Alibaba Group
阿里巴巴集团

Alibaba JDK协程

——免费的性能午餐

Alibaba JVM/郁磊

系统软件事业部　打造具备全球竞争力、效率最优的系统软件

极客时间

重拾极客精神·提升技术认知

每天10分钟，邀请顶级技术专家，为你传道授业解惑。

扫一扫，试读专栏

主办方 Geekbang› 极客邦科技 InfoQ

ArchSummit

全球架构师峰会 2017

12月8-9日 北京 · 国际会议中心

APSEC 2017

APSEC 2017

24th Asia-Pacific Software Engineering Conference

4-8 December 2017, Nanjing, Jiangsu, China

12月4–8日

中 国 南 京

了解详情

AiCon

全球人工智能技术大会 2018

助力人工智能落地

2018.1.13 - 1.14 北京国际会议中心

扫描关注大会官网

AJDK(Alibaba/Ant JDK)

基于OpenJDK的最适合云上环境的JDK

海量用户场景

服务阿里电商、菜鸟等大部分业务，拥有大量超大规模场景

社区合作

与Oracle展开合作，参与OpenJDK社区

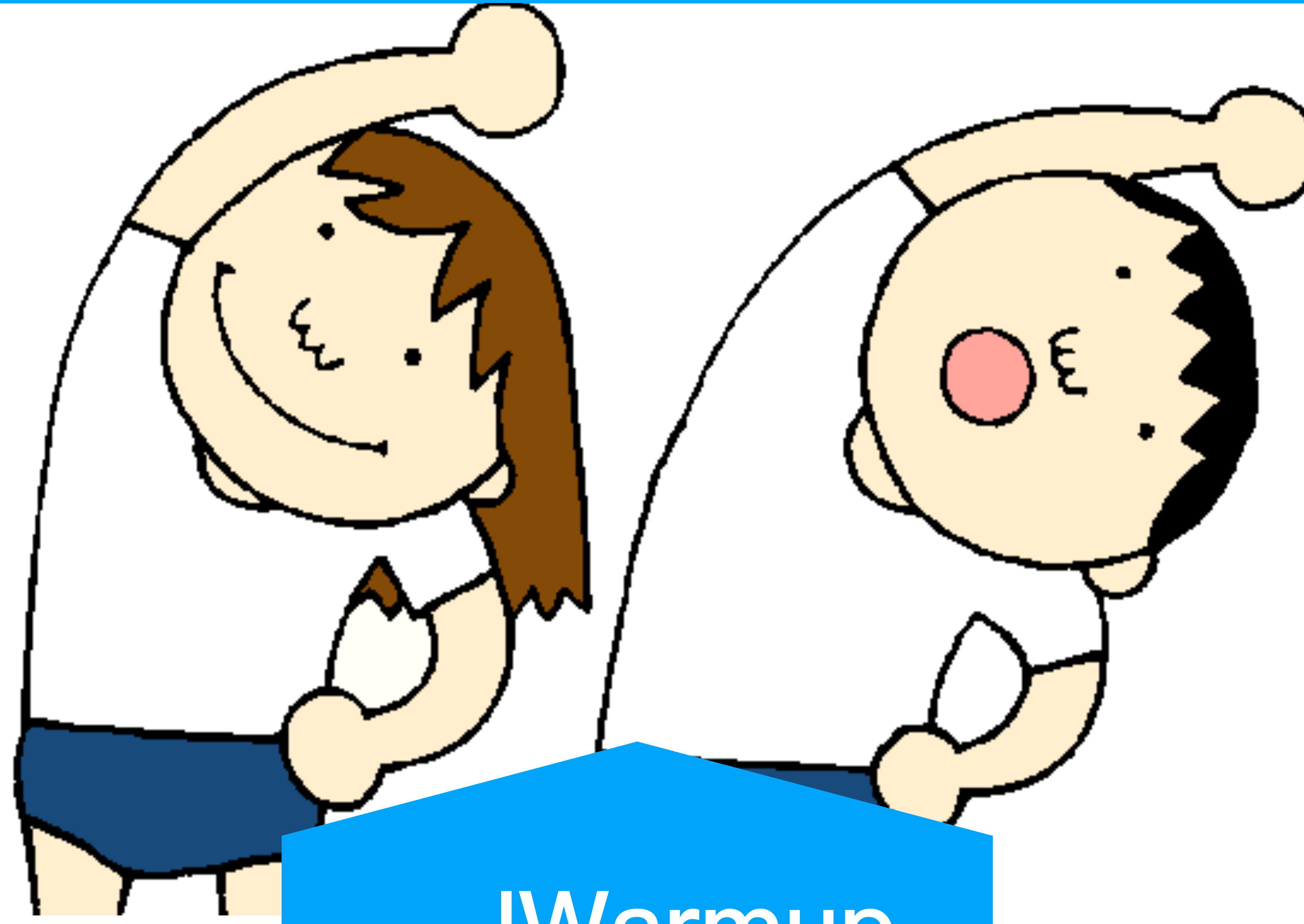
针对性优化

针对云上、互联网、容器、环境进行优化

AJDK新特性

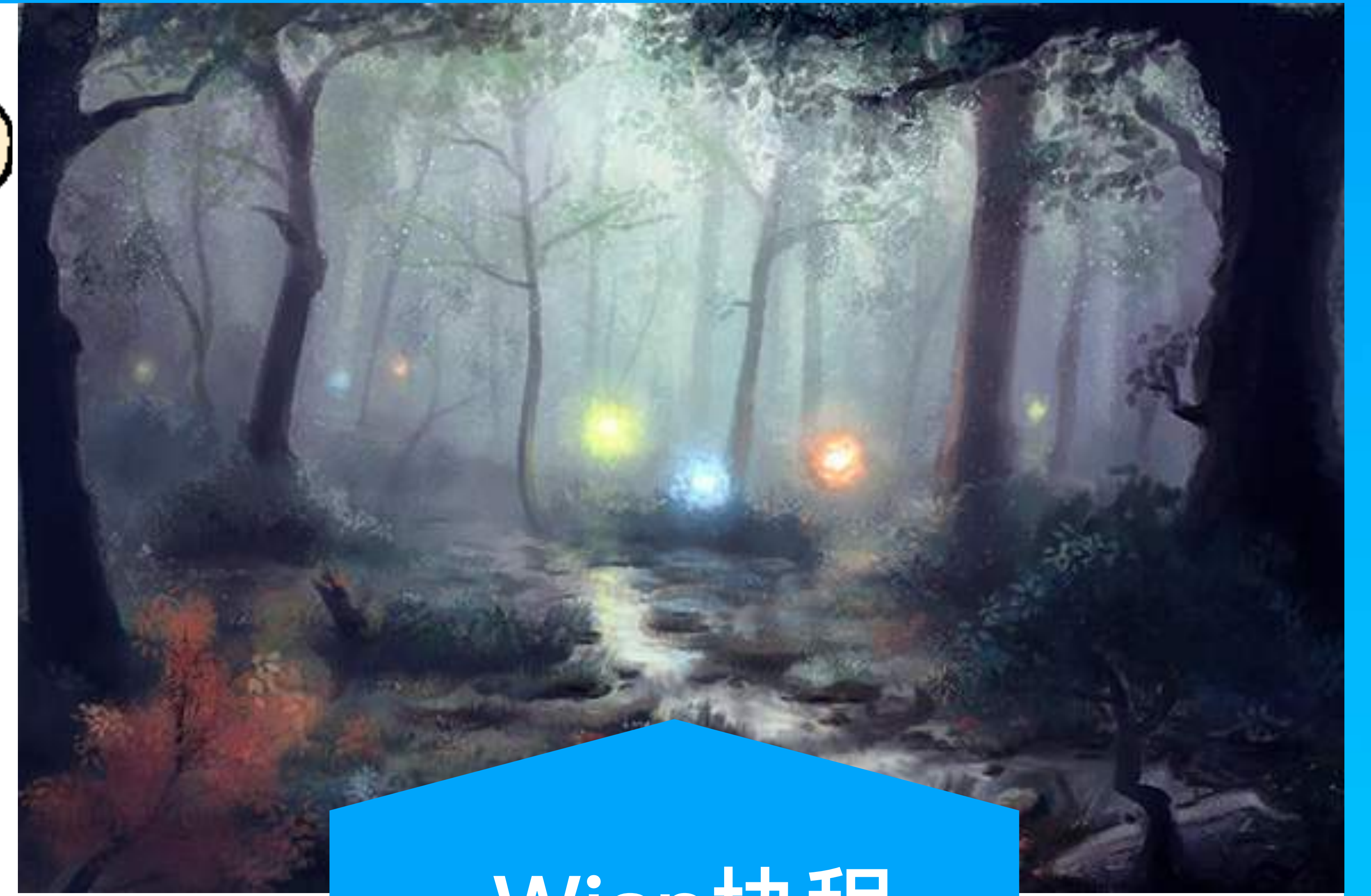
多租户

高密度部署:
应用间共享资源



JWarmup

快速上线:
解决启动瞬间的CPU飙高



Wisp协程

大量分布式调用:
以同步的方式进行异步编程

协程

Coroutine

01 服务端线程模型

02 使用协程

03 透明的Wisp协程

04 Wisp协程实践

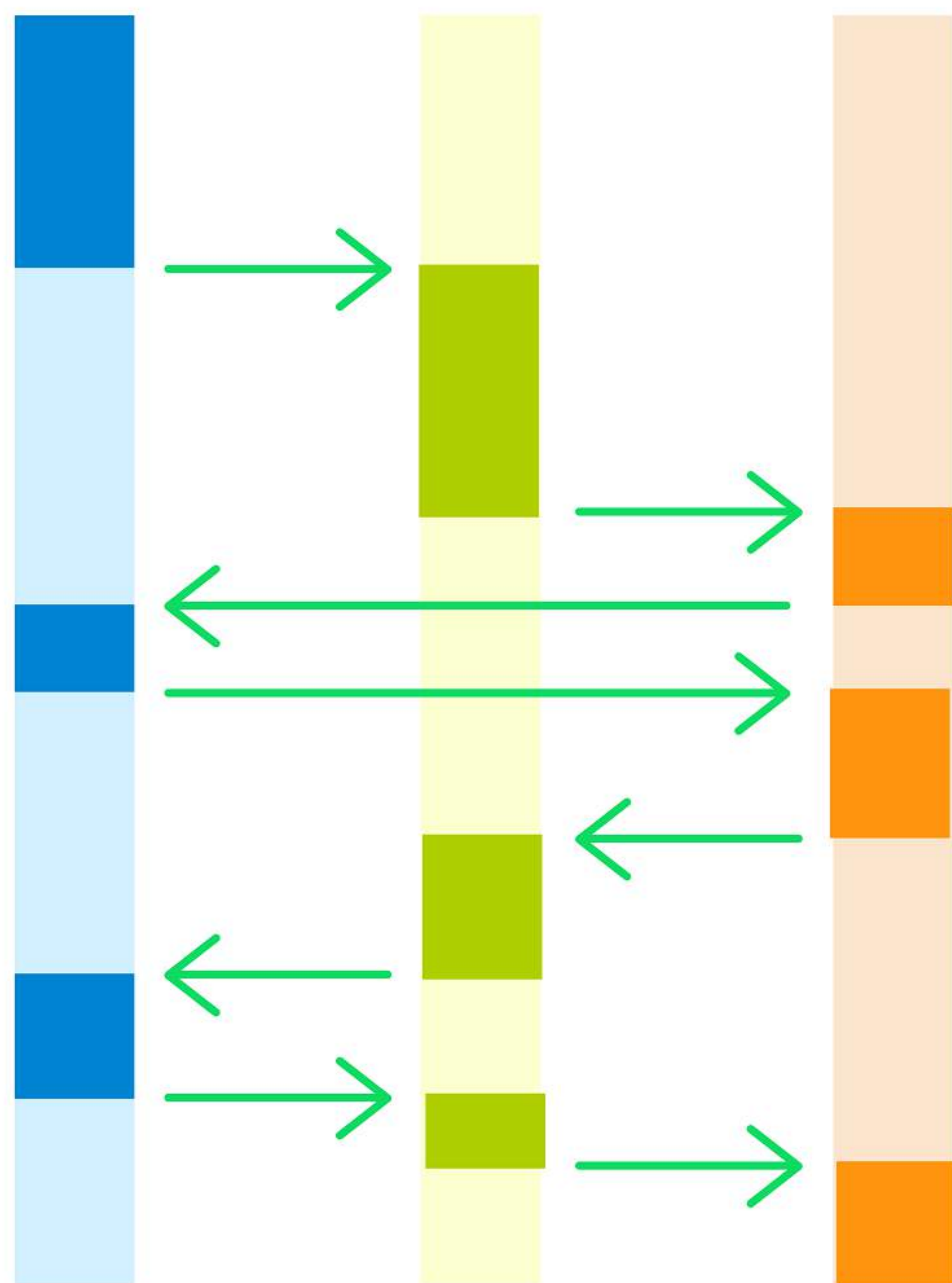
01 服务端线程模型

写出最快的WebServer

多线程模型和事件驱动模型

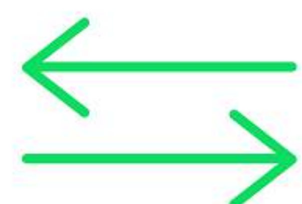
TRADITIONAL SERVER

PROCESS 1 PROCESS 2 PROCESS 3



NGINX WORKER

PROCESS

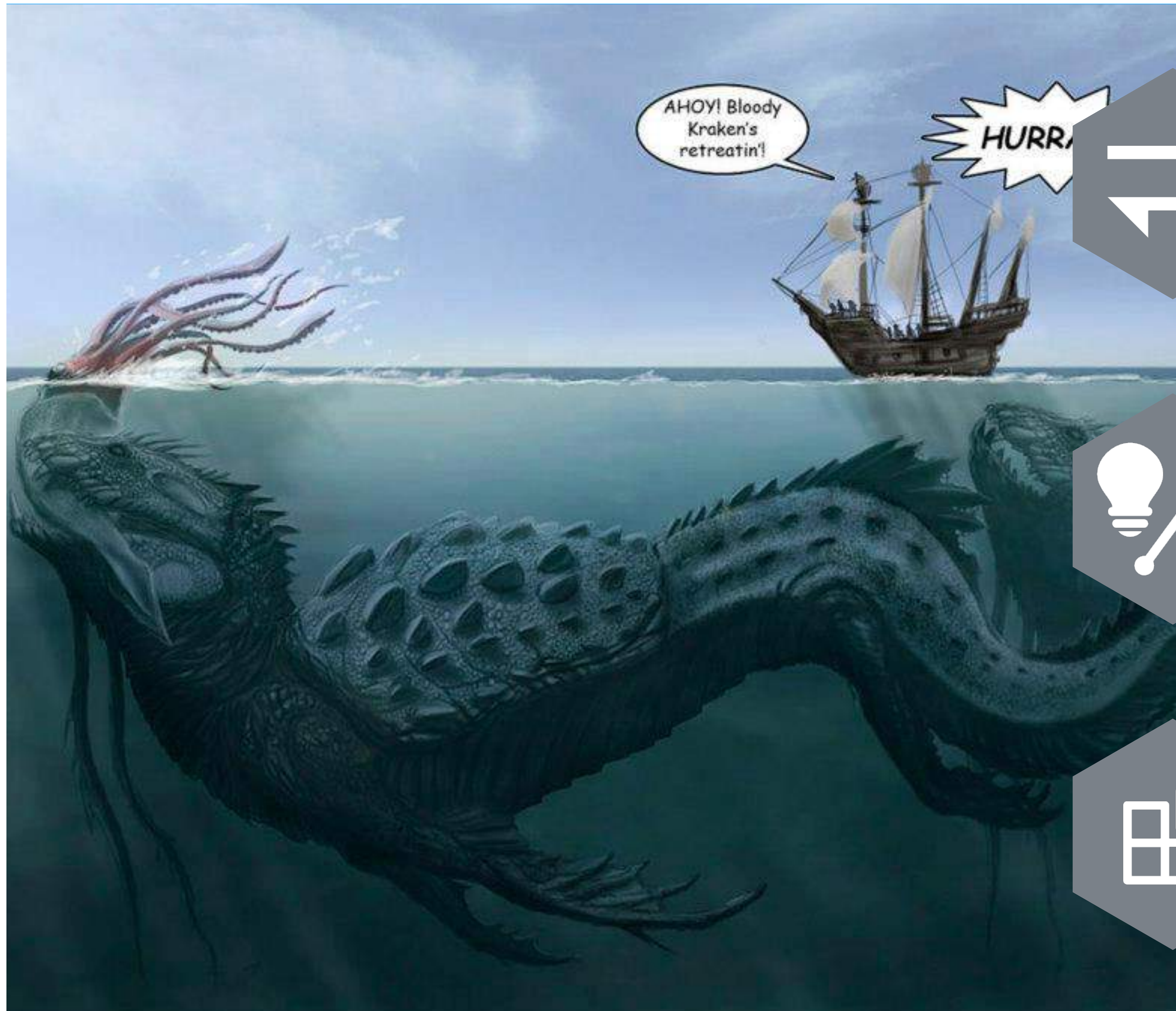

TASK SWITCHES

PROCESSING REQUEST 1

PROCESSING REQUEST 2

PROCESSING REQUEST 3

图片来自nginx.org



上下文切换

上下文切换吃掉了宝贵的CPU资源

对上下文切换的误区

进出内核 vs. 调度

冰山一角

真正的损耗远大于想象

上下文切换

Syscall vs context switch ns.

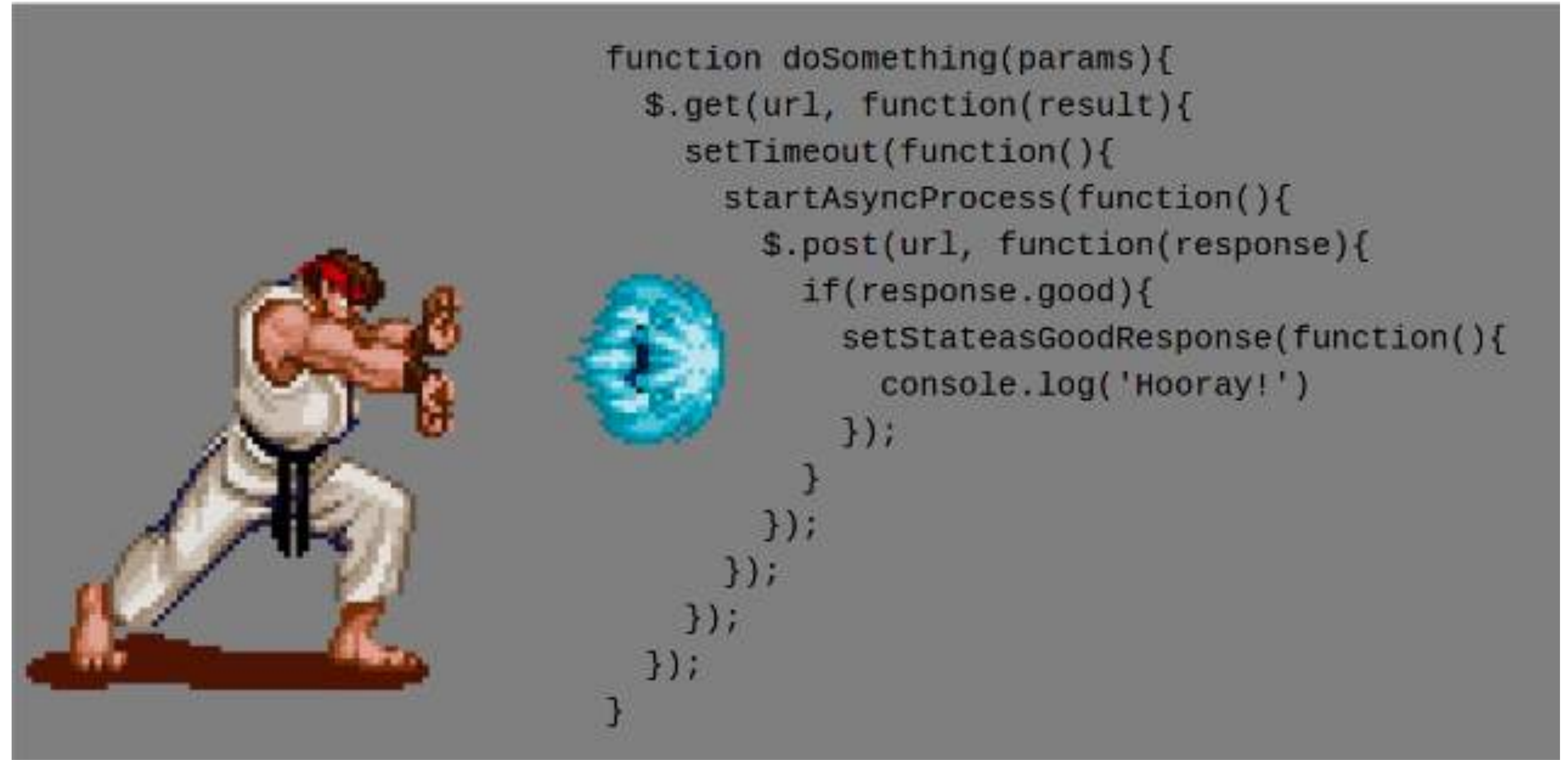
	syscall	cs
Intel 5150	105	4300
Intel E5520	58	4500
Intel X5550	52	3000
Intel L5630	58	3000
Intel E5-2620	67	3000

数据来源: http://blog.tsunanet.net/2010/11/how-long-does-it-take-to-make-context.html

使用异步编程

Callback hell

业务逻辑难写而难以维护

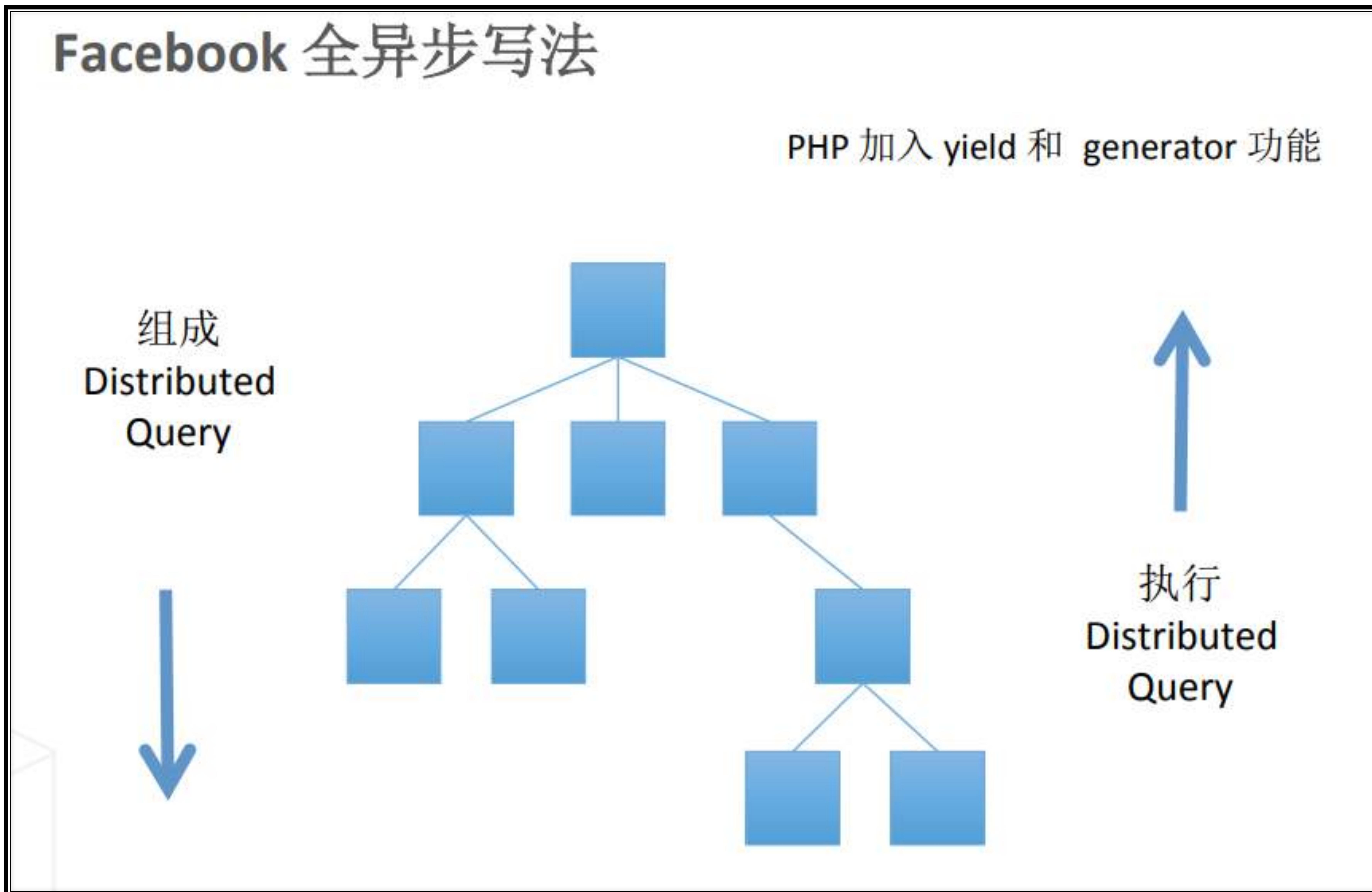


BLOCKING OPERATION

阻塞调用

Nginx近期引入了线程池

可以引入全异步处理



02 使用协程

简化异步编程

协程是什么

```c
/* Decompression code */
while (1) {
    c = getchar();
    if (c == EOF)
        break;
    if (c == 0xFF) {
        len = getchar(
        c = getchar();
        while (len--)
            emit(c);
    } else
        emit(c);
}
emit(EOF);
```

```c
/* Parser code */
while (1) {
    c = getchar();
    if (c == EOF)
        break;
    if (isalpha(c)) {
        do {
            add_to_token(c);
            c = getchar();
        } while (isalpha(c));
        got_token(WORD);
    }
    add_to_token(c);
    got_token(PUNCT);
}
```

用协程来组织逻辑

- emit() 和 parser::getchar()会切换到另一个协程
- 如果没有协程需要两个线程结合pipe来组织
- 逻辑清晰且性能高

Kotlin协程

上下文包含什么?

代码执行位置

方法编译成状态机，代码执行位置被保存在数据结构中

局部变量

编译器生成代码保存/恢复局部变量

调用栈

调用连被隐式串成链表

执行器

自定义的Dispatcher

Kotlin协程

如何实现

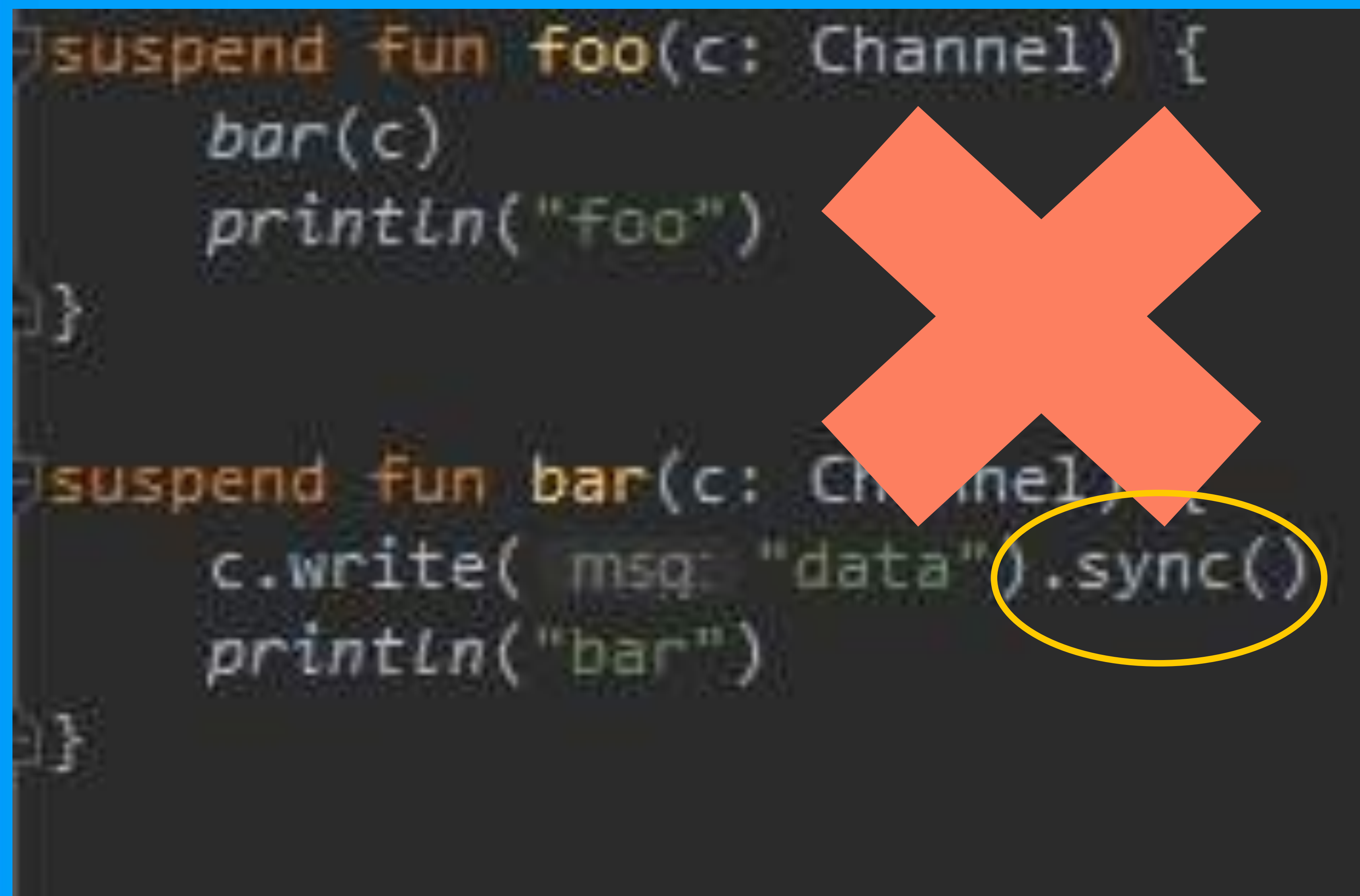
```
fun main(args: Array<String>) {
    val coro = async(CommonPool) {
        println("enter")
        delay(100)
        println("exit")
    }
    runBlocking {
        coro.await()
    }
}
```

```
ILOAD 1
PUTFIELD AppKt$foo$1.I$0 : I
# call delay....
# resume
GETFIELD AppKt$foo$1.I$0 : I
ISTORE 1
```

```
class AppKt$main$coro$1 extend CoroutineImpl {
    int label = 0;

    public void doResume(Object data) {
        switch (this.label) {
            case 0:
                System.out.println("enter");
                this.label = 1;
                data = DelayKt.delay(100);
                if (data == COROUTINE_SUSPENDED
                    return;
            case 1:
                System.out.println("exit");
                return;
            default:
                throw new IllegalStateException
        }
    }
}
```

使用kotlin协程简化异步



Control flow invert

Coroutine way

```
suspend fun foo(c: Channel) {
    bar(c)
    println("foo")
}

suspend fun bar(c: Channel) {
    c.aWrite( msg: "data")
    print("bar")
}
```

使用kotlin协程简化异步

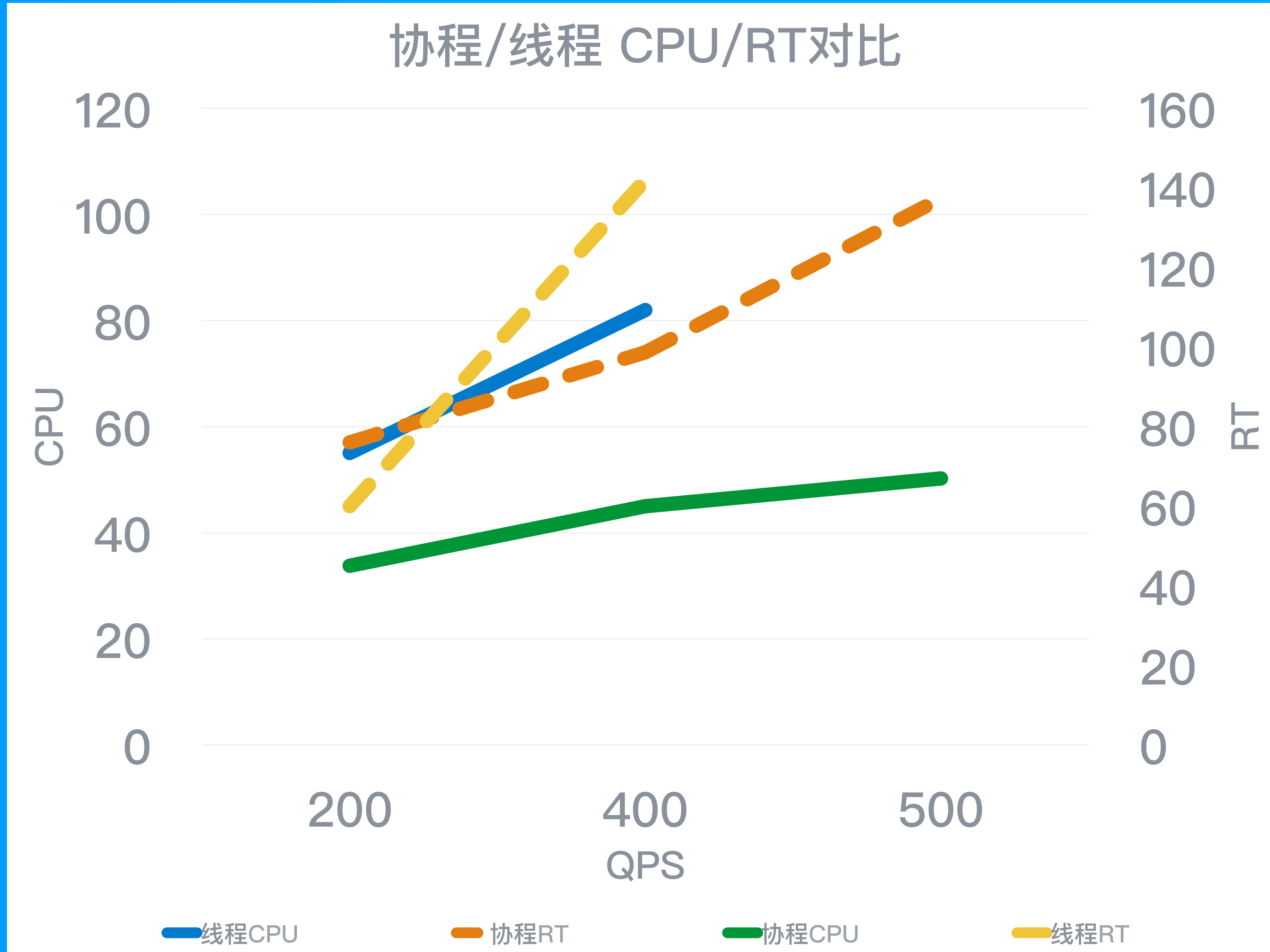
异步操作恢复协程

```kotlin
suspend fun Channel.aWrite(msg: Any): Int =
        suspendCoroutine { cont ->
            write(msg).addListener { future ->
                if (future.isSuccess) {
                    cont.resume(0)
                } else {
                    cont.resumeWithException(fut
                }
            }
        }
```

- 阻塞调用会挂起协程执行器
- 封装异步操作(如图)
- call/cc

在Java容器中使用协程代替线程

将协程引入服务端

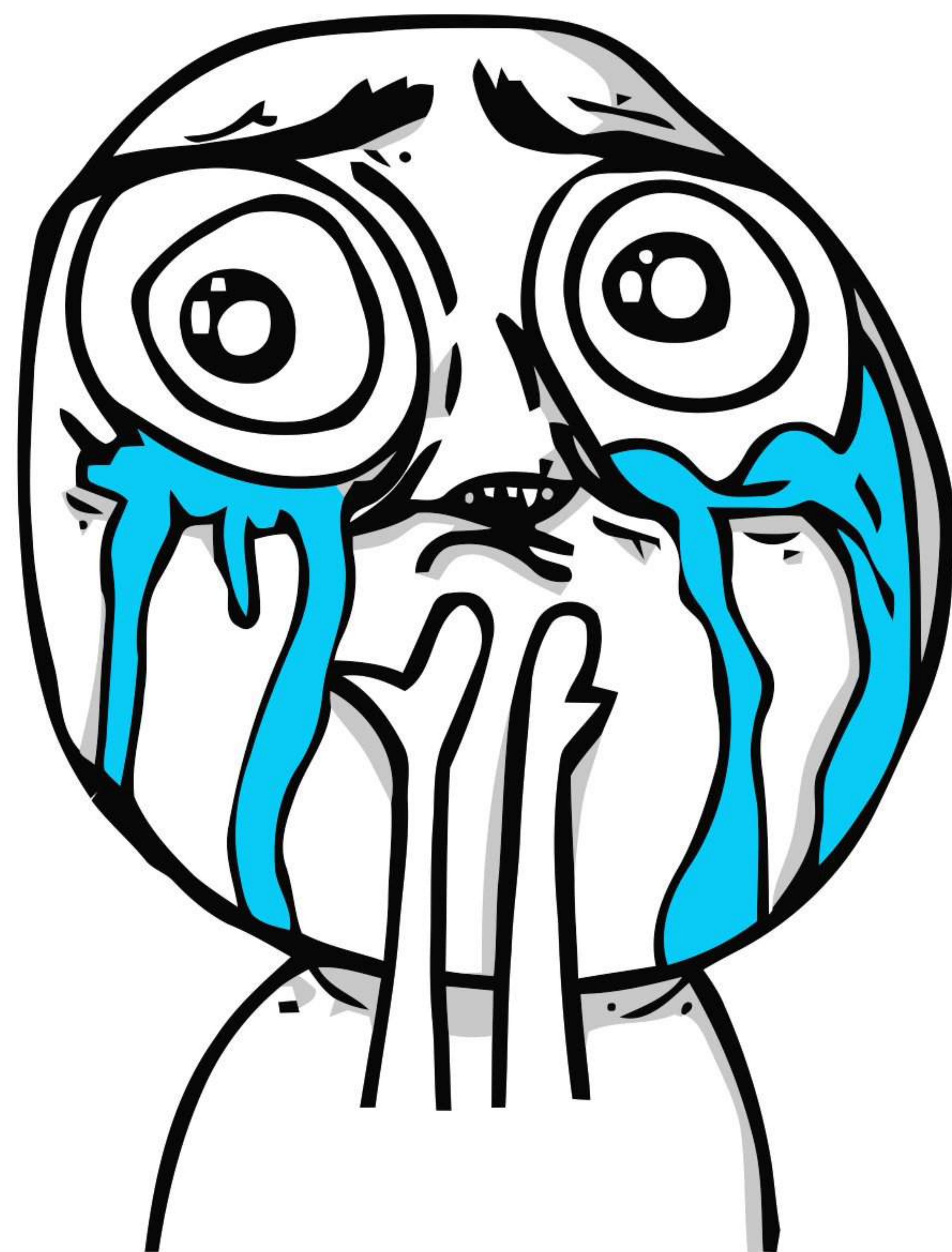


用kotlin协程跑一个demo server:

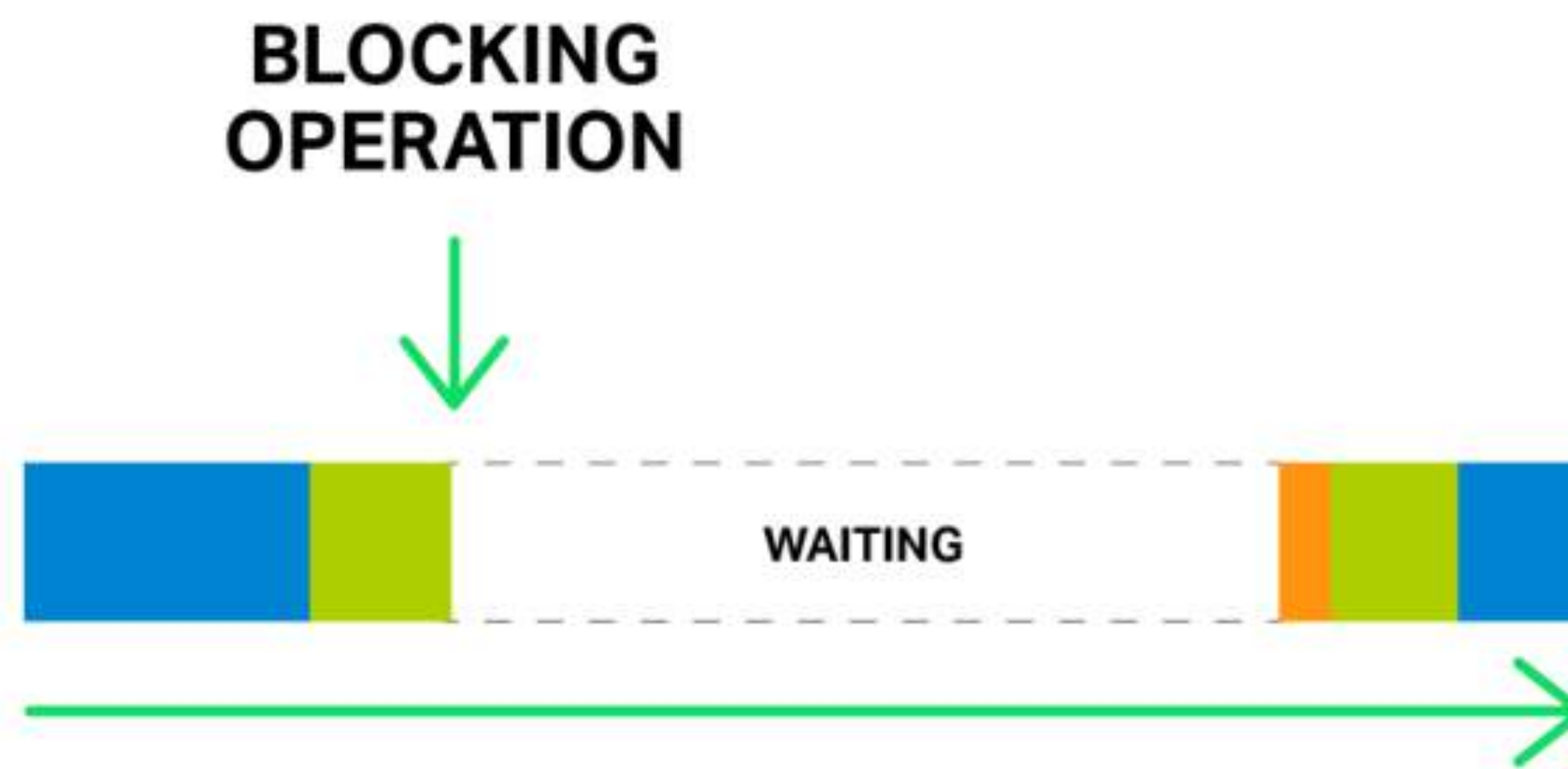
- 业务特点： 大量RPC,访问200次echo
- 仅仅将线程池替换成了创建协程
- 在IO密集情况下，协程获得异步的好处，大幅提高性能

那么，痛苦完结了吗?

更多的痛苦!



We need a completely coroutine aware Java runtime!



03 透明的Wisp协程

无痛地解决阻塞问题

轻量

希望协程的开销很小

透明

对使用者以及现有代码完全透明

Wisp

鬼火的意思

Wisp层次

干净的结构化设计

JKU协程

mlvm提供的栈式协程实现

The xv6 Context Switch Code

```
1 # void swtch(struct context **old, struct context *new);
2 #
3 # Save current register context in old
4 # and then load register context from new.
5 .globl swtch
6 swtch:
7         # Save old registers
8         movl 4(%esp), %eax          # put old ptr into eax
9         popl 0(%eax)                # save the old IP
10        movl %esp, 4(%eax)          # and stack
11        movl %ebx, 8(%eax)          # and other registers
12        movl %ecx, 12(%eax)
13        movl %edx, 16(%eax)
14        movl %esi, 20(%eax)
15        movl %edi, 24(%eax)
16        movl %ebp, 28(%eax)
17
18        # Load new registers
19        movl 4(%esp), %eax          # put new ptr into eax
20        movl 28(%eax), %ebp         # restore other registers
21        movl 24(%eax), %edi
22        movl 20(%eax), %esi
23        movl 16(%eax), %edx
24        movl 12(%eax), %ecx
25        movl 8(%eax), %ebx
26        movl 4(%eax), %esp          # stack is switched here
27        pushl 0(%eax)               # return addr put in place
28        ret                         # finally return into new ctxt
```

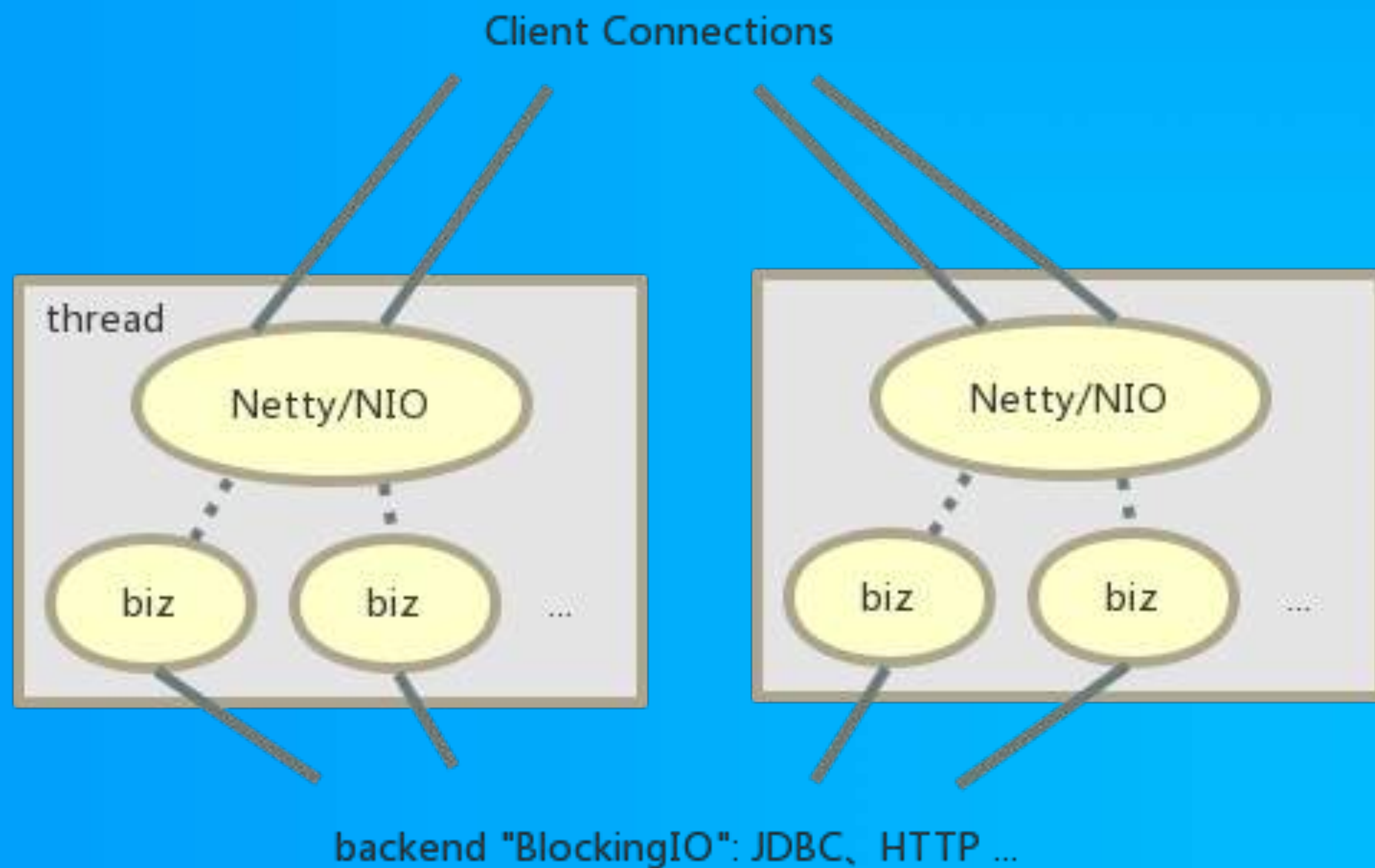
(1) (2) (3)

与OS内的线程切换方式一致

- (1) 保存pc
- (2) 保存sp
- (3) 保存callee-save寄存器

WispEngine线程模型

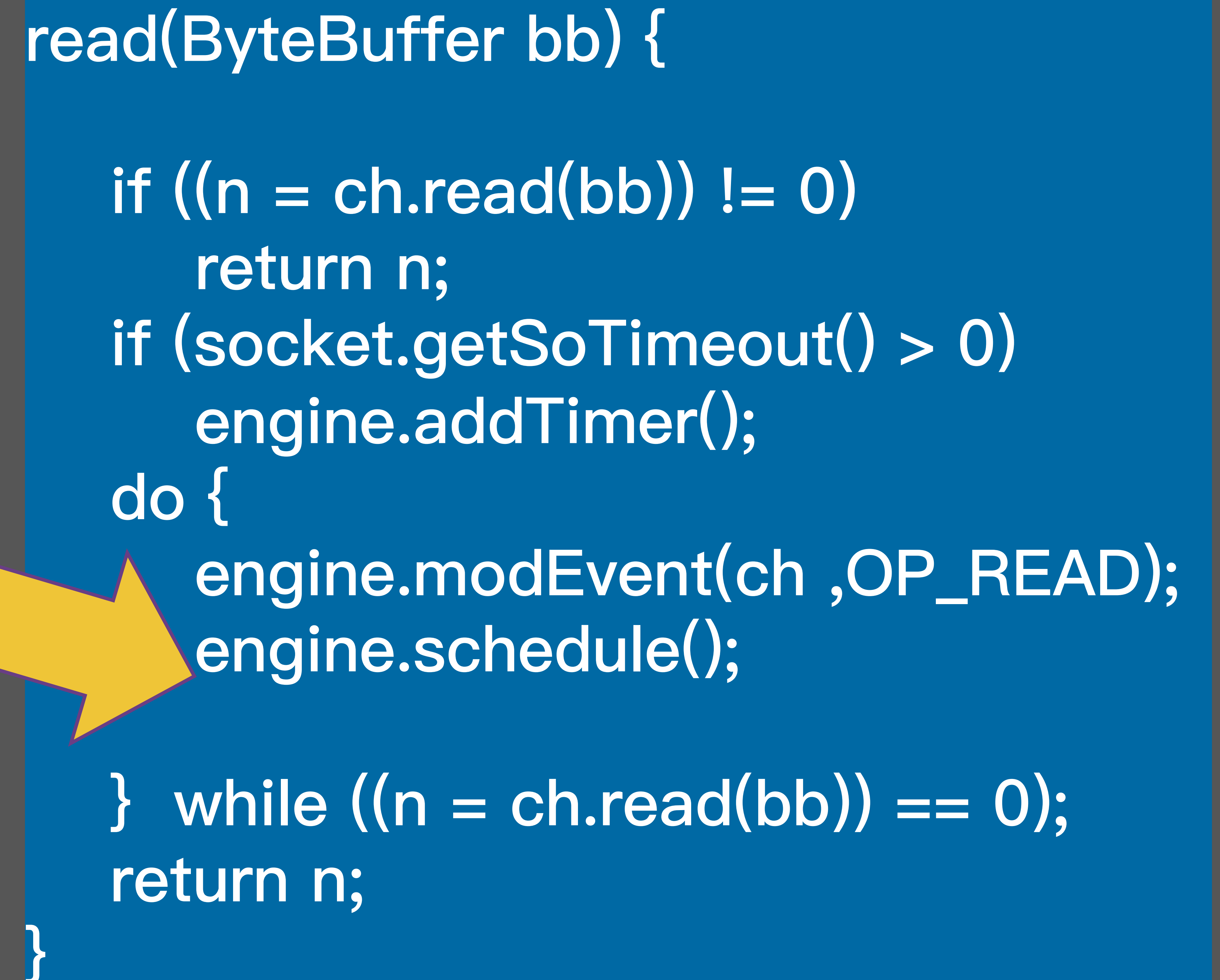
可自由组合的模型



- 协程挂在线程下面，1:N模型
- 维护队列、事件循环来监听网络、unpark、超时事件
- 支持跨线程的unpark、创建协程行为

运行时hook

```
Java：
synchronized(o){
        JVM:
        SharedRuntime::complete_monitor_locking_C
        {
            if (
                Java：
                WispEngine.schedule()
            }
        }
}
```



一个例子

```
Coroutine [0x7fa81989d3c0]
      at java.dyn.CoroutineSupport.symmetricYieldTo(CoroutineSupport.java:157)
      at java.dyn.Coroutine.yieldTo(Coroutine.java:110)
      at
      at
      at com.alibaba.wisp.engine.WispEngine.doSchedule(WispEngine.java:558)
      at
      at
      at                                  .accept
      at java.net.ServerSocket.accept(ServerSocket.java:558)
      at ServerTest.main(ServerTest.java:14)

Coroutine [0x7fa7f29f9080]
      at java.dyn.CoroutineSupport.symmetricYieldTo(CoroutineSupport.java:157)
      at java.dyn.Coroutine.yieldTo(Coroutine.java:110)
      at
      at
      at com.alibaba.wisp.engine.WispEngine.doSchedule(WispEngine.java:558)
      at
      at
      at
      at
      at
      at                                            .read
      at ServerTest.client(ServerTest.java:22)
      at ServerTest$$Lambda$4/705927765.run(Unknown Source)
      at
      at java.dyn.CoroutineBase.startInternal(CoroutineBase.java:60)
```

```java
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class ServerTest {
    public static void main(String[] args) throws Except
        ServerSocket ss = new ServerSocket( port: 2017);
        for (int i = 0; i < 3; i++) {
            WispEngine.dispatch(ServerTest::client);
        }
        while (true) {
            ss.accept();
            // do nothing
        }
    }

    private static void client() {
        try {
            Socket so = new Socket( host: "localhost",  p
            so.getInputStream().read();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

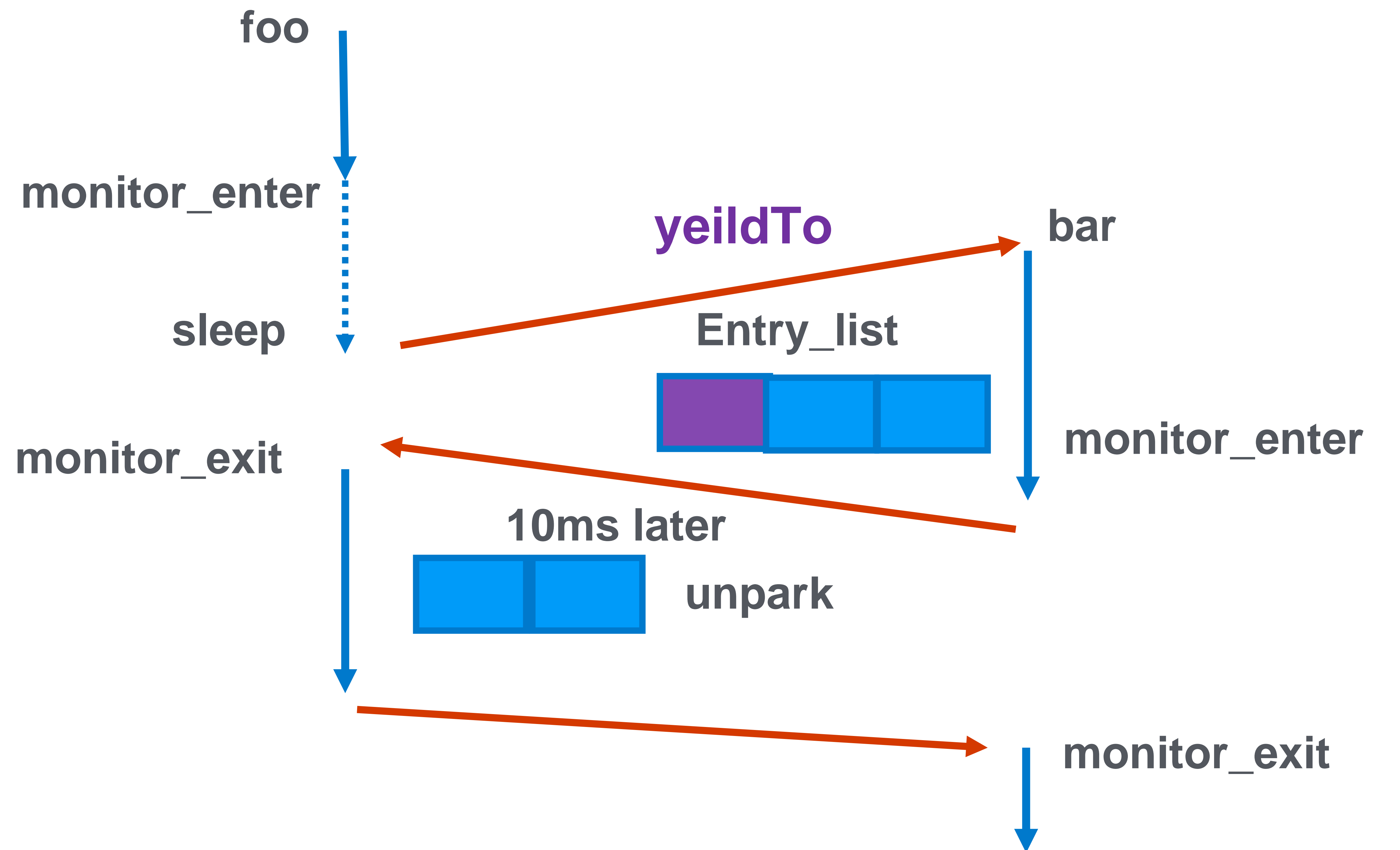
ObjectMonitor

- Fast Lock
  - 协程的栈不同，恰好包含了不同owner的语义
- Heavy Lock
  - 确保每个协程的Self值不同
  - 当获取不到锁时尝试调度

```
    WispEngine.dispatch(s::foo);
    WispEngine.dispatch(s::bar);
}

private synchronized void foo() {
    try {
        Thread.sleep( millis: 10);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

private synchronized void bar() {
}
```



复杂的例子

```java
private void bar() {
    assertEQ(count++, rhs: 2);
    synchronized (this) {
        assertEQ(count++, rhs: 5);
        finishCnt++;
        fooCond = true;
        notifyAll();
    }
}
```

```java
lic class WaitNotifyTest {
  public static void main(String[] args) throw
      for (int i = 0; i < 1; i++) {
          WaitNotifyTest s = new WaitNotifyTes
          assertEQ(s.count++, rhs: 0);
          WispEngine.dispatch(s::foo);
          WispEngine.dispatch(s::bar);
          assertEQ(s.count++, rhs: 3);
          s.latch.countDown();
          synchronized (s) {
              while (s.finishCnt < 2) {
                  s.wait();
              }
          }
      }
  }

  private int count = 0;
  private int finishCnt = 0;
  private CountDownLatch latch = new CountDown
  private boolean fooCond = false;
```

```java
private synchronized void foo() {
    assertEQ(count++, rhs: 1);
    try {
        latch.await();
    } catch (InterruptedException e)
        e.printStackTrace();
    }
    assertEQ(count++, rhs: 4);
    while (!fooCond) {
        try {
            wait();
        } catch (InterruptedExceptic
            e.printStackTrace();
        }
    }
    assertEQ(count++, rhs: 6);
    finishCnt++;
    notifyAll();
}
```

主要Hook的Java Runtime

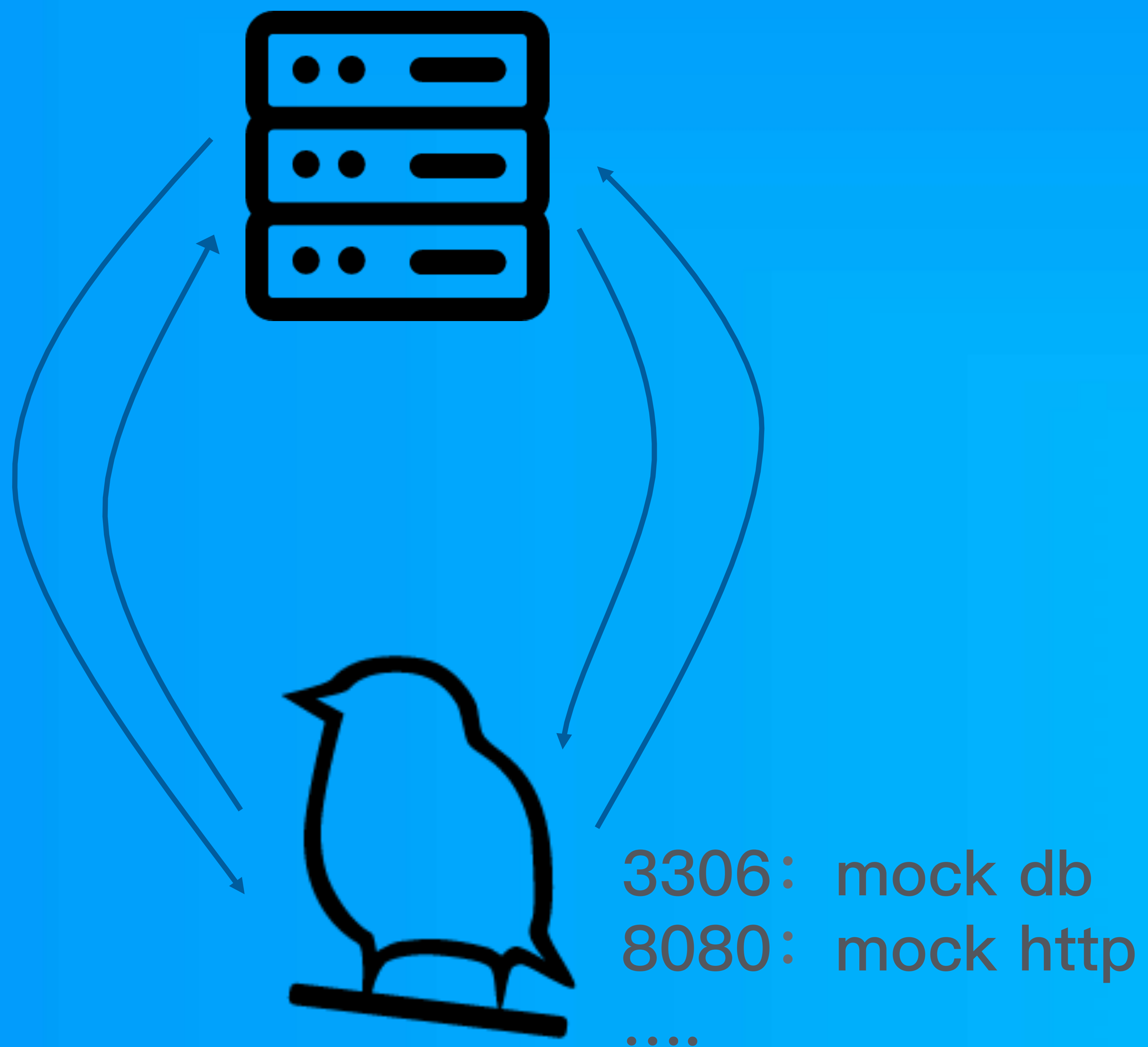
- Java.util.concurrent (Lock, CountDownLatch…)
- Blocking IO(java.net)
- ThreadLocal
- ObjectMonitor
  - synchronized
  - Object.wait
  - Object.notify/Object.notifyAll

API

-Dcom.alibaba.shiftThreadModel

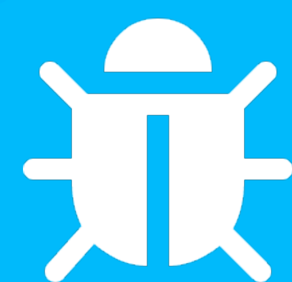
04 Wisp协程实践

协程在阿里巴巴



研发

专家review保证质量



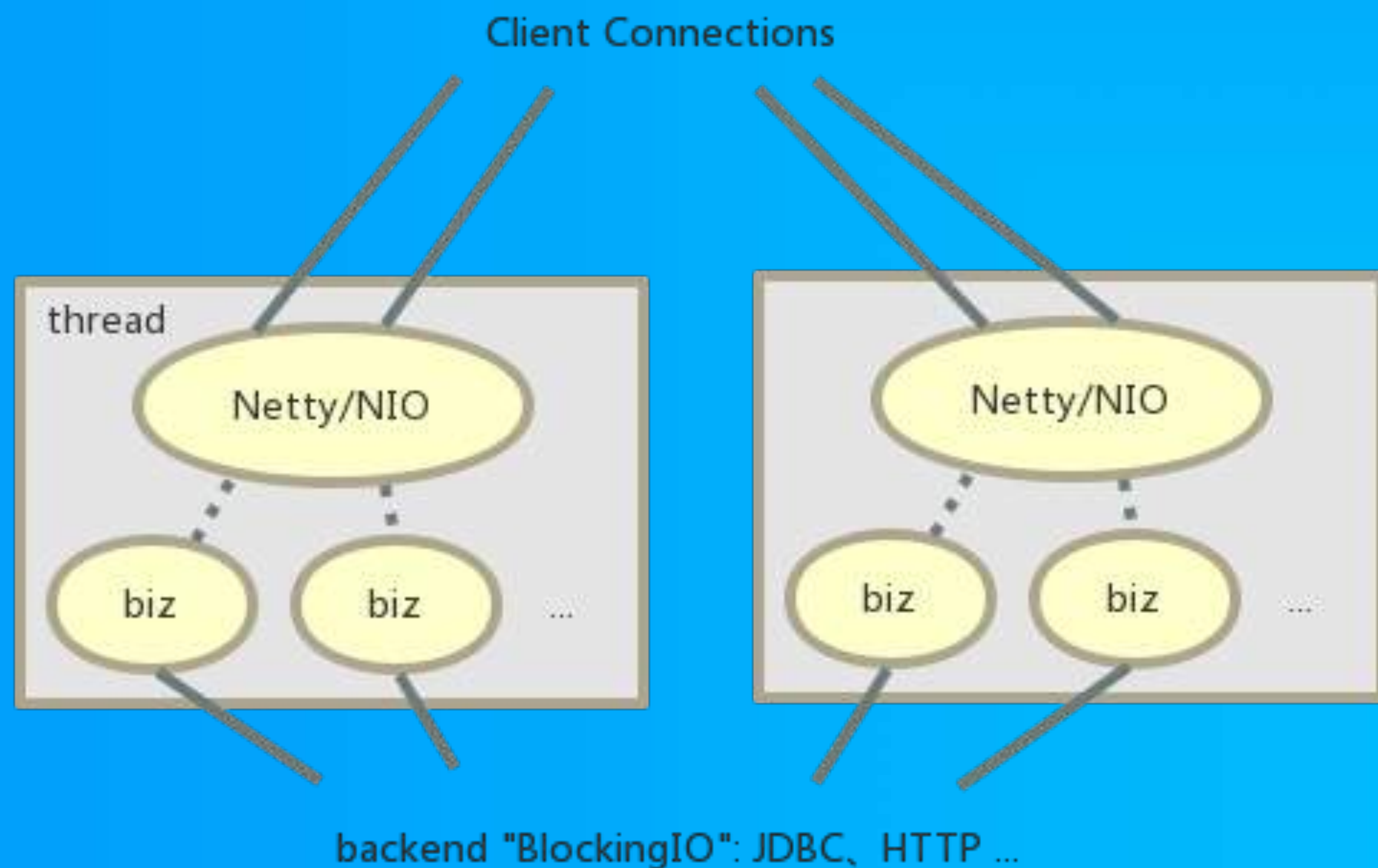
测试/压测

完备的自动化测试
录制流量，7*24压力测试

灰度

经历生产环境的长期考验

简单型应用



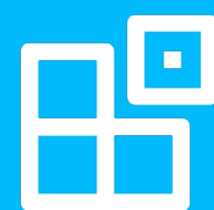
- IO密集型的网络应用
- 处理请求时只是访问数据库、HTTP接口等简单服务
- 比如代理、消息队列等等

极限吞吐量

提升30%

服务RT

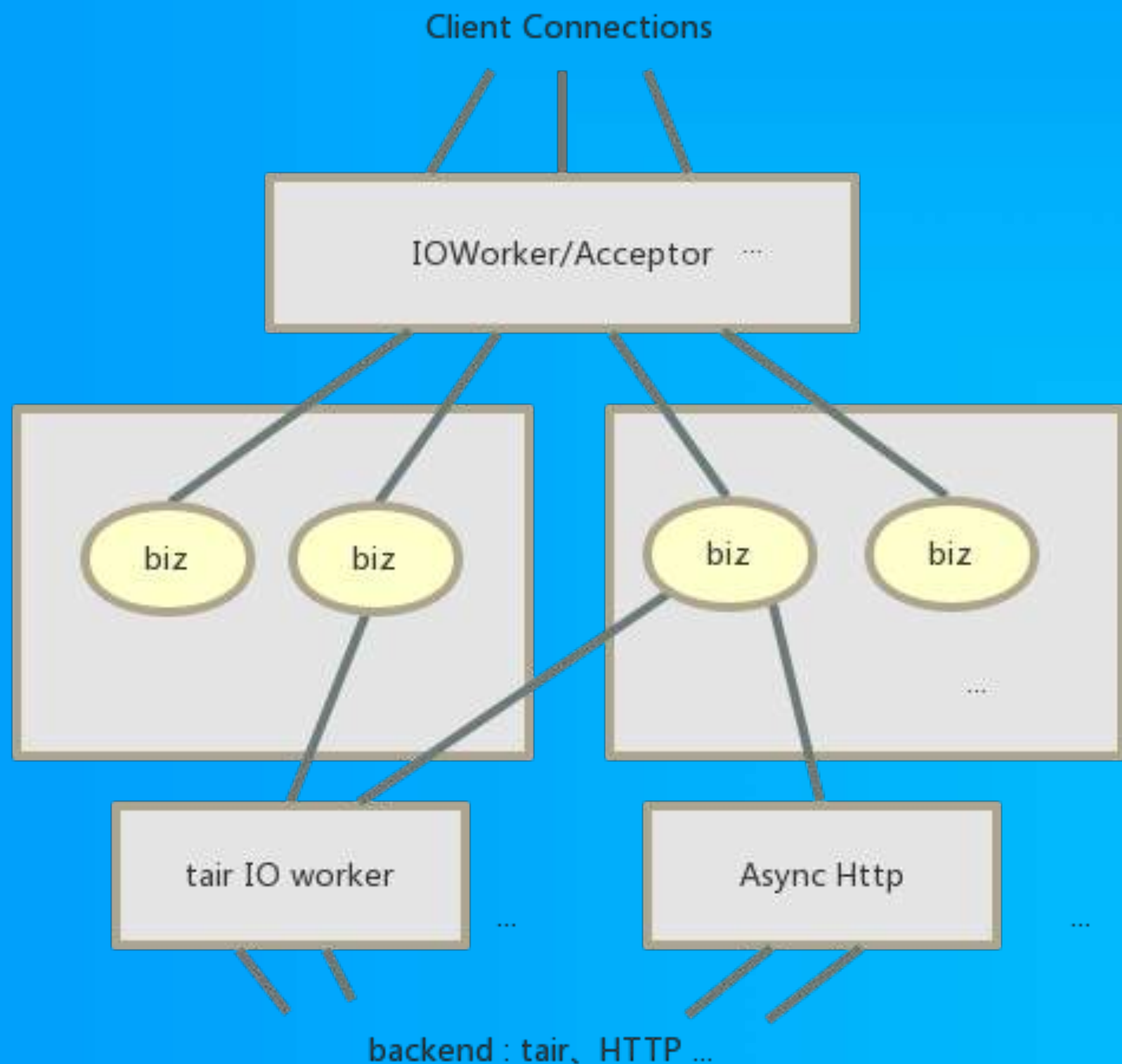
RT减少20%



代码改动

几行代码的改动

大量使用通信中间件的复杂应用



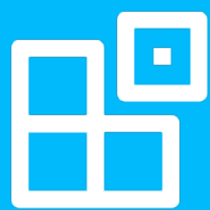
- 可能存在大量的CPU计算(序列化、压缩、加密)，同时又大量RPC
- 通过中间件来访问网络服务，这些中间件往往有自己的IO线程
- 大部分的电商应用都是如此

CPU节约

10%

服务RT

RT增加10%



代码改动

无改动

host	10.185.52.173	host	10.185.53.219
idc	su18	idc	su18
cpu_util	60.41%	cpu_util	54.76%
cpu_user	50.37%	cpu_user	47.00%
cpu_sys	8.70%	cpu_sys	6.36%
cpu_iowait	0.00%	cpu_iowait	0.00%
cpu_hardirq	0.00%	cpu_hardirq	0.00%
cpu_softirq	0.00%	cpu_softirq	0.00%
load_load1	13.28	load_load1	9.71
load_load5	15.51	load_load5	10.46
load_load15	15.93	load_load15	10.68
mem_util	13.16%	mem_util	13.76%
mem_used	7.89G	mem_used	8.26G
mem_buff	0.00B	mem_buff	0.00B
mem_cach	8.74G	mem_cach	10.86G
mem_free	43.36G	mem_free	40.89G
mem_total	0.00B	mem_total	0.00B
traffic_bytin	41.01M	traffic_bytin	41.08M
traffic_bytout	16.07M	traffic_bytout	16.25M
traffic_pktin	25309	traffic_pktin	25517
traffic_pktout	25158	traffic_pktout	25254
traffic_pkterr	0	traffic_pkterr	0
traffic_pktdrp	0	traffic_pktdrp	0
jvm_ygc	54	jvm_ygc	54
jvm_ygc_time	53ms	jvm_ygc_time	21ms
jvm_fgc	0	jvm_fgc	0
jvm_fgc_time	0ms	jvm_fgc_time	0ms

Alibaba Group 阿里巴巴集团

谢谢观看

Thanks

系统软件事业部　打造具备全球竞争力、效率最优的系统软件