PennState

CSE543- Computer and Network Security

Module: The Evolution of Secure Operating Systems

Asst. Prof. Syed Rafiul Hussain

Computer Science and Engineering Department
Pennsylvania State University

Need for Security

- The need for operating systems to enforce security requirements was recognized from the advent of multi-user operating systems

Multiprocessor Systems

- Major Effort: *Multics*
  - Multiprocessing system -- developed many OS concepts
    - Including security
  - Begun in 1965
    - Research continued into the mid-70s
  - Used until 2000
  - Initial partners: MIT, Bell Labs, GE (replaced by Honeywell)
  - *Other innovations*: hierarchical filesystems, dynamic linking

- Multics remains a basis for a secure operating systems design

Need for Security

- The need for operating systems to enforce security requirements was recognized from the advent of multi-user operating systems
  - F. J. Corbató and V. A. Vyssotsky. Introduction and overview of the Multics System. In *Proceedings of the 1965 AFIPS Fall Joint Computer Conference*, 1965.
  - "*Of considerable concern is the issue of privacy. Experience has shown that privacy and security are sensitive issues in a multi-user system where terminals are anonymously remote.*"

Questions

- So, were we done? No, still several difficult questions to address, including
- (1) What does security mean?
  - Policy: What degree of control and access should be allowed to enable a system to process user data securely?
- (2) How do we enforce security effectively?
  - Mechanism: What should be the requirements of a security mechanism to enforce security policies correctly?
- (3) How do we validate correctness in enforcement?
  - Validation: What methods are necessary to validate the correctness requirements for enforcing a security policy?

Multics Project (to 1977)

- Importantly, the Multics project explored all three big questions
  - And made important contributions to each

Multics Project (to 1977)

- Importantly, the Multics project explored all three big questions
  - And made important contributions to each
- What does security (policy) mean?
  - Security has to protect secrecy and integrity even when adversaries control processes (e.g., Mandatory Access Control)

Multics Project (to 1977)

- Importantly, the Multics project explored all three big questions
  - And made important contributions to each
- What does security (policy) mean?
  - Security has to protect secrecy and integrity even when adversaries control processes (e.g., Mandatory Access Control)
- What does enforcement mean?
  - Enforcement mechanisms must satisfy the reference monitor concept

Multics Project (to 1977)

- Importantly, the Multics project explored all three big questions
    - And made important contributions to each
- What does security (policy) mean?
    - Security has to protect secrecy and integrity even when adversaries control processes (e.g., Mandatory Access Control)
- What does enforcement mean?
    - Enforcement mechanisms must satisfy the reference monitor concept
- What does validation require?
    - Small code base; design for security; formal verification

Mandatory Access Control

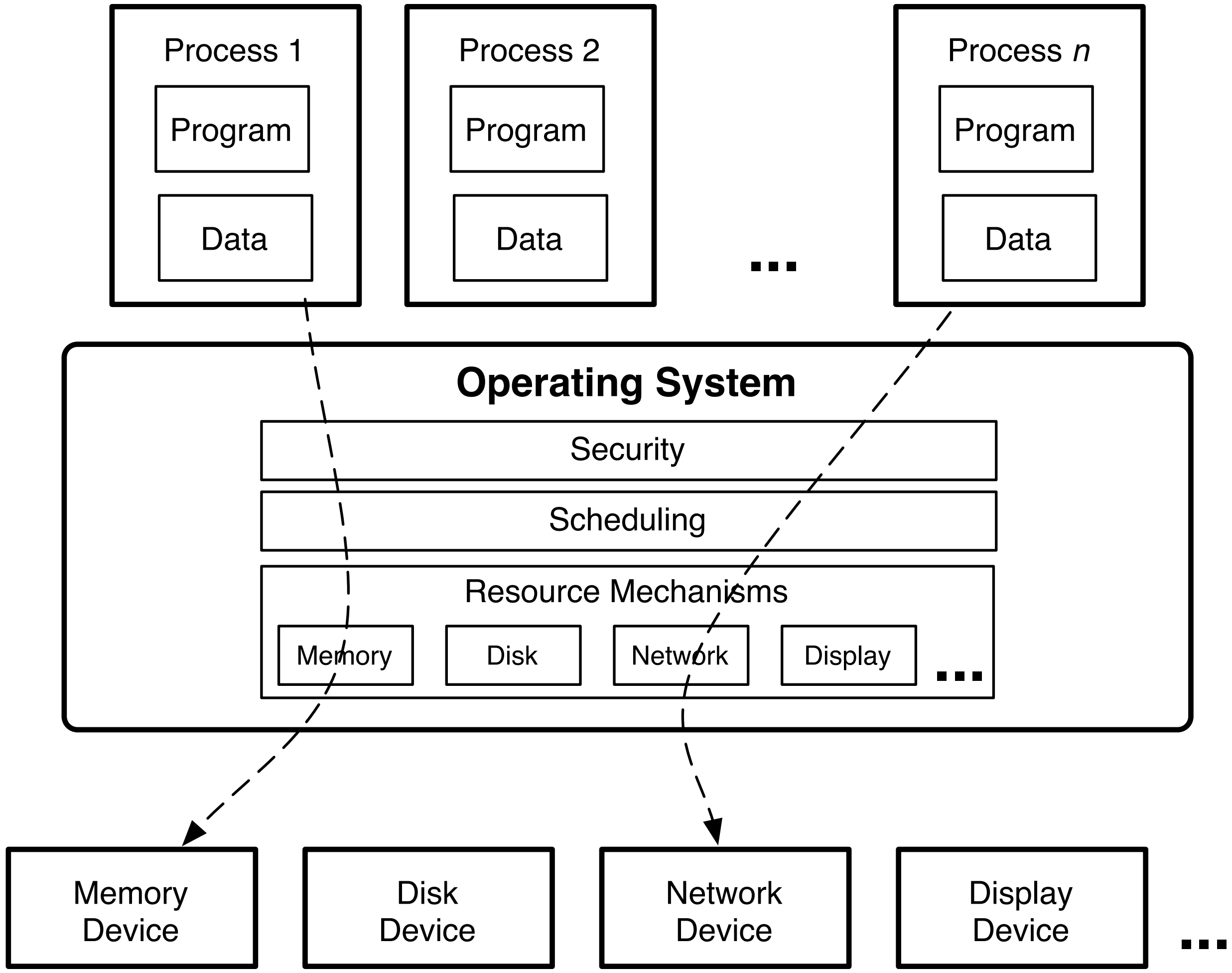
- Multics introduced mandatory access control (MAC) to enforce security
    - Mandatory – System-defined administration of policies
    - Access control – Information flow or MLS (e.g., Bell-La Padula, Biba)
- User programs are not authorized to
    - Read/Write to data to unauthorized files or processes
    - Or change the access control policy
- Prevents Trojan horse or compromised programs from violating expected data security

Multics Access Control

- Each resource is associated with an
  - Access Control List
  - Multilevel Security Level (secrecy)
    - Bell-La Padula
  - Access Brackets (integrity)
    - More later
- Last two are forms of mandatory access control

Enforcement in Multics

- How to apply policy to ensure correct enforcement?



Enforcement in Multics

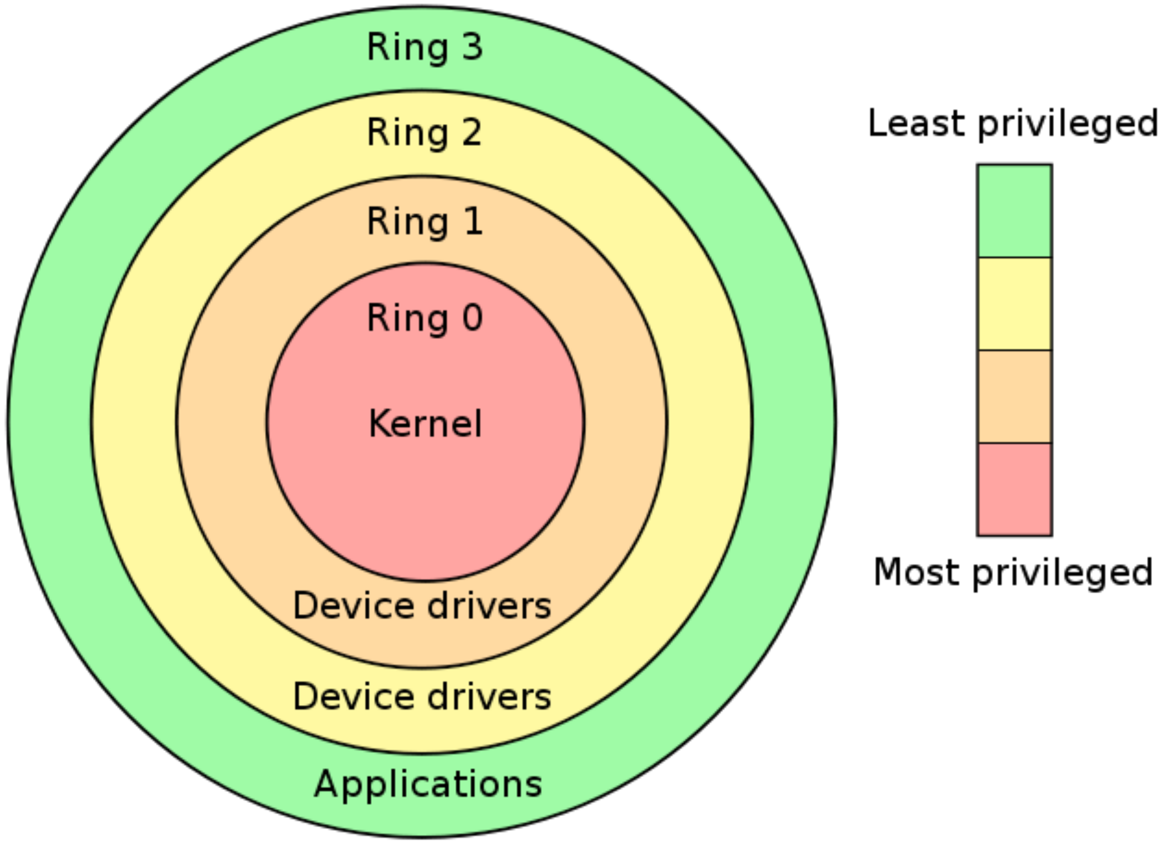
- Found that enforcement itself must be systematic and secured
    - Which OS operations should be protected?
    - How do authorization checks get processed correctly?
    - How do we know they were processed correctly?
- Clearly, an informal approach to the enforcement of policies is insufficient

Reference Monitor

- The Anderson report (USAF 1972) proposed the reference monitor concept to provide
  - *Explicit control must be established over each programs access to any system resource which is shared with any other user or system program.*
- Reference Monitor Concept requirements:
  - The reference validation mechanism must be tamperproof
  - The reference validation mechanism must always be invoked (complete mediation over security-sensitive operations)
  - The reference validation mechanism must be small enough to be subject to analysis and tests, the completeness of which can be assured (validation)

Protection Rings

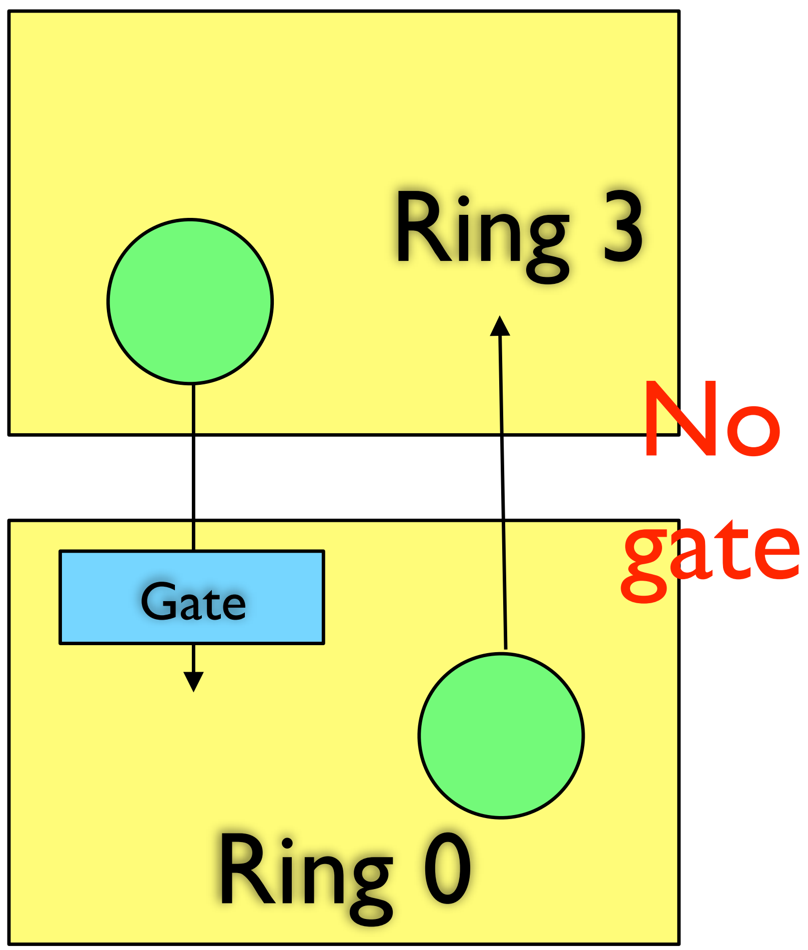
- Successively less-privileged "domains"
- Modern CPUs support 4 rings
  - Use 2 mainly: Kernel and user
- Intel x86 rings
  - Ring 0 has kernel
  - Ring 3 has application code



- Example: Multics (64 rings in theory, 8 in practice)

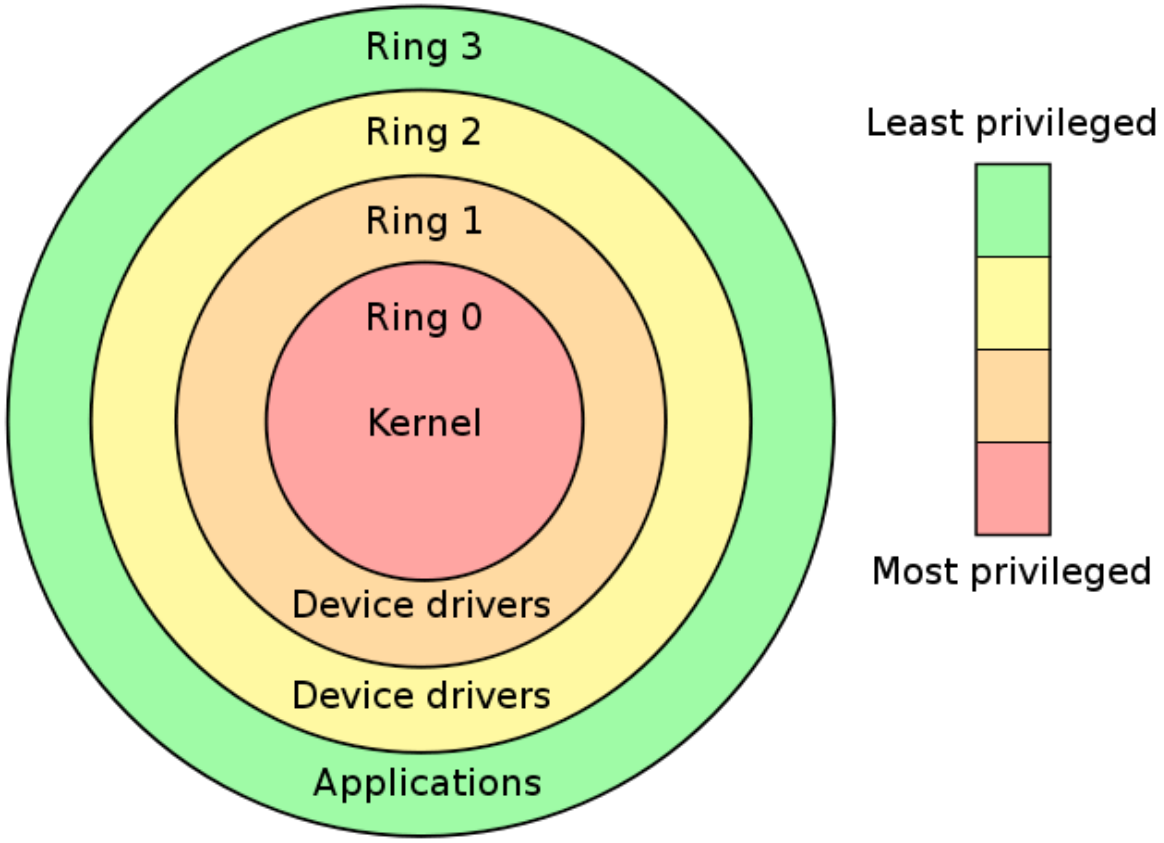
Protection Ring Rules

- Program cannot call code of *higher privilege* directly
  - Gate is a special memory address where lower-privilege code can call higher
    - Enables OS to control where applications call it (system calls)



What Are Protection Rings?

- Coarse-grained, Hardware Protection Mechanism
- Boundary between Levels of Authority
  - Most privileged -- ring 0
  - Monotonically less privileged above
- Fundamental Purpose
  - Protect system integrity
    - Protect kernel from services
    - Protect services from apps
    - So on...



Access Brackets

- Multics policy that governs access control based on the ring in which code is run
  - Subject – process's ring number
  - Object – resource's ring number
  - Operations – usual read, write and execute
- By default, processes cannot
  - Modify resources in lower (more privileged) rings
    - What access control model is that?
  - A bit too strong
    - Weakened to a contiguous sequence of rings that could modify (or execute) each object

Reference Monitor in Multics

- Tamperproofing
    - Protection rings
    - Kernel in ring 0
    - Gates protecting kernel entry and exit
- Complete mediation
    - Resources modeled as "segments"
    - Control all segment operations (ACLs, MLS, ring brackets)
- Validation
    - Come back to this

Karger-Schell Analysis

- Demonstrated the importance of following the reference monitor concept
  - Flaws in Tamperproofing
    - Untrusted "master mode" code run in Ring 0 for performance
    - No untrusted code in ring 0
  - Flaws in Complete Mediation
    - Failure to mediate some indirect memory accesses
    - Implementation bug in complete mediation
- However, these were both flaws in implementation, not design, that would have been alleviated by following the reference monitor concept correctly

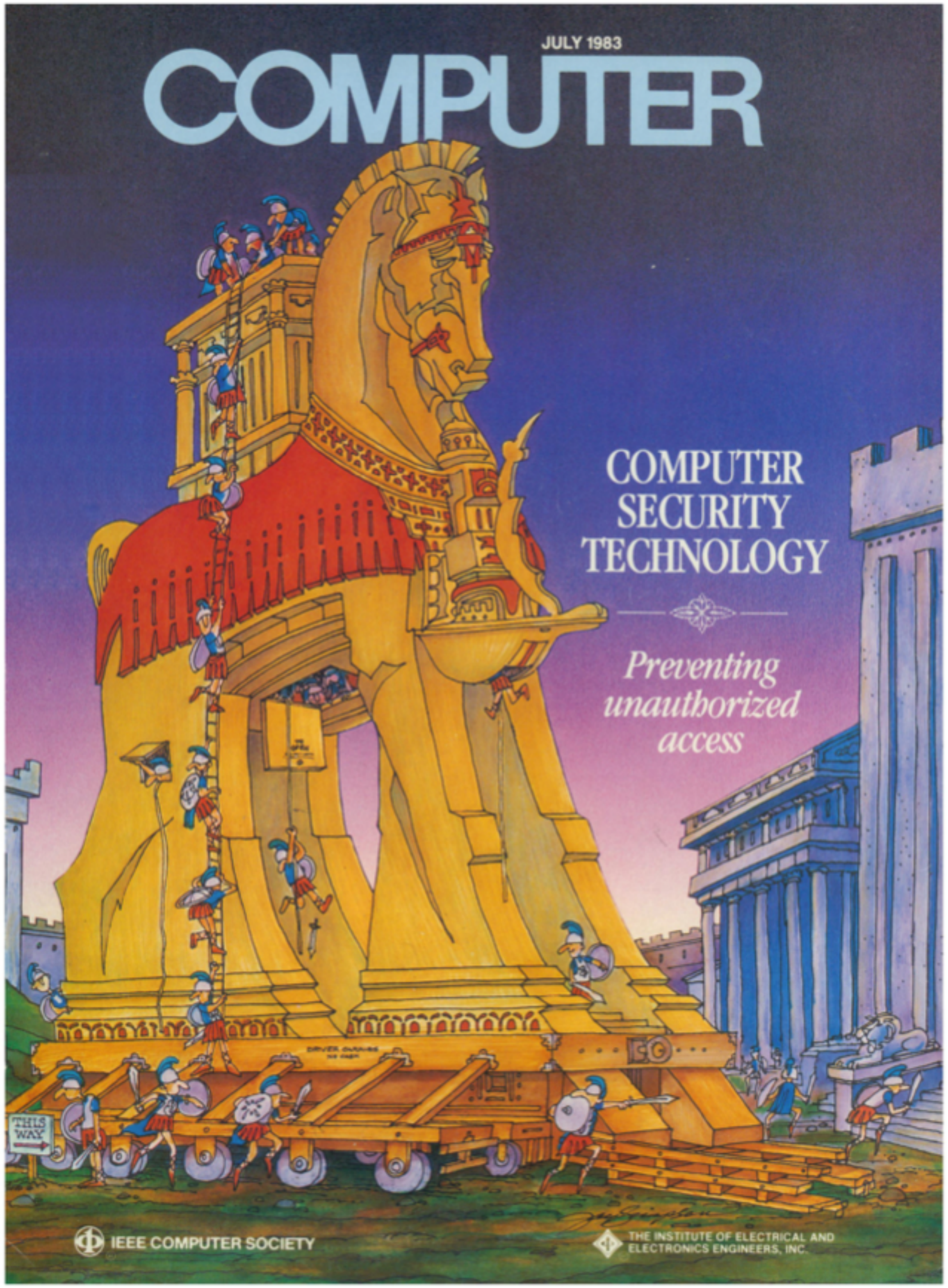
Validation in Multics

- Challenges were seen for validating Multics (circa 1977)
  - Size of the code base – 54 SLOC
    - Although the Multics Final Report suggests that the kernel size can be reduced by approximately half
  - How to do formal validation on a kernel?
    - To this point techniques had not been developed
- Ultimately, the Multics design formed the basis for the B2 assurance level of the Orange Book (now Common Criteria)
  - + Security policy model clearly defined and formally documented (B2)
  - - Satisfies reference monitor requirements (B3)

Security Kernel Experience

- A number of projects emerged to address the challenge of validating secure operating systems
  - Which came to be called security kernels
- To address three main challenges
  - Reduce size and complexity of operating systems and utility software
  - Define security enforced by the OS internal controls
  - Validate the correctness of the implemented security controls
- From Ames and Gasser, *IEEE Computer*, July 1983

July 1983, IEEE Computer

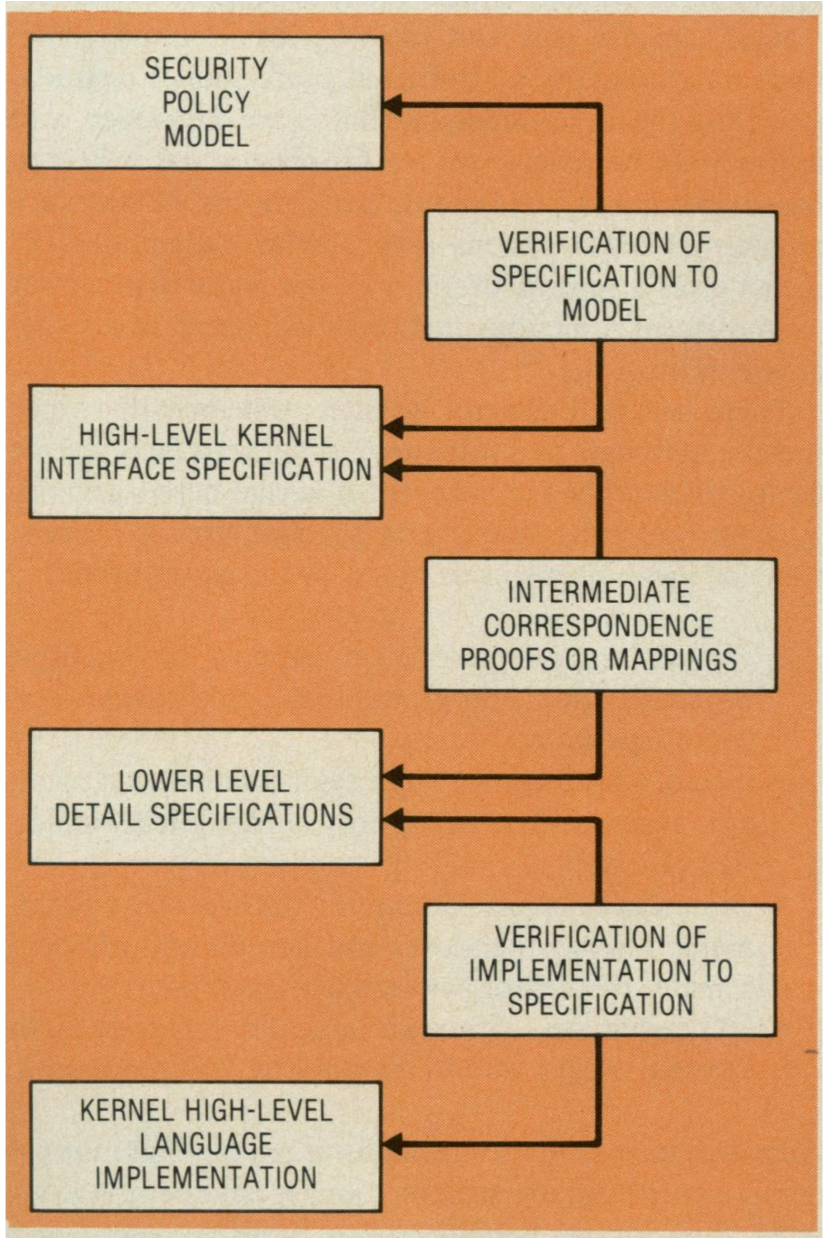


Security Kernel Approach

- Security Kernel Design: Ames, Gasser, and Schell
- Basic Principles
  - A formally defined security model
    - Complete, mandatory, and validated for security requirements
  - Faithful implementation
    - Transfer model to design incrementally and formally
- While addressing practical considerations
  - Extracting security relevant functionality from OS at large
  - Formal specification and validation methods

Security Kernel Approach

- From model to implementation



Formal Verification

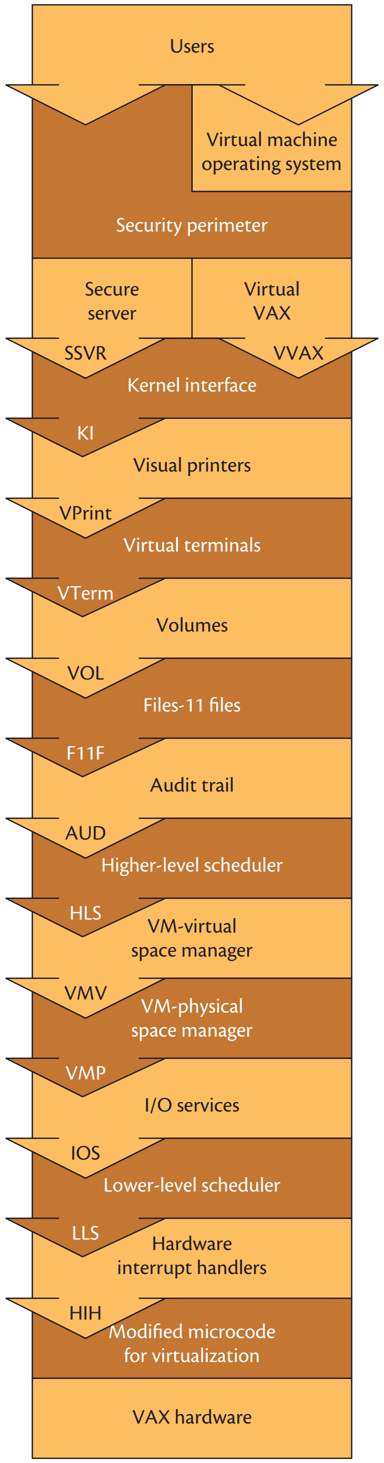
- What techniques are necessary to formally assure a kernel implementation satisfies a security model?
  - “verification has turned out to be more difficult than we expected”
- Goal: correctness
  - Techniques not ready to prove correctness
- Approaches (at this time)
  - Compare kernel security to information flows allowed
  - Specification and implementation correspondence

VMM Security Kernel

- Choices in bringing security kernel OS to market
  - (1) High-assurance version of existing OS
    - But, would trail the standard product development lifecycle
  - (2) Custom, high-assurance OS
    - Lack application and ecosystem support
- Alternative: high-assurance virtual machine monitor (VMM)
  - Motivation for the "VMM Security Kernel for VAX" in 1980 IEEE Symposium on Security and Privacy
    - VMM security kernel layers under commercial OSes
    - To support multiple OSes and versions

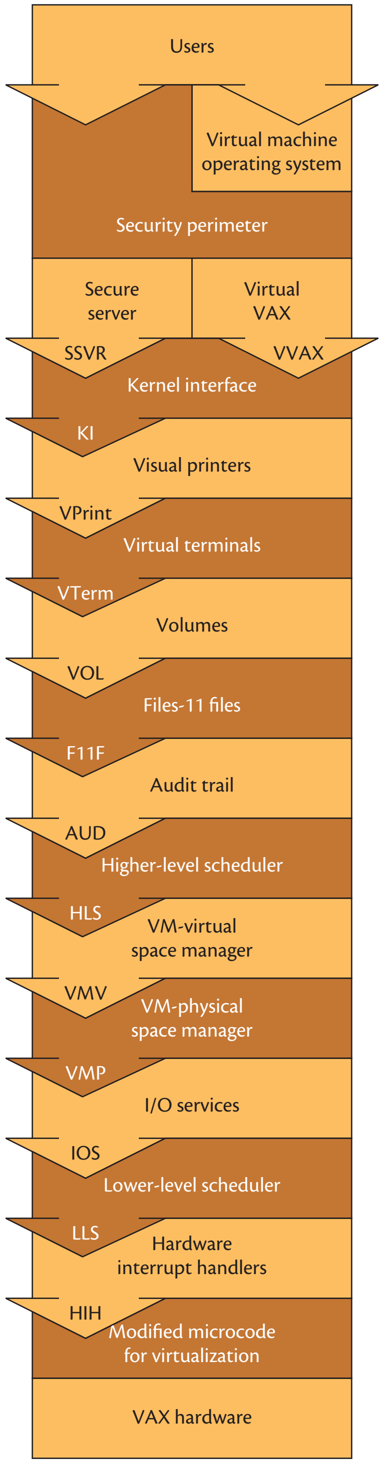
VAX/SVS Project

- Important design choices
  - Layered system design
    - Aimed to simplify design, test, and assurance
  - Enforce information flow for secrecy and integrity
    - Bell-La Padula and Biba
    - Coarse-grained: For VMs access to storage volumes
  - Paravirtualization with simple memory management
- Implemented in Pascal, PL/1, and assembly
  - About 48K SLOC altogether



VAX/SVS Project

- Project Successes
  - System was piloted in 1989 – “reasonably successful”
  - “A VMM Security Kernel for the VAX Architecture” was lead paper and Best Paper Award winner at the 1990 IEEE Symposium on Security and Privacy
  - Comprehensive effort for A1 assurance applying forma methods for system design, test, maintenance, and cover channels
- Nonetheless, the project was cancelled in 1990
  - Lack of customers – export controls did not help
  - Lack of features – e.g., no networking support



Users

Virtual machine operating system

Security perimeter

Secure server

Virtual VAX

SSVR

VVAX

Kernel interface

KI

Visual printers

VPrint

Virtual terminals

VTerm

Volumes

VOL

Files-11 files

F11F

Audit trail

AUD

Higher-level scheduler

HLS

VM-virtual space manager

VMV

VM-physical space manager

VMP

I/O services

IOS

Lower-level scheduler

LLS

Hardware interrupt handlers

HIH

Modified microcode for virtualization

VAX hardware

VAX/SVS Project

- Other issues that may have had an impact
  - Drivers are in the VMM security kernel
    - DMA enables malicious device to overwrite physical memory
    - Implications?
  - Multi-user and privileged VMs
    - Achieving A1 assurance in practice requires tracking individual users, but no visibility into VMs
    - Implications?
  - Assembly code
    - About 11K SLOC of the VMM security kernel was implemented in assembly

L4.Verified

1 microkernel

8,700 lines of C

0 bugs*

qed

*conditions apply

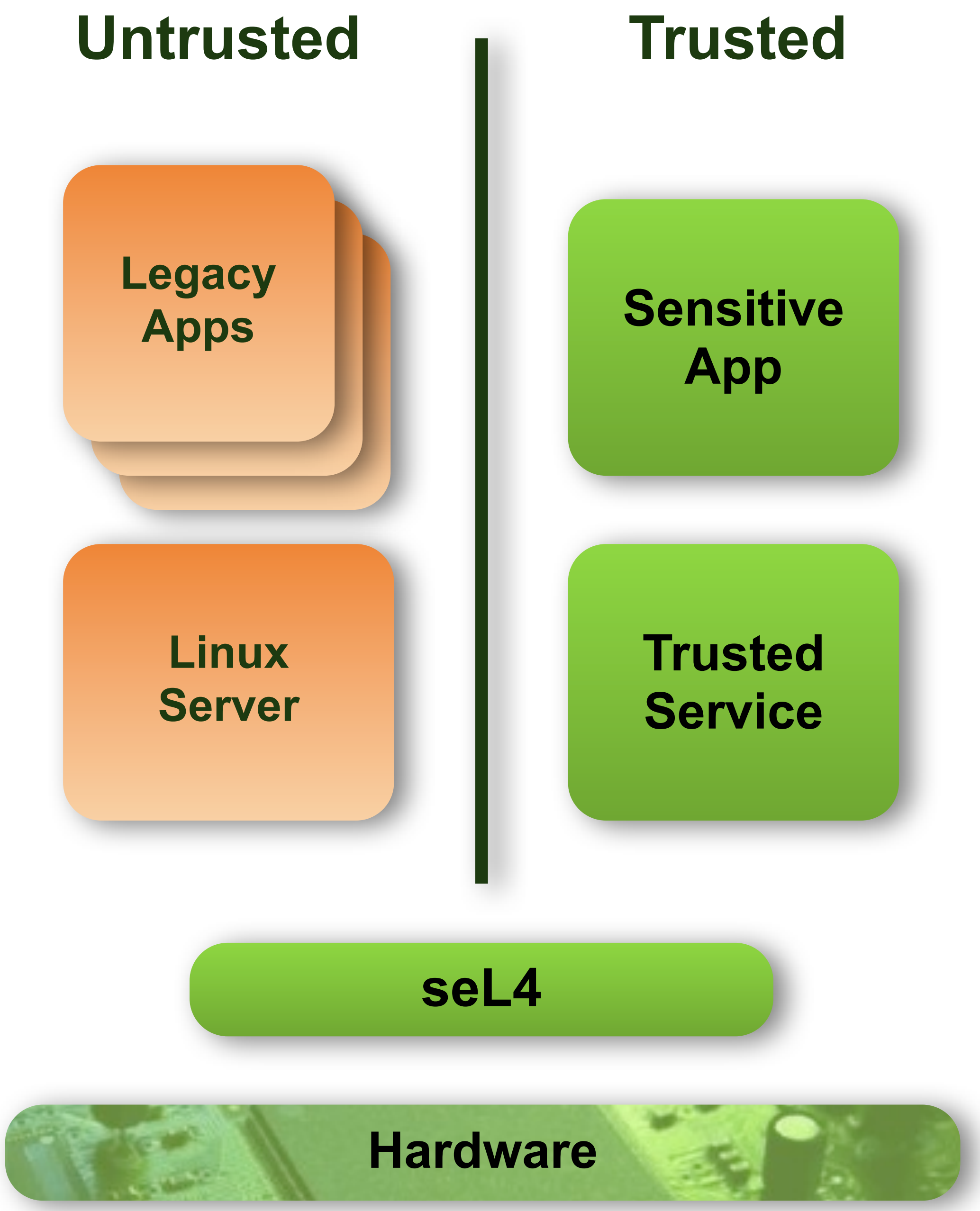
Small Kernels

Small trustworthy foundation

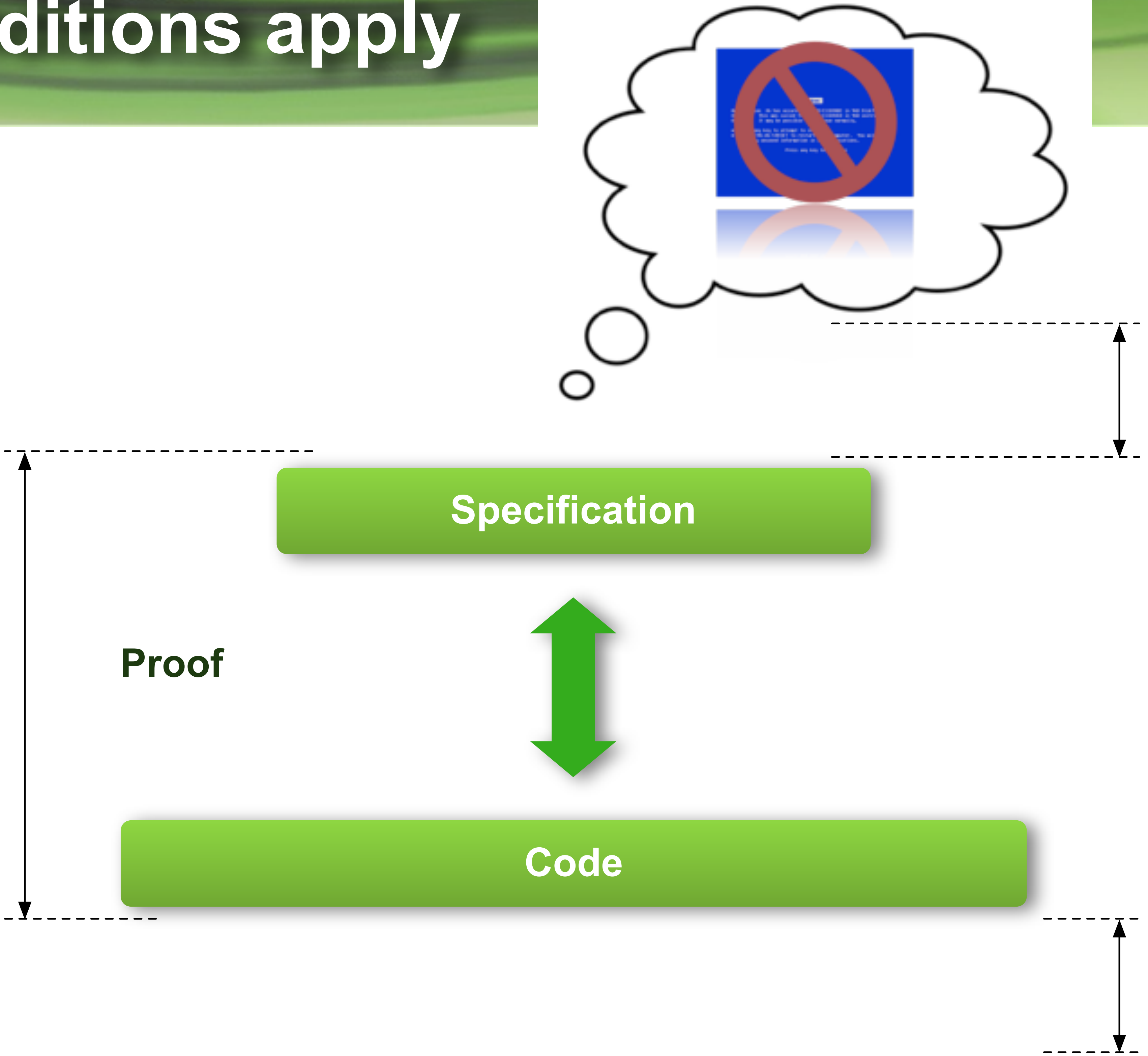
- hypervisor, microkernel, nano-kernel, virtual machine, separation kernel, exokernel ...
- High assurance components in presence of other components

seL4 API:

- IPC
- Threads
- VM
- IRQ
- Capabilities



*conditions apply



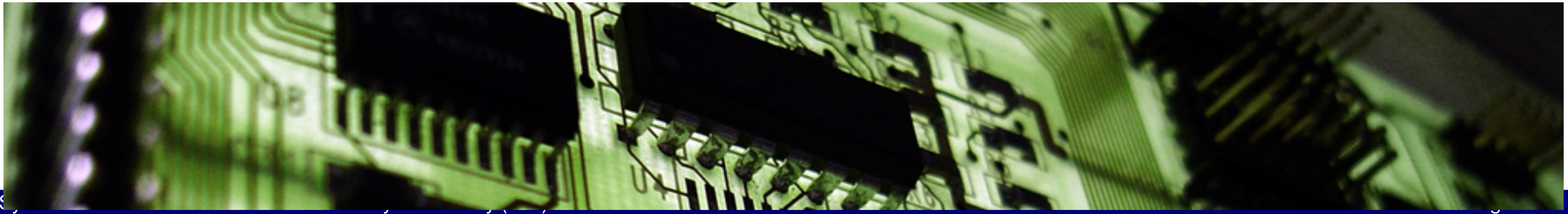
Expectation

Specification

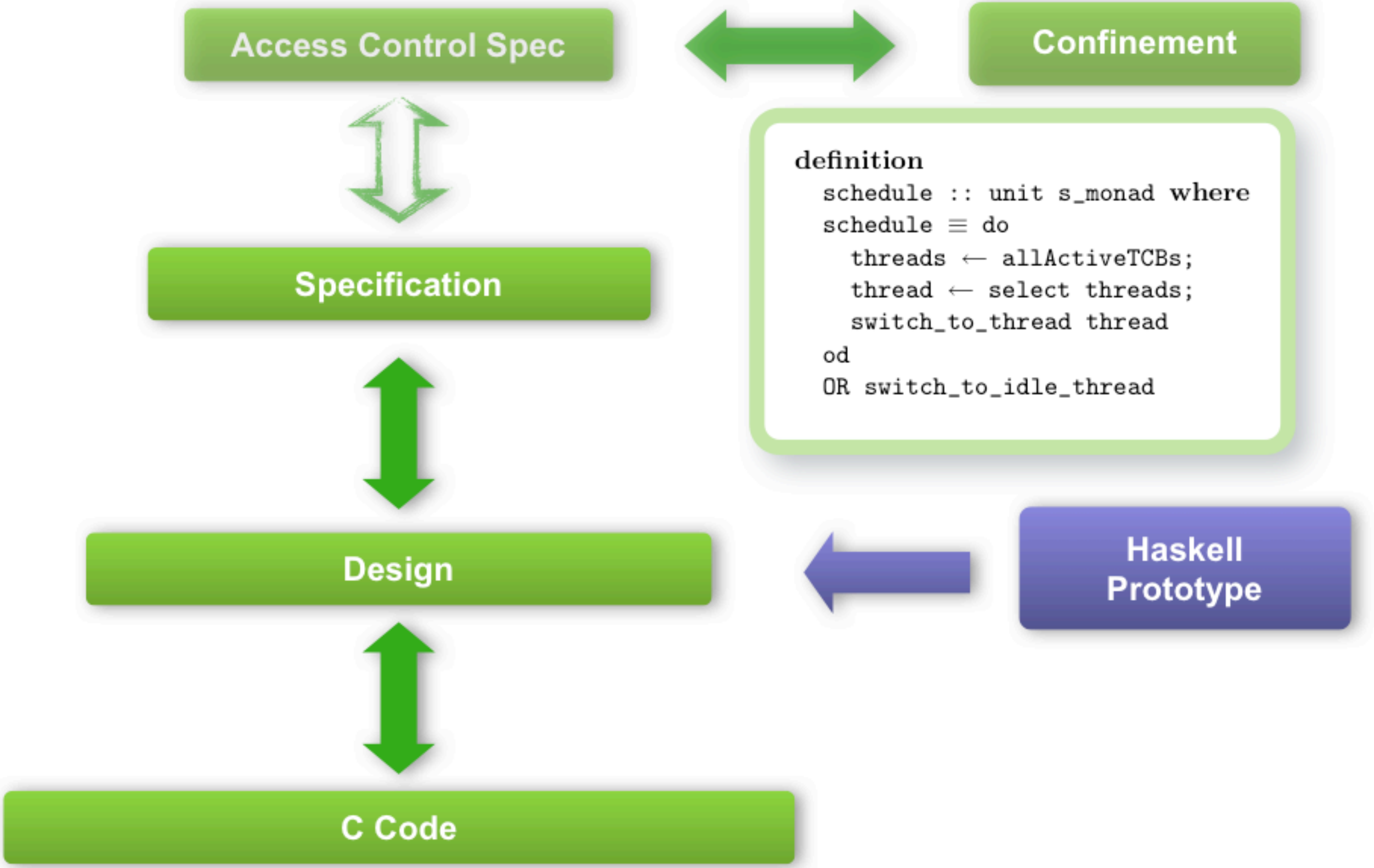
Proof

Code

Assumptions



Proof Architecture



Access Control Spec

Confinement

Specification

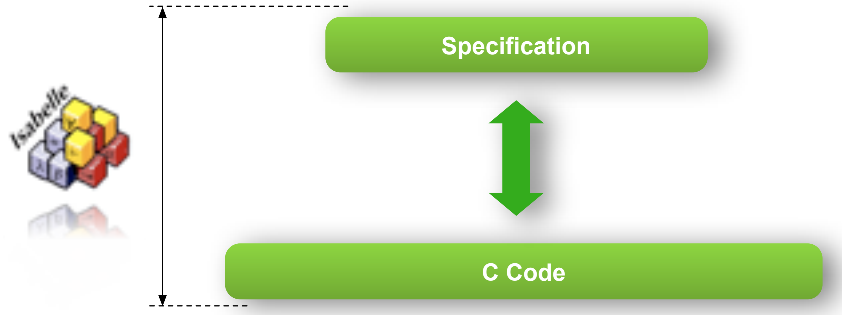
Design

C Code

Haskell Prototype

```
definition
  schedule :: unit s_monad where
  schedule ≡ do
    threads ← allActiveTCBs;
    thread ← select threads;
    switch_to_thread thread
  od
  OR switch_to_idle_thread
```

Implications



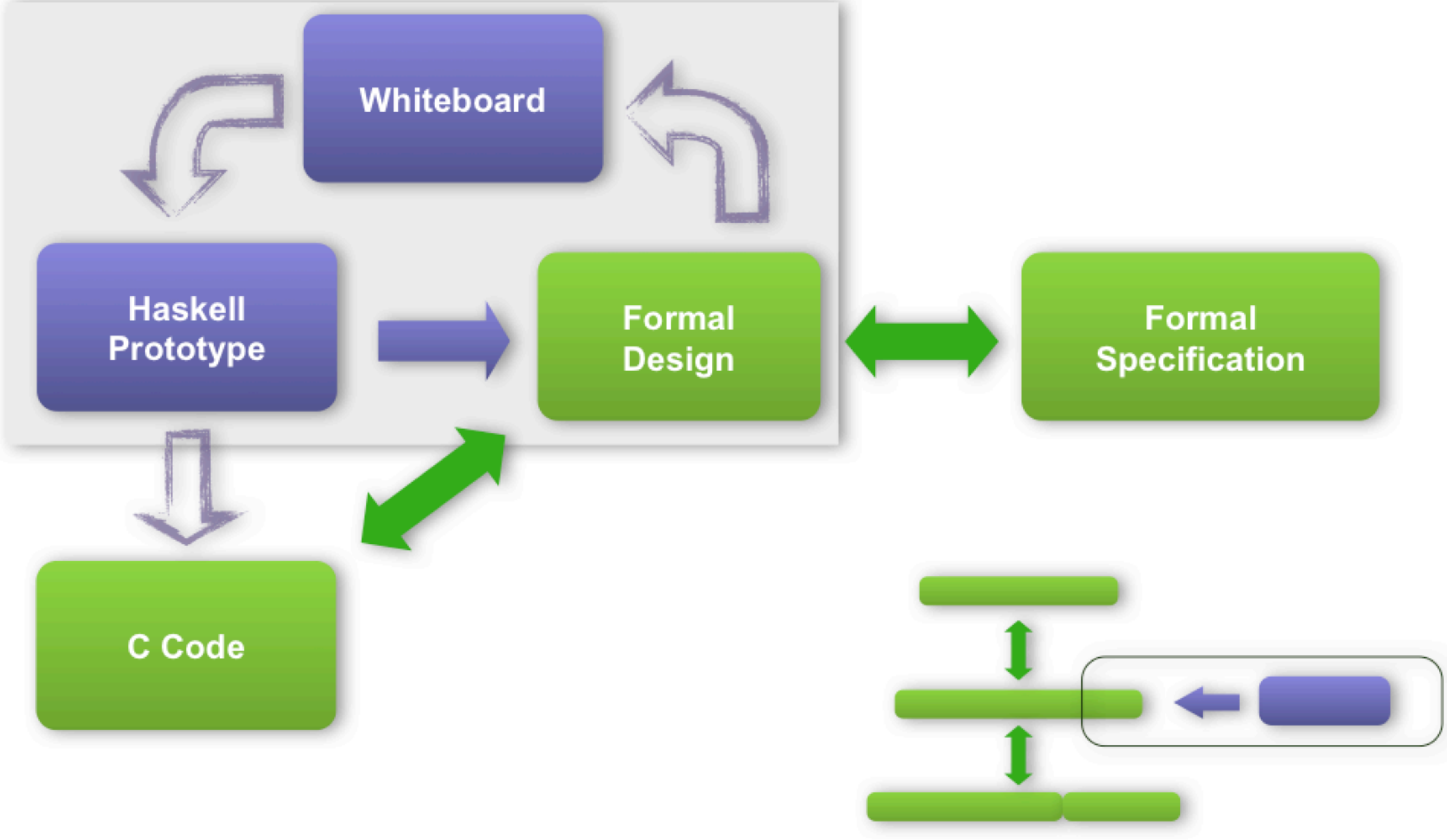
Execution always defined:

- no null pointer de-reference
- no buffer overflows
- no code injection
- no memory leaks/out of kernel memory
- no div by zero, no undefined shift
- no undefined execution
- no infinite loops/recursion

Not implied:

- “secure” (define secure)
- zero bugs from expectation to physical world
- covert channel analysis

Iterative Design and Formalisation



Did you find any Bugs?

NICTA

Bugs found

during testing: 16

during verification:

- in C: 160
- in design: ~150
- in spec: ~150

460 bugs

```
void
schedule(void) {
    switch ((word_t)ksSchedulerAction) {
        case (word_t)SchedulerAction_ResumeCurrentThread:
                                                                read;
                                                                read;
    }
}

void
chooseTh
    prio
    tcb_

    for(
                tcbSchedDequeue(thread);
            }
            else {
                switchToThread(thread);
                return;
            }
        }
    }

    switchToIdleThread();
}
```

Effort

Haskell design	2	py
First C impl.	2	weeks
Debugging/Testing	2	months
Kernel verification	12	py
Formal frameworks	10	py
Total	25	py

Cost

Common Criteria EAL6:	**$87M**
L4.verified:	**$6M**

Take Away

- The importance of enforcing security in operating systems has been long recognized
- Multics examined the dimensions of what to enforce (policy) how to enforce (mechanism), and need for validation
- Security kernel projects explore how to validate real systems based on security designs converted to implementations
- Recent and future work shows promise of overcoming some of the major challenges that have held back prior work
- With the availability of a formally verified core kernel, there is an opportunity to develop secure operating environment