Zephyr® Project
Developer Summit

Zephyr® Project
Developer Summit

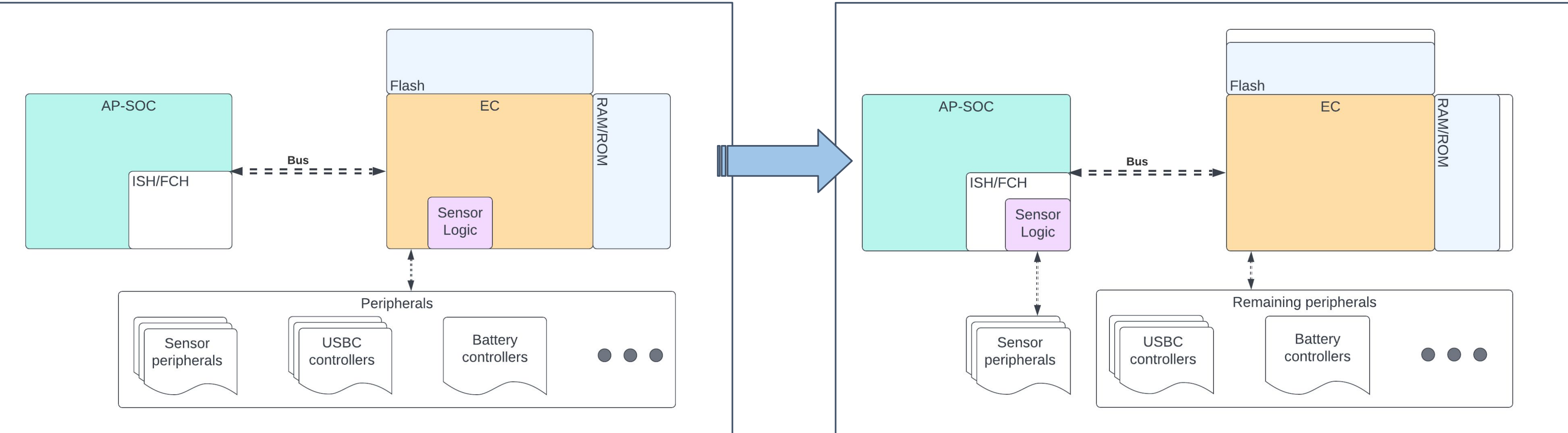
Flexible system design via RPC:
Embracing distributed computing in Zephyr

Yuval Peress, *Google*

Email: peress@google.com
Discord: yuval#5515
GitHub: yperess

#EMBEDDEDOSSUMMIT

- Chromebooks ship with an EC and a separate Application Processor (AP)
- Many APs (Intel and AMD) come with a dedicated sensor core
- We wanted to move the sensor logic to reduce the cost of the EC
- But how? Dependencies, tests, and prior designs were broken



Agenda

- Portable design
- Pigweed RPCs and protobuffers
- Transitioning from headers to services
- An example

Portable design

hint: they're microservices

Chromium’s EC has many tasks

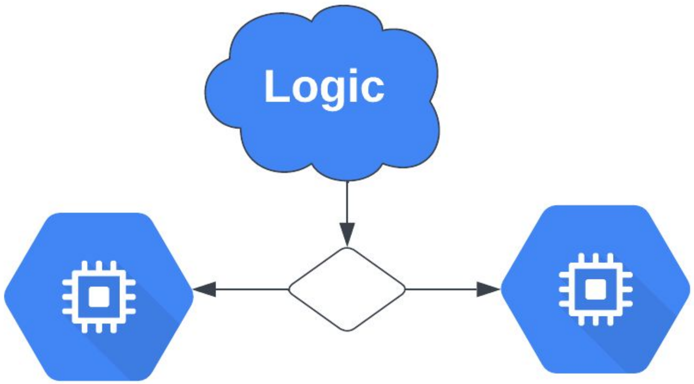
SYSWORKQ
SHELL
TASK_TOUCHPAD
TASK_CHG_RAMP
TASK_USB_CHG
TASK_DPS
TASK_CHARGER
TASK_CHIPSET

TASK_MOTIONSENSE
TASK_USB_MUX
TASK_HOSTCMD
TASK_KEYPROTO
TASK_POWERBTN
TASK_KEYSCAN
TASK_PD_C<port_num>
TASK_PD_INT_C<portn_num>

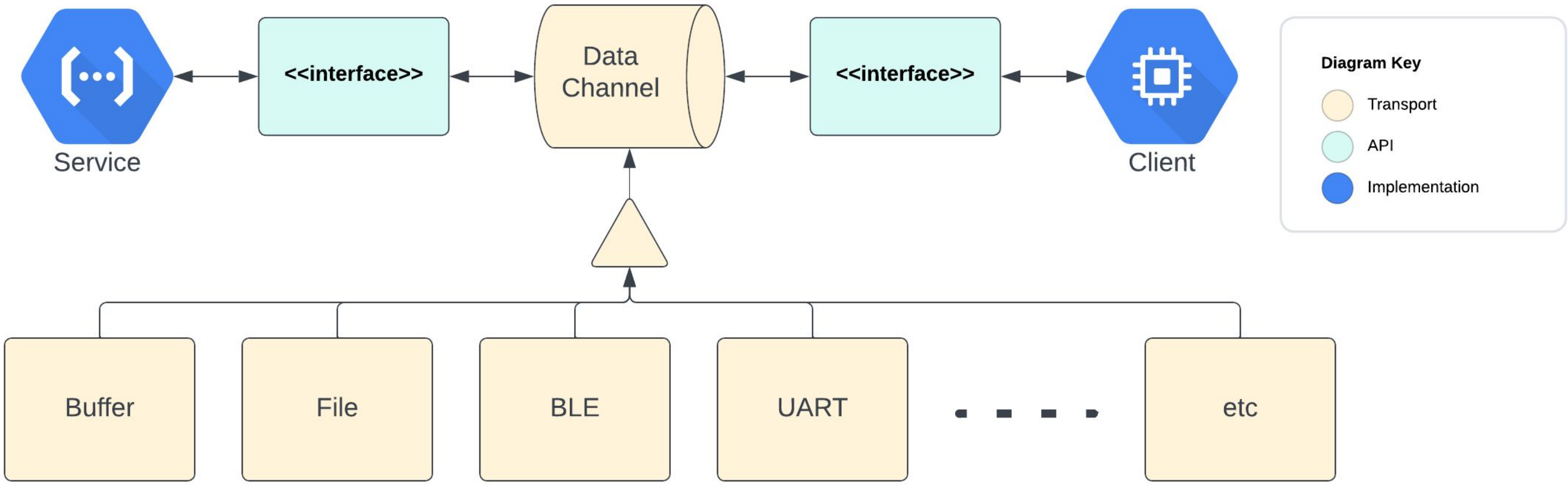
- Sensors
- Power Delivery

Does it matter where the task lives?

- Sensor logic can be on a dedicate core such as Intel's ISH or AMD's SFH.
- Power delivery (PD) logic can be on PD chip



With the right design, the service and clients never need to be rewritten



Zephyr® Project
Developer Summit

Pigweed RPCs and protobufs

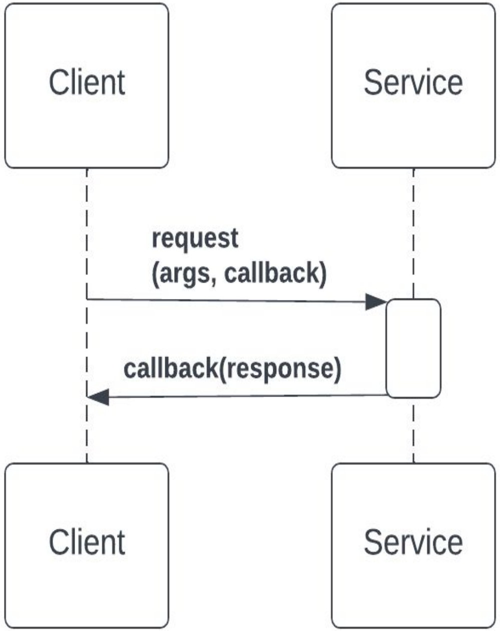
#EMBEDDEDOSSUMMIT

What is Pigweed?

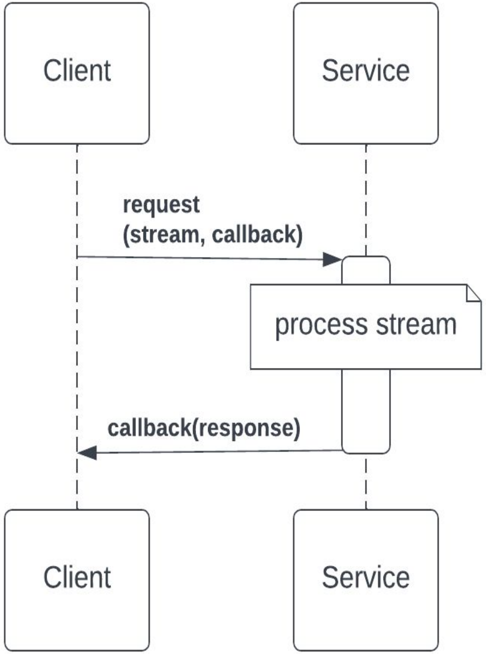
- Pigweed is a collection of tools/modules
- Highly tuned for embedded applications
- Modules in this talk:
  - pw_rpc
  - pw_hdlc

RPC concepts

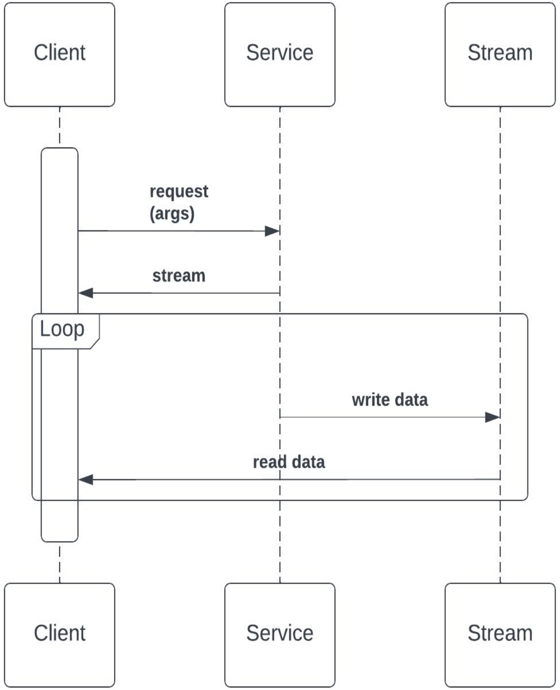
Unary RPC



Client Streaming

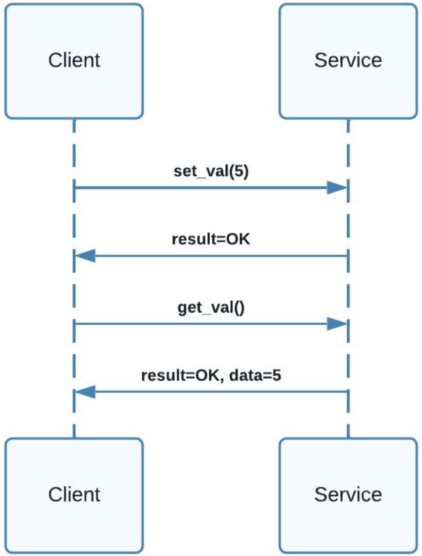


Server Streaming



A simple service

- Asynchronous
- set_val
  - passes an int and saves it on the service
  - returns a status when finished
- get_val
  - passes no args
  - returns a status and the current value



When to use streams?

- Pigweed uses server streams for logging (pw_log_rpc)
  - Client makes a request to the service (EC) and gets back a stream
  - Each message on the stream is a log message
- When data is generated with some latency

Zephyr® Project Developer Summit

How to set it up (west.yml)?

```yaml
# Add the remote
remotes:
  - name: pigweed
    url-base: https://pigweed.googlesource.com/pigweed

# Add pigweed to the project
projects:
  - name: pigweed
    remote: pigweed
    revision: main
```

How to set it up (kConfig)?

```
(Top) → Zephyr → C++ Language Support

[*] C++ support for the application
        C++ Standard (C++ 17)  --->
```

```
(Top) → Zephyr → Modules

    *** Available modules. ***
    pigweed (/home/peress/workspace/zds2023/pigweed)  --->
```

```
(Top) → Zephyr → Modules → pigweed (/home/peress/workspace/zds2023/pigweed)

    pw_assert  --->
    pw_base64  --->
    pw_bytes  --->
    pw_checksum  --->
    pw_chrono  --->
    pw_containers  --->
```

Enable C++

How to set it up (kConfig)?

```
(Top) → Zephyr → C++ Language Support

[*] C++ support for the application
      C++ Standard (C++ 17)  --->
```

```
(Top) → Zephyr → Modules

    *** Available modules. ***
    pigweed (/home/peress/workspace/zds2023/pigweed)  --->
```

```
(Top) → Zephyr → Modules → pigweed (/home/peress/workspace/zds2023/pigweed)

    pw_assert  --->
    pw_base64  --->
    pw_bytes  --->
    pw_checksum  --->
    pw_chrono  --->
    pw_containers  --->
```

Find the Pigweed module

Zephyr® Project
Developer Summit

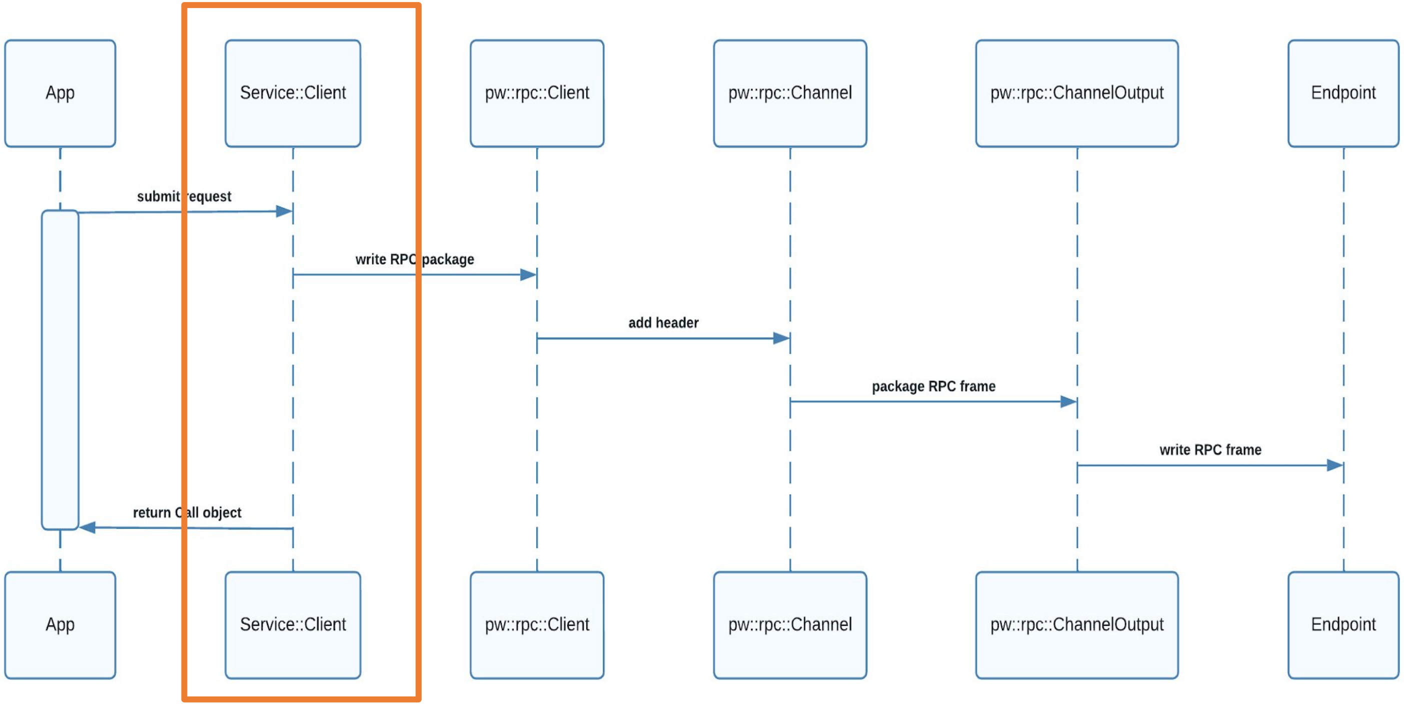
How to set it up (kConfig)?

```
(Top) → Zephyr → C++ Language Support

[*] C++ support for the application
        C++ Standard (C++ 17)  --->

(Top) → Zephyr → Modules

    *** Available modules. ***
    pigweed (/home/peress/workspace/zds2023/pigweed)  --->

(Top) → Zephyr → Modules → pigweed (/home/peress/workspace/zds2023/pigweed)

    pw_assert  --->
    pw_base64  --->
    pw_bytes  --->
    pw_checksum  --->
    pw_chrono  --->
    pw_containers  --->
```

Enable the pw_* libraries you need

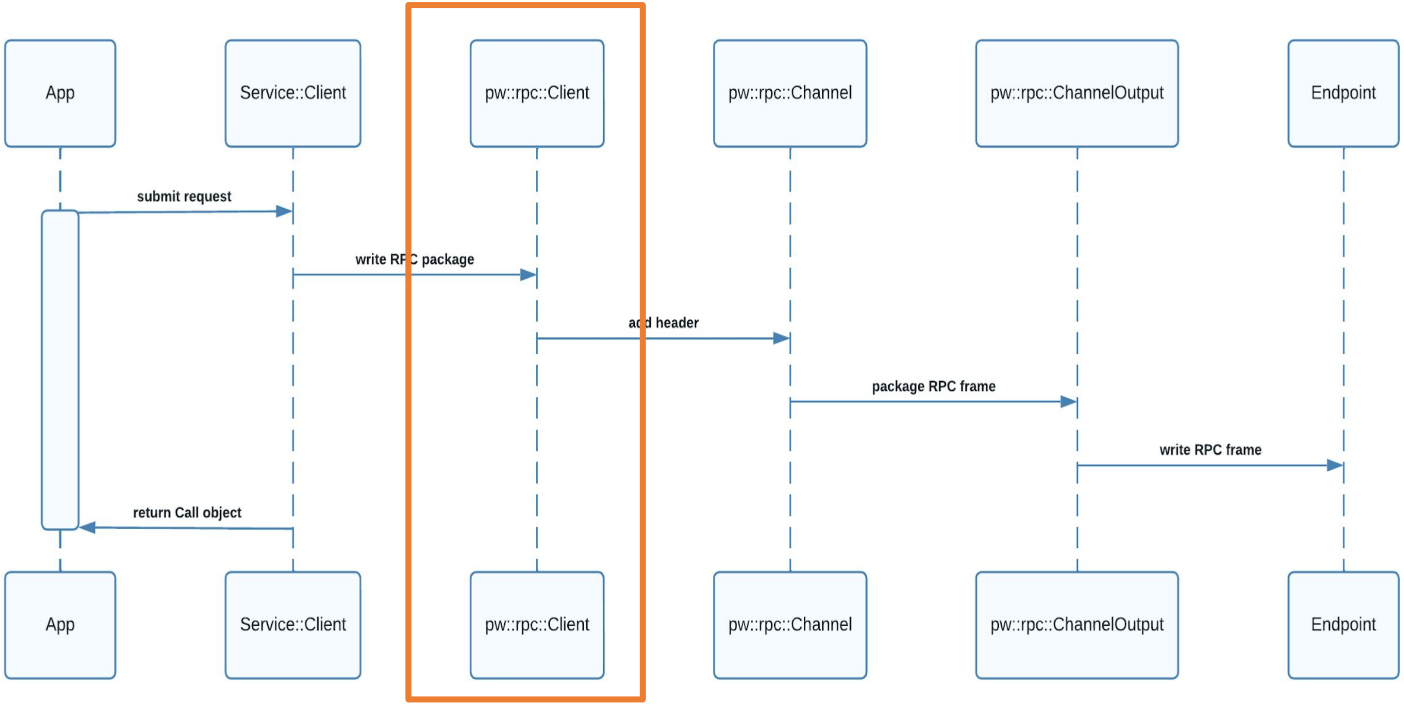
Data visualization of pw_rpc

Custom client that injects the service specific information automatically, like service and method IDs.



Data visualization of pw_rpc

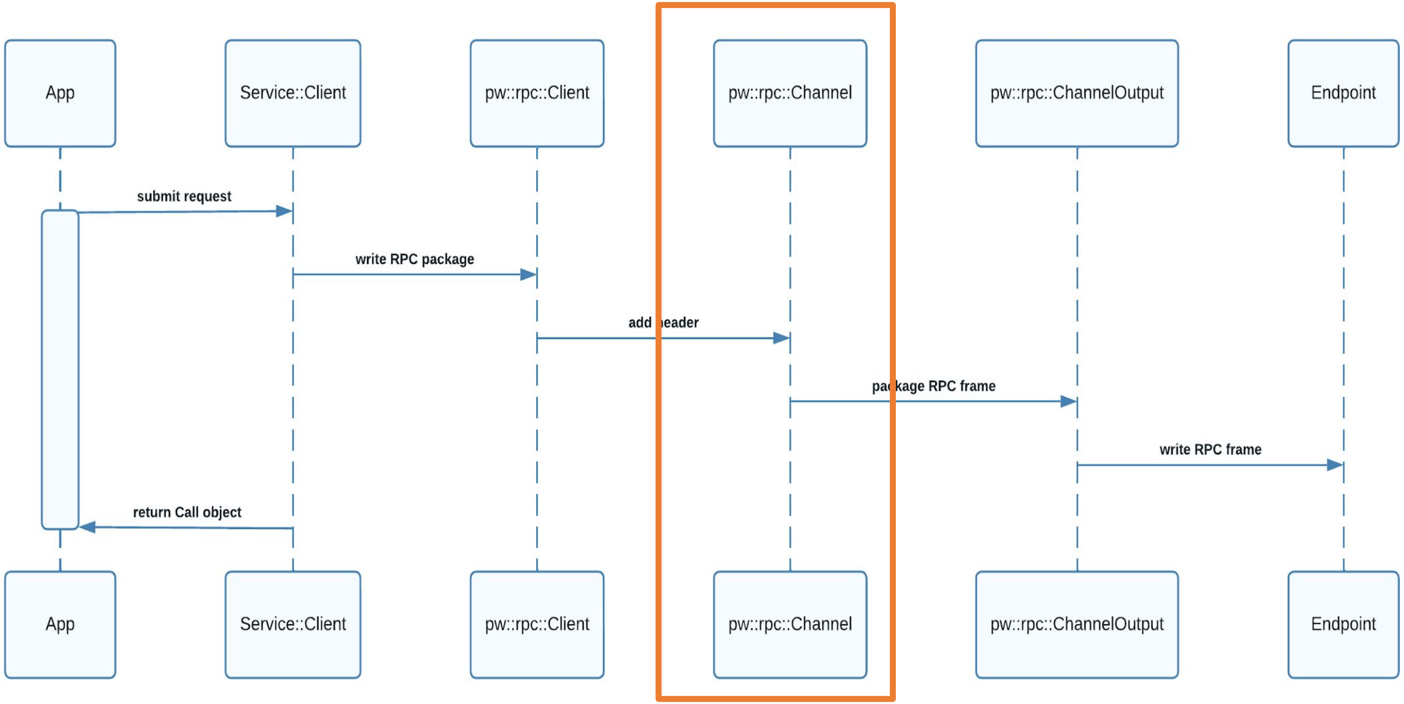
General purpose client which routes the package to the correct channel



Zephyr® Project
Developer Summit

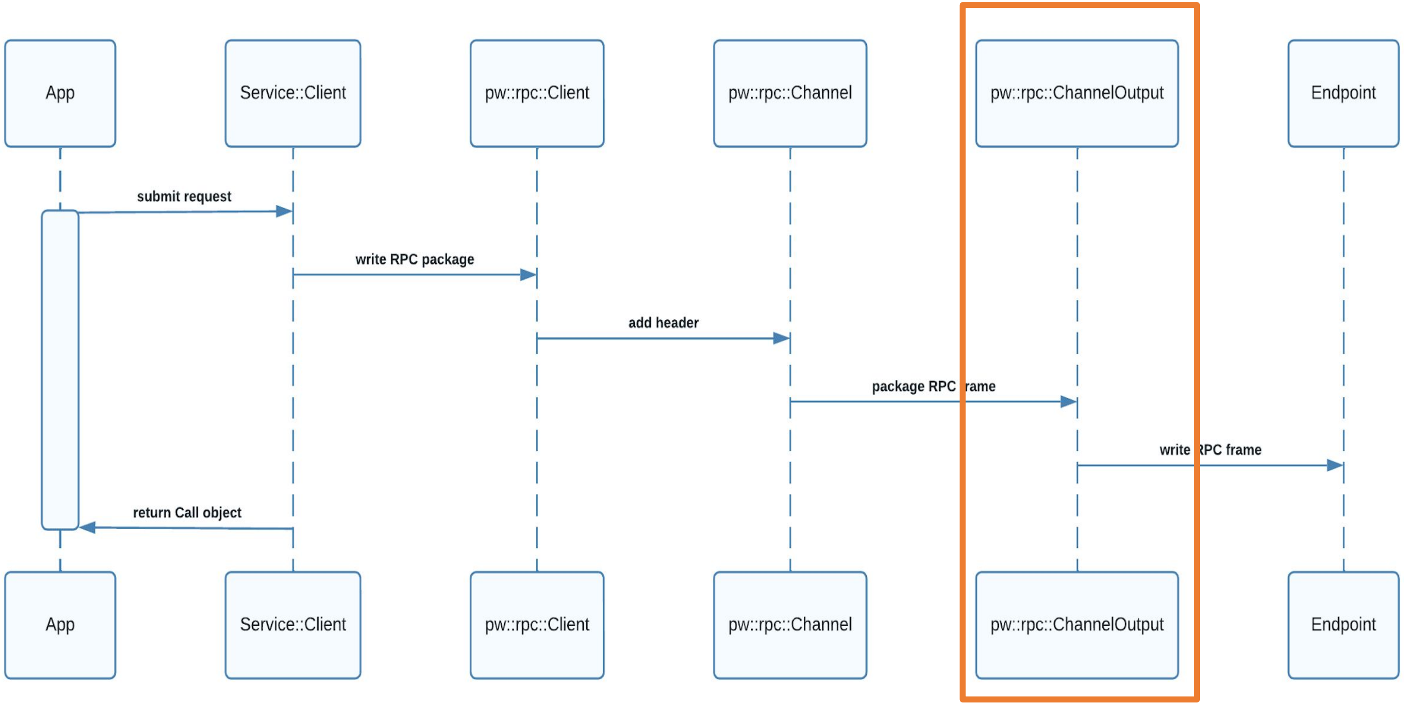
Data visualization of pw_rpc

Custom channel implementation controlling the RPC protobuf wire format



Data visualization of pw_rpc

Writes the RPC frame to the destination and potentially add another level of encoding



Transitioning from headers to services

Can’t I just use a .h file? Yes, but…

- Protobuffers are more flexible (client and service can be in different languages)
- Protobuffers make it easy to test by creating a mock client/service
- Protobuffers extend easier than structs (support multiple versions)

Proto files force us to think about API boundaries

- APIs + documentation provide a contract
- Keep our code interaction confined so refactors are easier to manage
- Boundaries make writing tests easier and faster
- Protos are designed to be extensible

What would the header look like?

```
/* The data structures used for the API */
struct SetValueRequest {
  int32_t value;
};
struct SetValueResponse {};
struct GetValueRequest {};
struct GetValueResponse {
  int32_t value;
};

/* APIs for setting and getting the value */
int client_set_value(
    const struct SetValueRequest *request,
    void(*callback)(int status, const struct SetValueResult *result)
);
int client_get_value(
    const struct GetValueRequest *request,
    void(*callback)(int status, const struct GetValueResult *result)
);
```

So what's the problem?

1. Data structs are hard to maintain as new arguments are added
2. The API is lacking a lot of features still
    a. How is the data is sent?
    b. What is the wire format?
    c. What thread is the callback called on?
    d. Can we cancel a request?
3. How do we test this service?

pw_rpc solves these issues

Zephyr® Project
Developer Summit

Example

#EMBEDDEDOSSUMMIT

Let’s build a simple service

- SetValue needs:
  - Passing the value as an argument

```
message SetValueRequest {
  int32 value = 1;
}
```

Let's build a simple service

- SetValue needs:
  - Passing the value as an argument
  - No return value

```
message SetValueRequest {
  int32 value = 1;
}
message SetValueResponse {}
```

Zephyr® Project
Developer Summit

Let's build a simple service

- SetValue need:
    - Passing the value as an argument
    - No return value
- GetValue needs:
    - Passing nothing

```
message SetValueRequest {
  int32 value = 1;
}
message SetValueResponse {}
message GetValueRequest {}
```

Let's build a simple service

- SetValue need:
  - Passing the value as an argument
  - No return value
- GetValue needs:
  - Passing nothing
  - Return the value

```
message SetValueRequest {
  int32 value = 1;
}
message SetValueResponse {}
message GetValueRequest {}
message GetValueResponse {
  int32 value = 1;
}
```

Let's build a simple service

- SetValue need:
  - Passing the value as an argument
  - No return value
- GetValue needs:
  - Passing nothing
  - Return the value
- Add the service

```
message SetValueRequest {
  int32 value = 1;
}
message SetValueResponse {}
message GetValueRequest {}
message GetValueResponse {
  int32 value = 1;
}
```

```
service Cache {
  rpc SetValue(SetValueRequest) returns (SetValueResponse) {}
  rpc GetValue(GetValueRequest) returns (GetValueResponse) {}
}
```

Service implementation header

```
class Cache : public pw_rpc::nanopb::Cache::Service<Cache> {
 public:
  Cache() : value_(0) {}
  ::pw::Status SetValue(const ::SetValueRequest& request, ::SetValueResponse& response);
  ::pw::Status GetValue(const ::GetValueRequest& request, ::GetValueResponse& response);
 private:
  int32_t value_;
};
```

Service implementation

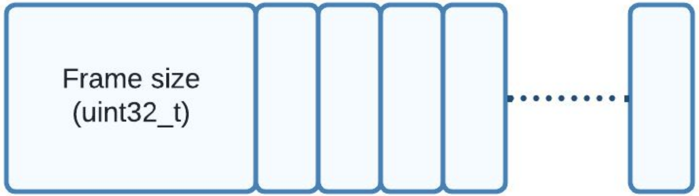
```
::pw::Status Cache::SetValue(const ::SetValueRequest& request, ::SetValueResponse& response) {
  value_ = request.value;
  return ::pw::OkStatus();
}

::pw::Status DemoService::GetValue(const ::GetValueRequest& request, ::GetValueResponse& response) {
  response.value = value_;
  return ::pw::OkStatus();
}
```

Abstracting away the ChannelOutput

- The ChannelOutput controls the wire format
- Generally, uses a pw::stream::Writer to write the final bytes
- Can enable us to efficiently switch how the service communicates with the client
- Example ChannelOutputs:
  - pw::hdlc::RpcChannelOutput
  - A custom SimpleChannelOutput used in this talk
- Both examples will use the same stream Writer to write to a ring buffer

ChannelOutput options

- pw_hdlc provides a ChannelOutput implementation which packs the data in an HDLC frame
- For local writes (between threads) I've implemented a simple ChannelOutput which simply writes frames as:



RingBufferReaderWriter

- Transactional
- Wraps a ring buffer
- Uses a mutex and condvar to control data availability

Performance?

- Creates 2 threads (client -> service & service -> client)
- On the main thread run 1,000 iterations of:
  - Call SetData, wait for response
  - Call GetData, wait for response
- Comparison setups:
  - [control] Read/write a plain serialized struct using SimpleChannelOutput to a plain service implementation
  - [experiment] Uses pw_rpc to write simple RPC frames using SimpleChannelOutput to RPC service implementation and server
- Things to consider:
  - The control is an oversimplification (no priority control, no call cancel, doesn’t account for extensibility)
  - Some code paths of pw_rpc were identified as bottlenecks are actively being optimized

Performance?

- [control] took 65,644,840 nanoseconds (~33 µs / call)
- [experiment] took 233,103,156 nanoseconds (~116 µs / call)

Overall slowdown 255%, but… with pw_rpc you get free upgrades :)

Future performance improvements

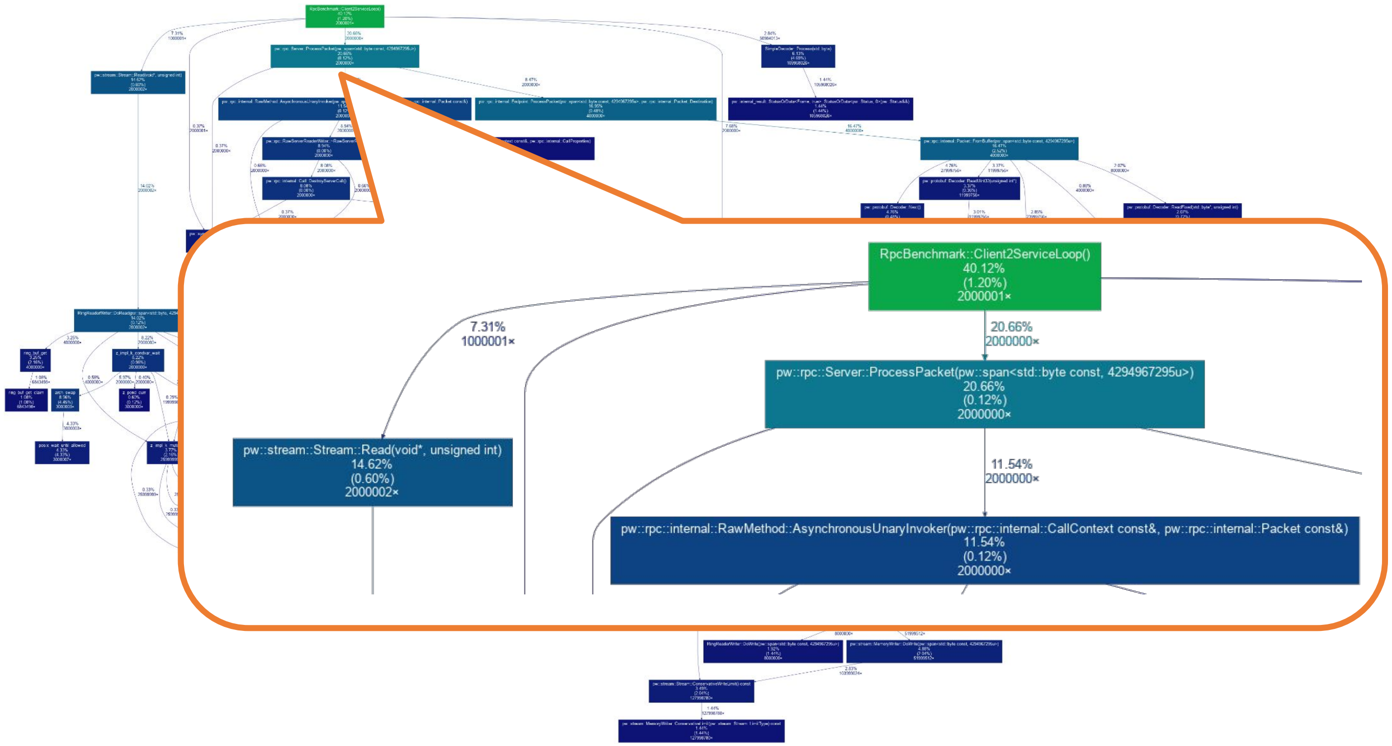
- Call graphs in the appendix show several points to improve on (when communicating between threads)
    a. Don’t use nanopb to serialize the RPC header (cost is 21%)
    b. Remove the need for a disconnect RPC packet on the Call destructor (38%)
- These would bring overall RPC cost closer to 45% (14 μs / call)

Zephyr® Project
Developer Summit

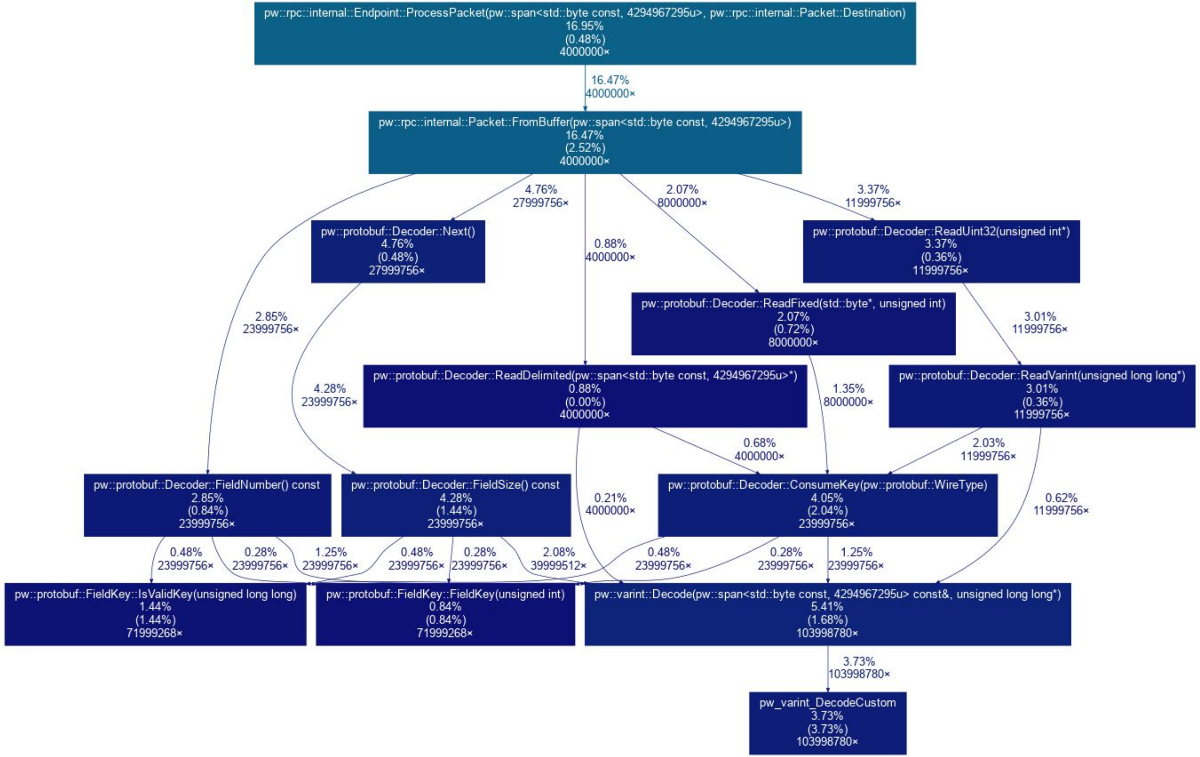
Questions?

Call graph of the client2service handler

Call graph of the client2service handler



Processing the header (21% of the cost)



Call destructor (38% of the cost)