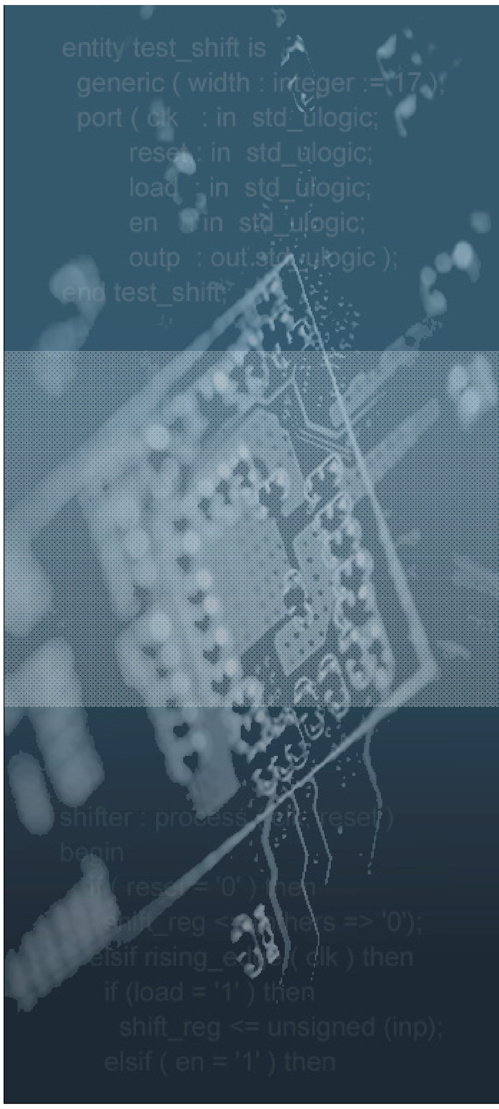
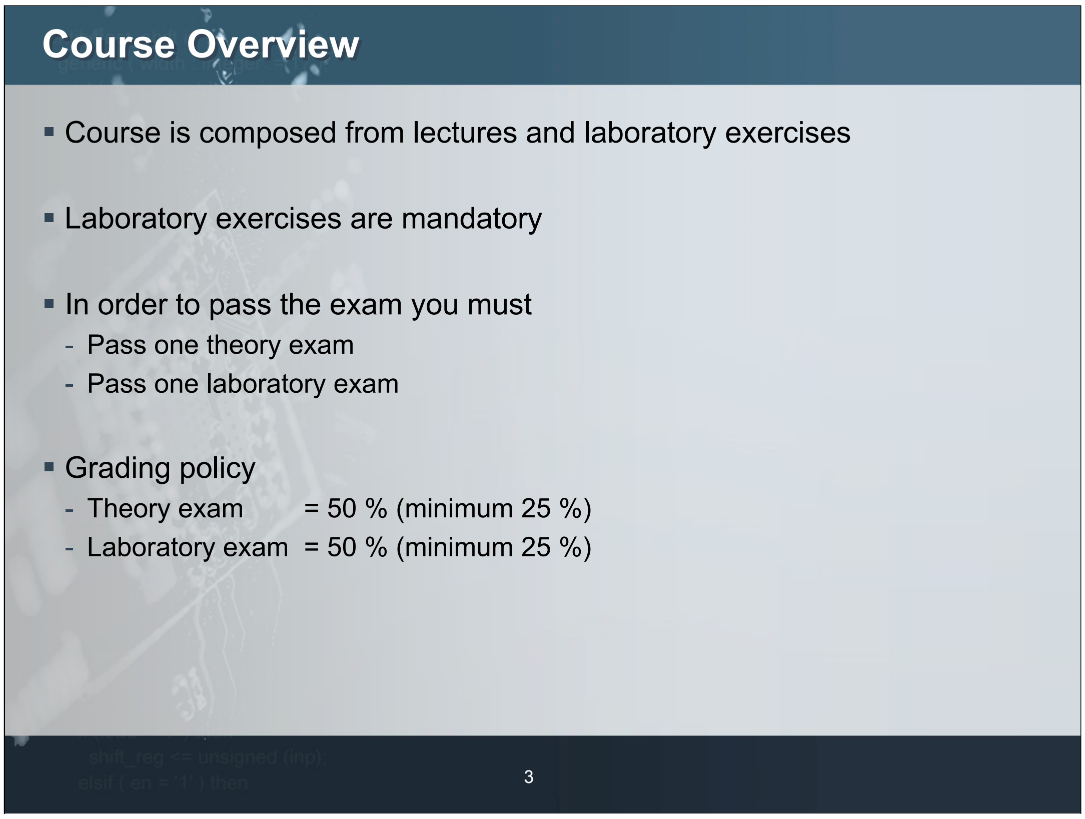


Digitalni sistemi otporni na otkaz

Predavanje I

Lecture Content

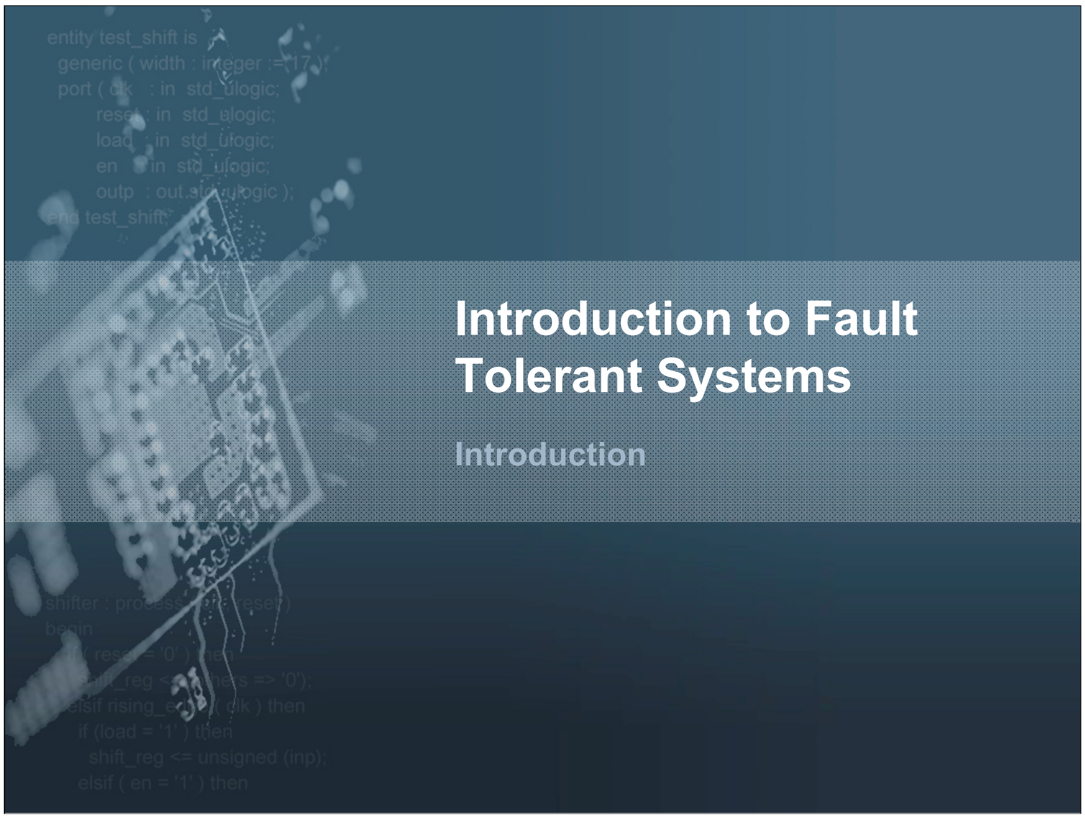
- Course overview
- Introduction
- Fundamentals of Dependability
  - Dependability Attributes
  - Dependability Impairments
  - Dependability Means



Course Overview

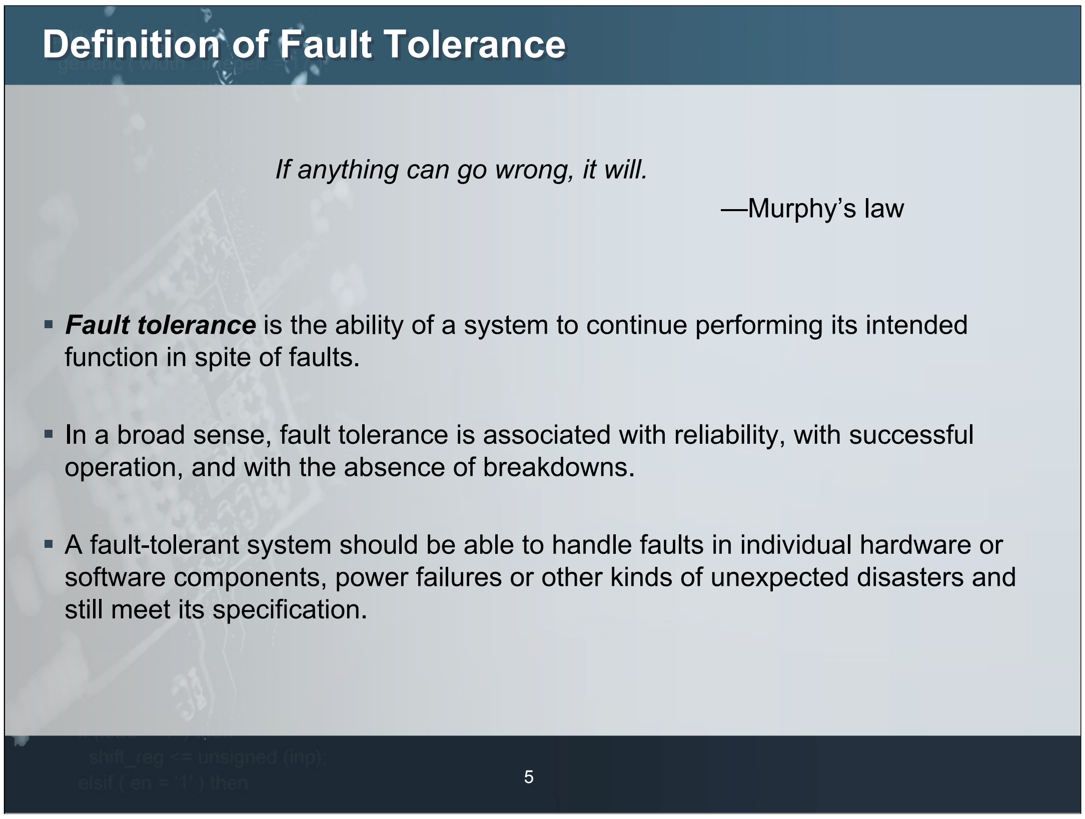
- Course is composed from lectures and laboratory exercises
- Laboratory exercises are mandatory
- In order to pass the exam you must
  - Pass one theory exam
  - Pass one laboratory exam
- Grading policy
  - Theory exam = 50 % (minimum 25 %)
  - Laboratory exam = 50 % (minimum 25 %)

3



Introduction to Fault Tolerant Systems

Introduction



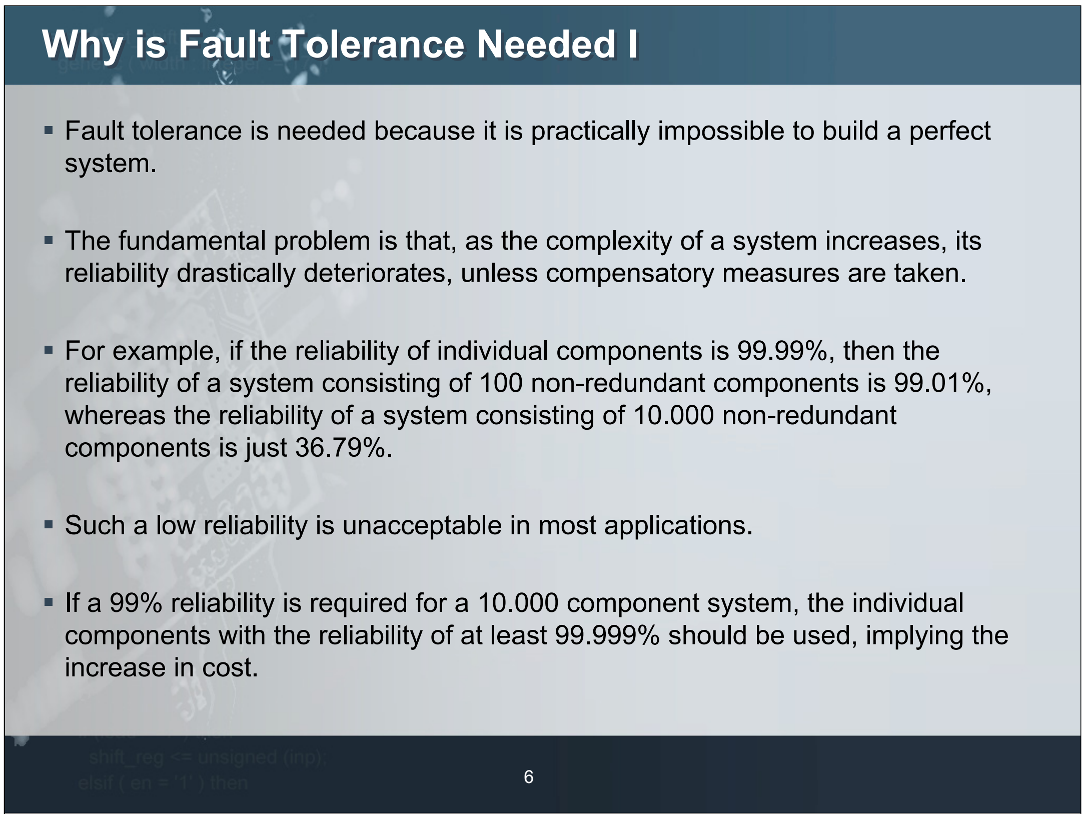
Definition of Fault Tolerance

If anything can go wrong, it will.

—Murphy's law

- ***Fault tolerance*** is the ability of a system to continue performing its intended function in spite of faults.
- In a broad sense, fault tolerance is associated with reliability, with successful operation, and with the absence of breakdowns.
- A fault-tolerant system should be able to handle faults in individual hardware or software components, power failures or other kinds of unexpected disasters and still meet its specification.

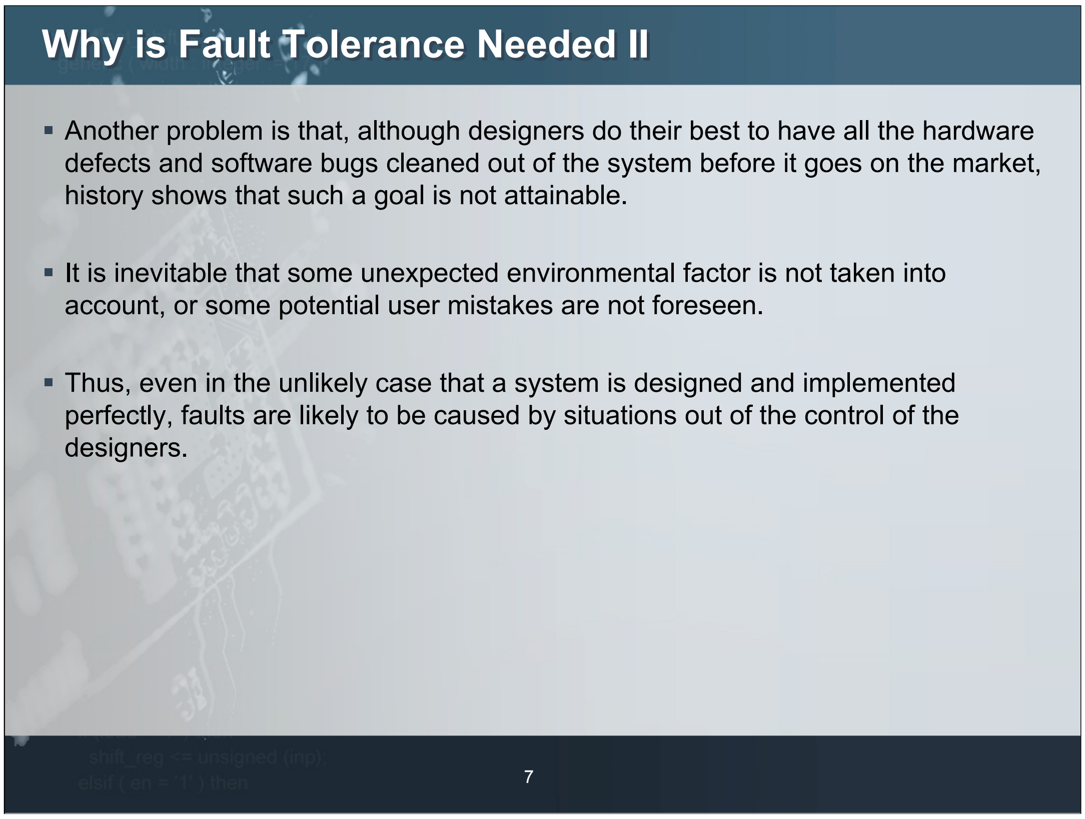
5



Why is Fault Tolerance Needed I

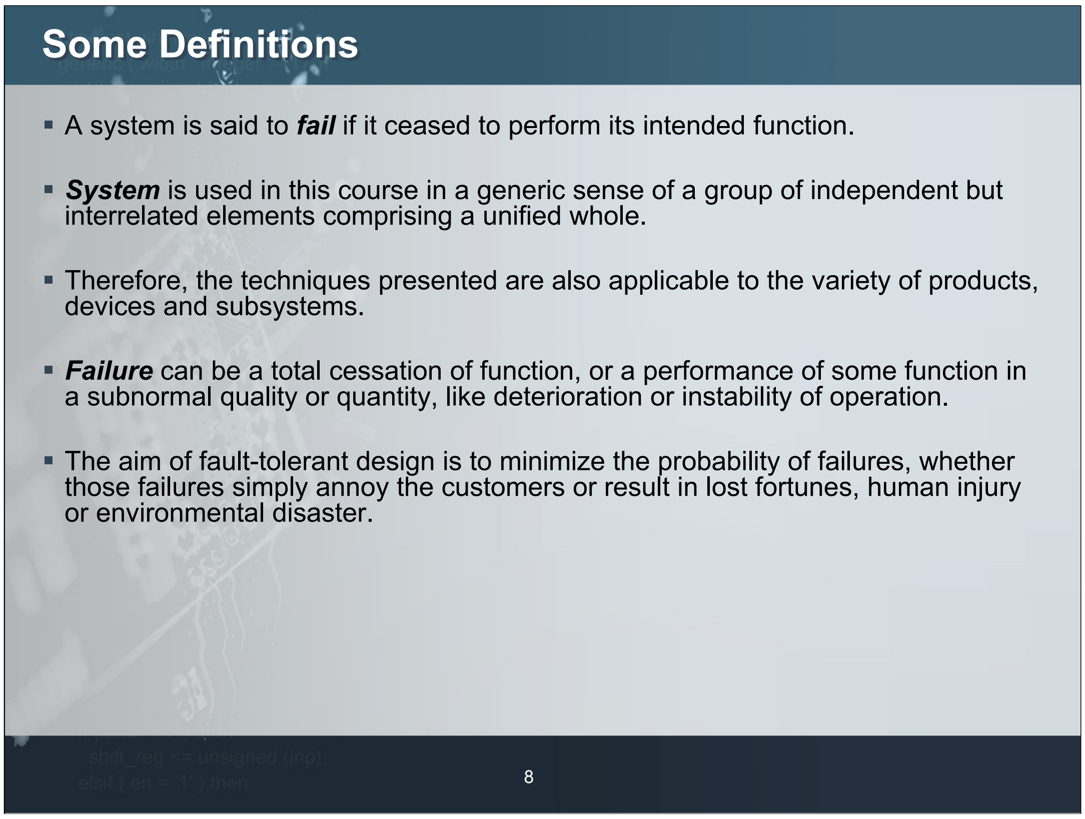
- Fault tolerance is needed because it is practically impossible to build a perfect system.
- The fundamental problem is that, as the complexity of a system increases, its reliability drastically deteriorates, unless compensatory measures are taken.
- For example, if the reliability of individual components is 99.99%, then the reliability of a system consisting of 100 non-redundant components is 99.01%, whereas the reliability of a system consisting of 10.000 non-redundant components is just 36.79%.
- Such a low reliability is unacceptable in most applications.
- If a 99% reliability is required for a 10.000 component system, the individual components with the reliability of at least 99.999% should be used, implying the increase in cost.

6



Why is Fault Tolerance Needed II

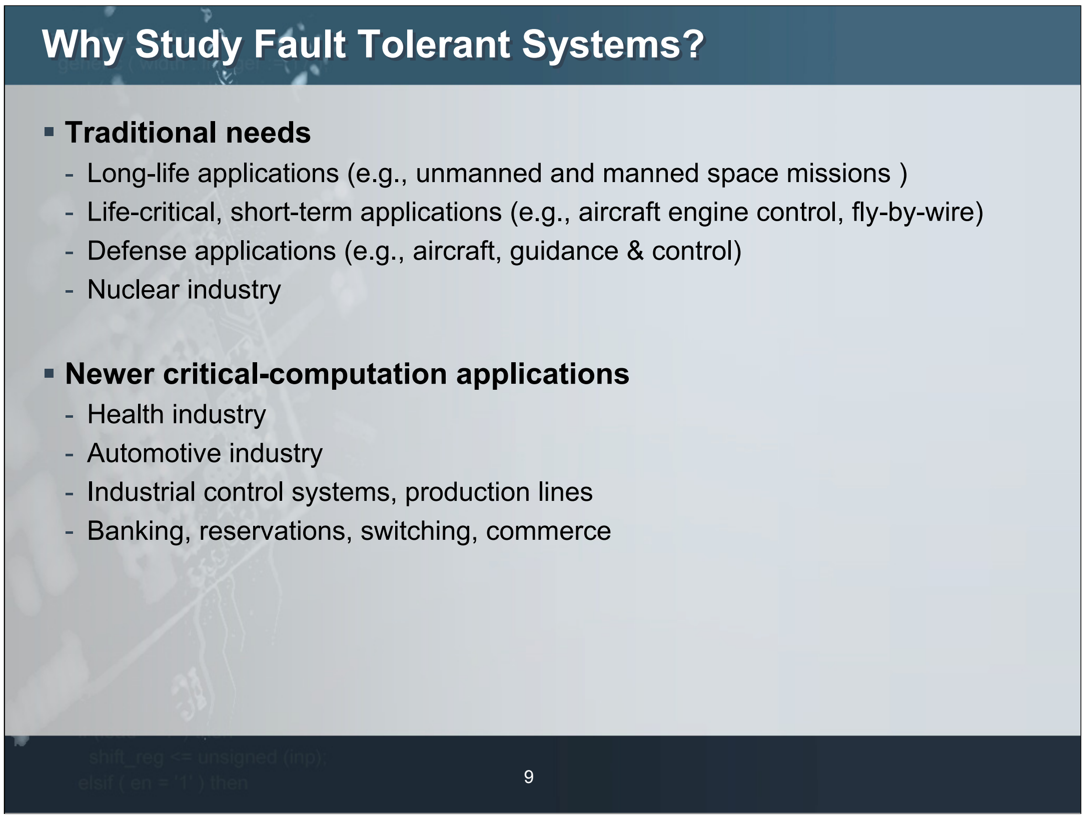
- Another problem is that, although designers do their best to have all the hardware defects and software bugs cleaned out of the system before it goes on the market, history shows that such a goal is not attainable.
- It is inevitable that some unexpected environmental factor is not taken into account, or some potential user mistakes are not foreseen.
- Thus, even in the unlikely case that a system is designed and implemented perfectly, faults are likely to be caused by situations out of the control of the designers.



Some Definitions

- A system is said to ***fail*** if it ceased to perform its intended function.
- ***System*** is used in this course in a generic sense of a group of independent but interrelated elements comprising a unified whole.
- Therefore, the techniques presented are also applicable to the variety of products, devices and subsystems.
- ***Failure*** can be a total cessation of function, or a performance of some function in a subnormal quality or quantity, like deterioration or instability of operation.
- The aim of fault-tolerant design is to minimize the probability of failures, whether those failures simply annoy the customers or result in lost fortunes, human injury or environmental disaster.

8

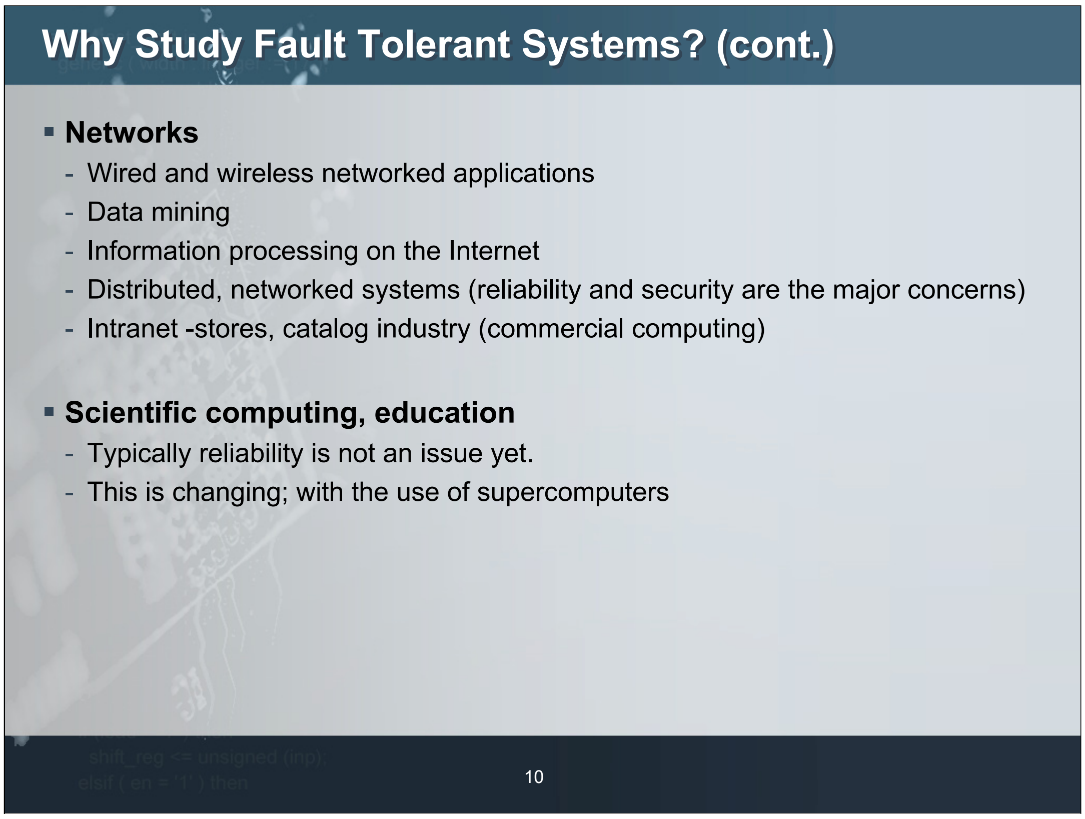


Why Study Fault Tolerant Systems?

- **Traditional needs**
  - Long-life applications (e.g., unmanned and manned space missions)
  - Life-critical, short-term applications (e.g., aircraft engine control, fly-by-wire)
  - Defense applications (e.g., aircraft, guidance & control)
  - Nuclear industry

- **Newer critical-computation applications**
  - Health industry
  - Automotive industry
  - Industrial control systems, production lines
  - Banking, reservations, switching, commerce

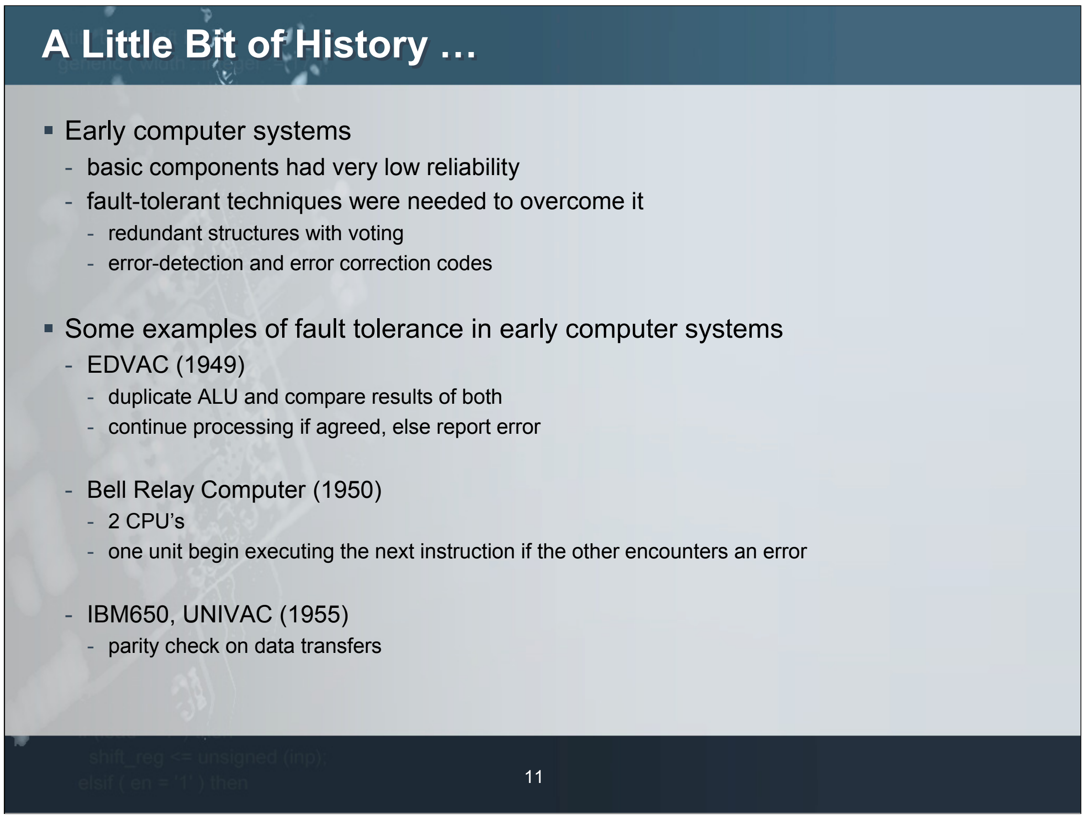
9



Why Study Fault Tolerant Systems? (cont.)

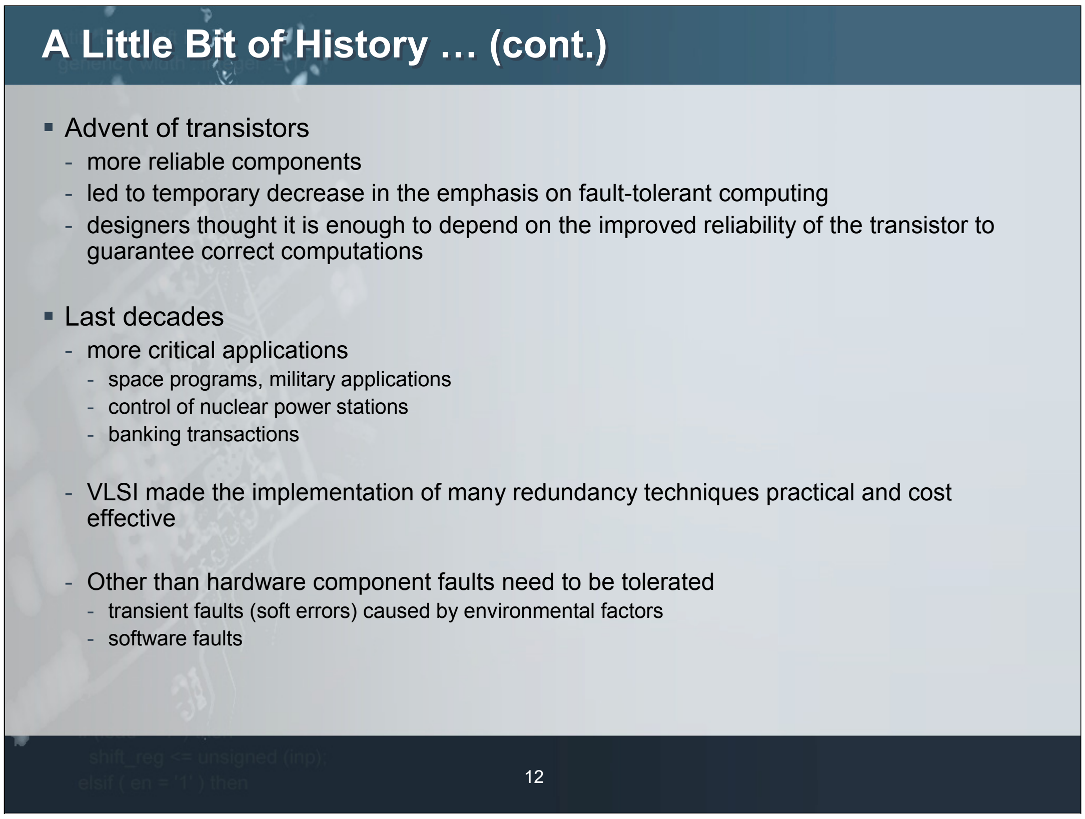
- **Networks**
  - Wired and wireless networked applications
  - Data mining
  - Information processing on the Internet
  - Distributed, networked systems (reliability and security are the major concerns)
  - Intranet -stores, catalog industry (commercial computing)
- **Scientific computing, education**
  - Typically reliability is not an issue yet.
  - This is changing; with the use of supercomputers

10



A Little Bit of History …

- Early computer systems
  - basic components had very low reliability
  - fault-tolerant techniques were needed to overcome it
    - redundant structures with voting
    - error-detection and error correction codes
- Some examples of fault tolerance in early computer systems
  - EDVAC (1949)
    - duplicate ALU and compare results of both
    - continue processing if agreed, else report error
  - Bell Relay Computer (1950)
    - 2 CPU's
    - one unit begin executing the next instruction if the other encounters an error
  - IBM650, UNIVAC (1955)
    - parity check on data transfers



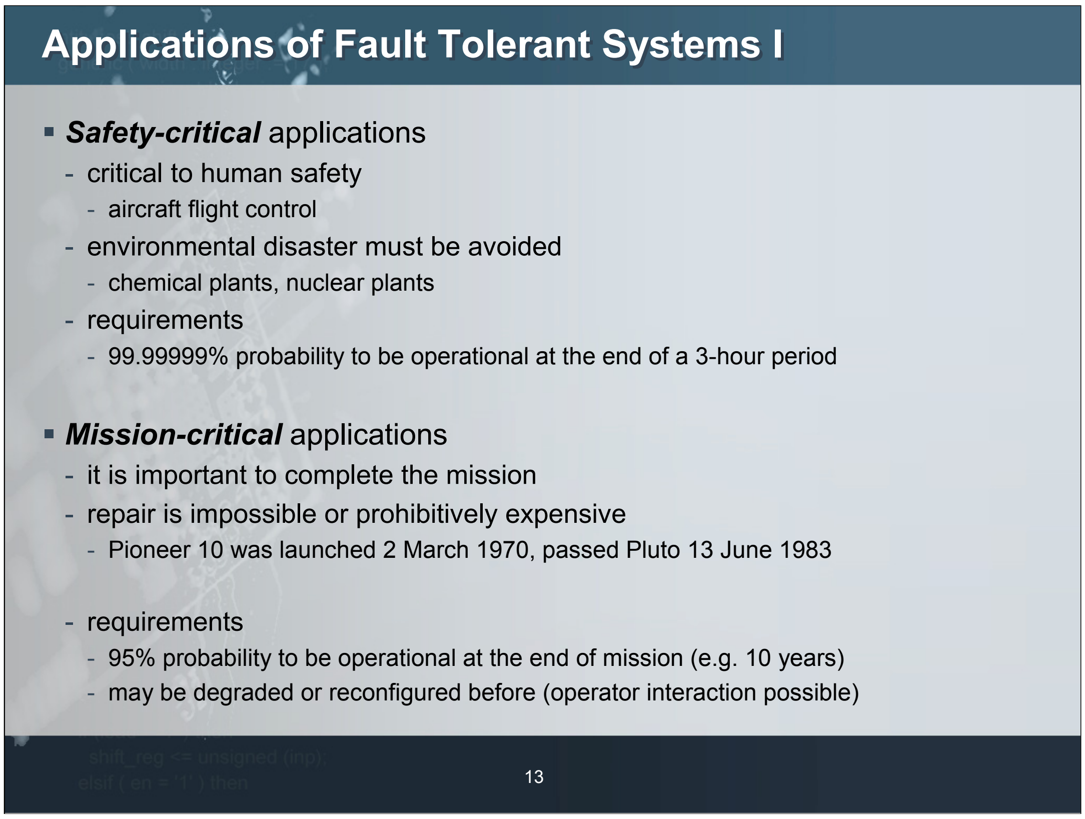
A Little Bit of History … (cont.)

- Advent of transistors
  - more reliable components
  - led to temporary decrease in the emphasis on fault-tolerant computing
  - designers thought it is enough to depend on the improved reliability of the transistor to guarantee correct computations
- Last decades
  - more critical applications
    - space programs, military applications
    - control of nuclear power stations
    - banking transactions
  - VLSI made the implementation of many redundancy techniques practical and cost effective
  - Other than hardware component faults need to be tolerated
    - transient faults (soft errors) caused by environmental factors
    - software faults

12

As complexity of systems grew, a need to tolerate other than hardware component faults has aroused. The rapid development of real-time computing applications that started around the mid-1990s, especially the demand for software embedded intelligent devices, made software fault tolerance a pressing issue. Software systems offer compact design, rich functionality and competitive cost. Instead of implementing a given functionality in hardware, the design is done by writing a set of instructions accomplishing the desired tasks and loading them into a processor. If changes in the functionality are needed, the instructions can be modified instead of building a different physical device.

An inevitable related problem is that the design of a system is performed by someone who is not an expert in that system. For example, the autopilot expert decides how the device should work, and then provides the information to a software engineer, who implements the design. This extra communication step is the source of many faults in software today. The software is doing what the software engineer thought it should do, rather than what the original design engineer required. Nearly all the serious accidents in which software has been involved in the past can be traced to this origin.

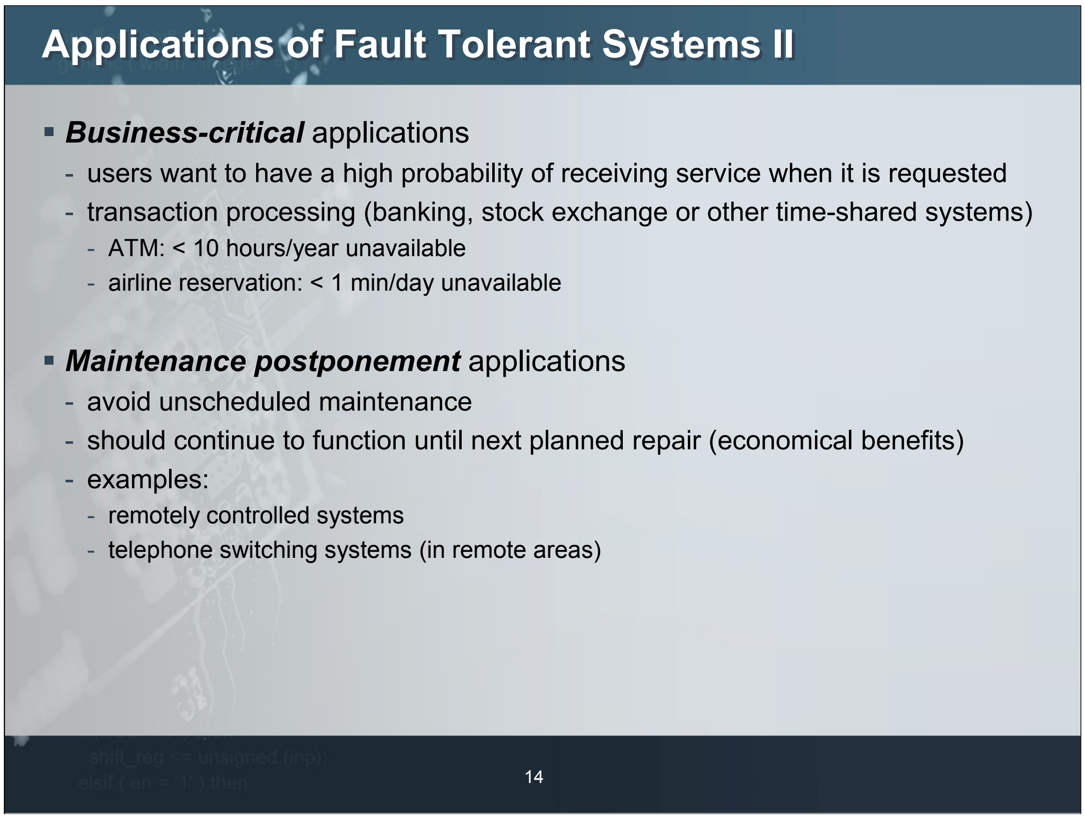


Applications of Fault Tolerant Systems I

- ***Safety-critical*** applications
  - critical to human safety
    - aircraft flight control
  - environmental disaster must be avoided
    - chemical plants, nuclear plants
  - requirements
    - 99.99999% probability to be operational at the end of a 3-hour period
- ***Mission-critical*** applications
  - it is important to complete the mission
  - repair is impossible or prohibitively expensive
    - Pioneer 10 was launched 2 March 1970, passed Pluto 13 June 1983
  - requirements
    - 95% probability to be operational at the end of mission (e.g. 10 years)
    - may be degraded or reconfigured before (operator interaction possible)

13

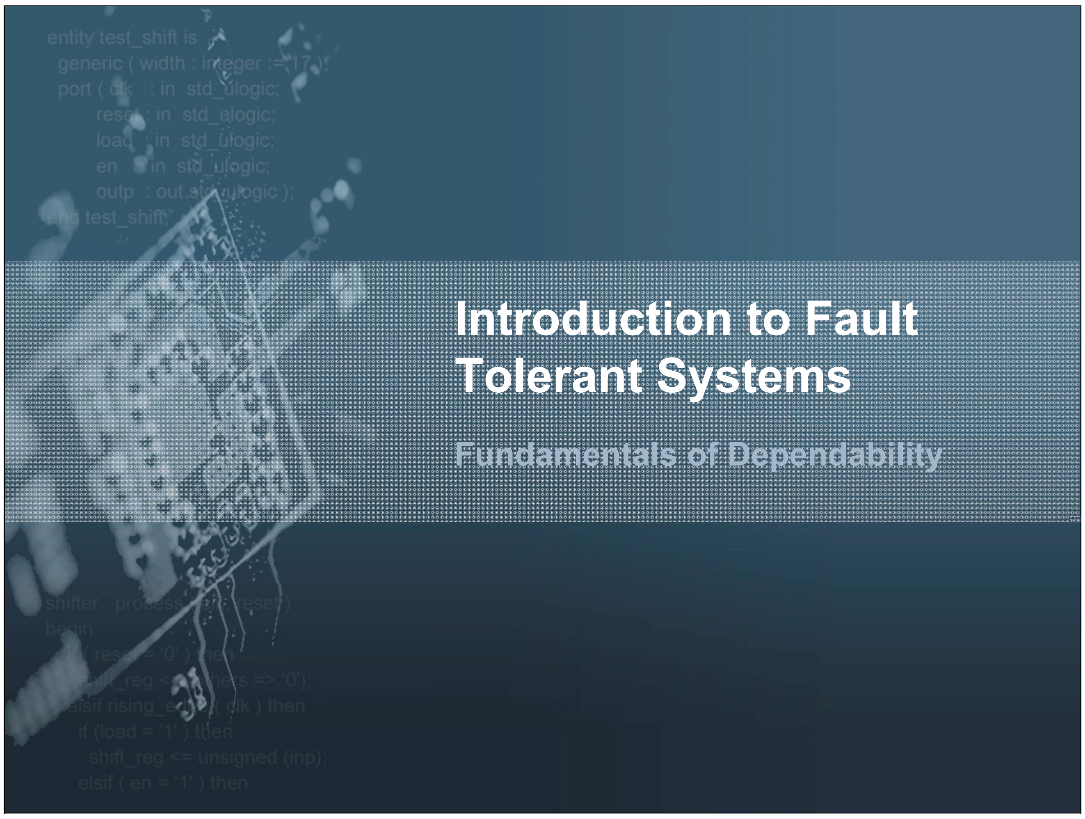
Safety-critical applications are those where loss of life or environmental disaster must be avoided. Examples are nuclear power plant control systems, computer-controlled radiation therapy machines or heart pace-makers, military radar systems. *Mission-critical* applications stress mission completion, as in case of an airplane or a spacecraft. *Business-critical* are those in which keeping a business operating is an issue. Examples are bank and stock exchange's automated trading system, web servers, e-commerce.



Applications of Fault Tolerant Systems II

- ***Business-critical*** applications
  - users want to have a high probability of receiving service when it is requested
  - transaction processing (banking, stock exchange or other time-shared systems)
    - ATM: < 10 hours/year unavailable
    - airline reservation: < 1 min/day unavailable
- ***Maintenance postponement*** applications
  - avoid unscheduled maintenance
  - should continue to function until next planned repair (economical benefits)
  - examples:
    - remotely controlled systems
    - telephone switching systems (in remote areas)

14



Introduction to Fault Tolerant Systems

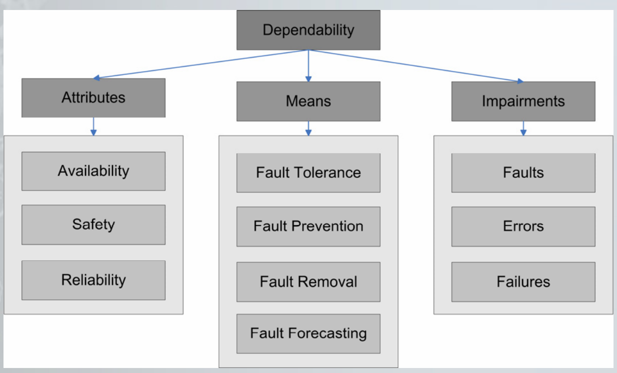
Fundamentals of Dependability

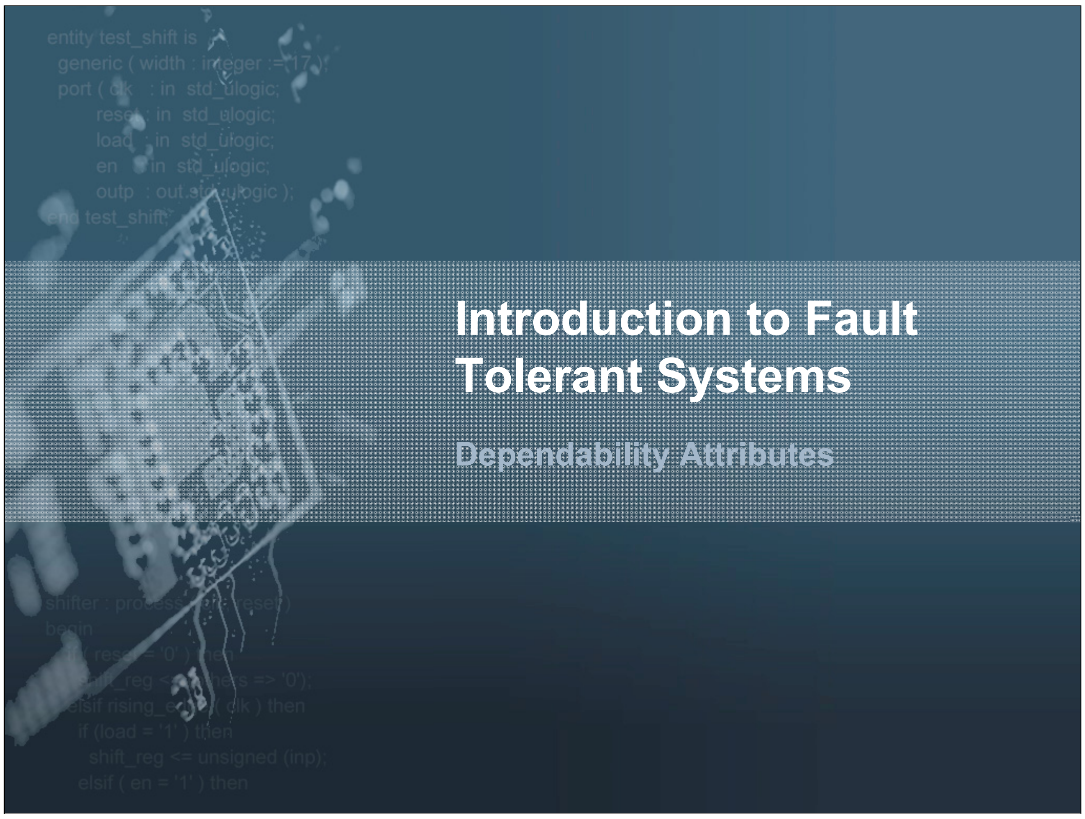
Introduction I

- The ultimate goal of fault tolerance is the development of a dependable system.
- In a broad term, ***dependability*** is the ability of a system to deliver its intended level of service to its users.
- As computer systems become relied upon by society more and more, the dependability of these systems becomes a critical issue.
- In airplanes, chemical plants, heart pace-makers or other safety critical applications, a system failure can cost people's lives or environmental disaster.

Introduction II

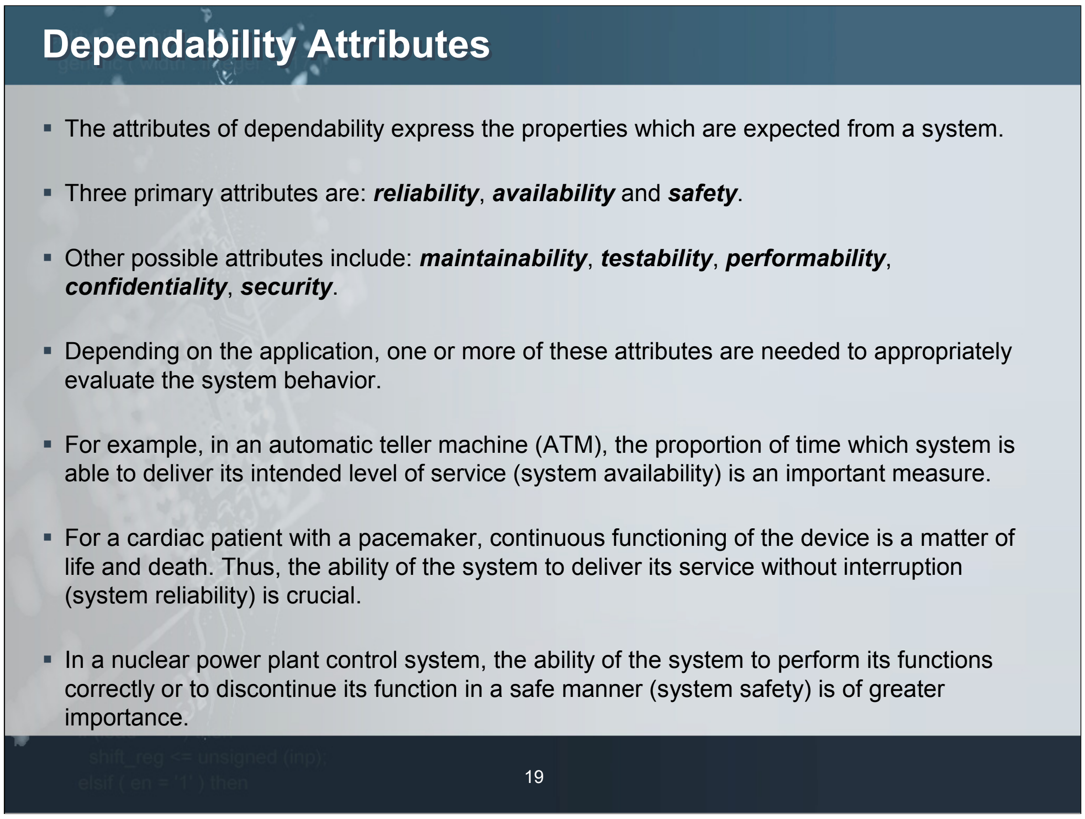
- Next, we will study three fundamental characteristics of dependability: attributes, impairment and means.
  - Dependability ***attributes*** describe the properties which are required from a system.
  - Dependability ***impairments*** express the reasons for a system to cease to perform its function or, in other words, the threats to dependability.
  - Dependability ***means*** are the methods and techniques enabling the development of a dependable computing system.





Introduction to Fault Tolerant Systems

Dependability Attributes



Dependability Attributes

- The attributes of dependability express the properties which are expected from a system.
- Three primary attributes are: ***reliability***, ***availability*** and ***safety***.
- Other possible attributes include: ***maintainability***, ***testability***, ***performability***, ***confidentiality***, ***security***.
- Depending on the application, one or more of these attributes are needed to appropriately evaluate the system behavior.
- For example, in an automatic teller machine (ATM), the proportion of time which system is able to deliver its intended level of service (system availability) is an important measure.
- For a cardiac patient with a pacemaker, continuous functioning of the device is a matter of life and death. Thus, the ability of the system to deliver its service without interruption (system reliability) is crucial.
- In a nuclear power plant control system, the ability of the system to perform its functions correctly or to discontinue its function in a safe manner (system safety) is of greater importance.

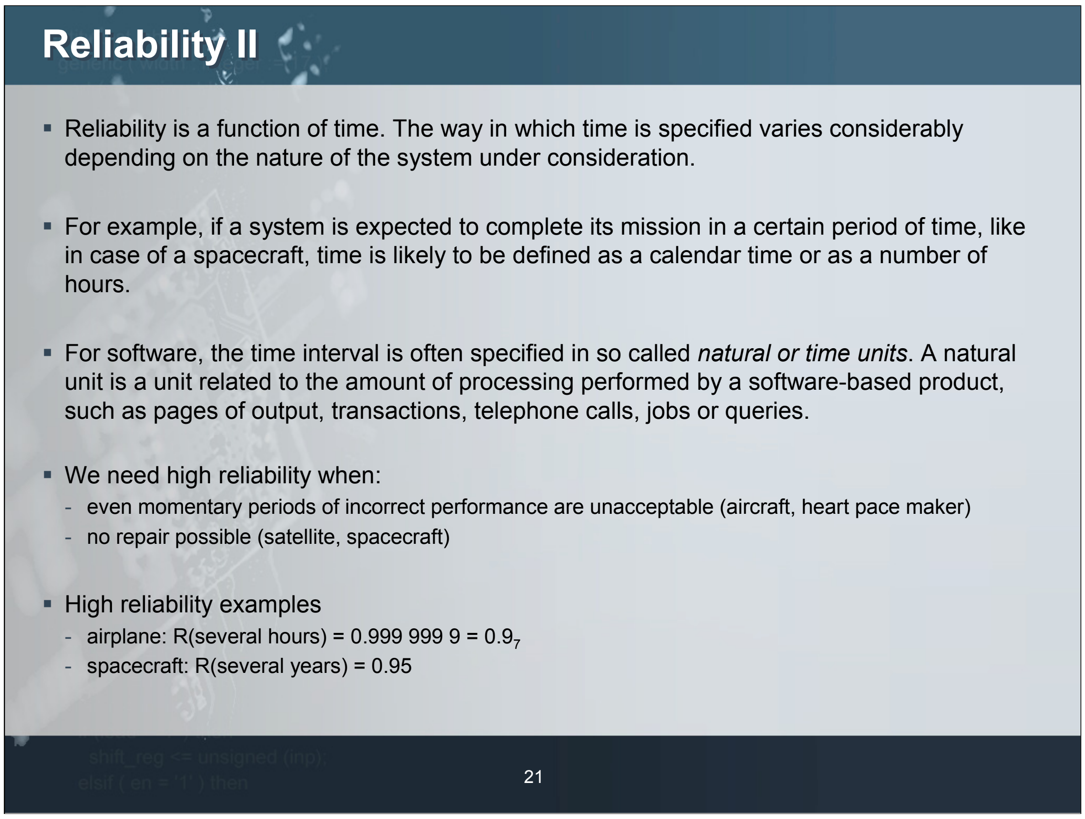
19

Reliability I

- *Reliability R*(*t)* of a system at time *t* is the probability that the system operates without failure in the interval [0, *t*] given that the system was performing correctly at time 0.

- Reliability is a measure of the continuous delivery of correct service. High reliability is required in situations when a system is expected to operate without interruptions, as in the case of a pacemaker, or when maintenance cannot be performed because the system cannot be accessed.

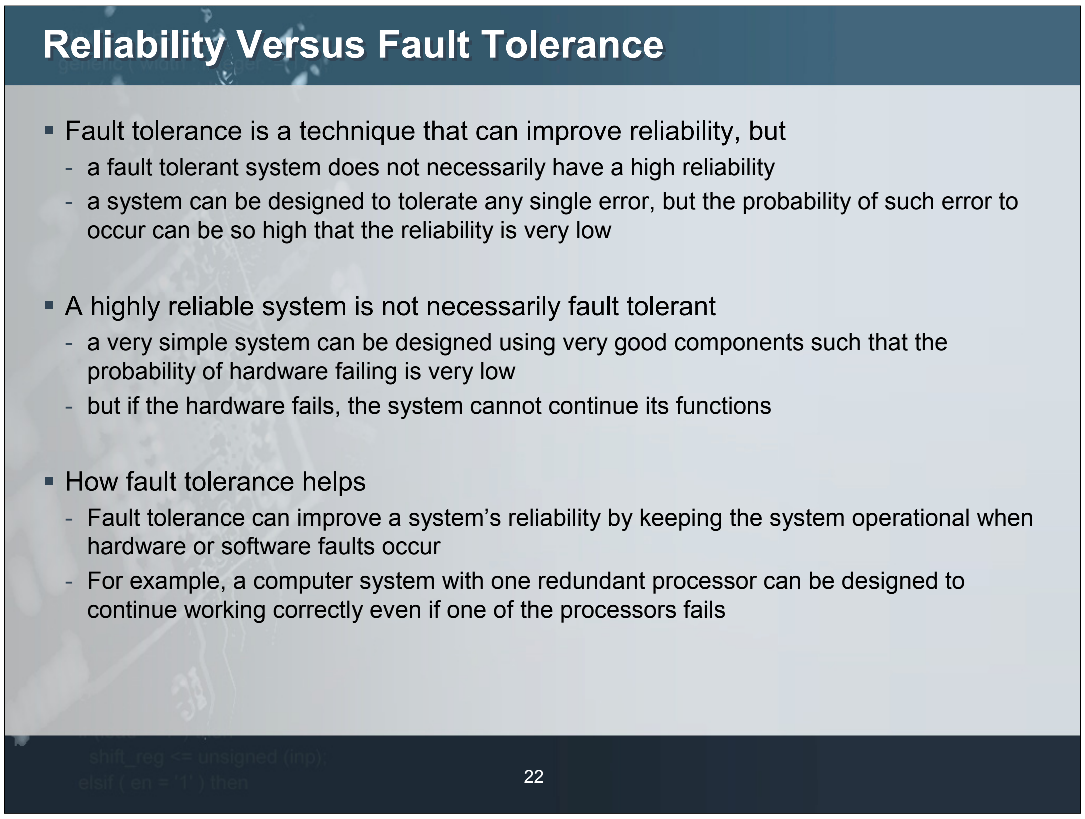
- For example, spacecraft mission control system is expected to provide uninterrupted service. A flaw in the system is likely to cause a destruction of the spacecraft as in the case of NASA's earth-orbiting Lewis spacecraft launched on August 23rd, 1997.
  - The spacecraft entered a flat spin in orbit that resulted in a loss of solar power and a fatal battery discharge. Contact with the spacecraft was lost, and it then re-entered the atmosphere and was destroyed on September 28th. According to the report of the Lewis Spacecraft Mission Failure Investigation, the failure was due to a combination of a technically flawed attitude-control system design and inadequate monitoring of the spacecraft during its crucial early operations phase.



Reliability II

- Reliability is a function of time. The way in which time is specified varies considerably depending on the nature of the system under consideration.
- For example, if a system is expected to complete its mission in a certain period of time, like in case of a spacecraft, time is likely to be defined as a calendar time or as a number of hours.
- For software, the time interval is often specified in so called *natural or time units*. A natural unit is a unit related to the amount of processing performed by a software-based product, such as pages of output, transactions, telephone calls, jobs or queries.
- We need high reliability when:
  - even momentary periods of incorrect performance are unacceptable (aircraft, heart pace maker)
  - no repair possible (satellite, spacecraft)
- High reliability examples
  - airplane: R(several hours) = 0.999 999 9 = 0.9_7
  - spacecraft: R(several years) = 0.95

21



Reliability Versus Fault Tolerance

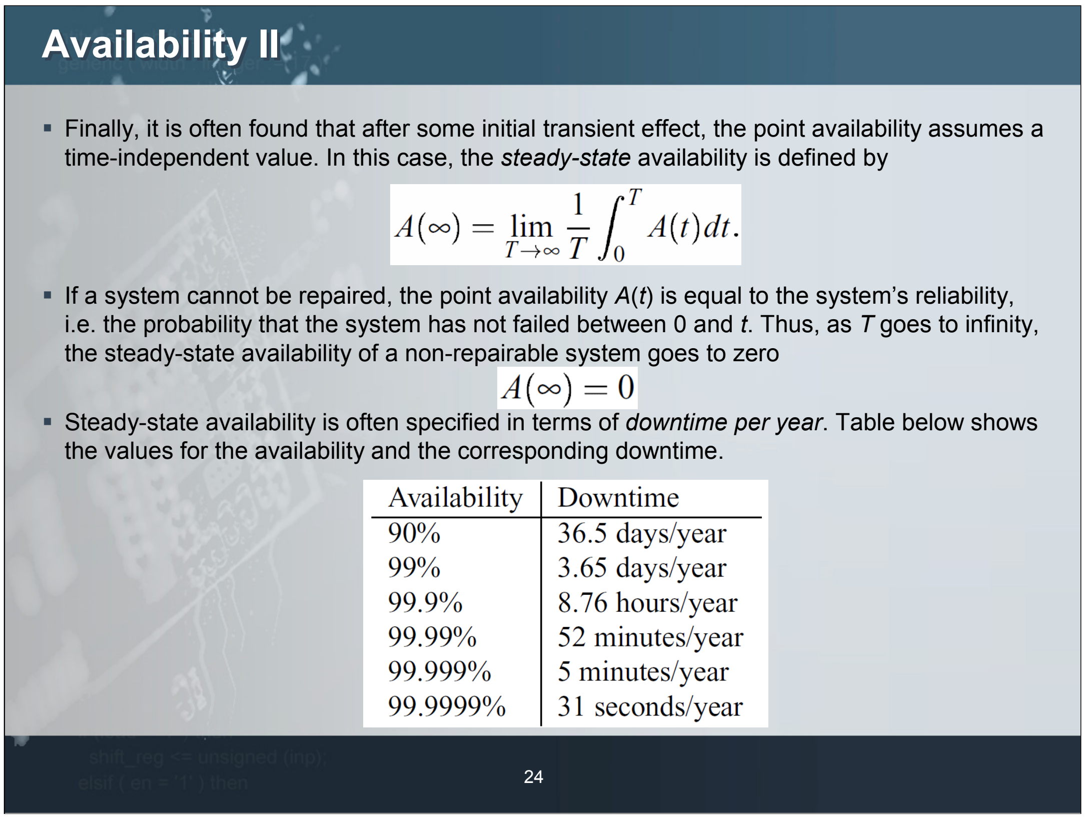
- Fault tolerance is a technique that can improve reliability, but
  - a fault tolerant system does not necessarily have a high reliability
  - a system can be designed to tolerate any single error, but the probability of such error to occur can be so high that the reliability is very low
- A highly reliable system is not necessarily fault tolerant
  - a very simple system can be designed using very good components such that the probability of hardware failing is very low
  - but if the hardware fails, the system cannot continue its functions
- How fault tolerance helps
  - Fault tolerance can improve a system’s reliability by keeping the system operational when hardware or software faults occur
  - For example, a computer system with one redundant processor can be designed to continue working correctly even if one of the processors fails

Availability I

- Relatively few systems are designed to operate continuously without interruption and without maintenance of any kind.
- In many cases, we are interested not only in the probability of failure, but also in the number of failures and, in particular, in the time required to make repairs.
- For such applications, the attribute which we would like to maximize is the fraction of time that the system is in the operational state, expressed by availability.
- ***Availability*** *A*(*t*) of a system at time *t* is the probability that the system is functioning correctly at the instant of time *t*.
- *A*(*t*) is also referred as *point* availability, or *instantaneous* availability. Often it is necessary to determine the *interval* or *mission* availability. It is defined by

$$A(T) = \frac{1}{T}\int_0^T A(t)dt.$$

- *A*(*T*) is the value of the point availability averaged over some interval of time *T*. This interval might be the life-time of a system or the time to accomplish some particular task.



Availability II

- Finally, it is often found that after some initial transient effect, the point availability assumes a time-independent value. In this case, the *steady-state* availability is defined by

$$A(\infty) = \lim_{T\to\infty} \frac{1}{T} \int_0^T A(t)dt.$$

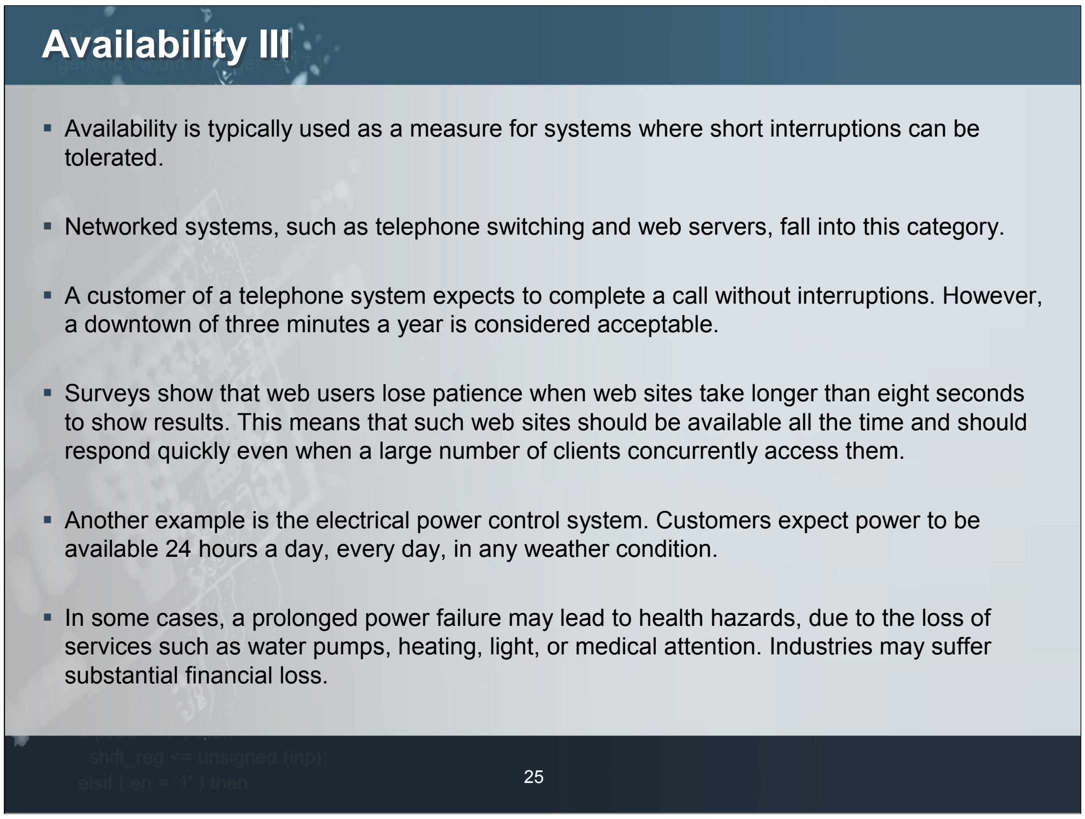
- If a system cannot be repaired, the point availability *A*(*t*) is equal to the system's reliability, i.e. the probability that the system has not failed between 0 and *t*. Thus, as *T* goes to infinity, the steady-state availability of a non-repairable system goes to zero

$$A(\infty) = 0$$

- Steady-state availability is often specified in terms of *downtime per year*. Table below shows the values for the availability and the corresponding downtime.

Availability	Downtime
90%	36.5 days/year
99%	3.65 days/year
99.9%	8.76 hours/year
99.99%	52 minutes/year
99.999%	5 minutes/year
99.9999%	31 seconds/year

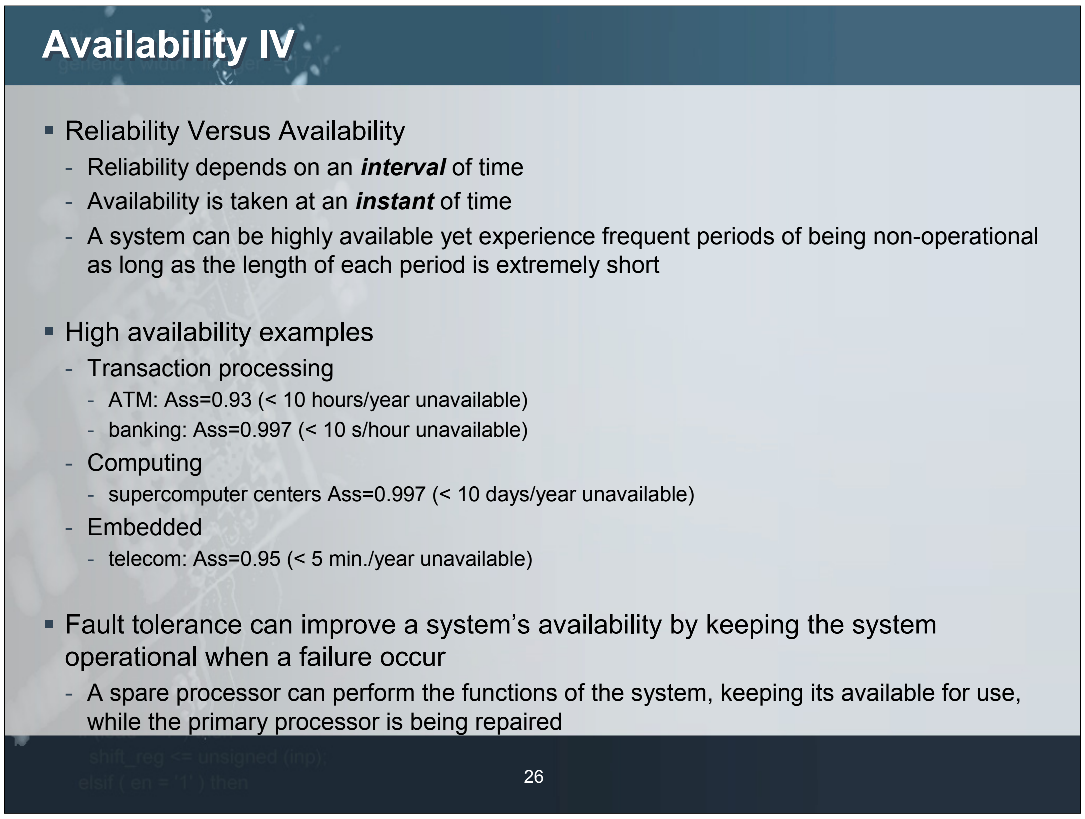
24



Availability III

- Availability is typically used as a measure for systems where short interruptions can be tolerated.
- Networked systems, such as telephone switching and web servers, fall into this category.
- A customer of a telephone system expects to complete a call without interruptions. However, a downtown of three minutes a year is considered acceptable.
- Surveys show that web users lose patience when web sites take longer than eight seconds to show results. This means that such web sites should be available all the time and should respond quickly even when a large number of clients concurrently access them.
- Another example is the electrical power control system. Customers expect power to be available 24 hours a day, every day, in any weather condition.
- In some cases, a prolonged power failure may lead to health hazards, due to the loss of services such as water pumps, heating, light, or medical attention. Industries may suffer substantial financial loss.

25



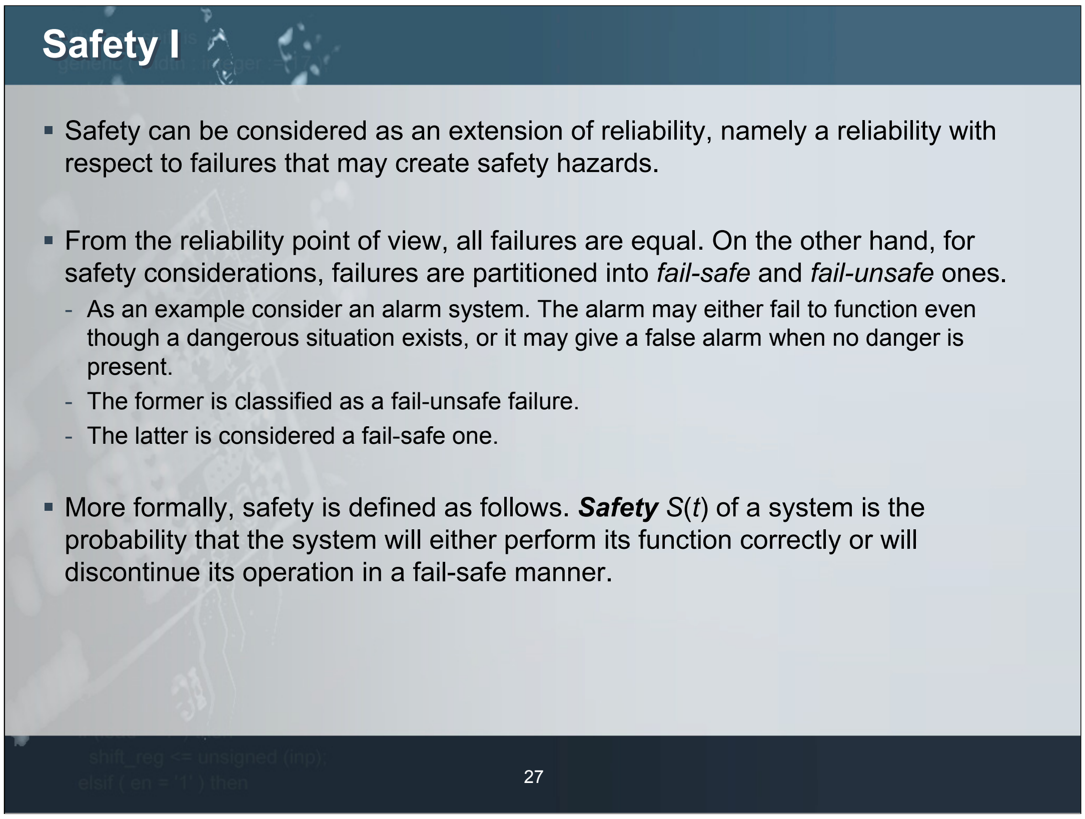
Availability IV

- Reliability Versus Availability
  - Reliability depends on an ***interval*** of time
  - Availability is taken at an ***instant*** of time
  - A system can be highly available yet experience frequent periods of being non-operational as long as the length of each period is extremely short

- High availability examples
  - Transaction processing
    - ATM: Ass=0.93 (< 10 hours/year unavailable)
    - banking: Ass=0.997 (< 10 s/hour unavailable)
  - Computing
    - supercomputer centers Ass=0.997 (< 10 days/year unavailable)
  - Embedded
    - telecom: Ass=0.95 (< 5 min./year unavailable)

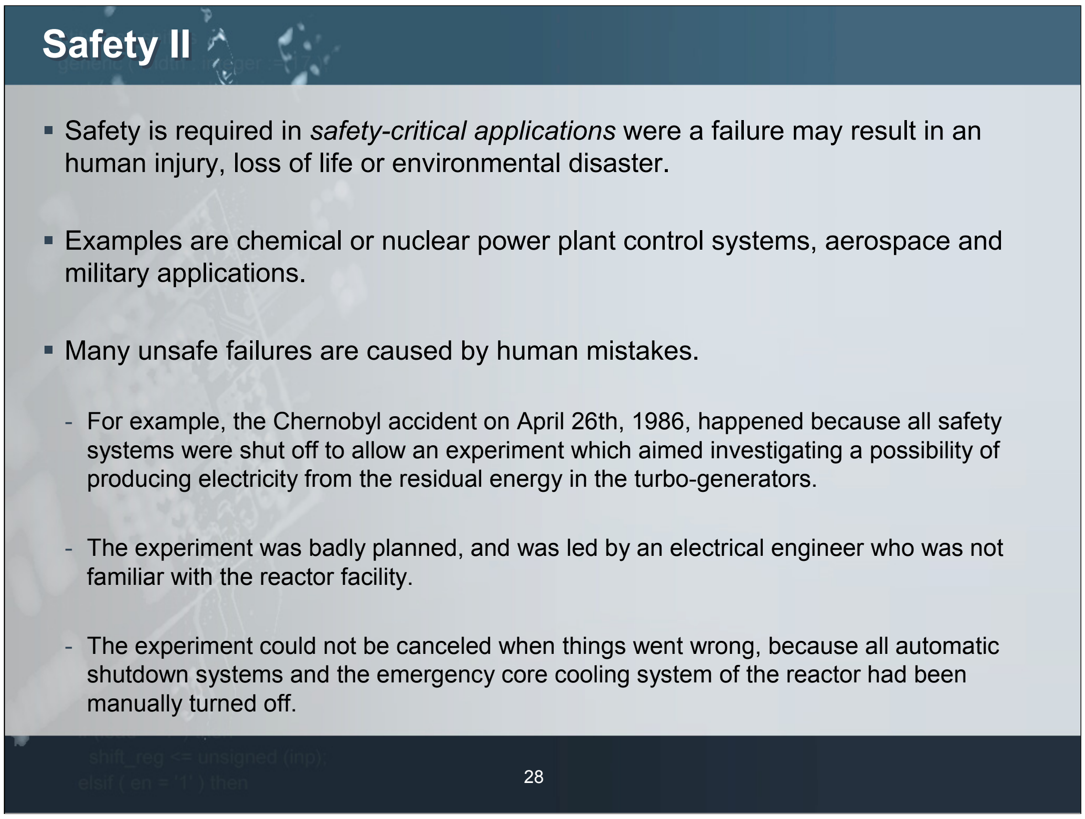
- Fault tolerance can improve a system's availability by keeping the system operational when a failure occur
  - A spare processor can perform the functions of the system, keeping its available for use, while the primary processor is being repaired

26



Safety I

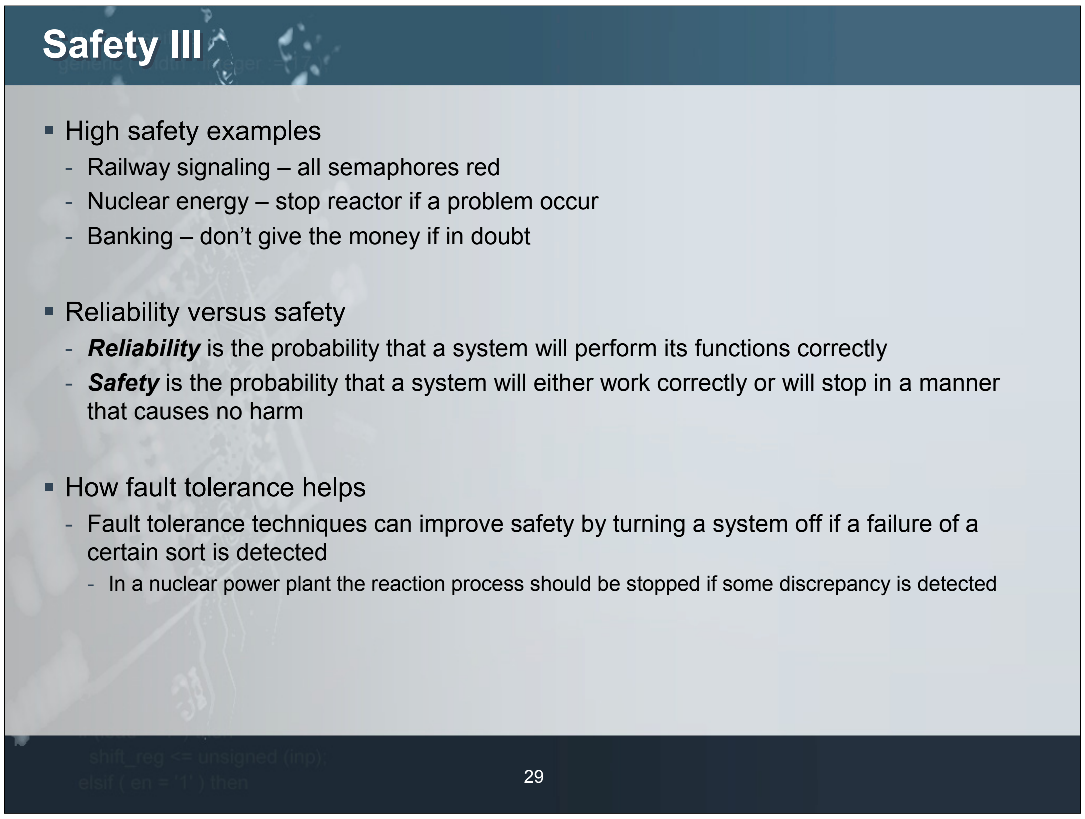
- Safety can be considered as an extension of reliability, namely a reliability with respect to failures that may create safety hazards.
- From the reliability point of view, all failures are equal. On the other hand, for safety considerations, failures are partitioned into *fail-safe* and *fail-unsafe* ones.
  - As an example consider an alarm system. The alarm may either fail to function even though a dangerous situation exists, or it may give a false alarm when no danger is present.
  - The former is classified as a fail-unsafe failure.
  - The latter is considered a fail-safe one.
- More formally, safety is defined as follows. ***Safety*** *S*(*t*) of a system is the probability that the system will either perform its function correctly or will discontinue its operation in a fail-safe manner.



Safety II

- Safety is required in *safety-critical applications* were a failure may result in an human injury, loss of life or environmental disaster.
- Examples are chemical or nuclear power plant control systems, aerospace and military applications.
- Many unsafe failures are caused by human mistakes.
  - For example, the Chernobyl accident on April 26th, 1986, happened because all safety systems were shut off to allow an experiment which aimed investigating a possibility of producing electricity from the residual energy in the turbo-generators.
  - The experiment was badly planned, and was led by an electrical engineer who was not familiar with the reactor facility.
  - The experiment could not be canceled when things went wrong, because all automatic shutdown systems and the emergency core cooling system of the reactor had been manually turned off.

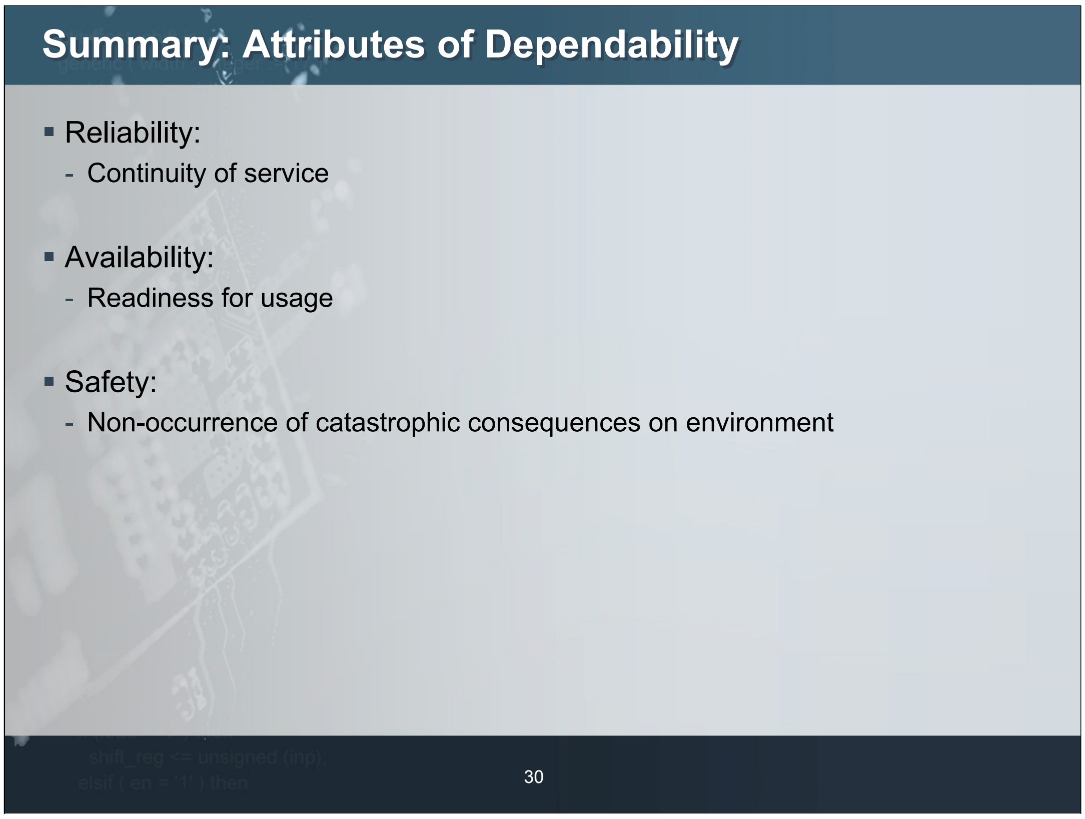
28



Safety III

- High safety examples
  - Railway signaling – all semaphores red
  - Nuclear energy – stop reactor if a problem occur
  - Banking – don’t give the money if in doubt
- Reliability versus safety
  - ***Reliability*** is the probability that a system will perform its functions correctly
  - ***Safety*** is the probability that a system will either work correctly or will stop in a manner that causes no harm
- How fault tolerance helps
  - Fault tolerance techniques can improve safety by turning a system off if a failure of a certain sort is detected
    - In a nuclear power plant the reaction process should be stopped if some discrepancy is detected

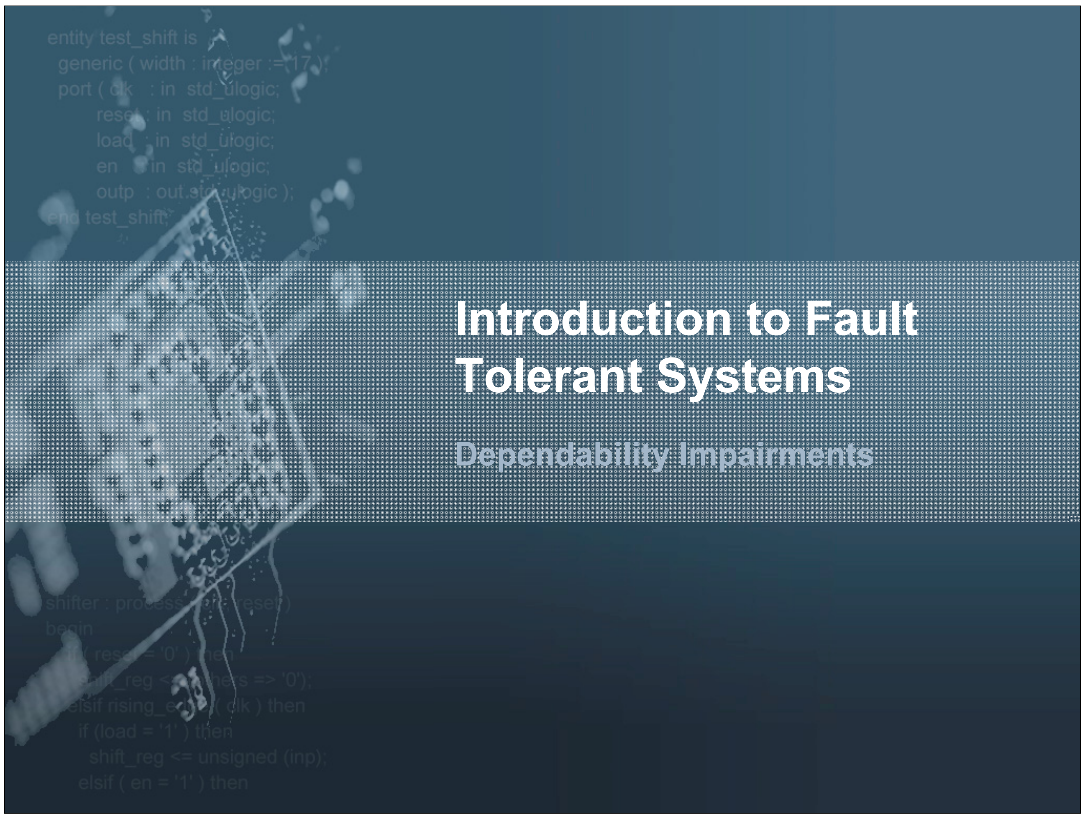
29



Summary: Attributes of Dependability

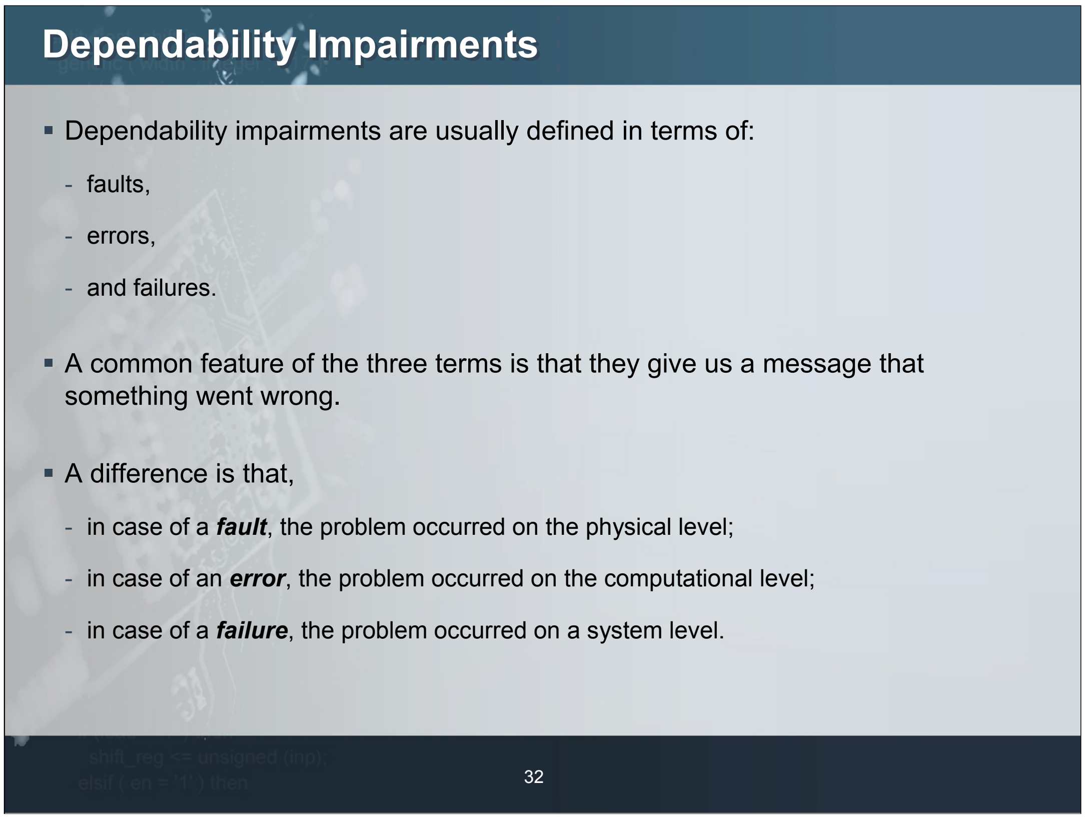
- Reliability:
  - Continuity of service
- Availability:
  - Readiness for usage
- Safety:
  - Non-occurrence of catastrophic consequences on environment

30



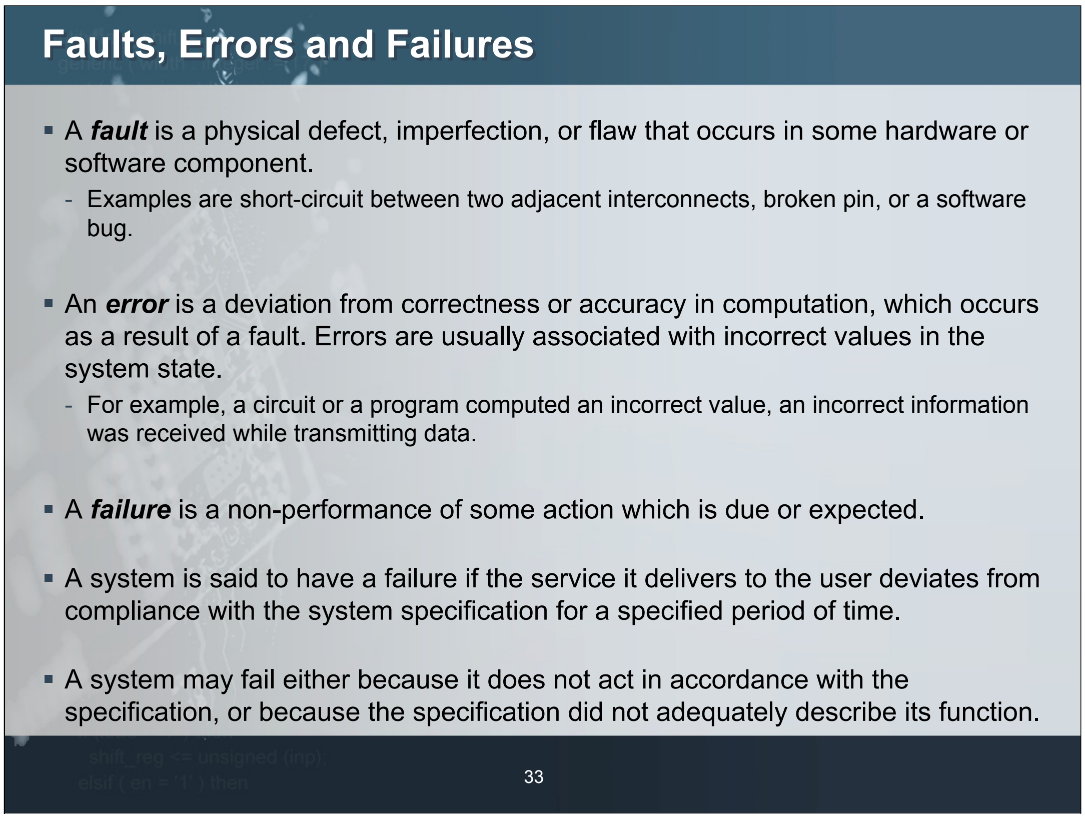
Introduction to Fault Tolerant Systems

Dependability Impairments



Dependability Impairments

- Dependability impairments are usually defined in terms of:
  - faults,
  - errors,
  - and failures.
- A common feature of the three terms is that they give us a message that something went wrong.
- A difference is that,
  - in case of a ***fault***, the problem occurred on the physical level;
  - in case of an ***error***, the problem occurred on the computational level;
  - in case of a ***failure***, the problem occurred on a system level.



Faults, Errors and Failures

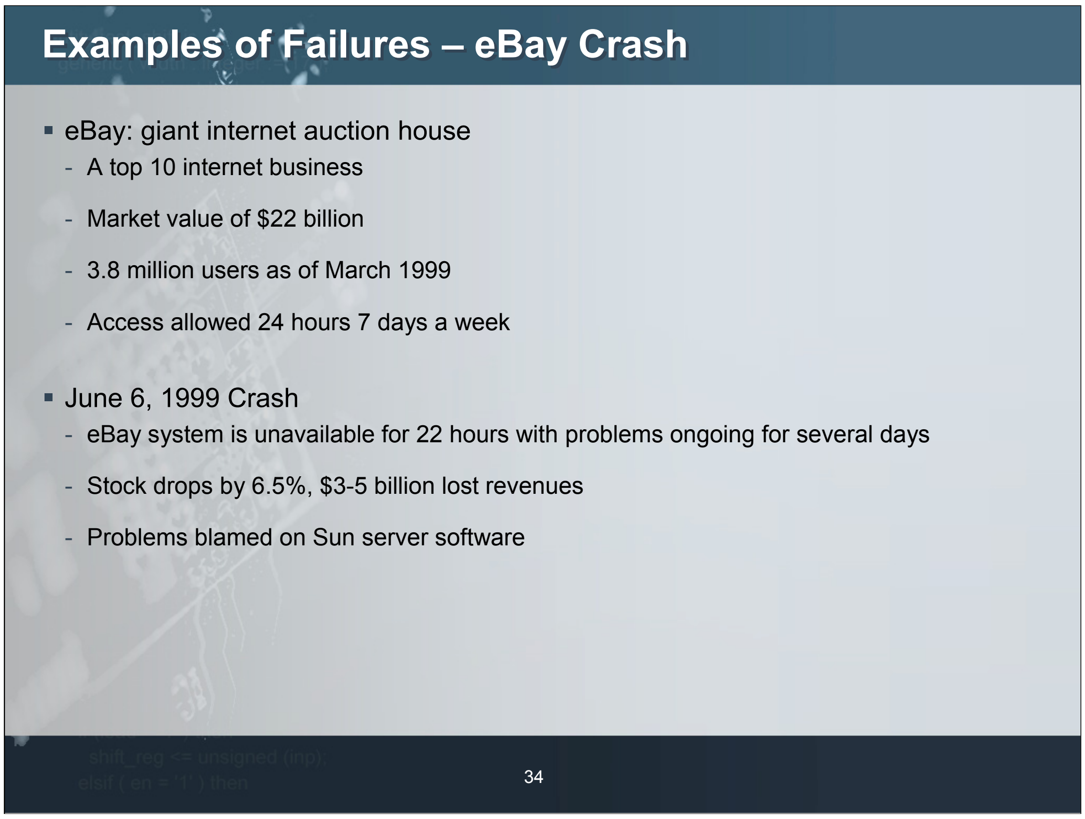
- A ***fault*** is a physical defect, imperfection, or flaw that occurs in some hardware or software component.
  - Examples are short-circuit between two adjacent interconnects, broken pin, or a software bug.
- An ***error*** is a deviation from correctness or accuracy in computation, which occurs as a result of a fault. Errors are usually associated with incorrect values in the system state.
  - For example, a circuit or a program computed an incorrect value, an incorrect information was received while transmitting data.
- A ***failure*** is a non-performance of some action which is due or expected.
- A system is said to have a failure if the service it delivers to the user deviates from compliance with the system specification for a specified period of time.
- A system may fail either because it does not act in accordance with the specification, or because the specification did not adequately describe its function.

33

Faults are reasons for errors and errors are reasons for failures. For example, consider a power plant, in which a computer controlled system is responsible for monitoring various plant temperatures, pressures, and other physical characteristics. The sensor reporting the speed at which the main turbine is spinning breaks. This fault causes the system to send more steam to the turbine than is required (error), over-speeding the turbine, and resulting in the mechanical safety system shutting down the turbine to prevent damaging it. The system is no longer generating power (system failure, fail-safe).

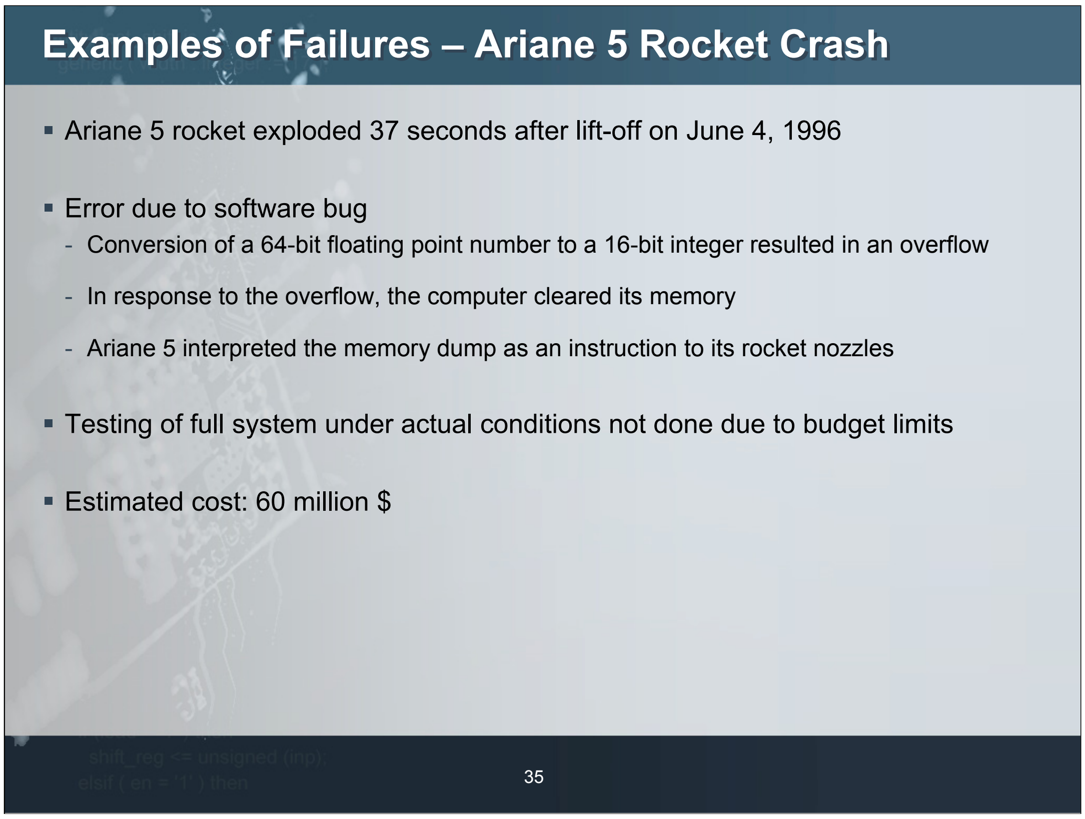
The definitions of physical, computational and system level are a bit more confusing when applied to software. In the context of this course, we interpret a program code as physical level, the values of a program state as computational level, and the software system running the program as system level. For example, an operating system is a software system. Then, a bug in a program is a fault, possible incorrect value caused by this bug is an error and possible crush of the operating system is a failure.

Not every fault causes an error and not every error causes a failure. This is particularly evident in the software case. Some program bugs are very hard to find because they cause failures only in very specific situations. For example, in November 1985, $32 billion overdraft was experienced by the Bank of New York, leading to a loss of $5 million in interests. The failure was caused by an unchecked overflow of an 16-bit counter. In 1994, Intel Pentium I microprocessor was discovered to compute incorrect answers to certain floating-point division calculations. For example, dividing 5505001 by 294911 produced 18.66600093 instead of 18.66665197. The problem had occurred because of the omission of five entries in a table of 1066 values used by the division algorithm. The five cells should have contained the constant +2, but because the cells were empty, the processor treated them as a zero.



Examples of Failures – eBay Crash

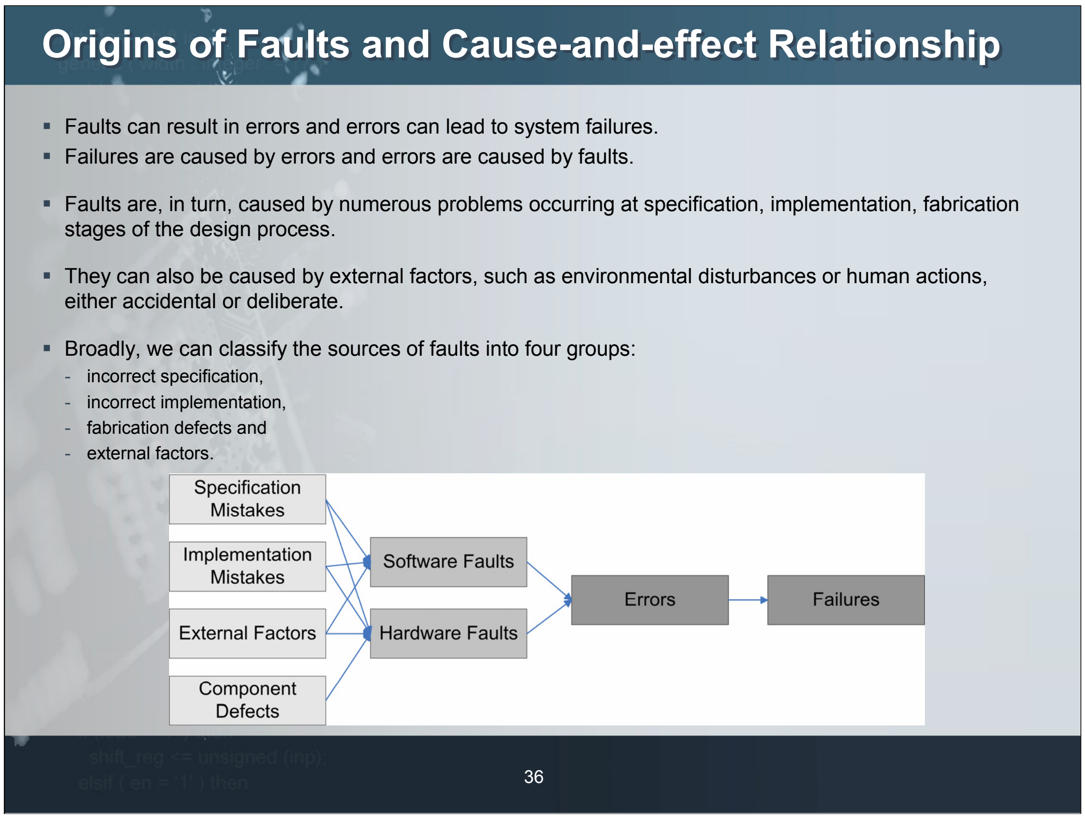
- eBay: giant internet auction house
  - A top 10 internet business
  - Market value of $22 billion
  - 3.8 million users as of March 1999
  - Access allowed 24 hours 7 days a week
- June 6, 1999 Crash
  - eBay system is unavailable for 22 hours with problems ongoing for several days
  - Stock drops by 6.5%, $3-5 billion lost revenues
  - Problems blamed on Sun server software



Examples of Failures – Ariane 5 Rocket Crash

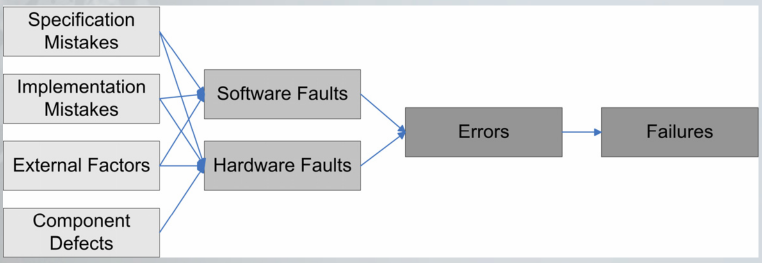
- Ariane 5 rocket exploded 37 seconds after lift-off on June 4, 1996
- Error due to software bug
  - Conversion of a 64-bit floating point number to a 16-bit integer resulted in an overflow
  - In response to the overflow, the computer cleared its memory
  - Ariane 5 interpreted the memory dump as an instruction to its rocket nozzles
- Testing of full system under actual conditions not done due to budget limits
- Estimated cost: 60 million $

35

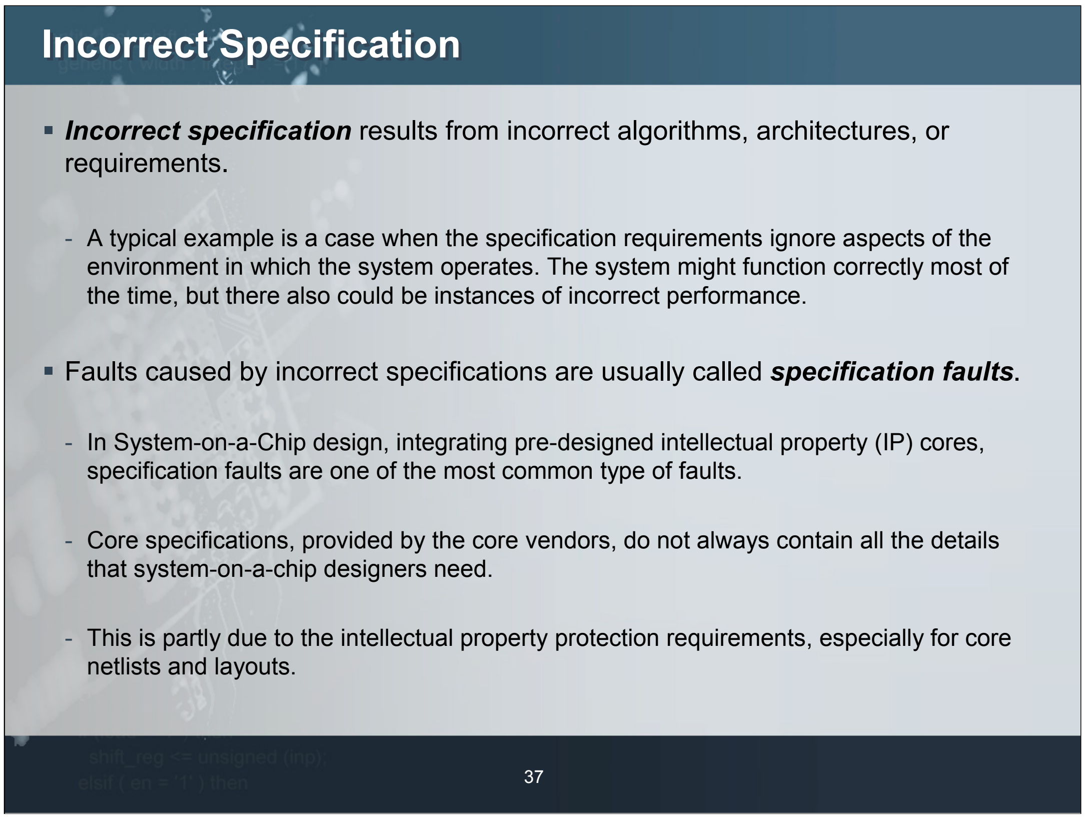


Origins of Faults and Cause-and-effect Relationship

- Faults can result in errors and errors can lead to system failures.
- Failures are caused by errors and errors are caused by faults.
- Faults are, in turn, caused by numerous problems occurring at specification, implementation, fabrication stages of the design process.
- They can also be caused by external factors, such as environmental disturbances or human actions, either accidental or deliberate.
- Broadly, we can classify the sources of faults into four groups:
  - incorrect specification,
  - incorrect implementation,
  - fabrication defects and
  - external factors.



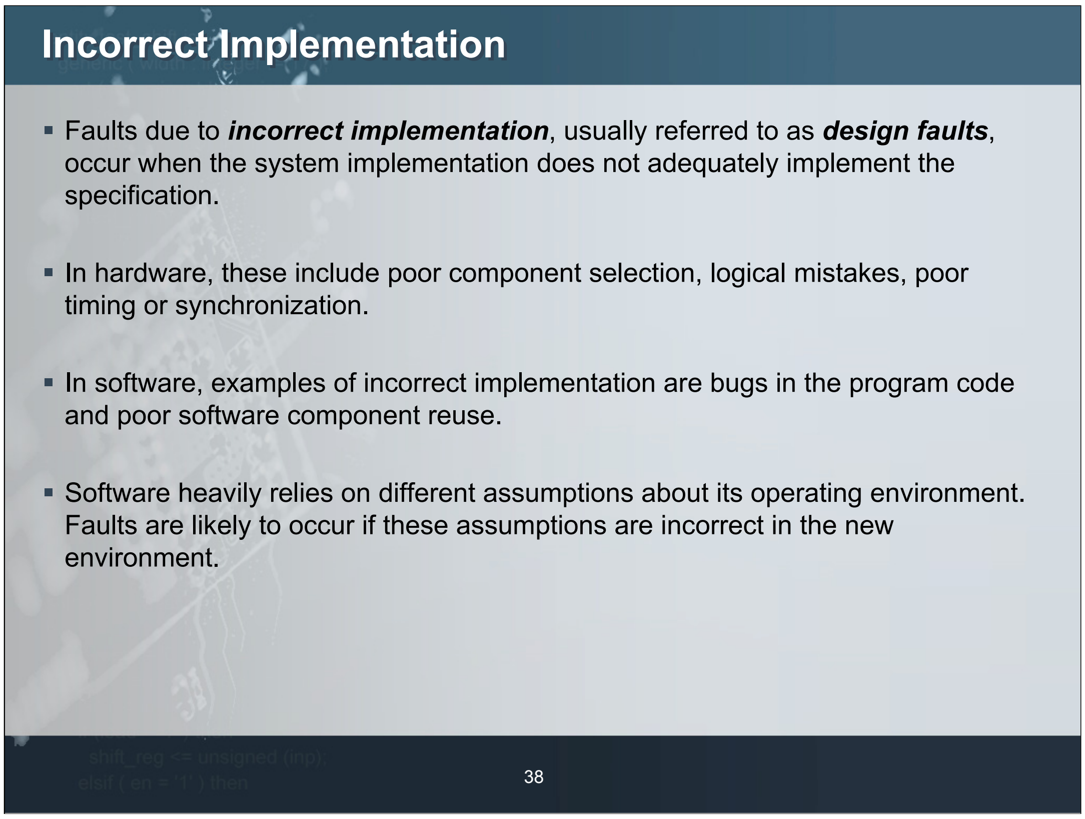
36



Incorrect Specification

- ***Incorrect specification*** results from incorrect algorithms, architectures, or requirements.
  - A typical example is a case when the specification requirements ignore aspects of the environment in which the system operates. The system might function correctly most of the time, but there also could be instances of incorrect performance.
- Faults caused by incorrect specifications are usually called ***specification faults***.
  - In System-on-a-Chip design, integrating pre-designed intellectual property (IP) cores, specification faults are one of the most common type of faults.
  - Core specifications, provided by the core vendors, do not always contain all the details that system-on-a-chip designers need.
  - This is partly due to the intellectual property protection requirements, especially for core netlists and layouts.

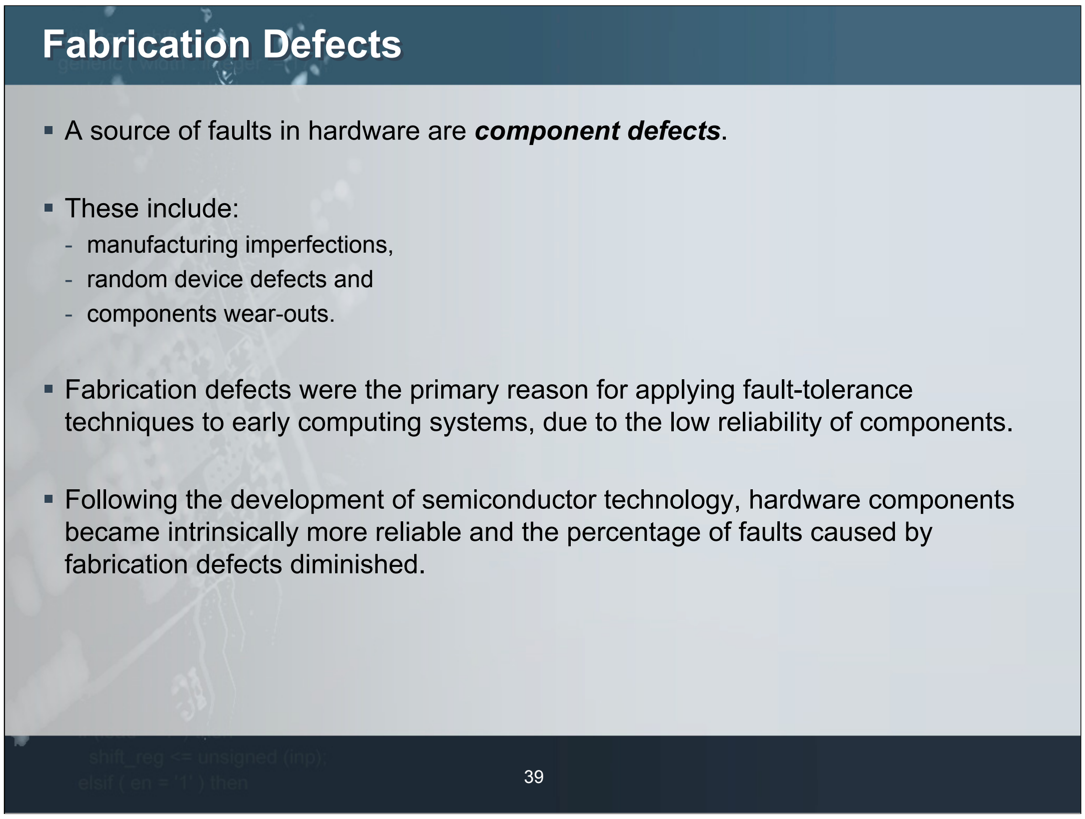
37



Incorrect Implementation

- Faults due to ***incorrect implementation***, usually referred to as ***design faults***, occur when the system implementation does not adequately implement the specification.
- In hardware, these include poor component selection, logical mistakes, poor timing or synchronization.
- In software, examples of incorrect implementation are bugs in the program code and poor software component reuse.
- Software heavily relies on different assumptions about its operating environment. Faults are likely to occur if these assumptions are incorrect in the new environment.

38



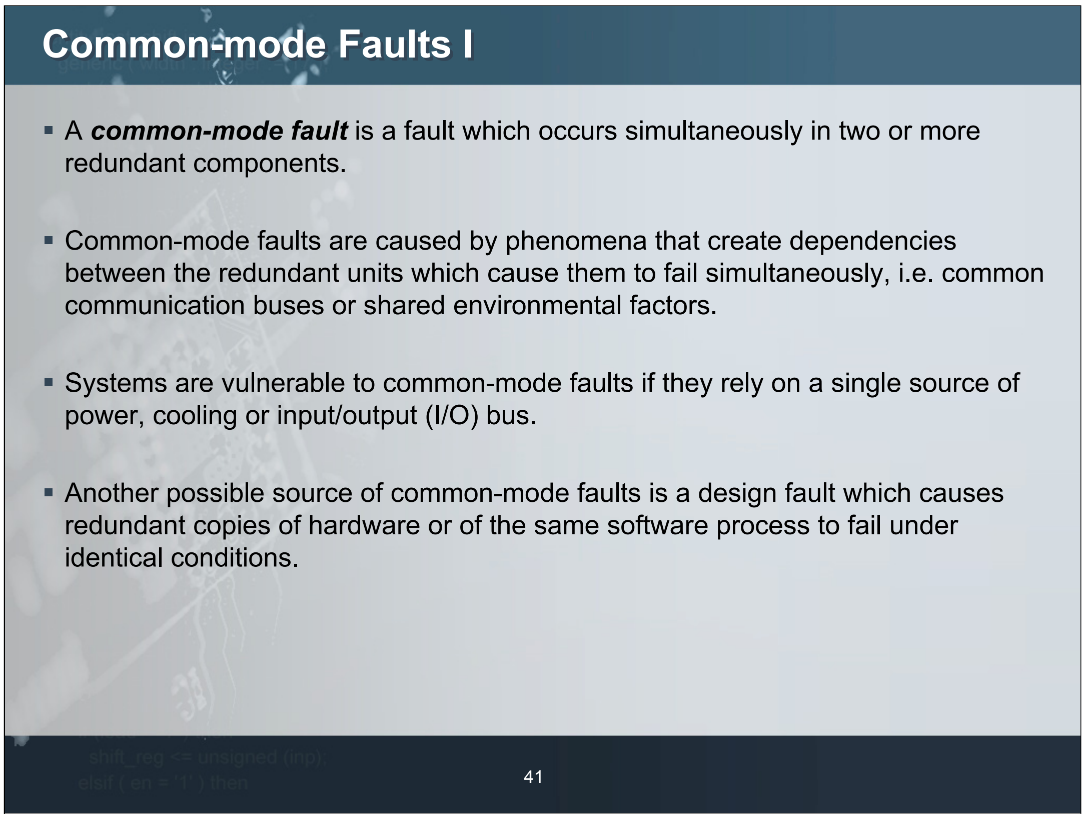
Fabrication Defects

- A source of faults in hardware are ***component defects***.
- These include:
  - manufacturing imperfections,
  - random device defects and
  - components wear-outs.
- Fabrication defects were the primary reason for applying fault-tolerance techniques to early computing systems, due to the low reliability of components.
- Following the development of semiconductor technology, hardware components became intrinsically more reliable and the percentage of faults caused by fabrication defects diminished.

39

External Factors

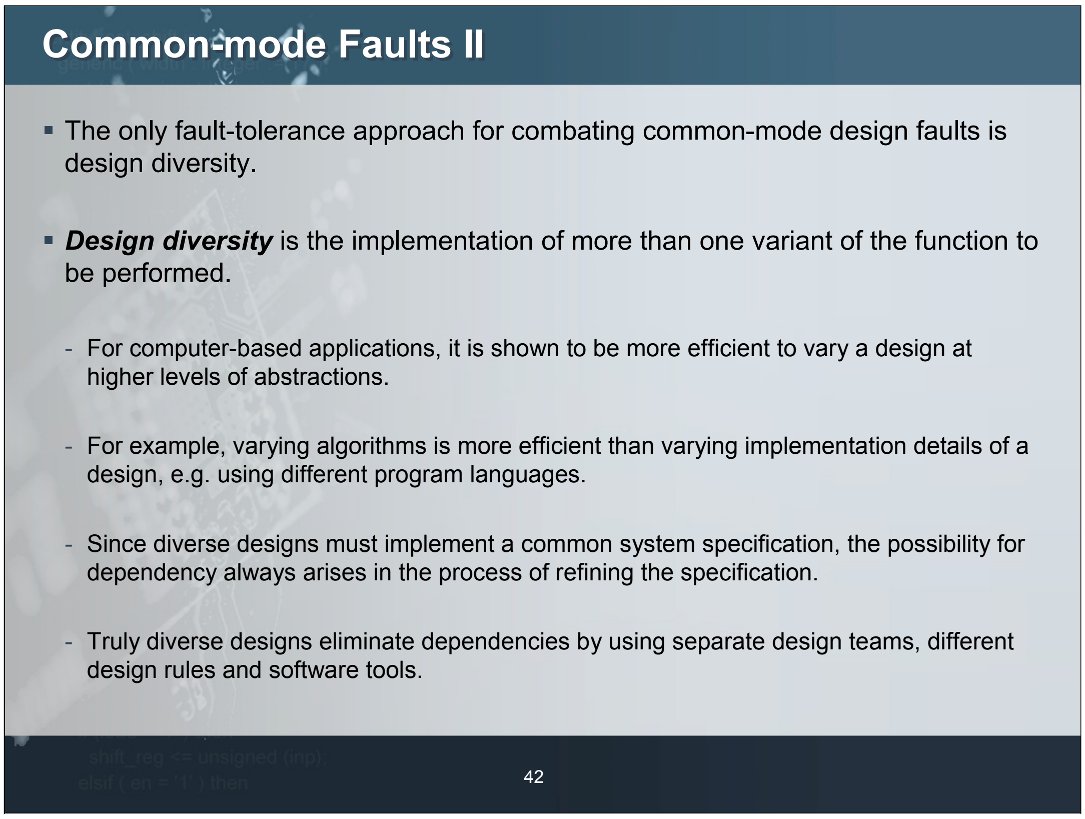
- The fourth cause of faults are ***external factors***, which arise from outside the system boundary, the environment, the user or the operator.
- External factors include phenomena that directly affect the operation of the system, such as temperature, vibration, electrostatic discharge, nuclear or electromagnetic radiation or that affect the inputs provided to the system.
- For instance, radiation causing a bit to flip in a memory location is a fault caused by an external factor.
- Faults caused by user or operator mistakes can be accidental or malicious.
- For example, a user can accidentally provide incorrect commands to a system that can lead to system failure, e.g. improperly initialized variables in software.
- Malicious faults are the ones caused, for example, by software viruses and hacker intrusions.



Common-mode Faults I

- A ***common-mode fault*** is a fault which occurs simultaneously in two or more redundant components.
- Common-mode faults are caused by phenomena that create dependencies between the redundant units which cause them to fail simultaneously, i.e. common communication buses or shared environmental factors.
- Systems are vulnerable to common-mode faults if they rely on a single source of power, cooling or input/output (I/O) bus.
- Another possible source of common-mode faults is a design fault which causes redundant copies of hardware or of the same software process to fail under identical conditions.

41



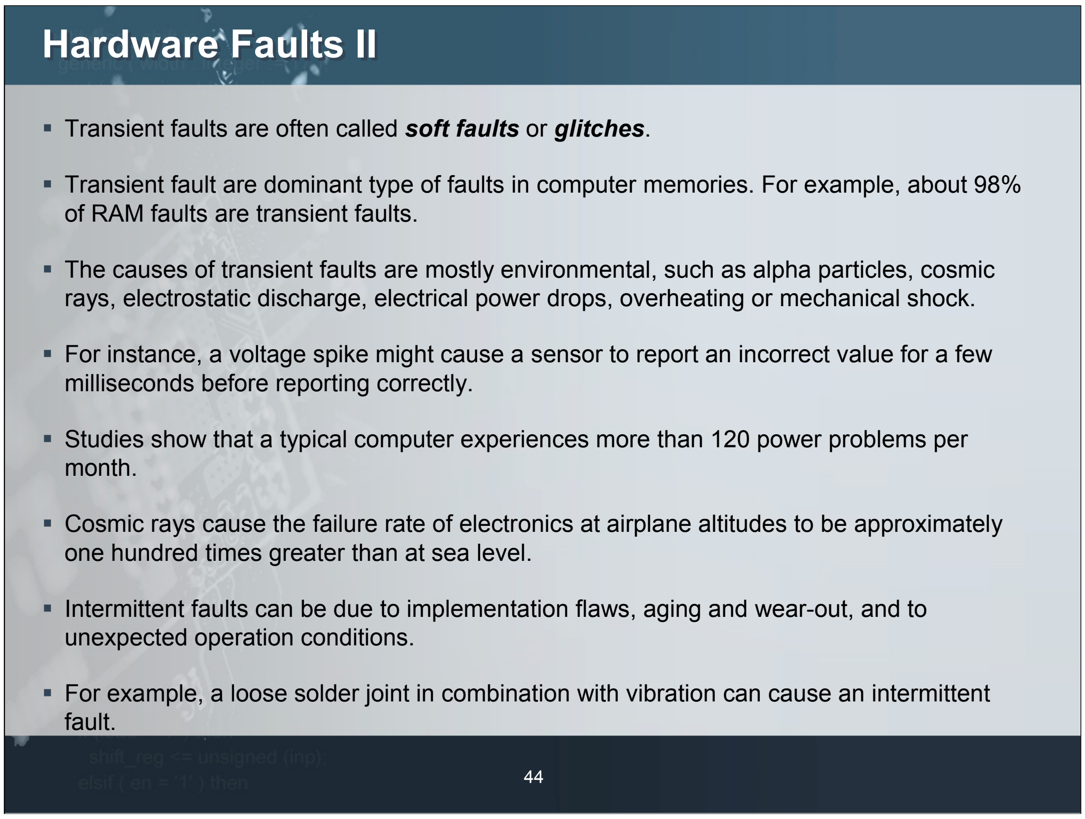
Common-mode Faults II

- The only fault-tolerance approach for combating common-mode design faults is design diversity.
- ***Design diversity*** is the implementation of more than one variant of the function to be performed.
  - For computer-based applications, it is shown to be more efficient to vary a design at higher levels of abstractions.
  - For example, varying algorithms is more efficient than varying implementation details of a design, e.g. using different program languages.
  - Since diverse designs must implement a common system specification, the possibility for dependency always arises in the process of refining the specification.
  - Truly diverse designs eliminate dependencies by using separate design teams, different design rules and software tools.

42

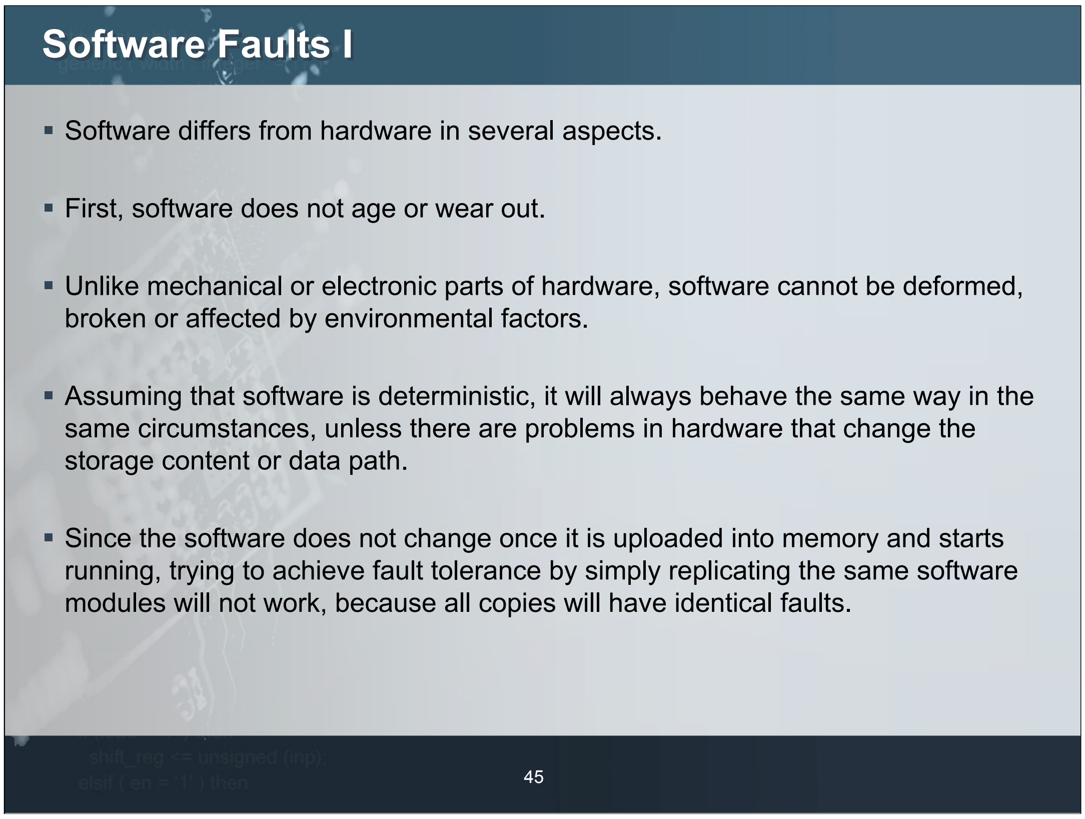
Hardware Faults I

- Hardware faults are classified with respect to fault duration into:
  - permanent,
  - transient and
  - intermittent faults.
- A ***permanent fault*** remains active until a corrective action is taken.
- These faults are usually caused by some physical defects in the hardware, such as shorts in a circuit, broken interconnections or stuck bits in the memory.
- Permanent faults can be detected by on-line test routines that work concurrently with the normal system operation.
- A ***transient fault*** remains active for a short period of time.
- A transient fault that becomes active periodically is an ***intermittent fault***.
- Because of their short duration, transient faults are often detected through the errors that result from their propagation.



Hardware Faults II

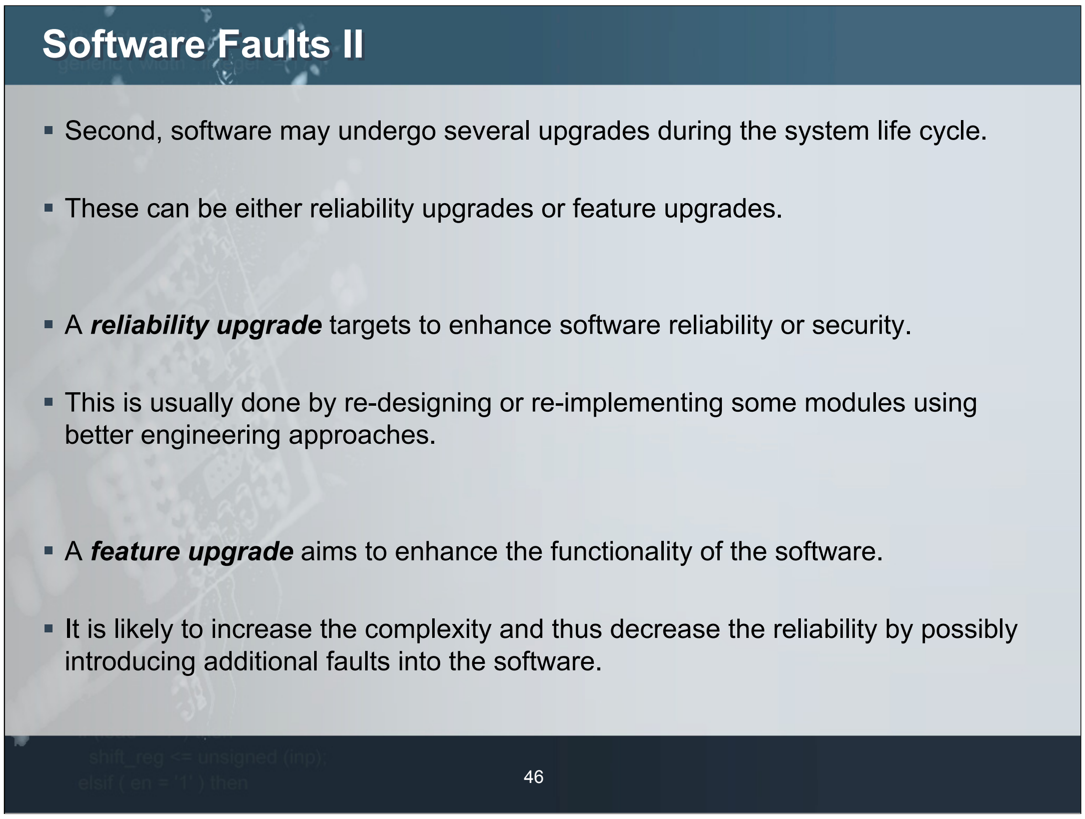
- Transient faults are often called ***soft faults*** or ***glitches***.
- Transient fault are dominant type of faults in computer memories. For example, about 98% of RAM faults are transient faults.
- The causes of transient faults are mostly environmental, such as alpha particles, cosmic rays, electrostatic discharge, electrical power drops, overheating or mechanical shock.
- For instance, a voltage spike might cause a sensor to report an incorrect value for a few milliseconds before reporting correctly.
- Studies show that a typical computer experiences more than 120 power problems per month.
- Cosmic rays cause the failure rate of electronics at airplane altitudes to be approximately one hundred times greater than at sea level.
- Intermittent faults can be due to implementation flaws, aging and wear-out, and to unexpected operation conditions.
- For example, a loose solder joint in combination with vibration can cause an intermittent fault.



Software Faults I

- Software differs from hardware in several aspects.
- First, software does not age or wear out.
- Unlike mechanical or electronic parts of hardware, software cannot be deformed, broken or affected by environmental factors.
- Assuming that software is deterministic, it will always behave the same way in the same circumstances, unless there are problems in hardware that change the storage content or data path.
- Since the software does not change once it is uploaded into memory and starts running, trying to achieve fault tolerance by simply replicating the same software modules will not work, because all copies will have identical faults.

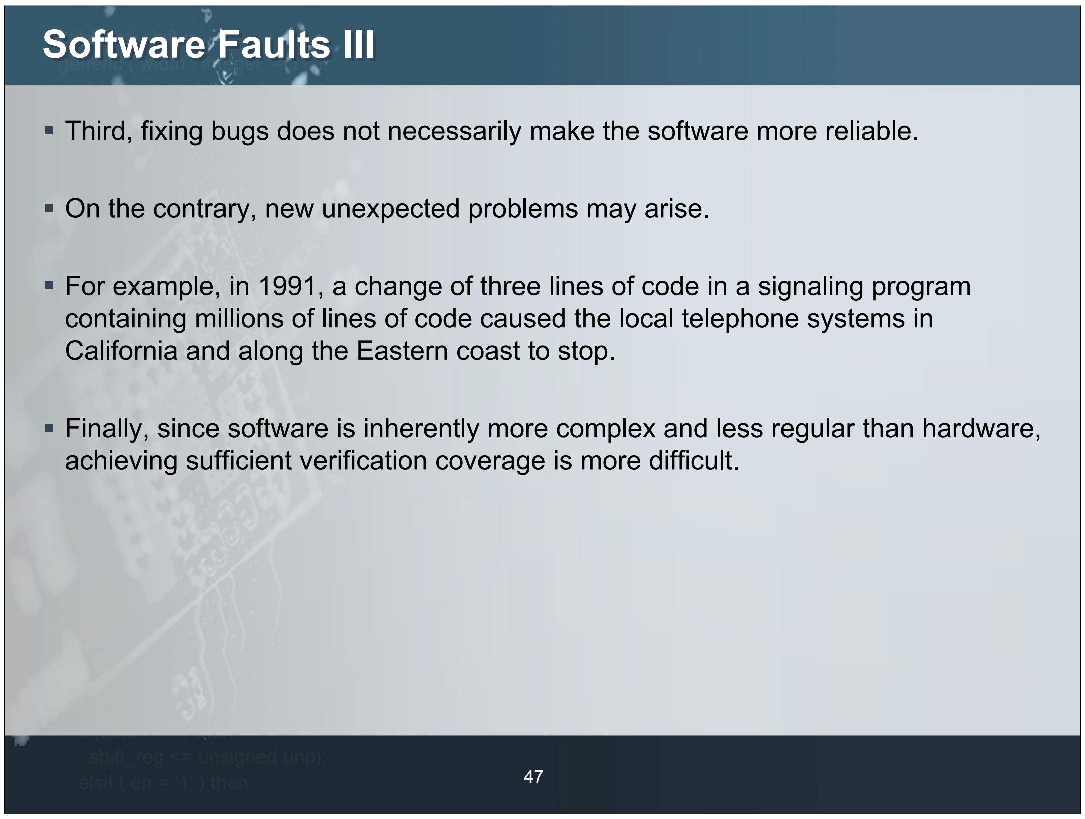
45



Software Faults II

- Second, software may undergo several upgrades during the system life cycle.
- These can be either reliability upgrades or feature upgrades.
- A ***reliability upgrade*** targets to enhance software reliability or security.
- This is usually done by re-designing or re-implementing some modules using better engineering approaches.
- A ***feature upgrade*** aims to enhance the functionality of the software.
- It is likely to increase the complexity and thus decrease the reliability by possibly introducing additional faults into the software.

46

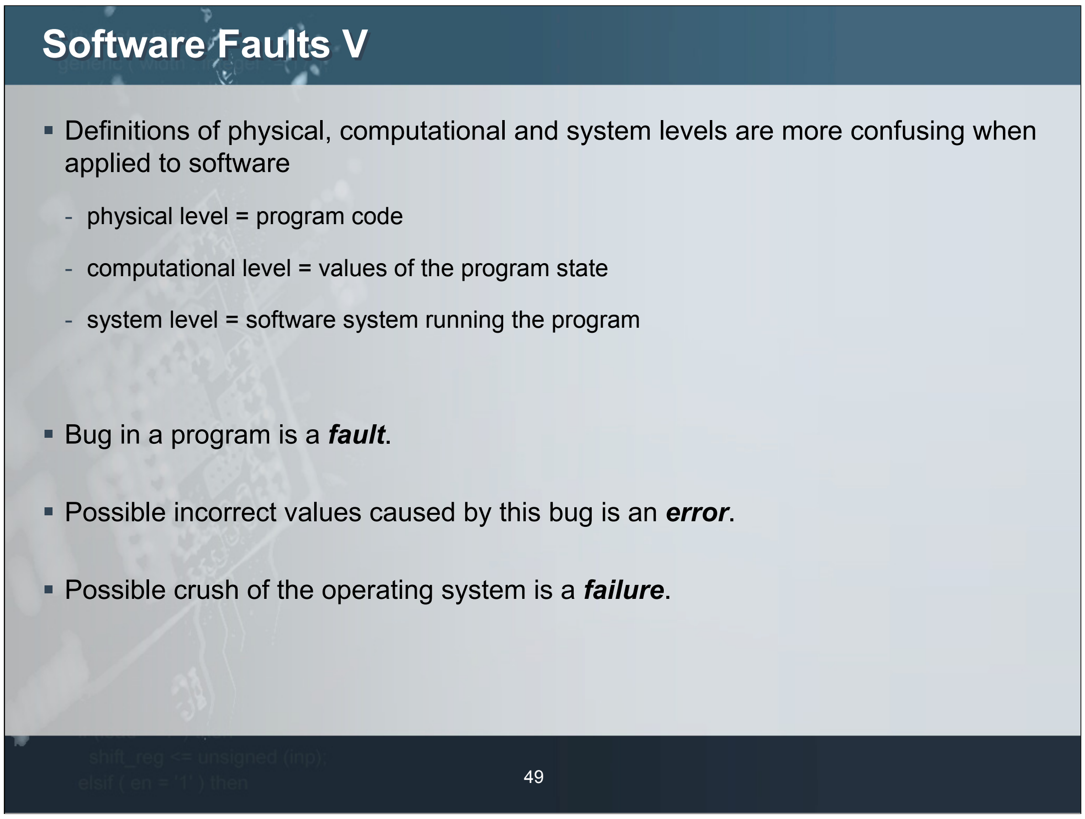


Software Faults III

- Third, fixing bugs does not necessarily make the software more reliable.
- On the contrary, new unexpected problems may arise.
- For example, in 1991, a change of three lines of code in a signaling program containing millions of lines of code caused the local telephone systems in California and along the Eastern coast to stop.
- Finally, since software is inherently more complex and less regular than hardware, achieving sufficient verification coverage is more difficult.

Software Faults IV

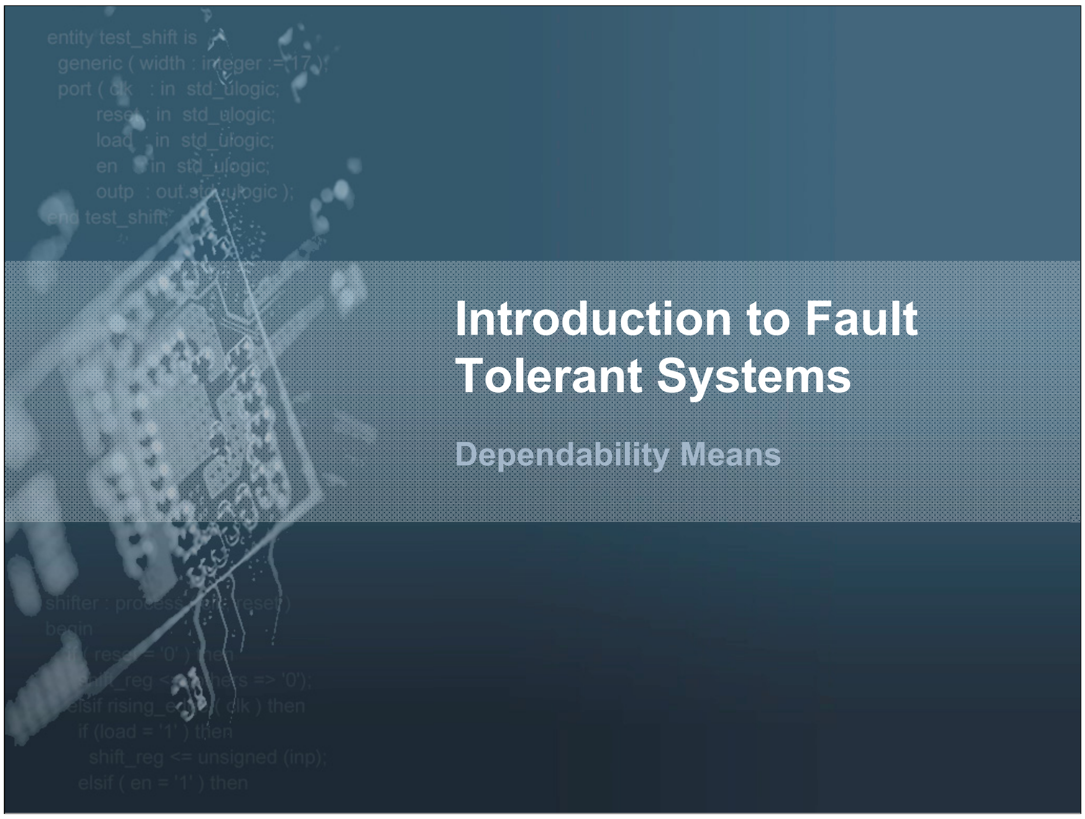
- Traditional testing and debugging methods are inadequate for large software systems.
- The recent focus on formal methods promises higher coverage, however, due to their extremely large computational complexity they are only applicable in specific applications.
- Due to incomplete verification, most of the software faults are design faults, occurring when a programmer either misunderstands the specification or simply makes a mistake.
- Design faults are related to fuzzy human factors, and therefore they are harder to prevent.
- In hardware, design faults may also exist, but other types of faults, such as fabrication defects and transient faults caused by environmental factors, usually dominate.



Software Faults V

- Definitions of physical, computational and system levels are more confusing when applied to software
  - physical level = program code
  - computational level = values of the program state
  - system level = software system running the program
- Bug in a program is a ***fault***.
- Possible incorrect values caused by this bug is an ***error***.
- Possible crush of the operating system is a ***failure***.

49



Introduction to Fault Tolerant Systems

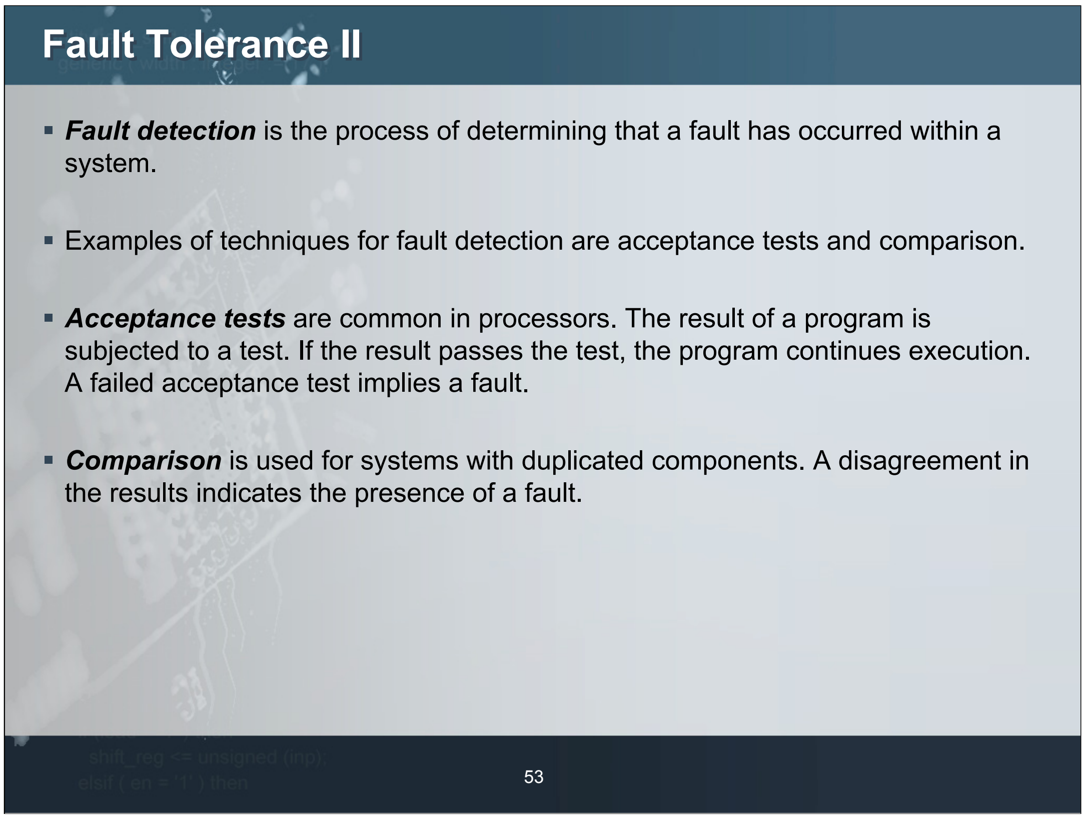
Dependability Means

Dependability Means

- ***Dependability means*** are the methods and techniques enabling the development of a dependable system.
- ***Fault tolerance***, which is the subject of this course, is one of such methods.
- It is normally used in a combination with other methods to attain dependability, such as fault prevention, fault removal and fault forecasting.
- ***Fault prevention*** aims to prevent the occurrences or introduction of faults.
- ***Fault removal*** aims to reduce the number of faults which are present in the system.
- ***Fault forecasting*** aims to estimate how many faults are present, possible future occurrences of faults, and the impact of the faults on the system.

Fault Tolerance I

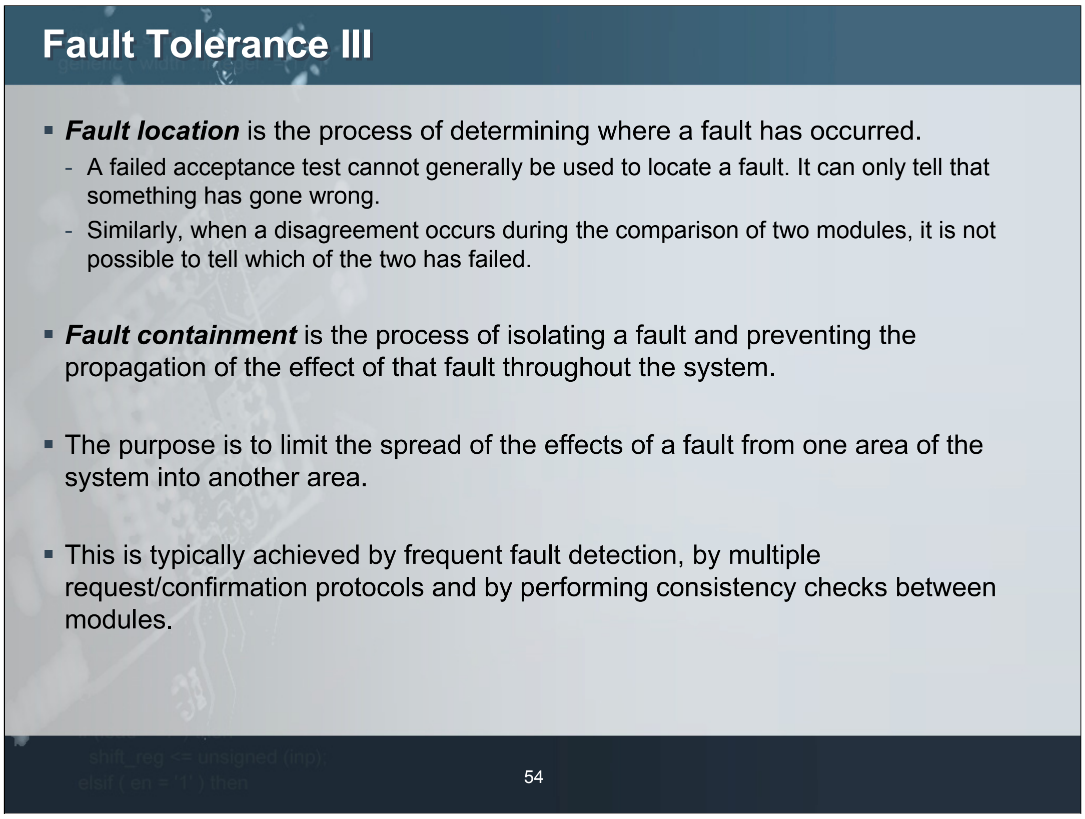
- Fault tolerance targets the development of systems which function correctly in presence of faults.
- Fault tolerance is achieved by using some kind of ***redundancy***.
- In the context of this course, ***redundancy*** is the provision of functional capabilities that would be unnecessary in a fault-free environment.
- The redundancy allows either to ***mask*** a fault, or to ***detect*** a fault, with the following ***location***, ***containment*** and ***recovery***.
- ***Fault masking*** is the process of insuring that only correct values get passed to the system output in spite of the presence of a fault.
- This is done by preventing the system from being affected by errors by either correcting the error, or compensating for it in some fashion.
- Since the system does not show the impact of the fault, the existence of fault is therefore invisible to the user/operator.
    - For example, a memory protected by an error-correcting code corrects the faulty bits before the system uses the data. Another example of fault masking is triple modular redundancy with majority voting.



Fault Tolerance II

- ***Fault detection*** is the process of determining that a fault has occurred within a system.
- Examples of techniques for fault detection are acceptance tests and comparison.
- ***Acceptance tests*** are common in processors. The result of a program is subjected to a test. If the result passes the test, the program continues execution. A failed acceptance test implies a fault.
- ***Comparison*** is used for systems with duplicated components. A disagreement in the results indicates the presence of a fault.

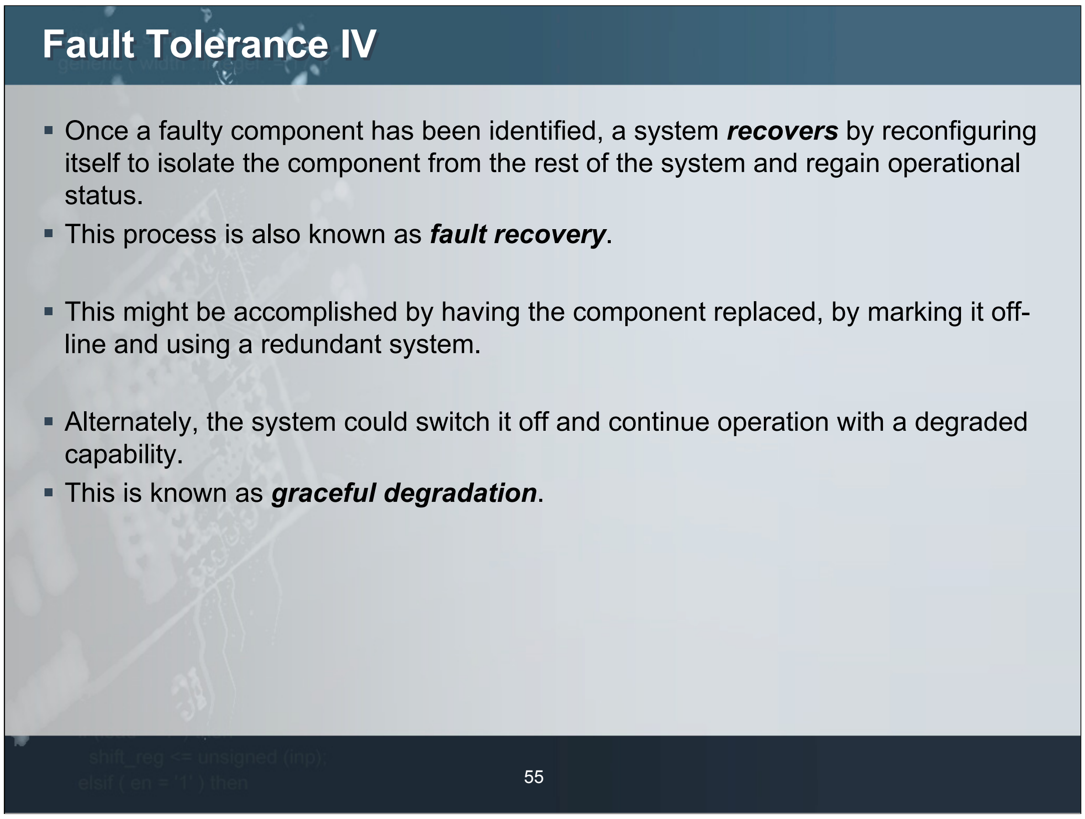
53



Fault Tolerance III

- ***Fault location*** is the process of determining where a fault has occurred.
  - A failed acceptance test cannot generally be used to locate a fault. It can only tell that something has gone wrong.
  - Similarly, when a disagreement occurs during the comparison of two modules, it is not possible to tell which of the two has failed.
- ***Fault containment*** is the process of isolating a fault and preventing the propagation of the effect of that fault throughout the system.
- The purpose is to limit the spread of the effects of a fault from one area of the system into another area.
- This is typically achieved by frequent fault detection, by multiple request/confirmation protocols and by performing consistency checks between modules.

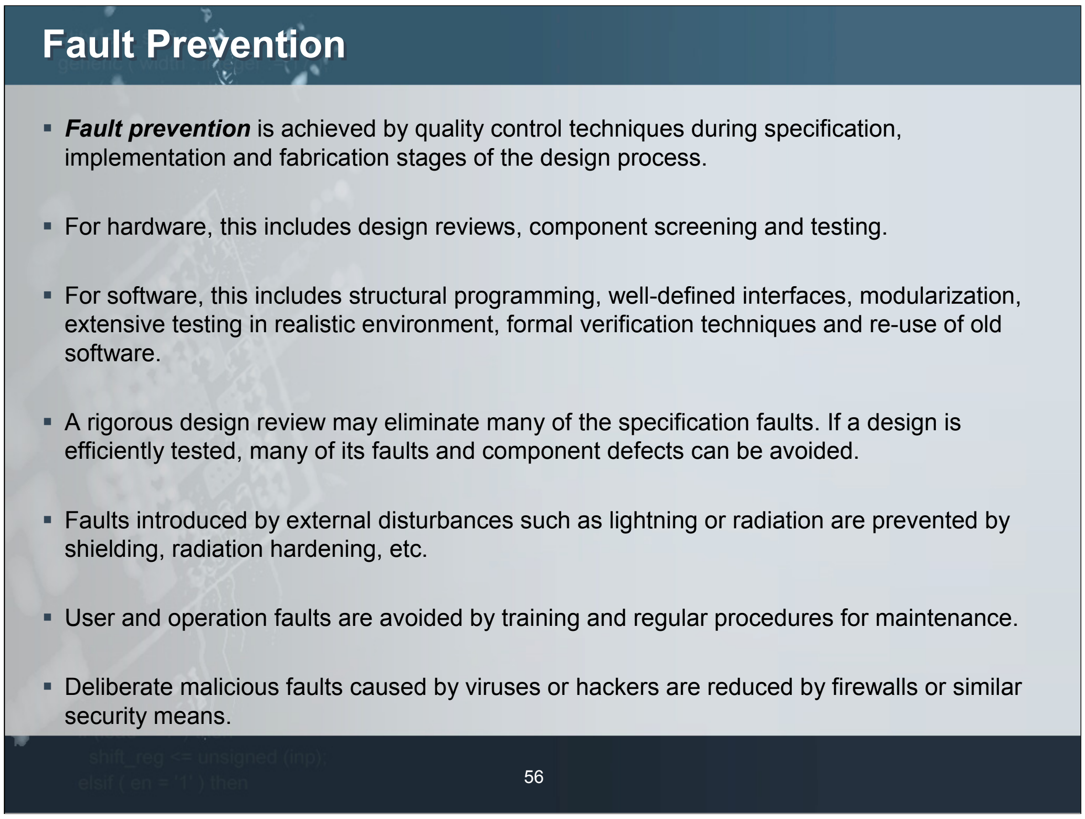
54



Fault Tolerance IV

- Once a faulty component has been identified, a system ***recovers*** by reconfiguring itself to isolate the component from the rest of the system and regain operational status.
- This process is also known as ***fault recovery***.
- This might be accomplished by having the component replaced, by marking it off-line and using a redundant system.
- Alternately, the system could switch it off and continue operation with a degraded capability.
- This is known as ***graceful degradation***.

55



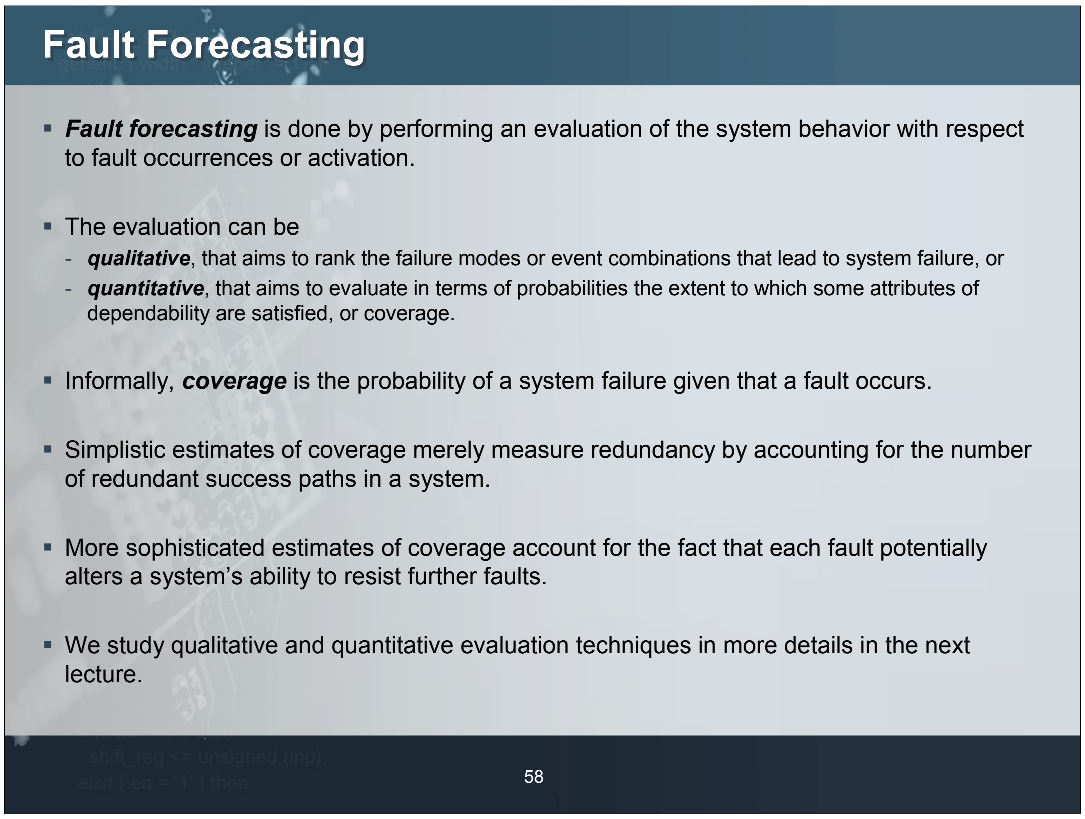
Fault Prevention

- ***Fault prevention*** is achieved by quality control techniques during specification, implementation and fabrication stages of the design process.
- For hardware, this includes design reviews, component screening and testing.
- For software, this includes structural programming, well-defined interfaces, modularization, extensive testing in realistic environment, formal verification techniques and re-use of old software.
- A rigorous design review may eliminate many of the specification faults. If a design is efficiently tested, many of its faults and component defects can be avoided.
- Faults introduced by external disturbances such as lightning or radiation are prevented by shielding, radiation hardening, etc.
- User and operation faults are avoided by training and regular procedures for maintenance.
- Deliberate malicious faults caused by viruses or hackers are reduced by firewalls or similar security means.

56

Fault Removal

- ***Fault removal*** is performed during the development phase as well as during the operational life of a system.
- During the development phase, fault removal consists of three steps: **verification**, **diagnosis** and **correction**.
- Fault removal during the operational life of the system consists of **corrective** and **preventive maintenance**.
- *Verification* is the process of checking whether the system meets a set of given conditions. If it does not, the other two steps follow: the fault that prevents the conditions from being fulfilled is diagnosed and the necessary corrections are performed.
- In ***preventive maintenance***, parts are replaced, or adjustments are made before failure occurs. The objective is to increase the dependability of the system over the long term by staving off the aging effects of wear-out.
- In contrast, ***corrective maintenance*** is performed after the failure has occurred in order to return the system to service as soon as possible.



Fault Forecasting

- ***Fault forecasting*** is done by performing an evaluation of the system behavior with respect to fault occurrences or activation.
- The evaluation can be
  - ***qualitative***, that aims to rank the failure modes or event combinations that lead to system failure, or
  - ***quantitative***, that aims to evaluate in terms of probabilities the extent to which some attributes of dependability are satisfied, or coverage.
- Informally, ***coverage*** is the probability of a system failure given that a fault occurs.
- Simplistic estimates of coverage merely measure redundancy by accounting for the number of redundant success paths in a system.
- More sophisticated estimates of coverage account for the fact that each fault potentially alters a system's ability to resist further faults.
- We study qualitative and quantitative evaluation techniques in more details in the next lecture.

58