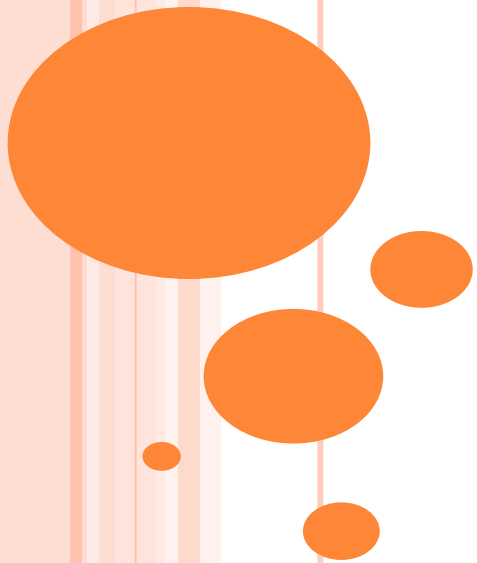


SOFTWARE ARCHITECTURE FOR AUTONOMOUS DRIVING

Yu Huang

yu.huang07@gmail.com

Sunnyvale, California



OUTLINE

- Software Architectures
- Agile Architecture
- CHARMY
- AUTOSAR
- Adaptive AUTOSAR
- ASAM
- ISO 26262
- ASIL and SIL
- Software Development Process
- Iterative Software Development
- Testing for Software Development
- Testing for Autonomous Driving
- Software Engineering for Autonomous Driving
- V-Model for Autonomous Vehicle Software
- Software Development Process in Vehicles
- A Cloud-assisted Design for Autonomous Driving
- Implementing a Cloud Platform for Autonomous Driving
- Autonomous Driving Algorithm Development on Amazon AI
- Data Driven Development of Autonomous Driving at BMW

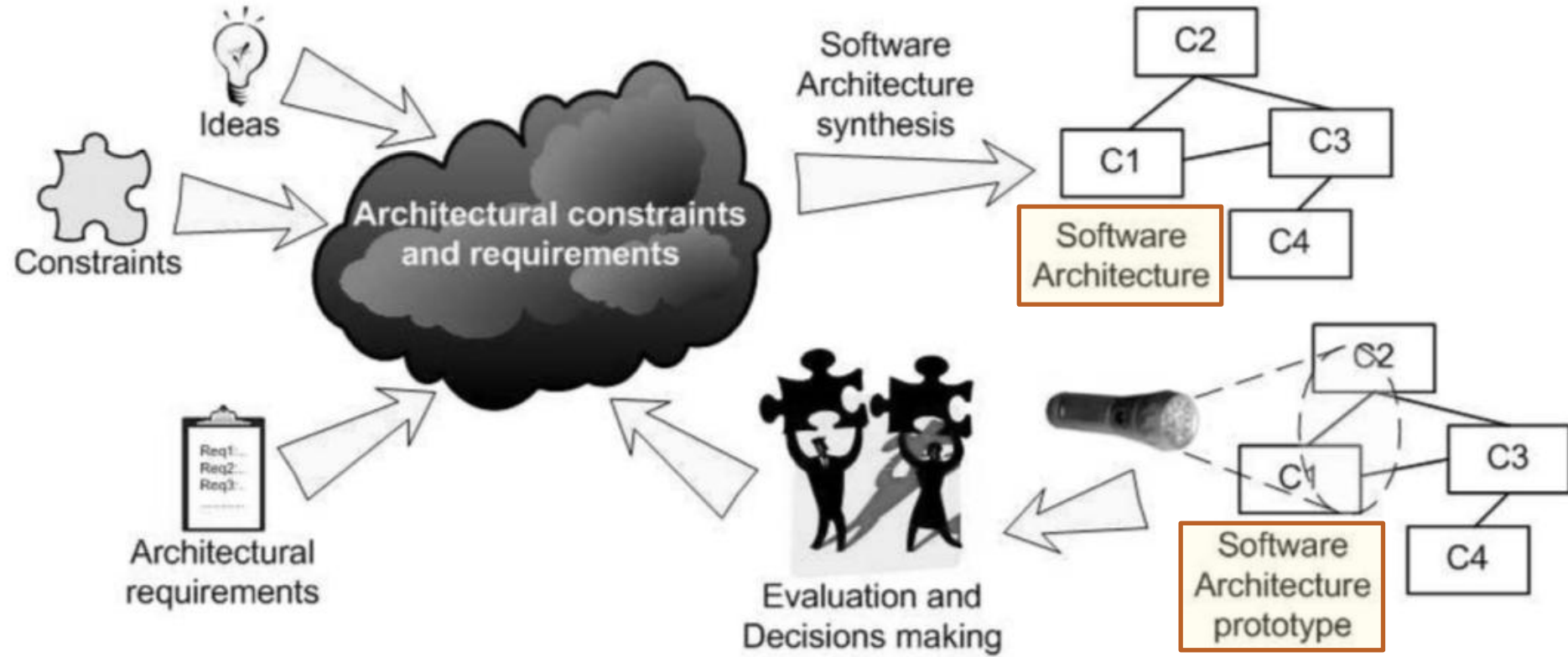


SOFTWARE ARCHITECTURES

- SOFTWARE Architectures (SAs) emerged in the '90 to structure complex SW systems while providing a high-level system description.
- SA is an autonomous discipline that focuses on the overall organization of a SW system by using abstractions to express the logical structure of distributed systems.
- SA specification emphasizes the system structure (i.e., the architecture topology) in terms of components and connectors and the architecture dynamics by identifying how components and connectors interact.
- When SW architects start defining SA of a system, ideas, needs, requirements, and challenges from users/customers constitute the dark set of architectural constraints.
- Decisions at the SA level have to be made and they encompass the organization of a SW system, the selection of the structural elements, their interfaces, their behavior, and the composition of these elements into progressively larger subsystems.
- SAs are typically used as a high-level design blueprint of the system.
- Assuring as early as possible, the correctness of an SA is a fundamental task in order to produce quality software.



SAs



AGILE ARCHITECTURE

- A **system or software architecture** that is versatile, easy to evolve, to modify, flexible in a way, while still resilient to changes
- An **agile way** to define an architecture, using an iterative lifecycle, allowing the architectural design to tactically evolve gradually, as the problem and the constraints are better understood
- The two are not the same
 - You can have a **non-agile development process** leading to a **flexible, adaptable architecture**, and *vice versa*, an **agile process** may lead to a rather **rigid and inflexible architecture**
 - One does not imply the other
- In the best of worlds, to have an **agile process**, leading to a **flexible architecture**



CHARMY

- CHARMY framework provides an automated, easy-to-use tool for the model-based formal design, and validation of SAs.
- CHARMY offers an integrated environment supporting the combination of simulation and verification.
- This combination is useful for gaining confidence on the system and understanding the verification results.
- The model checker used by CHARMY is SPIN.
- The SA, modeled in terms of CHARMY diagrams, is automatically translated in Promela that is the specification language of SPIN.
- Model checkers, as well as other finite-state verification techniques, allow for the automated checking of the system model compliance to given temporal properties.
- CHARMY temporal properties are specified as Properties Sequence Charts (PSCs).

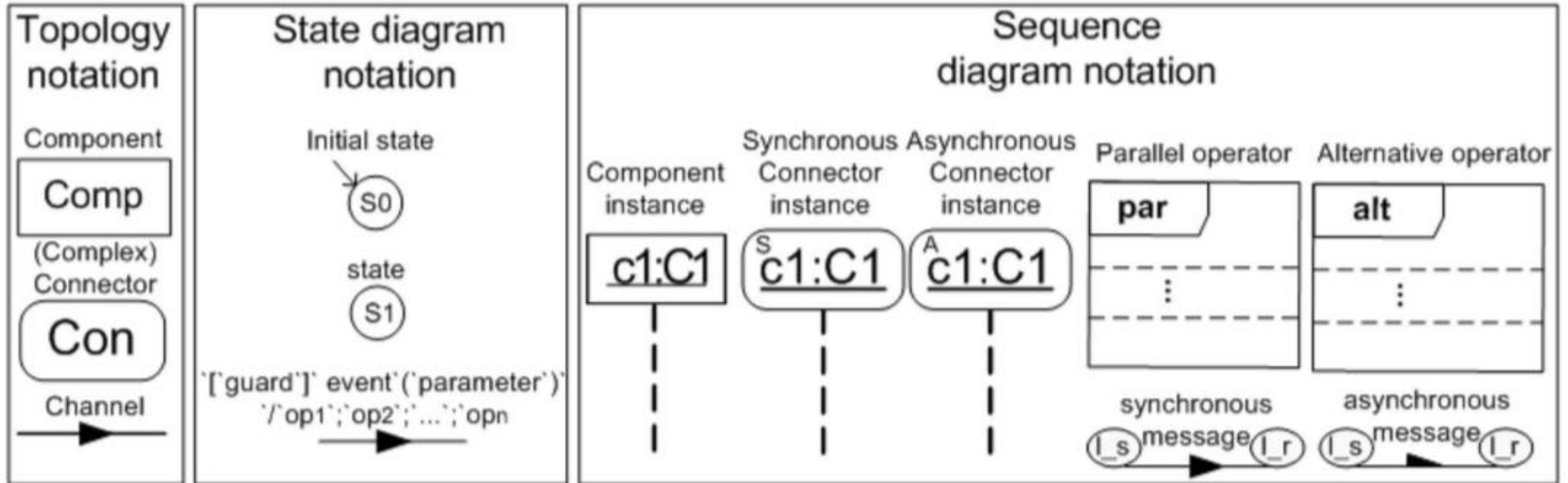


CHARMY

- The verified and final SA is the starting point of model transformations that produce a skeleton of the (Arch) Java code of the system.
- The produced code reflects structural SA constraints; more precisely, by means of the Arch Java API, the generated Java code explicitly contains architectural constructs (such as components, connectors, channels, etc.);
- The code will ensure communication integrity, e.g., at the implementation level;
- Each component can only communicate accordingly to what declared in the SA.
- The CHARMY approach for specifying and analyzing SAs hides most of the complexity of the modeling and analysis process and it is tool supported.
- The tool has a plugin-based architecture, which allows an easy and rapid integration of new modeling and analysis techniques.
- A notation close to the UML component diagram is used in CHARMY.



CHARMY



CHARMY

- The behavior of each component is specified in terms of CHARMY state diagrams, while CHARMY sequence diagrams are used to describe communication and synchronization among components.
- A notation general enough to encompass those used in current SW development practice and rich enough to allow for analysis, is used.
- State diagrams are described using the State Transition Diagram notation;
- The elements written into the guard are variables local to the state diagram.
- An event, a message sent or received, can have a parameter that is the value that will be exchanged between a pair of components when the transition fires.
- Components interact according to the semantics of their communication as expressed in a sequence diagram.
- Consistency checks implemented in CHARMY forbid the definition of two different kinds of communication for a specific channel that connects two components.

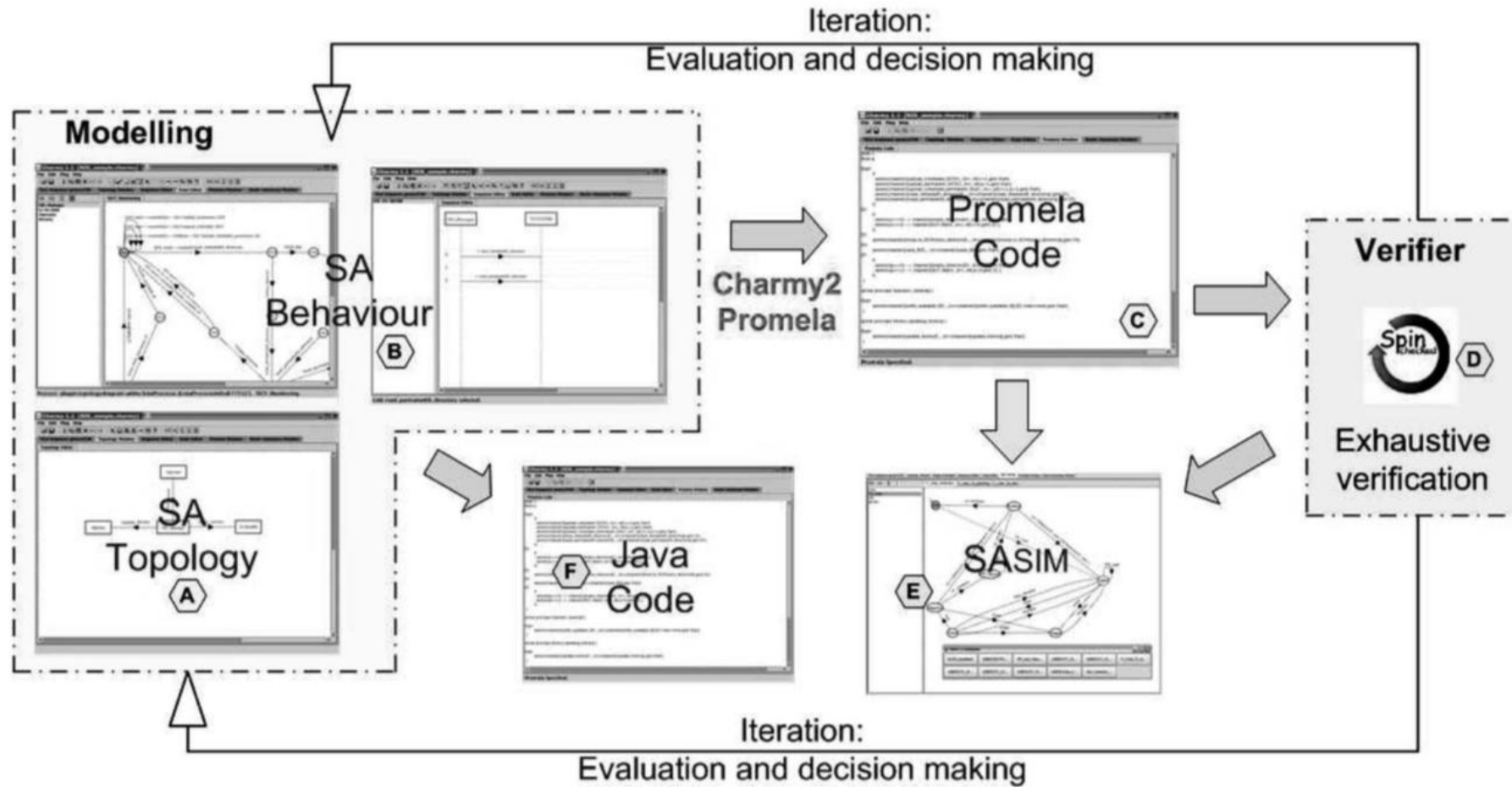


CHARMY


- SAs are modeled directly in the CHARMY tool by means of the CHARMY editors.
- Once designed a CHARMY specification, the Charmy2Promela translation is used to obtain an executable prototype in Promela from the model-based SA specification.
- Promela allows the specification of concurrent systems communicating via either messages or shared variables.
- On the generated Promela code, can use SPIN to find deadlocks, unreachable states or we can verify specific temporal properties.
- SASIM, the CHARMY Simulation feature, makes use of the SPIN simulation and interprets its results in terms of CHARMY state and sequence diagrams.
- SASIM is automatically started in case the verification finds an error or it is launched by the SW architect to study how the system reacts in particular situations.
- Whenever a final SA is reached, automatic Java code generation is performed.



CHARMY



AUTOSAR

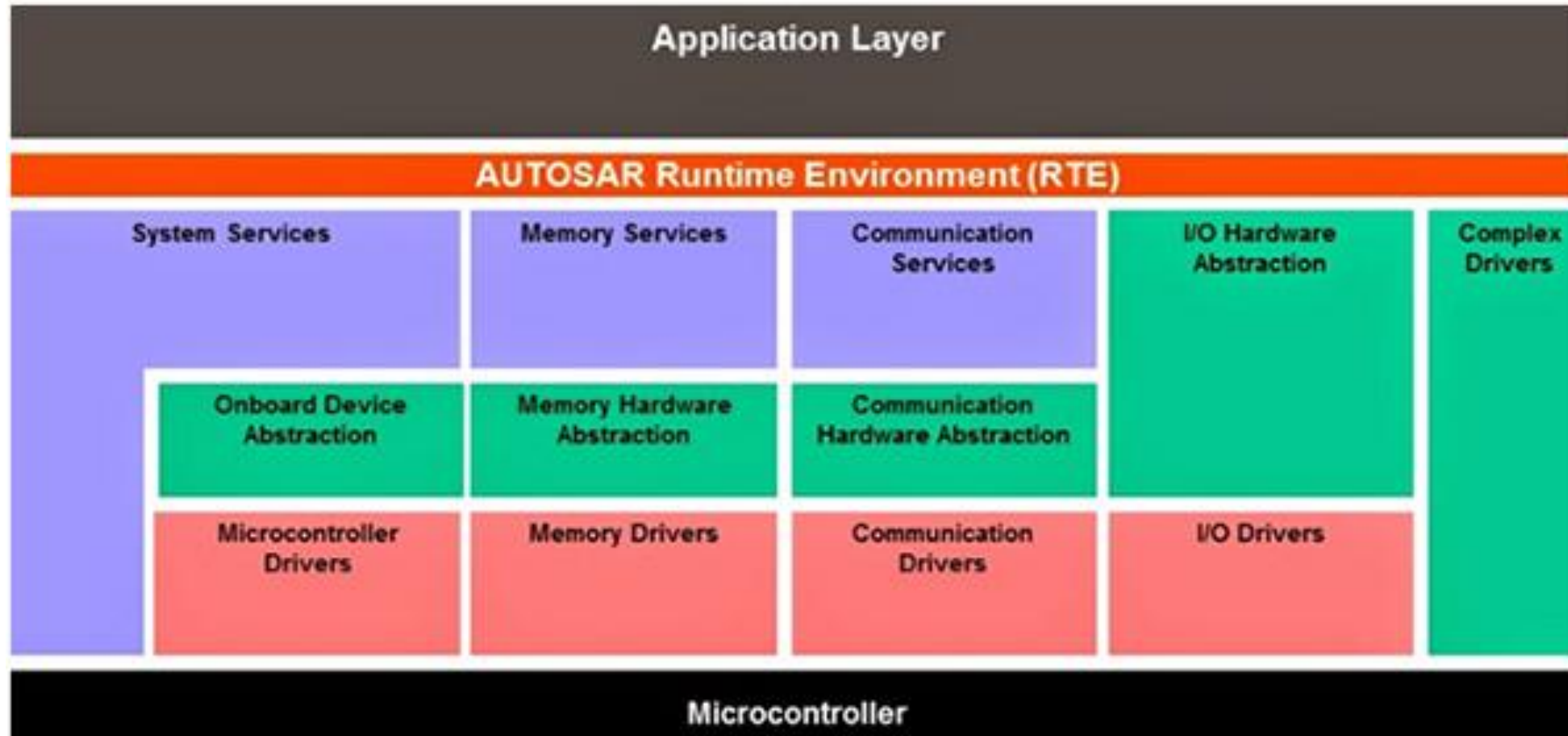
- AUTOSAR (AUTomotive Open System ARchitecture) pursues the objective of creating and establishing an open and standardized SW architecture for automotive ECUs.
 - AUTOSAR uses a three-layered architecture:
 - Basic Software (BSW): standardized software modules (mostly) without any functional job itself that offers services necessary to run the functional part of the upper SW layer.
 - Runtime environment(RTE): Middleware which abstracts from the network topology for the inter-/intra-ECU info. exchange btw the application SW components and between the BSW and the applications.
 - Application Layer: application software components that interact with the runtime environment.
 - System Configuration Description includes all system info. and the info. agreed btw different ECUs.
 - ECU extract: contains the info. from the System Configuration Description needed for a specific ECU (e.g. those signals where a specific ECU has access to).
 - ECU Configuration Description: contains all basic software configuration info. that is local to a specific ECU, then use this info. to build the executable software, the code of the basic software modules and the code of the software components out of it.
- 

AUTOSAR

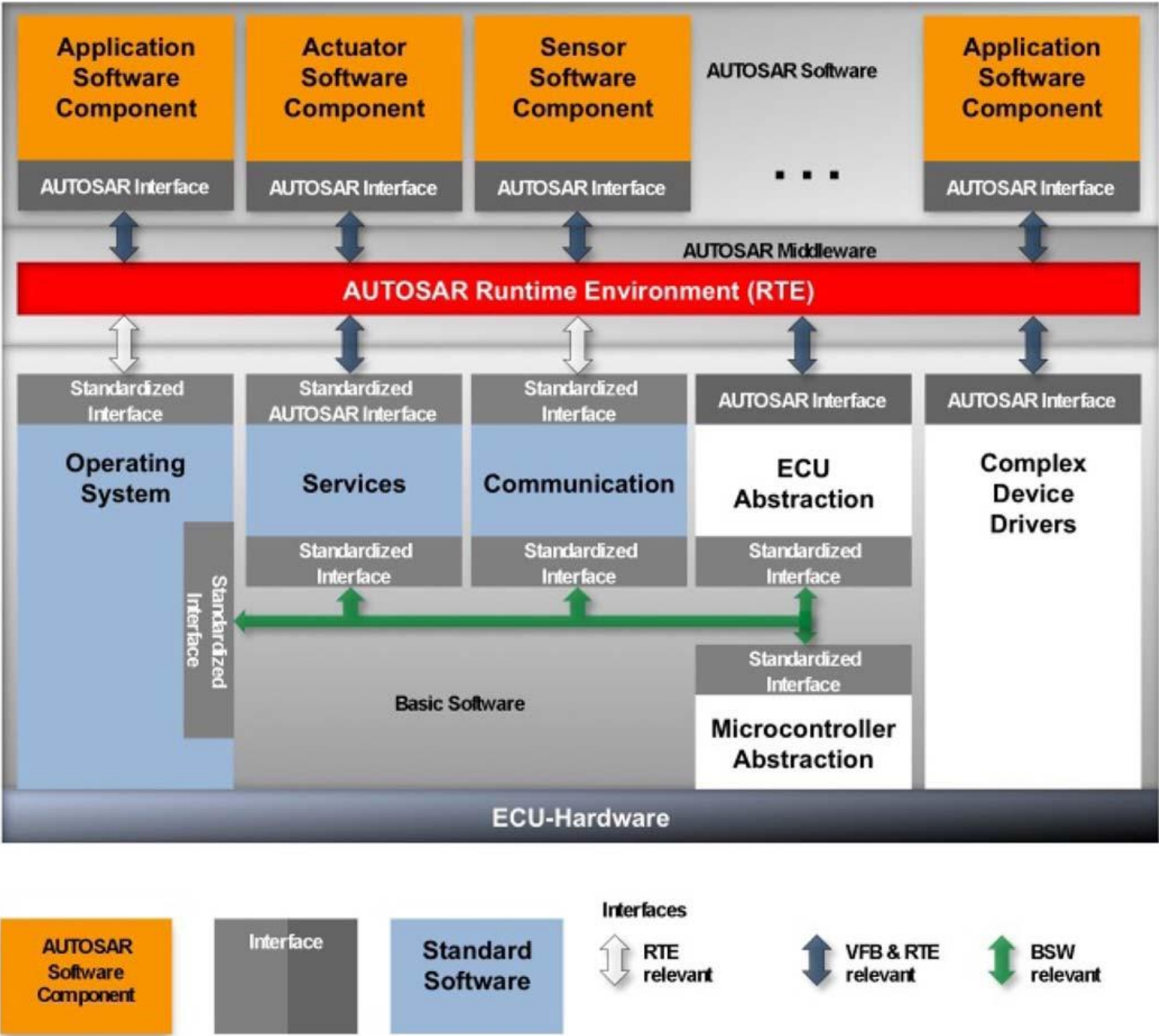
- The AUTOSAR Classic Platform architecture distinguishes on the highest abstraction level between three software layers that run on a microcontroller.
- The AUTOSAR Classic Platform is the standard for embedded real-time ECUs based on OSEK.
- Communication between software components and access to BSW happens via RTE, which represents the full interface for applications.
- BSW's three major layers and complex drivers: services, ECU abstraction, microcontroller abstraction;
- Services are divided into functional groups for system, memory and communication services.
- Virtual functional bus (VFB) is an abstract set of RTEs that are not yet deployed to specific ECUs and decouples the applications from the infrastructure.
- VFB communicates via dedicated ports, which means that the communication interfaces of the application software must be mapped to these ports.
- The VFB handles communication within the individual ECU and between ECUs.




AUTOSAR



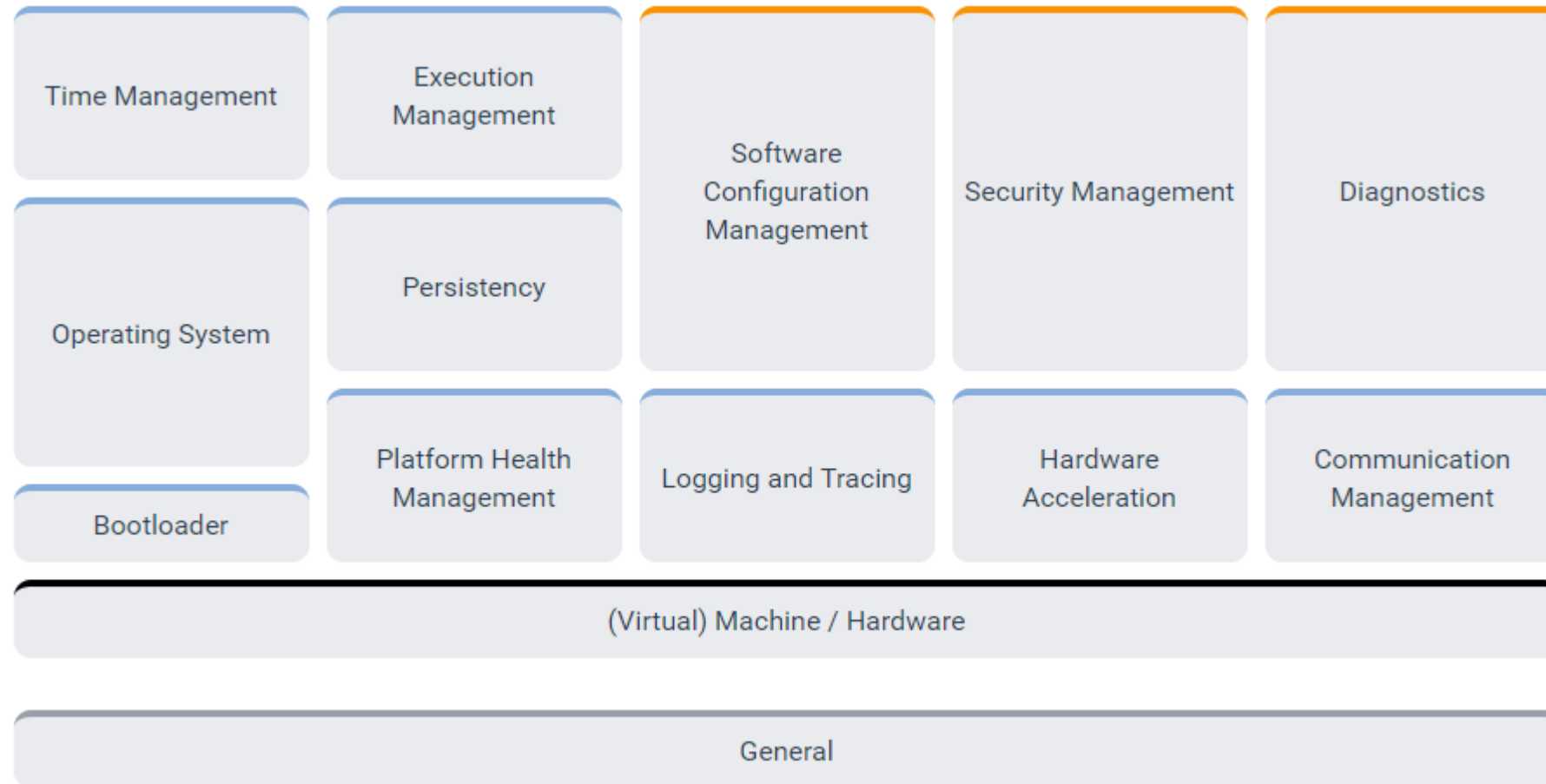
AUTOSAR



ADAPTIVE AUTOSAR

- MICROSAR is the embedded software for AUTOSAR ECUs.
 - It consists of the MICROSAR RTE and MICROSAR BSW modules.
 - The uniform and intuitive user interface of DaVinci Configurator Pro is used to configure the MICROSAR modules, automatic derivation of parameters and extensive validation functions.
 - **Adaptive AUTOSAR** is standardization of the AUTOSAR runtime for Adaptive Applications (ARA).
 - The standard contains two types of interfaces: services and APIs.
 - These interfaces allow OEMs to implement **autonomous driving, over-the-air** (OTA) software updates, IoT (Internet of Things) features, media streaming and other services.
 - Compared to AUTOSAR Classic, the Adaptive platform allows dynamic linking of services and clients during ECU runtime which makes it much more flexible for the application developers.
 - Besides **Elektrobit**, there are few companies that are publicly developing products that implement the Adaptive AUTOSAR.
 - This requires for instant communication with **traffic infrastructure** (e.g. traffic signs and -lights), **cloud servers** (e.g. to access the latest traffic information or map data) or the use of microprocessors and HPC hardware for parallel processing (e.g. GPUs).
- 

ADAPTIVE AUTOSAR



ASAM

- ASAM (Association for Standardization of Automation and Measuring Systems) promotes standardization for tool chains in automotive development and testing, under German law.
- The standards define protocols, data models, file formats and APIs in the development and testing of automotive ECUs.
- ASAM cover specific use-cases and are developed according to the following guiding principles:
 - Independence from Hardware and Operating System
 - Use of object-oriented models
 - Definition of semantics and syntax
 - Independence from the physical storage of data
- ASAM's three groups: AE (Automotive Electronics), CAT (Computer Aided Testing), COMMON (Common standards shared between AE and CAT);
- AE is applied during design/implementation of ECU SW development (left side of the **V-Model**);
- CAT is applied during the verification & validation phases of ECU SW (right side of the **V-Model**), and during automated calibration and system testing on engine and vehicle test beds.

ISO 26262

- **ISO 26262, "Road vehicles for Functional safety"**, int. standard for functional safety of electrical and/or electronic systems in production automobiles by ISO in 2011.
- Goals:
 - Provides an automotive safety lifecycle and supports tailoring the necessary activities.
 - Covers functional safety aspects of the entire development process.
 - Provides an automotive-specific risk-based approach for determining risk classes (ASILs).
 - Uses ASILs for specifying the item's necessary safety requirements.
 - Provides requirements for validation and confirmation measures.
- Parts (10):
 - Vocabulary, Management of functional safety, Concept phase (1-3);
 - Product development at the system level, hardware level and software level (4-6);
 - Production and operation, Supporting processes (7-8);
 - **Automotive Safety Integrity Level** (ASIL)-oriented and safety-oriented analysis (9);
 - Guideline (10).



ASIL AND SIL

- **Automotive Safety Integrity Level (ASIL)** is a risk classification scheme defined by the ISO 26262 - **Functional Safety for Road Vehicles** standard.
- There are 4 ASILs by the standard: ASIL A-B-C-D, from the lowest to the highest.
- The determination of ASIL is the result of hazard analysis and risk assessment.
- ASIL D represents potential for severely life-threatening or fatal injury in the event of a malfunction and requires the highest level of assurance that the dependent safety goals are sufficient and have been achieved.
- Referring to "Quality Management", the level QM means that risk associated with a hazardous event is not unreasonable and does not therefore require safety measures in accordance with ISO 26262.
- **IEC 61508** defines a widely referenced **Safety Integrity Level (SIL)** classification.
- ASIL is a qualitative measurement of risk, SIL is quantitatively defined as probability or frequency of dangerous failures depending on the type of safety function.
- ASIL D has been shown aligned to SIL 3 and ASIL A is comparable to SIL 1.



SOFTWARE DEVELOPMENT PROCESS

- Changing requirements and enhancements of technologies need to be incorporated into the development effectively.
 - Among development processes, **Extreme Programming (XP)**, **Scrum**, and the **Crystal** family of processes are the most prominent.
 - However, these development processes are dedicated to **software only** and seem to not support embedded systems.
- Demanded by the test-first development approach, tests for functions and algorithms are coded in a parallel manner to the actual code.
- The fulfillment of requirements needs to be tested, by running the SW in a virtual test-bed for avoiding race conditions on valuable resources like the real vehicle.
- The simulation allows to test– and to automatically re-test– SW behavior in a regression testing manner without any possible harm.



ITERATIVE SOFTWARE DEVELOPMENT

- SW development processes can be iterative and incremental to allow an evolutionary evolution towards a stable, robust, and mature result.
 - On the one hand, iterations are necessary to continuously evaluate the current state of the customer's needs.
 - On the other hand, iterations enforce to continuously integrate artifacts to a functional system.
- **Iterations** are a vital part to deal with the necessary improvement of existing SW, and an incremental process that decomposes SW into smaller pieces.
- In integrated SW and systems development projects with a heavy SW part an iterative SW development process needs to be used instead of the traditional engineering process.
- The best way is to **decouple** sub-projects in which individual processes can be followed.
- To **decouple** iterative SW development from HW e.g. through **virtualization**.
- With modern tools for version control and configuration management, continuous **integration** can be achieved rather easily.
- The most tricky part is to ensure that everyone are committed to an integration process, while everybody releases and **integrates** his/her SW at least several times a day.



TESTING FOR SOFTWARE DEVELOPMENT

- Version control is the basis for process, automated, integrated, unattended testing.
- **Testing** is the most important technique for QA and comes in many different flavors, beginning with **unit tests** to integration tests up to full system tests.
- The “testing trap”, as unnoticed SW bugs due to inconsequential SW testing, can only be escaped through automated replaying of tests, as **regression testing**.
- In unit tests, automation helps to know whether an error was introduced to the code, in which iteration it was introduced, and in which SW part.
- Each version, committed to the versioning server, triggered automatically a testing process, where all tests are run and feedback in the form of # failed tests is given.
- Testing also needs to deal with configuration variations, a popular C/C++ problem, allows code to behave differently on different platforms.



TESTING FOR AUTONOMOUS DRIVING

- For autonomous driving, automated testing has a number of challenges to tackle.
 1. SW is integrated and closely linked with HW, such as sensors and actuators.
 2. Abstractions for different parts of SW and HW are necessary to run tests efficiently.
 3. For the “intelligent part” of the SW, it is **not** necessary to run tests based on full sensory input, **but** to provide **distilled, aggregated** info. about obstacles and the pathway.
 4. An important abstraction for efficient test automation is to replace HW by **simulation**.
- One successful organizational and structuring tool were **story cards**.
- A **story card** describes the **goals** for a development iteration and thus leads to an efficient, and focused structuring of the requirements and the development project.
- The SW framework provides rudimentary system services like **concurrency** and **communication**.



SOFTWARE ENGINEERING FOR AUTONOMOUS DRIVING

- A modern IDE like **Eclipse** is used for direct code development in C++ and MATLAB/Simulink for the **control** algorithms.
- As MATLAB/Simulink does not scale for complex data structures or supports properly the design and implementation of SAs, most of the code is written in C++.
- For **planning** purposes, math functions are used to understand the algorithms, and UML class and deployment diagrams as well as state charts are used for the SA as well as the important state-based **behaviors** and data structures.
- Using **Trac** as an integrated and easy-to-use web based portal for the complete SW and system development process enables to track changes to the SW over time, and evaluate the actual state of the software generated by the back-end tool chain.
- Every **story card** is **virtually** available for every team member at any time to see the most important aspects for the current and next development iteration.



SOFTWARE ENGINEERING FOR AUTONOMOUS DRIVING

- A list of tasks and open bugs are available for every **virtual story card**.
- In an initial step, the requirements of the **story card** are translated into an initial number of **tests** which describe a scenario even before the SW is developed.
- To manage development and configurations in simulation and real HW, a release and configuration management tool based on **FSVS (Fast System Versioning)** is used.
- Configuration management ensures that HW changes fit to the loaded SW releases.
- Using FSVS as a version control system for filesystems enables the team to simply and safely test new SW versions and to maintain the **integration** between **parallel** developments as well as tracking of **open issues** and **potential bugs** which were found during real vehicle tests.
- Instead of FSVS, filesystems like **ZFS** or **Btrfs** can be used for a similar purpose.



SOFTWARE ENGINEERING FOR AUTONOMOUS DRIVING

- Combination of **virtual story cards** and a consistent **release and configuration management** enables the team to safely develop and test potentially dangerous new SW functions without breaking an running SW system on the vehicle.
- **Elements of Extreme Programming**, like the test-first approach, common code ownership, pair programming, and continuous integration are the **basis**.
- Subversion manages everything to build the project to ensure self-containedness.
- Use of Subversion and the SW build tools allow to setup the fully automated build process, including **test runs**, acting as a monitor to the repository needed for QA.
- Triggered by every commit against the repository, central **servers** start to check out the project **sources** and initiate an entire build.
- On the **developer** side these tools help to speed up the build process and ensure a consistently built result by analyzing changes and dependencies.
- On the **server** side it allows to build alternative targets for different forms of use, to run a system build with or without test code.

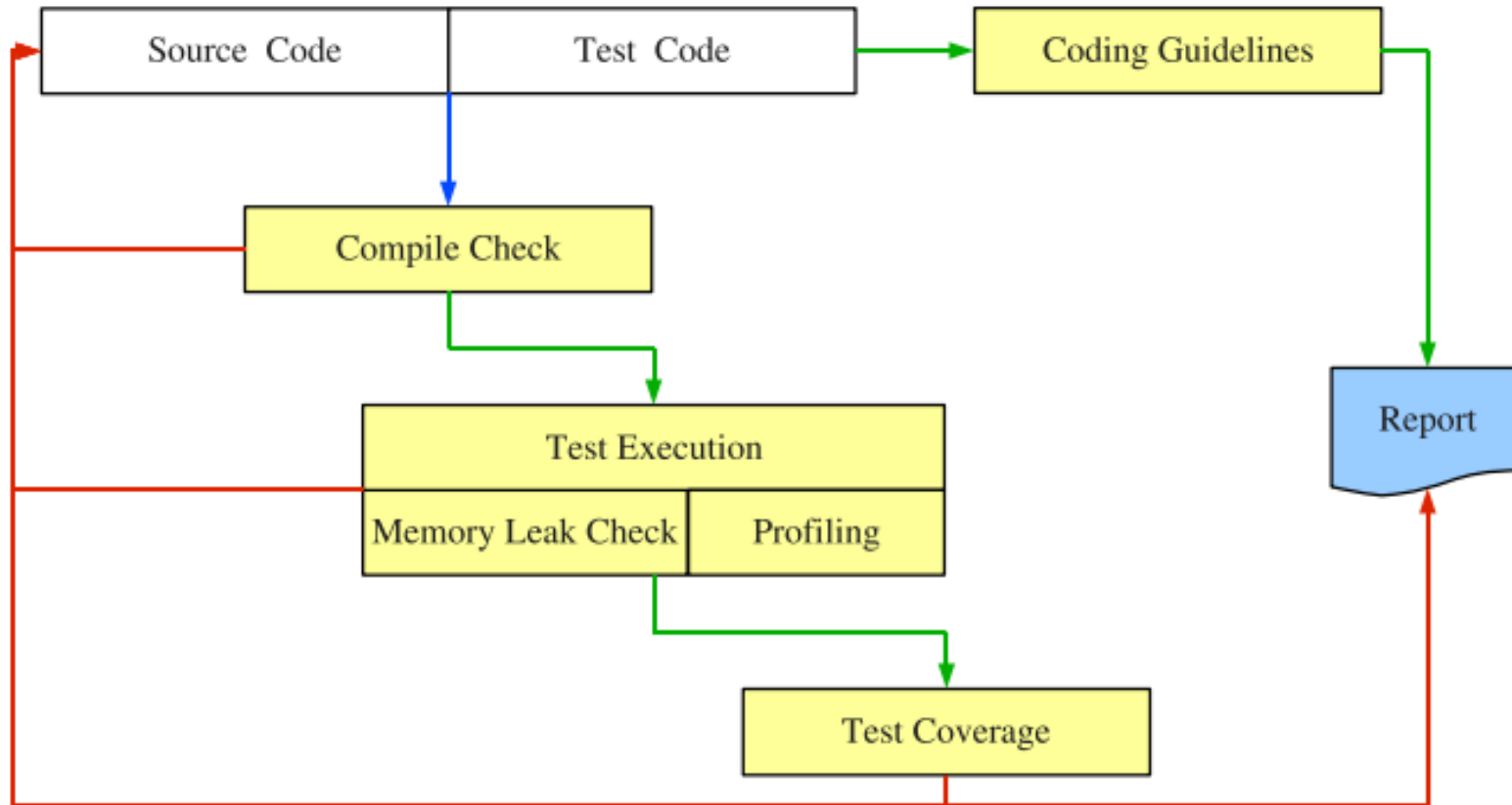


SOFTWARE ENGINEERING FOR AUTONOMOUS DRIVING

- The automated process for checking the software quality consists of several steps:
 - The 1st stage contains the **compilation** check, followed by a memory leak check which is executed during the test run to find potential memory leaks or NULL pointer access.
 - After executing the **test cases**, their coverage related to the source code is computed.
 - Different coverage criteria like statement or path/branch coverage is defined and checked.
 - They indicate where the developer should add or modify test cases to fulfill the coverage level.
 - After fixing bugs, memory leaks or more test cases, it is optimized using profiling tools.
 - The code can be checked using a given coding guidelines definition.
 - All results are aggregated into one single report.
 - It can step-wisely executed manually by the developer or completely unattendedly and automatically using the continuous integration system.



SOFTWARE ENGINEERING FOR AUTONOMOUS DRIVING



Multi-level test process

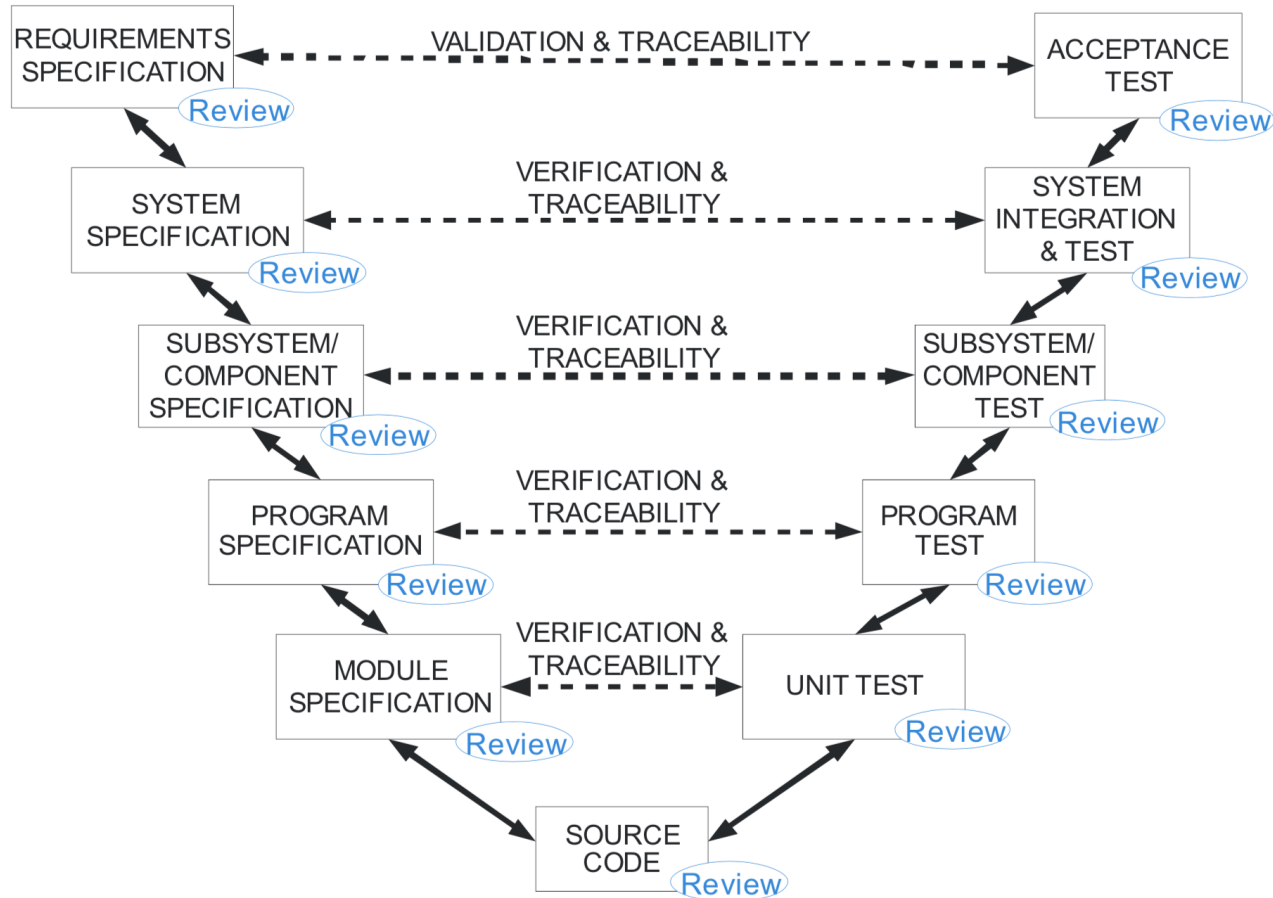


V-MODEL FOR AUTONOMOUS VEHICLES' SOFTWARE

- Five major challenge areas in testing according to **the V model** for autonomous vehicles:
 - driver out of the loop
 - complex requirements
 - non-deterministic algorithms
 - inductive learning algorithms
 - and fail- operational systems.
- General solution approaches that seem promising across these different challenge areas:
 - phased deployment using successively relaxed operational scenarios
 - use of a monitor/actuator pair architecture to separate the most complex autonomy functions from simpler safety functions
 - fault injection as a way to perform more efficient edge case testing.
- It seems within reach to instead architect the system and its accompanying design process to be able to employ existing software safety approaches.



V-MODEL FOR AUTONOMOUS DRIVING SOFTWARE



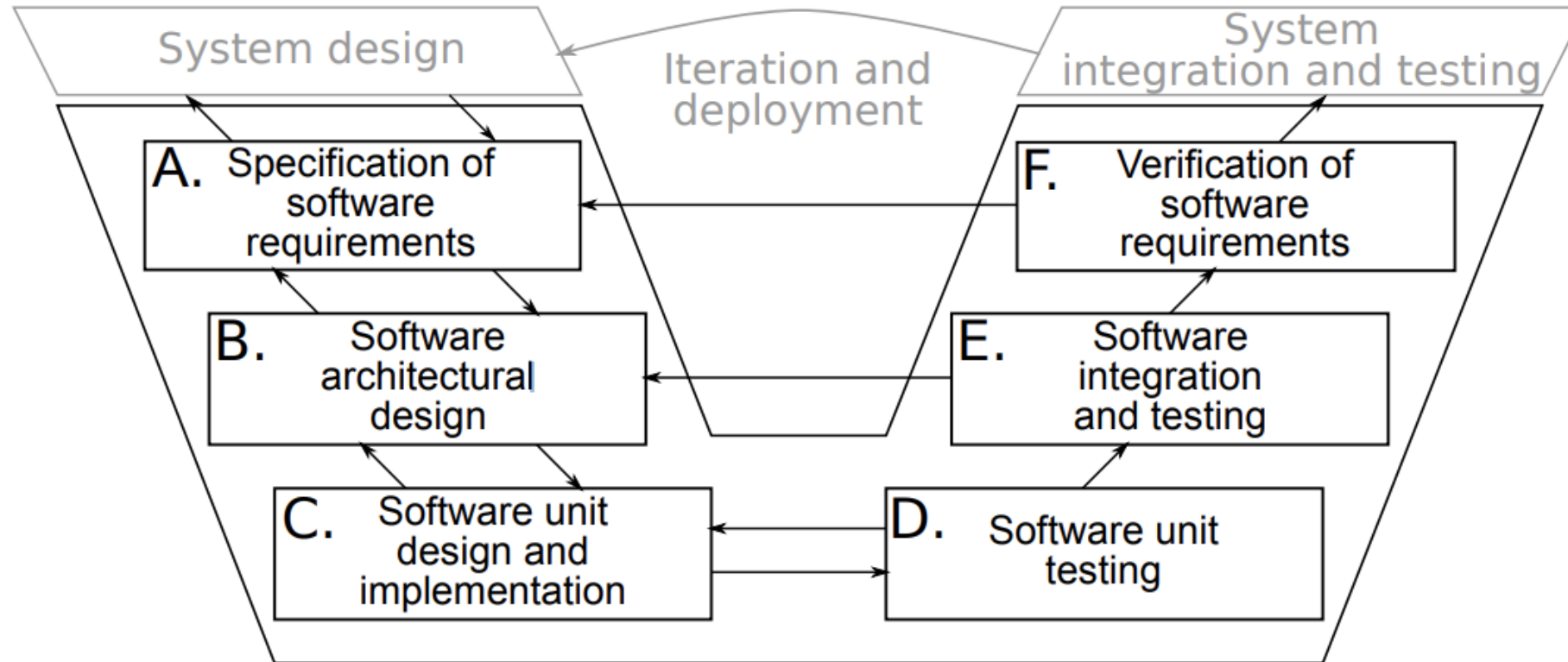
- The “V” SW development model has been applicable to vehicles for a long time.
- It was one of the development reference models incorporated into the MISRA Guidelines (> 20 years).
- It has been promoted to be the reference model that forms the basis of ISO 26262.
- The V model represents a methodical process of creation followed by verification and validation.
- The left side of the V works its way from requirements through design to implementation.
- At each step it is typical for the system to be broken into subsystems that are treated in parallel (e.g., there is one set of system requirements, but separate designs for each subsystem).
- The right side of the V iteratively verifies and validates larger and larger chunks of the system as it climbs back up from small components to a system-level assessment.

SOFTWARE DEVELOPMENT PROCESS IN VEHICLE

- Small-scale development is then based on the **Scrum** methodology.
- In every iteration, a new or improved driving function (e.g. autonomous lane change) is introduced to the vehicle.
- Beginning with the capturing of requirements, an impact analysis on the existing architecture is carried out and software units are designed.
- After their implementation, unit and system tests are run, and the new function is tested on a closed test track with other vehicles and obstacles according to a well-defined procedure.
- With fulfillment of all steps in the process, the new function is deployed into operation in the urban environment.
- The strong focus on the safety of road users is the main aspect of autonomous driving in a real urban environment.
- For safety-critical components the development process includes intense unit and system tests, and it meets ISO/DIS 26262 recommendations.



SOFTWARE DEVELOPMENT PROCESS IN VEHICLE



Software Development Process, adapted from ISO 26262, part 6.



SOFTWARE DEVELOPMENT PROCESS IN VEHICLE

- Each involved developer creates one or more **user stories** which describe functions necessary for the milestone from the **developers** domain.
- Every user story is described detailed in tickets which contain a short work product and a showcase, the developer has to present to the team after finishing a ticket.
- With the usage of **Trac and Agilo** many errors can be prevented in this early stage and the whole process is well documented.
- In the 1st step, interfaces and internal functions are designed and specified.
- After the design process, the components are implemented within the development platform **ADTF** (Automotive Data and Time-triggered Framework) from **Electrobit**.
- To shorten the development cycle and to allow early testing and benchmarking for different functions, these filters are implemented as prototypes and tested.
- To improve the quality standard methods like **code analyzing tools** are applied.



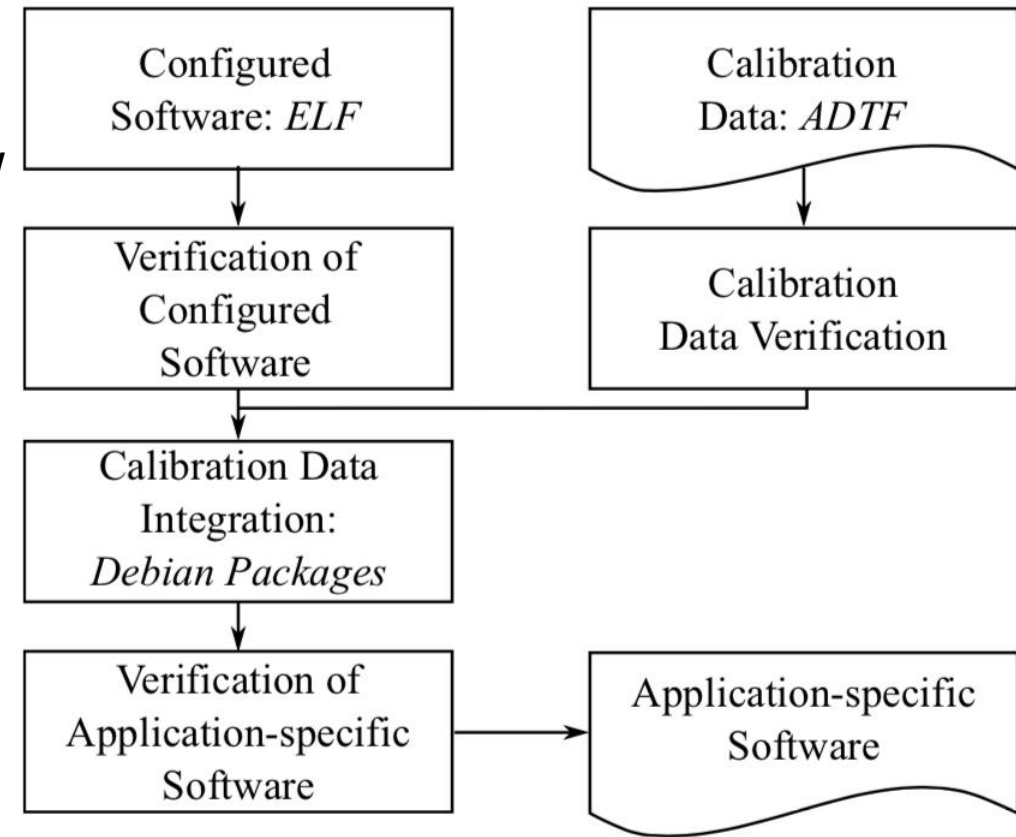
SOFTWARE DEVELOPMENT PROCESS IN VEHICLE

- With **Doxygen** the code gets documented simultaneously.
- With the use of SoA methods for C++ unit testing and the testing capabilities integrated in **ADTF** and continuous integration tools like **build-bot**, the test cases and the unit testing process is highly integrated in the development process.
- In simulation the system behavior is visualized and tested in desired traffic situations.
- If simulation succeeds, SW is applied and basic functionality like activating/deactivating, following a course and safety driver interaction are tested on an **empty** test track.
- After fulfilling all basic **test cases** the creation of SW packages is used to save SW versions that can be used to **test the functional behavior** in real urban traffic.
- Through packaging SW versions, it is easily possible to switch between different SW releases for the same HW configuration.
- Those releases have different driving abilities and can fit into different environments.



SOFTWARE DEVELOPMENT PROCESS IN VEHICLE

- The packaging process is based on Debian, with extensive and rich dependency info.
- Calibration data from ADTF and the configured SW in an ELF is verified and integrated into installable Debian packages for each vehicle PC.
- After installation this application-specific SW is verified again and then ready to use.
- It allows a very fast and flexible exchange for different SW releases, maintaining the consistency based on dependency models.
- To prevent a extensive test process, the desired scenarios are selected accord. to SW and component changes before testing starts.
- The last step in the development process are rides in real urban traffic with installed and packaged software versions.



SW configuration and packaging process, adopted from ISO 26262, part 6, appendix C.

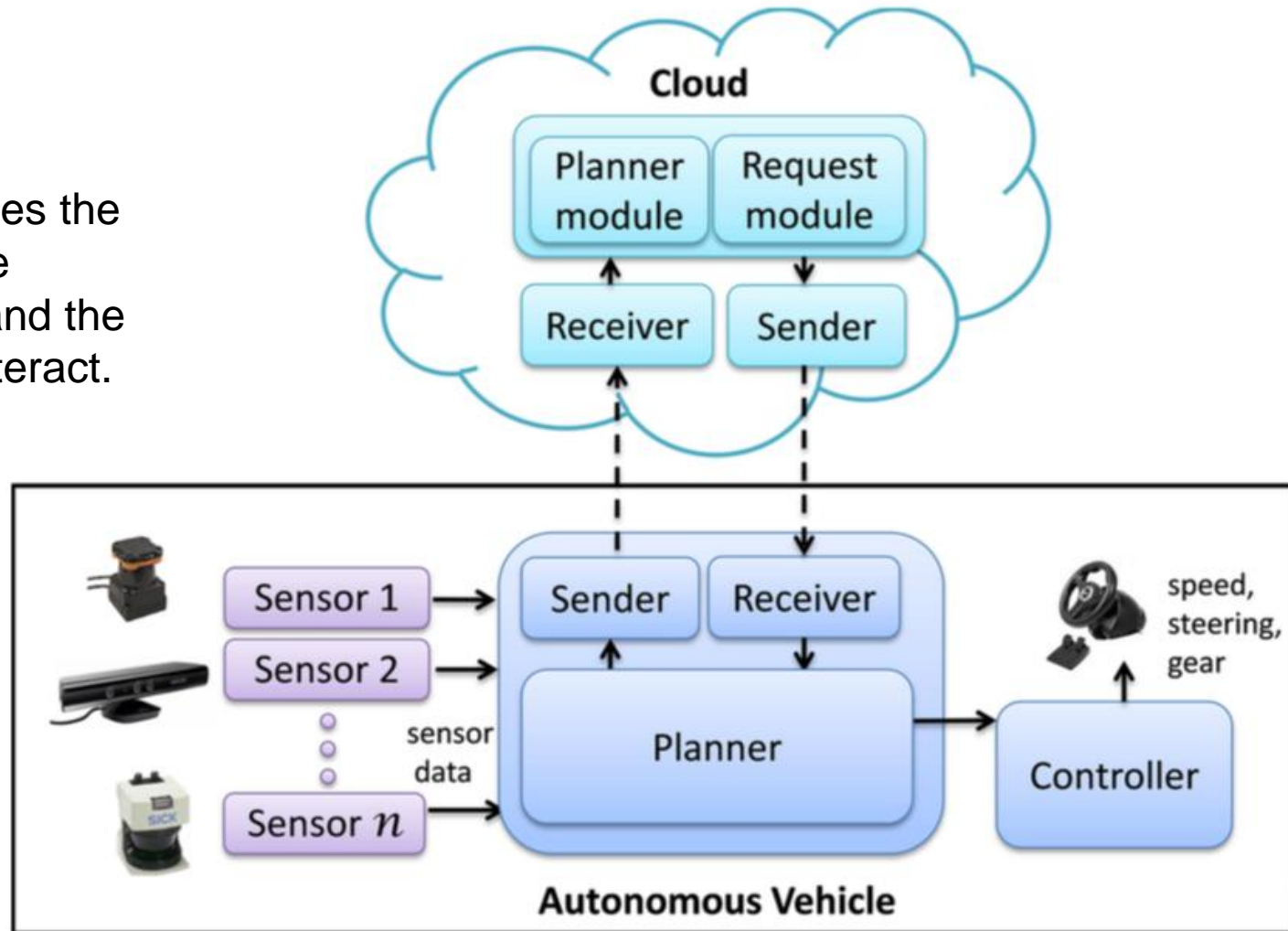
A CLOUD-ASSISTED DESIGN FOR AUTONOMOUS DRIVING

- **Carcel**: a cloud-assisted system for autonomous driving.
- Carcel enables the cloud to have access to sensor data from autonomous vehicles as well as the roadside infrastructure.
- The cloud assists autonomous vehicles that use this system to avoid obstacles such as pedestrians and other vehicles that may not be directly detected by sensors on the vehicle.
- Carcel enables vehicles to plan efficient paths that account for unexpected events such as road-work or accidents.
- A testbed as an autonomous golf car and 6 iRobot Create robots.
 - Results show that Carcel reduces the average time vehicles need to detect obstacles such as pedestrians by $4.6\times$.



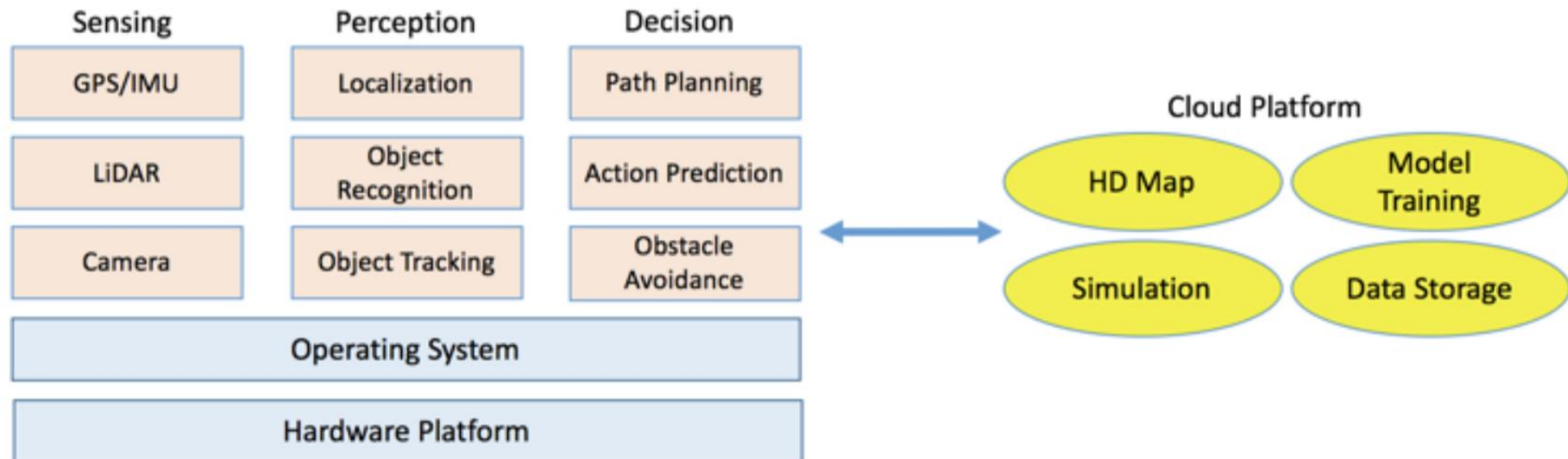
A CLOUD-ASSISTED DESIGN FOR AUTONOMOUS DRIVING

System Design illustrates the different modules in the autonomous vehicles and the cloud, and how they interact.




IMPLEMENTING A CLOUD PLATFORM FOR AUTONOMOUS DRIVING

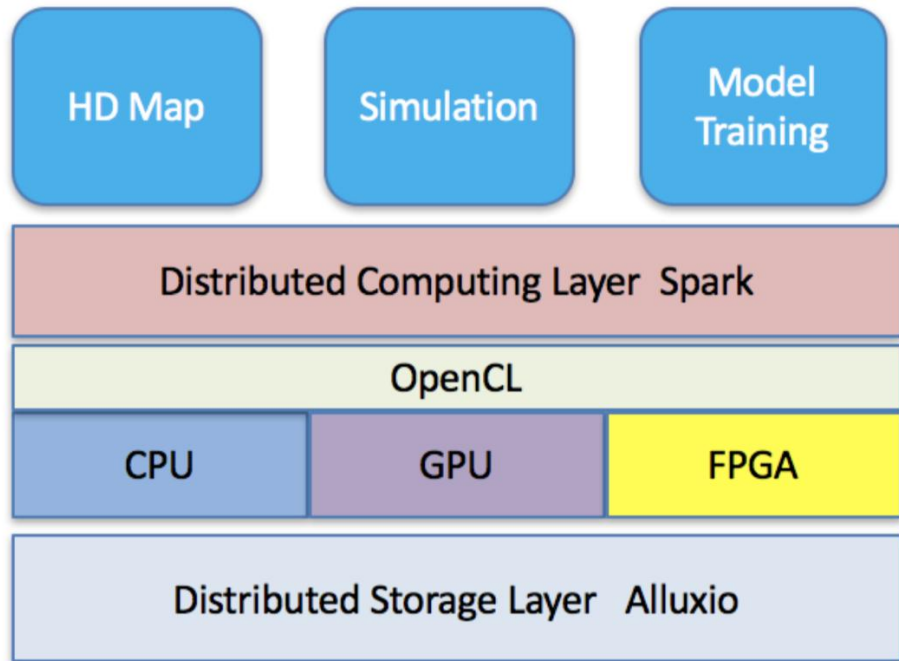
- Autonomous driving clouds provide essential services to support autonomous vehicles.
- It includes but not limited to distributed simulation tests for new algorithm deployment, offline deep learning model training, and High-Definition (HD) map generation.
- These services require infrastructure support including distributed computing, distributed storage, as well as heterogeneous computing.



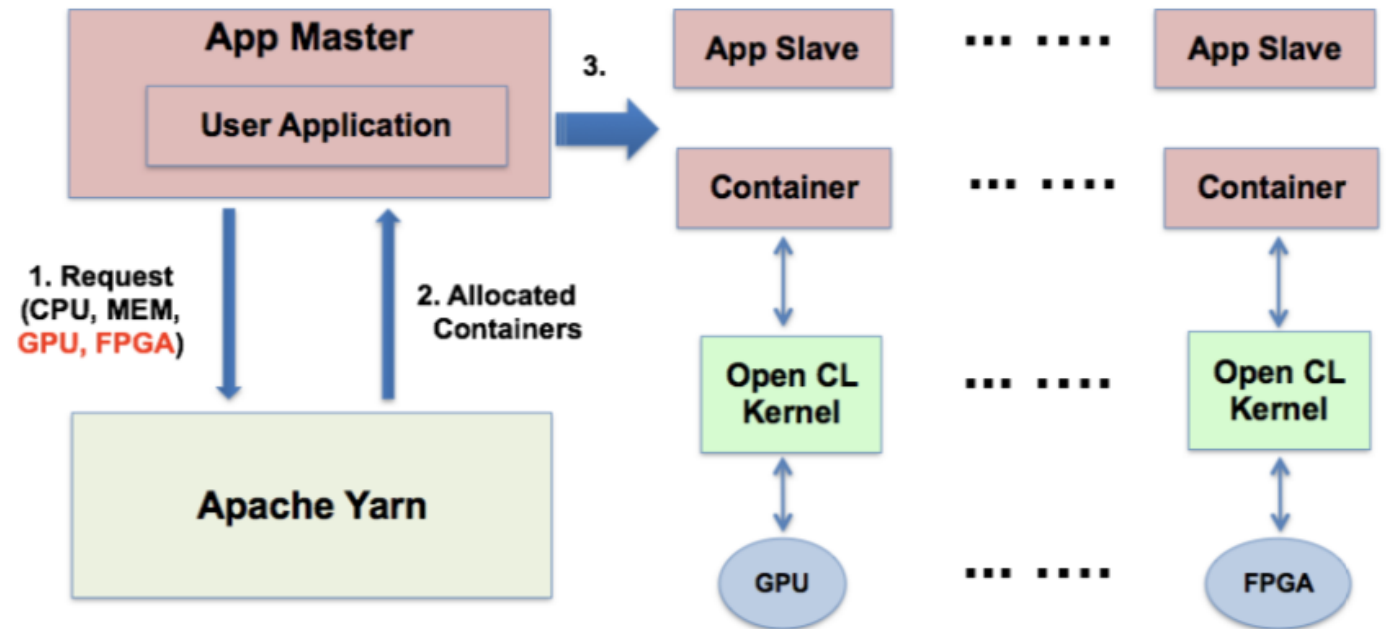
IMPLEMENTING A CLOUD PLATFORM FOR AUTONOMOUS DRIVING

- It use Spark for distributed computing, OpenCL for heterogeneous computing acceleration, and Alluxio for in-memory storage
 - Apache Spark provides programmers with an application programming interface centered on a data structure called the resilient distributed dataset (RDD), a read-only multiset of data items distributed over a cluster of machines maintained in a FT way.
 - To quantify perf., run production SQL queries on MapReduce and the Spark cluster.
 - With the same computing resources, Spark outperforms MapReduce by 5X on average.
 - It co-locates Alluxio with the compute nodes, and have Alluxio as a cache layer to exploit spatial locality; as a result, the compute nodes can read/write from/to Alluxio;
 - Alluxio then asynchronously persists data into the remote storage nodes.
 - Using it, manage to achieve a 30X speed up when compared to using HDFS only.
 - It uses YARN and Linux Container (LXC) for job scheduling and dispatch.
 - Utilizing these heterogeneous computing substrates will greatly improve performance as well as energy efficiency.
- 

IMPLEMENTING A CLOUD PLATFORM FOR AUTONOMOUS DRIVING



Cloud Platform for Autonomous Driving



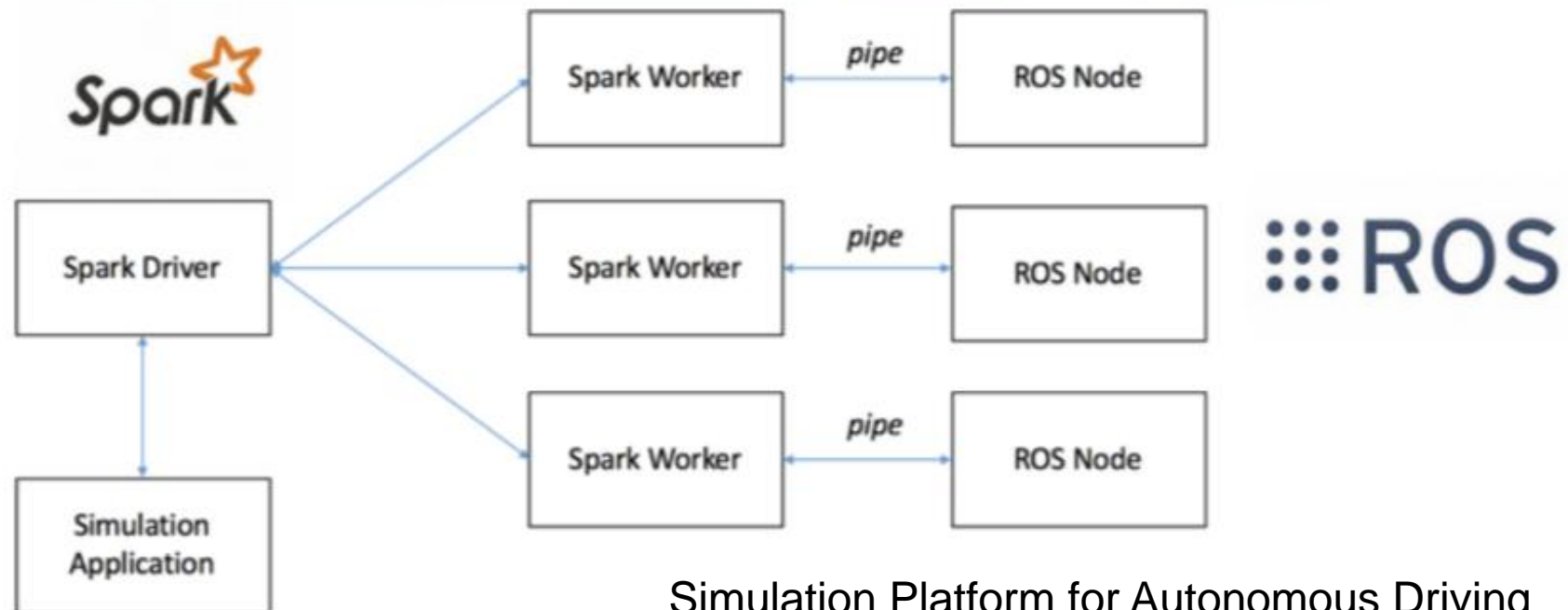
Distributed Heterogeneous Computing Platform

IMPLEMENTING A CLOUD PLATFORM FOR AUTONOMOUS DRIVING

- One simulation approach consists in replaying the data through ROS, where the newly developed algorithms are deployed for quick verification and early problem identification.
- Only after an algo. passes all simulation tests can it be to deploy for on-road testing.
- To seamlessly connect ROS and Spark, there are two problems to solve:
 - Spark by default consumes structured text data, for simulation let Spark consume multimedia binary data recorded by ROS such as raw or filtered readings from various sensors, detect obstacle bounding boxes from perception;
 - ROS needs to be launched in the native environ., where Spark lives in the managed environ.
- BinPipeRDD: a new design and implementation
 - The partitions of binary files go through encoding and serialization to form a binary stream;
 - The serialization stage will combine all bytes arrays into one single binary stream;
 - Then de-serialize and decode it into an understandable format;
 - After the target computation, the output would then be encoded and serialized;
 - The partitions can be returned to the Spark driver or be stored in HDFS as binary files.

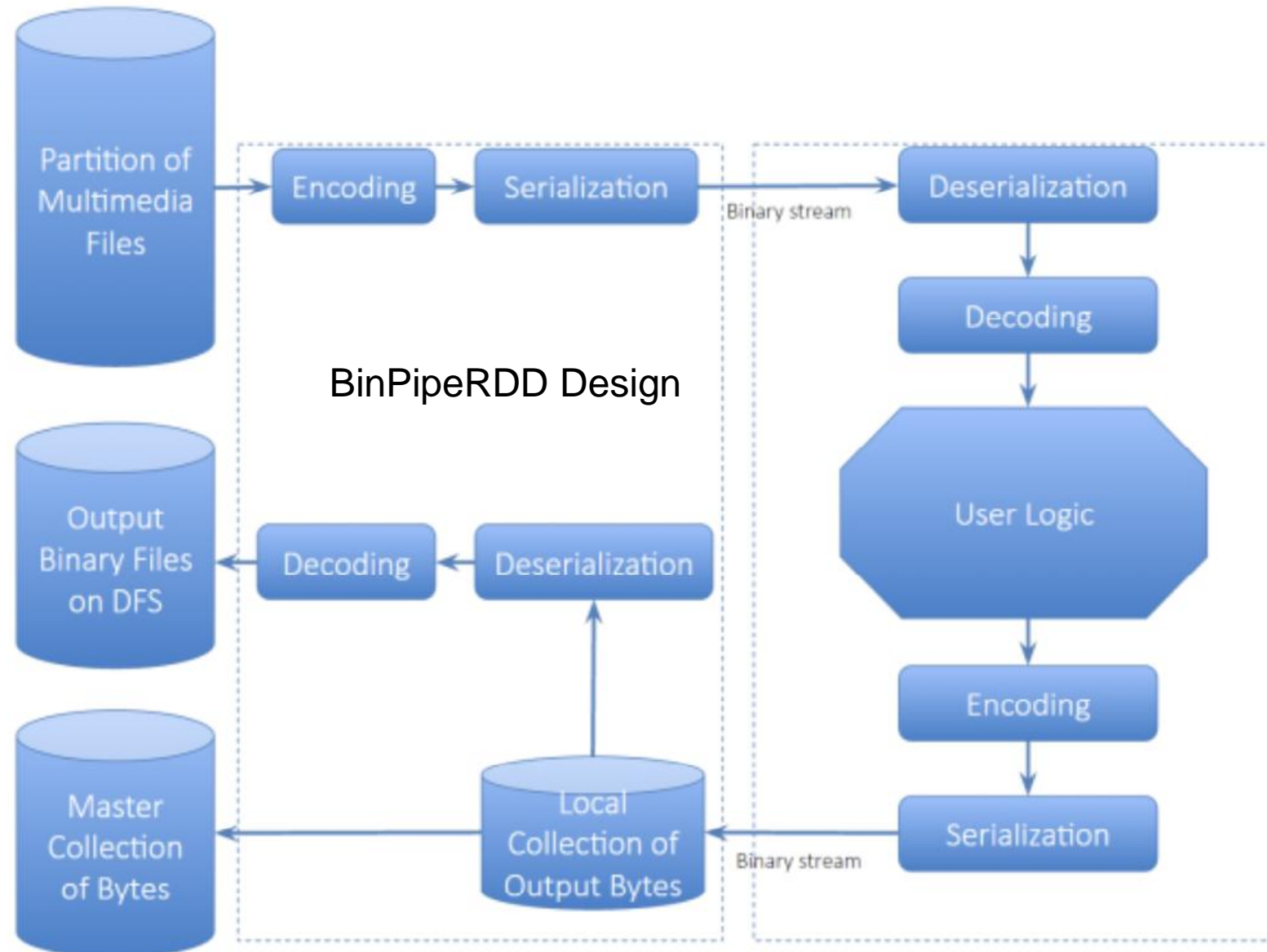


IMPLEMENTING A CLOUD PLATFORM FOR AUTONOMOUS DRIVING

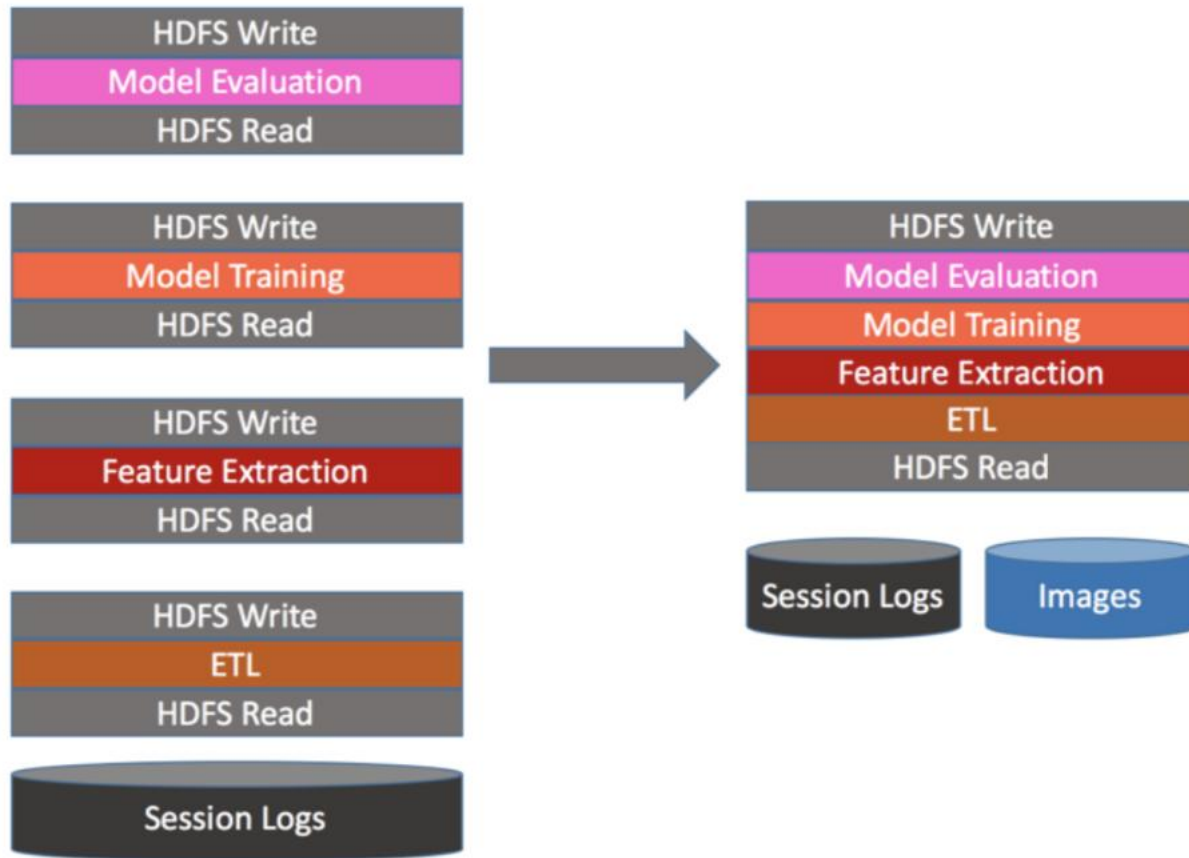


Simulation Platform for Autonomous Driving

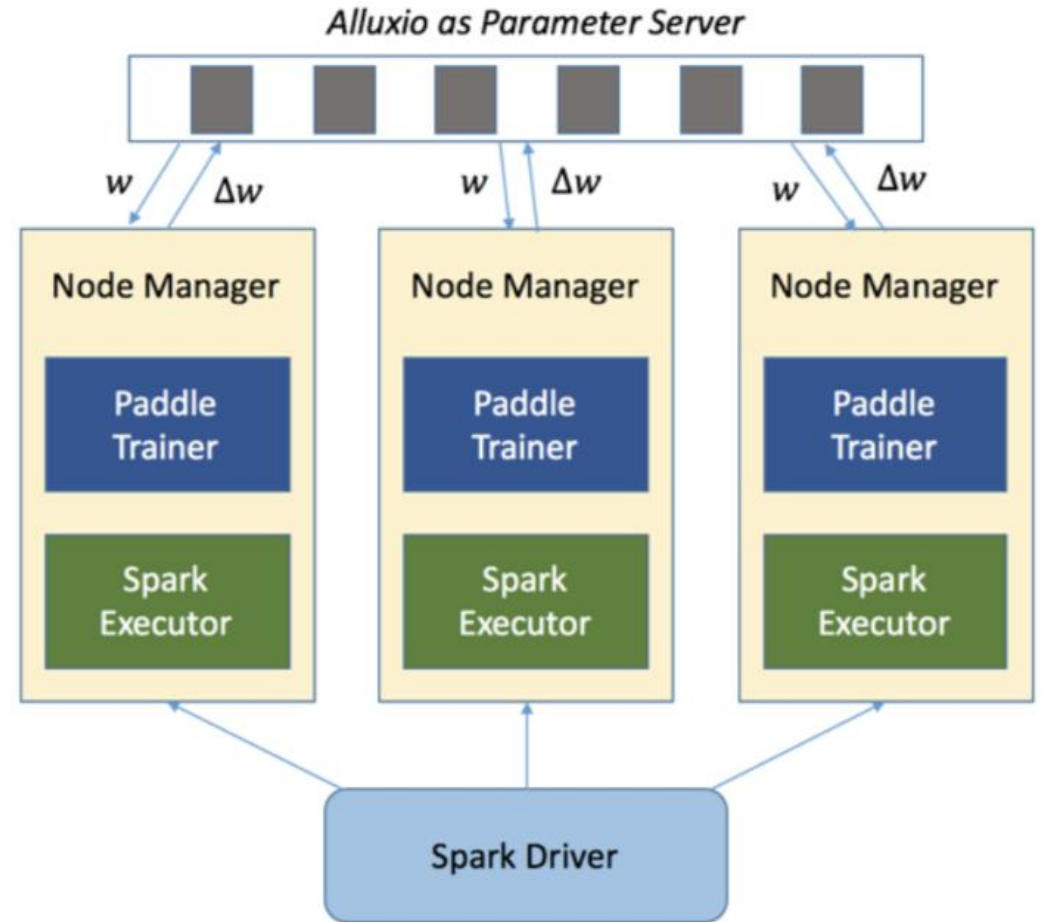
IMPLEMENTING A CLOUD PLATFORM FOR AUTONOMOUS DRIVING



IMPLEMENTING A CLOUD PLATFORM FOR AUTONOMOUS DRIVING

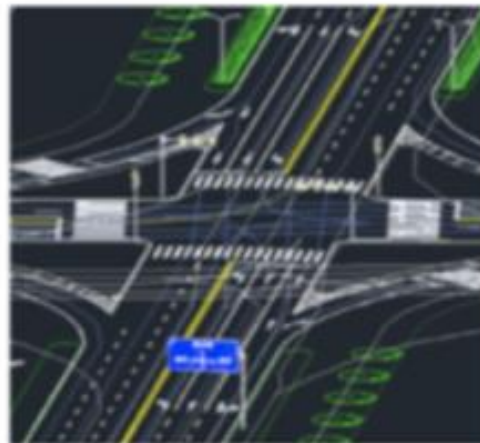
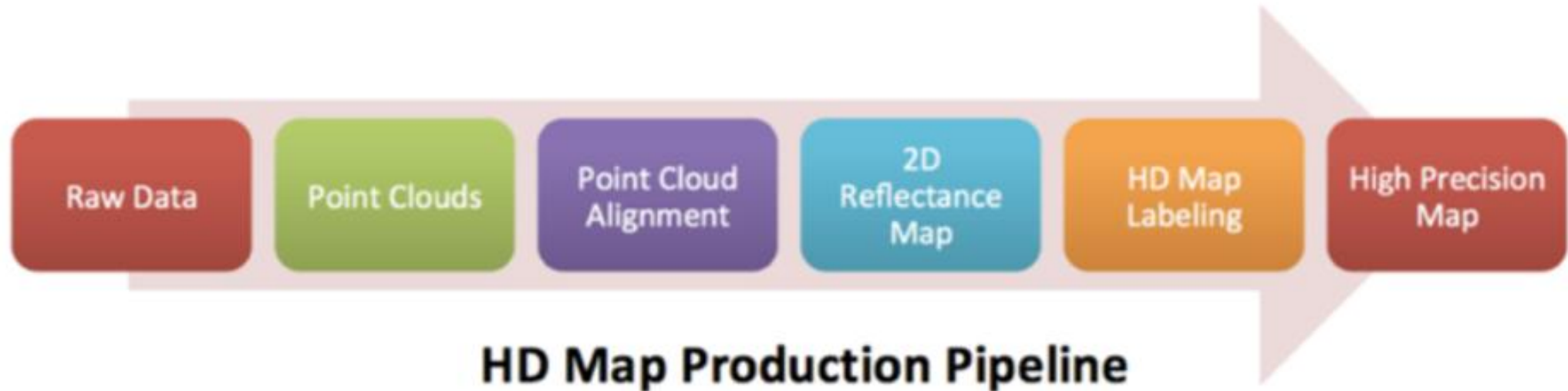


Training Platform for Model Training



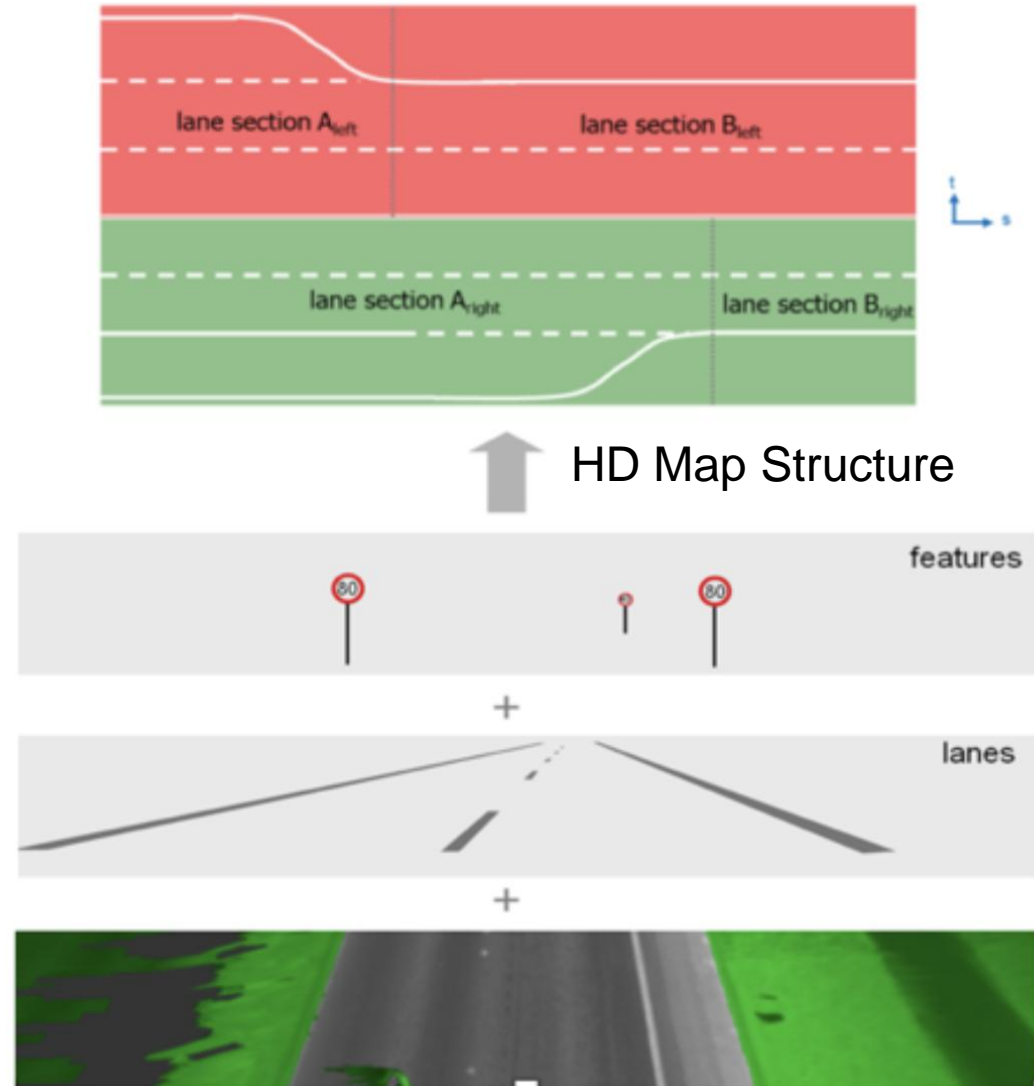
Training Platform for Autonomous Driving

IMPLEMENTING A CLOUD PLATFORM FOR AUTONOMOUS DRIVING



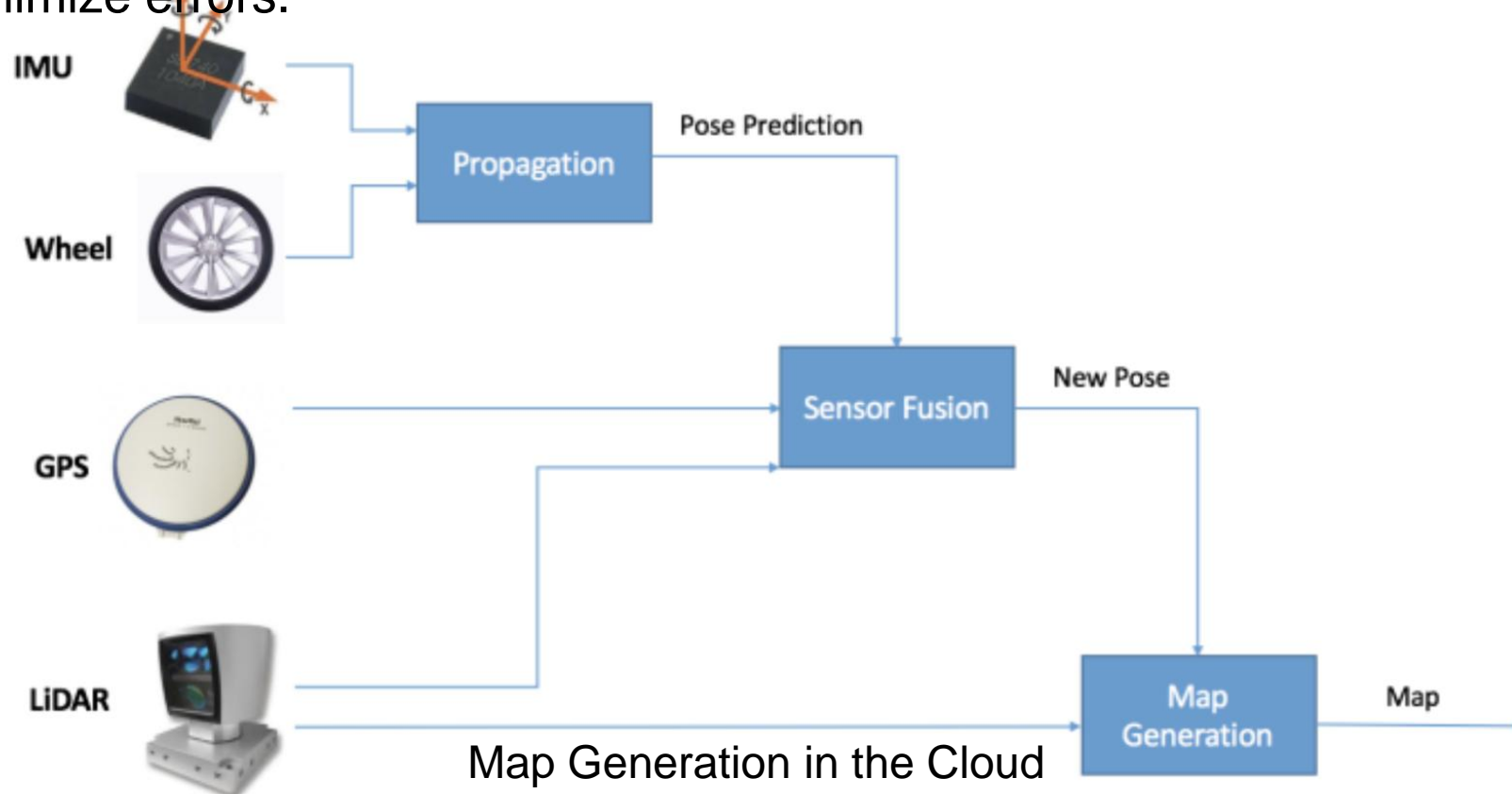
IMPLEMENTING A CLOUD PLATFORM FOR AUTONOMOUS DRIVING

- By using Spark and heterogeneous computing, reduce the IO between the pipeline stages and accelerate the critical path of the pipeline.
- HD maps have many layers of info.
 - At the bottom layer a grid map by LiDAR data, with a grid granularity of $\sim 5 \times 5 \text{cm}$.
 - It basically records elevation and reflection info. of the environ. in each grid.
 - As the autonomous vehicles are moving, compare in real time the new LiDAR scans against the grid map with initial position estimates provided by GPS/ IMU for precisely self-localizing in real-time.
 - On top of the grid layer, there are several layers of semantic info.: reference line, lane, traffic sign/light etc.



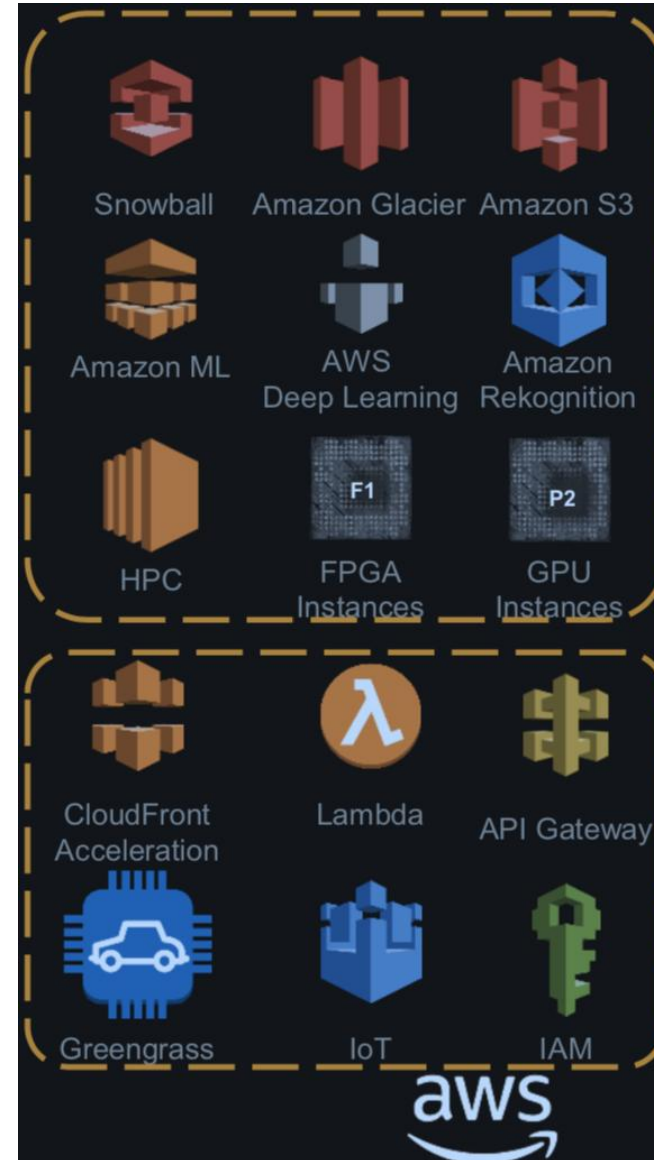
IMPLEMENTING A CLOUD PLATFORM FOR AUTONOMOUS DRIVING

HD map generation process fuses raw data from multiple sensors to derive accurate position info. The wheel odometry + IMU data are used to perform propagation, or to derive the displacement of the vehicle. Then the GPS + LiDAR data are used to correct the propagation results to minimize errors.

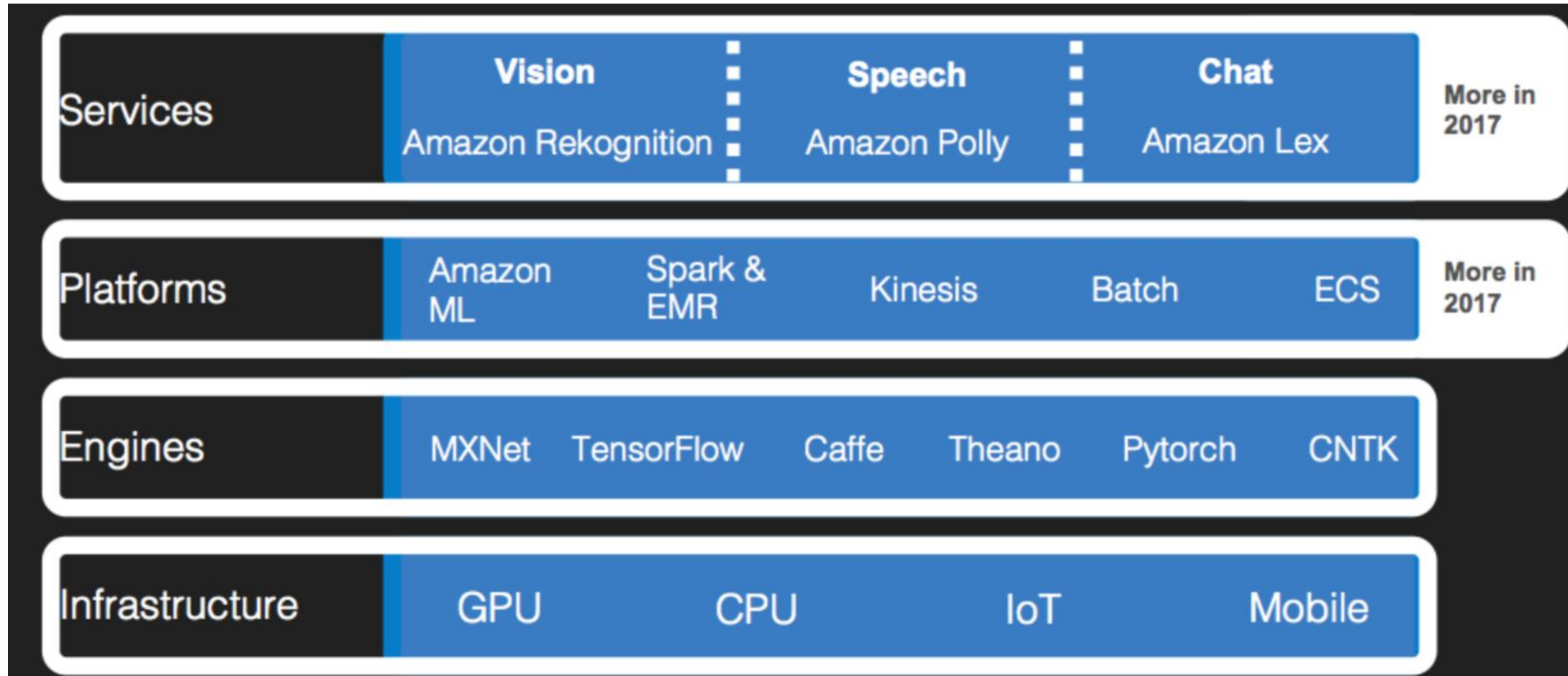


AUTONOMOUS DRIVING ALGORITHM DEVELOPMENT ON AMAZON AI

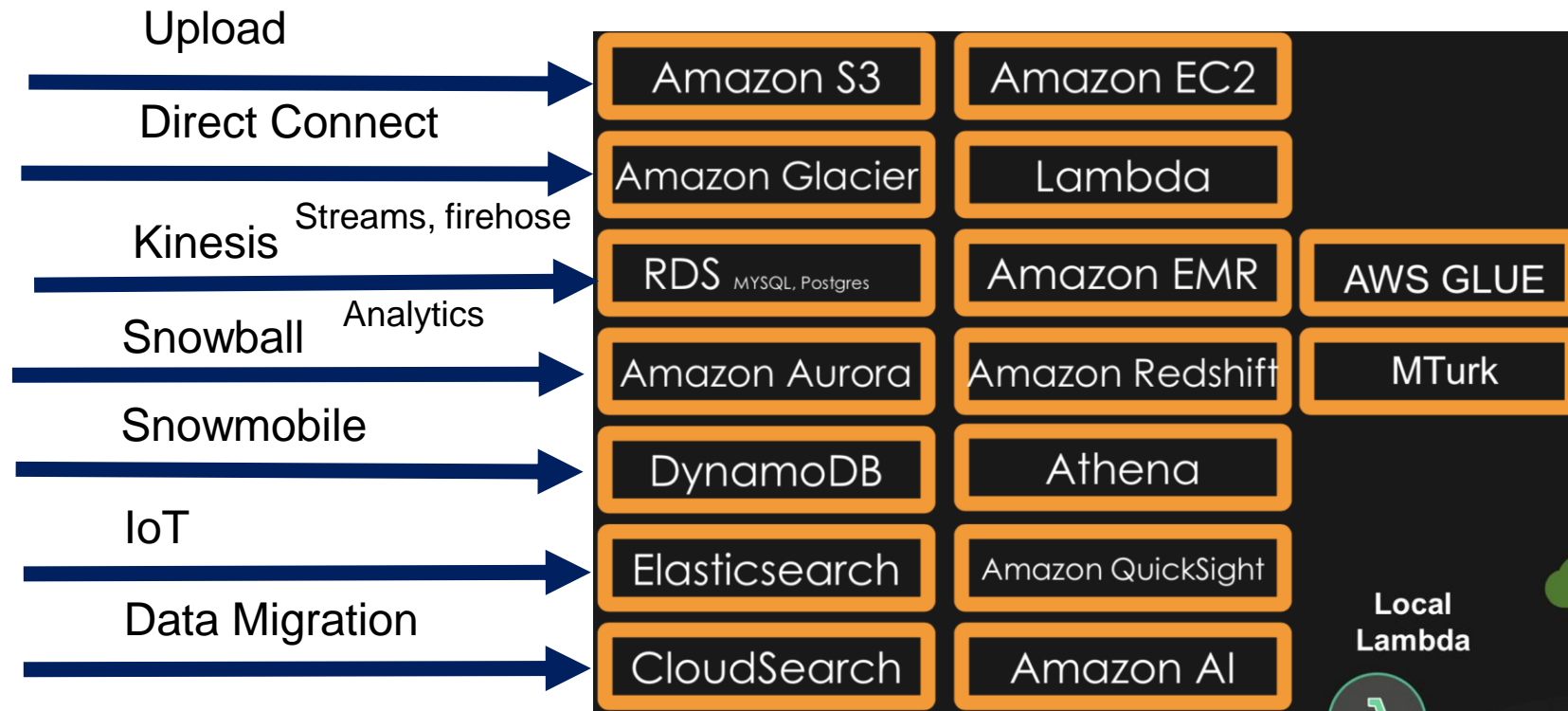
- AWS Capabilities:
 - Petabyte-scale, low cost, secure data transfer and storage
 - Cloud-native machine learning, deep learning, and AI
 - High performance compute including GPU and FGPA instances, on-demand
 - Serverless architecture
 - Content delivery with smart compression
 - Secure device integration with cloud edge compute



AUTONOMOUS DRIVING ALGORITHM DEVELOPMENT ON AMAZON AI



AUTONOMOUS DRIVING ALGORITHM DEVELOPMENT ON AMAZON AI



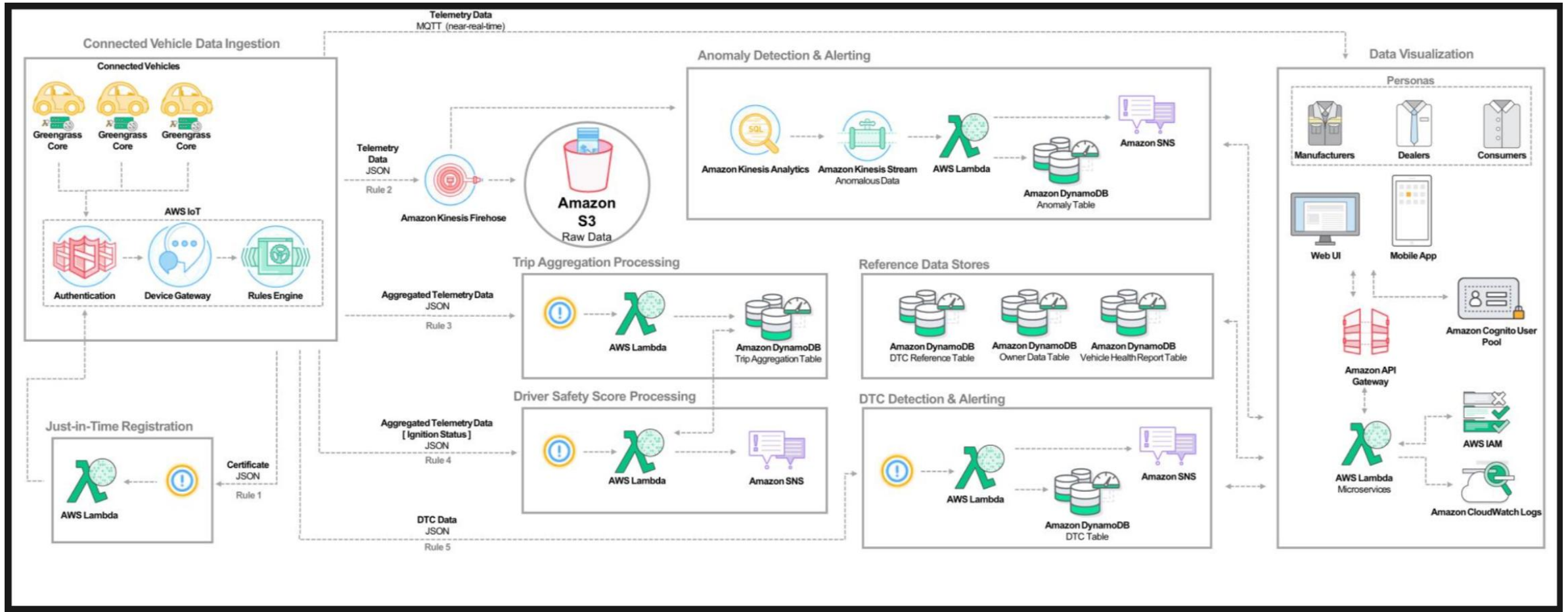
Data Collection and Organization

AWS-Connected Vehicle: Act locally

- Respond to local events quickly
- Operate offline
- Simplified device programming
- Reduce the cost of running IoT applications

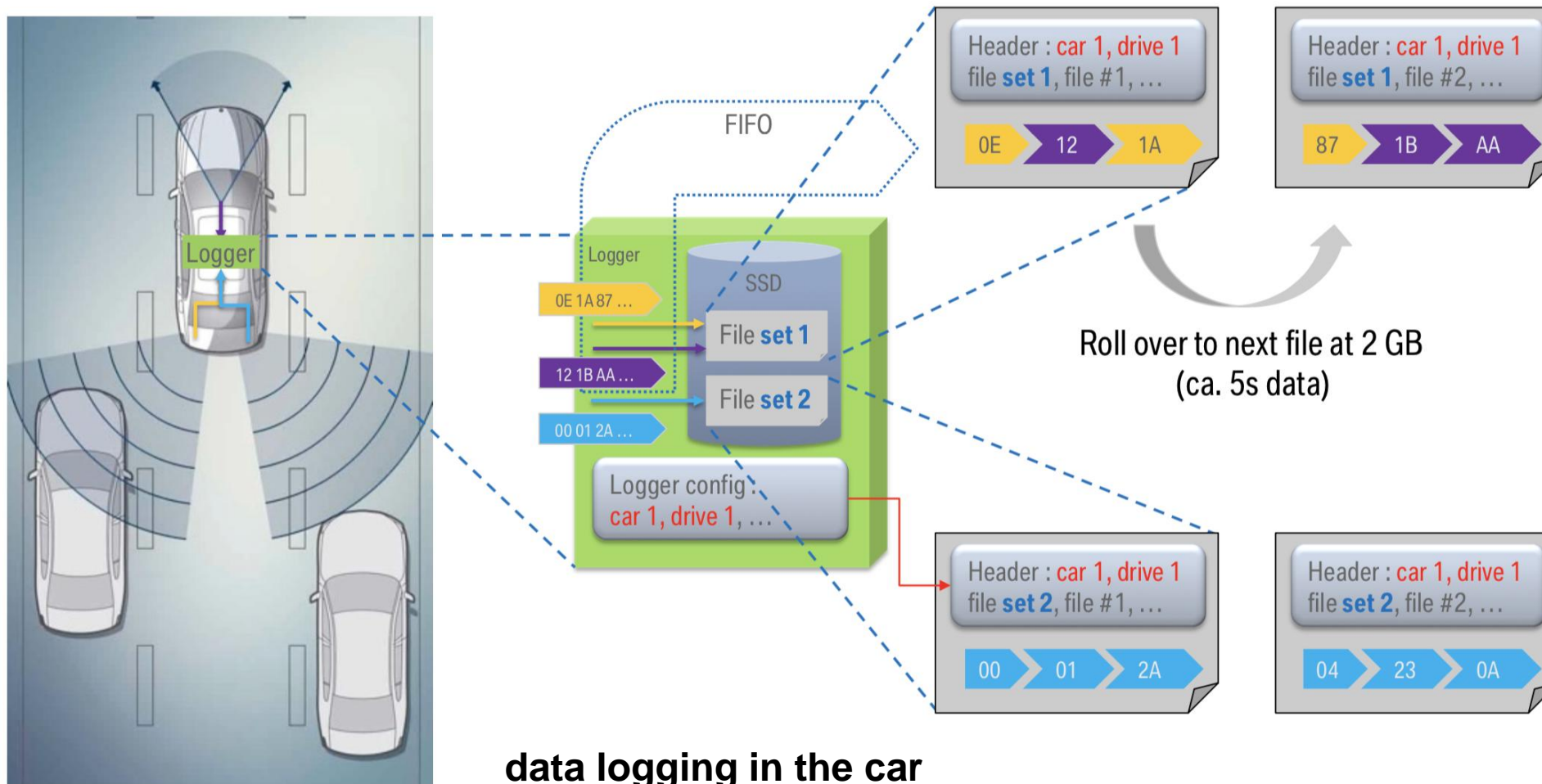


AUTONOMOUS DRIVING ALGORITHM DEVELOPMENT ON AMAZON AI

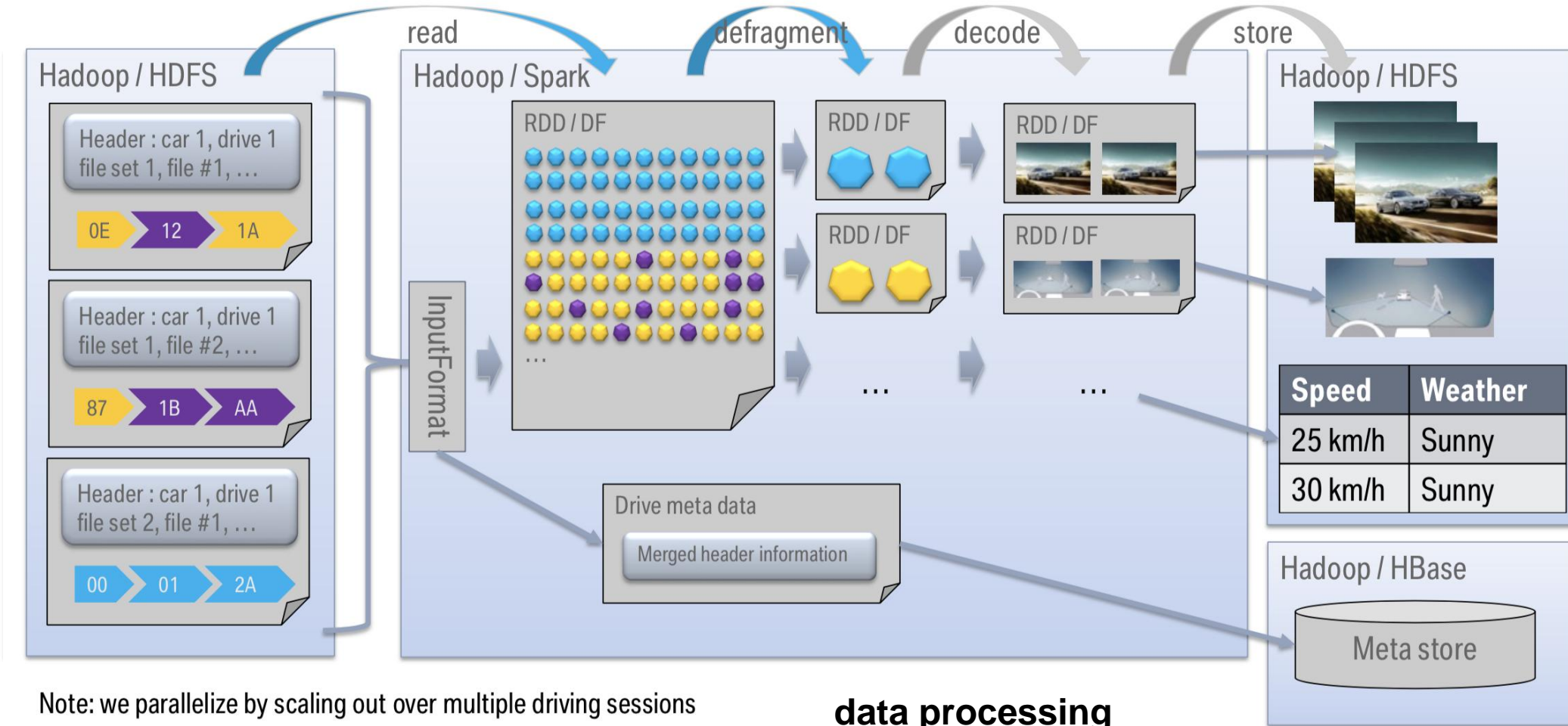


AWS-Connected Vehicle Architectures

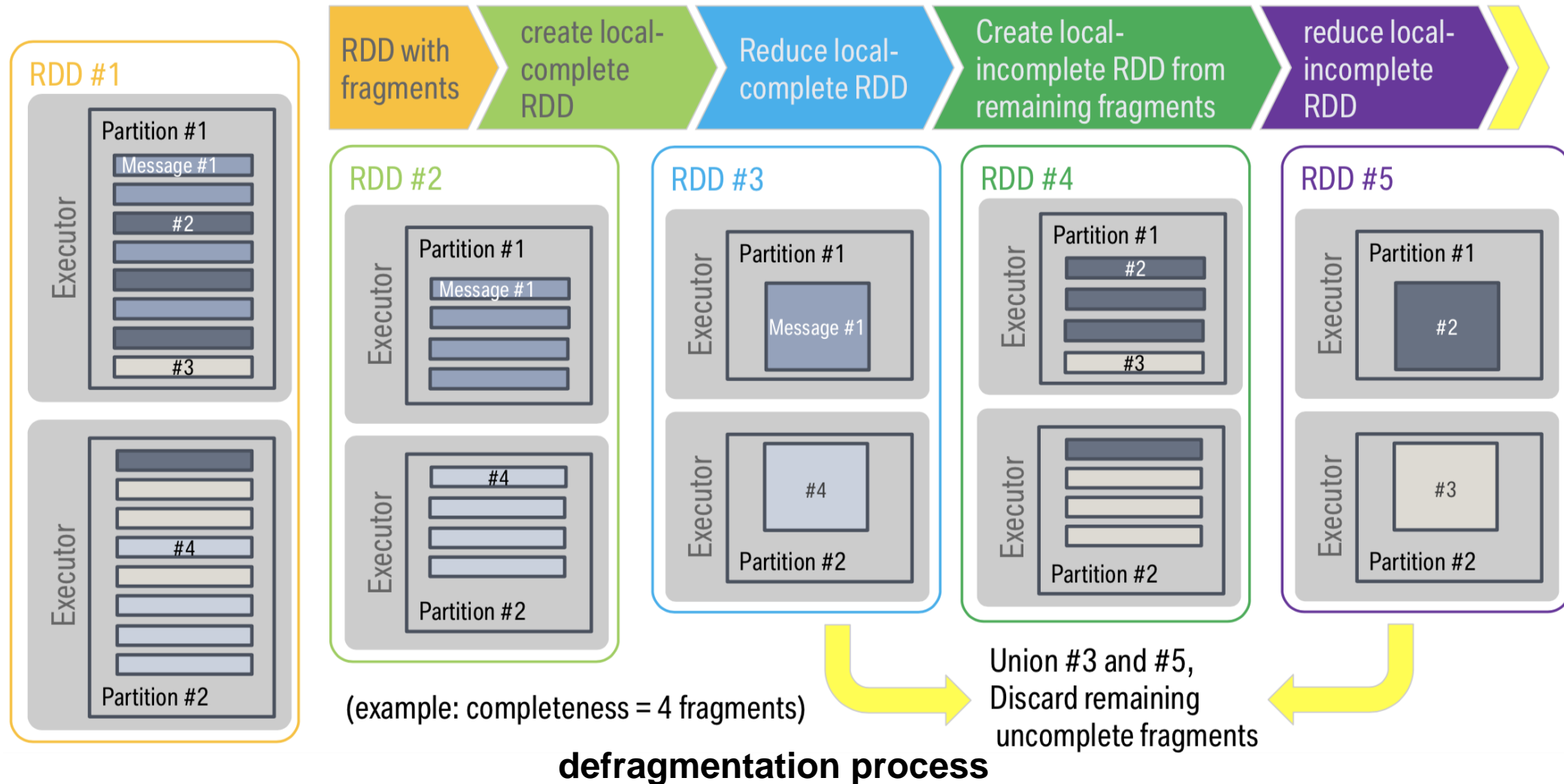
DATA DRIVEN DEVELOPMENT OF AUTONOMOUS DRIVING AT BMW



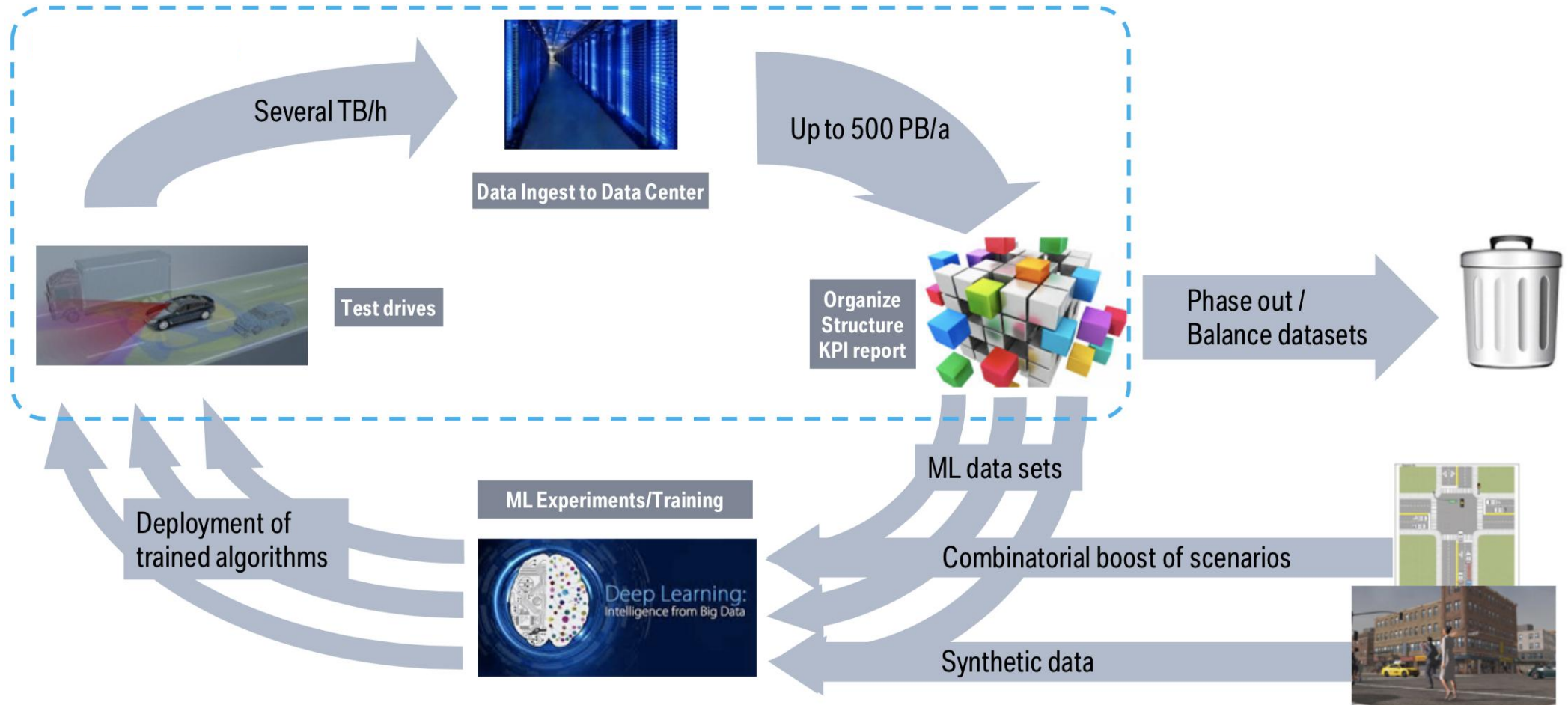
DATA DRIVEN DEVELOPMENT OF AUTONOMOUS DRIVING AT BMW



DATA DRIVEN DEVELOPMENT OF AUTONOMOUS DRIVING AT BMW



DATA DRIVEN DEVELOPMENT OF AUTONOMOUS DRIVING AT BMW



data driven development for autonomous driving

THANKS

