Modernizing Software Verification

Craig Christianson

SkiCAMP

309 SWEG USAF

Overview

- Challenges of Software Engineering
- Software Assurance
- Automated Reasoning
- seL4 Microkernel
- Microkit

Challenges of Software Engineering

Craftsmen

- Creation backed by:
  - Skill
  - Experience
  - Best Practices

Photo by zoe koskinioti: https:///www.pexels.com/photo/brown-wooden-house-under-blue-sky-2183521/

Engineers

- Creation backed by:
  - Mathematically defined specifications
  - Mathematically & Scientifically verified solutions
  - Compliance
  - Liability Transfer

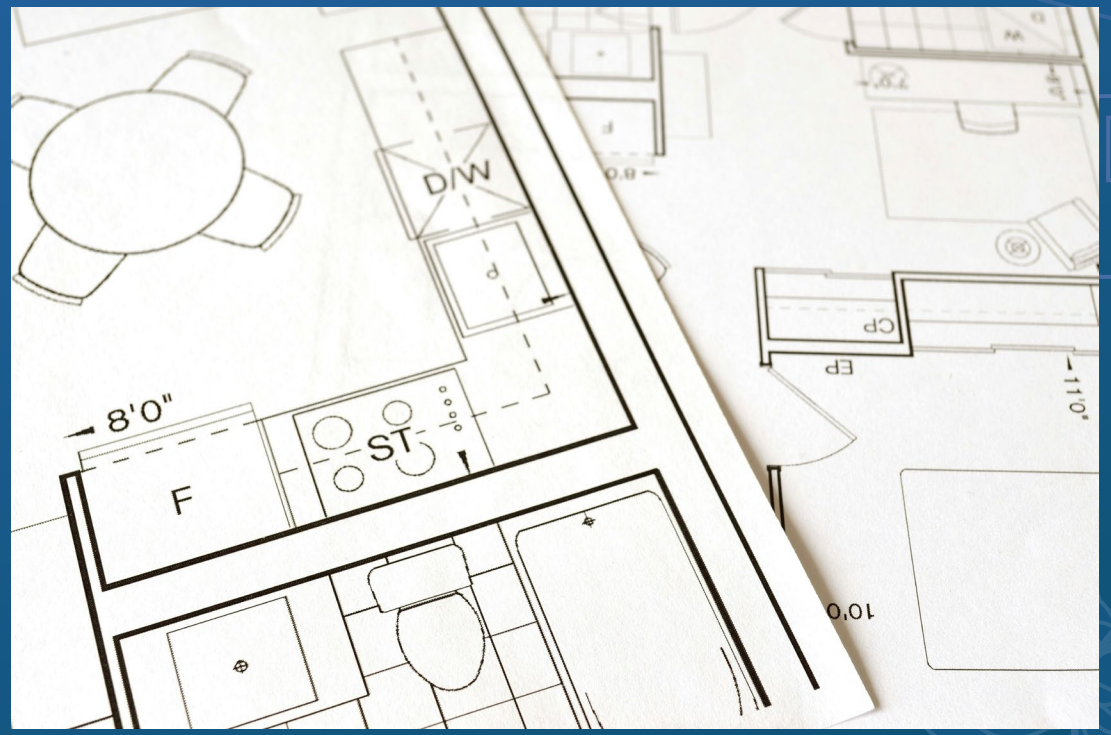


Photo by pixabay: https:///www.pexels.com/photo/house-floor-plan-271667/

Sodium Hydroxide (Lye)

DANGER!
Causes eye and skin burns. May cause blindness or deep, penetrating skin ulcers. May cause severe and permanent damage to digestive tract…

Safety Critical Software

Photo by Daryl Knee: https://media.defense.gov/2014/Aug/19/2000847521/-1/-1/0/140818-F-JZ627-061.JPG

Requirements from DO178C

- High-Level & Low-Level Requirements Verification
- Bidirectional Requirements Traceability
- 100% Code Coverage Tests
- Reviews and Analyses
- Extensive Testing at Every Integration Level

Photo by R. Nial Bradshaw:
https://media.defense.gov/2017/Mar/30/2001724711/-1/-1/0/160531-F-OD616-124.JPG

SULLY2 HUD
BFM-9
5 May 16

https://en.wikipedia.org/wiki/Automatic_Ground_Collision_Avoidance_System

Photo by Senior Airman Addie Peterson:
https://media.defense.gov/2025/Jul/02/2003747560/-1/-1/0/250626-Z-KH354-9223.JPG

Software Assurance

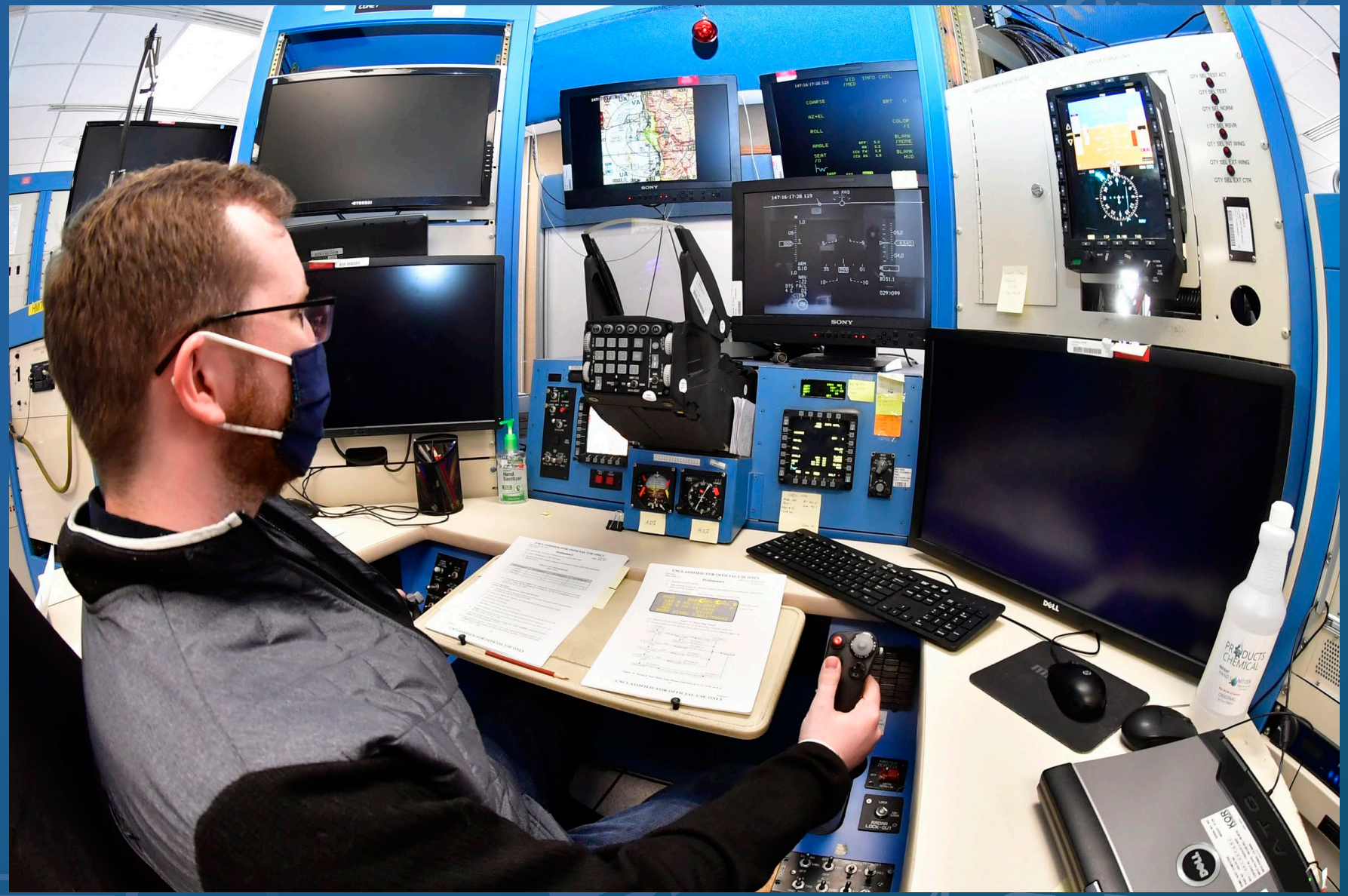


Photo by Todd Cromar:
https://media.defense.gov/2020/Jun/04/2002311161/-1/-1/0/200526-F-LS255-0003.JPG

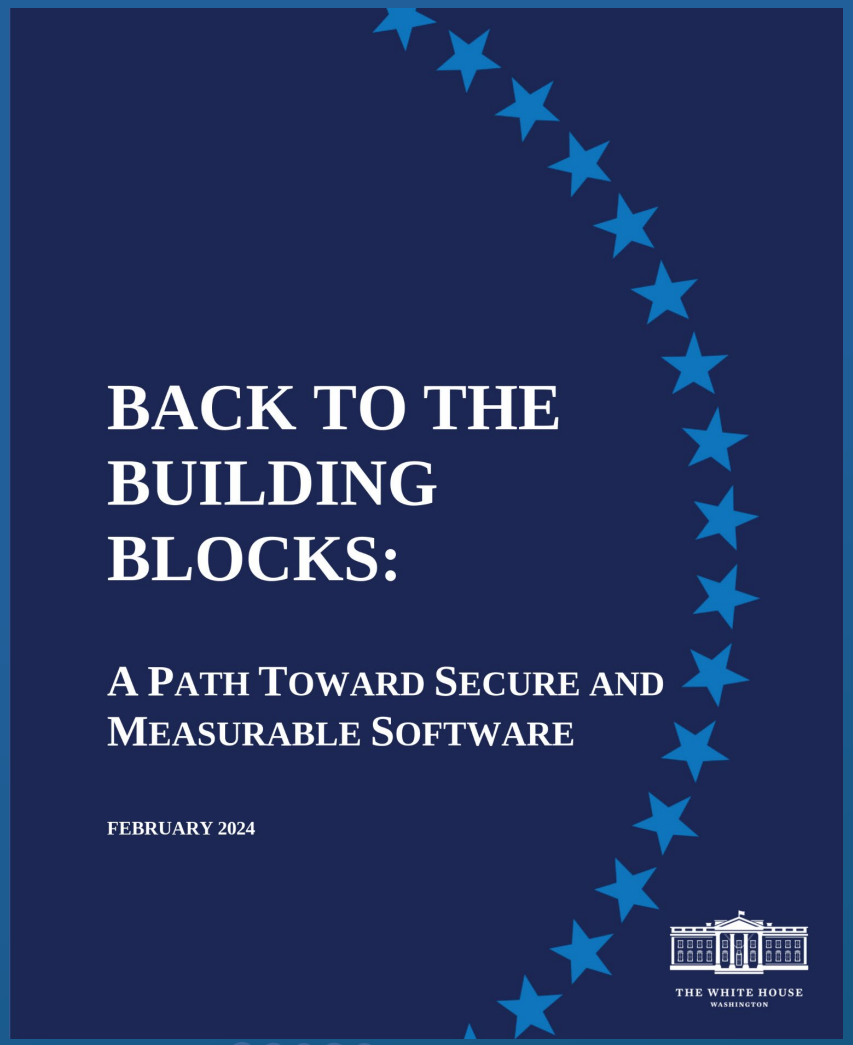
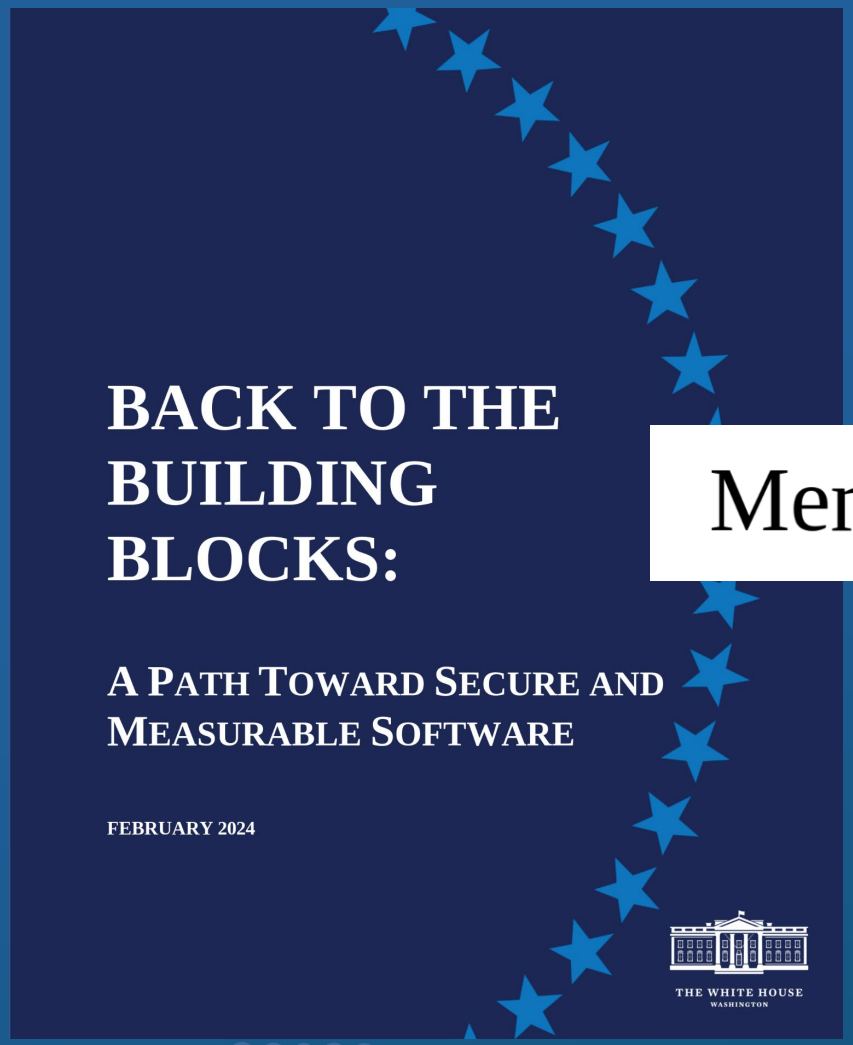


Table of Contents



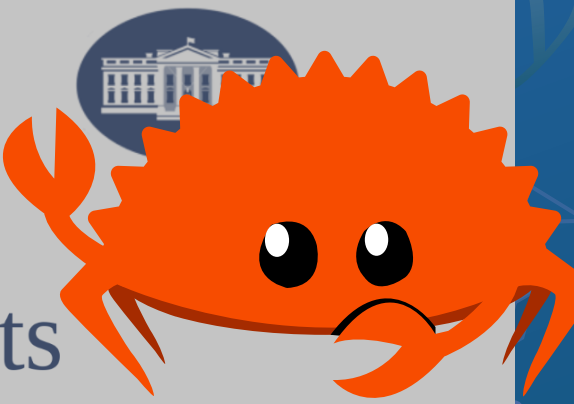
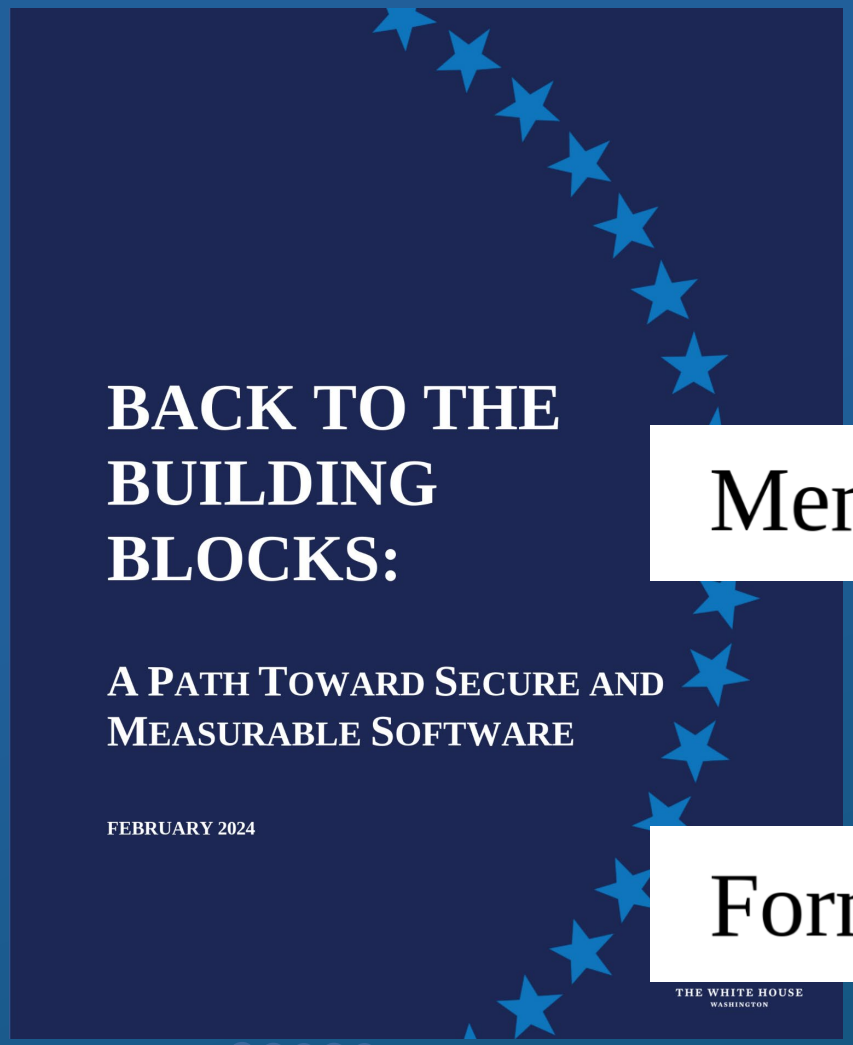


Table of Contents



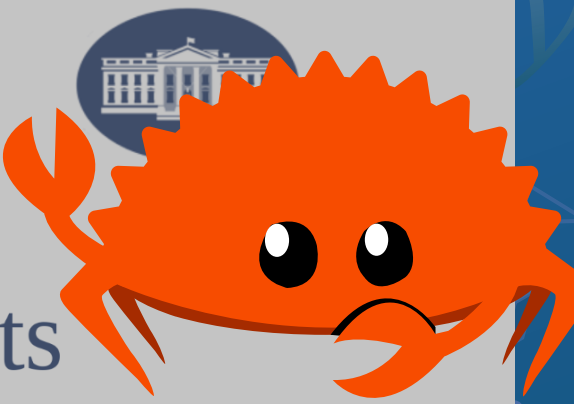


Table of Contents

Two Roads to Certification

Legacy Assurance

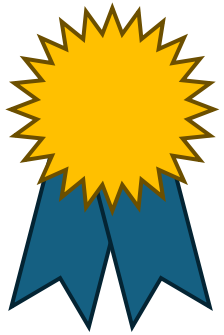
- Systematic evaluation & testing
- Essentially an intensive & onerous form of Quality Assurance

Formal Verification

- Uses mathematical models to define systems
- Uses proofs to establish traceability between models and the final system

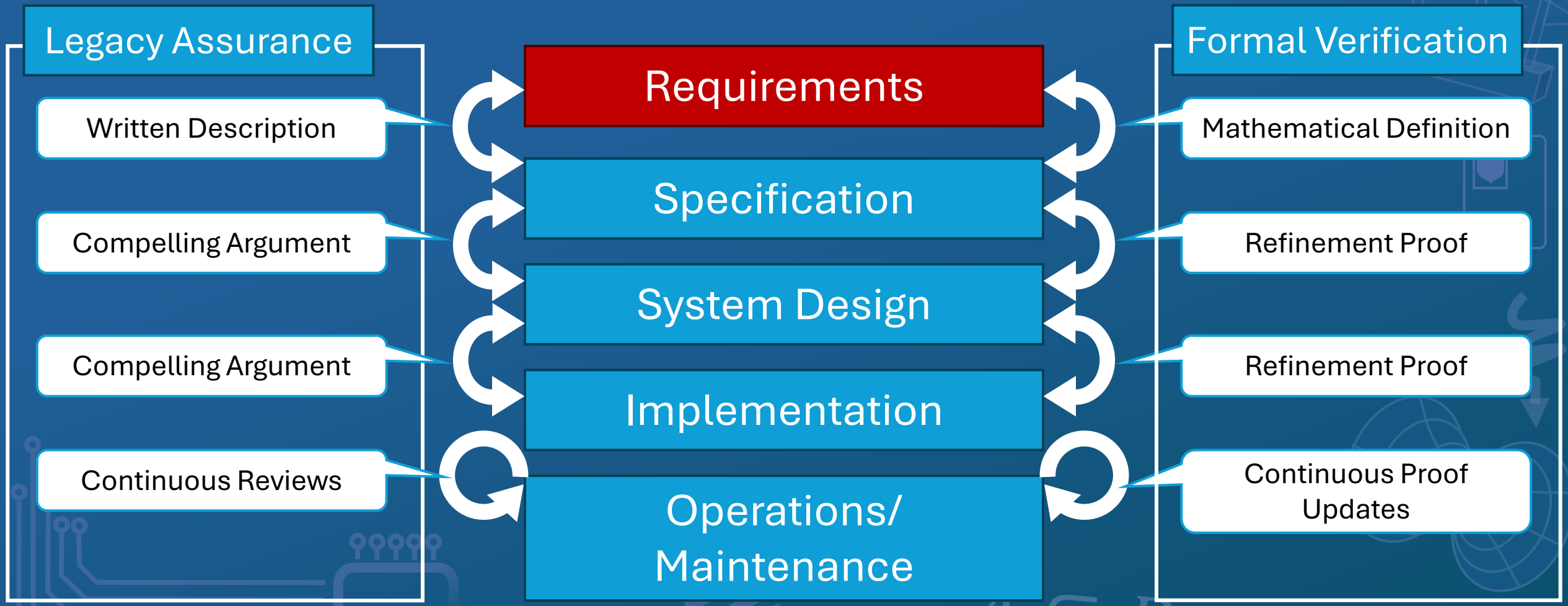
Certification

- Independent examination
- Confirms assurance process was done correctly

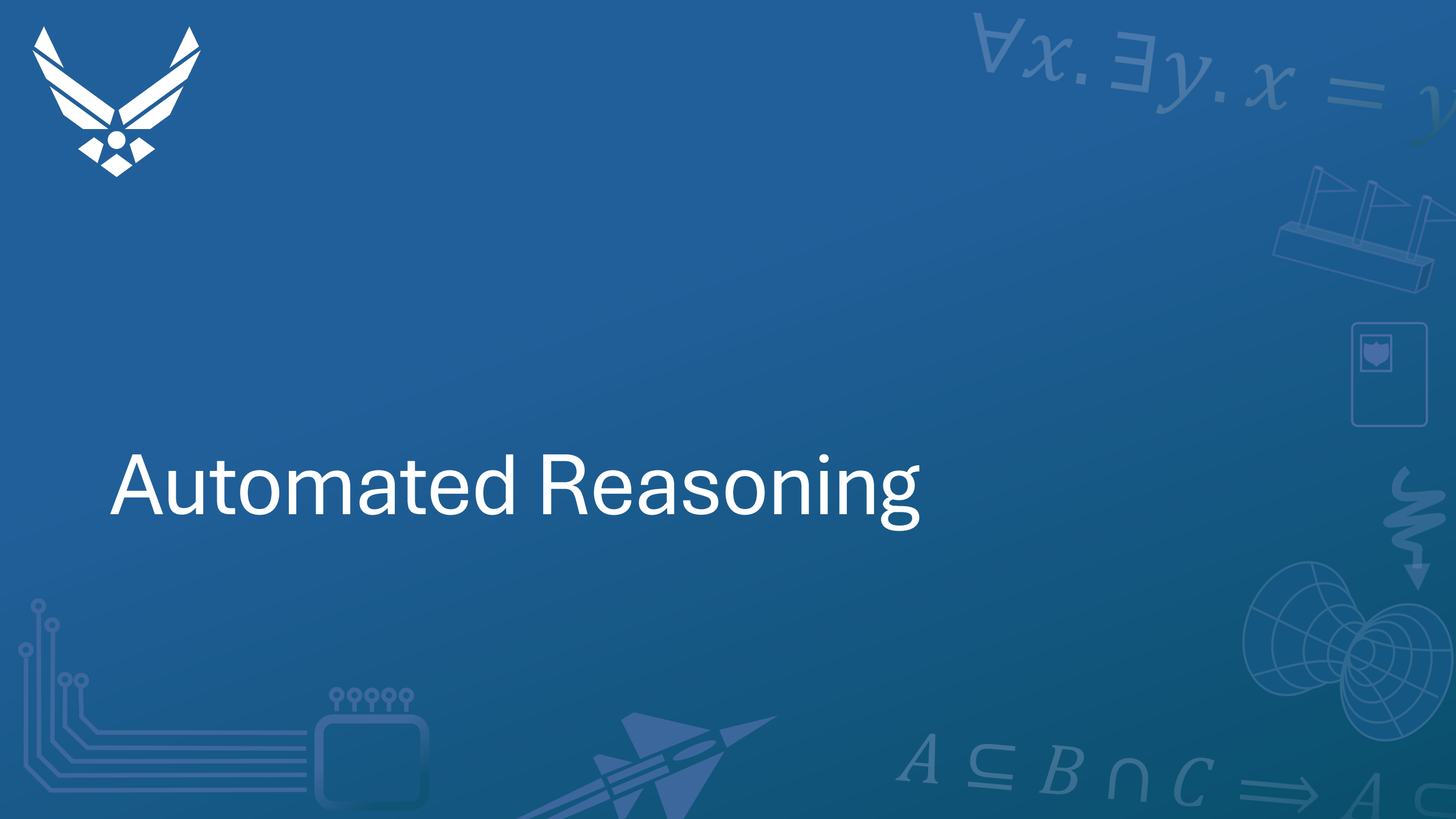


Credit: Gernot Heiser's Advanced Operating Systems Class at UNSW.

Assurance: Substantiating Trust



Credit: Gernot Heiser's Advanced Operating Systems Class at UNSW.



Automated Reasoning

Proof: Math is Broken

- Let $a = b$ (Initial assertion)
- Then $a^2 = ab$ (Mult both sides by a)
- Then $a^2 + a^2 = a^2 + ab$ (Add a^2 to both sides)
- Then $2a^2 = a^2 + ab$ (By simplification)
- Then $2a^2 - 2ab = a^2 + ab - 2ab$ (Sub $2ab$ both)
- Then $2a^2 - 2ab = a^2 - ab$ (By Simplification)

Proof: Math is Broken (Cont.)

- Then $2a^2 - 2ab = a^2 - ab$ (from previous)
- Then $2(a^2 - ab) = 1(a^2 - ab)$ (Rev. distributive)
- Then $2\frac{(a^2-ab)}{(a^2-ab)} = 1\frac{(a^2-ab)}{(a^2-ab)}$ (Divide both)
- Then $2 = 1$ (Simplification)
- Then 🤯

ISAR Programming Examples

```
fun logical_and :: "bool ⇒ bool ⇒ bool" where
 "logical_and True True = True" |
 "logical_and _ _ = False"


fun add :: "nat ⇒ nat ⇒ nat" where
 "add a 0 = a" |
 "add a (Suc v) = Suc (add a v)"
```

Proof Properties

- Machine Checkable
- Traceability
- Non-Ambiguity
- Reusability
- Compositionality
- Sharable
- Soundness
- Completeness
- Repeatable/ Automatable

Proof: Math is NOT Broken

```
theory MathIsBroken
  imports Main  HOL.Real
begin

lemma math_is_broken:
  assumes AB: "(a::real) = (b::real)"
  shows "1 = 2"
proof -
  have AB: "a = b" using assms(1) by simp
  then have "a * a = a * b" by auto
  then have "(a * a) + (a * a) = (a * a) + (a * b)" by simp
  then have "2 * (a * a) = (a * a) + (a * b)" by simp
  then have "2 * (a * a) - 2 * (a * b)  = (a * a) + (a * b) - (2 * (a * b))"
    using AB by simp
  then have "2 * (a * a) - 2 * (a * b) = (a * a) - (a * b)" by simp
  then have "2 * ( (a * a) - (a * b) ) = (a * a) - (a * b)"
    using AB by simp
  then have "2 * ( (a * a) - (a * b) ) / ( (a * a) - (a * b) ) =
                (a * a) - (a * b) / ( (a * a) - (a * b) )"
    using AB by simp
  then have "2 = 1" by simp
  then show ?thesis by simp
end
```

Proof sta

100%

```
Failed to finish proof⌂:
goal (1 subgoal):
 1. a = b ⟹ b = 0
```


```
Failed to finish proof⌂:
goal (1 subgoal):
 1. a = b ⟹ b = 0
```

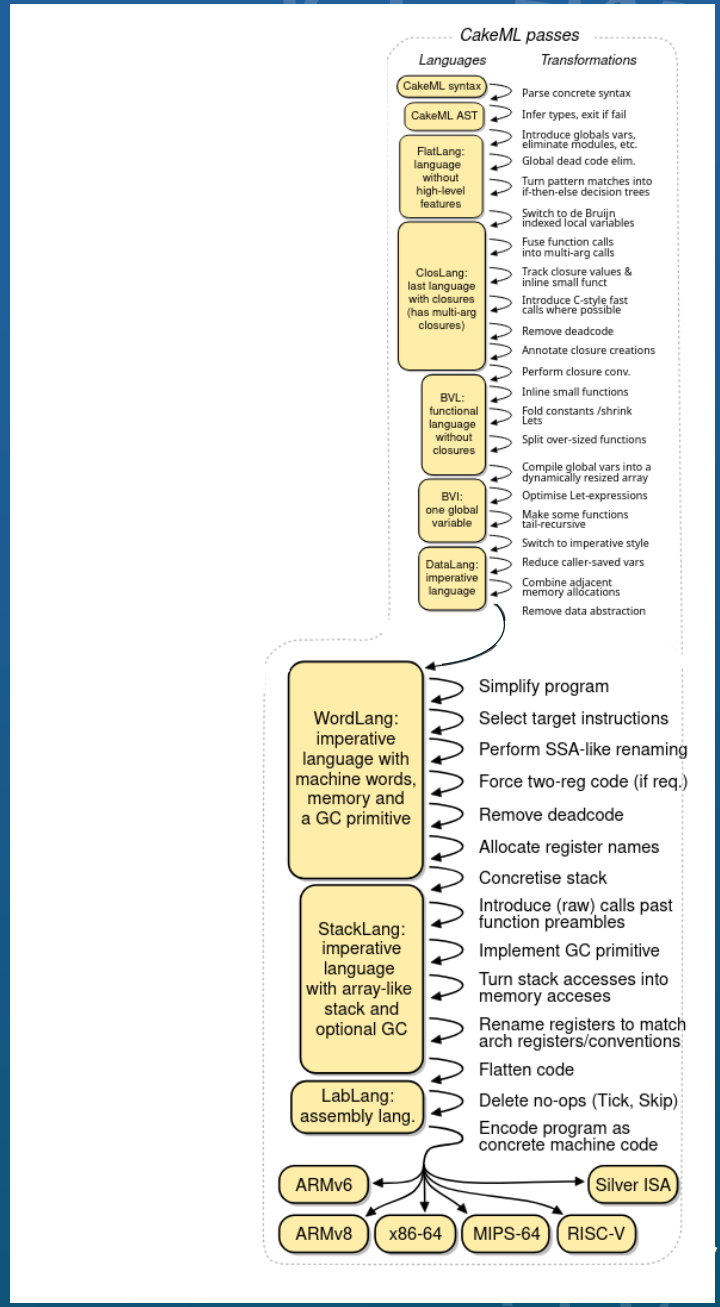
Pancake Verification

Pancake Verification

Pancake Verification

CAKEML
A Verified Implementation of ML

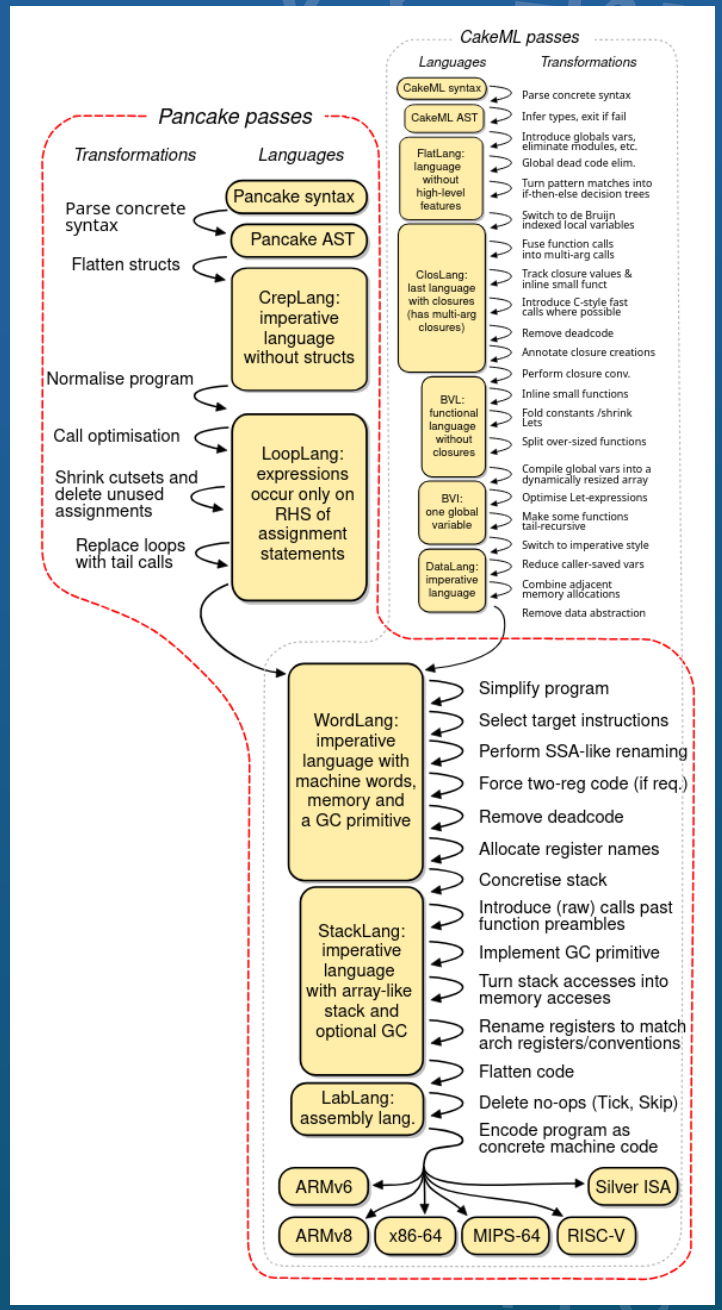
PANCAKE
A Language for
Verified Systems Programming



Pancake Verification

CAKEML
A Verified Implementation of ML

PANCAKE
A Language for
Verified Systems Programming



seL4 Microkernel

Microkernel Minimality Principle

- A concept is tolerated inside the microkernel only if moving it outside the kernel, i.e. permitting competing implementations, would prevent the implementation of the system's required functionality. [Liedtke SOSP'95]

Trusted Computing Base

Traditional Systems

- ~25 Million Lines of Code
- ~75% of TCB is device drivers

seL4

- Drivers in User Space
- ~10,000 Lines of Code
- 900,000 Lines of Proofs

Formally Verified Properties

- Functional Correctness
- Termination
- Memory Safety
- Integrity
- Confidentiality
- Information Flow Security
- Determinism

Modern Software Verification

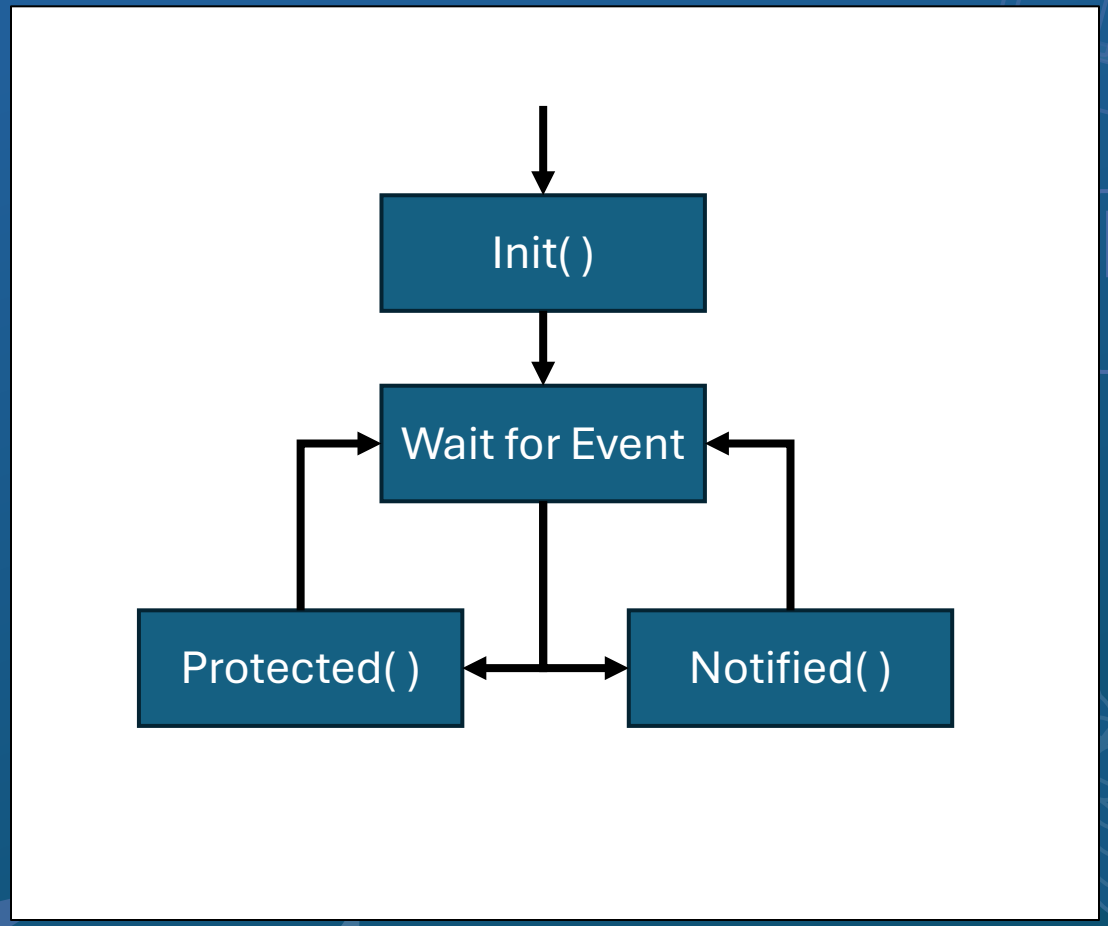
Key Take-Aways:

- Formal methods encouraged
- Automated reasoning
- Proof properties
  - Better assurance
  - Cost savings
- Consider using verified microkernel & formal methods

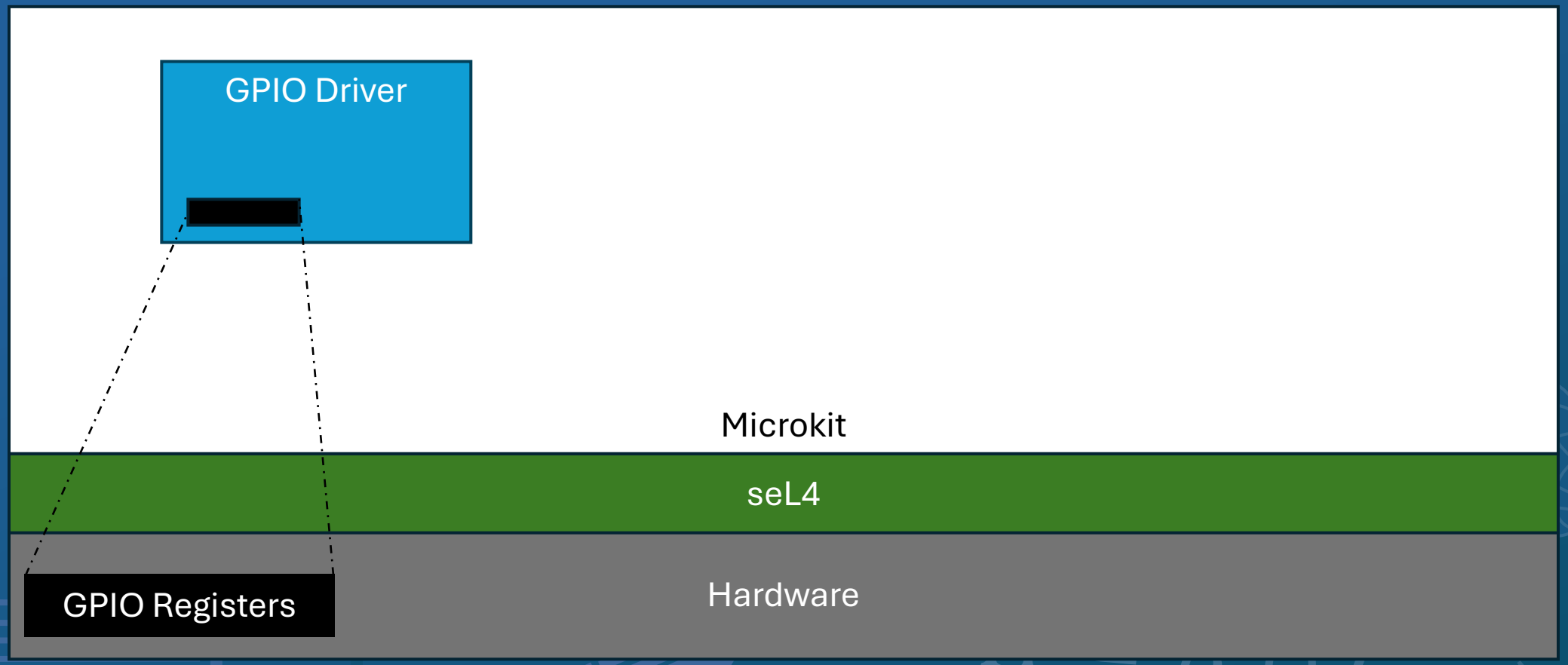
Microkit

Protection Domain

- Init( ) { … }
- Notified( ) { … }
- Protected( ) { … }

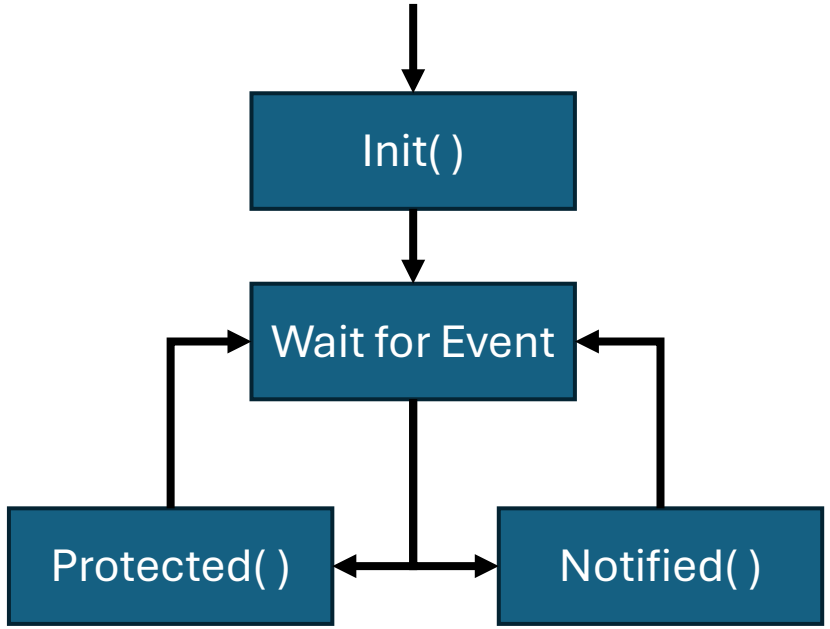


Example Microkit System

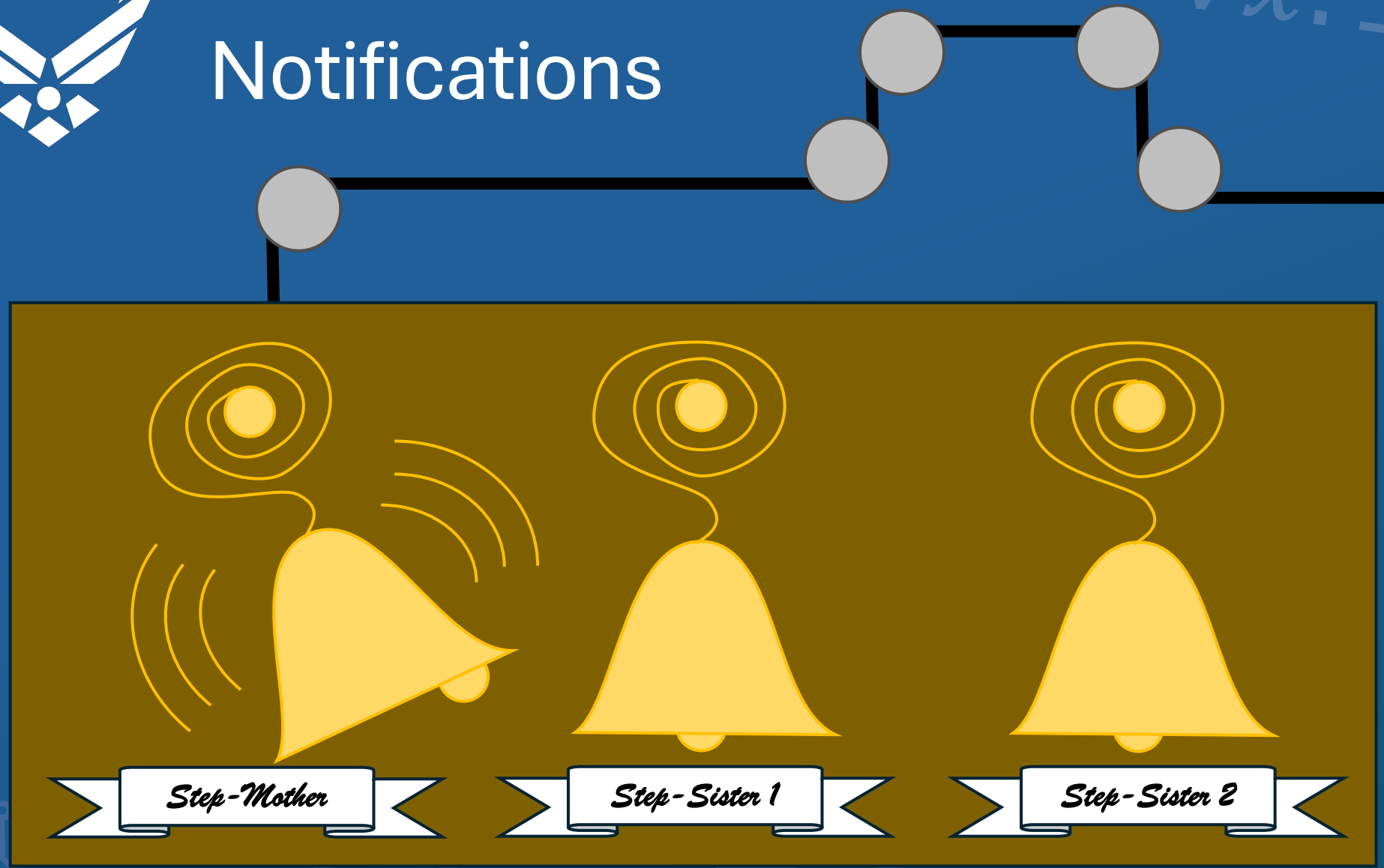


Protection Domain Code

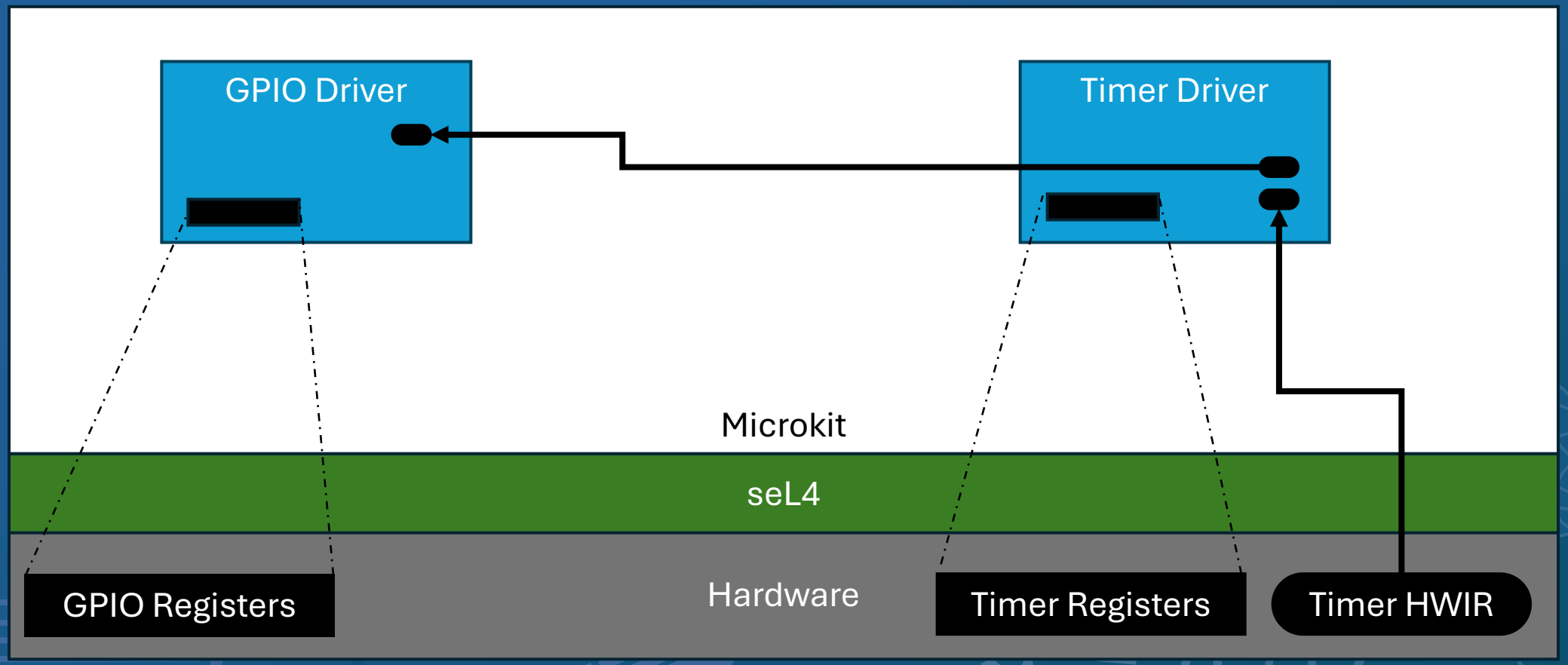
```
init() {
    init_gpio();
    while(true){
        sleep();
        toggle_light();
    }
}

sleep() {
    for(i=0; i<BIG_NUM; i++);
}
```



Notifications



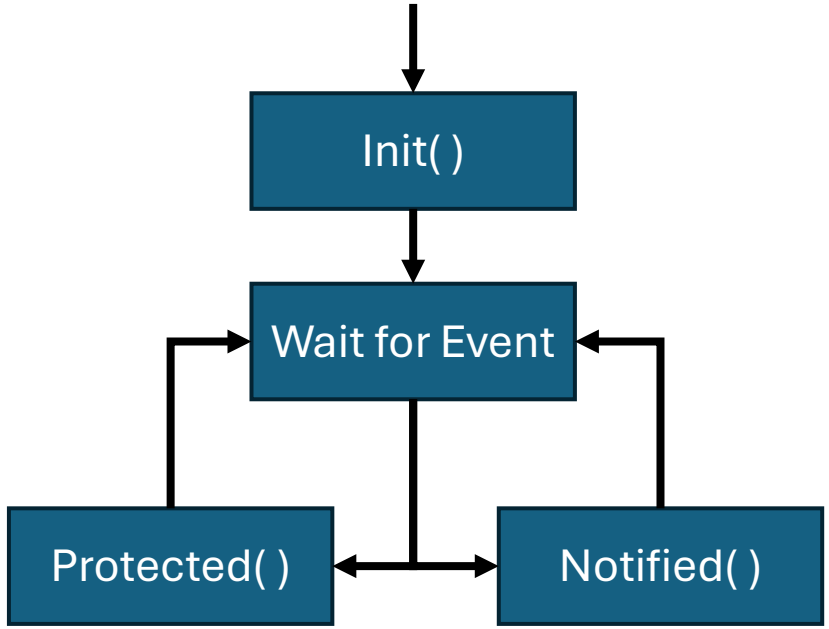
Example Microkit System



```
<?xml version="1.0"?>

<system>

  <memory_region name="gpio" phys_addr="0xFE200000" size="0x1000"/>
  <memory_region name="timer_regs" phys_addr="0xFE003000" size="0x1000"/>

  <protection_domain name="light_blink">
    <program_image path="light_blink.elf" />
    <map mr="gpio" vaddr="0x4000000" perms="rw" cached="false" />
  </protection_domain>

  <protection_domain name="timer_driver">
    <program_image="timer_driver.elf" />
    <map mr="timer_regs" vaddr="0x40000000" perms="rw" cached="false" />
    <irq irq="99" id=1" />
  </protection_domain>

  <channel>
    <end pd="timer_driver" id="2" />
    <end pd="light_blink"  id="1" />
  </channel

</system>
```

Protection Domain Code

```
init() {
    init_gpio();
}

notified (microkit_channel ch) {
    switch (ch){
        case 1:
            toggle_light();
            break;
    }
    microkit_irq_ack(ch);
}
```



Recap

- Challenges of Software Engineering
- Software Assurance
- Automated Reasoning
- seL4 Microkernel
- Microkit

Helpful Links

- https://sel4.org
- https://sel4.org/Learn
- https://docs.sel4.systems/Tutorials
- https://sel4.org/Research/courses.html
- https://www.youtube.com/@seL4
- craig.christianson@us.af.mil

Photo by Mr. Alex R. Lloyd (GS-12):
https://media.defense.gov/2017/Mar/17/2001718114/-1/-1/0/170314-F-EI321-0225.JPG

Backup Slides

Traditional Systems vs seL4

- Trusted Computing Base
- Access Control
- Drivers
- Licensing

Access Control

Traditional Systems

- Role Based
- Users are assigned roles
- Roles are given permissions
- Simple
- Not ideal for least privilege

seL4

- Capability Based
- Unforgeable tokens grant access
- Processes must present capability to access resources
- More complex to manage
- Supports delegation and revocation

Licensing

Traditional Systems

- Windows/iOS
  - Proprietary
- Linux
  - GPL 2.0 (syscall exception)

seL4

- GPL 2.0 (syscall exception)

Drivers

Traditional Systems

- Drivers in kernel space
- Plentiful Drivers

seL4

- Drivers in user space
- Few Drivers
  - File System
  - Storage Devices
  - Memory Controllers
  - Display Drivers