CENG 523

Adv. Topics of Real-Time Systems

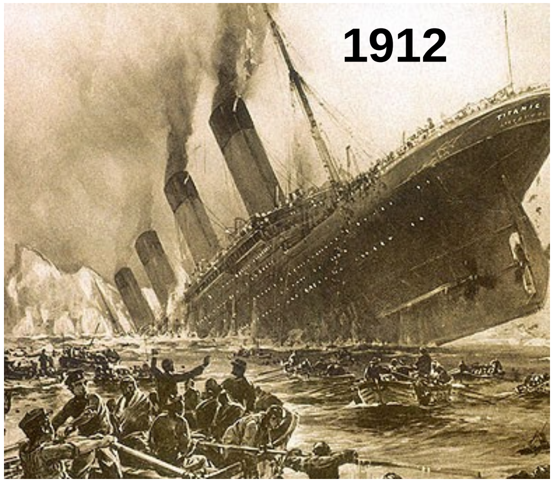
Fault-Tolerance

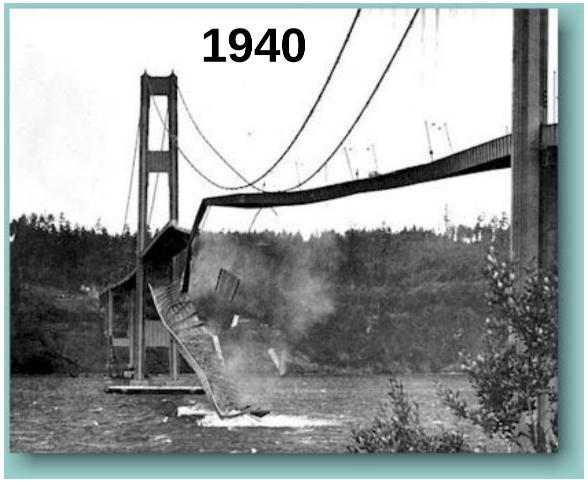
Assoc. Prof. Tolga Ayav, Ph.D.

Department of Computer Engineering

İzmir Institute of Technology

Faults->Errors->Failures





"When a complex system succeeds, that success masks its proximity to failure. . . . Thus, the failure of the *Titanic* contributed much more to the design of safe ocean liners than would have her success. That is the paradox of engineering and design."

Henry Petroski, *Success through Failure: The Paradox of Design*, Princeton U. Press, 2006, p. 95

Fault-Tolerance

- Assuming that the system is functionally correct.
- How do you keep on complying with the specifications in case of faults?
- How do you keep on satisfying real-time properties in case of faults?
- Under what fault assumptions?

Fault-Tolerance

Fault-tolerance can be defined as the ability to comply with the specification in spite of faults. Fault-Tolerance can be classified as:

Hardware Fault-Tolerance

Software Fault-Tolerance

(Software Implemented Hardware Fault-Tolerance)

In all types, fault-tolerance is achieved through redundancy:

Physical/Spatial Fault-Tolerance (Adding extra node)

Temporal Redundancy (Allowing extra time)

Fault-Tolerance Chain

FAULT → ERROR → FAILURE → FAULT → ...

In terms of duration:

Permanent faults

Transient faults

Intermittent faults

Over 80% of faults are transient or intermittent!

Physical Redundancy

(1) Passive Redundancy:

N-Modular Redundancy (NMR) technique is the most common technique among passive hardware redundancy techniques, and is used in this study as well. Two or more replicas of a node are run in parallel and a voter decides about the output of these replicas. 3MR, which is also abbreviated as TMR [10, 12], is the most common passive redundancy technique.

(2) Active Redundancy:

Relies on replacing the faulty node with an identical spare node as soon as a failure is detected [13].

(3) Hybrid Redundancy:

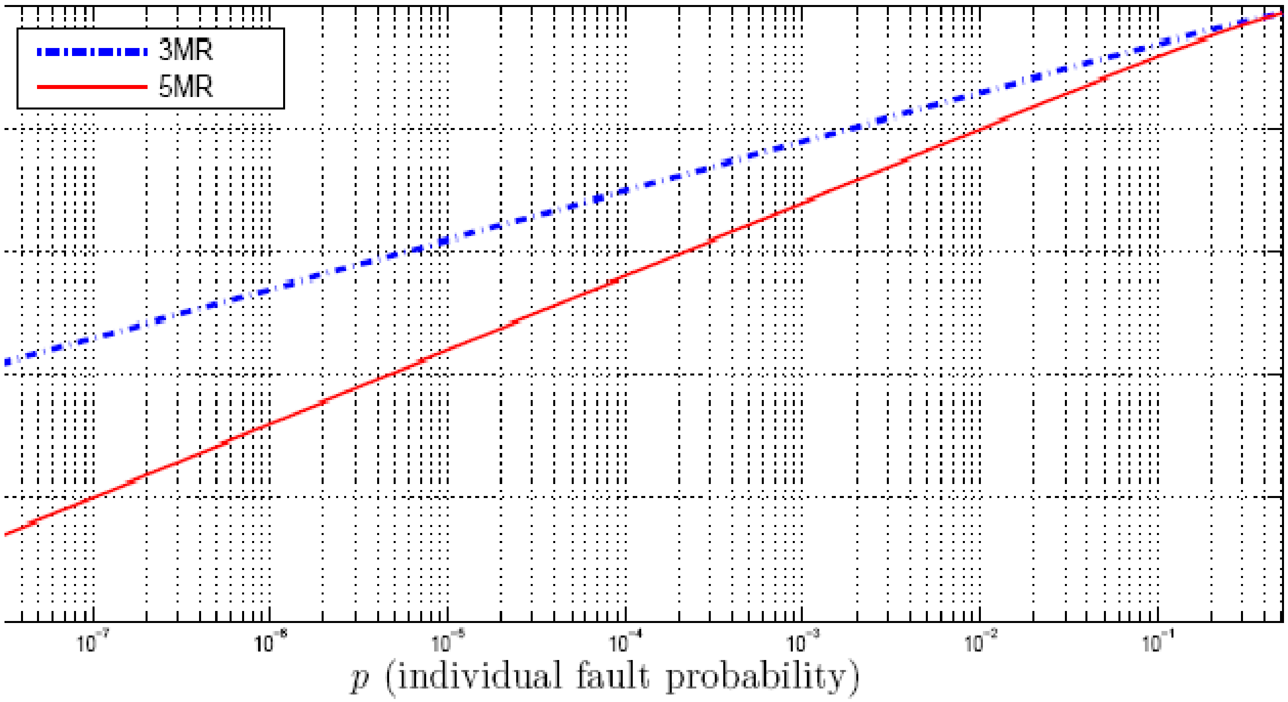
NMR and spare nodes are combined. For instance, a hybrid 3MR+2 (5 nodes) can tolerate three failed nodes whereas 5MR can only tolerate two failures.

N Modular Redundancy

For the NMR technique, if the *instantaneous* fault probability for an individual module is denoted by p, which is also known as SER, then it can be shown that the Probability of Error $P_e(p)$ with the NMR-FT system is given by

$$P_e(p) = \sum_{n=\frac{N+1}{2}}^{N} \binom{N}{n} p^n (1-p)^{N-n},$$

where N is an odd positive integer.



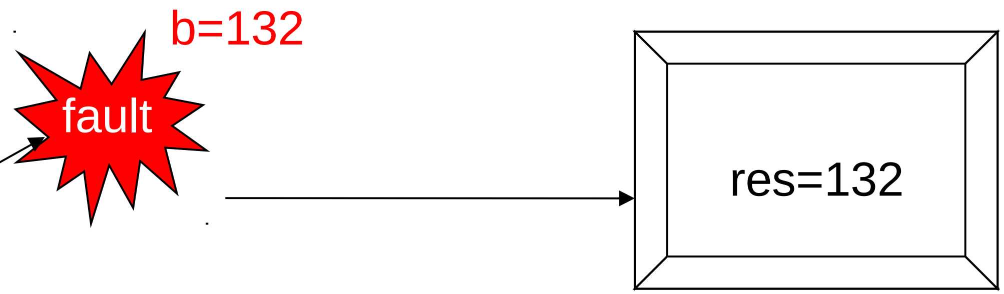
Software Implemented Hardware Fault-Tolerance

- Many techniques such as NMR can be implemented by software:

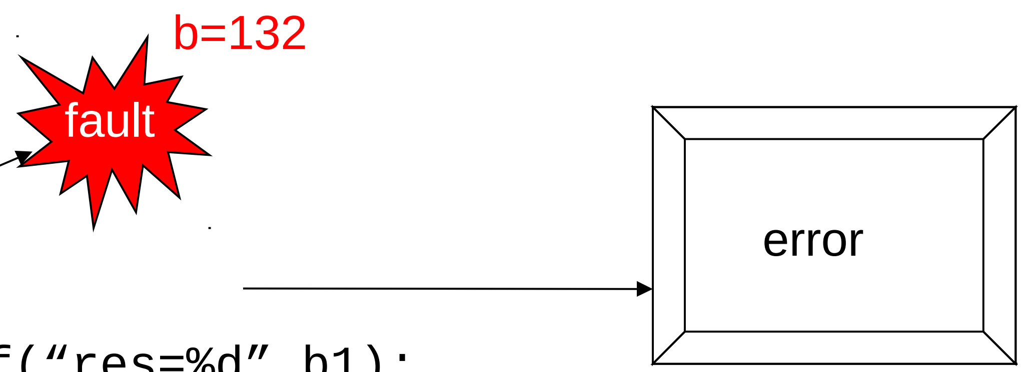
- Double, triple execution
- Repeating execution
- Replicating variables
- Re-sending information on network
- Signature checking
- ...

Double Execution

```
int a=3;
int b;
...

b=a+2;
printf(“res=%d”,b);
```

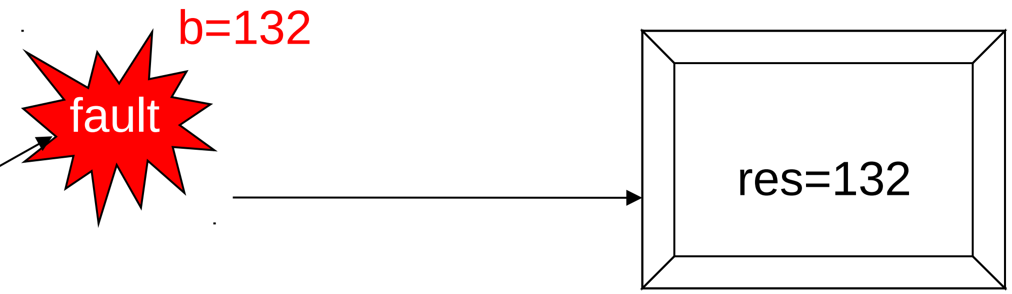



```
int a=3;
int b1,b2;
...

b1=a+2;
b2=a+2;
if(b1==b2)printf(“res=%d”,b1);
else printf(“error”);
```

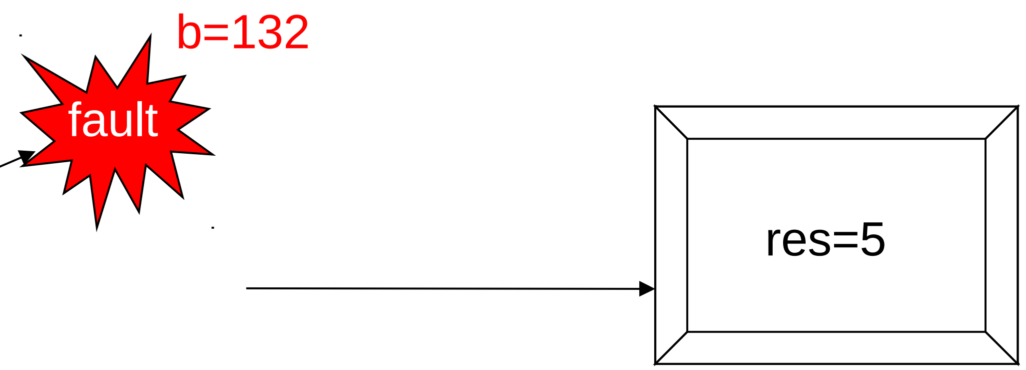


Triple Execution

```
int a=3;
int b;
...

b=a+2;
printf("res=%d",b);
```

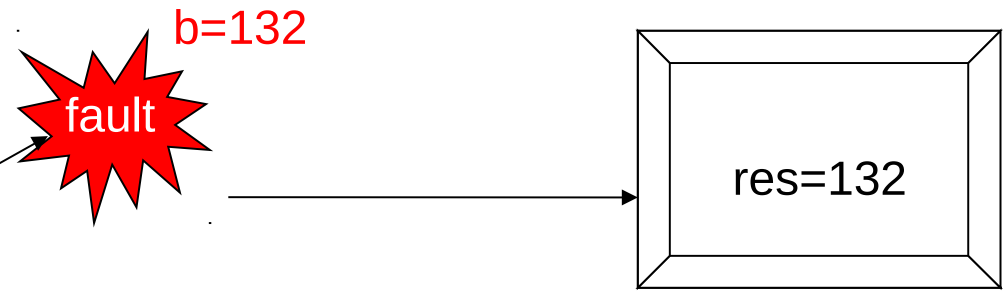



```
int a=3;
int b,b1,b2,b3;
...

b1=a+2;
b2=a+2;
b3=a+2;
if(majority_exists(b1,b2,b3))
{
 b=majority_vote(b1,b2,b3);
 printf("res=%d",b);
}
else printf("error");
```

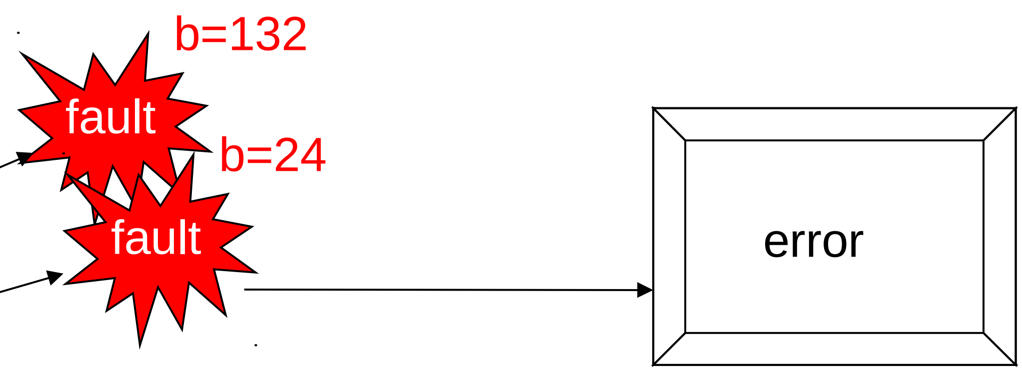


Triple Execution (2)

```
int a=3;
int b;
...

b=a+2;
printf("res=%d",b);
```

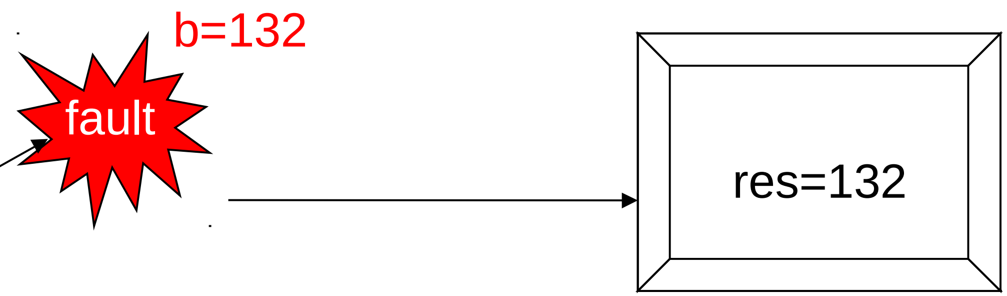



```
int a=3;
int b,b1,b2,b3;
...

b1=a+2;
b2=a+2;
b3=a+2;
if(majority_exists(b1,b2,b3))
{
 b=majority_vote(b1,b2,b3);
 printf("res=%d",b);
}
else printf("error");
```

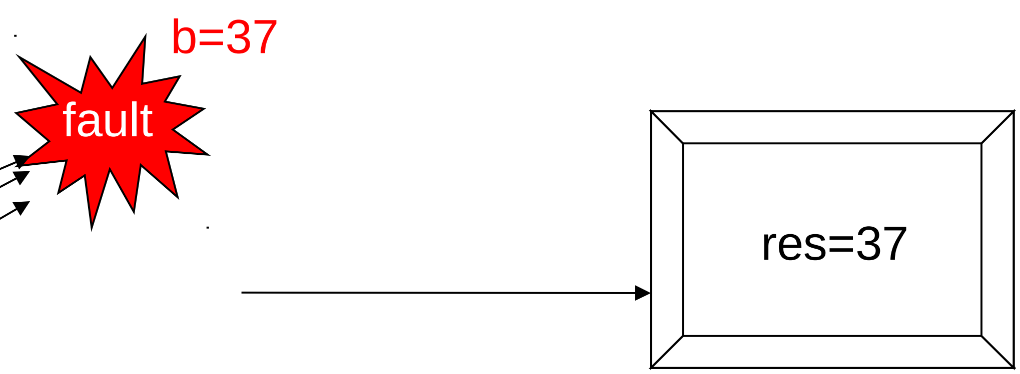


Triple Execution (3) at Stuck-at Fault

```
int a=3;
int b;
...

b=a+2;
printf("res=%d",b);
```

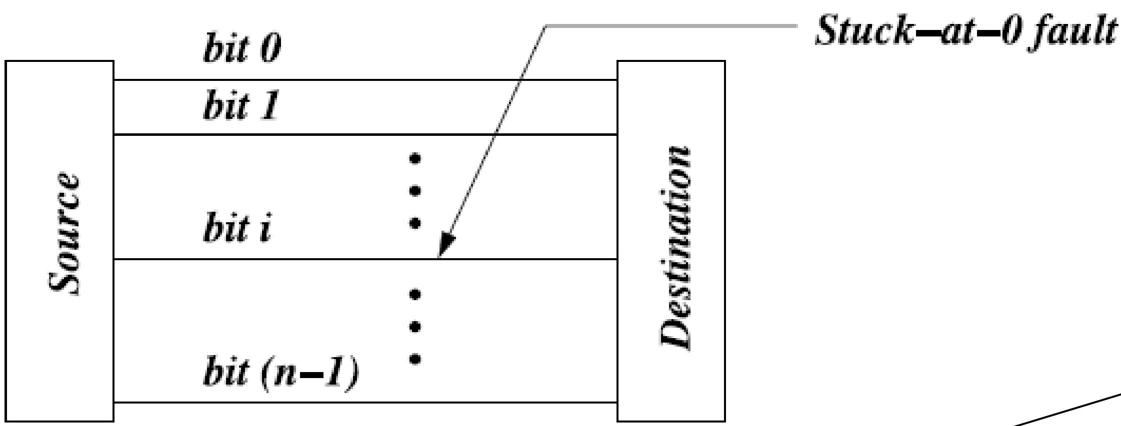



```
int a=3;
int b,b1,b2,b3;
...

b1=a+2;
b2=a+2;
b3=a+2;
if(majority_exists(b1,b2,b3))
{
 b=majority_vote(b1,b2,b3);
 printf("res=%d",b);
}
else printf("error");
```



Stuck-at Fault

- NMR or multiple execution do not help!



By executing these two programs and comparing the results, it is highly possible to detect a stuck-at fault

```
i = 0;
x = 3;
y = 1;
while (i < 5) {
    y = y * (x + i);
    i = i + 2;
}
z = y;
```

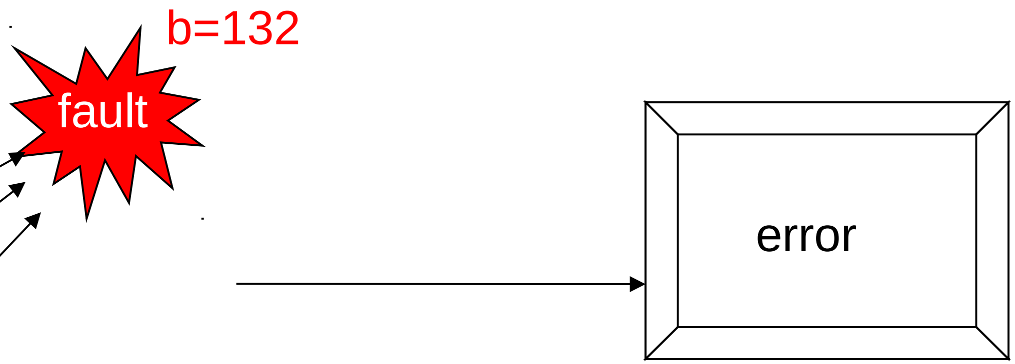
(a) The original program

```
i = 0;
x = 6;
y = 2;
while (i < 10) {
    y = y * (x + i)/2;
    i = i + 4;
}
z := y;
```

(b) The transformed program

Double Execution with Reference (Golden) Check

```
int a, a2=55;
int b,b1,b2,b3;
...

a=3;

b1=a+2;

b2=a2+2;
if(b2!=58) printf(“error”);

b3=a+2;
if(b1==b2)printf(“res=%d”,b1);
else printf(“error”);
```



Code Duplication

- Usually in assembly level, all commands are executed twice or more and the results are compared:

```
add ax, cx
```

```
push dx
add ax, cx
mov dx, ax
add ax, cx
cmp ax, dx
...
pop dx
```

Replicating Variables

```
int a, b;

a=3;

...

b=a+2;
```

```
int a, b;
int a1, b1;

a=a1=3;
...

if(a==a1)
   b=b1=a+2;
else
   printf(“error”);
```

Error Detection and Correction Code (EDAC)

```
int a, b, c;

...

a=2;

...

b=a+2;

printf("%d",b)
;
```

```
int *p;
int *a, *b, *c;
int crc;

p=malloc(6);
a=p; b=p+1;
c=p+2;

...

*a=2;
crc=_code(p,3);
...

*b=*a+2;
crc=_code(p,3);

printf("%d",*b);
```

insert the following
Check periodically
In the program:

```
if(_code(p,3)!=crc))
  printf("error");
```

Control Flow Error Detection

- Signature bits

```
...

if(a>3)
{

...

}
...
```

```
bool x=false;
...

if(x) printf("error");

if(a>3)
{
x=x+1;

...

x=x+1;
}

if(x) printf("error");
...
```

incorrect jump!

Control Flow Error Detection

- Control flow check with regular expressions

S_1;
`if` B_1 `then` S_2;
`while` B_2 `do` S_3
`else` S_4
S_5;

$g :=$ 'a'; S_1;
`if` B_1 `then` $g :=$ 'b'; S_2;
`while` B_2 `do` $g :=$ 'c'; S_3
`else` $g :=$ 'd'; S_4
$g :=$ 'e'; S_5;

The regular expression produced from the program is

$$\mathcal{R} = a(bc^*|d)e$$

After each assignment to g,

a check is done. For example:

`check`("ae", "$a(bc^*|d)e$") returns `false`

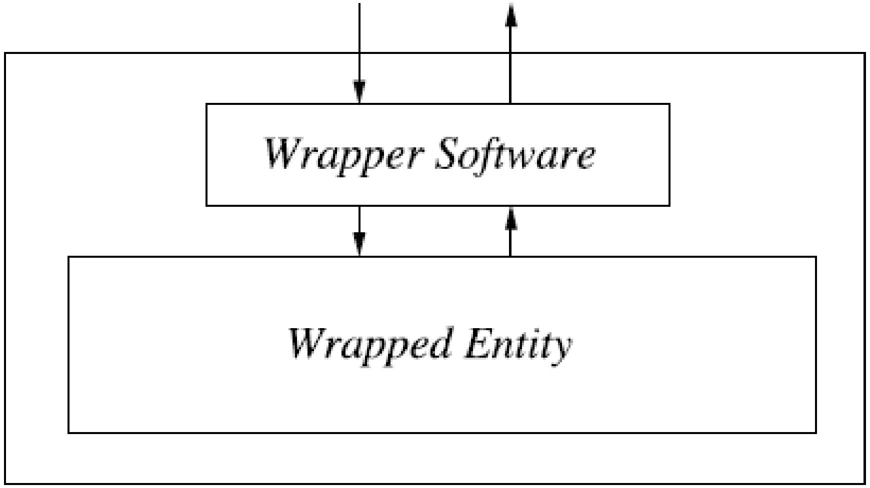
Software Fault-Tolerance

- N-Version Programming:

  N independent teams develop the same software. N-versions of software are executed at the same time and the results are compared at run-time.

Software Fault Tolerance

- Wrappers:



For example, C does not check buffer overflows:

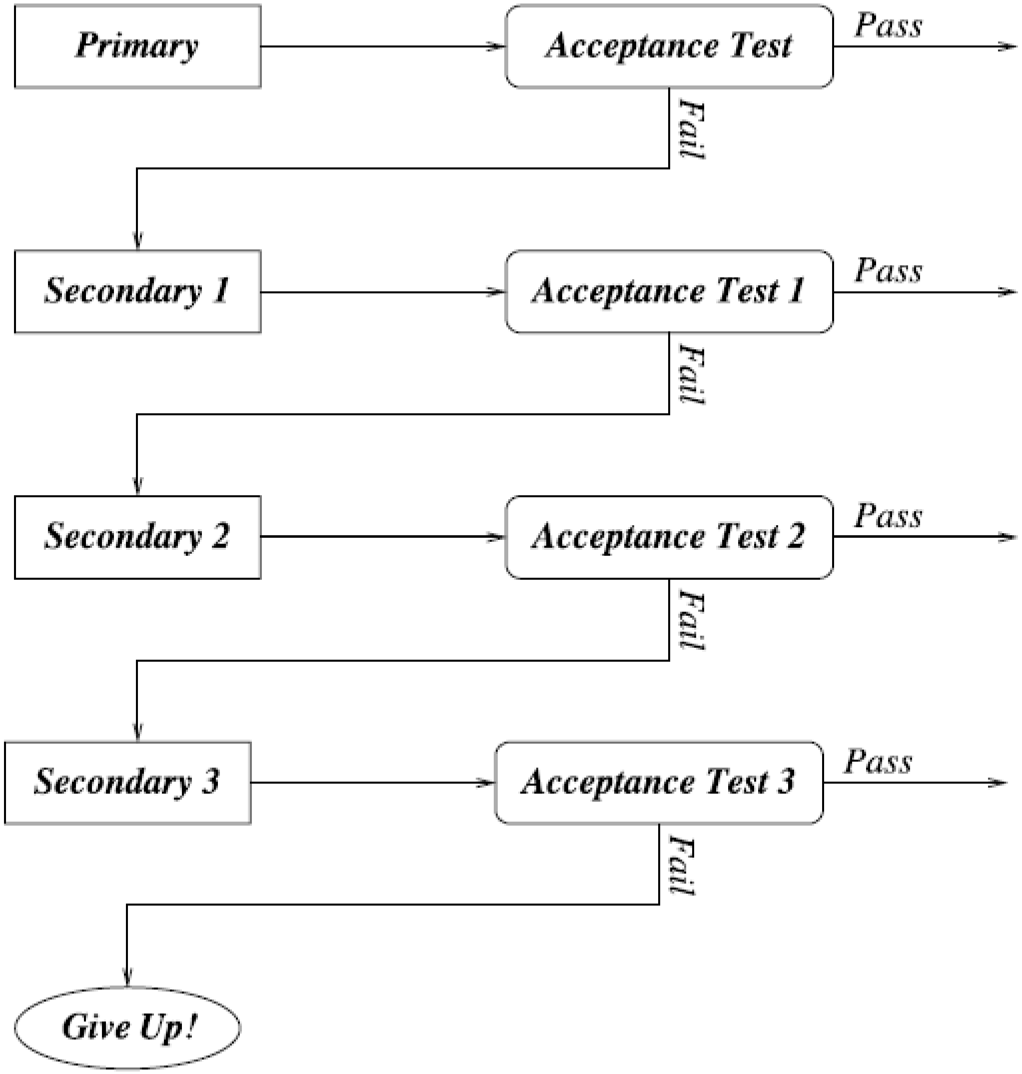
```
strcpy(str1, str2);
```

if str2 is bigger than str1 than buffer overflow occurs!

A wrapper can catch all the assignments to strings for instance and check their sizes.

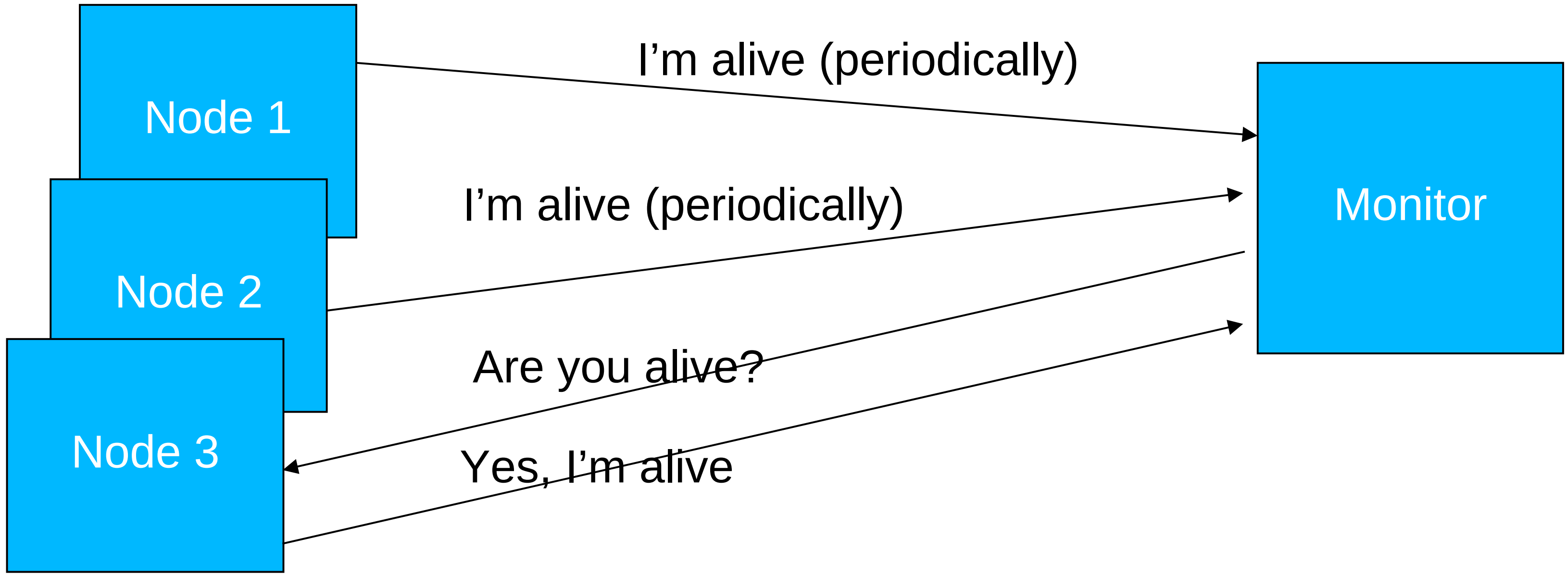
Software Fault-Tolerance

- Recovery Block Approach: N independent teams develop
  the same
  software.



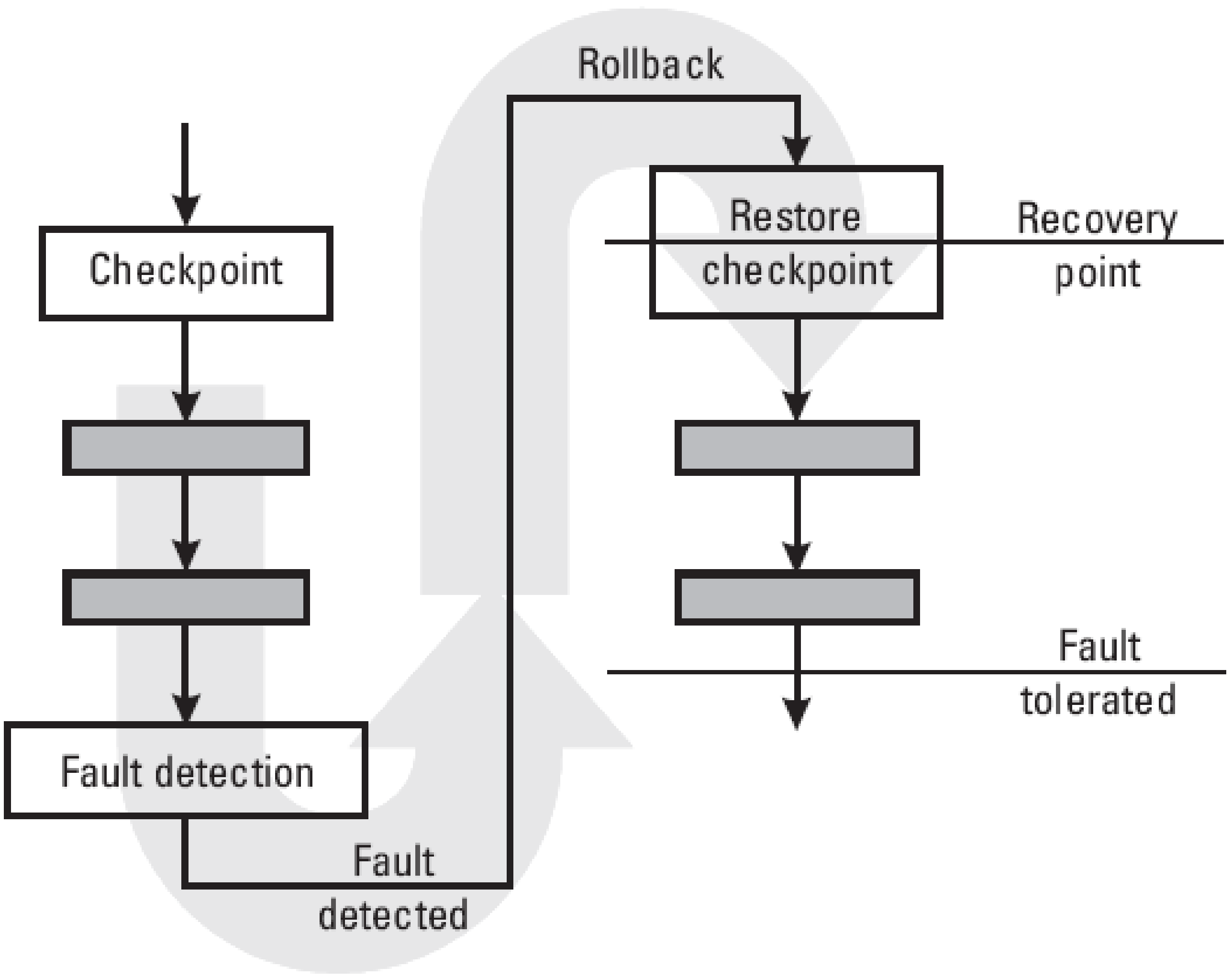
Fault Detection

- Push and Pull Messaging



One common example to push messaging is Watchdog timers used in microprocressors/microcontrollers.

Recovery: Checkpointing/Rollback



Checkpointing Level

- Kernel-level: Transparent to the user. Many OSs take checkpoints but it does not help to fault-tolerance.
- User-level: Library is provided to user. Application programs are linked to this library.
- Application-level: Application is responsible for carrying out all the functions. Provides user with the greatest control over the checkpointing process.

Checkpointing

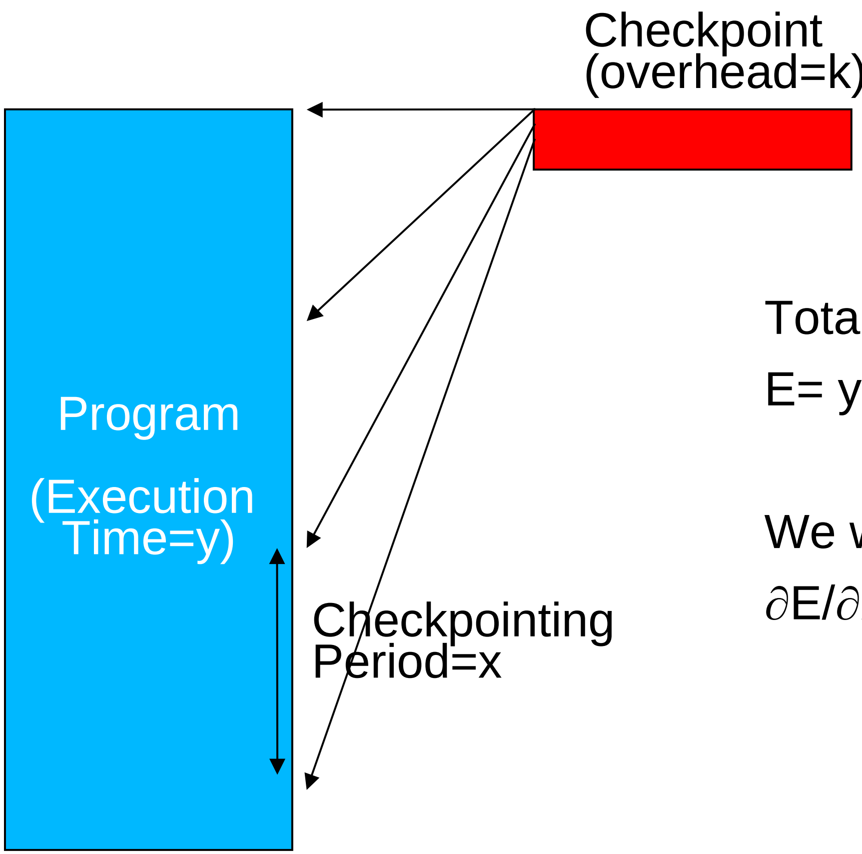
- Checkpoint context: Registers, program counter (or simply Task Control Block)
- Checkpointing overhead: The extra execution time needed to take a checkpoint
- Checkpointing latency: Generally identical to the overhead. But, writing to a disk may require more time! The size of checkpoint context plays an important role.
- Consider the following code:

```
for (i = 0; i < 1000000; i++)
    if (f(i) < min) {min = f(i); imin = i;}
for (i = 0; i < 100; i++) {
    for (j = 0; j < 100; j++) {
        c[i][j] += i * j / min;
    }
}
```

Here, checkpoint context includes *i, min* and *imin*.

Here, checkpoint context is large since it includes *i, j* and all *c[i][j]s*.

Optimum Checkpoint Period



k<<x

Total execution time in case of a fault is

E= y + y . k / x + x

We want to minimize E.

$\partial E/\partial x = 0$ =>

$$x = \sqrt{\frac{ky}{2}}$$