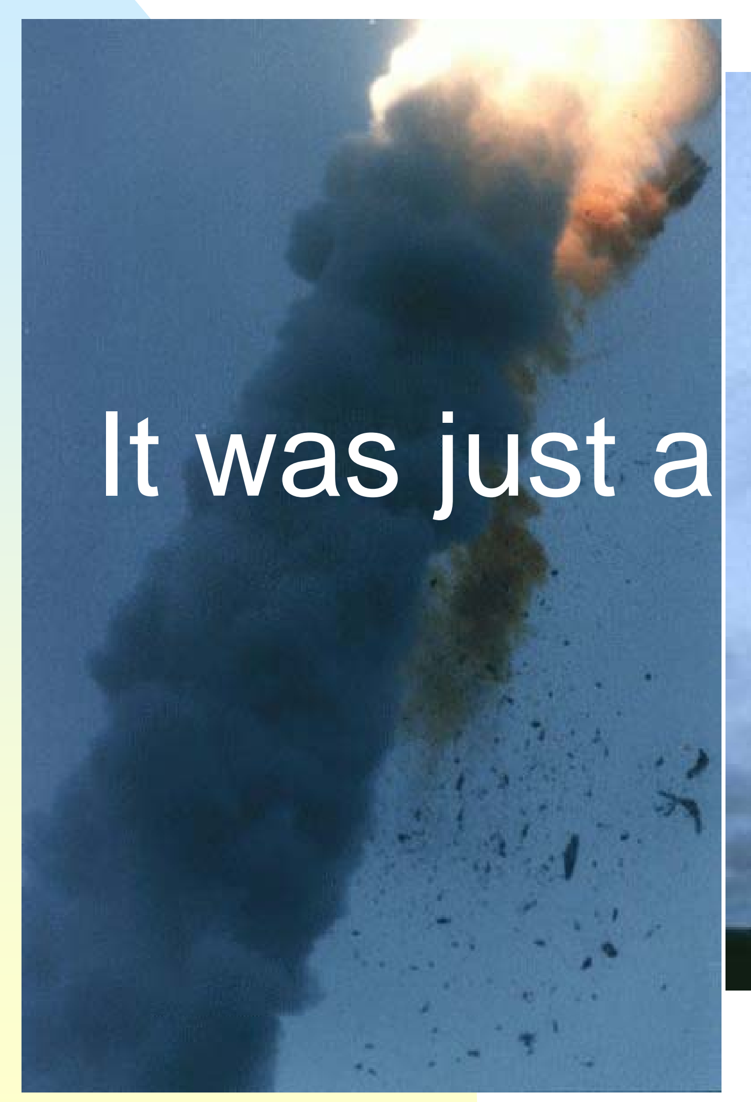
Hardware Monitoring for Proactive Computer Security

John Munson
Jack Meador
Rick Hoover

The Maiden Flight of Ariane 5



It was just a software problem

Software Process Control - A Proactive Approach

- Abstract software process management
  - Model normal activity
  - Sense abnormal activity
  - Control the software application
- Continuous process activity measurement
  - Steady state model
- Restore deviant activity to safe state
  - Identify and respond to any abnormal activity

Benefits

- Hide code protection mechanism in plain sight
- Knowledge of monitoring process of no comfort to an adversary
- Code cannot be altered
  - At runtime
  - At compilation
- System works synergistically with code encryption systems

Software Architecture

- Security cannot be added on
- Security is an integral aspect of software design

The Rationale

- Implement software process control
- Continuous monitoring of software
- Cannot hijack a software system operating under control
- We do not classify
- We control

Our Computer Security Theses

- Software vulnerabilities are not the central issue
- Impossible to design a perfect system
  - Impractical
  - Unnecessary
- Trust no software or no user
- Monitor and control everything and everyone
- Knowledge of the security system should be no comfort to an adversary

Learn from the Past

Control the Future

Classification Methodology

Software Process Control

Past

Present

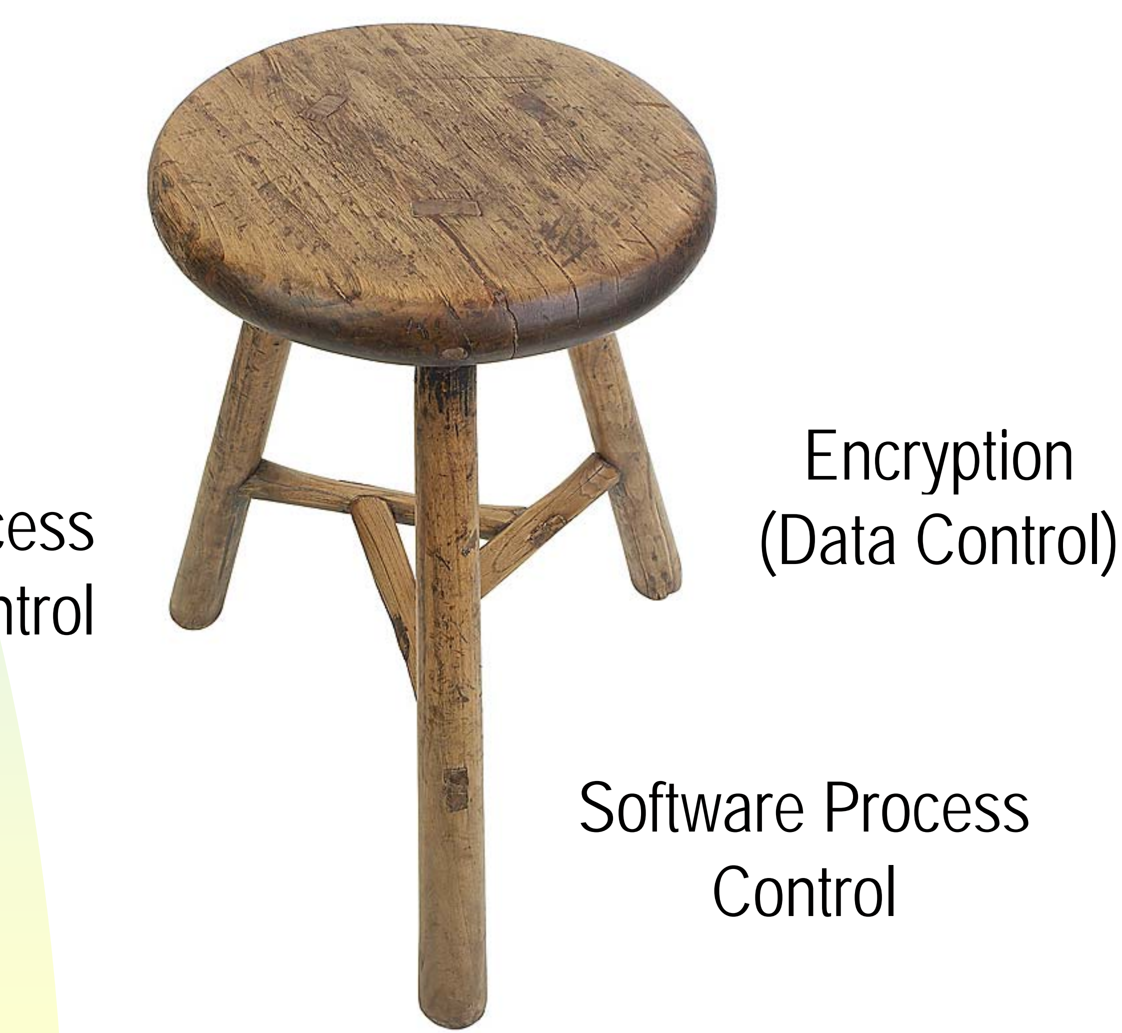
Future

In a future information warfare scenario

You will see your enemy's best shot once …

And it will completely debilitate your infrastructure

The Complete Security Solution



Traditional Security Paradigm

Data
Input Stream

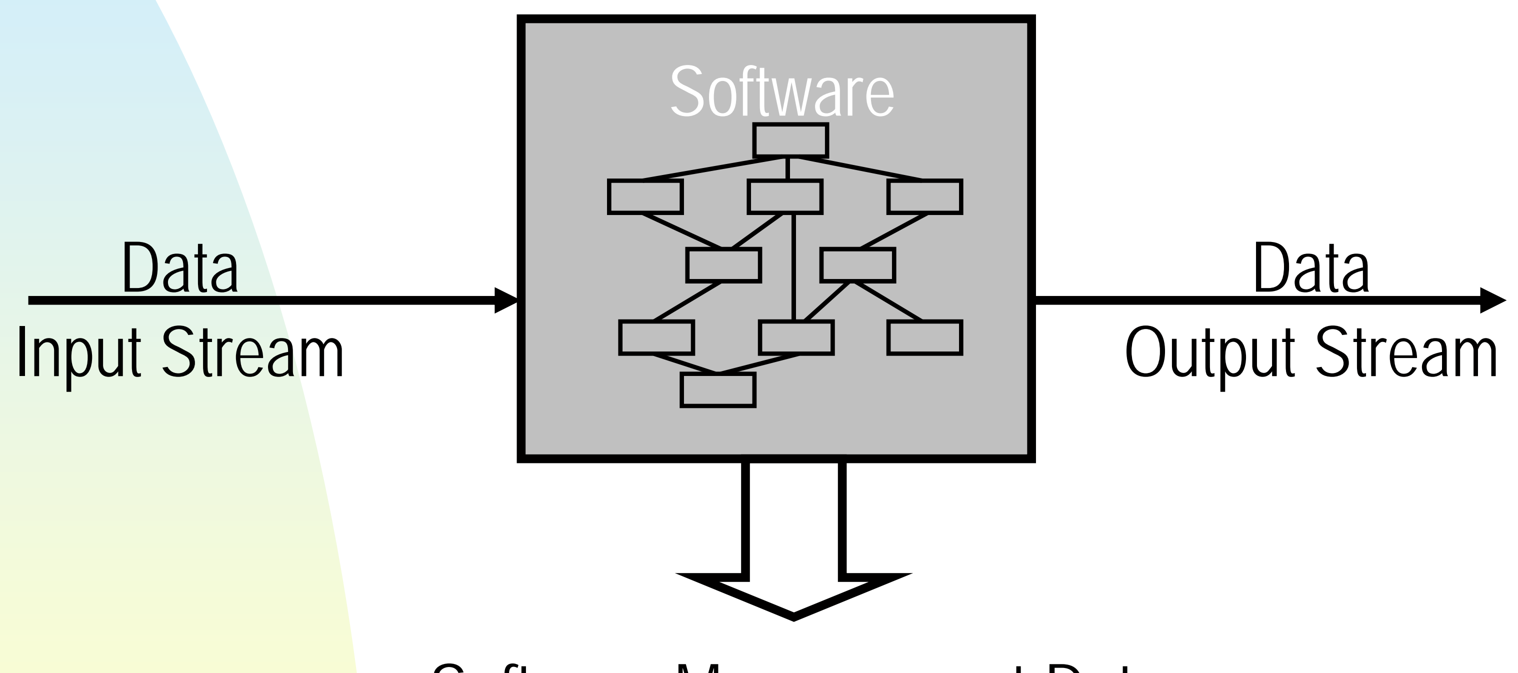
Software

Data
Output Stream

Security Sensor

Security Sensor

The Control Theoretic Approach



The Software Process Control Paradigm

Policy Engine

Analytical Engine

Adaptive Engine

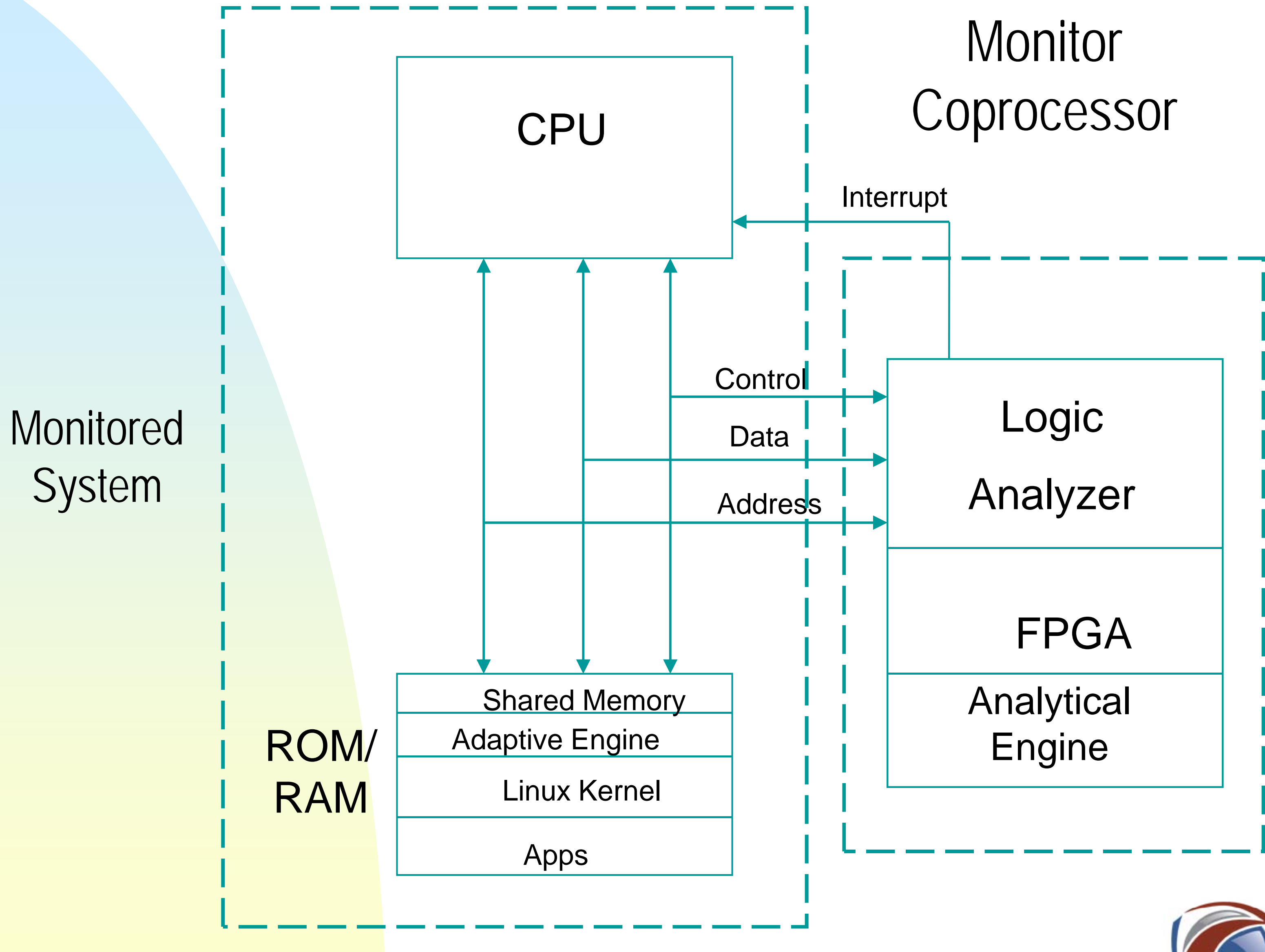
Telemetry

Data In

Software Process

Data Out

Software Process Control Internals



CML

Software Instrumentation

- Sensors placed in the software execution path to measure software activity
- Generate telemetry on the software execution
  - Control flow
  - Data flow
- Monitor all software execution
  - Kernel
  - Library function
  - Syscalls
  - Daemons
  - User programs

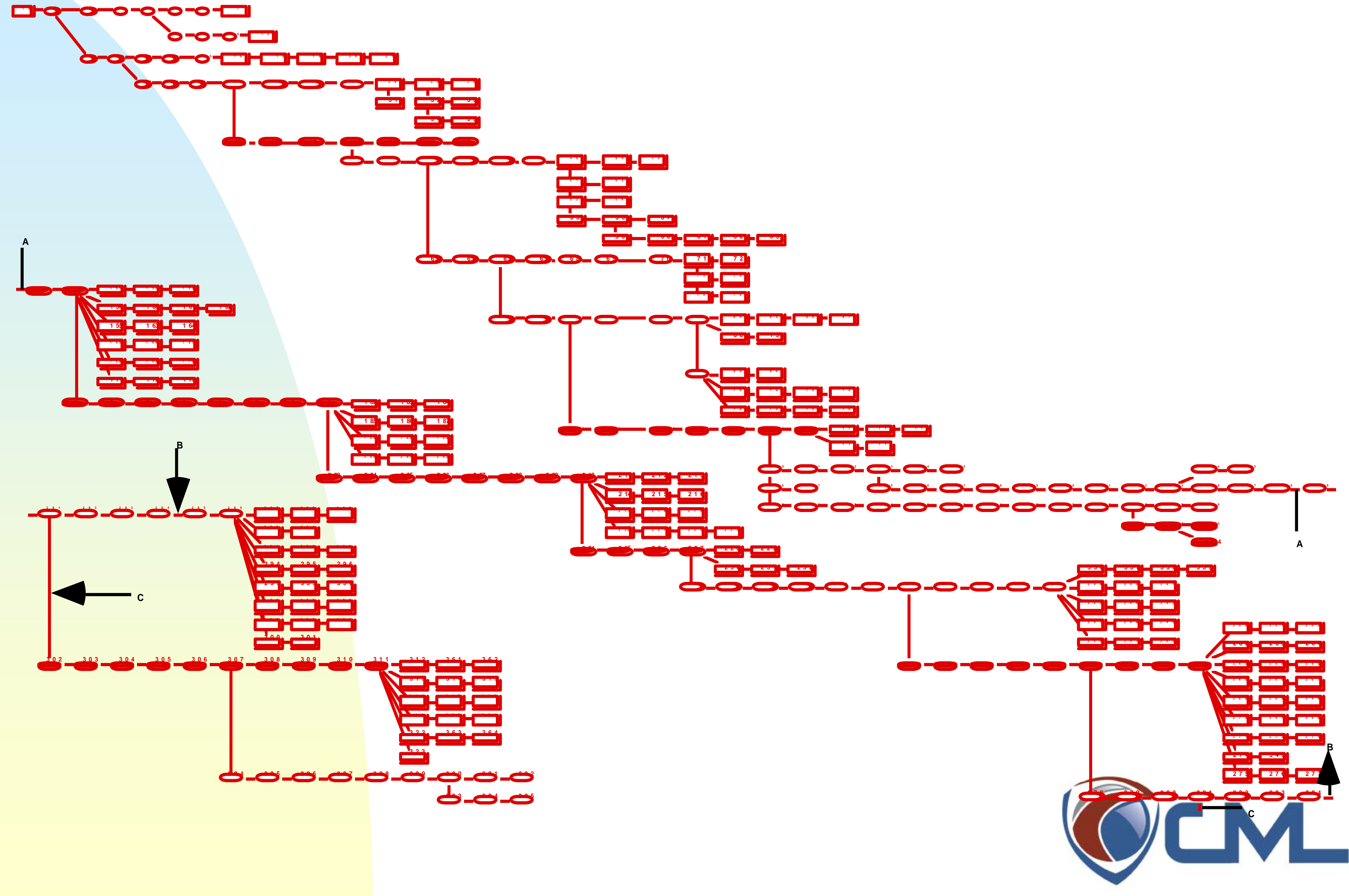
Embedded Platforms

- Target Customer & Applications
  - Avionics, UAV
  - Weapons systems, sensors
  - Future Combat System
  - Mobile encrypted communication systems
- Key Benefits
  - Anti-tamper through certified usage
  - Software recording and measurement
  - Remote monitor & control

Floating Craps Game

- Software systems continually change throughout their useful life
- Each new software release is a new program
- From version to version
  - Some modules are modified
  - Some modules are eliminated
  - Some modules are added

PASS: An Evolving Software System



The Vulnerability Process

Build N

Build N+1

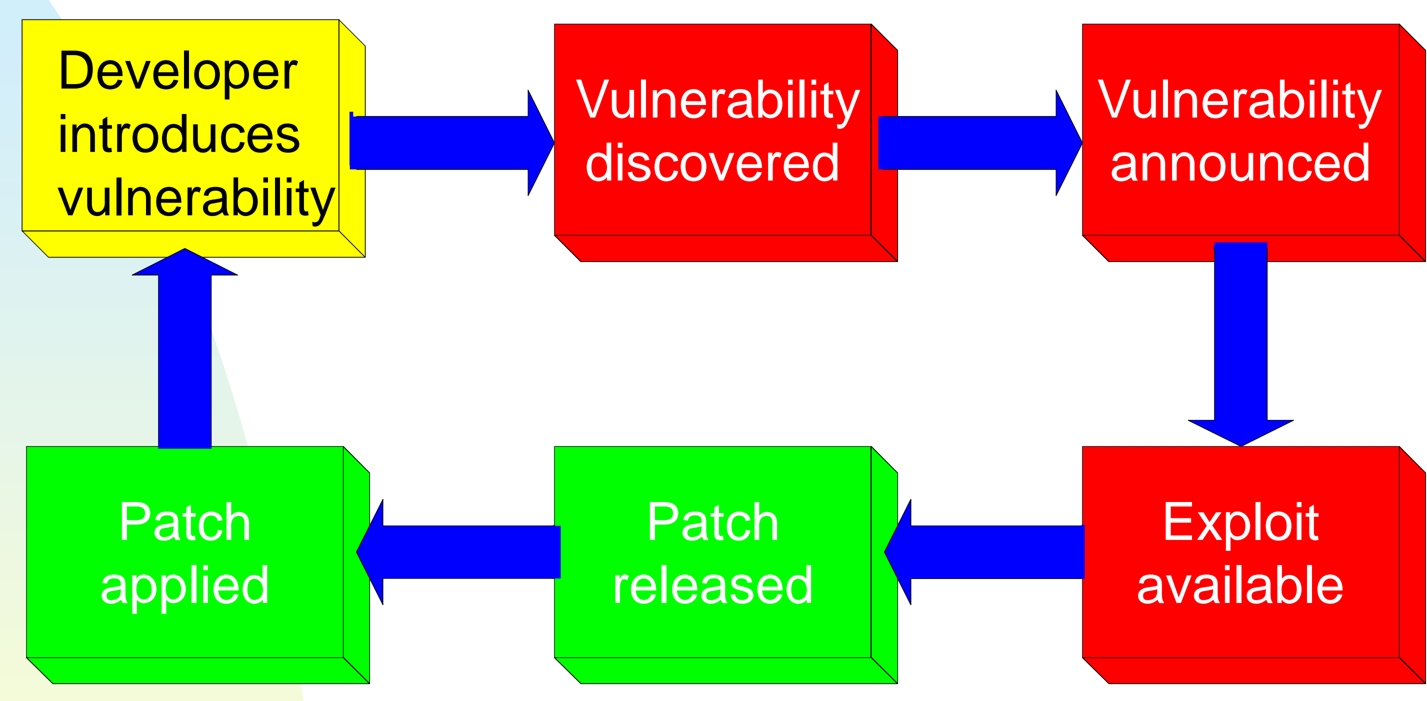
Latent Vulnerabilities

Latent Vulnerabilities

Vulnerabilities Removed

Vulnerabilities Added

The Software Vulnerability Cycle



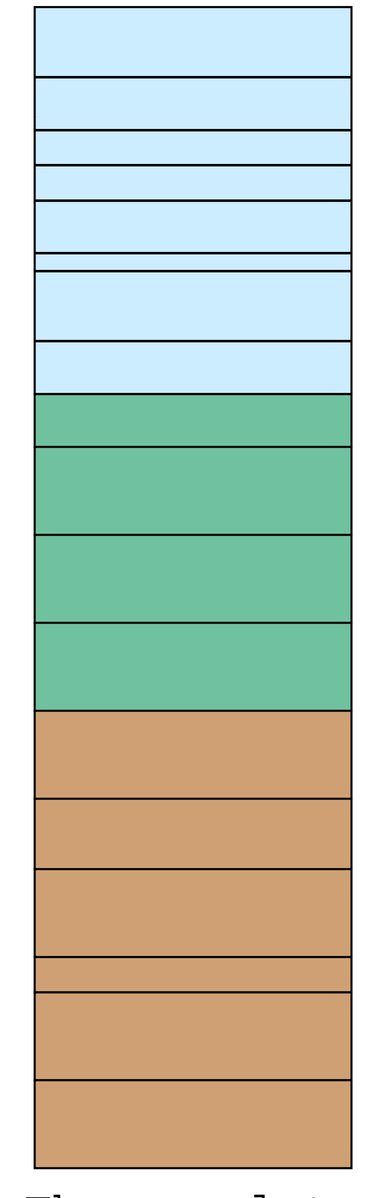
However, the vulnerability is not the problem

The exploit is the problem

The Concept of a Process

- A process is a program in execution
- Three components in a process
  - Binary code from compiler
  - Binary code from `libc`
    - Dynamic linking
    - Static linking
  - Binary code from `syscalls`

A Complete Process



Binary code

Library functions

syscalls

The complete program

Securing the Entire Software Environment

CARMA

Application

Presentation

Session

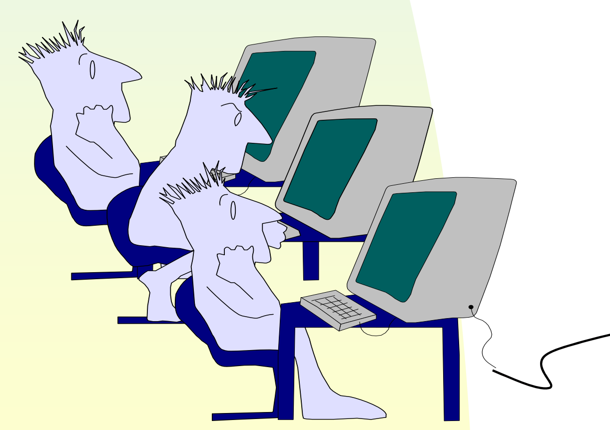
Transport

Network

Datalink

Physical

Users



Functional View of Program Execution

- Programs made up of many structurally independent modules
- Some modules are good (vulnerability free)
- Some modules are bad(vulnerabilities)
- Each functionality exercises only a small set of program modules
- Some functionalities execute secure code
- Some functionalities execute vulnerable code

The Underlying Model

Operations (features)

O x F

Functionalities

F x M

Modules

Execution Consequences

- User performs sequences of operations

$$o_1 \ \mathrm{o}_1 \ o_2 \ o_1 \ o_1 \bullet\bullet\bullet$$

- That result in the expressions of sequences of functionalities

$$f_1 \ f_2 \ f_1 \ f_2 \ f_2 \ f_3 \ f_4 \ f_1 \ f_2 \ f_1 \ f_2 \bullet\bullet\bullet$$

- The program generates a traverse through a call tree

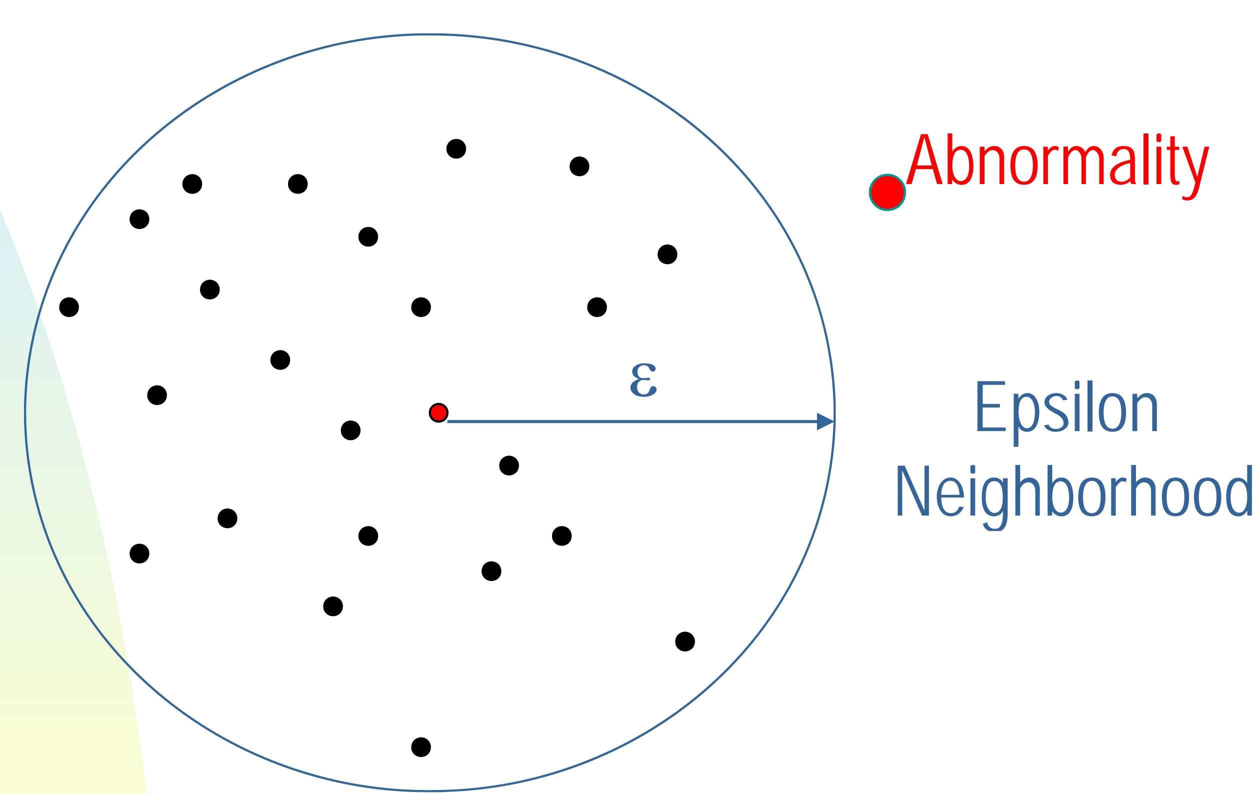
$$m_1 \ m_2 \ m_4 \ m_2 \bullet m_1 \ m_3 \bullet m_1 \ m_2 \ m_4 \bullet m_1 \ m_3 \ m_5 \bullet$$
$$m_1 \ m_3 \bullet m_1 \ m_6 \bullet m_1 \ m_3 \ m_5 \ m_6 \bullet\bullet\bullet$$

The Profiles

- Operational profile
  - describes how a user exercises the program operations $\langle p(O_1), p(O_2), \ldots, p(O_n) \rangle$
- Functional profile
  - describes how program activities are apportioned to functionalities $\langle p(F_1), p(F_2), \ldots, p(F_a) \rangle$
- Module profile
  - distribution of program activity to modules

$$\langle p(M_1), p(M_2), \ldots, p(M_b) \rangle$$

Software Measurement in 2D



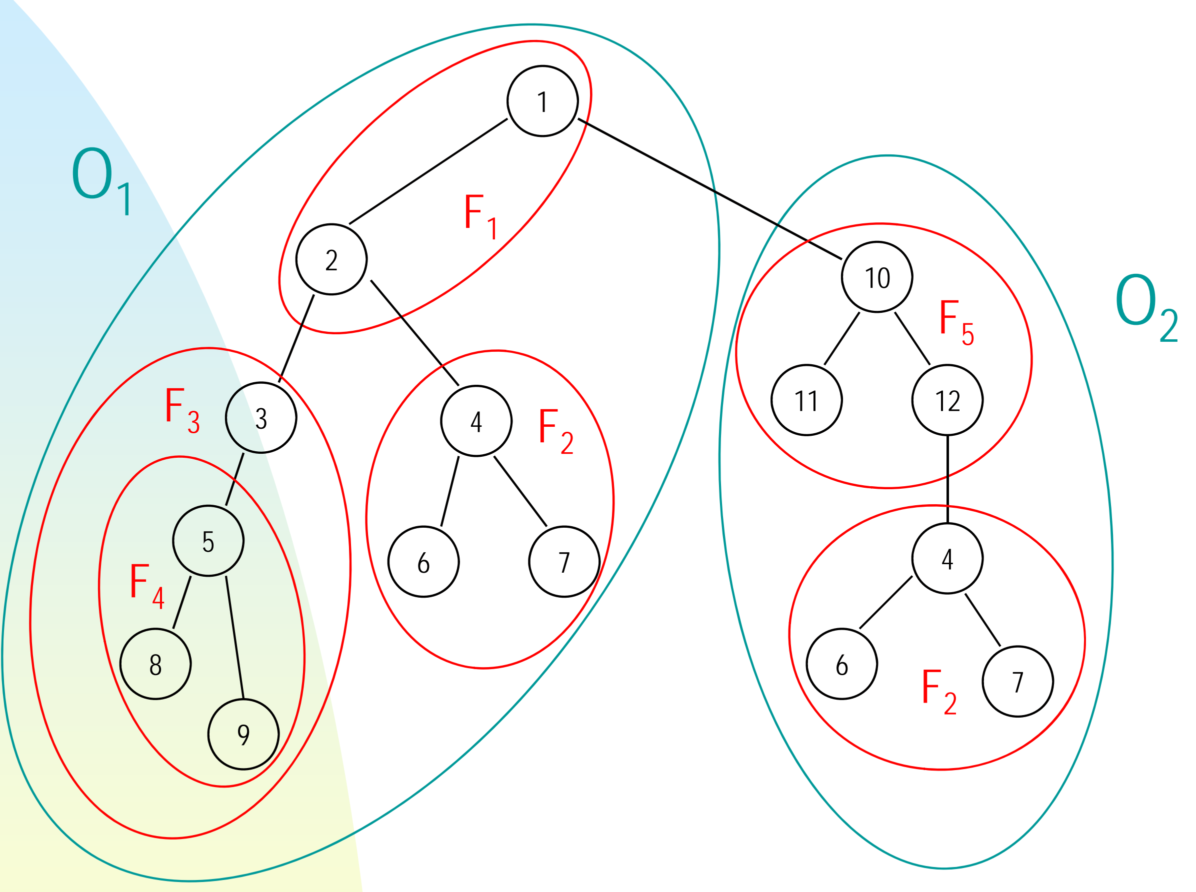
The profile of an attack

0.025
0.02
0.015
0.01
0.005
0
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91 94 97 100

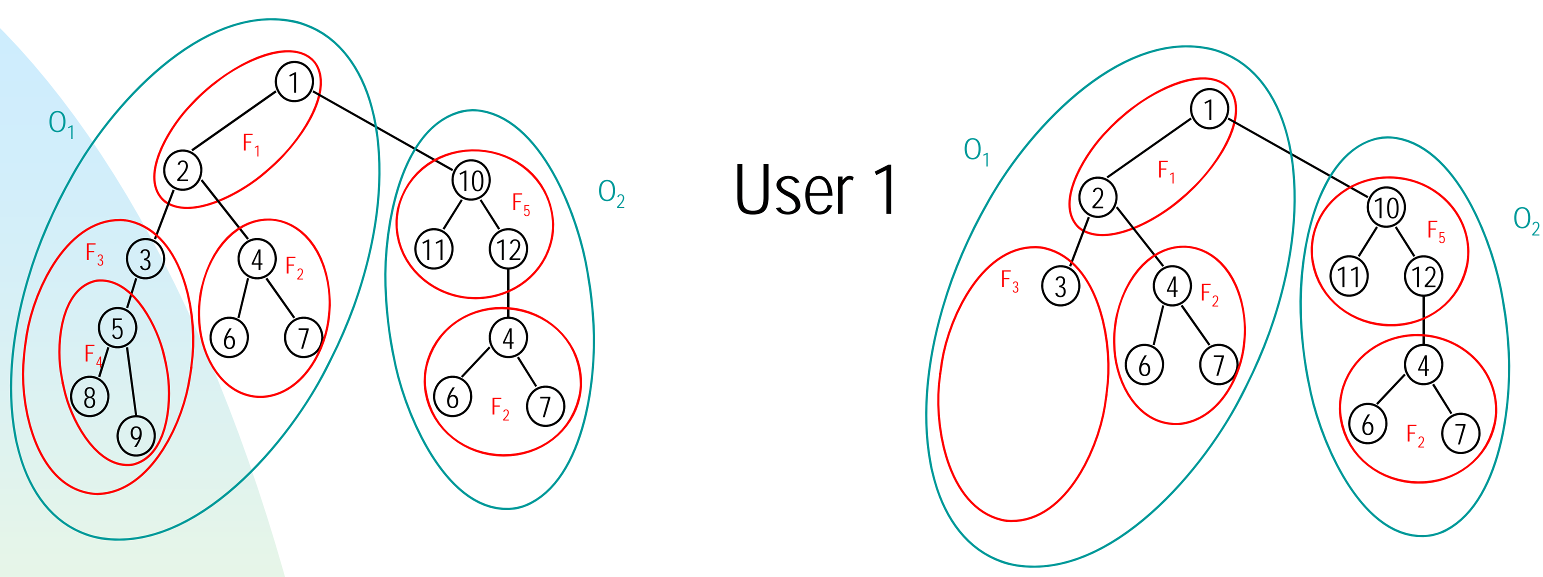
Distance Function

0.02
0.015
0.01
0.005
0
-0.005
-0.01
-0.015
-0.02

Modules are Organized into Call Trees

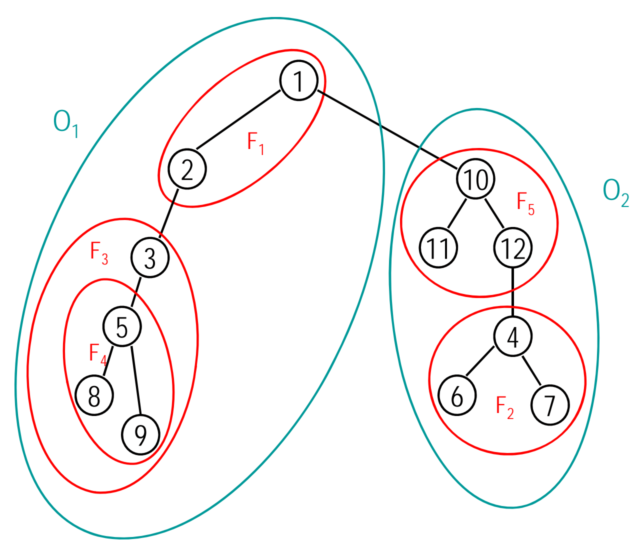


Different Users -- Different Trees



Feasible Tree

User 2



Evolution of Call tree

Program Vocabulary

- Program Execution Alphabet
  - Set of program modules
  - {a, b, c, d, e, f, g, h, i, j}
- Program Vocabulary
  - The vocabulary of the program is defined by the call stack contents
  - {w_1=b, w_2=bc, w_3=bce, w_4=bcf, w_5=bd, w_6=bdg, w_7=bdgh, w_8=bdghi, w_9= bdgk, w_{10}= bdgj }

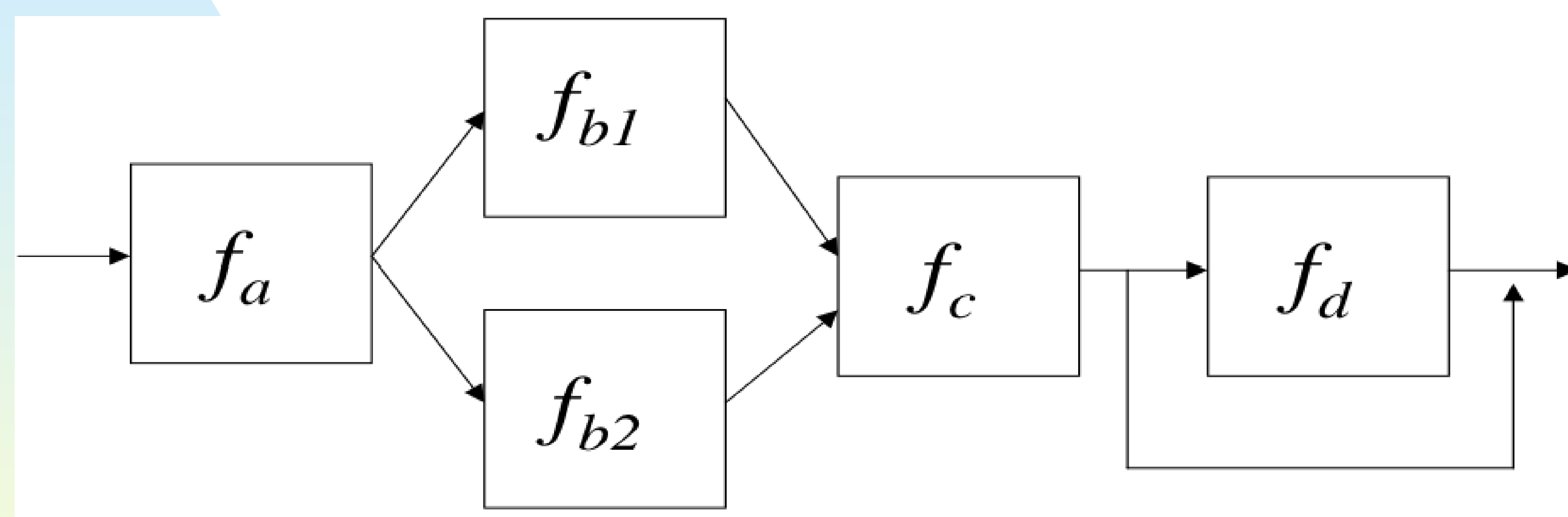
Dimensionality of the Attribute Space

- Dimensionality of dynamic measurement attribute space is very large
- Choose those measures that give the greatest coverage
  - Vocabulary set cardinality
  - Distribution of vocabulary
- Add new measurement attributes to extend coverage

Software Reliability

- Three different terms for the same thing
  - Software security
  - Software reliability
  - Software survivability

Adaptive Strategies



The CARMA System Overview

- Software Process Control
- Asymmetric Multiprocessing System
  - Single CPU dedicated to monitor function
- Private Communication Back Channel
- CARMA System Components
  - Analytical Engine
  - Adaptive Engine
  - Certificate Librarian
  - User Interface

General System Operation

- Each process must first be calibrated to determine is nominal operational characteristics
- The calibration will produce a certificate
- The process may then be monitored against the certificate

The Calibration Activity

Variance

30
25
20
15
10
5
0

1 19 37 55 73 91 109 127 145 163 181 199 217 235 253 271 289 307 325 343

Sequence Size 'n'

System Components

X86 Host

Host Linux Kernel

Adaptive Engine

Application Programs

µBlaze

Analytical Engine

PPC System

Certificate Librarian

PPC Linux Kernel

Reader

Writer

User Interface

PCI Express Bus

The PCI CARMA Analytical Engine

- Monitors program activity by task
- Follows Linux task switching
- Certificate creation
- Certified operation
- Embedded system on μBlaze
  - Runs in real time
  - Calibrate multiple systems simultaneously
  - Monitors all executing software
  - Hardware heap management

The PCI CARMA Certificate Librarian

- Maintains all certificates in the certificate directory in the Linux SD file system
  - Intermediate certificates
  - Operational certificates
- Certificates in library by hashed process signature
- CL maintains a cross-over table to convert signatures to names
- CL maintains a list of new processes to be certified

The PCI CARMA User Interface

- Two independent processes running as kernel modules in Linux
- Reader
  - Initiated as an SSH session by the user
  - Kicks off the writer and the CL
- Writer
  - Initiates an SSH session on the Policy machine
  - Transfers all output from PCI CARMA to the Policy machine

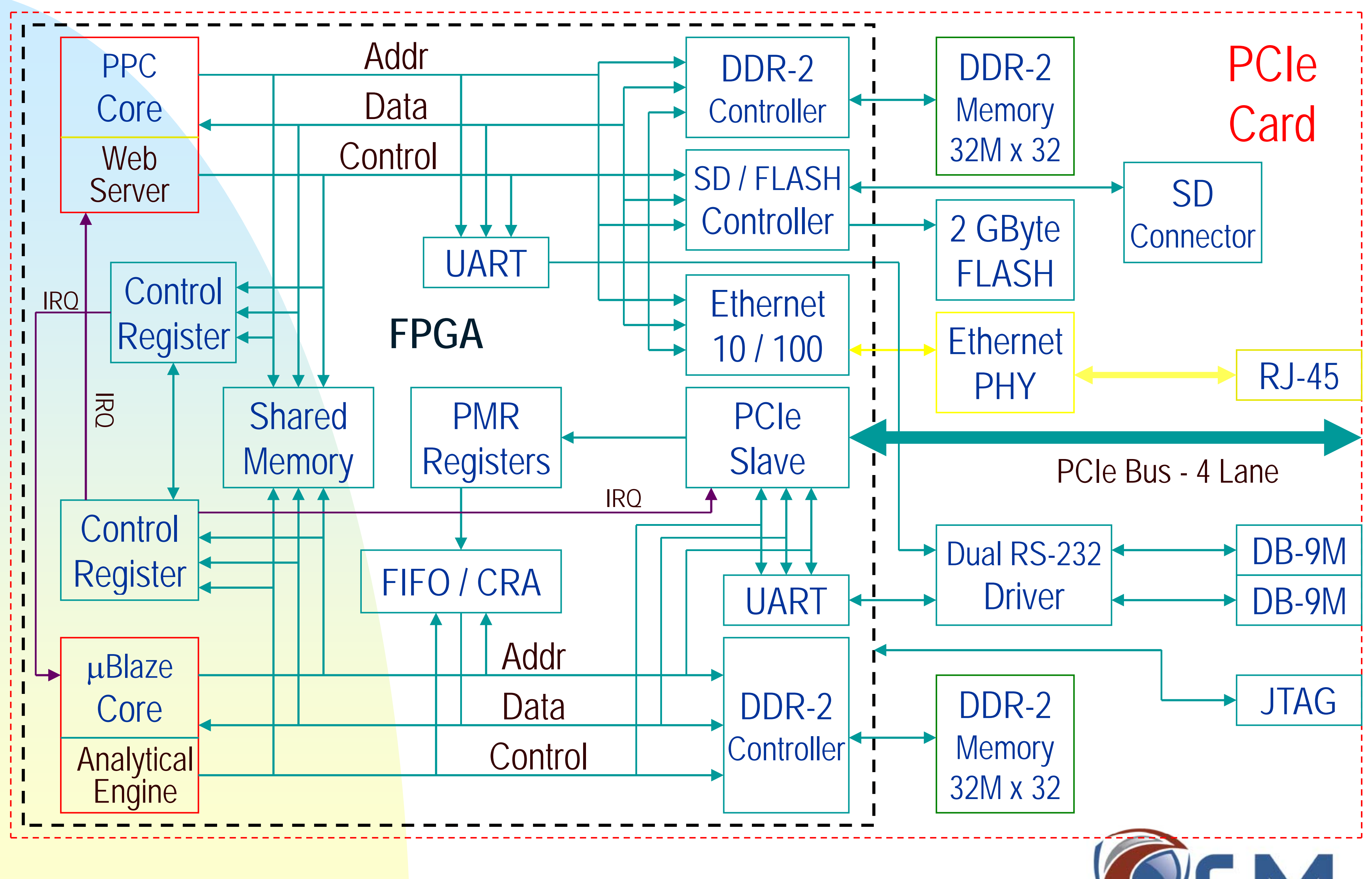
The PCI CARMA Adaptive Engine

- Runs as an interrupt service on the host Linux machine
- Gains control when a process diverges from its certified behavior
- Possible actions
  - Kill process
  - Nice process
  - Hold process
  - Modify process

The PCI CARMA System

- Implemented on our PCI Express card
- Two On-board CPUs
  - Power PC
    - Runs Linux
      - Certificate librarian
      - User interface
  - μBlaze
    - Runs analytical engine as embedded app (no operating system)
    - Host communication via PCIe

PCI CARMA Architecture



The PCI CARMA Board

Secure Process Monitoring

Router

Public Network

IP Sessions

Linux PC | CARMA

Linux PC | CARMA

Linux PC | CARMA

● ● ●

Linux PC | CARMA

Local Users

Private Network

Router

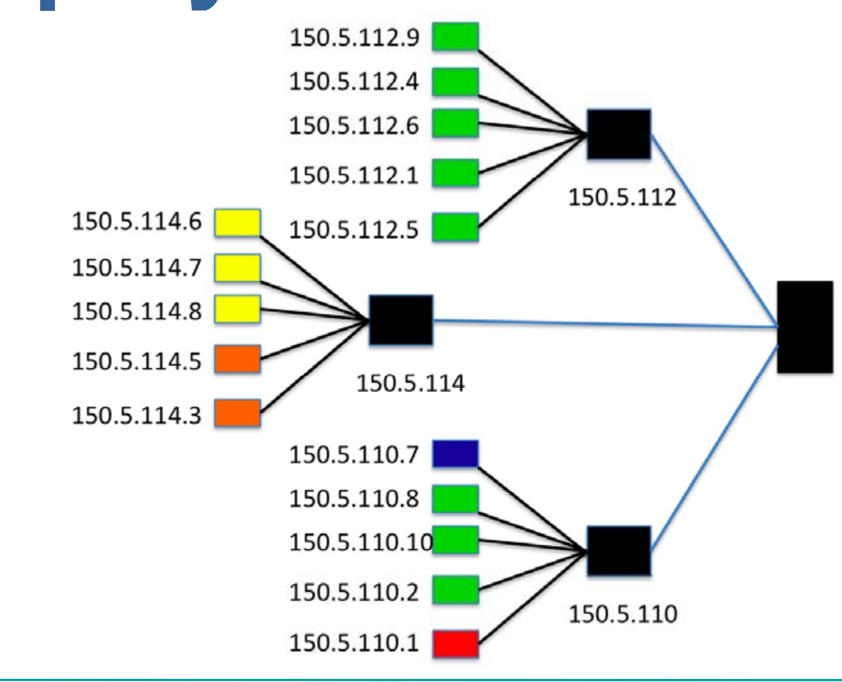
Policy Engine

Security Administrator

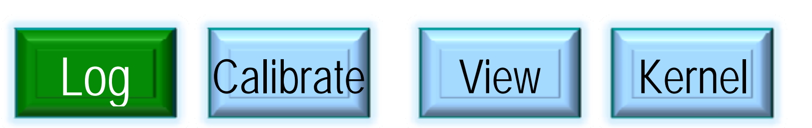
The Policy Engine Roles

- Initial calibration activity
  - Multiple calibrations
  - Multiple machines
  - Multiple users
- Monitor software execution
  - By user
  - By IP Session
- Forensic analysis of abnormal activity

Policy Engine Display



IP Address	Action	Proc Name	PID
150.5.110.1	killed	sed	1745
150.5.110.1	killed	lp	1841
150.5.114.3	suspended	master1	268
150.5.114.5	suspended	payroll	423
150.5.114.8	niced	user1	2078
150.5.114.7	niced	visual	1132
150.5.110.7	watch	testlib	1567



Log for 150.110.1 PID 1745

```
2009-06-01 14:33:53Z sed        0004 0006
2009-05-30 10:42:23Z sed        0008 0006
2009-05-29 11:36:53Z sed        0004 0007
2009-05-29 08:19:53Z sed        0005 0006
```

The Real PCI CARMA

- It is a measurement tool
- It can be used in many different engineering opportunities
  - Process Control
  - Performance
  - Testing
  - Reliability
  - Availability
  - Security

PCI μCARMA: Social security

- Structure certificates two ways
  - By process
  - By user (UID)
- Detect abnormal user activity
- Manage the activity
  - Notify Security Administrator
  - Log relevant attack data
- Render the attack harmless with sandbox

One of the most important concepts in the modern information warfare scenario is the notion of asymmetry.

Software process control provides a distinct defensive advantage

This is the most important contribution of the CARMA technology.