A case study: SIFT

From: D.P. Siewiorek, R. S. Swarz
Reliable Computer Systems (Design and Evaluation) Prentice Hall, 1998.
Chapter 10 – "The SIFT Case: Design and Analysis of a Fault Tolerant Computer for Aircraft Control".

SIFT

SIFT (Software Implemented Fault Tolerance)
is a Fault-Tolerant Computer for Aircraft Control

"a system capable of carrying out the calculations required for the control of an advanced commercial transport aircraft"

developed for NASA as an experimental case study for fault tolerant system research

The safety of the flight depends on the computer functions (controls derived from computer outputs).

Reliability requirement:
probability of failure less than $\mathbf{10^{-9}}$ per hour in a flight of ten hours' duration.

Reliability requirement similar to that demanded for manned space-flight systems.

SRI International (founded as Stanford Research Institute): responsable of the overall design, the software and the testing

Bendix Corporation: reponsable for the design and the construction of the hardware.

SIFT was delivered to NASA's Avionics Integration Research Laboratory in April 1982 (1978-1982).

SIFT

- A major objective of the SIFT design was to reduce the hardware failure rate by implementing as much of the system as possible in software (i.e., keeping the hardware component count to a minimum).

- This software-intensive design philosophy deliberately sacrificed performance to maximize reliability. It was implicitly assumed that failure due to software error would be eliminated by **formal proof of correctness**.

The SIFT effort began with broad, in-depth, studies stating the reliability and processing requirements for digital computers which would control flight-critical functions.

Detailed design studies were made of fault-tolerant architectures that could meet reliability and processing requirements.

SIFT

- Fully distributed configuration of processors
- Transparent fault tolerance (hw and sw replication and voting)
- Assignment of tasks to processors predetermined by a task schedule table defined by the designer
- As processors fail, the available hardware changes (reconfiguration), and a new task schedule is defined
- Processor synchronization fundamental to the correct functioning
- The decision to reconfigure is based on error information obtained when replicated data are voted

SIFT

Fault tolerance includes:

- error detection and correction,
- diagnosis,
- reconfiguration, and
- prevention of a faulty unit from having an adverse effect on the system

Use of a Consensus algorithm to tolerate “malicious processes” failure modes.

System overview

Main processors:

Computing is carried out by the main processors. Each processor's results are stored in a main memory that is uniquely associated with the processor.

A processor and its memory are connected by a conventional high bandwidth connection.

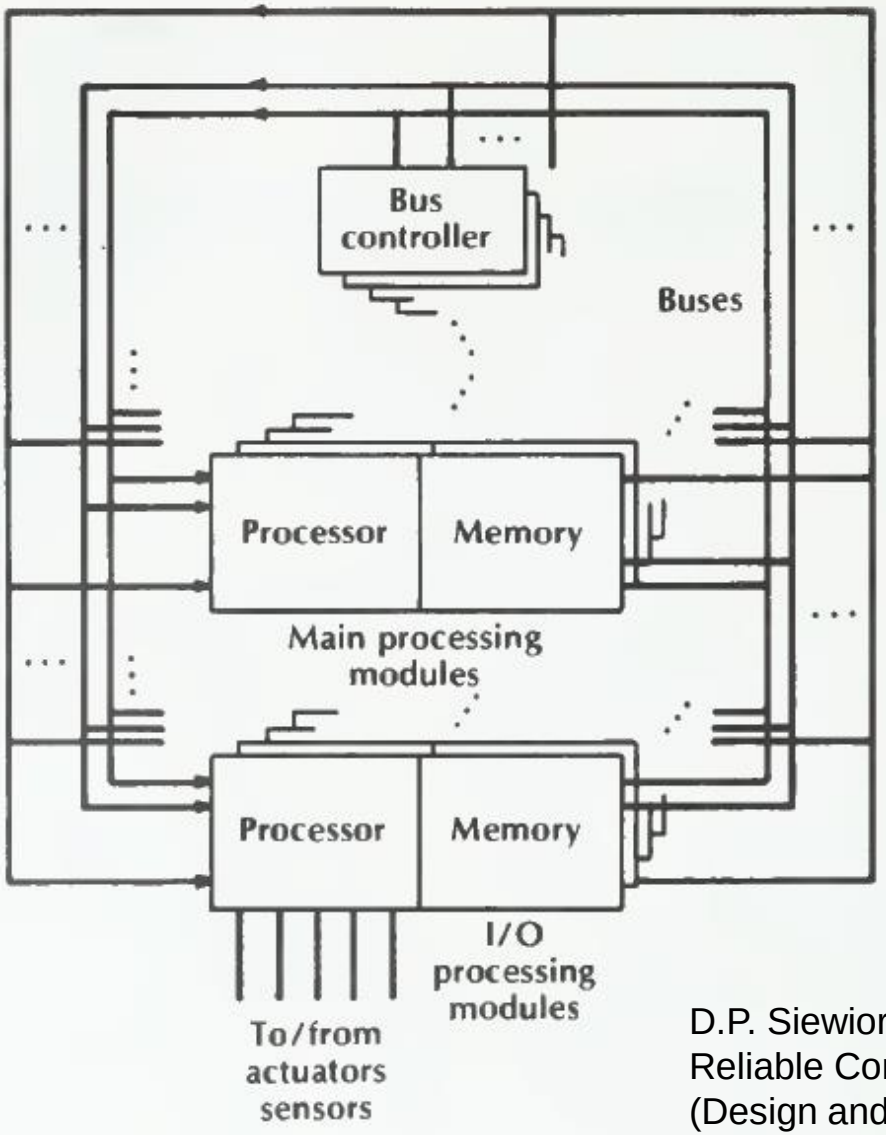
I/O processors:

The I/O processors and memories are structurally similar to the main processors and memories but are of much smaller computational and memory capacity. They connect to the input and output units of the system which are the sensors and actuators of the aircraft.

Processing module:

Each processor and its associated memory form a **processing module**, and each of the modules is connected to a **multiple bus system**.

Structure of SIFT hardware



multiple bus system connect processing modules

D.P. Siewiorek, R. S. Swarz
Reliable Computer Systems
(Design and Evaluation) Prentice Hall, 1998.
Chapter 10 – "The SIFT Case: Design and Analysis of a Fault Tolerant Computer for Aircraft Control".

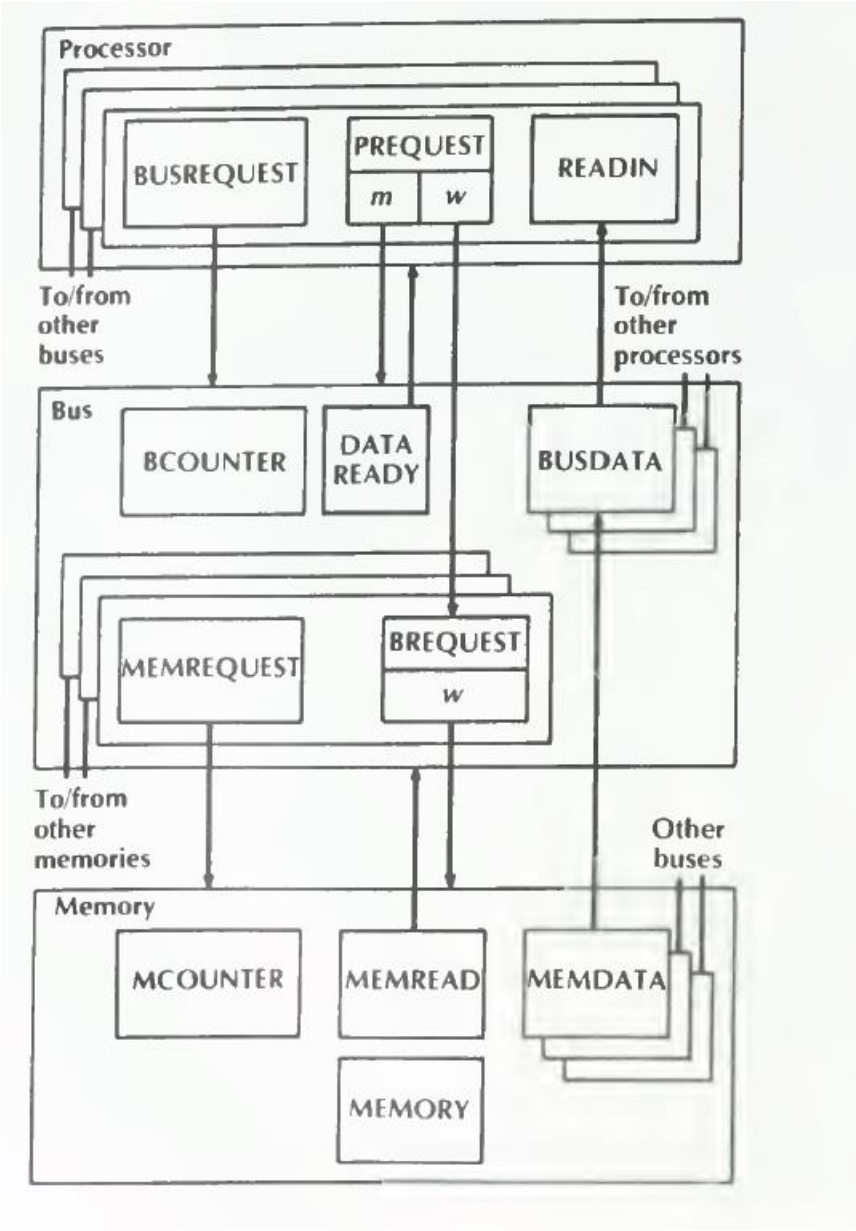
An abstract view of data transfer

Connections among processors, buses, and memories.

Within each unit are shown a number of abstract registers that contain data or control information.

Arrows that terminate at a register indicate the flow of data to the register.

Arrows that terminate at the boundary of a unit indicate control signals for that unit.



Assumption:

- a processor cannot write the memory of another processor
- a processor receive a control signal when data are available at the bus interface

D.P. Siewiorek, R. S. Swarz Reliable Computer Systems (Design and and Evaluation) Prentice Hall, 1998. Chapter 10 – "The SIFT Case: Design and Analysisof a Fault Tolerant Computer for Aircraft Control".

System overview: execution of tasks

The SIFT system executes a set of tasks, each of which consists of a **sequence of iterations**.

The **input data to each iteration of a task are the output data produced by the previous iteration of some collection of tasks** (which may include the task itself).

The input and output of the entire system is accomplished by tasks executed in the I/O processors.

Reliability is achieved by **replication + voting**:

each iteration of a task independently executed by a number of modules

System overview: loose synchronization

1) **voting is executed only at the beginning of each iteration**

   SIFT uses the **iterative nature of the tasks** to economize on the amount of voting

2) **processors need be only loosely synchronized**

   we must ensure only that the **different processors allocated to a task are executing the same iteration,** we do not need tight synchronization to the instruction or clock level.

An important benefit of this **loose synchronization** is that an iteration of a task can be scheduled for execution at slightly different times by different processors.

From the point of view of faillure:
simultaneous transient failures of several processors will be less likely to produce correlated failures in the replicated versions of a task.

System overview: task replication

The number of processors executing a task

1) can vary with the task
   a non critical task may be simplex;
   critical tasks may be replicated (3 or 5 replicas)

2) can be different for the same task at different times—for example, if a task that is not critical at one time becomes critical at another time.

The software system

The software of SIFT consists of the **application software** and the **executive software**.

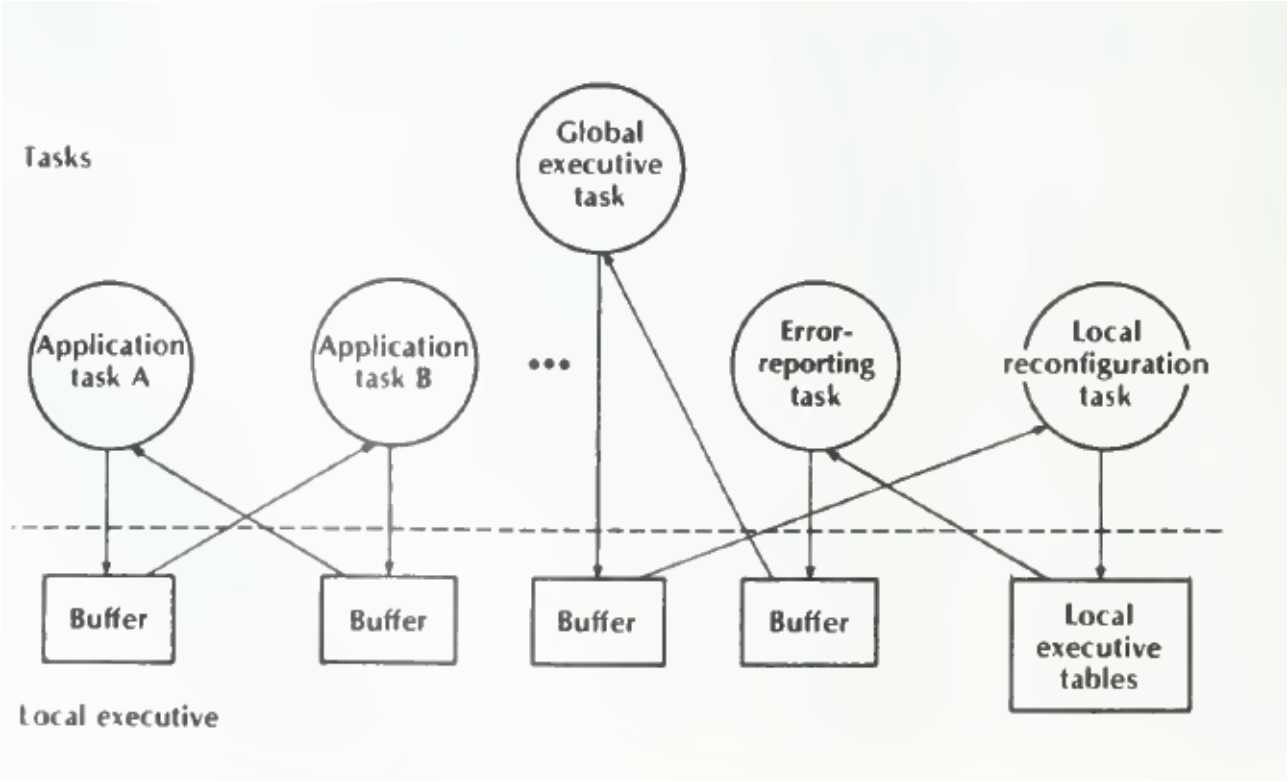
The **application software** performs the actual flight-control computations.

The **executive software** is responsible for the reliable execution of the application tasks and implements the error-detection and reconfiguration mechanisms.

Formal specifications of the executive software have been written in a rigorous form using the SPECIAL language [Robinson and Roubine, 1977] developed at SRI.

These formal specifications are needed for the proof of the correctness of the system. Moreover, they are also intended to force the designer to produce a well-structured system.

Logical structure of the SIFT software system



D.P. Siewiorek, R. S. Swarz Reliable Computer Systems (Design and and Evaluation) Prentice Hall, 1998. Chapter 10 – "The SIFT Case: Design and Analysisof a Fault Tolerant Computer for Aircraft Control".

From the point of view of the software, a **processing module, with its processor, memory, and associated registers, is a single logical unit.**

Application software

The application software is structured as a set of iterative tasks.

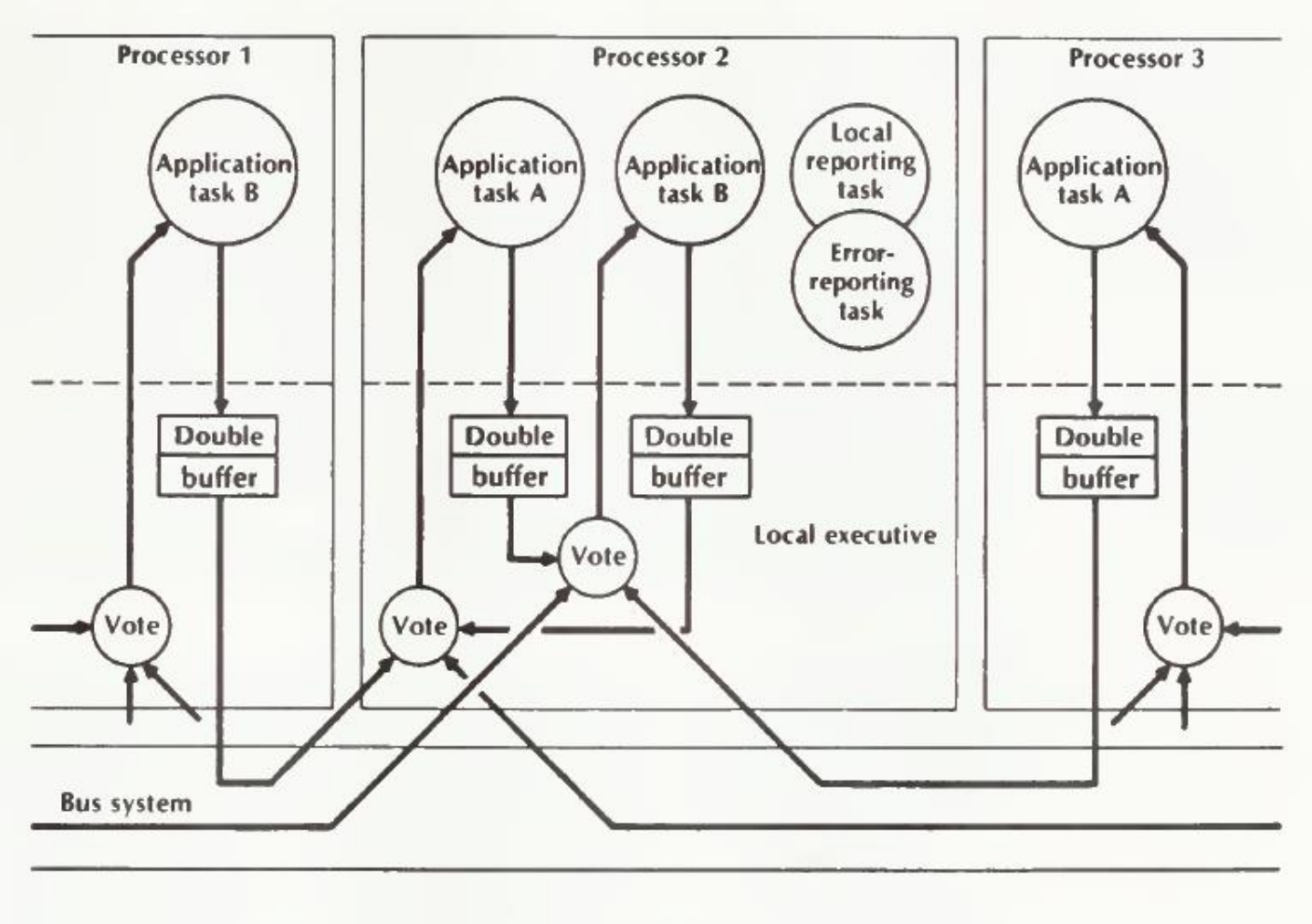
A task is executed by several processors; this fact is invisible to the application software.

In each iteration, an application task obtains its inputs by executing calls to the executive software.

After computing its outputs, it makes them available as inputs to the next iteration of tasks by executing calls to the executive software.

The input and output of a task iteration will consist of at most a few words of data.

Arrangement of application tasks within SIFT configuration



critical tasks are replicated on several processors

D.P. Siewiorek, R. S. Swarz Reliable Computer Systems (Design and and Evaluation) Prentice Hall, 1998. Chapter 10 – "The SIFT Case: Design and Analysisof a Fault Tolerant Computer for Aircraft Control".

Executive software

Performs the following functions:

1. Run each task at the required iteration rate.
2. Provide correct input values for each iteration of a critical task (masking any errors).
3. Detect errors and diagnose their cause.
4. Reconfigure the system to avoid the use of failed components.

To perform the last three functions, the executive software implements the techniques of redundant execution and majority voting.

The executive software is structured into three parts: **the global executive task, the local executive, and the local-global communicating tasks**.

Executive software

Global executive software:

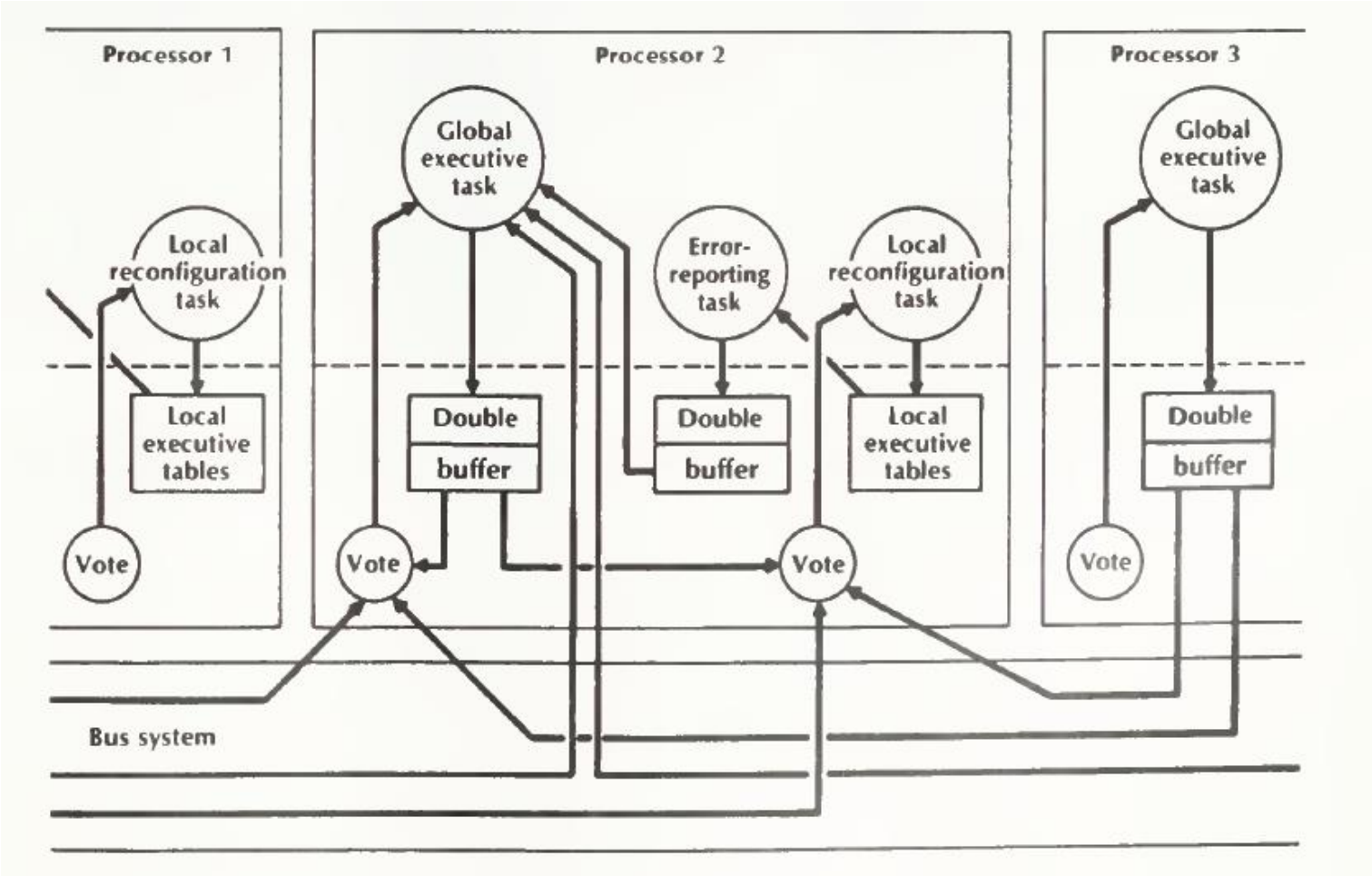
one global executive task is provided for the whole system.
It is run just like a highly critical application task, being executed by several processors and **using majority voting** to obtain the output of each iteration. It diagnoses errors to decide which units have failed and determines the appropriate allocation of tasks to processors.

Local executive software:

Each processing module has its own local executive and local-global communicating tasks.

The local-global communicating tasks are the error-reporting task and the local reconfiguration task (one for each processing module).

Arrangement of executive software within SIFT configuration



D.P. Siewiorek, R. S. Swarz Reliable Computer Systems (Design and and Evaluation) Prentice Hall, 1998. Chapter 10 – "The SIFT Case: Design and Analysisof a Fault Tolerant Computer for Aircraft Control".

The Global Executive Task: reconfiguration

The global executive task uses the results of every processor's error task to determine which processing modules and buses are faulty.

When the global executive decides that a component has failed, it initiates a reconfiguration by sending the appropriate information to the local reconfiguration task of each processor.

The global executive may also reconfigure the system as a result of directives from the application tasks. For example, an application task may report a change of flight phase that changes the criticality of various tasks.

The Local Executive.

The local executive is a collection of routines to perform the following functions:

(1) run each task allocated to it at the task's specified iteration rate;

(2) provide input values to and receive output values from each task iteration; and

(3) report errors to the local executive task.

Fault isolation

1) Damage isolation
preventing physical damage from spreading beyond carefully prescribed boundaries.

Techniques for damage isolation include physical barriers to prevent propagation of mechanical and thermal effects and electrical barriers (for example, high-impedance electrical connections and optical couplers).

In SIFT, such damage isolation is provided at the boundaries between processing modules and buses.

Fault isolation

2) Protection against the corruption of data provided in SIFT by the way in which units can communicate

- A processing module can read data from any processing module's memory, but it can write only into its own memory.
- Thus a faulty processor can corrupt the data only in its own memory and not in that of any other processing modules.

- **All faults within a module are treated as if they have the same effect**: namely, that **they produce bad data in that module's memory**.

  The system does not attempt to distinguish the nature of a module fault. In particular, it does not distinguish between a faulty memory and a processor that puts bad data into an otherwise nonfaulty memory.

- A processor can obtain bad data if those data are read from a faulty processing module or over a faulty bus. Preventing these bad data from causing the generation of incorrect results is solved by **fault masking (voting)**.

Fault isolation

3) Fault isolation also requires that invalid control signals not produce incorrect behavior in a nonfaulty unit.

In general, a faulty set of control signals can cause two types of faulty behavior in another unit:

(1) The unit carries out the wrong action (possibly by doing nothing), and

(2) the unit does not provide service to other units.

In SIFT these two types of fault propagation are prevented by making **each unit autonomous, with its own control**. Improper control signals are ignored, and time-outs are used to prevent the unit from hanging up, waiting for a signal that never arrives.

Scheduling

The types of timing requirements on the SIFT system:

- **Output to the actuators must be generated with specified frequency.**
- **The delay between the reading of sensors and the generation of output to the actuators based upon those readings must be kept below specified limits.**

To fulfill these requirements, an **iteration rate is specified for each task**. The scheduling strategy must guarantee that the processing of each iteration of the task will be completed within the time frame of that iteration. It does not matter when the processing is performed, provided that it is completed by the end of the frame.

Moreover, **the time needed to execute an iteration of a task is highly predictable**. The iteration rates required by different tasks differ, but they can be adjusted somewhat to simplify the scheduling.

The scheduling strategy chosen for the SIFT system is **a slight variant of the simply periodic method**.

Processor synchronization

The SIFT intertask and interprocessor communication mechanism allows a degree of asynchronism between processors. However the processors must periodically resynchronize their clocks to ensure that no clock drifts too far from any other.

Processor synchronization

The traditional clock synchronization algorithm for reliable systems is the **median clock algorithm**, requiring at least three clocks.

In this algorithm, each clock observes every other clock and sets itself to the median of the values that it sees.

The justification for this algorithm is that, in the presence of only a single fault, either the median value must be the value of one of the valid clocks (case 1, case 2) or else it must lie between a pair of valid clock values (case 3). In either case, the median is an acceptable value for resynchronization.

Clock A, Clock B, Clock C: faulty

1) C < A, B
2) C> A, B
3) A < C < B

The weakness of this algorithm is the Byzantine fault, that may cause other clocks to observe different values for the failing clock

Processor synchronization (Consensus problem)

In the presence of a fault that results in other clocks seeing different values for the failing clock, the median resynchronization algorithm can lead to a system failure.

Consider a system of three clocks A, B, and C, of which C is faulty. Assume clock A < clock B. Assume the failure mode of clock C is such that clock A sees a value for clock C that is slightly earlier than its own value, while clock B sees a value for clock C that is slightly later than its own value (Byzantine faults).

Clock C: faulty

A:10	B: 20	C: 8	-> Clock A=10
A:10	B:20	C: 22	-> Clock B=20

Median clock algorithm:

Clock A=10 Clock B= 20

Clocks A and B will both see their own value as the median value, and therefore not change it.

To synchronise clocks a Consensus algorithm is applied.

Fault Detection

Fault detection is the analysis of errors to determine which components are faulty.

Processor/bus error table, an m by n matrix, where m is the number of processors and n the number of buses in the system.

Each processor **p** has its own processor/bus error table **Xp** that is maintained by its local executive's error handler.

Xp[i,j] represents the number of errors detected by processor p's local executive that involve processor i and bus j.

Suppose that processor p is reading from processor i using bus j.

There are five distinct kinds of errors:

1. The connection from bus j to processor i is faulty.
2. The connection from processor p to bus j is faulty.
3. Bus j is faulty.
4. Processor i is faulty.
5. Processor p is faulty.

Fault Detection

Processor p's error-reporting task analyzes the processor/bus error table

If the number of errors is greater than a given threshold, an appropriate action can be taken.

In case 1, processor p will stop using bus j to talk to processor i.

In cases 2 and 3, processor p will stop using bus j, and will report to the global executive that bus j is faulty.

In case 4, processor p will report to the global executive that processor i is faulty.

The **global executive task makes the final decision about which unit is faulty**.

It reads the faulty processor reports provided by the error-reporting task.

- if two or more processors report that another processor is faulty, then the global executive decides that this other processor has indeed failed.

- if two or more processors report that a bus is faulty, then the global executive decides that the bus has failed.

Fault Detection

It can be shown that **in the presence of a single fault, the above procedure cannot cause the global executive to declare a nonfaulty unit to be faulty**.

With the appropriately malicious behavior, a faulty unit may generate error reports without giving the global executive enough information to determine that it is faulty.

For example, if processor p fails in such a way that it gives incorrect results only to processor q, then the global executive cannot decide whether it is p or q that is faulty.

However, the majority-voting technique will mask these errors and prevent a system failure.

Reliability prediction

Reliability requirement is that the probability of failure should be less than 10^{-9} per hour in a flight of 10 hours' duration.

High reliability of survival for a short period time (10-hour flight).

For a flight of T duration survival will occurr unless certain combination of failure events occurr within the interval T or have already occurred prior to the interval T and were undetected by the initial checkout of the system.

Show that the probability of a more catastrophic sequence of failures is sufficient small.

Finite state Markov process. The combined probability of all event sequences that lead to a failed state is the system failure probability. Failure rate of 10^{-9} for a 10-hour period T

Assumptions:

- Hardware-fault events are independent and exponentialy distributed in time (constant failure rate)
- All failures are permanent for the duration of the flight

Accurate because:

- the physical design of the system prevents fault propagation between functional units
- a mutiple fault in a functional unit is no more serious than a single fault

Effects of transient errors are masked by the executive system which requires a unit to make multiple errors before it consideres the unit to be faulty.

The execution of critical tasks in loose synchronism also helps protect against correlation of fast transient errors.

Failure rates for hardware have been estimated on the basis of active component counts, using typical figures for similar hardware :

main processors 10^{-4} per hour failure rate

State of the system in the reliability model (h, d, f) with h<=d<=f

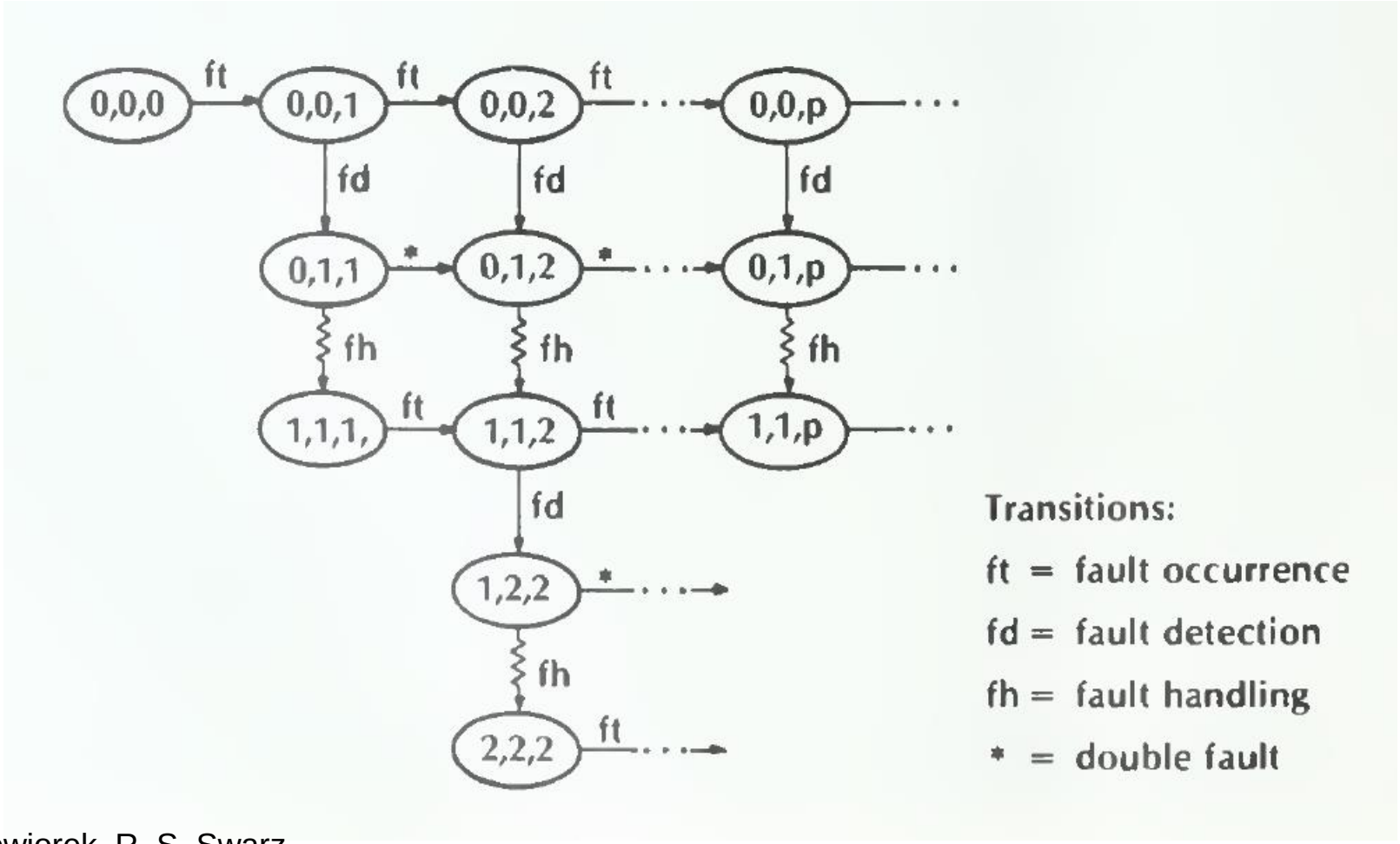
(h, d, f) represents a situation in which:
- f failures of individual components have occurred
- d of those failures have been detected
- h of these detected failures have been handled in reconfiguration

Three types of possible transitions:

- $(h, d, f) \rightarrow (h, d, f + 1)$, representing the failure of a processor
- $(h, d, f) \rightarrow (h, d + 1, f)$, $d < f$, representing the detection of a failure
- $(h, d, f) \rightarrow (h + 1, d, f)$, $h < d$, representing the handling of a detected failure

The first two types of transitions are represented by straight arrorws (costant probabilities for unit of time)

The third type of transition is represented by wave arrows, represents the completion of a reconfiguration procedure.
(these transitions are assumed to occurr within some fixed length of time τ)



D.P. Siewiorek, R. S. Swarz
Reliable Computer Systems
(Design and Evaluation) Prentice Hall, 1998.
Chapter 10 – "The SIFT Case: Design and Analysis
of a Fault Tolerant Computer for Aircraft Control".

A state (h,d,f) with h<d represents a situation in which the system is reconfiguring.

In make the system immune to an additional failure while in this state is a difficult problem because it means that the procedure to reconfigure around a failure must work despite an additional undetected failure.

Instead of solving this problem, designers took the approach of trying to

ensure that the time τ that the system remains in such state is small enough to make it highly unlikely for an additional failure to occurr before reconfiguration is completed.

They made the pessimistic assumption that a process failure that occurs while the system is reconfiguring will cause a system failure.

Designers calculated the probability of system failure through a double-fault transition and also through reaching a state with fewer than two nonfaulty processors, for which they said the system has failed because it has run out of spares.

Failure probabilitie for a 5 processor system and T= 10 hours

Failure Cause	Failure Probability
Exhaustion of spares	5×10^{-12}
Double fault (τ = 100 msec)	7×10^{-11}
Double fault (τ = 1 sec)	7×10^{-10}

D.P. Siewiorek, R. S. Swarz
Reliable Computer Systems
(Design and Evaluation) Prentice Hall, 1998.
Chapter 10 – "The SIFT Case: Design and Analysis of a Fault Tolerant Computer for Aircraft Control".

Summary

SIFT basic approach to fault tolerance involves the replication of standard components, relying upon the software to detect and analyze errors and to dyamically reconfigure the system to bypass faulty units.

Special hardware is needed only to isolate the units from one another, so a faulty unit does not cause the failure of a nonfaulty one.

Processor/memory modules and bus modules as the basic units of fault detection and reconfiguration have been used. These units make system reconfiguration easy and are small and inexpensive enough to allow sufficient replication to achieve the desired reliability.

By using software to achieve fault tolerance, SIFT allows considerable flexibility in the choice of error handling policies and mechanisms.

For example, algorithms for fault masking and reconfiguration can be easily modified on the basis of operational experience.

Novel approaches to the tolerance of programming errors can be incorporated. Moreover, it is fairly easy to enhance the performance of the system by adding more hardware.