Make RPC Faster than Fast

A detailed look at the performance optimization journey of Kitex to improve RPC speed.

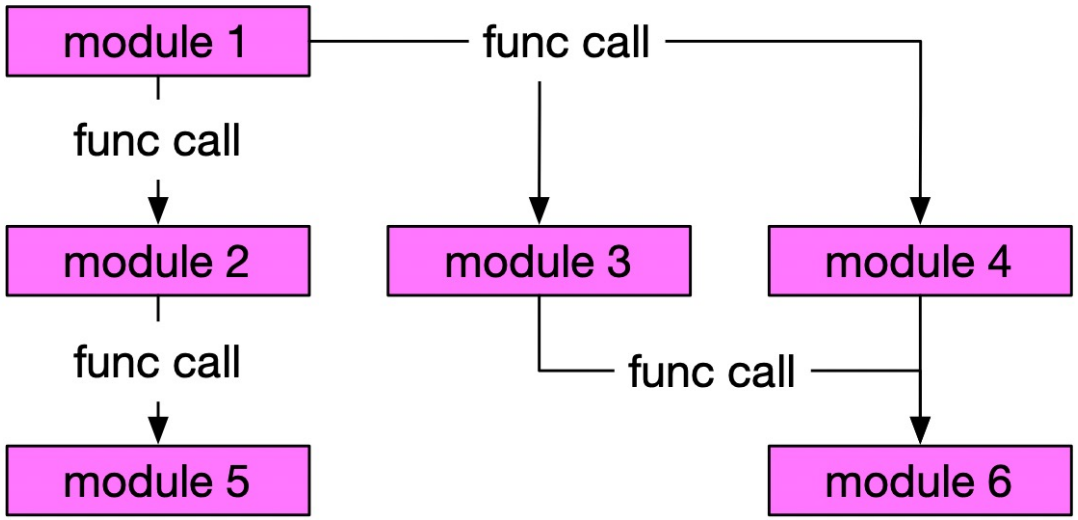
by Zhuowei Wang
CloudWeGo Kitex&Netpoll Maintainer

Content
目录

Part 01

The Costs of Microservices

Monolithic Architecture

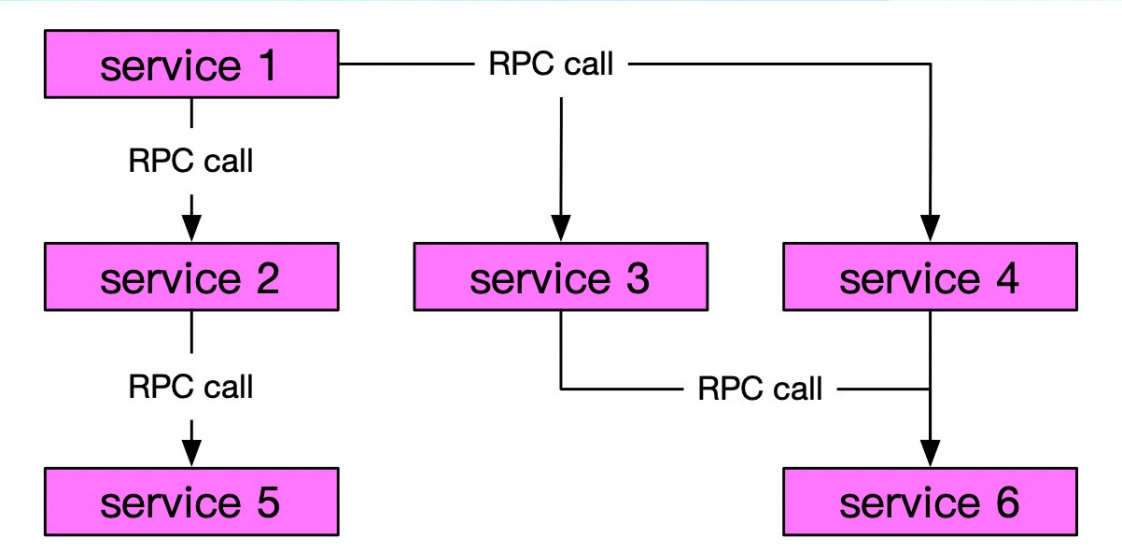


Function Call Cost: ~10 ns

But: Significant Maintainability and Stability Costs

- multiple teams develop in a same codebase
- If 1 module crash, the whole service process crash

Microservices Architecture

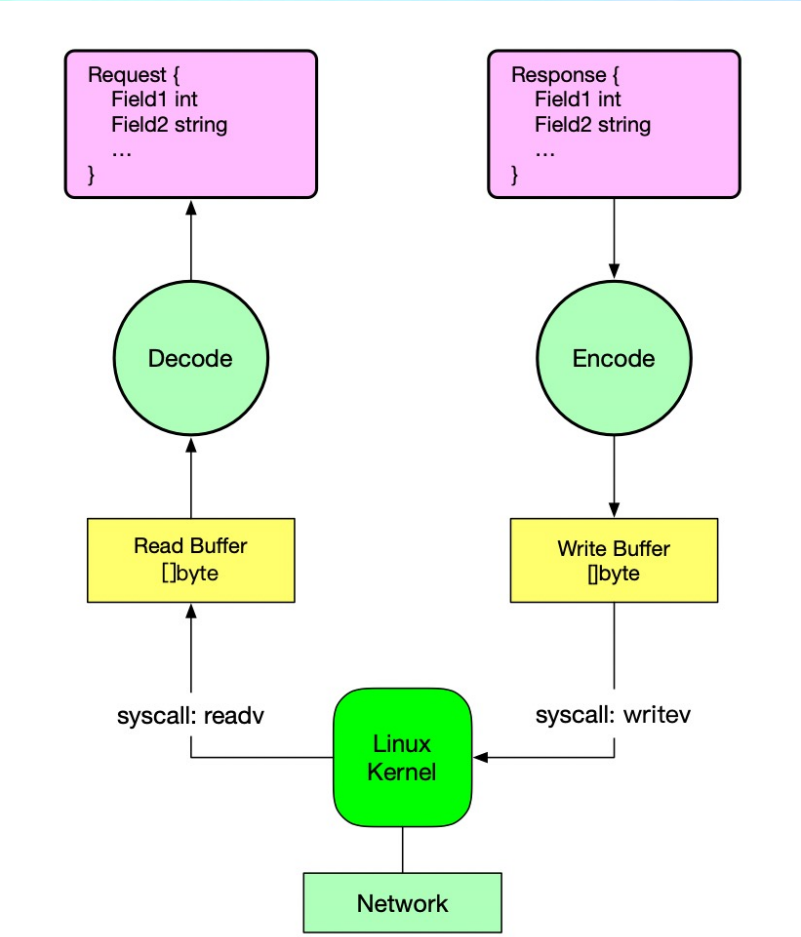


Every service have more Maintainability and Stability

- Different teams are responsible for different services and don't need to worry about other services
- If an RPC call failed, can retry to call another downstream instance

But: RPC Call Cost: 1~100 ms

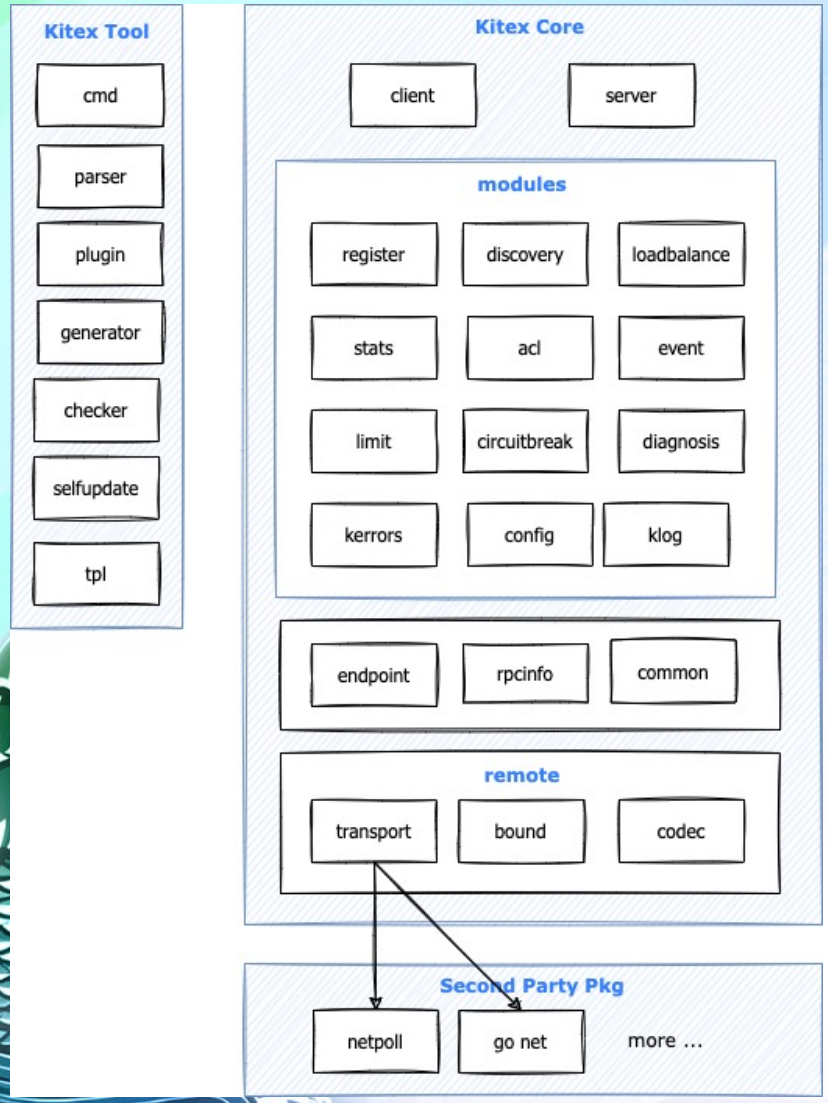
What constitutes the Cost of RPC



Costs:

- **CPU Cost**
  - Codec (Decode/Encode)
  - IO syscall (read/write)
  - ……
- **Latency Cost**
  - Kernel thread scheduling
  - Network transmission
  - ……

What is Kitex?



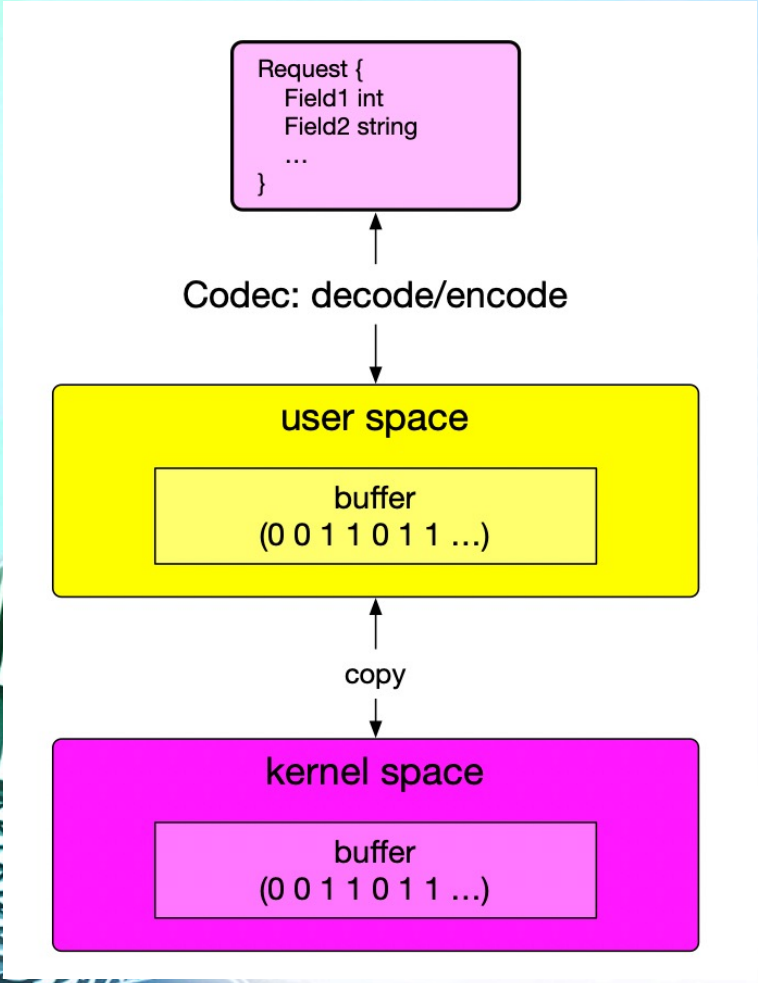
Kitex is a high-performance and strong-extensibility Golang RPC framework that helps developers to build microservices.

Part 02

Reducing CPU Cost

The Optimizations of Kitex Codec

What is codec, and why we need it?

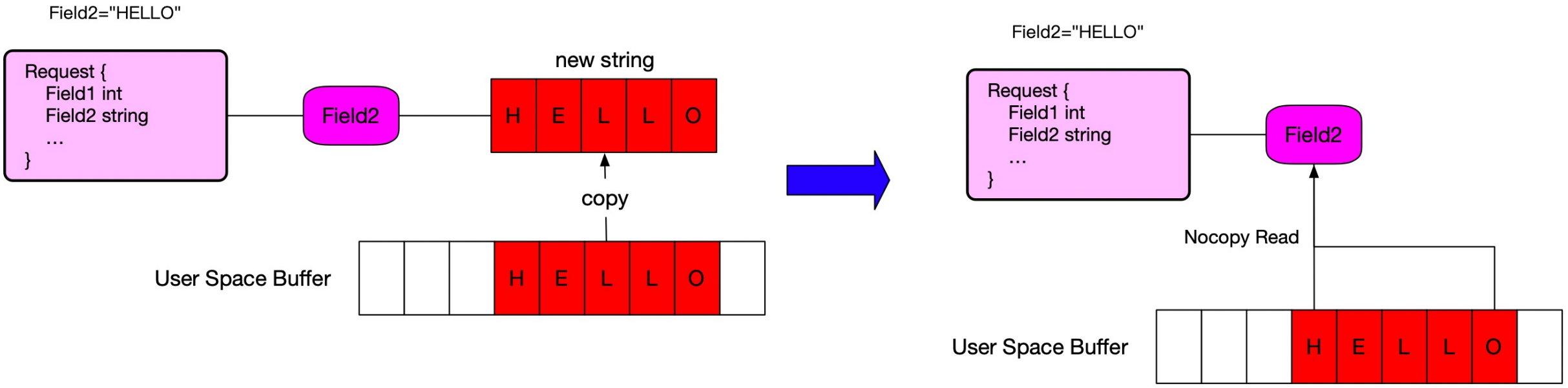


Codec helps to translate data view between:

Human View <=> Machine View

The codec is highly CPU-intensive, with our online machines consuming over 20% of CPU usage for codec.

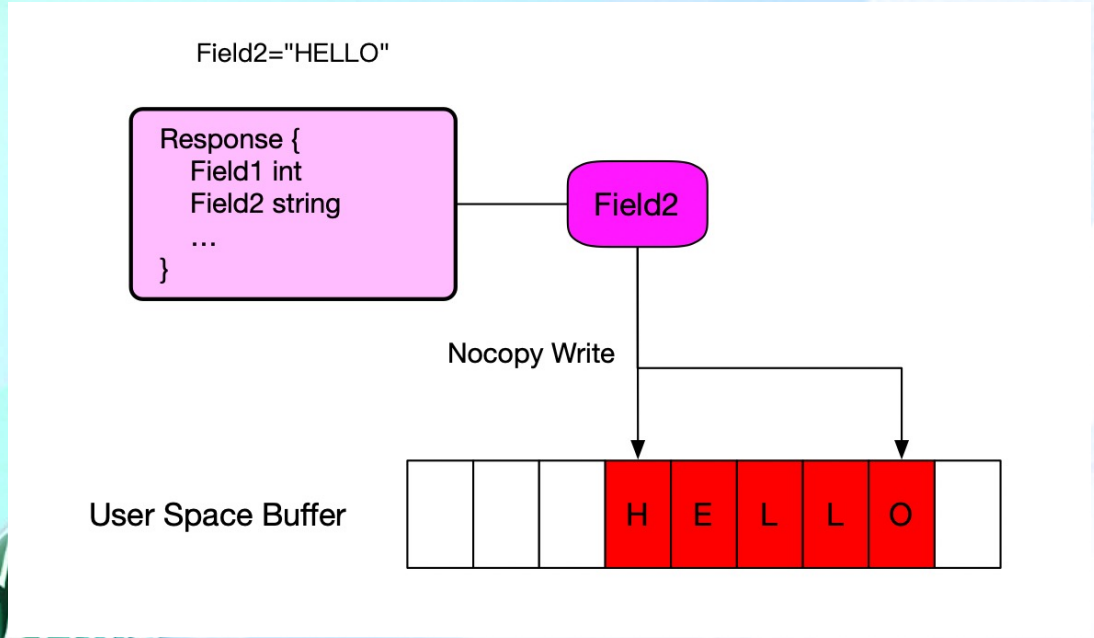
Kitex Codec Optimization: NoCopy Read



Nocopy read for string/bytes type fields

`request.Field2` uses the user space buffer memory directly, without any malloc

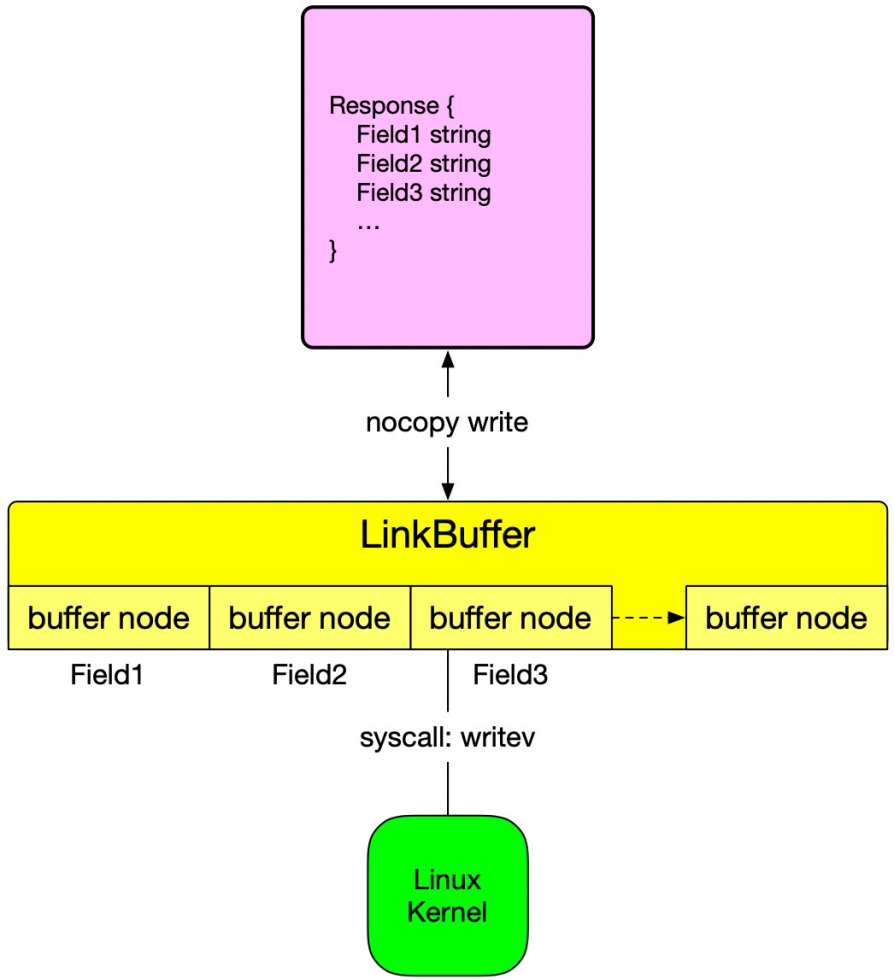
Kitex Codec Optimization: NoCopy Write



Nocopy write for string/bytes type fields

The user space buffer uses the memory of `request.Field2` directly, without any malloc

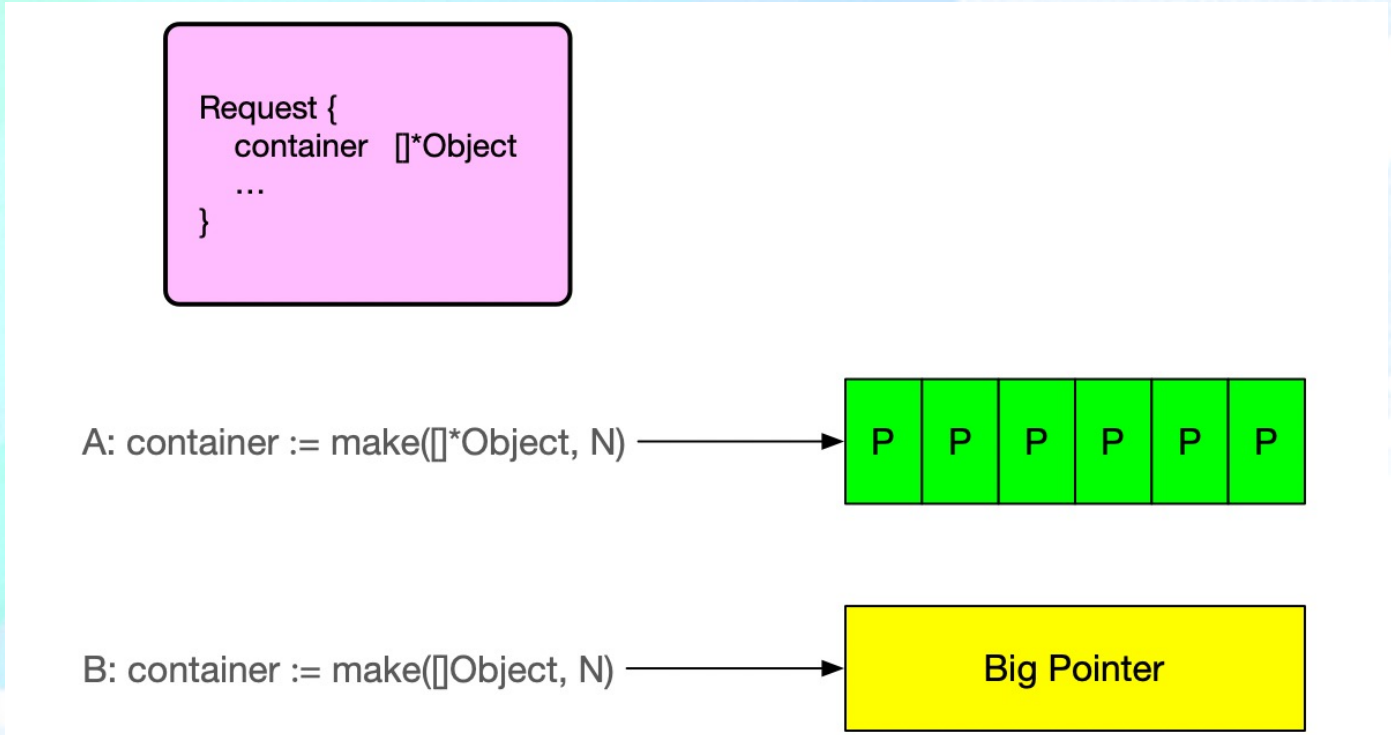
Kitex Codec Optimization: LinkBuffer



LinkBuffer: The Key to No-Copy Read/Write

writev syscall: Write multiple buffer nodes with a single system call

Kitex Codec Optimization: Reduce Object Malloc



code A: N Malloc calls and N pointers

code B: 1 Malloc calls and 1 big pointer

Why code B is better?

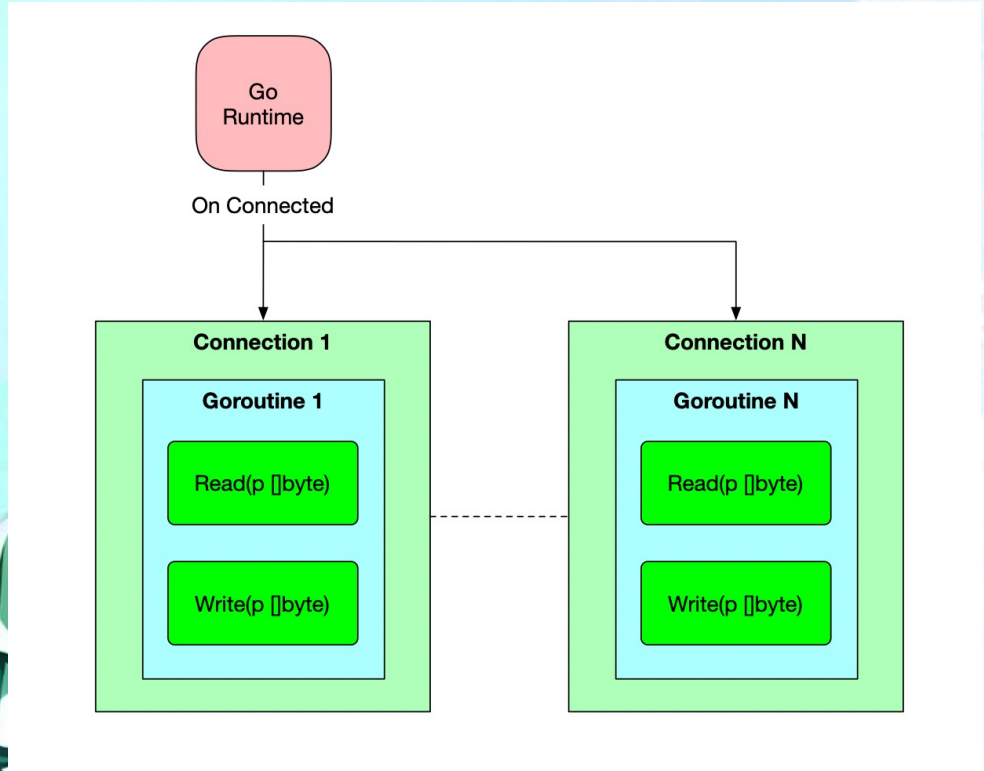
1. GC only need scan the big pointer once
2. The contiguous memory address make CPU cache happy

Part 03

Reducing Latency Cost

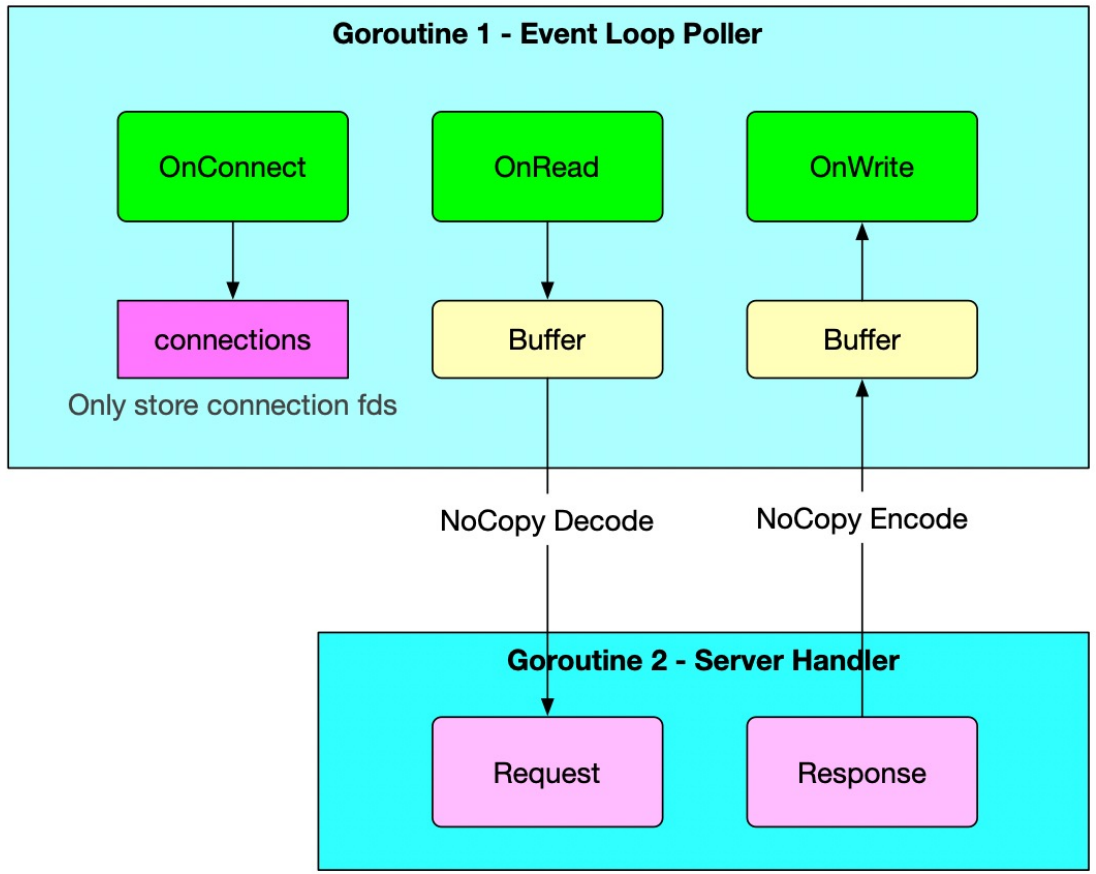
The Optimizations of Kitex network layer

The Issues with Go's Native Net Library



- N connections create N goroutines, leading to high goroutine scheduling latency.
- The native Read/Write API does not support NoCopy Read and NoCopy Write abilities.

cloudwego/netpoll: An alternative to the Go net library



Higher Throughput

- Million Connections with zero Goroutines
- Support No Copy Codec

Lower Latency

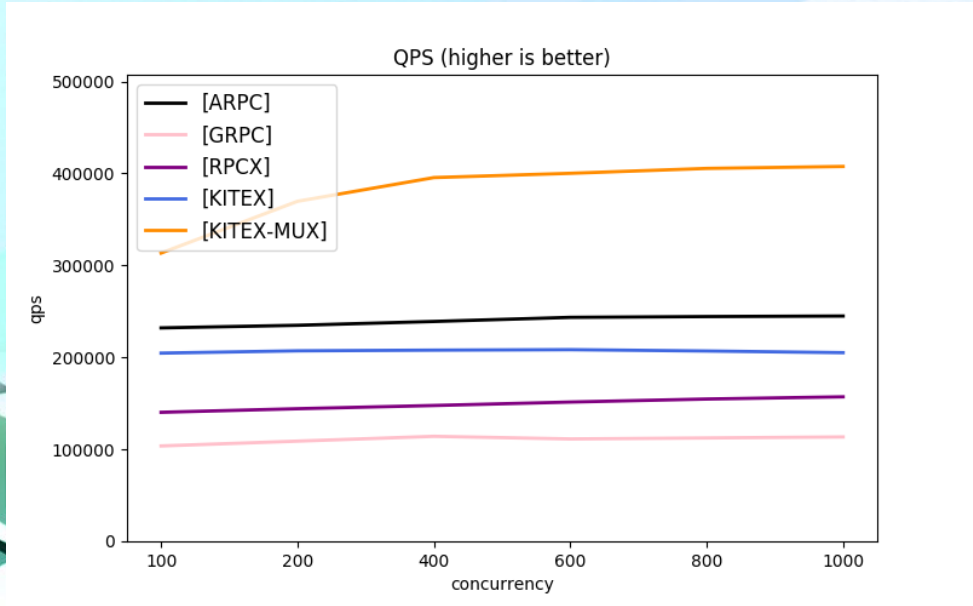
- Independently from Go runtime scheduler
- Read/Write data within the same goroutine to enhance cache efficiency

Part 04

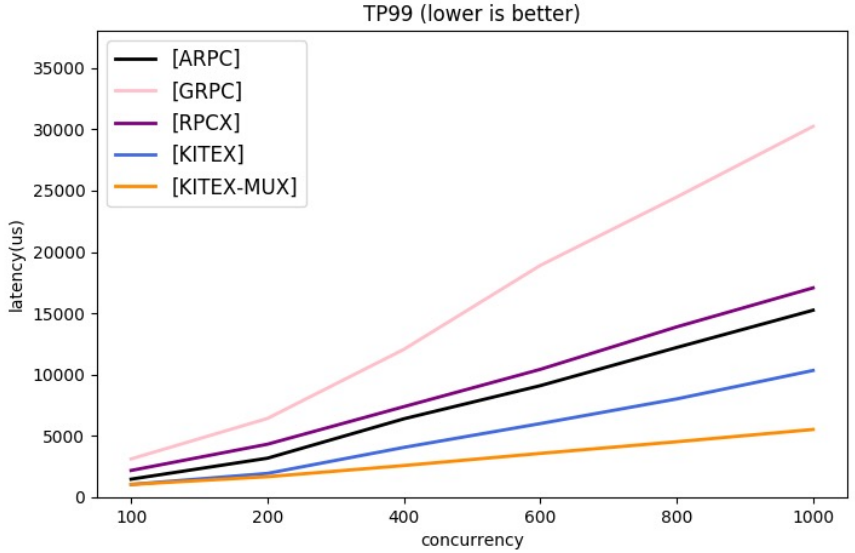
Benchmark

The Benchmark comparison of Kitex with other RPC frameworks.

Highest Throughput



lowest Latency

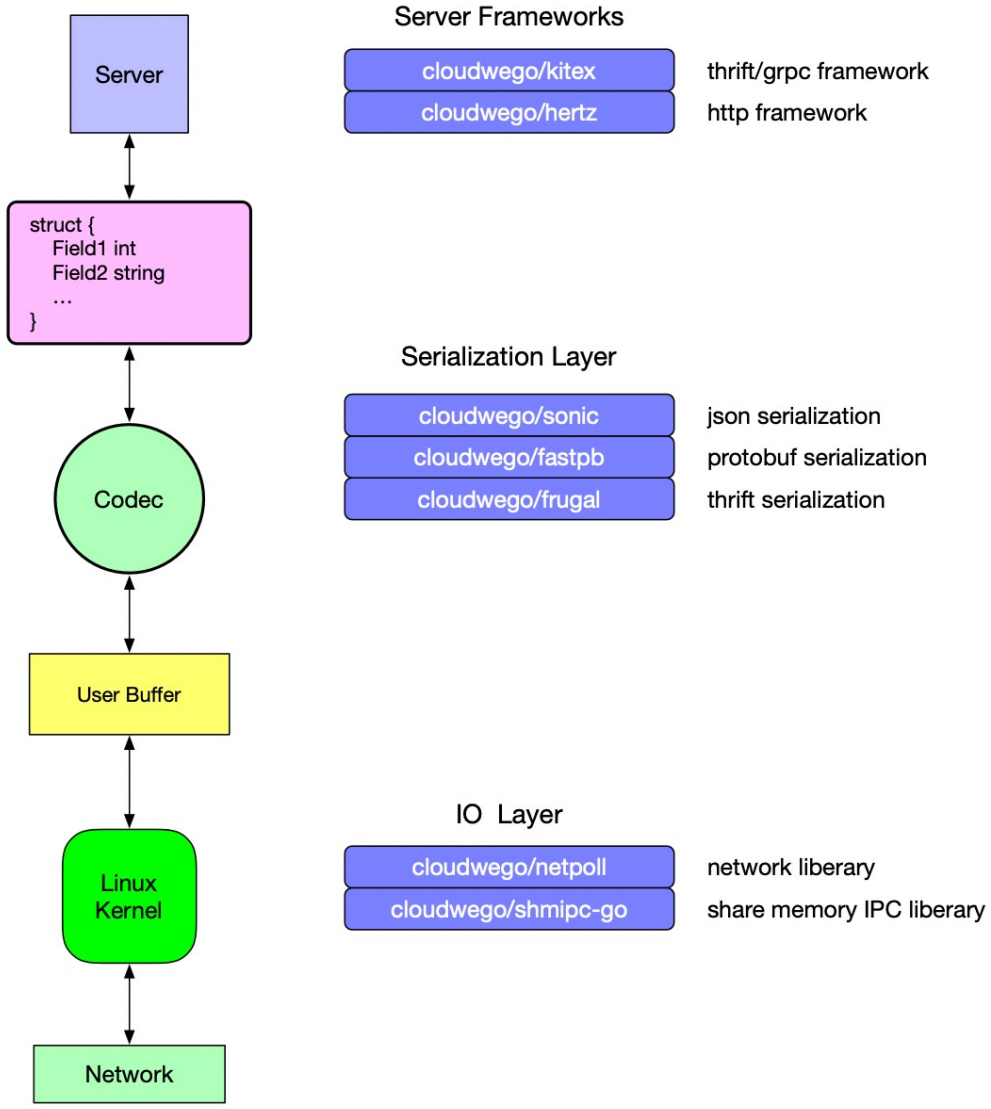


KITEX: kitex in long connection mode
KTIEX_MUX: kitex in multiplexing mode

data source: https://github.com/cloudwego/kitex-benchmark

What's more?

CloudWeGo Family



Server Frameworks

- https://github.com/cloudwego/kitex
- https://github.com/cloudwego/hertz

Serialization Library:

- https://github.com/cloudwego/sonic
- https://github.com/cloudwego/frugal
- https://github.com/cloudwego/fastpb

IO Library:

- https://github.com/cloudwego/netpoll
- https://github.com/cloudwego/shmipc-go

Thanks.