编程语言

Programming Languages

井明

计算机科学与技术学院

Fall 2021

目录

Programming Language Generations

- 1GL Machine-Level Programming Language
- 2GL Assembly Languages
- 3GL High-Level Computer Programming Language
- 4GL Domain-Specific Language
- 5GL Constraint-Based and Logic Programming Languages, Declarative Languages

编程语言发展史

1GL 机器级别编程语言

2GL 汇编语言

3GL 高级计算机编程语言

4GL 特定领域语言

5GL 基于约束和逻辑的编程语言，声明式语言

1GL: Machine-Level Programming Language

1. Instructions are made by Binary Numbers (1s and 0s)
2. No Translator is used to Compile or Assemble
3. Suitable for the understanding of the machine, but difficult to interpret and learn by the human programmer.
4. the code can run very fast and very efficiently, precisely because the instructions are executed directly by the central processing unit (CPU)
5. disadvantages : the code is not as easy to fix

1GL: 机器级别编程语言

1. 指令, 二进制 (1 或 0)
2. 不需要编译或汇编
3. 面向机器, 程序员难懂
4. CPU 直接执行, 高效
5. 难排错

机器代码

- 由指令组成. 指令控制 CPU, e.g. load, store, jump, ALU op on registers or memory.
- 指令集. RISC vs. CISC.

Bootloader

Magic Numbers at the end of the first 512 bytes.

```
0x55AA
0101 0101 1010 1010
```

MIPS 架构 (RISC 32 位)

```
6       5       5       5       5       6 bits
[  op   |  rs  |  rt  |  rd  |shamt| funct]   R-type
[  op   |  rs  |  rt  | address/immediate]   I-type
[  op   |          target address        ]   J-type
```

R-type (register)

R1 和 R2 求和保存到 R6.

```
[  op   |  rs  |  rt  |  rd  |shamt| funct]
    0       1       2       6       0       32      decimal
 000000  00001  00010  00110  00000  100000    binary
```

MIPS 架构 (RISC 32 位)(续)

I-type (immediate)

将 R3 地址值后 68 个内存数据加载到 R8.

```
[  op   |  rs |  rt | address/immediate]
   35      3     8            68          decimal
 100011 00011 01000 00000 00001 000100    binary
```

J-type (jump)

跳转到地址 1024.

```
[  op   |          target address         ]
    2                  1024                decimal
 000010 00000 00000 00000 10000 000000    binary
```

2GL: 汇编语言

```
1 .code16
2 .global init # makes our label "init" available to the outside
3
4 init: # this is the beginning of our binary later.
5   mov $0x0e, %ah # sets AH to 0xe (function teletype)
6   mov $msg, %bx   # sets BX to the address of the first byte of our message
7   mov (%bx), %al   # sets AL to the first byte of our message
8   int $0x10 # call the function in ah from interrupt 0x10
9   hlt # stops executing
10
11 msg: .asciz "Hello␣world!"  # stores the string (plus a byte with value "0")
12                             # and gives us access via $msg
13
14 .fill 510-(.-init), 1, 0 # add zeroes to make it 510 bytes long
15 .word 0xaa55 # magic bytes that tell BIOS that this is bootable
```

3GL: 高级编程语言

特点

- 独立于机器.
- 可读性高, 对程序员友好.

分类

- 结构式编程语言: C
- 面向对象编程语言: Java, C++, C#, Go, PHP

4GL: 特定领域编程语言

- 通用语言. Unix Shell, Visual FoxPro, PowerBuilder
- 数据库查询语言. FOCUS, SQL
- 报告语言.
- 数据操作分析和报表生成语言. R, SPSS, PL/SQL, XSLT
- 软件创作. LiveCode
- 数学优化. MATLAB
- 数据库驱动的可视化应用开发.
- 低代码或无代码开发发框架
- 屏幕打印和生成
- Web 开发语言. ActiveVFP

5GL: CP, LP, DL

解决问题的方式侧重于基于程序的约束, 而非由程序员写的算法代码.

例: 经典字母解谜. SEND+MORE=MONEY , Prolog 代码.

```
% This code works in both YAP and SWI-Prolog using the environment-supplied
% CLPFD constraint solver library.  It may require minor modifications to work
% in other Prolog environments or using other constraint solvers.
:- use_module(library(clpfd)).
sendmore(Digits) :-
    Digits = [S,E,N,D,M,O,R,N,Y],    % Create variables
    Digits ins 0..9,                 % Associate domains to variables
    S #\= 0,                         % Constraint: S must be different from 0
    M #\= 0,
    all_different(Digits),           % all the elements must take different values
                  1000*S + 100*E + 10*N + D     % Other constraints
                + 1000*M + 100*O + 10*R + E
    #= 10000*M + 1000*O + 100*N + 10*E + Y,
    label(Digits).                   % Start the search
```

代码分类

- 源代码 (source code)
- 目标代码 (object code). 编译器的输出.
- 字节代码 (bytecode). 由解析器执行的指令集. JVM.
- 机器代码 (machine code)
- 微代码 (microcode). 集成到处理器中的硬件级别的指令, 实现机器代码.

编译策略 (Compilation strategies)

- 即时编译 (just-in-time compilation)(JIT), 也称为动态翻译. 程序运行时翻译. Java(bytecode to machine code), .NET, 正则表达式.
- 提前编译 (ahead-of-time compilation)(AOT). C, C++. [1]
- 源到源编译器 (source to source compiler)(Transcompilation). e.g. Pascal to C, Typescript to Javascript.
- 动态重编译 (Recompilation). Rosetta 2(MacOS), x86 to ARM.

[1]静态编译, 侧重于强调性能.

罗塞塔石碑

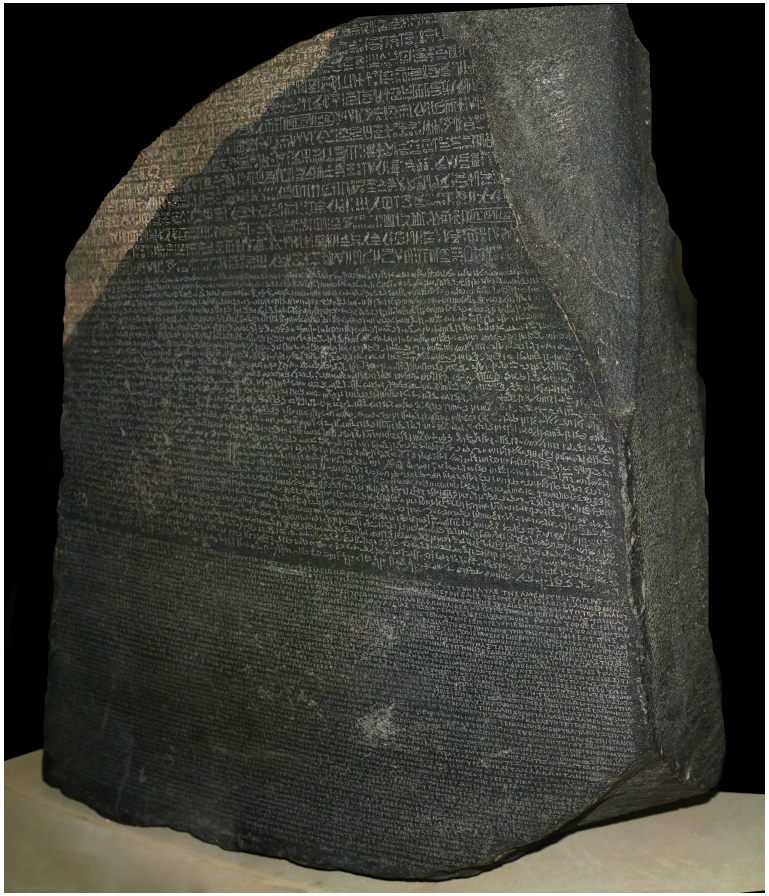


Figure: 罗塞塔石碑: 圣书体, 世俗体, 古希腊文

Apple M1 Chip (ARM)

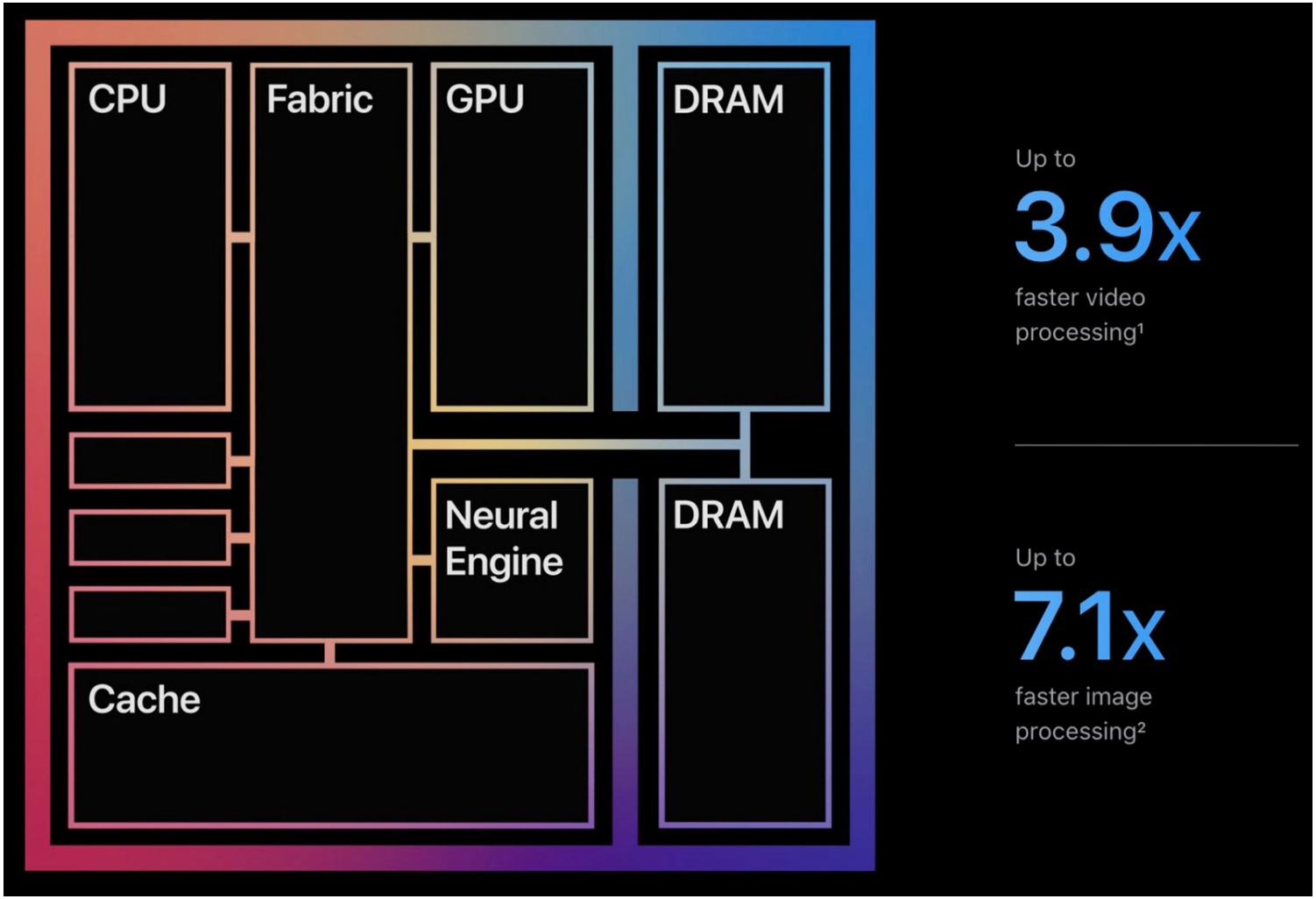


Figure: M1 芯片统一内存 (Unified Memory) 架构

运行环境

- Android Runtime（ART）.
  - Dalvik 采用的是 JIT 技术
  - ART 采用 Ahead-of-time（AOT）技术
- 通用语言运行库（CLR）. Microsoft .NET 虚拟机.
- crt0 是连接到 C 程序上的一组执行启动例程.
- Java 虚拟机（JVM）
- V8 (Node.js)(Chromium). JavaScript to Machine Code.
- PyPy
  - PyPy 是一种即时编译器
  - CPython 是一种解释器
  - PyPy 运行得比 CPython 更快
- Zend 引擎.(开源脚本引擎)(虚拟机) PHP

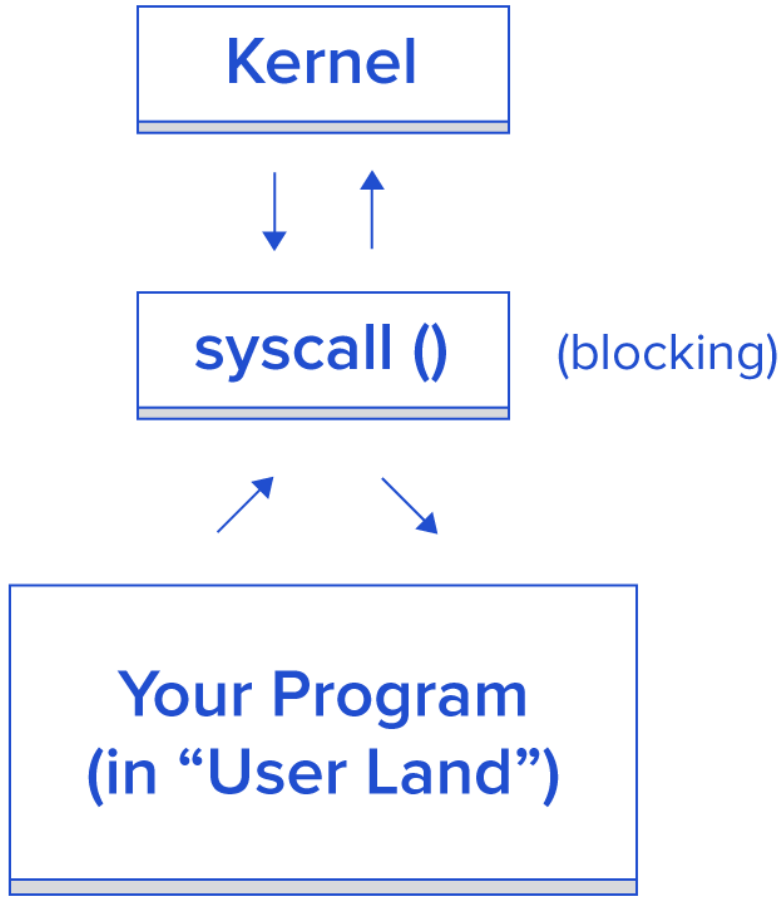
TIOBE 排名 (Aug 2021)

Programming Language	Ratings
C	12.57%
Python	11.86%
Java	10.43%
C++	7.36%
C#	5.14%
Visual Basic	4.67%
JavaScript	2.95%
PHP	2.19%
Assembly language	2.03%
SQL	1.47%
SQL	1.47%
Groovy	1.36%
Classic Visual Basic	1.23%
Fortran	1.14%
R	1.05%
Ruby	1.01%
Swift	0.98%
MATLAB	0.98%
Go	0.90%
Prolog	0.80%
Perl	0.78%

Figure: TIOBE Programming Community Index

Syscalls

1. 用户程序必须通过请求操作系统内核实现 I/O 操作
2. 系统调用 ("syscall") 将控制权由用户程序转给系统内核. Syscalls 是阻塞的. (用户程序需要等待内核返回结果.)
3. 系统内核在物理设备上进行 I/O 操作, 然后返回给 syscall.



Blocking vs. Non-blocking Calls

- blocking I/O: 从 IO 设备等待读取新的数据
- non-blocking I/O: 当 IO 设备有新的数据, 回调函数

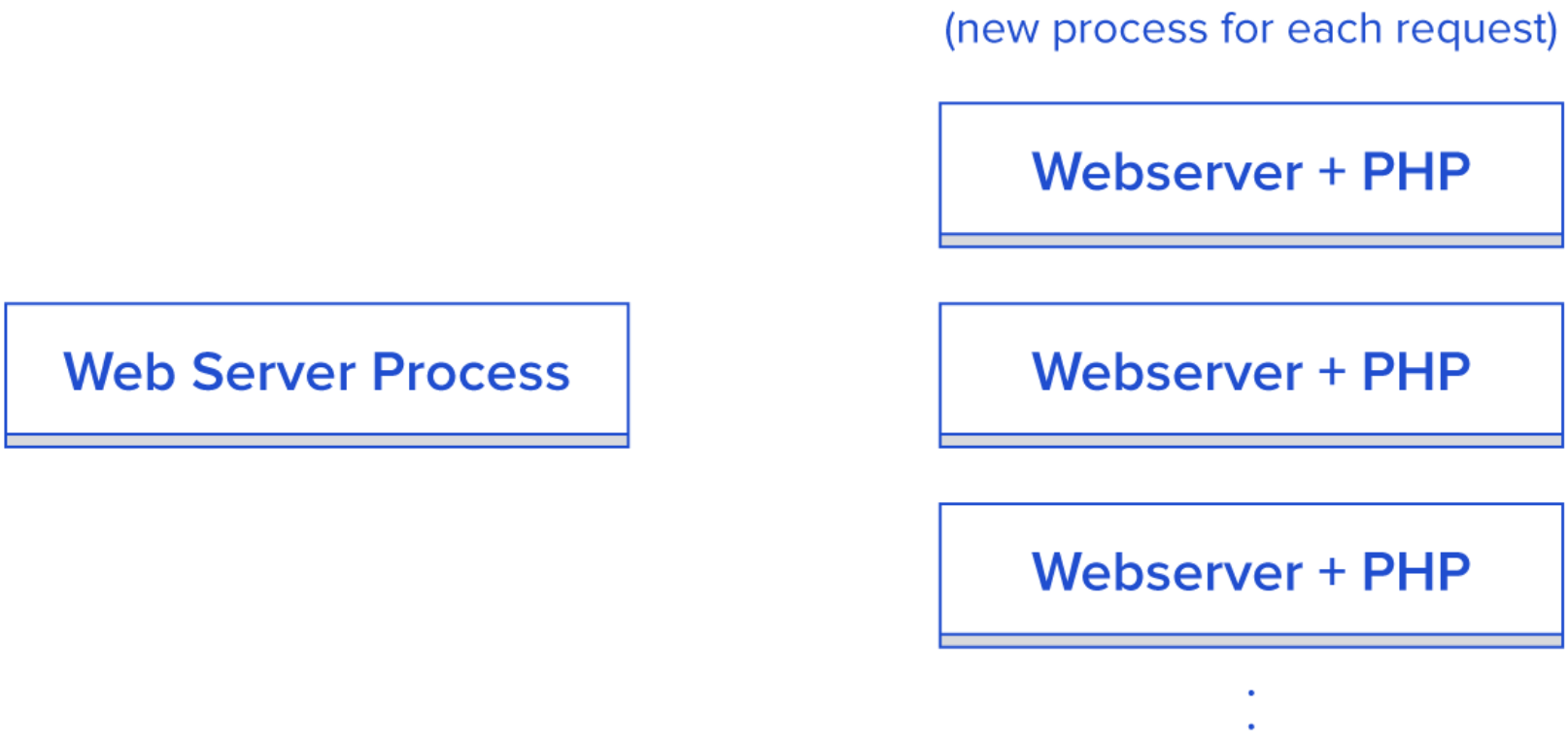
Scheduling

- Process
- Thread

PHP: Blocking I/O Model

```php
<?php

// blocking file I/O
$file_data = file_get_contents( '/path/to/file.dat' );

// blocking network I/O
$curl = curl_init('http://example.com/example-microservice');
$result = curl_exec($curl);

// some more blocking network I/O
$result = $db->query('SELECT id, data FROM examples
    ORDER BY id DESC limit 100');

?>
```

PHP: new process for each request



Java: multiple thread approach

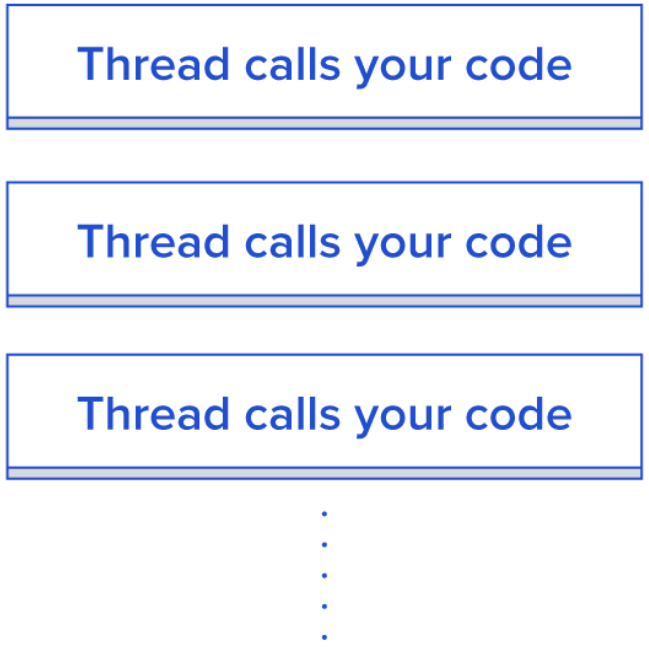
```java
public void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException
{
    // blocking file I/O
    InputStream fileIs = new FileInputStream("/path/to/file");

    // blocking network I/O
    URLConnection urlConnection = (new URL(
        "http://example.com/example-microservice"))
        .openConnection();
    InputStream netIs = urlConnection.getInputStream();

    // some more blocking network I/O
    out.println("...");
}
```

Java I/O Model

Java Server Process

(Non-blocking I/O available in the language but not used by most web servers.)

(new thread for each request)

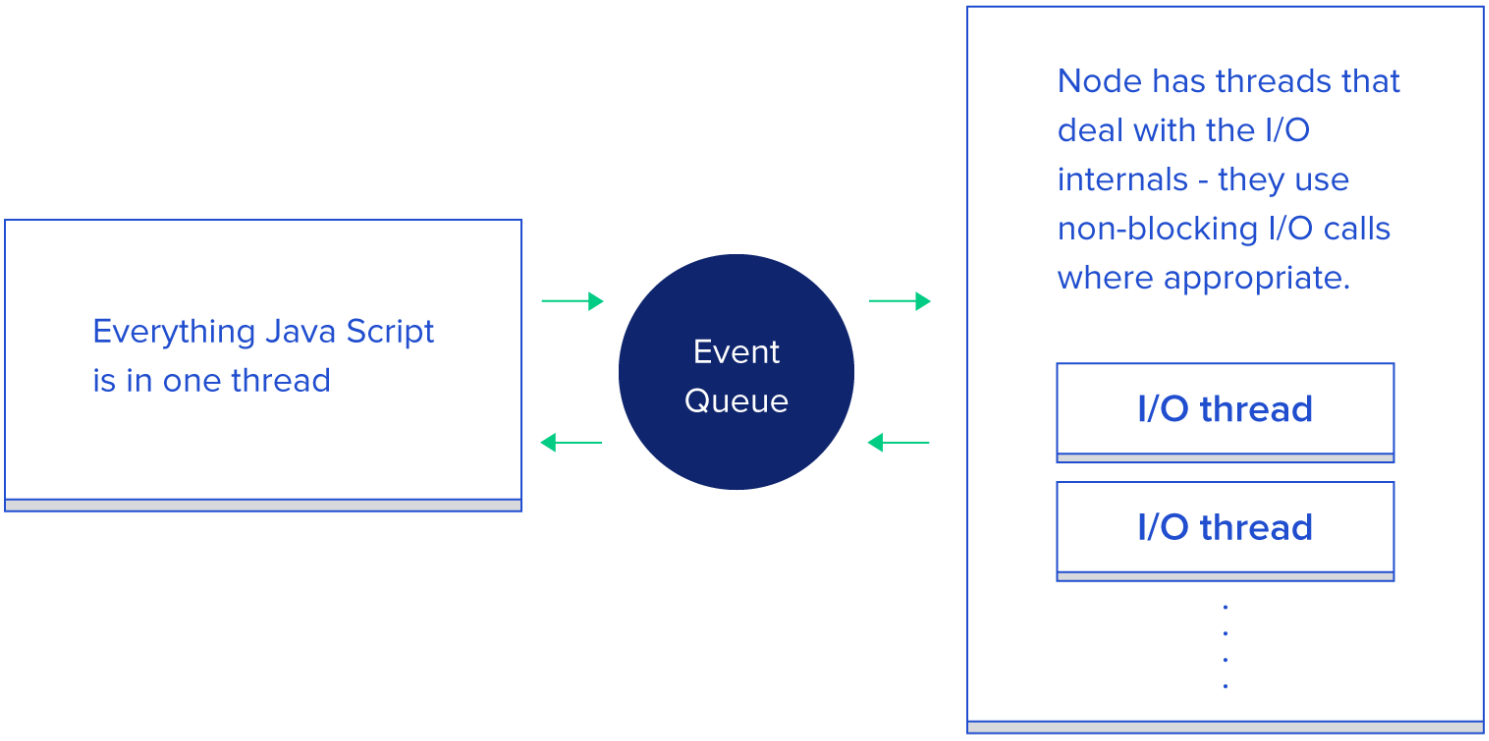


Node.js: Non-blocking I/O

```
1 http.createServer(
2     function(request, response) {
3         fs.readFile('/path/to/file', 'utf8',
4             function(err, data) {
5                 response.send(data);
6             }
7         );
8     }
9 );
```

Node.js I/O Model

I/O Model Node



Go: naturally non-blocking

```
1 func ServeHTTP(w http.ResponseWriter, r *http.Request) {
2
3     // the underlying network call here is non-blocking
4     rows, err := db.Query("SELECT ...")
5
6     for _, row := range rows {
7         // do something with the rows,
8         // each request in its own goroutine
9     }
10
11    w.Write(...) // write the response, also non-blocking
12 }
```

Go I/O Model

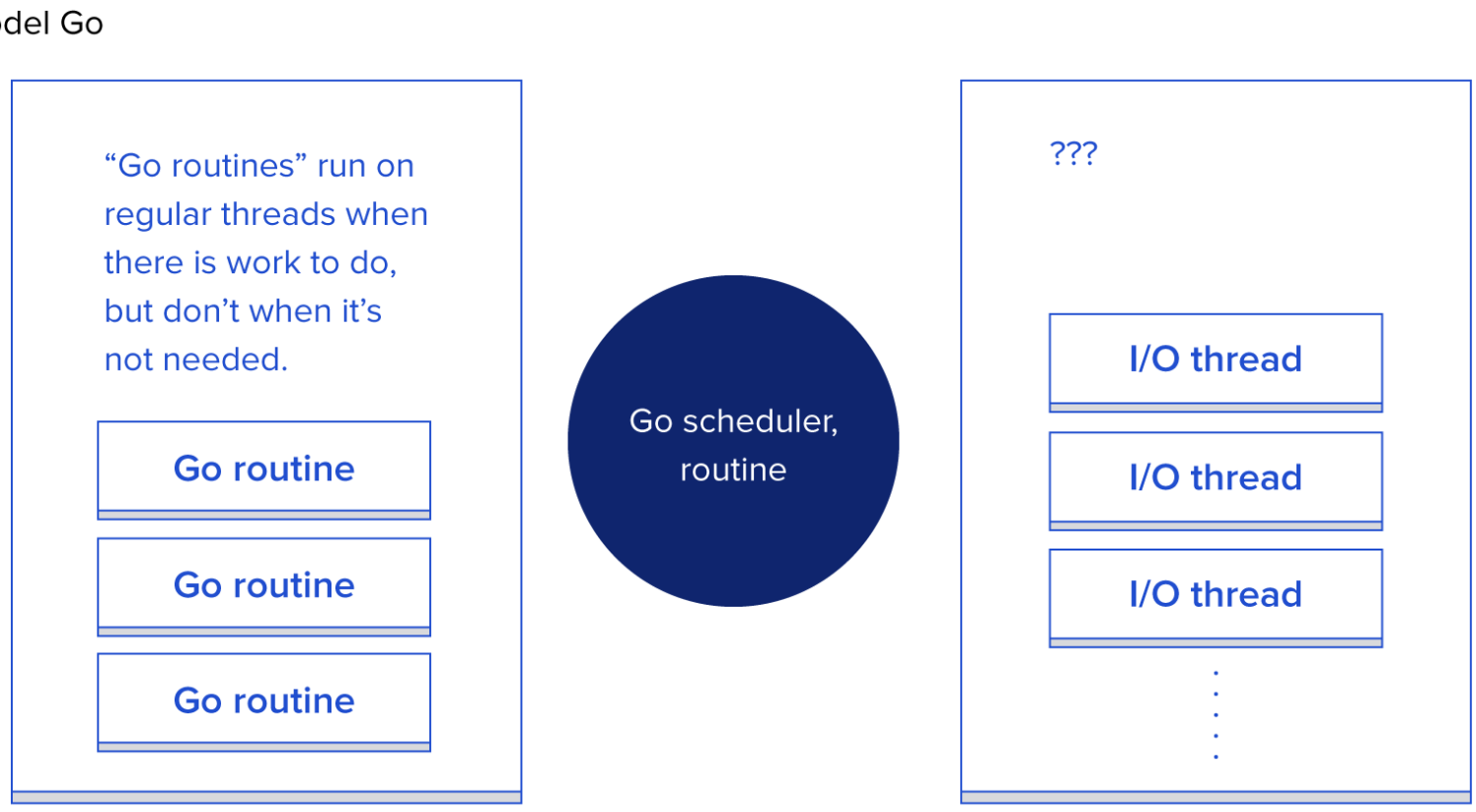


Figure: Go 运行环境的线程数由参数 **GOMAXPROCS** 决定, 通常是 CPU 核心数.

Goroutines vs Thread

Goroutine (协程)	Thread (线程)
Goroutine 由 Go 运行环境管理	Thread 由操作系统内核管理
Goroutine 独立于硬件	Thread 依赖于硬件
Goroutine 之间通过 channel 很便捷的通信	Thread 之间没有便捷的通信方式
Goroutine 之间借助 channel 可以低延迟通信	Thread 之间通信高延迟
Goroutine 没有分配 ID, 因为 Go 没有 Thread 的本地存储	Thread 有唯一 ID, Thread 有本地存储
Goroutine 开销低于 Thread	Thread 开销大于 Goroutine
Goroutine 协同调度	Thread 预先调度
Goroutine 启动快于 Thread	Thread 启动慢
Goroutine 使用可增长的分段堆栈 (2KB-1GB)	Thread 使用固定大小的分段堆栈 (2MB)

Performance: Go > Java > Node > PHP

Language	Threads vs. Processes	Non-blocking I/O	Ease of Use
PHP	Processes	No	
Java	Threads	Available	Requires Callbacks
Node.js	Threads	Yes	Requires Callbacks
Go	Threads (Goroutines)	Yes	No Callbacks Needed