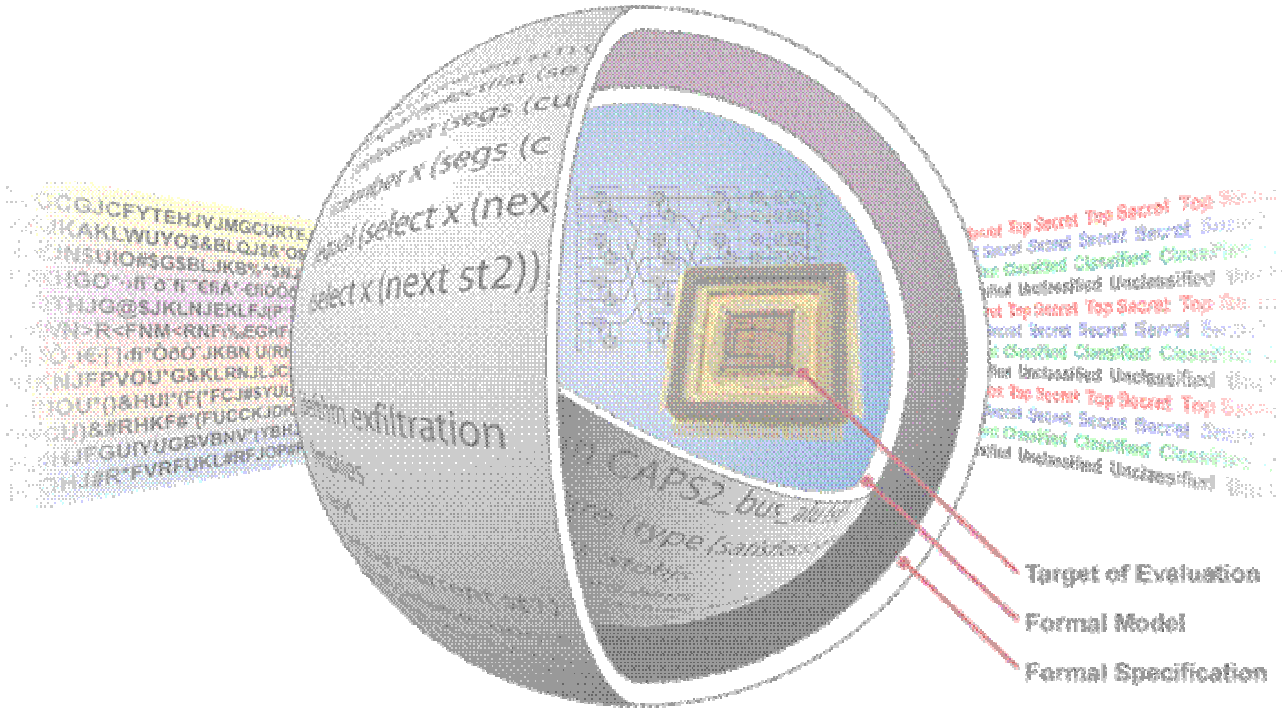
Formal Verification of AAMP7 Intrinsic Partitioning



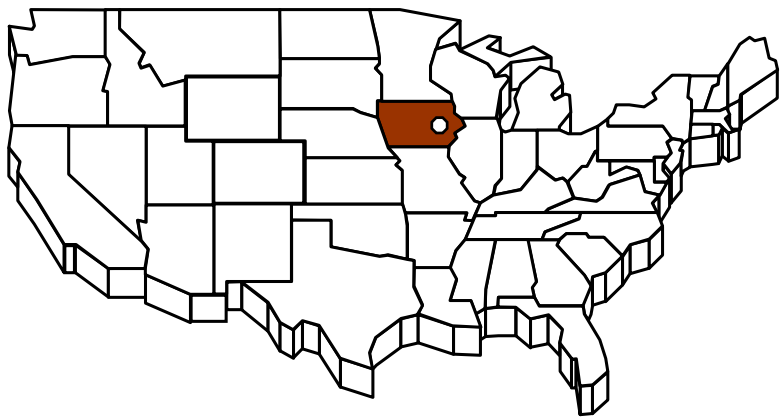
Rockwell Collins Advanced Technology Center
David Greve, Raymond Richards, Matthew Wilding

April 13-15, 2004

Rockwell Collins

- **Advanced Communication and Aviation Equipment**
  - Air Transport, Business, Regional, and Military Markets
  - $2.5 Billion in Sales

- **Headquartered in Cedar Rapids, IA**
  - 14,500 Employees Worldwide
  - Advanced Technology Center
    - Advanced Computing Systems



RCI Advanced Technology Center

Commercial Systems

Government Systems

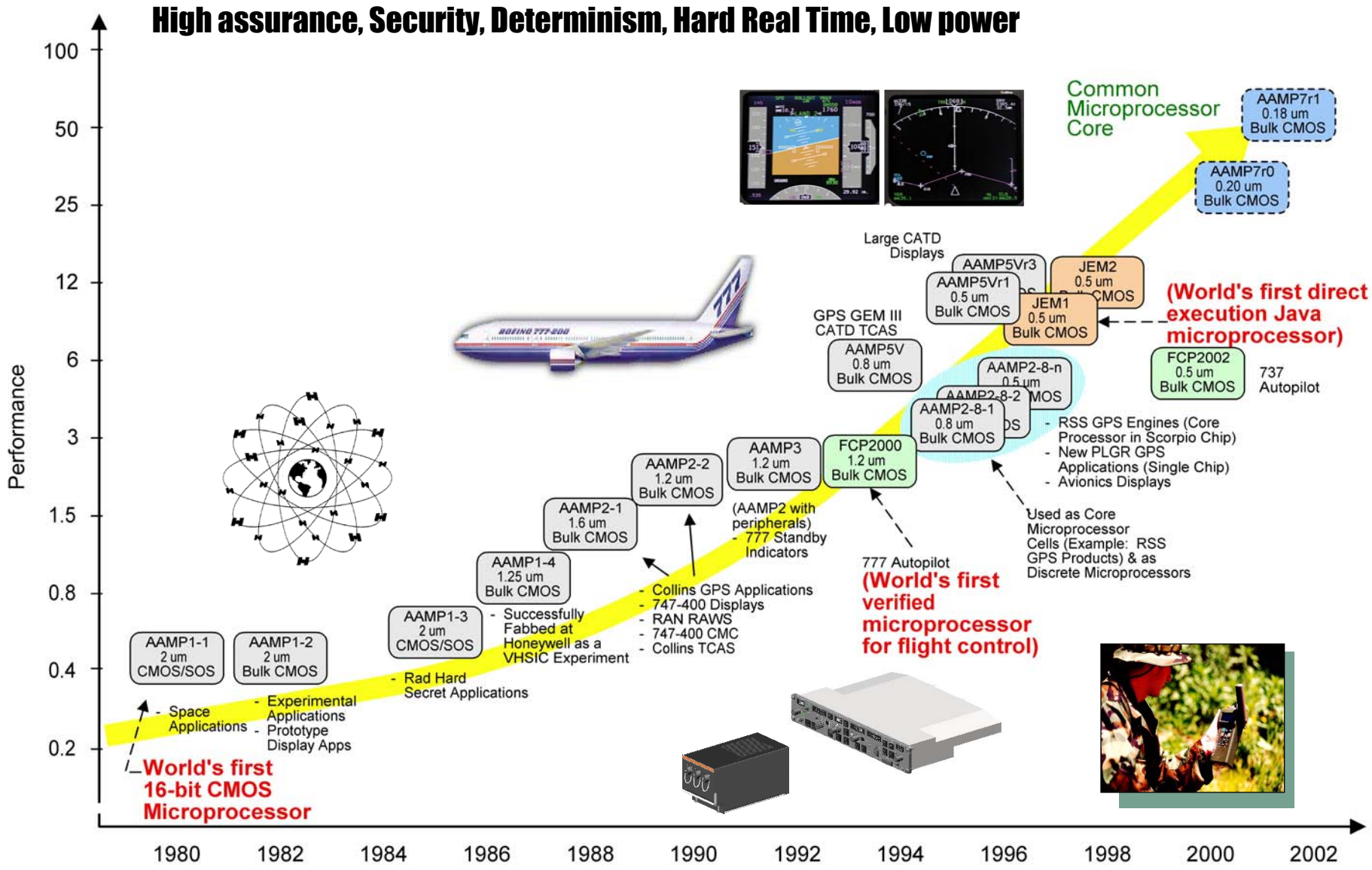
Advanced Technology Center

- The **Advanced Technology Center** (**ATC**) identifies, acquires, develops and transitions value-driven technologies to support the continued growth of Rockwell Collins.
- The **Advanced Computing Systems** department addresses emerging technologies for high assurance computing systems with particular emphasis on embedded systems.
- The **Automated Analysis** section applies mathematical tools and reasoning to the problem of producing high assurance systems.

Outline

- AAMP7 Intrinsic Partitioning
- Separation Kernel Formal Security Policy
- AAMP7 low-level design model
- Proof Architecture
- Future Efforts

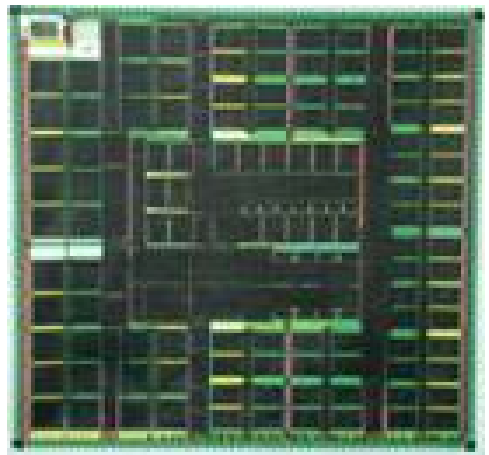
RC Microprocessor Technology



The AAMP7

AAMP7 microprocessor

- High Code Density (2:1 Over CISC, 4:1 Over RISC)
- Low Power Consumption
- Long life cycle relative to other commercial processors
- Screened for full military temp range (-55 C to +125 C)
- Supports legacy software applications
- Implements *intrinsic partitioning*



Intrinsic partitioning

- Computing Platform Enforces Data Isolation
- "Separation Kernel in Hardware"

AAMP7 use

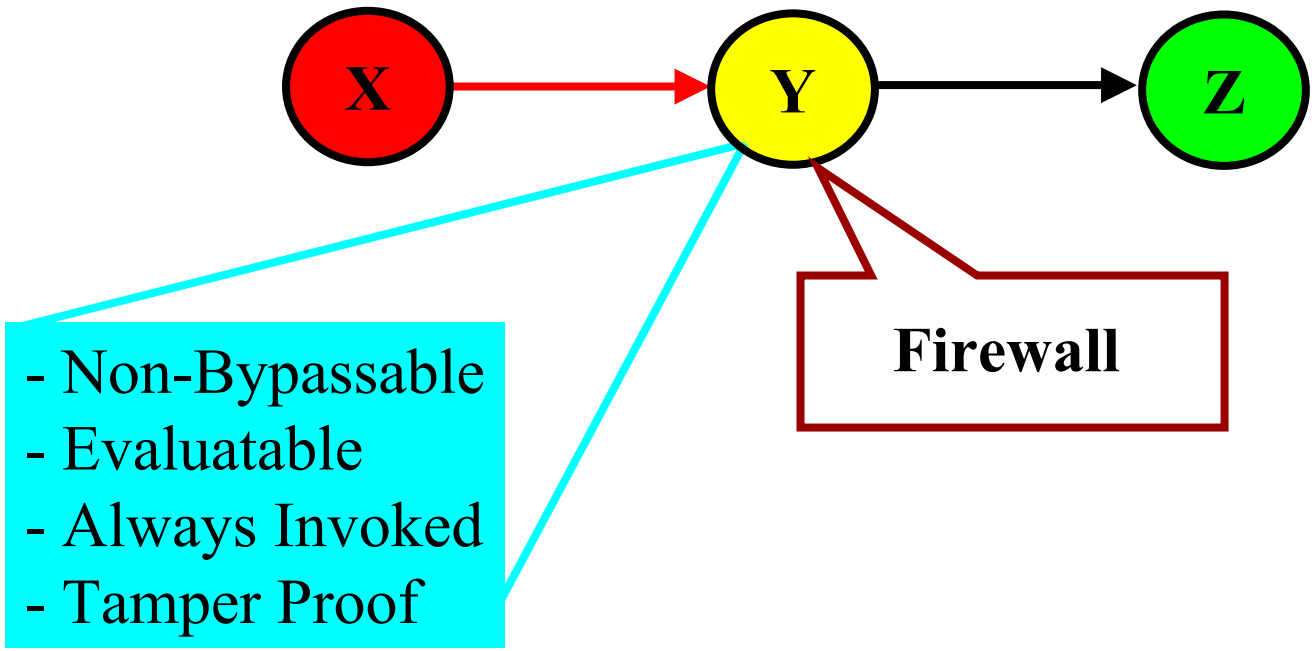
- AAMP7 to be used in variety of applications that require separation of data at different classification levels.
- Formal verification may be needed for system certification

Separation Kernel

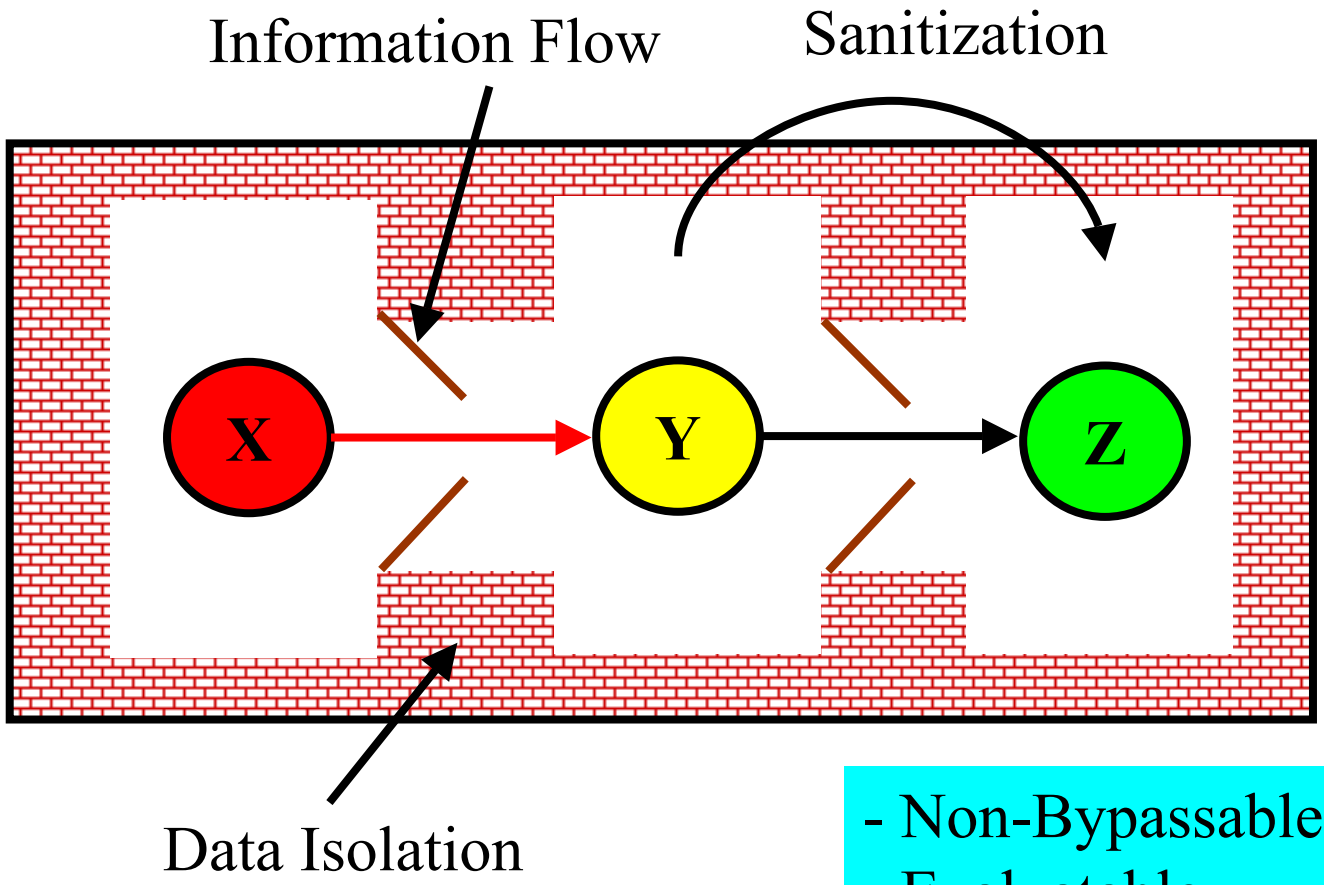
- **Concept First Published in 1980's**
  - **Building Block for Secure Systems**
  - **Decomposes Challenge of Building Secure System**
    - **Allows Applications to Enforce and Manage Own Security Policy**
  - **Provides High Assurance Separation**
- **Effective Security Policies Must Be NEAT**
  - **Non-Bypassable**
  - **Evaluatable**
  - **Always Invoked**
  - **Tamper Proof**
- **Separation Kernels Support Security Policies through**
  - **Information Flow Control**
  - **Data Isolaton**
  - **Sanitization (Periods Processing)**

"Security is about separation
Computers are about sharing"
Brian Snow, Dept. of Defense
April, 2003

Application Level Security Policy

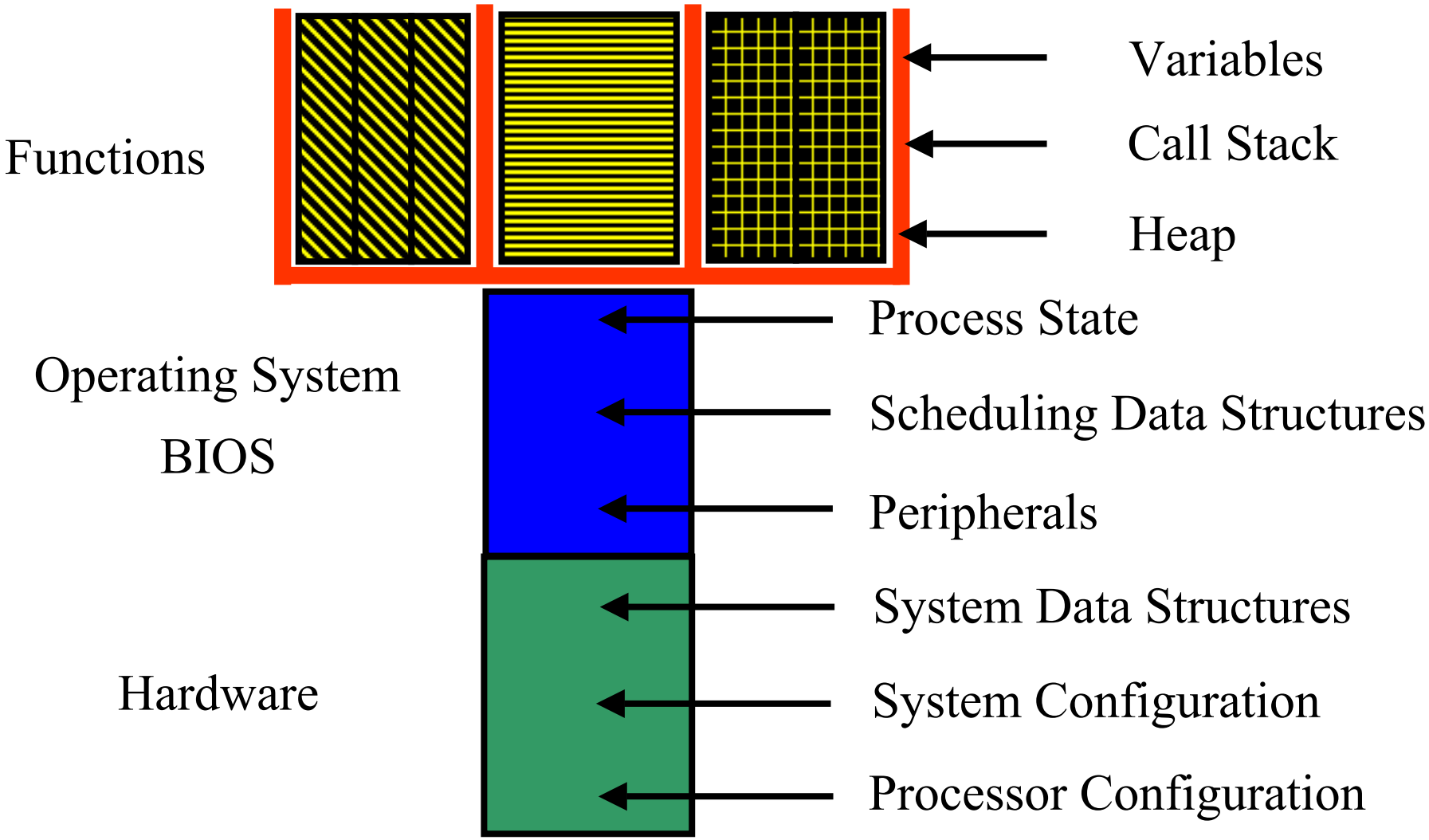


Separation Kernel Services



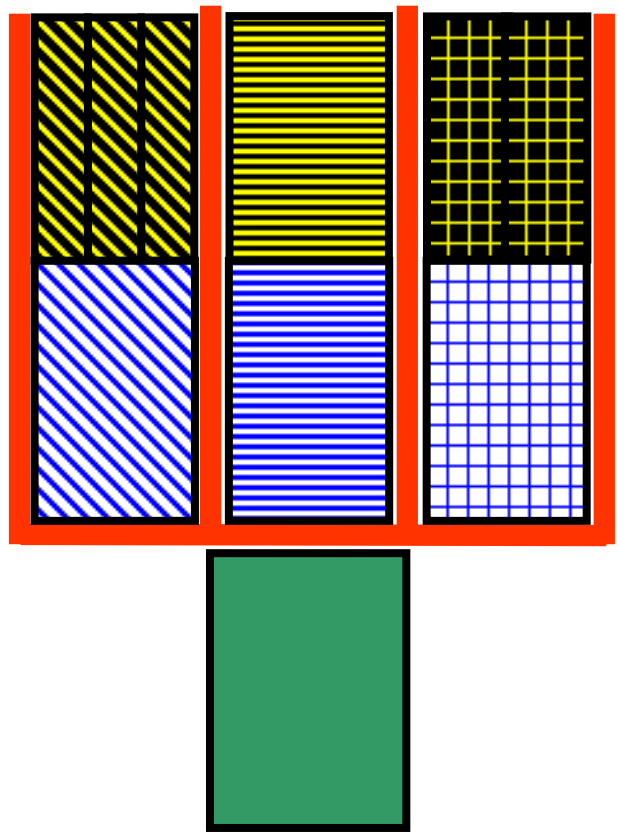
- Non-Bypassable
- Evaluatable
- Always Invoked
- Tamper Proof

Multi-Tasking OS



Intrinsic Partitioning

- **Microcoded Security Kernel**
  - Minimal Code, Functionality, and State
  - Analyzable, Fast, and Efficient
  - Implemented by the AAMP7
- **Simple Data Structures**
  - Supports “Virtual Machine” Partitioning
    - Each Partition Has Its Own Operating System
  - Hierarchical Scheduling
- **Dedicated Interrupts**
  - Partition Switch Interrupt
  - Power Down Warning Interrupt
  - Access Violation Interrupt
  - Partition-Aware Interrupts



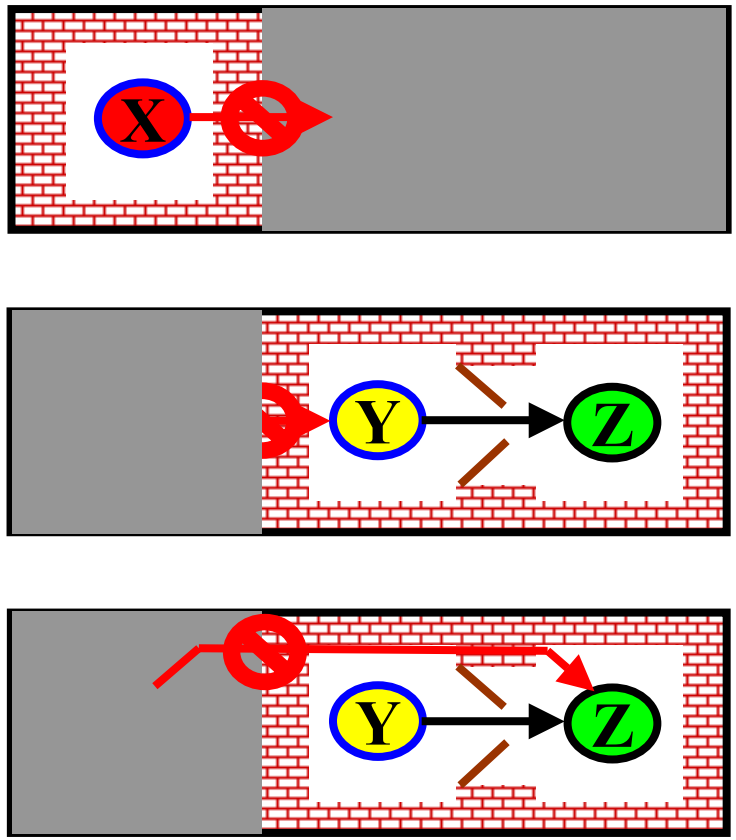
- **Supports High Assurance, Evaluatable Architectures**

Outline

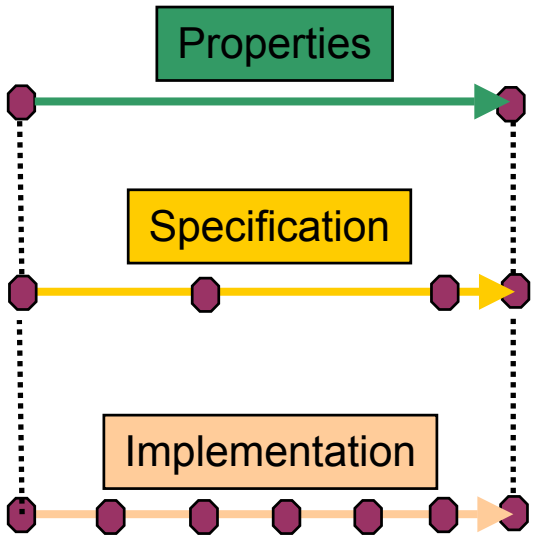
- AAMP7 Intrinsic Partitioning
- Separation Kernel Formal Security Policy
- AAMP7 low-level design model
- Proof Architecture
- Future Efforts

Formalized Separation Kernel Security Policy

- **Informal Security Policy**
  - Information Flow Control
  - Data Isolation
  - Sanitization
- **Need for Formalize**
  - Precise Mathematical Description
  - Suitable for Formal Analysis
  - Adapting Existing Security Policy
- **Formal Security Policy**
  - Infiltration
  - Exfiltration
  - Mediation



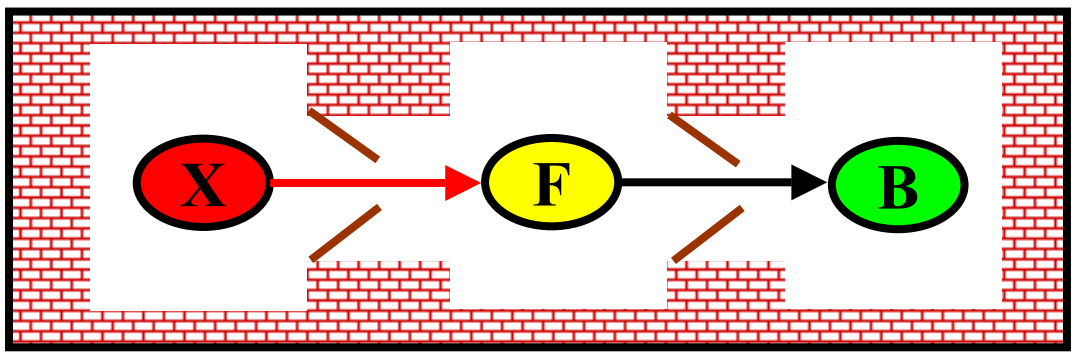
Validation: Using Specifications



- Showing correctness means relating a specification for how a system should behave to an implementation description.
- Specifications can be “stacked” by using a specification in one proof as the implementation of another
- Specifications can be both “proved” and “used”

Validating our Security Policy by Using It

- We changed our separation kernel security policy many times because we could not prove anything with our early attempts at specifying separation.
- Only an interesting use of our separation kernel specification would meaningfully validate it, and complete separation just won't do.
- We formalize and prove correct a firewall that is implemented using a separation kernel



Formal Security Policy

- **The firewall proof led us to our current formalization of correctness for the separation kernel**

```
(defthm separation
  (let ((segs (intersect (dia seg) (segs (current st1)))))
    (implies
     (and
      (equal (selectlist segs st1) (selectlist segs st2))
      (equal (current st1) (current st2))
      (equal (select seg st1) (select seg st2)))
     (equal
      (select seg (next st1))
      (select seg (next st2))))))
```

"A Separation Kernel Formal Security Policy in PVS"
John Rushby, SRI

- **This specification is enough to prove everything**
  - **Example firewall correct**
  - **infiltration, exfiltration, and mediation**

Described in more detail in
ACL2 workshop paper reprinted
In the HCSS proceedings

Outline

- AAMP7 Intrinsic Partitioning
- Separation Kernel Formal Security Policy
- AAMP7 low-level design model
- Proof Architecture
- Future Efforts

Purpose

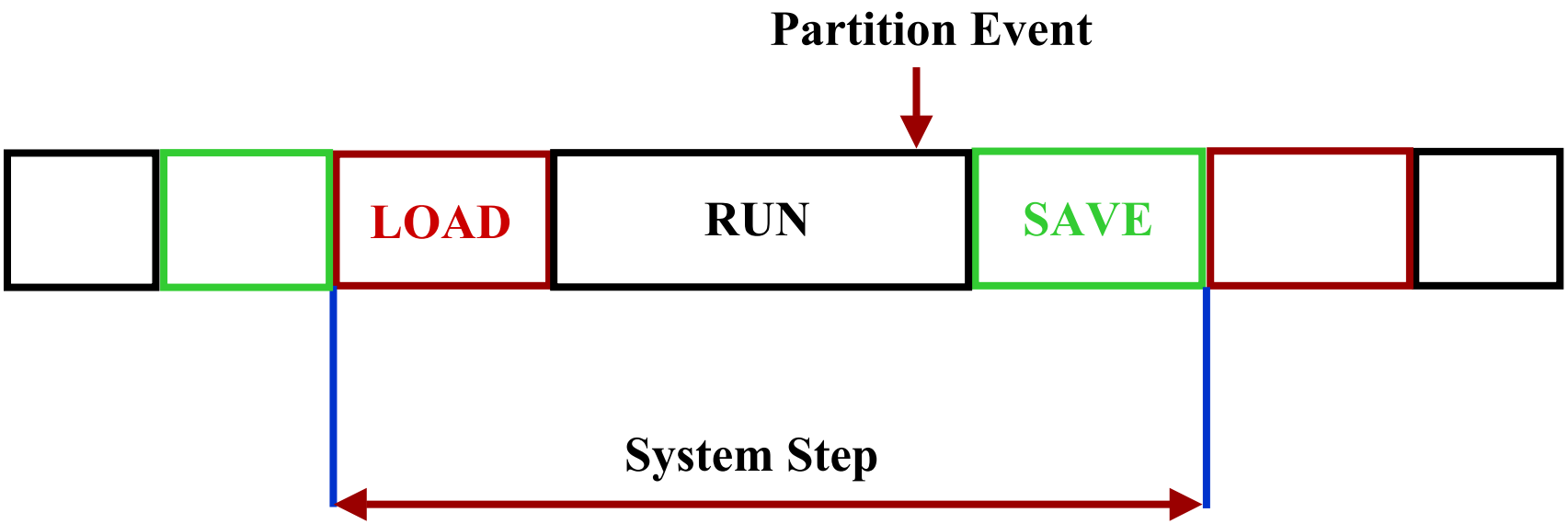
- **Model the Behavior of the AAMP7 Privileged Microcode**
  - Think of it as a simulator
  - Level of Detail: A Microcoder's View of the World
- **Acknowledge Partition Events**
  - Partition Interrupt
  - Power-Down Interrupt
  - Cold/Warm Start
- **Perform Partition Scheduling**
  - Load Partition State
  - Run Partition Code
  - Save Partition State
- **System Bookkeeping**
  - Maintain Global System State/Status
  - Mark Power-Down List

Low-Level Model

- **Model Written in Functional Common Lisp (ACL2)**
  - **Parenthesis, LET bindings and prefix notation**
  - **Just Another Programming Language**
    - **A Language with no Side-Effects**
  - **Lisp Macros Employed to Simplify Presentation**
    - **Minimize Parenthesis**
- **Includes State Pertinent to Space Partitioning**
  - **RAM**
  - **Selected PMU Registers**
  - **Selected Processor Registers**
  - **Selected Processor Inputs**
- **Models Behavior Pertinent to Space Partitioning**
  - **Loading, Clearing and Saving Partition State**
  - **Loading and Clearing PMU**
  - **Setting and Clearing Privileged (System) Bit**

Partition Execution Model

- Begins with the Loading of the Current Partition
- Ends with the Saving of the Current Partition State
  - And the updating of the value of “current partition”



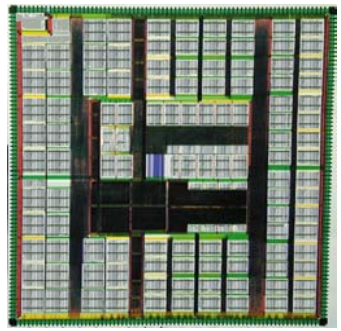
Top-Level Function

```
(defun Step-System (spex st)
   (%
      (st = (cond
             ((Imminent-Asynchronous-Event? st)
                (bsp spex))
             ((idle_system? st) (idle-system st))
             (t
                  (%
                    (st = (load-partition-state st))
                    (st = (run-until-partition-event st))
                    (if (Power-Down? st)
                          (do-power-down st)
                      (do-partition-switch st))
                    ))))
      (step-free-running-systems st)
   )
)
```

Model is approximately 3000 lines of code

Validation of Low-Level Model

- **Validation of Low-Level Model**
  - Is Low-Level Model an Accurate Representation
  - No "Proof of Correctness"
  - Must be done informally
- **Number of Validation Possibilities**
  - Correct by Construction
  - Simulation
  - Code-to-Spec Review
- **Code-to-Spec Review**
  - Verify that the "code" implements the "specification"
  - Requires some understanding of both
  - Implementers have a "meeting of the minds" with verifiers
- **More Detailed Model is Easier to Validate**



?

=

Microcode

Formal Model

```
;=== ADDR: 052F

   (st. ie = nil)
   (Tx = (read32 (vce_reg st) (VCE.VM_Number)))

;=== ADDR: 0530

   (st. Partition = Tx)

;=== ADDR: 0531

   (TimeCount = (read32 (vce_reg st) (VCE.TimeCount)))

;=== ADDR: 0532

   (PSL[0]= TimeCount st)
```

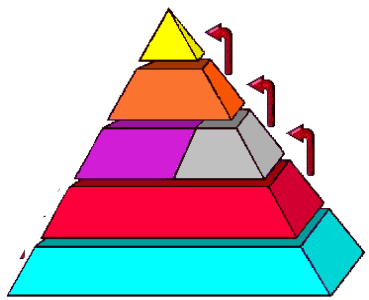
```
;-------------------------------------------------------------
;=== ADDR: 052F
A]
   CONT ;
H] clear InterruptEnable, read VM number
   IE=0                              \
                                     T=BADDR.READ32(T) ;
L] hold VM number (a.k.a. partition number) in T
                                     \
                                     T=T ;
;-------------------------------------------------------------
;=== ADDR: 0530
A]
   CONT ;
H] load VM number into MSQ partition register
   P=T                               \
                                     T=T ;
L] unused
                                     \
                                     T=T ;
;-------------------------------------------------------------
;=== ADDR: 0531
A]
   CONT ;
H] locate TimeCount in VCE
   R=VCE.TimeCount  W=RFB(VCE_REG)   \
                                     T=R+W ;
L] read TimeCount
                                     \
                                     T=BADDR.READ32(T) ;
```

Outline

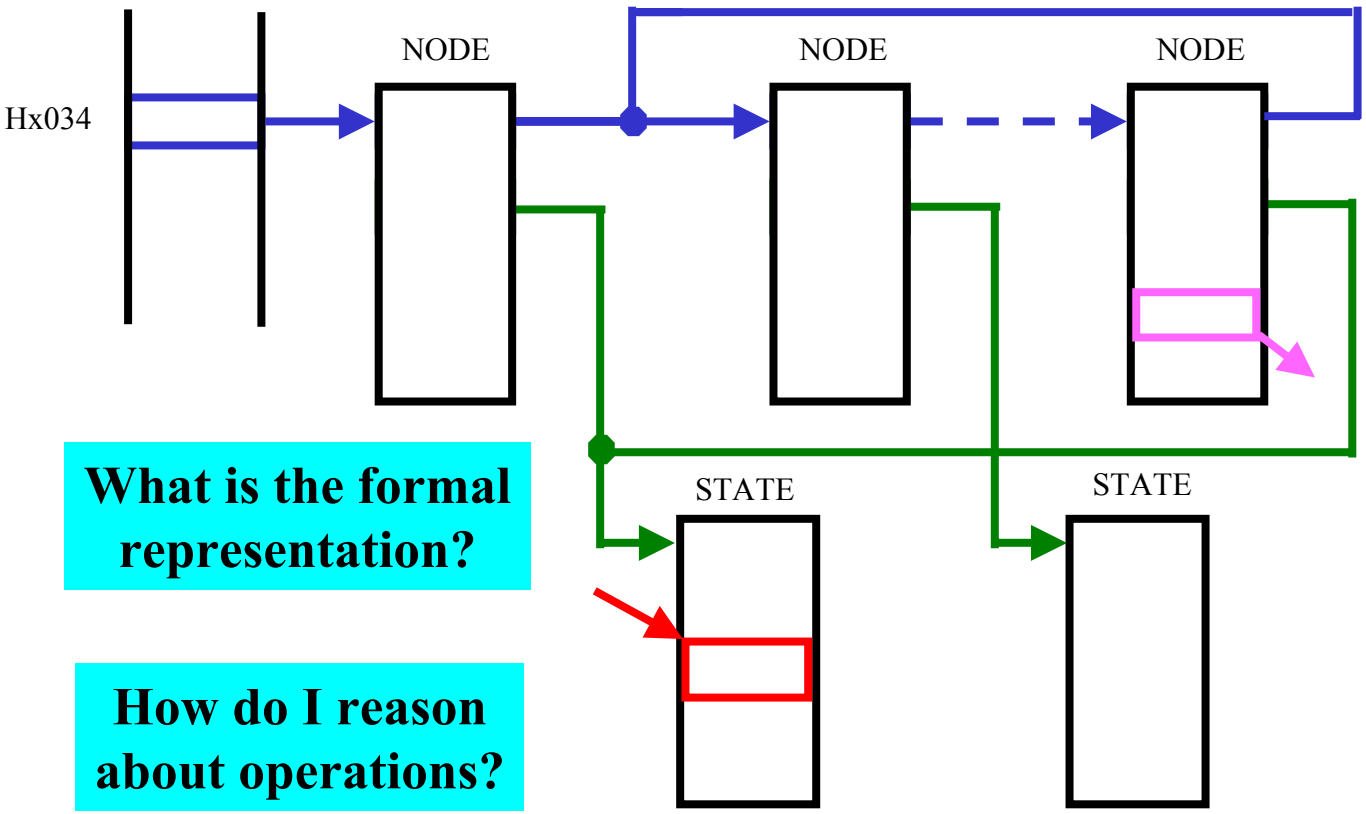
- AAMP7 Intrinsic Partitioning
- Separation Kernel Formal Security Policy
- AAMP7 low-level design model
- Proof Architecture
- Future Efforts

Proof Outline

- **Validation of Formal Security Policy**
  - Firewall Proofs
- **Proof that Abstract Model satisfies Security Policy**
  - High Level Proofs are less detailed, easier
- **Proof of correspondence with Low-Level Model**
  - Low-Level Model Implements Abstract Model
  - Implementation Details make this more difficult
- **Code-to-Spec review of microcode**
  - Verify that the Low-Level model accurately reflects the behavior of the microcode
  - Implementation Details make this easier

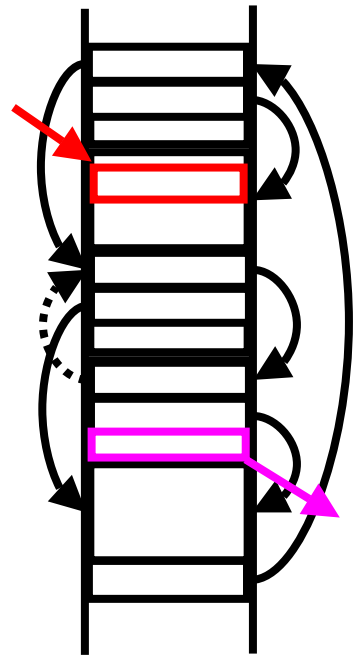


Primitive Data Structures



What is the formal representation?

How do I reason about operations?



GACC: Generalized Accessor Library

- A means of describing linearized data structures
  - Really just a list of addresses
  - Distinguishes pointer and data locations
- Rules for resolving read/write operations
  - (read list1 (write list2 values ram)) = (read list1 ram)
  - (read list1 (write list1 values ram)) = values
- Rules for preserving structure
  - Writes to data locations don't change data structure shape
- Efficient rules for disjoint/subset/unique relations
  - Linear Time/Space
  - Free-variable matching
  - Meta-rules

Theorem Proof

- We have checked all proofs using the ACL2 theorem prover.
- This has been a substantial effort
  - Major technical innovation of this work is the automation of reasoning about data structures
  - Checking all the proofs requires about 4 hours using ACL2 2.7 built on GCL running on our fastest PC.
- ACL2 is freely available, and will be used to replay our proofs, which we are providing in a machine-readable format for that purpose.
- Getting ACL2 to generate these proofs was the focus of 95%+ of our efforts, but the resulting proofs are easy to evaluate.

Final Formal Separation Theorem

We have proved a mathematical theorem about the AAMP7's intrinsic partitioning mechanism.

```
(implies
   (and
    (secure-configuration spex)
    (spex-hyp :any :trusted :raw spex fun::st1)
    (spex-hyp :any :trusted :raw spex fun::st2))
   (implies
    (let ((abs::st1 (lift-raw spex fun::st1))
          (abs::st2 (lift-raw spex fun::st2)))
      (and
       (let ((segs (intersection-equal
                        (dia-fs seg abs::st1)
                        (segs-fs (current abs::st1) abs::st1))))
           (equal (raw-selectlist segs abs::st1)
                  (raw-selectlist segs abs::st2)))
       (equal (current abs::st1)(current abs::st2))
       (equal (raw-select seg abs::st1)(raw-select seg abs::st2))))
    (equal
     (raw-select seg (lift-raw spex (fun::next spex fun::st1)))
     (raw-select seg (lift-raw spex (fun::next spex fun::st2))))))
```

This particular theorem is an instantiation of the basic separation theorem is described in the HCSS proceedings.

Our ACL2 model represents the low-level design of the AAMP7. Trusted-mode operations that must respect intrinsic partitioning are modeled in sufficient detail to be related to the AAMP7 microcode implementation.

Outline

- AAMP7 Intrinsic Partitioning
- Separation Kernel Formal Security Policy
- AAMP7 low-level design model
- Proof Architecture
- Future Efforts

Future Efforts

- **Continued Library Development**
  - Data Structure Representation
  - Proof Automation is Essential
- **Increased Automation: vFaat**
  - Proof Structure Management
  - Data-Flow Analysis
  - Allows Analysis of Larger, More Complex Systems
- **Stronger Links to Development Tools: SHADE**
  - Instruction-Level Proofs
  - Certifying Compilers