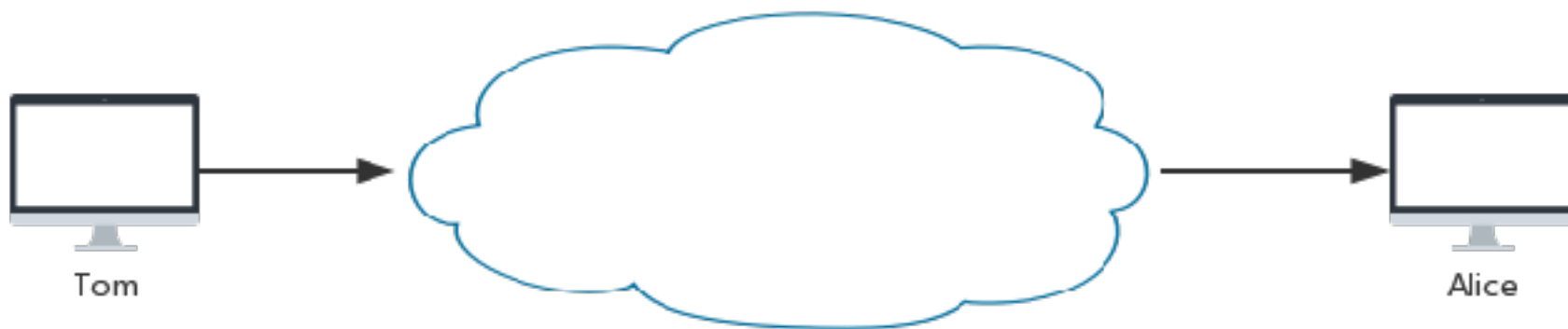


# 美图长连接消息通道

架构平台 任勇全

# 美图长连接消息通道

- Tom 如何将消息发给Alice



bifrost

# 什么是bifrost

- 北欧神话彩虹桥
- 适配多种业务场景
- 保证消息可靠传输

直播IM与传统IM是否具有相同的消息模型？

# 传统IM

- 一对一单聊
- 多对多群聊
- 群不会过大



- 百万人群嗨的聊天室
- 有些消息更重要（比如礼物）？



# 什么是bifrost

- 北欧神话彩虹桥
- 适配多种业务场景
- 保证消息可靠传输

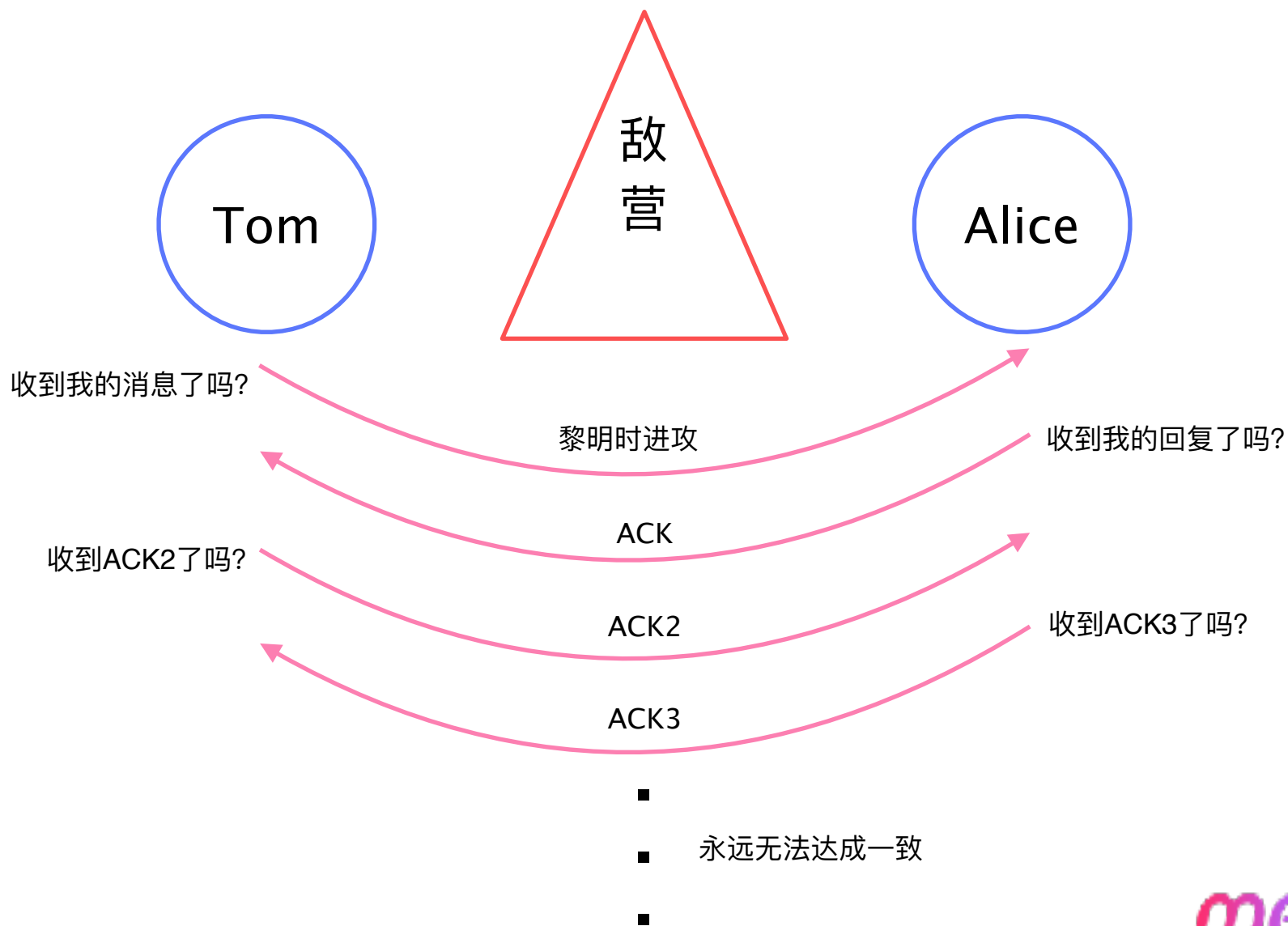


# 美图长连接消息通道

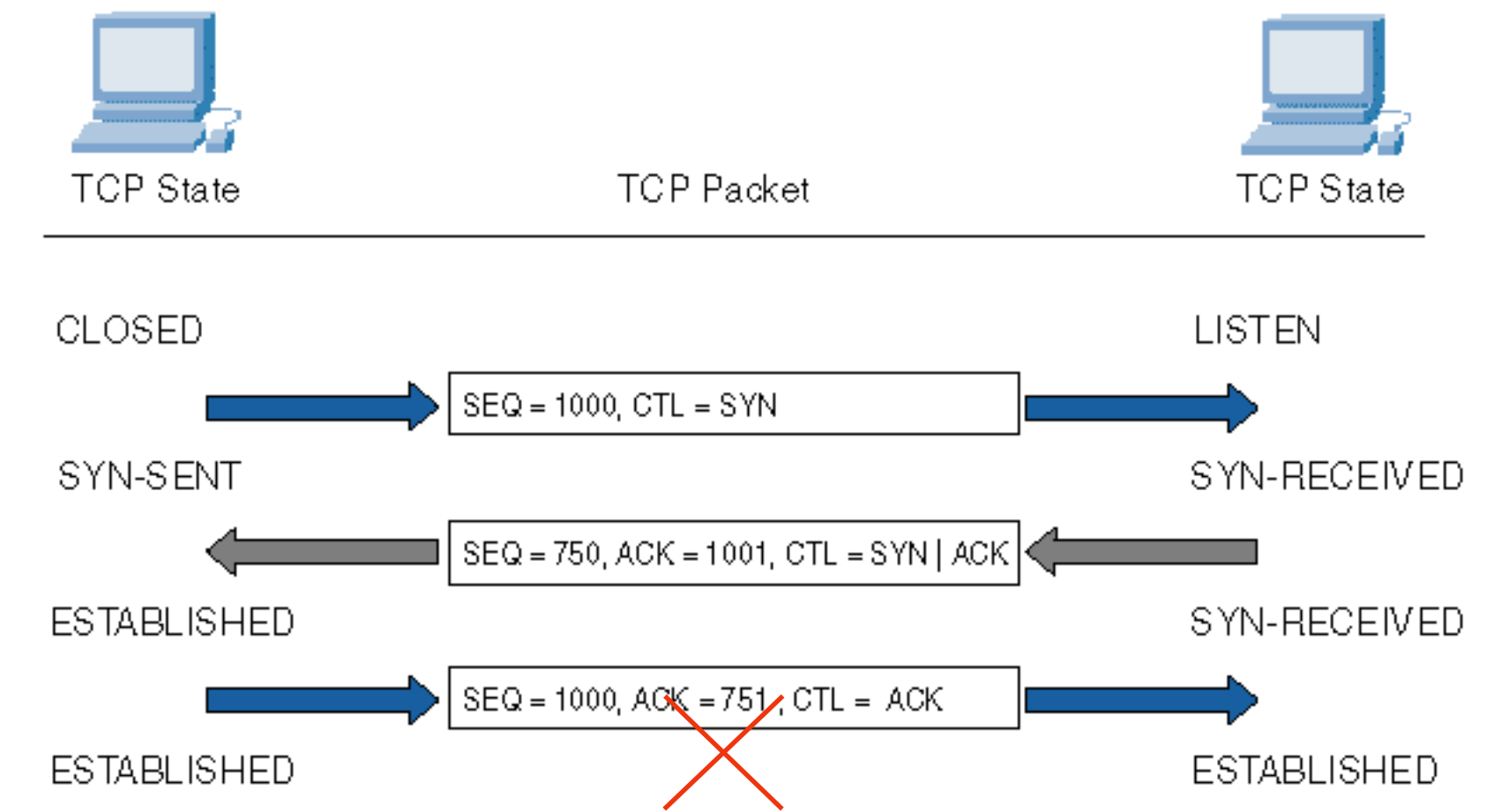


Tom和Alice约定在同一时刻发起进攻，单独进攻的话会攻击失败

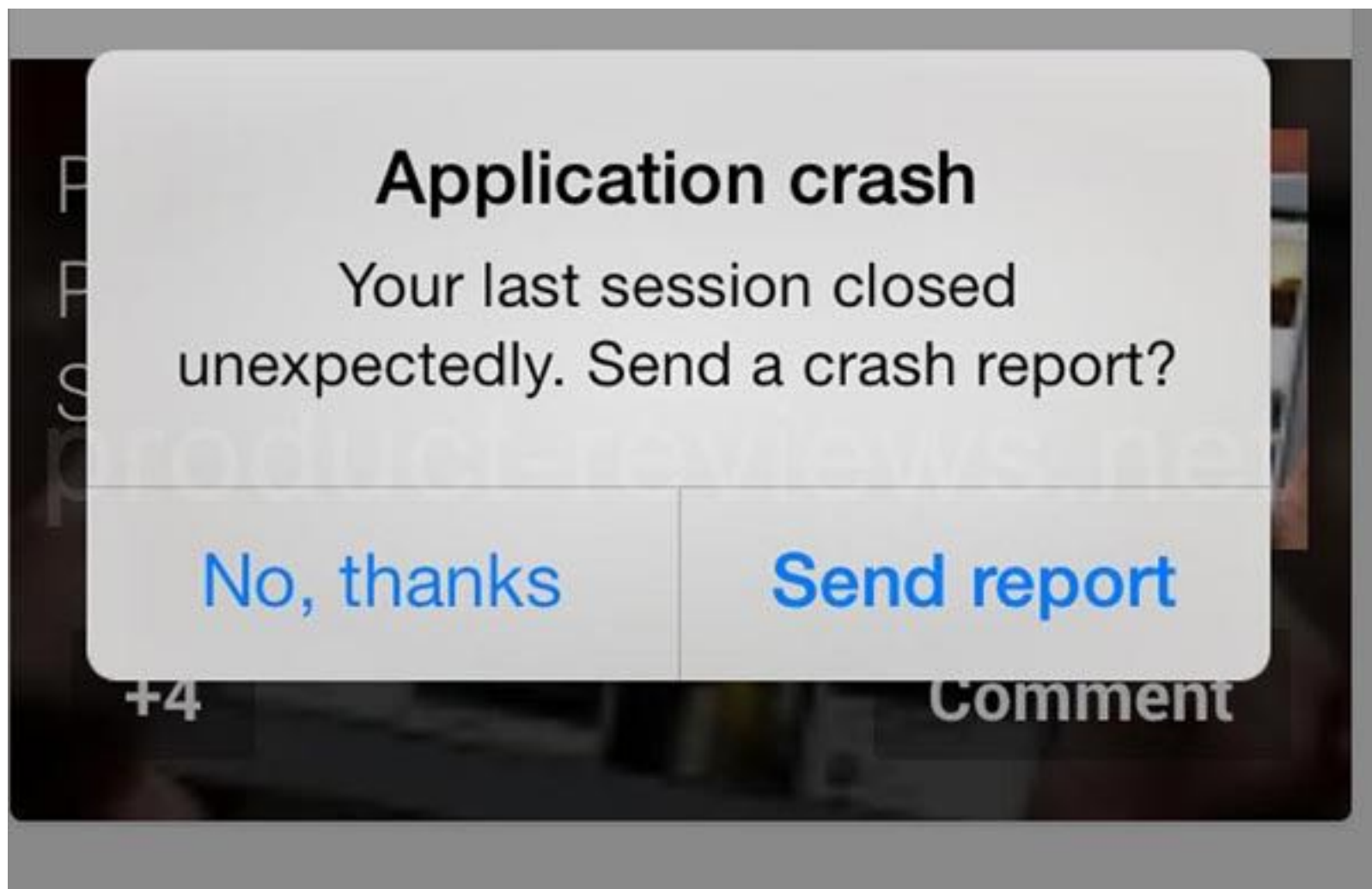
# 两军问题-永远无法达成一致



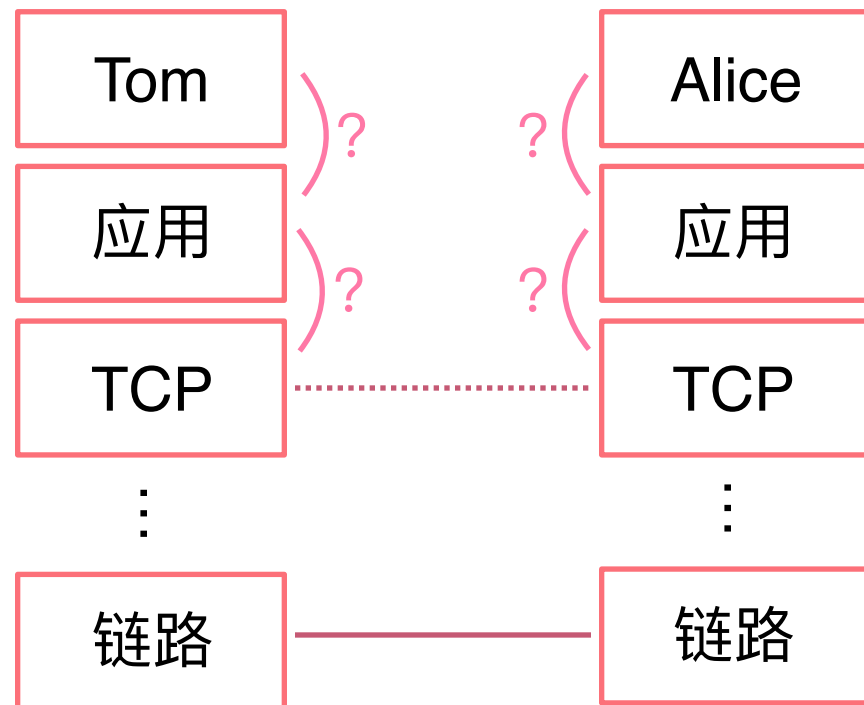
# TCP 状态总会一致吗？



TCP保证传输层可靠还不够吗？



# 应用层可靠



# 应用层可靠



**You**



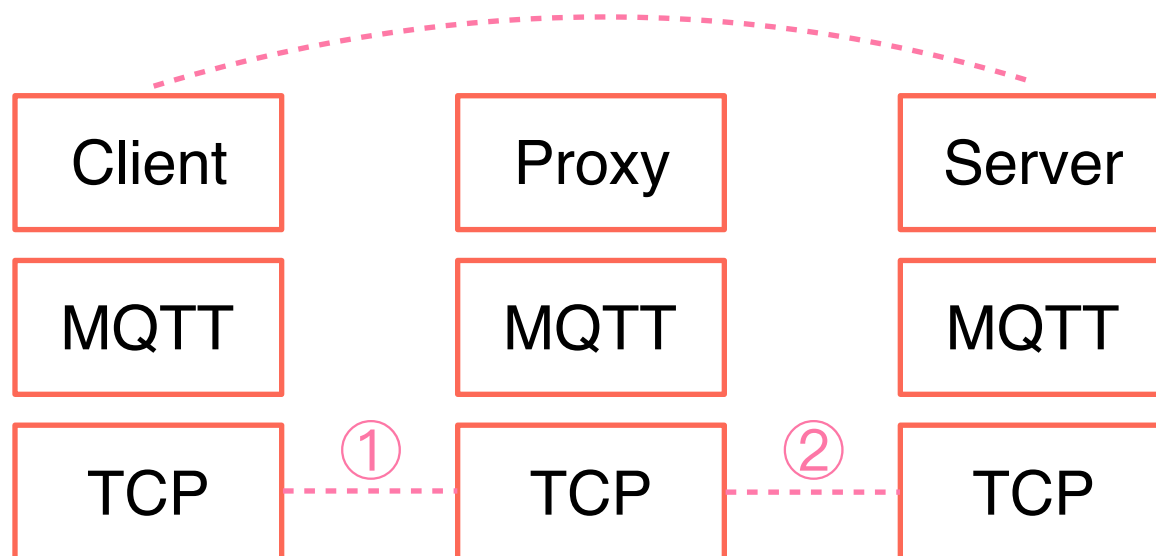
**Proxy**



**Web**

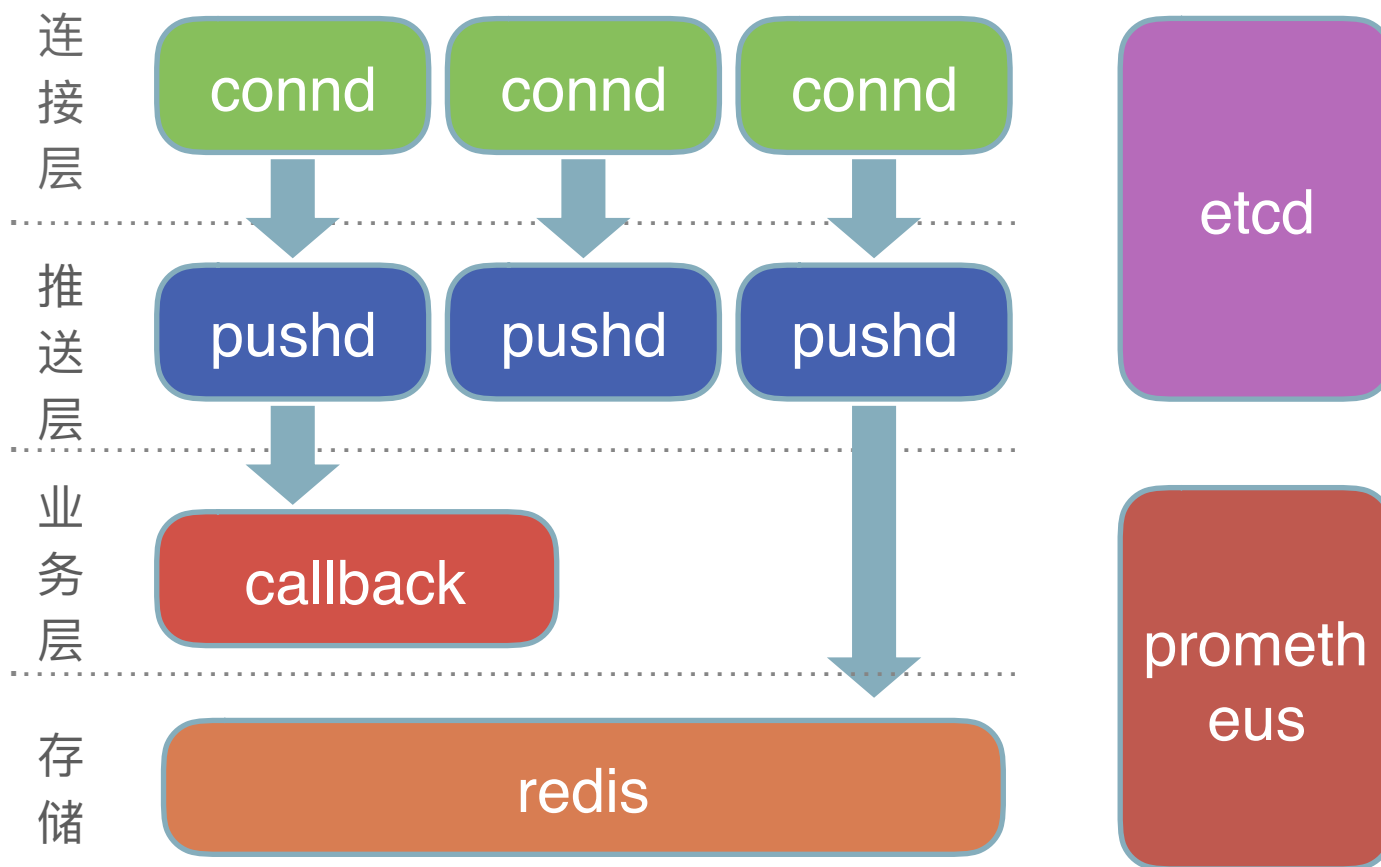


# 应用层可靠



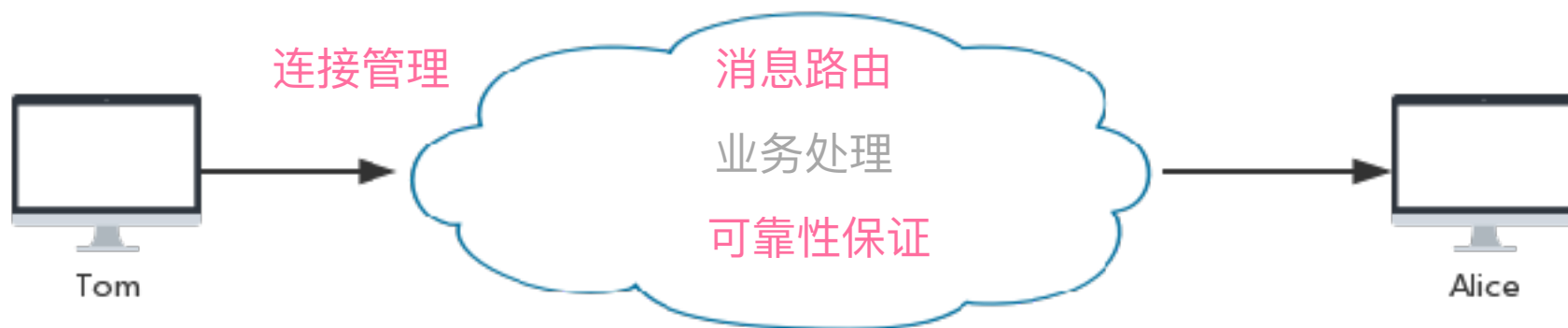
- 物联网传输标准
- 二进制协议，消息格式精简，适合移动设备
- Pub/Sub，适合多种消息传输模型
- 依赖TCP，并支持应用层可靠传输

# bifrost 架构



# 美图长连接消息通道

- Tom 如何将消息发给Alice



# 连接管理的挑战

- 千万级用户连接，并可能会不断增长
- 维护连接状态

# 线程模型

# 并发连接的难题—线程内存占用

- `ulimit -s(8192KB)`
- `PTHREAD_STACK_MIN(16384)`
- `pthread_attr_setstacksize`

# 并发连接的难题—线程内存占用

- **man pthread\_create**

On Linux/x86-32, the default stack size for a new thread is 2 megabytes. Under the NPTL threading implementation, if the RLIMIT\_STACK soft resource limit at the time the program started has any value other than "unlimited", then it determines the default stack size of new threads.



# 并发连接的难题—线程内存占用

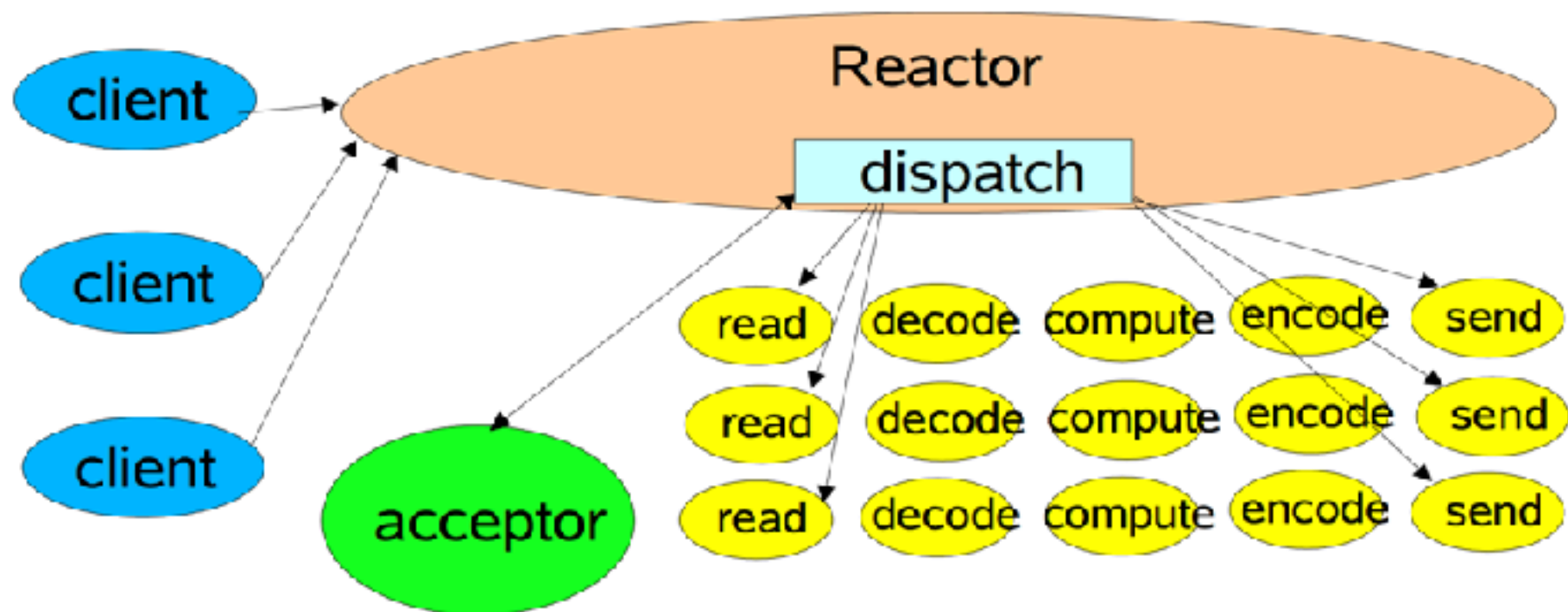
- `man pthread_setstacksize`  
    `pthread_attr_setstacksize()` can fail with the following error:  
    EINVAL The stack size is less than  
    PTHREAD\_STACK\_MIN (16384) bytes

# 并发连接的难题—线程上下文切换

The #1 cause of context switches is having more active threads than you have processors. As the ratio of active threads to processors increases, the number of context switches also increases – linearly if you're lucky, but often exponentially.

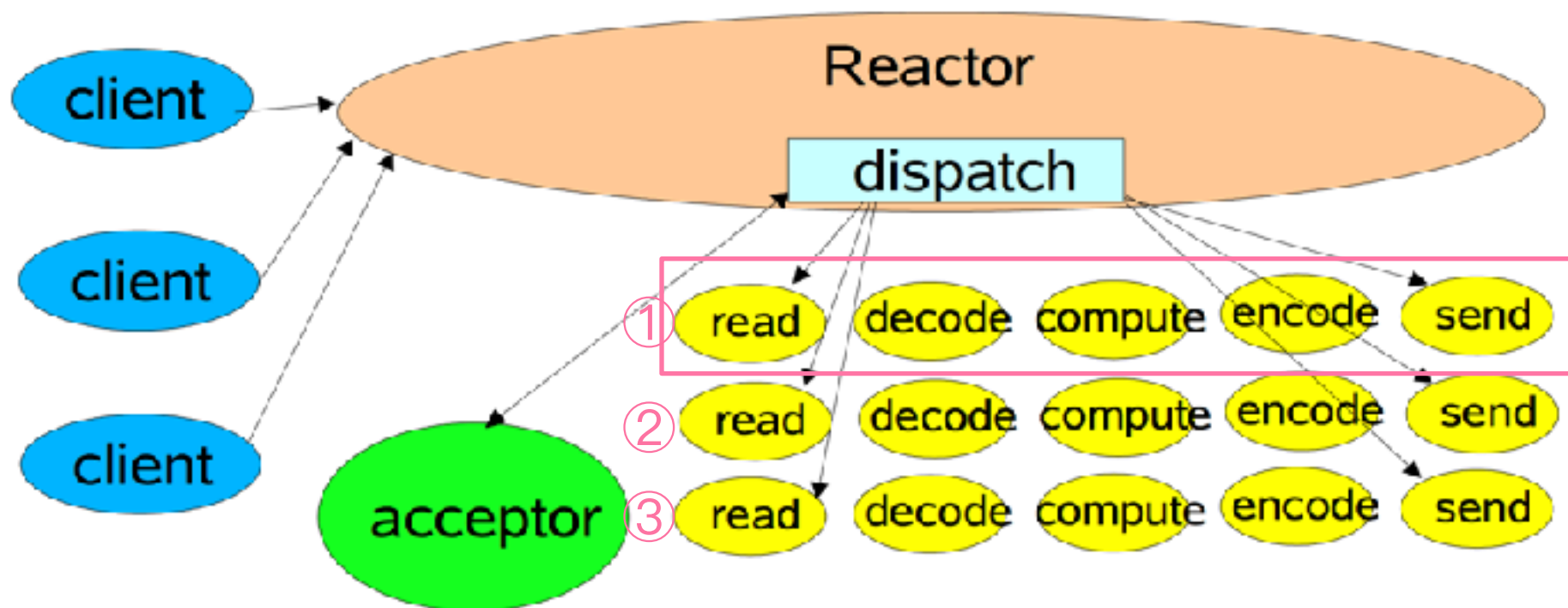
# 事件模型

# 并发连接的难题—select, poll, epoll



# 调度模型

# Golang的并发—抽象执行路径



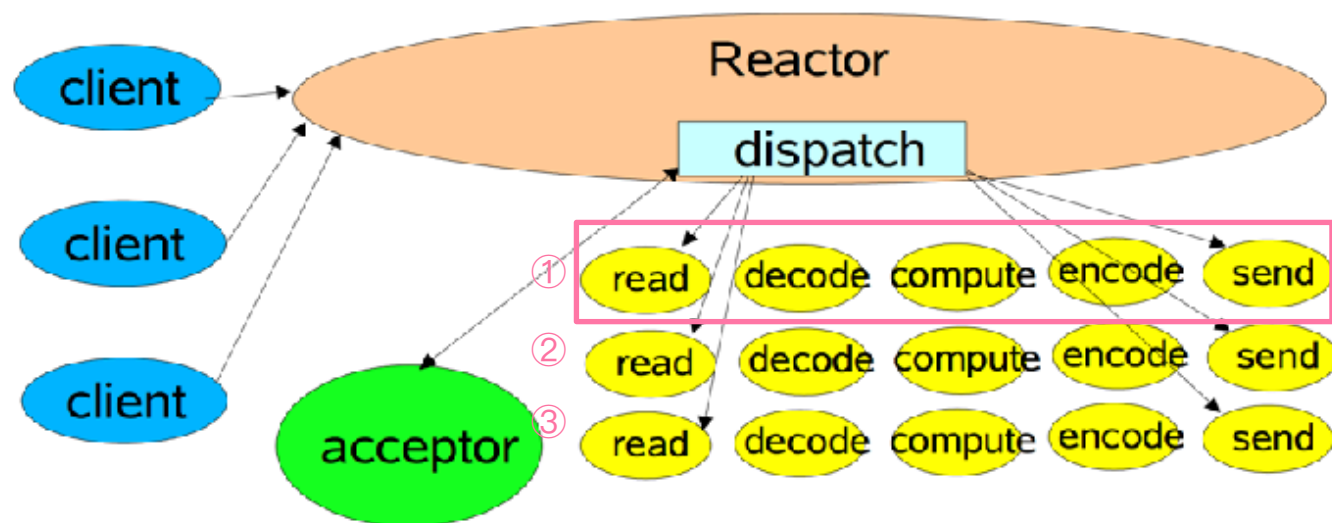
# Golang 的并发—goroutine 调度时机

网络IO

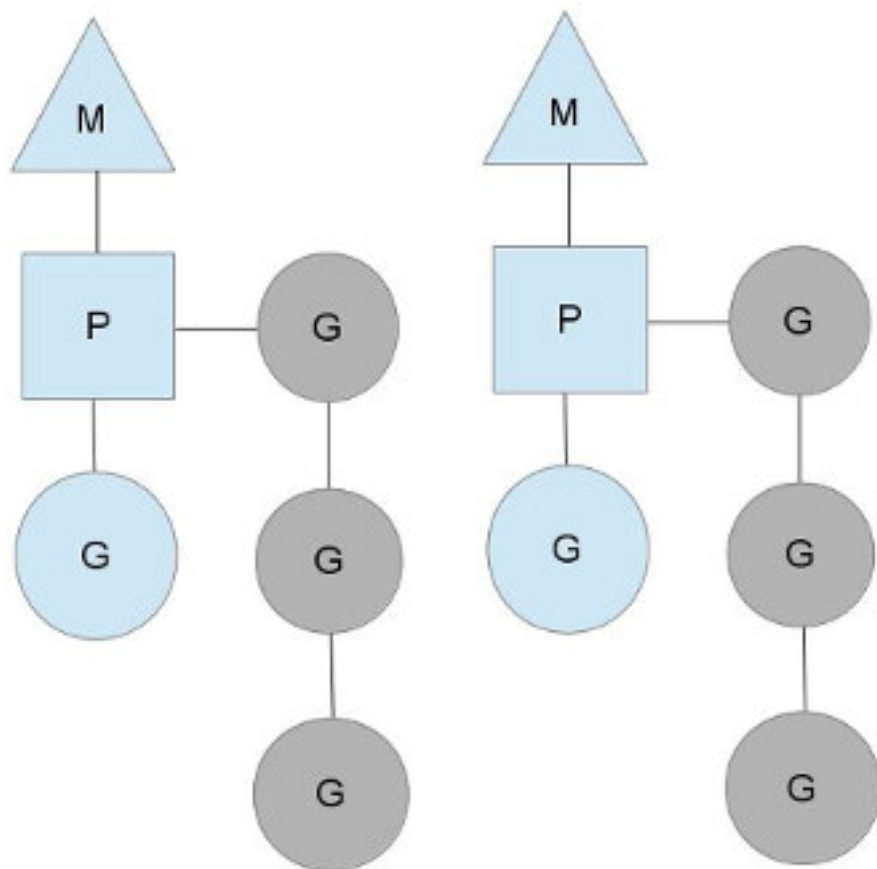
系统调用

channel读写

抢占

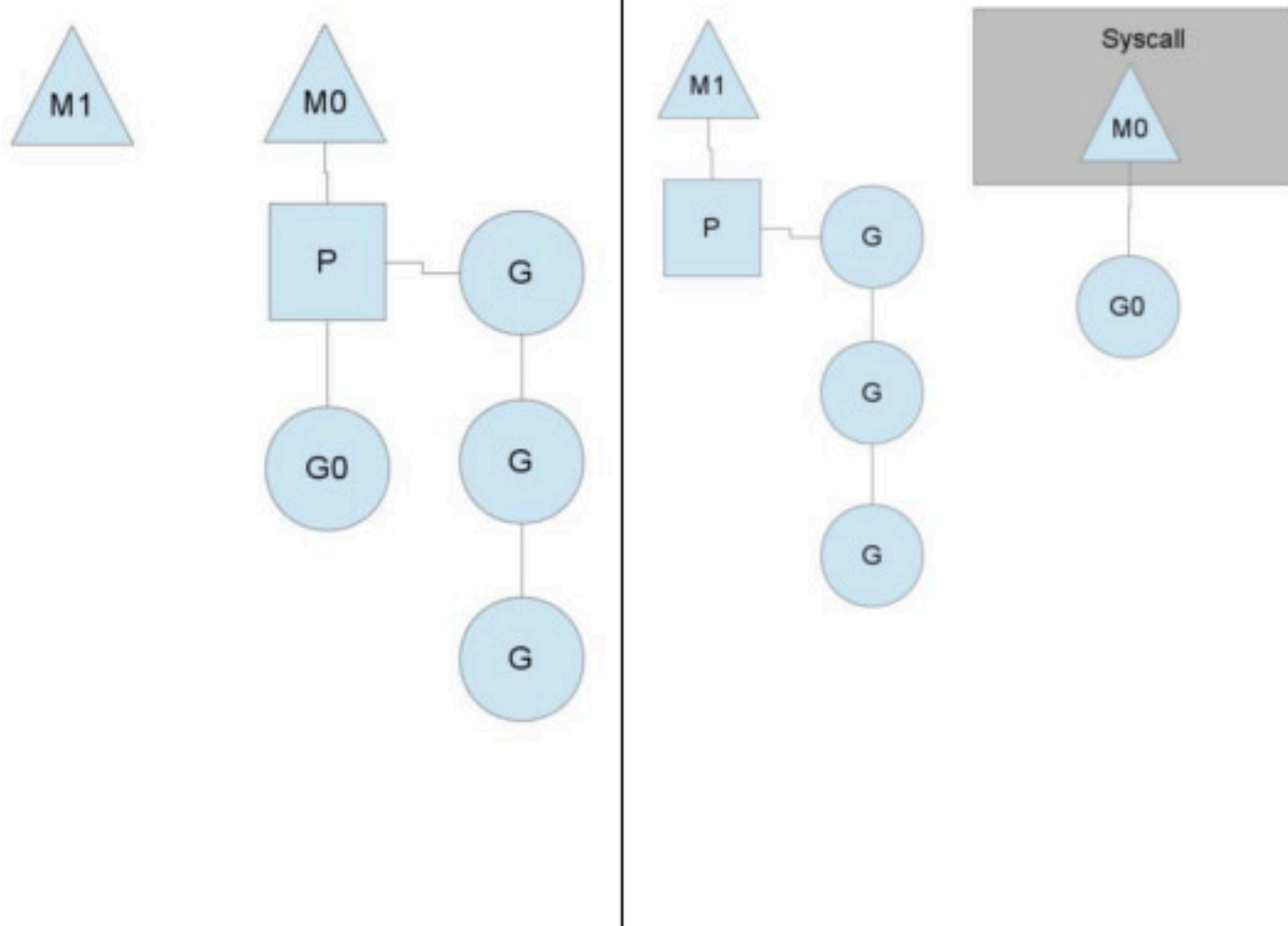


# Golang的并发—goroutine 调度





# Golang的并发—goroutine 调度—syscall



# Golang的并发—goroutine

- 用户态线程，调度更轻量
- 动态Stack大小，最小2KB，Stack无溢出风险

连接是两端互相拥有对方的识别信息及关联数据

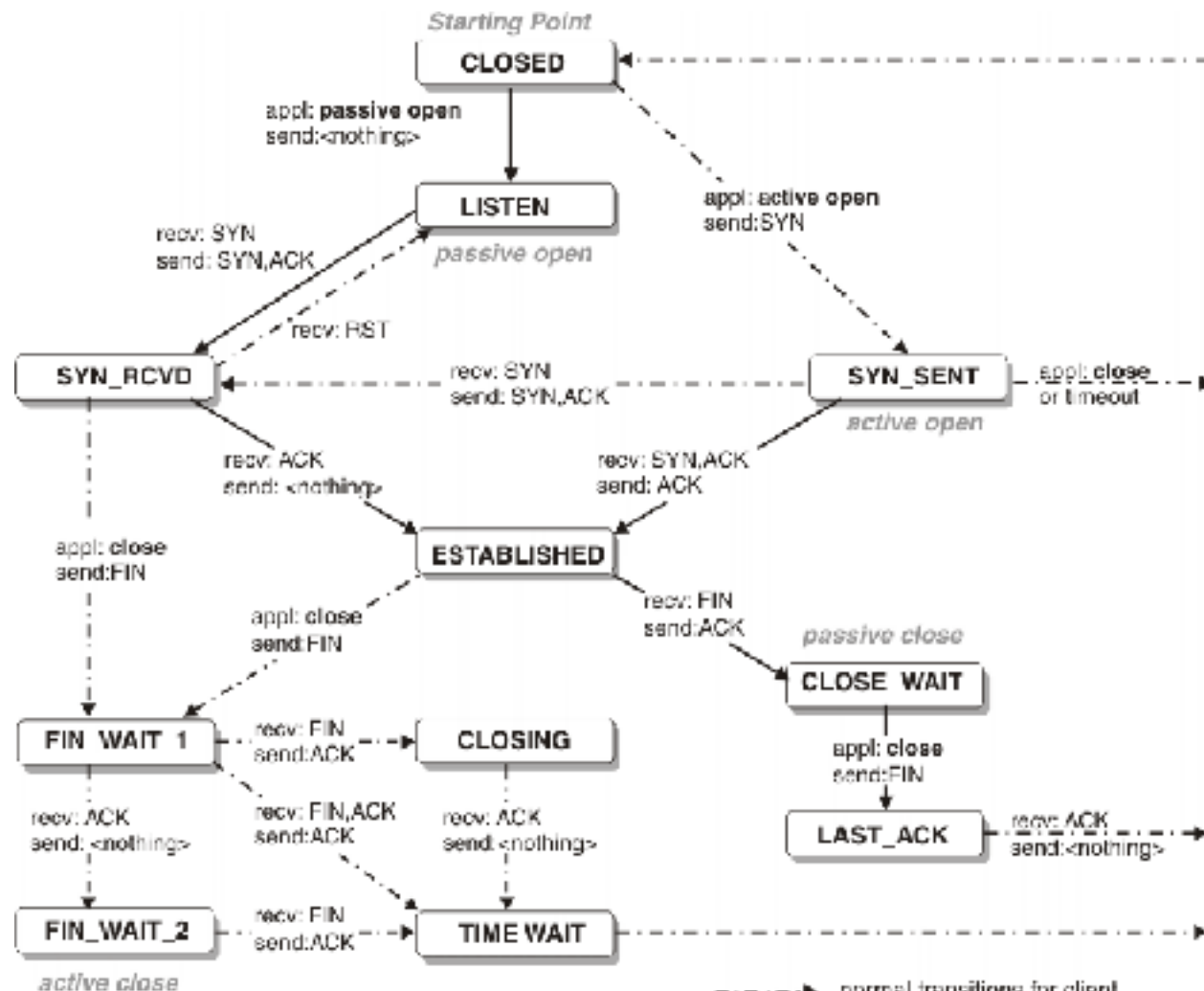
# 连接、状态与路由

	STATE	LOCAL ADDRESS	LOCAL PORT	REMOTE ADDRESS	REMOTE PORT
Connection 1					
Connection 2					
Connection 3					
Connection n					

In information technology and computer science, a program is described as stateful if it is designed to remember preceding events or user interactions; the remembered information is called the state of the system.

—Wikipedia

# 连接状态与路由

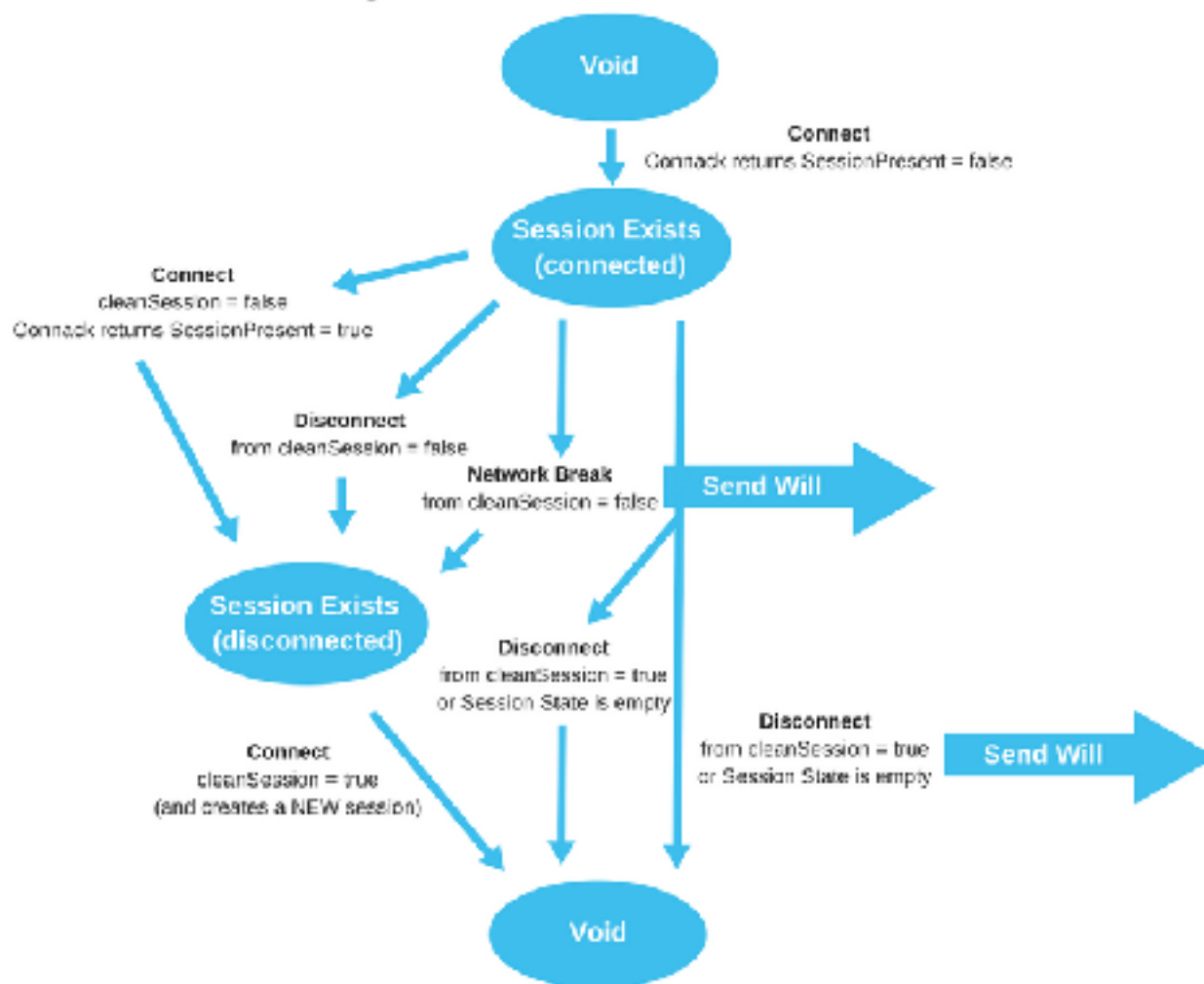


---> normal transitions for client  
-> normal transitions for server

appl: state transition taken when appl. issues operation  
recv: state transition taken when segment is received  
send: what is sent for this transition

# 连接状态与路由

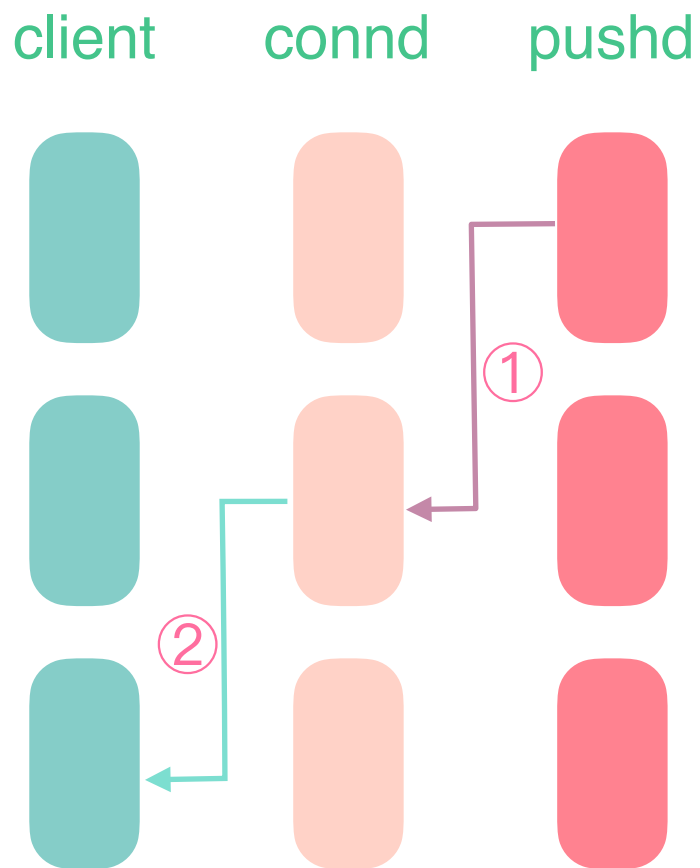
## MQTT V3.1.1 State Transition



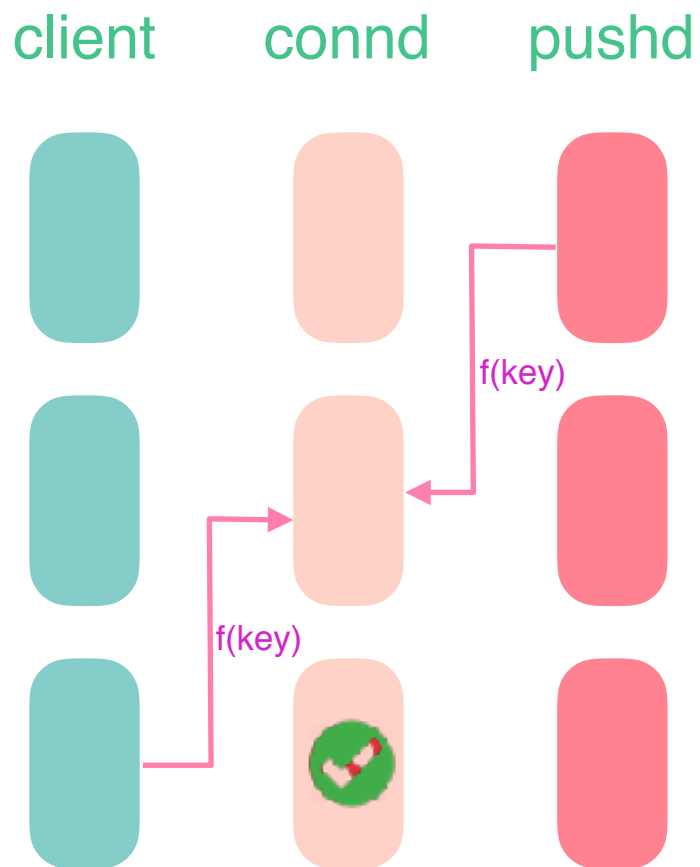
Bifrost 基于状态信息进行消息路由



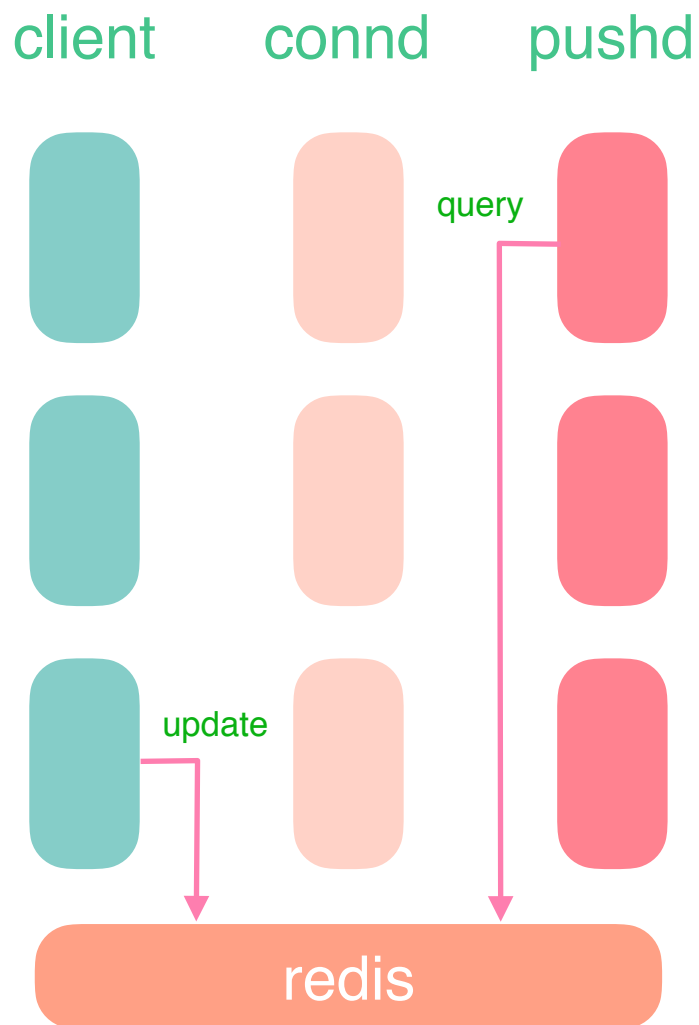
# 连接状态与路由



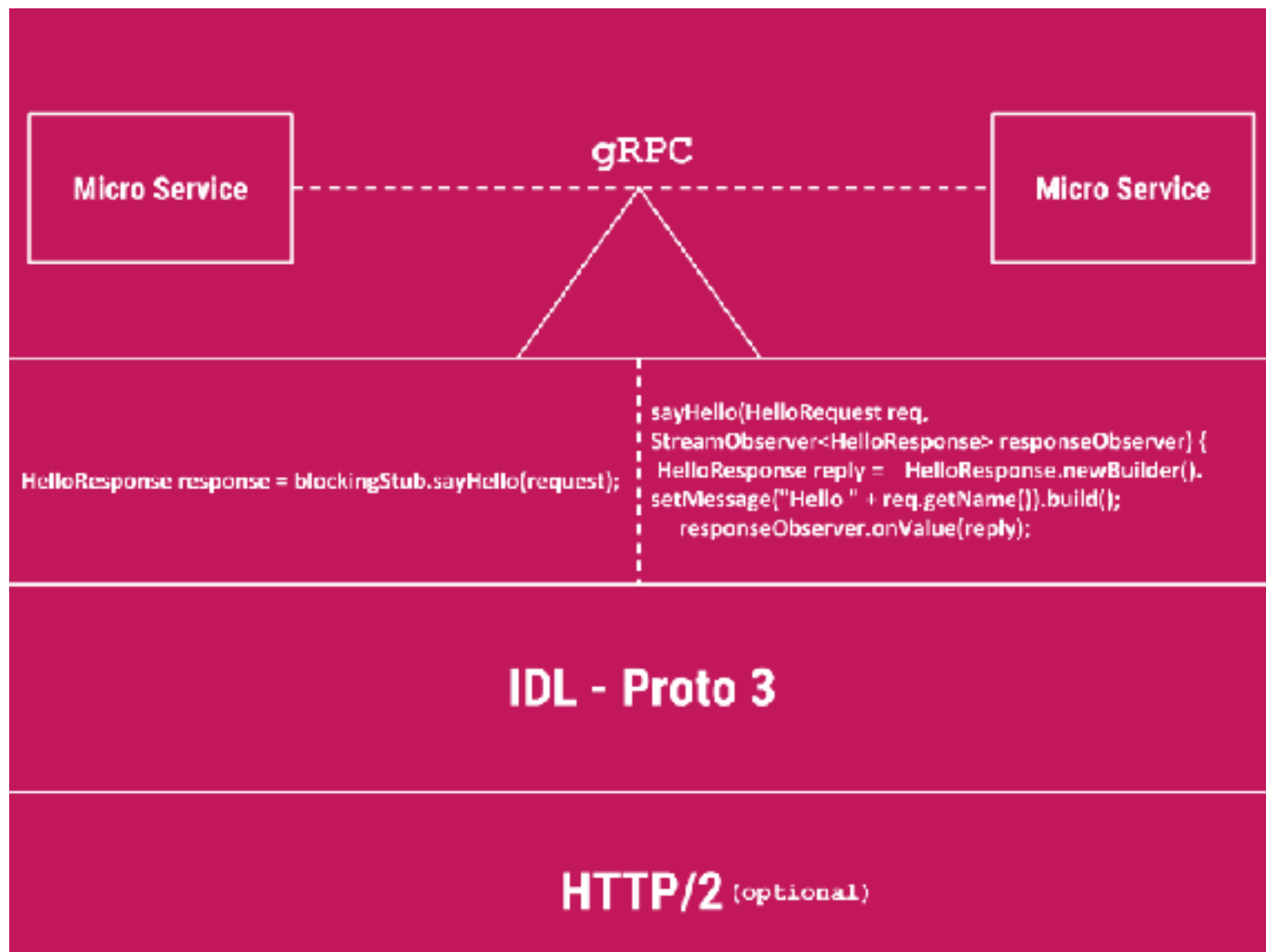
# 连接状态与路由—算法约定



# 连接状态与路由—外部存储

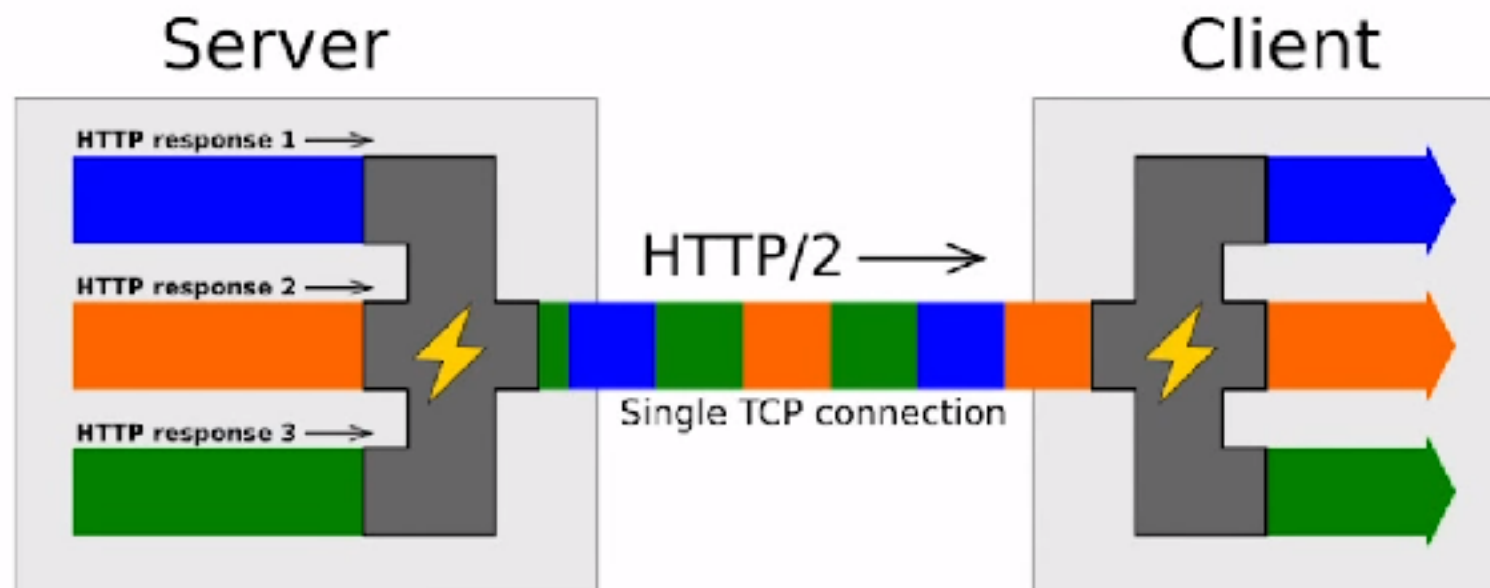


# 分层间通信



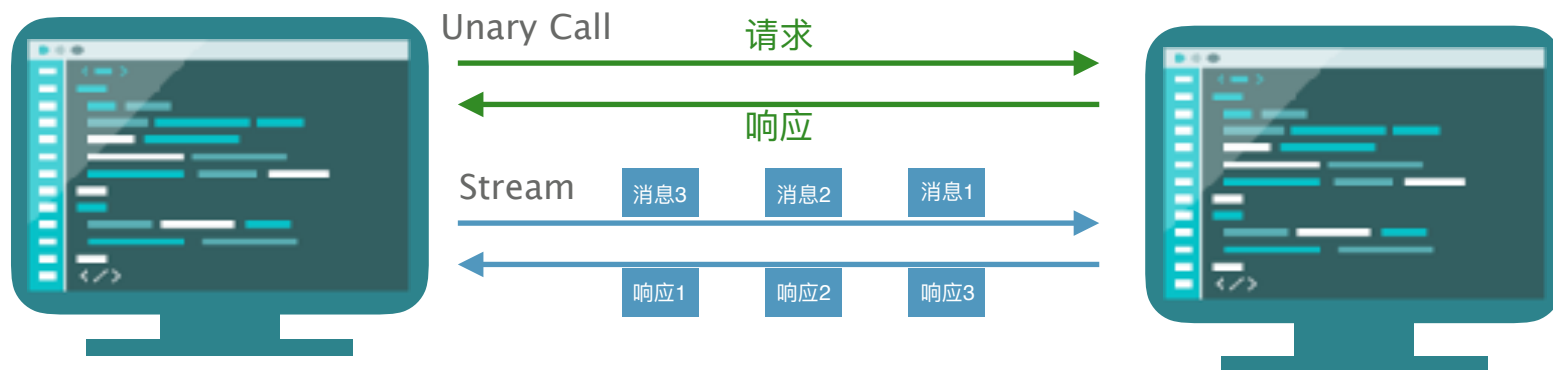
# 分层间通信—http/2 多路复用

## HTTP/2 Inside: multiplexing

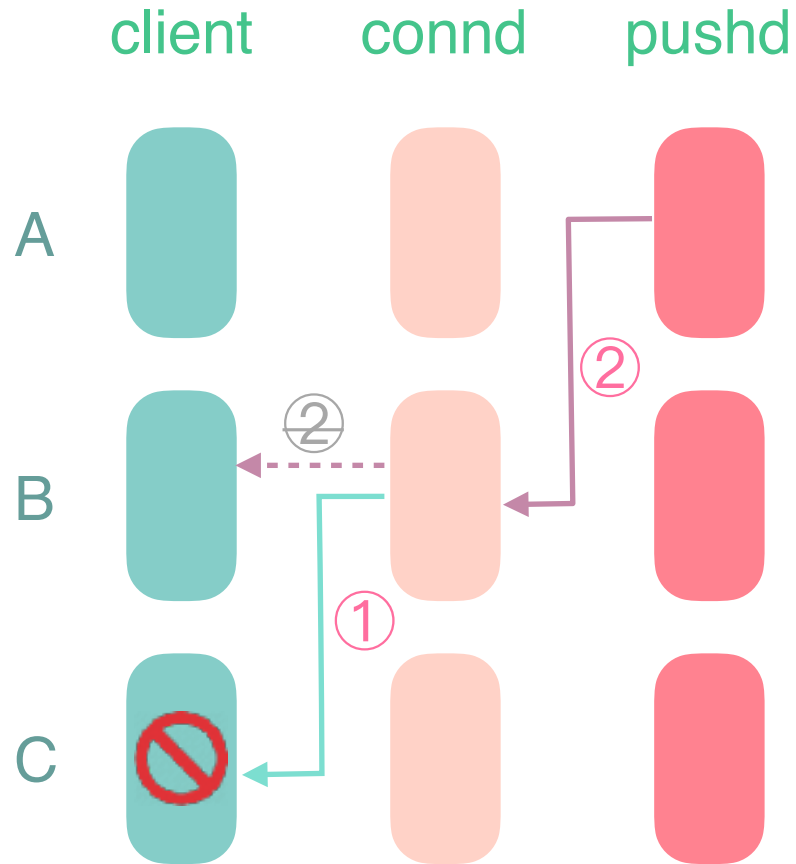


# 分层间通信—gRPC

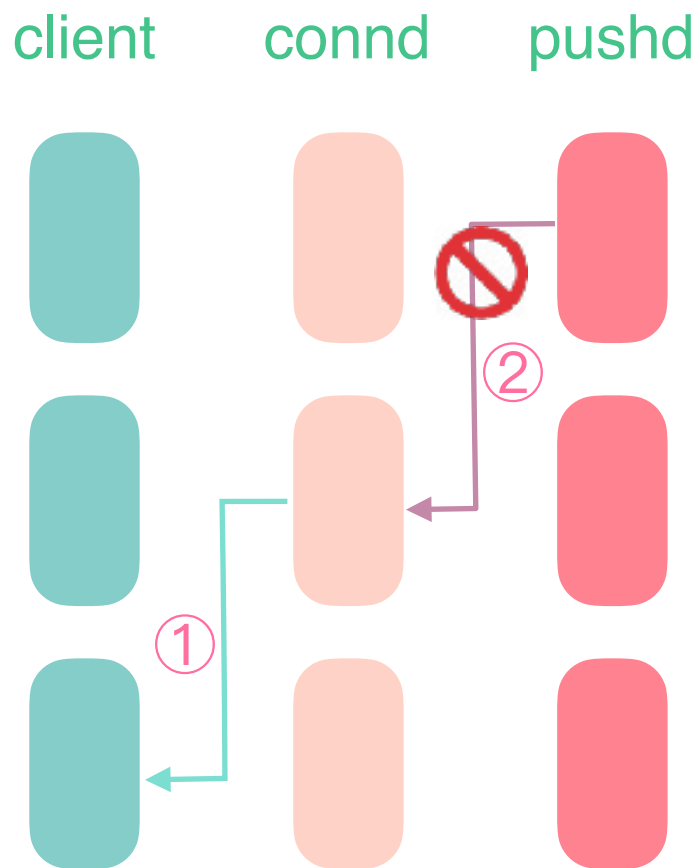
## Stream or Unary call ?



# Stream的难题—Head of line blocking



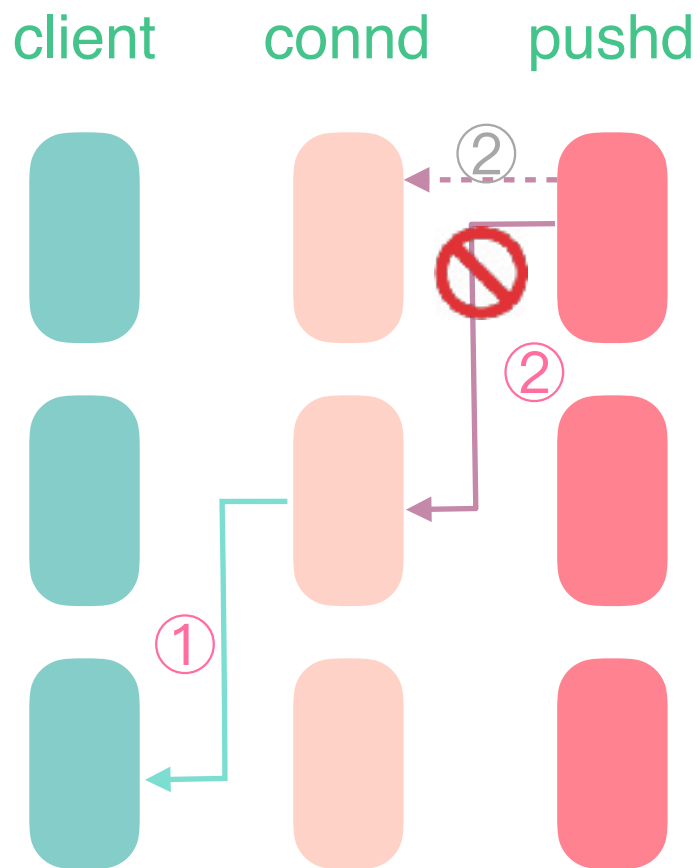
# Stream的难题—消息可靠



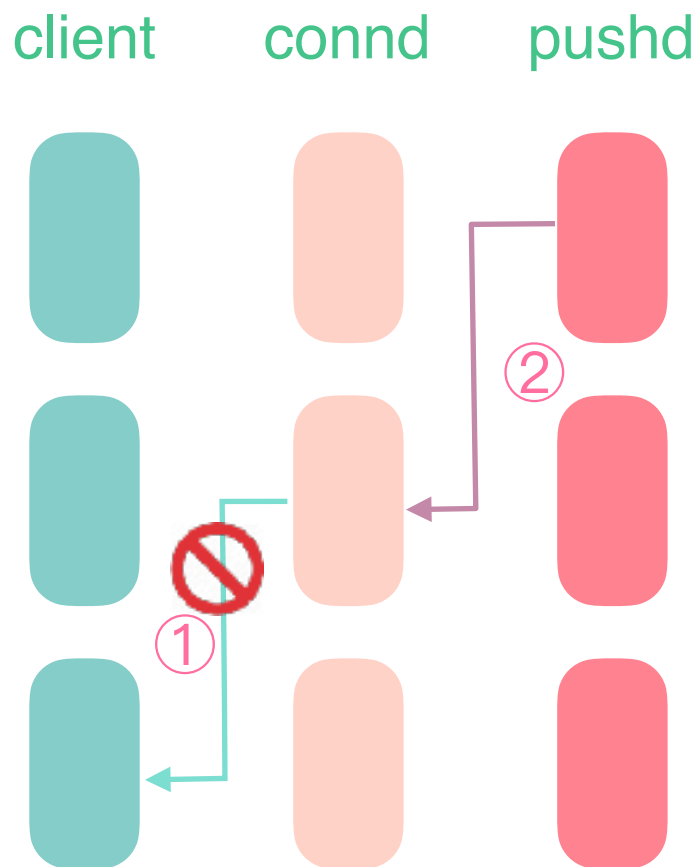


- Stream
  - 较高的传输性能
  - 单Stream容易造成HOL blocking
  - 消息可靠性实现困难
- Unary call
  - 性能比Stream差，但够用
  - 无HOL blocking
  - 重试或故障转移保证消息可靠

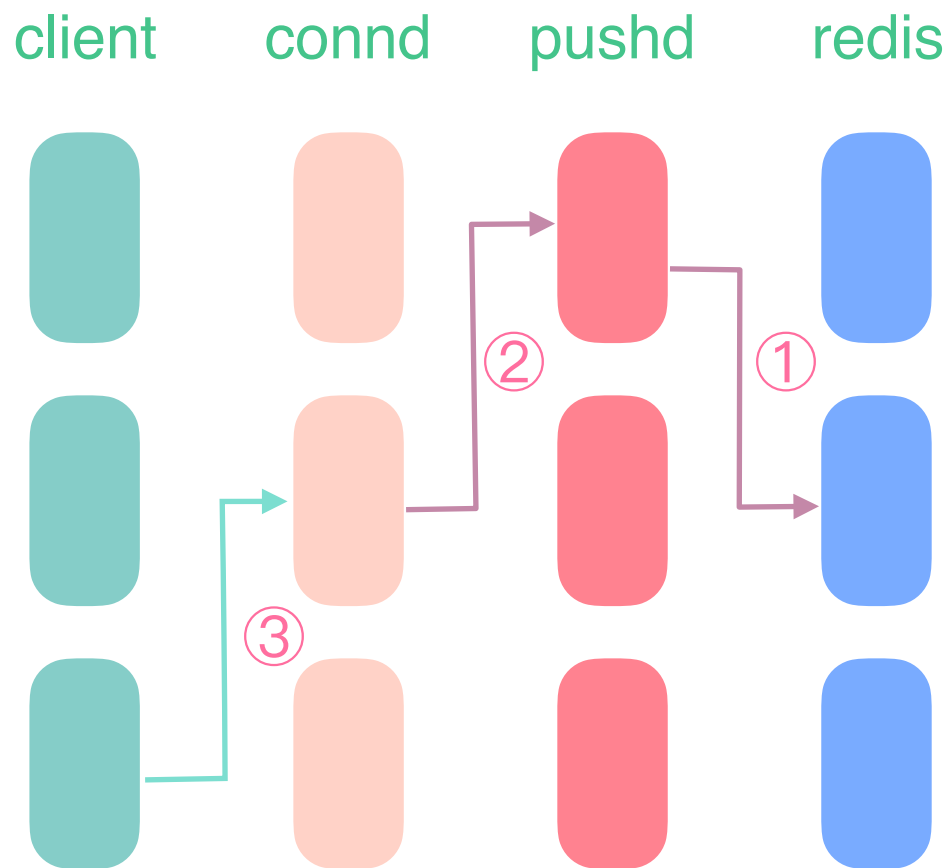
# 可靠性保证—pushd 到 connd



# 可靠性保证—connd 到 client



# 可靠性保证—持久化到redis



# 可靠性保证—持久化到redis

key-hash

message sent



message acked



message unacked

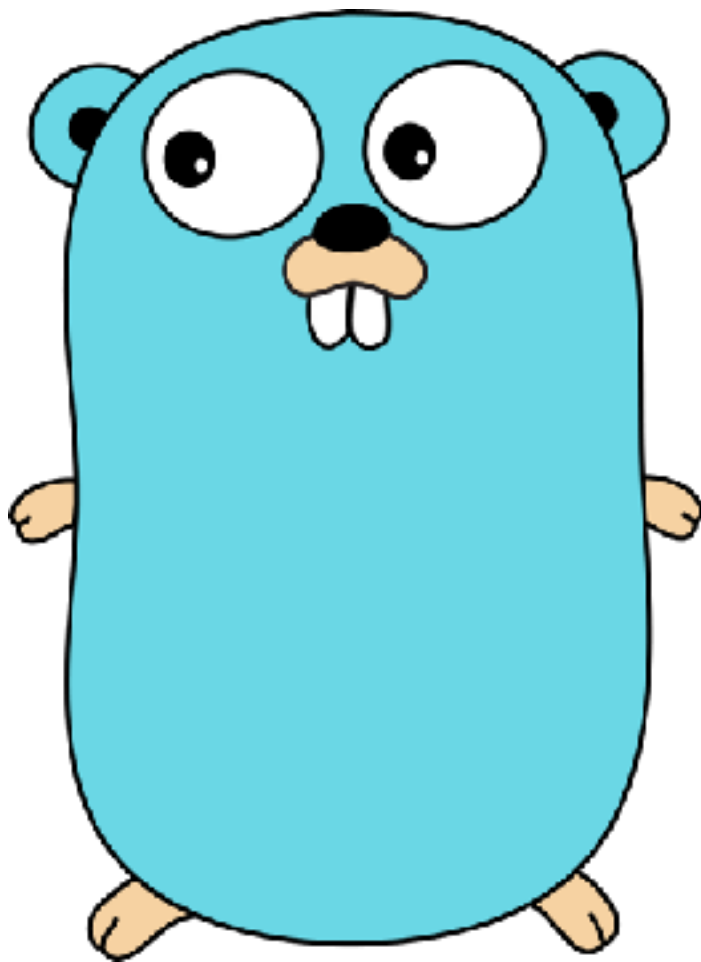


key-hash

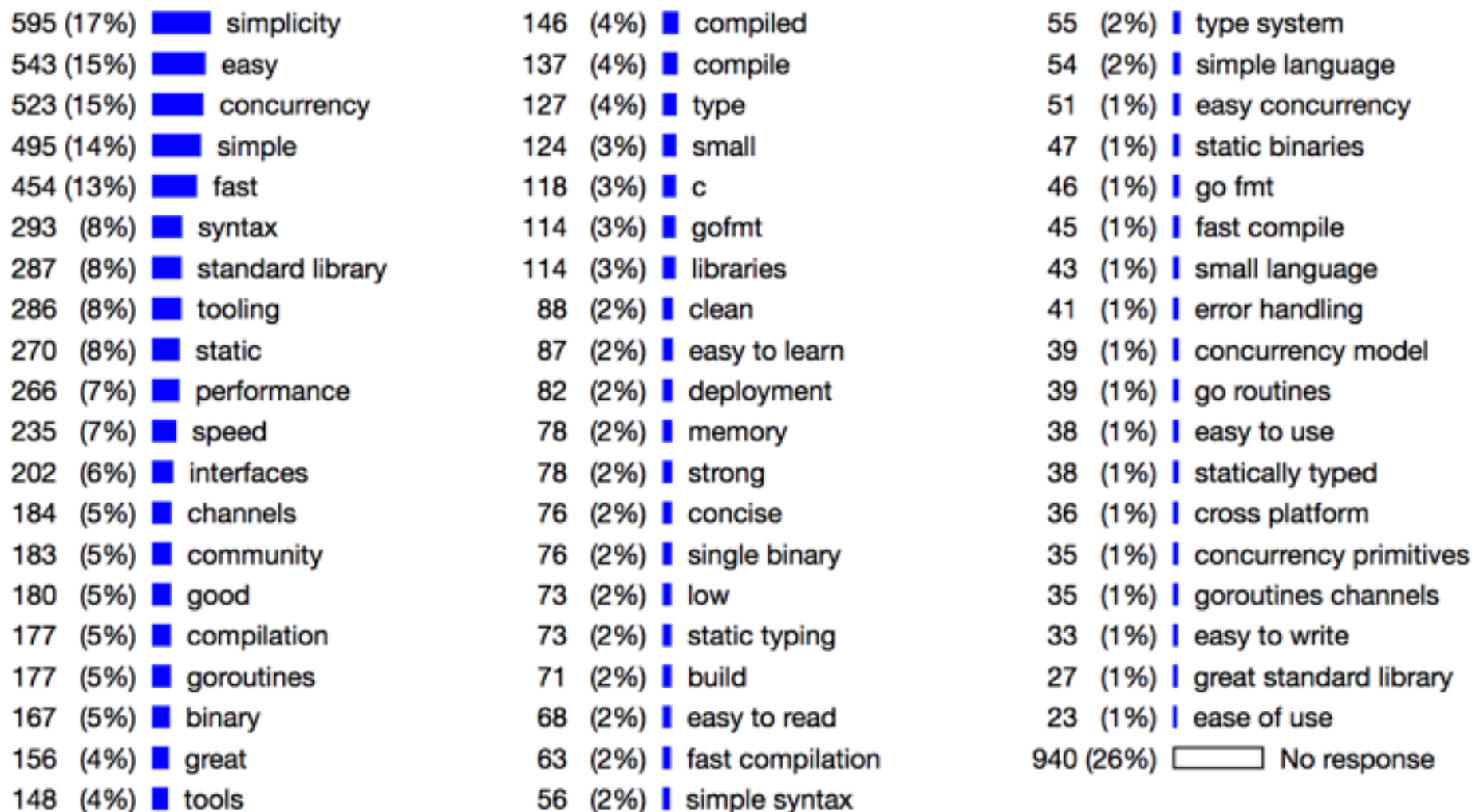
# 可靠性保证—持久化到redis

- 原子操作
  - multi-exec
  - lua脚本
- Key-hash 查询性能
  - hgetall
  - hscan
  - 降级为广播，避免查询

为什么是Go?



# Golang—简单





# Golang—简单

## Simplicity

Number of keywords is an approximate measure of complexity

C (K&R)	K&R	32
C++	1991	48
Java	3rd edition	50
C#	2010	77
C++0x	2010	72+11*
JavaScript	ECMA-262	26+16*
Python	2.7	31
Pascal	ISO	35
Modula-2	1980	40
Oberon	1990	32
Go	2010	25

# Golang—简单

## Hello, world 2.0

Serving `http://localhost:8080/world:`

```
package main

import (
    "fmt"
    "http"
)

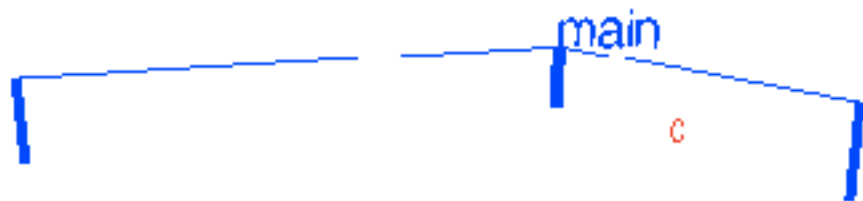
func handler(c *http.Conn, r *http.Request) {
    fmt.Fprintf(c, "Hello, %s.", r.URL.Path[1:])
}

func main() {
    http.ListenAndServe(":8080",
        http.HandlerFunc(handler))
}
```

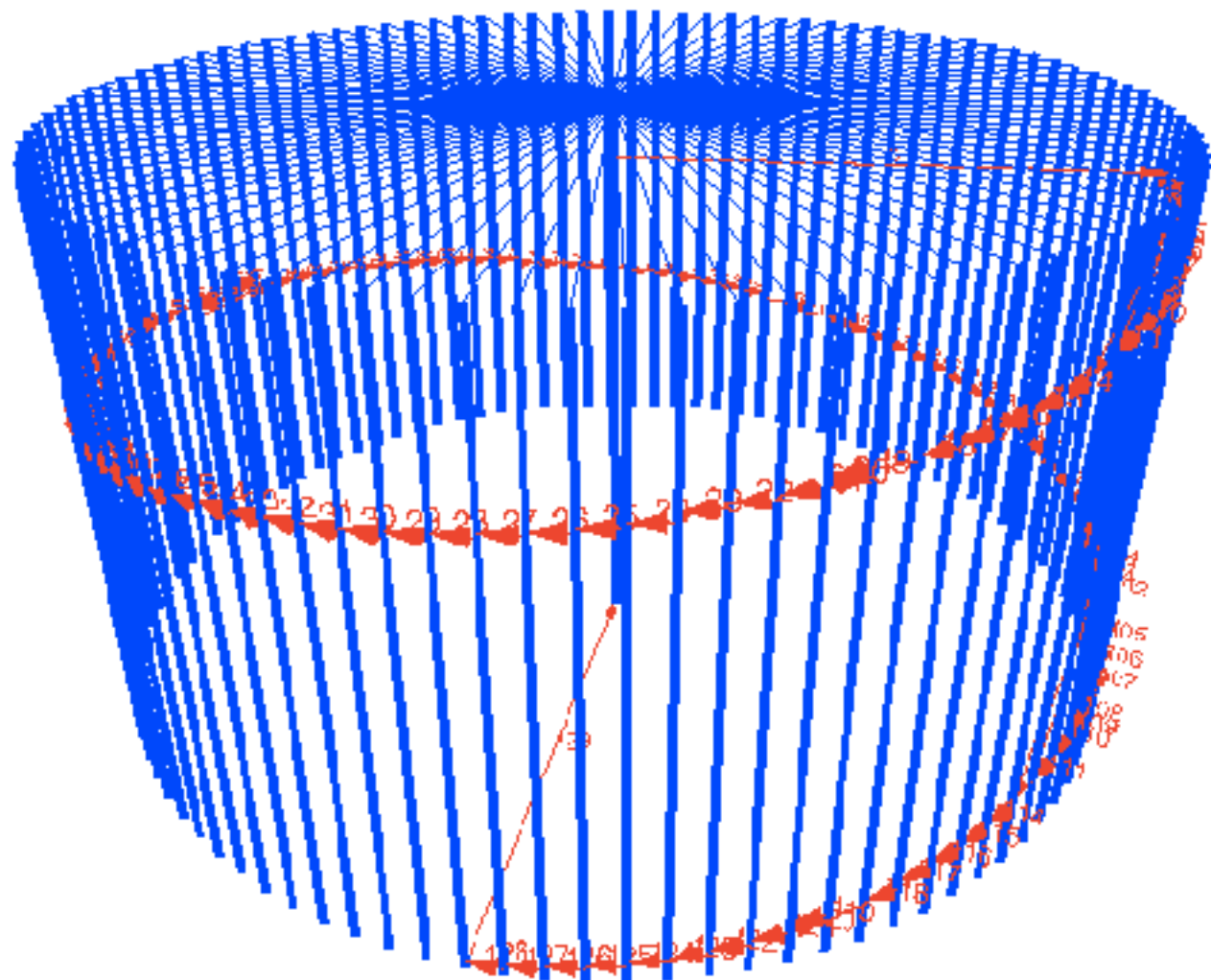
# Golang—并发

```
1  package main
2
3  import "time"
4
5  func main() {
6      var Ball int
7      table := make(chan int)
8      go player(table)
9      go player(table)
10
11      table <- Ball
12      time.Sleep(1 * time.Second)
13      <-table
14  }
15
16  func player(table chan int) {
17      for {
18          ball := <-table
19          ball++
20          time.Sleep(100 * time.Millisecond)
21          table <- ball
22      }
23  }
```

# Golang—并发



# Golang—并发





# 美图长连接消息通道



# 美图长连接消息通道

