Trusted Computing Base Platform Development

TCBPD

Table of Content

OPERATING SYSTEM

- An **operating system** (**OS**) is system software that manages hardware, software resources, and provides common services for computer programs with the help of the kernel.
- It provides protection and security.
- In addition, users can interact directly with the operating system through a user interface, such as a command-line interface (CLI) or a graphical UI (GUI).

OS TYPES

- General purpose operating system: - Examples could be Windows, Linux, Mac, Unix

- Mobile operating system: - E.g. Google Android and Apple iOS.

- Embedded operating system: - The associated computing device only does one major thing, so the OS is highly stripped down and dedicated to both performance and resilience.
  E.g. home digital assistants, automated teller machines (ATMs), airplane systems, retail point of sale (POS) terminals etc.

- Network operating system: - intended to facilitate communication between devices.
  E.g. Windows NT/2000, CISCO, Linux, Sun Solaris

- Real time operating system: - When a computing device must interact with the real world within constant and repeatable time constraints.
  E.g FreeRTOS and VxWorks.

KERNEL

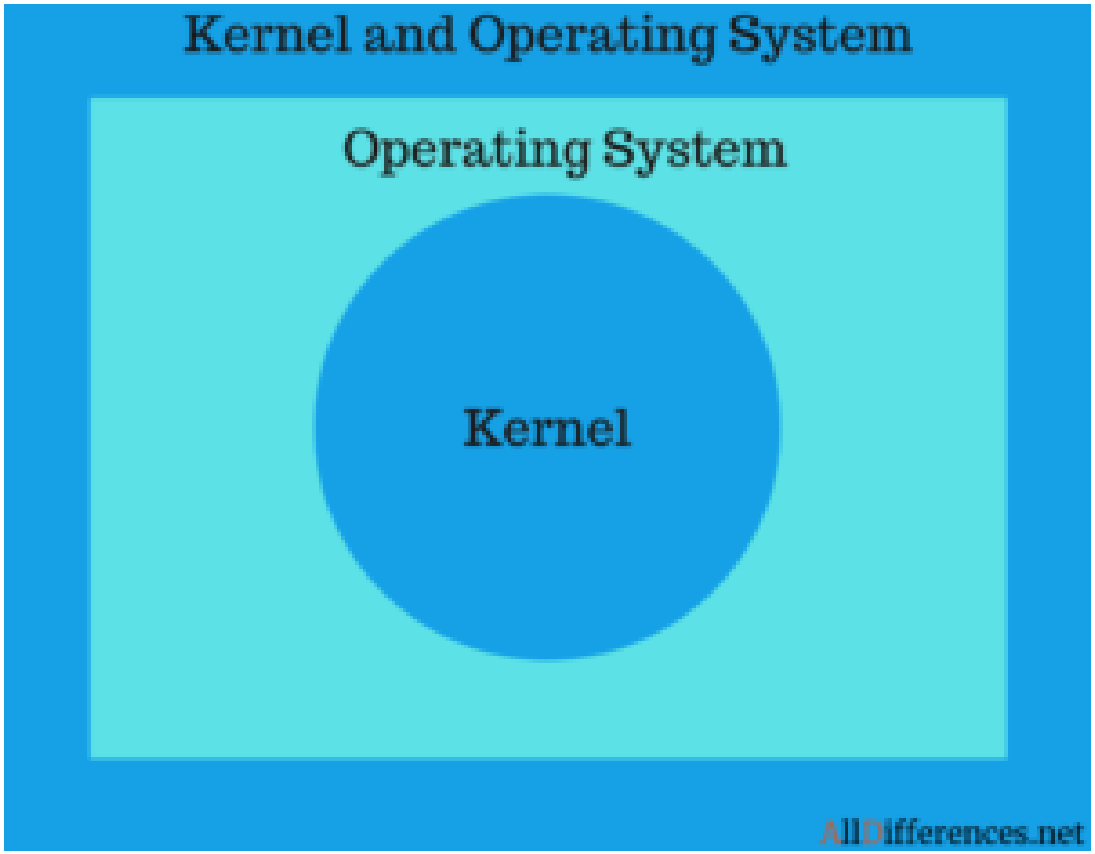
It is the core of an operating system and has complete control over everything in the system.

Some of the Kernel functionalities

- **Access Computer resource:** A Kernel can access various computer resources like the CPU, I/O devices and other resources.
- **Resource Management:** It is the duty of a Kernel to share the resources between various processes.
- **Memory Management:** Every process needs some memory space. So, memory must be allocated and deallocated for its execution. All these memory management is done by a Kernel.
- **Device Management:** The peripheral devices connected in the system are used by the processes. So, the allocation of these devices is managed by the Kernel.

KERNEL Con…

While all systems need operating system to function, similarly all operating systems need kernel to run.

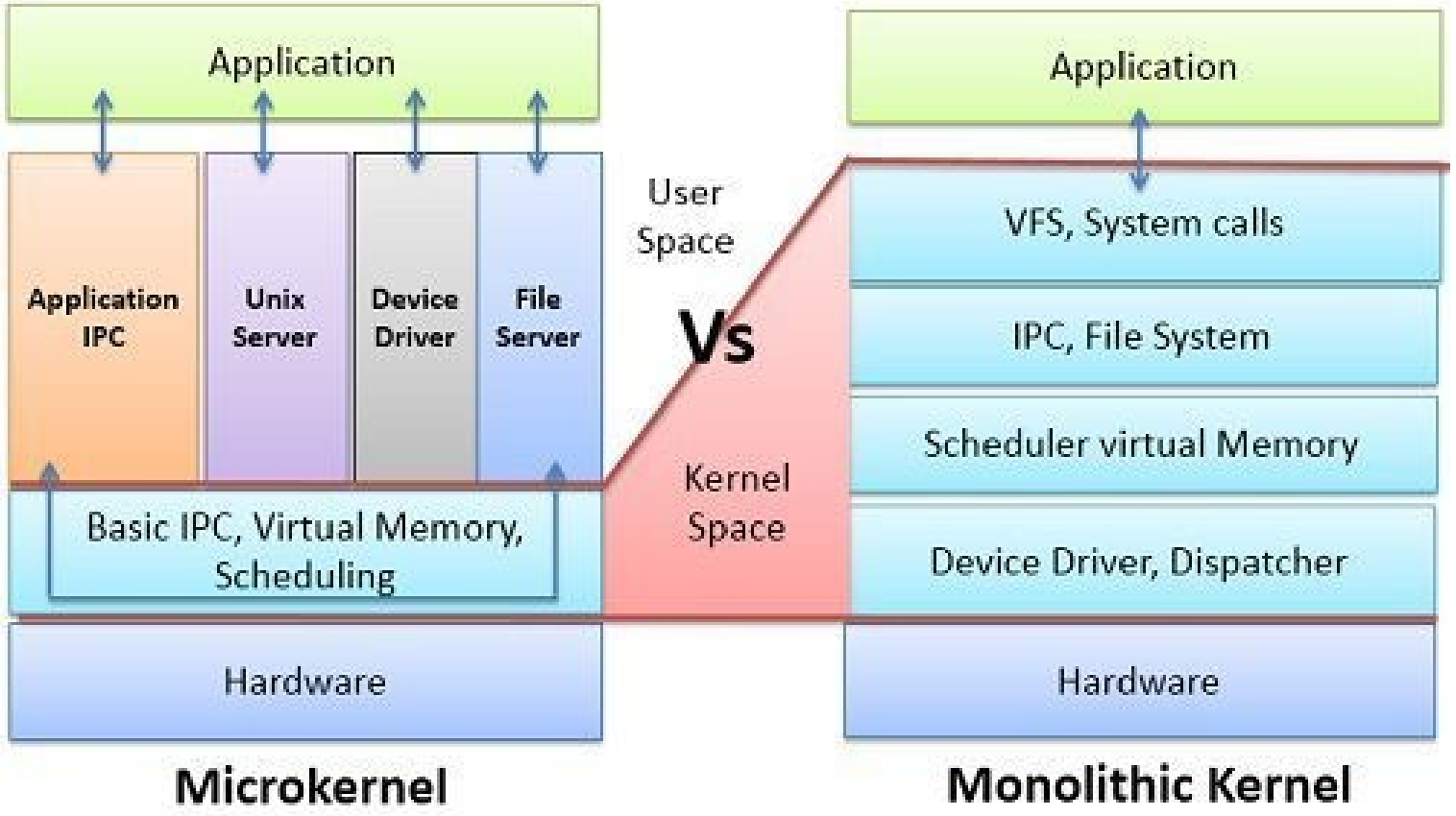


KERNEL TYPES

- **Monolithic Kernel: -** the entire operating system runs as a single program in kernel mode.
- **Microkernel**: - is a software which contains the required minimum amount of functions, data, and features to implement an operating system.
- **Hybrid Kernel**, **Nano Kernel**, **Exo Kernel**.

KERNEL TYPES Con...

Micro-kernel vs Monolithic kernel



Note: User space vs kernel space is all about privileges on hardware and software resources.

Trusted Computing Base

- The trusted computing base of a computer system is the set of all hardware, firmware, and/or software components that are critical to its security, in the sense that bugs or vulnerabilities occurring inside the TCB might jeopardize the security properties of the entire system.
- You're in Kernel space means you are inside TCB.

WHY USE OS?

- Easy for software development and use.
- Without an operating system, every application would need to include its own UI, a comprehensive code to handle all the low-level functionalities such as disk storage, network interfaces and so on.
- Without the OS, all kind of computers, mobile phones, network switches, servers, smart TV etc. became hard to function.
- Most importantly almost each and everything is directly or indirectly related to it. E.g. Cyber security

Microkernel

Microkernel-based operating systems come in many different flavors, each having a distinctive set of goals, features and approaches. Some of the most common cited reasons are flexibility, security, and fault tolerance.

Microkernel types

- **MINIX 3**: - A free, open-source, operating system designed to be highly reliable, flexible, and secure. It is based on a tiny microkernel running in kernel mode with the rest of the operating system running as a number of isolated, protected, processes in user mode

- **seL4**: - A high-assurance, high-performance microkernel developed, maintained and formally verified by NICTA and owned by General Dynamics C4 Systems.

- **Escape**, **F9**, **M³**, **Fuchsia**, **Genode** etc.

Focus Area

- Because of the above reasons and others, we will mainly focus on seL4.

- What is it & Why we choose seL4?
- How to use seL4 & its prerequisites?

seL4

seL4 = Secure Embedded L4 Microkernel

- It is a **microkernel** which contains the following functionalities
  - ✓ Address Spaces
  - ✓ Scheduling
  - ✓ Threads
  - ✓ Thin IPC implementation between isolated servers
- It is also a **hypervisor**
  seL4 supports virtual machines that can run a fully fledged guest OS such as Linux.
- seL4 is **proved correct**
  seL4 comes with a formal, mathematical, machine-checked proof of implementation correctness, meaning the kernel is in a very strong sense “bug free” with respect to its specification.

SeL4 Con...

- **seL4 is provably secure**
  The kernel guarantees the classical security properties of confidentiality, integrity and availability.
- **seL4 improves security with access control through capabilities**
  Capabilities are access tokens which support very fine-grained control over which entity can access a particular resource in a system. They support strong security according to the principle of least privilege (also called principle of least authority, POLA). This is a core design principle of highly secure system, and is impossible to achieve with the way access control happens in mainstream systems such as Linux or Windows.

SeL4 Con...

Note: seL4 is still the world's only OS that is both capability-based and formally verified, and as such has defensible claim of being the world's most secure OS.

- **seL4 ensures safety of time-critical systems**
- seL4 is the world's only OS kernel (at least in the open literature) that has undergone a complete and sound analysis of its worst-case execution time (WCET). This means, if the kernel is configured appropriately, all kernel operations are bounded in time, and the bound is known.

SeL4 Con...

- **seL4 is the world's most advanced mixed-criticality OS**

  seL4 provides strong support for mixed criticality real-time systems (MCS), where the timelines of critical activities must be ensured even if they co-exist with less trusted code executing on the same platform.

- **seL4 is the world's fastest microkernel**

  Traditionally, systems are either (sort-of) secure, or they are fast. seL4 is unique in that it is both.

SeL4 Con...

- **seL4 Is a Microkernel and a Hypervisor, It Is Not an OS**

  seL4 is a microkernel, and designed for generality while minimizing the TCB.

seL4 Verification

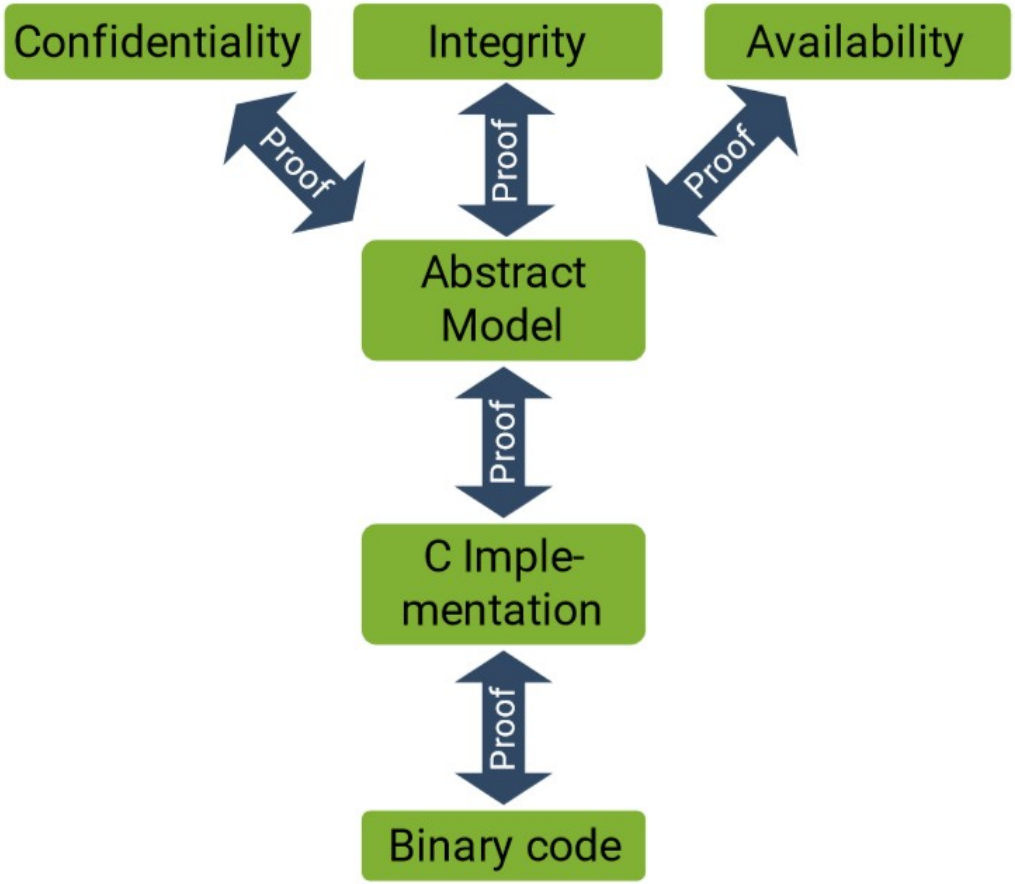


Figure 3.1: seL4's proof chain.

Formal Verification

- C is not a formal language; in order to allow reasoning about a C program it has to be transformed into mathematical (abstract) logic (HOL). This is done by a C parser written in Isabelle.
- The parser (Isabelle) defines the semantics of the C program, and gives it meaning in HOL according to this semantics. It is this formalization which we prove to be a refinement of the mathematical (abstract) model.
- The proof means that everything we want to know about the kernel's behaviour (other than timing) is expressed by the abstract spec, and the kernel cannot behave in ways that are not allowed by the spec.

seL4 verification Con...

Functional correctness

- The core of seL4's verification is the functional correctness proof, which says that the C implementation is free of implementation defects.
- The functional correctness proof then says that the C implementation is a refinement of the abstract model, meaning the possible behaviors of the C code are a subset of those allowed by the abstract model.

seL4 verification Con...

Translation validation

- Having a bug-free C implementation of the kernel is great, but still leaves us at the mercy of the C compiler. Those compilers (we use GCC) are themselves large, complex programs that have bugs. So we could have a bug-free kernel that gets compiled into a buggy binary.
- we prove that the binary is a correct translation of the (proved correct) C code, and thus that the binary refines the abstract spec.

seL4 verification Con...

seL4 enforces security properties

- Confidentiality: seL4 will not allow an entity to read (or otherwise infer) data without having been explicitly given read access to the data;
- Integrity: seL4 will not allow an entity to modify data without having been explicitly given write access (from unauthorized alteration) to the data;
- Availability: seL4 will not allow another entity to prevent, interrupt and delay authorized use of resources .

Proof assumptions

seL4 verification works with the following assumption

- Hardware behaves as expected.
- The kernel is at the mercy of the underlying hardware, and if the hardware is buggy (or worse, has Trojans),then all bets are off.
- The spec matches expectations.
- If the spec is informal or non-existent, then it is obviously impossible to precisely reason about correct behavior.
- One can reduce this risk by proving properties about the spec. The advantage of formal reasoning is that you know exactly what this gap is.

Proof assumptions Con…

- The theorem prover is correct.

  In reality this is the least concerning of the three assumptions. The reason is that the Isabelle/HOL theorem prover has a small core (of a few 10 kSLOC) that checks all proofs against the axioms of the logic.

- Proof status and coverage

  seL4 has been or is being verified for multiple architectures: Arm, x86 and RISC-V.

WhY seL4?

This implies that the seL4 kernel lacks the following:

- ✓ Buffer overflows
- ✓ Null point dereferences
- ✓ Memory leaks
- ✓ Arithmetic overflows and exceptions
- ✓ Undefined behavior
- ✓ General pointer errors

General Considerations to take before seL4 use

- The first step should be to identify the critical assets you need to protect. The aim should be to minimize this part of your trusted computing base, and make it as modular as feasible, with each module becoming an seL4-protected CAmkES component.
- The other important preparation is to check availability and verification status of seL4 on your platform. You must not make any verification claim if you are using a kernel that is not verified for your platform, or that is in any way modified.
- You furthermore will need to assess whether the available user-level infrastructure is sufficient for your purpose. If not, then this is where the community may help you. There are companies specializing in providing support for seL4 adoption.

DEMO *Boeing ULB autonomous helicopter*

The typical approach is what we call incremental cyber-retrofit, a term coined by then DARPA program director John Launchbury.

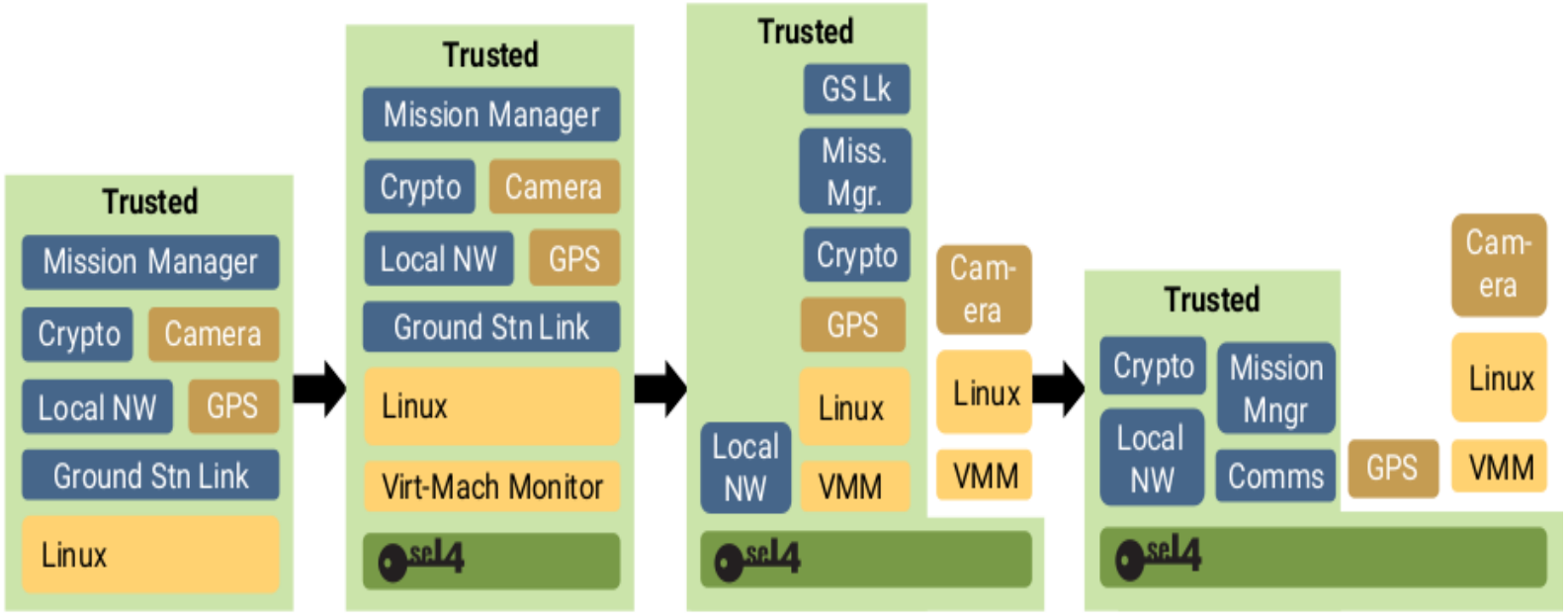


Figure 7.1: Incremental cyber-retrofit of the Boeing ULB mission computer during the DARPA HACMS program.

DEMO Con...

1. As Figure 7.1 shows, starts out by simply putting the whole existing software stack into a virtual machine running on seL4.
Obviously this step buys nothing in terms of security and safety, it only adds (very small) overhead. Its significance is that it provides a baseline from where to start modularising.
2. The next step broke out two components: The particularly untrusted camera software was moved to a second VM, also running Linux, with the two Linux VMs communicating via CAmkES channels.
At the same time, the network stack was pulled out of the VM and converted to a native CAmkES component, also communicating with the main VM.

DEMO Con…

3. The final step pulled all other critical modules, as well as the (untrusted) GPS software, into separate CAmkES components, removing the original main VM.
The final system consisted of a number of CAmkES components running seL4-native code, and a single VM running just Linux and the camera software.
The upshot was that while the initial system was readily hacked by the professional penetration testers hired by DARPA, the end state was highly resilient. The attackers could compromise the Linux system and do whatever they wanted with it, but were unable to break out and compromise any of the rest of the system.

How to implement seL4?

- Since L4 does not provide any specific model of operating system services such as process management, memory and address space management, access control, etc.
- This task is left up to a supervisory OS running in user-mode on top of the microkernel. In our case this OS is Iguana.
- Since the microkernel approach to designing systems is partially based on the componentization of system services, component-based software engineering is a particularly attractive approach to developing microkernel-based systems.
- There are presently two main component frameworks for seL4, both open source: CAmkES and Genode. However Genode, can't use all of seL4 security and safety features.

Iguana

- The basic security model provided by CAmkES is based on Iguana's capability model and involves controlling and restricting access to components.
- Iguana provides a single non-overlapping address space that is shared by all threads. In Iguana the concerns of memory protection and memory translation (i.e., providing address spaces) are separated.
- Iguana provides a client-server model of interaction. Applications and operating system services run as Iguana servers and interact with each other using IPC.

CAmkES

- Traditional methods for developing embedded systems are resulting in increasingly unreliable embedded software. The complexity of the software increases, the methods and technologies used to develop have not changed significantly.
- Overcoming this problem brings Component-based software engineering (CBSE) technique.
- CBSE provides a way to compose systems from independent, well-defined building blocks.
- Organizing software in this way helps to provide structure and improves the re-usability of code. It also improves flexibility by allowing components to be added and removed from a system (possibly at run-time), as well as allowing components developed in different languages to interact with each other.

CamkES con...

- microkernel approach to designing systems is partially based on the componentization of system services,
- The relationship between our component architecture and the underlying microkernel based operating system is tight integration with the operating system results in two requirements.

First, the architecture must directly make use of any mechanism provided by the OS (this includes inter-process communication, memory management and protection) and not re-implement similar mechanisms.

Second, all mechanisms provided by the component architecture must be efficient enough that they can be used by operating system components without creating significant performance penalties for the rest of the system.

CAmkES Architecture

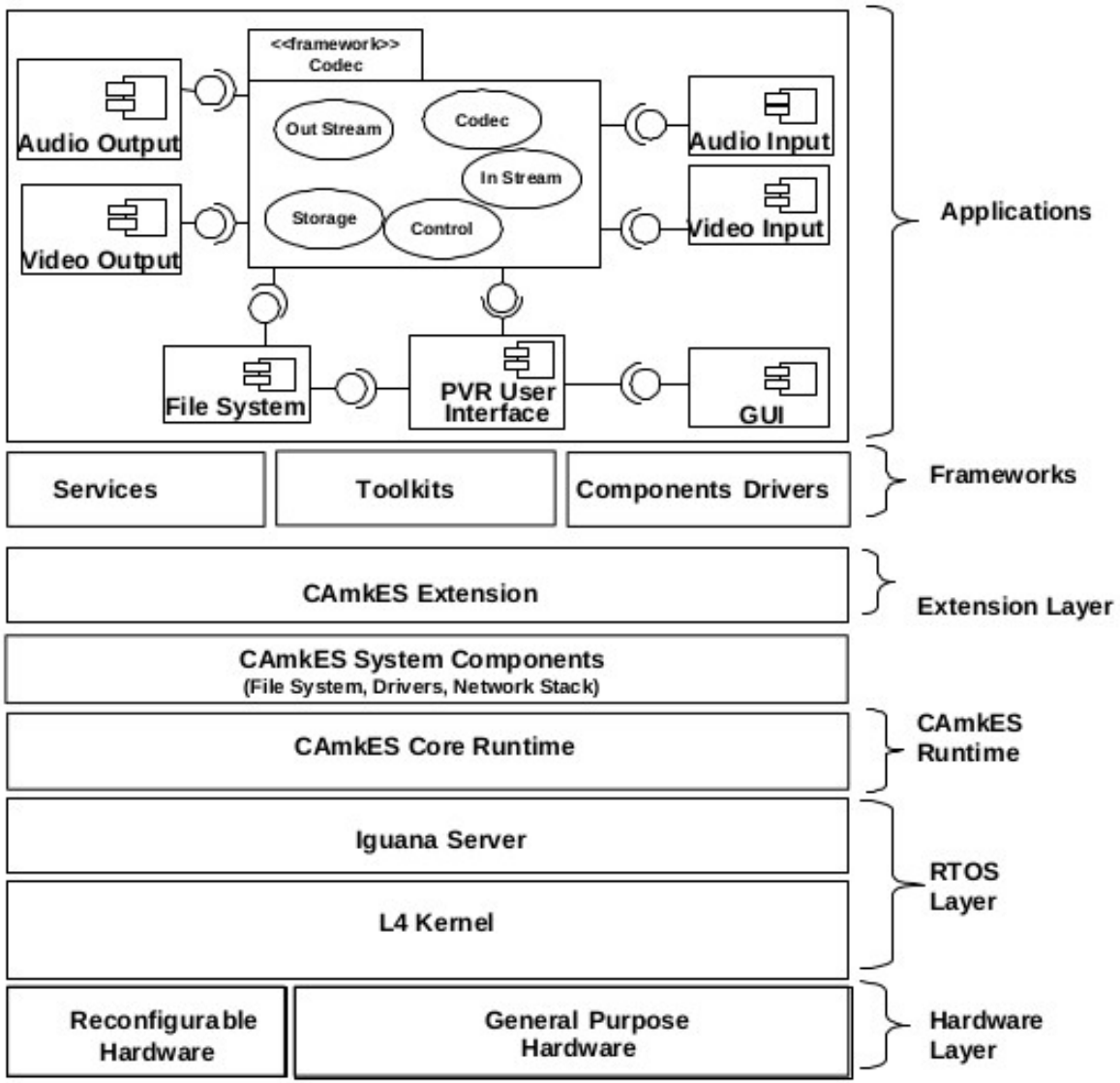


Fig. 1. The CAmkES layered architecture.

Mapping CAmkES to Iguana

- CAmkES components are generally placed in separate Iguana protection domains and are implemented as separate Iguana servers.
- This provides proper encapsulation and prevents other components (or processes) from purposefully or inadvertently accessing a component's internals.
- CAmkES RPC interfaces map indirectly to Iguana interfaces. Unlike CAmkES interfaces, Iguana interfaces act as units of protection rather than encapsulation.
- Compound components do not map directly onto any Iguana entities. Instead, any access to a compound component's interface is routed directly to the component actually implementing that interface.

Loading and initializing a CAmkES-based system

- A boot image containing L4, Iguana, and CAmkES components is loaded into the system's memory.
- L4 starts and loads the Iguana user-mode server.
- Once loaded, the Iguana server proceeds to load and initialize its services.
- After basic services such as chipset drivers, naming, etc. have been loaded, a CAmkES loader routine is run. The loader routine is responsible for loading all components, initializing them and establishing connections between them.
- Connection establishment involves the creation of Iguana sessions, allocation of shared memory sections and the distribution of capabilities according to the component configuration specifications.
- Finally, once all components and connections have been initialized, component dispatch and control threads are started.

?

- WHERE ARE WE NOW ?
- AND OUR NEXT TARGET ?