Apex.AI®

The vehicle OS company.

Apex.OS - A safety-certified
software framework
based on ROS 2

Jan Becker

My background

Dr. Jan Becker
CEO and C0-Founder Apex.AI
Lecturer at Stanford

2017- Apex.AI

2010- Lecturer at Stanford University

Stanford University

2009- ROS core development

ROS

2010-2014 Robotics at BOSCH



1997-2001 AD with Volkswagen

2002-2006 ADAS at BOSCH

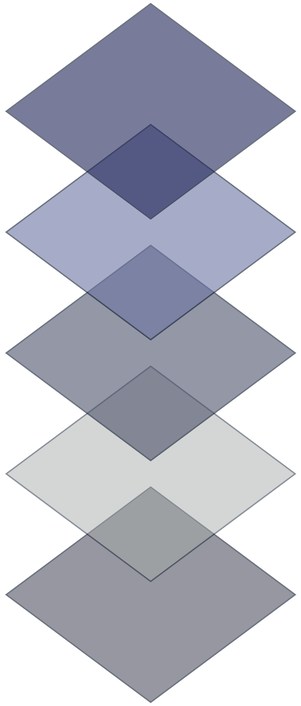

2007-2010 AD at Stanford

2011-2015 AD at BOSCH


2016-2017 FF


1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2021

The evolution from ROS 1 to Apex.OS Cert



Algorithms and Functions

Software Framework

Data Transport

Kernel, Scheduler, Driver

Hardware

2010

ROS 1

Linux

ROS 1
- Software framework for robotics
- Huge adoption in AV industry

E2017

ROS 2

DDS

Linux

ROS 1 → ROS 2
- Improved code quality
- Smaller, more optimized code
- DDS middleware
- Testing and documentation

Apex.OS

DDS

POSIX RTOS

ROS 2 → Apex.OS
- Real-time execution
- Real-time data logging
- Complete documentation
- Tests, tests, test
- Support for automotive hardware
- More tools
- 24/7 customer support

03/2021

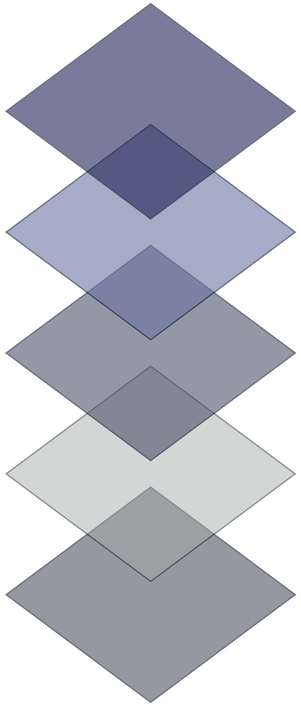
Apex.OS Cert

DDS

QNX for Safety

Apex.OS → Apex.OS Cert
- Functional safety certification (ISO 26262, SEooC, up to ASIL D)
- Fully deterministic software execution
- Apex.AI specific extensions

Apex.Middleware



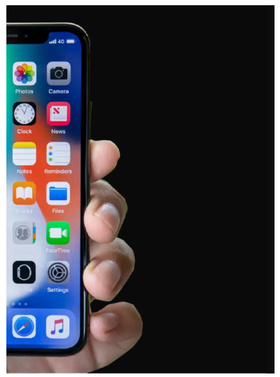
ROS 1	ROS 2	Apex.OS	Apex.OS Cert
	DDS	Apex.Middleware	Apex.Middleware Cert
Linux	Linux	POSIX RTOS	QNX for Safety

The challenge

IT and Telecommunication Industry

YESTERDAY
Hardware-defined phone

TODAY
Software-defined smartphone



Automotive Industry

TODAY
Hardware-defined vehicle

	Yesterday	Today	Today
Flexibility	Specific to compute hardware and OS	Apps are hardware-agnostic and run on every phone model	Specific to compute hardware and RTOS
Scalability	Applications don't scale	Applications scale across the whole ecosystem	Applications don't scale across domains
Labor	Function development is labor-intense	Every student can build robust apps	Function development is labor-intense
Cost	$$$ High and recurring application cost	$ Low application cost	$$$ High and recurring application cost

Automotive software does not scale to complex software systems required to solve the mega trends autonomous, connected, shared, electric mobility.

OEMs are in need of an end-to-end operating system that is robust and flexible to address all vehicle requirements (ADAS, AD, powertrain, body, chassis, infotainment).[1]

Existing prototype software does not scale to automotive production levels of safety.

1 https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/the-case-for-an-end-to-end-automotive-software-platform

The solution — SDK-like abstraction by Apex.OS

IT and Telecommunication Industry

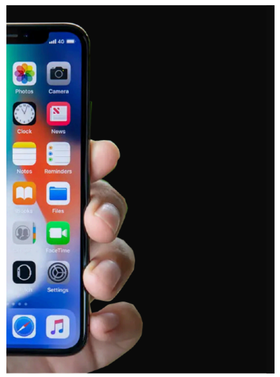
YESTERDAY
Hardware-defined phone

Few predefined apps

OS

Hardware

TODAY
Software-defined smartphone



Millions of apps enabled by SDK

SDK

OS

Hardware

Automotive Industry

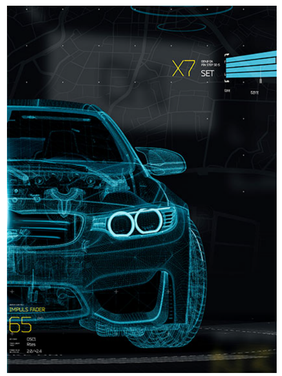
TODAY
Hardware-defined vehicle

Basic functions

RTOS

Hardware

TOMORROW
Software-defined vehicle



Complex functions

SDK

RTOS

Hardware

Android/iOS SDK has democratized App development.

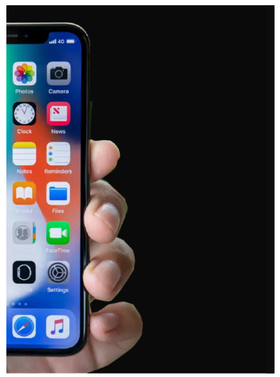
SDK-like abstraction for all vehicle domains.

The benefits

IT and Telecommunication Industry

YESTERDAY
Hardware-defined phone

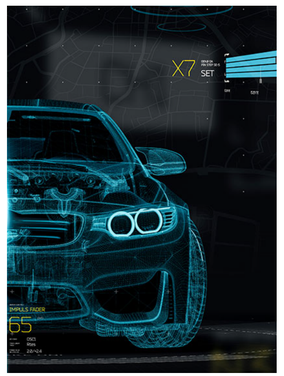
TODAY
Software-defined smartphone



Automotive Industry

TODAY
Hardware-defined vehicle

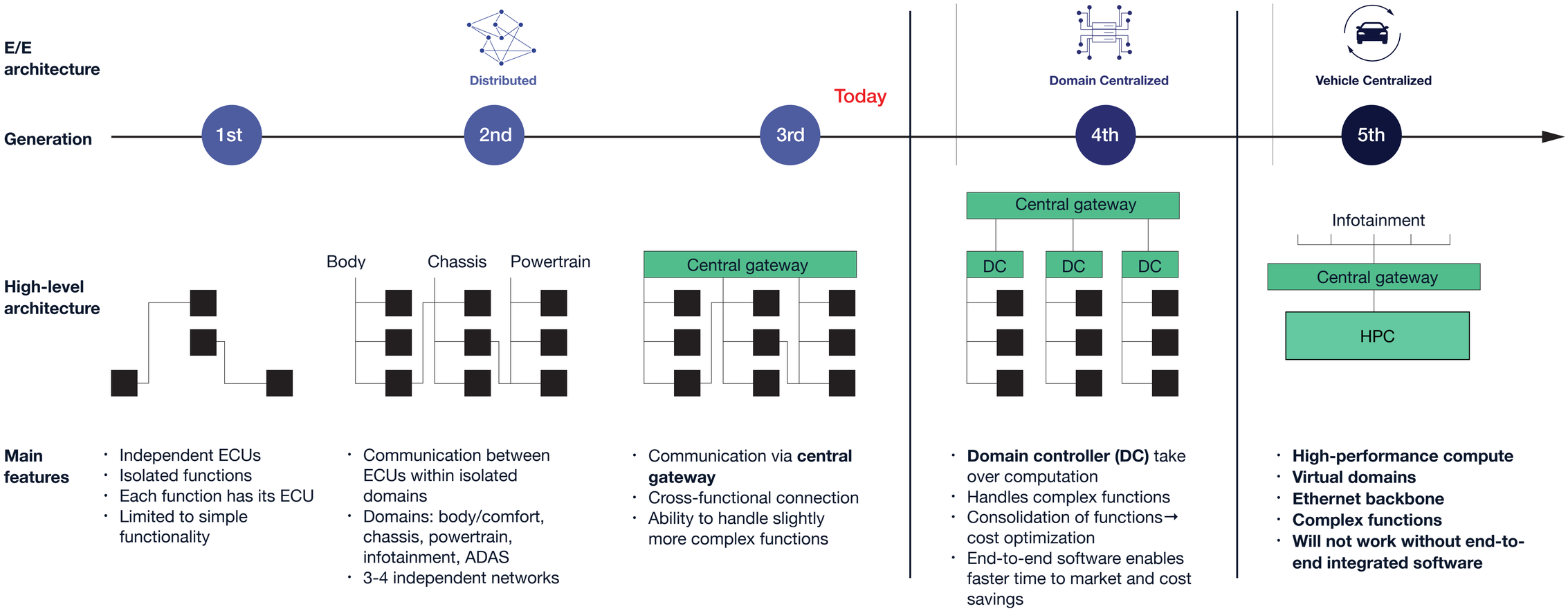
TOMORROW
Software-defined vehicle



	IT and Telecommunication Industry		Automotive Industry	
Flexibility	Specific to compute hardware and OS	Apps are hardware-agnostic and run on every phone model	Specific to compute hardware and RTOS	makes applications independent from hardware and operating systems
Scalability	Applications don't scale	Applications scale across the whole ecosystem	Applications don't scale across domains	enables software that scales massively
Labor	Function development is labor-intense	Every student can build robust apps	Function development is labor-intense	enables non-expert developers to develop reliable complex applications
Cost	$$$ High and recurring application cost	$ Low application cost	$$$ High and recurring application cost	$ reduces application cost

The situation — Automotive industry is moving to a centralized hardware architecture

But the required end-to-end operating system doesn't exist yet[1]



1 https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/the-case-for-an-end-to-end-automotive-software-platform

ROS in automotive

All major automotive and robotic players use ROS for prototyping — representing 80% of automotive ecosystem.

ROS provides access to the by far largest developer and user community.

>38,000,000 downloads
>200,000 users
>80,000 software packages
>20,000 developer
>1,000 robots and vehicles

ROS at universities

- >95% of universities use ROS for teaching and research.
- All university competitions sich as DARPA / Indy Autonomous Challenge use ROS.

Every robotics student leaving university knows ROS.

ROS is running in

- cars and trucks
- mining and construction
- agriculture
- medical robots
- industrial automation
- personal robots
- drones and eVTOL
- IoT

Largest developer and user community

Validated in many applications

Target

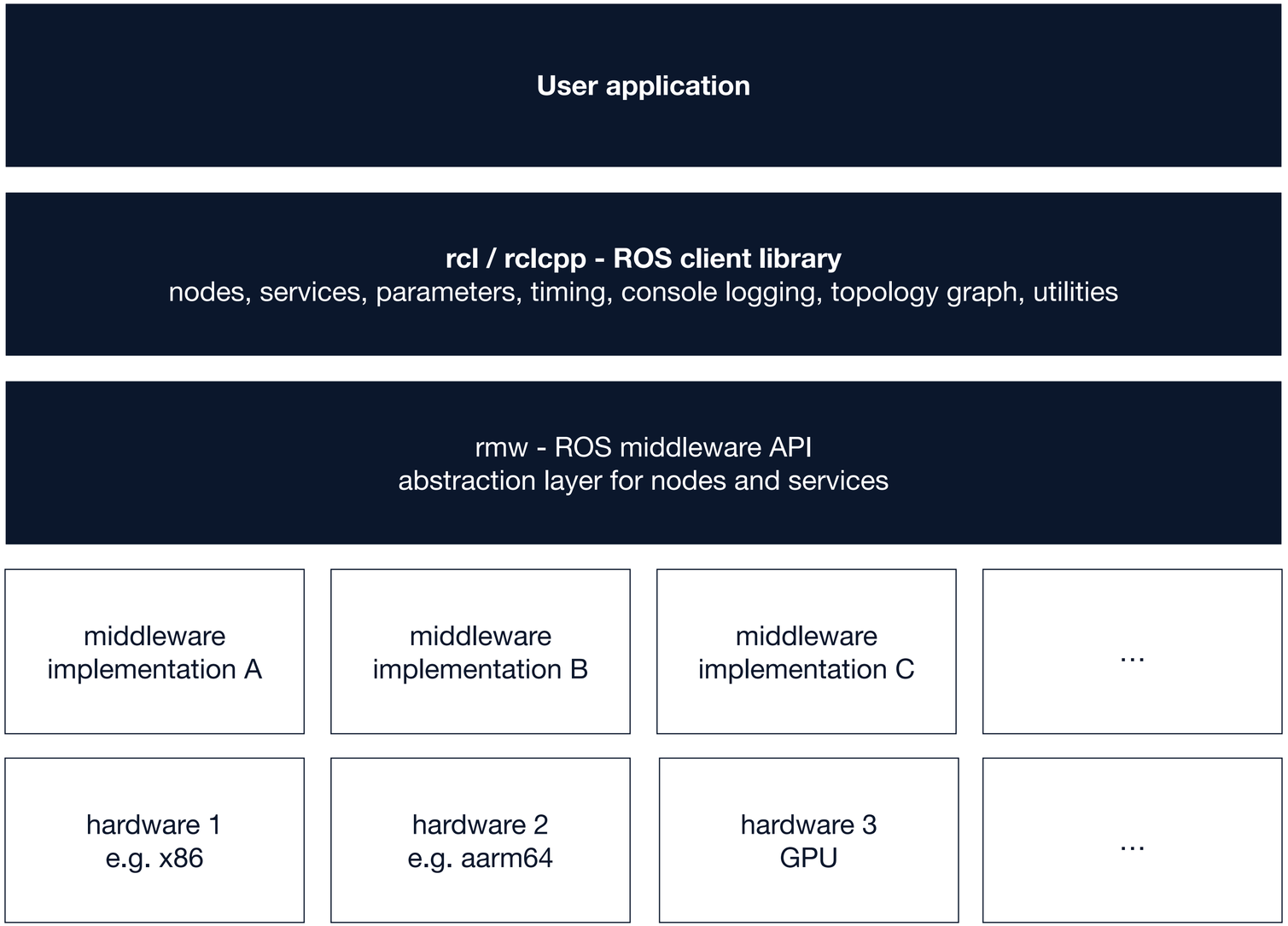
ROS

1. A standardized **software architecture with open APIs** to enable mutually compatible solutions ideally across many manufacturers, suppliers, and academia. ✓
2. An **awesome developer experience** to enable developer productivity – based on the understanding that the quality of the developer experience is directly related to their productivity. ✓
3. A **software architecture that scales** to massive software systems. ✓
4. A **software implementation based on modern software engineering practices**. ✓
5. **Abstraction of the complexity** of all underlying hardware and software. ✓
6. **Deterministic, real-time execution, automotive functional safety certification**.

Software architecture considerations

1. hardware abstraction layer
2. OS abstraction layer
3. runtime layer
4. support for various programming languages
5. non-functional performance
6. security
7. safety
8. software updates
9. tools for the development, debugging, recording & replay, visualization, simulation
10. tools for continuous integration and continuous deployment (CI/CD)
11. interfaces to the legacy systems (such as e.g., AUTOSAR Classic)
12. execution management for user applications
13. time synchronization
14. support for hardware acceleration
15. model-based development

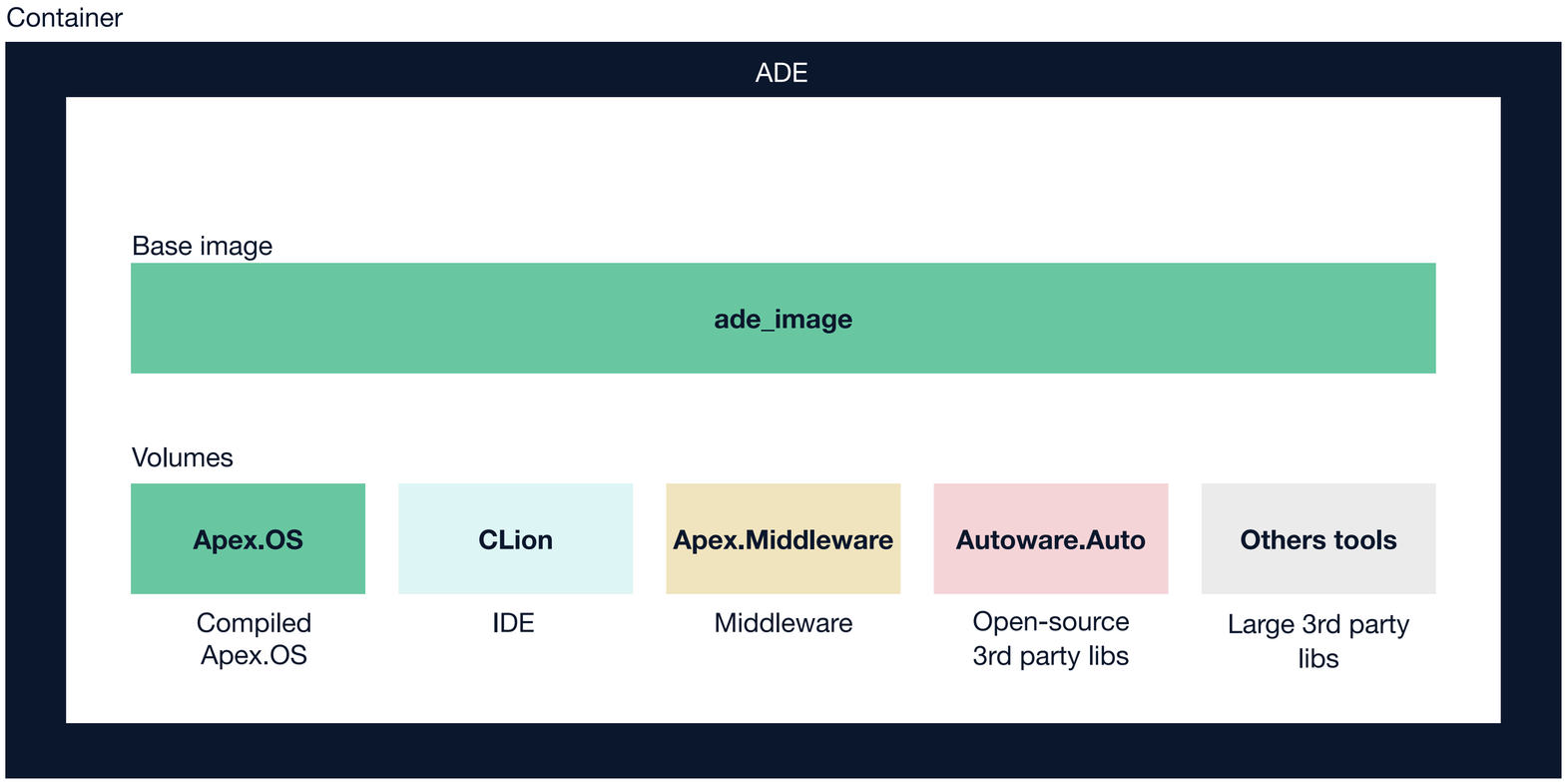
ROS architecture



Developer experience

Tools, tools, tools

1. Data Visualization
2. System visualization
3. Record and playback
4. Introspection
5. Emulation and simulation
6. Command line tools
7. Development environment
   https://ade-cli.readthedocs.io/
8. Many more at http://wiki.ros.org/Tools



Software architecture that scales

5C's principle of separation of concerns

1. Functional Entities (**Computations**) deliver the functional, algorithmic part of a system, that is, the continuous time and space behavior. A Functional Entity can be a composite entity in itself, following the same pattern of composition.
2. A **Coordinator** to select the discrete behavior of the entities within its own level of composition, that is, to determine which continuous behavior each of the Functional Entities in the composite must have at each moment in time.
3. Functional data **Communication** handles the data exchange behavior between Functional Entities.
4. A **Configurator** configures the entities within a level of composition.
5. A **Composer** constructs a composition by grouping and connecting entities.

Journal of Software Engineering for Robotics 5(1), May 2014, 17-35
ISSN: 2035-3928

The *5C*-based architectural *Composition Pattern*: *lessons learned* from re-developing the *iTaSC* framework for constraint-based robot programming

Dominick VANTHIENEN Markus KLOTZBÜCHER Herman BRUYNINCKX
Department of Mechanical Engineering, University of Leuven, Belgium

Abstract—The authors are part of a research group that had the opportunity (i) to develop a large software framework (±5 person year effort), (ii) to use that framework ("*iTaSC*") on several dozen research applications in the context of the specification and execution of a wide spectrum of mobile manipulator tasks, (iii) to analyse not only the functionality and the performance of the software but also its readiness for reuse, composition and model-driven code generation, and, finally, (iv) to spend another 5 person years on re-design and refactoring.
This paper presents our major *lessons learned*, in the form of two best practices that we identified, and are since then bringing into practice in any new software development: (i) the *5C meta model* to realise *separation of concerns* (the concerns being Communication, Computation, Coordination, Configuration, and Composition), and (ii) the *Composition Pattern* as an architectural meta model supporting the methodological coupling of components developed along the lines of the 5Cs.
These generic results are illustrated, grounded and motivated by what we learned from the huge efforts to refactor the *iTaSC* software, and are now behind all our other software development efforts, without any exception. In the concrete *iTaSC* case, the Composition Pattern is applied at three levels of (modelling) hierarchy: application, iTaSC, and task level, each of which consist itself of several components structured in conformance with the pattern.

Index Terms—Software pattern, architecture, composition, robot programming, task specification

Modern software engineering practices

1. An integrated development environment: e.g. centered around Gitlab/Github, CI/CD and docker.

2. An integrated IDE: e.g. Clion. Clion provides all of the state of the art features such as code completion, debugging but also integration of external tools such as e.g. gtest, valgrind, different build tools, doxygen, tool for code test coverage.

3. Deliver often: The steps implementing the development process must be fast to allow agile coding iterations.

4. Test constantly: The local development environment and the CI/CD must be equivalent to be able to reproduce CI failures.

5. Tools follow the purpose (and not the other way around): Integrations with the 3rd party tools, such as a requirements management tool, are tailored to the particular team to allow for the quick fixes and extensions.

6. Single source of truth: The main code repository should be as monolithic as possible and all of the development artifacts (design documents, code, tests, documentation, ...) should be as co-located as possible.

Certification process

ISO 26262, SEooC, part 3, part 6, part 8 processes →

Automotive Stakeholder Requirements (ASR)

Requirements	Architecture	Unit Design	V&V	Conf. Reviews
Elicitation, Safety Concept, SW Safety Requirements	UML (unified modeling language), FMEA	SCA (Static Code Analysis), SW practices outline, coverage, FMEA	Req., arch., unit, integration, system, performance, fault injection tests	Safety manual, Restrictions, Traceability

- builtin_interfaces_**cert**
- connext_micro_support_**cert**
- allocator_**cert**
- logging_**cert**
- rclcpp_**cert**
- threading_**cert**
- **Apex_ecu_monitor (native)**
- **Apex_utils (native)**

ROS

- builtin_interfaces
- connext_micro_support
- allocator
- logging
- rclcpp
- threading

Feature set reduction

Apply real-time and determinism constraints

1. Memory static
2. Remove blocking calls and recursions

Apex.OS Cert

Key steps

1. Making APIs memory static: real-time compliance: Rewrote all non-deterministic runtime memory allocations, blocking calls, and usage of standard STL packages (such as threading).

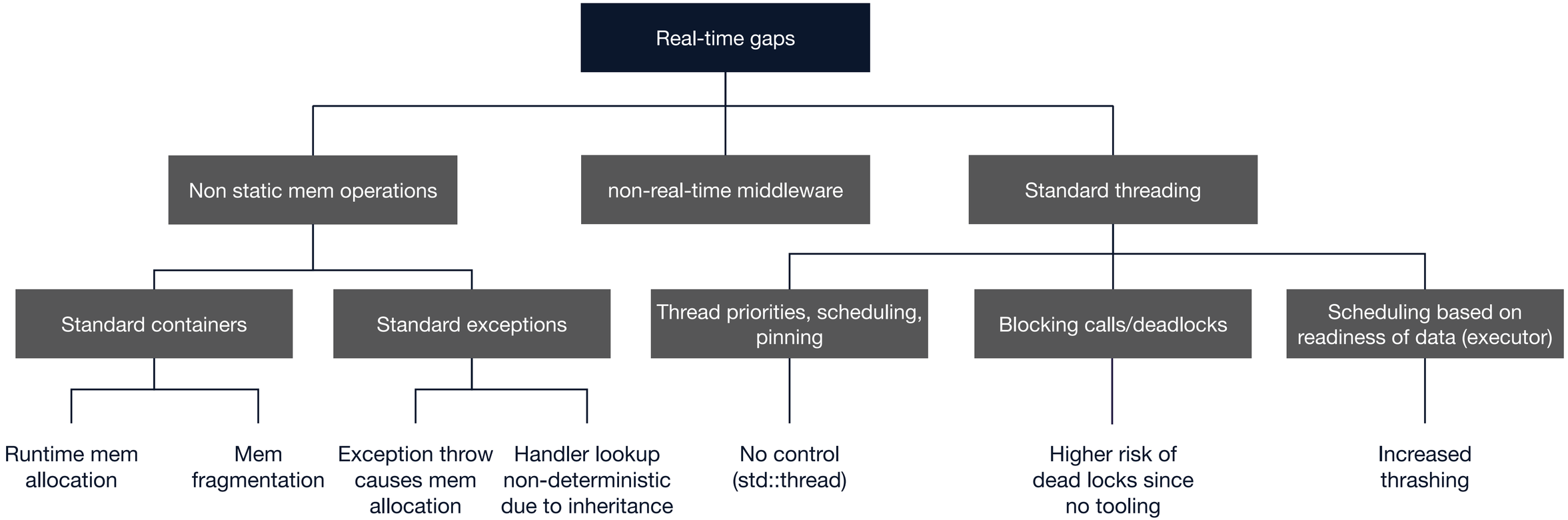
2. Structural Coverage: 100% statement, branch and MC/DC coverage for all Cert packages as mandated by ISO 26262- 6:2018 for ASIL D.

3. FMEA: Extensive safety analysis for every public API to derive additional safety requirements or R&R (restrictions and recommendations) for its users.

4. Requirements traceability:
    1. No formal requirements available from ROS 2 fork.
    2. Wrote several hundred safety and nominal requirements and traced them to codebase and tests using a certified requirement management tool.

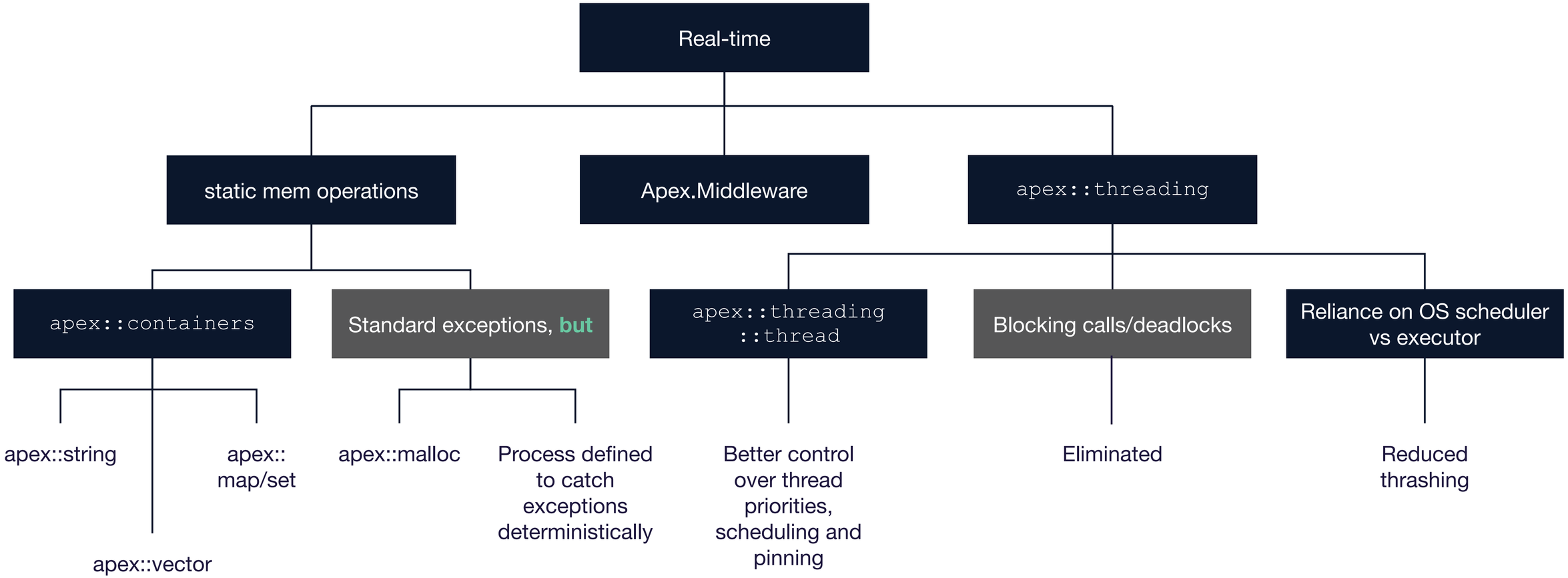
Real-time

ROS 2 exhibits the following gaps to enable **real-time performance:**



Real-time

Apex.OS addresses the following gaps **to achieve real-time performance:**



Apex.OS is retaining the rich ROS ecosystem

While providing real-time and automotive grade reliability and safety

ROS +


TÜV NORD

DAkkS

Certificate

No. SEBS-A.152852/17 V1.0

TÜV NORD Systems GmbH & Co. KG hereby certifies to

Apex.AI, Inc.
979 Commercial Street
Palo Alto, CA 94303
United States

that

Apex.OS Cert

meets the requirements listed in the below mentioned standards

- ISO 26262-2:2018, capable up to ASIL D
- ISO 26262-6:2018, capable up to ASIL D
- ISO 26262-8:2018, capable up to ASIL D
- ISO 26262-9:2018

Zertifizierungsprogramm Leitfaden (SEB-ZE-SEECERT-VA-320-20, Rev. 5.1/4.18)

The released versions of the certified Apex.OS are given in the "tracking-list". This list is an add-on to this certificate. The ASIL of the different versions and revisions are visible in the tracking list.

The certification is based on the reports in the valid version referenced in the "tracking-list". This certification entitles the holder to use the pictured "Safety Approved SEooC" mark.

Valid until: 2026-03-15
File reference: 8115523639

Hamburg, 2021-03-15

Blanca Pfuff

Certification Body SEECERT
TÜV NORD Systems GmbH & Co. KG
Große Bahnstraße 31, 22525 Hamburg, Germany

= Apex.OS

Largest software development framework for automotive, robotics, autonomous, smart machine applications.

Safety Certification (ISO 26262 ASIL-D)

First and only cross-application SDK certified to the highest level of automotive safety.
Certified Apex.OS was honored with CES Innovation Award 2021.

Summary: Enabling software-defined vehicles

1. A standardized **software architecture with open APIs** to enable mutually compatible solutions ideally across many manufacturers, suppliers, and academia.
2. An **awesome developer experience** to enable developer productivity – based on the understanding that the quality of the developer experience is directly related to their productivity.
3. A **software architecture that scales** to massive software systems.
4. A **software implementation based on modern software engineering practices**.
5. **Abstraction of the complexity** of all underlying hardware and software.
6. **Deterministic, real-time execution, automotive functional safety certification**.

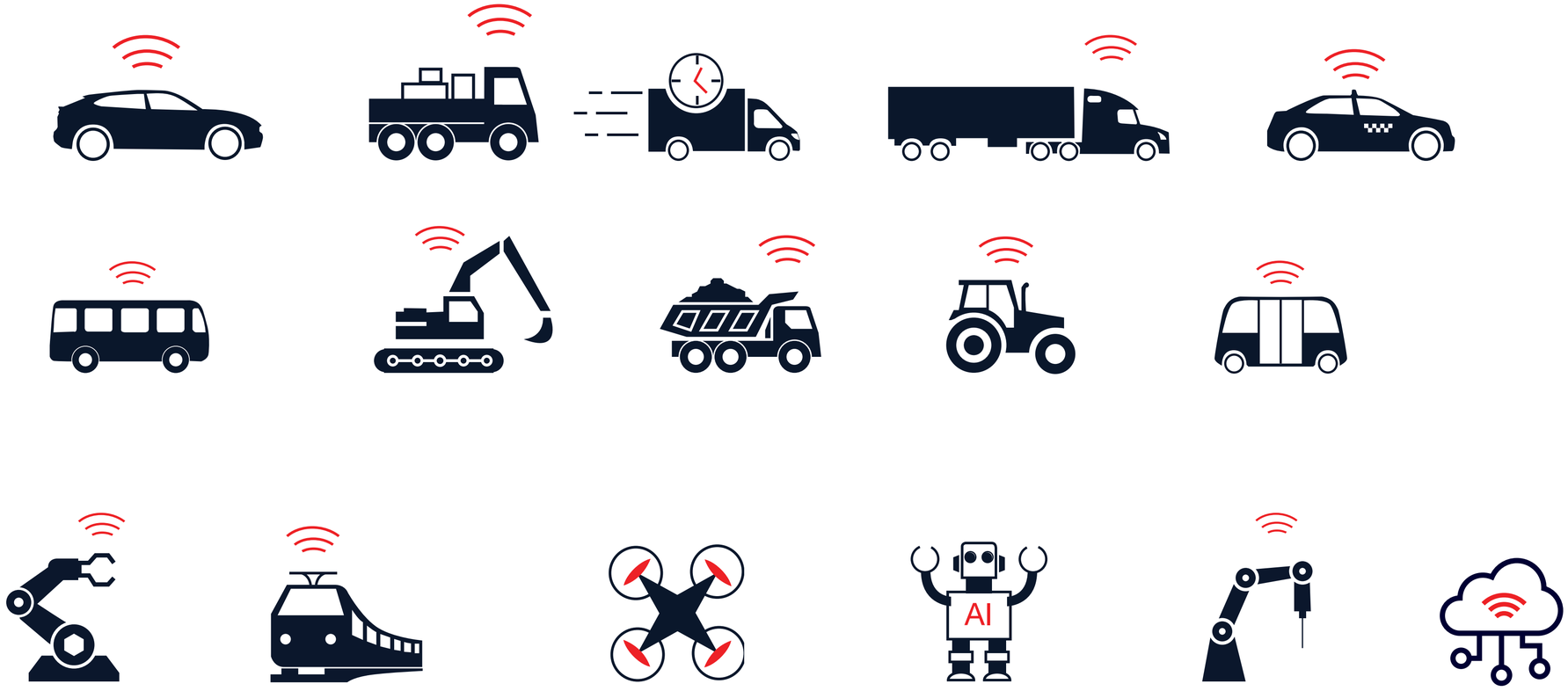





Outlook

	2020	2021	2022
	Components: • Eclipse Cyclone DDS • Eclipse iceoryx • SOME/IP	Integrated in **Apex**.Middleware: • Integrated with **Apex**.OS Cert • Interoperable with AUTOSAR Adaptive (ara::com and SOME/IP) • DDS Security • Automotive grade and supported	**Apex**.Middleware Cert: • Includes developer tools • Professionally supported • ISO 26262 certification
Scope	Proof-of-Concept (PoC)	Pre-production	Production

Outlook



Apex.AI®

The vehicle OS company.

Apex.OS - A safety-certified software framework based on ROS 2