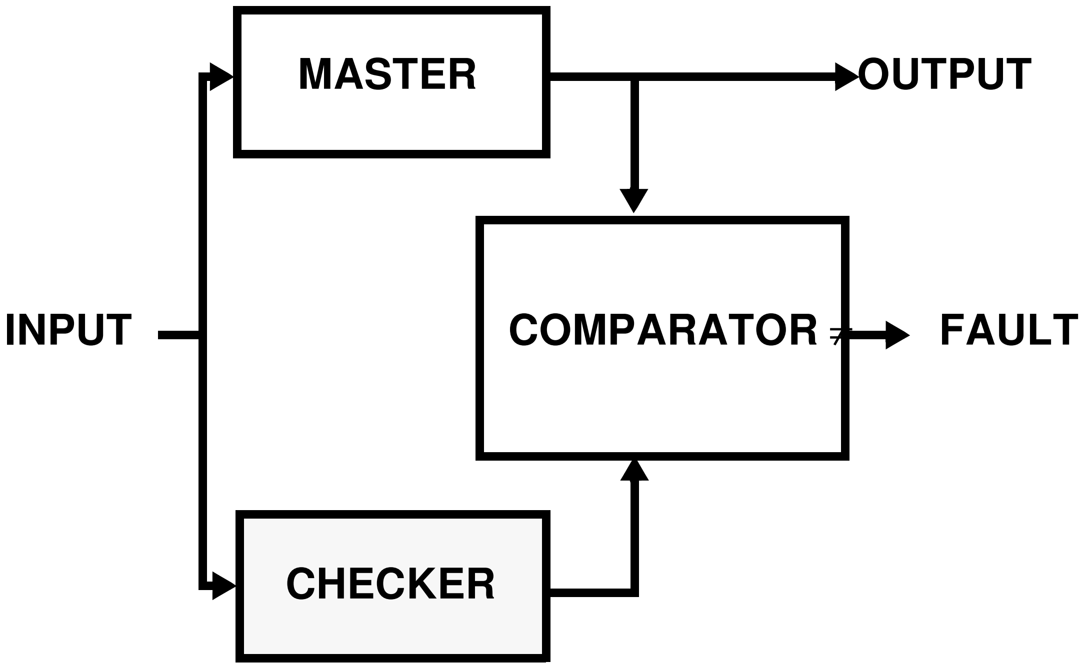
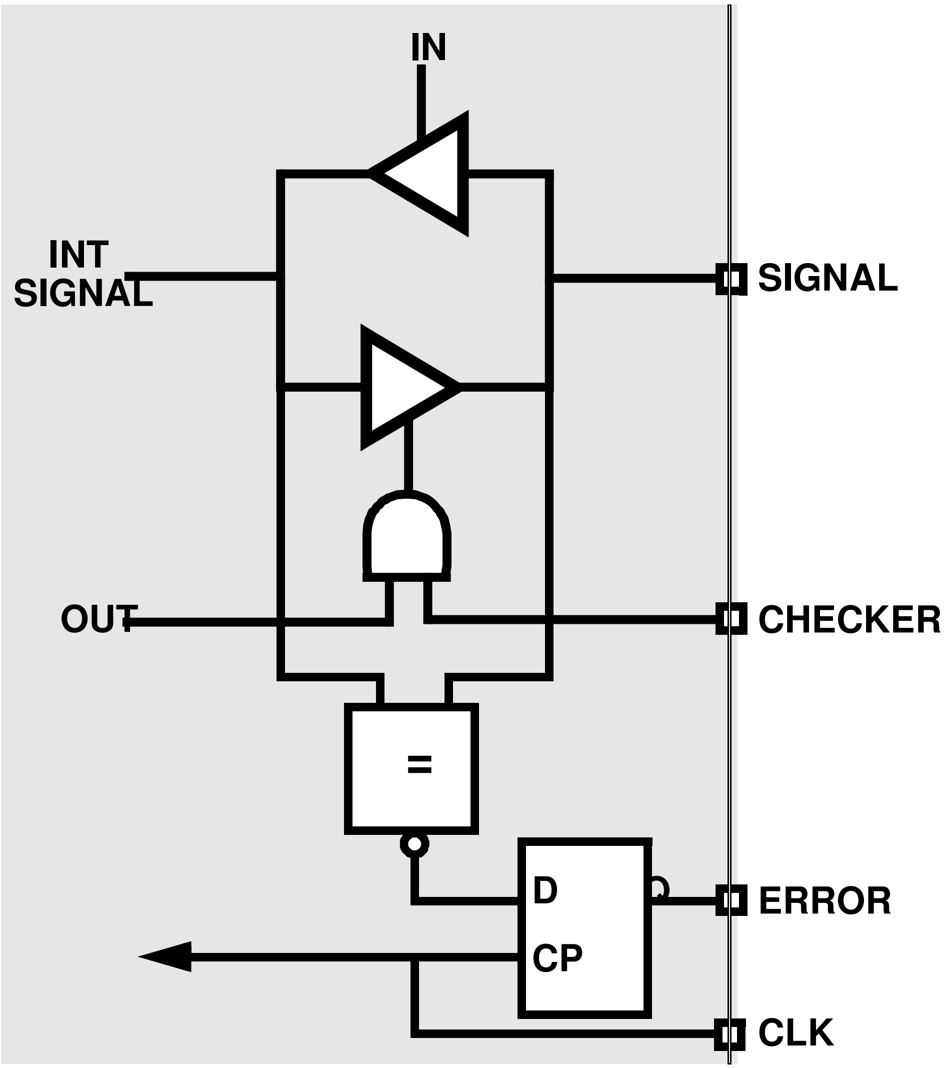
Master-checker setup

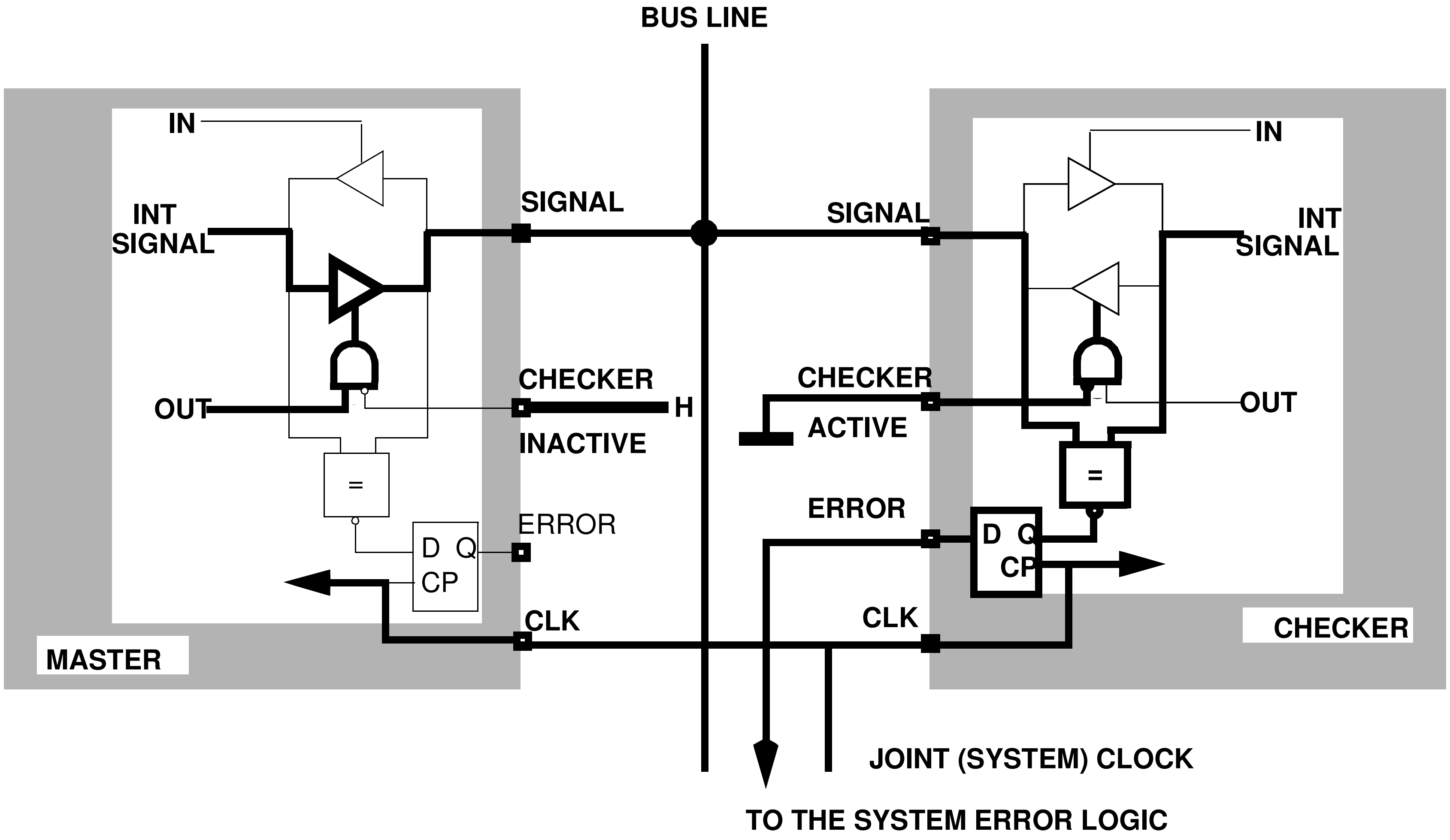


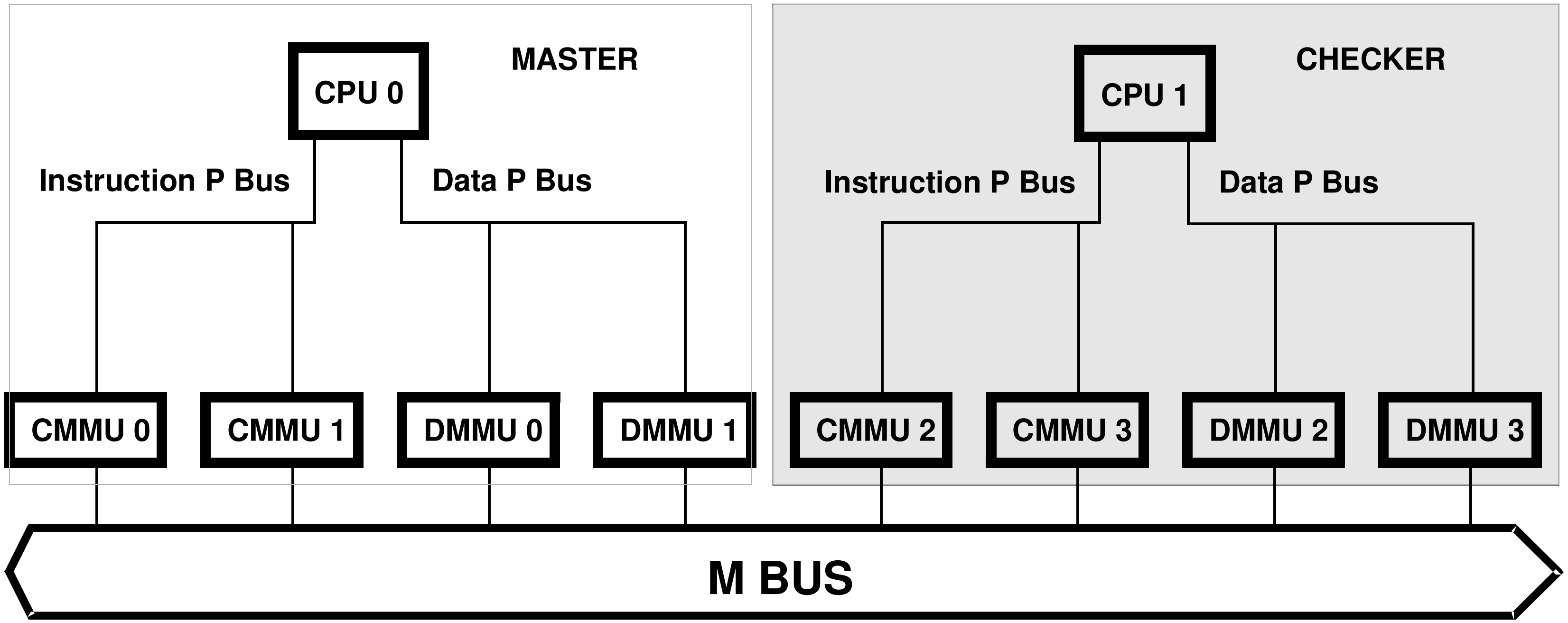
Circuit support



- **Motorola MC88k family, Intel Pentium**

The principle of the M-C mode





- **Two kinds of faults**
  - **Short transient errors (~ 1 clock phase)**
  - **Permanent error bursts (eliminated by RESET only)**

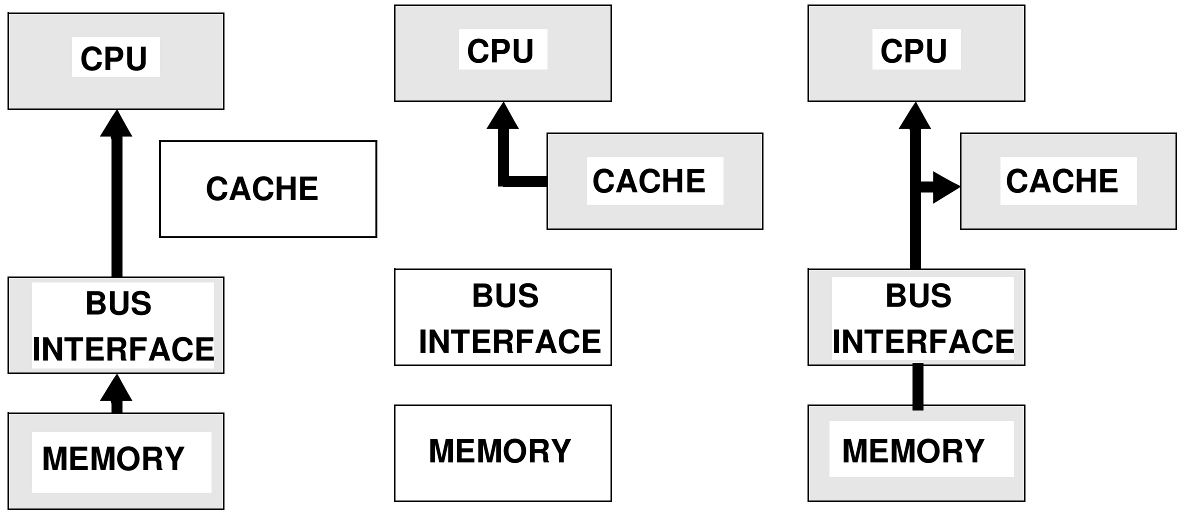
Conflict between performance and dependability

- **CPU-cache communication:**
  - **Speed: maximal (ca. 1 clock cycles/transfer)**
  - **Dependability:**
    - **Few of components**
    - **Unprotected**
    - **No observability without external communication by master-checker $\Rightarrow$ potentially long latency**
- **CPU-memory communication:**
  - **Reduced speed($\geq$ 2 clock cycles/transfer)**
  - **Reliability:**
    - **Large number of components**
    - **Parity protected**
    - **Observability by master-checker $\Rightarrow$ short latency possible**

Difference between P- and M-bus master-checker mode

- **P-bus:**
  - **Every instruction is checked**
  - **Every memory cycle is checked independently of the presence of the data in the cache or in the memory**
- **M-bus:**
  - **Only external bus cycles are checked, but transfers from cache NOT!**
  - **Only M-bus signals driven by the MMU are checked**

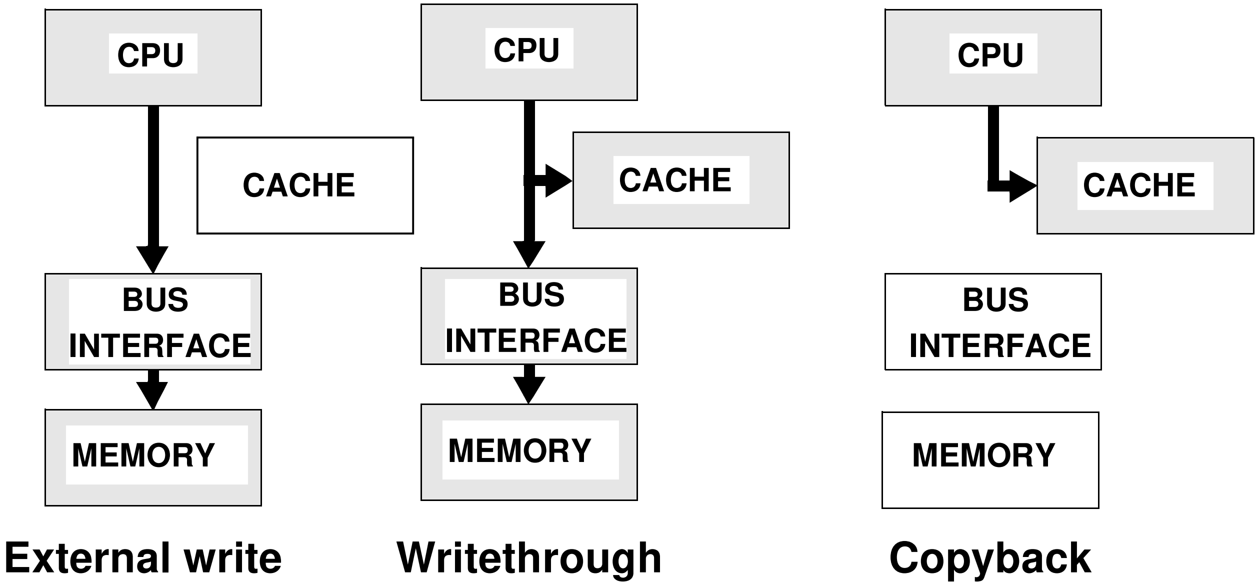
Read operation



No cache **Cache read** **Cache update**

- **Address and control lines are checked during external read and update operations**

Write operation

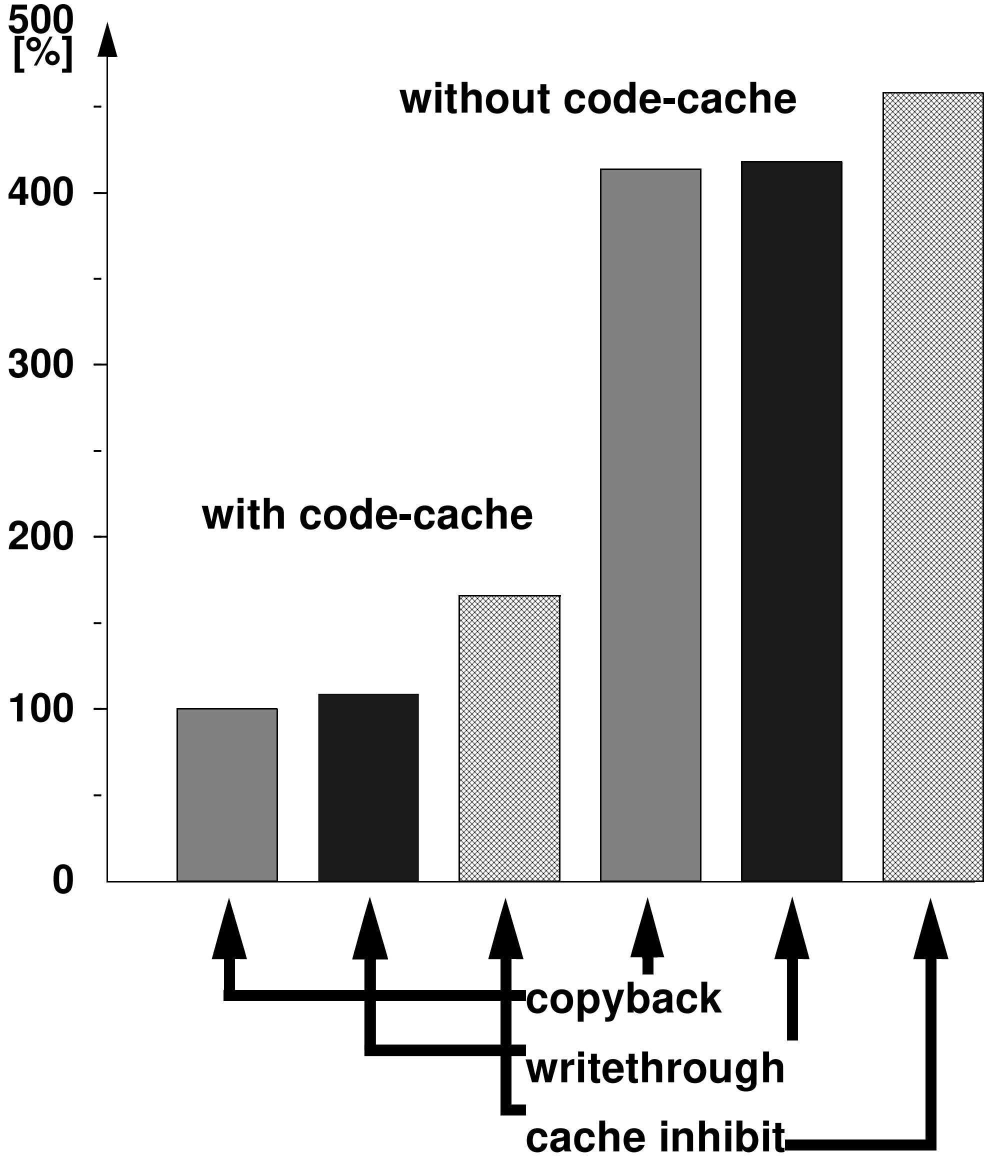


- **M-bus write: data, addresses and control signals checked**

When are M-bus write operations performed?

- **No cache: every read and write cycle is an M-bus operation**
  - **Maximal dependability by maximal observability of write operations**
  - **Due to frequent memory cycles:**
    - **Reduced dependability**
    - **Maximal overhead in time**
- **Writethrough: Every write cycle is an M-bus operation, but reads are dominantly cache related**
  - **Maximal dependability by maximal observability of write operations**
  - **Read operations unchecked**
    - **Moderate overhead in time**
- **Copyback: only exceptional M-bus cycles (cache flush, update)**
  - **Minima overhead in time**
  - **Minimal observability -> potentially long latency!**

Global cache management policies



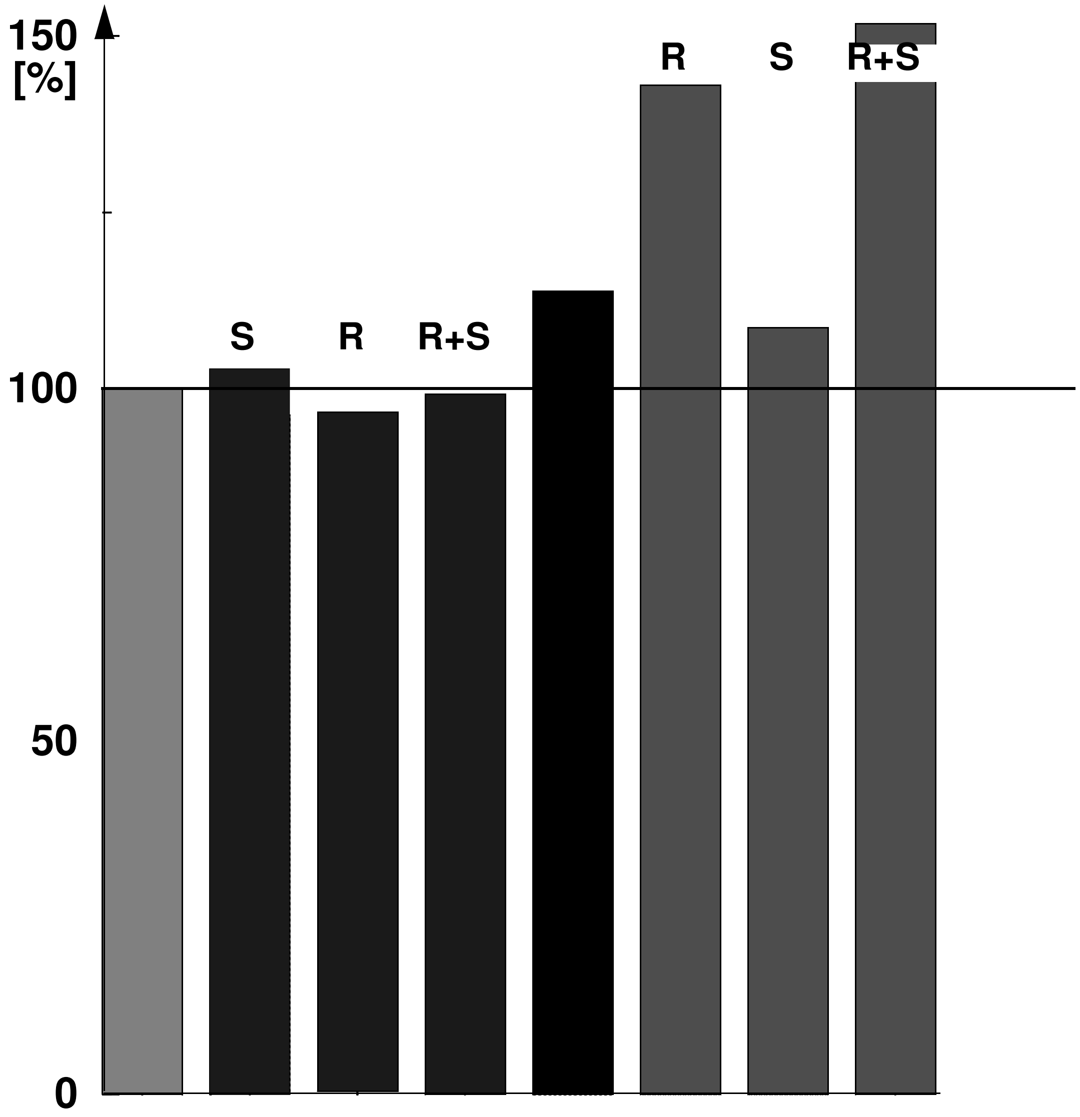
Evaluation

- Exhaustive checking of the control flow is non-reasonable
- Increased observability of data ⇒ reduced performance
- Selective strategy for increasing the observability:
- Low overhead in time ⇒ only the most important data undergo a check
  - *Stack:* Control flow check at procedure calls (return addresses)
  - Recovery data*:*
    - “Quintessence of the results”
    - Avoidance of erroneous recovery points
  - Other, application dependent data
- Non implemented in standard UNIX...

Expectations by a benchmark

- Memory allocation in a multigrid algorithm
  - Stack: <1% (large frequency of use)
  - Recovery data: ca. 60%
- predicted time overhead:
  - cc. 60%-70% of the time overhead of the global writethrough strategy
  - cc 10% of the total run time in the case of copyback-strategy

Measurement results: selective cache management policy



	Stack	Recovery	Andere
	copyback	copyback	copyback
	copyback	writethrough	copyback
	writethrough	copyback	copyback
	writethrough	writethrough	copyback
	writethrough	writethrough	cache inhibit
	copyback	cache inhibit	copyback
	cache inhibit	copyback	copyback
	cache inhibit	cache inhibit	copyback

Reason of the paradox

- **The values of recovery variables are depend of many variables, inc. themselves ⇒**
  - **many accesses to different data between write operations**
  - **The variable under estimation will be out- and backloaded (Last recently used cache update policy)**
  - **Granularity of a writeback is a line (4 words)**
    - **At copyback the entire line is written back**
    - **At writethrough no activity (valid copy already in the memory)**
- **Scalability?**

“Pseudo Watchdog”

- **Instead of sending the assigned signatures to a WDP,**
  - write to a location in the memory (writethrough)
  - automatic check on the M bus
  - Signatures: subroutine call
    (return address written onto the stack)

<table>
<tr><th colspan="2" rowspan="2">Fault</th><th colspan="3">Fault coverage in%</th></tr>
<tr><th>MC</th><th>WD</th><th>MC+WD</th></tr>
<tr><td rowspan="2">CPU-internal</td><td>Control fault</td><td rowspan="5">100</td><td>80</td><td rowspan="5">100</td></tr>
<tr><td>Data fault</td><td>0</td></tr>
<tr><td colspan="2">Fault in code cache</td><td rowspan="2">50-80</td></tr>
<tr><td rowspan="2">Fault in data cache</td><td>Control flow variable</td></tr>
<tr><td>Data variable</td><td>0</td></tr>
<tr><td rowspan="2">External fault</td><td>Control variable</td><td>0</td><td>50-80</td><td>50-80</td></tr>
<tr><td>Data variable</td><td>0</td><td>0</td><td>0</td></tr>
</table>