# НАДЕЖДНОСТ И СИГУРНОСТ НА СОФТУЕРНИ СИСТЕМИ

СУ „Св. Климент Охридски“
Факултет по Математика и Информатика
Увод в Софтуерното Инженерство

# 70s: Mainframes

**Systems/users:**

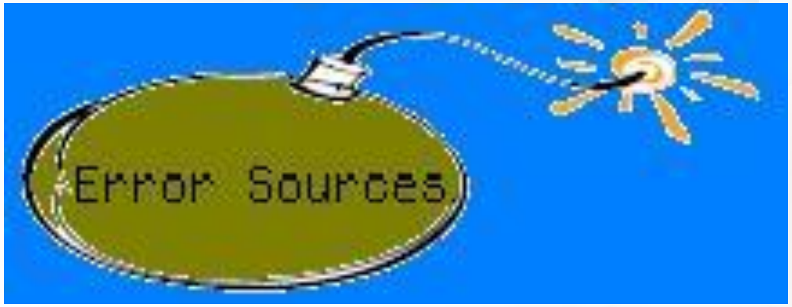
~$10^4$

**User competency:**

Engineers

**Integration complexity**

- Close systems
- Highly custom designs
- Hardware and software fully controlled by vendors



- Hardware

# 80s: Workstations

**Systems/users:**

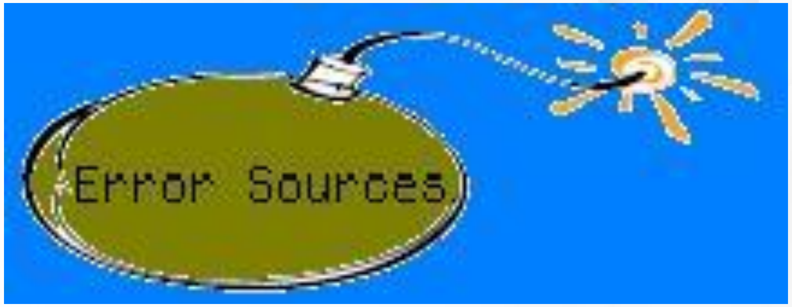
$\sim 10^6$

**User competency:**

Basic knowledge

**Integration complexity**

- Mostly close systems
- Network connectivity
- Standard interfaces exported for users



- Hardware
- Network

# 90s: Personal Computers

**Systems/users:**

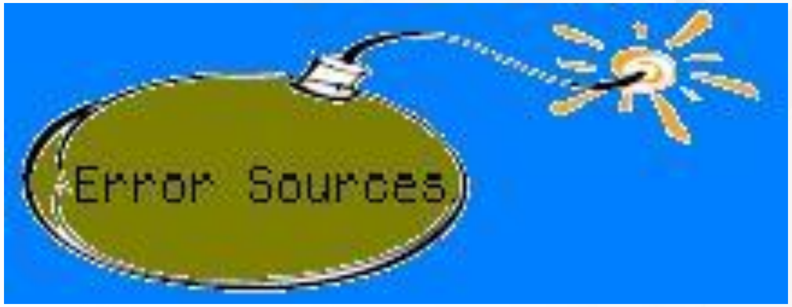
~$10^7$

**User competency:**

Computer Literacy

## Integration complexity

- Open systems
- Wide network access
- Commercial OS
- Third party software and hardware



- Hardware, Software
- Network
- Human mistakes

# 2000s: Mobile Devices

**Systems/users:**

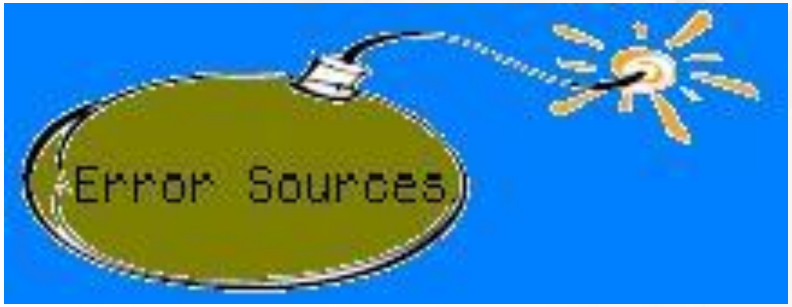
~$10^8$

**User competency:**

Undefined

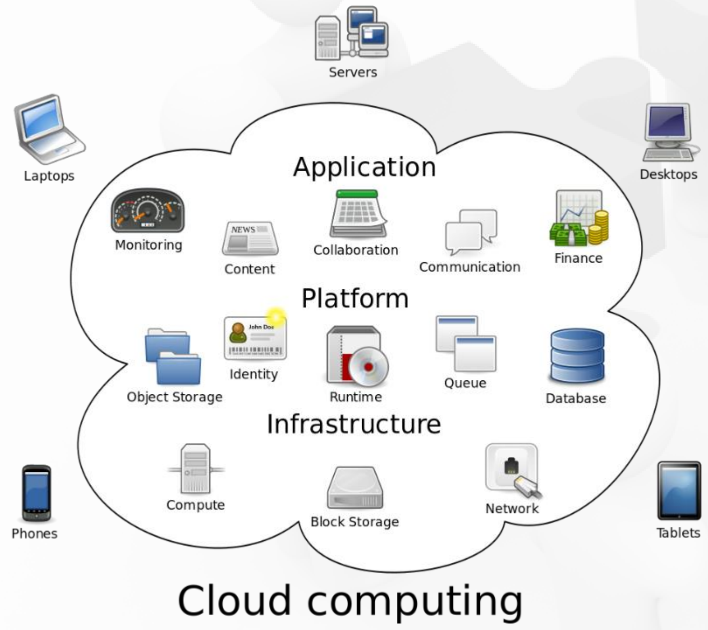
## Integration complexity

- Open systems
- COTS/proprietary OS
- Highly integrated computer systems
- Wider range of networks



- Hardware, Software
- Network (wired/wireless)
- Human mistakes
- Malicious faults

# 2010s: Cloud computing

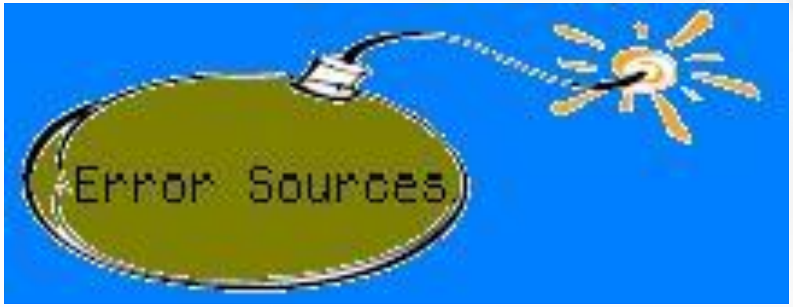


**Systems/users:**

~$10^9$ and more

**User competency:**

Undefined



Human mistakes

Malicious faults

Software itself

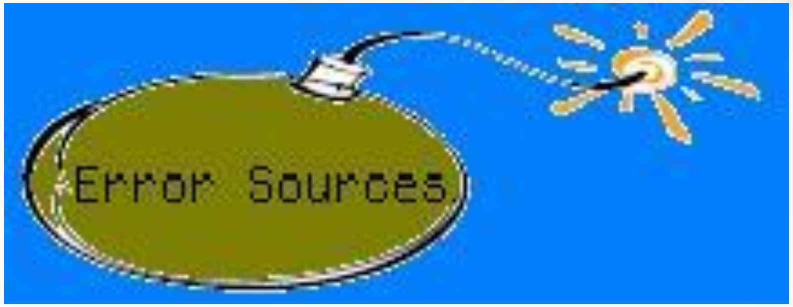
# 2020s: Artificial Intelligence

**Systems/users:**

???

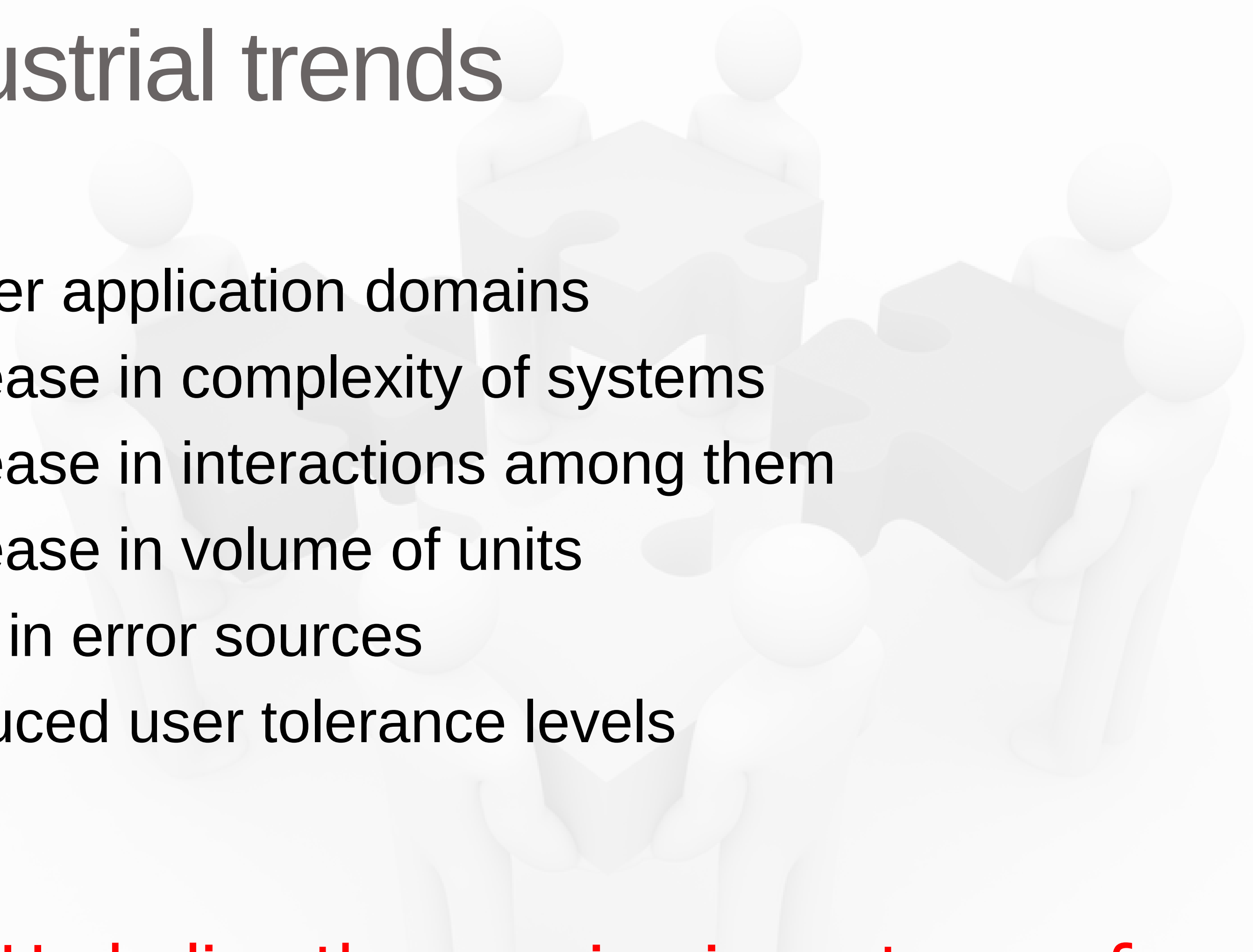
**User competency:**

???



???

# Industrial trends

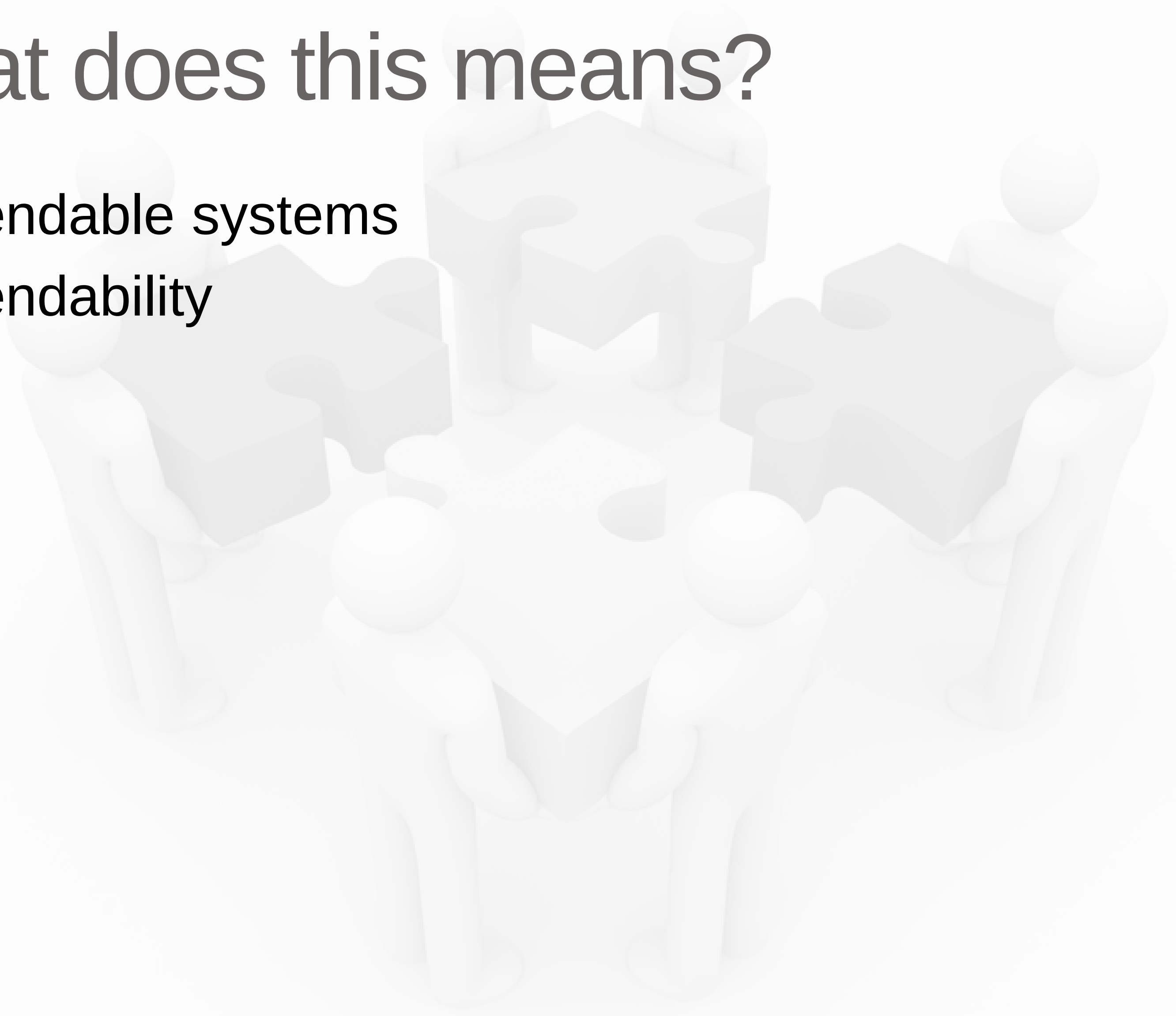


- Newer application domains
- Increase in complexity of systems
- Increase in interactions among them
- Increase in volume of units
- Shift in error sources
- Reduced user tolerance levels

Underline the growing importance of building dependable systems

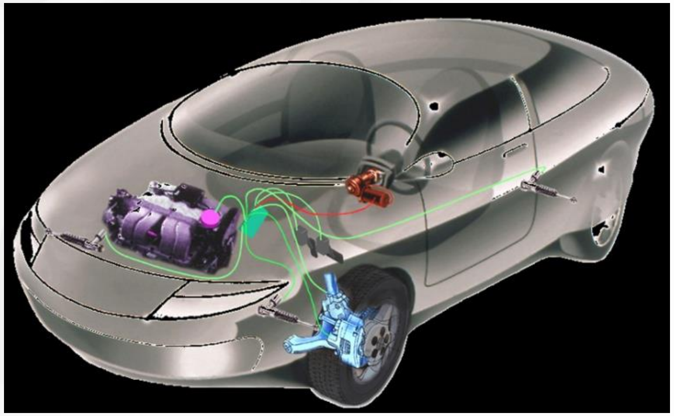
# What does this means?

- Dependable systems
- Dependability

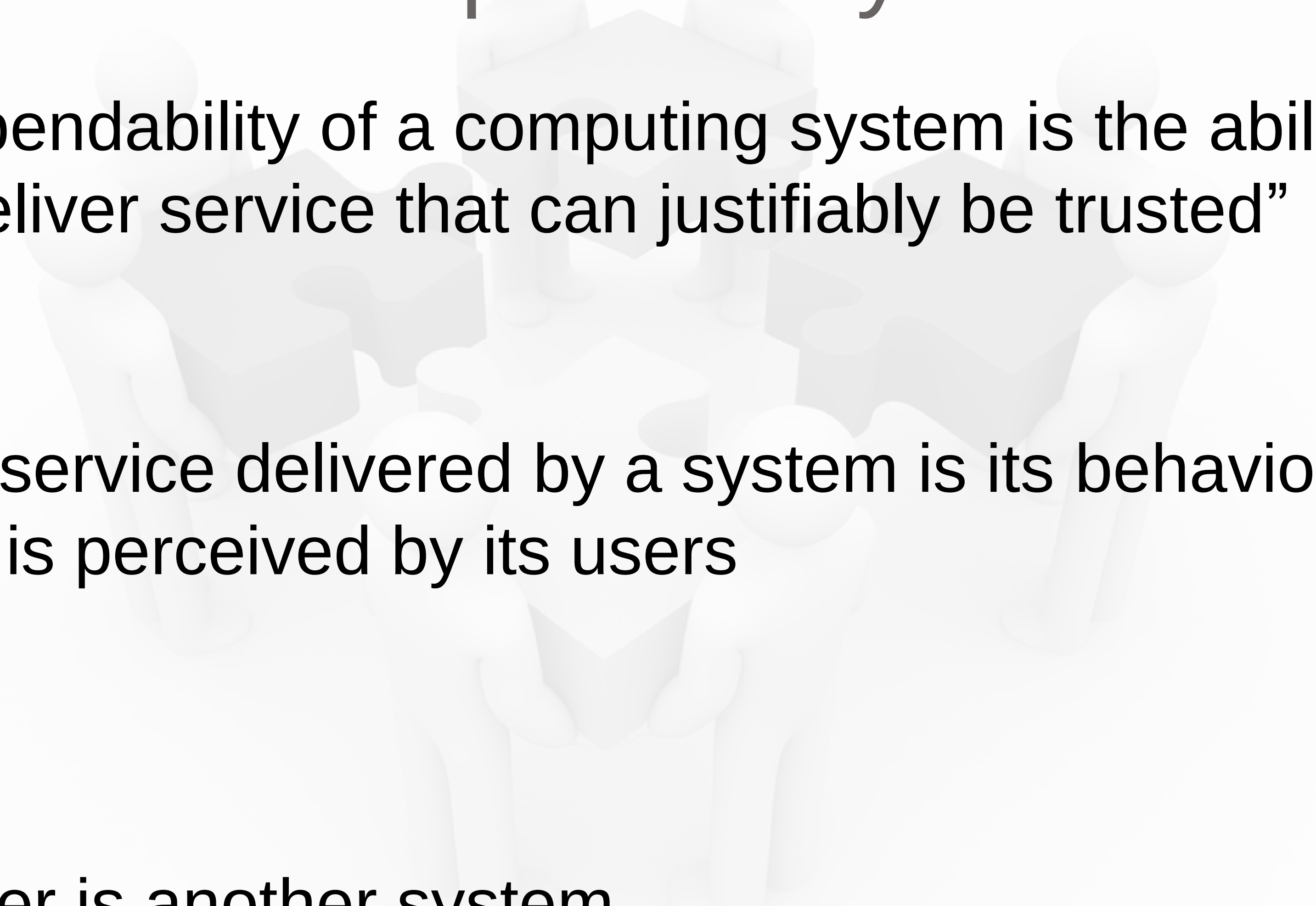


# Nonfunctional requirements

- Define HOW software should perform its functionality
- Also  known as “-ilities”
  - Dependability
  - Testability
  - Usability
  - Modifyability
  - Etc.



# Definition of dependability

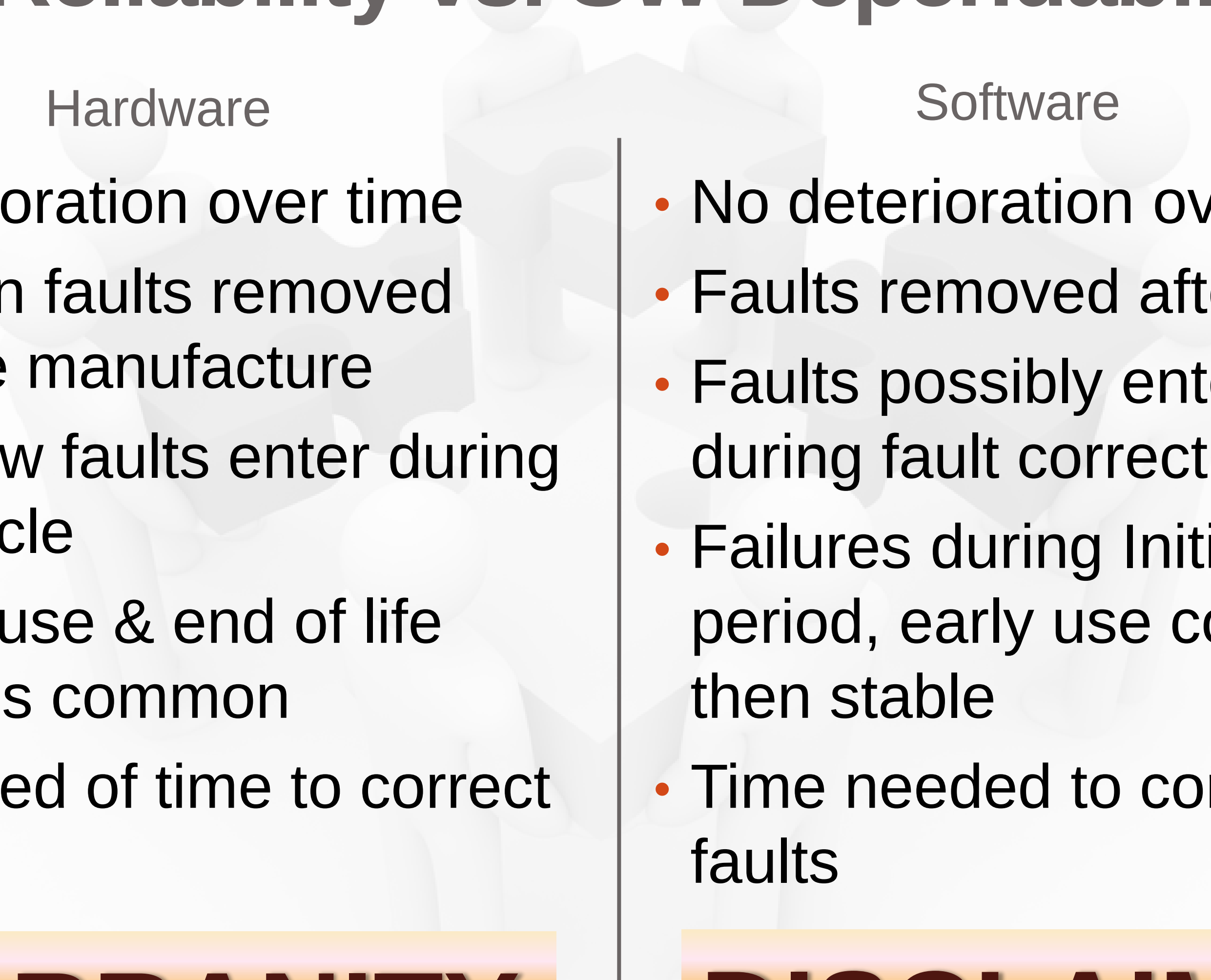


“Dependability of a computing system is the ability to deliver service that can justifiably be trusted”

The service delivered by a system is its behaviour as it is perceived by its users

A user is another system (physical,human)

# HW Reliability vs. SW Dependability



## Hardware

- Deterioration over time
- Design faults removed before manufacture
- No new faults enter during life-cycle
- Initial use & end of life failures common
- No need of time to correct fault

**WARRANITY**

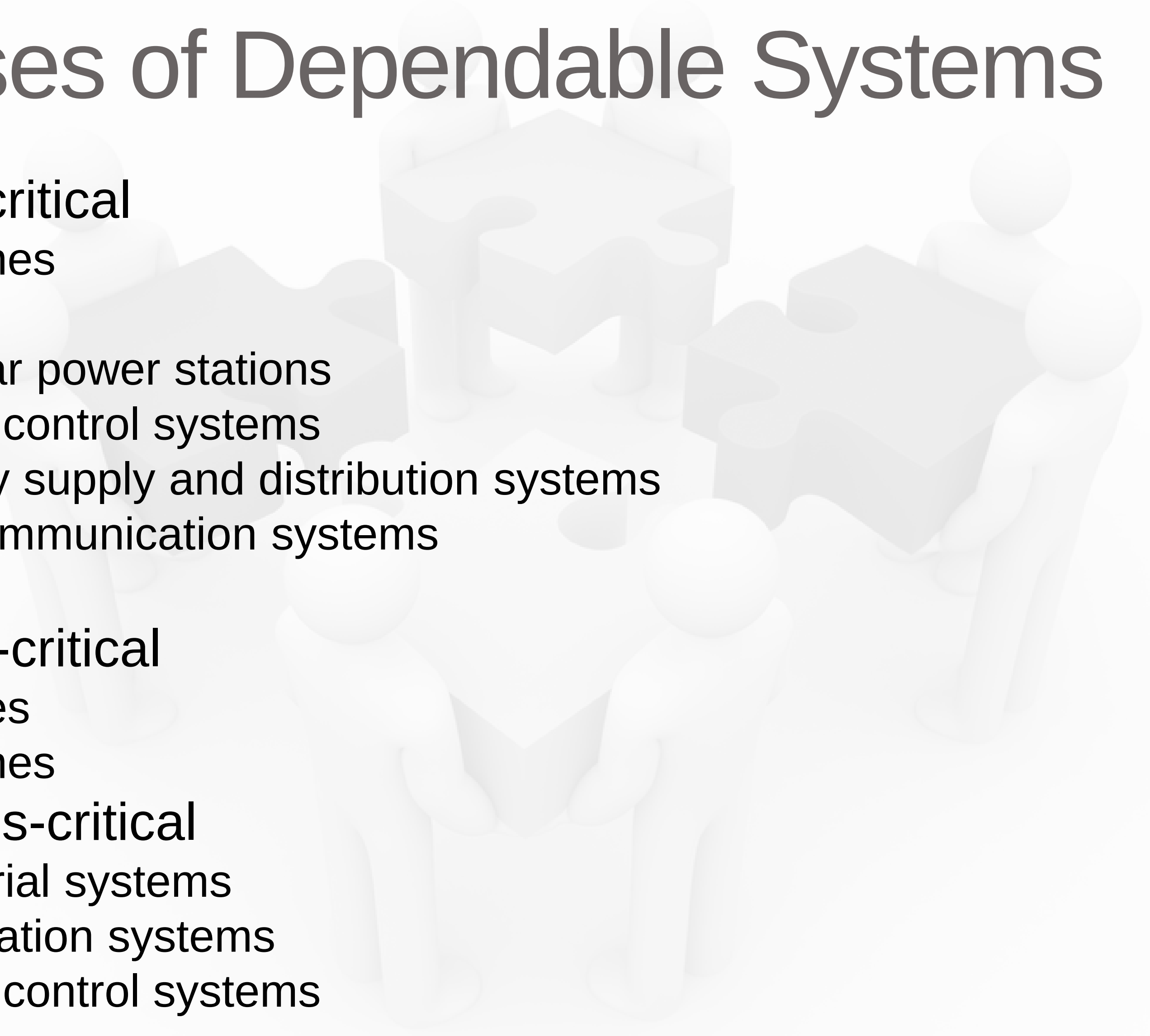
## Software

- No deterioration over time
- Faults removed after build
- Faults possibly enter during fault correction
- Failures during Initial test period, early use common, then stable
- Time needed to correct faults

**DISCLAIMER**

# Classes of dependable systems

# Classes of Dependable Systems

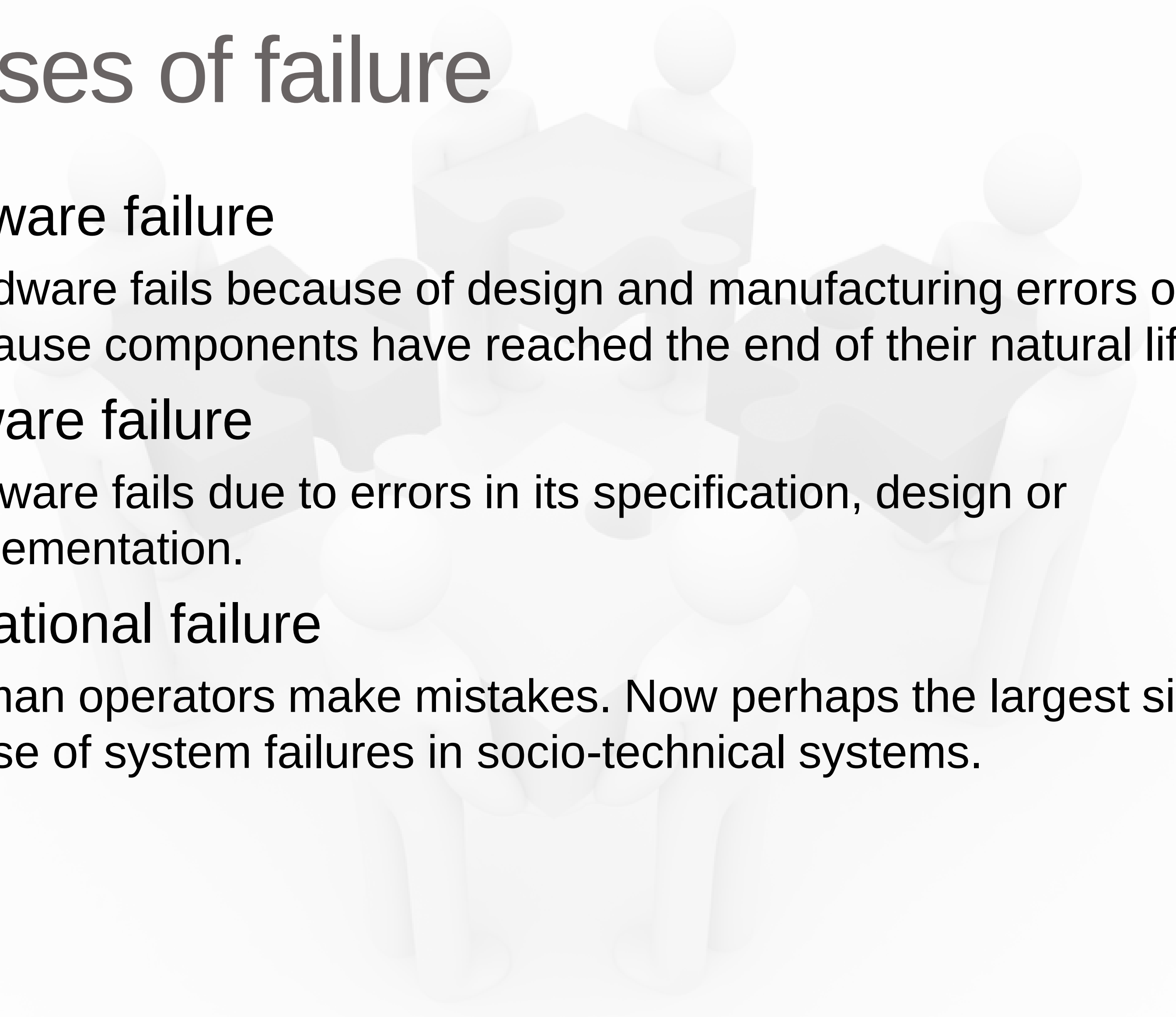


- Safety-critical
  - Airplanes
  - Cars
  - Nuclear power stations
  - Traffic control systems
  - Energy supply and distribution systems
  - Telecommunication systems
  - Etc…
- Mission-critical
  - Shuttles
  - Airplanes
- Business-critical
  - Industrial systems
  - Information systems
  - Traffic control systems

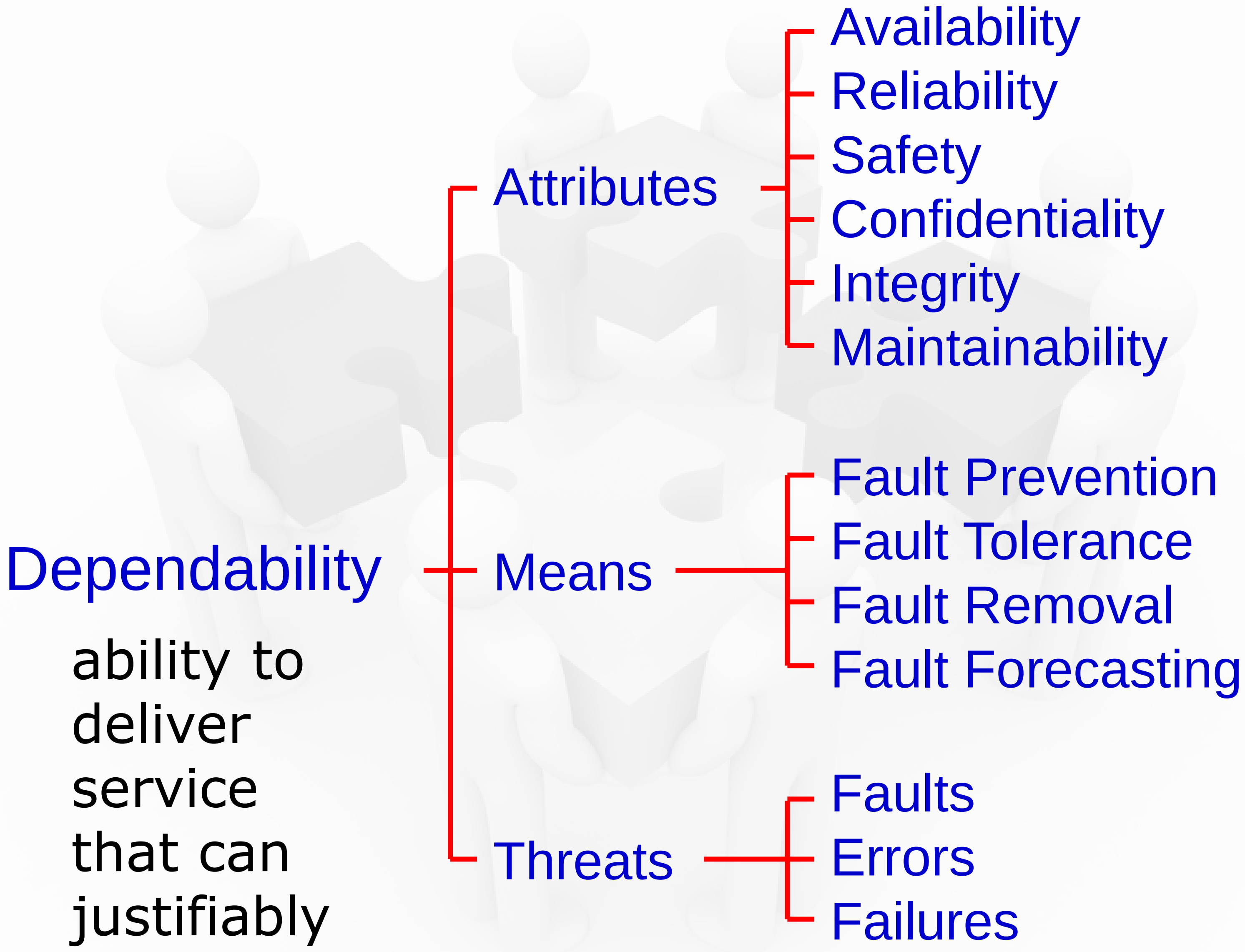
# Importance of dependability

- System failures may have widespread effects with large numbers of people affected by the failure.
- Systems that are not dependable and are unreliable, unsafe or insecure may be rejected by their users.
- The costs of system failure may be very high if the failure leads to economic losses or physical damage.
- Undependable systems may cause information loss with a high consequent recovery cost.

# Causes of failure



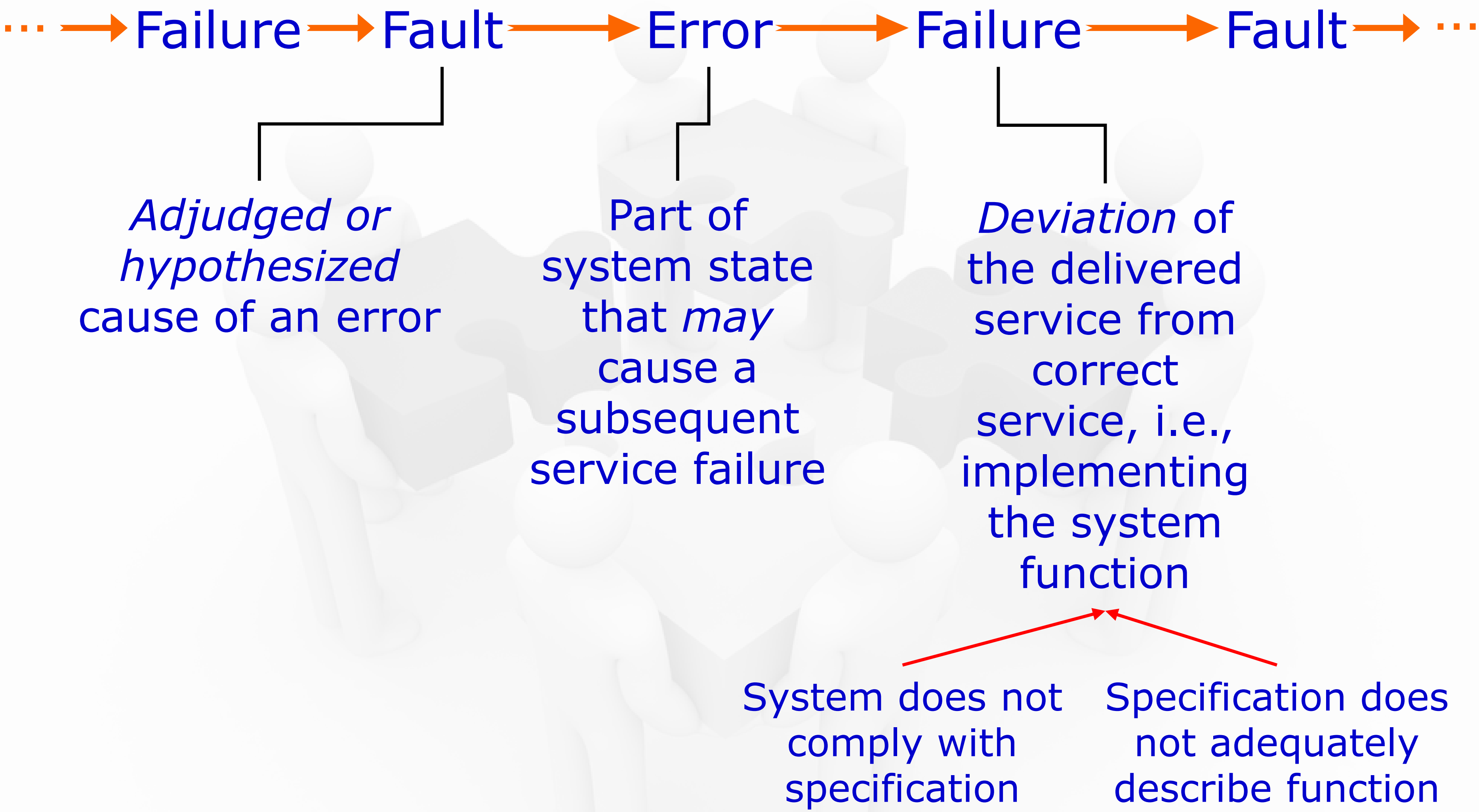
- Hardware failure
  - Hardware fails because of design and manufacturing errors or because components have reached the end of their natural life.
- Software failure
  - Software fails due to errors in its specification, design or implementation.
- Operational failure
  - Human operators make mistakes. Now perhaps the largest single cause of system failures in socio-technical systems.



Dependability

- Attributes
  - Availability
  - Reliability
  - Safety
  - Confidentiality
  - Integrity
  - Maintainability
- Means
  - Fault Prevention
  - Fault Tolerance
  - Fault Removal
  - Fault Forecasting
- Threats
  - Faults
  - Errors
  - Failures

ability to deliver service that can justifiably be trusted



… → Failure → Fault → Error → Failure → Fault → …

Fault: *Adjudged or hypothesized* cause of an error

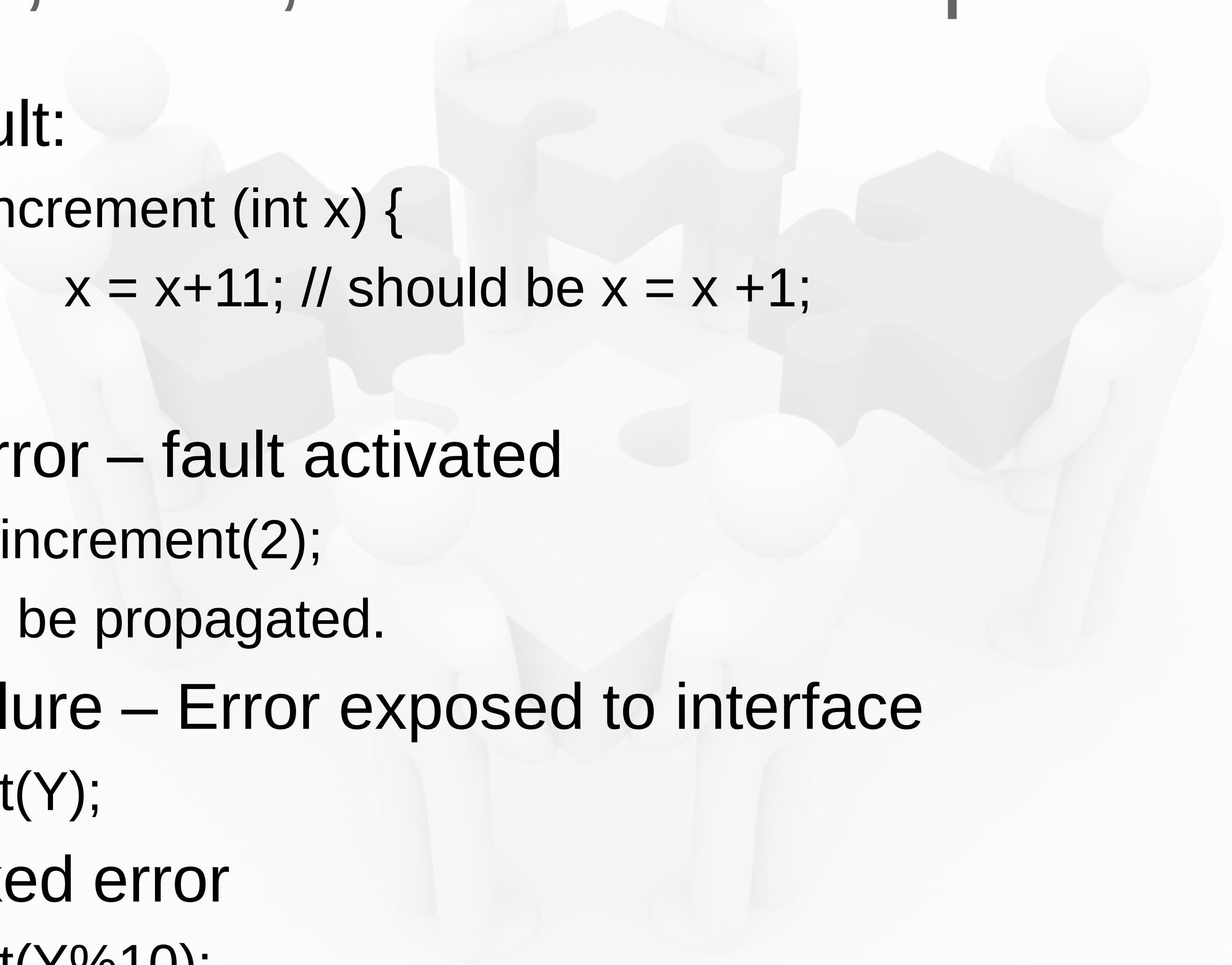
Error: Part of system state that *may* cause a subsequent service failure

Failure: *Deviation* of the delivered service from correct service, i.e., implementing the system function

System does not comply with specification

Specification does not adequately describe function

# Fault, Error, Failure - Example

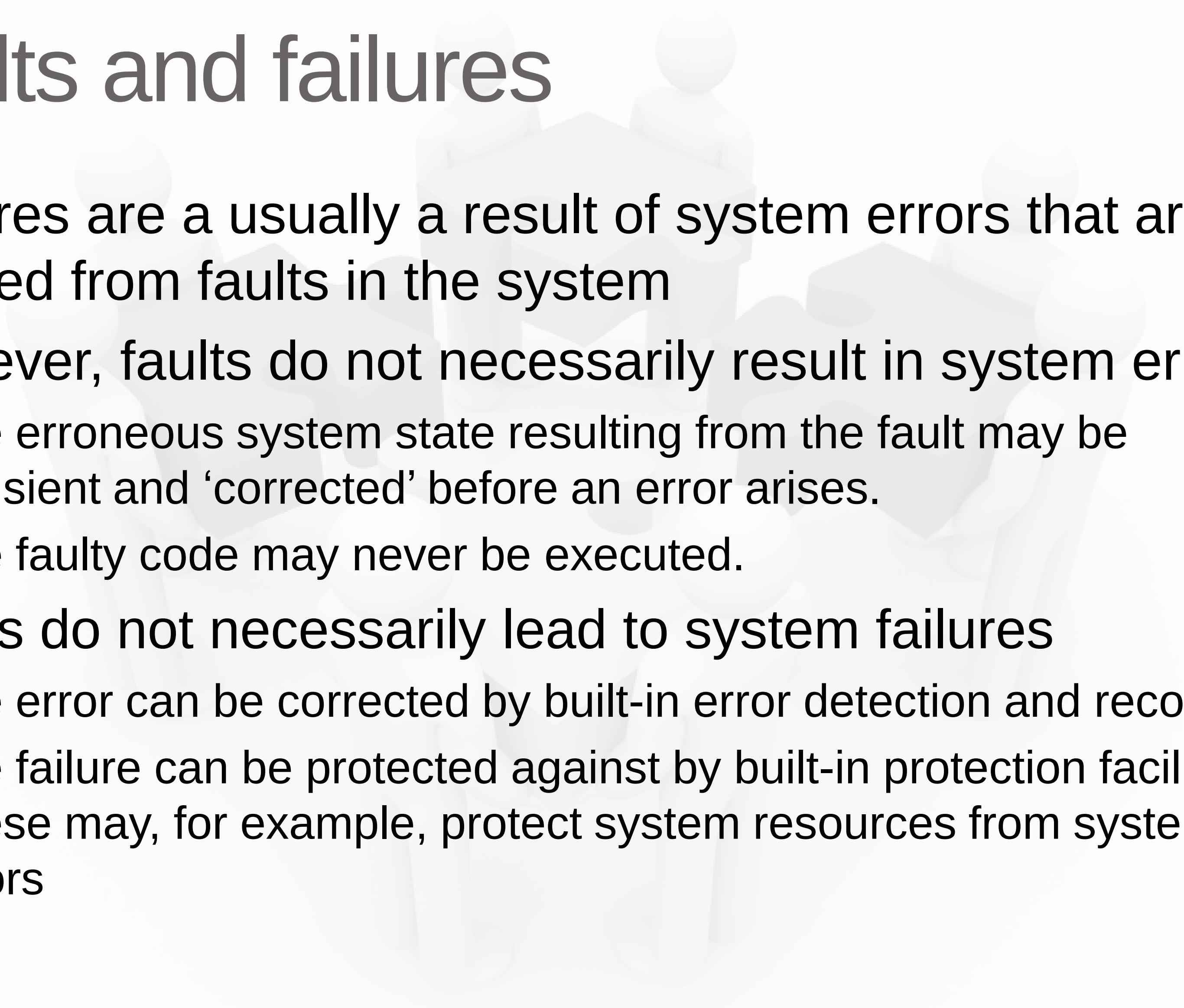


- A Fault:
  - int increment (int x) {
    ```
            x = x+11; // should be x = x +1;
    }
    ```
- An Error – fault activated
  - Y = increment(2);
  - Can be propagated.
- A Failure – Error exposed to interface
  - Print(Y);
- Masked error
  - Print(Y%10);

# Faults nature

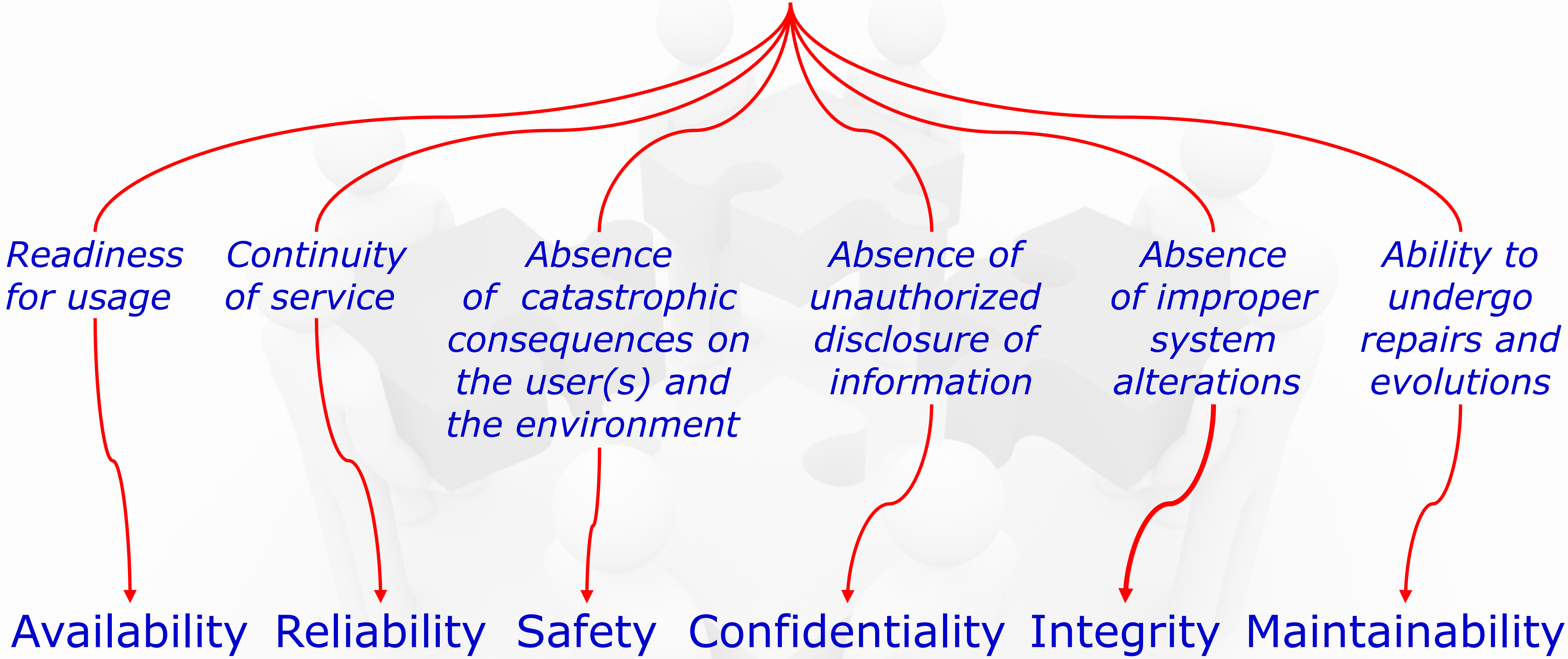
- Not all code in a program is executed. The code that includes a fault (e.g., the failure to initialize a variable) may never be executed because of the way that the software is used.
- Errors may be transient. A state variable may have an incorrect value caused by the execution of faulty code. However, before this is accessed and causes a system failure, some other system input may be processed that resets the state to a valid value.
- The system may include fault detection and protection mechanisms. These ensure that the erroneous behavior is discovered and corrected before the system services are affected.

# Faults and failures



- Failures are a usually a result of system errors that are derived from faults in the system
- However, faults do not necessarily result in system errors
  - The erroneous system state resulting from the fault may be transient and ‘corrected’ before an error arises.
  - The faulty code may never be executed.
- Errors do not necessarily lead to system failures
  - The error can be corrected by built-in error detection and recovery
  - The failure can be protected against by built-in protection facilities. These may, for example, protect system resources from system errors

# Dependability



*Readiness for usage*

Availability

*Continuity of service*

Reliability

*Absence of catastrophic consequences on the user(s) and the environment*

Safety

*Absence of unauthorized disclosure of information*

Confidentiality

*Absence of improper system alterations*

Integrity

*Ability to undergo repairs and evolutions*

Maintainability

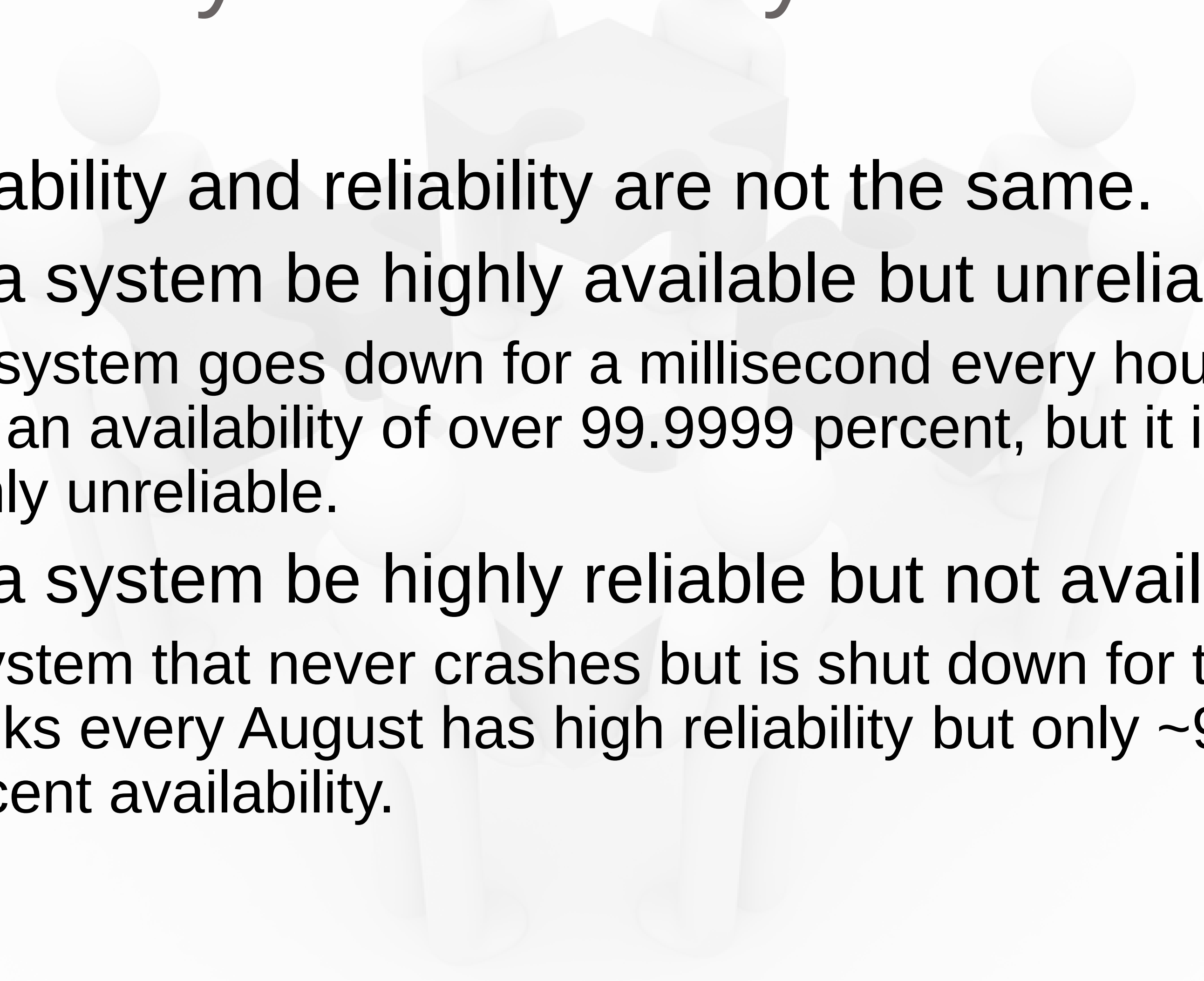
Dependability: Ability of the system to provide service that can justifiably trusted

# Strict definitions

- **Reliability:** The probability of failure-free  (as per specification) operation over a specified time, in a given environment, for a specific purpose.
  - Depends on the environment

- **Availability:** The probability that a system, at a point in time, will be operational and able to deliver the requested services.
  - Does not just depend on the number of system crashes, but also on the time needed to repair the faults that have caused the failure.

# Availability vs. Reliability



- Availability and reliability are not the same.
- Can a system be highly available but unreliable?
  - If a system goes down for a millisecond every hour, it has an availability of over 99.9999 percent, but it is still highly unreliable.
- Can a system be highly reliable but not available?
  - A system that never crashes but is shut down for two weeks every August has high reliability but only ~96 percent availability.

# Measure for availability

- Availability of a system is defined as the probability that it is available (i.e. dependable) when there exists a need for access to its functionality
- Availability may be measured as:

$$\alpha = \frac{\Delta t_f}{\Delta t_f + \Delta t_c},$$

- Where $\Delta t_f$ is the *Mean time between failures* and $\Delta t_c$ is the *Mean time to repair*

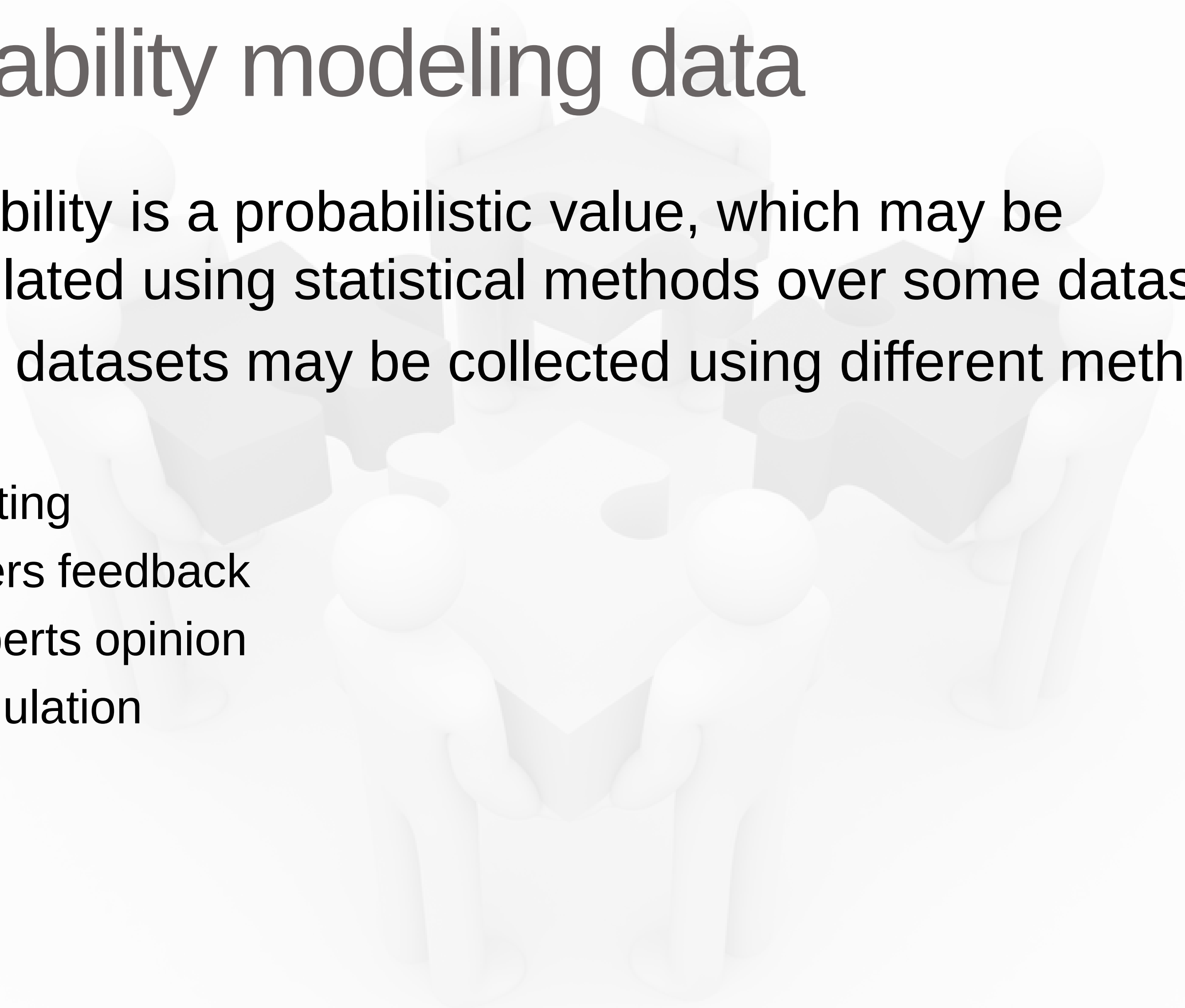
# Availability perception

- Availability is usually expressed as a percentage of the time that the system is available to deliver services e.g. 99.95%.
- However, this does not take into account two factors:
  - The number of users affected by the service outage. Loss of service in the middle of the night is less important for many systems than loss of service during peak usage periods.
  - The length of the outage. The longer the outage, the more the disruption. Several short outages are less likely to be disruptive than 1 long outage. Long repair times are a particular problem.

# Software reliability

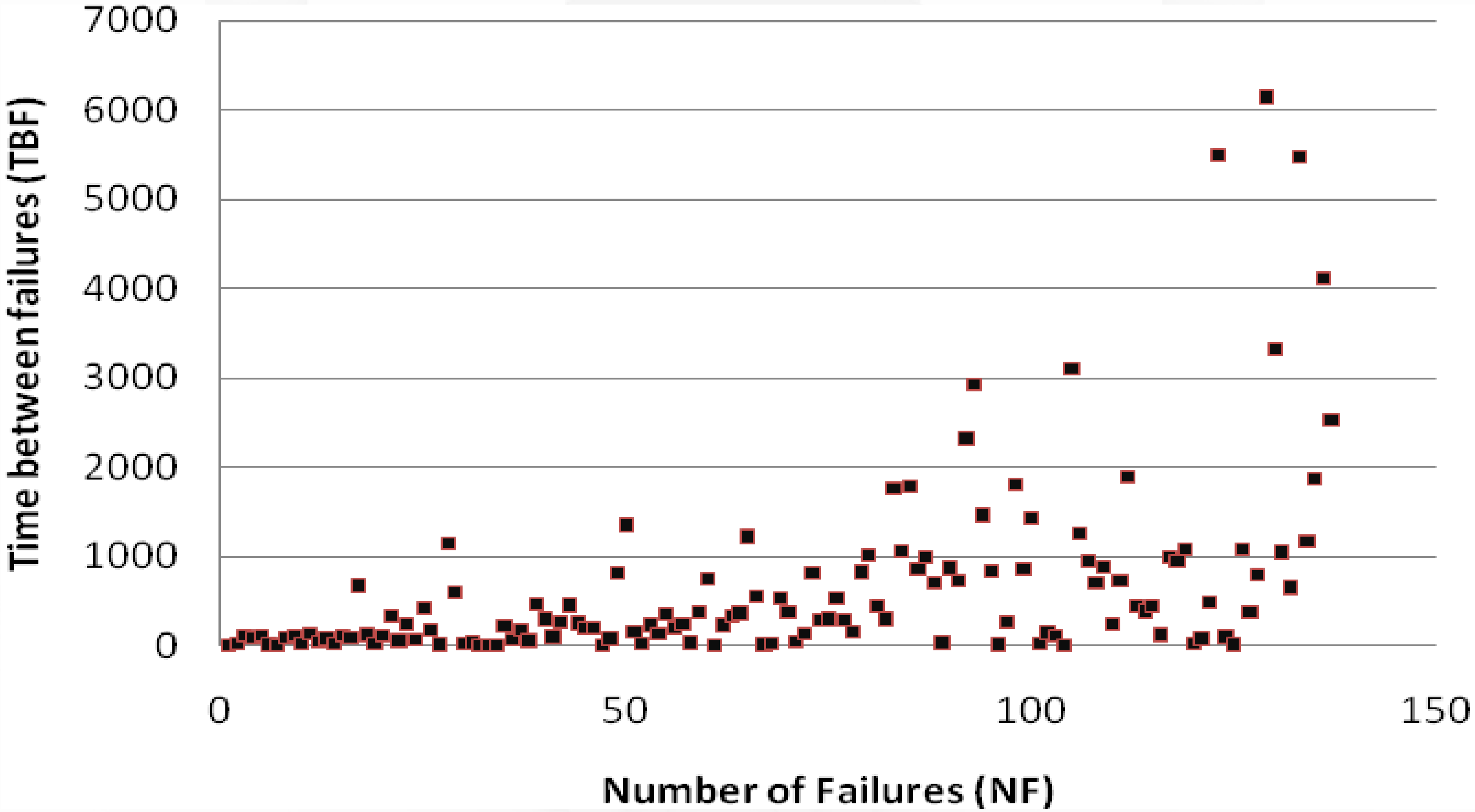
- Slightly different from traditional (hardware) reliability theory
  - Failure is deterministic, user behaviour is not
- Probabilistic value
  - Probability of failure (success)
  - Mean time to failure (μ)
  - Failure rate λ
    - λ = 1/μ

# Reliability modeling data

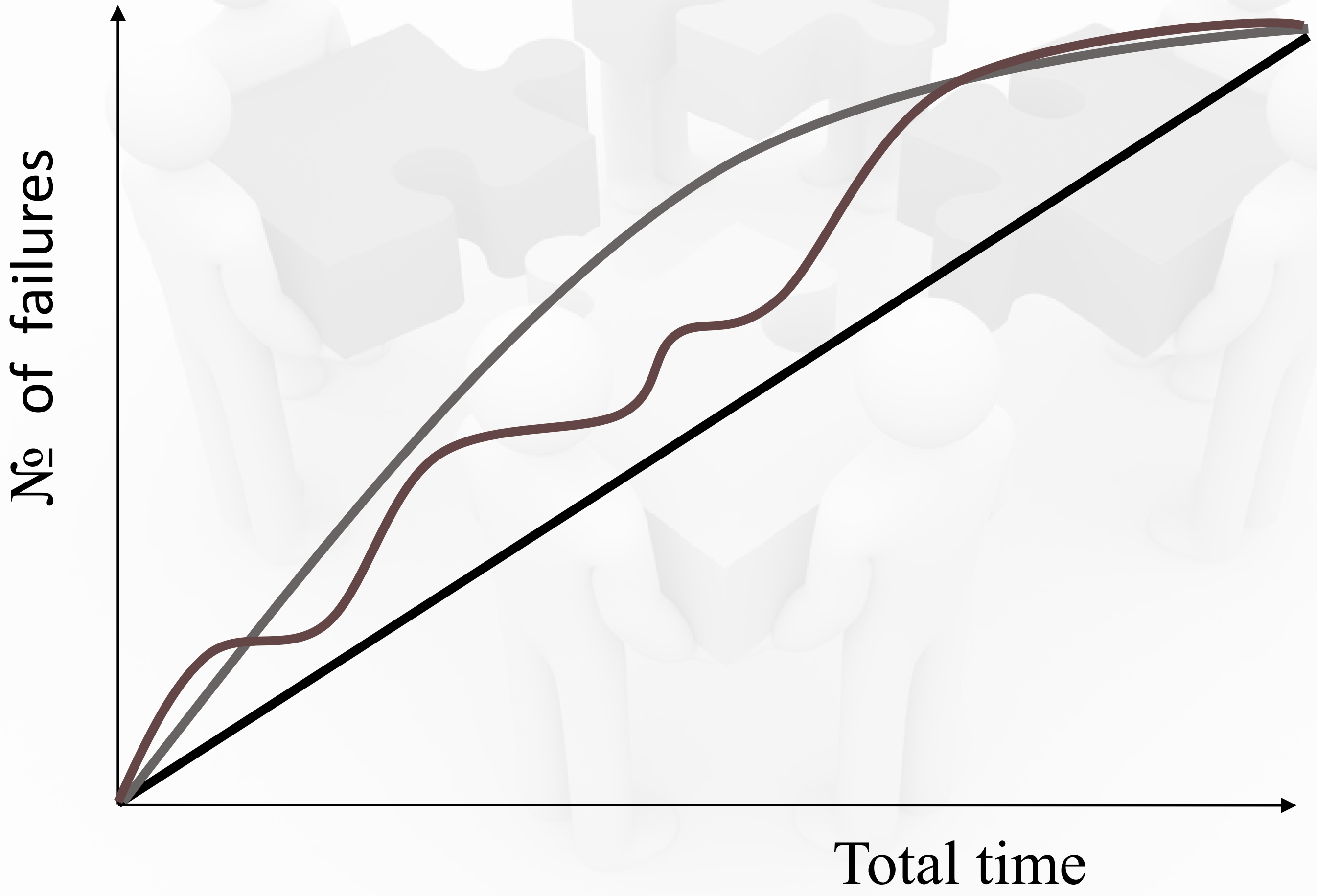


- Reliability is a probabilistic value, which may be calculated using statistical methods over some datasets
- Such datasets may be collected using different methods, like:
  - Testing
  - Users feedback
  - Experts opinion
  - Simulation

# Typical failure data set



# Kinds of software system failure behaviours



# Reliability estimation models

- White box models
  - Build an architectural model of the system
  - Integrate failure behaviour of individual components with architectural model
- Black box models
  - Statistic processing of data
  - Software Reliability Growth Models (SRGMs)

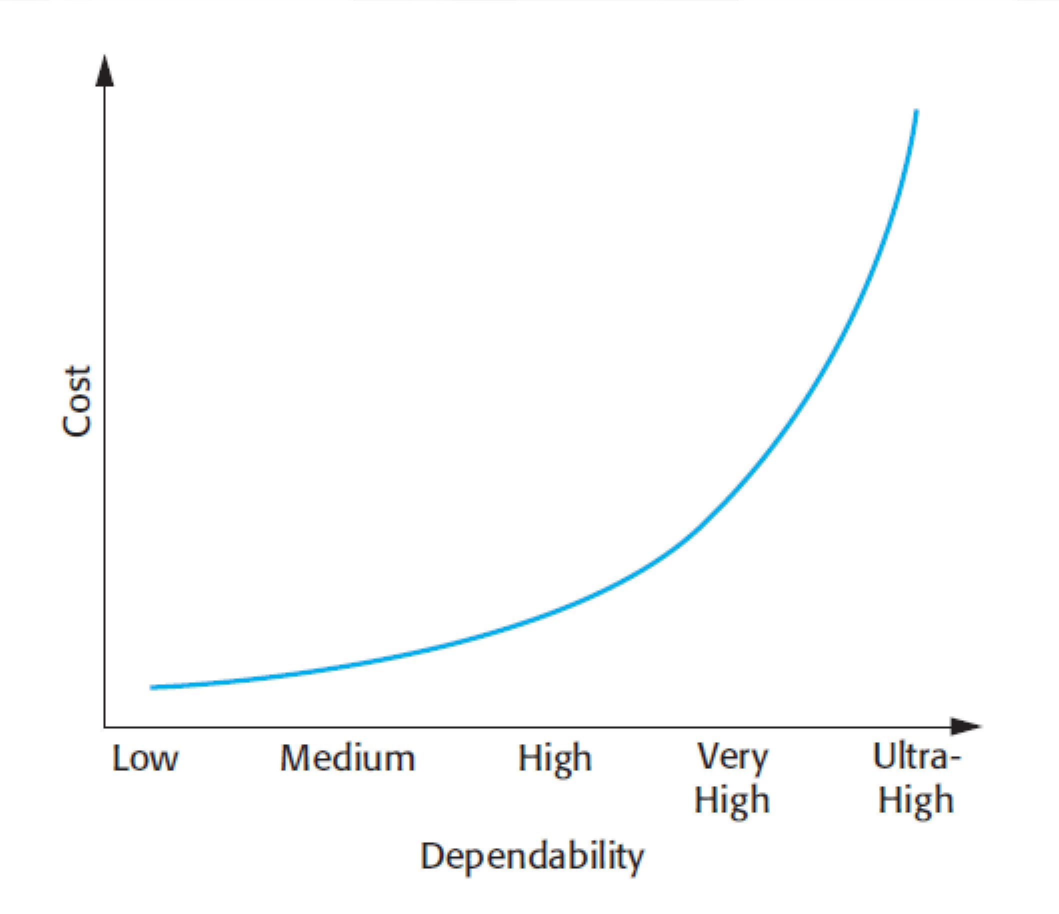
# Reliability in use

- Removing X% of the faults in a system will not necessarily improve the reliability by X%. A study at IBM showed that removing 60% of product defects resulted in a 3% improvement in reliability.
- Program defects may be in rarely executed sections of the code so may never be encountered by users. Removing these does not affect the perceived reliability.
- Users adapt their behaviour to avoid system features that may fail for them.
- A program with known faults may therefore still be perceived as reliable by its users.

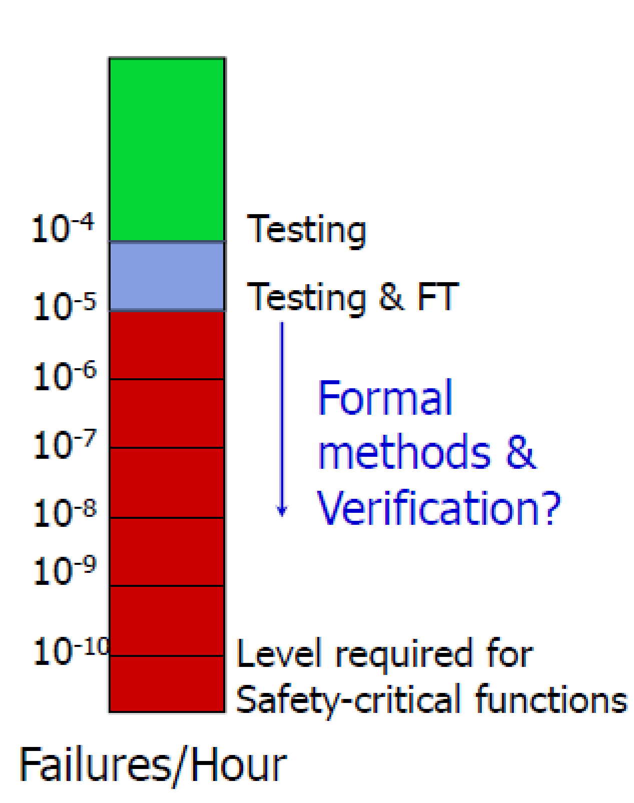
# The problem with testing

- Reliability of life-critical and real-time software is infeasible to be quantified by testing [Butler & Finelli 1993]
- Only small reliabilities (99,999%) are possible to be estimated in a obtainable period of time for testing
- For example assuring that a program has failure rate of about $10^{-7}$ per hour may require thousands (and even more) years of testing
- There may not exist effective oracle to carry out statistical testing

# Cost of dependability/reliability

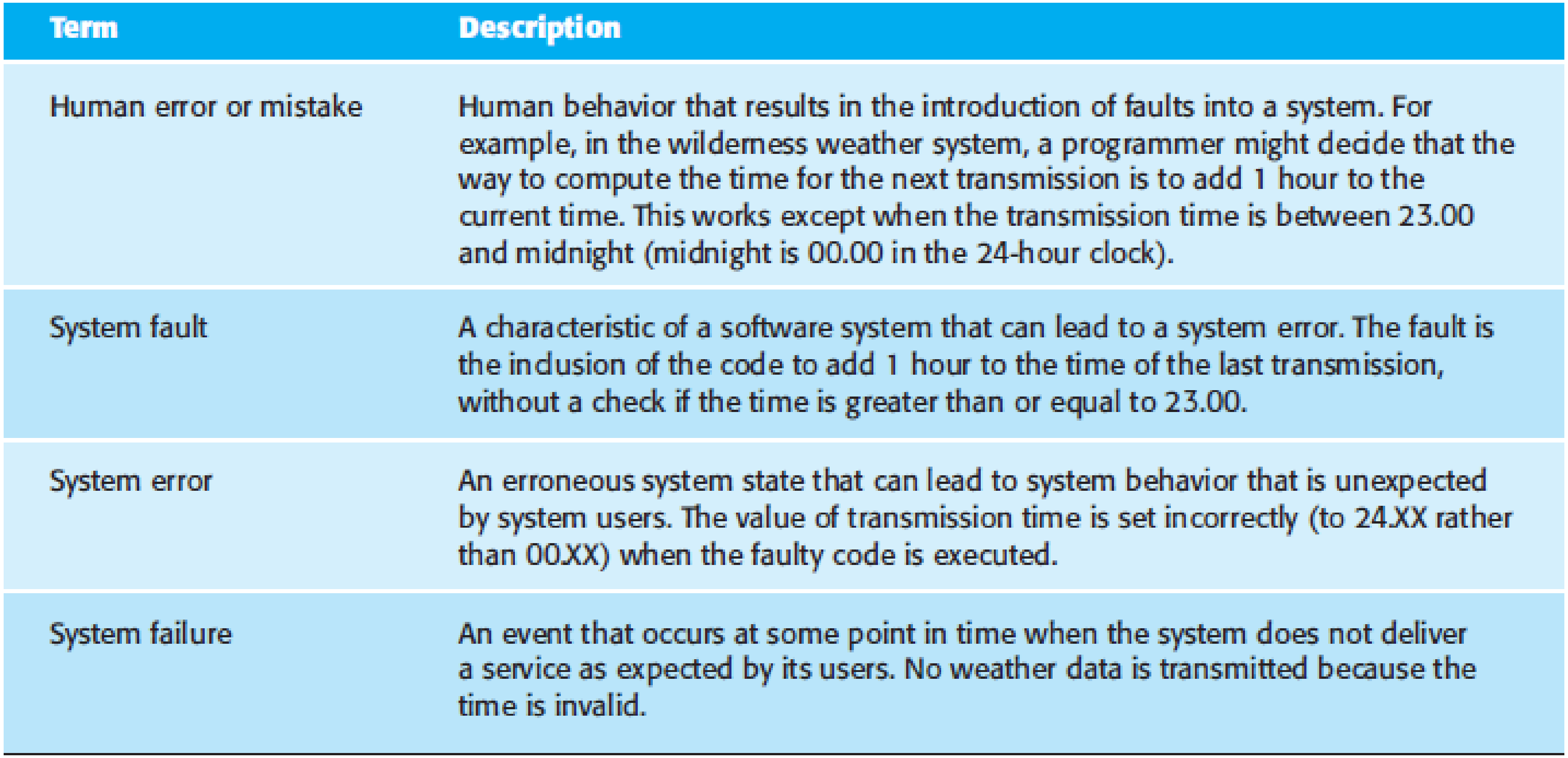


# Attainable levels of SW reliability


10-4
10-5
10-6
10-7
10-8
10-9
10-10
Testing
Testing & FT
Formal methods & Verification?
Level required for Safety-critical functions
Failures/Hour

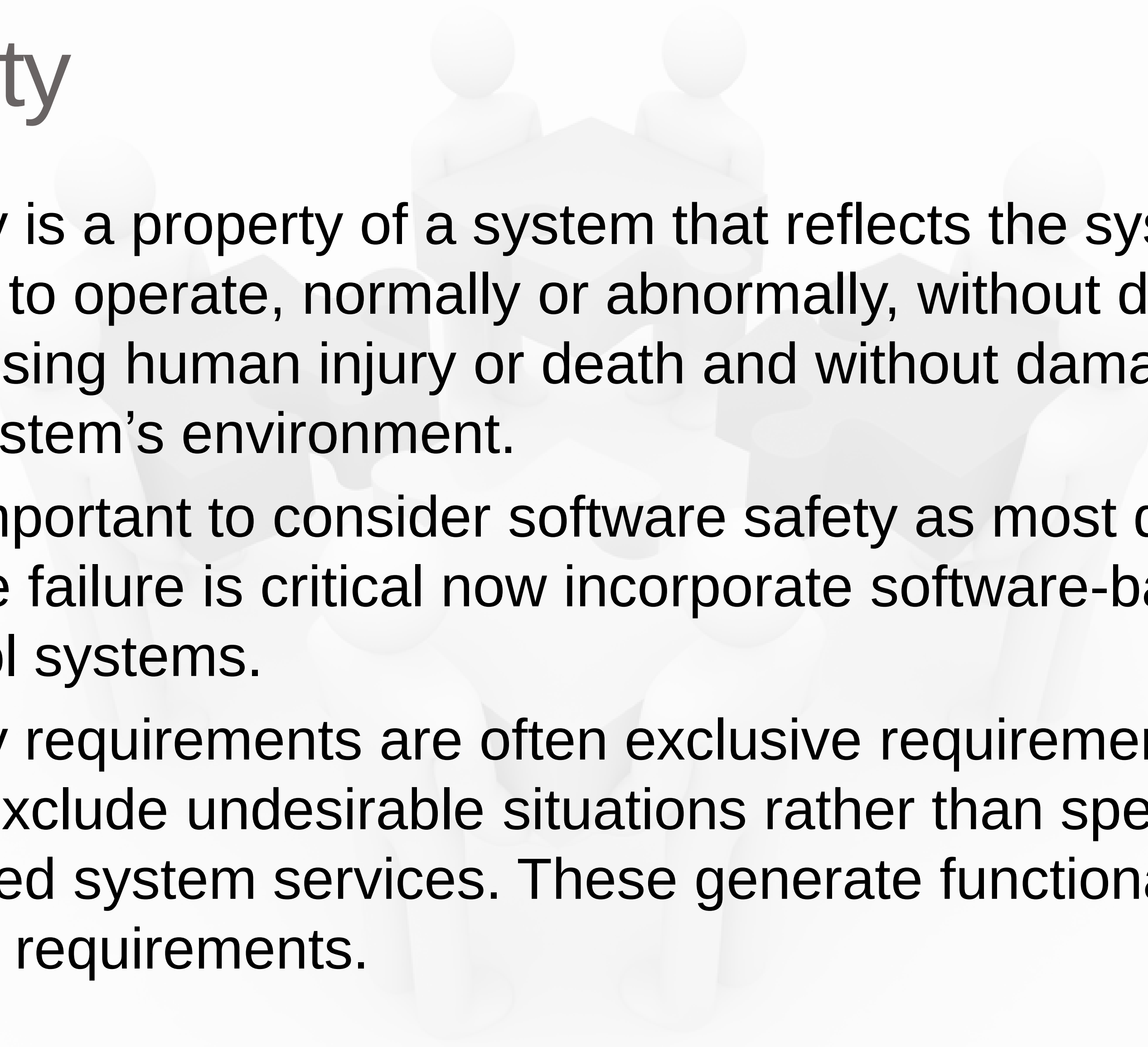
- FAA (Federal Aviation Administration) & NASA safety=critical requirement is less than $10^{-10}$ failures per 10 hrs of flight.

# Reliability/availability terminology



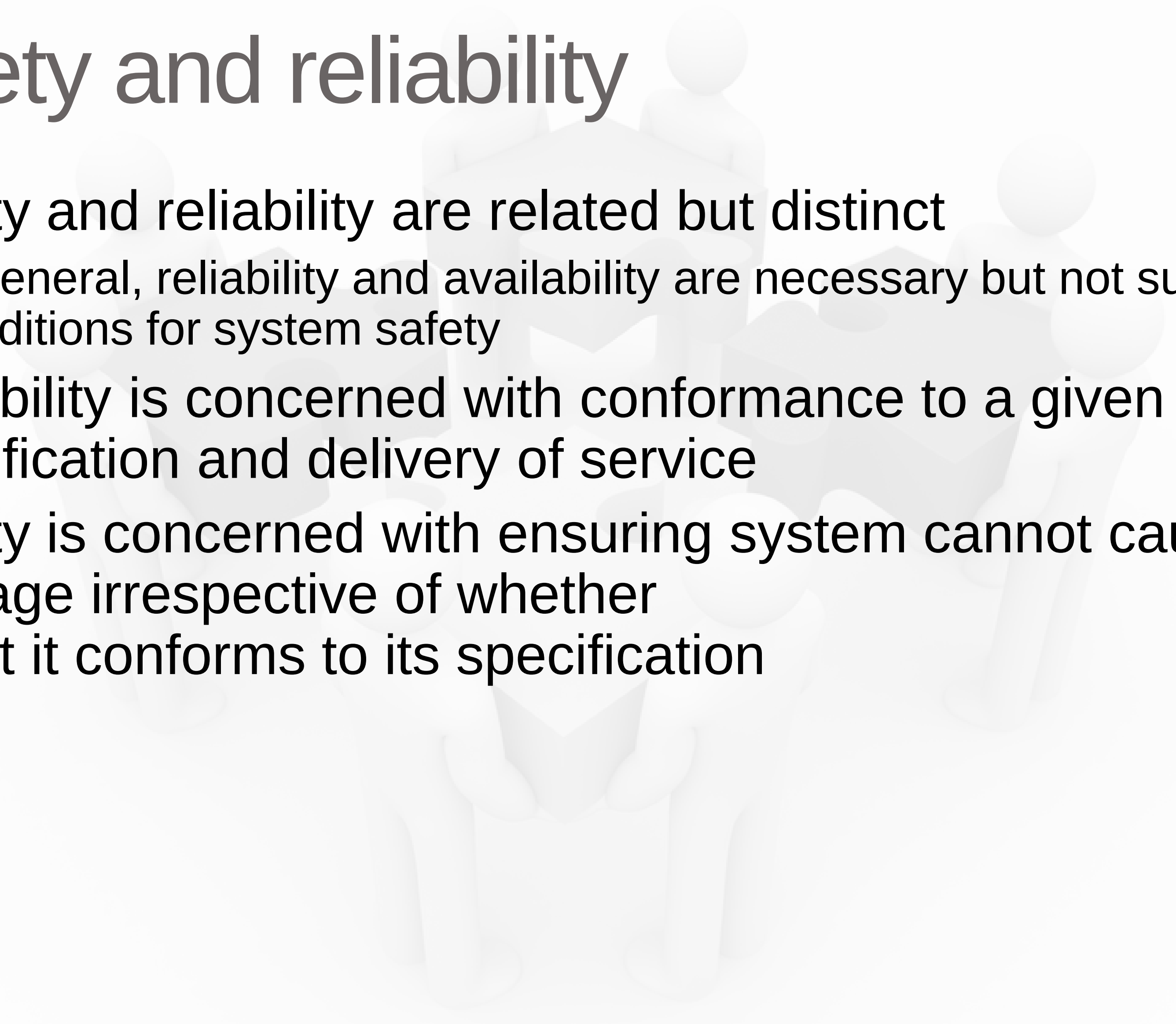
| Term | Description |
|---|---|
| Human error or mistake | Human behavior that results in the introduction of faults into a system. For example, in the wilderness weather system, a programmer might decide that the way to compute the time for the next transmission is to add 1 hour to the current time. This works except when the transmission time is between 23.00 and midnight (midnight is 00.00 in the 24-hour clock). |
| System fault | A characteristic of a software system that can lead to a system error. The fault is the inclusion of the code to add 1 hour to the time of the last transmission, without a check if the time is greater than or equal to 23.00. |
| System error | An erroneous system state that can lead to system behavior that is unexpected by system users. The value of transmission time is set incorrectly (to 24.XX rather than 00.XX) when the faulty code is executed. |
| System failure | An event that occurs at some point in time when the system does not deliver a service as expected by its users. No weather data is transmitted because the time is invalid. |

# Safety



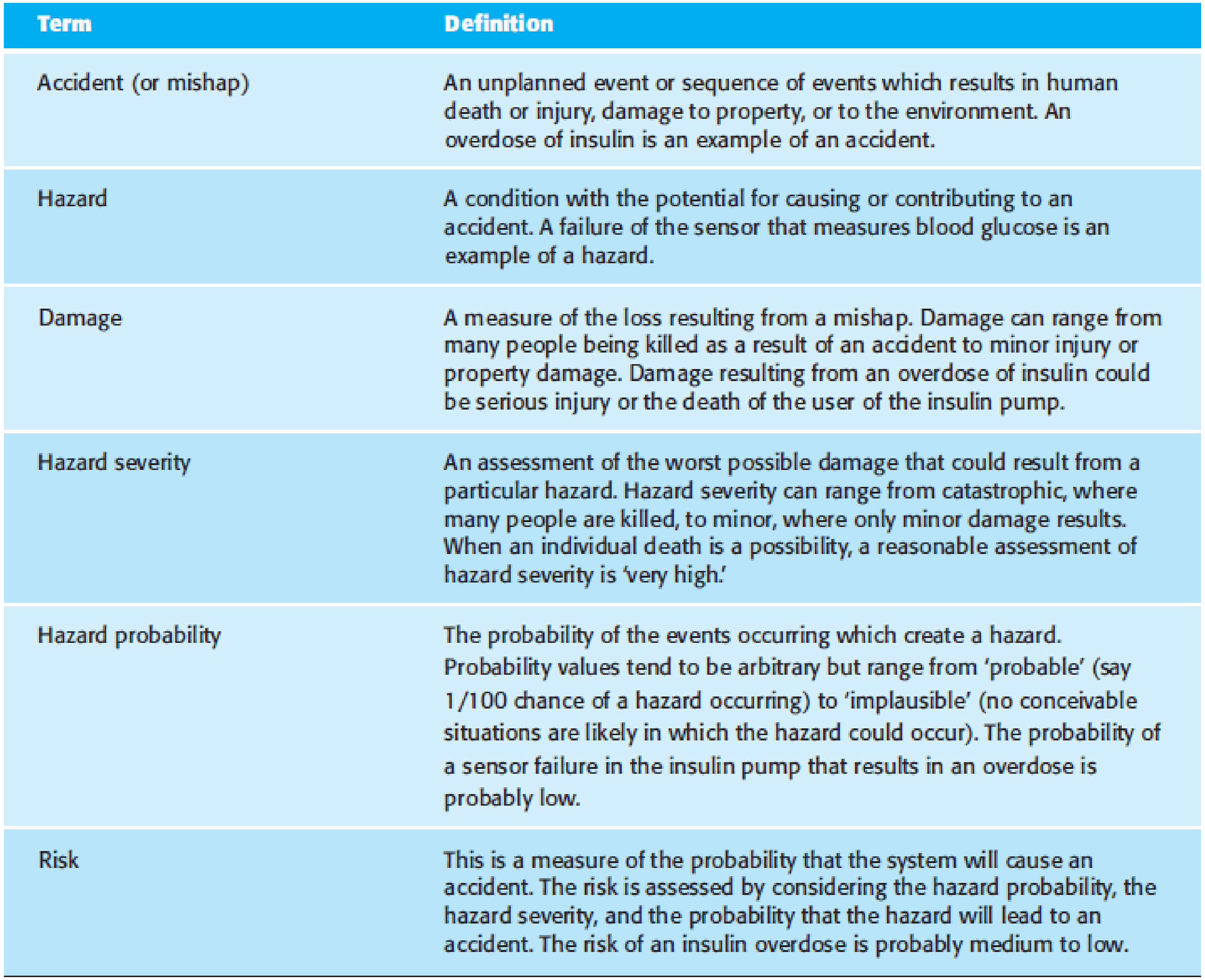
- Safety is a property of a system that reflects the system’s ability to operate, normally or abnormally, without danger of causing human injury or death and without damage to the system’s environment.
- It is important to consider software safety as most devices whose failure is critical now incorporate software-based control systems.
- Safety requirements are often exclusive requirements i.e. they exclude undesirable situations rather than specify required system services. These generate functional safety requirements.

# Safety and reliability



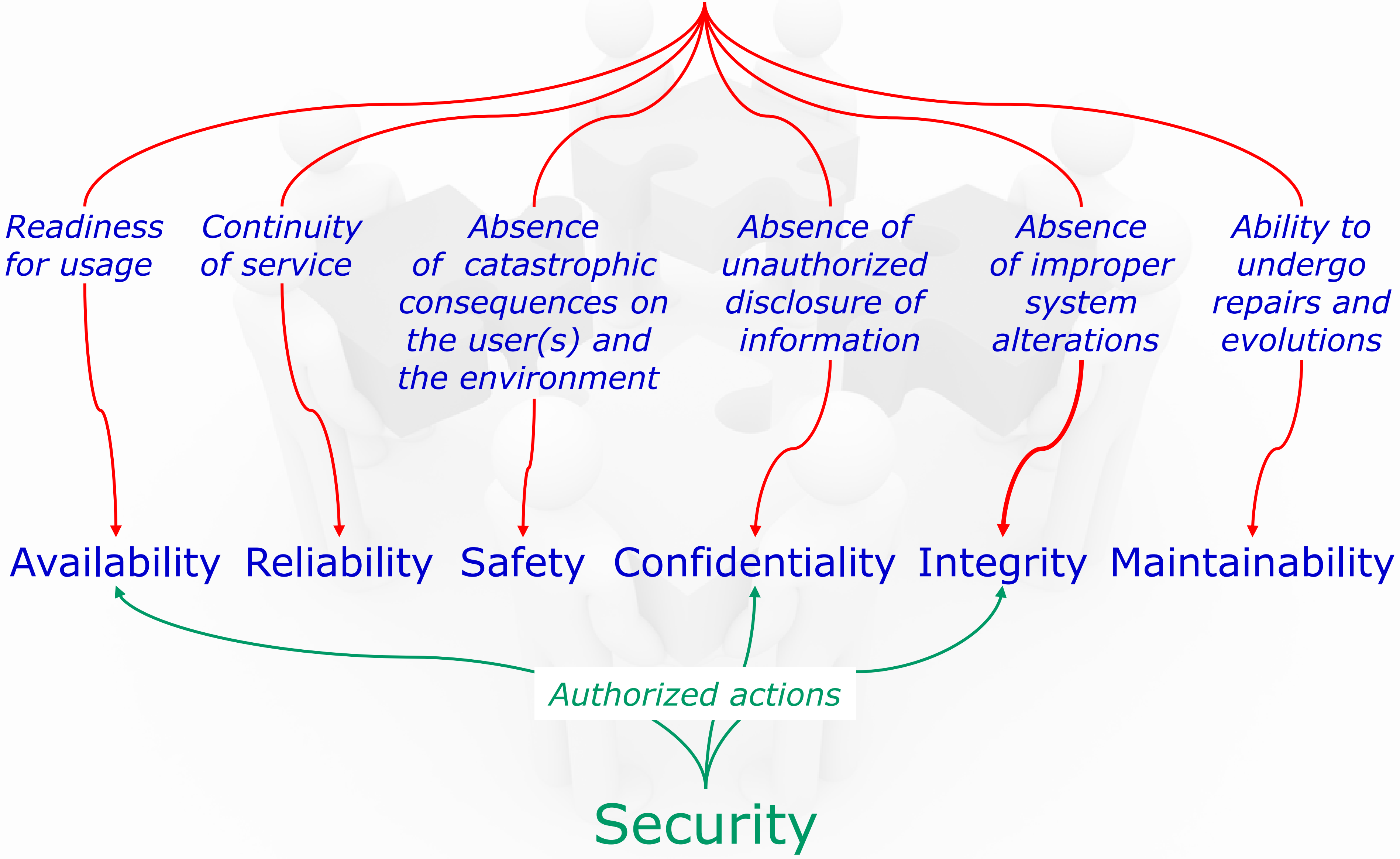
- Safety and reliability are related but distinct
  - In general, reliability and availability are necessary but not sufficient conditions for system safety
- Reliability is concerned with conformance to a given specification and delivery of service
- Safety is concerned with ensuring system cannot cause damage irrespective of whether
  or not it conforms to its specification

# Safety terminology



| Term | Definition |
|---|---|
| Accident (or mishap) | An unplanned event or sequence of events which results in human death or injury, damage to property, or to the environment. An overdose of insulin is an example of an accident. |
| Hazard | A condition with the potential for causing or contributing to an accident. A failure of the sensor that measures blood glucose is an example of a hazard. |
| Damage | A measure of the loss resulting from a mishap. Damage can range from many people being killed as a result of an accident to minor injury or property damage. Damage resulting from an overdose of insulin could be serious injury or the death of the user of the insulin pump. |
| Hazard severity | An assessment of the worst possible damage that could result from a particular hazard. Hazard severity can range from catastrophic, where many people are killed, to minor, where only minor damage results. When an individual death is a possibility, a reasonable assessment of hazard severity is ‘very high.’ |
| Hazard probability | The probability of the events occurring which create a hazard. Probability values tend to be arbitrary but range from ‘probable’ (say 1/100 chance of a hazard occurring) to ‘implausible’ (no conceivable situations are likely in which the hazard could occur). The probability of a sensor failure in the insulin pump that results in an overdose is probably low. |
| Risk | This is a measure of the probability that the system will cause an accident. The risk is assessed by considering the hazard probability, the hazard severity, and the probability that the hazard will lead to an accident. The risk of an insulin overdose is probably medium to low. |

# Dependability & Security



*Readiness for usage*

*Continuity of service*

*Absence of catastrophic consequences on the user(s) and the environment*

*Absence of unauthorized disclosure of information*

*Absence of improper system alterations*

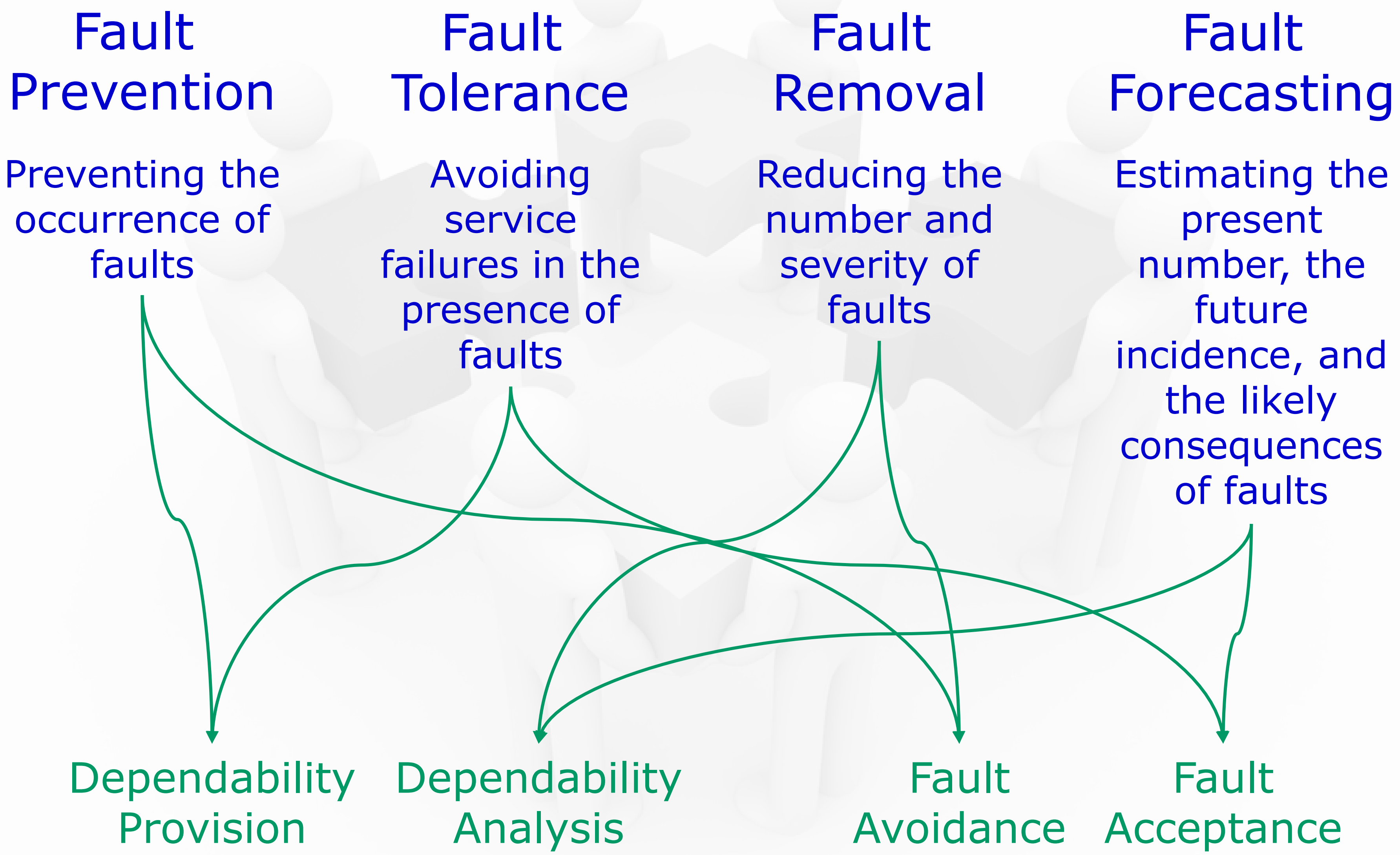
*Ability to undergo repairs and evolutions*

Availability Reliability Safety Confidentiality Integrity Maintainability

*Authorized actions*

Security

# Means to attain dependability



Fault Prevention

Preventing the occurrence of faults

Fault Tolerance

Avoiding service failures in the presence of faults

Fault Removal

Reducing the number and severity of faults

Fault Forecasting

Estimating the present number, the future incidence, and the likely consequences of faults

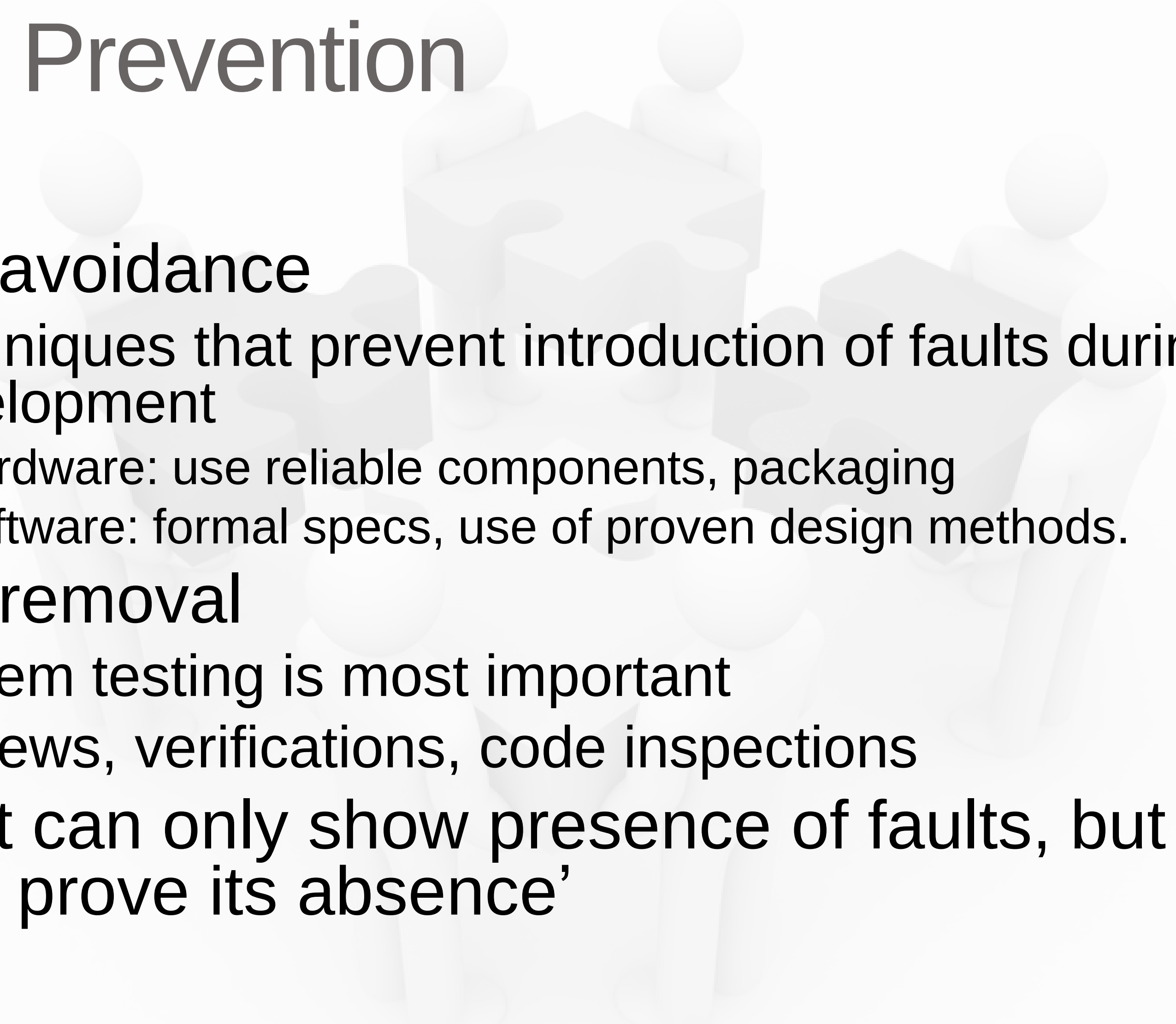
Dependability Provision

Dependability Analysis

Fault Avoidance

Fault Acceptance

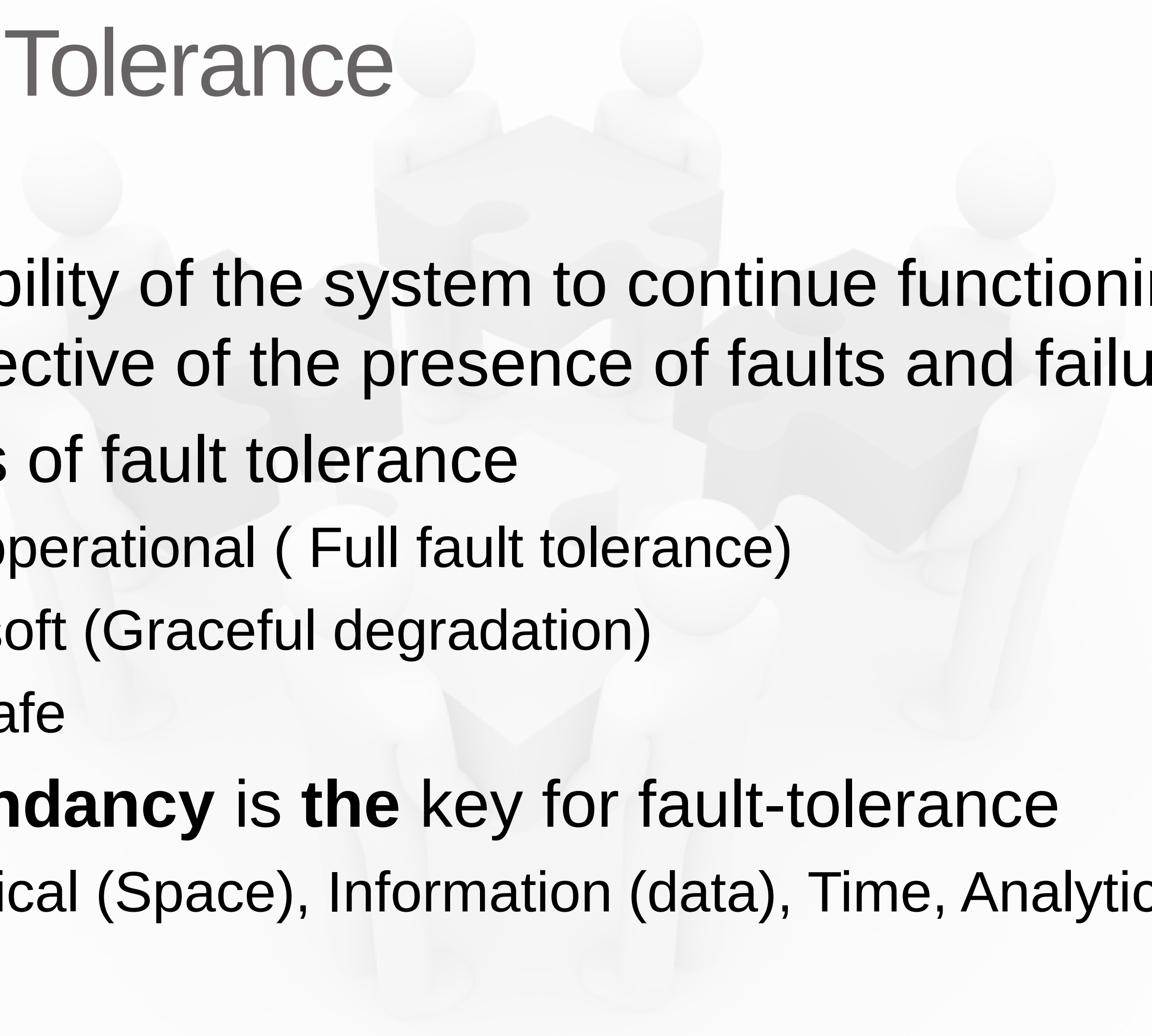
# Fault Prevention



- Fault avoidance
  - Techniques that prevent introduction of faults during development
    - Hardware: use reliable components, packaging
    - Software: formal specs, use of proven design methods.
- Fault removal
  - System testing is most important
  - Reviews, verifications, code inspections
- ’A test can only show presence of faults, but never prove its absence’

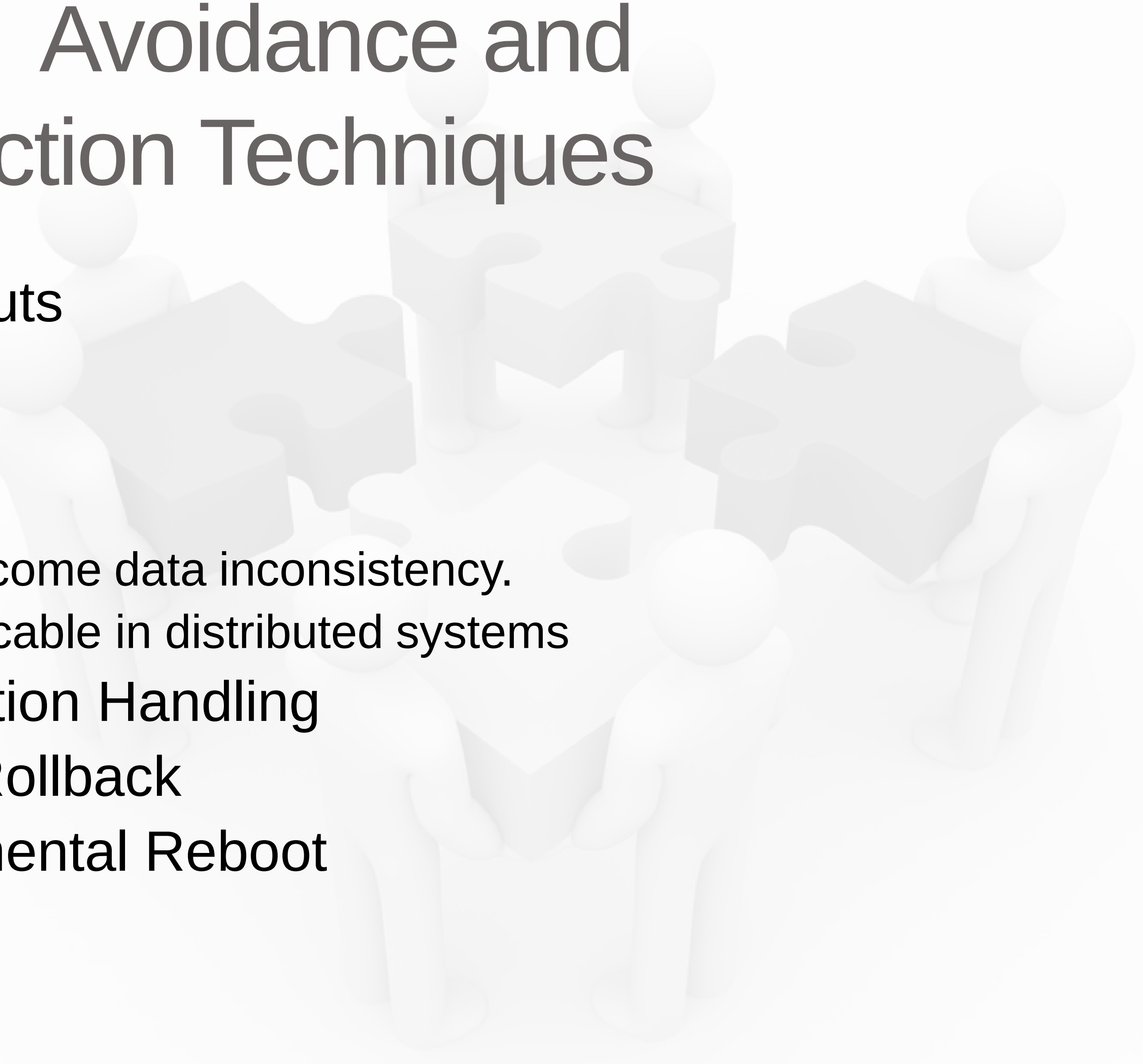
[Dijkstra]

# Fault Tolerance



- The ability of the system to continue functioning irrespective of the presence of faults and failures
- Levels of fault tolerance
  - Fail operational ( Full fault tolerance)
  - Fail soft (Graceful degradation)
  - Failsafe
- **Redundancy** is **the** key for fault-tolerance
  - Physical (Space), Information (data), Time, Analytical...

# Fault Avoidance and Detection Techniques



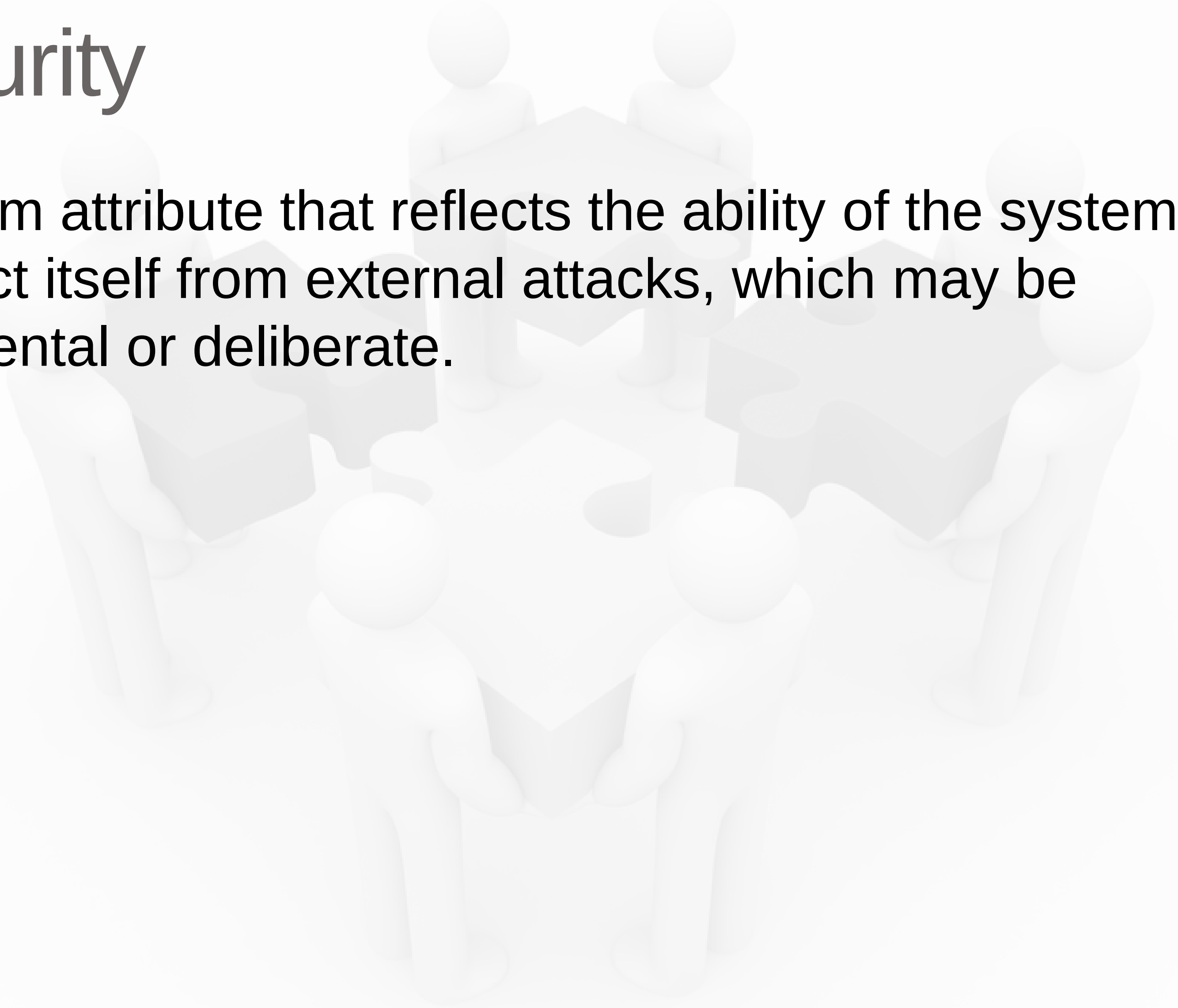
- Timeouts
  - Retry
  - Abort
- Audits
  - Overcome data inconsistency.
  - Applicable in distributed systems
- Exception Handling
- Task Rollback
- Incremental Reboot

# Fault forecasting

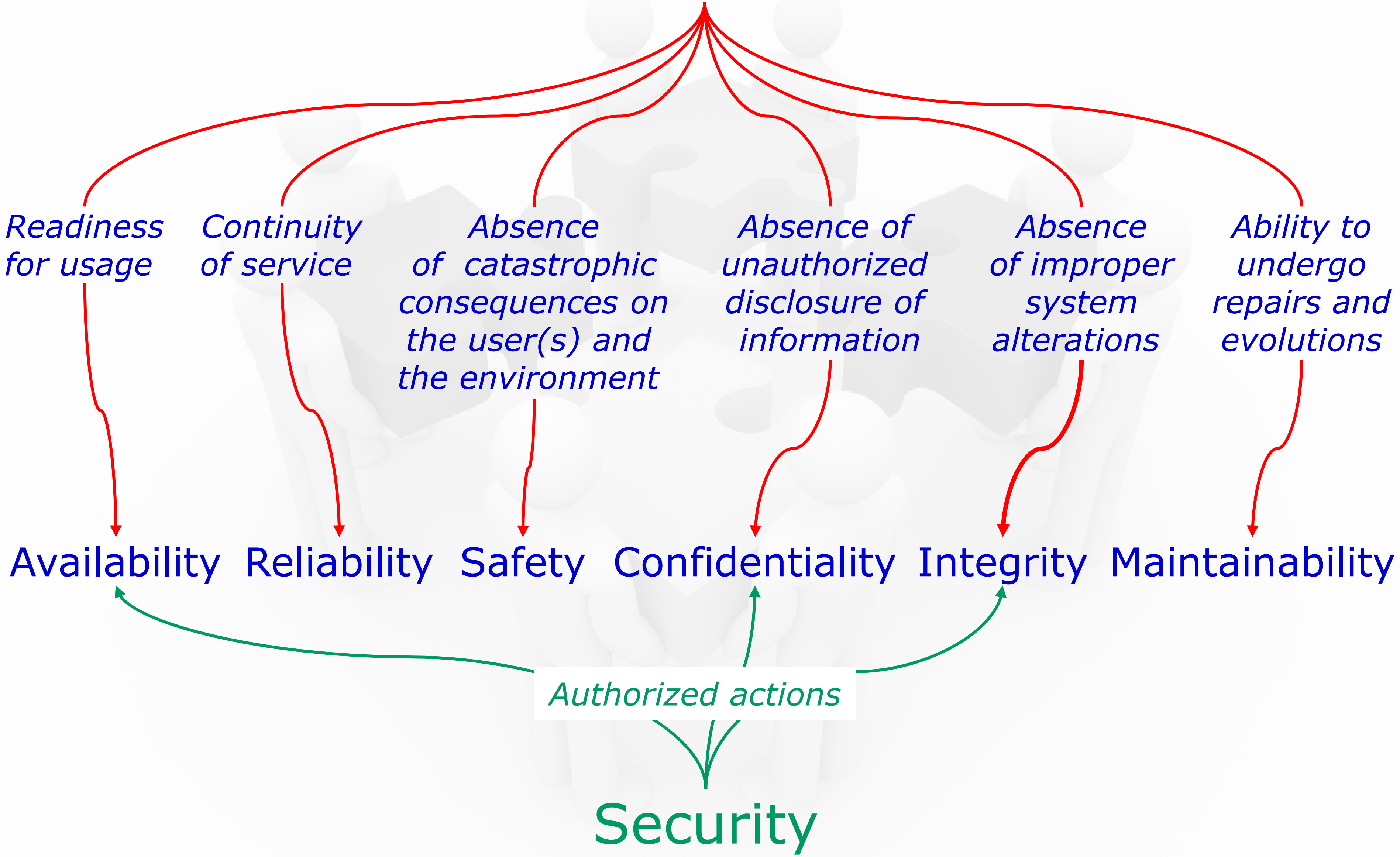
- Evaluation of system behavior
  - How to estimate the present number, the future incidents, the probability of different consequences
  - Qualitative (identify, classify, rank the failure modes, the event combinations, environmental conditions that would lead to system failures
  - Quantitative (probabilistic)
- Data analysis

# Security

- System attribute that reflects the ability of the system to protect itself from external attacks, which may be accidental or deliberate.



# Dependability & Security



*Readiness for usage*

*Continuity of service*

*Absence of catastrophic consequences on the user(s) and the environment*

*Absence of unauthorized disclosure of information*

*Absence of improper system alterations*

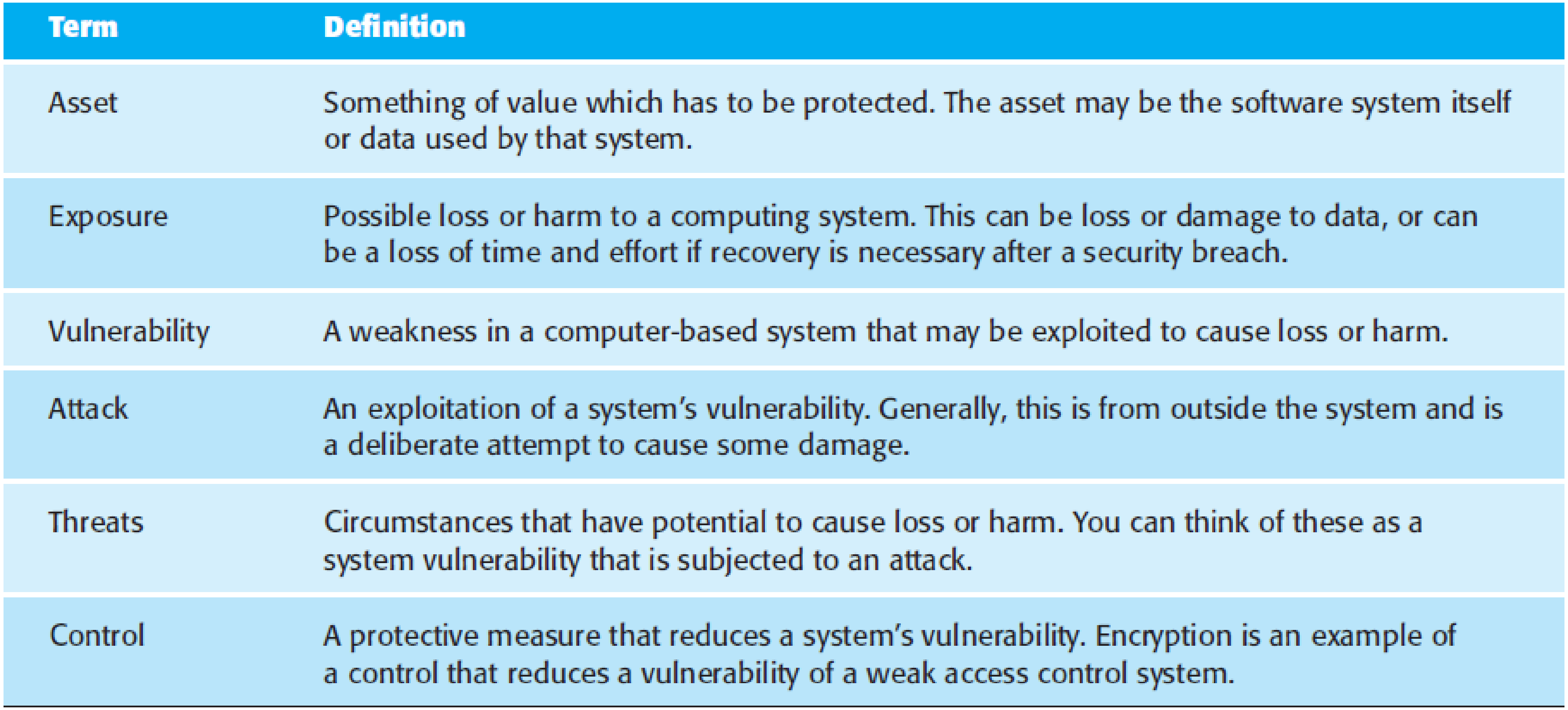
*Ability to undergo repairs and evolutions*

Availability Reliability Safety Confidentiality Integrity Maintainability

*Authorized actions*

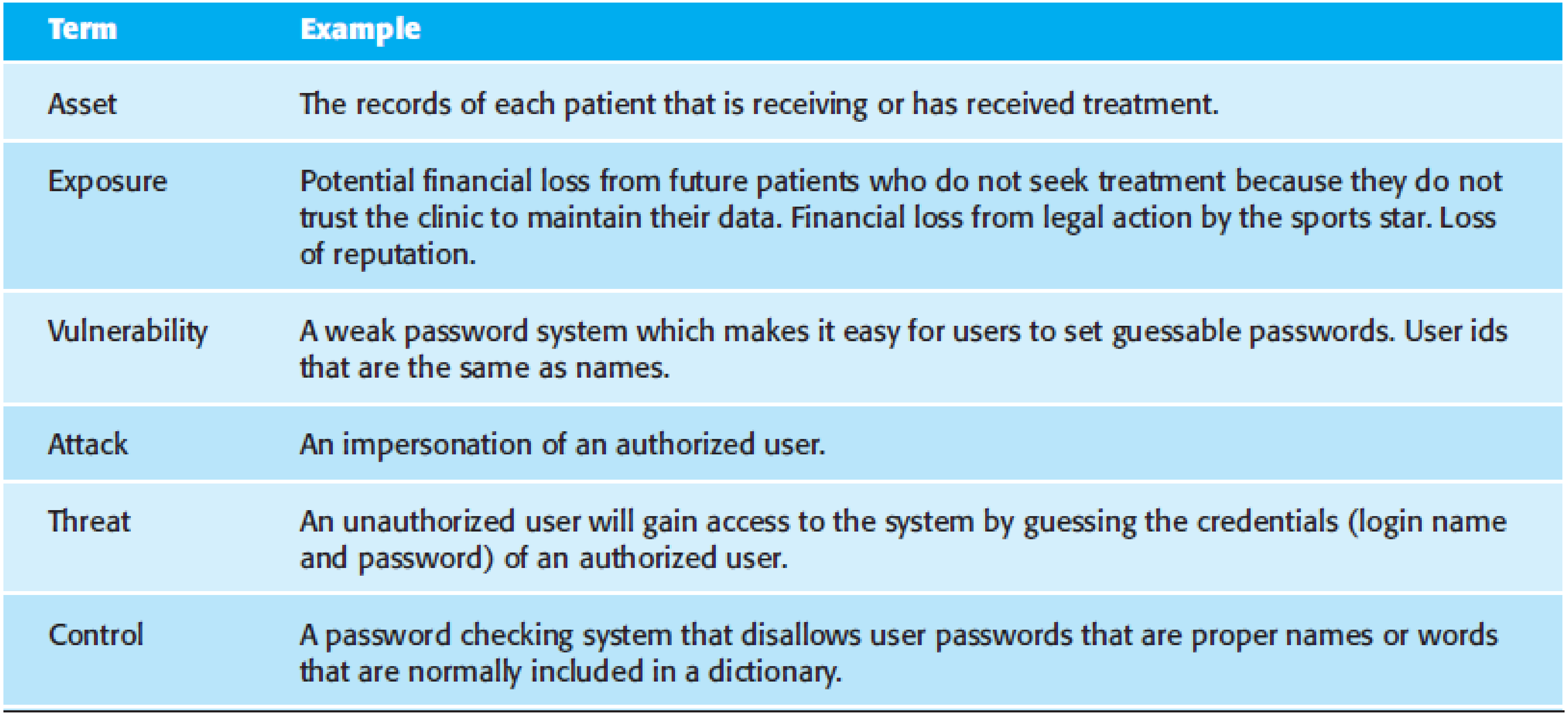
Security

# Security terminology



| Term | Definition |
|---|---|
| Asset | Something of value which has to be protected. The asset may be the software system itself or data used by that system. |
| Exposure | Possible loss or harm to a computing system. This can be loss or damage to data, or can be a loss of time and effort if recovery is necessary after a security breach. |
| Vulnerability | A weakness in a computer-based system that may be exploited to cause loss or harm. |
| Attack | An exploitation of a system's vulnerability. Generally, this is from outside the system and is a deliberate attempt to cause some damage. |
| Threats | Circumstances that have potential to cause loss or harm. You can think of these as a system vulnerability that is subjected to an attack. |
| Control | A protective measure that reduces a system's vulnerability. Encryption is an example of a control that reduces a vulnerability of a weak access control system. |

# Example

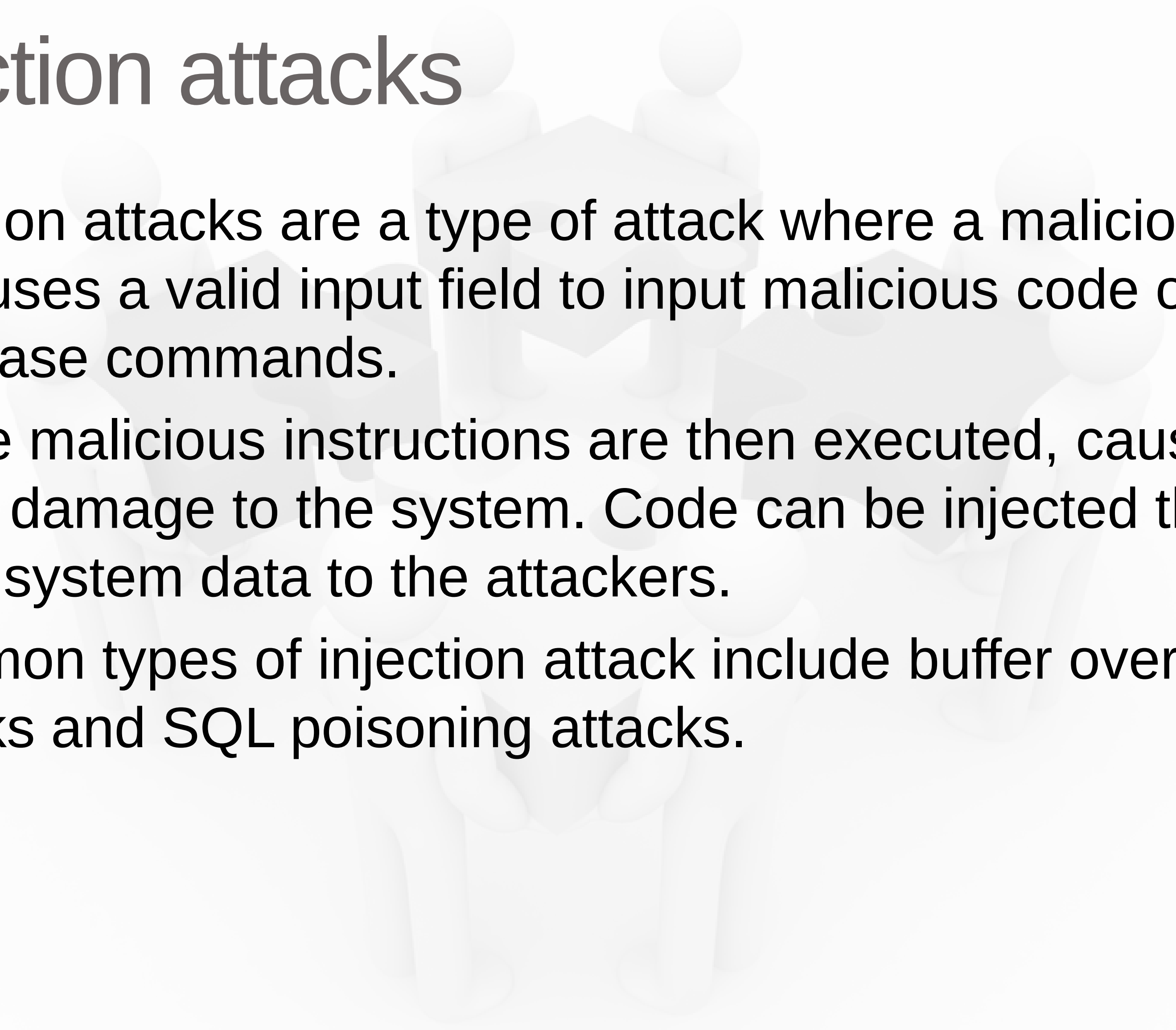


| Term | Example |
|---|---|
| Asset | The records of each patient that is receiving or has received treatment. |
| Exposure | Potential financial loss from future patients who do not seek treatment because they do not trust the clinic to maintain their data. Financial loss from legal action by the sports star. Loss of reputation. |
| Vulnerability | A weak password system which makes it easy for users to set guessable passwords. User ids that are the same as names. |
| Attack | An impersonation of an authorized user. |
| Threat | An unauthorized user will gain access to the system by guessing the credentials (login name and password) of an authorized user. |
| Control | A password checking system that disallows user passwords that are proper names or words that are normally included in a dictionary. |

# Types of security attacks

- Injection attacks
  - Buffer overflow
  - SQL injection (poisoning)
- Cross-site scripting attacks
- Session hijacking attacks
- Denial of service attacks
- Bruteforce attacks

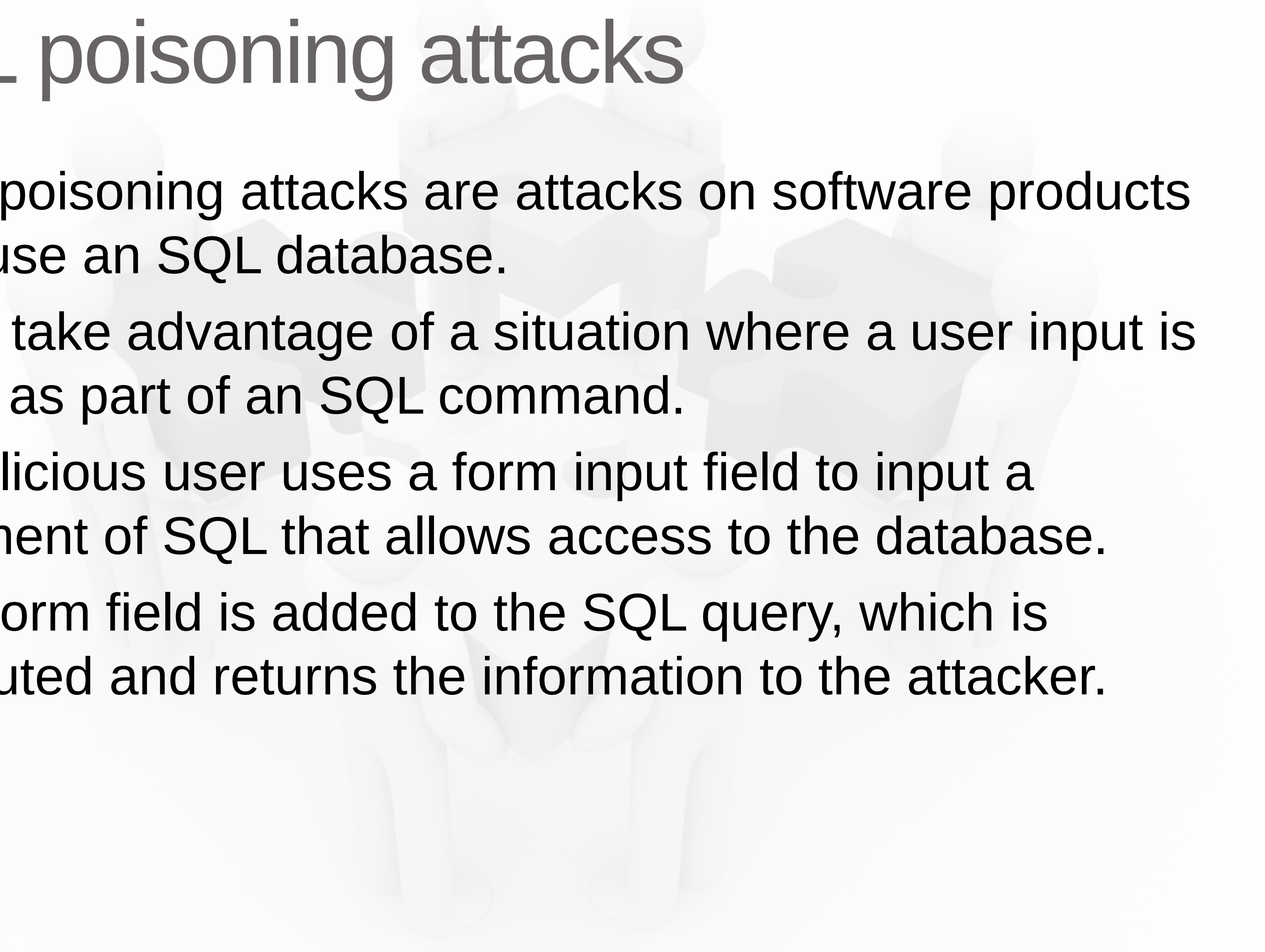
# Injection attacks



- Injection attacks are a type of attack where a malicious user uses a valid input field to input malicious code or database commands.
- These malicious instructions are then executed, causing some damage to the system. Code can be injected that leaks system data to the attackers.
- Common types of injection attack include buffer overflow attacks and SQL poisoning attacks.

# SQL poisoning attacks

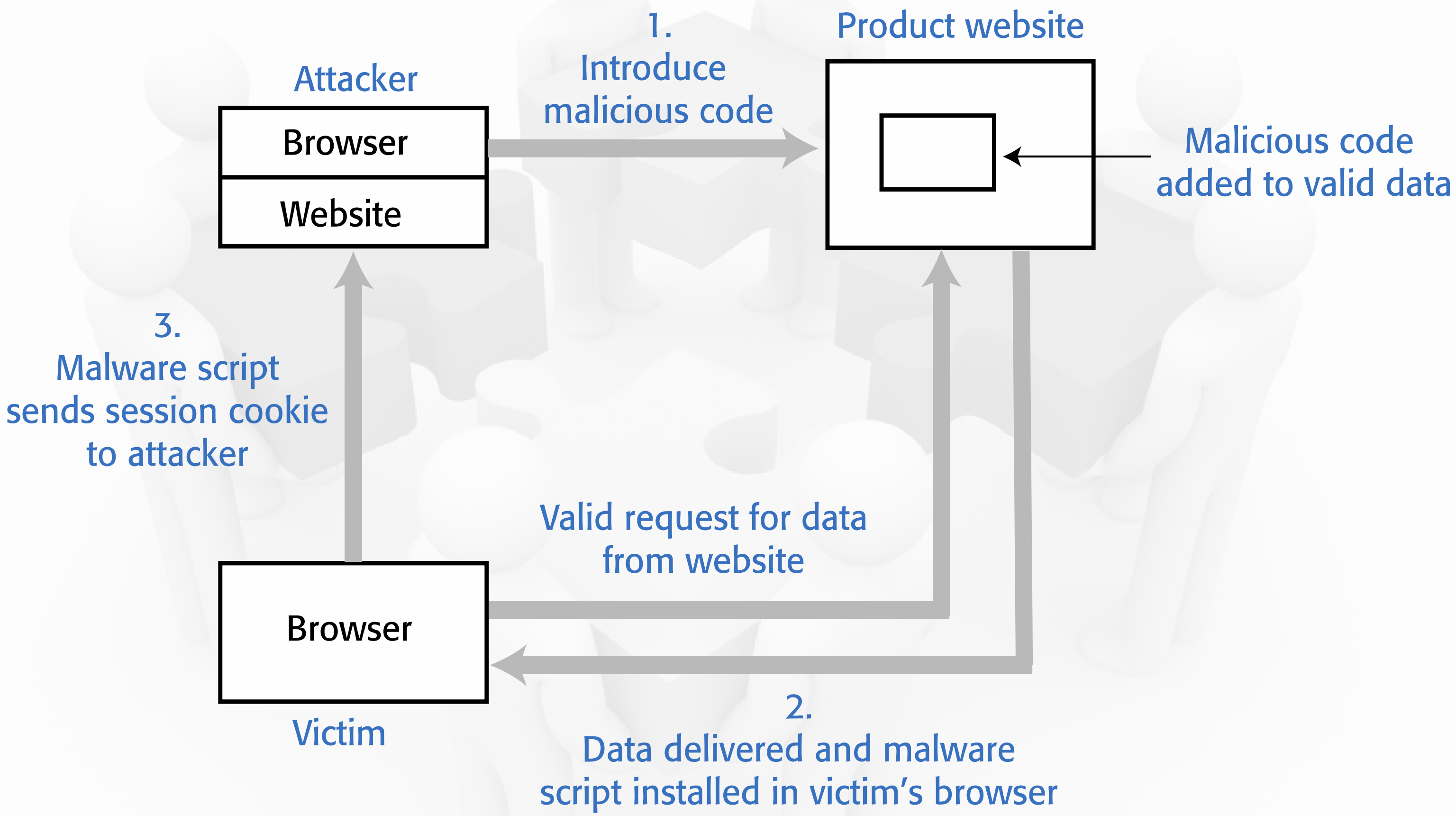
- SQL poisoning attacks are attacks on software products that use an SQL database.
- They take advantage of a situation where a user input is used as part of an SQL command.
- A malicious user uses a form input field to input a fragment of SQL that allows access to the database.
- The form field is added to the SQL query, which is executed and returns the information to the attacker.



# Cross-site scripting attacks

- Cross-site scripting attacks are another form of injection attack.
- An attacker adds malicious Javascript code to the web page that is returned from a server to a client and this script is executed when the page is displayed in the user’s browser.
- The malicious script may steal customer information or direct them to another website.
  - This may try to capture personal data or display advertisements.
  - Cookies may be stolen, which makes a session hijacking attack possible.
- As with other types of injection attack, cross-site scripting attacks may be avoided by input validation.

# Cross-site scripting attack



# Session hijacking attacks

- When a user authenticates themselves with a web application, a session is created.
  - A session is a time period during which the user’s authentication is valid. They don’t have to re-authenticate for each interaction with the system.
  - The authentication process involves placing a session cookie on the user’s device
- Session hijacking is a type of attack where an attacker gets hold of a session cookie and uses this to impersonate a legitimate user.
- There are several ways that an attacker can find out the session cookie value including cross-site scripting attacks and traffic monitoring.
  - In a cross-site scripting attack, the installed malware sends session cookies to the attackers.
  - Traffic monitoring involves attackers capturing the traffic between the client and server. The session cookie can then be identified by analysing the data exchanged.

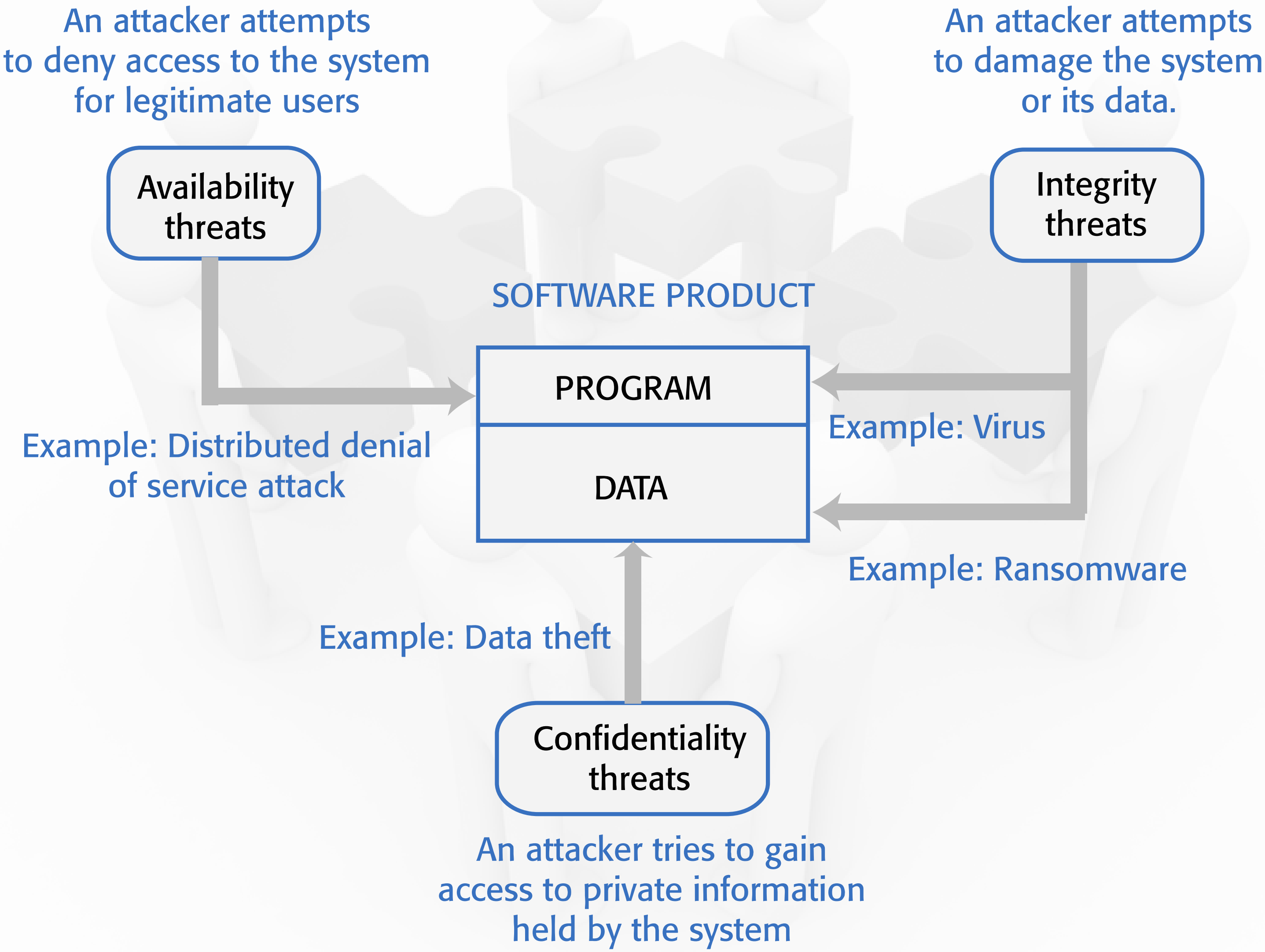
# Denial of service attacks

- Denial of service attacks are attacks on a software system that are intended to make that system unavailable for normal use.
- Distributed denial of service attacks (DDOS) are the most common type of denial of service attacks.
  - These involve distributed computers, that have usually been hijacked as part of a botnet, sending hundreds of thousands of requests for service to a web application. There are so many service requests that legitimate users are denied access.
- Other types of denial of service attacks target application users.
  - User lockout attacks take advantage of a common authentication policy that locks out a user after a number of failed authentication attempts. Their aim is to lock users out rather than gain access and so deny the service to these users.
  - Users often use their email address as their login name so if an attacker has access to a database of email addresses, he or she can try to login using these addresses.
- If you don’t lock accounts after failed validation, then attackers can use brute-force attacks on your system. If you do, you may deny access to legitimate users.

# Brute force attacks

- Brute force attacks are attacks on a web application where the attacker has some information, such as a valid login name, but does not have the password for the site.
- The attacker creates different passwords and tries to login with each of these. If the login fails, they then try again with a different password.
  - Attackers may use a string generator that generates every possible combination of letters and numbers and use these as passwords.
  - To speed up the process of password discovery, attackers take advantage of the fact that many users choose easy-to-remember passwords. They start by trying passwords from the published lists of the most common passwords.
- Brute force attacks rely on users setting weak passwords, so you can circumvent them by insisting that users set long passwords that are not in a dictionary or are common words.

## Types of security threats



An attacker attempts to deny access to the system for legitimate users

Availability threats

Example: Distributed denial of service attack

An attacker attempts to damage the system or its data.

Integrity threats

Example: Virus

Example: Ransomware

SOFTWARE PRODUCT

PROGRAM

DATA

Example: Data theft

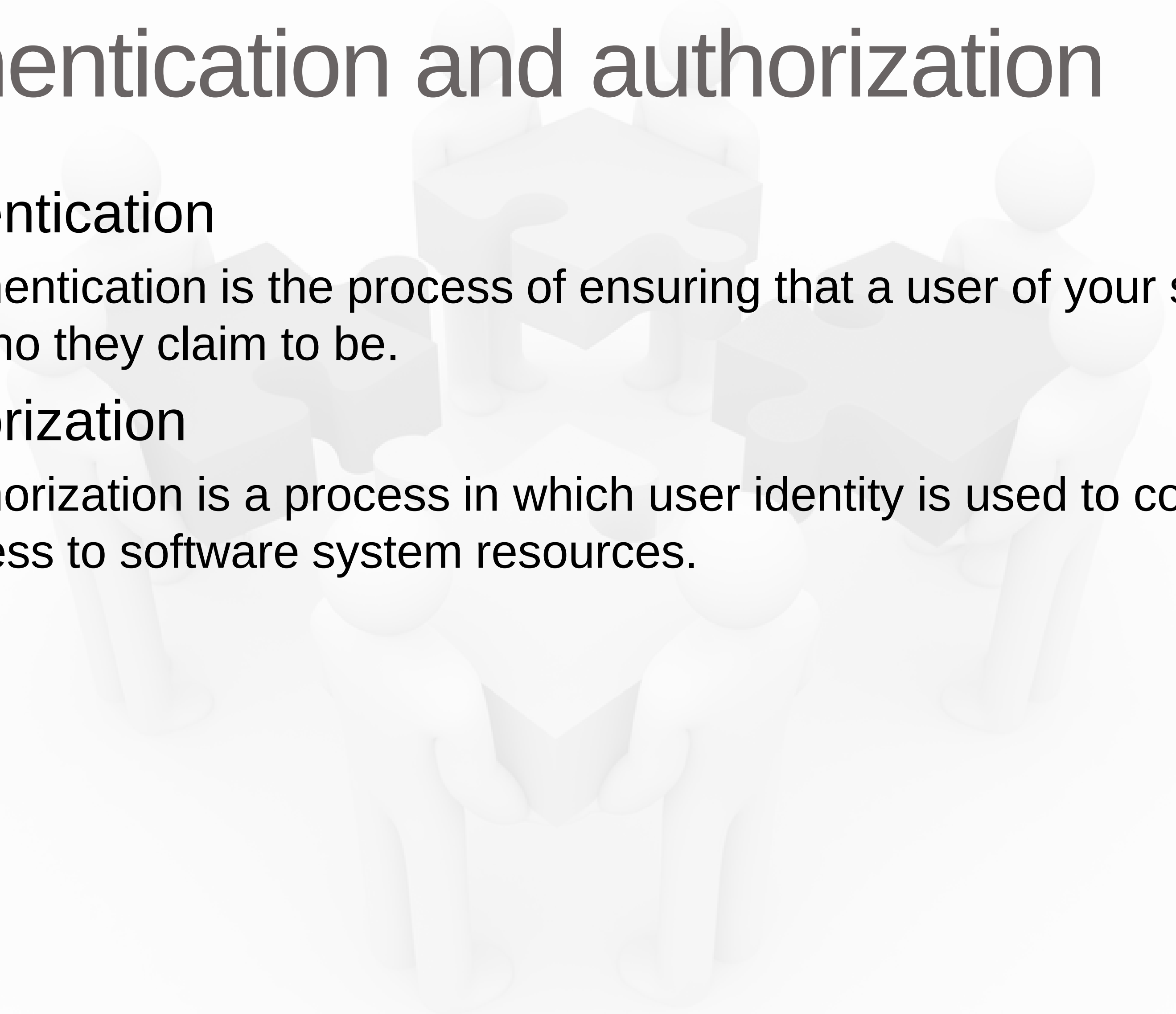
Confidentiality threats

An attacker tries to gain access to private information held by the system

# Actions to reduce the likelihood of hacking

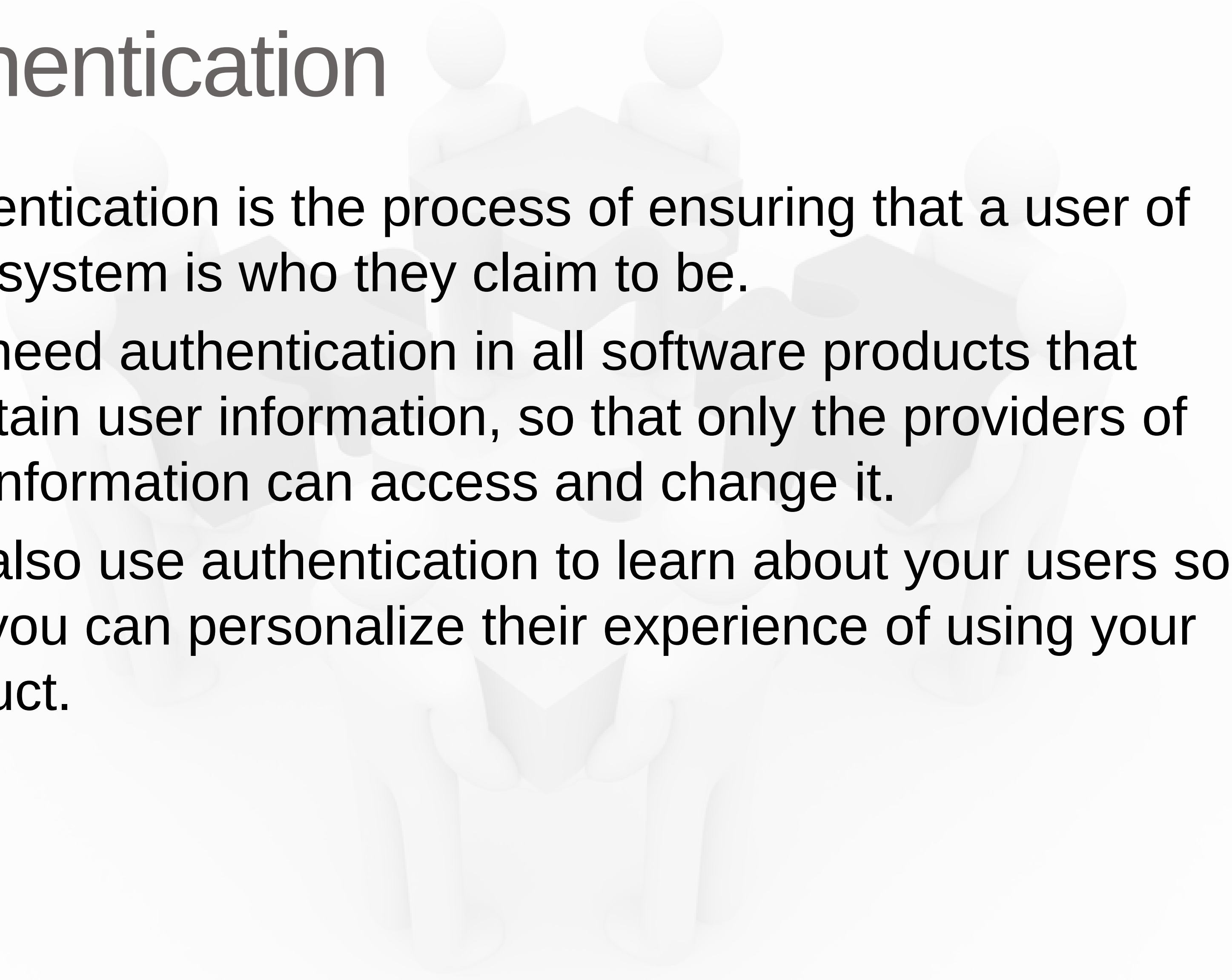
- ***Traffic encryption***
  Always encrypt the network traffic between clients and your server. This means setting up sessions using https rather than http. If traffic is encrypted it is harder to monitor to find session cookies.
- ***Multi-factor authentication***
  Always use multi-factor authentication and require confirmation of new actions that may be damaging. For example, before a new payee request is accepted, you could ask the user to confirm their identity by inputting a code sent to their phone.  You could also ask for password characters to be input before every potentially damaging action, such as transferring funds.
- ***Short timeouts***
  Use relatively short timeouts on sessions. If there has been no activity in a session for a few minutes, the session should be ended and future requests directed to an authentication page. This reduces the likelihood that an attacker can access an account if a legitimate user forgets to log off when they have finished their transactions.

# Authentication and authorization



- Authentication
  - Authentication is the process of ensuring that a user of your system is who they claim to be.
- Authorization
  - Authorization is a process in which user identity is used to control access to software system resources.

# Authentication

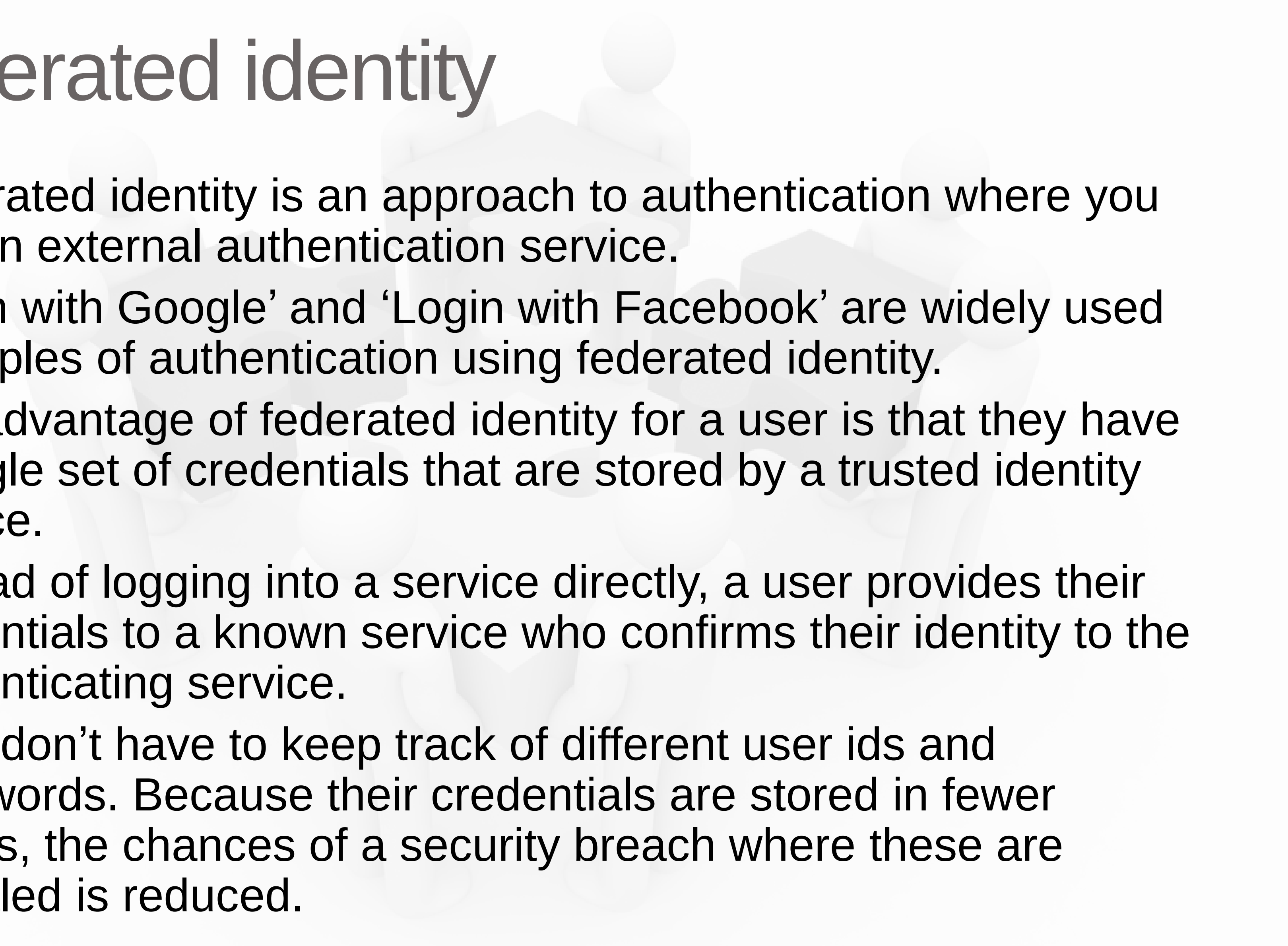


- Authentication is the process of ensuring that a user of your system is who they claim to be.
- You need authentication in all software products that maintain user information, so that only the providers of that information can access and change it.
- You also use authentication to learn about your users so that you can personalize their experience of using your product.

# Authentication methods

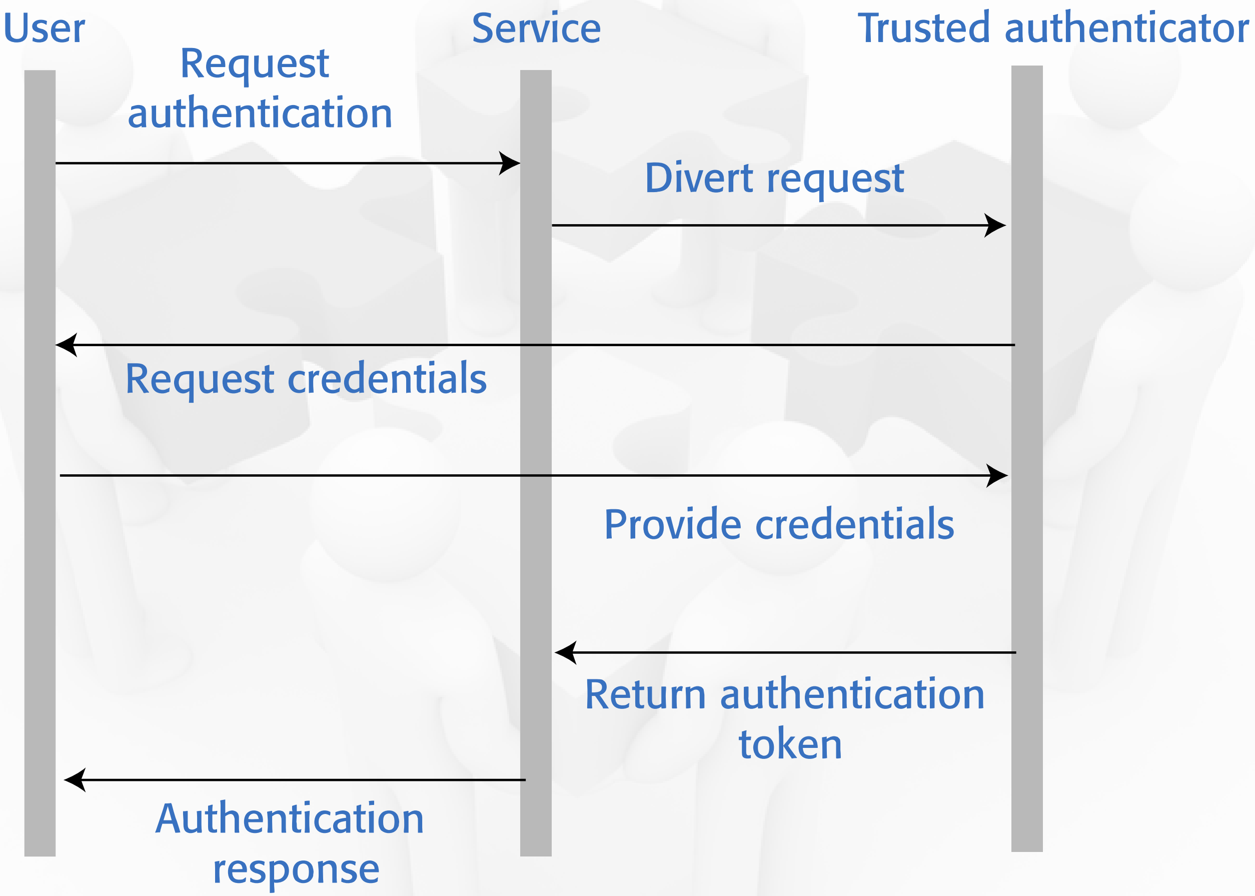
- Knowledge-based authentication
  - The user provides secret, personal information when they register with the system. Each time they log on, the system asks them for this information.
- Possession-based authentication
  - This relies on the user having a physical device (such as a mobile phone) that can generate or display information that is known to the authenticating system. The user inputs this information to confirm that they possess the authenticating device.
- Attribute-based authentication is based on a unique biometric attribute of the user, such as a fingerprint, which is registered with the system.
- Multi-factor authentication combines these approaches and requires users to use more than one authentication method.

# Federated identity



- Federated identity is an approach to authentication where you use an external authentication service.
- ‘Login with Google’ and ‘Login with Facebook’ are widely used examples of authentication using federated identity.
- The advantage of federated identity for a user is that they have a single set of credentials that are stored by a trusted identity service.
- Instead of logging into a service directly, a user provides their credentials to a known service who confirms their identity to the authenticating service.
- They don’t have to keep track of different user ids and passwords. Because their credentials are stored in fewer places, the chances of a security breach where these are revealed is reduced.

# Federated identity



# Authorization

- Authentication involves a user proving their identity to a software system.
- Authorization is a complementary process in which that identity is used to control access to software system resources.
  - For example, if you use a shared folder on Dropbox, the folder’s owner may authorize you to read the contents of that folder, but not to add new files or overwrite files in the folder.
- When a business wants to define the type of access that users get to resources, this is based on an access control policy.
- This policy is a set of rules that define what information (data and programs) is controlled, who has access to that information and the type of access that is allowed

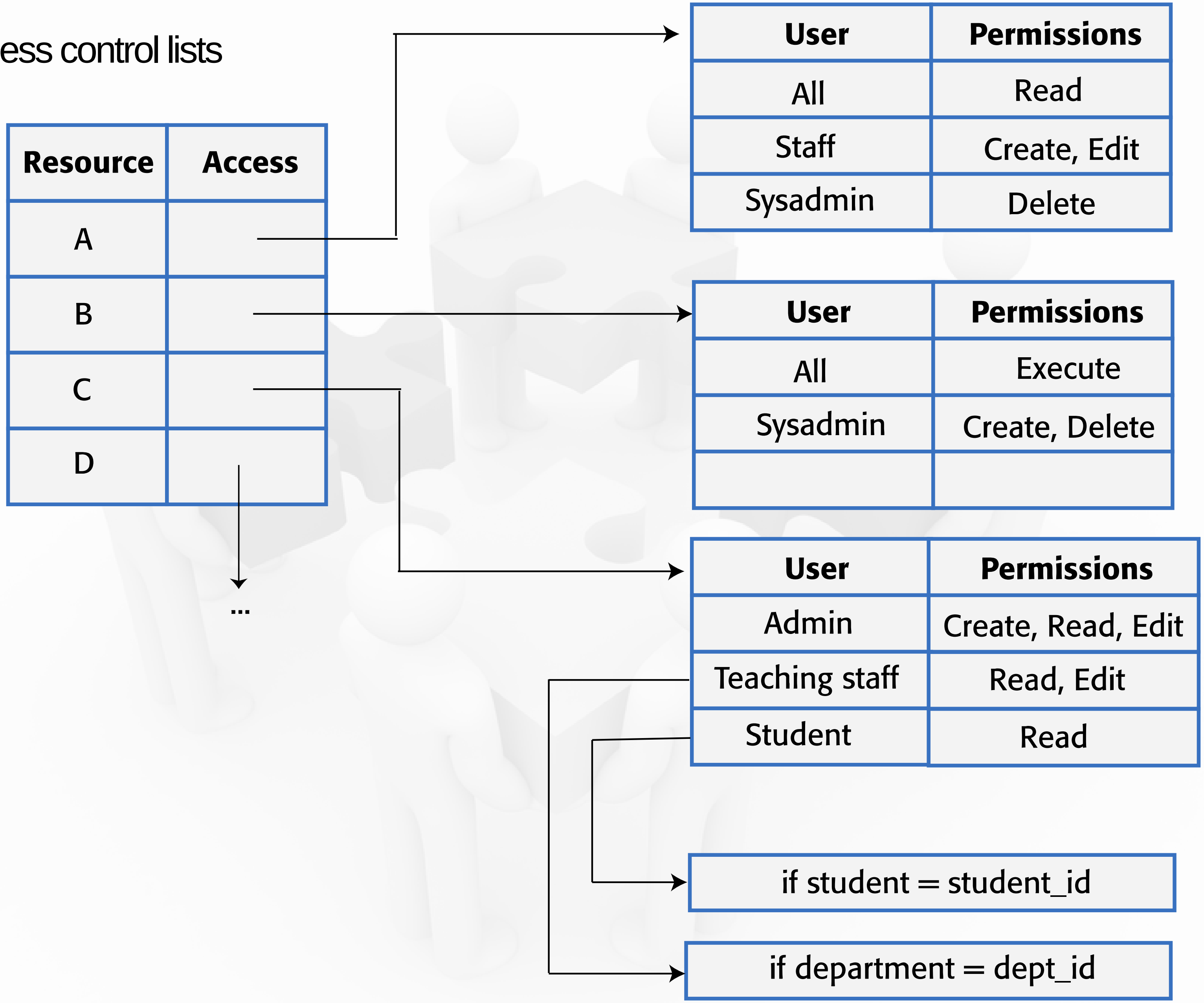
# Access control policies

- Explicit access control policies are important for both legal and technical reasons.
    - Data protection rules limit the access the personal data and this must be reflected in the defined access control policy. If this policy is incomplete or does not conform to the data protection rules, then there may be subsequent legal action in the event of a data breach.
    - Technically, an access control policy can be a starting point for setting up the access control scheme for a system.
    - For example, if the access control policy defines the access rights of students, then when new students are registered, they all get these rights by default.

# Access control lists

- Access control lists (ACLs) are used in most file and database systems to implement access control policies.
- Access control lists are tables that link users with resources and specify what those users are permitted to do.
  - For example, for this book I would like to be able to set up an access control list to a book file that allows reviewers to read that file and annotate it with comments. However, they are not allowed to edit the text or to delete the file.
- If access control lists are based on individual permissions, then these can become very large. However, you can dramatically cut their size by allocating users to groups and then assigning permissions to the group

# Access control lists



| Resource | Access |
|---|---|
| A | |
| B | |
| C | |
| D | |

...

| User | Permissions |
|---|---|
| All | Read |
| Staff | Create, Edit |
| Sysadmin | Delete |

| User | Permissions |
|---|---|
| All | Execute |
| Sysadmin | Create, Delete |
| | |

| User | Permissions |
|---|---|
| Admin | Create, Read, Edit |
| Teaching staff | Read, Edit |
| Student | Read |

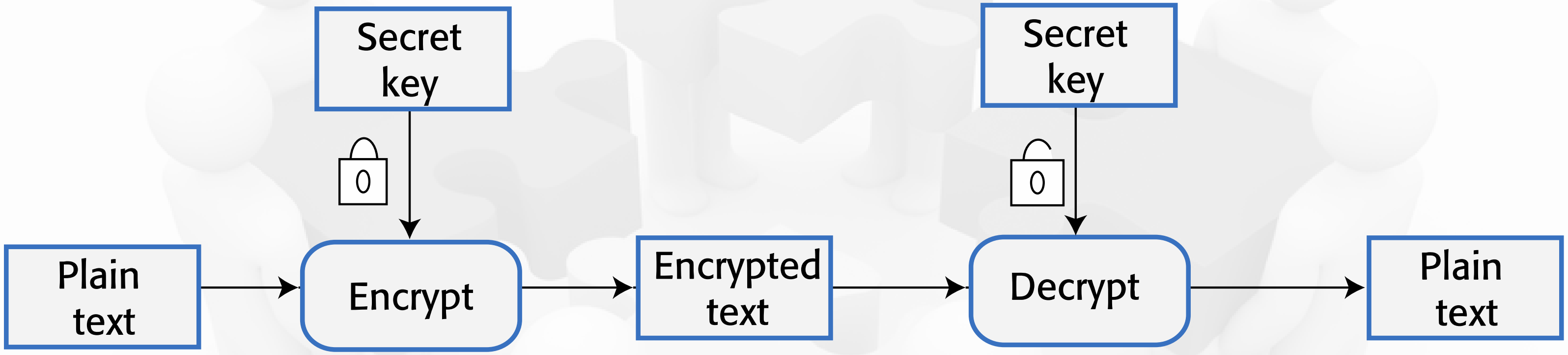
if student = student_id

if department = dept_id

# Encryption

- Encryption is the process of making a document unreadable by applying an algorithmic transformation to it.
- A secret key is used by the encryption algorithm as the basis of this transformation. You can decode the encrypted text by applying the reverse transformation.
- Modern encryption techniques are such that you can encrypt data so that it is practically uncrackable using currently available technology.
- However, history has demonstrated that apparently strong encryption may be crackable when new technology becomes available.
- If commercial quantum systems become available, we will have to use a completely different approach to encryption on the Internet.

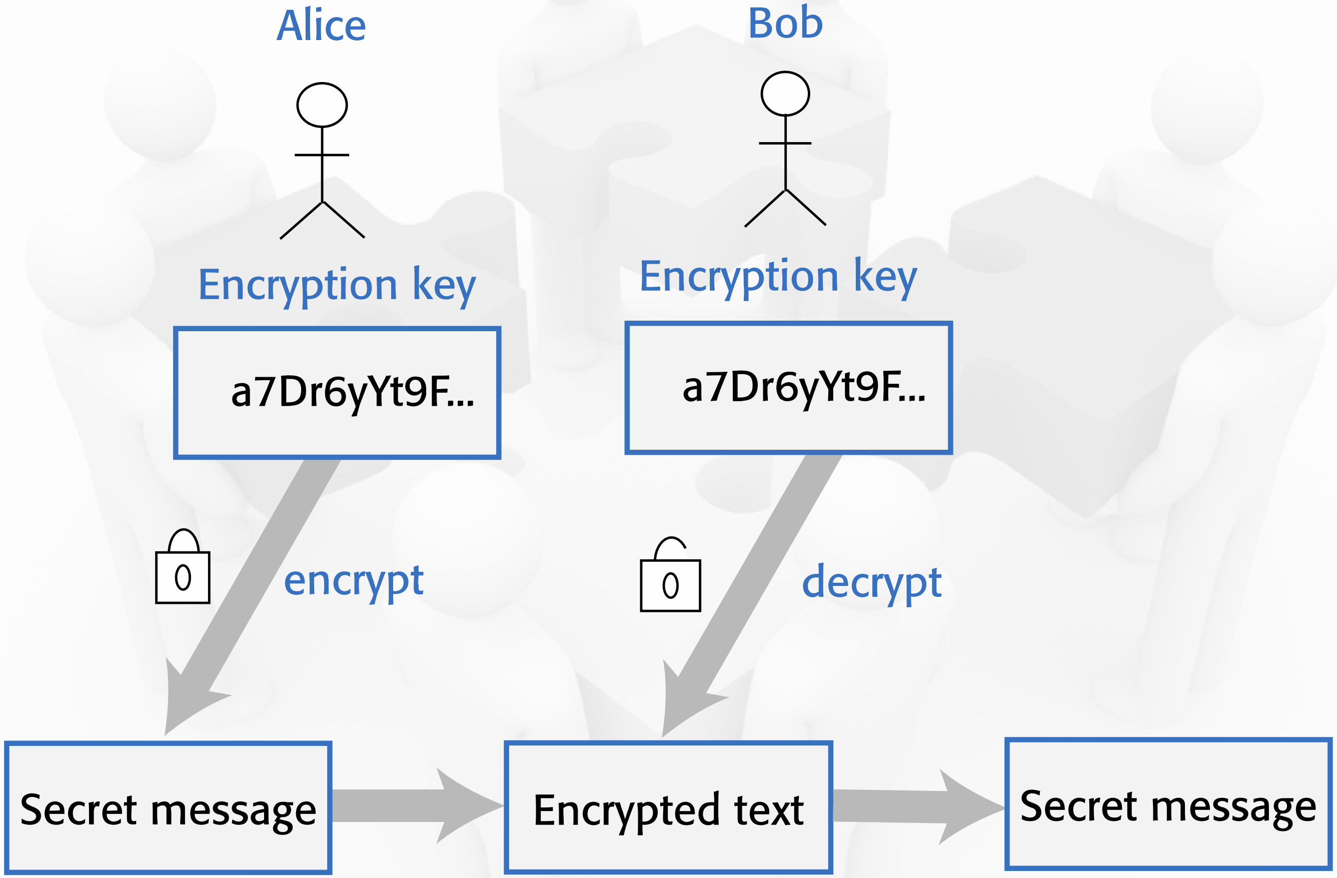
Figure 7.9 Encryption and decryption



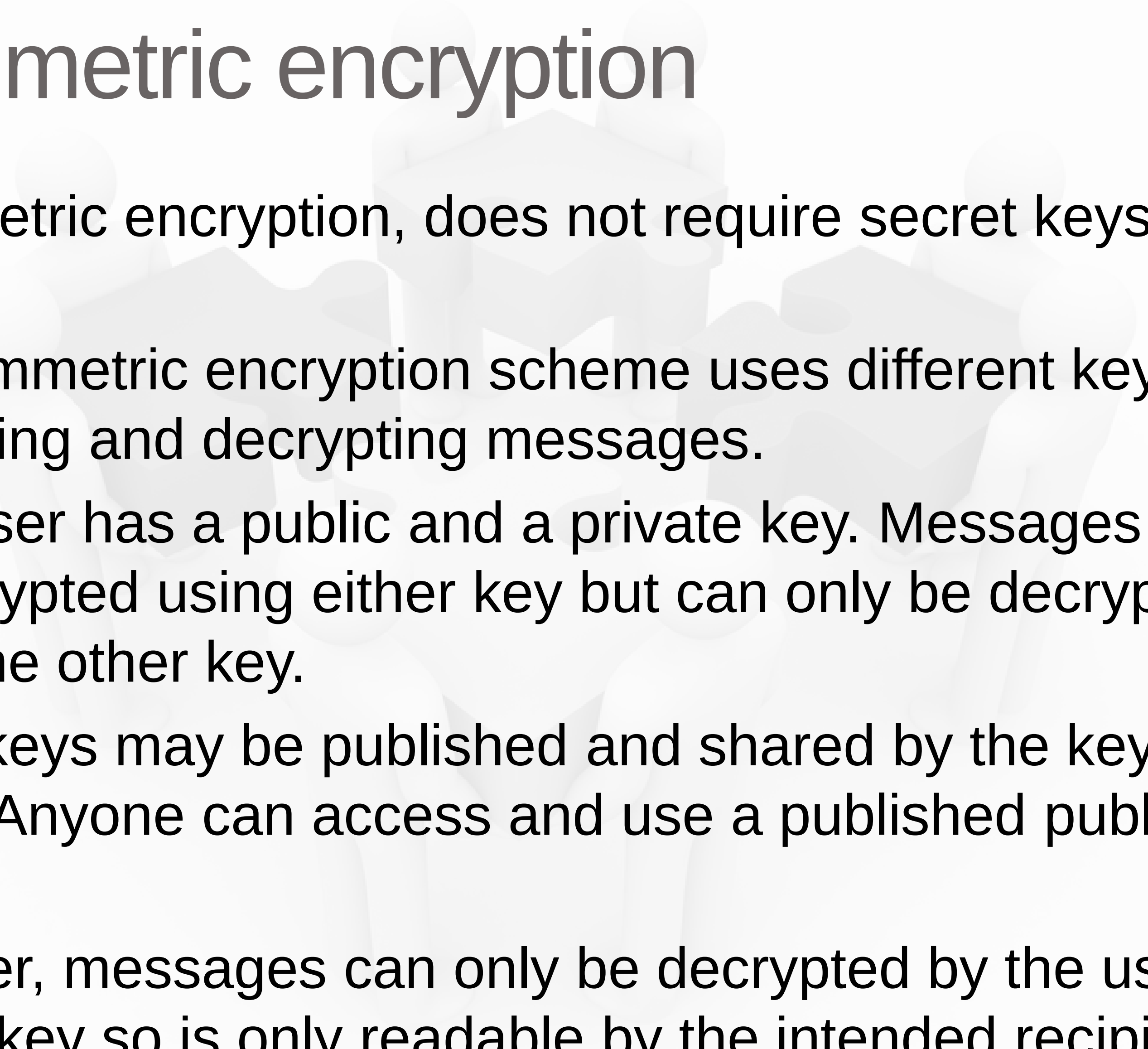
# Symmetric encryption

- In a symmetric encryption scheme, the same encryption key is used for encoding and decoding the information that is to be kept secret.
- If Alice and Bob wish to exchange a secret message, both must have a copy of the encryption key. Alice encrypts the message with this key. When Bob receives the message, he decodes it using the same key to read its contents.
- The fundamental problem with a symmetric encryption scheme is securely sharing the encryption key.
- If Alice simply sends the key to Bob, an attacker may intercept the message and gain access to the key. The attacker can then decode all future secret communications.

Figure 7.10 Symmetric encryption

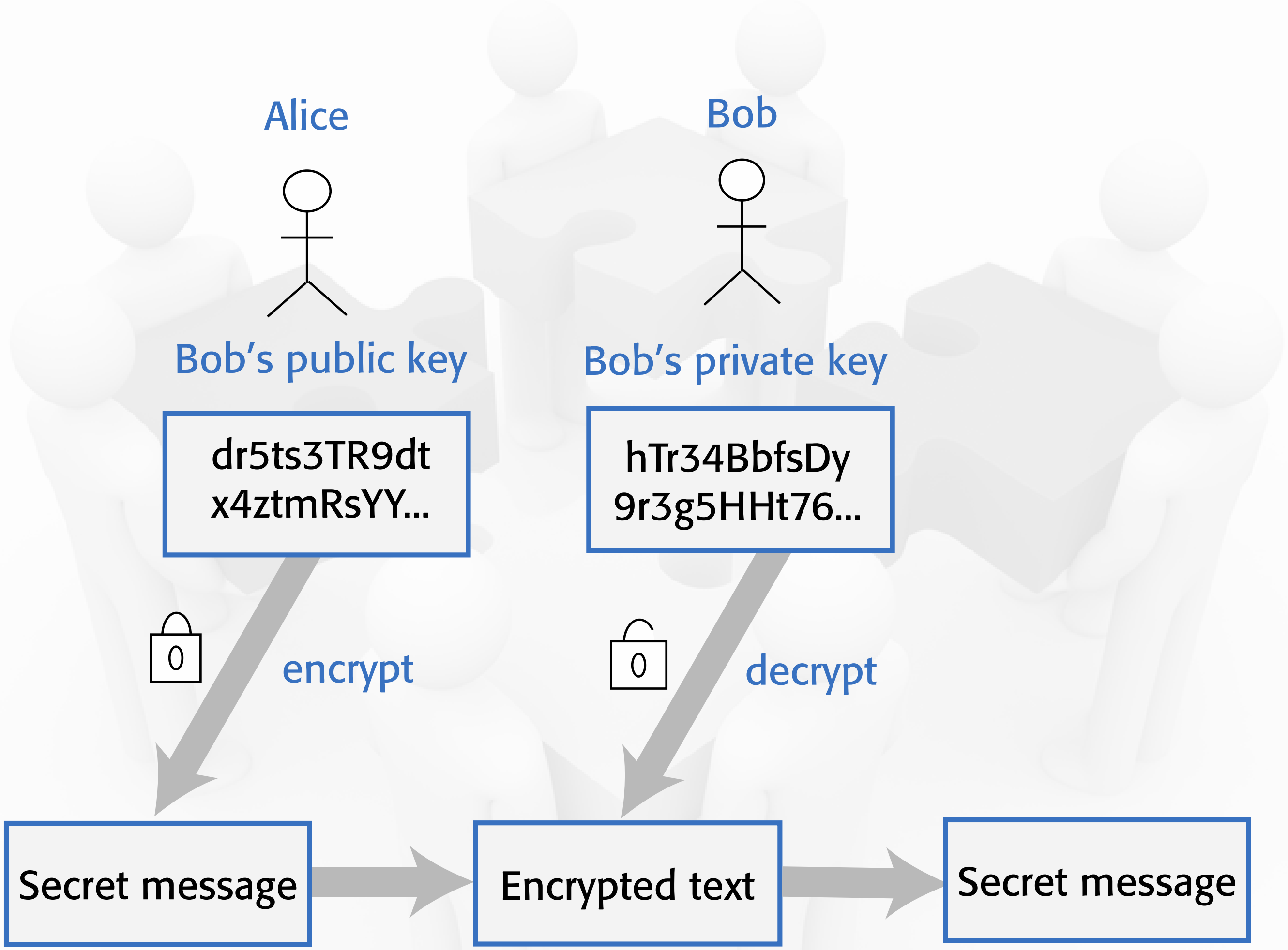


# Asymmetric encryption



- Asymmetric encryption, does not require secret keys to be shared.
- An asymmetric encryption scheme uses different keys for encrypting and decrypting messages.
- Each user has a public and a private key. Messages may be encrypted using either key but can only be decrypted using the other key.
- Public keys may be published and shared by the key owner. Anyone can access and use a published public key.
- However, messages can only be decrypted by the user’s private key so is only readable by the intended recipient

Figure 7.11 Asymmetric encryption


Alice
Bob
Bob’s public key
Bob’s private key
dr5ts3TR9dt
x4ztmRsYY...
hTr34BbfsDy
9r3g5HHt76...
encrypt
decrypt
Secret message
Encrypted text
Secret message