Test & Measurement Solutions

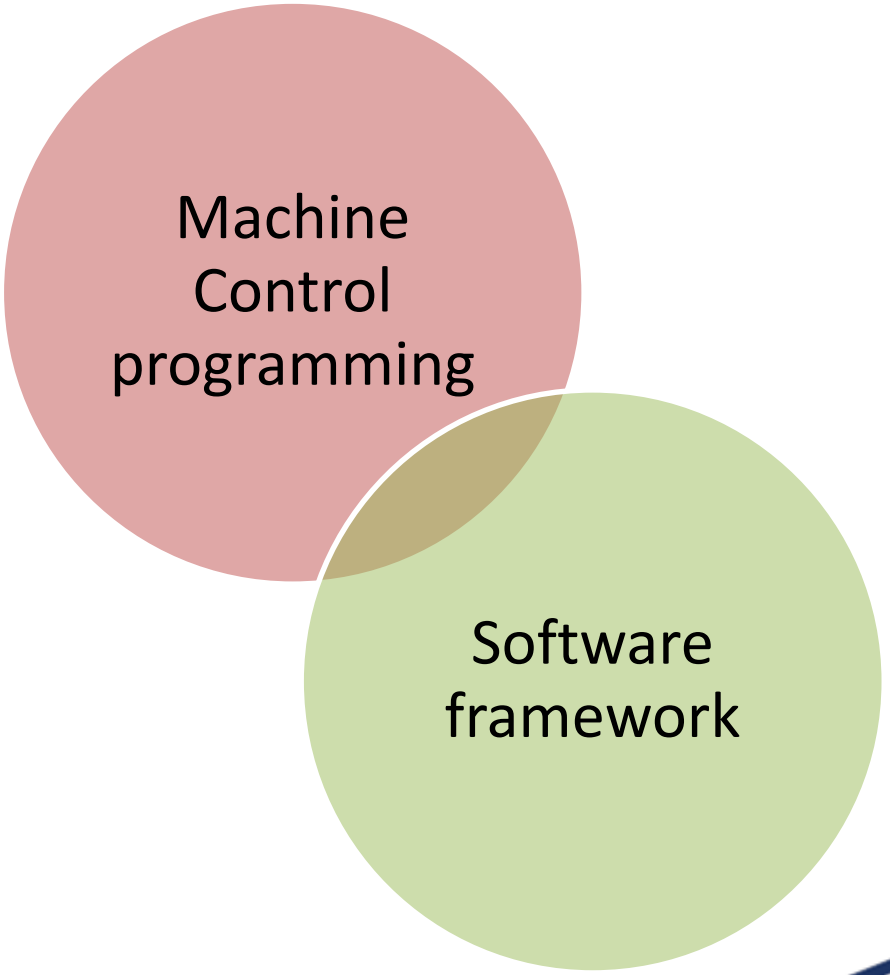


# Quality Solutions for Processes and Products

Presenter: Arnoud de Kuijper, Managing director NL
Author: Jan Bosmans, Software Team Leader

# Machine control framework?

- 2 separate parts in the presentation title:



# And that is why we need…

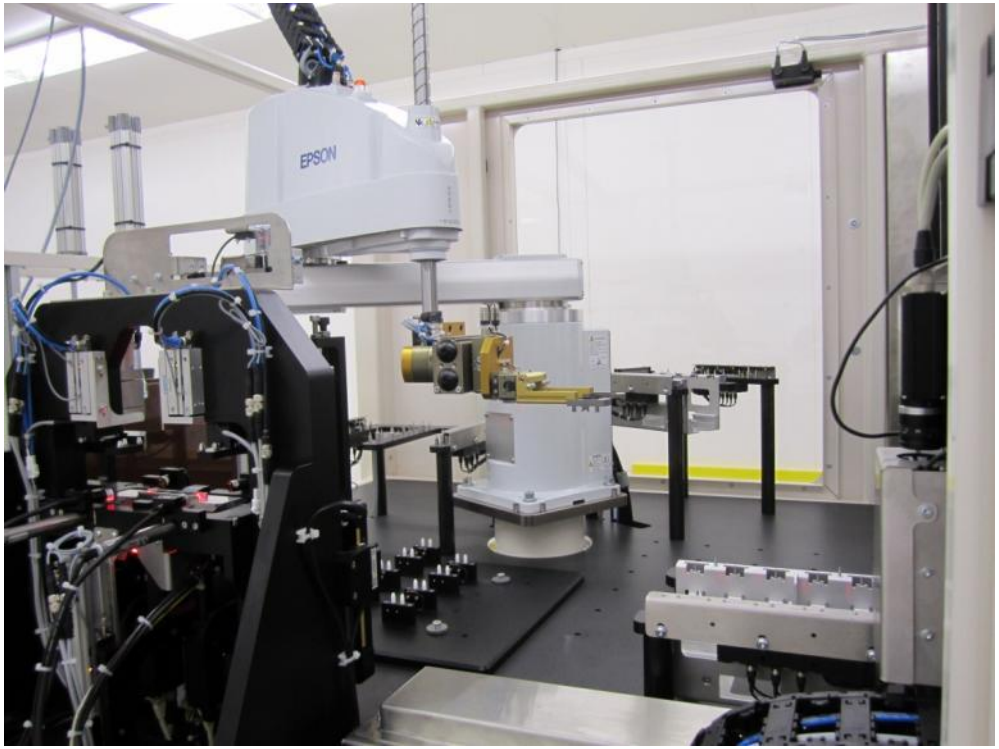
1. Machine control characteristics

# What do we mean by machine control?

- Machine control is the combination of a software application and a hardware platform that **autonomous controls** equipment.

# Fully automated machine

- **Fully automated**, these machine run continuously a repetitive sequence of tasks for a specific time (example: machines in a production line)



# Manual machine

- **Manual** machine that execute a sequence of tasks one time and must be initiated manually each time (example: electrical testers, prototype machines, …)

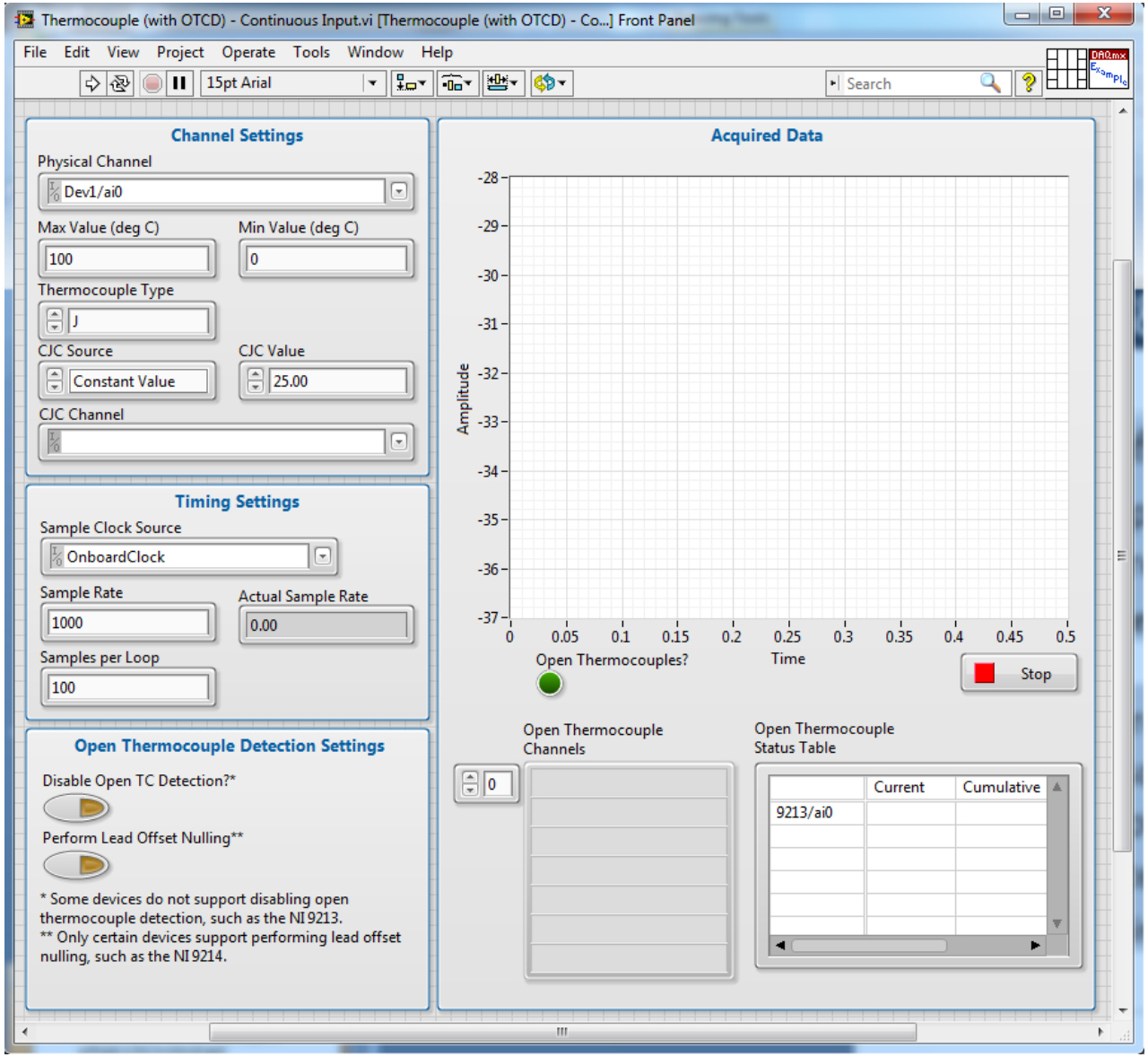
# Machine control with LabVIEW?

- Machine control used to be programmed with a PLC. There are advantages to move away from PLC:
  - Higher abstraction layer so more complex problems can be solved (Control algorithms, Vision inspection, Analysis, etc.)
  - More flexibility (more interfaces available)
  - Determinism at different clock speeds and platforms (FPGA, RT)
  - GUI and Visualization possibilities extended
  - Web-interfacing, DB interfacing more advanced.

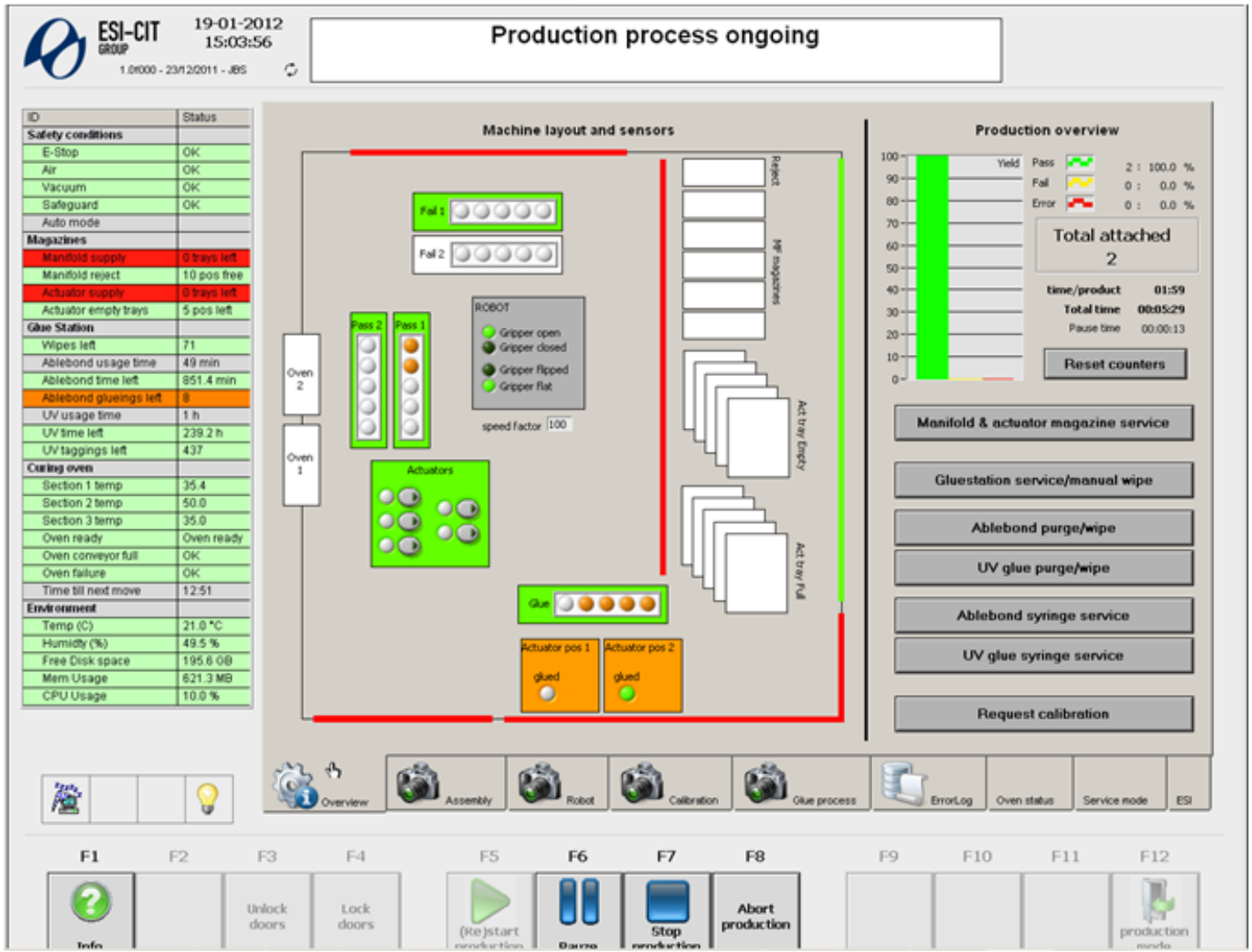
# Machine control software

- 2 parts in the software:
  - Actual process of the units under test/production (this part of software is the functional part)
    - *Interesting part for the end user*
  - Software to control the equipment itself (this part of the software is needed for the equipment but does not contribute to the end result of the items under test/production)
    - *Less interesting part for the end user*
    - *Often the largest part of the application*

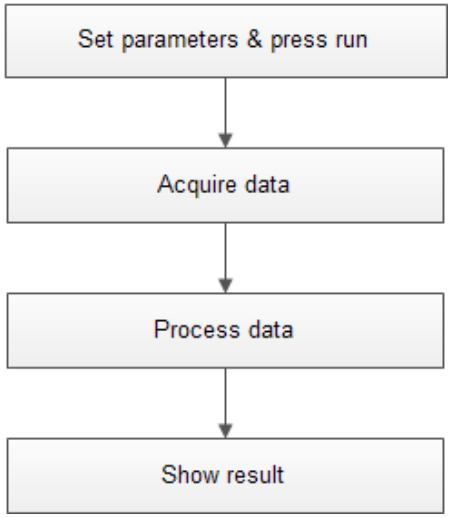
# 90% of labview applications



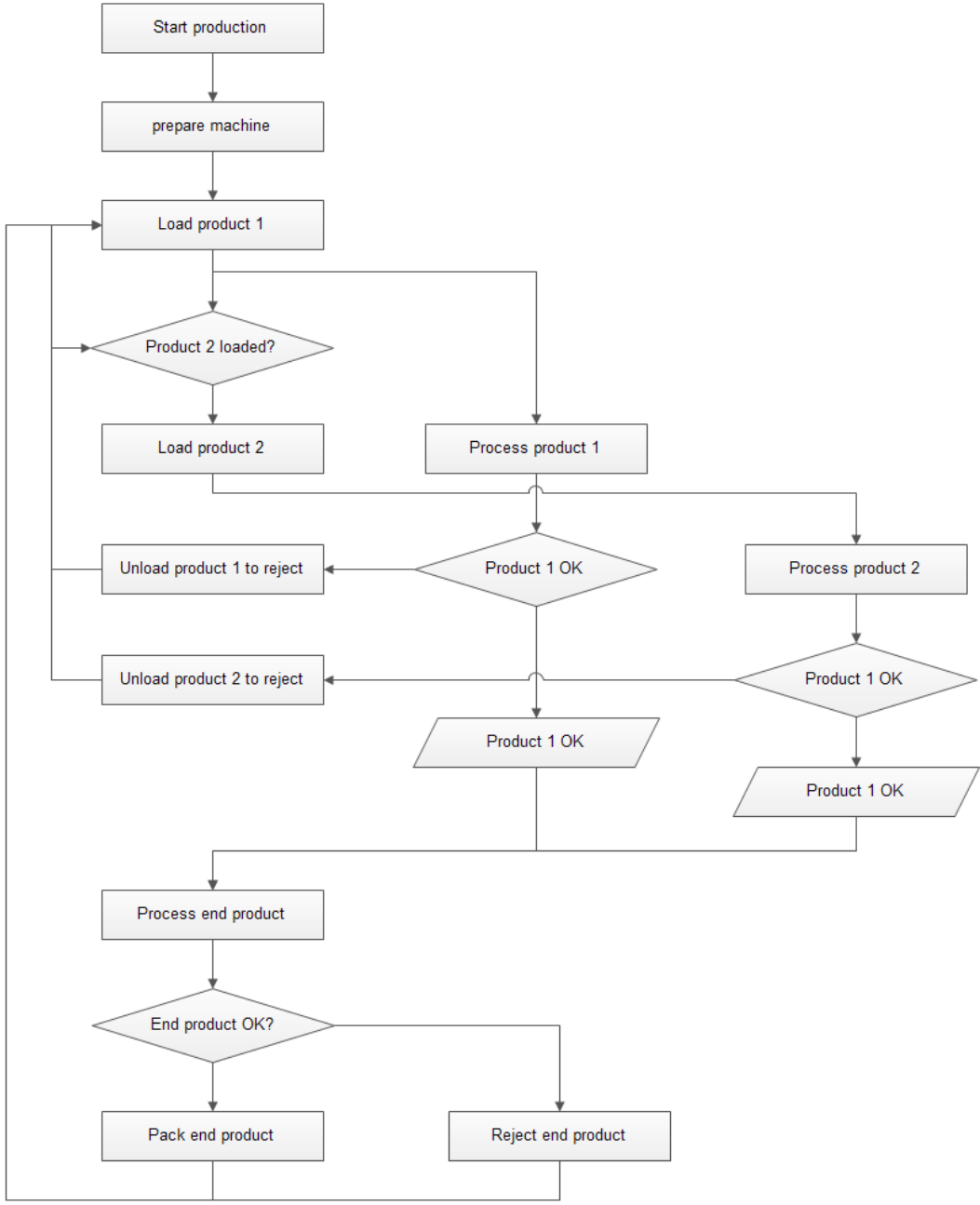
# 10% of labview applications



# What is the difference?



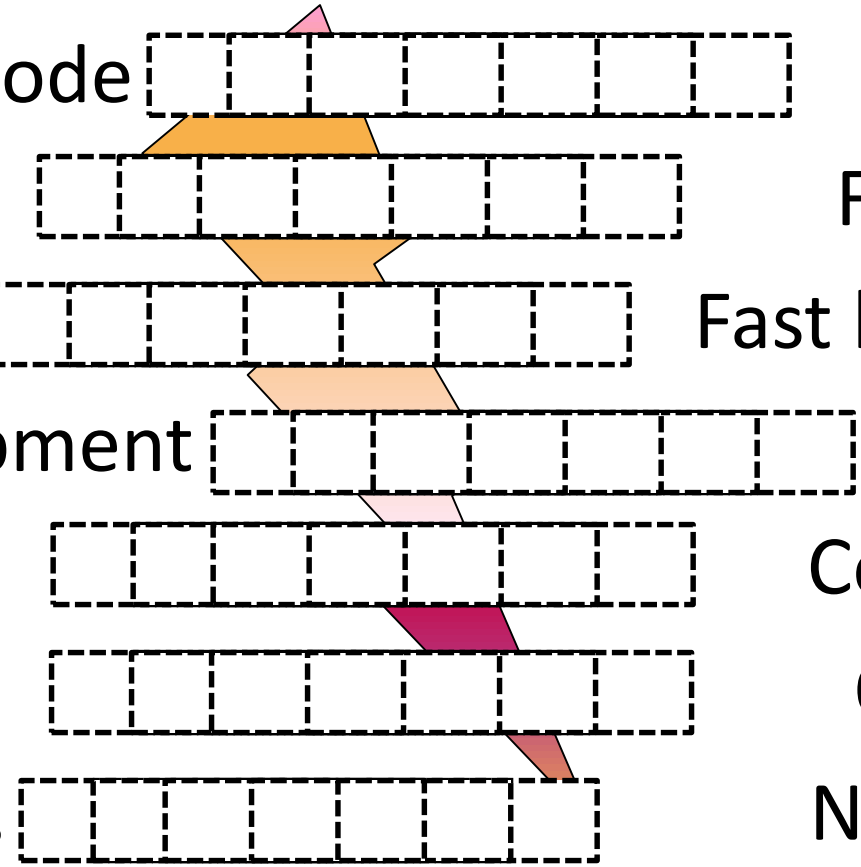

vs

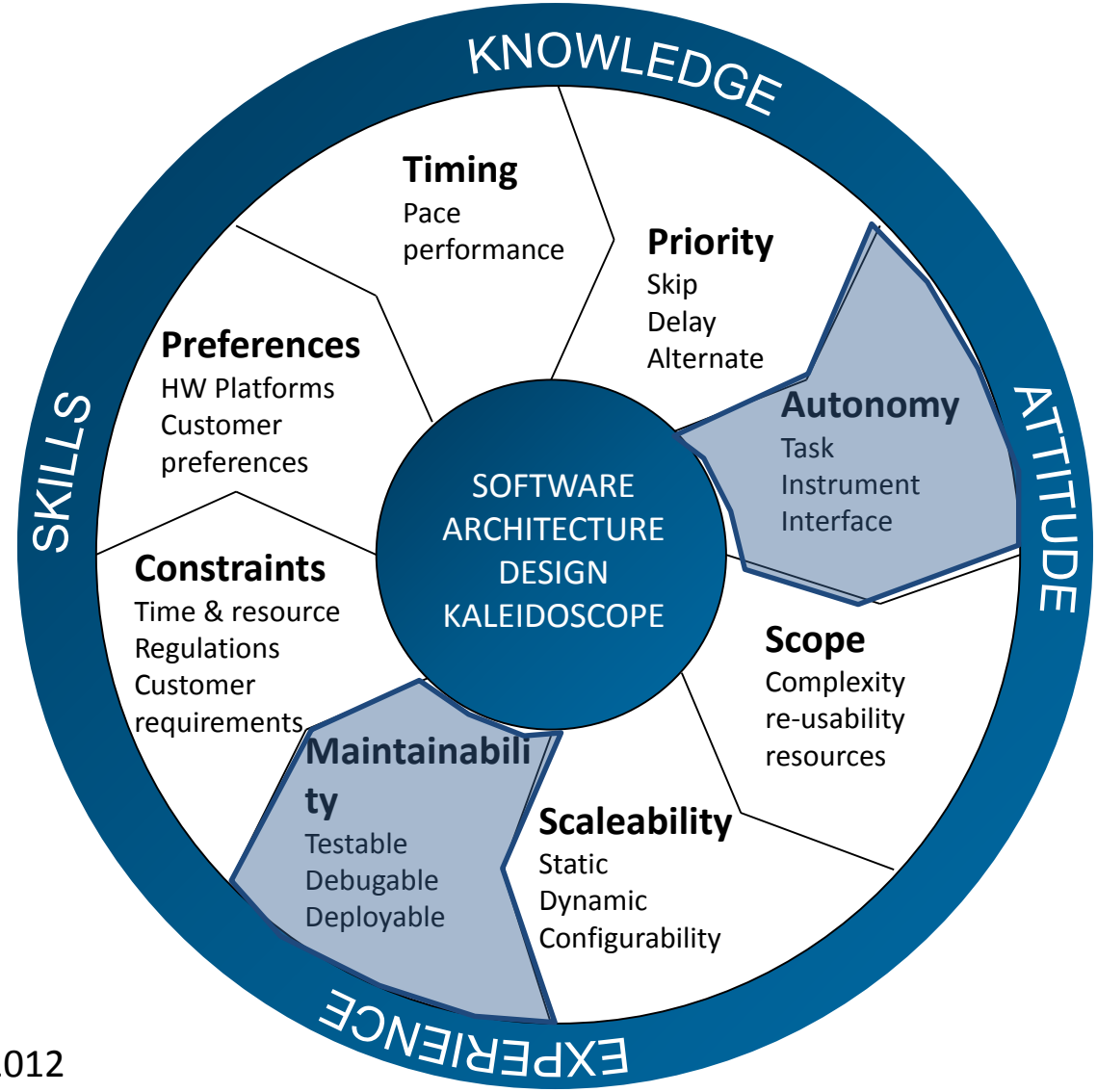


# And that is why we need…

## 2. Machine control FRAMEWORK

# Attitude – Balancing forces





# What is a framework?

- A **framework** is a collection of software components used to program an application.
- A framework consist of:
  - A “empty” base program to start with
  - Software modules *(placeholders)*
  - Libraries *(drivers, standard components,…)*
  - Code standards *(style)*
  - Rules on how to use the software modules and libraries

# Machine control framework

- A machine control framework is a framework that can be used as base for developing **machine control applications**. Key points of such a framework:
  - Appropriate error handling
  - Hardware abstraction
  - Simulation
  - Flexibility
  - Dynamic start/stop of parts of code
  - Step mode
  - Event logging

# Benefits of a framework

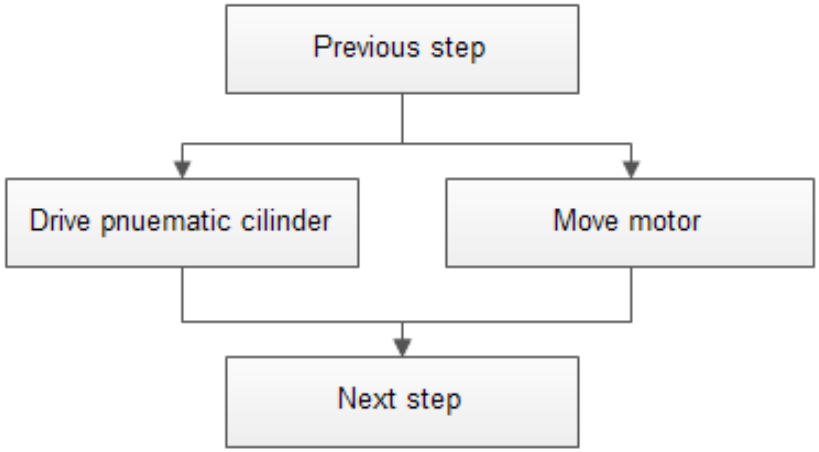
- The main task of the software developer is translating the functional descriptions of the end user into source code
- Test application offline for a major part of the source code
- Easy to work on with more developers
- Same look and feel of applications developed with framework

A few tips on developing an application framework

# DESIGN YOUR OWN FRAMEWORK

# 1. Sequential vs parallel tasks

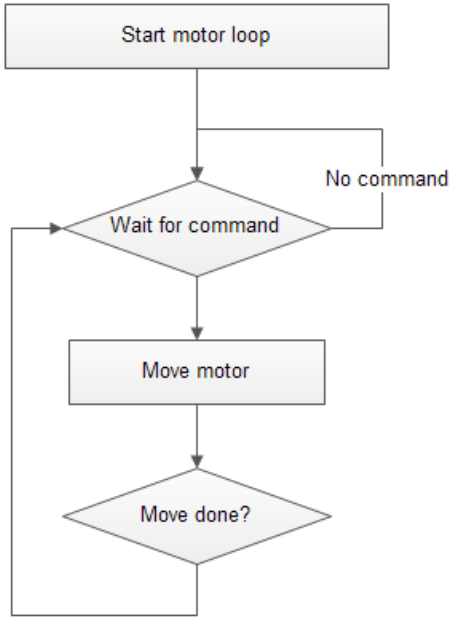
- STEP 1: Define and split up the main process from other processes
  - Do not use parallel process unless really necessary

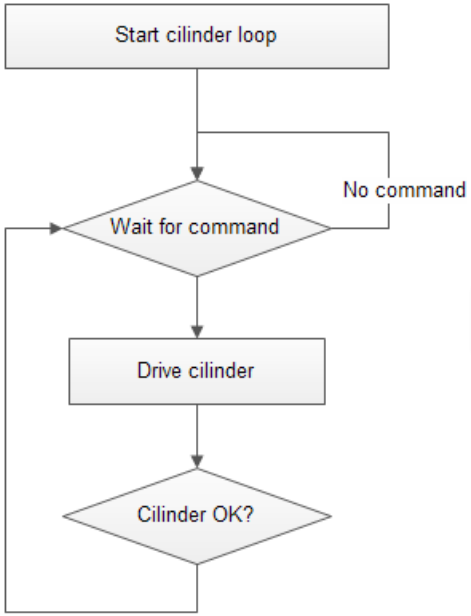



Parallel actions

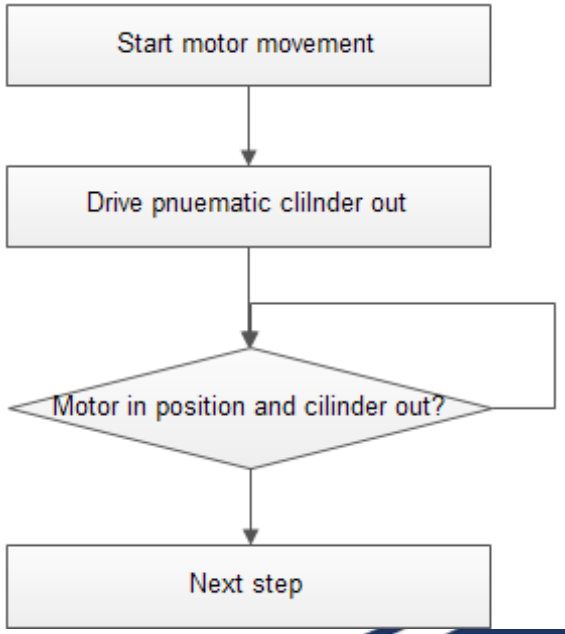
# 1. Sequential vs parallel tasks

- STEP 1: Define and split up the main process from other processes
  - Do not use parallel process unless really necessary



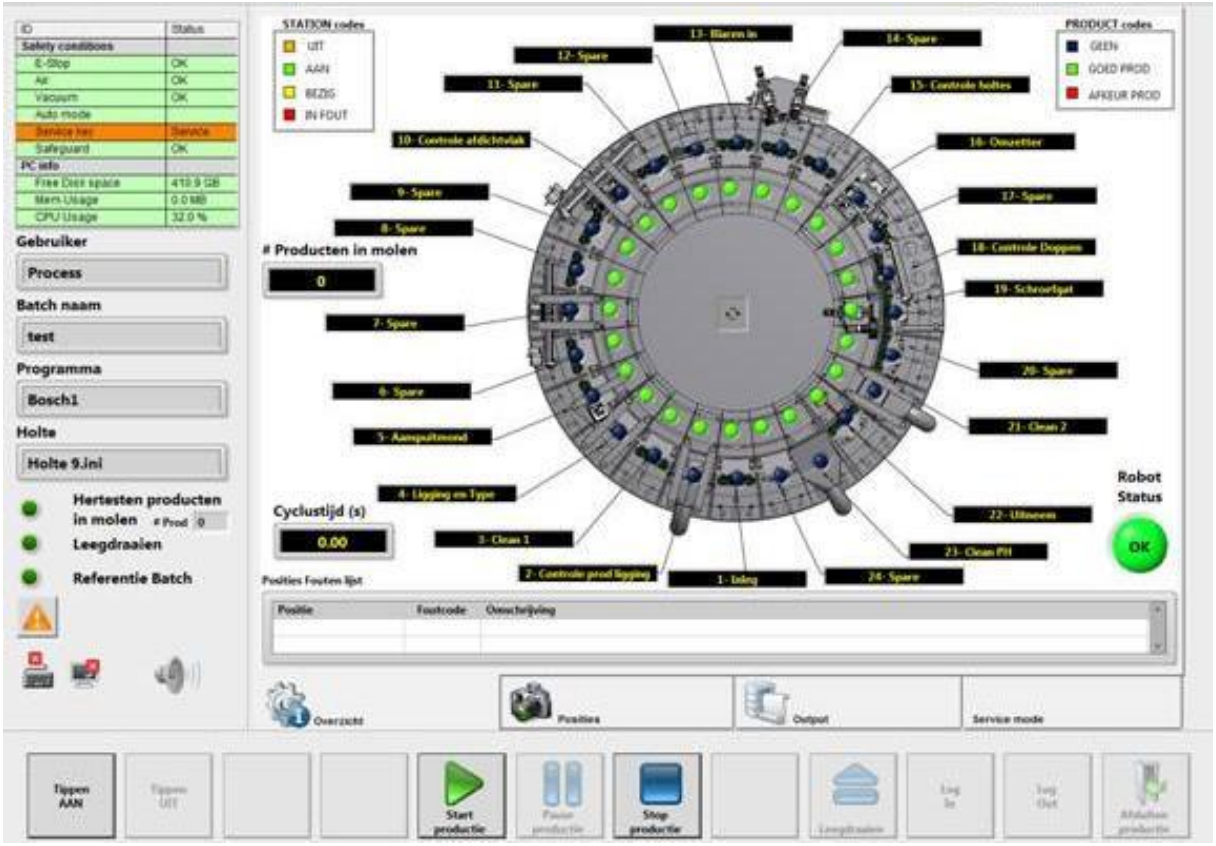

Parallel processes




sequential

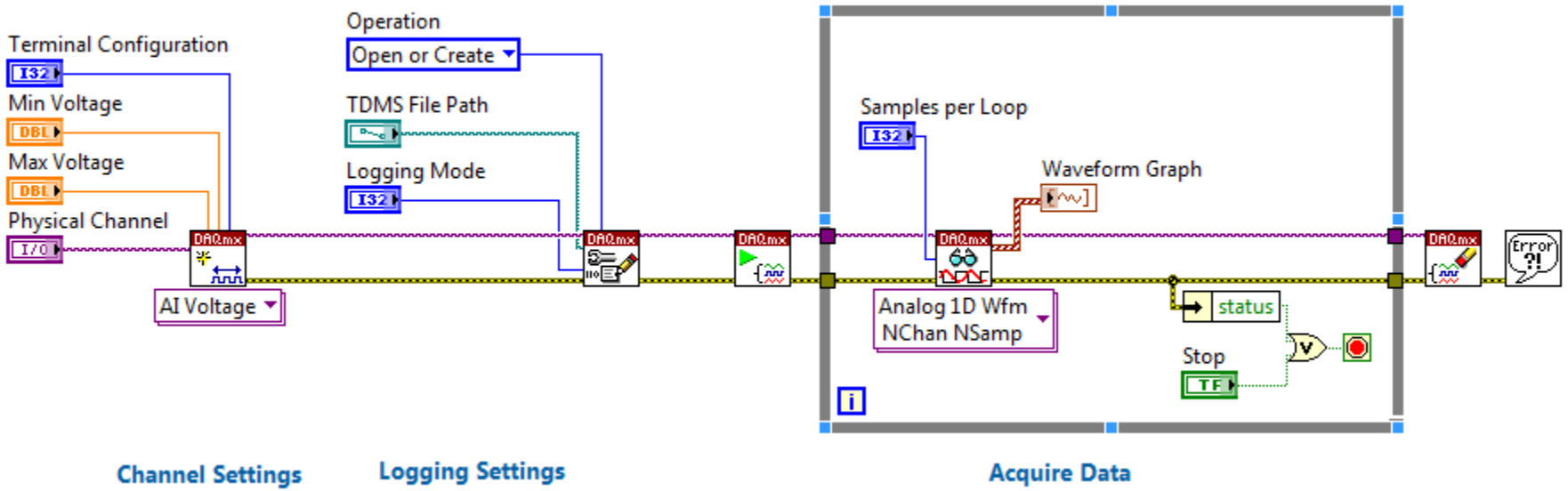
# 1. Sequential vs parallel tasks

– Example when parallel processes are needed:
  - Machine with multiple stations
  - Measurement and data processing are split up
  - Multiple PC's/platforms



# 2. Error handling

- Standard error handling – NOT SUFFICIENT!



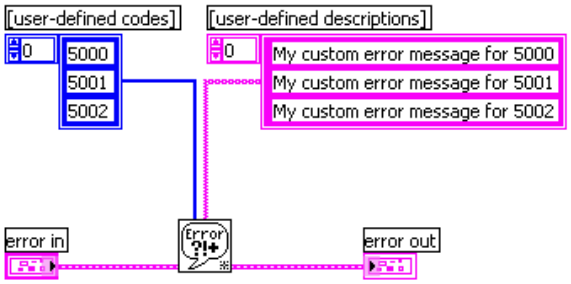
# 2. Error handling

- Categorize errors:
  - Product errors (eg validation of product not OK)
  - Machine errors
    - Recoverable errors (eg pneumatic cylinders read contacts not in expected position within a timeout)
      *When these errors occur, the machine is in a defined state and can continue after operator interaction*
    - Not recoverable errors (eg motion errors, emergency stop, …)
      *When these errors occur, parts of the machine are in an undefined state. Continuing is NOT possible*

# 2. Error handling

- Use user error codes and split up categories in different ranges from start of development

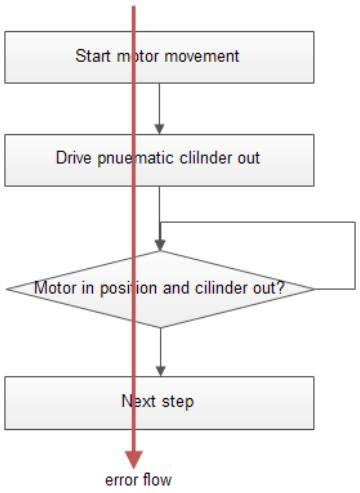
  User error code ranges are:
  5000 through 9999
  500.000 through 599.999



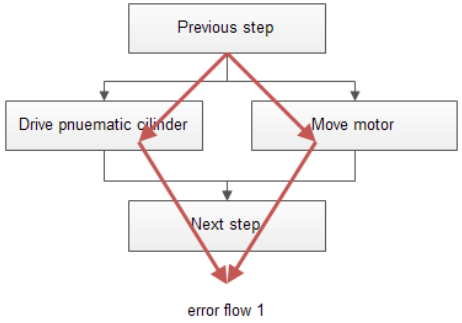

- Define user error codes in a text file instead of hard coding the error description.
  Complete this file systematically when creating new error codes

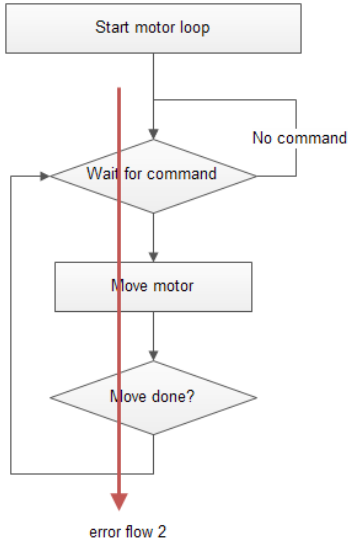
# 2. Error handling

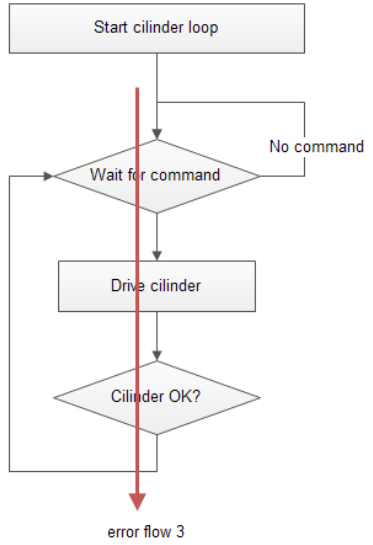
- Error handling in sequential vs parallel processes




vs

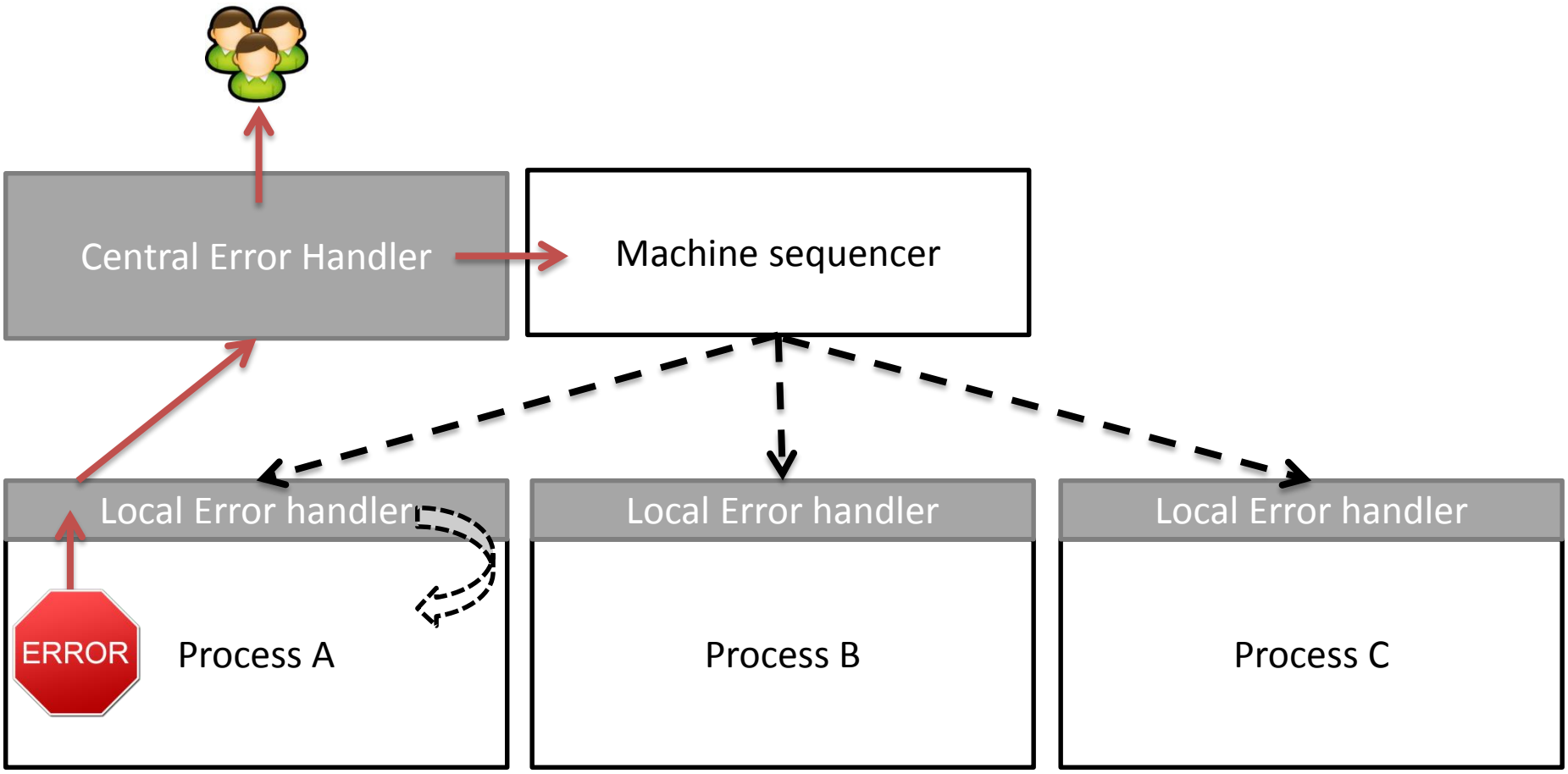



- *Parallel error handling is very complicated!!!*

# 2. Strategy for error handling



# 3. Dynamic calling of tasks

- Testing of parts of the application is crucial
- Example: a product is tested in a machine.
  - Mounting product = 2min
  - Conditioning product = 3min
  - Test product = 1min
  - Dismounting = 0.5min
    - Testing 1 time = 6.5min

    **To debug the “Test” without other steps**

    **saves 5.5min/trial run**

    - *This is possible when the test part is called dynamically*

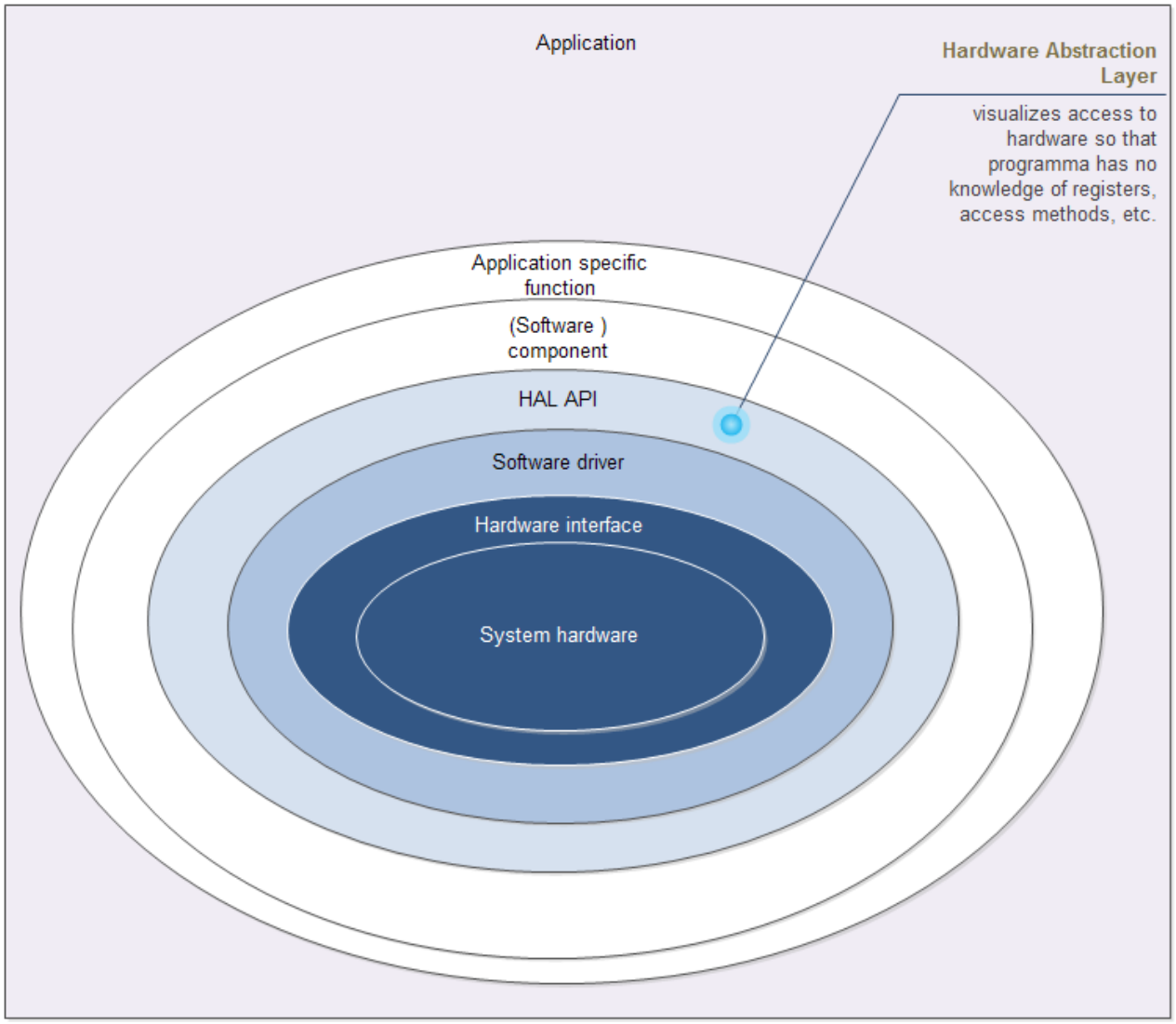
# 3. Develop good service interfaces

- A test/production machine is serviced by the end user
  - End user does not know Labview
  - End user does not know the “inside” of the machine
- Good service interfaces allow the user to
  - Test the machine from hardware point of view
    - Test motors, camera’s…
  - Test the machine from process point of view
    - Test product handling, the test process itself, …

# 4. Hardware abstraction

- Machines tend to stay active for a long time. Hardware used is sometimes obsolete and must be replace by other hardware
- Reusability of a hardware abstracted driver is much higher in future projects.
- Easier to add simulation when hardware drivers are abstracted

# 4. Hardware abstraction layer




## Typical high level functions

Initialize()
Send()
Recieve()
Close()

## Typical hardware specific functions in HAL

State conservation
Watchdog
Registers
value conversion
aggregation of functions
error catching
Hardware interface abstraction
device access

# EXAMPLE OF FRAMEWORK

# What makes T&M Solutions specialized

- Interactions with different aspects:
  - Electrical equipment
  - Mechanical equipment
  - Pneumatic/hydraulic equipment
  - …

- Software is only a part of a good test/production machine

- Development of a machine is a close cooperation between a software developer and electrical, mechanical,… engineers

- A good machine software developer is not only a software developer. A good understanding of hardware is required!

# More information

**A Reference Architecture for Local Machine Control**

- http://www.ni.com/white-paper/6145/en

- www.tm-solutions.eu

# Questions ?