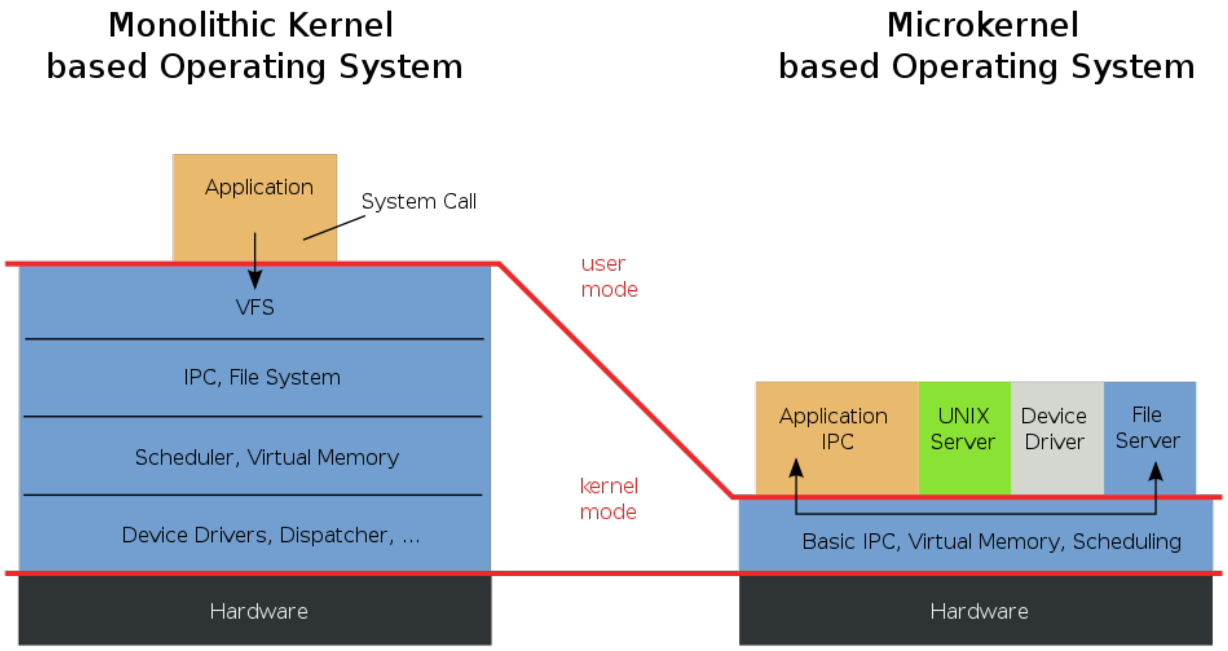
From L3 to seL4
What Have We Learnt in 20 Years of L4 Microkernels?

SOSP 2013

Contents

Introduction

- **What is microkernel?**
  - μ-kernel
  - Minimalist approach
  - Put the rest into user space(device driver, networking, etc…)


Monolithic Kernel based Operating System
Microkernel based Operating System
Application
System Call
VFS
IPC, File System
Scheduler, Virtual Memory
Device Drivers, Dispatcher, ...
Hardware
user mode
kernel mode
Application IPC
UNIX Server
Device Driver
File Server
Basic IPC, Virtual Memory, Scheduling
Hardware

< monolithic kernel vs. microkernel>

The L4 Microkernel Family

- **L4 microkernel**
  - a family of 2nd generation microkernels

- **“Original” version by Jochen Liedtke (93-95)**
  - “Version 2” API
  - i486/Pentium assembler
  - IPC 20 times faster than Mach microkernel

- **Other L4 V2 implementations**
  - L4/MIPS64: assembler + C (UNSW) (95-97)
  - L4/Alpha: PAL + C, First release SMP version(Dresden, UNSW), (95-97)
  - L4/Fiasco: C++(Dresden), fully preemptible (97-99)

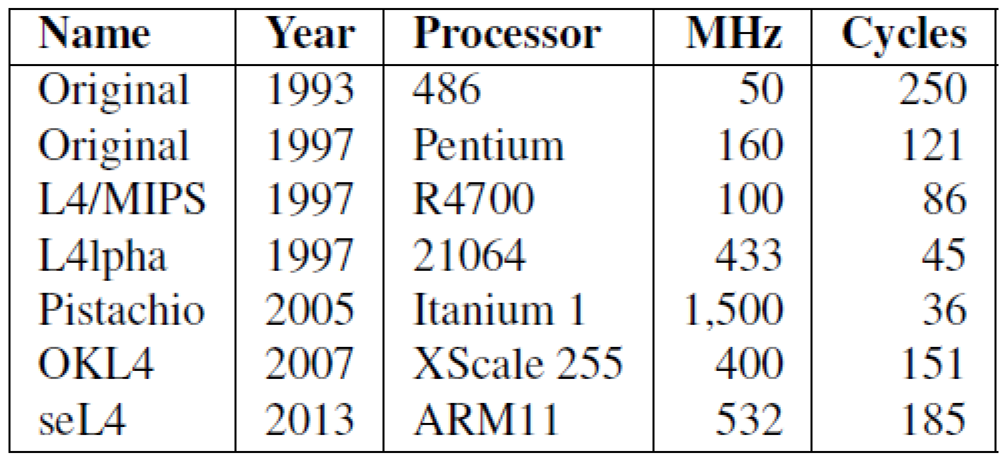
The L4 Microkernel Family

- **Experimental “Version X” API (X.1)**
  - Improved hardware abstraction
  - Various experimental features (performance, security, generality)
- **“Version 4” (X.2)**
  - Protability, API improvements
- **L4Ka::Pistachio**
  - C++ + assembler : "fast path”
  - x86, PPC32, Itanium (NICTA, UNSW) (02-03)
  - MIPS64, Alpha (NICTA, UNSW) (03)
  - ARM, PPC64 (NICTA, UNSW), x86-64(Karlsruhe), (03-04)

The L4 Microkernel Family

- **OKL4(Open Kernel Labs) (08)**
  - capability-based access control
  - OKL4 Microvisor (virtualization) (2010)

- **seL4 (Current)**
  - new L4 kernel (3rd generation microkernel)
  - for highly secure and reliable systems

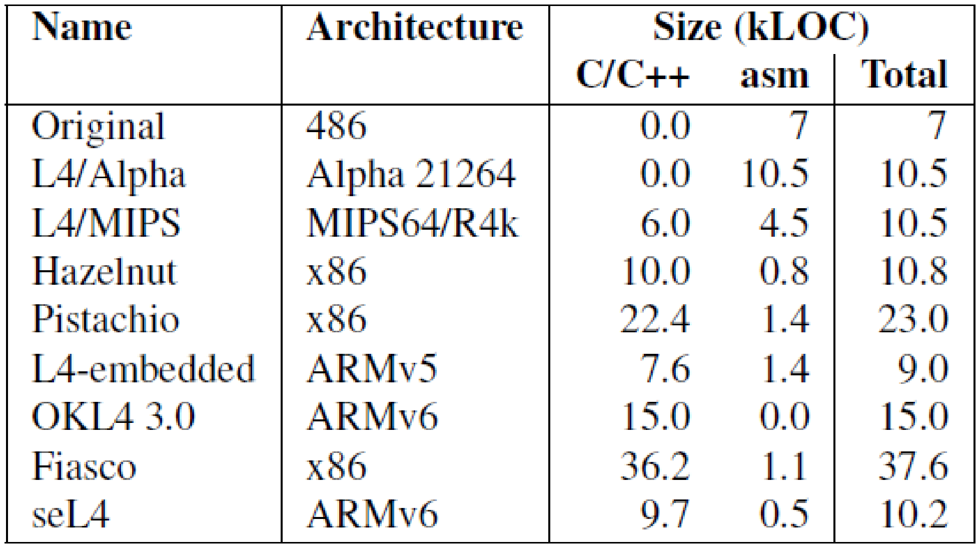


Name	Year	Processor	MHz	Cycles
Original	1993	486	50	250
Original	1997	Pentium	160	121
L4/MIPS	1997	R4700	100	86
L4lpha	1997	21064	433	45
Pistachio	2005	Itanium 1	1,500	36
OKL4	2007	XScale 255	400	151
seL4	2013	ARM11	532	185

Table 1. One-way IPC cost of various L4 kernels.

Principles and concepts

- **Minimality**
  - Liedtke: “only minimal mechanisms and no policy in the kernel”



Name	Architecture	Size (kLOC)		
		C/C++	asm	Total
Original	486	0.0	7	7
L4/Alpha	Alpha 21264	0.0	10.5	10.5
L4/MIPS	MIPS64/R4k	6.0	4.5	10.5
Hazelnut	x86	10.0	0.8	10.8
Pistachio	x86	22.4	1.4	23.0
L4-embedded	ARMv5	7.6	1.4	9.0
OKL4 3.0	ARMv6	15.0	0.0	15.0
Fiasco	x86	36.2	1.1	37.6
seL4	ARMv6	9.7	0.5	10.2

Table 2. Source lines of code (SLOC) of various L4 kernels. The size of the original kernel is an estimate derived from the 12 KiB binary size [Liedtke 1996b].

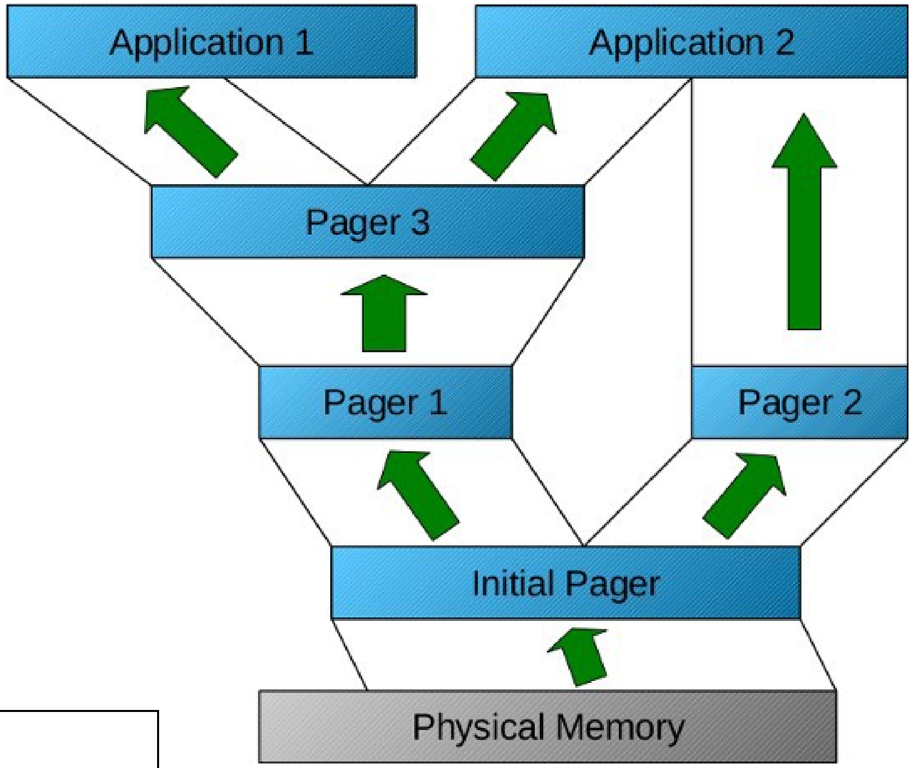
Retained: Minimality is still the key design principle.

Principles and concepts

- **Recursive address spaces**
  - 3 management operations
    - Map/Unmap
    - Grant
    - Flush
    - significant cost in terms of kernel
  - complexity & memory overhead

  - “mapping database” (NICTA)

Abandoned: Recursive address spaces.

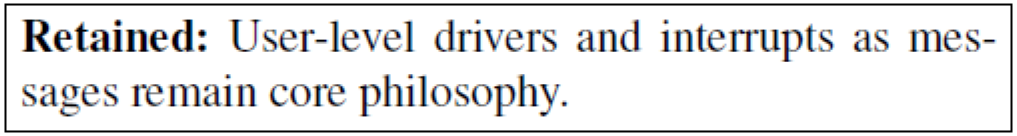


< recursive address spaces >

Principles and concepts

- **User-level device drivers and interrupts as IPC**
  - most radical novelty of L4
  - a single driver in the kernel : timer driver
  - in user mode : all other device drivers

  - sending interrupts from kernel to drivers : IPC messages



Retained: User-level drivers and interrupts as messages remain core philosophy.

Principles and concepts

- **Threads as IPC destinations**
  - poor information hiding
  - IPC endpoint and TCB(Thread Control Block)

Replaced: Thread IDs by port-like IPC endpoints as message destinations.

- **Synchronous IPC and long messages**
  - only synchronous IPC (blocking)
  - “long” IPC messages
  - a page fault during copying messages(user-level page-fault handling)
  - asynchronous notification(using bit masking)

Abandoned: “Long” IPC.

Added: Asynchronous notifications.

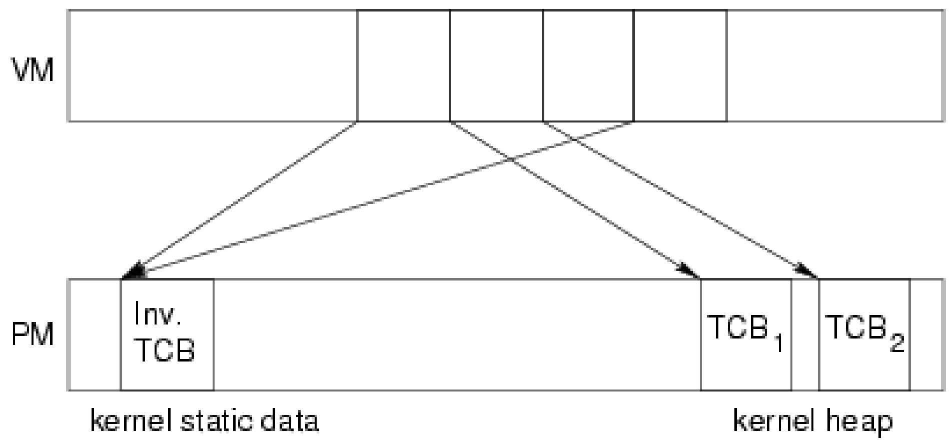
Principles and concepts

- **Hierarchical task management and communication control**
  - a process hierarchy : a set of task IDs
  - sending IPC message : only siblings or the parent (clans-and-chiefs model)
  - a significant overhead

Abandoned: Clans-and-chiefs.

Design and implementation tricks

- **Strict process orientation and virtual TCB array**
  - virtual TCB array for fast lookup from thread ID



  - cost : large VM consumption, increase TLB pressure
  - No performance benefit on modern hardware

Abandoned: Process kernel and virtual TCB addressing.

Design and implementation tricks

- **IPC timeouts**
  - to protect against denial of service
  - significant complexity
  - timeouts were of little use

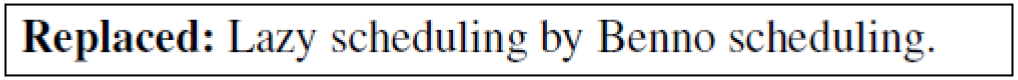
  - Replacement : a choice of polling or blocking using a single flag
  - only two flags : for the send and receive phase

Abandoned: Timeouts.

Design and implementation tricks

- **Lazy scheduling**
  - Frequent IPC : frequently blocking/unblocking
  - lots of run-queue manipulation

  - Replacement: “Benno scheduling”
  - every thread on the run queue : runnable!
  - context switches due to IPC involve no run-queue manipulation

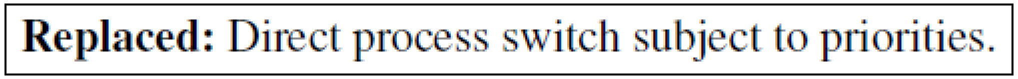

Replaced: Lazy scheduling by Benno scheduling.

Design and implementation tricks

- **Direct process switch**
  - to avoid running the scheduling during IPC

  - Replacement : direct process switch
  - Process Switch
    - thread block during IPC -> readily-identifiable runnable thread
    - ignore priorities

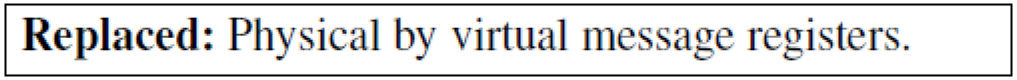
  - Modern L4 versions
    - run direct-process switch where it conforms with priorities



Replaced: Direct process switch subject to priorities.

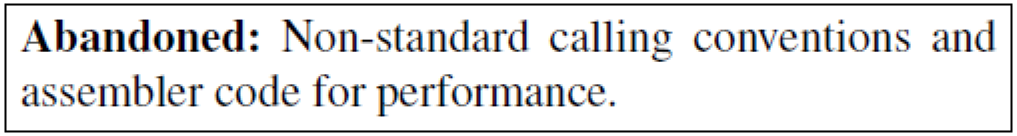
Design and implementation tricks

- **Register messages**
  - highly dependent on the architecture
  - Replacement : set of virtual message registers
  - map to physical registers & pin user-level TCB

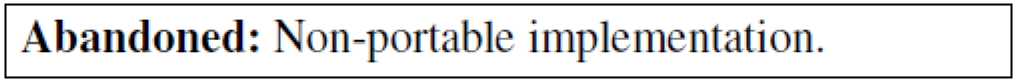

Replaced: Physical by virtual message registers.

- **Non-standard calling convention**

- **Non-portabili**


Abandoned: Non-standard calling conventions and assembler code for performance.

- **Is it still L4?**


Abandoned: Non-portable implementation.

seL4 Design

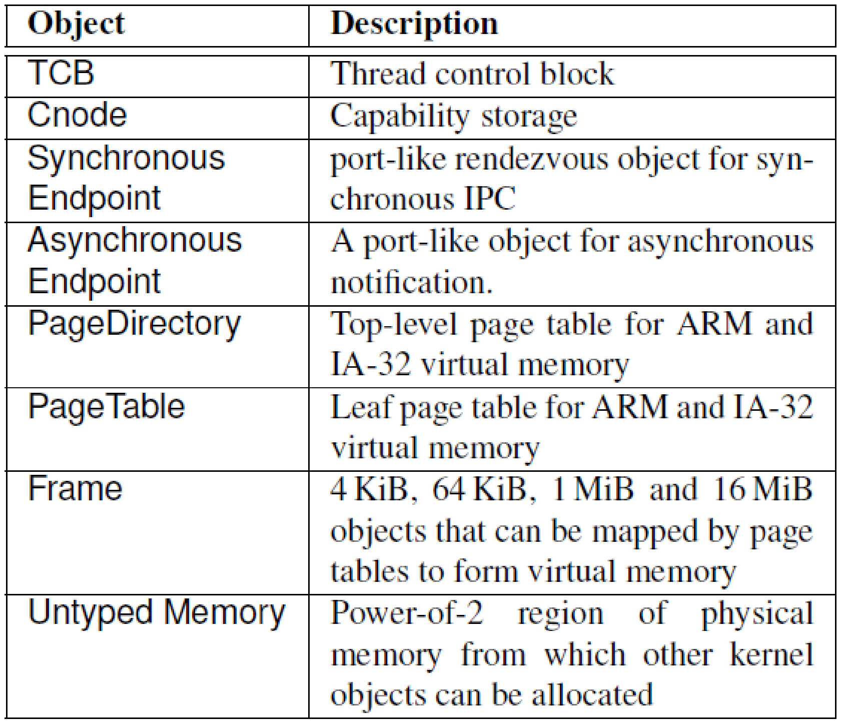
◆**security and safety**

1. All authority is explicitly conferred (via capabilities).
2. Data access and authority can be confined.
3. The kernel itself (for its own data structures) adheres to the authority distributed to applications, including theconsumption of physical memory.
4. All kernel objects can be reclaimed independent of any other kernel objects.
5. All operations are “short” in execution time, or are preemptible in short time.
6. Performance is not significantly worse than the fastest L4 kernels (say within 10%).

seL4 Design

- **Security Focus(Requirements 1. and 2.)**
  - Capability Derivation Tree(CDT)

- **Memroy Management Approach**
- all in-kernel allocated objects
  - first-class objects in the ABI
  - no-change their size after creation

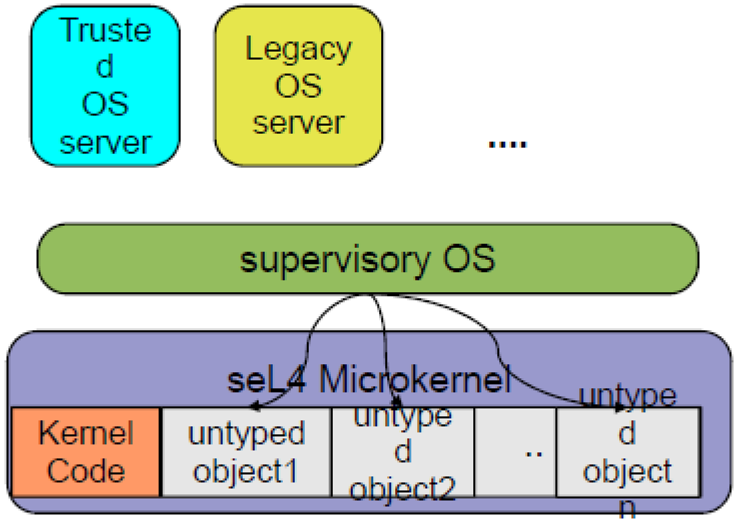


Object	**Description**
TCB	Thread control block
Cnode	Capability storage
Synchronous Endpoint	port-like rendezvous object for synchronous IPC
Asynchronous Endpoint	A port-like object for asynchronous notification.
PageDirectory	Top-level page table for ARM and IA-32 virtual memory
PageTable	Leaf page table for ARM and IA-32 virtual memory
Frame	4 KiB, 64 KiB, 1 MiB and 16 MiB objects that can be mapped by page tables to form virtual memory
Untyped Memory	Power-of-2 region of physical memory from which other kernel objects can be allocated

Table 3. seL4 kernel objects.

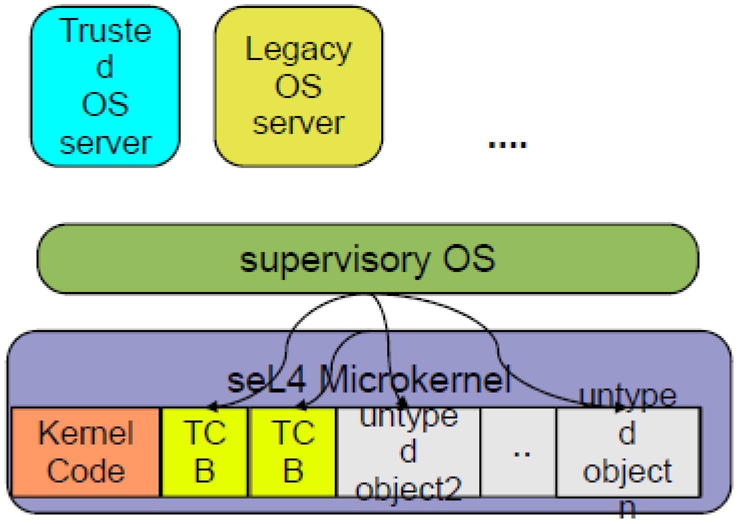
seL4 Design

- **Memory Management Model(allocation)**
  - Untyped Memory(UM) objects
  - UM capability : the authority to a region of memory
  - use to create typed memory
  - retype() method



seL4 Design

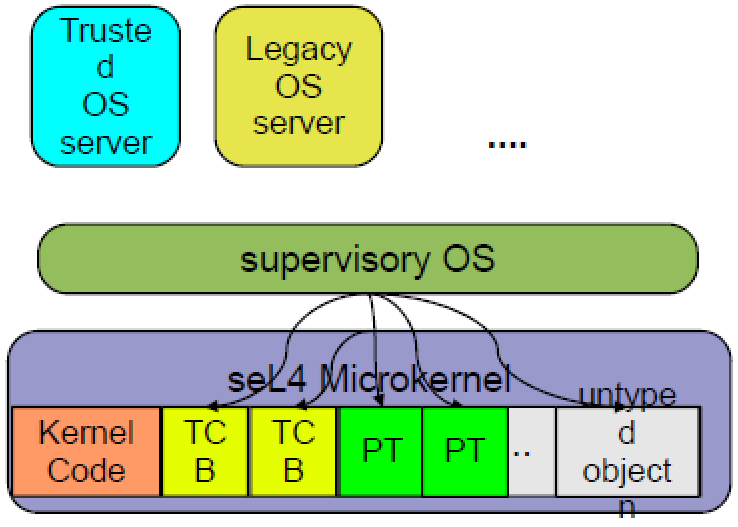
- **Memory Management Model(allocation)**
  - Untyped Memory(UM) objects
  - UM capability : the authority to a region of memory
  - use to create typed memory
  - retype() method



- Kernel objects
  - Untyped
  - TCB (Thread Control Blocks)
  - Capability tables (CT)
  - Comm. ports ....

seL4 Design

- **Memory Management Model(allocation)**
  - Untyped Memory(UM) objects
  - UM capability : the authority to a region of memory
  - use to create typed memory
  - retype() method



- Kernel objects
  - Untyped
  - TCB (Thread Control Blocks)
  - Capability tables (CT)
  - Comm. ports ....
- Objects are managed by user-level

seL4 Design

- **Memory Management Model(allocation)**
  - Untyped Memory(UM) objects
  - UM capability : the authority to a region of memory
  - use to create typed memory
  - retype() method

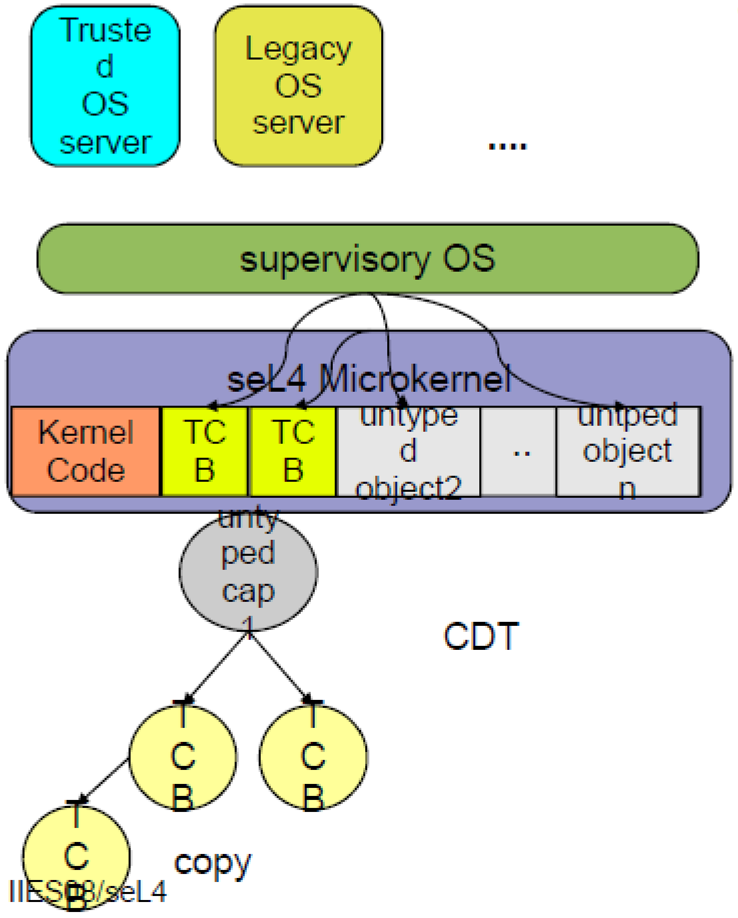
  - Delegate authority
  - Memory management policy
  - is completely in user-space

Isolation of physical memory
  - = Isolation of authority(capabilities)

seL4 Design

- **Memory Management Model(de-allocation)**
  - using Capability Derivation Tree
  - revoke() method
  - remove any in-kernel dependencies
  - preemptible
  - (revocation = long running operation)

  - re-use condition
    - should not have any CDT children
    - size of the object <= untyped object



seL4 Design

- **Object Independence**
  - facilitation of coupling and decoupling objects
  - three scenarios
  - Objects may refer to each other with internal pointers.
    1. : Endpoint
    2. Objects contain capabilities to other objects.
    3. : Automatically decoupling objects
    4. The capability contains the book-keeping data.
  - facilitation of coupling and decoupling objects

- **Preemption**
  - object initialization
  - revocation of capabilities
  - decoupling of objects from reclaimed objects

incrementally consistent

seL4 Design

- **Notifications**
  - Allow single thread to wait on both Sync and Async Endpoint types

  - Mechanism
    - Async Endpoint is bound to thread with BindAEP() syscall
    - Thread waits on Sync endpoint
    - Async message delivered as if been waiting on Async Endpoint