Rigorous Agentic Systems Engineering for seL4 using HAMR

seL4 Summit – Sept 3, 2026

Kansas State University

John Hatcliff
Robby
Jason Belt

Aarhus University

Stefan Hallerstede

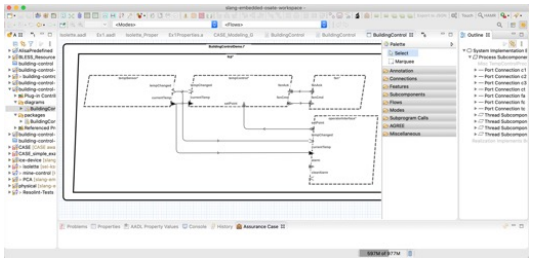
With collaborators at …

Collins Aerospace
Dornerworks
UNSW
Proofcraft
Carnegie Mellow Univ.
Univ. of Kansas

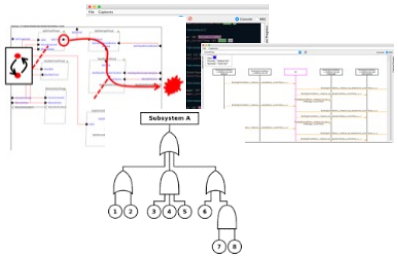
HAMR

HAMR – tool chain for building high-assurance embedded systems

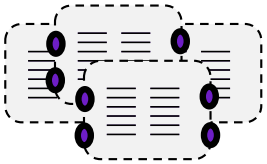
Modeling, analysis, and verification in the **SysMLv2** and **AADL** modeling languages



Model-level behavior specifications (e.g., contracts) and analyses



Component development and verification in multiple languages



- Rust (with Verus contract verification framework)
- C
- Slang (developed at Kansas State)
  - safety-critical subset of Scala
  - contract language + verification

Deployments on multiple platforms

JVM Deployment

Linux Deployment

seL4 Deployment

verified micro-kernel

CAmkES & microKit

Potential Benefits to seL4 Application Developers

Collins Aerospace INSPECTA: **A systems engineering environment** based on standardized modeling languages (SysMLv2, AADL) with accompanying analysis, verification, and assurance case tools

Systems Engineering Ecosystems

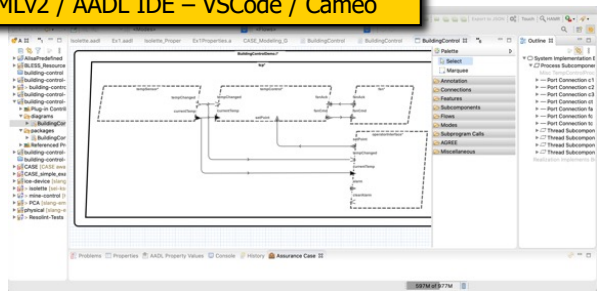
Requirements Engineering

Diagrams for Planning

Domain Concepts

Systems Engineering

SysMLv2 / AADL IDE – VSCode / Cameo

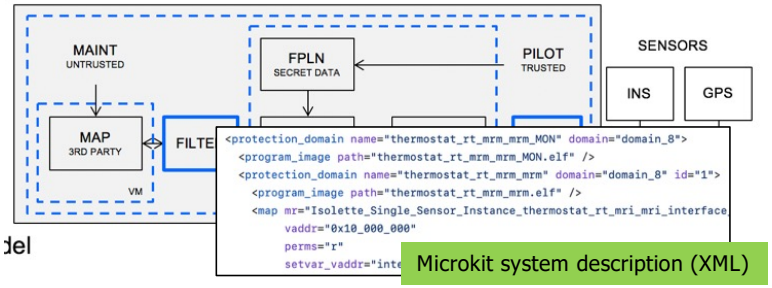


..integrate seL4 configuration with…

Software/Hardware/Middleware modeling
Formal specs + testing+ verification
Assurance case construction
..
Information Flow Analysis
Timing / Scheduleability

seL4 Deployment

Microkit / CAmKES Specification



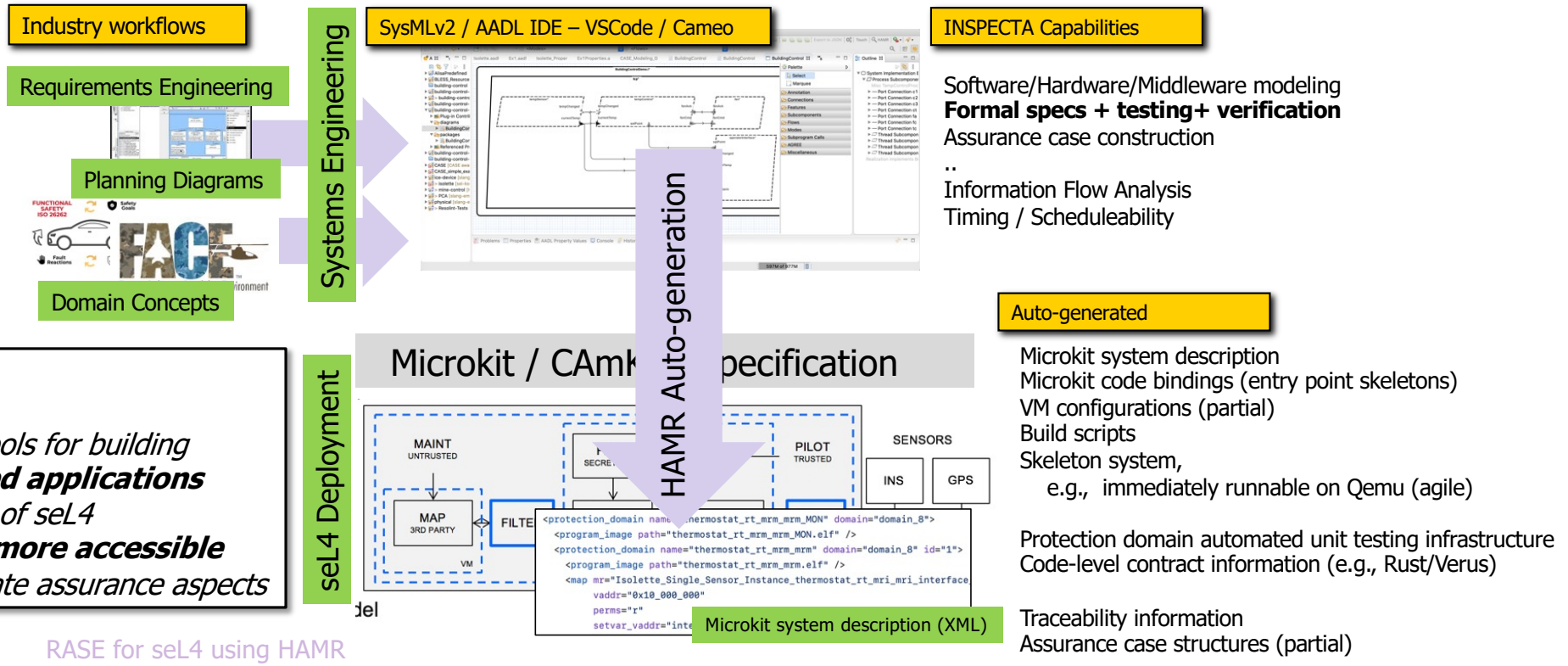
Microkit system description (XML)

seL4 ecosystem tools for kernel configuration…

- seL4 permissions (capabilities),
- partitioning,
- inter-partition communication
- virtual machines hosting

Potential Benefits to seL4 Application Developers

Collins Aerospace INSPECTA: **A systems engineering environment** based on standardized modeling languages (SysMLv2, AADL) with accompanying analysis, verification, and assurance case tools



Goals:

* *MBD tools for building* ***verified applications*** *on top of seL4*
* ***seL4 more accessible***
* *automate assurance aspects*

RASE for seL4 using HAMR

Challenges – In a World Rushing to Agentic Development

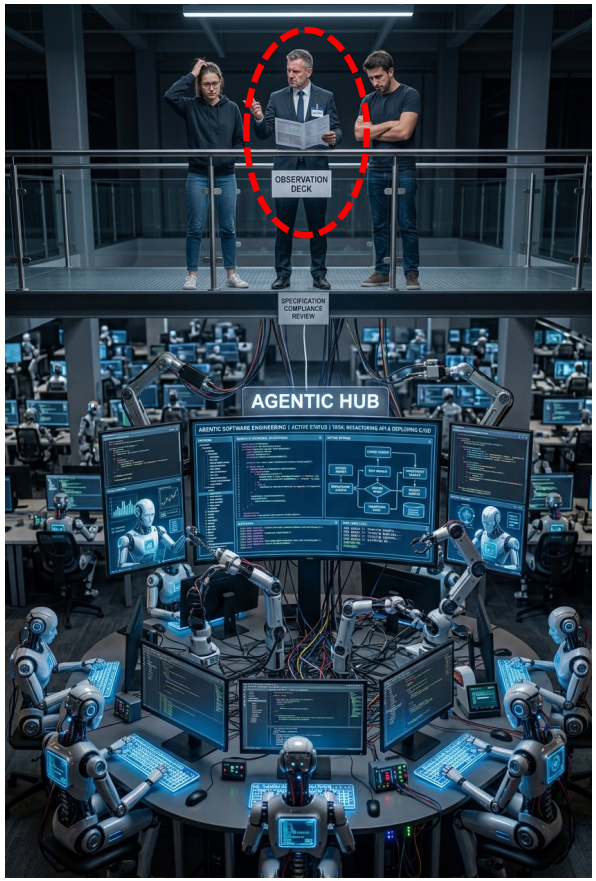
Enormous capacity for rapidly producing code, writing tests, running tools, performing reviews, …

Challenges for development…

- "Understanding is the new bottleneck"
- Agent outputs are non-deterministic and possibly wrong
  - New emphasis for developers, instead of writing code…
    - Writing "blueprints" - formal and informal specifications to direct agents
    - Using trustworthy methods to check that the *system realization satisfies the specifications*
- Did we get the specifications right? (understanding)

Reference: "Understanding is the new bottleneck" – Geoffrey Litt
https://www.geoffreylitt.com/2026/07/02/understanding-is-the-new-bottleneck

Challenges - In a World Rushing to Agentic Development



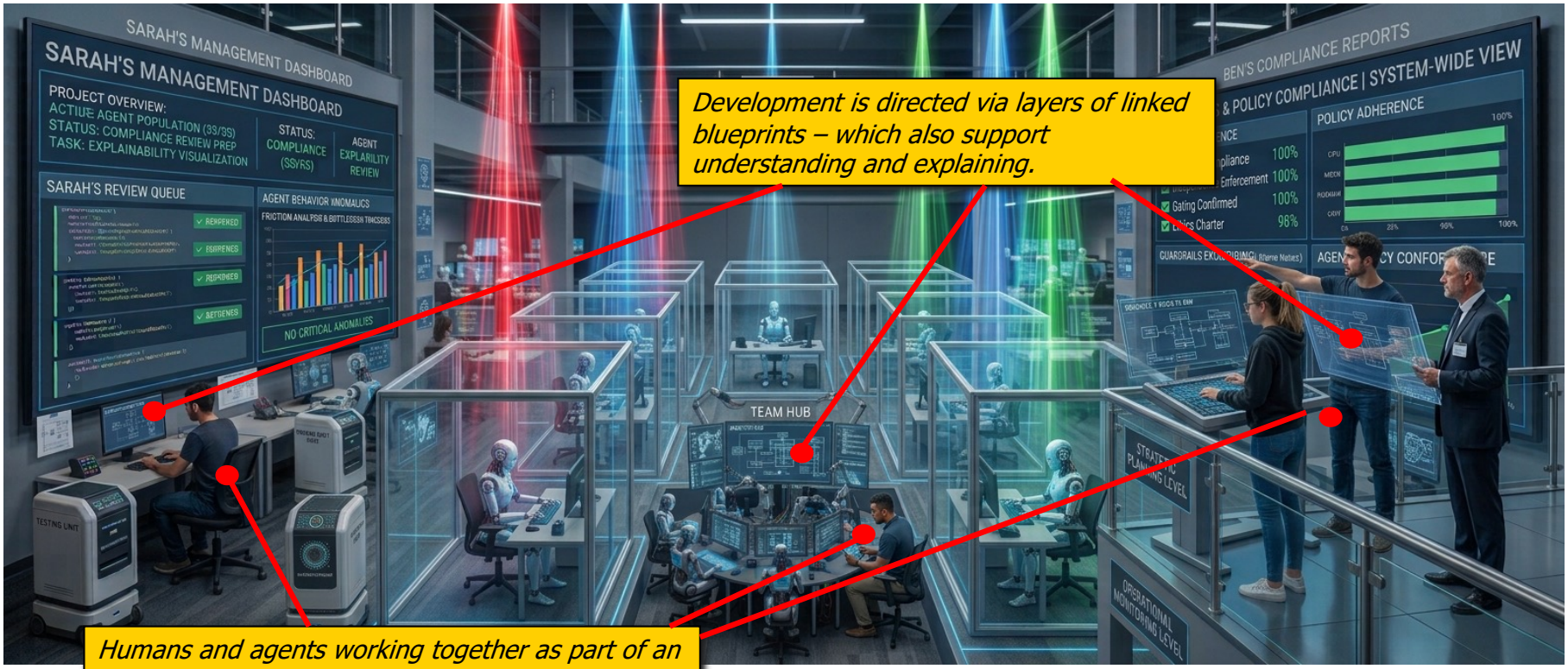
RASE for seL4 using HAMR

Challenges for certification – convincing auditors that our systems are safe, secure, and satisfy their specifications

- How to provide evidence-backed arguments to certifiers when we have trouble understanding ourselves?

- Certification took a long time before…
- To retain the benefits of agentic output, we need assurance at greater velocity and scale
  - "Explainability / Assurance / Compliance is the new bottleneck"

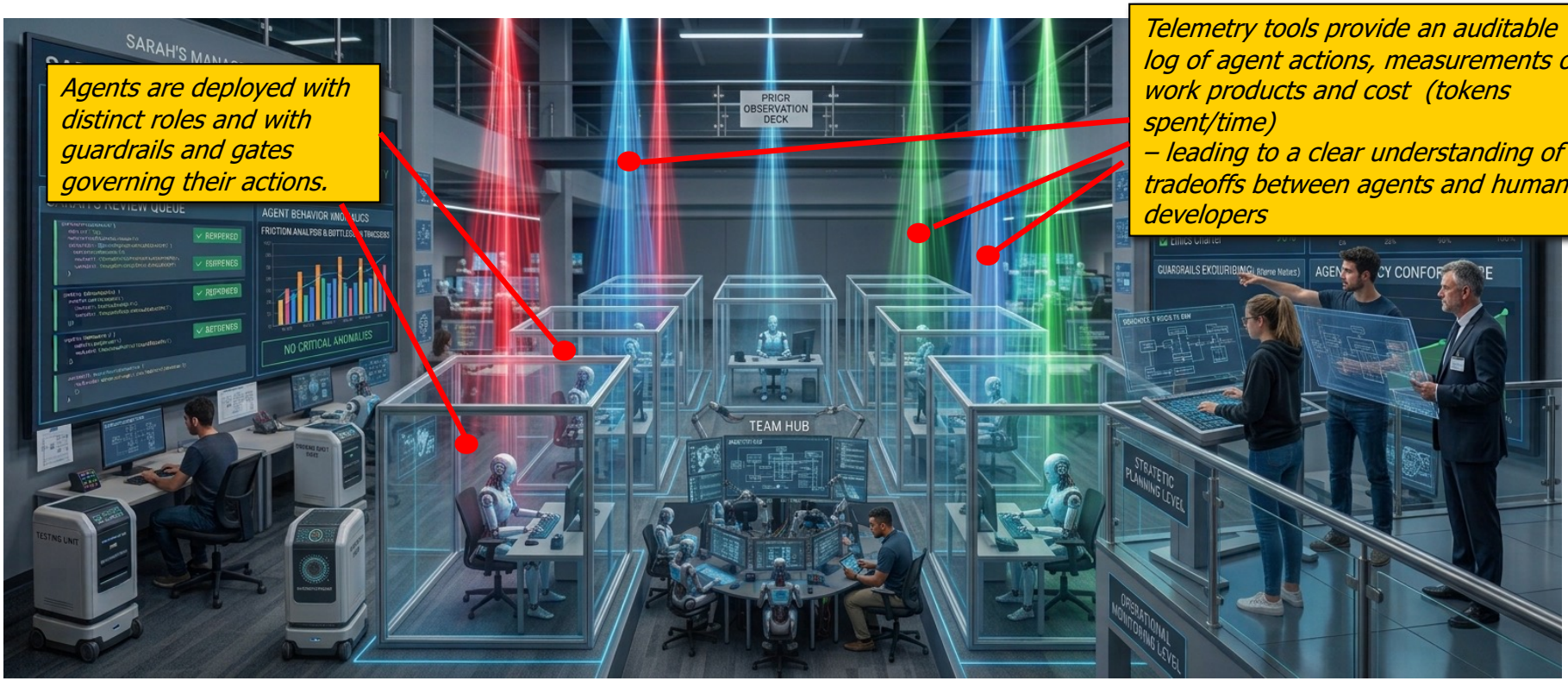
Our Vision:
Rigorous Agentic Systems Engineering



Development is directed via layers of linked blueprints – which also support understanding and explaining.

Humans and agents working together as part of an integrated repeatable engineering process

Rigorous Agentic Systems Engineering

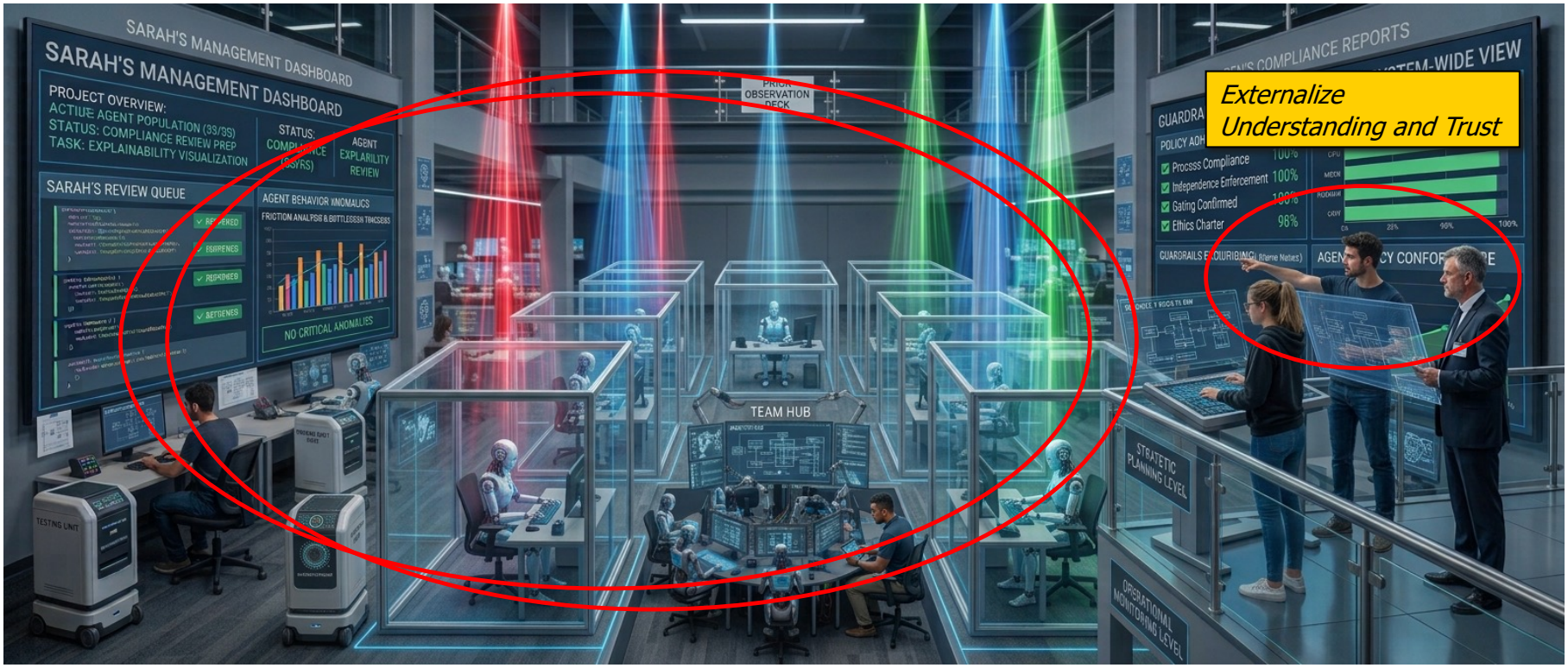


Rigorous Agentic Systems Engineering

Compliance Dashboards -- *summarize conformance of agent actions to specified processes and guardrails, and to compliance documents*

Development Reports & Dashboards -- *summarizing development state (how is the work decomposed and allocated, work completed/TBD), status of conformance to blueprints…*

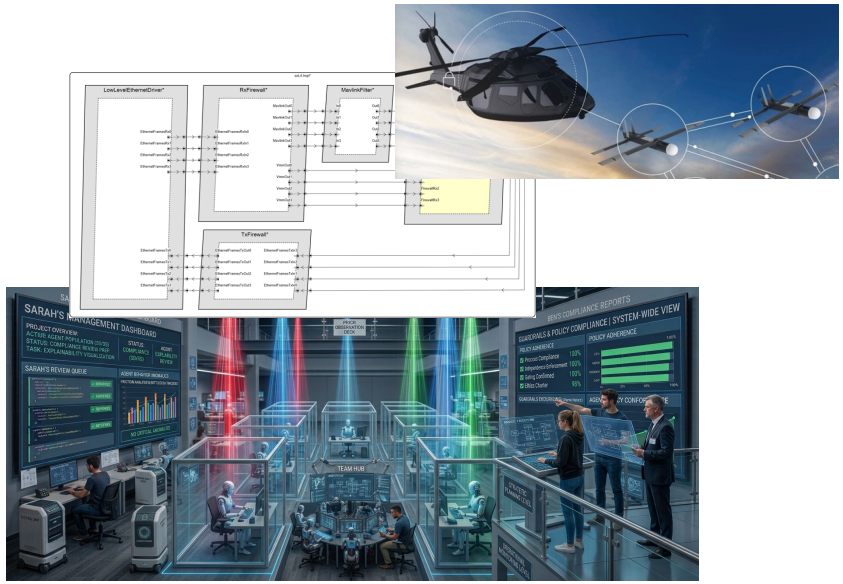
Rigorous Agentic Systems Engineering

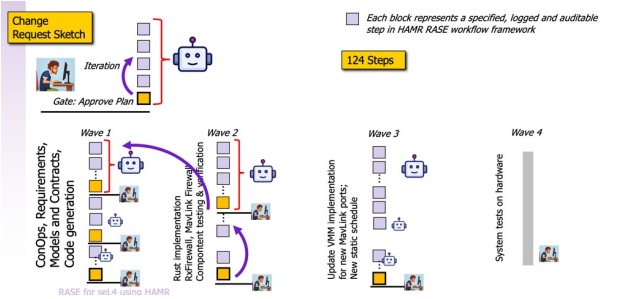


"Governed Autonomous Engineering Environment" that externalizes trust in both product and process "at velocity" and "at scale"

Talk Outline / Arc

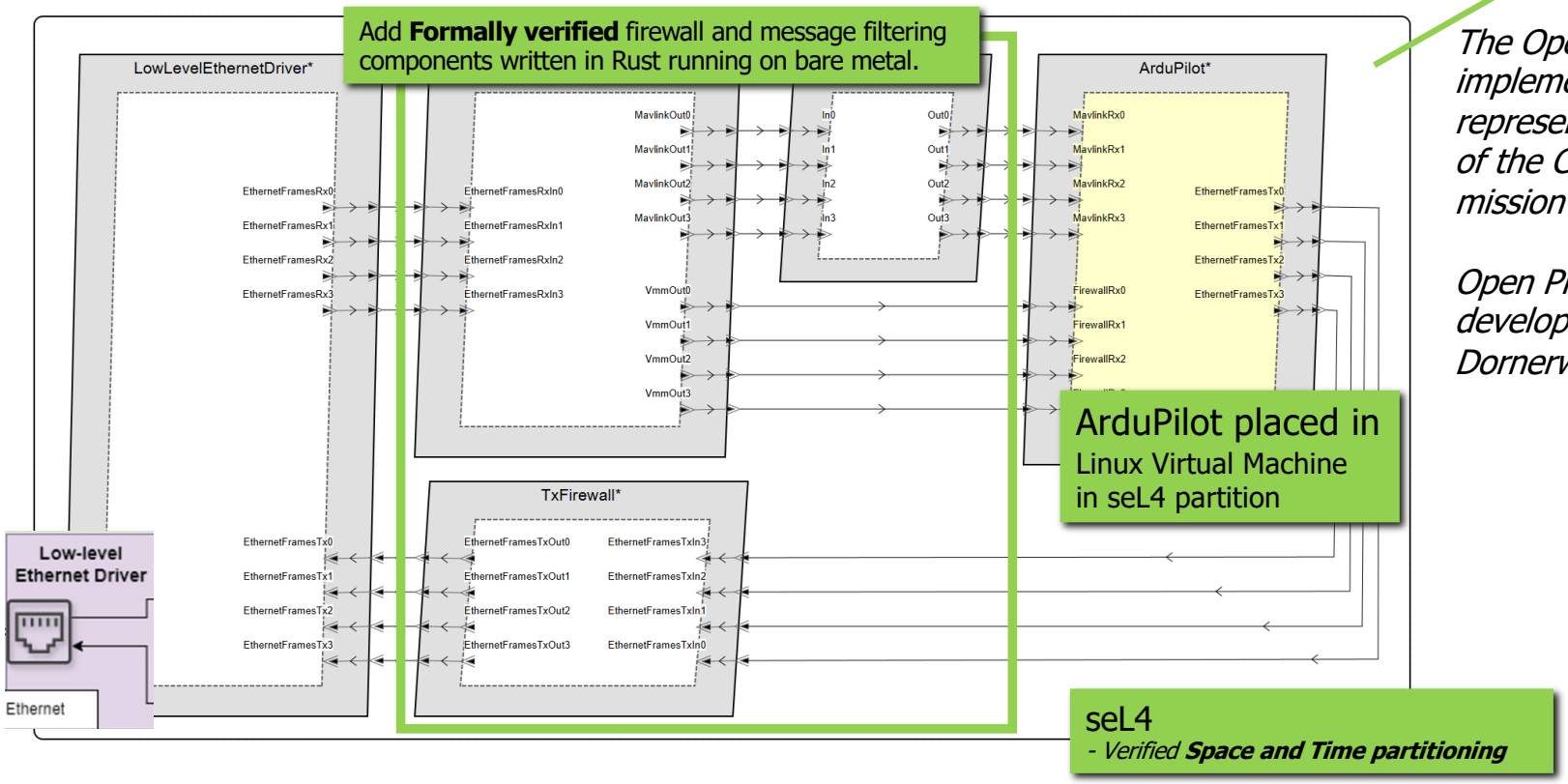
- Example Artifacts –
  - What automation HAMR provides already;
  - what it means for agents
- RASE Agent Workflow Framework
- Telemetry; Friction Reporting
- Experiments with Dornerworks as part of the Collins INSPECTA project





INSPECTA Public Demonstrator: *Open Platform* Basic Architecture

"Cyber-Retrofit" -- Improving Security & Cyber-Resiliency of Legacy System



The Open Platform implements representative features of the Collins RapidEdge mission computer

Open Platform development is led by Dornerworks.

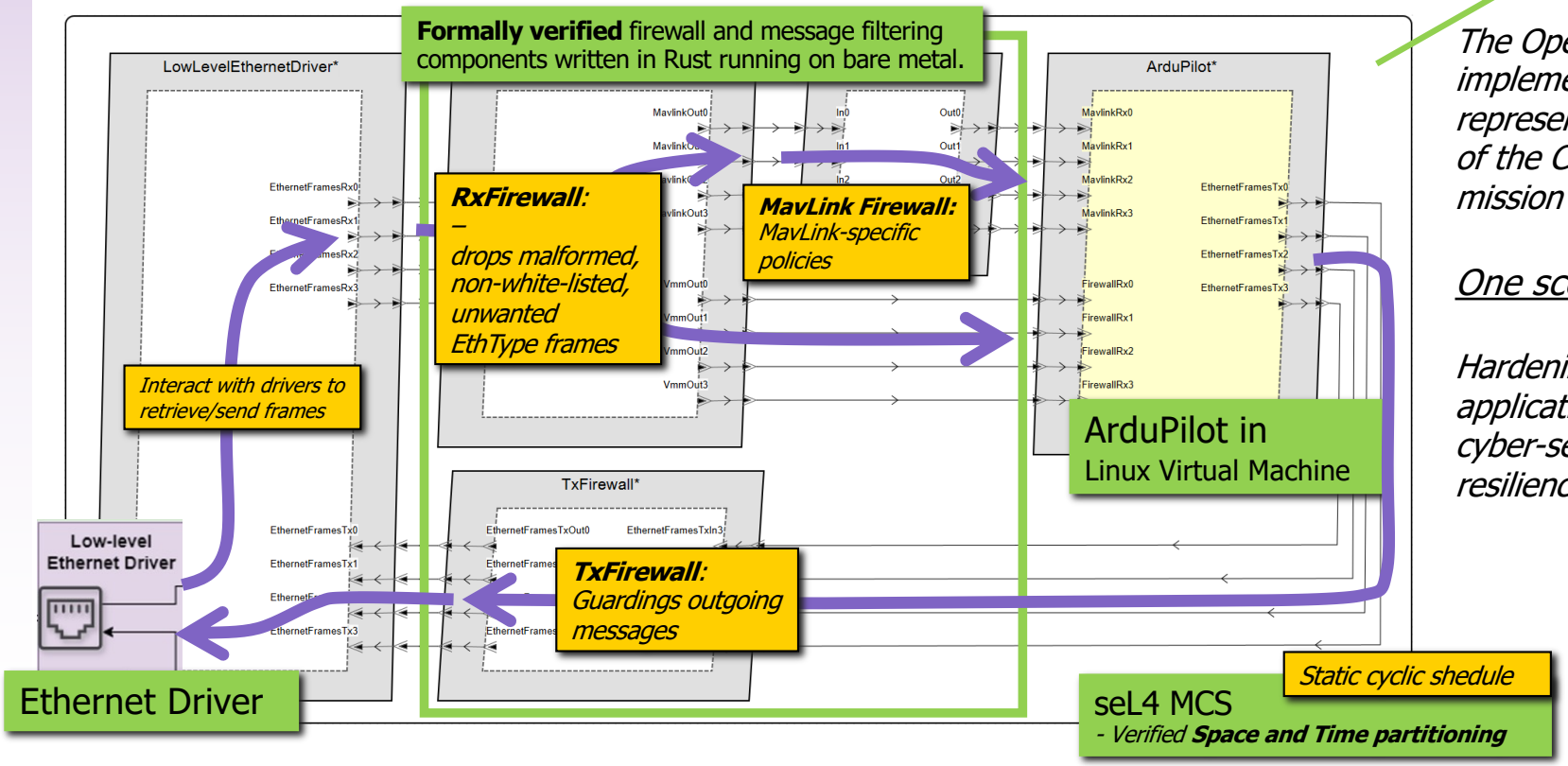
Functional Concepts

High-level concepts of message policy enforcement..

The Open Platform implements representative features of the Collins RapidEdge mission computer

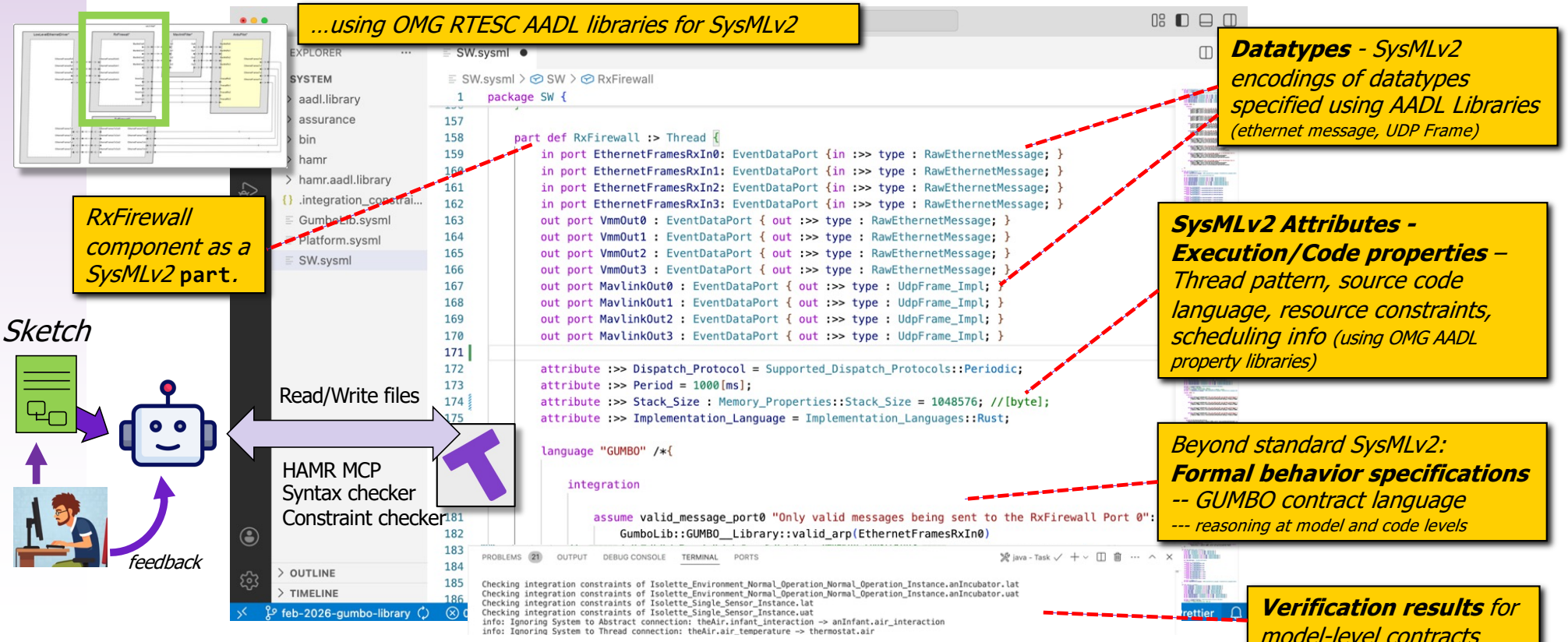
<u>*One scenario:*</u>

Hardening a legacy application for greater cyber-security and resiliency



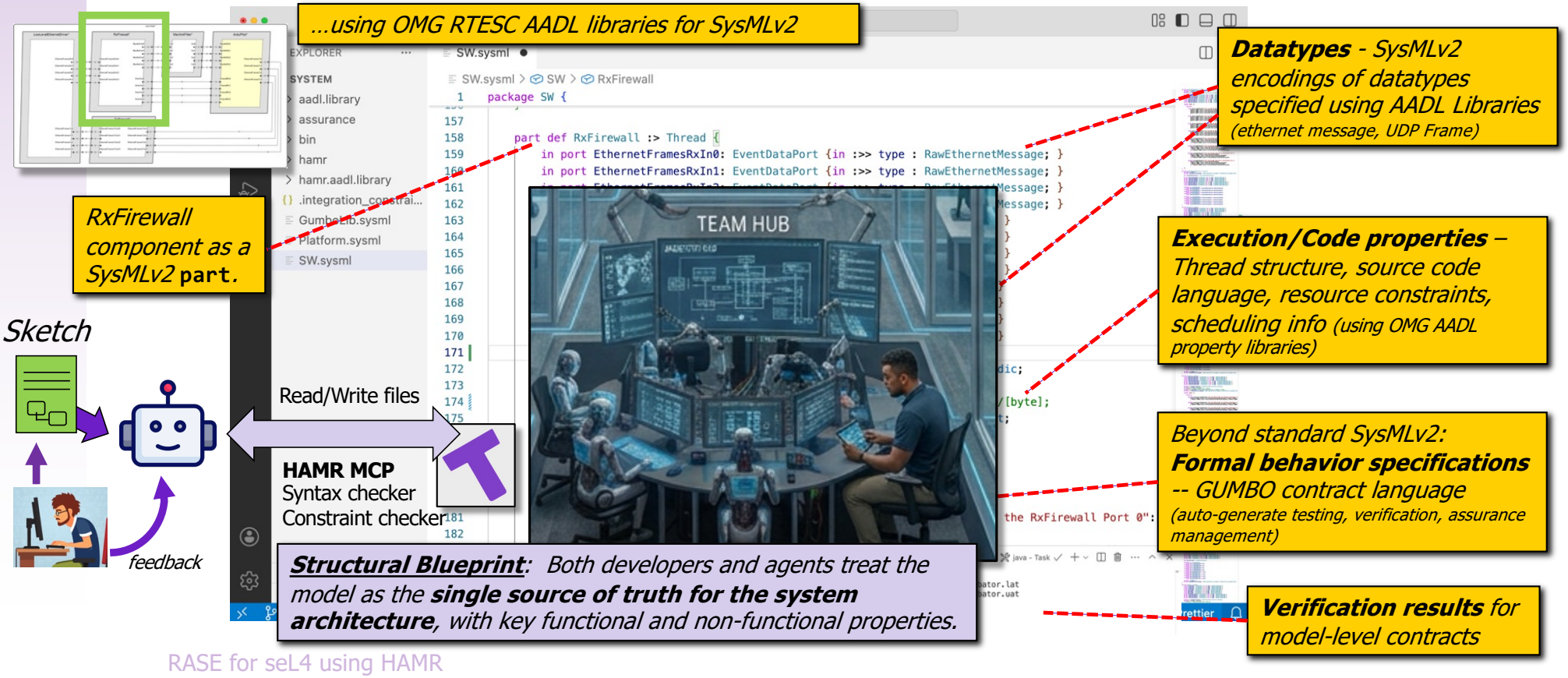
VSCode SysMLv2 HAMR Front End

SysMLv2 front-end for HAMR – enabling model-based development for seL4

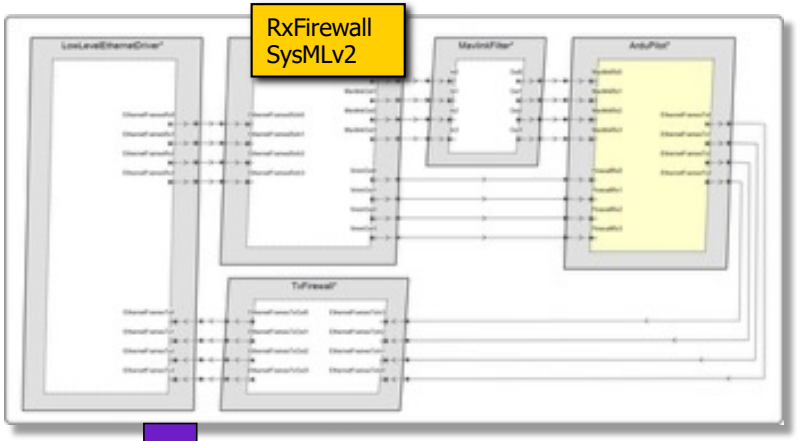


VSCode SysMLv2 HAMR Front End

SysMLv2 front-end for HAMR – Dornerworks specified the desired architecture of the system



HAMR Generates seL4 Kernel Configuration



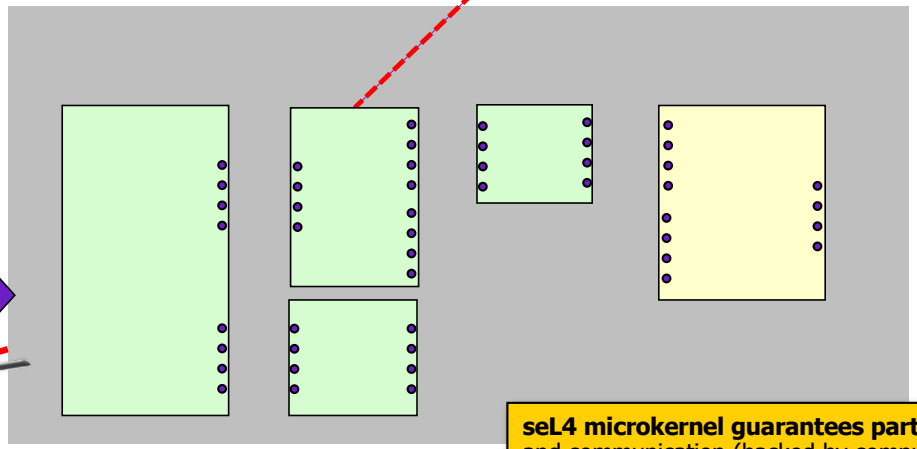
RxFirewall SysMLv2

```
<protection_domain name="seL4_MavlinkFirewall_MavlinkFirewall_MON" (
  <program_image path="seL4_MavlinkFirewall_MavlinkFirewall_MON.elf
  <protection_domain name="seL4_MavlinkFirewall_MavlinkFirewall" domain="domain_6" id="1">
    <program_image path="seL4_MavlinkFirewall_MavlinkFirewall.elf" />
    <map mr="ZCU102_Impl_Instance_seL4_MavlinkFirewall_MavlinkFirewall_Out0_1_Memory_Region"
         vaddr="0x10_000_000"
         perms="rw"
         setvar_vaddr="Out0_queue_1"
    />
    <!-- ... other ports omitted -->
  </protection_domain>
</protection_domain>
```

RxFirewall Microkit Protection Domain (partition) spec

- Auto-generated **partition declaration for component** as a seL4 Microkit protection domain,
- **memory block for each port, and**
- **read/write permissions --** information flow controls for component input/output

HAMR Code Generation for seL4 – a **bunch of artifacts for building and deploying**



seL4 microkernel guarantees partitioning of components and communication (backed by computer-checked proofs)

RASE for seL4 using HAMR

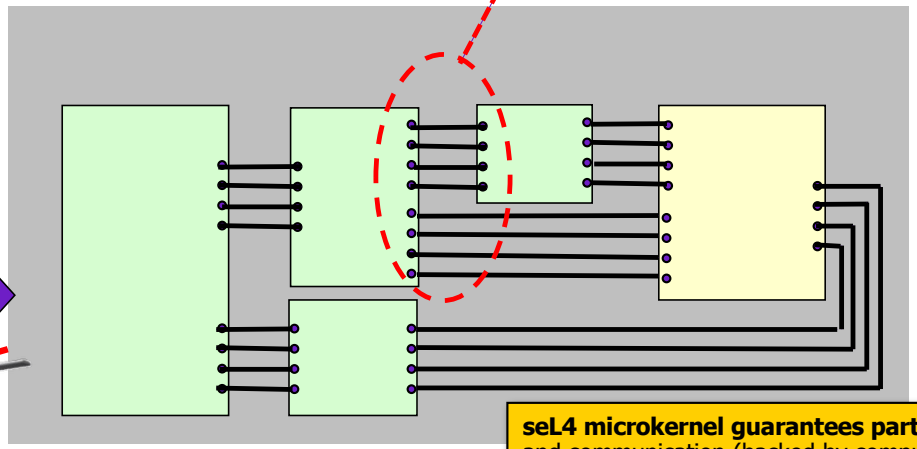
HAMR Allocates and Configures 1-Way Communication Paths Between Partitions

Specs for communication pathways between partitions

Memory region declarations implementing communication channels between each partition.

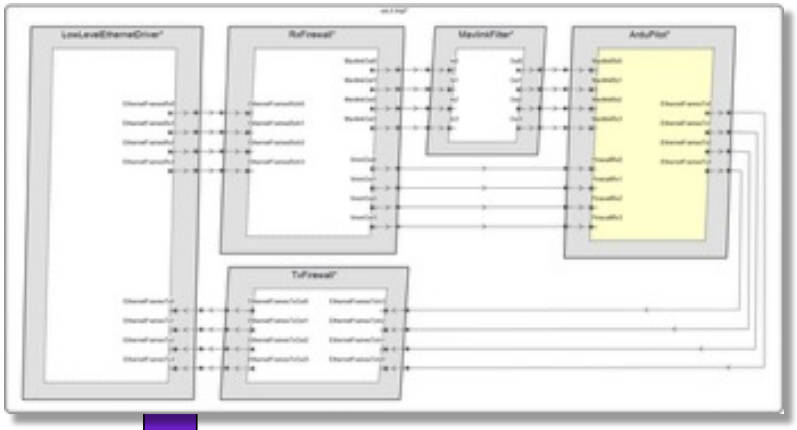
HAMR Code Generation for seL4 -- a **bunch of artifacts for building and deploying**

Critical!
HAMR guarantees that deployed system has **EXACTLY the partitioning and information flow properties reflected in the model**



seL4 microkernel guarantees partitioning of components and communication (backed by computer-checked proofs)

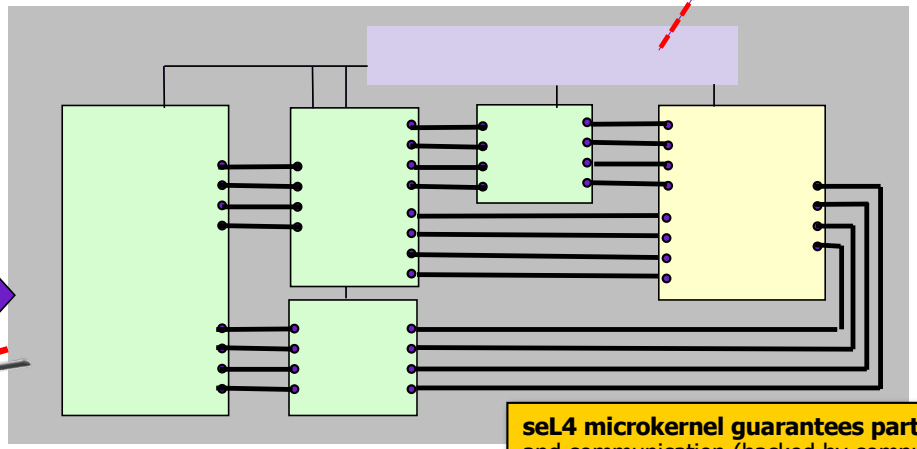
HAMR Generates Scheduling Infrastructure (Time Partitioning)



```
<domain_schedule>
  <domain name="domain_1" length="10" />
  <domain name="domain_2" length="50" />
  <domain name="domain_1" length="10" />
  <domain name="domain_3" length="50" />
  <domain name="domain_1" length="10" />
  <domain name="domain_4" length="50" />
  <domain name="domain_1" length="10" />
  <domain name="domain_5" length="50" />
  <domain name="domain_1" length="10" />
  <domain name="domain_6" length="50" />
  <domain name="domain_0" length="1550" />
</domain_schedule>
```

Scheduling infrastructure to enforce the time-partitioned execution of components using seL4 MCS

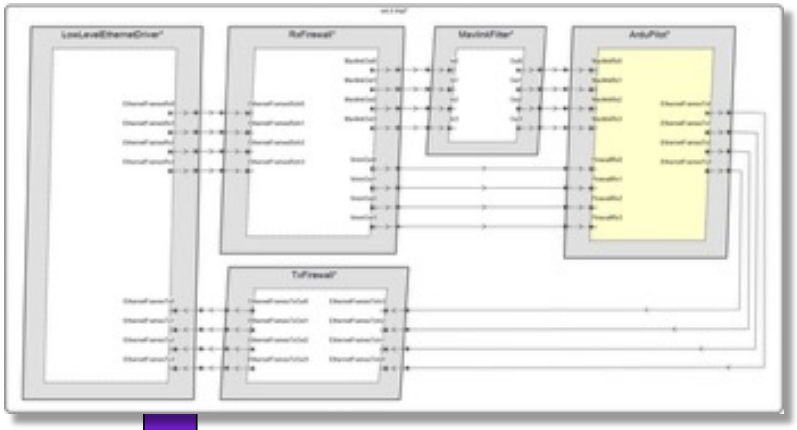
HAMR Code Generation for seL4 guarantees that **deployed system has the partitioning and information flow properties reflected in the model**



seL4 microkernel guarantees partitioning of components and communication (backed by computer-checked proofs)

RASE for seL4 using HAMR

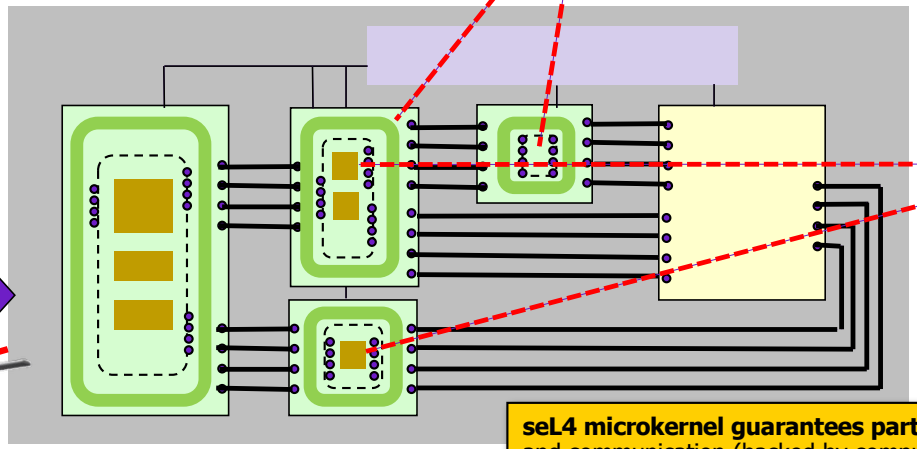
HAMR Auto-generates Support for Component Application Code



To help the developer…

For each thread component, HAMR generates…
- **Application code skeletons** in Rust or C
- **Port APIs** for reading/writing to ports
- **Automated unit testing** infrastructure

HAMR Code Generation for seL4 guarantees that **deployed system has the partitioning and information flow properties reflected in the model**

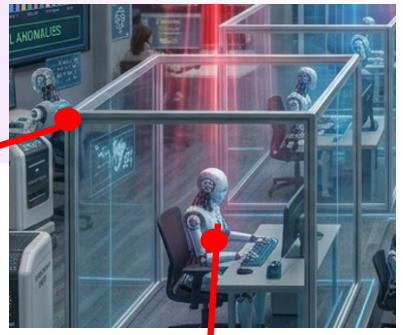


Developer codes application logic, e.g., in Rust, C

seL4 microkernel guarantees partitioning of components and communication (backed by computer-checked proofs)

RASE for seL4 using HAMR

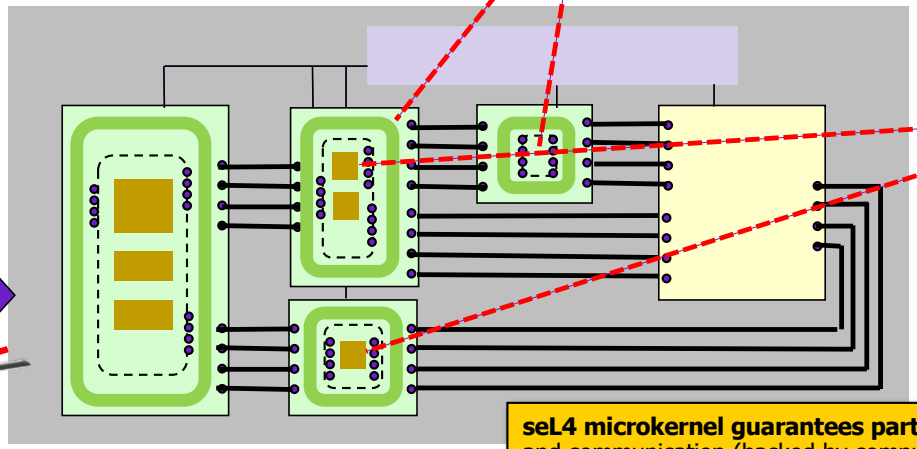
HAMR Auto-generates Support for Component Application Code

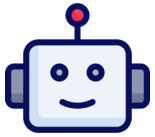


To guardrail the agents…

- Agents are prohibited from modifying the HAMR-generated APIs and infrastructure code
- They only add/modify code for application logic
- Ensures predictability of composition and soundness of the GUMBO contract framework

HAMR Code Generation for seL4 guarantees that **deployed system has the partitioning and information flow properties reflected in the model**





Agents code within the guardrails.

RASE for seL4 using HAMR

seL4 microkernel guarantees partitioning of components and communication (backed by computer-checked proofs)

HAMR Generates Correspondence Information

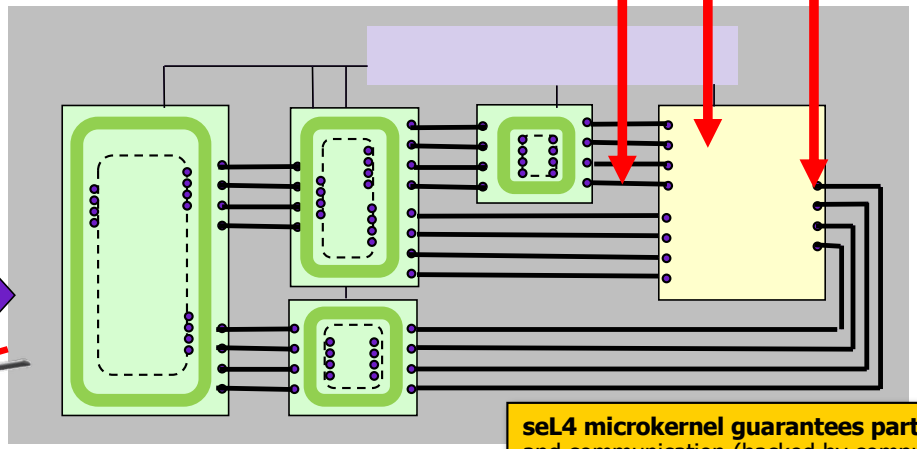
…evidence of no model/code drift

Bi-directional traceability reports document model/code alignment

For each **model** feature,
..corresponding **app code** element
..corresponding **infrastructure code**
..corresponding **kernel** element

..etc., etc.

HAMR Code Generation for seL4 guarantees that **deployed system has the partitioning and information flow properties reflected in the model**

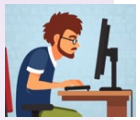


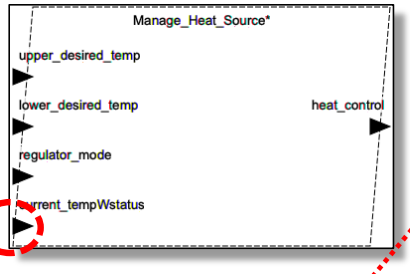
seL4 microkernel guarantees partitioning of components and communication (backed by computer-checked proofs)

RASE for seL4 using HAMR

Auto-generated System Feature Traceability for Manage Heat Source Port (GitHub Markdown)

HAMR auto-generates traceability reports, e.g., for a port – relationships between **model, code, kernel** artifacts





(Model) SysMLv2 port declaration

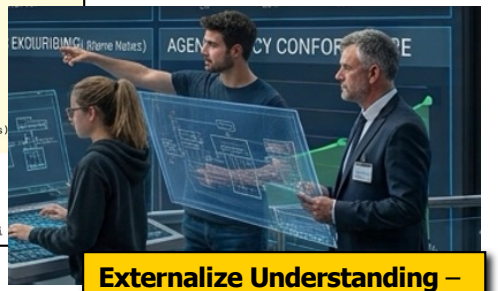
```
_Source extends Abstract_Definitions::Periodic_Thread
500          features
501            --
502            --
503            curr
```

(Code) HAMR-generated Rust API/implementation

```
Code  Blame  170 lines (152 loc) · 6.74 KB
123       }
124       pub fn get_current_tempWstatus(&mut self) -> (res : Isolette_Data_Model::TempWstatus_i)
125         ensures
126           old(self).current_tempWstatus == self.current_tempWstatus,
127           res == self.current_tempWstatus,
128           old(self).lower_desired_temp == self.lower_desired_temp,
129           old(self).upper_desired_temp == self.upper_desired_temp,
130           old(self).regulator_mode == self.regulator_mode,
131           old(self).heat_control == self.heat_control
132       {
133         self.api.unverified_get_current_tempWstatus(&Ghost(self.current_tempWstatus))
134       }
135     }
136
137     pub struct thermostat_rt_mhs_mhs_Initialization_Api;
138     impl thermostat_rt_mhs_mhs_Api for thermostat_rt_mhs_mhs_Initialization_Api {}
139     impl thermostat_rt_mhs_mhs_Put_Api for thermostat_rt_mhs_mhs_Initialization_Api
```

- **APIs**

Port Name	Direction	Kind	Payload	
current_tempWstatus	In	Data	Isolette_Data_Model::TempWstatus.i	Memory Map -> Rust API
lower_desired_temp	In	Data	Isolette_Data	
upper_desired_temp	In	Data	Isolette_Data	
regulator_mode	In	Data	Isolette_Data	
heat_control	Out	Data	Isolette_Data	



Externalize Understanding – of relationship between
- blueprint layers
- blueprints and code

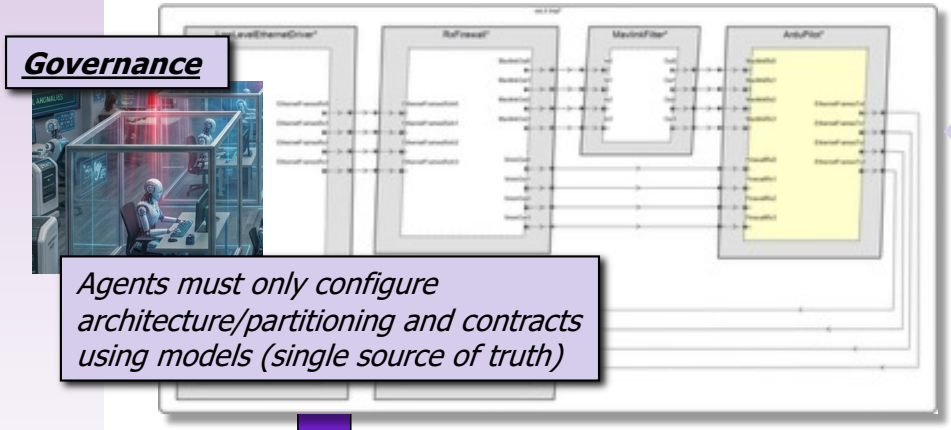
main INSPECTA-models / isolette / hamr / microkit / microkit.system

Code Blame 530 lines (474 loc) · 21 KB

```
104         />
105         <map mr="Isolette_Single_Sensor_Instance_temperature_sensor_cpi_thermostat_
106              vaddr="0x10_004_000"
107              perms="r"
108              setvar_vaddr="current_tempWstatus_queue_1"
109         />
110       </protection
111     </protection_d
```

(seL4 Microkernel) Declaration/protection of storage for port

HAMR Model / Code / Executable Alignment



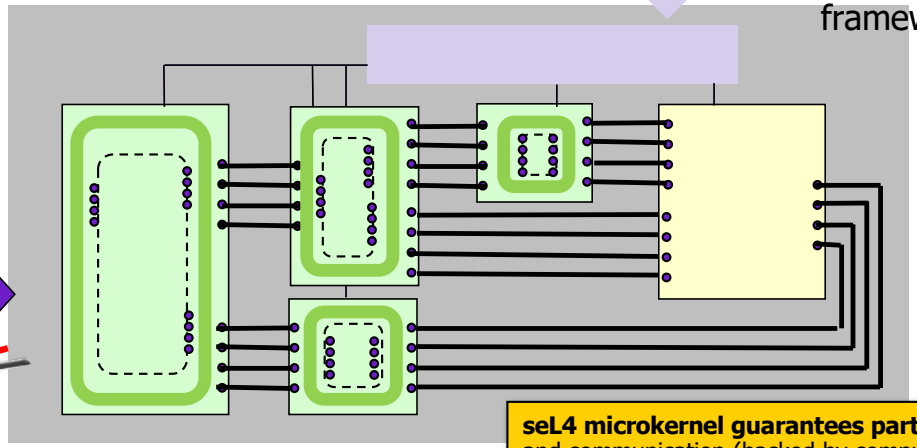
Governance

Agents must only configure architecture/partitioning and contracts using models (single source of truth)

Model / Code Alignment

HAMR Code Generation for seL4 guarantees that **deployed system has the partitioning and information flow properties reflected in the model**

RASE for seL4 using HAMR



seL4 microkernel guarantees partitioning of components and communication (backed by computer-checked proofs)

Evidence of "no drift"…

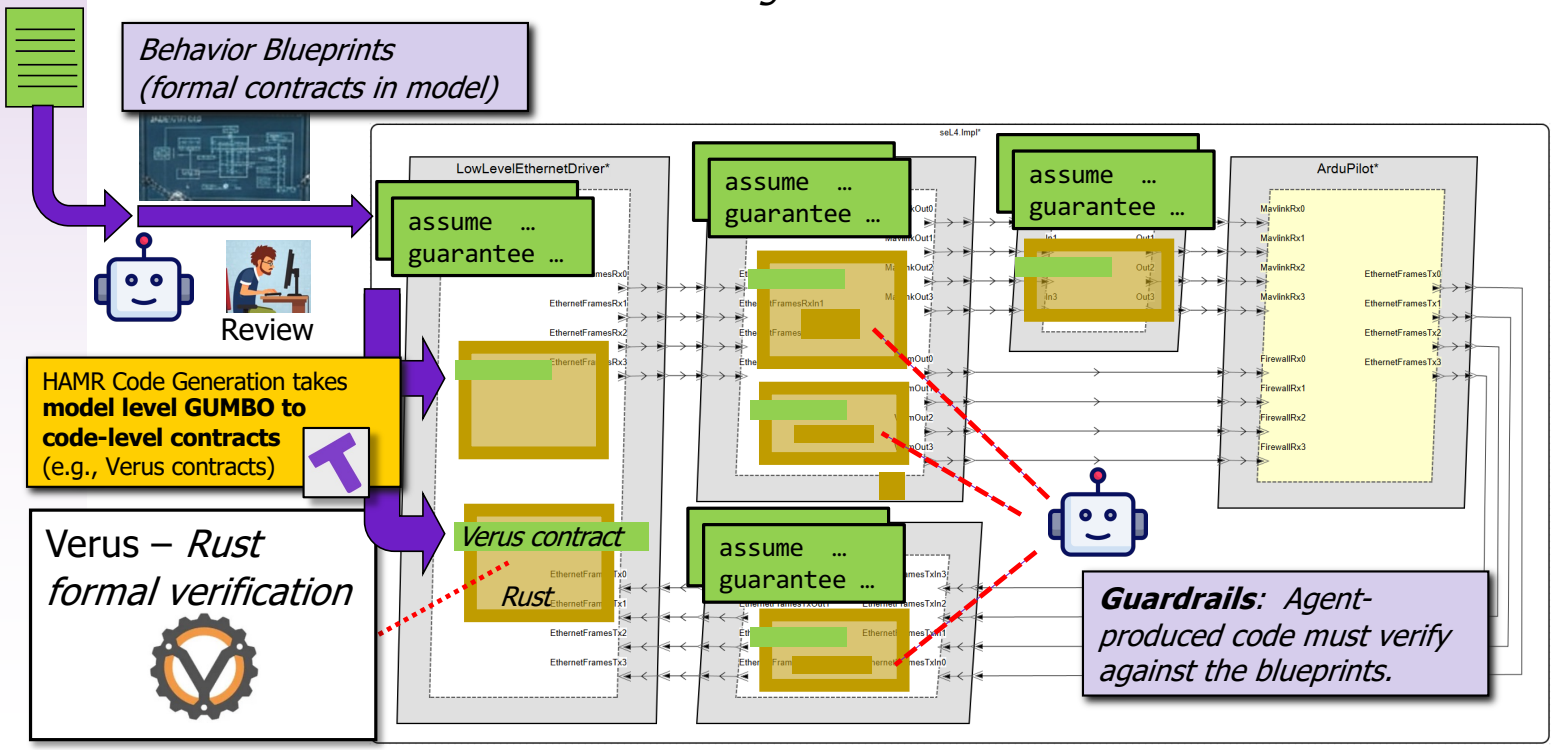
- HAMR generates **cryptographic alignment certificates**
- Will signal **violations** when..
  - generated kernel partitioning, infrastructure code, contracts have been changed "**behind the scenes**"
  - **models changed** without re-running HAMR code generation
- Enforced by **Life-cycle Attestation** framework (Univ of Kansas)

GUMBO - Behavioral Blueprints via Multi-Level Contract Framework

Formal specification, testing, and verification of functional properties is coordinated using GUMBO contracts

Requirements

...Linking model-level contracts to Verus code-level contracts

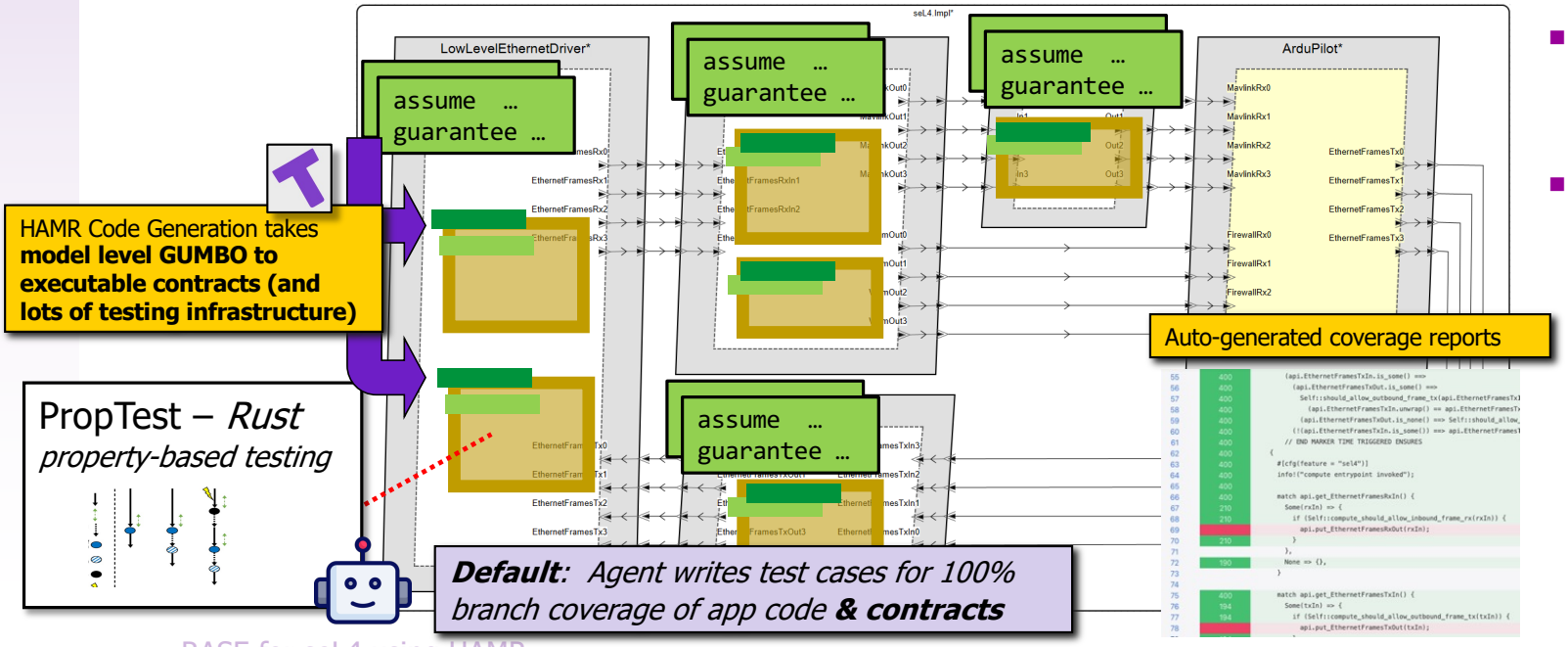


- GUMBO – Model-level contracts
- Programming language independent
- **Model-level contracts are automatically translated and woven into code**

GUMBO - Multi-Level Contract Framework

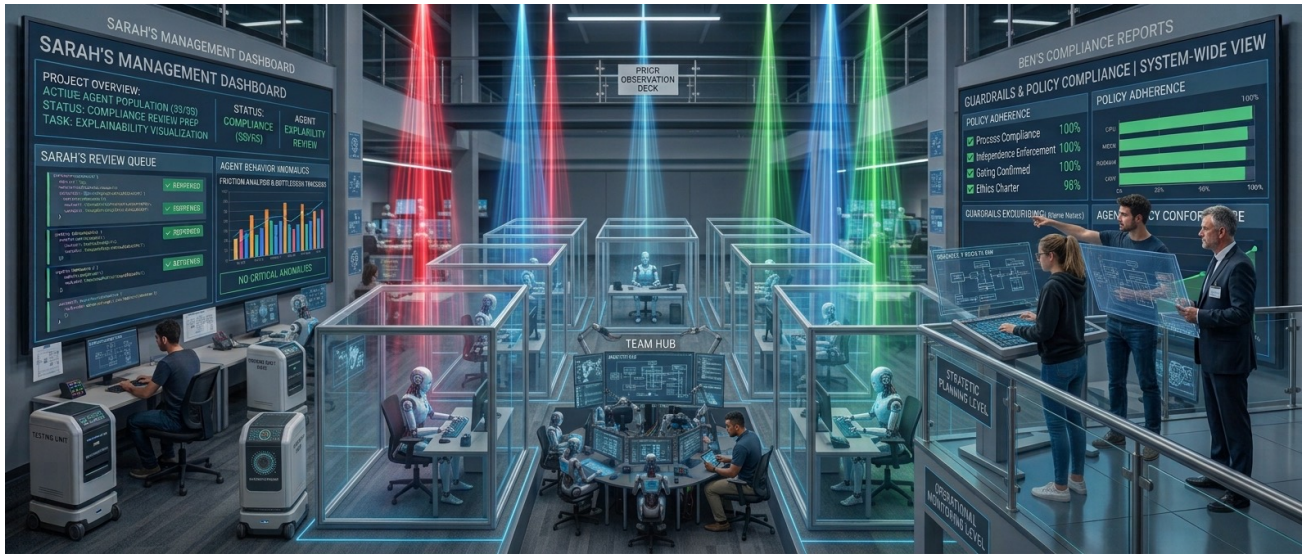
Formal specification, testing, and verification of functional properties is coordinated using GUMBO contracts

***HAMR Theme**: every formal functional spec is also executable/testable!*



- GUMBO – Model-level contracts
- **Model-level contracts also translated into executable contracts (bool functions)**
  - Property-based Randomized Testing
  - Run-time monitoring

Towards a Repeatable and Auditable Engineering Process

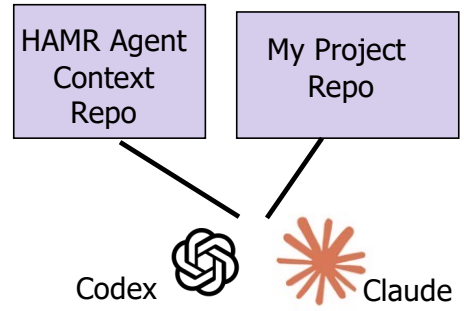


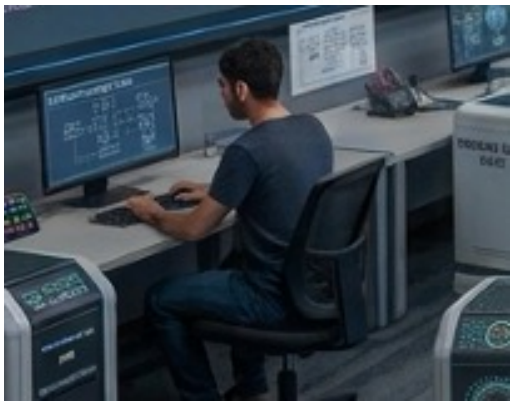
- How do developers "prompt" the agents into action in a consistent way?
- How do we coordinate handoffs between developers and agents? (e.g., stopping for human reviews and sign offs)
- How do we track planned and completed agent work as documented "work units"?
- How do we spot friction in agent activity and improve our framework over time?

HAMR Agent Context Repo

Team-shared curated respository of context material to support HAMR agentic development

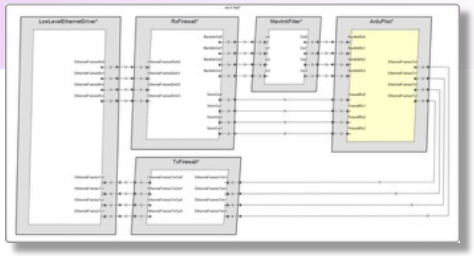
- General documention of HAMR workflows and tooling
- Micro-examples covering model and contract syntax
- Agent citable development guidelines, patterns, anti-patterns
- MCP and CLI usage of HAMR tools
- Semi-formal specification of HAMR workflow (with skill library for driving the development process)
- Skill library – Administration – generating transcripts and tutorials, experiment set up, audits and friction assessment

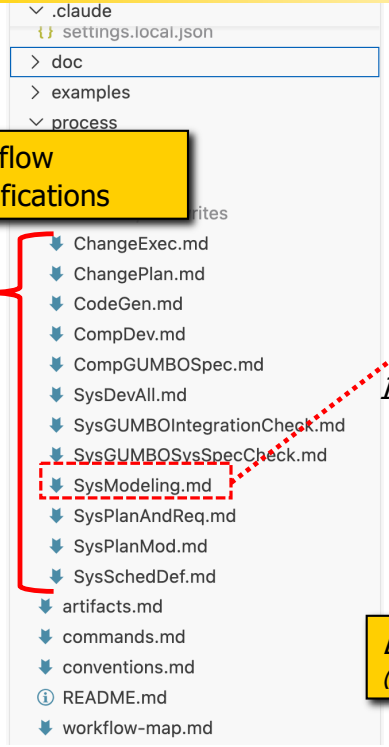




Agent Workflow Steps Specified in HAMR Agent Context

HAMR Agent Context includes workflow specifications
(context info automatically included in HAMR RASE projects)





Example (outline only): *Developing a SysMLv2 HAMR model that is well-formed, aligned with Requirements, aligned with HAMR developer guidance*

```
workflow SysModeling:
  1   do project-structure
  2   do data-types
  3   do components-ports
  4   do system-assembly
  5   until type-check clean:
          fix at 2-4, re-run 5
  AP-1  architecture-review
          on architecture gaps   -> revise 3/4, re-run 5
          on requirements wrong  -> escalate SysPlanAndReq, re-run 5
```

Iteration

Incrementally build (update) model, apply type-checking tool

`# sireum_hamr_sysml_tipe`

Agent audits + iteration points (when necessary)

Developer Audit Point
(review and approve architecture)

(This text is an agent-produced summary of the actual definition shown on the following slides)

Example Workflow Steps - System Modeling (excerpts)

Note: Understand this as structured *natural language script* that an agent follows

SysModeling.3 — components-ports

Define thread part defs with ports, dispatch protocol, period, and implementation language, plus their wrapping process definitions (as required for HAMR sel4 code generation)

SysModeling.4 — system-assembly

Define the top-level System with process instances, port connections, processor definition, processor bindings, and scheduling domain assignments.

- How: @doc/sysmlv2-gumbo-quick-reference.md §6–8 and @doc/modeling-tips.md (flat architecture required; domains start at 2)
- Reads: SysReqs | Writes: SysModel

ssignments.

Development prompt for current step

Specific references to documentation sections that provide guidance

Specific artifact reads and writes (guardrails)

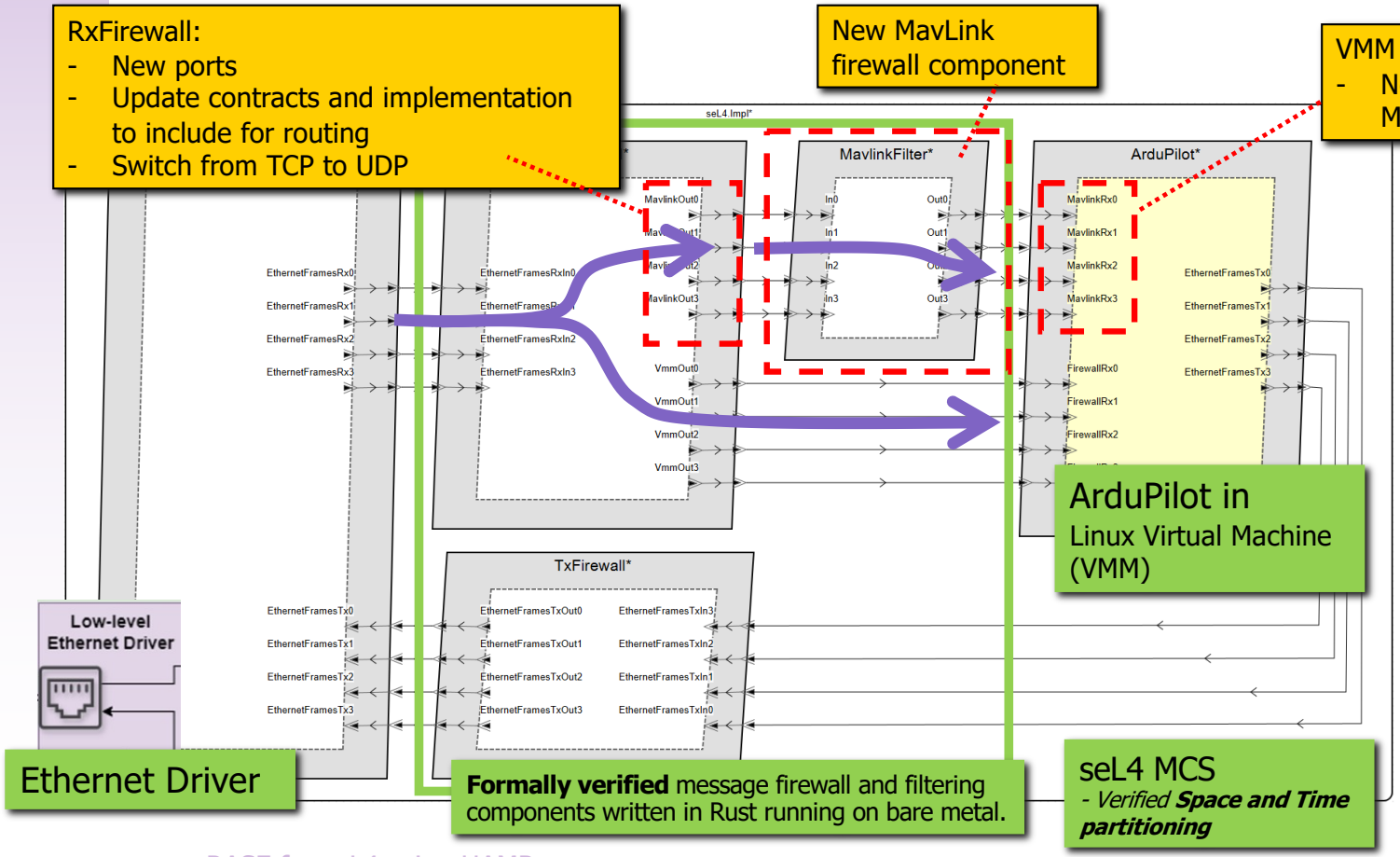
SysModeling.5 — type-check

Run HAMR type-checking and fix reported errors; iterate until clean.

- How: `sireum hamr sysml tipe` CLI (interim: known-gaps.md Gap 3; invocation in @doc/mcp-tools.md §2.4) — sourcepath conventions in @doc/mcp-tools.md
- Reads: SysModel | Writes: SysModel

HAMR Tool Use (model type/well-formedness checking)

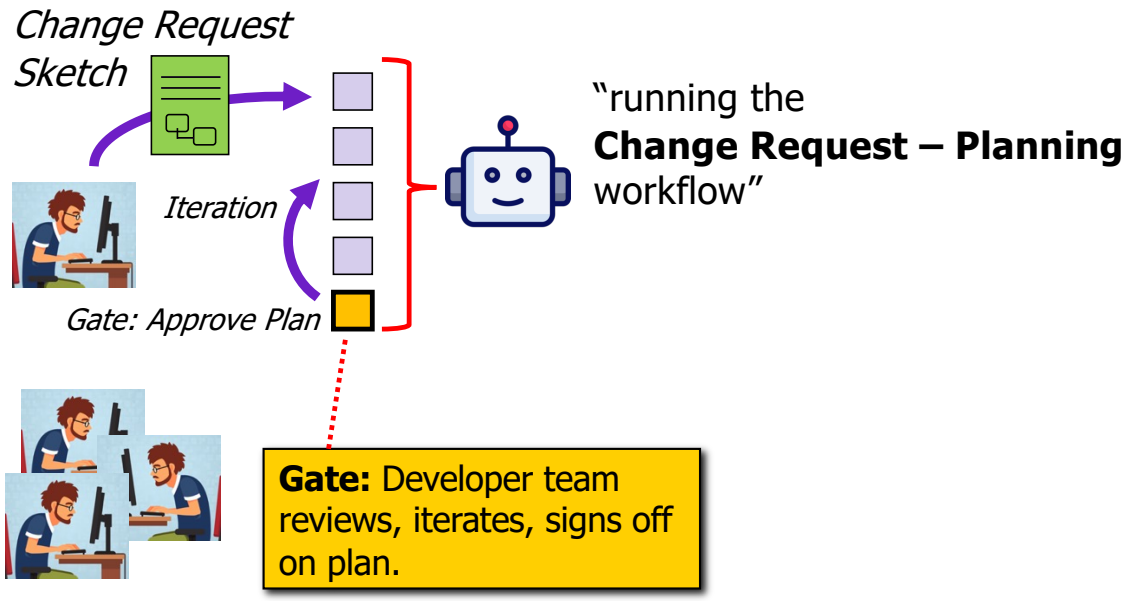
Example: Controlled Experiment to Add MavLink Firewall



- Update ConOps
- Update Requirements
- Update/new model elements
- Update/new model contracts for components
- Regenerate system build
  - New protection domain
  - Updated static schedule
- Code component app logic
- Testing to 100% branch coverage of application code
- Rust/Verus verification of application code to contracts
- All traceability, audit reports

Original Developer+HAMR:
~25 Hours

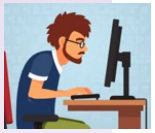
Bird's Eye View of Workflow Step Structure



Each block represents a specified, logged and auditable step in HAMR RASE workflow framework

Input: Change Request Sketch

Core of the developer-authored *sketch* is less 20 lines of text



Suggested approach

RxFirewall

To identify network packets bound for Ardupilot and facilitate identifying single messages, the Ardupilot VM was updated to use UDP instead of TCP. A packet is bound for Ardupilot if it is UDP, the source port is 14550, and the destination port is 14562.

The Rxfirewall should no longer allow any TCP packets through. If a UDP packet is bound for ardupilot, then it should be routed to the new Mavlink firewall component. The previously allowed packets should still route directly to the VMM (Ardupilot component).

Mavlink Firewall

The mavlink firewall should parse the payload of the UDP packet, since it is a single, serialized mavlink message.

Use `mavlink_spec/Packet Serialization _ MAVLink Guide.html` as the specification for how Mavlink messages are serialized and should be deserialized. Handle both versions of the Mavlink protocol.

Disallow messages that are malformed. Disallow messages that can update the ardupilot firmware.

We are using the ArduPilotMega Dialect, which means all of the following message definitions need to be considered:

- `mavlink_spec/ardupilotmega.xml`
- `mavlink_spec/common.xml`
- `mavlink_spec/standard.xml`
- `mavlink_spec/minimal.xml`

All allowed messages should be routed to the VMM.

VMM

The VMM needs to be updated to read the network packets coming from the Mavlink firewall.

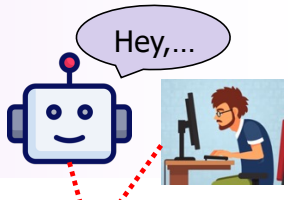
Rx Firewall (change): Indicate that VMM now uses UDP instead of TCP (new policy: drop TCP messages), concept of routing MavLink messages to the new firewall, while others go directly to VMM.

Mavlink Firewall (new): Parse payload of UDP packet as a single MavLink message (provide the MavLink documentation). Handle both versions of Mavlink protocol. Informal description of policy:

- disallow messages that are malformed
- disallow messages that can update the firmware
- forward all allowed messages to VMM

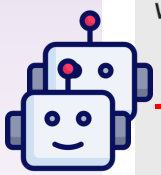
VMM (change): Add new ports to receive input from the MavLink firewall

Agents Transforms Change Request Sketch Into Engineering Plan



Using one of RASE's *Elicitation* workflows, an agent will "interview" the developers if more information is needed to produce a coherent plan.

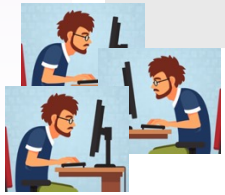
Example (outline only): *Preparing a Engineering Plan for a change*



```
workflow ChangePlan(cr):                                  # sketch -> reviewable change plan;
                                                          #   execution is ChangeExec
  1  do intake-sketch                                     (elicit ChangeScope)  # archive verbatim
  2  do baseline-survey                                   # pin commit + project state
  3  do consistency-check                                 # sketch internal + against baseline
       on blocking inconsistency -> clarify with developer, re-run 3
  4  do impact-analysis                                   # impact map + non-impact argument
  5  do wave-decomposition
  6  do write-plan                                        # action-requests/CR-<NN>-<slug>/change-plan.md
 AP-1  plan-approval                                      # may be external: committed-plan review
       on review feedback -> revise 4-6, re-present
```

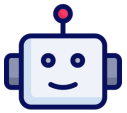
Agents perform *impact analysis* (what parts of the system and assurance will be updated)

Agents suggest options for *waves* (decomposition work into reviewable phases)



Gate: Developer team reviews, iterates, signs off on plan.

Agent-Produced Plan Waves (excerpts)



(1 of 10 sections of plan – 215 lines)

6. Waves (§5)

Wave	Intent	Invocations	Expected artifact deltas	Verification gate
W1	Establish requirements, architecture, contracts, and generated scaffold.	Delta `SysPlanAndReq`; delta `SysModeling`; `CompGUMBOSpec(component=RxFirewall)`; `CompGUMBOSpec(component=MAVLinkFirewall)`; `SysGUMBOIntegrationCheck`; `CodeGen`.	New requirements documents; RxFirewall/MAVLinkFirewall/ArduPilot model and connection deltas; generated MAVLinkFirewall and affected bridge scaffolds.	Requirements review complete; SysML tipe clean; integration checks passing and non-vacuous; generated tree reviewed with no LowLevelEthernetDriver/TxFirewall semantic delta.
W2	Implement and verify receive routing and MAVLink policy without regressing TxFirewall proofs.	`CompDev(component=RxFirewall)`; `CompDev(component=MAVLinkFirewall)`; direct development/testing of `firewall_core` and a separate MAVLink core as dependencies; `CompDev(component=TxFirewall)/VerifyOnly`.	RxFirewall, MAVLinkFirewall, core libraries, unit/property/GUMBOX tests, affected-component coverage/Verus evidence, and refreshed TxFirewall verification evidence with no TxFirewall source delta.	RD-1 policy encoded; valid `FILE_TRANSFER_PROTOCOL` messages pass; firmware-flash activation commands and malformed v1/v2 frames are rejected; full branch coverage of changed application logic and GUMBOX oracles; RxFirewall/MAVLinkFirewall Verus clean; TxFirewall verification remains clean against the changed `firewall_core`.
W3	Integrate the VMM and static schedule.		VMM receive-path changes; `microkit.schedule.xml` new domain slot; any necessary build glue.	Full system builds; schedule validation passes; four MAVLinkFirewall outputs reach the corresponding VMM inputs; existing transmit path remains

Expected updates to each category of artifacts

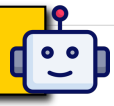
W1: Requirements, models, contracts, HAMR code generation

Wave IDs and Intent

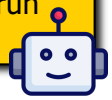
W2: Component implementations, testing, and verification

W3: Update VMM, integrate, new schedule

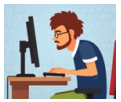
Workflow steps that agents will follow.



Recognition that TxFirewall is unchanged but because it also uses the shared core_library with RxFirewall, it should re-test and re-run contract verification.

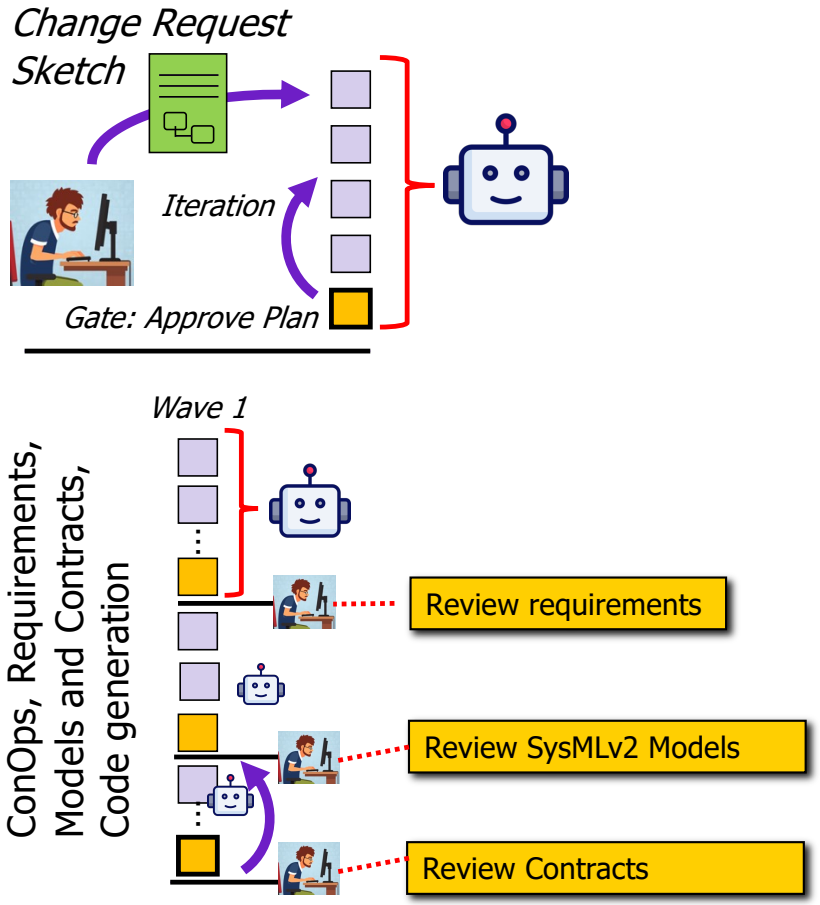


Bro, I've planned. What do you think?



Summary of formal / informal verification and validation gates, development state to be handed off to developers for review.

Bird's Eye View

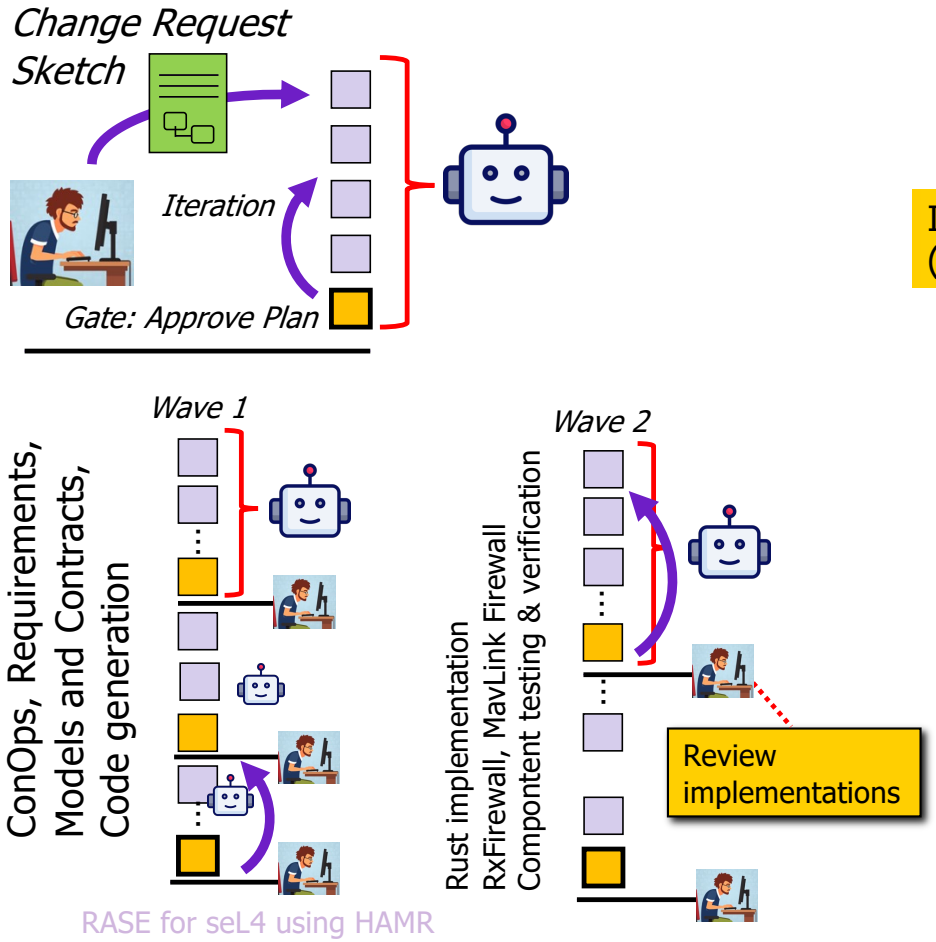


Iteration on contracts (based on developer reviews)

- Contract readability suffered due to absence of helper functions
  - Agent fixed
- Placement (scope) of helper functions (global vs component)
  - Agent fixed
- Abstraction approach of contracts
  - Robbie asked for justification; left as is

…this friction can be removed by improving HAMR agent context style guidelines

Bird's Eye View



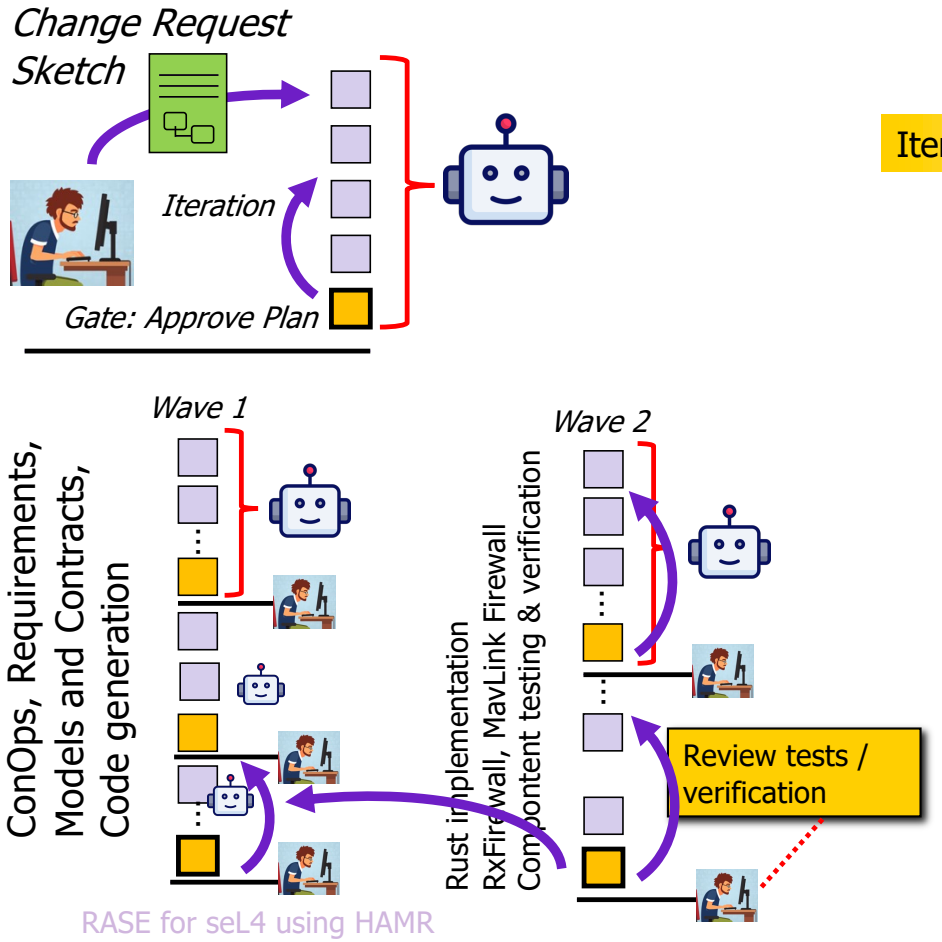
- Agent handling of CRC checks **improved base line**

Iteration on component implementations (based on developer reviews)

- Code readability suffered due to absence of helper functions (hardcoded hex values)
  - Agent fixed
- Component specific firewall logic placed inside of reusable parsing library, instead of client code that used the library
  - Agent fixed

…this friction can be removed by improving HAMR agent context style guidelines

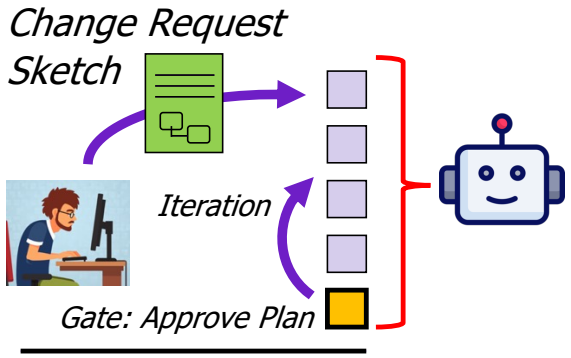
Bird's Eye View



Iteration on component testing / verification

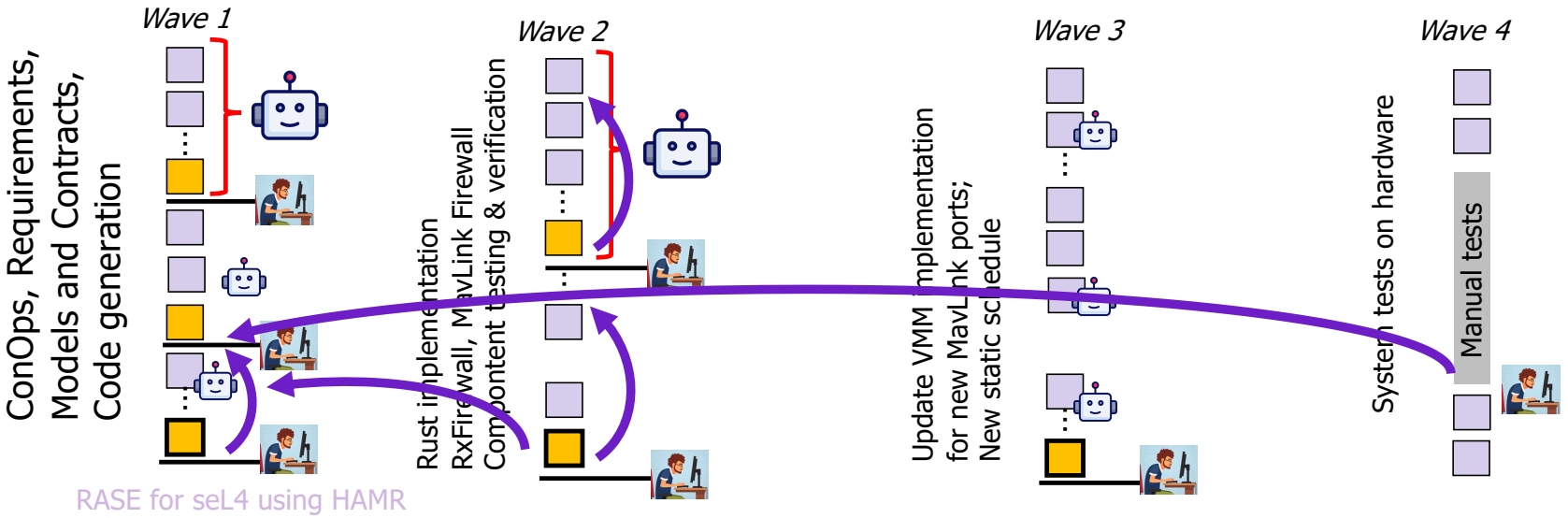
- Agent represented some critical logic using Verus `external_body` (code not checked)
  - Agent fixed
  - Improve: style guidelines; audit tool
- Agent with Verus found possible overflow issue in contract *specification* – caused iteration back to Wave 1
  - Agent fixed
  - Improve: style guidelines; audit tools
- Agent uncovered/fixed several other small issues via verification and testing

Bird's Eye View

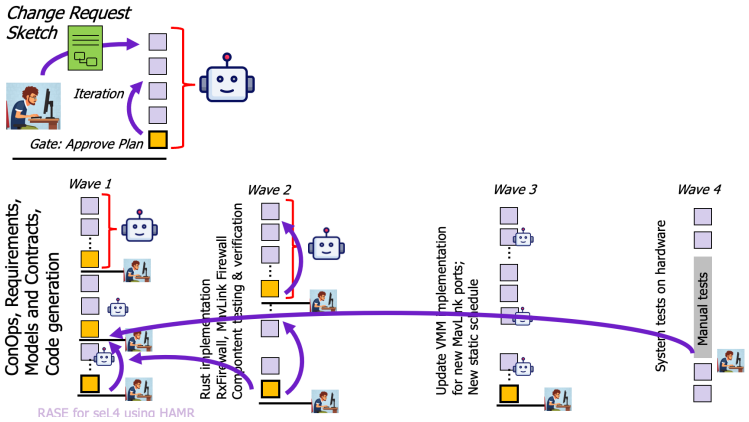


Iteration on system testing…

- (Manual) system testing (validation) reveal problem in specs (requirements, contracts) related to subtle "0 byte truncation" optimization for MavLink messages
  - When provided with manual capture of inter-component traffic; agent fixed



Assessment



- Baseline (developer only) ~ 25 hrs
- Agent supported
  - Agents: 3.75 hrs (~$35 in tokens)
    - Codex - gpt-5.6-sol
    - 124 descrete auditable workflow steps
  - Developer reviews: 4 hrs
- Developer background
  - Expert in HAMR and seL4; novice in Codex (less than 1 week experience)
- Understanding specs, code, technologies still seems critical to avoiding technical & cognitive debt

Next Steps

- Continued experiments – leading to full agentic build of the Open Platform
- Agent automation of system testing and formal verification
- Automated metrics framework
- Automation of assurance case construction
- Building our own agent harness that provides stronger governance, audit, multi-model capabilities

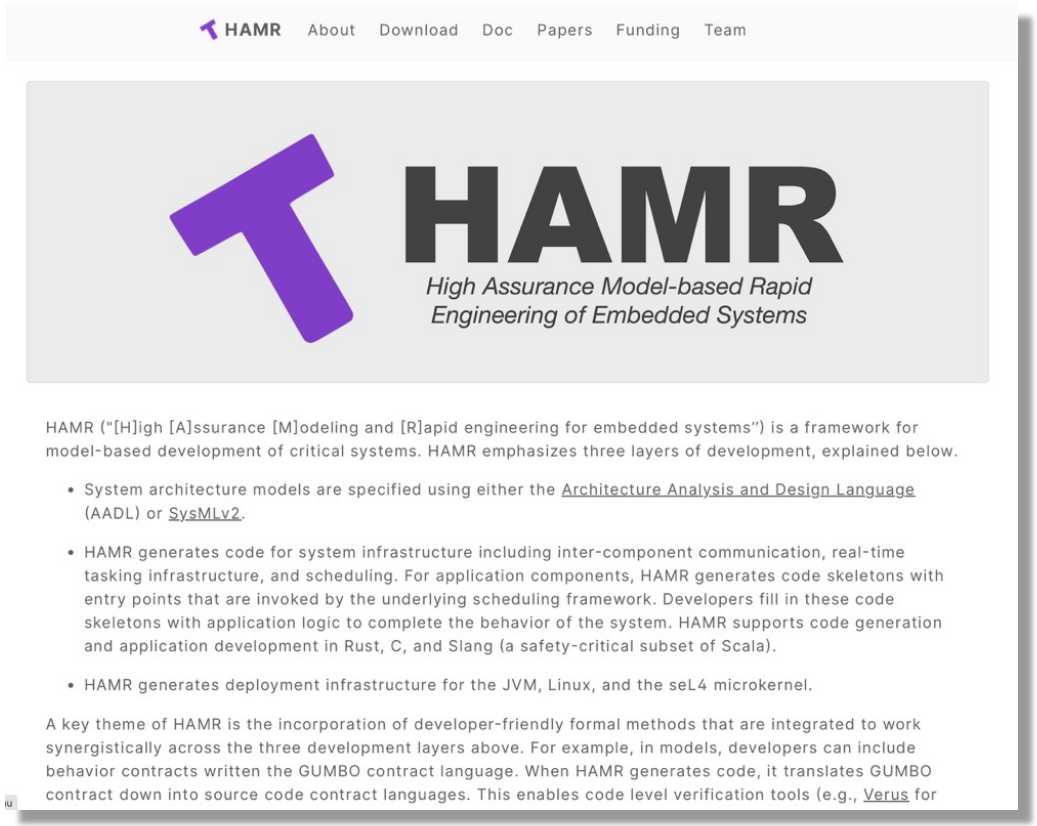
Key Takeaways

As we move into the future..

Human effort will shift
...from **engineering the code**
...to **engineering the development and assurance context**
...to **engineering layered, composable abstractions**
...that **provide direction but also explainability**

- Partitioning architectures (e.g., seL4) are **critically important, but not enough**...
- Also need..
  - Layered, composable, engineering blueprints and abstractions
  - ..that provide guideance but also support understanding and explainability

Website / Documentation



https://hamr.sireum.org

- Documentation
  - SysMLv2 with AADL Libraries
  - GUMBO contract language
  - Rust component development
  - Rust testing and verification
  - seL4 background
- Overview / Demo videos
- Worked examples
- Teaching material