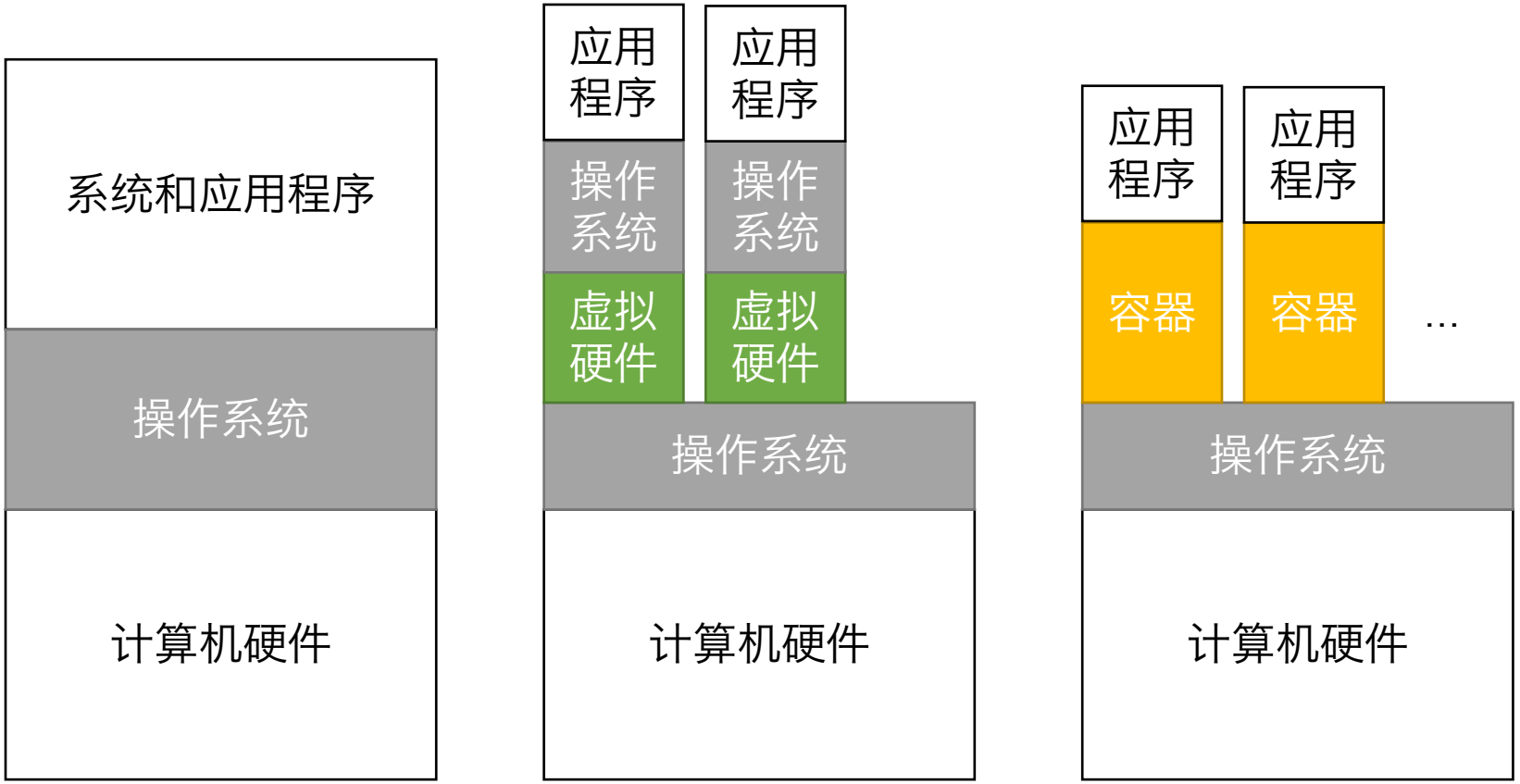
Java虚拟机

蒋炎岩

南京大学 | 计算机软件研究所 | 系统与软件分析研究组

回顾：系统虚拟化

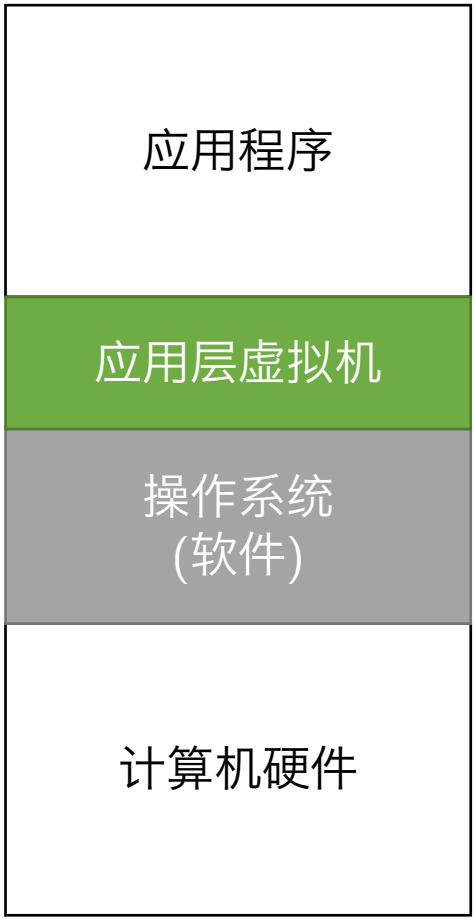

系统和应用程序
操作系统
计算机硬件
应用程序
应用程序
操作系统
操作系统
虚拟硬件
虚拟硬件
操作系统
计算机硬件
应用程序
应用程序
容器
容器
…
操作系统
计算机硬件

应用层的虚拟机

应用层的虚拟机

- 在操作系统抽象基础上提供统一的运行环境
  - 进程/线程、文件……
  - 应用层虚拟机可以提供更抽象的服务
    - 虽然是个虚拟机，但用于实现应用程序
    - 能够调用操作系统功能

```
public class HelloWorld {
  public static void
  main(String[] args) {
    System.out.println
      ("Hello World!");
  }
}
```



应用层的虚拟机：好处

- 虚拟机运行时系统对硬件和操作系统都作出了抽象
  - 更容易移植的代码
    - compile once, run everywhere (Linux, Windows, macOS, ...)
    - 和全系统虚拟机类似(你只需要写x86-64，不用管实际运行平台)
  - 虚拟运行环境
    - 没有指针运算：安全
    - 自动内存管理：只需要malloc/new，不需要free/delete ← 😍
    - 即时编译：根据程序实际运行情况优化代码

如果把编程语言比作汽车

- **C** —— 一辆赛车，它的速度是令人难以想象的快，可惜的是它每50公里就会损毁一次。
- **C++** —— 加大马力的C赛车，其有一堆新增的功能，而且只每250公里损毁一次。可是，一旦故障没人知道故障在哪。
- **Java** —— 家用旅行车，很容易驾驶，但不是很快，而且这是一个你无法伤害自己的车。
- **Python** ——一个相当不错的入门者的车。没有驾照也可以驾驶它，除非你真的想把它开得很快或是在很BT的地形上驾驶。
- **PHP** —— Oscar Mayer Wienermobile，它是一个很怪异的车，但是还是有很多的人喜欢去驾驶它。

一个最小的应用层虚拟机

Brain F**k

- 虚拟机计算机
  - 有一块(无穷大)的内存，和一个指针(ptr)
  - 有一套指令集

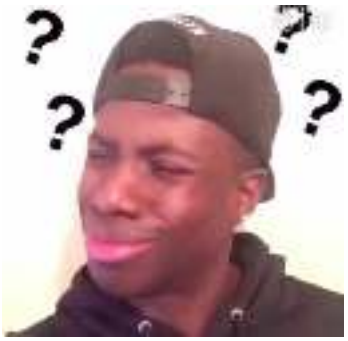
指令	解释	C代码表示
>	右移	++ptr;
<	左移	--ptr;
+	加一	++(*ptr);
-	减一	--(*ptr);
.	输出	putchar(*ptr);
,	输入	*ptr = getchar();
[	循环开始	while (*ptr) {
]	循环结束	}

指令的语义可以比机器指令更抽象

Brain F**k程序

- Hello, World

```
+[-->-[>>+>-----<<]<--<---]>-.>>>+.>
>..+++[.>]<<<<.+++.------.<<-.>>>>+.
```



- Brain f**k解释器

```
>>>+[[-]>>[-]++>+>+++++++[<++++>>++<-]++>>+>+>+++++[>++>++++++<<-]+>>>,<++[[>[
->>]<[>>]<<-]<[<]<+>>[>]>[<+>-[[<+>-]>]<[[[-]<]++<-[<+++++++++>[<->-]>>]>>]]<<
]<]<[[<]>[[>]>>[>>]+[<<]<[<]<+>>-]>[>]+[->>]<<<<[[<<]<[<]+<<[+>+<<-[>-->+<<-[>
+<[>>+<<-]]]>[<+>-]<]++>>-->[>]>>[>>]]<<[>>+<[[<]<]>[[<<]<[<]+[-<+>>-[<<+>++>-
[<->[<<+>>-]]]<[>+<-]>]>[>]>]>[>>]>>]<<[>>+>>+>>]<<[->>>>>>>>>]<<[>.>>>>>>>]<<[
>->>>>>]<<[>,>>>]<<[>+>]<<[+<<]<]
```

- 如果brain f**k是汽车

实现一个Brain F**k Runtime

- 一般用本地的语言实现(C/C++)
  - 尽量使用可移植的部分(stdlibc, STL, ...)
  - 为每个系统单独编写不可移植的部分(动态代码生成)
  - 命令行工具
    - bfvm --memory 64M programs/hello.bf
- 执行代码
  - 取指令 → 译码 → 执行 (NEMU)

让代码飞起来

- 虚拟机有OS支持，可以调用很多功能
  - 比如把代码翻译成C代码然后(O2)编译执行
  - 动态分配内存
  - ……

```
#define EMIT_OP(op) \
  { code += op "\n"; break; }

std::string code;
// ...
switch(op) {
  case '>': EMIT_OP("++ptr");
  case '<': EMIT_OP("--ptr");
  // ...
  case ']': EMIT_OP("}");
}
// ...
compile_and_run(code);
```

Java虚拟机

一些历史

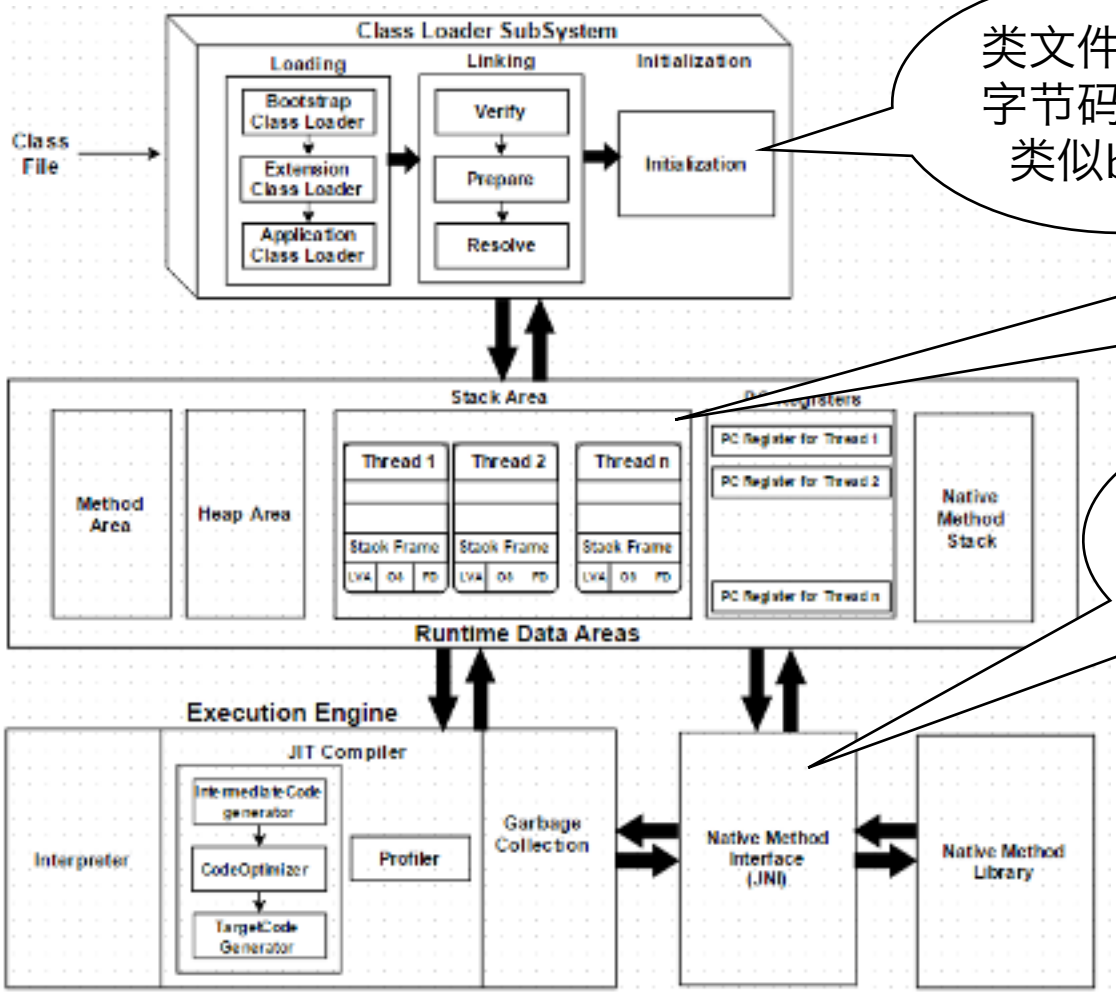
- In the early 90s, extending the power of network computing to the activities of everyday life was a radical vision. In 1991, a small group of Sun engineers called the "Green Team" believed that the next wave in computing was the union of digital consumer devices and computers. Led by James Gosling, the team worked around the clock and created the programming language that would revolutionize our world – Java.
  - 1995年发布1.0版本，之后借助互联网浪潮风靡全球
  - Write once, run everywhere
    - 跨平台、小而紧凑的通用代码

Java虚拟机(JVM)

- 为Java语言实现的虚拟机
  - 类封装了代码和数据的表示
  - 不支持指针运算，类的实例(对象)在堆区中分配
  - 运行时每个函数(方法)调用创建一个栈帧(stack frame)
  - 堆栈计算机：操作数从栈中取、输出到栈中
  - 虚拟机建立在多线程基础上

- 虚拟机的指令集成为“字节码”(Bytecode)

一个精细版本的Brain F**k虚拟机



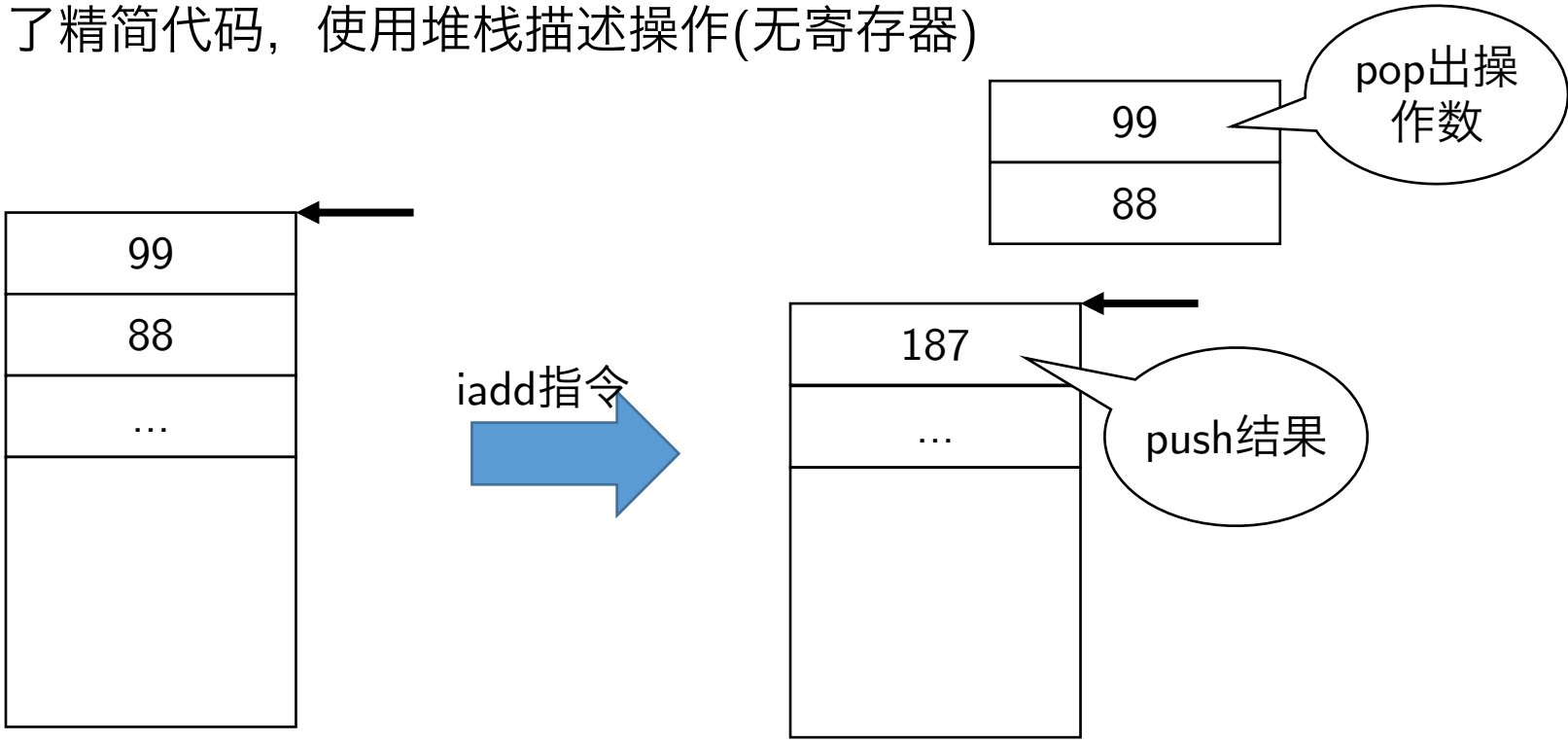
类文件描述了字节码序列，类似bf代码

与计算机类似：虚拟处理器(线程)、虚拟存储(栈/堆)

运行在操作系统上，指令集不受硬件约束

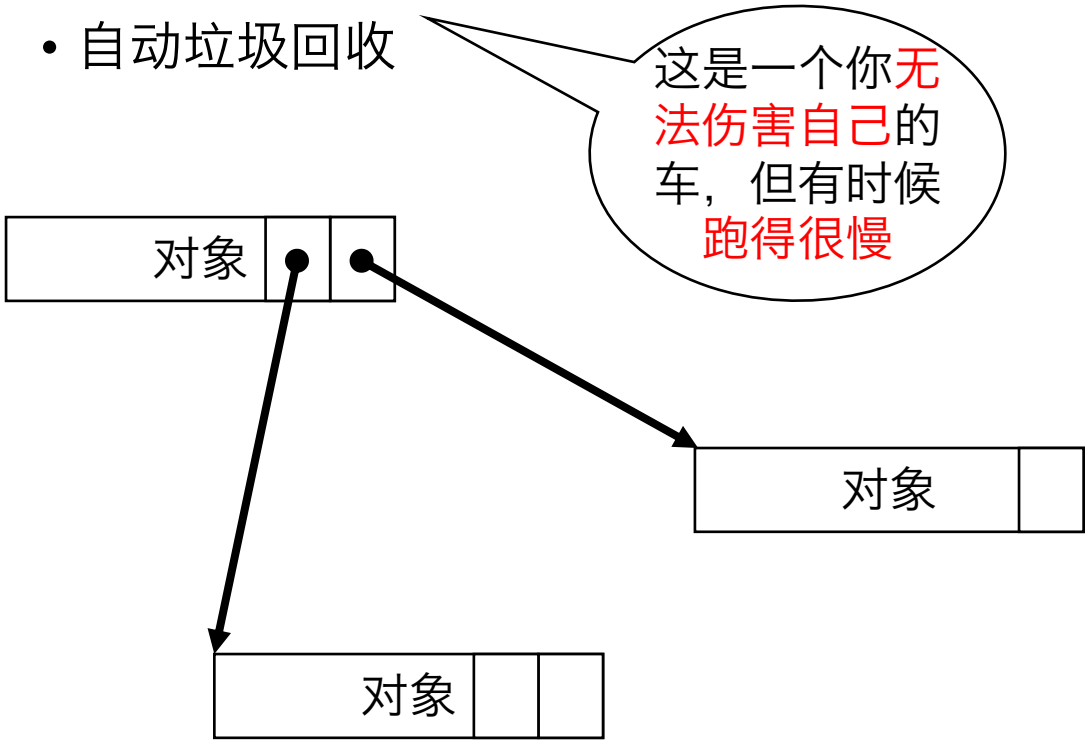
跨平台、小而紧凑

- 一开始是“解释型”的语言
  - 就和大家的NEMU一样，取指令 → 译码 → 执行
  - 为了精简代码，使用堆栈描述操作(无寄存器)



托管(Managed)的堆区

- 没有指针运算，没有undefined behavior
  - 非空指针(引用)一定是合法对象；访问空指针抛出异常
  - 自动垃圾回收

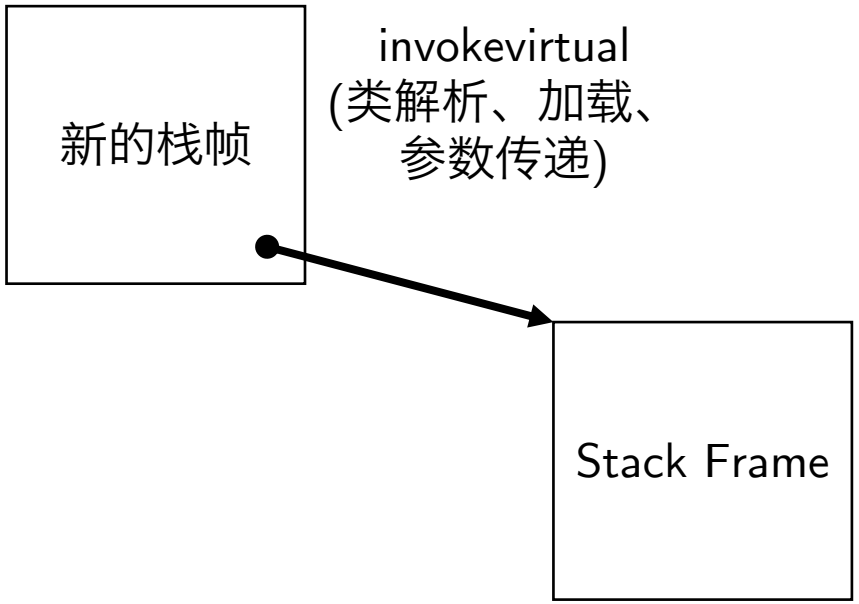


内存相关指令

- 对象分配
  - anew, anewarray, newarray, multianewarray
- 堆栈访问
  - iload, istore, iconst, dup, dup2, ...
- 对象的访问
  - 对象/类静态数据 getfield/putfield, getstatic/putstatic
  - 数组 Taload/Tastore (T = {i, s, f, d, j, a})

控制流指令

- 方法调用
  - invokevirtual, invokeinterface, invokestatic, invokespecial, invokedynamic (多态、动态语言支持)
- 普通控制流
  - goto, goto_w, lookupswitch
- 异常控制流
  - athrow

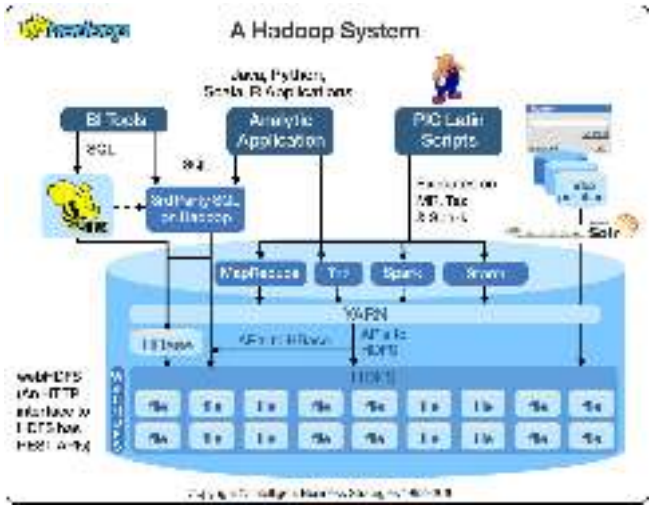


Java Bytecode：小结

- JVM脱离了硬件(用逻辑门实现)的约束
  - 可以用一条指令做很多non-trivial的事情
  - 甚至有lock/unlock (monitorenter/monitorexit)

- 从本质上和x86很像
  - 做过NEMU，就能实现一个Java虚拟机
    - RTFM (JVM Spec)，然后按标准实现

今天的JVM：最成熟的运行时平台之一



- 典型应用
  - 企业应用(阿里巴巴)
  - Android (Dalvik, ART虚拟机)
  - Hadoop生态圈
- Write Once, ~~Run~~ (Configure) Everywhere
  - 暴涨的应用不可避免带来的麻烦😂
- JVM是一个指令集，可以用来支持其他语言的运行
  - Scala, Kotlin, Jython, JRuby, ...