Agenda

- 1st day – Morning
  - Overview
    - Key message
    - Basics
    - Feature
    - Performance
    - Hardware support
  - Tools
  - STM32 MC Workbench
  - SDK components
    - Architectural Layer
    - MC Library
      - OOP – Object Oriented Programming
      - Our implementation of OOP
      - Interrupt Handling
      - Classes and interaction

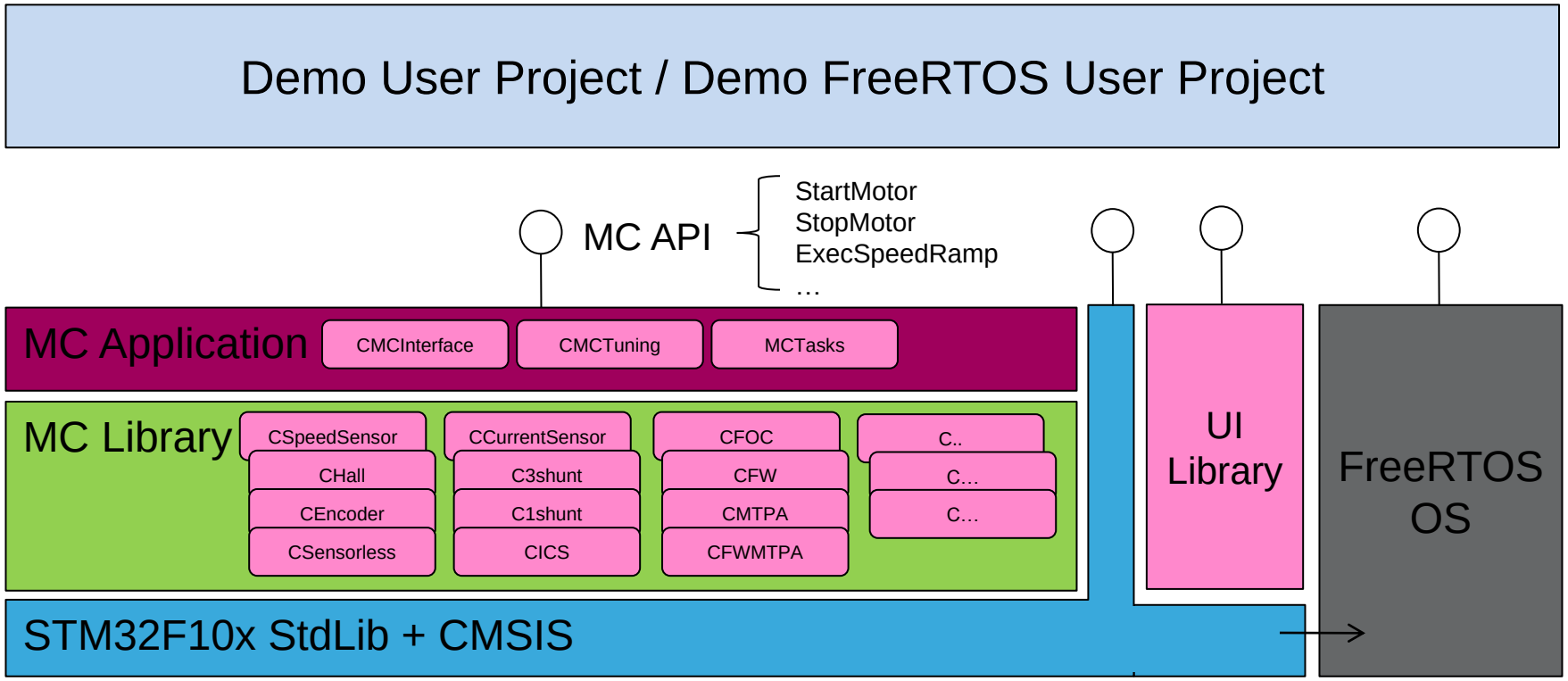
Architecture: SDK Components

- The STM32 PMSM FOC Library v3.2 is a Software Development Kit (SDK)

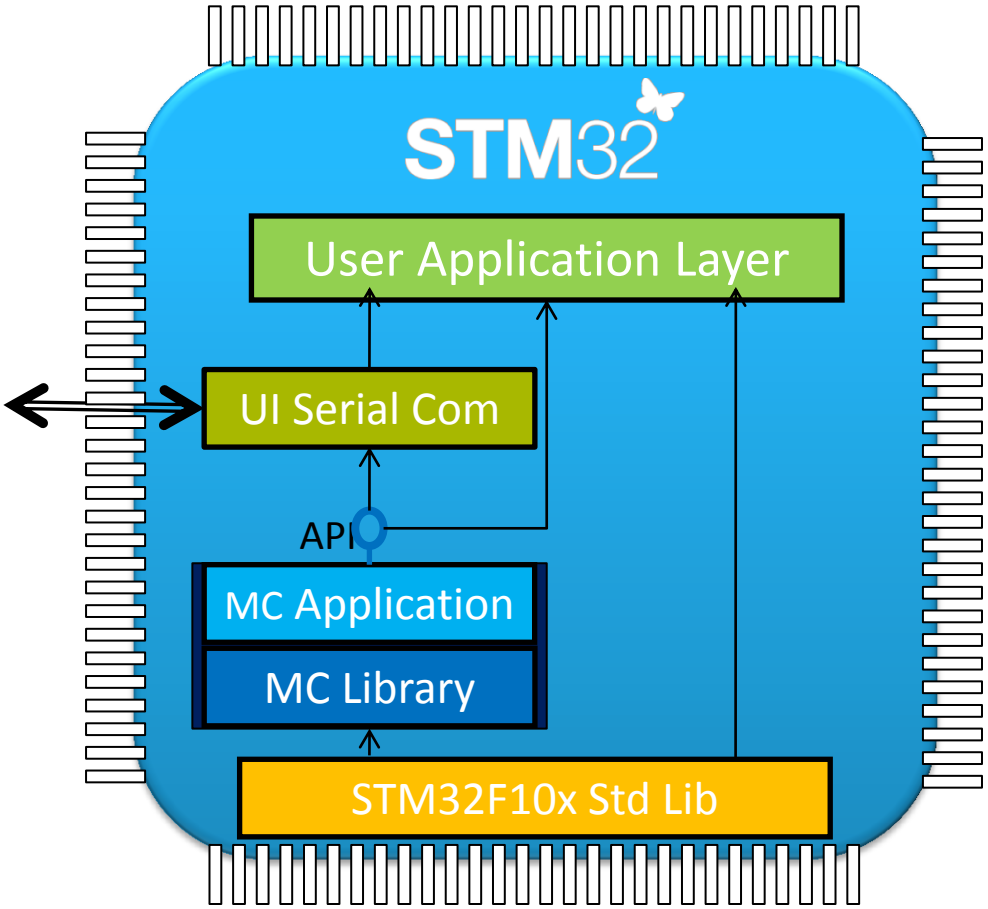
- The development tools included are:
  - MC Library (collection of classes, each describing elements involved in MC: sensors, algorithms…)
  - MC Application (implementation of the MC API, a set of high-level MC commands)
  - Demo User Project
  - FreeRTOS Demo User Project (alternative)
  - User Interface (utilities such as Serial communication, LCD/joystick manager,DAC..)
  - ST MC Workbench GUI (PC configuration tool)
  - Documentation (User Manual, Developer Manual, doxygen source file Help)

Architecture: Layers

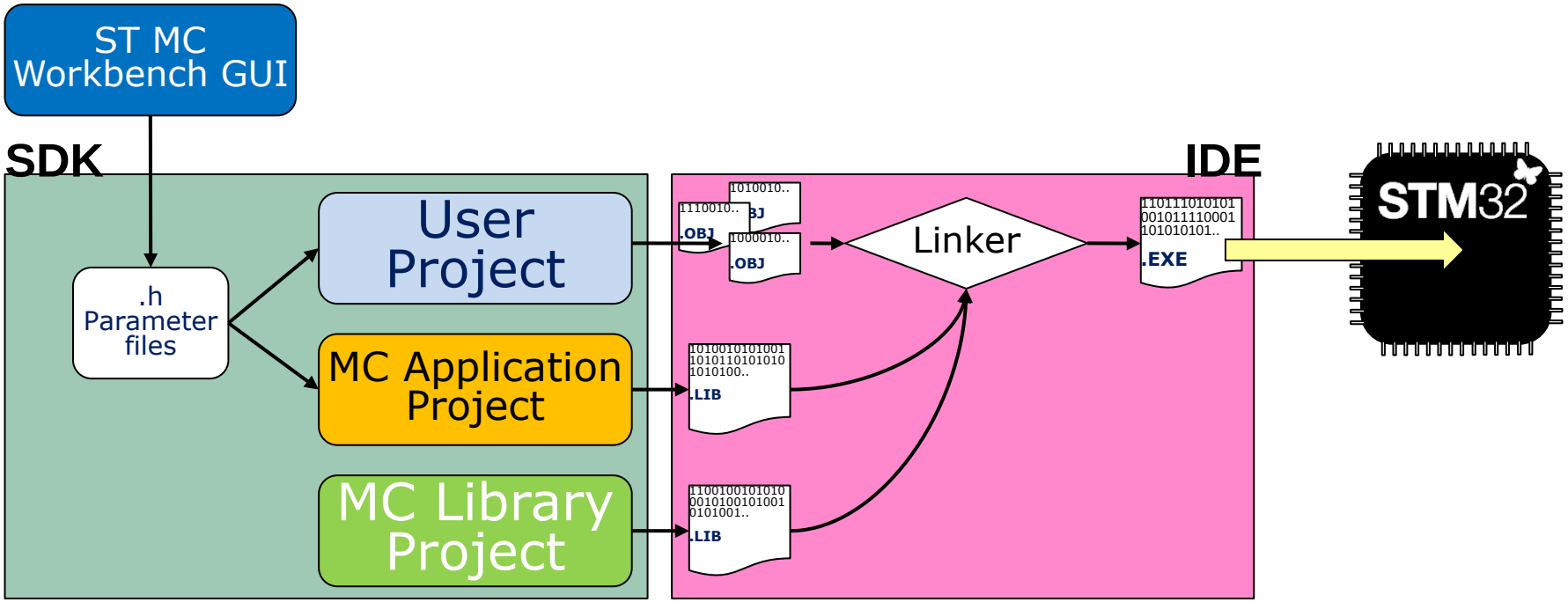
- According to components description, the SDK's architecture can be depicted as:



Architecture scenario

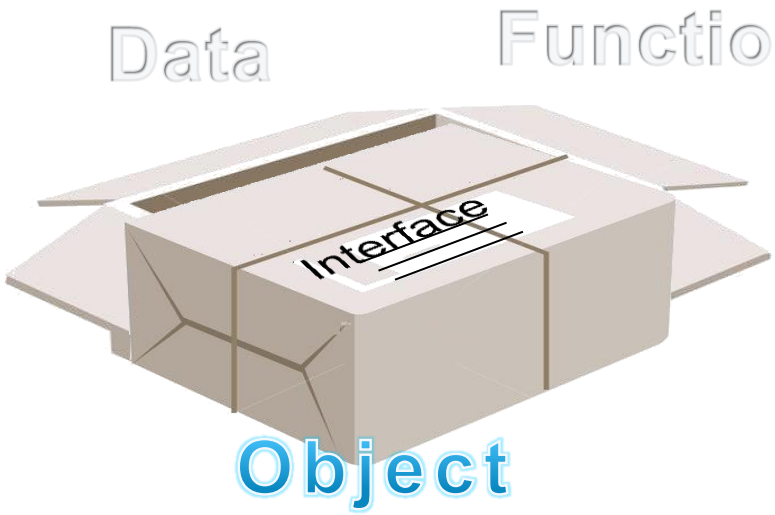


Architecture: Projects, Customization



- Parameter files, generated by the ST MC Workbench GUI, are used by the MC Application to instance objects from MC Library classes. The IDE rebuilds the Application project, links and creates the .exe

Object Oriented Programming (OOP)



- OOP is a programming style; the data and the functions that operate on it are packed together, forming an **Object**
- **Instance Methods** are functions associated with an object;
- Class is the factory from which objects are created. It is a user defined data type (variables, properties, methods).
- A dedicated C embodiment of OOP has been developed for implementing the MC Library

OOP fundamental concepts

- **Object**
  - The object is a collection of data structure (members) and functions (methods) allowed operating on the data structure itself.
  - Data structure contains both object properties and variables and can also be referenced as the state of the object
- **Class**
  - User defined data type containing variables, properties and methods
  - A class can be considered the factory from which individual objects are created
- **Method**
  - Methods are the only operations that can change the internal state of an object by modifying its variables and properties
  - Object internal variables are hided to object users: data encapsulation or data hiding
- **Interface**
  - The interaction of objects with the outside world is defined by the methods that they export
  - Methods form the interface with the outside world allowing a class to become more formal about the behavior it promise and provides

OOP fundamental concepts: Inheritance

- Inheritance
  - Process through which a class inherits the member and the methods of another class. This type of relationship is also known as child-parent or derived-base class.
  - Derived (child) classes are more specialized version of the base (parent) class as they inherit attribute and behavior from the base (parent) class but can also introduce their own.

State observer class

Speed sensor class

Methods	GetElSpeedDpp(…)
Variables	ElSpeedDpp
Parameters	bElToMecRatio

Specific methods IsObserverReliable(..)
Specific variables hBemf_alfa_est
Specific parameters hF1,…

Encoder class

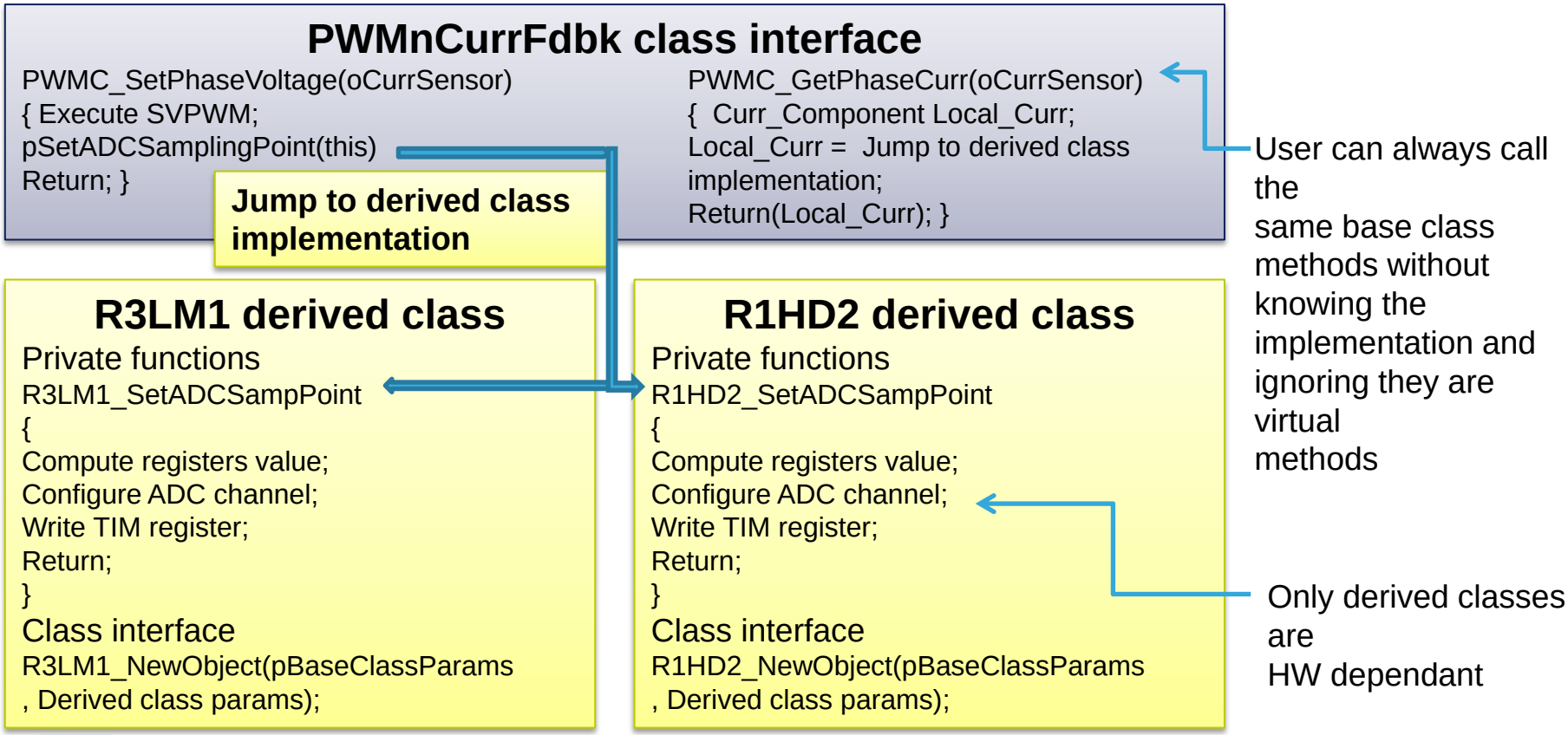
Speed sensor class

Methods	GetElSpeedDpp(…)
Variables	ElSpeedDpp
Parameters	bElToMecRatio

Specific methods ENC_SetDir…
Specific variables hPreviousCapture
Specific parameters ----

OOP fundamental concepts: Virtual methods

- Virtual methods
  - Base classes methods whose behavior depends on derived class specific implementation



OOP: advantages

- Code size efficient multiple motor control
  - Multiple instances of objects (e.g. two sensorless objects) do not require duplication of flash memory footprint
  - Base classes (e.g. GetElSpeedDpp) code is shared by any derived class instance
- Increased safety through data hiding
  - Object variables are only accessible through the object methods.
  - This prevents them from being accidentally modified → higher robustness for final applications (e.g. fuel pumps, applications related to human safety, …)
- Abstraction
  - Users only need to know how to use the object interface without taking care of background details or explanations
- Modularity
  - The source code for a class implementation can be written and maintained separately from other classes.
- Polymorphism
  - The implementation of the inheritance allows reducing the number of lines where it's required to distinguish function calls depending on configuration

… and the other side of the coin

- OOP it is not the best thing to be used in all circumstances:
  - Data hiding can make debug more difficult vs global variables
  - OOP can result in higher execution time and bigger flash memory footprint flash memory footprint
    - Our C implementation of OOP has been conceived so as to be slim and fast
    - Improvement brought to implementations of v2.0 functions compensated - in most of the cases - the introduction of the architecture related execution time
    - Sensor-less, 1 shunt configuration is even more performing compared to v2.0