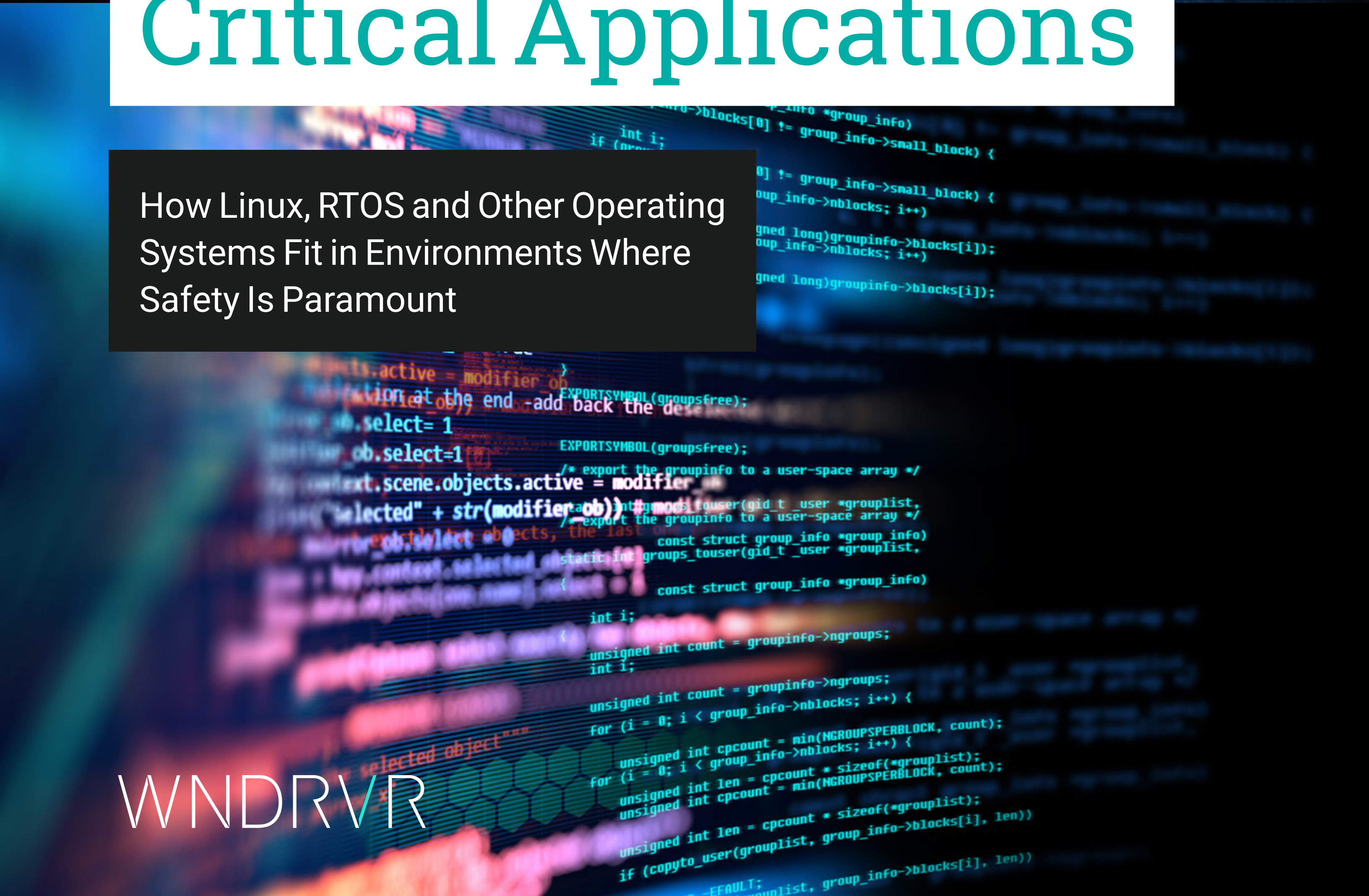
Balancing Operating Systems for Safety-Critical Applications

How Linux, RTOS and Other Operating Systems Fit in Environments Where Safety Is Paramount



WNDRVR

Executive Summary

Linux, as open source software, has become the backbone of the global technology infrastructure. Powering the majority of the world's servers and supercomputers — and mobile devices, via Android — Linux has become omnipresent in the modern digital ecosystem.

Linux, as open source software, has become the backbone of the global technology infrastructure. Powering the majority of the world's servers and supercomputers — and mobile devices, via Android — Linux has become omnipresent in the modern digital ecosystem.

It makes sense to reuse, to build on collective knowledge and drive economies of scale through common technology use. This enables organizations to accelerate development and avoid the pitfalls of older approaches.

This instinct to leverage the good work done in other industries has led to a lot of recent buzz around the idea of using Linux in automotive applications. Again, this approach makes sense — to a point.

But when it comes to safety-critical applications, Linux as we know it is not an appropriate operating system. Efforts to force-fit Linux into safety-critical situations have been attempted for more than a decade, and little success has been achieved.

Instead, automotive companies should use a software architecture that leverages best-fit solutions for the differing requirements of automotive software. Linux can be used for many non-safety–critical systems. Android has long since won the battle for in-vehicle infotainment. And proven and certified real-time operating systems (RTOSes) such as VxWorks® are best for safety-critical functions. This best-fit approach can even be accommodated on a single SoC, via high-performance hypervisors such as Wind River® Helix™ Virtualization Platform.

Importantly, this hybrid architecture can also achieve the goals that companies are looking for in next-generation software architectures and cloud-native paradigms, leveraging the advances developed for other industries in ways that make the most sense in the safety-critical world.

Understanding Linux and Safety

Safety-critical applications are essential in industries such as automotive, aerospace, healthcare, and industrial automation, where system failures can lead to severe consequences, including loss of life, significant financial loss, or environmental damage.

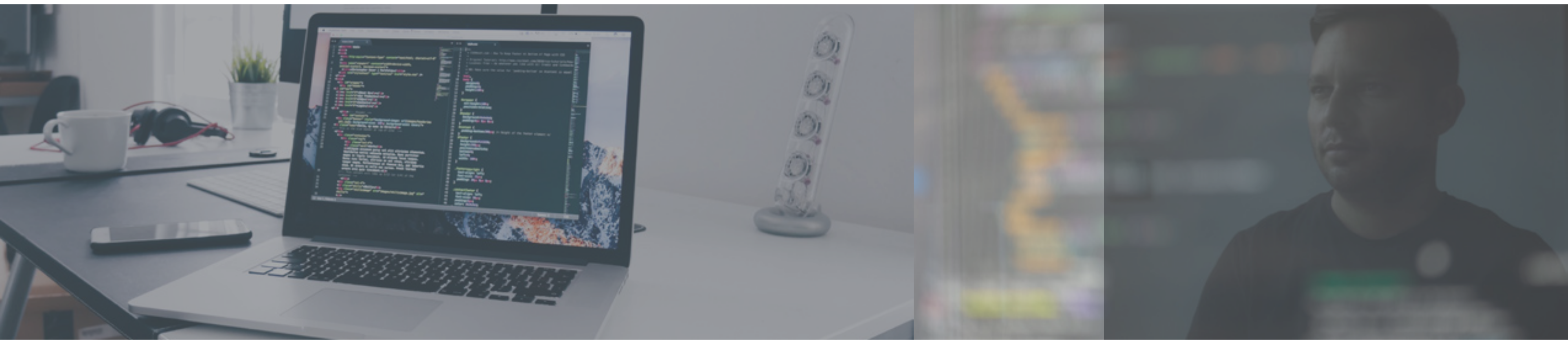
The operating system (OS) plays a crucial role in these applications by managing hardware resources, ensuring real-time performance, and maintaining system reliability and stability. A well-balanced OS for safety-critical environments must provide robust safety features, support real-time operations, and facilitate the certification process to meet stringent industry standards such as ISO 26262 or DO-178C. By carefully balancing these factors, the OS ensures that safety-critical applications perform reliably under all conditions, mitigating risks and safeguarding both users and the environment.

Wind River has a 20-year history with Linux, including 20 years of releasing Yocto Linux for embedded systems and 10 years of releasing CentOS Linux and Debian Linux for enterprise systems. In 2024, we founded a Debian (Enterprise and Edge) community project called eLxr. We are cofounders of multiple Linux open source projects, including the Open Handset Alliance (which became Android Linux), Yocto Linux, eLxr Debian Linux, and StarlingX (based on Debian Linux). This legacy includes decades of open source contributions and leadership. True open source principles, leadership, and Linux are core to Wind River's DNA.

This means that we have a deep understanding of Linux's strengths, and we are a vocal proponent of them. We continue to offer an expanding range of Linux operating system products for our customers. We value Linux's vast and active development ecosystem, the collaboration among developers across industries, and the ability to avoid vendor lock-in. Because of these strengths, we know that Linux plays a role in the software-defined vehicle.

We also know Linux has significant challenges when it comes to safety certification. These include:

- **Certification complexity:** Certifying Linux for safety standards such as ISO 26262 is extremely complex. The Linux kernel is vast, with millions of lines of code, so validating every component for safety compliance is challenging. Certification requires rigorous documentation, testing, and verification, which is difficult for large, general-purpose systems such as Linux. The Linux kernel contains more than 30 million lines of code. This number varies with each release as new features are added, bugs are fixed, and old code is removed. Compare that to a real-time operating system, which may have only 1 million lines of code.
- **Lack of pre-certified distributions:** Unlike some proprietary RTOSes that come with pre-certifications or have been designed with safety standards in mind, mainstream Linux distributions do not inherently meet ISO 26262 or similar safety standards. Companies need to invest significant time and resources in the certification process, often developing their own safety-certified versions of Linux or using specialized variants — which then end up as single-sourced, proprietary versions.



- **Long-term maintenance and support:** Ensuring long-term support and security updates for specific kernel versions can be challenging in a safety-certification context. Many Linux distributions do not maintain older versions for the long periods — often 10 or more years — required by safety-critical industries.
- **Need for continuous certification:** Linux, especially in its upstream and community-driven variants, undergoes frequent updates. These changes often include security patches, new features, and bug fixes. In a safety-critical context, every update must be thoroughly reviewed, tested, and possibly certified to ensure that it does not introduce new risks or compromise safety. The ISO 26262 certification process demands that any modification to a certified system must be reassessed for its impact on safety. This means that with every new kernel release or patch (even when security related), the system integrator must evaluate whether the patch affects the system's safety functions. This can lead to a cycle of continuous recertification or revalidation, driving up costs and delaying system updates.
- **Indeterminate timing:** Linux, by default, is not a real-time operating system. Without careful configuration (using real-time patches), it may exhibit nondeterministic behavior, which is unsuitable for safety-critical applications that require consistent timing guarantees.

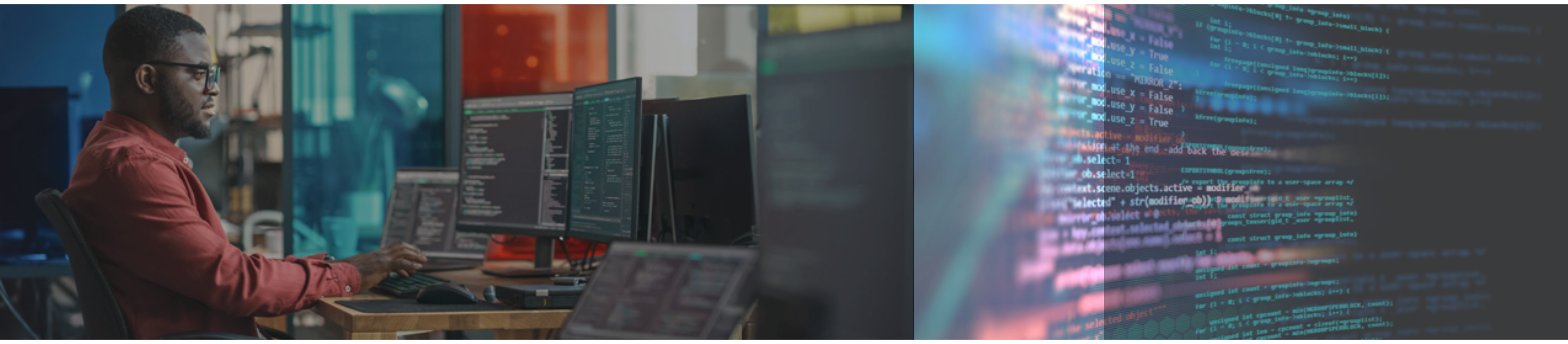
There are also several security risks that become difficult to manage in a safety certification setting, including:

- **Open nature of Linux:** The open source nature of Linux means that its codebase is visible and modifiable by anyone. While this transparency is often a strength for general security because Linux benefits from community scrutiny and contributions, it also makes Linux a target for attackers who can study the code for vulnerabilities. This can be problematic in safety-certified systems, since security breaches could compromise safety functions. A best-of-breed solution would provide the customer with the source but protect that source from general availability.
- **Lack of centralized governance for security:** Unlike proprietary operating systems whose security patches and updates are controlled and managed by a single vendor, Linux's decentralized development model relies on a global community of contributors. This can result in delayed responses to security threats and inconsistent application of patches across different distributions, making it harder to ensure that the Linux kernel used in safety-critical systems remains secure over time.
- **Impact on certification:** Safety certifications such as ISO 26262 require rigorous validation and verification processes to ensure that the system operates safely under all conditions, including during security threats. Ensuring that Linux meets these stringent security requirements necessitates extensive testing and risk assessments. Every security patch introduces potential changes in the system's behavior, necessitating revalidation efforts that can be costly and time-consuming.

Hard Roads to Safety

Despite these substantial obstacles, there are several initiatives attempting to make Linux suitable for safety-critical applications. We believe these efforts will not meet automotive functional safety requirements. These approaches include:

- **Brute force:** To comply with existing functional safety standards — assuming one can define a "safe usage" of each API without changing the implementation — the Linux kernel and Linux packages can be safety certified using traditional software certification methodologies. The problem is that the certification comes at a huge cost, based on the number of lines of code in open source Linux. Unless this cost can be leveraged across multiple projects, reusing the same Linux code, then these costs will likely not be reasonable, and we've already seen companies struggle with this.
- **Strip down:** The idea here is to reduce the code size of the Linux kernel and required Linux packages to restrict the code base to only what is required to run the software for the device — thereby reducing the level of safety certification work to be done. Unfortunately, this means forking Linux and creating a Linux-based kernel and root file system customized to specific hardware or applications. This opens the door to technical debt and significant maintenance costs over time. Every time there is an update from the Linux community, an analysis will have to be performed, along with customization of updates, bug fixes, and security fixes — and then the software will have to be recertified.
- **Kernel modifications:** Safety-critical systems often require deterministic behavior, and this means modifying the kernel for real-time performance. However, every kernel modification introduces new potential safety risks that must be assessed and tested, complicating the development and certification process.
- **Proprietary lock-in:** Many safety-certified systems attempt to mitigate the code change issue by "locking in" a specific version of the Linux kernel and freezing updates to ensure that the certified system remains stable. However, this approach can expose systems to security risks over time, as they may miss out on critical security patches and vulnerability fixes. Additionally, in the end, this approach results in a proprietary operating system — the very thing they were trying to avoid — only now it is more complex and costly.
- **Proven in use:** Some functional safety standards allow organizations to apply for a "proven in use" claim for software that has been used commonly with existing products that have been in operation for a significant period of time, and for which you can provide historical evidence. While these claims are more commonly used for low-complexity software, given the extensive use of Linux, in theory a case could be made for the use of Linux in functional safety applications. However, this hasn't been done successfully to date, and this approach skirts the rigorous certification process that is in place to make products safer.
- **Safety monitor:** One approach is to use a safety-certified "safety monitor" to monitor applications running on non–safety-certified Linux. While there is recent activity around this approach, Wind River implemented a similar system 10 years ago and found that there are several downsides. First, the safety monitor itself has to be sophisticated and proprietary. Second, it impacts performance. And third, it requires additional drivers in the hypervisor that are proprietary.



Modification of Functional Safety Standards

Another approach is to modify or amend the certification standards themselves to mitigate the risks of using of preexisting open source software in functional safety environments. One such approach that has received a lot of recent attention is the work in progress for the upcoming update to the ISO-26262 safety standard for automobiles (third edition). This proposal, defined in ISO/PAS 8926, seeks to define an alternative methodology for handling open source software in the ISO 26262 standard.

The main concept in this Publicly Available Specification (PAS) is the creation of classes for "preexisting architectural elements" based on the provenance (origins) and complexity of the software. Based on these ratings, a software qualification guidance would be defined. The theory is that Linux would be rated as having a reasonable provenance due to its extensive testing and long history, allowing well-used and tested Linux software to be accepted for use in ISO 26262 certified products without the need for the extensive software certification process required for "new" software.

While this approach could provide a more cost-effective path to use of open source software, existing safety certification standards are in place for a reason, and they have been effective in instilling confidence that functional safety in software can be achieved. Modifying the standards could undermine that confidence in the software. The update of the ISO 26262 standard with these proposed modifications is due to be ratified in 2027.

The Cost Is Too High

So, is it possible to make Linux work in safety-certified applications? Yes. Wind River has developed such an operating system multiple times in its history, under contract to automotive and aerospace companies. We continue to support customers who desire customized Linux solutions. However, these experiences educated us on the very high cost, high complexity, and low feasibility of a general-purpose Linux approach for safety-certified systems.

In the end, we realized that the goals articulated for a next-generation software development environment could be met with a best-fit approach to safety-critical systems.

Achieving the Same Benefits

Let’s take a step back and consider why there is industry pressure to force-fit Linux into a safety-critical environment, despite the substantial challenges. What are the benefits that organizations are trying to achieve with that approach — and how can they be achieved without changes to Linux?

- **Developer familiarity:** Linux is attractive to developers due its widespread use across industries. However, leveraging that familiarity can also be achieved through POSIX® interfaces. And since 2020, widely used OCI-compliant containers and Kubernetes have been supported in real-time operating systems.
- **Multi-use capability:** Linux is perceived to provide the ability to handle a wide range of workloads simultaneously, including non–safety-critical tasks such as infotainment, connectivity, and artificial intelligence (AI). We agree, with the caveat that safety-critical functions are best run on an RTOS.
- **Open source–driven cost efficiency:** Linux is perceived as “free,” enabling an ecosystem where the user can choose to “roll their own” Linux or select a commercially supported version. However, as illustrated above, when employing Linux in safety-certified systems, “free” goes out the window. A best-fit approach allows companies to leverage the lower costs associated with Linux in non–safety-critical applications, where it makes sense.
- **Continuous innovation:** Linux is perceived to be an innovative, fast-moving platform — and rightly so, due to the crowd-sourced innovation driven by thousands of developers. This is why Wind River believes Linux has a place in the architecture. In safety-critical applications, however, that level of innovation can be equally achieved through OCI-compliant containers, Kubernetes, and cloud-native microservices employed in an RTOS, just as it would be in Linux.

Wind River’s Approach: A Hybrid Architecture with a Certified RTOS

A hybrid architecture that combines Linux, Android, and an RTOS — all running on a Type 1 hypervisor — is a powerful approach to meet the demands of modern safety-critical systems. This architecture provides the best possible solution, embracing Linux for its advantages, VxWorks for hard real-time and safety-critical certification, and other operating systems such as Android for infotainment and UI applications.

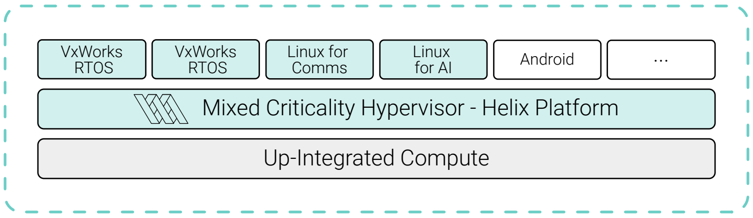
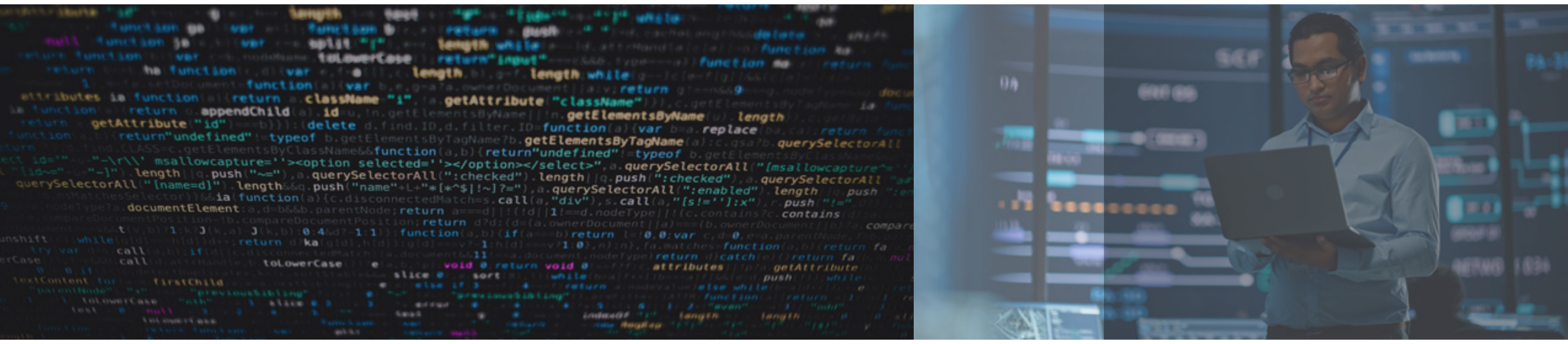


Figure 1. With the right hypervisor, mixed-criticality operation can be achieved on the same hardware.



By running Linux, Android, and the RTOS on separate virtual machines managed by a hypervisor, safety-critical functions can be isolated from non–safety-critical ones. The components look like this:

- **The hypervisor** provides isolation between the systems, meaning that any failure or security breach in the Linux or Android domain is less likely to affect the RTOS running safety-critical applications.
- **The RTOS** delivers the hard real-time guarantees needed for safety-critical applications (e.g., ASIL-D certification for ISO 26262), ensuring that timing and reliability requirements are met.
- **Linux** offers robust networking, device support, and cloud connectivity.
- **Android** excels in user-facing features and app ecosystems.

By leveraging Linux for connectivity and cloud services, Android for the user interface, and RTOS for critical timing, this architecture can scale across different products and use cases, from consumer devices to high-end automotive systems.

The separation of systems allows Linux and Android components to be updated or patched without affecting the RTOS. This is particularly beneficial for maintaining security while preserving the stability of safety-critical systems.

The architecture can support mixed-criticality workloads, allowing the system to run both high-performance, consumer-centric applications and safety-critical processes on the same hardware. This reduces overall system complexity and cost by avoiding the need for separate hardware for different functions. This consolidation is especially beneficial in industries such as automotive, where reducing hardware complexity can drive down costs significantly.

In short, rather than trying to force-fit Linux into solving every possible OS use case, we choose a best-of-breed, systemic approach, benefiting from the best of all contributors.

The Future Is Exciting

A hybrid architecture that integrates Linux, Android, and an RTOS through a Type 1 hypervisor provides a powerful, flexible solution for safety-critical systems that need advanced user interfaces and connectivity. The ability to combine real-time guarantees from the RTOS with the rich functionality of Linux and Android offers the best of both worlds.

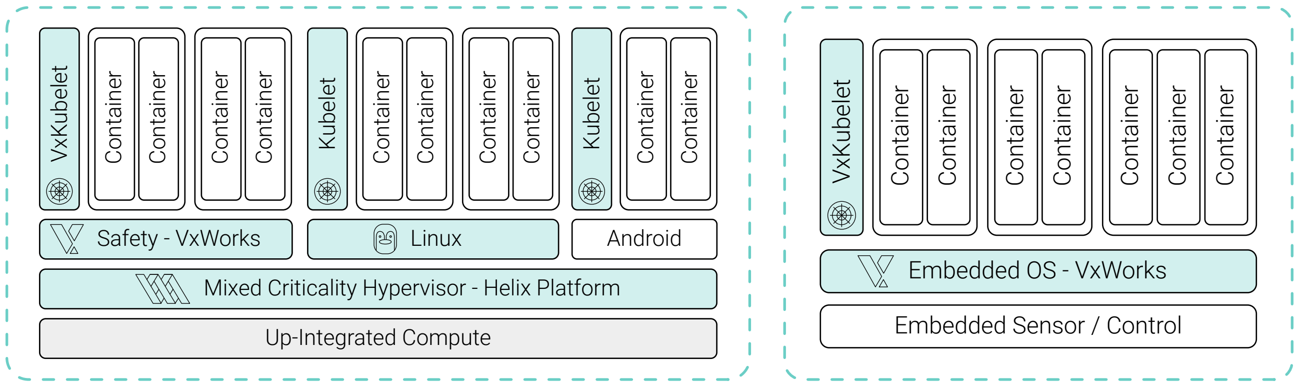


Figure 2. Kubernetes functionality can be integrated across operating systems via Kubelet nodes.

Wind River's approach makes the overall cost of safety certification manageable, affordable, and predictable over the entire lifecycle of a product. It fits well with the trend in the automotive industry to up-integrate multiple functions onto fewer compute platforms. And it can be implemented using software that is readily available today and has been accepted as a proven approach by safety certification authorities in many existing projects.

As we move forward, we can enable the "cloudification" of these embedded systems by leveraging the latest technologies from Linux, including OCI-compliant containers and Kubernetes. A hybrid system in which Android, Linux, and an RTOS all support containers and Kubernetes worker functionality (via Kubelet nodes) presents a highly attractive proposition, especially in terms of development, deployment, scalability, and lifecycle management. This setup essentially puts all operating systems into a unified, cloud-like architecture, with orchestration enabled through Kubernetes.

This path avoids the substantial challenges with adapting Linux while still converging embedded systems with modern cloud-native paradigms, making it an appealing solution for industries that need to manage diverse workloads, ranging from safety-critical real-time operations to infotainment and AI-based applications.