Secure Microkernel for Deeply Embedded Devices

Jim Huang (黃敬群)*, Louie Lu (呂紹榕)

*National Cheng Kung University, Taiwan

Feb 4, 2017 / FOSDEM

NCKU
National Cheng Kung University

Secure Microkernel for **Deeply Embedded Devices**

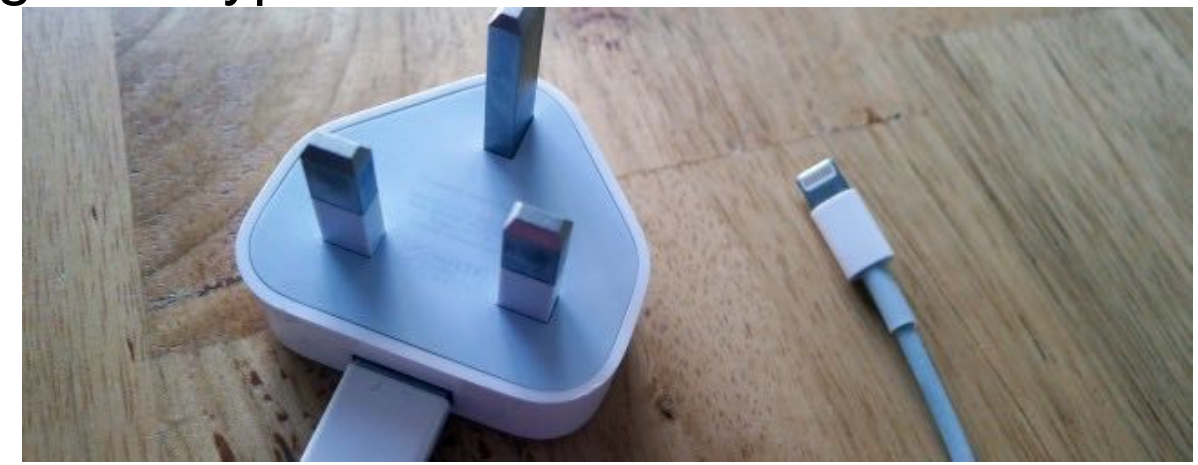
- The promise of the IoT won't be fulfilled until integrated software platforms are available that allow software developers to develop these devices efficiently and in the most cost-effective manner possible.
- F9 microkernel, new open source and secure implementation built from scratch, which deploys modern kernel techniques dedicated to deeply embedded devices.
- Characteristics of F9 microkernel / BitSec
  - Efficiency: performance + power consumption
  - Security: memory protection + isolated execution
  - Flexible development environment

“Security is not
a product, but a process”
- Bruce Schneier

(American cryptographer, computer security and privacy specialist)

Case Study: Attack iOS through USB charger!

- BlackHat 2013
  - MACTANS: INJECTING MALWARE INTO IOS DEVICES VIA MALICIOUS CHARGERS
- "we demonstrate how an iOS device can be compromised within one minute of being plugged into a malicious charger. We first examine Apple's existing security mechanisms to protect against arbitrary software installation, then describe how USB capabilities can be leveraged to bypass these defense mechanisms."



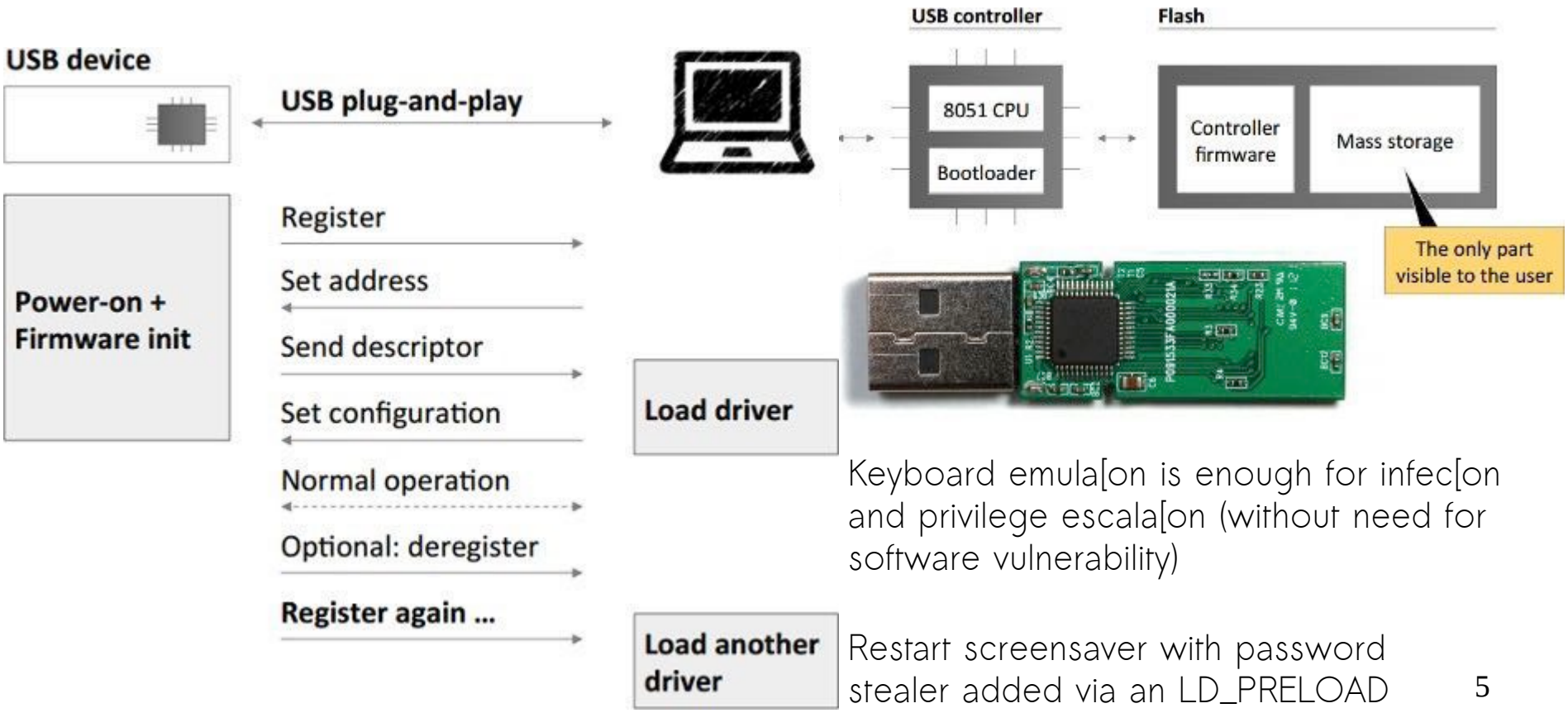
Source:http://www.blackhat.com/us-13/briefings.html#Lau

Case Study: BadUSB

- BlackHat 2014

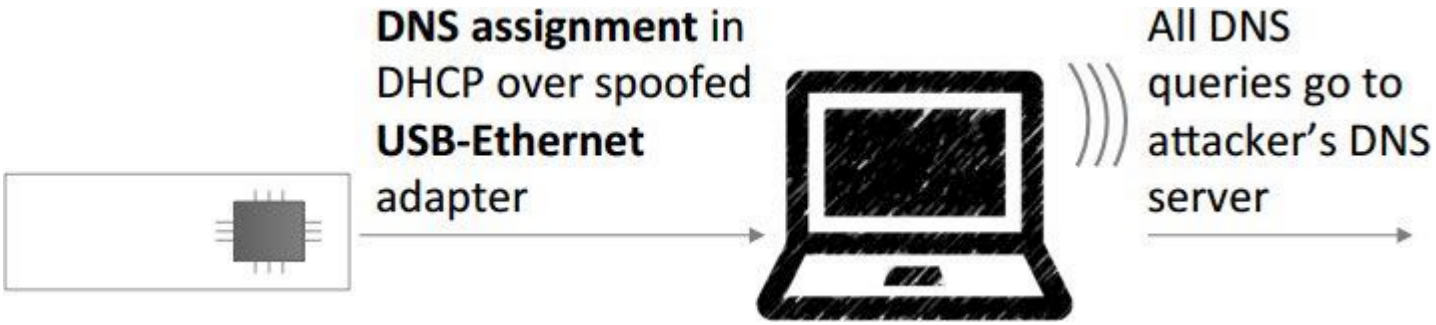
  BadUSB — On accessories that turn evil

  https://srlabs.de/badusb/



Keyboard emula[on is enough for infec[on and privilege escala[on (without need for software vulnerability)

Restart screensaver with password stealer added via an LD_PRELOAD library



- USB Redirection via RDP

  Easy Print / Drive Redirection / Smart Card Redirection

  Plug-and-Play Device Redirection / Input Redirection / Audio Redirection / Port Redirection

Source:USB attacks need physical access right? Andy Davis

Users Really Do Plug in USB Drives They Find

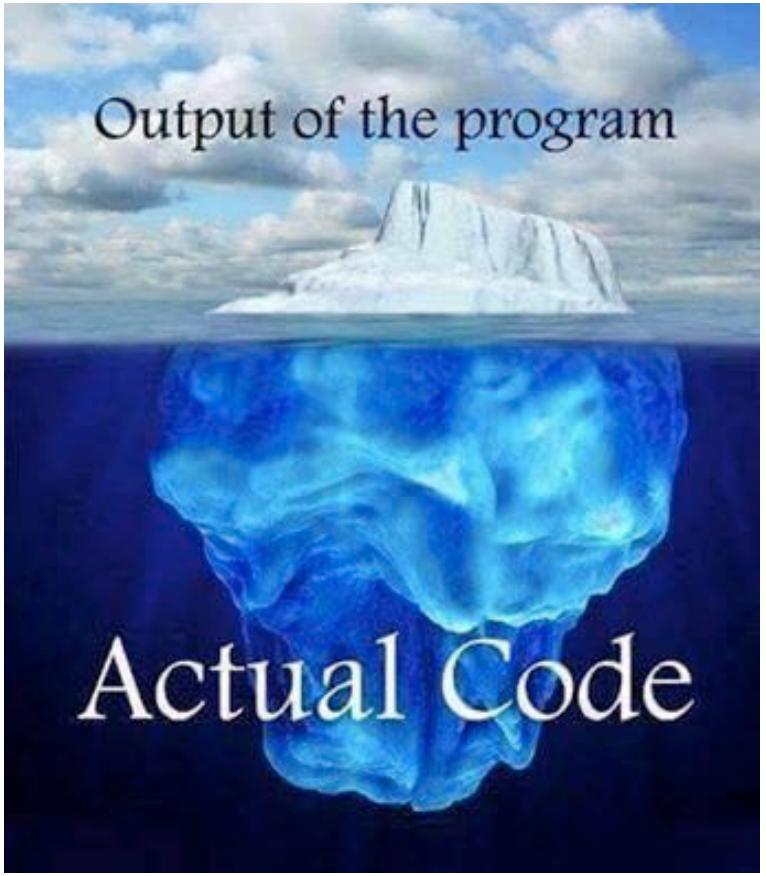
- "end users will pick up and plug in USB flash drives they find by completing a controlled experiment in which we drop 297 flash drives on a large university campus."
- "We find that the attack is effective with an estimated success rate of **45–98%** and expeditious with the first drive connected in less than six minutes."
- Researchers at University of Illinois, Urbana Champaign, University of Michigan, Google, Inc.

Related talk at OpenIoT 2016
Handling Top Security Threats for Connected Embedded Devices - Eystein Stenberg, Mender

Source:
https://zakird.com/papers/usb.pdf

TrustZone for ARMv8-M

- Enablesbuslevelprotectioninhardware
  - ARMv7-M requires software API filters for DMA access and other security critical operations
  - ARMv8-M can filter for DMA access for requests initiated by unprivileged code on bus level
- MPU banking reduces complexity of secure target OS
  - Secure OS partition own a private MPU with full control
  - OS keeps the privileged mode for fast IRQs
  - Fast interrupt routing and register clearing in hardware
  - Fast cross-box calls on TrustZone for ARMv8M – optimized call gateways

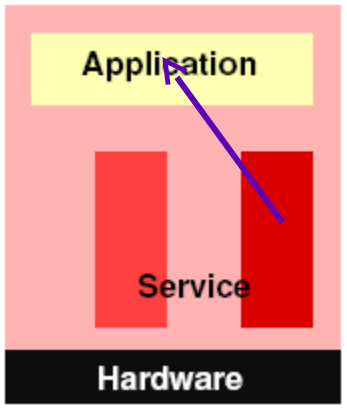


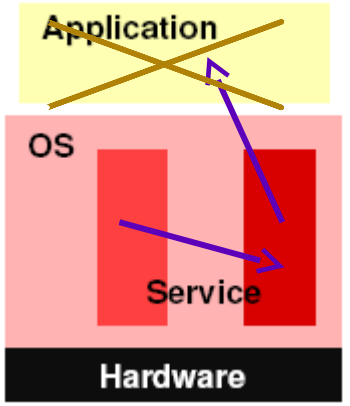
source: https://twitter.com/manisha72617183/status/ 816329693494317056

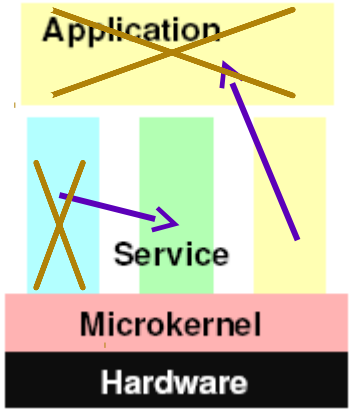
Wait!

Why do we need yet another kernel?

TCB (Trusted Computing Base)







System	traditional embedded	Linux/ Windows	Microkernel based
TCB	all code	100,000 LoC	10,000 LoC

source: Diagram from Kashin Lin (NEWS Lab)

Bugs inside “Bigger than Bigger” Kernels

- Drivers cause 85% of Windows XP crashes.
  - Michael M. Swift, Brian N. Bershad, Henry M. Levy: “Improving the Reliability of Commodity Operating Systems”, SOSP 2003
- Error rate in Linux drivers is 3x (maximum: 10x)
  - Andy Chou, Junfeng Yang, Benjamin Chelf, Seth Hallem, Dawson R. Engler: “An Empirical Study of Operating System Errors”, SOSP 2001
- Causes for driver bugs
  - 23% programming error
  - 38% mismatch regarding device specification
  - 39% OS-driver-interface misconceptions
  - Leonid Ryzhyk, Peter Chubb, Ihor Kuz and Gernot Heiser: “Dingo: Taming device drivers”, EuroSys 2009

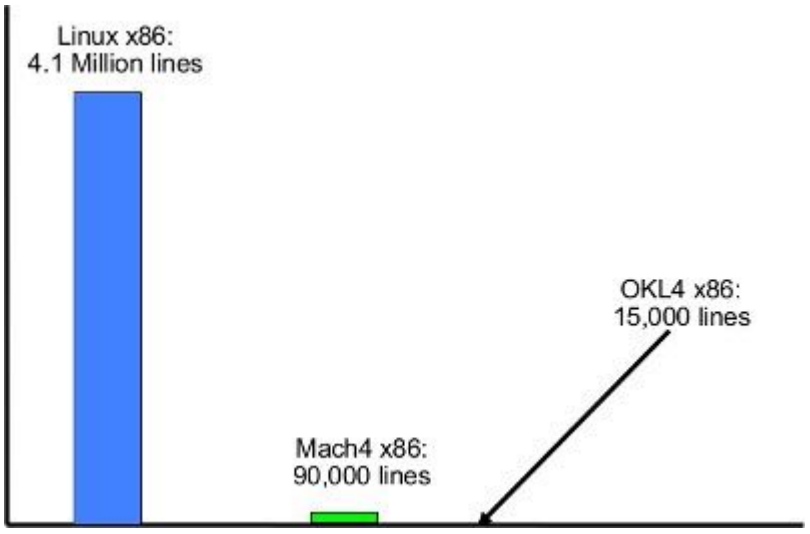
Linux Device Driver bugs

[Dingo: Taming device drivers, 2009]

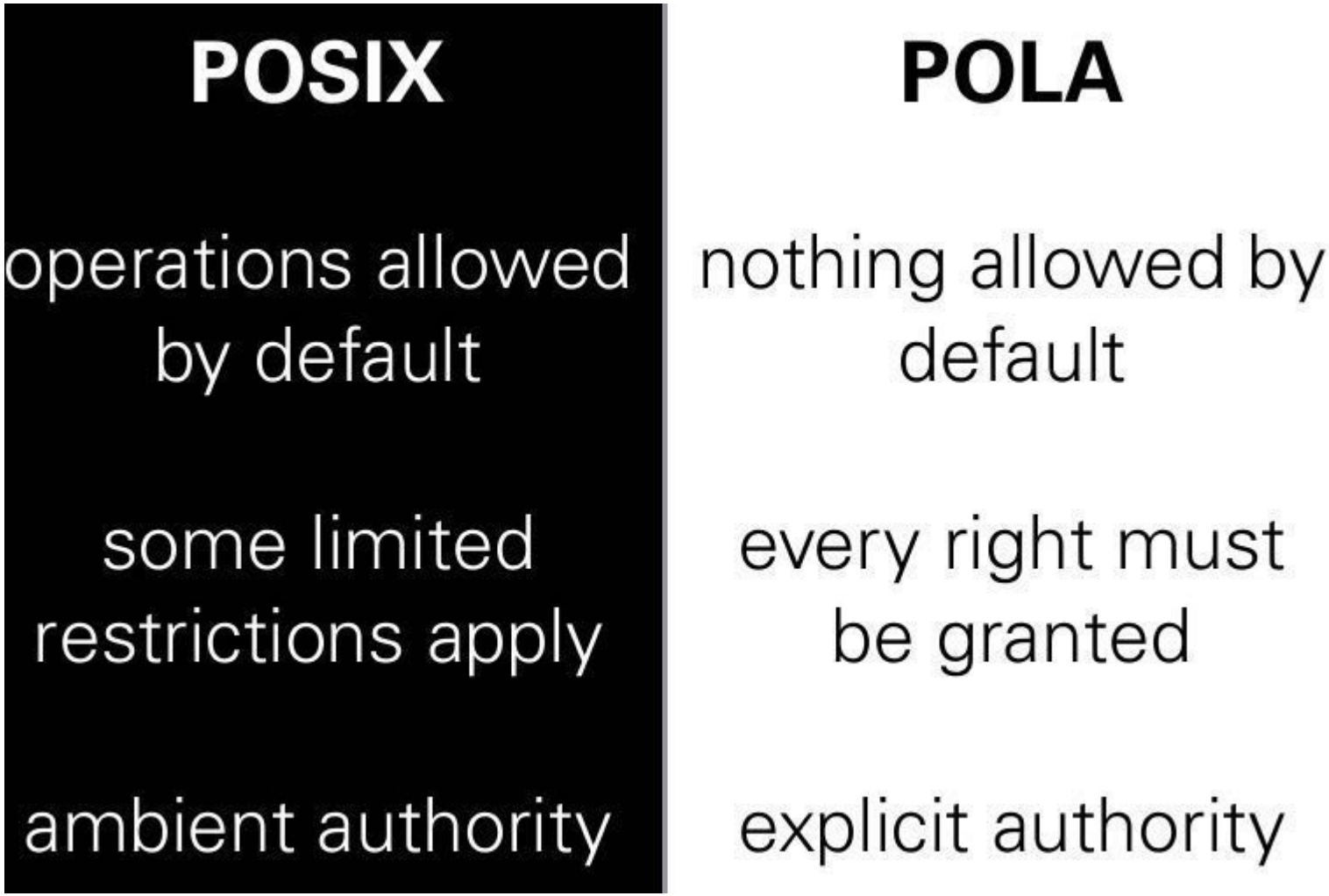
Driver	#loc	#bugs
USB		
RTL8150 USB-to-Ethernet adapter	827	16
EL1210a USB-to-Ethernet adapter	710	2
KL5kusb101 USB-to-Ethernet apapter	925	15
Generic USB network driver	1028	45
USB hub	2234	67
USB-to-serial converter	989	50
USB mass storage	803	23
Firewire		
IEEE1394 Ethernet controller	1413	22
SBP-2 transport protocol	1713	46
PCI		
Mellanox InfiniHost InfiniBand adapter	11718	123
BNX2 Ethernet adapter	5412	51
i810 frame buffer	2920	16
CMI8338 audio	2660	22
		498

Microkernel

- Minimalist approach
  - IPC, virtual memory, thread scheduling
- Put the rest into user space
  - Device drivers, networking, file system, user interface
- Disadvantages
  - Lots of system calls and context switches
- Examples: Mach, L4, QNX, MINIX, IBM K42



principle of least privilege (POLA)



A capability is a communicable, unforgeable token of authority. It refers to a value that references an object along with an associated set of access rights. A user program on a capability-based operating system must use a capability to access an object.

Microkernel Concepts

- Minimal kernel and hardware enforce separation
- Only kernel runs in CPU privileged mode
- Components are user level processes
- No restrictions on component software
- Reuse of legacy software

- *“A concept is tolerated inside the microkernel only if moving it outside the kernel, i.e., permitting competing implementations would prevent the implementation of the systems' required functionality.* “ – Jochen Liedtke

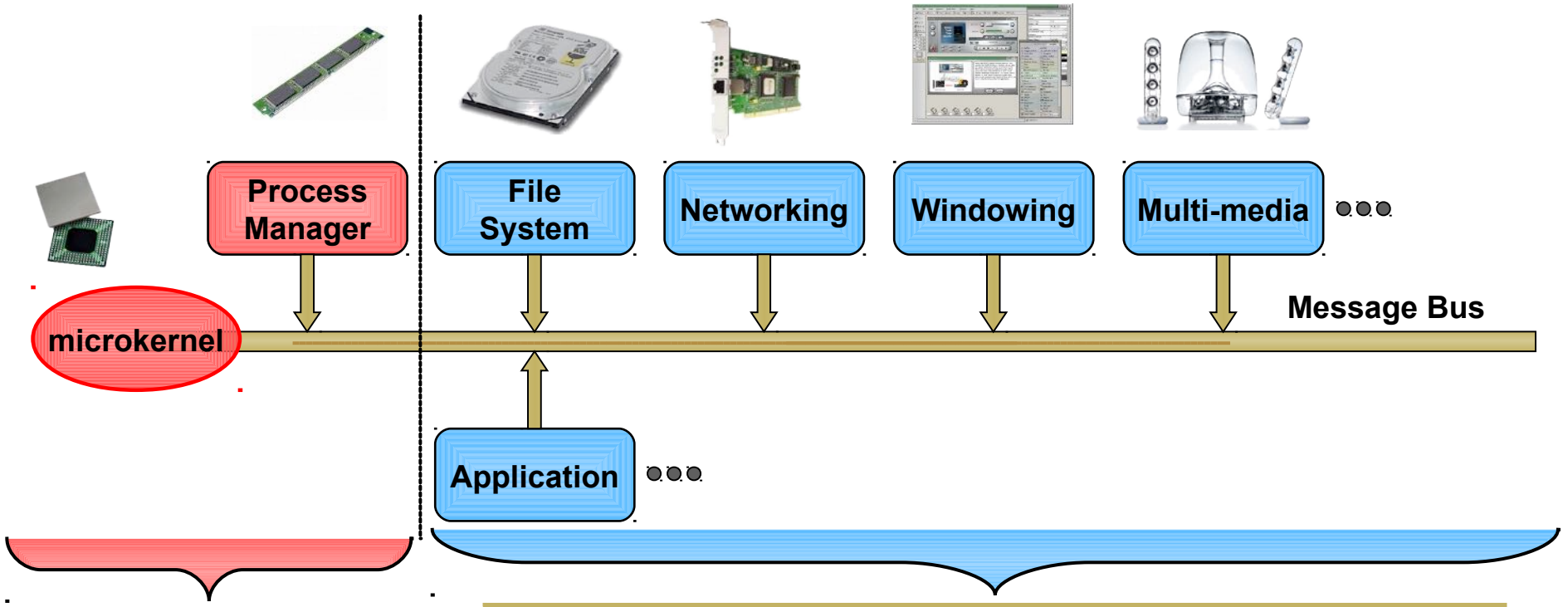
“Worse Is Better", Richard P. Gabriel

	New Jersey style [UNIX, Bell Labs]	MIT style [Multics]
Simplicity	No.1 consideration Implementation > Interface	Interface > Implementation
Correctness	mostly	100%
Consistency	mostly	100%
Completeness	de facto	mostly

- Design competition between New Jersey and MIT style
- Interface first [Multics] → Implementation first [Unix] → Interface first [Mach] → Implementation first [Linux] → Interface first [seL4]

Microkernel

- Put the rest into user space
  - Device drivers, networking, file system, user interface



Microkernel + Process Manager are the only trusted components

Applications and Drivers

- Are processes which plug into a message bus
  Reside in their own memory-protected address space
- Have a well defined message interface
- Cannot corrupt other software components
- Can be started, stopped and upgraded on the fly

Microkernel: Definitions

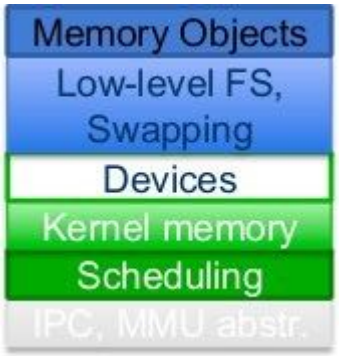
- A kernel technique that provides only the minimum OS services.
  - Address Spacing
  - Inter-process Communication (IPC)
  - Thread Management
  - Unique Identifiers
- All other services are done at user space independently.

3 Generations of Microkernel

- Mach (1985-1994)
  - replace pipes with IPC (more general)
  - improved stability (vs monolithic kernels)
  - poor performance
- L3 & L4 (1990-2001)
  - order of magnitude improvement in IPC performance
    - written in assembly, sacrificed CPU portability
    - only synchronus IPC (build async on top of sync)
  - very small kernel: more functions moved to userspace
- seL4, Fiasco.OC, Coyotos, NOVA (2000-)
  - platform independence
  - verification, security, multiple CPUs, etc.

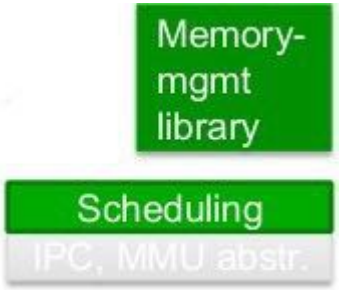
3 Generations of Microkernel

- Generation 1: Mach (1985-1994)



- Generation 2: L3 & L4 (1990-2001)

- Generation 3: seL4, Fiasco.OC, NOVA (2000-)



Performance of 1st Generation

CMU Mach (1985), Chorus (1987), MkLinux (1996)

- Does not prohibit caching
- Reduce number of copies of data occupying memory
  - Copy-to-use, copy-to-kernel
  - More memory for caching
- I/O operations reduced by a factor of 10
- Context switch overhead
  - Cost of kernel overhead can be up to 800 cycles.
- Address Space Switches
  - Expensive Page Table and Segment Switch Overhead
  - Untagged TLB = Bad performance

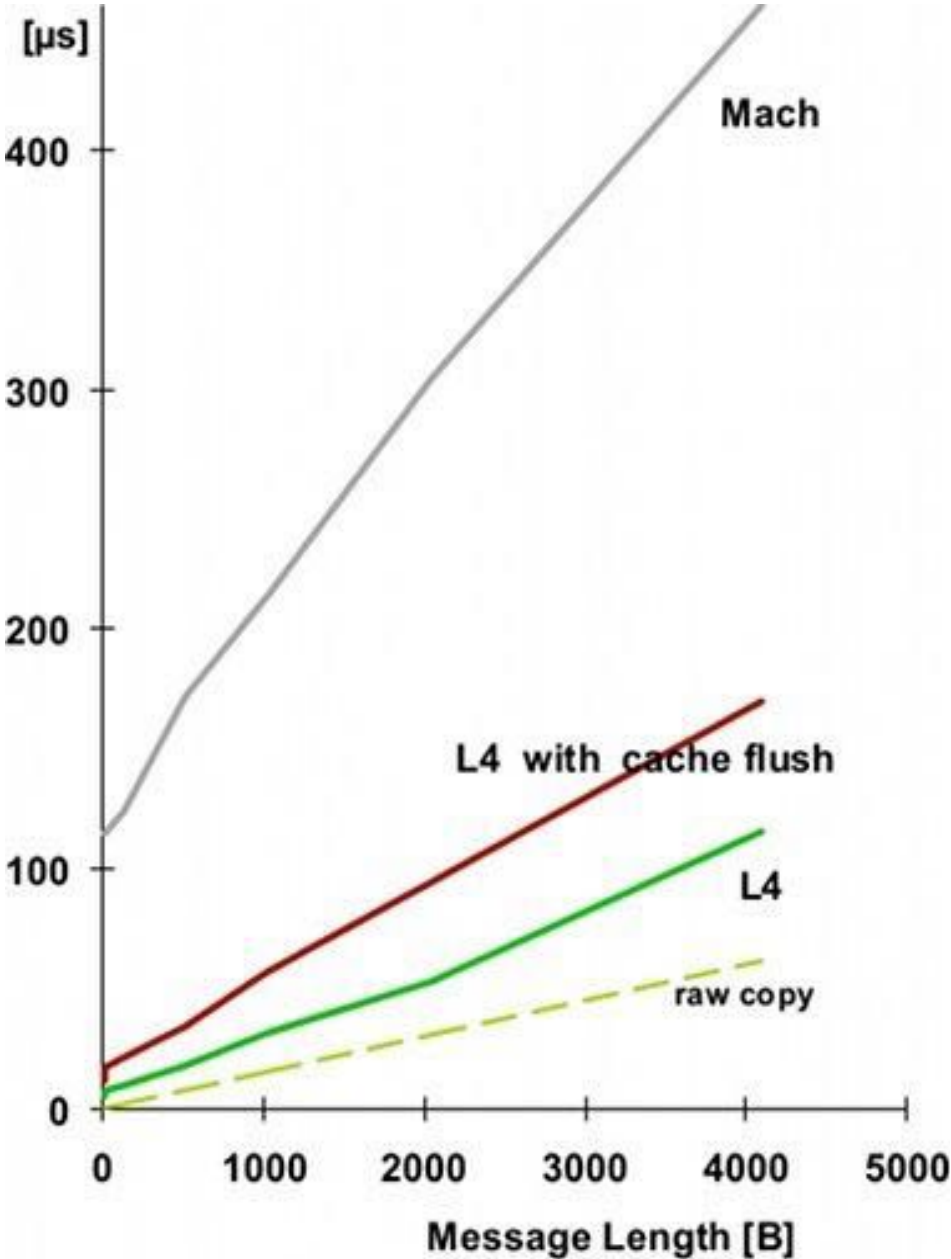
L4: the 2nd Generation

- Similar to Mach
  - Started from scratch, rather than monolithic
  - But even more minimal
- minimality principle for L4:

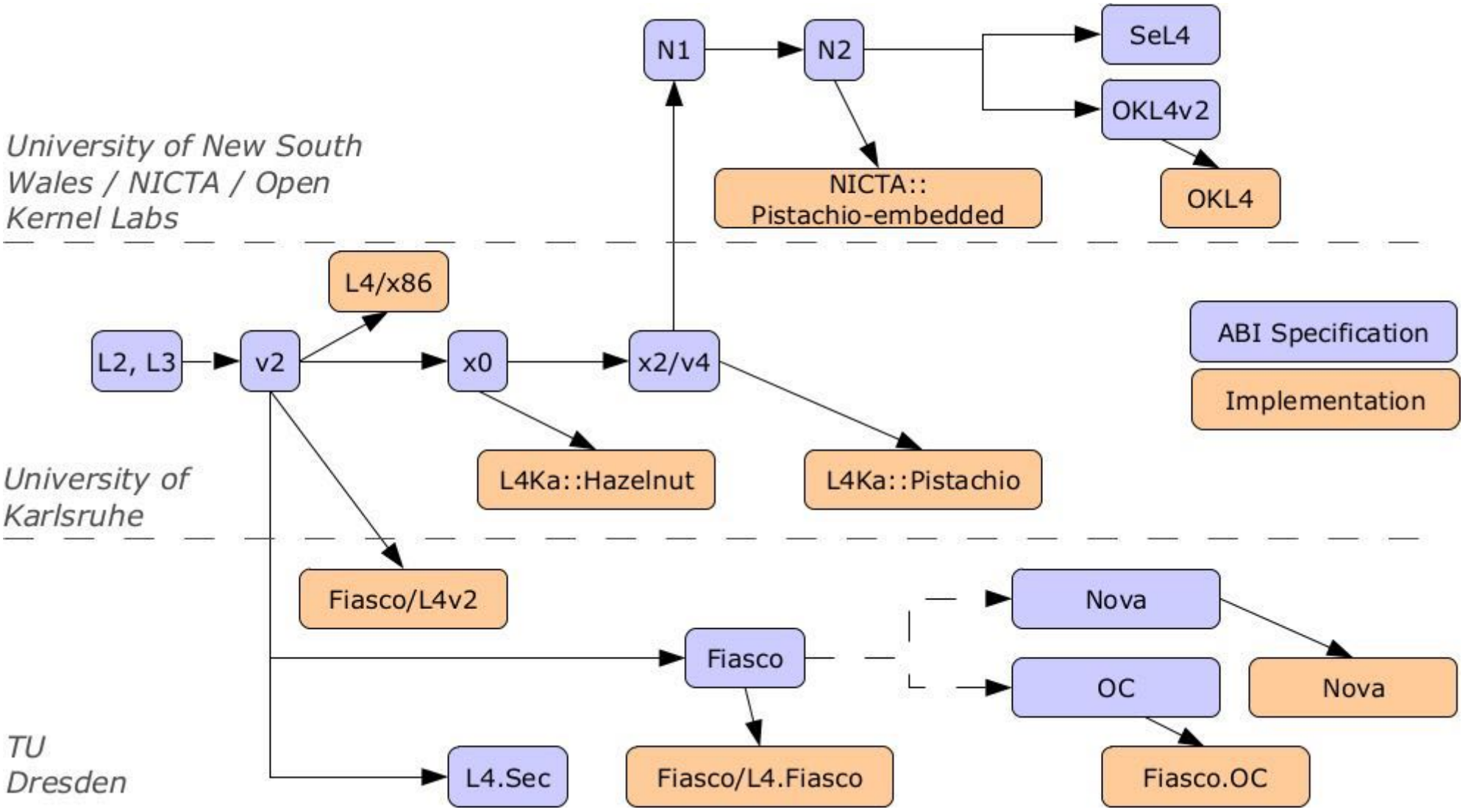
  A concept is tolerated inside the microkernel only if moving it outside the kernel, *i.e.*, permitting competing implementations, would prevent the implementation of the system's required functionality.
- Tasks, threads, IPC
  - Uses only 12k of memory
  - API size of Mach: 140 functions (Asynchronus IPC, Threads, Scheduling, Memory management, Resource access permissions)
  - API size of L4: 7 function (Synchronous IPC, Threads, Scheduling, Memory management)

Performance Gain (1st to 2nd Generation)

- Reason of being slow kernels: Poor design [Liedtke SOSP'95]
  - complex API
  - Too many features
  - Poor design and implementation
  - Large cache footprint ⇒ memory-bandwidth limited
- L4 is fast due to small cache footprint
  - 10–14 I-cache lines
  - 8 D-cache lines
  - Small cache footprint ⇒ CPU limited



L4 Family (incomplete)



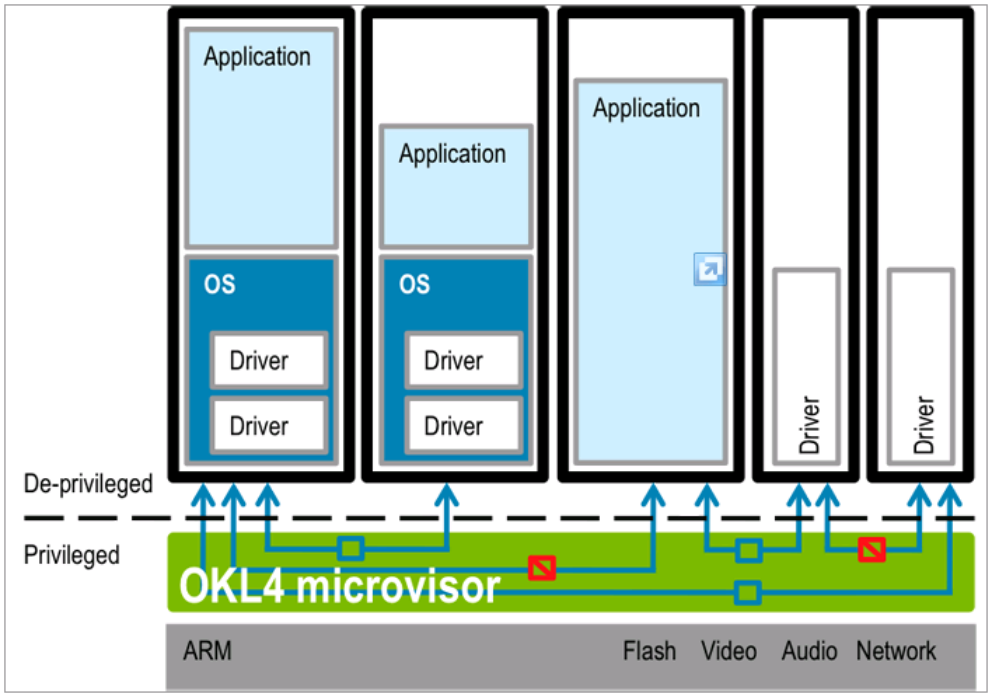
Source: Microkernel-based Operating Systems – Introduction, Carsten Weinhold, TU Dresden (2012)

Commercial L4: from NICTA to OKLabs

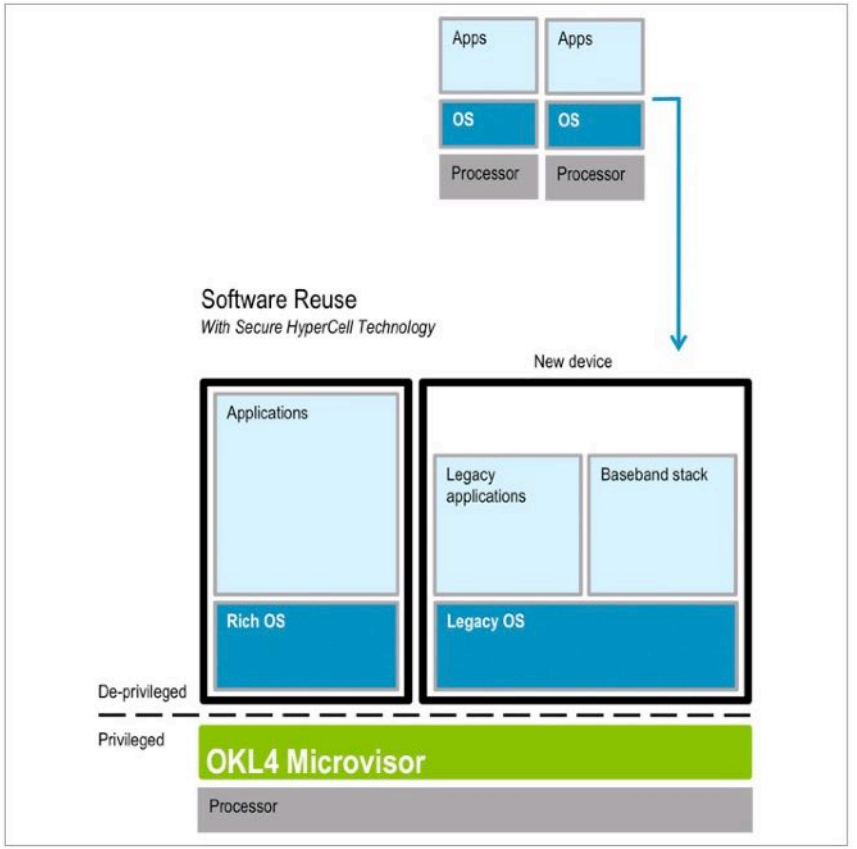
- L4::Pistachio microkernel was originally developed at Karlsruhe University. NICTA had ported it to a number of architectures, including ARM, had optimized it for use in resource-constrained embedded systems.
- In 2004, Qualcomm engaged NICTA in a consulting arrangement to deploy L4 on Qualcomm's wireless communication chips.
- The engagement with Qualcomm grew to a volume where it was too significant a development/engineering effort to be done inside the research organization.
  – Commercized! Open Kernel Labs
- Acquired by General Dynamics in 2012

Source: http://microkerneldude.wordpress.com/2012/10/02/giving-it-away-part-2-on-microkernels-and-the-national-interes/

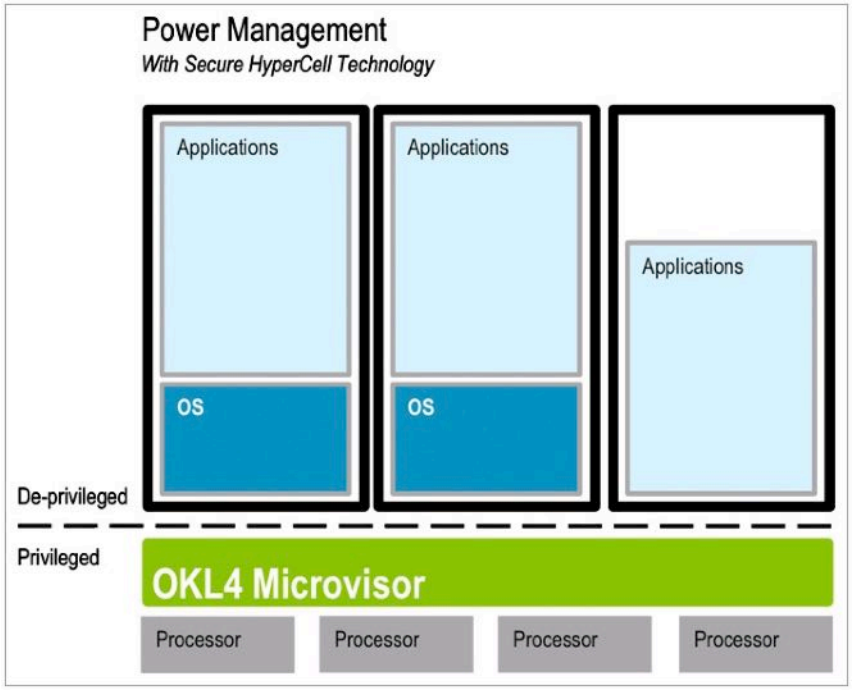
OKL4 Use Cases



Each secure cell in the system offers isolation from software in other cells

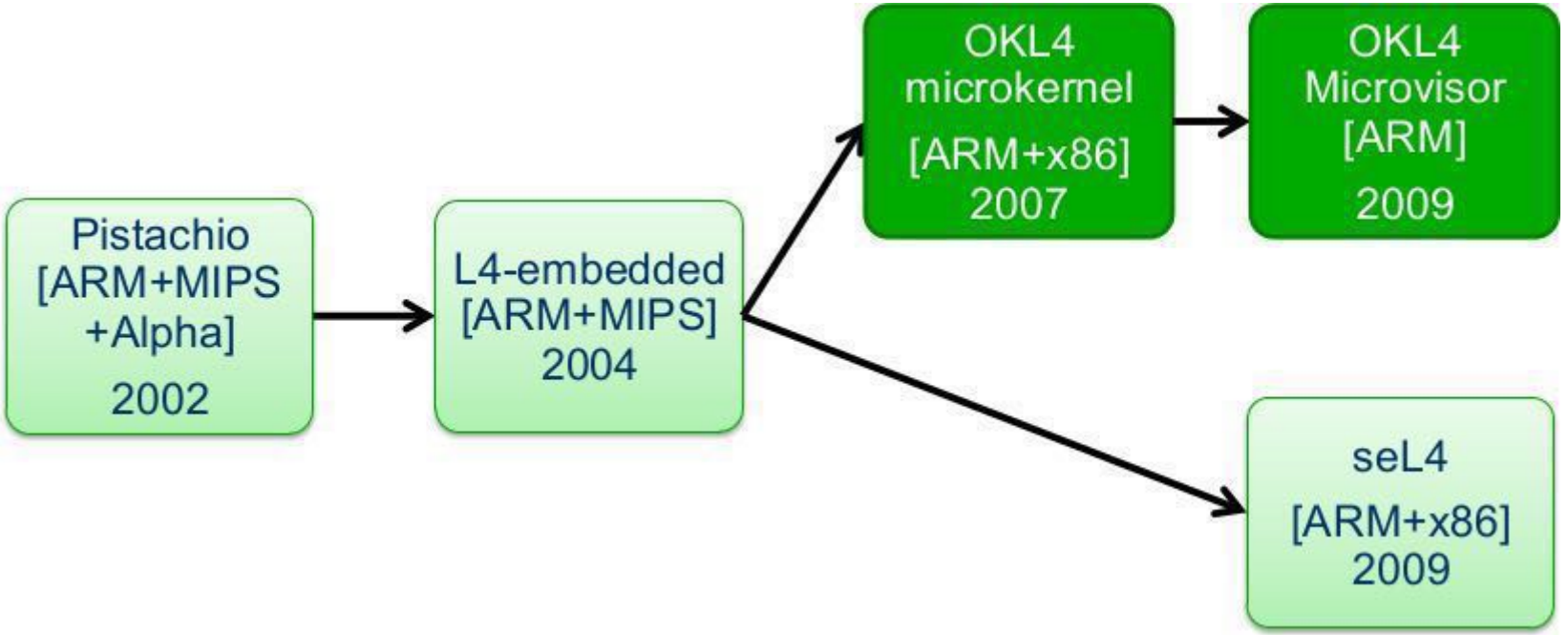


Existing software components can be reused in new designs



Microvisor tames the complexity of dispatching multi-OS workloads across multiple physical CPUs

Moving from 2nd to 3rd Generation



OKL4

- Dumped recursive address-space model
  - reduced kernel complexity
  - First L4 kernel with capability-based access control

OKL4 Microvisor

- Removed synchronous IPC
- Removed kernel-scheduled threads

seL4

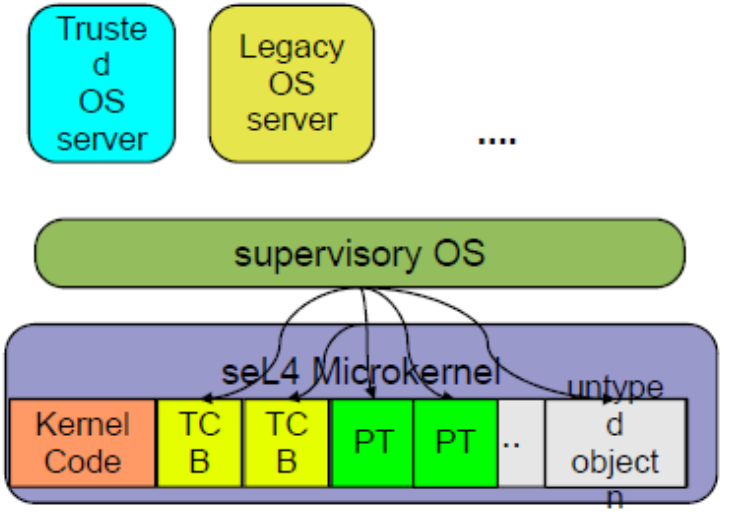
- All memory management at user level
  - no kernel heap!
- Formal proof of functional correctness
- Performance on par with fastest kernels
  - <200 cycle IPC on ARM11 without assembler fastpath

Problems in 2nd Generations

- microkernel needs memory for its abstractions
  - tasks: page tables
  - threads: kernel-TCB
  - capability tables
  - IPC wait queues
  - mapping database
  - kernel memory is limited
  - opens the possibility of DoS attacks

seL4 as 3rd Microkernel

- Functional Correctness [SOSP'09]
- Timeliness (known WCET) [RTSS'11,EuroSys'12]
- Translation Correctness [PLDI'13]
- Fast (258 cycle IPC roundtrip on 1GHz Cortex-A9)
- Safety: specifically temporal properties.
- Minimal TCB (~9000 SLoC)



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

F9: A new microkernel designed for Deeply Embedded Devices

Deeply Embedded Devices

- Power awareness; solid and limited applications
- Multi-tasking or cooperative scheduling is still required
- IoT (Internet of Things) is the specialized derivative with networking facility
- Communication capability is built-in for some products
- Example: AIRO wristband (health tracker)

  `http://www.weweartech.com/amazing-new-uses-smart-watches/`

Design Considerations of IoT

- Network
  - IoT networks must be scalable in order to support the dynamic nature of the IoT (as devices are added and removed from the network).
- Security
  - Integration of security protocols for encryption and authentication must always be required.
  - Before any data is transferred, the source of the data needs to be verified.
  - The use of encryption prevents the loss of data to passive listeners, but is does not prevent the alteration of data while traversing the network.
- Power Management

  Facilitate processors with many low-power features including DVFS and Hibernate.
- Need for full-featured RTOS framework

Advanced Software Requirements of IoT Products

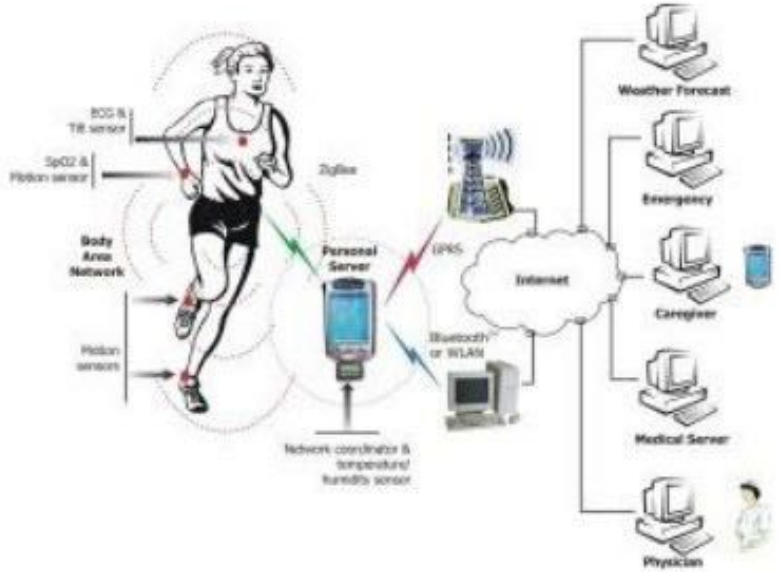



Photo: Philips

- Over-The-Air (OTA) update with a double bank firmware update mechanism. The switch to a new version is only operated when the newly downloaded content is fully validated.
- A dedicated first stage loader/diagnostic/recovery application is used for this update mechanism. It provides full access to all internal and external memories.

Characteristics of F9 Microkernel

https://github.com/f9micro

Unique Characteristics

- BSD Licensing (two-clause), suitable for both research and commercial usage.
  - Commercial adaptation since 2014
- Efficiency
  - Optimized for ARM Cortex-M3/M4
  - performance: fast IPC and well-structured designs
  - energy-saving: tickless scheduling, adaptive power management
- Security
  - memory protection: MPU guarded
  - Isolated execution: L4 based, capabilities model
- Flexible development
  - Kprobes
  - profile-directed optimizations

Why are current systems unreliable?

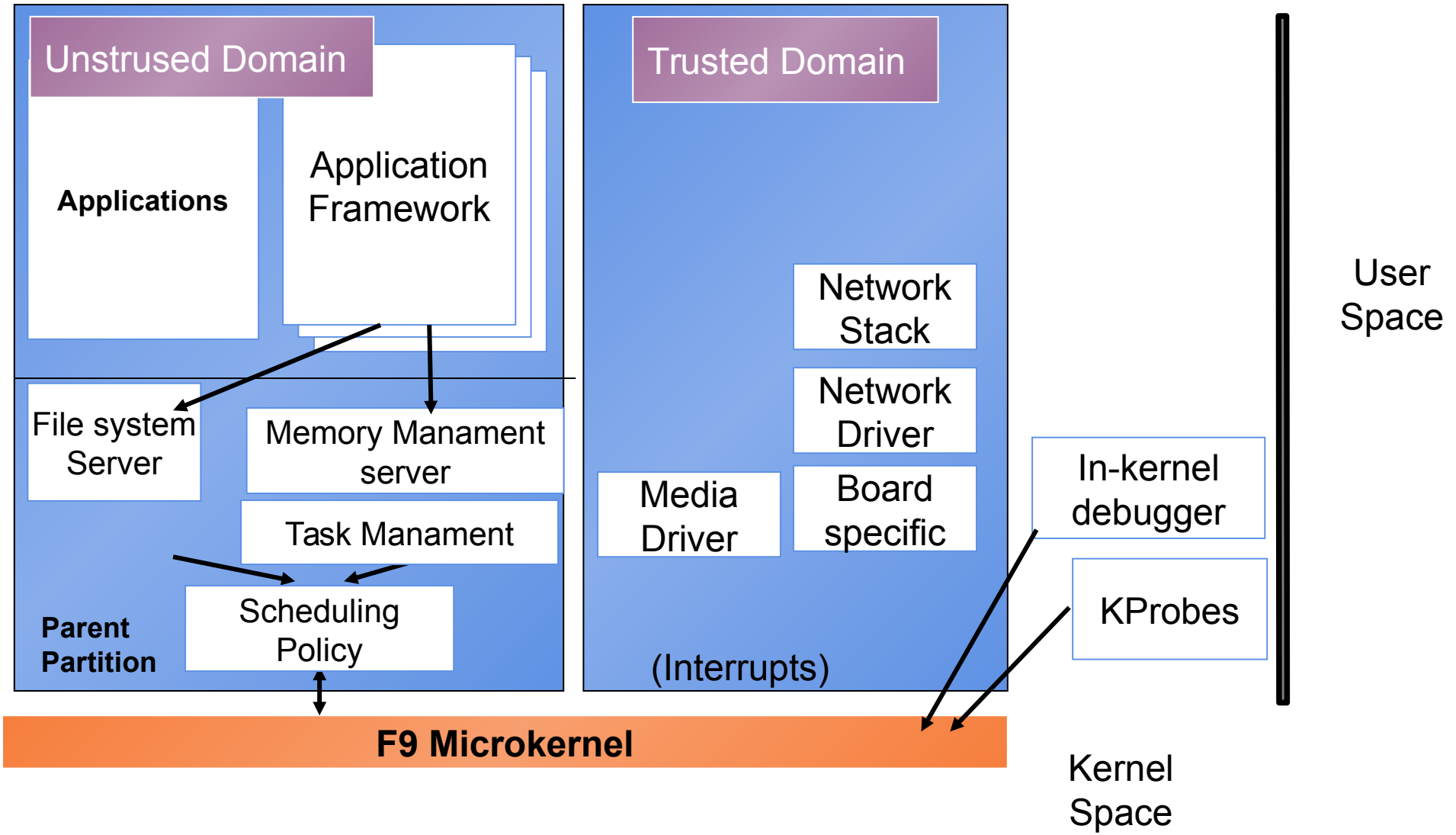
- Problem 1: “Systems are huge"
  - No single person can understand the whole system

    > F9 Microkernel has only 3K LoC of portable C
- Problem 2: “Bug fixes usually introduce new bugs."

  > F9 introduces execution domains and on-the-fly patches
- Problem 3: “Poor fault isolation"
  - No isolation between system components
  - OS contains hundreds of procedures linked together as a single binary program running on the kernel mode.

    > F9 is built from scratch and well-engineered for isolation

F9/BitSec Architecture



Principles

- F9 follows the fundamental principles of L4 microkernels
  - implements address spaces, thread management, and IPC only in the privileged kernel.
- Designed and customized for ARM Cortex-M, supporting NVIC (Nested Vectored Interrupt Controller), Bit Banding, MPU (Memory Protection Unit)

Thread

- Each thread has its own TCB (Thread Control Block) and addressed by its global id.
- Also dispatcher is responsible for switching contexts. Threads with the same priority are executed in a round-robin fashion.

Memory Management

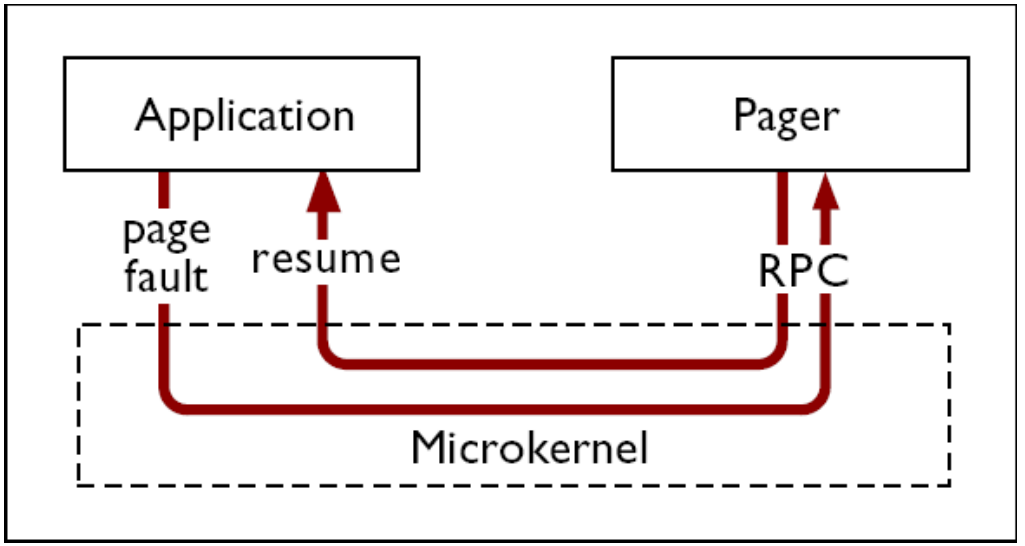
- split into three concepts:
  - **Memory pool**, which represent area of physical address space with specific attributes.
  - **Flexible page**, which describes an always size aligned region of an address space. Unlike other L4 implementations, flexible pages in F9 represent MPU region instead.
  - **Address space**, which is made up of these flexible pages.
- System calls are provided to manage address spaces:
  - **Grant**: memory page is granted to a new user and cannot be used anymore by its former user.
  - **Map**: This implements shared memory – the memory page is passed to another task but can be used by both tasks.
  - **Flush**: The memory page that has been mapped to other users will be flushed out of their address space.

IPC

- The concept of UTCB (user-level thread-control blocks) is being taken on. A UTCB is a small thread-specific region in the thread's virtual address space, which is always mapped. Therefore, the access to the UTCB can never raise a page fault, which makes it perfect for the kernel to access system-call arguments, in particular IPC payload copied from/to user threads.
- Kernel provides synchronous IPC (inter-process communication), for which short IPC carries payload in CPU registers only and full IPC copies message payload via the UTCBs of the communicating parties.

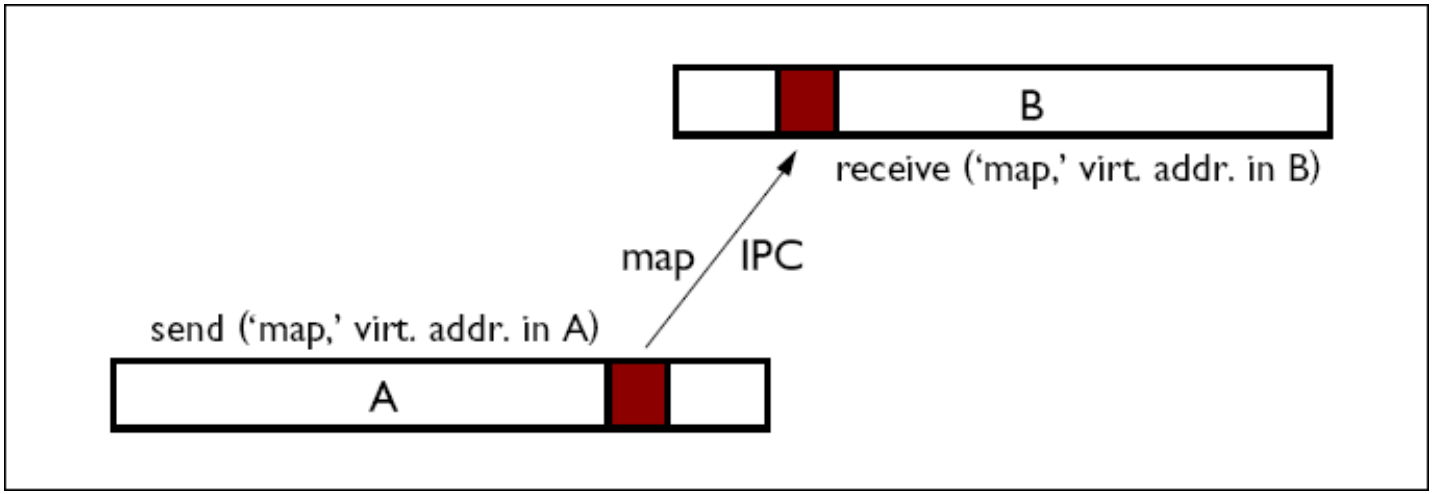
Microkernel Paging

- Microkernel forwards page fault to a pager server.
- Kernel or server decides which pages need to be written to disk in low memory situations.
- Pager server handles writing pages to disk.



Recursive Address Space

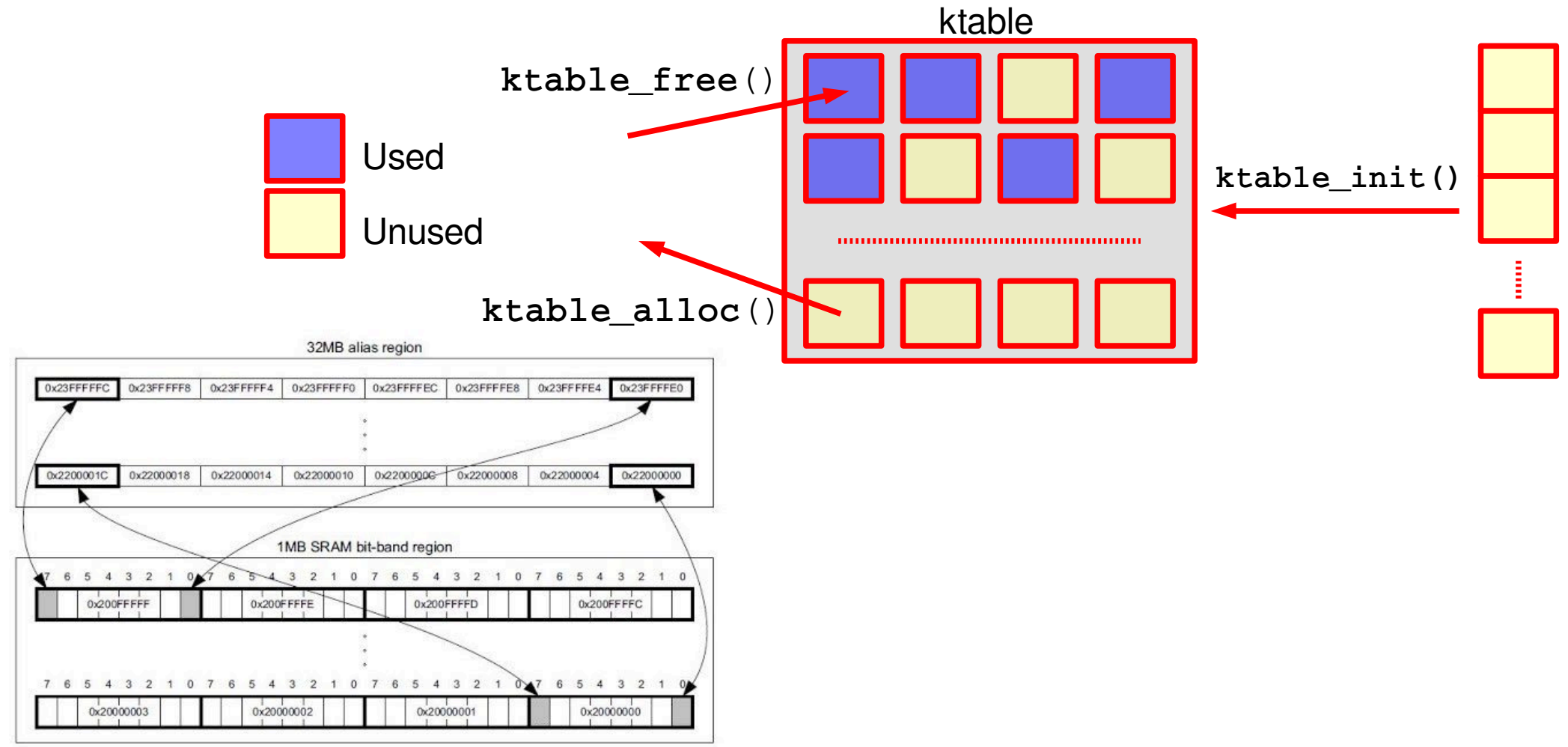
- Initial address space controlled by first process.
  - Controls all available memory.
  - Other address spaces empty at boot.
- Other processes obtain memory pages from first or from their other processes that got pages from first.
- Why is memory manager flexibility useful?
  - Different applications: real-time, multimedia, disk cache.



Grant
Map
Flush

Ktable: fast memory poll

- Ktable is in charge of the allocation / deallocation for the objects of pre-defined size and numbers easier
- Can be optimized with Bit-banding of ARM Cortex-M

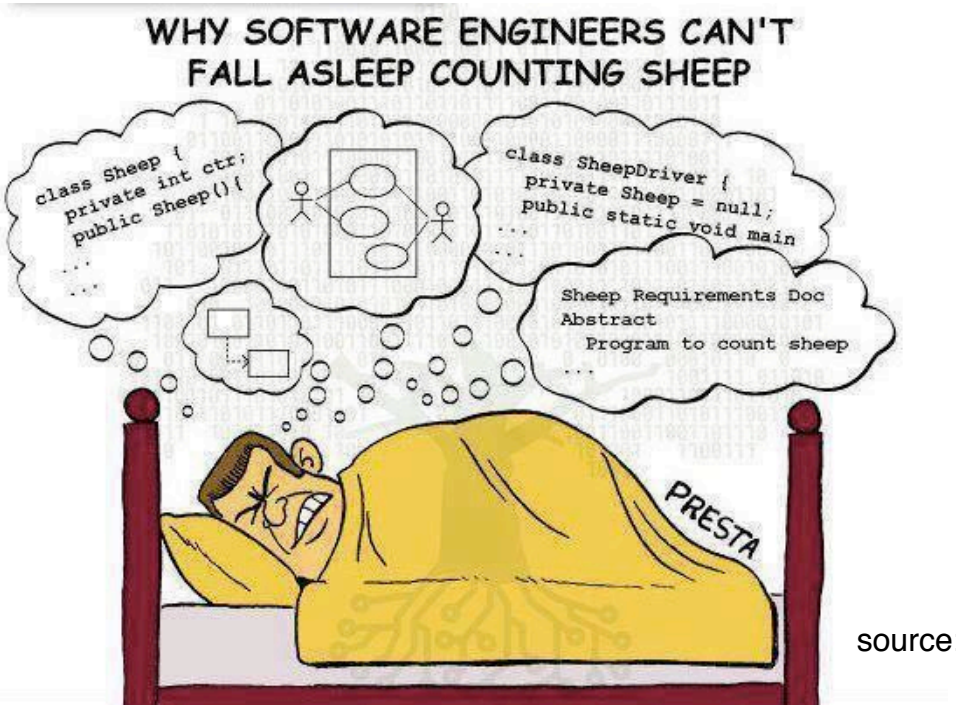


Interrupt Handling

- Two-stage interrupt handling
  - ISR: IRQ context
  - Softirq
    - Thread context
    - Real time preemptive characteristic
    - Can be scheduled like any other threads in the system
- Handled in both **kernel thread and user-space**

Energy efficiency: Tickless

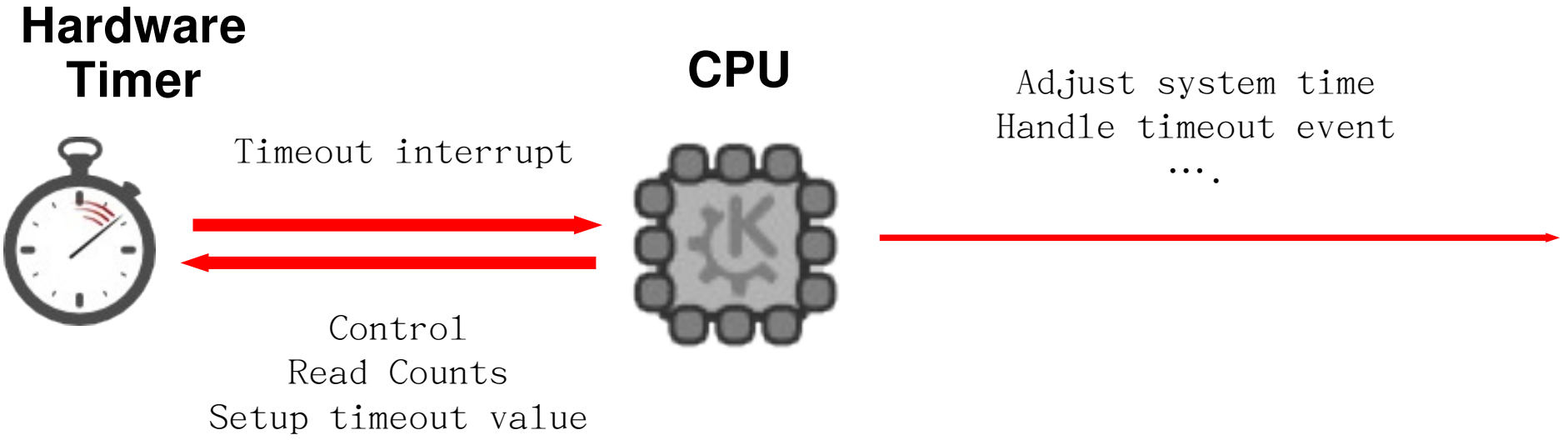
- Introduce tickless timer which allow the ARM Cortex-M to wake up only when needed, either at a scheduled time or on an interrupt event.
- Therefore, it results in better current consumption than the common approach using the system timer, SysTick, which requires a constantly running and high frequency clock.



source: https://twitter.com/manisha72617183/status/819837383319298048

How Tick is Implemented

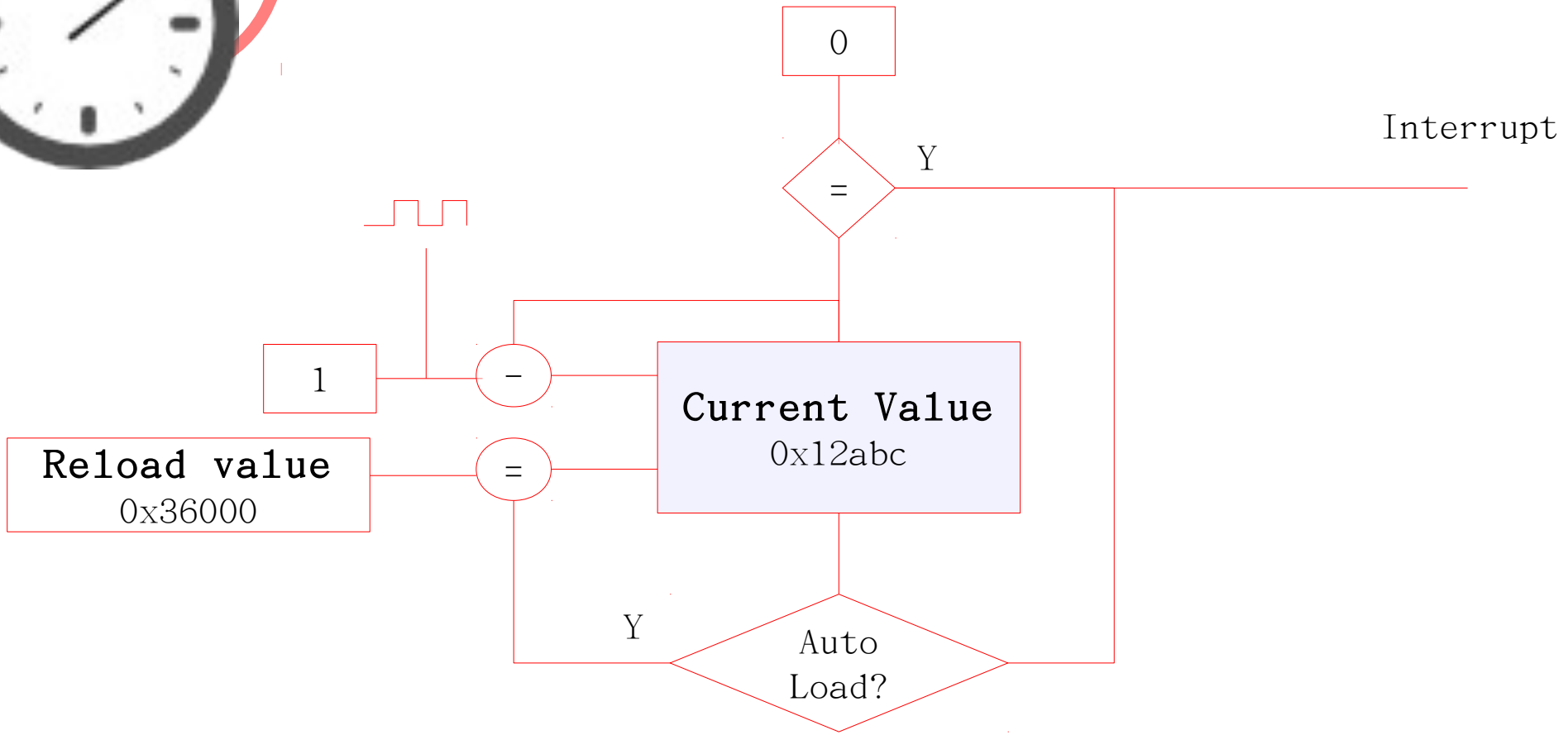
- Hardware timer device
  - Assert interrupt after a programmable inteval
  - Handling tick stuff in Timeout Interrupt Service Routine (ISR)



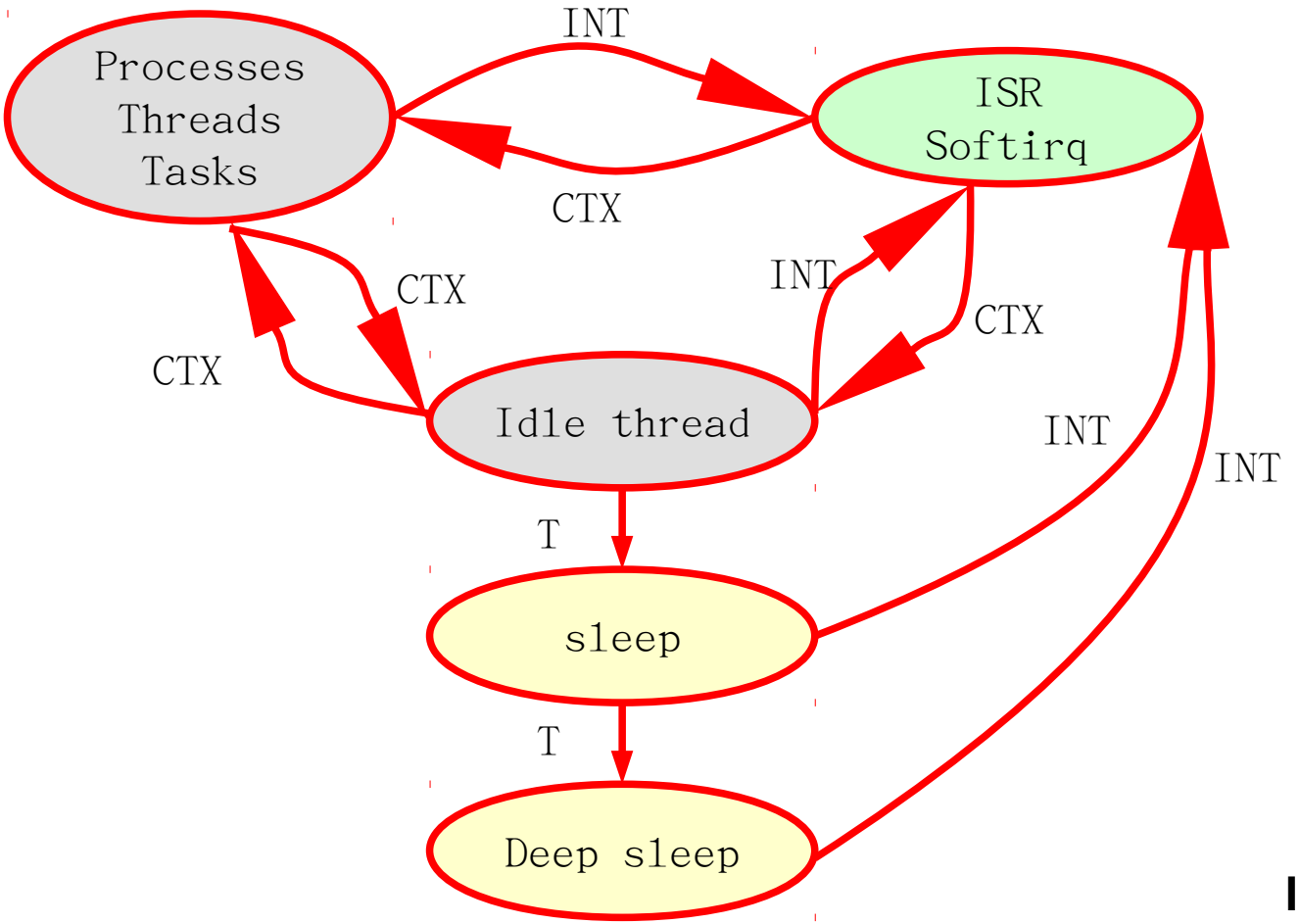
SysTick in ARM Cortex-M4

- Count-down timer

- Timeout ISR
  - Increase system ticks
  - Execute handler of timeout event
  - Re-schedule if required

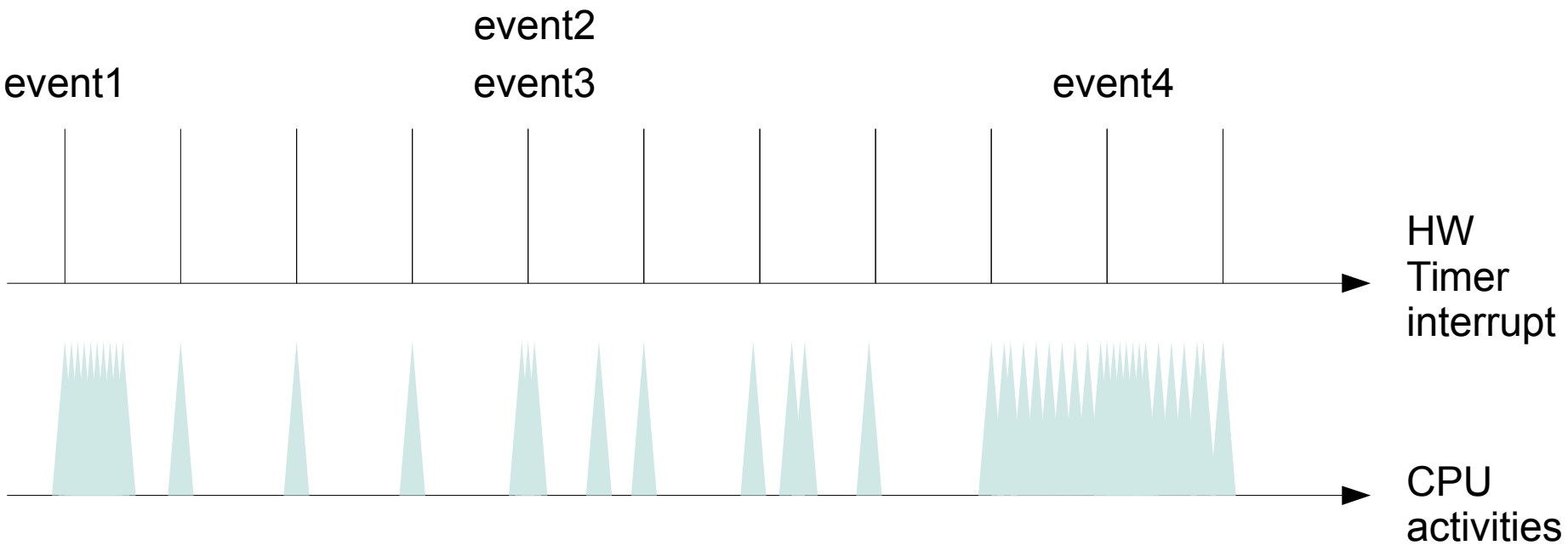


CPU Operating States

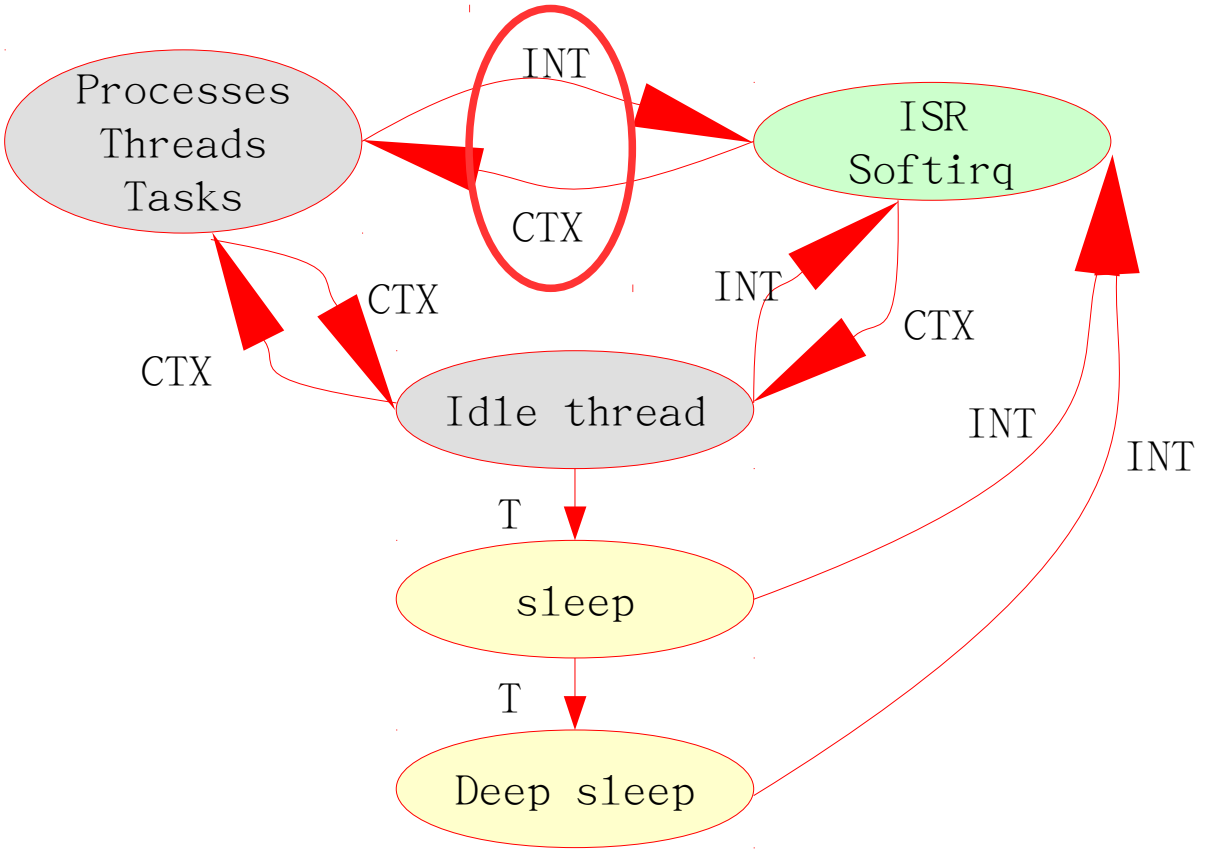


INT : interrupt
CTX: context switch
T : after a while

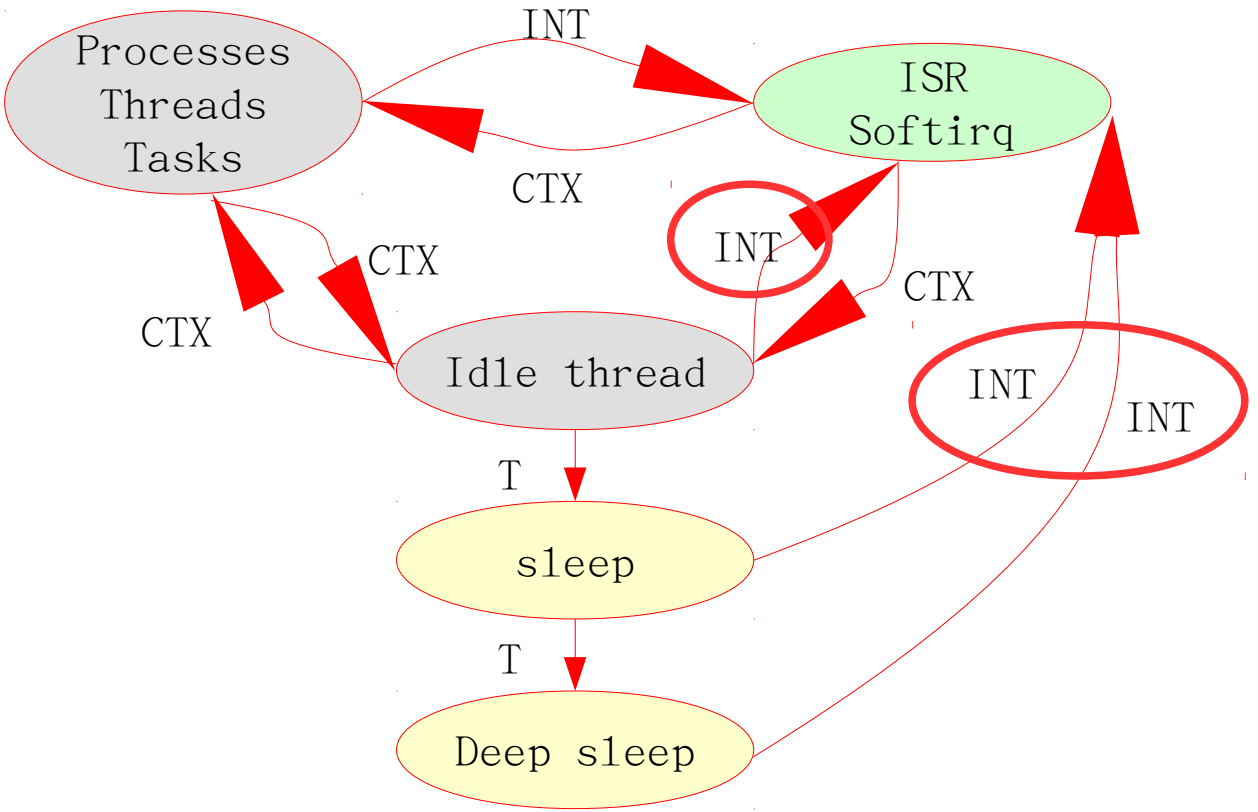
Time Diagram of Legacy Ticks



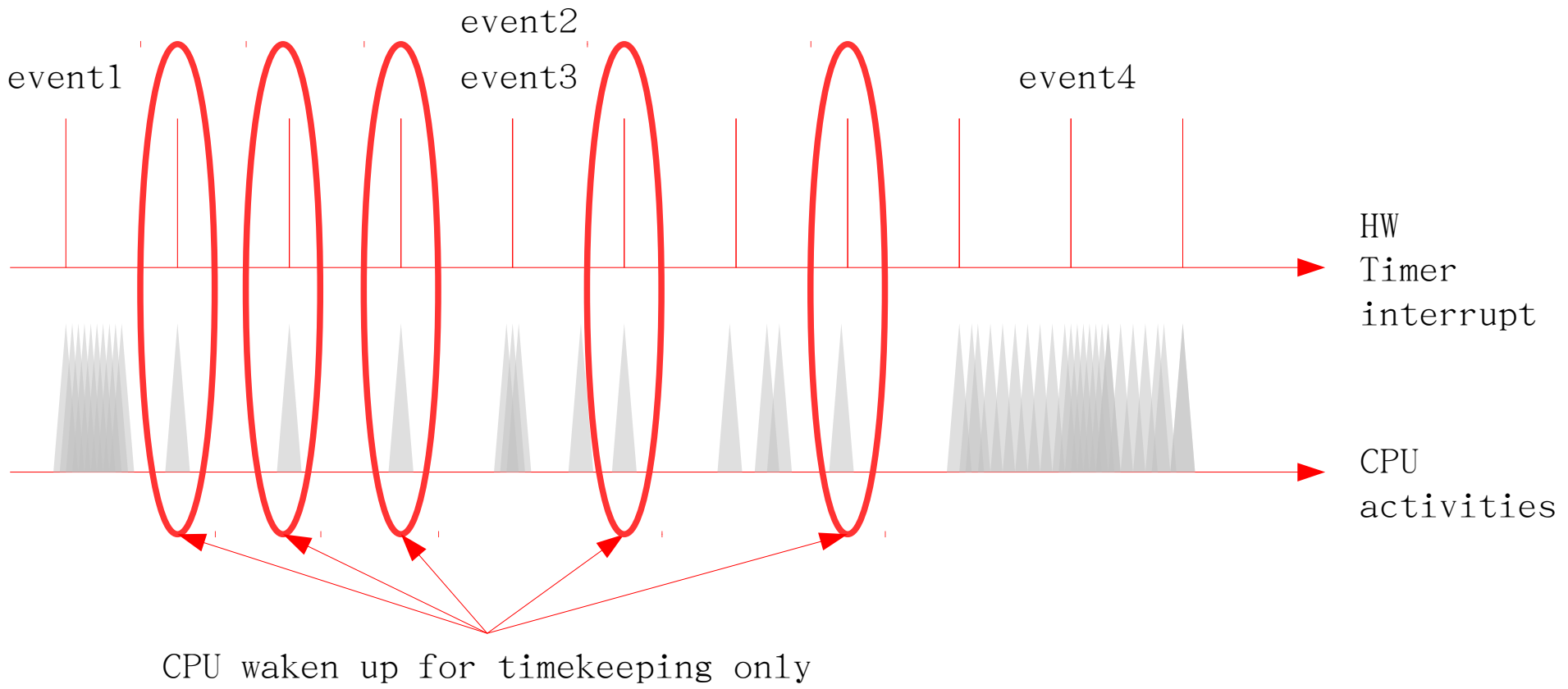
Context Switch overhead



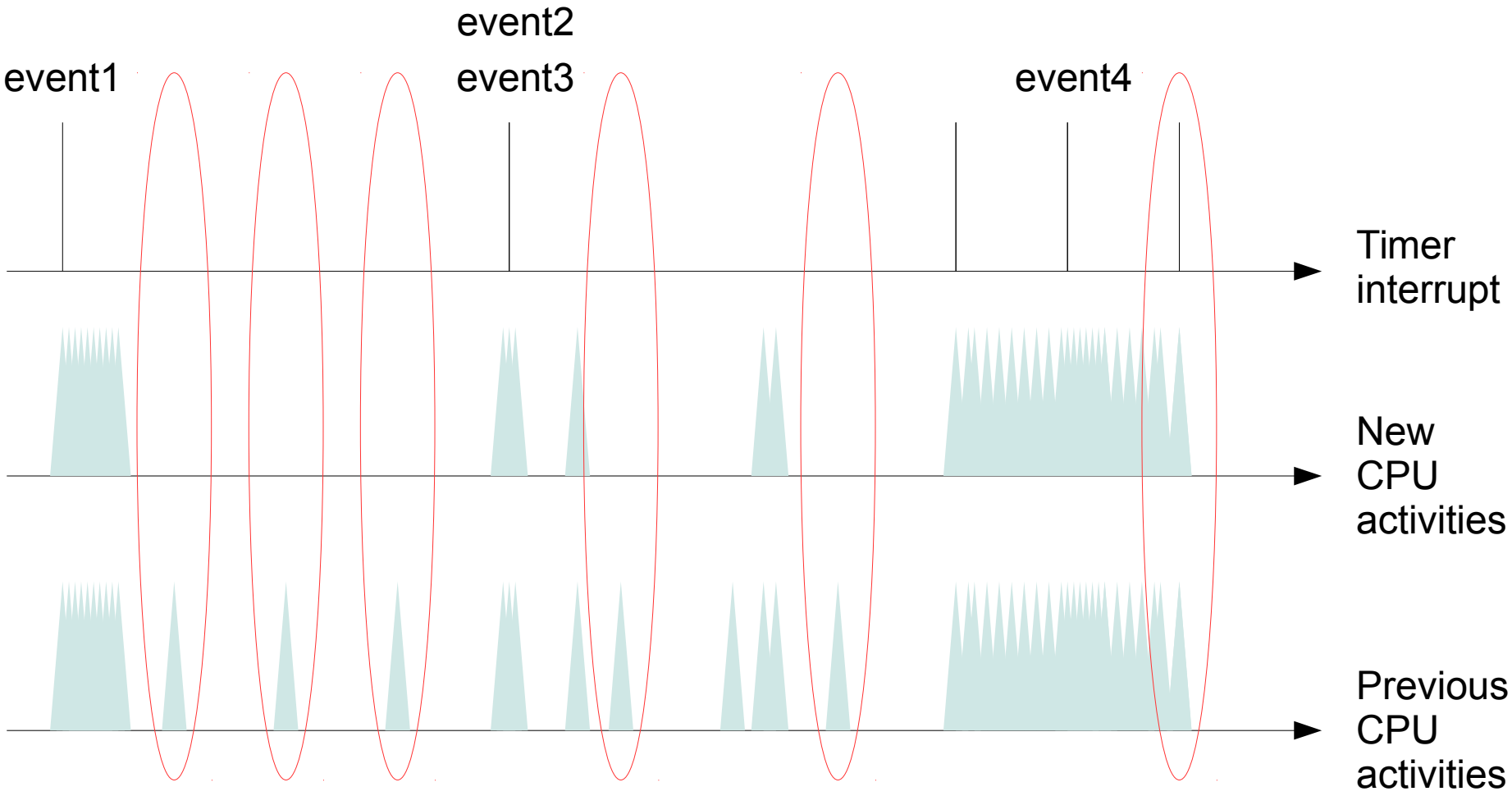
Regular Power Consumption



Time Diagram of Legacy Ticks



Solution: Tickless scheduling

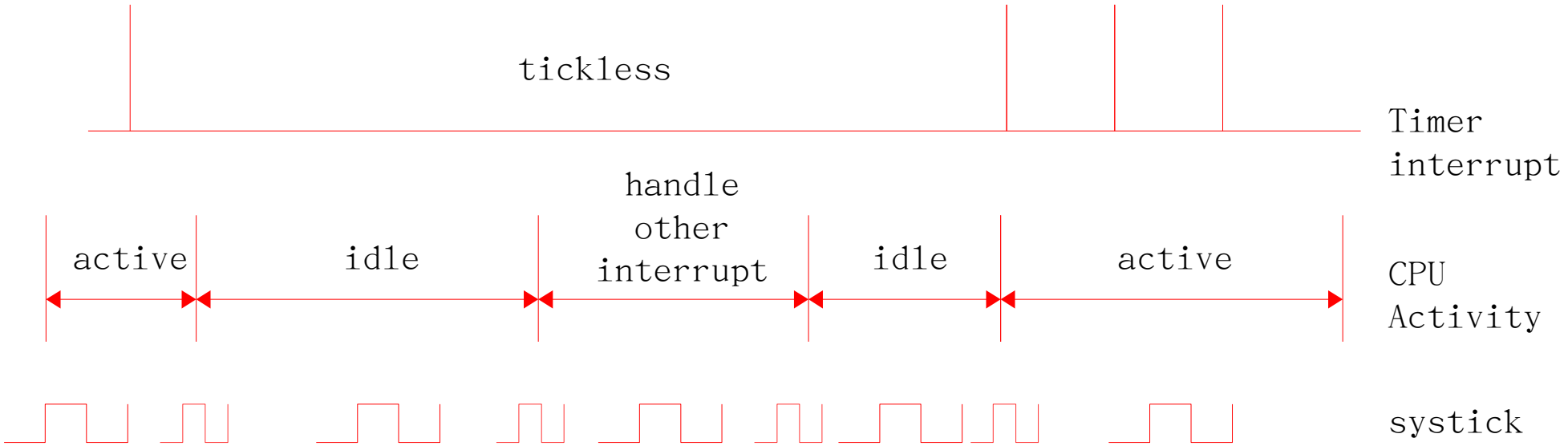


Drawback of Tickless scheduling

- Tickless is not free
  - *"It increases the number of instructions executed on the path to and from the idle loop."*
  - *"On many architectures, dyntick-idle mode also increases the number of expensive clock-reprogramming operations"*
  - *Source: P. E. McKenney (May 14, 2013), "NO_HZ: Reducing Scheduling-Clock Ticks"*
- Systems with aggressive real-time response constraints often run periodic tick

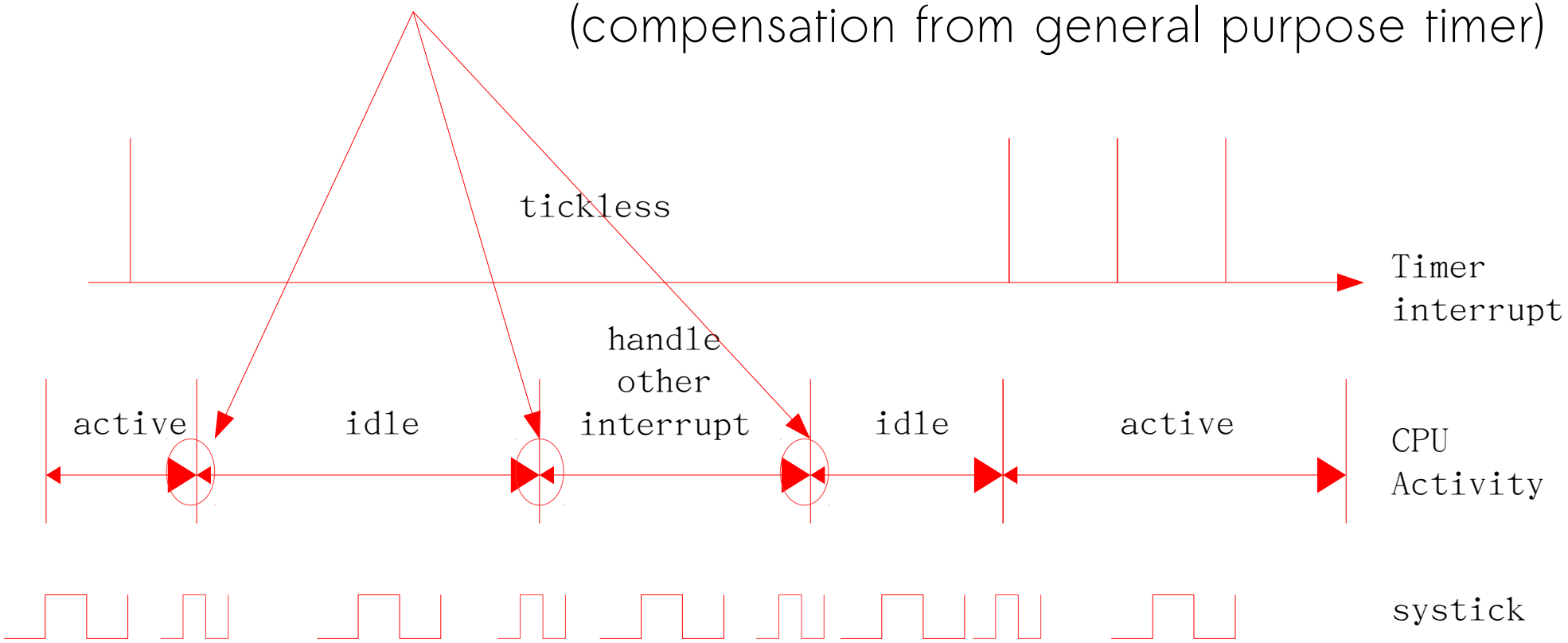
Tickless scheduling in F9

- Enter tickless right before going to CPU idle state
  - Set interval of next timer interrupt as delta of next event
  - Or **KTIMER_MAXTICKS**
- Adjust system time after waked upires a constantly running and high frequency clock.
- Tickless Compensation
  - SysTick frequency distortion when enter/exit standby mode



Tickless compensation

(compensation from general purpose timer)



- System activity during idle with and without periodic ticks

	# interrupts	#events	Avg CPU idle residency (uS)
With ticks	2002	59.59	651
Tickless	118	60.60	10161

- System activity during idle with and without deferrable timer usage in ondemand

	# interrupts	#events	Avg CPU idle residency (uS)
Ondemand	118	60.60	10161
Ondemand + deferrable timer	89	17.17	20312

Kprobes: dynamic instrumentation

- Inspired by Linux Kernel, allowing developers to gather additional information about kernel operation without recompiling or rebooting the kernel.
- It enables locations in the kernel to be instrumented with code, and the instrumentation code runs when the ARM core encounters that probe point.
- Once the instrumentation code completes execution, the kernel continues normal execution.

Application Development

- Partial POSIX support
- configurable debug console
- memory dump
- thread profiling
  – name, uptime, stack allocated/current/used
- memory profiling
  – kernel table, pool free/allocated size, fragmentation
- Link-Time Optimization (LTO) and PGO

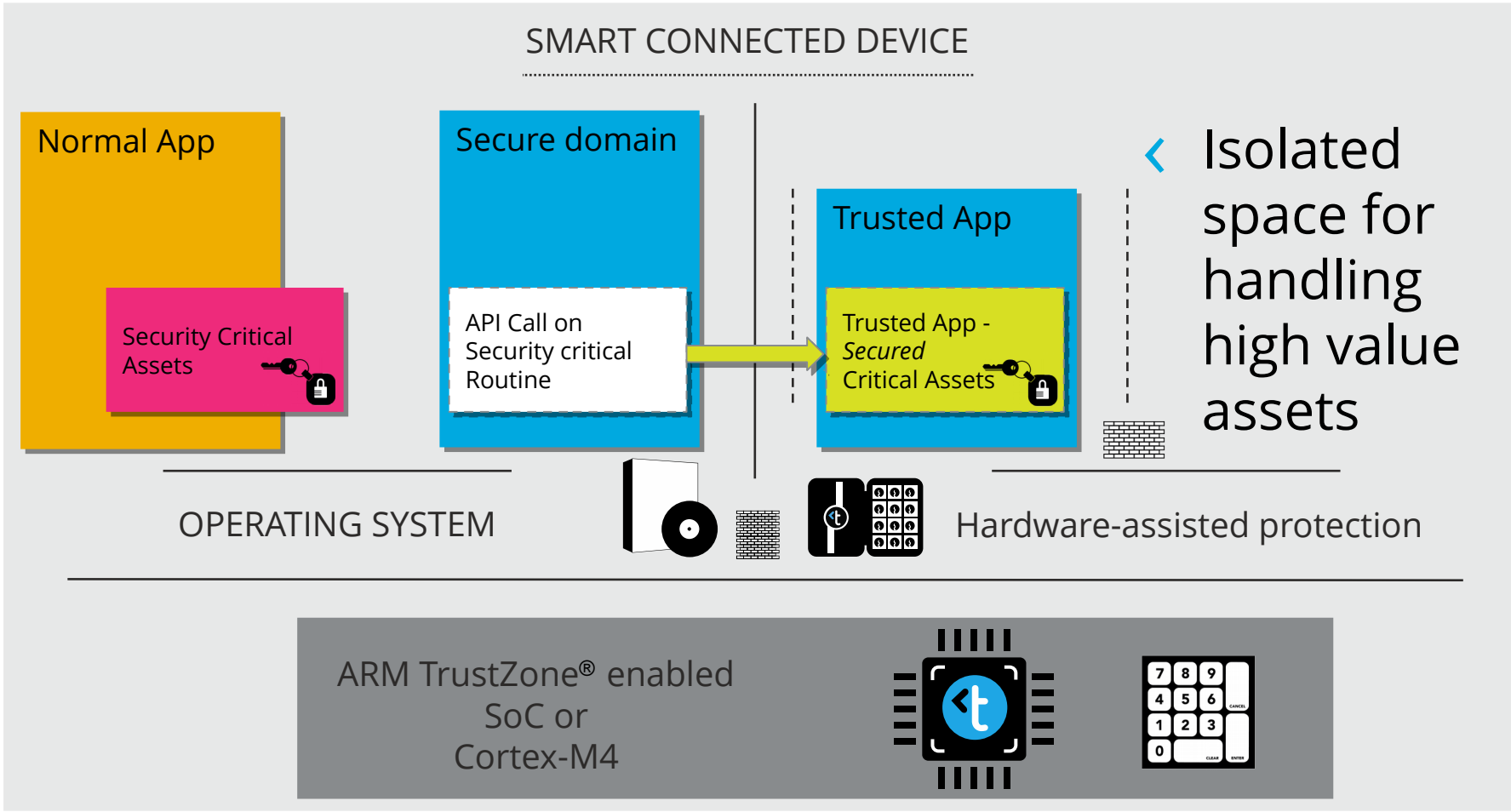
Commercial Adaptation

- F9 microkernel is used by Genesi USA, Inc. as smart solutions for the internet of things

  http://genesi.company/solutions/embedded
- Genesi's Radix K1 is a low cost embedded device built around Freescale ARM Cortex-M4
  - 100MHz based MCU with 512kB of FLASH and 128KB of built-in RAM and a 4G GSM module.
- The device ←→ server communication link uses WAMP, a WebSocket subprotocol and the data exchanged is encrypted using CycloneSSL.
- Basic memory protection is available through built-in MPU.

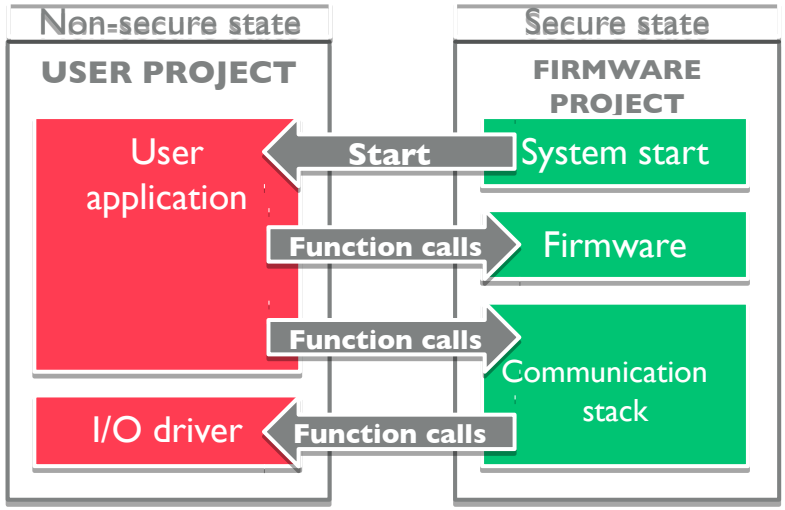
BitSec: secure microkernel / hypervisor

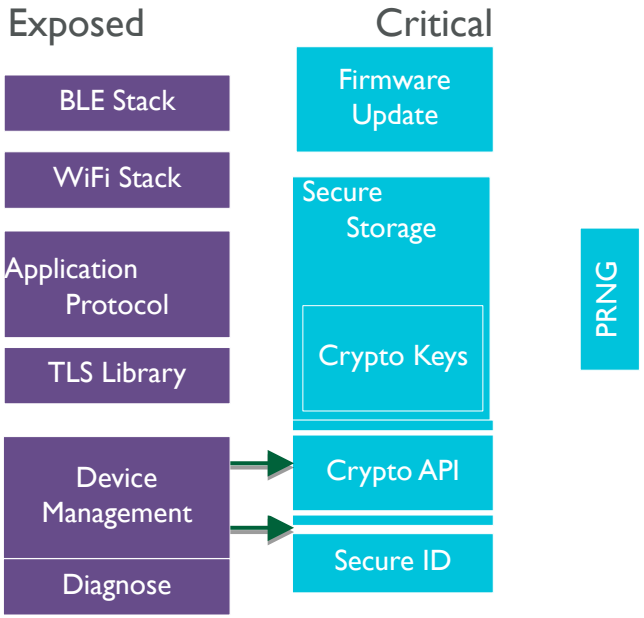
- Key assets exposed
- Key assets protected



Background of BitSec

- Learnt from uVisor, part of ARM mbed
  - Hardware-enforced security sandboxes
  - “Princle of Least Privilege”
  - Boxes are protected against each other and malicious code is contained
  - Per-box access control lists (ACL)
  - Restrict access to selected peripherals
  - Shared memories for box-box communication
- but, BitSec is lightweight and faster





Properties of BitSec

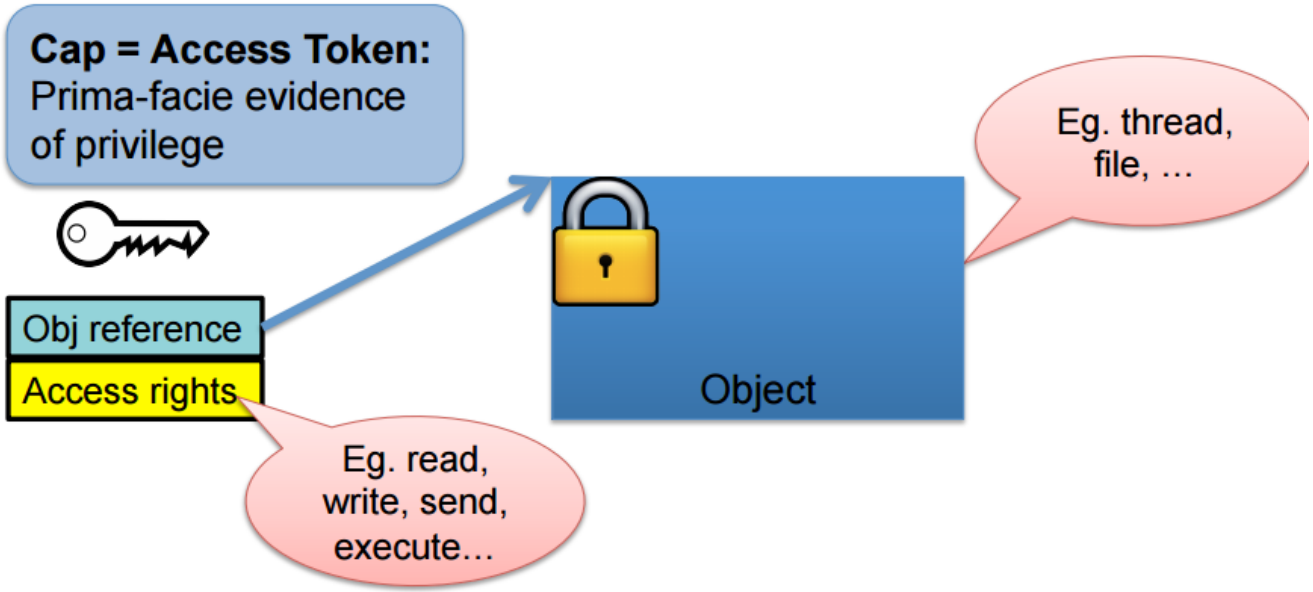
- ARMv7-M friendly: efficient application isolation
  - designed to use the ARMv7-M MPU for isolation
  - Ready for ARMv8-M TrustZone enablement
- third-generation microkernel
  - heavily inspired by seL4
- Focuses on minimality and security,
- Expresses all authority through explicit capabilities,
- Moves other mechanisms with security implications outside the kernel,
- explicitly targets systems with between 16 and 200 kiB of RAM. 2K LoC

Basic Concepts

- Object-oriented
  - object bundling together state and operations
- Capability-oriented
  - use of a capability, or key
  - object reference and a set of rights
- Messaging-oriented
  - single efficient message-transfer operation called IPC
  - operate on kernel objects
  - communicate between application tasks.

Capabilities

- without holding additional authority, programs can only perform three operations on a key
  - Copy the key into a different key register
  - Send a message to the object designated by the key
  - Receive a message from the object designated by the key

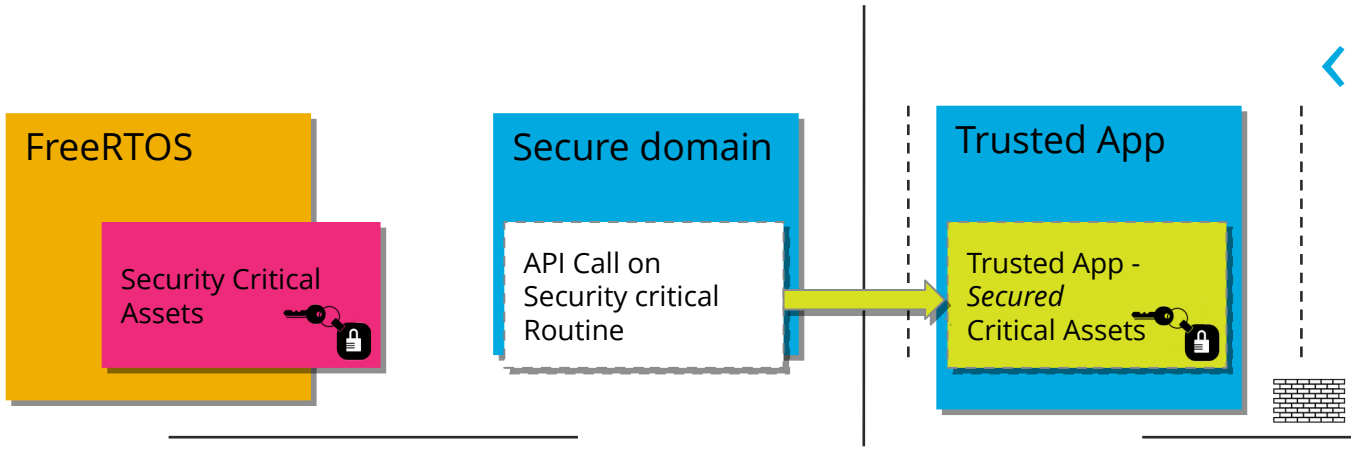


BitSec System Calls

- similar design as seL4
  - send, receive, yield
- IPC
  - synchronous rendezvous messaging model
  - messages are sent from one object to another directly
  - without being buffered in the kernel
- Copy key
  - Reads a key from one of current Context's Key Registers
  - Writes a duplicate of it into another

Case Study: RTOS Integration

- context switch latency between FreeRTOS tasks: 2x overhead
- RTOS on BitSec gains several features that are missing from the ARM Cortex-M3/M4 port
  - memory-protected environment
  - Ability to run entirely in unprivileged code
  - run a hybrid system
    - RTOS drivers + (trusted) native BitSec drivers



- Isolated space for handling high value assets

Virtual interrupts for guest OS

- Messages Model Supervisor Calls
  - Task and Interrupt Contexts share access to a Gate
    - called the System Gate (SG)
  - RTOS sends BitSec IPC messages through SG
    - Requesting a context switch
    - Enabling/disabling interrupts
  - Interrupt context holds Service Key to task context
- Context Switches Multiplex the Task Context
- Message Dispatch Loop Multiplexes the Interrupt Context

Conclusion

- Minimizing TCB is vital for building secure IoT systems, and L4 based designs bring temporal isolation, asymmetric protection, safe bounded resource sharing achieved through scheduling contexts, criticality, and temporal exceptions.
- ARM Cortex-M processor enables highly deterministic real-time applications to develop high-performance low-cost platforms, and F9 microkernel utilizes Cortex-M advantages to build the efficient and secure TCB.
- The value of open source is the community made up of people who have dedicated their time and their life to see its success. So, commercial adaptation is feasible.

Reference

- From L3 to seL4: What Have We Learnt in 20 Years of L4 Microkernels? Kevin Elphinstone and Gernot Heiser, NICTA/UNSW
- Microkernel Construction"

  http://os.inf.tu-dresden.de/Studium/MkK/
- Microkernel-based Operating Systems

  http://www.inf.tu-dresden.de/index.php?node_id=1314
- Getting maximum mileage out of tickless, Intel Open Source Technology Center
- F9 Microkernel ktimer, Viller Hsiao