TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute of Systems Architecture, Operating Systems Group

MICROKERNEL-BASED OPERATING SYSTEMS

based on material by
Maksym Planeta and Björn Döbel

Dependable Operating Systems

https://tud.de/inf/os/studium/vorlesungen/mos

HORST SCHIRMEIER

Murphy’s Law

“If there's more than one way to do a job, and one of those ways will result in disaster, then somebody will do it that way.”

– Edward Murphy jr.

Goal of this Lecture

- **Operating systems in critical environments**
  - Safety
  - Security
  - Performance
- Focus in this lecture: **Safety**

Agenda

- Dependability: Attributes, Threats and Means
- Software Faults
  - Empirical Study: Linux
  - MISRA C/C++ and Safe Languages
  - Compartmentalization and Redundancy
  - Software Verification
- Hardware Faults
  - Coarse- and Fine-grained Redundant Multithreading
- Summary

Agenda

- **Dependability: Attributes, Threats and Means**
- Software Faults
  - Empirical Study: Linux
  - MISRA C/C++ and Safe Languages
  - Compartmentalization and Redundancy
  - Software Verification
- Hardware Faults
  - Coarse- and Fine-grained Redundant Multithreading
- Summary

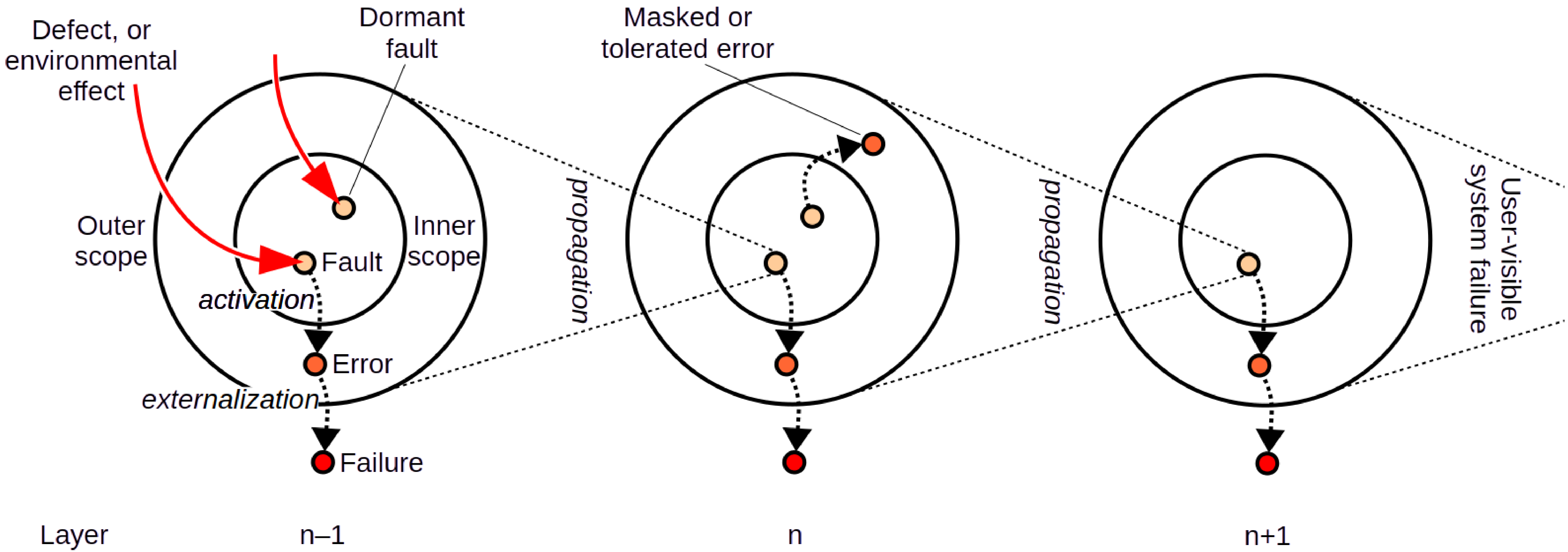
Dependability: Attributes

- **Availability**: readiness for correct service
- **Reliability**: continuity of correct service
- **Safety**: absence of catastrophic consequences (on the user(s) and the environment)
- **Integrity**: absence of improper system alterations
- **Maintainability**: ability to undergo modifications and repairs

Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. *Basic concepts and taxonomy of dependable and secure computing*. IEEE Transactions on Dependable and Secure Computing, 2004, 1. Jg., Nr. 1, S. 11-33.

Dependability: Threats

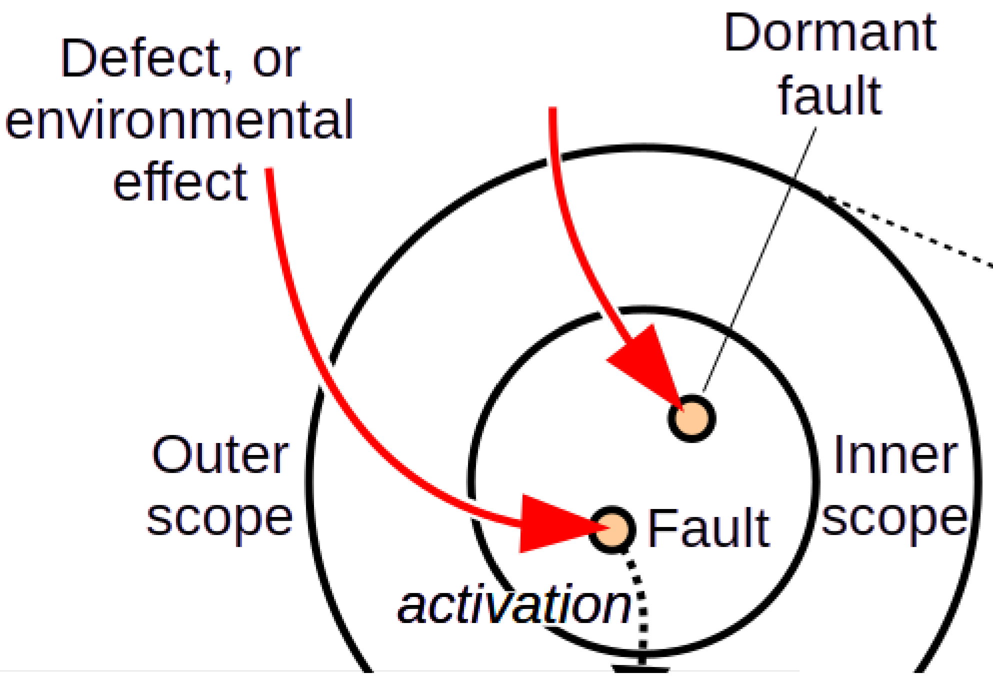
- Chain of dependability threats: **fault**, **error**, **failure**



H. Schirmeier. *Efficient Fault-Injection-based Assessment of Software-Implemented Hardware Fault Tolerance.* Dissertation, Technische Universität Dortmund, July 2016.

Dependability: Fault Categories

- **Software faults** (a.k.a. bugs)
  - Defects in design or implementation
  - Toolchain (e.g., compiler) bugs
- **Hardware faults**
  - transient: ***soft errors***
  - intermittent
  - permanent



Dependability: Means

- **Fault prevention** (or fault avoidance): preemptive measures
  - e.g. better shielding
- **Fault tolerance**: avoid service failures in the presence of faults
  - add redundancy, e.g. ECC memory, variable duplication, …
- **Fault removal**: reduces the number and severity of faults.
  - at development time (hardening system components) or runtime (replace faulty components)
- **Fault forecasting**: estimates the present number, the future incidence, and the expected consequences of faults.
  - e.g. using fault-injection (FI) experiments

Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. *Basic concepts and taxonomy of dependable and secure computing*. IEEE Transactions on Dependable and Secure Computing, 2004, 1. Jg., Nr. 1, S. 11-33.

Agenda

- Dependability: Attributes, Threats and Means
- **Software Faults**
  - Empirical Study: Linux
  - MISRA C/C++ and Safe Languages
  - Compartmentalization and Redundancy
  - Software Verification
- Hardware Faults
  - Coarse- and Fine-grained Redundant Multithreading
- Summary

Software Faults in Operating Systems: Linux

- 2001: Chou et al.'s classic study of software faults in Linux 1.0–2.4
- **Approach:**
  - Automated bug detection using static analysis (today: proprietary Coverity tool)
  - Target: several Linux-kernel versions (1.0–2.4)
- **Analysis:**
  - **Where** are the bugs?
  - What **bug types** do exist?
  - **How long** do they persist?
  - Do bugs **cluster** in certain locations?

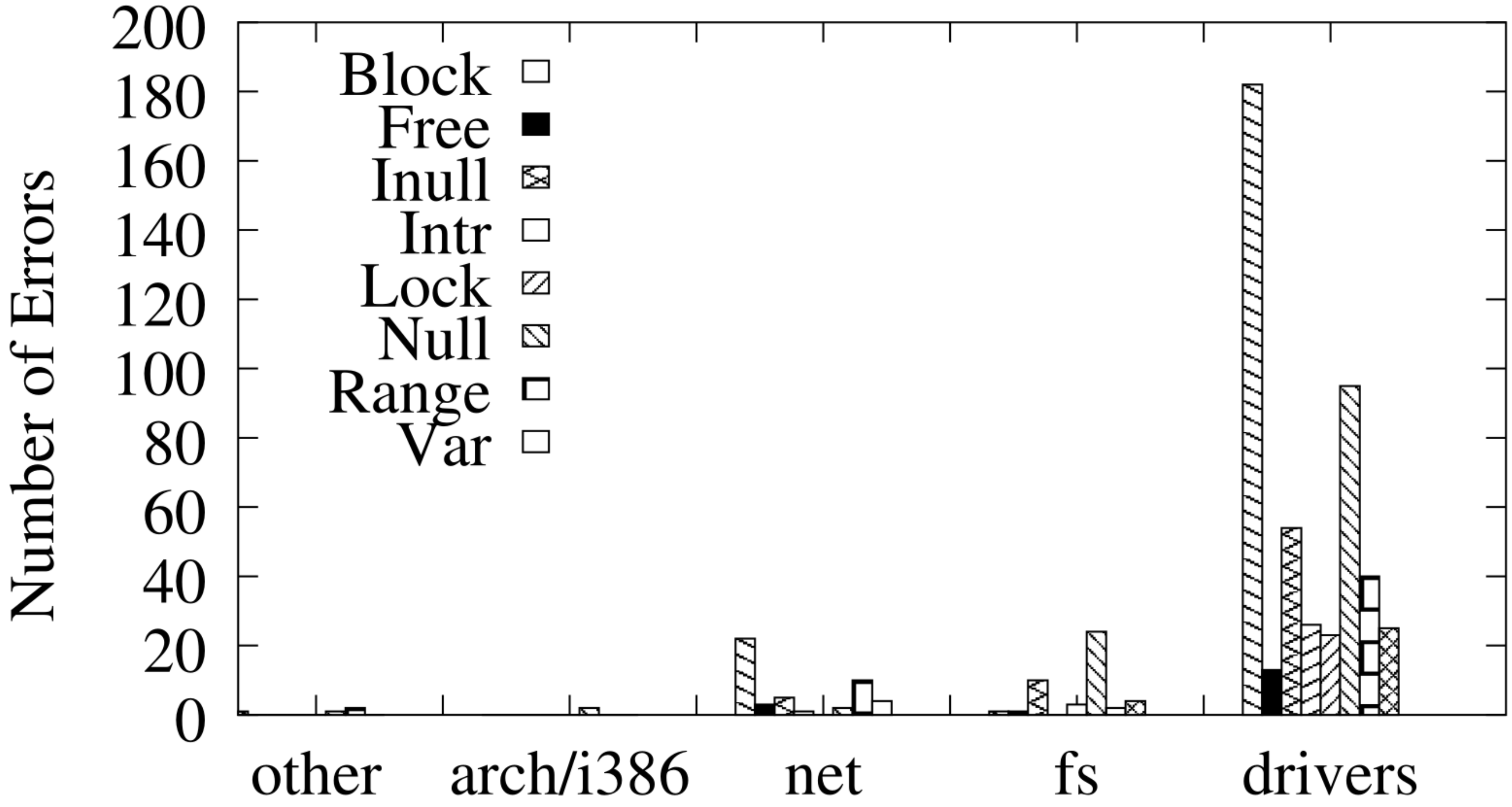
A. Chou, J. Yang, B. Chelf, S. Hallem, D. Engler. *An empirical study of operating systems errors.* In Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP), Oct. 2001, pp. 73-88.

Software Faults in Operating Systems: Linux

- 2011: Revalidation by N. Palix et al.
- **Approach:**
  - Target: newer Linux-kernel versions (2.6.0–2.6.33, 2003–2010)
- **Analysis:**
  - Impact of 10 years of code-quality improvement efforts?

Nicolas Palix, Gaël Thomas, Suman Saha, Christophe Calvès, Julia Lawall, and Gilles Muller. *Faults in Linux: Ten Years Later.* SIGPLAN Not. 46, 3 (March 2011), 305–318.

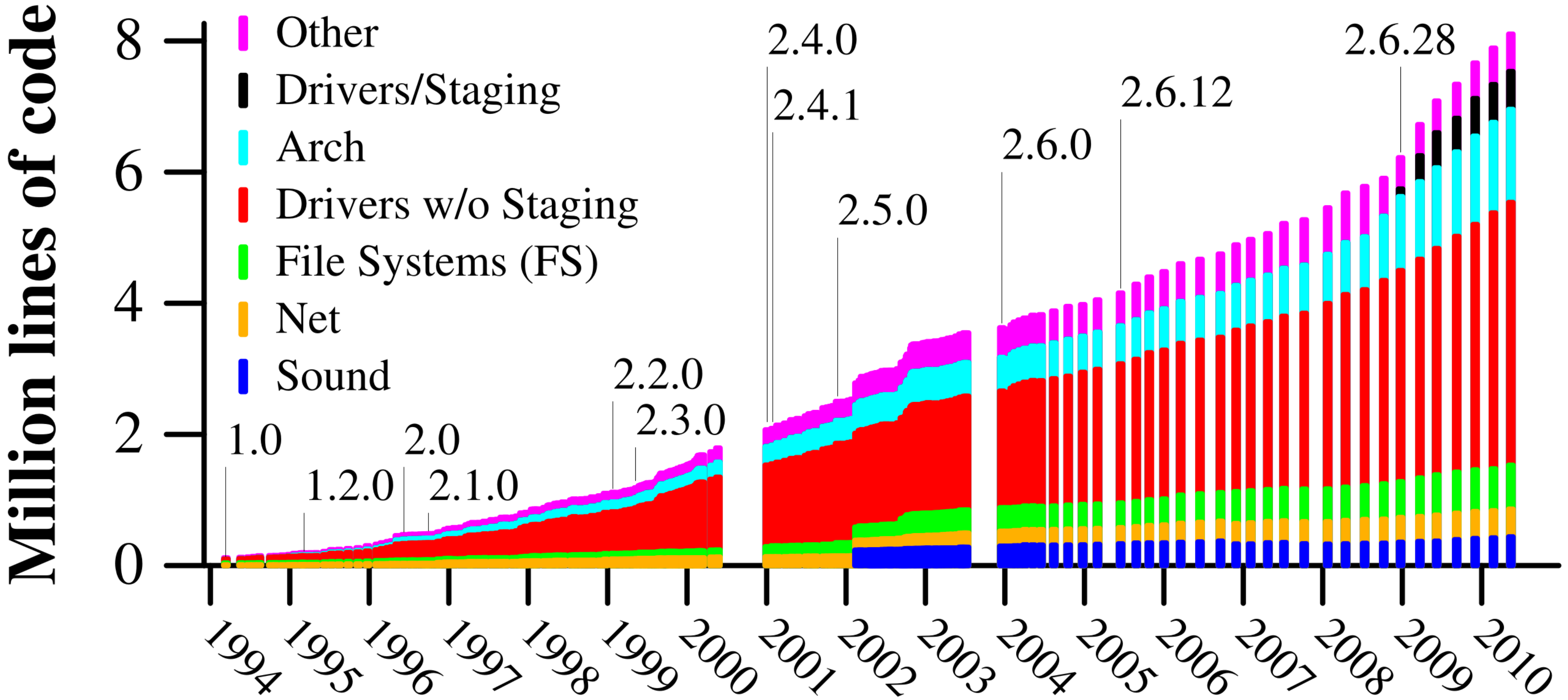
Linux: Faults per Subdirectory (Chou 2001)



Number of faults per directory in Linux

A. Chou, J. Yang, B. Chelf, S. Hallem, D. Engler. *An empirical study of operating systems errors.* In Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP), Oct. 2001, pp. 73-88.

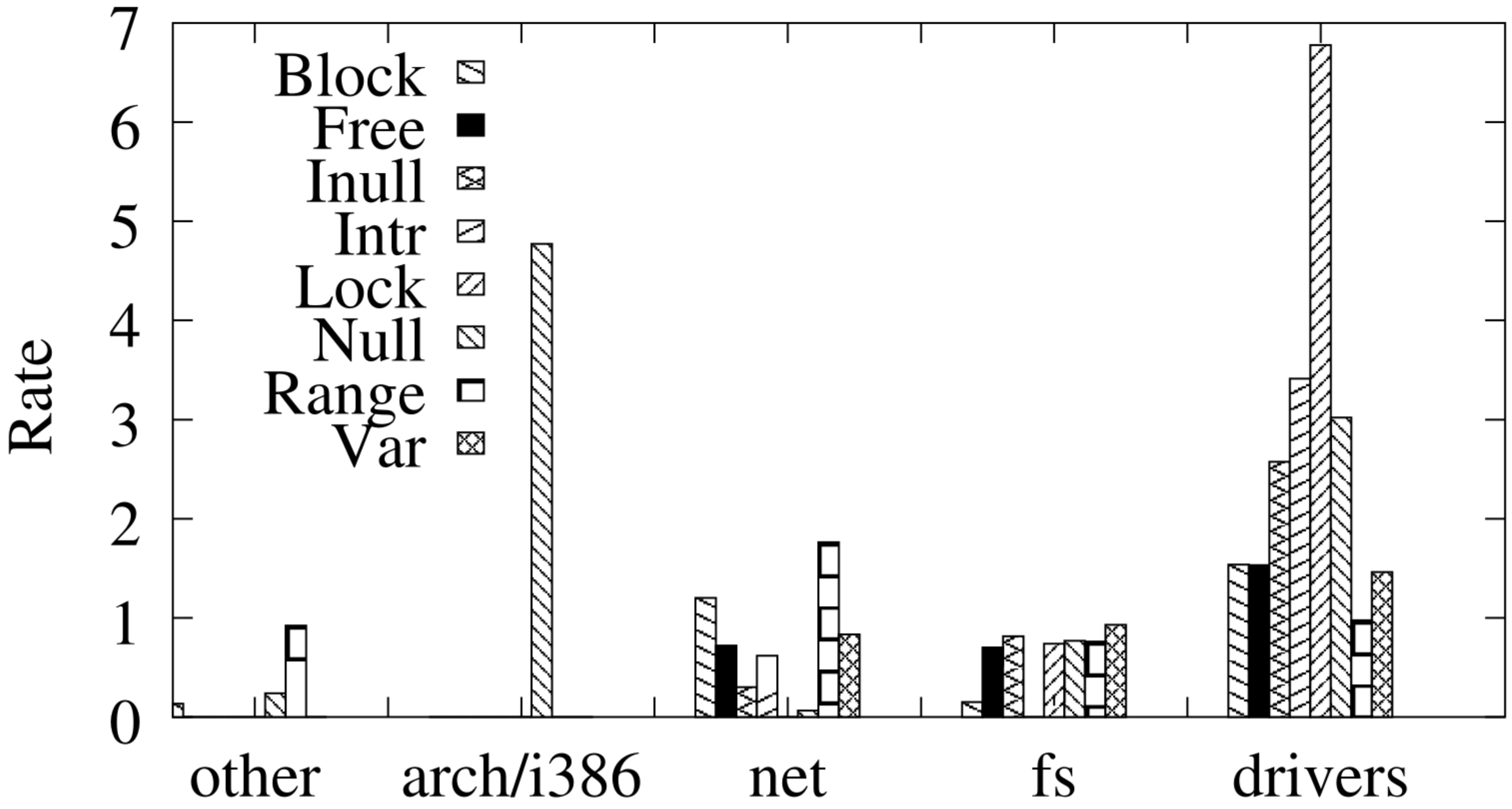
Linux: Lines of Code



Linux directory sizes (in MLOC)

Nicolas Palix, Gaël Thomas, Suman Saha, Christophe Calvès, Julia Lawall, and Gilles Muller. *Faults in Linux: Ten Years Later.* SIGPLAN Not. 46, 3 (March 2011), 305–318.

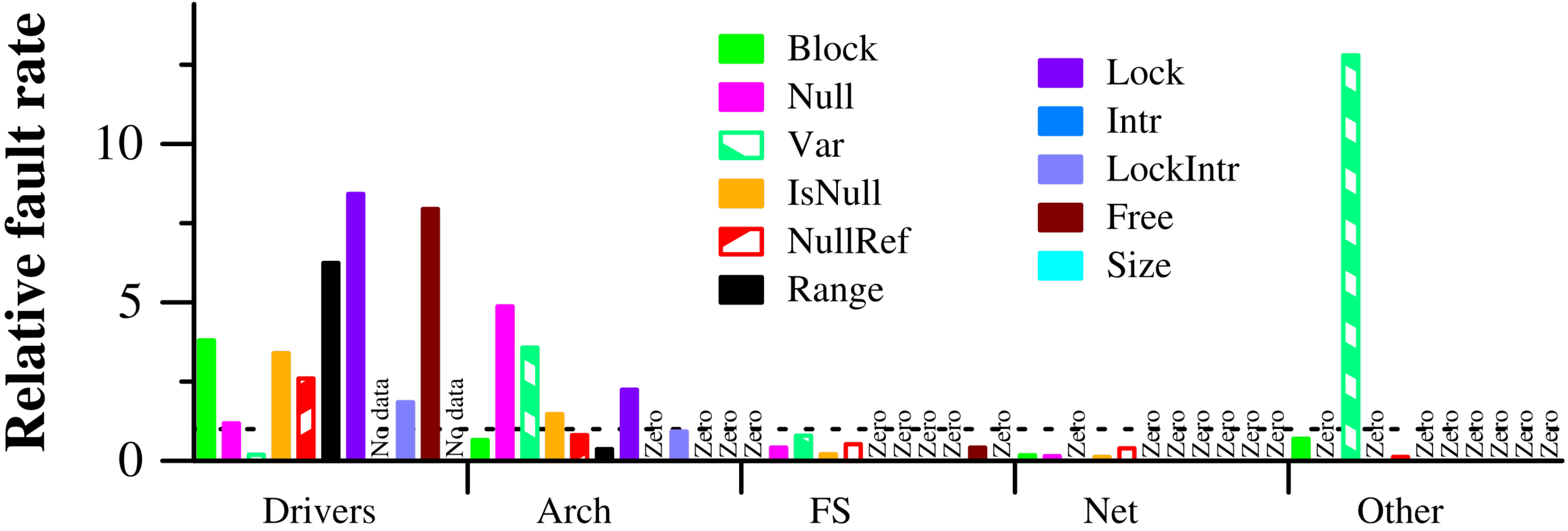
Linux: Fault Rate per Subdirectory (Chou 2001)



Per-directory rate of faults, normalized to average

A. Chou, J. Yang, B. Chelf, S. Hallem, D. Engler. *An empirical study of operating systems errors.* In Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP), Oct. 2001, pp. 73-88.

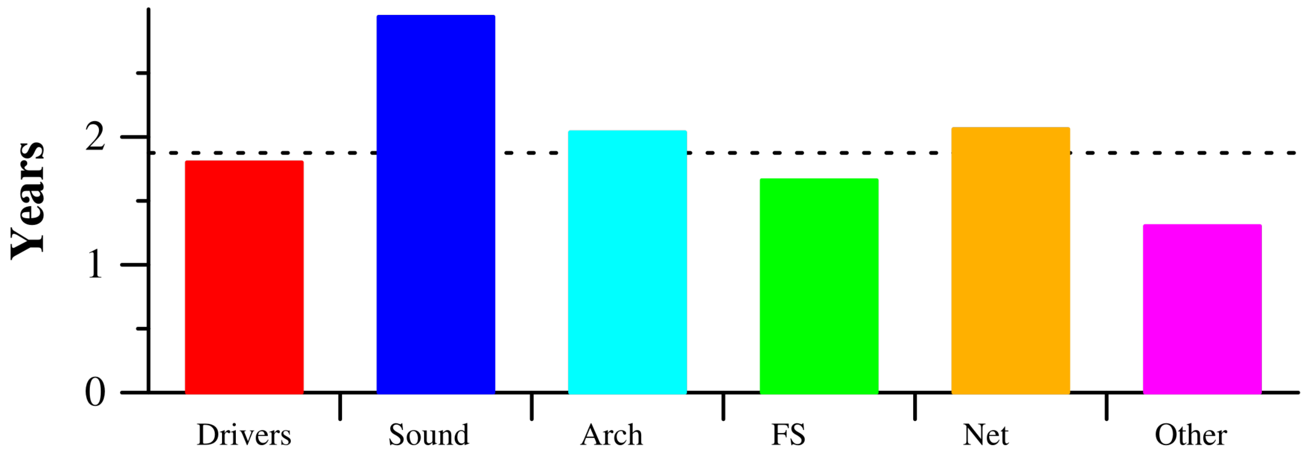
Linux: Fault Rate per Subdirectory (Palix 2011)



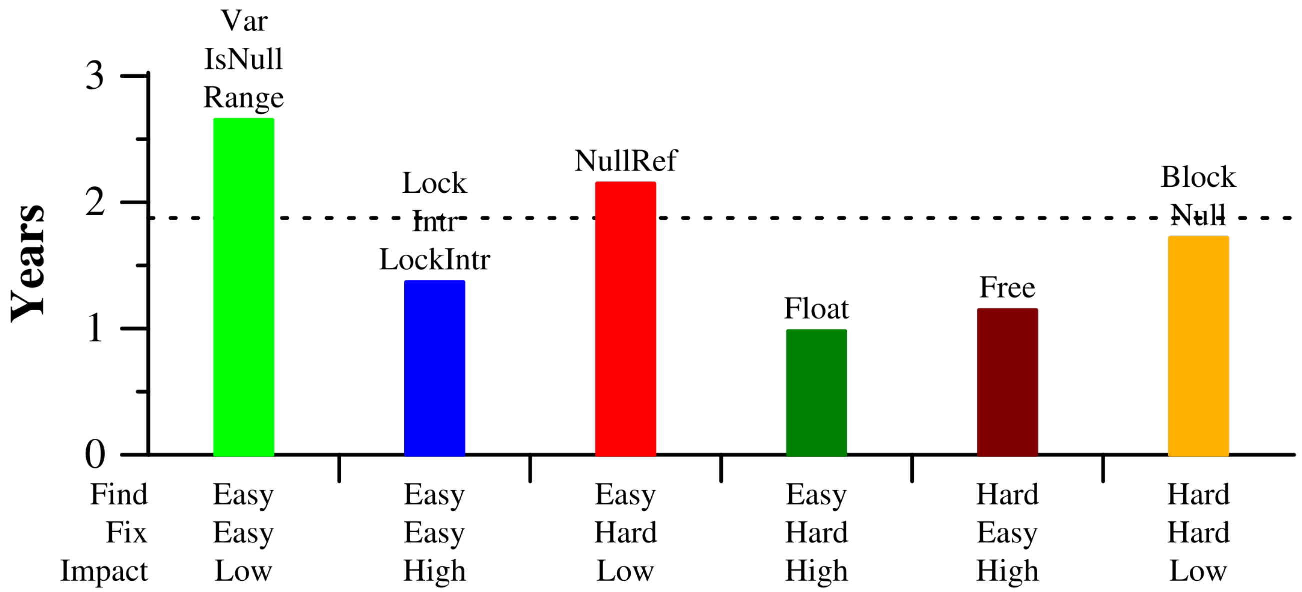
Per-directory rate of faults, normalized to average

Nicolas Palix, Gaël Thomas, Suman Saha, Christophe Calvès, Julia Lawall, and Gilles Muller. *Faults in Linux: Ten Years Later.* SIGPLAN Not. 46, 3 (March 2011), 305–318.

Linux: Bug Lifetimes (Palix 2011)



… per directory



… per finding and fixing difficulty, and impact likelihood

Nicolas Palix, Gaël Thomas, Suman Saha, Christophe Calvès, Julia Lawall, and Gilles Muller. *Faults in Linux: Ten Years Later.* SIGPLAN Not. 46, 3 (March 2011), 305–318.

Means: Software Engineering

- Quality Assurance, e.g. manual testing, automated testing, fuzzing
- Continuous Integration
- Static analysis
- Using safer languages
- Guidelines, best practices, etc.
  - Examples: MISRA C++, C++ Guideline Support Library

Example: MISRA C++

- **Rule 0-1-7**
  - *The value returned by a function having a non-void return type that is not an overloaded operator shall always be used.*

- **Rule 3-9-3**
  - *The underlying bit representations of floating-point values shall not be used.*

- **Rule 6-4-6**
  - *The final clause of a switch statement shall be the default-clause.*

MISRA C++: Rule 3-4-1

- **(Required)** *An identifier declared to be an object or type shall be defined in a block that minimizes its visibility.*

Rationale

Defining variables in the minimum block scope possible reduces the visibility of those variables and therefore reduces the possibility that these identifiers will be used accidentally. A corollary of this is that global objects (including singleton function objects) shall be used in more than one function.

MISRA C++: Rule 3-4-1 – Example

```
void f(int32_t k)
{
    int32_t j = k * k; // Non-compliant
    if (k > 8) {
        int32_t i = k; // Compliant
        std::cout << i << j << std::endl;
    }
}
```

- Definition of `j` should be moved into the inner block

  → Reduce the chance to incorrectly use `j` later within `f( )`

MISRA C++: Rule 8-18-2

- *The result of an assignment operator should not be used.*

```
if ((x = y) == 0) {  // Non-compliant
    // …
}
```

```
x = y;
if (y == 0) {  // Compliant
    // …
}
```

Means: Safe(r) Programming Languages

- Garbage collection (Go)
- Memory safety (Rust)
- No unused variables (Go, Rust)
- Check error return codes (Go, Rust)
- No uninitialized memory (Go, Rust)
- …

Biscuit: A Monolithic Kernel written in Go

- **High-level features:** closures, channels, garbage collection
- Development effort: 28k lines in Go and 1.5k lines in assembly
- Implemented drivers: AHCI SATA disk controllers and Intel 82599-based Ethernet controllers
- Out of 64 **CVE-listed Linux kernel bugs**, ≈40 would be alleviated by Go
- 5–15% slower, up to 600μs latencies for GC

Cody Cutler, M. Frans Kaashoek, and Robert T. Morris. *The benefits and costs of writing a POSIX kernel in a high-level language.* In: OSDI. Oct. 2018.

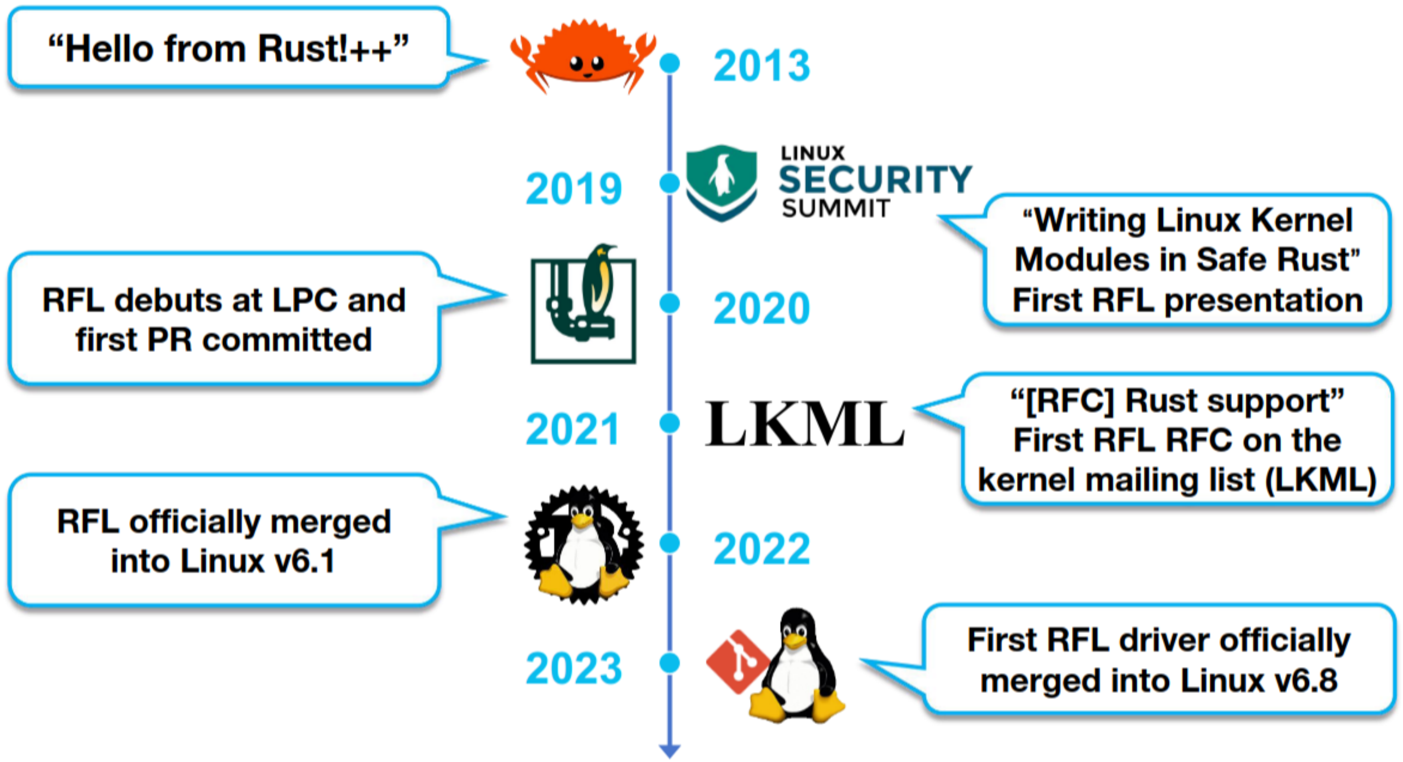
Tock: An Embedded OS implemented in Rust

- **Compiler-enforced rules:**
  - Several immutable XOR one mutable reference
  - No null pointers
  - No reading undefined memory
  - etc.
- Unsafe code is annotated
- Memory or synchronization problems are impossible in safe code
- Performance like in C or C++ code
- Some software patterns don’t work well with (safe) Rust

Amit Levy et al. ‘Multiprogramming a 64kb computer safely and efficiently.’ In: SOSP. 2017.

Rust for Linux

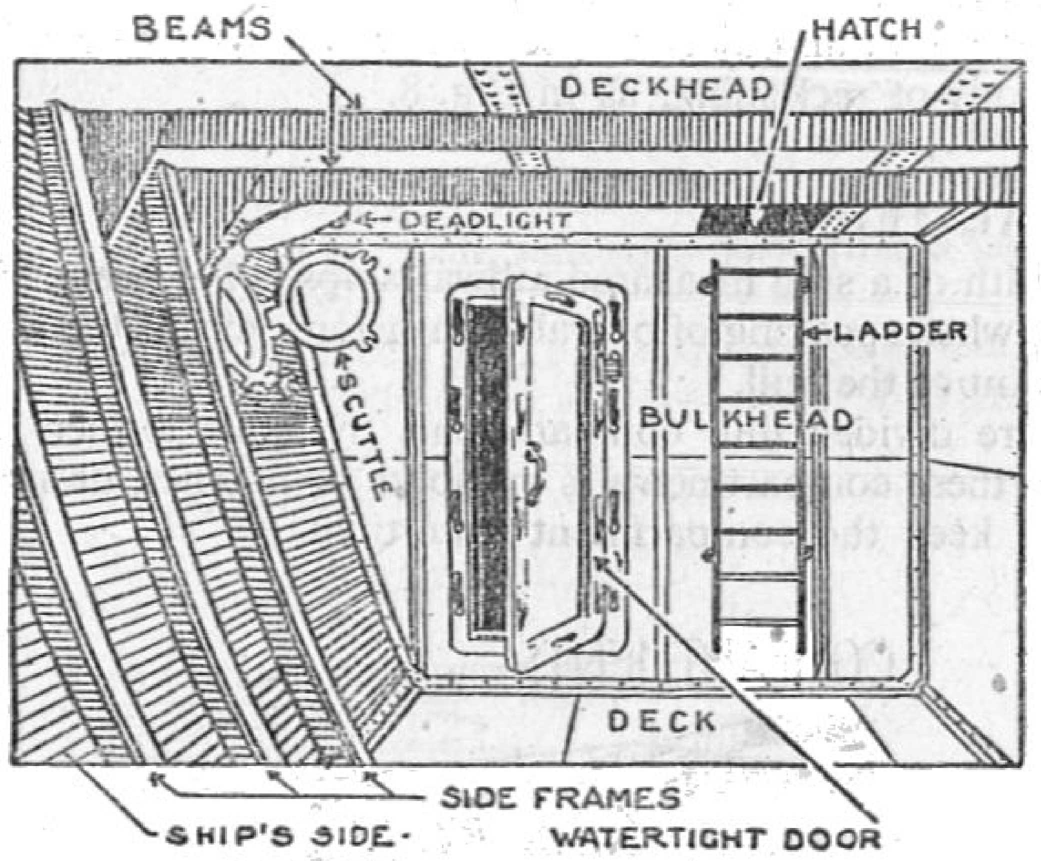
- Linux: Historically implemented in C and assembler
- **Rust for Linux** project (since 2020): Add Rust as a programming language
  - 2023: first driver accepted
  - Since then, more drivers + FS implementations



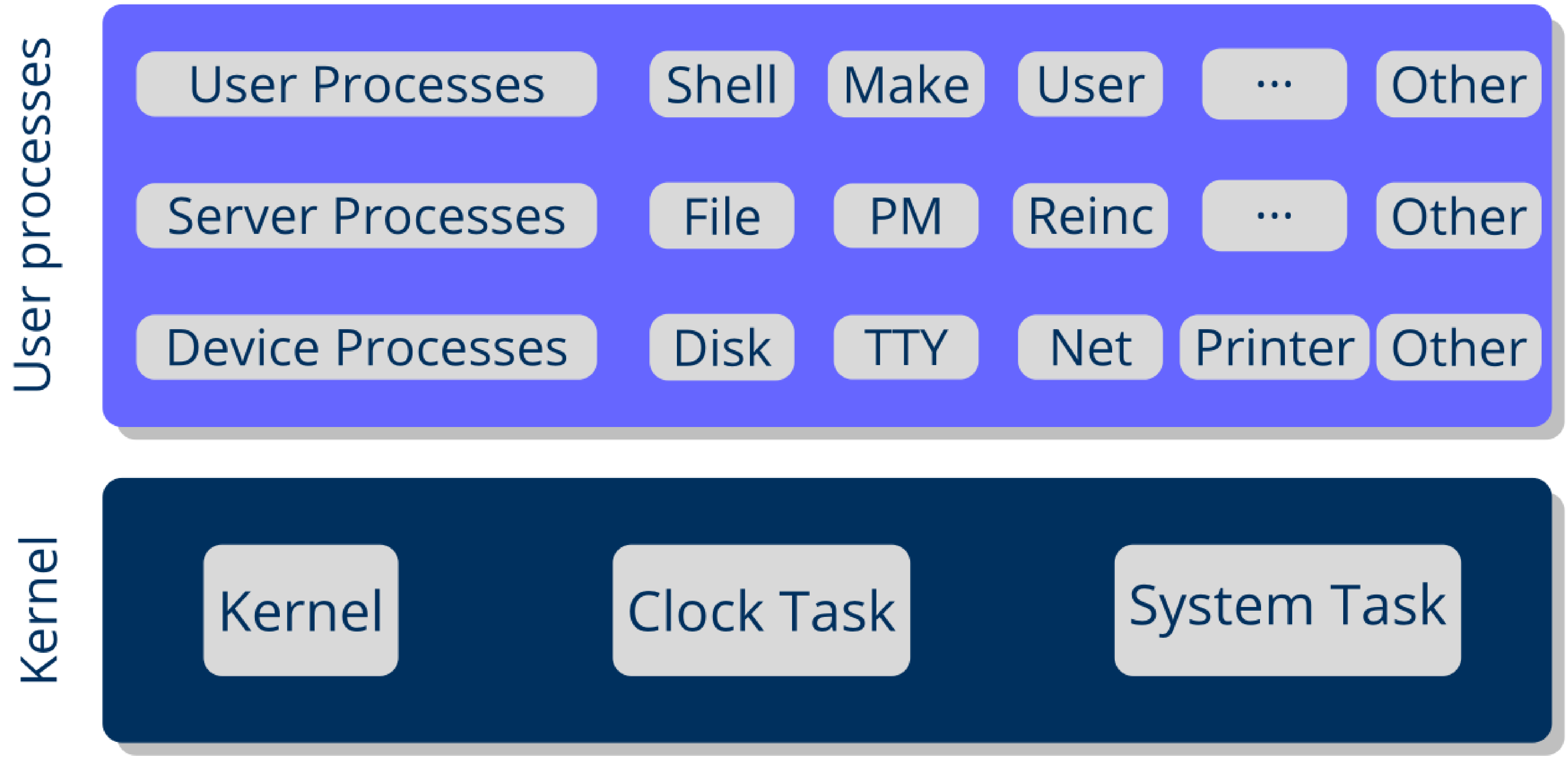
H. Li, L. Guo, Y. Yang, S. Wang, and M. Xu. *An Empirical Study of Rust-for-Linux: The Success, Dissatisfaction, and Compromise.* In USENIX Annual Technical Conference (ATC), 2024 (pp. 425-443).

Means: Software Architecture

- **Means:**
  - **Compartmentalization**
  - **Redundancy**
  - **Hardening**
- Address hardware faults
- Recovery:
  - **Rollback**: return to a previous state
    - Transactions
    - Checkpoint/Restart
  - **Roll-forward**: everything else
    - Error correcting codes
    - Triple modular redundancy + majority voting



MINIX 3: A Fault-tolerant OS



MINIX 3: Fault Tolerance

- Address Space Isolation
  - Applications only access private memory
  - Faults do not spread to other components
- User-level OS services
  - Principle of Least Privilege
  - Fine-grained control over resource access (e.g., DMA only for specific drivers)
- Small components
  - Easy to replace (“micro-reboot”)

Jorrit N Herder et al. *Fault isolation for device drivers.* In: DSN. 2009, pp. 33–42.

MINIX 3: Fault Detection

- **Fault model:** transient errors caused by software bugs
  - Fix: Component restart
- **Reincarnation server** monitors components
  - Program termination (crash)
  - CPU exception (e.g., division by zero)
  - Heartbeat messages
- Users may also indicate that something is wrong

Jorrit N Herder et al. *Fault isolation for device drivers.* In: DSN. 2009, pp. 33–42.

MINIX 3: Repair

- Restarting a component is insufficient:
  - Applications may **depend** on restarted component
  - After restart, **component state** is lost
- MINIX 3: explicit mechanisms
  - Reincarnation server signals applications about restart
  - Applications store state at data-store server
  - In any case: program interaction needed
    - Restarted app: store/recover state
    - User apps: recover server connection

Jorrit N Herder et al. *Fault isolation for device drivers.* In: DSN. 2009, pp. 33–42.

L4ReAnimator: Restart on L4Re

- L4Re Applications
  - Loader component: ned
  - Detects application termination: parent signal
  - Restart: re-execute Lua init script (or parts of it)
  - Problem after restart: capabilities
    - No single component knows everyone owning a capability to an object
    - MINIX 3 store/recover-state signals won't work

Dirk Vogt, Björn Döbel, and Adam Lackorzynski. *Stay strong, stay safe: Enhancing reliability of a secure operating system.* In: Workshop on Isolation and Integration for Dependable Systems. 2010, pp. 1–10.

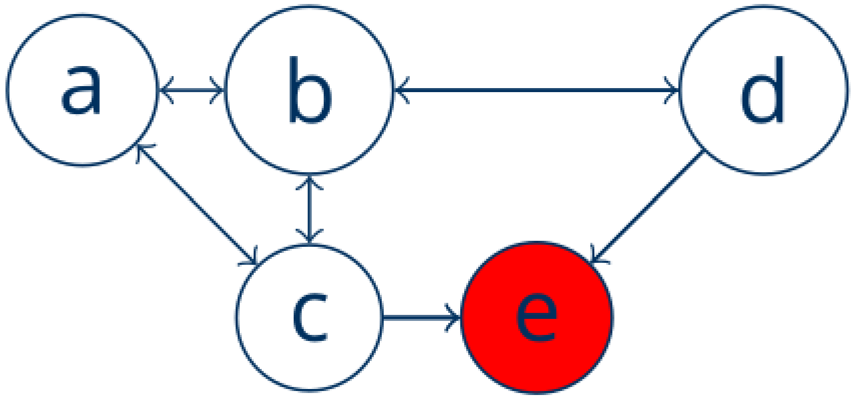
L4ReAnimator: Lazy recovery

- Only the application itself can detect that a capability vanished
  - Kernel raises ***Capability fault***
- Application must re-obtain the capability:
  - Execute app-specific ***capability fault handler***
  - Create new communication channel
  - Restore session state
- Programming model:
  - Capfault handler **provided by server implementer**
  - Handling transparent for application developer
  - Semi-transparency

Dirk Vogt, Björn Döbel, and Adam Lackorzynski. *Stay strong, stay safe: Enhancing reliability of a secure operating system.* In: Workshop on Isolation and Integration for Dependable Systems. 2010, pp. 1–10.

Means: Software Verification

- Combines software engineering and software architectures
- Define good and bad states
- Define axioms (e.g. initial state is good)
- Prove bad states (e.g. null-pointer dereference) are unreachable
- Special theorem-prover languages



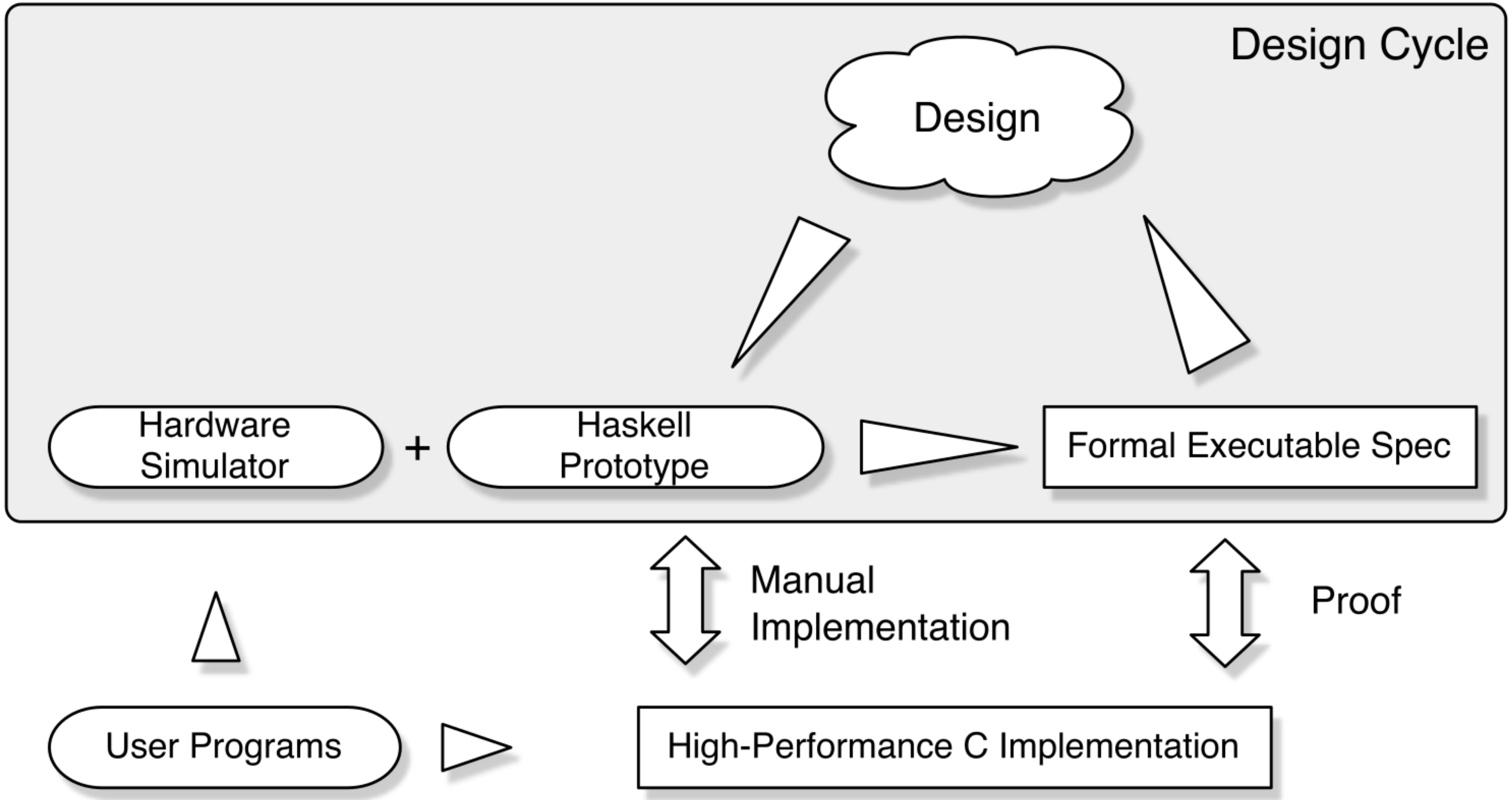
seL4: Formal verification of an OS kernel

- seL4: https://sel4.systems/
- Formally verify that system adheres to specification
- Microkernel design allows to separate components easier

  → Verification process becomes easier

Gerwin Klein et al. *seL4: Formal verification of an OS kernel.* In: SOSP. 2009, pp. 207–220.

seL4: Formal verification of an OS kernel



Gerwin Klein et al. *seL4: Formal verification of an OS kernel.* In: SOSP. 2009, pp. 207–220.

seL4: Summary

- Assumes correctness of compiler, assembly code, and hardware
- DMA over IOMMU
- Architectures: arm, x86
- Virtualization
- Future: Verification on multicores

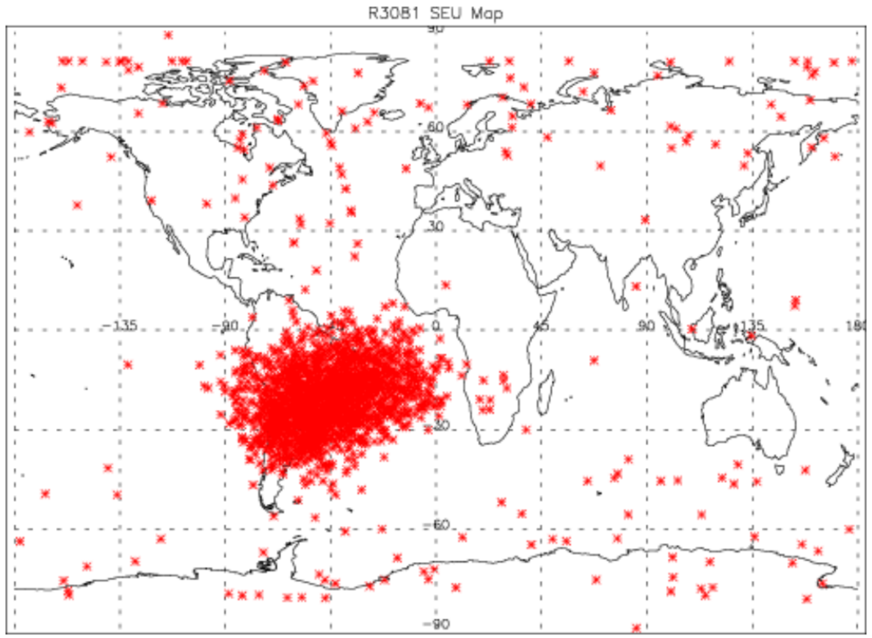
Gerwin Klein et al. *seL4: Formal verification of an OS kernel.* In: SOSP. 2009, pp. 207–220.

Agenda

- Dependability: Attributes, Threats and Means
- Software Faults
  - Empirical Study: Linux
  - MISRA C/C++ and Safe Languages
  - Compartmentalization and Redundancy
  - Software Verification
- **Hardware Faults**
  - Coarse- and Fine-grained Redundant Multithreading
- Summary

Transient Hardware Faults

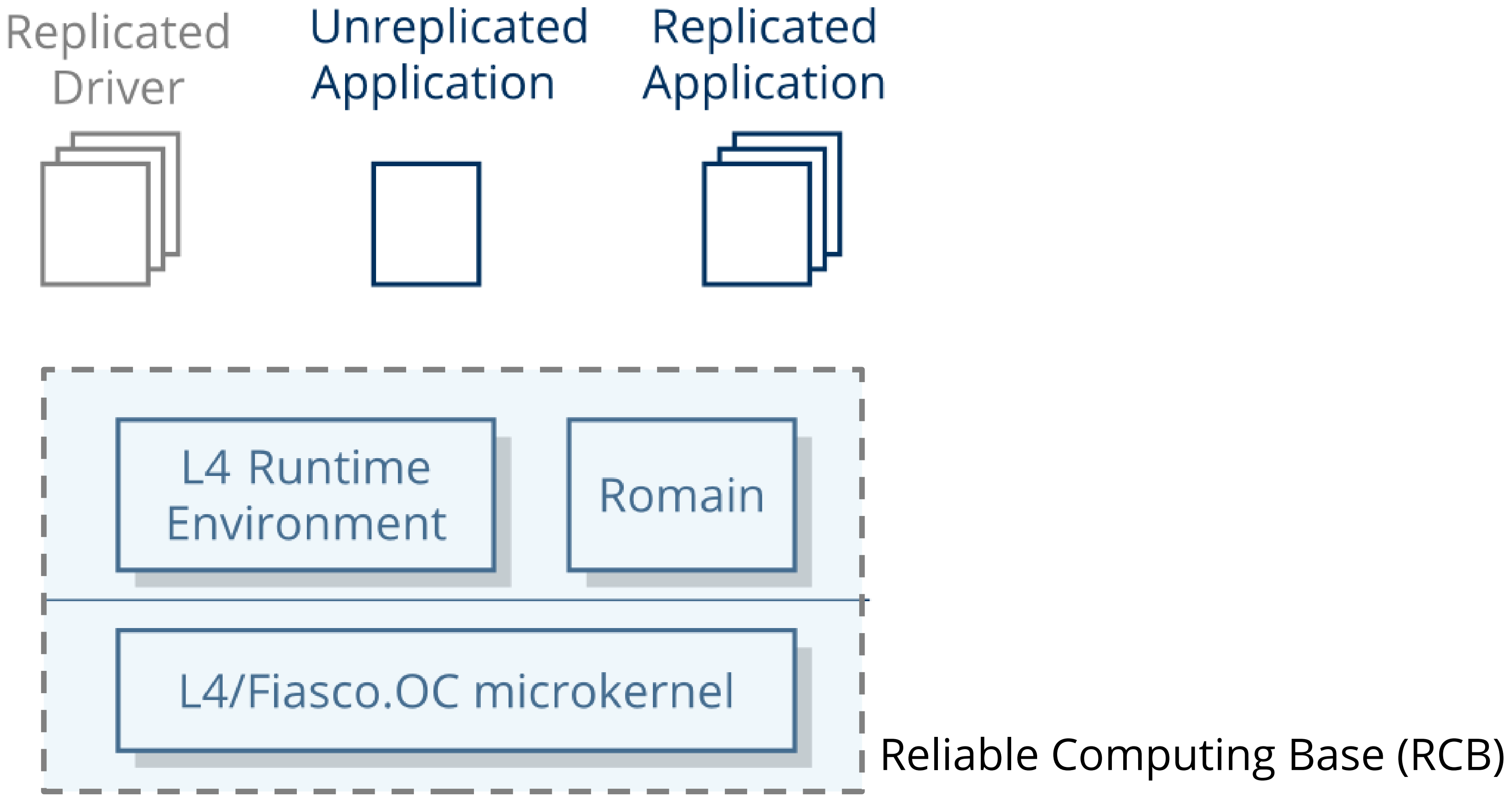
- Radiation-induced soft errors
  - Mainly an issue in avionics+space?
- DRAM errors in large data centers
  - Google: >2% failing DRAM DIMMs per year [Schroeder2009]
  - ECC insufficient [Hwang2012]
- Decreasing transistor sizes → higher fault rate in CPU functional units [Dixit2011]



[Lovellette2002]

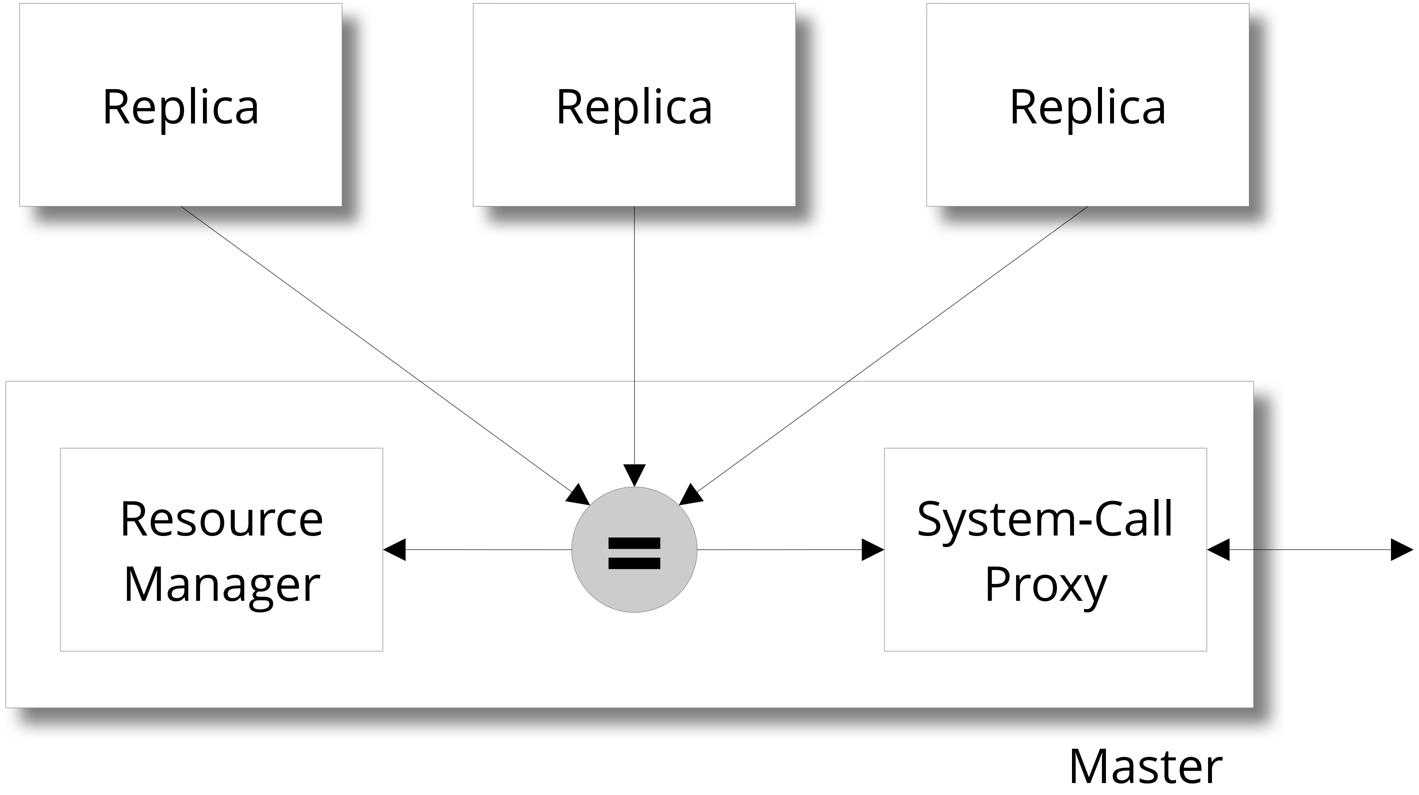
[Schroeder2009] Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. *DRAM errors in the wild: a large-scale field study.* In: SIGMETRICS/Performance. 2009, pp. 193–204.
[Hwang2012] Andy A Hwang, Ioan A Stefanovici, and Bianca Schroeder. *Cosmic rays don't strike twice.* In: ASPLOS. 2012, pp. 111–122.
[Dixit2011] Anand Dixit and Alan Wood. *The impact of new technology on soft error rates.* In: International Reliability Physics Symposium. 2011, 5B–4.
[Lovellette2002] Michael N. Lovellette, K. S. Wood, D. L. Wood, Jim H. Beall, Philip P. Shirvani,Namsuk Oh, and Edward J. McCluskey. Strategies for fault-tolerant, space-based computing: Lessons learned from the ARGOS testbed. In Proceedings of the 2002 IEEE Aerospace Conference, pages 5–2109–5–2119. IEEE Computer Society Press, 2002.

Romain: Transparent Replication as OS Service



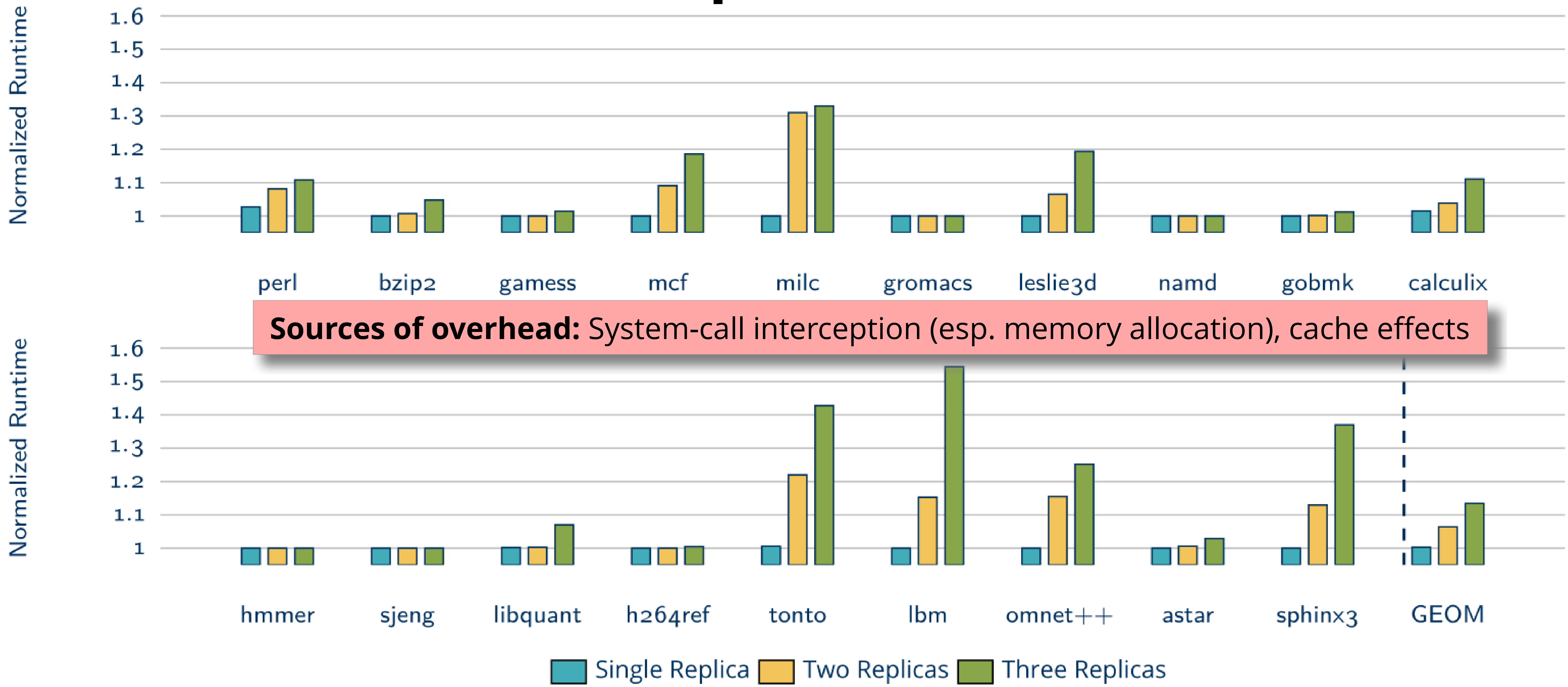
Björn Döbel and Hermann Härtig. *Can we put concurrency back into redundant multithreading?* In: EMSOFT. 2014, pp. 1–10.
Björn Döbel, Hermann Härtig, and Michael Engel. *Operating system support for redundant multithreading.* In: EMSOFT. 2012, pp. 83–92.

Romain: Structure



Björn Döbel and Hermann Härtig. *Can we put concurrency back into redundant multithreading?* In: EMSOFT. 2014, pp. 1–10.
Björn Döbel, Hermann Härtig, and Michael Engel. *Operating system support for redundant multithreading.* In: EMSOFT. 2012, pp. 83–92.

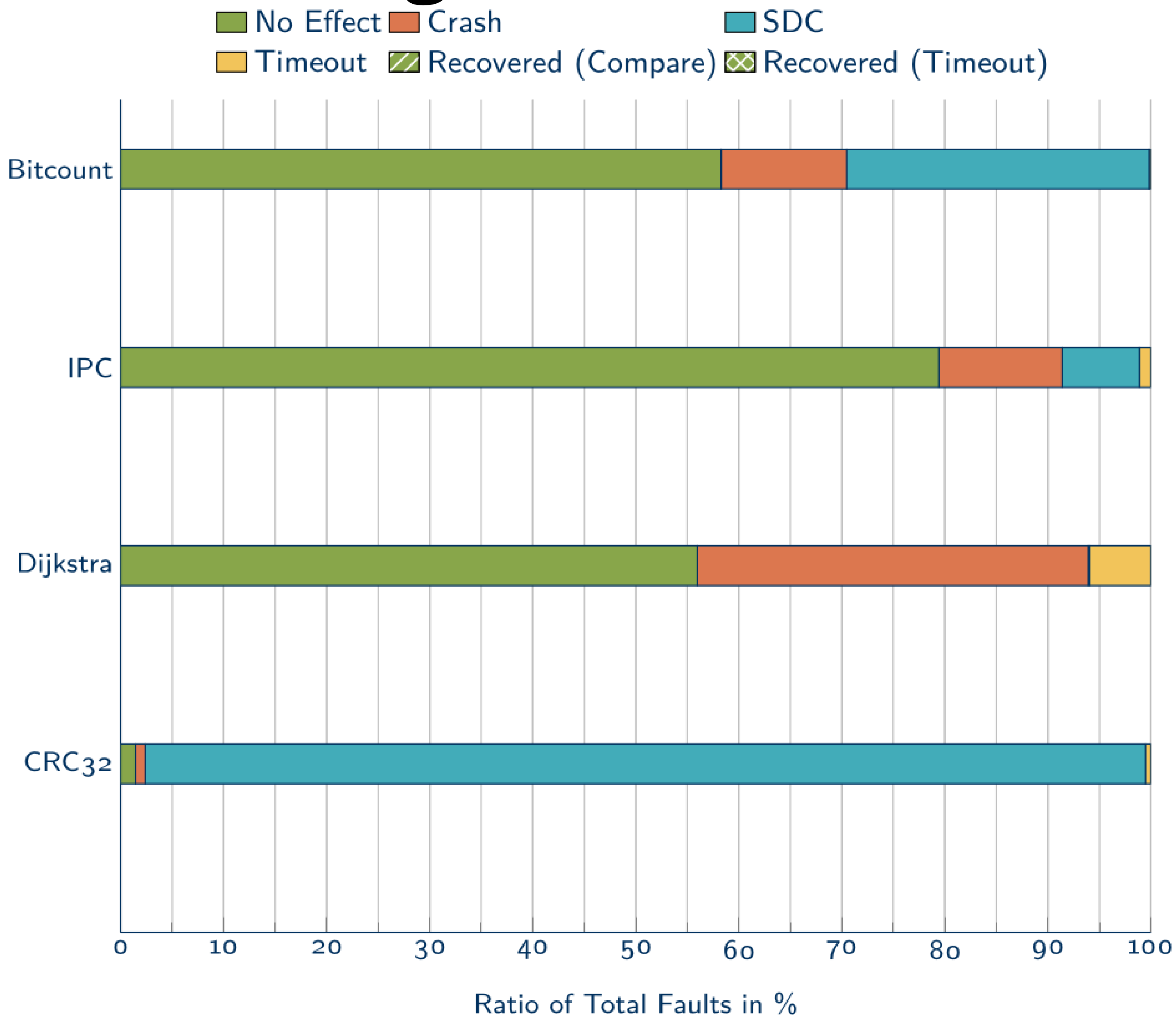
Romain: Performance (replicated SPEC CPU 2006)



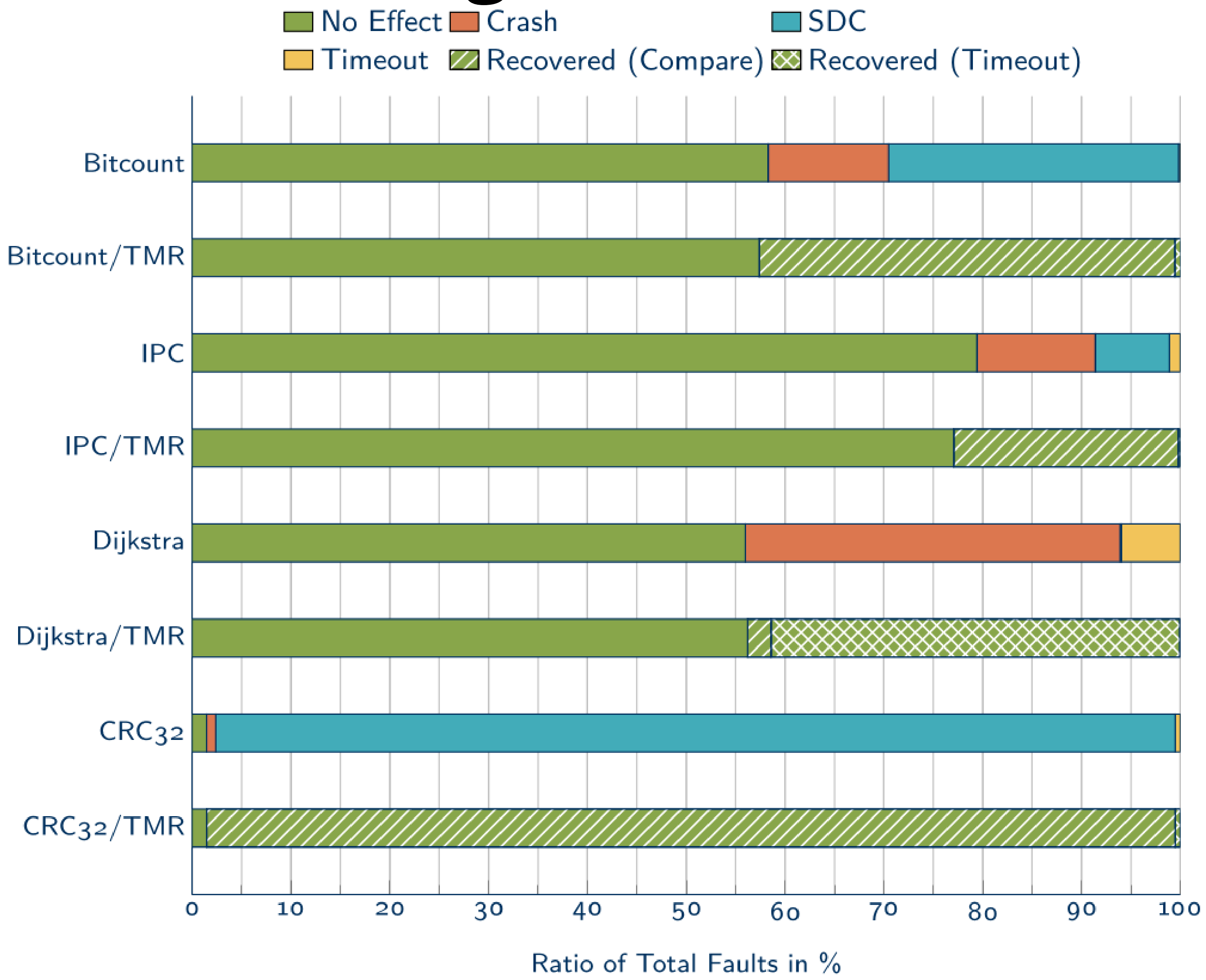
Sources of overhead: System-call interception (esp. memory allocation), cache effects

Björn Döbel. *Operating System Support for Redundant Multithreading.* Dissertation. TU Dresden, 2014.

Romain: Error Coverage



Romain: Error Coverage



Romain: Summary

- Faults: CPU and memory bit-flips
- Best-effort resilience
- Triple modular redundancy (TMR) with small increase in makespan
- Multithreading support with deterministic multithreading

HAFT: Hardware-Assisted Fault Tolerance

- Fault model: CPU single-event upsets (SEU)
- Instruction-level redundancy for fault detection
- Hardware transaction memory for fault recovery
- *Best-effort* fault tolerance
- Improve efficiency through instruction-level parallelism (ILP) and compiler optimizations

Dmitrii Kuvaiskii et al. *HAFT: hardware-assisted fault tolerance.* In: Proceedings of the Eleventh European Conference on Computer Systems (EuroSys '16). London, United Kingdom: ACM, Apr. 18, 2016, pp. 1–17.

HAFT: Hardware-Assisted Fault Tolerance

(a) Native

```
z = add x, y





ret z
```

Native

(b) ILR

```
z = add x, y
z2 = add x2, y2
d = cmp neq z, z2
br d, crash

ret z
```

DMR

(b) ILR

```
loop:
  r1 = add r1, r2
  r1' = add r1', r2'
  r1'' = add r1'', r2''
  majority(r1, r1', r1'')
  majority(r3, r3', r3'')
  cmp r1, r3

  jne loop
```

TMR

(c) HAFT

```
xbegin
  z = add x, y
  z2 = add x2, y2
  d = cmp neq z, z2
  br d, xabort
xend
ret z
```

HAFT

Dmitrii Kuvaiskii et al. *HAFT: hardware-assisted fault tolerance.* In: Proceedings of the Eleventh European Conference on Computer Systems (EuroSys '16). London, United Kingdom: ACM, Apr. 18, 2016, pp. 1–17.

HAFT: Performance

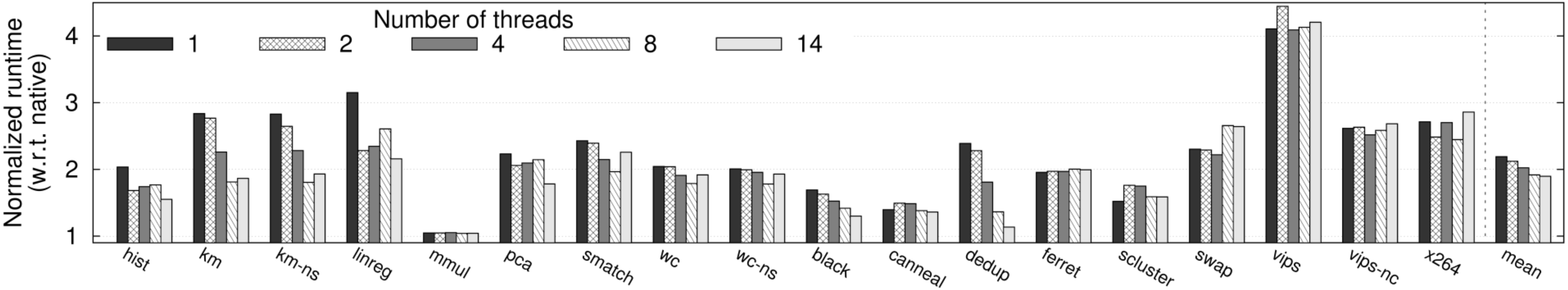


Figure 6: Performance overhead over native execution with the increasing number of threads (on a machine with 14 cores).

Dmitrii Kuvaiskii et al. *HAFT: hardware-assisted fault tolerance.* In: Proceedings of the Eleventh European Conference on Computer Systems (EuroSys ’16). London, United Kingdom: ACM, Apr. 18, 2016, pp. 1–17.

Comparison: Romain vs. HAFT

	Romain	HAFT
Granularity	Syscall	Instruction
Parallelism	Thread-level	Instruction-level
Runtime overhead	~10%	~100%
Resource overhead	~210%	~100%
Fault model	CPU & (some) memory	CPU
Implementation	OS	Compiler & CPU Features

Agenda

- Dependability: Attributes, Threats and Means
- Software Faults
  - Empirical Study: Linux
  - MISRA C/C++ and Safe Languages
  - Compartmentalization and Redundancy
  - Software Verification
- Hardware Faults
  - Coarse- and Fine-grained Redundant Multithreading
- **Summary**

Summary

- **Dependability:** robust development practices + reliability techniques
- Do not let failures propagate
- Prevent the worst-case failure mode: silent data corruptions (SDC)
- Fail fast!