Safety-Critical Systems

Introduction to Functional Safety

Safety and Security

- Safety
  - the condition of being protected from or unlikely to cause danger, risk, or injury
- Security
  - is freedom from, or resilience against, potential harm (or other unwanted coercive change) caused by others

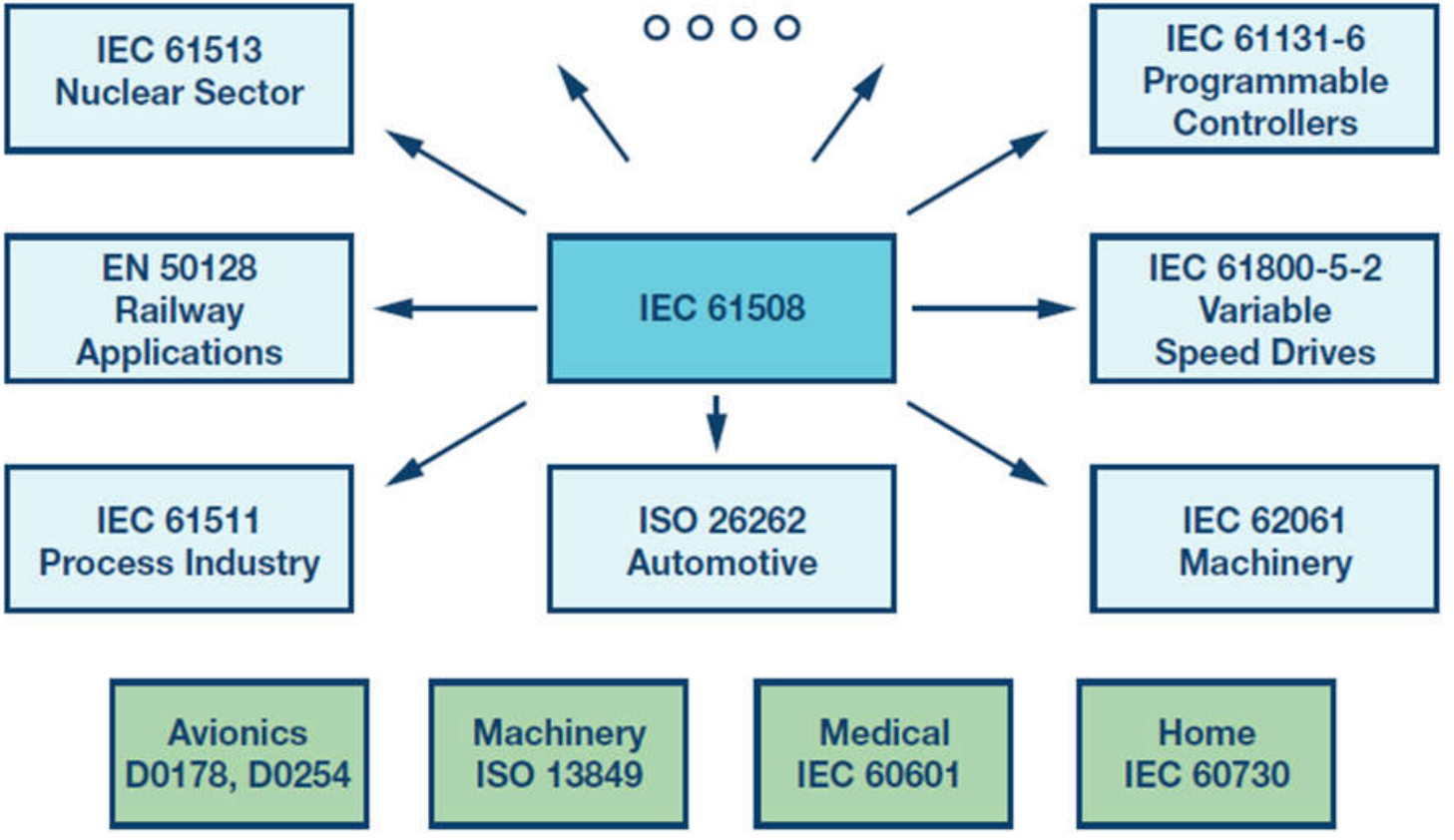
Functional safety

- The part of the overall safety of a system or piece of equipment that depends on automatic protection operating correctly in response to its inputs or failure in a predictable manner (fail-safe)
- The automatic protection system should be designed to properly handle likely human errors, hardware failures and operational/environmental stress

When F5 and reboot doesn't do the trick

- Process control systems, process shut down systems, rail signaling equipment, avionics, automotive controls, medical treatment equipment
- Systems with potential hazardous failures which lead to serious consequences (e.g. death)
- Safety-related equipment - equipment whose failure may lead to hazard
- Safety function - maintains the system in a safe state, or brings it to a safe state
- **Availability**
  - The failure to provide an answer within the time that makes it useful is termed an *availability* problem
- **Reliability**
  - the timely presentation of the wrong answer is a *reliability* problem

IEC61508 – Functional safety



Type of failures - quantification

- **Random hardware failures**
  - failures which can be quantified and assessed in terms of failure rates
- **Systematic failures**
  - cannot be quantified, can be addressed by levels of rigor in the design techniques and operating activities
- **Software failures**
  - **Heisenbug** a software bug that seems to disappear or alter its behavior when one attempts to study it
  - **Bhorbug** a repeatable bug one that manifests reliably under a possibly unknown but well-defined set of conditions

SIL

- **Safety-integrity levels - SIL**
  - maximum tolerable failure rate that we set, for each hazard, will lead us to an integrity target
  - IEC 61508 and its derivatives are firmly rooted in the idea that the study of device safety is the study of device failure
- SIL 4: the highest target and most onerous to achieve, requiring state of the art techniques (usually avoided)
- SIL 3: less onerous than SIL 4 but still requiring the use of sophisticated design techniques
- SIL 2: requiring good design and operating practice to a level such as would be found in an ISO 9001 managements system
- SIL 1: the minimum level but still implying good design practice,
- SIL 0: referred to as “not-safety related” in terms of compliance

SIL4 probability of failure

Hours	Years	Probability of a failure
100,000	11	0.001
1,000,000	114	0.00995
10,000,000	1,141	0.09516
100,000,000	11,408	0.63212
1,000,000,000	114,077	0.99995

ASIL

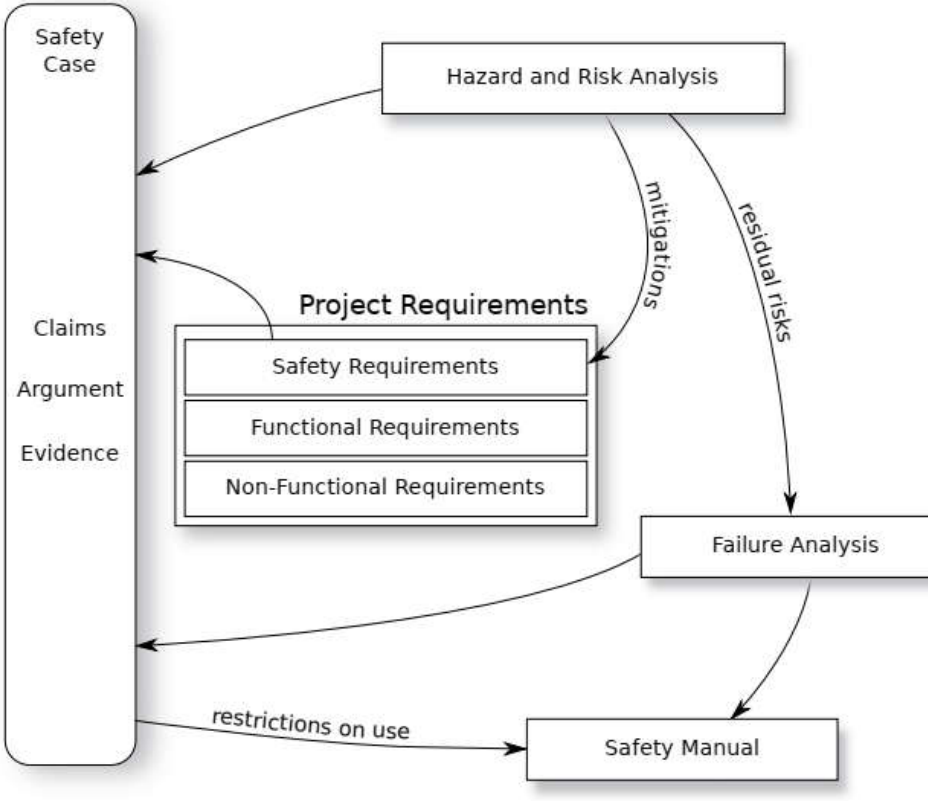
- Not based on probability of failure per hour of use. Instead, the events that could cause injury are listed and an ASIL is calculated for each of them
- ***What type of injuries could result from this event?***
  - escriptions can be used to assess the severity of the risk on a scale from S0 (no injuries) to S3 (life-threatening and fatal injuries)
- ***How likely is the event to occur in normal operation?***
  - from E0 (incredible) to E4 (high probability) and gives further guidance that E1 is applied to events that occur less often than once a year and E2 to events that occur a few times per year
- ***How many drivers could control the situation and avoid injury?***
  - The scale here is from C0 (controllable in general) to C3 (difficult to control or uncontrollable)

ASIL classification

Severity	Likelihood	Controllability		
		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	ASIL A
	E4	QM	ASIL A	ASIL B
S2	E1	QM	QM	QM
	E2	QM	QM	ASIL A
	E3	QM	ASIL A	ASIL B
	E4	ASIL A	ASIL B	ASIL C
S3	E1	QM	QM	ASIL A
	E2	QM	ASIL A	ASIL B
	E3	ASIL A	ASIL B	ASIL C
	E4	ASIL B	ASIL C	ASIL D

Safety requirements

- **Safety plan**
  - This lays out the specific practices to be followed during the development of the product. For example, a company may have several coding standards, each suitable for a different type of coding; the safety plan specifies which is to be used for the project. It also defines the precise responsibilities of each project member
- **Safety case**
  - in the legal sense: The case for the defense
  - reasoned argument explaining why you believe the product you are developing is safe

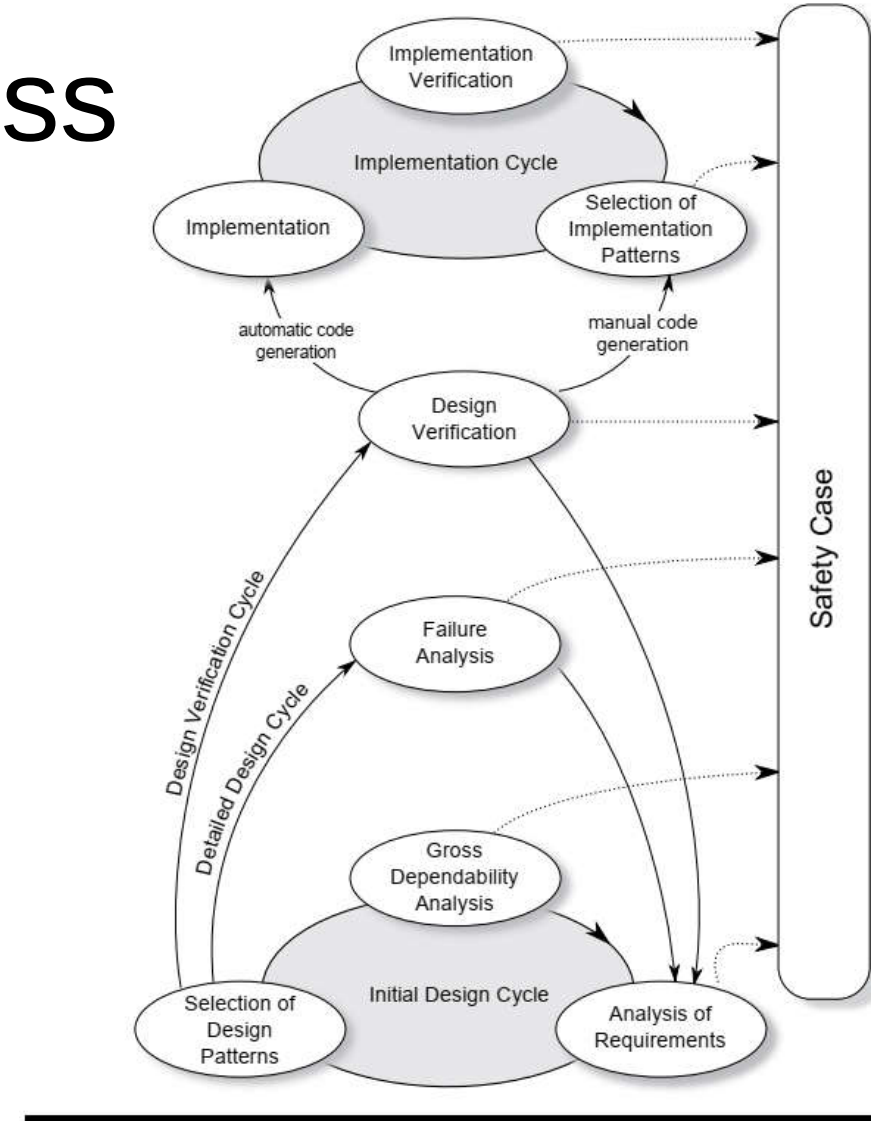


Risk assessment

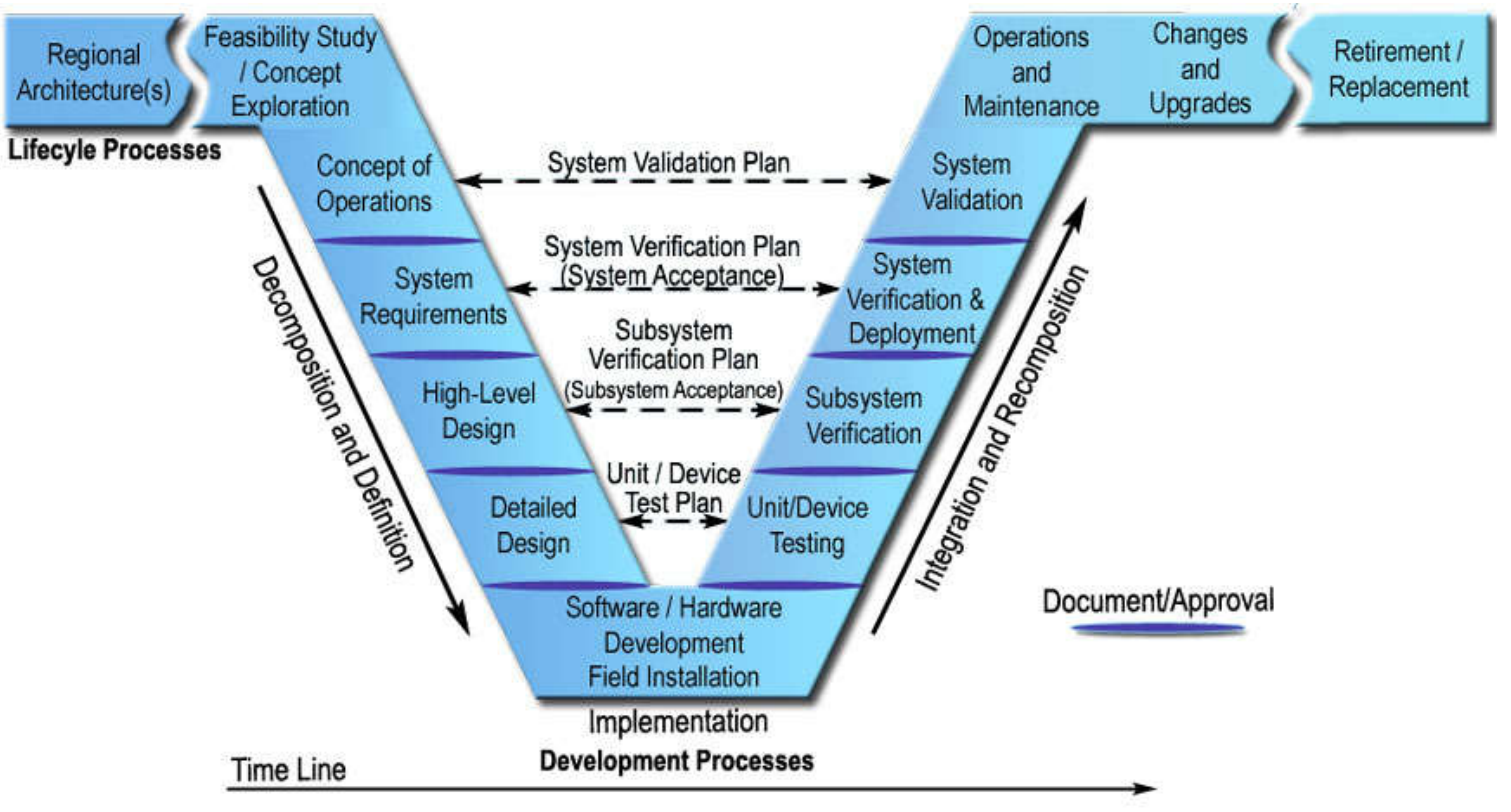
- **Hazard and risk analysis.**
  - This identifies the hazards and risks associated with the use of the final product (for the distinction between a hazard and a risk and the meaning of mitigation
  - This analysis is particularly crucial because it is from this that the project’s safety requirements and data for the failure analysis are derived
- **Failure analysis**
  - All systems eventually fail, and it is important to understand in what ways, and how often, your system is expected to fail
  - **FMEA** If this component were to fail, what dangerous condition might occur - bottom u approach. British Handbook of Reliability Data for Electronic Components used in Telecommunication Systems - HRD5, MIL reliability data, tables list every type of component (resistor, capacitor, inductor, etc.) and provide rates for each type of failure
  - **FTA** What might cause this particular dangerous condition to occur?" - top down approach

Development process

- Initial design cycle
- Detailed design cycle
- Design verification
- Implementation
- Verification
- Validation
- Certification



V process design model



Software development process

- ***Software development process***
  - Subprocesses covering the steps from initial planning to software release
- **Software maintenance process**
  - Covers effectively the same eight subprocesses
- **Software configuration management process**
  - Includes the identification and correct handling of SOUP and the handling of requests for changes to the device
- **Software problem resolution process**
  - This includes the obligation to inform users of the device of potentially dangerous bugs

The Balance

- **Availability safety balance**
  - It is trivially easy to build a very safe system
  - The balance that has to be achieved is that between usefulness and safety, and this is where a highly reliable system (e.g., 2oo2) can become unacceptable
- **Design Balance**
  - It is possible to program for high performance (fast, tightly-knit code), for testability, for ease of maintenance, for efficiency of the static analysis tools, for efficient runtime error detection, or for code readability
  - These demands in general work against each other

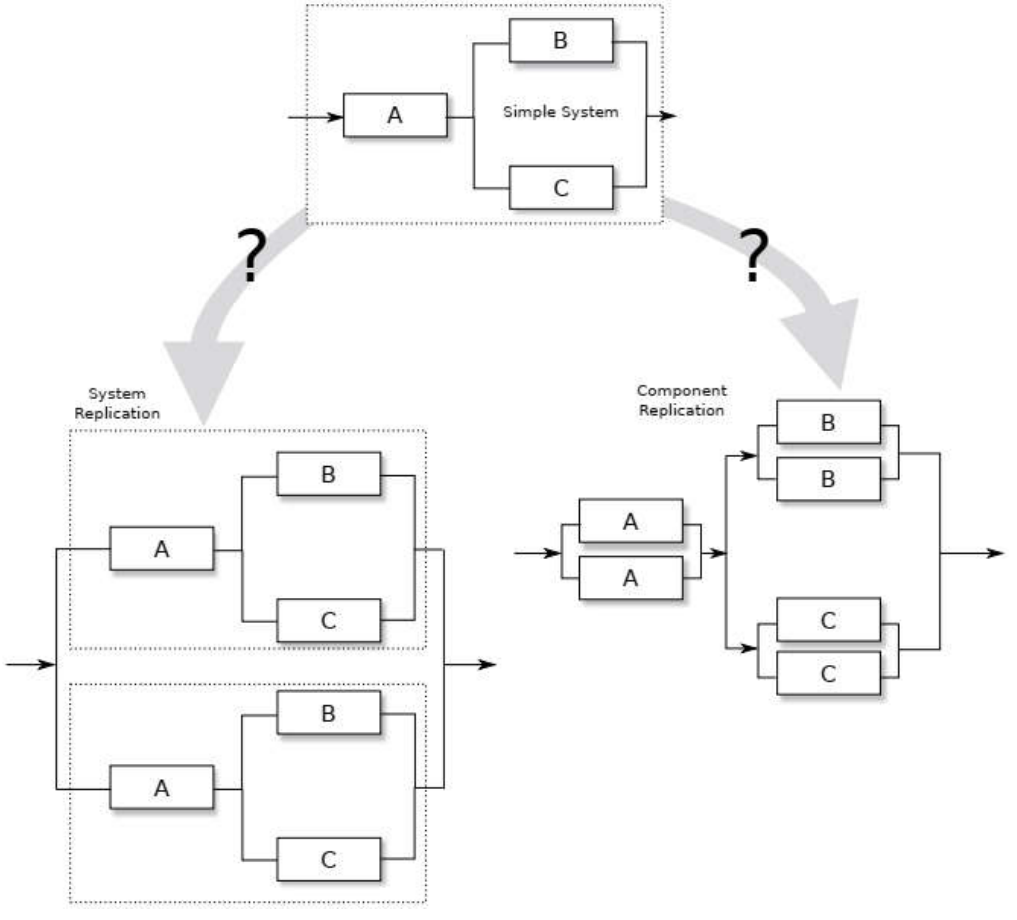
Challenges

- ***Hardware Error Detection***
  - processor and memory hardware has become significantly less reliable over the last decade
- **Software integrity**
  - Code quality, code coverage, traceability
- **Real time determinism**
  - As real-time applications migrate to running alongside other applications on multicore processors with non-deterministic instruction and data caches, these hard guarantees are disappearing and being replaced by guarantees of the form \the probability of deadline *X* being missed is less than 10−6 per hour of operation.
- **Data Integrity**
  - The development of safety-critical systems has only recently begun to take data, particularly configuration data, seriously

Error handling

- ***Backward Error Recovery***
  - Once an error is discovered, the system returns to a previously stored state that is known (or believed) to be consistent and sane. Whether the input that apparently caused the error is then reapplied or discarded, and how the client is notified, vary between implementations. One of the disadvantages of backward error recovery is the need to store the system's state information before handling an incoming request so that it is available if rollback is required.
- ***Forward Error Recovery***
  - This removes the main disadvantage of backward error recovery the need to save the system's state | by moving to a predefined sane state independent of both the previous state and the input that caused the error.
- ***Safe state design***
  - The action to be taken is to move, if possible, to the system's currently applicable \design safe state" defined during the system design.

Replication

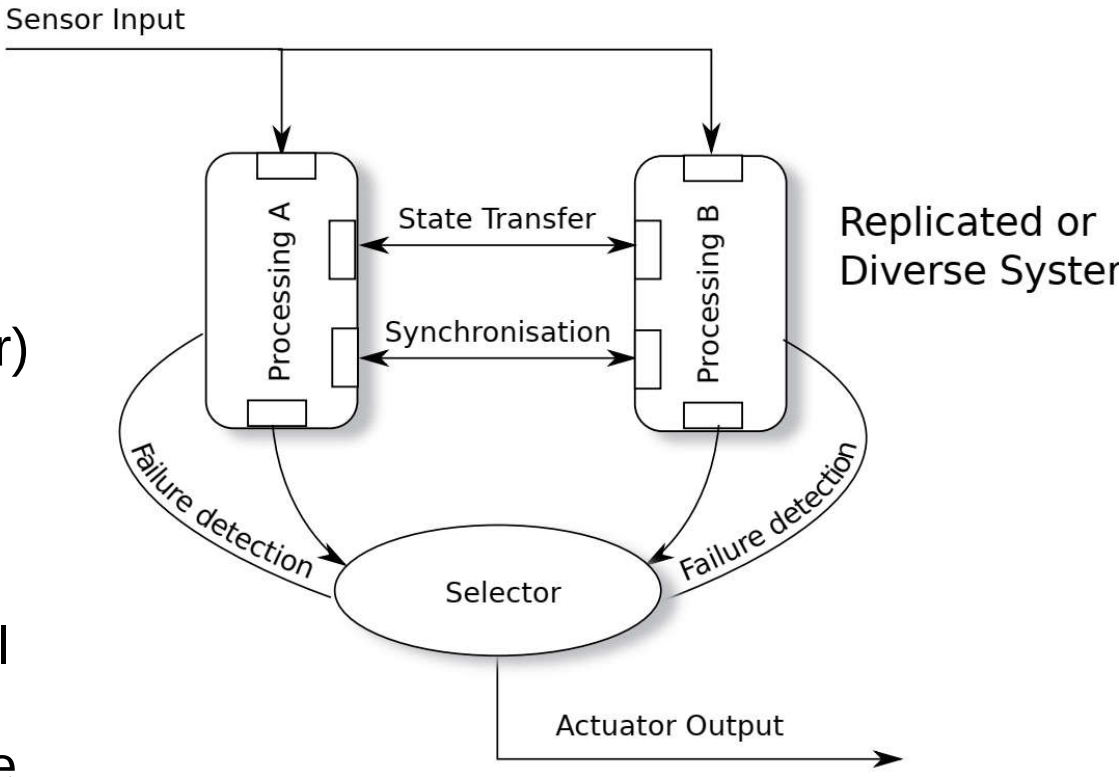


- I*t can be tuned to provide reliability or availability*
  - By changing the algorithm of how the middleware handles responses, the system can be configured to provide availability or reliability

$$\phi(\vec{x} \coprod \vec{y}) \geq \phi(\vec{x}) \coprod \phi(\vec{y})$$

Diversity

- ***Hardware Diversity***
  - It would be possible for the two subsystems in to be running on different processors (perhaps an ARM and an x86 processor)
- ***Software Diversity***
  - There are many levels of software diversity. Perhaps the simplest is to use a tool to transform one source code into an equivalent one



Software diversification

```
uint8_t do_and_regular(uint8_t x, uint8_t y)
{
    unsigned char s;

    s = x & y;

    return s;
}

uint8_t do_and_morgan(uint8_t x, uint8_t y)
{
    uint8_t s, x1, y1;

    x1 = ~x;
    y1 = ~y;
    s = ~(x1 | y1);

    return s;
}
```

Diversity continued

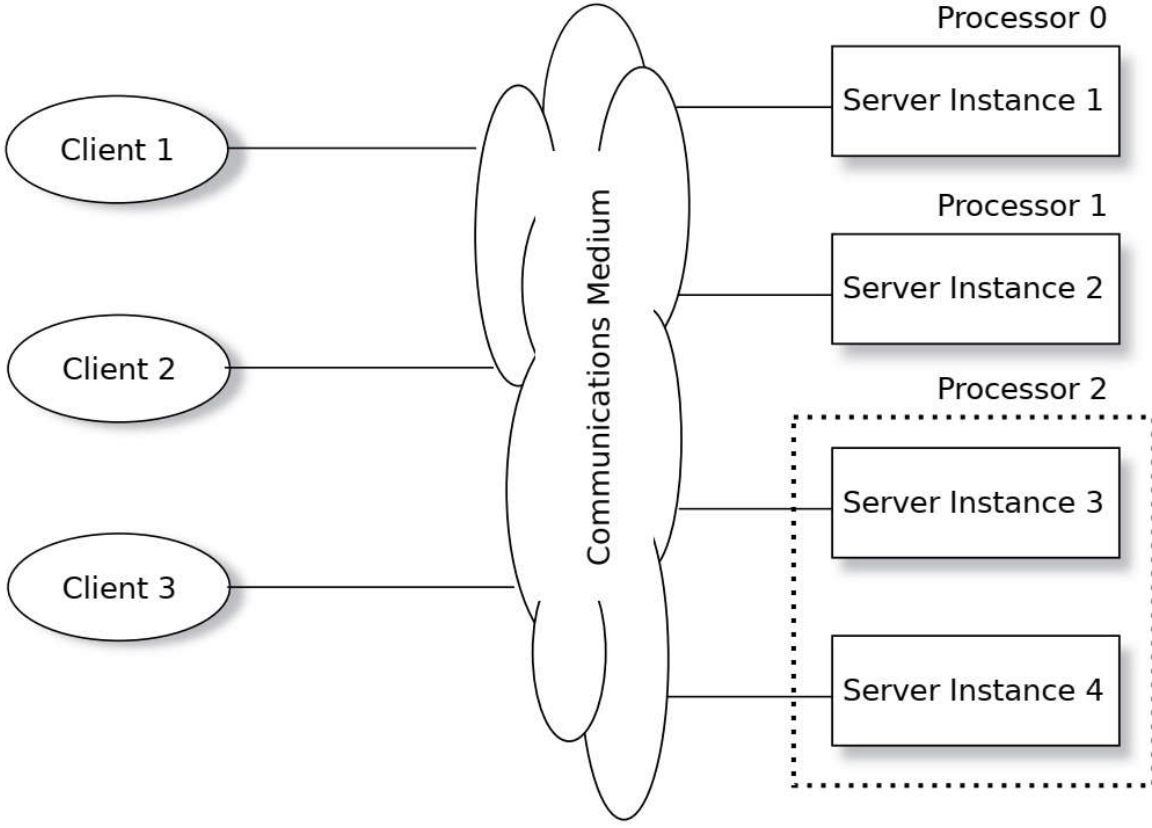
- **Coded processors**
  - is an extension of the concept of program diversity, whereby a code is stored with each variable and is used to detect computational errors caused by incorrect compilers, hardware errors (bit flips), and incorrect task cycling. As variables are manipulated (e.g., $z = x + y$ or $x > 0$), so are the appropriate codes. Software or hardware can then check the computation by checking the validity of the new code

$$x_c = A \times x_f + B_x + D$$

- ***N-version programming***
  - *is defined as the independent generation of N ≥ 2 functionally equivalent programs from the same initial specification.*
  - For the particular problem that was programmed for this experiment, we conclude that the assumption of independence of errors that is fundamental to some analyses of N-Version programming does not hold
- ***Data Diversity***
  - Of all the forms of diversity, data diversity is perhaps the most useful, and least controversial. The idea is to store the same information using two or more different data representations
  - The same information might be held as data in both the time and frequency domains

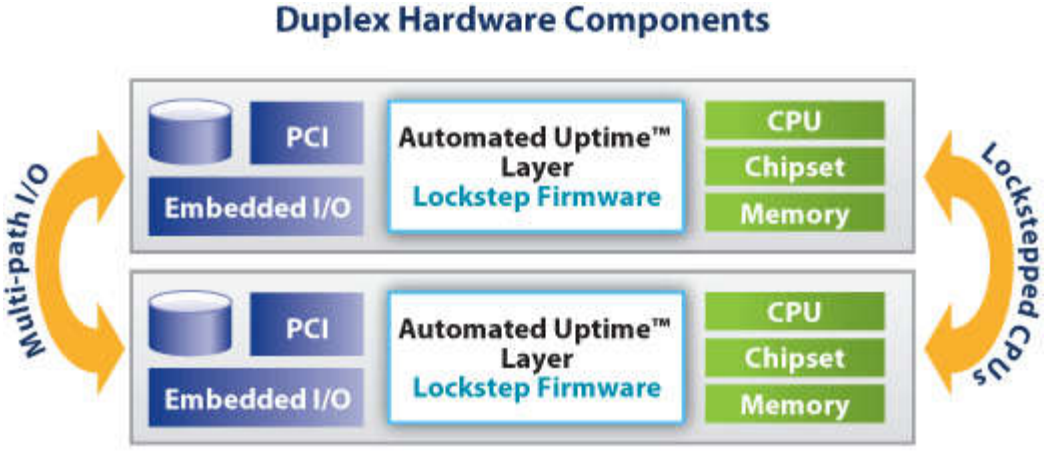
Virtual synchronous

- Servers all start in the same state (and the updating of a new group member's state is also part of the protocol) and if each server has received the same messages in exactly the same order, then all the servers must eventually arrive at the same state



Lock-Step processing

- One surprisingly popular form of replication has two processors running the same code in locked step, each executing the same machine code instruction at the same time
- Locked step processing may again have a positive effect on the reliability aspect of the failure analysis, while having a negative effect on the availability
- Will not catch either form of software bug



Arbitration and monitoring

- **Diverse Monitor**
  - The monitor observes the inputs and outputs, and may also observe internal states of the main system
  - In many systems, it may be implemented in hardware without any software, possibly using a field-programmable gate array (CPLD or FPGA)
- ***Watchdog Diverse Monitor***
  - One very common diverse monitor is the watchdog
  - the main system is required to “kick" the watchdog (the monitor) periodically by sending it a message or by changing a memory value