TECHNISCHE UNIVERSITÄT DRESDEN

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

Operating Systems meet Fault Tolerance

Björn Döbel

Dresden, 2012-01-24

„If there's more than one possible outcome of a job or task, and one of those outcomes will result in disaster or an undesirable consequence, then somebody will do it that way."

(Edward Murphy jr.)

Outline

- Murphy and OS software: Is it really that bad?
- Fault-Tolerant Operating Systems
  - Minix3
  - CuriOS
  - L4ReAnimator
- Creative OS Debugging
  - DataCollider

Why things go wrong (TM)

- “Hey, this pointer is certainly never going to be NULL.”
  Using C as a programming language
- “Of course, someone in the higher layers will already have checked this return value.”
  Layering vs. responsibility
- “This struct is shared between an interrupt handler and a thread. But they will never run in parallel.”
  Concurrency
- “But the device spec said, this was not allowed to happen!”
  Hardware
- “I'm a cool OS hacker. I won't make mistakes, so I don't need to test my code.”
  Hypocrisy

A Classic

- "An Empirical Study of Operating System Errors", Andy Chou et al., SOSP 2001
- Automated software error detection (today: http://www.coverity.com)
- Target: Linux Kernel (versions 1.0 – 2.4)
  - Where are the errors?
  - How are they distributed?
  - How long do they survive?
  - Do bugs cluster in certain locations?

Revalidation:

- “Faults in Linux: 10 years later”, N. Palix et al., ASPLOS 2011
- Work on tool support for decreasing error counts.
- Repeated Chou's analysis until Linux 2.6.34

TECHNISCHE UNIVERSITÄT DRESDEN

Lines of Code in the Kernel

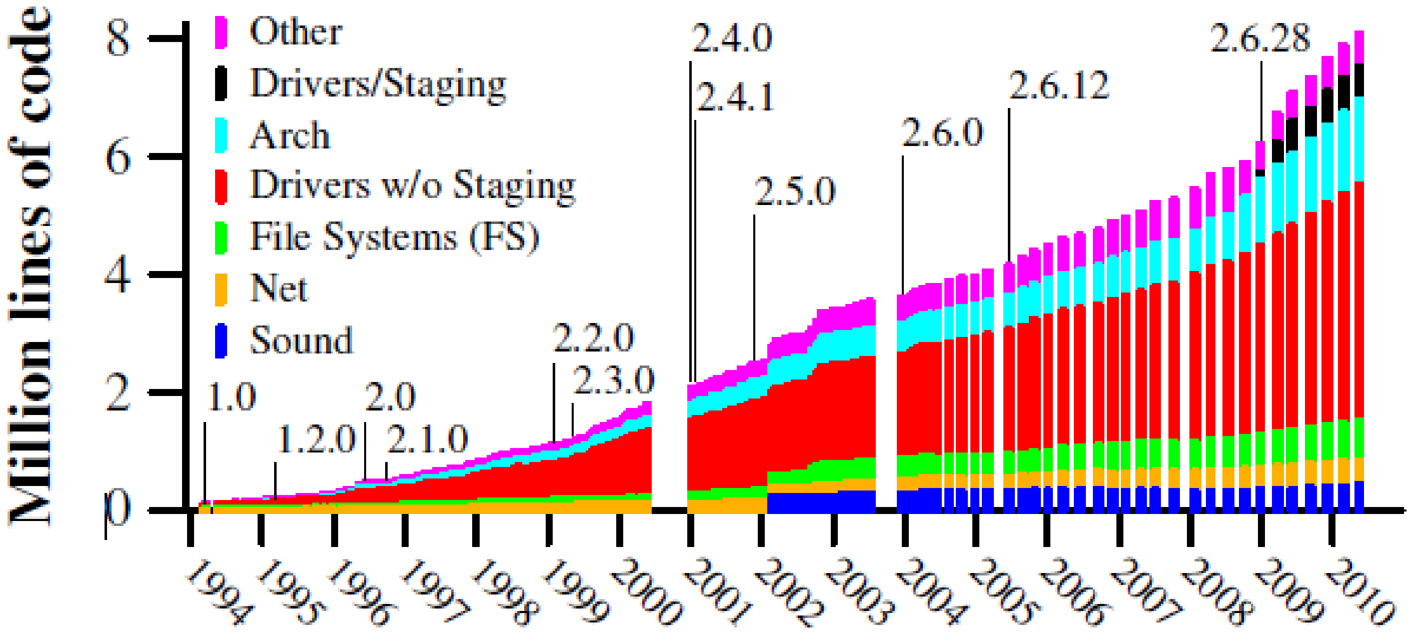
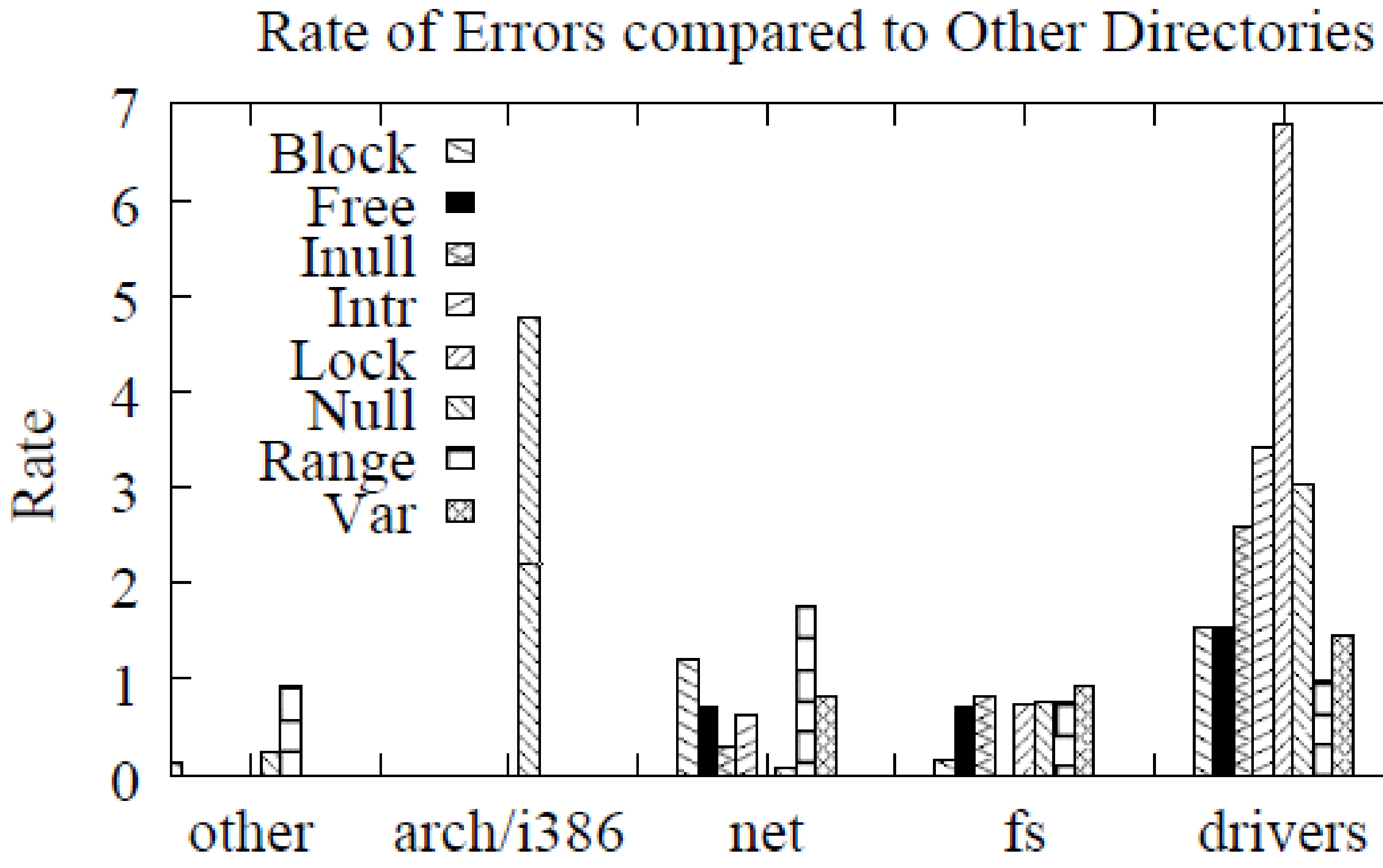


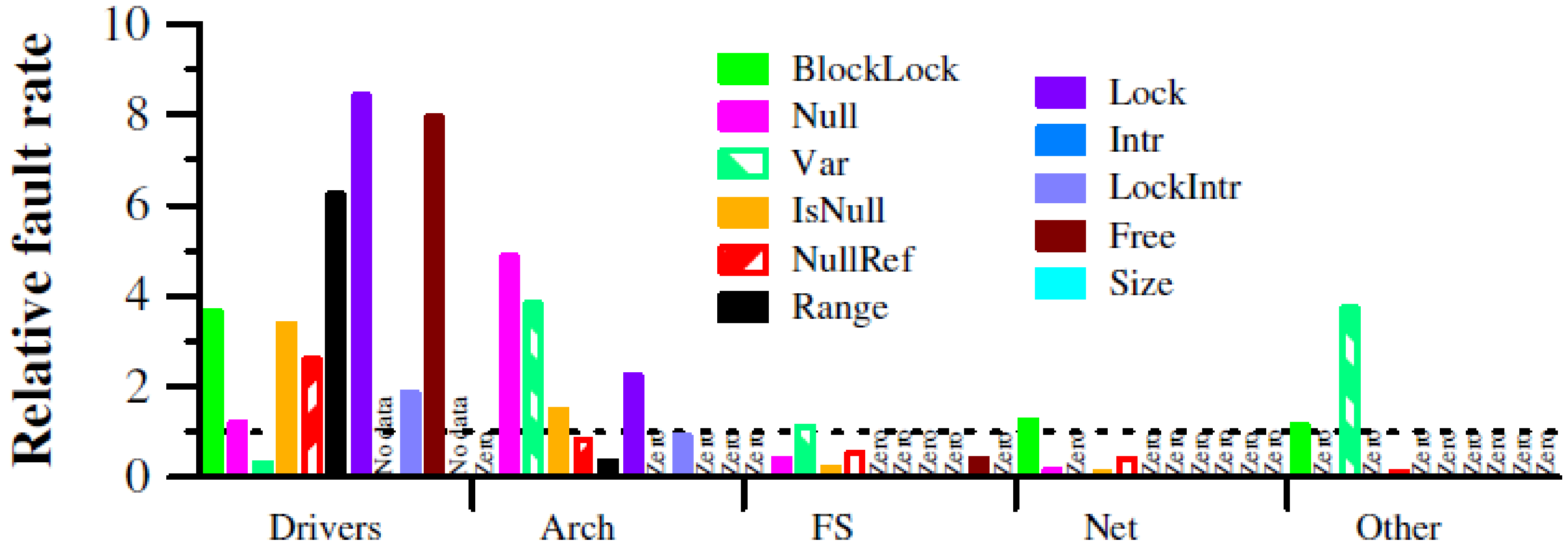
Figure 1. Linux directory sizes (in MLOC)

Fault Rate per Sub-Directory (2001)



TECHNISCHE UNIVERSITÄT DRESDEN

Fault Rate per Sub-Directory (2011)



(b) Rate of faults compared to all other directories

TECHNISCHE UNIVERSITÄT DRESDEN

Bug Lifetimes (2011)

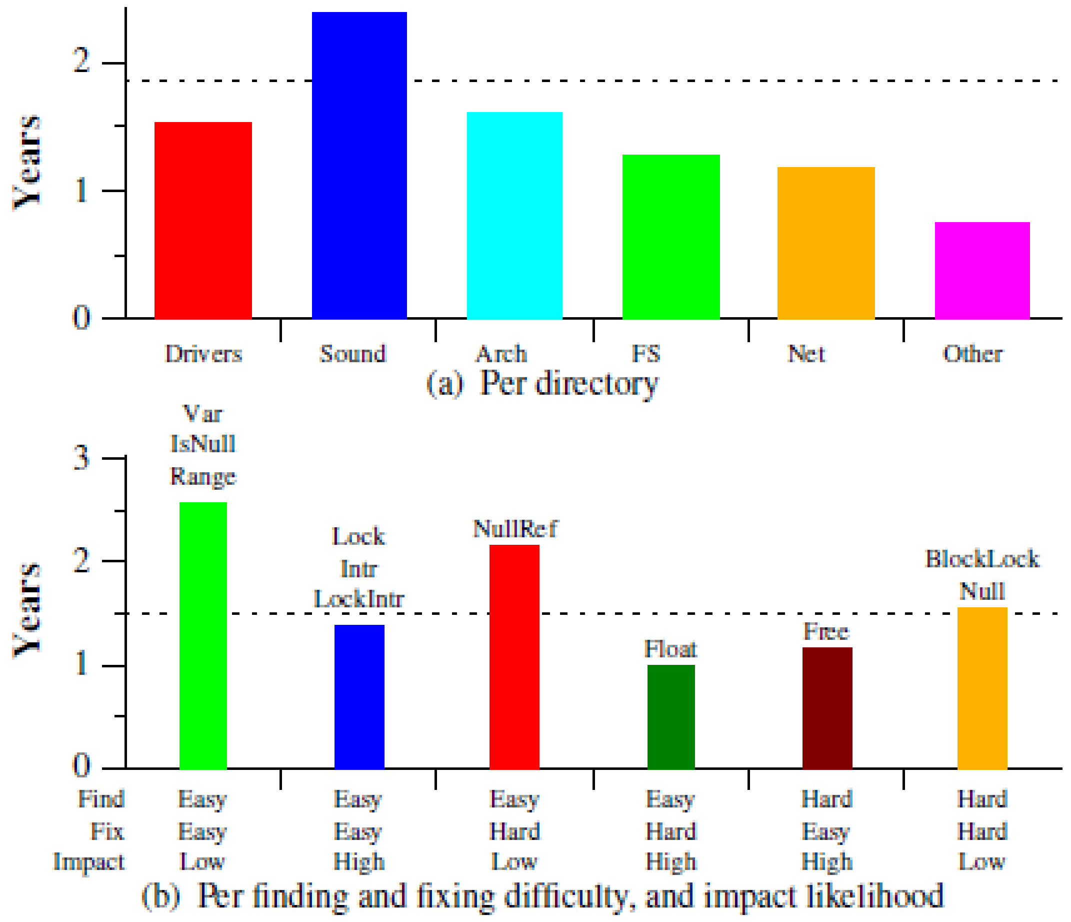


Figure 13. Average fault lifespans (without staging)

A Bug's Life

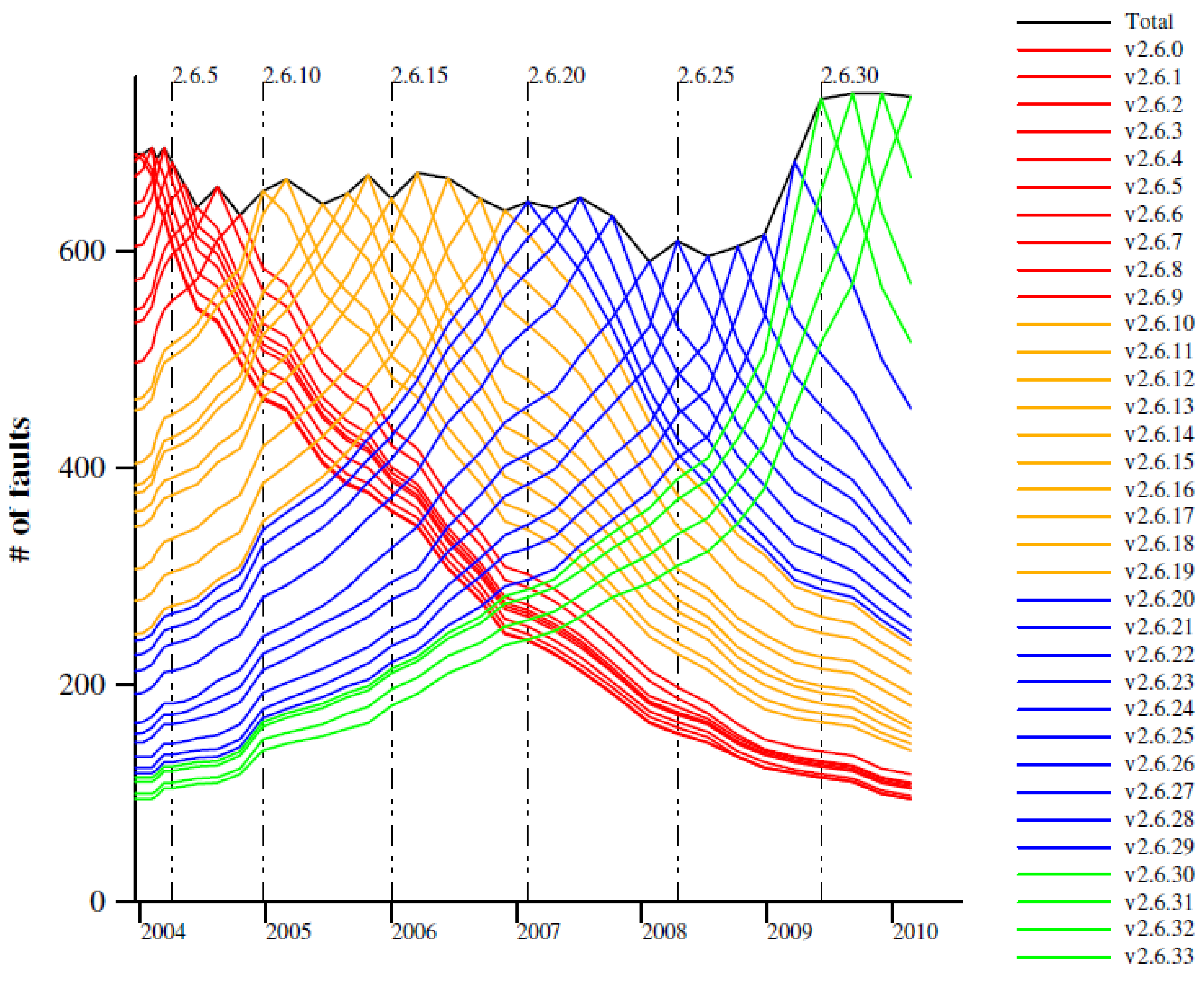


Figure 16. Lifetime of faults across versions

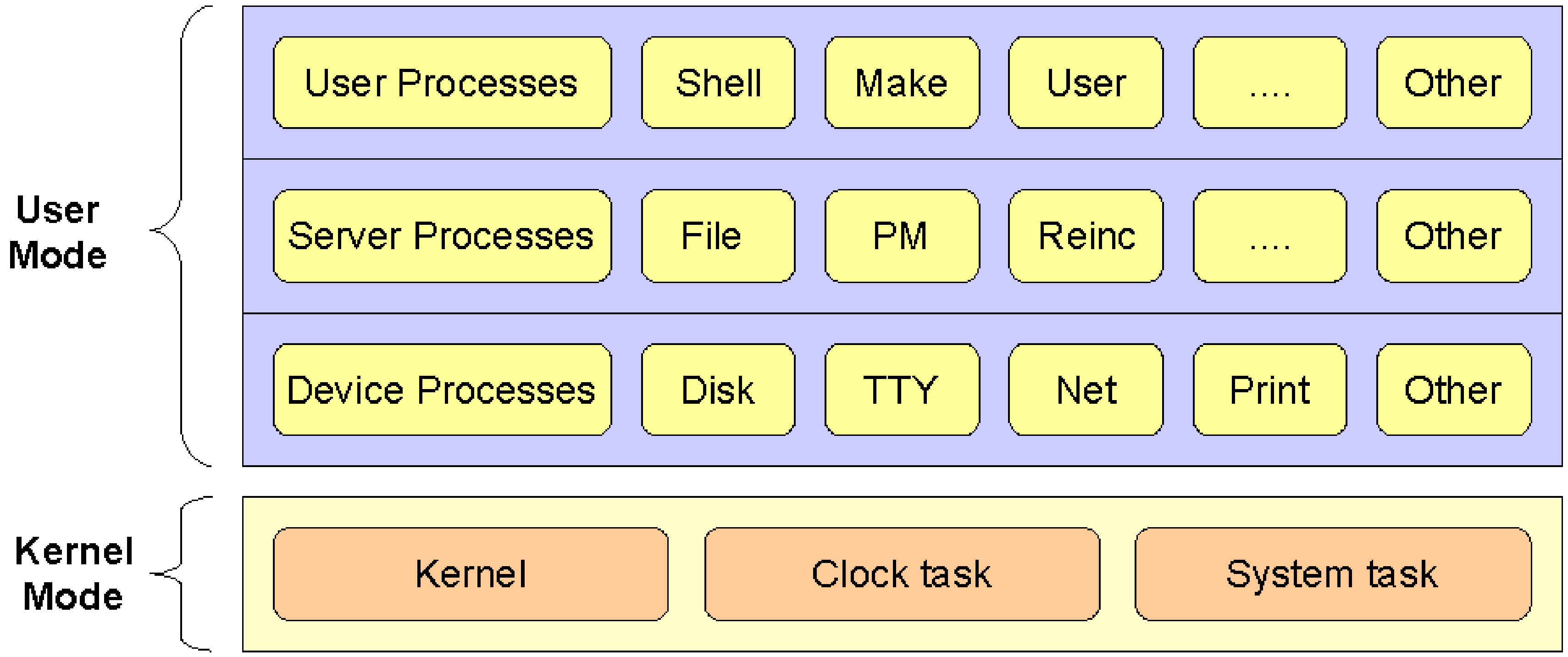
Break

- Faults are a problem.
- Hardware-related stuff is worst.

- Now, what can the OS do about it?

TECHNISCHE UNIVERSITÄT DRESDEN

Minix3 – A Fault-Tolerant OS



The MINIX 3 Microkernel Architecture

Minix3: Microkernel Fault Tolerance

- Address space isolation
  - Application can only access its private memory.
  - Faults within one application do not spread into other components.
- User-level OS services
  - Principle of Least Privilege
  - Fine-grain control over resource access
    - e.g., DMA only allowed for specific drivers
- Small components
  - Easy to replace in case of error

Minix3: Fault Detection

- Minix3 fault model: transient errors caused by software bug
  - Fix: component restart

- ***Reincarnation server*** monitors components for
  - Program termination (e.g., a crash)
  - CPU exceptions (e.g., division by zero)
  - Heartbeat messages

- Users may also indicate something is wrong
  - e.g., audio playing weirdly

Minix3: Repair

- Restarting a component is not enough:
  - Applications may **depend** on the restarted component.
  - After restart, component **state is lost**.
- Minix3 uses *explicit* mechanisms:
  - Reincarnation server notifies other applications about restart.
  - Applications store state in data store server
    - May retrieve this state after a restart
  - In any case: programmer interaction needed
    - Although, the compiler can help.

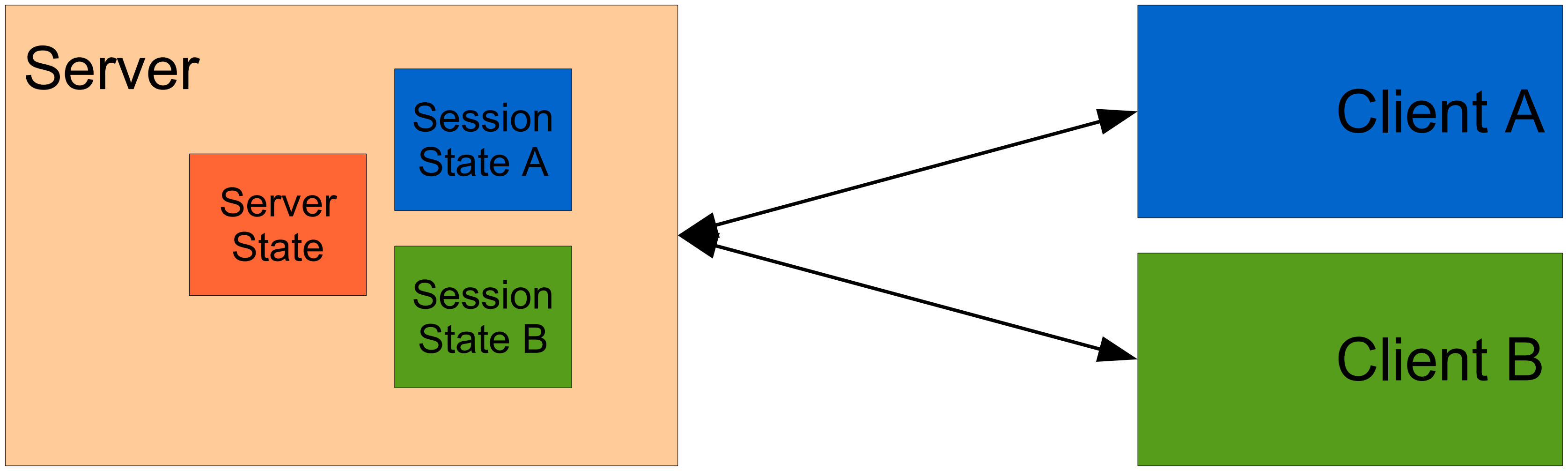
Break

- Minix3 provides fault tolerance by
  - Architectural isolation
  - Explicit monitoring and notifications

- Next:
  - CuriOS – smart session state handling
  - L4ReAnimator – semi-transparent restart in a capability-based system

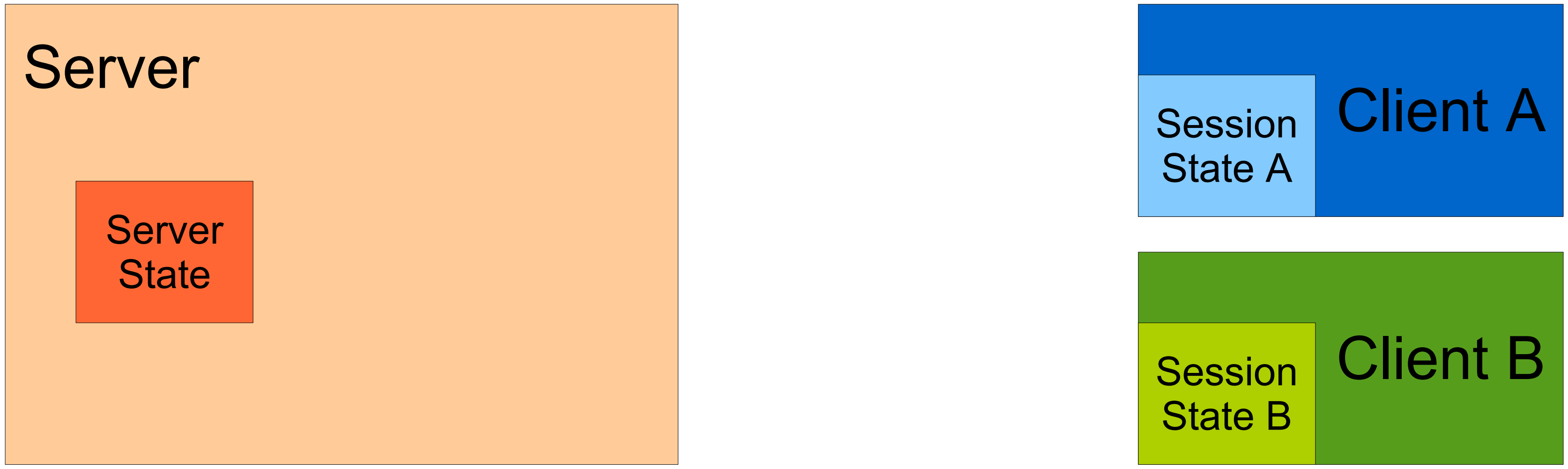
CuriOS: Servers and Sessions

- State recovery is tricky
  - Minix3: Data store keeps per-application data
  - But: often applications interact
    - Servers store client-specific **session state**
    - Would need to roll back every participant when a server is restarted



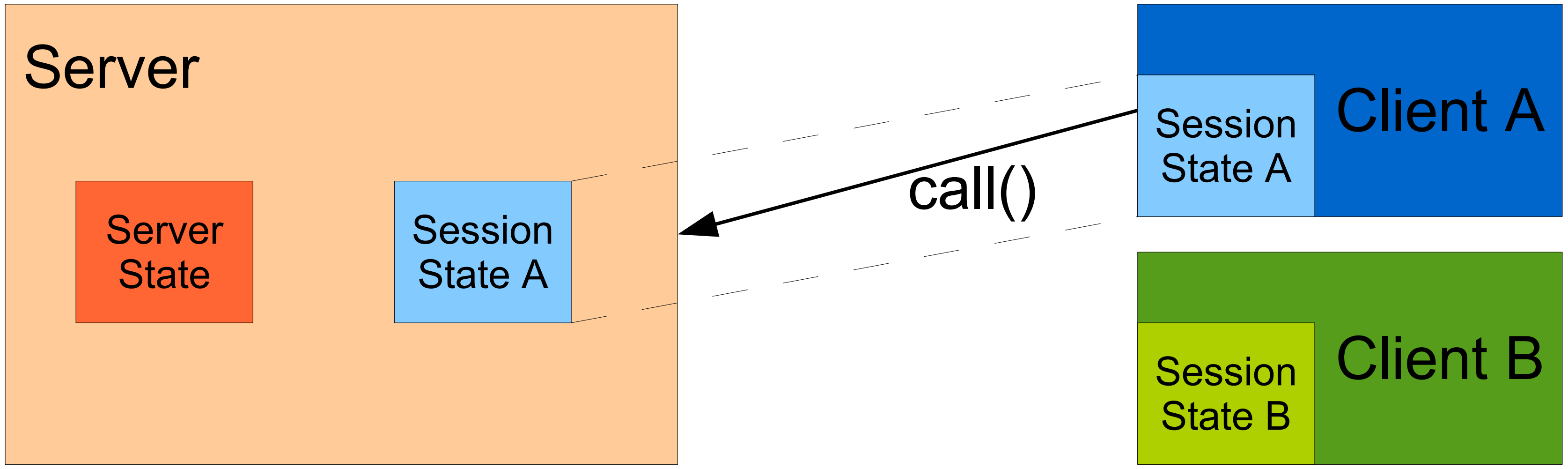
CuriOS: Server State Regions

- CuriOS kernel (CuiK) manages dedicated session memory: ***Server State Regions***
- SSRs are managed by the kernel and attached to a session (client-server connection)
  - Sessions survive server restart, because they simple are not part of the server!



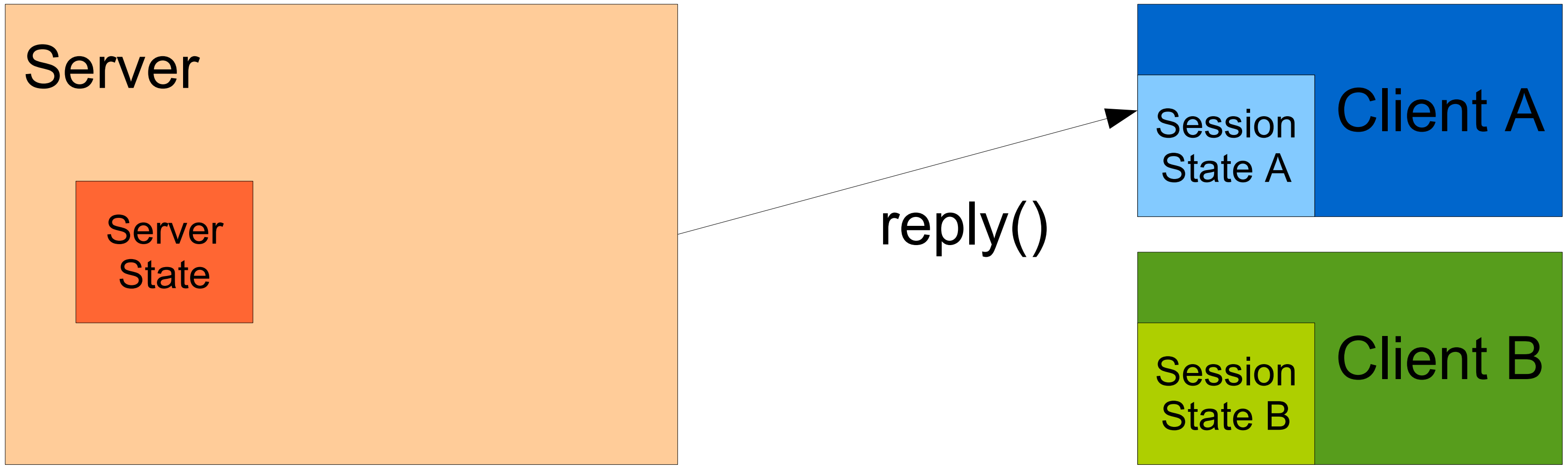
CuriOS: Protecting client sessions

- CuriOS servers get SSRs mapped only when the client actually invokes them
- Solves another problem: Failure while handling client A's request will never corrupt client B's session state



CuriOS: Protecting client sessions

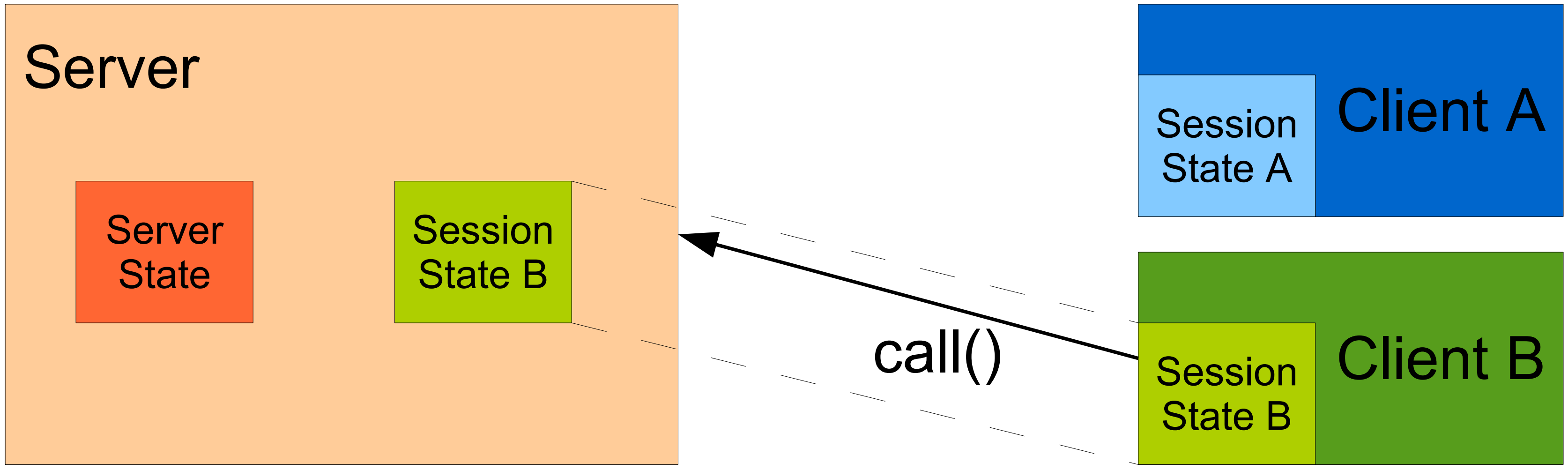
- CuriOS servers get SSRs mapped only when the client actually invokes them
- Solves another problem: Failure while handling client A's request will never corrupt client B's session state



TECHNISCHE UNIVERSITÄT DRESDEN

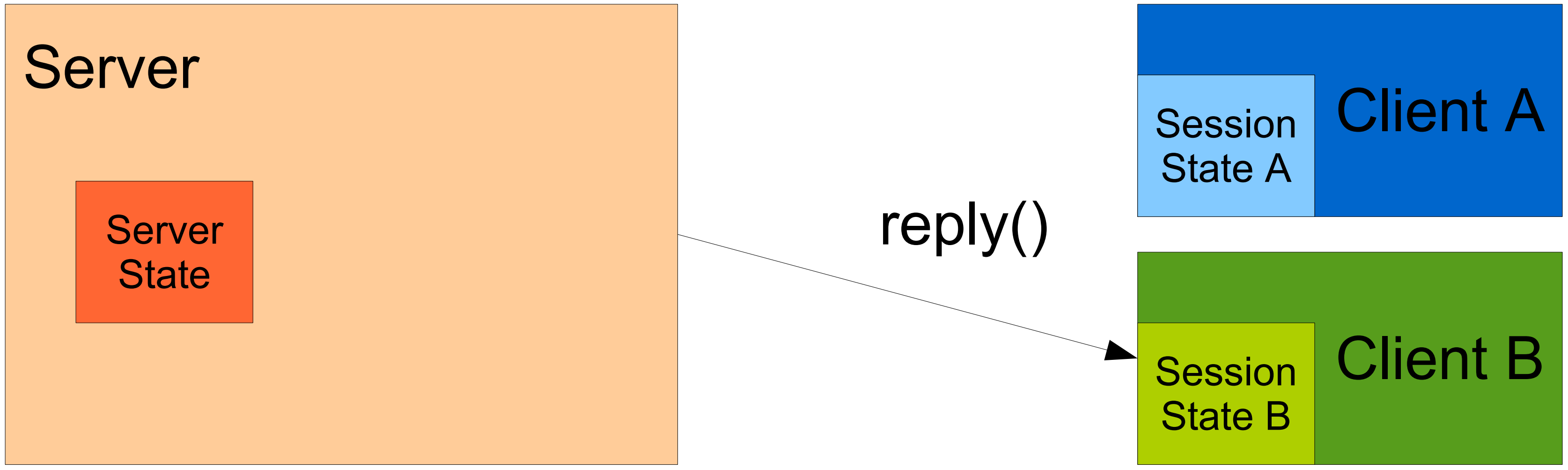
CuriOS: Protecting client sessions

- CuriOS servers get SSRs mapped only when the client actually invokes them
- Solves another problem: Failure while handling client A's request will never corrupt client B's session state



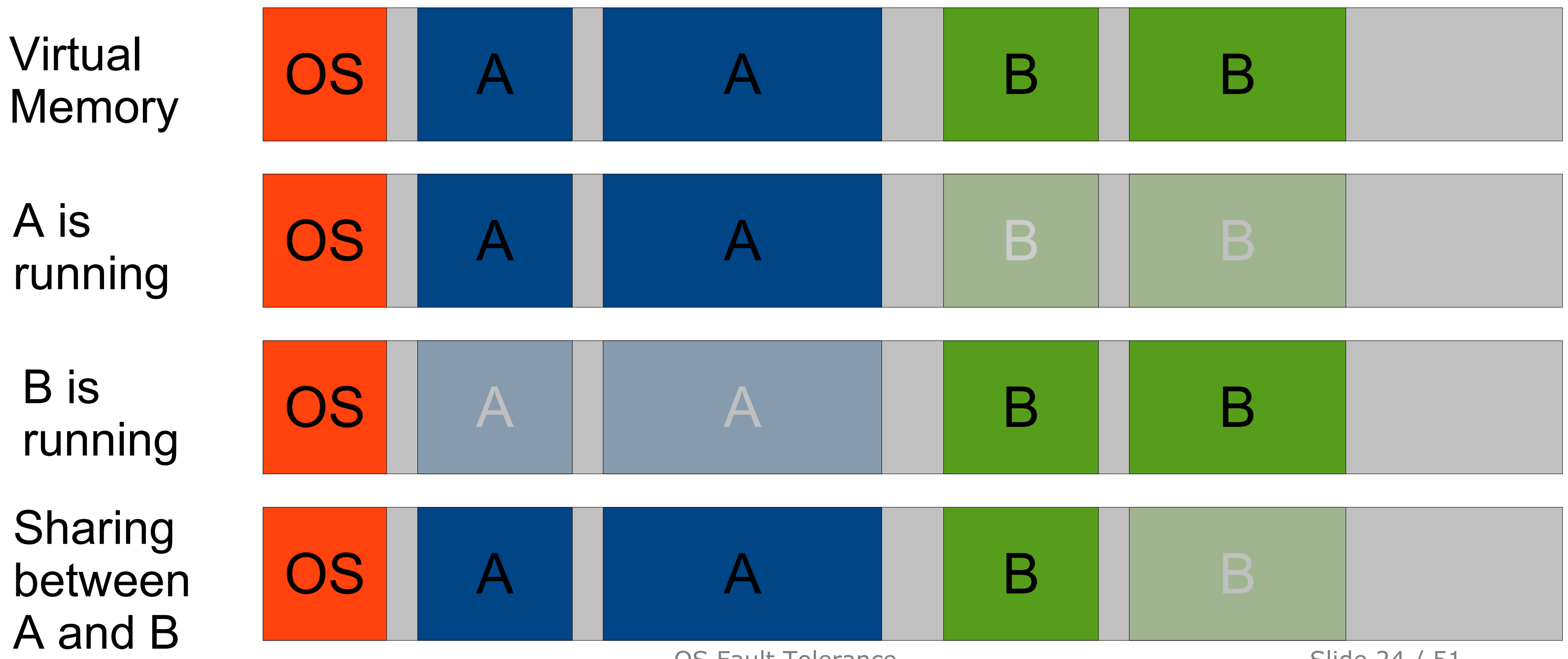
CuriOS: Protecting client sessions

- CuriOS servers get SSRs mapped only when the client actually invokes them
- Solves another problem: Failure while handling client A's request will never corrupt client B's session state



CuriOS: Transparent Restart

- CuriOS is a single-address-space OS:
  - Every application runs on the same page table (only access rights are modified)



CuriOS: Transparent Restart

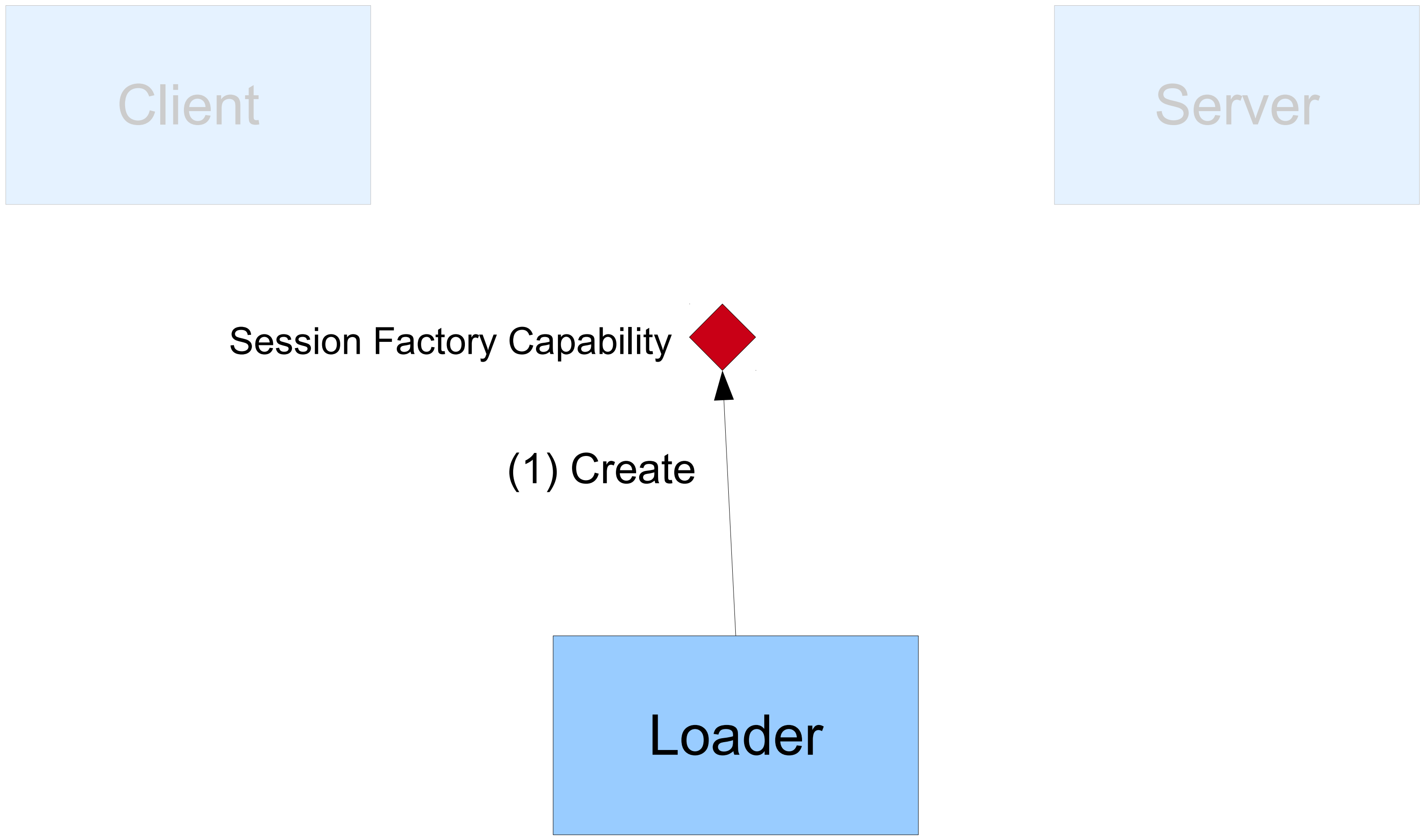
- Single Address Space
  - Each object has a unique address
  - The address is identical within all programs
  - Makes it easy to implement servers as C++ objects
- Restart
  - Restart means to replace old C++ object with a new one
  - Can easily reuse previously allocated memory location
  - All references in other applications remain valid
  - OS needs to block accesses during restart.

TECHNISCHE UNIVERSITÄT DRESDEN

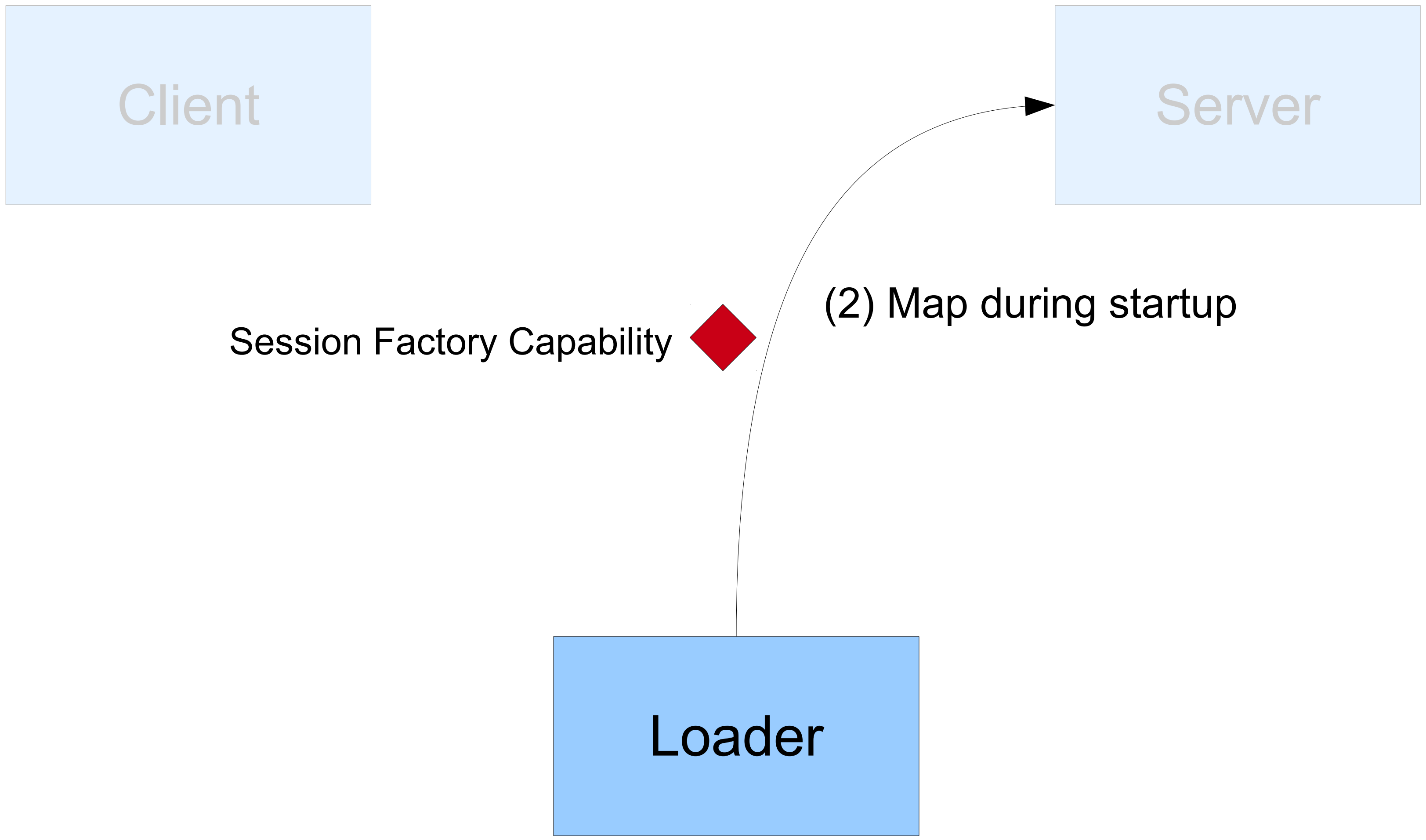
L4ReAnimator: restart on L4Re

- L4Re applications
  - Loader component: ned
  - Detects application termination: Parent signal
  - Restart: simply re-execute Lua init script (or parts of it)
- Problem after restart: capabilities (communication channels)
  - No single component that knows everyone owning a capability → Minix3 explicit signals won't work

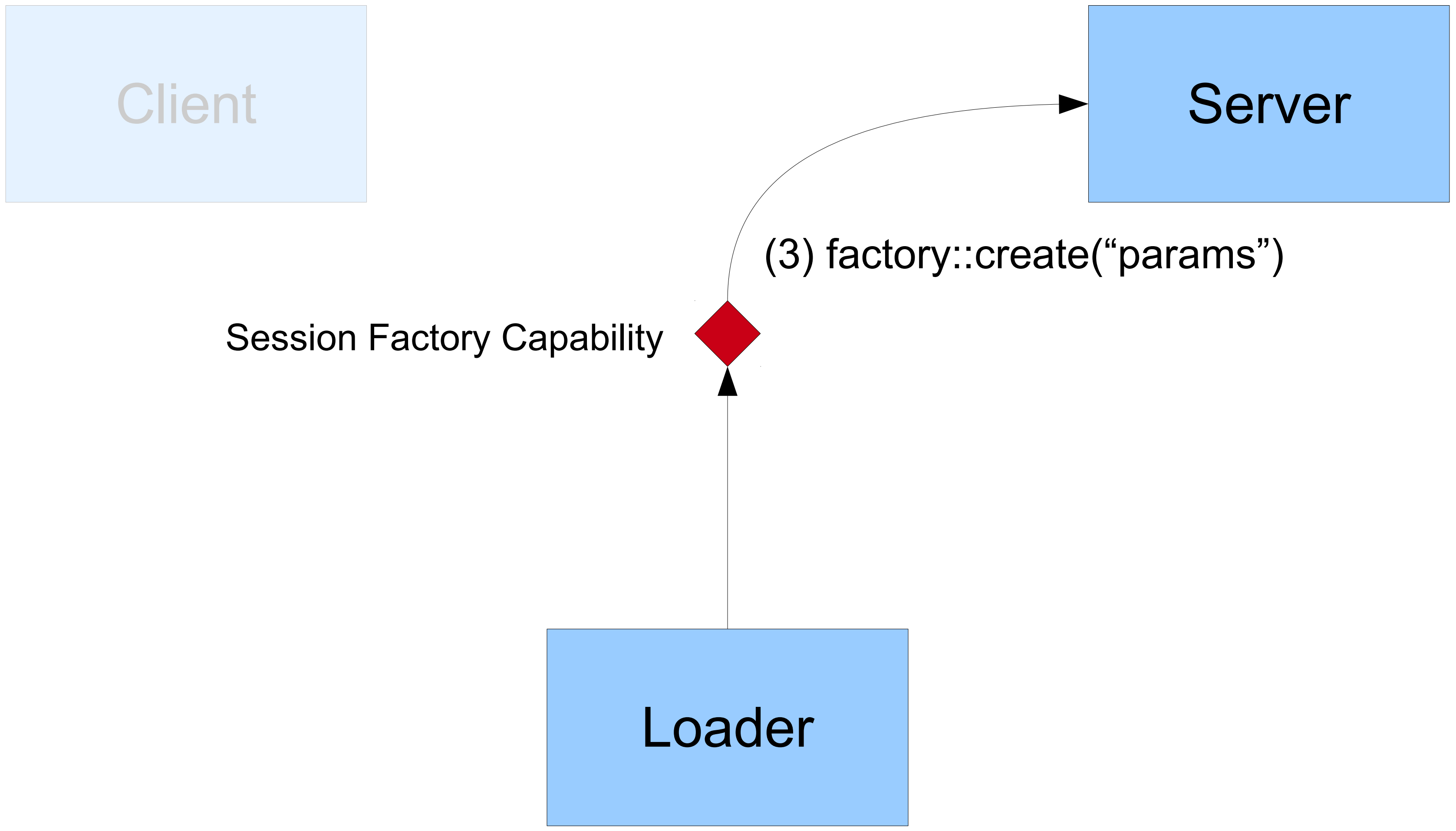
L4Re: Session Startup



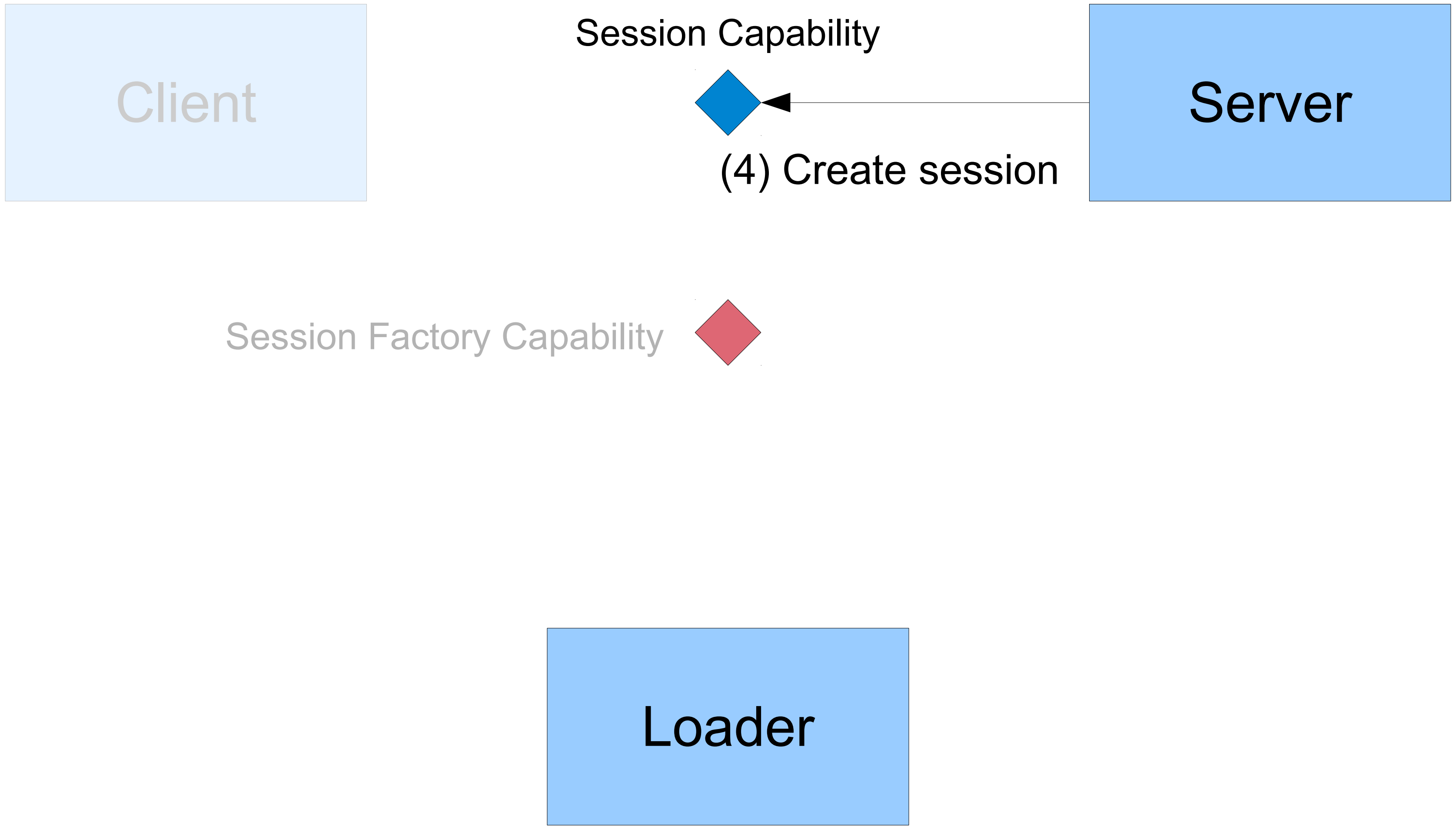
L4Re: Session Startup



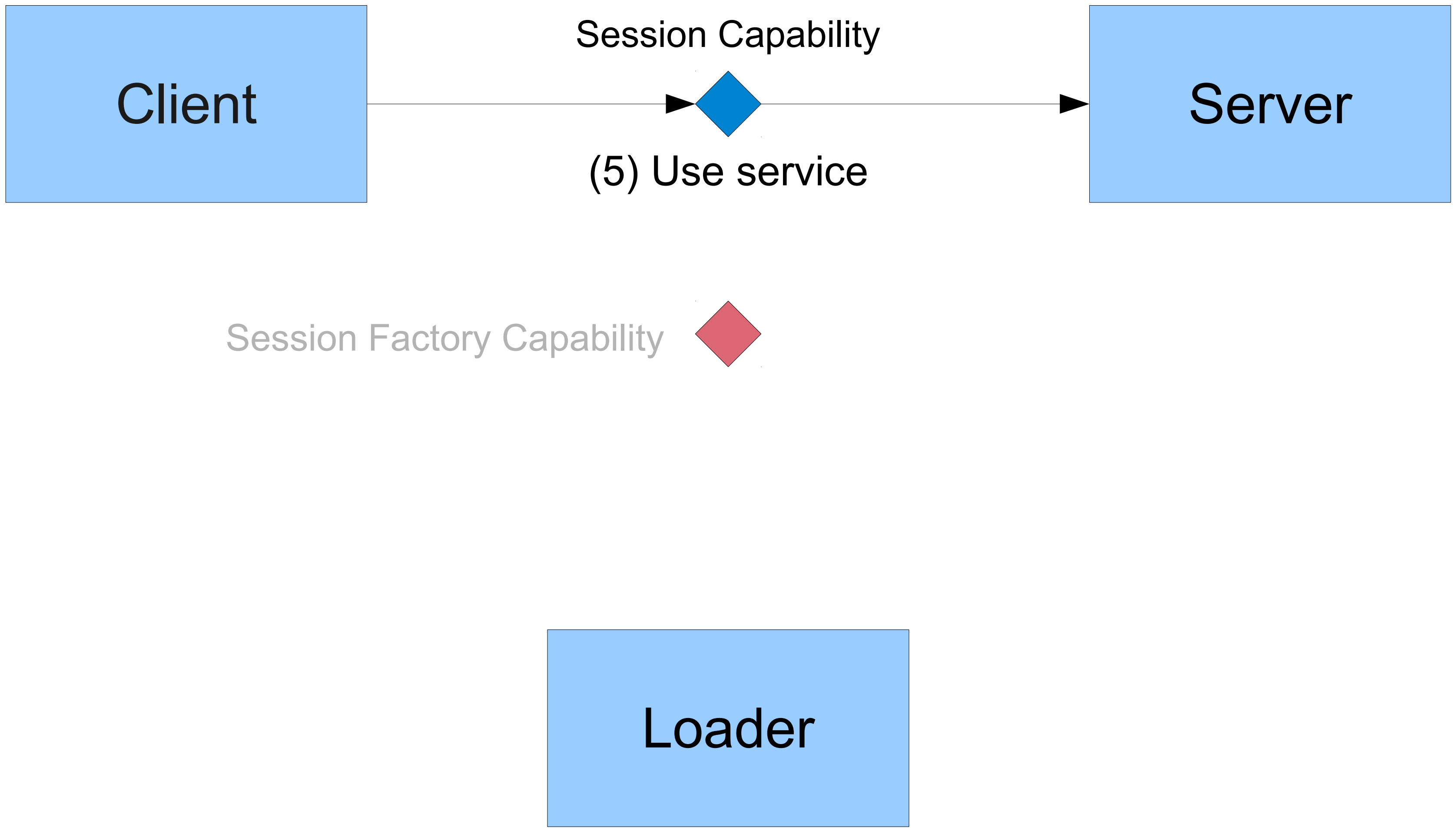
L4Re: Session Startup



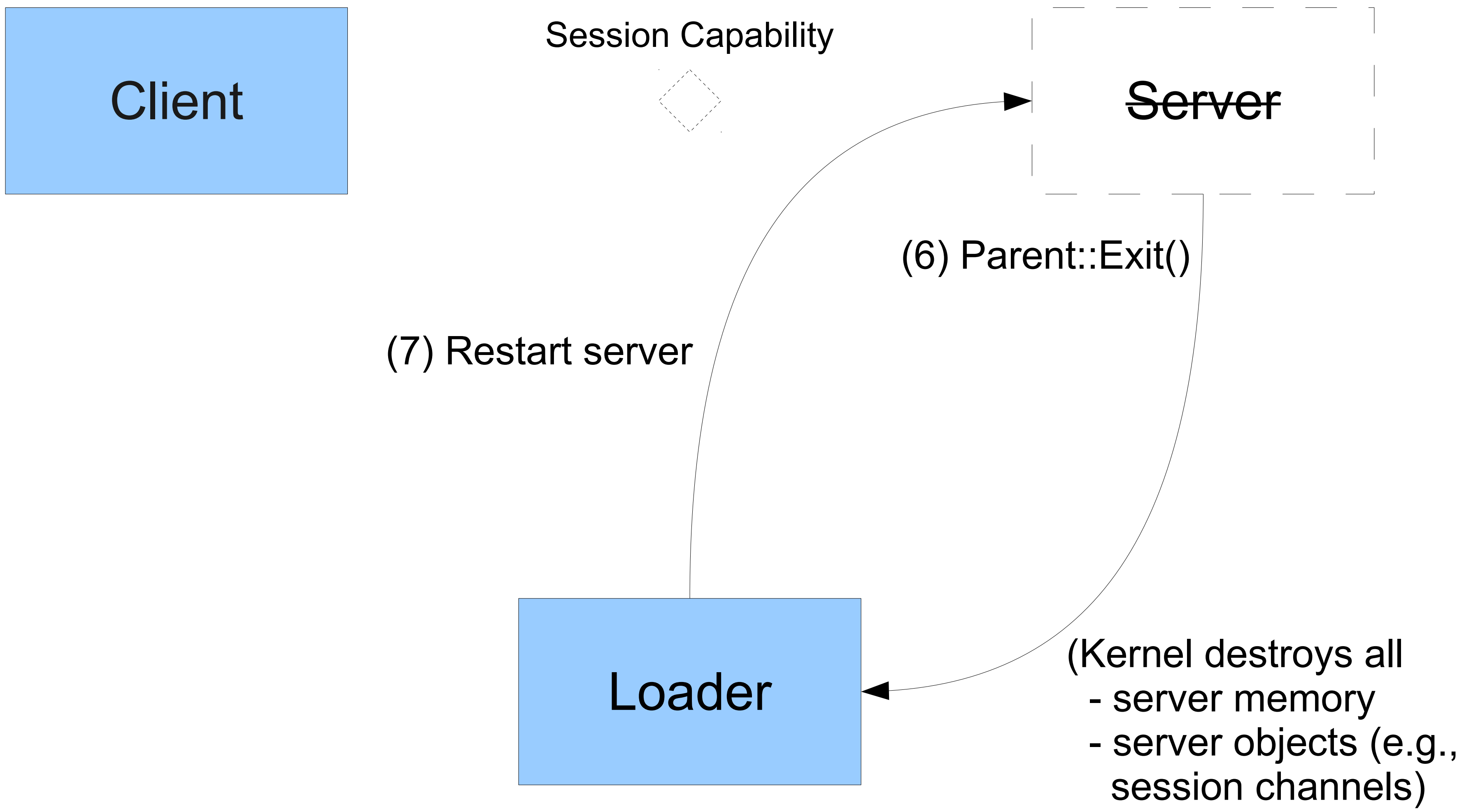
L4Re: Session Startup



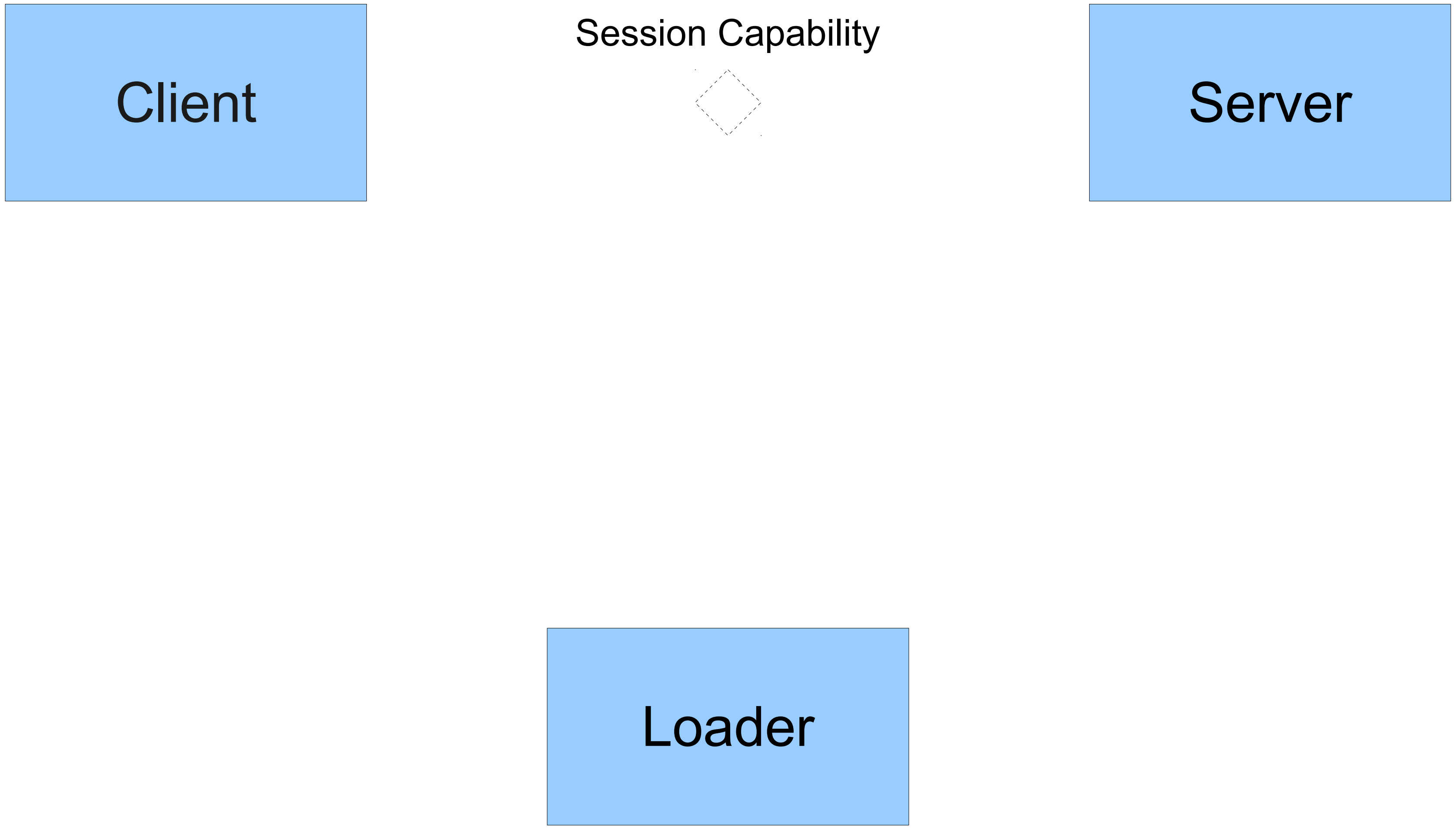
L4Re: Session Startup



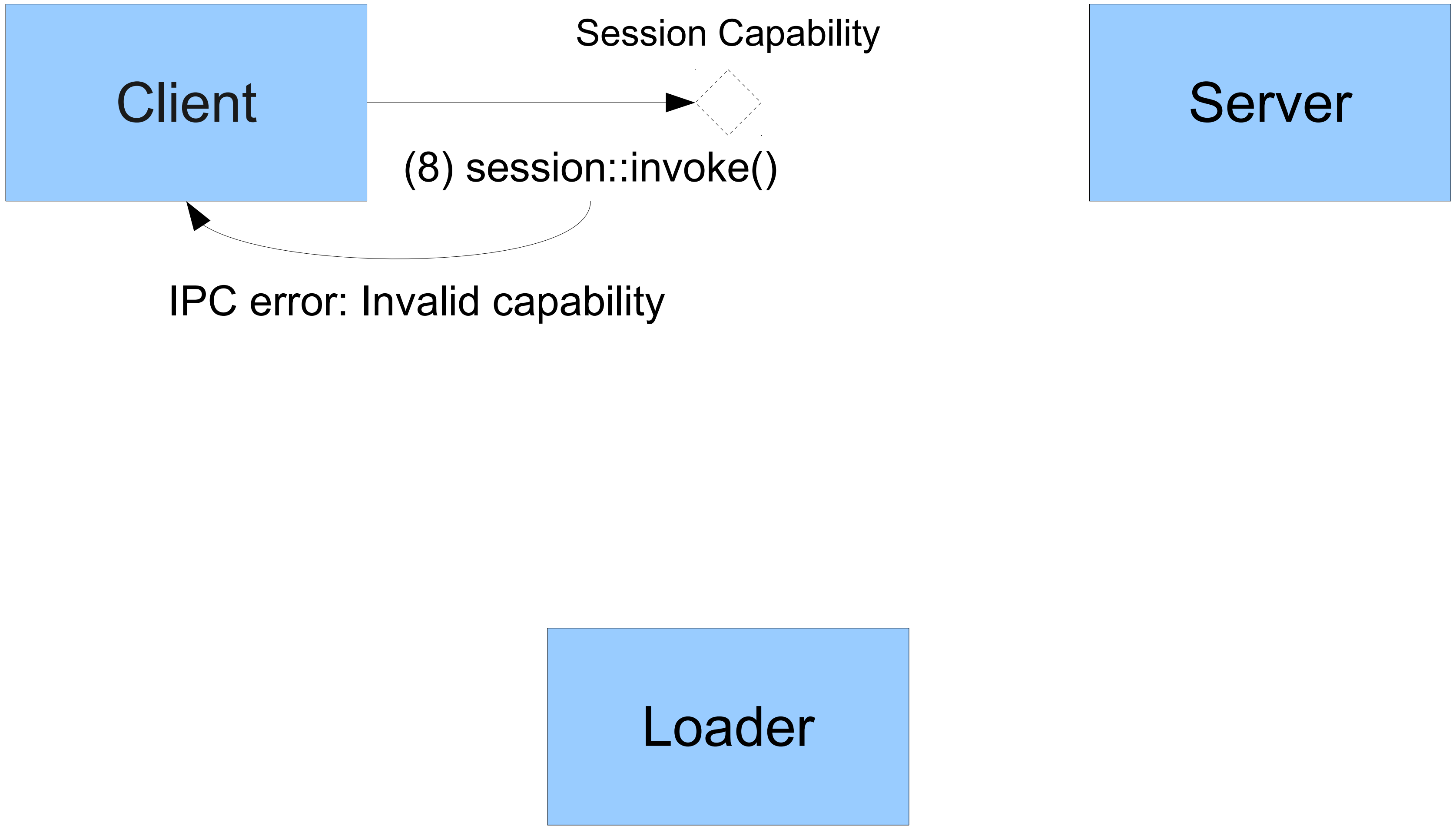
L4Re: Server crashes



L4Re: Server restarted



L4Re: Client detects destroyed session



TECHNISCHE UNIVERSITÄT DRESDEN

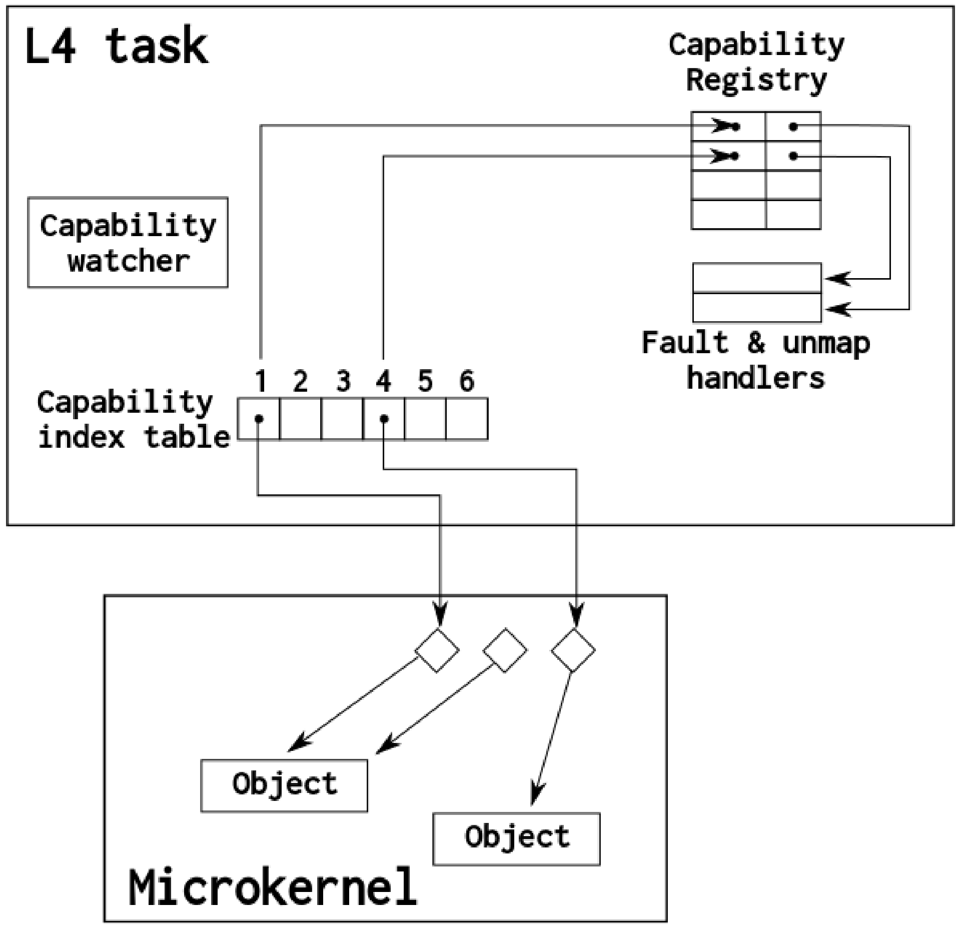
L4ReAnimator

- Only the application itself can detect that a capability vanished: kernel raises ***capability fault***
- Application needs to take care of re-obtaining the capability: execute ***capability fault handler***
- Capability fault handler: application-specific
  - Create new communication channel
  - Restore session state (e.g., from a checkpoint)
- Programming model: cap fault handlers provided by the server implementor → handling transparent for application developer → ***semi-transparency***

L4ReAnimator: cleanup

- Some channels have resources attached
  - e.g., graphical console ↔ frame buffer
- Resources may come from a different source
  - e.g., frame buffer comes from phys. memory manager
- Resources remain intact upon server crash
  - They come from a different server!
- Client ends up using an old version of a resource instead of a new one.
- Requires additional application-specific cleanup during cap fault handling → ***unmap handler***

L4ReAnimator: Architecture



- Capability watcher:
  - periodically check all capabilities for existence
  - cleanup resources on demand

TECHNISCHE UNIVERSITÄT DRESDEN

Break

- Error detection & recovery
  - Minix3 → (mostly) stateless, explicit notifications
  - CuriOS → OS-protected session state
  - L4Re → lazy reintegration
- Next: concurrency in the OS

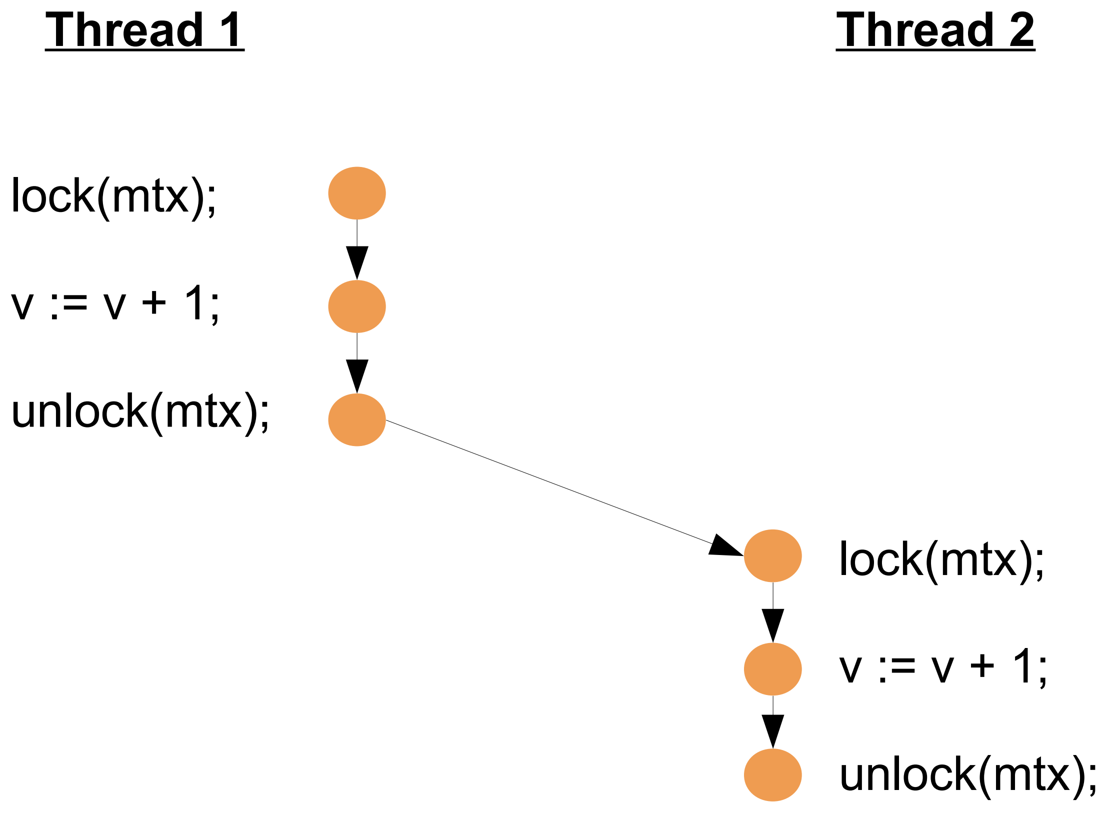
Data races

- Common definition:
  - n >= 2 threads access a memory location concurrently
  - At least one access is a write.
  - No explicit mechanism to prevent simultaneous access is used.
- Many uncritical errors → ***benign data race***
    - Update of statistical values
    - Spinlocks & Co. → ***ad-hoc synchronization***
    - only know with thorough understanding of the code
  - Synchronization primitives have multiple uses
- Reproducing a race is tricky.
- Tool-based analysis:
  - Happens-before relations
  - Lockset analysis

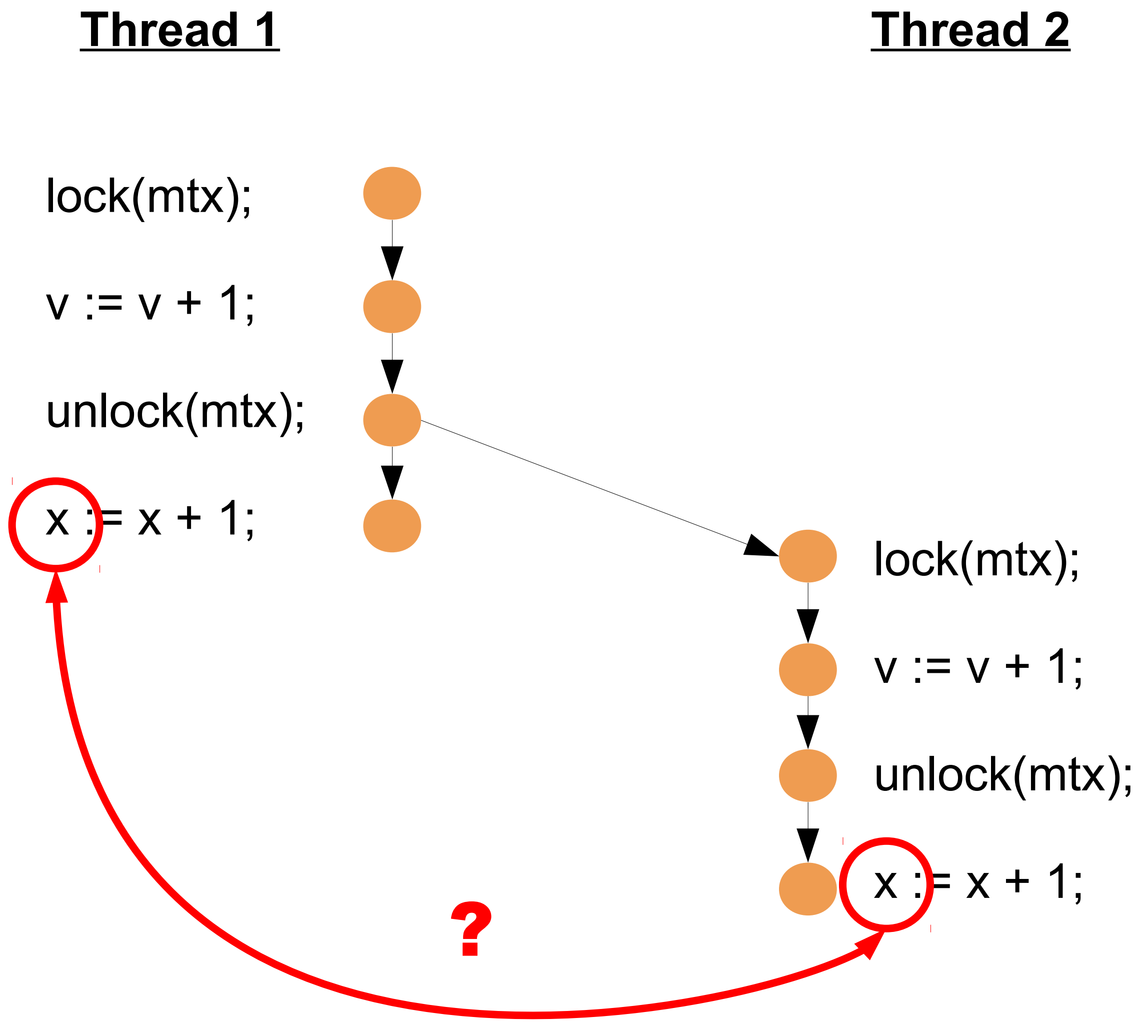
Happens-Before Relations

- Obtain trace of
  - Memory accesses
  - Use of locking primitives
  - [Netzer1991]: use of MPI messaging primitives
- Order instructions in happens-before relation:
  - Sequentially executed instructions in one thread
  - unlock()/lock() on a mutex implies inter-thread happens-before
- Memory accesses are flagged as races, if no happens-before relation can be established
- [Lamport 1978] [Netzer 1991,1993]

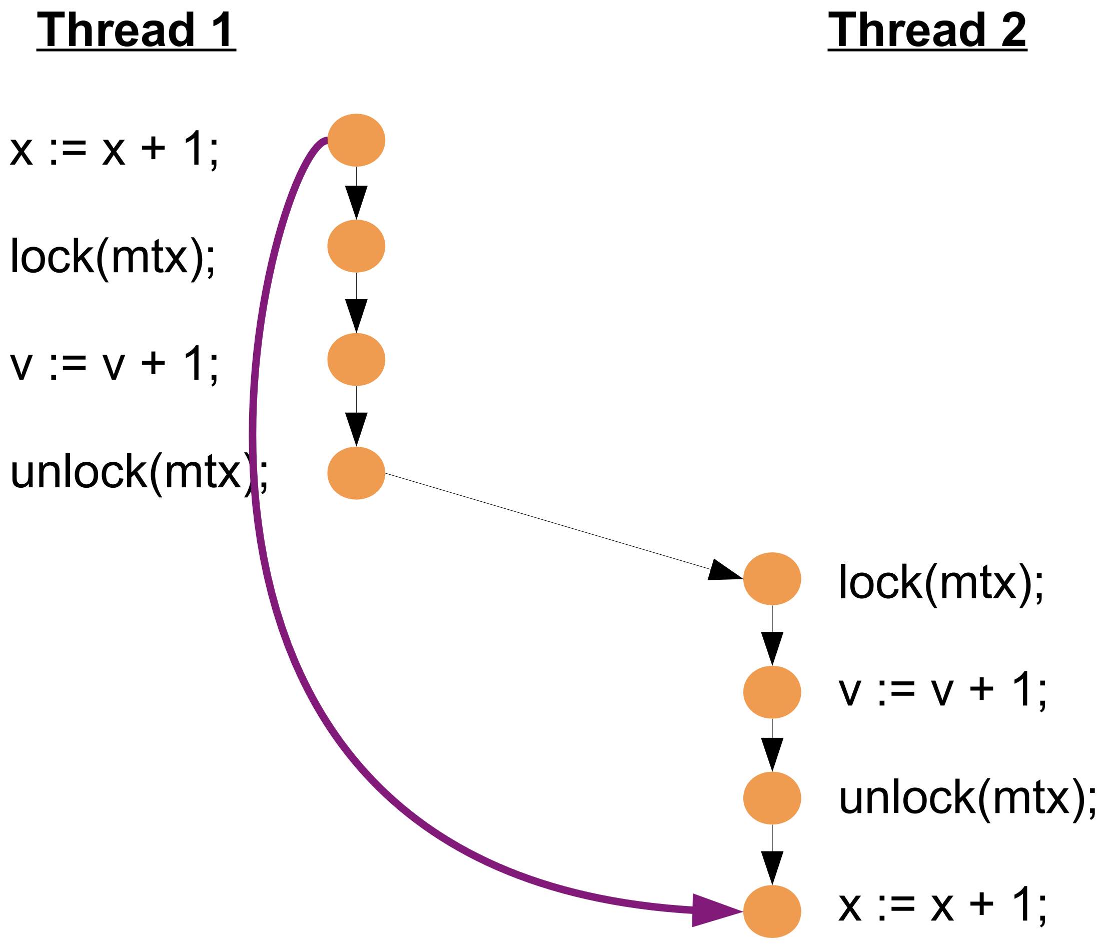
Happens-Before: Example



Happens-Before: Detected Race



Happens-Before: False negative



Race detection: Lockset analysis

- Monitor locking primitives only

- Let LOCKS(t) be the set of locks held by thread t
- For each value V initialize C(V) to the set of all locks
- For each memory access to V by t:

$$C(V) := C(V) \cap LOCKS(t)$$
$$Error\ if\ C(V) = \emptyset$$

- Dynamic [Savage 1997] and static [Engler 2003] versions available

TECHNISCHE UNIVERSITÄT DRESDEN

Race detection: Classification

- State of the art: race detection works, but produces false positives (ad-hoc synchronization) or false negatives (as seen in example)
- Next step: eliminate false positives/negatives
  - Ignore benign races
  - Identify ad-hoc synchronization
- Automation using record/replay analysis
  - Record/replay makes reproduction trivial.
  - Classification:
    - Try out all possible schedules in replay
    - Compare states after a certain point
  - Binary-level [Naraynasamy 2007] vs. language-level [Shen 2008]
  - Add optimizations to determine which schedules are interesting [Musuvathi 2008]

TECHNISCHE UNIVERSITÄT DRESDEN

Race detection and the OS

- Threads may execute in different contexts
  - No clean abstraction w.r.t. data races
  - Racy accesses may be observed in the same thread
- Many more synchronization primitives, e.g.
  - Spinlocks
  - CLI/STI
  - Semaphores
- Accesses to/from device memory
  - External state changes modify memory content
  - DMA
- Must not have unacceptable overhead

DataCollider

- Preprocessing
  - generate a set M of all memory accesses of a program
- At runtime, periodically
  - Pick k random elements from M
  - Set instruction breakpoints (x86: INT3)
- On instruction breakpoint:
  - Perform conflict detection
  - Randomly pick another element from M and set an IBP
- Post-processing
  - Identify benign races before reporting
  - Requires: Manual inspection & database of known benign races

TECHNISCHE UNIVERSITÄT DRESDEN

Conflict detection

- Approach: ***Read & Sleep & Read***
  - Read value of location
  - Delay execution
  - Read value again
  - On mismatch: ERROR
  - Issues
    - Reproduction: only one of the race participants is known
    - False negatives: all participants write the same value
- Approach: ***Hardware breakpoints***
  - Set data breakpoint (r or rw) on memory location
  - Delay execution
  - If breakpoint hits: ERROR
  - Issues
    - Doesn't work for device memory
    - Limited # of HW breakpoint registers
    - Miss: virtual address mapped to same physical address
    - False pos: same virtual address in different address spaces

TECHNISCHE UNIVERSITÄT DRESDEN

Summary

- Bugs are a fact, even in the OS.
- Careful design may help prevent / recover from lots of troubles.
- Chicken vs. egg: Who provides fault tolerance for the fault tolerance layer?

TECHNISCHE UNIVERSITÄT DRESDEN

Further Reading

- Error studies
  - Chou et al.: *"Empirical Study of OS Errors"*, SOSP 2001
  - Palix et al.: *"Faults in Linux – 10 years later"*, ASPLOS 2011
- Minix3
  - Herder et al.: *"Failure Resilience for Device Drivers"*, DSN 2007
  - Herder et al.: *"Fault Isolation for Device Drivers"*, DSN 2009
- CuriOS
  - David et al.: *"CuriOS: Improving Reliability through Operating System Structure"*, OSDI 2008
- L4ReAnimator
  - Vogt et al.: *"Stay Strong, Stay Safe – Enhancing Reliability of a Secure Operating System"*, IIDS 2010

TECHNISCHE UNIVERSITÄT DRESDEN

Further Reading

- Data Race Detection
  - L. Lamport: *"Time, clocks, and the ordering of events in a distributed system"*, 1978
  - R. Netzer: *"Optimal tracing and replay for debugging shared-memory parallel programs"*, PADD 1993
  - S. Savage et. al.: *"Eraser: A Dynamic Data Race Detector for Multithreaded Programs"*, ACM TOCS 1997
  - D. Engler et al.: *"RacerX: Effective, Static Detection of Race Conditions and Deadlocks"*, SOSP 2003
  - Musuvathi et al.: *"Finding and Reproducing Heisenbugs in Concurrent Programs"*, OSDI 2008
  - Musuvathi et al.: *"Effective Data Race Detection for the Kernel"*, OSDI 2010