gRPC Performance

Tuning Applications and Libraries

Noah Eisen (email, github, twitter handle: ncteisen)

Agenda

- gRPC Overview
- Tooling, Benchmarks, and Data
- Tuning the gRPC Library
  - Undoing Death by 1000 Paper Cuts
  - Case Study
- Breaking Down the Layers
- Tuning gRPC Applications
  - Low Hanging Fruit
  - Case Study

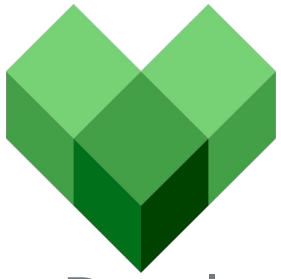
gRPC Overview

gRPC Overview - History

Borg → 

Kubernetes

TensorFlow

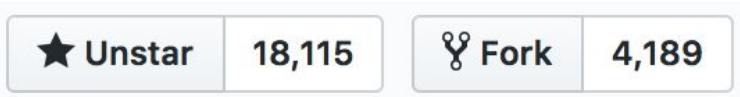
Blaze → 

Bazel

Stubby → 

gRPC Overview - Basics

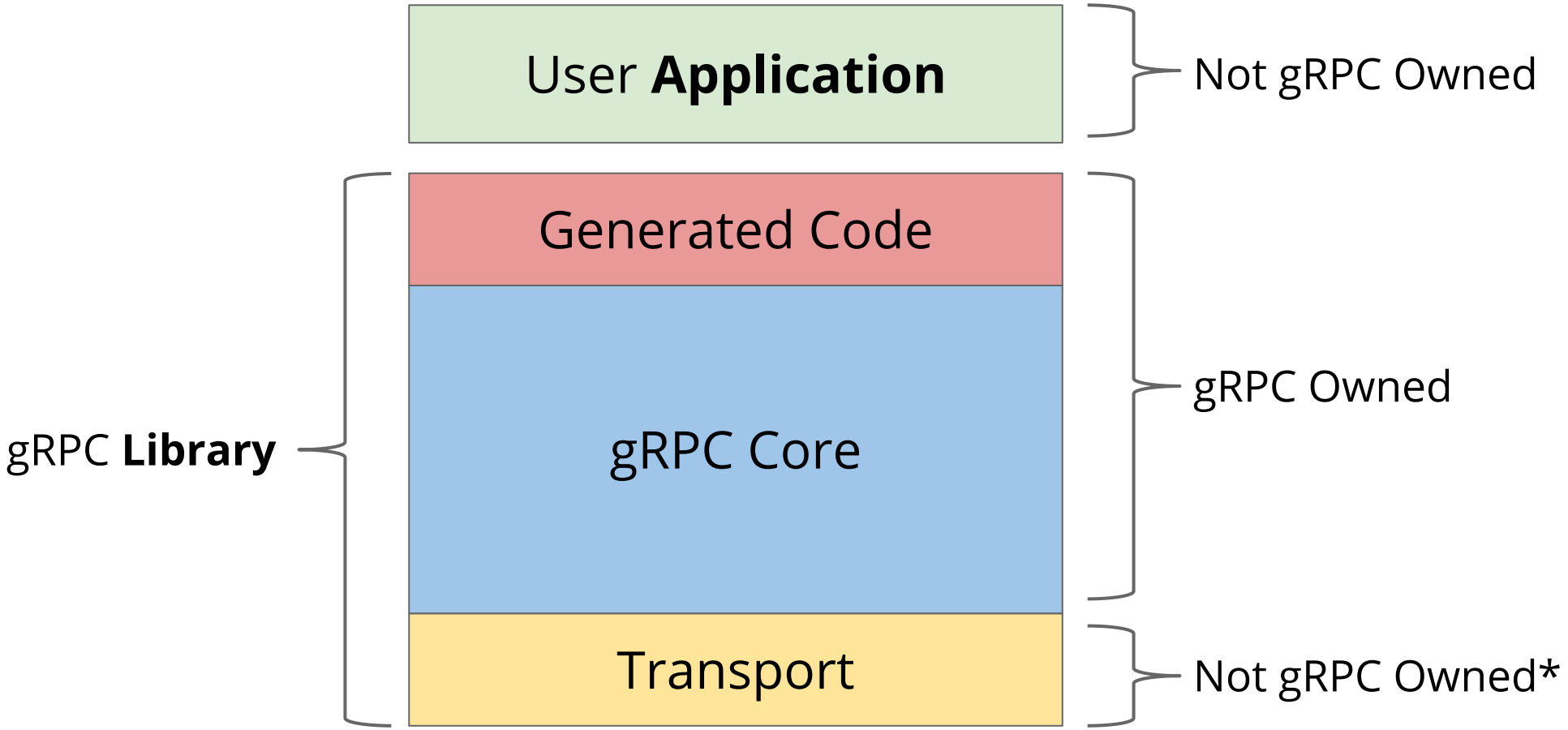
gRPC stands for **g**RPC **R**emote **P**rocedure **C**all.



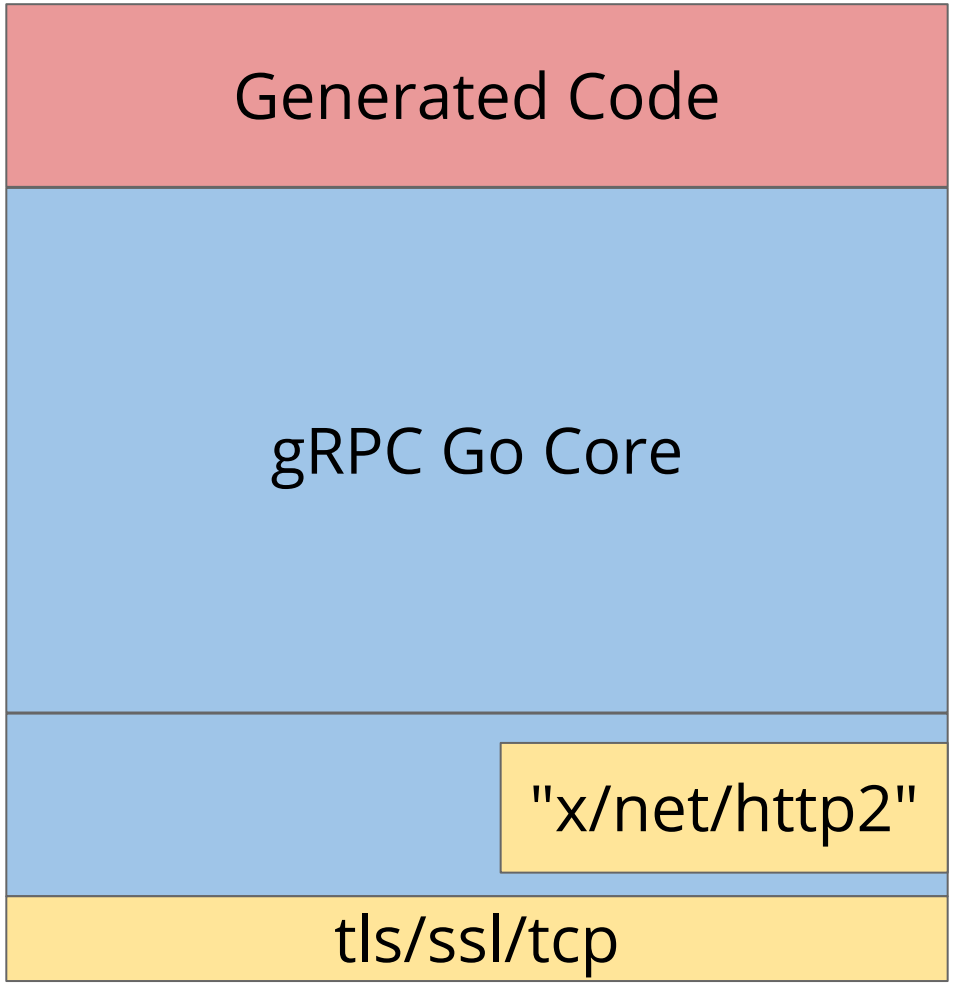
A **high performance**, open source, standards based, general purpose, polyglot, feature-rich RPC framework.

Actively developed and production-ready.

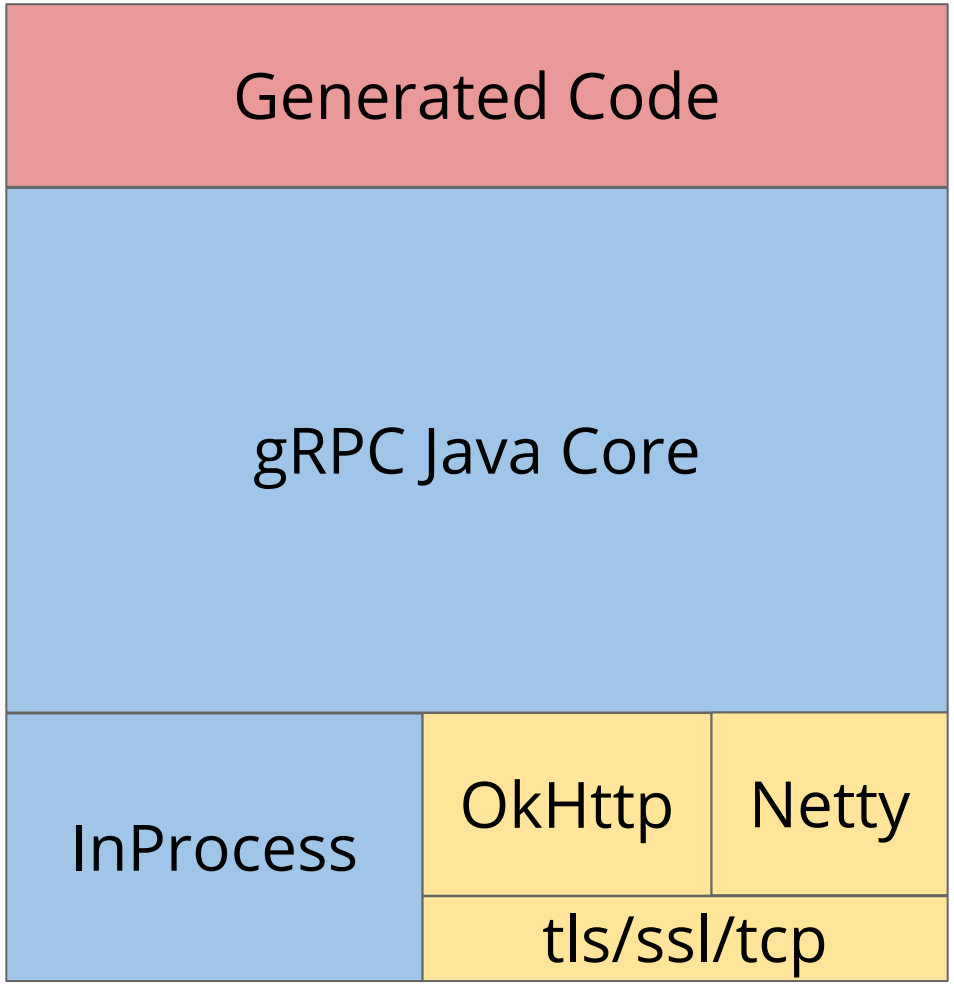
gRPC Overview - Generic Stack



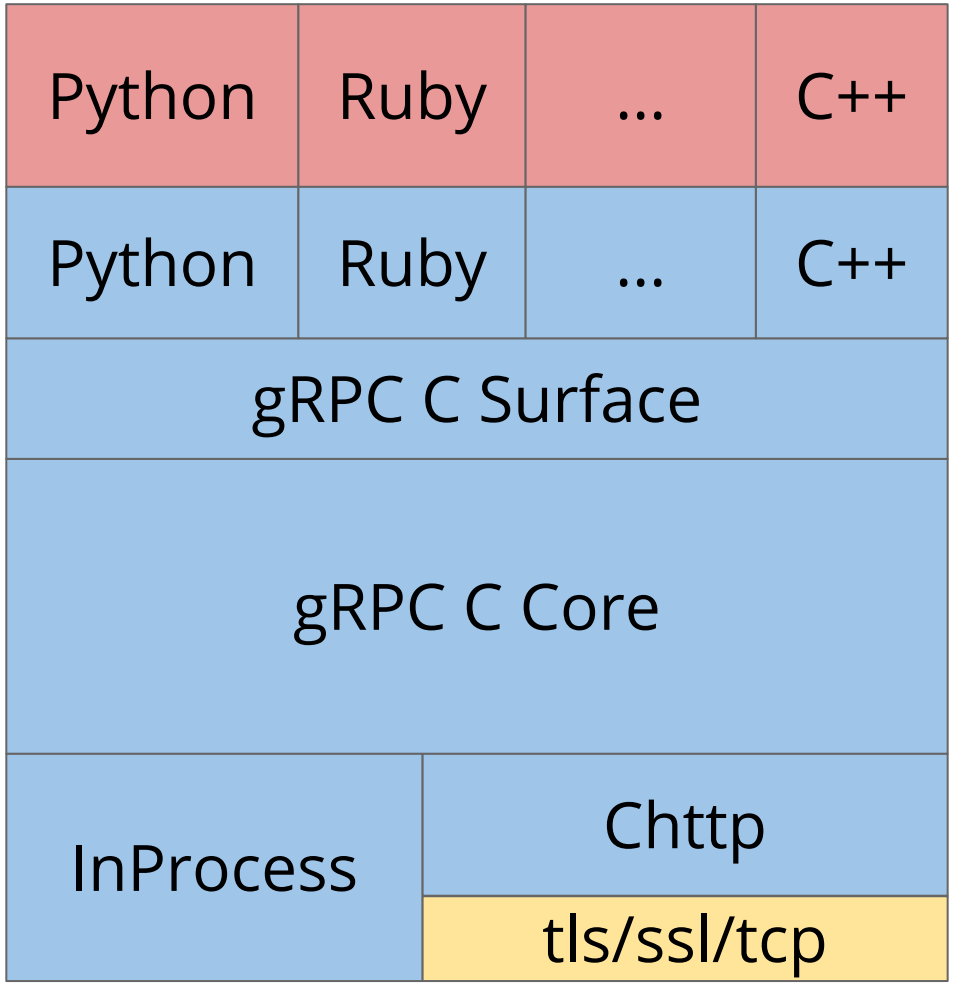
gRPC Overview - Go Stack



gRPC Overview - Java Stack



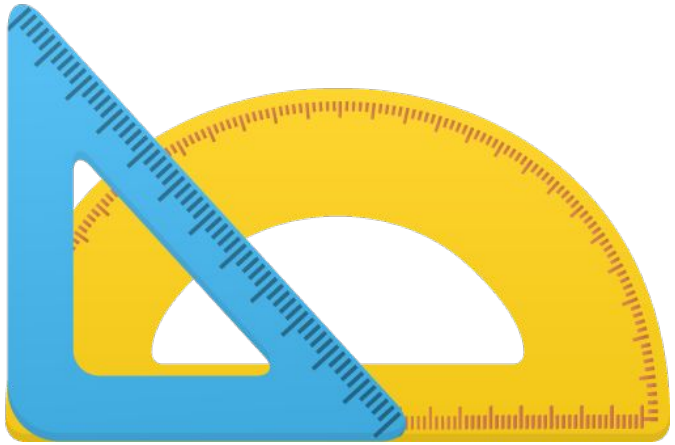
gRPC Overview - C Stack



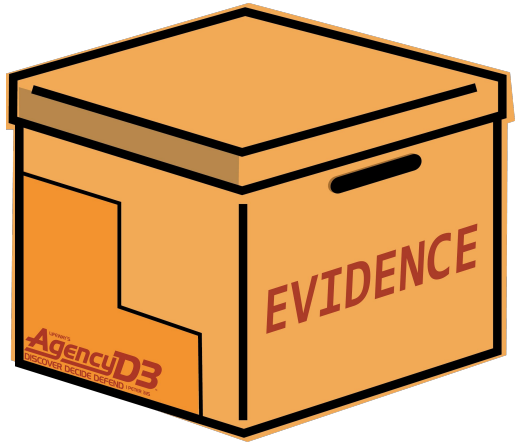
Tuning Libraries - Key Points

Tooling

Benchmarks

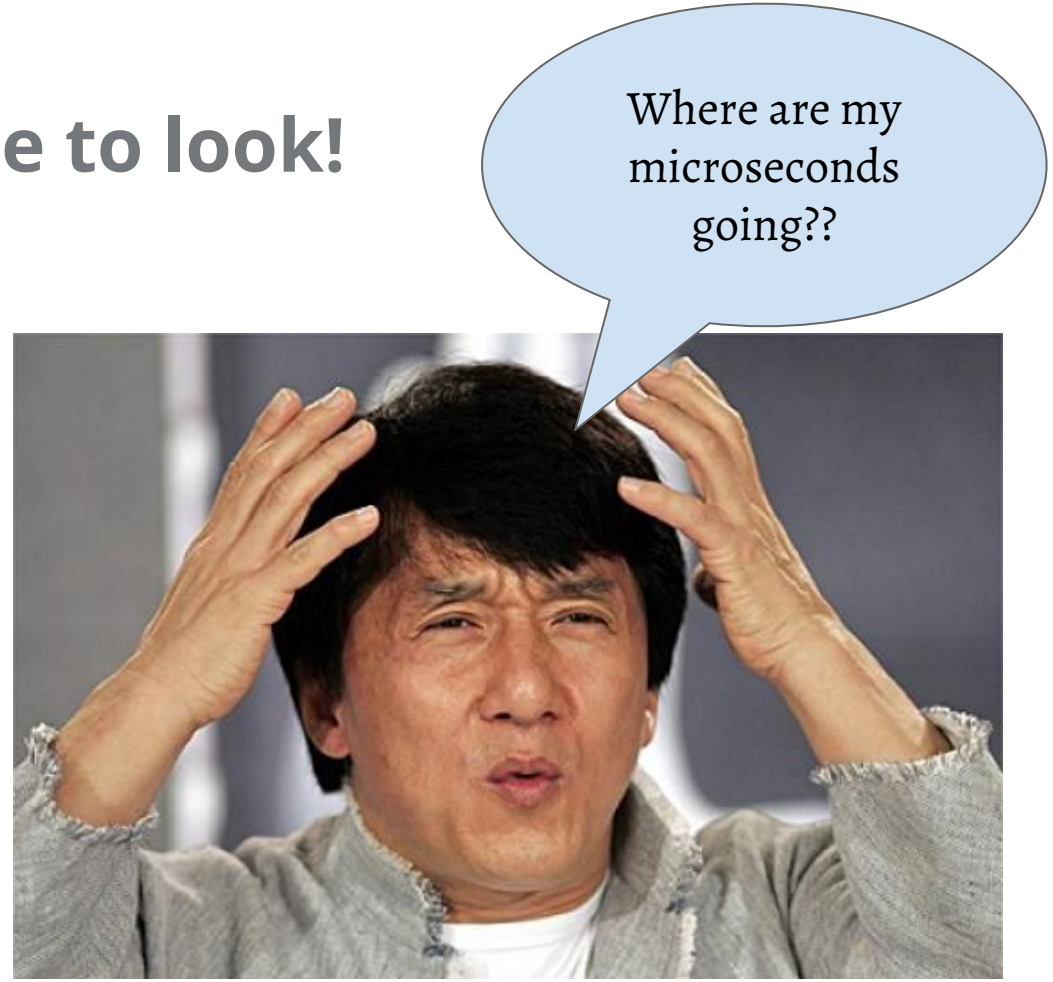


Data



Tooling

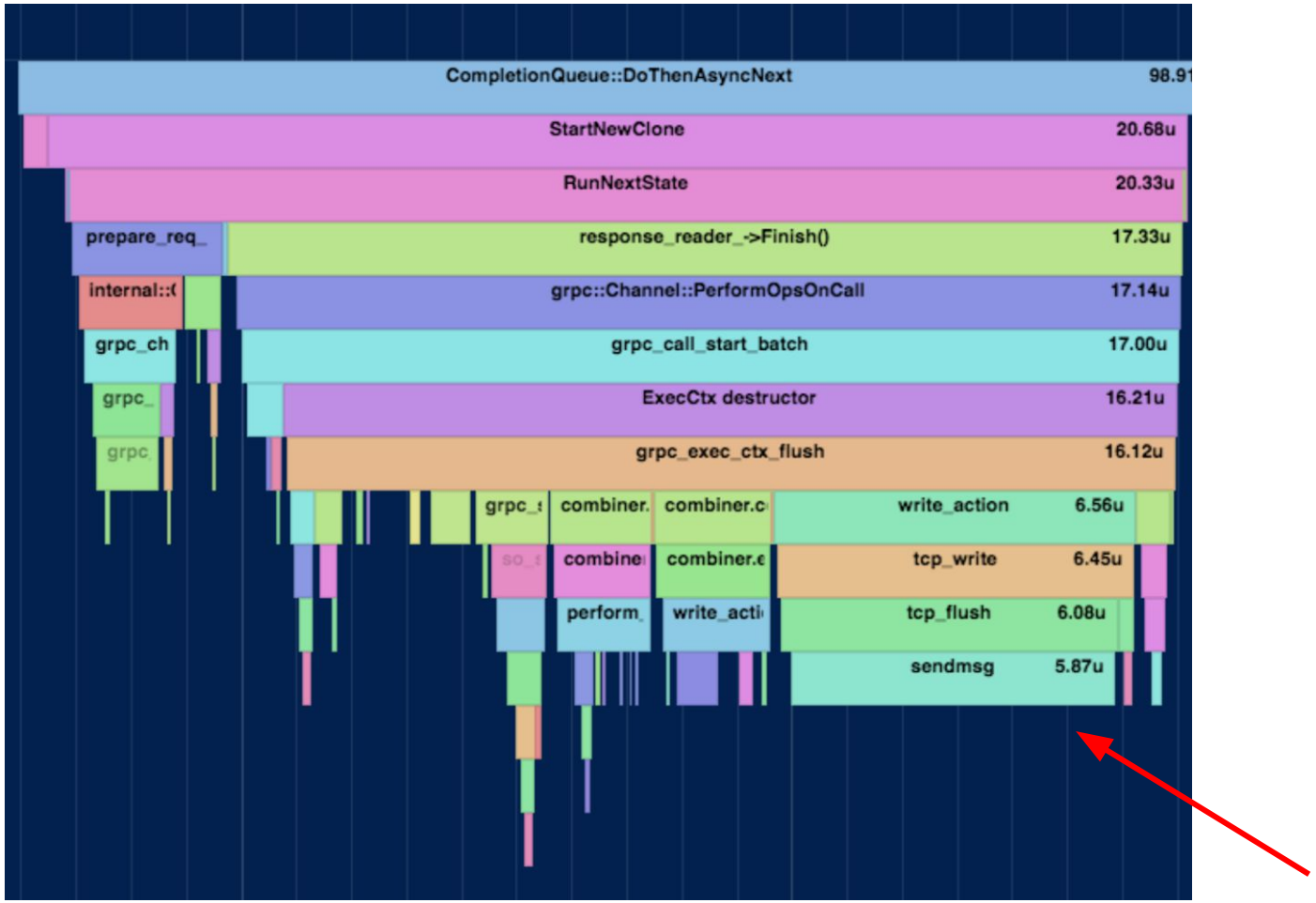
- **In order to optimize, know where to look!**
- Tooling narrows problem scope.
- No such thing as "perfect tool".



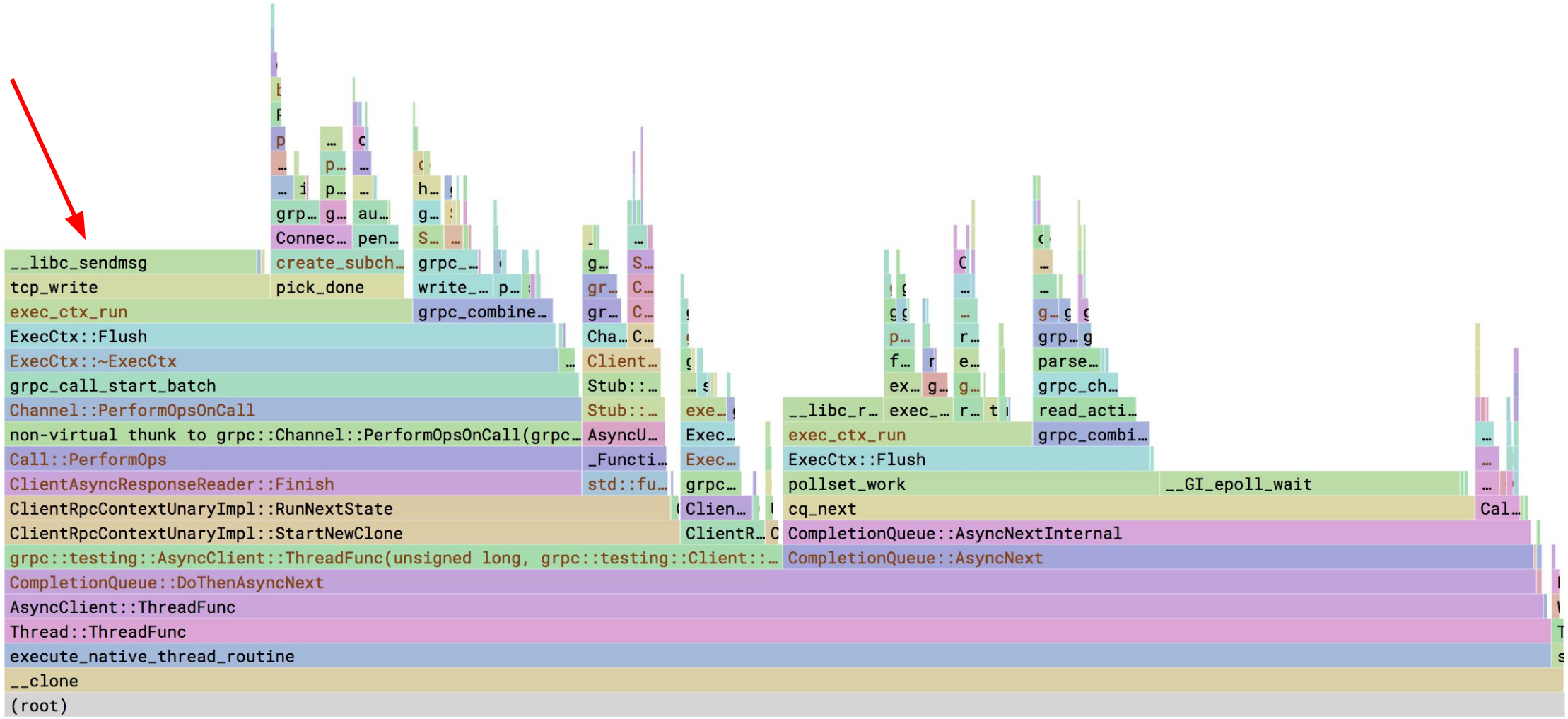
Tooling - Latency Traces


```
void foo() {
  GPR_TIMER_SCOPE("foo");
  bar();
}

void bar() {
  GPR_TIMER_SCOPE("bar");
  do_more_work();
}
```



Tooling - CPU Profiles

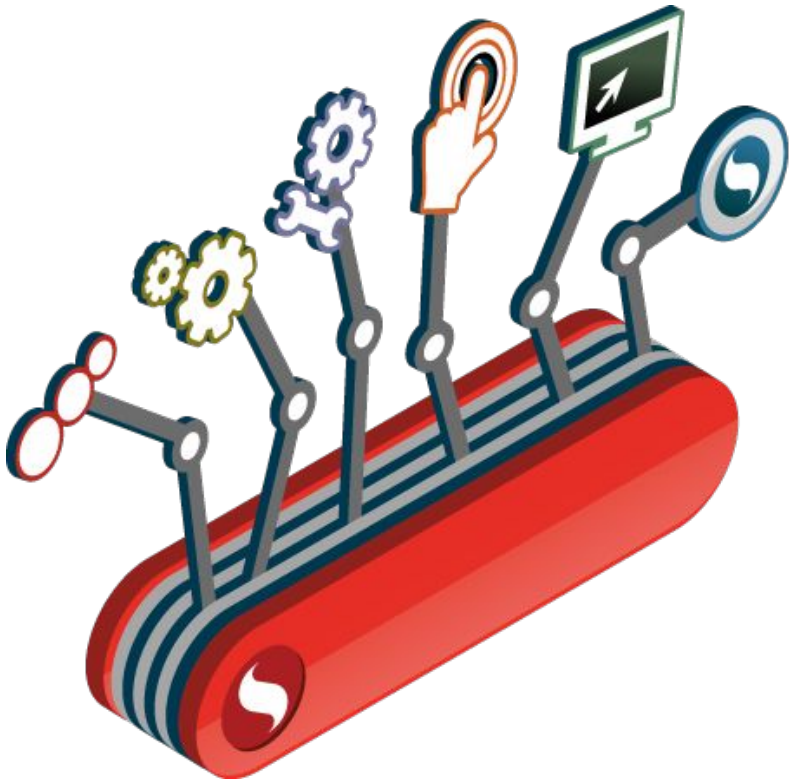


Tooling - Other Tools

- Lock contention measuring tools (mutrace)
- Customs counters for allocs, atomics.
- Kernel tools:
  - perf (general analysis)
  - strace (syscall introspection)
  - pahole (c++ struct packing)

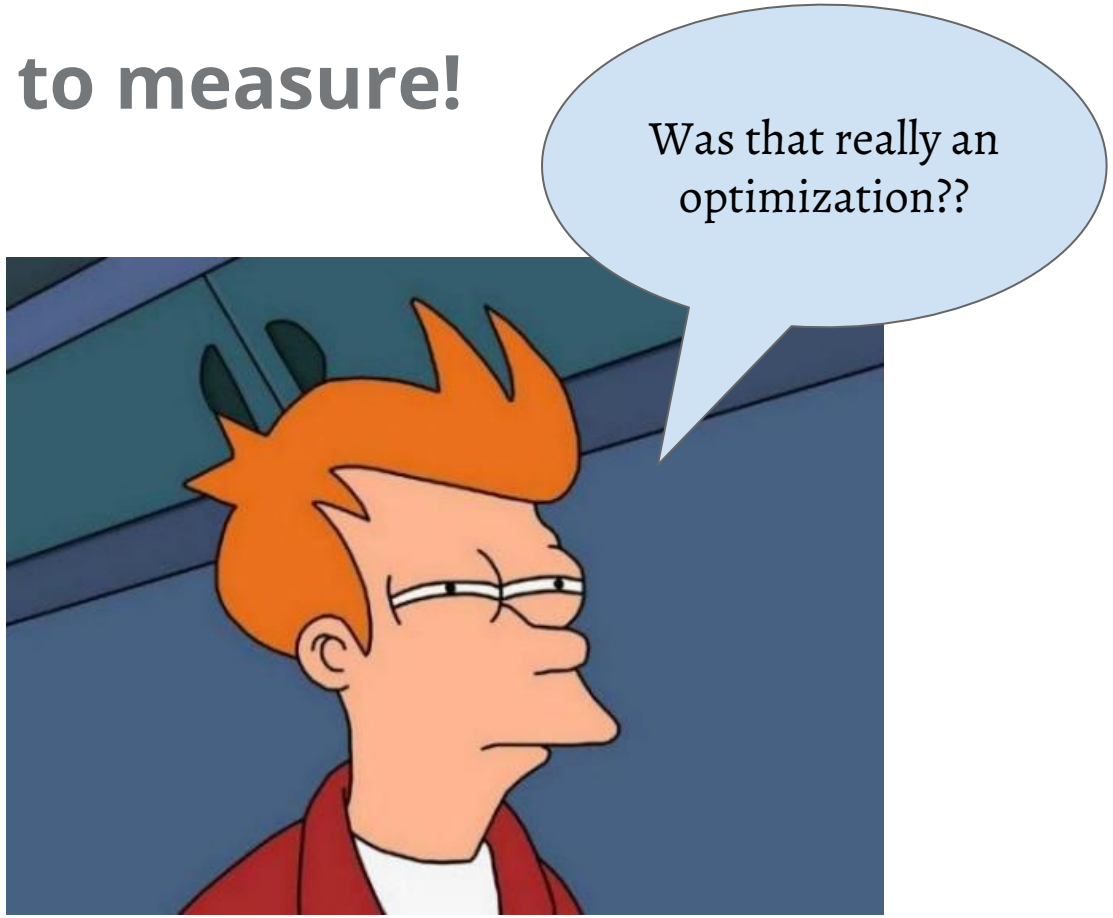
Tooling - Bottom Line

- Obtain an **arsenal of tools**
- Use tools in conjugation
- Grow your arsenal



Benchmarks

- **In order to optimize, know how to measure!**
- Benchmarks widen scope.



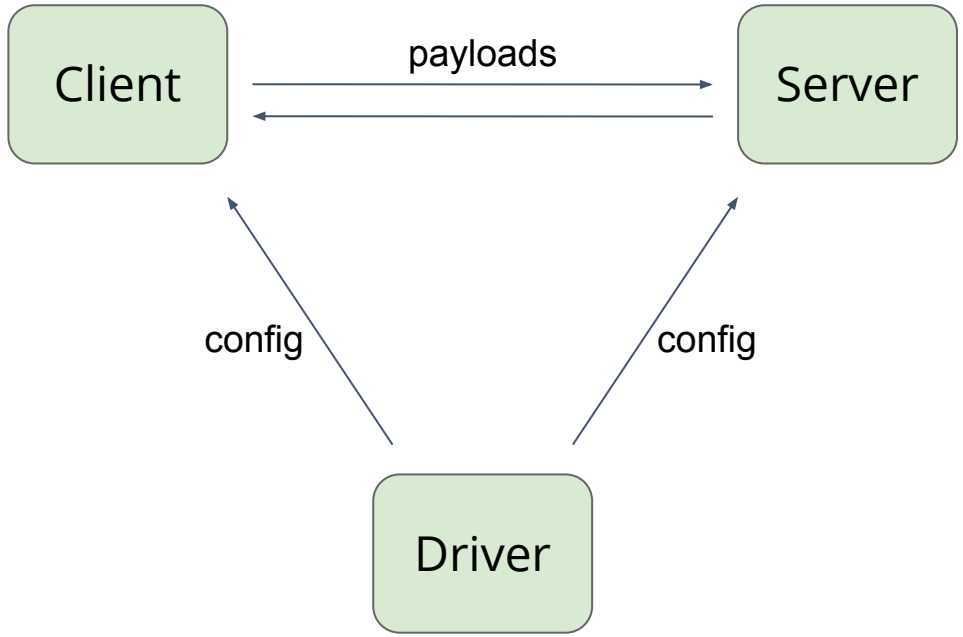
Benchmarks - Microbenchmarks


```
static void BM_ErrorCreate(State& state) {
  while (state.KeepRunning()) {
    GRPC_ERROR_UNREF(GRPC_ERROR_CREATE("Error"));
  }
}
BENCHMARK(BM_ErrorCreate);
```

```
Run on (12 X 3800 MHz CPU s)
------------------------------------------------------------------------------
Benchmark                                       Time           CPU Iterations
------------------------------------------------------------------------------
BM_ErrorCreate                                119 ns        118 ns    5516723
```

Benchmarks - Synthetic


```
{
  "name":"1-channel-1-byte",
  "warmup_seconds":30,
  "benchmark_seconds":120,
  "num_servers":1,
  "server_config":{
    "async_server_threads":1,
    "server_type":"ASYNC_SERVER"},
  "num_clients":1,
  "client_config":{
    "client_type":"ASYNC_CLIENT",
    "payload_config":{
      "simple_params":{
        "resp_size":1,
        "req_size":1}},
    "client_channels":1,
    "async_client_threads":1,
    "rpc_type":"UNARY",
    "load_params":{
      "closed_loop":{}}}
}
```



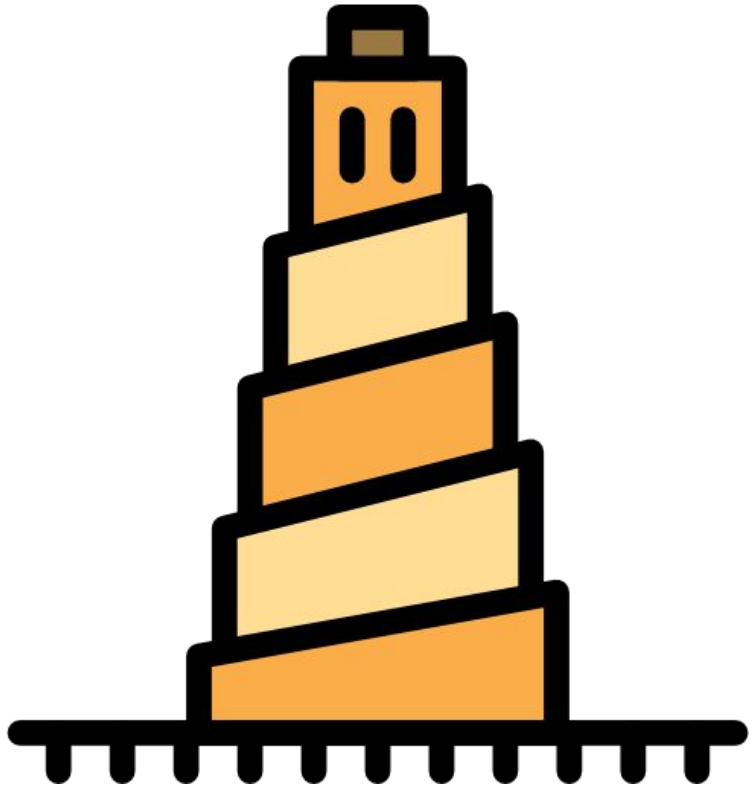
Benchmarks - Application

- Written by other teams.
- Exercises the stack in new ways.
- Only applies to libraries.

- From TensorFlow: rpcbench_test.cc

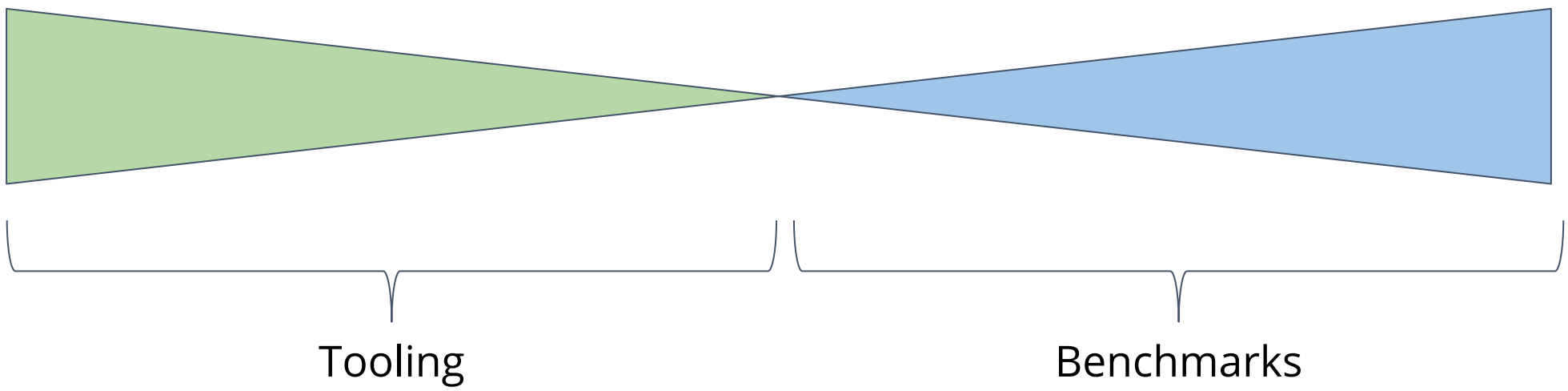
Data

- Team needs a *lingua franca*
- Optimizations come with:
  - data from tooling
  - data from benchmarks



Narrowing and Widening

- Tooling narrows scope
- Benchmarks widen scope
- End result is **data**



Tuning the gRPC Library

Undoing Death by 1000 Paper Cuts

What to do once the "low hanging fruit" has been taken?

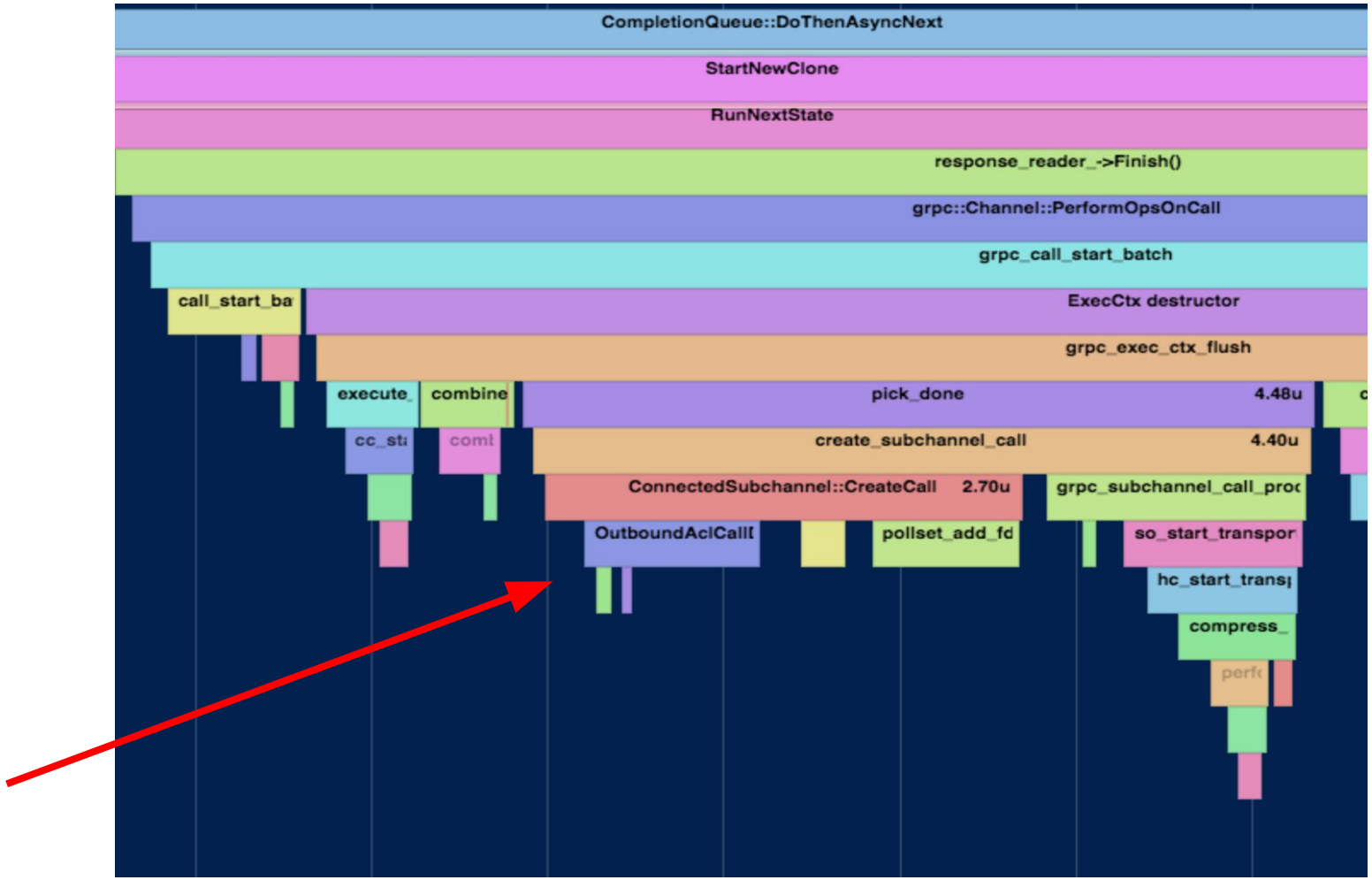
- Features can cause small regressions.
- Sometimes, *below margin of detection*.
- Consistent, slow, degradation of performance.

How do we reverse this process?

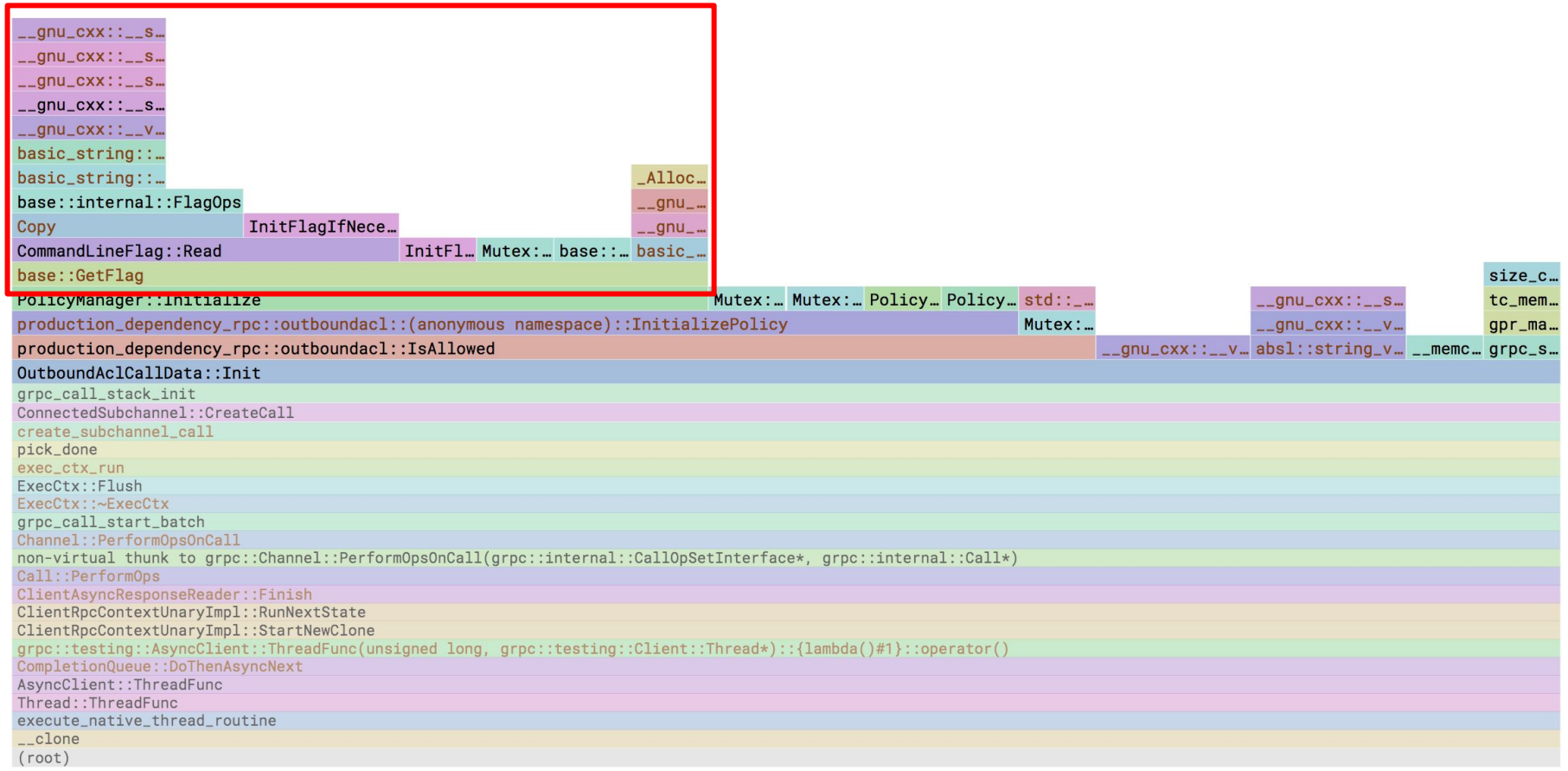
Undoing Death by 1000 Paper Cuts

- New benchmark: **Minimal RPC**
  - Ping pong of 1 byte payloads
  - No security
  - No census
  - Focused on median latency
- New tooling to use
- Noise reduction

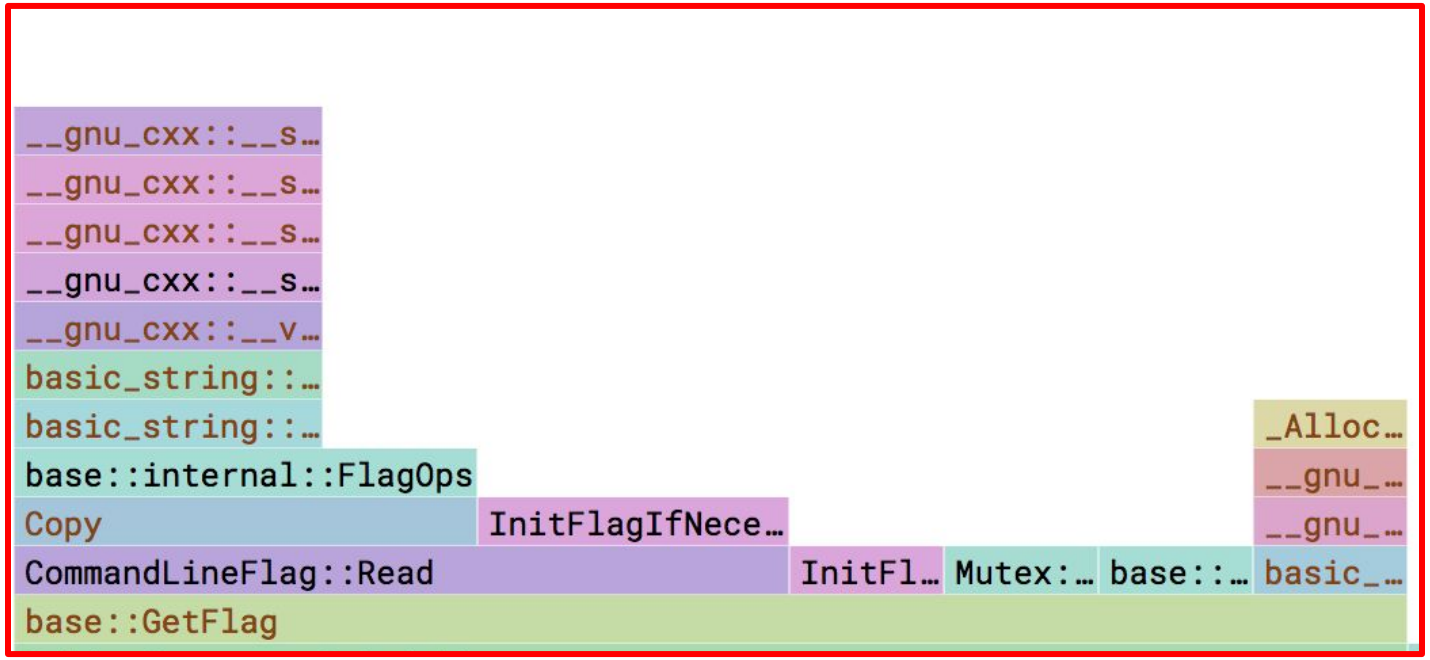
Case Study


CompletionQueue::DoThenAsyncNext
StartNewClone
RunNextState
response_reader_->Finish()
grpc::Channel::PerformOpsOnCall
grpc_call_start_batch
call_start_ba
ExecCtx destructor
grpc_exec_ctx_flush
execute_
combine
pick_done
4.48u
cc_sta
comt
create_subchannel_call
4.40u
ConnectedSubchannel::CreateCall
2.70u
grpc_subchannel_call_proc
OutboundAclCall
pollset_add_fd
so_start_transpor
hc_start_transp
compress_
perfo

Case Study


__gnu_cxx::__s…
__gnu_cxx::__s…
__gnu_cxx::__s…
__gnu_cxx::__s…
__gnu_cxx::__v…
basic_string::…
basic_string::…
base::internal::FlagOps
Copy
InitFlagIfNece…
CommandLineFlag::Read
InitFl…
Mutex:…
base::…
basic_…
_Alloc…
__gnu_…
__gnu_…
base::GetFlag
PolicyManager::Initialize
Mutex:…
Mutex:…
Policy…
Policy…
std::_…
production_dependency_rpc::outboundacl::(anonymous namespace)::InitializePolicy
Mutex:…
production_dependency_rpc::outboundacl::IsAllowed
__gnu_cxx::__v…
__gnu_cxx::__s…
__gnu_cxx::__v…
absl::string_v…
__memc…
size_c…
tc_mem…
gpr_ma…
grpc_s…
OutboundAclCallData::Init
grpc_call_stack_init
ConnectedSubchannel::CreateCall
create_subchannel_call
pick_done
exec_ctx_run
ExecCtx::Flush
ExecCtx::~ExecCtx
grpc_call_start_batch
Channel::PerformOpsOnCall
non-virtual thunk to grpc::Channel::PerformOpsOnCall(grpc::internal::CallOpSetInterface*, grpc::internal::Call*)
Call::PerformOps
ClientAsyncResponseReader::Finish
ClientRpcContextUnaryImpl::RunNextState
ClientRpcContextUnaryImpl::StartNewClone
grpc::testing::AsyncClient::ThreadFunc(unsigned long, grpc::testing::Client::Thread*)::{lambda()#1}::operator()
CompletionQueue::DoThenAsyncNext
AsyncClient::ThreadFunc
Thread::ThreadFunc
execute_native_thread_routine
__clone
(root)

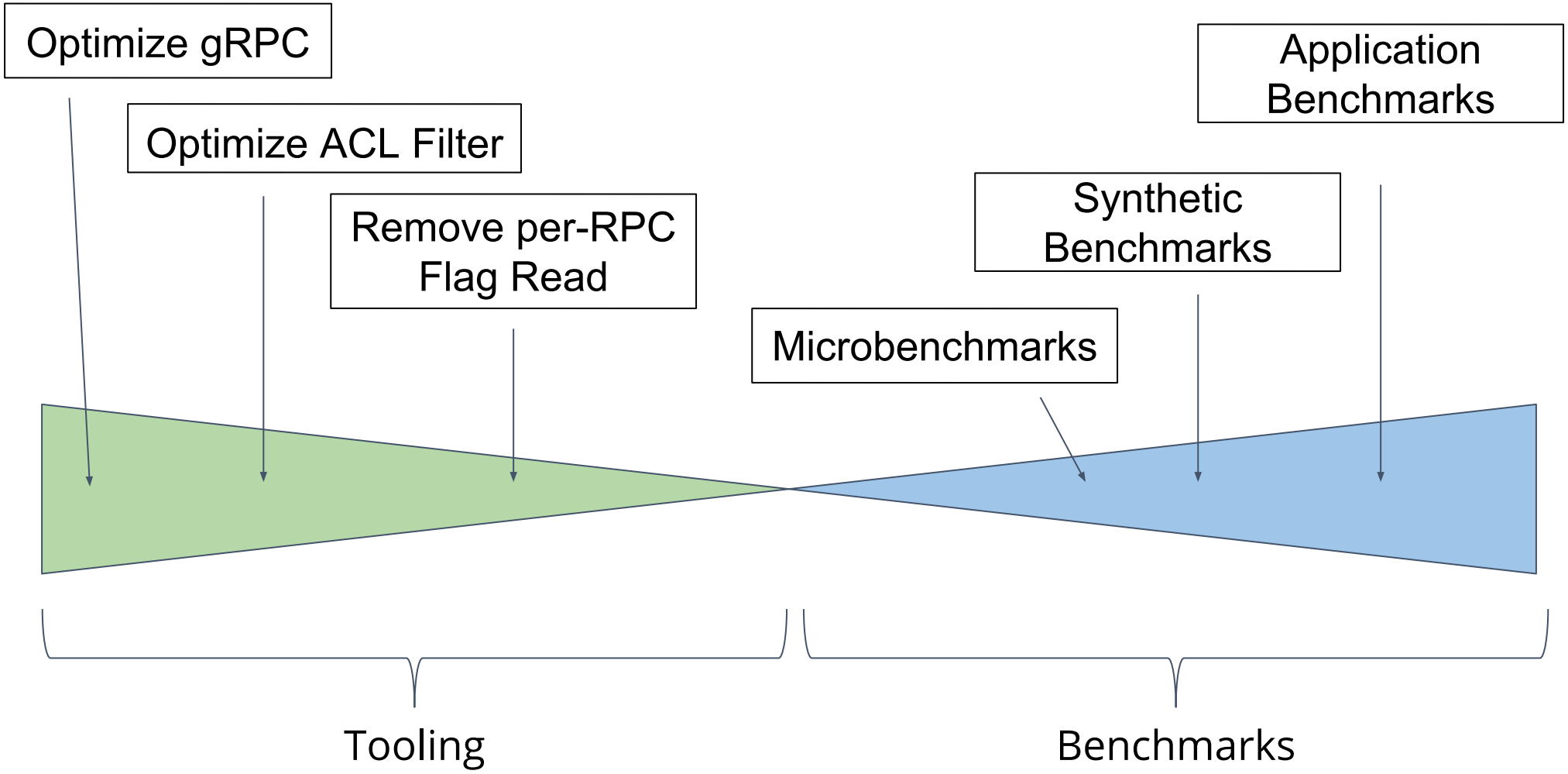
Case Study


__gnu_cxx::__s…
__gnu_cxx::__s…
__gnu_cxx::__s…
__gnu_cxx::__s…
__gnu_cxx::__v…
basic_string::…
basic_string::…
_Alloc…
base::internal::FlagOps
__gnu_…
Copy
InitFlagIfNece…
__gnu_…
CommandLineFlag::Read
InitFl…
Mutex:…
base::…
basic_…
base::GetFlag

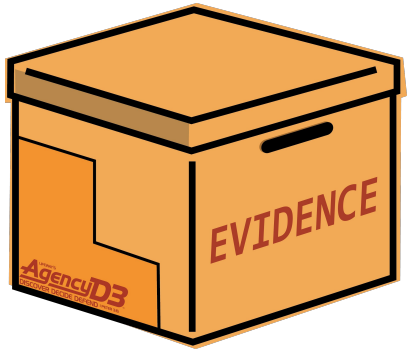
Case Study

- Things we said:
  - "ACL filter causes **~2us** per minimal RPC"
  - "Run this synthetic scenario to reproduce"

Case Study

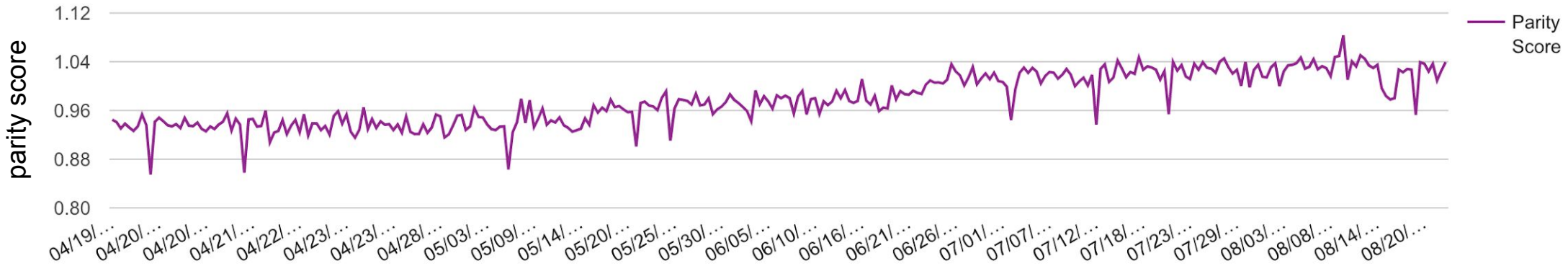


Data



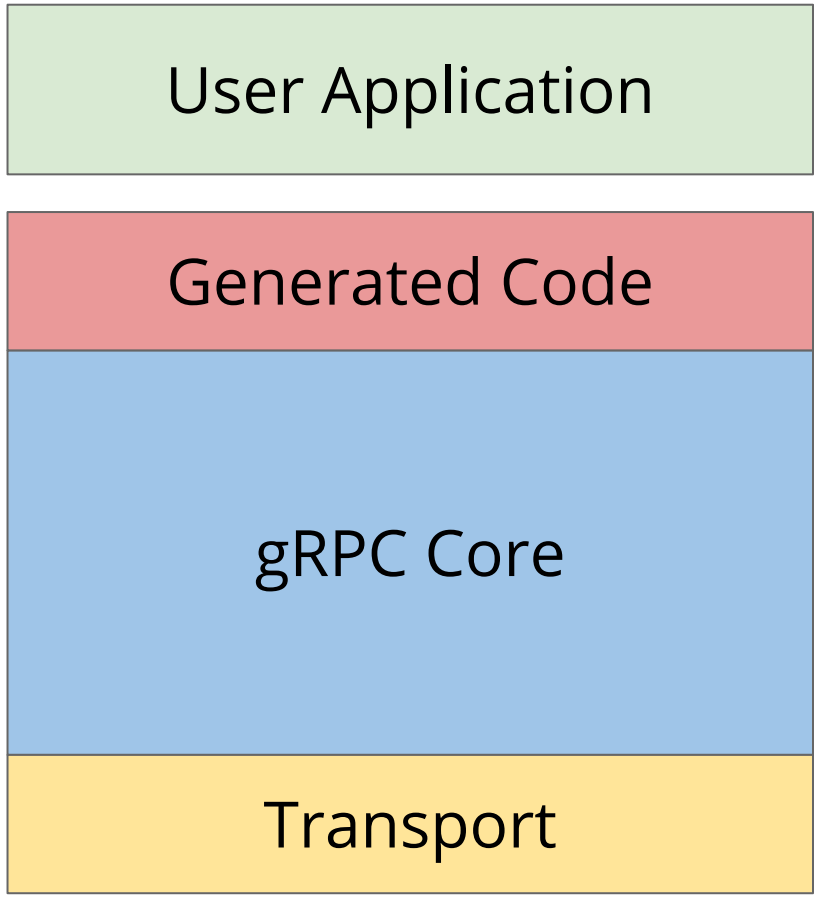
Results

- Minimal RPC Latency: **67us —> 55us**
- Comparison score with Stubby:



Breaking Down the Layers

- Tuning below:
  - Contributions to Netty, OkHttp.
  - Contributions to x/net/http.
  - Tuning work with TCP team.
- Tuning above:
  - Next part of this talk.



Tuning gRPC Applications

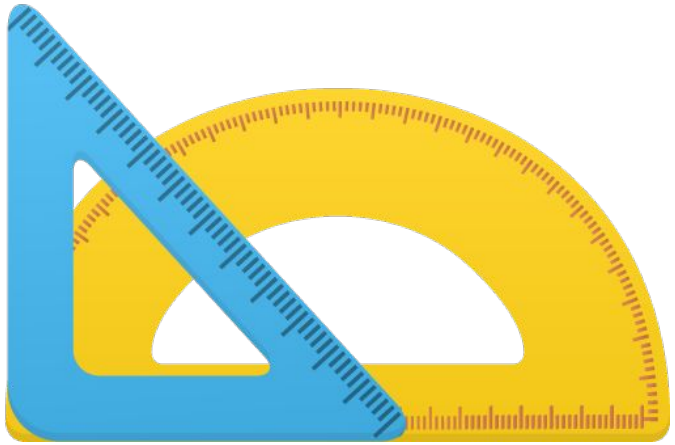
Low Hanging Fruit

- All Language Stacks
  - Reduce allocations
  - Reduce copies
  - Reduce syscalls
  - Reduce contention
- Java Stack
  - Use async API
  - Tune threading model
  - Tune gRPC memory pool
- C++ Stack:
  - Use async API
  - Tune threading model
  - # of completion queues
  - # of outstanding RPCs
- Go Stack
  - Reduce allocations!!

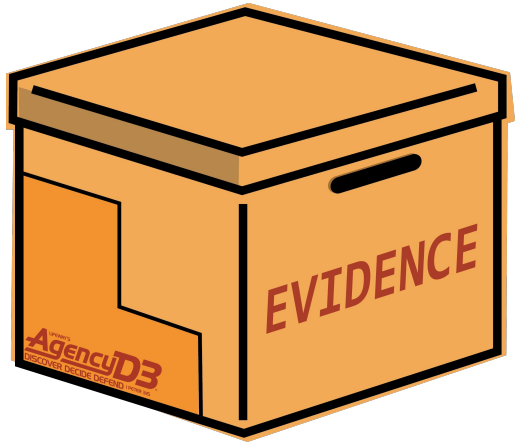
And of Course

Tooling

Benchmarks



Data



Case Study

- Distributed TensorFlow sits atop RPC layer
- **Goal**: move internal TensorFlow to gRPC
- gRPC's performance lagged slightly

Case Study

- Learned TensorFlow tracing system:

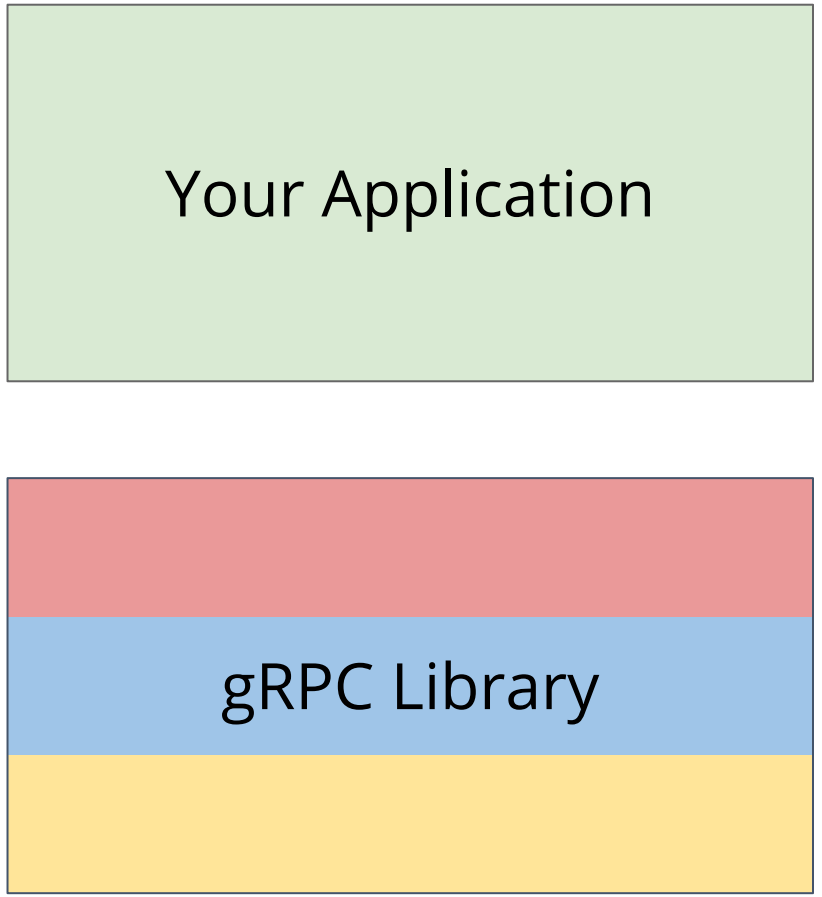
Case Study

- New benchmarks:
  - Rpcbench
  - Real TensorFlow training tasks

- Impactful changes:
  - better threading
  - serialization threadpool

Breaking Down the Layers (again)

- Tuning below:
  - Contribute to gRPC!



Thank You!

- gRPC Resources:
  - http://grpc.io
  - http://grpc.io/contribute
  - https://github.com/grpc
  - https://github.com/grpc-ecosystem

- Personal Contact:
  - Email: ncteisen@google.com
  - GitHub: https://github.com/ncteisen
  - Website: http://noaheisen.com


KubeCon
CloudNativeCon
North America 2018

Appendix

Optimization: DoThenAsyncNext

- **Author:** kpayson64@
- **Change:** #13084
- **Location:** gRPC Core.
- **Context:** gRPC has a asynchronous completion queue API. Work is placed on the queue by the application, driven by calls to AsyncNext, and then completion events are returned to application.
- **Optimization:** New API in which application can pass a lambda to be executed before AsyncNext. If this lambda triggers a completion event, it is returned by the call to AsyncNext.
- **TL;DR:** Reduced thread hops in a common case.

Optimization: TF Threading

- **Author:** ncteisen@
- **Change:** 0d5fb10
- **Location:** TensorFlow application layer.
- **Context:** TensorFlow has GrpcWorker class, which is responsible for encapsulating the gRPC network layer from the TensorFlow application.
- **Optimization:** Allow multiple threads to service the GrpcWorker's completion queues.
- **TL;DR:** More parallelism.
-

Optimization: Epoll Exclusive

- **Author:** ctiller@
- **Change:** #12789
- **Location:** gRPC Core.
- **Context:** gRPC has an internal polling system to efficiently interact with network I/O. It has gone through several iterations and optimizations.
- **Optimization:** New polling system, epollex, that relies on the EPOLLEXCLUSIVE flag for epoll_ctl.
- **TL;DR:** Thread are woken up more efficiently.

Optimizations to the Minimal RPC

- hcaseyal@
  - #15839, #15879, #15883 (moving allocations to call arena)
- kpayson64@
  - #13947 (adds fd cache to avoid epoll_ctl)
- ncteisen@
  - #15578 (compile out spammy tracer)
- yashykt@
  - #15280 (compile out stats machinery in opt builds)
  - #15200 (adds new closure scheduling mechanism)
  - #15044 (adds compiler hints)