Models of Distributed Computing (7)

4. RPC - Client/Server Model

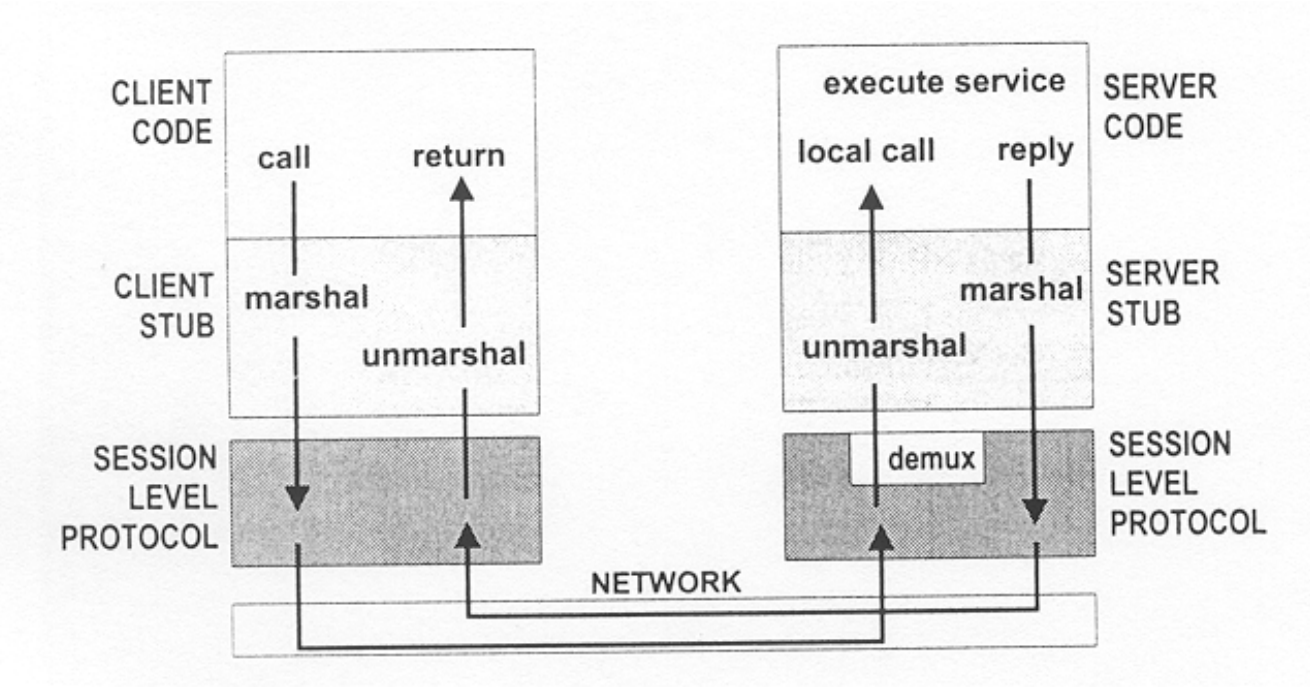
This is the model still mostly used in distributed programming and distributed computing (WWW).

Idea: structuring the application around *servers* which provide services, and *clients* which *request* them;
remote requests should look like to the client like a local service request

RPC Architecture

A software architecture is required that provides tools and mechanisms to support distributed programming within this model.

RPC middleware



Models of Distributed Computing (14)

5. Distributed Shared Memory (DSM) Model

Idea: Emulating the environment of a shared memory multiprocessor in a distributed system

---> Providing the illusion of a centralized memory system (e.g. including library functions)

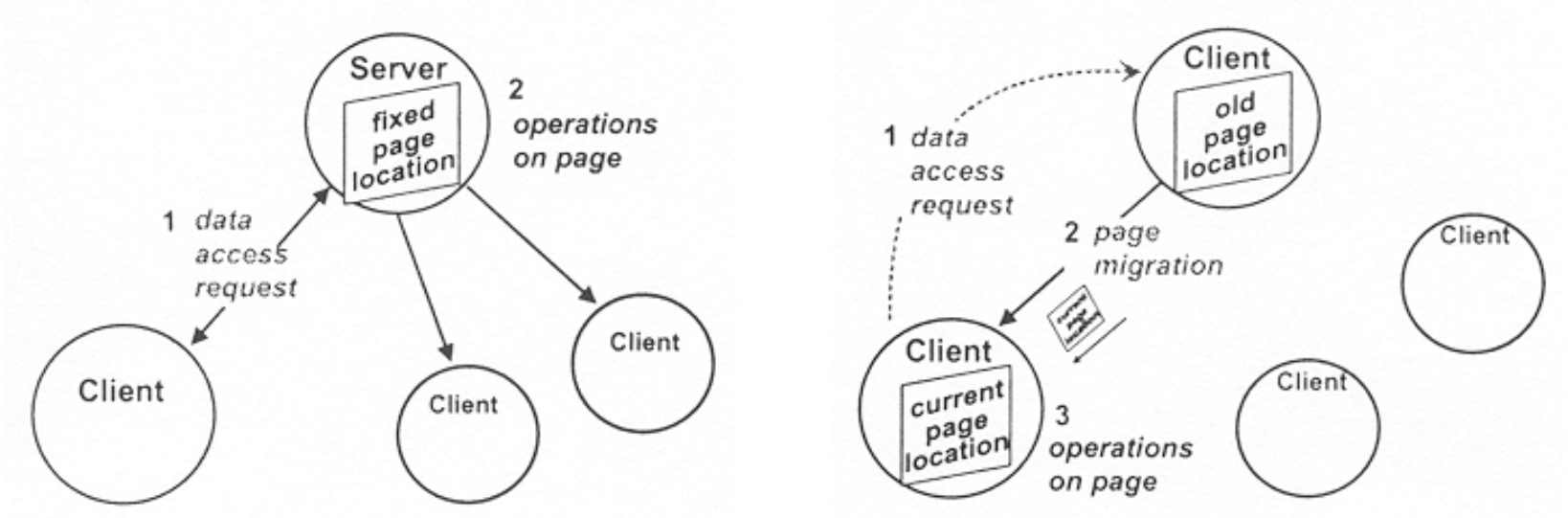
---> memory distribution is made transparent to the application programmer

Main application areas: where clusters of workstations are used to accomplish high-performance computing

Main problem: Maintaining consistency by, if possible,

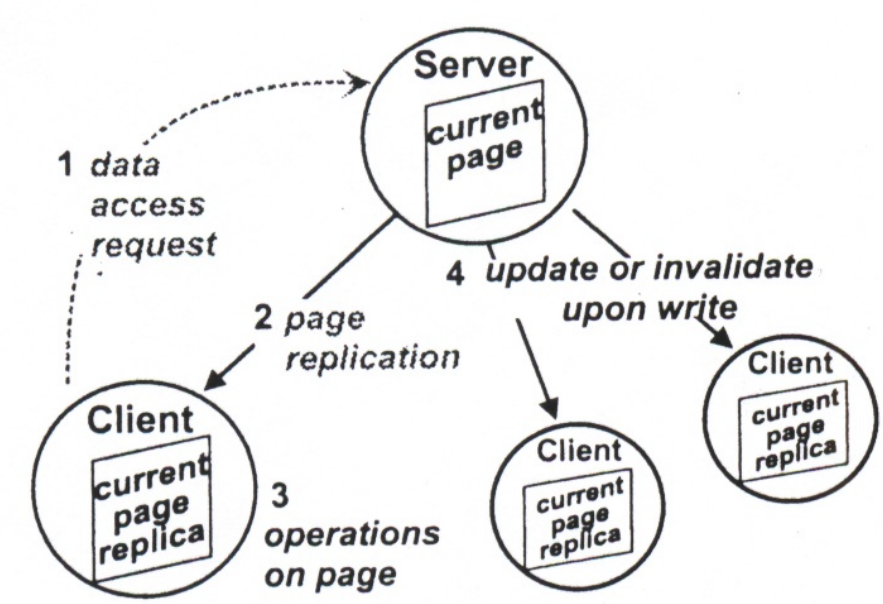
- minimizing the number of messages (now "remote object access" type instead of RPC)
- minimizing the latency

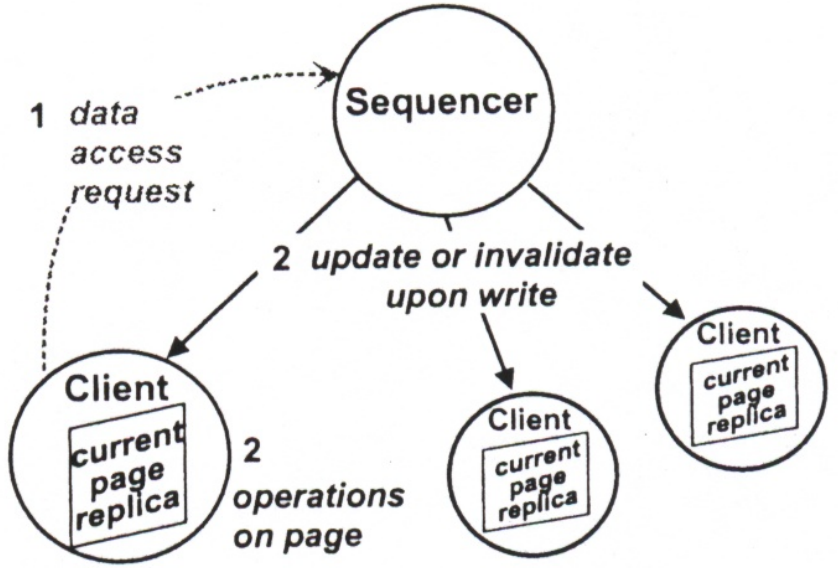
Various DSM Architectures implementing Atomic Consistency



Models of Distributed Computing (15)

Still, two clients cannot access the same page concurrently. More options:





Models of Distributed Computing (17)

6. Message Buses Model

Idea: allowing processes (characterized as publishers and subscribers) to communicate through an intermediate component, called the *bus*

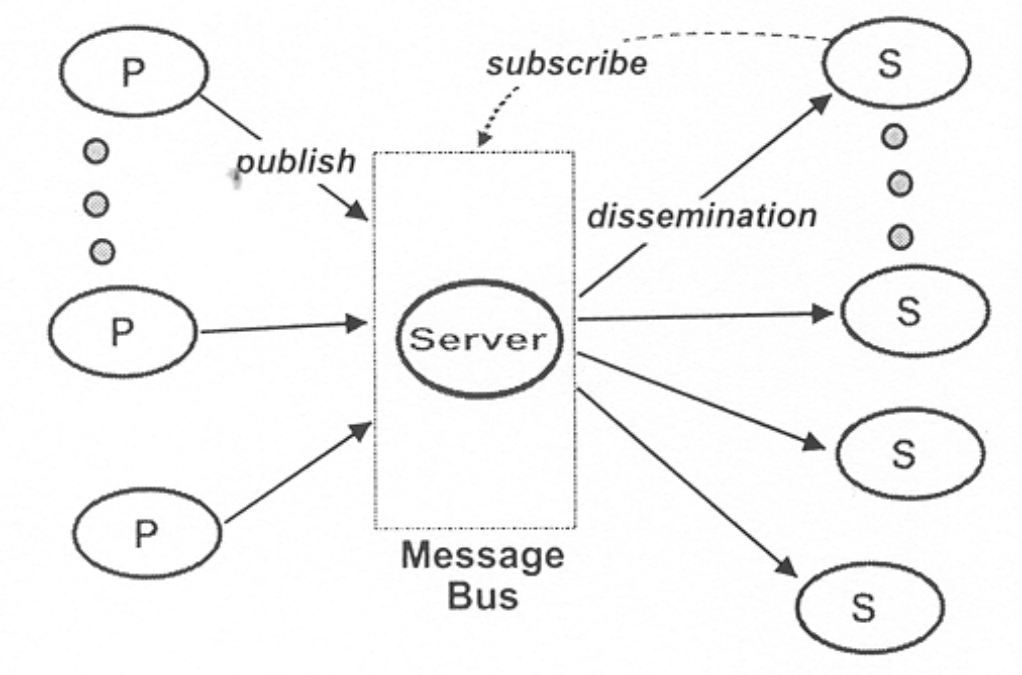
Much lower level communication abstraction than RPC and DSM

---> not adequate for complex distributed applications

Pro´s:

- simple and easy to understand
- allows non-synchronized interaction between message producer (*publisher*)and consumer (*subscriber*)
- easier to reconfigure and to scale (e.g. changing number, identity, location of subscribers transparent to the publishers and vice versa)

Simplest Implementation Architecture:



Inhalt

Teil 2: Verlässliche Verteilte Systeme

1. **Einführung zum Begriff Verlässlichkeit (Grundbegriffe)**

2. **Zuverlässige (Fehlertolerante) Systeme**

   - **Paradigmen**
     (Fehlererkennung, Kommunikation, Replikation, Fehlerbehebung)

   - **Modelle insbesondere für VS**
     (Transaktionen, Atomare Aktionen)

Dependability (1)

Computer System Dependability:
The *quality* of its delivered *service* is such that reliance (confidence) can justifiably be placed on this service

*Service***:**
The behavior of the system as it is perceived at the interface to its users

Quality:
The fact that the delivered service complies with the specified service

How to achieve dependability:

- understanding the *impairment*s, i.e. the potential causes for incorrect behavior of the system
- defining *measures* to express the level of dependability desired
- learning about the *mean*s that can be applied
  - what are the potential techniques to *procure* dependable behavior
  - how to *validate* whether the desired values are achieved.

Dependability (2)

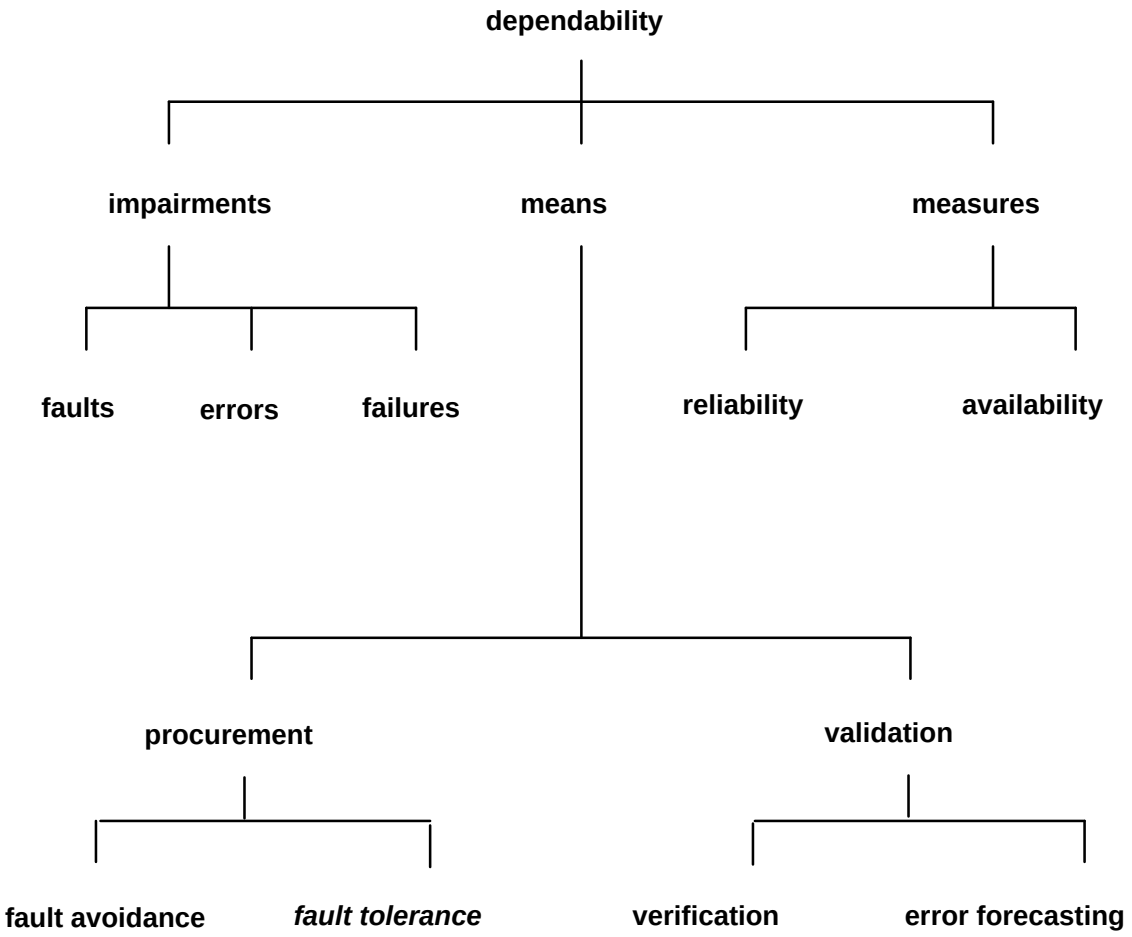


Fig. 2.1: Classification of dependability

Dependability (3)

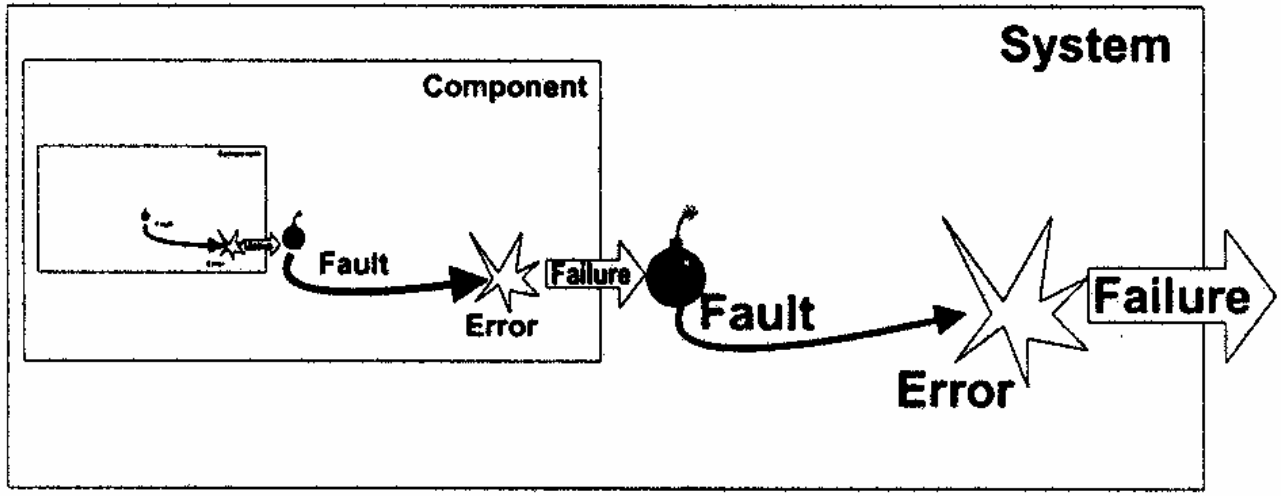
Impairments:

fault: occurrence of an irregular event in (some component of) the system

error: manifestation of the fault in (part of) the system state which becomes erroneous

failure: fact that the system behavior (delivered service) violates (deviates from) its specification

This chain can be applied recursively when the (distributed) system comprises several components:



Faults are subdivided into classes depending on whether the are caused by hardware, software, or interaction.

Dependability (4)

Measures :

Reliability: measure of the continuous correct (failure-free) service (behavior) of the system expressed by probabilistic functions like

MTTF (mean time to failure), MTBF (mean time between failure), or by giving a failure rate probability (e.g.10^{-9} failures per hour)

Maintainability: measure of the time to restoration of correct service (to recover from a failure)

Availability: measure of the probability of the system being operational at any given time.Given MTTR (mean time to repair), availability can be expressed as MTBF/(MTBF+MTTR)

Safety: the conditional probability that, given a failure, the system remains in a non-catastrophic state

Security: considered later

Dependability (5)

Means:

Fault prevention: eliminating the conditions that make fault occurrence possible at system design and implementation, e.g. using high quality component or design techniques (HW and SW)

Fault removal: detecting faults before they have a chance of causing an error (system testing)

Fault avoidance: combination of fault prevention and fault removal, i.e. aiming at a fault-free operational system

***Fault tolerance**:* making the system capable to provide correct service despite the existence of errors due to occurring faults. In consequence, complementary mechanisms must be provided that block the effect of errors before they generate failures. Applying these mechanisms is called *error processing.*

Error forecasting and *verification* should care for the confidence in the provided means. The former can be seen as complementing fault removal by predicting the amount of remaining faults in the system. The latter validates the correctness of the system, e.g. by applying formal methods, fault injection techniques, statistical trend analysis.

Fault Tolerance (1)

All methods for achieving fault tolerance depend on the effective deployment and utilization of redundancy.

Redundancy

denotes the enlargement of a system by additional resources only for realizing fault tolerance, i.e. they would not be required for the delivering of the specified service if it could be guaranteed that the system is free from faults.

Space (Structural) Redundancy

refers to the addition of several copies of the same HW or SW component

Value (Information) Redundancy

refers to the addition of extra information about the value of the data being stored or sent

Time Redundancy

refers to doing the same thing more than once, in the same or different, ways, until the desired effect is reached.

A still open question is when the redundant resources are actually used

Static Redundancy

denotes error (fault-tolerant) processing that is performed when the system is in the normal operation mode

Dynamic Redundancy

denotes error processing that is performed when the system is in the exceptional operation mode

Fault Tolerance (5)

Error Processing

Error Processing Techniques

error detection	detecting the error after it occurs aims at: confining it to avoid propagation; triggering error recovery mechanisms; triggering fault treatment mechanisms
error recovery	recovering from the error aims at: providing correct service despite the error
backward recovery:	the system goes back to a previous state known as correct and resumes
forward recovery:	the system proceeds forward to a state where correct provision of service can still be ensured
error masking	the system state has enough redundancy that the correct service can be provided without any noticeable glitch

Fault Tolerance (6)

Latent error processing (error compensation)

error correction (Fehlerkorrektur)

the erroneous state is provided with enough static information redundancy that allows the automatic correction upon activation

error masking (Fehlermaskierung)

enough static space redundancy is supplied which allows the masking of the latent error

Effective error processing (error recovery and error passivation)

damage assessment (Schadensbewertung)

assessing the extent to which the system state is already erroneous (e.g. due to error propagation)

error recovery (Fehlerbehebung)

the transformation of the current erroneous state into a well defined and error-free state from which the normal operation can continue

Fault Tolerance (7)

backward error recovery

restoring a prior, correct (error-free) system state

forward error recovery

producing a new correct system state

error passivation

taking care that the detected error cannot be activated again

fault localization

determining the component that was affected by a fault

reconfiguration

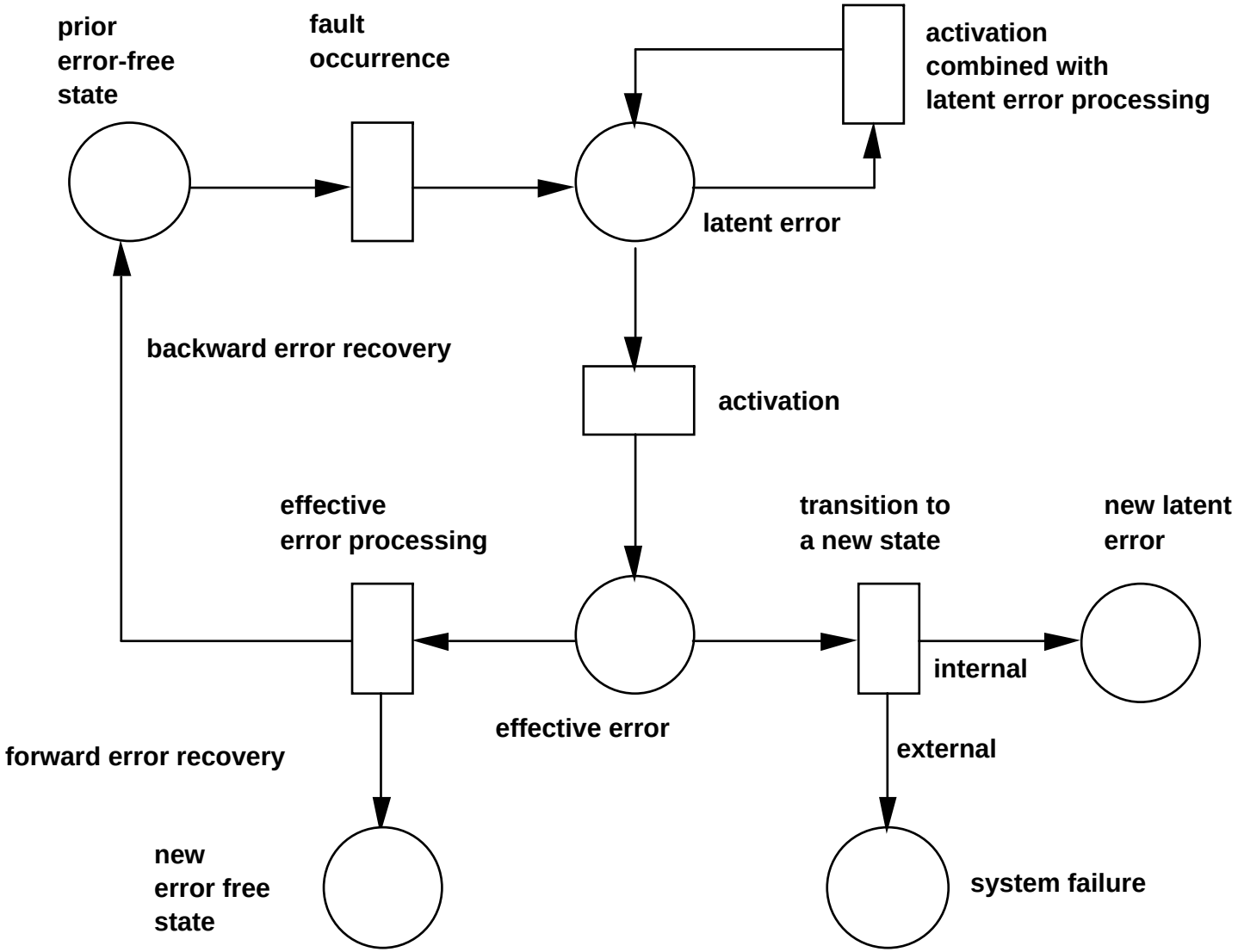
elimination of the corrupted component and either

replacing it by an identical one or

take over of the functionality of the eliminated component by another, already available component of the system

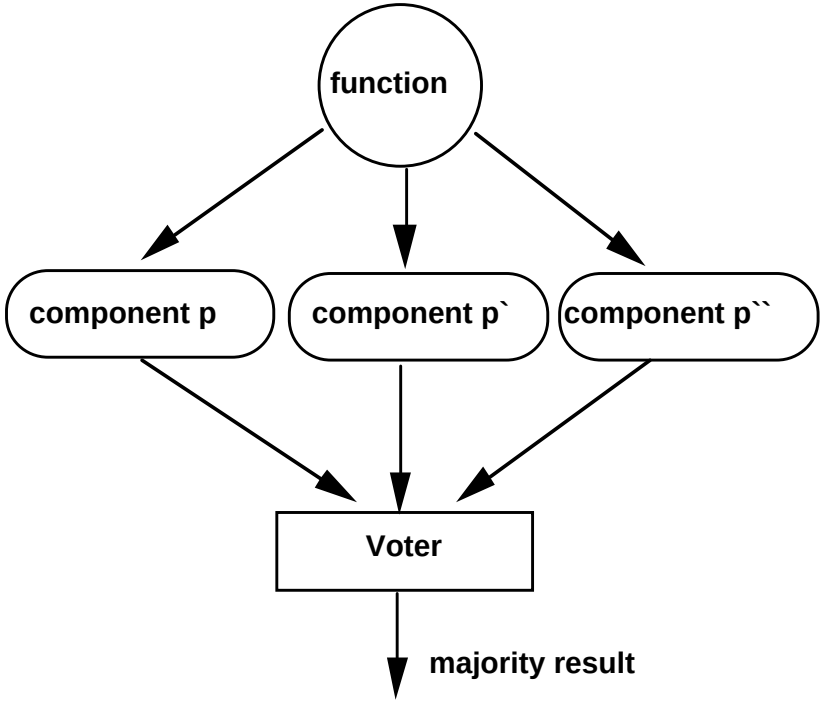
Fault Tolerance (8)

Transitions between different erroneous states



Fault Tolerance (9)

Illustration of the Triple Modular redundancy scheme (exploiting space redundancy)



The reliability of this scheme is $R_{TMR} = R_V * (R_C^3 + 3\ R_C^2(1 - R_C)$ where R_V and R_C denote the reliability of the voter and the three identical components, respectively.

Assuming that $R_V = 1$ ---> $R_{TMR} > R_c$ for $R_c > 0{,}5$.

If only one component is erroneous, ie. producing wrong results ---> $R_{TMR}' = R_C^2$

---> below the reliability each of the two still correctly working components

---> error passivation is highly recommended

Fault Tolerance (10)

Methods of Fault Tolerance