# vFaat: von Neumann Formal Analysis and Annotation Tool


**David Greve**
**Dr. Matthew Wilding**

**Rockwell Collins**
**Advanced Computing Systems**
**Cedar Rapids, IA**

**dagreve@rockwellcollins.com**
**mmwildin@rockwellcollins.com**


**April 2003**

# Overview

- **Rockwell Collins History**
  - Have 10+ years experience in applying Formal Methods
    - Have embraced and extended a variety of techniques
- **Observations**
  - Advantages to Accurate Low-Level Models
    - Abstraction can be tedious
  - Domain knowledge
    - Missing from generic theorem provers
    - Can be codified
  - Techniques generalize to
    - Different Verification Targets
    - Different Theorem Provers
- **Conclusion**
  - Judicious use of automation simplifies verification task
    - Improve Productivity, Extend Scope

> vFaat: A collection of tools and techniques to help simplify reasoning about complex systems"

# Outline

- Motivating History
- Observations
- Future Directions

# RC Formal Methods Projects

- AAMP5/AAMP-FV
- JEM1: Symbolic Simulation
- JEM2: Executable Formal Models
- CAPS: High-Assurance Processor
- AAMP7: Intrinsic Partitioning

# AAMP5/AAMP-FV

"Formal Verification of the AAMP5 Microprocessor", NASA Report 1995
"Formal Verification of the AAMP-FV Microcode", NASA Report 1999

- **Goal**
  - Demonstrate use of Formal Methods in Industrial Setting
- **Project**
  - Sponsored by NASA Langley
  - Used PVS and teamed with SRI
  - Abstract representation of instruction execution created
  - Functionally described Microarchitecture
  - Standard Commuting Diagram Proof

# AAMP5/AAMP-FV

- Inspections connected model to implementation
  - Found variety of errors in formal specification
- Was there a better way to validate the model?
  - Better Integration with design process?

- Verified a number of instructions
  - Even found a few bugs
- Brute Force Formalization
  - Microcode specified by hand
  - “Clock functions” crafted by hand
- Automated microcode specification an obvious next step

# JEM1: Symbolic Simulation

**"Symbolic Simulation of the JEM1 Microprocessor", FMCAD-98**

- **Goals**
  - Integration of Formal Models into Design Process
  - Leverage Automated Analysis
  - Detect Microcode Errors (Bug Finding)
- **Project**
  - Specified JEM1 Microarchitecture in PVS
  - Used PVS to execute symbolically the model
    - Generated Symbolic Results for Microcode Basic Blocks
  - Analyzed Symbolic Results as part of Microcode Inspections

# JEM1: Symbolic Simulation

- **Symbolic Results Difficult to Read**
  - Good for detecting data-flow errors (Definition/Use)
    - Unexpected Side-Effects
    - Unexpected Data-Dependencies
- **Demonstrated Effective Use of Automation**
  - Codified knowledge of problem domain
    - Control Flow Analysis Dictated Proof Architecture
  - Employed Automatic Generation of:
    - Function Definitions (uCode)
    - Theorems and Theory Structure (Proof Architecture)
    - Symbolic Results (batch mode PVS)

# JEM2: Executable Formal Models

**"Efficient Simulation of Formal Processor Models", FMSD 2002**

- **Goals**
  - Integration of Formal Models into Design Process
  - Improve Model Validation Technique
    - Replace Microcode Simulator with Executable Formal Model
- **Project**
  - Modeled JEM2 in logic of ACL2
    - Subset of Common Lisp
  - Compiled model to C and linked into GUI development environment

# JEM2: Executable Formal Models

- **Impressive Results**
  - Final simulator ran as fast as original
    - No penalty to developers for using formal model
  - Successfully executed regression tests
    - Guaranteed validity of formal model

- **Exposed ACL2 tool limitations**
  - Model was large and complex
  - Is it possible to reason about such models?

# CAPS: High-Assurance Processor

**"Evaluatable, High-Assurance Microprocessors", HCSS-02**

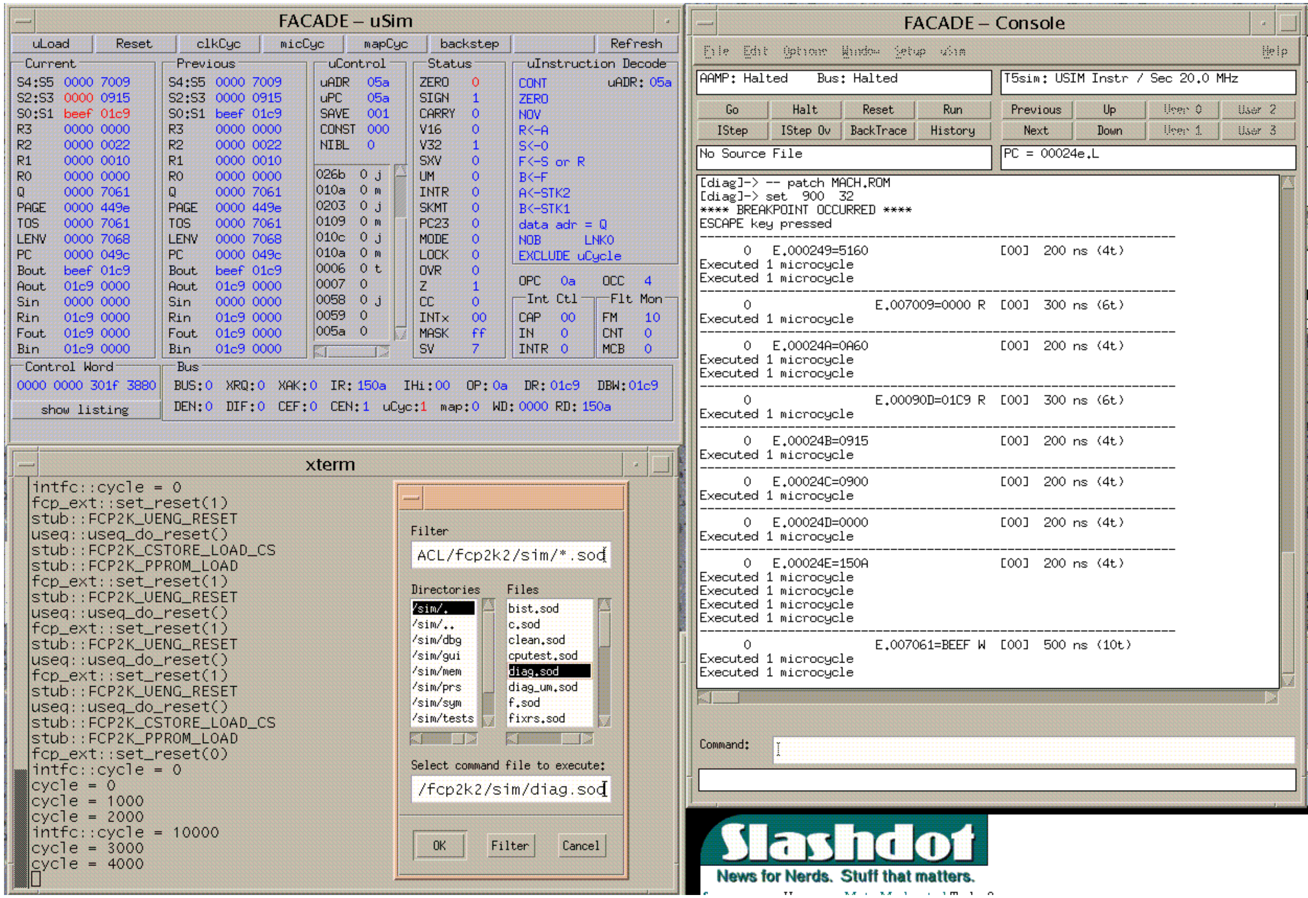
- **Goals**
  - Reason about an Executable Formal Model
  - Perform Instruction Level Proofs of CAPS processor
- **Project**
  - Modeled Microarchitecture of CAPS in ACL2
  - Executed Standard Regression Tests to Validate Model
  - Formalized a set of CAPS instructions
  - Constructed Instruction Level Proofs

# CAPS Microarchitecture Model

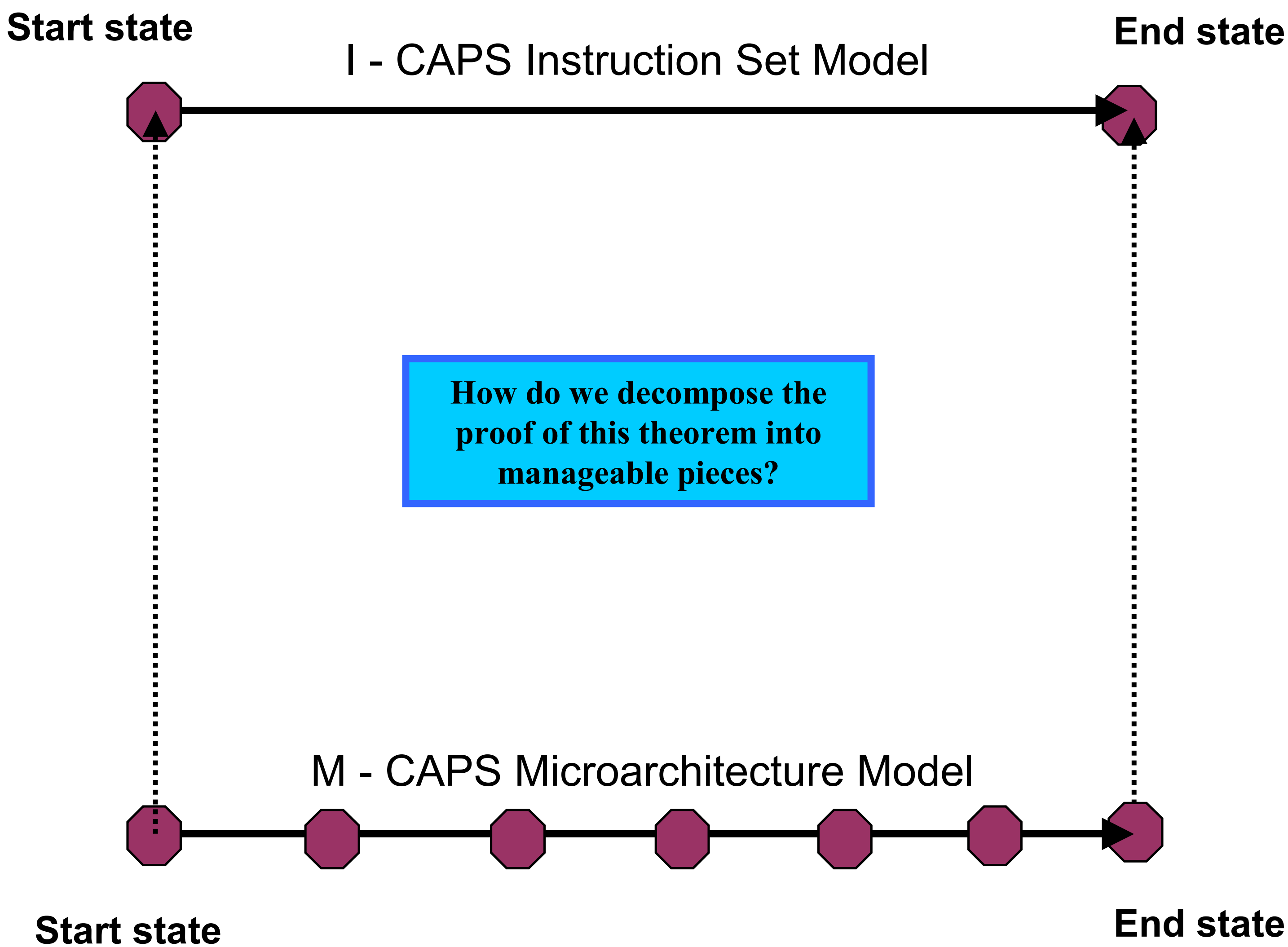
**The ACL2 CAPS uarch model replaces the C model in the CAPS microcode simulator. The replacement is not observable to users.**

***CAPS ACL2 uarch model passes 3-hr standard CAPS regression test!***

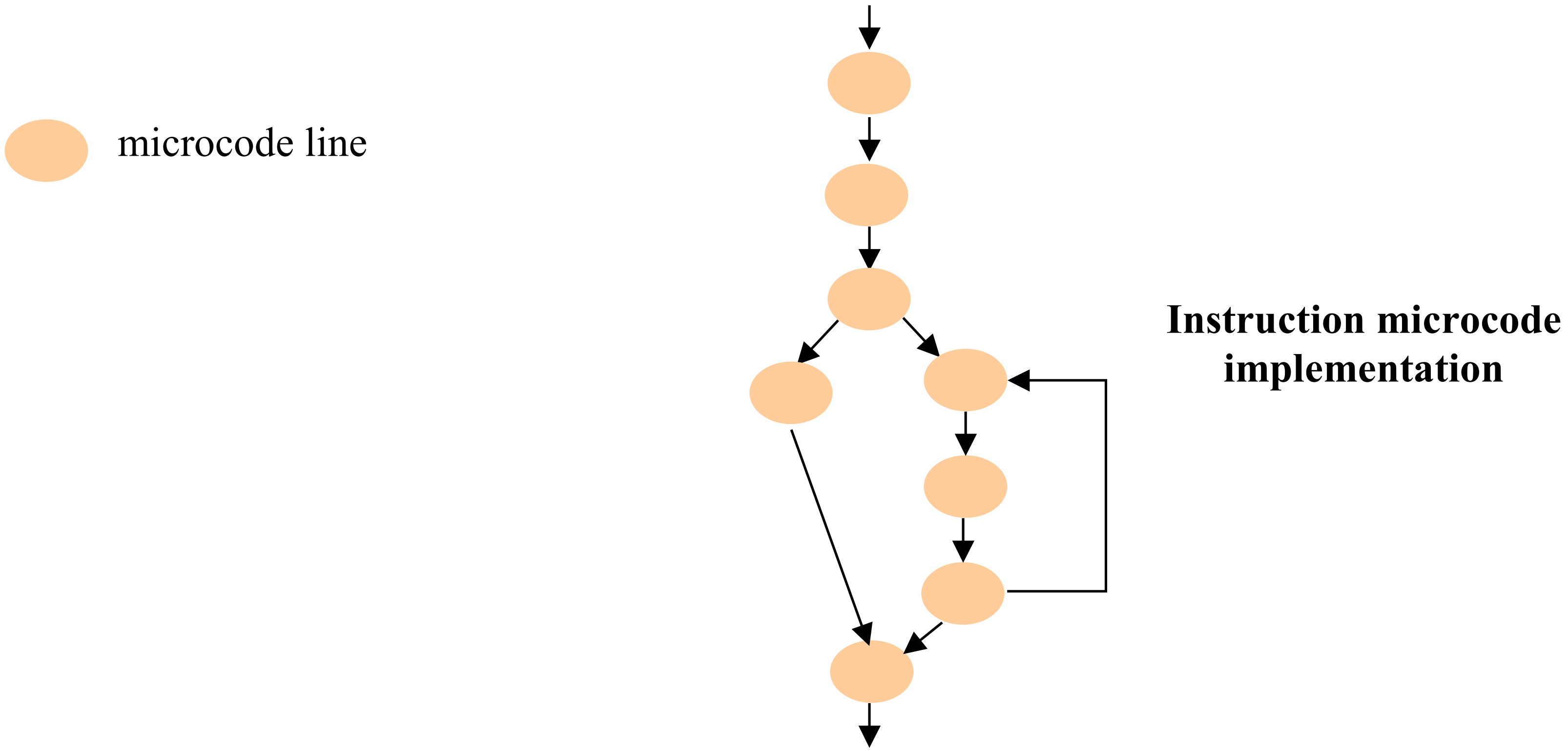
High-speed, formal models provide for evaluatability (looks like C, passes regression tests, integrated into dev process, proofs checked)



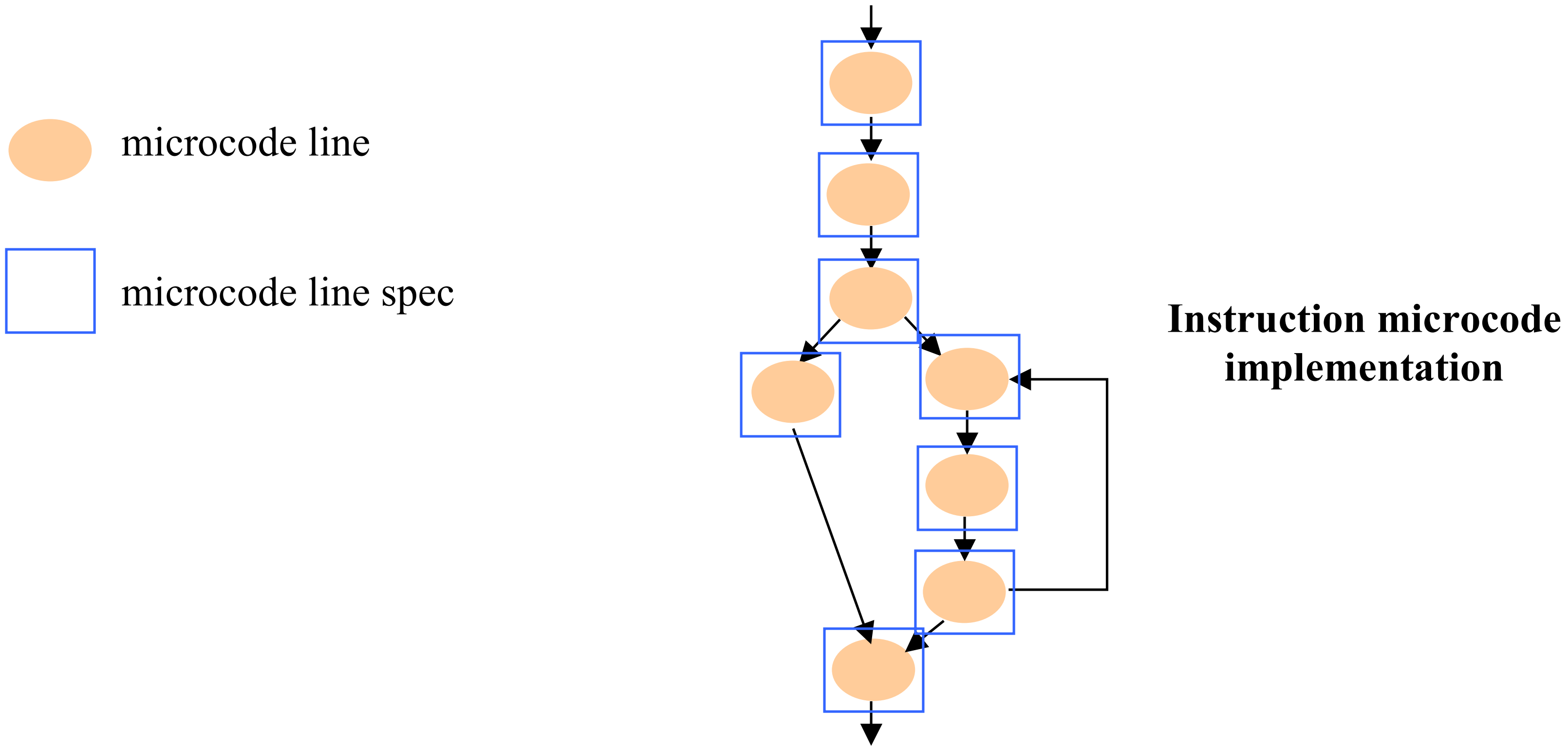
# CAPS Correctness Theorem



# Microcode sequences can be specified and verified in steps.



# Microcode sequences can be specified and verified in steps.

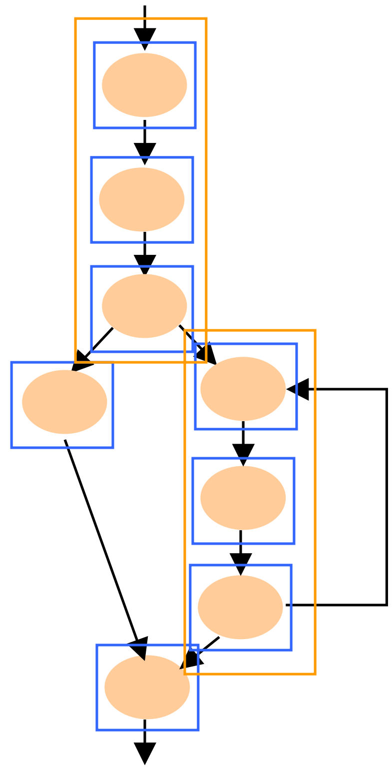


# Microcode sequences can be specified and verified in steps.

microcode line

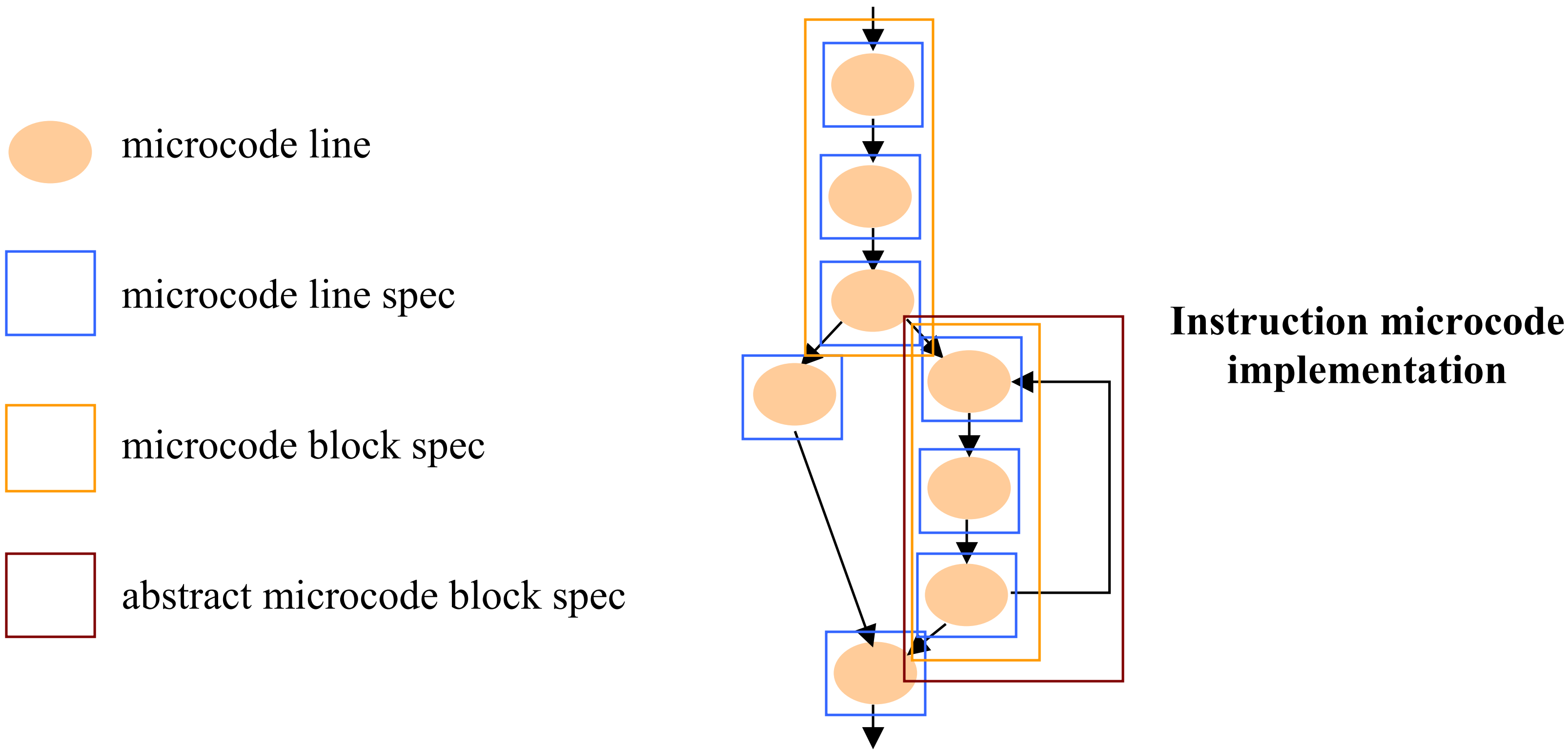
microcode line spec

microcode block spec



**Instruction microcode implementation**

# Microcode sequences can be specified and verified in steps.



# Microcode sequences can be specified and verified in steps.

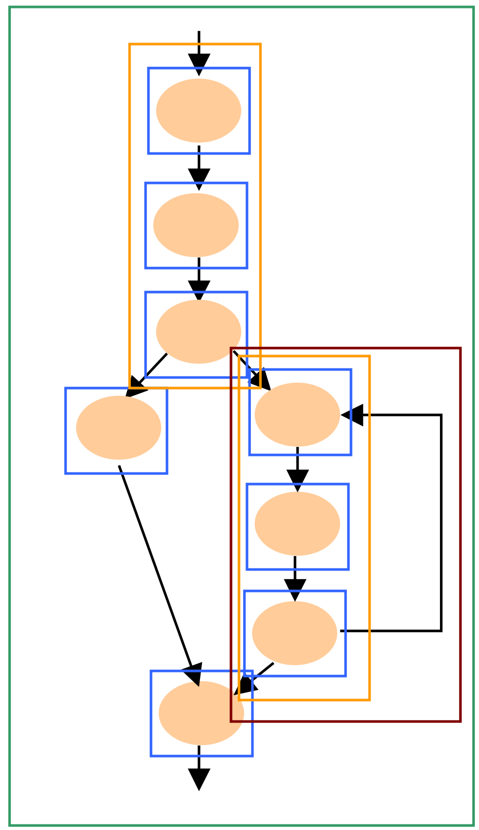
microcode line

microcode line spec

microcode block spec

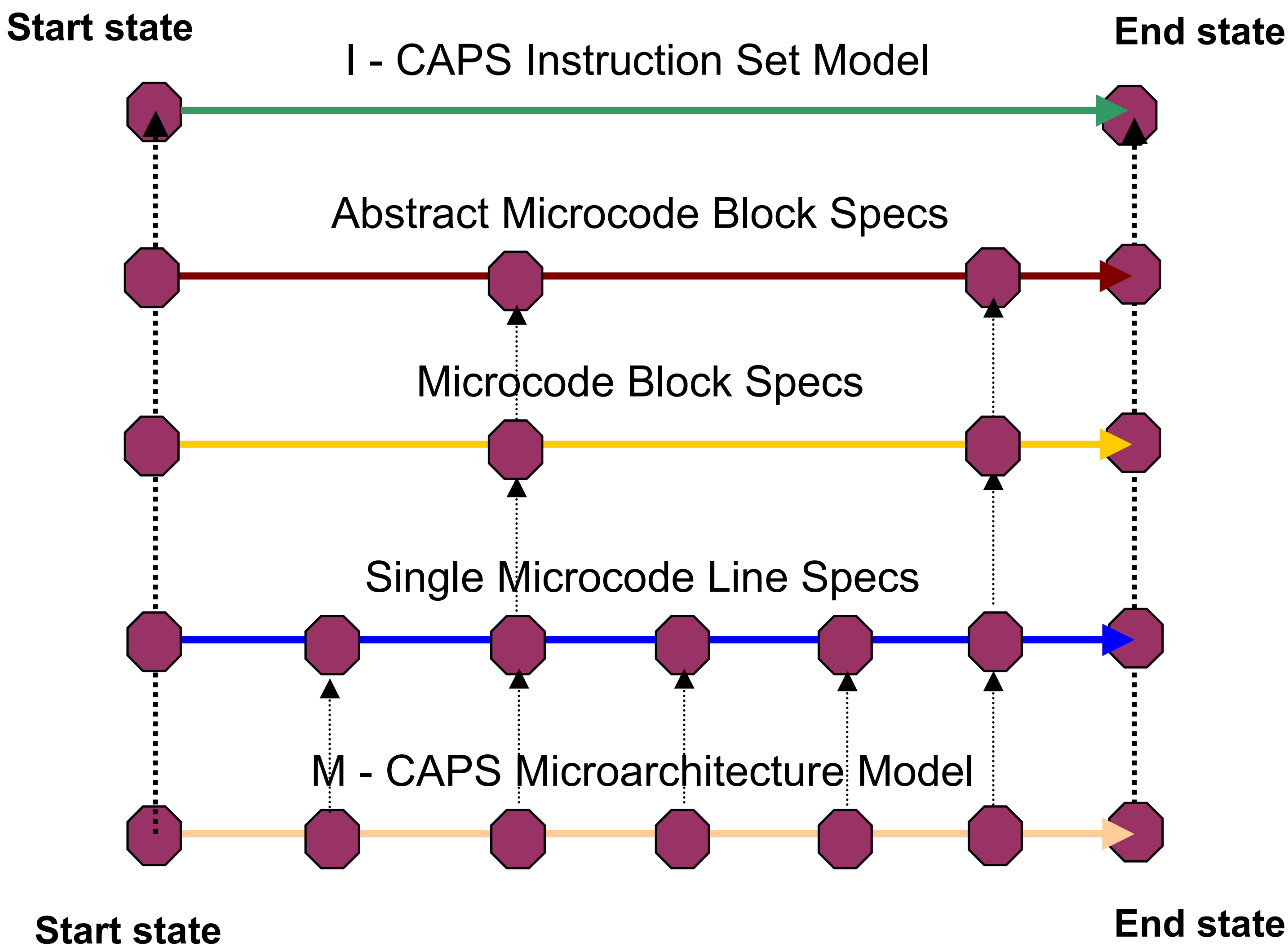
abstract microcode block spec

instruction



**Instruction microcode implementation**

# Proving the CAPS Correctness Theorem



# CAPS: High-Assurance Processor

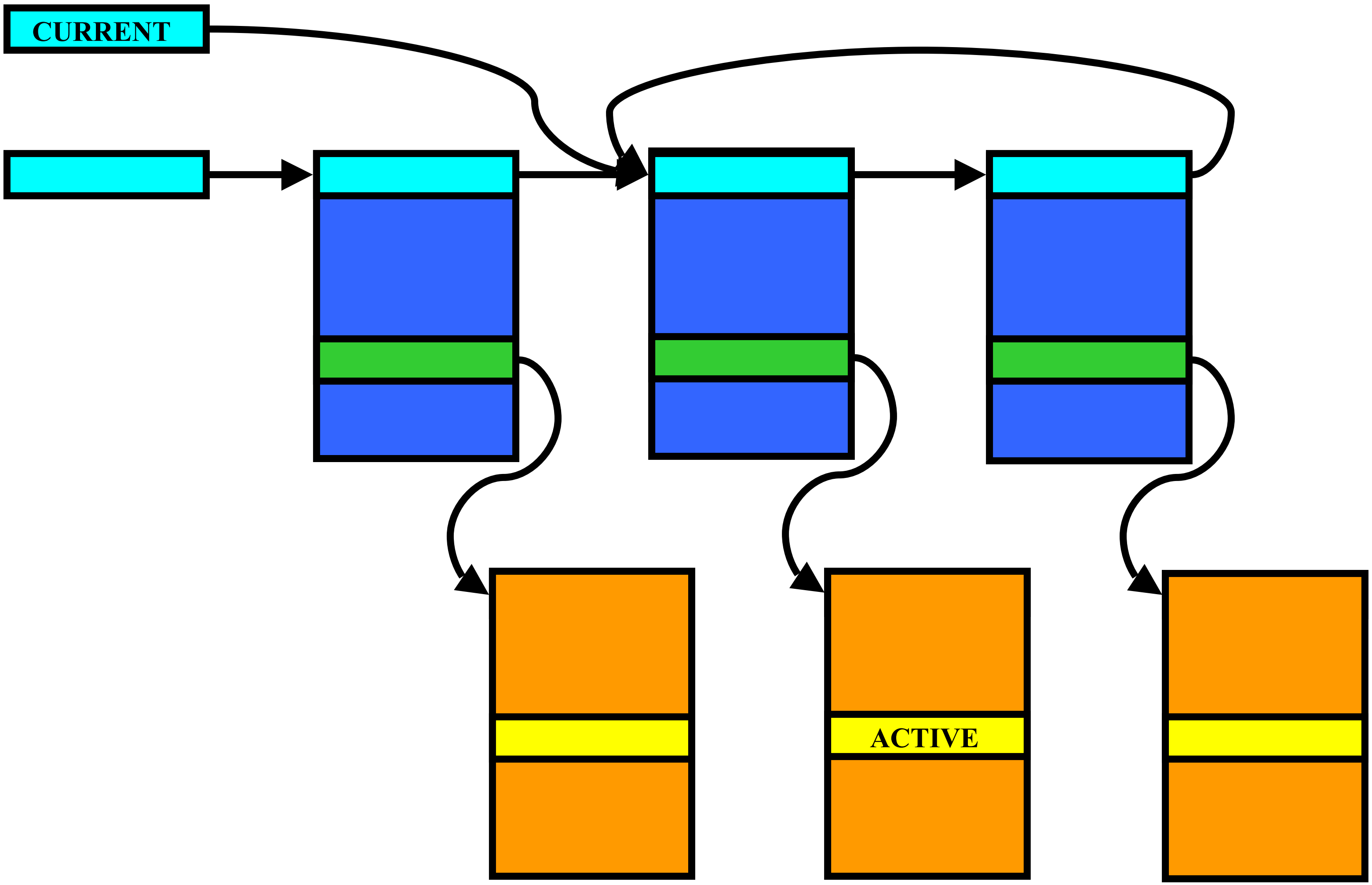
- **Considerable effort expended on EFM reasoning**
  - ACL2 enhanced to deal efficiently with single threaded expressions
- **Techniques to manage complexity**
  - Proof libraries
    - Bit vectors
  - Using low level model to help define abstract model
    - Simplifies abstract specification and proof process
  - Proof-generating Macros
    - Similar to techniques constructed for JEM1 symbolic simulation

# AAMP7: Intrinsic Partitioning

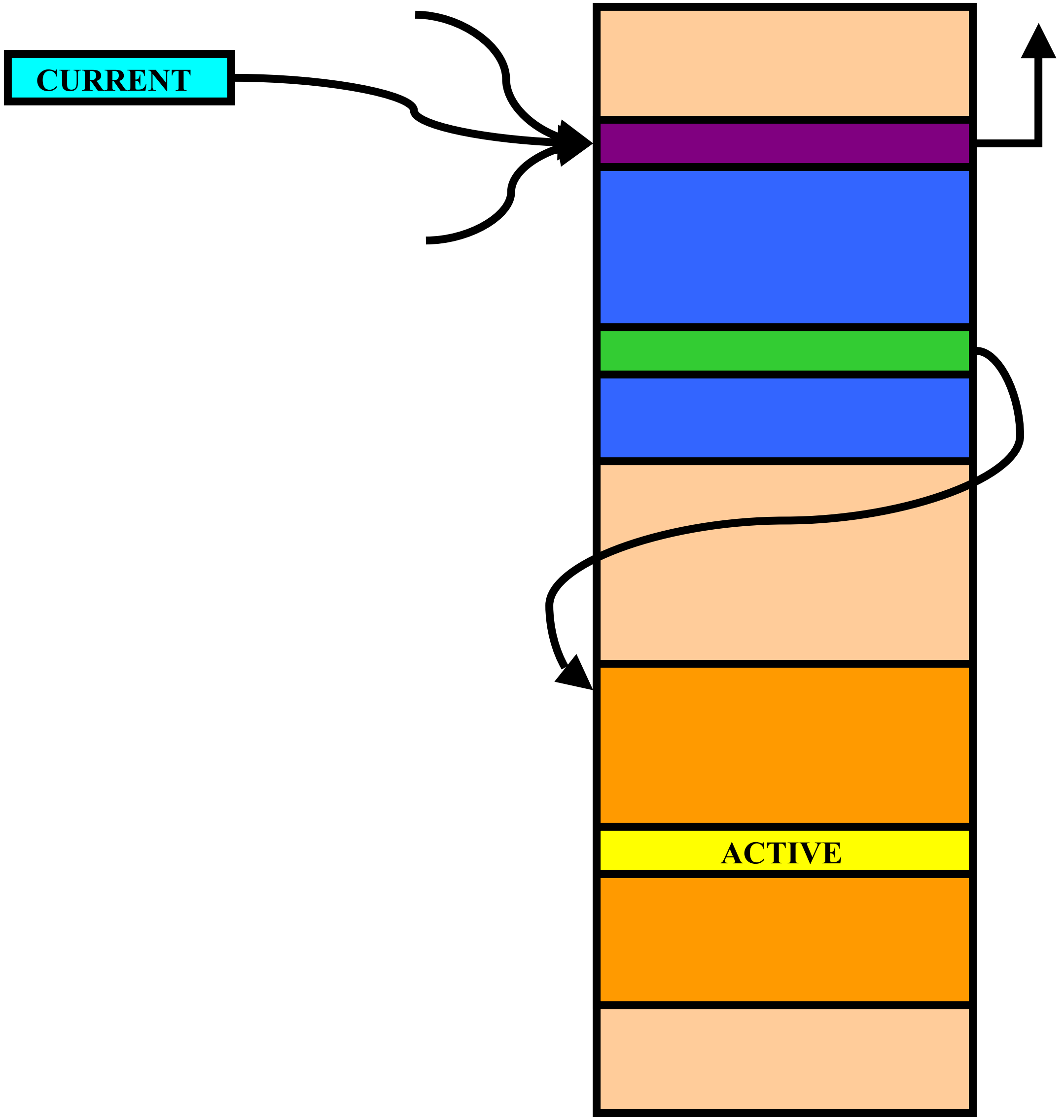
"High-Assurance Intrinsic Partitioning", HCSS-03

- **Goals**
  - Verify Security Properties of AAMP Intrinsic Partitioning Mechanism
- **Project**
  - Formalize Security Property in ACL2
  - Formalize Intrinsic Partitioning Functionality
    - "Instruction Level" Model
    - Linear Address Space
  - Prove that Intrinsic Partitioning satisfies Security Property

# Linear Address Space Reasoning



# Linear Address Space Reasoning



# AAMP7: Intrinsic Partitioning

- **Reasoning about Linear Address Spaces**
  - Identify orthogonal functionality
    - Techniques that scale
    - Make explicit for theorem prover
  - Could Leverage Data Flow (Definition/Use) Analysis
- **Proof Architecture**
  - Block structured decomposition
    - Similar to CAPS work
  - Over function boundaries
    - Not Microcode blocks

# Outline

- Motivating History
- Observations
- Future Directions

# Observations

- **Advantages to Accurate Low-Level Models**
  - Model Validation
  - Tie to Design Process via Simulation
- **Domain knowledge**
  - Control Flow, Data Flow Analysis
  - Can be codified in 3rd party tools
    - Results represented in language of theorem prover
- **Techniques generalize to**
  - Different theorem provers (PVS,ACL2)
  - Many different processor models
  - Different levels of abstraction

# Outline

- Motivating History
- Observations
- Future Directions

# Future Directions

- Extend techniques to larger, more complex problems
  - Additional microprocessors in the AAMP/JEM/CAPS family
  - Operating System Kernels
  - Mathematical and Cryptographic Libraries
  - Virtual Machines
- Enable the use of low-level models
  - Model validation
  - Avoid trusting or reasoning about compilers/abstractions
  - Assembly/micro code is not uncommon in these domains
    - Performance
    - Access to features unavailable in high-level languages
- Encourage continued industrialization of theorem proving technology
  - More powerful
  - More capable

# vFaat: Overview

**"A collection of tools and techniques to help simplify reasoning about complex systems"**

- **Input**
  - Domain specific: object files, microcode
  - Produces device and theorem prover independent representation
- **Annotation**
  - Pre/Post conditions
  - Proof Composition
- **Analysis**
  - 3rd party tools automate capture of domain specific knowledge
  - Stores results as annotations
- **Output**
  - Generates input for theorem prover
  - Isolates 3rd party tools from correctness argument

# Conclusion

- **Rockwell Collins History**
  - Have 10+ years experience in applying Formal Methods

- **Observations**
  - Low Level Models are useful in a variety of domains
  - Domain knowledge can be codified and used in theorem provers
  - Effective techniques generalize over domains and theorem provers

- **Conclusion**
  - Combining automated analysis and theorem proving
    - Improves productivity
    - Continues to provide high-assurance results
    - Extends the scope of what is currently feasible