Cedrus response box and Pyxid

# Aim

The aim of this document is to give the reader some more insight into how the Cedrus response box works and how to interact with it via PsychoPy. Of main interest is the way in which to clear the response buffer before the response registration, so that you can be sure that no previous responses are carried over to the response period. In addition, we also demo how to register bimanual responses.

Many thanks to Jonas Simoens for helping to figure out the code!

# Pyxid module

All the code for interacting between PsychoPy and the Cedrus response box are contained in the [pyxid module](https://www.psychopy.org/api/hardware/cedrus.html) of PsychoPy. Note: as of writing (november 2019) there is a pyxid2 module that replaces the older pyxid module. So make sure that you load in your pyxid module like this to make sure you use the most up to date functions:

| import pyxid2 as pyxid |
| --- |

# Basic Cedrus demo

Before we start with a detailed overview of the interaction between the Cedrus response box and the pyxid module, we recommend having a look at the [Chapter\_5\_Cedrus.py](https://drive.google.com/file/d/1mH71YoM18NoPmTP4Kiaf7Rq-YMEgy1w_/view?usp=sharing) demo script (included in [Chapter 5 on hardware communication](https://sites.google.com/view/pp02psychopy3/chapter-5)).

# How does the interaction between the Cedrus response box and Pyxid work?

## Structure and functions

First of all, the Cedrus has an **internal memory buffer** that contains all the response information that hasn’t been polled by Pyxid yet. In this buffer, presses and releases of keys are stored separately. When this buffer is polled, the oldest item in the buffer is offered up and subsequently cleared from the buffer. The next item in the buffer now becomes the oldest item and will be offered up and cleared when we poll the buffer a second time. If we keep on polling the Cedrus buffer it will eventually become empty (until a response button is pressed by the participant).

Second, the PsychoPy script maintains in its memory a Pyxid **response queue** where we store the items that are polled from the Cedrus buffer. Each item that is polled from the Cedrus buffer is appended to the end of the Pyxid response queue. The command .poll\_for\_response() polls the oldest item in the Cedrus buffer (so it disappears from the buffer) and places it at the end of the Pyxid response queue.

Third, you can inspect the content of the Pyxid response queue one at a time, by moving from the head to the tail of the response queue (so moving from the oldest to the most recent response information). The command .get\_next\_response() gives you the oldest response in the response queue and clears it from the queue. The next item in the response queue now becomes the oldest item and will be offered up and cleared when we execute .get\_next\_response() again.

The entire Pyxid response queue can be cleared with .clear\_response\_queue(). Note that there is no equivalent command to clear the Cedrus buffer entirely. If you want to make sure that you have fully depleted the Cedrus buffer, you’ll have to employ a while loop like the following:

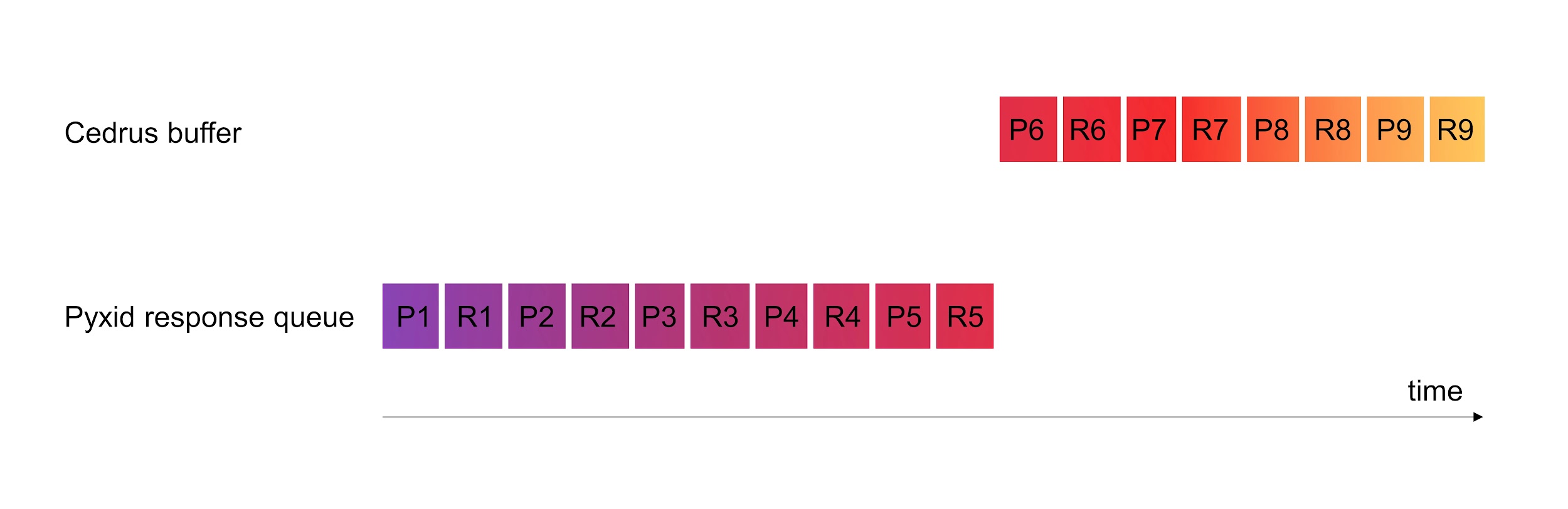
| # Deplete the Cedrus buffer  def clearBuffer(CedrusID):  while True:  CedrusID.poll\_for\_response()  if CedrusID.response\_queue\_size() == 0: break  if CedrusID.response\_queue\_size() > 0: CedrusID.clear\_response\_queue() |
| --- |

With this clearBuffer() function, we continuously poll the Cedrus buffer until all responses have been transferred to the Pyxid response queue and the queue stops growing. Finally, the Pyxid response queue is cleared so we are ready to register the actual response.

Note that you can use the command .response\_queue\_size() to inspect the number of items in the Pyxid response queue. This allows us to verify whether new response information is still being added to the Pyxid response queue by executing the .poll\_for\_response() command. There is however no equivalent code to know the number of items in the Cedrus buffer, which is why we need the while loop to continuously poll the Cedrus buffer until all items have been transferred to the Pyxid response queue. This is the only way we can detect that the Cedrus buffer has been fully emptied.

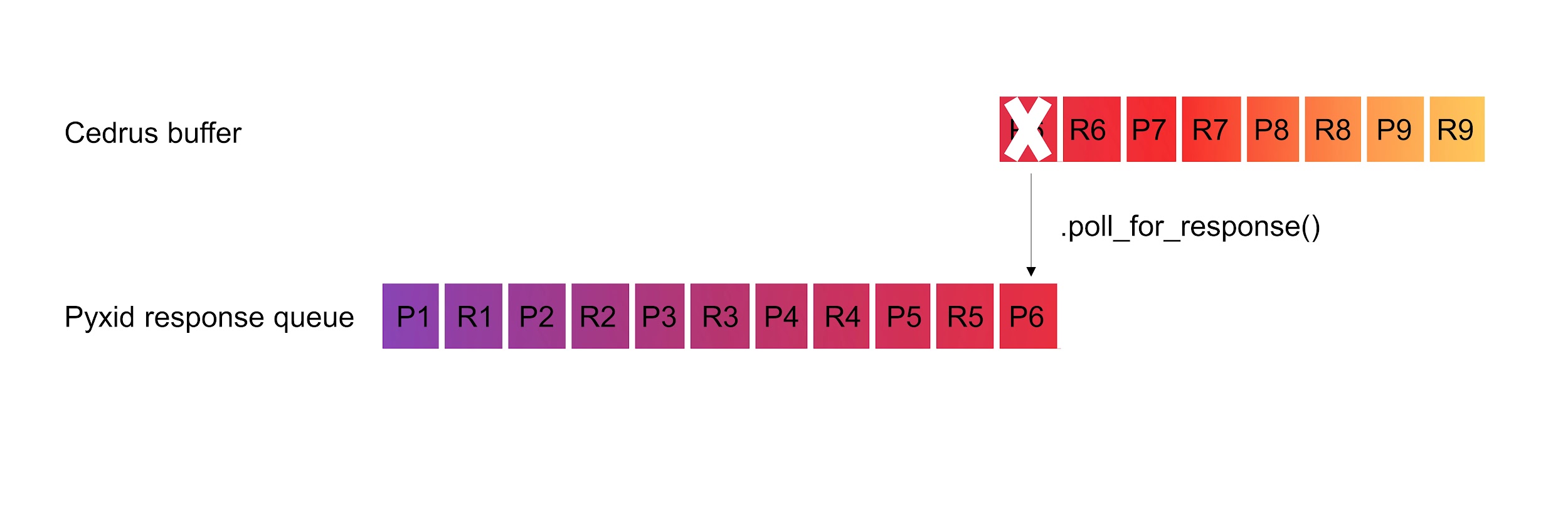
## Illustration

Suppose that a participant made 9 button presses since the start of the experiment. This means that in total 2\*9 response events have occurred: 9 presses (P) and 9 releases (R). We have already polled the Cedrus buffer 10 times, placing the full information on the first 5 reactions in the Pyxid response queue. This leaves us with the presses and releases of the 4 last responses still in the Cedrus buffer. This is illustrated schematically below:

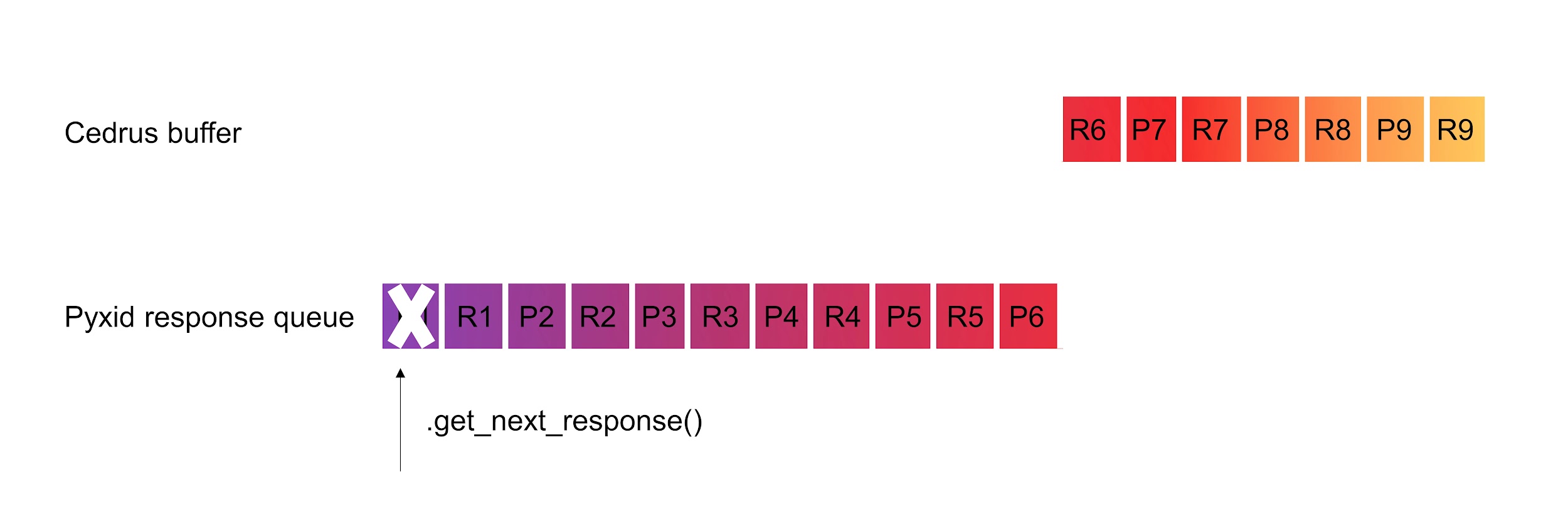


At this point, executing the command .response\_queue\_size() would inform us that there are 10 items in the Pyxid response queue.

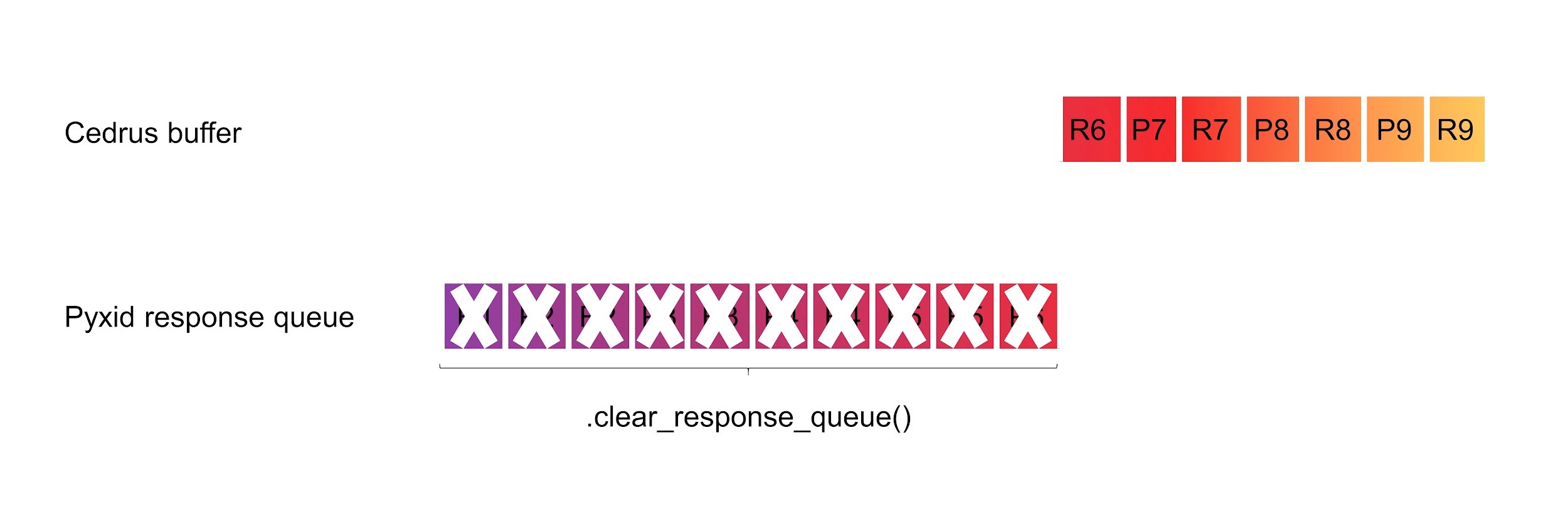
If we now execute the .poll\_for\_response() command, the press for the 6th response will be moved from the Cedrus buffer to the Pyxid response queue. Note that this action removes P6 from the Cedrus buffer. This is illustrated schematically below. If we would execute the .poll\_for\_response() command 7 more times, we would have completely depleted the Cedrus buffer and moved all the information to the Pyxid response queue.



At this point, executing the command .response\_queue\_size() would inform us that there are 11 items in the Pyxid response queue. If we want to process the items from the Pyxid response queue, we use the .get\_next\_response() command, which gives us the first item from the response queue. In this case this will be P1, as we assume that we haven’t been sampling the response queue yet. This is illustrated schematically below. Note that when we sample the response queue, the sampled item is immediately removed from the response queue.



Finally, if we want to remove all the information from the response queue, we use the command .clear\_response\_queue() which removes all the remaining items in the queue in one fell swoop. This is illustrated schematically below.



Note that there is no equivalent command to clear the Cedrus buffer entirely. If you want to make sure that you have fully depleted the Cedrus buffer, use the clearBuffer() function we mentioned above.

## Demo code

In the demo code [Chapter\_5\_Cedrus\_spurious\_responses.py](https://drive.google.com/file/d/1O9oNq8-VAfYPcZza2MQ9s0QYKCttr3C3/view?usp=sharing) we demonstrate the use of the clearBuffer() function in a very simple experiment. The participant is given two seconds to generate spurious responses before the actual response registration starts. By commenting out the clearBuffer() function, you can see how failing to empty the Cedrus buffer causes the experiment to proceed based on responses that were previously made.

# Bimanual responses

In some experiments you might require the participant to make a bimanual response, for instance by pushing with the two index fingers versus pushing with the two middle fingers. Some code to illustrate how this can be done can be found in the [Chapter\_5\_Cedrus\_bimanual.py](https://drive.google.com/file/d/1krijiqWmrozHtP4n47VZgX4_y9jQNxgJ/view?usp=sharing) demo script.