Go RPC Comparison

Brandon Philips

CoreOS

CTO, co-founder

Go RPC Comparison

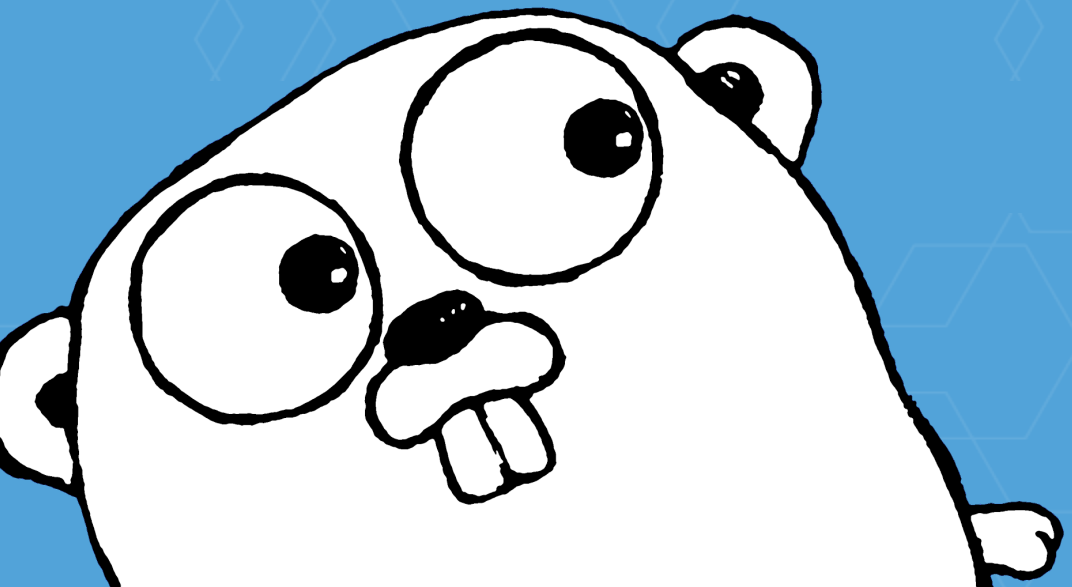
~~**Brandon Philips**~~
Jonathan Boulle

CoreOS

~~CTO, co-founder~~

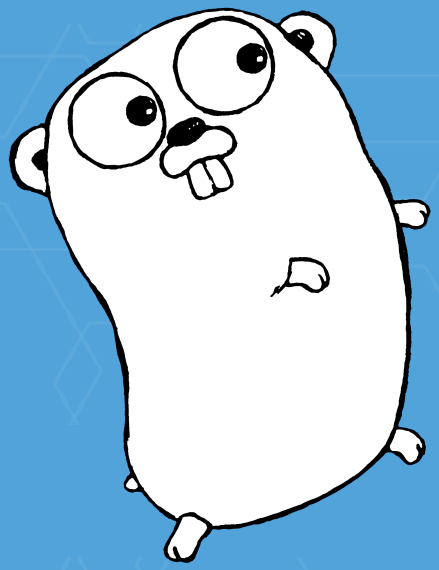
Developer, rkt lead

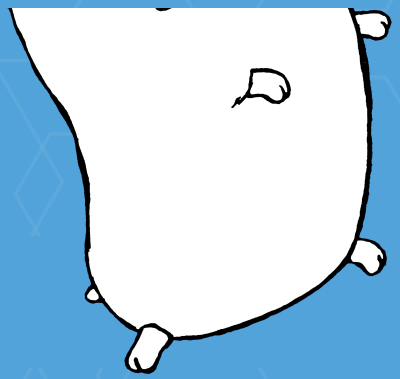

CoreOS



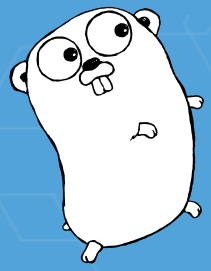
CoreOS

Go Gopher Credit: Renee French

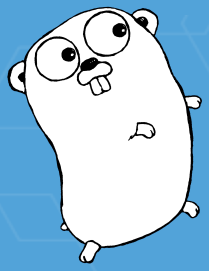




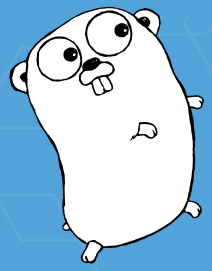
why do we RPC?



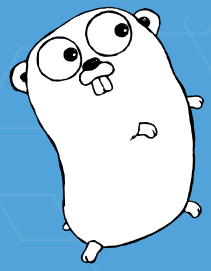
queue service



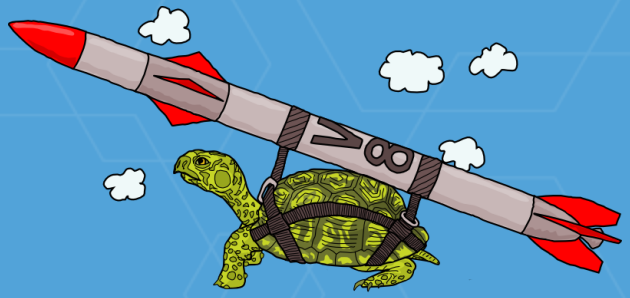
queue service

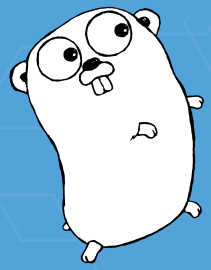


queue service

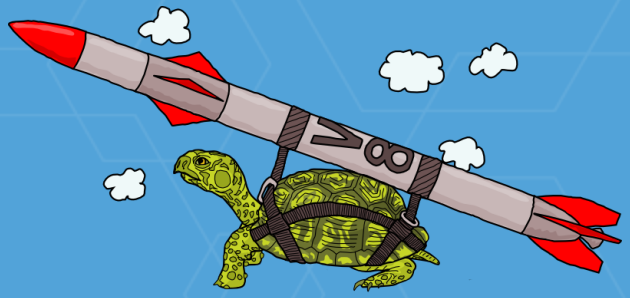


queue service

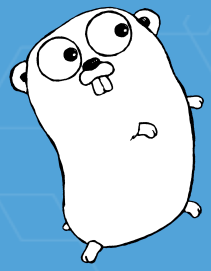




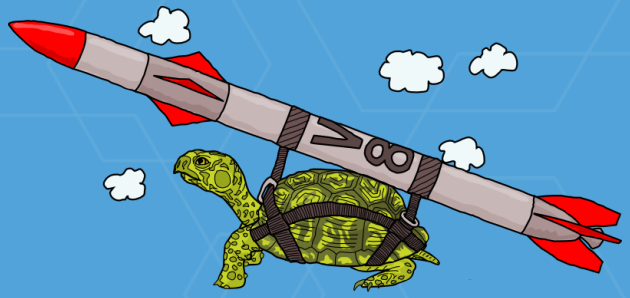
queue service



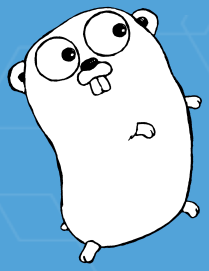
enqueue( )



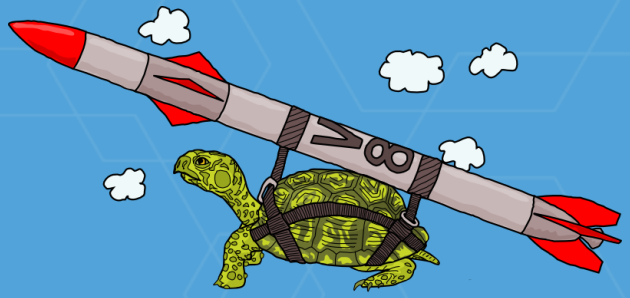
queue service



enqueue({ msg: "Hello" })

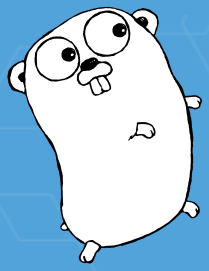


queue service



enqueue({ msg: "Hello" })

network



queue service

network

RPC Properties

- ease of use in Go
- client gen & interop
- compat with frontend
- mem/bandwidth/latency

ideal: ease of use in go

```
type Args struct {
    A, B string
}
func (s *srv) Cat(args *Args, reply *string) error {
    *reply = *args.A + *args.B
    return nil
}
```

suboptimal: ease of use in go

```
Cat(w http.ResponseWriter, req *http.Request) {
        body, err := ioutil.ReadAll(req.Body)
        args := Args{}
        err = json.Unmarshal(body, &args)
        reply = []bytes(args.A) + []bytes(args.B)
        io.Copy(w, bytes.NewBuffer(reply))
}
```

suboptimal: client gen & interop

- Only generate struct/type bindings
  - protobuf
  - json schema
  - capnproto

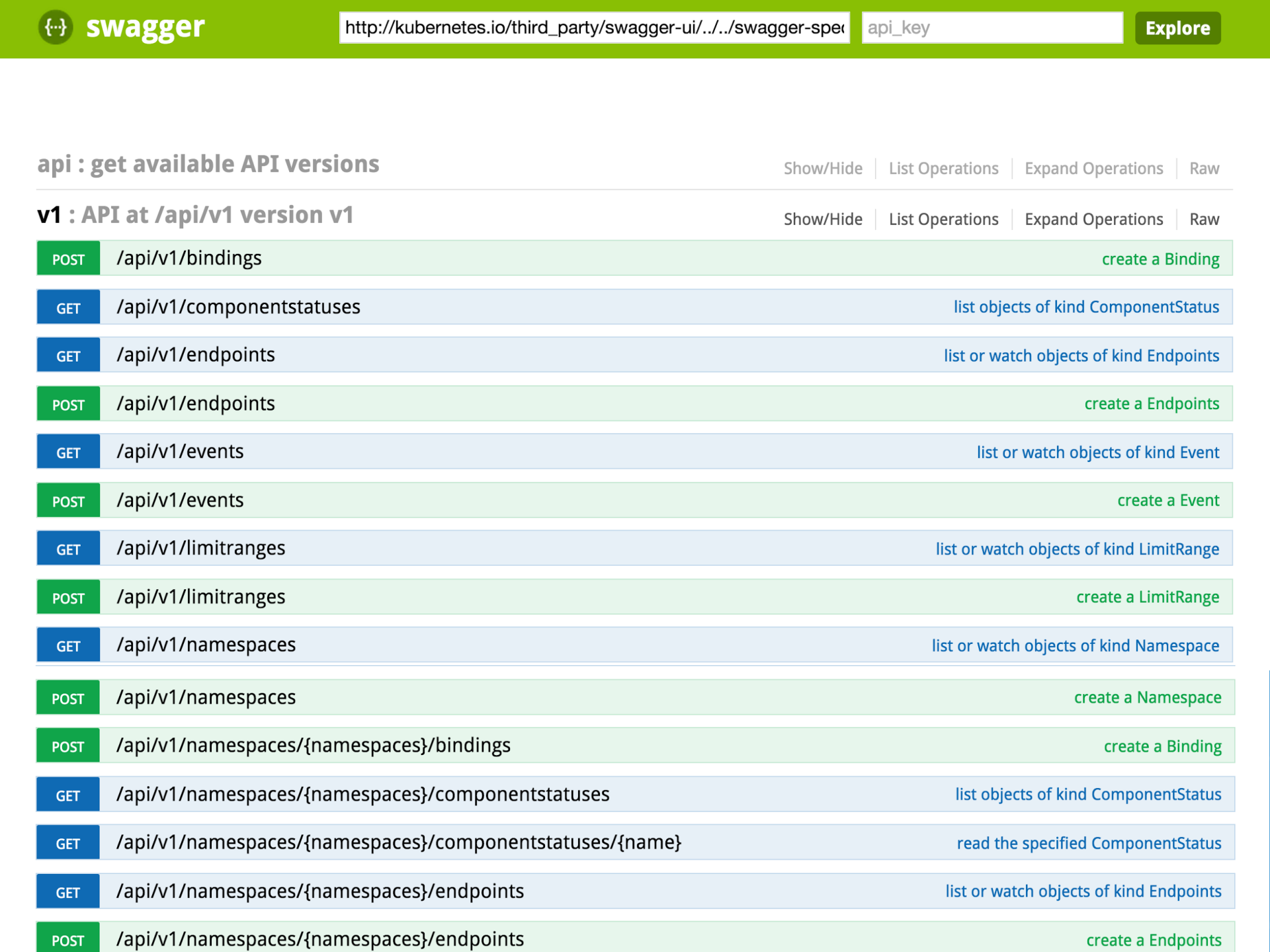
- Expose objects w/o description
  - net/rpc/jsonrpc
  - net/rpc

aside - why client gen & interop?

- Everyone says they want a REST API
- When really they want an API & client library
- You don't want to maintain a client library

ideal: client gen & interop

- Download a document describing API
- Generate API library for target language
  - types
  - methods
- Use API w/o thinking about encoding/transport


swagger
http://kubernetes.io/third_party/swagger-ui/../../swagger-spec
api_key
Explore
api : get available API versions
Show/Hide
List Operations
Expand Operations
Raw
v1 : API at /api/v1 version v1
Show/Hide
List Operations
Expand Operations
Raw
POST /api/v1/bindings create a Binding
GET /api/v1/componentstatuses list objects of kind ComponentStatus
GET /api/v1/endpoints list or watch objects of kind Endpoints
POST /api/v1/endpoints create a Endpoints
GET /api/v1/events list or watch objects of kind Event
POST /api/v1/events create a Event
GET /api/v1/limitranges list or watch objects of kind LimitRange
POST /api/v1/limitranges create a LimitRange
GET /api/v1/namespaces list or watch objects of kind Namespace
POST /api/v1/namespaces create a Namespace
POST /api/v1/namespaces/{namespaces}/bindings create a Binding
GET /api/v1/namespaces/{namespaces}/componentstatuses list objects of kind ComponentStatus
GET /api/v1/namespaces/{namespaces}/componentstatuses/{name} read the specified ComponentStatus
GET /api/v1/namespaces/{namespaces}/endpoints list or watch objects of kind Endpoints
POST /api/v1/namespaces/{namespaces}/endpoints create a Endpoints

bold red = required

Execute

calendar.calendarList.get executed moments ago time to execute: 476 ms

Request

GET https://www.googleapis.com/calendar/v3/users/me/calendarList/?key={YOUR_API_KEY}

Response

200 OK

- Show headers -


```
{
 "kind": "calendar#calendarList",
 "etag": "\"1434504281888000\"",
 "nextSyncToken": "00001434504281888000",
 "items": [
  {
   "kind": "calendar#calendarListEntry",
   "etag": "\"1428595452318000\"",
```

ideal: compat with frontend

- Use JSON/XML
- Use websockets or https
- Generated API bindings

ideal: mem/bandwidth/latency

- Serialization requires few small allocations
- Bandwidth is minimized to only required bits
- Latency is dominated by the network

golang RPC options (abbreviated)

- HTTP+JSON+discovery/swagger
- HTTP+protobuf
- gRPC
- capnproto RPC

HTTP+JSON+discovery/swagger

HTTP+JSON

- HTTP provides the "RPC" part: verbs and request/response framing

- JSON provides the encoding of types

discovery/swagger

- discovery is used in CoreUpdate and fleet
  - Angular.js bindings built, works nicely!

- discovery JSON has an unknown path forward, using swagger for new projects

- nice-"ish" framework in Kubernetes for swagger

HTTP+JSON+discovery/swagger

+ Lots of existing tools work
+ Swagger support in Go
+ Works on frontend
+ Client generation
+ Frontend libraries for both

- Not great code gen for Python, Java, etc
- JSON encoding in Go is slow and memory intensive by default
- HTTP/1.1 not optimized for streaming etc

HTTP+protobuf

HTTP+protobuf

- Used in etcd internal cluster RPC
- Solved the efficiency problems around JSON
- Continue to have "options" because of HTTP

HTTP+protobuf

+ HTTP debugging tools work
+ Improvements in latency/memory over JSON

+/- Frontend is possible but requires lots of deps

- Uncommon hybrid, Go marshaling is left to you

gRPC

gRPC

- shiny
- protobufs + HTTP/2
- duplex streaming
- single TCP connection, multiple streams

gRPC

+ efficient
+ client gen in 10+ languages
+ streaming semantics

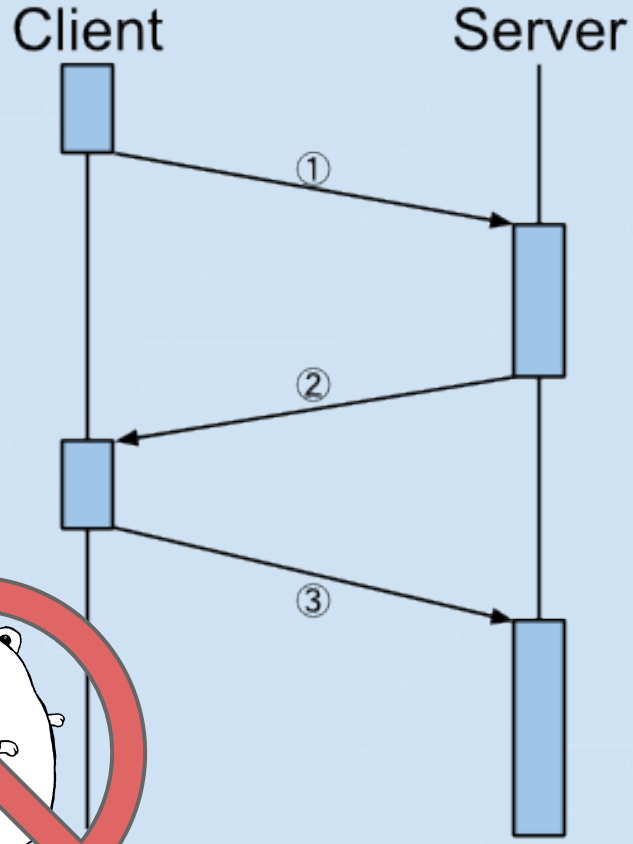
- young
- lack of tooling (especially around HTTP/2)
- no current plan for frontend (browser) support

capnproto RPC

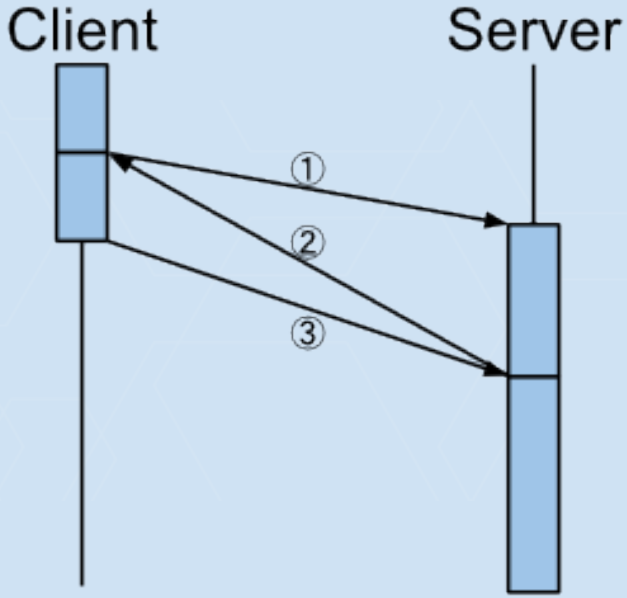
capnproto RPC

- Interfaces and objects part of the model
  - Similar to objects internal to Go, or D-Bus
- Efficient serialization
- Limited language support: C, C++, Javascript, Python, Node.js
- Fancy promise pipelining

Traditional RPC



Cap'n Proto RPC



① call foo()
② foo() returns x
③ call bar(x)

WIP github.com/zombiezen/go-capnproto

http+capnproto

benchmarks

¯_(ツ)_/¯

serialization
https://github.com/cloudflare/goser

rpc and serialization wip
https://github.com/philips/hacks/tree/master/golang-rpc-comparison

an ideal world?

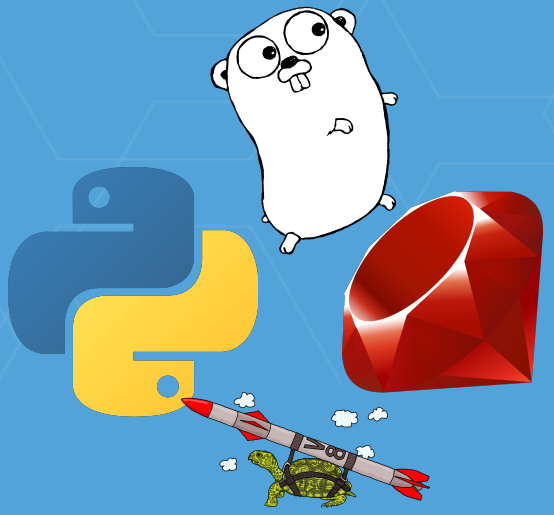
gRPC + grpc-gateway

github.com/gengo/grpc-gateway

Big Idea: For smart modern backends use the native gRPC mechanisms and get

- low-latency encoding & efficient bandwidth
- streaming endpoints
- multiplexing

For everyone else: use HTTP 1.1 and JSON generated against the gRPC spec and swagger doc.



grpc-gateway

gRPC

Remaining Problems/TBD

- Running gRPC + grpc-gateway on the same port
  - github.com/grpc/grpc-go/issues/75
  - https://github.com/philips/hacks/pull/1
- Generating swagger spec from gRPC proto
  - https://github.com/gengo/grpc-gateway/issues/9
- Potential fast-path between gRPC and gateway

recommendations

- Using JSON? Speed it up:
  - github.com/pquerna/ffjson
  - github.com/ugorji/go/codec

- Looking towards the future?
  - hack on gRPC + grpc-gateway

- Can ignore Javascript/frontend?
  - use HTTP and capnproto or protobuf

Thank You

coreos.com/careers