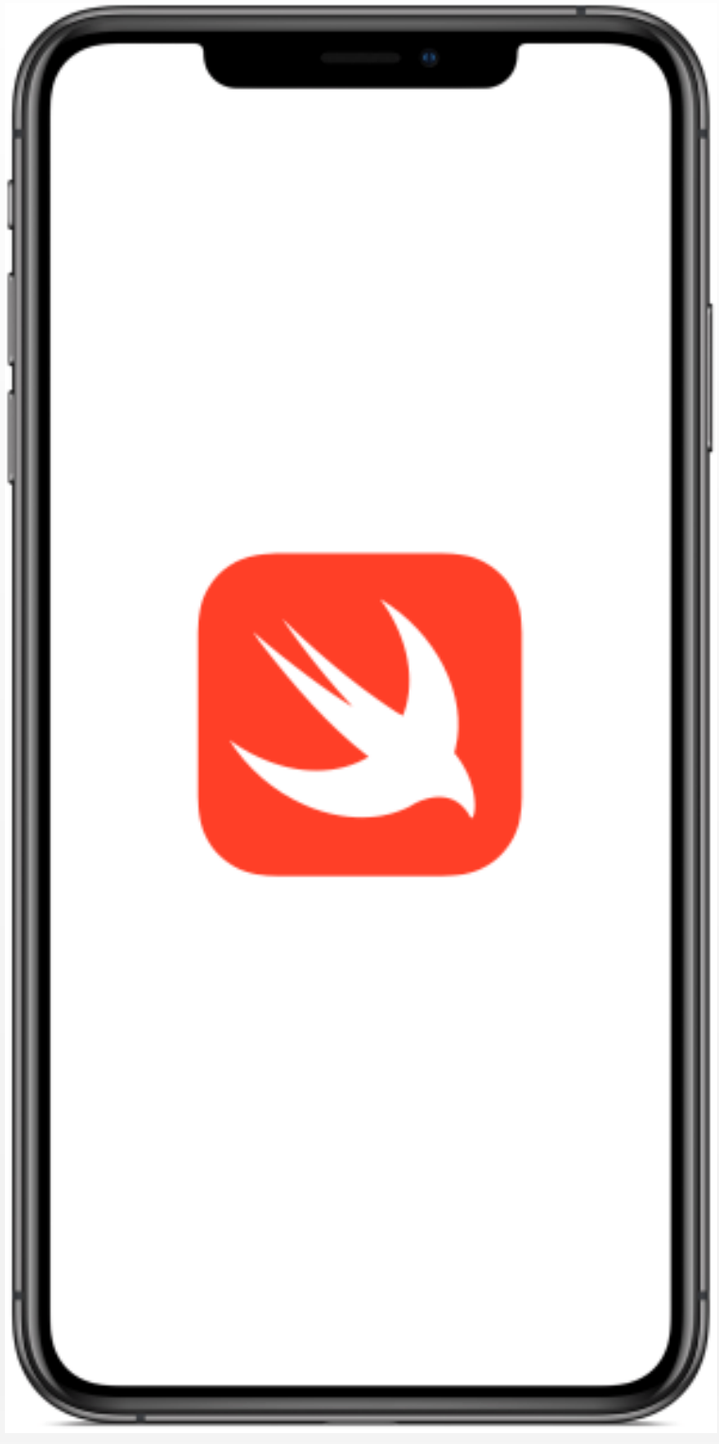
Swift在MCU中的应用技术

深圳市菲凯特科技有限公司
刘鹏



今日议题

- Swift语言背景简介
- 为什么要将Swift用于MCU应用开发
- 项目的基本技术原理
- 案例分析

Swift语言背景简介

Swift是一门广泛吸收现代语言优势的通用编程语言，它专注于**安全**，**性能**和**现代的设计模式**

Swift语言背景简介

安全

- 所有变量在使用前必须进行初始化
- 数组索引会检查越界错误，整数会检查溢出
- 内存可以自动管理，不需要手动申请和释放
- 引入可选类型，避免了null引起的bug

性能

- 纯编译型静态类型语言，没有VM，代码直接编译为机器代码
- 没有GC，运行状态一致和可预测，这是实时系统的必要条件

```swift
// Swift版本的Hello World
print("Hello, world!")

// var声明变量
var myVariable = 42
myVariable = 50

// let声明常量
let myConstant = 666

// 可选类型才能为空，使用时必须特殊处理
var name: String? = nil
name = "andy"
if let value = name {
    print(value)
} else {
    print("字符串为空")
}
```

Swift语言背景简介

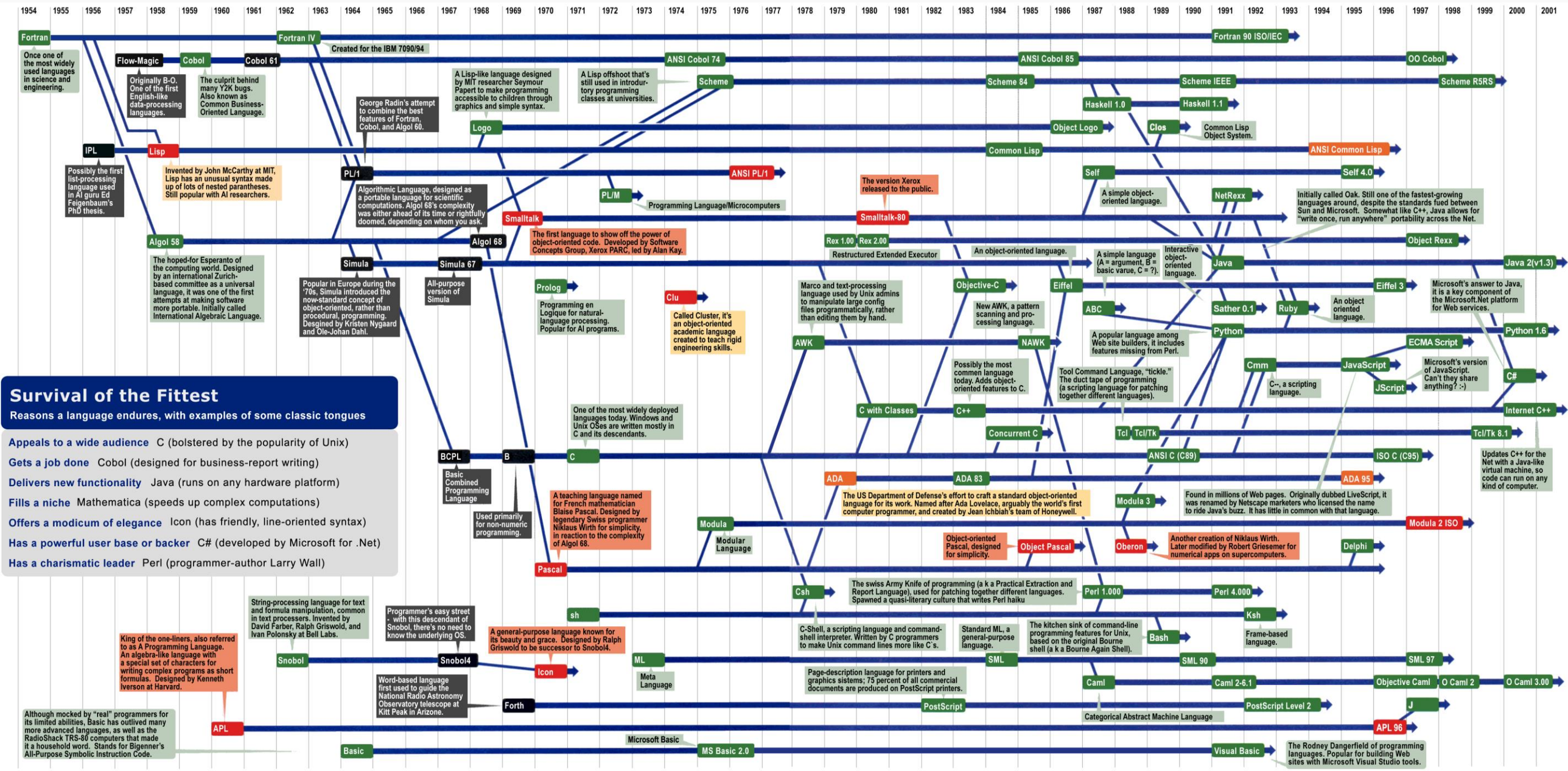
富有表现力

- 第一次实现了让工业级系统语言有着脚本语言一样的表现力
- 代码范式多样，支持面向对象，面向协议，函数式编程
- 学习曲线平滑，可作为初学者的第一门编程语言

为什么要将Swift用于MCU应用开发

MCU开发只能用C语言吗?

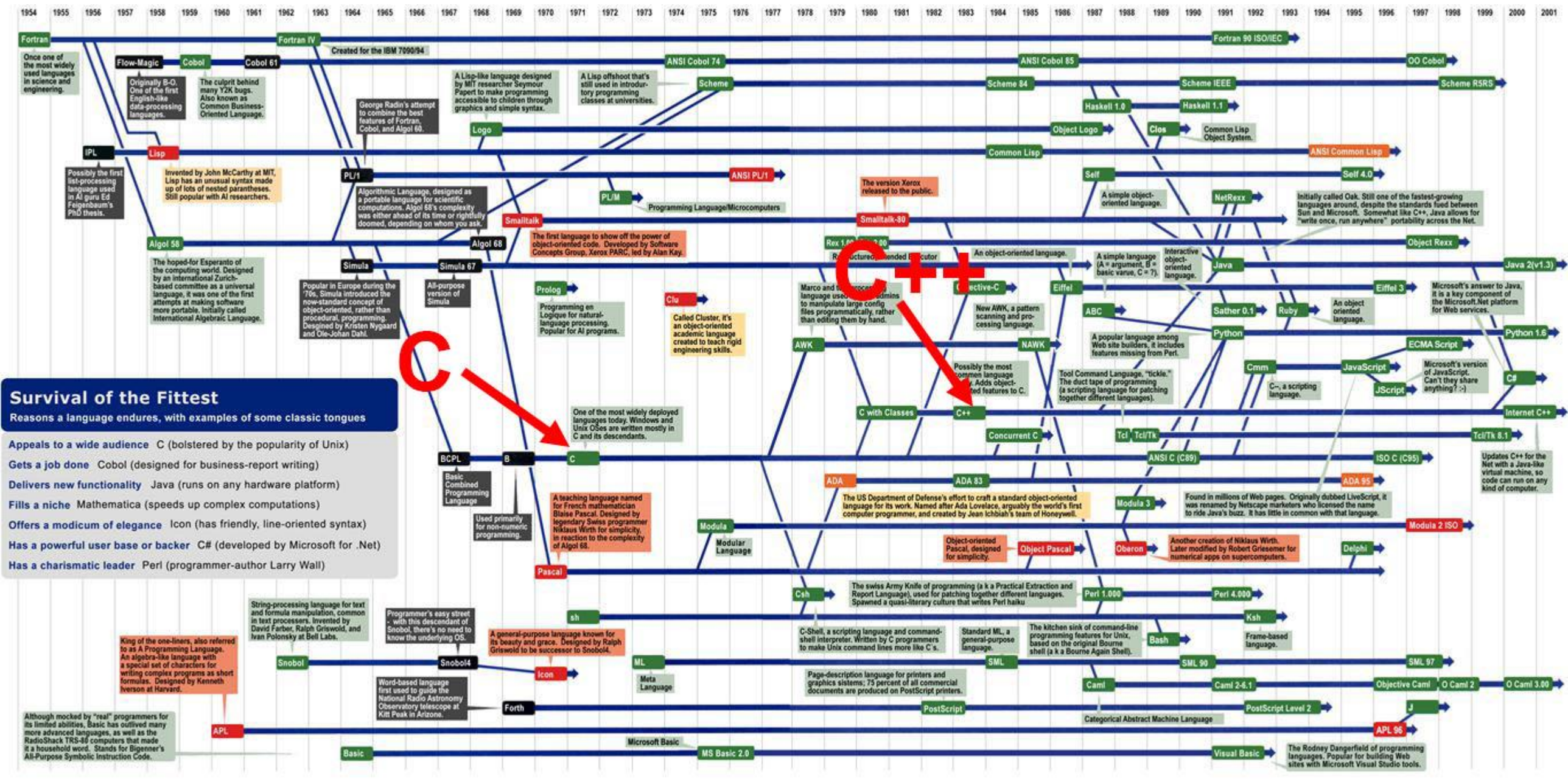
- 最经典的8051 MCU诞生于1981年，至今已接近40年
- C语言诞生于1972年，至今已经接近半个世纪
- C++语言诞生于1983年，至今已接近40年



为什么要将Swift用于MCU应用开发

MCU开发只能用C语言吗?

- 最经典的8051 MCU诞生于1981年，至今已接近40年
- C语言诞生于1972年，至今已经接近半个世纪
- C++语言诞生于1983年，至今已接近40年

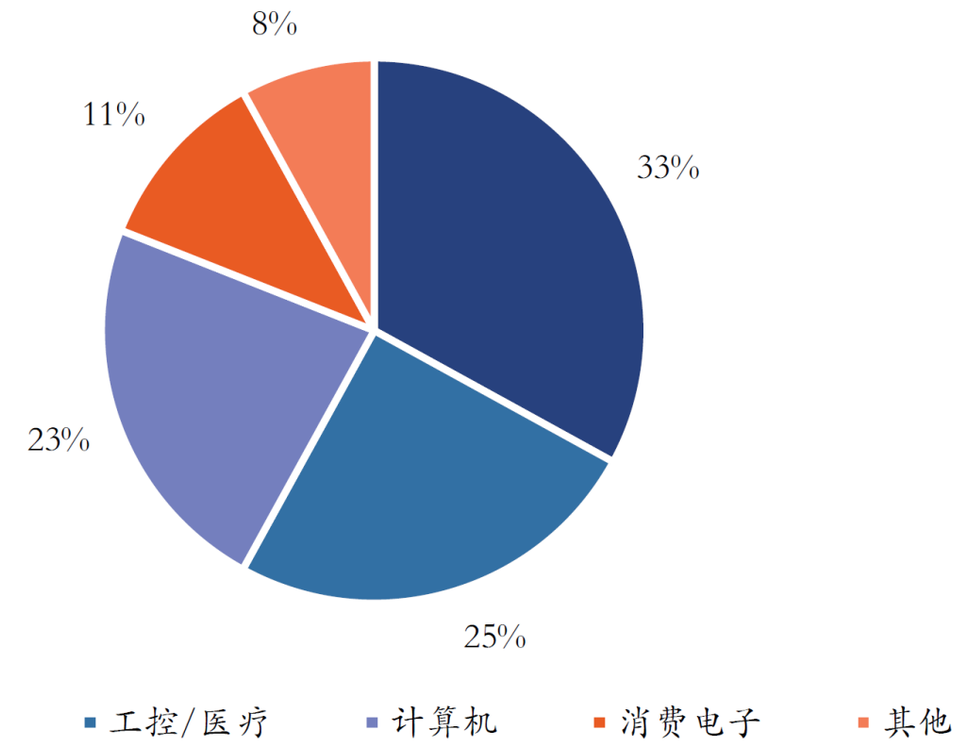


为什么要将Swift用于MCU应用开发

MCU硬件碎片化，软件开发方式单一

- **硬件碎片化：** MCU诞生的初衷是为了控制机器运转，应用场景各不相同，MCU硬件也是百花齐放
- **业务模型简单：**在MCU控制领域，软件业务模型通常比较固定，更多工作在于完成底层硬件的衔接驱动
- **各自为政：**传统包含MCU产品的设计和实施通常是由内部团队独立完成，与外界协同开发的需求不高

全球 MCU 各应用领域占比

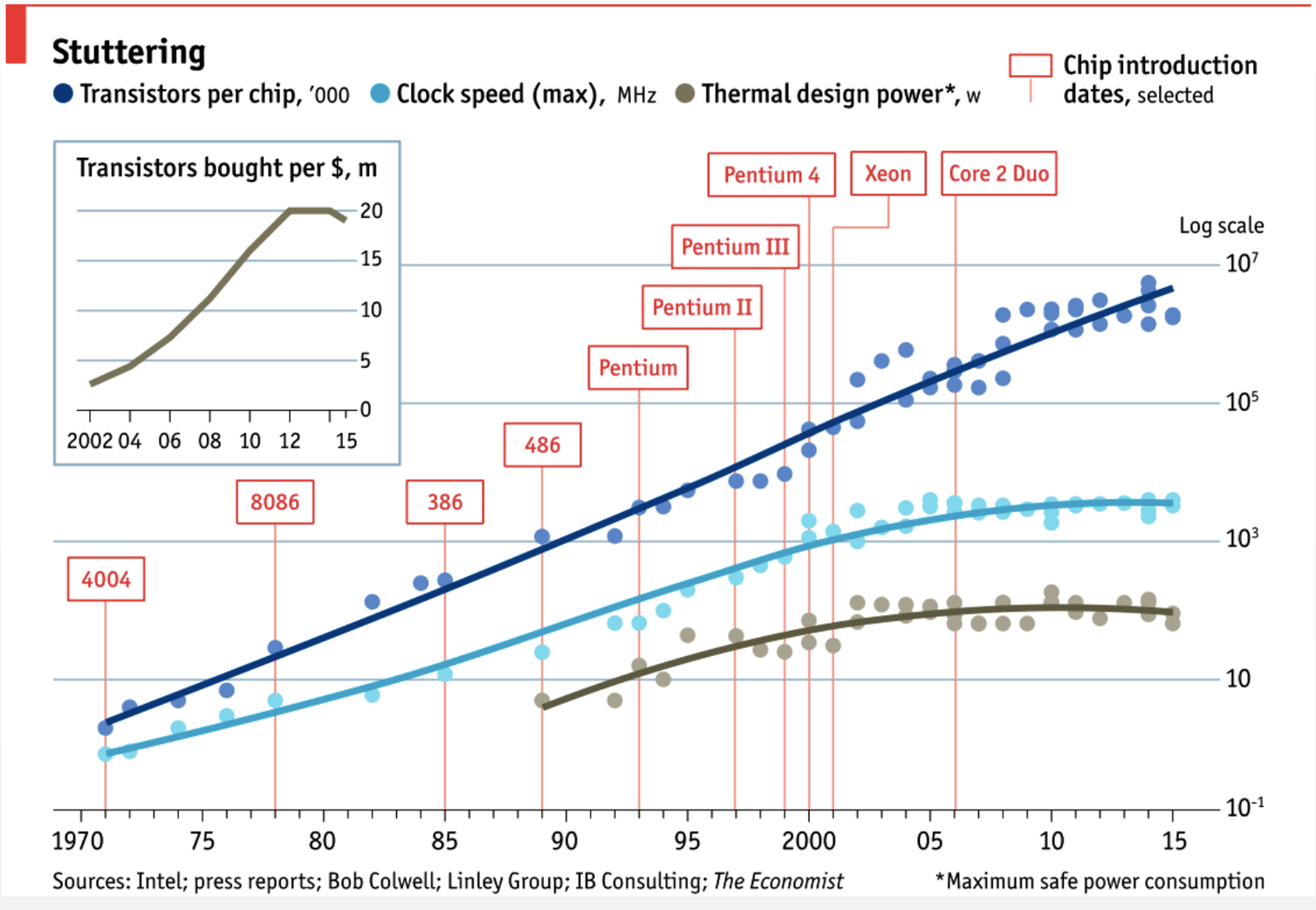


资料来源：iFind，基业常青

为什么要将Swift用于MCU应用开发

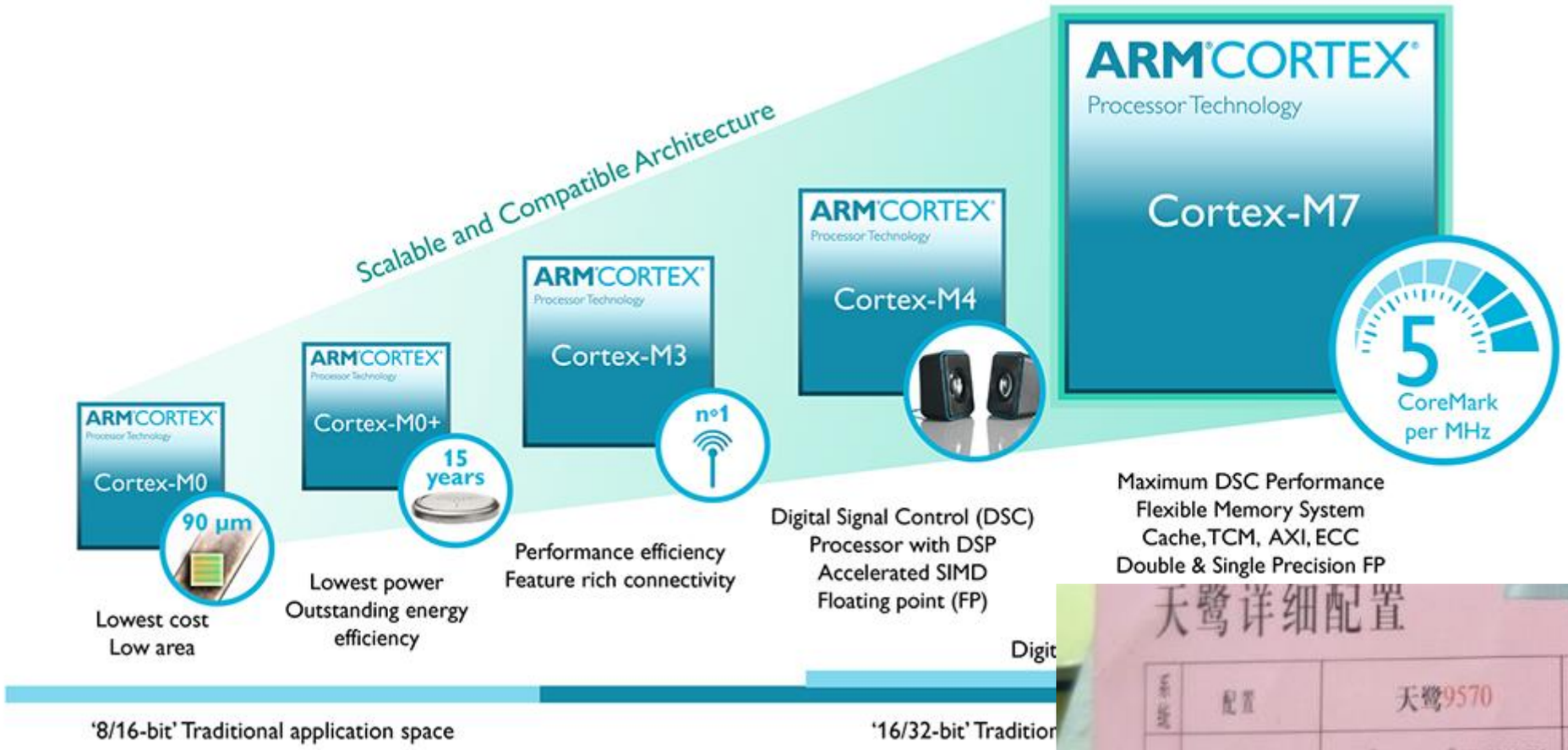
CPU硬件兼容统一，软件开发多样化

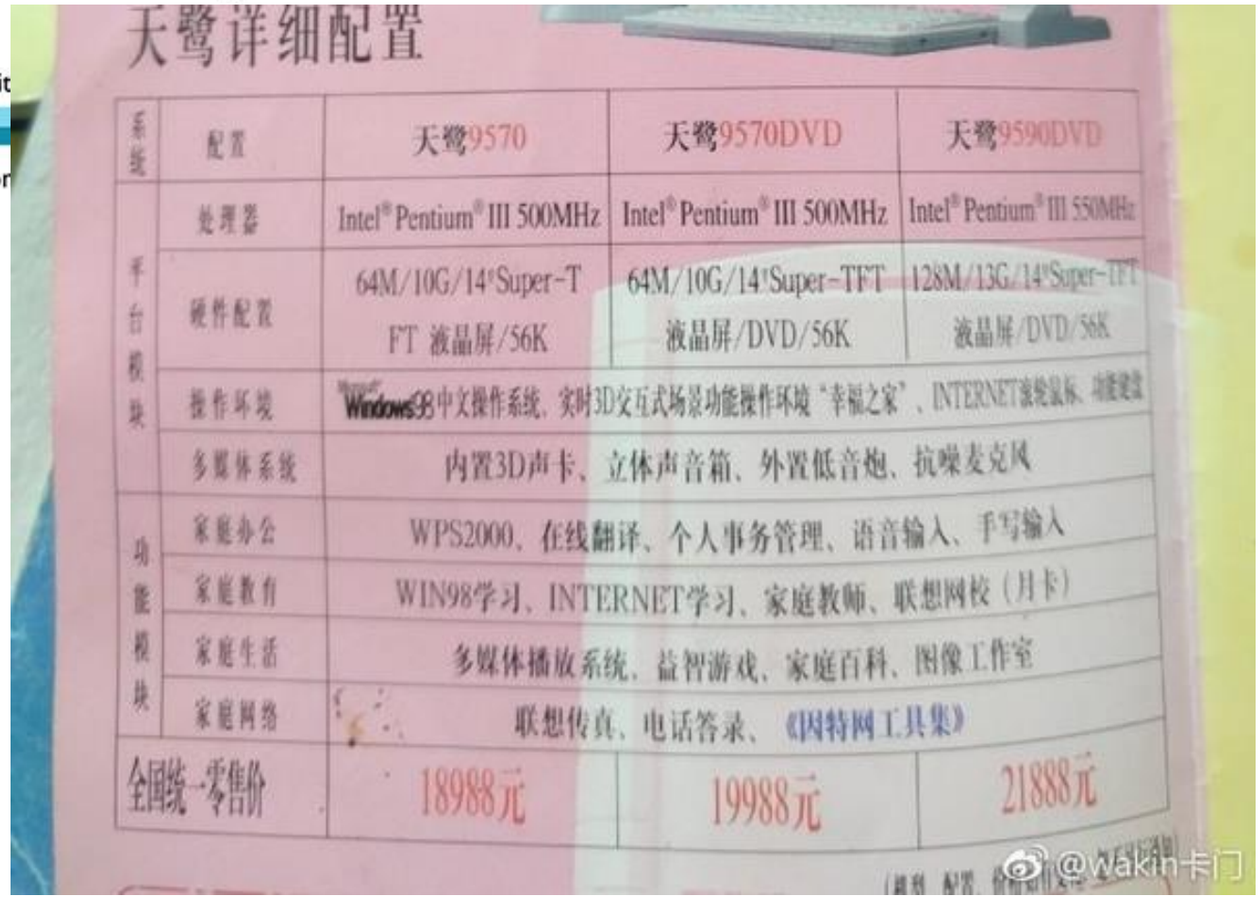
- 在80-90年代，x86架构逐渐占据了事实垄断地位，硬件大统一，市场开始爆炸式增长，硬件性能也持续地飞速提升
- 进入90年代后，为了满足人们的各种应用需求，软件开发方式变得百花齐放，软件工程/设计模式于这一时间段在理论和实践上得到了充分发展



为什么要将Swift用于MCU应用开发

MCU硬件开始逐渐兼容，性能开始飞速提升

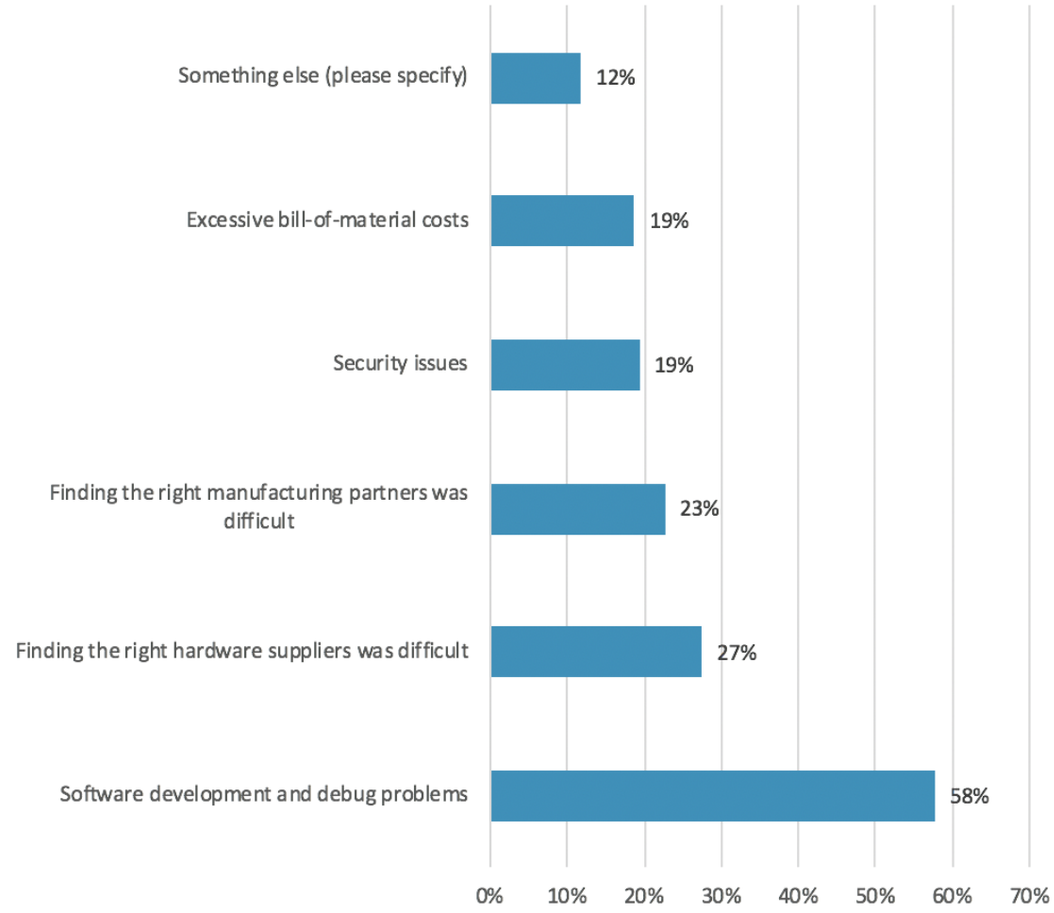




为什么要将Swift用于MCU应用开发

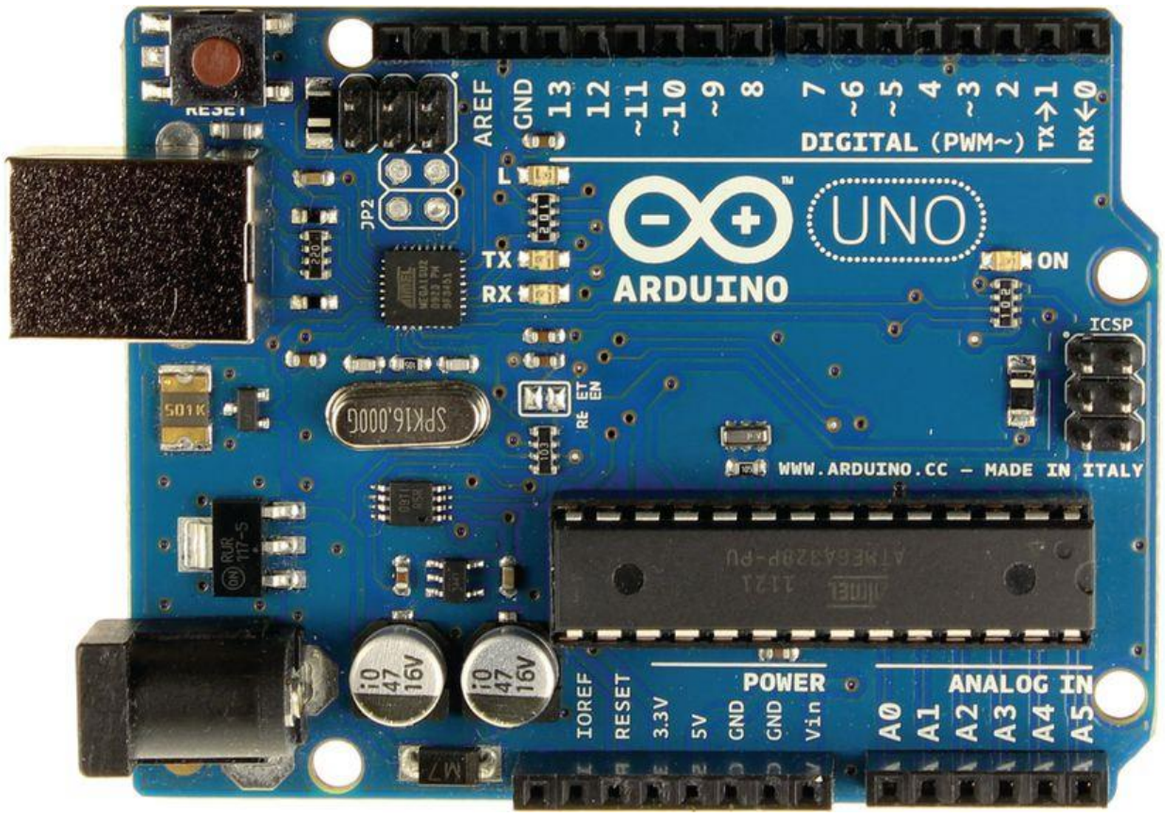
MCU硬件开始逐渐兼容，性能开始飞速提升

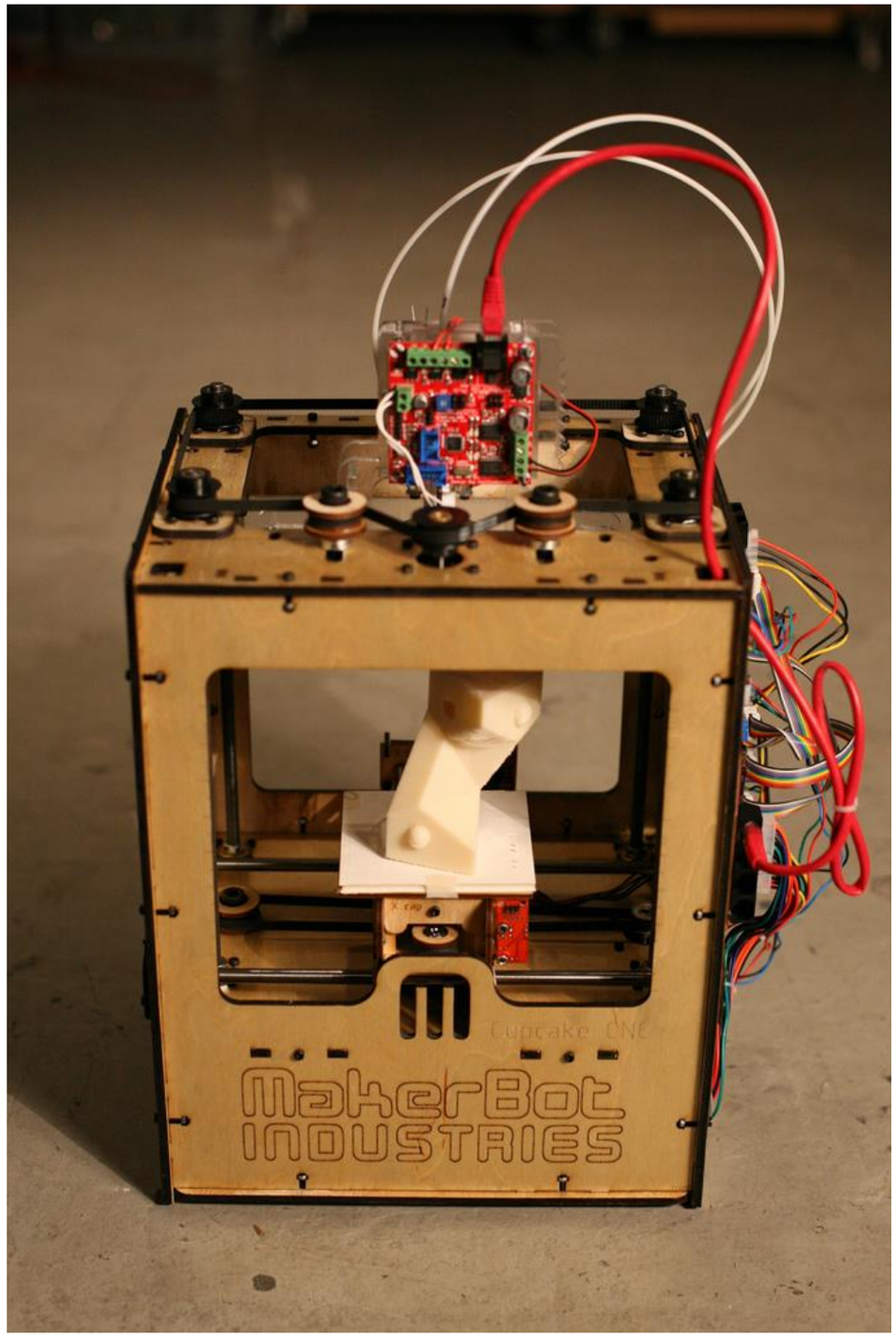
ARM公司于2018年的调查报告显示，58%的嵌入式开发者表示软件开发和调试是嵌入式开发过程中最头疼的事



为什么要将Swift用于MCU应用开发

简化MCU软件开发第一次成功尝试





为什么要将Swift用于MCU应用开发

各种不同的软件框架开始发力

ARDUINO

Zephyr™

arm MBED

RT-Thread
小而美的物联网操作系统

Espruino

AWorks™
ZLG Internet Of Things

为什么要将Swift用于MCU应用开发

为什么还要多一个Swift?

Dec 2019	Dec 2018	Change	Programming Language	Ratings	Change
1	1		Java	17.253%	+1.32%
2	2		C	16.086%	+1.80%
3	3		Python	10.308%	+1.93%
4	4		C++	6.196%	-1.37%
5	6	^	C#	4.801%	+1.35%
6	5	v	Visual Basic .NET	4.743%	-2.38%
7	7		JavaScript	2.090%	-0.97%
8	8		PHP	2.048%	-0.39%
9	9		SQL	1.843%	-0.34%
10	14	^^	Swift	1.490%	+0.27%

- 能保证实时性的同时又有着丰富现代特性的语言只有Swift和Rust
- 越容易上手的语言越容易吸引更多的应用开发者，比如Python
- 未来的嵌入式开发，必将重走CPU开发走过的路，越来越多样化

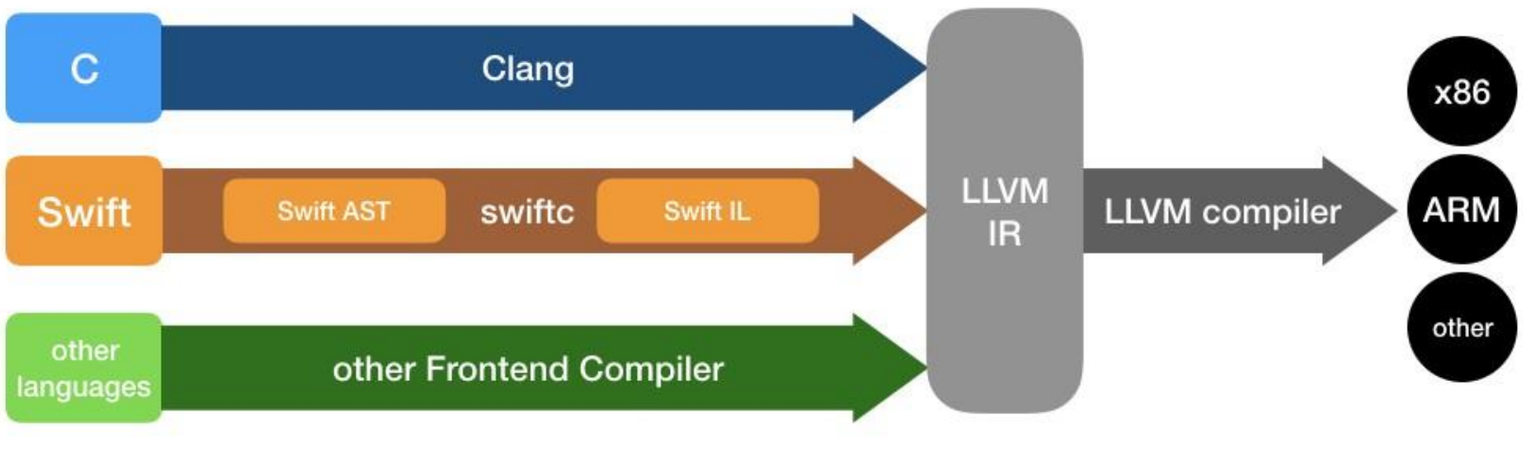
项目的基本技术原理

一句话解释编译器

- gcc
- iccarm: IAR用编译器
- armcc：KEIL用编译器

项目的基本技术原理

LLVM工具链架构

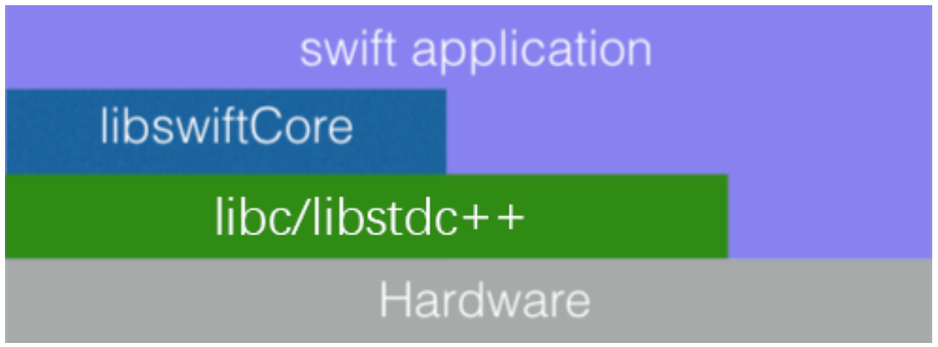


- 广义LLVM：泛指整个编译框架，比如基于GCC的编译链或者基于LLVM的编译链
- 狭义LLVM：特指将中间语言IR编译到不同平台机器码的编译器后端
- Clang：C，C++，Objective-C编译器前端，可将这三种语言编译为LLVM IR中间语言
- Swift：Swift编译器前端，可将Swift语言编译为LLVM IR中间代码

项目的基本技术原理

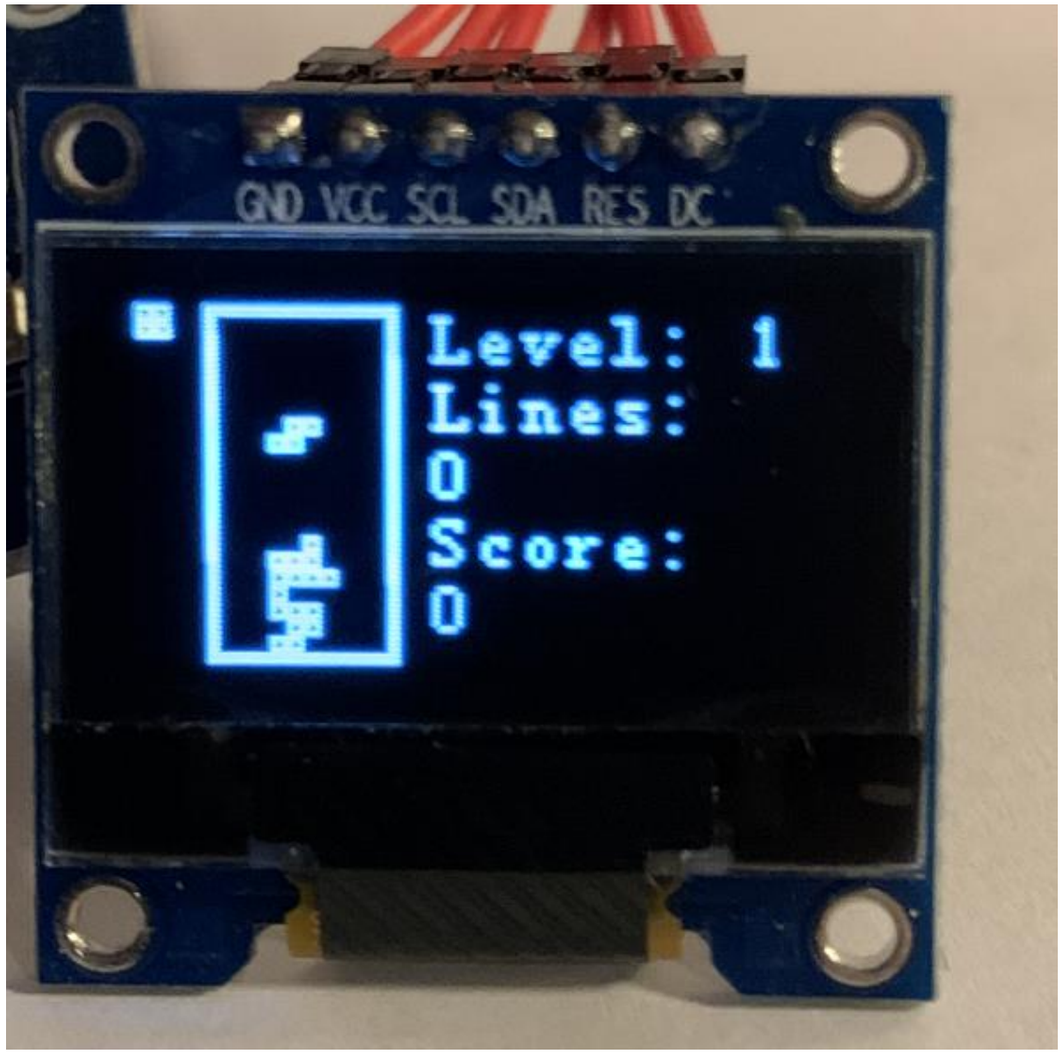
光得到机器码还不够，还得有底层库的支持

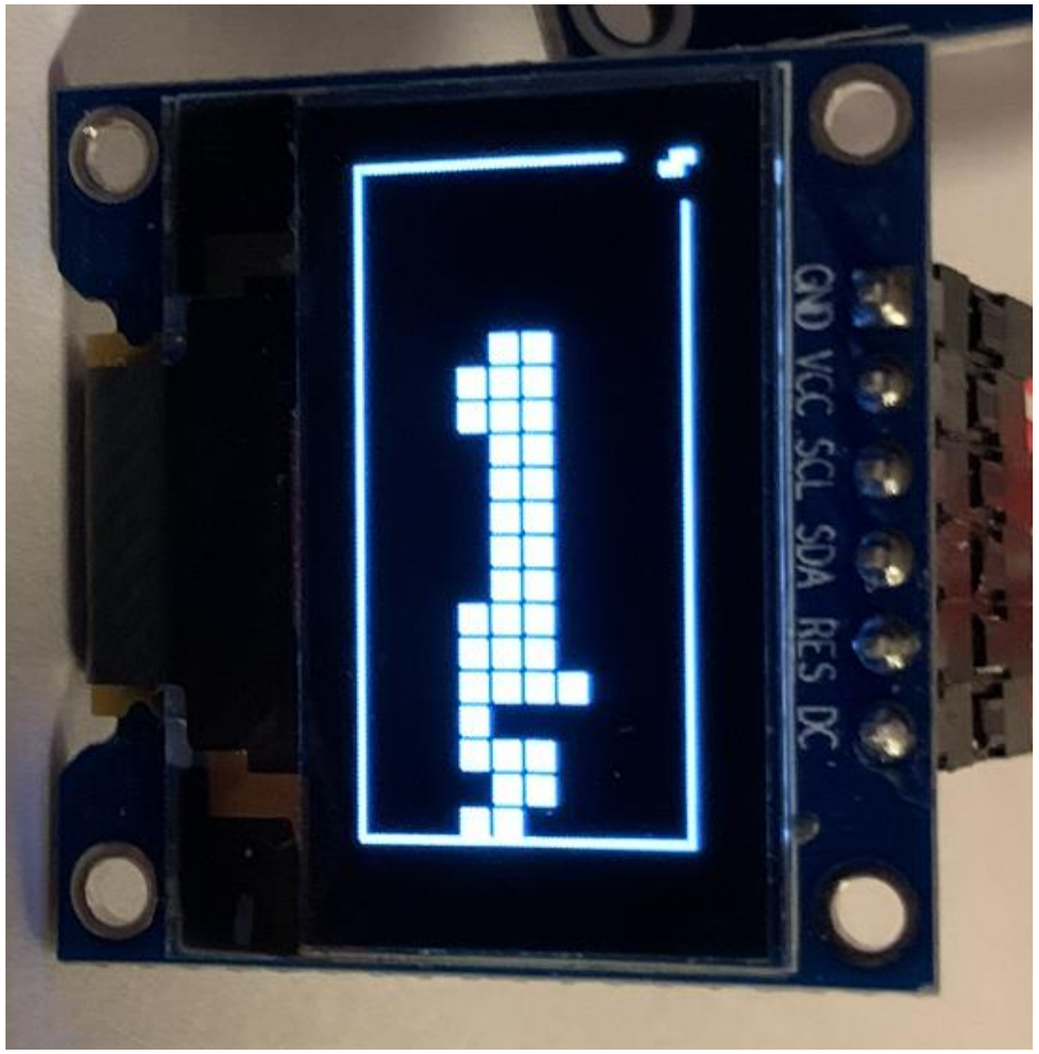
- **Swift标准库**：要涉及更为高级的功能，就不能避开Swift标准库，Swift标准库由Swift语言写成，内含各种数据类型，各种方便用户使用的常用函数
- **Swift Runtime**: 由C++写成，实现更为底层的操作，比如内存申请，保护，ARC的实现，函数的派发。就是这一部分代码实现了Swift的大部分安全特性。Swift标准库与Runtime经过链接后成为**libswiftCore**
- **libstdc++:** 为C++代码提供运行环境
- **libc**: 为C代码提供运行环境



案例分析

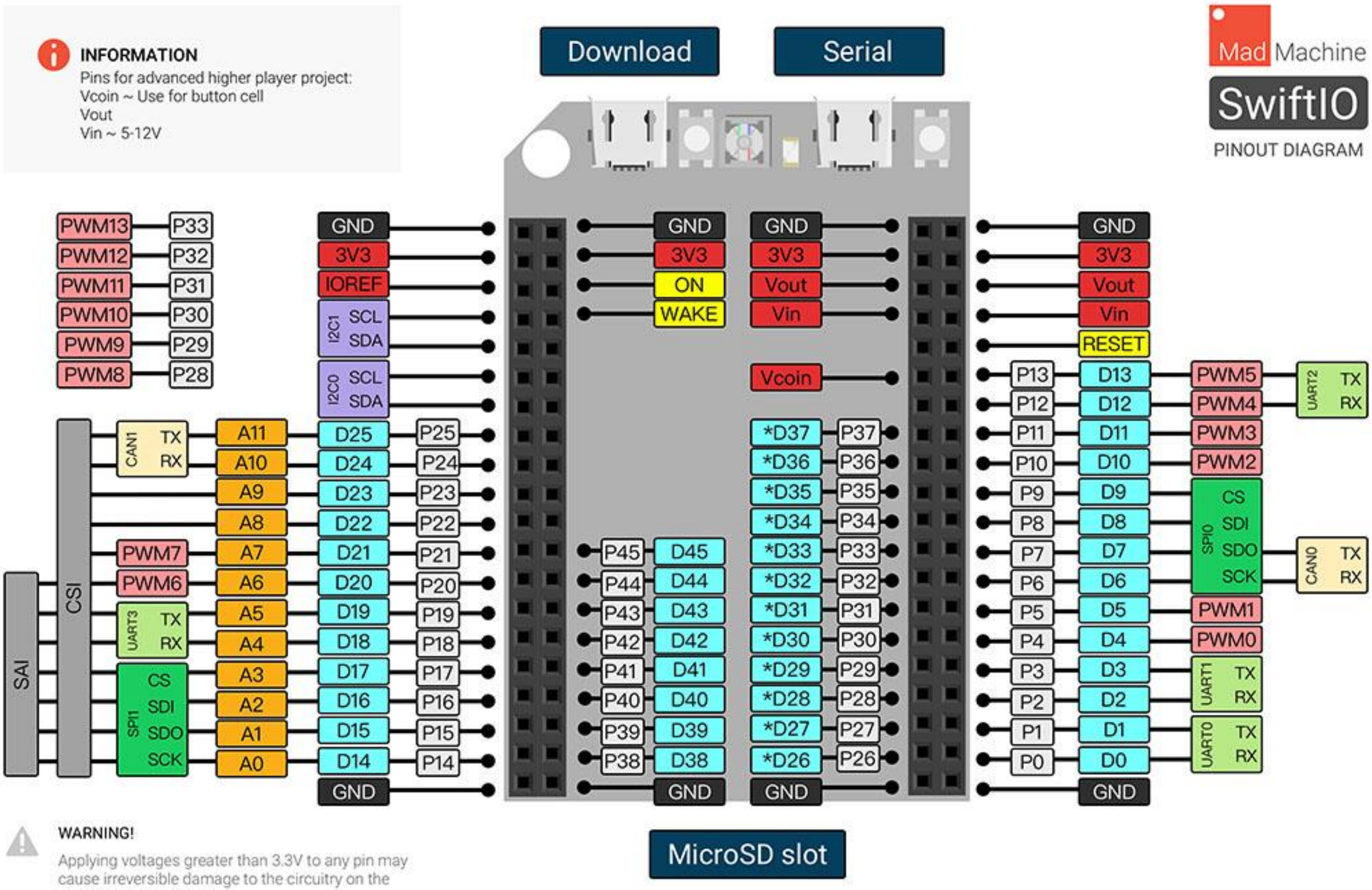
一个完整的俄罗斯方块游戏





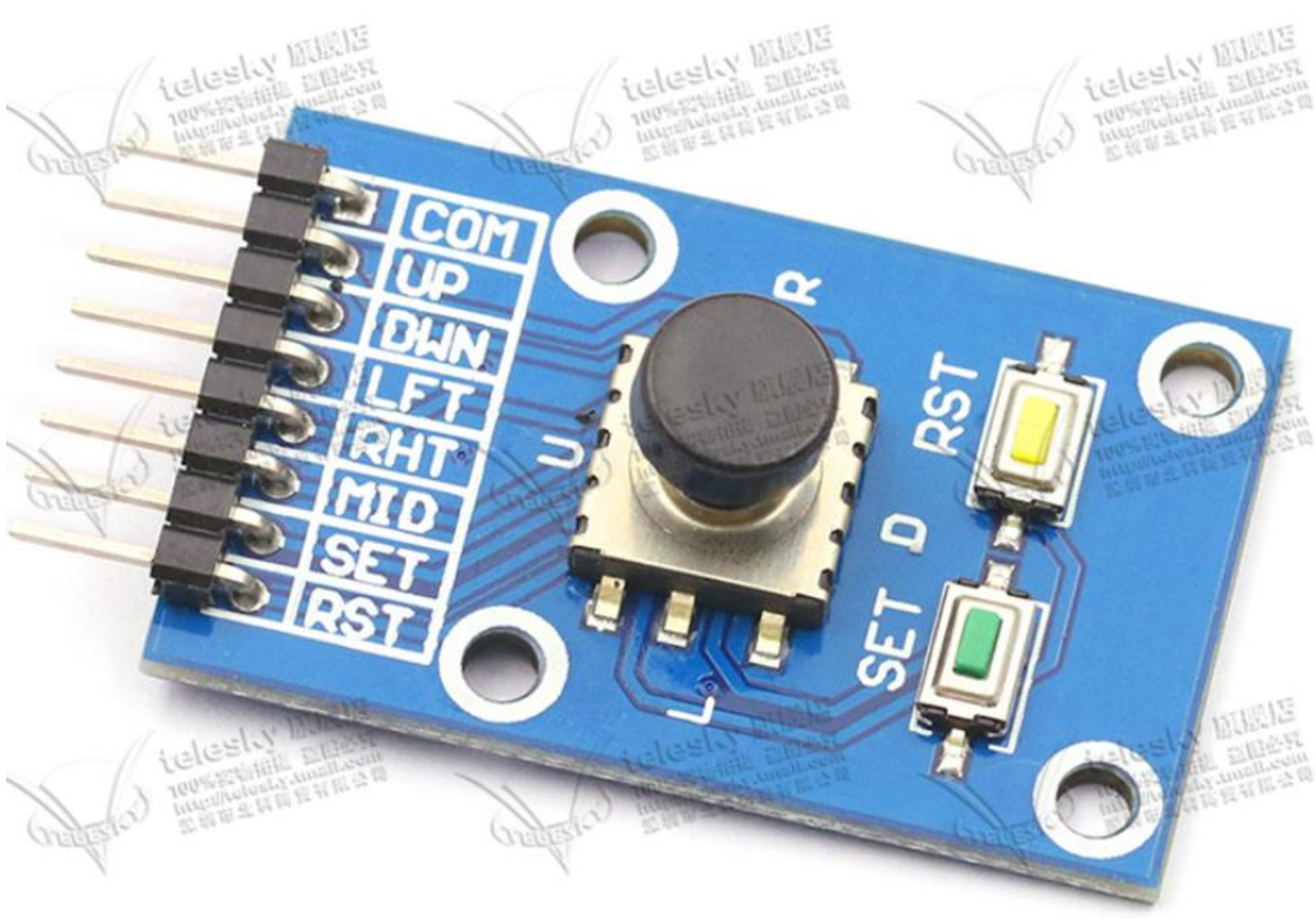
案例分析

核心板


INFORMATION
Pins for advanced higher player project:
Vcoin ~ Use for button cell
Vout
Vin ~ 5-12V
Download
Serial
Mad Machine
SwiftIO
PINOUT DIAGRAM
PWM13 P33
PWM12 P32
PWM11 P31
PWM10 P30
PWM9 P29
PWM8 P28
GND
3V3
IOREF
I2C1 SCL SDA
I2C0 SCL SDA
CAN1 TX RX
A11 D25 P25
A10 D24 P24
A9 D23 P23
A8 D22 P22
PWM7 A7 D21 P21
PWM6 A6 D20 P20
UART3 TX RX
A5 D19 P19
A4 D18 P18
SPI1 CS SDI SDO SCK
A3 D17 P17
A2 D16 P16
A1 D15 P15
A0 D14 P14
GND
SAI
CSI
GND
3V3
ON
WAKE
P45 D45
P44 D44
P43 D43
P42 D42
P41 D41
P40 D40
P39 D39
P38 D38
GND
GND
3V3
Vout
Vin
Vcoin
*D37 P37
*D36 P36
*D35 P35
*D34 P34
*D33 P33
*D32 P32
*D31 P31
*D30 P30
*D29 P29
*D28 P28
*D27 P27
*D26 P26
GND
GND
3V3
Vout
Vin
RESET
P13 D13 PWM5
P12 D12 PWM4
P11 D11 PWM3
P10 D10 PWM2
P9 D9
P8 D8
P7 D7
P6 D6
P5 D5 PWM1
P4 D4 PWM0
P3 D3
P2 D2
P1 D1
P0 D0
GND
SPI0 CS SDI SDO SCK
UART2 TX RX
CAN0 TX RX
UART1 TX RX
UART0 TX RX
WARNING!
Applying voltages greater than 3.3V to any pin may cause irreversible damage to the circuitry on the SwiftIO, except for Vin.
MicroSD slot

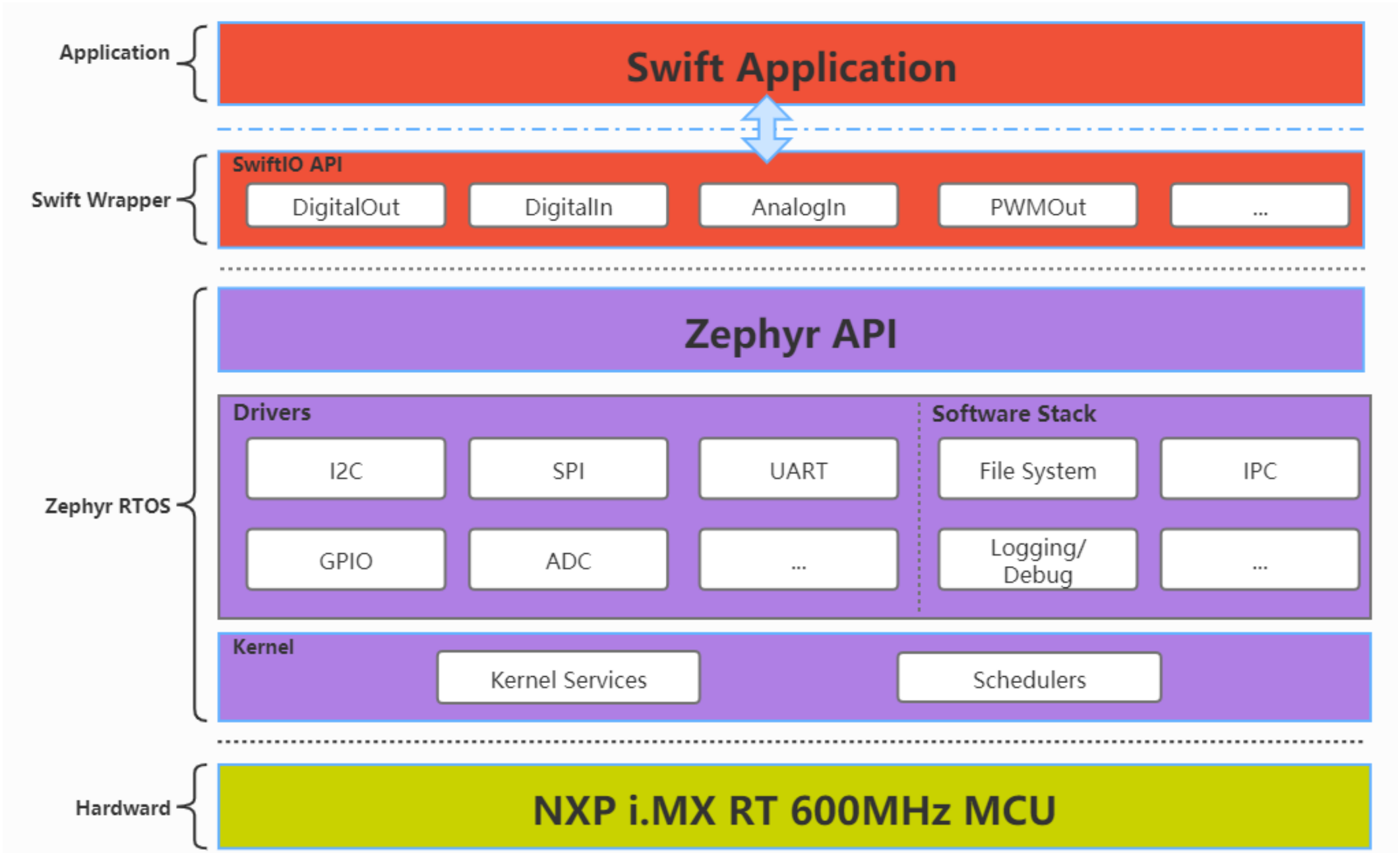
案例分析

OLED，KeyPad



案例分析

基于Zephyr的SwiftIO软件框架



案例分析

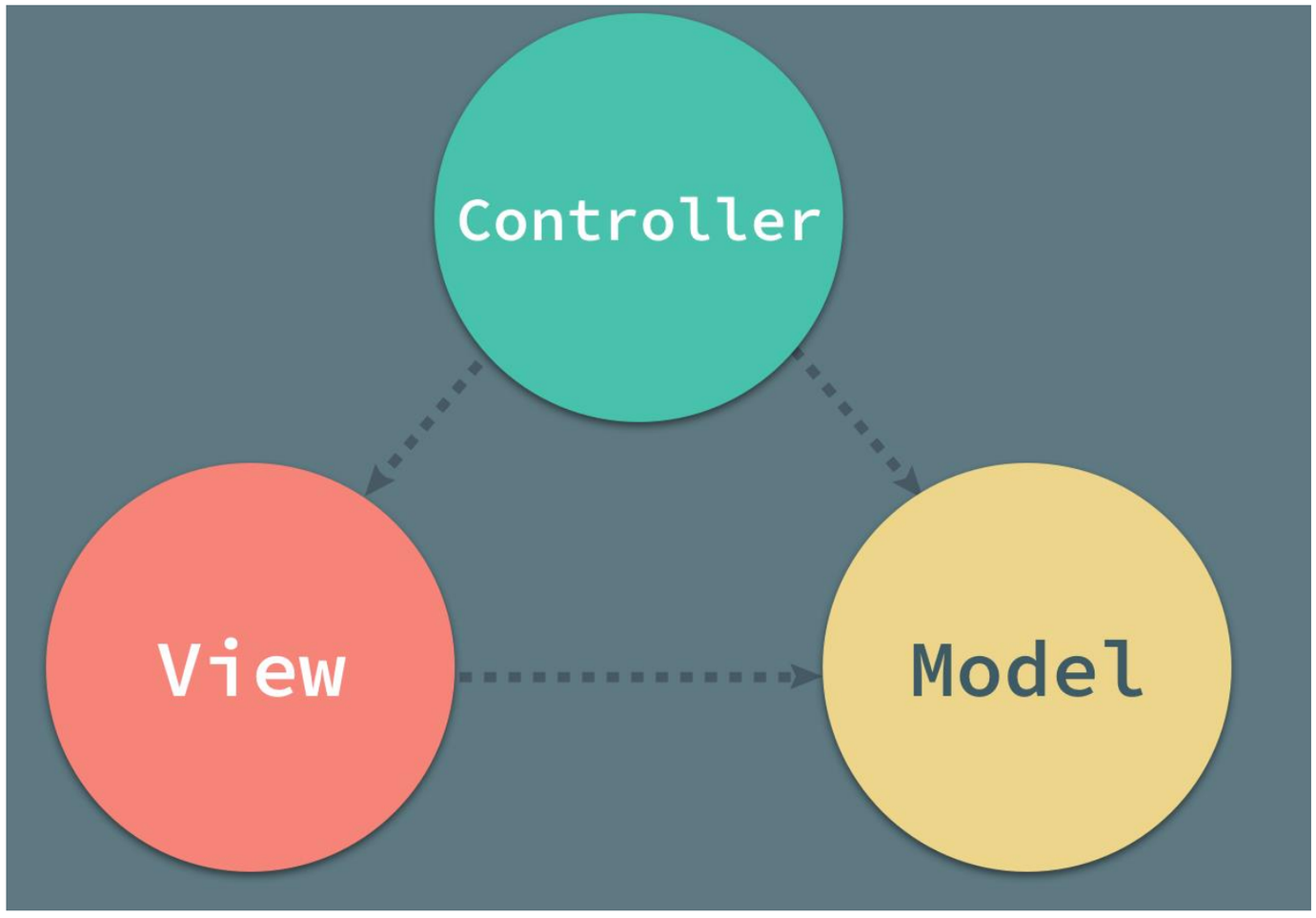
SwiftIO用法示例

```
import SwiftIO

func main() {
    //Create a DigitalOut to .D0
    let pin = DigitalOut(.D0)

    //Reverse the output value every 1 second
    while true {
        pin.reverse()
        sleep(1000)
    }
}
```

案例分析

游戏软件架构

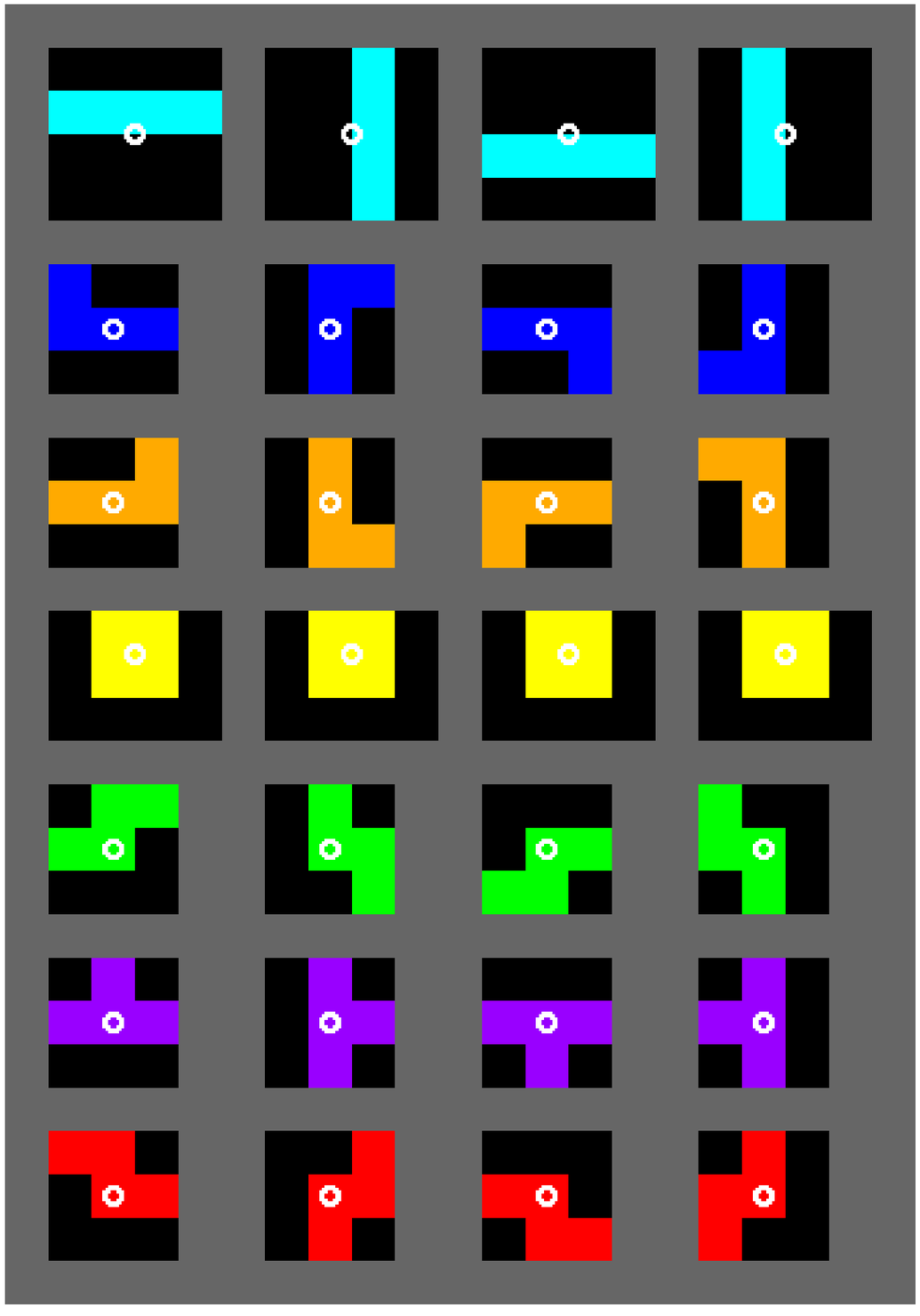
- **Model**： 游戏业务逻辑和数据
- **View**：屏幕显示
- **Controller**：胶合Model与View



案例分析

游戏软件架构

- **Block：** 7种block，每个block有4种造型
- **本质：**4x4的二维数组



案例分析

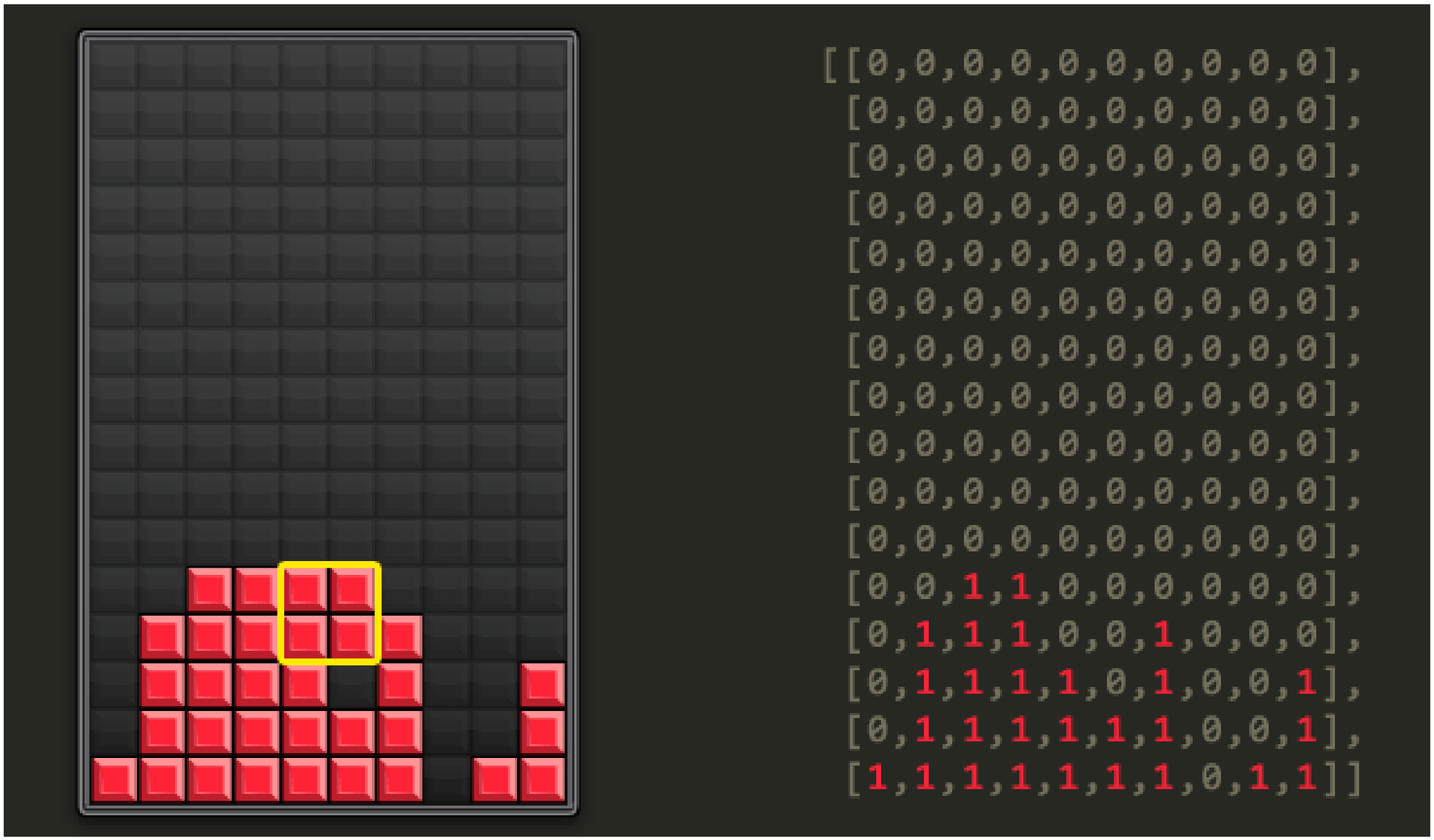
Block数据结构

- 记录block坐标及图片数据
- 对坐标进行操作

```
1   struct Block {
2       enum Name: Int {
3           case J = 1, L, Z, S, T, I, O
4       }
5
6       let name: Name
7       var x = 3, y = -2
8       var currentDirection = 0
9       var currentBlock = [[[Bool]]]()
10
11  >   init() {…
29      }
30
31  >   func getImage() -> [[Bool]] {…
33      }
34
35  >   func getRotateImage() -> [[Bool]] {…
44      }
45
46      mutating func rotate() {
47          let maxDirections = currentBlock.count
48          currentDirection += 1
49          if currentDirection == maxDirections {
50              currentDirection = 0
51          }
52      }
53
54      mutating func stepDown() {
55          y += 1
56      }
57
58      mutating func stepLeft() {
59          x -= 1
60      }
61
62      mutating func stepRight() {
63          x += 1
64      }
65  }
```

案例分析

游戏软件架构



- **Playground**：10x20的二维数组
- **游戏本质**：block数据结构与playground数据结构的交互

案例分析

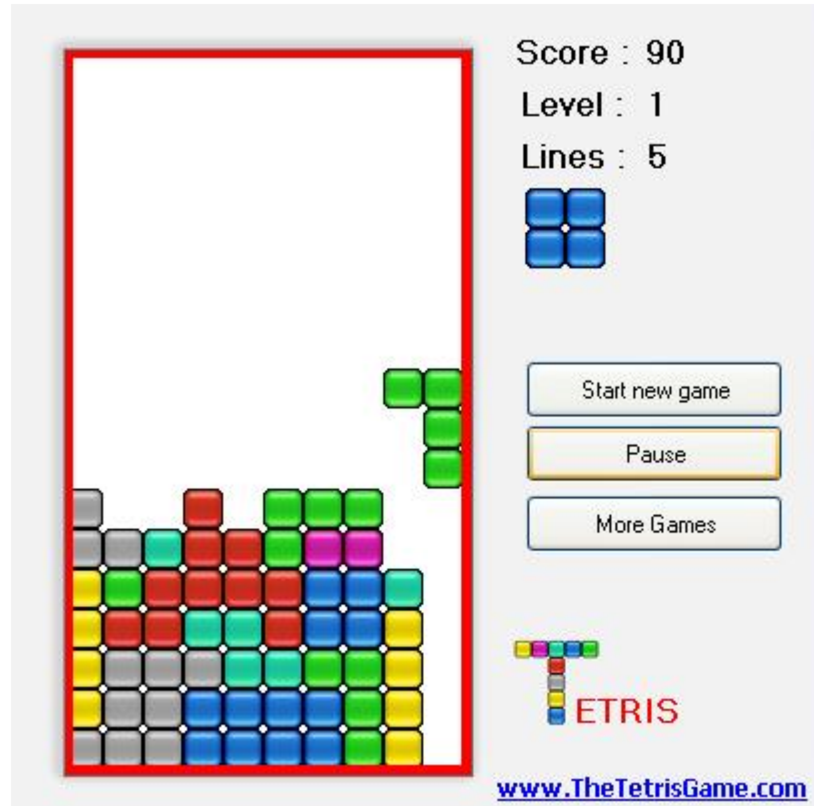
Playground数据结构

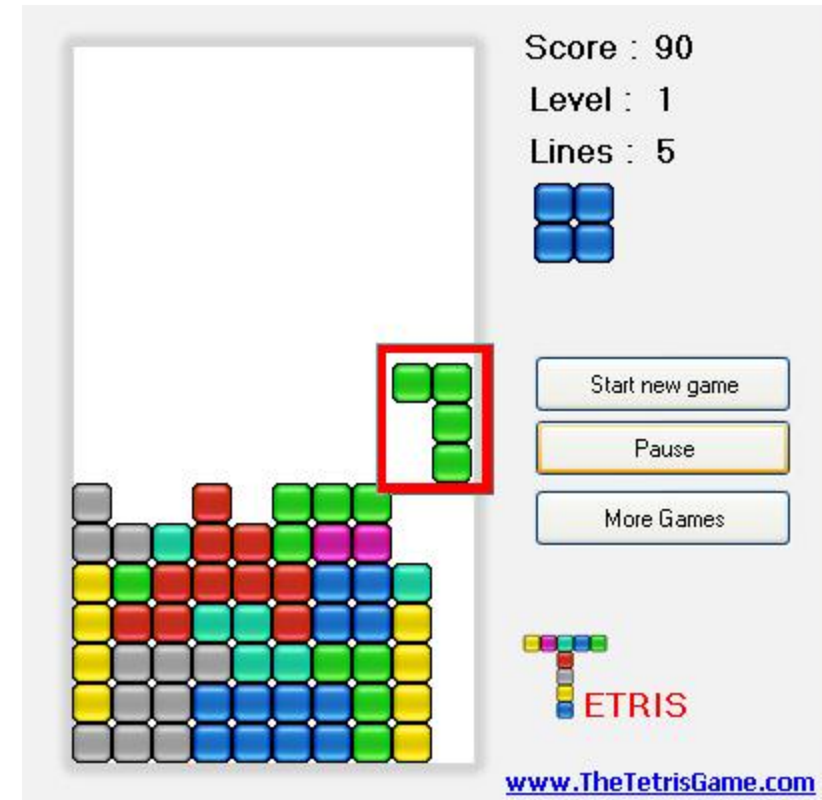
```
1   struct Playground {
2       enum MoveDirection {
3           case left, right, down
4       }
5
6       var screen = [[Bool]](repeating: [Bool](repeating: false, count: 10), count: 20)
7
8       func getScreen() -> [[Bool]] {
9           return screen
10      }
11
12  >   mutating func tryMove(_ block: inout Block, to dir: MoveDirection) -> Bool {…
25      }
26
27  >   mutating func tryRotate(_ block: inout Block) -> Bool {…
81      }
82
83  >   func isGameOver(_ block: Block) -> Bool {…
88      }
89
90  >   mutating func merge(_ block: Block) {…
109     }
110
111 >   func checkFullLines() -> [Int] {…
125     }
126
127 >   mutating func clearLine(at lines: [Int]) {…
134     }
135 }
136
```

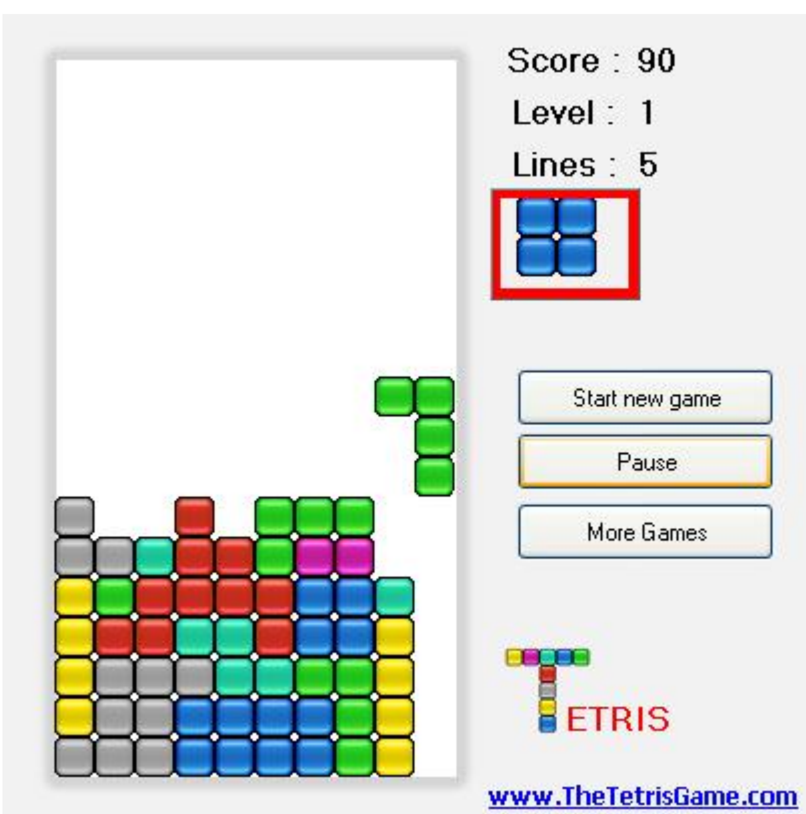
案例分析

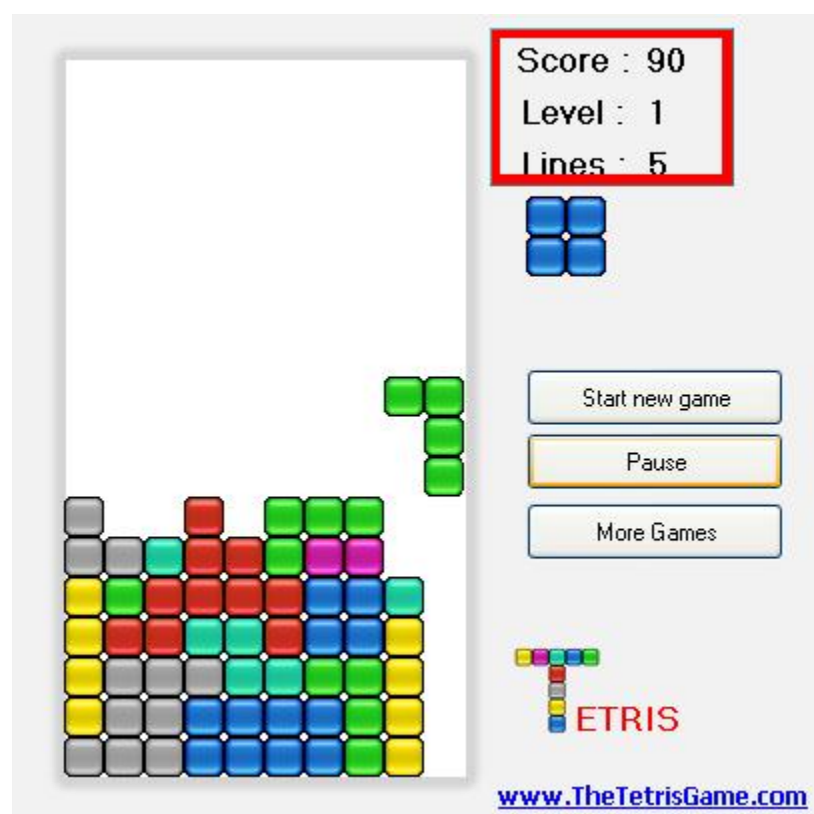
View层需要显示的内容

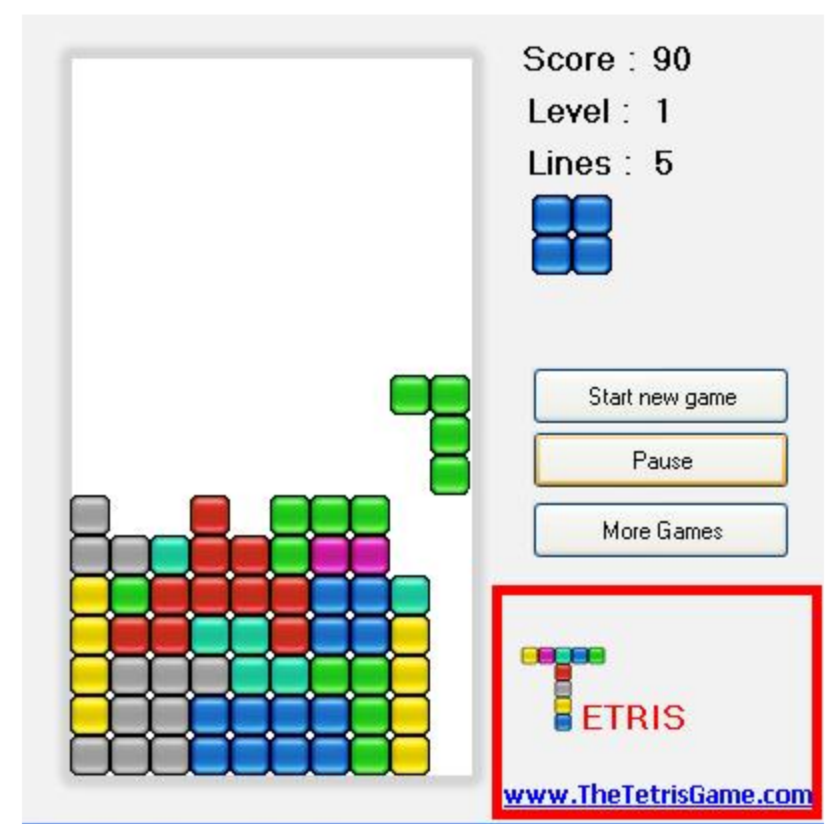
- **Playground**
- **Block**
- **NextBlock**
- **GameData**
- **数据无关的图像，字符**











案例分析

只要实现了TetrisView协议，就可以显示游戏画面

```
4   protocol TetrisView {
5       func updatePlayground(_ playground: [[Bool]])
6       func updateBlock(_ x: Int, _ y: Int, image: [[Bool]])
7       func updateNextImage(_ image: [[Bool]])
8       func updateData(score: Int, lines: Int, level: Int)
9       func displayInit()
10      func display()
11  }
```

案例分析

完整的Controller代码

```
62      var timer = 0
63      var viewDelegate: TetrisView
64
65      while true {
66          timer += 10
67          sleep(ms: 10)
68
69          let keyValue = keypad.getKeyState()
70
71          if keyValue == .rotate {
72              playground.tryRotate(&block)
73          }
74
75          if keyValue == .left {
76              playground.tryMove(&block, to: .left)
77          }
78
79          if keyValue == .right {
80              playground.tryMove(&block, to: .right)
81          }
82
83  >       if timer % gameData.timerDuration == 0 || keyValue == .down {…
98          }
99
100         viewDelegate.updatePlayground(playground.getScreen())
101         viewDelegate.updateBlock(block.x, block.y, image: block.getImage())
102         viewDelegate.updateData(score: gameData.score, lines: gameData.lines, level: gameData.level)
103
104         if timer % 50 == 0 {
105             viewDelegate.display()
106         }
107
```

游戏进程控制

显示控制

屏幕硬件刷新控制

案例分析

完整的Controller代码

```
83          if timer % gameData.timerDuration == 0 || keyValue == .down {
84              if playground.tryMove(&block, to: .down) {
85                  // 下落成功
86              } else if playground.isGameOver(block) {
87                  // 游戏结束
88              } else {
89                  // 下落失败
90                  playground.merge(block)                                  // 合并block到playground
91                  let fullLines = playground.checkFullLines()              // 检查满行
92                  if fullLines.count != 0 {
93                      gameData.update(fullLines)                           // 更新游戏得分
94                      playground.clearLine(at: fullLines)                  // 清除满行
95                  }
96                  block = nextBlock                                        // 修改当前block
97                  nextBlock = Block()                                      // 重新生成下一个block
98                  viewDelegate.updateNextImage(nextBlock.getImage())       // 更新view中下一个block图片
99              }
100         }
```

案例分析

完整的Controller代码

```
62      var timer = 0
63      var viewDelegate: TetrisView
64
65      while true {
66          timer += 10
67          sleep(ms: 10)
68
69          let keyValue = keypad.getKeyState()
70
71          if keyValue == .rotate {
72              playground.tryRotate(&block)
73          }
74
75          if keyValue == .left {
76              playground.tryMove(&block, to: .left)
77          }
78
79          if keyValue == .right {
80              playground.tryMove(&block, to: .right)
81          }
82
83  >       if timer % gameData.timerDuration == 0 || keyValue == .down {…
98          }
99
100         viewDelegate.updatePlayground(playground.getScreen())
101         viewDelegate.updateBlock(block.x, block.y, image: block.getImage())
102         viewDelegate.updateData(score: gameData.score, lines: gameData.lines, level: gameData.level)
103
104         if timer % 50 == 0 {
105             viewDelegate.display()
106         }
107
```

游戏进程控制

显示控制

屏幕硬件刷新控制

案例分析

横屏竖屏的切换

```
108         if keyValue == .select {
109             if viewDelegate is HView {
110                 viewDelegate = vView
111             } else {
112                 viewDelegate = hView
113             }
114             viewDelegate.displayInit()
115             viewDelegate.updatePlayground(playground.getScreen())
116             viewDelegate.updateBlock(block.x, block.y, image: block.getImage())
117             viewDelegate.updateNextImage(nextBlock.getImage())
118         }
119     }
```