Intrusion tolerance

Prof. Cinzia Bernardeschi
Department of Information Engineering
University of Pisa

Pisa, 15 Novembre 2011

Classical security work:

- intrusion prevention
- intrusion detection

A new approach that emerged during the past decade:

- intrusion tolerance

Intrusion tolerance: coined by Joni Fraga and David Powell
"A Fault- and Intrusion-Tolerant File System", IFIP SEC,1985

What is intrusion tolerance?

Assume (and accept) that systems remain to a certain extent vulnerable

Assume (and accept) that attacks on components or subsystems can happen and some will be successful

→assures that the overall system neverthless remains secure and operational

(the tolerance paradigm in security)

Typical security properties are:

- Confidentiality
- Authenticity
- Integrity

The question is:

How do we let data be read or modified by an intruder; and still ensure confidentiality, authenticity or integrity?

An **intrusion-tolerant system** can maintain its security properties despite some of its components being compromised.

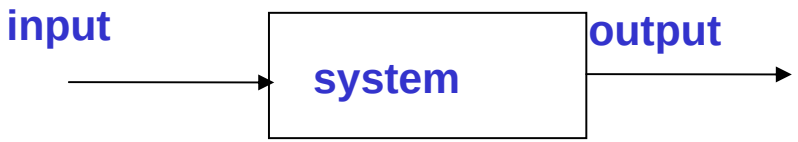
Appeal:
since it's impossible to prove that a system has no vulnerabilities, it is more safe to assume that intrusions can happen.

Reliability & Security

The “tolerance” paradigm was first applied in Reliability

Reliability and fault tolerant computing:

a fault-tolerant system is a system that can continue to deliver a correct service in the presence of faults



System service

what the system is intended for. It is given by the input-otput relation

Types of faults

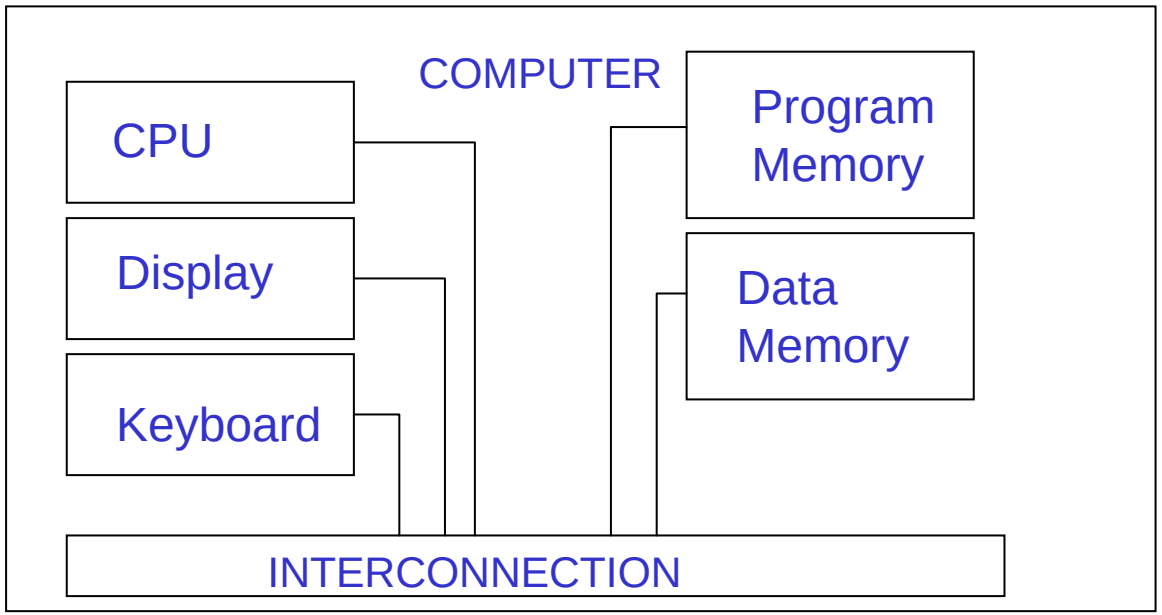
- Chip (motor) suffers permanent electrical damage
- Undersized fan (design fault) allows overheating on hot days: chip malfunction (physical fault); the machine work OK after cooling down (the fault is transient)
- Operator pushes the wrong button
- One or two fans “burns out” (physical fault)
- Repairman switches off the wrong fan (maintenance fault)
- Cosmic ray particle causing transient “upsets” in execution
- Defect in software
- Software virus

A fault tolerant system is designed to keep working even when some of its components fail

Systems and components

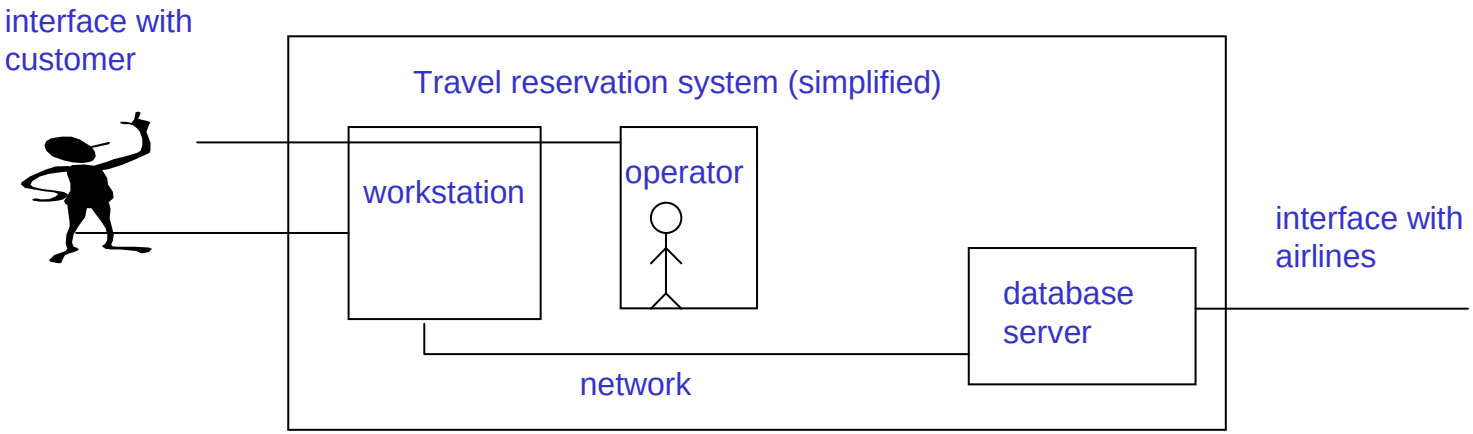
A system is made out of components

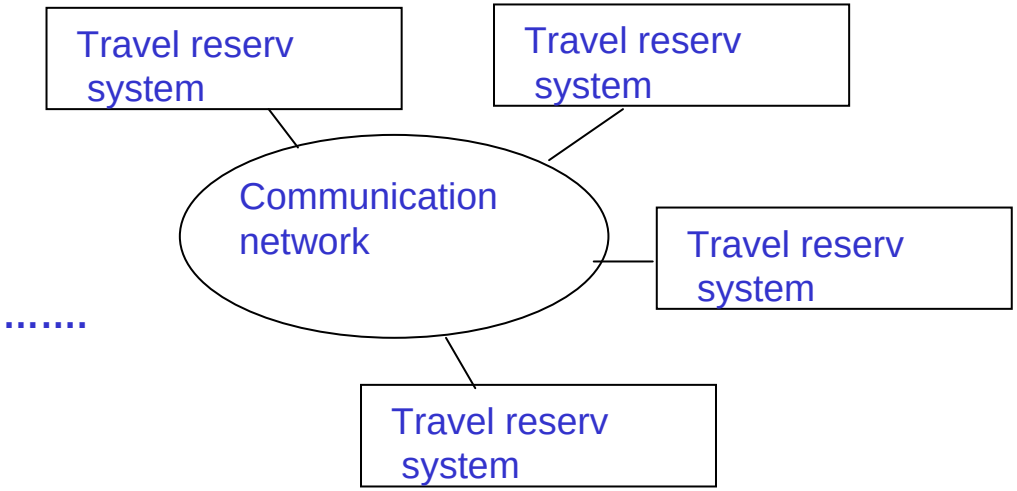
For instance:



Each component is a system in its own right

Other examples





Fault, error and failure

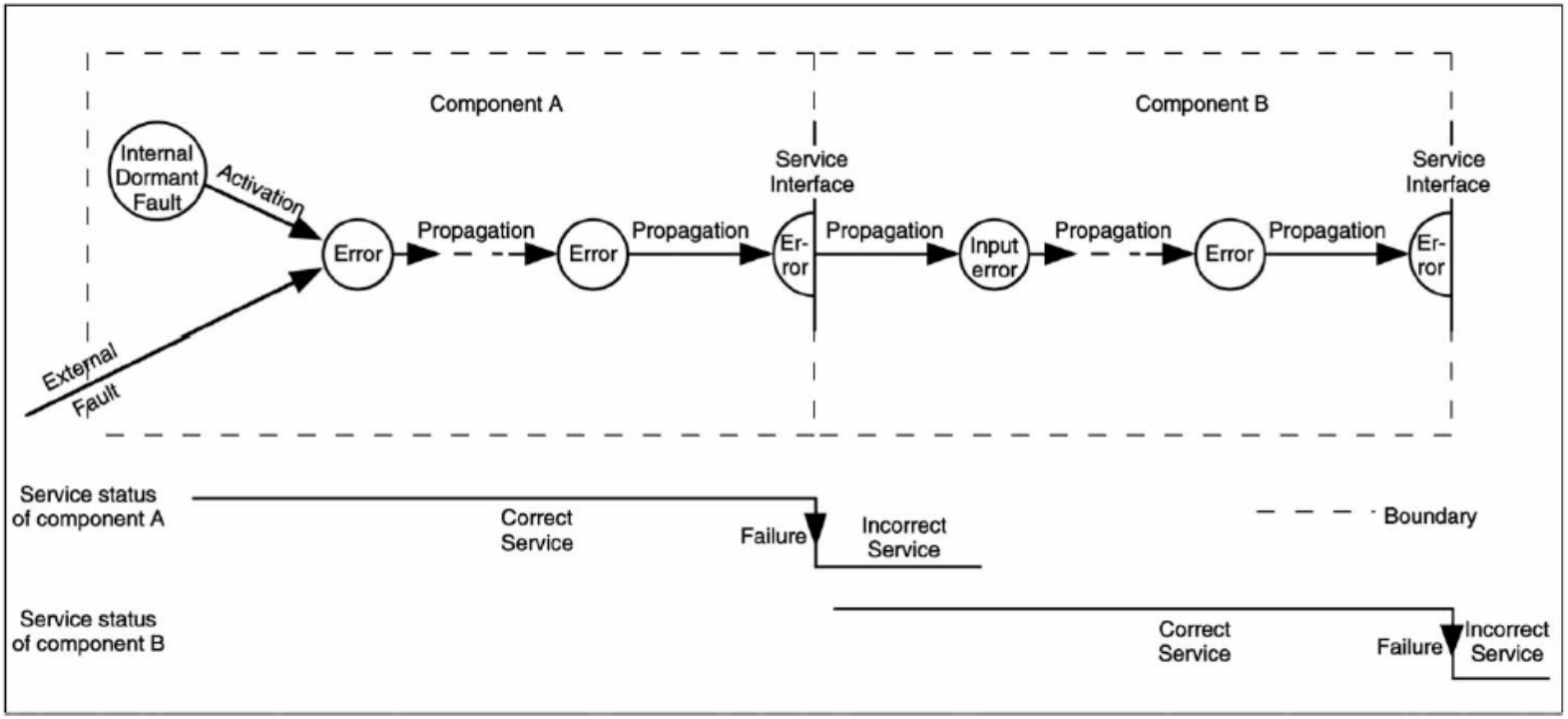
- A **system** is designed to provide a certain **service**

- If the system stops delivering the intended service, we call this a **failure** (the correct service may later restart)

- For instance, a computer
  - prints the wrong number
  - puts garbage in a file, stops working
  - stalls the car's engine or revs it too fast
  - delivers 200 euro when you asked for 10 euro

- We we call the cause of failures, **faults**
- A fault causes an **error** in the internal state of the system. The error cause the system to fail

Fault / error/ failure chain



From: A. Avizienis, J.C. Laprie, B. Randell, C.E. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing.IEEE Trans. Dependable Sec. Comput. 1(1): 11-33 (2004)

Computer failures

Computer failures differ from failures of other equipment

Subtler failure than breaking down, stopping working

the computer is used to store information: there are many ways information can be wrong, many different effects both within and outside the computer

Transient faults have permanent effects

Small, hidden faults may have large effects (digital control systems)

Computing systems are complex design hierachies relying on hidden components

Reliability & Security

Reliability and Security are different aspects of Dependability

Dependability
is “that property of a computer system such that reliance can justifiably be placed on the service it delivers”

J.C. Laprie (ed.),
Dependability: Basic Concepts and Terminology, Springer-Verlag, 1992

Dependability

- **Reliability of a system**
  a measure of the continuous delivery of correct service

- **Availability of a system**
  a measure of the alternation between delivery of correct service and incorrect service

- **Safety of a system**
  a measure of the time to catastrophic failure

- **Security of a system**
  prevention of unauthorized access and/or handling of information

Another attribute is:

- **Maintainability**
  a measure of the time to restoration

Dependability Measures

The different dependability properties can be measured in terms of probabilities:

- **Reliability of a system**
  the reliability of a system as a function of time, R(t), is the conditional probability that the system performs correctly throughout the *interval of time* [t0, t], given that the system was performing correctly at time t0.

- **Availability of a system**
  the availability of a systems as a function of time, A(t), is the probability that the system is operating correctly and is available to perform its functions at the *instant* of time t.

  (difference: repairs affect availability not reliability)

Development of high dependable systems

- **Safety critical systems**
  systems in which the computer is serving a critical function that cannot be suspended even for the duration of a repair (as in railway-signalling control systems, flight computers on aircrafts, ...)

- when digital control systems (DCS) are used in safety critical applications, we need to have sufficient confidence that the system will fail
  - **sufficiently infrequently** or
  - **with sufficiently low probability during a particular length of time**

- Examples of Reliability requirements in safety critical systems:
  - Flight control systems (excluding software faults)
    10^{-9} prob. of failure per flight hours
  - Railway signalling and control
    10^{-12} prob. of failure per hour for urban trains
  - .....

L. Strigini and B. Littlewood, Validation of Ultra-High Dependability for Software-based Systems, Communications of the ACM, Vol. 36, No. 11, November 1993.

Digital control systems

- **REDUNDANCY**
- **DIVERSITY**

defense-in-depth approach

- **INDEPENDENCE**

different kinds of diversity on the assumption of maximal independence of redundant channels realizing design versions

Common Cause Failure
a failure of two or more structures, systems or components due to a single specific event or cause (fault)

Defense-in-depth approach

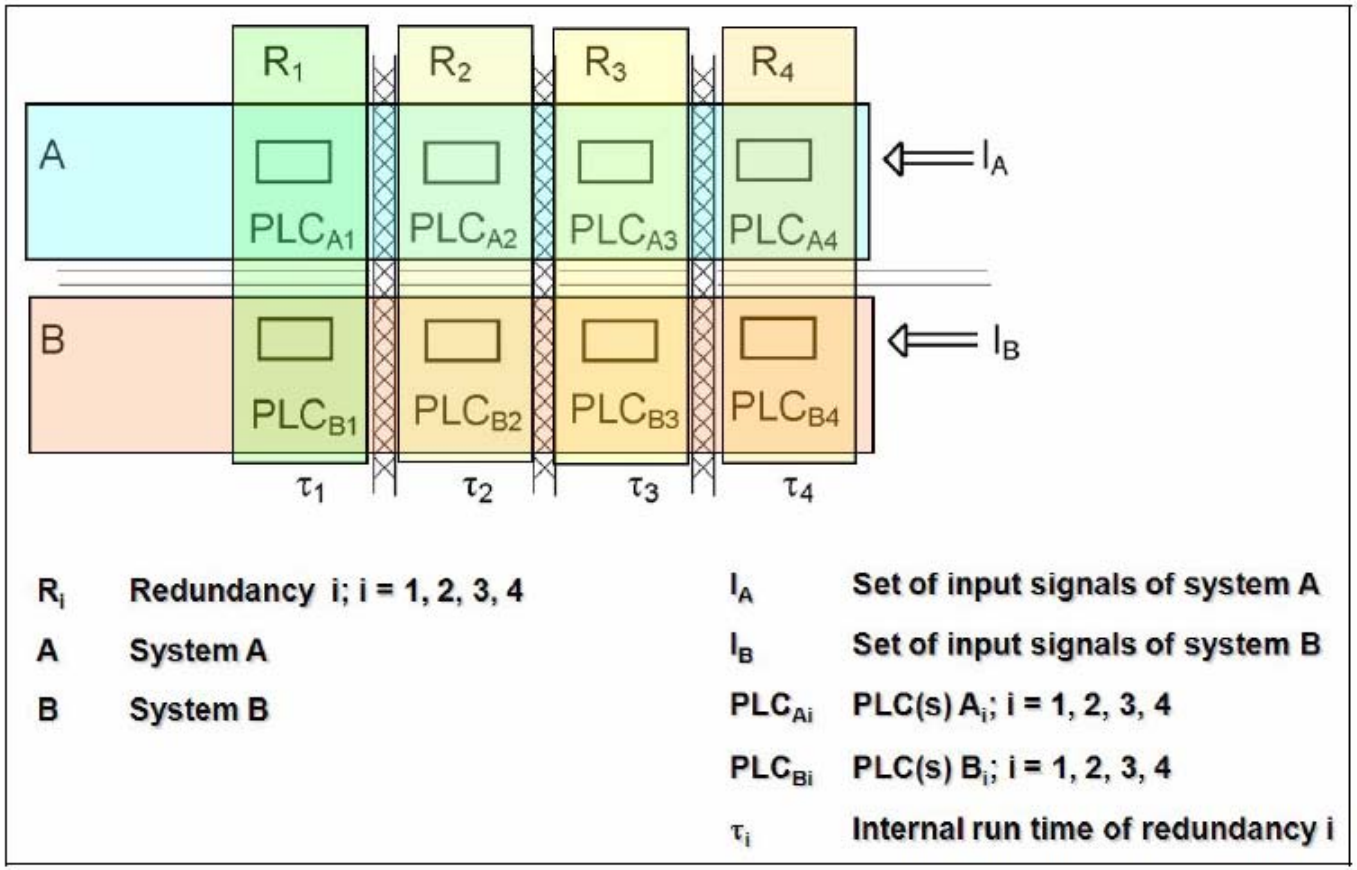
Redundancy, diversity and independence are basic concepts of a **defense-in-depth** approach:

1. **redundancy**
alternative systems and components are included, so that any one can perform the required function if the others fail

2. **diversity**
for a particular function, two or more redundant systems or components with different attributes are included in the design. In practice, it may mean using different components based on different designs and principles, from different vendors

3. **independence**
prevents the propagation of failures and common cause failures due to common internal plant hazards. Independence is achieved by electrical isolation, physical separation and independence of communications between systems

Scheme of a digital safety I&C system



From “The standard IEC 62340 Nuclear Power Plant – I&C system important to safety – requirements for coping with CCF”, A. Lindner, H-W Bock, NPIC&HMIT 2009, American Nuclear Society.

- Scheme of a fourfold redundant safety I&C system comprising two independent I&C systems A and B

- The channels of independent systems and the systems are spatially separated

- The systems A and B can be designed by
  - different Programmable Logical Controllers
  - different set signals
  - different I&C functions

- Replicas have different starting time to avoid failures triggered by the same runtime

Functional diversity

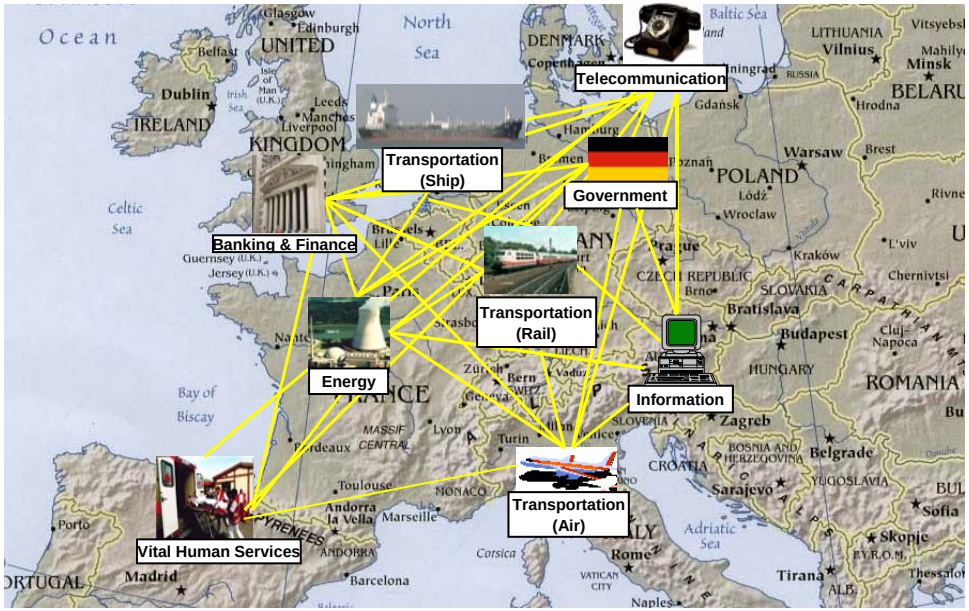
Existence of diverse measurement signals, actuators and functions in the plant system

Diverse functions:
functions that ensure independently that the plant safety targets are met

Diverse I&C functions have to be assigned to independent I&C systems

From Dependability to Resilience

Ubiquitous systems
consider large, networked, evolving, systems either fixed or mobile, with demanding requirements driven by their domain of application (critical infrastructures, transportation systems, …)



Resilience

- resilience builds on the initial definition of dependability and cover the ability to successfully accommodate unforeseen environmental perturbations or disturbances

- A shorthand definition of Resilience is:
  - the persistence of dependability in spite of continuous changes

Security

Fault Model for Security

Attacks, Vulnerabilities, Intrusions

An **intrusion** has two underlying causes:

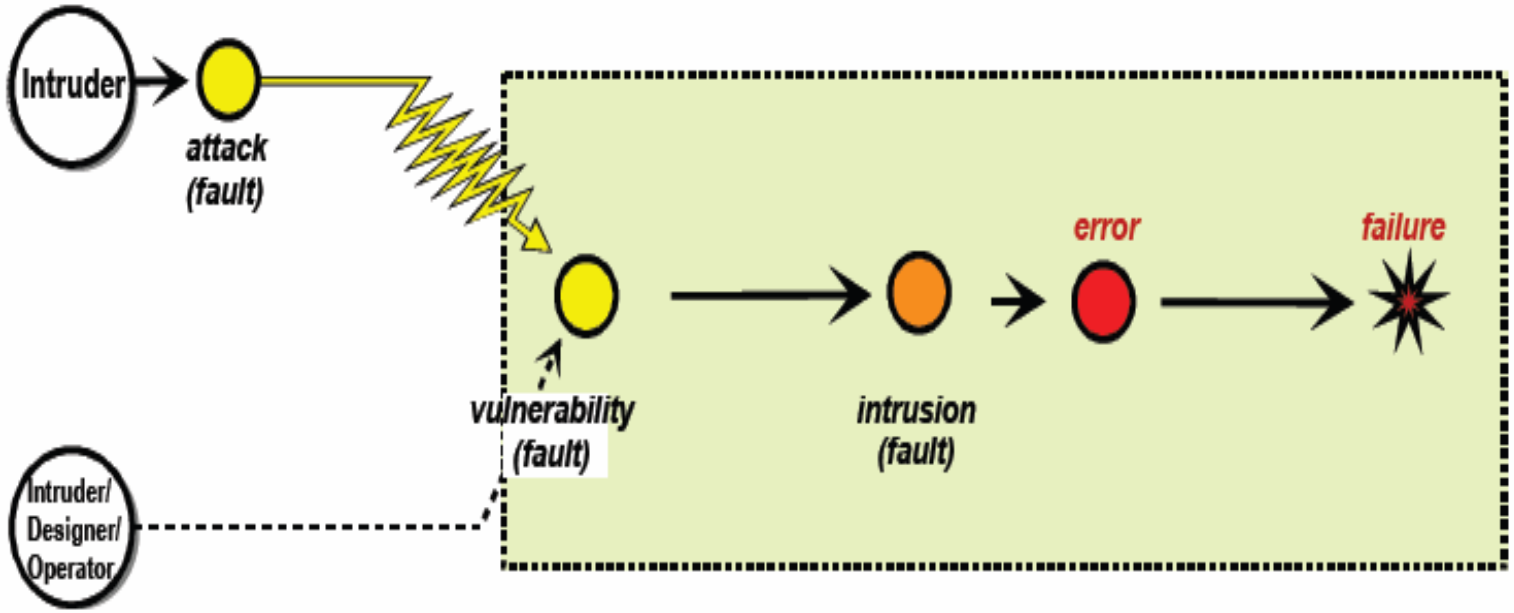
- **Vulnerability**
  - malicious or non-malicious weakness in a computing or communication system that can be exploited with malicious intention

- **Attack**
  - malicious intentional fault introduced in a computing or communication system, with the intent of exploiting a vulnerability in that system

Interesting corolaries:
- without attacks, vulnerabilities are harmless
- without vulnerabilities, there cannot be successful attacks

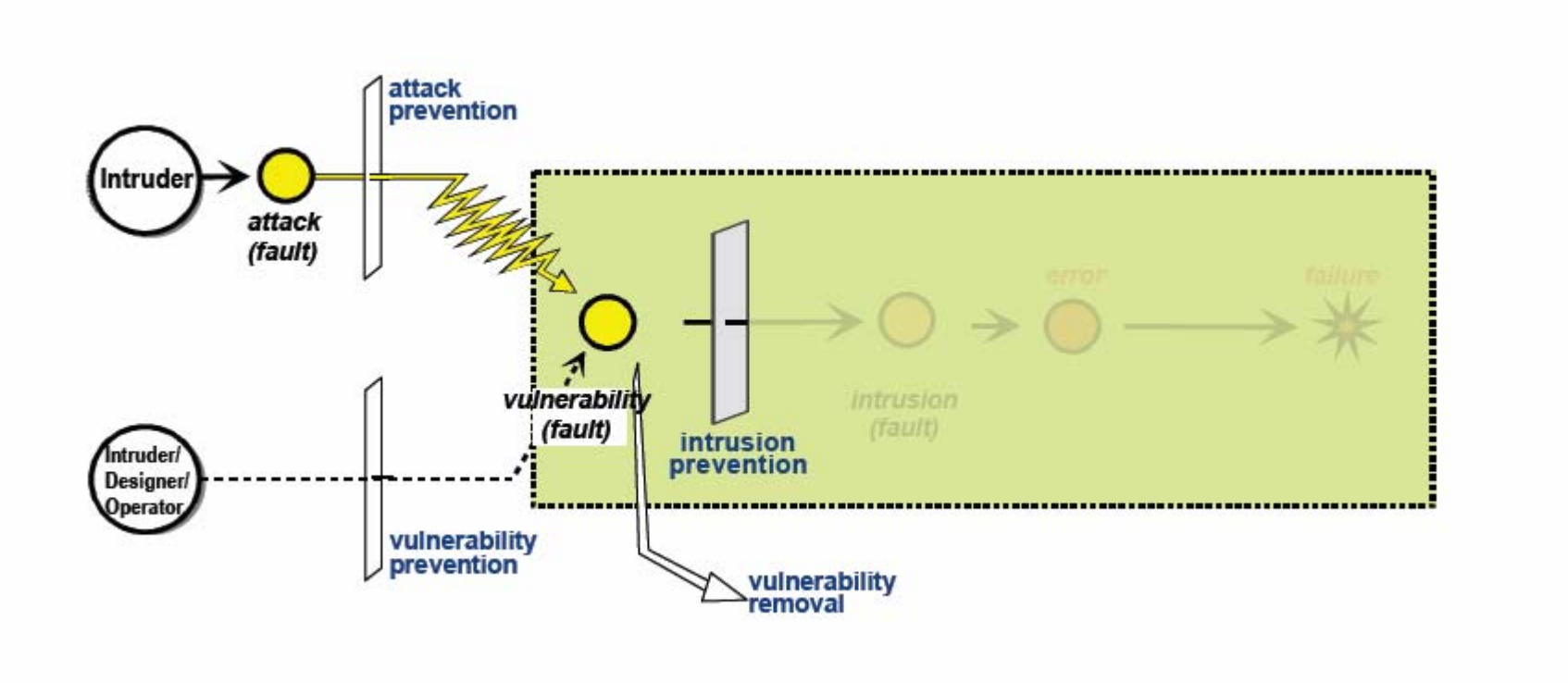
Fault Model for Security

AVI sequence:
attack + vulnerability
-> intrusion -> error -> failure



A specialization of the generic
"fault,error,failure" sequence

Intrusion prevention



Attack: **malicious intentional external fault** attempted at a computer or communication system

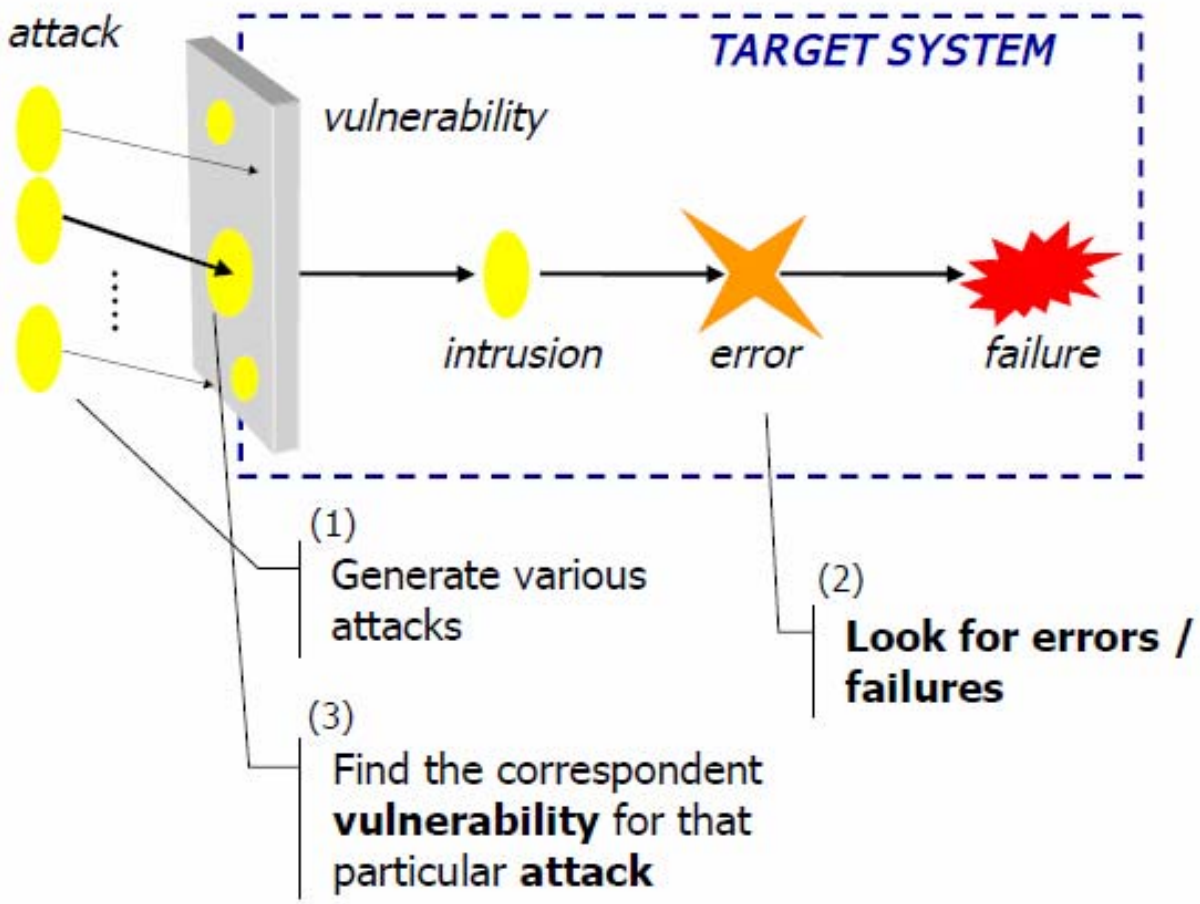
Vulnerability: internal fault

Intrusion prevention

Traditionally security has evolved as a combination of:

- Attack prevention
  ensuring attacks do not take place against certain components
- Attack removal
  taking measures to discontinue attacks that took place
- Vulnerability prevention
  ensuring vulnerabilities do not develop in certain components
- Vulnerability removal
  eliminating vulnerabilities in certain components (e.g. bugs)

Intrusion prevention: using attacks to find vulnerabilities



Risk of intrusion

Risk = probablity of intrusions + serverity of intrusions

Probability of an intrusion: given by the probability of an attack activating a vulnerability that is sensitive to it.
It depends on:

- level of threat to which the system is exposed
- degree of vulnerability of the system.

Serverity of intrusions: the impact of a failure caused by the intrusions

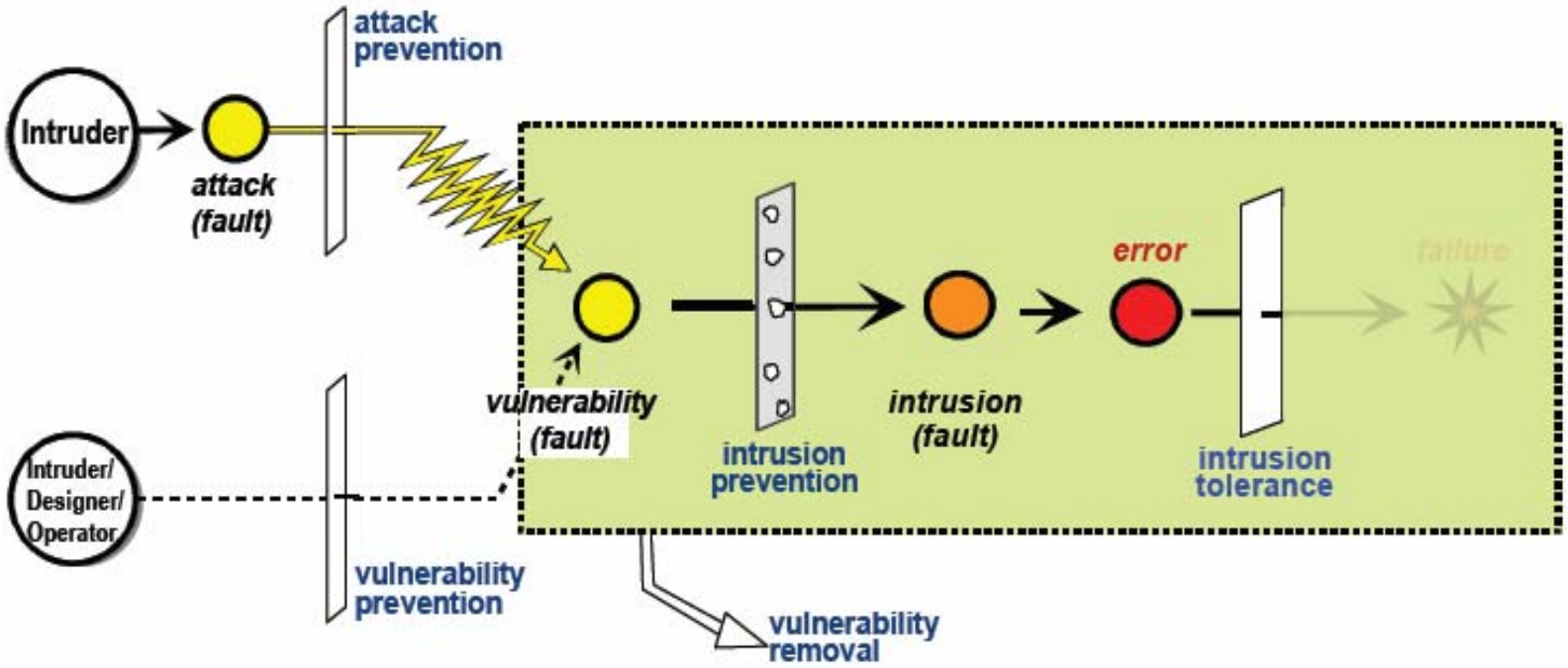
Should we try and bring the risk yo zero? Is that feasible at all?

-> it is too costly and/or too complex

We talk about acceptable risk: a measure of the failure probability we are prepared to accept, given the value of of the service or data we are trying to protect.

Intrusion tolerance (IT)

Intrusion: operational intentional fault that leads to component failure



Effort is put in tolerating residual intrusions in the system

Ensure continued correct service provision despite attacks, vulnerabilities and intrusions

Error processing

Errors must not lead to system failures

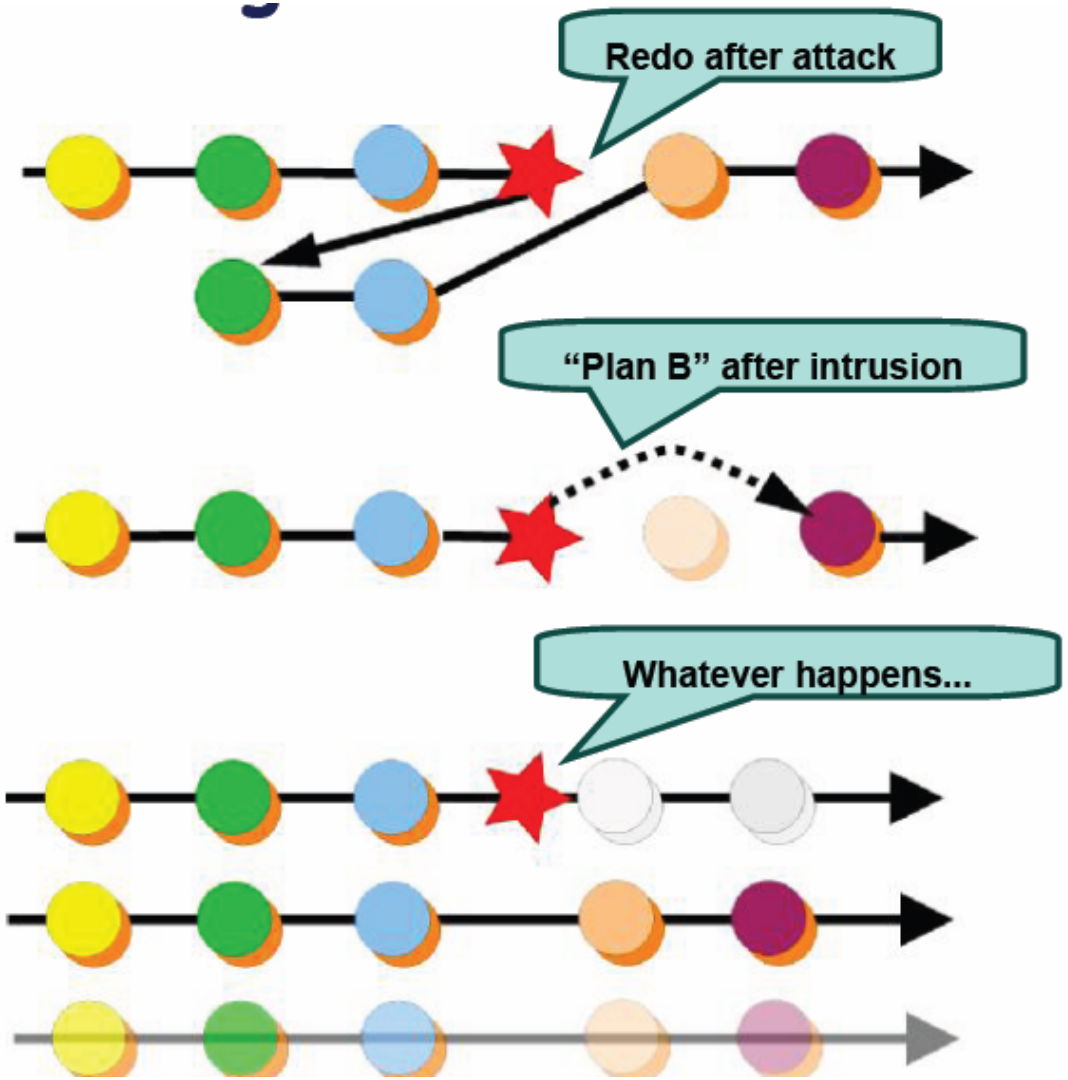
Error detection and recovery

- Backward recovery
- Forward recovery

Error masking

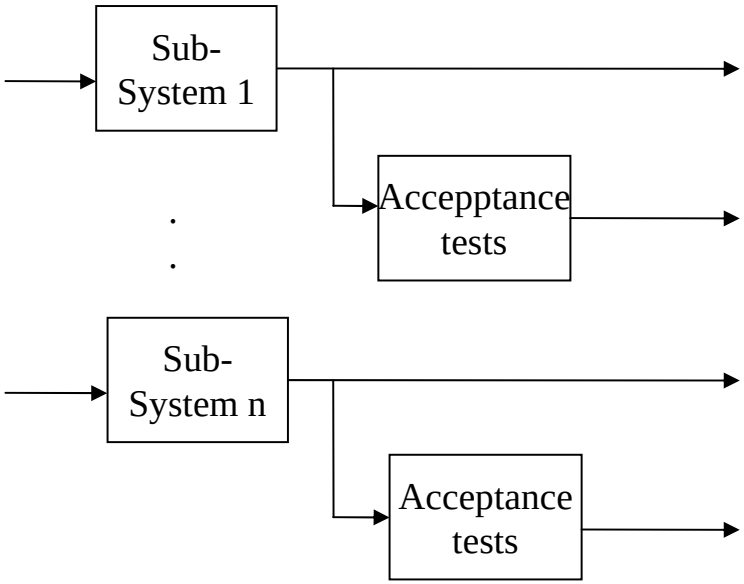
Error processing

- backward recovery
- forward recovery
- error masking



Error detection

- for example, based on acceptance tests



Error processing

- **Forward recovery**
  transform the erroneous state in a new state from which the system can operate correctly

- **Backward recovery**
  bring the system back to a state prior to the error occurrence
  - Checkpointing
  - Recovery block

- **Error masking**
  redundancy techniques

Forward error recovery

- Requires to assess the damage caused by the detected error or by errors propagated before detection

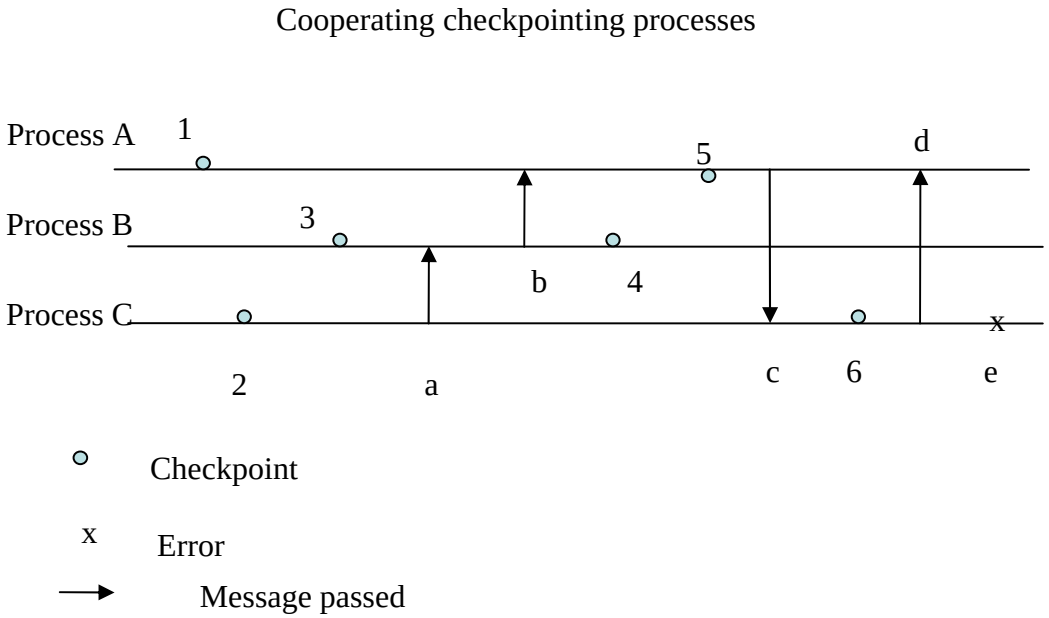
Example of application:

real-time control systems, an occasional missed response to a sensor input is tolerable

The system can recover by skipping its response to the missed sensor input.

Backward error recovery
Checkpointing

- some subset of the system state is saved at specific points during program execution (that is necessary to the continued successful execution)



- Rollback: resetting the system and process state to the state stored at the latest checkpoint.

Checkpointing

- Loss: computation time between the checkpointing and the rollback; data received during that interval

- overhead of saving system state/computation time for rollback

- *Basic issues*:
  - selecting checkpoints to minimize the amount of state information that must be saved
  - deciding which information must be backed up

- restraining multiple rollbacks that arise when multiple concurrent processes communicate with each other (domino effect)

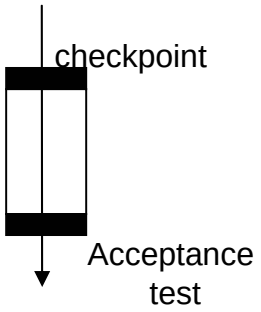
Backward error recovery

Recovery block

- Accettability of the result is decided by an acceptance test **T**

- Basic structure:

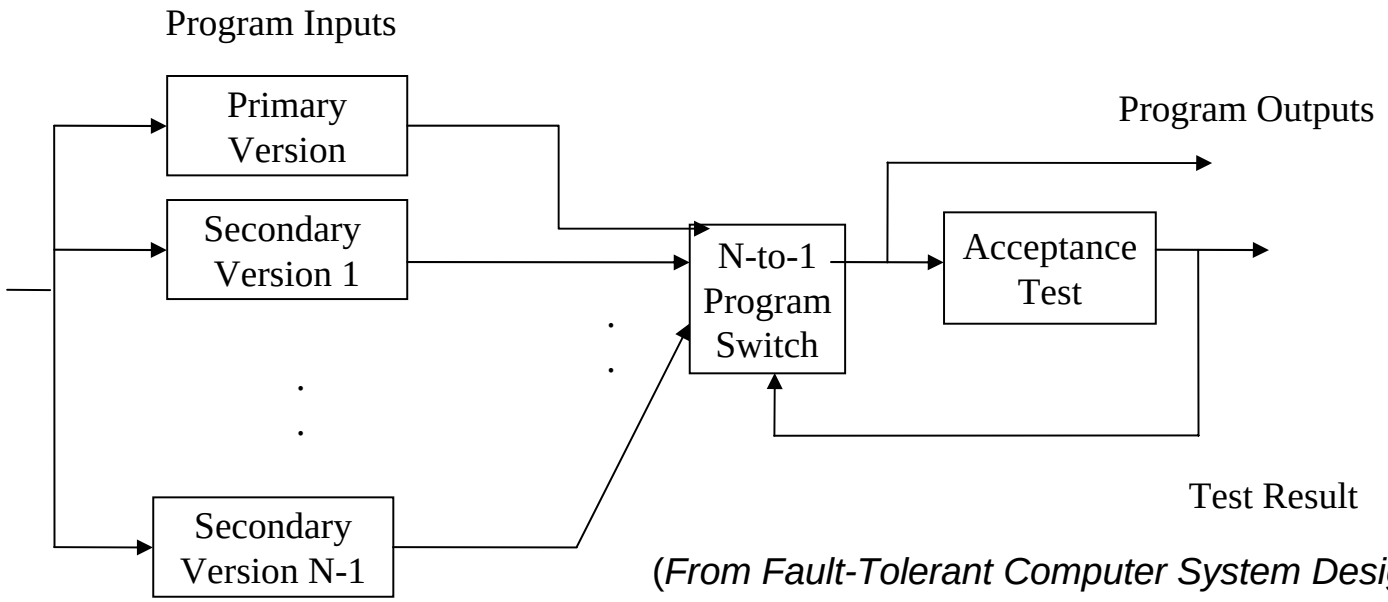
ensure T
by P
else by Q
else error



- Each recovery block contains variables global to the block that will be automatically checkpointed if they are altered within the block.
- Upon entry to a recovery block, the primary alternate is executed and subjected to an acceptance test (T) to detect any error in the result.
  If the test is passed, the block is exited.
  If the test is failed or the alternative fails to execute, the content of the recovery cache pertinent to the block is reinstated, and the second alternate is executed.
  This cycle is executed until either an alternative is successful or no more alternatives exist. In this case an error is reported.

- Only one single implementation of the program is run at a time

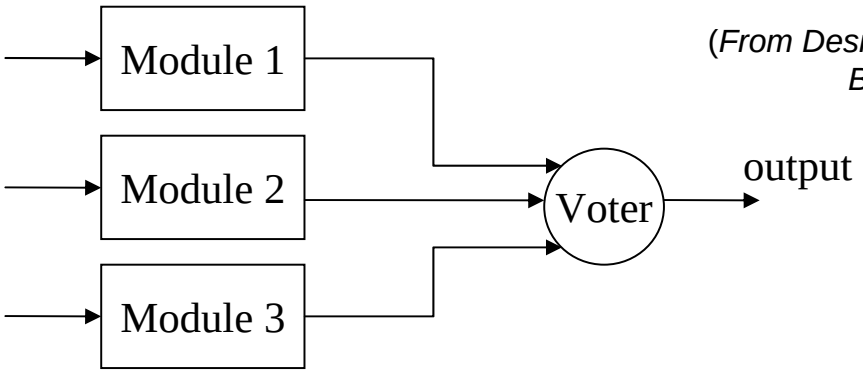
Backward error recovery: Recovery block



(*From Fault-Tolerant Computer System Design D. K. Pradhan, Prentice Hall, 1996)*

- A single acceptance test
- Combines elements of checkpointing and backup
- Minimizes the information to be backed up
- Releases the programmer from determining which variables should be checkpointed and when

Error masking
Triple Modular Redundancy (TMR)



(*From Design and Analysis of Fault-tolerant Digital Systems*
B. W. Johnson, , Addison-Wesley 1989)

Triplicate the hw (processors, memories, ..) and perform a majority vote to determine the output of the system

→ effects of faults neutralised without notification of their occurrence

→ masking of a failure in any one of the three copies

Sometimes some failures in two or more copies may occurr in such a way that an error is avoided:

Example

- stuck at 1 in a module line; stuck at 0 in another copy;
the voted result is correct (**compensating failures)**
- failure at location 127 in a memory; failure at location 10 in another copy;
the voted result is correct

Intrusion tolerance (IT)

How do we model the mind of an hacker?

The hacker is the perpretator of attacks on systems, a fault model would be a description of what he/she can do.

In the general case, we can not make assumptions about how the hacker can act, a malicious-fault modelling methodology is required.

Fault models

Node failures

- -Byzantine
- -Crash
- -Fail-stop
- -...

Communication failures

- -Byzantine
- -Link (message loss, ordering loss)
- -Loss (message loss)
- -...

Byzantine

Processes :

- can crash, disobey the protocol, send contradictory messages, collude with other malicious processes,...

Network:

- Can corrupt packets (due to accidental faults)
- An attacker can modify, delete, and introduce messages in the network

Architecting IT systems

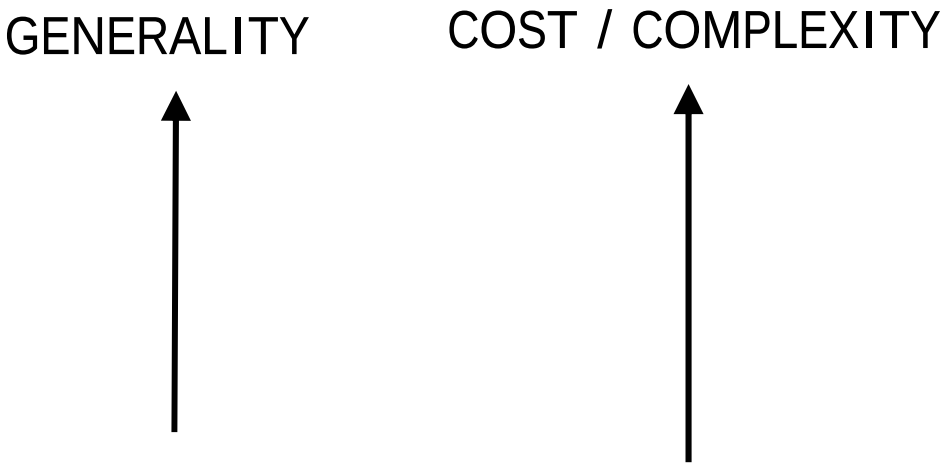
The more general the fault model, the more costly and complex the solution (for the same problem)

Byzantine

Crash

Fail-stop

No failure



How do we model the mind of an hacker?

Arbitrary failure approach (Byzantine failure mode)

Architecting IT systems

We must consider the system model:

- Asynchronous
- Synchronous
- Partially synchronous
- …

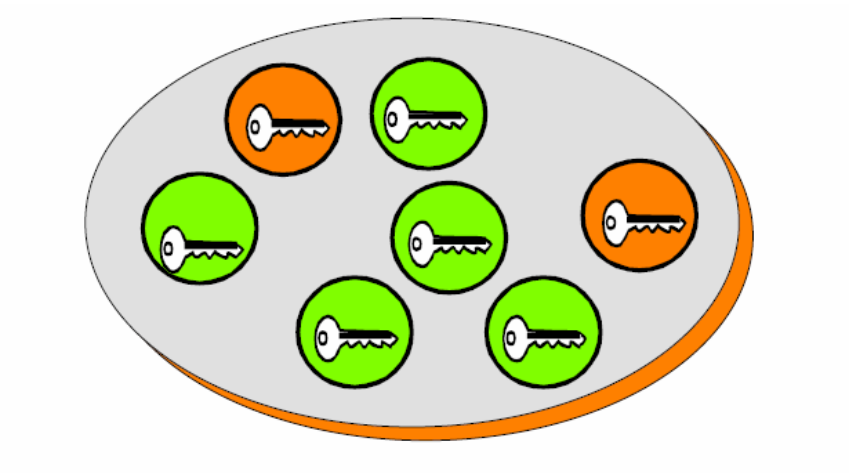
Useful building blocks for the architect of IT systems:

- Tunnels, secure channels and envelopes
- Firewalls
- Criptographic protocols
- …..

Architecting IT systems

- **Topological separation makes the intruder life more difficult, in term of attack effort**

  a secret can be split through several sites, and getting parts of it reveals nothing about the whole



Threshold cryptography

Threshold cryptography

Given N processes each holding part of crypto secret

Secret sharing:

- Example: a shared secret key
- Any k-out-of-N processes combine their shares and reconstruct secret s
- Any f=k-1 colluding or intruded processes cannot reconstruct s

Function sharing:

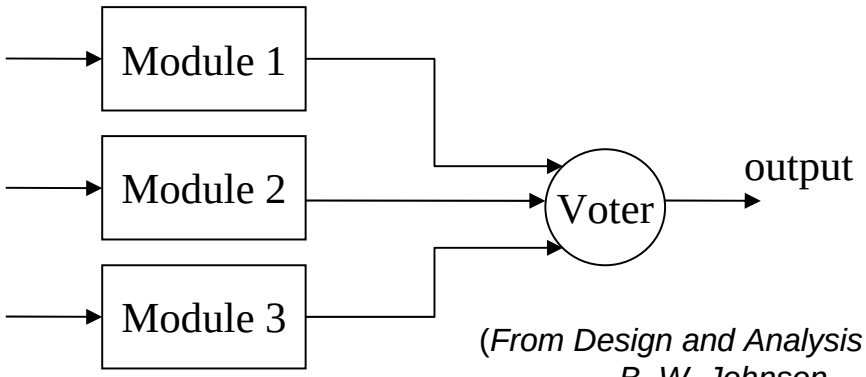
- Example: a threshold signature
- k processes together execute function F
- f=k-1 colluding or intruded processes cannot execute F

Architecting IT systems

- **Replication makes components more resilient to damages in terms of integrity and availability, but can also benefit confidentiality and authenticity**

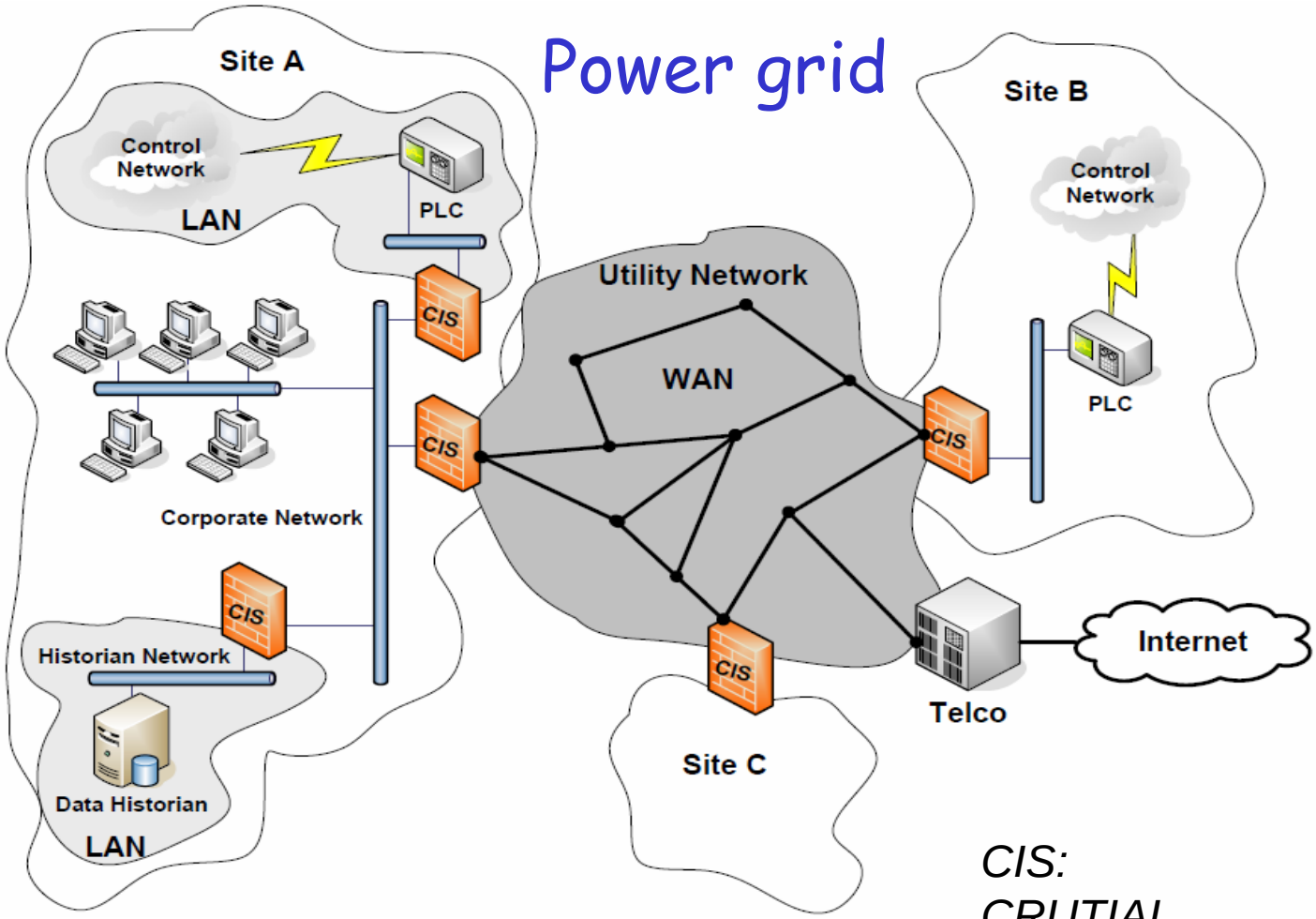
  (e.g. replicated execution decisions to decide authorization of access to a piece o data)

Basic replication scheme: Triple Modular Redundancy scheme (TMR)



(*From Design and Analysis of Fault-tolerant Digital Systems B. W. Johnson, , Addison-Wesley 1989*)

Intusion-tolerant protection for Critical Infrustructure



WAN of LANs and Computers

Security policy

CIS: CRUTIAL Information Switch

CIS: Intrusion-Tolerant Firewall

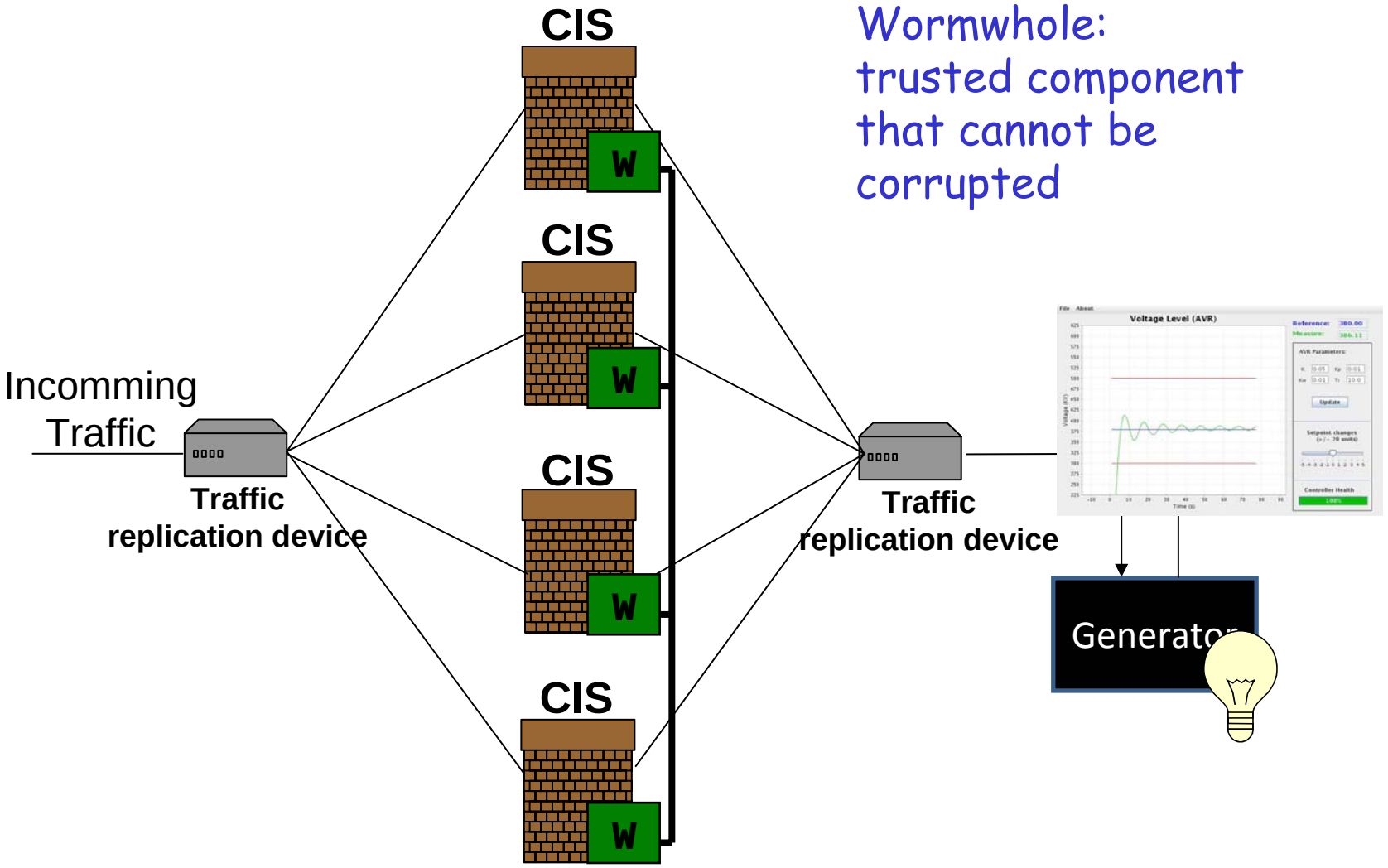
- CIS deployed on the network border and inside the network to protect critical equipment

- CIS uses a rich control access model
  take into account different organizations and access control rules that depend on context information

- CIS is intrusion tolerant:
  operates correctly even if there are intrusions in some of its components

CIS: architecture

- replicated in a set of different machines
-uses wormwhole W



Wormwhole:
trusted component
that cannot be
corrupted

CIS architecture

- CIS is replicated in a set of n >= 2f +1 machines

- Each replica receives all packets to and from the LAN and verifies if this packet satisfies some predefined application level security policy

  The "Traffic replication" devices are responsable of broadcasting the Wan and LAN traffic to all replicas

- Intrusions modelled as Byzantine faults in at most f replicas are masked (all valid packets are accepted and all invalid packets are dropped)

- Local wormhole are connected thrugh an isolated network

- Each CIS replica deployed in a different OS
  OSs use different passwords and internal firewalls

CIS architecture

Challanges:

- CIS cannot modify the protocols themselves to obtain intrusion tolerance

- Recipient nodes ignore any internal CIS intrusion tolerance mechanism

- Recipients cannot protect themselves from messages forwarded by faulty replicas not satisfying the security policy

CIS Intrusion-Tolerant Firewall

- A Msg approved by a replica is sent to the W
- Local Ws vote between themselves. If the message is approved by at least f+1 replicas, it is signed using a secret key installed in the trusted component.

- One of the replica (leader) is responsable of forwarding the approved msg to the its destination.

- Failure detection, leader election and recovery are services provided by the wormwhole.

- When a quorun of replicas suspect some replica, it is recovered.

Architecting IT systems

- **Byzantine fault-tolerant (BFT) algorithms**

  are intrusion tolerance devices: they perform error processing or masking and ensure message delivery despite actual intrusions

  they do not depend on trusted components for their correct operation

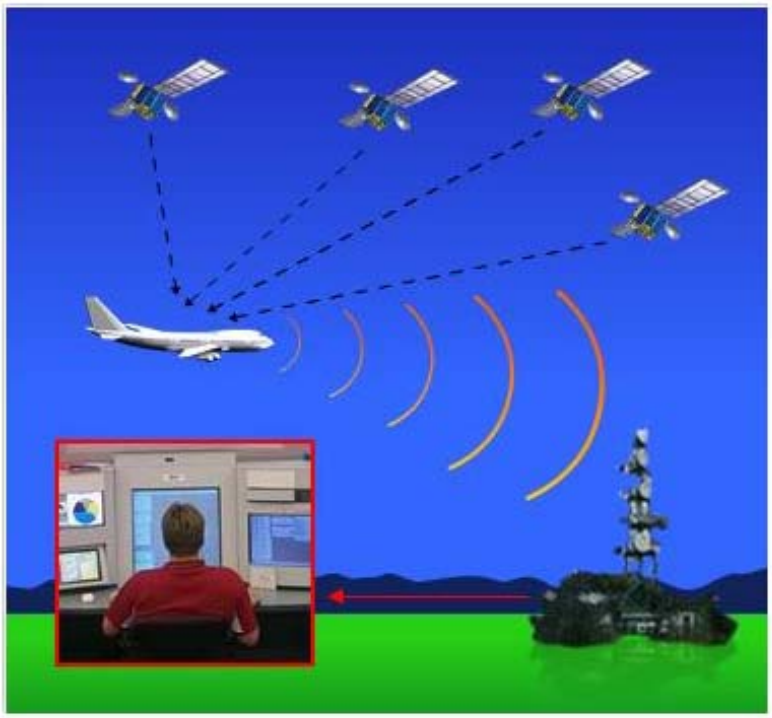
  they must build trust during execution without trusting each other initially, and some maybe being malicious

  they tolerate attacks, intrusions and bugs

Air Traffic Management

- Air Traffic Control (ATC) is a service provided by ground-based controllers who are responsible for maintaining a safe and efficient air traffic flow.

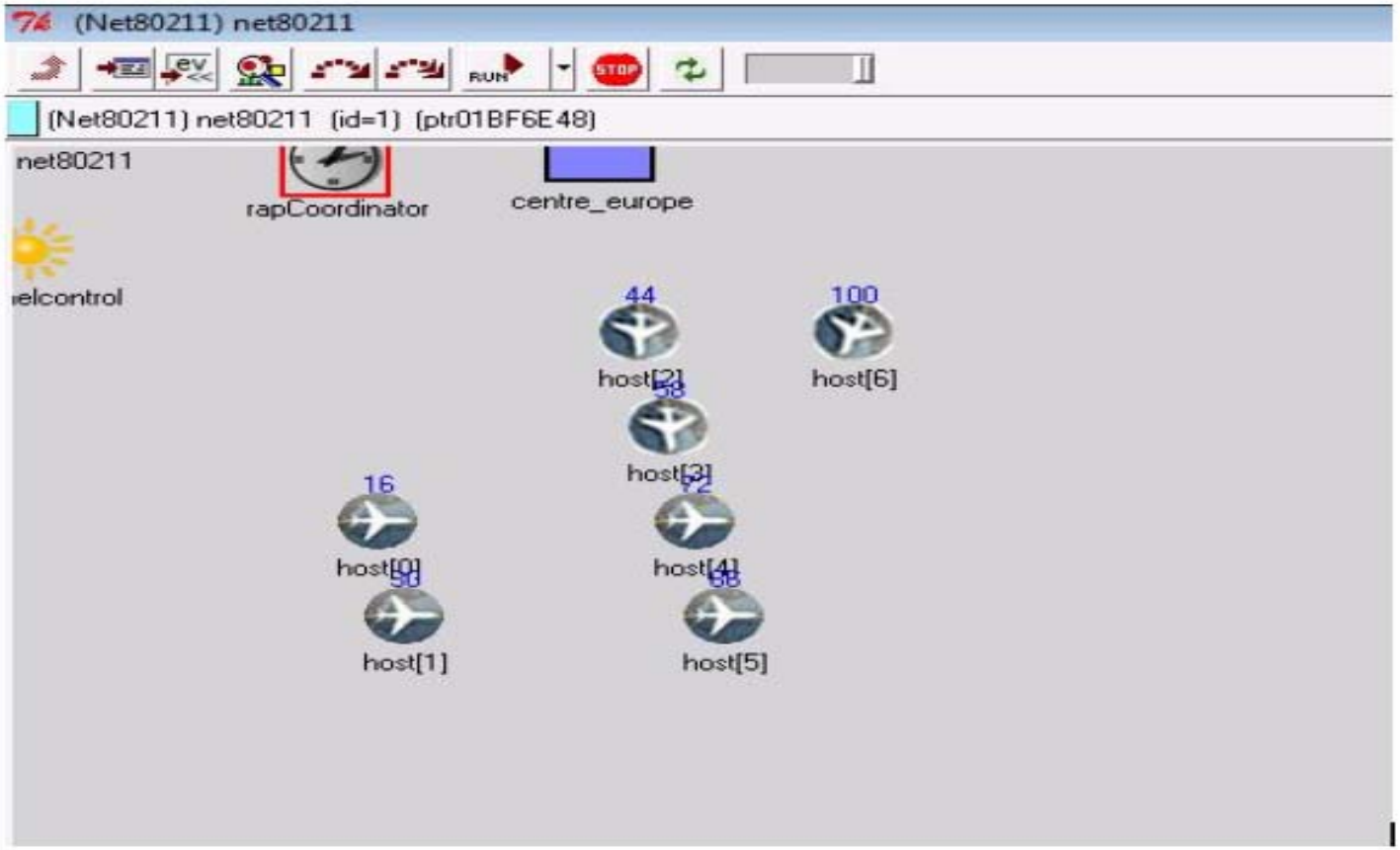
- Future generation of ATC: Airborne Self-Separation, an operating environment where pilots are allowed to select their flight paths in real-time.



ADS-B (AUTOMATIC DEPENDENT SURVELLEINCE BROADCAST): based on the Global Navigation Satellite System (*GNSS*) - broadcast communication links -

Airbone Self-Separation

- guarantee the correct behaviour of the system (i.e., the set of aircraft in a given area) even in the presence of component failures, or malicious attacks .



Airbone Self-Separation

Main challenge in Airborne Self-Separation:

coordination between aircraft within a dynamic environment, where the set of surrounding aircraft is constantly changing, and where there is the possibility of arbitrary failures and malicious threats.

Conflict Resolution (and Traffic Optimisation) problem

Conflict Resolution algorithms are

decentralized and cooperative with the cooperation between aircraft being based on "emergent" properties of the system

Conflict Resolution algorithms

based on theory of decision-making based on a multi-agent approach and Game Theory (SGT)

- SGT as decision procedure

- a fault-tolerant **Byzantine agreement protocol** that provides SGT the necessary services to execute correctly

- the agreement protocol is supported by suitable communication primitives realised for wireless networks

Conflict Resolution algorithms

System model:

multi-agent system

Aicraft: agent

Aircraft state (local information):
state_i = (aircraft id, destination, current_time,
coordinates, speed, ….)

Region:
surrounding aircrafts involved in the calculation
of the flight path (changing)

Decision procedure:

algorithm applied at every agent i based on

- agent state (local information)
- state of the agents in the region
(information received from sorrounding aicrafts)

Decision procedure

Every aircraft must decide its flight path: to execute correctly it is necessary that every agent has the correct view of the system

Let n be number of agents in a region,

Agent_i: decision_procedure(i, state_1, state_2, …, state_n)

every agent applies the decision procedure starting from the same information on the state of the aircrafts in the region

Assume an attacker changes information exchanged through wireless communications.

What happen if information on the value of the position of aircraft 2 is modified and arrives wrong at some destination?

Agent_i: decision_procedure(i, state_1, state_2, …, state_n)

…

Agent_j: decision_procedure(j, state_1, state*_2, …, state_n)

Correctness of the decision procedure is compromised

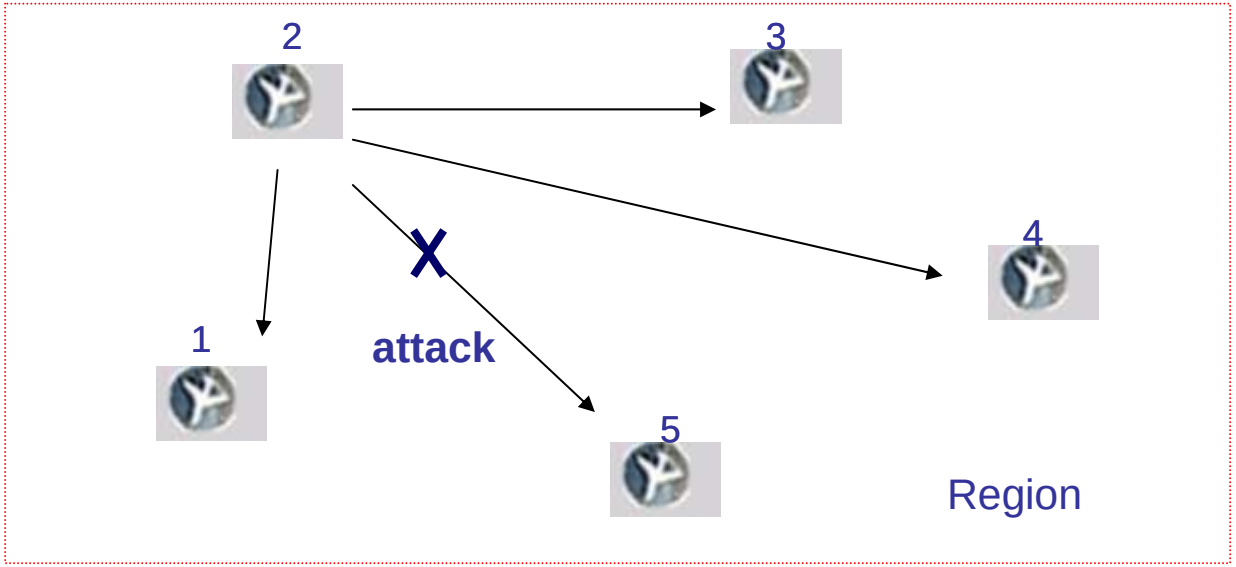
Agent_i and Agent_j must have the correct value of the state of the aircraft in the region before applying the decision procedure.

Consensus problem

The Consensus problem can be stated informally as:

How to make a set of distributed agents achieve agreement on a value despite a number of threats?

Example: value of the state of aircraft 2



Byzantine Agreement protocol

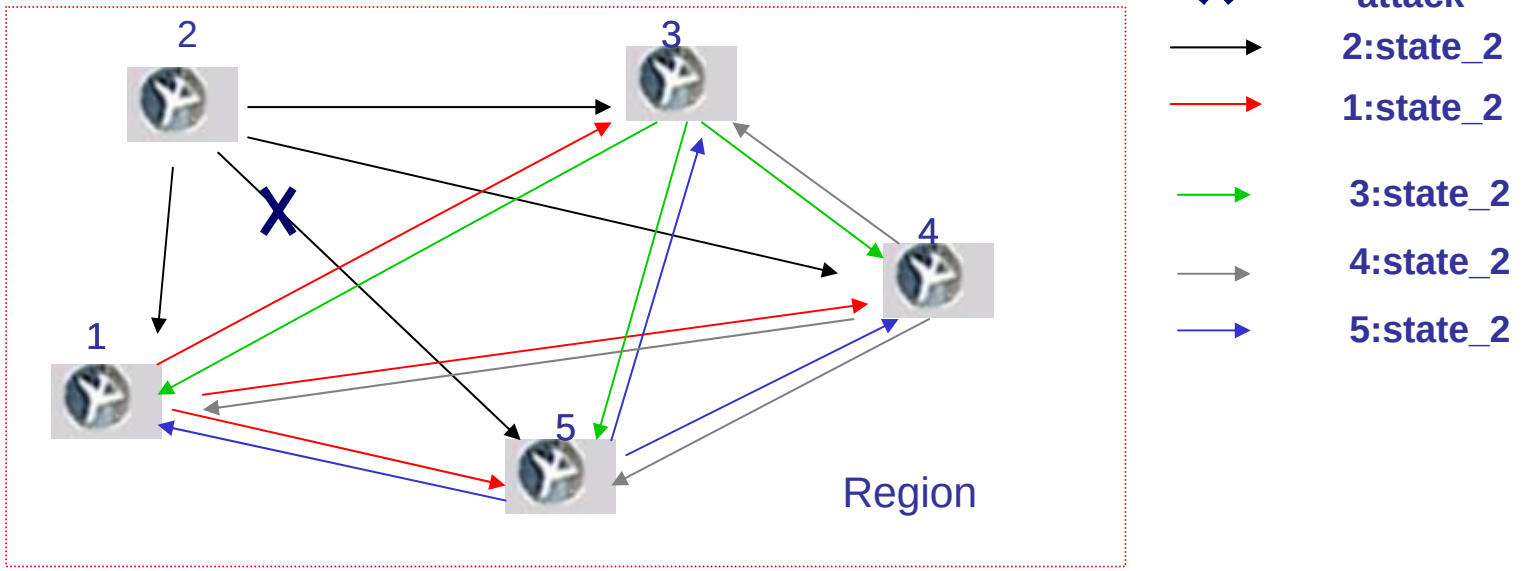
L. Lamport, R. Shostak, M. Pease, The Byzantine Generals Problem
ACM Trans. on Progr. Languages and Systems, 4(3),1982

Consensus problem

Idea:

- Aircrafts send messages back and forth among themselves reporting the position received by the other aircrafts

Example: value of the state of aircraft 2



- Each aircraft final decision on the state of other aircrafts obtained by: majority vote among the values received

Agent_1: majority(2:state_2, 3:state_2, 4:state_2, 5:state_2)

……

Agent_5: majority(1:state_2, 3:state_2, 4:state_2, 2:state_2)

Consensus problem

Let n be the number of agents partecipanting into the protocol and state_i be the state of agent i. Let the number of attacks be equal to one.

Let us consider one source agent and n-1 destination agents.

Destination agents must agree on the position of the source agent.

The protocol consists of different rounds

First round:

source agent i communicate its value to each other agent j

Second round:

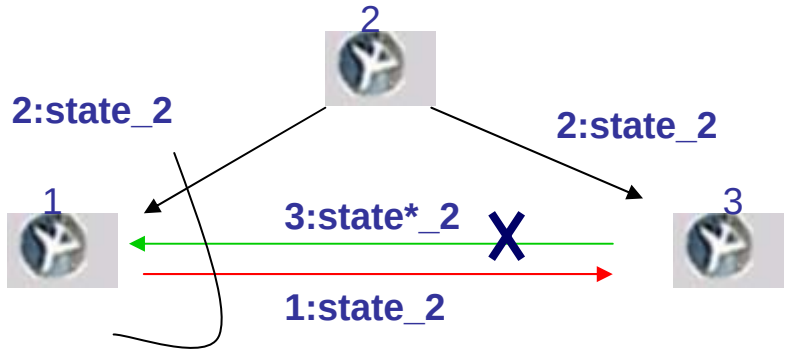
each destination agent send the received value to the other n-2 destination agents

Each destination agent j uses the value obtained by the majority function applied to the received values (n-1 values)

3 aicrafts -> no solution exists in presence of one attack

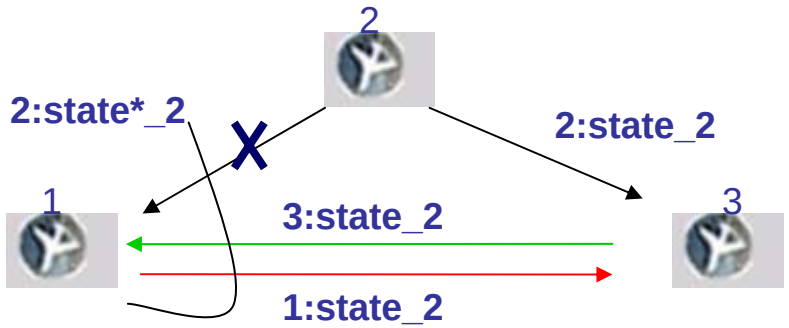
Two different positions, one directly from the aircraft and the other relayed from another aircraft.

CASE A



aircraft 1 does not distinguish between the two situations

CASE B



Assume one must use the value
- receiveid directly from the aircraft -> case A correct, case B wrong
- relayed by another aircraft -> case A wrong, case B correct

There not exists a majoirity among values

Consensus problem

Assumptions:

the system is synchronous (The absence of a message can be detected**)**
any two processors had direct communication across a network
the sender of a message can be identified by the receiver

Moreover, if a message does not arrive, we consider a default message equal to “no information”.

Similarly the function majority returns “no information” if there not exists a majoirity among values

The decision procedure uses the previous position of the aircraft, speed etc… to make assumptions on the new position of the aircraft and to assure system safety

What percentage of threats can the algorithm tolerate and still correctly determine a consensus on the correct information?

Byzantine fault-tolerant (BFT) algorithms

From the abstract of Castro & Liskov OSDI'99 paper:

> *"We believe that Byzantine fault-tolerant algorithms will be increasingly important in the future because malicious attacks and software errors are increasingly common and can cause faulty nodes to exhibit arbitrary behavior."*

Consensus in distributed systems with n processes
and at most m faulty processes under Bizantyne faults
hypothesis, no assumption of the behaviour of faulty processes)

The Oral messages algorithm *OM(m)* – where m is the number of faulty processes

Bizantyne Generals problem (values: attack/retreat)
(loyal/traitor generals)

Consensus:

IC1: All correct processes decides the same value for S
IC2: The value of the decision must agree with the value sent by the process S if he is correct.

Algorithm OM(m)

OM(0)

1. S sends its value to every Di, $i \in \{1, ..., n-1\}$
2. Each Di uses the received value, or a the default value "retreat" if no value is received

OM(m), m>0

1. S sends its value to every Di, $i \in \{1, ..., n-1\}$
2. Let vi be the value received by Di from S (vi =retreat if Di receives no value) Di acts as S in OM(m-1) to send vi to each of the n-2 other destination processes
3. For each i and $j \neq i$, let vj be the value that Di received from Dj in step 2 (vj = retreat if Di receives no value). Di uses the value of majority(v1, ..., vn-1)

OM(m) is a recursive algorithm that invokes n-1 separate executions of OM(m-1), each of which invokes n-2 executions of O(m-2), etc..

The Oral message algorithm (OM)

OM(m) solves the Byzantine Generals Problem for (3m+1) or more generals, in presence of at most m traitors

1 traitors, at least 4 generals
2 traitors, at least 7 generals
.....

OM(m) requires :

m+1 rounds
message size O(nm+1) - message size grows at each round

Original Byzantine Generals Problem
Solved assigning the role of source to every general, and running the algorithms concurrently

Agreement between multiple processes is a basic building block in fault-tolerant distributed computing

The ability of the traitor to lie makes the Byzantine Generals problem difficult:

- No assumptions on the characteristics of faulty processors

- Conflicting values are solved taking a deterministic majority vote on the values received at each processor (completely distributed).

→ restrict the ability of the traitor to lie

Consensus in asynchronous systems

Consensus: cannot be solved deterministically in an asynchronous distributed system that is subject even to a single crash failure [Fisher, Lynch and Paterson 85]

→ due to the difficulty of determining whether a process has actually crashed or is only very slow.

If no assumptions are made about the upper bound on how long a message can be in transit t, nor *the upper bound on the relative rates of processors* t', then a single processor running the consensus protocol could simply halt and delay the precedure indefinitely.

Stopping a single process at an inopportune time can cause any distributed protocol to fail to reach consensus

Techniques to circumvent the FLP impossibility result in asynchronous systems

Sacrificing Determinism

- using randomization to design probabilistic algorithms
- substitute one of the properties that define consensus by a similar property that is satisfied only with a certain probability

Adding Time to the Model

- *using the notion of* partial synchrony introduced by Dwork, Lynch and Stockmeyer in [Dwork et al., 1988].

Augmenting the System Model with an Oracle

Failure detectors

- *the idea is* to suspect the crash of a process. Each process has attached a failure detector module and the set of all these modules formed the failure detector.

Wormholes

- an extension to a system model with stronger properties than the rest of the system

The Consensus problem has been defined for a set of *n* known processes.

Consensus in large dynamic systems in which the number of involved processes is unknown [Mostefaoui et al., 2005, Aguilera, 2004] is an open area of research. Consensus is still not defined in this context.

Conclusions

Intrusion tolerance a new paradigm for computer system security

- intrusion tolerant protocols (I/T protocols) and intrusion tolerant systems (I/T systems) have been developed

The main motivation has been the poor state of security in Internet

Intrusion tolerance is usually obtained by replicating the system in a set of servers, which behave accordingly to the system specification even if there are intrusions in up to a certain threshold of the servers

- Each server protected using the current best practices
- Diversity between the servers in such a way that they do not share the same vulnerability, the overall system is ensured to be more thrustworthy than if it was centralised
- small thrusted components

Workshop on Recent Advances in Intrusion-Tolerant Systems
WRAITS 2011 (In conjunction with The 41th IEEE/IFIP International Conference on Dependable Systems and Networks - DSN 2011)
http://wraits11.di.fc.ul.pt/