L4 in Sydney: seL4, OKL4 and Friends

Gernot Heiser
NICTA and University of New South Wales
Sydney, Australia

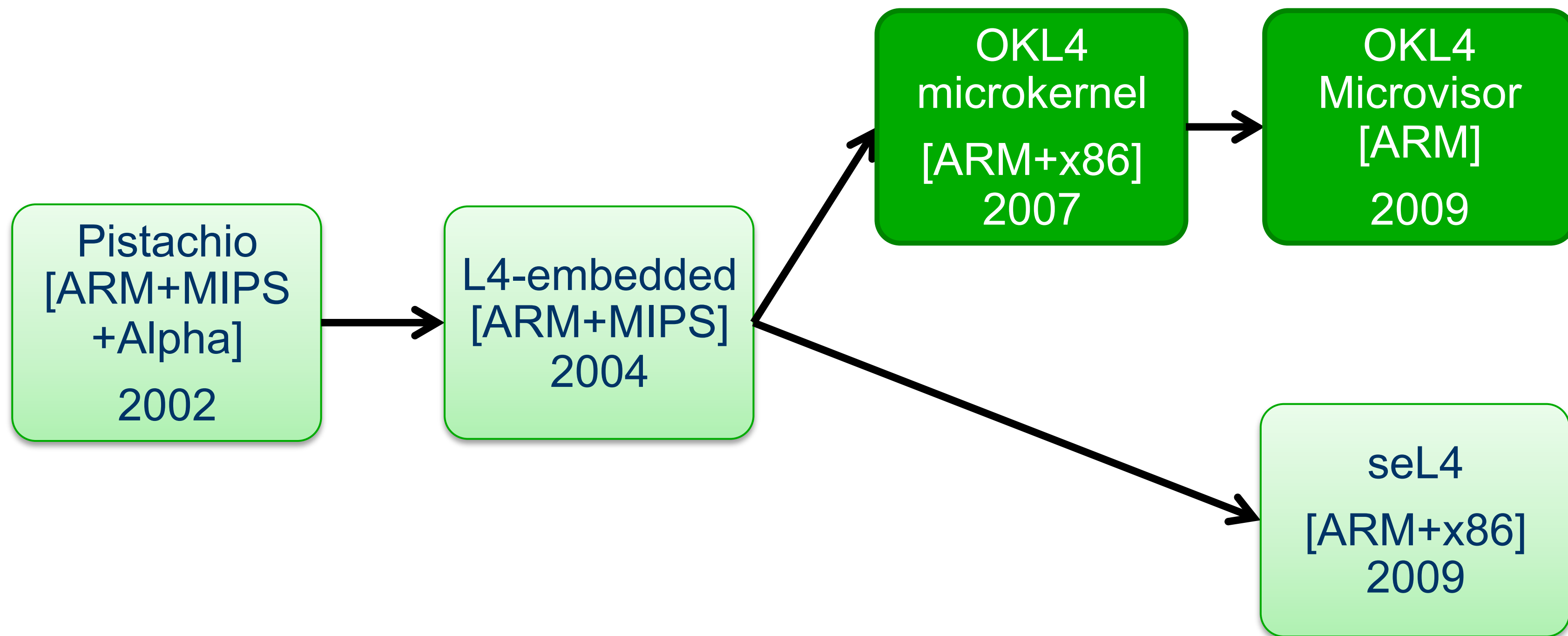
Australian Government

Department of Broadband, Communications and the Digital Economy

Australian Research Council

NICTA Funding and Supporting Members and Partners

L4 Made in Australia



Track Record of Innovation

L4-embedded:

- Fast context-switching on ARMv5
  - context switching without cache flush on virtually-addressed caches
  - 155-cycle IPC on XScale
  - virtualized Linux faster than native
- Event-based kernel (single kernel stack)
  - halved kernel memory use
- Removed IPC timeouts, “long” IPC
  - reduced kernel complexity
- Introduced asynchronous notifications

Track Record of Innovation

OKL4 microkernel:

- Dumped recursive address-space model
  - halved kernel memory use (again!)
  - reduced kernel complexity
- First L4 kernel with capability-based access control

Track Record of Innovation

OKL4 Microvisor:

To appear in cars, military phones

- Removed synchronous IPC
- Removed kernel-scheduled threads

Track Record of Innovation

seL4:

- All memory management at user level
  - no kernel heap!
- Formal proof of functional correctness
- Formal proof of integrity enforcement
- Sound worst-case execution-time model
- Performance on par with fastest kernels
  - <200-cy IPC on ARM11 without assembler fastpath!
- 100% microkernel: 9 kLOC
  - smaller than all others

World first!

World first!

World first!

World first!

What Mechanisms?

Hypervisor vs microkernel abstractions

Resource	OKL4 Microvisor	seL4 Microkernel
Memory	Virtual MMU (vMMU)	Address space
CPU	Virtual CPU (vCPU)	Thread or scheduler activation
Interrupt	Virtual IRQ (vIRQ)	IPC message
Communication	async Channel	Message-passing IPC
Synchronization	Virtual IRQ	IPC message

NICTA Vision: Trustworthy Systems

We will change the *practice* of designing and implementing critical systems, using rigorous approaches to achieve *true trustworthiness*

Suitable for real-world systems

Hard *guarantees* on safety/security/ reliability

©2012 Gernot Heiser NICTA

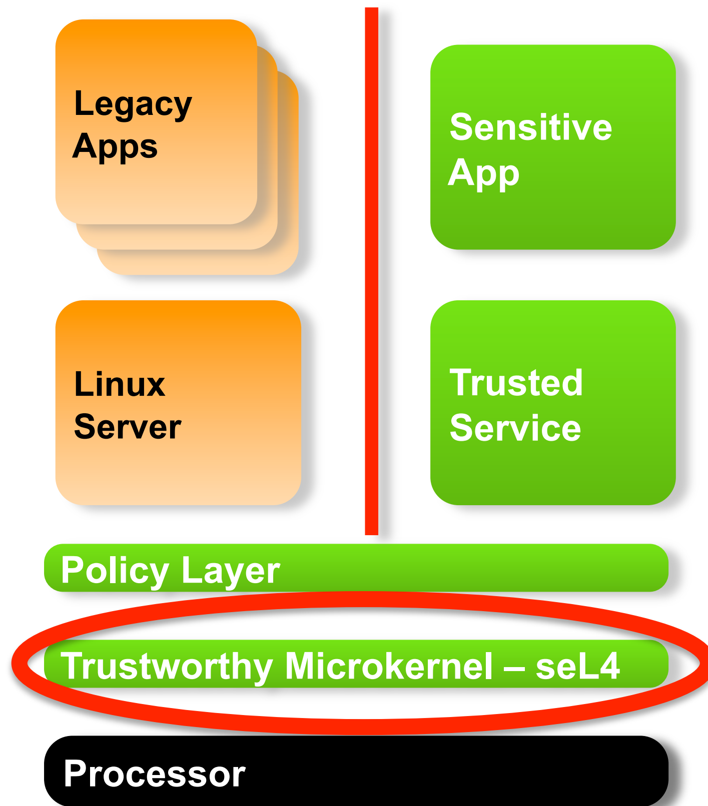
NICTA Trustworthy Systems Agenda

1. **Dependable microkernel (seL4) as a rock-solid base**
   - Formal specification of functionality
   - Proof of functional correctness of implementation
   - Proof of safety/security properties

2. **Lift microkernel guarantees to whole system**
   - Use kernel correctness and integrity to guarantee critical functionality
   - Ensure correctness of balance of trusted computing base
   - Prove dependability properties of complete system
     - despite 99 % of code untrusted!

©2012 Gernot Heiser NICTA

seL4 Design Goals



1. Isolation
   - Strong partitioning!
2. Formal verification
   - Provably trustworthy!
3. Performance
   - Suitable for real world!

Requirements for Trustworthy Systems

Safety

Security

Availability

Functional Correctness

Timeliness

Termination

Integrity

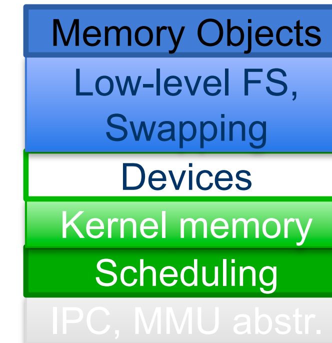
Confident. / Info Flow

Isolation!

Brief History of Microkernels

1st Generation: mid-1980 (Mach, Chorus etc)

- Stripped-down monolithic OSes
- Lots of functionality and policy
- Big
- Slow: 100 µs IPC

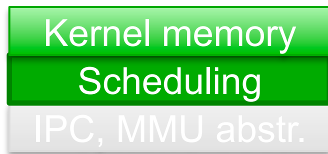


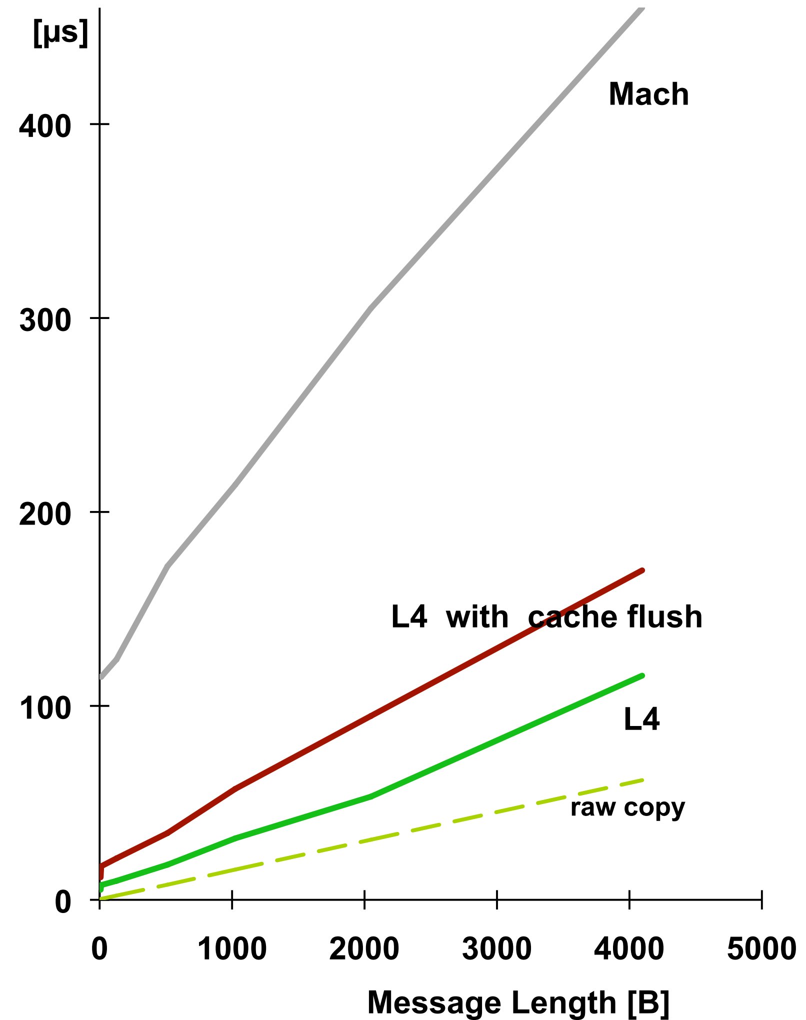
©2012 Gernot Heiser NICTA

Brief History of Microkernels

2nd Generation: mid-1990s – L4

- "Radical" approach [Liedtke'93, Liedtke '95]:
  - Strict minimality
  - From-scratch design and implementation
- Fast!

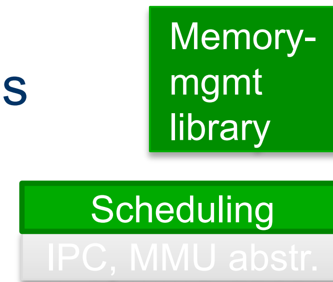




Brief History of Microkernels

3rd Generation: seL4 [Elphinstone et al 2007, Klein et al 2009]

- Security-oriented design
  - capability-based access control
  - strong isolation
- Hardware resources subject to user-defined policies
  - including kernel memory (*no kernel heap*)
  - except time ☹
- Designed for *formal verification*



©2012 Gernot Heiser NICTA

Issues of 2G L4 Kernels

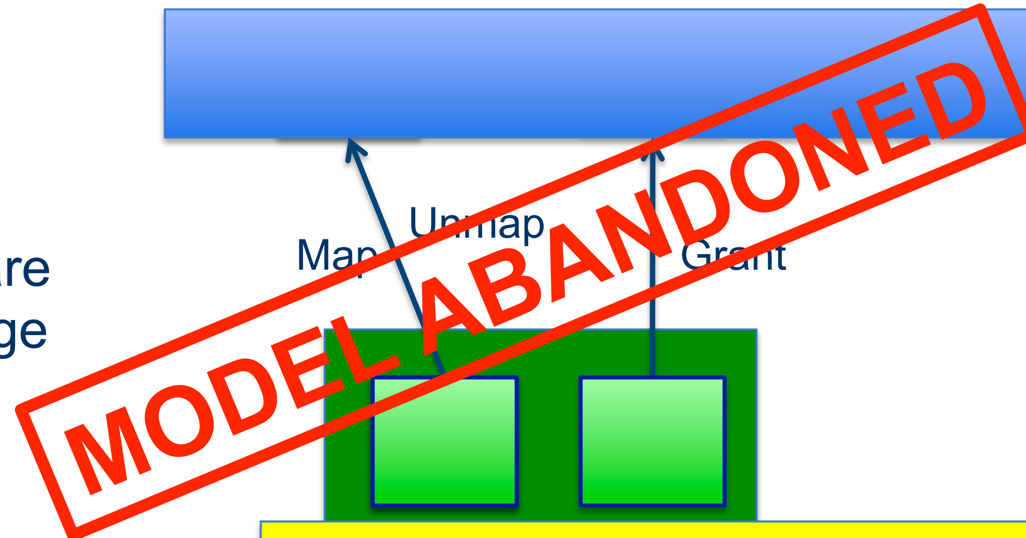
- L4 solved performance issue [Härtig et al, SOSP'97]
  … but left a number of security issues unsolved

- Problem: ad-hoc approach to protection and resource management
  - Global thread name space ⇒ covert channels
  - Threads as IPC targets ⇒ insufficient encapsulation
  - Single kernel memory pool ⇒ DoS attacks
  - Insufficient delegation of authority ⇒ limited flexibility, performance

Traditional L4: Recursive Address Spaces

- Mappings are page → page

- Magic initial address space to anchor recursion



Reasons:
- Complex & large mapping database
  - may account for 50% of memory use!
- Lack of control over resource use
  - implicit allocation of mapping nodes
- Potential covert channels

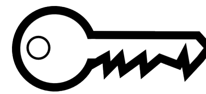
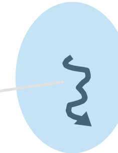
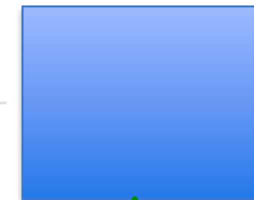
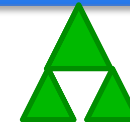
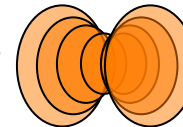
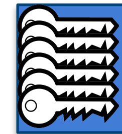
Physical Memory

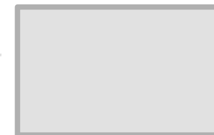
Fundamental Design Decisions for seL4

1. Memory management is user-level responsibility
   - Kernel never allocates memory (post-boot)
   - Kernel objects controlled by user-mode servers
2. Memory management is fully delegatable
   - Supports hierarchical system design
   - Enabled by *capability-based access control*
3. "Incremental consistency" design pattern
   - Fast transitions between consistent states
   - Restartable operations with progress guarantee
4. No concurrency in the kernel
   - Interrupts never enabled in kernel
   - Interruption points to bound latencies
   - Clustered multikernel design for multicores

seL4 Concepts

- Capabilities (Caps)
  - mediate access
- Kernel objects:
  - Threads (thread-control blocks, TCBs)
  - Address spaces (page table objects, PDs, PTs)
  - IPC endpoints (EPs, AsyncEPs)
  - Capability spaces (CNodes)
  - Frames
  - Interrupt objects
  - Untyped memory
- System calls
  - Send, Wait (and variants)
  - Yield

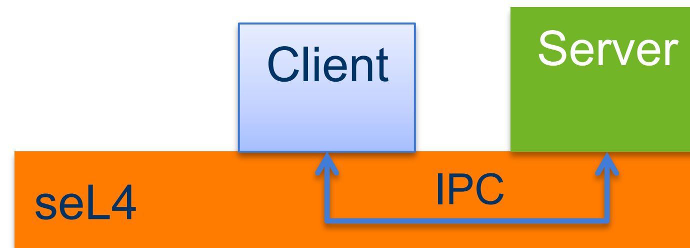









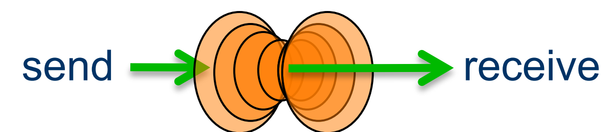
©2012 Gernot Heiser NICTA

Inter-Process Communication (IPC)

- Fundamental microkernel operation
  - Kernel provides no services, only mechanisms
  - OS services provided by (protected) user-level server processes
  - invoked by IPC



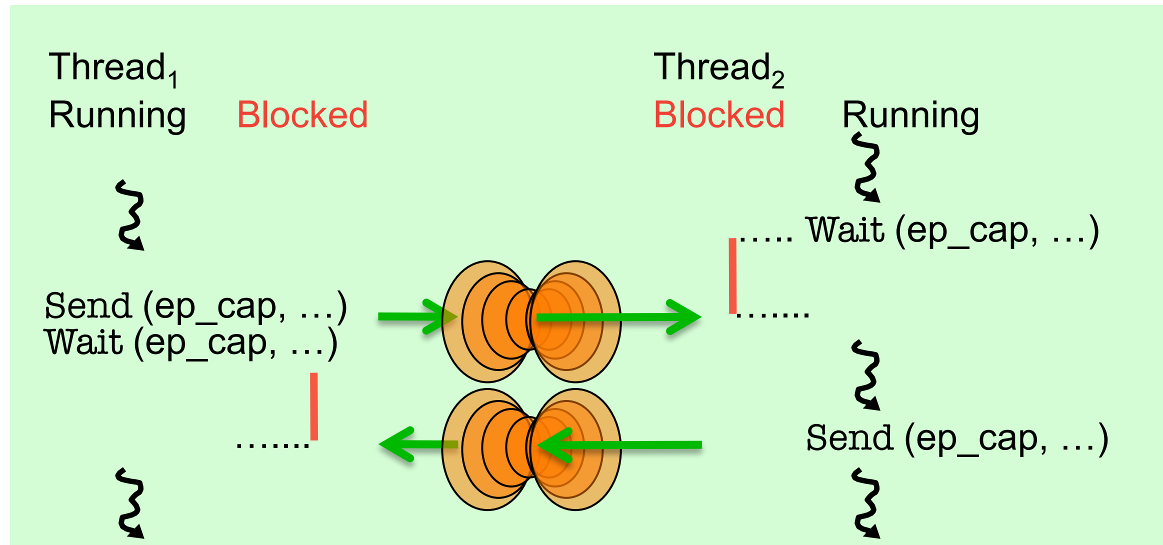
- seL4 IPC uses a handshake through *endpoints*:
  - Transfer points without storage capacity
  - Message must be transferred instantly
    - One partner may have to block
    - Single copy user → user by kernel



- Two endpoint types:
  - Synchronous (*Endpoint*) and asynchronous (*AsyncEP*)

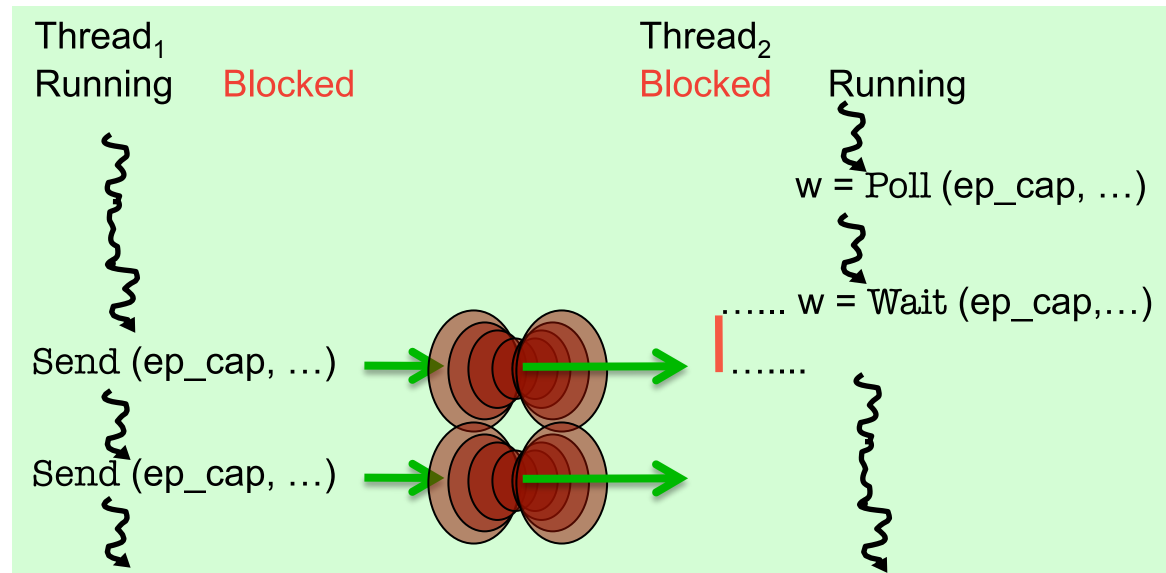
©2012 Gernot Heiser NICTA

Synchronous Endpoint



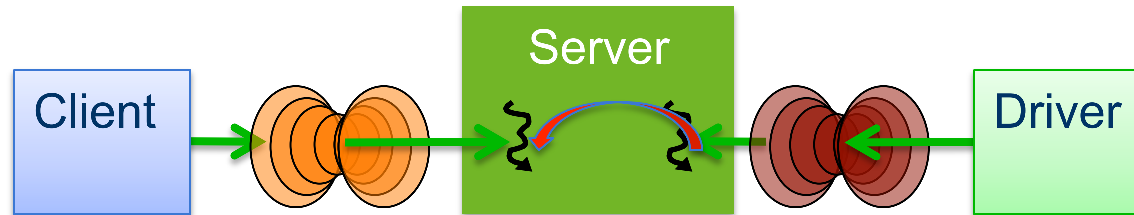
- Threads must rendez-vous for message transfer
  - One side blocks until the other is ready
- Message copied from sender's to receiver's message registers
  - Message is combination of caps and data words
    - presently max 121 words (484B, incl message "tag")

Asynchronous Endpoint



- Avoids blocking
  - send transmits 1-word message, OR-ed to receiver data word
  - no caps can be sent
- Receiver can poll or wait
  - waiting returns and clears data word
  - polling just returns data word
- Similar to interrupt (with small payload)

Receiving from Sync *and* Async Endpoints



Server with synchronous and asynchronous interface

- Example: file system
  - synchronous (RPC-style) client protocol
  - asynchronous notifications from driver
- Could have separate threads waiting on endpoints
  - forces multi-threaded server, concurrency control
- Alternative: allow single thread to wait on both EP types
  - Mechanism:
    - AsyncEP is *bound* to thread with `BindAEP()` syscall
    - thread waits on synchronous endpoint
    - async message delivered as if been waiting on AsyncEP

seL4 User-Level Memory Management

seL4 Memory Management Mechanics: Retype

Capability to "untyped"

Retype (Untyped, 2^1)

Retype (Frame, 2^2)

User memory

r,w r,w r,w r,w

Retype (Untyped, 2^1)

Thread control block

Retype (CNode, 2^m, 2^n)

Retype (TCB, 2^n)

Capability storage

F_0	F_1	F_2	F_3	…	…

Incremental Consistency

Avoids concurrency in (single-core) kernel

Disable interrupts

Kernel entry

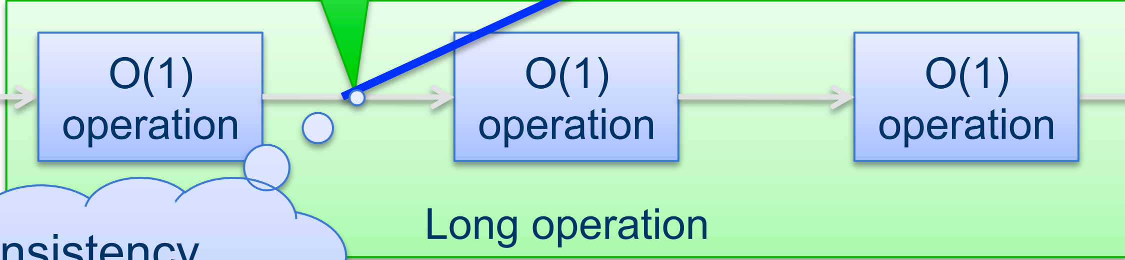
O(1) operation

Abort & restart later

Enable interrupts

Kernel exit

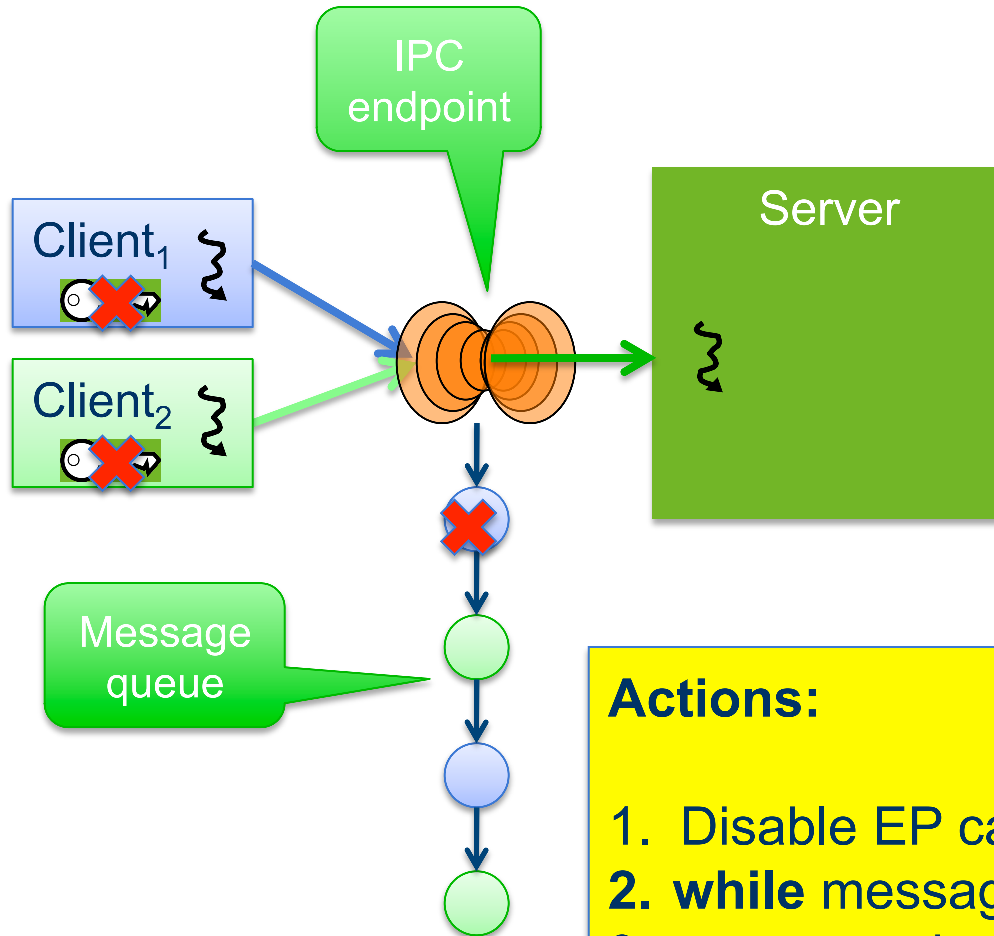
Check pending interrupts



- Consistency
- Restartability
- Progress

©2012 Gernot Heiser NICTA

Example: Destroying IPC Endpoint



Actions:

1. Disable EP cap (prevent new messages)
2. while message queue not empty **do**
3. remove head of queue (abort message)
4. check for pending interrupts
5. done

©2012 Gernot Heiser NICTA

Difficult Example: Revoking IPC "Badge"

Client$_1$

Client$_2$

Server

Client$_1$ state

Client$_2$ state

Badge

Removing orange badge

State to keep across preemption:
- Badge being removed
- Point in queue where preempted
- End of queue at time operation started
- Thread performing revocation

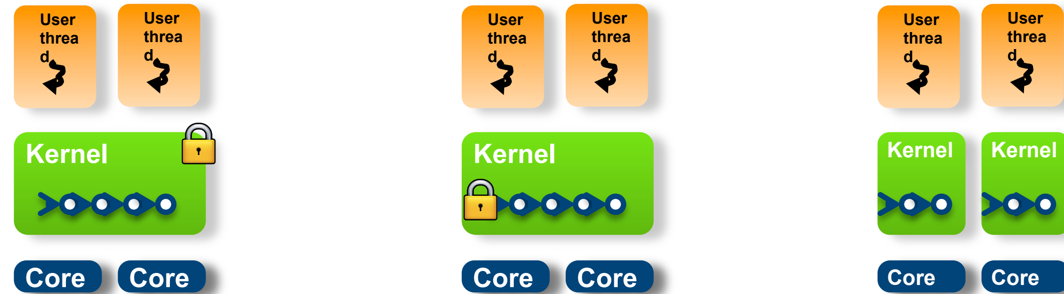
Need to squeeze into endpoint data structure!

Invariants to maintain!

Approaches for Multicore Kernels

SMP big lock	SMP fine-grained locks	Multikernel no locks
User thread, User thread	User thread, User thread	User thread, User thread
Kernel	Kernel	Kernel, Kernel
Core, Core	Core, Core	Core, Core

Multicore Kernel Trade-Offs



Property	Big Lock	Fine-grained Locking	Multikernel
Data structures	shared	shared	distributed
Scalability	poor	good	excellent
Concurrency in kernel	zero	high	zero
Kernel complexity	low	high	low
Resource management	centralised	centralised	distributed

Reality of Multicore is NUMA!

Fast communi-cation

Multi-threading

Core

HW context

HW context

L1 cache

Core

HW context

HW context

L1 cache

Core

HW context

HW context

L1 cache

L2 cache

L2 cache

L3 cache / Main memory

Slow communi-cation

Microkernel Principle: Policy Freedom

- Kernel must not dictate policy
- Kernel must not introduce avoidable overhead

Core
HW context
HW context
L1 cache
Core
HW context
HW context
L1 cache
Core
HW context
HW context
L1 cache
Core
HW context
HW context
L1 cache
L2 cache
L2 cache
L3 cache / Main memory

Share (SMP) where it is cheap!

Don't share (multikernel) where it is expensive!

Performance of Big Kernel Lock

©2012 Gernot Heiser NICTA

Resulting Design: Clustered Multikernel

SMP Linux

Virtu-al CPU | Virtu-al CPU | Virtu-al CPU | Virtu-al CPU | Virtu-al CPU | Virtu-al CPU | Virtu-al CPU | Virtu-al CPU

Kernel | Kernel

Core: HW context, HW context, L1 cache | Core: HW context, HW context, L1 cache | Core: HW context, HW context, L1 cache

L2 cache | L2 cache

L3 cache / Main memory

Still no concurrency in the kernel!

seL4 as Basis for Trustworthy Systems

Safety

Security

Availability

Functional
Correctness

Timeliness

Termination

Integrity

Confident. /
Info Flow

Proving Functional Correctness

Abstract Model

Proof — 117,000 lop

Executable Model

Proof — 50,000 lop

C Implementation

30–35 py
4.5 years

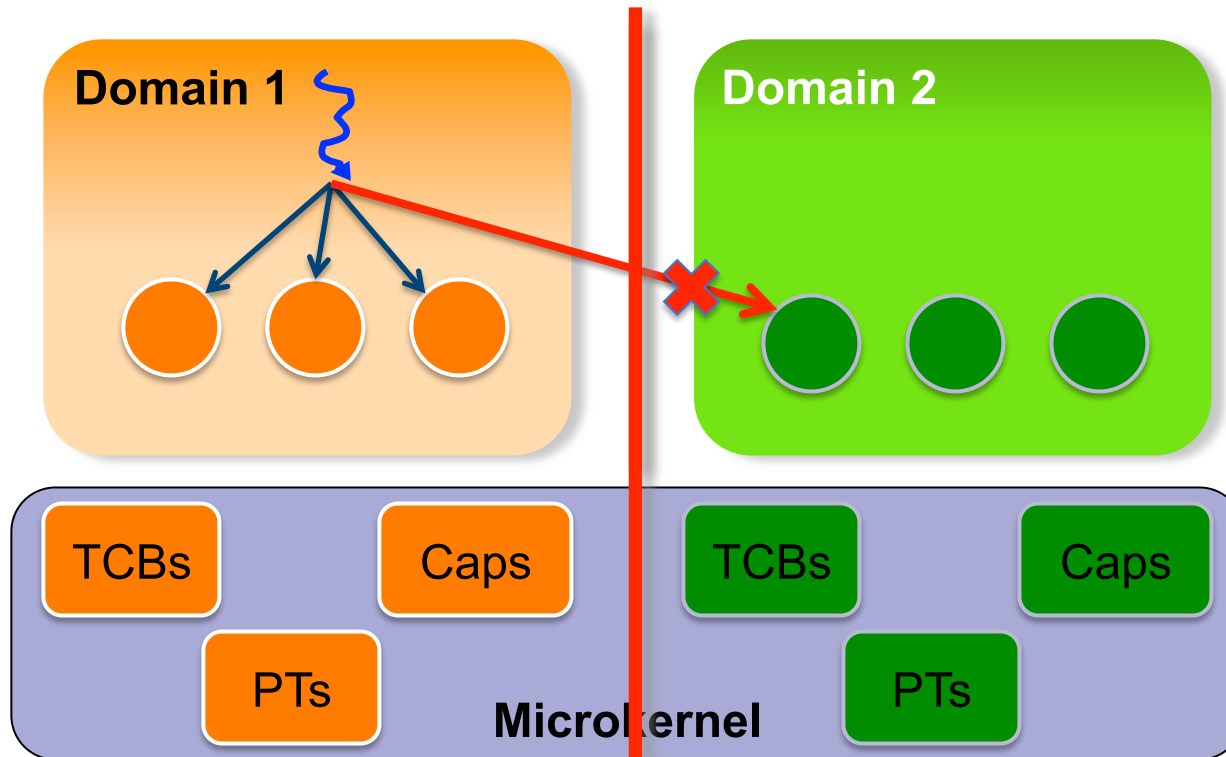
Refinement: All possible implementation behaviours are captured by model

Why So Long for 9,000 LOC?

seL4 call graph

seL4 as Basis for Trustworthy Systems

Integrity: Limiting Write Access



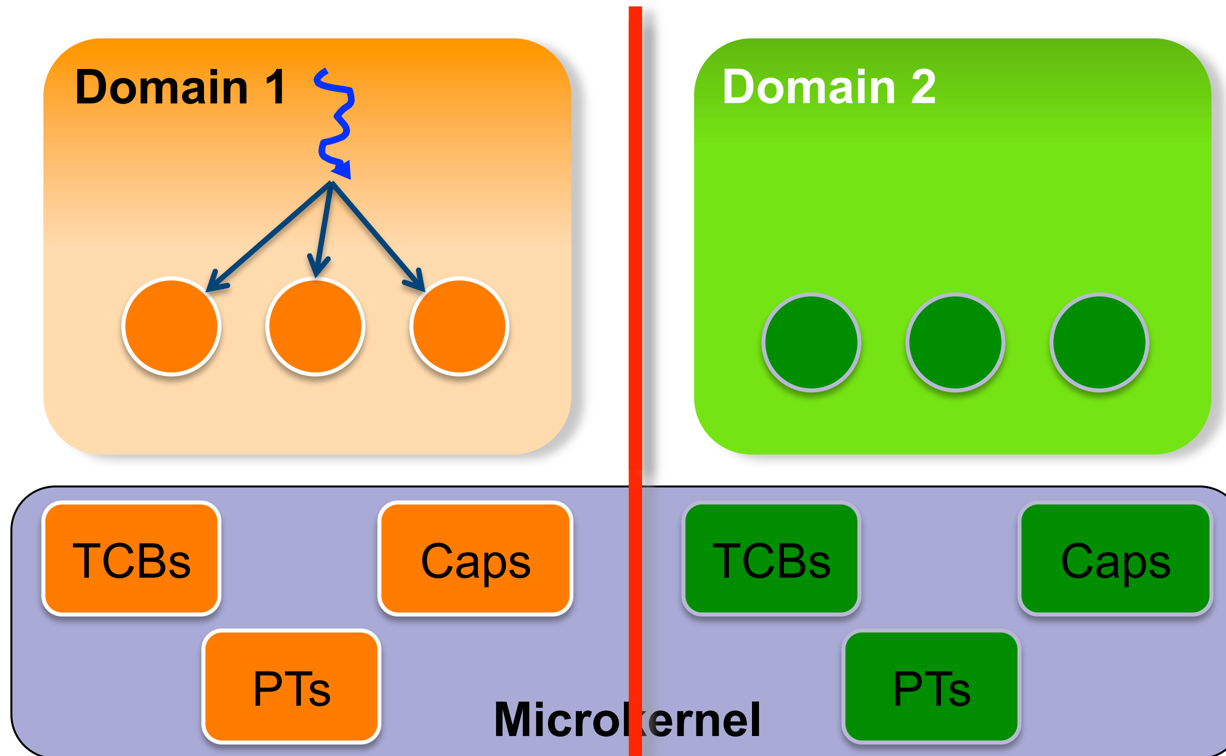
To prove:

- Domain-1 doesn't have write *capabilities* to Domain-2 objects
  ⇒ no action of Domain-1 agents will modify Domain-2 state
- Specifically, *kernel does not modify on Domain-1's behalf!*
  - Prove kernel only allows write upon capability presentation

seL4 as Basis for Trustworthy Systems

Safety
Security
Availability
Functional Correctness ✔
Timeliness
Confident. / Info Flow
Memory Safety ✔
Termination ✔
Integrity ✔

Availability: Ensuring Resource Access



- Strict separation of kernel resources
  ⇒ agent cannot deny access to another domain's resources

seL4 as Basis for Trustworthy Systems

Safety

Security

Availability ✔

Functional Correctness ✔

Timeliness

Confident. / Info Flow

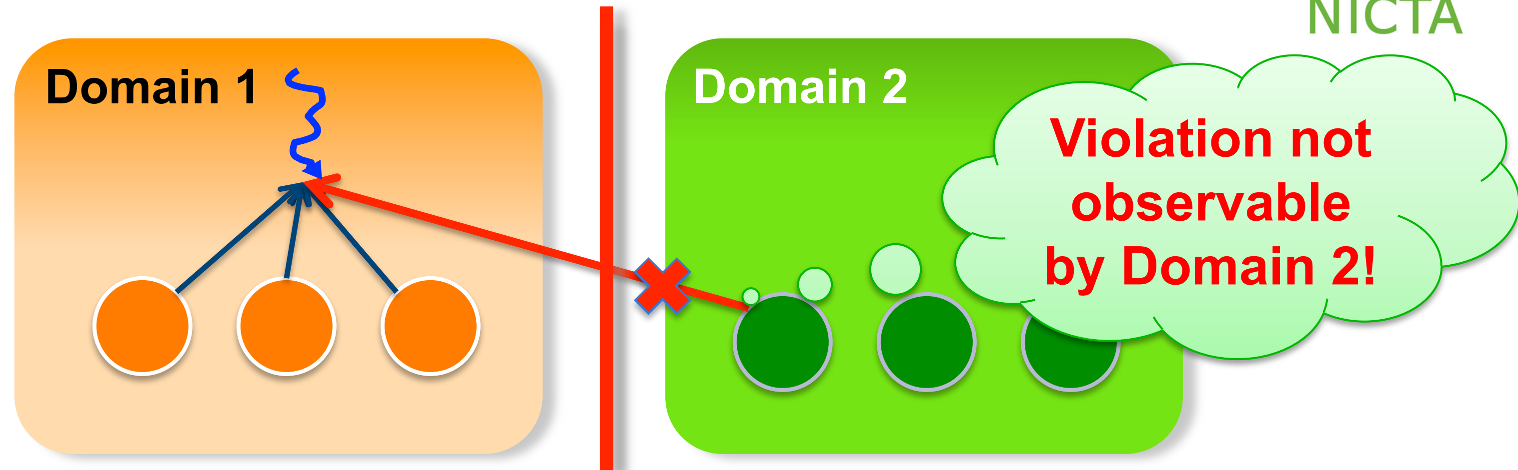
Memory Safety ✔

Termination ✔

Integrity ✔

©2012 Gernot Heiser NICTA

Confidentiality: Limiting Read Accesses



To prove:

- Domain-1 doesn't have read capabilities to Domain-2 objects
  ⇒ no action of any agents will reveal Domain-2 state to Domain-1

Non-interference proof in progress:

- Evolution of Domain 1 does not depend on Domain-2 state
- Presently cover only overt information flow

seL4 as Basis for Trustworthy Systems

Safety

Security

Availability ✔

Functional Correctness ✔

Timeliness

Confident. / Info Flow ✔

Memory Safety ✔

Termination ✔

Integrity ✔

Timeliness

Makes arbitrary system calls

Delivery with bounded latency

Domain 1

Domain 2

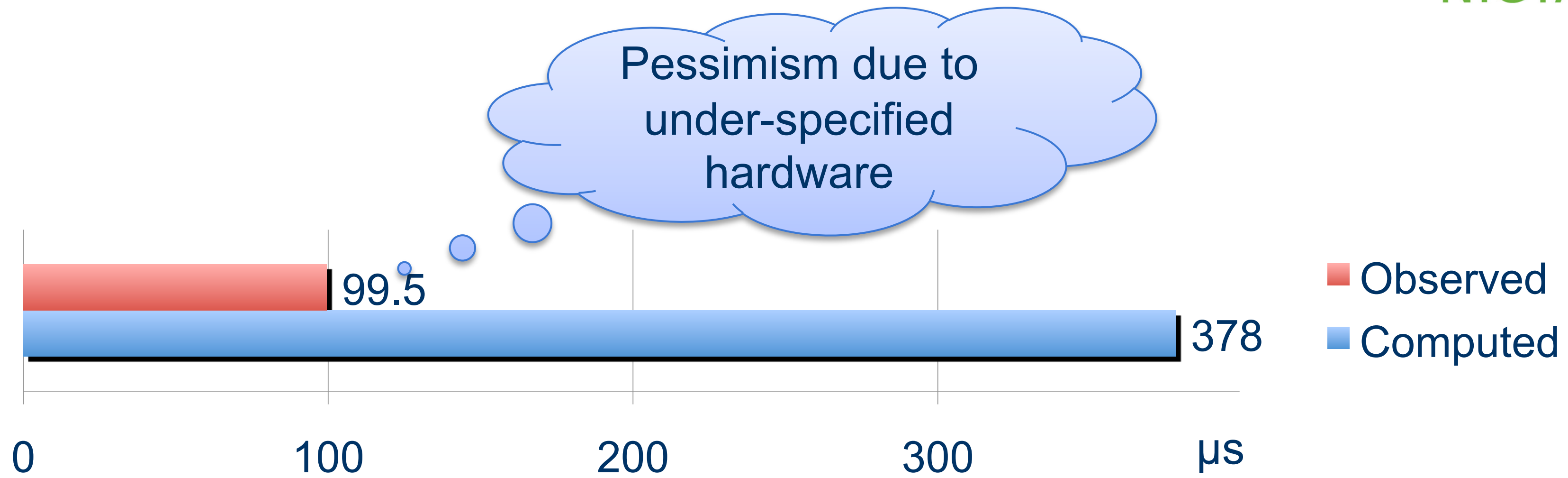
IRQ

Microkernel

Non-preemptible

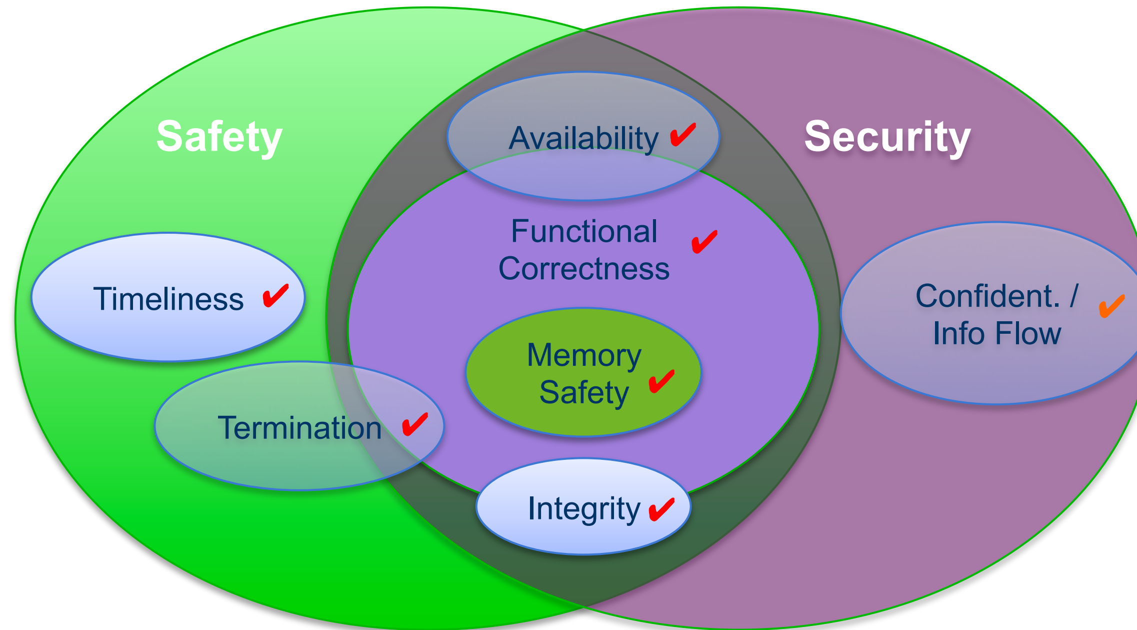
Need worst-case execution time (WCET) analysis of kernel

Result



WCET presently limited by verification practicalities
- **10 µs seem achievable**

Trustworthy Systems – seL4 is the Foundation!



Thank You!

mailto:gernot@nicta.com.au
Twitter: @GernotHeiser
Google: “nicta trustworthy systems”