Multithreading Model in Allpix Squared

Koen Wolters, *Victor Sonesten, Mohamed Moanis Ali, Simon Spannagel, Paul Schütze, and other contributors*

2nd Allpix Squared User Workshop - 15 August 2021

About me

- Involved in Allpix Squared development from the beginning
- Developed foundations of the framework as CERN Technical Student from February - August 2017
    - Designed fundamental framework architecture (v1.0)
    - Implemented initial modules (party ported from AllPix)
- Continued to contribute to the project afterwards
    - Reviewed framework modifications, and fixed several issues
    - Involved in the design of the next major release v2.x
- Left the field, currently working as Software Engineer at Google Zurich

Outline

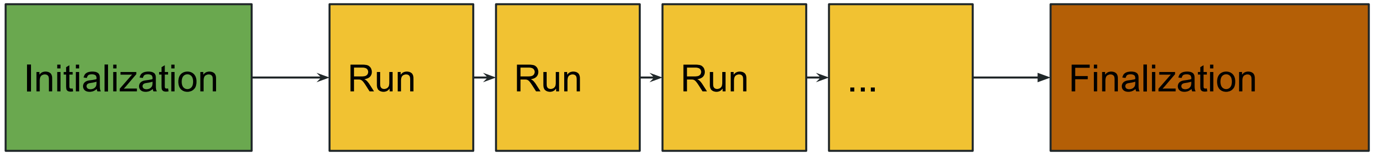
- Improved Multithreading Model
- Design Challenges
- Performance Results
- Conclusion

*This talk covers framework design **fundamentals**, although skipping many details, understanding those foundations is **not** required to be an effective AP2 user*

Improved Multithreading Model

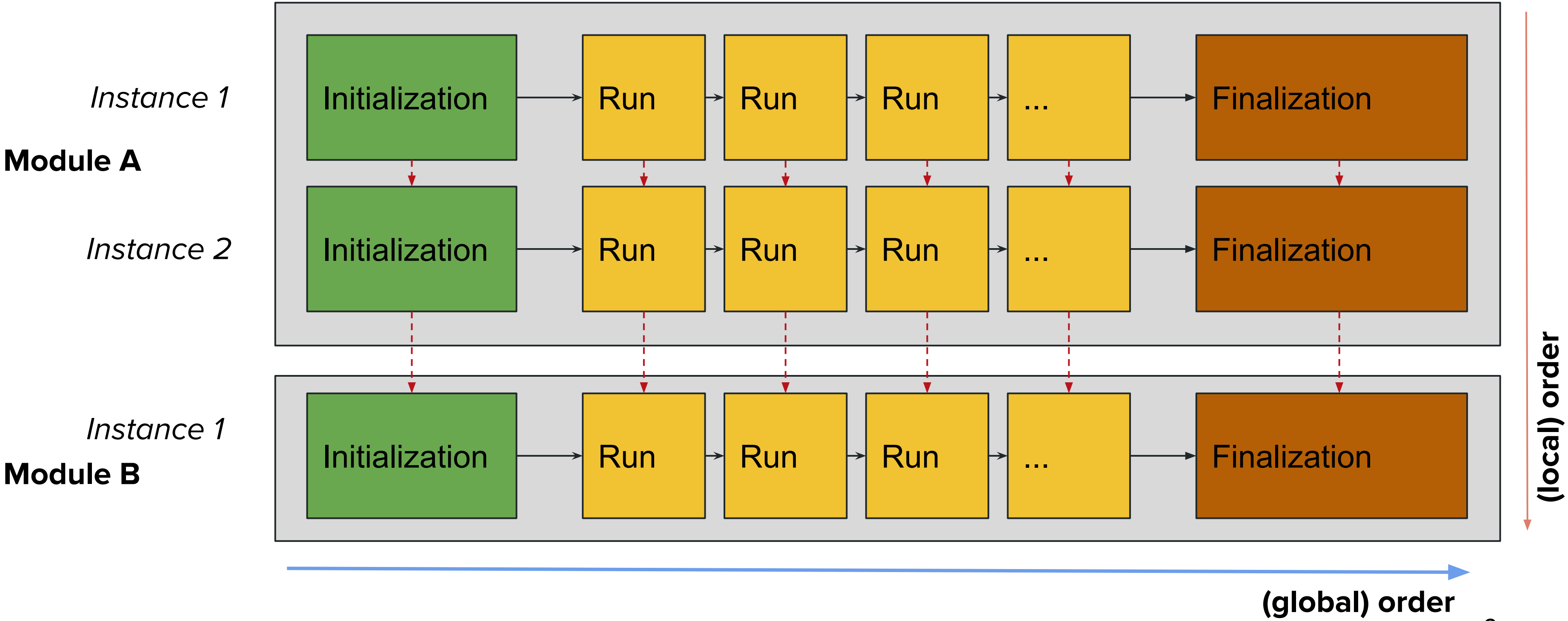
Modular System

- **Module** is an independent component with **inputs** (configuration, internal and/or external data) and **outputs** (internal data, results and/or visualizations)
- Three main stages
  - Initialization (construction)
  - Executing independent events
  - Finalization (destruction)



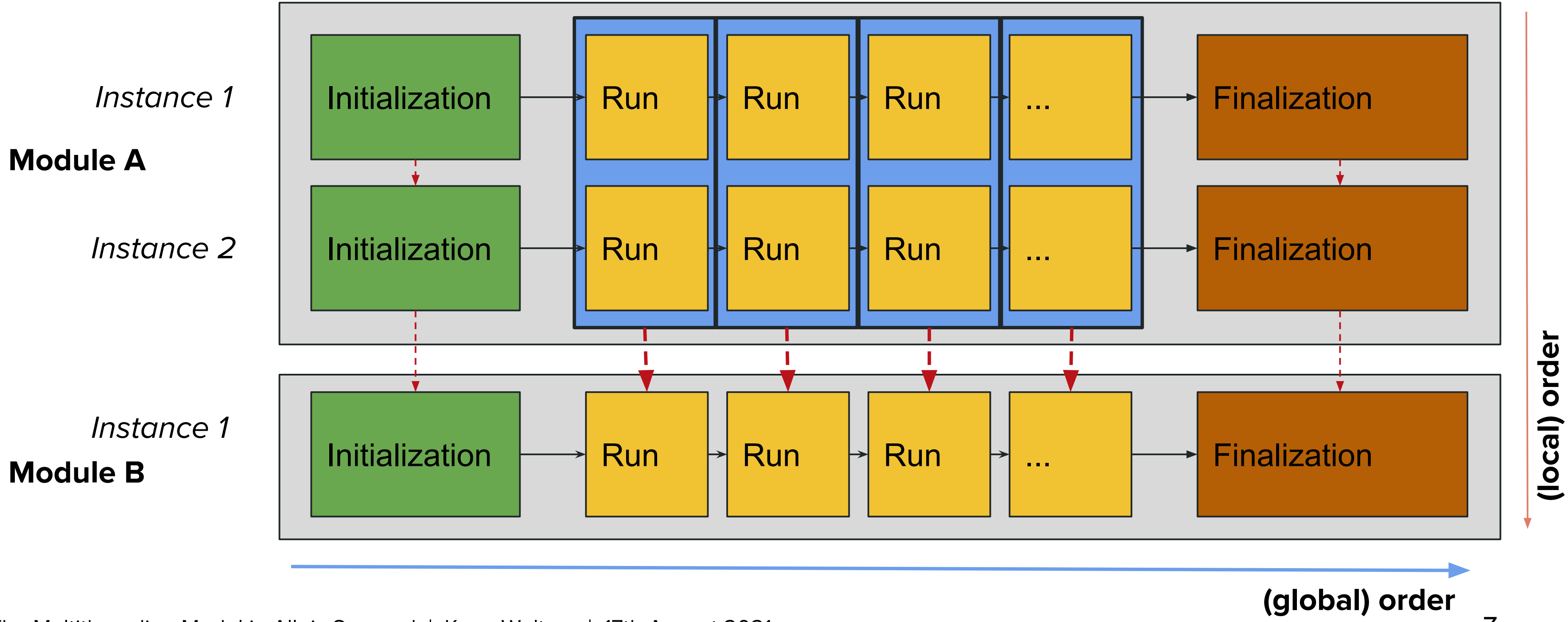
Module Instances

Modules are *unique* or *detector*-specific per input/output ➜ multiple **instances**



Parallel Execution in First Release

Instance-based parallelisation (local)



First Multithreading Model Advantages and Limitations

Advantage: Requires *thread pool*, but thread-safety (relatively) **easy** to achieve MR 22

Framework: only some core logic shared between modules had to be changed

- Data should handle parallel dispatch: possible with *trivial* mutex locking
- Logging should support parallelism: possible with *trivial* mutex locking
- ROOT has a *global* data model: enable ROOT *internal* thread safety

Module: instantiations are **independent**, class data member variables **never** accessed in parallel, execution of *run* function practically '**single-threaded**'

- Shared data between instances not possible
    - No Geant4 support
    - No global statistics (without atomics or locking)
    - No global ROOT TDirectory changes, for example to write plots

Intermezzo: Thread Pool

Note: Creation and destruction of threads has a substantial **overhead**, kernel-level data has to be initialized and maintained (thread => lightweight process)

Observation: Threads per module instance per event is **expensive**

Idea: **Reuse** threads and run *lightweight* tasks ➔ Thread Pool

- Initialize number of threads based on number of CPU threads (cores)
- Submit tasks (functions with data inputs attached) to thread-safe queue
- Thread workers pop tasks (in thread-safe way) from queue and execute them
- Listen to task completion signals (*futures*) to order tasks

First Multithreading Model Advantages and Limitations

Limitations: **Impossible** to generically achieve maximum parallel throughput

Instance-bound: Parallel speed-up is constrained by the number of instances, typically bound by number of **detectors** (or input/output params)

- (Almost) no speed-up for unique modules (none without multiple input/output)
- Speed-up limited by slowest instantiation, barely any performance improvement if only DUT simulation is expensive for example

Module-constrained: Complete modules are still executed without multithreading (only instances are parallelized)

- No scalability for multiple computationally expensive modules

New Multithreading Model

Principle of Allpix Squared: Events are **independent** passages of one or multiple particles ('reflect the physics')

Observation: No (direct) data dependencies between different events exists

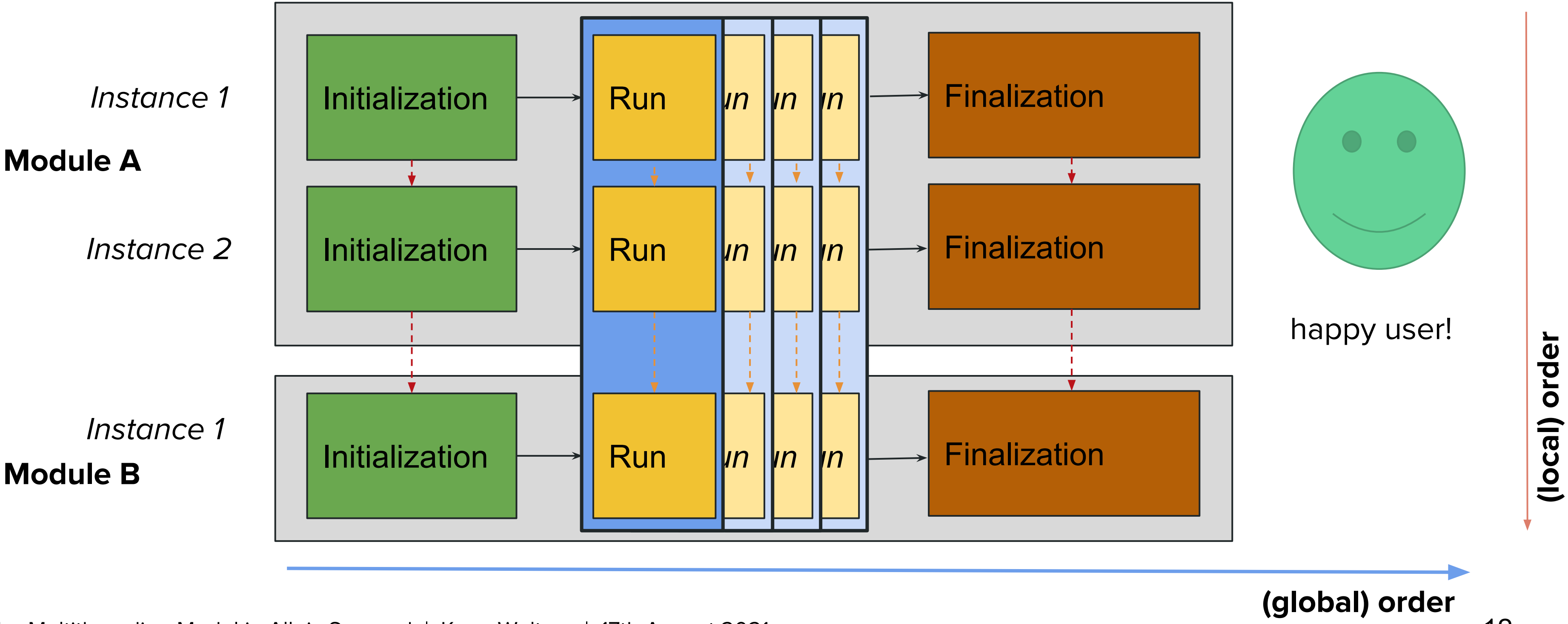
Conclusion: Independent Monte-Carlo simulations are *embarrassingly parallel*

Idea: Entire events can be run in parallel

Advantage: Multithreading bound by number of events (>>> instance count)

Parallel Execution in Second Release

Event-based parallelisation (global)



Let’s do it!

Awesome idea! Let’s go run all those independent events in parallel (on my beefy multi-core machine), achieve a enormous speed-up and be happy! :)

Well, unfortunately it hasn’t been that easy…

Design Challenges

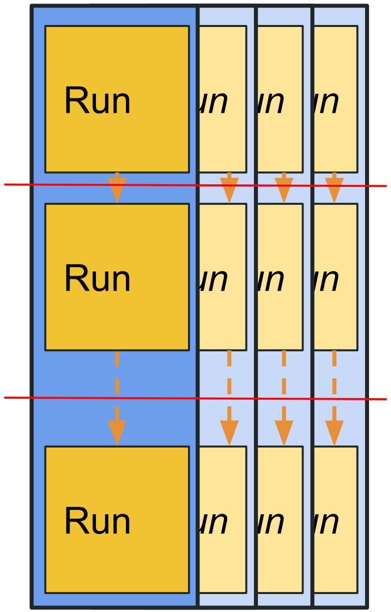
Challenge #1: Parallel Dataflow

Old: Only parallel *dispatch*, **no** parallel data streams, input to *instances* received **in-order**, *sequential* run function

New: **Parallel** data flow, *instances* receive data from multiple events together, *parallel* run function

Implication: More elaborate data handling required

- Data separated per event: global ➔ **local** messenger
- Member variables **not** implicitly thread-local anymore
  - Need to use (function) local variables
  - No binding of messages to member variables



Intermezzo: Messenger

Note: Allpix Squared passes objects with simulation data around using messages (initially converting simulation input to messages and converting it back at the end)

Observation: Messages need to be passed around between module *instances*

Idea: Abstract data passing away from users using a messenger

- Allows instance to **bind** to messages to listen to (source module unspecified)
- Instances **dispatch** messages, messenger *magic* forwards to listening instance
- Instances **fetch** the right data from the listening module
  - Old: (most) messages **assigned** to local class variables (no support for parallel data flow)
  - New: messages fetched via messenger **function call** (supports parallel data flow)

Personal note: I consider using member variables for binding messages the most significant design flaw in the first release

Challenge #2: Parallelisation in Geant4

Geant4 interface through **RunManager** (*note*: AP2 **event** → Geant4 '**run**')

- Original version does not support parallel execution
- New MTRunManager add multithreading support

Problem: Manager uses internal thread pool, not compatible with AP2

Solution: Implement custom run manager (compatible with MT disabled)

- Creates worker-specific run managers to generate *beams* in parallel
- Required investigation into various **complex** Geant4 internals

Challenge #3: Run Reproducibility

Allowing to **reproduce** simulations results is **important** for many reasons

Problem: Event multithreading execute events in **arbitrary** order, **no** common order of random number generation ⇒ no reproducibility

Solution Idea: Initialize **fixed** order seeds to individual events and generate local random numbers per event (instance order within events is **fixed**)

- Use event-based seeds, having event generators is too **expensive**
- Testing was difficult due to **STL** random non-fixed ➔ use **Boost**
- **Violations** especially in **Geant4**: we found a bug with reproducibility

Challenge #4: Modules with Order Requirements

Problem: **Not** every modules can be run in parallel, especially **writers** (and readers) need sequential data to preserve reproducibility of events

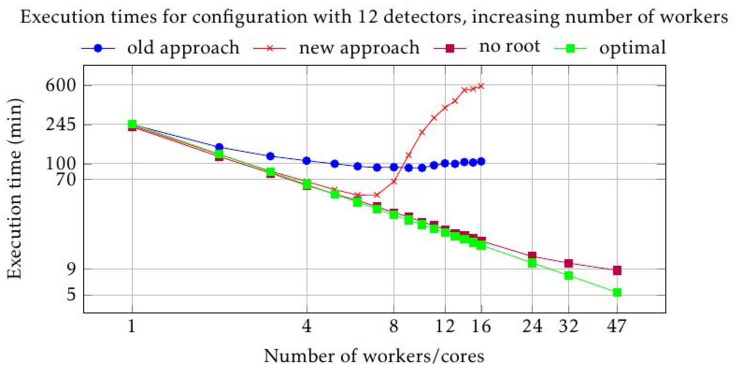
Solution: Allow **buffering** of events to execute certain modules sequentially

- First version uses abstraction layer await completion of earlier events
- Led to intricate **deadlocks** due to limited buffer size (restricted RAM)
- Expanded event task system to allow **resubmission** of buffered events
- Interesting issues building task system: exception handling, and more...

Challenge #5: Multithreading Issues in ROOT

ROOT was started in a time were multithreading was not a thing yet, and that pain continues to exist: **many** performance issues with parallel ROOT

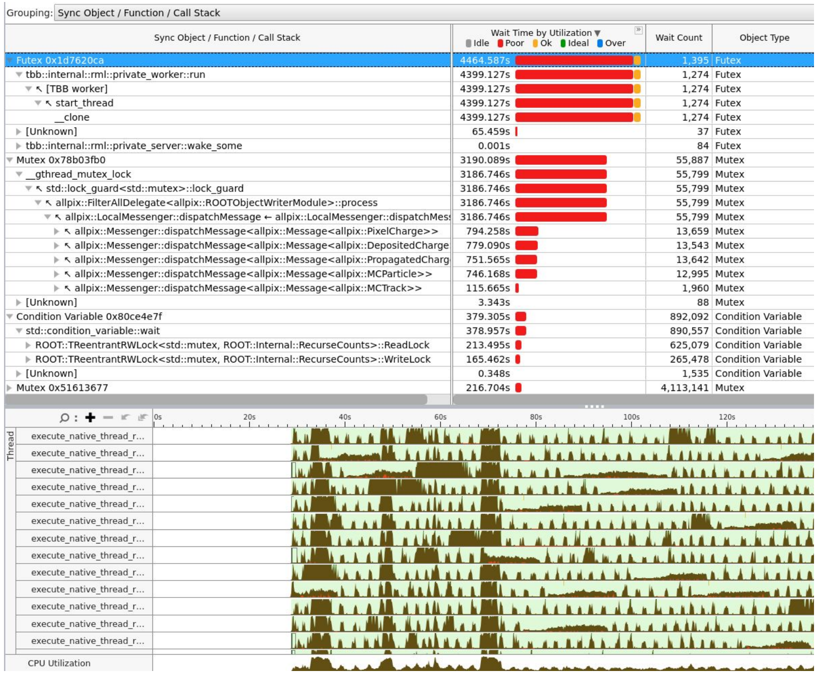
- Implicit MT uses internal thread pools
- Global locking in Allpix (ROOT-based) object creation and destruction
  - Workaround for object ID handling
  - Explicit locking for data races
- Parallel histogramming



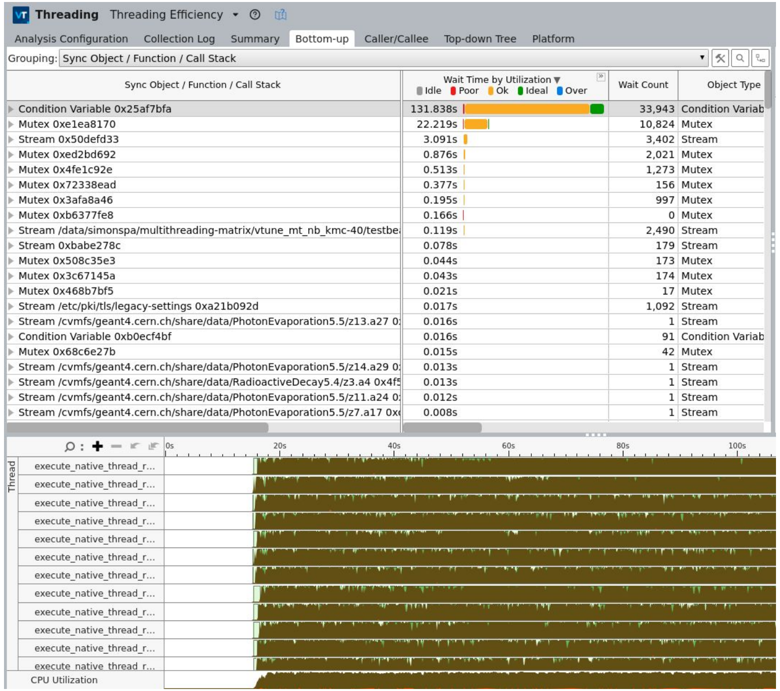
Multiple discussions and with ROOT team (and bugs...)

Challenge #X: ....

Just a **sneak-peek** into challenges, many obstacles to overcome



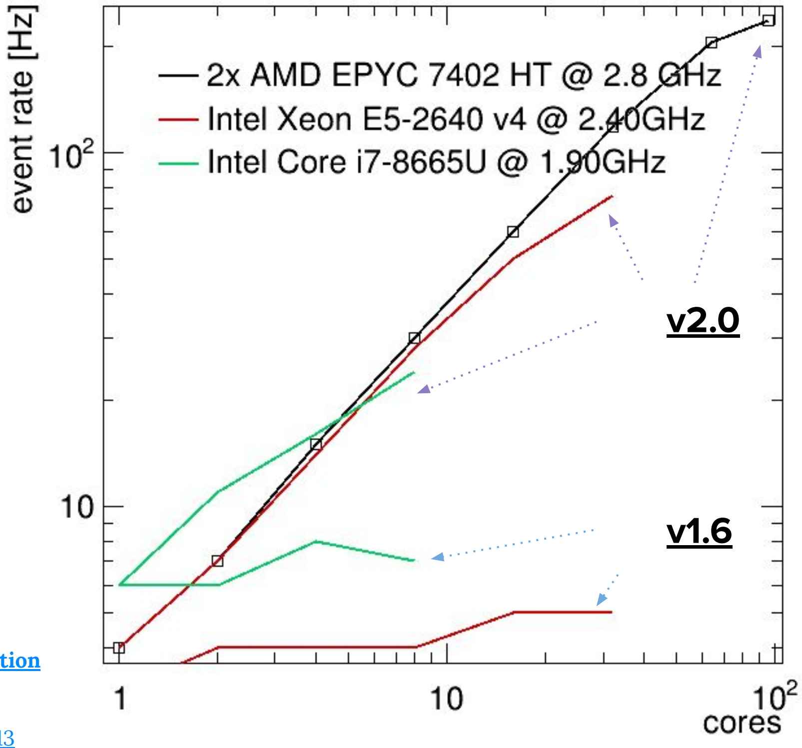
lots of profiling...



Results

Performance Results

configuration from paper†



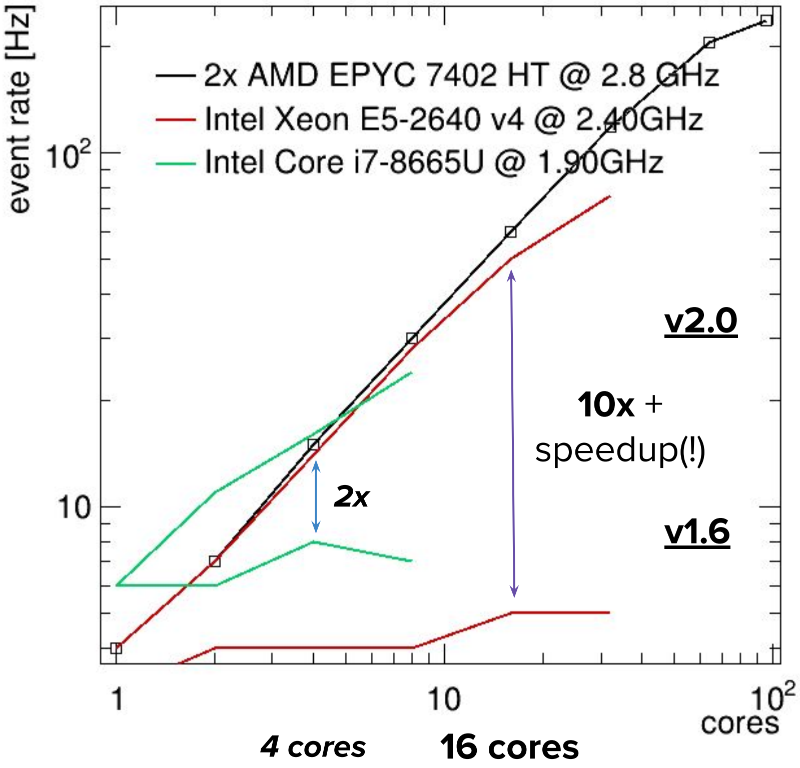
†S. Spannagel et al., [Allpix2: A modular simulation framework for silicon detectors](), Nucl. Instr. Meth. A 901 (2018) 164 – 172, [doi:10.1016/j.nima.2018.06.020](), [arXiv:1806.05813]()

Comparison First and Second Release

configuration from paper†

7 detectors

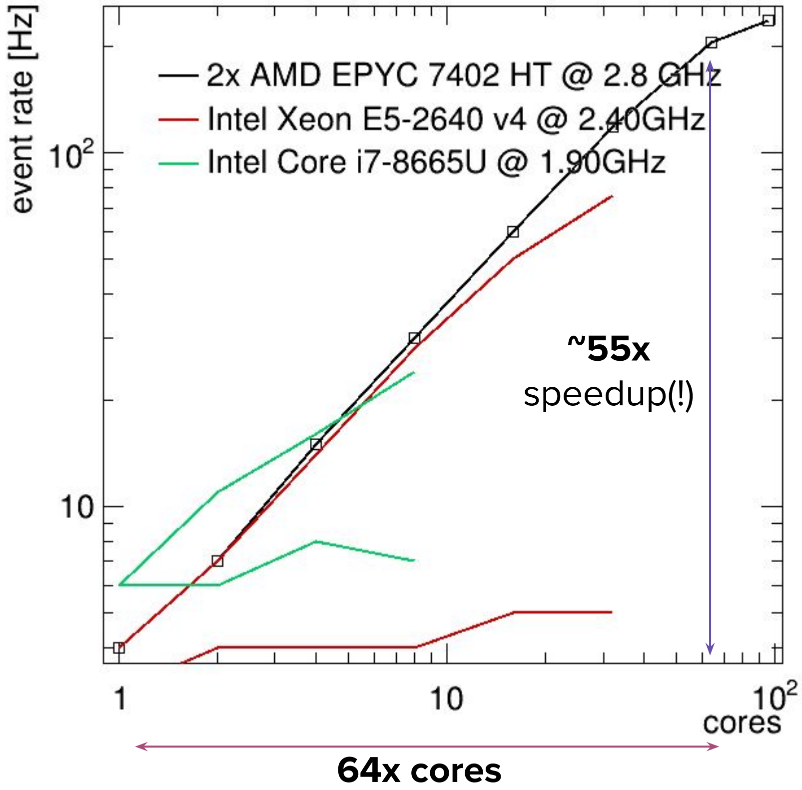
speed-up can be even more significant on other configurations!



Comparison First and Second Release

configuration from paper†

hour → minutes



Conclusion

Conclusion

- Move from **instance-based** to **event-based** multithreading
- Major restructure of the framework fundamentals
- Various kind of **challenges** to resolve on the way
- **Learned**: multithreading is hard
  - Deadlocks and contention are easy
  - Debugging issues is difficult (lack of reproducibility)
  - Impact of single contention spot can become very significant
  - Surprising huge performance improvements
- **Result**: impressive speed-up and excellent scalability

Thank you for your attention!