Caution: this is a mere introduction to this vast subject

MSE | MASTER OF SCIENCE IN ENGINEERING

MA_EmbReal

Robust Patterns for Reliable Systems (I)

Version: 1.3

Recall: Our Mission

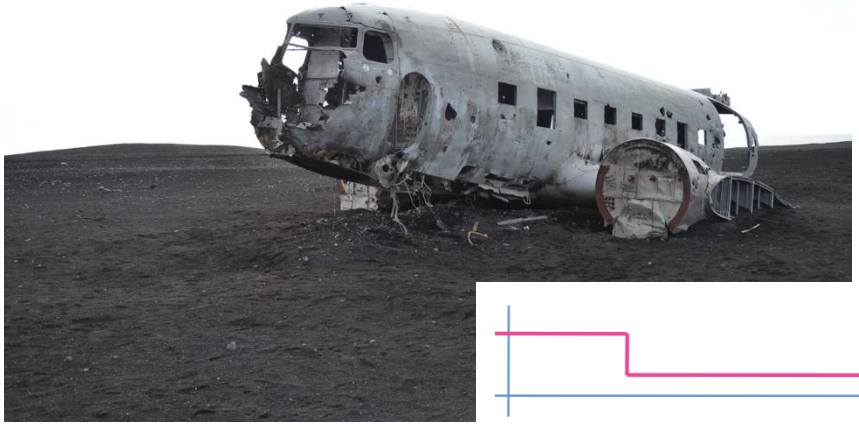
Recall: Our mission

- Program with a mix of periodic / aperiodic tasks
  - Address first scheduling of periodic tasks
  - Add aperiodic tasks
  - Add dependencies among tasks
- Demonstrate that a schedule is feasible given a set of tasks with their constraints and dependencies
  - Use known bounds and elaborate a feasible schedule
  - Compute bounds for blocking times
- Use the appropriate scheduling algorithm in simulation and practice
- Implement a system that meets timing constraints
  - **With functional safety concepts**
  - With timing constraints watchdogs

SIL Levels : A simplified recap

- Low SIL (1-2) are used in non life-threatening systems
- High SIL (3-4) are demanded in life-critical systems

When a fault does happen…



Degraded Operation

May be acceptable

Recovery

May be acceptable

Malfunction

Unacceptable

Crash

Unacceptable

swissuniversities

Hes·so Master

A Few Terms: Fail-???

Fail-safe

The system goes into safe mode when a failure occurs

Fail-silent

the system recognizes that it is receiving the wrong information due to a fault, so the ongoing operation moves to degraded mode (or, often, stops working entirely)

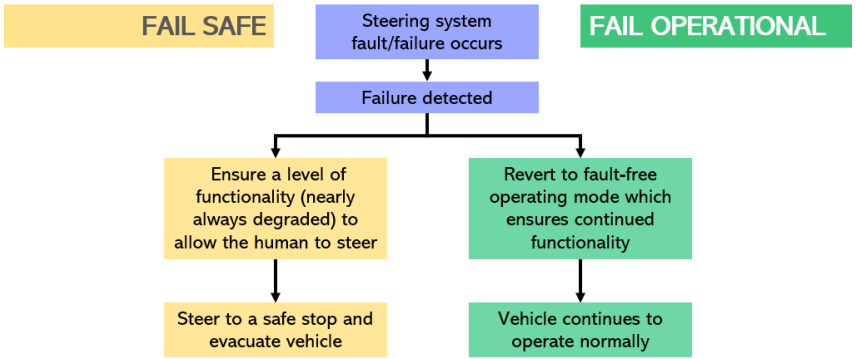
Fail-operational (also known as fault-tolerant)

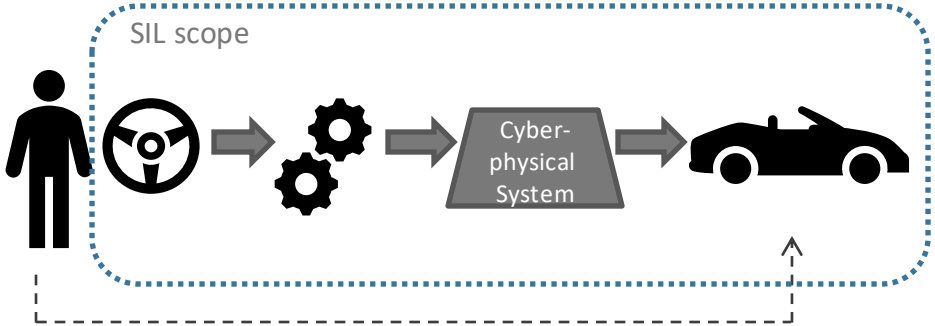
a failure in one component does not stop the whole system from working correctly, the system reconfigures itself to compensate for the fault

High-dependability

this is advanced failure prediction

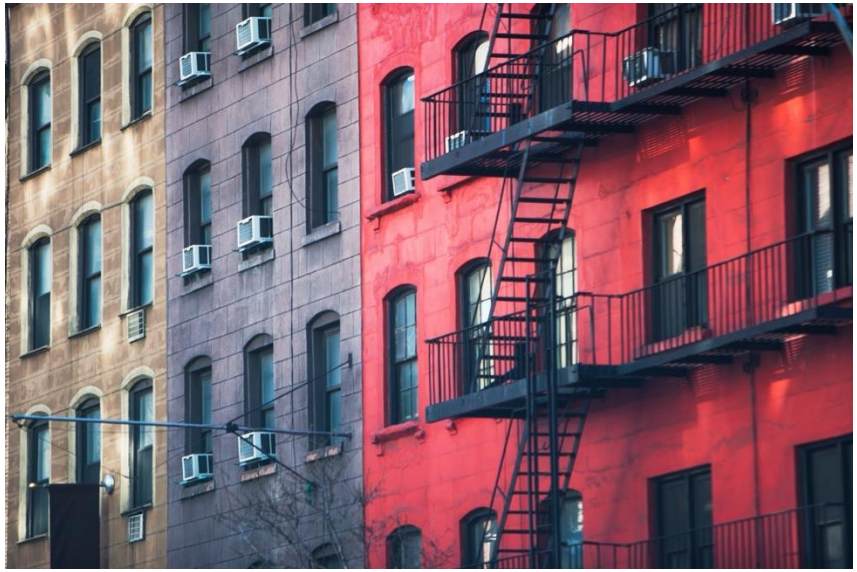
A Few Terms: an example



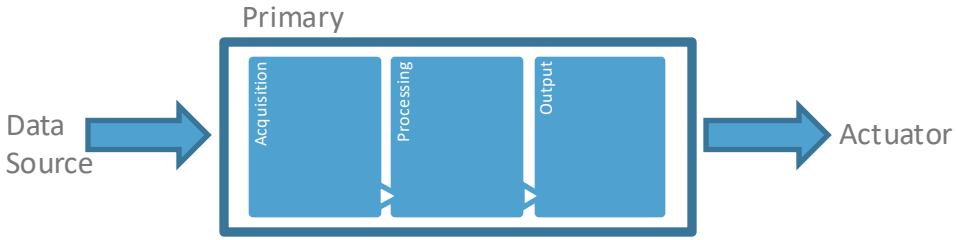


Word of caution (=> Antipattern)

- Have multiple, different SIL without separation
- Confuse fault detection & availability



Simplest System Pattern: 1-Channel



HW:
- 1 CPU

SW:
- -

- Advantages
  - Cheapest option
  - Simple
  - Suitable for SIL << MTBF
- Disadvantage
  - All SW needs to be according to highest SIL
  - Adapted to low SIL only

Pattern: 1-Channel + BIST

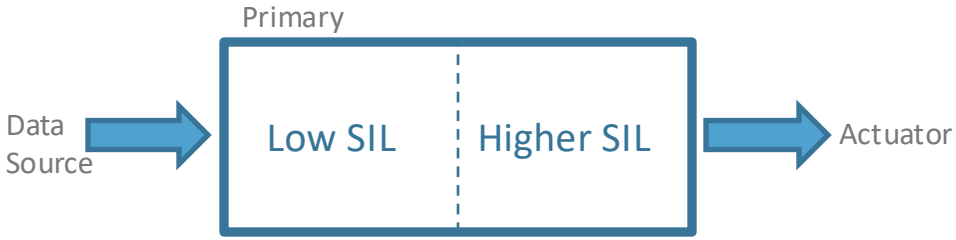
HW:
- 1 CPU

SW:
- Self-test libraries

- Advantages
  - Cheapest option
  - Simple
  - Suitable for SIL < MTBF
- Disadvantage
  - All SW needs to be according to highest SIL
  - Adapted to low SIL only
  - BIST covers HW failure rate detection – but still not SIL 3/4 ready

BIST : Built-In Self Test

Pattern: 1-Channel + SW Isolation



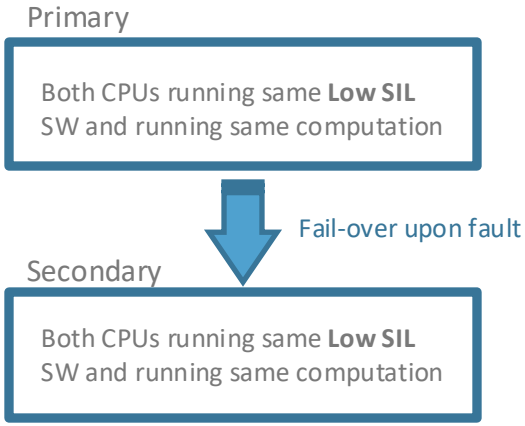
HW:
- 1 CPU

SW:
- Low/Higher SIL (1/2) partitioning

- Advantages
  - Simplest HW/SW SIL pattern
  - Relatively cheap HW
- Disadvantage
  - “Separation” needs to be proven
  - Adapted to low SIL (1-2) only
  - May be complex when 2 sides need to share information (-> partitioning shall not be made weaker)

Pattern: 2-Channel Failover

Fail-Operational

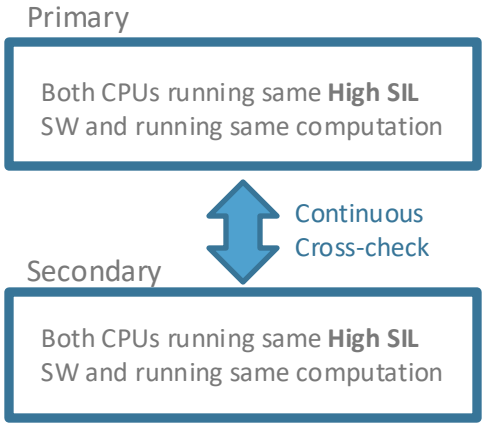


HW:
- 2 CPU

SW:
- -

- Advantages
  - Simplest high-availability pattern
  - Failover for simple failure modes
- Disadvantage
  - All SW needs to be according to highest SIL
  - Requires standby monitoring
- Critical note:
  - Secondary system does not improve SIL but availability

Pattern: 2-Channel



Fail-Silent

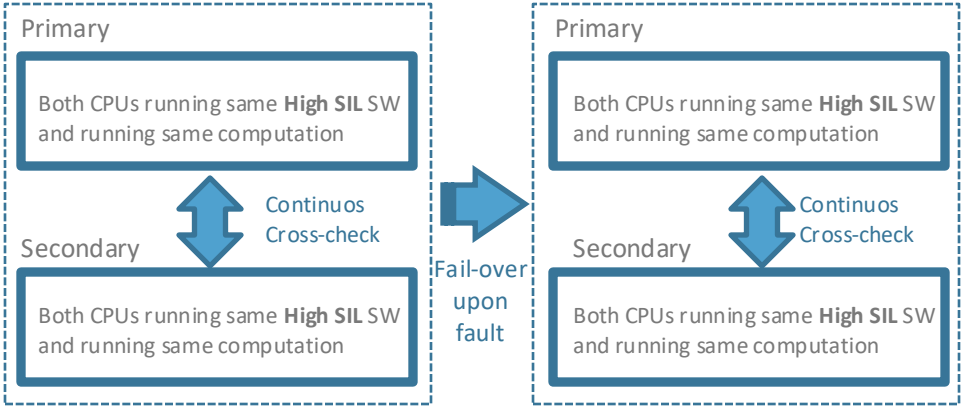
- Advantages
  - Simplest high-SIL pattern
- Disadvantage
  - All SW needs to be according to highest SIL (thus $$$$ SW)
  - Fails silently…

HW:
- 2 CPU

SW:
- -

Pattern: Dual 2-Channel

Fail-Operational

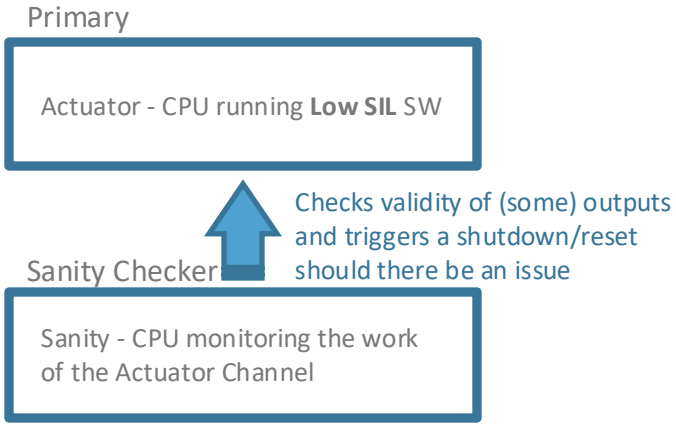


HW:
- 4 CPU

SW:
- -

- Advantages
  - Simplest high-SIL pattern
  - Fail situation handled with Secondary System
- Disadvantage
  - All SW needs to be according to highest SIL (thus $$$$ SW)
  - Requires Secondary System to be ready at “all times”

Pattern: Sanity Check



HW:

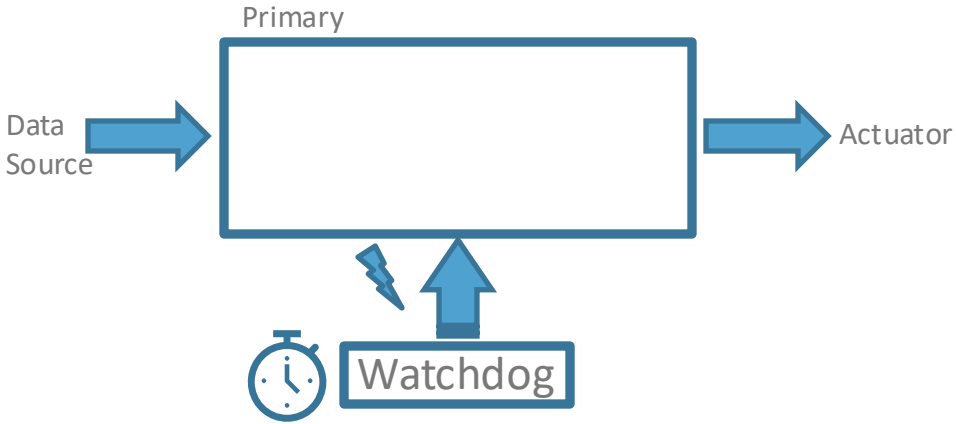
- 2 CPUs (though not the same)

SW:

- -

- Advantages
  - Gives a somewhat low-cost solution for checking, qualitatively, the Primary System
- Disadvantage
  - Only for Low SIL
  - Checker needs self-testing
  - Increasing the quality of the Sanity Checker increases the price
- Note
  - This pattern may be seen on a single CPU using SW isolation

Pattern: 1-Channel + Watchdog

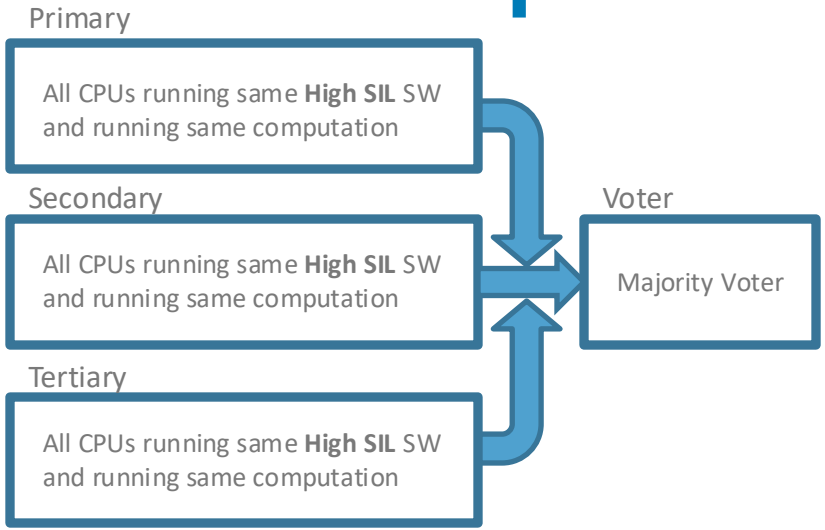


HW:
- 1 CPU
- 1 WD

SW:
- -

- Advantages
  - Very simple variant of Sanity Check pattern
  - Cheap, simple HW
- Disadvantage
  - All SW needs to be according to highest SIL
  - Adapted to low SIL only
  - Limited coverage (“all-or-nothing”)

Pattern: Triple Modular Redundancy



HW:
- 3 CPU
- 1 Voter HW

SW:
- -

- Advantages
  - High-SIL pattern with high-availability
  - Voter HW may be inexpensive
  - Faulty Channel “outvoted”
- Disadvantage
  - All SW needs to be according to highest SIL
  - Voter is a single point of failure
  - Not the cheapest option

Pattern: 1-Channel + Watchdog

- Let’s put this into practice (embreal homepage ->Codelabs->Robust Design Patterns – Part 1)

Defense Programming

Defense Programming

- Is an attitude
- Whose aim is
  - detect potential abnormalities proactively
  - make the SW predictable
  - improve quality

```
int add(int a, int b)
{
  int result = a + b;
  if (a < 0 || b < 0) {
    return -1;
  }
  if (result < 0) {
    report_overflow();
  }
  return result;
}
```

Defense Programming in Practice

- Let’s put this into practice (embreal homepage ->Exercices->Robust Design Patterns – Part 1)

References

- Systematic pattern approach for safety and security co-engineering in the automotive domain (https://api-depositonce.tu-berlin.de/server/api/core/bitstreams/eb16c756-d7fa-46b1-a96d-3c2c854a3063/content)
- Design Patterns for Safety-Critical Embedded Systems (https://d-nb.info/1007034963/34)
- Red Hat Defensive Coding Guide (https://developers.redhat.com/articles/defensive-coding-guide)
- Defensive Programming - Friend or Foe? (https://interrupt.memfault.com/blog/defensive-and-offensive-programming)