast, safe programming

Outcome:

- 10k spec generates 30k locs code and ~2k locs properties, mostly
  - Arithmetic checks to prevent overflow
  - Checks on interface values
- New backends are created in about ~1k locs (C code, models, test cases)



Data61: seL4



seL4 is the world's first formally verified high-performance microkernel

The binary code correctly implements the behavior described in its abstract specification

…and nothing more

The specification and the binary satisfy the security properties called *integrity* and *confidentiality*

Now open source

sel4.systems

Bugs eliminated:

- Buffer overflow
- Null pointer dereference
- Pointer errors
- Memory leaks
- Arithmetic overflow/exceptions
- Undefined behaviors

System Construction: Trusted Build

Why Trusted Build?

Ensure fidelity between models and system image

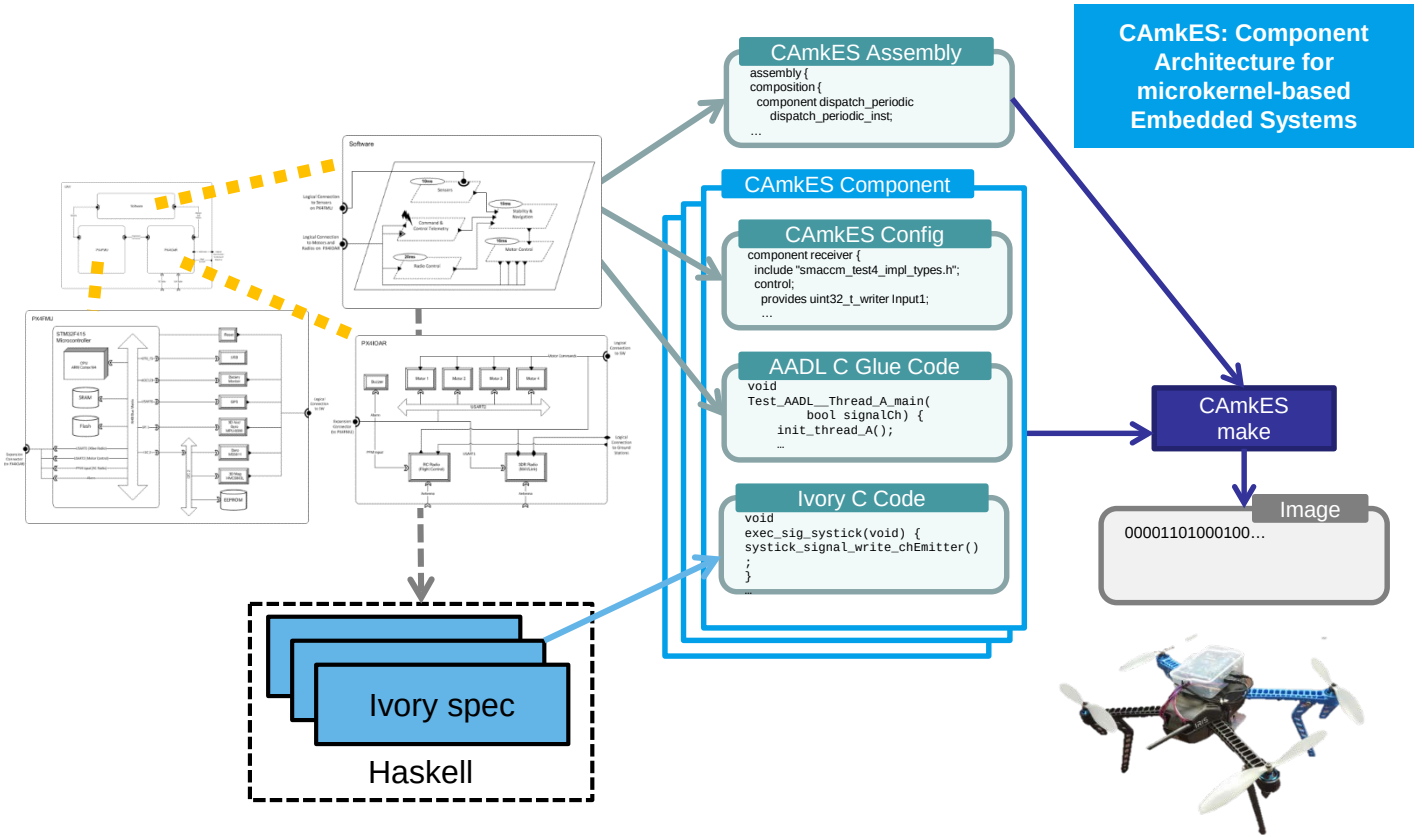
Proofs are over architectural models

- Information flow between processes and threads
- Well-formedness of architecture: scheduling, memory limits and safety, etc.

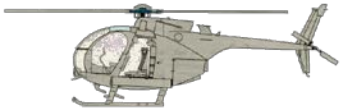
Trusted build **generates** system image from architectural model

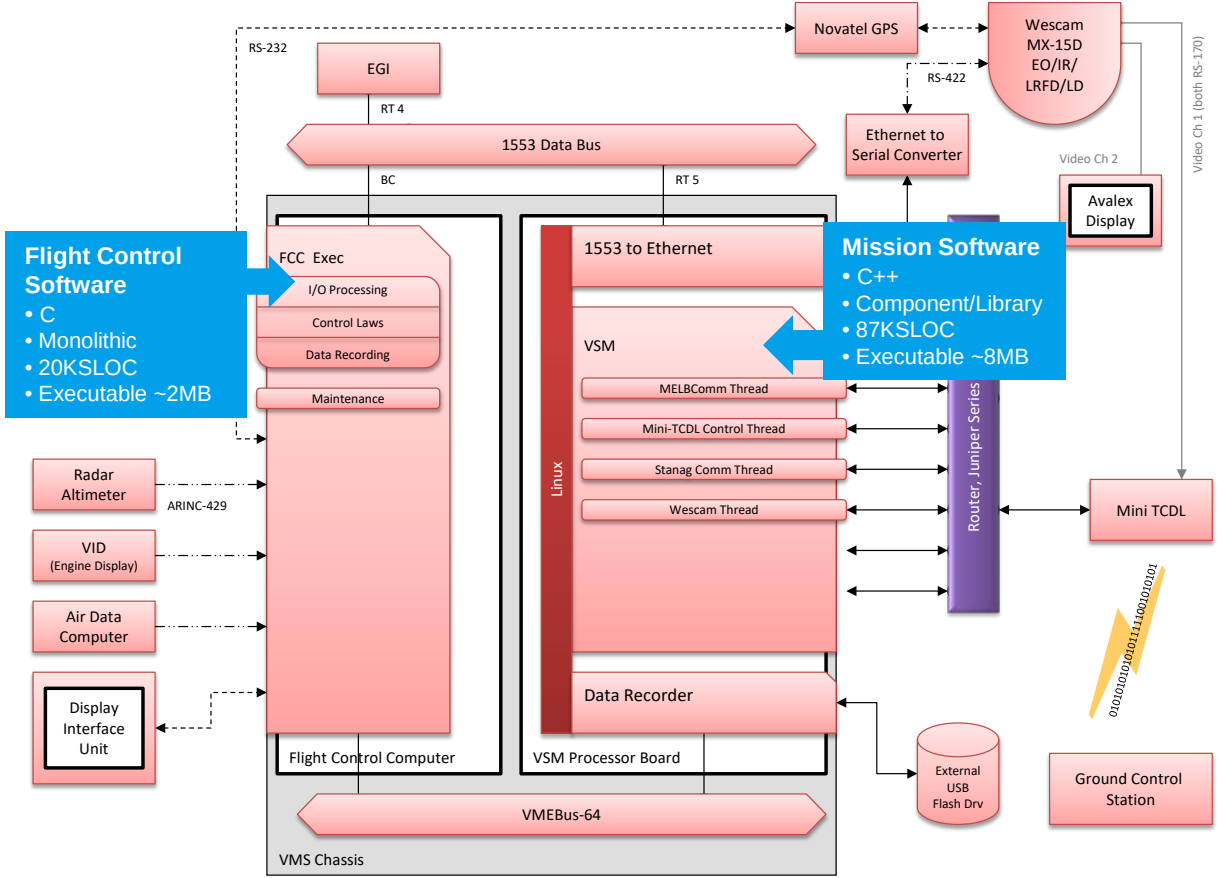
- Prevents stupid errors
  - Mismatches on unit types between modules [Mars Polar Lander]
  - Mismatches on alignment of data, data representation, and data location

Build Process for seL4



Baseline ULB Architecture





Phase 3 Architecture

AADL models for both VSM and FCC

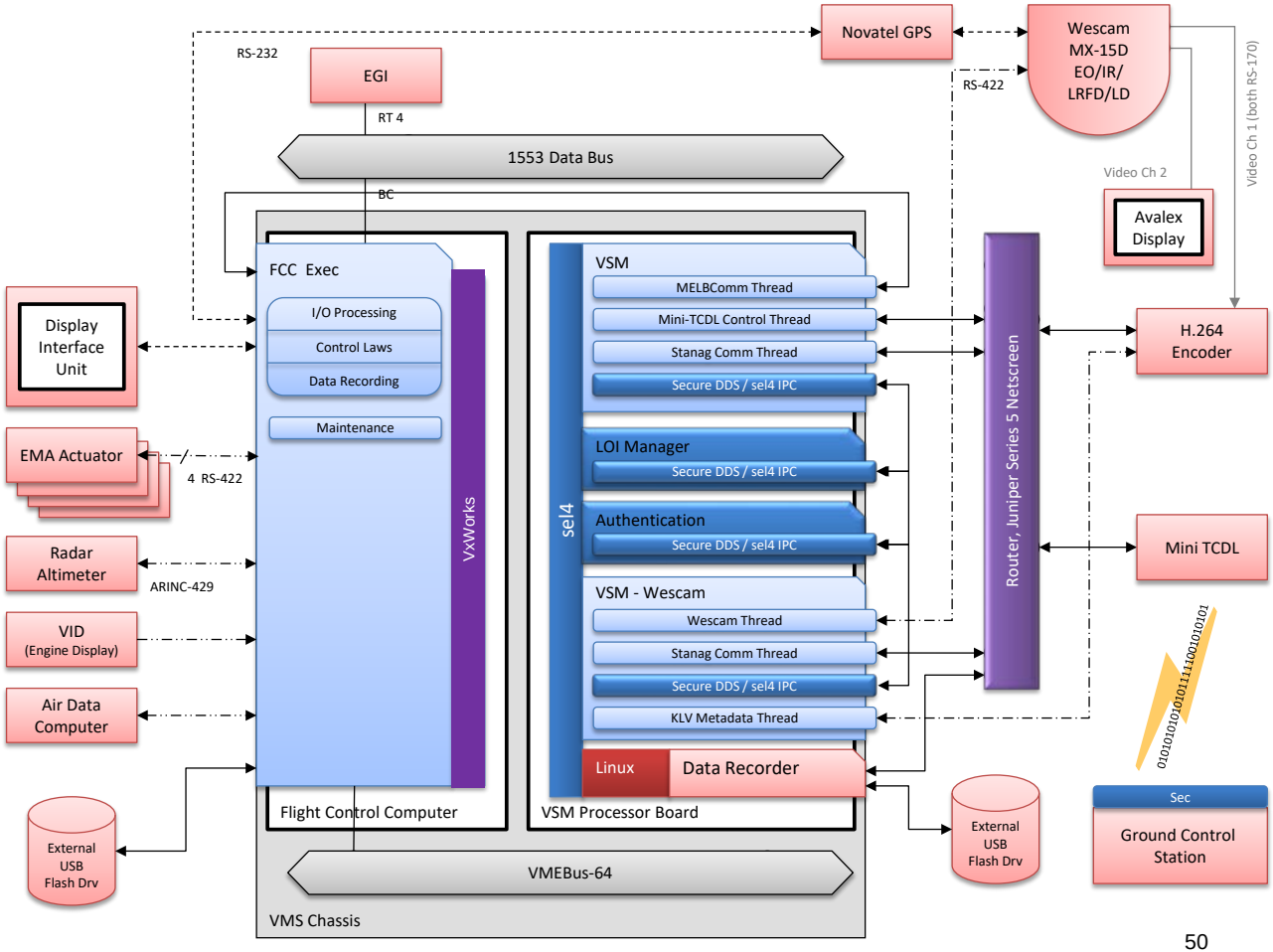
VSM Models deployed to seL4

FCC AADL model deployed to VxWorks

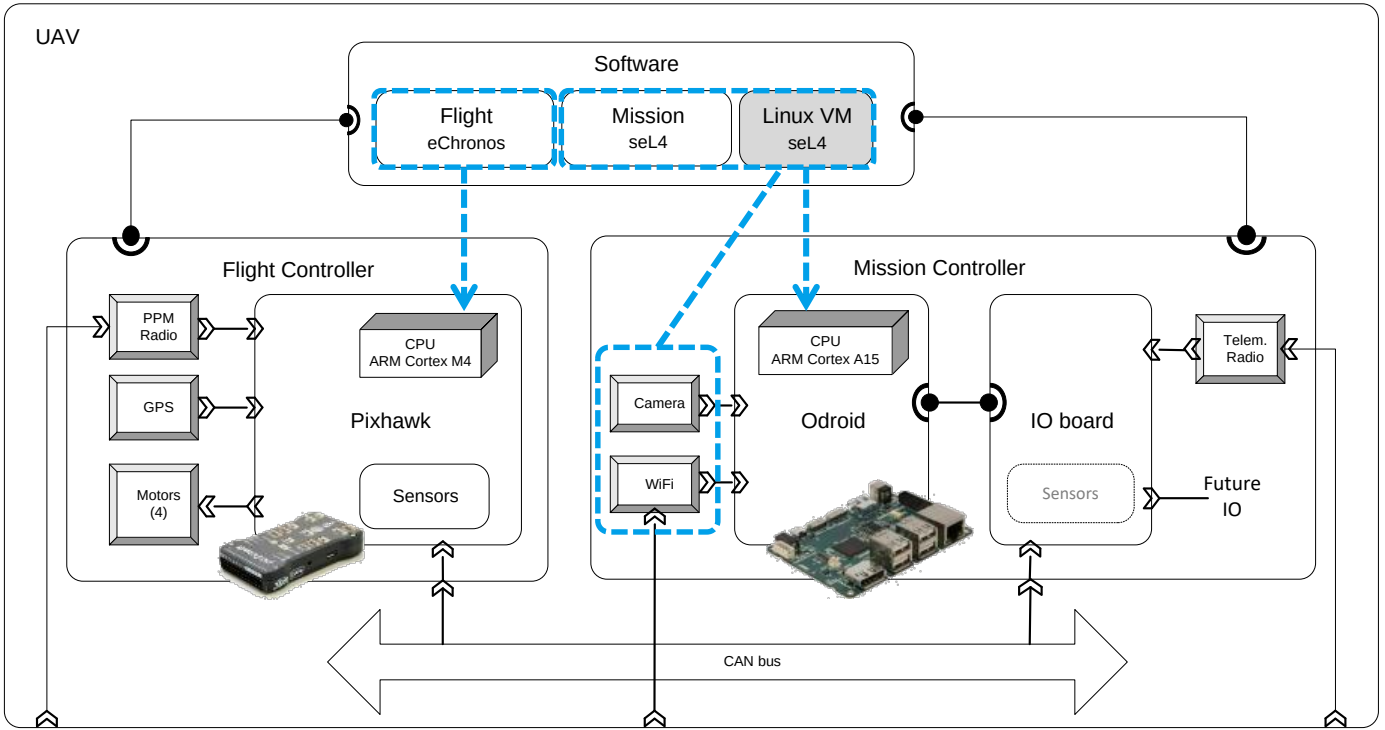
Some FCC components implemented in Ivory/Tower

VSM split into “navigation” and camera VSMs
Authentication and LOI Ivory components
Other VSM components in Ivory

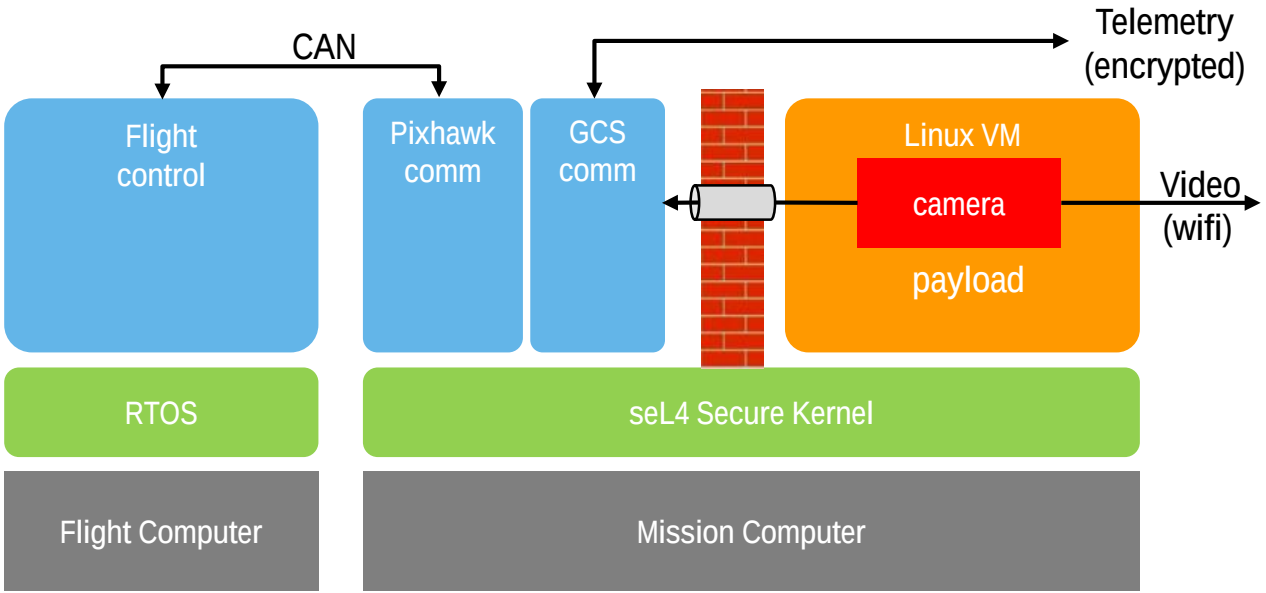
VSMs natively on seL4
Simplified VSM to FCC Communication



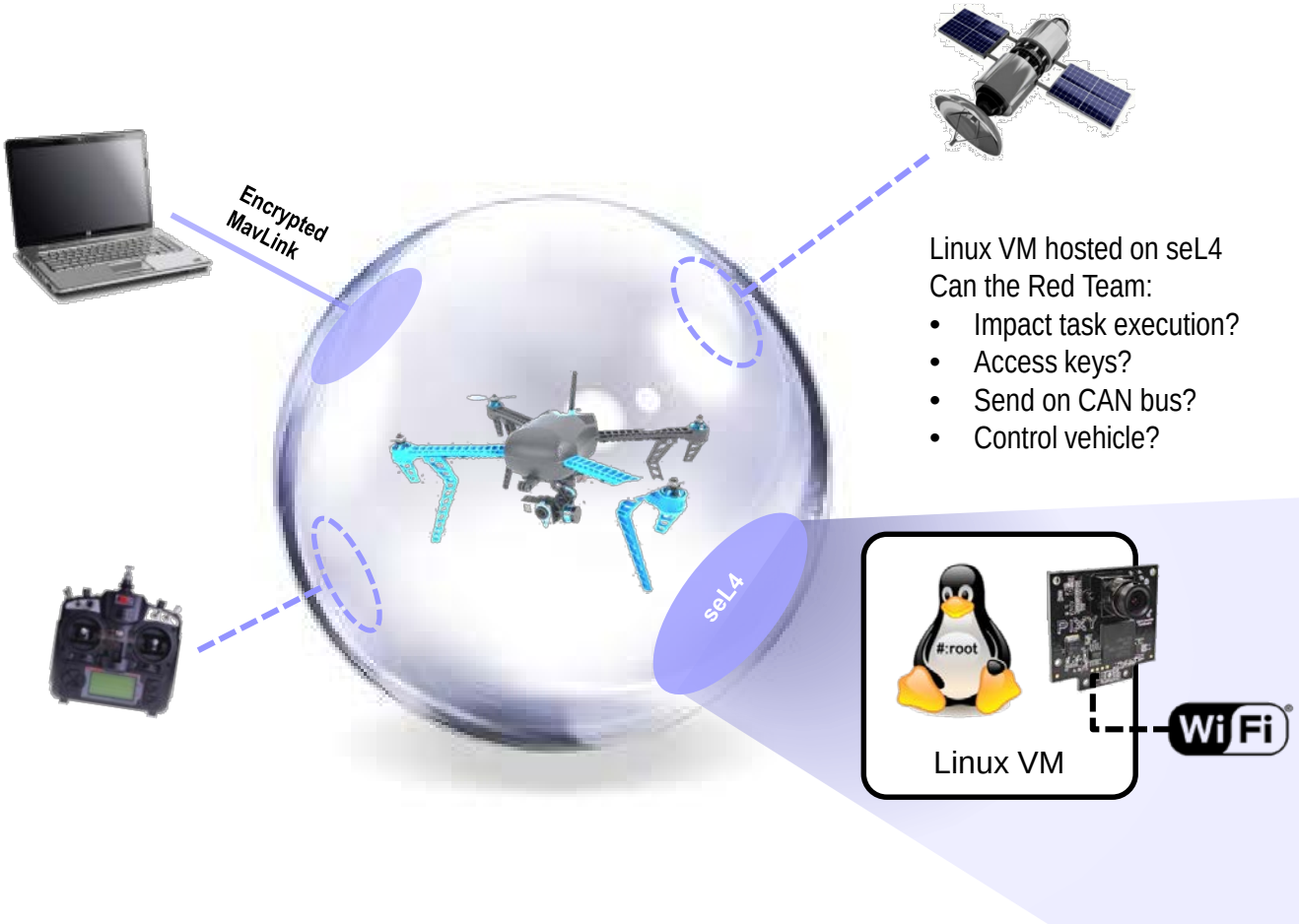
SMACCMcopter Phase 3 Architecture



Secure Software Architecture



Evaluation: Attack Surface



Linux VM hosted on seL4
Can the Red Team:

- Impact task execution?
- Access keys?
- Send on CAN bus?
- Control vehicle?

Evaluation: Assessment / Attack

3 month assessment by *Red Team*

- Team of professional penetration testers
  - In other efforts, the red team has broken a large number of both commercial and military systems (including the baseline Little Bird)
- Provided all artifacts: design documents, source code, architecture models, system images

For secured attack surfaces, no vulnerabilities found!

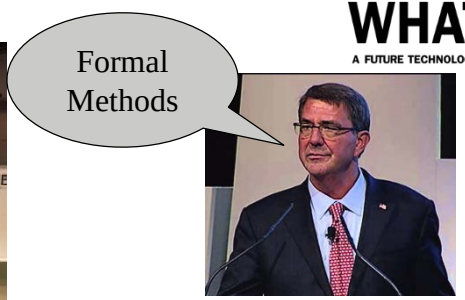
- For each phase, for each vehicle
- For both the SMACCMCopter and the Little Bird

DARPA “Wait, What?”

St. Louis, 9-11 Sept 2015

WAIT? WHAT

A FUTURE TECHNOLOGY FORUM



Conclusion

Vehicles can now come under remote attack due to networking

Current vehicle architectures are not built for cyber-security: vulnerable to a broad range of attacks

For security, we need secure software architectures:

- Secure communications with ground
- Enforced software task separation onboard
- Correct data flows and behaviors in system architecture

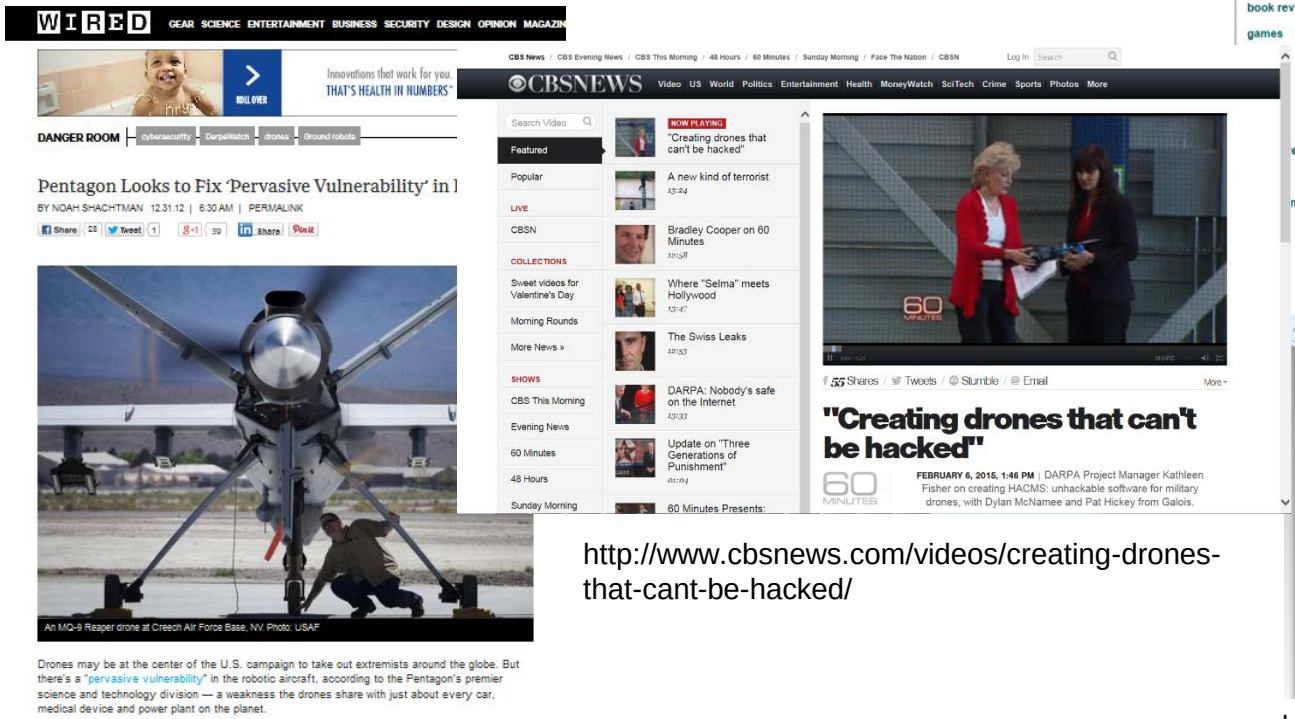
…and secure software components:

- Analysis and construction based on *formal methods*

Untrusted code can be safely used within trusted boundaries

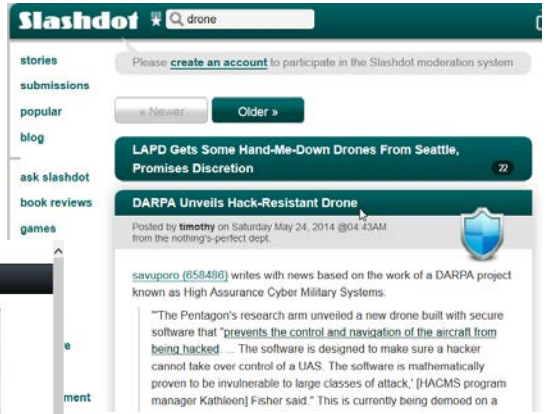
Secure systems can be built at scale without sacrificing performance

Some Press…

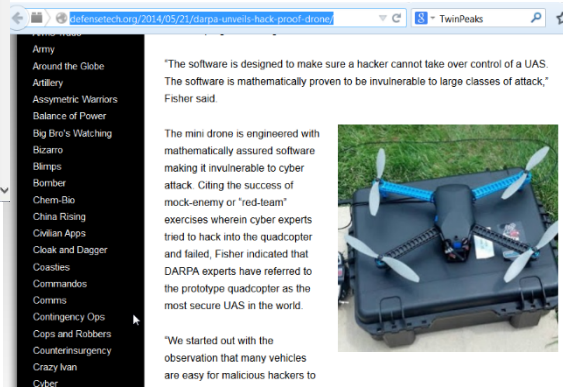


http://www.wired.com/2012/12/darpa-drones/

http://www.cbsnews.com/videos/creating-drones-that-cant-be-hacked/



http://slashdot.org/index2.pl?fhfilter=drone



http://defensetech.org/2014/05/21/darpa-unveils-hack-proof-drone/

HEADLINE IN WHITE, ARIAL BOLD 22PT

BODY HEADLINE ARIAL BOLD 18PT WITH SINGLE LINE SPACING

Body Arial Regular 18 pt with single line spacing

- Bullet one using “Increase List Level” button, Arial 18pt with single line spacing

HEADLINE IN WHITE, ARIAL BOLD 22PT

BODY HEADLINE ARIAL BOLD 18PT WITH SINGLE LINE SPACING

Body Arial Regular 18 pt with single line spacing

- Bullet one using “Increase List Level” button, Arial 18pt with single line spacing

What is the AADL?

SAE International Architecture Analysis and Design Language (AADL) is a *standard architecture modeling language, developed by and for the avionics, aerospace, automotive, and robotics communities.**

Uses component-based notation for the *specification* of task and communication *architectures of real-time*, *embedded, fault-tolerant, secure, safety-critical, software-intensive systems*.

The language & associated tools are used to *model, analyze*, and generate embedded real-time systems

Tool-based analysis in Eclipse framework

A modeling infrastructure that supports ***model-based engineering concepts***

Based on 15 Years of DARPA funded ***research*** technologies

First published Nov 2004 (***V1)*** - revised standard Jan 2009 ***(V2)***

SAE International

**** SAE International standard document AS 5506A (R)***