TECHNISCHE UNIVERSITÄT DRESDEN

OPERATING SYSTEM SUPPORT FOR REDUNDANT MULTITHREADING

Björn Döbel (TU Dresden)

Brussels, 02.02.2013

Hardware Faults

- Radiation-induced soft errors
  - Mainly an issue in avionics+space[1]
- DRAM errors in large data centers
  - Google Study: $> 2\%$ failing DRAM DIMMs per year[2]
  - ECC is not going to even detect a significant amount[3]
  - Disk failure rate about 5%[4]
- Furthermore: decreasing transistor sizes, higher rate of transient errors in CPU functional units[5]

[1] Shirvani, McCluskey: *Fault-Tolerant Systems in A Space Environment: The CRC ARGOS Project*, 1998

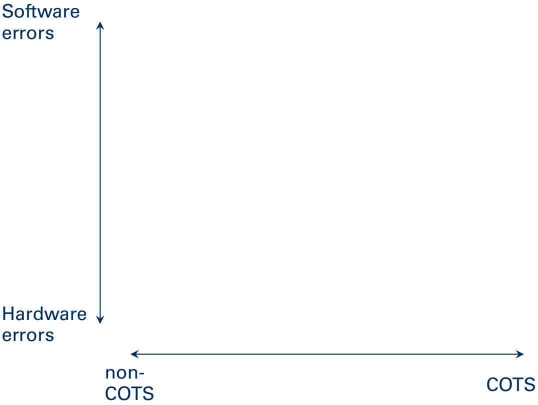
[2] Schroeder, Pinheiro, Weber: *DRAM Errors in the Wild: A Large-Scale Field Study*, SIGMETRICS 2009

[3] Hwang, Stefanovici, Schroeder: *Cosmic Rays Don't Strike Twice: Understanding the Nature of DRAM Errors and the Implications for System Design*, ASPLOS 2012

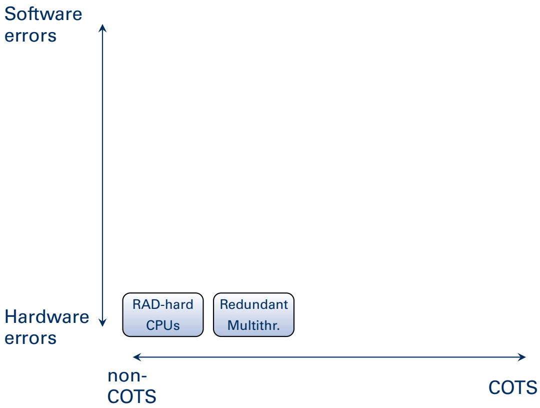
[4] Pinheiro, Weber, Barroso: *Failure Trends in a Large Disk Drive Population*, FAST 2007

[5] Shivakumar, Kistler, Keckler: *Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic*, DSN 2002

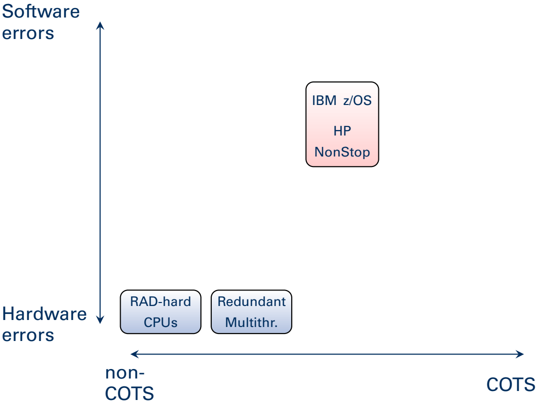
Fault Tolerance: State of the Union



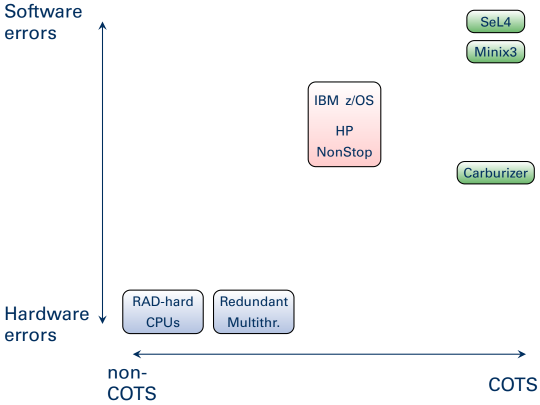
Fault Tolerance: State of the Union



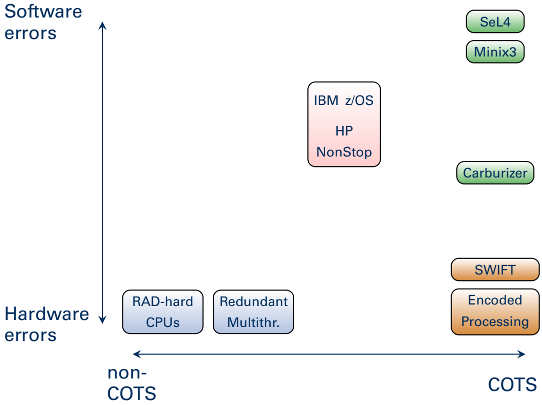
Fault Tolerance: State of the Union



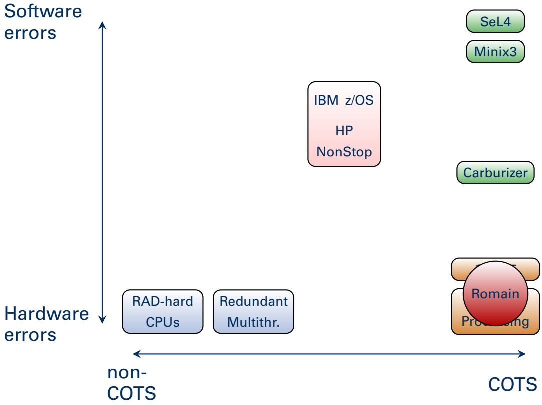
Fault Tolerance: State of the Union



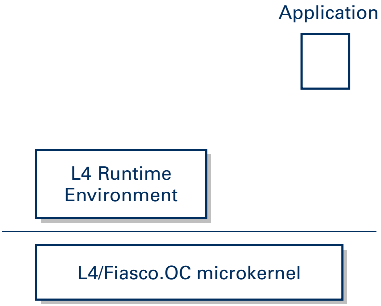
Fault Tolerance: State of the Union



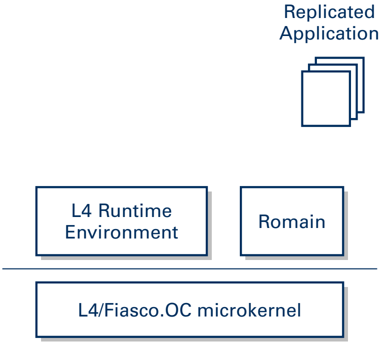
Fault Tolerance: State of the Union



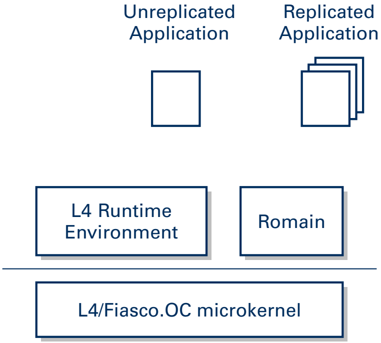
Transparent Replication as OS Service



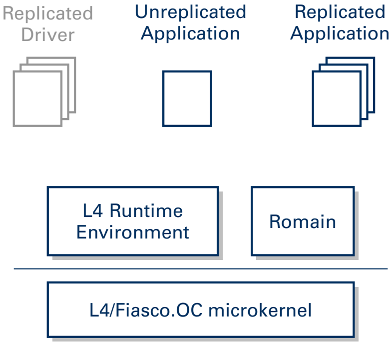
Transparent Replication as OS Service



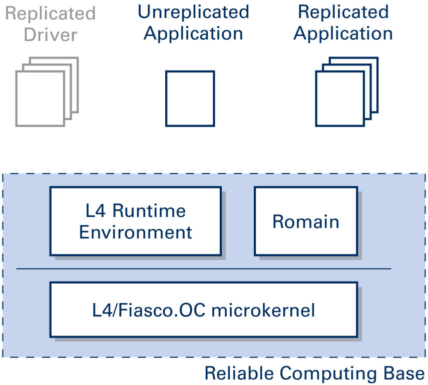
Transparent Replication as OS Service



Transparent Replication as OS Service



Transparent Replication as OS Service



Process-Level Redundancy[6]

Binary recompilation

- Complex, unprotected compiler
- Architecture-dependent

System calls for replica synchronization

Virtual memory fault isolation

- Restricted to Linux user-level programs

[6] Shye, Blomsted, Moseley, Reddi, Connors: *PLR: A software approach to transient fault tolerance for multicore architectures*, DSN 2009

Process-Level Redundancy[6]

Binary recompilation

- Complex, unprotected compiler
- Architecture-dependent

Reuse OS mechanisms

System calls for replica synchronization

Additional synchronization events

Virtual memory fault isolation

- Restricted to Linux user-level programs

Microkernel-based

[6] Shye, Blomsted, Moseley, Reddi, Connors: *PLR: A software approach to transient fault tolerance for multicore architectures*, DSN 2009

Why A Microkernel?

- Small components
  - Microrebootable[7]
  - Custom-tailor reliability to application needs[8]

[7] Herder: *Building a Dependable Operating System – Fault Tolerance in MINIX3*, PhD Thesis, 2010

[8] Sridharan, Kaeli: *Eliminating microarchitectural dependency from architectural vulnerability*, HPCA 2009

Why A Microkernel?

- Small components
  - Microrebootable[7]
  - Custom-tailor reliability to application needs[8]
- That's what we do in Dresden (tm).
  - Reuse Fiasco.OC mechanisms instead of adding new code to the RCB

[7] Herder: *Building a Dependable Operating System – Fault Tolerance in MINIX3*, PhD Thesis, 2010

[8] Sridharan, Kaeli: *Eliminating microarchitectural dependency from architectural vulnerability*, HPCA 2009

Why A Microkernel?

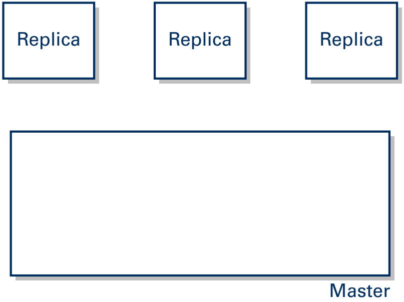
- Small components
  - Microrebootable[7]
  - Custom-tailor reliability to application needs[8]
- That's what we do in Dresden (tm).
  - Reuse Fiasco.OC mechanisms instead of adding new code to the RCB
- Lean system call interface
  - Need to add special handling to fewer syscalls

[7] Herder: *Building a Dependable Operating System – Fault Tolerance in MINIX3*, PhD Thesis, 2010

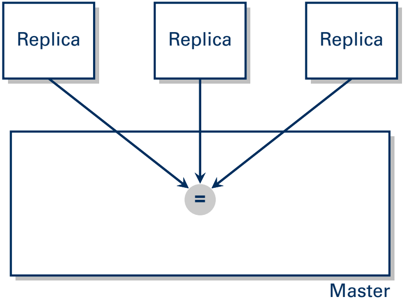
[8] Sridharan, Kaeli: *Eliminating microarchitectural dependency from architectural vulnerability*, HPCA 2009

Romain: Structure

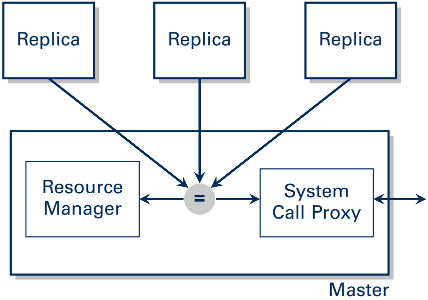
Romain: Structure



Romain: Structure

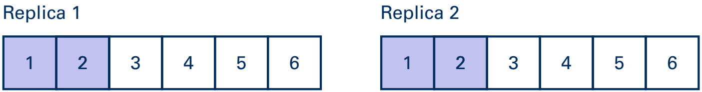


Romain: Structure

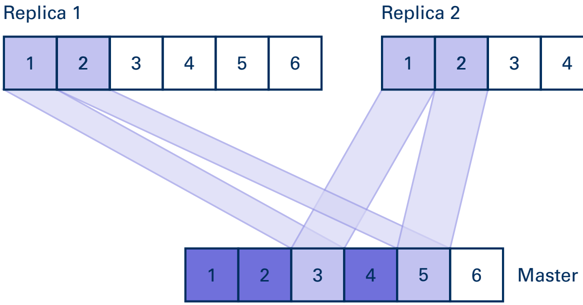


Resource Management: Capabilities

Resource Management: Capabilities

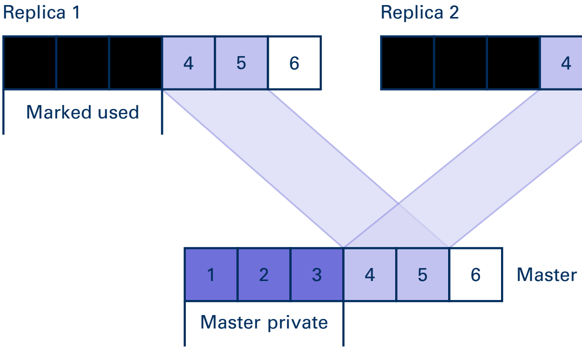


Resource Management: Capabilities

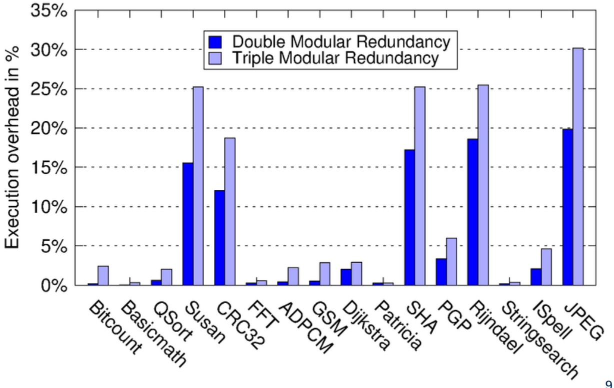


TECHNISCHE UNIVERSITÄT DRESDEN

Partitioned Capability Tables



Overhead vs. Unreplicated Execution



[9]

[9] Döbel, Härtig, Engel: *Operating System Support for Redundant Multithreading*, EMSOFT 2012

Romain Lines of Code

Base code (main, logging, locking)	325
Application loader	375
Replica manager	628
Redundancy	153
Memory manager	445
System call proxy	311
Shared memory	281
Total	2,518
Fault injector	668
GDB server stub	1,304

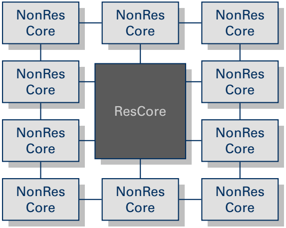
Hardening the RCB

- **We need:** Dedicated mechanisms to protect the RCB (HW or SW)
- **We have:** Full control over software
- Use FT-encoding compiler?
  - Has not been done for kernel code yet
  - Only protects SW components
- RAD-hardened hardware?
  - Too expensive

Hardening the RCB

- **We need:** Dedicated mechanisms to protect the RCB (HW or SW)
- **We have:** Full control over software
- Use FT-encoding compiler?
  - Has not been done for kernel code yet
  - Only protects SW components
- RAD-hardened hardware?
  - Too expensive

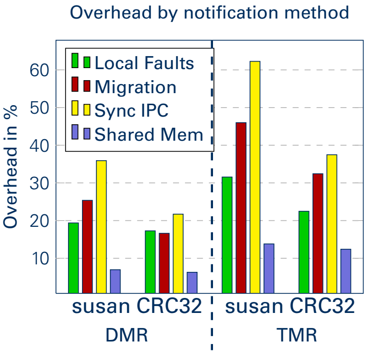
Our proposal: Split HW into ResCores and NonRes-Cores



Signaling Performance

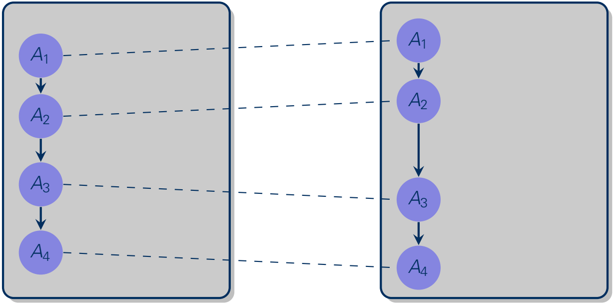
Exploring master–replica communication[10]

- 12x Intel Core2 2.6 GHz
- Replicas pinned to dedicated physical cores
- Hyperthreading off



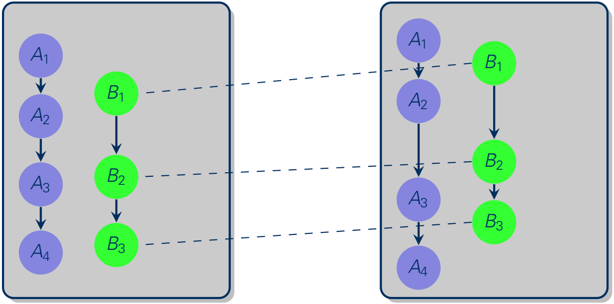
[10] Döbel, Härtig: *Who watches the watchmen? – Protecting Operating System Reliability Mechanisms*, HotDep 2012

How About Multithreading?

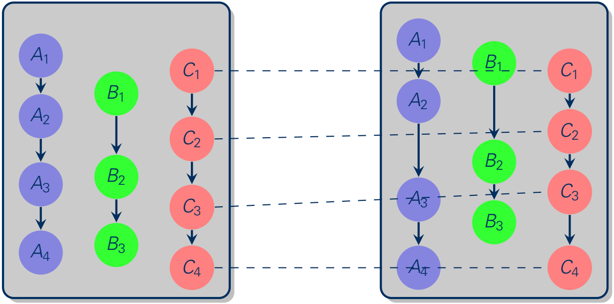


TECHNISCHE UNIVERSITÄT DRESDEN

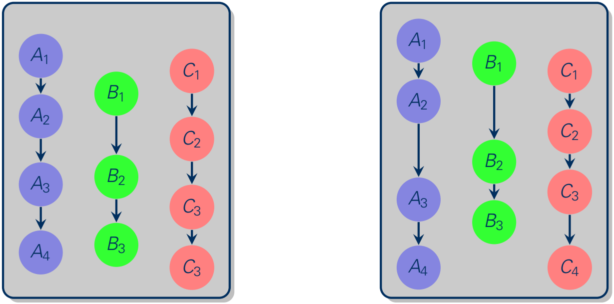
How About Multithreading?



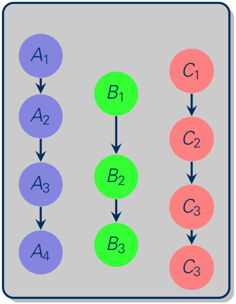
How About Multithreading?

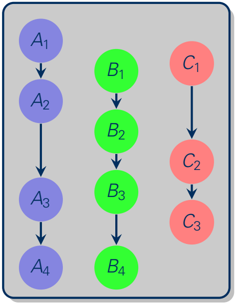


Problem: Nondeterminism



Problem: Nondeterminism





Deterministic Multithreading

- Related work: Debugging multithreaded programs
  - Slightly different requirement: determinism across runs

[11] Liu, Curtsinger, Berger: *DThreads: Efficient Deterministic Multithreading*, OSDI 2011

[12] Olszewski, Ansel, Amarasinghe: *Kendo: Efficient Deterministic Multithreading in Software*, ASPLOS 2009

Deterministic Multithreading

- Related work: Debugging multithreaded programs
  - Slightly different requirement: determinism across runs
- **Strong Determinism**: All accesses to shared resources happen in the same order[11].
  - Requires heavy involvement with shared memory accesses
  - Replicating SHM is slow

[11] Liu, Curtsinger, Berger: *DThreads: Efficient Deterministic Multithreading*, OSDI 2011

[12] Olszewski, Ansel, Amarasinghe: *Kendo: Efficient Deterministic Multithreading in Software*, ASPLOS 2009

Deterministic Multithreading

- Related work: Debugging multithreaded programs
  - Slightly different requirement: determinism across runs
- **Strong Determinism**: All accesses to shared resources happen in the same order[11].
  - Requires heavy involvement with shared memory accesses
  - Replicating SHM is slow
- **Weak Determinism**: All lock acquisitions in a program happen in the same order[12]
  - Intercept calls to `pthread_mutex_{lock,unlock}`

[11] Liu, Curtsinger, Berger: *DThreads: Efficient Deterministic Multithreading*, OSDI 2011

[12] Olszewski, Ansel, Amarasinghe: *Kendo: Efficient Deterministic Multithreading in Software*, ASPLOS 2009

Externally Enforced Determinism

- Patch entries to `pthread_mutex_{lock,unlock}`
- Catch exception for every call
- Enforce ordering in side the master

Microbenchmark: 2 threads, global counter, 1 lock

Replication kind	Execution time	Overhead
Native execution	0.24 s	1.00 x
Unreplicated *RomainMT*	4.50 s	18.75 x
RomainMT: DMR	12.72 s	53.06 x
RomainMT: TMR	18.02 s	75.00 x

Internal Determinism

- Exception for every lock/unlock hurts a lot!
- Non-contention case: get progress without exception

Internal Determinism

- Exception for every lock/unlock hurts a lot!
- Non-contention case: get progress without exception
- Replication-aware `pthreads` library
  - Shared memory between replicas
  - Exchange per-lock progress information
  - Only block if lock currently used by different thread in other replica

Internal Determinism

- Exception for every lock/unlock hurts a lot!
- Non-contention case: get progress without exception
- Replication-aware `pthreads` library
  - Shared memory between replicas
  - Exchange per-lock progress information
  - Only block if lock currently used by different thread in other replica
- Problems:
  - Replacing `pthreads` vs. binary-only support $\rightarrow$ `pthreads` is a shared lib anyway
  - No support for different synchronization mechanisms
  - New `pthreads` code becomes part of the RCB.

Internal Determinism

Microbenchmark: 2 threads, global counter, 1 lock

Replication kind	Execution time	Overhead	External det.
Native execution	0.24 s	1.00 x	
Unreplicated *RomainMT*	0.27 s	1.13 x	18.75 x
RomainMT: DMR	0.46 s	1.92 x	53.06 x
RomainMT: TMR	1.43 s	6.01 x	75.00 x

TECHNISCHE UNIVERSITÄT DRESDEN

Conclusion

- Redundant Multithreading as an OS service
- Support for binary-only applications
- Benefit from microkernel by reuse and design
- Overheads $<30\%$, often $<5\%$
- Multithreading – external vs. internal determinism
- Work in progress (not in this talk):
  - Shared memory handling is slow
  - Bounded detection latency using a watchdog
  - Dynamic adjustment of replication level and resource usage

Nothing to see here

This slide intentionally left blank.

Except for above text.

What about signalling failures?

Missed CPU exceptions	$\rightarrow$	detected by watchdog
Spurious CPU exceptions	$\rightarrow$	detected by watchdog / state comparison
Transmission of corrupt state	$\rightarrow$	detected during state comparison

Overwriting remote state during transmission

- NonResCore memory
- Accessible by ResCores, but not by other NonResCores
- Prevents overwriting other states
- Already available in HW: IBM/Cell

Romain

http://www.dynamo-dresden.de