Virtual Platform definition for Automotive Hypervisor Environments

Read intro below, and/or the preparation work at the bottom of this page. Then look at actual Document Work and Release page.

The Automotive Virtualization Platform

To allow further success of hypervisor environments in automotive it is essential that all vendors are able to provide compelling guest runtime environments that make the usual automotive I/O devices available. The essential devices could be defined and agreed upon by the industry. If such a set of devices and device features is defined a document can be crafted that **defines these devices and their features** that in their **combination create a virtual platform**.

PORTABLE

- The virtual platform definition would allow the development of virtual machine guests that like appliances in the enterprise world, could be moved among different hypervisor systems without (minimal) modification.

SPECIFIED

- A **clear and detailed specification** is required to achieve *true portability*, and to give *real support* to vendors.

Hand-waving about some virtual architecture is not our goal. The specification shall enable efficient reuse and collaborative progress among companies in this industry.

REUSE

- Specifications like these are best developed as open specifications and just like open-source code, it should stand on the shoulders of previous work.

A virtual platform specification should be built upon already published work, where VIRTIO is the most prominent standard we have found, but add to it if this is not enough. It seems that the automotive industry may needs to extend some areas. At minimum it should review and bolster support for the parts most needed in the automotive industry, clarify which parts are required vs optional, and this work result should (will) one way or another make sure that different automotive initiatives combine into common agreements.

The project group has discussed and found consensus on how that automotive virtual platform specification can achieve this:

- Principles of meta/delta-specification (formal specification with reference/reuse)

For an automotive virtualized platform, VIRTIO is primarily a starting point although increasing in breadth over time. Participating companies in this Hypervisor Group are also involved in proposing new functionality in future VIRTIO specifications, to cover audio, multimedia, and hardware accelerators and other unique hardware devices often found in automotive environments.

Instead of developing additional domain or even vendor specific device frameworks and models a collaborative development could greatly reduce the individual development of project efforts and thus spur the adoption of hypervisor based environments in the automotive sector.

Additional efforts definitely need to be spent in the area of audio virtualization and the in virtualization of co-processors like DSPs, image processors and codec accelerators. Developing a standard model for the diverse buses found in cars like CAN or ethernet AVB could also be additional fields of investigation.

In the work done here (see table below) we categorize device types and ensure that each need has been thoroughly analyzed so that as an industry we can truly define the right specification, to be applicable in >95% of the typical situations.

About VIRTIO

The VIRTIO standard aims to provide a standardized interface and device models for device para-virtualization in hypervisor environments.

With development going back until 20XX the virtio device model was first introduced for the educational "lguest" hypervisor and became the default I /O virtulaization method used with qemu/KVM and recently the default model used by many cloud providers. The virtio devices have been partially standardized by the OASIS standardisation body in 2015 with the VIRTIO 1.0 specification which describes the transport layer and a limited set of device models.

The currently standardized device models are: network, block, console, entropy, memory ballooning and SCSI devices. Additionally to the formally standardized devices several additional devices exists as "implemented" devices such as GPU, input, 9pfs, vsock and crypto. Some of which are currently in the process of standardization.

Virtio relies on a dma-like memory model meaning that all allocations are handled by the "driver" part of the device and the "device" implementation can directly access the allocated buffers for reading or writing. This enables a resource saving and fast operating mode. Metadata is transported using so called virt-queues that resemble ring-buffers. Depending on the architecture used, different transport and device discovery modes are supported: PCI for x86, mmio for ARM and channel-IO for s390. These transports are geared toward the most efficient implementation per CPU architecture and allow for efficient implementations depending on the environment.

In recent years some hardware devices, like network controllers and NVMe based storage systems have evolved to be similar or compatible with the VIRTIO protocol, to allow hardware assisted I/O virtualization using para-virtualized device models.

VIRTIO Benefits

VIRTIO's main benefit for the automotive industry lies in it's sheer existence and operating system support. The fact that standardized device models exists allows for multiple compatible implementations of both driver and server parts of the system. The ubiquity of VIRTIO in cloud and enterprise virtualization makes drivers readily available in all major operating systems which keeps driver maintenance effort to a minimum.

Due to the driver defined memory allocation model, vendors can choose to limit the resource usage and define safety properties to their own requirements without impeding the standardized model and stay interoperable with existing device implementations.

The DMA-like nature of the devices allows for high-performant implementations that the easily compete with hardware assisted I/O virtualization models while still providing relative ease of implementation.

Many vendors of automotive grade hypervisors have already adopted virtio based devices into their system offerings due to the above mentioned reasons and the benefit of building upon these open source technologies has greatly improved the availability of commodity devices like network and block storage.

GENIVI and AGLs Role

(Opensynergy's proposal)

Vendor neutral industry bodies like GENVI and AGL can act a forum between hypervisor vendors, users and the hardware manufactures in a form that allows open collaboration in development and maintaining a standard platform definition.

The regular events can be used as occasions for interoperability testing and standard steering. The participation in the OASIS-VIRTIO committee can be delegated to such organization to voice the automotive industries concerns and advice in the technical committee.

GENIVI has in the past maintained domain specific APIs and standards can easily act as a body that makes sure the standard is not only maintained but also advanced as new technologies evolve.

AGL can provide the necessary collaboration with the upstream kernel project and the Linux Foundation which in-turn opens up for collaboration with the key industry players in cloud and enterprise computing.

Links

- **VIRTIO v1.0 specification.** VIRTIO is the starting point for our investigation into the definition of an Automotive-Wide virtualization platform
- **VIRTIO v1.1 specification**
  - What's new in **VIRTIO 1.1** (presentation 1, presentation 2 FOSDEM 2018 + video)
- **Latest development: VIRTIO git master**

Evaluation Process for existing specifications (e.g. VIRTIO)

For each topic:

1. Discuss and write down the automotive requirements
2. Read *VIRTIO* chapter

3. Decide if *VIRTIO* is appropriate and complete for requirements (Gap Analysis)
4. Write down what the industry needs to do to close the gap

Consider topics not yet listed (e.g. unique automotive requirements)

Note also: Bottom of this page has a "brainstorm" list of criteria to consider.

Virtual Device Categories

Notice

This table is now somewhat outdated and not used to drive the work any longer.
Please review the working document for the up to date status, or the JIRA kanban view which is used for planning the content.
NOTE: All the project's JIRA tickets are public but the JIRA kanban view for can only be viewed when logged in (This is a JIRA limitation we cannot change)
Therefore to participate in the planning of AVPS development we suggest you request a login by selecting "Sign Up" in the JIRA login page

The key challenge for defining a shared virtual platform definition is to first identify the various device driver types such a platform must provide, and to evaluate if existing work so far (e.g. VIRTIO) covers what the automotive industry needs:

(Virtual) Device	Explanation	Champion + interested people	Completeness / Applicability evaluation ✅⚠❌	Comments and discussion	Spec complete	Include in draft 1	Ticket to track completion
Block Storage	Flash/Disk /persistent storage	Kai	✅	Included also automotive persistence requirements.	✅ ready for review	Yes	
Network	Access to (shared) physical ethernet and guest-to-guest communication		✅	Think about writing info how to share a physical network in practice (Create bridge between virtual device and physical)	✅ ⚠ vsock should be moved to separate section? Agreed. ⚠ WiFi seems to be not well covered. Shall we wait on including it?	Yes ⚠ except WiFi - leave comment that it is work in progress)	
Console	Text terminal input	Gunnar	✅		✅ ready for review	Yes	☑ HV-9 - AVPS:Console CHAPTER DONE
crypto	Access to cryptographic services (hardware accelerated)		✅ With new features, it is enough. We also added some	Now includes: • RNG - *OK. just clean up discussion text.* TEE ✅ RPMB - *open question, leave it as WIP in draft.* Crypto acceleration ★ new text is OK for draft.	✅ ready for review *Discussion part needs cleanup*	Yes	

GPU	Graphics hardware	Matti/Dmitry	See Graphics Virtualization, VIRTIO GPU Operation Highlights pages ✅ Draft spec – requirements written ⚠ Uncertainty (and lots of ongoing development) around 3D APIs - Vulkan progress, etc.	See Graphics Virtualization page and requirements in spec draft	★ Still a moving target (3D). This is reflected in specification	✅ 2D 3D: Proposal: include a discussion but requirements are not in Draft. **Dmitry Morozov** please finish according to this. 3D requirements that are not accepted upstream were dropped. Check status of EDID introduction.	
Input	Traditionally keyboard/mouse /etc - for automotive = expanded?	Matti	Matti	Now part of VIRTIO 1.1 Mouse/touch events may need to remap coordinates in combined virtual systems but interface may still not be affected by this.	❌	Yes	
vsock	Communication between guest (VM) and host (hypervisor)		✅	Covered in networking chapter - to be put in its own (sub)chapter.	✅ ready for review	Yes	
Filesystem **9pfs and other**	9P = protocol to expose host (hypervisor) file systems to the guest. FS=filesystem.	Gunnar	**Completeness:** Protocol: ✅, VIRTIO spec: ⚠ (see comment) **Need in Embedded /Automotive:** ✅ None? Can we find a use-case? **Applicability**: ✅ For what it does, seems ok. But might not be really needed and therefore "not applicable". Is there something else/more needed?	Links: Virtio 1.0 spec : {PCI-9P, 9P device type}. Kernel support: Xen/Linux 4.12+ FE driver Xen implementation details A note on its documentation/definition not being very precise A set of man pages seemingly defining P9? intro, others QEMU instruction how to set up a VirtFS (P9). Example/info how to natively mount a 9P network filesystem, Source code for 9pfs FUSE driver	✅ ready for review *(cut down chapter, should be OK)* ❓ OK to have such a verbose chapter? May be some more work...	Yes	
vIOMMU	IOMMU coordinates of DMA devices' connection to memory.	Dmitry	See IOMMU Summary page **Applicability**: ✅ Limit guest devices' scope to access system memory during DMA ✅ Enable scatter-gather accesses due to remapping (DMA buffers do not need to be physically-contiguous) ❓ Nested virtualization. Any use-cases for automotive? ✅ Group conclusion: Not needed nested virtualization - however there are still two levels because applications in guest are used to set up IOMMU (protection between applications) and then the VMs themselves are another level. These levels drive the need for a virtualized IOMMU layer (and/or hardware support for the same)	ARM is actively working on the specification, more features are coming. Nested virtualization? The use of Linux Containers inside a VM was mentioned. That in itself is not really nested virtualization. Namespace-based *containers,* is just a kernel feature providing separation independent of a hypervisor. However, Kata Containers is an approach to tie Linux containers into a hypervisor layer, making them "fully" virtualized. A theoretical situation arises that involves the use of Kata Containers on a Linux system that itself already runs in a VM. That might constitute an example of nested virtualization, but it was decided that this is not a mainstream idea, possibly not supported or feasible, and in each case likely more trouble than it is worth. "Flattening" the virtualization approach so that all units still run on one hypervisor is a likely outcome. Further research into partitioning methods is likely but for now this falls outside of a mainstream automotive virtual platform definition. We highlighted that Linux containers in their normal namespace based implementation are already a very useful system partition tool and it can be trivially applied also if the Linux kernel runs in a VM.	⚠ Chapter has been written. ⚠ Xen working on more secure implementation (memory visibility problem, security of VIRTIO approach) - want to keep this open requirement wise to have flexibility. ⚠ Latest proposal to VIRTIO was close to go in but withdrawn to address final comments. But: Need a group review of text (verbose) and consider the comments here on the left. And also VIRTIO parts have not been merged to official spec version.	❌ Not final Include discussion and what we have learned. TBD. Dmitry Morozov Artem Mygaiev Requirements to be removed?	

Audio		Matti	⚠ VIRTIO proposal is being discussed - still pending 2019-09-25	Some info on Linux/Xen code here: HVWS: Xen input and experience on Audio, Display, Input and TEE Artem Mygaiev - can this comment be removed? Should it affect the spec?	⚠ Information is quite complete. and good understanding written. Needs cleanup to become a proper chapter	Requirement set is NOT ready (merged). ★ Leave comment and/or discussion and future outlook but not requirement.	
Sensors	Automotive sensors:	Artem	✗ Not covered by VIRTIO specifically. Considering SCMI over VIRTIO as a future standard.	Artem proposed that **Systems Control Management Interface** (SCMI) protocol as a flexible and an appropriate abstraction for sensors. It is also appropriate for controlling power-management and related things. The hardware access implementation is according to ARM offloaded to a "Systems Control Processor" but this is a virtual concept. It could be a dedicated core in some cases, perhaps in others not. EPAM/Xen tried out putting code in ARM-TF, to act as this SCP. SCMI destined (?) to become a ARM-wide standard in a currently fragmented reality. Presentation attached (PDF) Upper protocol defined, but could imagine different lower transport. One mailbox-style transport is *kind-of* defined by ARM spec? Discussion if VIRTIO transport would be appropriate. A "SCMI device" type added to VIRTIO? Challenges: • Current situation in ARM is fragmented with many overlapping unique APIs across chip vendors. • Is this doable also on x86, and is it likely to be adopted? • Discuss applicability beyond "sensors" and where boundaries are drawn. Reference: • Related: ARM SCMI "Platform Design document" What about PINCTRL, and handling the many multiplexed pins in a modern SoC. Any remaining need for lower-level protocols for accessing/virtualizing hardware? --- CPUs/SoCs have "internal" sensors too. Relating to temperature and power mgmt. Some internal control tweaks for power management (core frequency / voltage) are like tiny internal actuators. Virtual access to those? Same or different APIs? Some OS have requirements that must be met by "platform" - eg. Android requires orientation sensor. ---	⚠ Good work done. Split out GPIO to separate chapter. Placeholder also for describing HW passthrough (in general) All 3 need another review and cleanup to be complete. ⚠ Consider platform requirement for sensors that must exist (for Android etc.)	No requirements possible in draft spec. Possibly some of discussion and future outlook...	
Media Acceleration (VPUP, IPU, CODEC)	*Hardware support for codec /processing*	~~Artem~~ Dmitry	⚠ Proposal to VIRTIO might come (OpenSynergy) VPU = "AI" CPU optimized for visual recognition		Gunnar AnderssonPlease check status - in VIRTIO mailing list...	✗ Not ready in time. Placeholder referencing currrent proposals.	
coprocessors and other dedicated hardware features	*Abstraction of SoC specifics* *DSPs* *Tensor processors*		✗ Not really in VIRTIO scope	Matti: virtualize functions, not devices. Gunnar: Analysis might extract some functions out of these...	✗ (not sure yet if we expect to cover it)	✗	
USB			Example Assigning Host USB device to a Guest VM in KVM, here: https://www.linux-kvm.org/page/USB_Host_Device_Assigned_to_Guest	Which use cases do we want to address? •USB 2.0 (EHCI controller) •USB 3.0 (xHCI controllers will replace ECHI) •USB C •Host only •Device Classes: • Mass storage. Enable use of USB device with volume provider • Communications (e.g. serial, Ethernet) • Human interface (e.g. keyboard, mouse) •On-The-Go (system can function as both USB host and USB device) •Hot-plug (partial support): • Static configuration of device “tree”. A device can be plugged into a port. Dynamically detect device type. • Device tree cannot grow dynamically, i.e. cannot plug in a hub	⚠ Needs update – see minutes.	✗ placeholder for future	

Other Serial devices?
 	
... and LIN bus
			❌ VIRTIO applicability needs analysis
			
			❌ Spec chapter needs to be written.
				LIN-bus:
				
				• Source code for linux-lin driver (for Linux, not necessarily virtual environment):
				• Paper by Czech Technical University & Volkswagen Group Research:
				LIN based on SocketCAN 1. OSADL article, 2. paper (PDF).
				  ◦ The paper concludes that LIN data frames are similar enough to CAN frames that it can reuse CAN software infrastructure (such as the SocketCAN standard). LIN is a serial bus, implemented with a UARTs, and therefore standard UART device drivers would be used. For virtual environments, we can rely on the same conclusions, and therefore refer to the answer given for CAN.
				  ◦ On the other hand, LIN is most popular for its simplicity / low cost (even lower than CAN) and used in very simple ECUs or to/from input devices like switches, knobs and buttons. On the larger CPU it is likely to be run by a separate dedicated microcontroller, or at least small on-chip CPU core. Therefore it can often be considered out-of-scope for the CPU that implements virtualization.
				
				UARTs are normally passed through (VM has access to memory mapped hardware) or forwarded (hardware access is done by HV and some abstract interface provided to the VMs) = virtio-console standard.
				
				SBSA specifies some access to UART but it seems tailored for debugging.
				
				Virtio console starts too late. What about early access for logging boot issues. There is an early driver for console (in Linux) - uses configuration registers as a FIFO, output only. Something nicer is desired.
				
				PL011 = ARM fast model UART controller, reference implementation in versatile-express. Provided in RPi and some other hw and virtual platforms.
				
				Unknown User (anup) seems to have added emergency-write standard to the VIRTIO spec? Follow up with Anup.
					❌ TODO
					
					LIN should not be mentioned in spec. Comment on implementation later after analysis.
					
					Fold discussion into console chapter.
					
					⚠ Uncertain what parts are specified and implemented (in VIRTIO and Linux)
						✅ Cover this in console chapter
						
						emergency-write / early debugging could be left out if we are not done with it.	

CAN			❌	virtio-can: VIRTIO-based CAN driver	❌ Decide opinion and write chapter	⚠ Possible, no firm standard. No specification - just an example driver implementation. Unknown User (anup) - can we summarize again?	
Time Sensitive Networks		~~Nikola (TSN)~~ *Need new volunteer to complete it, perhaps from GHS?*	The required features are not present in the network virtio devices as of virtio 1.0. ⚠ Is this applicable enough to move into specification as requirements? -->	Must have requirements: • IEEE 802.1AS compatible egress and ingress timestamps on ethernet frames available in the virtio consumer OS Good to have: • IEEE 802.1Q-2011 queue enhancement mechanisms available in the virtio consumer OS ◦ Interesting read about KVM: https://www.linux-kvm.org/page/Multiqueue General architectural considerations: • What if there is more than one consumer of the IEEE 802.1 AS defined network timebase on the same system?	❌	⚠ Comment on the possibility of implementing TSN is not prevented, (even without a virtual interface standard) Need some more confirmation	
Bluetooth		OpenSynergy with BT experience ?	❌ Not in VIRTIO scope	Virtualization of BT hardware might not be required. However, commenting on various system designs seems appropriate. Example: There exists an interface for virtualized audio device (virtio-sound), but Bluetooth is also an audio device (among other things...) What does this mean for how to build an architecture that (for example) uses both virtualization for audio, and bluetooth technologies.	⭐ Write at least a Discussions chapter. It is an important topic, common topic. *The HCI interface is the likely level of passthrough or virtualization. The VM should have access to HCI since it can then implement standard Bluetooth stacks on top of it.*	Not ready in time for first draft. ✅ Must be mentioned, at least a comment and future plan.	

Memory Balloon Device		Gunnar	In VIRTIO. Applicability to automative is questionable.	*May be partly applicable - I could write something here to get us started. - Gunnar* *RAM device is being discussed as a better solution later on. Proprietary protocols.*	TODO	Not strictly necessary in the first draft version	
Random Number Generator				Covered in the Crypto chapter.			
Watchdog			Very important for embedded systems... Let's see what is there and what we need to do.	SBSA has a generic interface, it should be the closest one. Aim for simple interface. Avoid VIRTIO/virt-queue type solution... http://infocenter.arm.com/help/topic/com.arm.doc.den0029c/Server_Base_System_Architecture_v6_0_ARM_DEN_0029C_SBSA_6_0.pdf	Write chapter. Adam Lackorzynski	Want to include TODO	

All related JIRA Tickets

Key	Summary	T	Created	Updated	Due	Assignee	Reporter	P	Status	Resolution
HV-43	AVPS:General Questions		Apr 19, 2021	Apr 19, 2021		Unassigned	Gunnar Andersson		NEW	Unresolved
HV-42	AVPS:ISP = Image Signal Processors		Apr 12, 2021	Apr 12, 2021		Unassigned	Gunnar Andersson		TO DO	Unresolved
HV-41	AVPS: Interrupts		Mar 01, 2021	May 17, 2021		Unassigned	Gunnar Andersson		CHAPTER DONE	Unresolved
HV-40	AVPS:References Chapter		Jun 22, 2020	May 31, 2021		Unassigned	Gunnar Andersson		CHAPTER DONE	Unresolved
HV-39	AVPS:Tensor processors and similar accelerators		May 18, 2020	Jun 08, 2020		Unassigned	Gunnar Andersson		WAIT	Unresolved
HV-38	AVPS:Communication Networks: WiFi		May 04, 2020	May 31, 2021		Unassigned	Gunnar Andersson		CHAPTER DONE	Unresolved
HV-37	AVPS:Communication Networks: TSN		May 04, 2020	May 17, 2021		Unassigned	Gunnar Andersson		WAIT	Unresolved
HV-36	AVPS: (Document) Restructure chapters		Apr 27, 2020	Jun 08, 2020		Unassigned	Gunnar Andersson		WAIT	Unresolved
HV-35	AVPS:Development Support		Apr 06, 2020	Jun 08, 2020		Unassigned	Gunnar Andersson		WAIT	Unresolved
HV-34	AVPS:Media codecs and Cameras		Feb 24, 2020	May 17, 2021		Dmitry Sepp	Gunnar Andersson		CHAPTER DONE	Unresolved
HV-33	AVPS:GPIO		Jan 20, 2020	Dec 07, 2020		Dmitry Sepp	Gunnar Andersson		CHAPTER DONE	Unresolved
HV-32	AVPS:Power Management		Jan 20, 2020	May 31, 2021		Oleksandr Tyshchenko [X]	Gunnar Andersson		CHAPTER DONE	Unresolved
HV-31	AVPS:MOST		Jan 20, 2020	Jan 20, 2020		Unassigned	Gunnar Andersson		CHAPTER DONE	Unresolved
HV-30	AVPS:CAN-XL		Jan 20, 2020	Jan 20, 2020		Unassigned	Gunnar Andersson		CHAPTER DONE	Unresolved
HV-29	AVPS:FlexRay		Jan 20, 2020	Jan 20, 2020		Unassigned	Gunnar Andersson		CHAPTER DONE	Unresolved
HV-28	AVPS:CAN		Jan 20, 2020	Feb 24, 2020		Unassigned	Gunnar Andersson		WAIT	Unresolved

HV-27	AVPS:Storage	☑	Jan 20, 2020	May 17, 2021	Oleksandr Tyshchenko [X]	Gunnar Andersson	⌄	CHAPTER DONE	Unresolved
HV-26	AVPS:Booting	☑	Jan 20, 2020	Mar 15, 2021	Kai [X]	Gunnar Andersson	⌄	CHAPTER DONE	Unresolved
HV-20	AVPS:Watchdog	☑	Jul 30, 2019	May 31, 2021	Matti Möll	Matti Möll	⌄	CHAPTER DONE	Unresolved
HV-19	AVPS:Random Number Generation	☑	Jul 30, 2019	Mar 01, 2021	Unassigned	Matti Möll	⌄	CHAPTER DONE	Unresolved

Showing 20 out of 32 issues

VIRTIO-defined devices

The VIRTIO 1.0 specification is organized a bit differently, and more generic than our detailed list above. Here is a much abbreviated table of contents for VIRTIO 1.0, just to give an overview on the most important parts. Consider, especially, the **limited types of devices.** All defined devices are under these categories only for the **1.0 version.**

2 Basic Facilities of a Virtio Device
2.4 Virtqueues
3.1 Device Initialization
3.2 Device Operation
3.3 Device Cleanup
4 Virtio Transport Options
4.1 Virtio Over PCI Bus
4.2 Virtio Over MMIO
4.3 Virtio Over Channel I/O
5 Device Types
  5.1 Network Device
  5.2 Block Device
  5.3 Console Device
  5.4 Entropy Device
  5.5 Traditional Memory Balloon Device
  5.6 SCSI Host Device

7 Conformance
7.2 Driver Conformance
7.3 Device Conformance
7.4 Legacy Interface: Transitional Device and Transitional Driver Conformance

B Creating New Device Types

When looking at a particular proposed device standard, evaluate characteristics/criteria:

Criteria (brainstorm):

- Availability?
- Is there a proposal for standard (specification)?
- Is it accepted in VIRTIO?
- Is it a de-facto standard?
- Implementation status
- ...In QEMU/Linux kernel?
- ...FOSS in a GitHub Repo?
- ...Commercial/closed-source implementations?
- ... Number of implementations?

- Complexity estimation?
- ... e.g. CAN Device class, vs GPU (need to consider large User-space
library, complex HALs,

- Performance?
- ...mostly implementation dependent? Are the technical requirements hindering implementing it efficiently, for some reason? Does it matter?

- Code Maturity?
-- ...implementation dependent, but evaluate that which exists

- Evaluate: Security aspects
- Evaluate: Functional Safety aspects

Importance for automotive use-case

- ...Is it generally applicable for many use cases or for a special case?