High-Performance Web-Services in C++

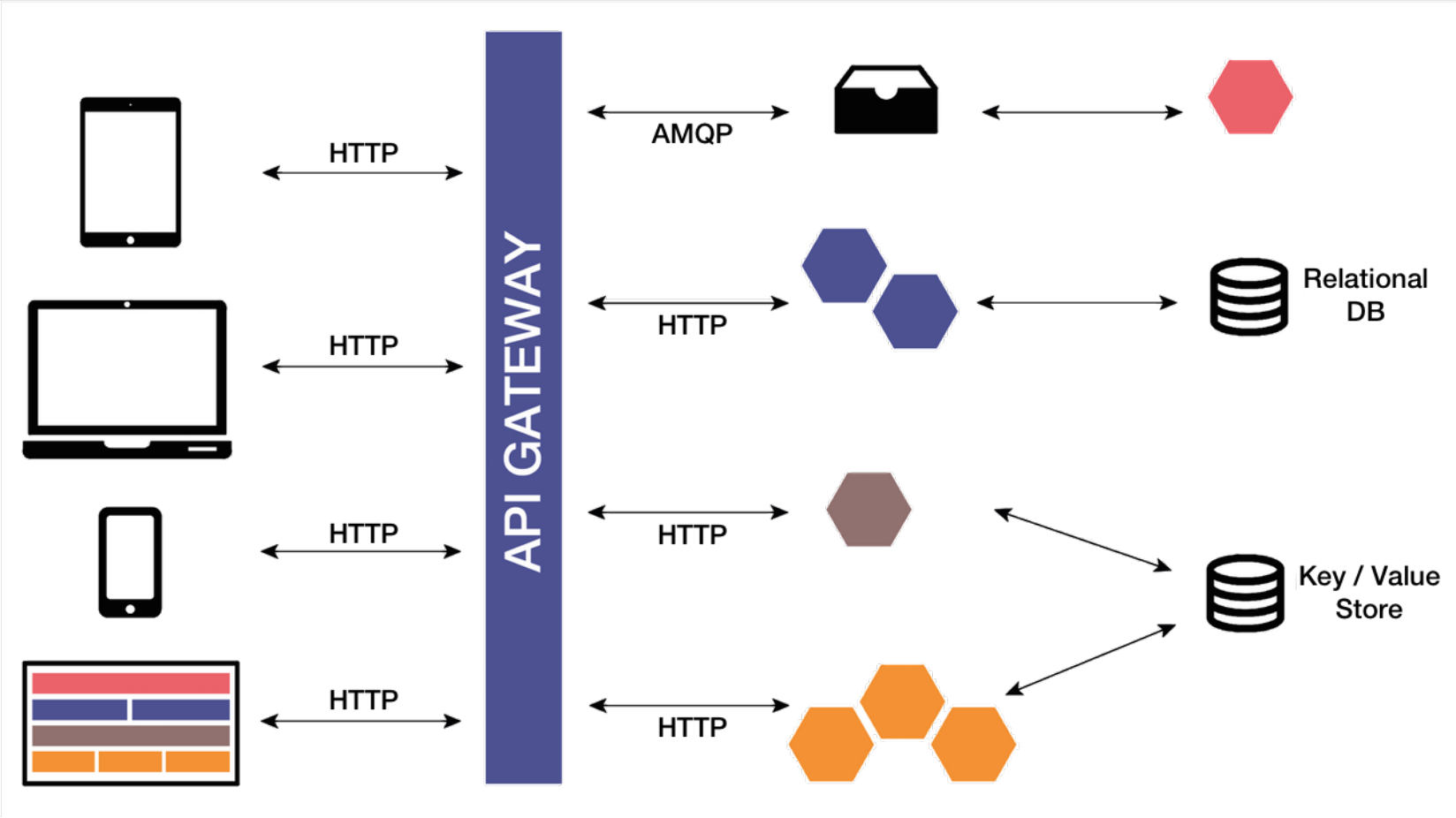
C++ User Group Dresden
Maximilian Haupt
2016/09/08

Agenda

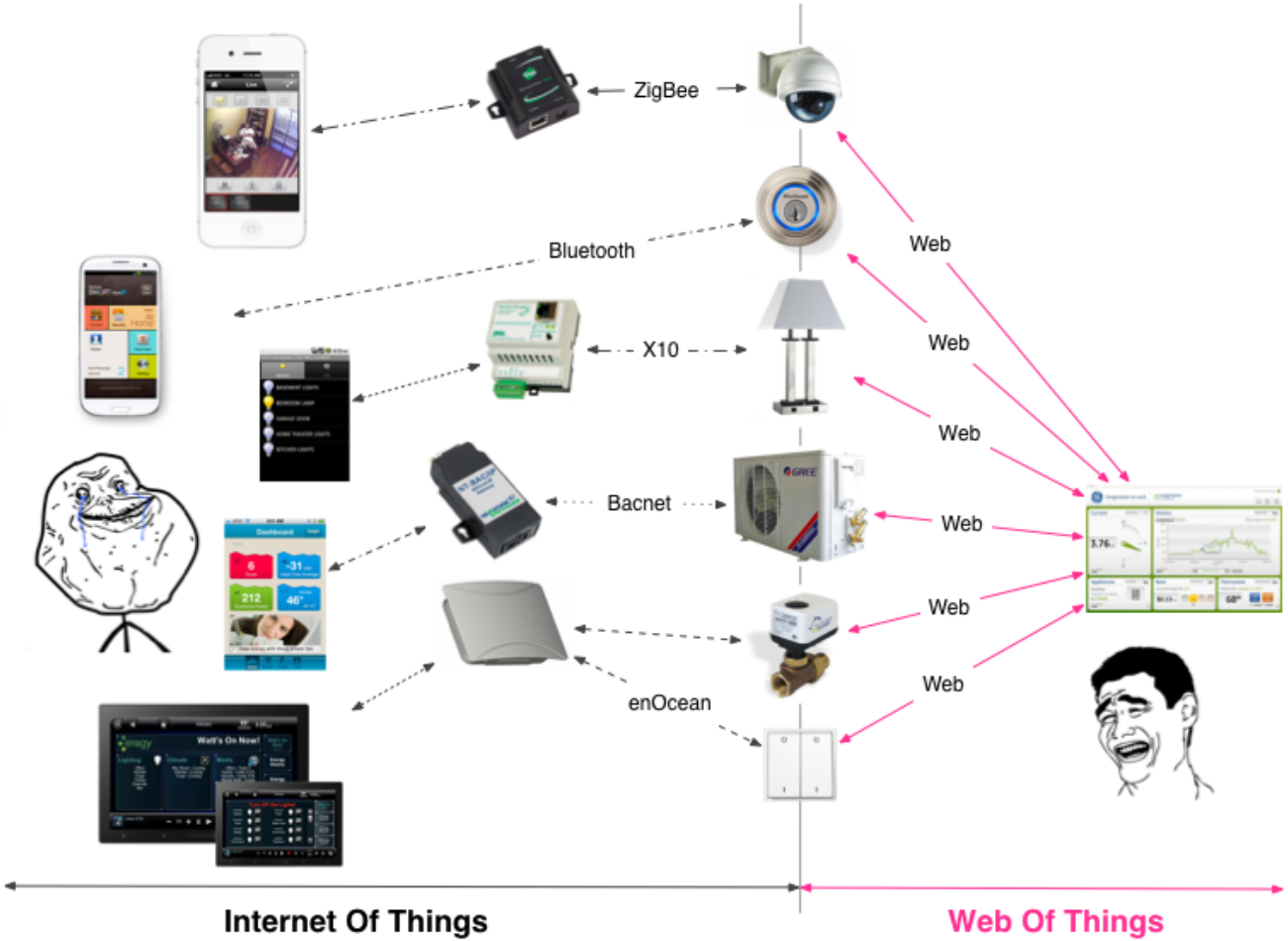
- What is a Web-Service? Why C++?
- Evolution of a real-time bidding service
- Plot twist: JSON love & hate
- Getting nerdy
- Sprinkled with some benchmarks

¿Web-Service?

http://born2invest.com/cdn/wp-content/uploads/2015/04/21.png



http://bits.citrusbyte.com/images/posts/2015-08-24-microservices-architecture.png

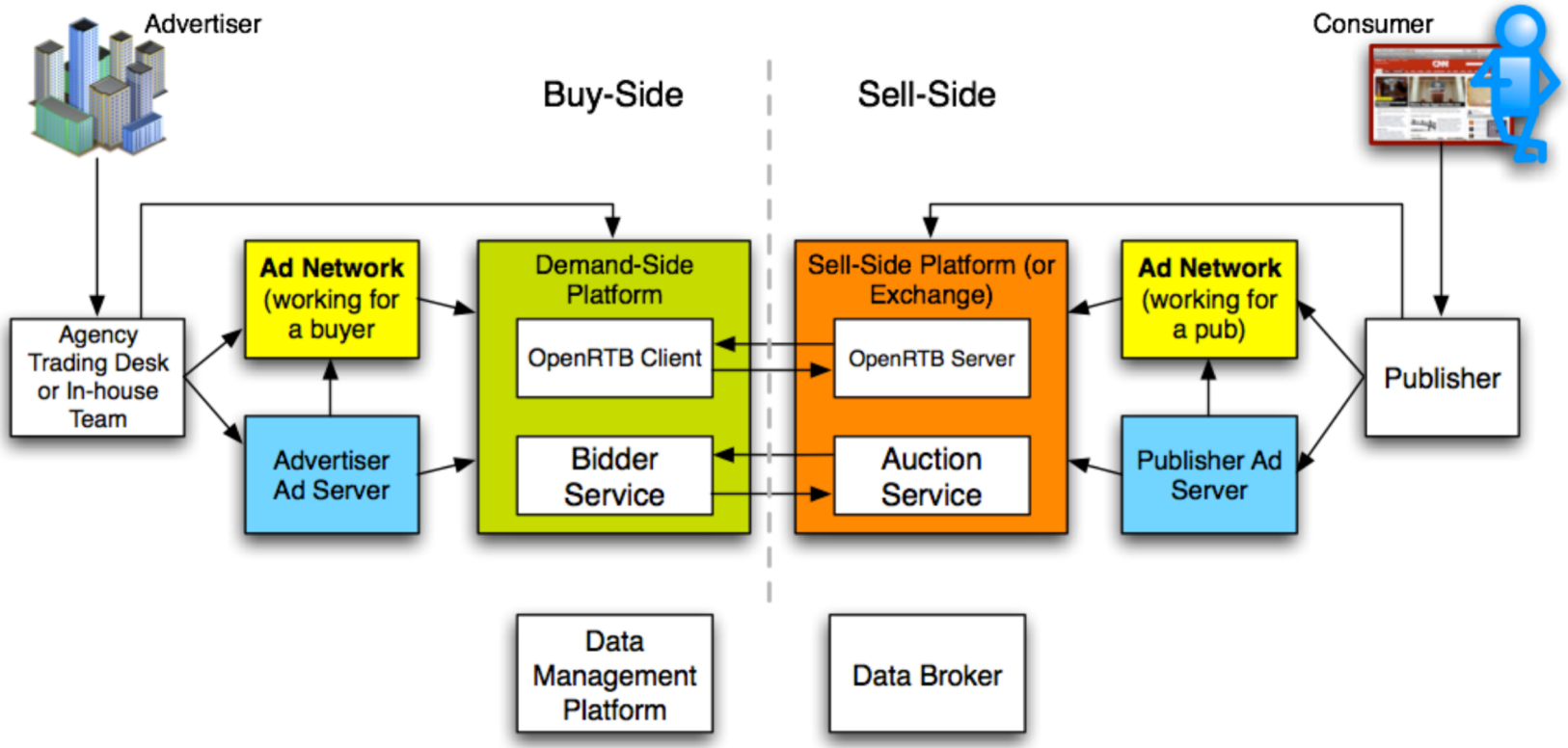


https://dzone.com/storage/temp/357042-wot-iot.png

Why Web-Services in C++?

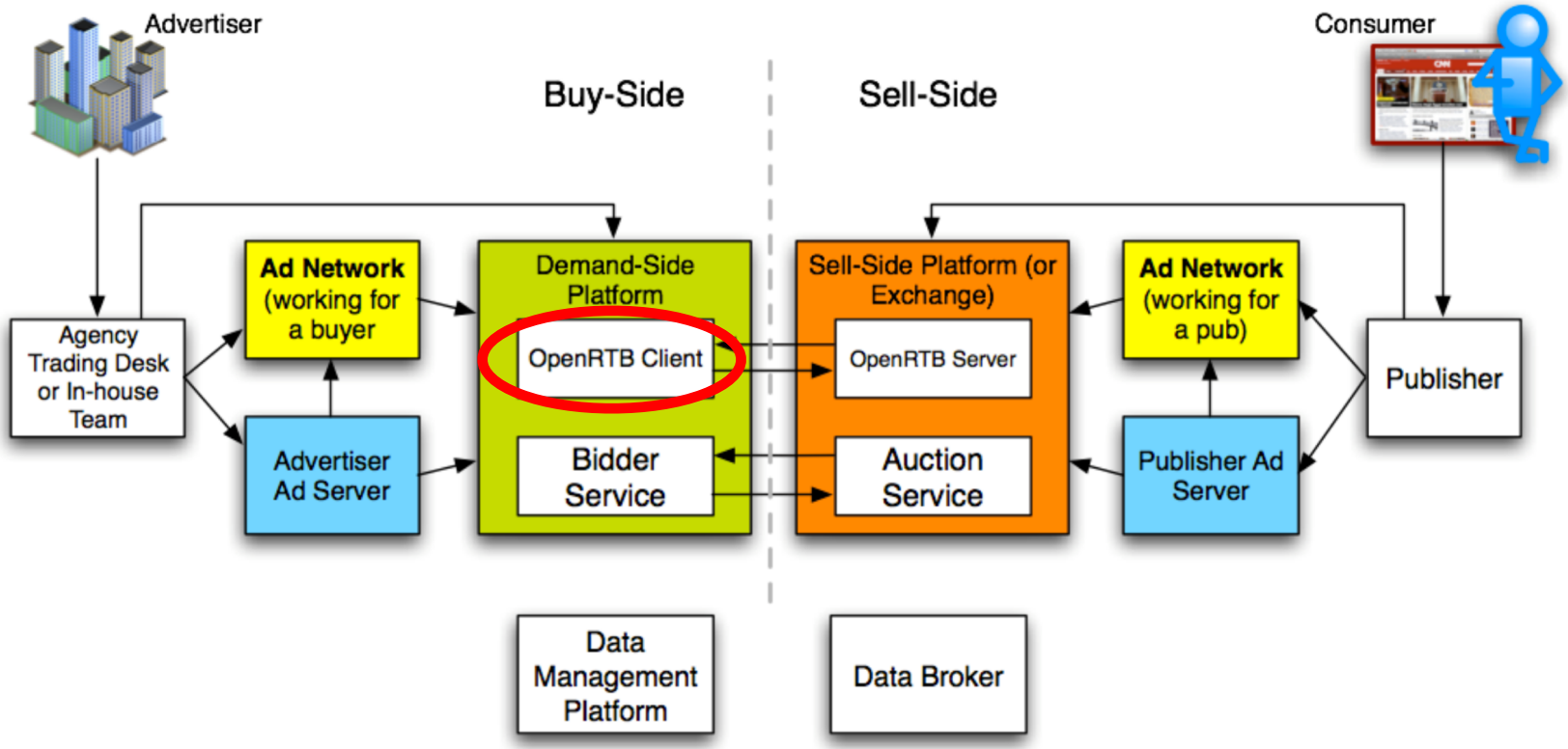
- Response time of single request (e.g. AWS Lambda)
- Flexibility, e.g. shared memory
- Scarce resources
- Many great libraries (TensorFlow, Vowpal Wabbit, OpenCV, ...)
- Existing code-base
- Extensive and mature tools
- C11 and C++11 are beautiful languages ;-)

Real-time bidding



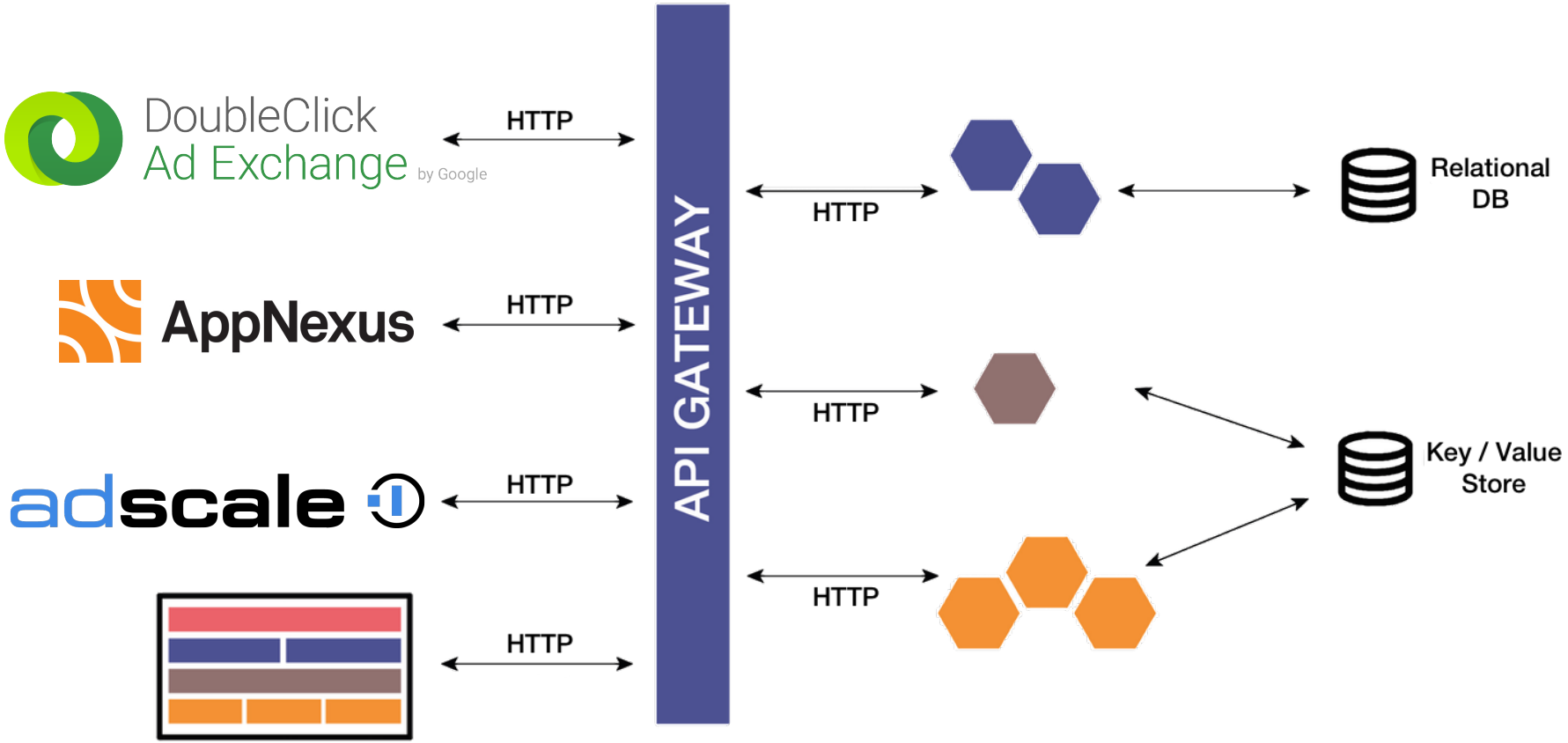
https://github.com/openrtb/OpenRTB/raw/master/OpenRTB-API-Specification-Version-2-3-1-FINAL.pdf

Real-time bidding



https://github.com/openrtb/OpenRTB/raw/master/OpenRTB-API-Specification-Version-2-3-1-FINAL.pdf

mbr targeting's real-time stack (kind of…)



http + json + /bid

HTTP

```
POST /bid HTTP/1.1
Host: localhost:8080
Content-Type: application/json
Content-Length: 13

{"id":"4223"}


HTTP/1.1 200 OK
Content-Length: 12
Content-Type: text/plain

Hello World!
```

HTTP + X

- http-parser: github.com/nodejs/http-parser
- httpp: github.com/daedric/httpp
- lwan: github.com/lpereira/lwan
- libevhtp: github.com/ellzey/libevhtp
- CivetWeb: github.com/civetweb/civetweb
- Klone: github.com/koanlogic/klone
- Proxygen: github.com/facebook/proxygen
- and many more: Mongoose, Mongrel2, libwebsockets, QtWebApp, Wt, ...

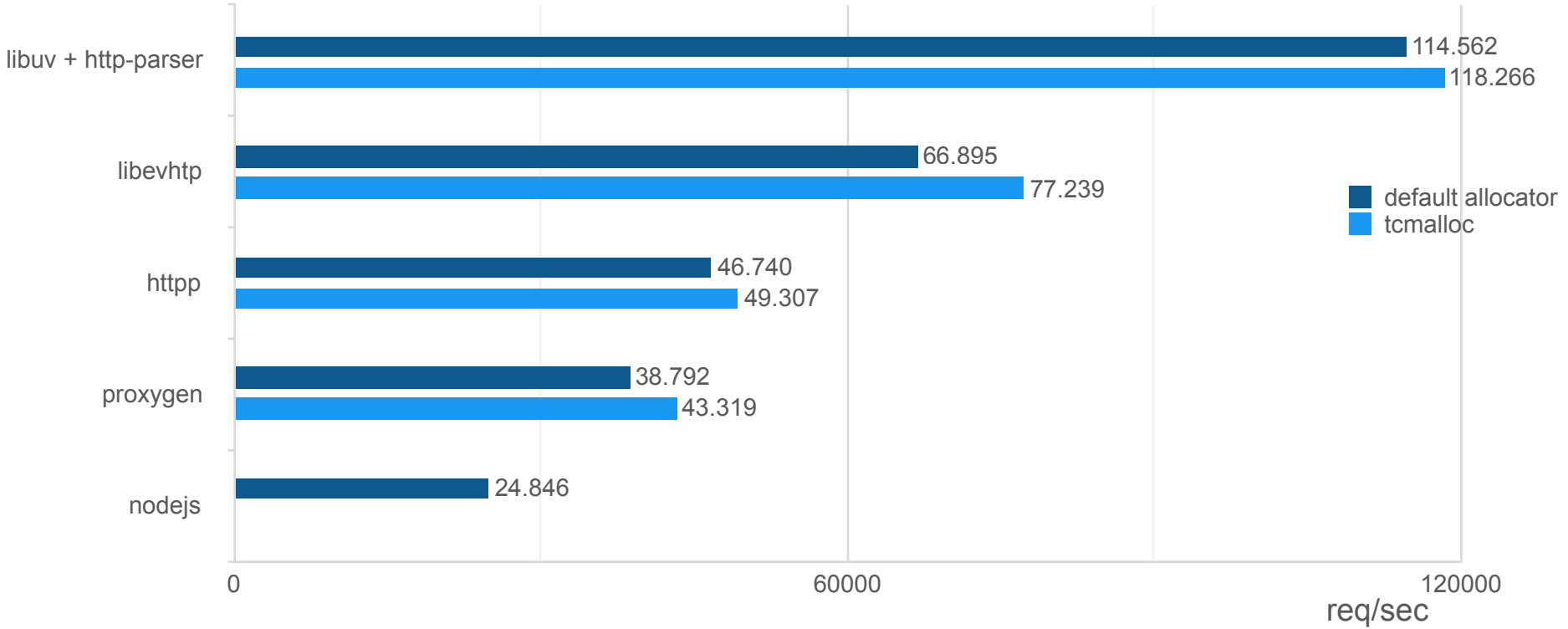
libevhtp hello world

```c
#include <evhtp.h>

void hellocb(evhtp_request_t* req, void* data) {
    evbuffer_add_printf(req->buffer_out, "Hello World!");
    evhtp_send_reply(req, EVHTP_RES_OK);
}

int main() {
    evbase_t* evbase = event_base_new();
    evhtp_t* htp = evhtp_new(evbase, NULL);
    evhtp_set_cb(htp, "/hello", hellocb, NULL);
    evhtp_use_threads_wexit(htp, NULL, NULL, 4, NULL);
    evhtp_bind_socket(htp, "0.0.0.0", 8080, 2048);
    event_base_loop(evbase, 0);
}
```

Benchmark interlude

- Never trust other people's benchmarks
- Highly recommend Brendan Gregg's talks and blog: brendangregg.com
- All code is online: github.com/0x7f/cpp-meetup
- MacBook Pro i7 2.3 GHz; wrk (c=100, t=3) POST bidrequest.2.json (~1.5kb)
- Also see abuv (github.com/0x7f/abuv), libuv-based ab drop-in replacement

Benchmark #1 – raw HTTP

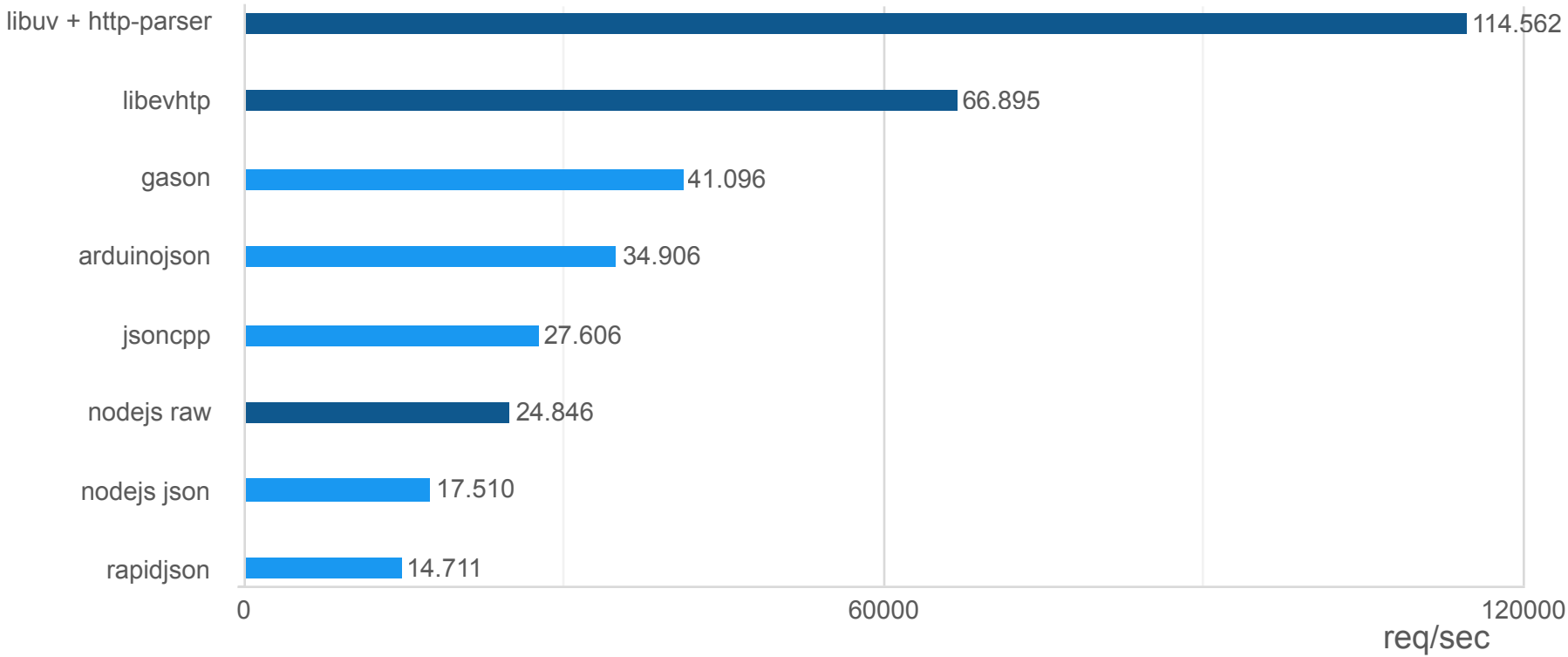


JSON

- JsonCpp github.com/open-source-parsers/jsoncpp
- RapidJSON github.com/miloyip/rapidjson
- Json11 github.com/dropbox/json11
- yajl lloyd.github.io/yajl/
- gason github.com/vivkin/gason
- ArduinoJson github.com/bblanchon/ArduinoJson

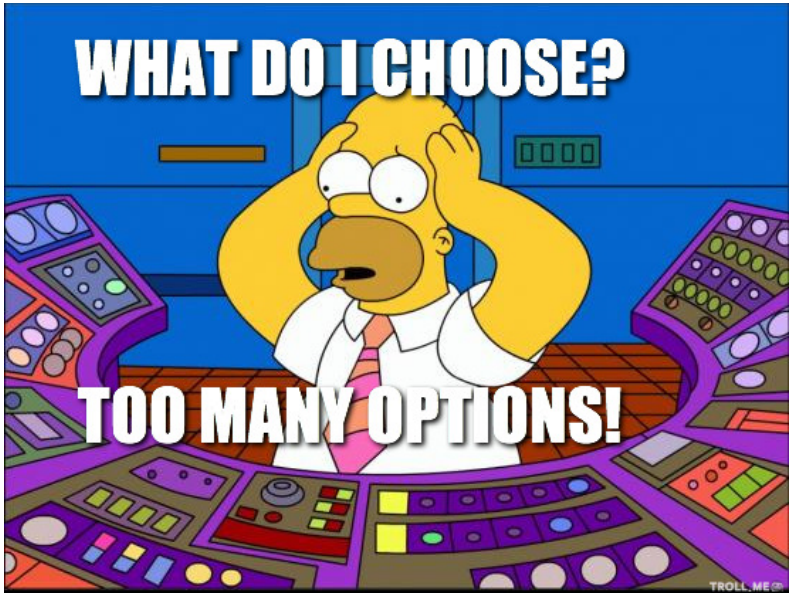
For a more complete list, e.g.: github.com/miloyip/nativejson-benchmark

Benchmark #2 – HTTP + JSON



There is no silver bullet

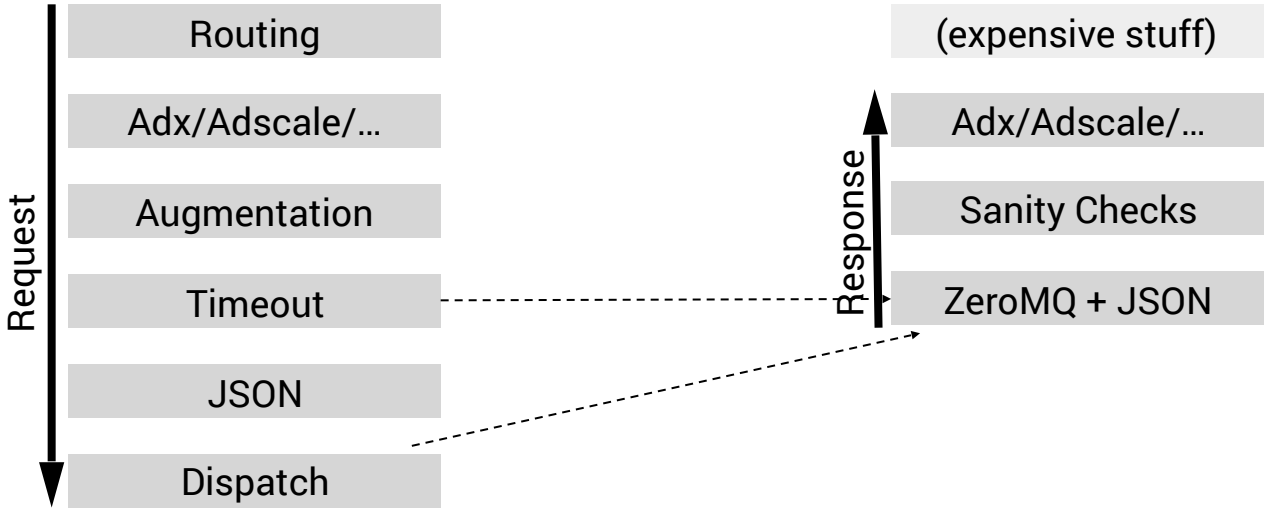
- Green field vs existing code
- Standalone vs embedded http server
- Platform & License requirements
- Multi-purpose backend vs single task
- Serialization vs meat
- Normal vs high throughput
- Hard memory constraints



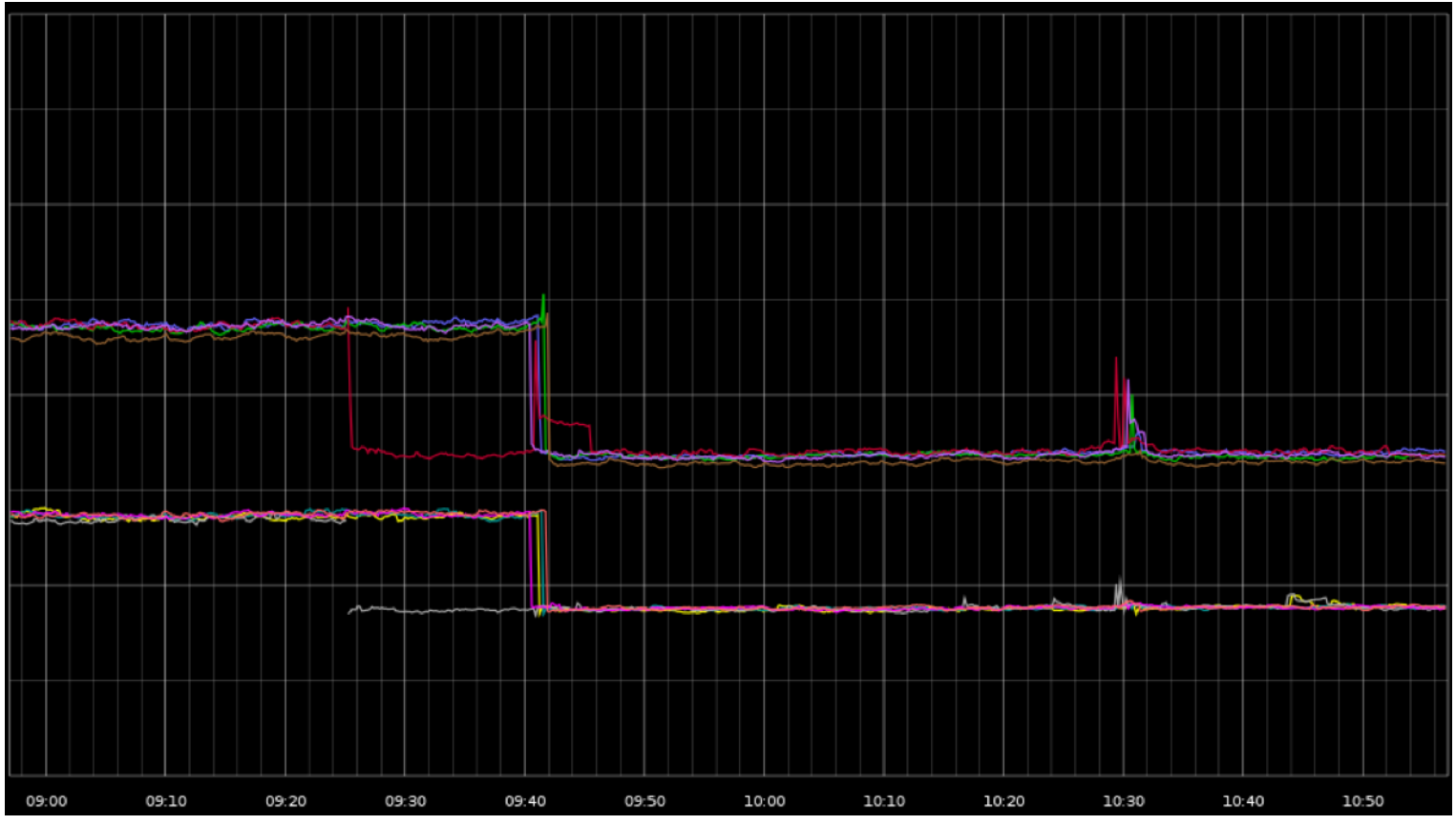
https://lisavandore.com/2015/04/17/8-popular-countertop-materials-the-pros-and-the-cons/

TODO: add some nice graphs of broker architecture

Linux, C++14 (g++ 4.8), boost::asio, httpp, JsonCpp, ZeroMQ, tcmalloc, Intel TBB



Btw: use latest JsonCpp and link it statically!



Broker evolution

- Augmentation with additional data
- Logging to Kafka
- QoS, load balancing, rate limiting
- Data scientists love data, i.e. traffic

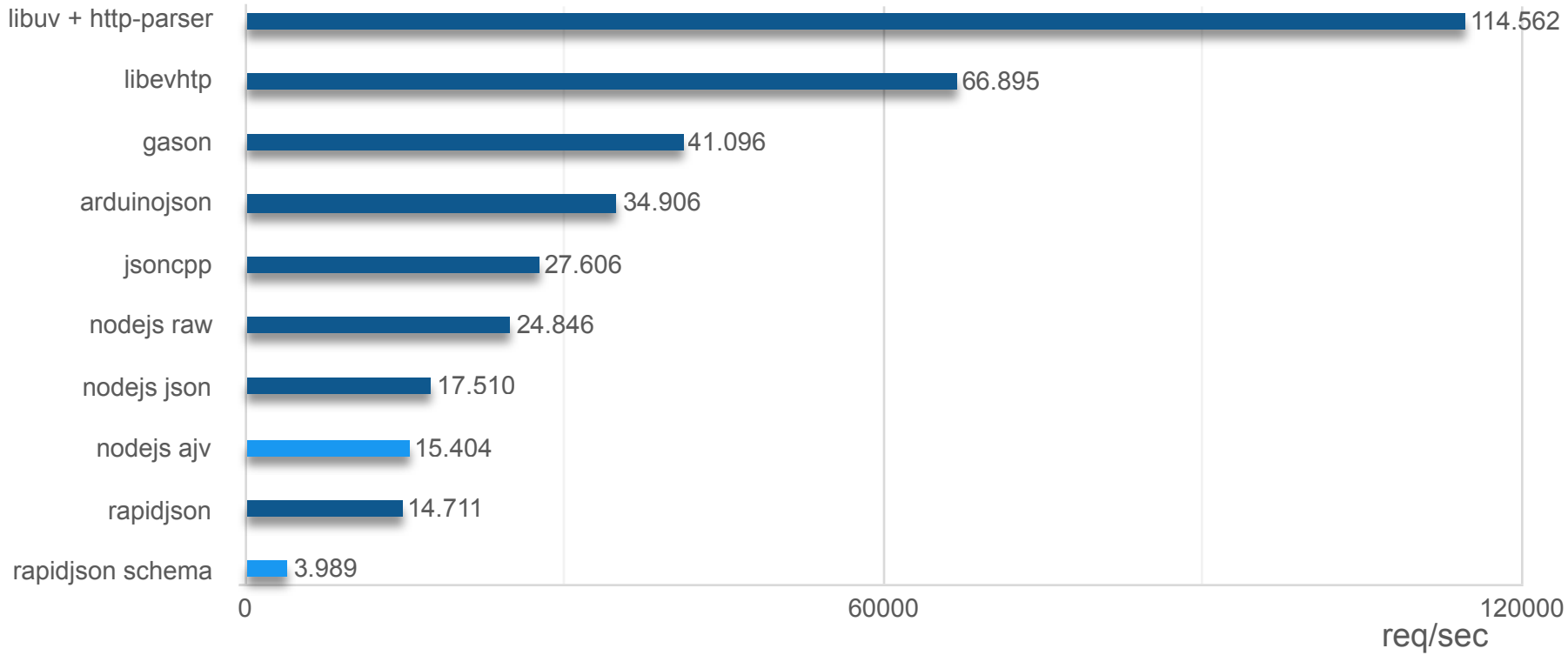
JSON love & hate

```
void doSomething(Json::Value& json) {
  const auto& v = json["myfield"];
  if (!v.isNull() && !v.isString()) {
    throw InvalidBidRequestFormatException();
  }
  if (v.isNull()) {
    return ...;
  }
  const auto i = v.asUint64();
  ...
}
```

Schemas to the rescue

- JsonSchema helps. a bit. maybe
  - Typos are still an issue
  - Knowledge about types still spread throughout the code
  - Still accessing JSON members through strings
- Start writing you own serialization
  - Structs plus hand-crafted de-/serialization code

Benchmark #3 – HTTP + JSON + Schema



Protobuf to the rescue!!!11elf

```
option java_outer_classname = "Example";
package io.echtzeit.example;

message MsgA {
  optional int32 a = 1;
  required MsgB b = 2;
}

message MsgB {
  required string c = 1;
}
```

Protobuf to the rescue!!!11elf – generated Classes

```
option java_outer_classname = "Example";
package io.echtzeit.example;

message MsgA {
  optional int32 a = 1;
  required MsgB b = 2;
}

message MsgB {
  required string c = 1;
}
```

```
// optional int32 a = 1;
inline bool has_a() const;
inline void clear_a();
static const int kAFieldNumber = 1;
inline ::google::protobuf::int32 a() const;
inline void set_a(::google::protobuf::int32 value);

// required .io.echtzeit.example.MsgB b = 2;
inline bool has_b() const;
inline void clear_b();
static const int kBFieldNumber = 2;
inline const ::io::echtzeit::example::MsgB& b() const;
inline ::io::echtzeit::example::MsgB* mutable_b();
inline ::io::echtzeit::example::MsgB* release_b();
inline void set_allocated_b(::io::echtzeit::example::MsgB* b);
```

Protobuf to the rescue!!!11elf – Reflection

```
Message* msg = new MsgA();
const Descriptor* descriptor = msg->GetDescriptor();

const FieldDescriptor* numbers_field = descriptor->FindFieldByName("a");
assert(numbers_field != NULL);
assert(numbers_field->type() == FieldDescriptor::TYPE_INT32);

// Parse the message.
msg->ParseFromString(data);

const Reflection* reflection = msg->GetReflection();
assert(reflection->GetInt32(msg, numbers_field) == 1);
```

Protobuf to the rescue!!!11elf – Serialization

```
// Parse a protocol buffer from a file descriptor.  If successful, the entire
// input will be consumed.
bool ParseFromFileDescriptor(int file_descriptor);
// Like ParseFromFileDescriptor(), but accepts messages that are missing
// required fields.
bool ParsePartialFromFileDescriptor(int file_descriptor);
// Parse a protocol buffer from a C++ istream.  If successful, the entire
// input will be consumed.
bool ParseFromIstream(istream* input);
// Like ParseFromIstream(), but accepts messages that are missing
// required fields.
bool ParsePartialFromIstream(istream* input);

// Serialize the message and write it to the given file descriptor.  All
// required fields must be set.
bool SerializeToFileDescriptor(int file_descriptor) const;
// Like SerializeToFileDescriptor(), but allows missing required fields.
bool SerializePartialToFileDescriptor(int file_descriptor) const;
// Serialize the message and write it to the given C++ ostream.  All
// required fields must be set.
bool SerializeToOstream(ostream* output) const;
// Like SerializeToOstream(), but allows missing required fields.
bool SerializePartialToOstream(ostream* output) const;
```

Protobuf to the rescue!!!11elf – Message::Clear()

```
MsgA* New() const;
void CopyFrom(const ::google::protobuf::Message& from);
void MergeFrom(const ::google::protobuf::Message& from);
void CopyFrom(const MsgA& from);
void MergeFrom(const MsgA& from);
void Clear();
bool IsInitialized() const;
```

Protobuf to the rescue!!!11elf – Documentation

- Thanks Google: [openrtb.proto (github.com/google/openrtb)]

```
message BidRequest {
  // Unique ID of the bid request, provided by the exchange.
  // REQUIRED by the OpenRTB specification.
  // [AdX: BidRequest.id]
  required string id = 1;

  // Array of Imp objects (Section 3.2.2) representing the impressions offered.
  // At least 1 Imp object is required.
  // [AdX: BidRequest.AdSlot]
  repeated Imp imp = 2;

  oneof distributionchannel_oneof {
```

JSON to Protobuf and back

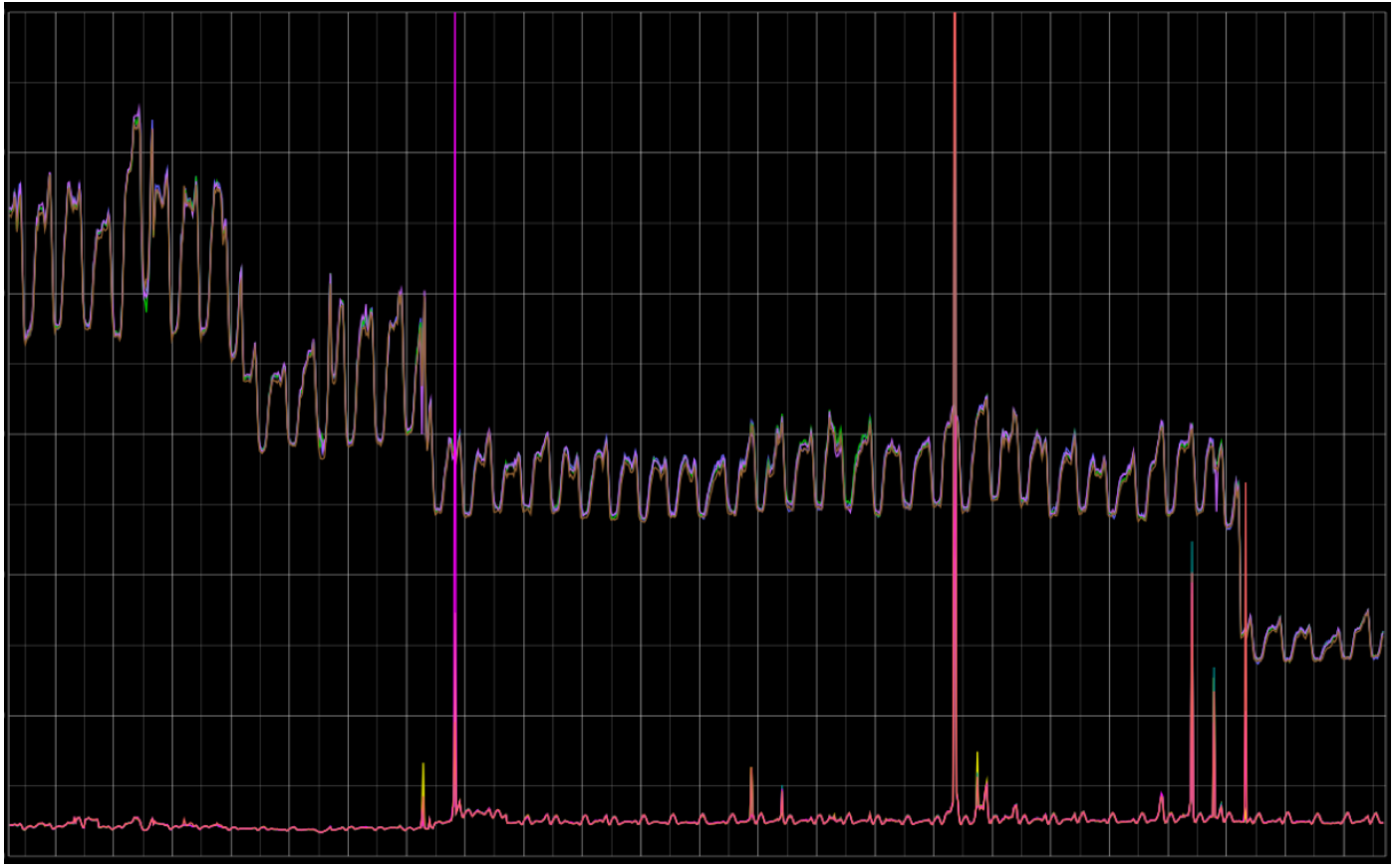
- Google Protocol Buffers 3: google/protobuf/util/json_util.h
- json-protobuf mapper using the generated reflection
  - github.com/yinqiwen/pbjson (RapidJSON based)
  - github.com/shramov/json2pb (jansson based)
  - github.com/shafreeck/pb2json (jansson based)
  - github.com/bivas/protobuf-java-format (Java)
  - github.com/mafintosh/protocol-buffers (node.js)

→ Single place for schema validation and performance improvements

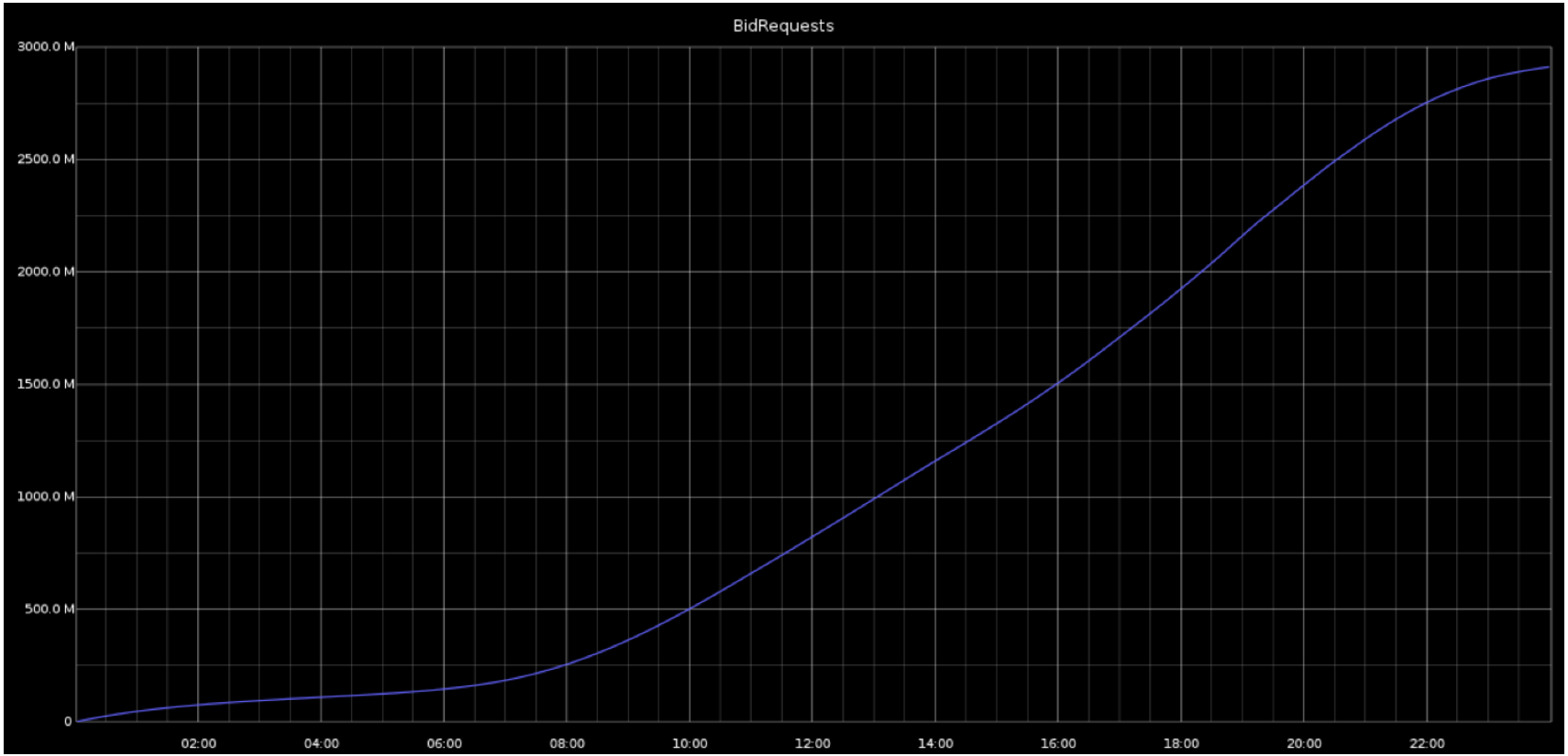
JSON to Protobuf and back

```
BidRequest bidRequest;
auto result = JsonStringToMessage(body, &bidRequest);
if (result.ok()) {
    auto greeting = "Hello " + bidRequest.id();
    response.setCode(HttpCode::Ok).setBody(greeting);
} else {
    response.setCode(HttpCode::BadRequest)
        .setBody(std::string(result.error_message()));
}
```

Drop the ~~bass~~ response times



Now ~3 billion requests/day



Let it sink

...

Use Protobuf for all data structures!

http://knowyourmeme.com/memes/x-all-the-y

Sky is the limit

- Still more memory used than needed (raw buffer + JSON + Protobuf)
- Use event-based JSON parser (e.g. yajl or RapidJson) for memory-efficiency
- Get rid of reflection (e.g. by also generating parser code)

Generate JSON state machine from proto file

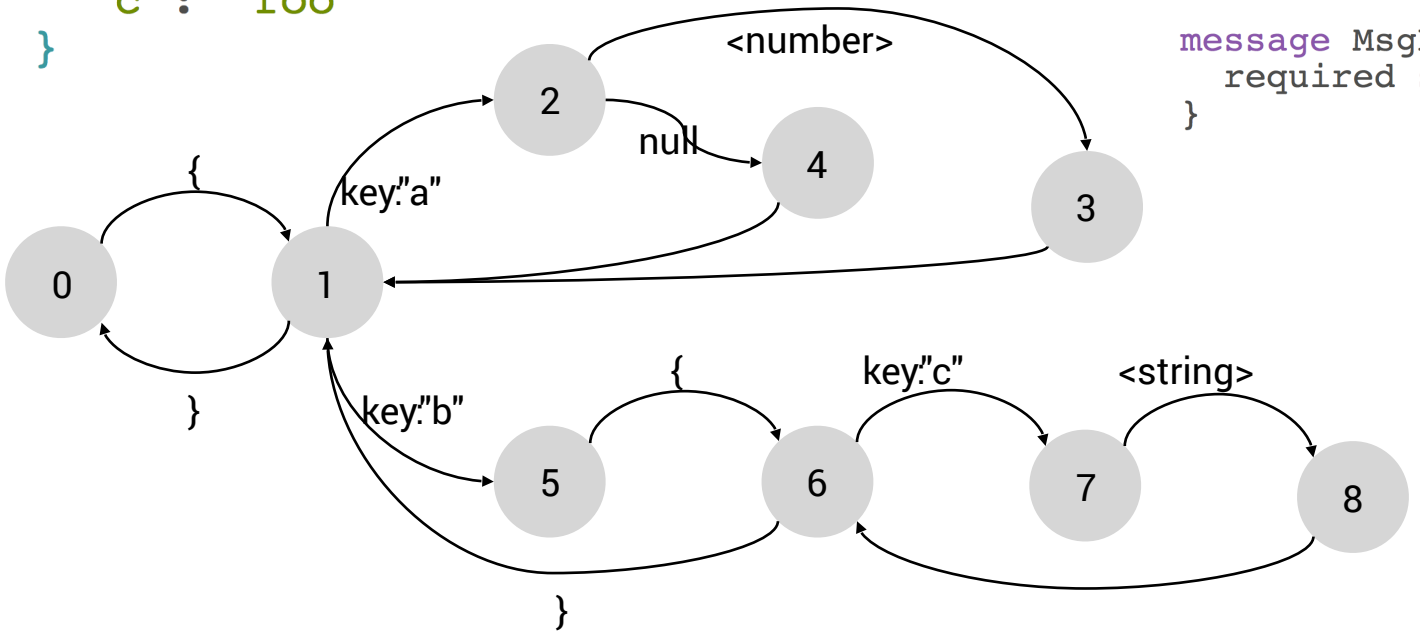
```
{
  "a": 100,
  "b": {
    "c": "foo"
  }
}
```

```
message MsgA {
  optional int32 a = 1;
  required MsgB b = 2;
}

message MsgB {
  required string c = 1;
}
```

Generate JSON state machine from proto file

```
{
  "a": 100,
  "b": {
    "c": "foo"
  }
}
```

```
message MsgA {
  optional int32 a = 1;
  required MsgB b = 2;
}

message MsgB {
  required string c = 1;
}
```



Proof-of-concept: protog

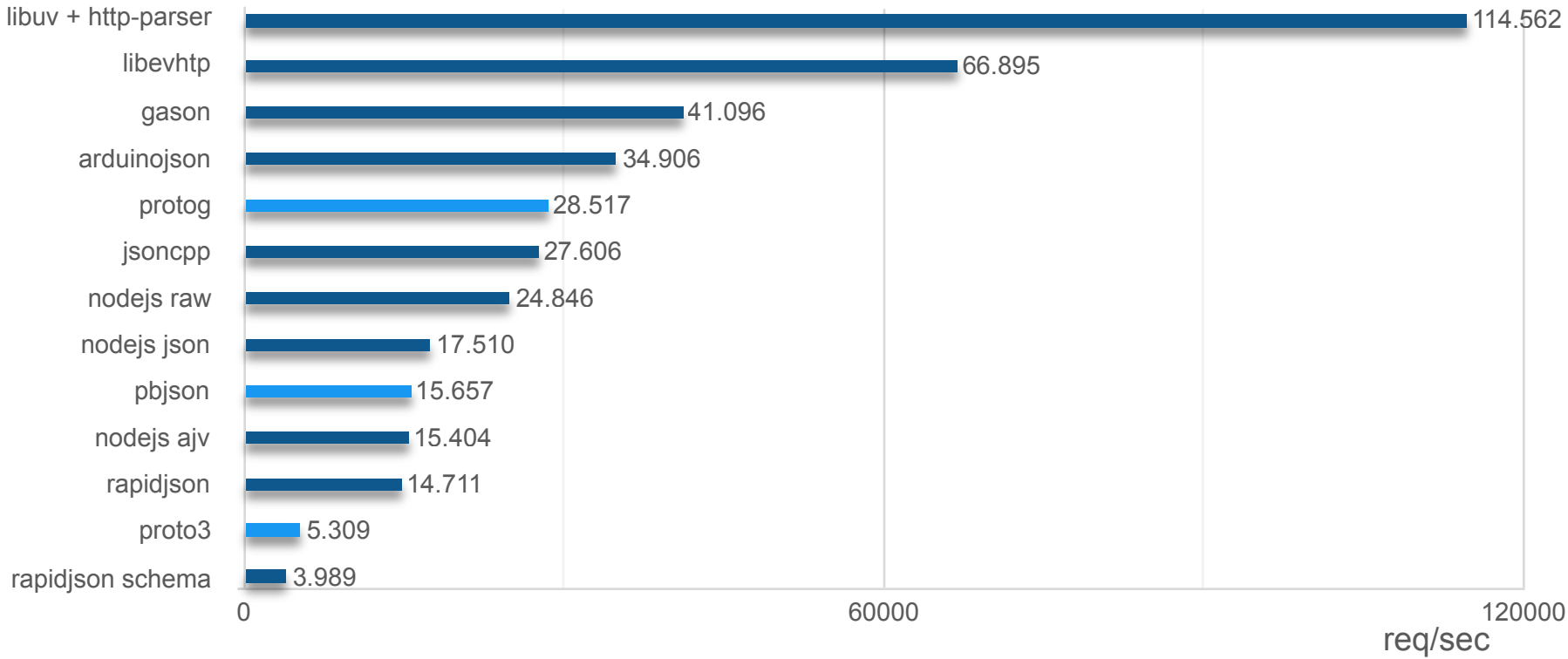
A protobuf json-parser generator!

https://github.com/0x7f/protog

```
$ protoc --cpp_out=. openrtb.proto
$ protog -p openrtb.proto -i openrtb.pb.h \
    -m com.google.openrtb.BidRequest
```

Will generate `openrtb.pb.{cc,h}` and `bidrequest_parser.pb.{cc,h}`

Benchmark #4 – HTTP + JSON + Protobuf



Summary

- Measure! Measure! Measure!
- Web-services are more than HTTP+JSON
- Mastering JSON is hard
- C++ is not fast by default
- But can be very fast when used correctly
- No silver bullet – heavily depends on the use-case

Questions?

Maximilian Haupt
mhaupt@echtzeitstudios.com
github.com/0x7f