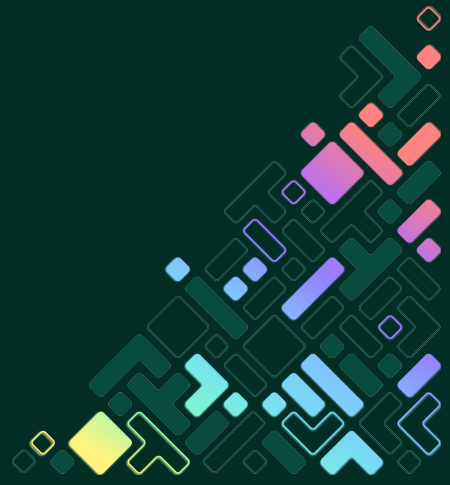
Zephyr® Project

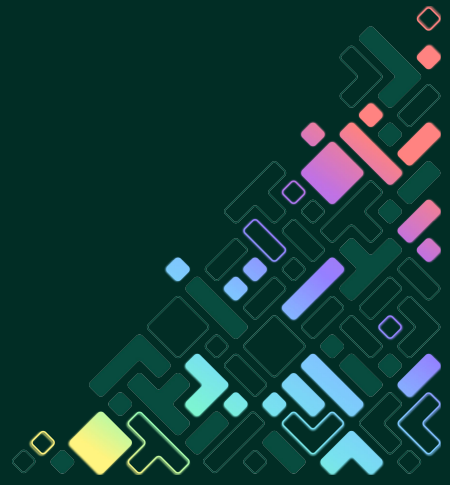
Developer Summit



Zephyr® Project
Developer Summit

ZBUS - NEW FEATURES AND ROADMAP

Rodrigo Peixoto - Edge-UFAL/Citrinio: rodrigopex@gmail.com



MOTIVATION: ONE-TO-ONE

FIFO ✅

LIFO ✅

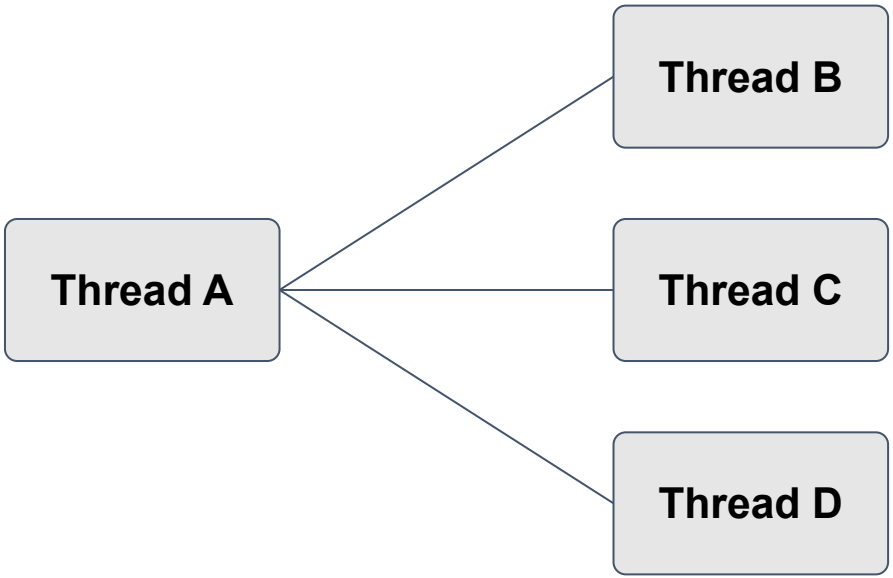
Stack ✅

Message queue ✅

Mailbox ✅

Pipe ✅

MOTIVATION: ONE-TO-MANY



FIFO ❌

LIFO ❌

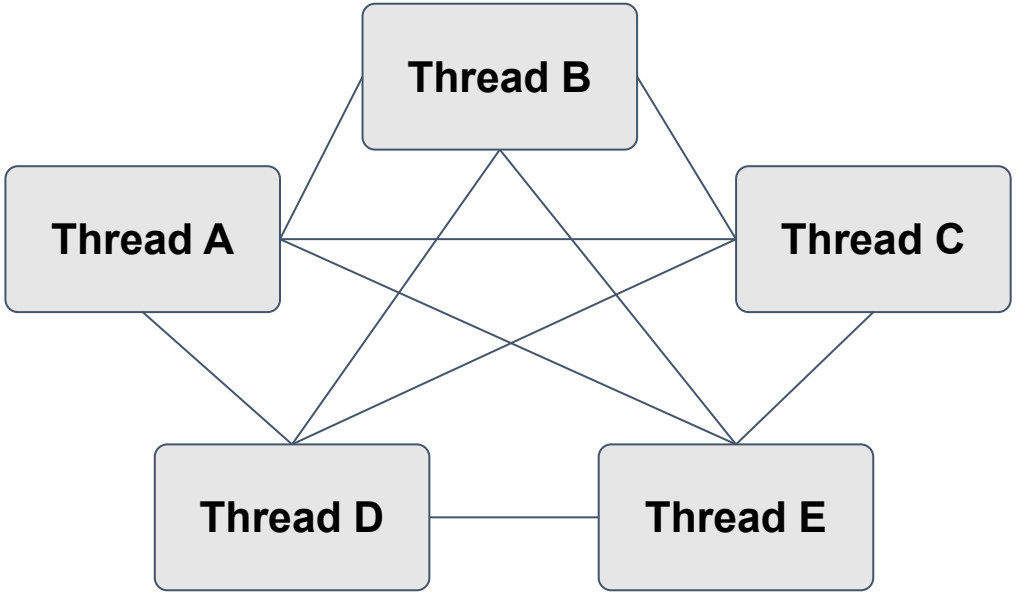
Stack ❌

Message queue ❌

Mailbox ➖

Pipe ❌

MOTIVATION: MANY-TO-MANY



FIFO ❌

LIFO ❌

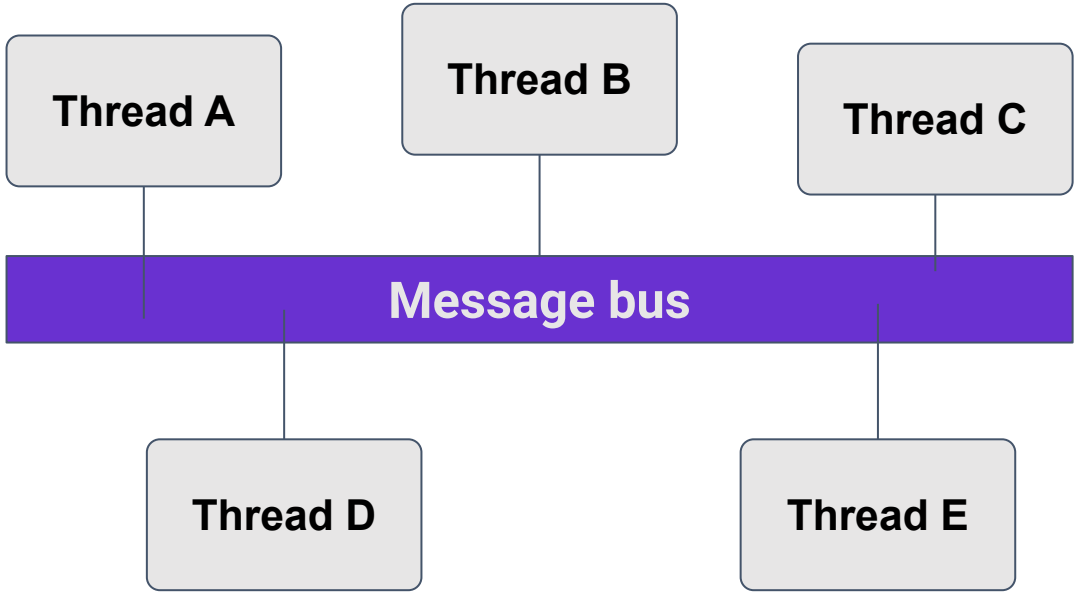
Stack ❌

Message queue ❌

Mailbox ❌

Pipe ❌

SOLUTION: THE BUS



FIFO ❌

LIFO ❌

Stack ❌

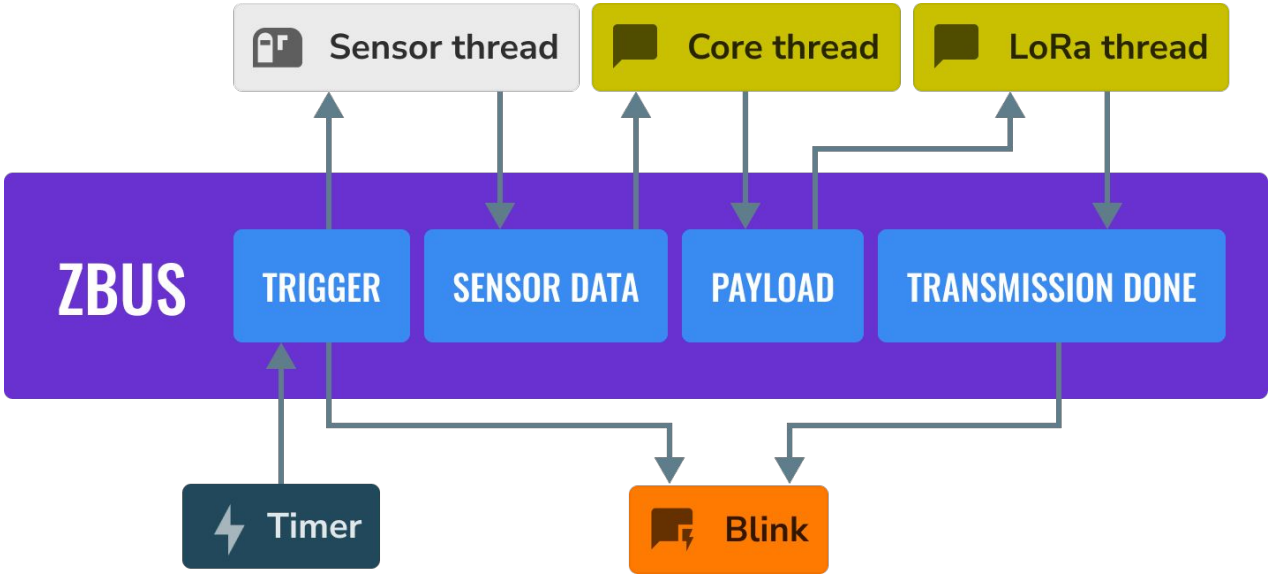
Message queue ❌

Mailbox ❌

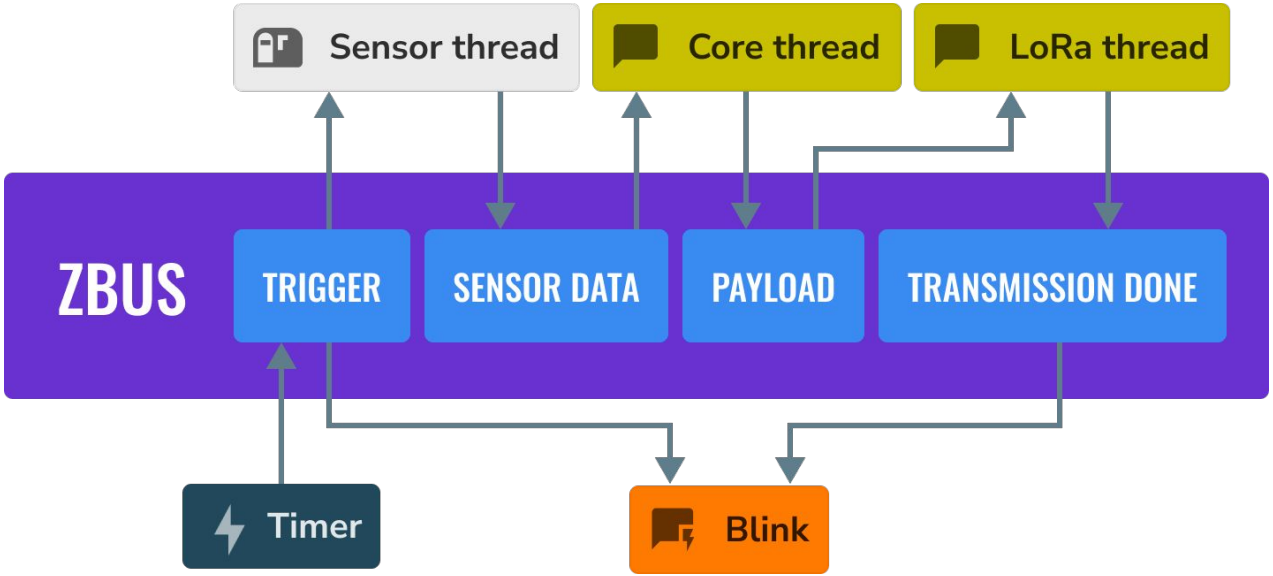
Pipe ❌

Bus ✅

ZBUS


Sensor thread
Core thread
LoRa thread
ZBUS
TRIGGER
SENSOR DATA
PAYLOAD
TRANSMISSION DONE
Timer
Blink

ZBUS

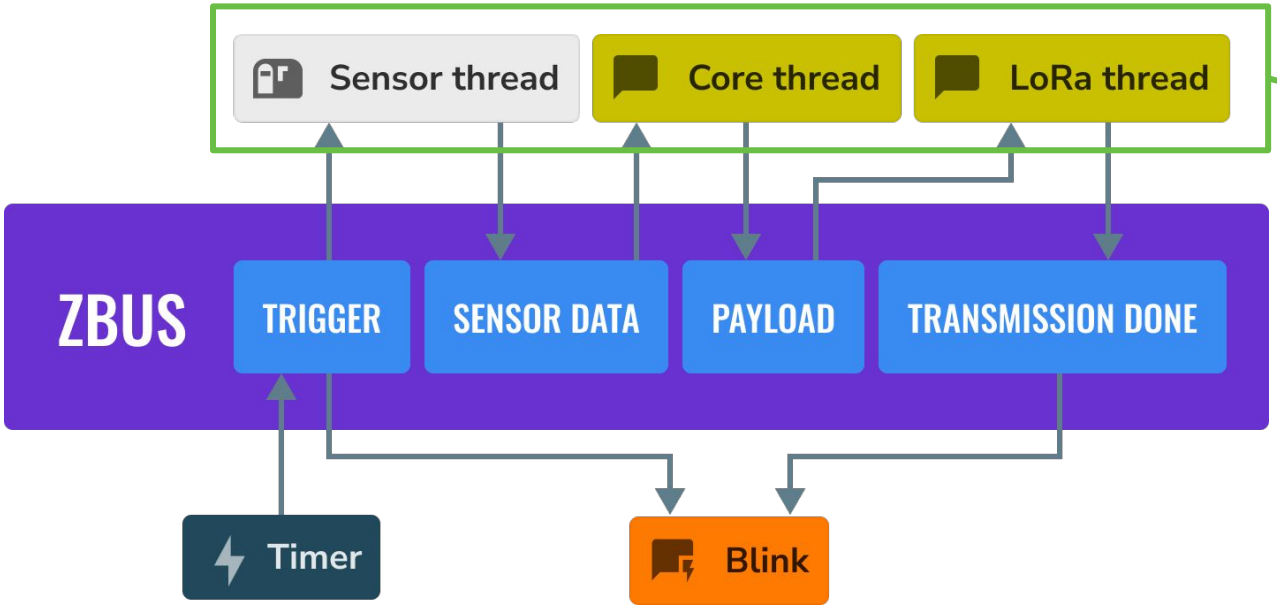


Space

Time

Synchronization

ZBUS

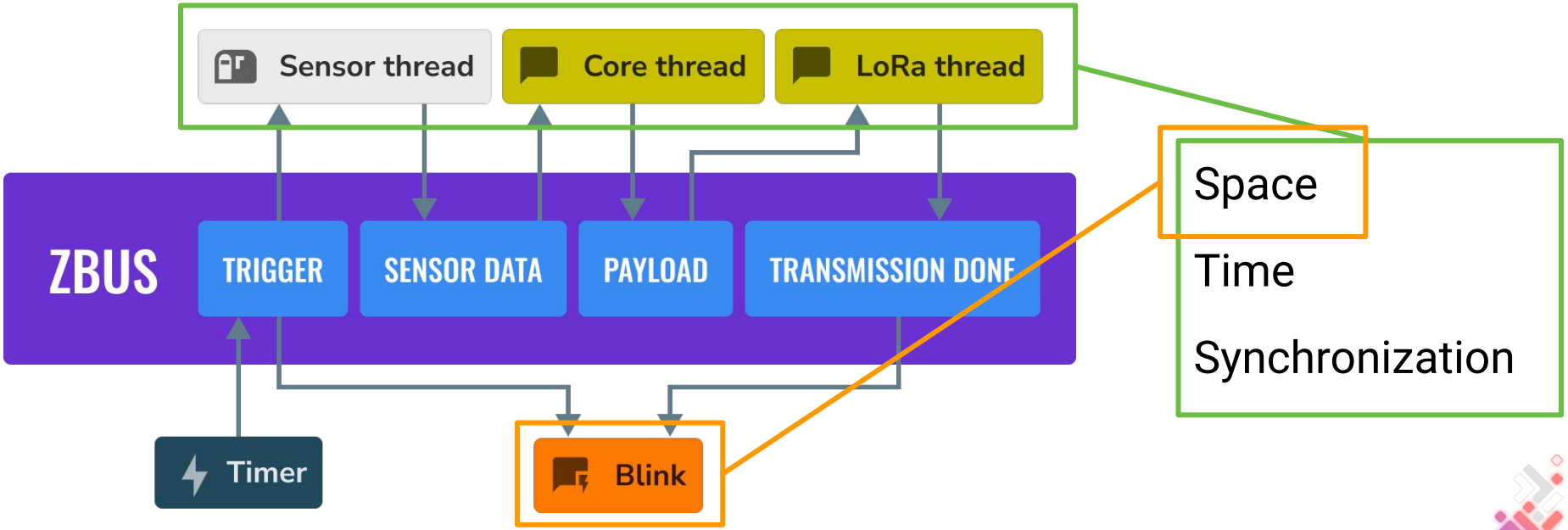


Space

Time

Synchronization

ZBUS


Sensor thread
Core thread
LoRa thread
ZBUS
TRIGGER
SENSOR DATA
PAYLOAD
TRANSMISSION DONE
Timer
Blink
Space
Time
Synchronization

CONTRIBUTION STATISTICS - ZBUS

Github search: *is:pr zbus in:title created:>2023-06-01 is:merged*

28 Pull Requests merged

2 Test improvements

3 Core features

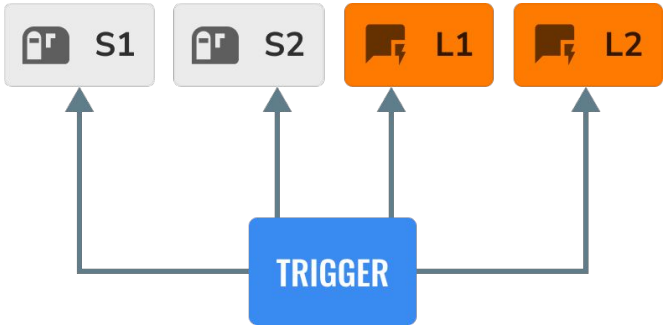
3 Sample improvements or addition

13 Documentation improvements

7 Other improvements

OBSERVATION STORAGE AND OBSERVER MASKS

OBSERVATION STORAGE

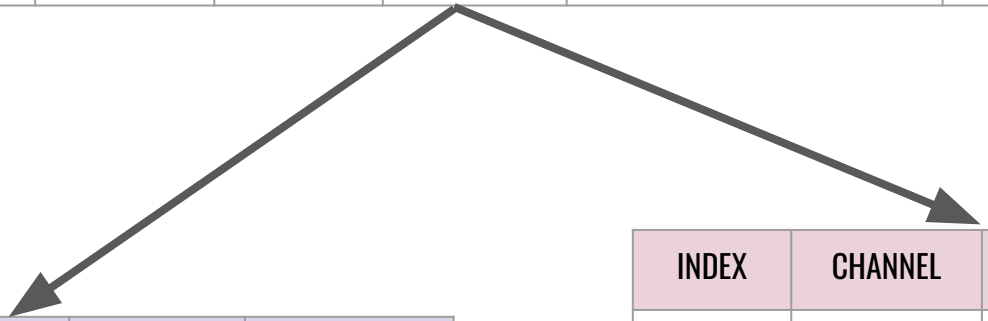



```
struct zbus_channel trigger_chan = {
  ...

  .observers = {&l1, &s1, &l2, &s2, NULL},

  ...
};
```

INDEX	NAME	USER DATA	VALIDATOR	OBSERVERS	INITIAL VALUE
1	trigger	NULL	NULL	l1, s1, l2 and s2	…
2	…	…	…	…	…

PR# 60713

INDEX	NAME	USER DATA	VALIDATOR	OBSERVERS	INITIAL VALUE
1	trigger	NULL	NULL	l1, s1, l2 and s2	…
2	…	…	…	…	…



INDEX	NAME	USER DATA	VALIDATOR	INITIAL VALUE
1	trigger	NULL	NULL	…
2	…	…	…	…

INDEX	CHANNEL	OBSERVER	ENABLED
1	&trigger	&l1	true
2	&trigger	&s1	true
3	&trigger	&l2	true
4	&trigger	&s2	true

PR# 60713

```c
//Trigger channel implementation file

ZBUS_CHAN_DEFINE(trigger_chan,
         struct trigger_msg,
         NULL,
         NULL,
         ZBUS_OBSERVERS(l1, s1),
         ZBUS_MSG_INIT(0)
);
```

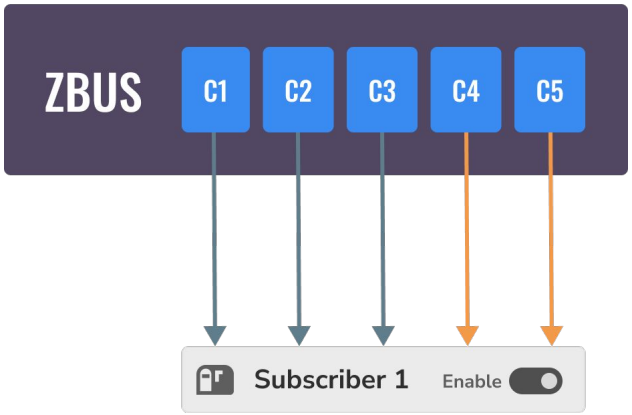
```c
//Listener2 implementation file

ZBUS_CHAN_ADD_OBS(trigger_chan, l2, 3);
```

```c
//Subscriber2 implementation file

ZBUS_CHAN_ADD_OBS(trigger_chan, s2, 4);
```

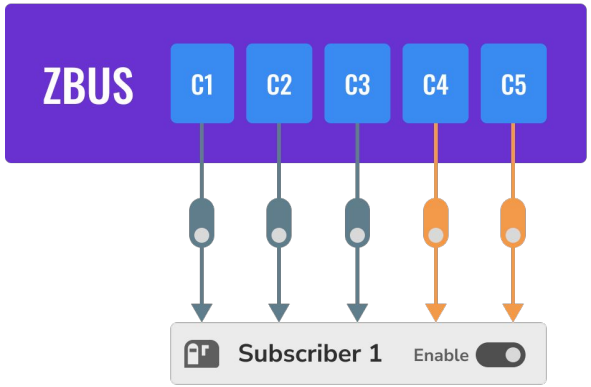

PR# 60713

OBSERVER MASKS

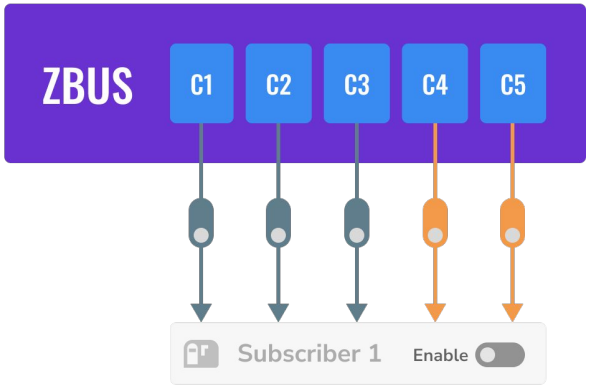
before



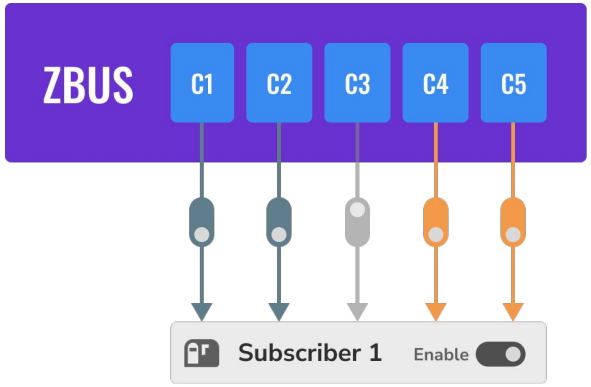
(a)



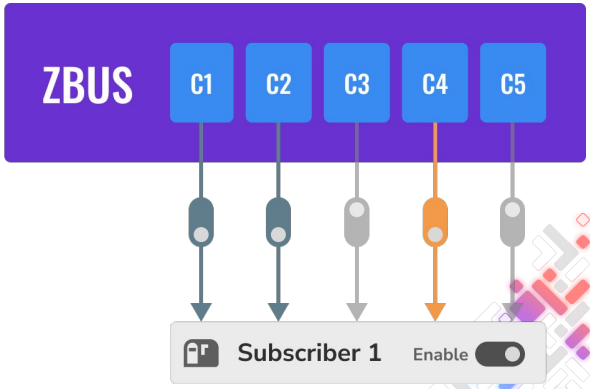
(b)



(c)



(d)

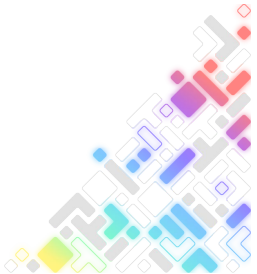


PR# 60713

OBSERVATION STORAGE AND OBSERVER MASKS - SUMMARY

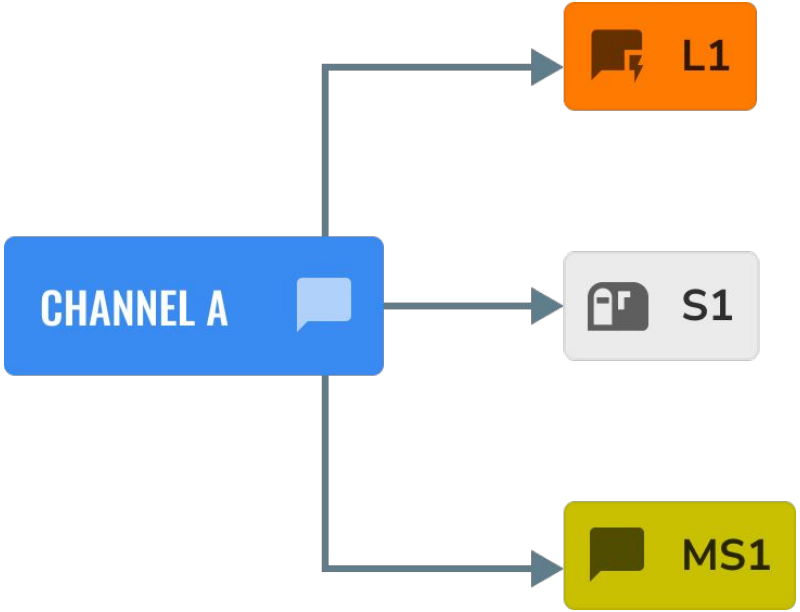
- Static observers in separated files
- Individual mask
- Sequence priority

PR# 60713

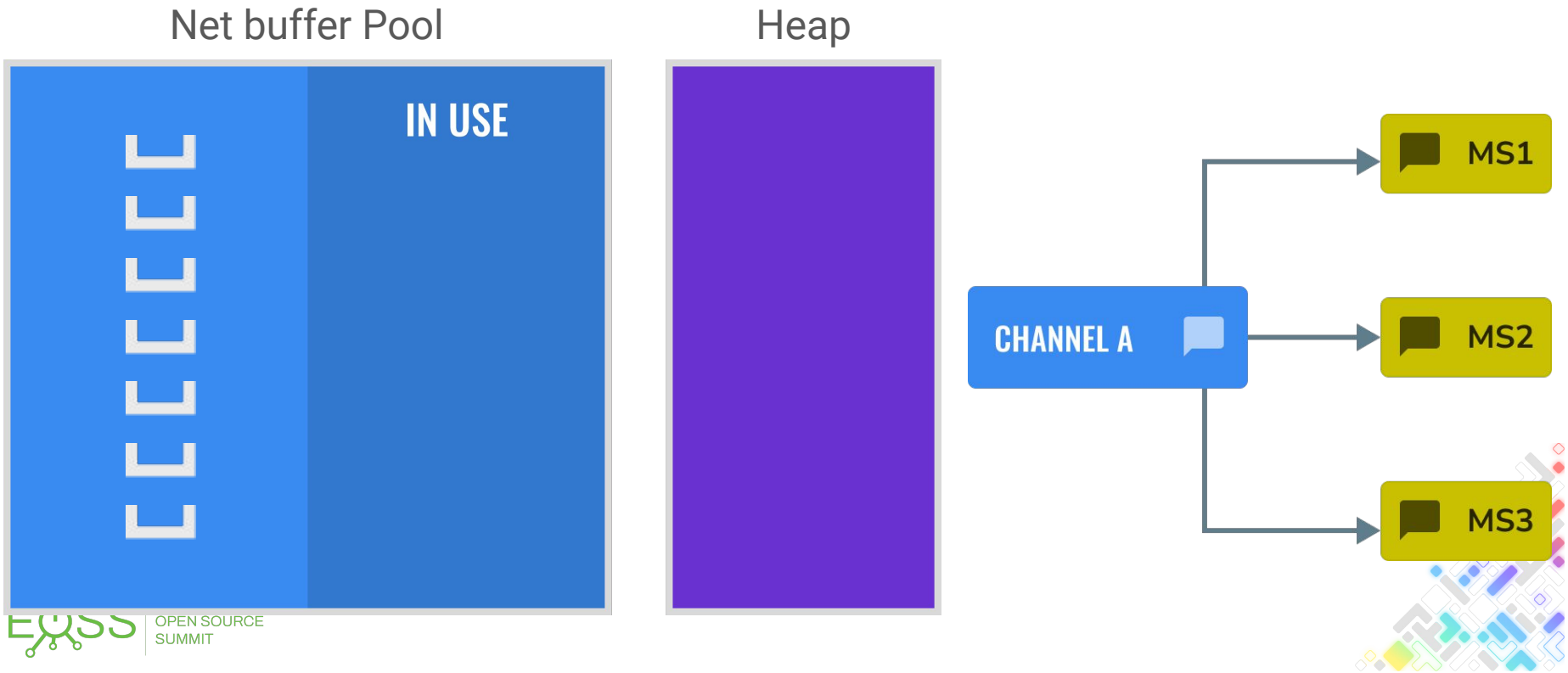


MESSAGE SUBSCRIBERS

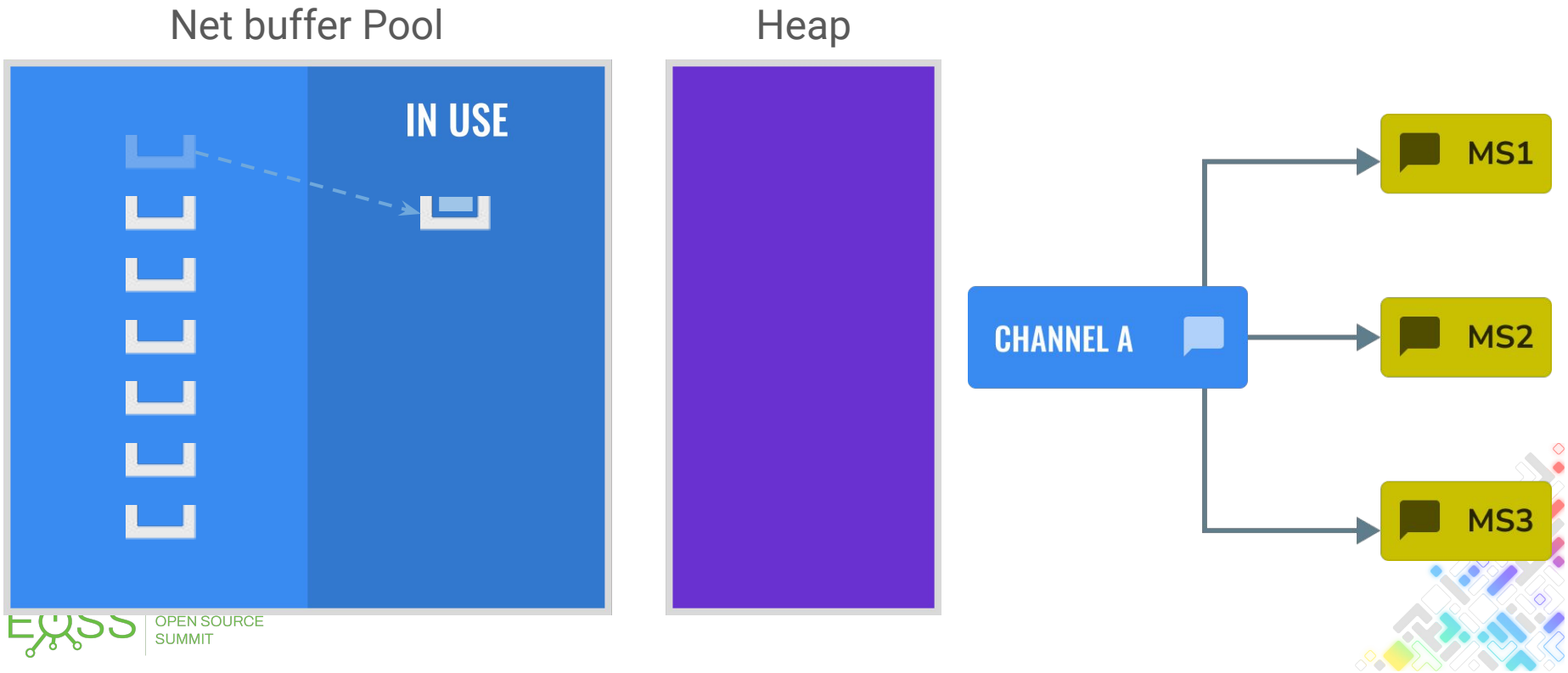
MESSAGE SUBSCRIBERS



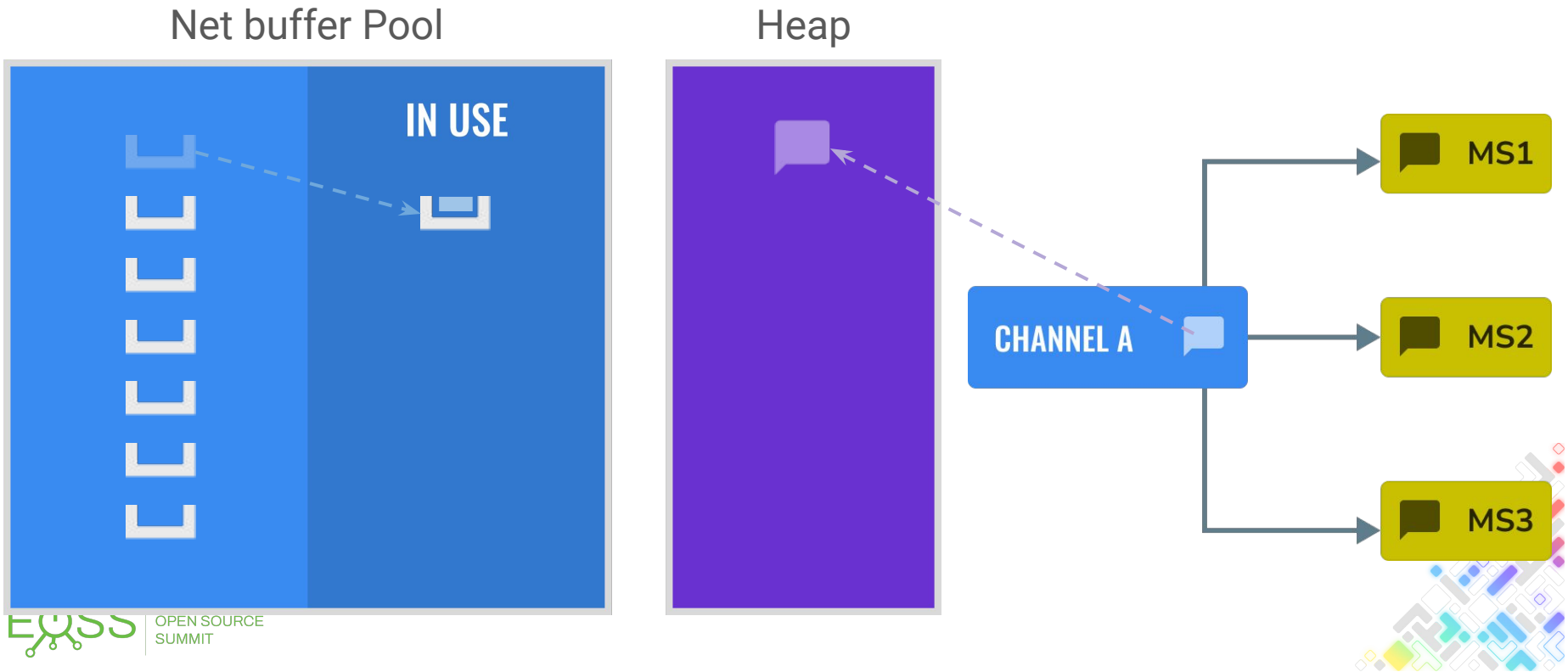
MESSAGE SUBSCRIBER - IMPLEMENTATION



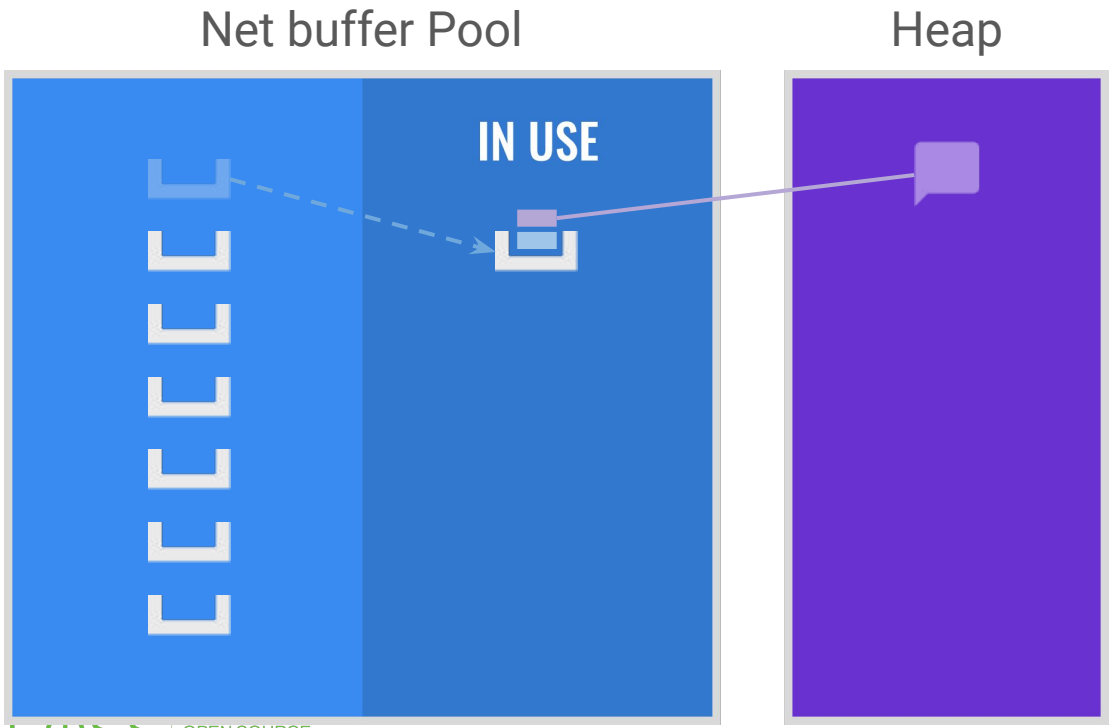
MESSAGE SUBSCRIBER - IMPLEMENTATION

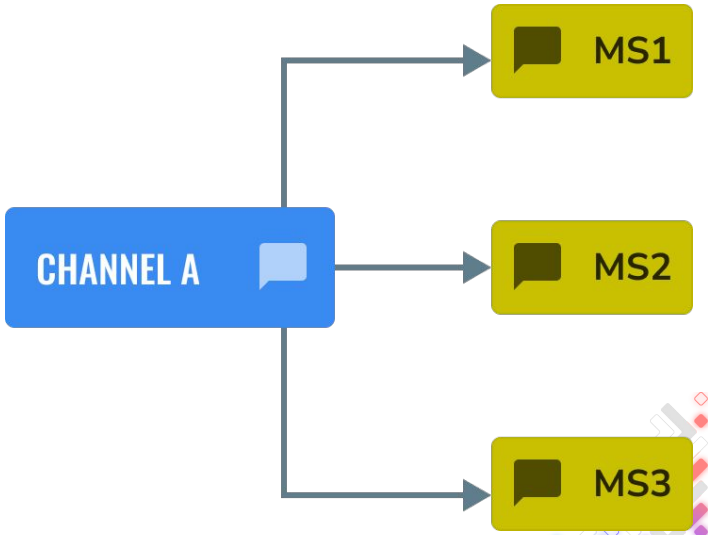


MESSAGE SUBSCRIBER - IMPLEMENTATION

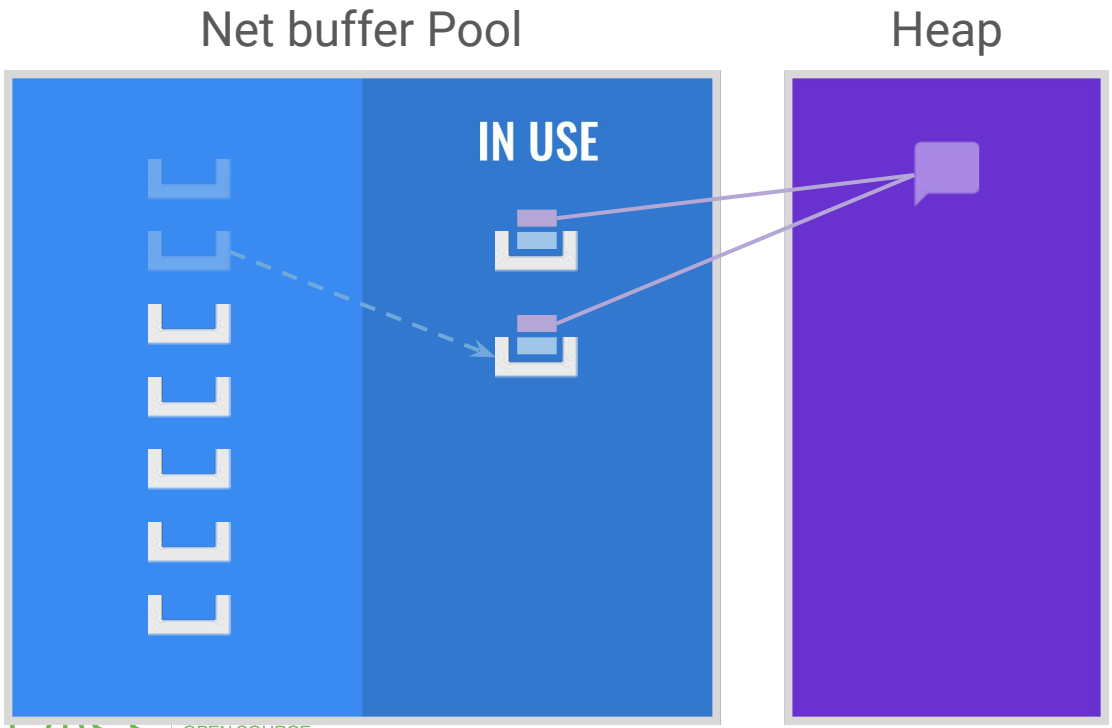


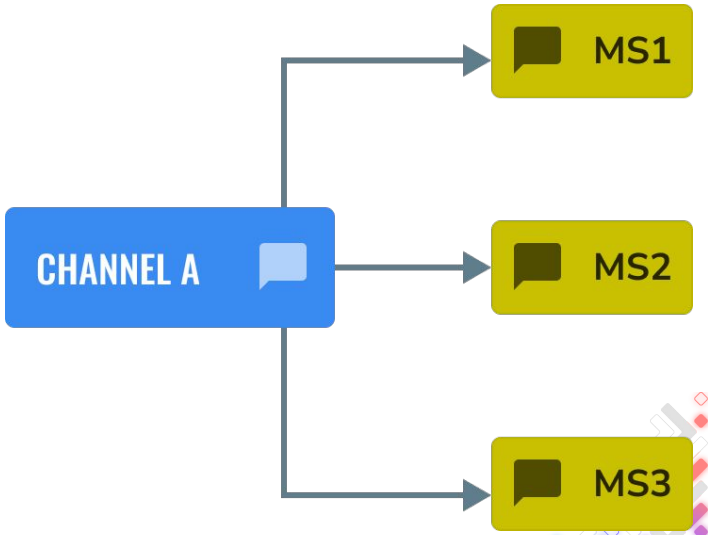
MESSAGE SUBSCRIBER - IMPLEMENTATION



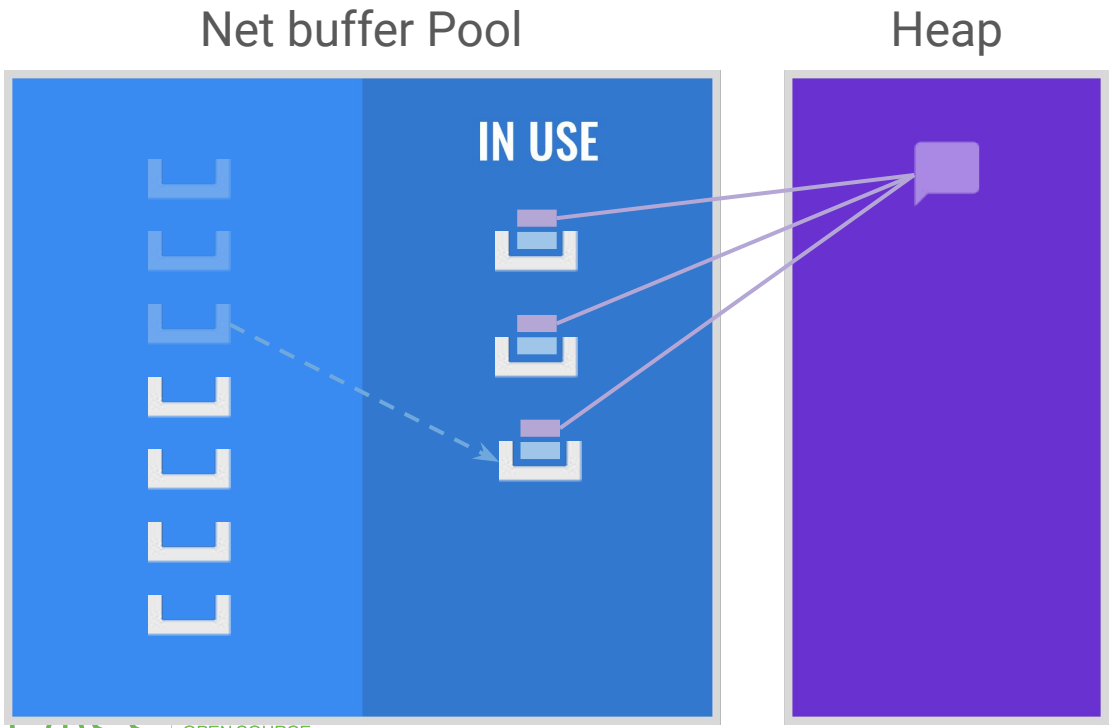


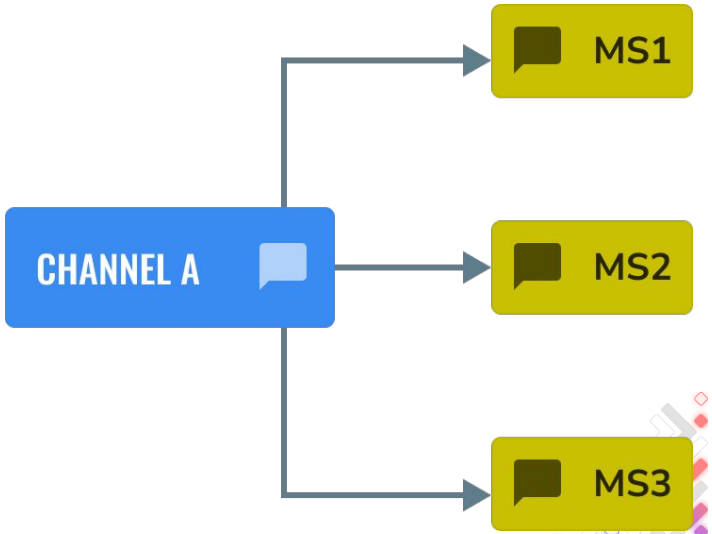
MESSAGE SUBSCRIBER - IMPLEMENTATION



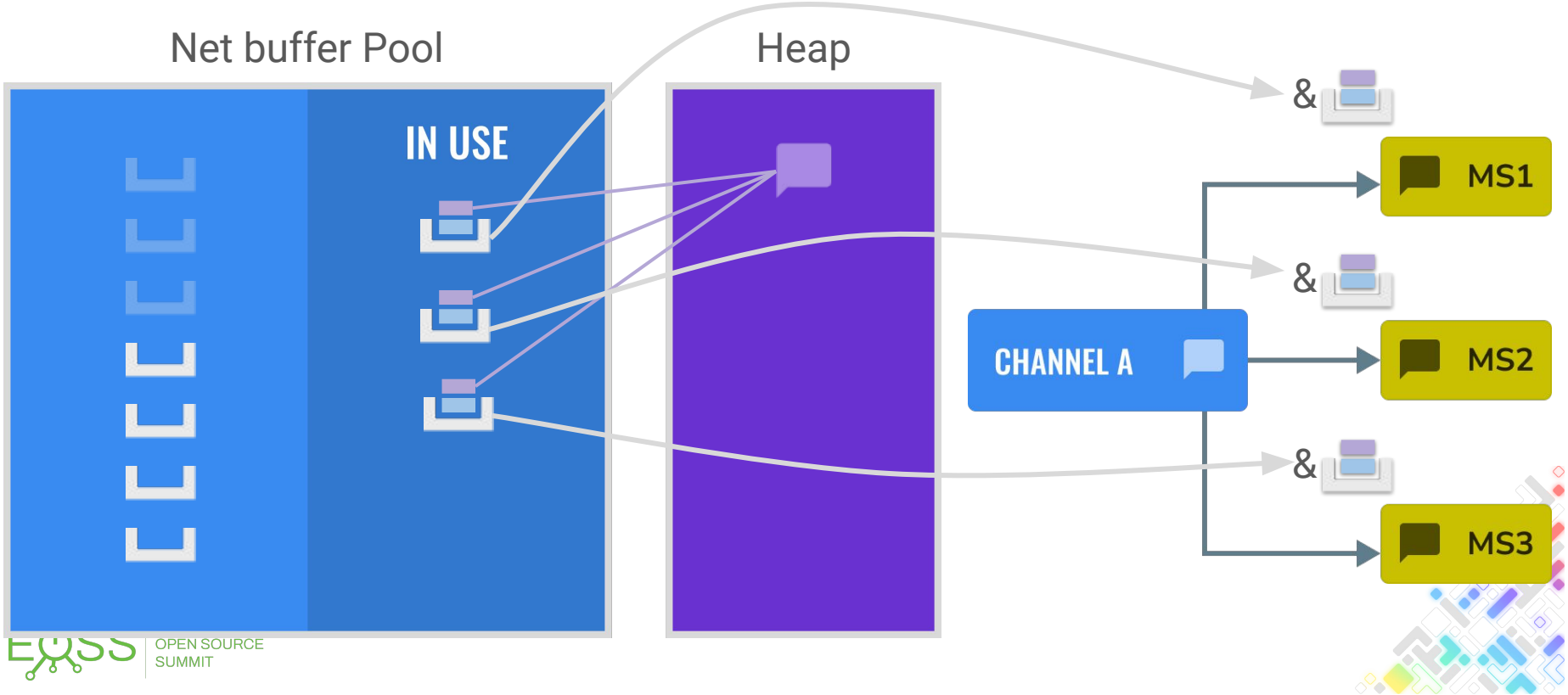


MESSAGE SUBSCRIBER - IMPLEMENTATION

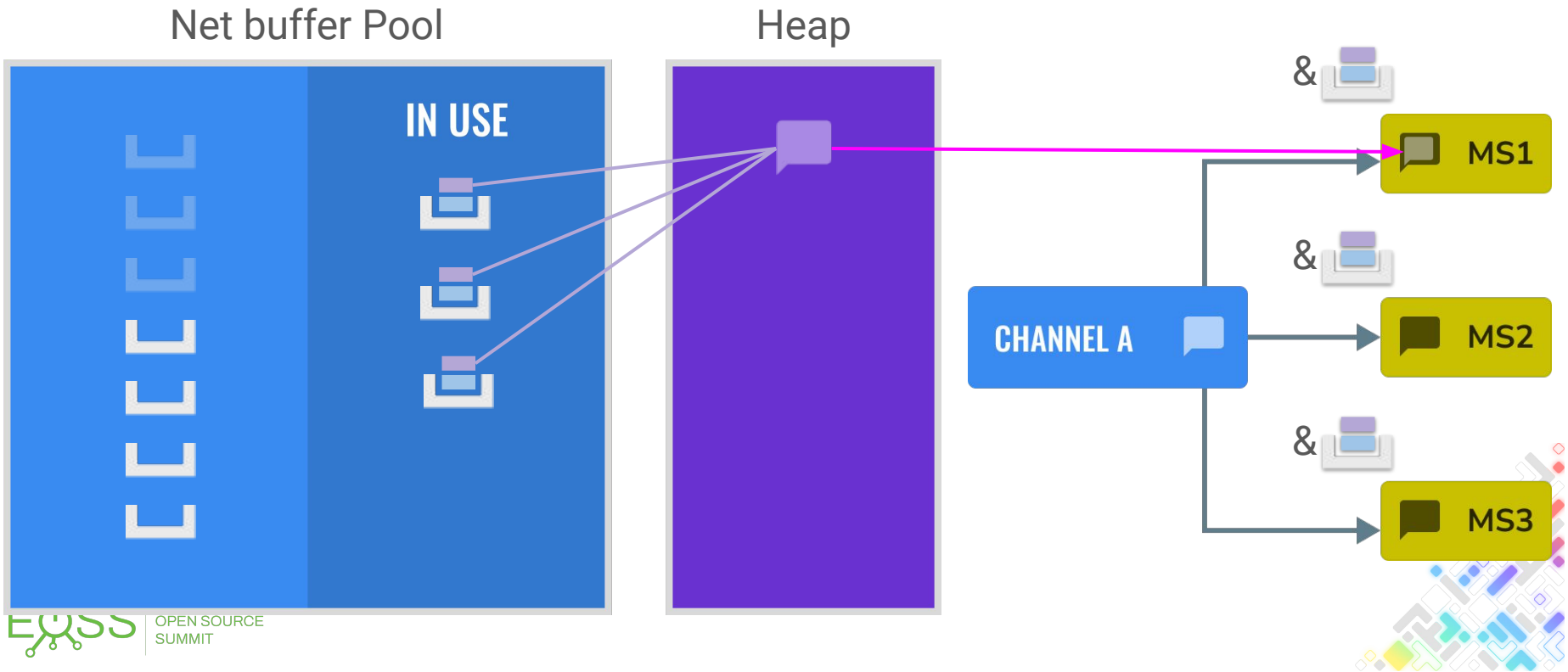




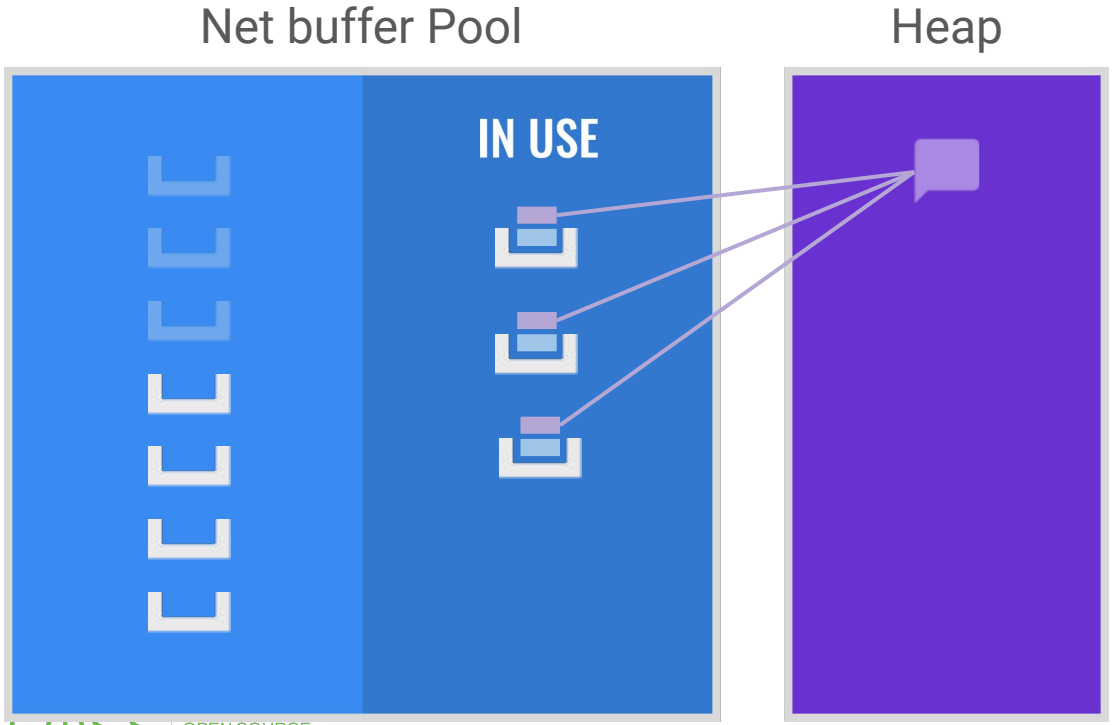
MESSAGE SUBSCRIBER - IMPLEMENTATION

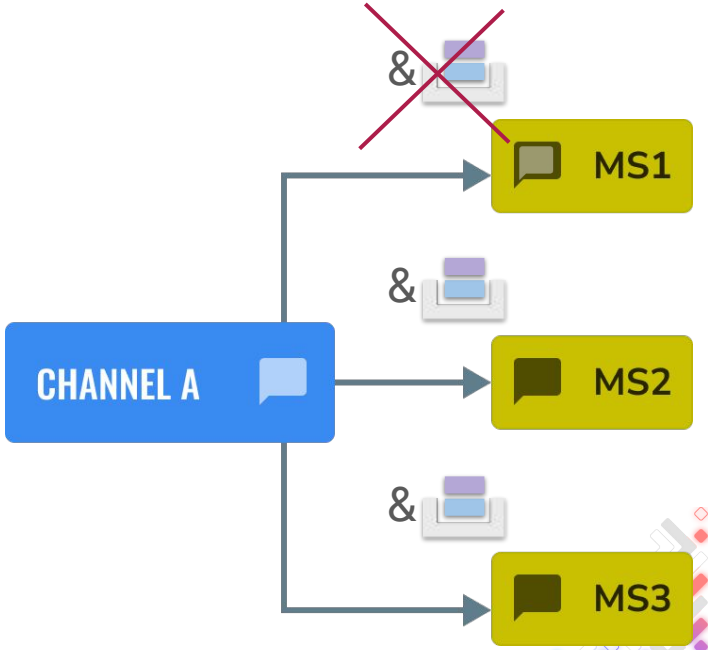


MESSAGE SUBSCRIBER - IMPLEMENTATION

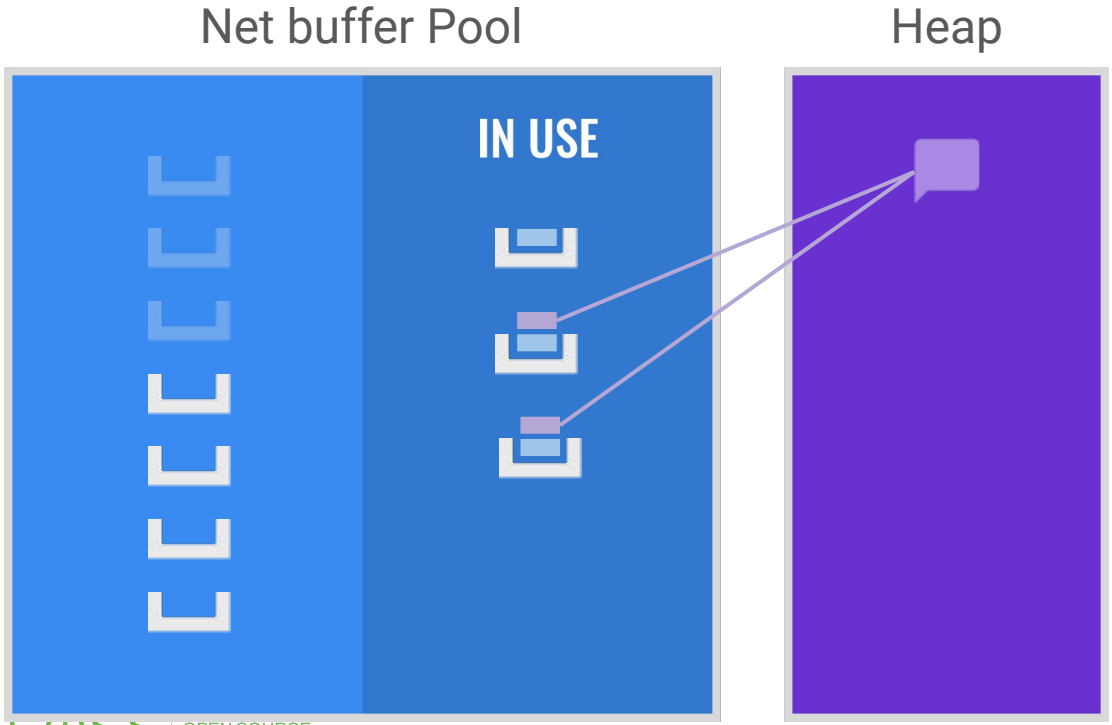


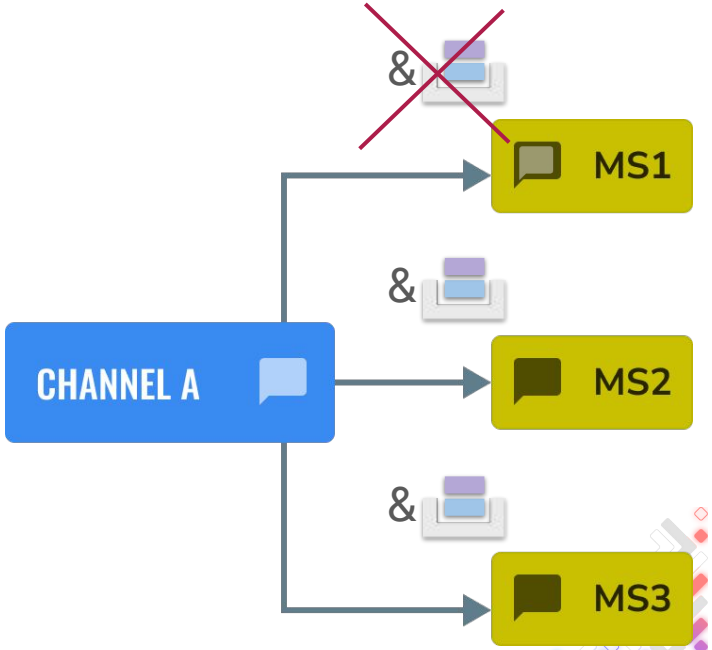
MESSAGE SUBSCRIBER - IMPLEMENTATION



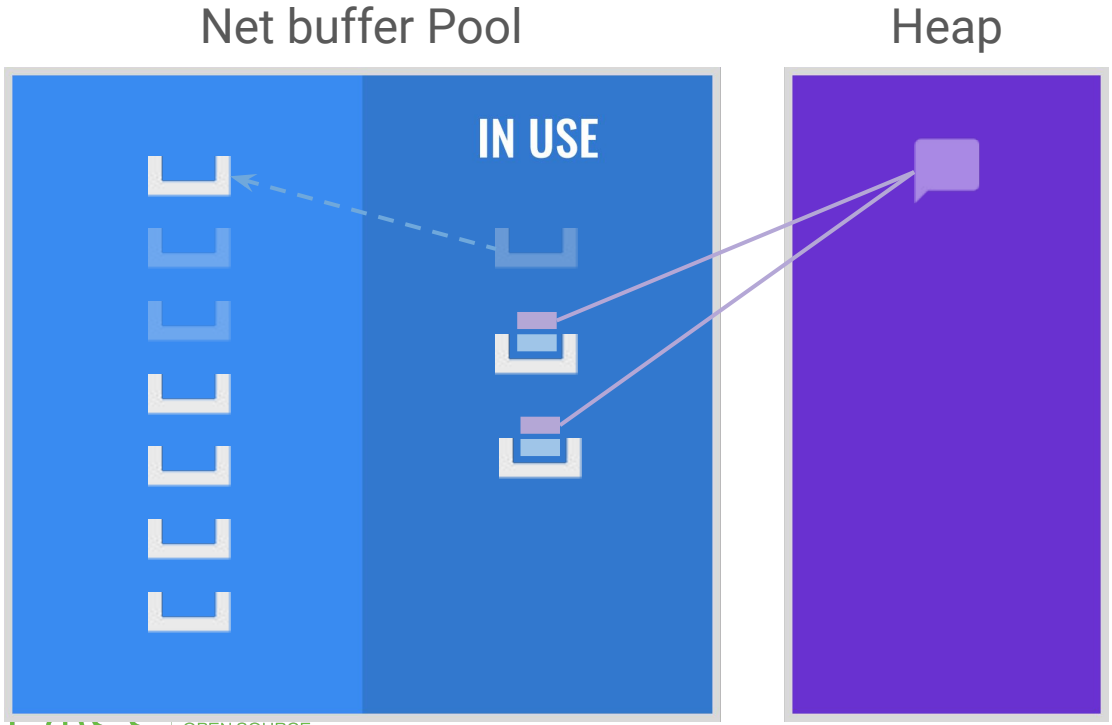


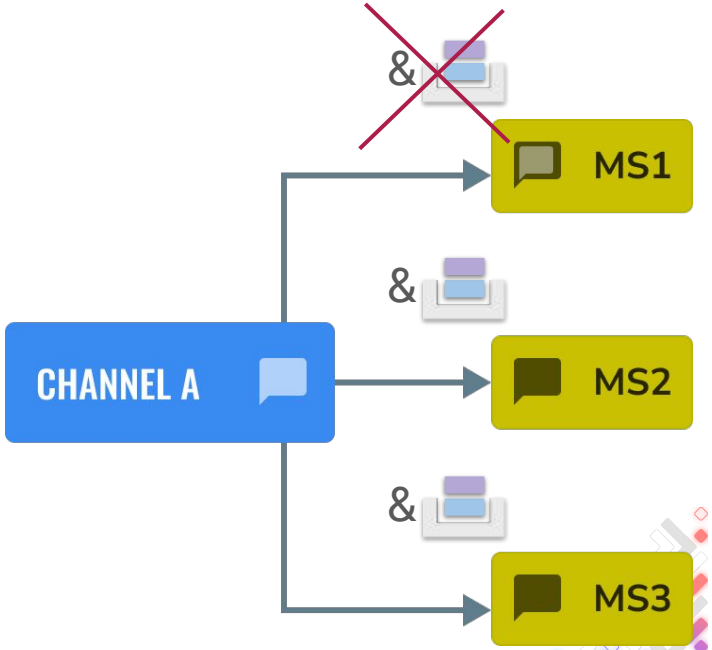
MESSAGE SUBSCRIBER - IMPLEMENTATION



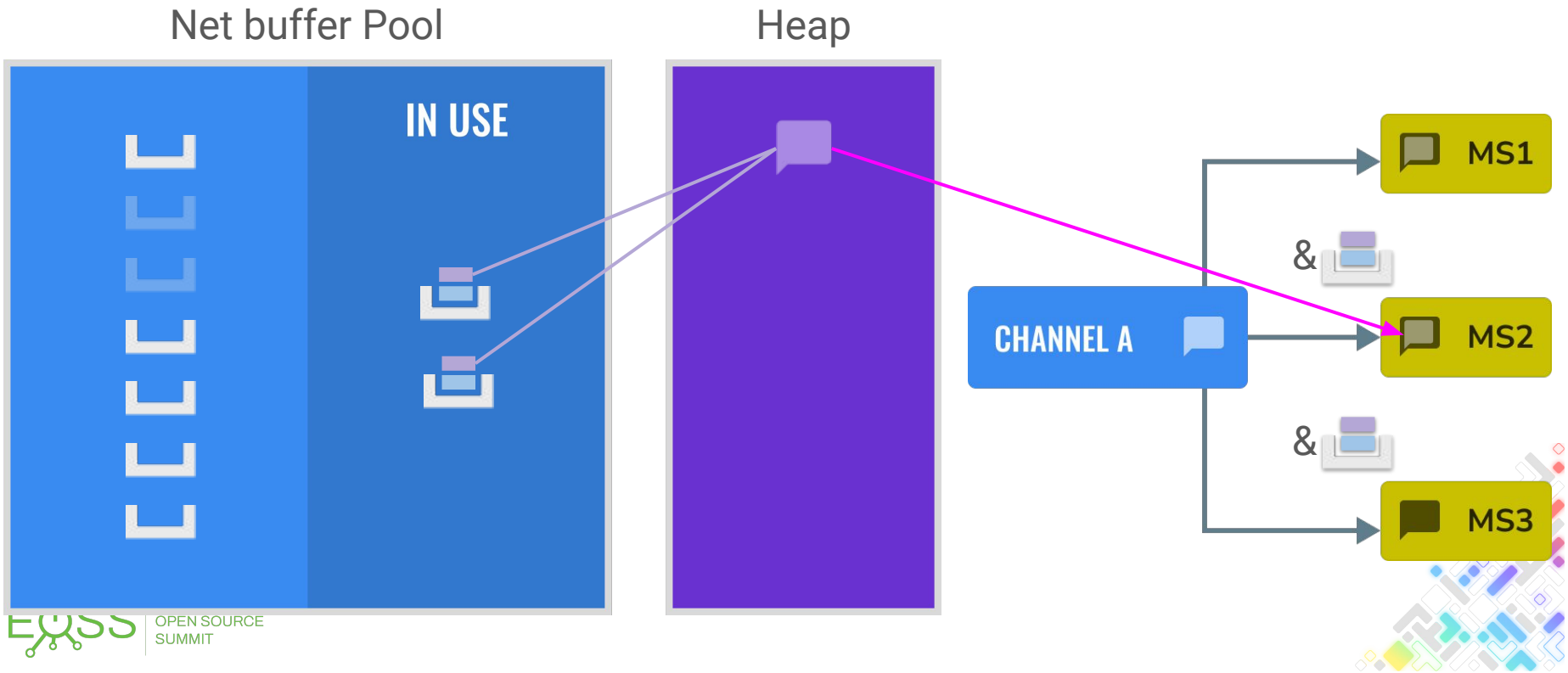


MESSAGE SUBSCRIBER - IMPLEMENTATION

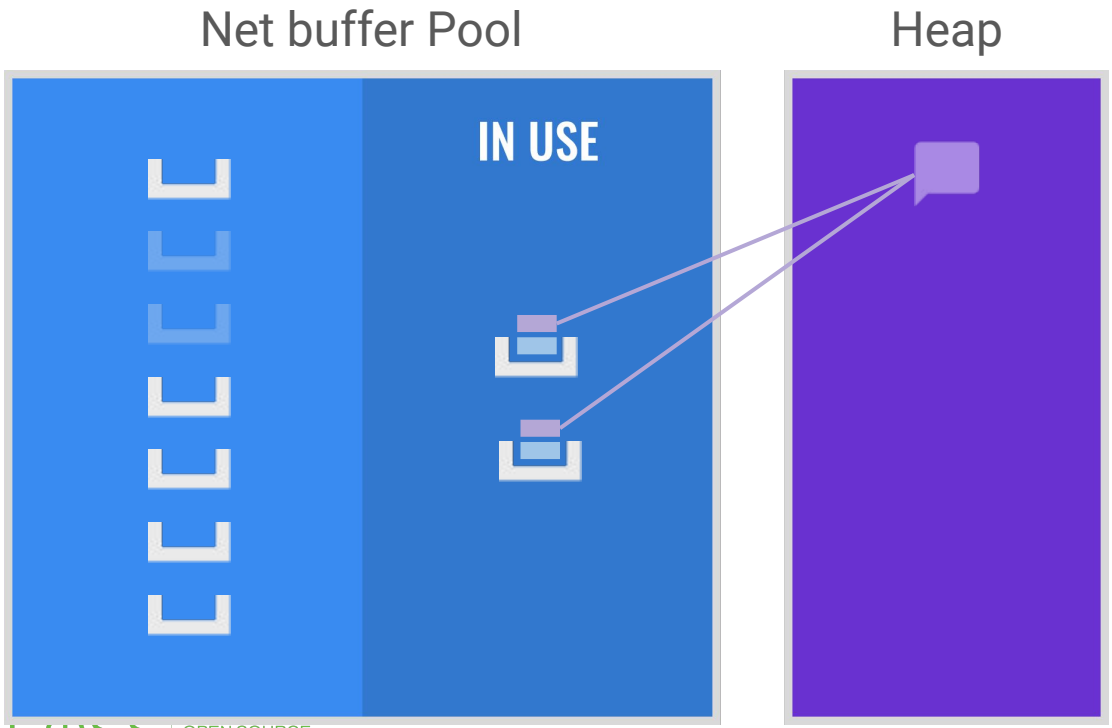


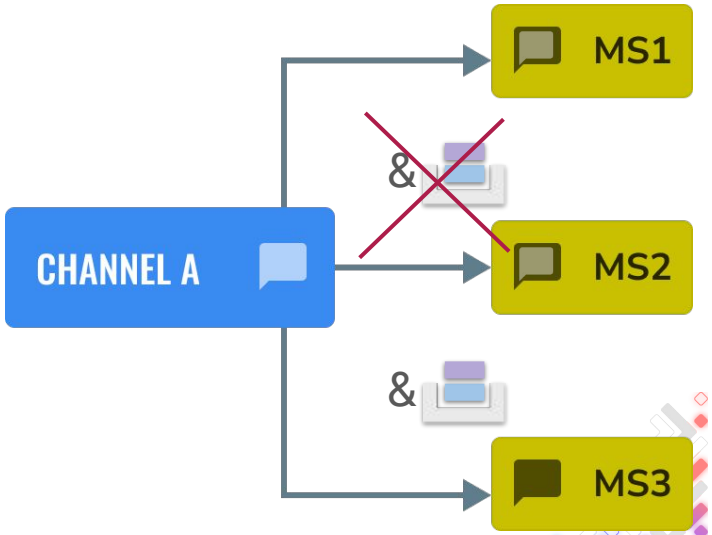


MESSAGE SUBSCRIBER - IMPLEMENTATION

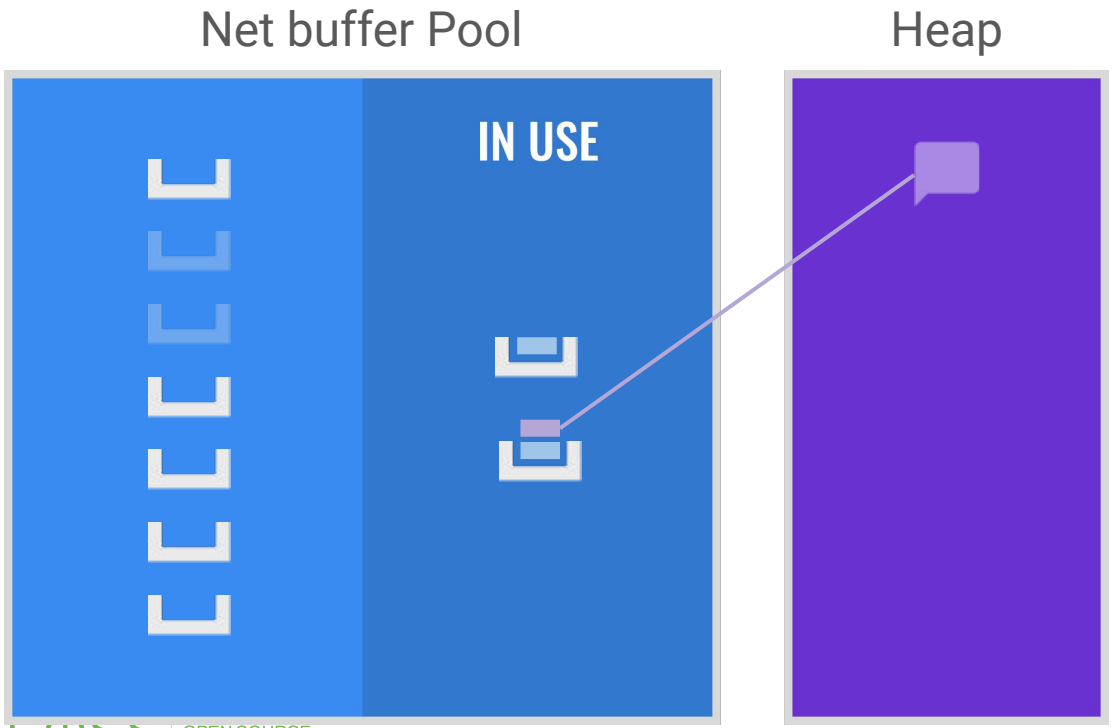


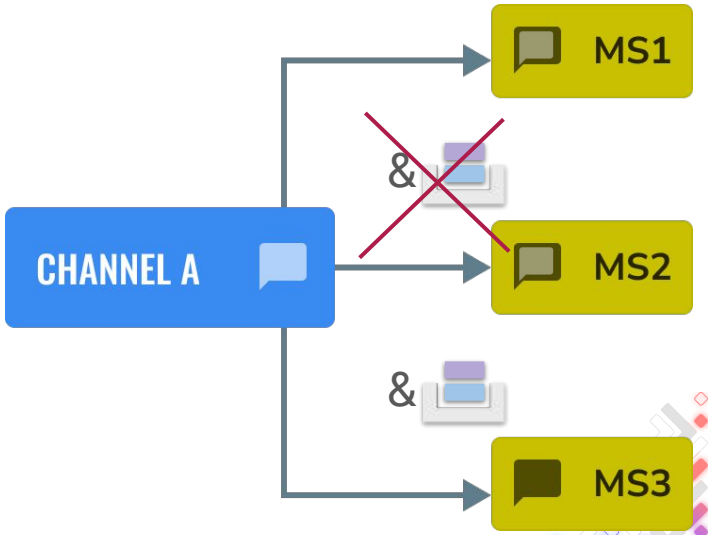
MESSAGE SUBSCRIBER - IMPLEMENTATION



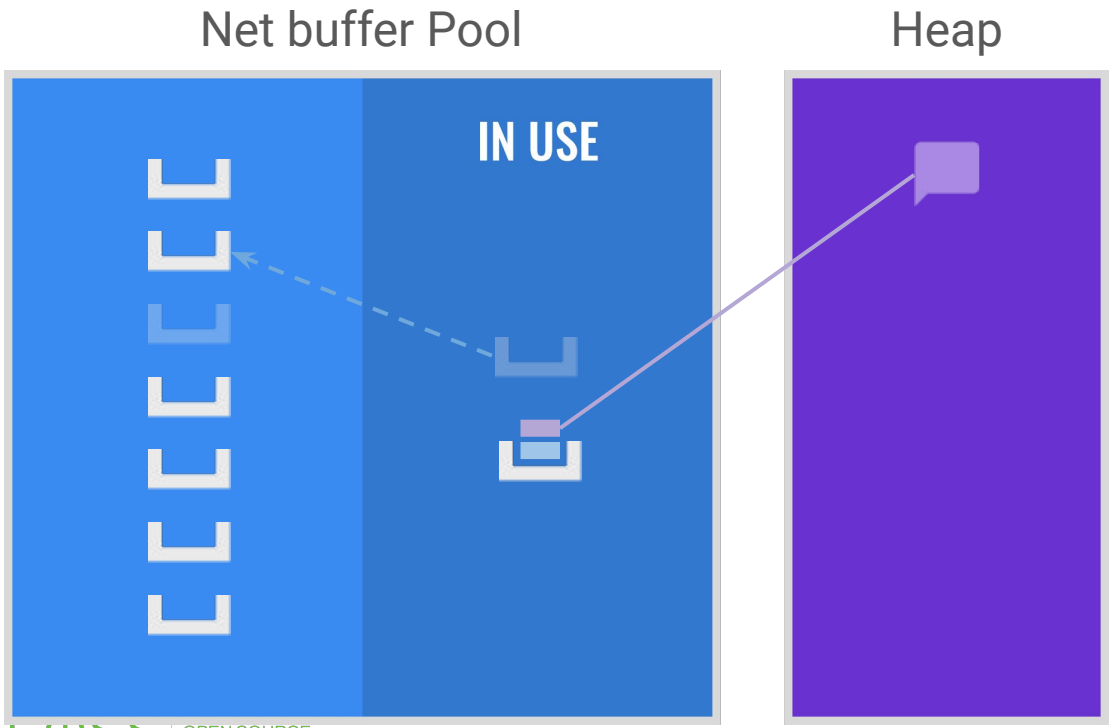


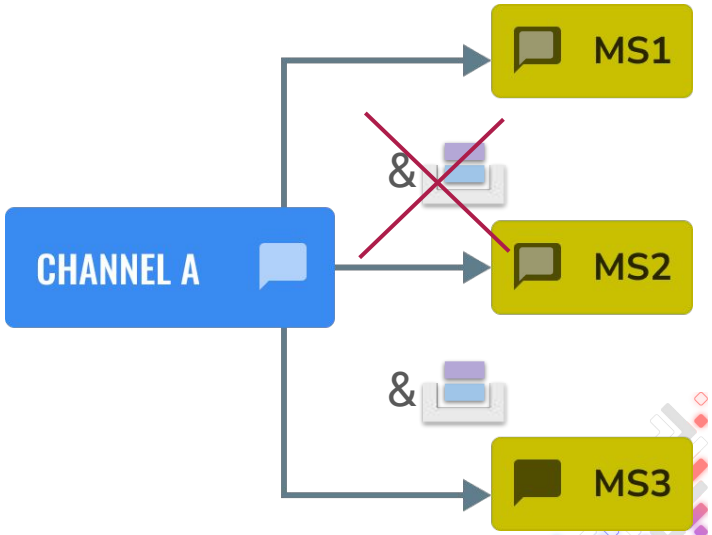
MESSAGE SUBSCRIBER - IMPLEMENTATION



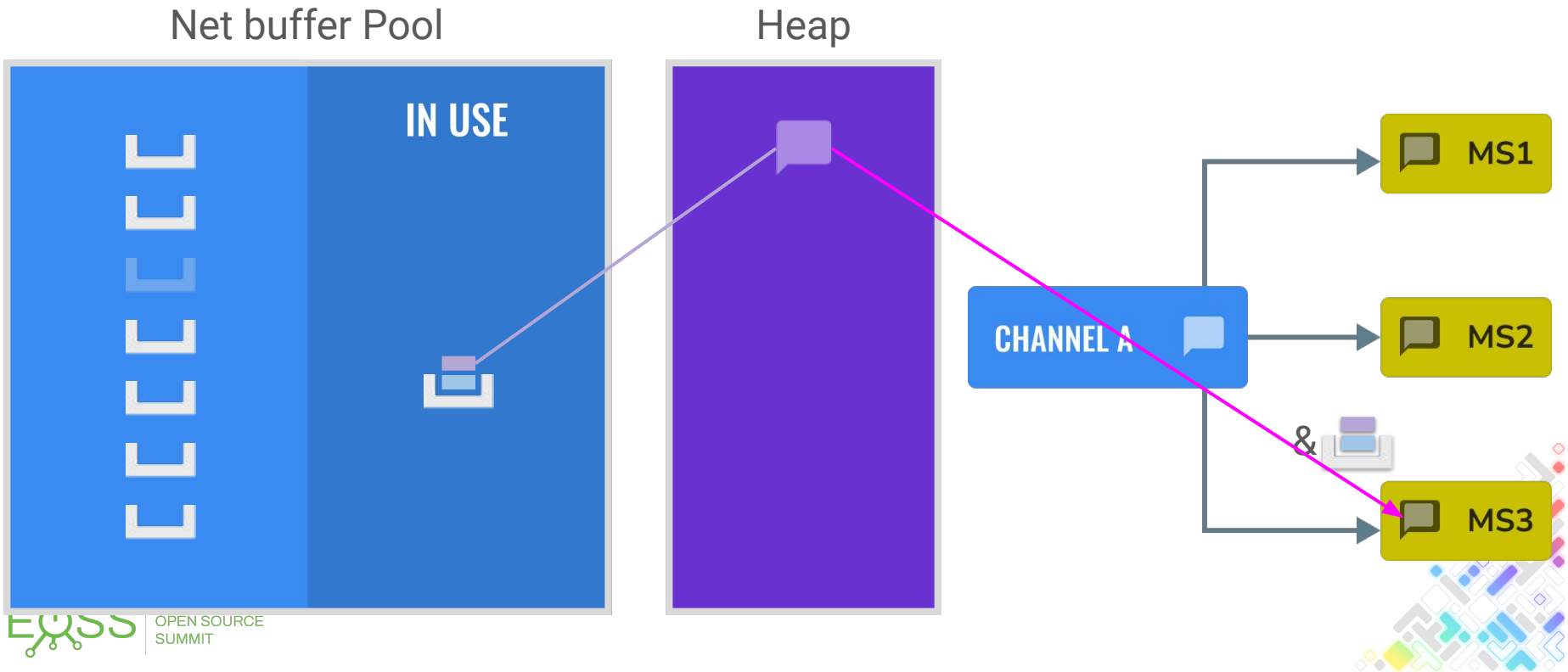


MESSAGE SUBSCRIBER - IMPLEMENTATION

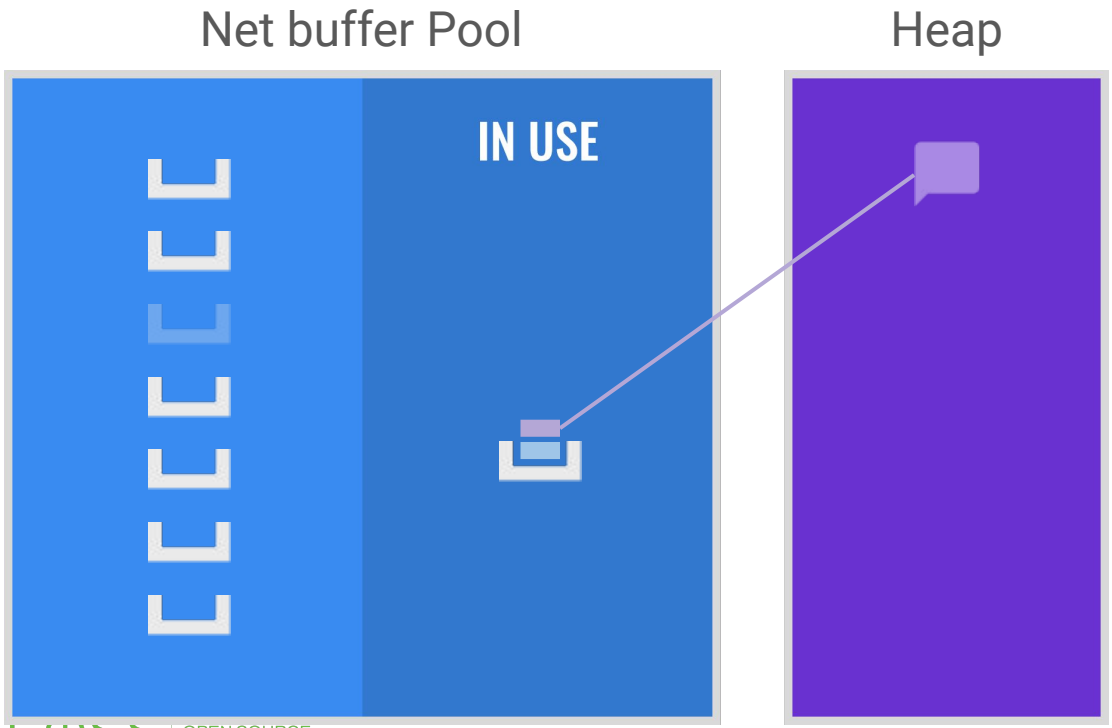


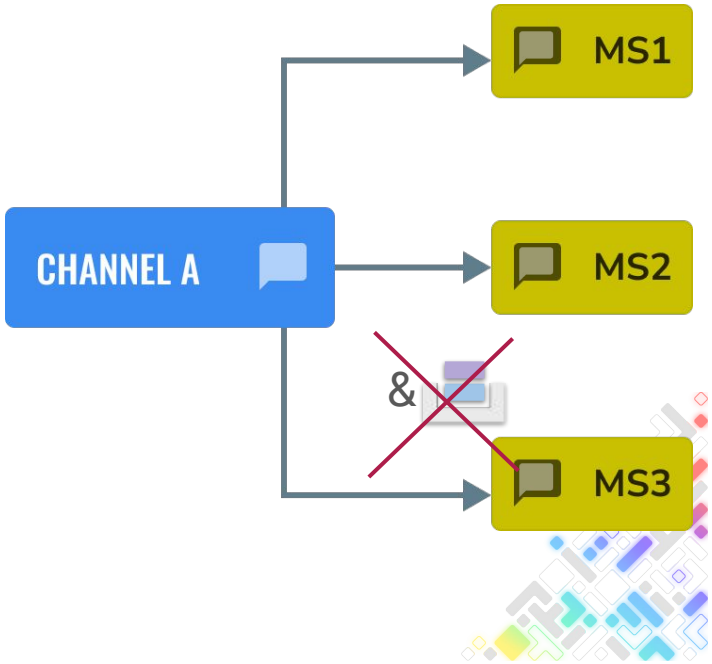


MESSAGE SUBSCRIBER - IMPLEMENTATION

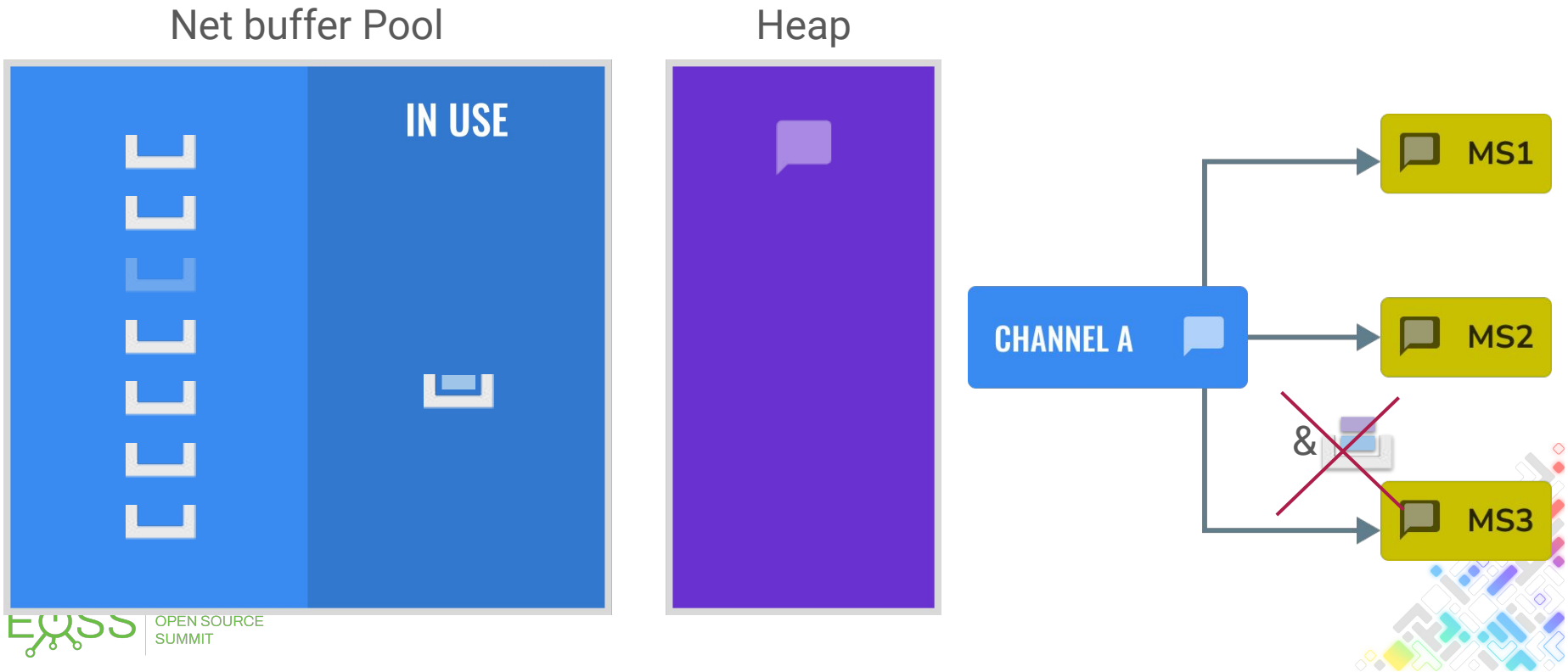


MESSAGE SUBSCRIBER - IMPLEMENTATION

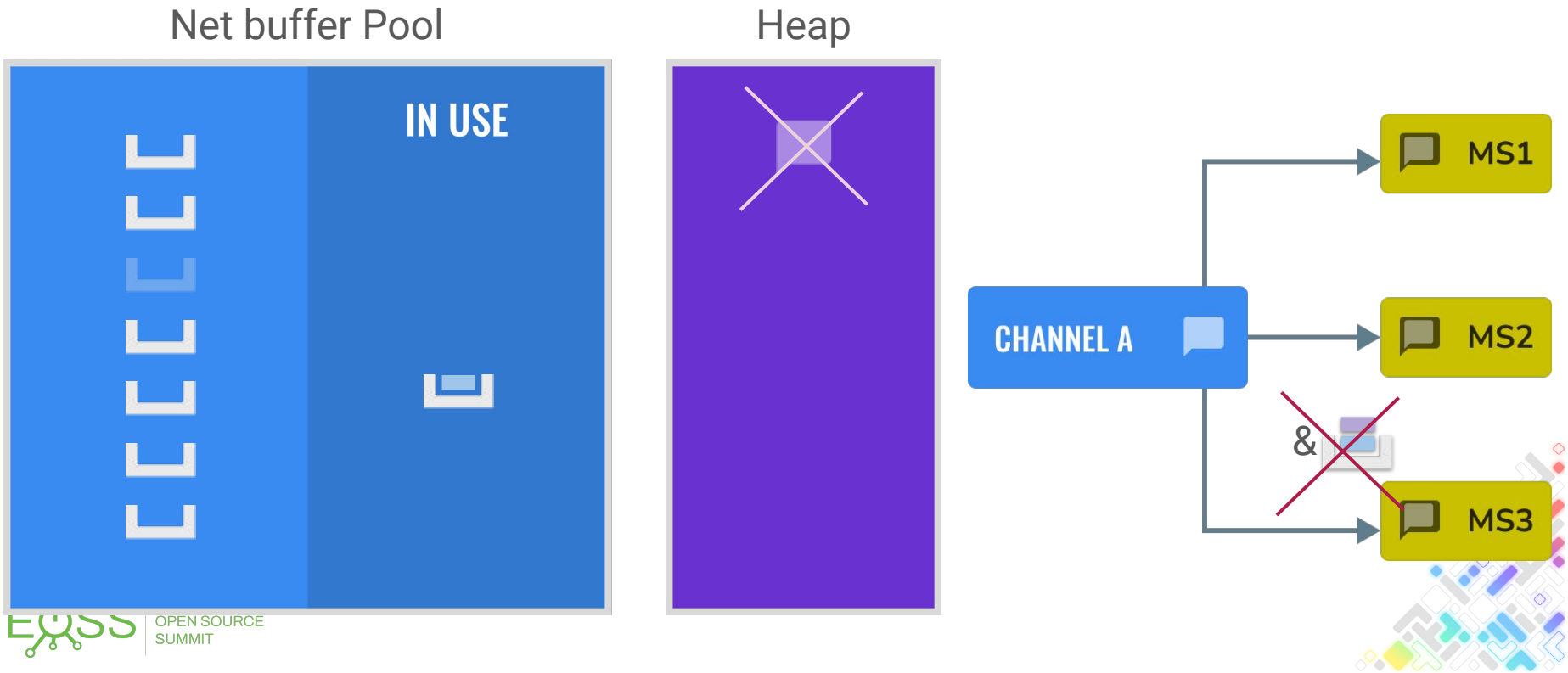




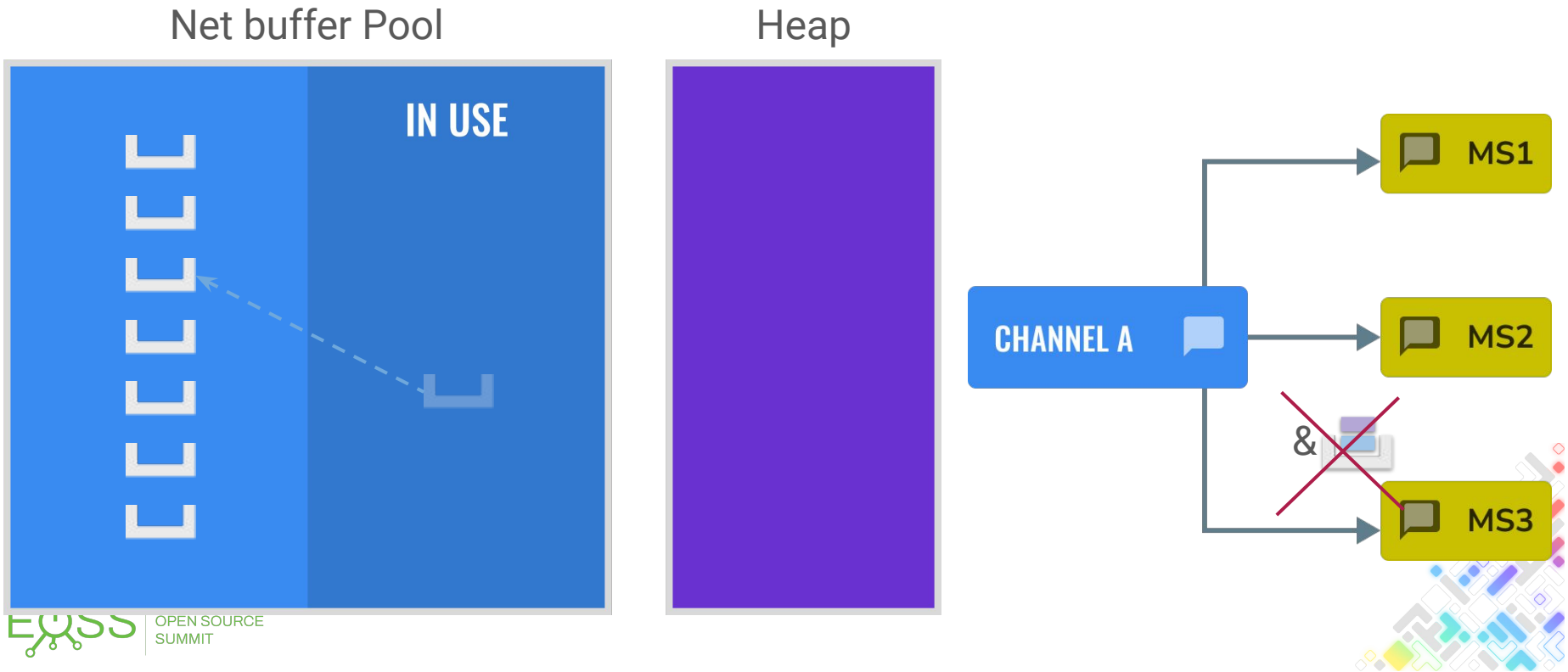
MESSAGE SUBSCRIBER - IMPLEMENTATION



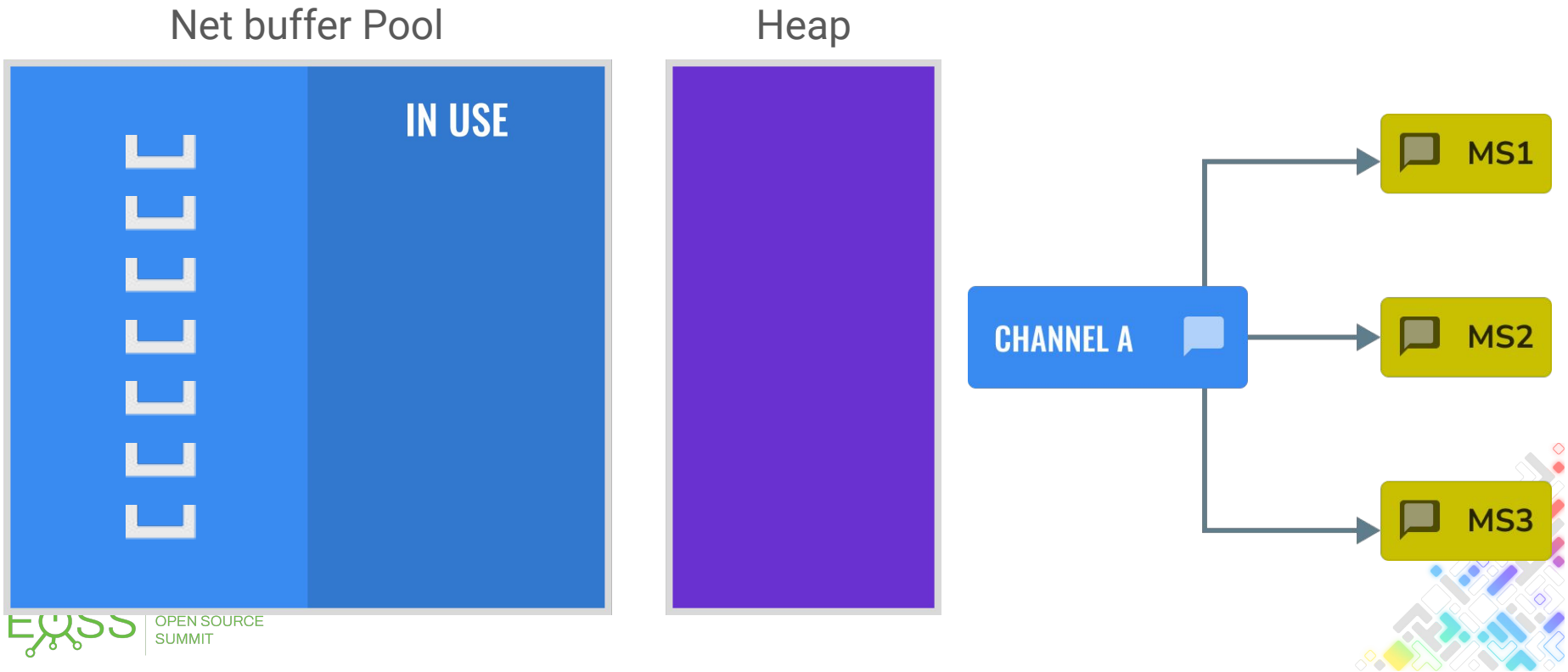
MESSAGE SUBSCRIBER - IMPLEMENTATION



MESSAGE SUBSCRIBER - IMPLEMENTATION



MESSAGE SUBSCRIBER - IMPLEMENTATION



MESSAGE SUBSCRIBERS - TIPS

Static allocation is also possible (kconfig)
Take care of the pool (shared among all the channels)
Unions for multiple channels with different messages

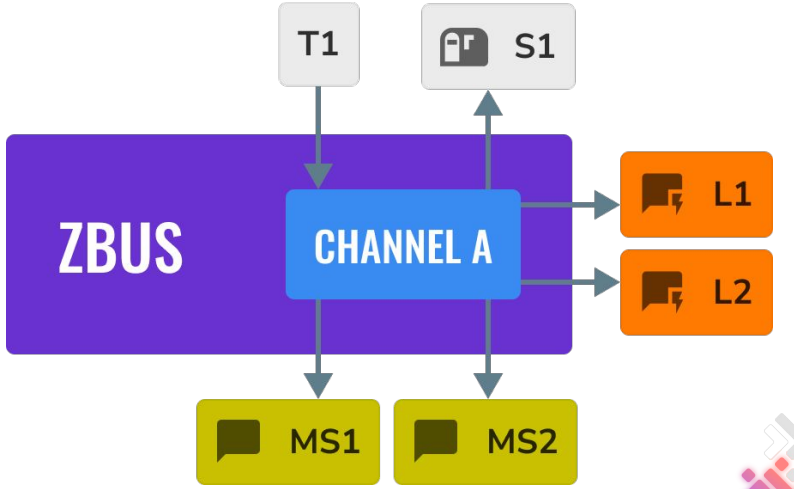
```
union {
    struct msg_a ma;
    struct msg_b mb;
} msg;

zbus_sub_wait_msg(&msub, &chan, &msg, K_FOREVER);
```

HIGHEST LOCKER PRIORITY PROTOCOL

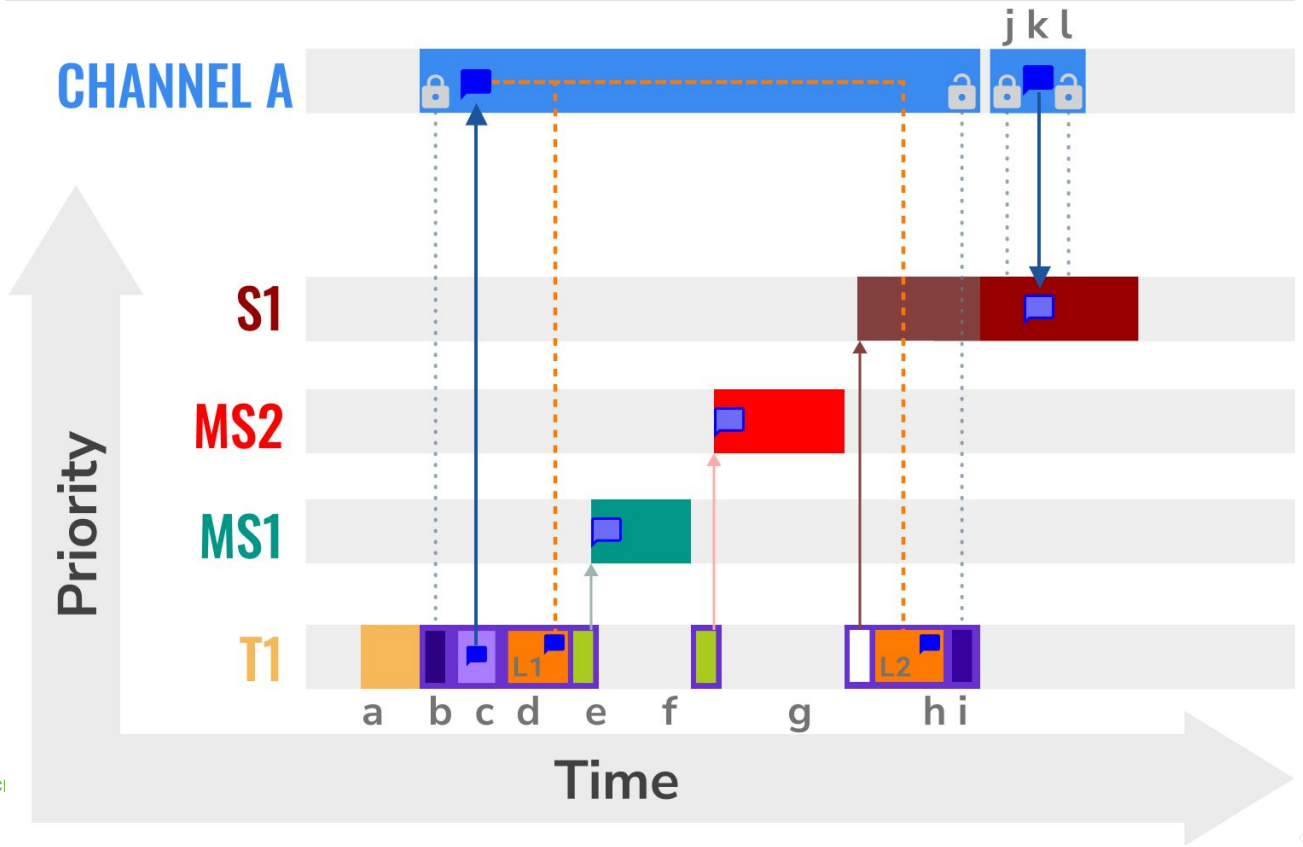
PRIORITY INVERSION - SCENARIO

Priorities S1 > MS2 > MS1 > T1

```
ZBUS_CHAN_DEFINE(a_chan,
         struct a_msg,
         NULL,
         NULL,
         ZBUS_OBSERVERS(L1, MS1, MS2, S1, L2),
         ZBUS_MSG_INIT(0)
);
```



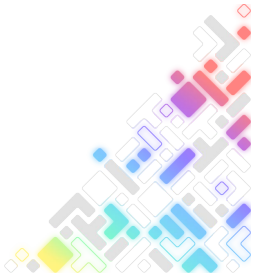
PRIORITY INVERSION - SCENARIO



HIGHEST LOCKER PRIORITY PROTOCOL

*"… for every critical resource is assigned a **ceiling priority value**. This value is the maximum of priorities of all those tasks which **may** request to hold this critical resource."*

Source *https://www.geeksforgeeks.org/highest-locker-protocol-hlp/*

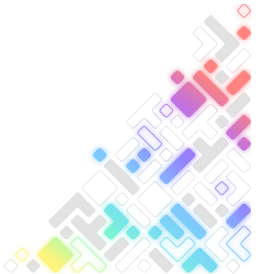


HIGHEST LOCKER PRIORITY PROTOCOL

Channel

*"… every ~~critical resource~~ is assigned a **ceiling priority value**. This value is the maximum of priorities of all those tasks which **may** request to hold this ~~critical resource~~."*

Source https://www.geeksforgeeks.org/highest-locker-protocol-hlp/



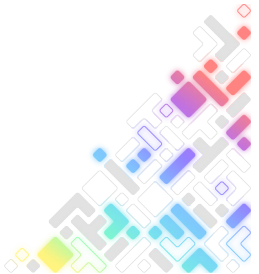
HIGHEST LOCKER PRIORITY PROTOCOL

Channel

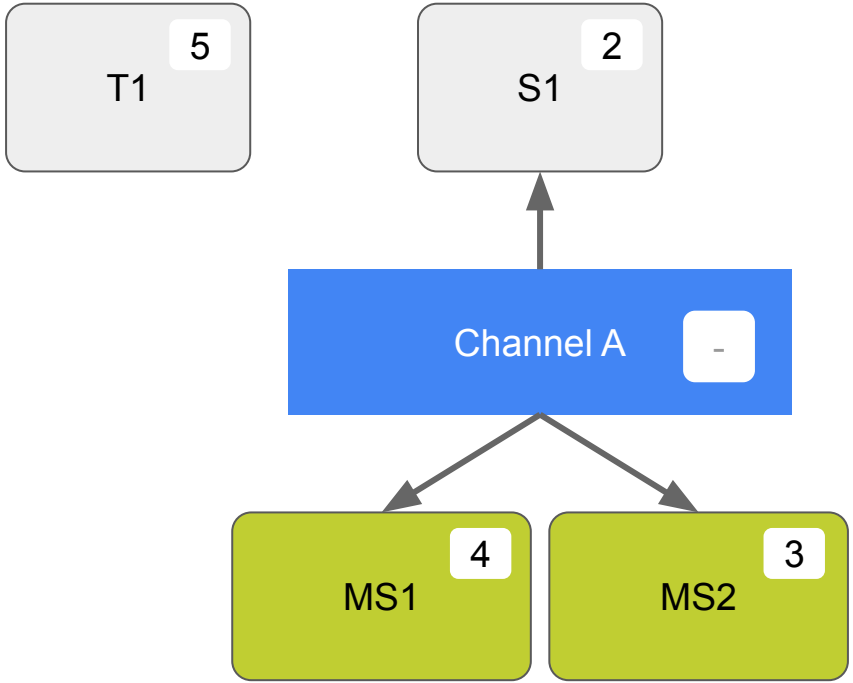
*"... every ~~critical resource~~ is assigned a **ceiling priority value**. This value is the maximum of priorities of all those ~~tasks~~ which **may** request to hold this ~~critical resource~~."*

Observers

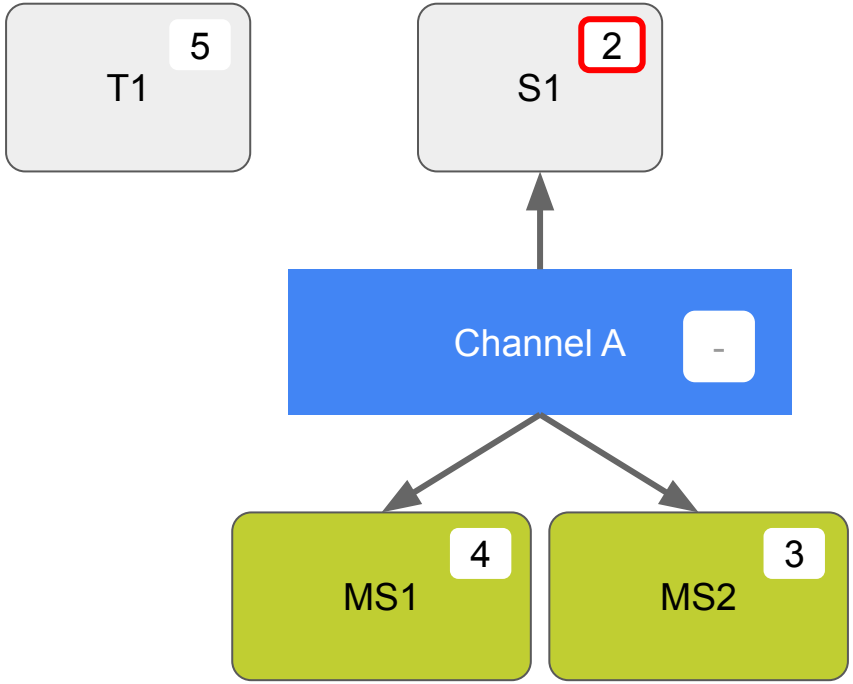
Source https://www.geeksforgeeks.org/highest-locker-protocol-hlp/



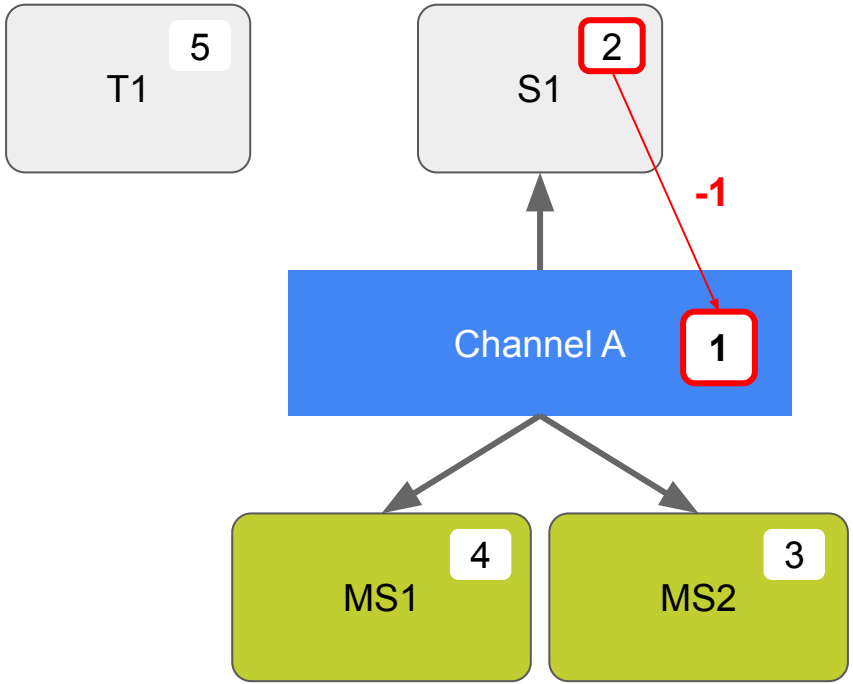
HLP PRIORITY PROTOCOL



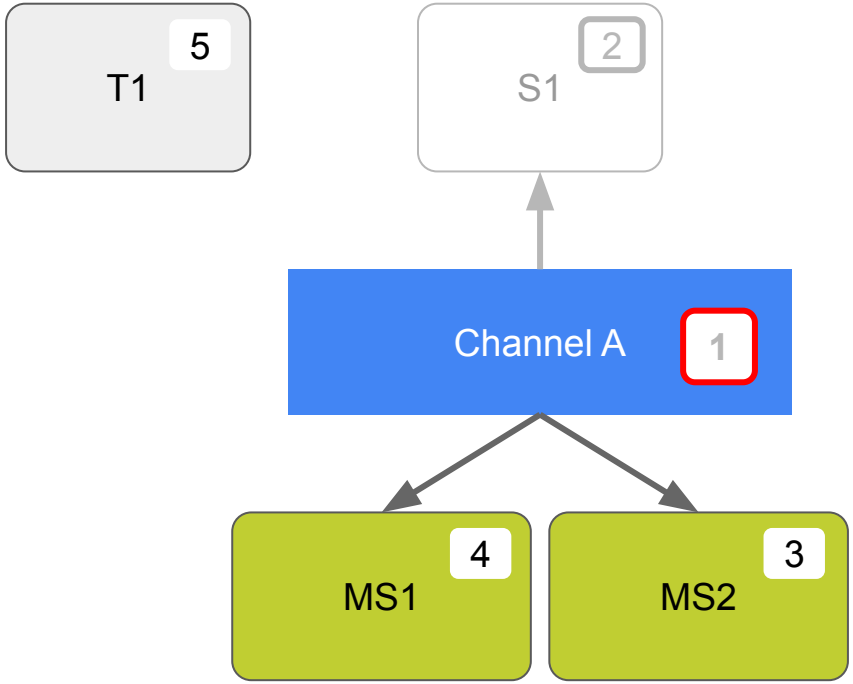
HLP PRIORITY PROTOCOL



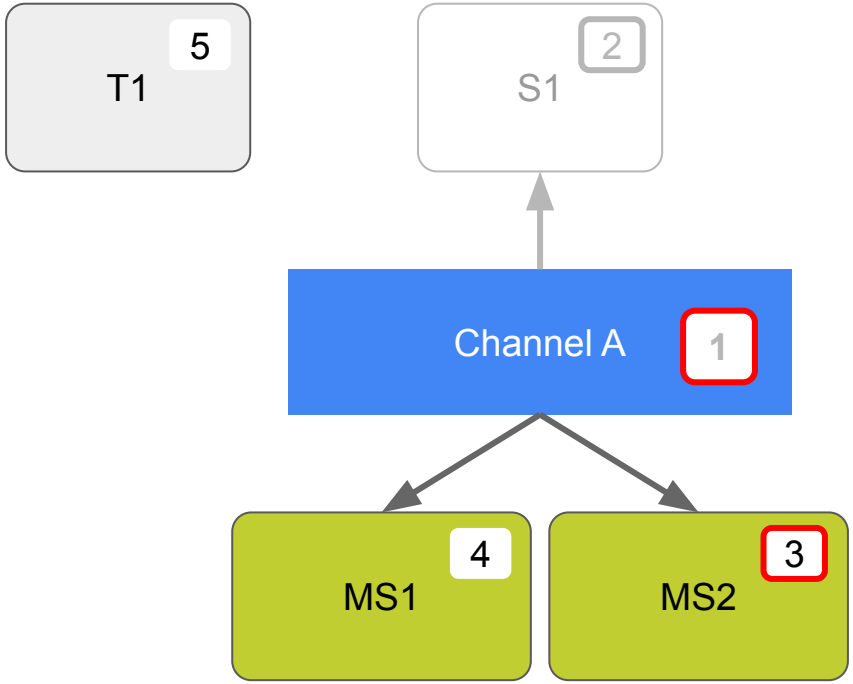
HLP PRIORITY PROTOCOL



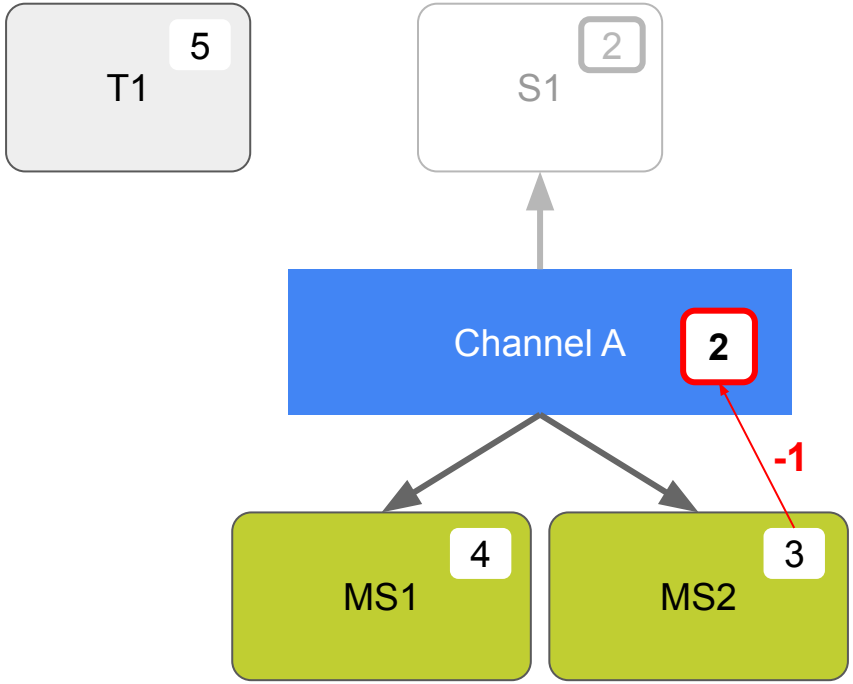
HLP PRIORITY PROTOCOL



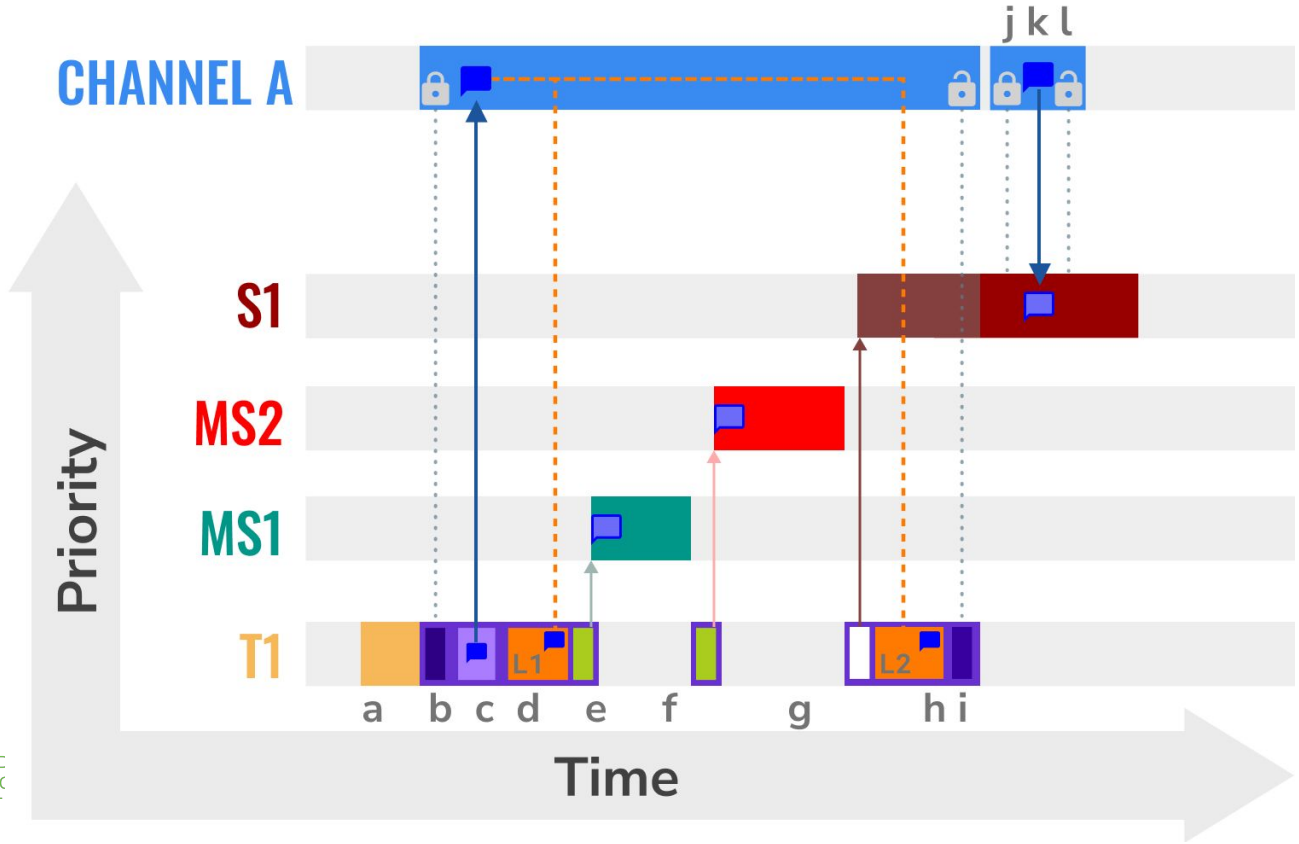
HLP PRIORITY PROTOCOL



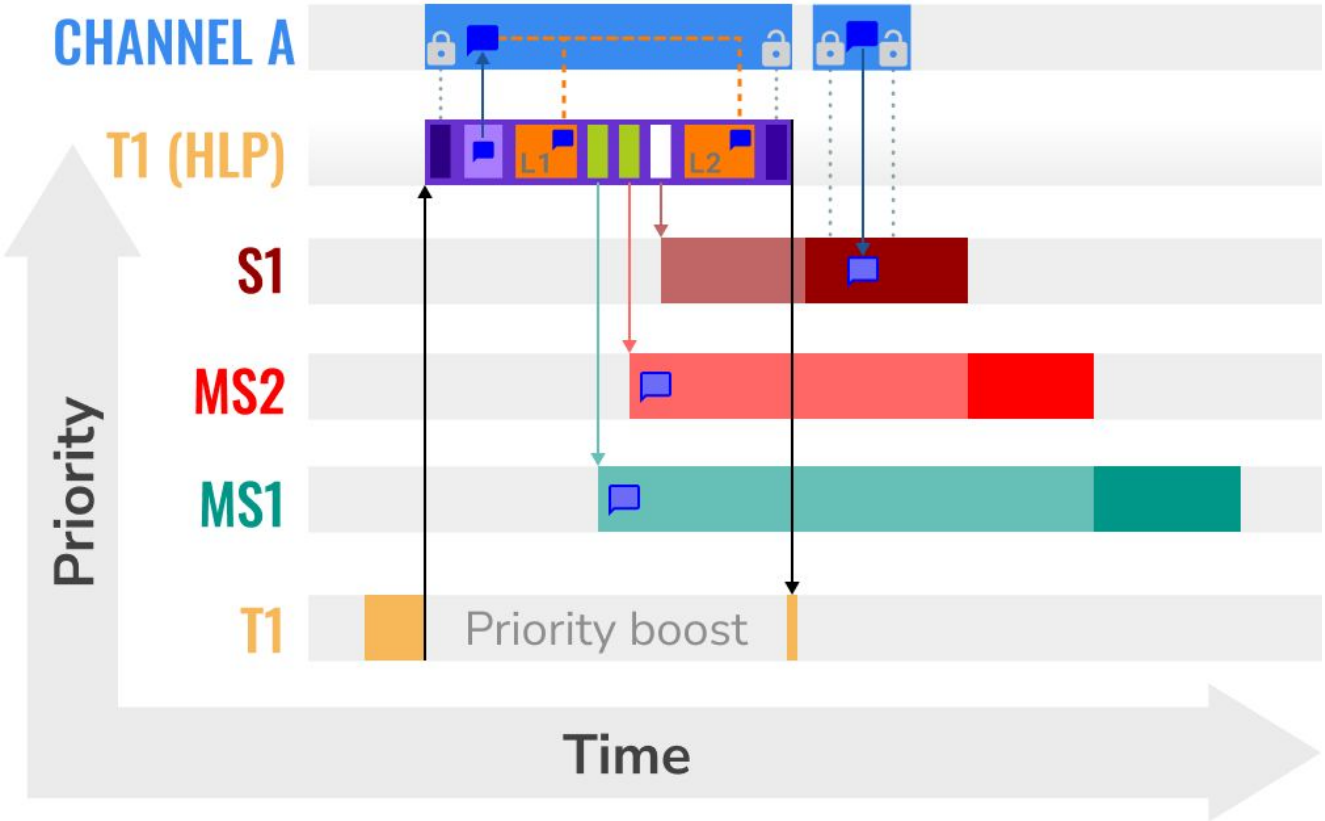
HLP PRIORITY PROTOCOL



PRIORITY INVERSION - SCENARIO



HLP PRIORITY PROTOCOL

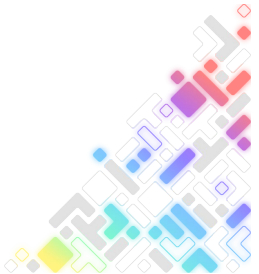


HLP REQUIRES THREAD ATTACHMENT

```c
ZBUS_SUBSCRIBER_DEFINE(s1, 4);
void s1_thread()
{

        const struct zbus_channel *chan;

        zbus_obs_attach_to_thread(&s1);

        while (1) {
                zbus_sub_wait(&s1, &chan, K_FOREVER);

                /* Subscriber implementation */
        }

K_THREAD_DEFINE(s1_id, CONFIG_MAIN_STACK_SIZE, s1_thread, NULL, NULL, NULL, 2, 0, 0);
```

HLP PRIORITY PROTOCOL - SUMMARY

- Short VDED execution
- No priority inversions (bounded or unbounded)
- Listeners before subscribers
- There is almost no overhead if the observer sequence and thread priorities are designed to avoid inversions
- The feature can be disabled (use plain semaphores as lock)
- We can publish to zbus channels inside ISRs! 💪

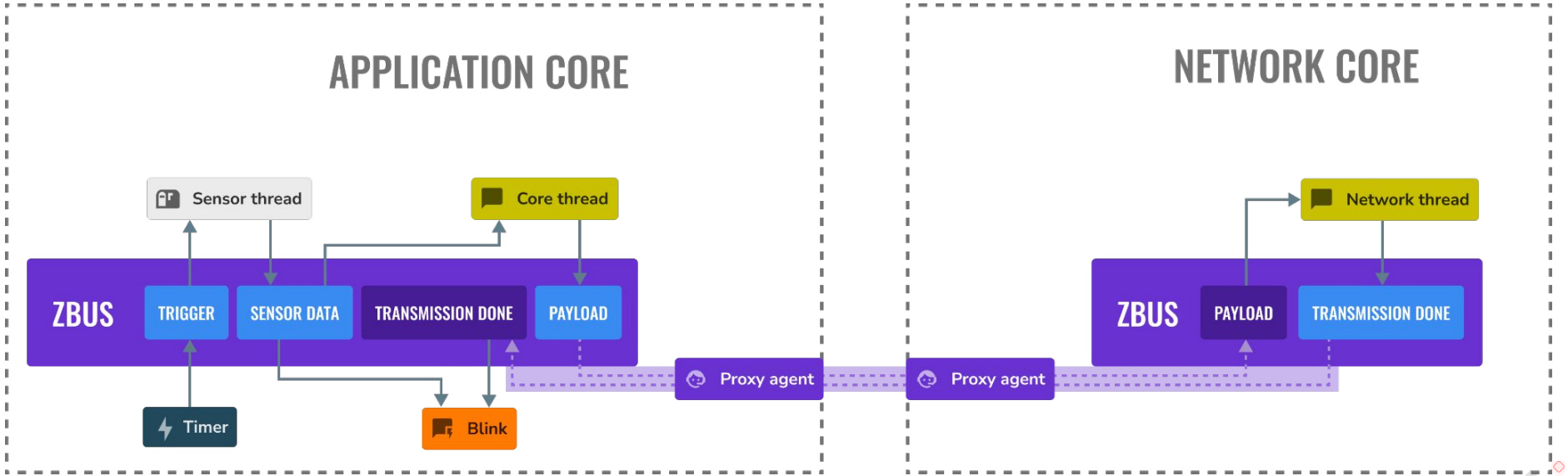


ROADMAP

INDIVIDUAL POOL FOR CRITICAL MSG-SUBSCRIBERS

```c
ZBUS_MSG_SUBSCRIBER_HEAP_POOL_DEFINE(local_heap_msub_pool, /* pool name */
                                     8                     /* pool items count */
);
ZBUS_MSG_SUBSCRIBER_DEFINE_WITH_POOL(my_critical_msub1, &local_heap_msub_pool);

ZBUS_MSG_SUBSCRIBER_STACK_POOL_DEFINE(local_stack_msbu_pool, /* pool name */
                                      8 ,                    /* pool items count */
                                      sizeof(struct item)    /* Item size */
);
ZBUS_MSG_SUBSCRIBER_DEFINE_WITH_POOL(my_critical_msub2, &local_stack_msub_pool);
```

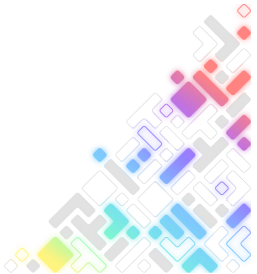
MULTI-CORE AND MULTI-TARGET ZBUS



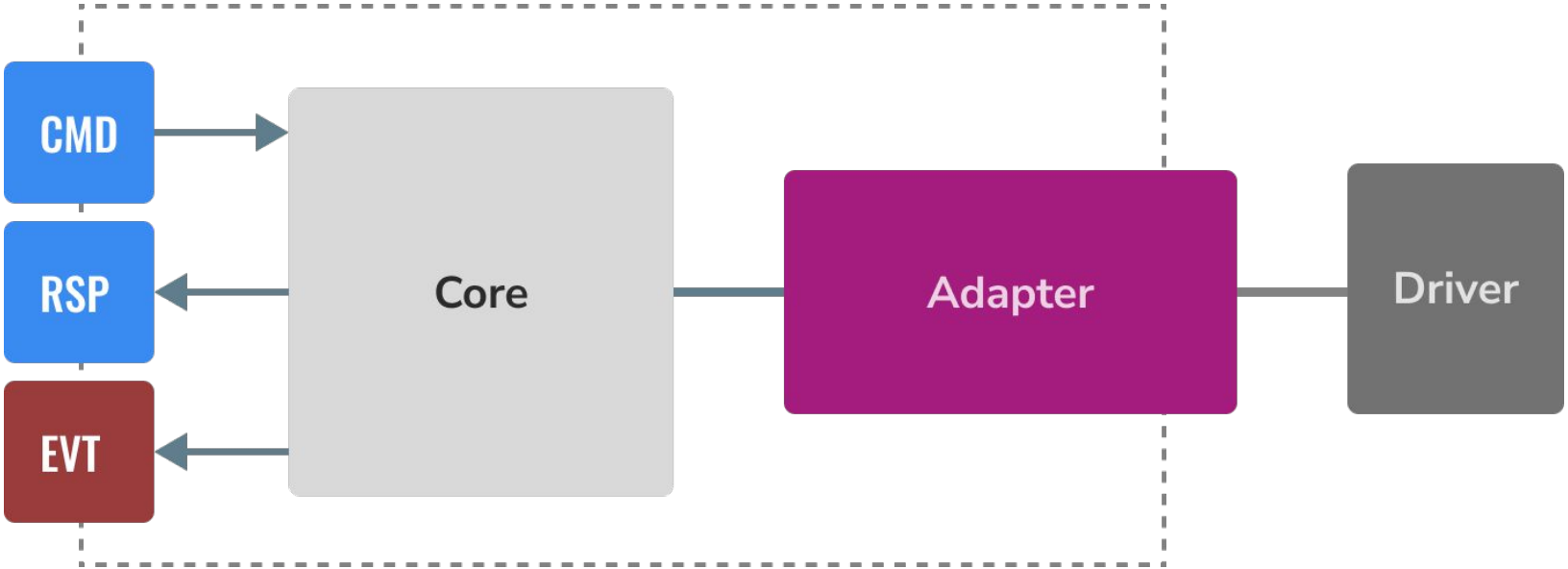
ZEPHYR “DAEMONS”

“A daemon is a service process that runs in the background and supervises the system or provides functionality to other processes.”

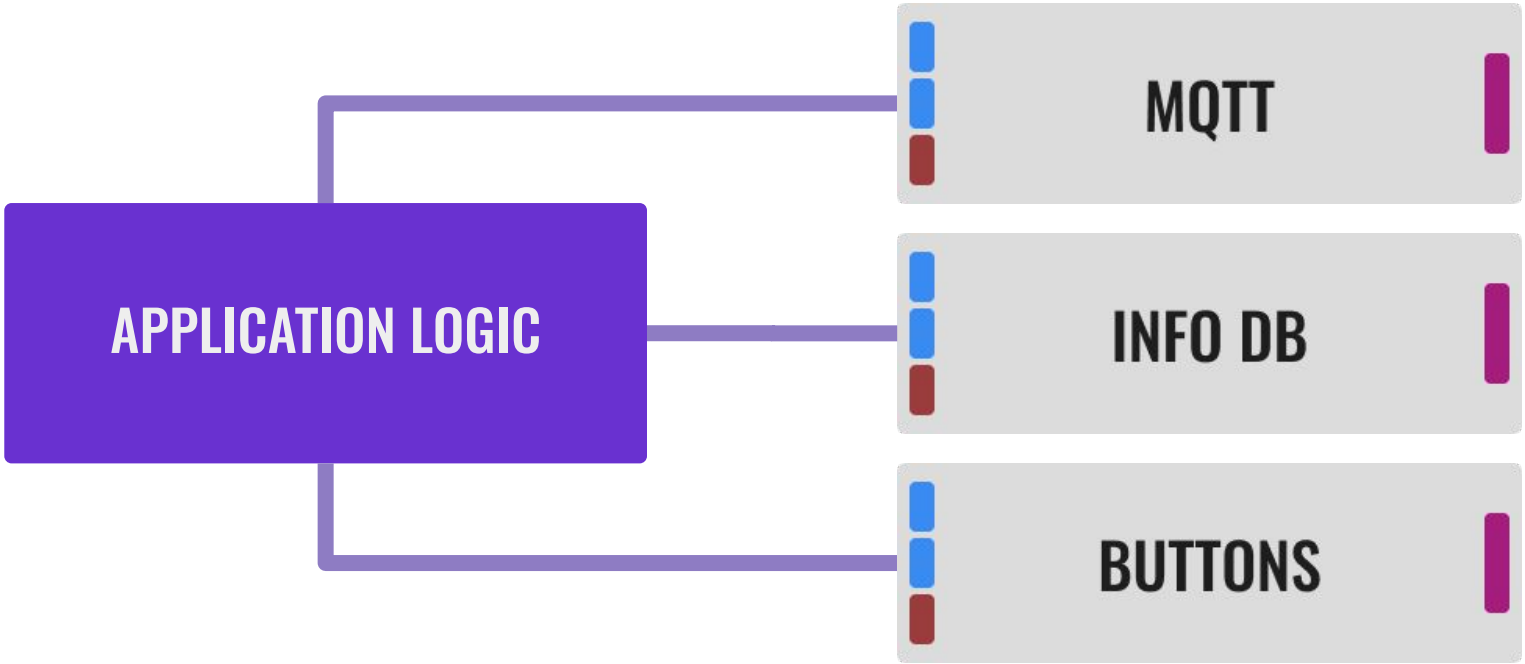
Source *https://man7.org/linux/man-pages/man7/daemon.7.html*



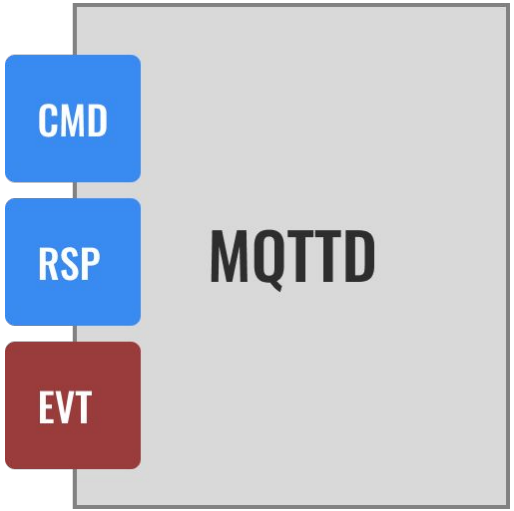
ZEPHYR “DAEMONS”



EXAMPLE SCENARIO

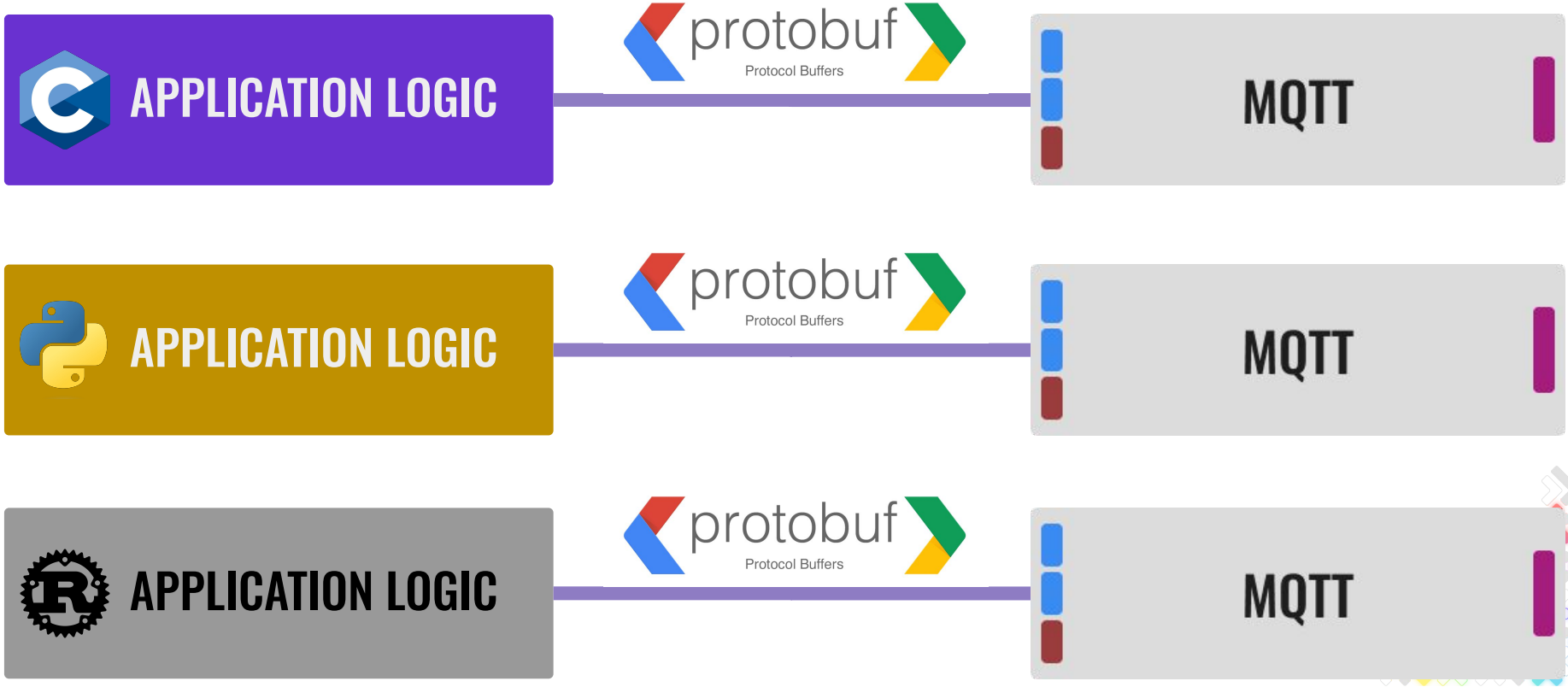


MQTT DAEMON



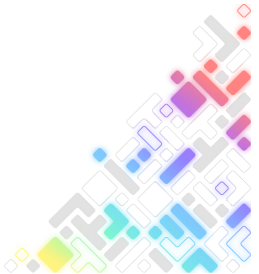

```c
#include <zephyr/daemon/mqttd.h>

int main() {
  struct mqttd_config config = { /* ... */};

  zbus_chan_pub(&mqttd.cmd, &config, K_FOREVER);

  struct mqtt_config_status status;

  zbus_chan_read(&mqttd.rsp, &status, K_NO_WAIT);

  struct mqttd_publish pub_params = {
    .topic = "sometopic/subtopic",
    .payload = buffer,
    .payload_size = sizeof(buffer)
  };

  zbus_chan_pub(&mqttd.cmd, &pub_params, K_FOREVER);

  return 0;
}
```

EXAMPLE SCENARIO



SESSION SUMMARY

- New features
  - Observation storage
  - Message subscribers
  - HLP protocol
- Roadmap
  - Individual pools
  - ZBus Multi-core
  - Daemons



Zephyr® Project

Developer Summit