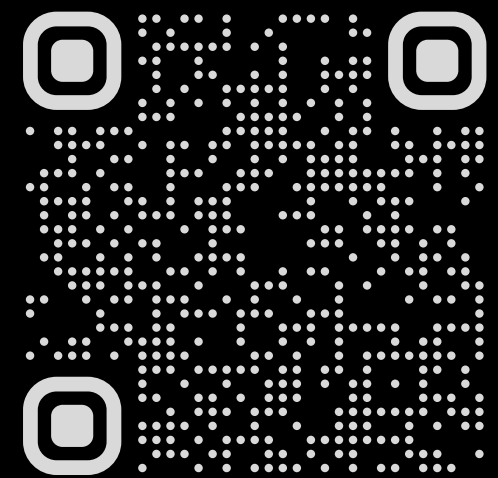




unicoap: A Unified and Modular CoAP Suite in RIOT

Carl Seifert
carl.seifert@tu-dresden.de
Dresden, September 3, 2025

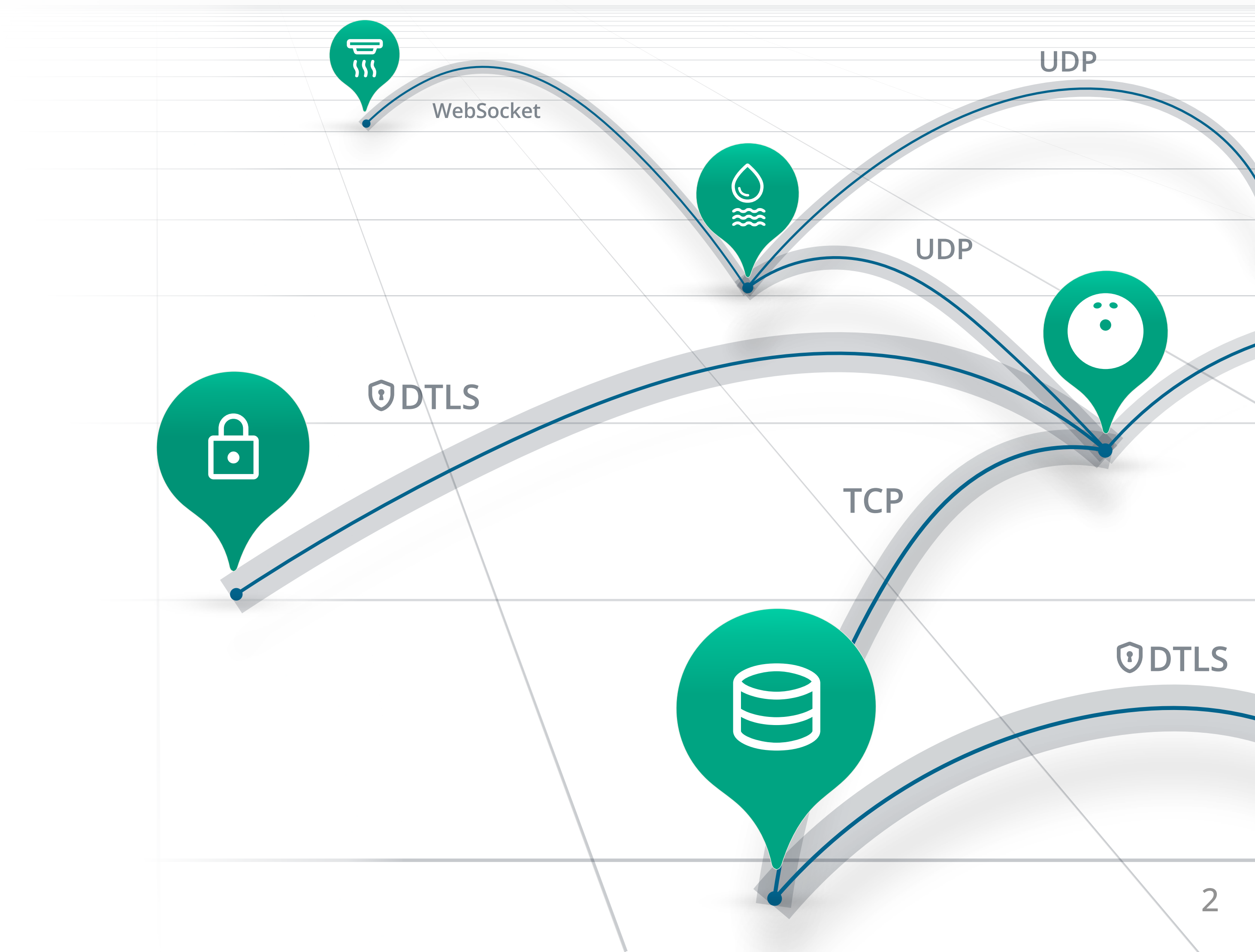


Recording

youtube.com/watch?v=HtzV0rZVJAI

unicoap

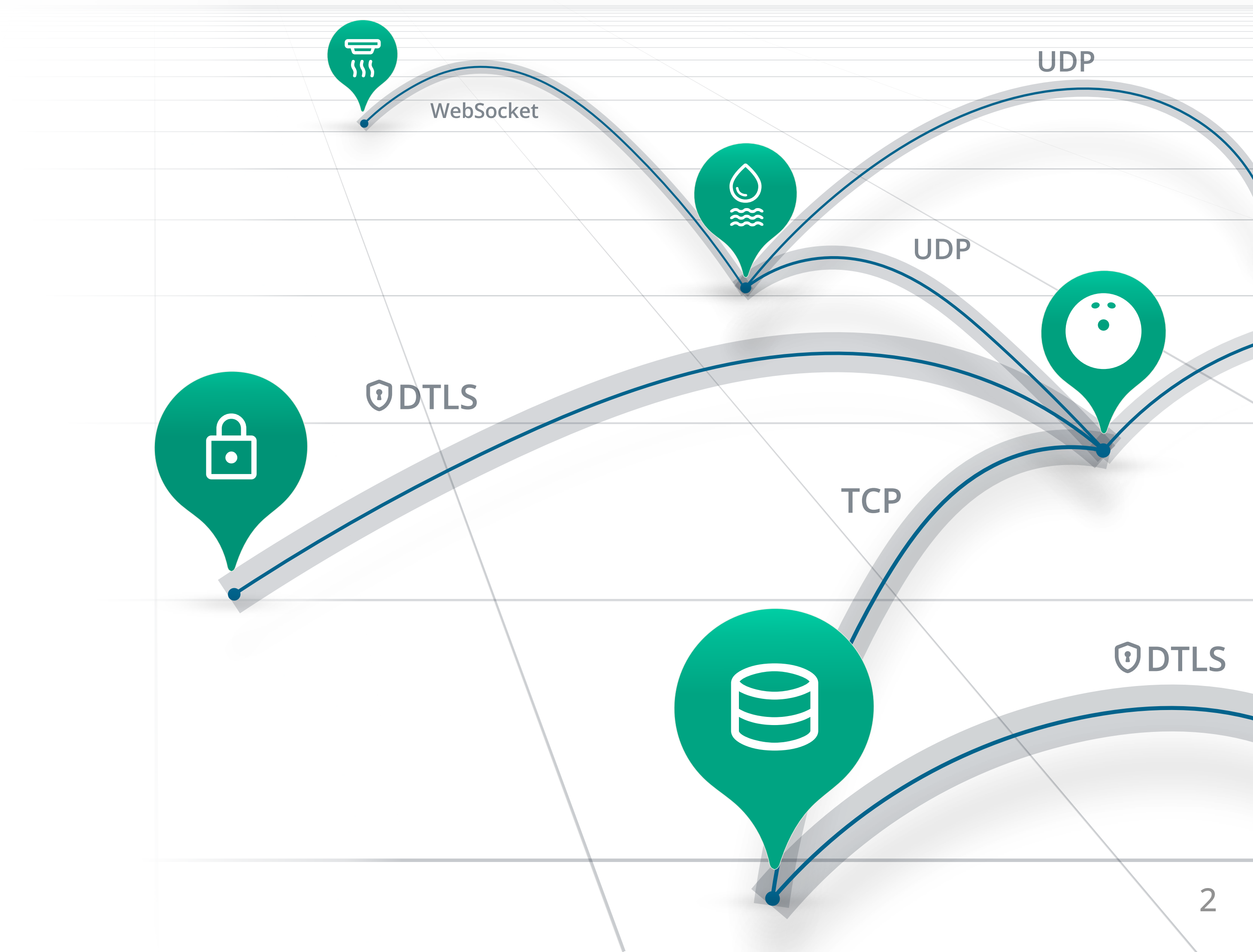
Motivation



unicoap

Motivation

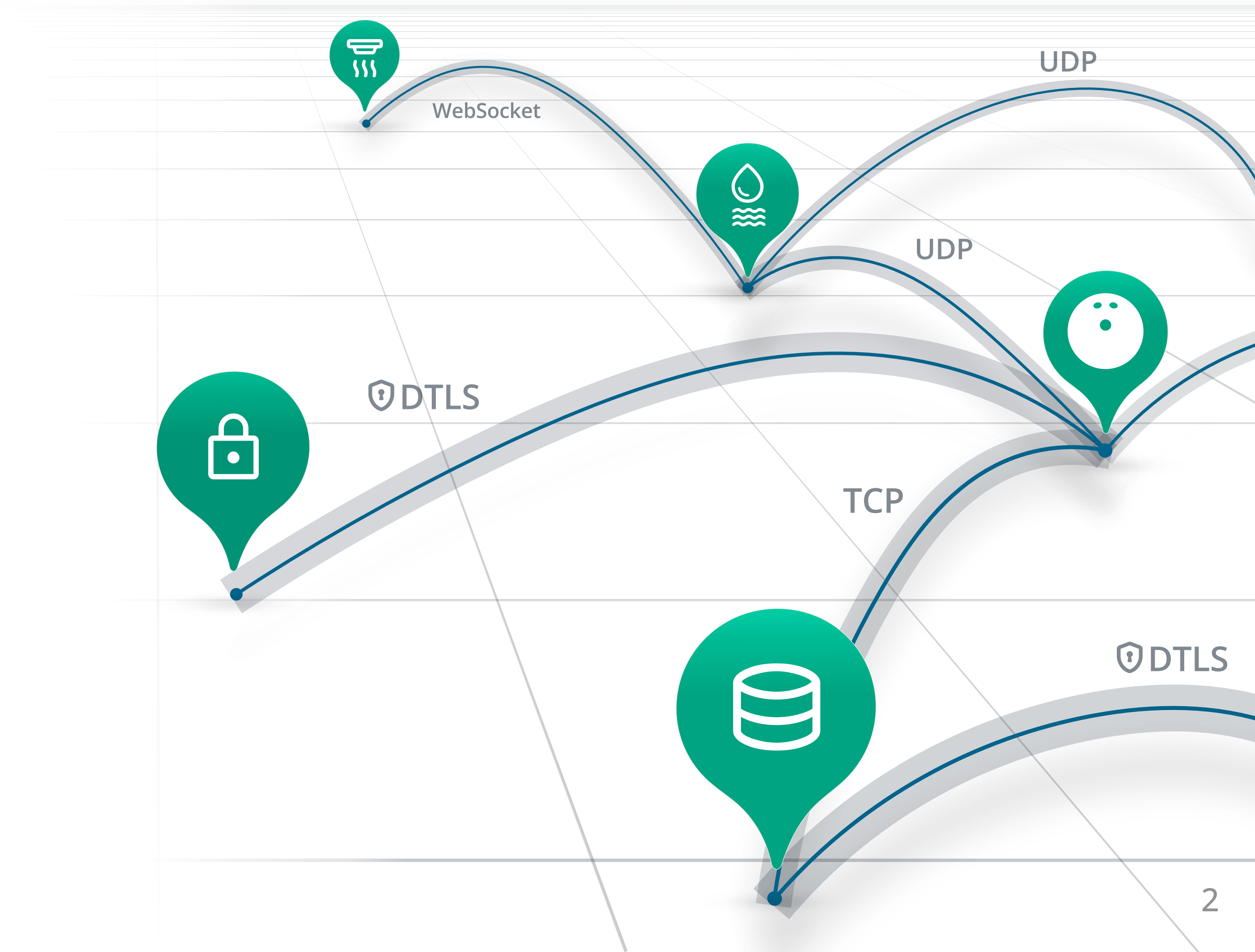
Messaging, PDU format varies



unicoap

Motivation

Messaging, PDU format varies
Interdependent features

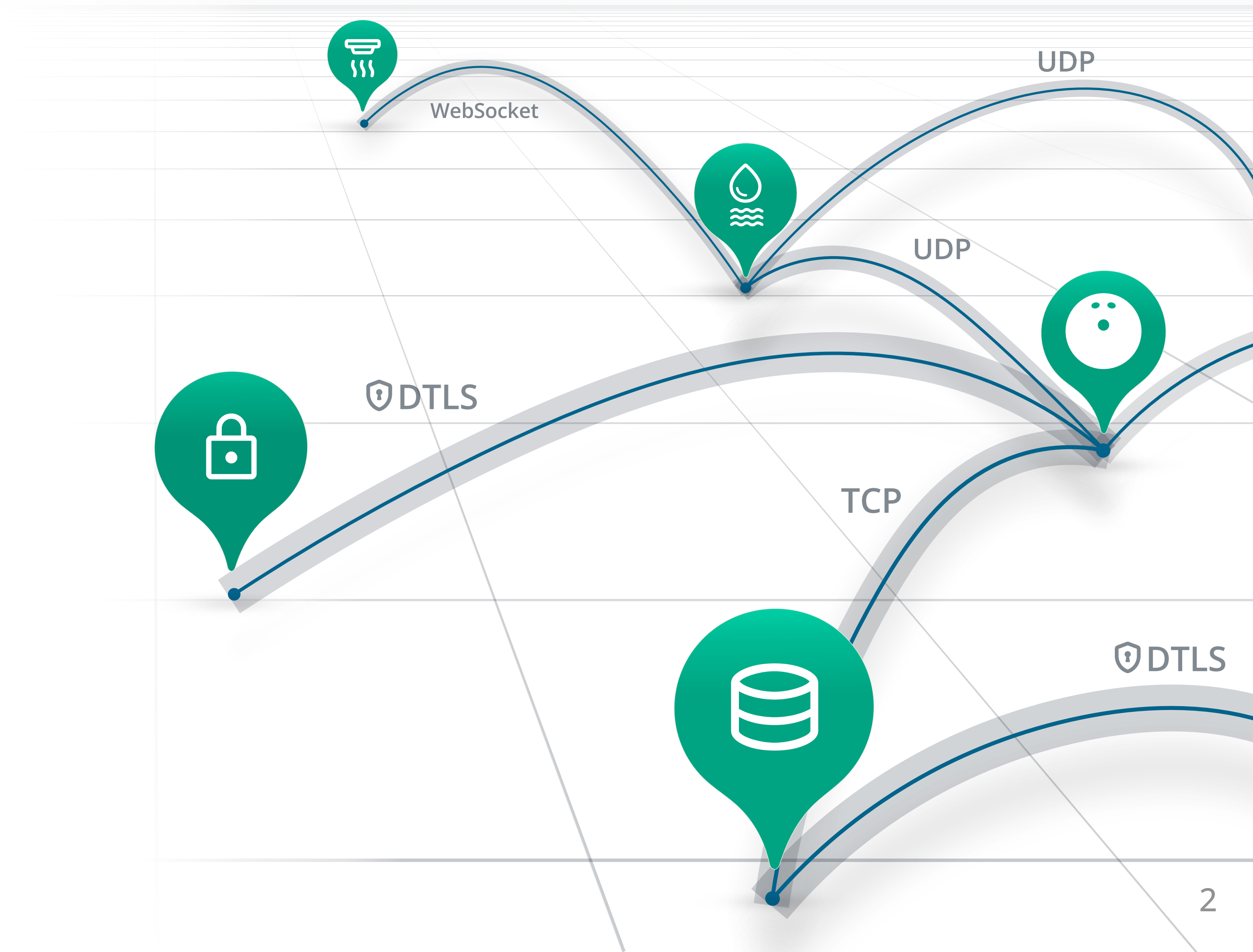


unicoap

Motivation

Messaging, PDU format varies
Interdependent features

- High-level, extensible API
- Transport abstraction



unicoap

Motivation

Messaging, PDU format varies

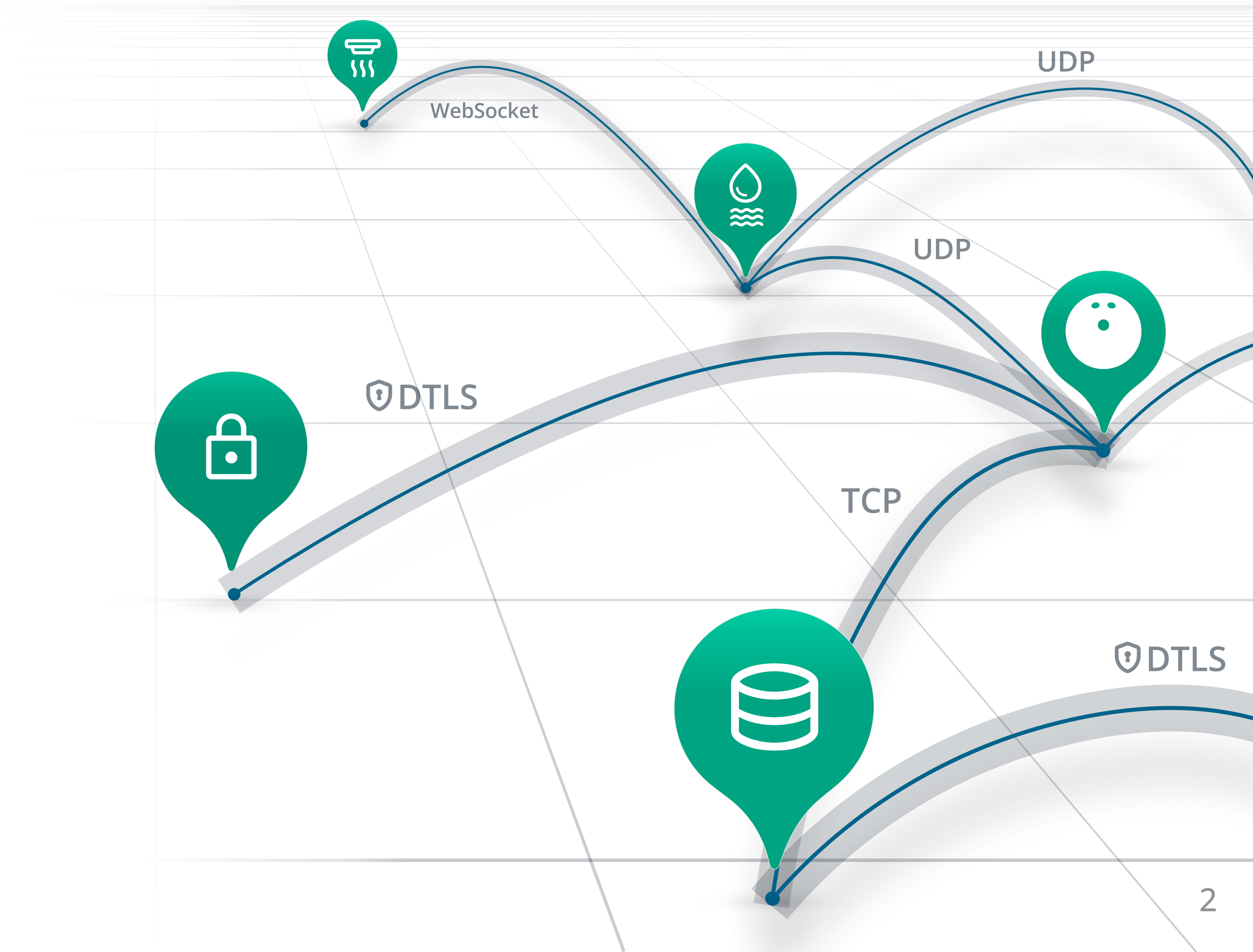
Interdependent features

→ High-level, extensible API

→ Transport abstraction

 Layering

 Modularization



unicoap

Motivation

Messaging, PDU format varies

Interdependent features

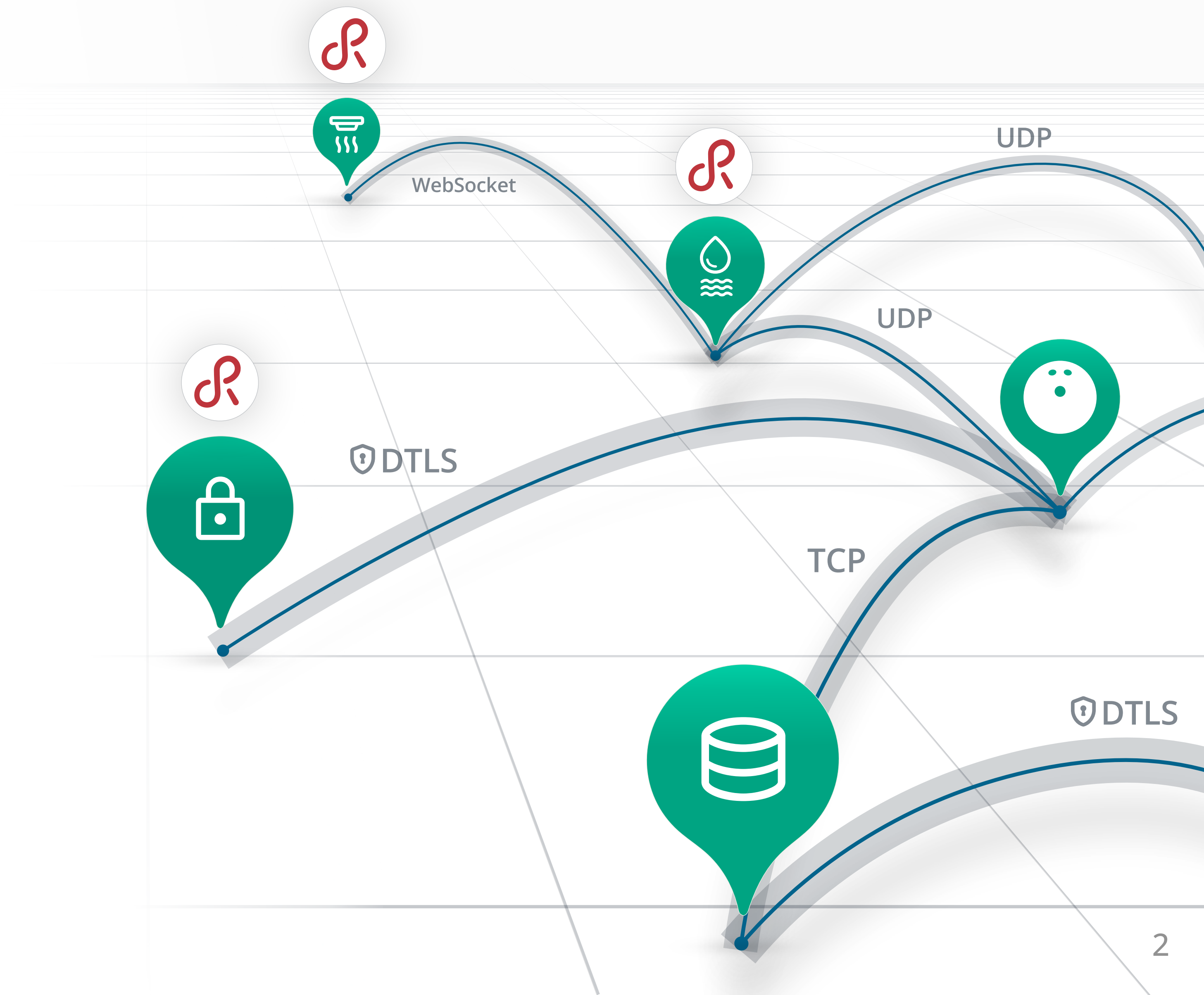
→ High-level, extensible API

→ Transport abstraction

 Layering

 Modularization

Implementation in  RIOT



CoAP in RIOT

CoAP in RIOT

Feature	nanoCoAP Sock	GCoAP
---------	---------------	-------

CoAP in RIOT

Feature	nanoCoAP Sock	GCoAP
Parser	nanoCoAP	nanoCoAP

CoAP in RIOT

Feature	nanoCoAP Sock	GCoAP
Parser	nanoCoAP	nanoCoAP
Transport abstraction	×	×

CoAP in RIOT

Feature	nanoCoAP Sock	GCoAP
Parser	nanoCoAP	nanoCoAP
Transport abstraction	×	×
Client API	Only sync (limited)	Only async (limited)

CoAP in RIOT

Feature	nanoCoAP Sock	GCoAP
Parser	nanoCoAP	nanoCoAP
Transport abstraction	×	×
Client API	Only sync (limited)	Only async (limited)
OSCORE	×	×

CoAP in RIOT

Feature	nanoCoAP Sock	GCoAP
Parser	nanoCoAP	nanoCoAP
Transport abstraction	×	×
Client API	Only sync (limited)	Only async (limited)
OSCORE	×	×
Resource observation	✓	✓

CoAP in RIOT

Feature	nanoCoAP Sock	GCoAP
Parser	nanoCoAP	nanoCoAP
Transport abstraction	✗	✗
Client API	Only sync (limited)	Only async (limited)
OSCORE	✗	✗
Resource observation	✓	✓
Automatic block-wise transfers	partially	partially

CoAP in RIOT

Feature	nanoCoAP Sock	GCoAP	unicoap
Parser	nanoCoAP	nanoCoAP	Convenient parser
Transport abstraction	✗	✗	✓
Client API	Only sync (limited)	Only async (limited)	Sync and async
OSCORE	✗	✗	✓
Resource observation	✓	✓	✓
Automatic block-wise transfers	partially	partially	✓

unicoap

Design and Implementation

DESIGN

Layered Architecture

DESIGN

Layered Architecture



DESIGN

Layered Architecture

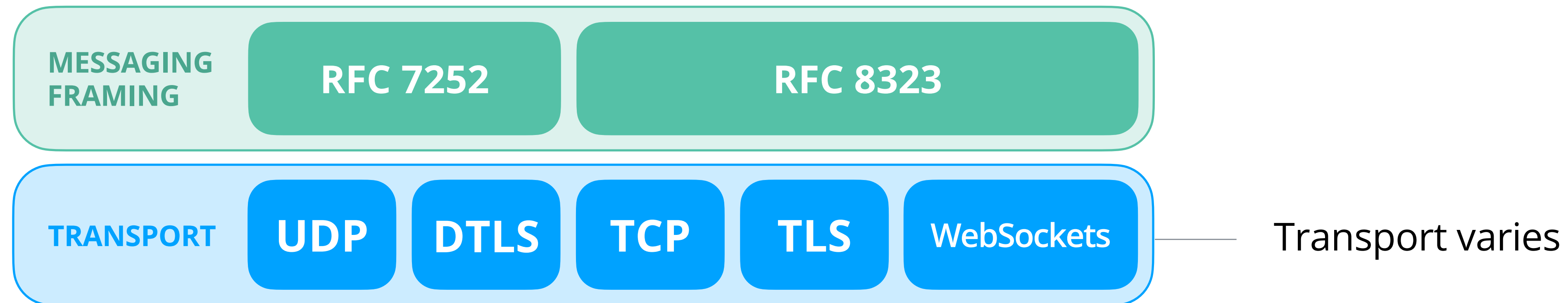


DESIGN

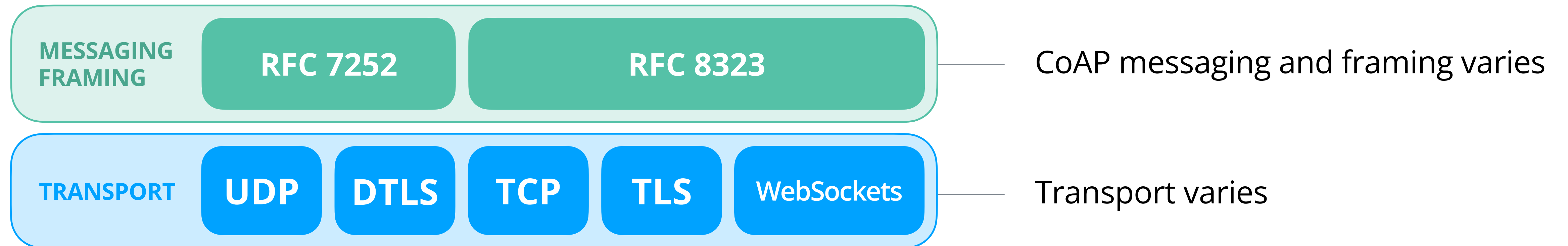
Layered Architecture



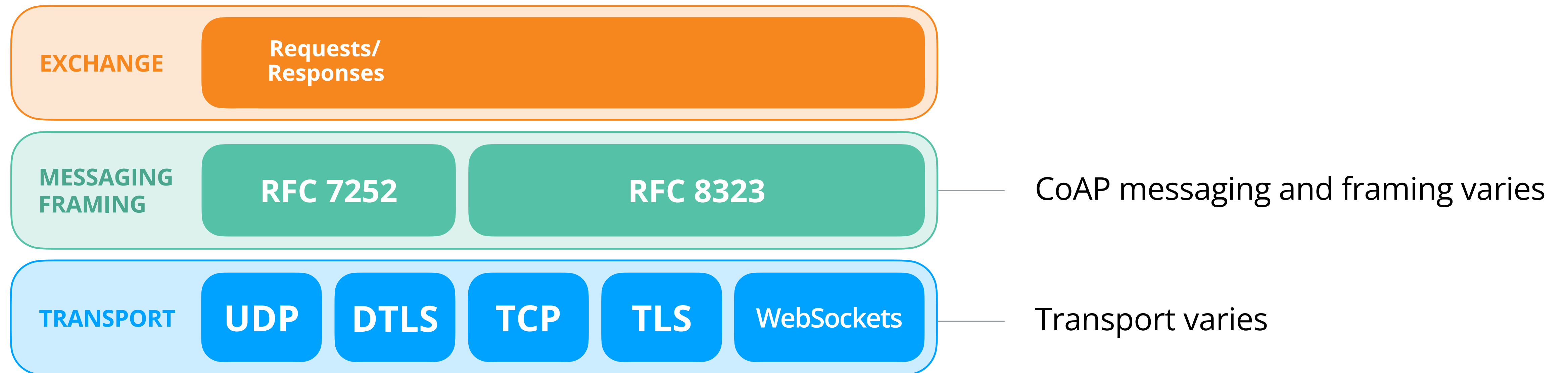
Layered Architecture



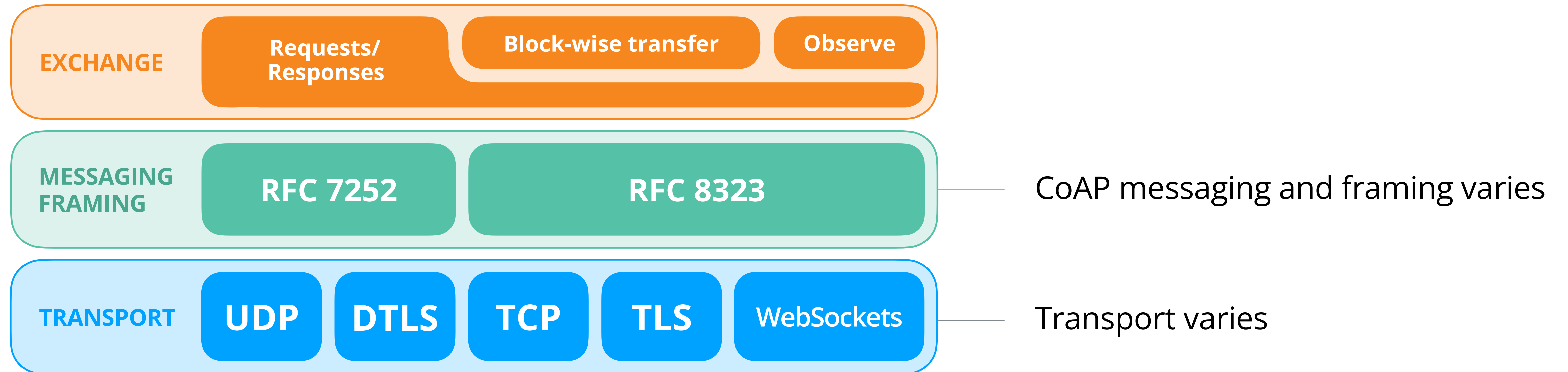
Layered Architecture



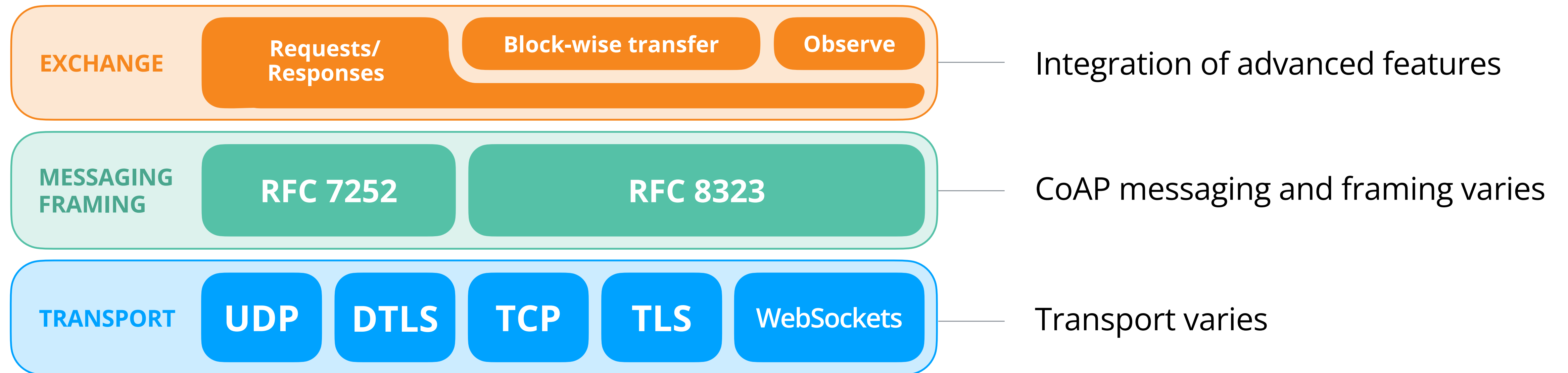
Layered Architecture



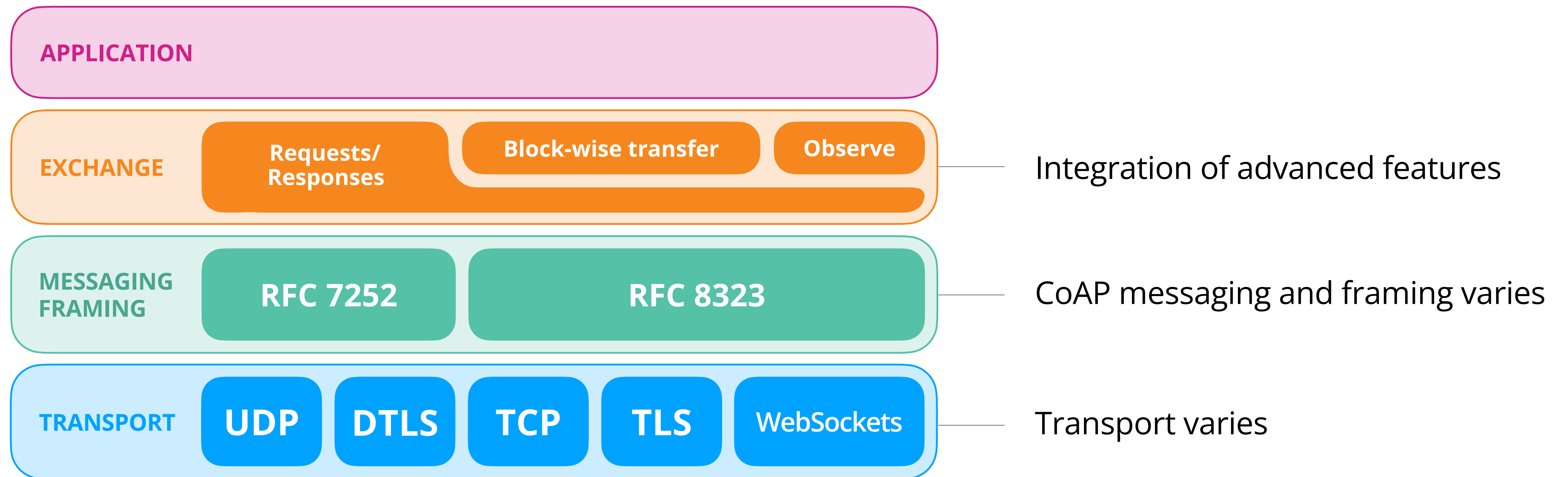
Layered Architecture



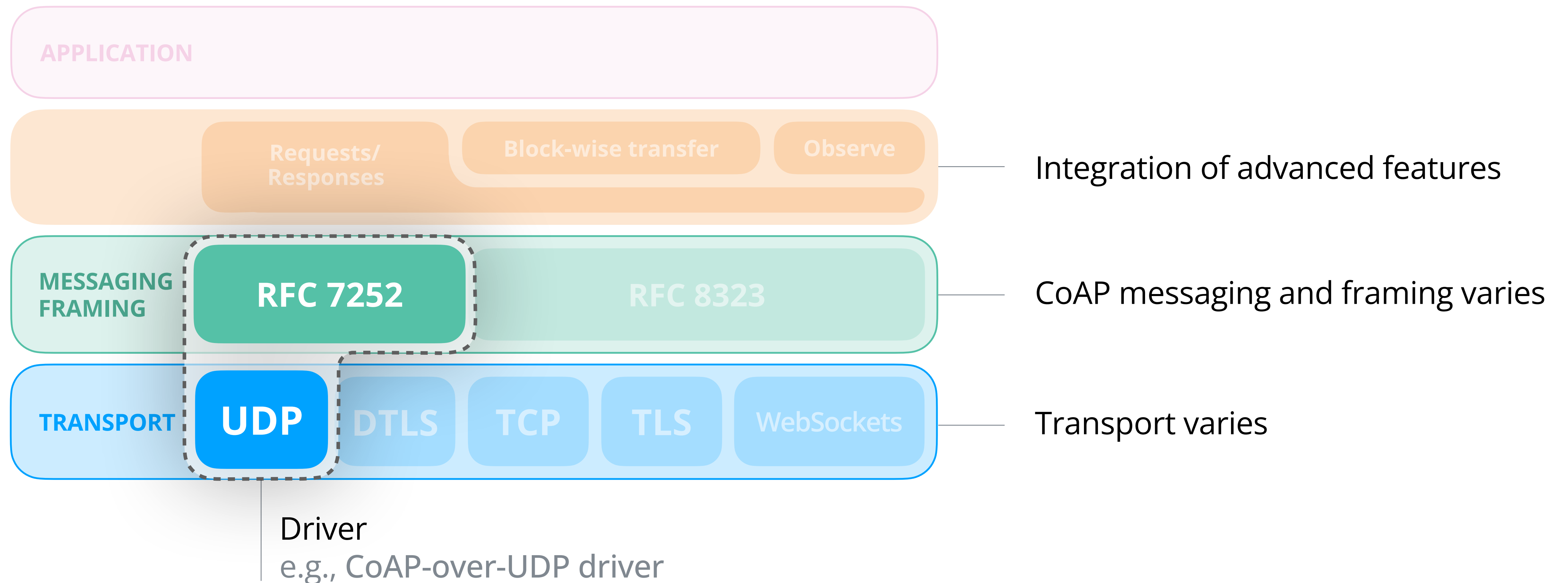
Layered Architecture



Layered Architecture



Layered Architecture



DESIGN

Layered Architecture

APPLICATION

EXCHANGE

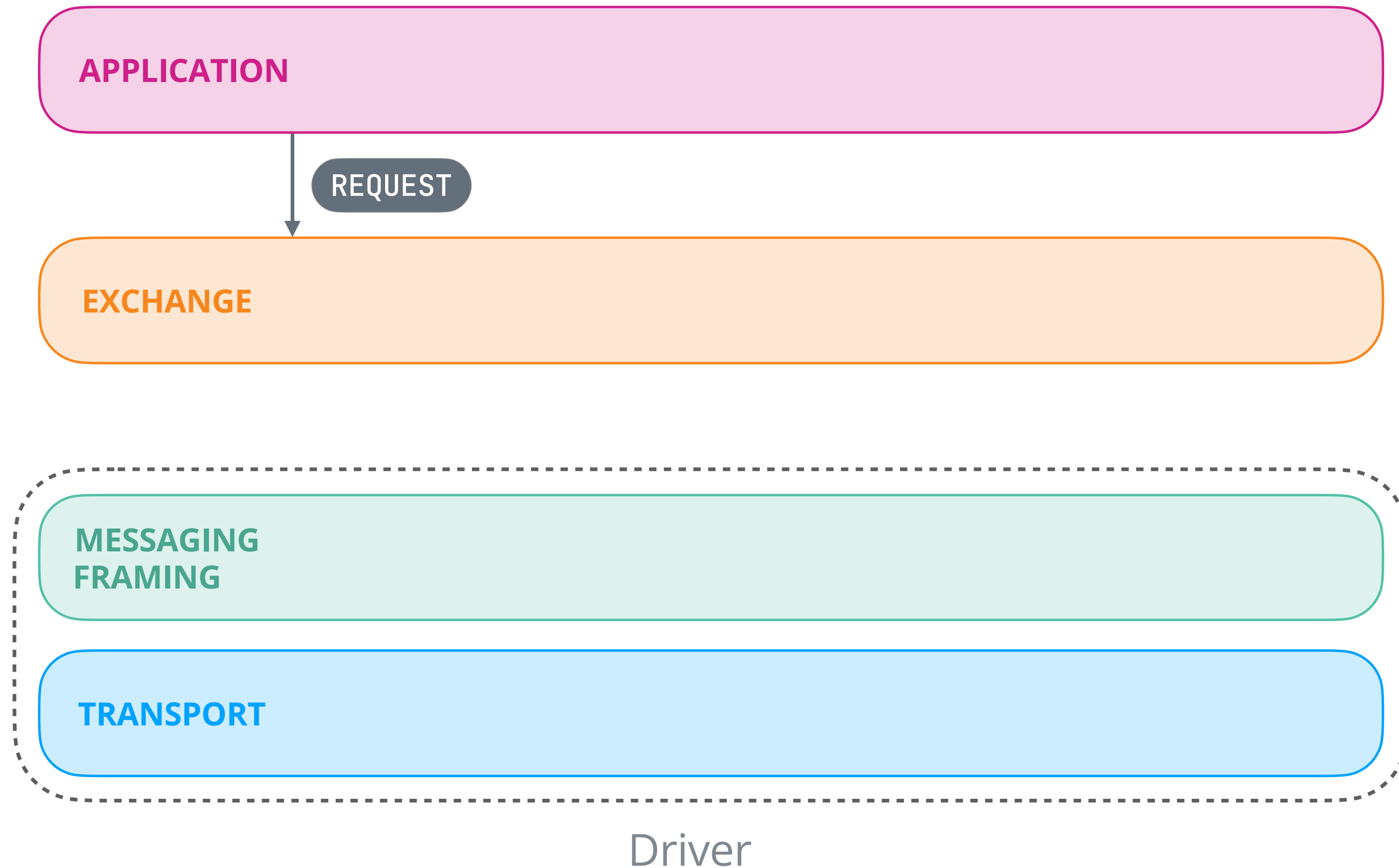
MESSAGING
FRAMING

TRANSPORT

Driver

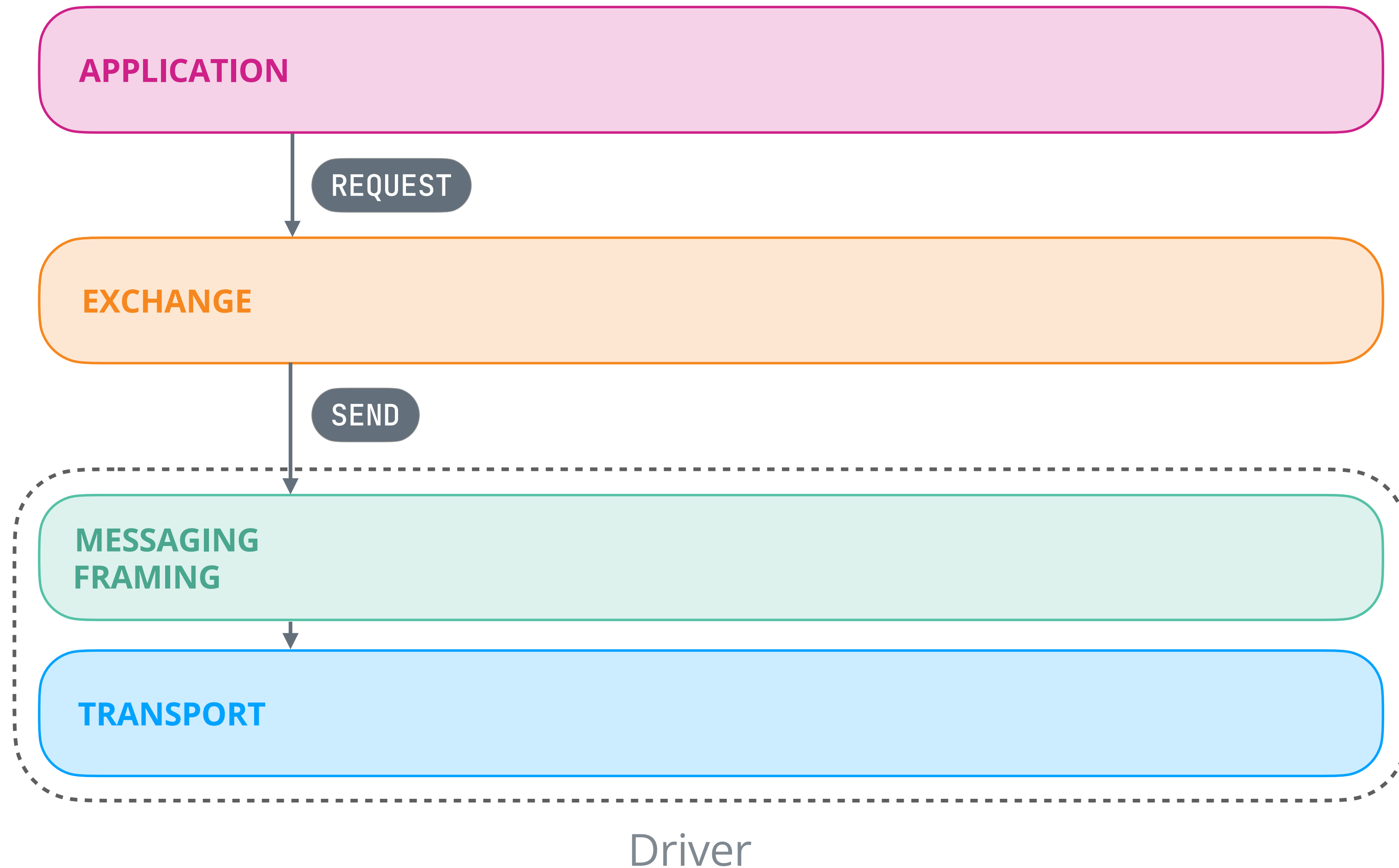
DESIGN

Layered Architecture

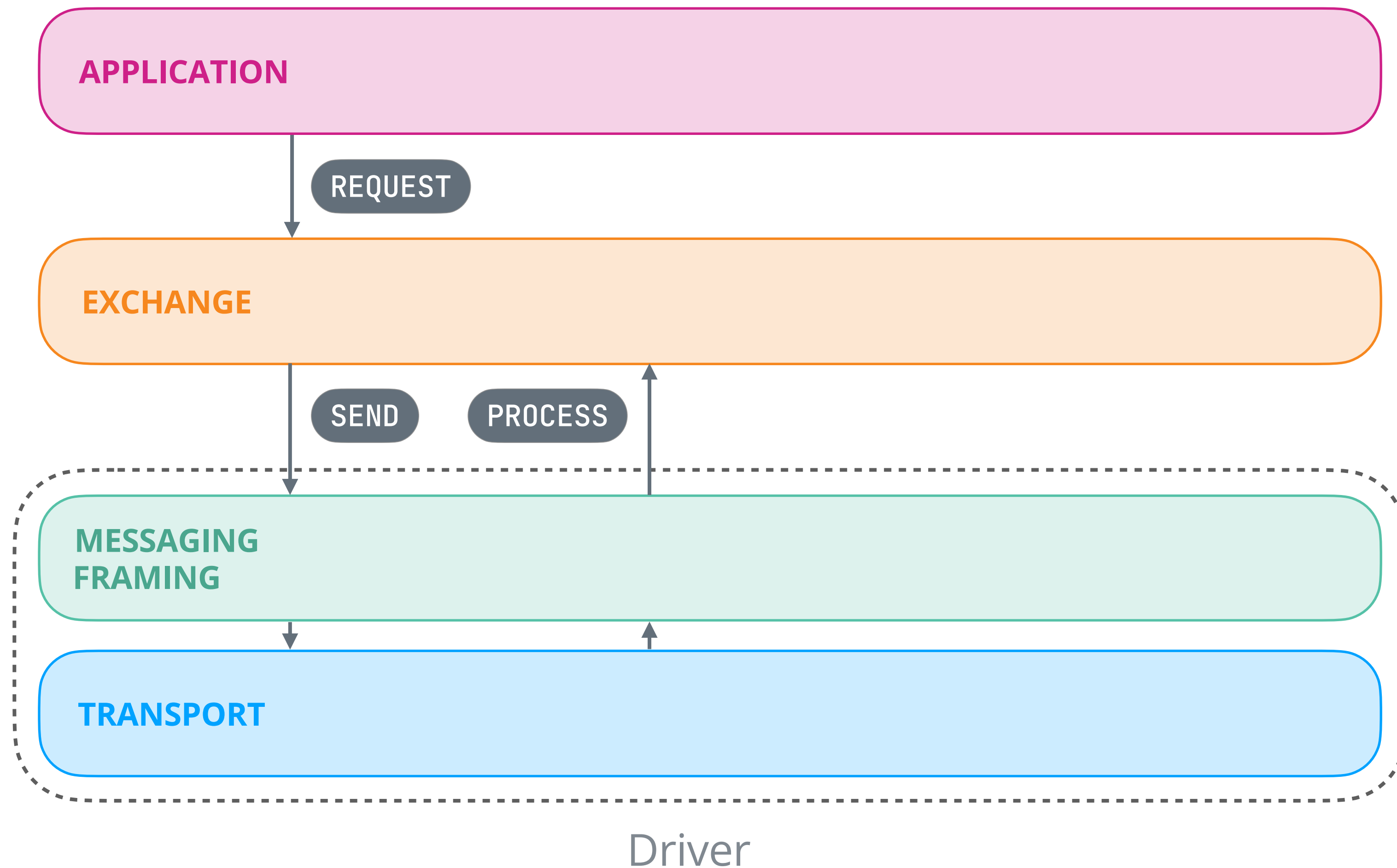


DESIGN

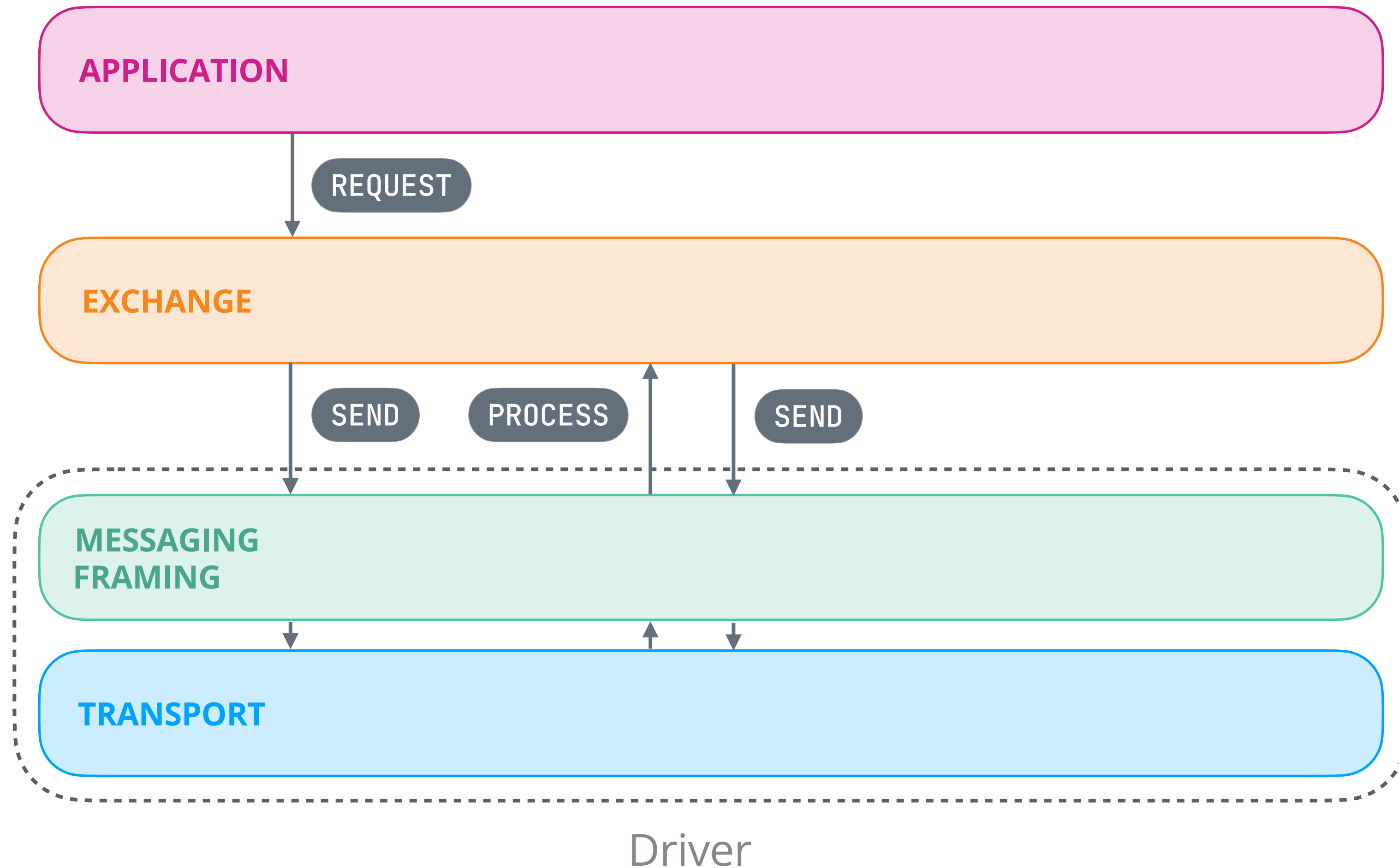
Layered Architecture



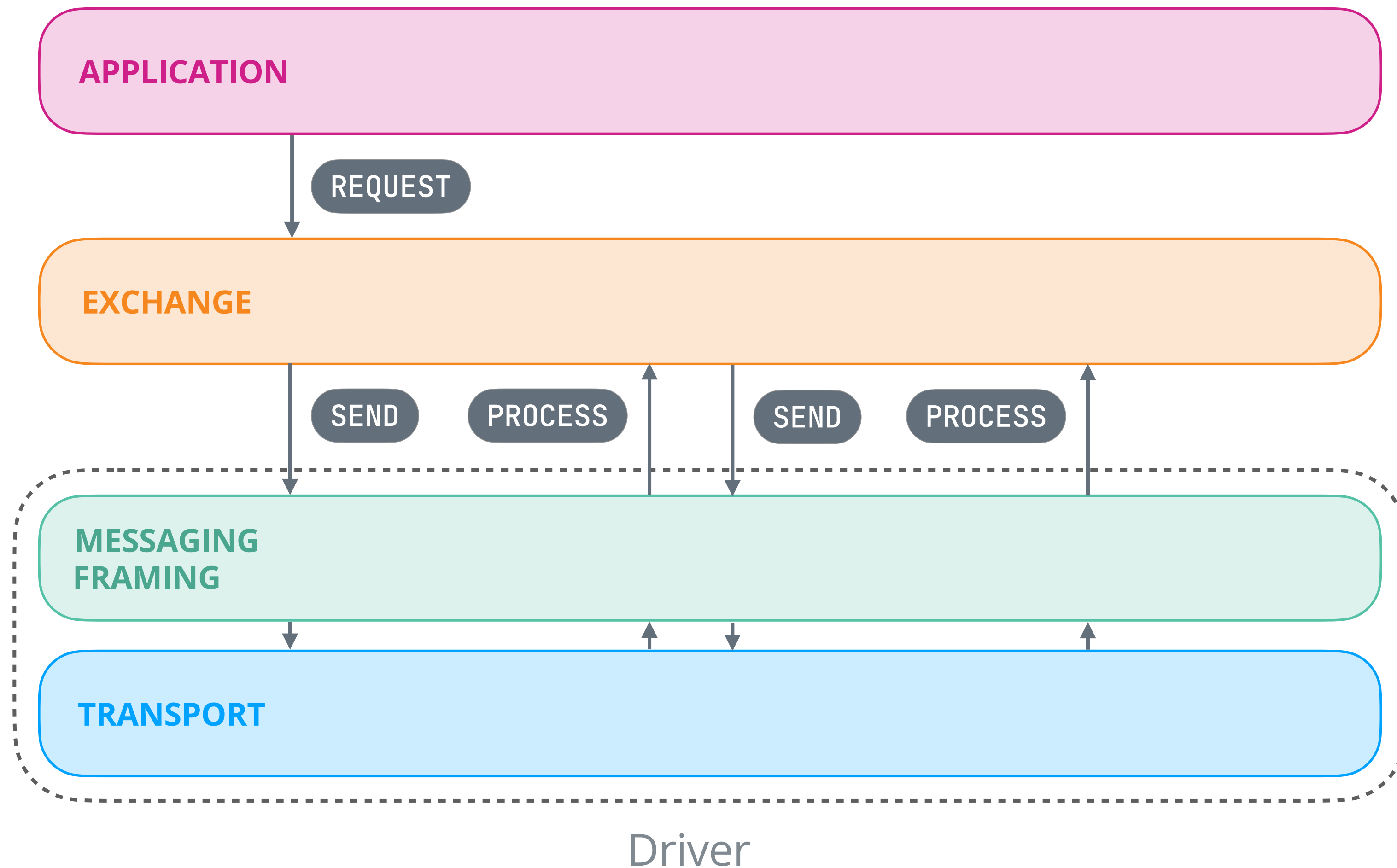
Layered Architecture



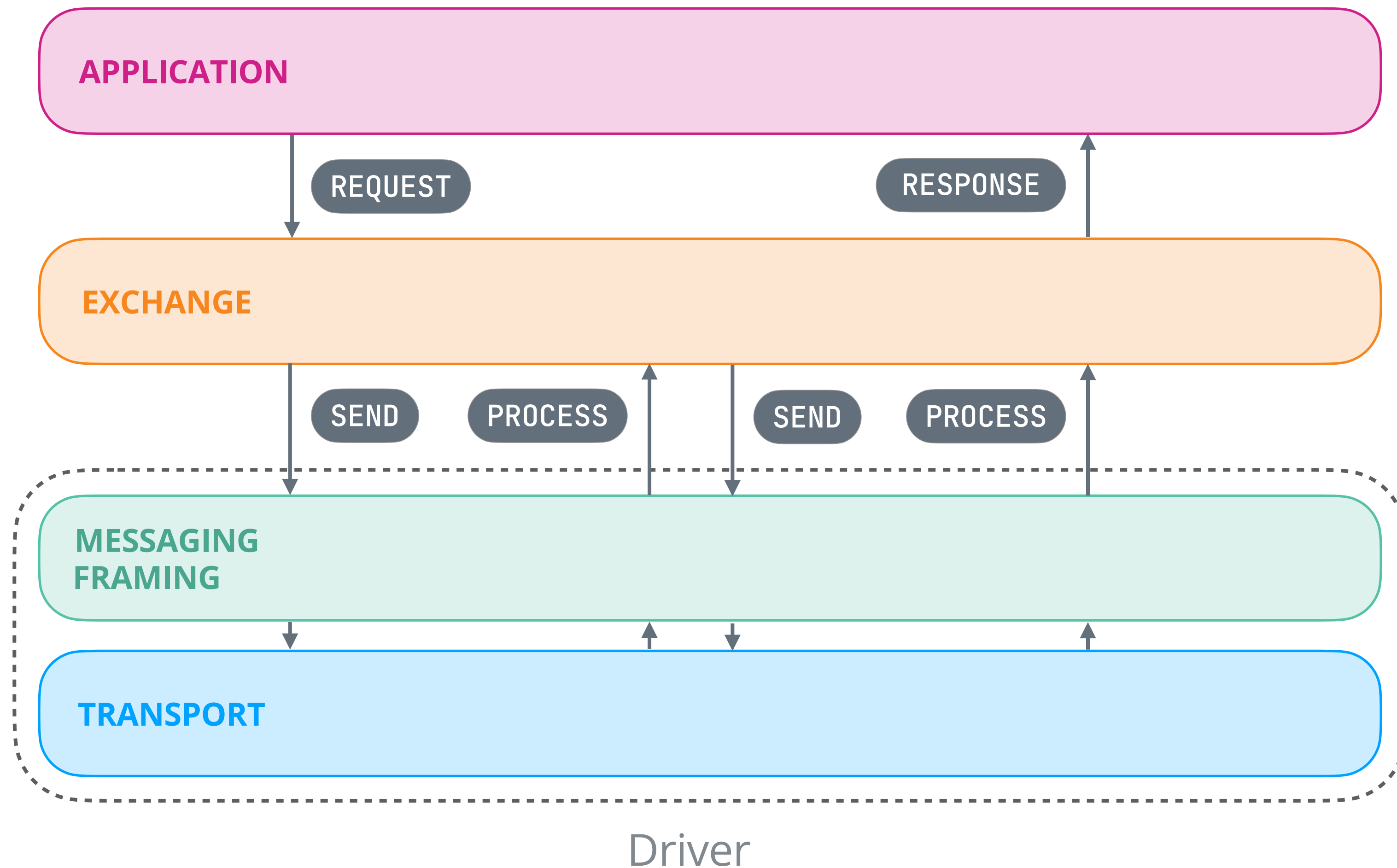
Layered Architecture



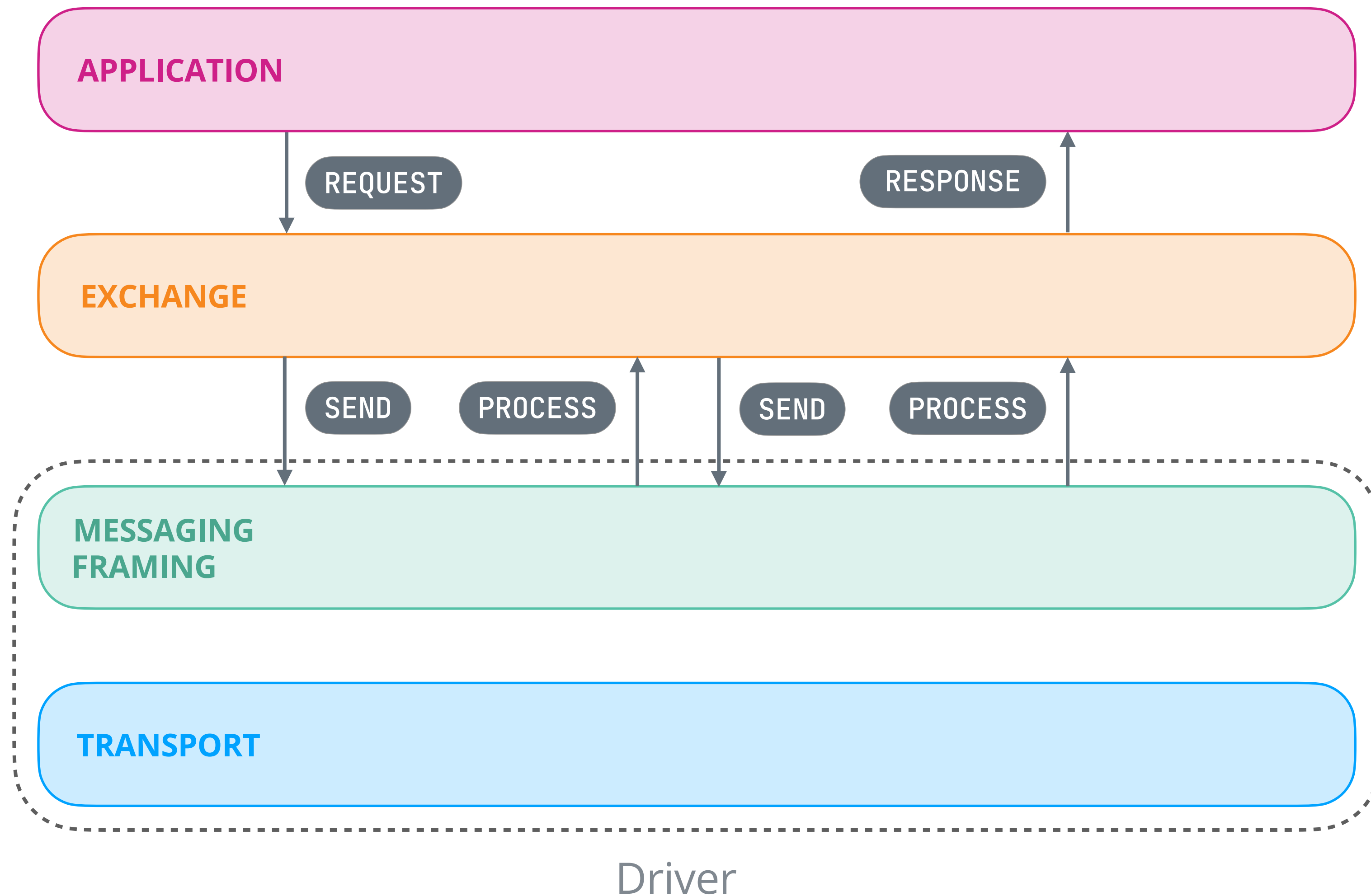
Layered Architecture



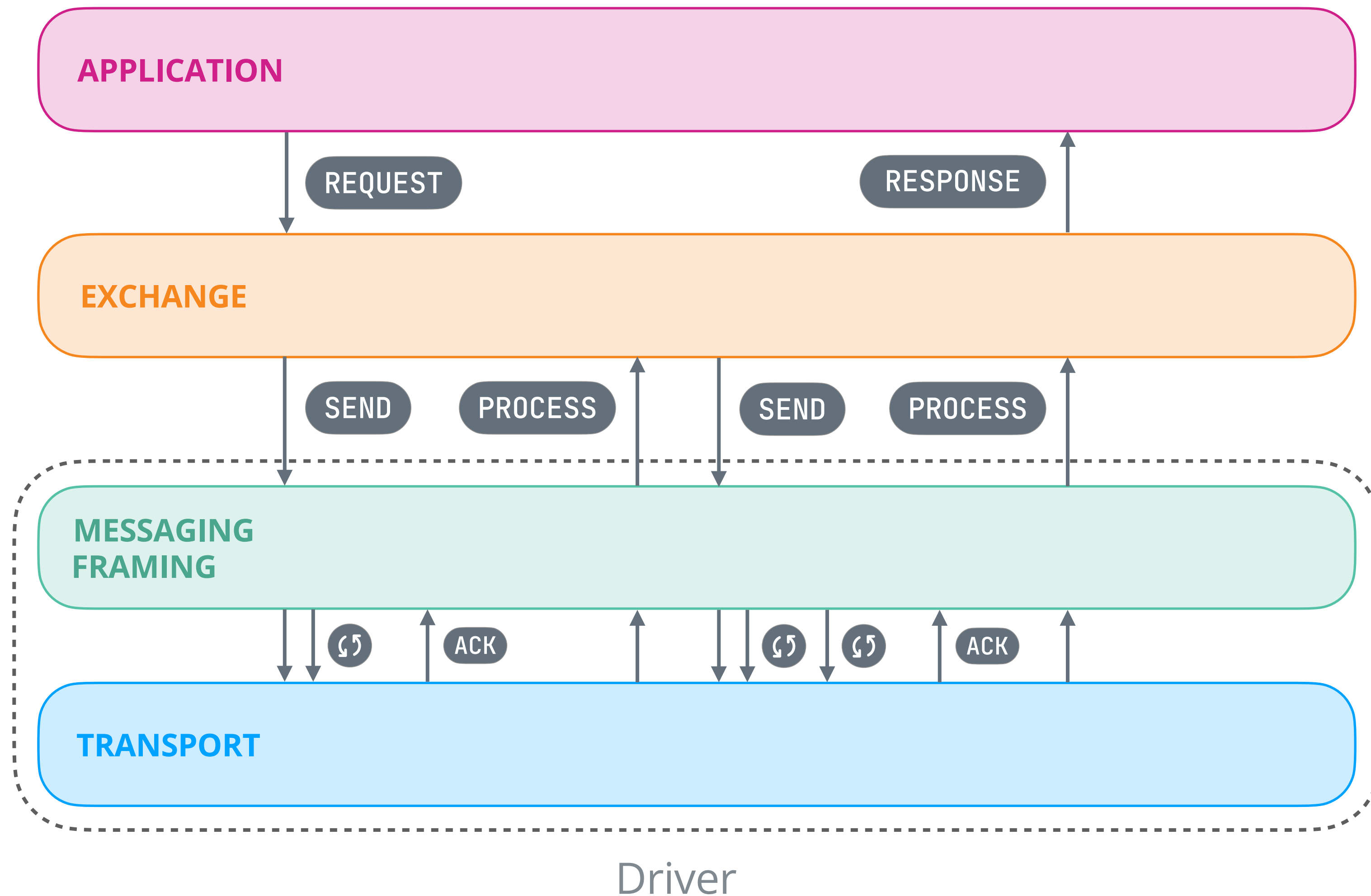
Layered Architecture



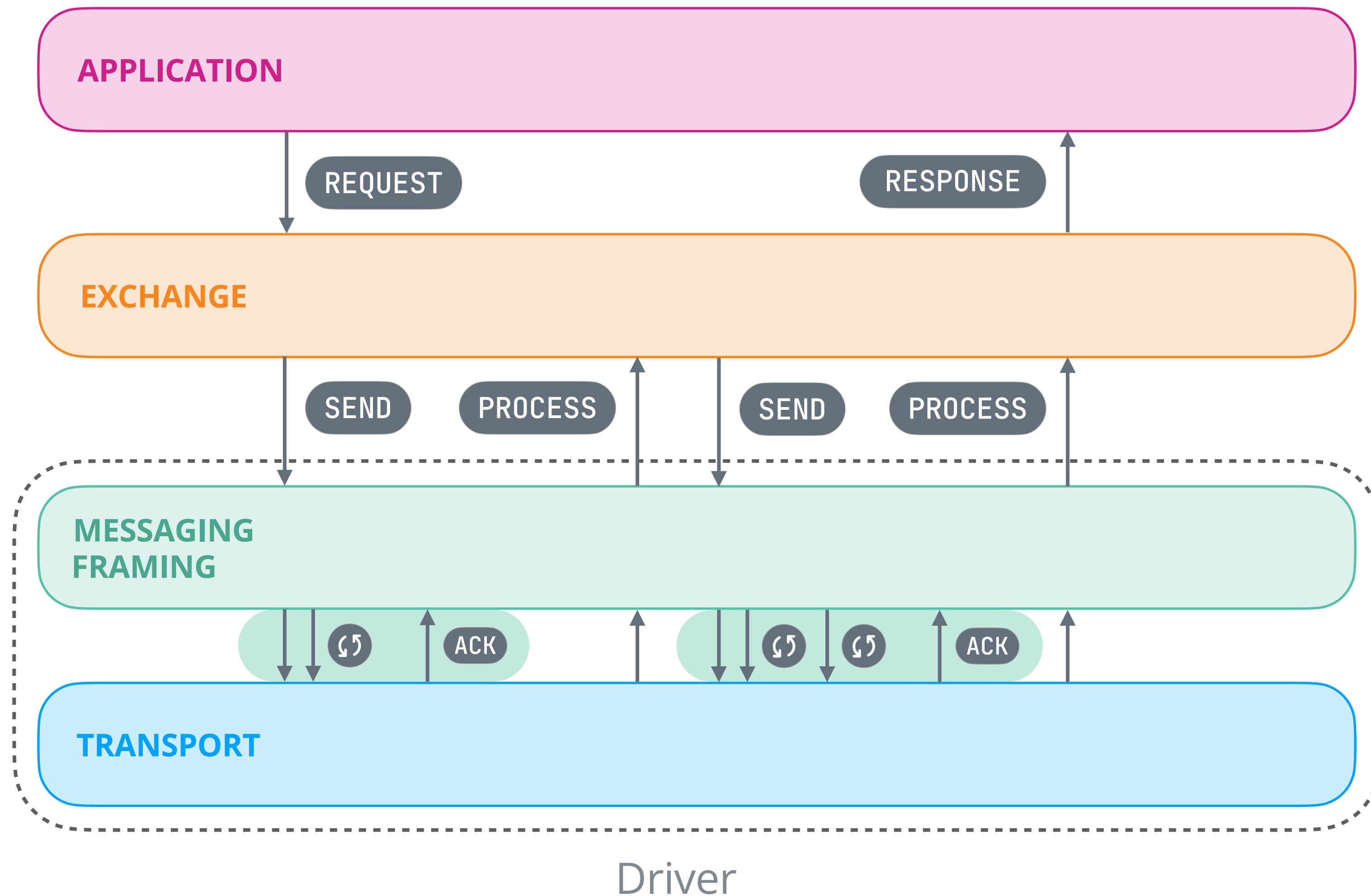
Layered Architecture



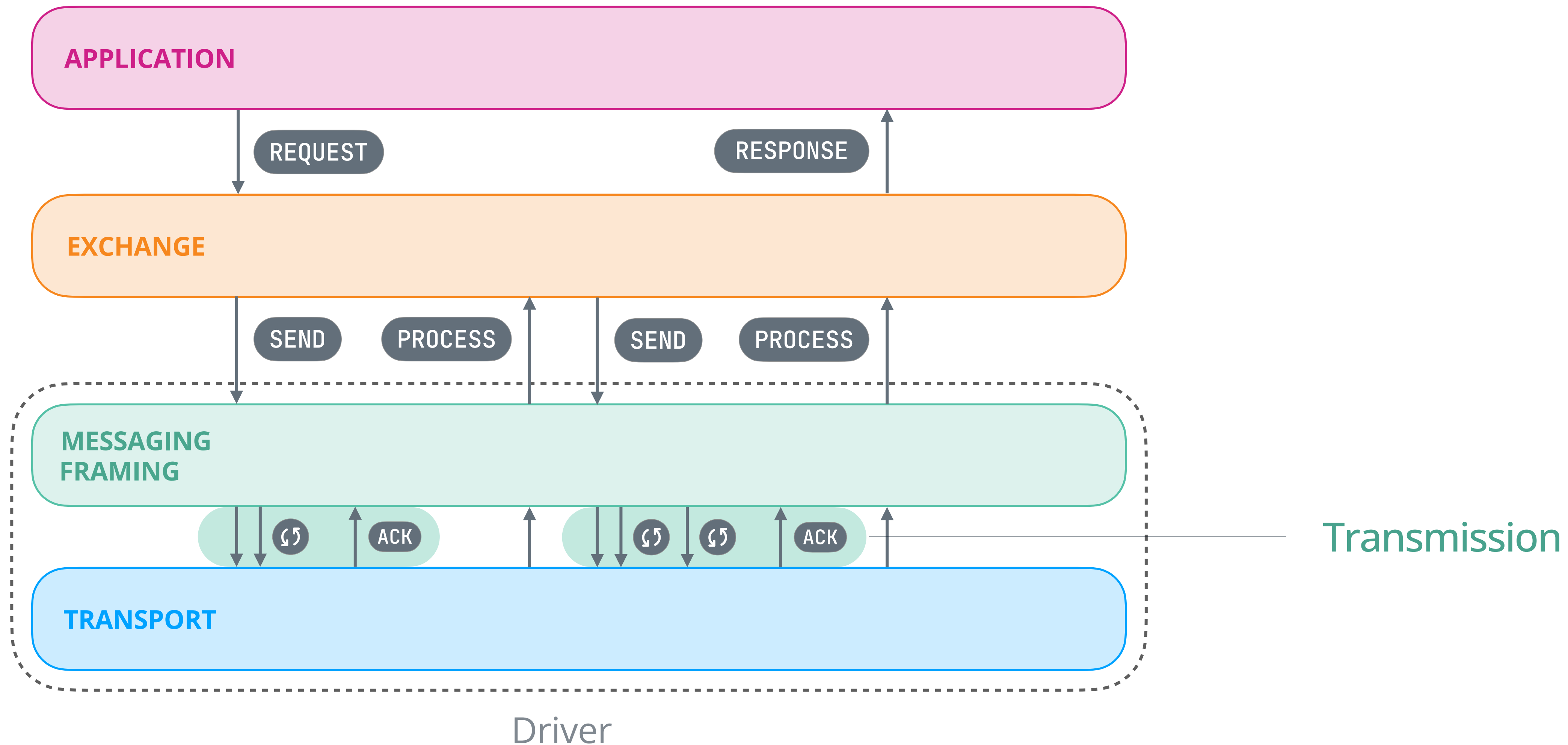
Layered Architecture



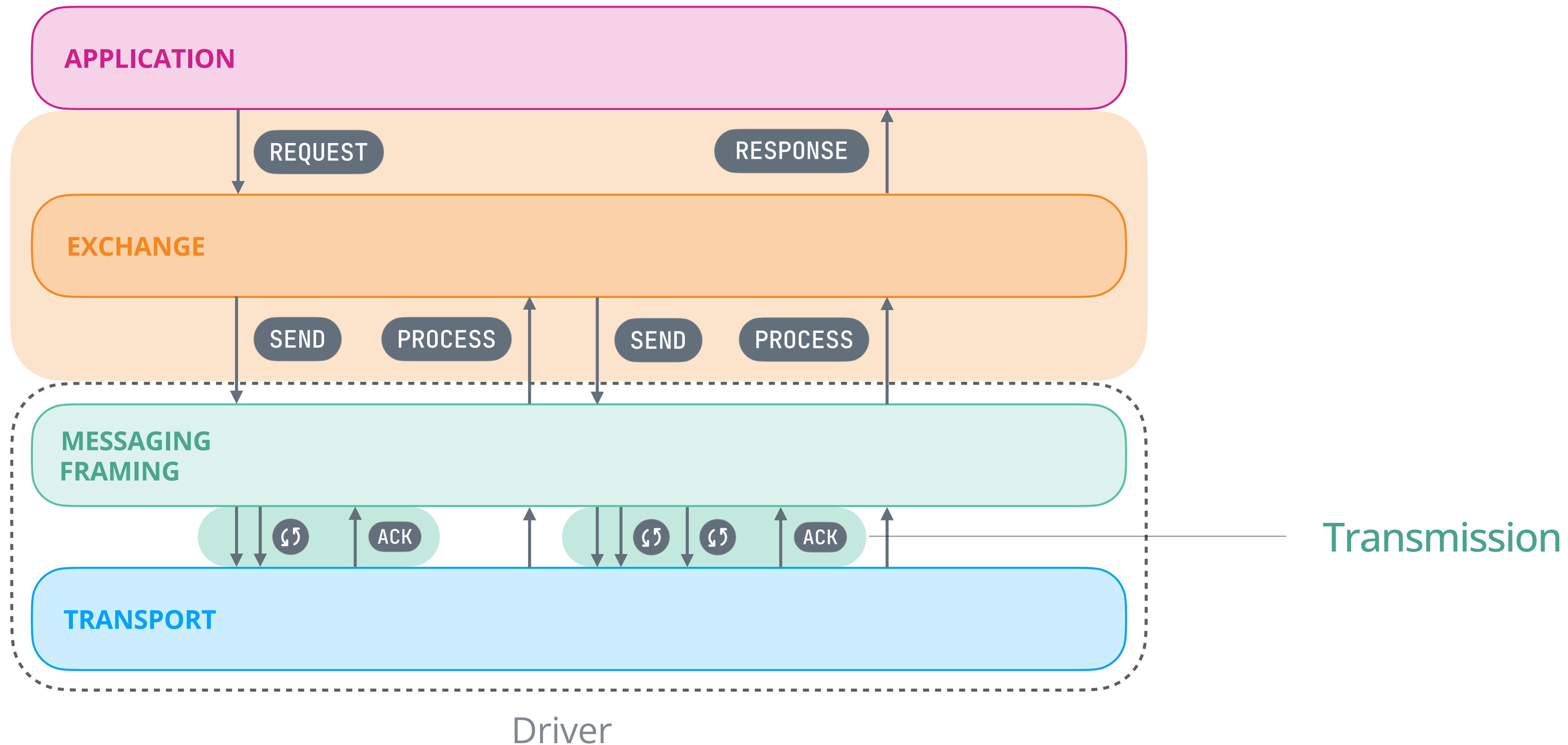
Layered Architecture



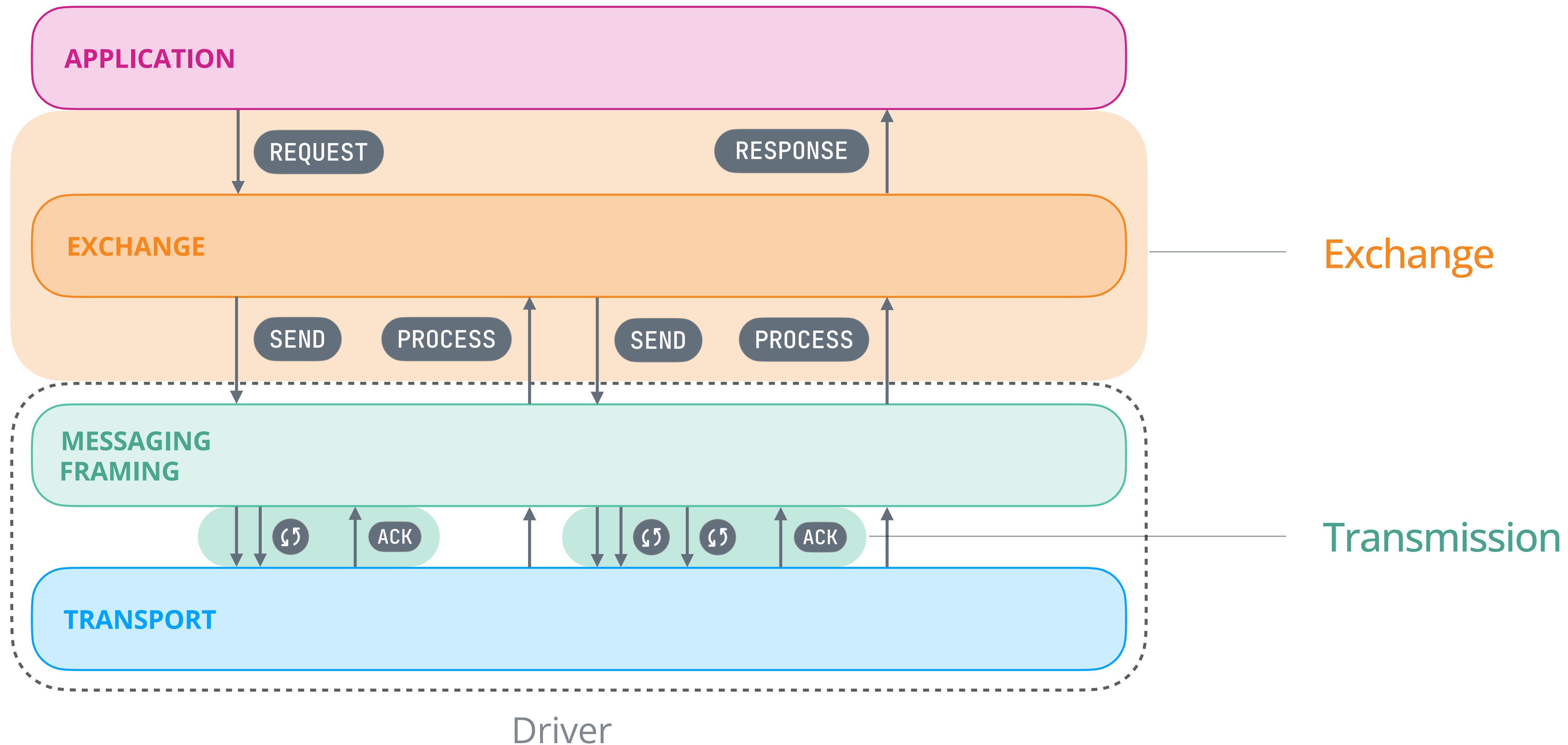
Layered Architecture



Layered Architecture



Layered Architecture



IMPLEMENTATION

Client API

```
static int handle_response(  
    const uniconap_message_t* response, const uniconap_aux_t* aux,  
    int error, void* arg) {  
  
    printf("status=%s (%zu bytes)\n",  
        uniconap_label_from_status(response->status), response->payload_size);  
  
    return 0;  
}  
  
static int main(void) {  
    uniconap_message_t request = { .method = UNICONAP_METHOD_GET };  
    uniconap_destination_t destination =  
        uniconap_destination_uri("coap://example.org:10123/led/color?a=1&b=2");  
    int res = uniconap_send_request_async(&request, &destination,  
        handle_response, UNICONAP_CLIENT_FLAG_RELIABLE, NULL);  
  
    // ...  
    return 0;  
}
```

IMPLEMENTATION

Client API

```
static int handle_response(  
    const uniconap_message_t* response, const uniconap_aux_t* aux,  
    int error, void* arg) {  
  
    printf("status=%s (%zu bytes)\n",  
        uniconap_label_from_status(response->status), response->payload_size);  
  
    return 0;  
}  
  
static int main(void) {  
    uniconap_message_t request = { .method = UNICONAP_METHOD_GET };  
    uniconap_destination_t destination =  
        uniconap_destination_uri("coap://example.org:10123/led/color?a=1&b=2");  
    int res = uniconap_send_request_async(&request, &destination,  
        handle_response, UNICONAP_CLIENT_FLAG_RELIABLE, NULL);  
  
    // ...  
    return 0;  
}
```

IMPLEMENTATION

Client API

```
static int handle_response(  
    const uniconap_message_t* response, const uniconap_aux_t* aux,  
    int error, void* arg) {  
  
    printf("status=%s (%zu bytes)\n",  
        uniconap_label_from_status(response->status), response->payload_size);  
  
    return 0;  
}  
  
static int main(void) {  
    uniconap_message_t request = { .method = UNICONAP_METHOD_GET };  
    uniconap_destination_t destination =  
        uniconap_destination_uri("coap://example.org:10123/led/color?a=1&b=2");  
    int res = uniconap_send_request_async(&request, &destination,  
        handle_response, UNICONAP_CLIENT_FLAG_RELIABLE, NULL);  
  
    // ...  
    return 0;  
}
```

IMPLEMENTATION

Client API

```
static int handle_response(  
    const unicoap_message_t* response, const unicoap_aux_t* aux,  
    int error, void* arg) {  
  
    printf("status=%s (%zu bytes)\n",  
        unicoap_label_from_status(response->status), response->payload_size);  
  
    return 0;  
}  
  
static int main(void) {  
    unicoap_message_t request = { .method = UNICOAP_METHOD_GET };  
    sock_udp_ep_t udp_ep = { ... };  
    unicoap_destination_t destination = unicoap_destination_endpoint(&udp_ep);  
    int res = unicoap_send_request_async(&request, &destination,  
        handle_response, UNICOAP_CLIENT_FLAG_RELIABLE, NULL);  
  
    // ...  
    return 0;  
}
```

IMPLEMENTATION

Client API

DESTINATION

type

○

endpoint

○

OR

uri

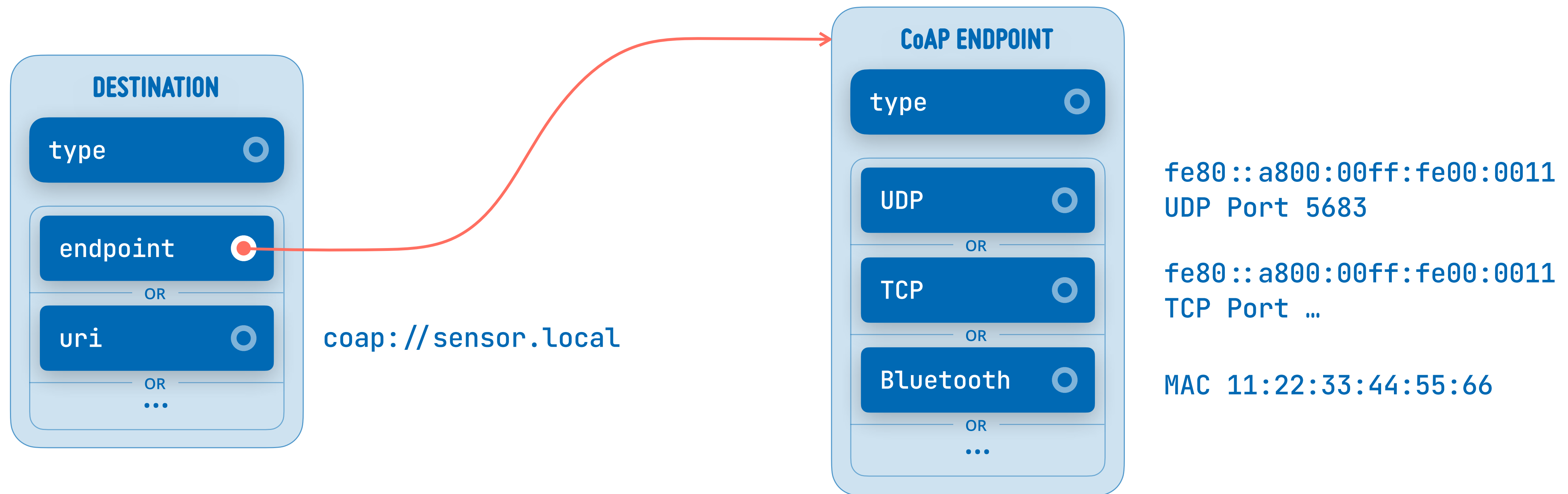
○

OR

...

`coap://sensor.local`

Client API



IMPLEMENTATION

Client API

```
static int handle_response(  
    const uniconap_message_t* response, const uniconap_aux_t* aux,  
    int error, void* arg) {  
  
    printf("status=%s (%zu bytes)\n",  
        uniconap_label_from_status(response->status), response->payload_size);  
  
    return 0;  
}  
  
static int main(void) {  
    uniconap_message_t request = { .method = UNICONAP_METHOD_GET };  
    uniconap_destination_t destination =  
        uniconap_destination_uri("coap://example.org:10123/led/color?a=1&b=2");  
    int res = uniconap_send_request_async(&request, &destination,  
        handle_response, UNICONAP_CLIENT_FLAG_RELIABLE, NULL);  
  
    // ...  
    return 0;  
}
```

IMPLEMENTATION

Client API

```
static int handle_response(  
    const uniconap_message_t* response, const uniconap_aux_t* aux,  
    int error, void* arg) {  
  
    printf("status=%s (%zu bytes)\n",  
        uniconap_label_from_status(response->status), response->payload_size);  
  
    return 0;  
}  
  
static int main(void) {  
    uniconap_message_t request = { .method = UNICONAP_METHOD_GET };  
    uniconap_destination_t destination =  
        uniconap_destination_uri("coap://example.org:10123/led/color?a=1&b=2");  
    int res = uniconap_send_request_async(&request, &destination,  
        handle_response, UNICONAP_CLIENT_FLAG_RELIABLE, NULL);  
  
    // ...  
    return 0;  
}
```

MODULE
uniconap_uri

IMPLEMENTATION

Client API

```
static int handle_response(  
    const unicoap_message_t* response, const unicoap_aux_t* aux,  
    int error, void* arg) {  
  
    printf("status=%s (%zu bytes)\n",  
        unicoap_label_from_status(response->status), response->payload_size);  
  
    return 0;  
}  
  
static int main(void) {  
    unicoap_message_t request = { .method = UNICOAP_METHOD_GET };  
    unicoap_destination_t destination =  
        unicoap_destination_uri("coap://example.org:10123/led/color?a=1&b=2");  
    int res = unicoap_send_request_async(&request, &destination,  
        handle_response, UNICOAP_CLIENT_FLAG_RELIABLE, NULL);  
  
    // ...  
    return 0;  
}
```

IMPLEMENTATION

Client API

```
static int handle_response(  
    const uniconap_message_t* response, const uniconap_aux_t* aux,  
    int error, void* arg) {  
  
    printf("status=%s (%zu bytes)\n",  
        uniconap_label_from_status(response->status), response->payload_size);  
  
    return 0;  
}  
  
static int main(void) {  
    uniconap_message_t request = { .method = UNICONAP_METHOD_GET };  
    uniconap_destination_t destination =  
        uniconap_destination_uri("coap://example.org:10123/led/color?a=1&b=2");  
    int res = uniconap_send_request_async(&request, &destination,  
        handle_response, UNICONAP_CLIENT_FLAG_RELIABLE, NULL);  
  
    // ...  
    return 0;  
}
```

IMPLEMENTATION

Client API

```
static int handle_response(  
    const uniconap_message_t* response, const uniconap_aux_t* aux,  
    int error, void* arg) {  
  
    printf("status=%s (%zu bytes)\n",  
        uniconap_label_from_status(response->status), response->payload_size);  
  
    return 0;  
}  
  
static int main(void) {  
    uniconap_message_t request = { .method = UNICOAP_METHOD_GET };  
    uniconap_destination_t destination =  
        uniconap_destination_uri("coap://example.org:10123/led/color?a=1&b=2");  
    int res = uniconap_send_request_sync(&request, &destination,  
        handle_response, UNICOAP_CLIENT_FLAG_RELIABLE, NULL);  
  
    // ...  
    return 0;  
}
```

IMPLEMENTATION

Client API

```
static int handle_response(  
    const uniconap_message_t* response, const uniconap_aux_t* aux,  
    int error, void* arg) {  
  
    printf("status=%s (%zu bytes)\n",  
        uniconap_label_from_status(response->status), response->payload_size);  
  
    return 0;  
}  
  
static int main(void) {  
    uniconap_message_t request = { .method = UNICONAP_METHOD_GET };  
    uniconap_destination_t destination =  
        uniconap_destination_uri("coap://example.org:10123/led/color?a=1&b=2");  
    int res = uniconap_send_request_async(&request, &destination,  
        handle_response, UNICONAP_CLIENT_FLAG_RELIABLE, NULL);  
  
    // ...  
    return 0;  
}
```

IMPLEMENTATION

Client API

```
static int handle_response(  
    const uniconap_message_t* response, const uniconap_aux_t* aux,  
    int error, void* arg) {  
  
    printf("status=%s (%zu bytes)\n",  
        uniconap_label_from_status(response->status), response->payload_size);  
  
    return 0;  
}  
  
static int main(void) {  
    uniconap_message_t request = { .method = UNICONAP_METHOD_GET };  
    uniconap_destination_t destination =  
        uniconap_destination_uri("coap://example.org:10123/led/color?a=1&b=2");  
    int res = uniconap_send_request_async(&request, &destination,  
        handle_response, UNICONAP_CLIENT_FLAG_RELIABLE, NULL);  
  
    // ...  
    return 0;  
}
```

IMPLEMENTATION

Client API

```
static int handle_response(  
    const uniconap_message_t* response, const uniconap_aux_t* aux,  
    int error, void* arg) {  
  
    printf("status=%s (%zu bytes)\n",  
        uniconap_label_from_status(response->status), response->payload_size);  
  
    return 0;  
}  
  
static int main(void) {  
    uniconap_message_t request = { .method = UNICONAP_METHOD_GET };  
    uniconap_destination_t destination =  
        uniconap_destination_uri("coap://example.org:10123/led/color?a=1&b=2");  
    int res = uniconap_send_request_async(&request, &destination,  
        handle_response, UNICONAP_CLIENT_FLAG_RELIABLE | UNICONAP_CLIENT_FLAG_REASSEMBLE, NULL);  
  
    // ...  
    return 0;  
}
```

IMPLEMENTATION

Client API

```
static int handle_response(  
    const uniconap_message_t* response, const uniconap_aux_t* aux,  
    int error, void* arg) {  
  
    printf("status=%s (%zu bytes)\n",  
        uniconap_label_from_status(response->status), response->payload_size);  
  
    return 0;  
}  
  
static int main(void) {  
    uniconap_message_t request = { .method = UNICONAP_METHOD_GET };  
    uniconap_destination_t destination =  
        uniconap_destination_uri("coap://example.org:10123/led/color?a=1&b=2");  
    int res = uniconap_send_request_async(&request, &destination,  
        handle_response, UNICONAP_CLIENT_FLAG_RELIABLE | UNICONAP_CLIENT_FLAG_REASSEMBLE, NULL);  
  
    // ...  
    return 0;  
}
```

MODULE
uniconap_blockwise

IMPLEMENTATION

Client API

```
static int handle_response(  
    const uniconap_message_t* response, const uniconap_aux_t* aux,  
    int error, void* arg) {  
  
    printf("status=%s (%zu bytes)\n",  
        uniconap_label_from_status(response->status), response->payload_size);  
  
    return 0;  
}  
  
static int main(void) {  
    uniconap_message_t request = { .method = UNICONAP_METHOD_GET };  
    uniconap_destination_t destination =  
        uniconap_destination_uri("coap://example.org:10123/led/color?a=1&b=2");  
    int res = uniconap_send_request_async(&request, &destination,  
        handle_response, UNICONAP_CLIENT_FLAG_RELIABLE, NULL);  
  
    // ...  
    return 0;  
}
```

IMPLEMENTATION

Client API

```
static int handle_response(  
    const uniconap_message_t* response, const uniconap_aux_t* aux,  
    int error, void* arg) {  
  
    printf("status=%s (%zu bytes)\n",  
        uniconap_label_from_status(response->status), response->payload_size);  
  
    return 0;  
}  
  
static int main(void) {  
    uniconap_message_t request = { .method = UNICONAP_METHOD_GET };  
    uniconap_destination_t destination =  
        uniconap_destination_uri("coap://example.org:10123/led/color?a=1&b=2");  
    int res = uniconap_send_request_async(&request, &destination,  
        handle_response, UNICONAP_CLIENT_FLAG_RELIABLE, NULL);  
  
    // ...  
    return 0;  
}
```

IMPLEMENTATION

Server API

```
UNICOAP_RESOURCE(my_resource) {  
    .path = "/led/color",  
    .handler = handle_color_request,  
    .methods = UNICOAP_METHODS(UNICOAP_METHOD_GET),  
};
```

```
static int handle_color_request(  
    uniconap_message_t* message, const uniconap_aux_t* aux,  
    uniconap_request_context_t* context, void* arg) {  
  
    uniconap_response_init_string(&message, UNICOAP_STATUS_CONTENT, "rebeccapurple");  
    return uniconap_send_response(&message, context);  
}
```

IMPLEMENTATION

Server API

```
UNICOAP_RESOURCE(my_resource) {  
    .path = "/led/color",  
    .handler = handle_color_request,  
    .methods = UNICOAP_METHODS(UNICOAP_METHOD_GET),  
};
```

```
static int handle_color_request(  
    uniconap_message_t* message, const uniconap_aux_t* aux,  
    uniconap_request_context_t* context, void* arg) {  
  
    uniconap_response_init_string(&message, UNICOAP_STATUS_CONTENT, "rebeccapurple");  
    return uniconap_send_response(&message, context);  
}
```

IMPLEMENTATION

Server API

```
UNICOAP_RESOURCE(my_resource) {  
    .path = "/led/color",  
    .handler = handle_color_request,  
    .methods = UNICOAP_METHODS(UNICOAP_METHOD_GET),  
};
```

```
static int handle_color_request(  
    uniconap_message_t* message, const uniconap_aux_t* aux,  
    uniconap_request_context_t* context, void* arg) {  
  
    uniconap_response_init_string(&message, UNICOAP_STATUS_CONTENT, "rebeccapurple");  
    return uniconap_send_response(&message, context);  
}
```

IMPLEMENTATION

Server API

```
UNICOAP_RESOURCE(my_resource) {  
    .path = "/led/color",  
    .handler = handle_color_request,  
    .methods = UNICOAP_METHODS(UNICOAP_METHOD_GET),  
};
```

```
static int handle_color_request(  
    uniconap_message_t* message, const uniconap_aux_t* aux,  
    uniconap_request_context_t* context, void* arg) {  
  
    uniconap_response_init_string(&message, UNICOAP_STATUS_CONTENT, "rebeccapurple");  
    return uniconap_send_response(&message, context);  
}
```

IMPLEMENTATION

Server API

```
UNICOAP_RESOURCE(my_resource) {  
    .path = "/led/color",  
    .handler = handle_color_request,  
    .methods = UNICOAP_METHODS(UNICOAP_METHOD_GET, UNICOAP_METHOD_PUT),  
};
```

```
static int handle_color_request(  
    uniconap_message_t* message, const uniconap_aux_t* aux,  
    uniconap_request_context_t* context, void* arg) {  
  
    uniconap_response_init_string(&message, UNICOAP_STATUS_CONTENT, "rebeccapurple");  
    return uniconap_send_response(&message, context);  
}
```

IMPLEMENTATION

Server API

```
UNICOAP_RESOURCE(my_resource) {  
    .path = "/led/color",  
    .handler = handle_color_request,  
    .methods = UNICOAP_METHODS(UNICOAP_METHOD_GET, UNICOAP_METHOD_PUT),  
    .protocols = UNICOAP_PROTOCOLS(UNICOAP_PROTO_DTLS),  
};
```

```
static int handle_color_request(  
    uniconap_message_t* message, const uniconap_aux_t* aux,  
    uniconap_request_context_t* context, void* arg) {  
  
    uniconap_response_init_string(&message, UNICOAP_STATUS_CONTENT, "rebeccapurple");  
    return uniconap_send_response(&message, context);  
}
```

IMPLEMENTATION

Server API

```
UNICOAP_RESOURCE(my_resource) {  
    .path = "/led/color",  
    .handler = handle_color_request,  
    .methods = UNICOAP_METHODS(UNICOAP_METHOD_GET, UNICOAP_METHOD_PUT),  
    .protocols = UNICOAP_PROTOCOLS(UNICOAP_PROTO_DTLS),  
};
```

MODULE
unicoap_udp

MODULE
unicoap_dtls

```
static int handle_color_request(  
    unicoap_message_t* message, const unicoap_aux_t* aux,  
    unicoap_request_context_t* context, void* arg) {  
  
    unicoap_response_init_string(&message, UNICOAP_STATUS_CONTENT, "rebeccapurple");  
    return unicoap_send_response(&message, context);  
}
```

IMPLEMENTATION

Server API

```
UNICOAP_RESOURCE(my_resource) {  
    .path = "/led/color",  
    .handler = handle_color_request,  
    .methods = UNICOAP_METHODS(UNICOAP_METHOD_GET, UNICOAP_METHOD_PUT),  
    .protocols = UNICOAP_PROTOCOLS(UNICOAP_PROTO_DTLS),  
    .flags = UNICOAP_RESOURCE_FLAG_RELIABLE  
};
```

```
static int handle_color_request(  
    uniconap_message_t* message, const uniconap_aux_t* aux,  
    uniconap_request_context_t* context, void* arg) {  
  
    uniconap_response_init_string(&message, UNICOAP_STATUS_CONTENT, "rebeccapurple");  
    return uniconap_send_response(&message, context);  
}
```

IMPLEMENTATION

Server API

```
UNICOAP_RESOURCE(my_resource) {  
    .path = "/led/color",  
    .handler = handle_color_request,  
    .methods = UNICOAP_METHODS(UNICOAP_METHOD_GET, UNICOAP_METHOD_PUT),  
    .protocols = UNICOAP_PROTOCOLS(UNICOAP_PROTO_DTLS),  
    .flags = UNICOAP_RESOURCE_FLAG_RELIABLE | UNICOAP_RESOURCE_FLAG_DEDUPLICATE  
};
```

MODULE
unicoap_deduplication

```
static int handle_color_request(  
    unicoap_message_t* message, const unicoap_aux_t* aux,  
    unicoap_request_context_t* context, void* arg) {  
  
    unicoap_response_init_string(&message, UNICOAP_STATUS_CONTENT, "rebeccapurple");  
    return unicoap_send_response(&message, context);  
}
```

IMPLEMENTATION

Server API

```
UNICOAP_RESOURCE(my_resource) {  
    .path = "/led/color",  
    .handler = handle_color_request,  
    .methods = UNICOAP_METHODS(UNICOAP_METHOD_GET, UNICOAP_METHOD_PUT),  
    .protocols = UNICOAP_PROTOCOLS(UNICOAP_PROTO_DTLS),  
    .flags = UNICOAP_RESOURCE_FLAG_RELIABLE | UNICOAP_RESOURCE_FLAG_DEDUPLICATE  
};
```

```
static int handle_color_request(  
    uniconap_message_t* message, const uniconap_aux_t* aux,  
    uniconap_request_context_t* context, void* arg) {  
  
    uniconap_response_init_string(&message, UNICOAP_STATUS_CONTENT, "rebeccapurple");  
    return uniconap_send_response(&message, context);  
}
```

IMPLEMENTATION

Server API

```
UNICOAP_RESOURCE(my_resource) {  
    .path = "/led/color",  
    .handler = handle_color_request,  
    .methods = UNICOAP_METHODS(UNICOAP_METHOD_GET, UNICOAP_METHOD_PUT),  
    .protocols = UNICOAP_PROTOCOLS(UNICOAP_PROTO_DTLS),  
    .flags = UNICOAP_RESOURCE_FLAG_RELIABLE | UNICOAP_RESOURCE_FLAG_DEDUPLICATE  
};
```

```
static int handle_color_request(  
    uniconap_message_t* message, const uniconap_aux_t* aux,  
    uniconap_request_context_t* context, void* arg) {  
  
    uniconap_response_init_string(&message, UNICOAP_STATUS_CONTENT, "rebeccapurple");  
    return uniconap_send_response(&message, context);  
}
```

IMPLEMENTATION

Server API

```
UNICOAP_RESOURCE(my_resource) {  
    .path = "/led/color",  
    .handler = handle_color_request,  
    .methods = UNICOAP_METHODS(UNICOAP_METHOD_GET, UNICOAP_METHOD_PUT),  
    .protocols = UNICOAP_PROTOCOLS(UNICOAP_PROTO_DTLS),  
    .flags = UNICOAP_RESOURCE_FLAG_RELIABLE | UNICOAP_RESOURCE_FLAG_DEDUPLICATE  
};
```

```
static int handle_color_request(  
    uniconap_message_t* message, const uniconap_aux_t* aux,  
    uniconap_request_context_t* context, void* arg) {  
  
    uniconap_response_init_string(&message, UNICOAP_STATUS_CONTENT, "rebeccapurple");  
    return uniconap_send_response(&message, context);  
}
```

Evaluation

EVALUATION

Measurement Setup

Measurement Setup

Platform: nRF52840, Cortex-M4 MCU, 256 KiB RAM, 1 MiB RAM

Measurement Setup

Platform: nRF52840, Cortex-M4 MCU, 256 KiB RAM, 1 MiB RAM

Networking: GNRC, 6LoWPAN, 802.15.4

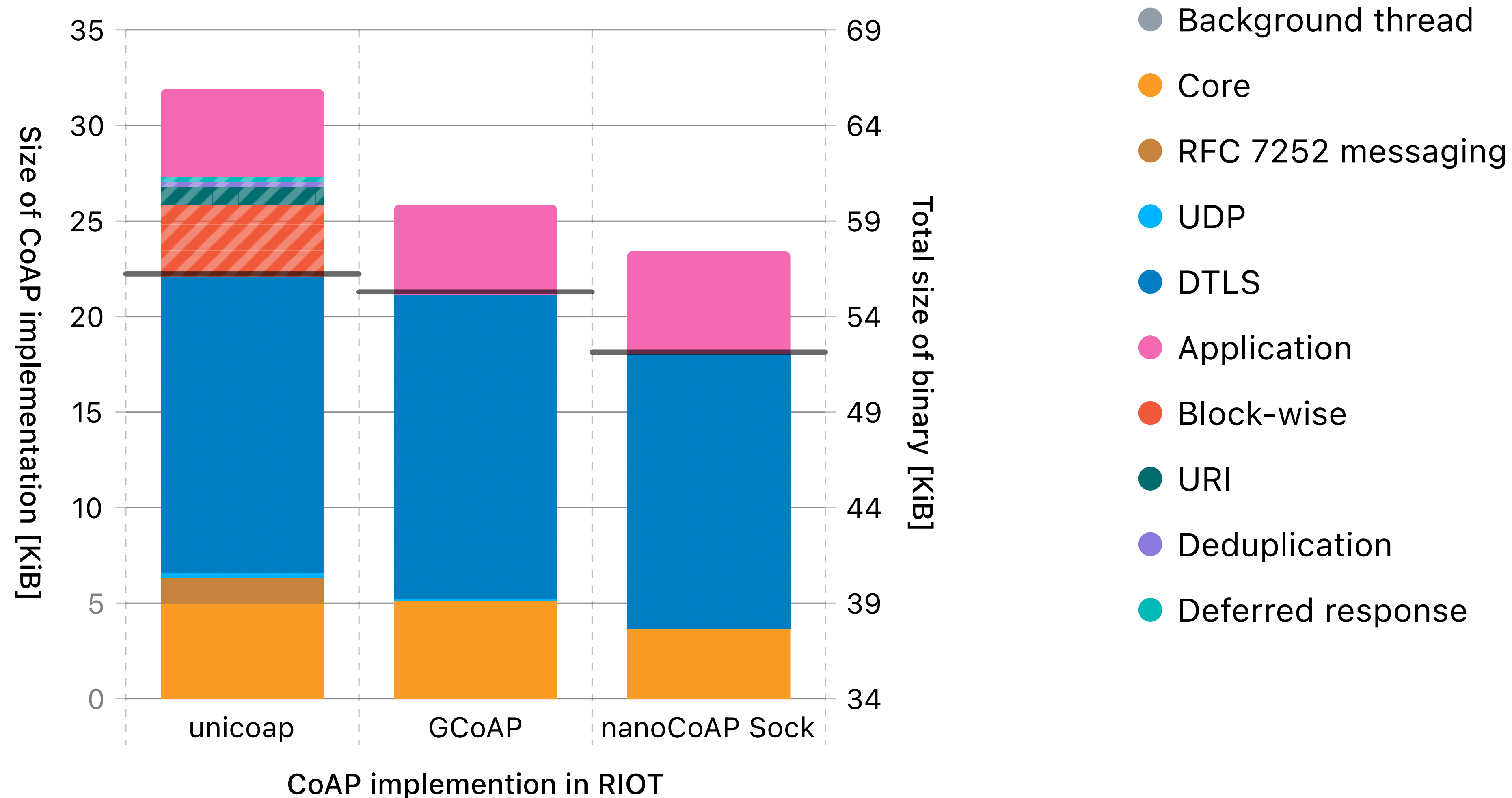
Measurement Setup

Platform: nRF52840, Cortex-M4 MCU, 256 KiB RAM, 1 MiB RAM

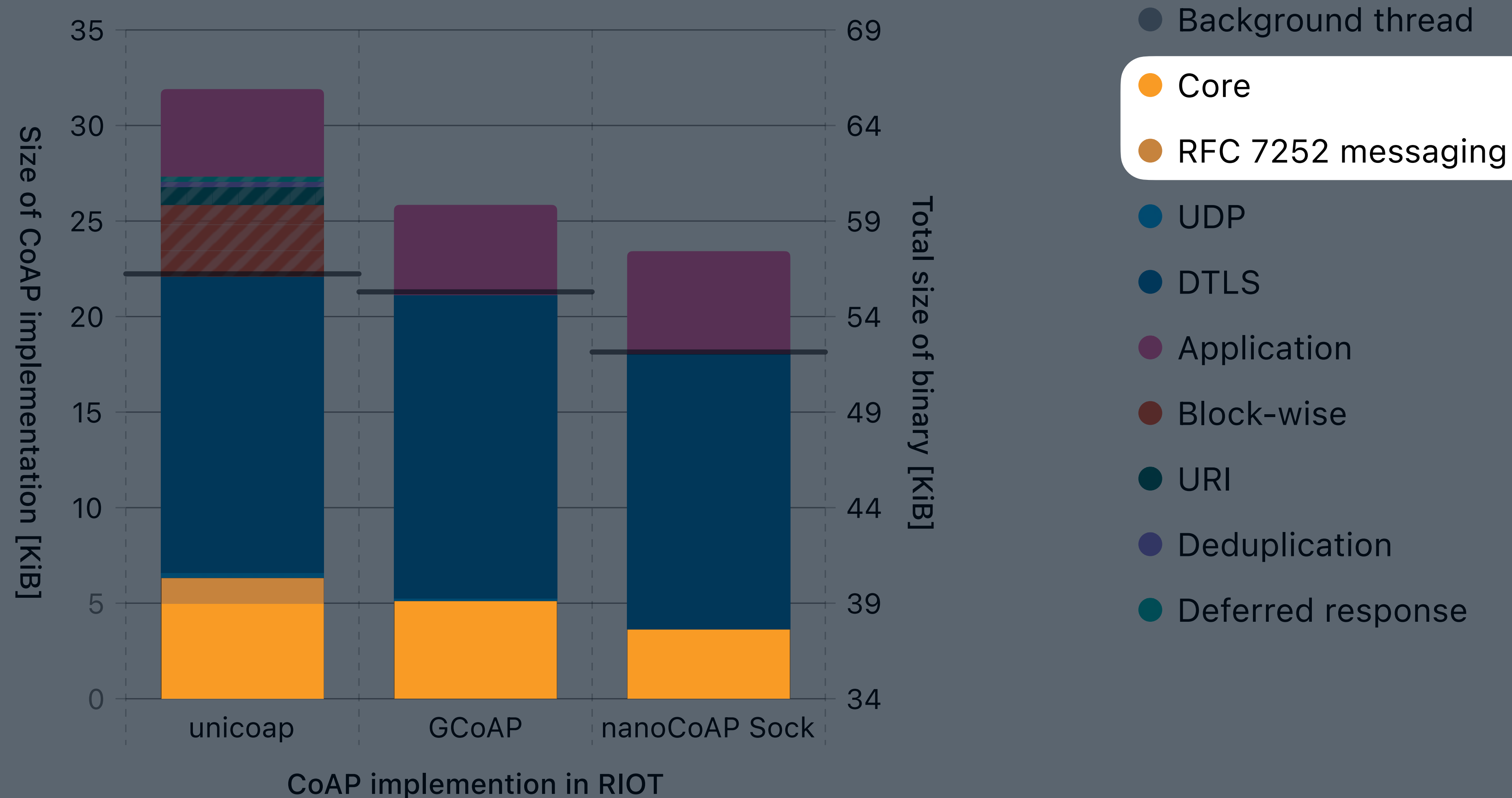
Networking: GNRC, 6LoWPAN, 802.15.4

Optimizations: -O3 optimization level, DEVELOPMENT=0, NDEBUG

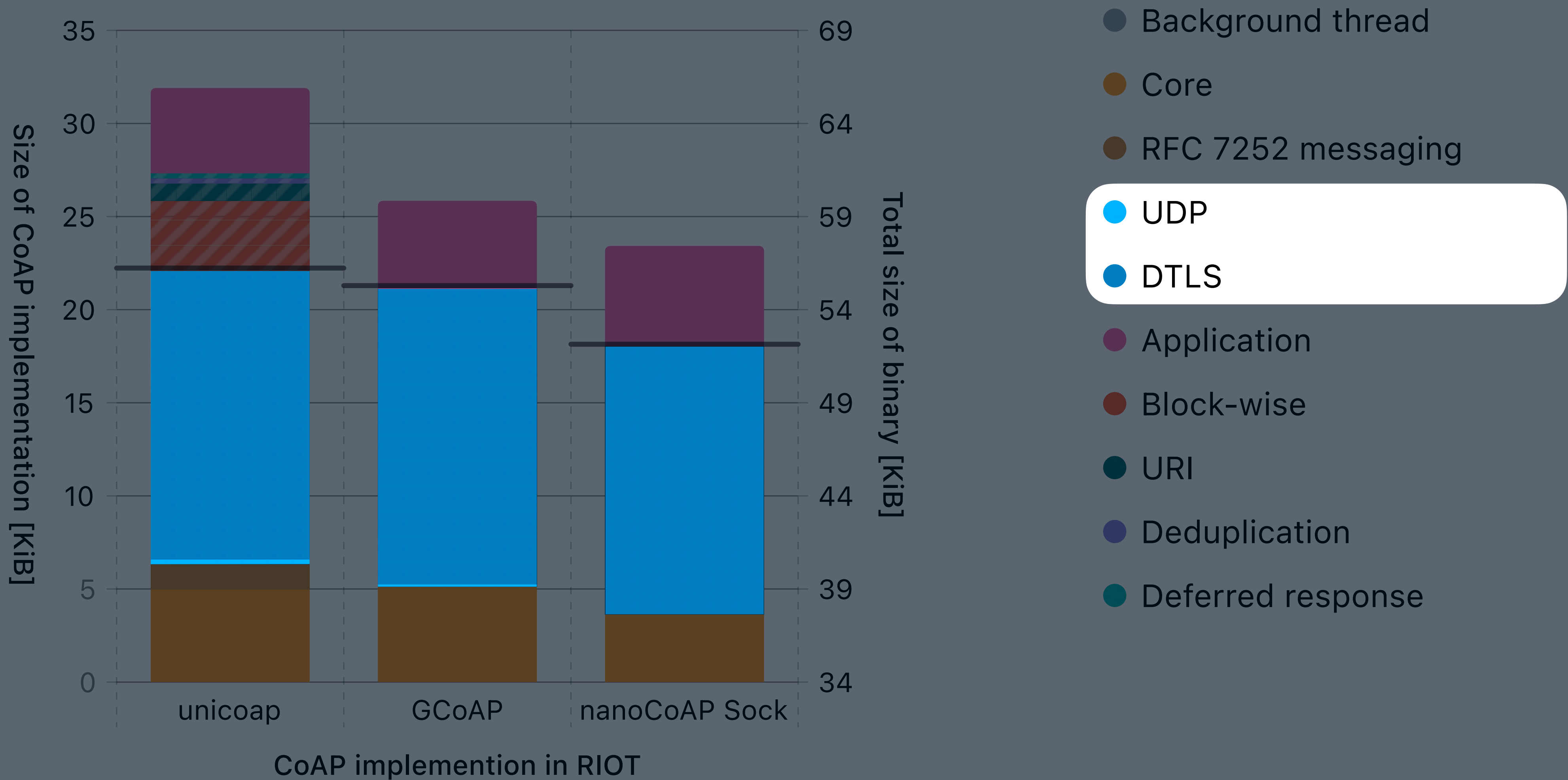
Build Sizes (ROM)



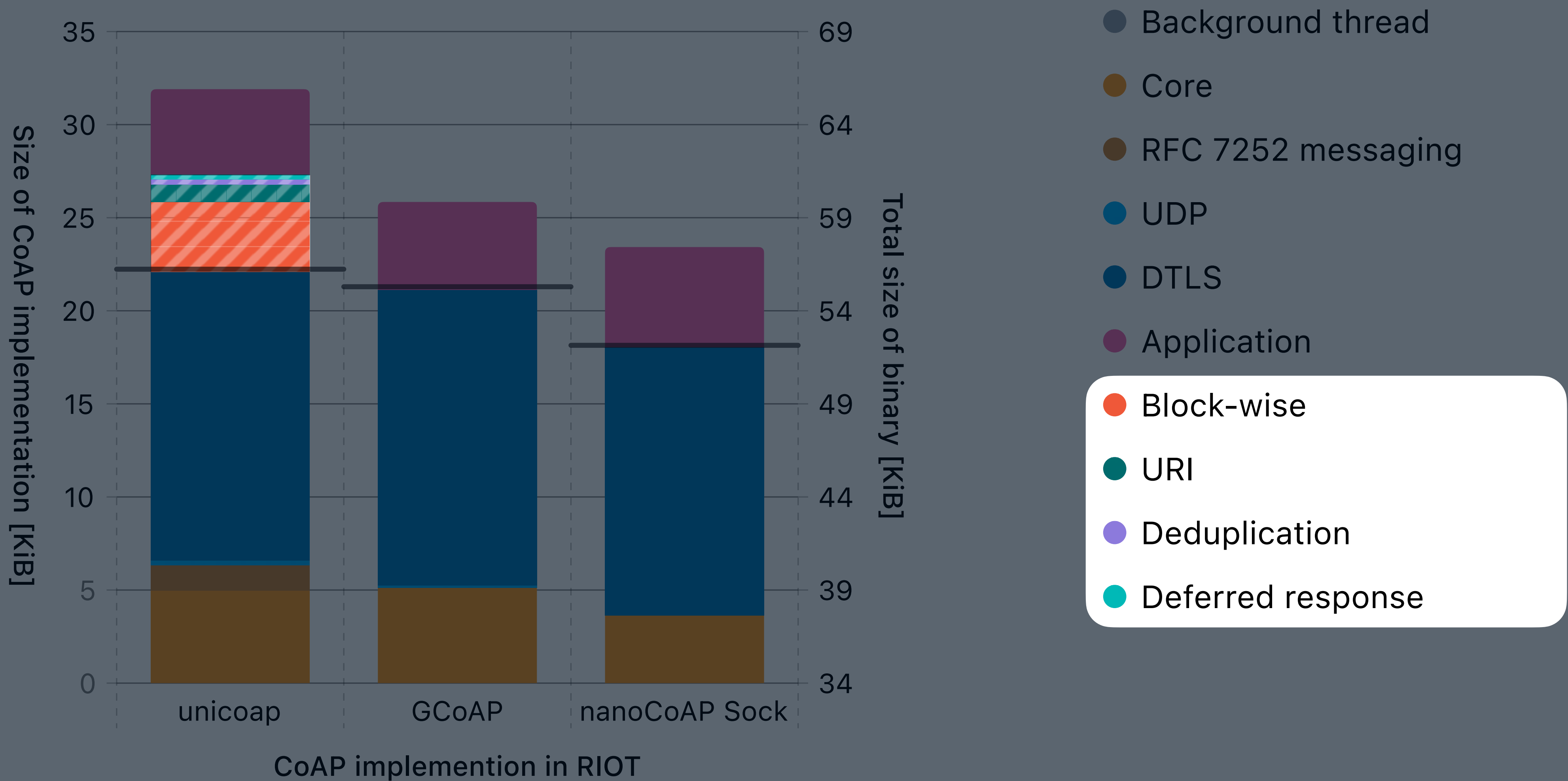
Build Sizes (ROM)



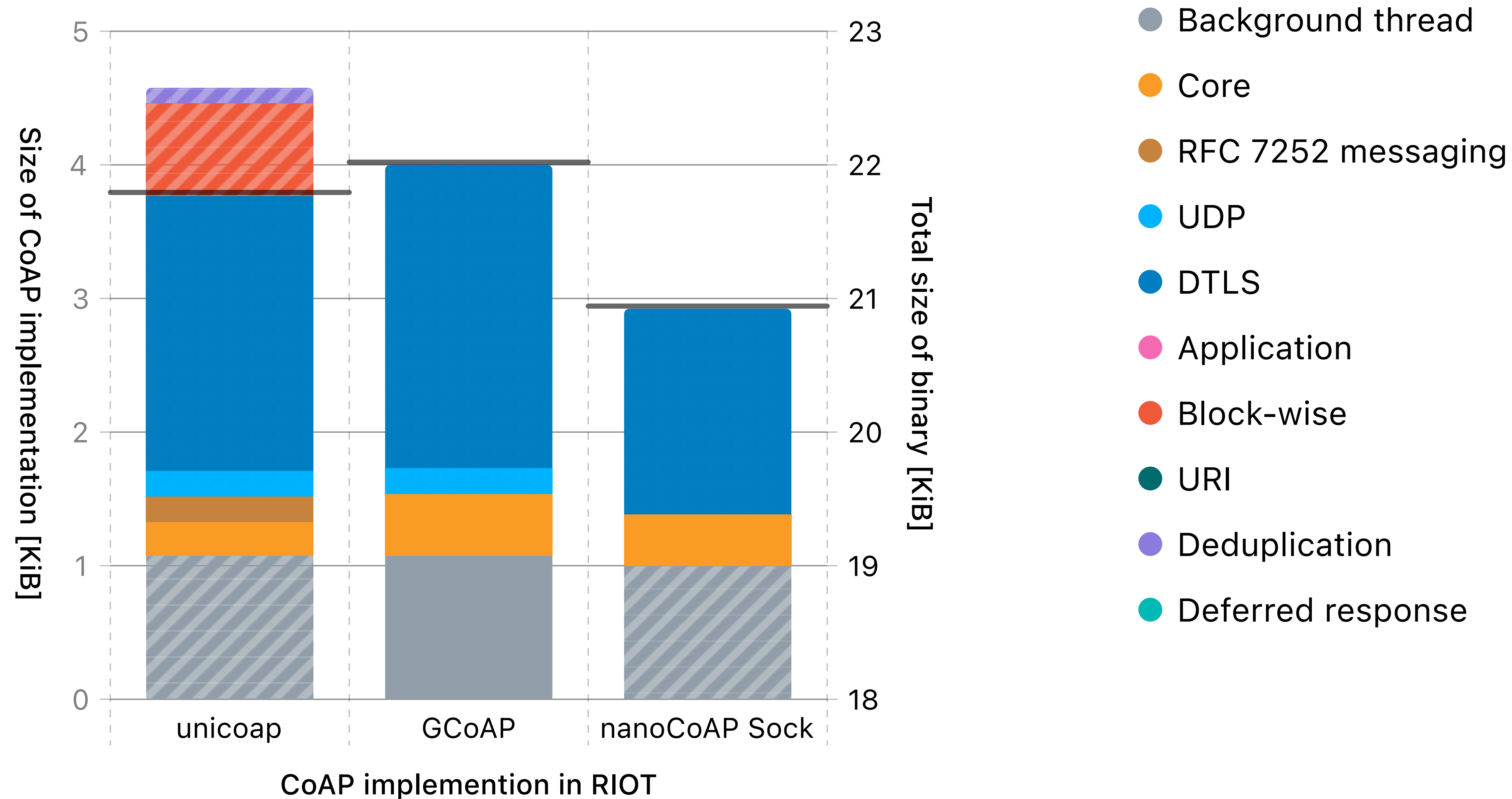
Build Sizes (ROM)



Build Sizes (ROM)

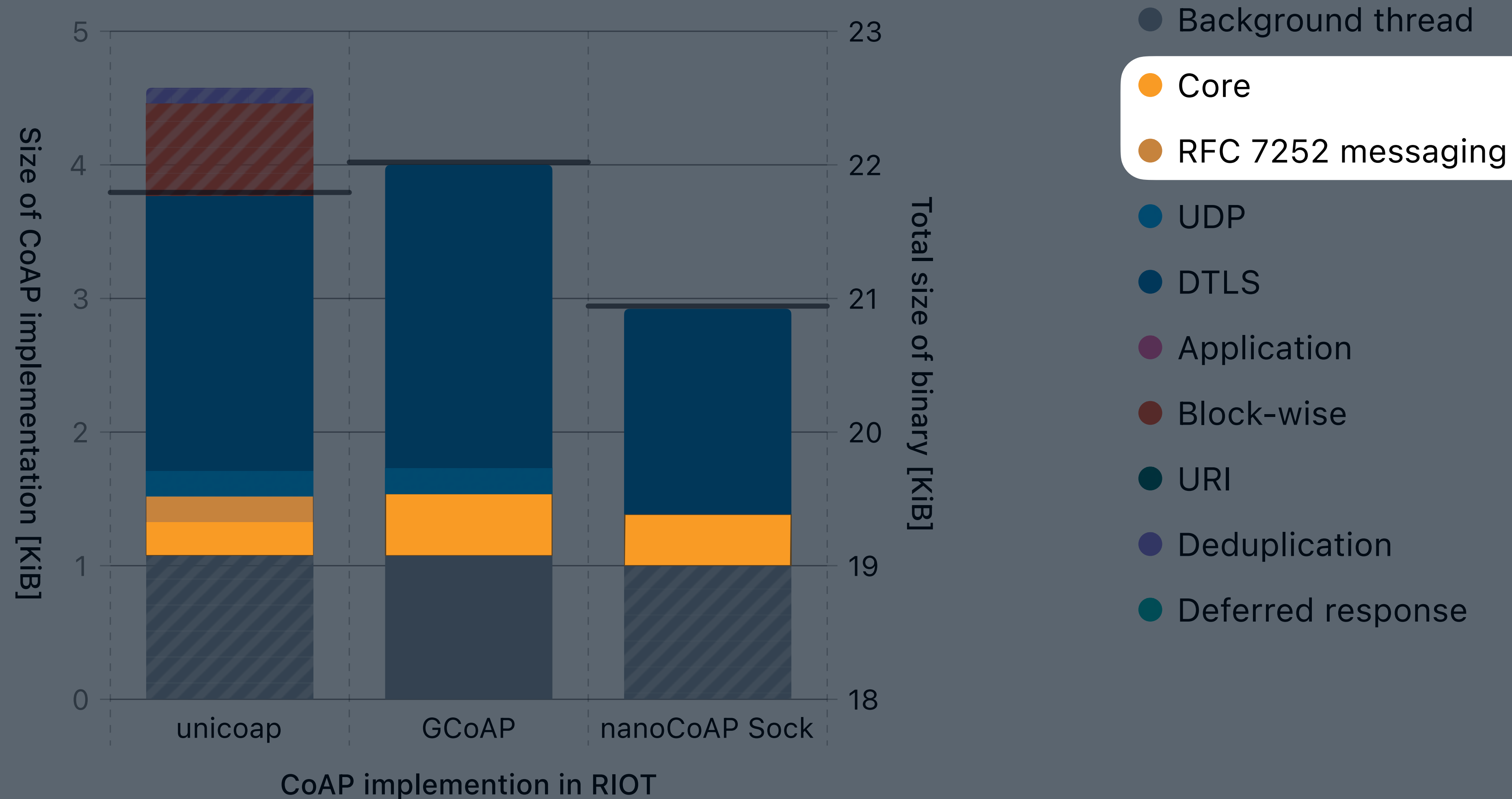


Build Sizes (RAM)

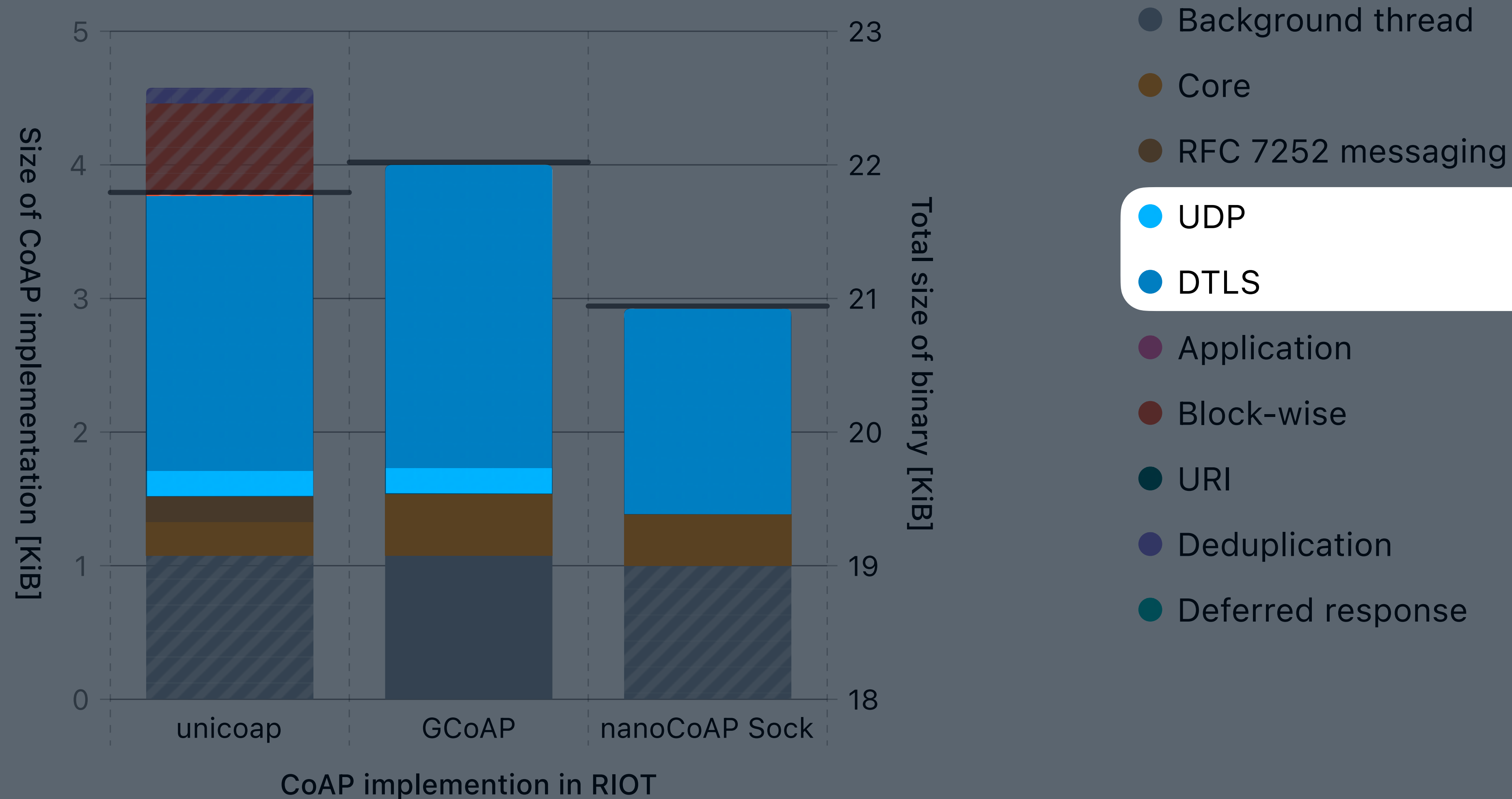


EVALUATION

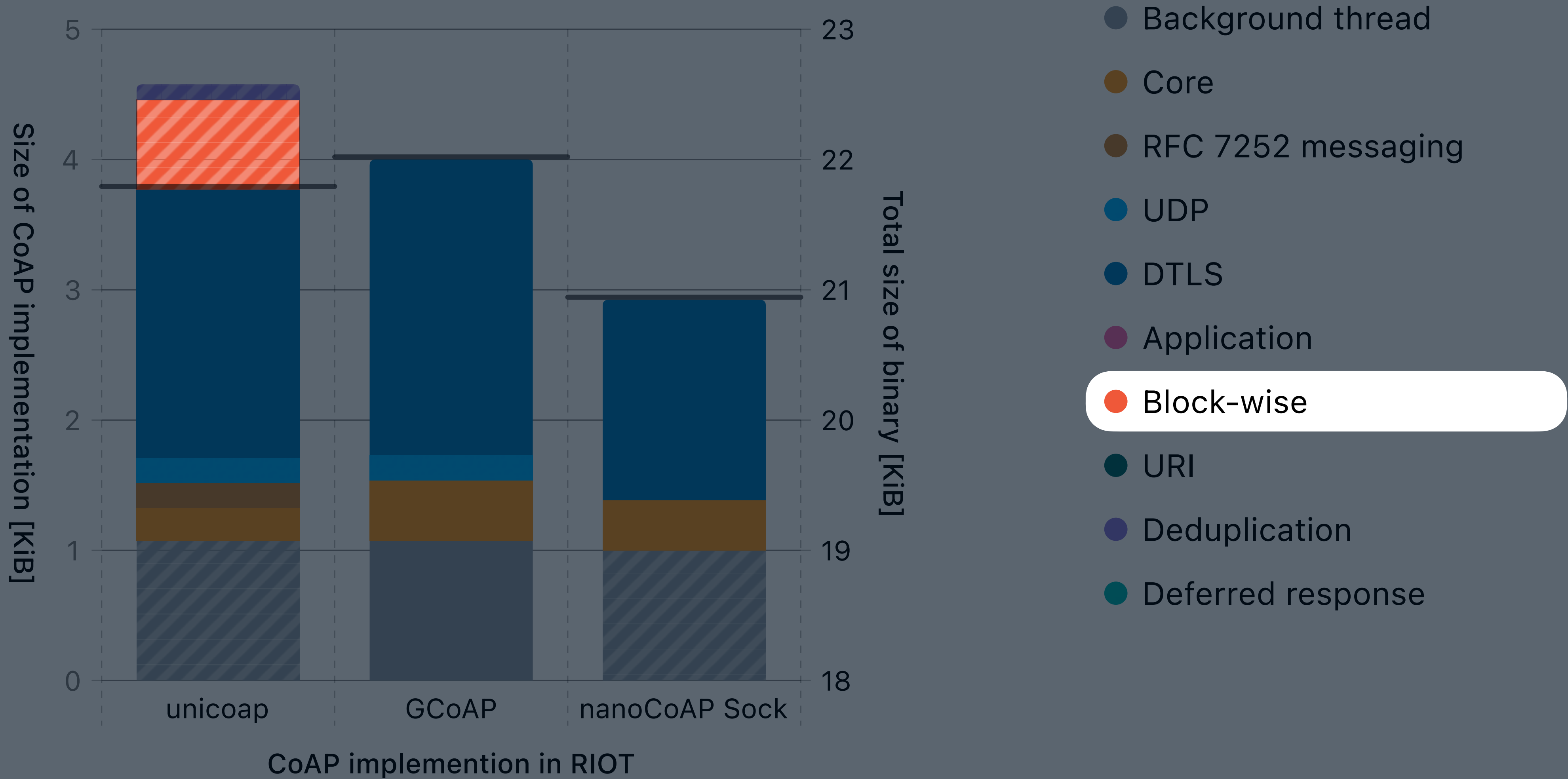
Build Sizes (RAM)



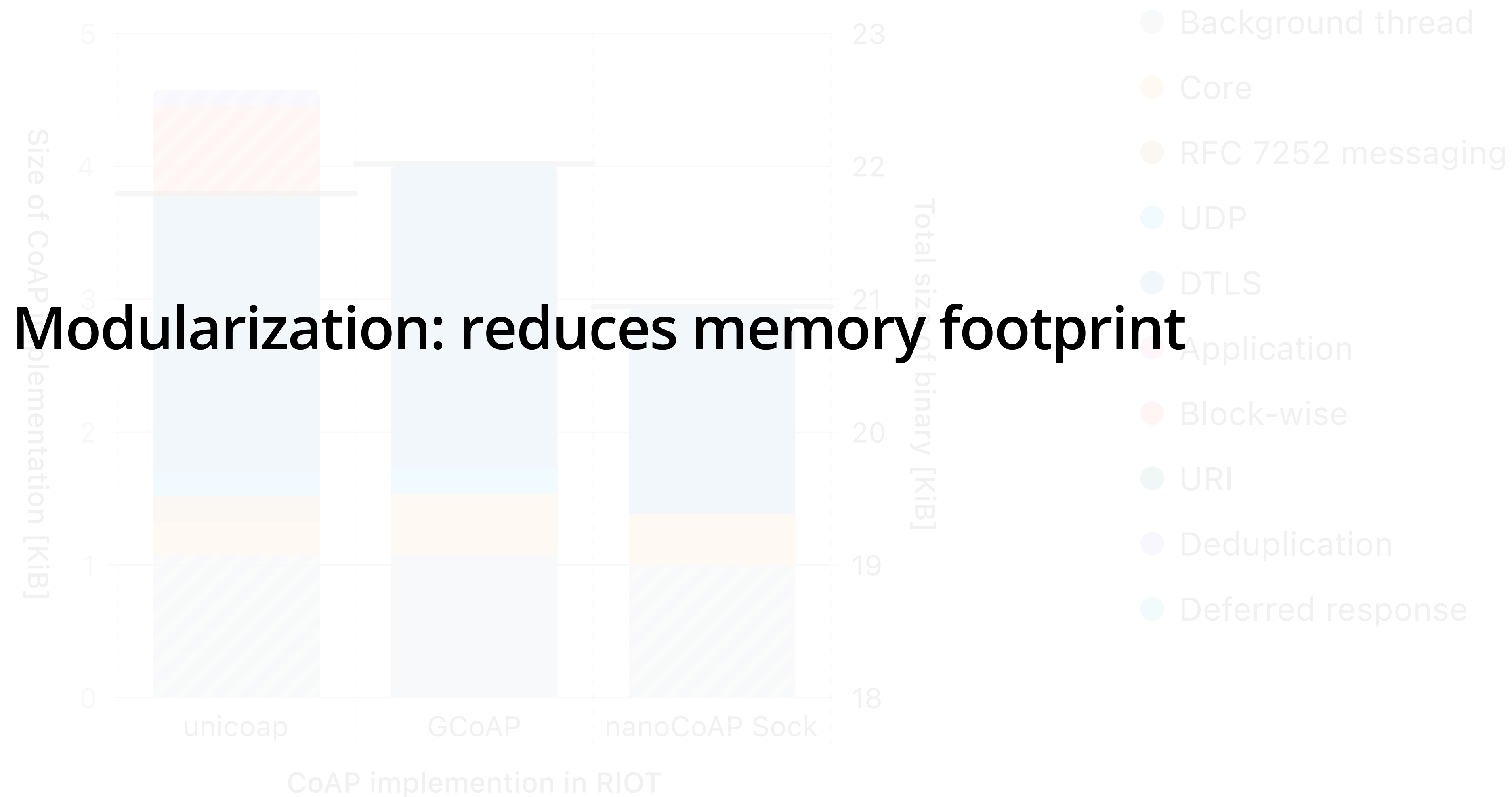
Build Sizes (RAM)



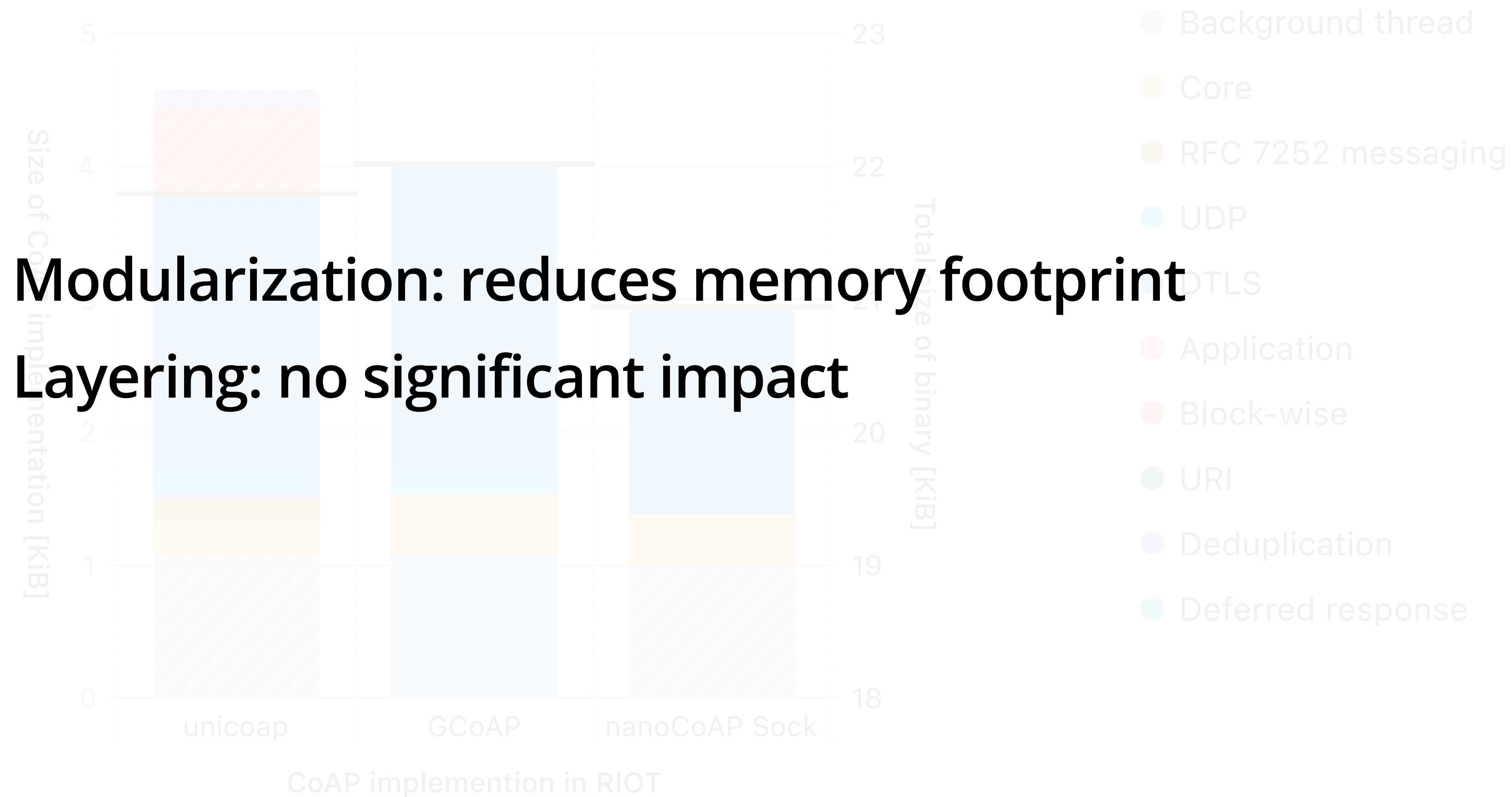
Build Sizes (RAM)



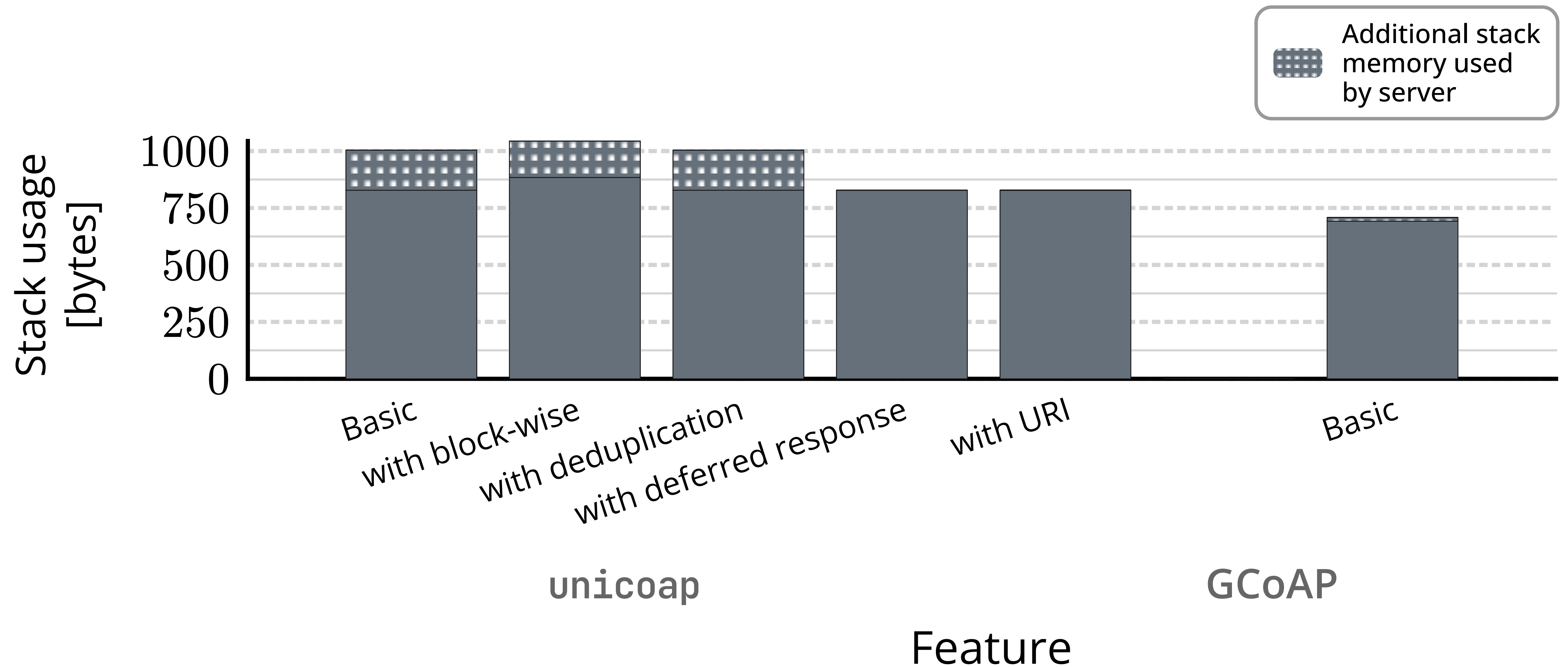
Build Sizes (RAM)



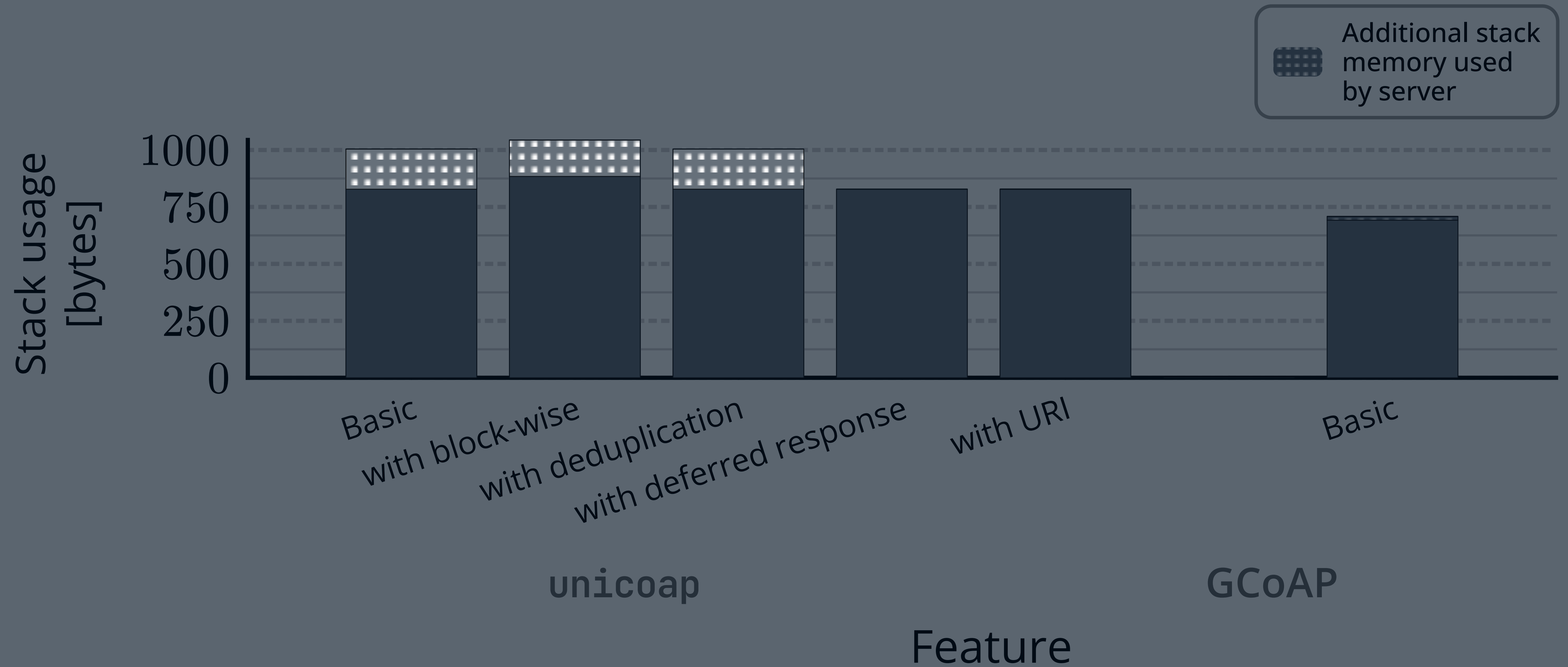
Build Sizes (RAM)



Stack Usage



Stack Usage



Stack Usage

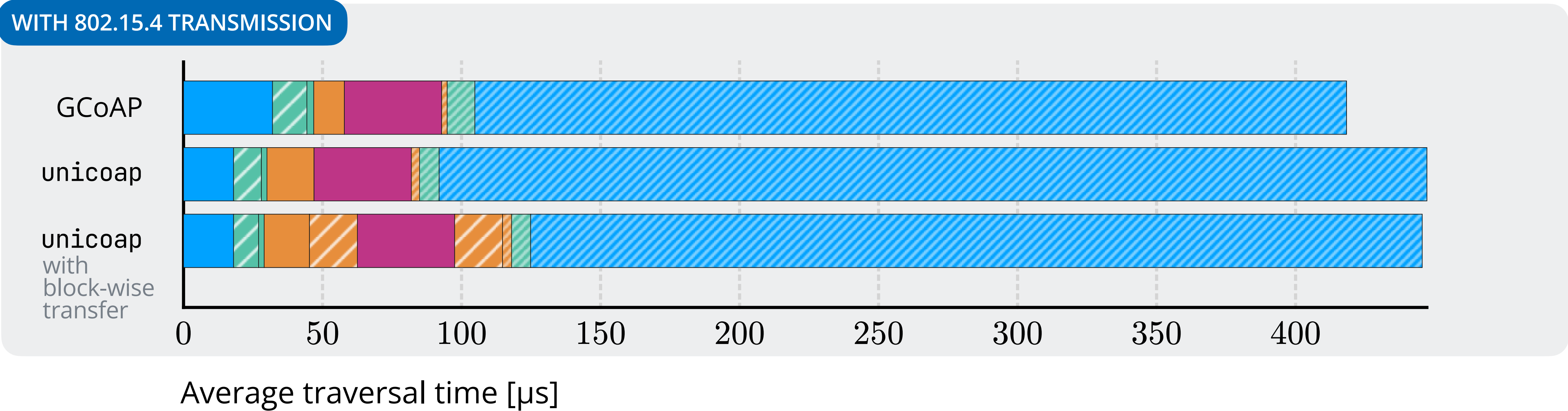
Layering: small effect
Advanced features: minimal impact



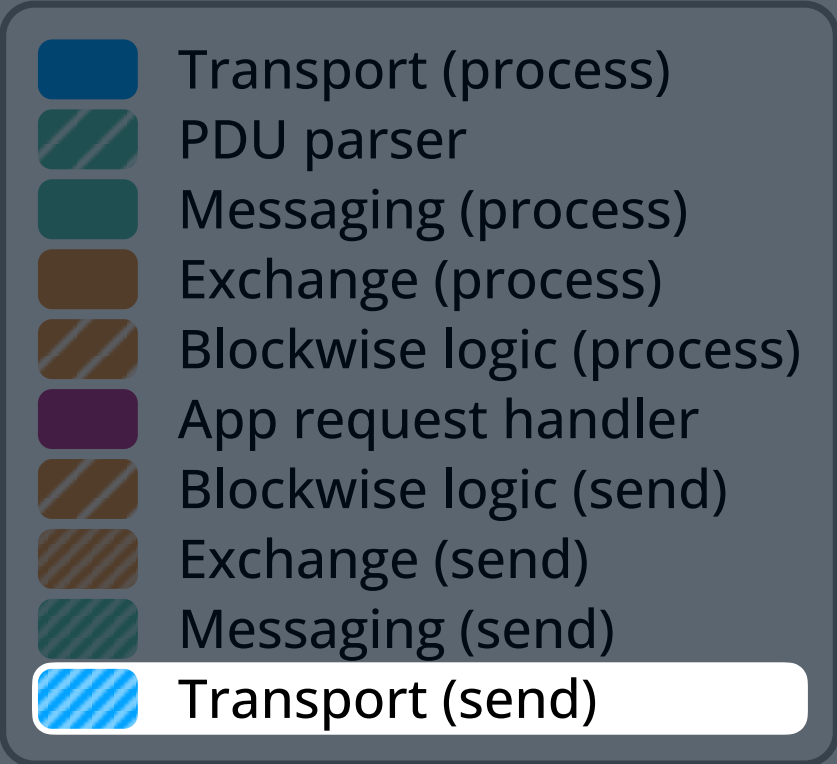
Traversal Times

- Transport (process)
- PDU parser
- Messaging (process)
- Exchange (process)
- Blockwise logic (process)
- App request handler
- Blockwise logic (send)
- Exchange (send)
- Messaging (send)
- Transport (send)

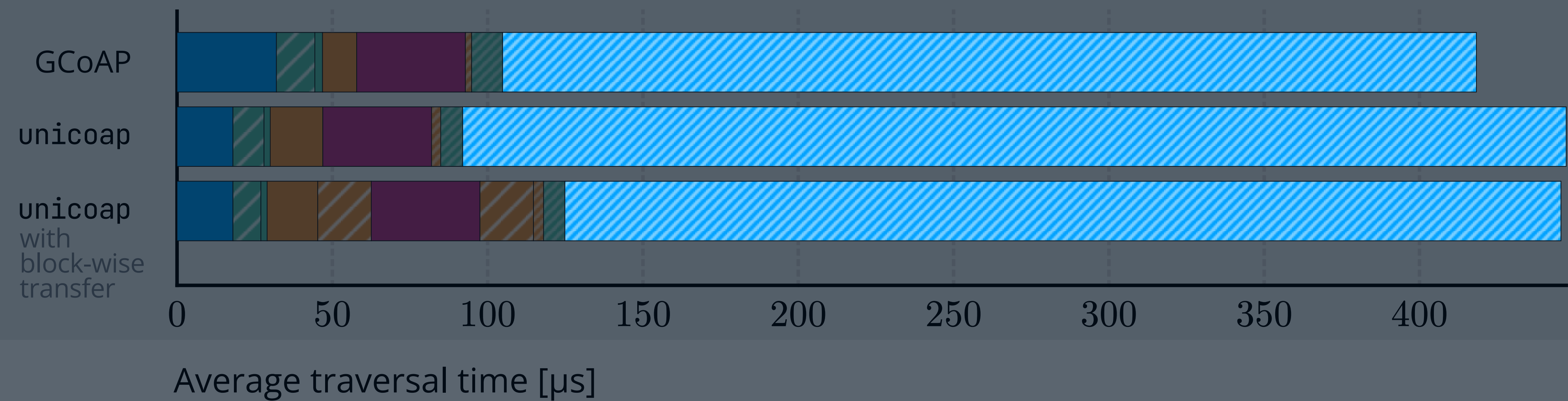
WITH 802.15.4 TRANSMISSION



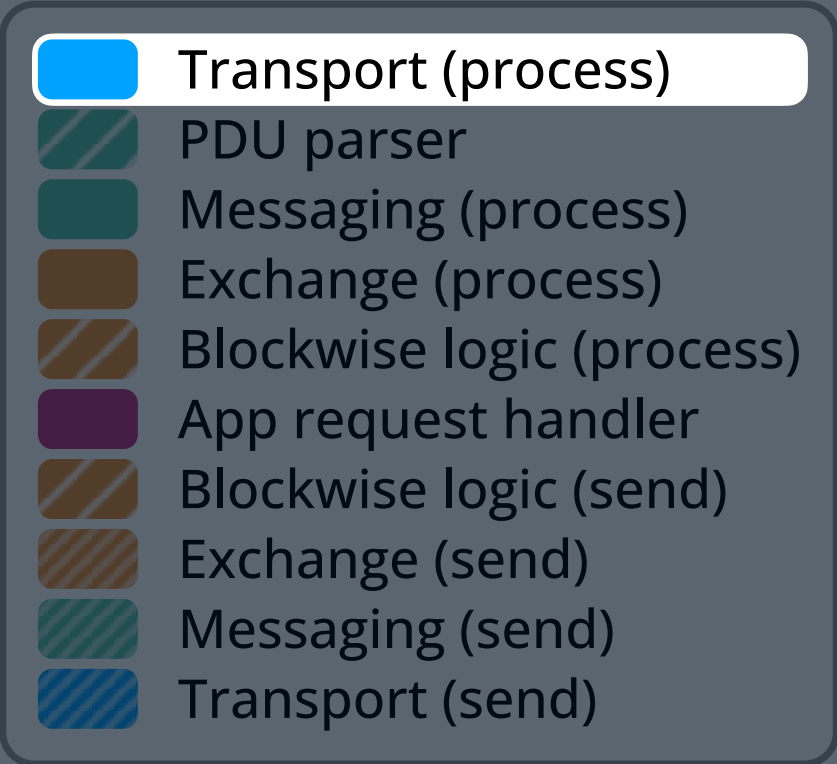
Traversal Times



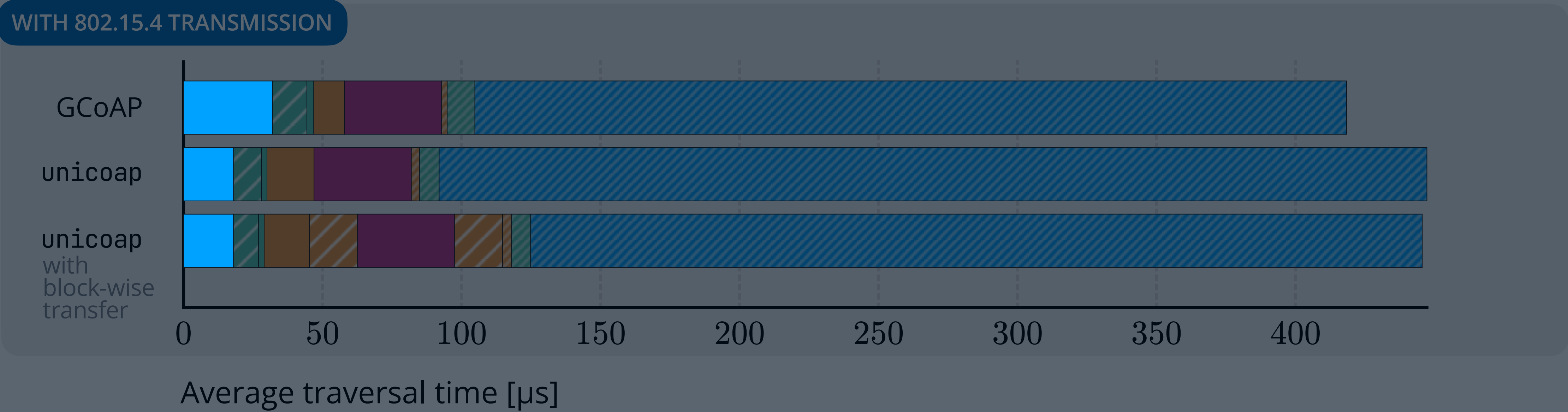
WITH 802.15.4 TRANSMISSION



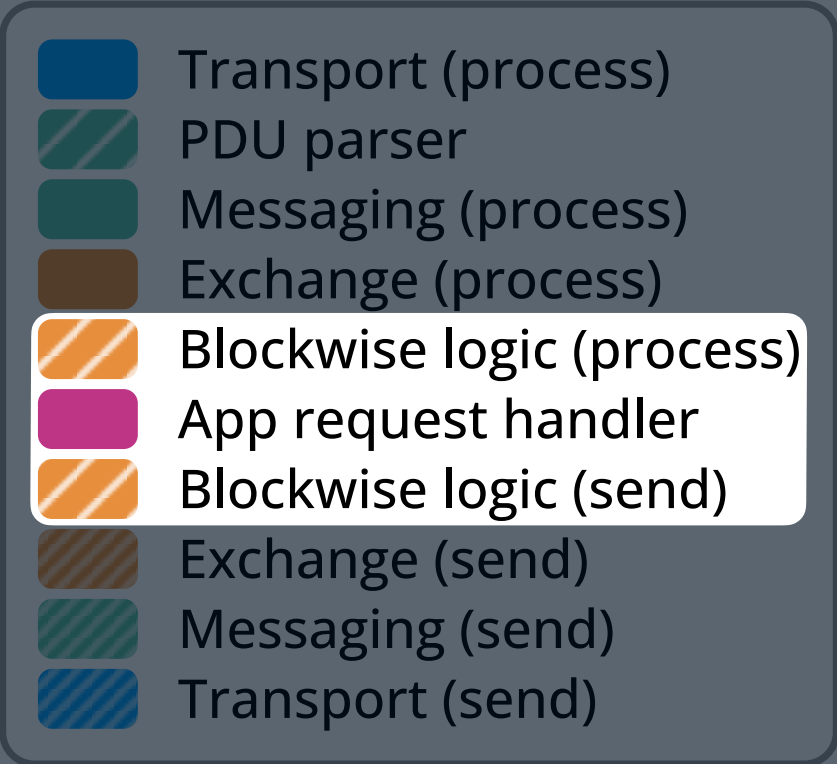
Traversal Times



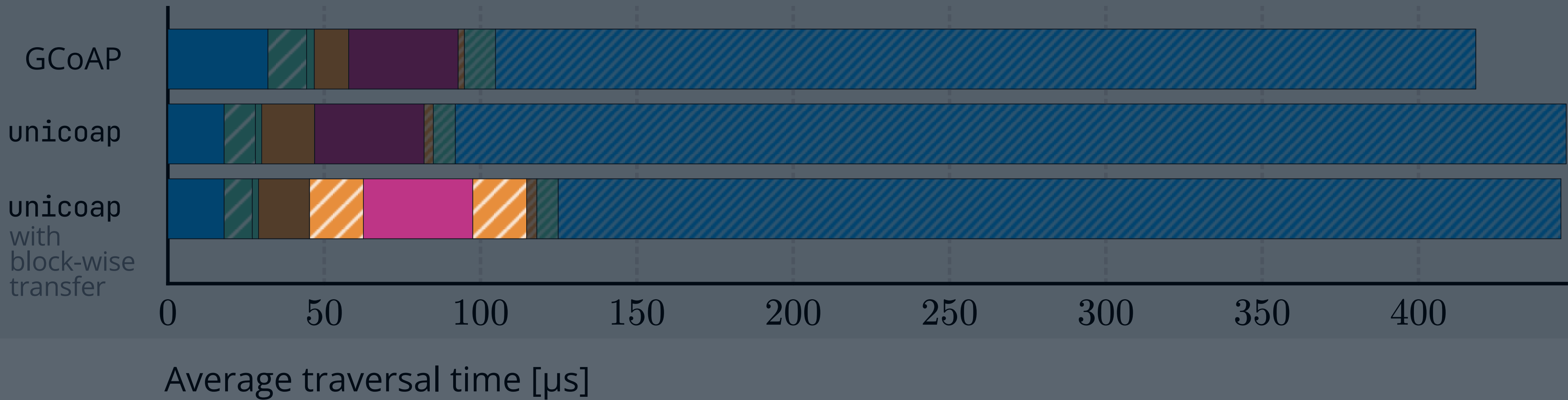
WITH 802.15.4 TRANSMISSION



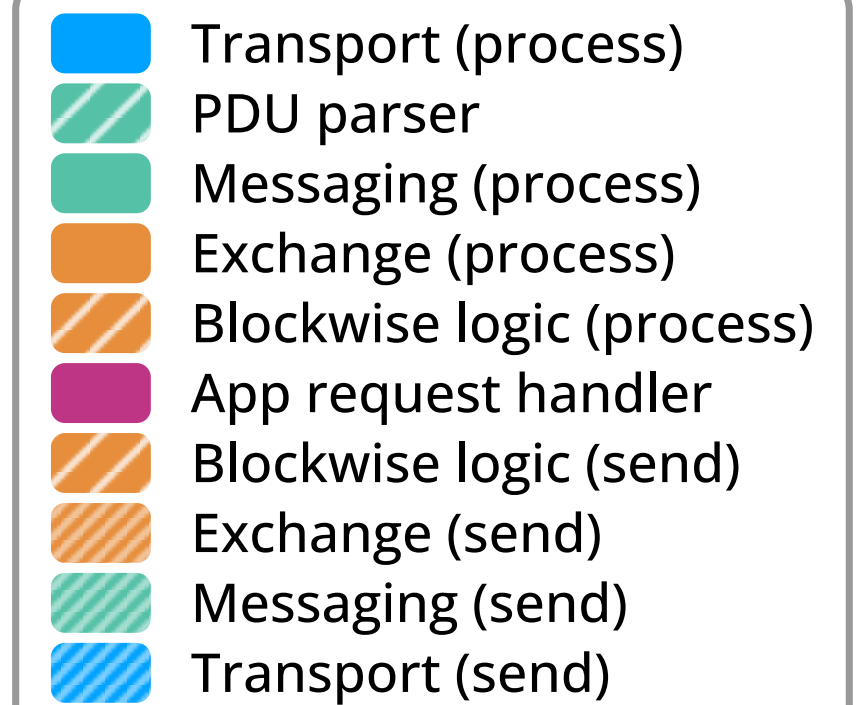
Traversal Times



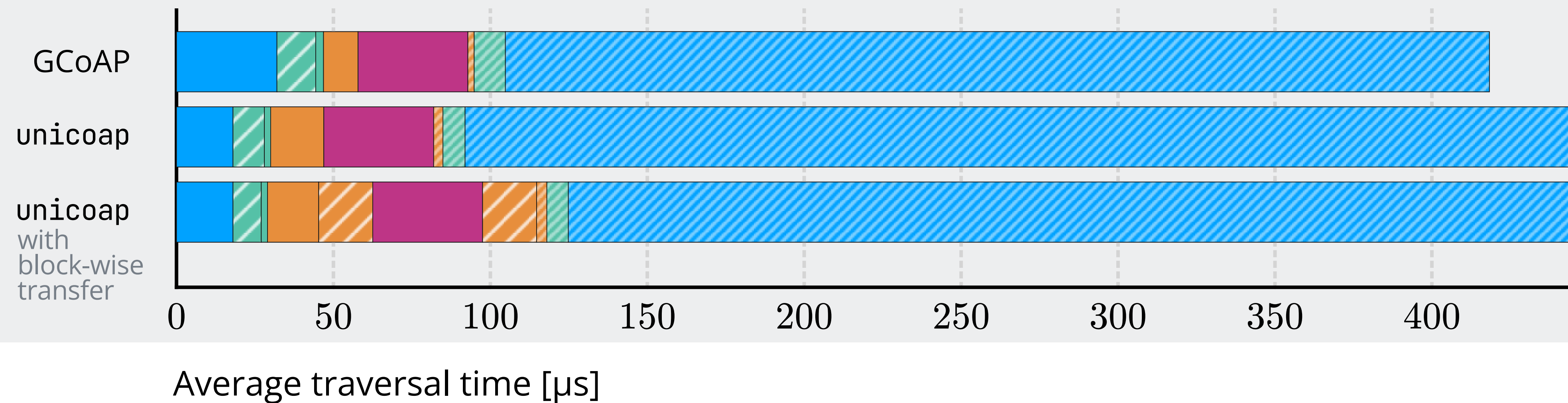
WITH 802.15.4 TRANSMISSION



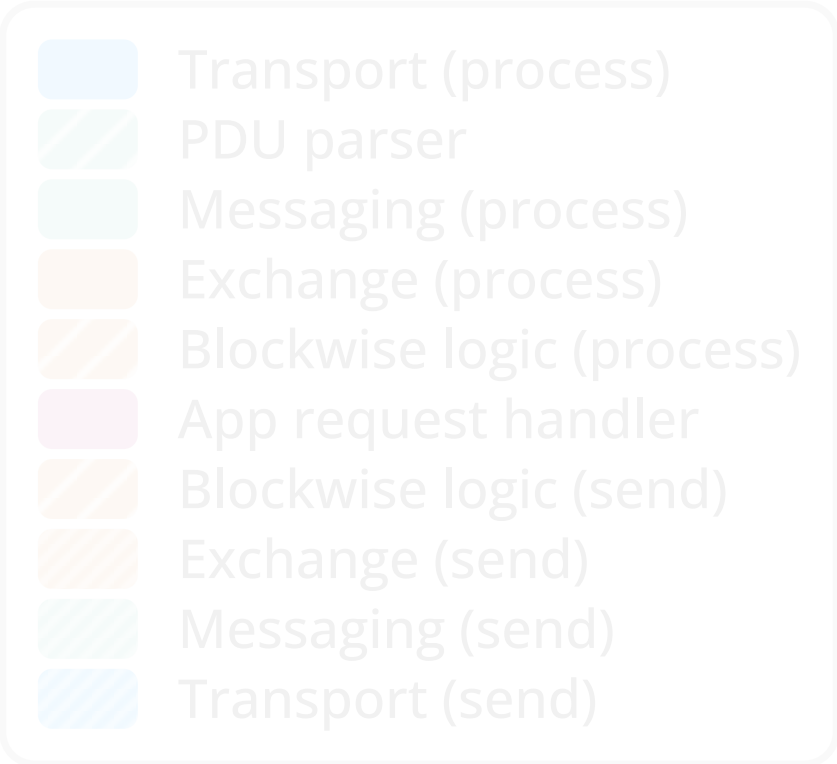
Traversal Times



WITH 802.15.4 TRANSMISSION



Traversal Times

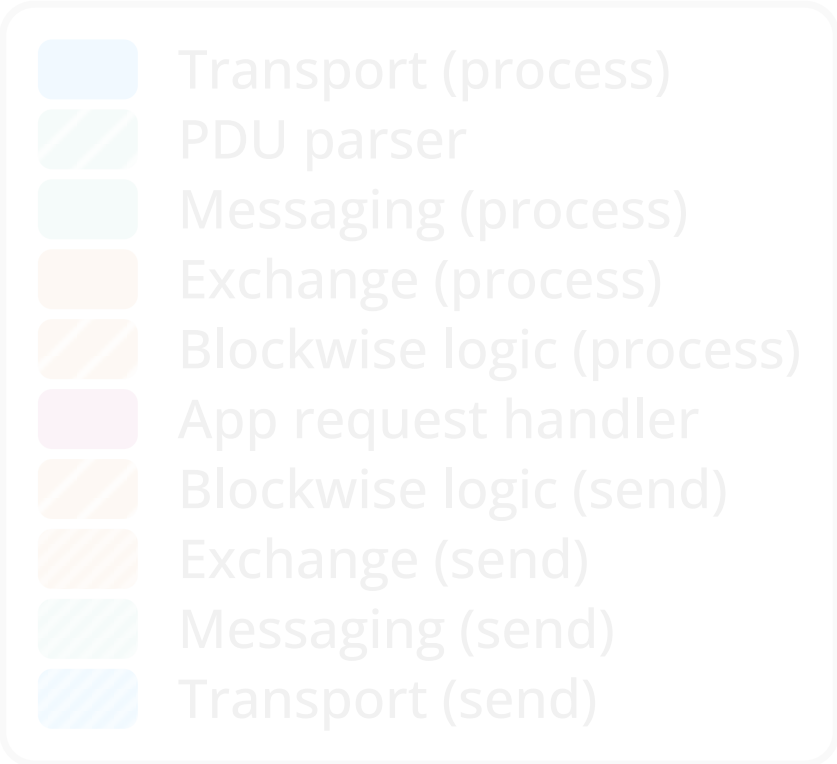


WITH 802.15.4 TRANSMISSION

Similar performance compared to GCoAP

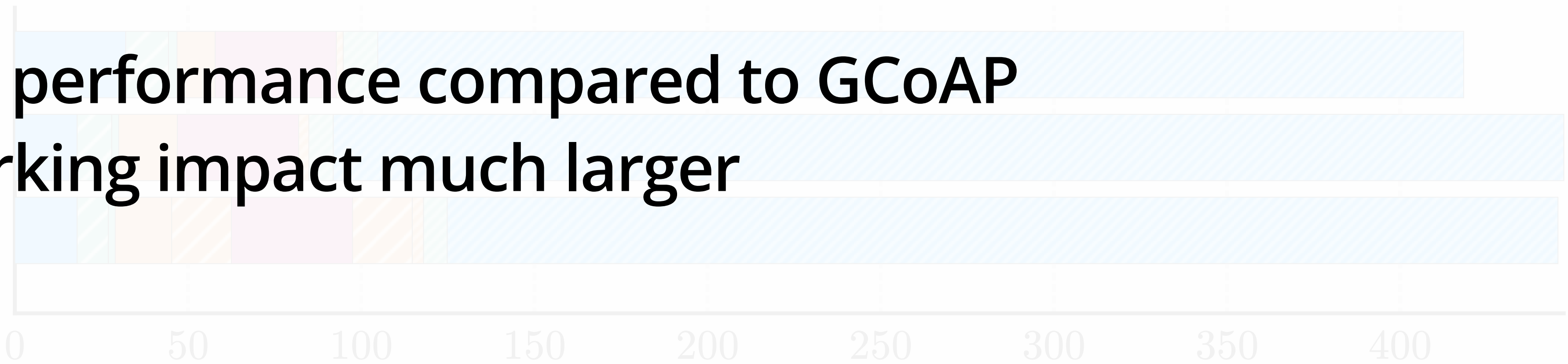


Traversal Times



WITH 802.15.4 TRANSMISSION

Similar performance compared to GCoAP
Networking impact much larger



Conclusion

Conclusion and Outlook

Conclusion and Outlook

`unicoap`: unified and modular CoAP suite for the constrained IoT

Conclusion and Outlook

unicoap: unified and modular CoAP suite for the constrained IoT

✔ Layered architecture enables **transport flexibility**, is feasible in the constrained IoT

Conclusion and Outlook

unicoap: unified and modular CoAP suite for the constrained IoT

- ✓ Layered architecture enables **transport flexibility**, is feasible in the constrained IoT
- ✓ Modularization reduces memory footprint, enables **advanced features**

Conclusion and Outlook

unicoap: unified and modular CoAP suite for the constrained IoT

- ✓ Layered architecture enables **transport flexibility**, is feasible in the constrained IoT
- ✓ Modularization reduces memory footprint, enables **advanced features**
- ✓ High-level, flexible API **reduces boilerplate**

Conclusion and Outlook

unicoap: unified and modular CoAP suite for the constrained IoT

- ✓ Layered architecture enables **transport flexibility**, is feasible in the constrained IoT
- ✓ Modularization reduces memory footprint, enables **advanced features**
- ✓ High-level, flexible API **reduces boilerplate**
- ✓ Implementation in  RIOT unifies **nanoCoAP**, **nanoCoAP Sock**, and **GCoAP**

Conclusion and Outlook

unicoap: unified and modular CoAP suite for the constrained IoT

- ✓ Layered architecture enables **transport flexibility**, is feasible in the constrained IoT
- ✓ Modularization reduces memory footprint, enables **advanced features**
- ✓ High-level, flexible API **reduces boilerplate**
- ✓ Implementation in  RIOT unifies nanoCoAP, nanoCoAP Sock, and GCoAP

FUTURE WORK

Conclusion and Outlook

unicoap: unified and modular CoAP suite for the constrained IoT

- ✓ Layered architecture enables **transport flexibility**, is feasible in the constrained IoT
- ✓ Modularization reduces memory footprint, enables **advanced features**
- ✓ High-level, flexible API **reduces boilerplate**
- ✓ Implementation in  RIOT unifies nanoCoAP, nanoCoAP Sock, and GCoAP

FUTURE WORK

- OSCORE support

Conclusion and Outlook

unicoap: unified and modular CoAP suite for the constrained IoT

- ✓ Layered architecture enables **transport flexibility**, is feasible in the constrained IoT
- ✓ Modularization reduces memory footprint, enables **advanced features**
- ✓ High-level, flexible API **reduces boilerplate**
- ✓ Implementation in  RIOT unifies nanoCoAP, nanoCoAP Sock, and GCoAP

FUTURE WORK

- OSCORE support
- Drivers

Conclusion and Outlook

unicoap: unified and modular CoAP suite for the constrained IoT

- ✓ Layered architecture enables **transport flexibility**, is feasible in the constrained IoT
- ✓ Modularization reduces memory footprint, enables **advanced features**
- ✓ High-level, flexible API **reduces boilerplate**
- ✓ Implementation in  RIOT unifies nanoCoAP, nanoCoAP Sock, and GCoAP

FUTURE WORK

- OSCORE support
- Drivers
- Integration of CoAP applications

unicoap Pull Requests

<https://github.com/RIOT-OS/RIOT/issues/21389>



Please help
reviewing