On-Line Software Integrity Checking
Software Implemented Hardware Fault Tolerance

Stefano Di Carlo, Politecnico di Torino, Italy

RE-TRUST:
Remote EnTrusting by RUn-time Software auThentication

“Kick-off” Workshop - Centro Congressi Panorama (TN)
18th,19th September 2006

Outline

- Introduction
- Code integrity checking
- Contribution to Re-Trust
- Conclusions

Outline

- **Introduction**
- Code integrity checking
- Contribution to Re-Trust
- Conclusions

Context

- Computer-based systems pervade all areas of our lives:
  - Automotive systems
  - Aircrafts
  - Trains
  - Medical systems
  - Common house land applications

Context (Cont'd)

- In many situations computers play a key role in critical tasks with respect to human safety and data security

High reliability is a must

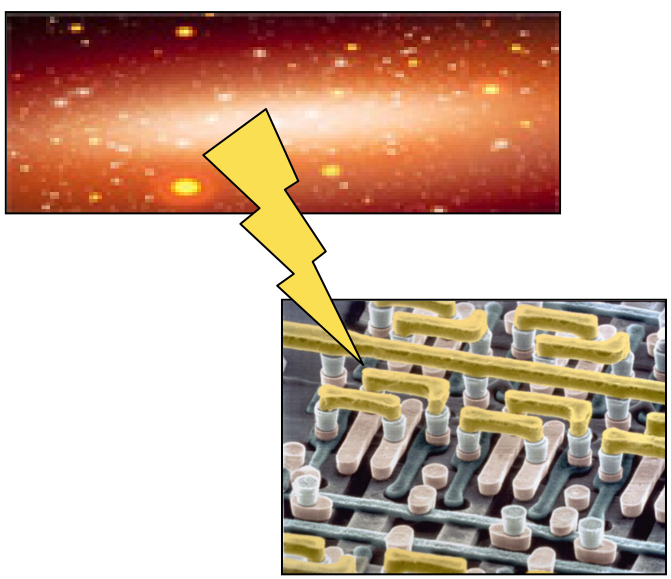
Soft-Errors

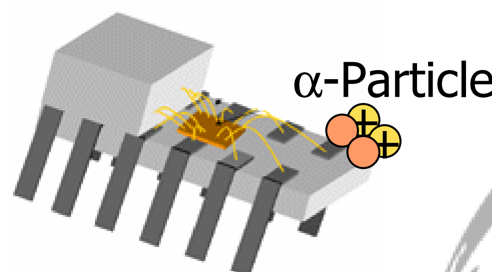
- Soft-errors pose a major challenge for the reliability of computer-based systems
- Soft-errors are transient faults that occur in integrated circuits due to environmental stresses

Soft-Errors (Cont'd)

Two common causes:

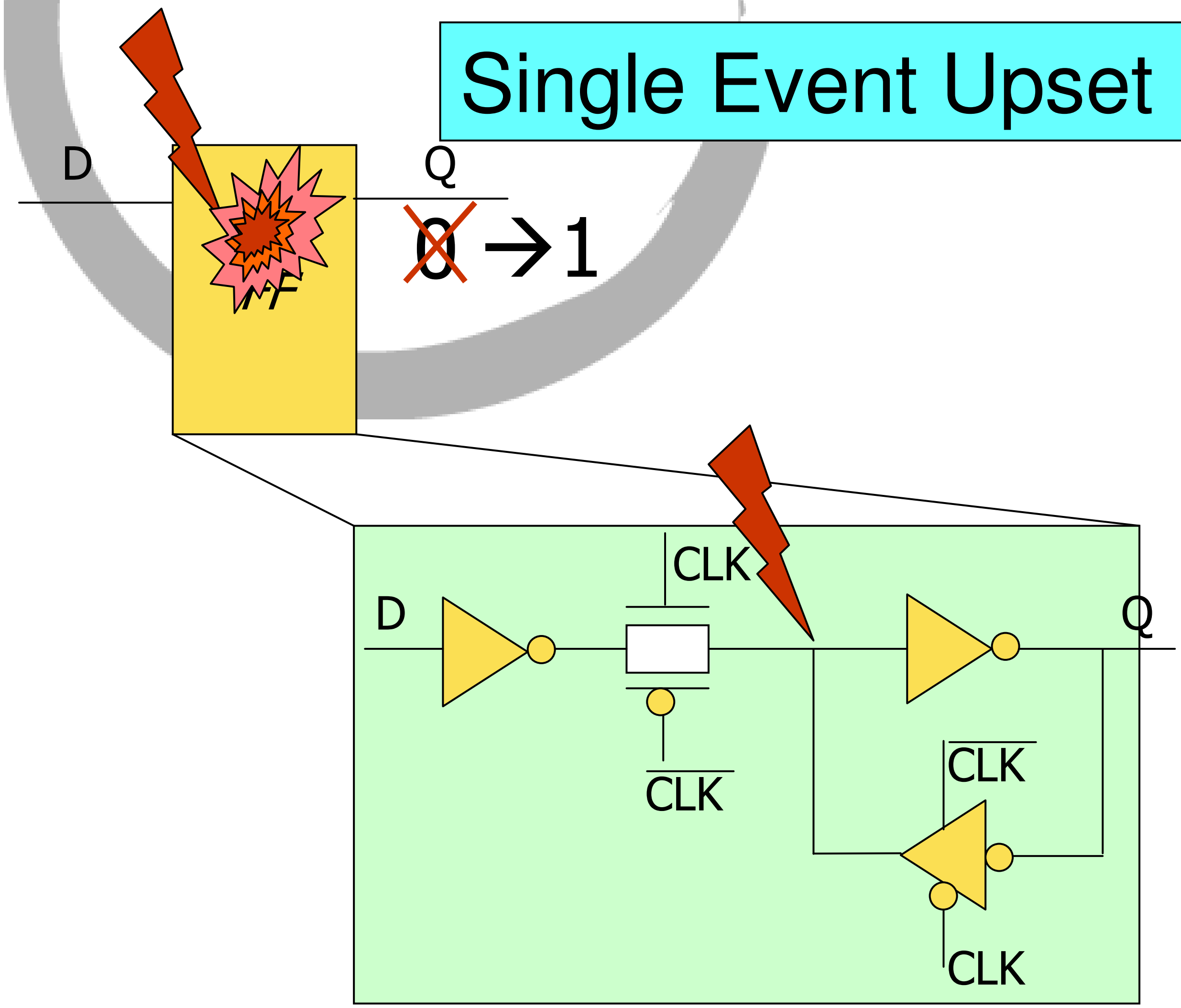
- **Neutron radiations** from cosmic rays
- **Alpha particles** from packaging materials





Soft-Errors (Cont'd)

Single Event Upset (SEU)



Soft-Errors Effects

- How soft-errors manifest themselves:
  - Modification of the code of the software running on the system
  - Corruption of data processed by the software

Soft-Errors Effects (Cont'd)

- Three possible effects on the system:
  - No Effect
  - Crashes
  - Fail Silent Violations

Goal

- To guarantee the correct behavior of computer-based systems in presence of soft-errors
- On-Line detection
- Software Implemented Hardware Fault Tolerance (SIHFT)

Outline

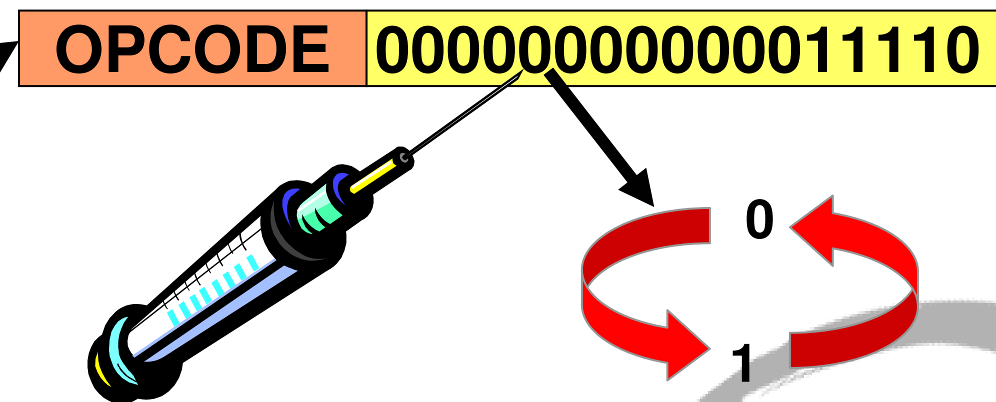
- Introduction
- **Code integrity checking**
- Contribution to Re-Trust
- Conclusions

Control Flow Checking

- A well established approach to check the integrity of a program code
- Monitors that instructions are executed in the correct sequence

Fault Model

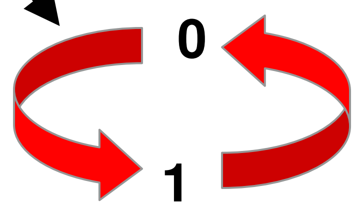
MOV AX,(01)H
ADD AX,BX
INC BX
CP AX,BX
JZ (30)H → **JZ (XX)H**
...
30: SUB AX,(0F)H
...



Fault Model (Cont'd)

MOV AX,(01)H
ADD AX,BX
INC BX
CP AX,BX
JZ (30)H
…
30: SUB AX,(0F)H
...

OPCODE	0000000000001111



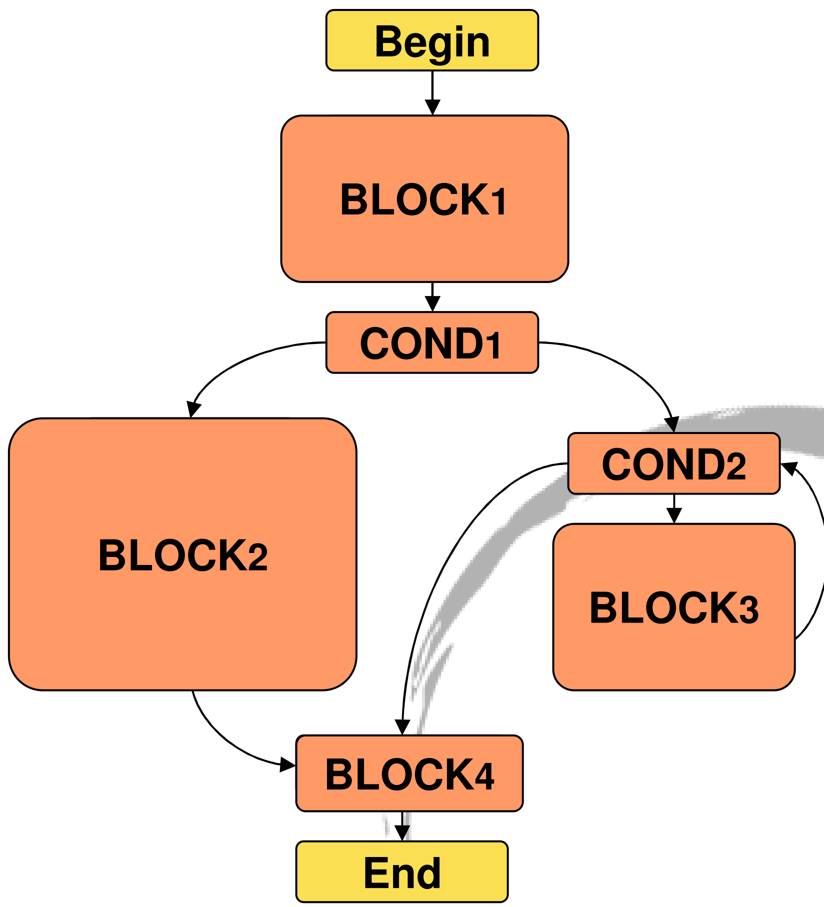
JMP (XX) H
CALL (XX) H
RET
INT X

Control Flow Graph

- Programs represented by a **Control Flow Graph**:
  - Nodes represent an instruction
  - Edges represent a valid sequence of two instructions
- Groups of consecutives sequential nodes grouped in **Branch Free Blocks**

Control Flow Graph (Cont'd)

```
void main ()
{
instr1;
instr2;
if (cond1) {
        instr4;
        instr5;
        instr6; } else {
        while (cond2) {
                instr8;
                instr9;}
                }
        instr10;
}
```

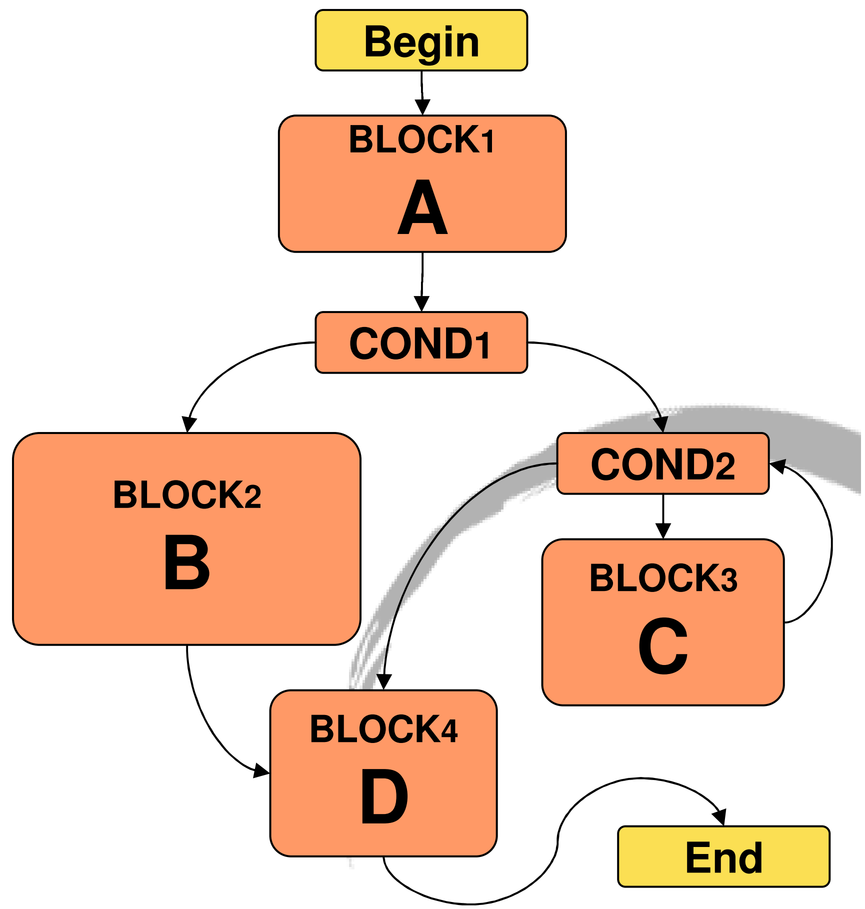


Signature via Regular Expressions

- Each branch-free block in the graph labeled and signed with a **block symbol**
- Each valid program execution identified by a **control string** composed of block symbols

Signature via Regular Expressions (Cont'd)

- "ACCCD" is a valid execution
- "ABCD" is not a valid execution



Signature via Regular Expressions (Cont'd)

- The set of possible control strings generates a language L = (A , R)
  - A is the input alphabet composed by all the block symbols
  - R is a regular expression able to generate the valid strings

Signature via Regular Expressions (Cont'd)

- The language L is a signature for the program control flow
- A program execution is allowed if the related control string belongs to the language L

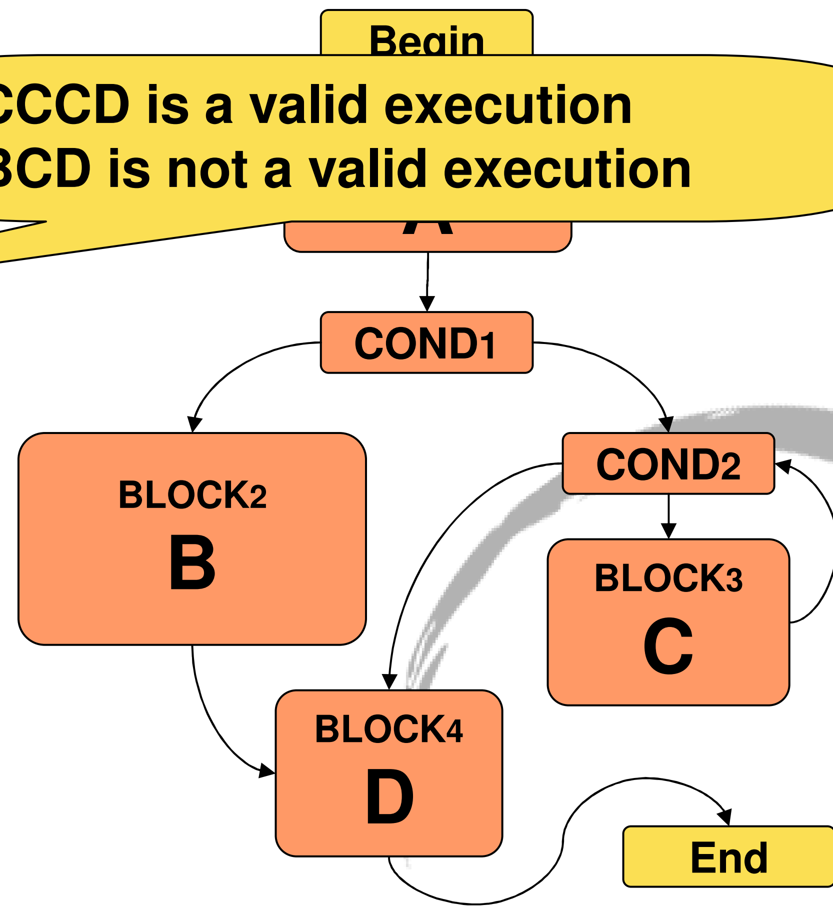
Signature via Regular Expressions (Cont'd)

L = (A,R)

A = ('A','B','C','D')

R = A(B|(C)*)D

ACCCD is a valid execution
ABCD is not a valid execution



Implementation

- Off-Line program analysis
- Control graph and control strings computation
- Code instrumentation

Implementation (Cont'd)

- Original code split into two processes:
  - Modified Original Program
  - Checker
- Two processes communicating via IPC mechanisms

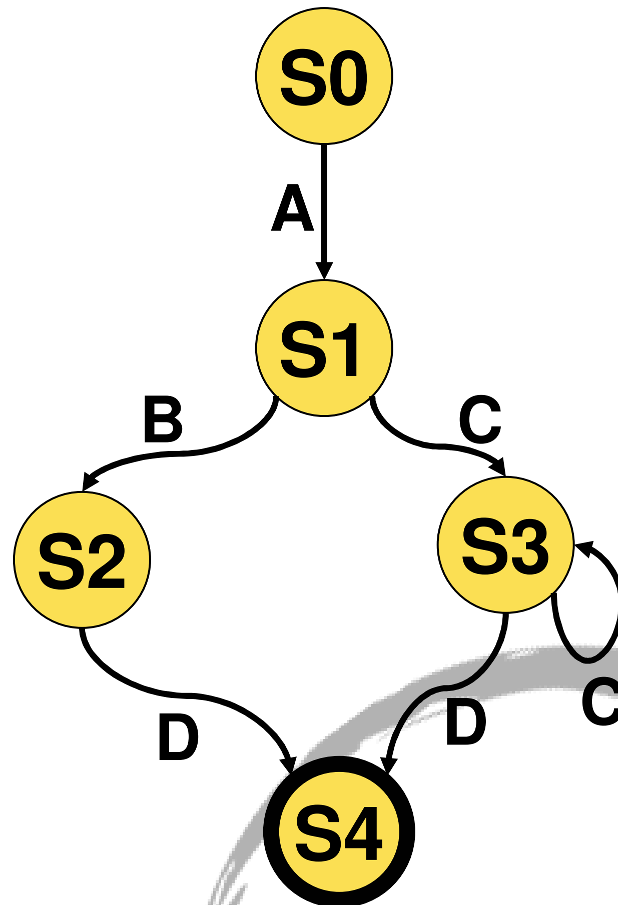
CODE INTEGRITY CHECKING

Modified Original Program

```
void main ()
{
int p[2];
pipe (p);
instr1;
instr2;
write (p,'A');
if (cond1) {
  instr4;
  instr5;
  instr6;
  write (p,'B');
} else {
  while (cond2) {
   instr8;
   instr9;
   write (p,'C');
  }
}
instr10;
write (p,'D');
}
```

IPC

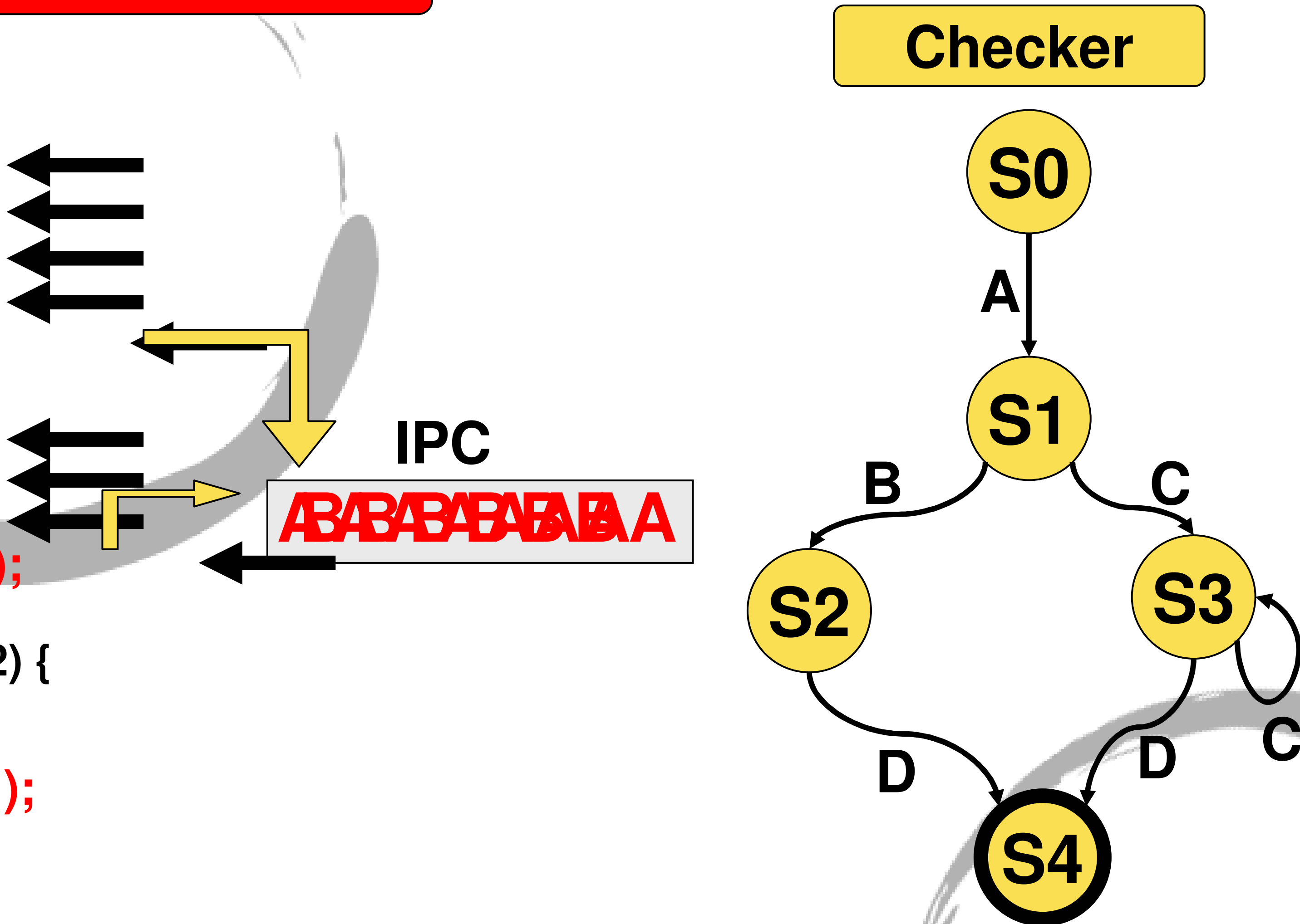
Checker



CODE INTEGRITY CHECKING

Modified Original Program

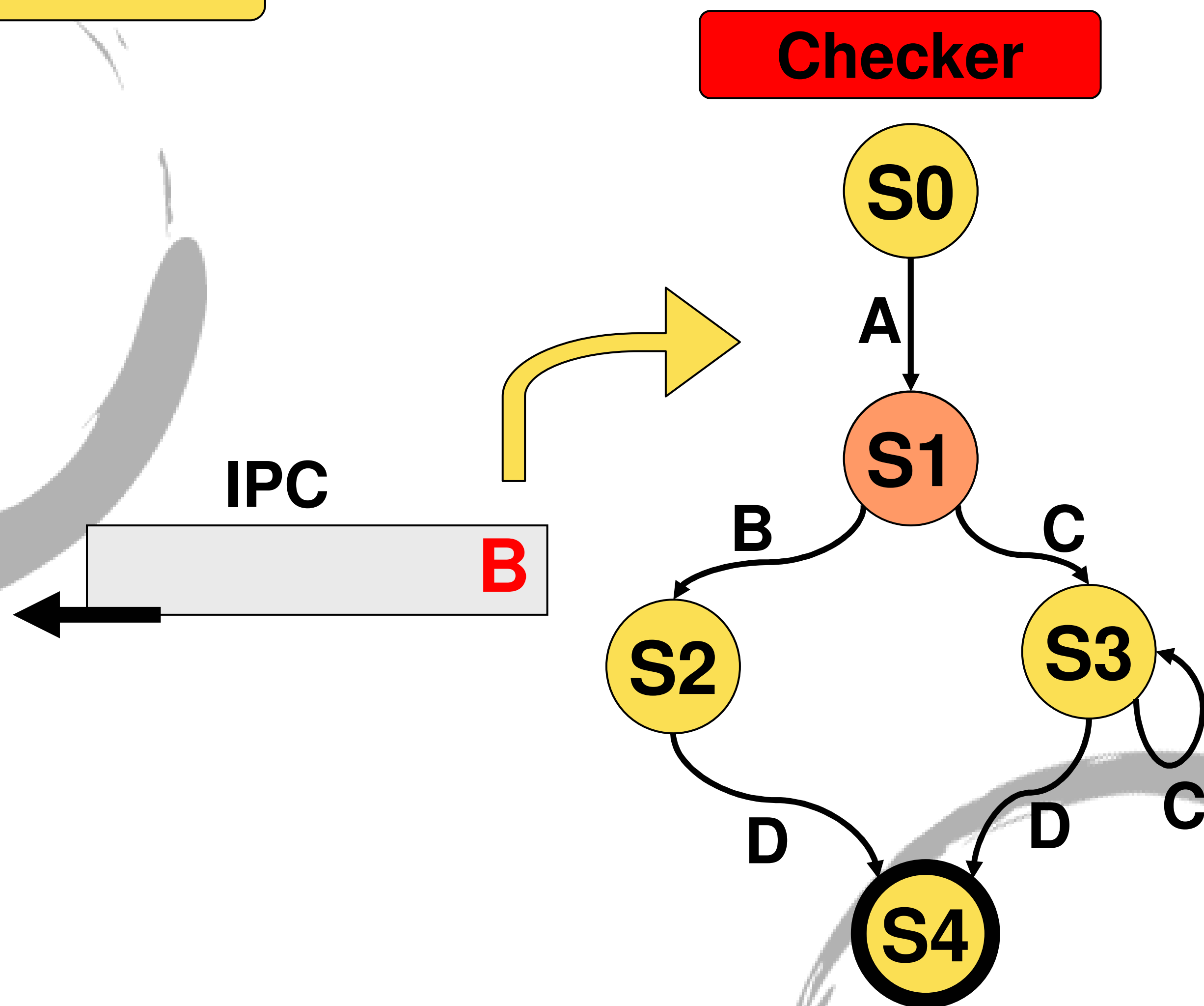
```
void main ()
{
int p[2];
pipe (p);
instr1;
instr2;
write (p,'A');
if (cond1) {
   instr4;
   instr5;
   instr6;
   write (p,'B');
} else {
   while (cond2) {
    instr8;
    instr9;
    write (p,'C');
   }
}
instr10;
write (p,'D');
}
```



CODE INTEGRITY CHECKING

Modified Original Program

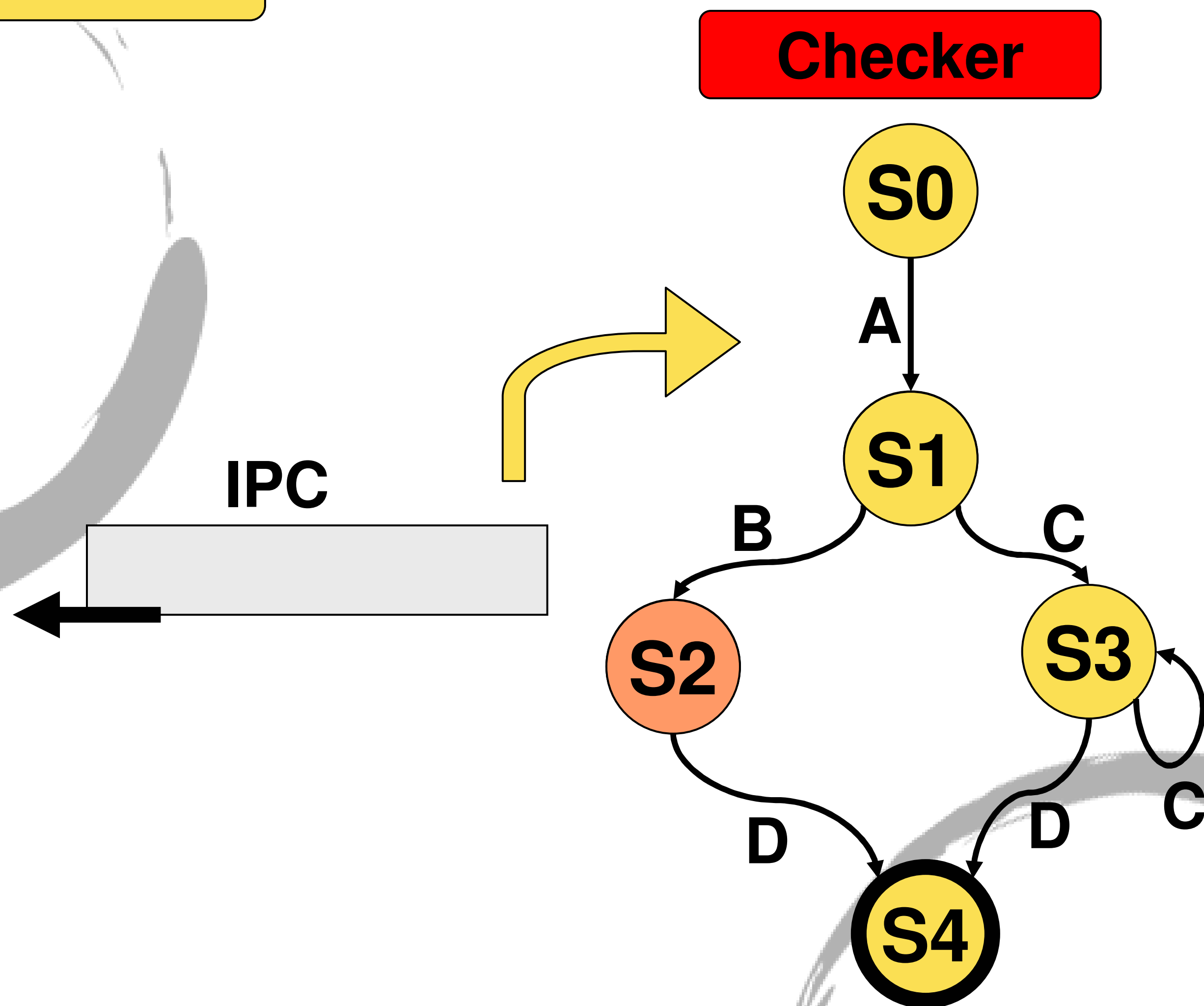
```
void main ()
{
int p[2];
pipe (p);
instr1;
instr2;
write (p,'A');
if (cond1) {
  instr4;
  instr5;
  instr6;
  write (p,'B');
} else {
  while (cond2) {
   instr8;
   instr9;
   write (p,'C');
  }
}
instr10;
write (p,'D');
}
```



CODE INTEGRITY CHECKING

Modified Original Program

```
void main ()
{
int p[2];
pipe (p);
instr1;
instr2;
write (p,’A’);
if (cond1) {
  instr4;
  instr5;
  instr6;
  write (p,’B’);
} else {
  while (cond2) {
   instr8;
   instr9;
   write (p,’C’);
  }
}
instr10;
write (p,’D’);
}
```



CODE INTEGRITY CHECKING

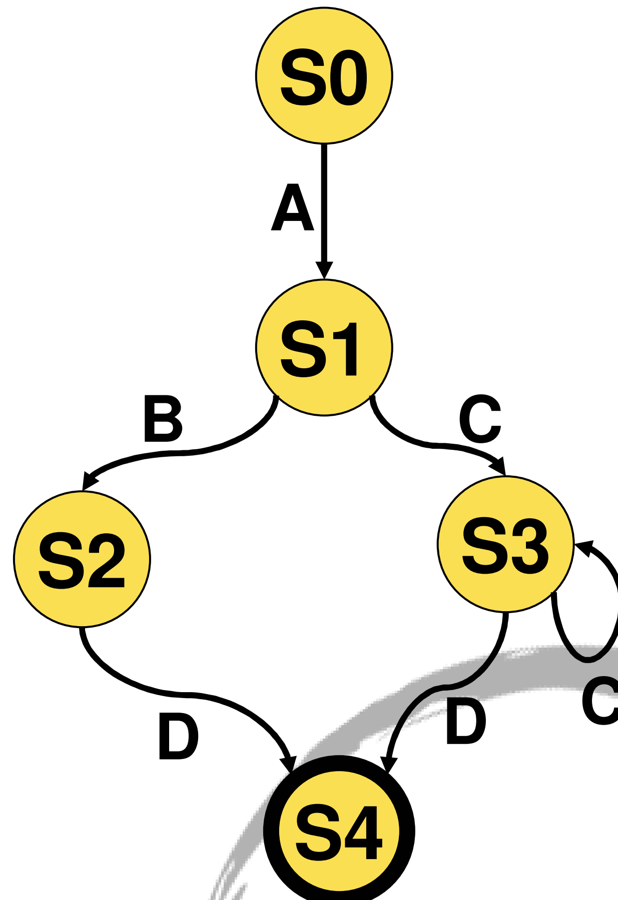
Modified Original Program

```
void main ()
{
int p[2];
pipe (p);
instr1;
instr2;
write (p,'A');
if (cond1) {
  instr4;
  instr5;
  instr6;
  write (p,'B');
} else {
  while (cond2) {
   instr8;
   instr9;
   write (p,'C');
  }
}
instr10;
write (p,'D');
}
```

IPC

DDDDDDD

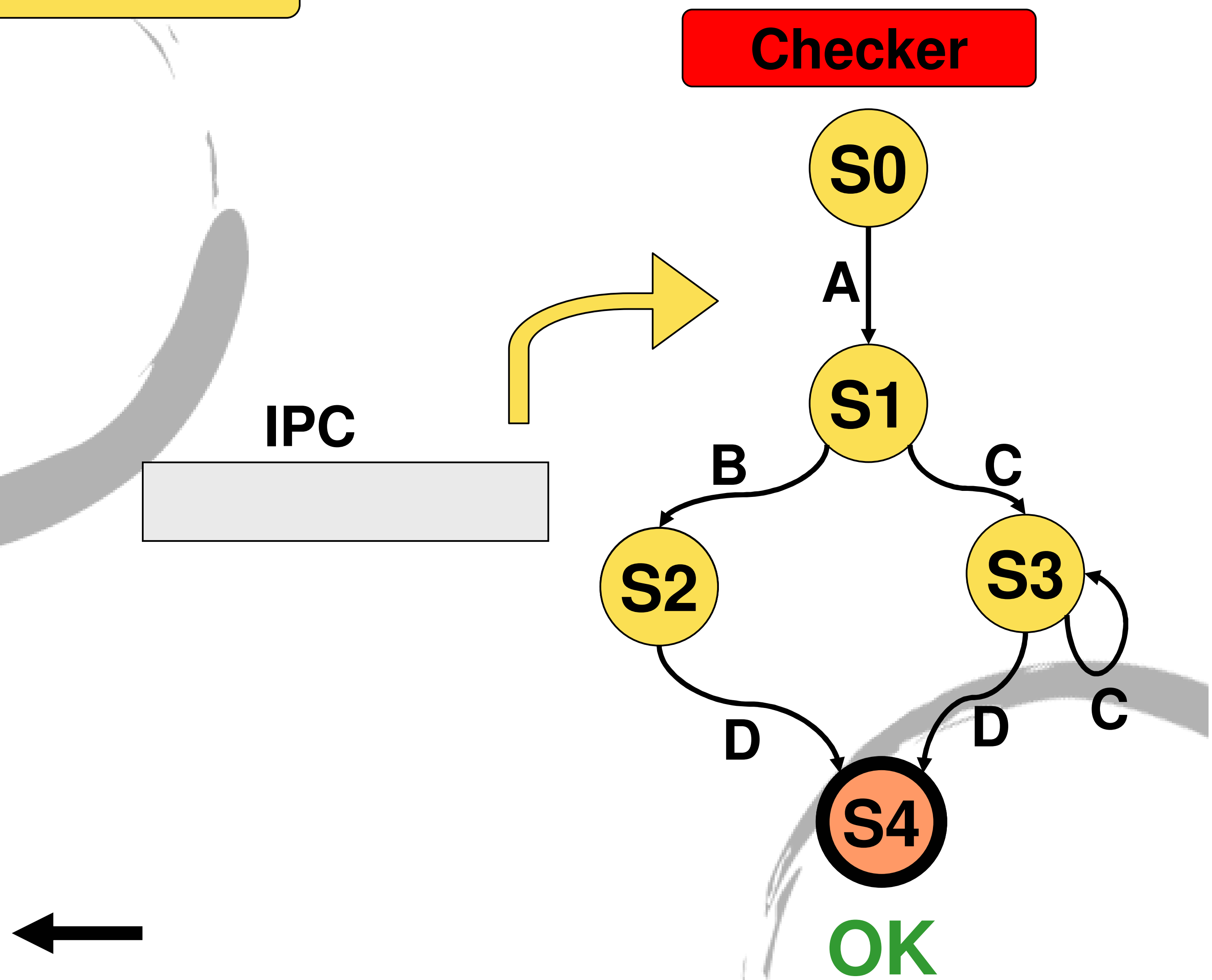
Checker



CODE INTEGRITY CHECKING

Modified Original Program

```
void main ()
{
int p[2];
pipe (p);
instr1;
instr2;
write (p,'A');
if (cond1) {
   instr4;
   instr5;
   instr6;
   write (p,'B');
} else {
   while (cond2) {
    instr8;
    instr9;
    write (p,'C');
   }
}
instr10;
write (p,'D');
}
```



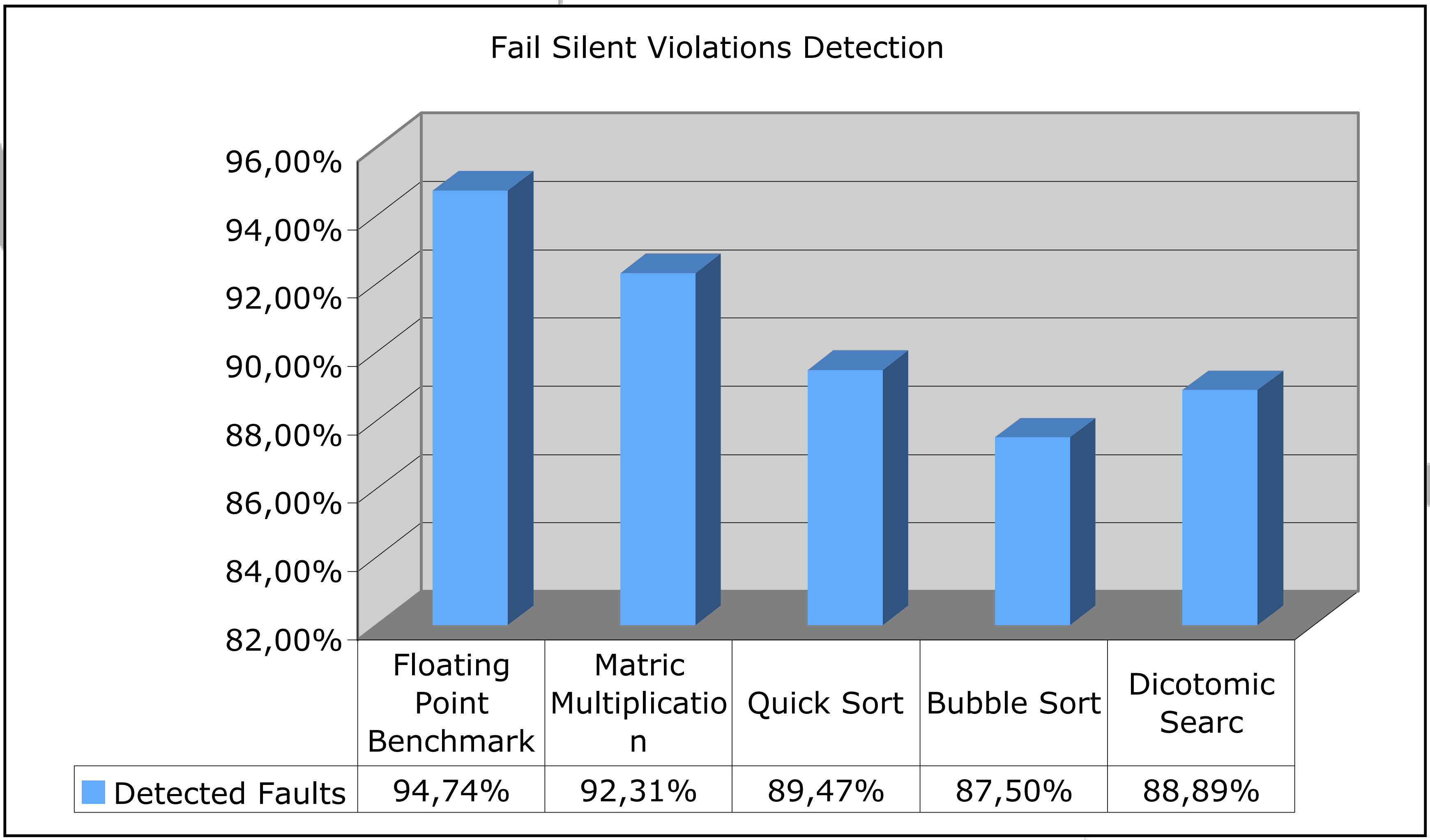
Experimental Results

- Source to Source compiler implemented to automate the code instrumentation
  - Working on C/C++ applications
  - Dealing with both Linux and Windows NT/2000 Operating Systems

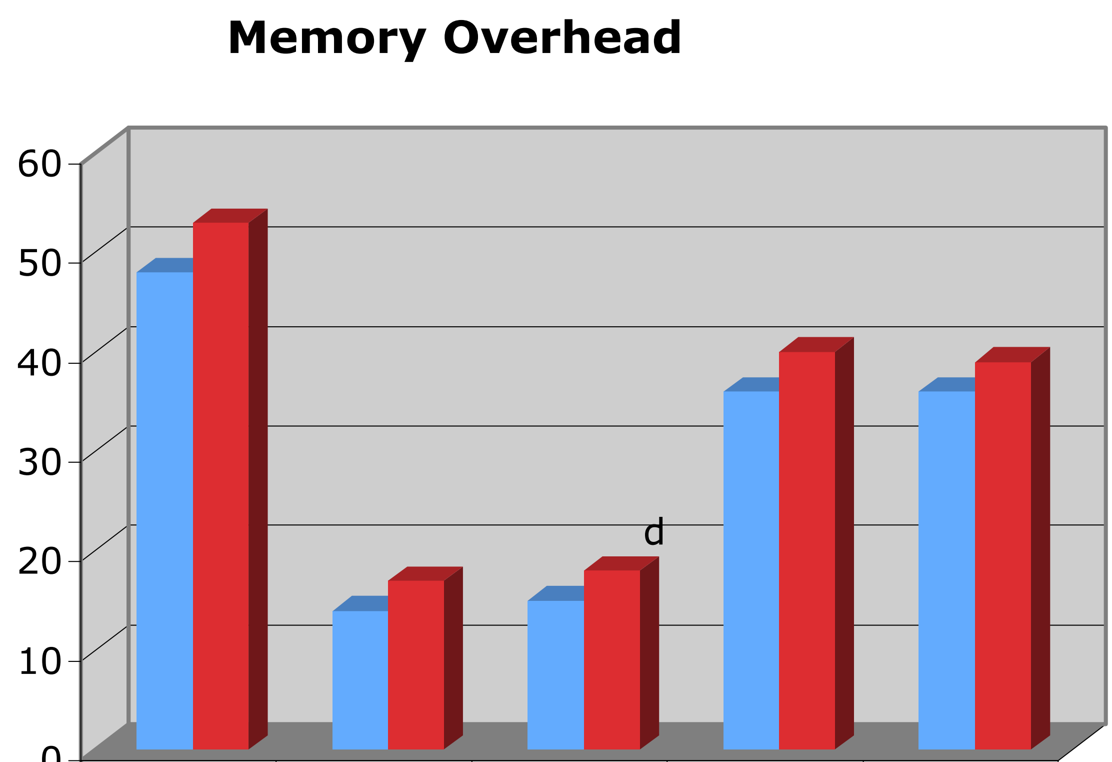
Experimental Results (Cont'd)

- Fault Injection experiments on 5 different benchmarks:
  - Floating Point Benchmark
  - Matrix Multiplication Benchmark
  - Quick Sort
  - Bubble Sort
  - Binary Search
- 10'000 faults injected fro each application

Experimental Results (Cont'd)



Experimental Results (Cont'd)



	Floating Point	Matric Multiplica	Quick Sort	Bubble Sort	Dicotomic Searc
Original (KB)	48	14	15	36	36
Modified (KB)	53	17	18	40	39

Experimental Results (Cont'd)

- A demo board running Windows NT with instrumented software is working as an experiment on the International Space Station GALILEO

Open problems

- Choosing the right symbols
- Large applications
- DLL and dynamic libraries
- How to protect the Operating System
- ….

Outline

- Introduction
- Code integrity checking
- **Contribution to Re-Trust**
- Conclusions

Where in RE-Trust?

- WP2: Software-based Tamper Resistance Methods for Remote Entrusting
  - Task2.2 – Secure interlocking and authenticity checking

Contribution to the Project

- On-Line software integrity checking/protection techniques:
  - Code protection
    - Control flow checking techniques

      [1] Control-flow checking via regular expressions Benso, A.; Di Carlo, S.; Di Natale, G.; Prinetto, P.; Tagliaferri, L.; Test Symposium, 2001. Proceedings. 10th Asian 19-21 Nov. 2001, Kyoto (Japan), Page(s):299 - 303

Contribution to the Project

(Cont'd)

- Data protection
  - PROMON: Programming by contract/Assertion
    [2] *PROMON: a profile monitor of software applications* Benso, A:; Di Carlo, S.; Di Natale, G.; Prinetto, P.; Tagliaferri, L.; Tibaldi, C.Design and Diagnostics of Electronic Circuits and Systems 2005. DDECS 2005. 8th IEEE International Workshop on13-16 April 2005 Page(s): 81 - 86
  - RECCO: Code reordering, variable duplication
    [3] SEU effect analysis in a open-source router via a distributed fault injection environment Benso, A.; Di Carlo, S.; Di Natale, G.; Prinetto, P. Design, Automation and Test in Europe, 2001. Conference and Exhibition 2001. Proceedings 13-16 March 2001, Munich (G), Page(s):219 – 223

Future works

- Software integrity checking:
  - Move from soft errors to data/code corruption due to malicious attacks
  - Move the code instrumentation into an external module (JVMTI)
  - Use of code mobility and code obfuscation to enhance the security of the integrity checking
  - Move from C/C++ code to Java Language

Outline

- Introduction
- Code integrity checking
- Contribution to Re-Trust
- **Conclusions**

Conclusions

- Are the proposed contributions useful for Re-Trust?
- How to evaluate the effectiveness of the proposed solutions?

Questions?