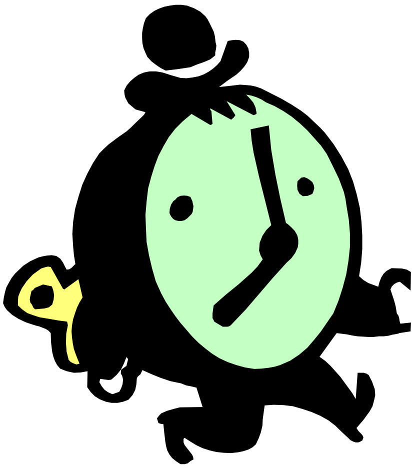
Real-Time Systems Programming

Summer-Semester 2002
Lecture 13
30 May 2002



Dependability Contd.

The 5-Minute Review Session

1) What are *real-time entities*/*representatives/images*? What may cause them to differ?
2) What is a rule of thumb for selecting a *sampling rate*?
3) How can we compensate a *sampling delay*? How can we compensate a *sampling jitter*?
4) What is *temporal accuracy*?
5) What is the difference between *parametric* and *phase-sensitive* RT images?

Overview

1) Failures
2) Errors
3) Faults
4) Fault Prevention vs. Fault Tolerance

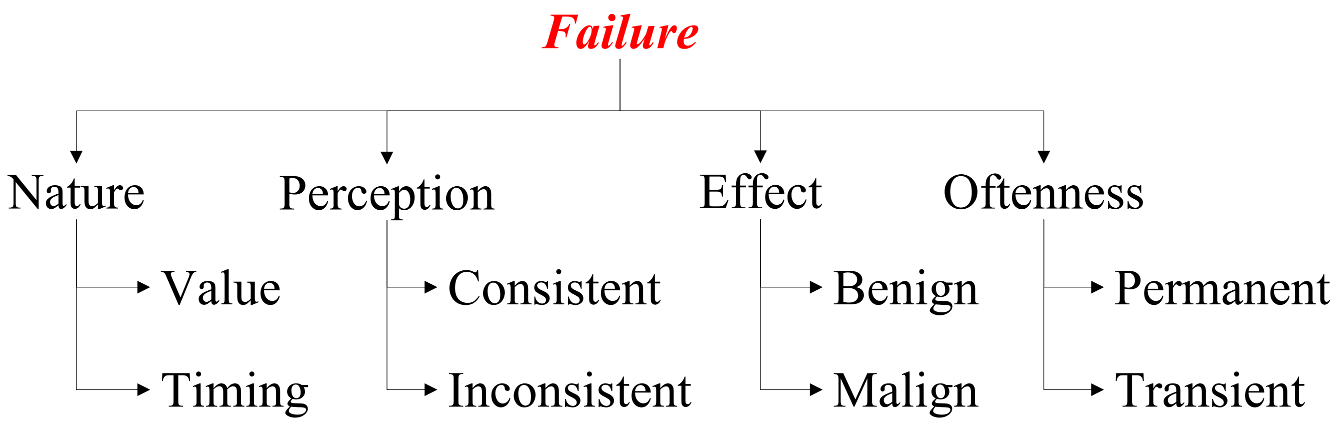
Where are we?

1) ***Failures***
   - ***Nature***
   - ***Perception***
   - ***Effect***
   - ***Oftenness***
   - ***Origins***
2) Errors
3) Faults
4) Fault Prevention vs. Fault Tolerance

Recall: Fault, Error, Failure

- ***Failure*** (***"Ausfall"***):
  - Deviation of actual service from specified service (***external*** state)
    - Control surface on wing moves erroneously
    - Airbag does not ignite
- ***Error*** (***"Fehlzustand"***):
  - Unintended (***internal***) system state
    - Short circuit (excessive current, low voltage)
    - Variable out of range
- ***Fault*** (***"Fehler"***):
  - Cause of an error
    - Broken isolator, software bug
    - Specification fault

Classification of Failures

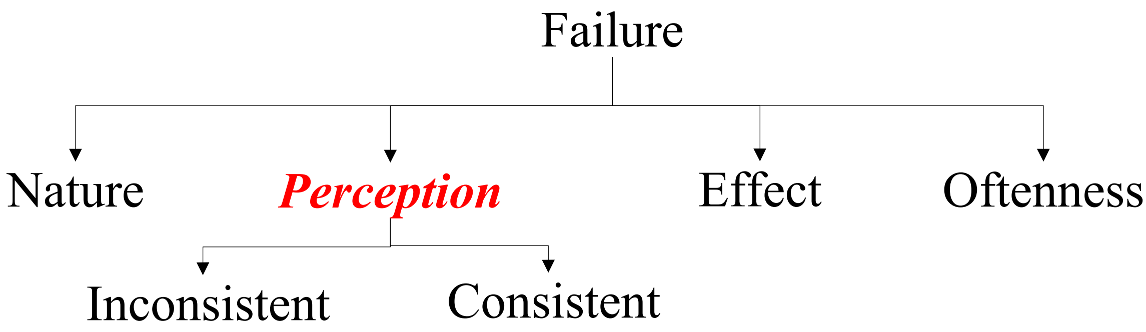


Failure Nature

- Failure
  - ***Nature***
    - Value Domain
      - Constraint Failure (Detectable)
      - Value Failure (Possibly undetectable)
    - Timing Domain
      - Commission
      - Early
      - Late
      - Omission
  - Perception
  - Effect
  - Oftenness

- ***Arbitrary failures:***
  - Combinations of value and timing domain failures

Failure Perception

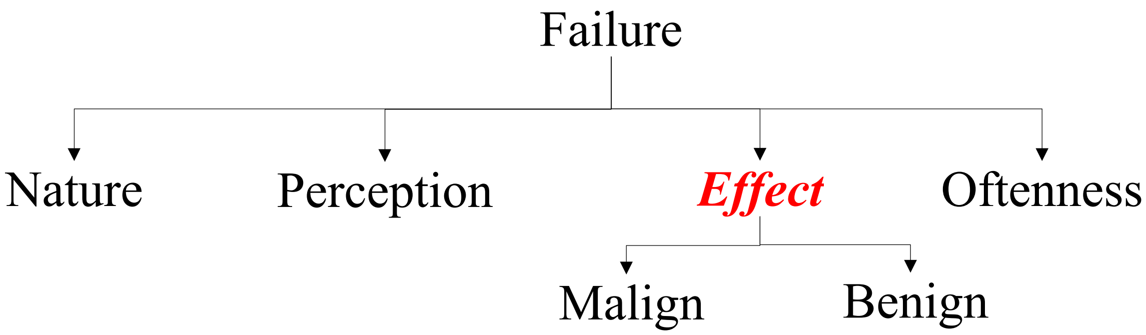


- In system with more than one user:
  - ***Consistent*** failures:
    - Perceptions of the users are the same
  - ***Inconsistent*** failures:
    - Perceptions are different
    - Also referred to as ***two-faced*** failures, ***malicious*** failures, or ***Byzantine*** failures

System Classification

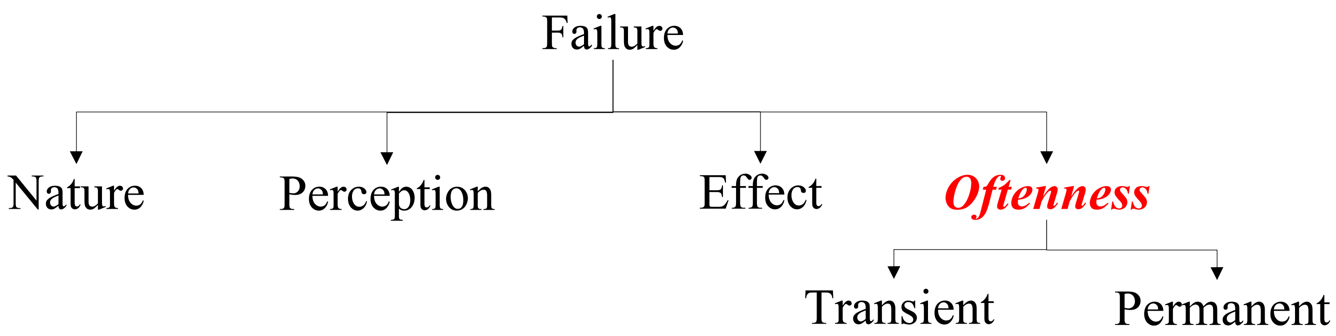
- ***Given:*** consistent failure perception
- ***Fail silent***: System produces either correct results (both in value and time domains) or no results at all
- ***Fail crash***: Fail-silent system that stops operating after the first failure
- ***Fail stop***: Fail-crash system that makes its failure known to other systems
- ***Fail (un-)controlled***: System that fails in a(n) (un-) controlled manner
- ***Fail-never***: System that always provides correct services in both the timing and value domains

Failure Effect



- The classification of a failure effect depends on the characeristics of the controlled application
- ***Safety-critical*** applications are those where a malign failure can occur

Failure Oftenness

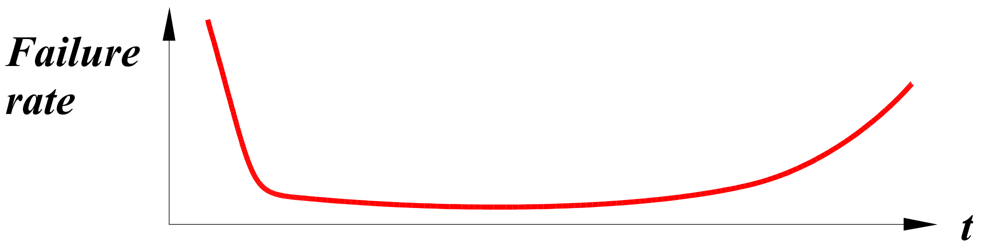


- ***Single failure***
  - Failure occurs only once within a given time interval
- ***Permanent failure***
  - System ceases to provide service until repair
- ***Transient failure***
  - System continues service
- ***Intermittent failure***
  - Frequently occurring transient failure

- Example of permanent failure:
  - Broken wire
- Example of intermittent failure:
  - Heat-sensitive hardware device

Permanent Failures

- A typical VLSI device failure rate develops according to the “bathtub pattern”:
  - A relatively high failure rate for the first few hundred hours of operation (***burn-in***)
  - After that, stabilization at about 10-100 FIT (= Failures per 10^9 hrs – MTTF of about 115 Kyrs)
  - At some point, an increased failure rate again (***aging***)



Preventive Maintenance

- Failure rate of a VLSI chip
  - Depends mainly on physical parameters (pins, packaging)
  - Not very sensitive to the number of transistors
- ***Preventive maintenance***
  - Exchange of components before they fail
  - Limits effects of aging
- *If there is no aging, then there is no point in preventive maintenance!*

Transient Failures

- Transient chip failure rate
  - Can be 10 – 100 000 x permanent failure rate
  - Depends on physical environment
- Most common causes are
  - Electromagnetic interferences (EMI)
  - Power supply glitches
  - High-energy particles (e.g., α-particles)
- Example from radar monitoring [Gebman et al. 1988]:
  - Malfunctions noticed every 6 flight hrs
  - Maintenance request every 31 hrs
  - *Only every 3rd failure could be reproduced!*

Origins of Failure

- Rule of thumb (JPL data):
  - 1 major fault every 3 pages of requirements
  - 1 major fault every 21 pages of code
- Fault statistics for some NASA space projects:
  - Coding faults: 6% of overall faults (!!!)
  - Function faults: 71% (due to requirements/design problems)
  - Interface faults: 23% (due to poor comm. between teams)
- Observation:
  - Most severe faults are introduced early but are detected late! *(often during system integration)*

- These statistics were kindly provided by Gerald Luettgen (University of Sheffield)

Origins of Failure

- Results of one study on large information systems (Tandem):
  - **>40%** of failures due to ***human operator faults***
  - **25%** caused by ***software faults***
  - Large contribution by ***environmental factors***
    - Power outages
    - Fires, floods
  - ***Smallest contributor: (random) hardware faults***
- ***One of the lessons:***
  - Need not only hw fault tolerance, *but also sw fault tolerance!*

- J. Gray, "Why do Computers Stop and What Can be done About It?," *Proceedings of the 5th IEEE Symposium on Reliability in Distributed Software and Database Systems*, Los Angeles, USA, p. 3-12, 1986

Where are we?

1) Failures
2) ***Errors***
   - ***Classification***
3) Faults
4) Fault Prevention vs. Fault Tolerance

Errors

- Most controller failures can be traced to an incorrect internal state – i.e., a wrong data element
- Similarly to failures, we can classify errors as
  - ***Transient errors***: exists only for short interval, disappears again without explicit repair action
  - ***Permanent errors***: require explicit repair
- ***Fault-tolerant architecture***
  - Every error confined to an ***error containment region***
  - This avoids ***error propagation***
- ***Error detection interfaces***
  - Protect boundaries of error containment regions

Transient Errors

- Errors are predominantly transient
- Typical, ***simple control cycle*** structure:
  - Read inputs (sensors)
  - Compute reaction
  - Write outputs (actuators)
- Wrong input on one cycle does not affect next cycle
- Typically, each cycle can release only a finite amount of energy
  - Results in ***transient error tolerant*** design

Permanent Errors

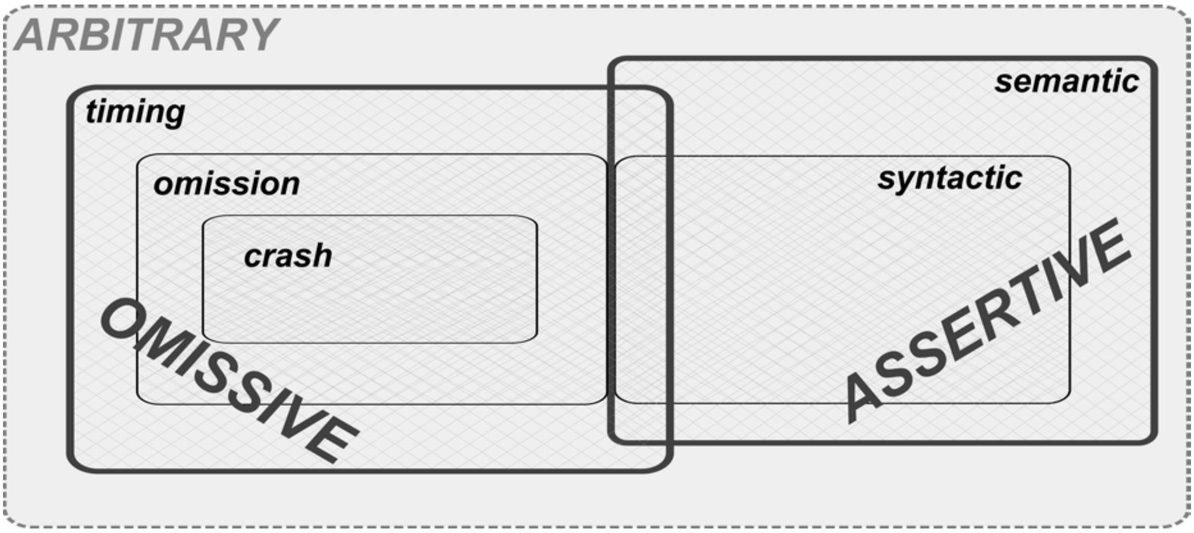
- ***Example***: database
  - Maintains large state
  - Any introduced error is likely to be permanent – i.e., requires an explicit correction
  - Without corrections, ***data base erosion*** occurs

Where are we?

1) Failures
2) Errors
3) ***Faults***
   - ***Models***
   - ***Classification***
4) Fault Prevention vs. Fault Tolerance

Interaction Fault Models

- First step to building fault-tolerant system:
  - Define a ***fault model***



[Veríssimo and Rodrigues 2001]

- *Omissive Faults:* A component does not perform some interaction when specified to
- *Assertive Faults:* A component does perform some interaction when not specified to
  - *Syntactic Faults:* construction of interaction is incorrect (e.g., Temp = “+ab”)
  - *Semantic Faults:* meaning conveyed by interaction is incorrect (e.g., Temp = “-99”)

Fault Classification

Fault

- Nature
  - Chance
  - Intentional
- Perception
  - Physical
  - Design
- Boundaries
  - Internal
  - External
- Origin
  - Development
  - Operation
- Persistence
  - Transient
  - Permanent

- See [Laprie 1992] for more details

Where are we?

1) Failures
2) Errors
3) Faults
4) ***Fault Prevention vs. Fault Tolerance***
   - ***Hardware fault avoidance***
   - ***Software fault avoidance***

Approaches to Achieving Reliable Systems

- ***Fault prevention***
  - Attempts to eliminate any possibility of faults creeping into a system before it goes operational
  - ***Fault avoidance***
    - Limit introduction of faults during system construction
  - ***Fault removal***
    - Find and remove the causes of errors
- ***Fault tolerance***
  - Enables system to continue functioning even in the presence of faults
- Both approaches attempt to produces systems which have well-defined failure modes

Hardware Fault Avoidance

- Use of the most ***reliable components*** within the given cost and performance constraints
- Use of ***thoroughly-refined techniques*** for interconnection of components and assembly of subsystems
  - Plugs and soldered connections are often the weakest points
- Packaging the hardware to screen out expected forms of ***interference***
  - E.g. EMI shielding, Single Event Upset (SEU) resistence in avionics and space applications

Software Fault Avoidance

- Software
  - Does not deteriorate (by itself) with use
  - Often much more complex than hw counterparts
  - Virtually impossible to design fault-free
- ***Banana software approach***
  - "*Ripes at the customer*"
  - Not untypical in consumer and business sw
  - With RT systems usually ***not*** an option

Software Fault Avoidance

- SW can be improved by
  - ***Rigorous***, if not formal, specification of ***requirements***
  - Use of *proven design methodologies*
  - Use of languages with
    - *data abstraction*
    - *modularity*
  - Use of *sw engineering environments* to manage ***complexity***

Summary

- We distinguish between *fault*, *error*, and *failure*
- Among the predominant causes are ***human operator error*** and ***software faults***
- *Hardware faults* are less common causes
- In simple control structures, transient input faults may only lead to transient system failures

s

- Reliability
- Safety
- Confidentiality
- Integrity
- Maintainability

- Fault Prevention
- Fault Tolerance
- Fault Removal
- Fault Forecasting

ents

- Faults
- Errors
- Failures