Compiler Testing
for use with Safety-Critical and/or Embedded Systems

日本語ノート版を用意しました。
興味頂ける場合は、メールにてご請求ください。

FUJI SETSUBI

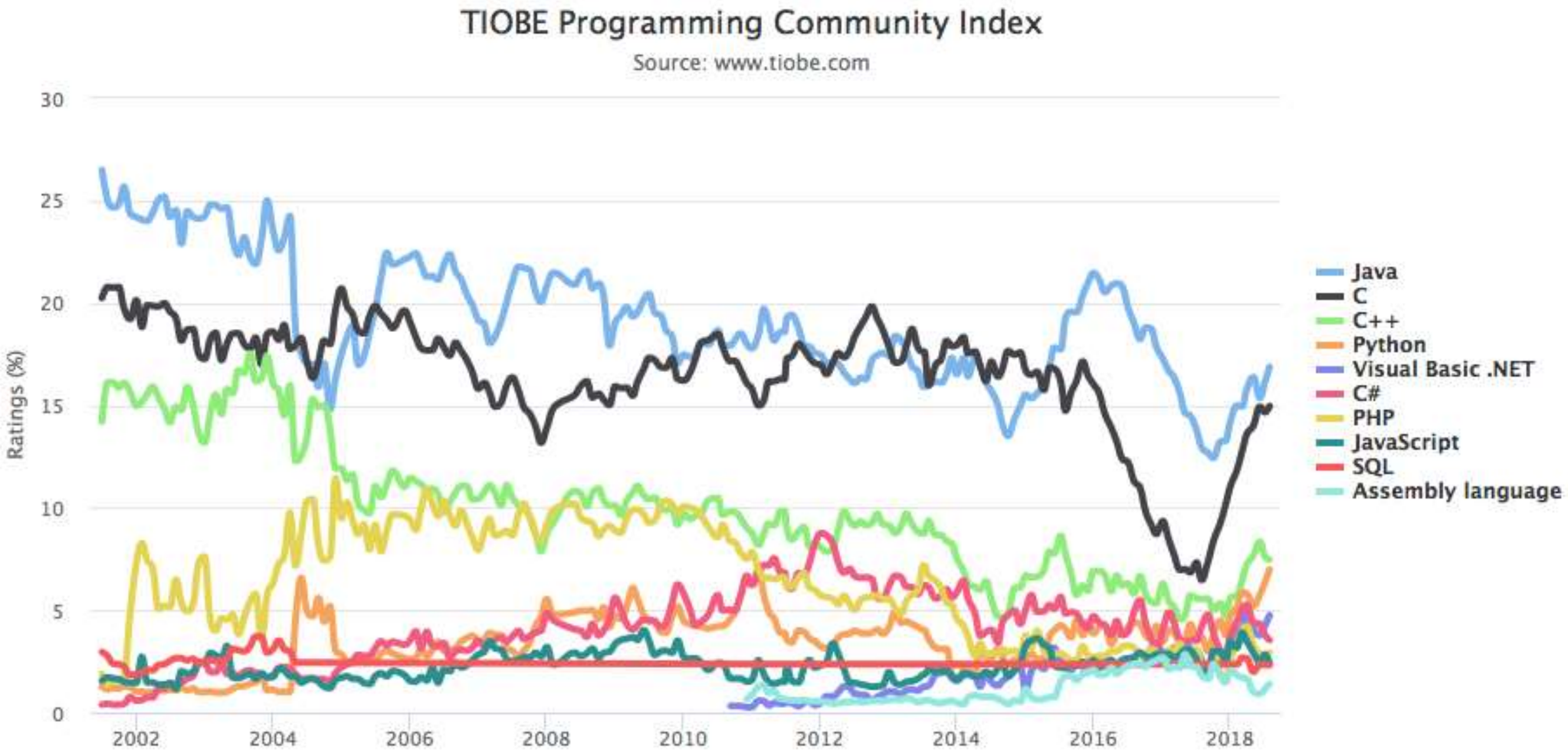
www.fuji-setsu.co.jp

Solid Sands

PART 1:
The Importance of C and C++ Compilers

- The C and C++ languages
- What a compiler does
- The (internal) complexity of a compiler
- Using a compiler in practice

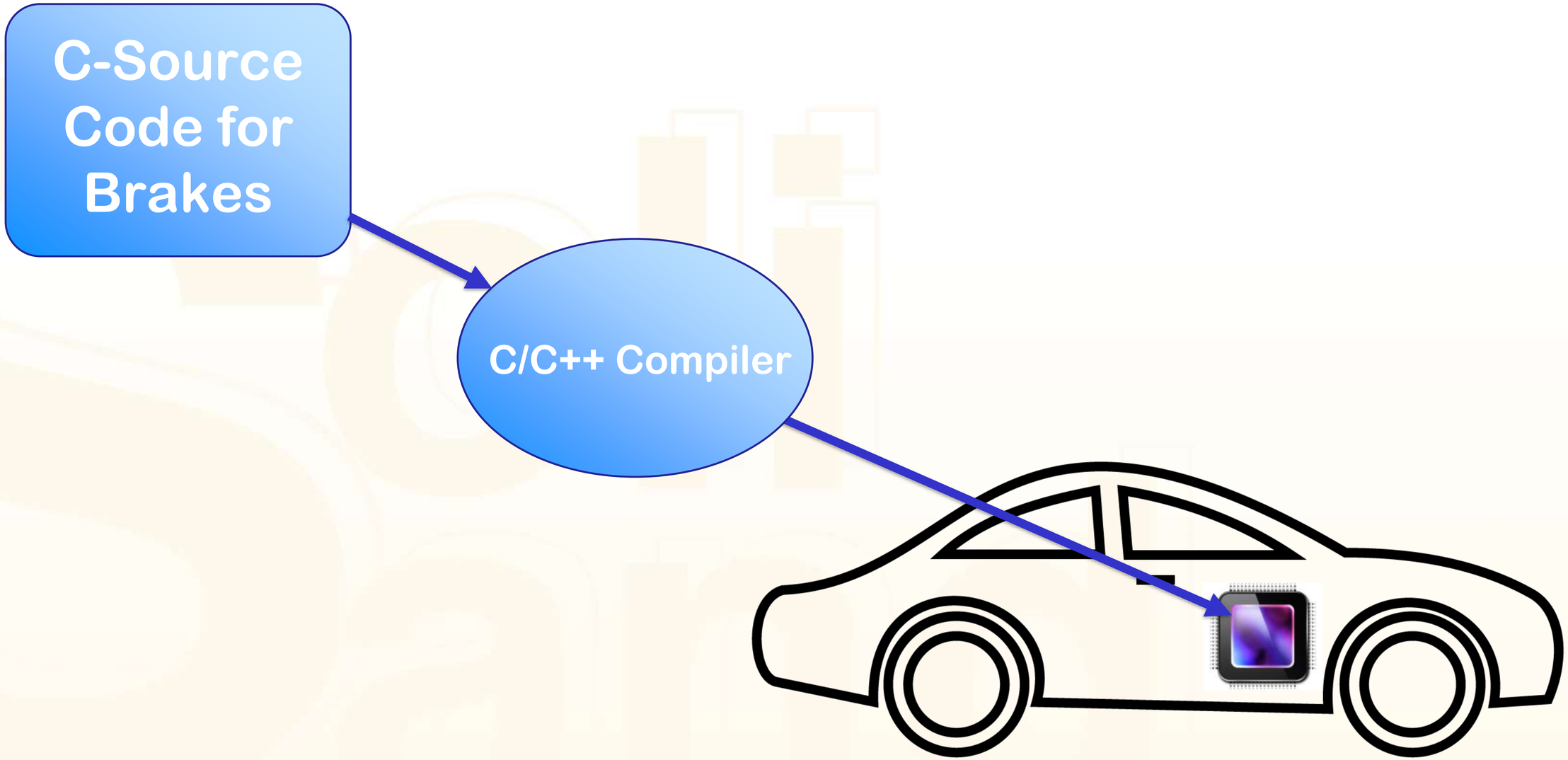
The Importance of C and C++



The Qualities of C and C++

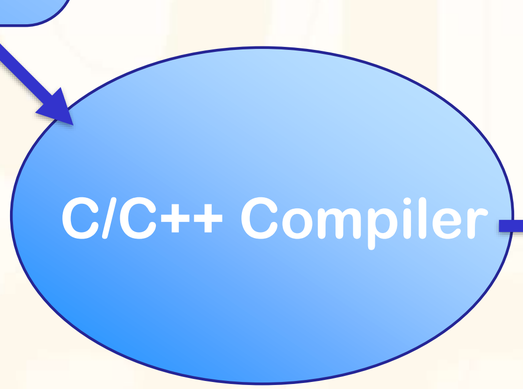
- Portable to many different architectures ("implementation defined features")
- Can run on "bare-metal", access to hardware
- Little or no run-time system (no hidden overhead, real-time)
- High performance
- Standardized by ISO (well-known - many tools, knowledgable developers)

What a Compiler Does



What a Compiler Does

```
int factorial (int n) {
    int res = 1;
    while (n > 1) {
        res *= n--;
    }
    return res;
}
```



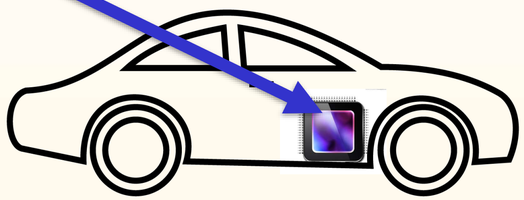

```
factorial:
    cmpl    $1, %edi
    jle     .L4
    movl    $1, %eax
.L3:
    leal    -1(%rdi), %edx
    imull   %edi, %eax
    movl    %edx, %edi
    cmpl    $1, %edx
    jne     .L3
    rep ret
.L4:
    movl    $1, %eax
    ret
```

Application vs Compiler LoC Perspective

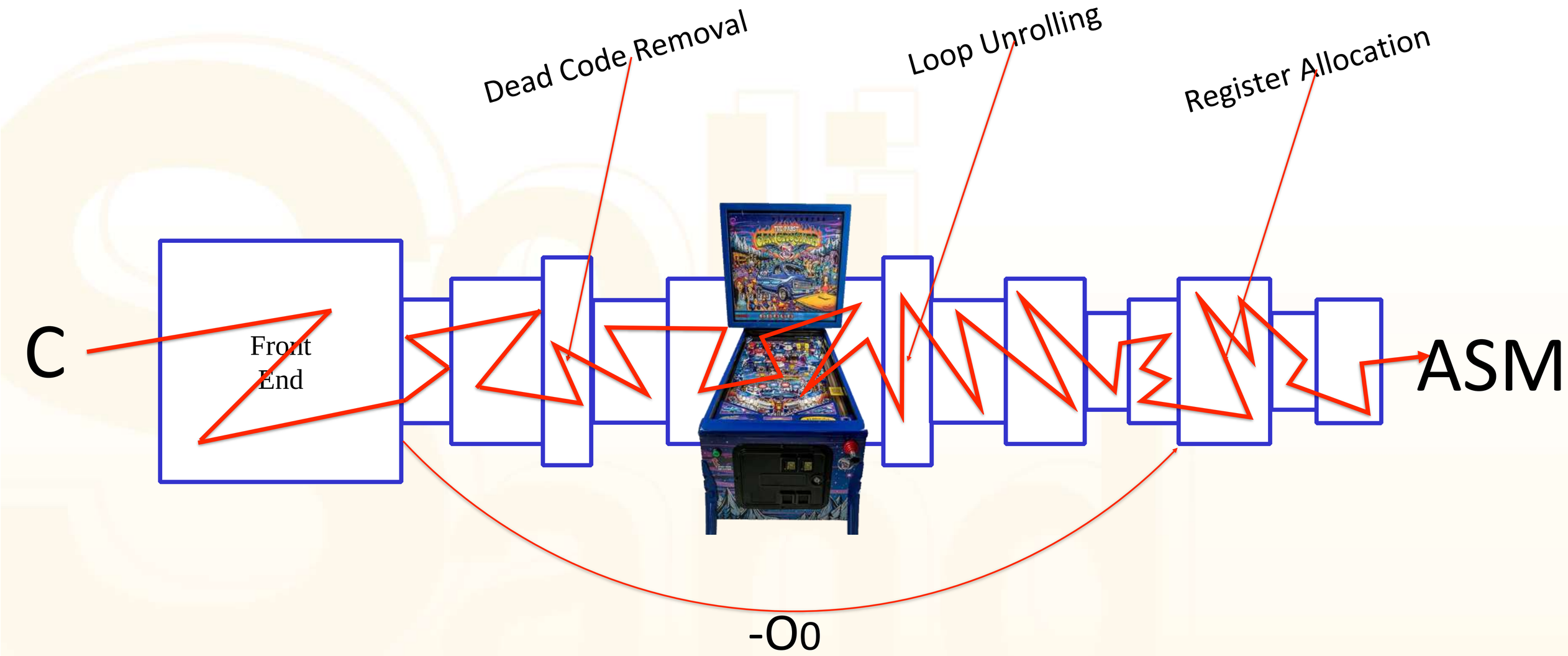
C-Source Code

5.000.000 Lines of C/C++ Compiler Source Code

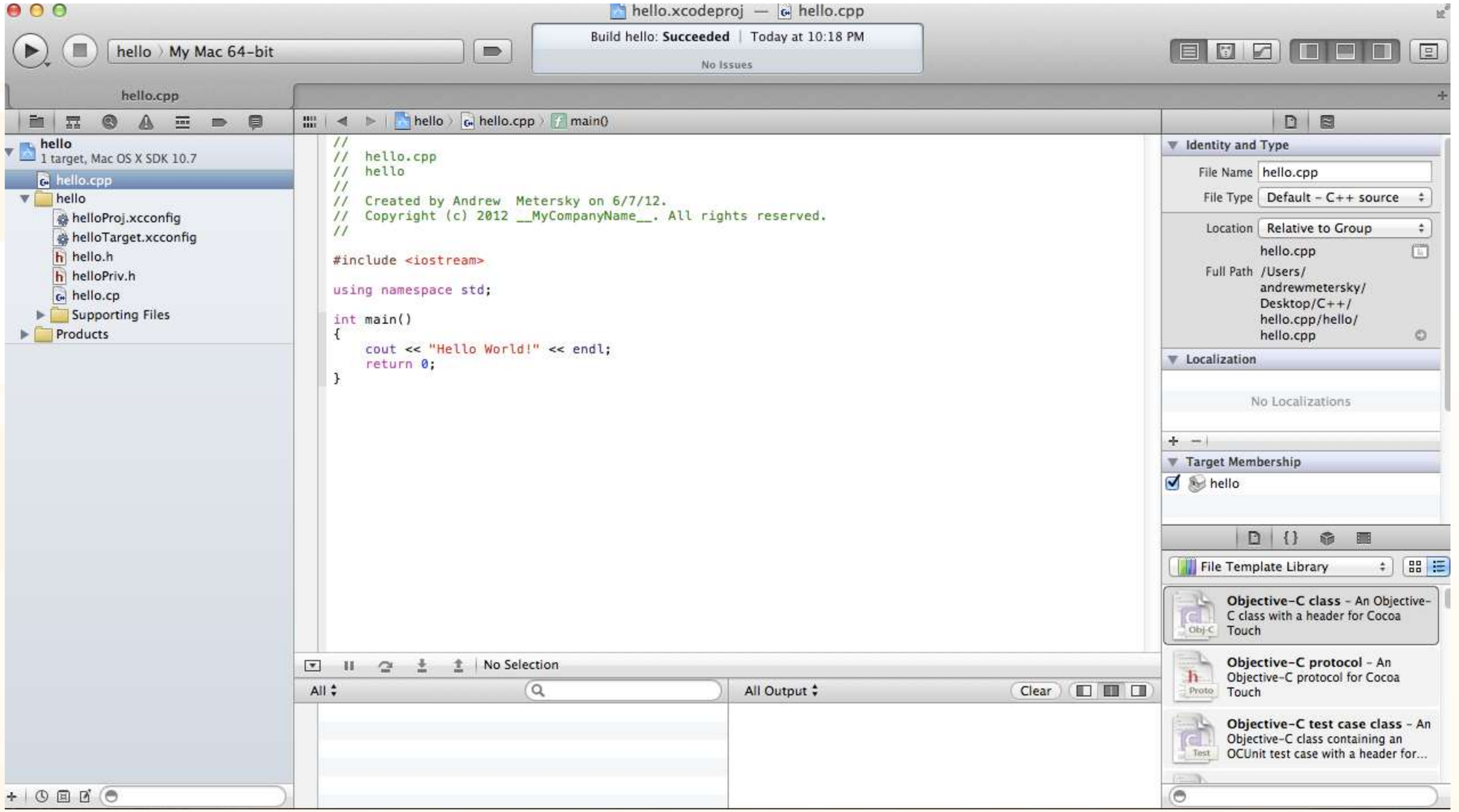
- No (MC/DC) coverage
- Not MISRA compliant
- Very many contributors
- Not designed with safety in mind
- Often open source



The Internal Structure of a Compiler



An IDE is Not a Compiler



Using the Compiler on the Command Line

```
$ cat hello.c
#include <stdio.h>
int main (void) {
    printf ("Hello Japan\n");
    return 0;
}
  Mac:~/tmp:
$ cc -O2 -Wall -std=c11 hello.c -o hello
  Mac:~/tmp:
$ ./hello
Hello Japan
  Mac:~/tmp:
```

PART 2: Testing a Compiler

- Testing compiler behavior
- Verifying diagnostics of ill-formed programs
- The organization of SuperTest
- Using SuperTest, requirements and configuration

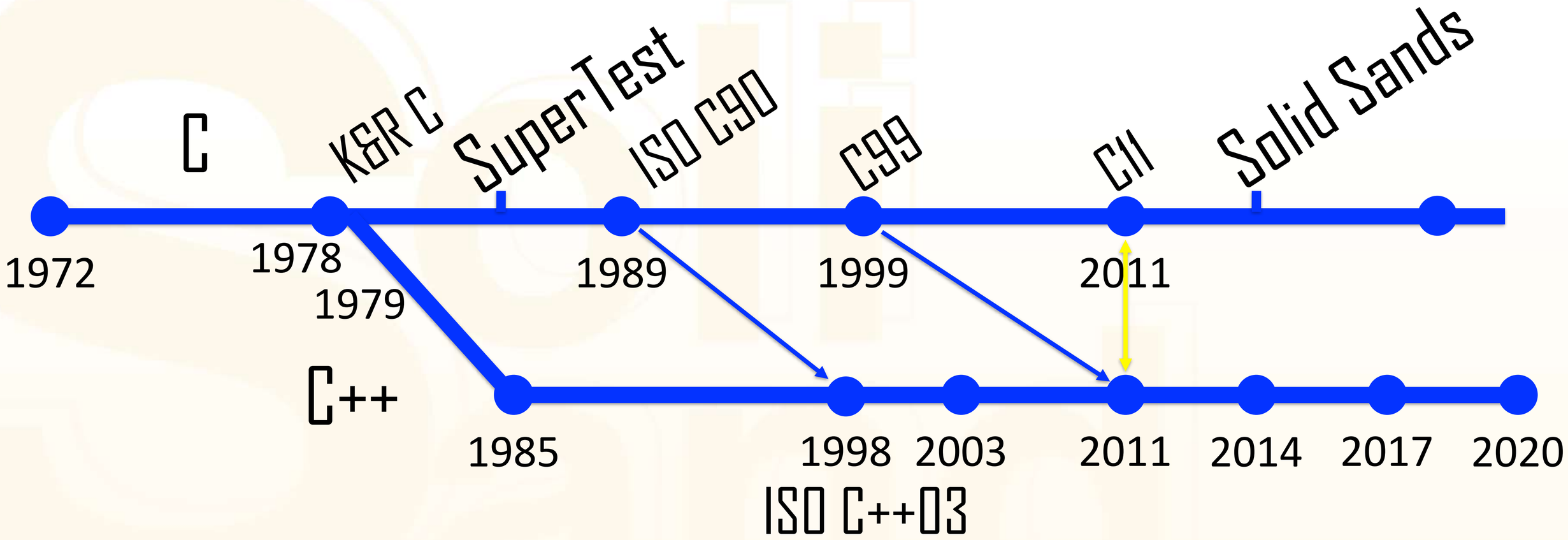
The Compiler Specification

=

The Language Definition

- The long history of C and C++
- ISO Standards:
  - C90, C99, C11, C++03/11/14/17
- Implementation defined behavior

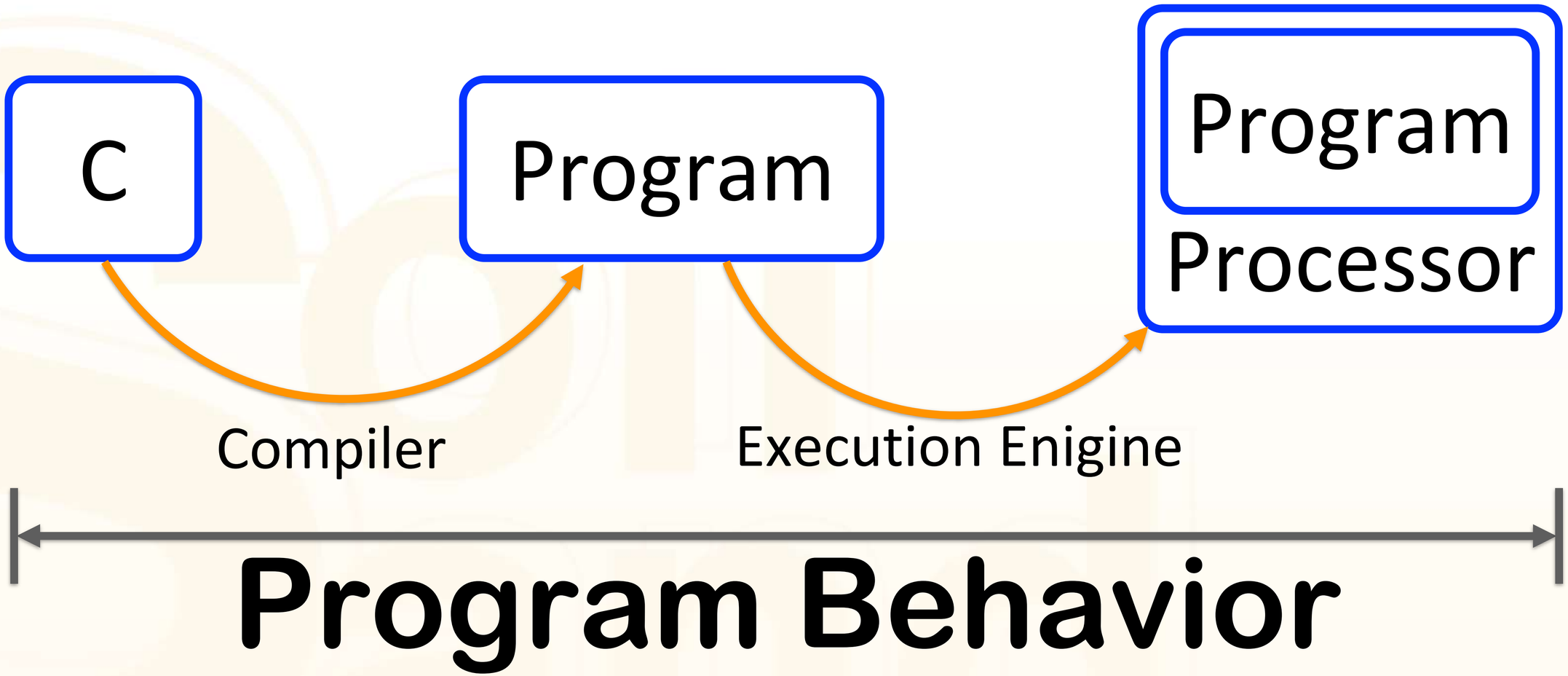
The History of C and C++



Characteristics that Make a Compiler Testable

- Well defined and stable specification
- Single Input, Single Output
- Deterministic
- Command-line Scriptable

The Language Specification Defines Program Behavior



Example from ISO C90

```
3.3.17 Comma operator

Syntax

          expression:
                  assignment-expression
                  expression ,  assignment-expression

Semantics

   The left operand of a comma operator is evaluated as a void
expression; there is a sequence point after its evaluation.  Then
the
right operand is evaluated; the result has its type and value./43/
```

Testing the Comma Operator SuperTest: 3/3/17/t2.c

```
void ge(int *p) {
    *p = 2;
}

int test_it(void) {
    int a, *p, r;
    p = &a;
    r = (ge(p), a++, a+=3, a+=8, a+8);
    return r == 22;
}
```

Non-conforming Programs and Diagnostics

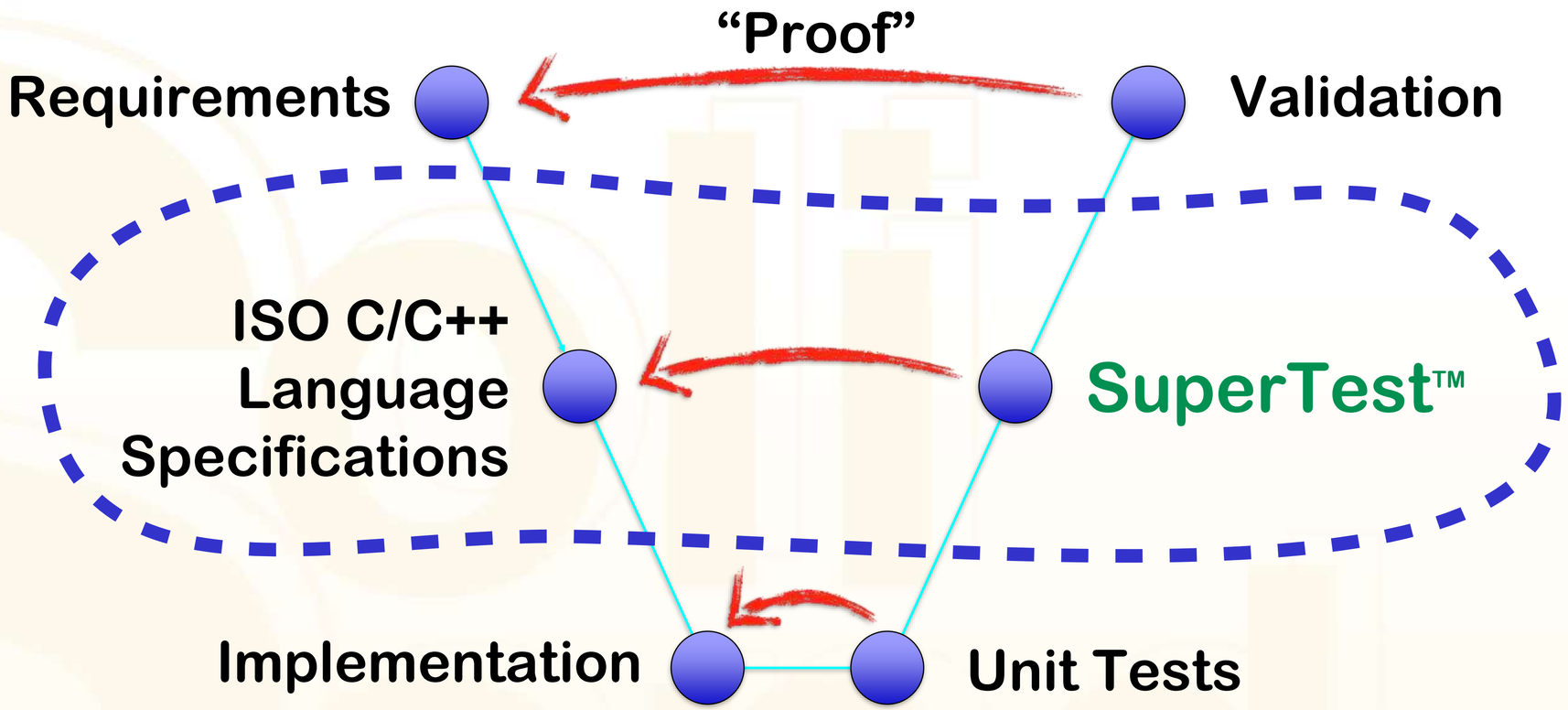

```
3.3.13 Logical AND operator

Syntax

          logical-AND-expression:
                  inclusive-OR-expression
                  logical-AND-expression &&  inclusive-OR-expression

Constraints

   Each of the operands shall have scalar type.

Semantics

   The && operator shall yield 1 if both of its operands compare
unequal to 0, otherwise it yields 0.  The result has type int.

   Unlike the bitwise binary & operator, the && operator guarantees
```

Testing Operator Precedence SuperTest: 3/3/13/xspr2089.c


```
void test_it( void ) {
        int i = 1, j = 5;
        while (--i && j >>= 1);
}
```

```
$ cc -c xspr2089.c
xspr2089.c:33:29: error: expression is not assignable
        while (--i && j >>= 1)
                   ~~~~~~~~ ^
1 error generated.
```

SuperTest Supports Requirements Traceability



The "V-Model"

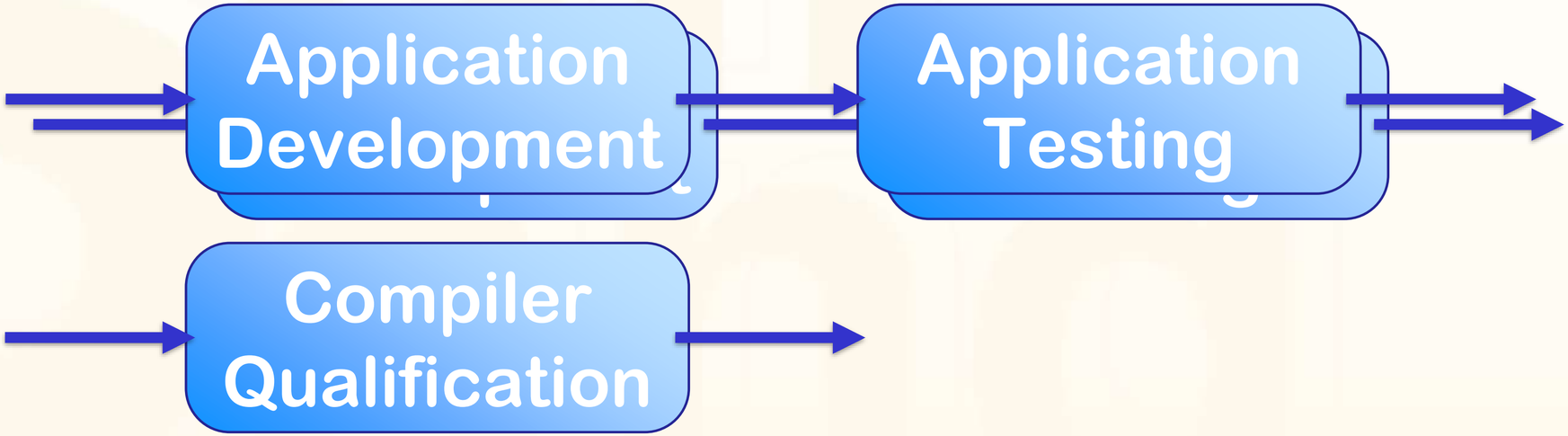
Formalizing Compiler Testing with ISO 26262

- ISO 26262-8.11: "Confidence in the use of software tools"
- Describes the process of compiler qualification for a specific use case
- Established practice for the automotive industry (see also: Rail, Nuclear, Aviation, Medical)

Benefit: Reduce Time-to-Market

Without Compiler Qualification:

With Compiler Qualification:



Benefits of In-House Continuous Compiler Qualification

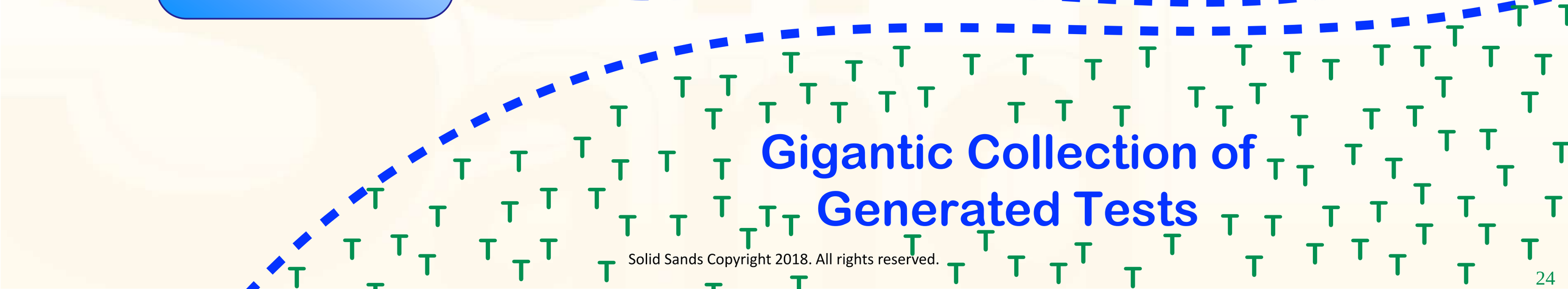
- Freedom in your choice of compiler (version)
  - Stay up to date with new compiler updates
  - No compiler vendor lock-in

- Flexibility in your choice of use case
  - Freedom to adapt compiler options and compilation environment

SuperTest: Basics

Test Driver

Test Reporter



SuperTest: What You Need

Host Computer

(Cross-)Compiler
1 GB Disk

- POSIX Environment
  - Linux
  - Mac
  - Solaris/HP-UX/…
  - Windows+Cygwin
- Native Windows platform

Execution Environment

- Native
- Target Board
- Simulator

4KB RAM

Exit-value to Host

Part 3: SuperTest Demo

Part 4: SuperTest Advanced Topics

- Optimization Testing
- ABI-tester
- Arithmetic depth suites
- MISRA C
- Compiler Qualification for Functional Safety

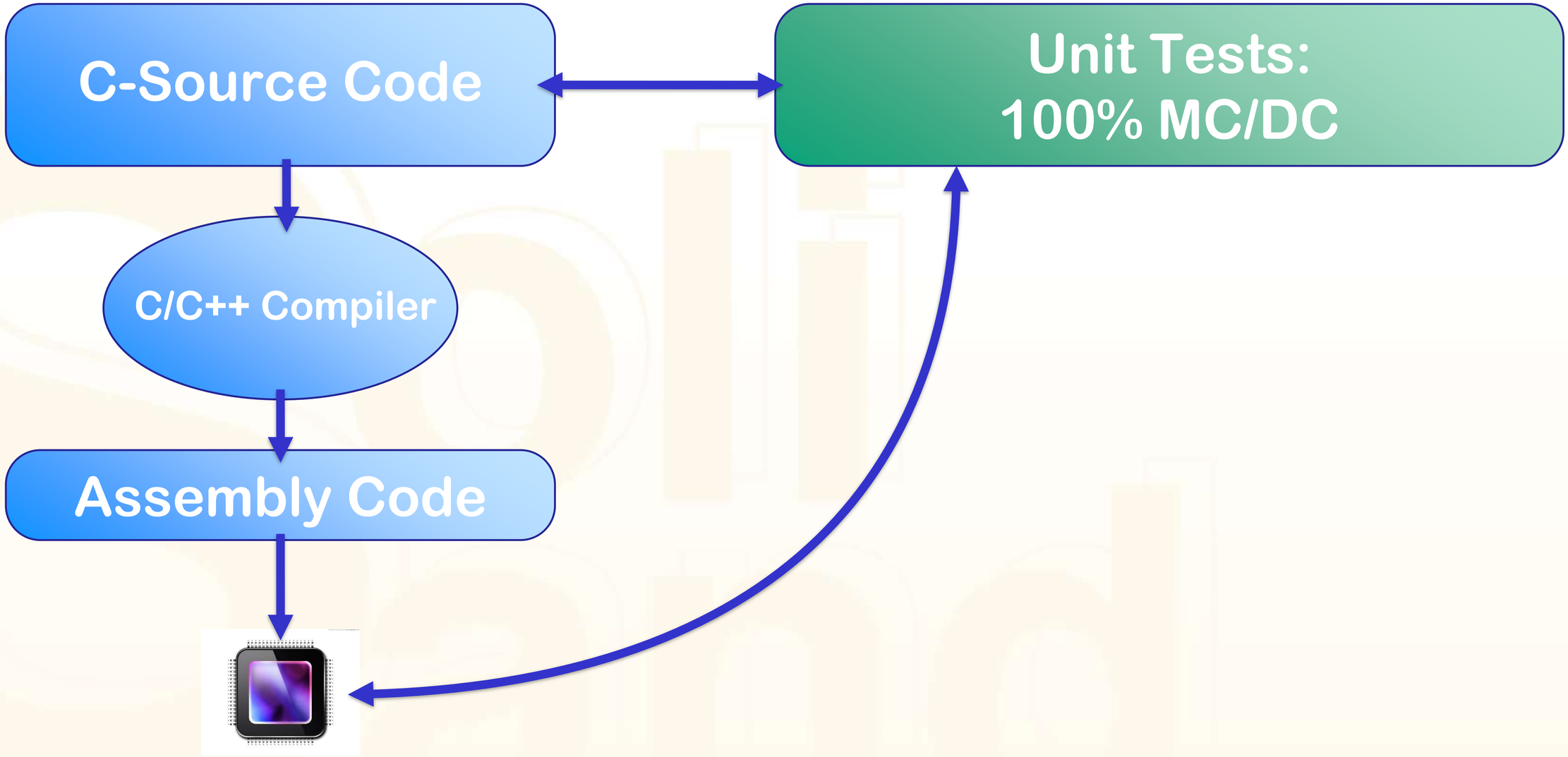
Compiler Optimizations

- **Improve execution speed of programs**
- **Do not change their behavior**
- **The language specifications says:**

The semantic descriptions in this International Standard describe the behavior of an abstract machine in which issues of optimization are irrelevant.

- **Improve a non-functional behavior of C, but are an important function of compilers**
- **So, how to test optimizations?**

Confidence to Detect Optimization Errors with Coverage Testing



Example loop, MC/DC coverage

```
int f(int n) {
    int total = 0;
    for (int i=0; i<n; i++){
        total += i & n;
    }
    return total;
}
```

Full MC/DC at source with unit test:

f(1)

Compiled at -O0, assembly level coverage:

f(1)

```
+: push   rbp
+: mov    rsp,rbp
+: mov    edi,-0x4(rbp)
+: movl   0x0,-0x8(rbp)
+: movl   0x0,-0xc(rbp)
+: mov    -0xc(rbp),eax
+: cmp    -0x4(rbp),eax
+: jge    0x40051b <f+0x3b>
+: mov    -0xc(rbp),eax
+: and    -0x4(rbp),eax
+: add    -0x8(rbp),eax
+: mov    eax,-0x8(rbp)
+: mov    -0xc(rbp),eax
+: add    0x1,eax
+: mov    eax,-0xc(rbp)
+: jmpq   0x4004f5 <f+0x15>
+: mov    -0x8(rbp),eax
+: pop    rbp
+: retq
```

```
int f(int n) {
    int total = 0;
    for (int i=0; i<n; i++){
        total += i & n;
    }
    return total;
}
```

Compile at -O2:
No more than 18% structural coverage at assembly with unit test:

f(1)

Needs 5 tests for full structural coverage.
Full branch coverage not possible

```
+: test   edi,edi
v: jle    0x4004c2 <f+0x12>
+: xor    edx,edx
+: cmp    0x7,edi
v: ja     0x4004c5 <f+0x15>
+: xor    eax,eax
+: jmpq   0x4005d0 <f+0x120>
-: xor    eax,eax
-: retq
-: mov    edi,ecx
-: and    0xfffffff8,ecx
-: mov    0x0,eax
-: je     0x4005d0 <f+0x120>
-: movd   edi,xmm0
-: pshufd 0x0,xmm0,xmm0
-: lea    -0x8(rcx),edx
-: mov    edx,eax
-: shr    0x3,eax
-: bt     0x3,edx
-: jb     0x40051a <f+0x6a>
-: movdqa 0x17c(rip),xmm1
-: pand   xmm0,xmm1
-: movdqa 0x180(rip),xmm3
-: pand   xmm0,xmm3
-: movdqa 0x184(rip),xmm5
-: mov    0x8,edx
-: test   eax,eax
-: jne    0x400530 <f+0x80>
```

```
-: jmpq   0x4005a7 <f+0xf7>
-: pxor   xmm1,xmm1
-: movdqa 0x14a(rip),xmm5
-: xor    edx,edx
-: pxor   xmm3,xmm3
-: test   eax,eax
-: je     0x4005a7 <f+0xf7>
-: mov    ecx,eax
-: sub    edx,eax
-: movdqa 0x163(rip),xmm8
-: movdqa 0x16a(rip),xmm9
-: movdqa 0x172(rip),xmm6
-: movdqa 0x17a(rip),xmm7
-: nopw   cs:0x0(rax,rax,1)
-: movdqa xmm5,xmm2
-: paddd  xmm8,xmm2
-: movdqa xmm5,xmm4
-: pand   xmm0,xmm4
-: pand   xmm0,xmm2
-: paddd  xmm1,xmm4
-: paddd  xmm3,xmm2
-: movdqa xmm5,xmm1
-: paddd  xmm9,xmm1
-: movdqa xmm5,xmm3
-: paddd  xmm6,xmm3
-: pand   xmm0,xmm1
-: pand   xmm0,xmm3
-: paddd  xmm4,xmm1
```

```
-: paddd  xmm2,xmm3
-: paddd  xmm7,xmm5
-: add    0xfffffff0,eax
-: jne    0x400560 <f+0xb0>
-: paddd  xmm3,xmm1
-: pshufd 0x4e,xmm1,xmm0
-: paddd  xmm1,xmm0
-: pshufd 0xe5,xmm0,xmm1
-: paddd  xmm0,xmm1
-: movd   xmm1,eax
-: cmp    edi,ecx
-: mov    ecx,edx
-: je     0x4005dc <f+0x12c>
-: nopw   0x0(rax,rax,1)
+: mov    edx,ecx
+: and    edi,ecx
+: add    ecx,eax
+: inc    edx
+: cmp    edx,edi
v: jne    0x4005d0 <f+0x120>
+: retq
```

SuperTest has Optimizations Covered

- Investigate MC/DC coverage of machine code
- Investigate structural coverage of the compiler

Case Study: Verum uses DSL to Create Verified Software Components

verum®
software by design

dezyne™
create explore verify trust

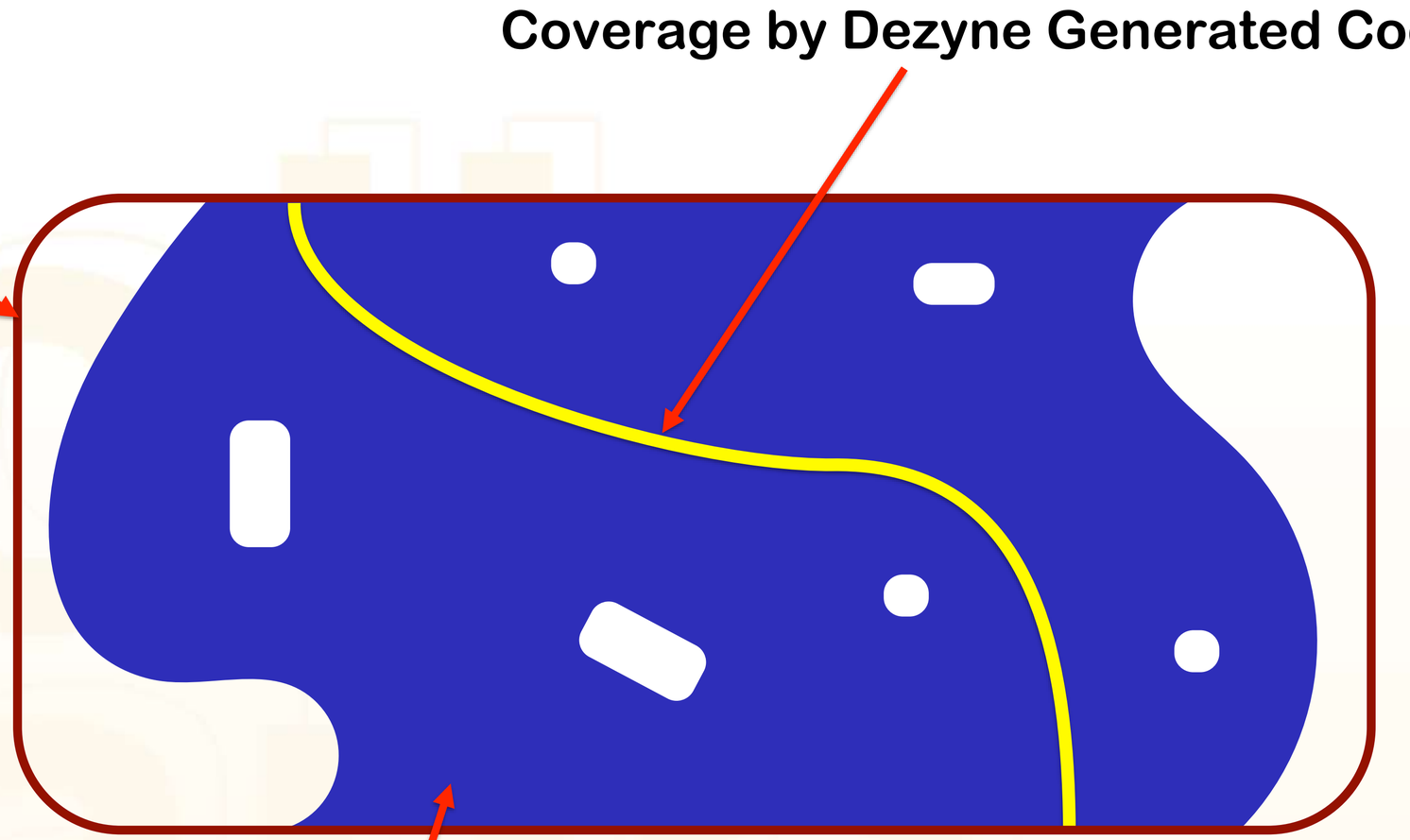
Dezyne Code Generator

```
static int8_t total_e(addint* self) {
        (void)self;
        int8_t l = self->left->in.e(self->left);
        int8_t r = 0;
        r = self->right->in.e(self->right);
        self->reply__int = l + r;
        return self->reply__int;
}
```

C-Compiler

Demonstrate Equivalence & Coverage

Comparing Test-Suite to Application Coverage to Create Confidence

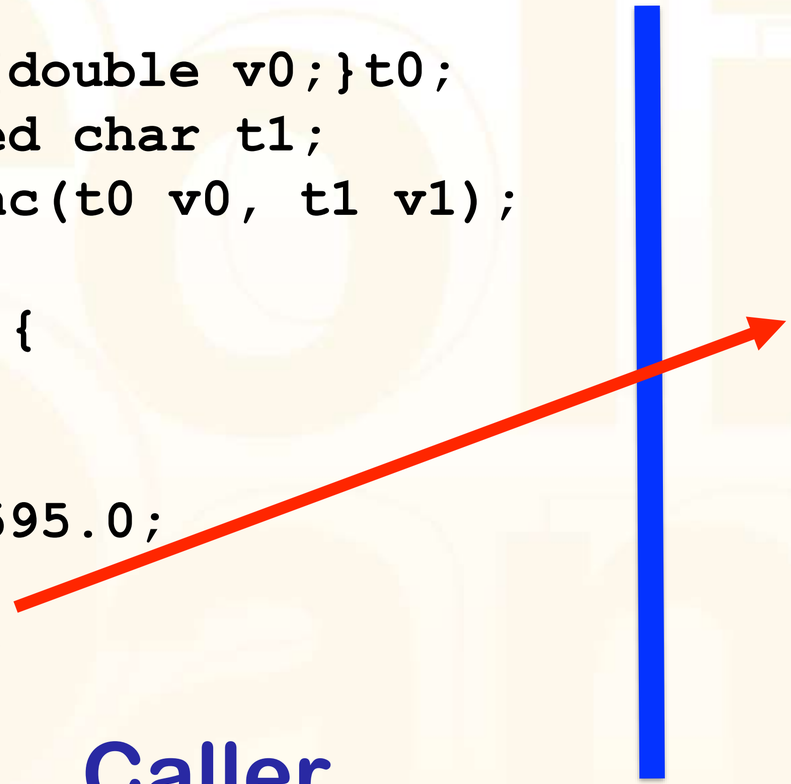


ABI Tester for Calling Conventions

- Calling conventions define how arguments are passed between functions

```
typedef struct{double v0;}t0;
typedef unsigned char t1;
extern void func(t0 v0, t1 v1);

int main (void){
  t0 v0;
  t1 v1;
  v0.v0 = 9071595.0;
  v1 = 234U;
  func(v0, v1);
  return 0;
}
```

Caller



```
#include <assert.h>
typedef struct{double v0;}t0;
typedef unsigned char t1;


void func(t0 v0, t1 v1)
{
  assert (v0.v0 == 9071595.0);
  assert (v1 == 234U);
}
```

Callee

Case-study: DENSO ABI Testing

- DENSO is interested in calling conventions for reasons of:
  - Performance: avoid unnecessary instructions at function call
  - Efficiency in development: avoid extra programming to match different ABIs when implementing an assembly function
- For these reasons, the compiler should conform to the ABI standard
- ABI-tester efficiently detects some ARM compilers they use that do not conform to the ABI standard

Implementation Defined Arithmetic

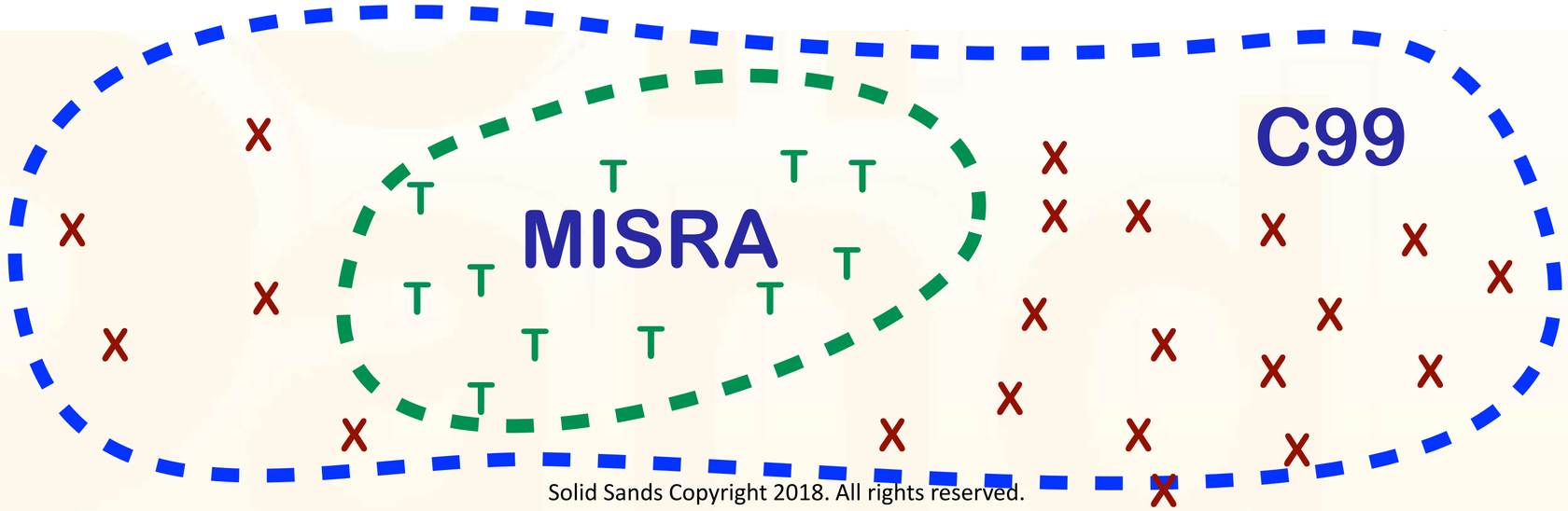
- C's data model is implementation defined: int can be 16/24/32/19/... bits
- Once the data model is set, C has strict rules for arithmetic: promotion, conversion, overflow, signedness

- SuperTest has generated 'depth' suites for every known data model

Testing MISRA C Analysis Tools

- **The MISRA Guidelines restricts the use of 'unsafe' constructs in C**

Rule 13.3	A full expression containing an increment (++) or decrement (––) operator should have no other potential *side effects* other than that caused by the increment or decrement operator



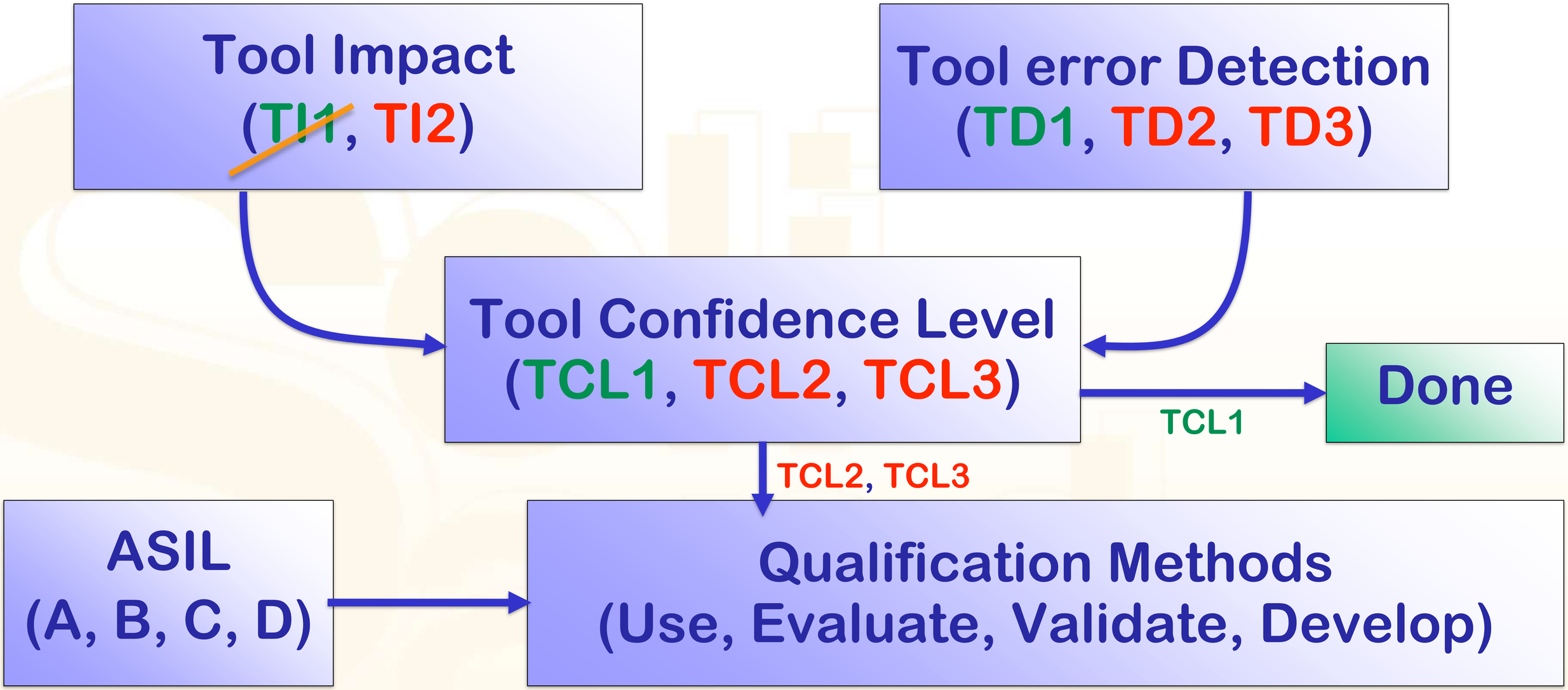
Fine C99 program that violates MISRA C:2012:13.3 (from SuperTest)

```
int32_t main (void)
{
    int32_t b[7]={0,1,2,3,4,5,6};
    int32_t *c=b;
    int32_t d=1;
    int32_t rv;
    rv = --c[d]; /* Violation */
    return rv;
}
```

Compiler Qualification for Functional Safety

- Goal: develop confidence in the use of the compiler for a safety critical application
  - For a specific use case (application specific)
  - By verification against the language specification
  - Define mitigations for compiler failures

ISO 26262 Compiler Evaluation



Compiler Qualification Service by Solid Sands

- In accordance with ISO 26262 "Tool Qualification"
- Jointly define validation procedure, definition of "Use Case"
- Test run, analysis of results, definition of mitigations
- Full report of test results including raw results

But Do Not Wait Too Long

- Solid Sands recently qualified a much used compiler for embedded systems
- We found six serious run-time failures - no mitigation possible
- This was supposed to be the final step in the project, but now faced delays:
  - Change use-case
  - Wait for compiler update
  - Change to different compiler

Summary

- SuperTest is used all over the world, from Canada to Germany to India to Japan
- SuperTest is easy to use for compiler developers and compiler users
- Solid Sands offers expertise and services in the domain of functional safety, like ISO 26262

SuperTest is the best test-suite for C and C++ compilers and libraries

SuperTest 成果

* 同じコンパイラを1000本以上使用している
  ・バージョンアップごとにSuperTestでチェックして、使ってはいけない最適化などの社内ルールを施行することで、過去のバージョンに戻さなければならないような問題が起こっていない。数千人に関わることなので非常に高い成果である。
* ABI-Tester 機能でABIの規則に準拠
  ・他のサプライヤーの製品と繋がることで問題が発生しても、お相手の問題として切り分けすることができる。
  各社がSuperTestを使ってくれると、もっと助かるのだが、、

SuperTest 成果

- ライブラリのテスト

  ・C99 なのにC89のライブラリが混同されていて、sin, cosなどの数学関数で問題になるところを、事前に回避できた。

  ・自動運転などで画像認識等はC++のライブラリに依存することになるので、SuperTestのライブラリテスト機能にも注目している。またロードマップ内のFloat16のテストや、AUTOSAR C++14 対応にも期待している。

SuperTest 成果

* Depth Suite 機能
  ・アーキテクチャ固有のデータ長（例えば char も int も24ビット）に合わせて演算精度のテストができる。

Thank You!

Marcel Beemster
marcel@solidsands.nl

Solid Sands

FUJI SETSUBI