Continuous Buffers in the HLT Data Transport

David Rohr, drohr@cern.ch

Frankfurt Institute for Advanced Studies

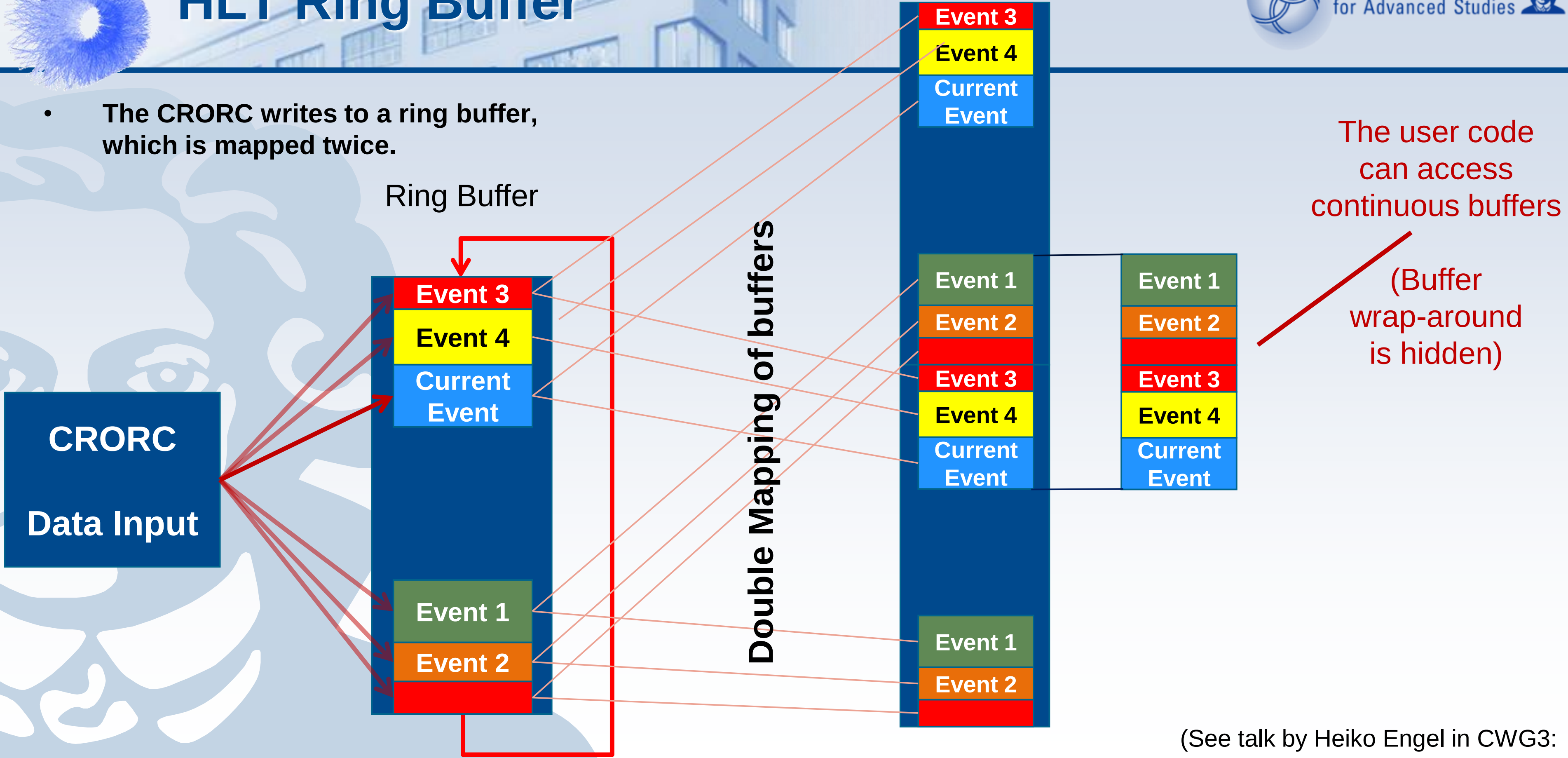
CERN, 11.2.2016

HLT PCIe Data Input Buffers

- **The HLT data transport frameworks uses shared buffers to transfer data between processing components.**
  - SYSV Shared Buffer for transfer between two processing components.
  - libPDA CRORC Buffers for the transfer from the CRORC to the first component.
- **All buffers are contiguous, i.e. there are no pages.**
  - Components may opt to create multiple output data buffers if desired, but there is no general limitation of the buffer size.
  - In physical memory, the buffers are not continuous, but in user-space they are.
- **libPDA has different methods of buffer access:**
  - It can create kernel-space buffers itself, and map them into user-space. (Currently done in the HLT)
  - It can take existing user-space shared POSIX buffers and register them for the PCIe device. (Planned for CBM)
    - The second option is forseen to have common buffers for Infiniband and Data Read Our Receiver.
- **Everything is transparent to the user code in the processing component:**
  - The component does not know whether its input buffer is filled by the CRORC or by another component.
  - The buffer is always continuous, the component does not have to bother with buffer pages → simplifies data access.

HLT Ring Buffer

- The CRORC writes to a ring buffer, which is mapped twice.



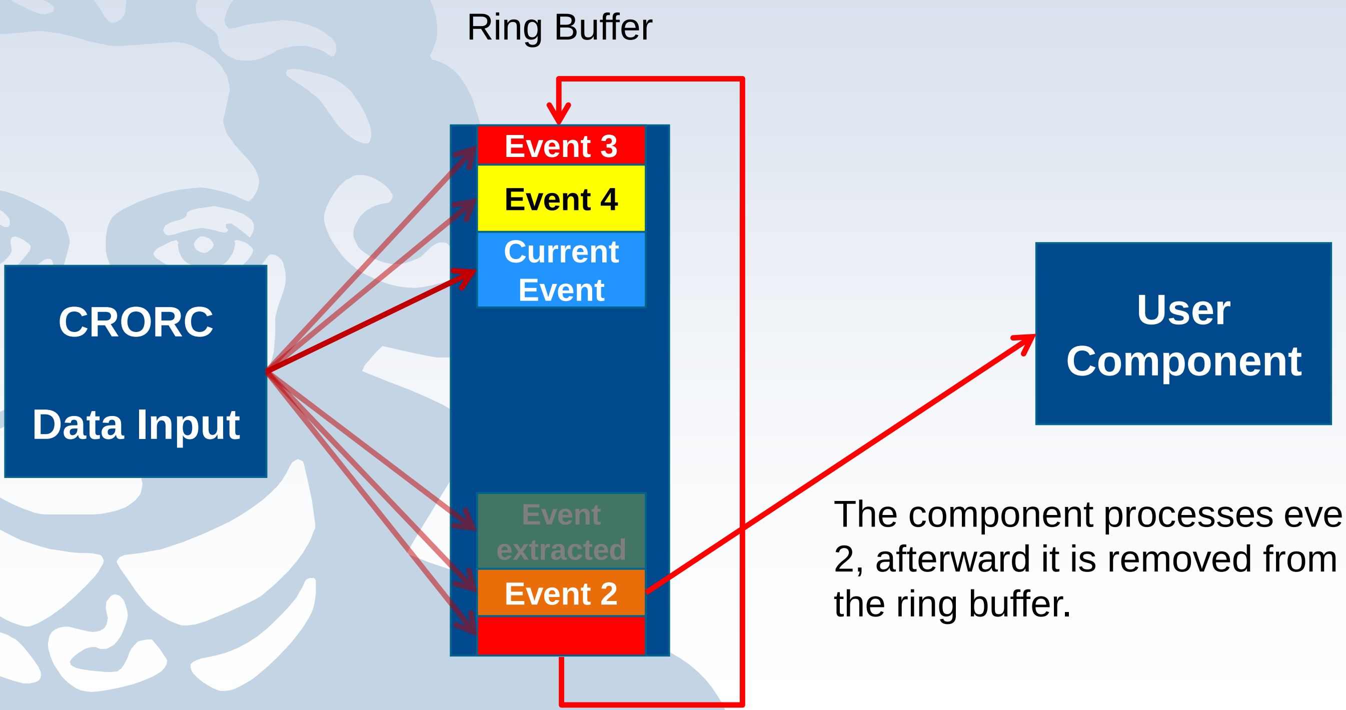
The user code can access continuous buffers

(Buffer wrap-around is hidden)

(See talk by Heiko Engel in CWG3:
https://indico.cern.ch/event/485802/contribution/5/attachments/1215705/1775226/PDA.pdf)

HLT Ring Buffer

- The user code extracts the data from the ring buffer in the same order as it is inserted.



The component processes event 2, afterward it is removed from the ring buffer.

Ring Buffer Limitation

- **The ring buffer poses a limitation, when the events are not removed fast enough.**
- **In particular, when the later events are removed in the same order they are inserted.**
  - We remove the events in the order we insert them.
  - But: In parallel.
    - → Hence, if there is one big event, and then many small events, the small events are removed faster.
    - → The big event stays in the buffer, and could possibly block it.

- **In reality, we do not have this problem:**
  - Our FEP (Input nodes) have as much memory as the processing nodes.
    - → We can make the buffer really large, which will hide the problem (in worst case sacrificing some processing capabilities on the input nodes).
  - If a single ring-buffer is insufficient, one could a more elaborate construct like a multi-ring buffer.

- **Finally, in the HLT we have 1-2 GB buffer size per DDL, and it is well sufficient, so it uses only 10% of the memory.**
- **We did not have to spend any additional effort to make sure the buffers are sufficient.**