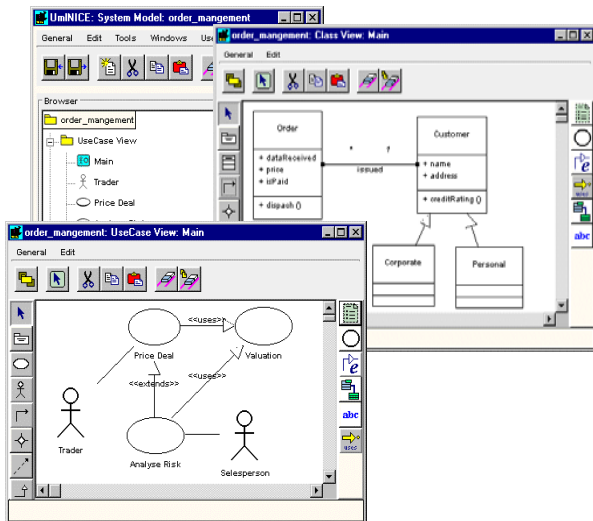




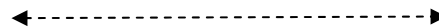
Safety Issues in Automotive Software

Paolo Panaroni, Giovanni Sartori

INTECS S.p.A.



“SAFWARE”





INTECS & Safety

- A very large number of safety software development, V&V activities and research project on safety, since 1974 (defense, space, nuclear, railway, automotive)
- SW RAMS Methods for the European Space Agency, ECSS standards contribution (on board sw)
- Software Reliability Modeling (with Center for Software Reliability, UK)
- International Workshop on Software Reuse and Safety, Turin, June 2006 (Program Organization)



Common misconceptions

- Software “per-se” does not harm anybody
(but it can order to harm !)
- Software Safety = Software Reliability
(“a car that does not start is safe”)

Nancy Leveson



Safety vs. Reliability

- Reliability has to cope with MTBF while safety has to cope with failure occurrences, consequences and severity
- A number of safety **protective measures** may even adversely affect reliability (in suspect => fail safe).



Quantitative Analysis

- The application of quantitative analysis to automotive software is generally considered **impractical** since it is not possible to quantify the probability of software failure
- An increasing level of rigour is applied, **instead**, to the development and verification process (e.g. ASIL A,B,C,D)



Automotive “safeware” examples

- **ABS** (*Anti-lock Braking Systems*)
- **ACC** (*Adaptive Cruise Control*)
- **ESC** (*Electronic Stability Control*)
- **BBW, EPB**
- **SBW**
- **Airbags, light control, dashboard**
- **Tyre pressure, lane departure warning**
- **etc.**



Automotive Safety Standards

- **IEC 61508 Part 3 (generic, cross domain)**
- **MISRA Integrity Guidelines**
- **MISRA C:2004**
- **WD 26262 part 6 (work in progress, started Nov. 2005)**
- **MIL STD 882D, DO 178B, ECSS Q30/Q40, DEF STAN**
- **Studies/Projects (examples)**
 - **DRIVE Safely**
 - **PASSPORT**
 - **EASIS**



Safety relevant functions

- software function output =>
 system malfunction =>
 driver loss of control =>
 accident =>
 harm
- if a software function output may lead to an harm that software function is said **safety relevant** (**safety critical**)



Safety Integrity Level

Safety relevant software function implies to:

1. Verify that what is **specified is safe**
2. Ensure **implementation meets specification**:
adopt a rigorous (high integrity) process for its development and verification
3. Develop anyhow **protective measure** (either HW or SW or both) (includes Fault Detection, Isolation and Recovery - FDIR)



Software Failure Modes

- Omission (output is not provided)
- Commission (output is provided when not required)
- Late
- Early
- Wrong Value coarse (detectable/not palusible)
- Wrong Value subtle (undetectable/plausible)
- Stuck (Value does not change)



FAILURE TOLERANCE

- FAIL
- FAIL SILENT
- FAIL SAFE
- FAIL DEGRADATED/FAIL REDUCED
- FAIL (FULL) OPERATIONAL



FAULT TREATMENTS

- FAULT PREVENTION
- FAULT REMOVAL
- FAULT DETECTION/RECOGNITION
- FAULT ISOLATION/CONTAINMENT
- FAULT RECOVERY



FAULT PREVENTION

- Mature and rigorous software process (e.g. SPICE, CMMI)
- Software life cycle (e.g V life cycle)
- Simplicity (KISS), FMEA
- Semi-formal methods (e.g. UML, Simulink)
- Standard architectures (AutoSAR)
- MISRA C, Certified compilers, kernels, libs (e.g. OSEK)
- Dual programming
- Metrics conformances (e.g. nesting depth, cyclomatic complexity, also at Model level)
- Schedulability Analysys (WCETA)
- Memory Usage Analysis (WCMUA)
- Safety oriented organization (independence, competence, planning, verification, assessment, suppliers control)



FAULT REMOVAL

- Peer Reviews/Code inspections
- Static Analysis Tools (on code*, on models)
- Unit test (100% MD/MC, branch coverage)
- Re-run the test with no instrumentation and on the target
- Boundary value and Equivalence class
Partitioning tests
- Integration test (100% call-pair coverage)
- Validation test (100% requirement coverage)
- Defects seeding/Fault injection

() Tools like polyspace perform an extensive analysis capable to intercept typical run time errors. Run time errors may be as high as 30%- 40% of total errors*



FAULT DETECTION

- Plausibility checks/defensive programming/assertions/pre-post conditions
- Checksum
- Data redundancy

- Watchdog
- Intelligent watch dog (Reference value check Q/A game)
- Redundant/diversified function



ISOLATION/CONTAINMENT

- Damage assessment
- Fault region identification
- Confinement
- Prevent propagation/cascade



FAULT RECOVERY/HANDLING

- **REMAIN SILENT / SEND ERROR CODE/STORE Diagnostic Trouble Code**
- **CORRECT** (e.g. parity checks)
- **RESET/RETRY**
- **ALTERNATIVE SAFE** (e.g. switch to mechanical link)
- **ALTERNATIVE DEGRADATED** (e.g. simplified break control based only on the pedal stroke value)
- **ALTERNATIVE OPERATIONAL** (diversified programming/N version programming)

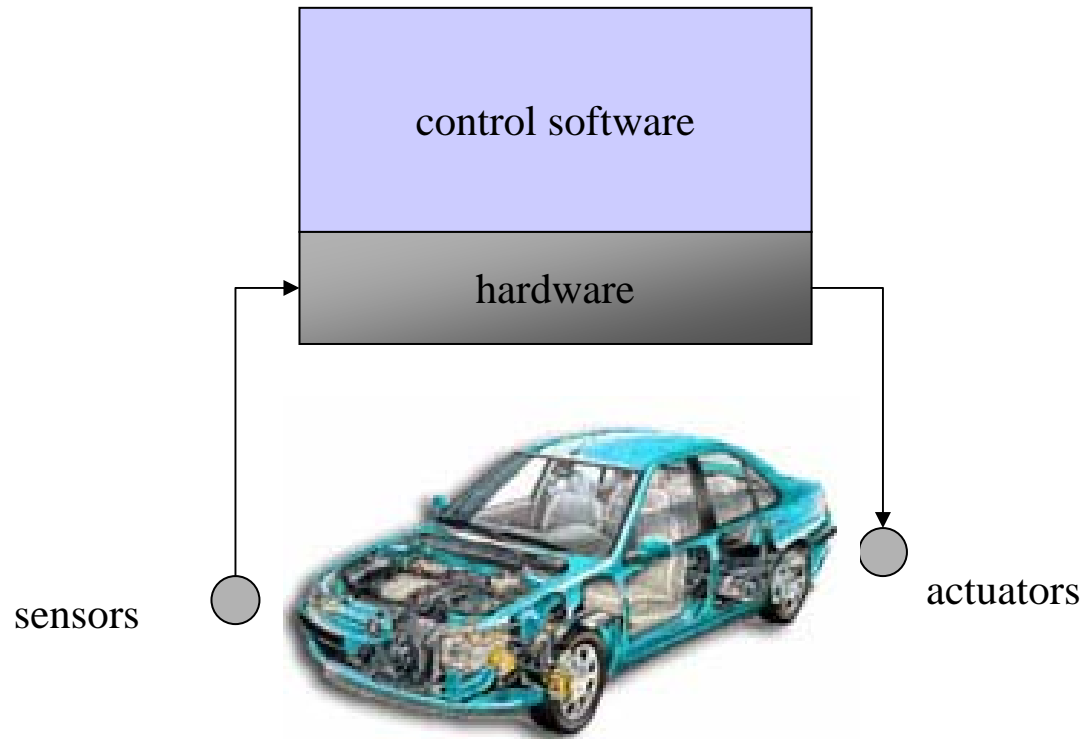


Software Safety Architectures

an overview

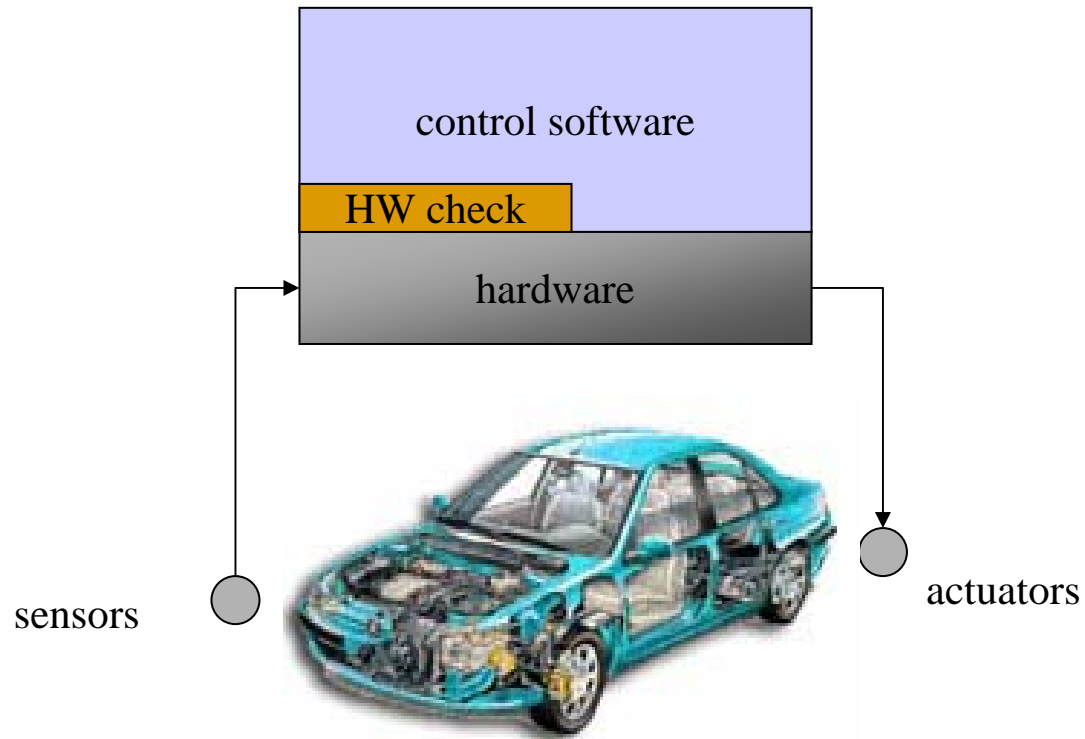


Simplex approach



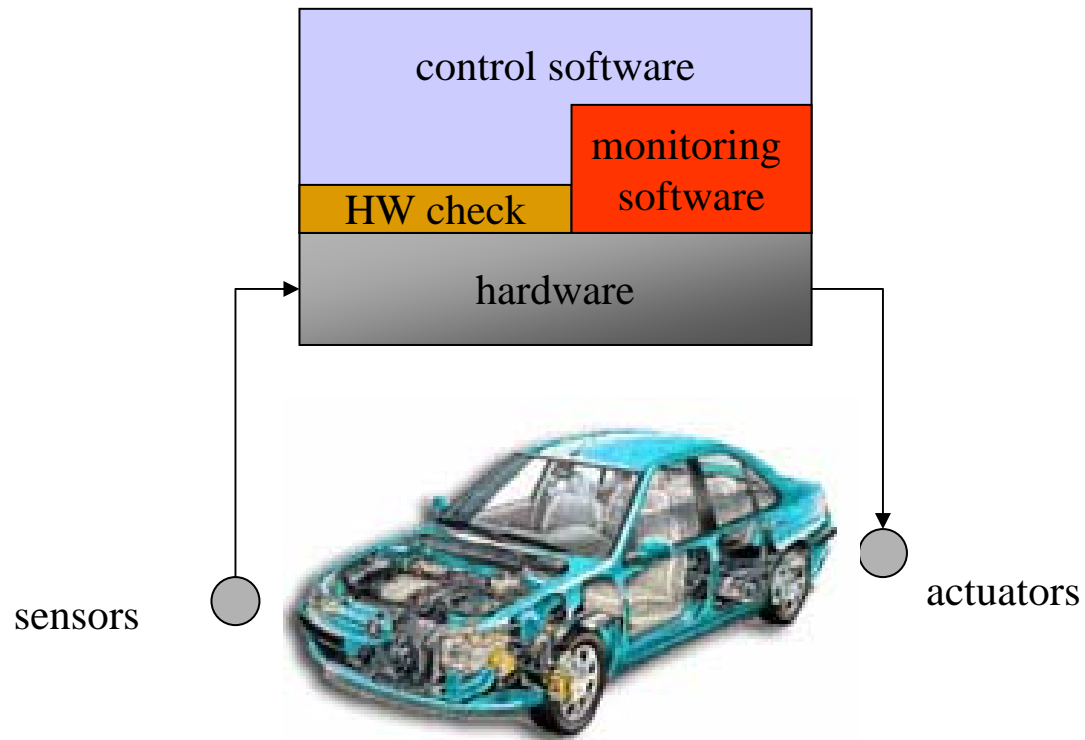


Software checks Hardware



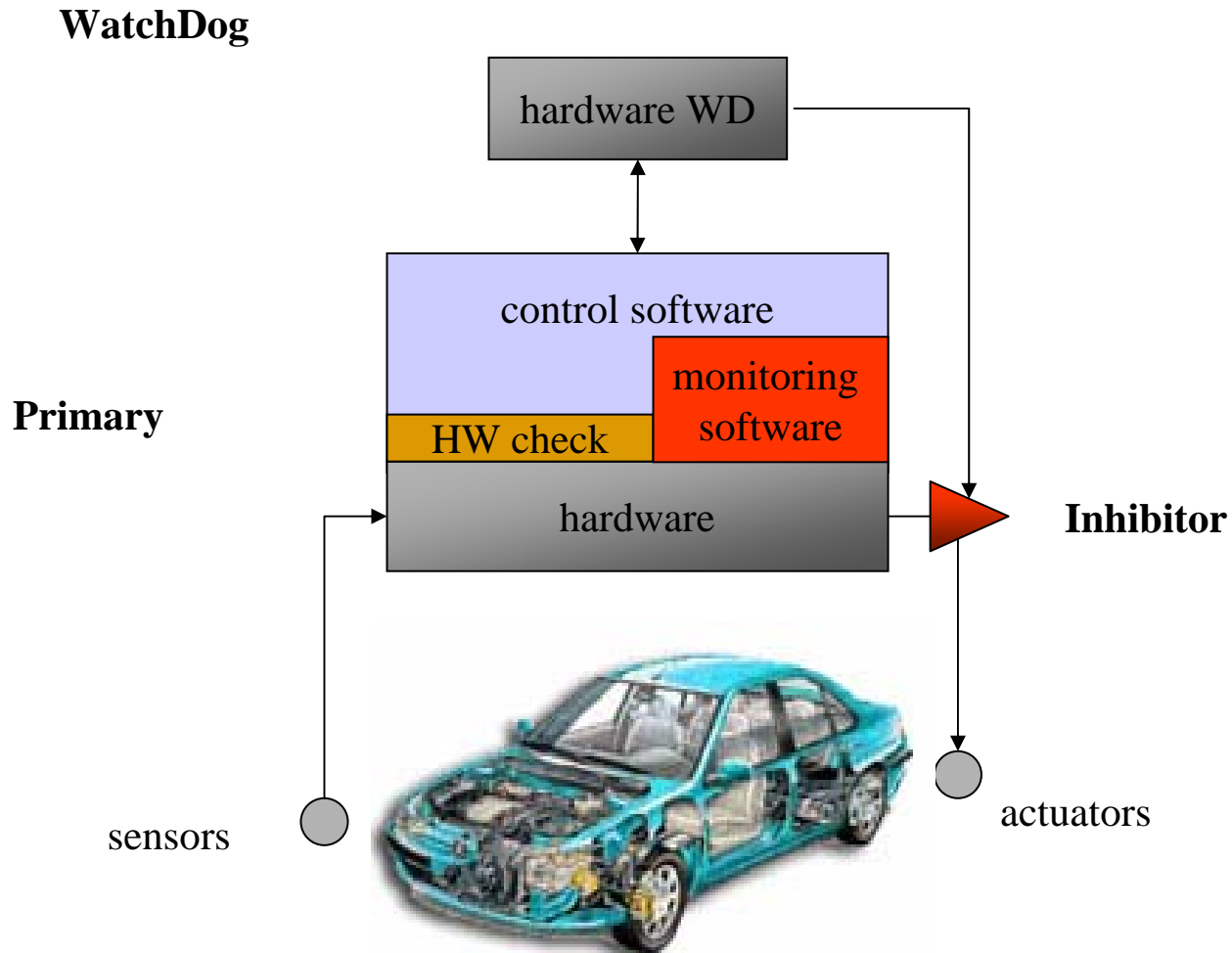


Self checking software



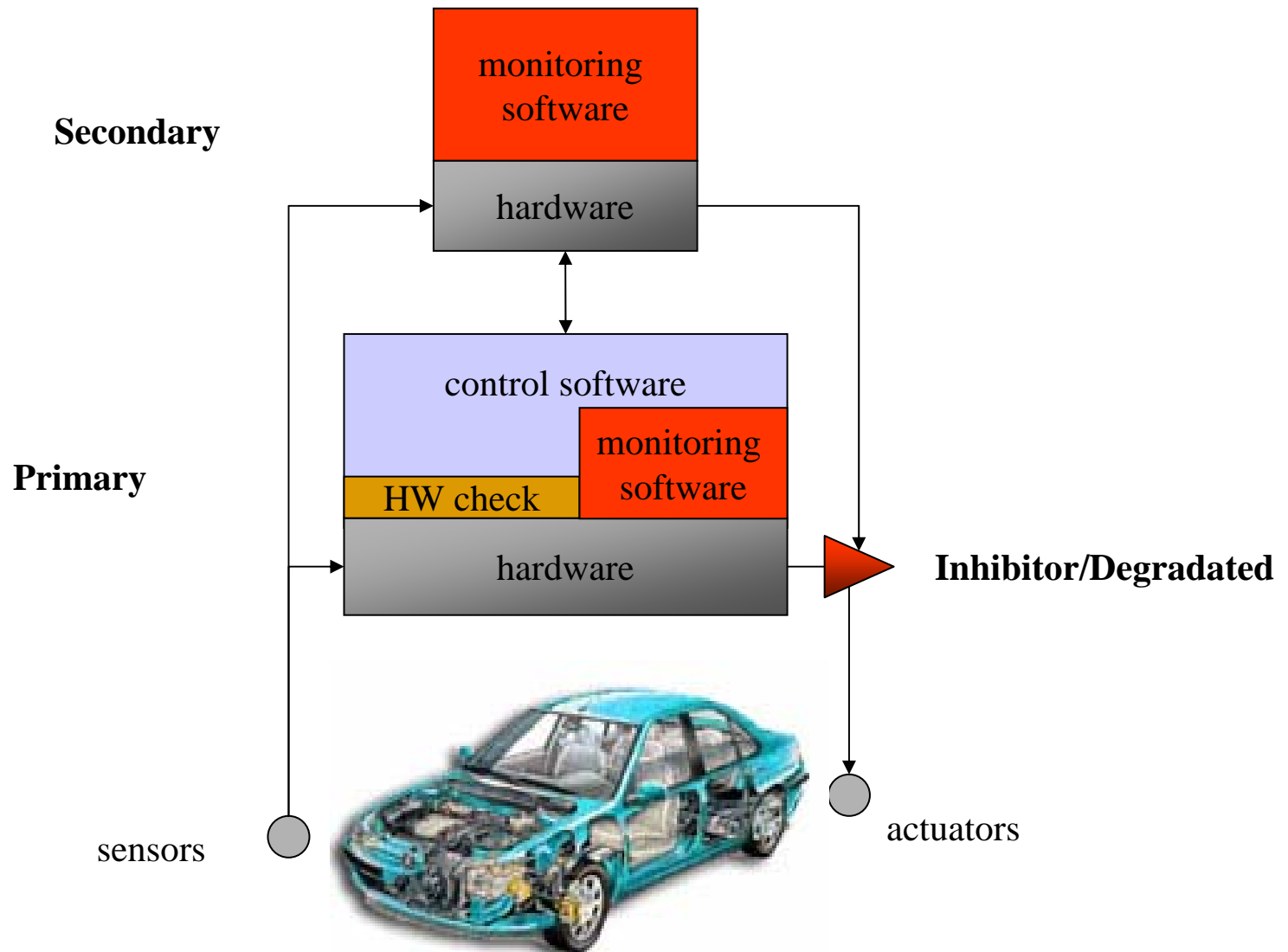


Watchdog



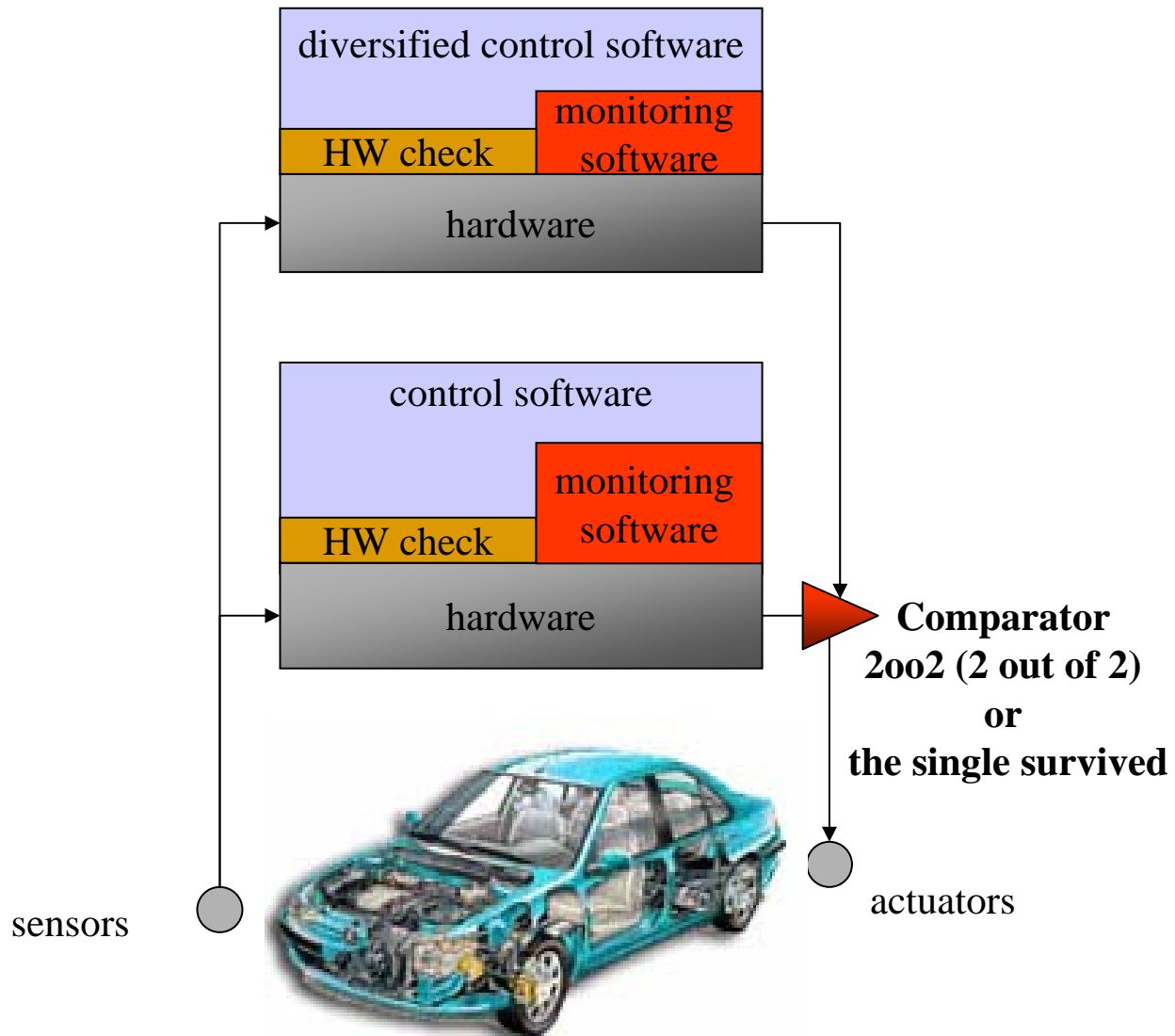


Asymmetric/Intelligent Watchdog



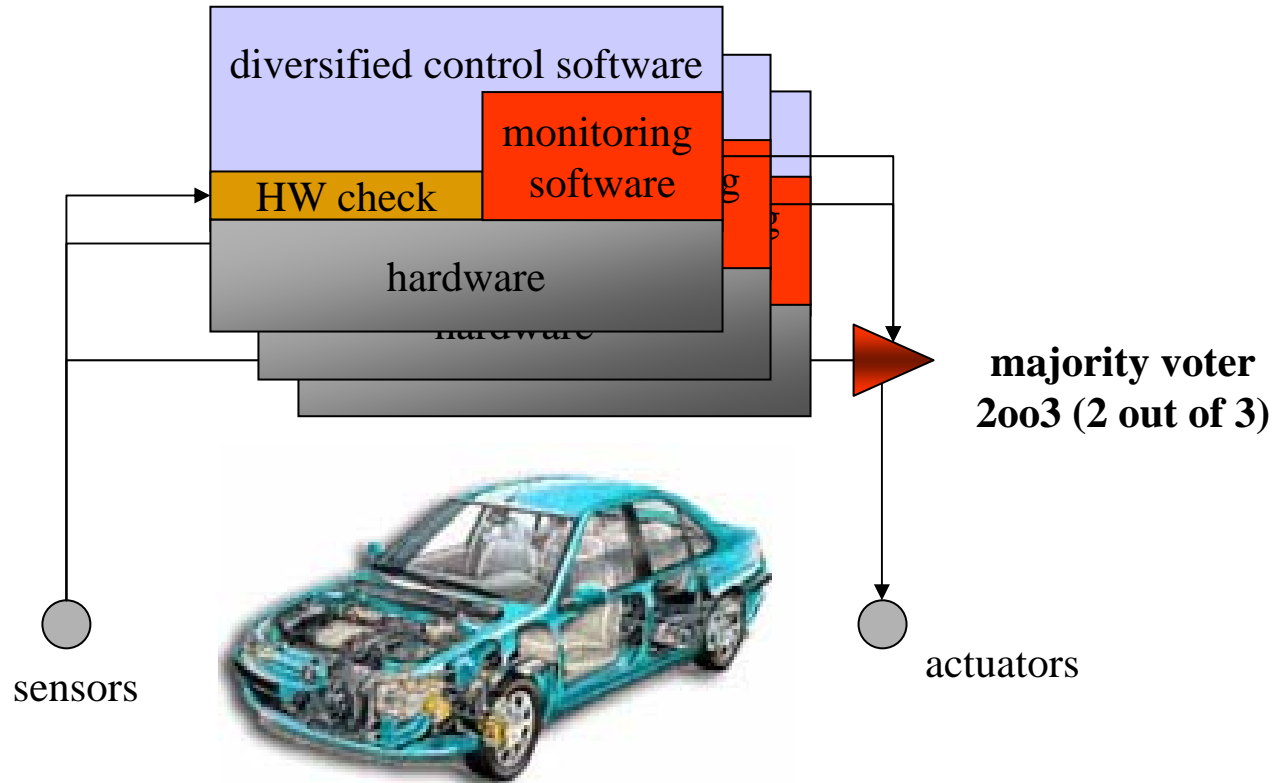


Duplex/Dual redundant





Triplex/Triple Redundant



It is not proven that the effort required to develop diversified software versions provides better results than building a **single** robust software version (golden version)
Common Mode failures may be very high