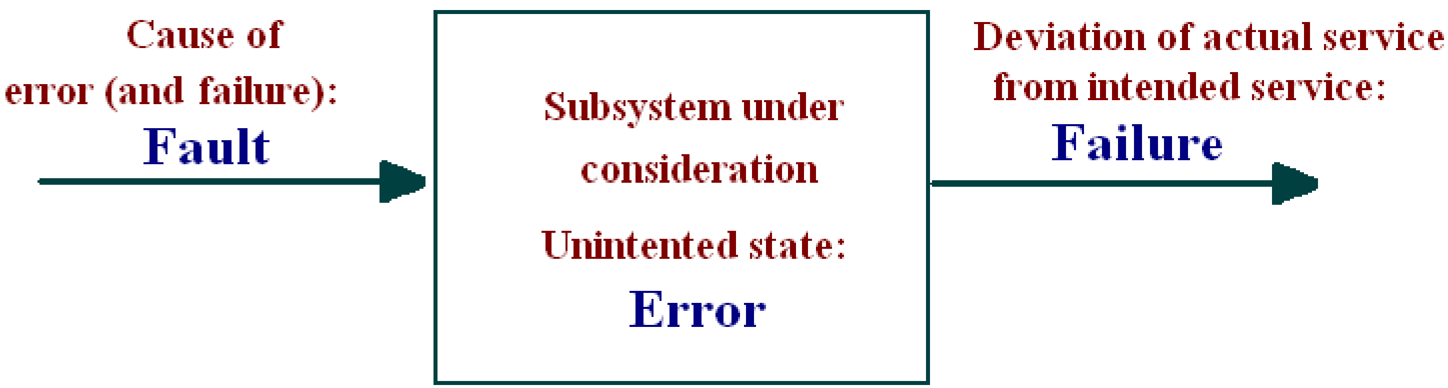
ES623 Networked Embedded Systems

Fault Tolerance

12th March 2013

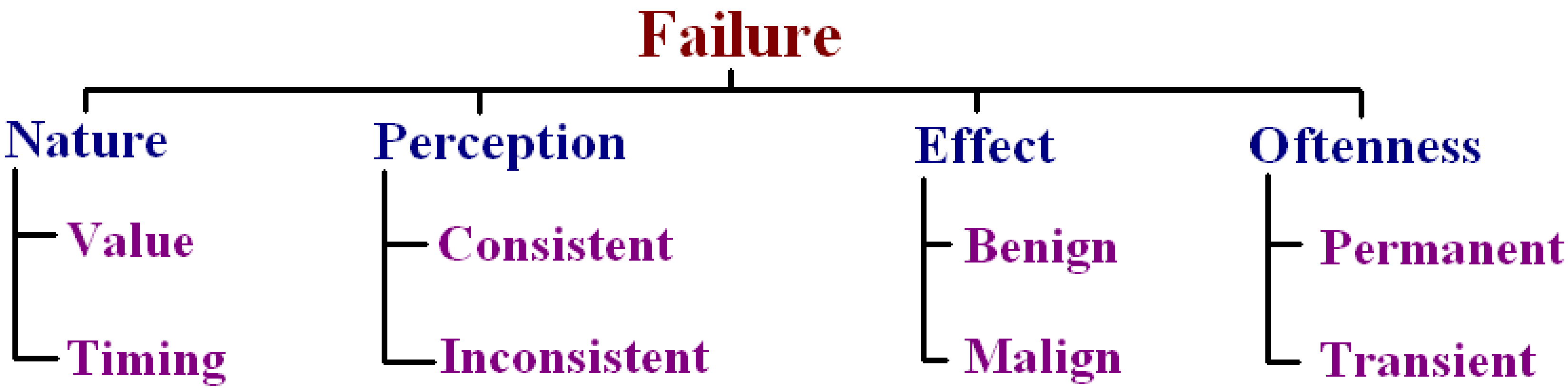
Failures, Errors and Faults

- Computer systems are installed to provide dependable service to users
- Whenever the service of the system deviates from the agreed specification of the system, the system is said to have *failed*



Failures

- Deviation between the actual service and the specified or intended service, occurring at a particular point in real time



Failure Nature

- *Value failure - an incorrect value is* presented at the system-user interface

- *Timing failure -* a value is presented outside the specified interval of real-time
  - only exist if the system specification contains information about the expected temporal behavior of the system

Failure Perception

- *Consistent failure scenario* - all users see the same (possibly wrong) result

- If a subsystem cannot deliver the correct service -*fail-silent failure*
- If a system stops operating after the first fail-silent failure, the failure is a *crash failure*
- Crash failure that is made known to the rest of the system is a *fail-stop failure*

Failure Perception

- *Inconsistent failure situation - different users may perceive different false results*
- Malicious subsystem can disturb correctly operating subsystems by showing contradictory faces of a failure - *two-faced failures*, *malicious failures*, *or Byzantine failures*
- To tolerate *k failures of a certain type, we need:*
  - (i) *k+1 components if the failures are fail-silent,*
  - (ii) *2k +1 components if the failures are fail-consistent, and*
  - (iii) *3k +1 components if the failures are malicious*

Failure Effect

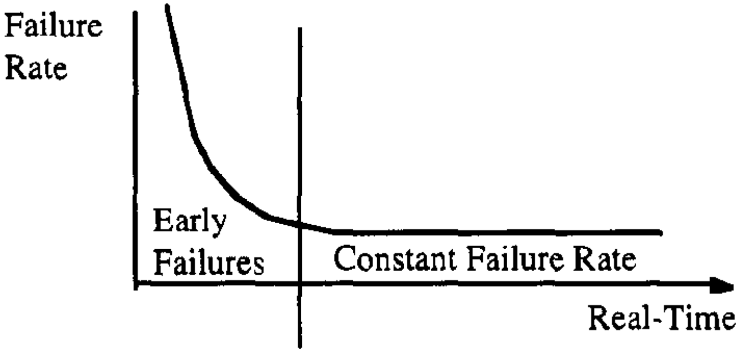
- Benign failure *can only cause* failure costs that are of the same order of magnitude as the loss of the normal utility of the system
- Malign failure can cause failure costs that are orders of magnitude higher than the normal utility of a system
  - e.g., malign failure can cause a catastrophe such as the crash of an airplane
  - *safety-critical applications*

Failure Oftenness

- Within a given time interval, a failure can occur only once or a repeated number of times.
- If it occurs only once, it is called a *single failure.*
- *Permanent failure - after which the* system ceases to provide a service until an explicit repair action has eliminated the cause of the failure.
- System continues to operate after the failure - *transient failure.*
  - *A frequently occurring transient failure is* an *intermittent failure.*

Failure Oftenness

- Permanent Failures: The failure rate (permanent failures) of a typical VLSI device changes over time.
    - The failure rate of a chip is sensitive physical parameters, such as the number of pins and the packaging



- Transient Failures: depending on the physical environment of the installation
    - common causes are electromagnetic interference (EMI), disturbances in the power supply, and high energy particles (e.g., α -particles)

Errors

- Incorrect internal state
  - wrong data element in the memory or a register of computer

- If the error exists only for a short interval of time, and disappears without an explicit repair action, it is a *transient error*

- If the error persists permanently until an explicit repair action removes it, is a *permanent error*

Transient Errors

- Predominant error class in many computer systems
- number of applications, in real-time systems, where the system behavior can be characterized by periodic duty cycles (e.g., control loops)
- cycle starts with sampling of input data, continues with computation using a given control algorithm, and terminates after output of the results to an actuator in the environment
- In such system, transient data error that occurs in one of the duty cycles cannot have direct impact on any of subsequent duty cycles
  - By design, such systems are tolerant to transient errors

Permanent Errors

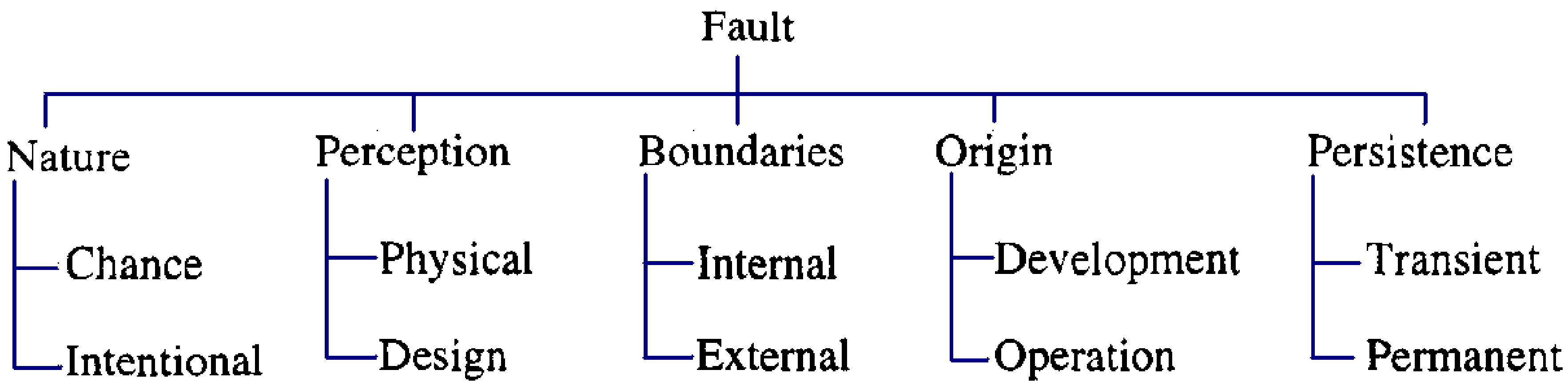
- Error that remains in the system until an explicit repair action is invoked to repair the state

- If database transaction is disturbed by transient fault, and resulting error is not immediately detected, then, a wrong value will be written into the database and remains as a permanent error in database.

- Transient fault can lead to a permanent error
  - *database erosion*

Faults

- The cause of an error, and thus the indirect cause of a failure, is a *fault*



Fault Nature

- *chance fault* - that has its origin in a chance event
  - e.g., the random break of a wire

- *intentional fault* - traced to an intentional action by someone
  - e.g., introduction of a Trojan horse by a programmer in order to break the security of a system

Fault Perception

- Fault caused by some *physical* phenomenon
  - e.g., the breakdown of a computer chip, or by an error in the design, such as a programmer's mistake or an error in the system specification
- *Design faults* in large systems are difficult to avoid, and it is nearly hopeless to diagnose them by testing

Fault Boundary

- Fault caused by a deficiency within the system or by some external disturbance
  - e.g., a lightning stroke causing spikes in the power supply line

Fault Origin

- Faults that have their origin in the incorrect development of the system
  - e.g., a wrong input by the operator

Fault Persistence

- Faults that occur only once and disappear by themselves
  - e.g., lightning stroke
- Faults that remain in a system until they are removed by an explicit repair action

Fault Tolerance

- No complex system will survive for an extended period of time without fault tolerance.

- The designer of a safety-critical system has two options to implement the necessary fault tolerance:
  - At the architecture level, *systematic fault tolerance*
  - At the application level, *application-specific fault tolerance*

Error Detection

- Goal of the fault-tolerant computing unit to detect and mask or repair errors before they show up as failures at the system-user service interface.
- Error detection requires, the information about the current state and knowledge about the intended state of a system
- This can arise from two different sources:
  - *a priori knowledge* about the intended properties of states and behaviors of the computation,
  - from the comparison of the results of two redundant computational channels.

Error Detection based on a priori knowledge

- known *a priori* about the properties of correct states and the temporal patterns of correct behavior of a computation, the more effective are the error detection techniques
- If a subsystem is to be flexible in the temporal domain and in the value domain, i.e., there are no known regularity assumptions that restrict the system behavior beforehand
- Then error detection based on *a priori knowledge is hardly possible.*

Error Detection based on a priori knowledge

- Examples of the use of error-detecting codes are:
  - parity bits and error-detecting codes in memory, CRC polynomials in data transmission, and check digits at the man-machine interface.

- In a real-time system, the worst-case execution time (WCET) of the hard real-time tasks must be known *a priori* for the calculation of the schedules

Error Detection based on redundant computations

- Performs computations twice
  - *Time redundancy*
  - *Hardware redundancy*
  - *Diverse software on same hardware*
  - *Diverse software on diverse hardware*

Error Detection based on redundant computations

Type of Redundancy	Implementation	Type of Detected Errors
Time redundancy	The same software is executed on the same hardware during two different time intervals	Errors caused by transient physical faults in the hardware with a duration of less than one execution time slot
Hardware redundancy	The same software executes on two independent hardware channels	Errors caused by transient and permanent physical hardware faults
Diverse software on the same hardware	Different software versions are executed on the same hardware during two different time intervals	Errors caused by independent software faults and transient physical faults in the hardware with a duration of less than one execution time slot
Diverse software on diverse hardware	Two different versions of the software are executed on two independent hardware channels	Errors caused by independent software faults and by transient and permanent physical hardware faults

Duplicate Execution of Tasks

- Duplicate execution of application tasks at different times is an effective technique for the detection of transient hardware errors.