

# Introduction

## Concept

### **Dependable System**

It is a system for which the probability of a system failure is negligible (below certain threshold).

### **System Failure**

It is the deviation of the behavior of a system from its specification.

### **Failure Propagation**

Fault cause of an error, incurs incorrect internal state. Finally cause failure which is externally visible deviation from specification.

## Failure/Error/Fault

### **Definition**

- Failure is externally visible deviation from the expected result.
- Error is incorrect internal state
- Fault is the incorrect which cause of an error.

## Detection

- Failure, observe the behavior of component, detects deviations from its specification.
- Fault, check input before updating state
- Error, detect when the state become incorrect.

Note: The detector may not detect all the system failures. The probability of undetected failure =  $p \cdot q$ ;  $p$  = failure probability,  $q$  = probability of detector makes wrong detection\*

## Fault Masking

### Triple Modular Redundancy

3 components provide the same service, and voter takes the inputs and masking the fault uses majority of 3 components's outputs.

#### Properties

- Failure of component should be independent, 3 components won't interfere each other. (Implemented in different version. A.k.a N-Version Programming)
- Outputs are deterministic.

### Fault Masking via Detection

C3 uses fault detector for C1 and C2 to select correct output. If one of C1 ~~and~~ C2 is correct, C3 can mask the fault, otherwise fail-stop.

#### Compare to TMR

- Detector normally use different implementation, and can remove the common mode failure and improve independence
- It's more cost effective than TMR.

## Fault Masking via Reconfiguration

If one component is fail, then start a new component to mask that fault.

## Fault tolerance

If the system can prevent the propagation of system failure.

- Fault Masking: after detect a fault, the system mask the fault or stop it becomes a error.
- Error Detection and Recovery: role back to good state; or roll forward to good state, escape the corrupted state.
- Reconfiguration: switch to a better component to continue the task. When switching the component may hot/warm/cold.<sup>1</sup>

## Design Dependable Systems

$10^{-9}$  Challenge = 1 failure /  $10^9$  hours

---

<sup>1</sup> Hot: spare component is synced to current state, can take over the task immediately.  
 Warm: spare component is not in latest state, but can recover very quickly.  
 Cold: spare component doesn't have any state, needs to initiate.

## Divide & Conquer

- program is correct
- program is executed correctly
- correct program is executed
- our failure assumptions are correct

## Commercial System

### Fundmental Knowledge

#### Availability Requirements

- Continuous availability: the system will operation 24/7, won't stop
- High availability: the system will available for 100% **during operation** it has scheduled downtime
- Basic availability: short interrupt is ok, apply to most of the apps

#### MTTF/MTTR

- MTTF = mean time to failure
- MTTR = mean time to repair
- Availability =  $MTTF / (MTTF + MTTR)$
- Unavailability =  $MTTR / (MTTF + MTTR) \approx MTTR / MTTF$

## Scheduled/Unscheduled Outage

- Unscheduled: are unanticipated failures in any part of the technology infrastructure that supports the application.
- Scheduled: are routine interruptions planned ~~well~~ in advance such as those scheduled for routine maintenance or inspection of equipment.

## Soft error

Soft error(a.k.a transits error) caused by bit-flip. It can be detected and repair by the system.

Solution:

- Avoid soft error by using better material and change the mounting of RAM chips.
- Detection and Correction:
  - Parity
  - ECC Hamming Code (7,4)
  - Memory Scrubbing

## Hard Error

Hard error is permanent, it cannot be corrected by software.

Solution: replace by new hardware

# Error Detection and Correction

## Parity

Detects one bit failure, cannot correct failure.

### Even Parity

010010|0

011001|1

Even number of 1, then the parity bit is 0;

Odd number of 1, then the parity bit is 1;

### Odd Parity

010010|1

011001|0

Even number of 1, then the parity bit is 1;

Odd number of 1, then the parity bit is 0;

## ECC

Detect 2 bits failure, can correct 1 bit failure. Hamming distance is 4.

## Memory Mirroring

Each word is mirrored:

- on write: the word is copied to two RAM banks
- On read: when multi-bit error is detected, read data from mirror; permanent switch to spare bank

## Memory Scrubbing

### Why we need memory scrubbing?

If a memory location doesn't read/write for a long time, the bit flips will increment, ~~after its humming distance is greater than 3~~, we cannot detect the error anymore.

#### Mechanism:

- read ECC memory periodically
- correct bit flips before 2nd bit flip occurs

## Fail-Fast<sup>2</sup>

Fail-fast component will detect a failure as fast as possible, stop this failure to prevent it propagate to other components.

## IBM zSeries used Mechanisms<sup>3</sup>

Prioritized Principles:

1. Ensure system/application integrity
2. Provide uninterrupted applications
3. Repair without disruption

## Faut Model

1. Transient: soft error

---

<sup>2</sup> a.k.a Fail-Stop

<sup>3</sup> More details in Paper 02-01

2. Permanent: hard error

## Hardware Retry

When CPU detects a bit flip on bus by parity, simple retry to access memory.

## ECC Memory

ECC DIMMs typically have nine memory chips on each side, one more than usually found on non-ECC DIMMs.

## Duplicate Execution to solve bit flips in ALU

Pending instruction, results are kept in both a store through microprocessor cache (L1) and an ECC-protected store buffer and completed results are immediately stored into L2. L1 data is thus always replicated, allowing byte parity to be sufficient within the cache.

A transient L1 failure is recovered by CPU **instruction retry**.

For a permanent failure, depending on the scope, a cache line or quarter cache delete is performed dynamically. A deleted line may be restored with a spare line at the next power on. Shared Level 2 (L2) cache is protected by ECC. Permanent faults in L2 that might result in an uncorrectable data error can be avoided by using a cache delete capability.

Faulty locations either in the data array or in the address directory can be dynamically marked as invalid and a spare line can be substituted for the failed one.




## Instruction Transparent Retry

When an error is detected: Except for the R-unit itself, the CPU is reset and store buffer contents are sent to L2.

If retry is successful, the failure was transient and the CPU resumes pipelined instruction processing. If not, the CPU will stop.

## zSeries Parallel

Multiple mainframes can  connected together via a *coupling facility*. All mainframes in the system sharing data have equal access to shared disks as well as to the coupling facilities which manage the sharing.

From a software perspective, each mainframe is a clone of the others. Transactions are distributed to the mainframes via a Workload Manager that determines which mainframe is least busy for the particular type of job.

## Tandem NonStop Mechanism<sup>4</sup>

The general design principle was that there are at least two of everything, including power supplies and fans as well as the more obvious processors, controllers, and peripherals. Dual-ported controllers and dual-ported peripherals provided four paths to each device.

## Fault Model

1. Recoverable
2. Nonrecoverable

---

<sup>4</sup> More details in paper [02-01] Section 2

## **Lockstep processors\***

Two microprocessors using the same clock have their outputs compared after each operation and immediately signal any discrepancy.

### **Approach**

- compare the output of two CPUs
- very unlikely that both CPUs produce the same incorrect failure
- if the result is not the same abort the result

**Problem: CPUs are non-deterministic**

## **End-to-End disk checksums**

For each block, a checksum covering the data and block address is calculated in the processor, written to disk along with the data, and verified by the processor when the block is read. In the event of a checksum error, the data is read from the other member of the mirrored pair.

## **CRC**

Packets are protected by a (CRC) checksum; lost or corrupted packets are re-transmitted.

## **NonStop Advance<sup>5</sup>**

Instead of lockstepping microprocessors, the NSAA detects processor failures by comparing the outputs of I/O operations (both IPC and device I/O)

---

<sup>5</sup> More details on paper [02-02] section 3

## Loose Lockstep

Each process can execute in different clock and retries, etc. After the execution instead comparing the memory CPU writes to memory (as Lockstep). It will compare I/O stream instead of CPU writes.

**The problem is** Interrupts to the processor, such as I/O completion interrupts, arrive at each PE at slightly different times. If the processor were to immediately handle interrupt, each PE would be interrupted at a different point in its instruction stream. This divergent execution would result in asymmetric memory state in the logical processor. In order to keep memory symmetric, each processor element must handle interrupt at the exact same point in the instruction stream as the other PEs in the logical processor.

**Solution:** Rendezvous

## Rendezvous\*

**Goal:** agree at which VRO an interrupt should be processed

### Protocol

~~After receiving an interrupt, each PE initiates a rendezvous operation. A rendezvous operation consists of each PE writing special rendezvous registers in the voter logic.~~  
~~RO = 0~~

~~RO ++~~

~~After all the writes complete, the voter reflects the data from the rendezvous registers back to a specific block of memory in each PE.~~

~~RT = 2 //Rendezvous Threshold~~

~~Rendezvous code in each PE then reads the block of data to see what the other PEs wrote. The PEs use the block of data to propose where in the instruction stream they intend to execute the interrupt handler code for a specific interrupt.~~

~~if R0 = RT, excute interupt.~~

~~The intent is to execute VRO code periodically.~~

~~The VRO is a small section of code inserted into the code stream. At a minimum, the VRO code will be inserted into every privilege level transition.~~

~~Short Version:~~ 1. Upon an interrupt arrival, each CPU proposes a VRO at which he can handle the interrupt. 2. A voter component chooses a maximum VRO from the proposed VROs. 3. Each CPU memorizes the maximum VRO chosen by the voter. 4. Each CPU continues execution until the chosen VRO occurs. 5. Each CPU handles the interrupt.

## UNPC-Store & UNPC-Trace

The UNCP-Store algorithm identifies process **state** that could be different in the PEs. It then chooses one PE (the source PE) and copies its values for that state to the other (target) PEs, putting the target PEs at the same execution point as the source PE.

Due to copying some state from the source to the target PE(s), the UNCP-Store algorithm is **vulnerable to error propagation**. ~~We expect this risk to be quite small, both because the algorithm should be executed infrequently and the amount of state that is copied should be small. The UNCP-Trace algorithm addresses this vulnerability.~~

~~The second algorithm, UNCP-Trace, does not copy state from one PE to the others. Instead, it~~ determines which PE is ahead, and by how far, and executes **instructions** in the trailing PEs until they are synchronized with the leader.

## Reintegration<sup>6</sup>

The reintegration procedure copies the state of the running, on-line PE's memory into the memory of the newly added PE. After the memory state is copied, the new PE starts executing the exact same instruction stream as the other PEs.

# Main Memory

## Memory Architecture

CPU - Channel(Bus) - Memory Controller - Memory Modules (DIMM)

**Memory Controller:** Memory controller contains the logic necessary to read and write to DRAM, and to "refresh"<sup>7</sup> the DRAM. Without constant refreshes, DRAM will lose the data written to it as the capacitors leak their charge within a fraction of a second.

### SRAM vs DRAM

---

<sup>6</sup> More details on paper [02-02] section 3

<sup>7</sup> Implication: The refresh rate should be high enough to avoid bits flip.

- SRAM is fast and most of the time used as CPU cache (L1,L2), the data stays on SRAM is longer than DRAM.
- DRAM is slower and uses transistor, it's more dense than SRAM, used as main memory.

**Memory Rank:** is a set of DRAM chips connected to the same **chip select** and can be accessed simultaneously. They also share all the other command and **control signals** and only the **data pins** for each DRAM are separate.

**Memory Bank<sup>8</sup>:** a bank consists of multiple rows and columns of storage units and is usually spread out across several chips.

**Channel:** is a synchronous bus, connect between DIMM and memory controller.

**RDIMM:** has a register which placed between the memory and the memory controller, and the register re-drives command, address and clock signals.

**Load Reduced DIMM (LRDIMM):** modules are similar to RDIMM. LRDIMM modules has **memory buffer which buffer both control and data lines while keeping the parallel nature of all signals**. LRDIMM memory provides large overall **maximum memory capacities**, and reduce channel signals.

---

<sup>8</sup> Smaller bank can reduce the latency; More page hits, lower latency

## Dependability Implication

1. Power: the main memory will lose all the state if the power is off.  
We should design a redundant power supply to tolerate this kind of failure.
2. Crash: program or process crashes will cause the corrupted memory state. After the process recovered, the previous state may be lost.
3. Temperature: higher temperature means higher leakage on DRAM, we need to adjust the rate of refresh DRAM dynamically.

## RAS Mechanisms

- Reliability: keeps data integrity (1. error detection and correction; 2. prevent error propagation)
- Availability: guarantees access data uninterruptedly
- Serviceability: provides simplified method to deal with failures

## DRAM Correctness

If CPU reads memory cell  $i$  at time  $t$ , it receives the most recent value written to cell  $i$  before time  $t$ .

## DRAM Fault Model

- Stuck at fault: stuck at a certain value
- transition fault: bit-flip
- Coupling fault: transition of cell  $i$  causes transition of cell  $j$

- Address fault: R/W the wrong cell

## Detect and Correct

- Parity
- ECC
- ChipKill: if one chip provide wrong data, the data will be eliminated. Use extra ECC bits to mask that fault.

## Parity Correct in Example

Parity data is used by some RAID levels to achieve redundancy. If a drive in the array fails, remaining data on the other drives can be combined with the parity data (using the Boolean XOR function) to reconstruct the missing data.

For example, supposes two drives in a three-drive RAID 5 array contained the following data:

Drive 1: 01101101  
Drive 2: 11010100

To calculate parity data for the two drives, an XOR is performed on their data:

01101101  
XOR 11010100  
10111001



The resulting parity data, **10111001**, is then stored on Drive 3. Any of the three drives fail, the contents of the failed drive can be reconstructed by subjecting the data from the remaining drives to the same XOR operation.

If Drive 2 were to fail, its data could be rebuilt using the XOR results of the contents of the two remaining drives, Drive 1 and Drive 3:

```
Drive 1:    01101101
Drive 3:    10111001
XOR Parity: 01101101
Rebuilt:    11010100
```

## SoftWare Implemented Fault Tolerance

### Using memory errors to attack JVM

#### JavaVM type checking mechanism

1. Translate to byte code
2. Byte code verifier guarantee safety of the program
3. Untrusted code and trusted code store run in the same address space. When untrusted code system call via API.

#### Goal

To obtain two pointers of incompatible types that point to the same location, then execute an function to reads and writes arbitrary memory location, hence executes arbitrary code.

## Fault Mode

Outside hardware cause the bit flip happens, the pointer will change. Then we can have two pointer in two different object point to the same object.

## Why we can use this kind of attack in JAVA VM?

Because JAVA uses link-time type-checking instead of run-time checking. Java translates the program to byte code and store them with trusted code. JAVA VM only check the program at Link-time, not the run time to improve the performance. But run time program can be changed by hardware method.

## Attack Code

We assume we have equal pointers **p** and **q** of types A and B,

```
A p;  
B q;  
int offset = 6 * 4;  
void write(int address, int value) {  
    p.i = address - offset ; (1)  
    q.a6.i = value ;(2)  
}
```

(1) writes **address - offset** at the field **q.a6**<sup>9</sup>. (2) writes value at an **offset** of **offset** from **q.a6**. Thus the procedure writes value at **offset + (address - offset) = address**.

---

<sup>9</sup> it is a ~~pointer address now~~

## Input/Output Via System

In order to make the program runs correctly, we need check the **arguments** passed to a system call.

**Solution:** end-to-end checksums

## Control flow checking

To make sure all program branches are executed correctly.

Comparing a run-time signature with a pre-computed signature.

### Control flow error\*

Deviation from the program's correct instruction flow execution:

1. Branch-error: error in branch instruction, program doesn't execute as expected branch.
2. IP error: error in Instruction Pointer.<sup>10</sup>

### Type of control flow error\*

A: condition error

B: jump to self node beginning

C: jump to self node middle;

D:        j u m p        t o        o t h e r        n o d e - b e g i n n i n g

E: jump to other node-middle

F:        j u m p        o u t        o f        t h e        s e g m e n t a t i o n

---

<sup>10</sup> This kind of error is not easy to detect by CFC, but can be detect by DFC.

A-E can be detected by using CFE with signature. F can be easily detected and protected by (hardware) memory access protection mechanisms.

## **Solution for CFE with Signature\***

D1: enter calculate constant

D2: exit calculate constant

S: unique Signature in program (Random)

```
S = S xor D1
if (S! = 1001)
// 1001 is unique signature
about();
S = S xor D2
```

**Node X (xor) Type:** if more than one predecessor node, then all predecessors have exactly one successor node

```
Entry: S = S xor D1
Exit:  S = S xor D2
```

**Node A (and) Type:** more than one predecessor and one predecessor has at least two successors

```
Entry: S = S and D1
Exit:  S = S xor D2
```

Note: Cannot detect Class C, jump to the beginning and Class A if it has multiple successors.

## Data flow checking

Make sure the correct values are computed. Data-flow checking rely on **redundancy**.

## Fault Mode

- Operator error:  $a=x+y \rightarrow a=x-y$
- Modified operand error:  $a= x+4 \rightarrow a=x+5$
- Wrong operand error:  $a=b[5]+1 \rightarrow a=b[4]+1$
- Operation error:  $a=x+y \rightarrow a=x+y+4$
- Lost updates:  $a=x+y \rightarrow b=x+y$

## Sphere of Protection

- Instruction stream level: checker cpu
- Machine level: ~~redundancy~~ binary code
- Assembly code level: redundancy to assembly code
- Source code level: redundancy to source code

*Higher abstract level covers lower level's fault and failure.*

## Time Redundancy

Execute the code multiple **times**.

## SWIFT

SWIFT is a compiler-based transformation which duplicates the instructions in a program and inserts comparison instructions at **strategic points**.

Advantage: Don't need external hardware to support; can adapted to vary transient fault policy; efficient;

### **Duplicate Instruction**

Each duplication of the program uses different registers and different memory locations(prevent interfere).

At certain **synchronization points** in the combined program to make sure that the original instructions and their redundant copies agree on the computed values.

### **Synchronization Points**

- Store function: since we define the correctness by the output of program, **store** maps the output to the memory, so if the **store** values are correct, we consider that the program executed correctly.
- Branch instructions: **branches** can cause store to be skipped, incorrect stores to be executed, or incorrect values to ultimately feed a store.

### **Window of Vulnerability**

Two primary points-of-failure:

1. there can be a delay between **validation** and **use** of the validated register values, any strikes during this gap might corrupt state.
2. if an instruction opcode is changed from a non-store instruction to a store instruction by a transient fault. These stores are unprotected because it happens after the compiler translated the code.

### **Conquer WoV with SWIFT-R**

Approach: use extra copy to permit recovery from errors.<sup>11</sup>

1. Triplicate loaded values and operations
2. Use **majority** mechanism to load values or store

## Replicator

Replicate memory and instructions, same as NonStop, instead on different hardware, execute them on same hardware.

### Check point

- In-going wrapper: un-replicated code wants to call replicated code; **replicate the arguments** and **call the replicated** code.
- Out-going wrapper: ~~replaced~~ code wants to call un-replicated code; compare replicated arguments, iff they are the same call the un-replicated code.

## Space Redundancy

Execute on multiple hardware (E.g. HP NonStop Mainframe).

**Con:** Using hardware fault tolerant mechanisms is too expensive for many processor markets. And these mechanisms will not **differ** the low critical and high critical subsystem. They will deal the problem in a same level.



## Information Redundancy

Encode information, add extra information to detect the error of data.

---

<sup>11</sup> Similar as TMR

## Assumption Coverage

- Failure assumption: defines how a component can fail
- Assumption Coverage: Probability that the failure assumption is true in case a failure has occurred.
- Implication: if the fail-stop failure is high. For example, 99% of the system failure is fail-stop failure, and 1% is unconstrained failure. Then we only need  $F+1$  instead of  $3F+1$  components to deal the fault tolerance, and still provide better availability.

## Encode processing

Each output assigned a unique random signature. Compare the output's signature with pre-set signature, if it is not the same fail-stop the program.

**Signature has  $N$  bits, detected number of failure will be  $2^N$ , and undetected will be  $2^{(-N)}$ ; increase the  $N$ , improve the assumption coverage.**

## Arithmetic Codes

Detect hardware design failures and compiler failures.

- AN Code
- AN+B Code
- AN+B+D Code
- Signature Computations: calculate the right result ahead, if the program is wrong can compare the sign



## ParExC

- One thread runs the original program as speculator. It pre-calculates the Signature
- Periodically take a snapshot, runs encode program on other threads.
- After certain point, check the results and Signature to prove the speculator is right.

## Determinate Replay

The Encode program on other threads may execute some instruction before the original program, cause the replay out of order. So we need a synch mechanism to solve this problem.

Using the **kernel extended module**, when uninstrumented codes execute system calls, it will speculatively execute. After the Encode program sync with them, it will actually executed in kernel.

## ~~Speculative Checker Variables~~

~~Malloc on one thread executes behind the write. We need set an obligation for that variable. When malloc is executing, check the obligation from other threads.~~

# Perfect Failure Detector

## Failure Model

- Fail-operational model: a safety-critical<sup>12</sup> system which is without a safe state when the computer subsystem has failed. (e.g. Fly-by-wire)
- Fail-safe model: a safety-critical system that has a safe state. (e.g. Railway crossing. When the system is failure, we can stop this system and it won't cause further problem)

Sometimes one can convert a fail-op into fail-safe **by using backup system**

## Fail-over Operation

- Basic idea: when Primary server fails, switch to Secondary server.
- Practical problem: sometimes the Primary is just delayed, but Secondary server has already took over. When Primary is up again, will cause inconsistent data.
- Hardware solution: use shared SCSI bus.
- **Software** solution: perfect failure detector.

---

<sup>12</sup> most of real-time system is safety-critical system, when it failures, may cause life at risk

## PFD Properties\*

- Accuracy: each process which is suspected, is crashed.
- Completeness: eventually the crashed process will be suspected by each correct process.
- Monotonicity: if process  $p$  suspects  $q$ , then  $p$  will permanently suspect  $q$ .

*PFD to MPFD is just add the assumption that crashed process will never recover*

## Completely Synchronous System

If a system is synchronous system<sup>13</sup>, it has to fulfill these properties:

- all messages are delivered within  $\delta$
- scheduled events are executed within  $\sigma$  of their scheduled time
- clocks are internally synchronized<sup>14</sup>
- the number of process is finite  $N$



## Process Service

A process  $P$  can request to execute some action at clock time  $T$ , and should be awoken and execute action within  $[T, T+\sigma]$

---

<sup>13</sup> alias Real-time System

<sup>14</sup>  $|C_p(t) - C_q(t)| \leq \Delta$ ,  $\Delta$  is the maximal internal clock deviation

## Process Failure Assumption

A process crashes by stopping to execute its program (fail-stop)

- only crash failures are permitted and at most  $F < N$  processes can crash
- process  $p$  is correct in time interval  $[s, t]$  iff  $p$  is at no point in  $[s, t]$  crashed

## Datagram Service

The message transmission delays should be less than  $\delta$

## Clock in Synchronous System

- Drift Rate
- Internal clock synchronization<sup>15</sup>
- External clock synchronization<sup>16</sup>

## Implement Synchronous System

*Contention and queuing delays are problems when we try to build a Synchronous System in real world. If you cannot predict and avoid this delay, the upper bound for time in code won't work.*

**Solution: Time-triggered ethernet**

---

<sup>15</sup> the deviation with other process is in a certain bound

<sup>16</sup> The deviation with realistic time is in a certain bound

- Assigned different time slot to different process to avoid contention.
- Segmentation of network (build more path for connection).
- Sometimes 2 sensors send message to process, but the rate is too fast for process. Process will drop off some message. We need to handle this problem by restrict some nodes' speed.

### **Fault tolerant**

*When there is only one TTE in the system, it suffers single point of failure, and sometimes there is babbling idiots in the system keep send meaningless message*

#### **Solution:**

- Secondary TTE for safe-critical process. When one of the switch is not working, the backup TTE still can handle the important messages.
- Guardian: if the sensor sent a message out of his time slot, the guardian will stop forwarding this messages.

## **Implement PFD in Sync. System**

### **Pull mode**

- Basic idea: failure detector polls information from remote process.  
P sends FD request to Q, if Q replies in, P considers Q is still alive.

- Time constant:  $2(\delta + \sigma)$ ;  $\delta$  is transmission delay,  $\sigma$  is scheduled execution time.

## Push mode

- Basic idea: failure detector broadcast alive messages to other process periodically. If after certain period time, P didn't get message from Q, will consider Q is failed.
- Time constant:  $P + \sigma + \delta + \sigma + \Delta$ ; P is periodic interval,  $\Delta$  is internal clock deviation.

## Comparison



In pull mode, detector knows the status of crashed process very short, only in  $2(\delta + \sigma)$ ; if the process only interested in certain process and ask infrequently, the message will be more less.

## Timed Asynchronous System

### Process Service

A process P can request to execute some action at clock time T, and should be awoken and execute action within  $[T, T + \sigma]$

### Process Failure Assumption

- **Performance failures:** process execute the event later than  $[T, T + \sigma]$

- **Crash failures:** process crashed

## Datagram Service

At most once, request is sent again in case of failure, but request is filtered on the server for duplicates.

**Performance failure:** message transmission delay  $> \delta$

**Omission failure:** message transmission delay  $= \infty$

## Local Hardware Clock Service

Each process has a correct hardware clock, that means the clock has the following property:

$$(t-s)(1-p) \leq H_p(t) - H_p(s) \leq (t-s)(1+p)$$

**No external or internal clock sync required**

## PFD in Asynchronous System

- Why we cannot have PFD in asynchronous system?

Consider the situation in which somebody accidentally disconnects the network cables of participant **c** for more than  $2\delta$  time units. After  $2\delta$  time units, failure detection on participant **d** will incorrectly suspect **c**. However **c** connected after certain time. **D** will violate the properties of perfect failure detector.



- Simulated PFD. Paper section 3.2
- Alternative. Paper Section 6. & 7.

# Comparison of Synchronous and Asynchronous System

1. Failure/fault detector, triple modular redundancy
2. UNCP-store, UNCP-trace
3. Scheduled/unscheduled outage(diagram)
4. What is availability, MTTR
5. Draw memory architecture, which errors can occur, DIMM, memory rank, memory bank
6. Describe 4 methods of memory's protections - 2 hardware and 2 software, comparing two technologies of your choice
  - Memory Scrubbing
  - ECC
  - Parity
  - ChipKill
  - Memory Mirroring
7. Something about ParEX (about its kernel)
  - Kernel Module
8. Compare some approach from the lecture with transactional memory (it should be in some paper)
  - a) parallelizing the application and checks
  - b) parallelizing only the checks
9. A lot of questions about paper"javaVM attacks"
- 10.Synchronous vs Asynchronous system