CEC Continuing Education Center

DesignNews

Techniques for Interfacing with Modern Sensors

DAY 2 : Designing Sensor Interfaces

Sponsored by

Digi-Key ELECTRONICS

informa markets

Webinar Logistics

- Turn on your system sound to hear the streaming presentation.
- If you have technical problems, click “Help” or submit a question asking for assistance.
- Participate in ‘Group Chat’ by maximizing the chat widget in your dock.
- Submit questions for the lecturer using the Q&A widget. They will follow-up after the lecture portion concludes.

Course Sessions

- Introduction to Modern Sensor Interfacing
- Designing Sensor Interfaces
- Sensor Driver Techniques Part 1
- Sensor Driver Techniques Part 2
- Leveraging C++ in Sensor Interfacing

Example System

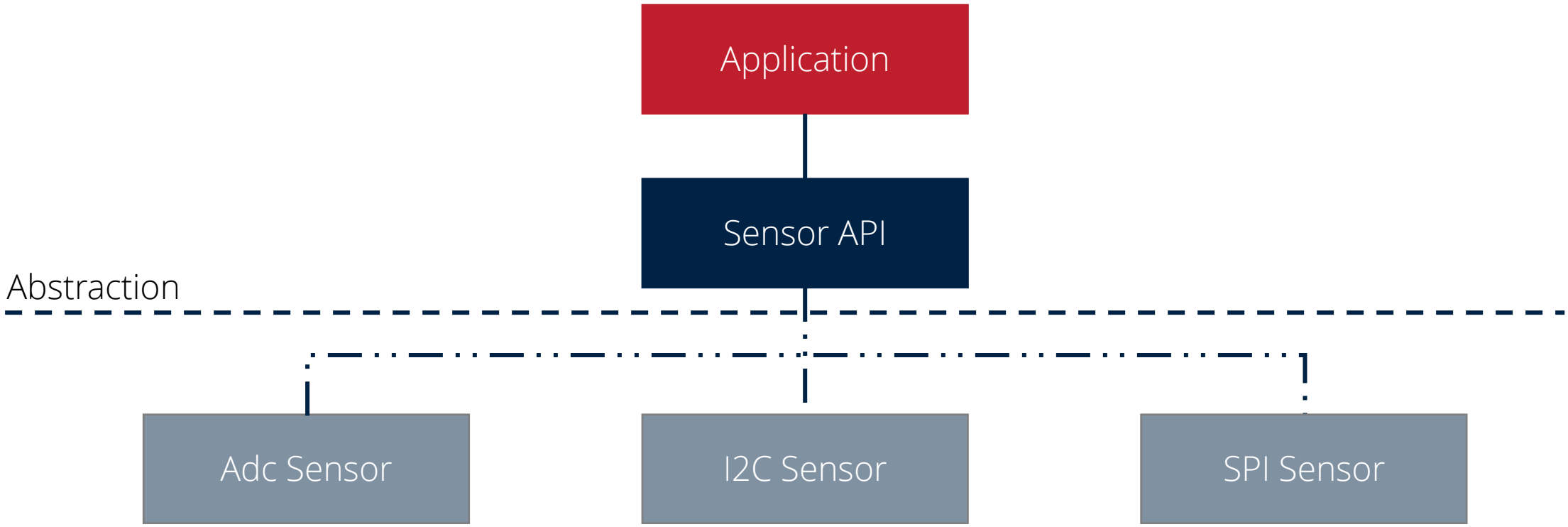
How many peripheral buses does your typical embedded system use for sensor interfacing?

- 1
- 2 – 4
- 5 – 6
- 7+

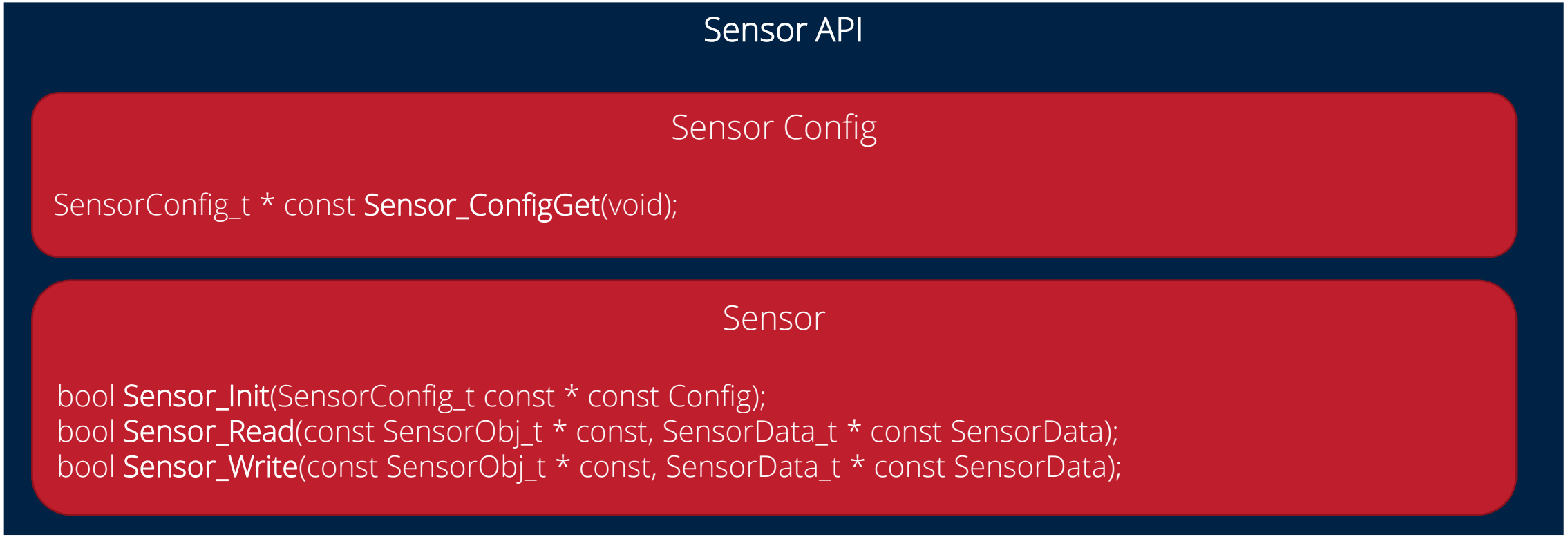
Benefits to creating an interface

- Reversing the code dependency direction
- Enhancing portability
- Abstracted complexity and low-level details
- Increasing reuse and scalability
- Simplifying software maintenance

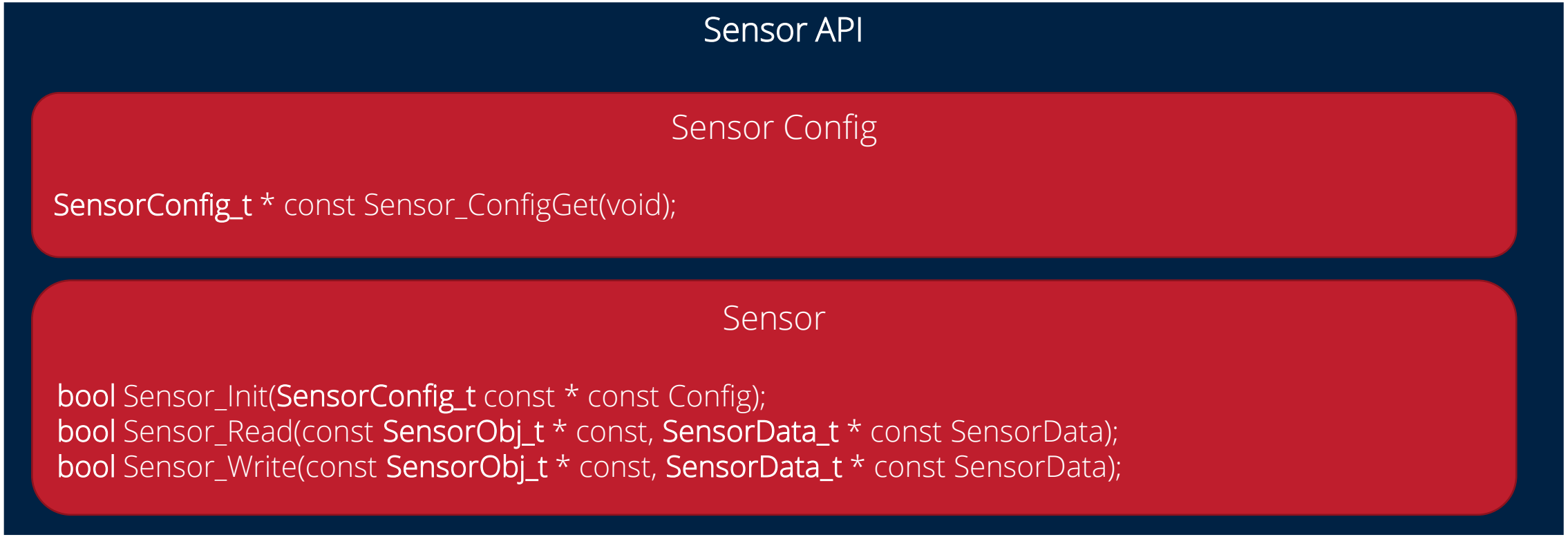
Abstraction



The Basic Sensor APIs – The Interface



The Basic Sensor APIs – The Types



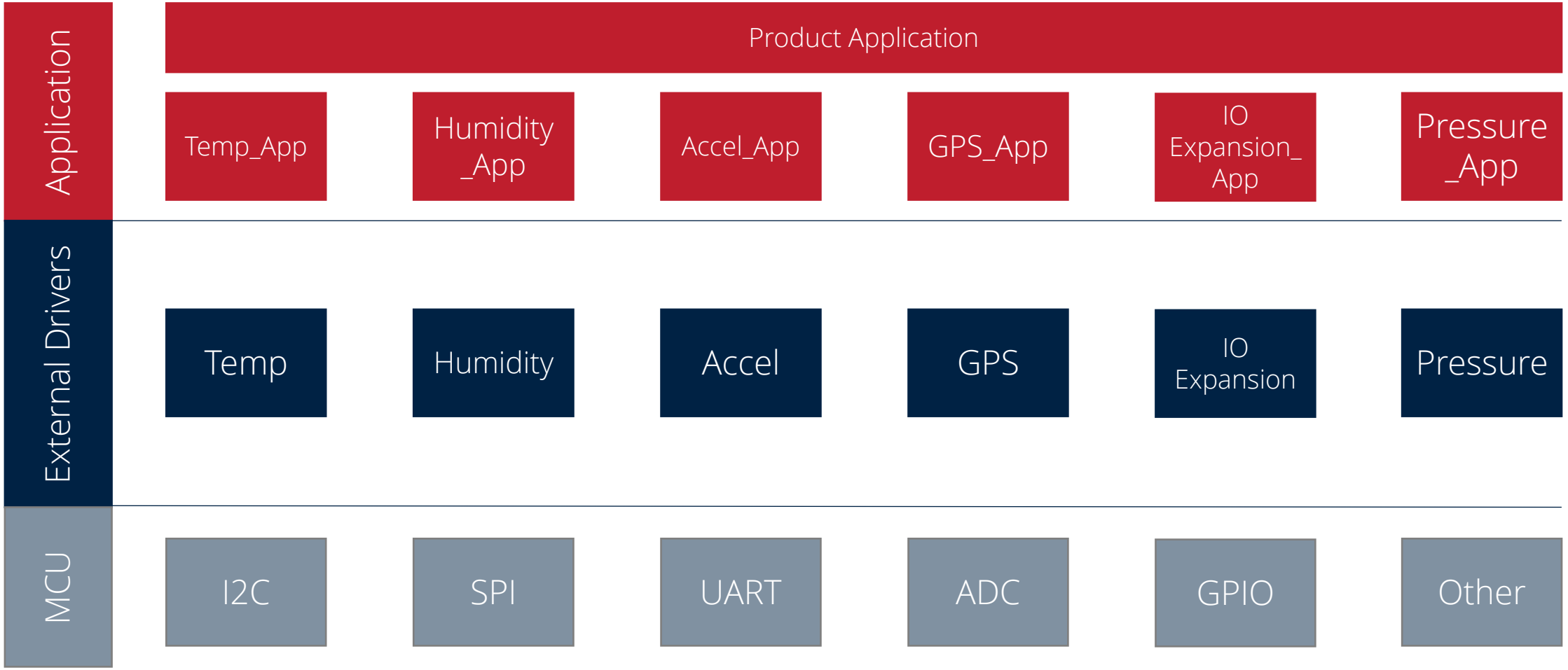
What consideration for the interface is most important to you?

- Retrieving an error codes
- Making sure that pointers are treated as constants
- The size of the interface
- Naming conventions used in the interface

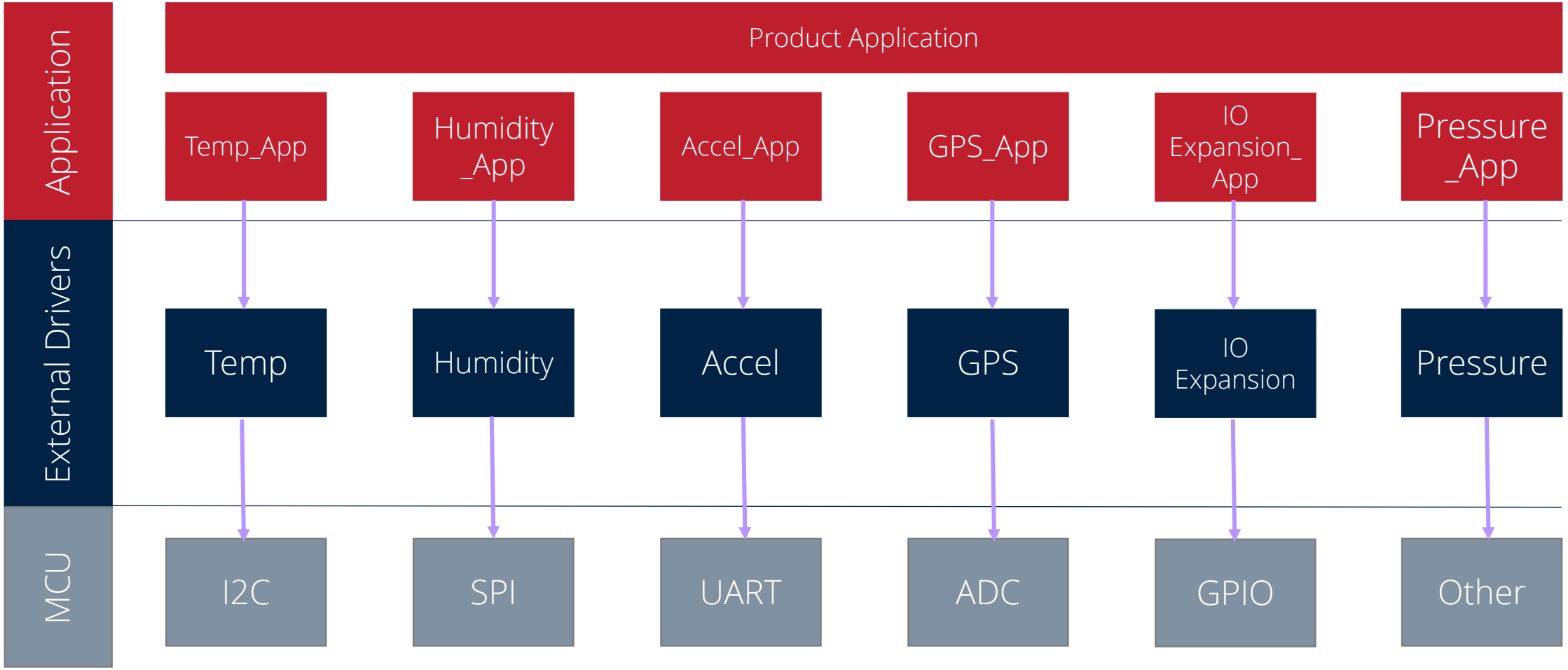
Interface Design Considerations

- Keep the interface to a dozen or less functions
- Functions should be memorable
- Function names should be descriptive
- Don’t leave out vowels
- Make sure that configurations are const
- Make pointers const
- Separate the application code from the driver

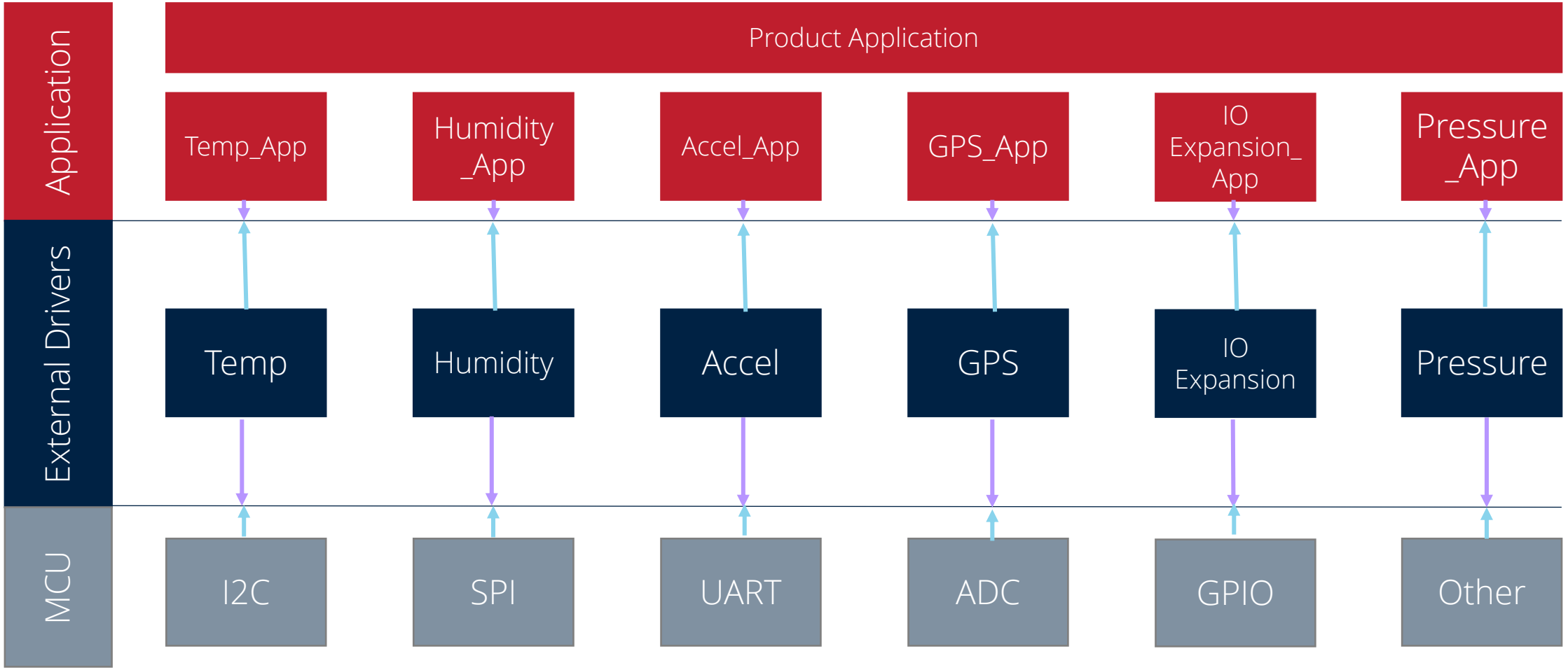
The Software Stack-up



Control Interface Dependencies – The Dependency Web



Control Interface Dependencies – Break Dependency



Is breaking dependencies and separating concerns worth the effort when using C?

- Yes
- No
- Not sure yet

The Sensor Interface as an Object

An object is a collection of data and operations that can be performed on that data.

- Sensors are an object
- Keep the object separate from the interface
- Keep the object separate from the implementation

The Sensor Interface as an Object

Function pointers can be used to specify the interface for a sensor!

```
typedef struct
{
    bool (*Init)( const SensorConfig_t * const Config);
    bool (*Read)(const SensorObj_t * const, SensorData_t * const SensorData);
    bool (*Write)(const SensorObj_t * const, SensorData_t * const SensorData);
} Sensor_t;
```

The Sensor as a separate object

```
const Sensor_t Analog =
{
    Adc_Init,
    Adc_Read,
    Adc_Write
};
```

```
const Sensor_t Gyro =
{
    Gyro_Init,
    Gyro_Read,
    Gyro_Write
};
```

The Sensor type behaves like an object!

```
Analog.Init(AdcConfig);
Analog.Write( … , … );
Analog.Read( … , … );

Gyro.Init(AdcConfig);
Gyro.Write( … , … );
Gyro.Read( … , … );
```

The trick, is that the sensor has its own custom behavior but “inherits” the interface that is defined by the typedef sensor structure!

We can also keep any sensor specific settings separate in an object structure, so the implementation does not contain the sensor information. This allows for multiple sensors of the same type without collision.

Thank you for attending

Please consider the resources below:

- www.beningo.com
  - Blog, White Papers, Courses
  - Embedded Bytes Newsletter
    - http://bit.ly/1BAHYXm

From www.beningo.com under

- Blog > CEC – Techniques for Interfacing with Modern Sensors

CEC Continuing Education Center

DesignNews

Thank You

Sponsored by

Digi-Key ELECTRONICS

informa markets