



# THE RELIABLE COMPUTING BASE

## A Paradigm for Software-Based Reliability

Michael Engel (TU Dortmund),  
Björn Döbel (TU Dresden)

Braunschweig, 19.09.2012

# Motivation

- Increasing hardware error rate
- Hardening all hardware is too expensive

---

<sup>1</sup> Arlat et al.: *Dependability of COTS microkernel-based systems*, IEEE ToC 2002

<sup>2</sup> Saggese et al.: *An experimental study of soft errors in microprocessors*, IEEE Micro 2005

<sup>3</sup> Engel et al.: *Unreliable yet Useful – Reliability Annotations for Data in Cyber-Physical Systems*, WS4C 2011

# Motivation

- Increasing hardware error rate
- Hardening all hardware is too expensive
- ... and unnecessary due to masking:
  - Arlat:<sup>1</sup> 30% software masking in a microkernel
  - Saggese:<sup>2</sup> 30% hardware masking in a microprocessor
  - Engel:<sup>3</sup> Data exposes different levels of vulnerability
- Software-implemented fault tolerance tries to address this

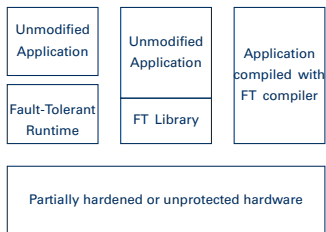
---

<sup>1</sup> Arlat et al.: *Dependability of COTS microkernel-based systems*, IEEE ToC 2002

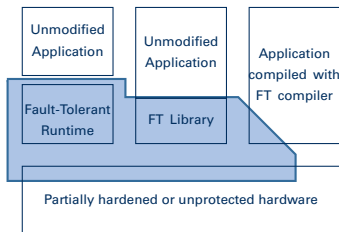
<sup>2</sup> Saggese et al.: *An experimental study of soft errors in microprocessors*, IEEE Micro 2005

<sup>3</sup> Engel et al.: *Unreliable yet Useful – Reliability Annotations for Data in Cyber-Physical Systems*, WS4C 2011

# Making Fault Tolerance Fault-Tolerant



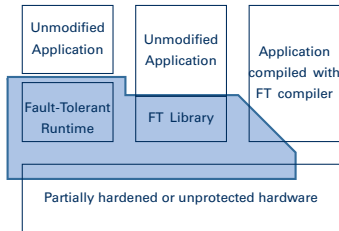
## Making Fault Tolerance Fault-Tolerant



SW Fault Tolerance splits the software stack in two parts:

1. Protected set of software components
2. Set of components providing protection – The **Reliable Computing Base (RCB)**

# Making Fault Tolerance Fault-Tolerant



SW Fault Tolerance splits the software stack in two parts:

1. Protected set of software components
2. Set of components providing protection – The **Reliable Computing Base (RCB)**

## Research questions

1. Which components (hardware and software) are part of the RCB?
2. How can we ensure that RCB components are protected against soft errors?
3. How can we minimize the RCB (and do we need to do it at all)?

# Digression: Trusted Computing Base

## Rushby

*... a combination of a kernel and trusted processes, which are permitted to bypass a system's security policies ...<sup>a</sup>*

---

<sup>a</sup> J.M.Rushby: *Design and Verification of Secure Systems*, SOSP 1981

## Digression: Trusted Computing Base

### Rushby

*... a combination of a kernel and trusted processes, which are permitted to bypass a system's security policies ...<sup>a</sup>*

---

<sup>a</sup> J.M.Rushby: *Design and Verification of Secure Systems*, SOSP 1981

### Lampson

*... a small amount of software and hardware that security depends on and that we distinguish from a much larger amount that can misbehave without affecting security.<sup>a</sup>*

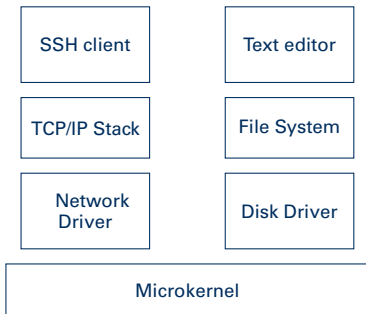
---

<sup>a</sup> Lampson et al.: *Authentication in Distributed Systems – Theory and Practice*, SOSP 1991



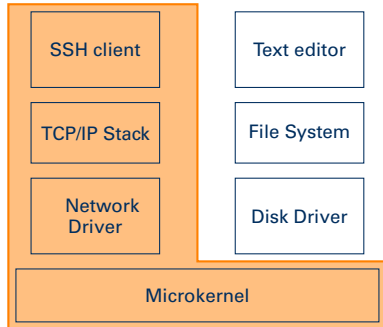
## TCB: Measuring Trust

- Define set of hardware and software component a user needs to trust
- Intuition: smaller TCB implies more trustworthy system
- Common metric: lines of code
- Application-specific TCB
  - Applications only require a subset of whole system's features
  - Subset is known in advance
  - Isolate TCB components from non-TCB components



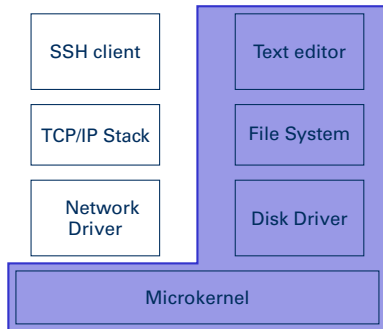
# TCB: Measuring Trust

- Define set of hardware and software component a user needs to trust
- Intuition: smaller TCB implies more trustworthy system
- Common metric: lines of code
- Application-specific TCB
  - Applications only require a subset of whole system's features
  - Subset is known in advance
  - Isolate TCB components from non-TCB components



## TCB: Measuring Trust

- Define set of hardware and software component a user needs to trust
- Intuition: smaller TCB implies more trustworthy system
- Common metric: lines of code
- **Application-specific TCB**
  - Applications only require a subset of whole system's features
  - Subset is known in advance
  - Isolate TCB components from non-TCB components



## Reliable Computing Base

The *Reliable Computing Base* (RCB) is a subset of software and hardware components that ensures the operation of software-based fault-tolerance methods and that we distinguish from a much larger amount of components that can be affected by faults without affecting the program's desired results.

## Reliable Computing Base

The *Reliable Computing Base* (RCB) is a subset of software and hardware components that ensures the operation of software-based fault-tolerance methods and that we distinguish from a much larger amount of components that can be affected by faults without affecting the program's desired results.

## Reliable Computing Base

The *Reliable Computing Base* (RCB) is a subset of software and hardware components that ensures the operation of software-based fault-tolerance methods and that we distinguish from a much larger amount of components that can be affected by faults without affecting the program's desired results.

## Reliable Computing Base

The *Reliable Computing Base* (RCB) is a subset of software and hardware components that ensures the operation of software-based fault-tolerance methods and that we distinguish from a **much larger amount of components** that can be affected by faults without affecting the program's desired results.

## Reliable Computing Base

The *Reliable Computing Base* (RCB) is a subset of software and hardware components that ensures the operation of software-based fault-tolerance methods and that we distinguish from a much larger amount of components that can be affected by faults without affecting the program's desired results.



# Minimizing the RCB

- RCB requires additional resources → minimize those resources
- TCB minimization: simply aim to reduce lines of code
- However, no single metric for the RCB:

---

|                |   |
|----------------|---|
| Energy         | Watts                                   |
| Chip Area      | mm <sup>2</sup> , number of logic gates |
| Execution Time | seconds                                 |
| Design Effort  | lines of code, person months            |
| Vulnerability  | AVF (hardware), PVF (software)          |

---

- Practical minimization will probably be a combination of several metrics

## Minimizing the RCB

- RCB requires additional resources → minimize those resources
- TCB minimization: simply aim to reduce lines of code
- However, no single metric for the RCB:

---

|                |   |
|----------------|---|
| Energy         | Watts                                   |
| Chip Area      | mm <sup>2</sup> , number of logic gates |
| Execution Time | seconds                                 |
| Design Effort  | lines of code, person months            |
| Vulnerability  | AVF (hardware), PVF (software)          |

---

- Practical minimization will probably be a combination of several metrics
  - Please let's not call it **energy-area-vulnerability-delay product**, though!

## Minimizing the RCB

- RCB requires additional resources → minimize those resources
- TCB minimization: simply aim to reduce lines of code
- However, no single metric for the RCB:

---

|                |   |
|----------------|---|
| Energy         | Watts                                   |
| Chip Area      | mm <sup>2</sup> , number of logic gates |
| Execution Time | seconds                                 |
| Design Effort  | lines of code, person months            |
| Vulnerability  | AVF (hardware), PVF (software)          |

---

- Practical minimization will probably be a combination of several metrics
  - Please let's not call it energy–area–vulnerability–delay product, though!

## Digression: Measuring Program Vulnerability

- Hardware analysis: Architectural Vulnerability Factor<sup>4</sup>
  - Inputs: Hardware component  $H$ , workload run of  $N$  cycles
  - Ratio of architecturally correct bits (ACE bits) during one run
  - Computation of  $H$ 's AVF:

$$AVF_H := \frac{\sum_{i=1}^N (\text{ACE bits in } H \text{ at cycle } i)}{\text{Bits in } H \times N}$$

---

<sup>4</sup> Mukherjee et al.: *A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor*, IEEE Micro 2003

# Digression: Measuring Program Vulnerability

- Hardware analysis: Architectural Vulnerability Factor<sup>4</sup>
  - Inputs: Hardware component  $H$ , workload run of  $N$  cycles
  - Ratio of architecturally correct bits (ACE bits) during one run
  - Computation of  $H$ 's AVF:

$$AVF_H := \frac{\sum_{i=1}^N (\text{ACE bits in } H \text{ at cycle } i)}{\text{Bits in } H \times N}$$

- Common AVFs<sup>4</sup>:

|                   |        |
|-------------------|--------|
| Program counter   | ~ 100% |
| Branch predictor  | ~ 0%   |
| Instruction Queue | 28%    |
- Program from software developer's perspective: depends on a proper hardware model to be available

---

<sup>4</sup> Mukherjee et al.: *A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor*, IEEE Micro 2003

# The Program Vulnerability Factor

- Idea: replace hardware with abstract resources<sup>5</sup>
  - Registers
  - Memory words
  - Instruction classes (e.g., ALU instructions)
- Input: instruction trace of a workload of  $N$  cycles
- Computation of the workload's PVF with respect to resource  $r$ :

$$PVF_r := \frac{\sum_{i=1}^N (\text{ACE bits in resource } r \text{ at cycle } i)}{\text{Bits in } r \times N}$$

---

<sup>5</sup> Sridharan, Kaeli: *Using Hardware Vulnerability Factors to Enhance AVF Analysis*, ISCA 2010  
Braunschweig, 19.09.2012

# Minimal RCB – Where are we?

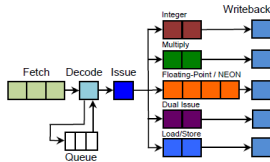


Figure 1 Cortex-A7 Pipeline

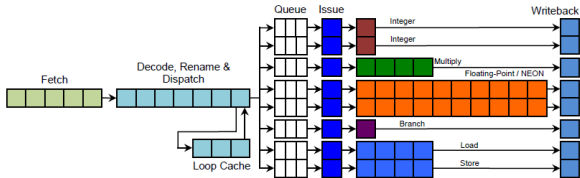


Figure 2 Cortex-A15 Pipeline

[ARM Ltd: *Big.LITTLE Processing with ARM Cortex*, Whitepaper 2011]

## Minimal RCB – Where are we?

- Heterogeneous cores are the future
  - Have a few resilient cores and many non-resilient ones<sup>6</sup>
- Heterogeneous memory architectures
  - Restrict code execution to scratchpad memory with error detection<sup>7</sup>

### Research issues

- General: How do we determine the RCB and its size?
- Hardware: How do we build RCB-protecting components? Which of these components are needed?
- Operating System: How do we manage RCB-heterogeneous resources?
- Software: Can we leverage dedicated RCB hardware to protect our applications?

<sup>6</sup> Döbel, Härtig: *Who watches the watchmen? – Protecting Operating System Reliability Mechanisms*, HotDep 2012

<sup>7</sup> Falk, Kleinsorge: *Optimal Static WCET-aware Scratchpad Allocation of Program Code*, DAC 2009