Leverageable Semantics Definitions and Contract Reasoning for a Technical Architecture Description Language

HCSS 2023 – May 8, 2023

Kansas State University

John Hatcliff
Jason Belt
Robby

Software Engineering Institute (Carnegie Mellon University)

Jerome Hugues
Lutz Wrage

Galois

Danielle Stewart
Todd Carpenter
Ryan Peroutka
August Schwerdfeger

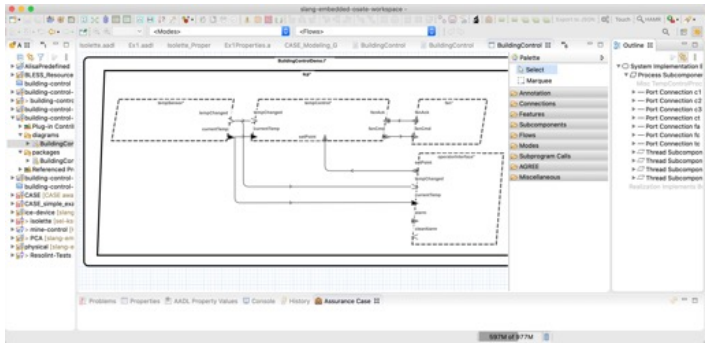
+ collaborators from Collins Aerospace and seL4 team

This material is based on research sponsored in part by US Army and DARPA

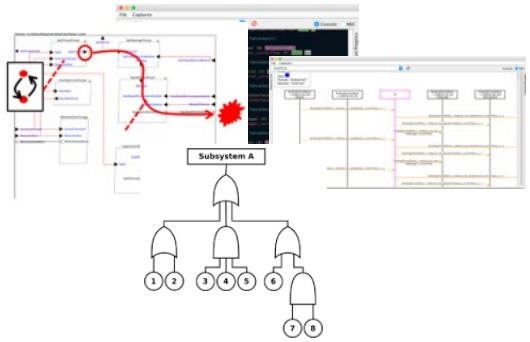
HAMR

HAMR – tool chain for model-driven development of high-assurance embedded systems (from Adventium Labs/Galois and Kansas State)

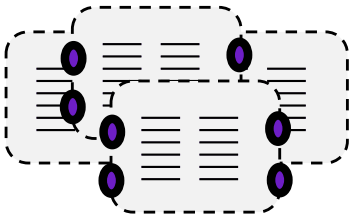
Modeling, analysis, and verification in the **AADL** modeling language



Leveraging analyses from AADL community



Component development and verification in multiple languages



- C
- Slang
  - high integrity subset of Scala
  - contract verification framework
  - translates to C
- CakeML (ML-variant with verified compiler)

Deployments aligned with AADL run-time on multiple platforms

JVM Deployment

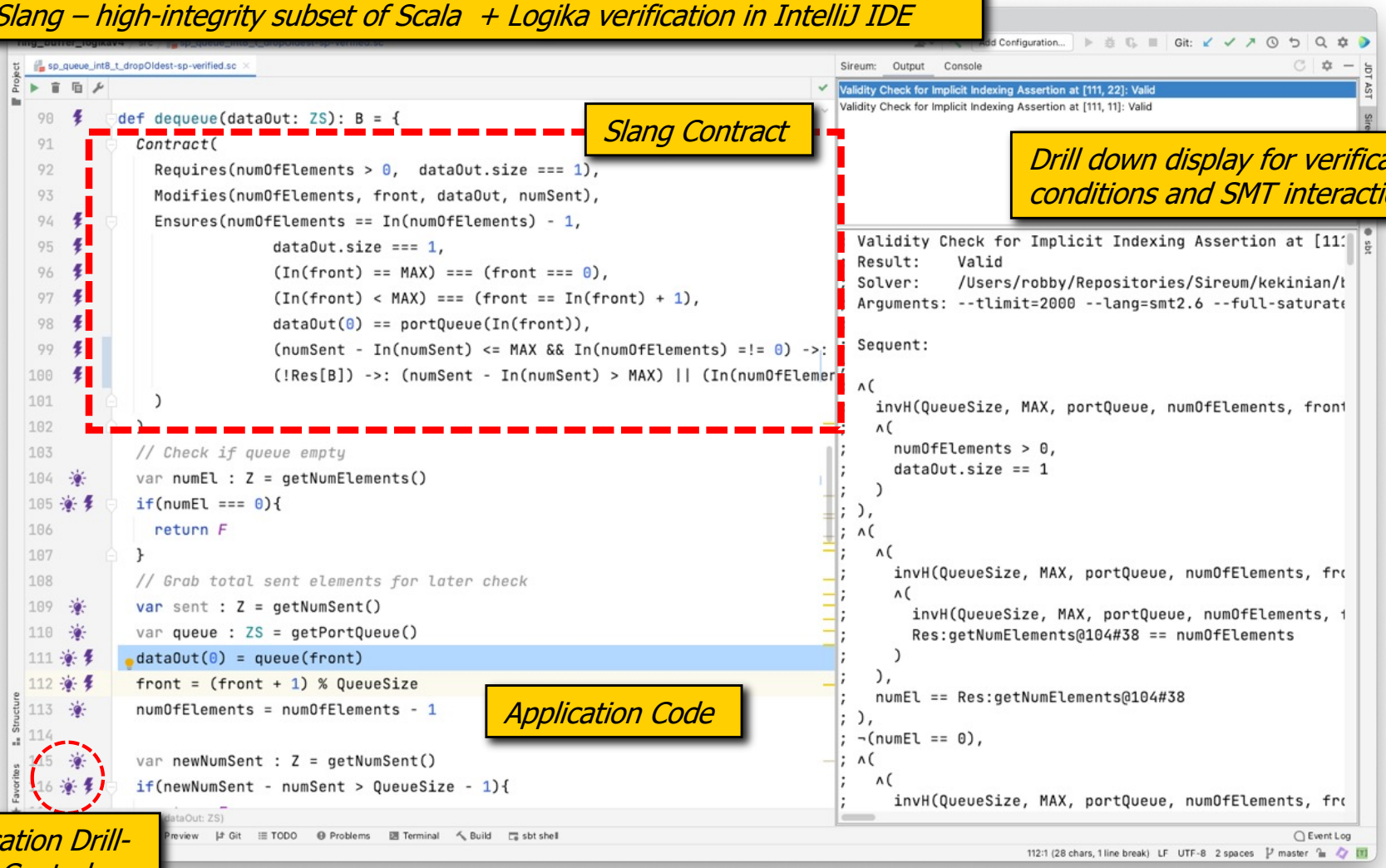
Linux Deployment

seL4 Deployment

Slang Contracts
and Automated Verification via Symbolic Execution (Logika)

DARPA SBIR w/
Adventium (Galois)

Slang – high-integrity subset of Scala + Logika verification in IntelliJ IDE



Slang applications can be integrated with Scala and Java and executed on JVM or transpiled to JS or C. The generated C has bounded memory usage and no garbage collection & compatible with verified CompCert compiler.

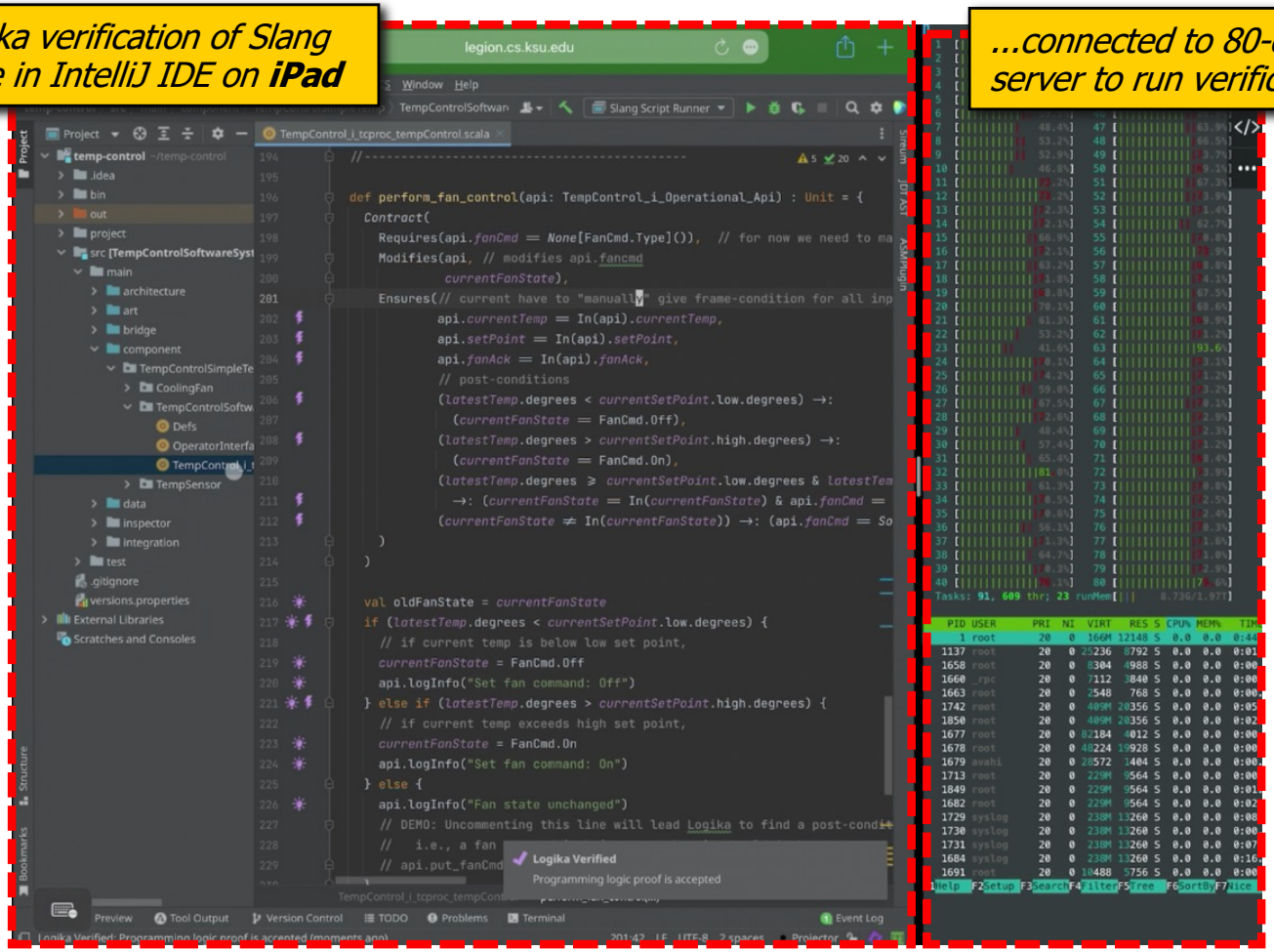
Logika Verification

Featureful, Integrated Capabilities

Logika uses a **server-based architecture** with a suite of SMT solvers (Z3, CVCx, Alt-Ergo), massive parallelization, with "always on" smart incremental checking

*Logika verification of Slang code in IntelliJ IDE on **iPad***

...connected to 80-core server to run verification



From a TCCOE conference demo video of Logika in January 2022

See https://drive.google.com/uc?export=download&id=1vkBNWM8pocSz8jUG-E16zdVleELZr2Sk for Slang / Logika overview talk given at the Trusted Computing Center of Excellence Symposium

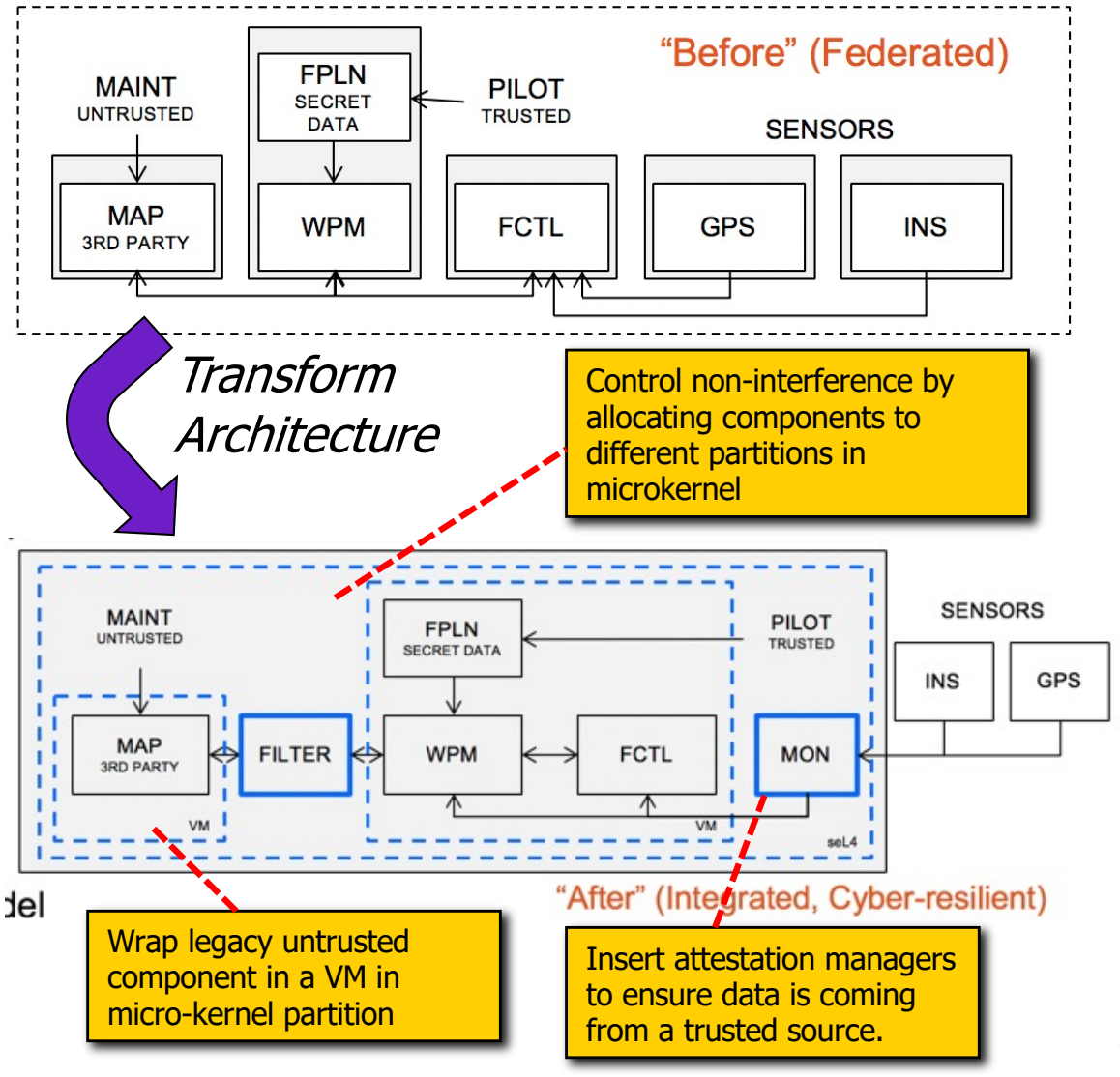
DARPA CASE Approach

HAMR was developed by Kansas State and Galois researchers on a team led by Collins Aerospace (Darren Cofer)

- **Capture requirements** for cyber-resiliency
- **Analyze** design
- **Transform** design
- **Verify new design** against requirements
- **Build / Deploy**

HAMR Focus

seL4 verified micro-kernel technology is a core technology



DARPA CASE Final Demonstration

HAMR used with Collins BriefCASE tool chain adding new functionality to CH-47 mission computing...

COLLINS COMMON AVIONICS ARCHITECTURE SYSTEM (CAAS)

- Use BriefCASE tools to add new functionality to helicopter avionics system with cybersecurity guarantees
- Demonstrated on CH-47 mission computing platform – relevant for Future Vertical Lift

https://www.youtube.com/watch?v=77xCISlJpkk

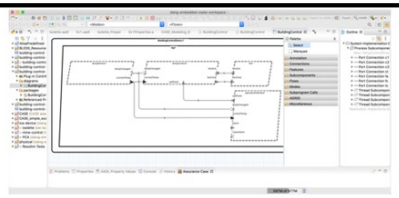
/Released

Collins Aerospace

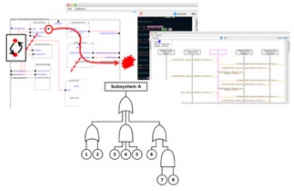
Goal: Semantic Consistency End-to-End

HAMR is supported by a suite of inter-related formal semantics artifacts to aid end-to-end semantic consistency

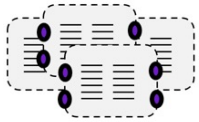
Modeling, analysis, and verification in the **AADL** modeling language



Leveraging analyses from AADL community



Component development and verification in multiple languages



- **Slang (high integrity subset of Scala with a contract verification framework)**
- C
- CakeML (ML-variant with verified compiler)

Deployments aligned with AADL run-time on multiple platforms

JVM Deployment

Linux Deployment

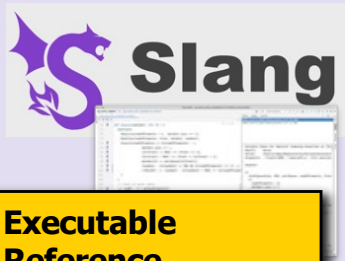
seL4 Deployment

Analysis and verification results moved up and down abstraction layers

Semantic Consistency

Operational Semantics Rules (in Latex)

$$\frac{\begin{array}{ll} ReceiveInputData: & \\ \mathcal{M}.isInPort[t](p) & \\ \mathcal{M}.isDataPort[t](p) & \\ I_t^i(p) = \langle v \rangle & A_t^{i\prime} = A_t^i[p \mapsto \langle v \rangle] \end{array}}{\langle I_t^i, A_t^i, A_t^o, I_t^o, V_t, D \rangle \xrightarrow{\text{RecInP}(p)}_t \langle I_t^i, A_t^{i\prime}, A_t^o, I_t^o, V_t, D \rangle}$$



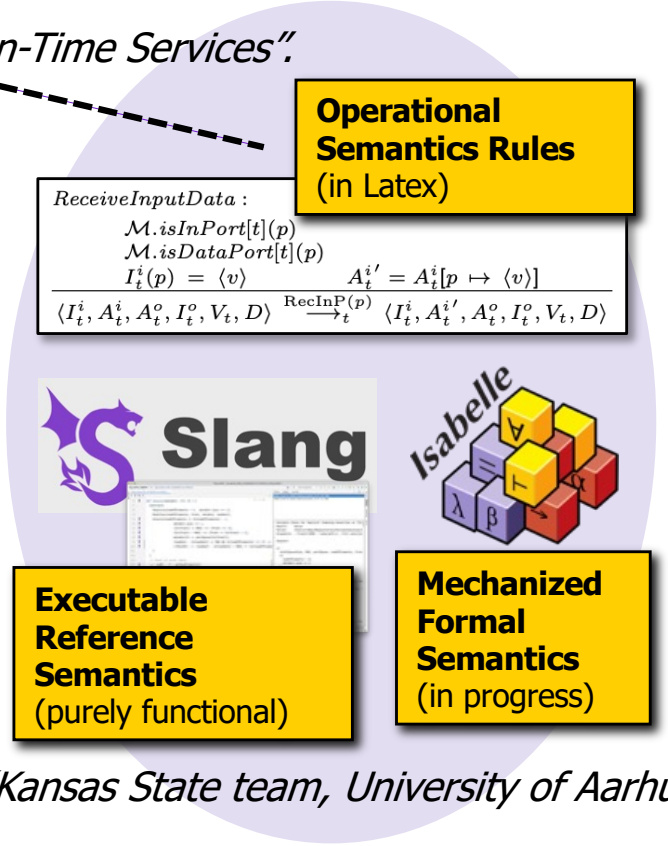
Executable Reference Semantics (purely functional)

Mechanized Formal Semantics (in progress)

Goal: Semantic Consistency End-to-End

HAMR is supported by a suite of interlated formal semantics artifacts to aid end-to-end semantic consistency

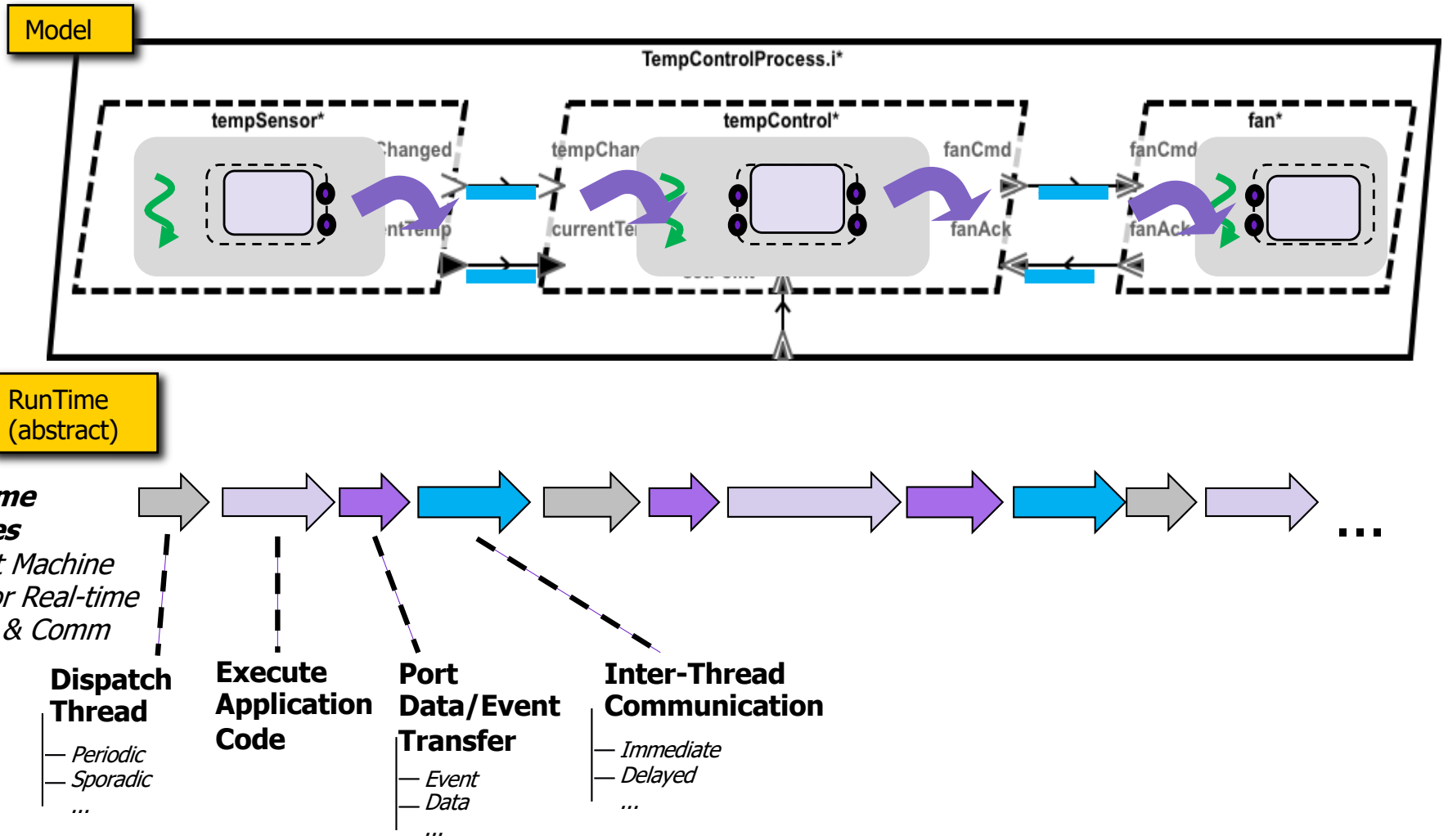
John Hatcliff, Jerome Hugues, Danielle Stewart, and Lutz Wrage.
"Formalization of the AADL Run-Time Services".
(ISOLA 2022)



(Kansas State team, University of Aarhus)

Formalizing AADL Run-Time Services

The formal semantics in this talk focuses on the AADL Run-Time Services...



AADL Standard Description is Informal

Description (excerpts) of the **Receive Input** service in the previous version of AADL standard…

Narrative description..

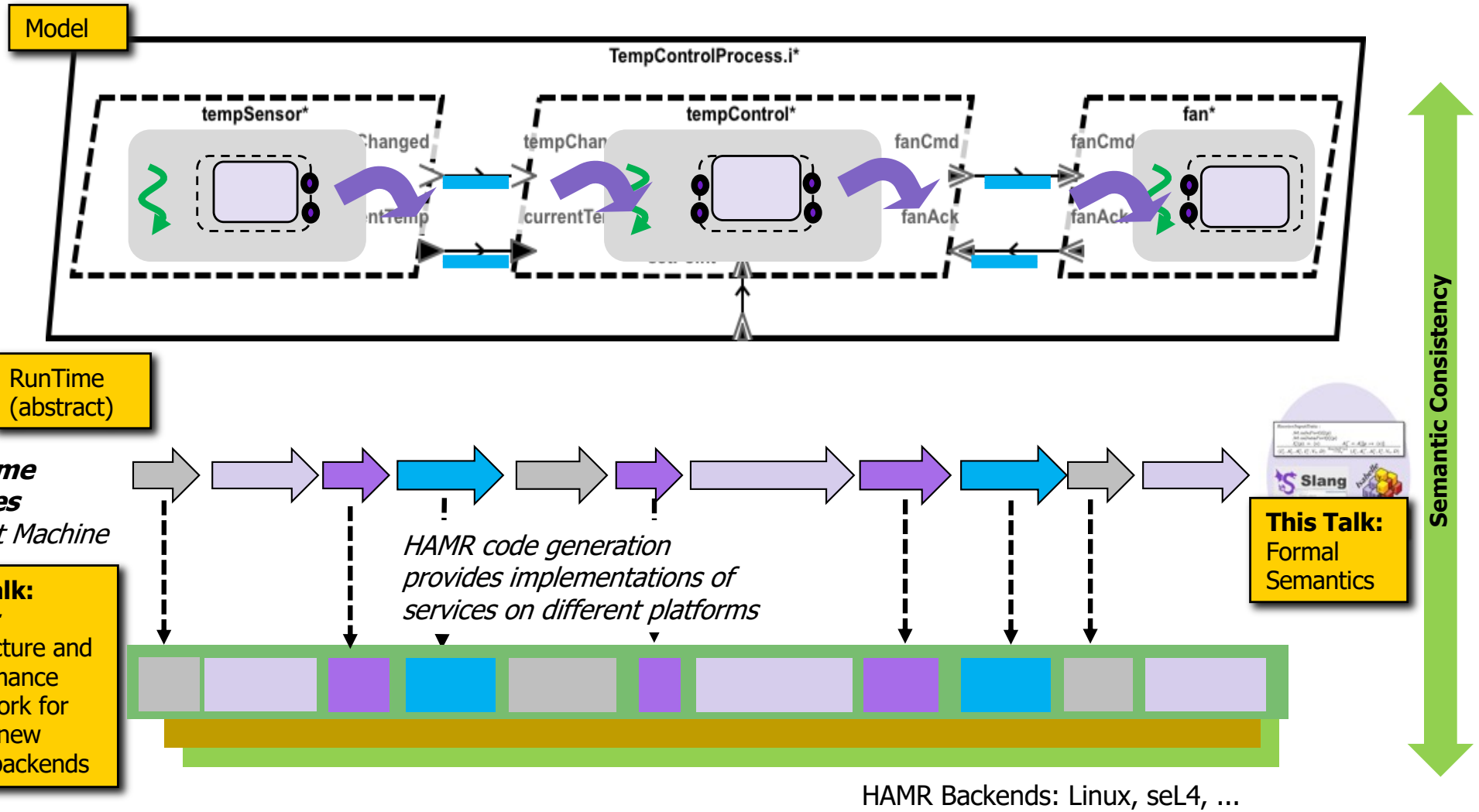
A **Receive_Input** runtime service allows the source text of a thread to explic itly request port input on its incoming ports to be frozen and made accessible through the port variables. Any previous content of the port variable is over-written, i.e., any previous queue content not processed by **Next_Value** calls is discarded. The **Receive_Input** service takes a parameter that specifies for which ports the input is frozen. Newly arriving data may be queued, but does not affect the input that thread has access to (see Section 9.1). **Receive_Input** is a non-blocking service.

```
subprogram Receive_Input
features
   InputPorts: in parameter <implementation-depe
   -- List of ports whose input is frozen
end Receive_Input;
```

High-level API that **omits** specifics of what aspects of thread/system state are updated and the semantics of the updates…

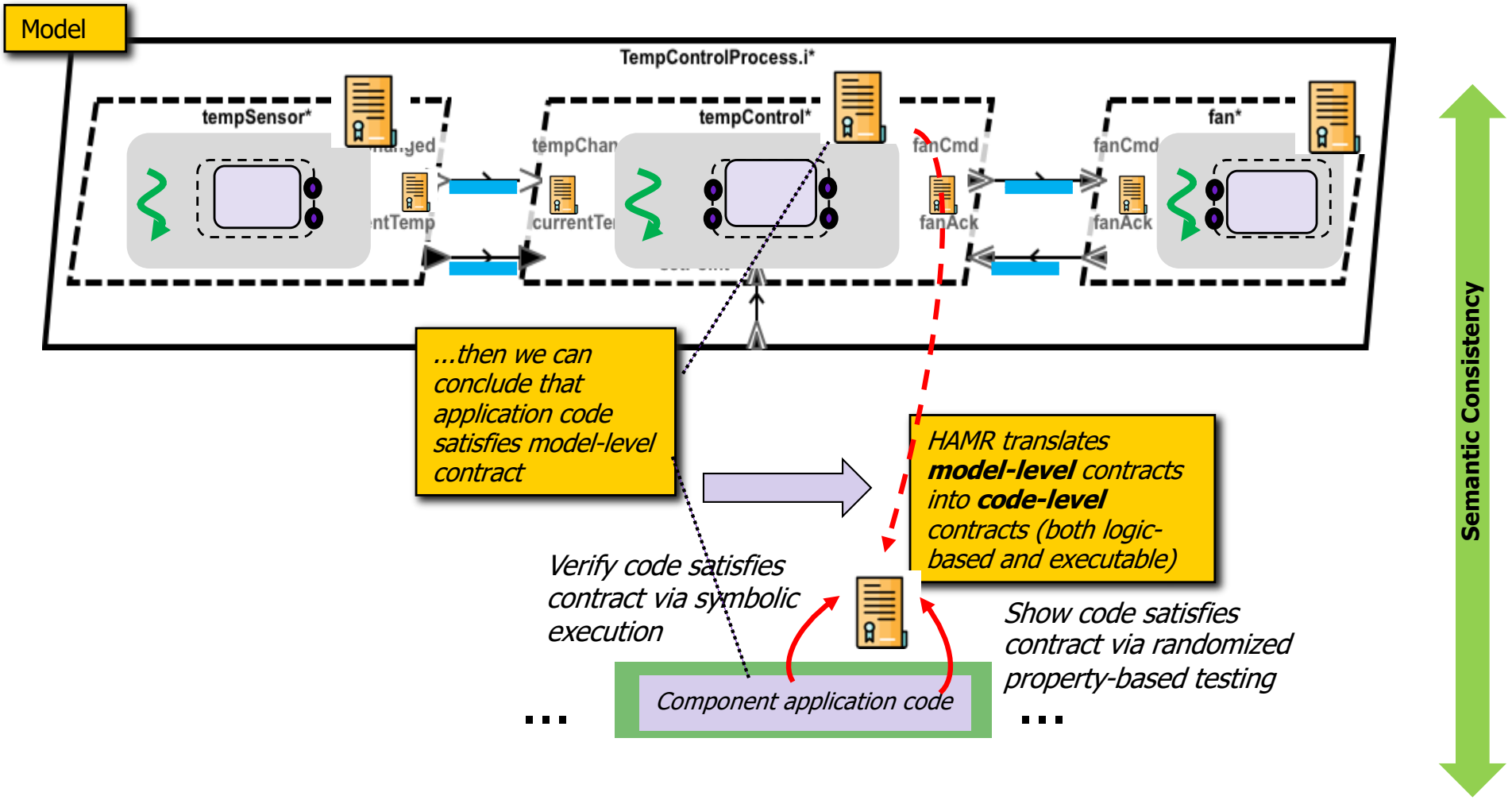
Formalizing AADL Run-Time Services

The formal semantics is a **specification for implementing AADL run-time on different platforms**



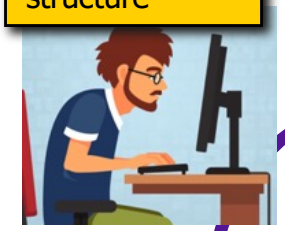
Formalizing AADL Run-Time Services

The formal semantics guides the design of an **integrated model and code-level contract framework** that supports both verification and property-based testing



AADL Modeling Concepts

Developer configures computational structure



AADL Thread Property Options

Periodic
Sporadic
Hybrid
…

+ timing, scheduling constraints, etc.

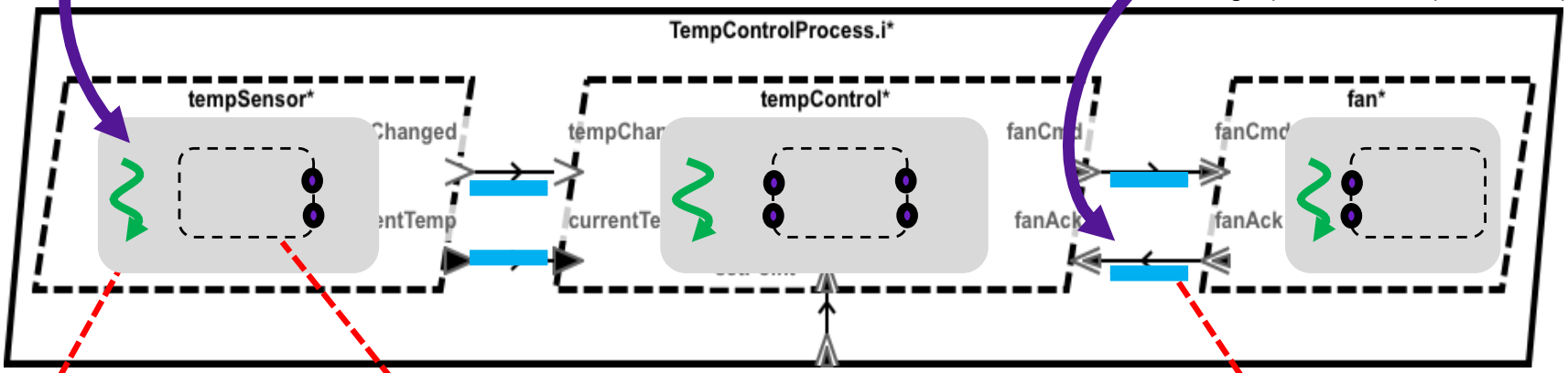
AADL Port & Connection Property Options

Event
Data

Event Data
…

buffered notifications
shared data cells
(or data distribution service)
buffered messages
(message passing middleware)

+ QoS, buffer sizes, latencies, etc



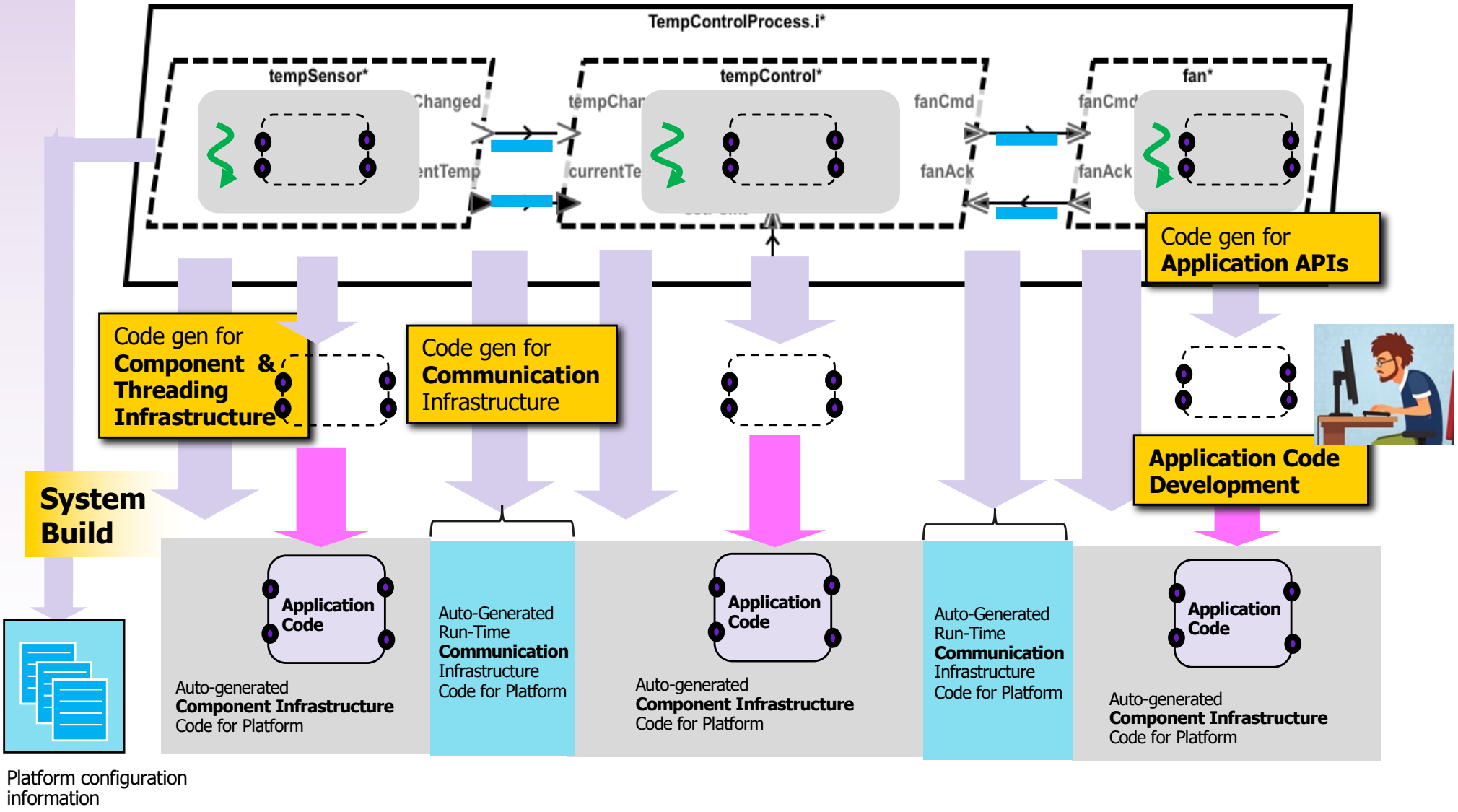
Selected thread pattern

Implied API Pattern for application code to access port communication, etc.

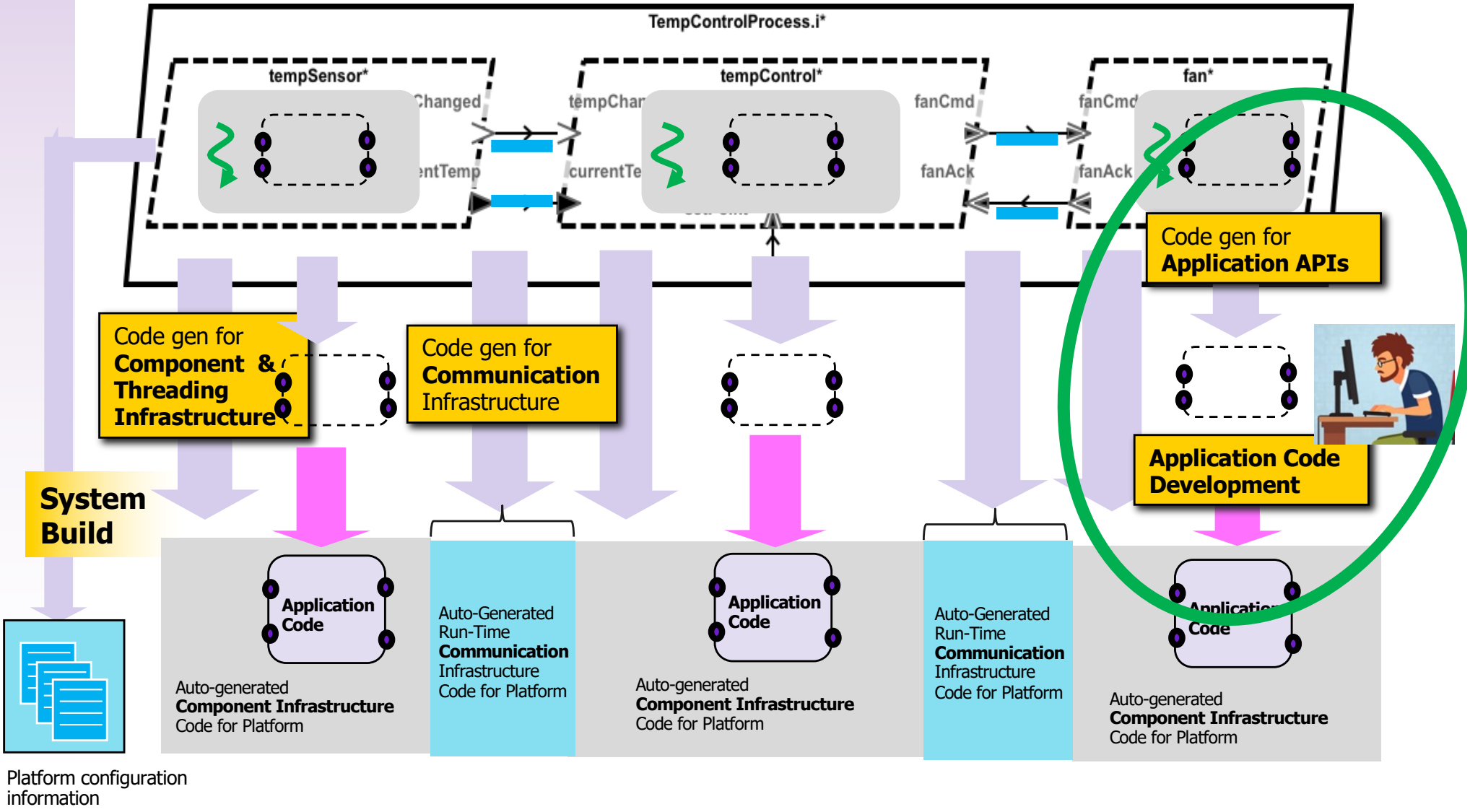
Selected communication pattern

...an AADL contract language should be aligned with these patterns.

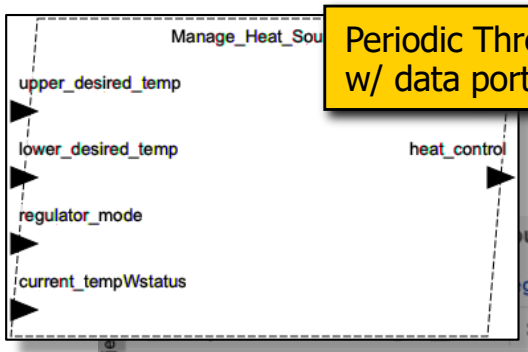
HAMR Code Generation



HAMR Code Generation



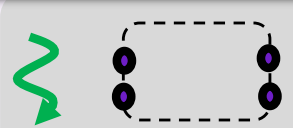
Component Application Code Interfaces Generated from AADL Model



Periodic Thread w/ data ports

...Interfaces/APIs/Skeletons for application code are auto-generated from AADL model

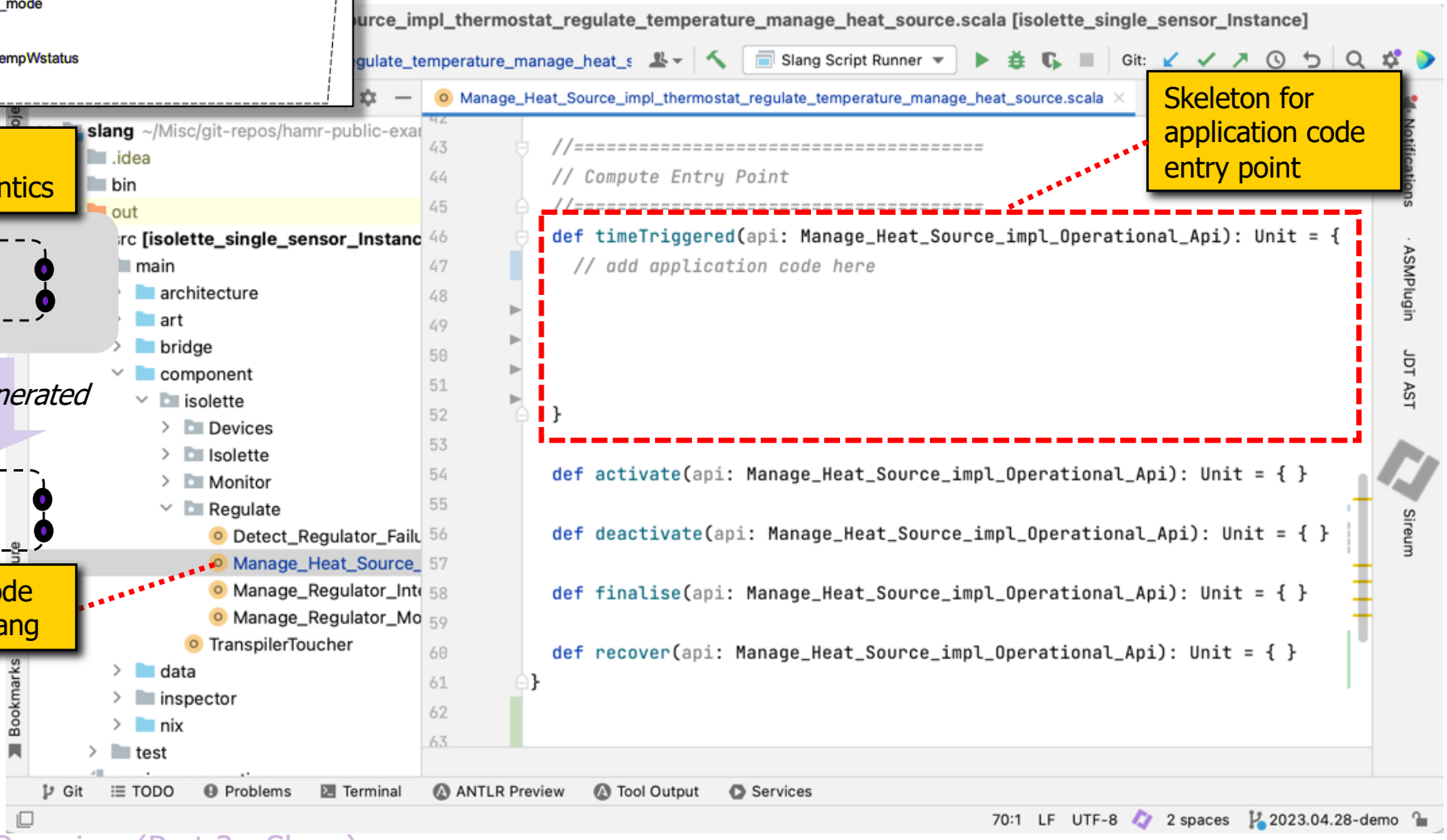
AADL Model Implied Semantics



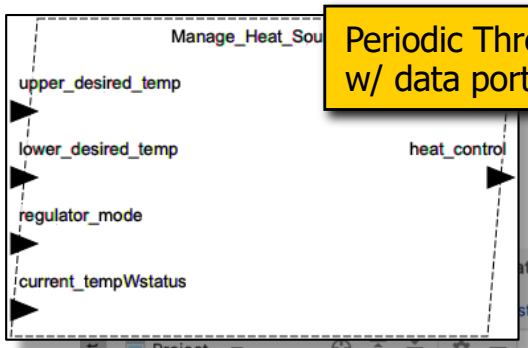
auto-generated

Application Code Skeleton in Slang

Skeleton for application code entry point



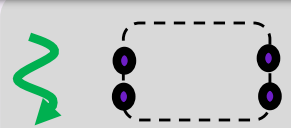
Component Application Code Interfaces Generated from AADL Model



Periodic Thread w/ data ports

...Interfaces/APIs/Skeletons for application code are auto-generated from AADL model

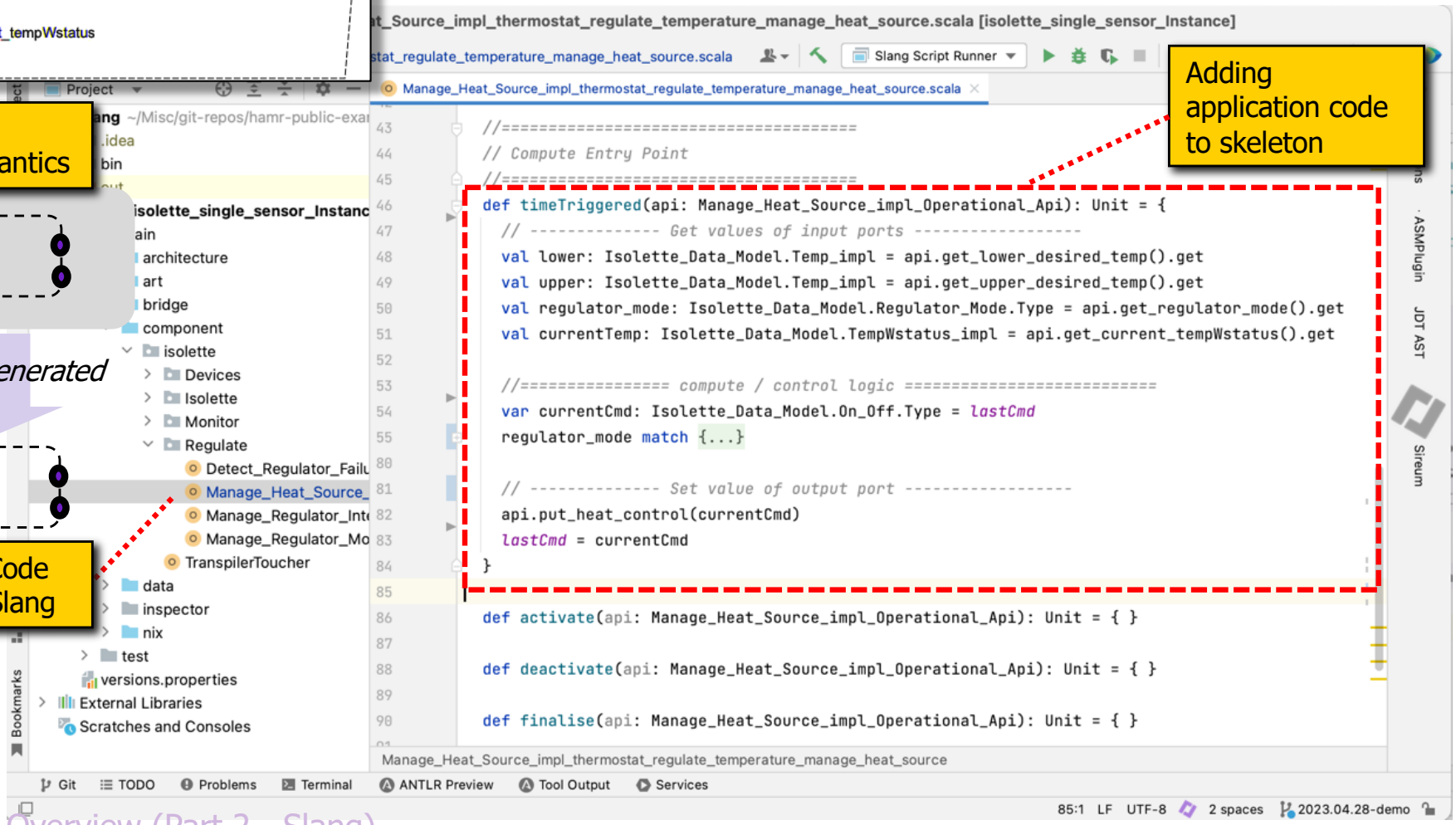
AADL Model Implied Semantics



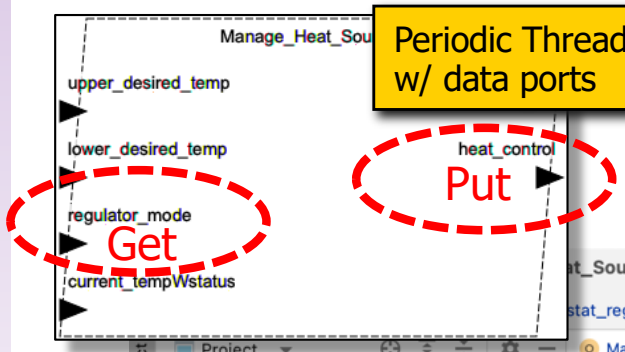
auto-generated

Application Code Skeleton in Slang

Adding application code to skeleton



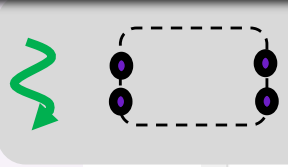
Component Application Code Interfaces Generated from AADL Model



Periodic Thread w/ data ports

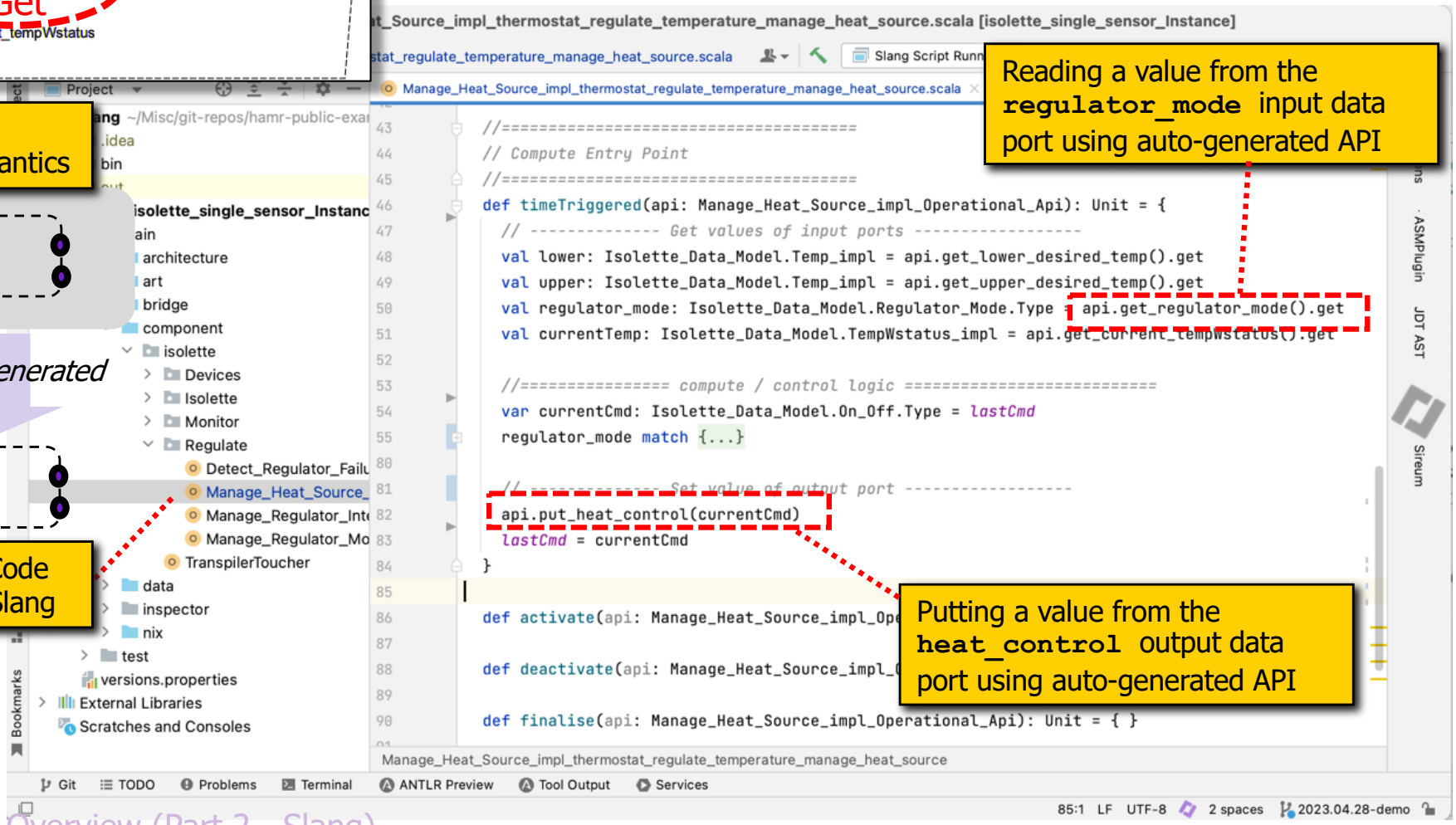
...Interfaces/APIs/Skeletons for application code are auto-generated from AADL model

AADL Model Implied Semantics

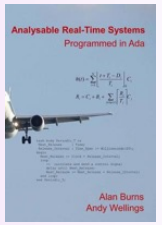


auto-generated

Application Code Skeleton in Slang

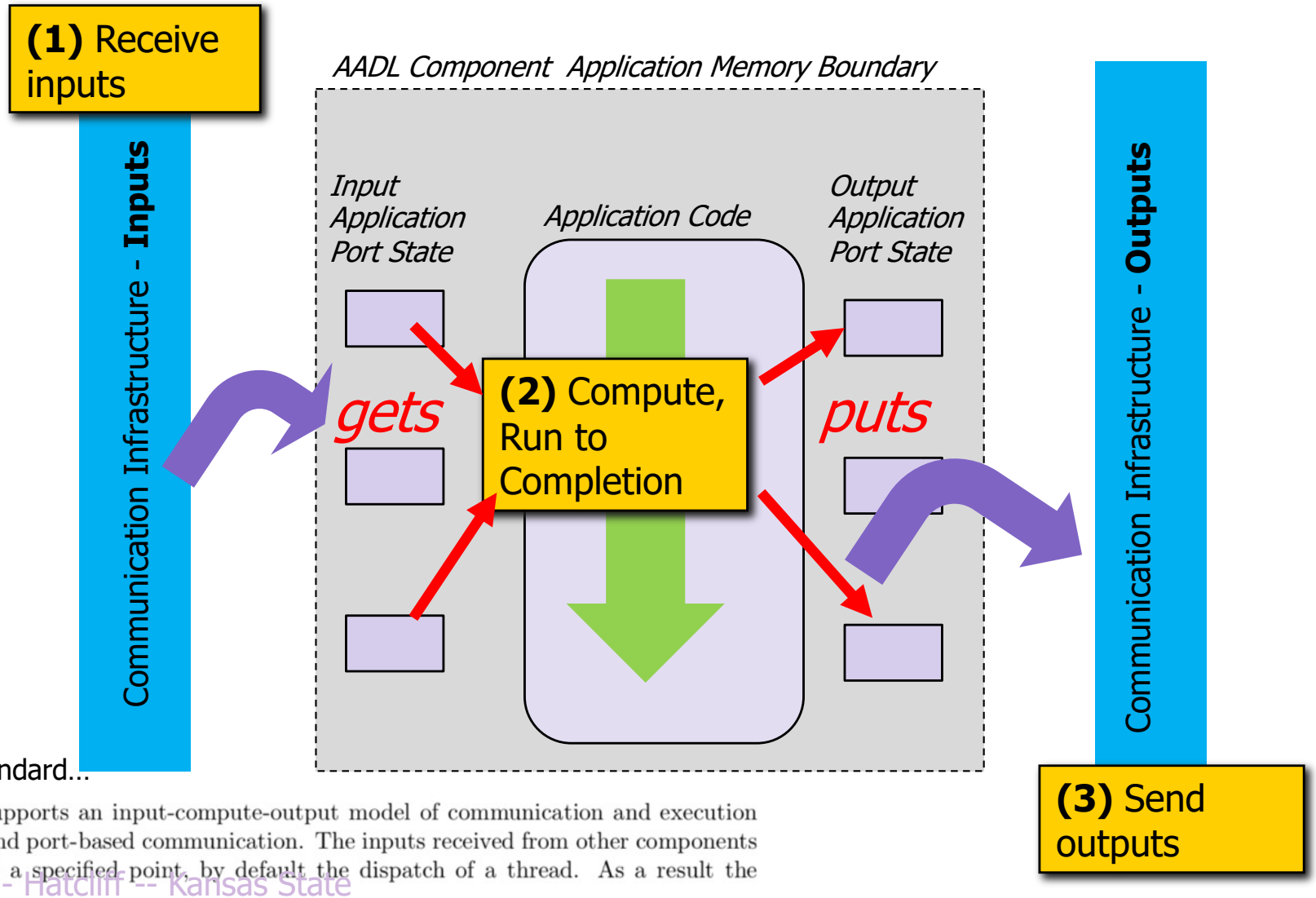


AADL Port and Thread Execution Semantics



"Analyzeable Real-Time Systems" Burns & Wellings

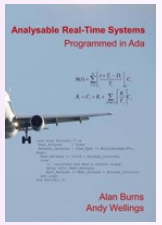
On each dispatch, AADL threads follow a well-known **input-compute-output** pattern for real-time tasks that aid analysis and verification…



From AADL standard…

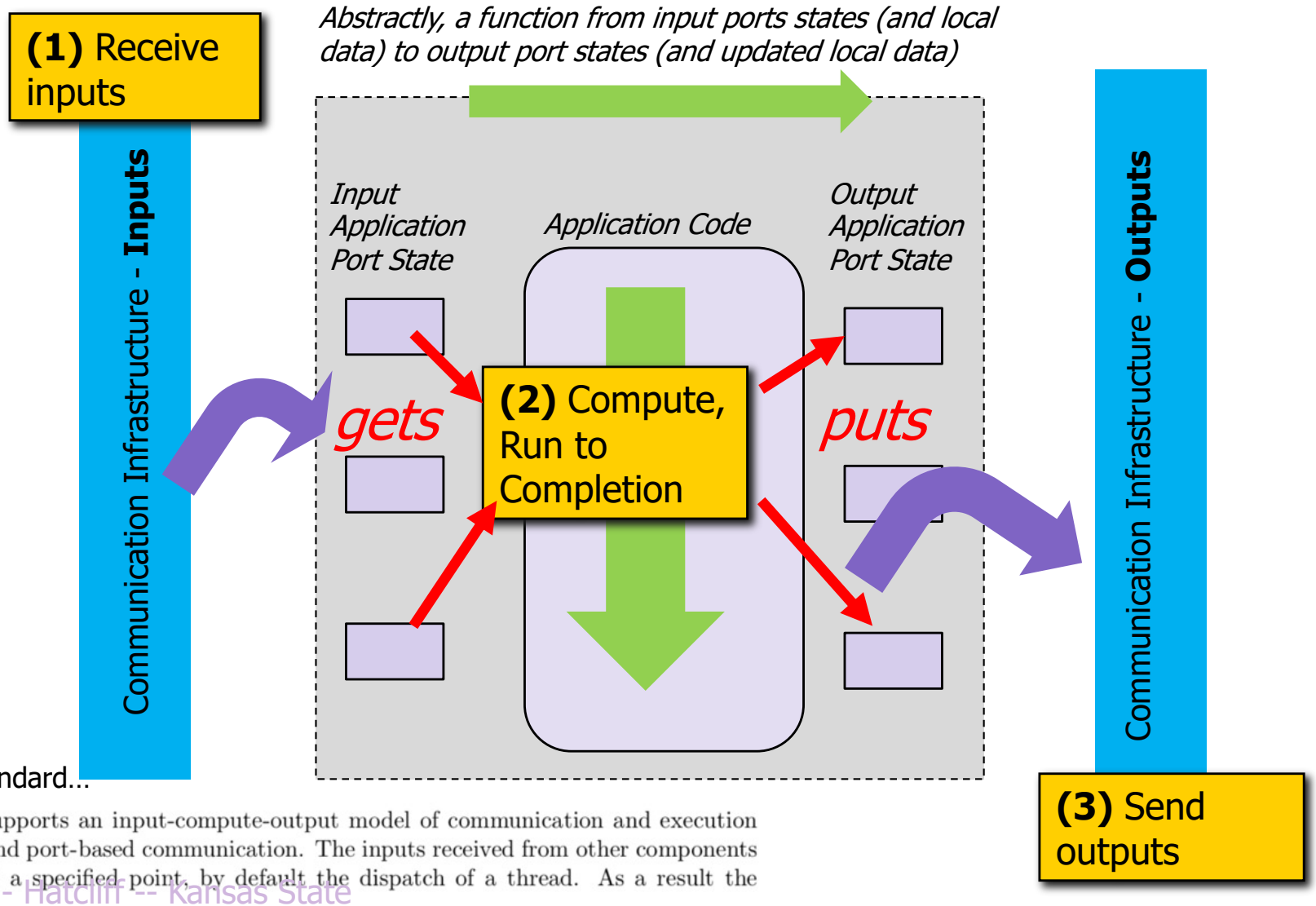
(2) AADL supports an input-compute-output model of communication and execution for threads and port-based communication. The inputs received from other components are frozen at a specified point, by default the dispatch of a thread. As a result the

AADL Port and Thread Execution Semantics



"Analyzeable Real-Time Systems" Burns & Wellings

On each dispatch, AADL threads follow a well-known **input-compute-output** pattern for real-time tasks that aid analysis and verification…

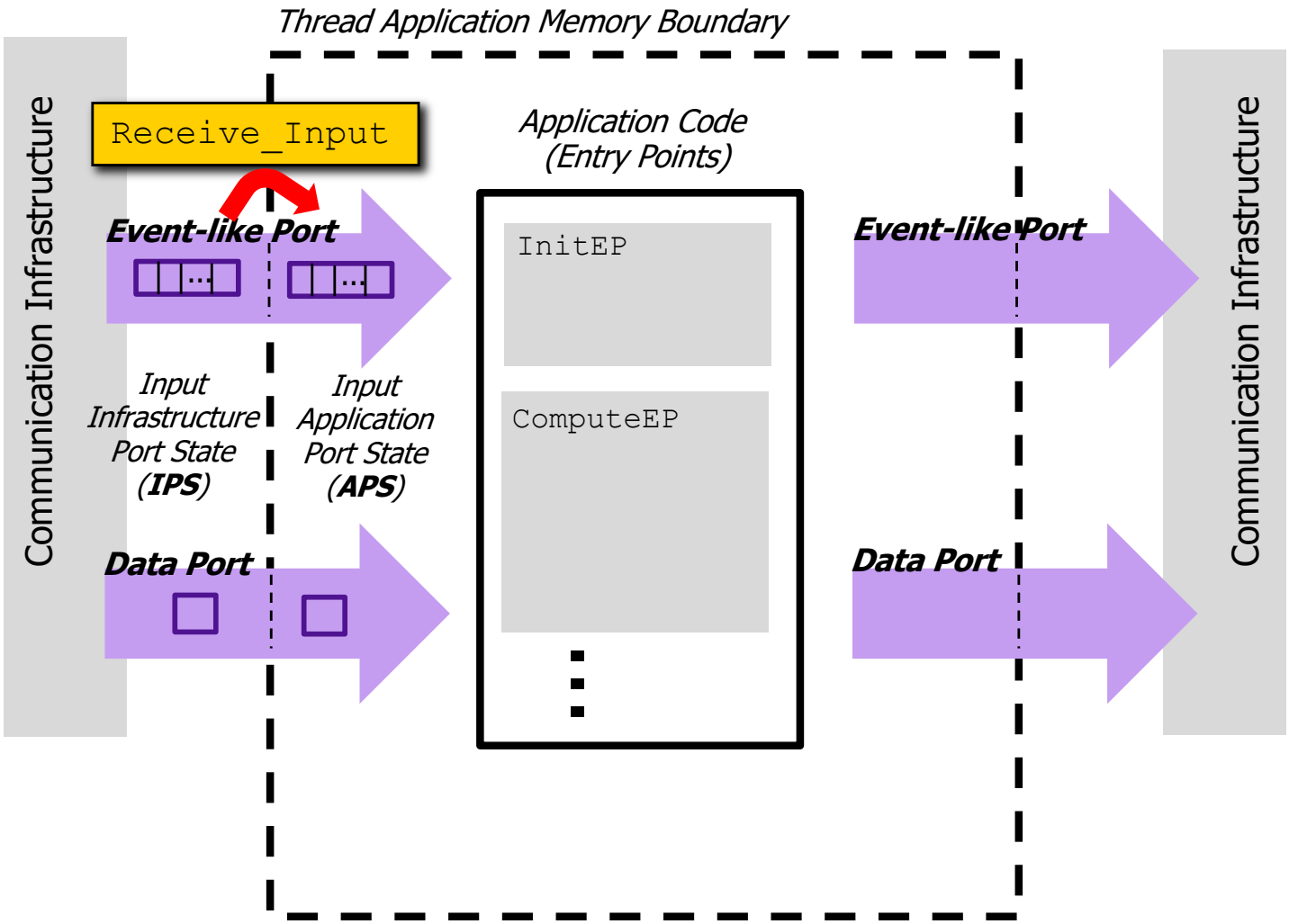


From AADL standard…

(2) AADL supports an input-compute-output model of communication and execution for threads and port-based communication. The inputs received from other components are frozen at a specified point, by default the dispatch of a thread. As a result the

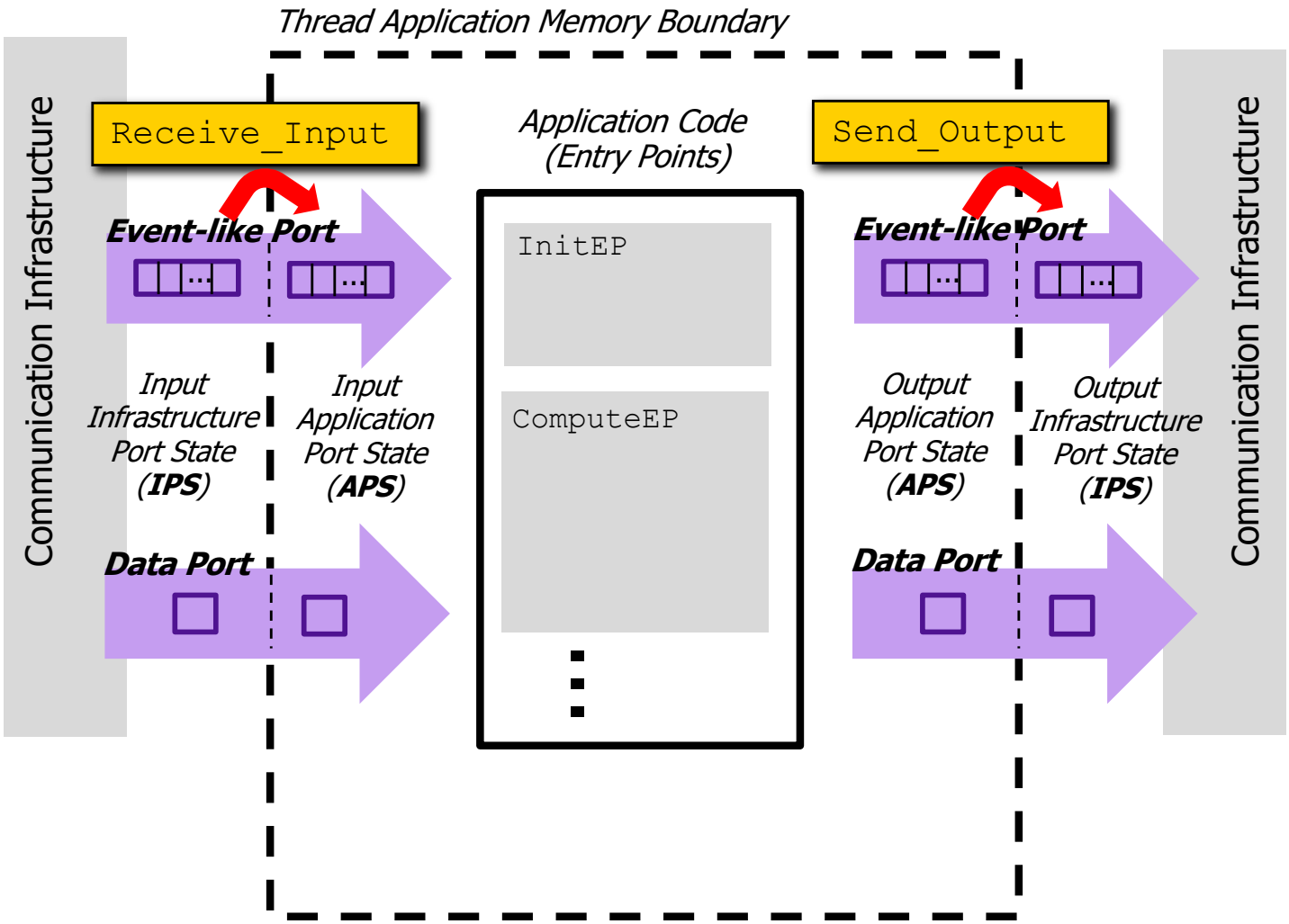
Key Concepts

Towards Formalism: Clarify key elements of the thread state and the run-time service operations on elements



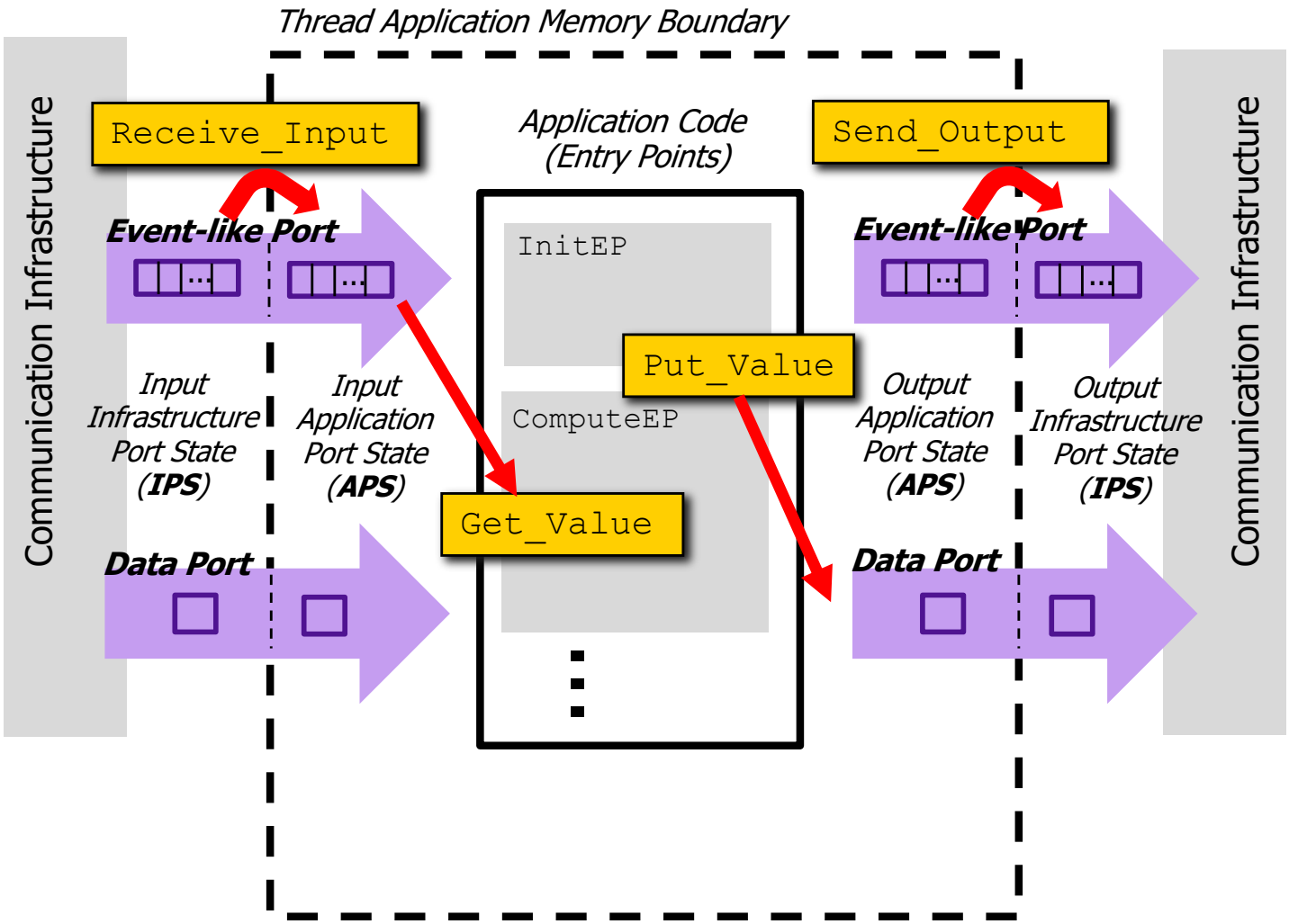
Key Concepts

Towards Formalism: Clarify key elements of the thread state and the run-time service operations on elements



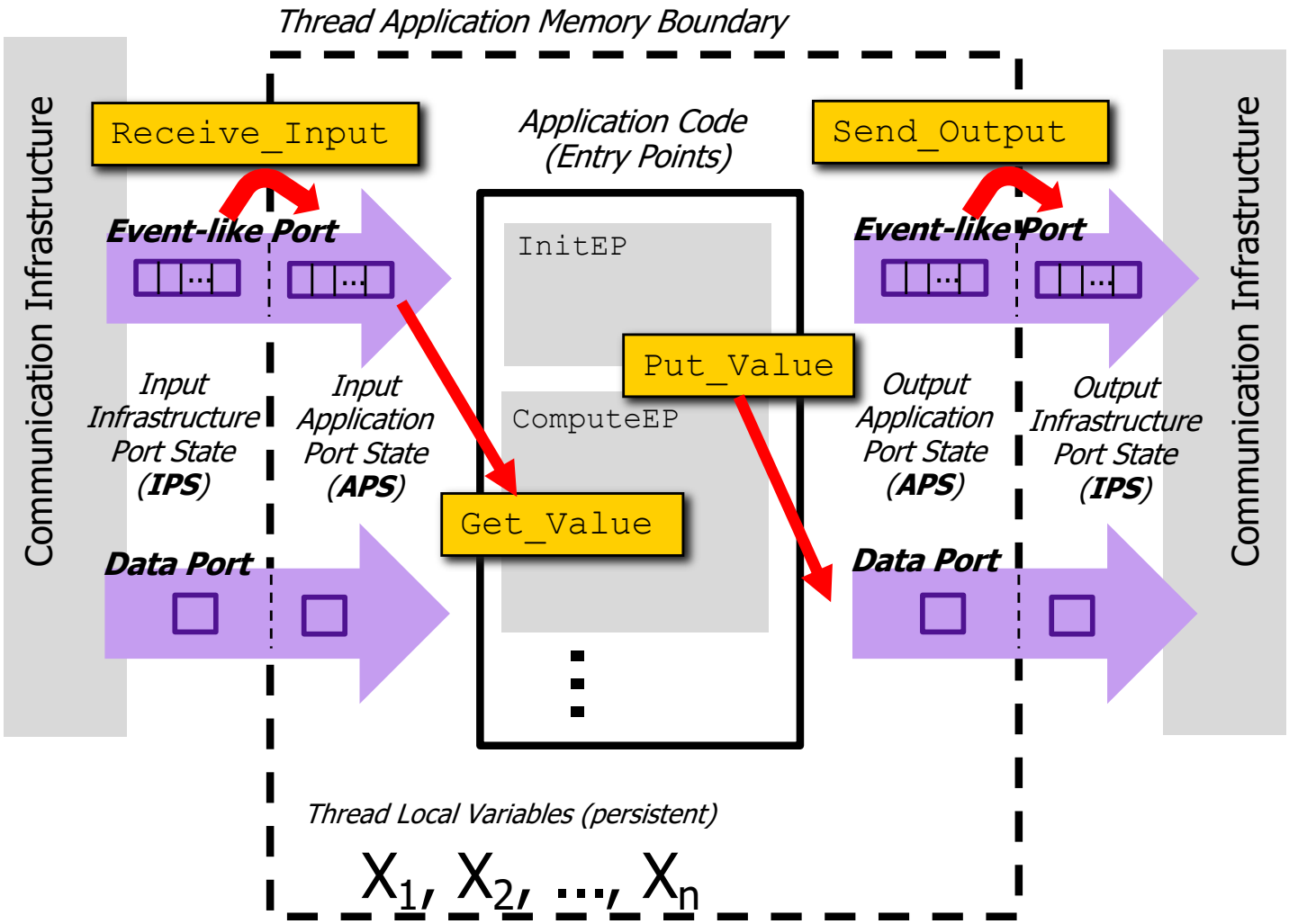
Key Concepts

Towards Formalism: Clarify key elements of the thread state and the run-time service operations on elements



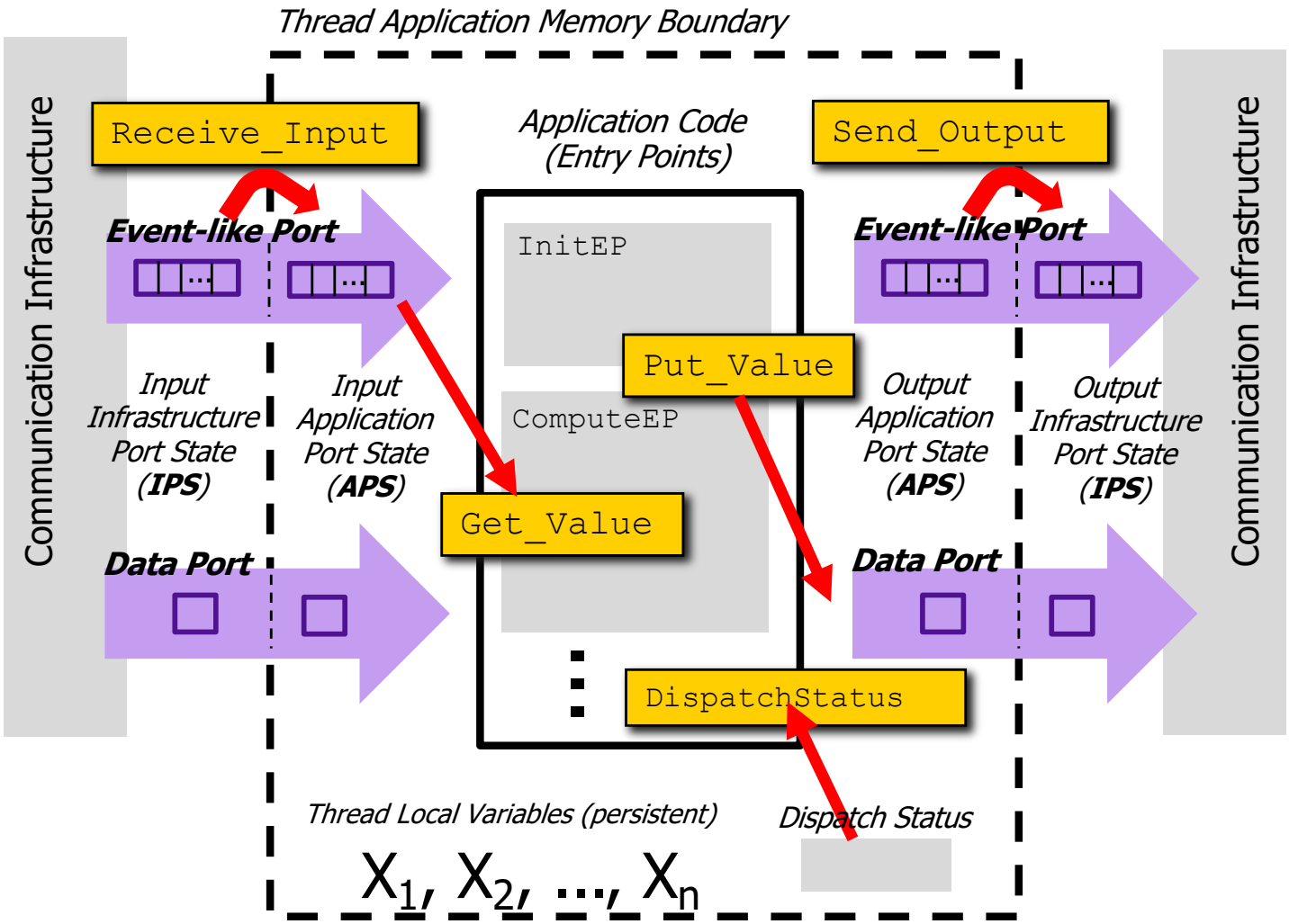
Key Concepts

Towards Formalism: Clarify key elements of the thread state and the run-time service operations on elements



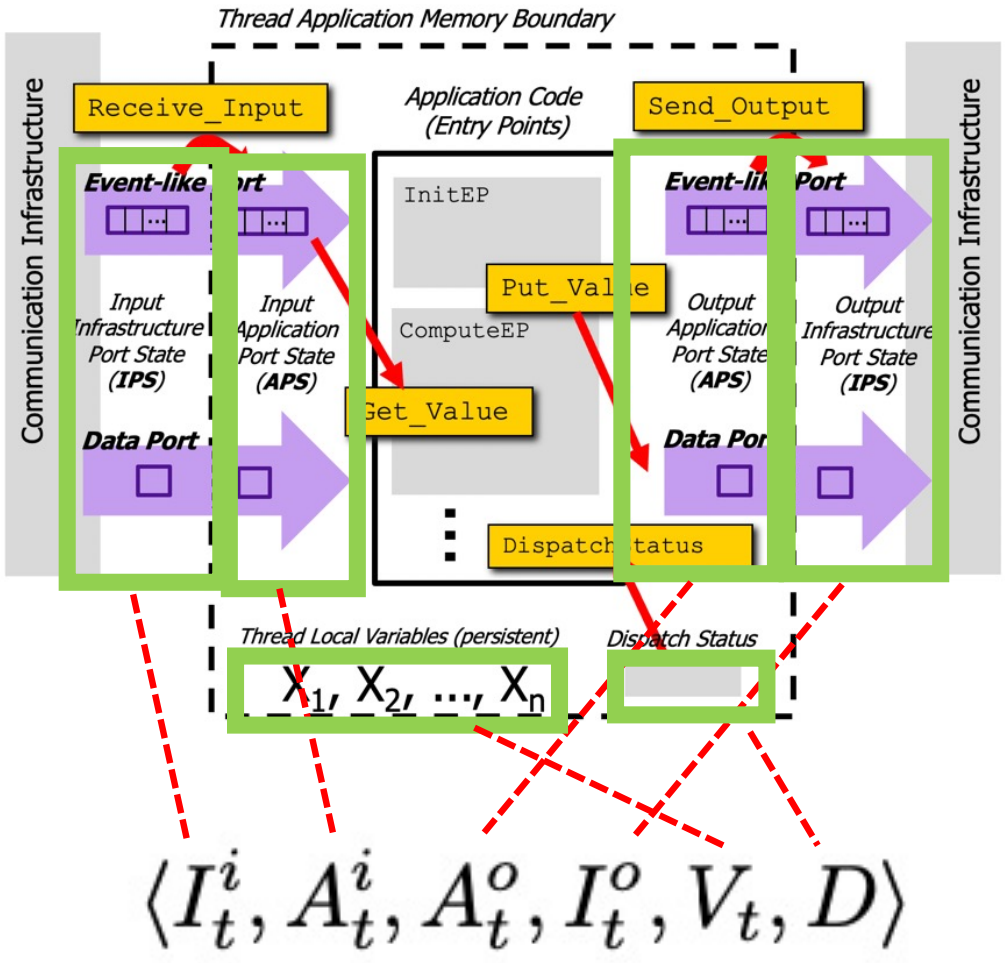
Key Concepts

Towards Formalism: Clarify key elements of the thread state and the run-time service operations on elements



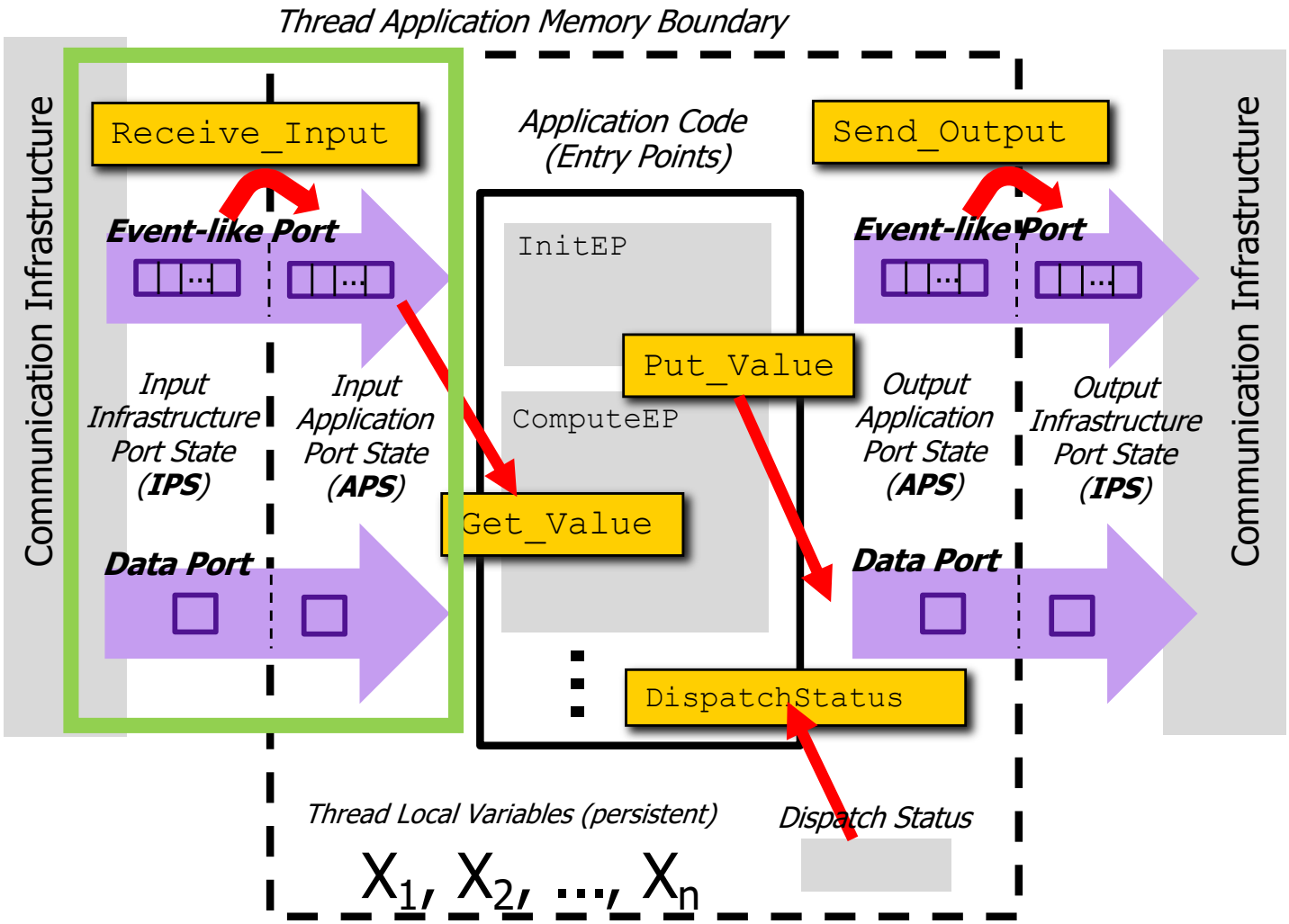
Formalization of Thread State

The concepts of state in this diagram now become part of the formalization of a thread state…



`Receive Input` Runtime Service

Let's consider the formalization for `Receive Input` run-time service...



`Receive Input` Runtime Service

Rules formalizing the behavior of **`Receive Input`** runtime service

Formalization clarifies that on these portions of the thread state are modified by the service.

ReceiveInputData :

$$\frac{\begin{array}{l}\mathcal{M}.isInPort[t](p)\\ \mathcal{M}.isDataPort[t](p)\\ I_t^i(p) = \langle v \rangle \qquad A_t^{i\prime} = A_t^i[p \mapsto \langle v \rangle]\end{array}}{\langle I_t^i, A_t^i, A_t^o, I_t^o, V_t, D\rangle \xrightarrow{\text{RecInP}(p)}_t \langle I_t^i, A_t^{i\prime}, A_t^o, I_t^o, V_t, D\rangle}$$

ReceiveInputEventLikeOneItem :

$$\frac{\begin{array}{l}\mathcal{M}.isInPort[t](p)\\ \mathcal{M}.isEventLikePort[t](p)\\ \mathcal{M}.DequeuePolicy[t](p) = OneItem \qquad I_t^{i\prime} = I_t^i[p \mapsto q]\\ I_t^i(p) = \langle q \triangleright v \rangle \qquad A_t^{i\prime} = A_t^i[p \mapsto \langle v \rangle]\end{array}}{\langle I_t^i, A_t^i, A_t^o, I_t^o, V_t, D\rangle \xrightarrow{\text{RecInP}(p)}_t \langle I_t^{i\prime}, A_t^{i\prime}, A_t^o, I_t^o, V_t, D\rangle}$$

ReceiveInputEventLikeAllItems :

$$\frac{\begin{array}{l}\mathcal{M}.isInPort[t](p)\\ \mathcal{M}.isEventLikePort[t](p)\\ \mathcal{M}.DequeuePolicy[t](p) = AllItems \qquad I_t^{i\prime} = I_t^i[p \mapsto \langle\rangle]\\ \neg\, isEmpty(I_t^i(p)) \qquad A_t^{i\prime} = A_t^i[p \mapsto I_t^i(p)]\end{array}}{\langle I_t^i, A_t^i, A_t^o, I_t^o, V_t, D\rangle \xrightarrow{\text{RecInP}(p)}_t \langle I_t^{i\prime}, A_t^{i\prime}, A_t^o, I_t^o, V_t, D\rangle}$$

ReceiveInputEventLikeEmpty :

$$\frac{\begin{array}{l}\mathcal{M}.isInPort[t](p)\\ \mathcal{M}.isEventLikePort[t](p)\\ isEmpty(I_t^i(p)) \qquad A_t^{i\prime} = A_t^i[p \mapsto \langle\rangle]\end{array}}{\langle I_t^i, A_t^i, A_t^o, I_t^o, V_t, D\rangle \xrightarrow{\text{RecInP}(p)}_t \langle I_t^i, A_t^{i\prime}, A_t^o, I_t^o, V_t, D\rangle}$$

Different behaviors for *data* ports vs *event data* ports

Rules take into account different dequeuing policies configured by AADL properties

Artifact Correspondence/Traceability

Slang Executable Semantics: There is a 1-to-1 correspondence between the mathematical definition of the state and the representation in the executable specification…

$$\langle I_t^i, A_t^i, A_t^o, I_t^o, V_t, D \rangle$$

```
@record class ThreadState(
      IIn: PortStates,
      AIn: PortStates,
      AOut: PortStates,
      IOut: PortStates,
      V: VarStates,
      dispatchStatus: Option[DispatchLogic.DispatchStatusInfo]
)
```

…from Executable Functional Model in *Slang*

…in the executable model, simple logging gives a nice way to see the state transitions of the semantics

Receive Input Runtime Service

Executable functional specification

```
def receive_input_onePort(portId: Model.PortId, iin: PortStates, ain: PortStates) : (Port
  Model.getPortKind(portId) match {
    // ----- D a t a   P o r t s -----
    case Model.PortProperties.Kind.Data =>
      // == Latex Rule: ReceiveInputData
      // i n f r a s t r u c t u r e   p o r t s  (i i n)
      //   -- leave values unchanged - next dispatch can read old value
      val iin_new : PortStates = iin
      // a p p l i c a t i o n   p o r t  (a i n)
      //   - "freeze value" of port by copying value from infrastructure port to port variable
      val queueValue: PortQueue.Type = iin.get(portId).get // get current infrastructure port value
      assert(queueValue.size == 1)             // the size of a data port queue should always be 1
      val ain_new : PortStates = ain + portId ~> queueValue // copy current value to port variable
      return (iin_new, ain_new)
    // ----- E v e n t   P o r t s -----
    case Model.PortProperties.Kind.Event =>
      val iin_portId_queue: PortQueue.Type = iin.get(portId).get
      // val ain_port: PortQueue.Type = ain.get(portId).get  // consider seeding the helper below with empty queue
      val (iin_portId_queue_new, ain_portId_queue_new) = receive_input_event_port_dequeue_policy(iin_portId_queue)
      /* iin_new queue is emptied for this port */
      val iin_new = iin + portId ~> iin_portId_queue_new
      val ain_new = ain + portId ~> ain_portId_queue_new
      return (iin_new, ain_new)
  }
}
```

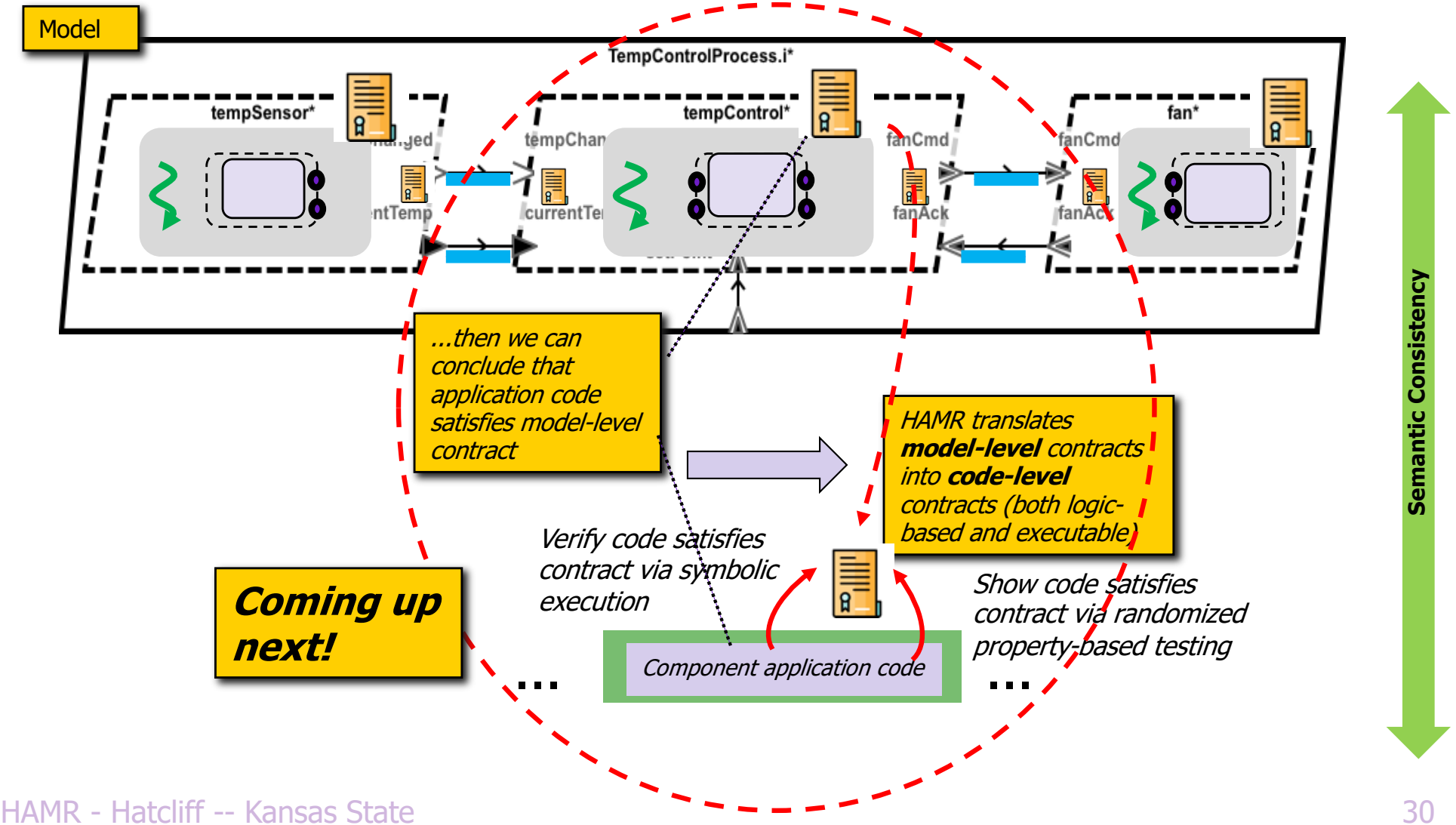
Code clarifies that only these parts of the thread state are modified by the service.

Different behaviors for *data* ports vs *event* ports

Following the rules, the code takes into into account different dequeuing policies configured by AADL properties

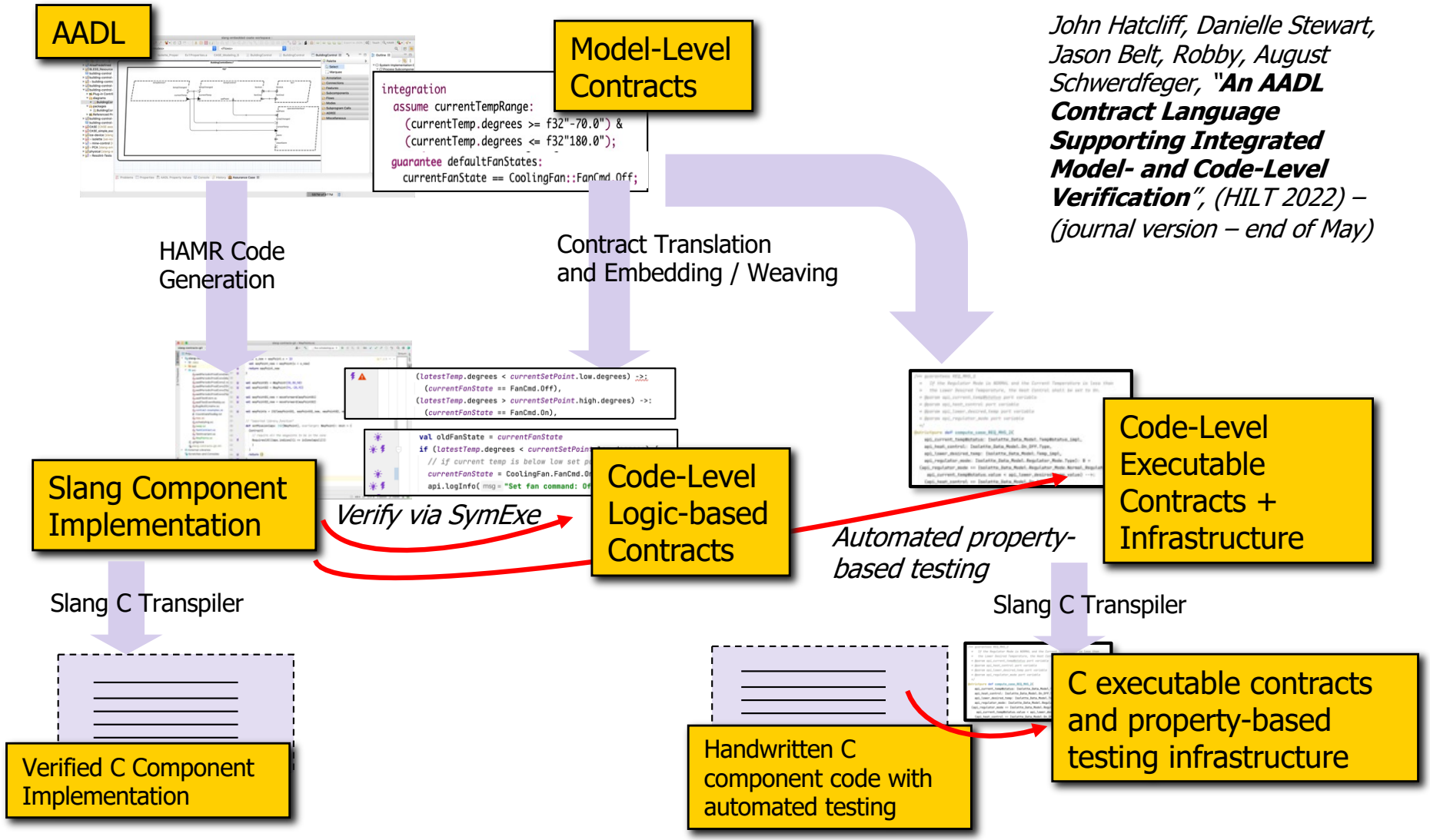
Application: Contracts

The formal semantics guides the design of an **integrated model and code-level contract framework** that supports both verification and property-based testing



Application:
Integrated Model/Code Contract Language

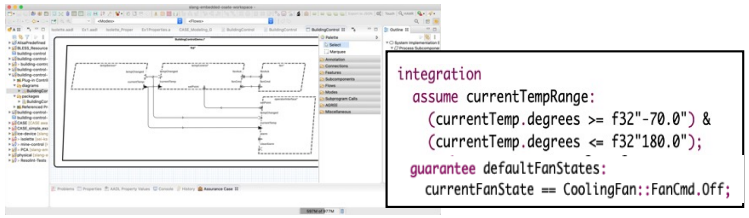
KSU / Adventium Labs (now Galois) ARMY SBIR Phase II...



*John Hatcliff, Danielle Stewart, Jason Belt, Robby, August Schwerdfeger, "**An AADL Contract Language Supporting Integrated Model- and Code-Level Verification**", (HILT 2022) – (journal version – end of May)*

GUMBO - AADL Contract Language

GUMBO Contract Language Features



Inspired by previous work on AGREE and BLESS

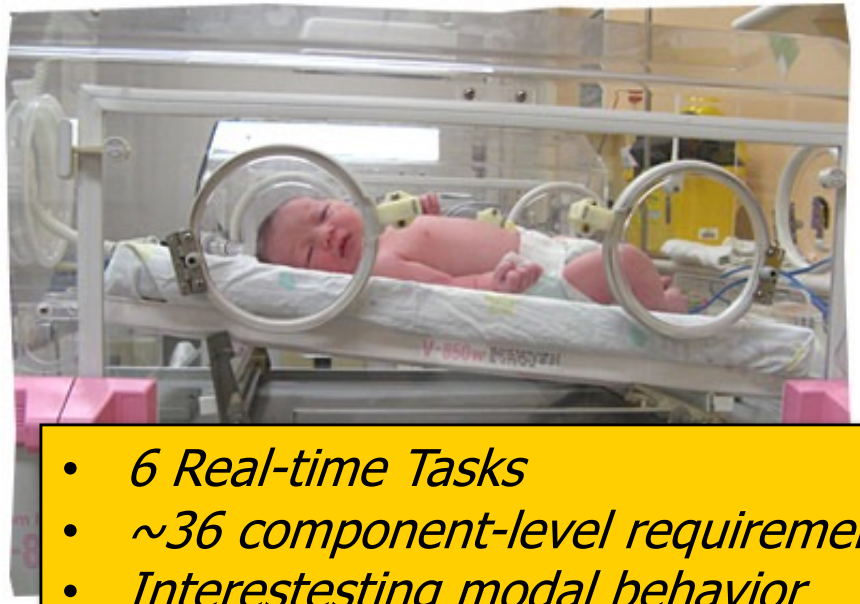
- Data type invariants
- Port invariants (integration constraints)
- Event-based / Shared-data based inter-thread communication
- Local state declarations with invariants
- Pre/Post conditions for AADL thread code entrypoints
  - Initialize Entry Point
  - Compute Entry Point
    - Periodic
    - Sporadic (collection of event handlers)
- Support for fixed width scalars (e.g., Float32)
- Support for almost all of the AADL Data Modeling Annex

Example in this talk

FAA Requirements Engineering Management Handbook (REMH)

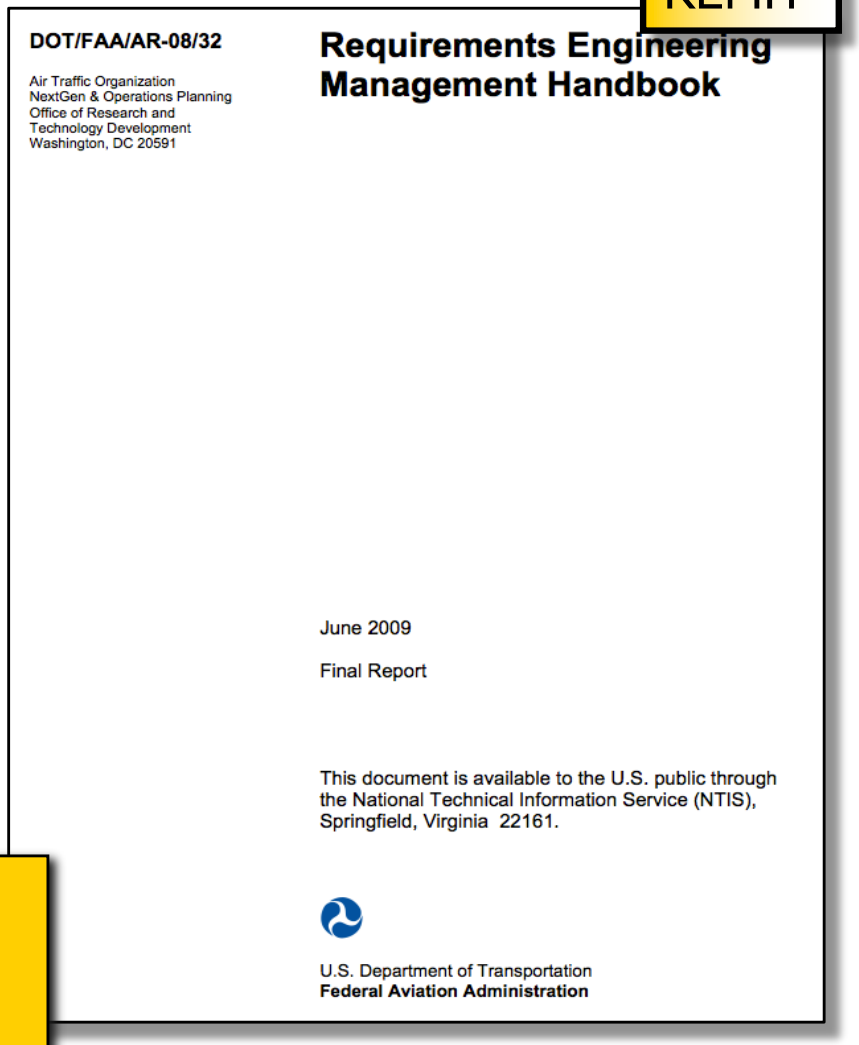
Illustrate with the Isolette example from FAA REMH

- Written for the FAA by engineers at Rockwell Collins (David L. Lempia, Steven P. Miller)
- Includes example of an "Isolette" (infant incubator)



- *6 Real-time Tasks*
- *~36 component-level requirements*
- *Interestesting modal behavior*

REMH



DOT/FAA/AR-08/32

Air Traffic Organization
NextGen & Operations Planning
Office of Research and
Technology Development
Washington, DC 20591

Requirements Engineering Management Handbook

June 2009

Final Report

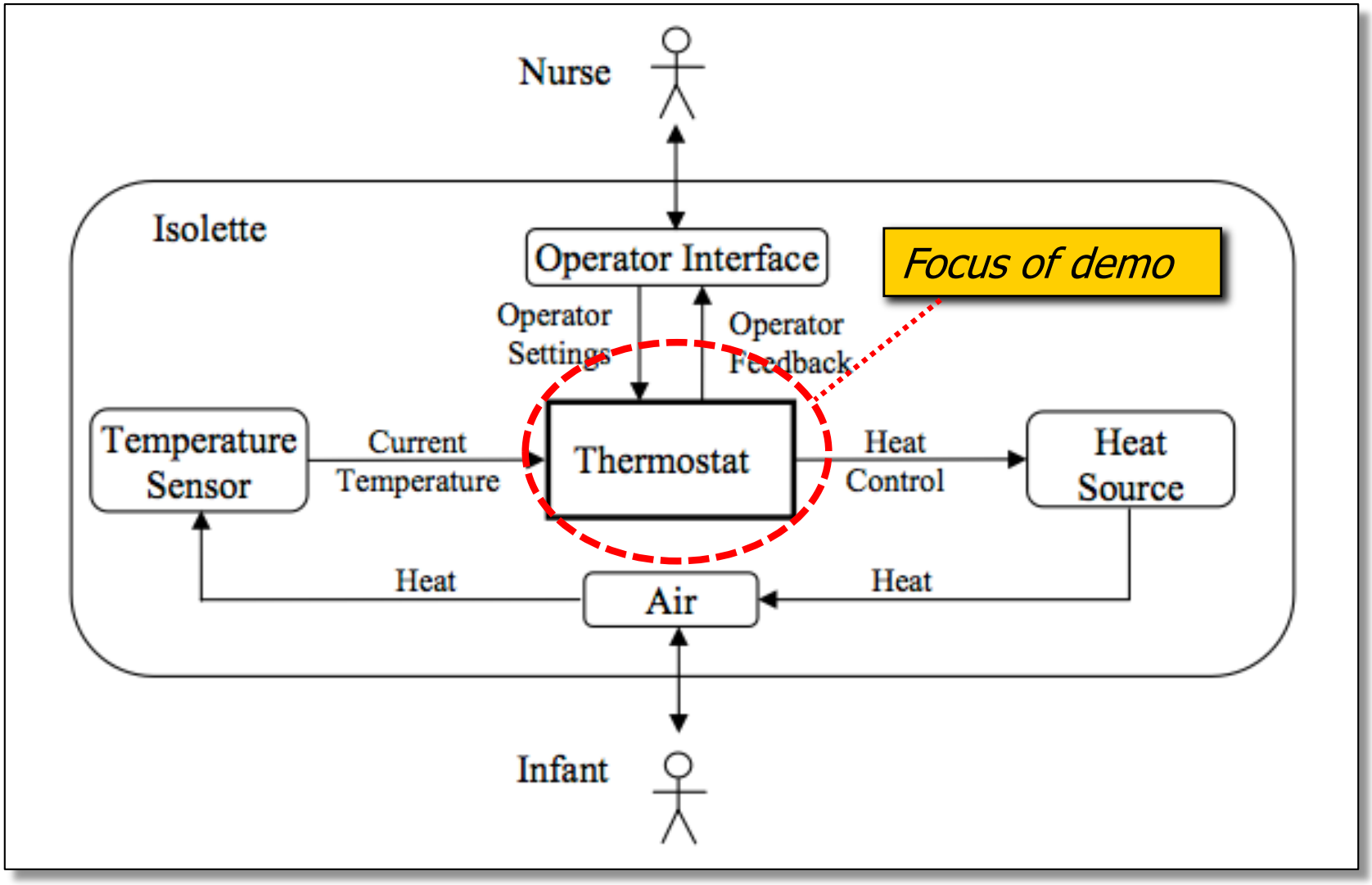
This document is available to the U.S. public through the National Technical Information Service (NTIS), Springfield, Virginia 22161.

U.S. Department of Transportation
Federal Aviation Administration

http://www.faa.gov/aircraft/air_cert/design_approvals/air_software/research/

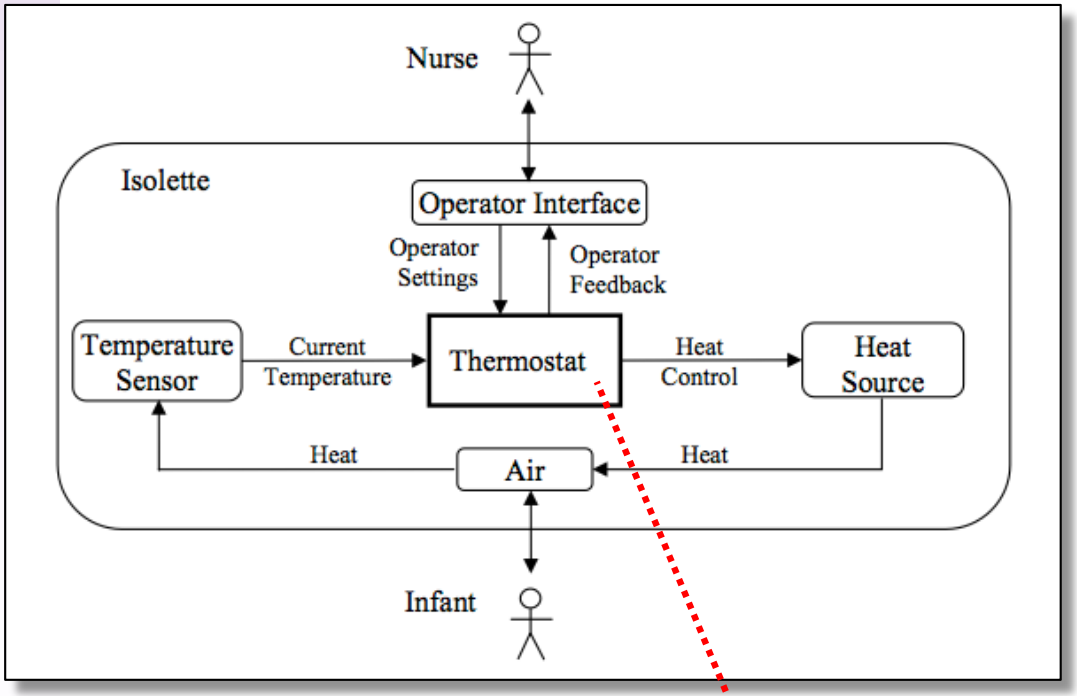
Focus of Example

Isolette Thermostat – heat controller for incubator

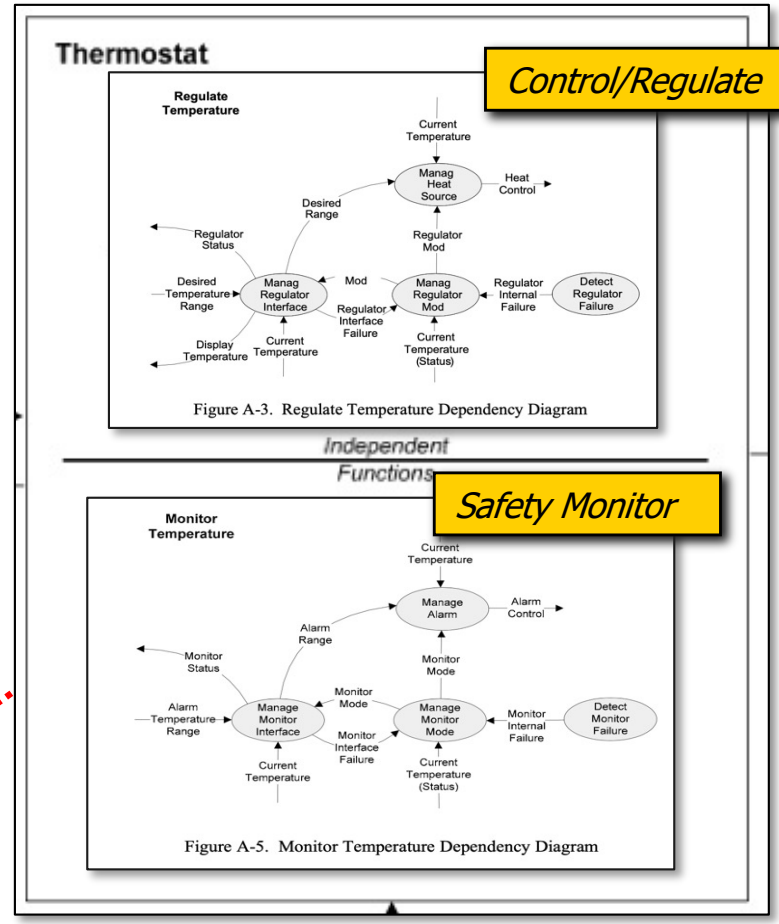


Decomposing Thermostat

The FAA REMH decomposes the Isolette into a control system and safety monitor with three tasks each



Decomposing the Thermostat into Regulate Temperature and Monitor Temperature functions.



Using AADL to Represent Design

AADL Model

AADL model originally developed by Brian Larson
(creator of BLESS specification and verification framework)

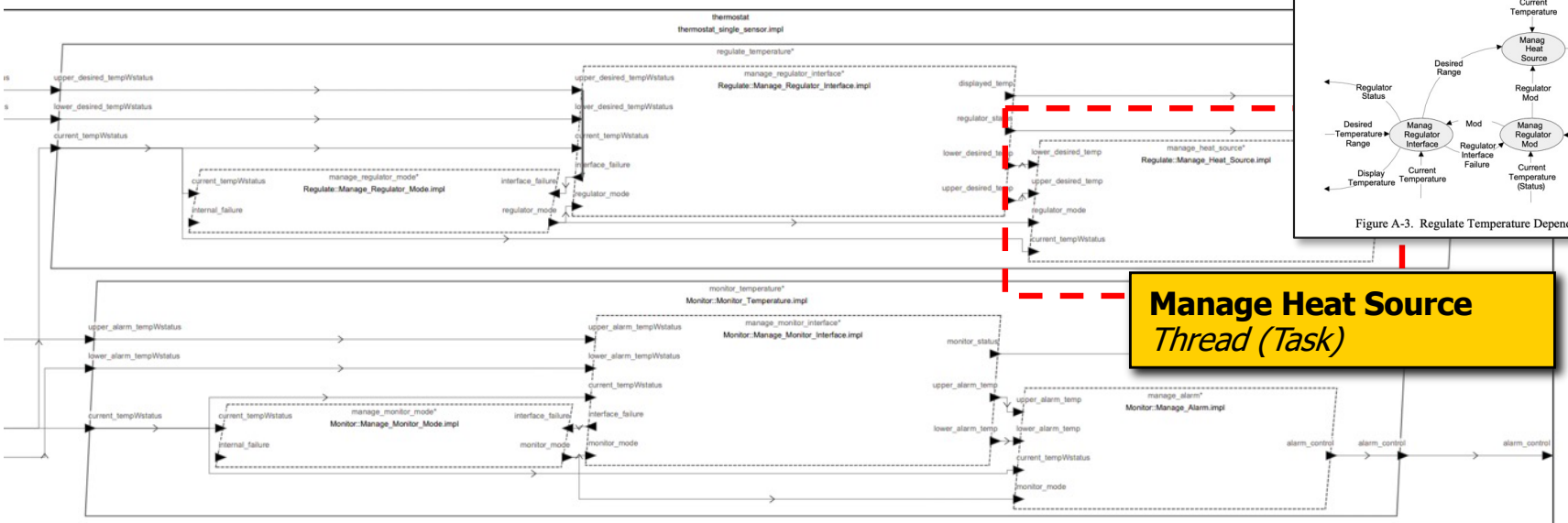


Figure A-3. Regulate Temperature Dependency Diagram

Manage Heat Source
Thread (Task)

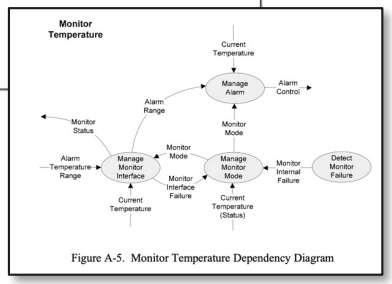
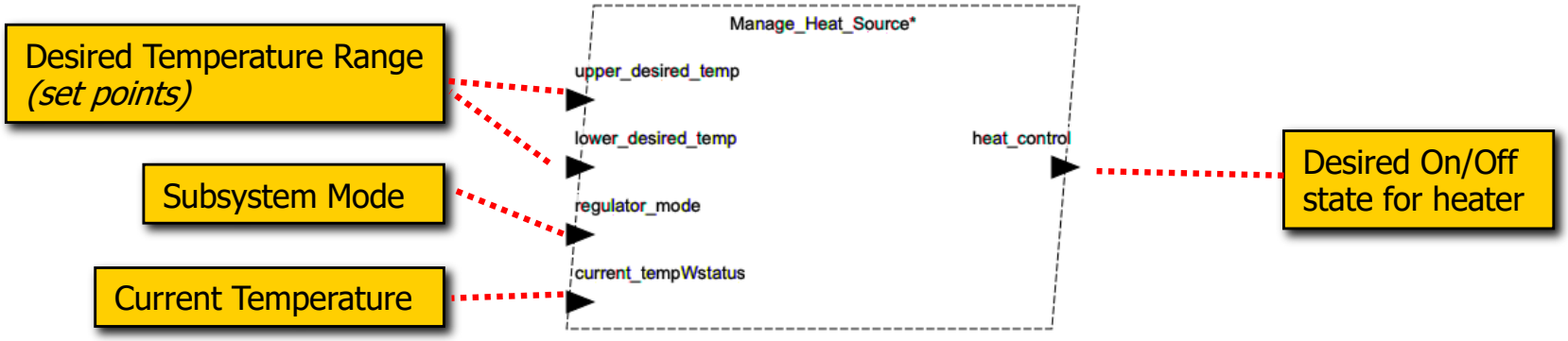


Figure A-5. Monitor Temperature Dependency Diagram

This example is worked completely from end-to-end from requirements, to contracts, to automatically tested and verified code, to deployment on seL4, Linux, JVM, JavaScript, and the artifacts are publicly available.

Manage Heat Source Thread

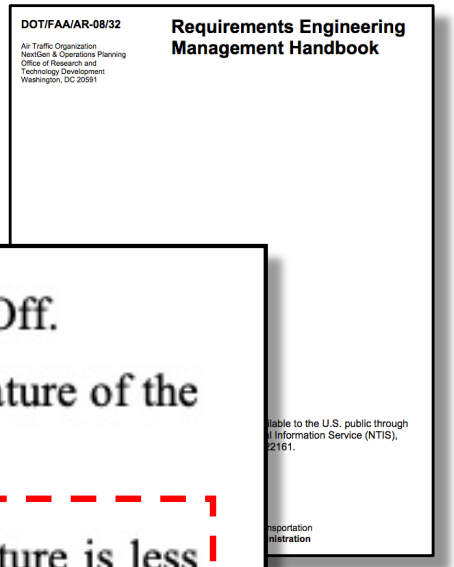
AADL Interface for **Manage Heat Source** Thread




```
thread Manage_Heat_Source
  features
    -- ======== INPUTS =======
    -- ("Current Temperature") - current temperature (from temp sensor)
    current_tempWstatus: in data port Isolette_Data_Model::TempWstatus.impl;
    -- ("Desired Range") - lowest and upper bound of desired temperature range
    lower_desired_temp: in data port Isolette_Data_Model::Temp.impl;
    upper_desired_temp: in data port Isolette_Data_Model::Temp.impl;
    -- ("Regulator Mode") - subsystem mode
    regulator_mode: in data port Isolette_Data_Model::Regulator_Mode;

    -- ======== OUTPUTS =======
    -- ("Heat Control") - command to turn heater on/off (actuation command)
    heat_control: out data port Isolette_Data_Model::On_Off;
```

Requirements to Contracts

FAA REMH requirements for **Manage Heat Source** task


DOT/FAA/AR-08/32

Air Traffic Organization
NextGen & Operations Planning
Office of Research and
Technology Development
Washington, DC 20591

Requirements Engineering Management Handbook

Requirements for control laws of this task...

REQ-MHS-1: If the Regulator Mode is INIT, the Heat Control shall be set to Off.

Rationale: A regulator that is initializing cannot regulate the Current Temperature of the Isolette and the Heat Control should be turned off.

REQ-MHS-2: If the Regulator Mode is NORMAL and the Current Temperature is less than the Lower Desired Temperature, the Heat Control shall be set to On.

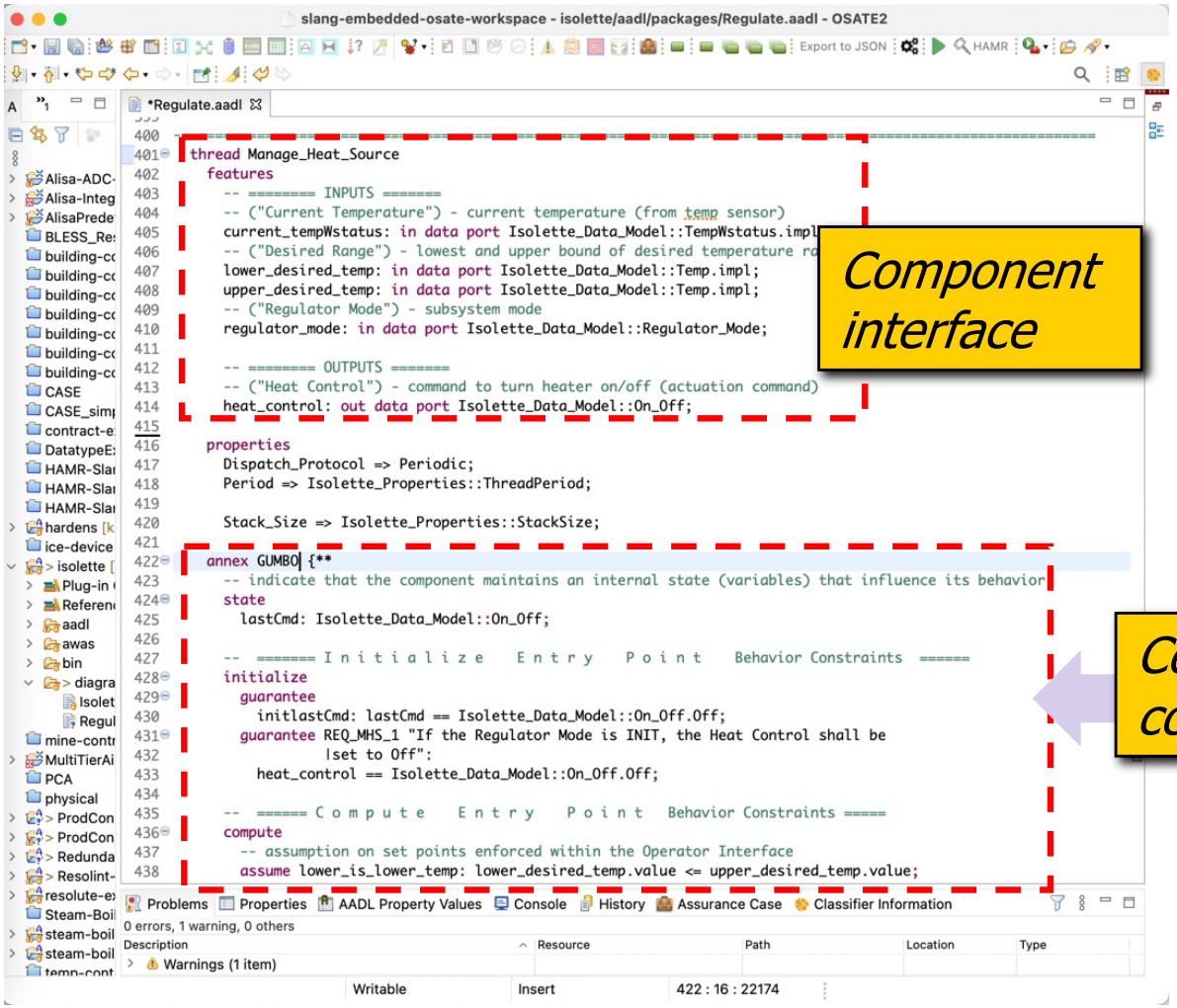
REQ-MHS-3: If the Regulator Mode is NORMAL and the Current Temperature is greater than the Upper Desired Temperature, the Heat Control shall be set to Off.

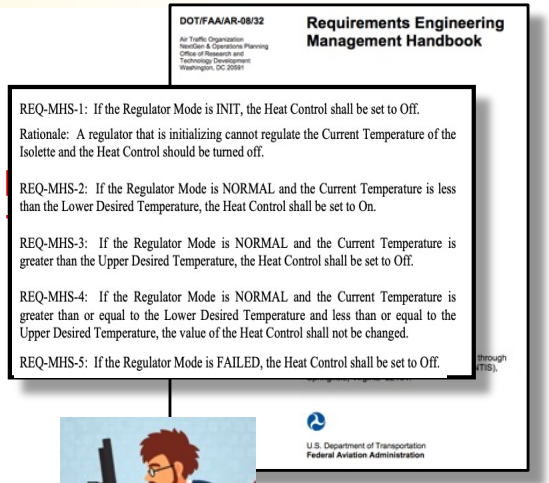
REQ-MHS-4: If the Regulator Mode is NORMAL and the Current Temperature is greater than or equal to the Lower Desired Temperature and less than or equal to the Upper Desired Temperature, the value of the Heat Control shall not be changed.

REQ-MHS-5: If the Regulator Mode is FAILED, the Heat Control shall be set to Off.

Requirements to Contracts

GUMBO contracts are written together with the thread interface in the AADL OSATE IDE (using AADL Annex clause)



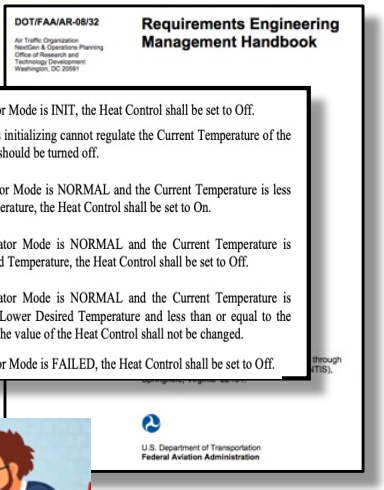


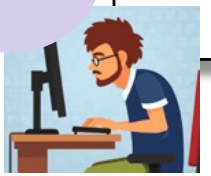
Developer formalizes requirements

Manage Heat Source Contracts

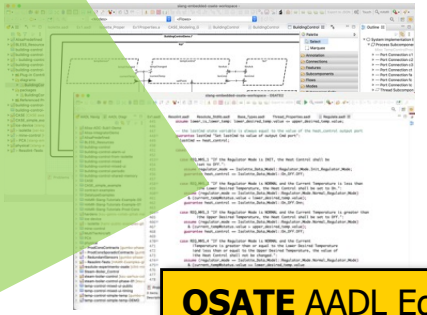
AADL GUMBO Contracts for **Manage Heat Source** Thread, with traceability to REMH requirements.

```
case REQ_MHS_1 "If the Regulator Mode is INIT, the Heat Control shall be
        |set to Off.":
  assume regulator_mode == Isolette_Data_Model::Regulator_Mode.Init_Regulator_Mode;
  guarantee heat_control == Isolette_Data_Model::On_Off.Off;

case REQ_MHS_2 "If the Regulator Mode is NORMAL and the Current Temperature is less than
        |the Lower Desired Temperature, the Heat Control shall be set to On.":
  assume (regulator_mode == Isolette_Data_Model::Regulator_Mode.Normal_Regulator_Mode)
    & (current_tempWstatus.value < lower_desired_temp.value);
  guarantee heat_control == Isolette_Data_Model::On_Off.Onn;

case REQ_MHS_3 "If the Regulator Mode is NORMAL and the Current Temperature is greater than
        |the Upper Desired Temperature, the Heat Control shall be set to Off.":
  assume (regulator_mode == Isolette_Data_Model::Regulator_Mode.Normal_Regulator_Mode)
    & (current_tempWstatus.value > upper_desired_temp.value);
  guarantee heat_control == Isolette_Data_Model::On_Off.Off;

case REQ_MHS_4 "If the Regulator Mode is NORMAL and the Current
      |Temperature is greater than or equal to the Lower Desired Temperature
      |and less than or equal to the Upper Desired Temperature, the value of
      |the Heat Control shall not be changed.":
  assume (regulator_mode == Isolette_Data_Model::Regulator_Mode.Normal_Regulator_Mode)
    & (current_tempWstatus.value >= lower_desired_temp.value
    & current_tempWstatus.value <= upper_desired_temp.value);
  guarantee heat_control == In(lastCmd);

case REQ_MHS_5 "If the Regulator Mode is FAILED, the Heat Control shall be
        |set to Off.":
  assume regulator_mode == Isolette_Data_Model::Regulator_Mode.Failed_Regulator_Mode;
  guarantee heat_control == Isolette_Data_Model::On_Off.Off;
```





Developer formalizes requirements



OSATE AADL Editor

Manage Heat Source Contracts

AADL GUMBO Contracts for **Manage Heat Source** Thread, with traceability to REMH requirements.

Mode condition

Compare current temperature to desired range

...

```
case REQ_MHS_2 "If the Regulator Mode is NORMAL and the Current Temperature is less than
        |the Lower Desired Temperature, the Heat Control shall be set to On.":
  assume (regulator_mode == Isolette_Data_Model::Regulator_Mode.Normal_Regulator_Mode)
    & (current_tempWstatus.value < lower_desired_temp.value);
  guarantee heat_control == Isolette_Data_Model::On_Off.Onn;
```

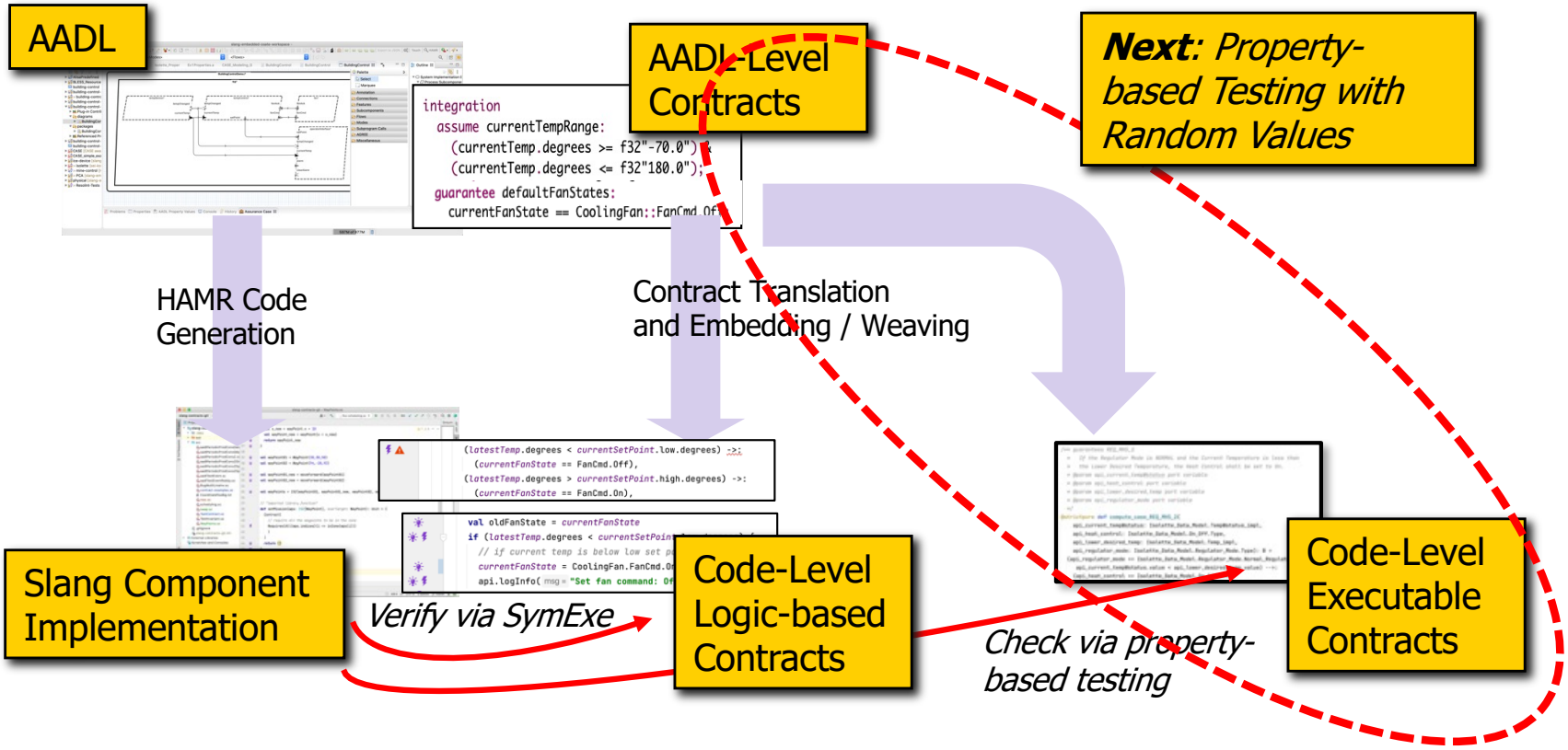
...

Set the desired state of the heater

OSATE AADL Editor

Application:
Property-based (Contract-based) Testing

KSU / Adventium Labs (now Galois) ARMY SBIR Phase II...



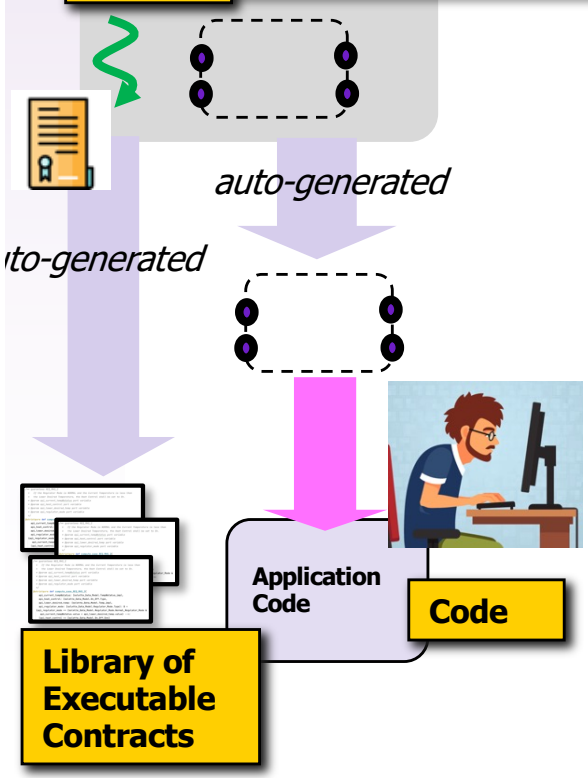
Manage Heat Source Contracts

Translation of **model-level** GUMBO contracts to Slang **code-level executable contracts**

AADL GUMBO Contract (clause)

```
case REQ_MHS_2 "If the Regulator Mode is NORMAL and the Current Temperature is less than
        |the Lower Desired Temperature, the Heat Control shall be set to On.":
  assume (regulator_mode == Isolette_Data_Model::Regulator_Mode.Normal_Regulator_Mode)
    & (current_tempWstatus.value < lower_desired_temp.value);
  guarantee heat_control == Isolette_Data_Model::On_Off.Onn;
```

Model



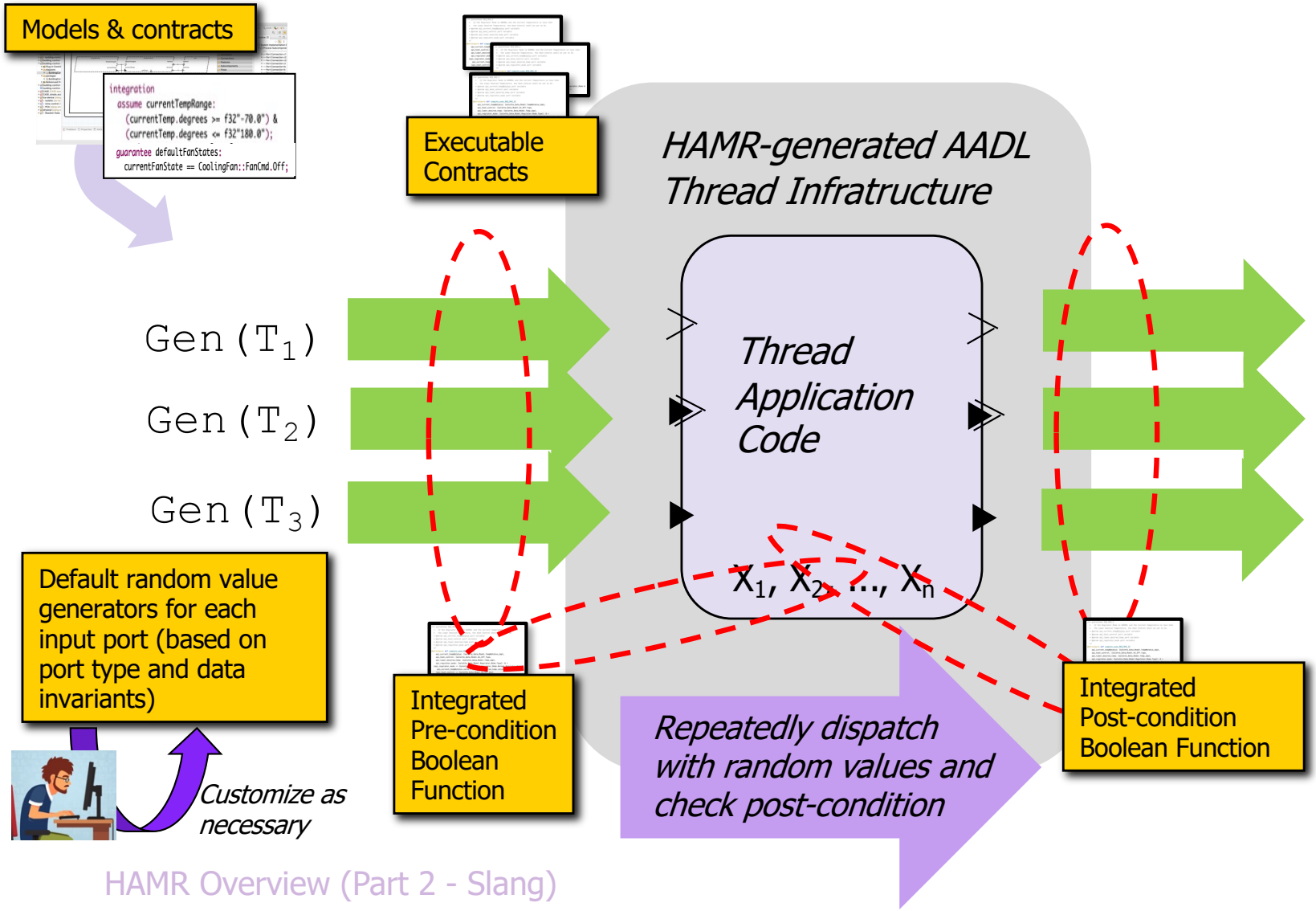
Slang Executable Contract (clause)

```
/** guarantees REQ_MHS_2
  *   If the Regulator Mode is NORMAL and the Current Temperature is less than
  *   the Lower Desired Temperature, the Heat Control shall be set to On.
  * @param api_current_tempWstatus port variable
  * @param api_heat_control port variable
  * @param api_lower_desired_temp port variable
  * @param api_regulator_mode port variable
  */
@strictpure def compute_case_REQ_MHS_2(
    api_current_tempWstatus: Isolette_Data_Model.TempWstatus_impl,
    api_heat_control: Isolette_Data_Model.On_Off.Type,
    api_lower_desired_temp: Isolette_Data_Model.Temp_impl,
    api_regulator_mode: Isolette_Data_Model.Regulator_Mode.Type): B =
  (api_regulator_mode == Isolette_Data_Model.Regulator_Mode.Normal_Regulator_Mode &
     api_current_tempWstatus.value < api_lower_desired_temp.value) -->:
    (api_heat_control == Isolette_Data_Model.On_Off.Onn)
```

Traceability info automatically embedded

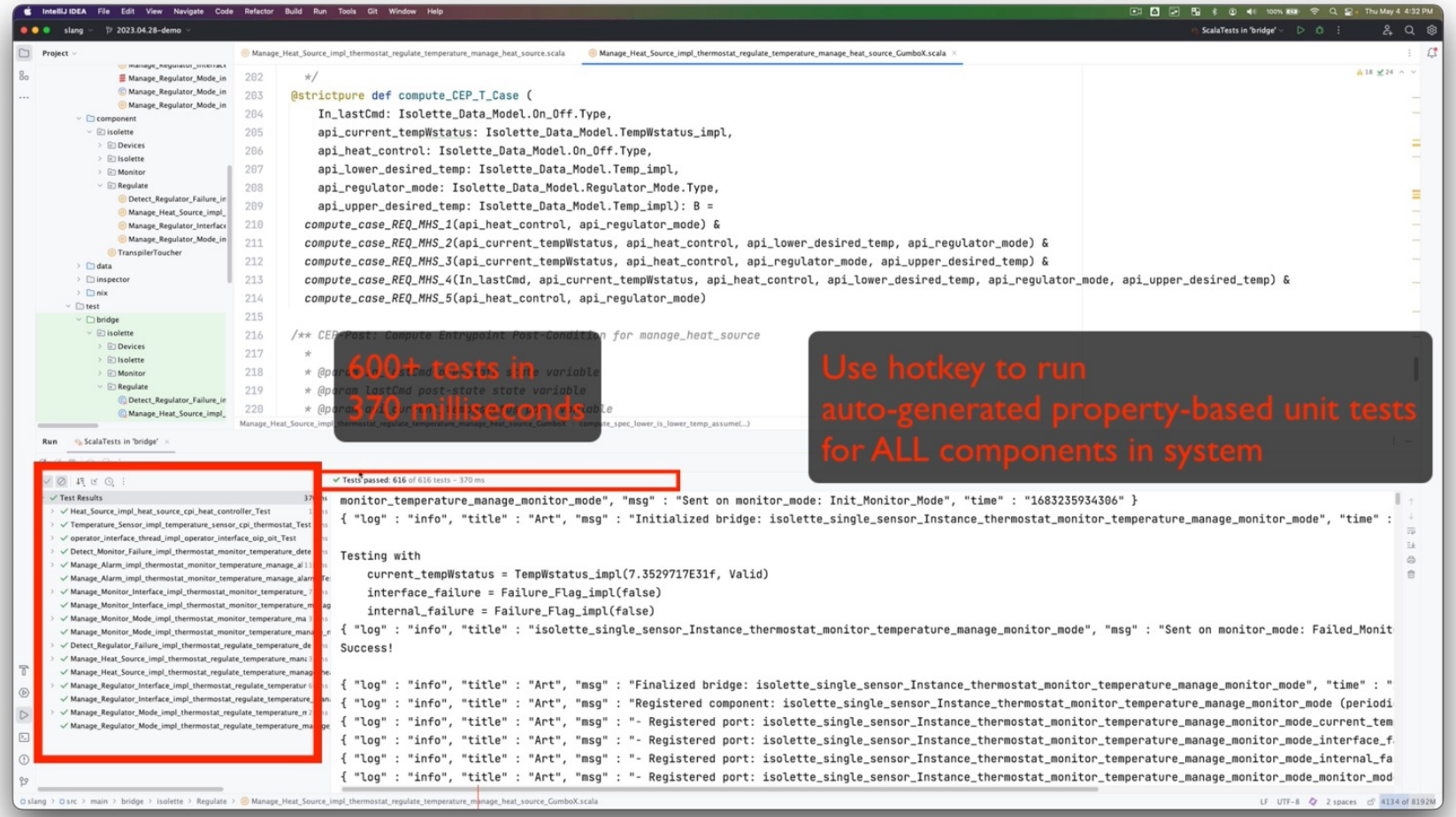
Auto-Generated Property-based Testing Harness

For every thread component, HAMR auto-generates property-based testing infrastructure for inserting values into component input ports and for checking values of output ports.



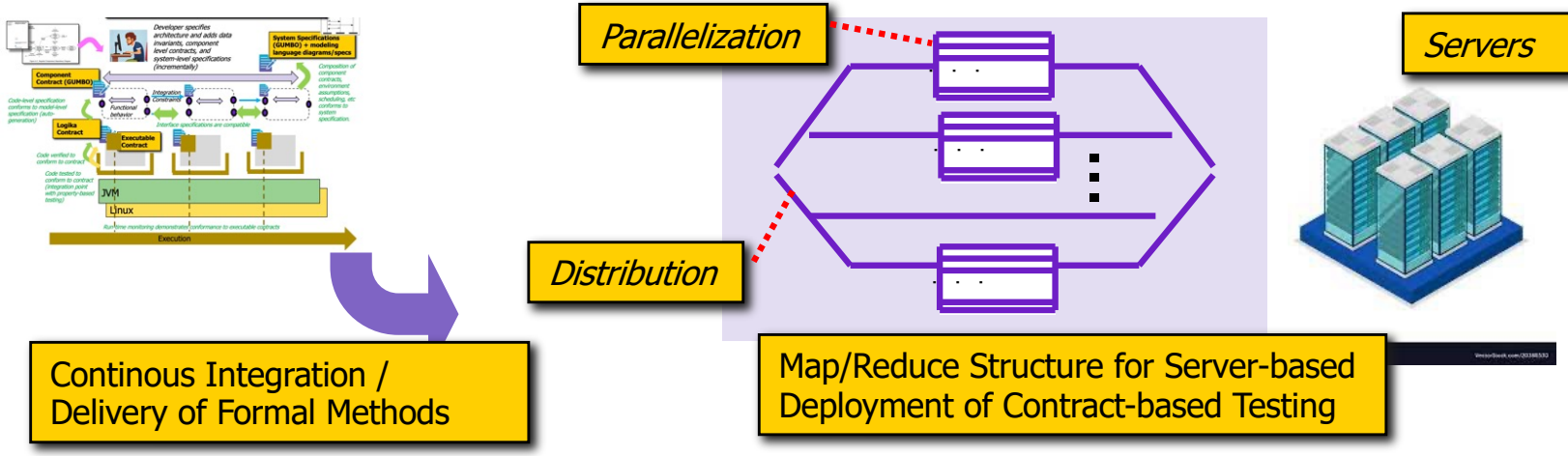
Demo

Property-based Testing



Scaling Up - Server-Based Deployment

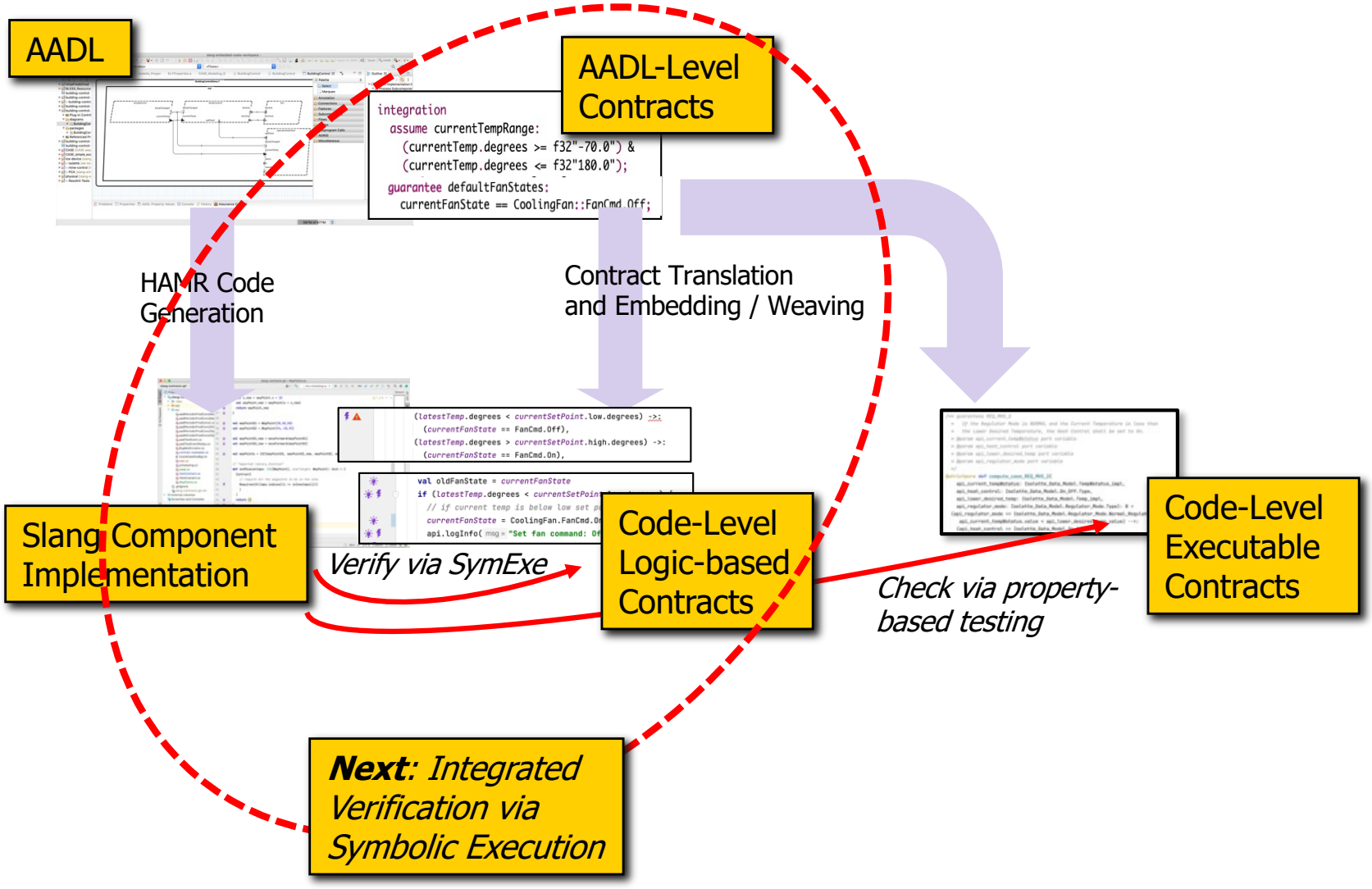
HAMR generates a server-based deployment to run the framework in a distributed/parallel fashion...



- Random generators and contract-based tests are farmed out to a configurable family of servers
- Test vectors and results are serialized for flexible deployment, reporting, and replay of the tests
- Currently hosted using our Jenkins setup, but easy for HAMR to automatically generate deployment scripts, e.g., for AWS, in the future

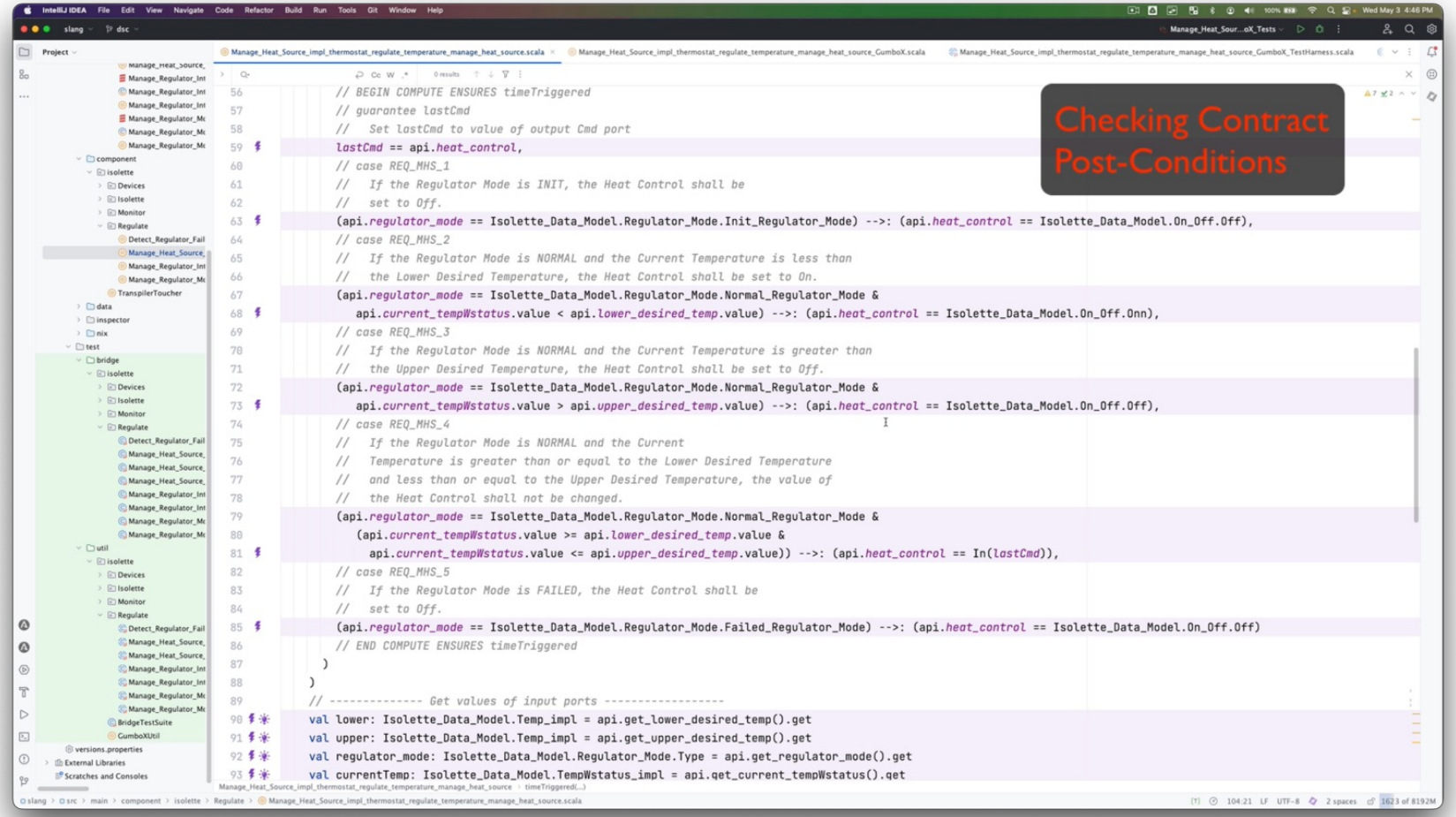
Application:
Usable, Workflow Integrated Verification

KSU / Adventium Labs (now Galois) DARPA SBIR Phase II...



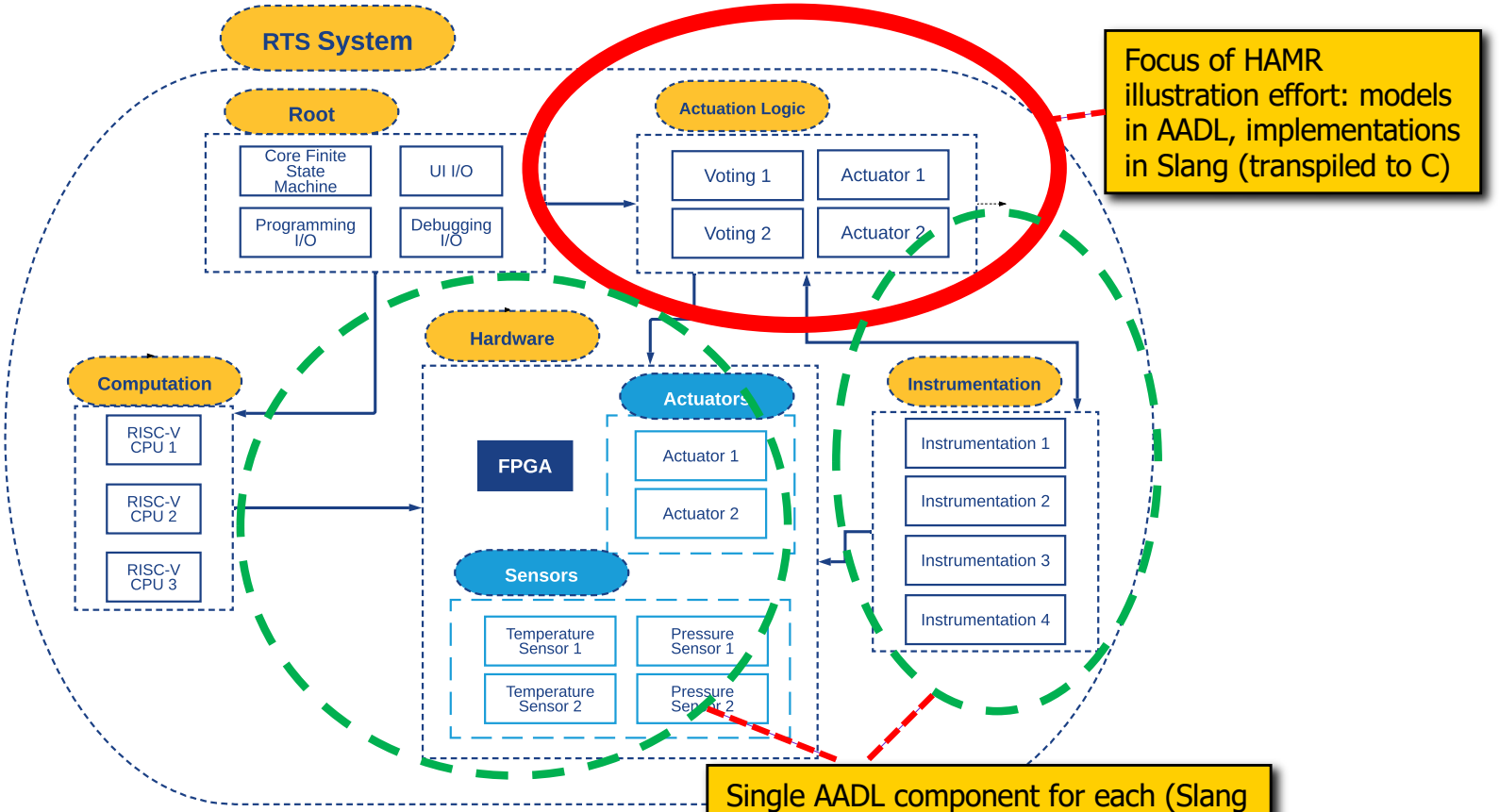
Demo

Verification against contracts using Symbolic Execution



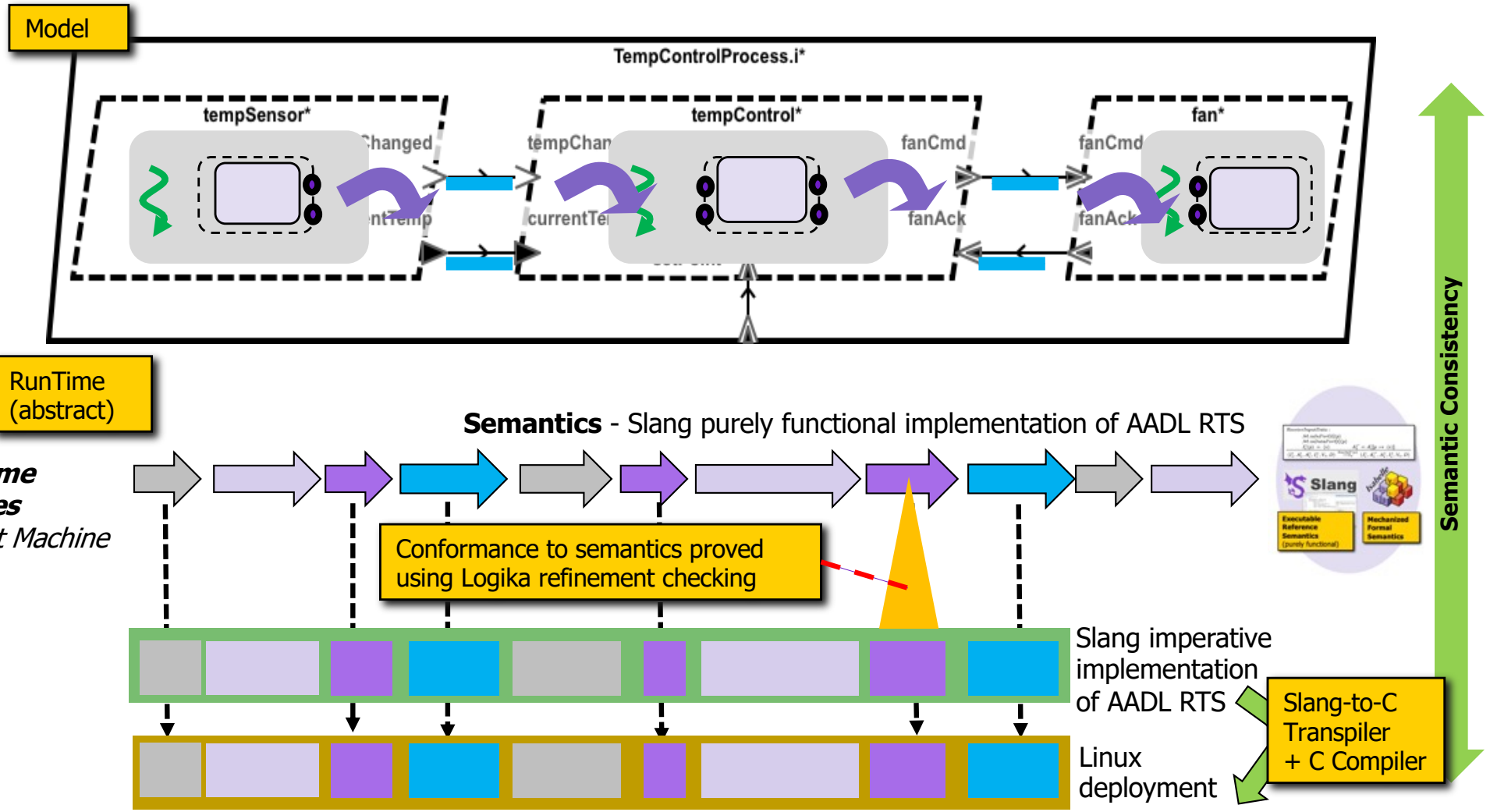
Galois / NRC RTS Example

This methodology has also been applied end-to-end (down to JVM, Linux, seL4) for Galois' open source implementation of a **nuclear reactor trip system** (US NRC funded work) – *available end of May*



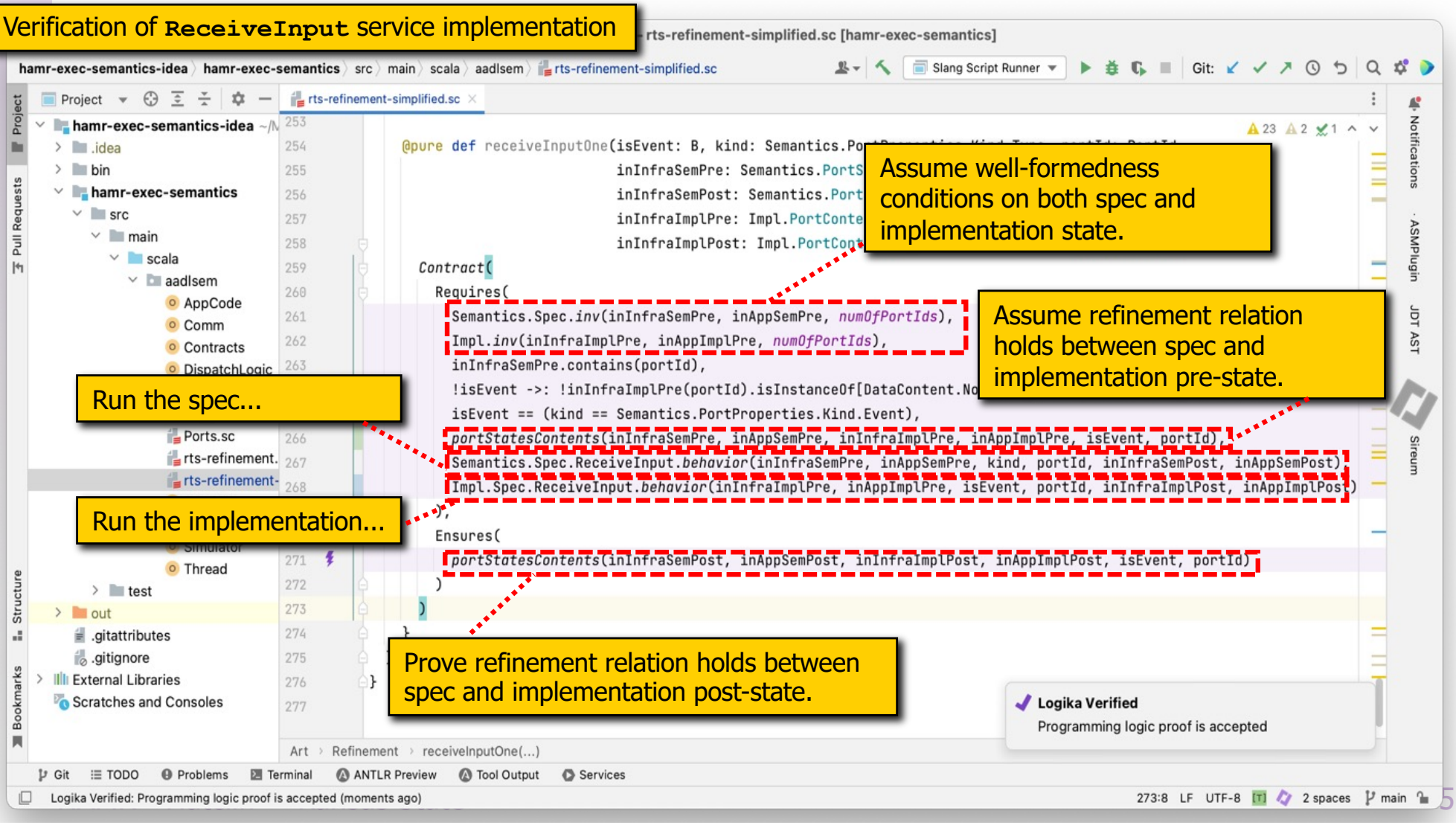
Verifying Deployed Infrastructure Code

Ongoing work: Using Logika refinement checking to verify that deployed infrastructure satisfies the semantics...



Automatically Verifying AADL RTS Implementation

Using Logika refinement checking, we can automatically verify that Slang-based implementation of AADL run-time services conforms to Slang purely functional refence semantics



Conclusion

HCSS Theme: "*Semantically Rigorous and Integrated High-Level Abstractions"*

- Foundations provided by a formal / mechanized semantics of AADL Run-Time Services
  - Semantics artifacts will be in the HAMR release in mid-summer
  - Next steps: adding timing information, code-generation verification with connections to seL4 proofs
- Developer-friendly tool integrated contract framework for both testing and verification
  - Already available in HAMR distribution
- Continuing work on system-level verification and testing
- Looking for funding to develop additional HAMR backends for DoD platforms of interest