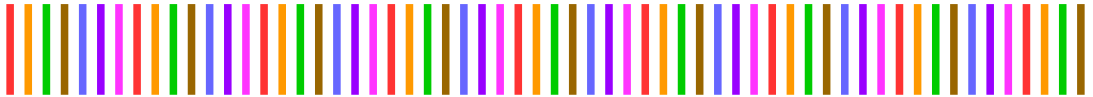


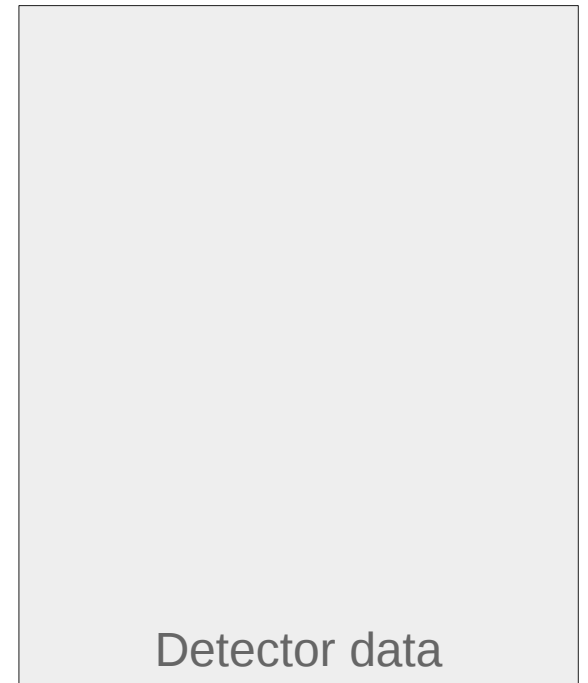
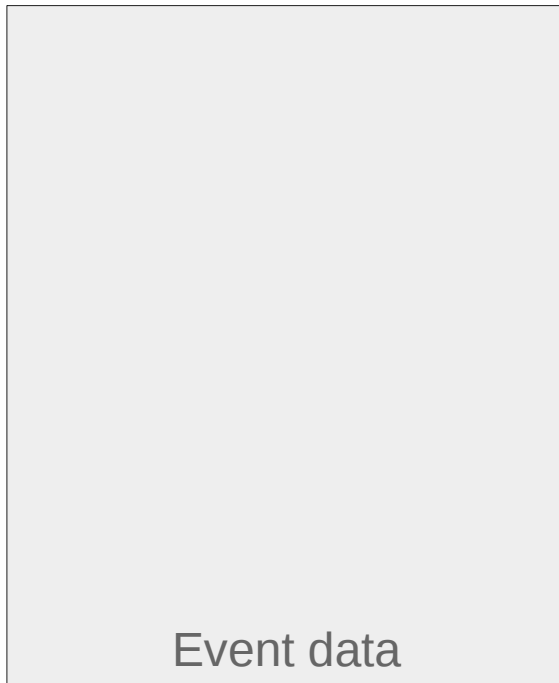
Multi-threaded condition handling:

Easing the upgrade path

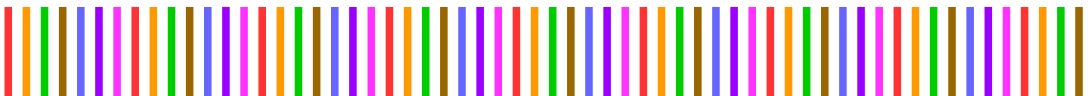
Hadrien Grasland

Sequential event processing

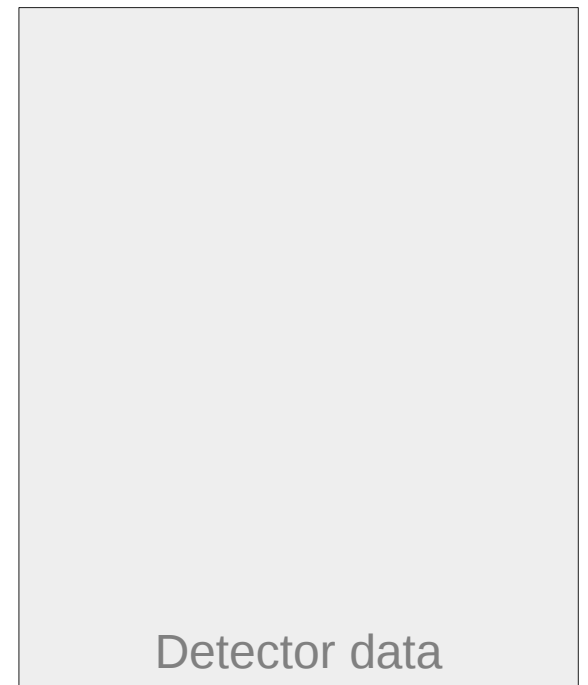
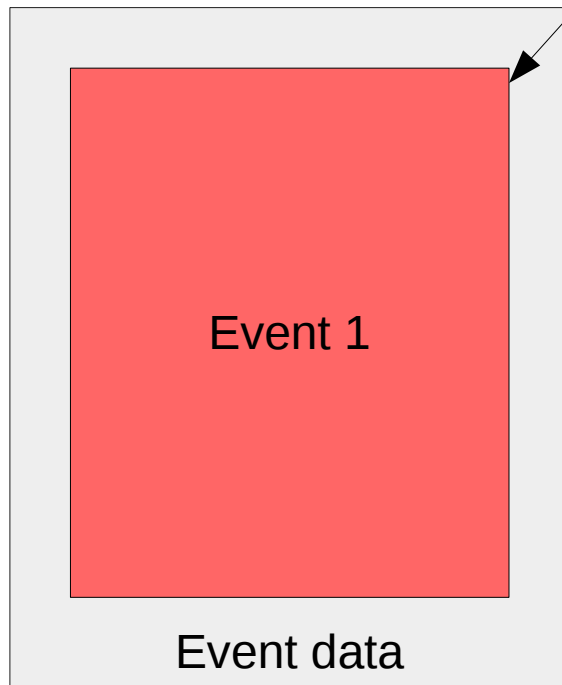
Input event stream 



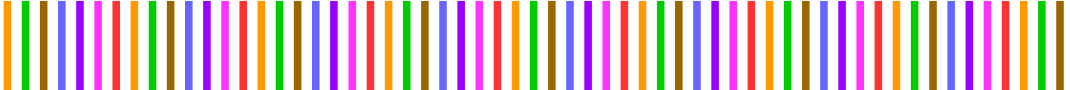
Sequential event processing

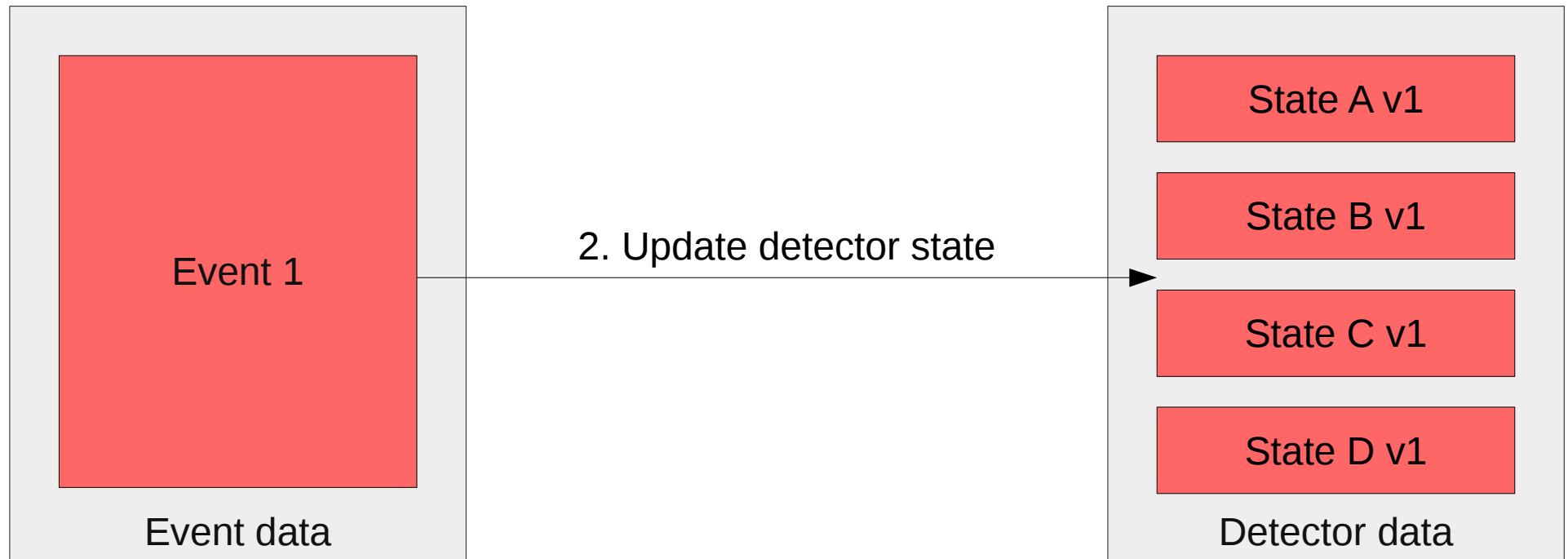
Input event stream 

1. Load an input event

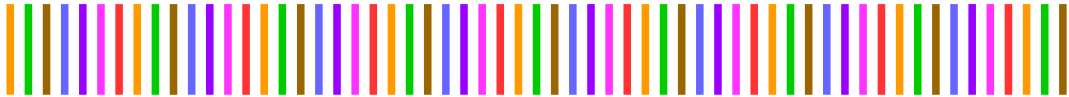


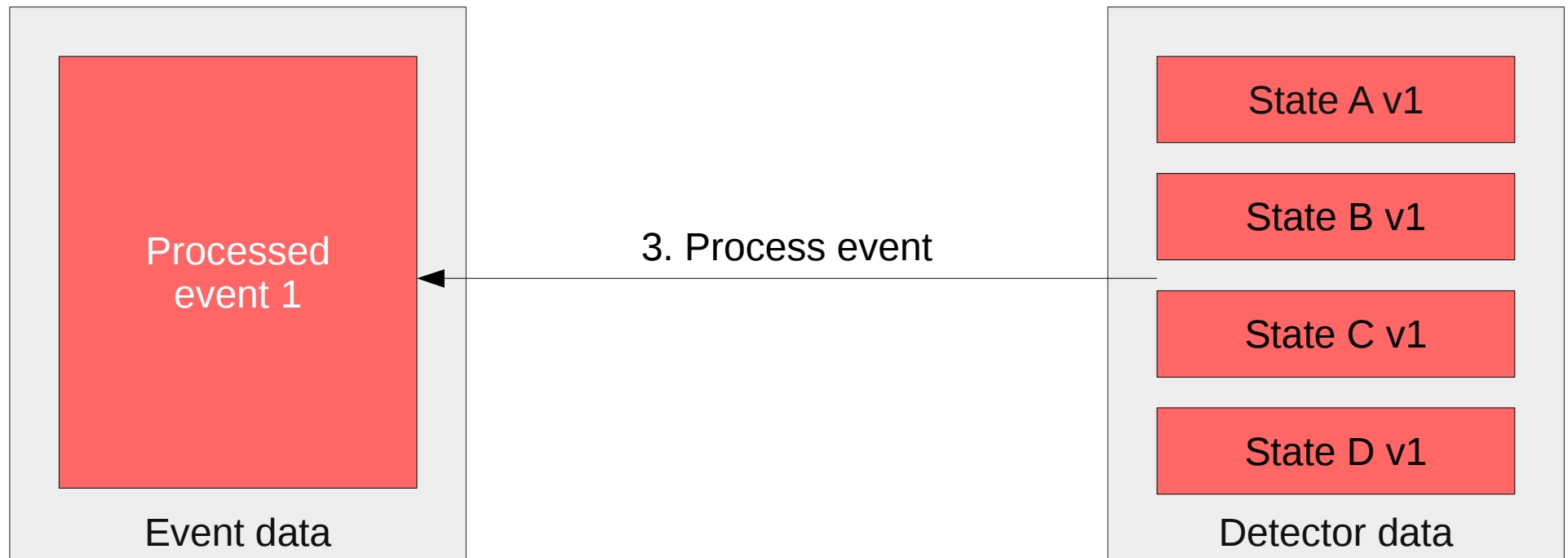
Sequential event processing

Input event stream 

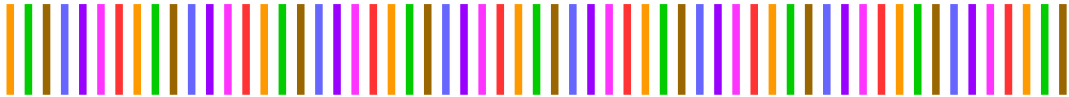


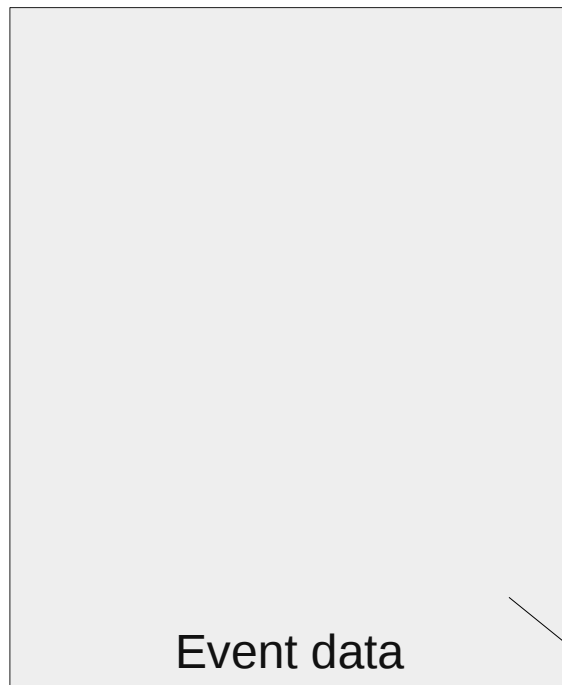
Sequential event processing

Input event stream 

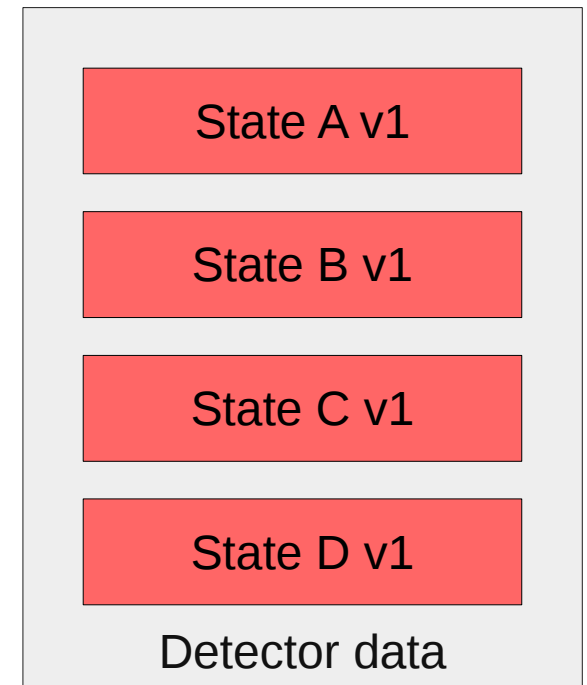


Sequential event processing

Input event stream 

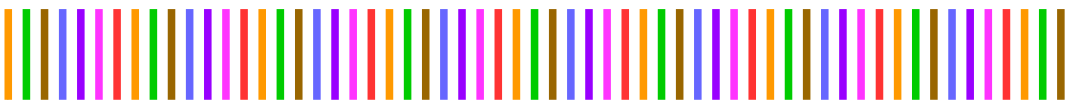


4. Output results to disk

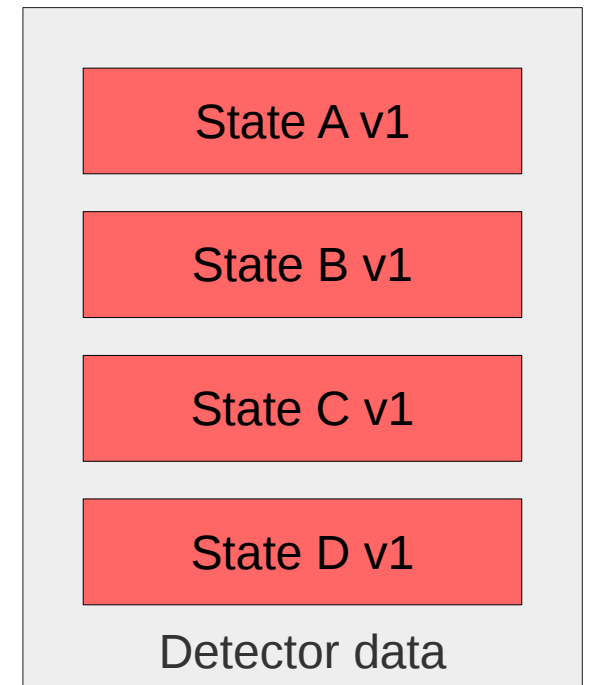
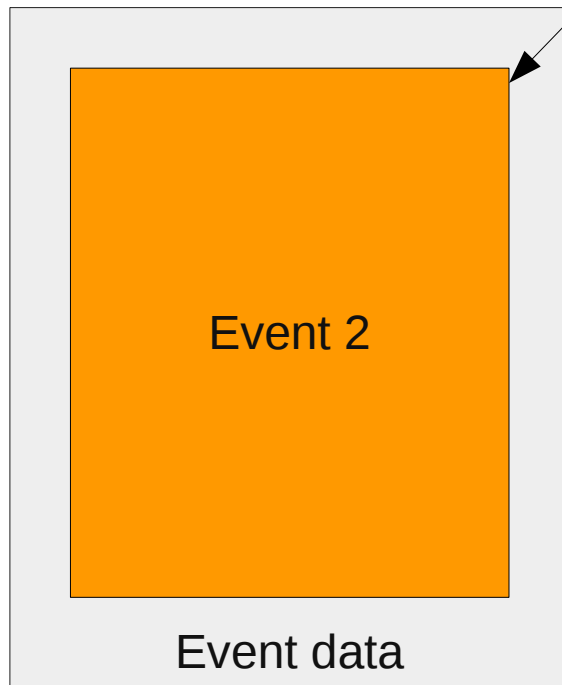


Sequential event processing

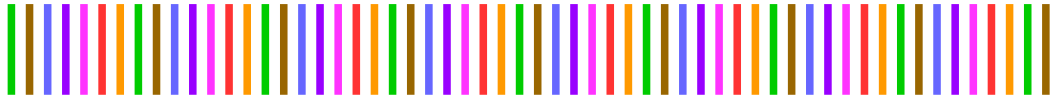
Input event stream

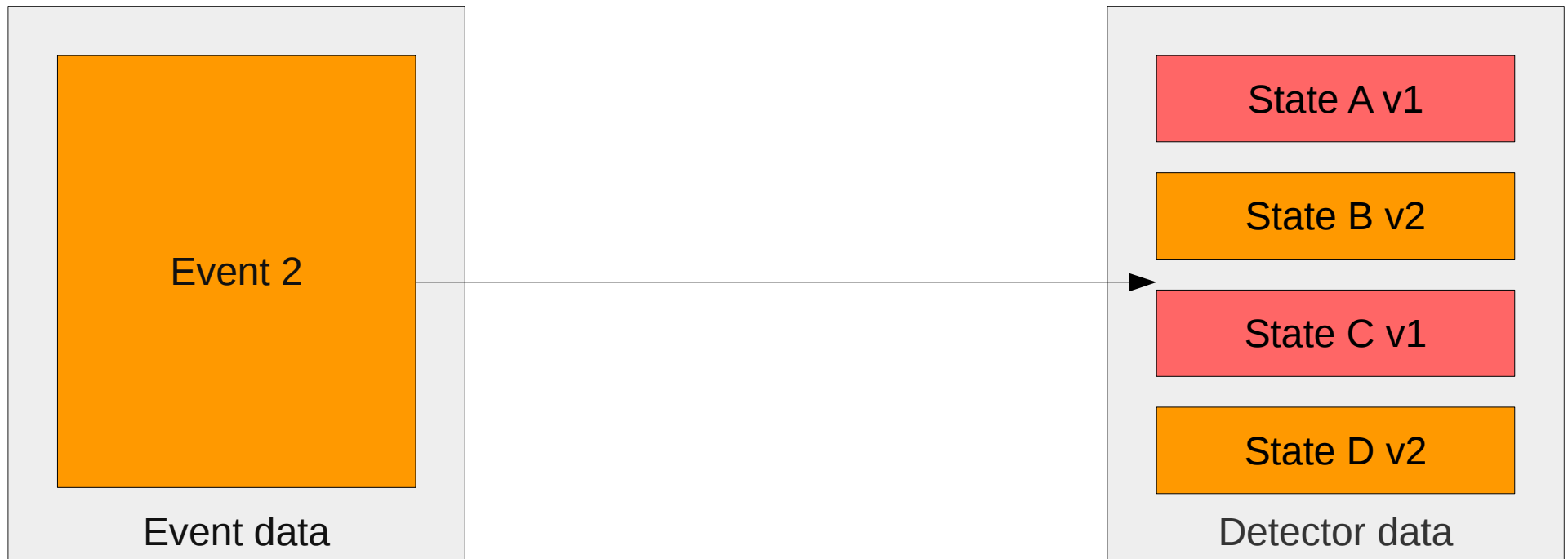


6. Repeat

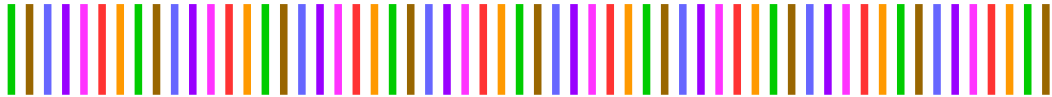


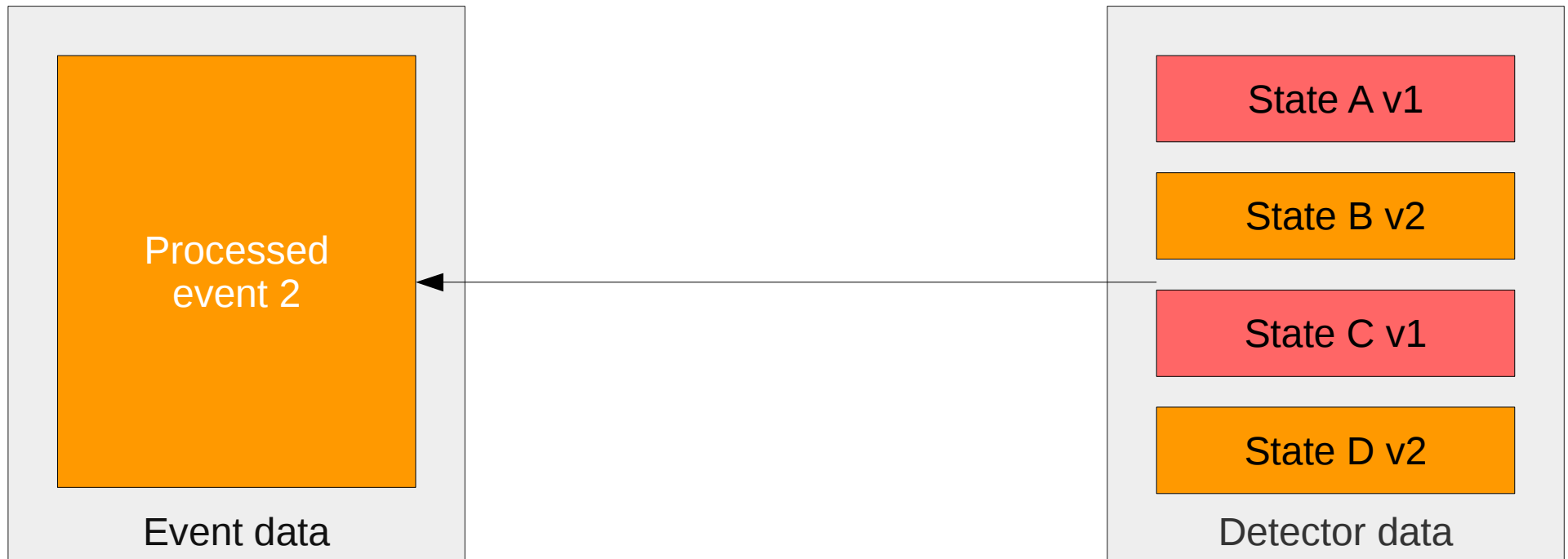
Sequential event processing

Input event stream 

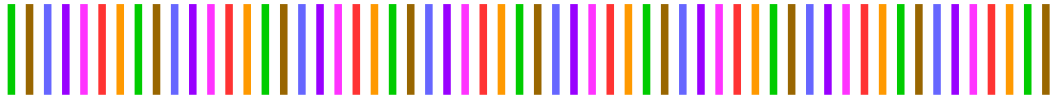


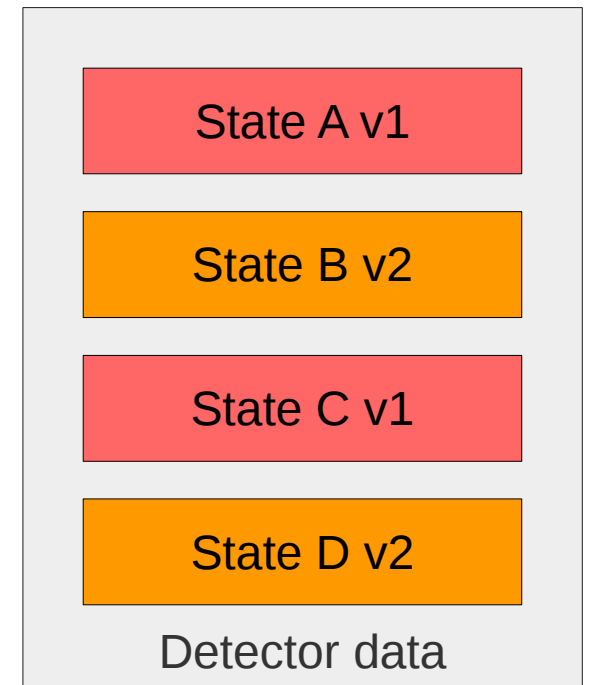
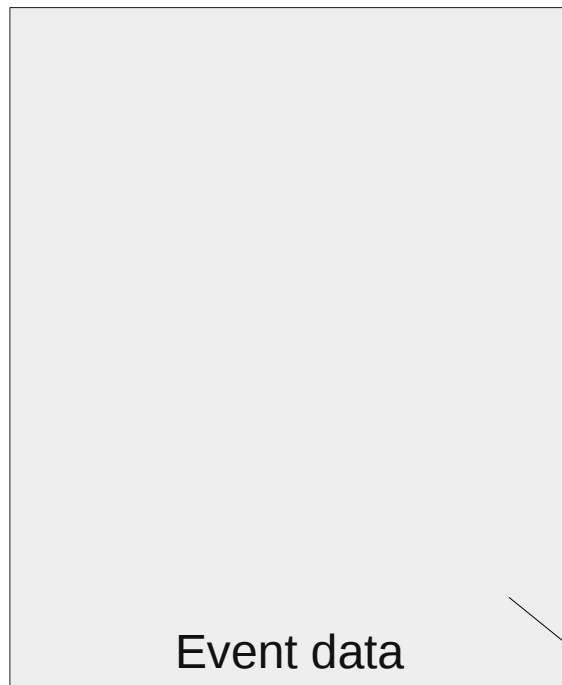
Sequential event processing

Input event stream 



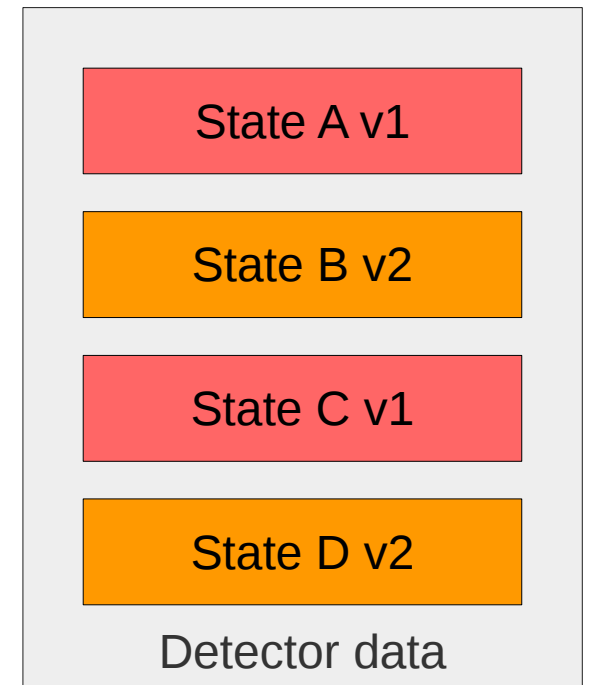
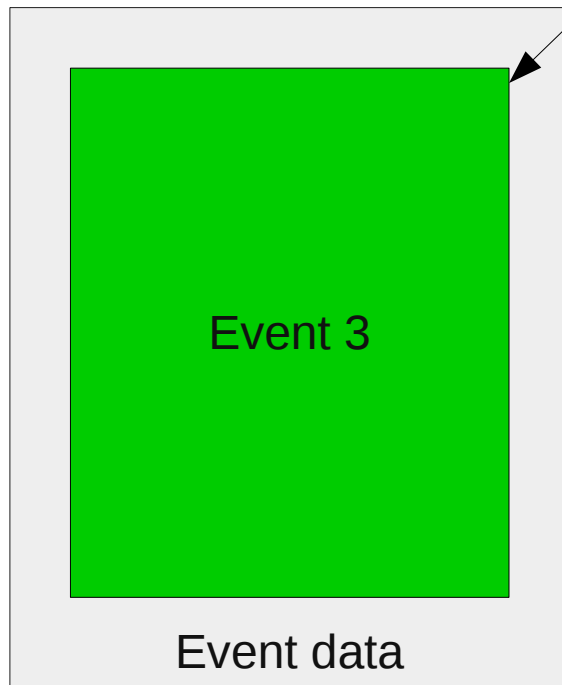
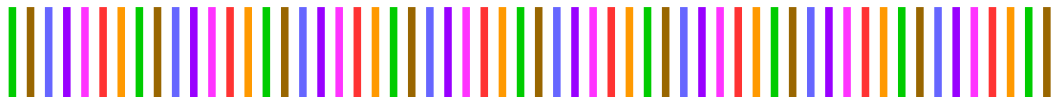
Sequential event processing

Input event stream 



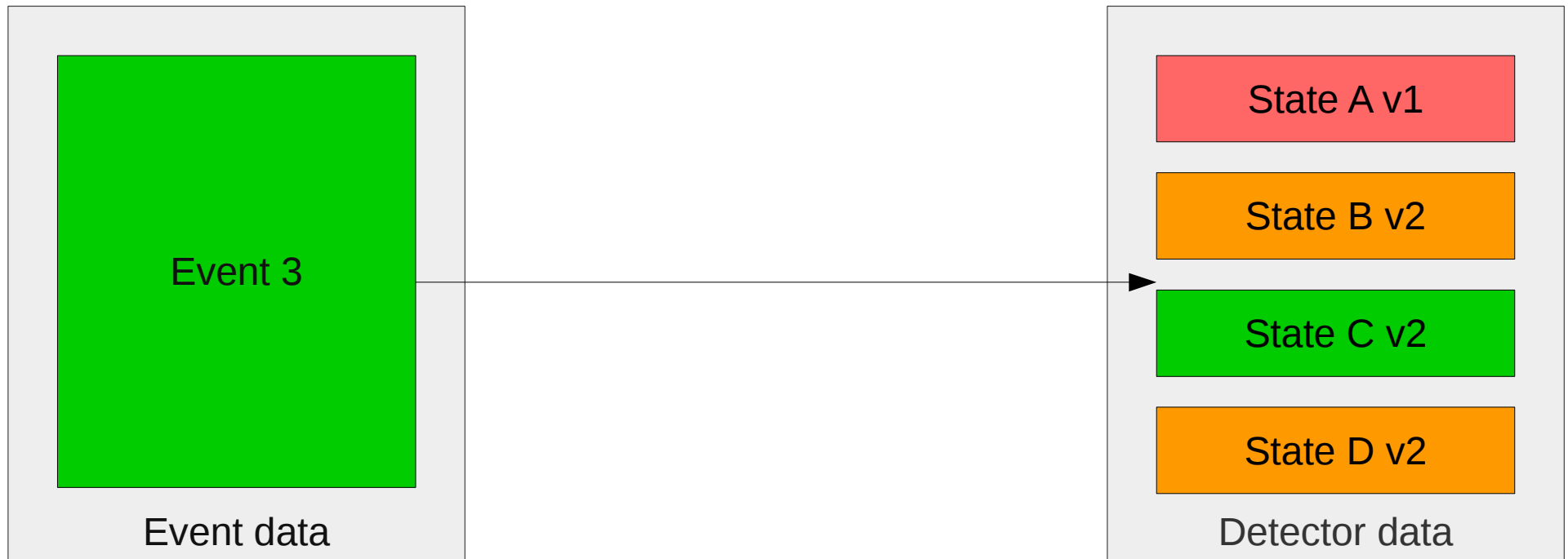
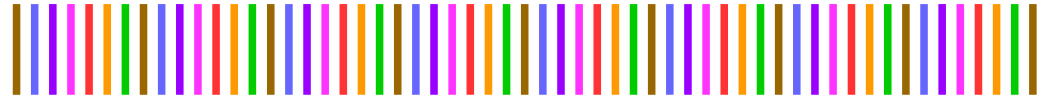
Sequential event processing

Input event stream

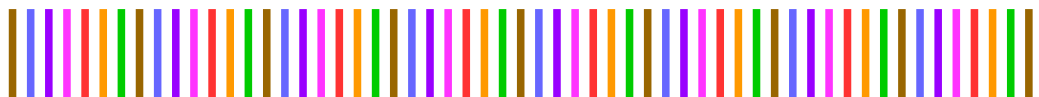


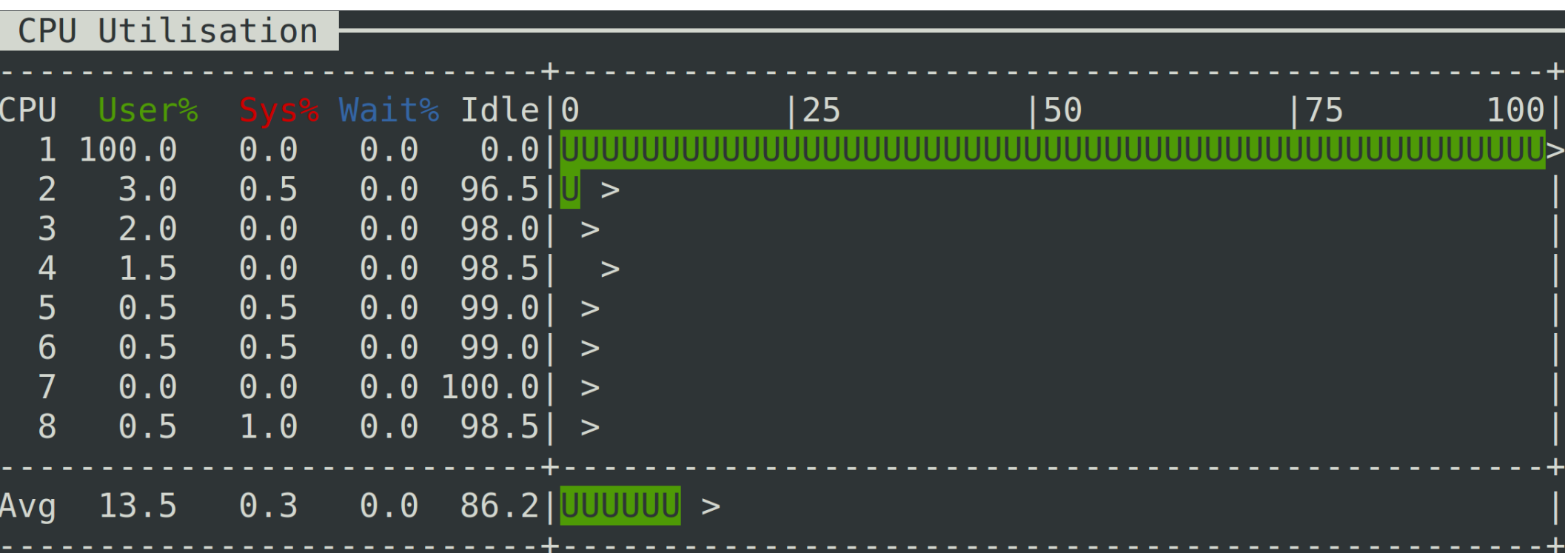
Sequential event processing

Input event stream

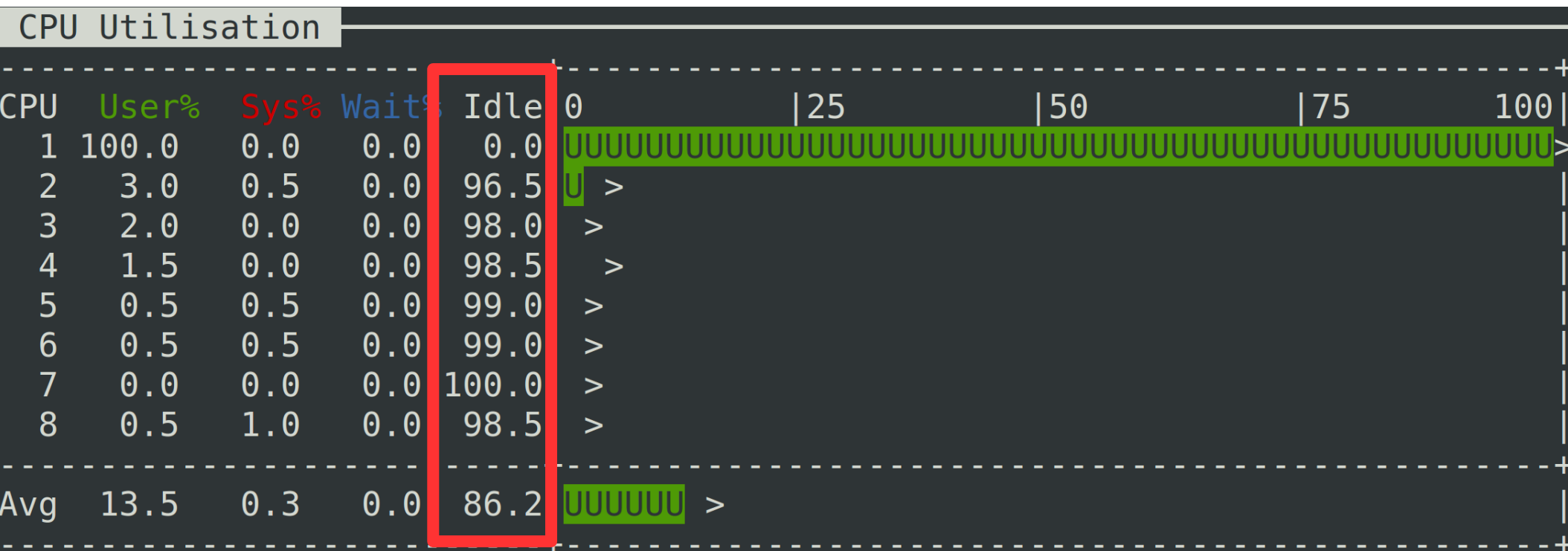
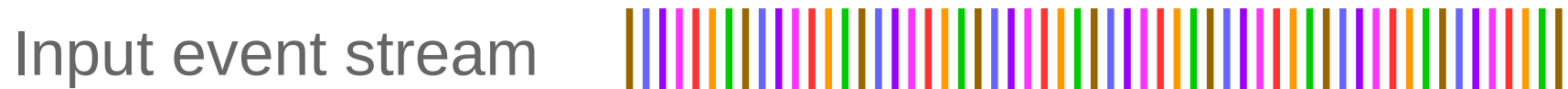


Sequential event processing

Input event stream 



Sequential event processing



How to use more cores?

- Spawn multiple framework processes?
 - See computing grid, GaudiMP, basf2 (Belle II)
 - Minimizes upgrade complexity
 - **BUT** too RAM-hungry for new hardware!

How to use more cores?

- Spawn multiple framework processes?
 - See computing grid, GaudiMP, basf2 (Belle II)
 - Minimizes upgrade complexity
 - **BUT** too RAM-hungry for new hardware!
- Move towards multithreading
 - Demonstrated in GaudiHive
 - Thread-safe components required :-/

Thread-safety

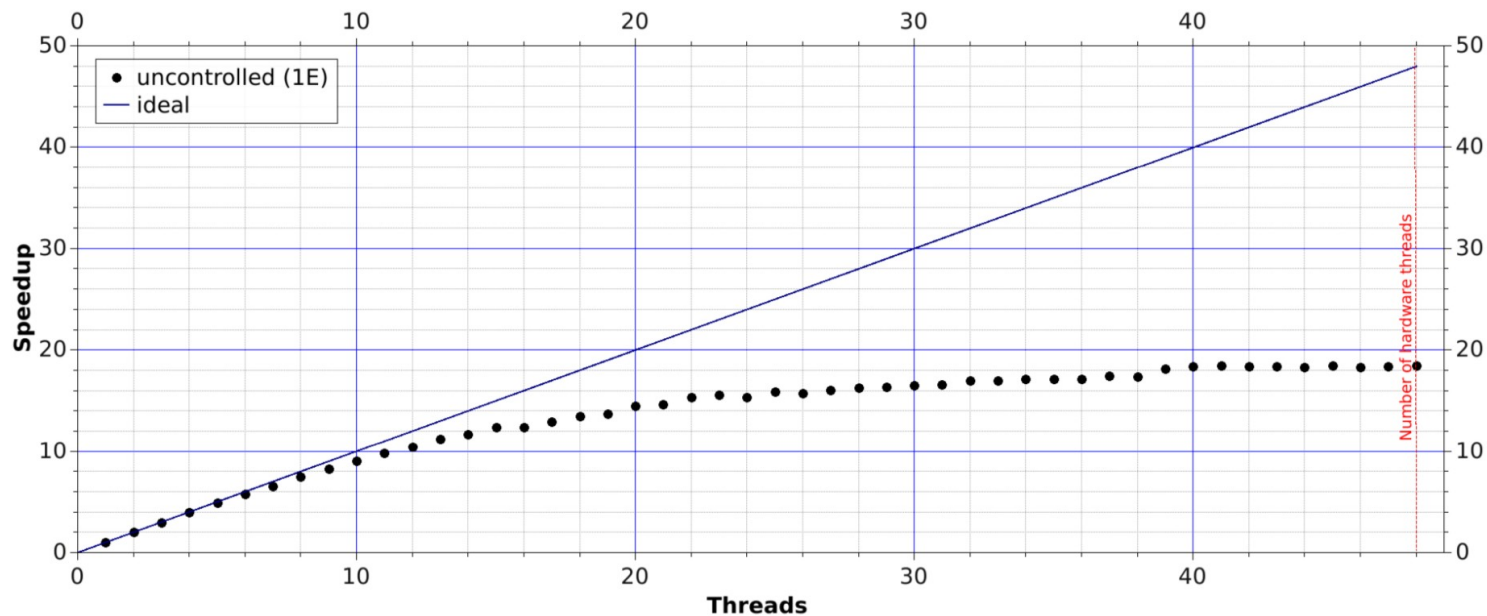
- Mostly about avoiding data races
 - Read-after-write, write-after-read, write-after-write
 - Hard to reproduce, hard to debug

Thread-safety

- Mostly about avoiding data races
 - Read-after-write, write-after-read, write-after-write
 - Hard to reproduce, hard to debug
- Several ways to achieve correctness
 - Locks (with overhead, contention, deadlocks...)
 - Lock-free & wait-free algorithms (experimental)
 - **Keep data private or read-only when possible!**

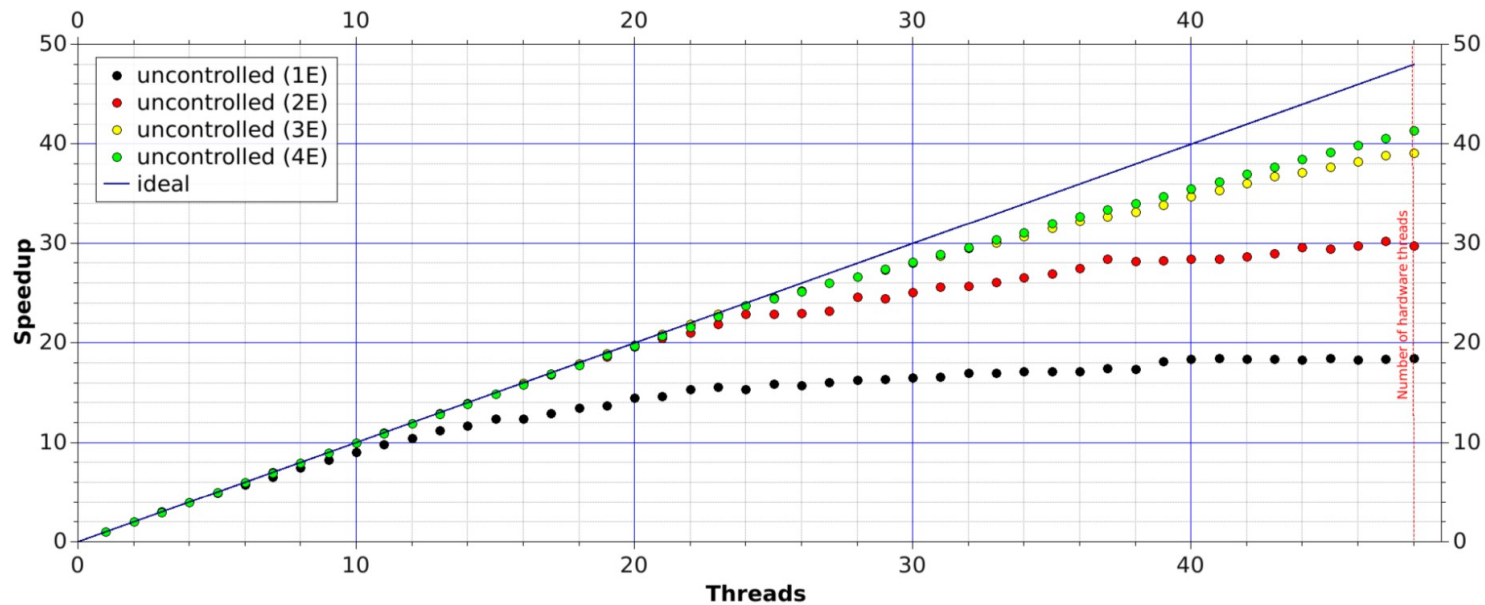
The detector state issue

- For optimal performance, processing one event with multiple threads is not sufficient



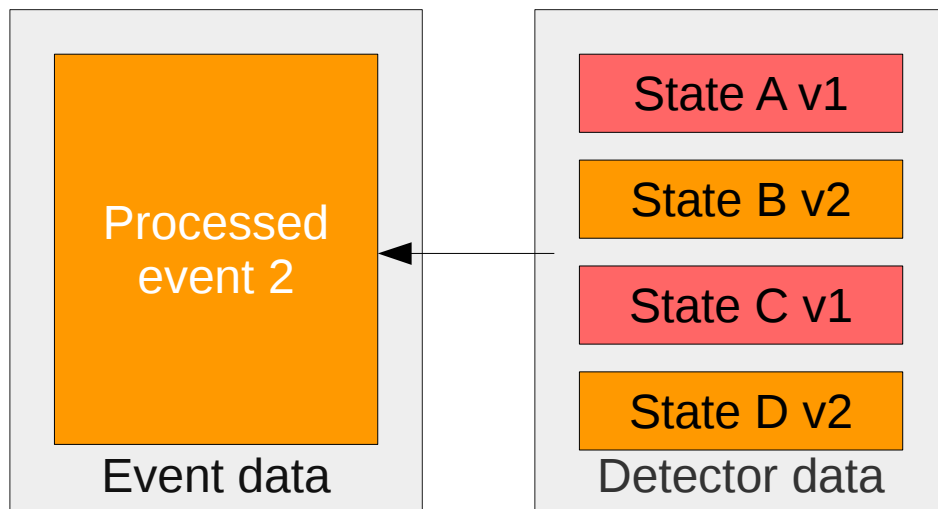
The detector state issue

- For optimal performance, processing one event with multiple threads is not sufficient
 - Need to process multiple events at once



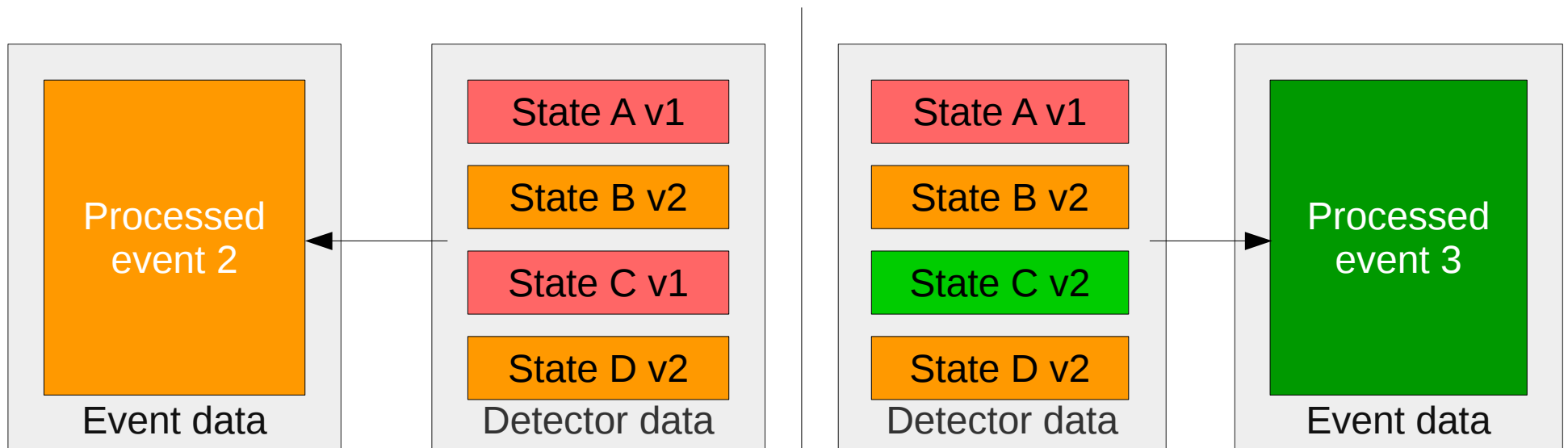
The detector state issue

- For optimal performance, processing one event with multiple threads is not sufficient
 - Need to process multiple events at once
 - Detector state changes are problematic



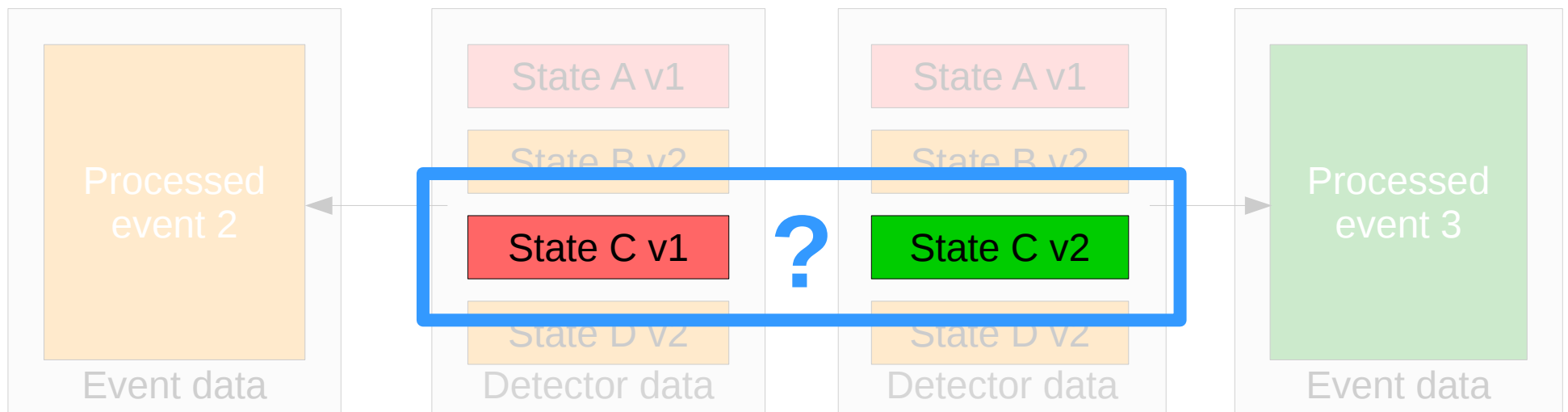
The detector state issue

- For optimal performance, processing one event with multiple threads is not sufficient
 - Need to process multiple events at once
 - Detector state changes are problematic



The detector state issue

- For optimal performance, processing one event with multiple threads is not sufficient
 - Need to process multiple events at once
 - Detector state changes are problematic

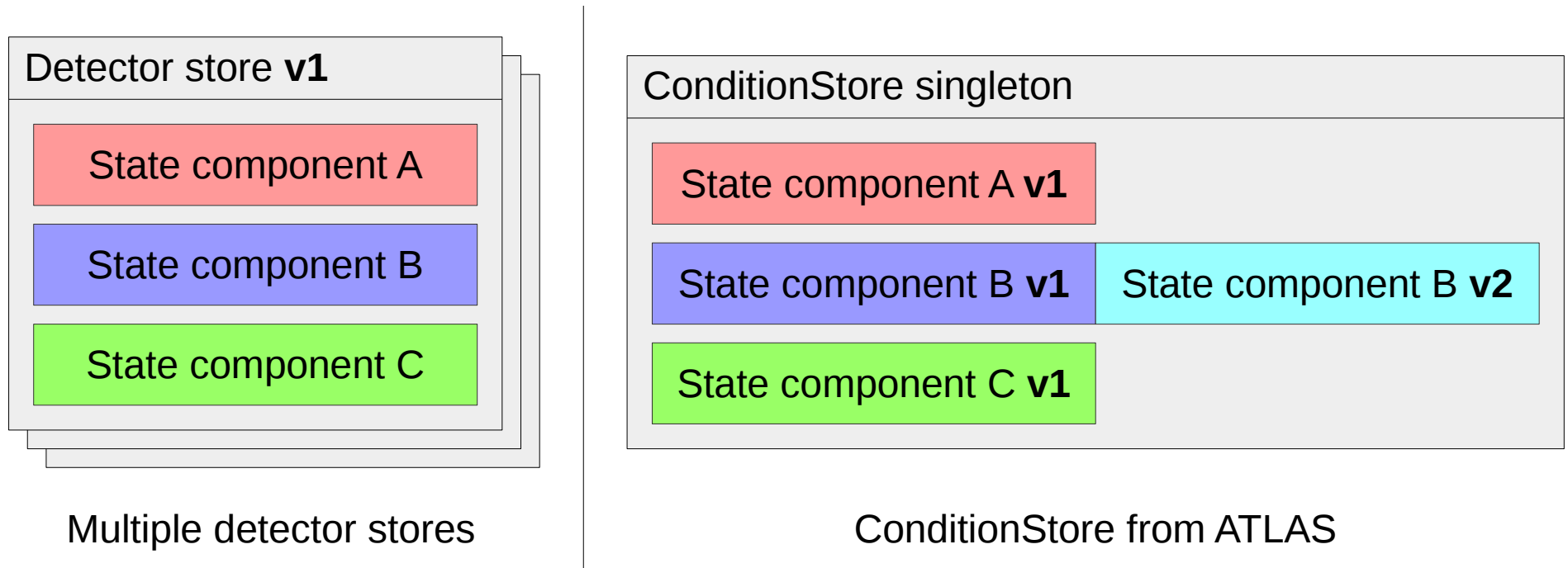


Defining conditions

- “Condition” definition depends on who is asked.
In this talk, I will define a condition as:
 - Detector state needed for event processing
 - That changes over time (has an interval of validity)
 - And cannot be stored in event data

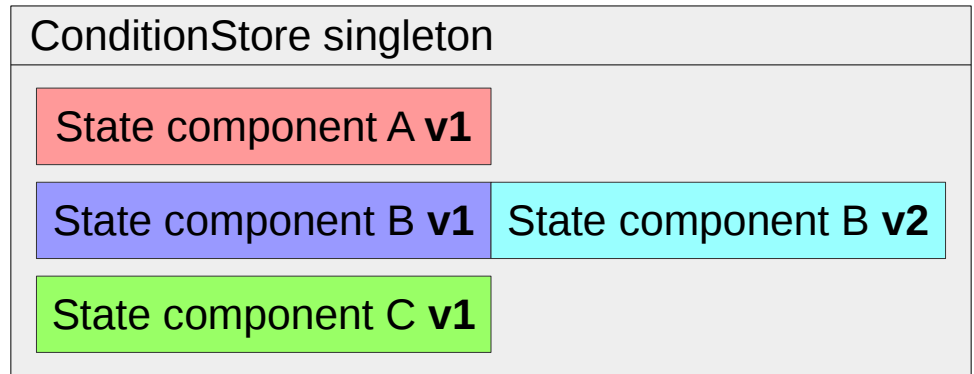
Concurrent condition handling

- Hold multiple detector states in RAM at once
- Solution currently explored by the ATLAS team



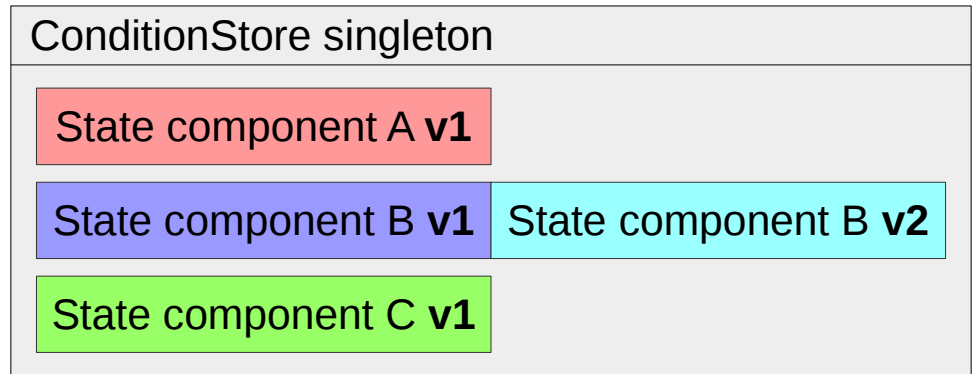
ConditionStore

- + Large development effort from ATLAS
- Other experiments largely unconvinced



ConditionStore

- + Large development effort from ATLAS
- Other experiments largely unconvinced



- Criticism has focused on two areas:
 - General design of this approach
 - Suitability for other experiments

ConditionStore criticism

- Inconsistent with Gaudi(Hive) design
- Mutable shared containers are problematic
- Nested layout makes garbage collection difficult
- All condition code is subjected to data races
- Too complex for simple use cases
- Tied to ATLAS framework & data model

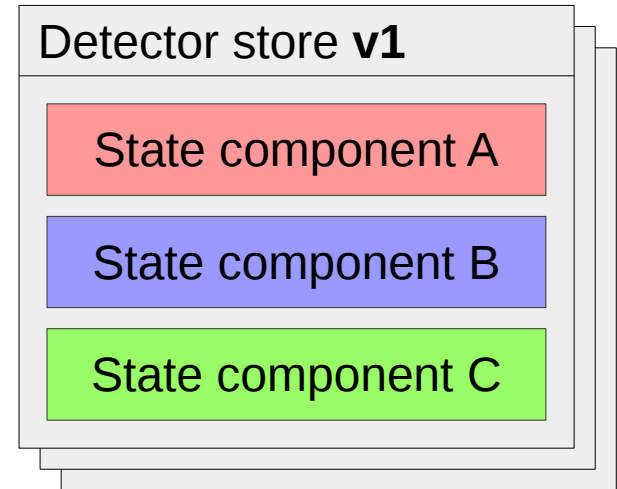
ConditionStore criticism

- Inconsistent with Gaudi(Hive) design
- Mutable shared containers are problematic
- Nested layout makes garbage collection difficult
- All condition code is subjected to data races
- Too complex for simple use cases
- Tied to ATLAS framework & data model

=> Reuse outside ATLAS seems unlikely :-)

Multiple detector stores

- + Consistent with the Gaudi design and interface
- + Read-only detector data
- + Garbage collection is simple



- Memory optimization (e.g. copy-on-write) is critical
- Migration still complex due to data/pointer caching
- Approach currently lacks experiment support

General remarks

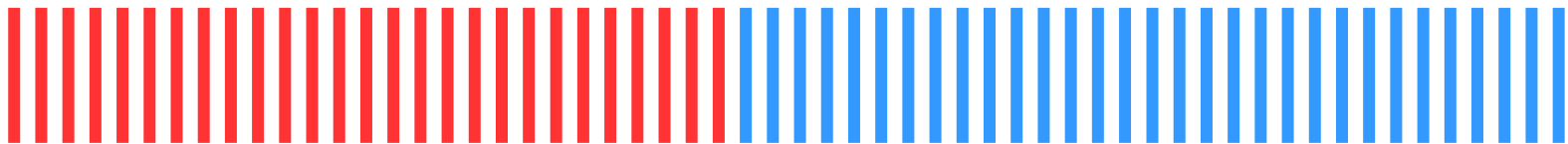
- Many components access condition state directly (estimated ~ 100 in ATLAS)
 - Moving to a new data model is a large effort

General remarks

- Many components access condition state directly (estimated ~ 100 in ATLAS)
 - Moving to a new data model is a large effort
- Memory management requires much care
 - Too many conditions in RAM goes against the memory efficiency goals of multi-threading
 - 100 threads should never mean 100 conditions
 - Heterogeneous condition size makes this harder

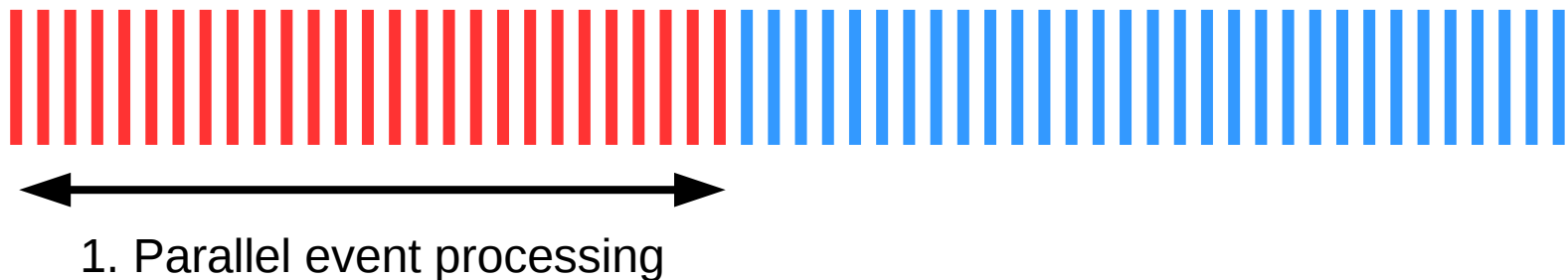
Scheduling barrier

- Only process multiple events concurrently if they are associated with the same detector state
- Flush event pipeline on detector state change



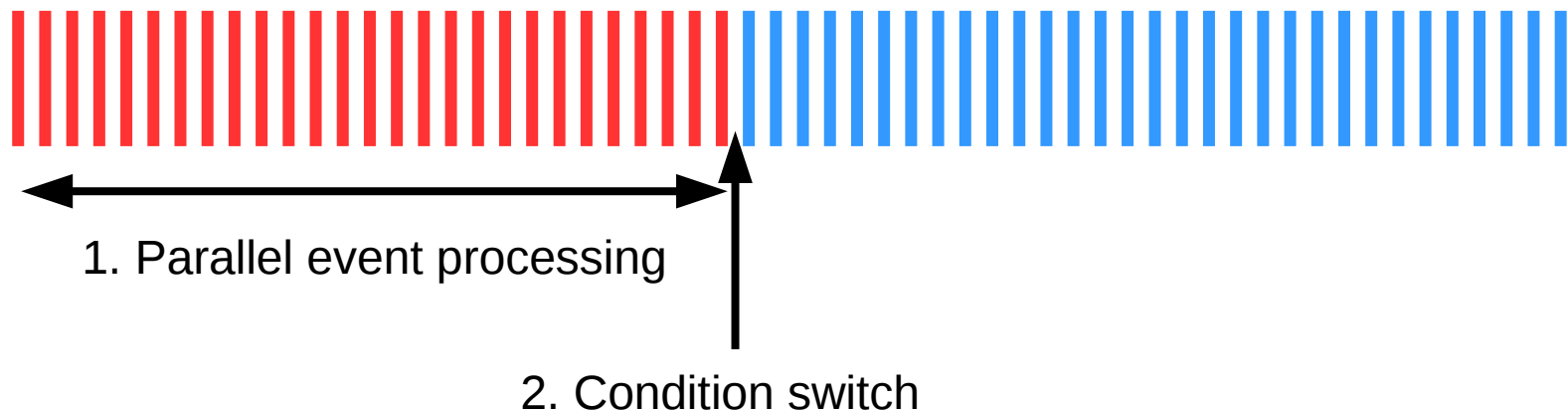
Scheduling barrier

- Only process multiple events concurrently if they are associated with the same detector state
- Flush event pipeline on detector state change



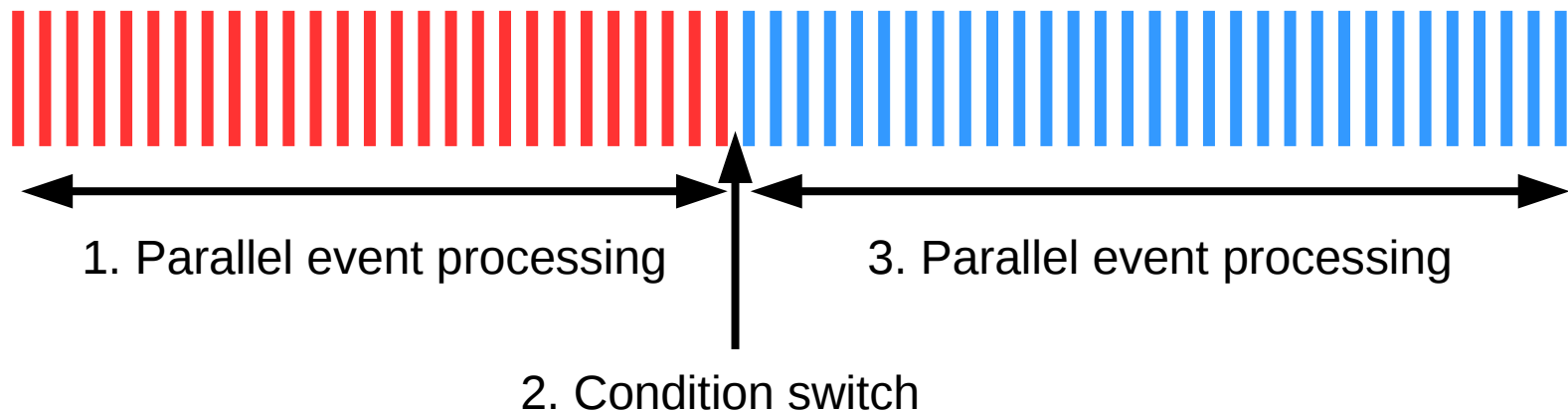
Scheduling barrier

- Only process multiple events concurrently if they are associated with the same detector state
- Flush event pipeline on detector state change



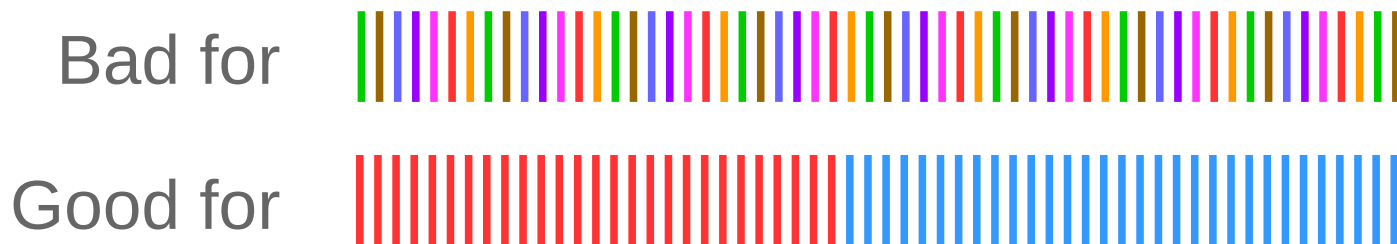
Scheduling barrier

- Only process multiple events concurrently if they are associated with the same detector state
- Flush event pipeline on detector state change



Scheduling barrier (2)

- + Memory usage is kept reasonable by design
- + Thread-safety much easier to achieve (detector state is read-only during event processing)
- Less general (conditions must change rarely)



Condition change rate?

- Consider reconstruction jobs (most CPU- and condition-intensive process after simulation)

Condition change rate?

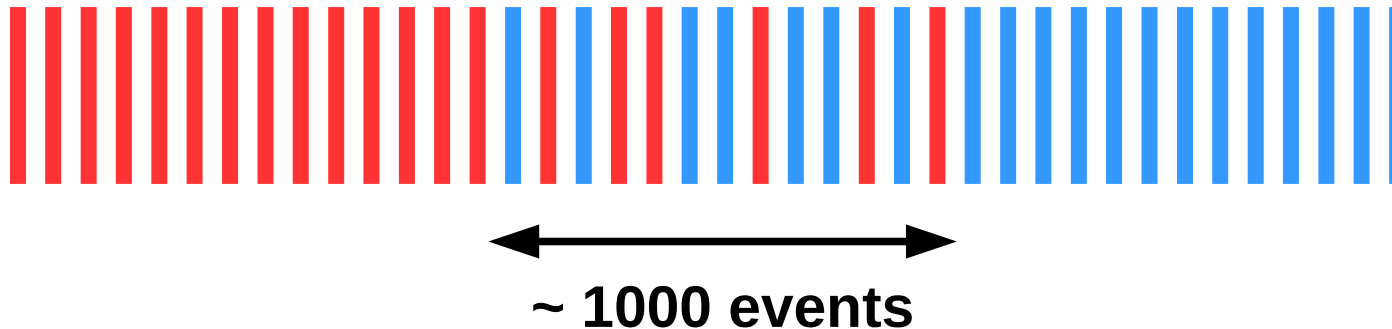
- Consider reconstruction jobs (most CPU- and condition-intensive process after simulation)
 - LHCb: ~ 1 change per hour in online processing, offline jobs have homogeneous conditions

Condition change rate?

- Consider reconstruction jobs (most CPU- and condition-intensive process after simulation)
 - LHCb: ~ 1 change per hour in online processing, offline jobs have homogeneous conditions
 - ATLAS: Condition change rate ~ 1 / minute
 - Trigger acceptance rate ~ 1 kHz
 - Ergo, ~ 60k events per condition
 - A typical event batch is ~ 1000 events

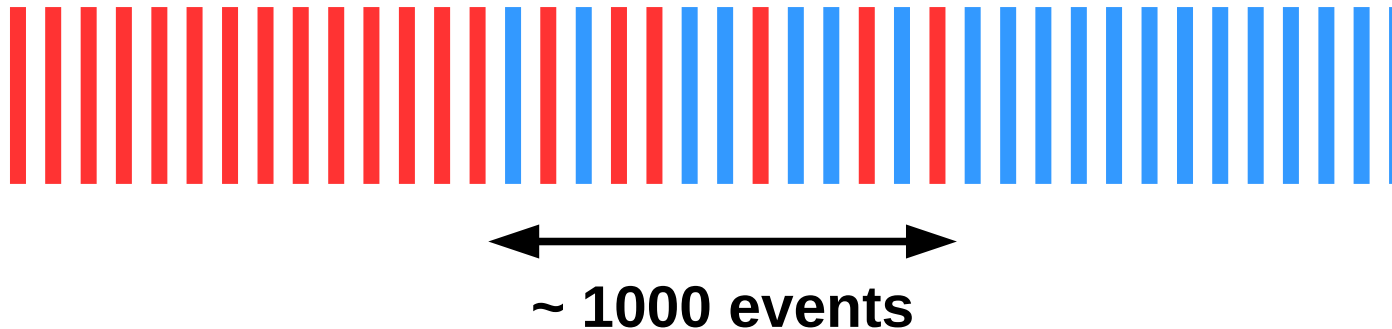
Trigger-induced smearing

- In many experiments, the HLT smears out transitions between detector states



Trigger-induced smearing

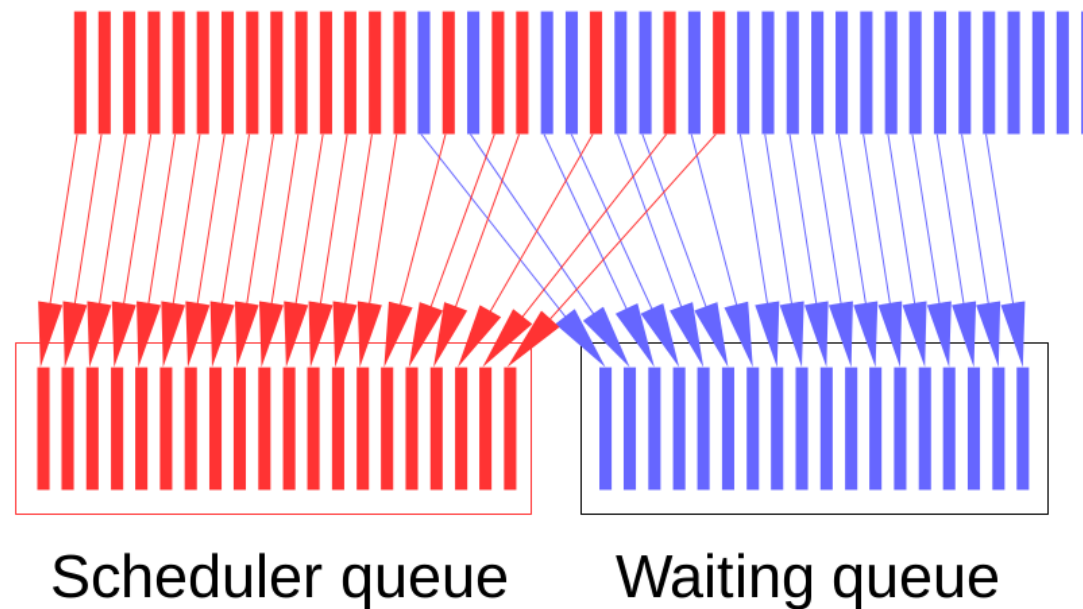
- In many experiments, the HLT smears out transitions between detector states



- Not well handled by barrier approach
- A significant performance issue for CMS

Reordered scheduling

- Possible solution: reorder the event stream using a waiting queue
 - Works best if only event timestamp info is loaded



Proposal

- Implement reordered scheduling in Gaudi
 - Reduces need for experiment-specific code
 - Simpler upgrade path for experiments with light requirements (strong support from LHCb)
 - Design is general enough for reuse outside Gaudi

Proposal

- Implement reordered scheduling in Gaudi
 - Reduces need for experiment-specific code
 - Simpler upgrade path for experiments with light requirements (strong support from LHCb)
 - Design is general enough for reuse outside Gaudi
- Compare/combine with ATLAS approach?
 - When the ATLAS implementation is ready
 - Reordering could help shrinking condition caches!

Roadmap

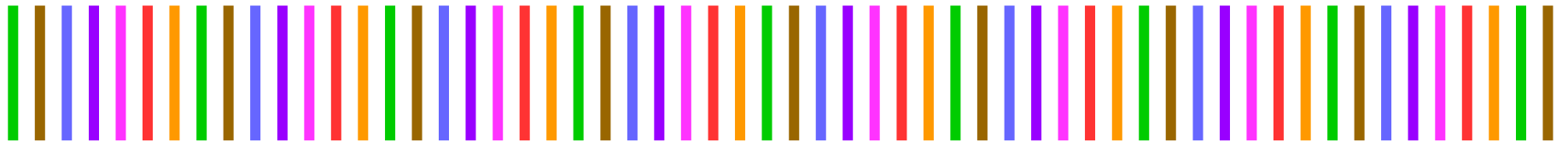
- July-October: Add basic support to Gaudi
 - ConditionSvc
 - Reordered scheduling
 - Experiment interface
- November-December: Integrate and test in an LHCb development branch, refine iteratively
- Medium-term: Try integration in ATLAS, comparison with ConditionStore

Milestone document

- We summarized the feedback of many stakeholders in a milestone document
- Still some remarks to be integrated, final version coming next week

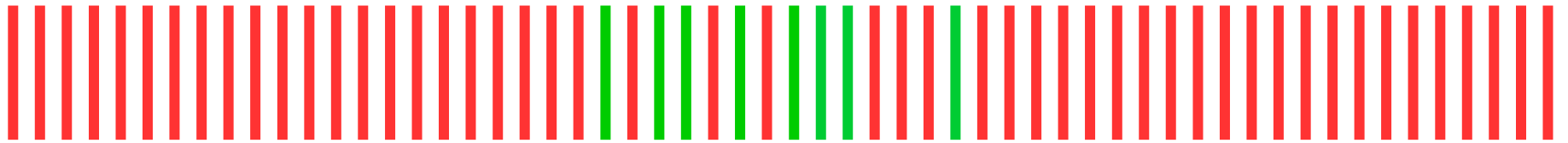
Questions?

What about the worst case?



- Challenging for any multithreaded solution
 - With concurrent conditions, keeping RAM usage reasonable is difficult
 - With reordering, performance drops slightly below that of the serial framework
- Not foreseen to be a frequent use case

What about short transients?



- Much discussion on this during design phase
 - In the context of ATLAS' calo noise bursts
 - Turns out managing these using a condition IoV was already a performance issue in serial Athena
 - These are thus managed lumiblock-wide
 - Is there any other use case?