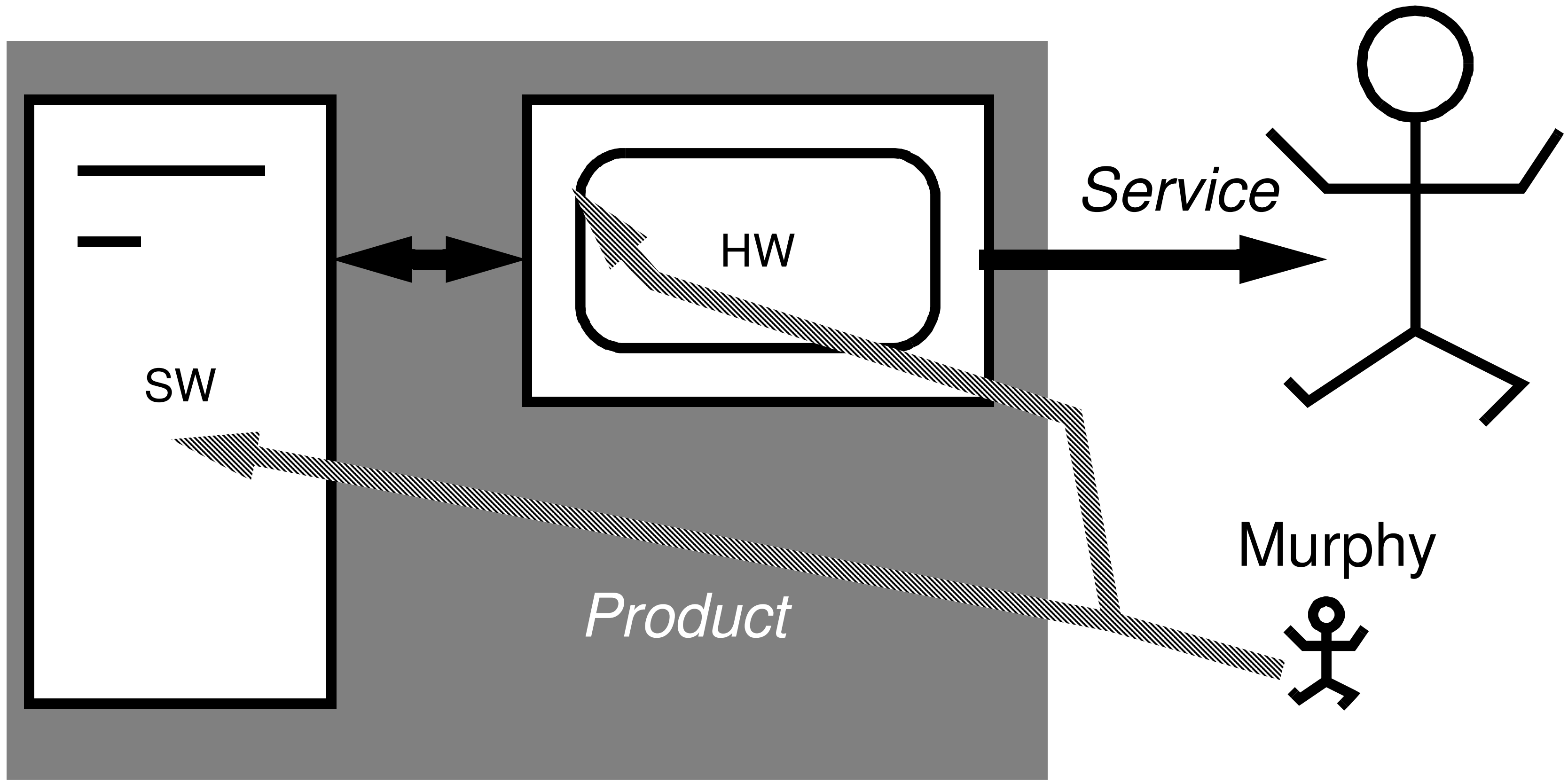
Paradox approaches

- **The designer offers:** **Quality of *product* (?)**
- **The user wants:** **Quality of *services***

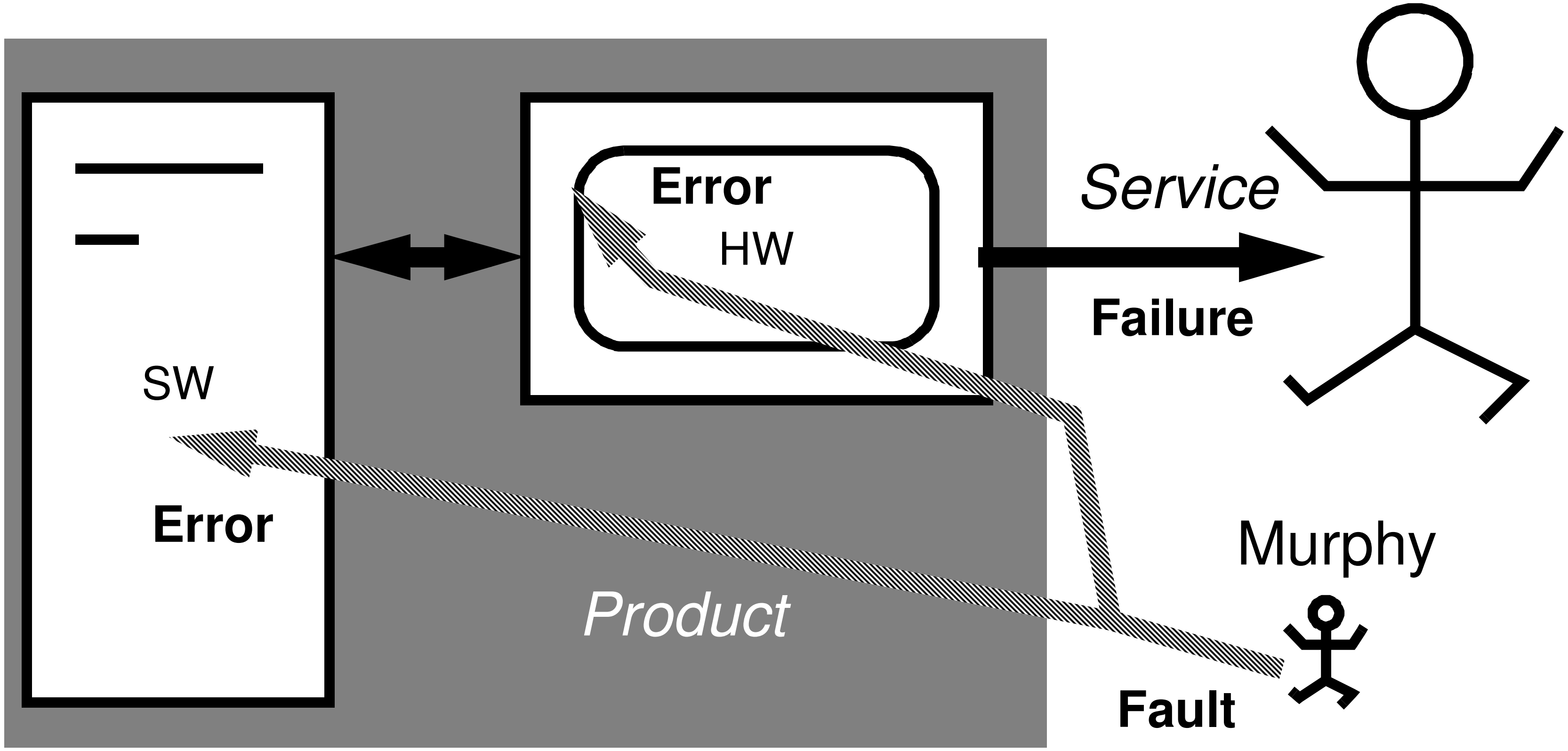


Importance of dependability:

- **Is the story of computer applications a SUCCESS or a HORROR story?**
- **Some NEGATIVE experiences collected from a *VERY* long list:**
  - **USA: approximately 4 billion $/year damage/year**
  - **Several accidents:**
    - **Radiation therapy**
    - **Airplane**
    - **Ariane 5**
  - **Collapse of stock exchange, phone and banking systems**
  - **Administration:**
    - **Invoice on autopsy (vivisection?)**
  - **Internet**

Dependability

- **"The thrustworthyness of a computer system such that reliance can justifiably placed on the services it delivers" (IFIP WG 10.4 / Laprie)**
- **Fault $\Longrightarrow$ Error $\Longrightarrow$ Failure**

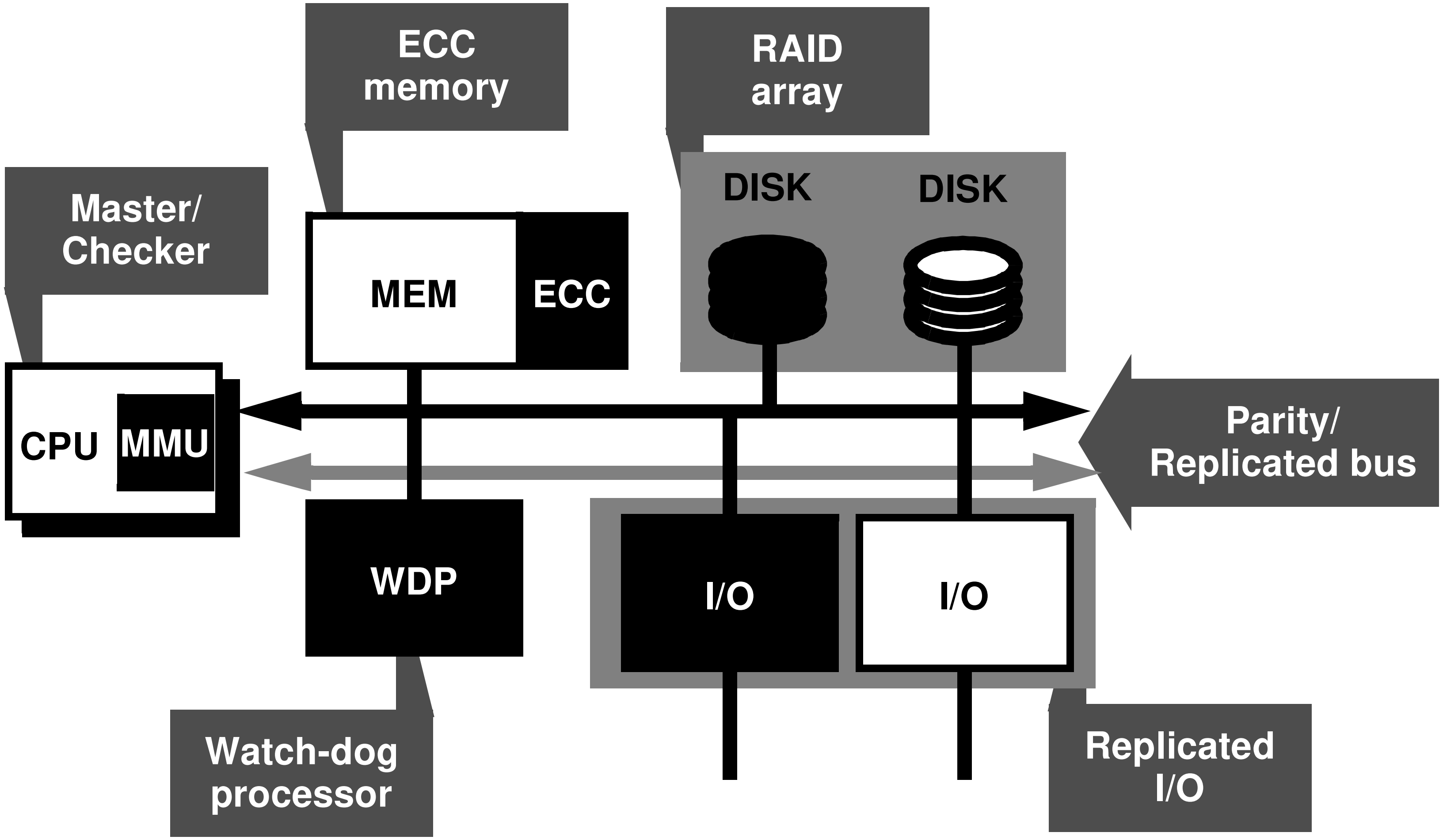


Means for dependability

Phase	Fault handling		Implementation	Target
Before service	Prevention (= no faults, please)	Avoidance	Design & manufacturing technology	Fault
		Removal	Testing	
During service	Robustness (=Russian roulette)		Self-confidence	?
	Tolerance (= proper service in spite of faults)		Redundancy - time - performance - hardware -information	Failure
	Forecasting (= be ready)		- data acquisition prediction	Fault

- **Dependability attributes (IEC 1069-5)**
  - *Reliability*: correct operation for a defined period of time
  - *Availability*: ability in an correct operational state for a defined period of time
  - Safety: avoidance of catastrophes
  - Security: assurance to withstand unauthorized/ incorrect inputs
  - Credibility: Recognition/signalization of correct/ incorrect system state

Hardware implementation
(Mono)computer level



Standard solutions

CPU-MMU

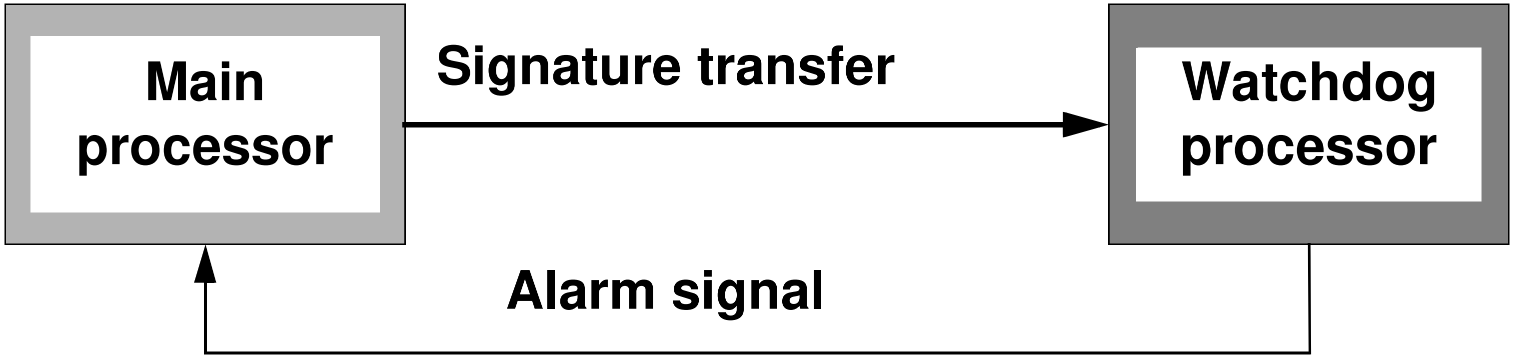
- Illegal opcode
- Arithmetic errors
- Bus error
- Memory address range access rights (fetch, read, read/write) checks
  only task level $\Longrightarrow$ no fine granular checks
  (the entire address range of a task is uniformly mapped)
- Similar object/function oriented implementation, like index checks not (really) supported (need for user - supervisor - user mode context switching + MMU prg.)
- Fault tolerance/ latency?
- Supervisor mode unprotected

Master-checker

- Two *processors* connected pin-by-pin on the bus running the same application
- *Lock-step* operation
- *Comparison* of the values at each signal write operation
- 100% fault coverage for CPU-*internal* single errors, but 0% for CPU-*external* errors
- Hardware overhead: 1 chip???/ 100%???
- No performance loss
- Integrated comparators: e.g. Intel 585.995
- Similar principle at the higher levels of HW: Sun Sparc High Availability server,...

Watch-dog processor

- Simple co-processor checking the *control-flow* of the application program
- Signatures (*numerical labels*) assigned to high-level *instructions* at precompile time
- Signatures *transferred* to the WDP at run time



- Syntactic check of the label sequence but: no data dependencies
- IF-THEN-ELSE: both THEN and ELSE branches accepted, selection unchecked

Summary

- **Trend: COTS solutions**
  - **Economical solutions: dedicated CPU MIPS/ COTS MIPS = 10^6**
- **Only *partial solutions* offered as products by equipment vendors**
  - **No fault tolerant bus standard in the traditional COTS products $\Longrightarrow$**
    - **only add-on**
    - **cOST EXPLOSION**
- **Solution:**
  - **Dependable system architecture**
  - **Computer level redundancy**
  - **Software + hardware**
- **Hardware fault detection techniques can be adopted to software, especially in parallel/distributed environments:**
  + independence of the checker and checks assured
  - communication bandwidth
  - error latency $\Longrightarrow$ fail safety not assured

Software solution: Hierarchy of checks/strategies

- **Algorithm based fault tolerance (ABFT):**
  - **Problem related additional information**
    - credibility checks (limit value or simplified model based *assertion checks*)
    - inverse calculations
  - **Hard to implement as a uniform *mechanism*, only as a problem dependent *measure***
    + To a limited extent human errors (input data) can be checked as well
- ***Syntax based checks - compile-time structure needed:***
  - **Strict type, range checks**
  - **Interface checks**
  - **Restricted call structure (OO?)**
  - **Predictable task sequence (CSP-like?)**
- ***Checkpoints, rollback recovery:***
  - **Problems in reactive systems**
- ***Redundant data structures:***
  - **Double linking of lists**
  - **Checksum-like protection of operations and data structures**
  - **("total" in financial tables, parity-like protected matrix operations)**
- ***Fault tolerant elementary operations:***
  - **Atomic transaction processing**
  - **Fault tolerant commit protocols**

- **Uniform interface: *exception handling***
- **Validation:**
  - **Huge cardinality of the candidate faults $\Rightarrow$ statistical methods**
  - **Simulated fault injection (radiation, bus signal or software)**
  - **No validated fault model on the effect of the transient faults**

Conclusion:

- Quality Control assures, that it works fine, if it works
- Dependability assures, that it works not so very fine, but a little bit if it does not work
- Can we really stop a high-tech car with a defective ABS on the ice???

Let be a depressed schizoid, in order to sleep well!