Fault Tolerant Software

Prof. Raul Ceretta Nunes
UFSM

Introduction

- HW is very reliable and its reliability continues to improve with time
- SW is not so reliable
- Making a system FT to faults in software is a desirable goal
- Software faults is always design faults
- FT software

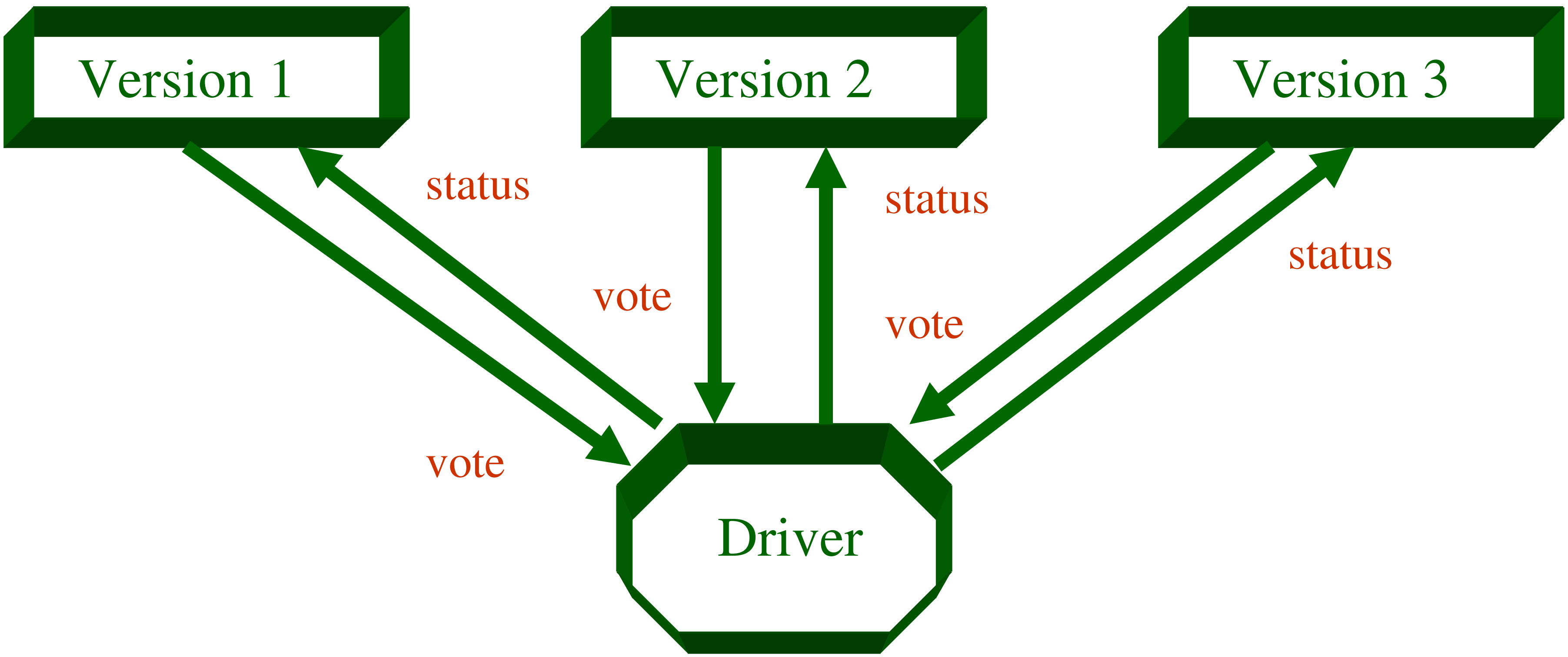
Fault Tolerance Software

- Used for detecting design errors
- Static — N-Version programming
- Dynamic
  - Detection and Recovery
  - Recovery blocks: backward error recovery
  - Exceptions: forward error recovery

N-Version Programming

- Design diversity
- The independent generation of N (N > 2) functionally equivalent programs from the same initial specification
- No interactions between groups
- The programs execute concurrently with the same inputs and their results are compared by a driver process
- The results (VOTES) should be identical (considering the consensus result)

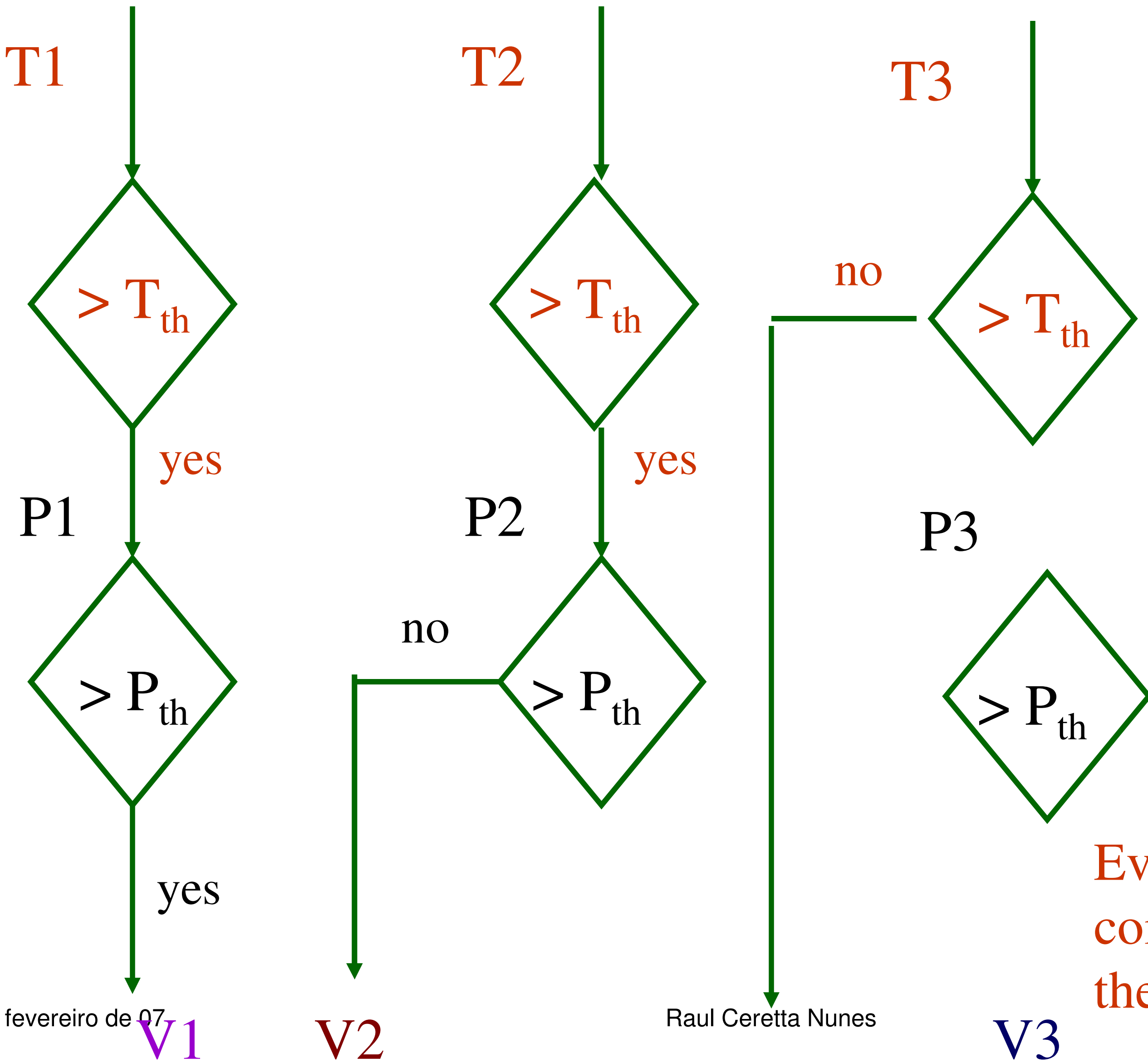
N-Version Programming



Vote Comparison

- To what extent can votes be compared?
- Text or integer arithmetic will produce identical results
- Real numbers => different values
- Need inexact voting techniques

Consistent Comparison Problem



Each version will produce a different but correct result

Even if use inexact comparison techniques, the problem occurs

N-version programming depends on

- Initial specification — The majority of software faults stem from inadequate specification? A specification error will manifest itself in all N versions of the implementation
- Independence of effort — Experiments produce conflicting results. Where part of a specification is complex, this leads to a lack of understanding of the requirements. If these requirements also refer to rarely occurring input data, common design errors may not be caught during system testing
- Adequate budget — The predominant cost is software. A 3-version system will triple the budget requirement and cause problems of maintenance. Would a more reliable system be produced if the resources potentially available for constructing an N-versions were instead used to produce a single version?

military versus civil avionics industry

Software Dynamic Redundancy

It is organized on four phases:

- error detection — no fault tolerance scheme can be utilised until the associated error is detected
- damage confinement and assessment — to what extent has the system been corrupted? The delay between a fault occurring and the detection of the error means erroneous information could have spread throughout the system
- error recovery — techniques should aim to transform the corrupted system into a state from which it can continue its normal operation (perhaps with degraded functionality)
- fault treatment and continued service — an error is a symptom of a fault; although damage repaired, the fault may still exist

Error Detection

- Environmental detection
  - hardware — e.g. illegal instruction
  - O.S/RTS — null pointer
- Application detection
  - Replication checks
  - Timing checks
  - Reversal checks
  - Coding checks
  - Reasonableness checks
  - Structural checks
  - Dynamic reasonableness check

Damage Confinement and Assessment

- Damage assessment is closely related to damage confinement techniques used

- Damage confinement is concerned with structuring the system so as to minimise the damage caused by a faulty component (also known as firewalling)

- Modular decomposition provides static damage confinement; allows data to flow through well-define pathways

- Atomic actions provides dynamic damage confinement; they are used to move the system from one consistent state to another

Error Recovery

- Probably the most important phase of any fault-tolerance technique
- Two approaches: forward and backward
- Forward error recovery continues from an erroneous state by making selective corrections to the system state
- This includes making safe the controlled environment which may be hazardous or damaged because of the failure
- It is system specific and depends on accurate predictions of the location and cause of errors (i.e, damage assessment)
- Examples: redundant pointers in data structures and the use of self-correcting codes such as Hamming Codes

Backward Error Recovery (BER)

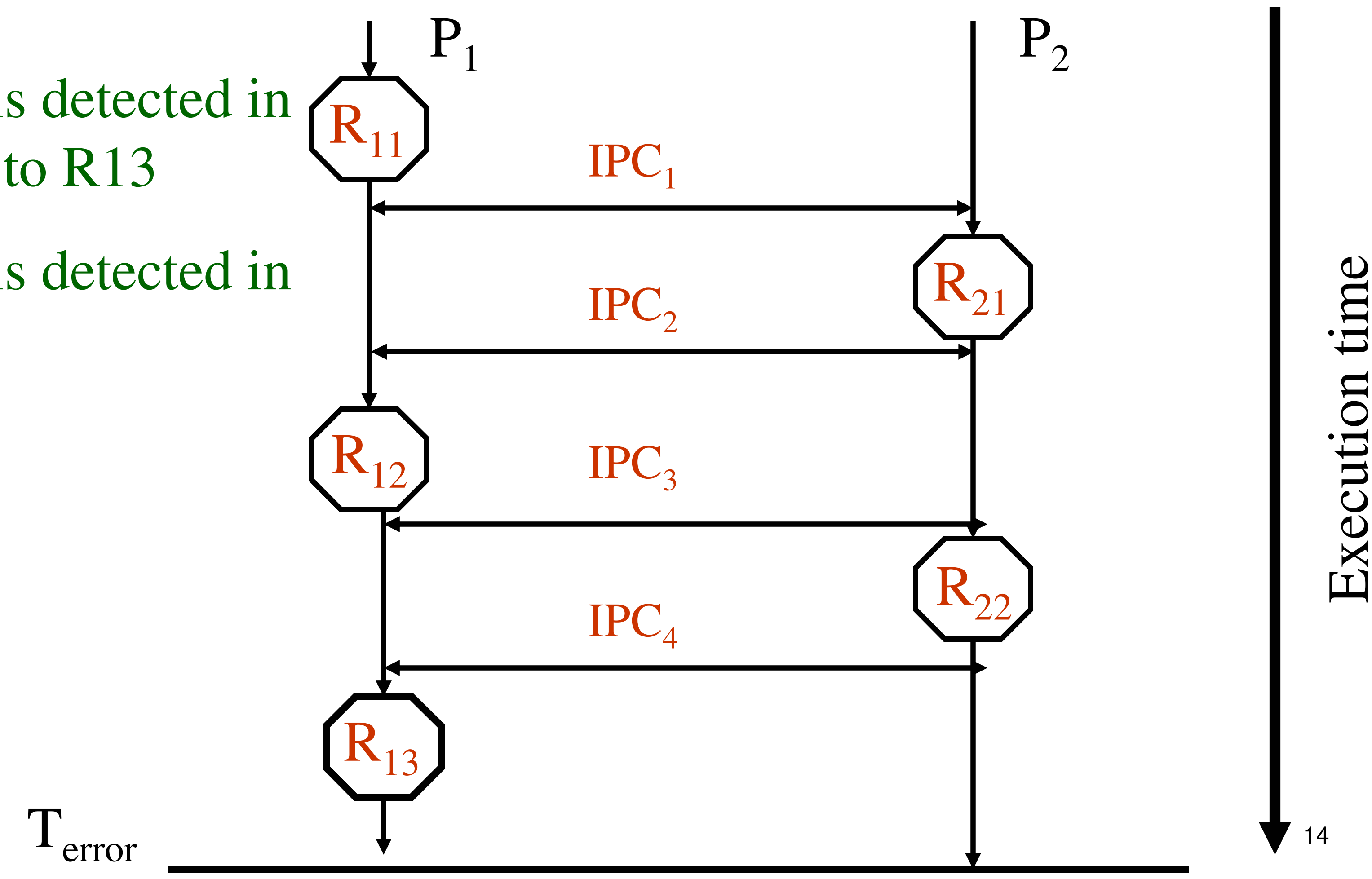
- BER relies on restoring the system to a previous safe state and executing an alternative section of the program
- This has the same functionality but uses a different algorithm (c.f. N-Version Programming) and therefore no fault
- The point to which a process is restored is called a recovery point and the act of establishing it is termed checkpointing (saving appropriate system state)
- Advantage: the erroneous state is cleared and it does not rely on finding the location or cause of the fault
- BER can, therefore, be used to recover from unanticipated faults including design errors
- Disadvantage: it cannot undo errors in the environment!

The Domino Effect

- With concurrent processes that interact with each other, BER is more complex. Consider:

If the error is detected in P1 rollback to R13

If the error is detected in P2 ?



Fault Treatment and Continued Service

- ER returned the system to an error-free state; however, the error may recur; the final phase of F.T. is to eradicate the fault from the system
- The automatic treatment of faults is difficult and system specific
- Some systems assume all faults are transient; others that error recovery techniques can cope with recurring faults
- Fault treatment can be divided into 2 stages: fault location and system repair
- Error detection techniques can help to trace the fault to a component. For, hardware the component can be replaced
- A software fault can be removed in a new version of the code
- In non-stop applications it will be necessary to modify the program while it is executing!

The Recovery Block approach to FT

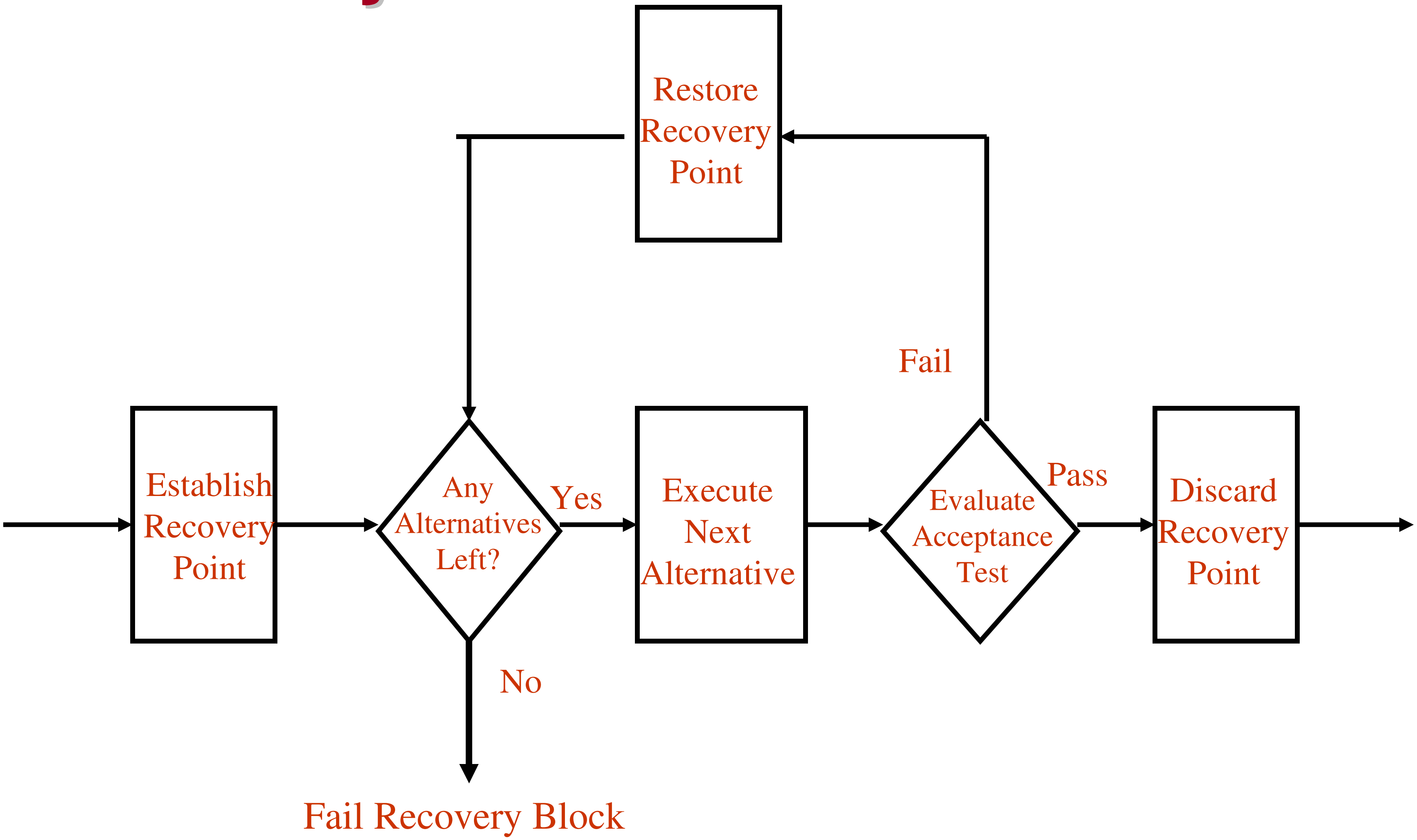
- Language support for BER
- At the entrance to a block is an automatic recovery point and at the exit an acceptance test
- The acceptance test is used to test that the system is in an acceptable state after the block's execution (primary module)
- If the acceptance test fails, the program is restored to the recovery point at the beginning of the block and an alternative module is executed
- If the alternative module also fails the acceptance test, the program is restored to the recovery point and yet another module is executed, and so on
- If all modules fail then the block fails and recovery must take place at a higher level

Recovery Block Syntax

```
ensure <acceptance test>
by
   <primary module>
else by
   <alternative module>
else by
   <alternative module>
   ...
else by
   <alternative module>
else error
```

- Recovery blocks can be nested
- If all alternatives in a nested recovery block fail the acceptance test, the outer level recovery point will be restored and an alternative module to that block executed

Recovery Block Mechanism



Example: Solution to Differential Equation

```
ensure Rounding_err_has_acceptable_tolerance
by
  Explicit Kutta Method
else by
  Implicit Kutta Method
else error
```

- Explicit Kutta Method fast but inaccurate when equations are stiff
- Implicit Kutta Method more expensive but can deal with stiff equations
- The above will cope with all equations
- It will also potentially tolerate design errors in the Explicit Kutta Method if the acceptance test is flexible enough

Nested Recovery Blocks

```
ensure rounding_err_has_acceptable_tolerance
by
  ensure sensible_value
  by
    Explicit Kutta Method
  else by
    Predictor-Corrector K-step Method
  else error
else by
  ensure sensible_value
  by
    Implicit Kutta Method
  else by
    Variable Order K-Step Method
  else error
else error
```

The Acceptance Test

- The acceptance test provides the error detection mechanism which enables the redundancy in the system to be exploited
- The design of the acceptance test is crucial to the efficacy of the RB scheme
- There is a trade-off between providing comprehensive acceptance tests and keeping overhead to a minimum, so that fault-free execution is not affected
- Note that the term used is acceptance not correctness; this allows a component to provide a degraded service
- All the previously discussed error detection techniques discussed can be used to form the acceptance tests
- However, care must be taken as a faulty acceptance test may lead to residual errors going undetected

N-Version Programming vs Recovery Blocks

- Static (NV) versus dynamic redundancy (RB)
- Design overheads — both require alternative algorithms, NV requires driver, RB requires acceptance test
- Runtime overheads — NV requires N * resources, RB requires establishing recovery points
- Diversity of design — both susceptible to errors in requirements
- Error detection — vote comparison (NV) versus acceptance test (RB)
- Atomicity — NV vote before it outputs to the environment, RB must be structure to only output following the passing of an acceptance test

Dynamic Redundancy and Exceptions

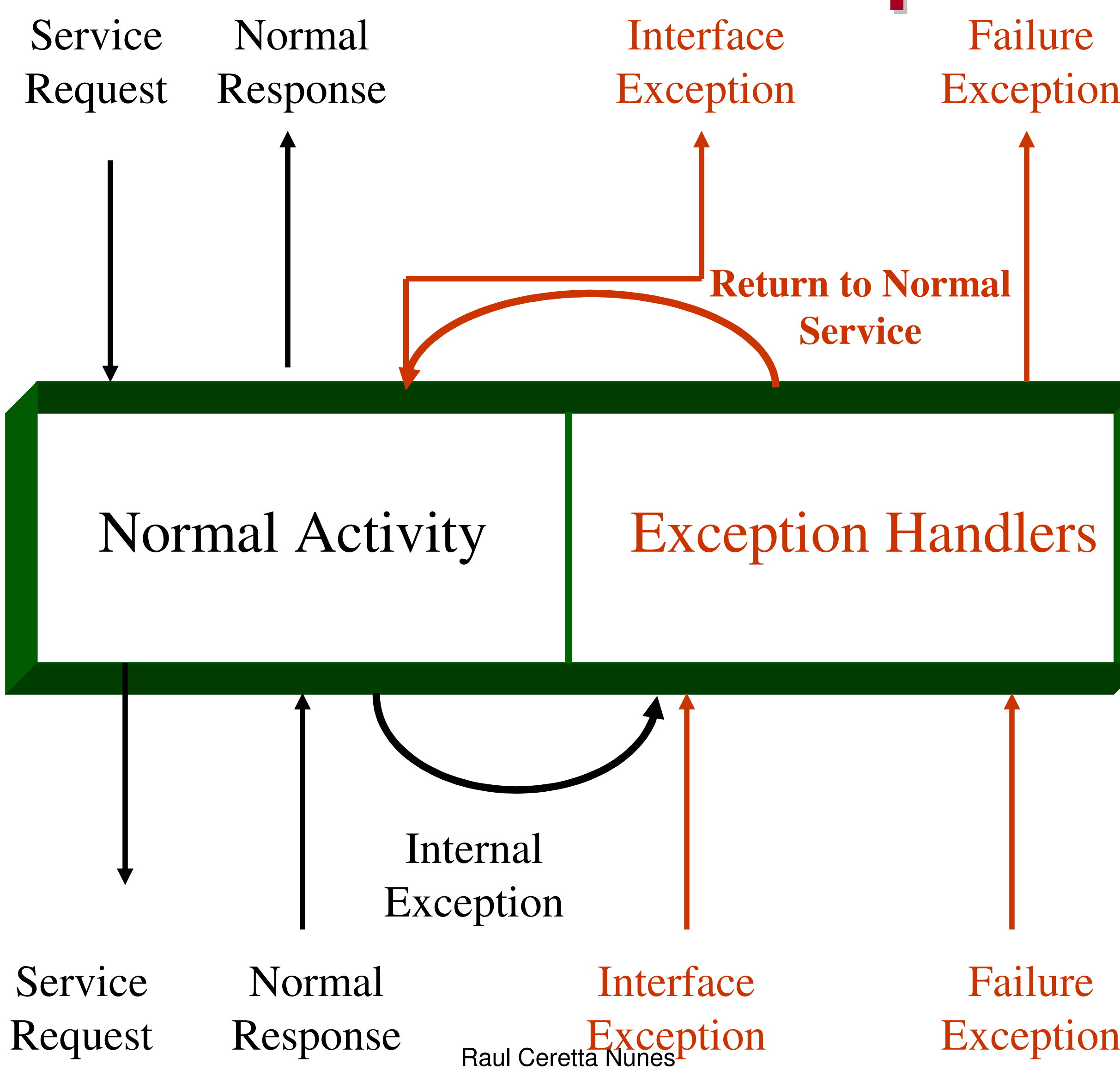
- An exception can be defined as the occurrence of an error
- Bringing an exception to the attention of the invoker of the operation which caused the exception, is called raising (or signally or throwing) the exception
- The invoker's response is called handling (or catching) the exception
- Exception handling is a forward error recovery mechanism, as there is no roll back to a previous state; instead control is passed to the handler so that recovery procedures can be initiated
- However, the exception handling facility can be used to provide backward error recovery

Exceptions

Exception handling can be used to:

- cope with abnormal conditions arising in the environment
- enable program design faults to be tolerated
- provide a general-purpose error-detection and recovery facility

Ideal Fault-Tolerant Component



Summary

- N-version programming: the independent generation of N (where N >= 2) functionally equivalent programs from the same initial specification
- Based on the assumptions that a program can be completely, consistently and unambiguously specified, and that programs which have been developed independently will fail independently
- Dynamic redundancy: error detection, damage confinement and assessment, error recovery, and fault treatment and continued service

Summary

- With backward error recovery, it is necessary for communicating processes to reach consistent recovery points to avoid the domino effect
- For sequential systems, the recovery block is an appropriate language concept for BER
- Although forward error recovery is system specific, exception handling has been identified as an appropriate framework for its implementation
- The concept of an ideal fault tolerant component was introduced which used exceptions

1. Other Uniprocess Approaches

- Deadline Mechanism
- Distributed Recovery Block
- Data Diversity

1.1. Deadline Mechanism

```
service service-name
within response-period
by
  primary algorithm
else by
 alternate algorithm
end
```

- Based on recovery-block mechanism
- slack time = response-time – maximum execution time of the alternate algorithm
- Used on real-time systems to avoid timing failures

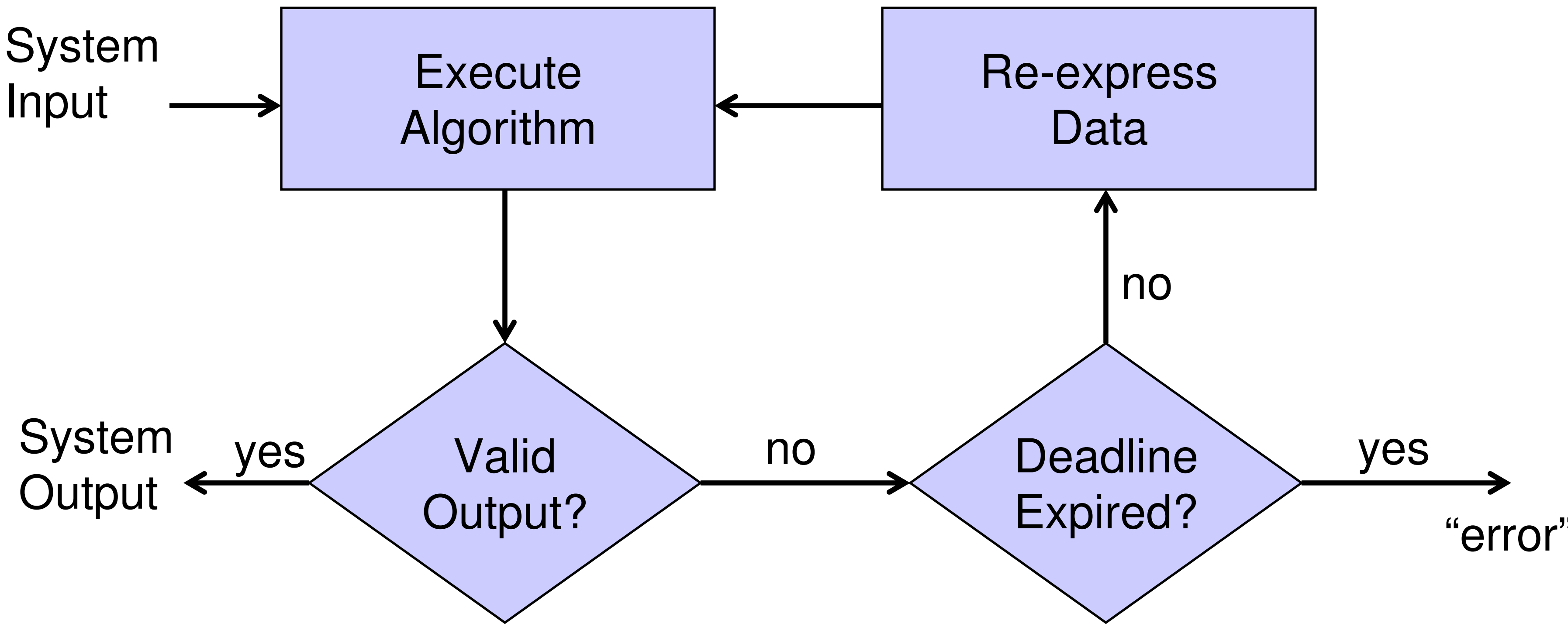
1.2. Distributed Recovery Block - DRB

- Meant to avoid transient hardware errors, because acceptance-test does not indicate the cause of the error
- Key idea: to distribute RB and AT on different nodes and execute them concurrently.
- If the primary fails, it sends a notice to the backup node, that forward its result. The primary erroneous state can also be trigged by backup from a watchdog timer.
- If the primary succeed, it also sends a notice to the backup, that does not forward its result.

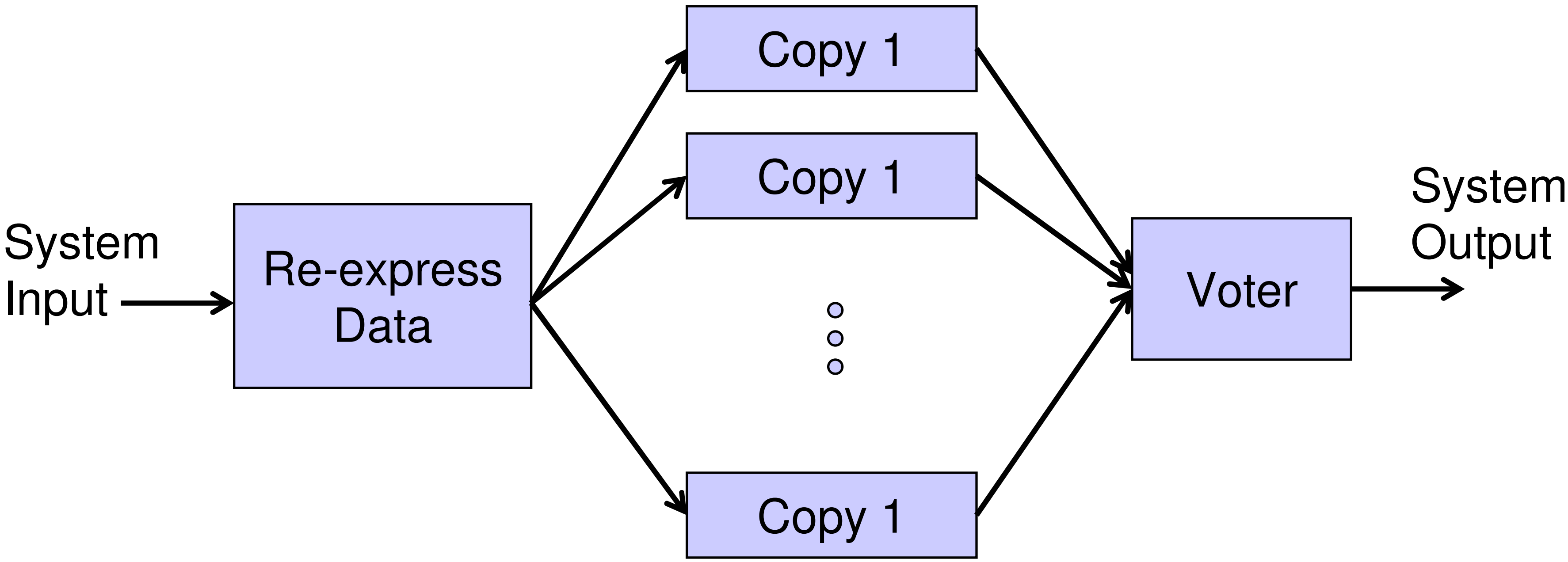
1.3. Data Diversity

- Meant as a less expensive alternative to design diversity
- Depends on the *data re-expression*, a generation of logically equivalent data sets
- Two structures:
  - Retry block
  - N-copy programming

1.3.1. Retry block



1.3.2. N-copy programming



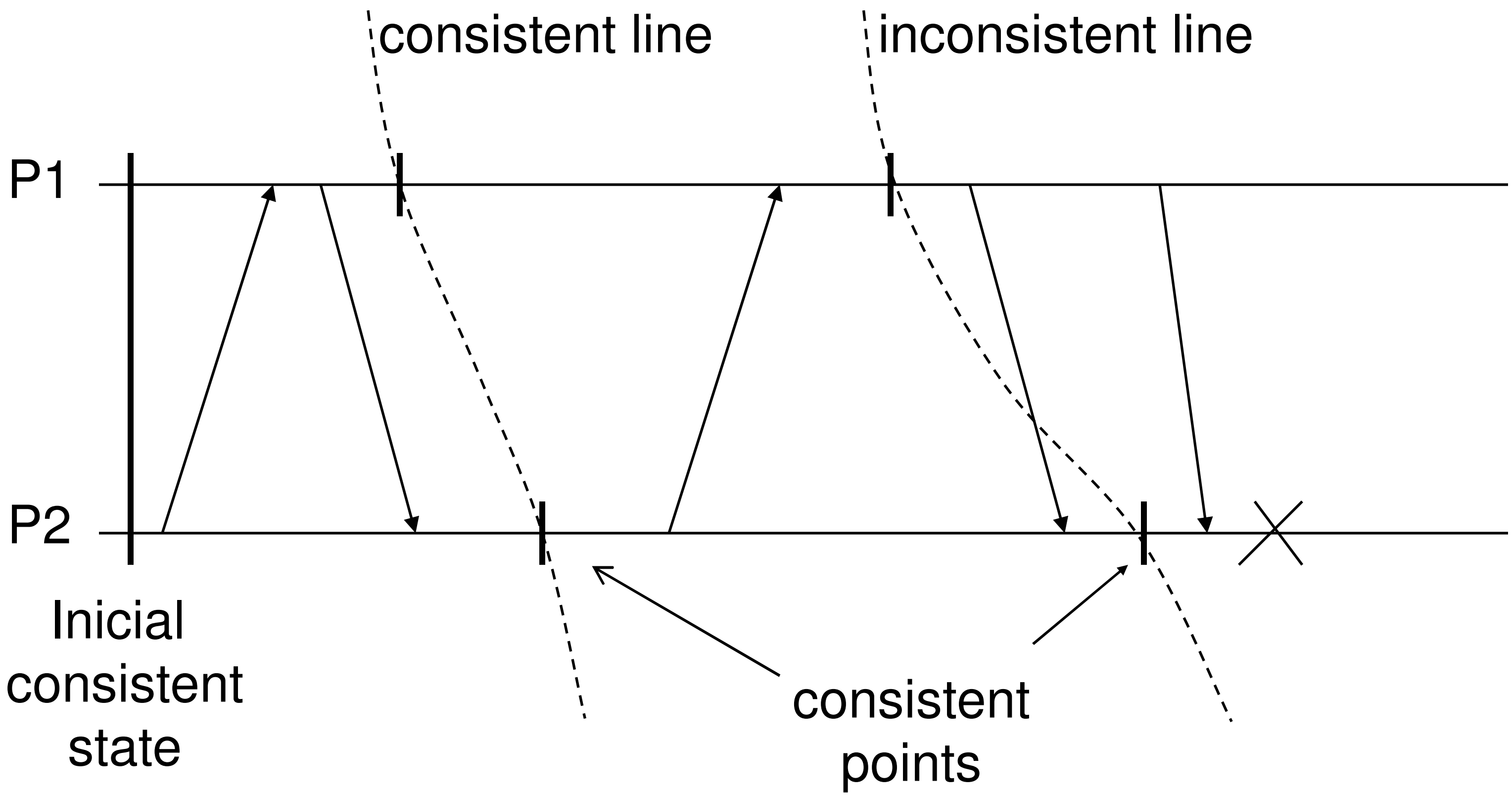
2. Backward Recovery in Concurrent Systems

- Simple method to support design faults in a concurrent system
  - Reset the process to some previous consistent state and reexecute it (it may not fail because on new time execution the environment is different)
  - Only error detection capabilities are required
  - This approach works to transient faults

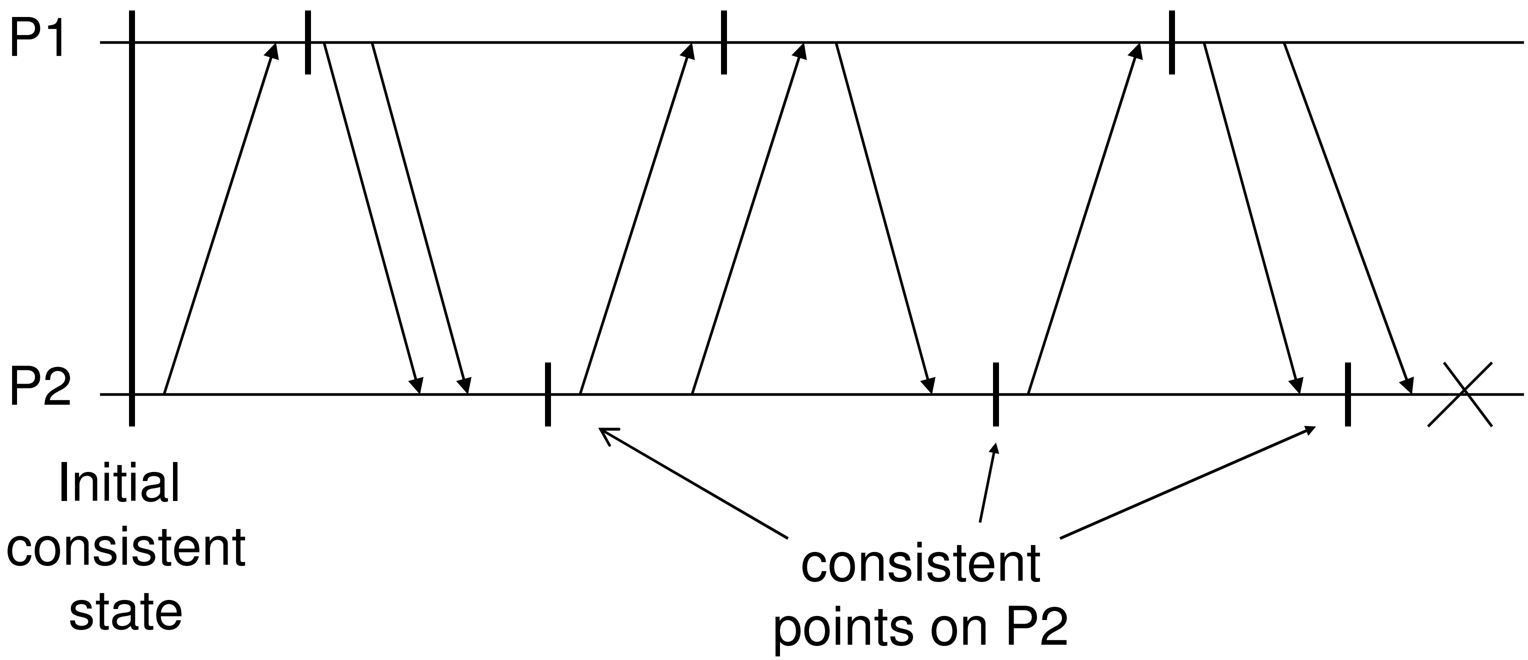
2.1 Domino Effect

- Rollback is employed in recovery blocks for error recovery
- A forced rollback may be needed on rolling back a process in a concurrent system
- Uncontrolled rollbacks may cause a domino effect, i.e., rollback to first consistent point

A consistent and an Inconsistent recovery line



The domino effect



Domino effect reason: recovery points on different processes are not coordinated with communication commands.

2.2. Conversations

- A language construct.
- It prevents domino effect.
- In a conversation a process can only communicates with another process in the same conversation.
- If any process fails an acceptance test or otherwise detects na exception, every process in the conversation performs a rollback to its recovery point, established on entry to the conversation, and uses an alternate algorithm.
- The set of processes taking part in a conversation are fixed.

2.3. FT-Action

- An atomic action
- A planned atomic action – an one that is planned during design and supported by some run-time mechanism
- A basic atomic action (indivisible)
  - A recoverable atomic action (indivisible and recoverable, ou all-or-nothing) is not suitable
- An atomic action where different recovery techniques, like exception handling, could be used
- An atomic action used on conversation or recovery block to provide recovery and FT

Conversations using monitors

Using Distributed FT-Action

3. Forward Recovery in Concurrent Systems

3.1 Exception Resolution

Exception Handling with FT-Action