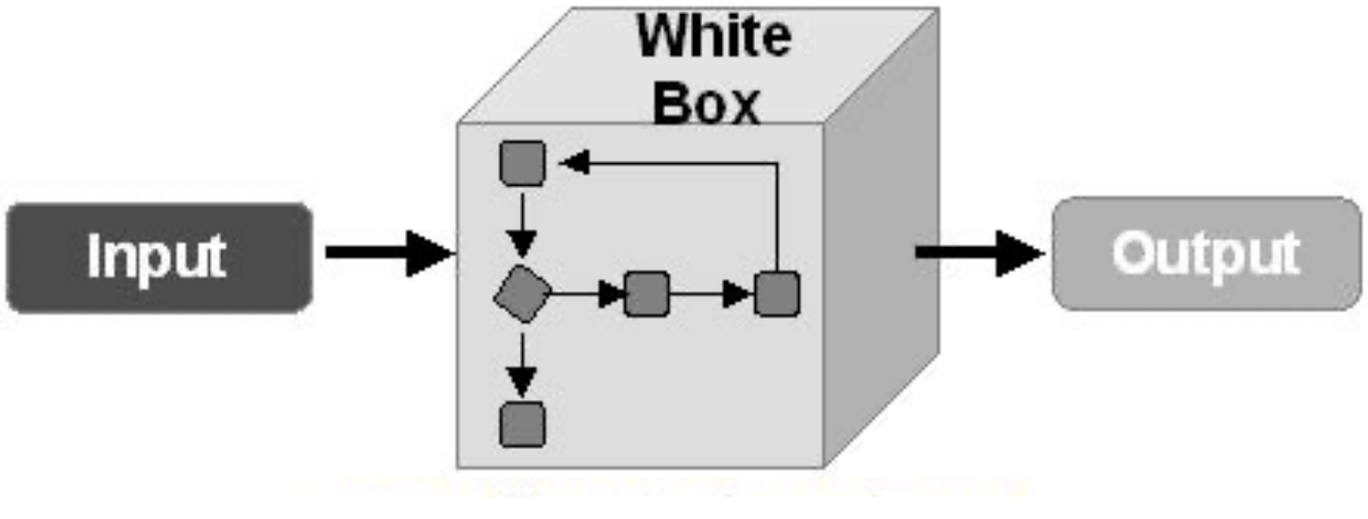
白盒测试 & 黑盒测试

White-Box Testing & Black-Box Testing

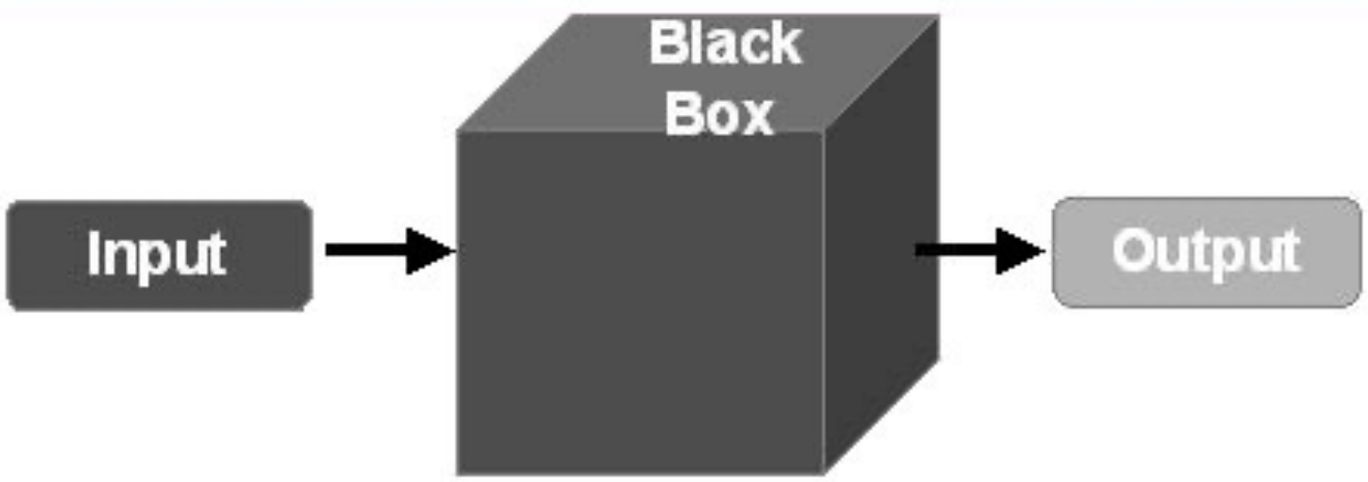
2026

白盒测试 & 黑盒测试



白盒测试

基于代码设计测试用例



黑盒测试

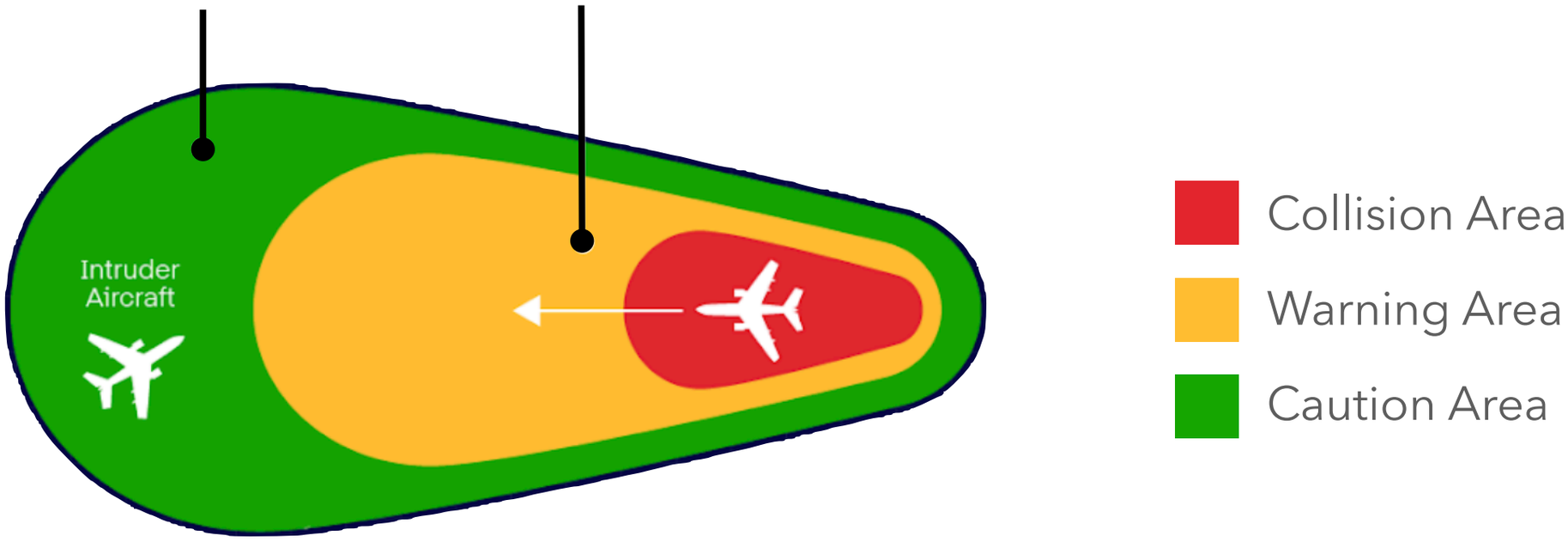
基于规格说明设计测试用例

TCAS

Traffic Collision Avoidance System

Traffic Advisory
(a warning to look for traffic
– no action required)

Resolution Advisory
(a command to climb or descend
immediately to avoid a collision)



TCAS

Traffic Collision Avoidance System

```
int Cur_Vertical_Sep;                // 当前垂直间隔
bool High_Confidence;                // 数据是否高度可信
bool Two_of_Three_Reports_Valid;     // 三份报告中是否至少两份有效
int Own_Tracked_Alt;                 // 本机高度
int Own_Tracked_Alt_Rate;            // 本机垂直速度
int Other_Tracked_Alt;               // 对方飞机高度
int Alt_Layer_Value;                 // 高度层级
int Up_Separation;                   // 向上避让的安全余量
int Down_Separation;                 // 向下避让的安全余量
int Other_RAC;                       // 对方飞机的避让意图
int Other_Capability;                // 对方飞机是否具有 TCAS 能力
int Climb_Inhibit;                   // 是否抑制爬升
```

```
// Resolution Advisory
#define UNRESOLVED 0                 // 不发出垂直避让建议
#define UPWARD_RA 1                  // 建议本机向上避让
#define DOWNWARD_RA 2                // 建议本机向下避让
```

白盒测试

回顾软件缺陷是如何触发的：Fault → Error → Failure

🤔 Fault 一定存在某代码片段中

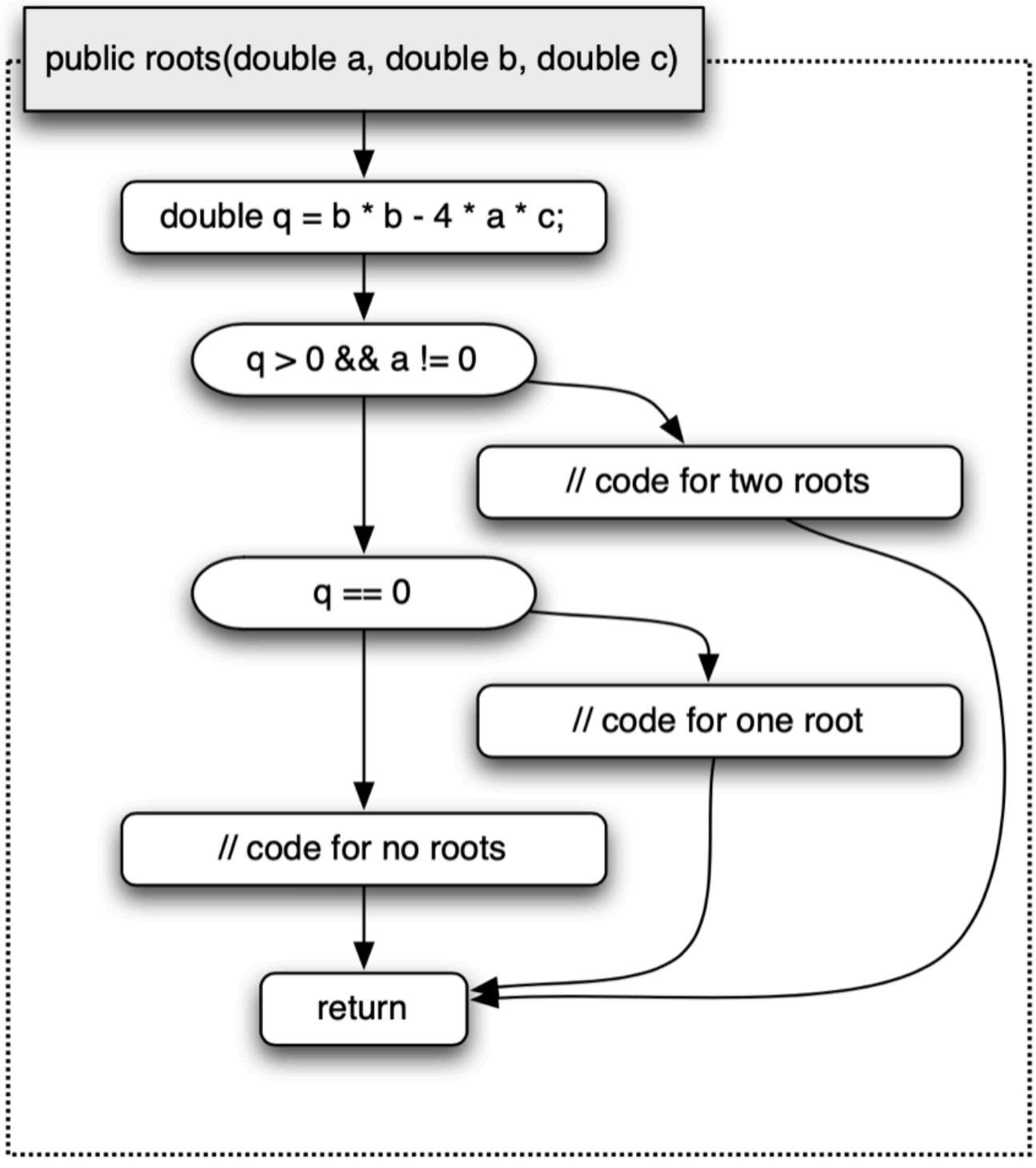
🤔 只有当特定代码片段被执行时才会触发 Fault

💡 把各种代码结构都覆盖一遍
(Structure Coverage)

白盒测试

控制流图 (CFG) 提供了刻画
代码结构覆盖的一种方式

- 覆盖 Nodes 和 Edges (Paths)
- 如果能覆盖得更多，则有更大的机会触发软件故障



语句覆盖

Statement Coverage

覆盖每个语句

```
if( High_Confidence &&                  // A
    Own_Tracked_Alt_Rate <= 600 &&      // B
    Cur_Vertical_Sep > 600 )            // C
```

	Confidence	Alt Rate	Vertical Sep
t_1	1	500	700

A	B	C
T	T	T

分支覆盖

Branch Coverage

覆盖每个判定的 True 分支和 False 分支

```
if( High_Confidence &&                  // A
    Own_Tracked_Alt_Rate <= 600 &&      // B
    Cur_Vertical_Sep > 600 )            // C
```

	Confidence	Alt Rate	Vertical Sep
t_1	1	500	700
t_2	0	500	700

A	B	C
T	T	T
F	T	T

Demo

LCOV - code coverage report

Current view:	top level		Coverage	Total	Hit
Test:	TCAS Coverage	**Lines:**	79.7 %	69	55
Test Date:	2026-08-23 20:36:31	**Functions:**	88.9 %	9	8
		Branches:	43.8 %	48	21

Directory	Line Coverage				Branch Coverage			Function Coverage		
	Rate		Total	Hit	Rate	Total	Hit	Rate	Total	Hit
tcas/		79.7 %	69	55	43.8 %	48	21	88.9 %	9	8

Generated by: LCOV version 2.5-0

分支覆盖

Branch Coverage

覆盖每个判定的 True 分支和 False 分支

(see tcas.c.gcov)

```
if( High_Confidence &&                    // A
    Own_Tracked_Alt_Rate <= 600 &&        // B
    Cur_Vertical_Sep > 600 )              // C
```

```
branch 0 taken 1
branch 1 taken 1
branch 2 taken 0
branch 3 taken 1
```

	Confidence	Alt Rate	Vertical Sep
t_1	1	500	700
t_2	0	500	700

A	B	C
T	T	T
F	T	T

🌶️ 但真实情况远比这个复杂 (C 语言的短路求值)

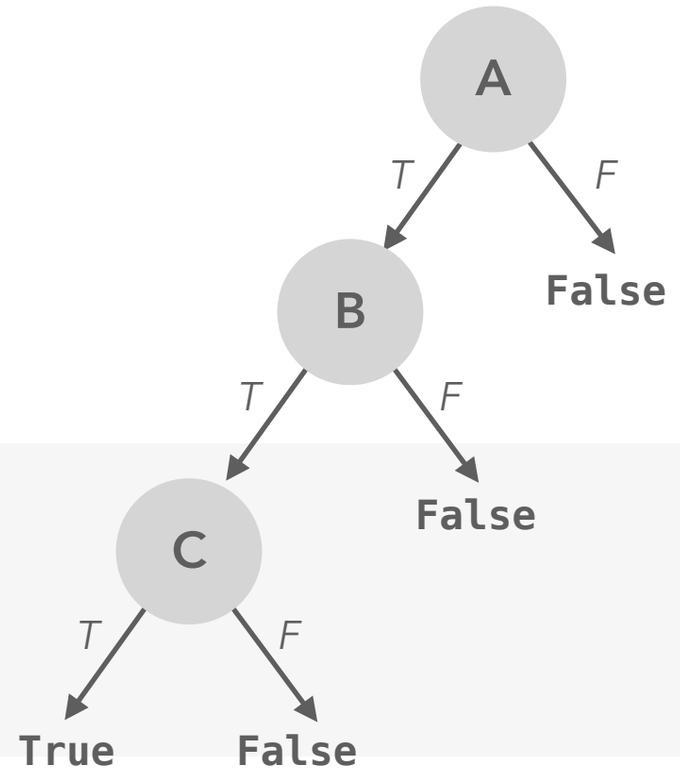
覆盖的是编译后的 CFG 中的每条控制流边

分支覆盖

Branch Coverage

覆盖每个判定的 True 分支和 False 分支

```
if( High_Confidence &&                    // A
    Own_Tracked_Alt_Rate <= 600 &&        // B
    Cur_Vertical_Sep > 600 )              // C
```



	Confidence	Alt Rate	Vertical Sep
t_1	1	500	700
t_2	0	500	700
t_3	1	700	700
t_4	1	500	500

A	B	C
T	T	T
F	T	T
T	F	T
T	T	F

🌶️ 此时才是 100% 的 Branch Coverage

条件覆盖
Condition Coverage

覆盖每个条件的 True 和 False 取值

```
if( High_Confidence &&                  // A
    Own_Tracked_Alt_Rate <= 600 &&      // B
    Cur_Vertical_Sep > 600 )            // C
```

	Confidence	Alt Rate	Vertical Sep
t_1	1	500	700
t_3	0	700	500

A	B	C
T	T	T
F	F	F

🌶️ 这里也存在 C 语言的短路求值

条件覆盖
Condition Coverage

覆盖每个条件的 True 和 False 取值

```
if( High_Confidence &&                  // A
    Own_Tracked_Alt_Rate <= 600 &&      // B
    Cur_Vertical_Sep > 600 )            // C
```

	Confidence	Alt Rate	Vertical Sep
t_1	1	500	700
t_2	0	500	700
t_3	1	700	700
t_4	1	500	500

A	B	C
T	T	T
F	T	T
T	F	T
T	T	F

条件组合覆盖

Compound Condition Coverage

覆盖 *N* 个条件的所有 2^N 种取值组合

```
if( High_Confidence &&                  // A
    Own_Tracked_Alt_Rate <= 600 &&      // B
    Cur_Vertical_Sep > 600 )            // C
```

	Confidence	Alt Rate	Vertical Sep
t_1	1	500	700
t_2	0	500	700
t_3	1	700	700
t_4	1	500	500
t_5	1	700	500
t_6	0	500	500
t_7	0	700	700
t_8	0	700	500

A	B	C
T	T	T
F	T	T
T	F	T
T	T	F
T	F	T
F	T	T
F	F	F
F	F	T

修改决策条件覆盖

Modified Condition Decision Coverage (MC/DC)

每个条件独立影响判定的真假 (条件值的改变导致整个判定值的改变)

```
if( High_Confidence &&                    // A
    Own_Tracked_Alt_Rate <= 600 &&        // B
    Cur_Vertical_Sep > 600 )              // C
```

	Confidence	Alt Rate	Vertical Sep
t_1	1	500	700
t_2	0	500	700
t_3	1	700	700
t_4	1	500	500

A	B	C
T	T	T
F	T	T
T	F	T
T	T	F

修改决策条件覆盖

Modified Condition Decision Coverage (MC/DC)

每个条件独立影响判定的真假 (条件值的改变导致整个判定值的改变)

```
if( High_Confidence &&                    // A
    Own_Tracked_Alt_Rate <= 600 &&        // B
    Cur_Vertical_Sep > 600 )              // C
```

	Confidence	Alt Rate	Vertical Sep
t_1	1	500	700
t_2	0	500	700
t_3	1	700	700
t_4	1	500	500

A	B	C
T	T	T
F	T	T
T	F	T
T	T	F

改变条件 A 的值导致整个判定结果的改变

修改决策条件覆盖

Modified Condition Decision Coverage (MC/DC)

每个条件独立影响判定的真假 (条件值的改变导致整个判定值的改变)

```
if( High_Confidence &&                  // A
    Own_Tracked_Alt_Rate <= 600 &&      // B
    Cur_Vertical_Sep > 600 )            // C
```

	Confidence	Alt Rate	Vertical Sep
t_1	1	500	700
t_2	0	500	700
t_3	1	700	700
t_4	1	500	500

A	B	C
T	T	T
F	T	T
T	F	T
T	T	F

改变条件 B 的值导致整个判定结果的改变

修改决策条件覆盖

Modified Condition Decision Coverage (MC/DC)

每个条件独立影响判定的真假 (条件值的改变导致整个判定值的改变)

```
if( High_Confidence &&                    // A
    Own_Tracked_Alt_Rate <= 600 &&        // B
    Cur_Vertical_Sep > 600 )              // C
```

	Confidence	Alt Rate	Vertical Sep
t_1	1	500	700
t_2	0	500	700
t_3	1	700	700
t_4	1	500	500

A	B	C
T	T	T
F	T	T
T	F	T
T	T	F

改变条件 C 的值导致整个判定结果的改变

修改决策条件覆盖

Modified Condition Decision Coverage (MC/DC)

每个条件独立影响判定的真假 (条件值的改变导致整个判定值的改变)

```
if( High_Confidence &&                  // A
    Own_Tracked_Alt_Rate <= 600 &&      // B
    Cur_Vertical_Sep > 600 )            // C
```

	Confidence	Alt Rate	Vertical Sep
t_1	0	700	500
t_2	1	700	500
t_3	1	500	500
t_4	1	500	700

A	B	C
F	F	F
T	F	F
T	T	F
T	T	T

这满足 MC/DC 吗?

Unique-Cause MC/DC 🆚 Masking MC/DC

Quiz

请为下列代码设计满足修改决策条件覆盖 (MC/DC) 的测试用例集

```
if ( enabled &&
        ((tcas_equipped && intent_not_known) ||
         !tcas_equipped)
   )
```

修改决策条件覆盖

Modified Condition Decision Coverage (MC/DC)

😊 MC/DC 是在测试很多关键软件时需要满足的测试覆盖准则

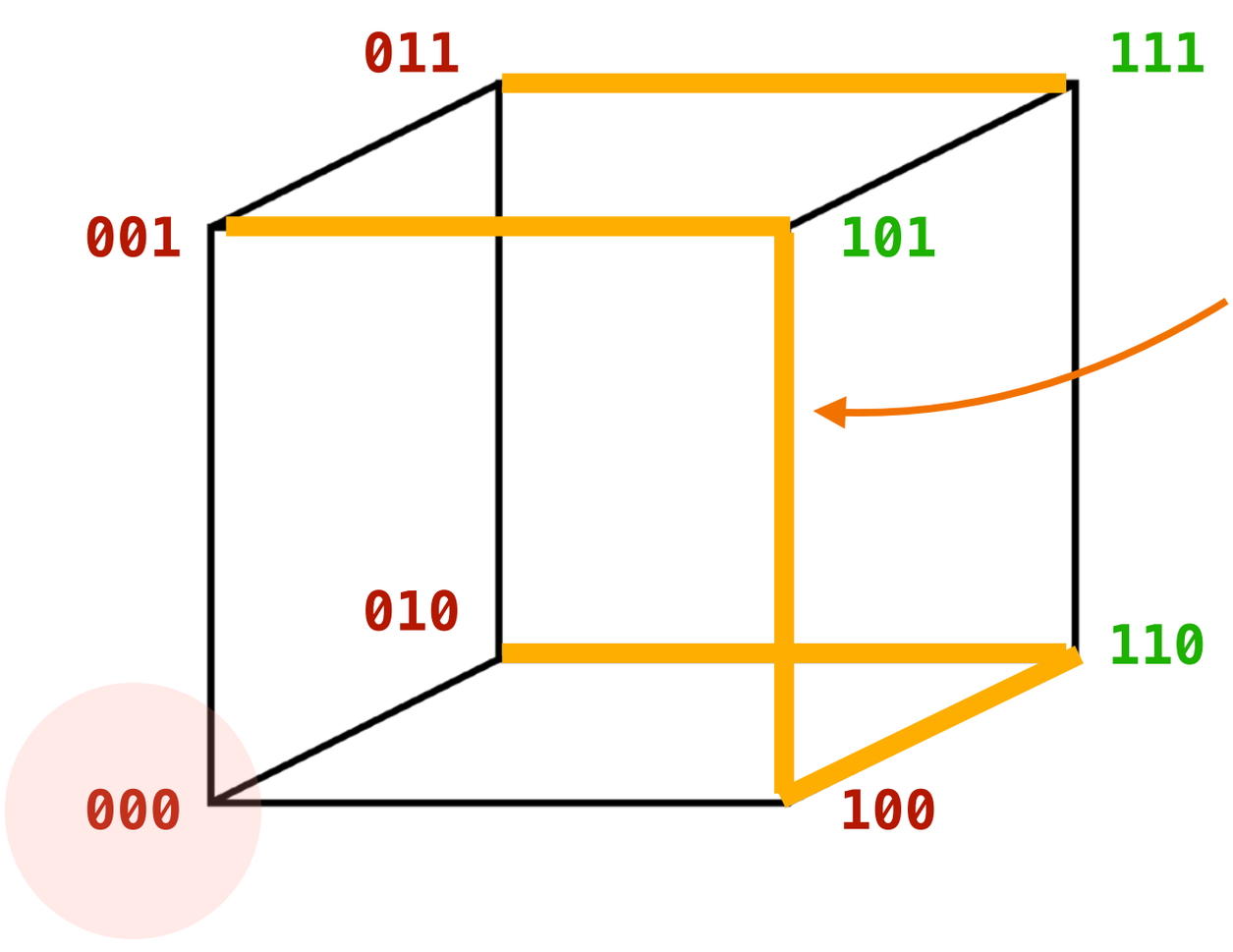
- 航空 (DO-178B/C)、汽车 (ISO 26262)、电子 (IEC 61508)、医疗 (IEC 62304)、铁路 (EN 50128) 等
- 其对 "每个条件显示对决策结果产生独立影响" 的要求也在一定程度上涉及了对 Specification 的覆盖

修改决策条件覆盖

Modified Condition Decision Coverage (MC/DC)

☹️ 但满足 MC/DC 未必意味着一定 “测试充分”

```
if( A && (B || C) )
```



符合 MC/DC 的测试用例对

修改决策条件覆盖

Modified Condition Decision Coverage (MC/DC)

☹️ 但满足 MC/DC 未必意味着一定 “测试充分”

```
if( A && (B || C) )    // 正确实现应为 A XNOR (B || C)
```

"同或"：两个输入相同时结果为 True，不同时结果为 False

A	B	C
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Expected	Actual
1	0
0	0
0	0
0	0
0	0
1	1
1	1
1	1

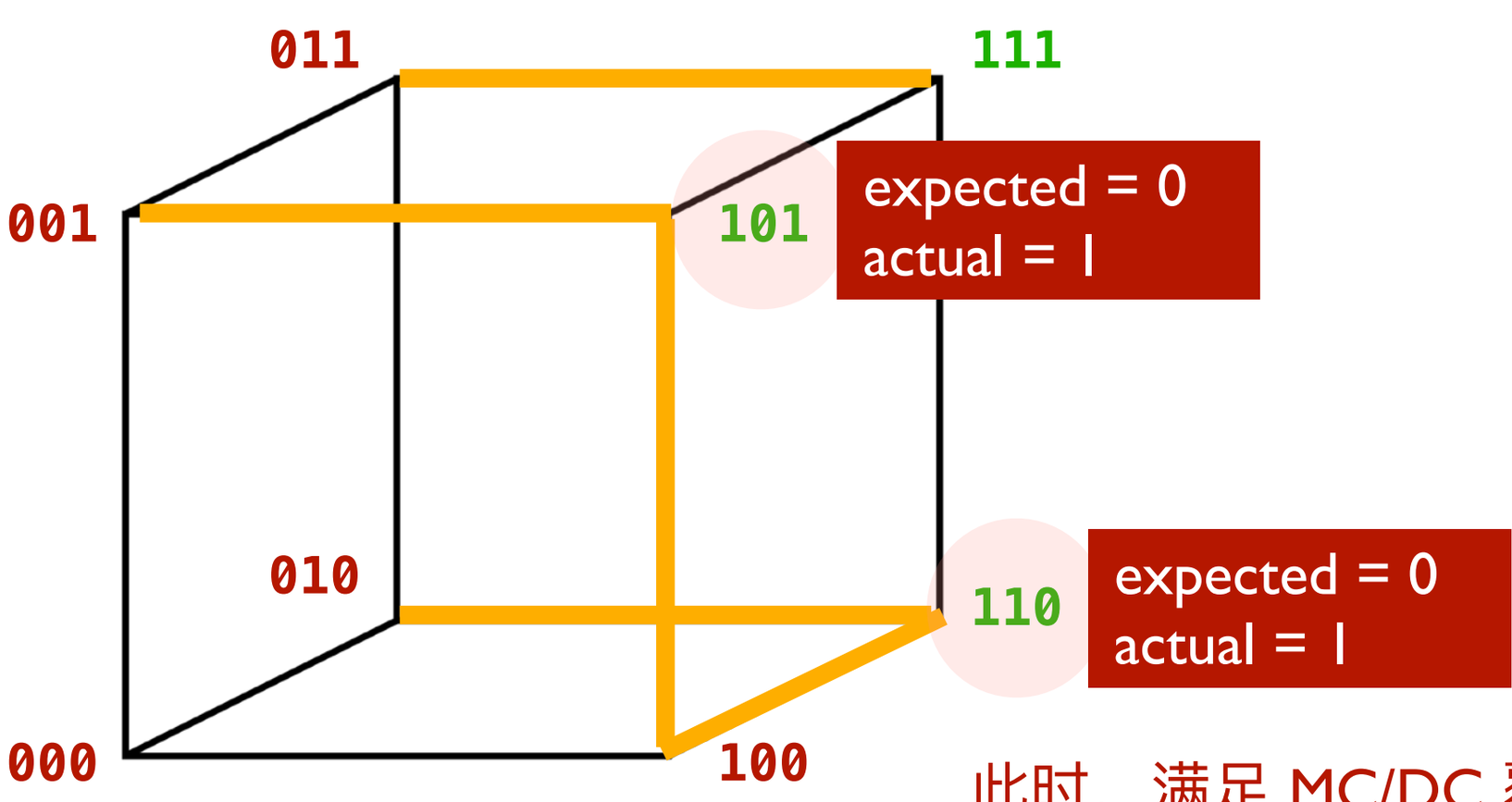
此时，唯一能触发缺陷的测试用例并不会被 MC/DC 选到

修改决策条件覆盖

Modified Condition Decision Coverage (MC/DC)

☹️ 开发人员甚至可以通过 “改写” 源程序来让 MC/DC 覆盖变得简单

```
if( A && (B || C) )      // 正确实现应为 A && (B && C)
```



此时，满足 MC/DC 覆盖的测试用例集能触发该缺陷

修改决策条件覆盖

Modified Condition Decision Coverage (MC/DC)

☹️ 开发人员甚至可以通过 “改写” 源程序来让 MC/DC 覆盖变得简单

```
// if( A && (B || C) )   // 正确实现应为 A && (B && C)
bool S = B || C
if (A && S)
```

A	B	C
0	0	1
0	1	0
1	0	0
1	1	1

满足对 B || C 的 MC/DC 覆盖

满足对 A && S 的 MC/DC 覆盖

修改决策条件覆盖

Modified Condition Decision Coverage (MC/DC)

☹️ 开发人员甚至可以通过 “改写” 源程序来让 MC/DC 覆盖变得简单

```
// if( A && (B || C) )   // 正确实现应为 A && (B && C)
bool S = B || C
if (A && S)
```

A	B	C
0	0	1
0	1	0
1	0	0
1	1	1

Expected	Actual
0	0
0	0
0	0
1	1

但这四个测试用例并不满足对原始表达式的 MC/DC 覆盖

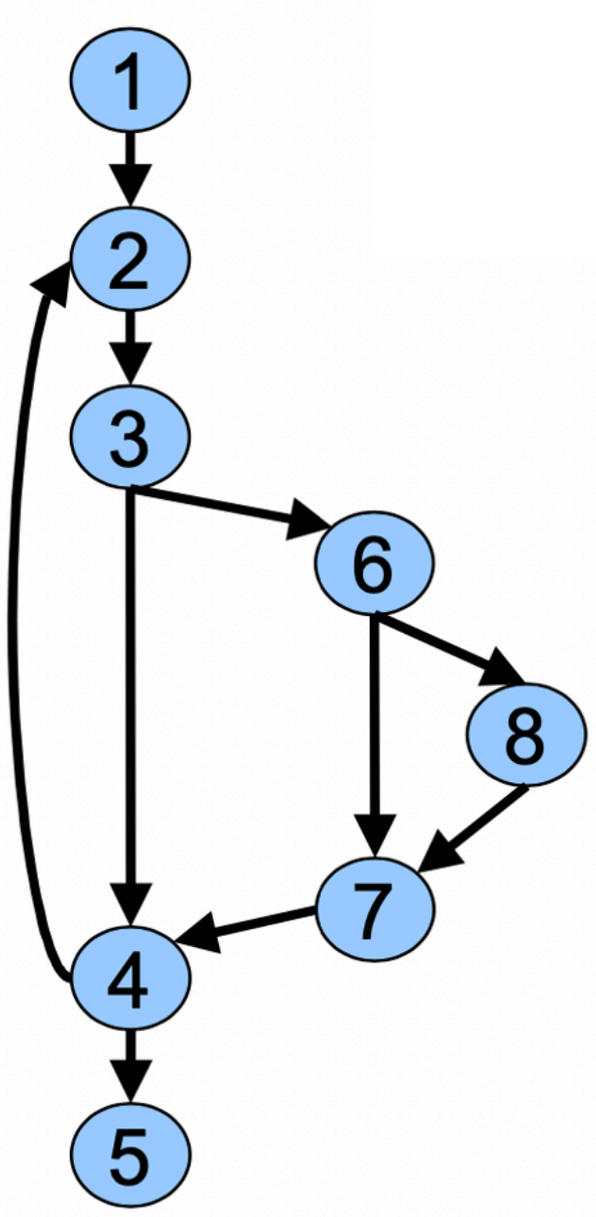
能让更多的测试用例集满足 MC/DC，但也要付出 “降低测试能力” 的代价

路径覆盖

All Path Coverage

覆盖代码中所有可能的执行路径

- 针对 CFG 的穷尽测试
- 由于 unbounded loops 的存在显然不可行



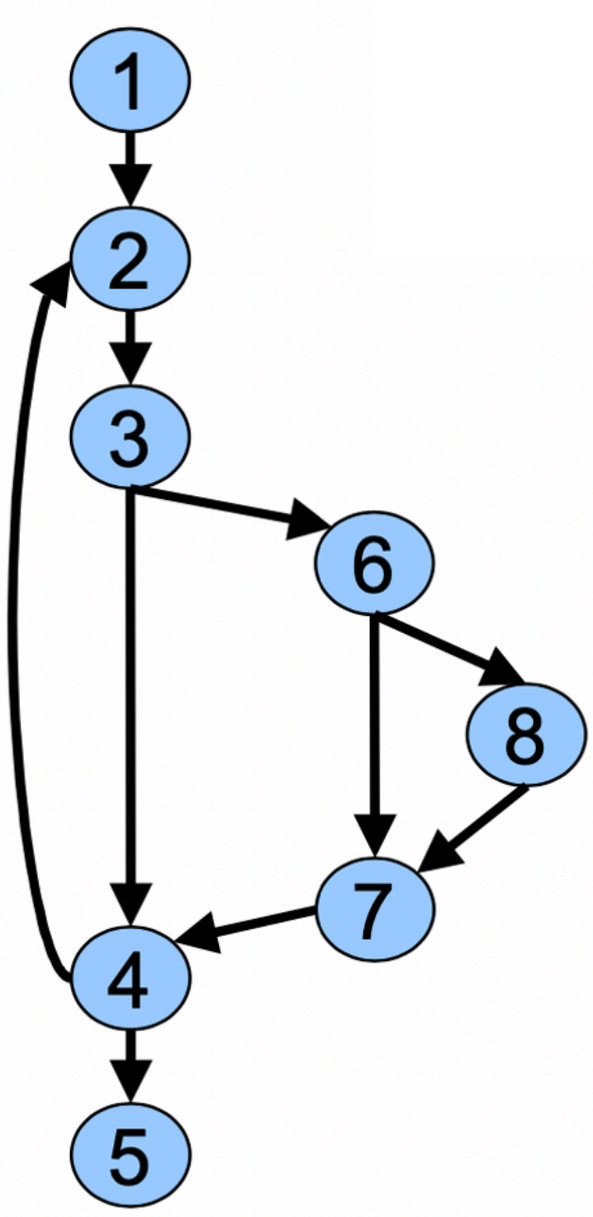
内部边界路径覆盖

Boundary Interior Path Coverage

覆盖一次循环中的所有可能路径

- 在重复执行循环内的路径时，将相同的子路径进行合并

Path 1	2 → 3 → 4 → 2
Path 2	2 → 3 → 6 → 7 → 4 → 2
Path 3	2 → 3 → 6 → 8 → 7 → 4 → 2



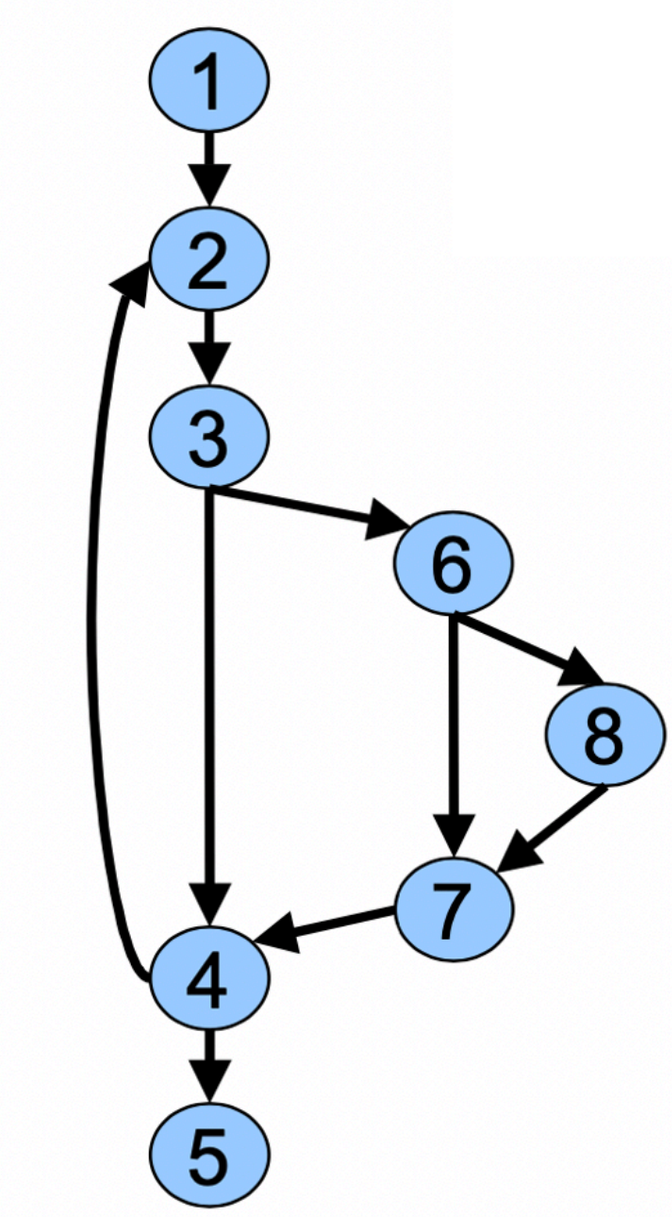
- 所需覆盖的路径数量仍然可能十分庞大
- 还可限定每个循环执行特定的次数 (Loop Boundary Coverage)

基本路径覆盖

Basis Path Coverage

覆盖代码中的独立路径

- 以一个相互线性独立 (linearly independent) 的基本路径集合来刻画整个 CFG 的路径空间
- 基本路径的数量 (upper bound) = CFG 的圈复杂度 (cyclomatic complexity)



$$V(G) = E - N + 2P = 4$$

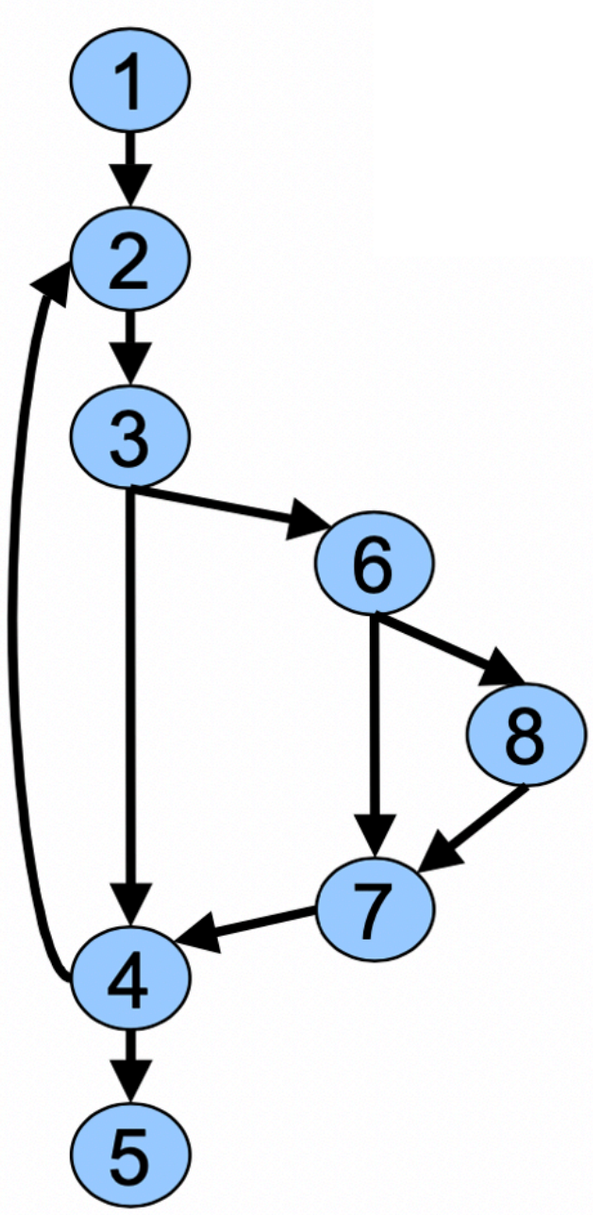
基本路径覆盖

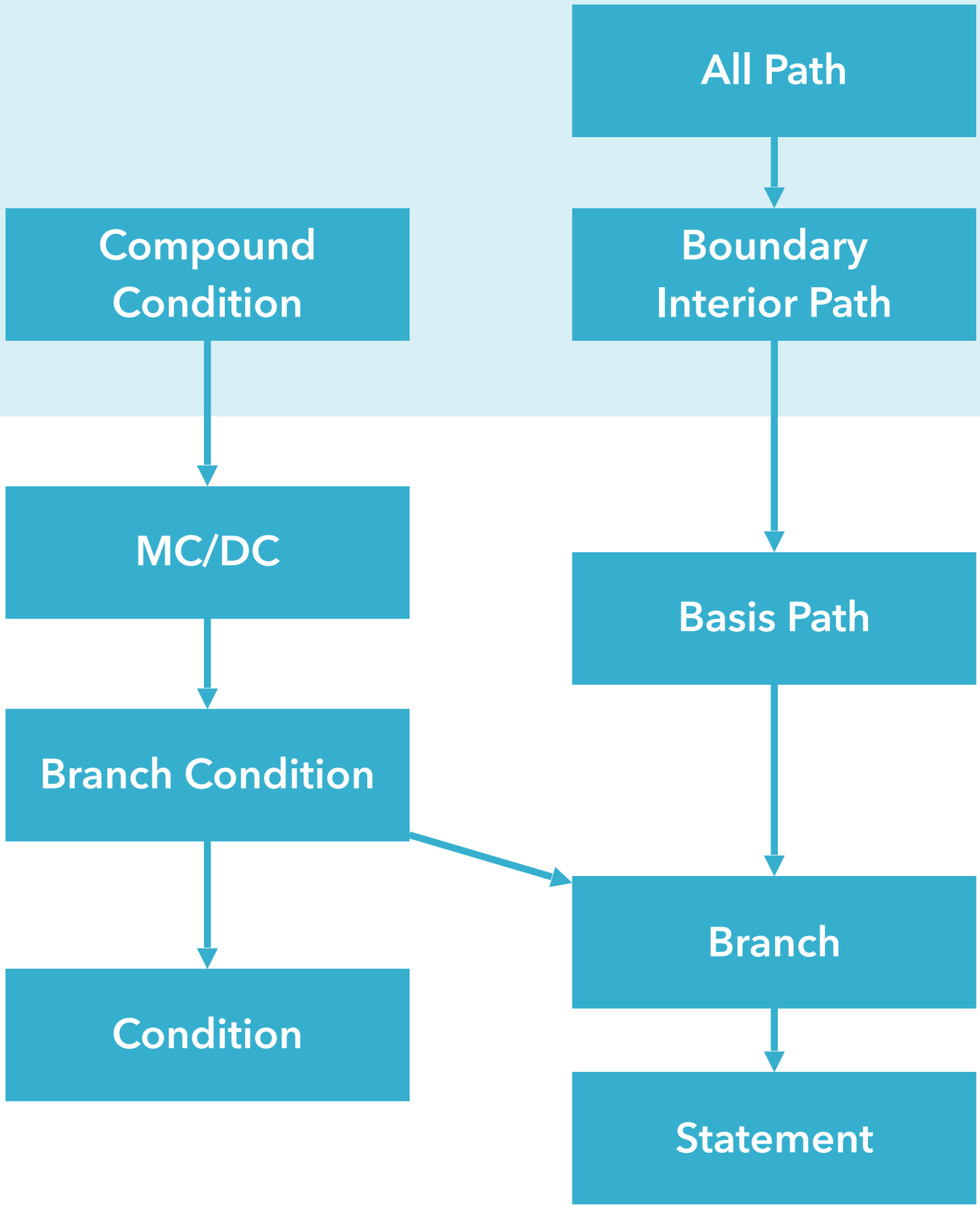
Basis Path Coverage

覆盖代码中的独立路径

- 为每条基本路径设计一条测试用例，该路径至少包含一条未曾访问过的边

Path 1	1 → 2 → 3 → 4 → 5
Path 2	1 → 2 → 3 → 4 → 2 → 3 → 4 → 5
Path 3	1 → 2 → 3 → 6 → 7 → 4 → 5
Path 4	1 → 2 → 3 → 6 → 8 → 7 → 4 → 5
Path 5	1 → 2 → 3 → 4 → 2 → 3 → 6 → 7 → 4 → 5





* 基于理论上的覆盖率定义

代码覆盖并不容易做到

Reaching specific code can be very hard

⚠️ 即使是最良好设计和维护的软件代码中仍然可能存在无法执行和覆盖的代码 (e.g., due to defensive programming)

```
// require Own_Tracked_Alt < Other_Tracked_Alt
need_upward_RA = Non_Crossing_Biased_Climb() &&
                 Own_Below_Threat();

// require Other_Tracked_Alt < Own_Tracked_Alt
need_downward_RA = Non_Crossing_Biased_Descend() &&
                   Own_Above_Threat();

if (need_upward_RA && need_downward_RA)
  alt_sep = UNRESOLVED;
```

代码覆盖也并不保证任何事情

Coverage does not guarantee anything

⚠️ 代码覆盖仅能告诉我们哪部分代码得到了执行，而不能告诉我们代码是否正确地执行

```
int Positive_RA_Alt_Thresh[4];

int ALIM(void) {
  // using any value in [0, 3] will reach 100% code coverage
  // but still fail to trigger the array out-of-bound access
  return Positive_RA_Alt_Thresh[Alt_Layer_Value];
}
```

正确认识代码覆盖

测试人员设计测试用例的首要目的就是提高代码覆盖率？

- 较高的代码覆盖率并不总是意味着较高的测试质量
  - 但较高的代码覆盖率终归是有益的
  - 什么都测一点总比大部分代码都没有被测到要好
- 尝试将代码覆盖率提高到某个级别 (e.g., 60% as acceptable, 75% as commendable, 90% as exemplary)，而不是盲目追求绝对高覆盖率
- 分析没有被覆盖的代码及其潜在行为，评估测试风险
- 不同的代码覆盖准则之间有强弱关系，一个较强的覆盖准则往往需要更多的测试用例，同时也更有可能在测试中触发故障 (cost-effectiveness 的权衡)