Concolic Testing for High Test Coverage and Reduced Human Effort in Automotive Industry

Yunho Kim, SWTV group
KAIST, South Korea

KAIST Moonzoo Kim

HYUNDAI MOBIS Dongju Lee
Junki Baek

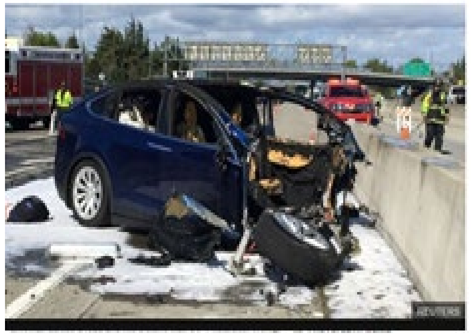
Manual Testing in Automotive Industry: - **Large Human Effort Required**

Manual Testing

- SW reliability is critical for cars

Toyota "Unintended Acceleration" has killed 89

Tesla fatal crash was on Autopilot



- Achieving high reliability requires **large human effort**

…

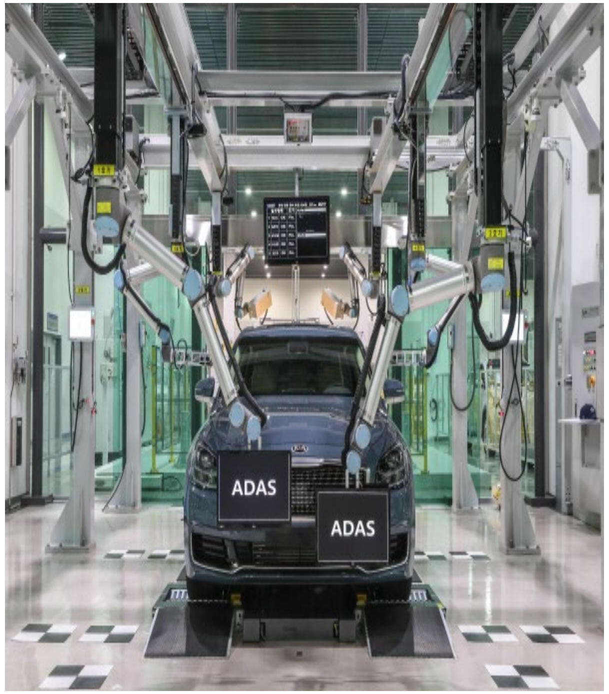
÷

1MLoC / a car * 10 models ÷ 7KLoC/MM

= **120MYr** (for coverage testing)

Automated Testing can **Achieve High Reliability** and **Reduce Human Testing Effort**

Automated Testing



- Automated testing achieves **high reliability**

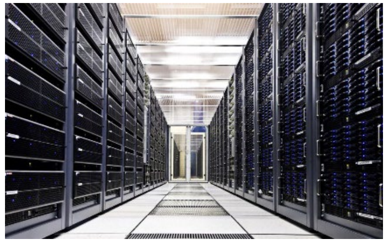
Achieving high code coverage

Detecting hidden bugs

- Automated testing can be **100x faster** than manual one

…

÷



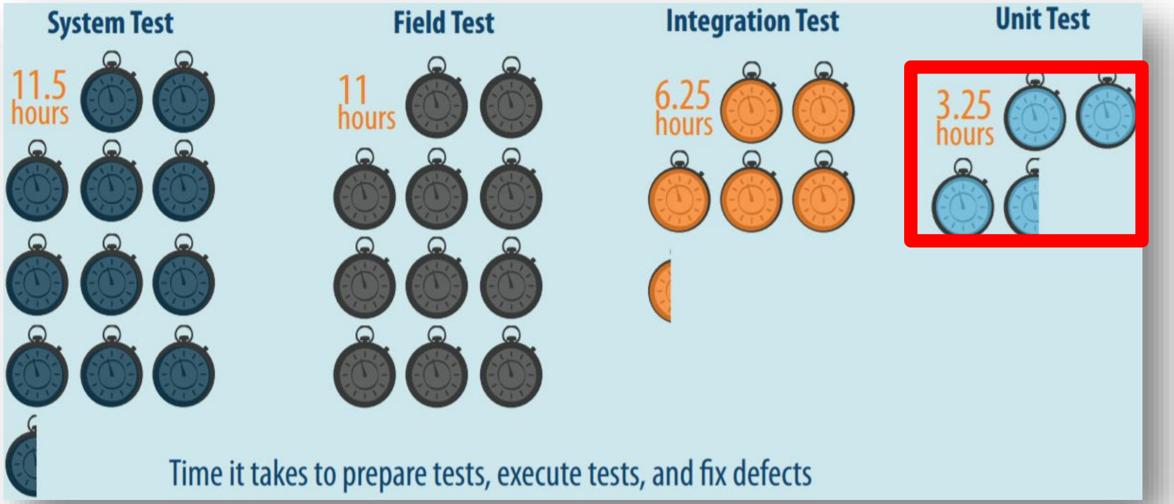
1MLoC/a car * 10 models ÷ **700KLoC/1CM**

= **1.2CYr** (**= 0.1M on 100 cloud cores**)

Benefits of Unit Testing

- **Bug correction cost: 7x cheaper** than system tests
  - $937 (unit test) vs $7,136 (system test)
- **Bug correction time: 3x faster** than system testing
  - 3.25 hours vs 11.5 hours

Source: B. Boehm and V. Basil, Software Defect Reduction Top 10 List, IEEE Computer, January 2001



Source: Capers Jones, Applied Software Measurement: Global Analysis of Productivity and Quality

Excerpted from the CONBRIO paper slides [Kim et al., ICSE18]

Importance of Automated **Unit Testing** In Automotive SW

System testing is **expensive** and **less effective**

- **Full vehicle HW** and **human drivers** are required
- Driving a car with **various physical environments** spends **a lot of time**
- **Hard-to-achieve high test coverage** due to low controllability

VS

Unit testing is **cheap** and **more effective**

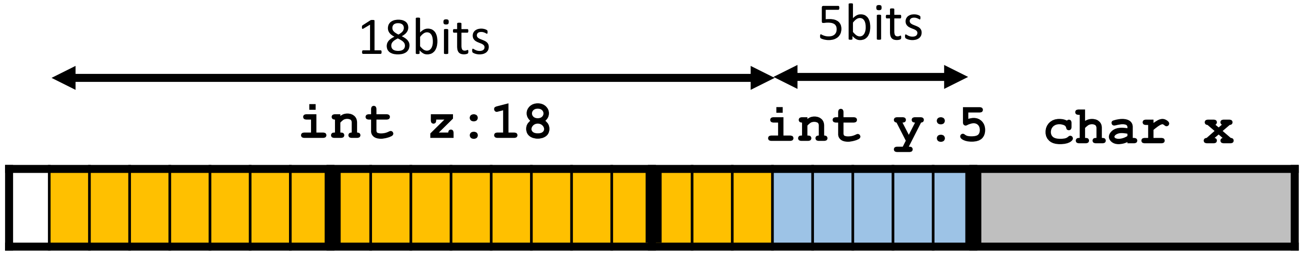
- Full vehicle HW and human drivers are **NOT necessary**
- No need to drive a car
- **Easy-to-achieve high coverage** due to high controllability

3 Technical Challenges for Automated Unit Testing

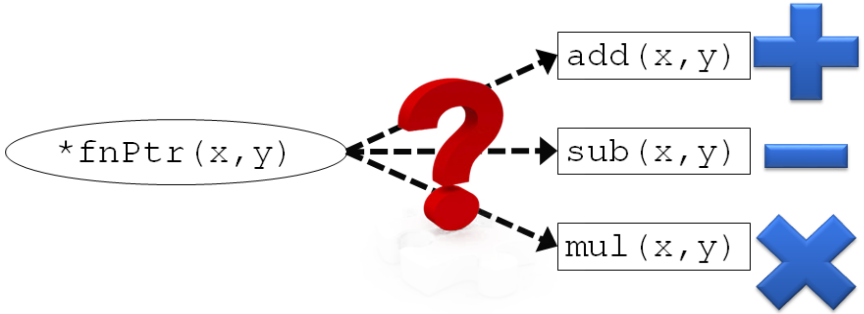
False alarms

Bit-fields

`struct S` : 4 Bytes

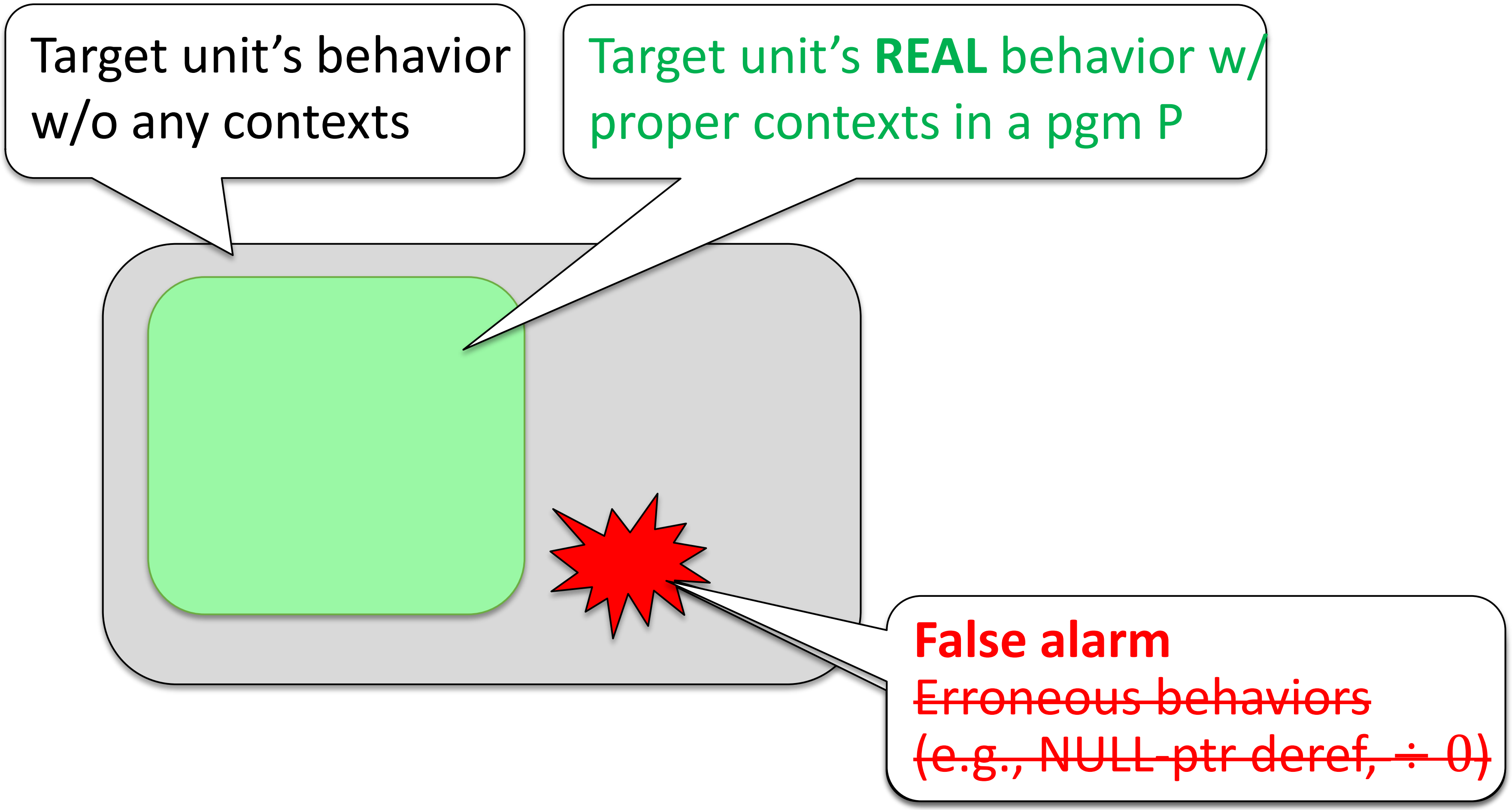


Function pointers



False Alarms due to Infeasible Unit Executions

- **False alarms** can be raised due to missing contexts



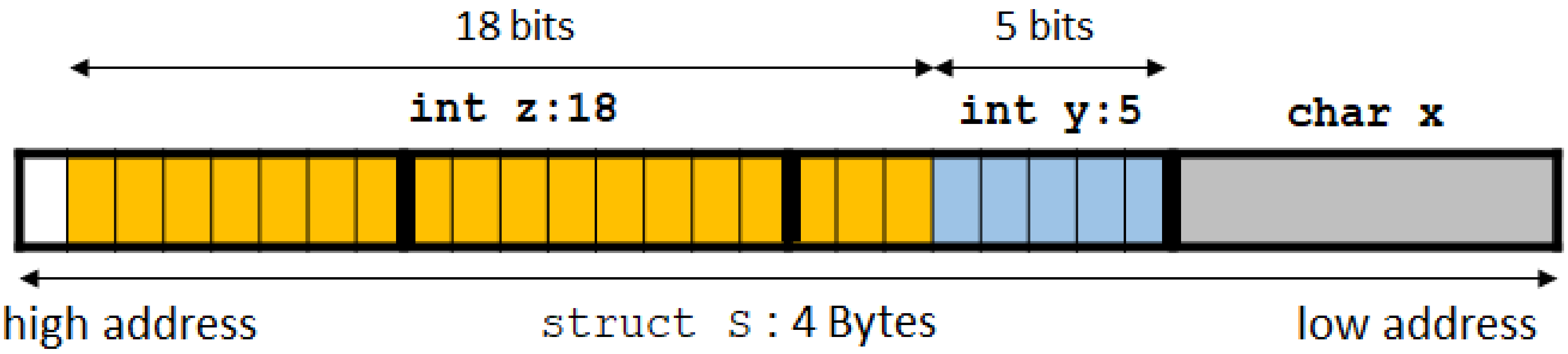
No Support of Symbolic Bit-fields

- Tracking symbolic expression for bit-fields is **NOT** possible
  - bit-fields are **NOT addressable (i.e., &(s.y) is not allowed)**

```
struct S{
   char x;
   int y:5;
   int z:18;};
```

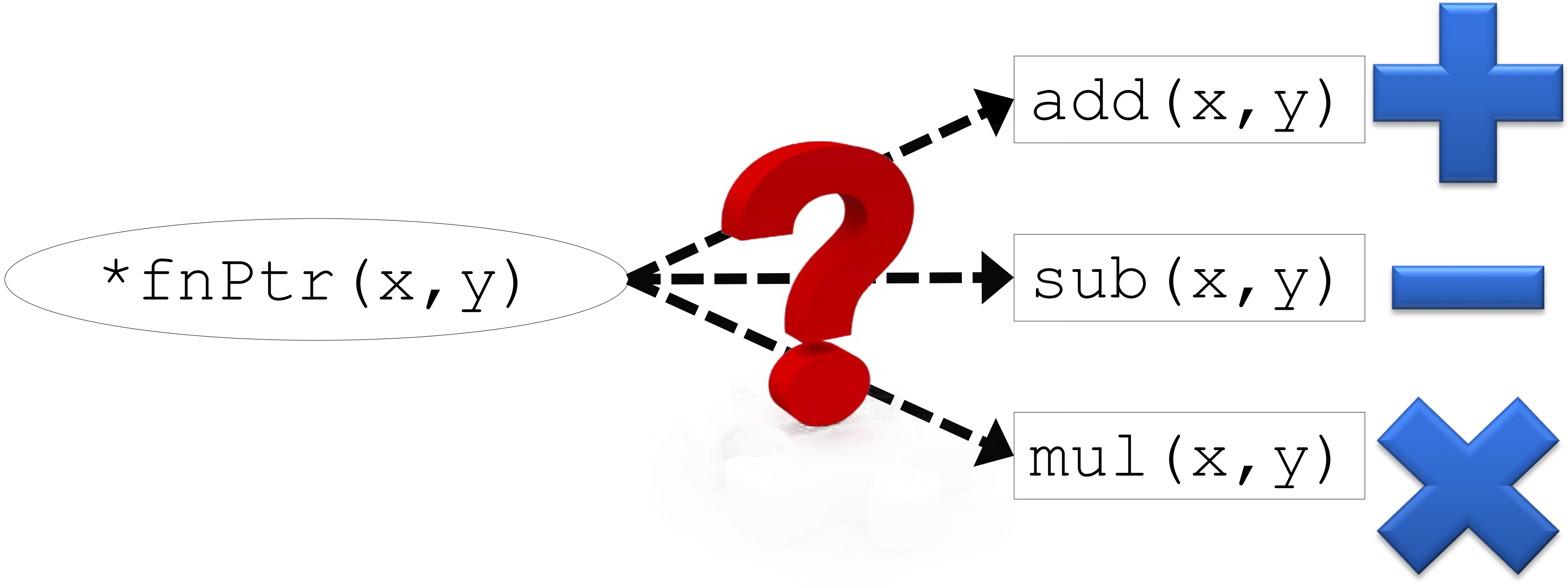
Symbolic Memory Map

Memory address (key)	Sym. Exp. (value)
&(S.x)	$S.x_0+3$
&(S.y) **NOT allowed**	**N/A**
&(S.z) **NOT allowed**	**N/A**



No Support of Symbolic Function Pointers

› Hard-to-know which function a given func. ptr. points to

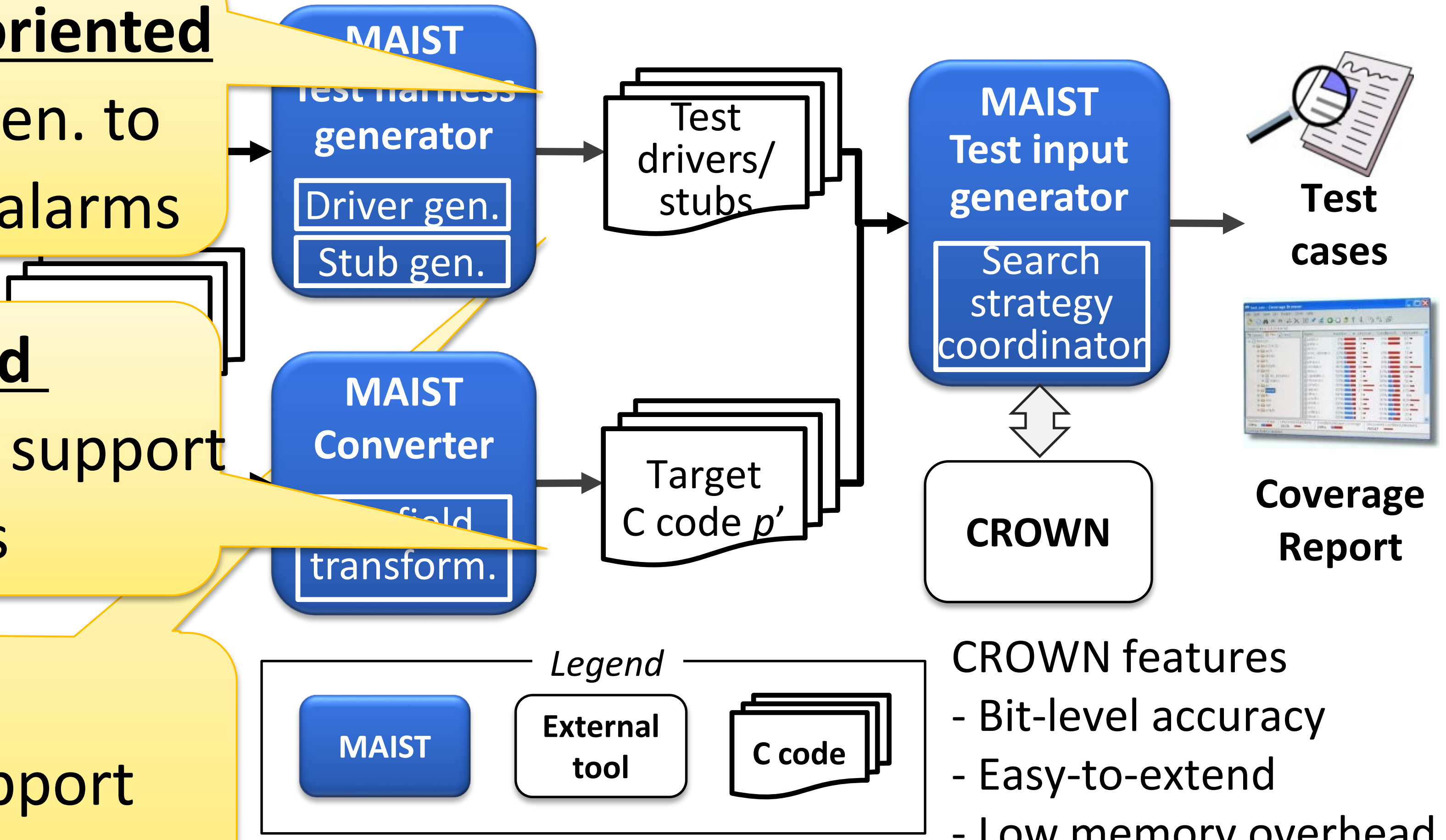


Overview of MAIST

Sol. 1: **<u>Task-oriented</u>** driver/stub gen. to reduce false alarms

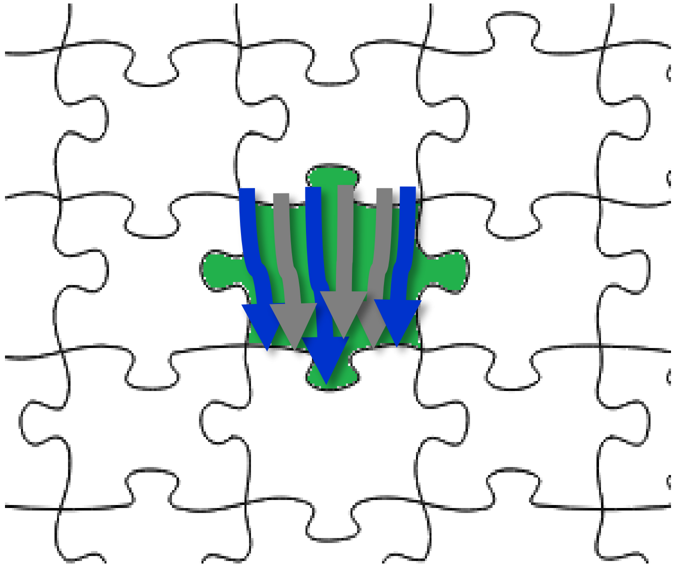
Sol. 2: **<u>Bit-field transform</u>**. to support sym. bit-fields

Sol. 3: Static analysis to support **<u>sym. function pointer</u>**

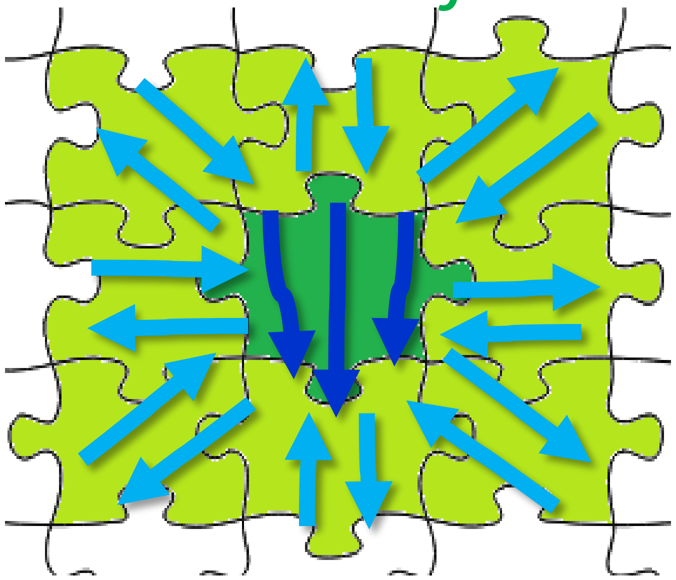


CROWN features
- Bit-level accuracy
- Easy-to-extend
- Low memory overhead
- Fast through lightweight instr.

Unit Testing *f* without Contexts of *f*



Unit Testing *f* with Contexts of *f*



Without Contexts of *f*

Pros: fast exploration of target unit execution paths

Cons: infeasible target unit executions

With Contexts of *f*

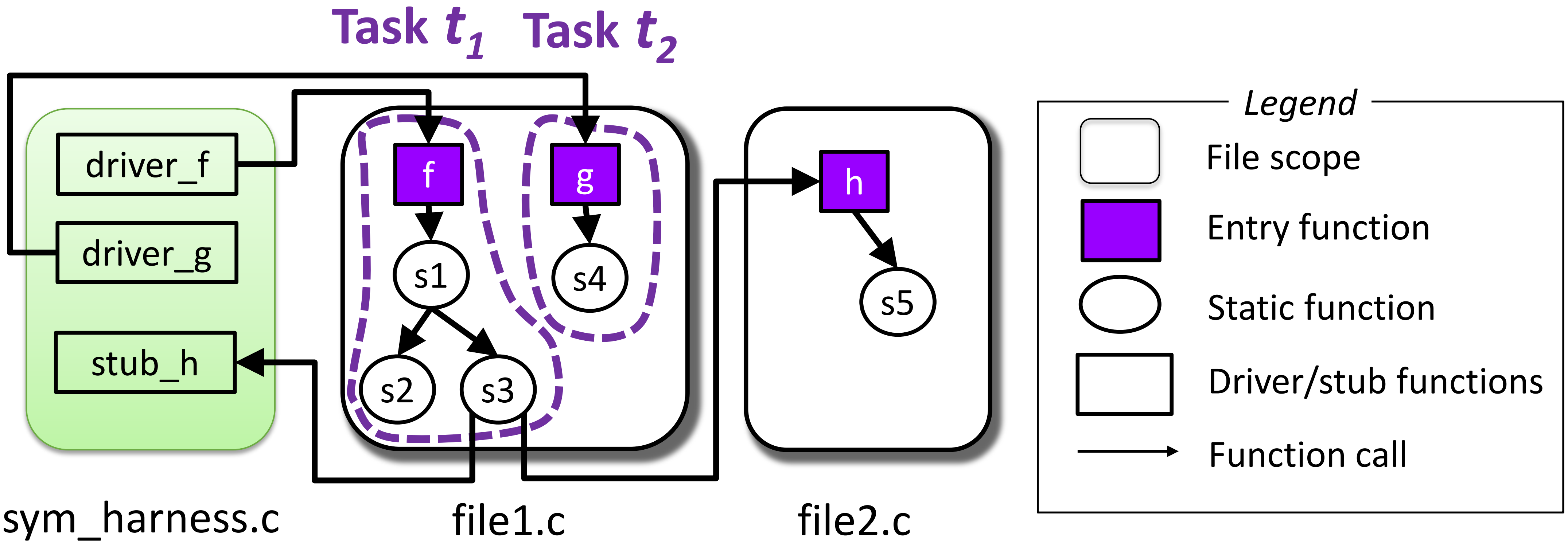
Pros: reduced infeasible target unit executions

Cons: slow exploration of target unit execution due to large cost of exploring context functions

Excerpted from the CONBRIO paper slides [Kim et al., ICSE18]

Task-oriented Unit-test Driver/Stubs Generation

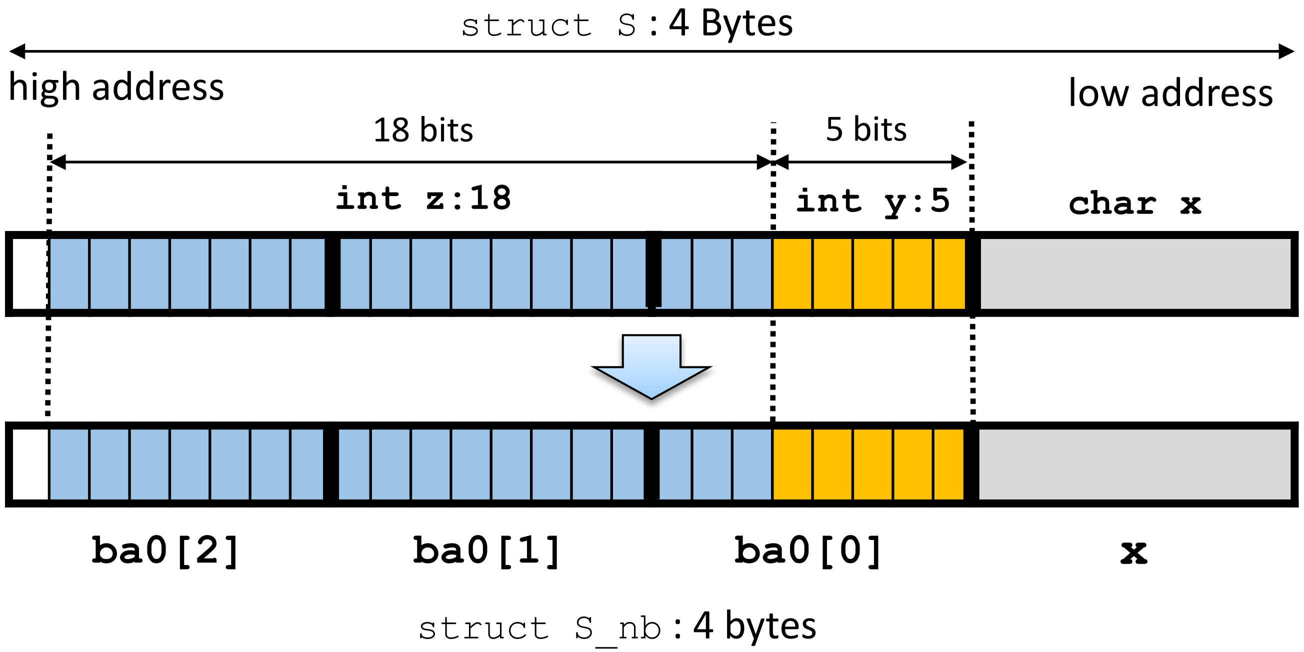
- A ***Task*** is a mostly **minimal independent unit**



Bit-field Transformation

```
struct S{
  char x;
  int y:5;
  int z:18;};
```

```
struct S_nb{
  char x;
  unsigned char ba0[3];
} ;
```



Original program *p*	Transformed program *p'*
`n=S.z+1;`	`n=(((S_nb.ba0[2]&127)<<16) | (S_nb.ba0[1]<<8) | S_nb.ba0[0])>>5 +1;`

`S.z=S.x+3;`	`S_nb . ba0 [2]= ((((S_nb.x+3) <<5) > >(2*8)) &127)| (S_nb .ba0 [2]&128);`
	
	`S_nb . ba0 [1]= (((S_nb.x+3) <<5) > >(1*8)) &255;`
	
	`S_nb . ba0 [0]= (((S_nb.x+3) <<5) &224) | ( S_nb .ba0 [0]&31);`

Symbolic Function Pointer Support

Step 1: Identifying candidate functions using static analysis

```
target.c
 1:int (*fnPtr)(int,int);
 2:int add(int,int) {…}
 3:int sub(int,int) {…}
 3:int mul(int,int) {…}
 4:void f() {…
 5:  fnPtr=add;}
 6:int g(){…
 7:  fnPtr=sub;}
 7:int h(){…
 8:  fnPtr=mul;}
 8:void do_chk(int
 9:…
10: int ret=fnPtr(x,y);
11: …}
```

Function pointer input

Candidate1: add()

Candidate2: sub()

Candidate3: mul()

Step 2: Set *fnPtr as one of the candidates using a symbolic var sym_choice

```
harness_target.c
 1:void drv_do_chk(){
 2: …
 3:   SYM_int(sym_choice);
 4:   switch(sym_choice){
 5:   case 0:
 6:     fnPtr=add;
 7:     break;
 8:   case 1:
 9:     fnPtr=sub;
10:     break;
11:   case 2:
12:     fnPtr=mul;
13:   default: break;}
14:   SYM_int(p1);
15:   do_chk(p1);}
```

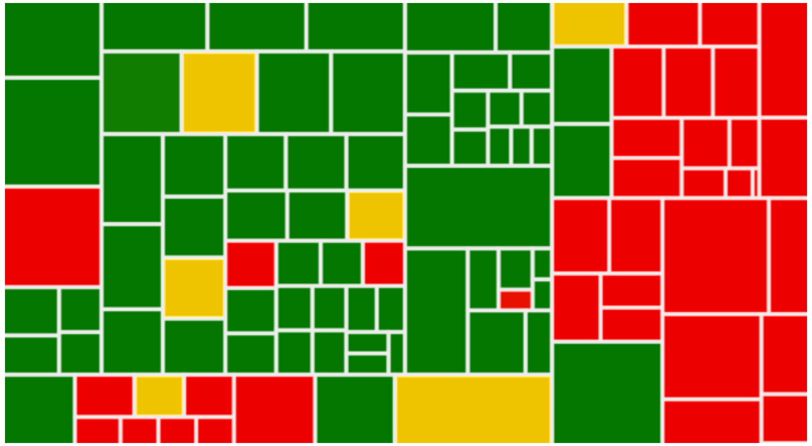
Set *fnPtr as add() if sym_choice==0

Set *fnPtr as sub() if sym_choice==1

Set *fnPtr as mul() if sym_choice==2

Research Questions

› RQ1-2: Overall benefit of MAIST

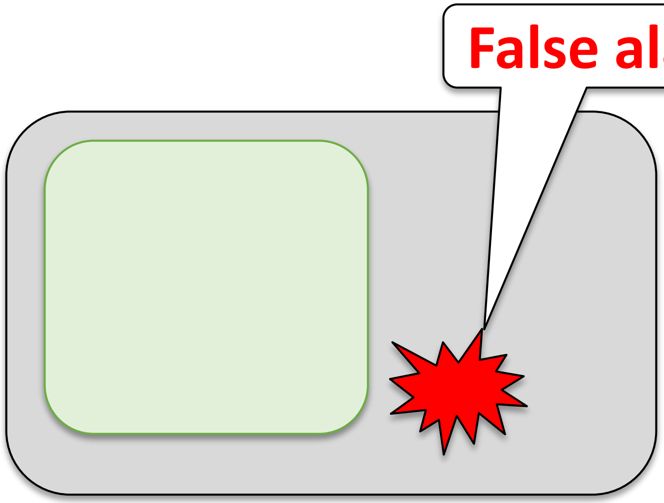


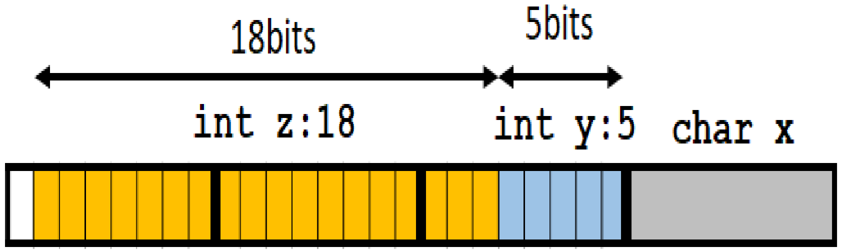
RQ1: **How much**
test coverage
does MAIST achieve?

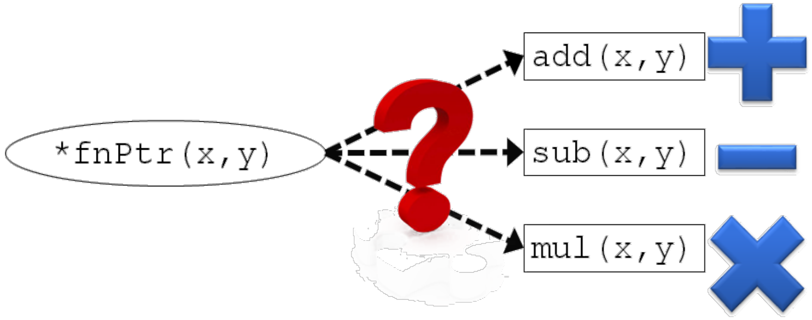
RQ2: **How much**
human test effort
does MAIST reduce?

Research Questions

› RQ3-5: Effectiveness of MAIST for addressing the technical difficulties







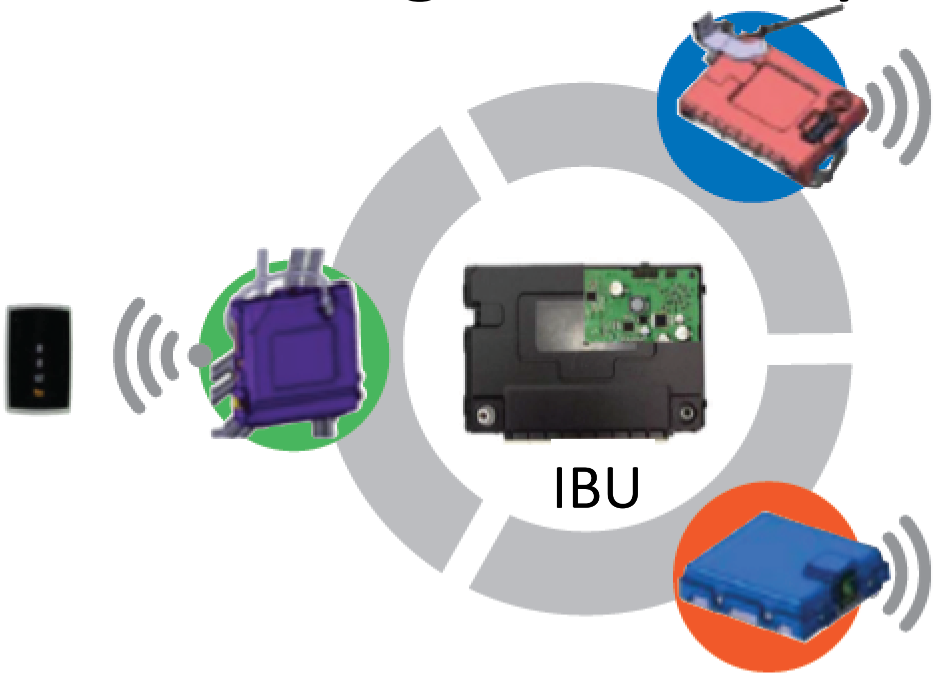
RQ3: **Effect of Task-oriented test Gen.**

RQ4: **Effect of Sym. bit-fields**

RQ5: **Effect of Sym. Func. Ptr.**

Overview of Target SW: Integrated Body Unit

- Smart Key (SMK)
  - Remote door lock/unlock
  - Button start



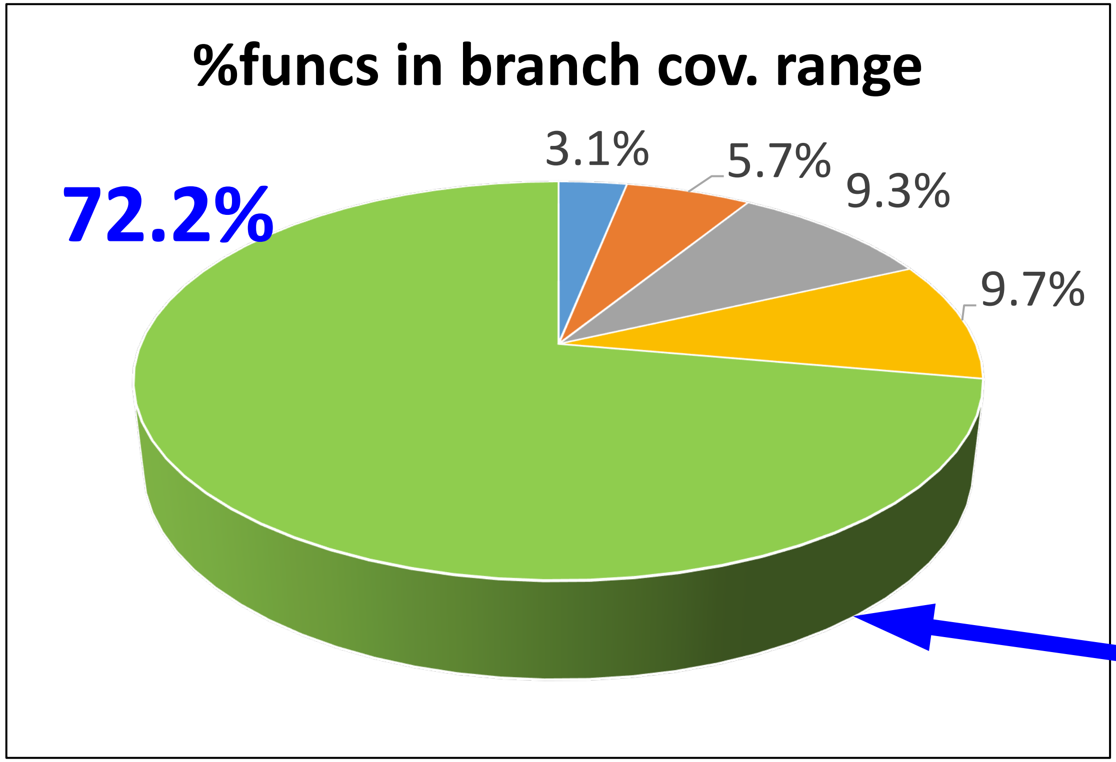
- Body Control Module (BCM)
  - Wiper, door control
  - Burglar alarm
- Tire Pressure Monitor (TPM)
  - Tire pressure sensing

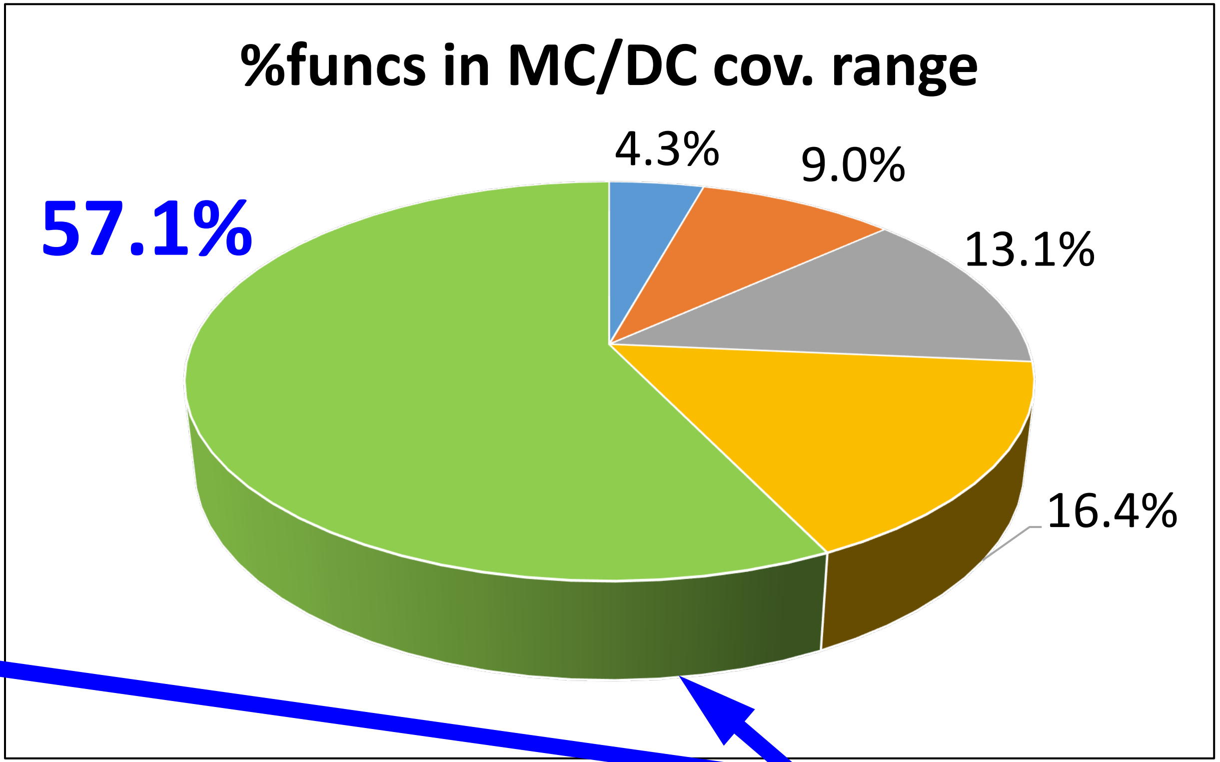
Module	#files	#tasks	#functions	LoC	Avg. CC per func.
BCM	27	143	656	53,690	4.3
SMK	198	554	2,521	135,877	5.8
TPMS	29	68	302	16,951	4.1
Total	**254**	**767**	**3,479**	**206,518**	**4.9**

RQ1:MAIST Achieved **90.5% Branch** and **77.8% MC/DC Cov.**

100% branch cov. of 72.2% of funcs

100% MC/DC cov. of 57.1% of funcs

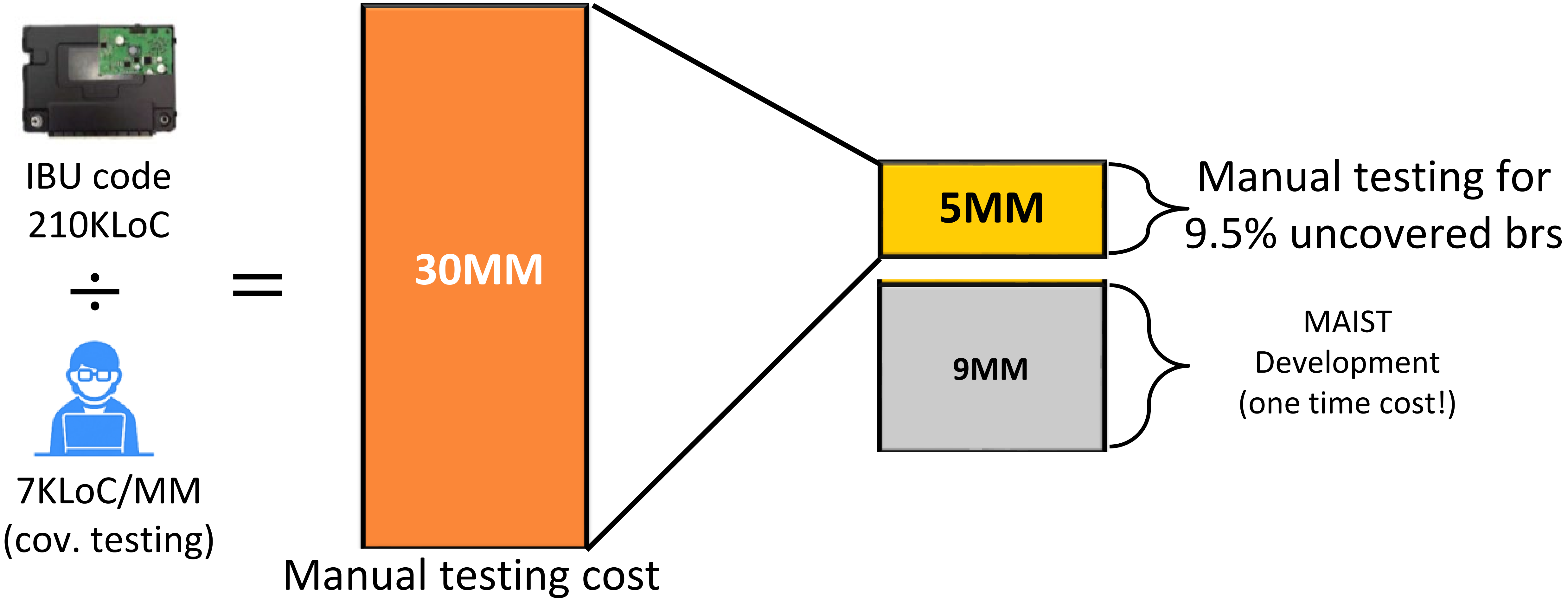




[20%,40%) [40%,60%) [60%,80%) [80%,100%) 100%

* Running 20 hours on 12 CPU cores (3.0GHz)

RQ2. MAIST Reduced Testing Cost from 30MM to 5MM (↓83.3%)

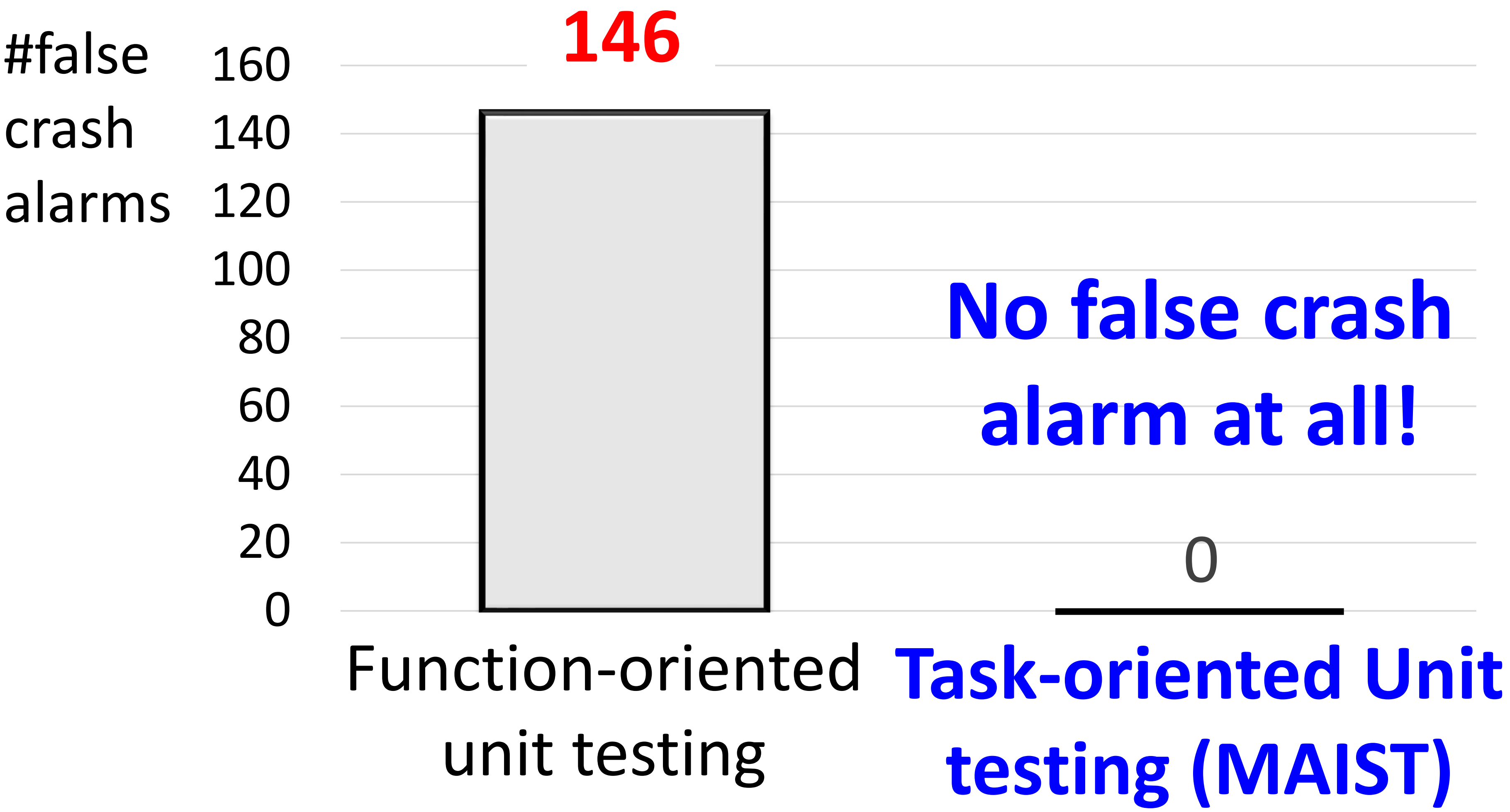


RQ2: Analysis of the Uncovered Branches

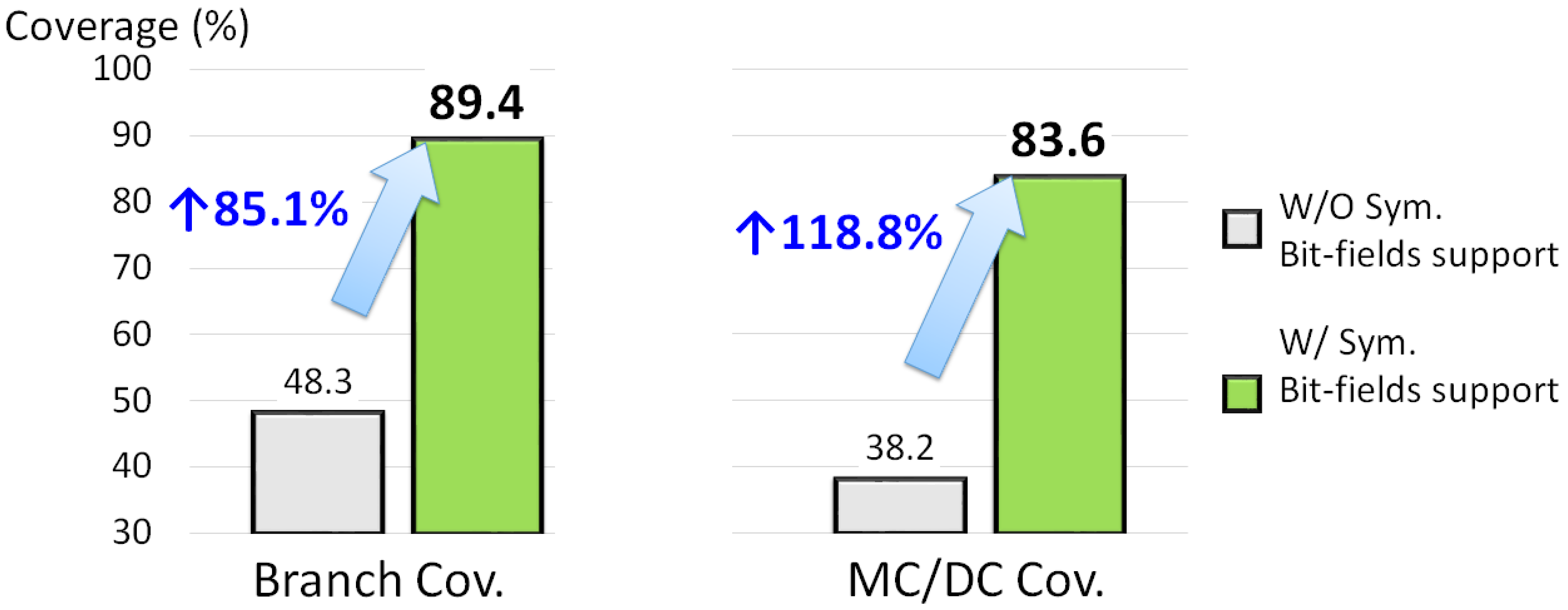
- **Imprecise stub** is the main cause of uncovered branches
  - Manually analyze 33 BCM functions having <60% br. Cov.

Reasons	#uncovered brs	Ratio (%)
Imprecise stub	43	41.7
Path explosion	21	20.4
Static local variable	21	20.4
Imprecise driver	10	9.7
Unreachable branches	8	7.8
Total	**103**	**100**

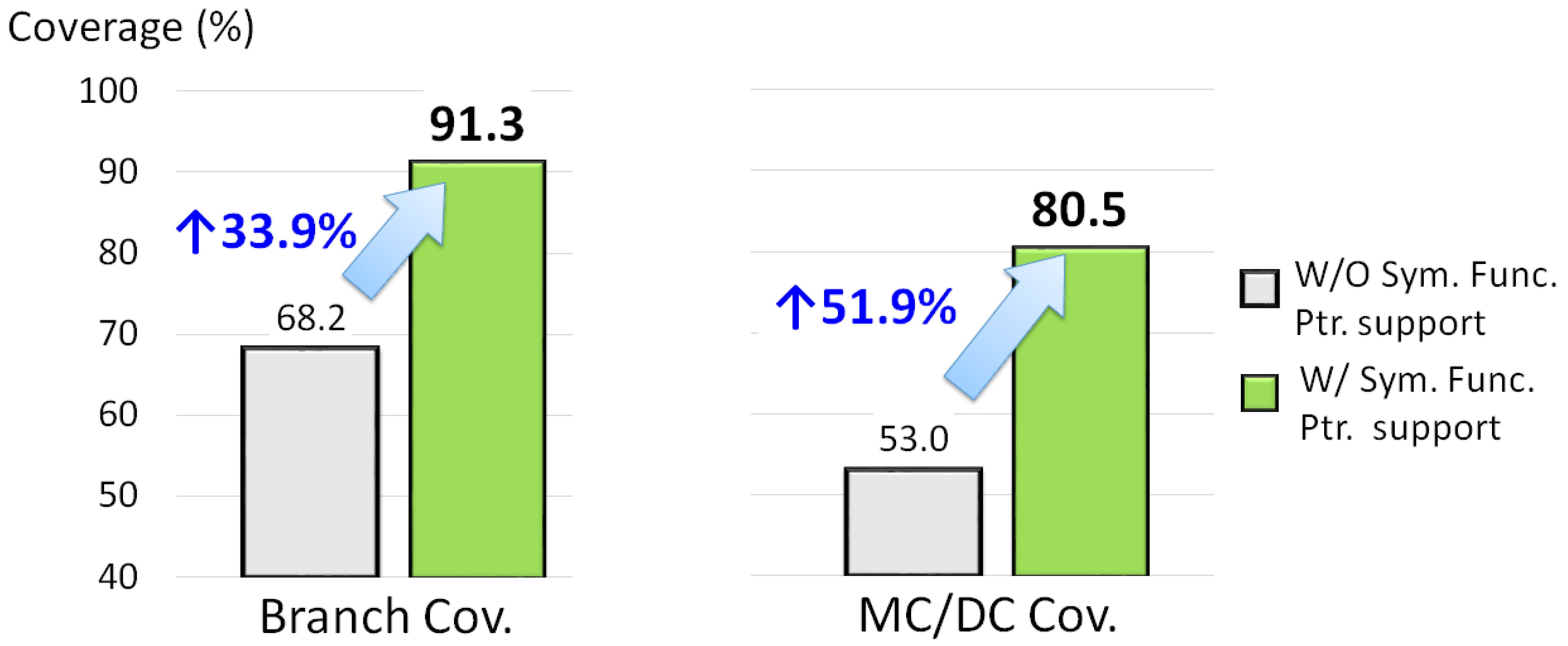
RQ3: **No false crash alarms** by Task-oriented driver/stub generation



RQ4: **Symbolic Bit-fields Support** Highly Increases Test Coverage



RQ5: **Sym. Function Pointer Support** Increases Test Coverage



Conclusion

Manual Testing in Automotive Industry:
- **Large Human Effort Required**

Manual Testing

- SW reliability is critical for cars

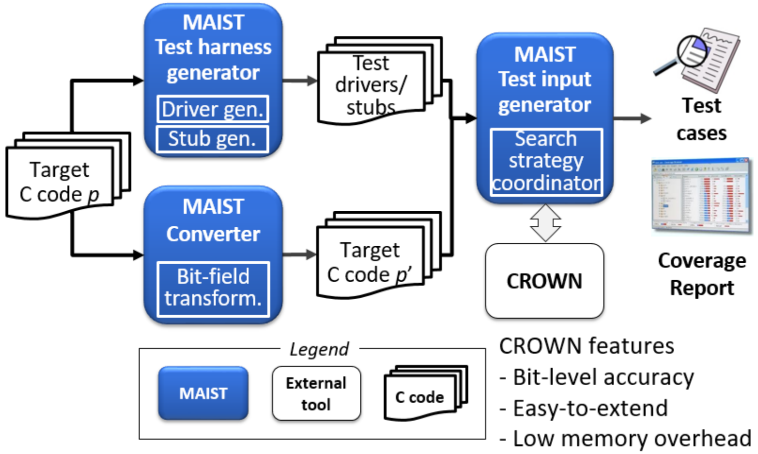
Toyota "Unintended Acceleration" has killed 89

Tesla fatal crash was on Autopilot

- Achieving high reliability requires **large human effort**

1MLoC / a car * 10 models ÷ 7KLoC/MM
= **120MYr** (for coverage testing)

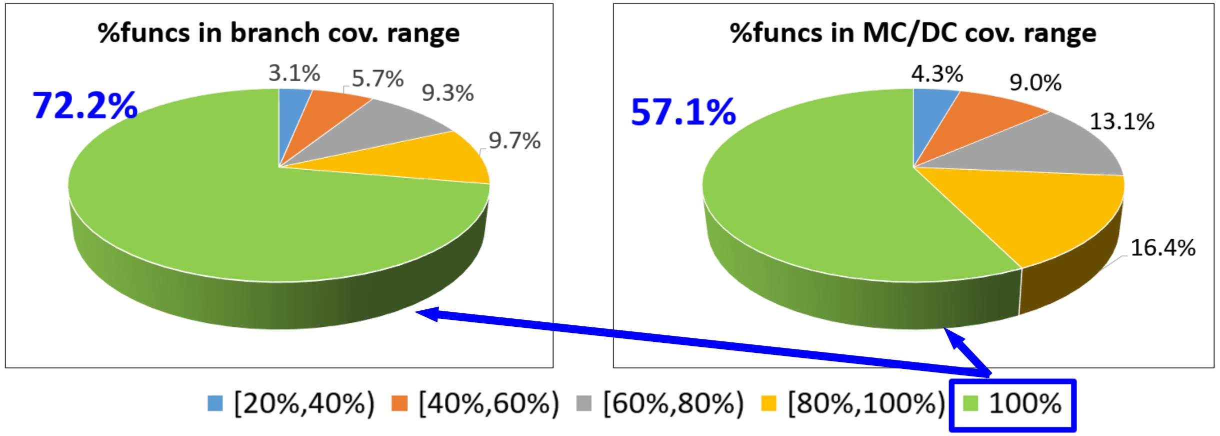
Overview of MAIST



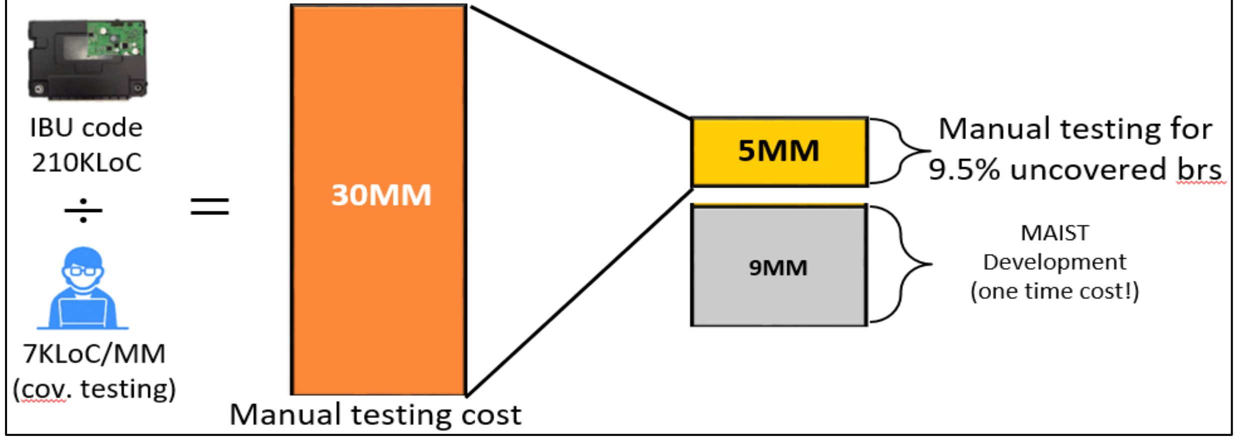
CROWN features
- Bit-level accuracy
- Easy-to-extend
- Low memory overhead
- Fast through lightweight instr.

RQ1:MAIST Achieved **90.5% Branch** and **77.8% MC/DC Cov.**

100% branch cov. of 72.2% of funcs **100% MC/DC cov. of 57.1% of funcs**



RQ2. MAIST Reduced Testing Cost from **30MM** to **5MM (↓83.3%)**



Concolic Testing for High Test Coverage and Reduced Human Effort in Automotive Industry

Yunho Kim, SWTV group
KAIST, South Korea

KAIST Moonzoo Kim

HYUNDAI MOBIS Dongju Lee
Junki Baek