



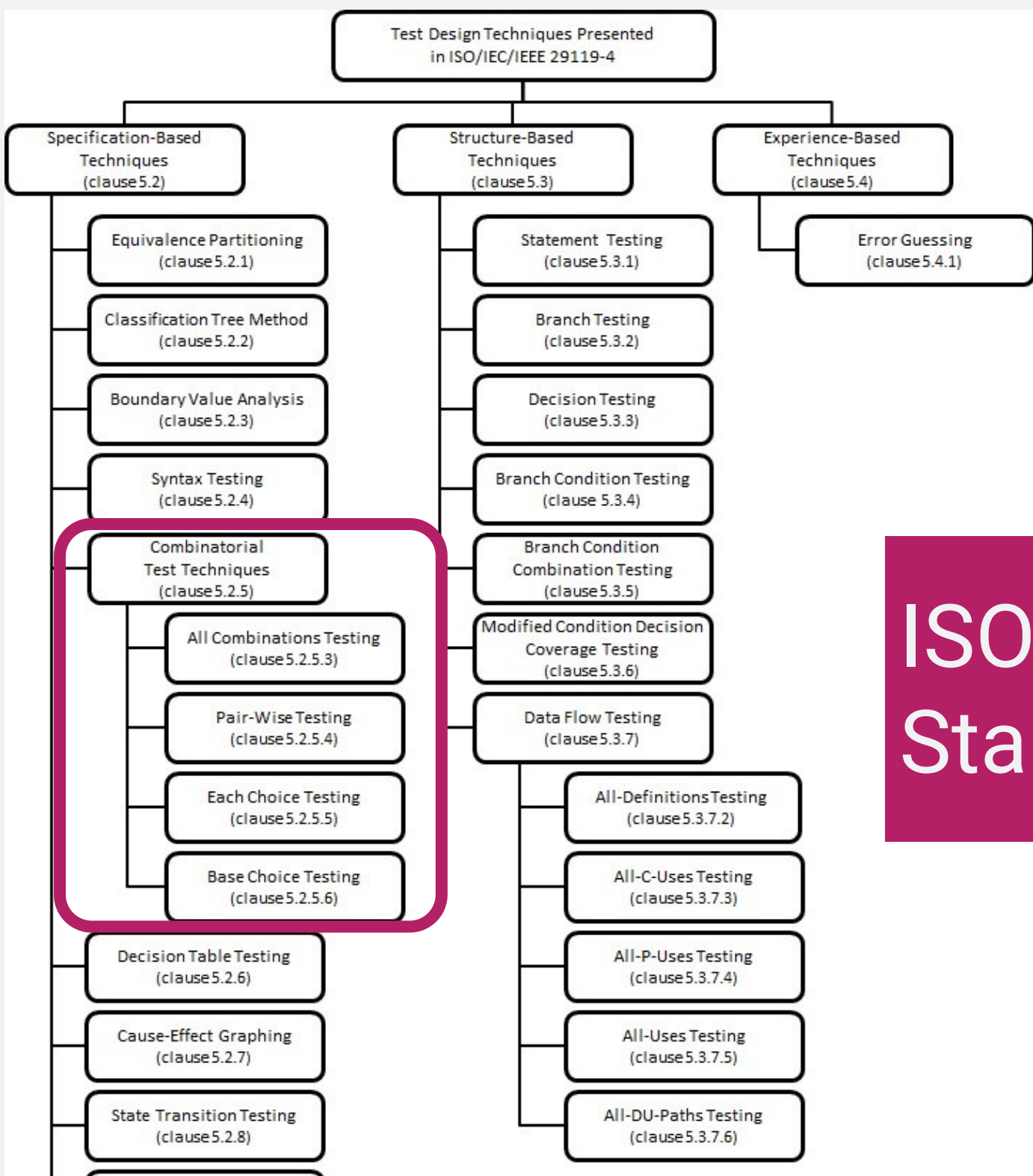
# 组合测试中的故障定位问题

计算机科学与技术系

钮鑫涛

# 「研究背景」

# 组合测试是一种测试因素之间交互的黑盒测试方法



ISO/IEC/IEEE 29119 Software Testing Standard

# 组合测试

Configuration Testing

Input Parameter Testing

Event Driven Software (GUI)

Software Product Line

Concurrent Program Testing

Web Service Testing

Security Testing

App Testing

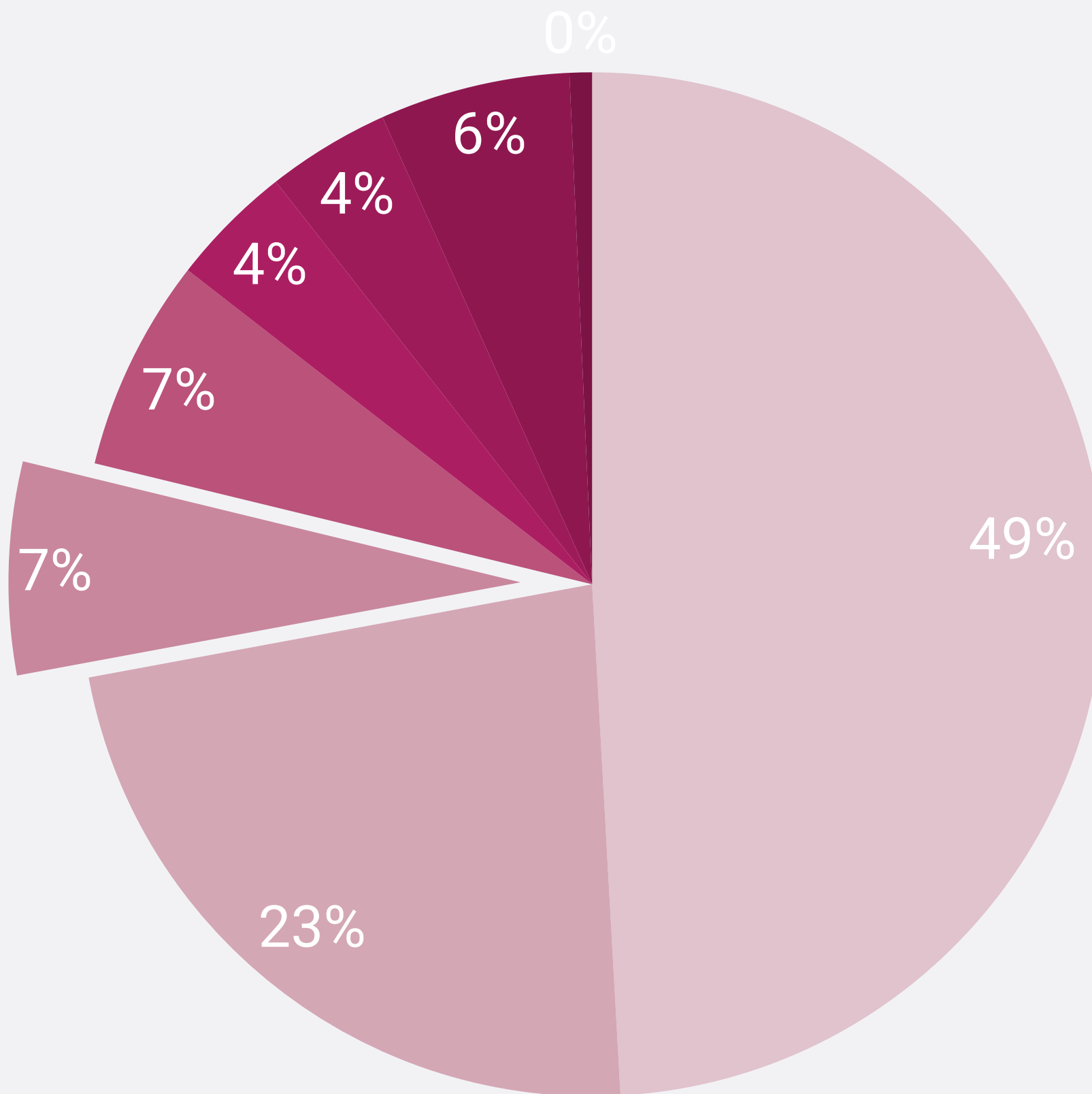
...



LOCKHEED MARTIN



# 组合测试



# 什么是组合测试的故障定位？



# 什么是组合测试的故障定位？



# 什么是组合测试的故障定位?

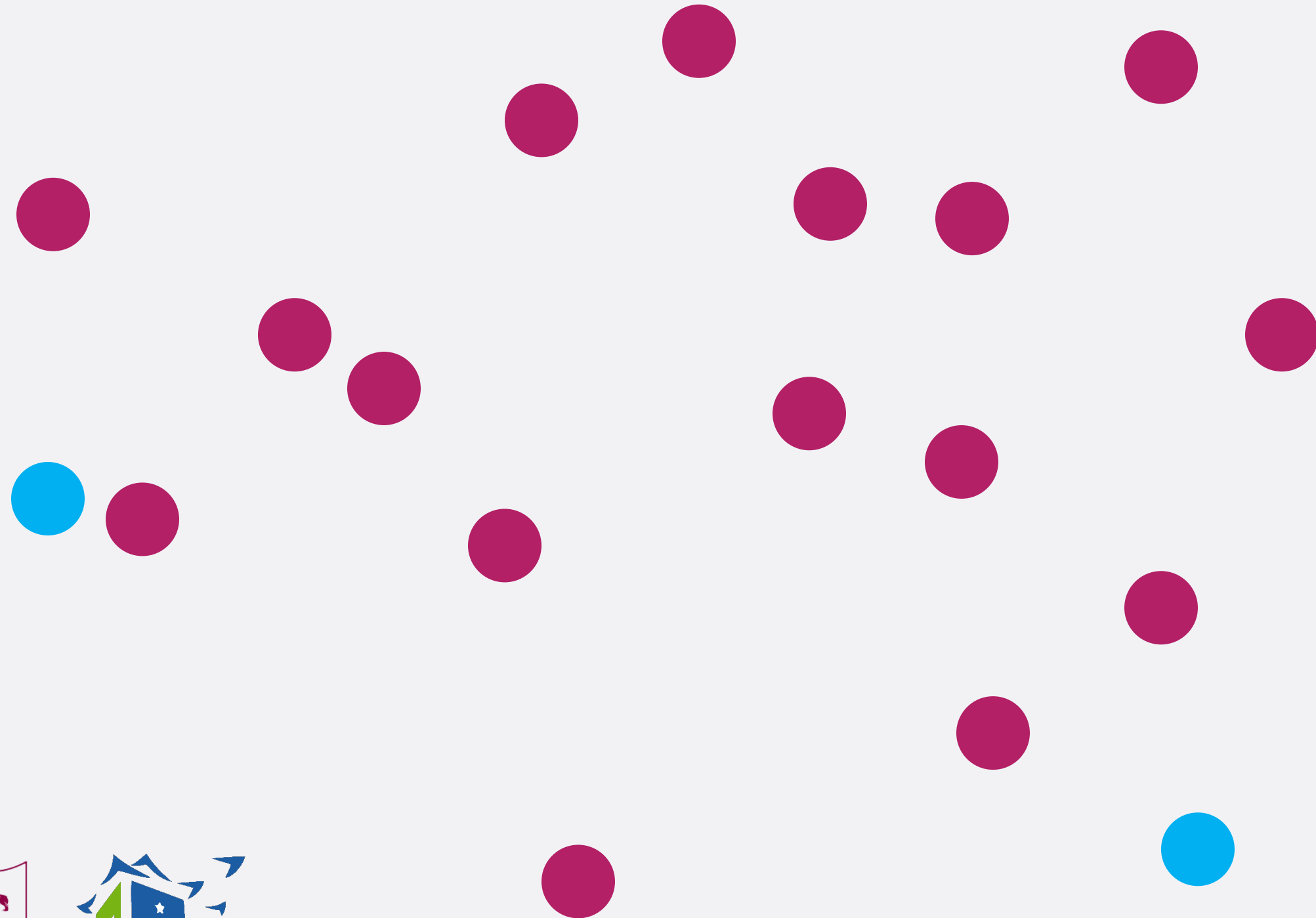




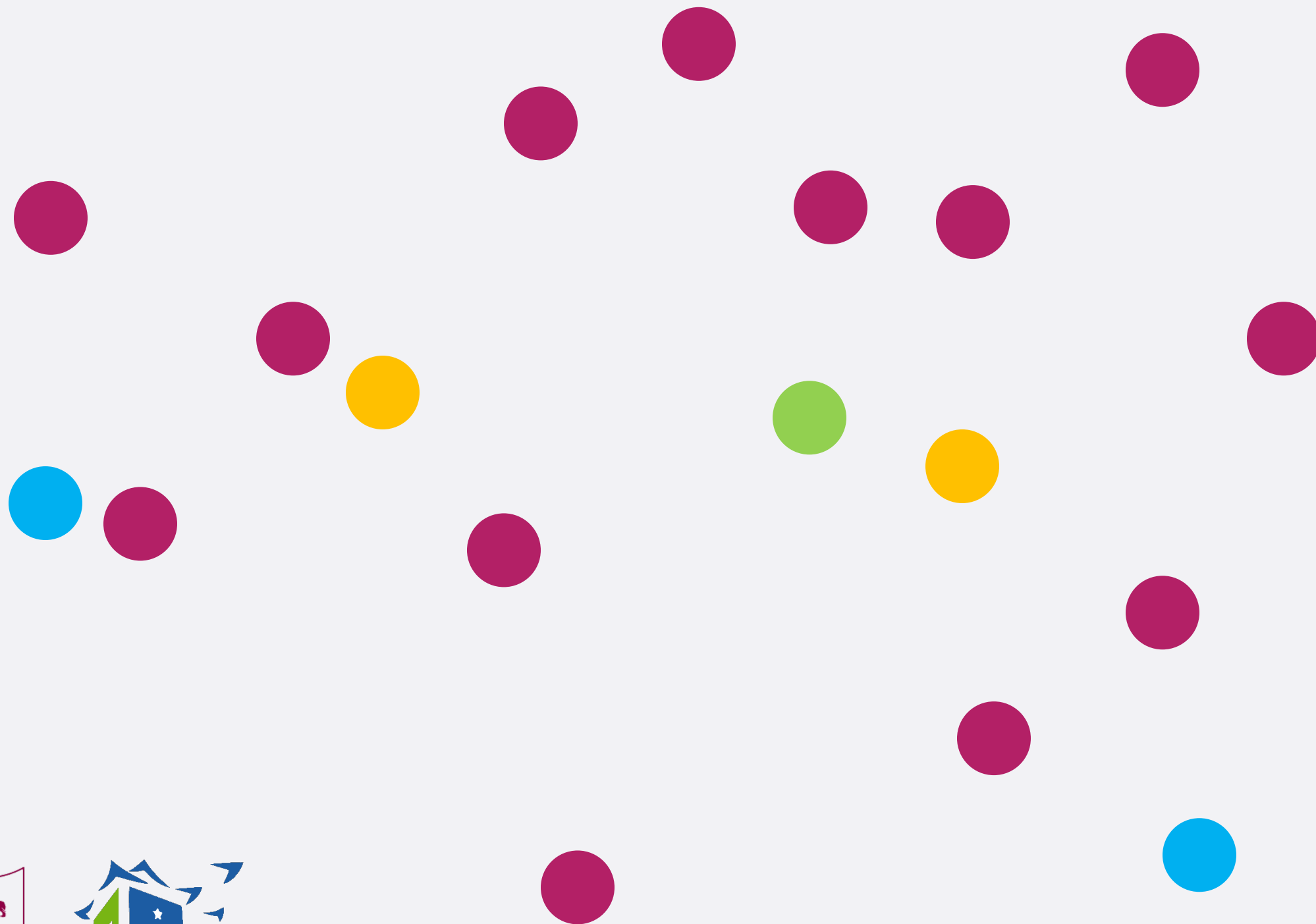
# 什么是组合测试的故障定位?



# 什么是组合测试的故障定位?



# 什么是组合测试的故障定位?



# 什么是组合测试的故障定位？

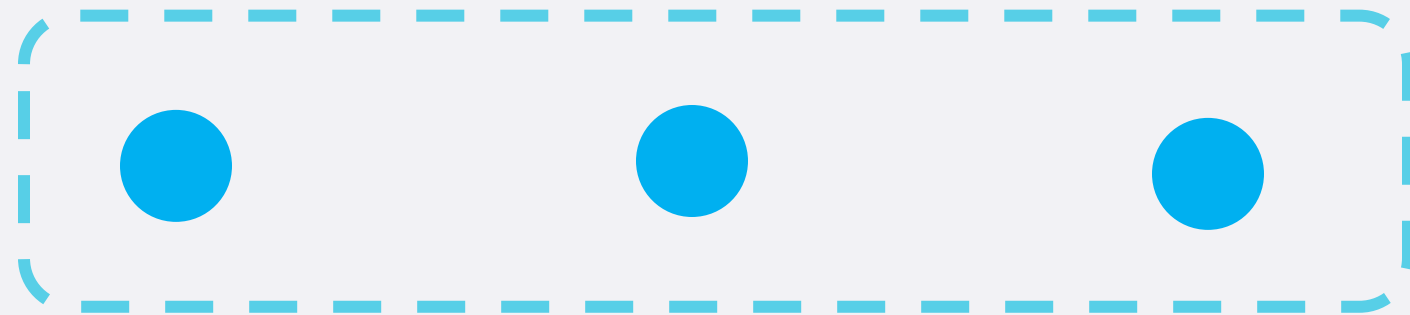


1-degree



2-degree

**模式(Schema)**



3-degree



# 为什么要定位这些模式？



这些模式是触发故障的关键因素 [Nie 2011]



可以有效减少代码审查的范围 [Ghandehari 2012]



避免重复bug report [Zeller 2002]



有利于debugging和修复 [Song 2012]



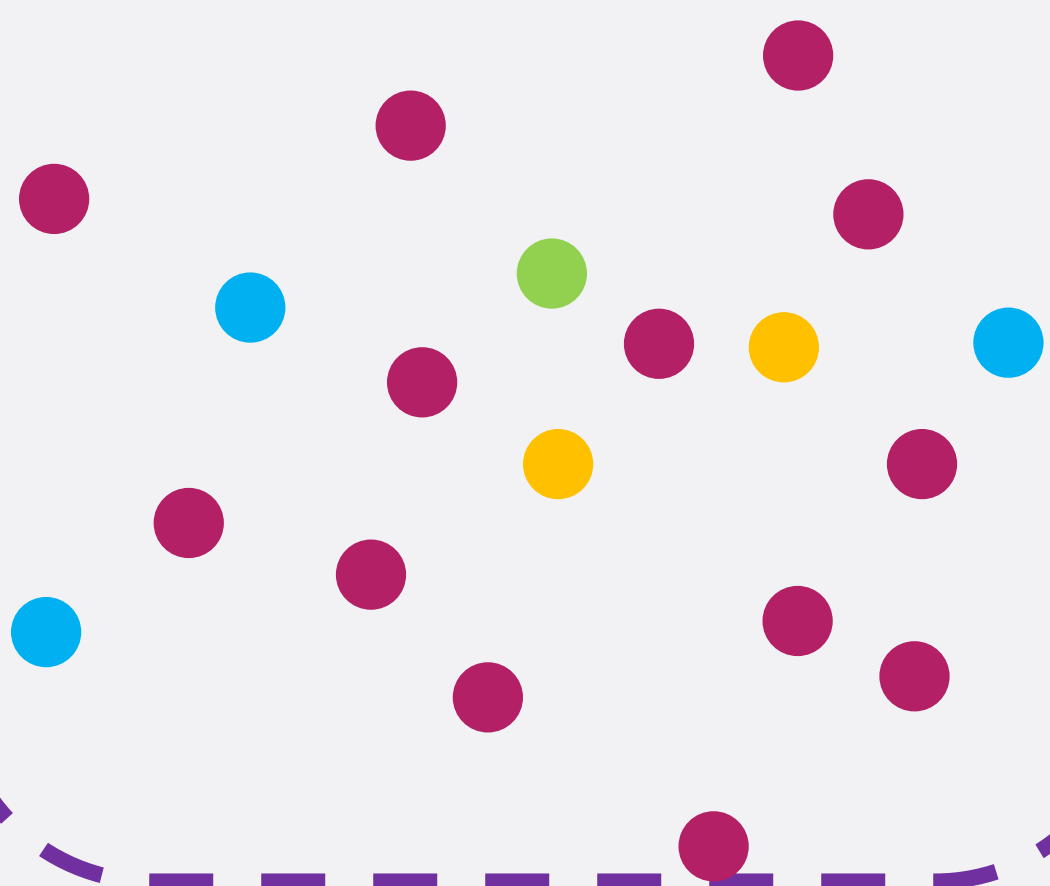
# 存在挑战



$2^n$  模式

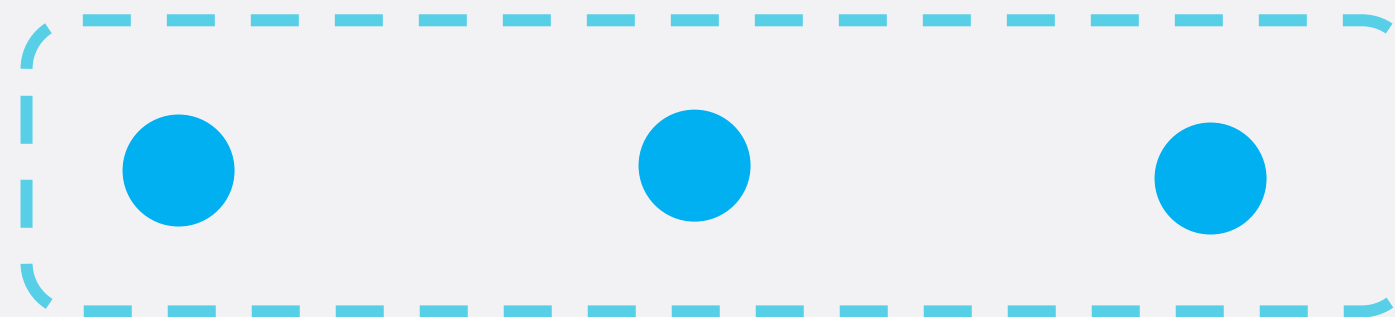
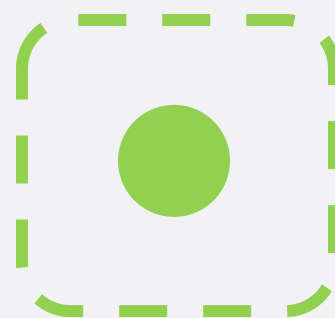
# 存在挑战

解空间巨大  $2^n$



# 存在挑战

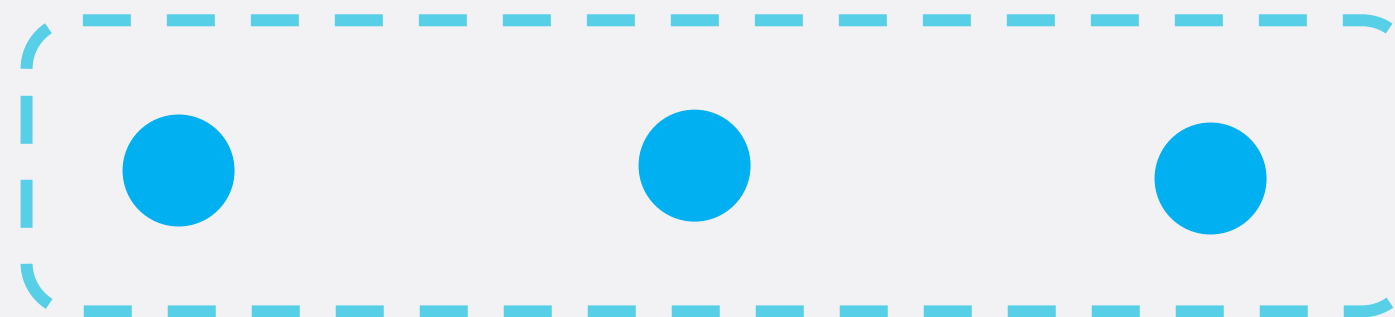
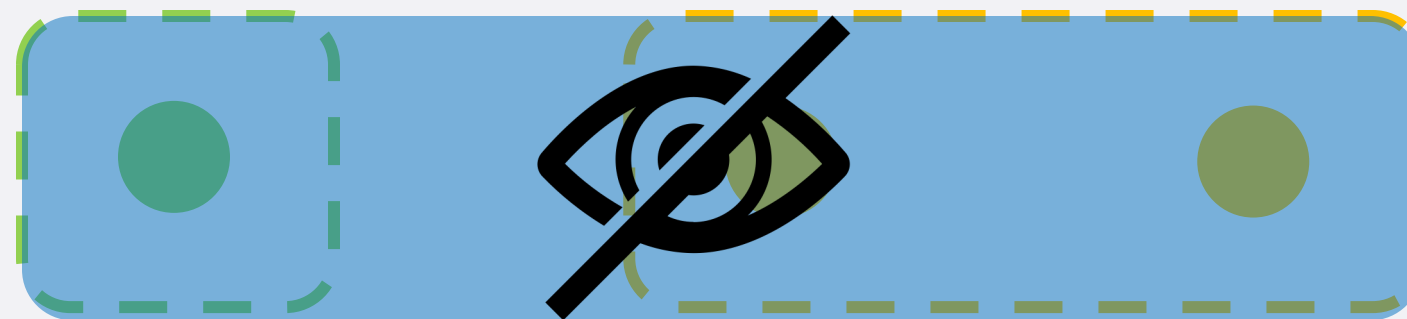
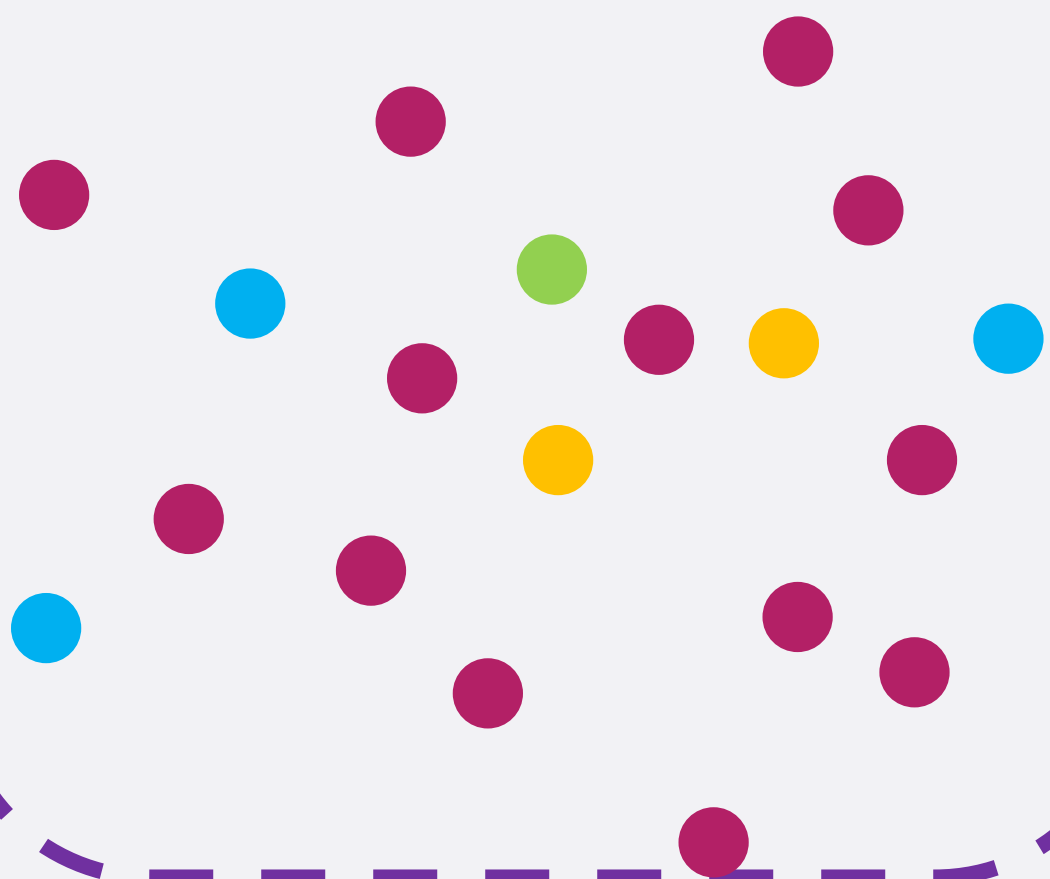
解空间巨大  $2^n$





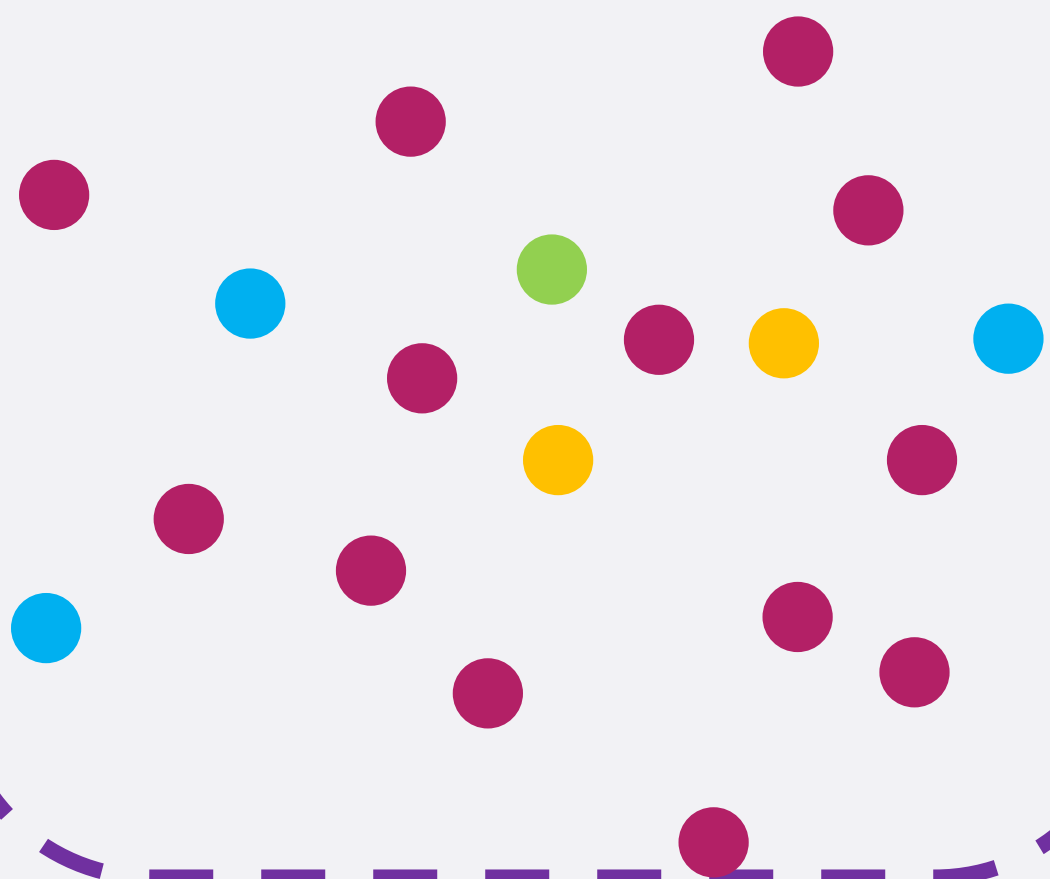
# 存在挑战

解空间巨大  $2^n$

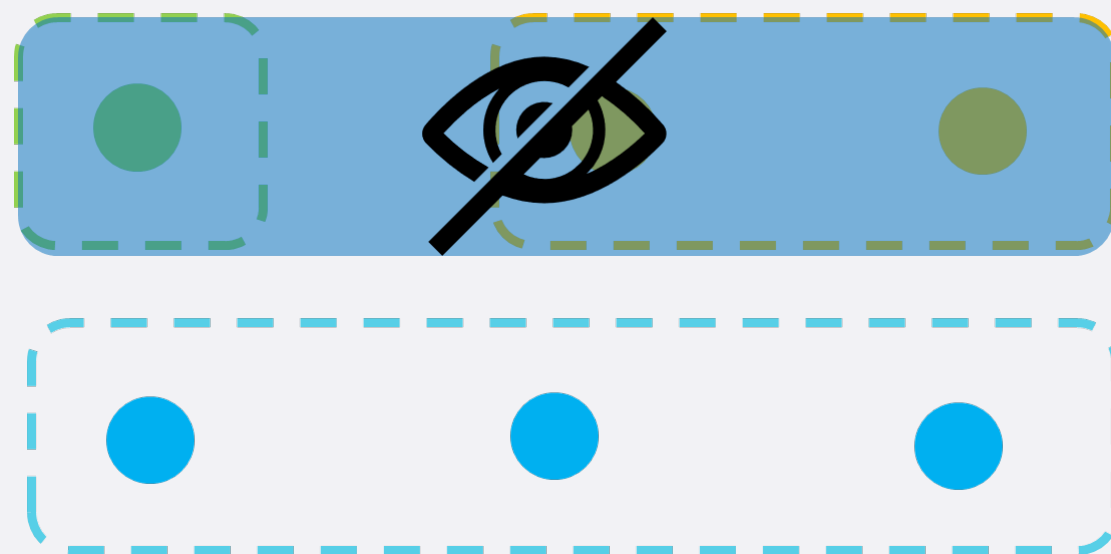


# 存在挑战

解空间巨大  $2^n$

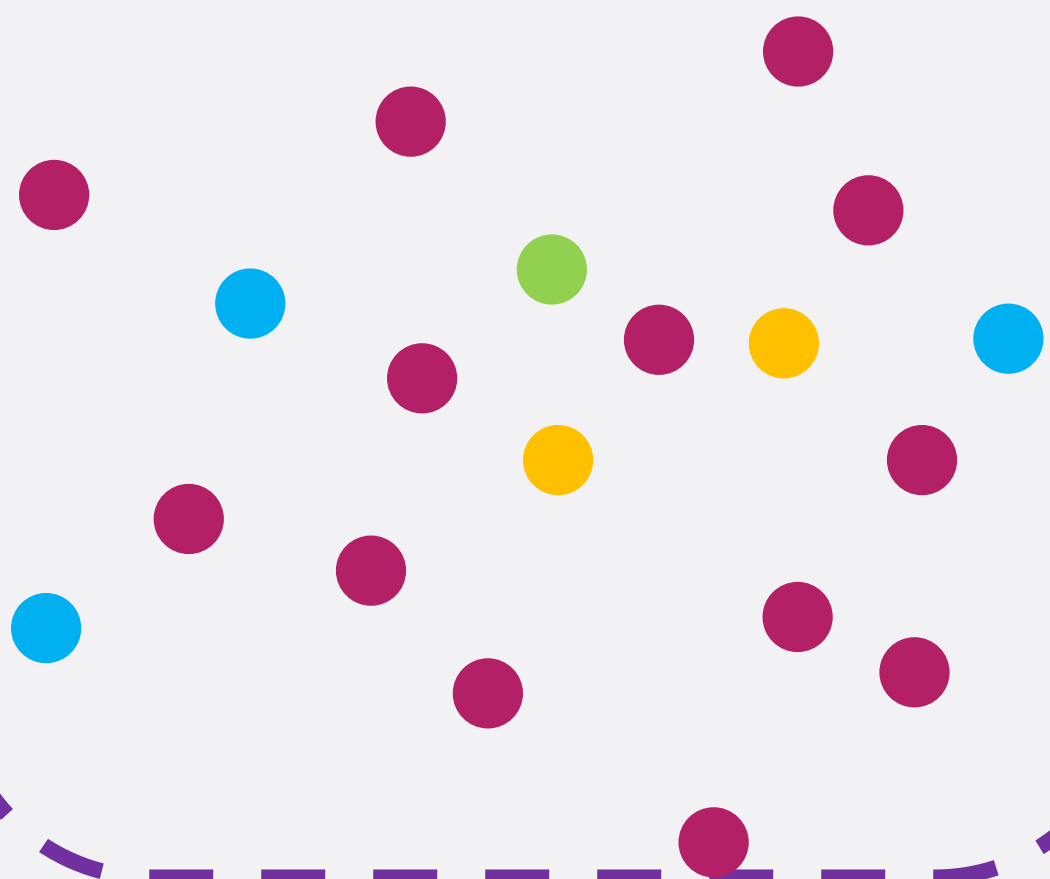


多故障掩盖

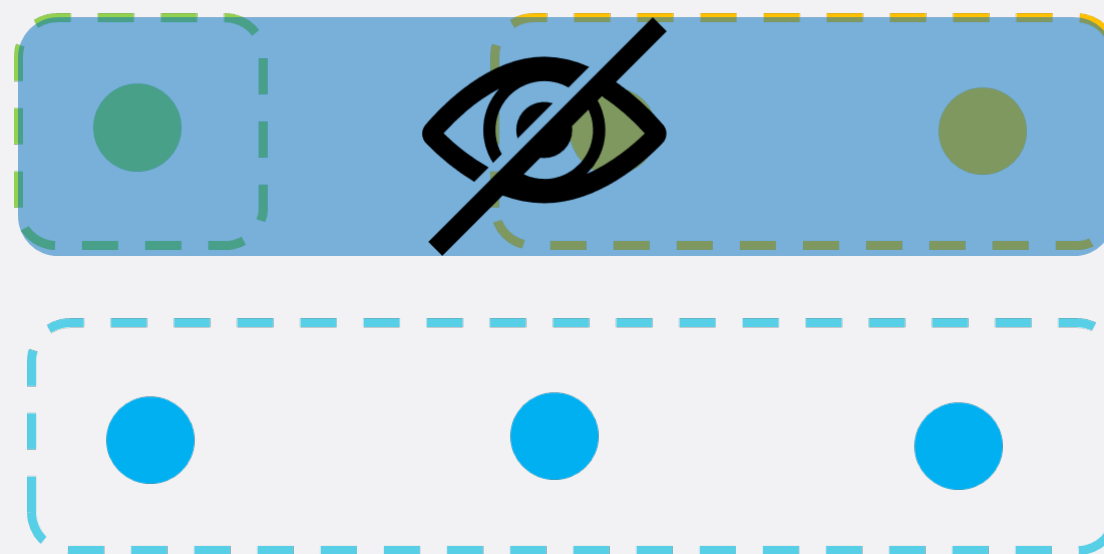


# 存在挑战

解空间巨大  $2^n$



多故障掩盖



Sequential

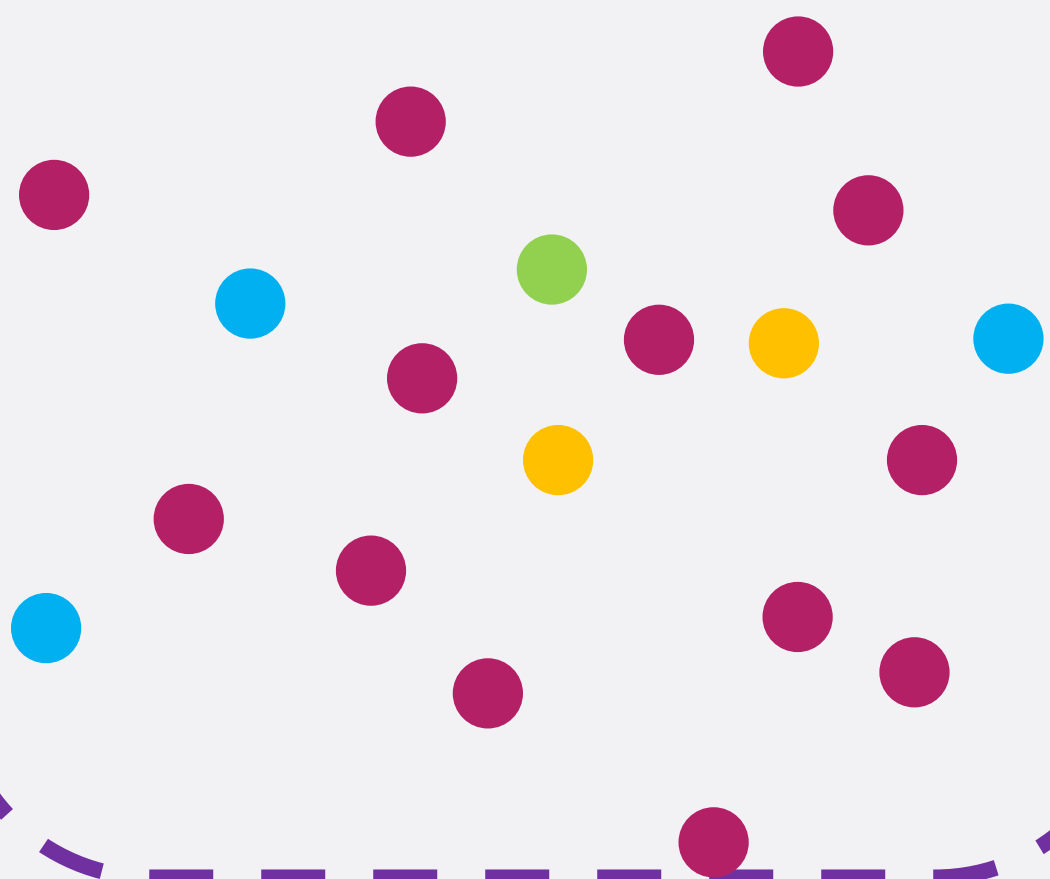
Detect



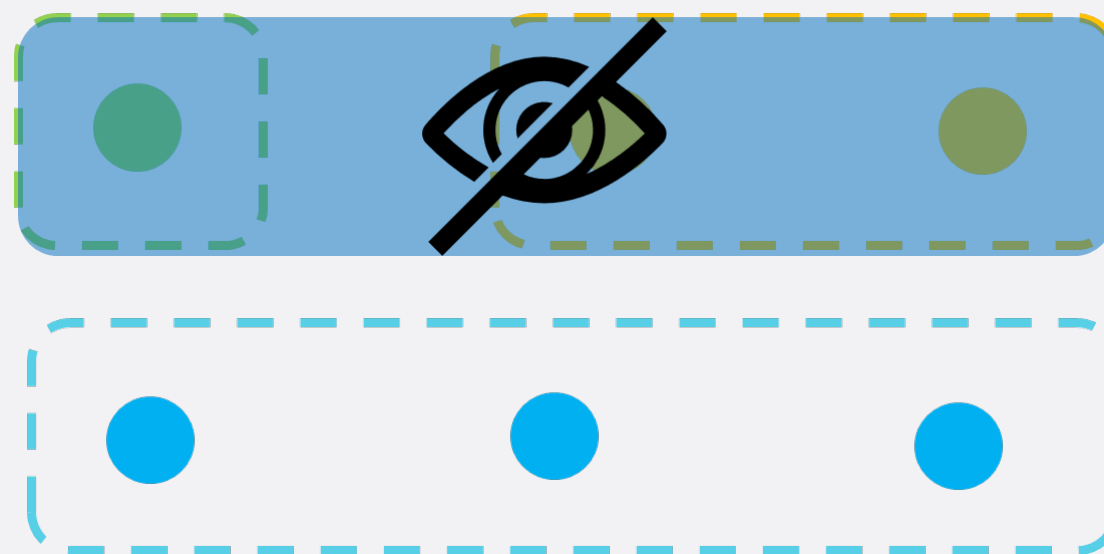
Locate

# 存在挑战

解空间巨大  $2^n$



多故障掩盖

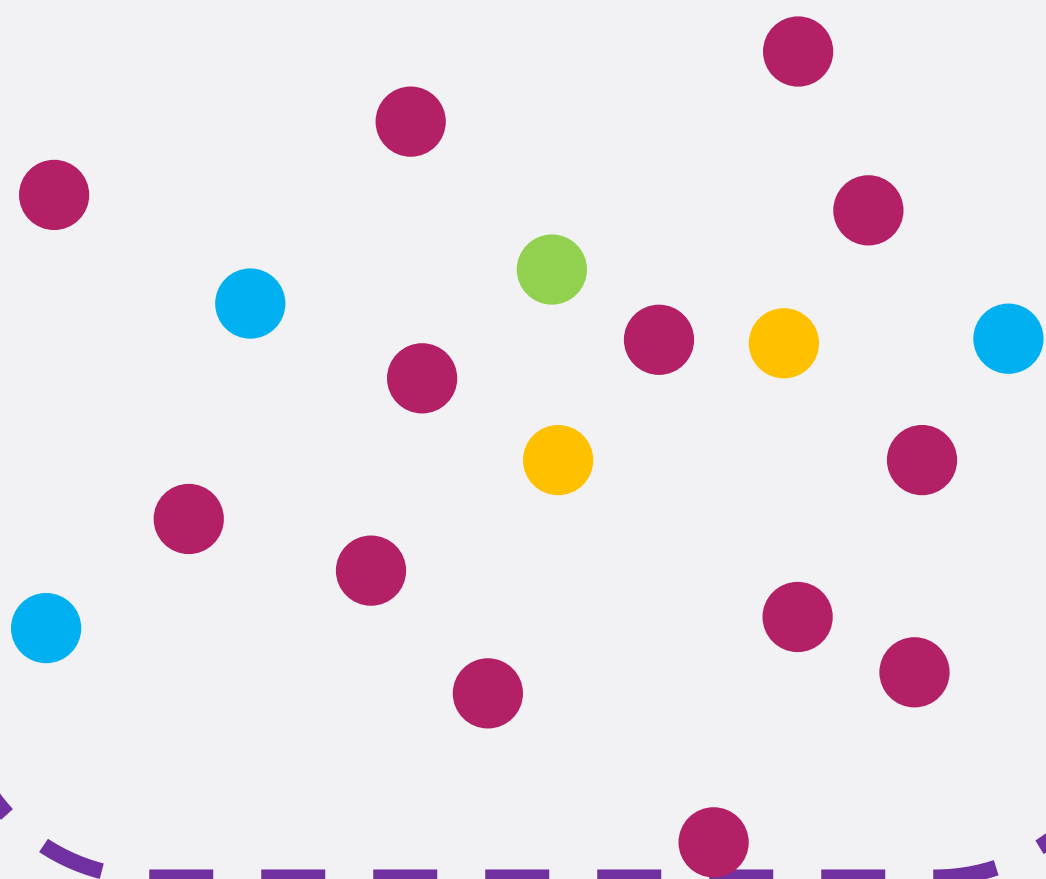


Sequential

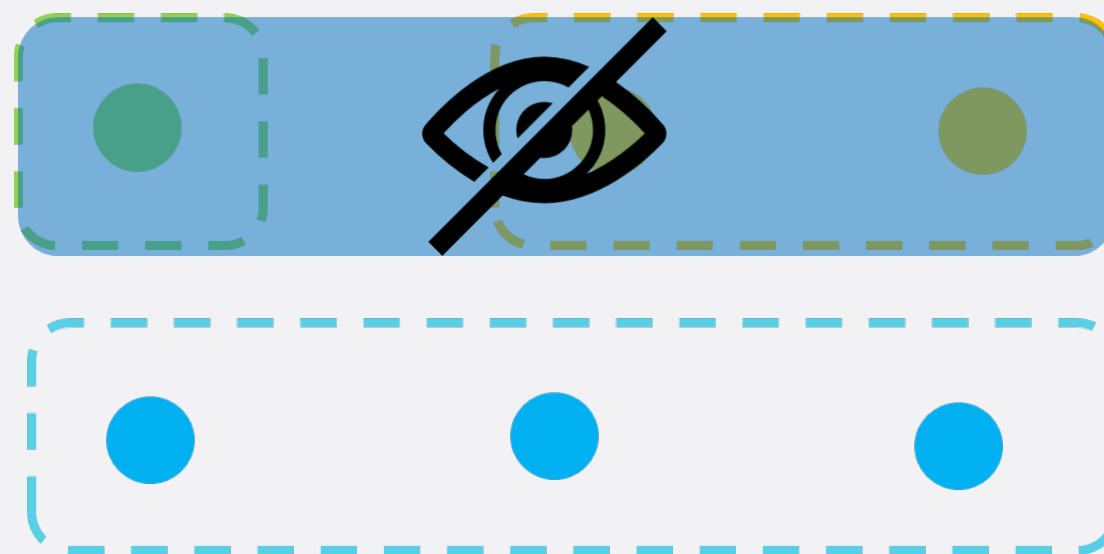


# 存在挑战

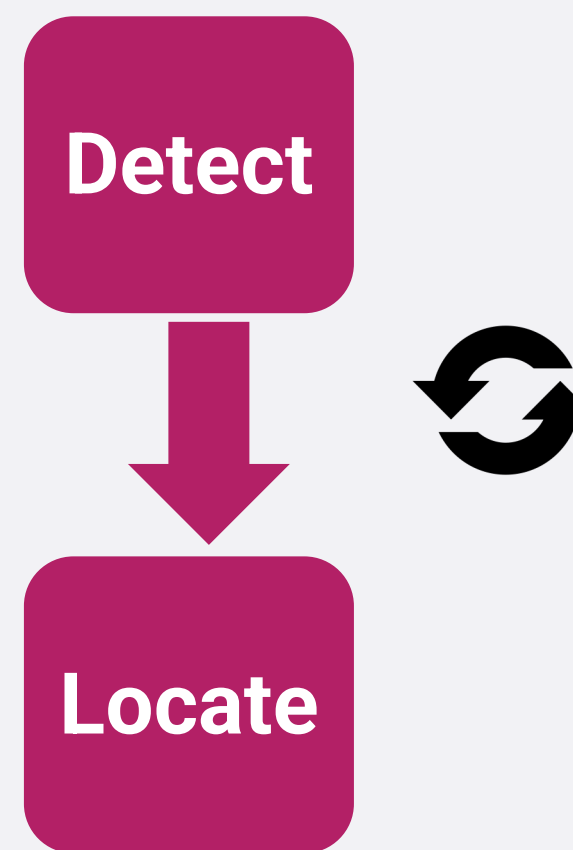
解空间巨大  $2^n$



多故障掩盖



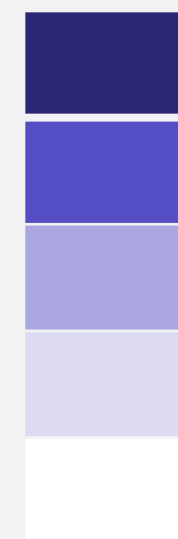
传统框架的低效  
Sequential



# 研究现状

	解空间		故障性质		框架	
	维度	个数	掩盖	确定	测试	引入
CTA			深		深	
Zhang			深	深	深	
LDA	中		深	深	深	
ELA_safe	浅		深	深	深	深
ELA_no_safe			深	深	深	
SOFOT		深	深	深	深	
OFOT		深	深	深	深	深
IterAIFL			深	深	深	深
FIC		深	深	深	深	深
FIC_BS		深	深	深	深	深
NOVLP		浅	深	深	深	深
RI		深	深	深	浅	深
Spectrum based	浅		深	中	浅	
Martinez_safe	深		深	深	深	深
Martinez_no_safe	深		深	深	深	

严重程度依颜色深度下降

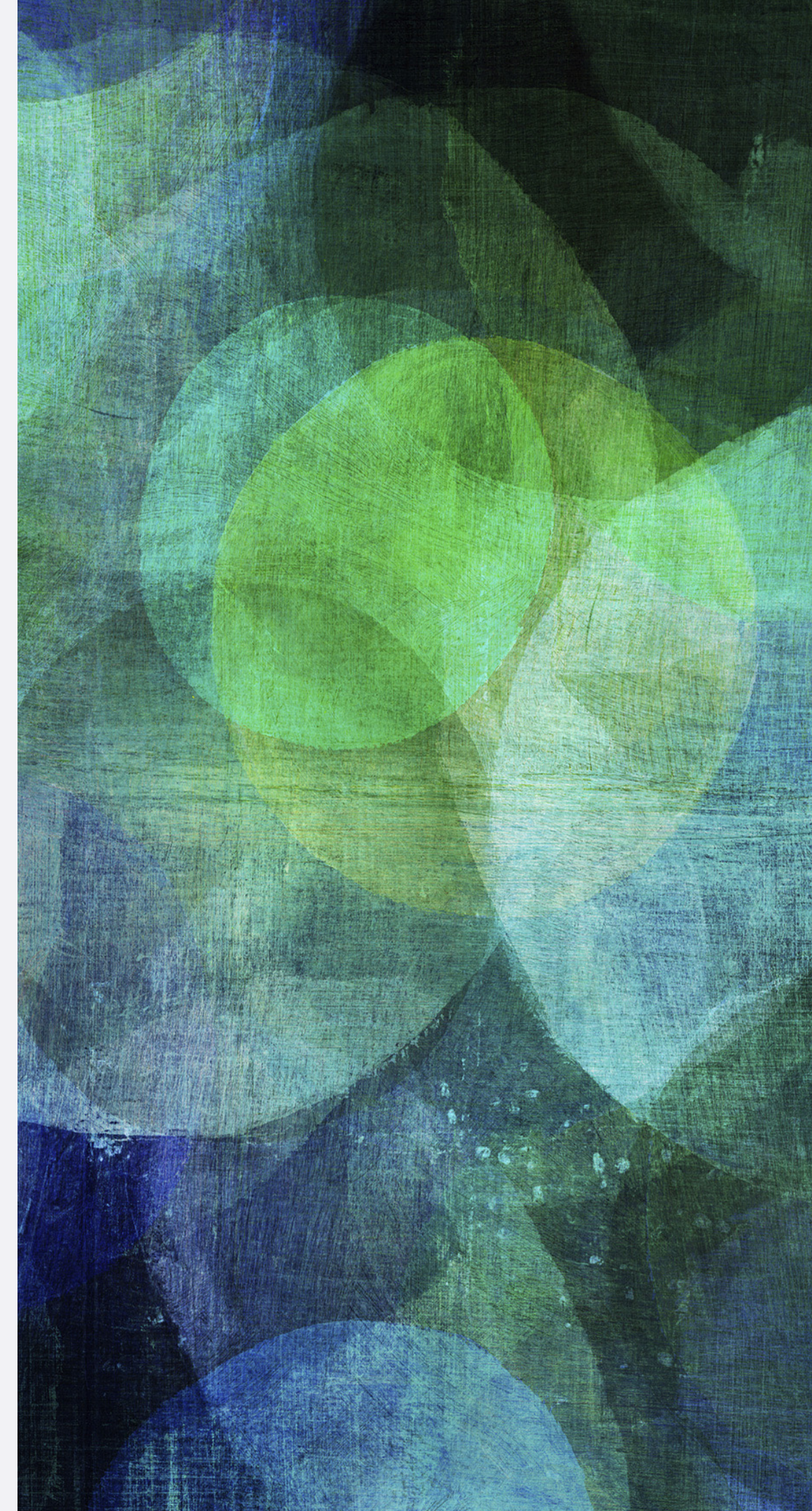


# 研究成果

# 基于关系树的最小故障模式定位

**钮鑫涛**, 聂长海, Alvin T. S. Chan, 组合测试故障定位的关系树模型. *计算机学报*, 37(12):2505-2518, 2014.

**Xintao Niu**, Changhai Nie, Yu Lei, Alvin T. S. Chan, Identifying Failure-Inducing Combinations Using Tuple Relationship. *ICST Workshops 2013*: 271-280





# 动机

含有 $n$ 个参数值的测试用例一共有  $2^n$  个模式。

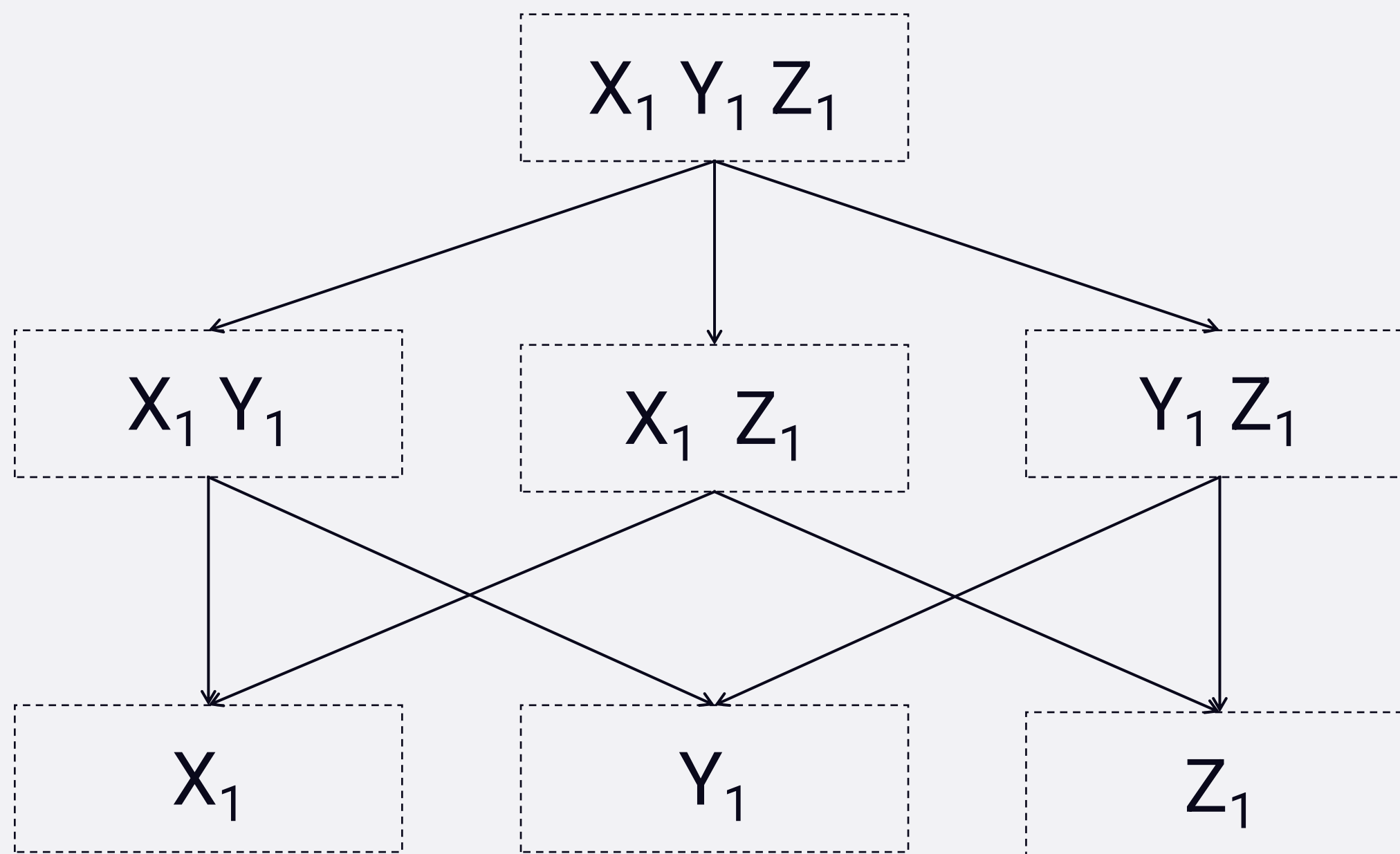
已有工作	通过方法	解空间约减
[Colbourn 08] [Martínez 08, 09]	假设已知最小故障模式个数和维度	$\binom{n}{t}$
[Nie 11][Zhang 11]	假设只有单个故障	$n$
[Zhang 11][Li 12]	假设多个不重叠故障	$d*n$
[Ghandehari 49]	假设已知最小故障模式维度	$\binom{n}{t}$
[Zhang 12]	通过约束求解器 (依赖求解器效率)	-
[Yilmaz 04]	通过分类树 (不准确, 依赖测试集)	-
[Martínez 08, 09]	假设最小故障模式维度2	$\binom{n}{2}$



# 模式关系树

将模式按照包含关系关联起来的树状数据结构

例子: 假设 $X_1$   $Y_1$   $Z_1$ 构成的测试用例触发故障



# 模式关系树的优势

$X_1 Y_1 Z_1$

$X_1$

$Y_1$

$Z_1$

$X_1 Y_1$

$X_1 Z_1$

$Y_1 Z_1$

$X_1 Y_1 Z_1$



# 模式关系树的优势

不忽略 $2^n$ 任何一个模式

$X_1 Y_1 Z_1$

$X_1$

$Y_1$

$Z_1$

$X_1 Y_1$

$X_1 Z_1$

$Y_1 Z_1$

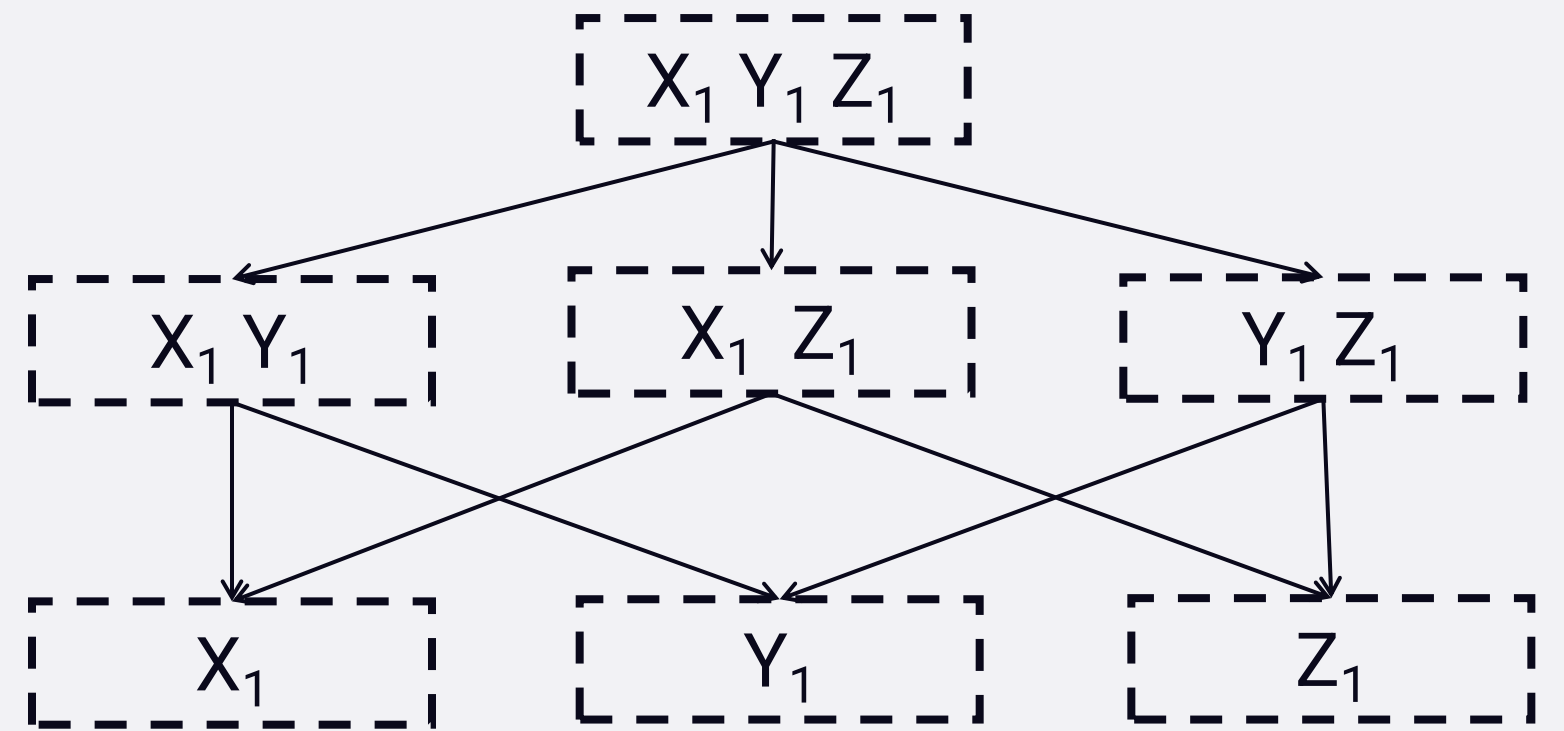
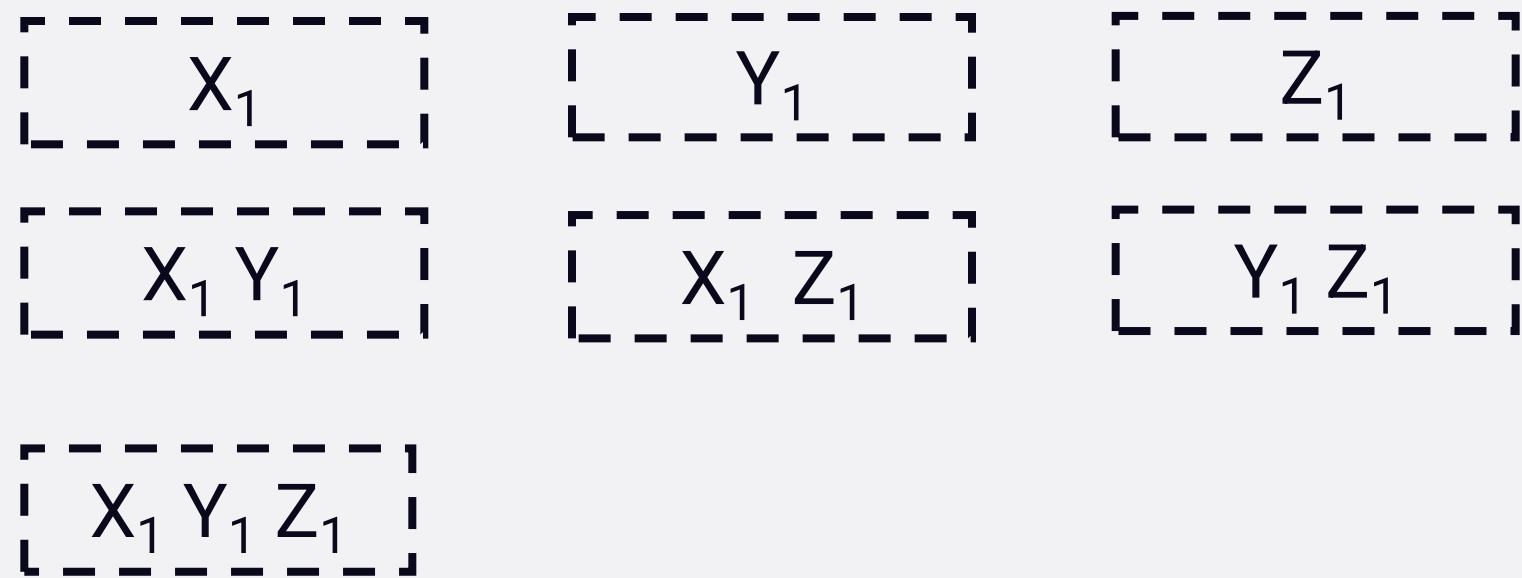
$X_1 Y_1 Z_1$



# 模式关系树的优势

不忽略 $2^n$ 任何一个模式

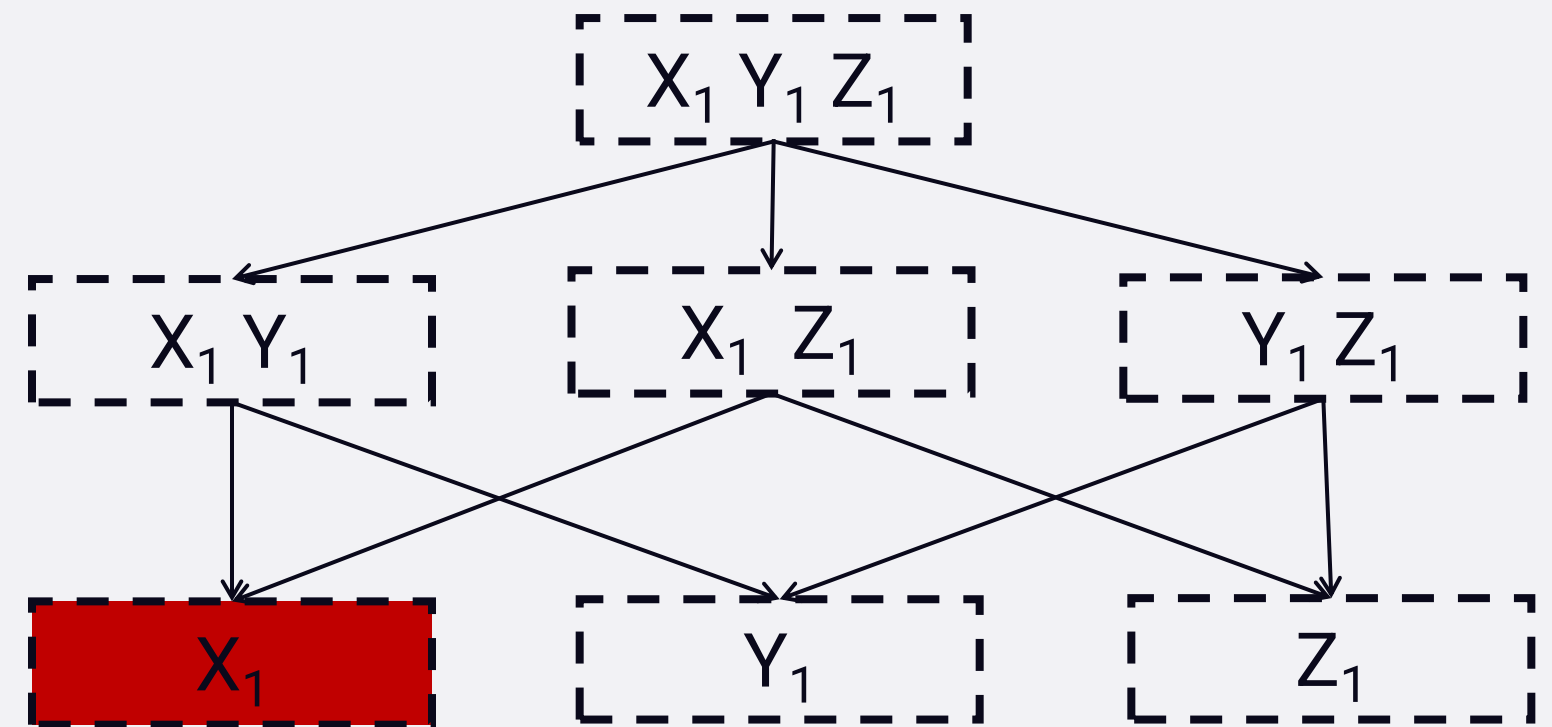
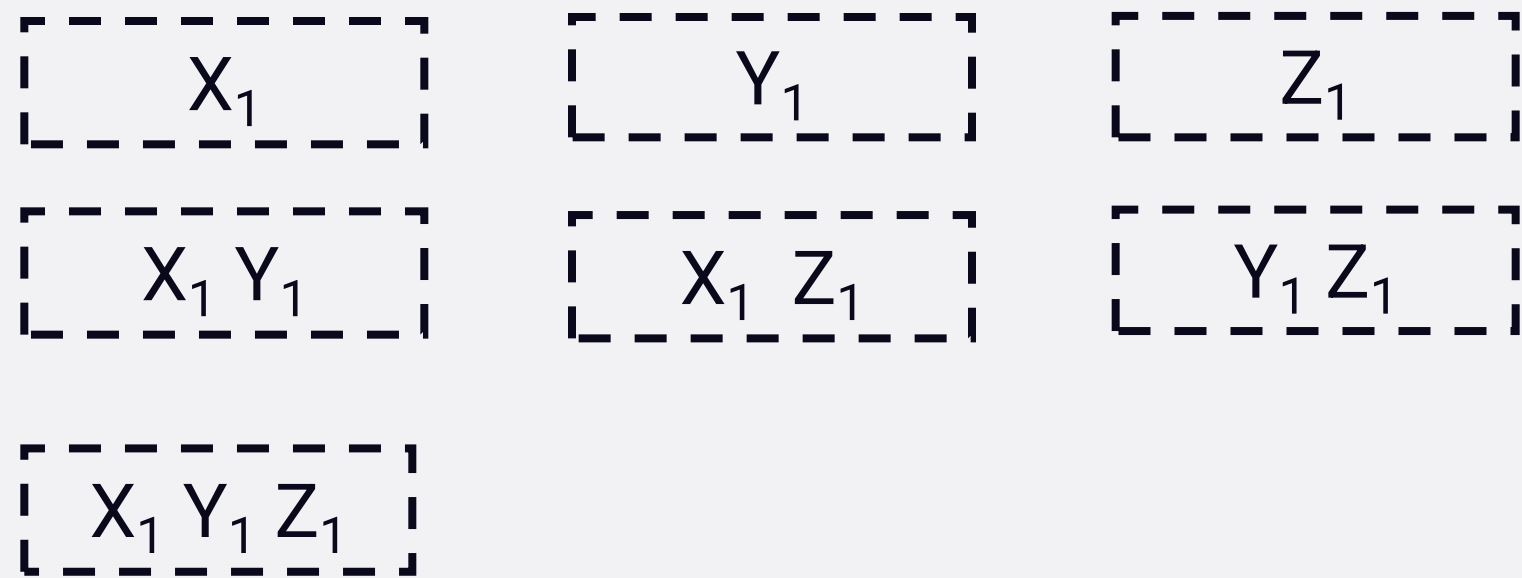
$X_1 Y_1 Z_1$



# 模式关系树的优势

不忽略 $2^n$  任何一个模式

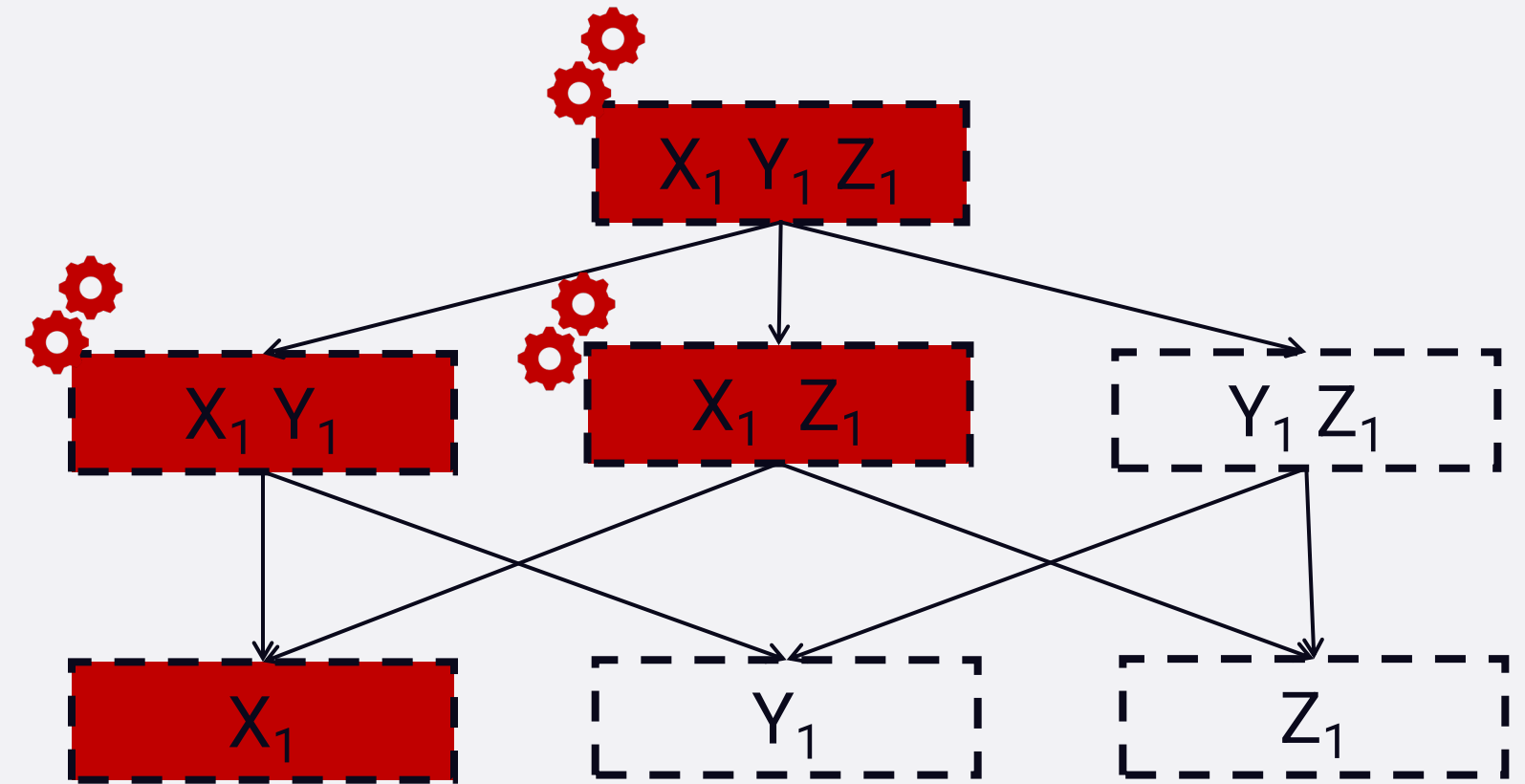
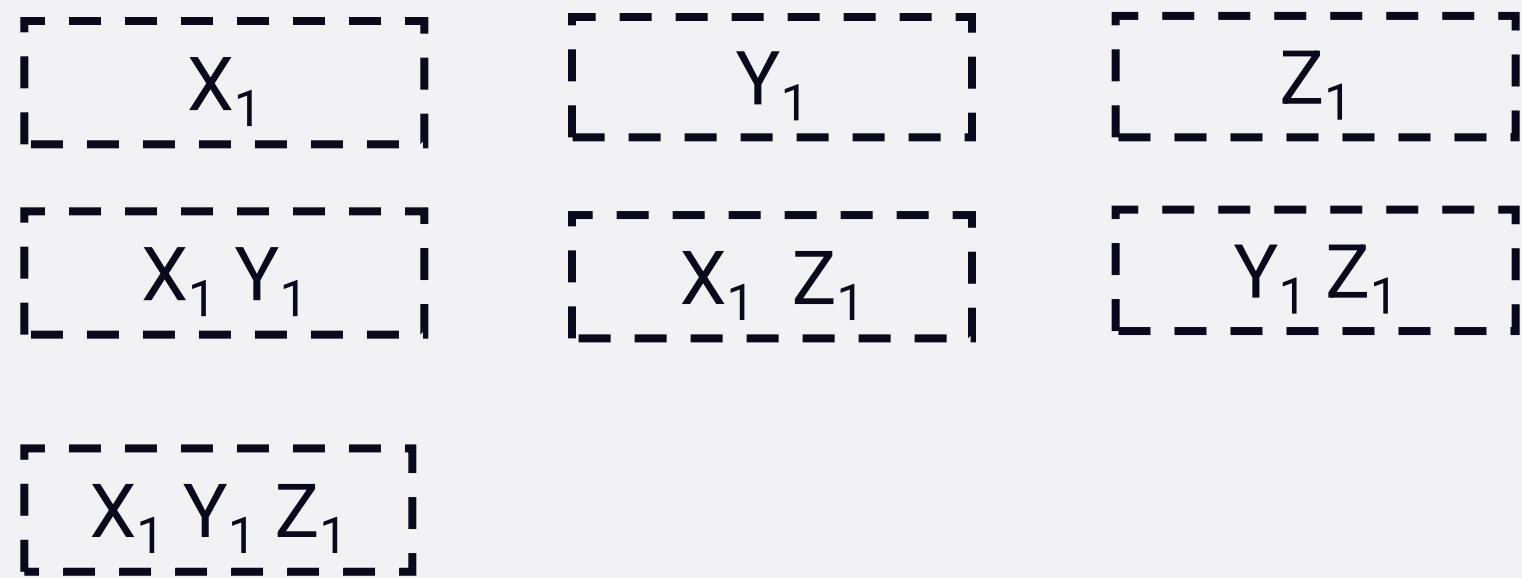
$X_1 Y_1 Z_1$



# 模式关系树的优势

不忽略 $2^n$ 任何一个模式

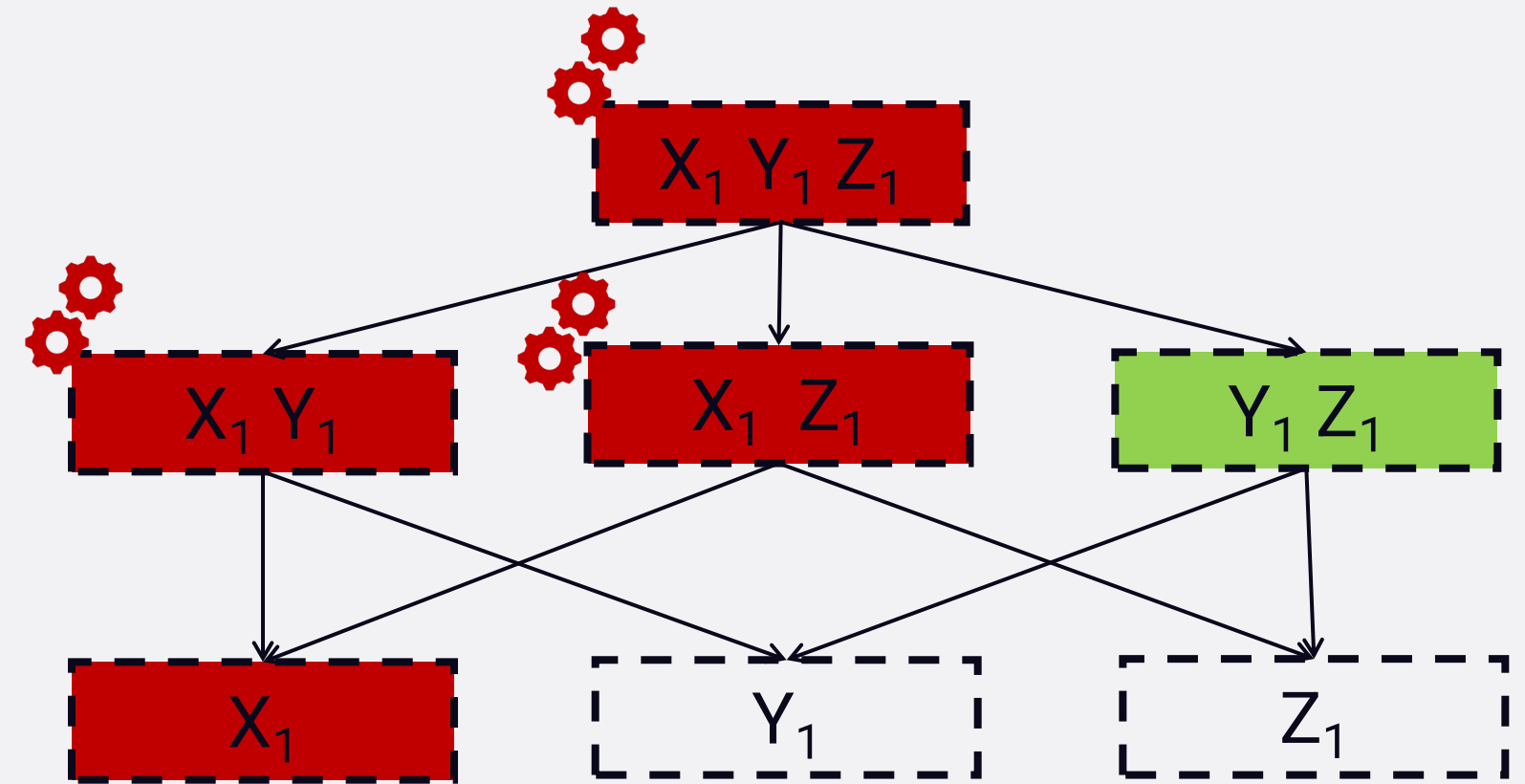
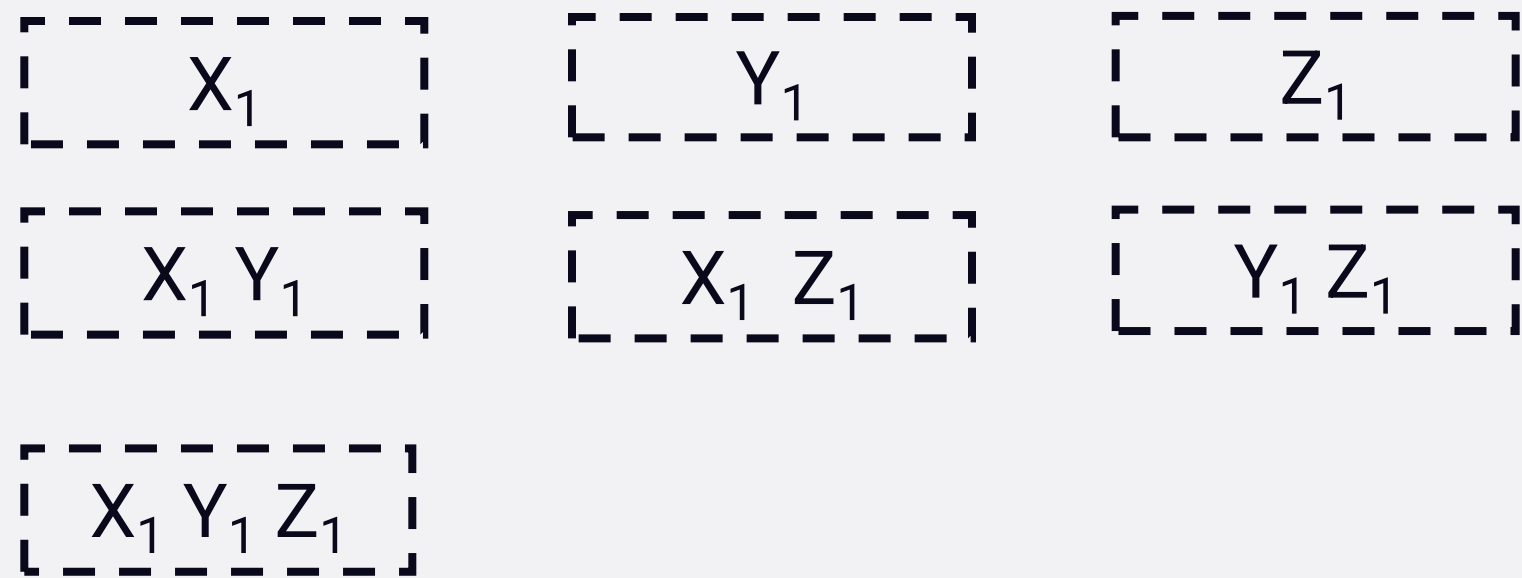
$X_1 Y_1 Z_1$



# 模式关系树的优势

不忽略 $2^n$  任何一个模式

$X_1 Y_1 Z_1$

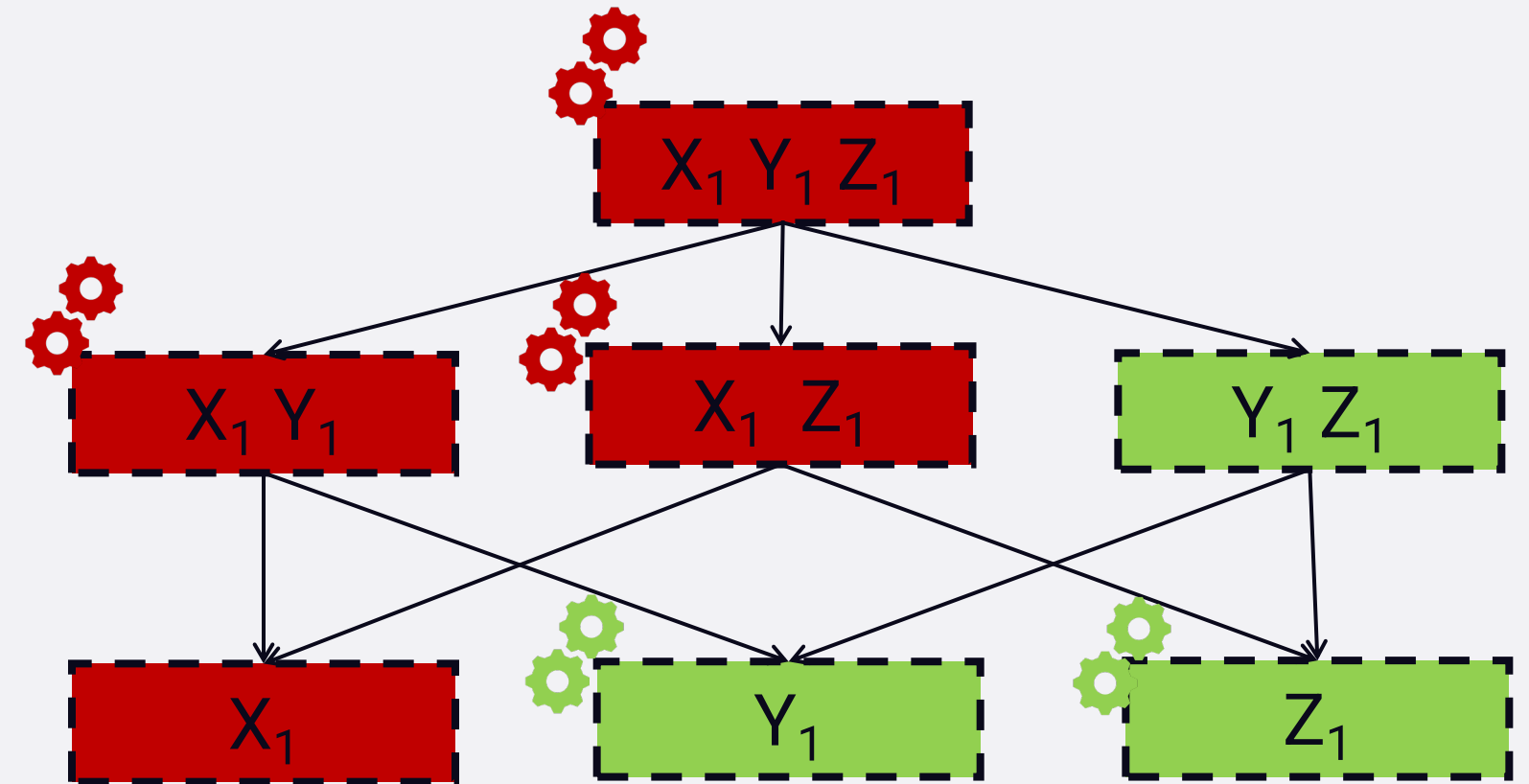
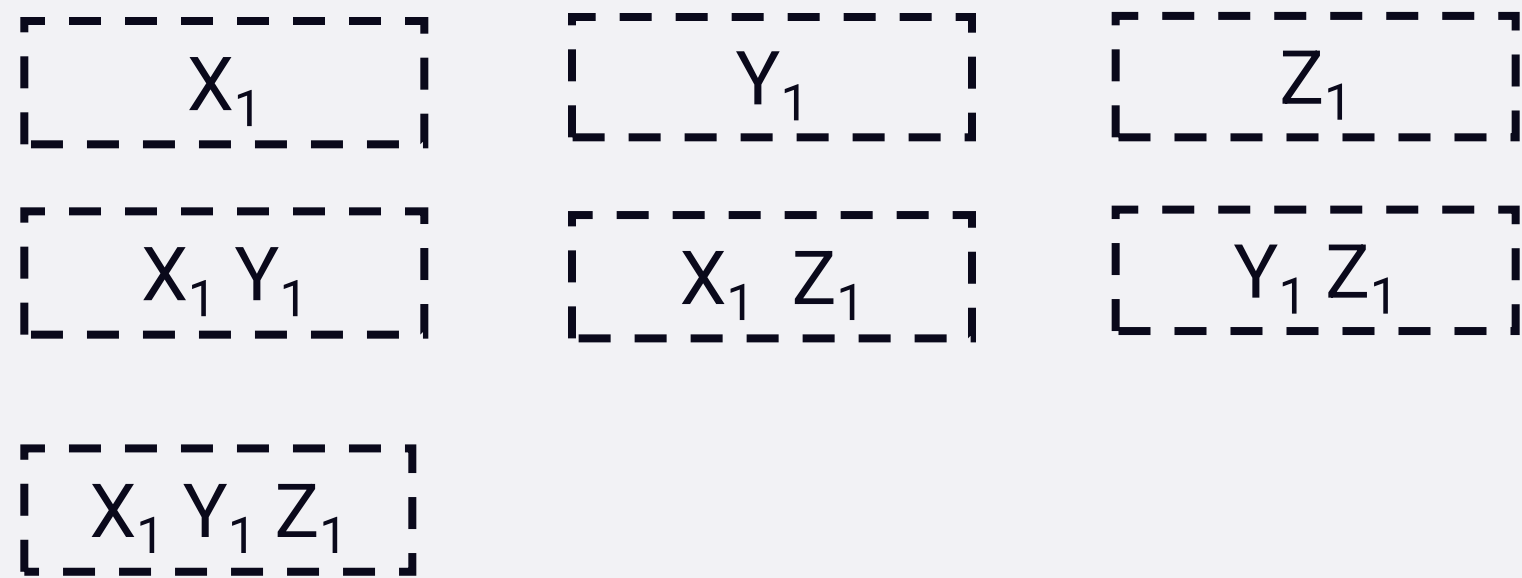




# 模式关系树的优势

不忽略 $2^n$ 任何一个模式

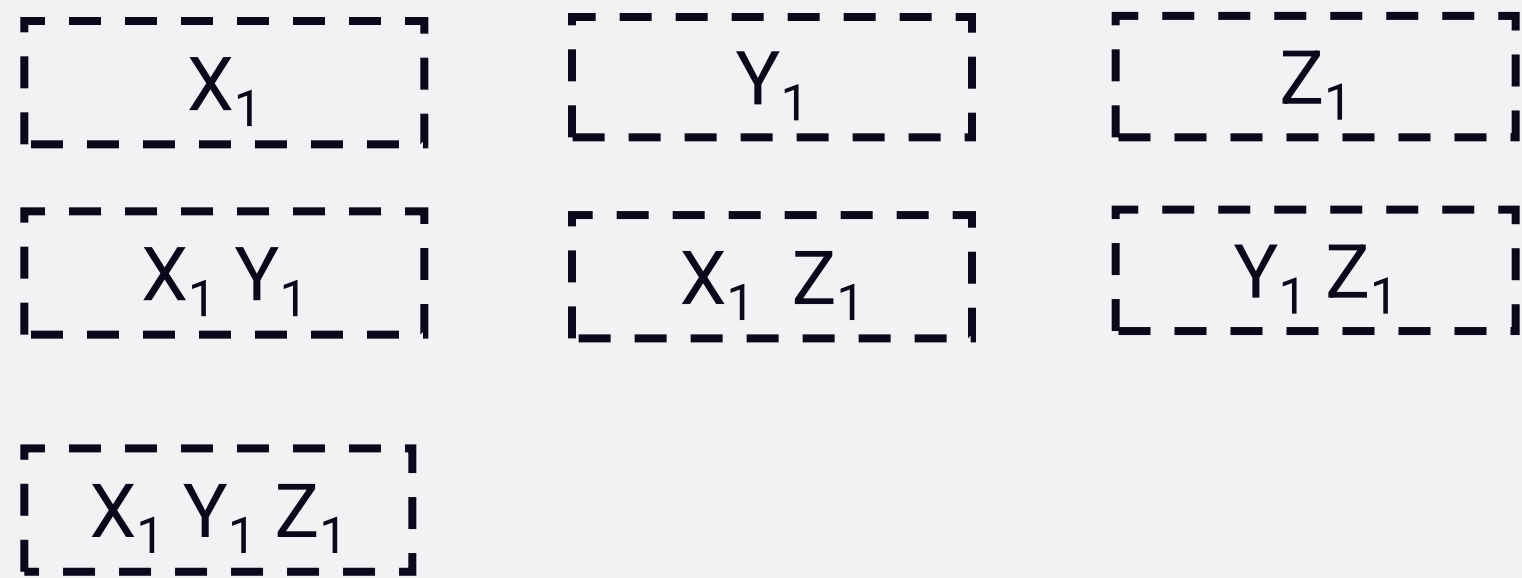
$X_1 Y_1 Z_1$



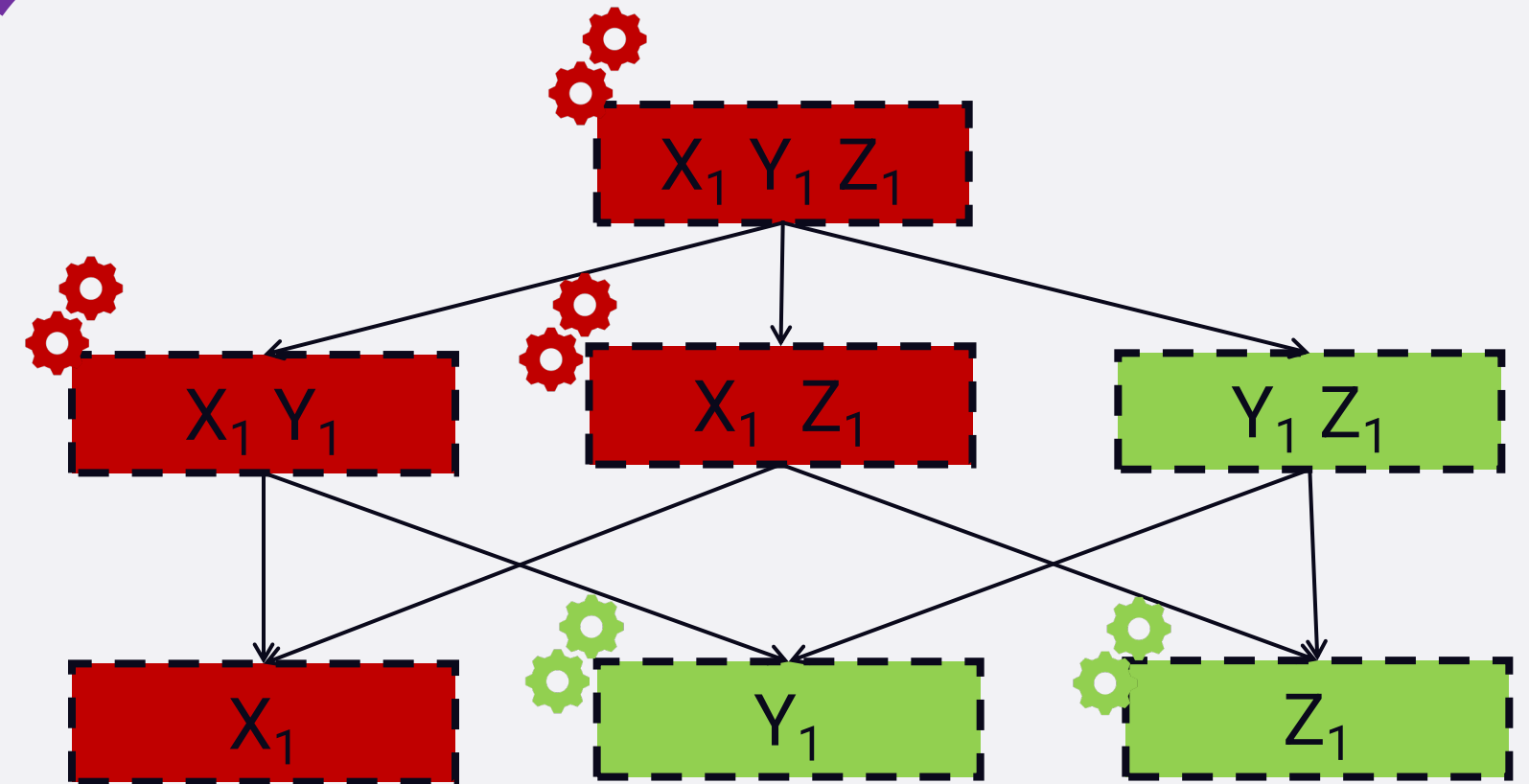
# 模式关系树的优势

不忽略 $2^n$  任何一个模式

$X_1 Y_1 Z_1$



减少实际需要检验的模式个数



具体减少多少要看如何挑选模式  
(顺序)来检验

# 实验评估

TABLE VI. COMPARING RESULT OF SINGLE FAULTY TUPLE

SUT	t	Average extra test configurations			Average covered tuples			Efficiency		
		PATH	PATH_NA	FIC_BS	PATH	PATH_NA	FIC_BS	PATH	PATH_NA	FIC_BS
1	2	10.8	23.8	9.0	255	255	163.3	23.7	10.7	18.2
	3	12.4	25.7	12.0	255	255	140.4	20.5	9.9	11.5
	4	12.7	23.9	14.9	255	255	147.5	20.1	10.7	9.7
2	2	11.3	24.7	9.6	511	511	318.8	45.1	20.7	33.2
	3	13.7	27.9	12.9	511	511	263.8	37.4	18.3	20.0
	4	14.9	28.8	16.1	511	511	269.8	34.3	17.7	16.4
3	2	11.8	26.2	9.9	1023	1023	624.6	86.7	39.0	63.1
	3	14.2	29.4	13.3	1023	1023	500.0	72.0	34.8	36.8
	4	16.0	31.4	16.7	1023	1023	498.4	64.0	32.5	29.0
4	2	13.2	30.1	10.3	2047	2047	1227.6	155.3	68.0	120.1
	3	15.2	32.3	13.8	2047	2047	954.5	134.2	63.4	68.0
	4	17.0	34.1	17.3	2047	2047	928.8	120.2	60.0	52.4
5	2	13.7	31.3	10.4	4095	4095	2419.2	299.6	130.7	232.4
	3	16.7	36.2	14.1	4095	4095	1832.6	246.7	113.0	127.9
	4	18.2	37.3	17.7	4095	4095	1743.6	225.0	109.7	96.0

TABLE VII. COMPARING RESULT OF TWO NON OVERLAPPING FAULTY TUPLES

SUT	t	Average extra test configurations			Average covered tuples			Efficiency		
		PATH	PATH_NA	FIC_BS	PATH	PATH_NA	FIC_BS	PATH	PATH_NA	FIC_BS
1	2	25.0	55.7	17.2	255	255	158.0	10.2	4.6	9.2
	3	34.0	66.8	21.7	255	255	119.5	7.5	3.8	5.5
	4	41.0	72.5	24.2	255	255	111.6	6.2	3.5	4.6
2	2	27.0	58.9	17.9	511	511	307.2	18.9	8.7	17.1
	3	39.6	78.4	23.3	511	511	221.7	12.9	6.5	9.4
	4	48.0	85.3	27.6	511	511	198.0	10.6	6.0	7.1
3	2	28.6	63.5	18.6	1023	1023	600.7	35.8	16.1	32.3
	3	42.6	85.0	24.4	1023	1023	416.4	24.0	12.0	17.0
	4	54.6	99.7	29.4	1023	1023	356.7	18.7	10.3	12.0
4	2	31.0	69.7	19.9	2047	2047	1179.5	65.9	29.4	59.6
	3	46.1	93.7	25.5	2047	2047	790.2	44.4	21.8	30.9
	4	59.1	110.2	31.0	2047	2047	651.4	34.6	18.6	20.8
5	2	34.0	77.7	20.3	4095	4095	2323.8	120.4	52.7	114.8
	3	49.3	101.5	26.9	4095	4095	1512.2	83.1	40.4	56.0
	4	63.5	120.5	32.1	4095	4095	1203.3	64.4	34.0	37.1

TABLE VIII. COMPARING RESULT OF TWO OVERLAPPING FAULTY TUPLES

SUT	t	Average extra test configurations			Average covered tuples in TRT			Efficiency			failure-inducing tuples identified		
		PATH	PATH_NA	FIC_BS	PATH	PATH_NA	FIC_BS	PATH	PATH_NA	FIC_BS	PATH	PATH_NA	FIC_BS
1	2	17.4	41.7	8.5	255	255	149.3	14.6	6.1	17.6	2	2	1
	3	22.6	48.5	11.4	255	255	113.5	11.3	5.3	9.9	2	2	1
	4	24.5	48.9	14.2	255	255	114.0	10.4	5.2	7.9	2	2	1
2	2	18.3	43.1	9.0	511	511	293.1	27.9	11.8	32.8	2	2	1
	3	26.2	56.4	12.1	511	511	212.6	19.5	9.1	17.3	2	2	1
	4	29.7	58.8	15.3	511	511	203.2	17.2	8.7	13.0	2	2	1
3	2	19.1	45.4	9.4	1023	1023	577.4	53.5	22.5	62.0	2	2	1
	3	28.0	60.4	12.6	1023	1023	402.6	36.5	16.9	31.7	2	2	1
	4	34.7	70.1	15.8	1023	1023	367.6	29.5	14.6	22.8	2	2	1
4	2	20.7	49.9	9.8	2047	2047	1140.6	98.9	41.0	118.0	2	2	1
	3	30.4	66.5	13.1	2047	2047	769.1	67.3	30.8	58.2	2	2	1
	4	38.6	78.6	16.5	2047	2047	673.9	53.1	26.1	40.1	2	2	1
5	2	22.4	54.5	10.0	4095	4095	2257.7	183.1	75.1	227.8	2	2	1
	3	32.8	72.6	13.4	4095	4095	1480.0	124.7	56.4	109.4	2	2	1
	4	42.1	86.7	16.9	4095	4095	1249.1	97.2	47.2	72.3	2	2	1

TABLE IX. RESULT OF IDENTIFYING NEW IMPORTED FAILURE-INDUCING TUPLE

SUT	t	correctly Identified tuples			new imported faulty tuples Identified			incorrectly Identified tuples		
		PATH	PATH_NA	FIC_BS	PATH	PATH_NA	FIC_BS	PATH	PATH_NA	FIC_BS
1	2	601	784	364	0	241	0	364	0	420
	3	1173	1568	728	0	618	0	478	0	840
	4	1736	1960	1260	0	297	0	245	0	700
2	2	1170	1296	1080	0	189	0	190	0	216
	3	2720	3024	2520	0	478	0	362	0	504
	4	3190	4536	1890	0	2396	0	1607	0	2646
3	2	1777	2025	1575	0	410	0	495	0	450
	3	4182	5400	2880	0	2053	0	1625	0	2520
	4	7569	9450	6300	0	3350	0	2156	0	3150
4	2	2688	3025	2475	0	710	0	913	0	550
	3	6691	9075	4455	0	4194	0	4455	0	4620
	4	12363	18150	8910	0	11466	0	7635	0	9240
5	2	3579	4356	2970	0	1252	0	1255	0	1386
	3	10663	14520	8360	0	7119	0	5298	0	6160
	4	23943	32670	18810	0	22370	0	14225	0	13860



# 实验评估

TABLE 10. Comparison Results of the Number of Test Cases

Method	#	Method 1			Method 2			Method 3		
		Test Case	Time (s)	Cost	Test Case	Time (s)	Cost	Test Case	Time (s)	Cost
A	1	100	100	100	100	100	100	100	100	100
	2	100	100	100	100	100	100	100	100	100
	3	100	100	100	100	100	100	100	100	100
B	1	100	100	100	100	100	100	100	100	100
	2	100	100	100	100	100	100	100	100	100
	3	100	100	100	100	100	100	100	100	100
C	1	100	100	100	100	100	100	100	100	100
	2	100	100	100	100	100	100	100	100	100
	3	100	100	100	100	100	100	100	100	100
D	1	100	100	100	100	100	100	100	100	100
	2	100	100	100	100	100	100	100	100	100
	3	100	100	100	100	100	100	100	100	100

TABLE 11. Comparison Results of the Number of Test Cases

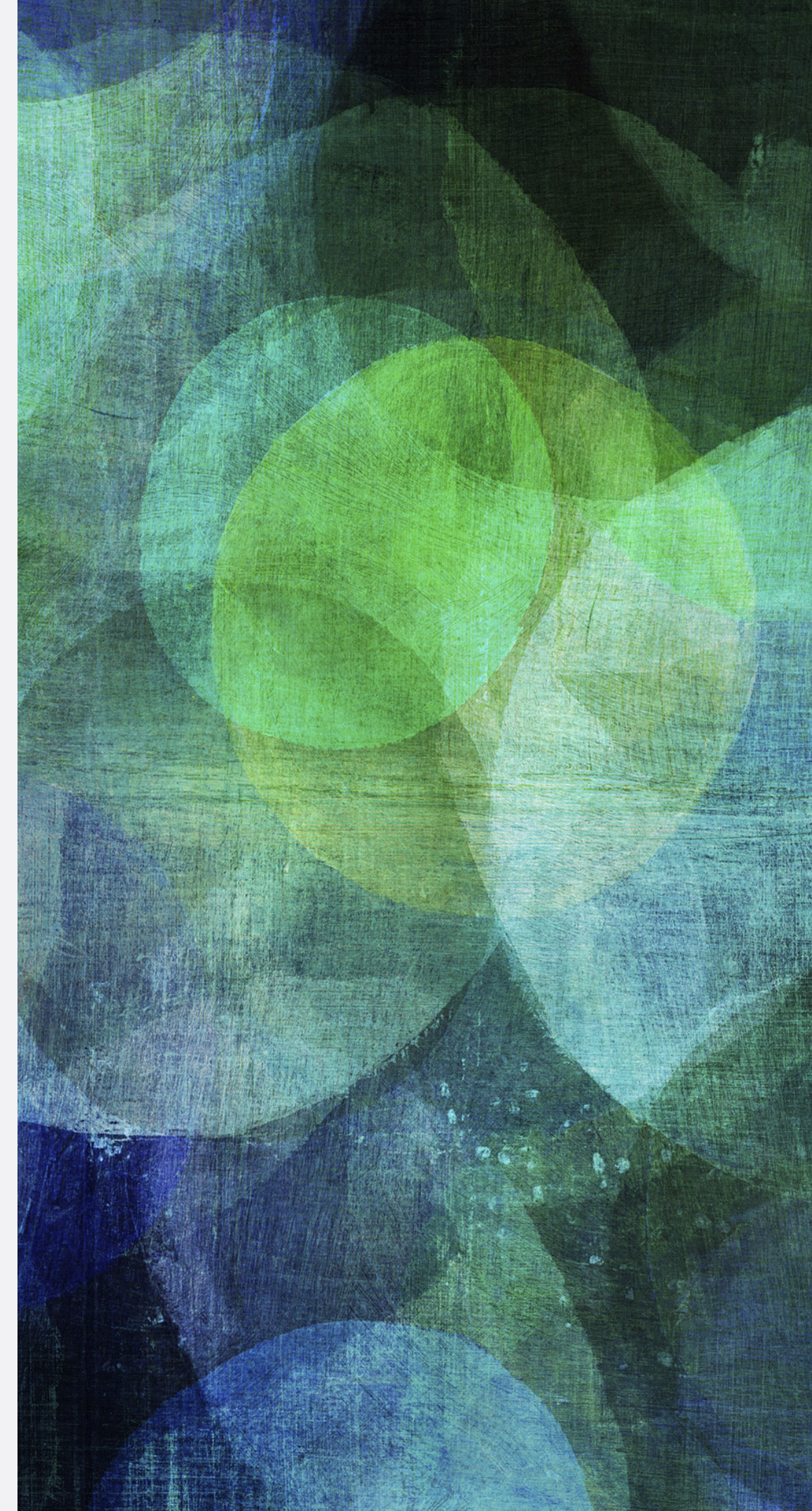
Method	#	Method 1			Method 2			Method 3		
		Test Case	Time (s)	Cost	Test Case	Time (s)	Cost	Test Case	Time (s)	Cost
A	1	100	100	100	100	100	100	100	100	100
	2	100	100	100	100	100	100	100	100	100
	3	100	100	100	100	100	100	100	100	100
B	1	100	100	100	100	100	100	100	100	100
	2	100	100	100	100	100	100	100	100	100
	3	100	100	100	100	100	100	100	100	100
C	1	100	100	100	100	100	100	100	100	100
	2	100	100	100	100	100	100	100	100	100
	3	100	100	100	100	100	100	100	100	100
D	1	100	100	100	100	100	100	100	100	100
	2	100	100	100	100	100	100	100	100	100
	3	100	100	100	100	100	100	100	100	100

和当前所需代价最少方法[Zhang 2011]相比，测试用例的增加有限。此外，得到更加准确和完整的故最小障模式定位结果。

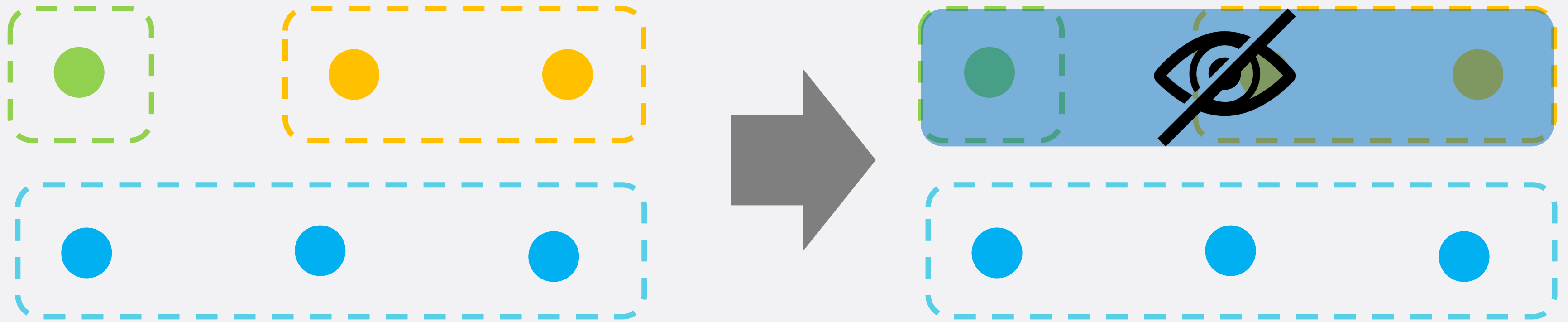


# 最小故障模式的掩盖问题

**Xintao Niu,** Changhai Nie, Yu Lei, Hareton Leung and Xiaoyin Wang,  
Identifying Failure-Causing Schemas in the Presence of Multiple Faults, *IEEE Transactions on Software Engineering (TSE)*, in press, 2018(CCF-A).



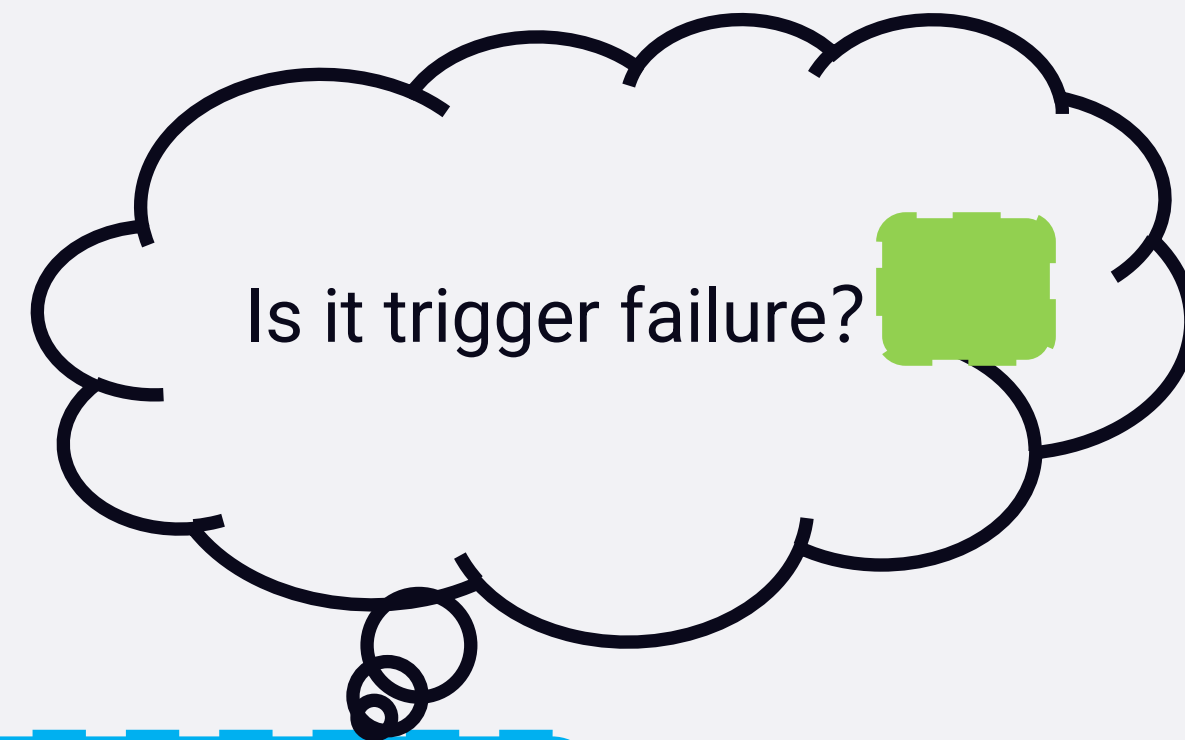
# 动机



掩盖现象一般多故障之间 [Dumlu 2011, Ylimaz 2014]。

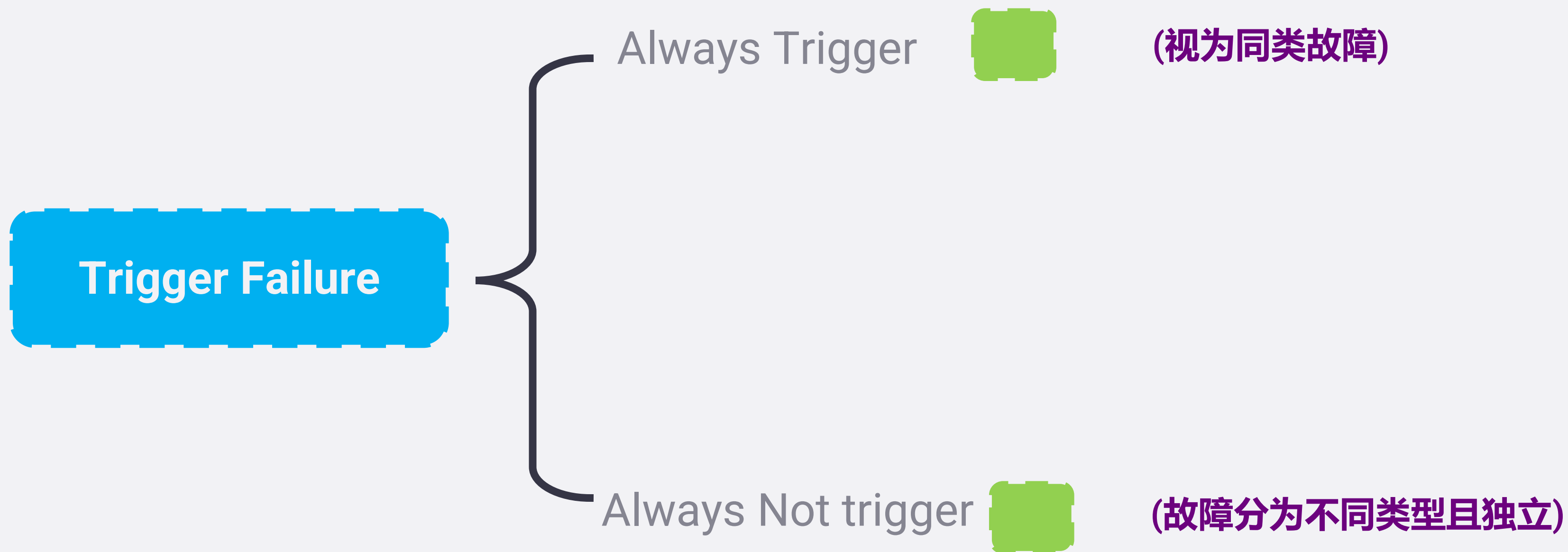
掩盖现象会对故障定位算法产生影响。

# 两种传统策略



**Trigger Failure**

# 两种传统策略

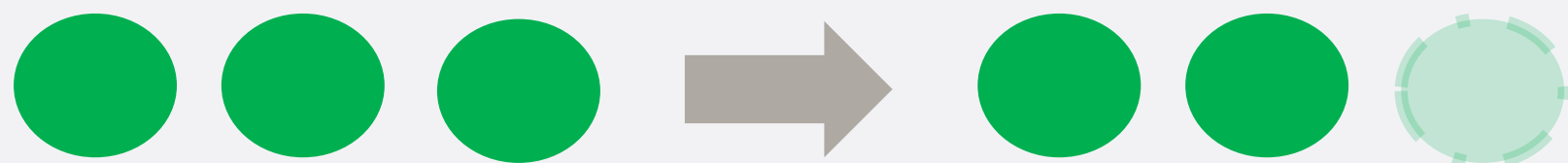




# 理论分析

## 视为同类故障

得到的最小故障模式偏向真实最小故障模式的子模式

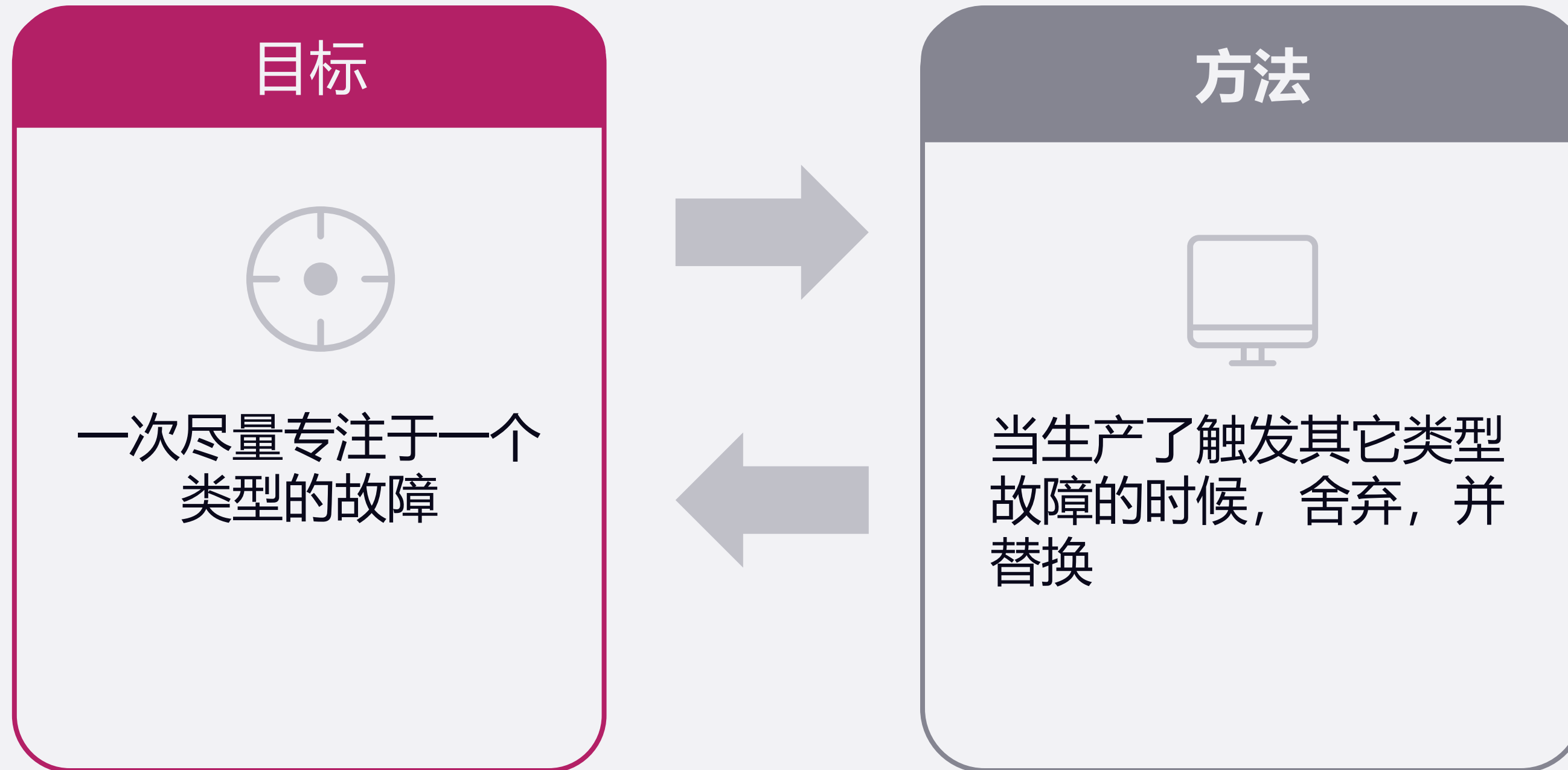


## 故障分为不同类型且独立








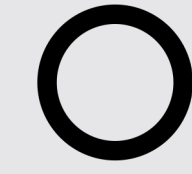


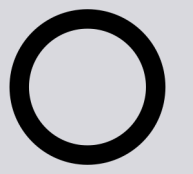

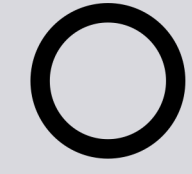







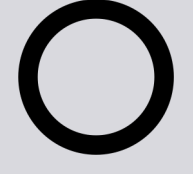

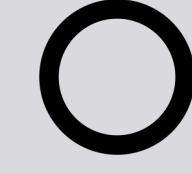


得到的最小故障模式偏向真实最小故障模式的父模式



# 方法



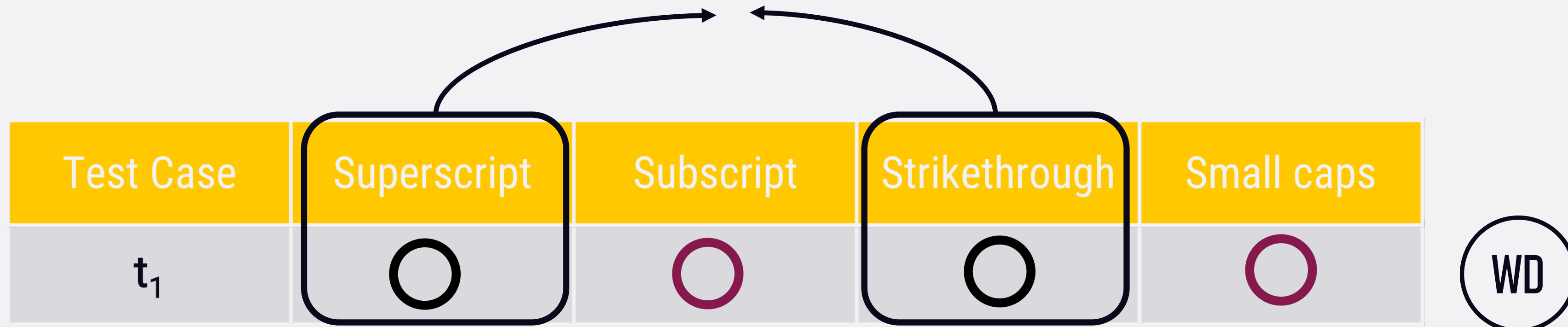
# 例子

Test Case	Superscript	Subscript	Strikethrough	Small caps	
t <sub>1</sub>					
t <sub>6</sub>					
t <sub>7</sub>					
t <sub>8</sub>					
t <sub>9</sub>					



# 例子

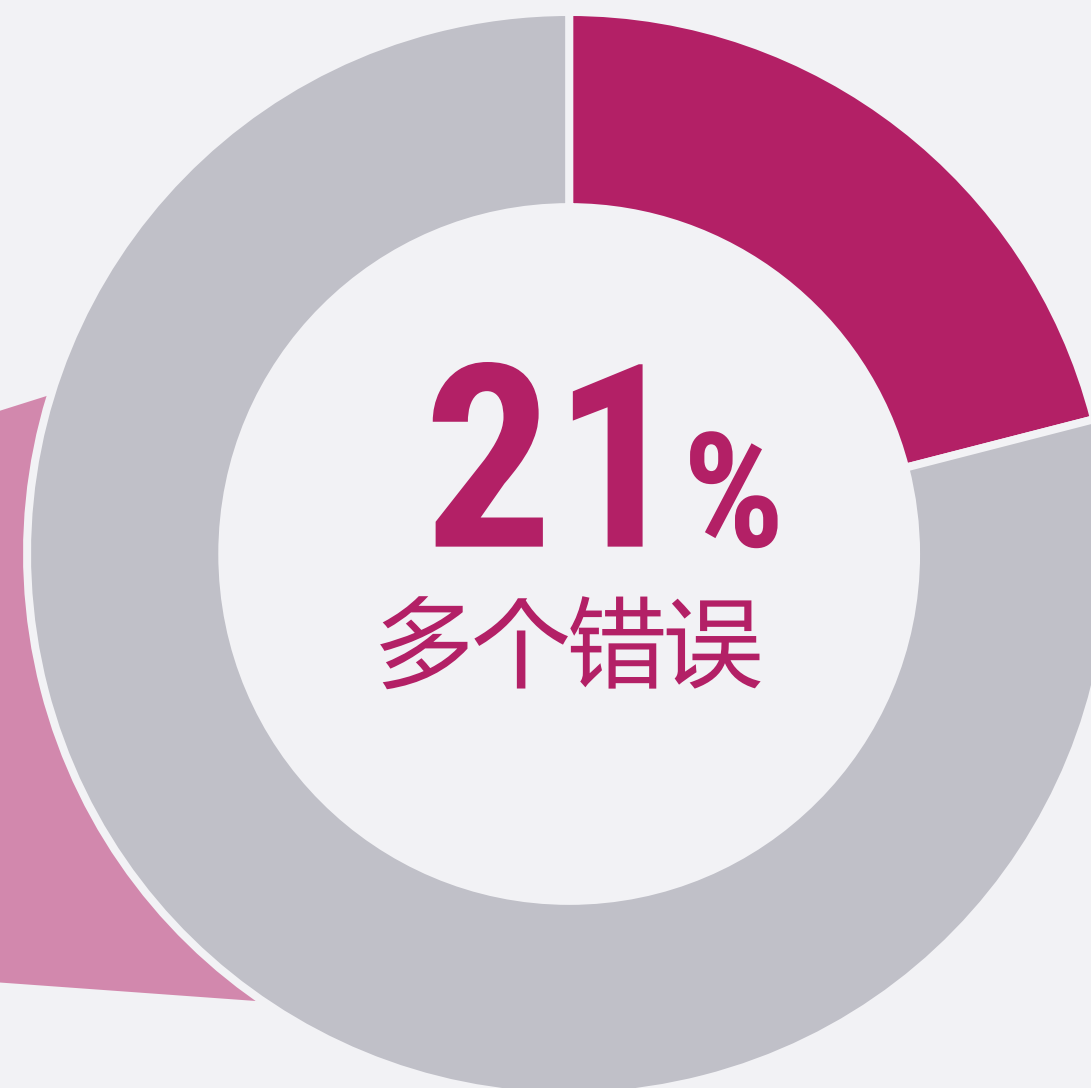
## 最小故障模式



# 实验评估

调研了**16**个不同的版本上的在**7**个开源软件。

使用CT建模，执行了**150,896**测试用例，有**76,132**失败的测试用例。



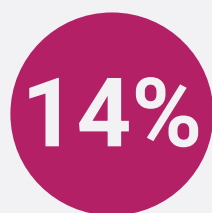
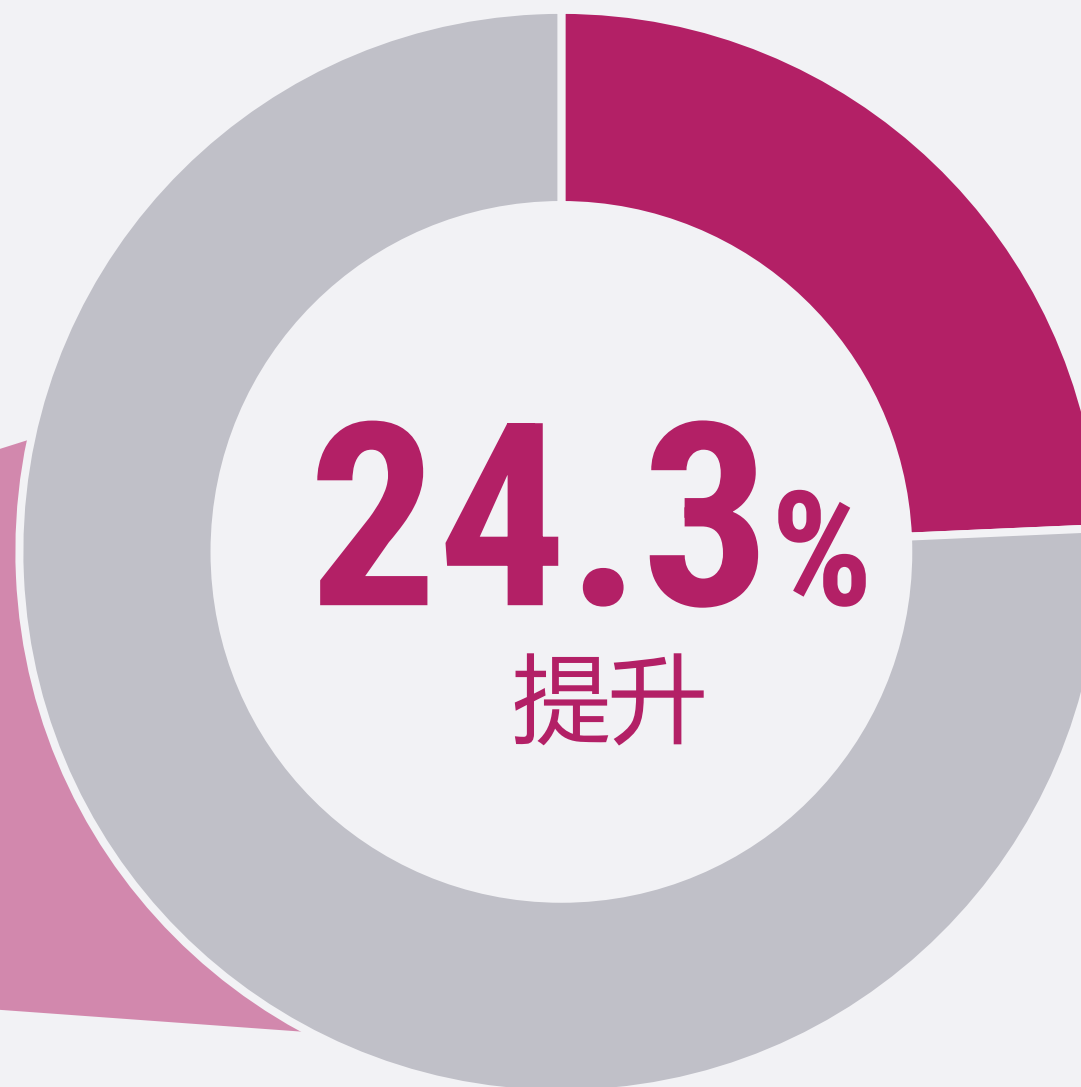
大约 **16056** 本该侦测多个错误，但是发生了掩盖作用导致没有发现，



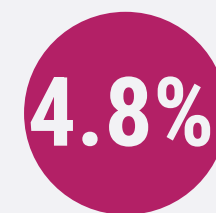
# 实验评估

在**93,308** 最小故障模式替换法发现了**65,836** 个。

这个数字要比其它方法都要多。



视为同类故障



故障分为不同类型且独立

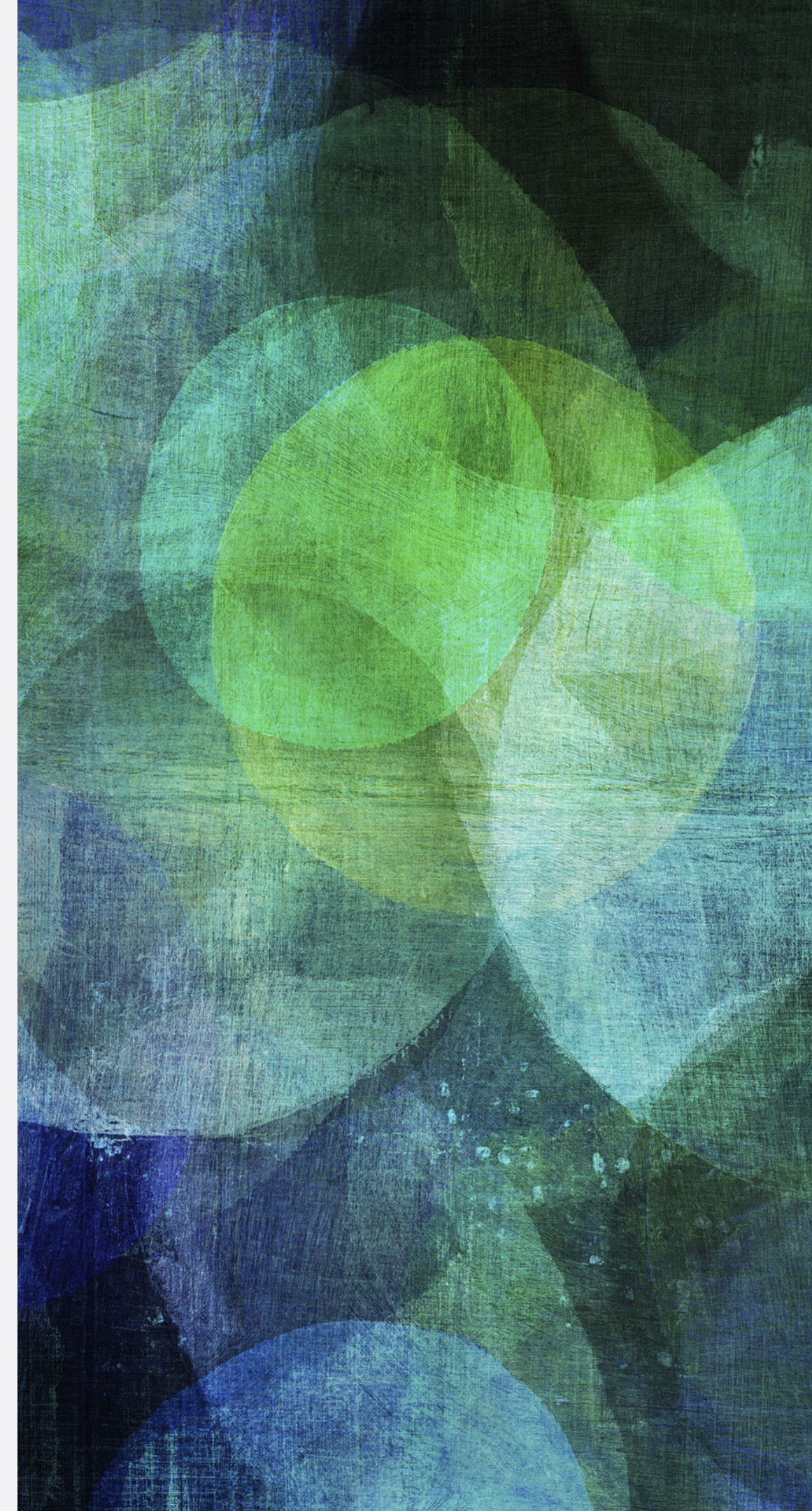


FDA-CIT



# 最小故障模式的定位和测试用例生成的交互式框架

**Xintao Niu**, Changhai ie, Lei Yu, Hareton Leung, Xiaoyin Wang, Jiaxi Xu, and Yan Wang, An Interleaving Approach to Combinatorial Testing and Failure-inducing Interaction Identification, *IEEE Transactions on Software Engineering (TSE)*, in press, 2018(CCF-A).



**Grep**      **-o**      **-E**      **-F**      **-c**

**Detect**

0	0	0	0	PASS
0	1	1	1	PASS
1	0	1	1	FAIL
1	1	0	1	FAIL
1	1	1	0	PASS

**Extract**

**locate**

**Locate**

1	0	1	1	FAIL
0	0	1	1	PASS
1	1	1	1	FAIL
1	0	0	1	FAIL
1	0	1	0	PASS

**-o**      **-c**

1	1	0	1	FAIL
0	1	0	1	PASS
1	0	0	1	FAIL
1	1	1	1	FAIL
1	1	0	0	PASS

**-o**      **-c**

```
Usage: grep [OPTION]... PATTERN [FILE]...
Search for PATTERN in each FILE or standard input.
PATTERN is, by default, a basic regular expression (BRE).
Example: grep -i 'hello world' menu.h main.c

Regex selection and interpretation:
-E, --extended-regexp  PATTERN is an extended regular expression (ERE)
-F, --fixed-strings    PATTERN is a set of newline-separated fixed strings
-G, --basic-regexp    PATTERN is a basic regular expression (BRE)
-P, --perl-regexp     PATTERN is a Perl regular expression
-e, --regexp=PATTERN  use PATTERN for matching
-f, --file=FILE       obtain PATTERN from FILE
-i, --ignore-case     ignore case distinctions
-w, --word-regexp     force PATTERN to match only whole words
-x, --line-regexp     force PATTERN to match only whole lines
-z, --null-data       a data line ends in 0 byte, not newline

Miscellaneous:
-s, --no-messages    suppress error messages
-v, --invert-match    select non-matching lines
-V, --version        print version information and exit
--help              display this help and exit
--mmap              deprecated no-op; evokes a warning

Output control:
-m, --max-count=NUM  stop after NUM matches
-b, --byte-offset    print the byte offset with output lines
-n, --line-number    print line number with output lines
--line-buffered     flush output on every line
-H, --with-filename  print the file name for each match
-h, --no-filename   suppress the file name prefix on output
--label=LABEL       use LABEL as the standard input file name prefix
-o, --only-matching  show only the part of a line matching PATTERN
-q, --quiet, --silent  suppress all normal output
--binary-files=TYPE assume that binary files are TYPE;
                    TYPE is 'binary', 'text', or 'without-match'
-a, --text          equivalent to --binary-files=text
-I, --ignore-binary equivalent to --binary-files=without-match
-d, --directories=ACTION how to handle directories;
                    ACTION is 'read', 'recurse', or 'skip'
-D, --devices=ACTION  how to handle devices, FIFOs and sockets;
                    ACTION is 'read' or 'skip'
-r, --recursive     like --directories=recurse
-R, --dereference-recursive likewise, but follow all symlinks
--include=FILE_PATTERN search only files that match FILE_PATTERN
--exclude=FILE_PATTERN skip files and directories matching FILE_PATTERN
--exclude-from=FILE   skip files matching any file pattern from FILE
--exclude-dir=PATTERN directories that match PATTERN will be skipped.
-L, --files-without-match print only names of FILES containing no match
-l, --files-with-matches print only names of FILES containing matches
-c, --count          print only a count of matching lines per FILE
-T, --initial-tab    make tabs line up (if needed)
-Z, --null          print 0 byte after FILE name

Context control:
-B, --before-context=NUM print NUM lines of leading context
-A, --after-context=NUM  print NUM lines of trailing context
-C, --context=NUM       print NUM lines of output context
-NUM                     same as --context=NUM
--color[=WHEN],
--colour[=WHEN]        use markers to highlight the matching strings;
                    WHEN is 'always', 'never', or 'auto'
-U, --binary          do not strip CR characters at EOL (MSDOS/Windows)
-u, --unix-byte-offsets report offsets as if CRs were not there
                    (MSDOS/Windows)
```



# Sequential Combinatorial Testing (SCT)

0	0	0	0	PASS
0	1	1	1	PASS
1	0	1	1	FAIL
1	1	0	1	FAIL
1	1	1	0	PASS

测试用例冗余 (定位阶段也存在coverage)

1	0	1	1	FAIL
0	0	1	1	PASS
1	1	1	1	FAIL
1	0	0	1	FAIL
1	0	1	0	PASS
-0			-c	

重复定位最小故障模式

1	1	0	1	FAIL
0	1	0	1	PASS
1	0	0	1	FAIL
1	1	1	1	FAIL
1	1	0	0	PASS
-0			-c	

# Interleaving Combinatorial Testing (ICT)

Detect

一次生成一条测试用例直到发现错误

0 0 0 0 ...	Pass
0 1 1 1 ...	Pass
1 0 1 1 ...	Fail
1 1 0 0 ...	Pass

相比SCT生成的13条，ICT只生产了8条

只定位了一次，但得到了相同的结果

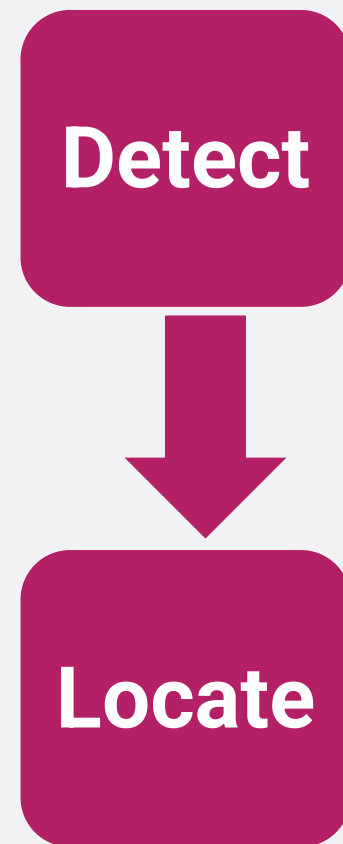
Locate

定位其中存在的最小故障模式

0 0 1 1 ...	Pass
1 1 1 1 ...	Fail
1 0 0 1 ...	Fail
1 0 1 0 ...	Pass
-o	-c

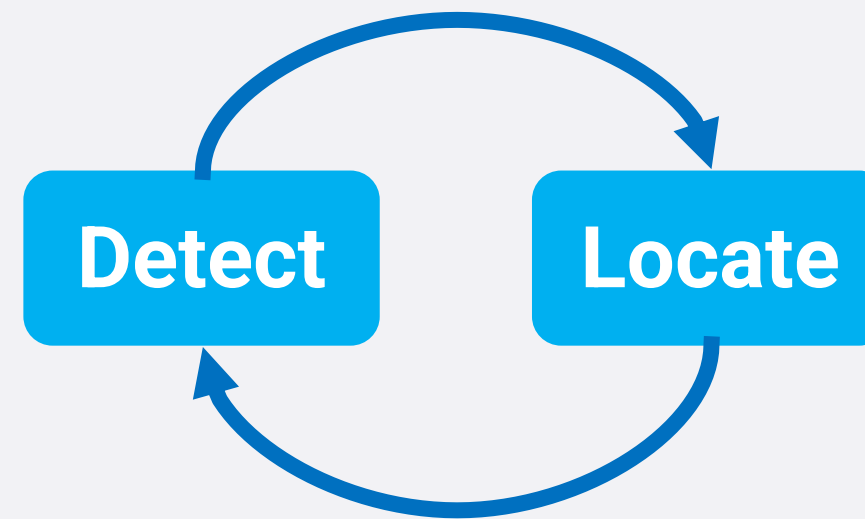
# Interleaving Combinatorial Testing (ICT)

Sequential



VS

Interleaving



避免了重复定位

减少冗余测试用例

# 反馈检验机制



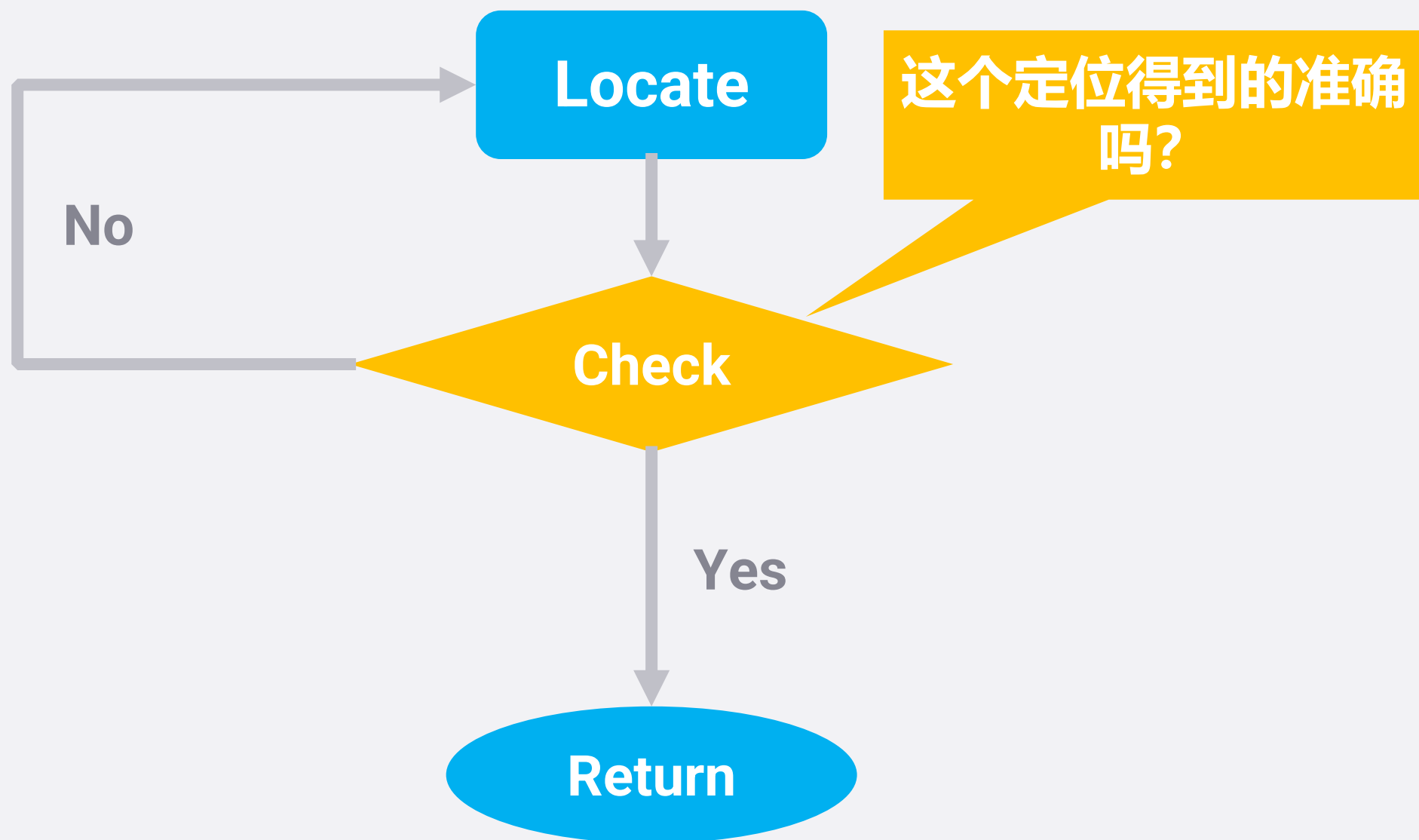
1	0	1	1	FAIL
0	0	1	1	PASS
1	1	1	1	FAIL
1	0	0	1	FAIL
1	0	1	0	PASS

-c

错误的



# 反馈检验机制



1	0	1	1	FAIL
0	0	1	1	FAIL
1	1	1	1	FAIL
1	0	0	1	FAIL
1	0	1	0	PASS

-c

2	2	2	1	PASS
---	---	---	---	------

1	0	1	1	FAIL
2	0	1	1	PASS
1	2	1	1	FAIL
1	0	2	1	FAIL
1	0	1	2	PASS

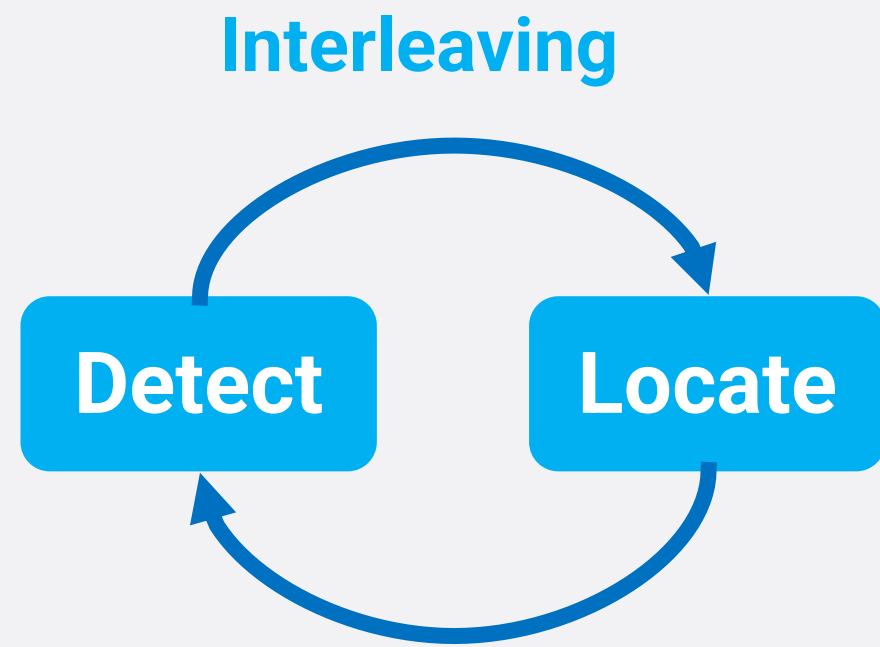
-0 -c

1	2	2	1	FAIL
---	---	---	---	------

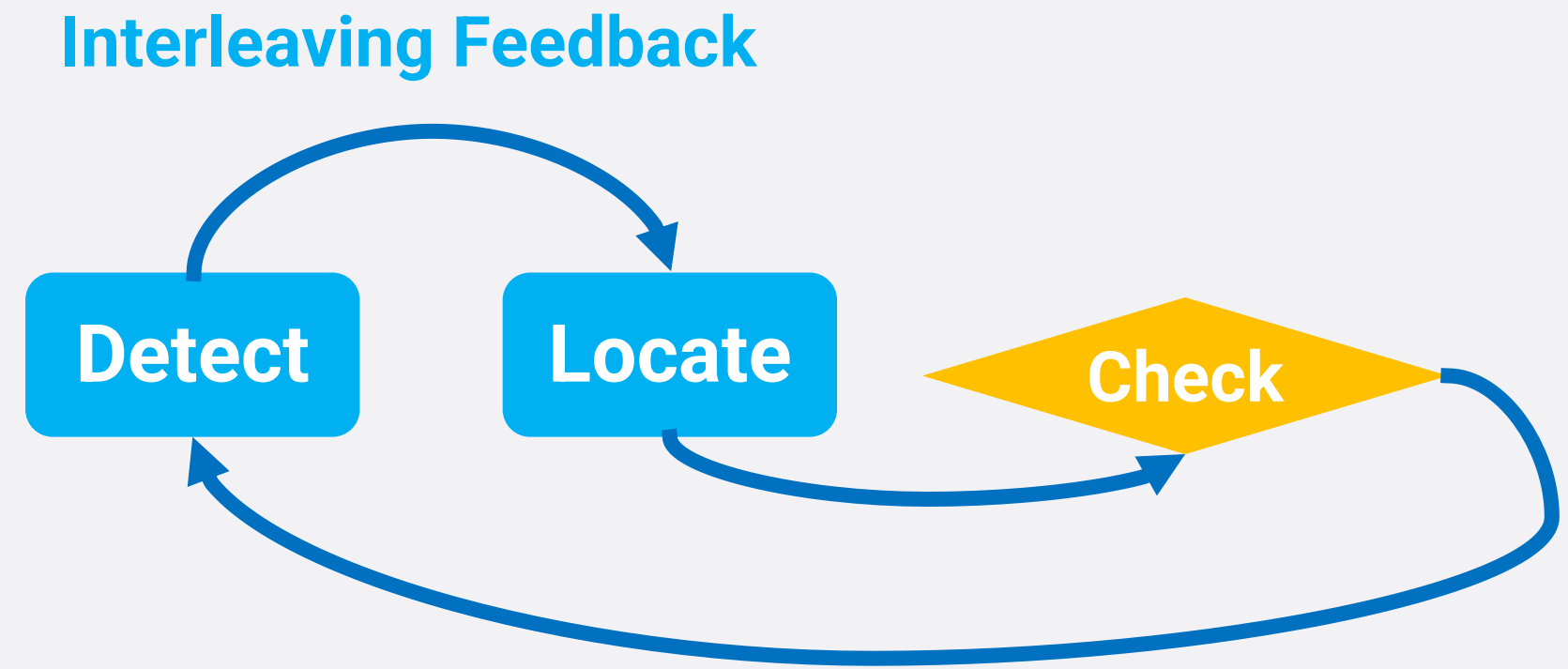
正确



# 反馈检验机制



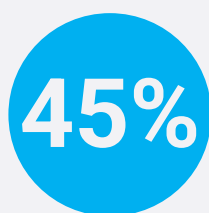
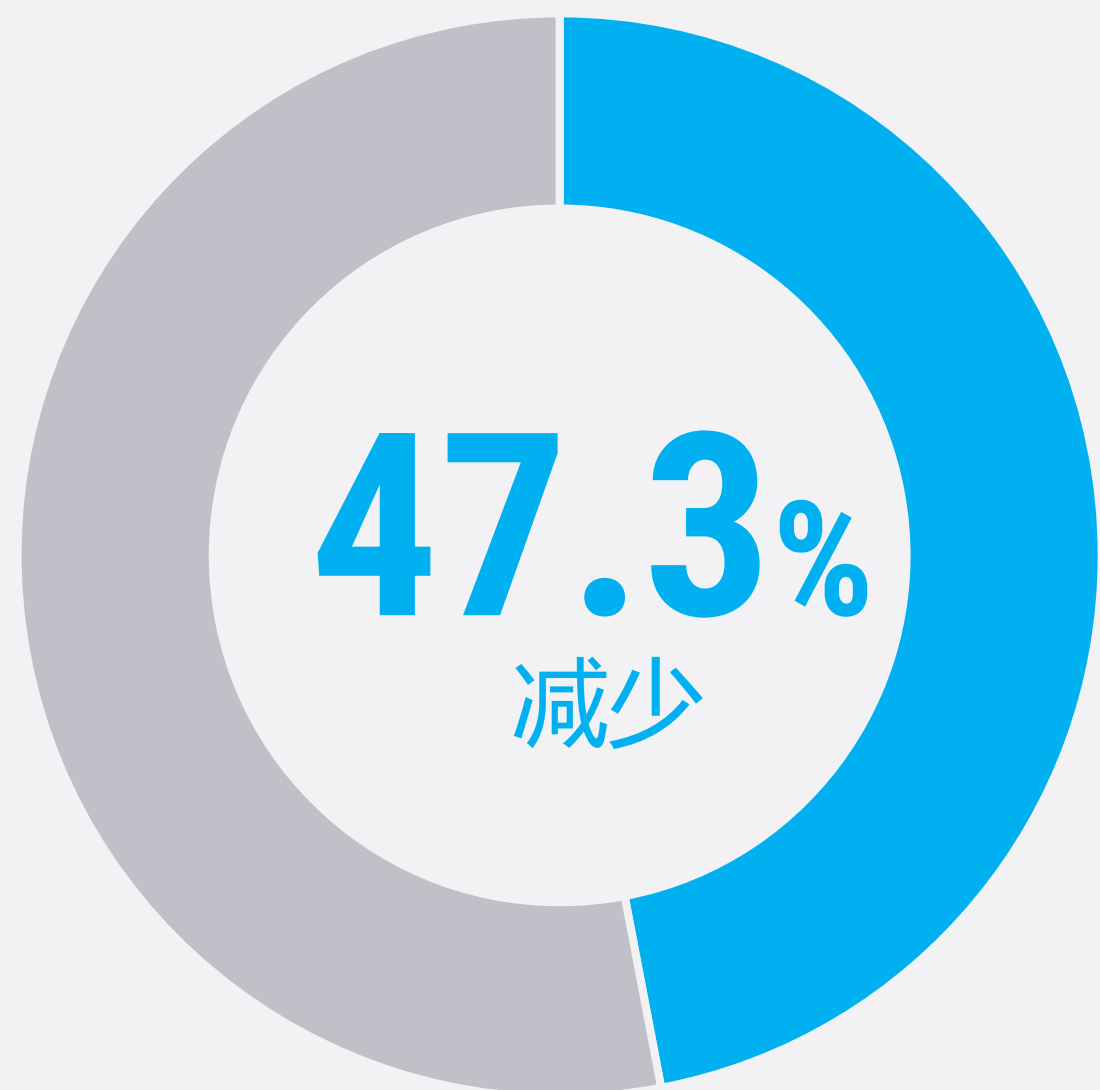
VS



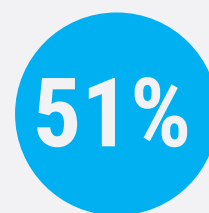
获得了多余的可以修复错误的机会

# 实验比较SCT和ICT

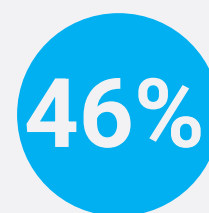
代价



2-way



3-way

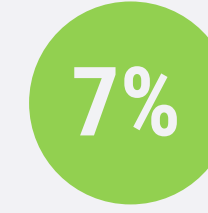


4-way

定位结果



2-way



3-way



4-way



# 实验比较ICT and ICT feedback

代价



定位结果



15%

2-way

16%

3-way

18%

4-way

6%

2-way

20%

3-way

30%

4-way







*Thanks*

钮鑫涛

南京大学计算机科学与技术系