Developing Embedded Software before hardware is available

Michael Sørensen Loft

Education:

2002: B.Sc. EE. @ IHA

2009: Master of IT, Computer Science @ IT-Vest

Work:

SYSTEMATIC

2002 – 2006:

Software Engineer

Mjølner INFORMATICS

2006 – now:

Senior Software Architect

(in the embedded software department)

SKOV LINAK TERMA

Vestas VELUX

Mjølner

INFORMATICS

A **DESIGN** & **SOFTWARE** CONSULTANCY

FOUNDED IN 1988

- A dynamic team of 80 creative professionals
- Close collaboration between the UX design team and the technical development team
- Iterative and agile development process involving customer and user stakeholders

Embedded software

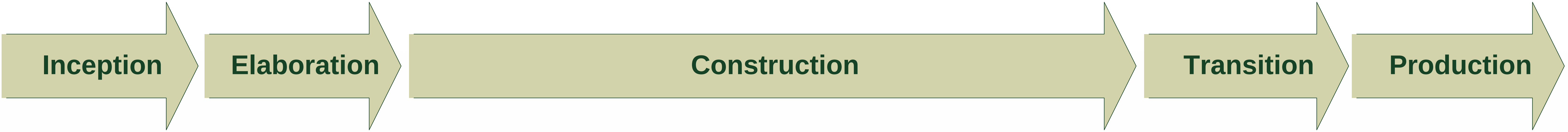
Embedded software

Characteristics

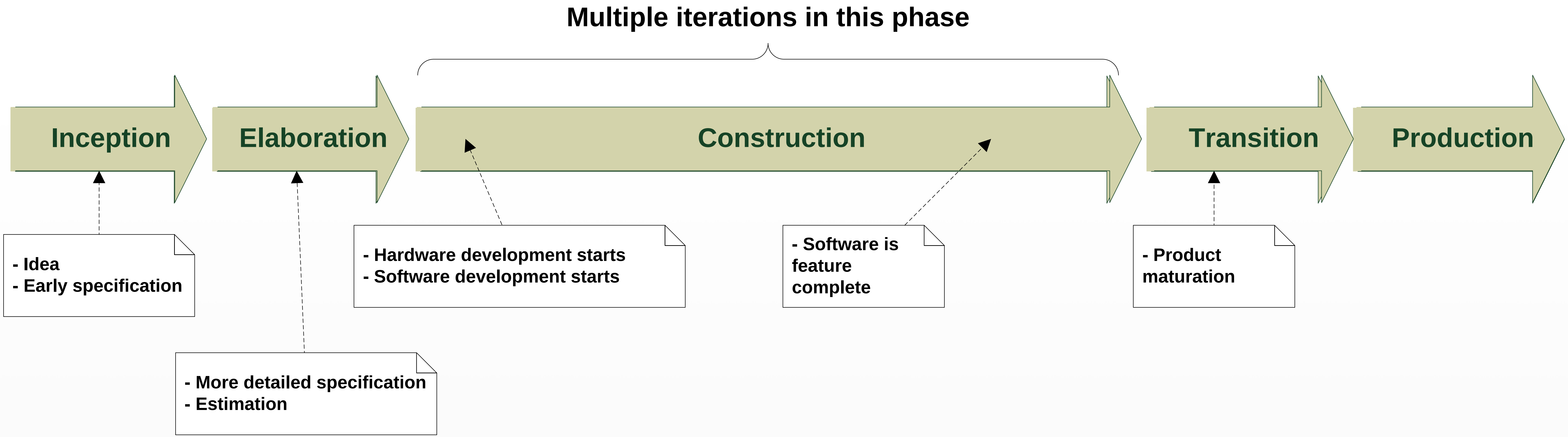
- Interacts with the real world -> people can get hurt or things can get destoyed
- Custom made hardware
- Often large number of devices
- Software bugs may result in recall of the product
- Time-to-market can be key to product succes

Project lifecycle

Project lifecycle



Project lifecycle



Multiple iterations in this phase

Inception

Elaboration

Construction

Transition

Production

- Idea
- Early specification

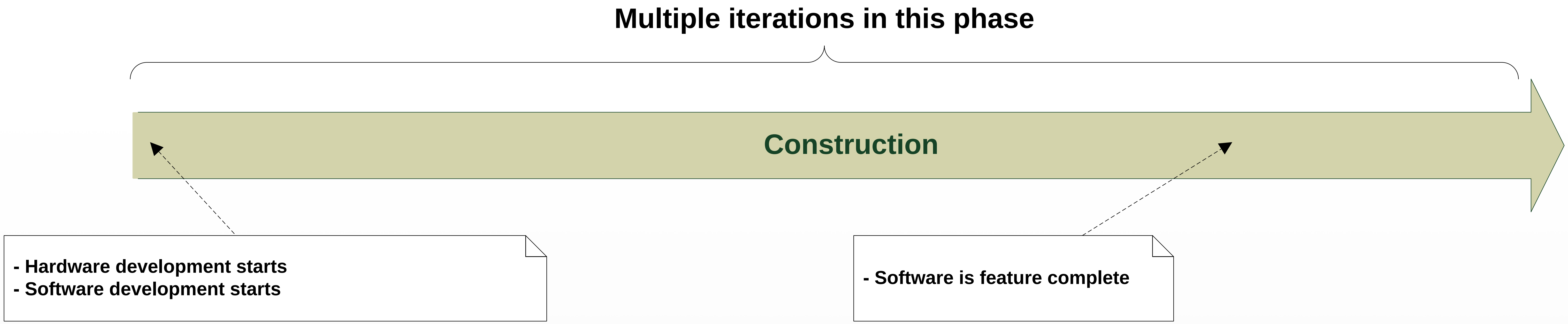
- More detailed specification
- Estimation

- Hardware development starts
- Software development starts

- Software is feature complete

- Product maturation

Project lifecycle



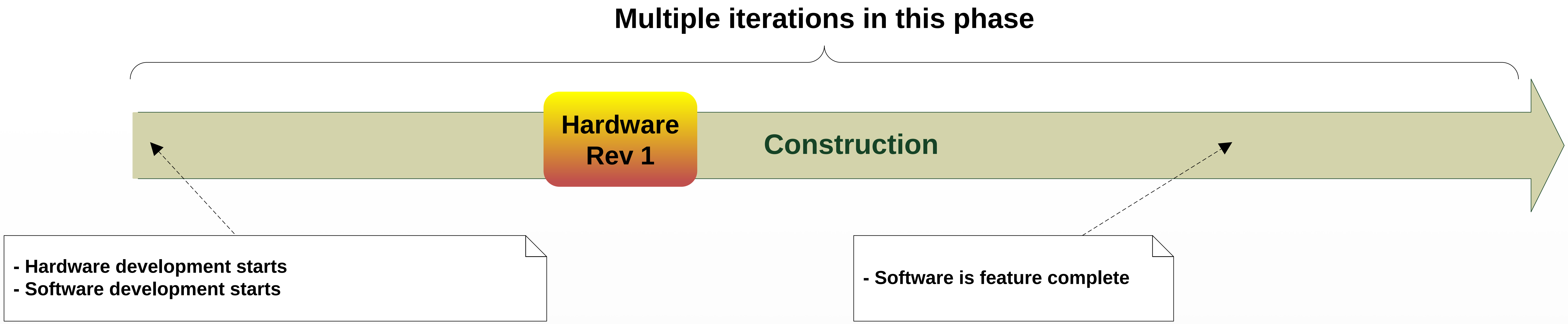
Multiple iterations in this phase

Construction

- Hardware development starts
- Software development starts

- Software is feature complete

Project lifecycle



Multiple iterations in this phase

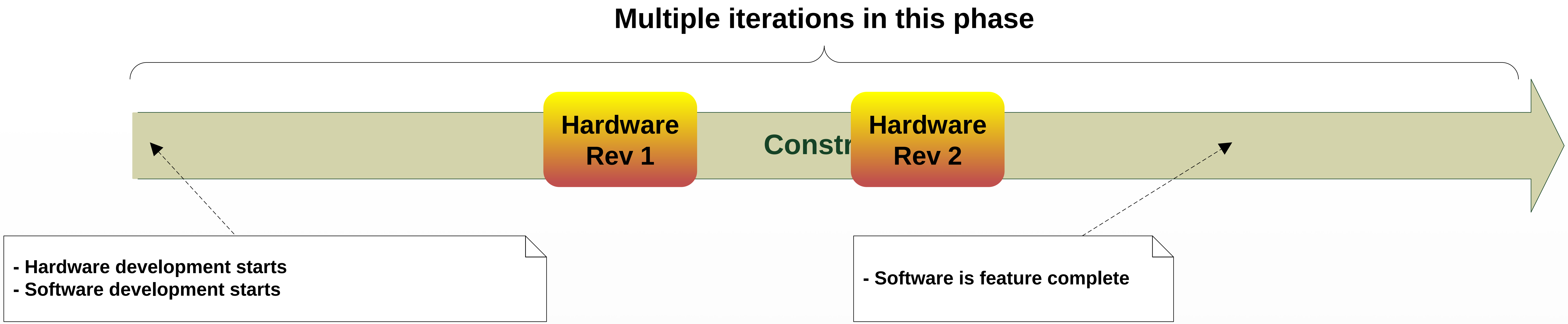
Hardware Rev 1

Construction

- Hardware development starts
- Software development starts

- Software is feature complete

Project lifecycle



Multiple iterations in this phase

Hardware Rev 1

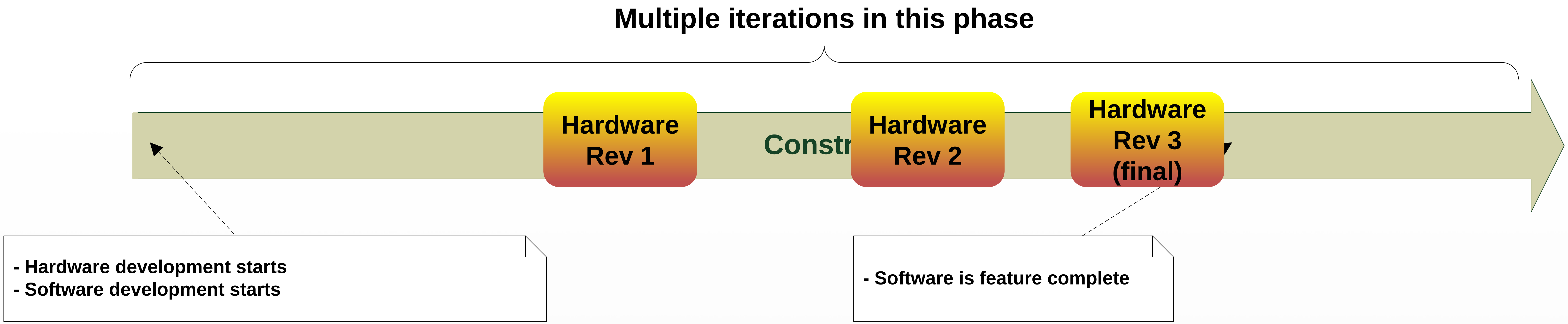
Constr

Hardware Rev 2

- Hardware development starts
- Software development starts

- Software is feature complete

Project lifecycle



Multiple iterations in this phase

Constr

Hardware Rev 1

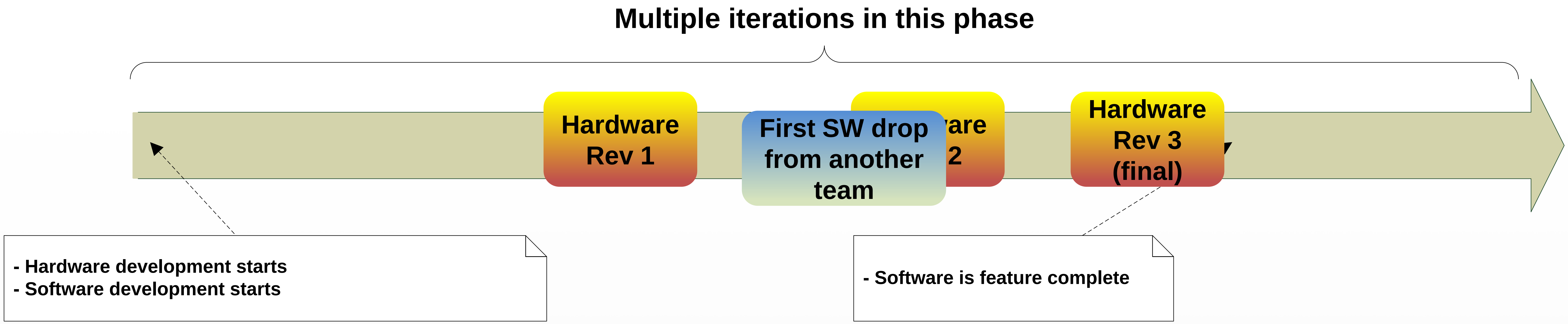
Hardware Rev 2

Hardware Rev 3 (final)

- Hardware development starts
- Software development starts

- Software is feature complete

Project lifecycle



Multiple iterations in this phase

- Hardware development starts
- Software development starts

- Software is feature complete

Dependencies

How to deal with dependencies?

Todays subjects

- Hardware & software abstraction layers.
- Using simulators under development and in unit test.
- Some potential problems when the hardware becomes available.
- Interface design between components.

How to deal with dependencies?

My objectives

- I want to run my software on another platform (both hardware and operating system).
- This very often means running the software on a PC.

- I need to create stubs for the components and hardware which is not yet available.

- Some of the ”stubs” probably need to be simulators rather than stubs.

Abstractions

Multiple views

You need multiple views of the architecture

- Active processes/tasks
- Interrupts

- Methods calls
- Message passing

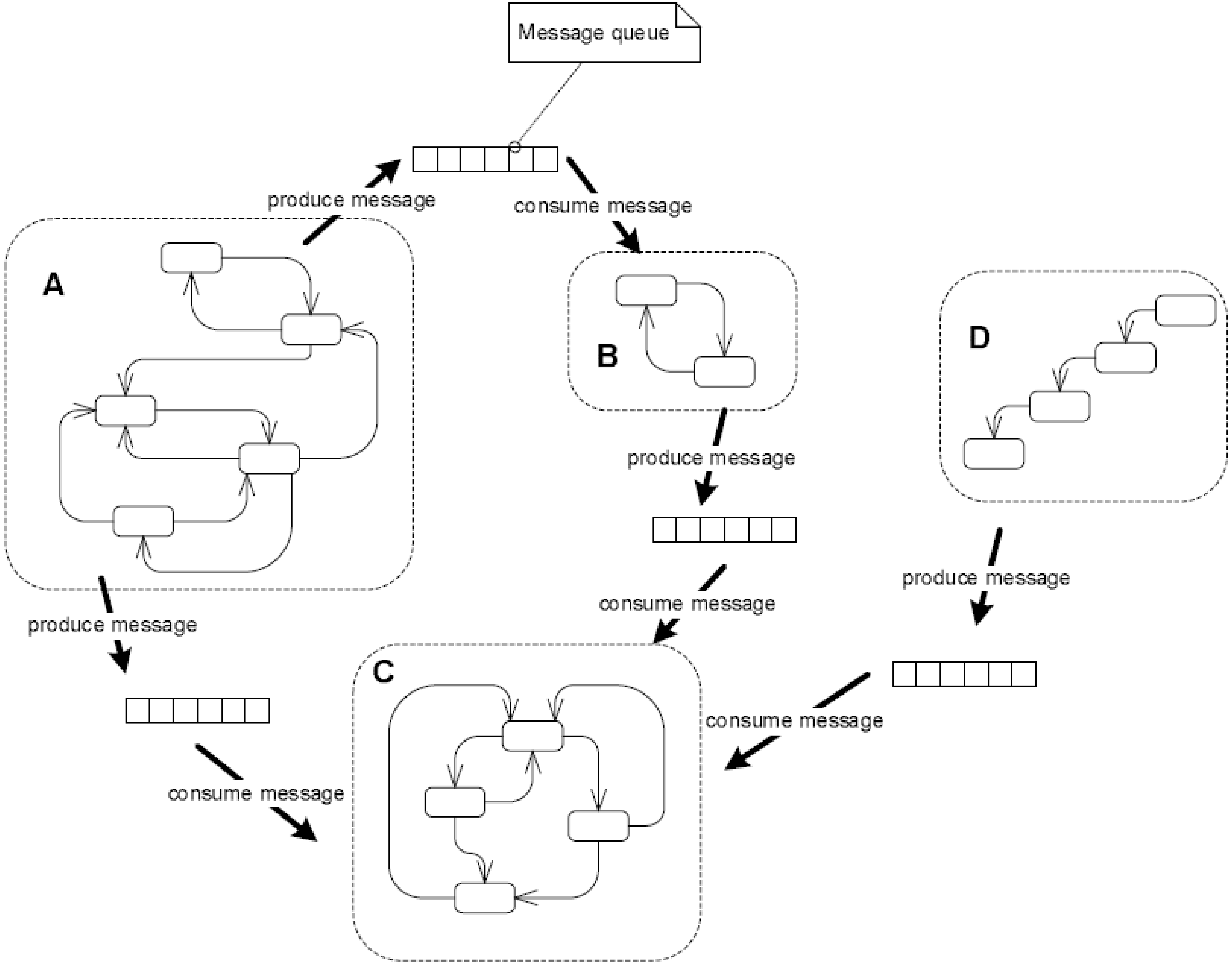
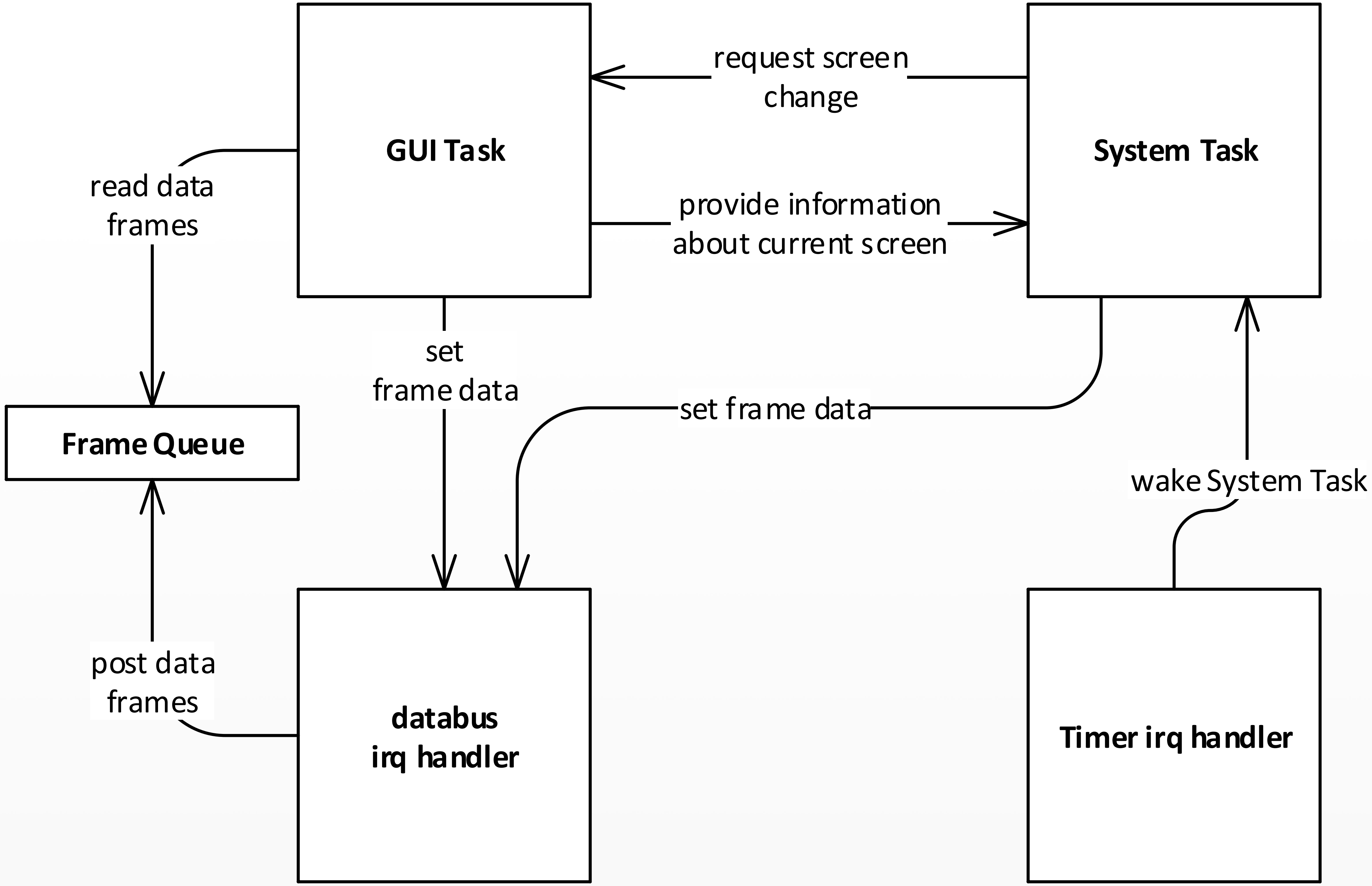
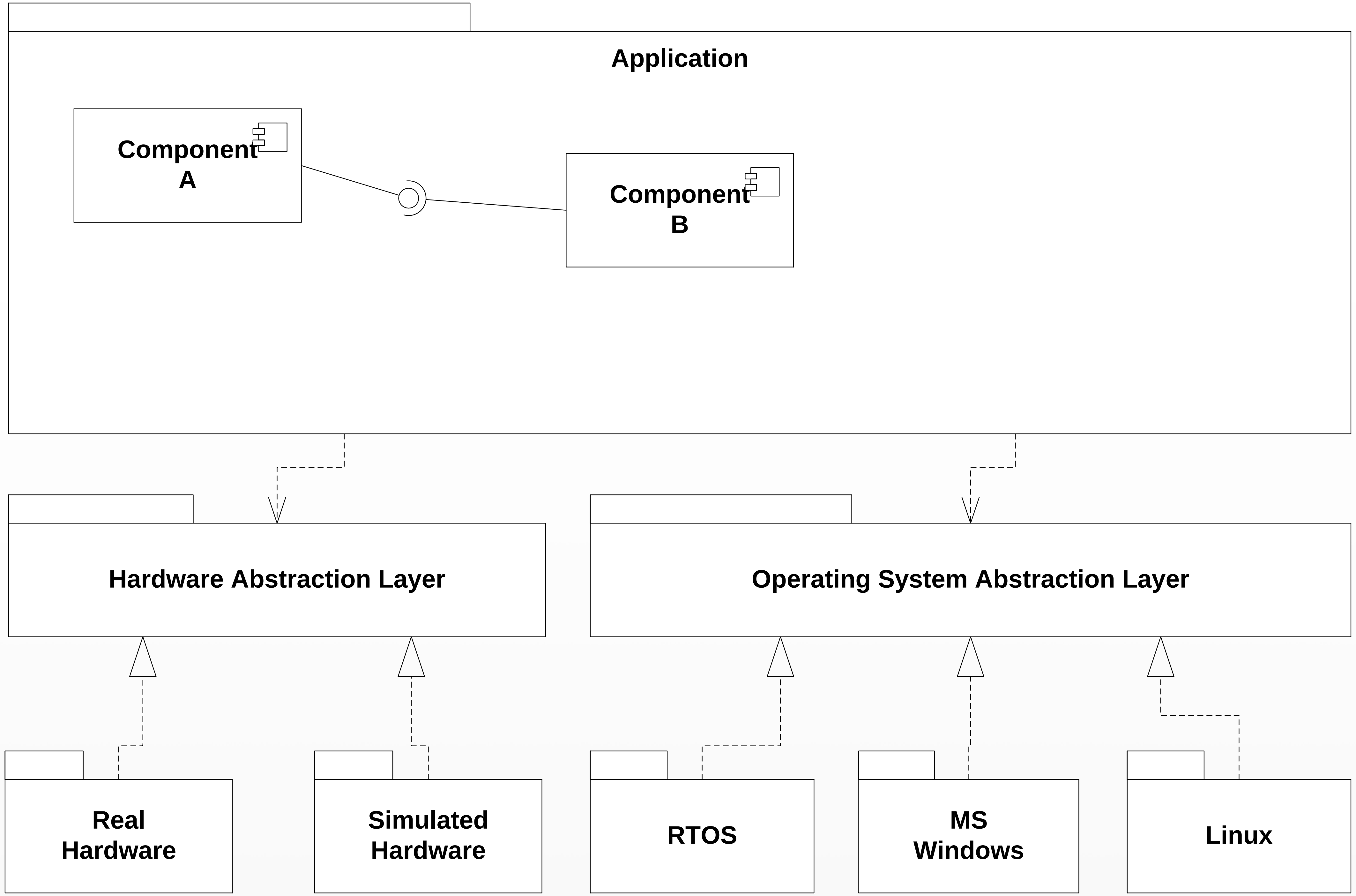


Figure 3.3: Model of computation

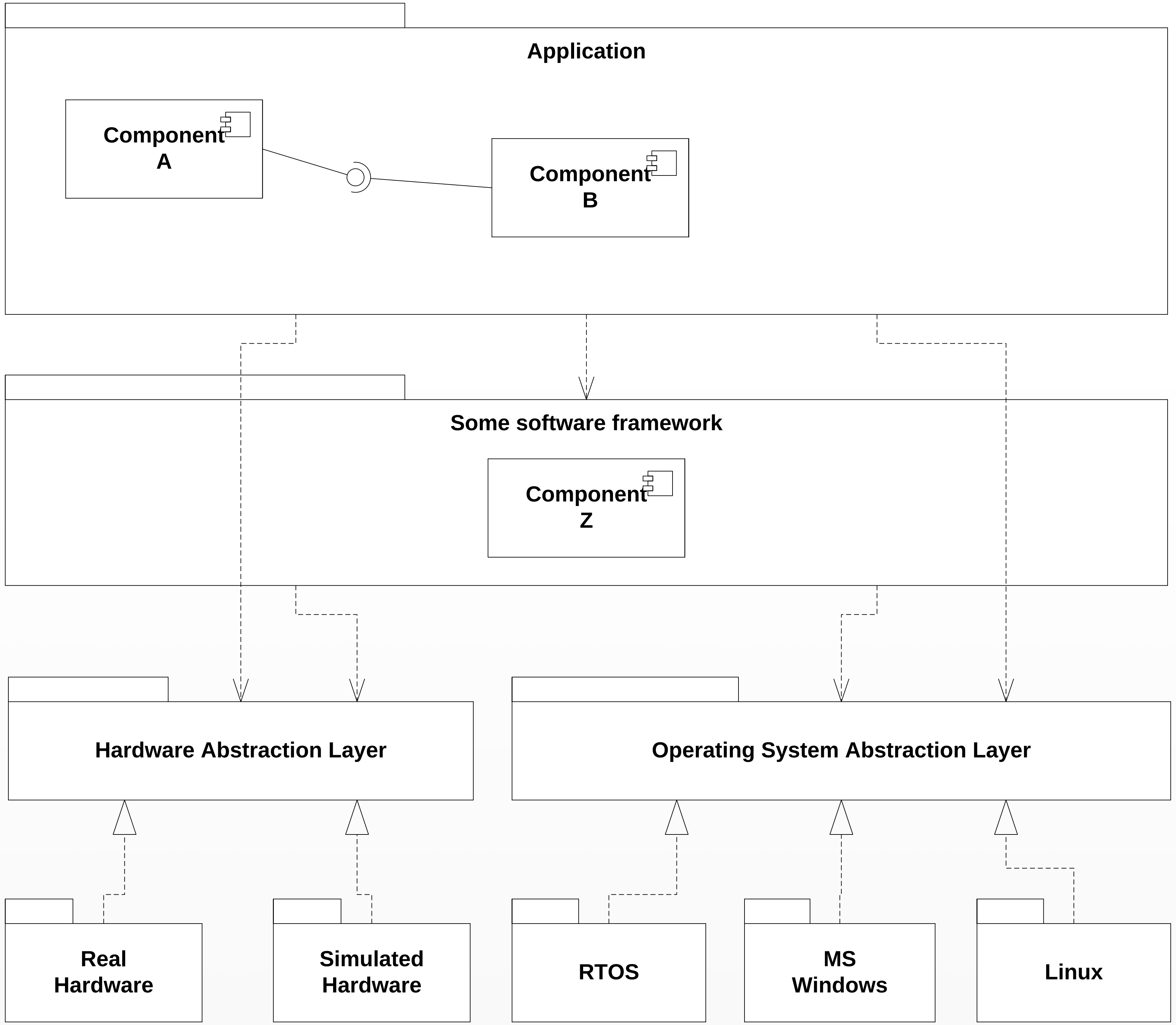
Create a process view



Abstractions and a layered architecture



Layered architecture



Abstractions

Whats in an OSAL?

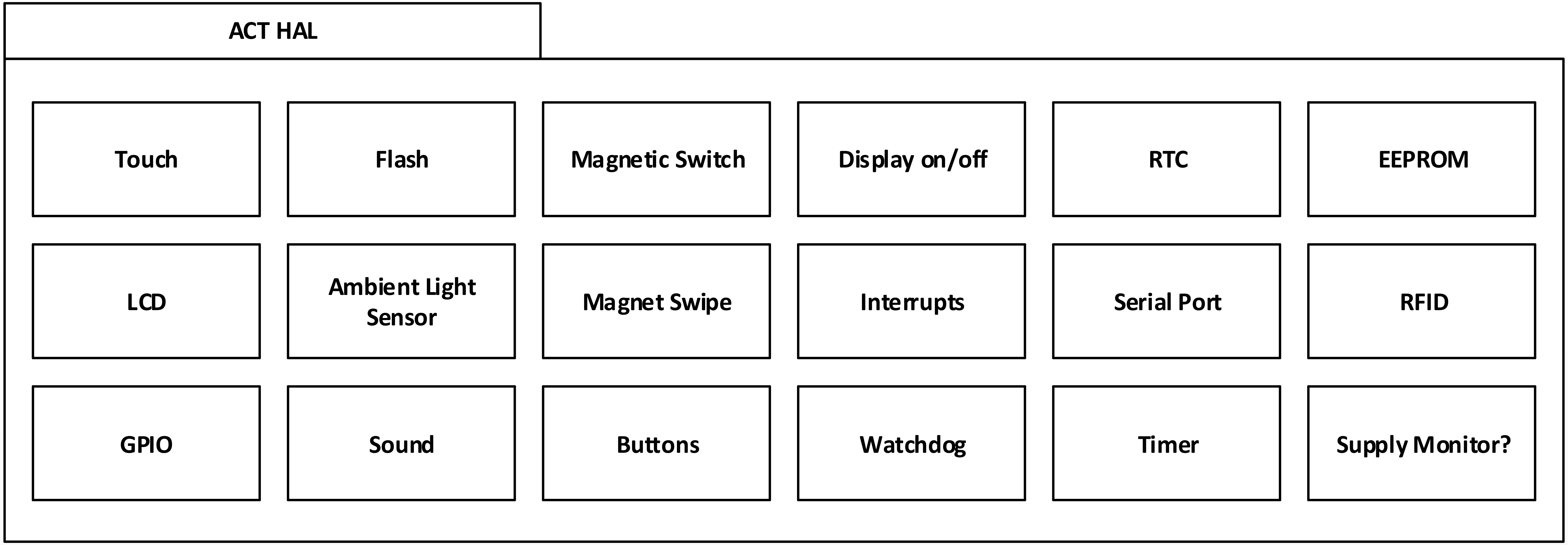
- Process/thread/task
- Mutex, semaphore
- Message queues
- File access
- Networking, sockets, TCP/IP, UDP
- Serial ports
- Time, Timers

Abstractions

Whats in a HAL?

- Touch input, Keypress, Display
- Outputs: switches, PWM signals
- Analog/Digital converters
- Real time clock
- Sensors – IR, proximity, light, RFID, hall
- Flash disk
- EEPROM
- Memory mapped I/O
- Interrupts
- …

Example from a project



ACT HAL

Touch

Flash

Magnetic Switch

Display on/off

RTC

EEPROM

LCD

Ambient Light Sensor

Magnet Swipe

Interrupts

Serial Port

RFID

GPIO

Sound

Buttons

Watchdog

Timer

Supply Monitor?

What shall my abstraction be?

How to define the HAL?

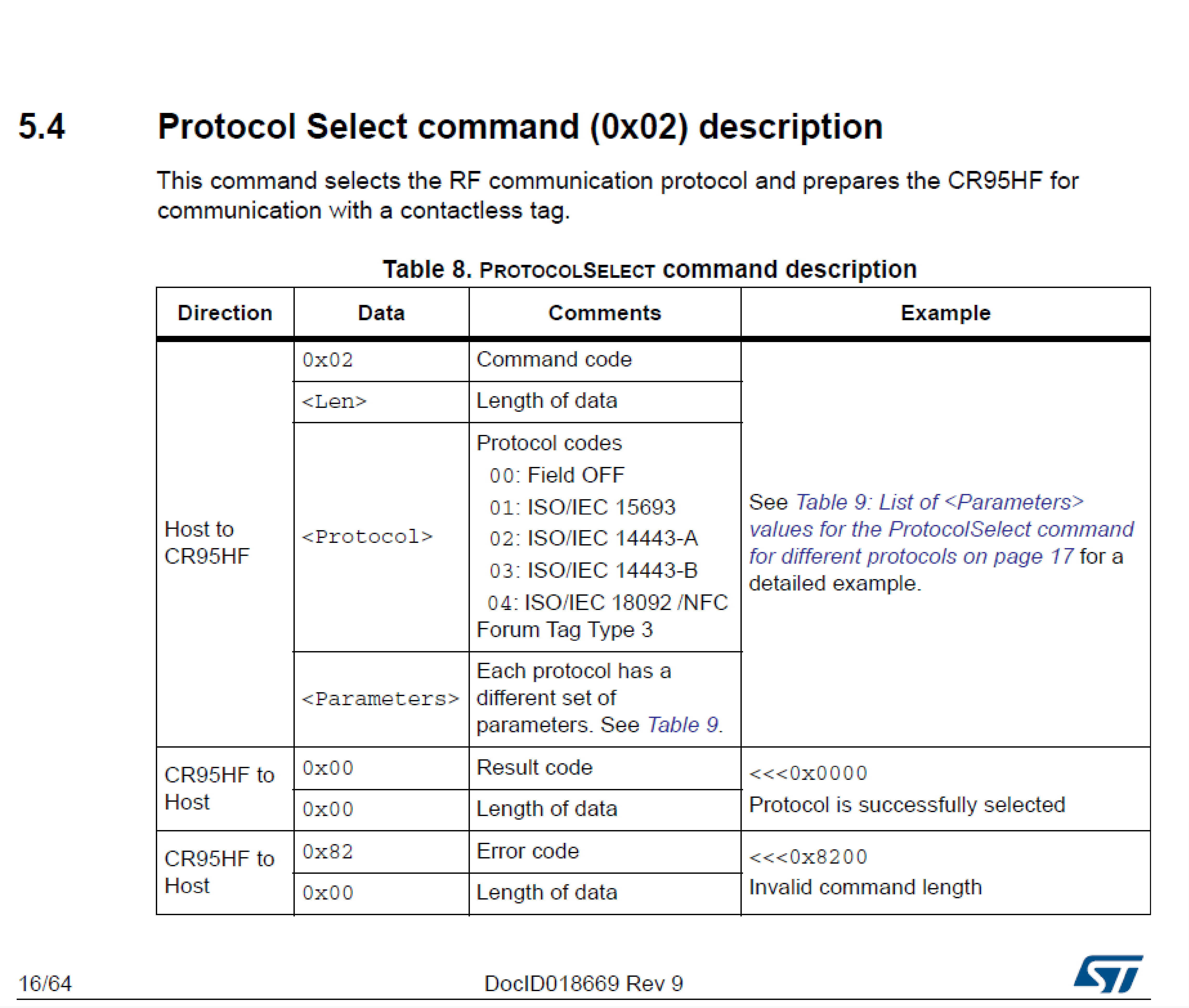
- Look at the functional requirements, e.g. ”Using an RFID token, the service technician can access the configuration menu”.

- How do I read an RFID token?
- What shall I read from the token?

How to define the HAL?

- Look at the hardware diagrams.

- The hardware designer has chosen an RFID chip.
- Get the data sheet and other manuals to learn what the chip can do.



5.4 Protocol Select command (0x02) description

This command selects the RF communication protocol and prepares the CR95HF for communication with a contactless tag.

Table 8. ProtocolSelect command description

Direction	Data	Comments	Example
Host to CR95HF	`0x02`	Command code	See *Table 9: List of <Parameters> values for the ProtocolSelect command for different protocols on page 17* for a detailed example.
	`<Len>`	Length of data	
	`<Protocol>`	Protocol codes `00`: Field OFF `01`: ISO/IEC 15693 `02`: ISO/IEC 14443-A `03`: ISO/IEC 14443-B `04`: ISO/IEC 18092 /NFC Forum Tag Type 3	
	`<Parameters>`	Each protocol has a different set of parameters. See *Table 9*.	
CR95HF to Host	`0x00`	Result code	`<<<0x0000` Protocol is successfully selected
	`0x00`	Length of data	
CR95HF to Host	`0x82`	Error code	`<<<0x8200` Invalid command length
	`0x00`	Length of data	

Interface granularity and dynamics

- Polled / Callbacks
- Blocking / Non-blocking
- How to report ”no data” / ”invalid data”?

Interfaces between software components

Interface specification

Considerations

- Specify both the signatures of the interfaces
  - method signatures
  - message payload
  - boundaries
  - data types
- And the dynamics of how the interface is intended to be used.

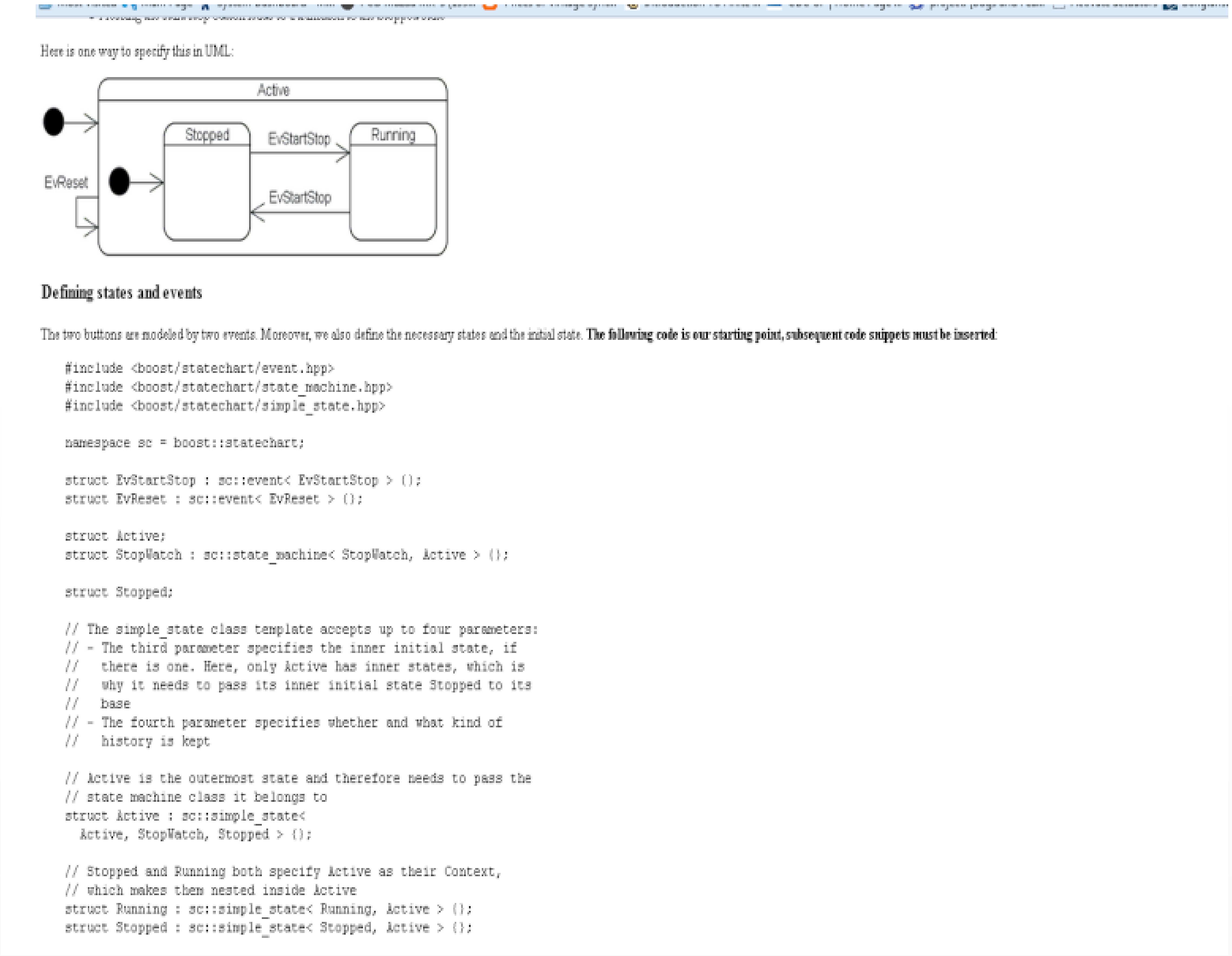
Interface dynamics

What is the intented use of the interface?

- Is there a specific order of method calls?
- Will my call block? For how long?
- In which thread context does the method call run?
- If I provide a callback, when will it be called? And how long can I expect to wait before it happens?
- In which thread context does the callback run?
- Which errors can happen?
- Are any errors critical?

Intended use

- Example from boost.org:
- http://www.boost.org/doc/libs/1_55_0/libs/statechart/doc/tutorial.html



Here is one way to specify this in UML:

Defining states and events

The two buttons are modeled by two events. Moreover, we also define the necessary states and the initial state. **The following code is our starting point, subsequent code snippets must be inserted:**

```
#include <boost/statechart/event.hpp>
#include <boost/statechart/state_machine.hpp>
#include <boost/statechart/simple_state.hpp>

namespace sc = boost::statechart;

struct EvStartStop : sc::event< EvStartStop > {};
struct EvReset : sc::event< EvReset > {};

struct Active;
struct StopWatch : sc::state_machine< StopWatch, Active > {};

struct Stopped;

// The simple_state class template accepts up to four parameters:
// - The third parameter specifies the inner initial state, if
//   there is one. Here, only Active has inner states, which is
//   why it needs to pass its inner initial state Stopped to its
//   base
// - The fourth parameter specifies whether and what kind of
//   history is kept

// Active is the outermost state and therefore needs to pass the
// state machine class it belongs to
struct Active : sc::simple_state<
  Active, StopWatch, Stopped > {};

// Stopped and Running both specify Active as their Context,
// which makes them nested inside Active
struct Running : sc::simple_state< Running, Active > {};
struct Stopped : sc::simple_state< Stopped, Active > {};
```

Interface specification

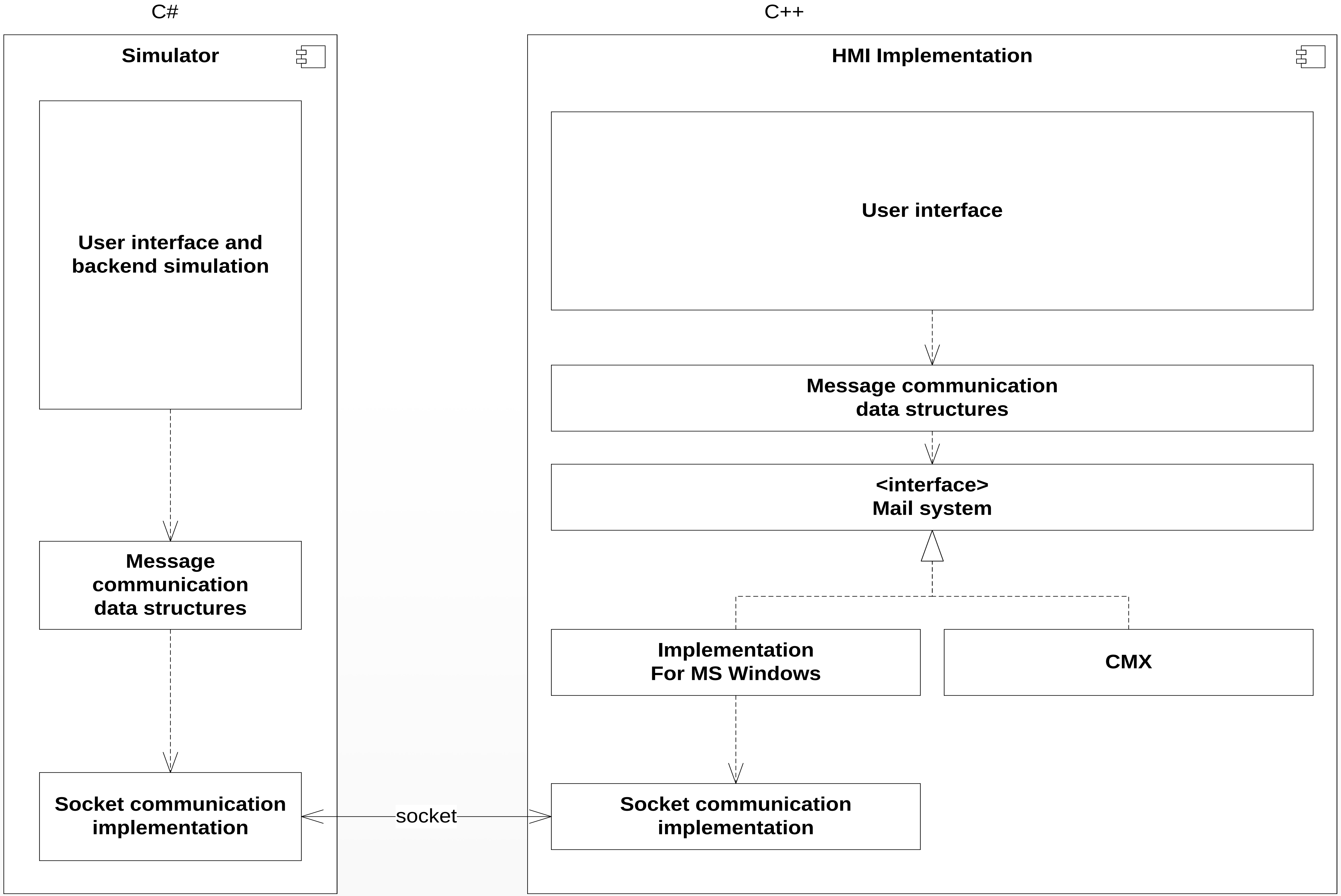
Considerations

- Interfaces shall be self contained
- And only depend on common data definitions and type definitions

Simulation

Simulation

- Stubs + dependency injection
- Stimulus functions
- Simulation of other components
- Simulation of external hardware (both internal and external to the application software)


C#
C++
Simulator
HMI Implementation
User interface and backend simulation
Message communication data structures
Socket communication implementation
socket
User interface
Message communication data structures
<interface> Mail system
Implementation For MS Windows
CMX
Socket communication implementation

Surprise!

Behavior

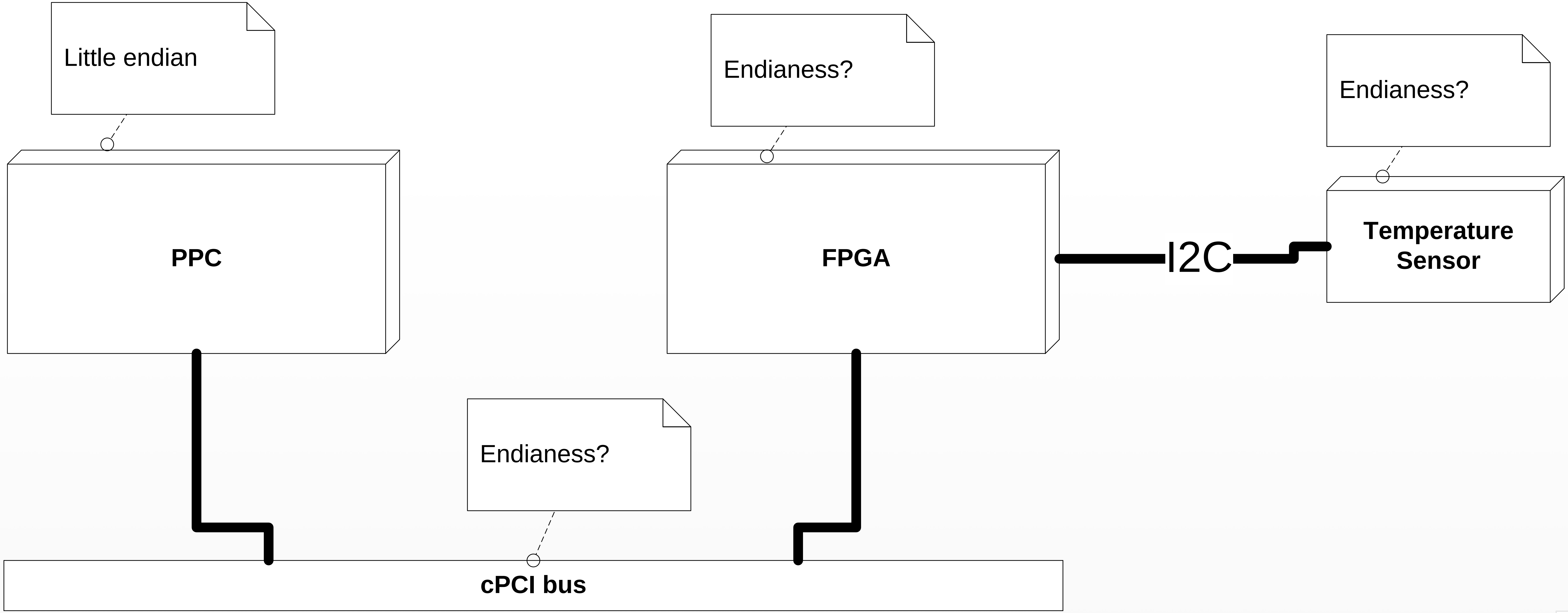
- I wrote both my software and the simulator… the real thing often behaves differently.
- Timing is not as expected.
- Code executes in the wrong context.
- Semaphore/Mutex deadlocks.

Endianess

Endianess

- Different endianess on different CPU architectures.
- Intel x86 is little endian.
- Some ARM and PPC CPUs can run both in little and big endian.

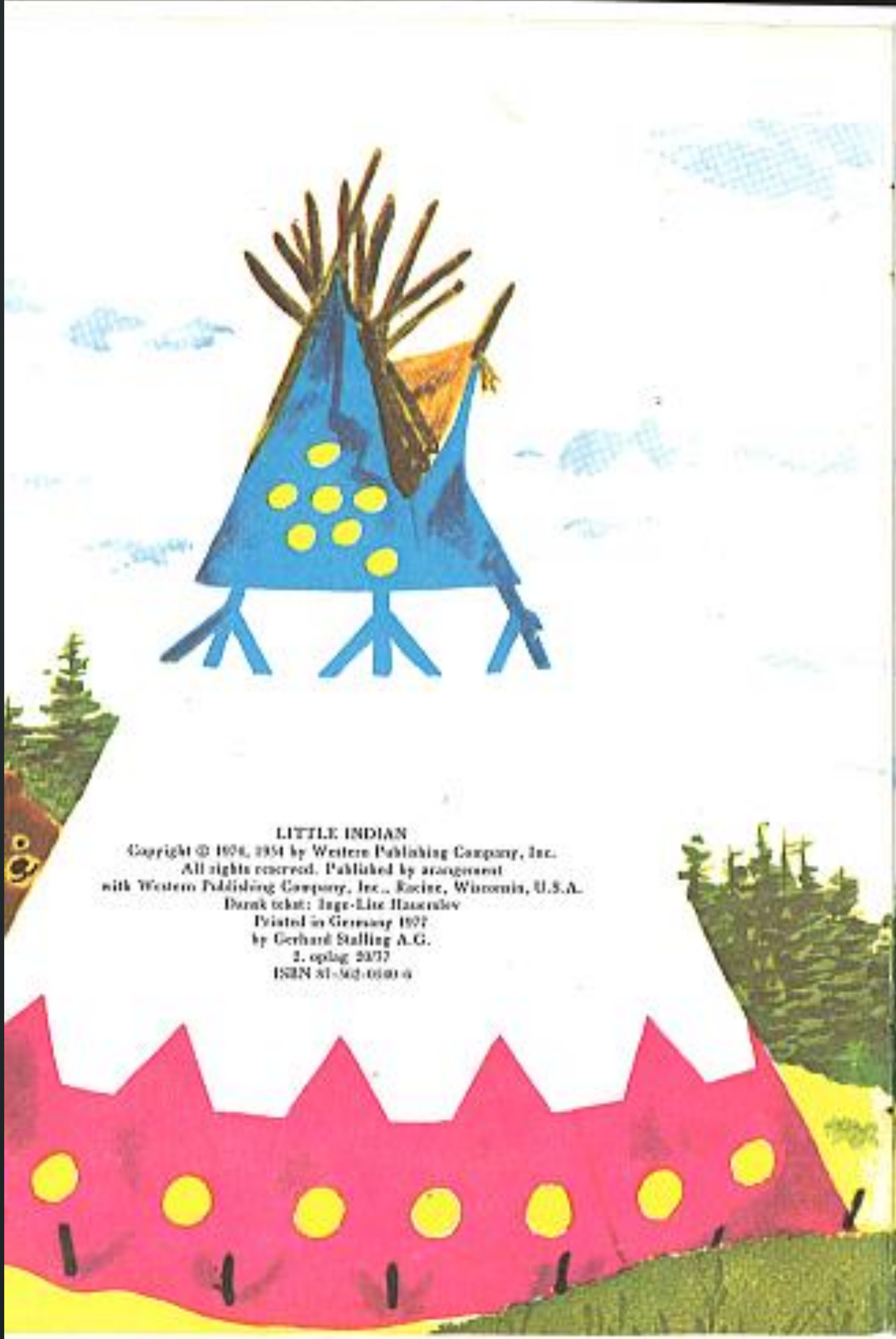
Reading a 32 bit value from a sensor



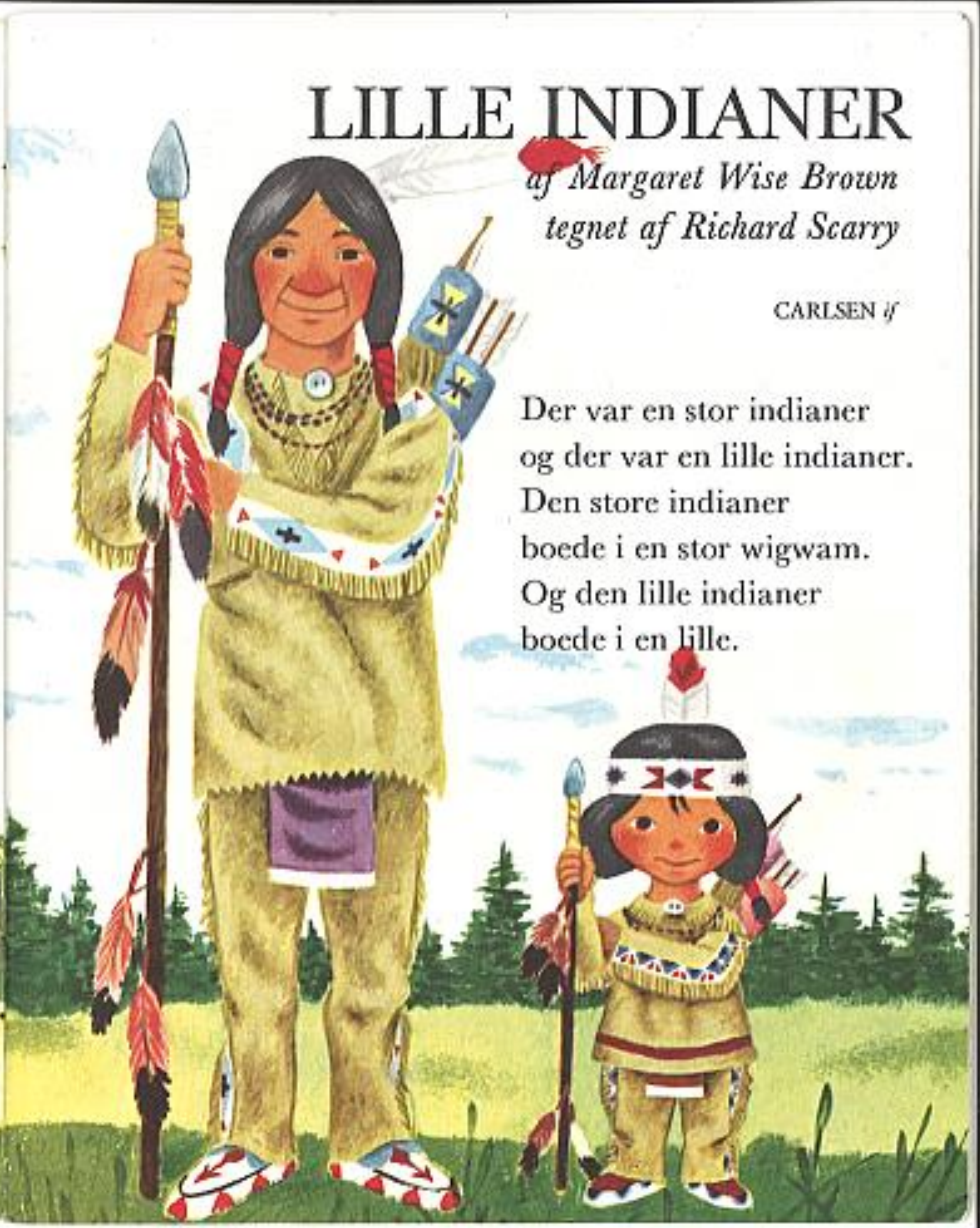
Endianess

- I always struggle with this!
- Do the endian conversions at the boundaries of your software:
  - In the drivers or the network layer: hton() ntoh()

- Test and test again



LITTLE INDIAN
Copyright © 1974, 1954 by Western Publishing Company, Inc.
All rights reserved. Published by arrangement
with Western Publishing Company, Inc., Racine, Wisconsin, U.S.A.
Dansk tekst: Inge-Lise Hauerslev
Printed in Germany 1977
by Gerhard Stalling A.G.
2. oplag 20/77
ISBN 87-562-0580-6



LILLE INDIANER

af Margaret Wise Brown
tegnet af Richard Scarry

CARLSEN if

Der var en stor indianer
og der var en lille indianer.
Den store indianer
boede i en stor wigwam.
Og den lille indianer
boede i en lille.

The devil is in the detail

```
#include <stuffThisStuffNeeds.hpp>
```

- When you write C++, the #include statements in headers can result in inclusion of other headers . Obviously for this to compile, the compiler needs to be able to find all the headers.
- This is usually not a problem if you are building a monilithic application, but if I want differerent teams to build different parts of the application, I need to have an interface, which is self contained.
- I have seen interfaces to components, where when you followed the header definitions, you ended up looking at the type definitions for the RTC at the lowest level of that hardware layer.

- Forward declarations are a way to avoid including other header files.

Integers

Integers

- Integer sizes in bytes are not defined in C++
- But you need to know the size of your data when you develop embeded SW.
- Companies and often also projects, tend to define their own datatypes. This causes troubles if you want to reuse code from a project or include a 3rd party library.
- One problem is that e.g. a WORD is processor specific, so when you change from a 16 bit processor to 32 bits, a WORD is really 32 bits, but by the old definition it is 16 bits. Even Microsoft is guilty of this.

- Need size because: Bitmanipulations, shifting operators, memory mapped io, casting structs on to memory mapped io.

Integers

- Integer sizes in bytes are not defined in C++
- But you need to know the size of your data when you develop embeded SW.
- Every company have their own integer definitions:
  - Uint8, uInt8_t, uint8, short, SHORT, WORD, …. Etc..

Integers

- Integer sizes in bytes are not defined in C++
- But you need to know the size of your data when you develop embeded SW.
- Every company have their own integer definitions:
  - Uint8, uInt8_t, uint8, short, SHORT, WORD, …. Etc..
  - PLEASE try not to do this!

stdint.h

Available on all modern compilers!

```
/* 7.18.1.1  Exact-width integer types */
typedef signed char int8_t;
typedef unsigned char   uint8_t;
typedef short  int16_t;
typedef unsigned short  uint16_t;
typedef int  int32_t;
typedef unsigned   uint32_t;
typedef long long  int64_t;
typedef unsigned long long   uint64_t;
```

Mjølner INFORMATICS

February 2014

Michael Sørensen Loft

Senior Software Architect

@ mls@mjolner.dk ☎ + 45 xx xx xx xx