MATLAB EXPO 2016

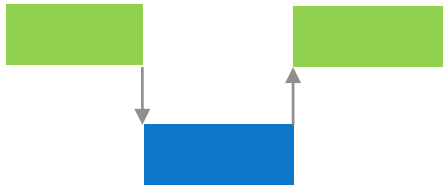
Rückwirkungsfreiheit zwischen Embedded SW-Komponenten – Polyspace hilft!

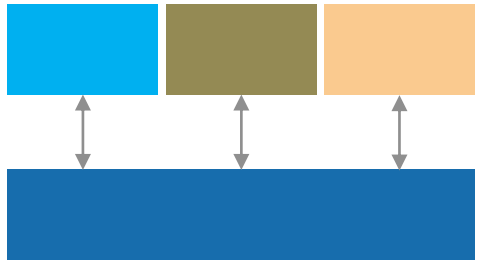
By Christian Guß

Freedom of Interference

What is that?

When **processes and modules working together on shared resources** some **interference issues** could occur which are very hard to find…





Timing and Execution

- Deadlocks
- Race conditions
- Sequence error

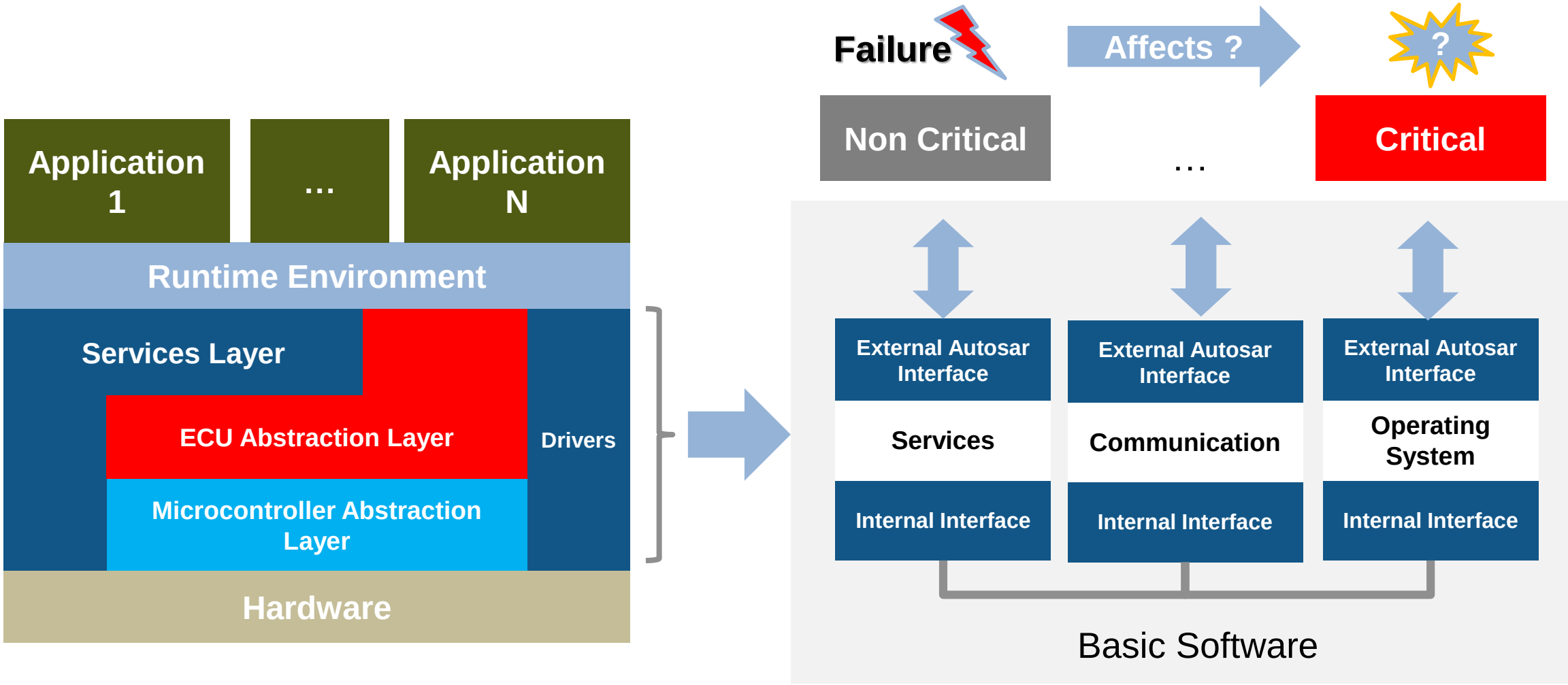
Memory

- Corruption of content
- Access out of bounds
- Invalid r/w access

Exchange of Information

- Interface violation
- Non initialized data
- Null-Pointers
- Data size mismatch

Typical Automotive Software Architecture



ISO 26262-6: Freedom from interference (Annex D)

Goal: **Prevent or detect** faults that can **cause interference** between software elements (e.g. different software partitions)

D2.2 Timing and execution

- Deadlocks
- Race Conditions

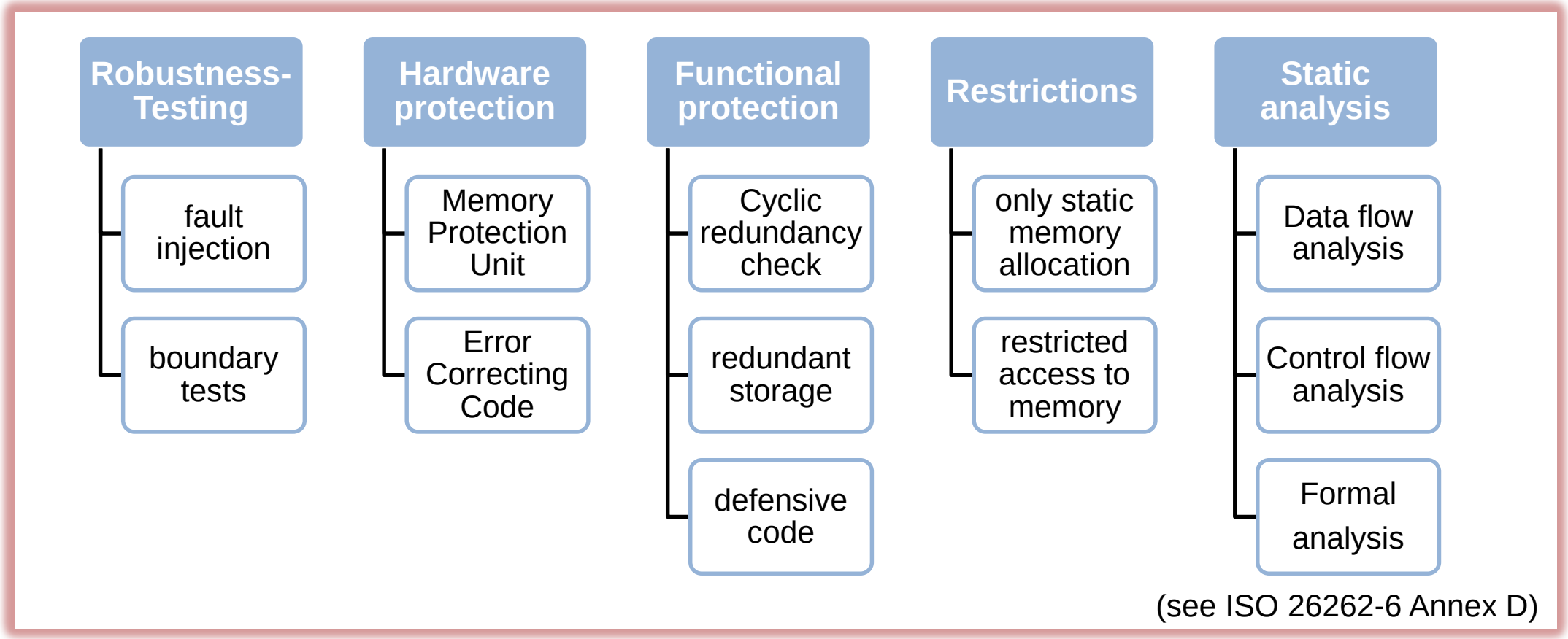
D2.3 Memory

- corruption of content
  - out-of-bound pointers and arrays, etc.
- read or write access to memory allocated to another software element
  - exhaustive identification of unprotected shared variables
  - documentation of read-/write access to global variable

D2.4 Exchange of information

- corruption of information
- loss of information

What you could do is…

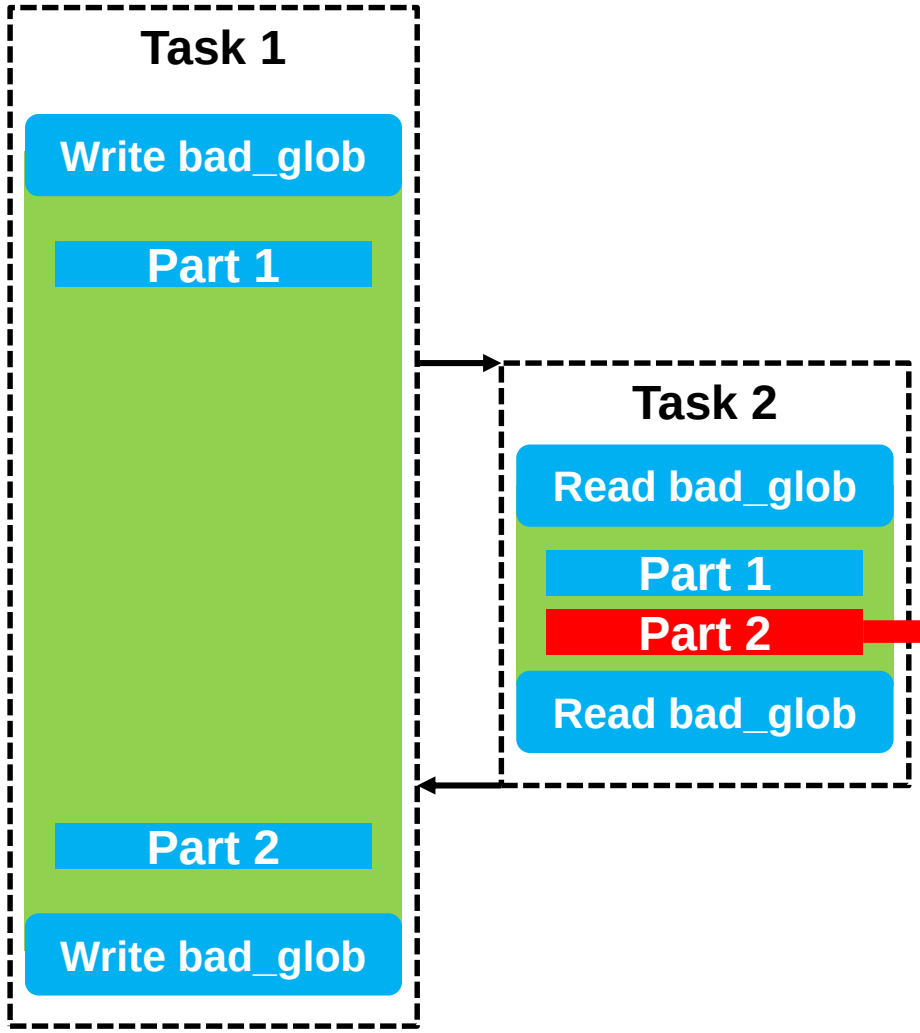


(see ISO 26262-6 Annex D)

Problem: Testing, Hardware protection, restrictions and functional protection could be:

- **very expensive** to implement,
- **not completely protective**,
- **reducing performance**.

Let‘s make an example…




```
1  /*============================================================================
2   *  DATA RACE
3   *============================================================================*/
4  int bad_glob1;      /* Defect Data race                                      */
5
6  void bug_datarace_task1(void)
7  {
8      bad_glob1 = 1;  /* Non-atomic write access                               */
9  }
10
11 void bug_datarace_task2(void)
12 {
13     int local_var;
14     local_var = bad_glob1;  /* Non-atomic read access                        */
15     printf("%d", local_var);
16 }
17
```

Data Race

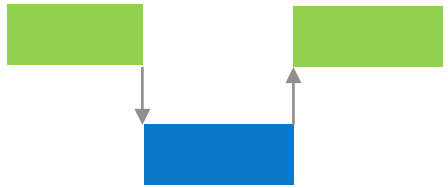
Fix: Critical Section!

```
BEGIN_CRITICAL_SECTION();
good_glob1 = 1;
END_CRITICAL_SECTION();
```

Problem: When needed?

Overusing can degrade system performance!

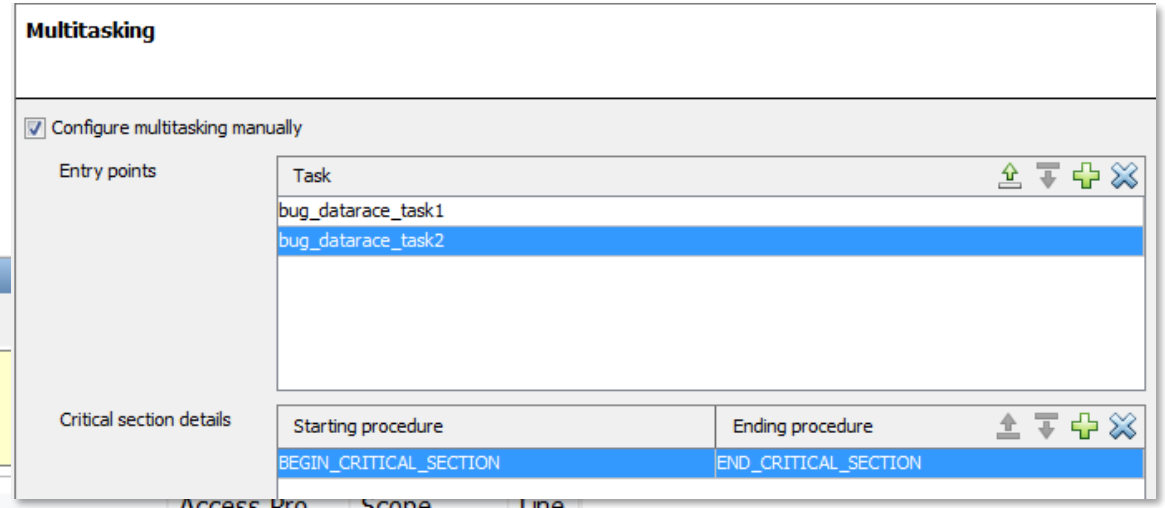
How to reduce efforts with „Timing and Execution“ Safety?



With static analysis!

Polyspace – Data race checks

Find Timing Issues with Multitasking



Check Details

Variable trace

ID 2: Data race

Certain operations on variable 'bad_glob2' can interfere with each other and cause unpredictable value.
To avoid interference, operations on 'bad_glob2' must be in the same critical section.

Access	Race Conditions	Access Pro...	Scope	Line
Write #1 (non-atomic) Operation might involve multiple machine instructions	This write in 'bug_task3()' conflicts with **Read #1** in 'bug_task4()'	No protection	bug_task3()	95
Read #1 (non-atomic) Operation with 64-bit variable on a 32-bit target	This read in 'bug_task4()' conflicts with **Write #1** in 'bug_task3()'	No protection	bug_task4()	101

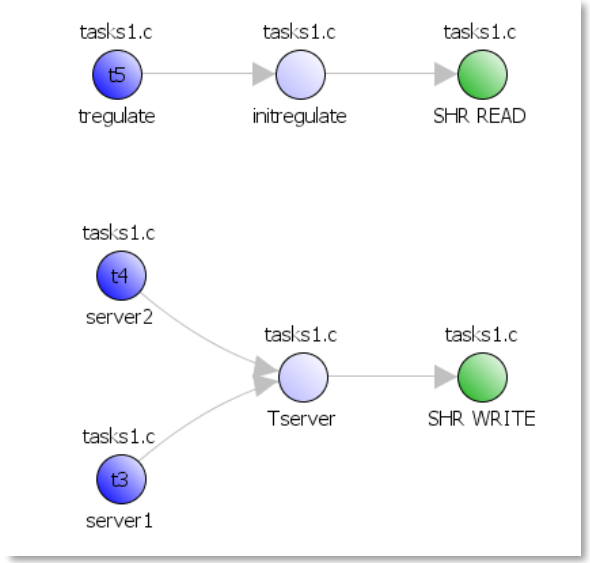
```
91  long long bad_glob2;
92
93  void bug_task3(void)
94  {
95      bad_glob2 += 1;
96  }
97
98  void bug_task4(void)
99  {
100     long long local_var;
101     local_var = bad_glob2;
```

```
91  long long bad_glob2;
92
93  void bug_task3(void)
94  {
95      bad_glob2 += 1;
96  }
97
98  void bug_task4(void)
99  {
100     long long local_var;
101     local_var = bad_glob2;
```

Polyspace - Global Variable Usage Protection

Global Variable		2	3	2	17
Shared			3	2	
Potentially unprotected variable			3		
	Variable: PowerLevel	tasks1.c			_init_globals()
	Variable: SHR4	tasks1.c			_init_globals()
	Variable: SHR2	tasks1.c			_init_globals()
Protected variable				2	
	Variable: SHR5	tasks1.c			_init_globals()
	Variable: SHR	tasks1.c			_init_globals()
Not shared		2			17
Unused variable		2			
	Variable: second_pai...	initialisations.c			_init_globals()
	Variable: __huge_val	huge_val.h			_init_globals()
Used non-shared variable					17

- ✓ **Shared protected global variable**
  Global variables shared between multiple tasks and protected from concurrent access by the tasks
- ✓ **Shared unprotected global variable**
  Global variables shared between multiple tasks but not protected from concurrent access by the tasks
- ✓ **Non-shared used global variable**
  Global variables used in a single task
- ✓ **Non-shared unused global variable**
  Global variables declared but not used

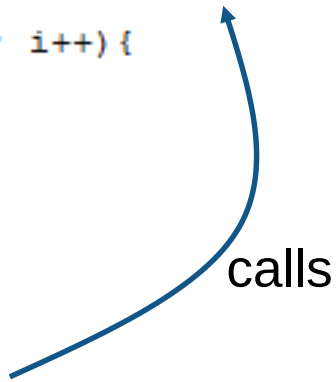


Protected variable
Variable 'tasks1.SHR' is shared among several tasks. All operations on 'tasks1.SHR' are protected by critical section.
Read by task: tregulate
Written by task: server1 server2

Event	File	Scope	Line
Written value: 0	tasks1.c	_init_globals()	30
Written value: 22	tasks1.c	Tserver()	81
Read value: 0 or 22	tasks1.c	initregulate()	53

Let‘s make another example…

```
1  char myarray[10];
2  int  VeryImportantData;
3
4  void myarray_init(char array[], int array_size)
5  {
6      for (int i = 0; i < array_size; i++){
7          array[i] = 0;
8      }
9  }
10
11 void integration_context()
12 {
13     // ... before ...
14     myarray_init(&array[0], 15);
15     // ... behind ...
16     lets_use_my_important_data(VeryImportantData);
17 }
```



- Is it safe to use myarray_init Function?

NO !

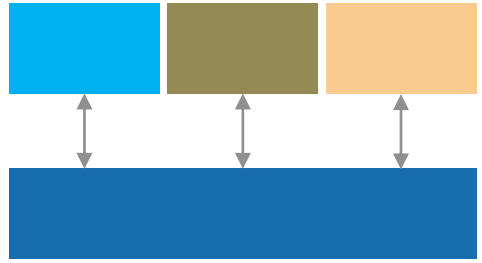
hard to find!

Problem with testing: Tests aren't exhaustive

"Program testing can be used to show the presence of bugs, but never to show their absence" (Dijkstra [1])

[1] Dijkstra, "Notes On Structured Programming", 1972

How to reduce efforts with „Memory“ Safety?



With static analysis!

Polyspace – Proving Memory Safety

With **Polyspace** …
you can **proof** the **existence** and **absence** of memory access errors like:

? Out of bounds array index
Warning: array index may be outside bounds : [0..9]
array size: 10
array index value: [1 .. 10]

```
int buffer[10],i = {0};
while (i++ < 10){
        buffer[i] = 0;
}
```

! Illegally dereferenced pointer
Error: pointer is outside its bounds
This check may be an issue related to unbounded input values
Dereference of parameter 'p' (pointer to int 32, size: 32 bits):
Pointer is null.

```
void foo(int *p)
{
        if (p == 0){
                *p = 42;
        }
}
```

! Non-initialized local variable
Error: local variable is not initialized (type: int 32)
This check may be a path-related issue, which is not dependent on input values

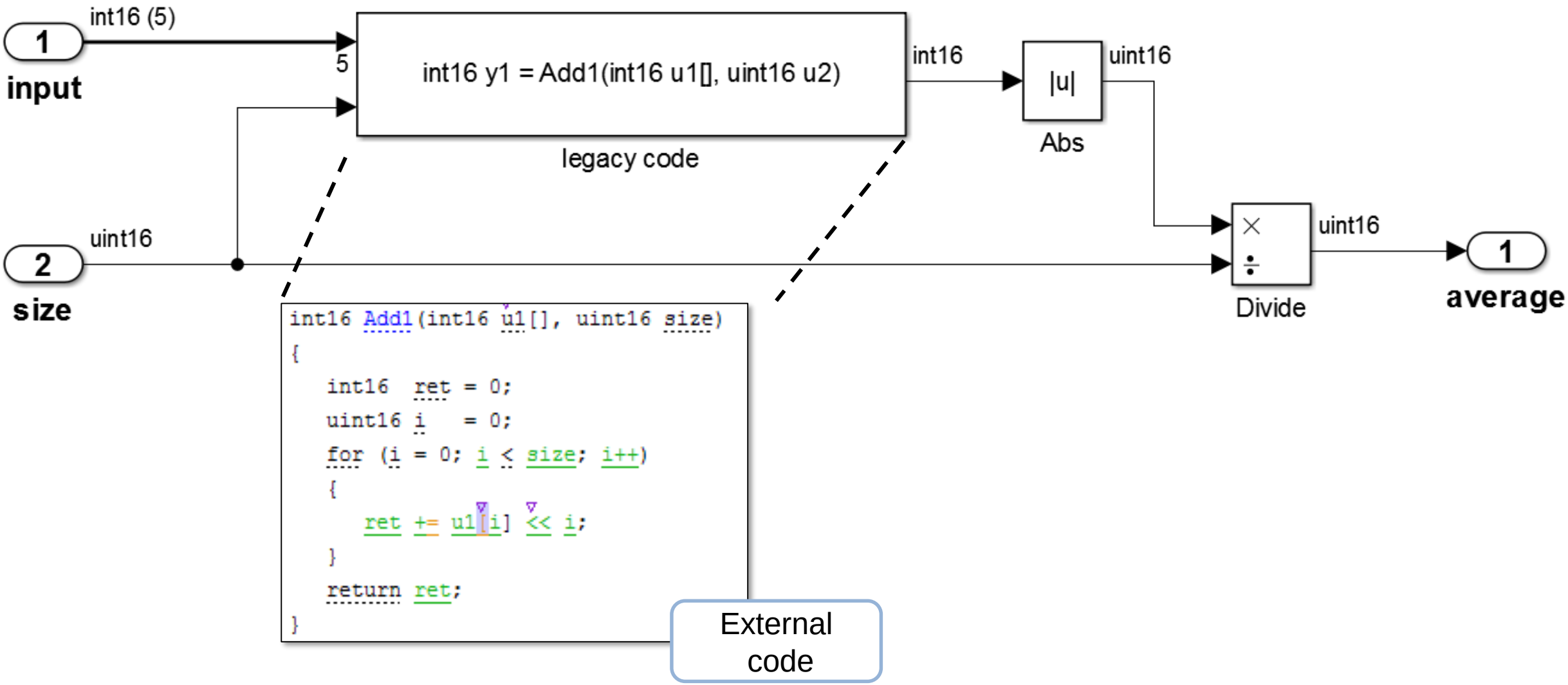
```
void foo(int *p)
{
        int va;
        if (p != 0){
                *p = va;
        }
}
```

Memory safety

- aims to avoid software errors that cause **safety and security vulnerabilities**
- dealing with **random-access memory (RAM) access**,
- such as **corruption of content** and **read/write access to memory allocated by another software element**.

Computer **languages such as C and C++** that support arbitrary pointer arithmetic, casting, and deallocation **are typically not memory safe**.

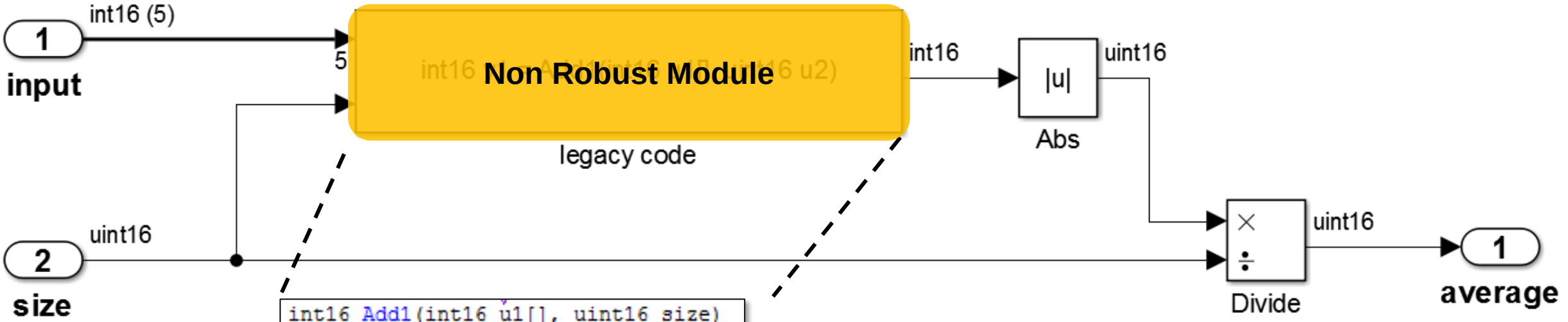
Let's make one last example…



How to reduce efforts with „Exchange of Information“ Safety?

With static analyis!

Example: Optimize design and architecture




```
int16 Add1(int16 u1[], uint16 size)
{
   int16  ret = 0;
   uint16 i   = 0;
   for (i = 0; i < size; i++)
   {
      ret += u1[i] << i;
   }
   return ret;
}
```

External code

Illegally dereferenced pointer

Warning: pointer may be outside its bounds

Dereference of expression (pointer to int 16, size: 16 bits):

Pointer is not null.

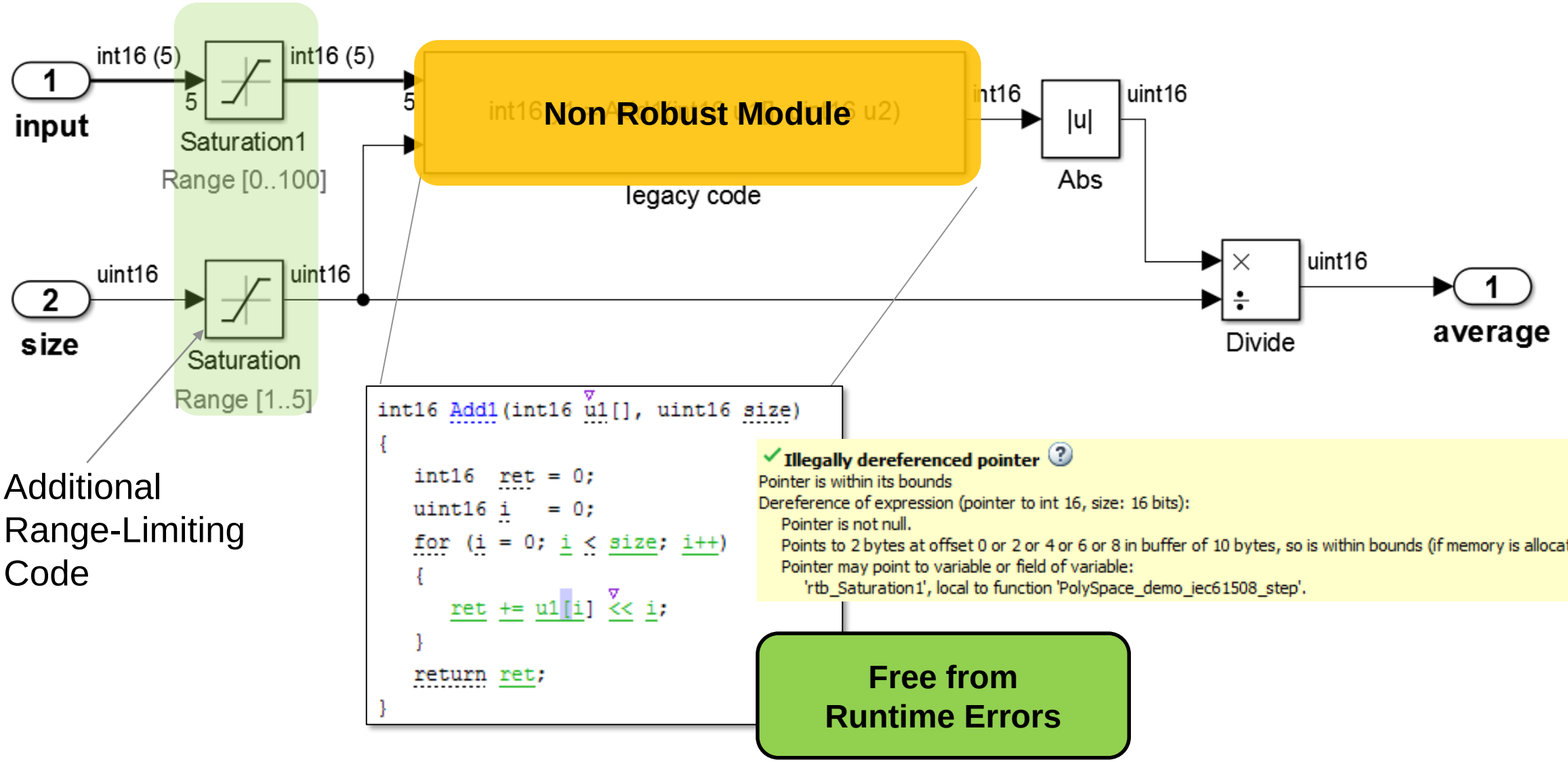
Points to 2 bytes at offset 0 or 2 or 4 or 6 or 8 or 10 in buffer of 10 bytes, so may be outside bounds.

Pointer may point to variable or field of variable:

'rtU'.

Potential Runtime Error inside!!!

Example: Optimize design and architecture



Summary

- Do you have Multicore applications?
- Do you have HW/SW protections?
- Do you like to reduce testing effort?

---

ask for our static analysis solutions TODAY