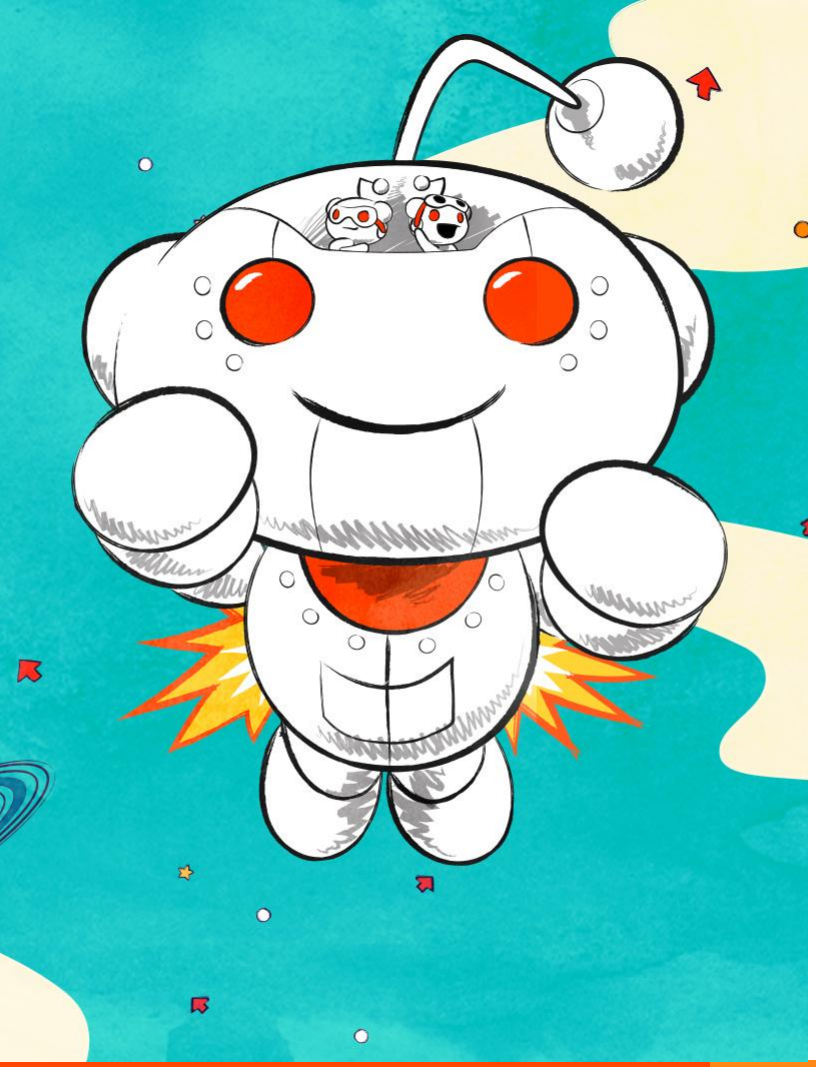
From Prototype to Production

Lessons from Building and Scaling Reddit's Ad Serving Platform with Go

Deval Shah, Software Engineer, Reddit

Agenda

- Introduction to Reddit
- Ads Architecture Overview
- Our experience with Go
  - Lessons Learned
  - Challenges



What is Reddit?

Reddit is the **frontpage** of the internet

A social network where there are tens of thousands of **communities** around whatever **passions or interests** you might have

It’s where people **converse** about the things that are most important to them

Reddit by the numbers

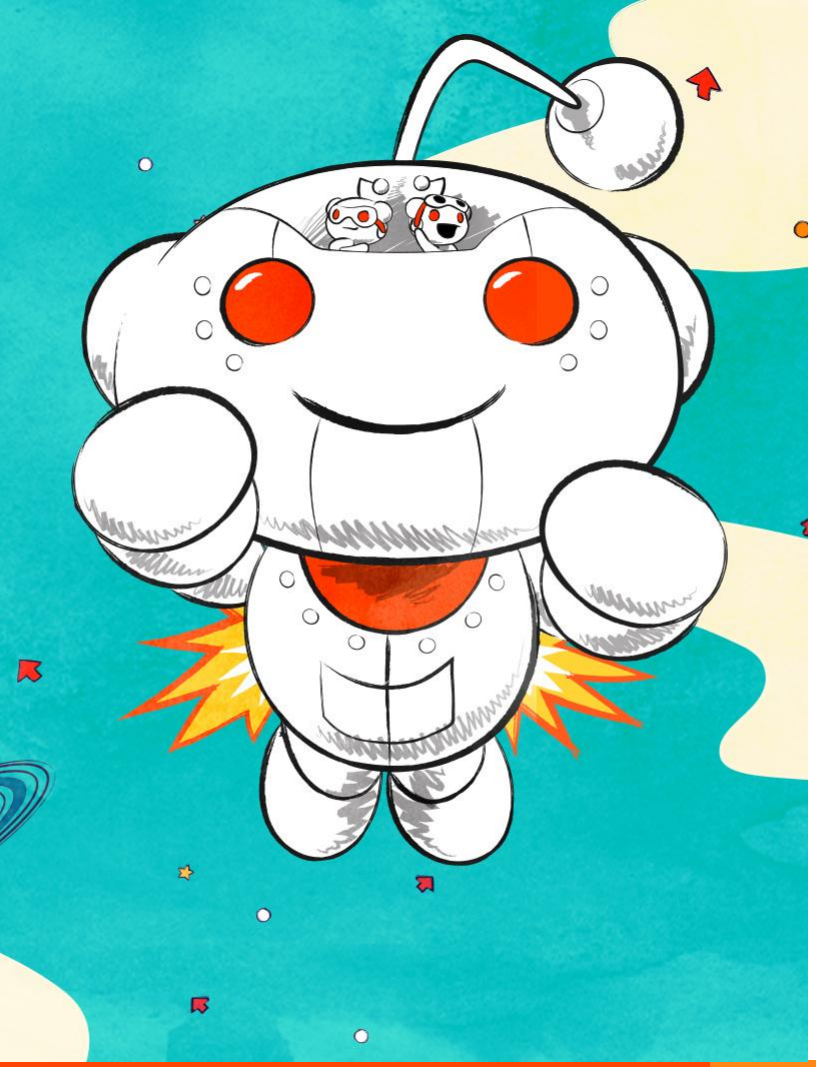
5th/18th Alexa Rank (US/World)

330M+ MAU

138K Active Communities

12M Posts per month

2B Votes per month



Ad Serving @ Reddit

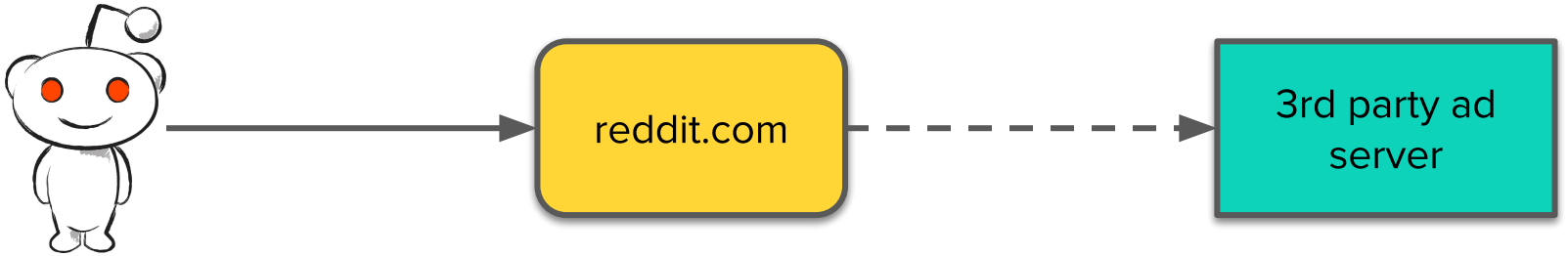
Scale

Speed

Auction

Pacing

Ad Serving @ Reddit: Before



- Slow
- Less Customizable
- Operationally Opaque

Ad Serving Infrastructure

Prometheus

Apache Thrift ™

Apache MESOS

APACHE Spark™

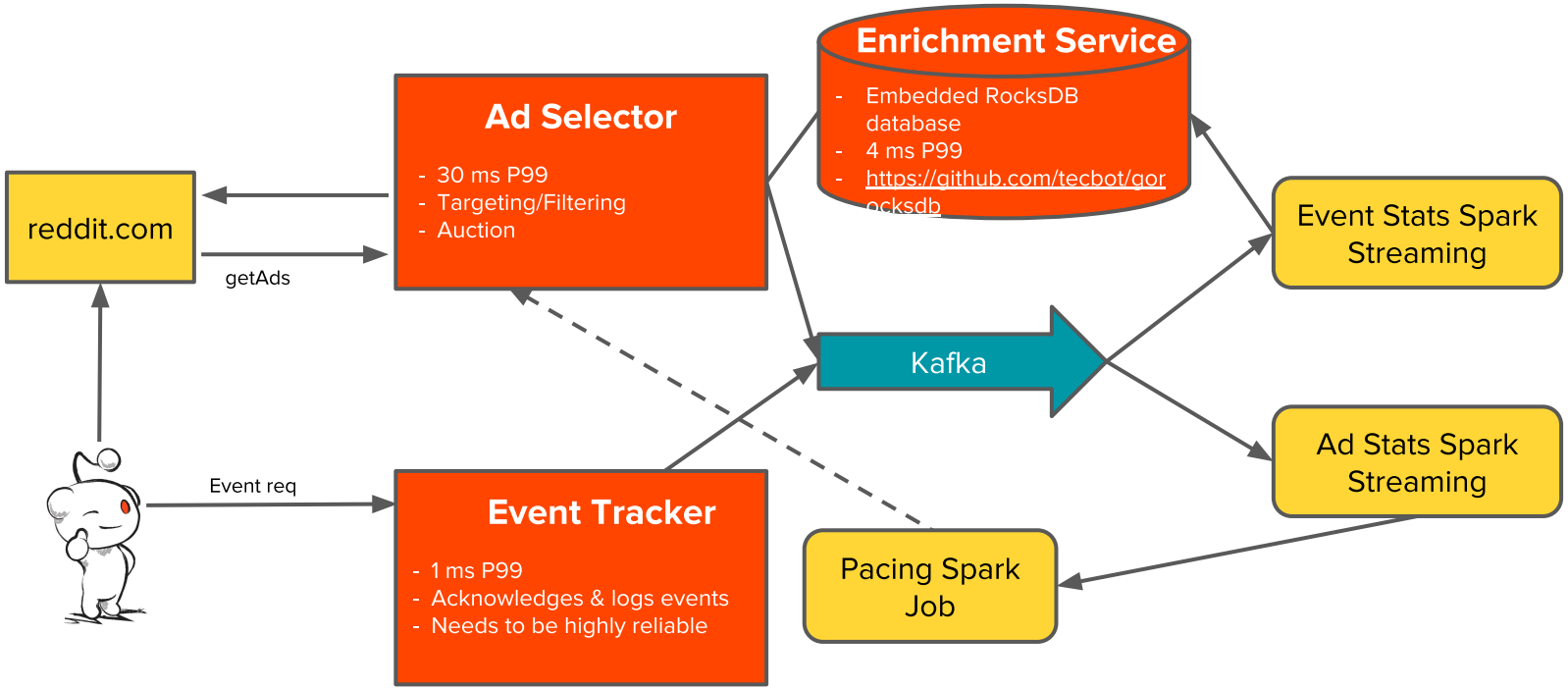
Amazon S3

RocksDB

APACHE kafka®

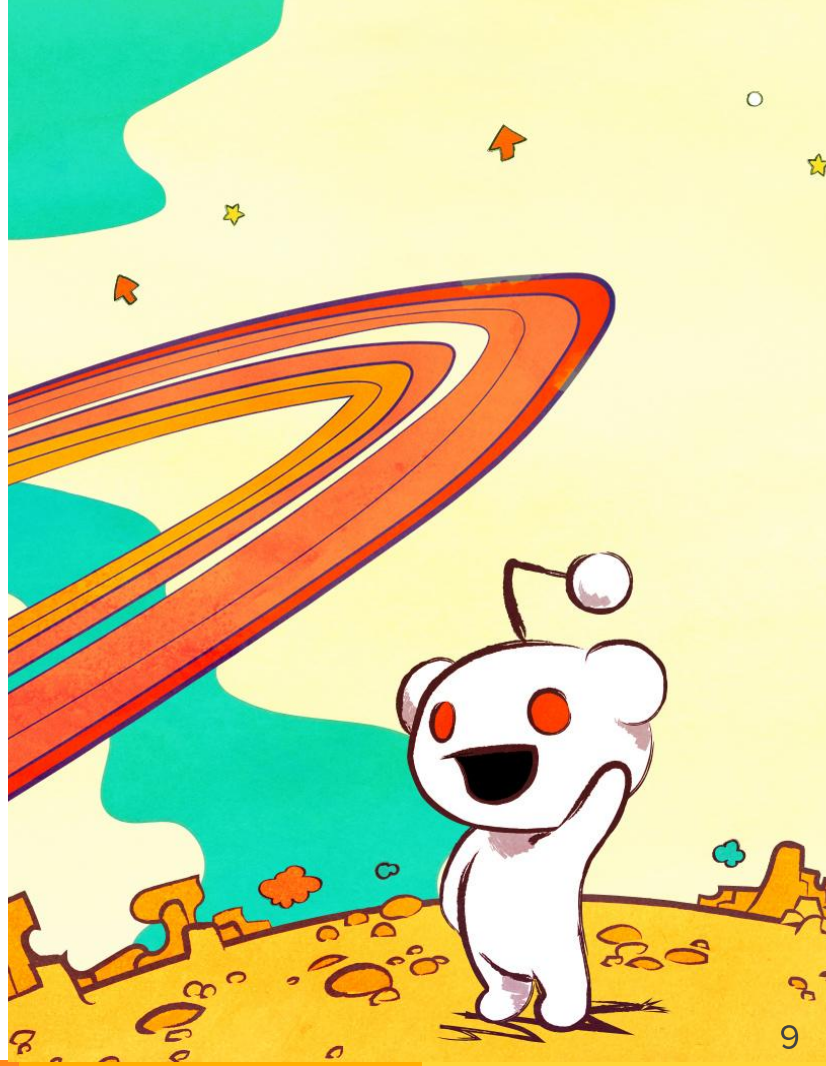
Apache ZooKeeper™

Ad Server Architecture



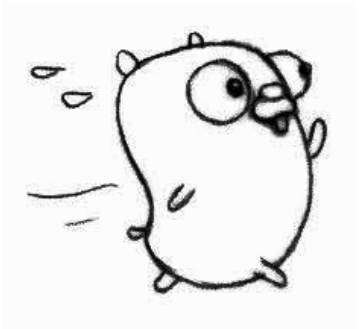
Other Go Tools & Services

- Reporting Service
- Vault Administration Tool
- Ad Event Generation Service



Our experience with Go

- Increased developer velocity
- Great performance out of the box
- Easy to focus on business logic
- Ad Serving Latency decreased drastically



Impact on ad serving latency

Lessons Learned

Problem 1

How to build production ready microservices?

- Initial prototype worked, but was not going to scale with developers
- Logging, metrics, etc. was all over the place
- Changing the transport layer was hard
- We needed repeatable patterns

Some Options

Go-Micro

"Pluggable RPC framework for distributed systems development"

Enforces gRPC

Great for beginners, more opinionated than others

https://github.com/micro/go-micro

Gizmo

"A microservice toolkit that provides packages to put together server and pubsub daemons"

Supports gRPC, http

Good for pub-sub patterns

https://github.com/NYTimes/gizmo

Go-Kit

"A programming toolkit for building microservices in Go"

Supports gRPC, http, thrift

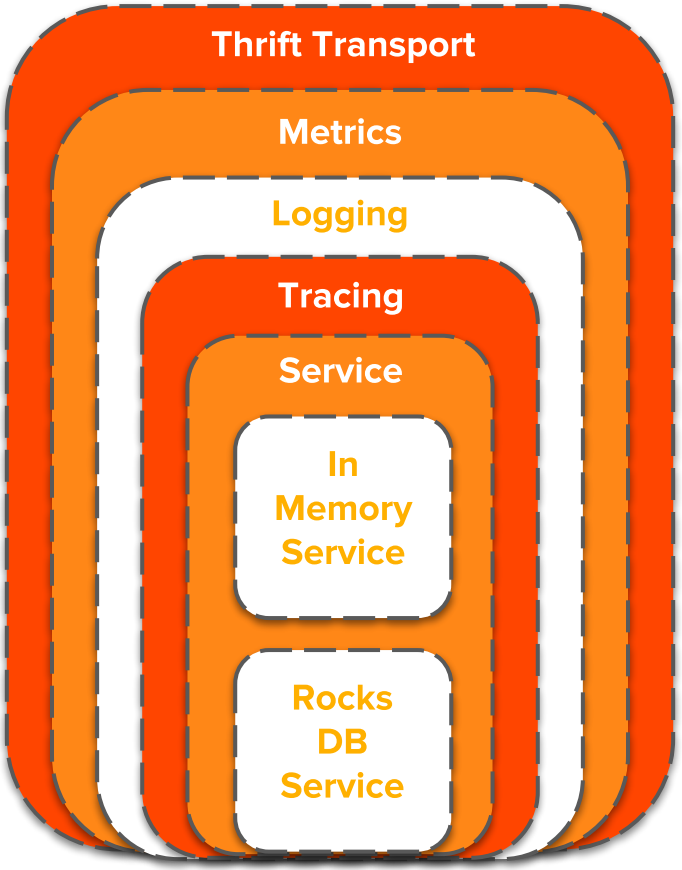
Flexible, not very opinionated

https://gokit.io/

Why we picked Go-Kit

- Supports thrift
- Is flexible
- Has tools for logging, metrics, rate limiting, tracing, circuit breaking, etc.
- For more reasons, check out Peter Bourgon's talk at GopherCon 2015

Go-kit @ Reddit



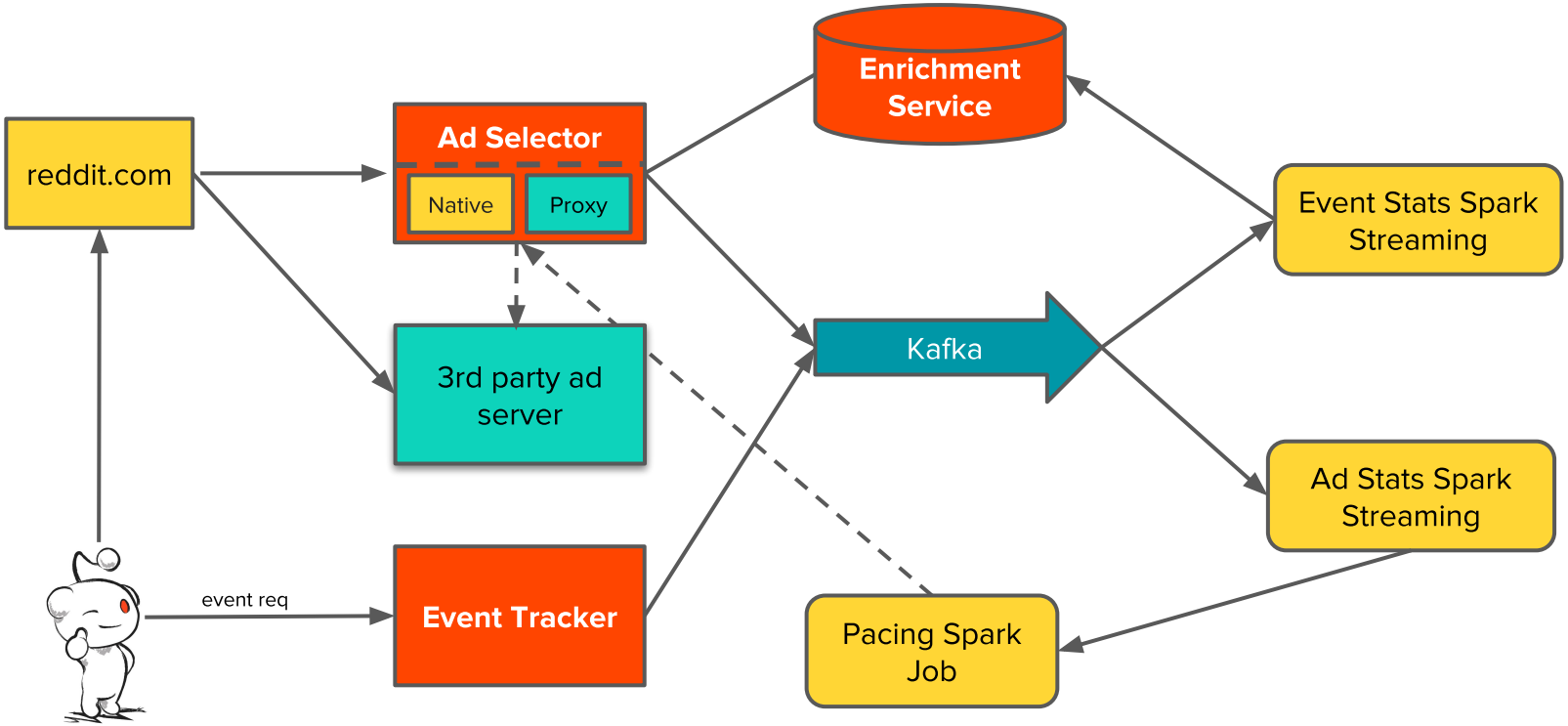
1. Use a framework/toolkit

Problem 2

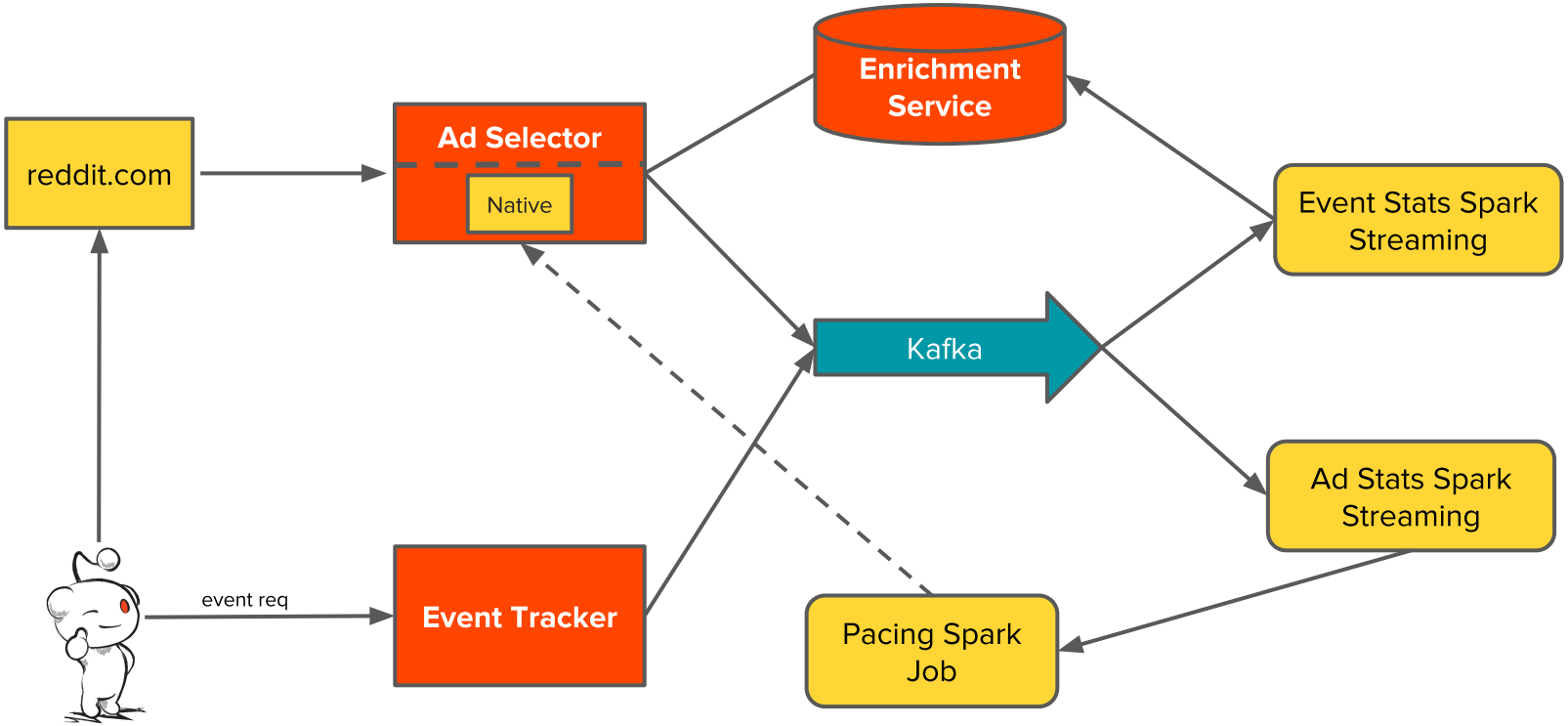
How to roll out the new system safely & quickly?

- Our goal was to roll out with minimal impact to
  - Reddit users
  - Advertisers
  - Other internal teams
- The third party ad server was a black box to us
  - Needed a way to iterate rapidly, learn and get better

Changing Airplanes Mid-Flight (During Rollout)



Changing Airplanes Mid-Flight (After Rollout)



How did Go help with this?

- The Go Compiler is fast
- Cross platform compilation support
- Self-contained binary
- Strong concurrency primitives

2. Go makes rapid iteration easy & safe

Problem 3

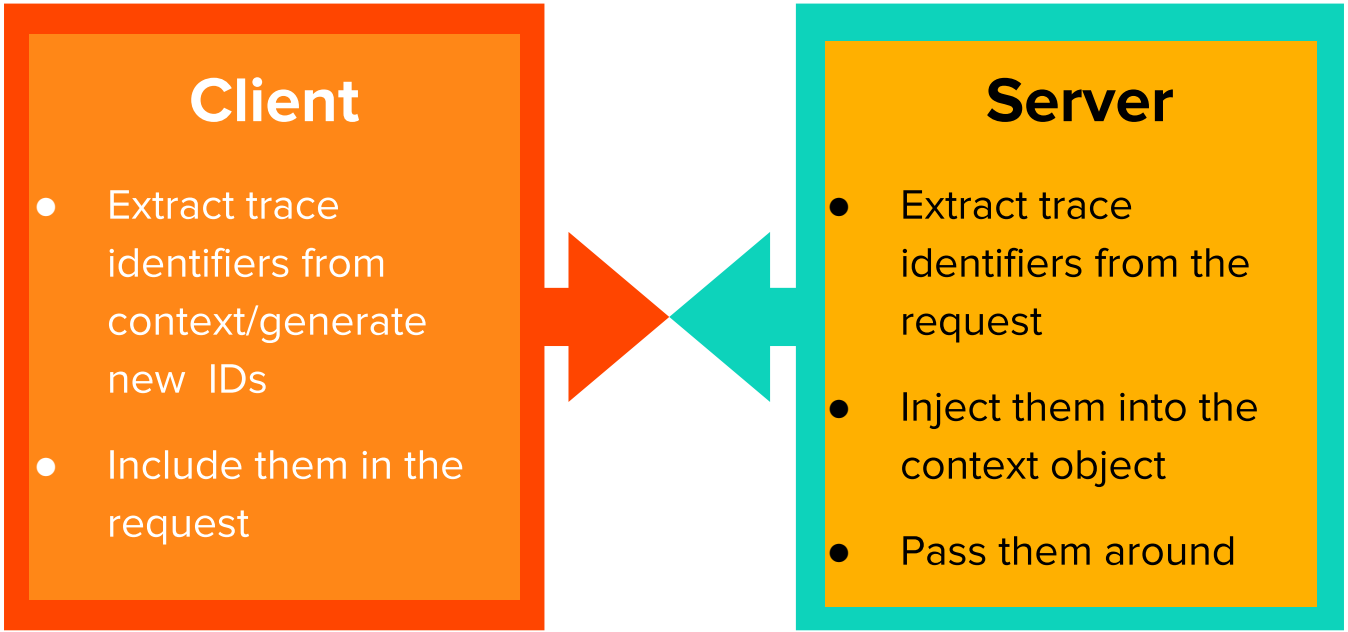
How to debug latency issues?

- pprof is great if you know which service is having issues
- Distributed Tracing gives visibility across services

Why is Tracing useful?

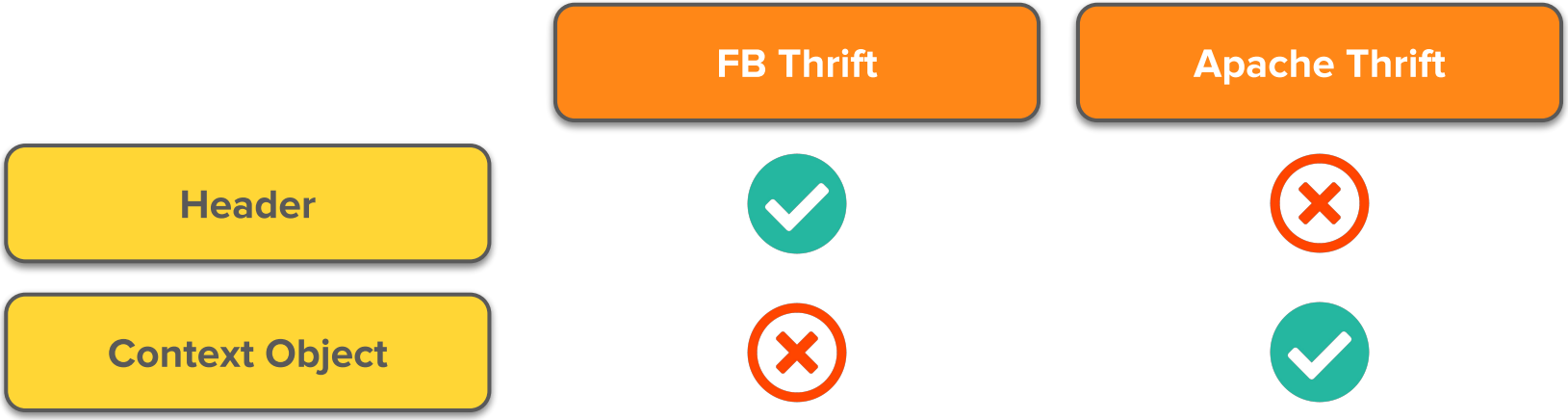
- Identifies hotspots that cause high overall latency
- Helps find other errors/unexpected behavior

Tracing is usually easy



- See go-kit's tracing package for examples

Thrift Alternatives



Solution: Add headers to Apache Thrift

- We added THeader to Apache Thrift
  - Context object is now supported
  - Headers can store trace identifiers
- https://github.com/devalshah88/thrift

Client Wrapper

```go
// Call injects trace identifiers from ctx into the thrift headers
func (p *TracingThriftClient) Call(ctx context.Context, method string, args, result thrift.TStruct) error {

    headerProto, ok := p.oprot.(*thrift.THeaderProtocol)
    if ok {
        // Get trace info from ctx, start new span if one doesnt exist
        var clientSpan opentracing.Span
        if parentSpan := opentracing.SpanFromContext(ctx); parentSpan != nil {
            clientSpan = p.tracer.StartSpan(
                method,
                opentracing.ChildOf(parentSpan.Context()),
            )
        } else {
            clientSpan = p.tracer.StartSpan(method)
        }
        defer clientSpan.Finish()
        ext.SpanKindRPCClient.Set(clientSpan)

        // Inject trace identifers into the header protocol
        if err := p.tracer.Inject(clientSpan.Context(),
            opentracing.TextMap,
            THeaderReaderWriter{headerProto}); err != nil {
            p.logger.Log("err", err)
        }
    }

    return p.Client.Call(ctx, method, args, result)
}
```

Server Wrapper

```
// Extract headers into context object
headerProto, ok := iprot.(*thrift.THeaderProtocol)
if ok {
    var span opentracing.Span
    wireContext, err := w.tracer.Extract(opentracing.TextMap, THeaderReaderWriter{headerProto})
    if err != nil && err != opentracing.ErrSpanContextNotFound {
        w.logger.Log("err", err)
    } else {
        span = w.tracer.StartSpan(w.name, ext.RPCServerOption(wireContext))
    }
    ctx = opentracing.ContextWithSpan(ctx, span)
}
```

https://github.com/devalshah88/thrift-tracing

Success!


Zipkin
Investigate system behavior
Find a trace
View Saved Trace
Dependencies
Go to trace
Duration: 22.549ms
Services: 3
Depth: 2
Total Spans: 2
JSON
Expand All
Collapse All
Filter Service...
ad-selector x2
ad-selector-client x1
enrichment-service x1
Services
4.510ms
9.020ms
13.529ms
18.039ms
22.549ms
ad-selector
22.549ms : getads
enrichment-service
4.796ms : getenrichment

3. Distributed tracing with Thrift & Go is hard

Problem 4

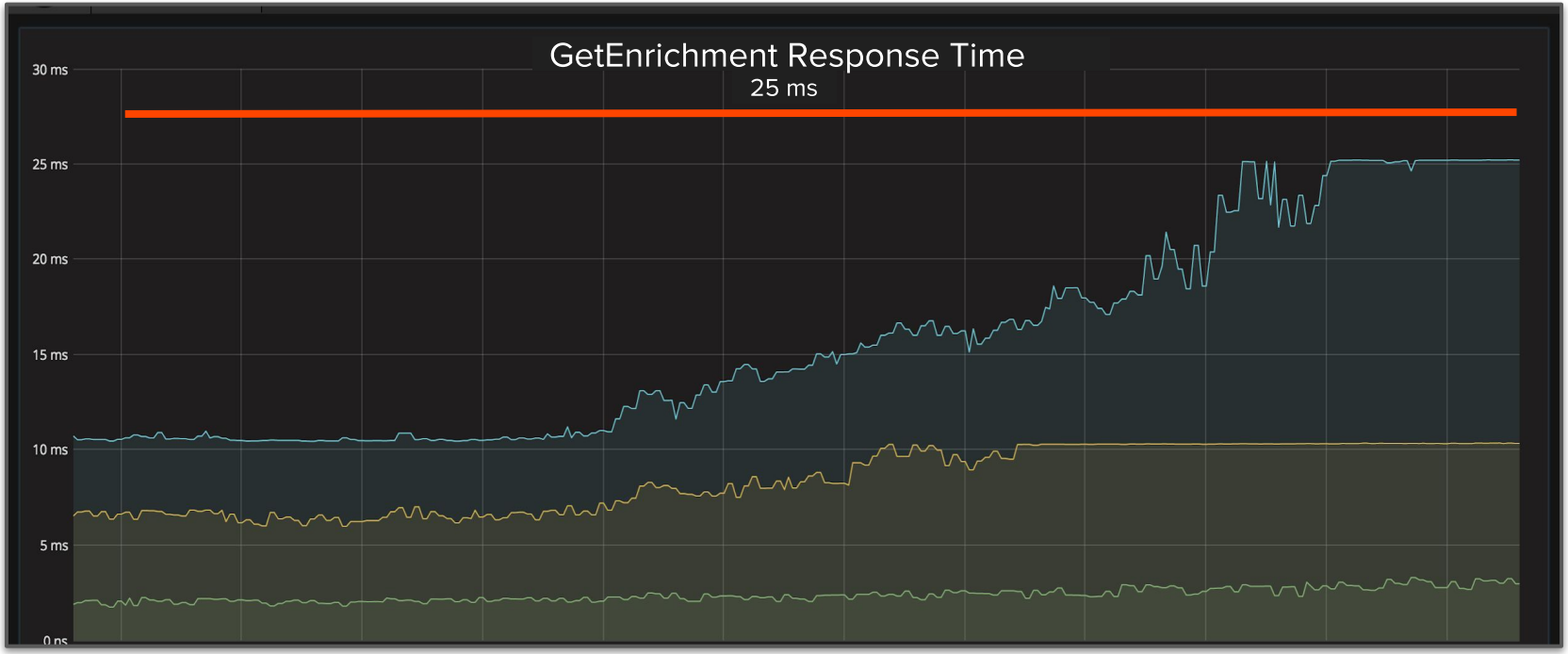
How to handle slowness/timeouts?

- Don’t make users wait for too long
- Don’t waste resources doing unnecessary work

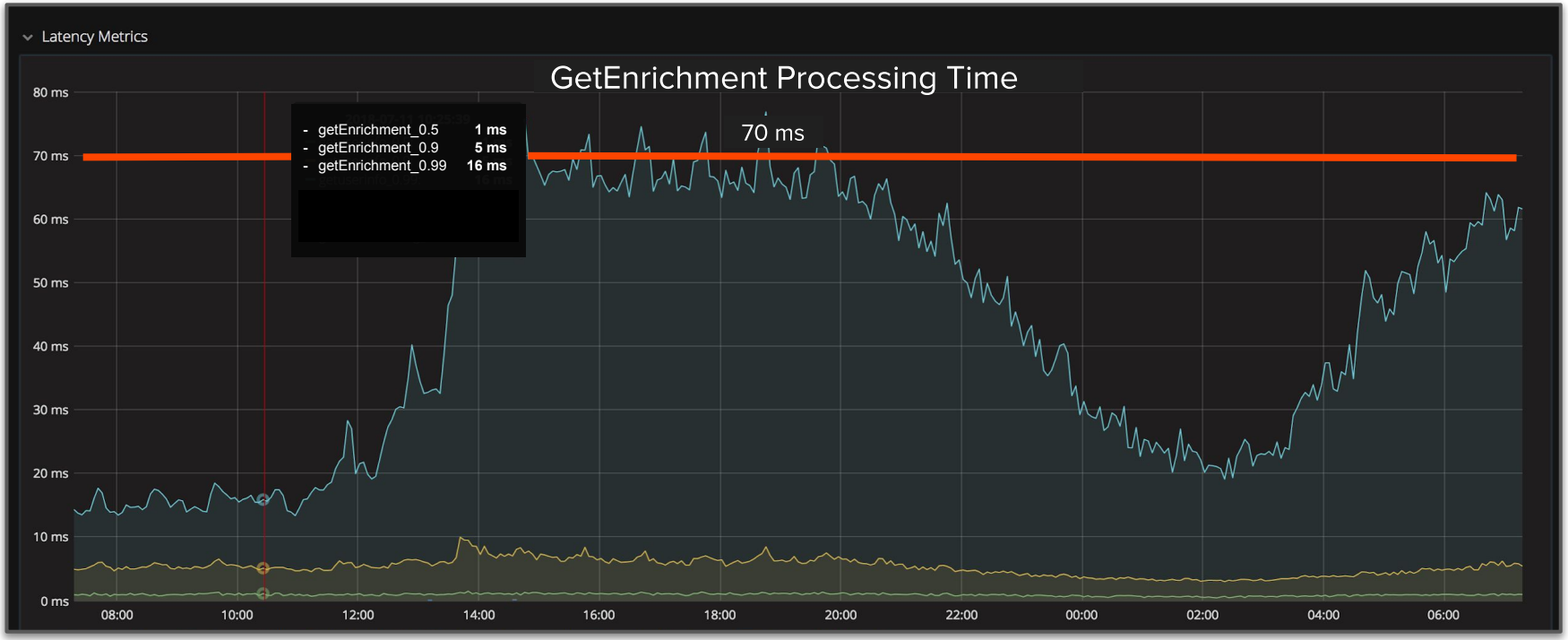
Use context to enforce timeouts within a service

```
d := time.Now().Add(20 * time.Millisecond)
ctx, cancel := context.WithDeadline(context.Background(), d)

defer cancel()

ch := make(chan EnrichmentData, 1)

go getEnrichmentData(ctx, ch)

select {

case data := <-ch:
    return &data, nil

case <-ctx.Done():
    return nil, ctx.Err()
}
```

This is good


GetEnrichment Response Time
25 ms
30 ms
25 ms
20 ms
15 ms
10 ms
5 ms
0 ns

But not enough



Propagating deadlines with HTTP

```
uri := "https://example.com"
req, err := http.NewRequest("GET", uri, nil)
if err != nil {
    log.Fatalf("NewRequest() failed with '%s'\n", err)
}

ctx, cancel := context.WithTimeout(ctx, time.Millisecond*100)
defer cancel()

req = req.WithContext(ctx)

resp, err := http.DefaultClient.Do(req)

if err != nil {
    log.Fatalf("failed with:\n'%s'\n", err)
}
defer resp.Body.Close()
body, err := ioutil.ReadAll(resp.Body)
return body
```

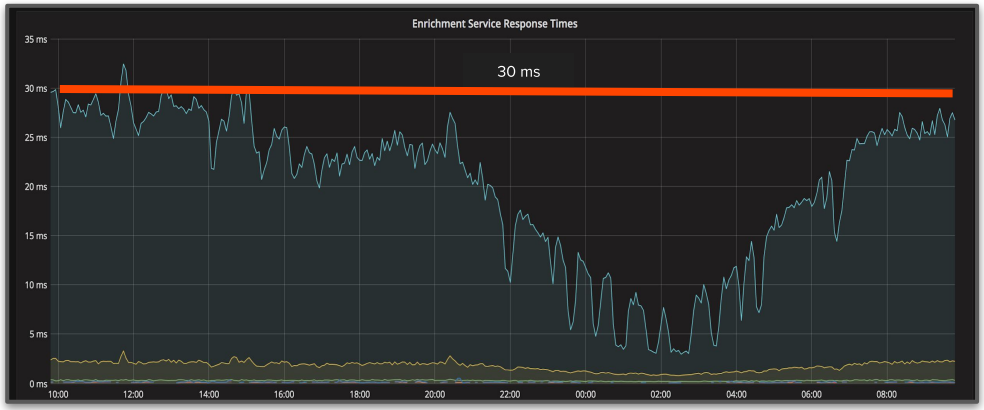
Thrift makes this hard

```
func (p *TSimpleServer2) AcceptLoop() error {
    for {
        client, err := p.serverTransport.Accept()
        if err != nil {
            select {
            case <-p.quit:
                return nil
            default:
            }
            return err
        }
        if client != nil {
            p.Add(1)
            go func() {
                if err := p.processRequests(client); err != nil {
                    log.Println("error processing request:", err)
                }
            }()
        }
    }
}
```

- Context object not used for cancellations/timeouts
- Goroutine leak

But it can be done

- Add deadline to the request payload
  - Client: Include deadline in request
  - Server: Inject deadline into context and use it
- Pass deadline as a thrift header
  - Similar to how we pass trace identifiers



4. Use Deadlines within & across services

Problem 5

How to ensure new features don’t degrade performance?

- Rapid iteration & complex business logic could lead to performance issues
- Needed processes/tools to ensure we could move fast without violating our latency SLA

Load Testing using Bender

```
func main() {
    flag.Parse()

    intervals := bender.ExponentialIntervalGenerator(float64(*qps))
    exec := bthrift.NewThriftRequestExec(
        thrift.NewTHeaderTransportFactory(thrift.NewTBufferedTransportFactory(8192)),
        GetAdsExecutor, 10*time.Second, *addr)
    recorder := make(chan interface{}, 10000)
    requests := GetAdsRequests(*count)
    bender.LoadTestThroughput(intervals, requests, exec, recorder)

    l := log.New(os.Stdout, "", log.LstdFlags)
    h := hist.NewHistogram(60000, int(time.Microsecond))
    bender.Record(recorder, bender.NewLoggingRecorder(l), bender.NewHistogramRecorder(h))
    fmt.Println(h)
}
```

Load Testing

```
Percentiles:
 Min:      536
 Median:   3707
 90th:     5796
 99th:     6202
 99.9th:   10356
 99.99th:  30503
 Max:      60000
Stats:
 Average: 3873.150700
 Total requests: 10000
 Elapsed Time (sec): 19.8343
 Average QPS: 504.18
 Errors: 0
 Percent errors: 0.00
```

https://github.com/pinterest/bender

Benchmarking

```go
func BenchmarkFetchAds(b *testing.B) {
    // Setup
    selector := Selector{&testDomain, &testSecret, &blankCampaignID}
    request := makeRequest(())
    selector.UpdateAdAvailability(testCtx, makeAdEngagements(1000, 8))

    // Run Benchmark
    for n := 0; n < b.N; n++ {
        selector.FetchNativeAds(context.Background(), request)
    }
}
```

Benchmarking

```
Running tool: /usr/local/go/bin/go test -benchmem -run=^$ ad_selector/native -bench ^BenchmarkFetchAds$

goos: darwin
goarch: amd64
pkg: ad_selector/native
BenchmarkFetchAds-8        10000        112662 ns/op       58762 B/op        605 allocs/op
PASS
ok      ad_selector/native  1.179s
Success: Benchmarks passed.
```

5. Load testing & Benchmarking is easy. Do it!

Let’s Recap

1. Use a framework/toolkit

2. Go makes rapid iteration safe & easy

3. Distributed tracing with thrift & Go is hard

4. Use Deadlines within & across services

5. Load testing & Benchmarking is easy. Do it!

Conclusion

- Go has been instrumental in helping reddit build and scale the new ad serving platform
- We shared 5 key takeaways from our experience
- Try to use at least one of them in your next Go project!

Thanks!

Deval Shah

twitter.com/devalshah