Parallel Computing of SNiPER based on Intel TBB

Jiaheng Zou, IHEP, China

On behalf of the SNiPER framework working group

ACAT 2017, Seattle Aug. 24, 2017

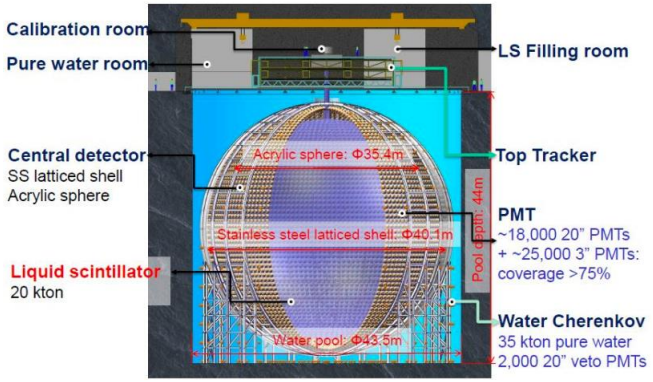
The SNiPER software framework

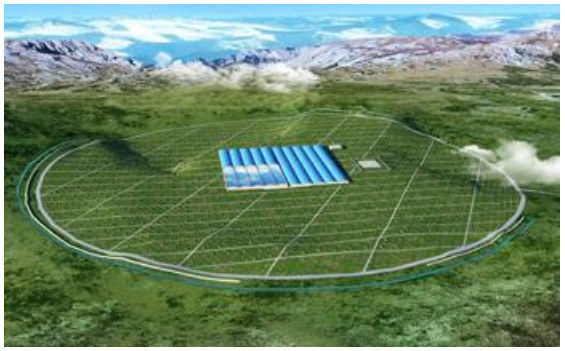
SNiPER: Software for Non-collider Physics Experiment

- a general purpose offline software framework
- lightweight and simple to use

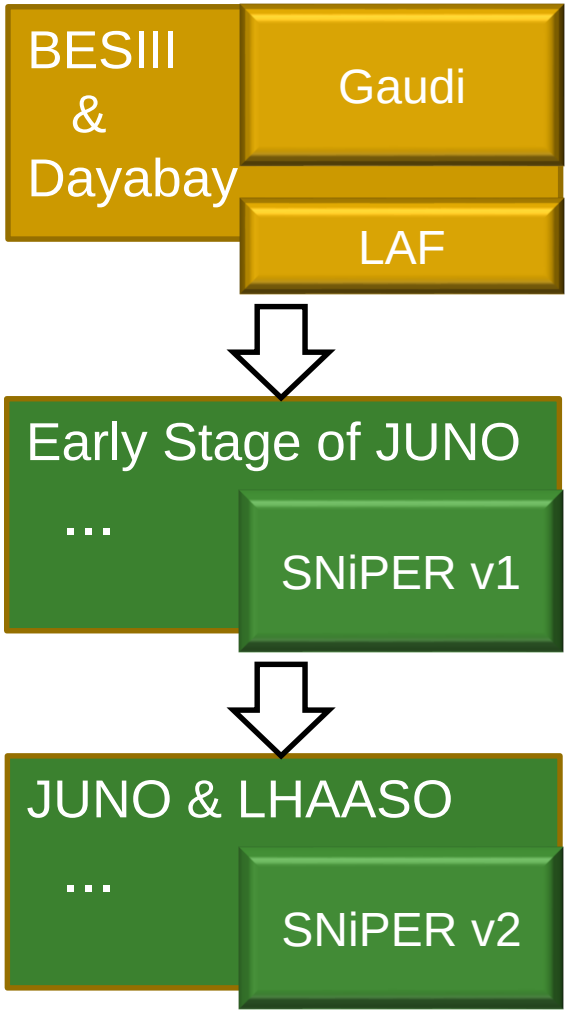
Current Status

- Performs well for JUNO and LHAASO experiments
- Other users or potential users, such as CSNS and CEPC





The Evolution of SNiPER



Usage of the famous Gaudi framework

- Offline software of BESIII and Dayabay exp.
- Learned a lot from Gaudi

Development of LAF (~2012)

- Lightweight Analysis Framework from scratch
- Fulfill specific requirements of Dayabay analysis

SNiPER v1 (~2013)

- Expanding of LAF in the early stage of JUNO

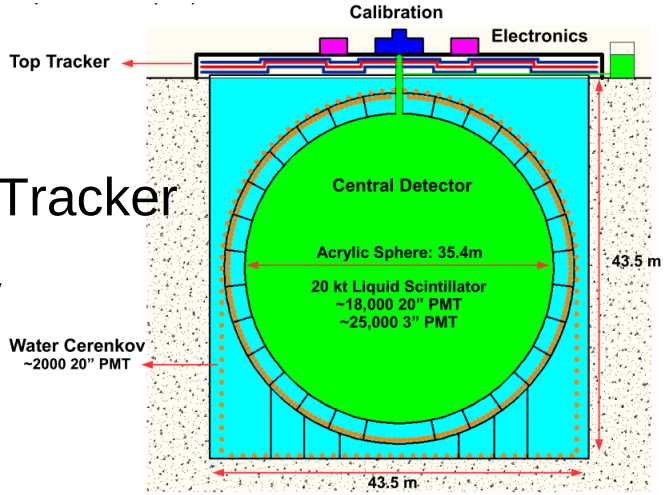
SNiPER v2

- More new features and enhanced utility
- Applied to more experiments besides LHAASO

Key Features

- Hybrid programming of C++ and Python (Boost.Python)
  - C++ for functional modules to ensure efficiency
  - Python for friendly and flexible configuration
- Dynamic loading and configuration
- Fundamental concepts
  - Algorithm and Service that learned from Gaudi
  - Task and Incident – flexible event processing controlling
  - Data Buffer – powerful memory management

Specific Requirements of JUNO



Three sub-detectors

- Central Detector, Water Cherenkov, Top Tracker
- Each sub-detector is triggered separately

Anti-neutrino Inverse Beta Decay

$$\bar{\nu}_e + p \rightarrow e^+ + n$$

$$e^+ + e^- \rightarrow 2\gamma$$

$$n + Gd \rightarrow Gd^* \rightarrow Gd + \gamma_s \ (8\mathrm{MeV})$$

- In Central Detector, e^+ and n are triggered separately, too

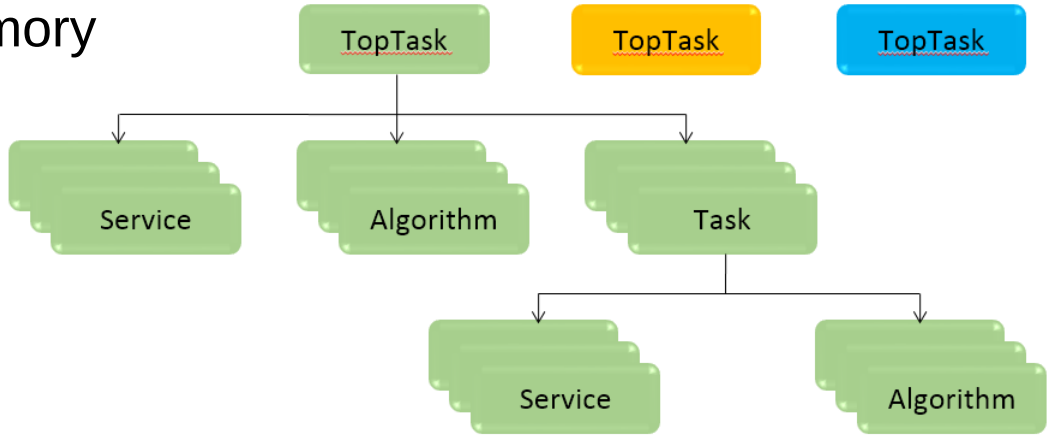
Two main challenges

- Physics events number is different from triggers (logical events) number
- It is mandatory to handle (logical) events correlations

SNiPER Task

SNiPER Task seems like application manager

- manage its own algorithms and services
- manage its own data memory



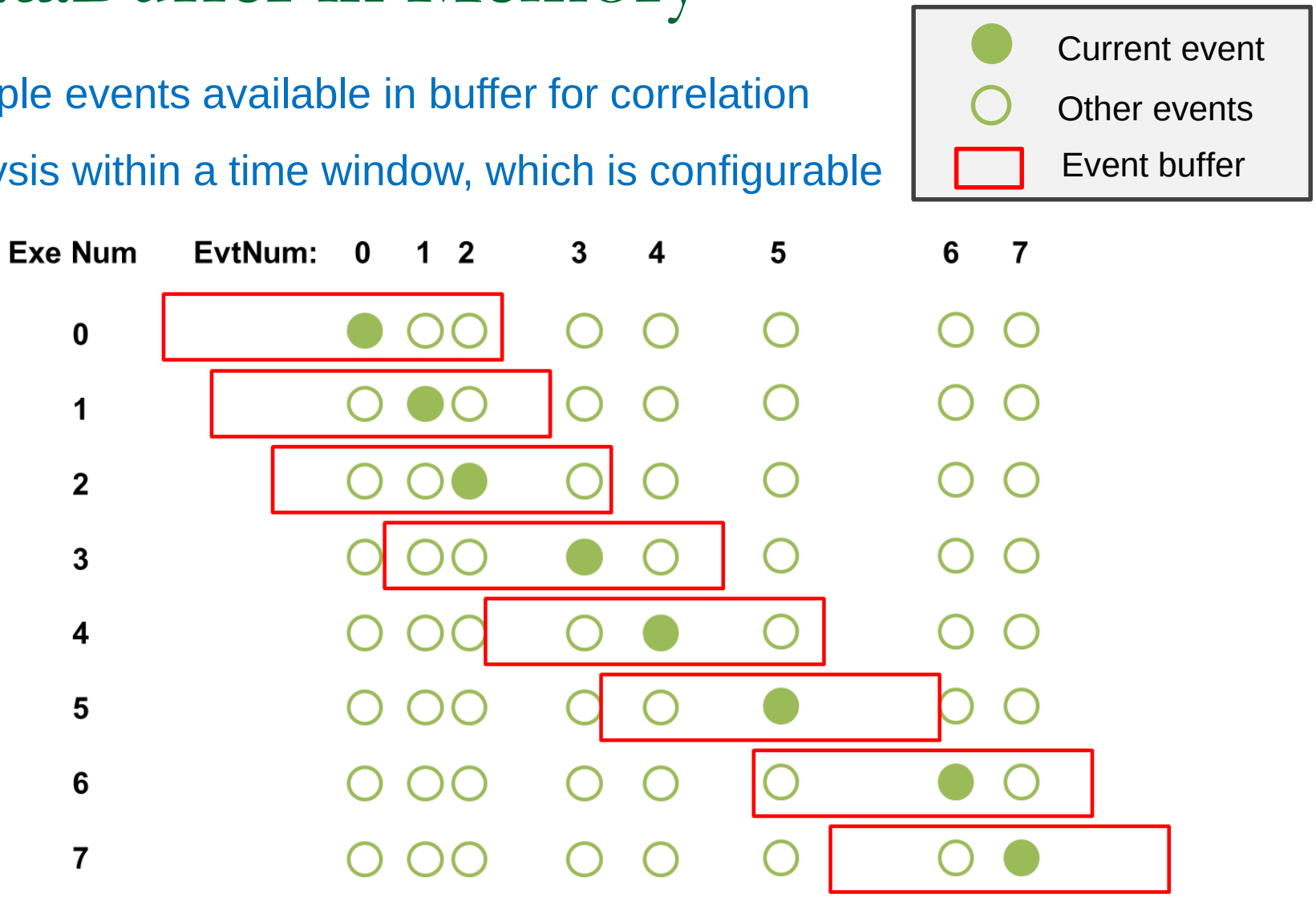
SNiPER Task is not application manager

- there can be more than 1 Task instances in a job
- a TopTask can own many (sub) Tasks
- Task instances can interact with each other via “Incidents”

Each Task, as a subroutine, can handle different number of events on demand

DataBuffer in Memory

Multiple events available in buffer for correlation analysis within a time window, which is configurable



- Only serial computing is supported in the original SNiPER
- We began parallel computing of SNiPER in 2017, the development and testing is ongoing

Goals of multi-threaded SNiPER

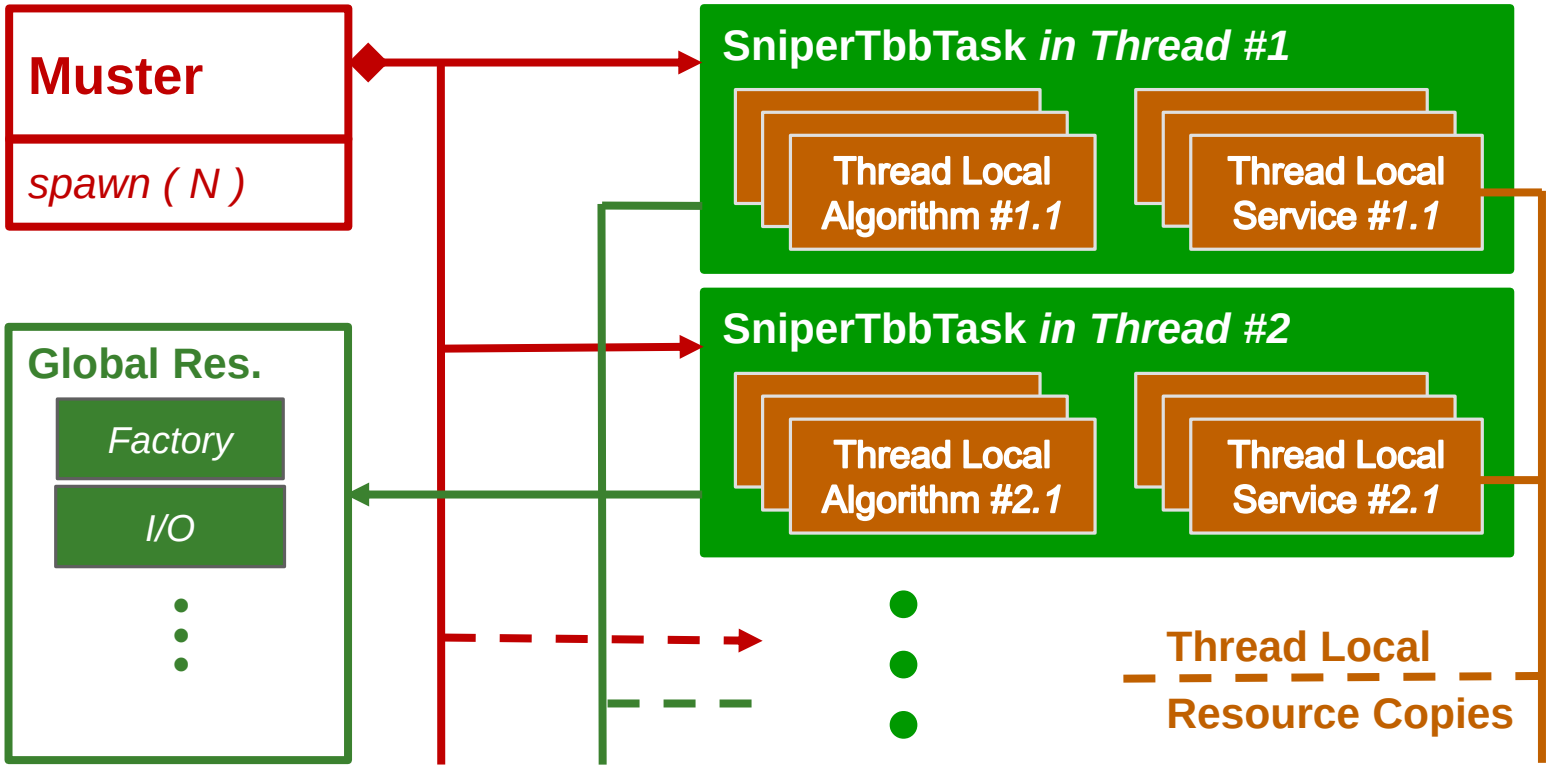
- (Almost) transparent to users
  - Very few constraints to user algorithms
  - Minimize the migration costs from serial to parallel computing
- Compatible with both serial and parallel user algorithms
  - Advanced users can implement parallel algorithms via TBB, OpenMP or Raw Threads
- Non-invasive to SNiPER kernel modules
  - Parallel features are implemented as a wrapper of the kernel
  - We can apply serial computing as before without the wrapper
  - Smoothly switch between serial and parallel mode

Top Level Concept Design

- Feasibility
  - Event level parallel processing
  - Create “thread local” SNiPER Task copies to suppress conflicts
  - Based on the Intel TBB library
- A coincidence: SNiPER Task and Intel TBB task
  - Bind a TBB task with a SNiPER Task
  - Appoint each event to a TBB task
  - Events are processed simultaneously in different SNiPER Task copies that invoked by TBB tasks

Muster: the multi-threaded wrapper

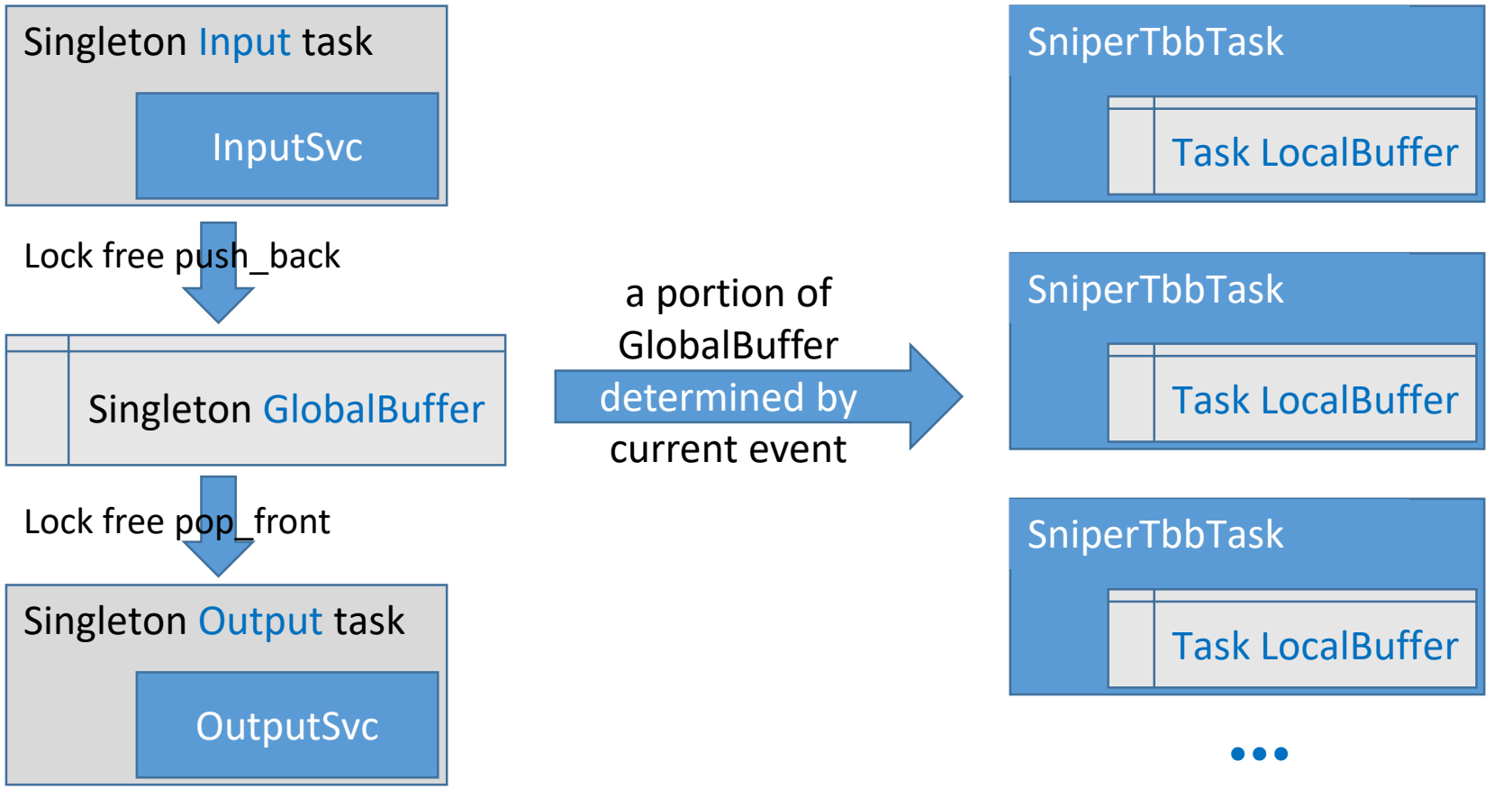
- Muster — Multiple SNiPER Task Scheduler
- SniperTbbTask — Binding of a SNiPER Task to a TBB task



I/O Concept Design

Each I/O stream is handled by a separate Input/Output task

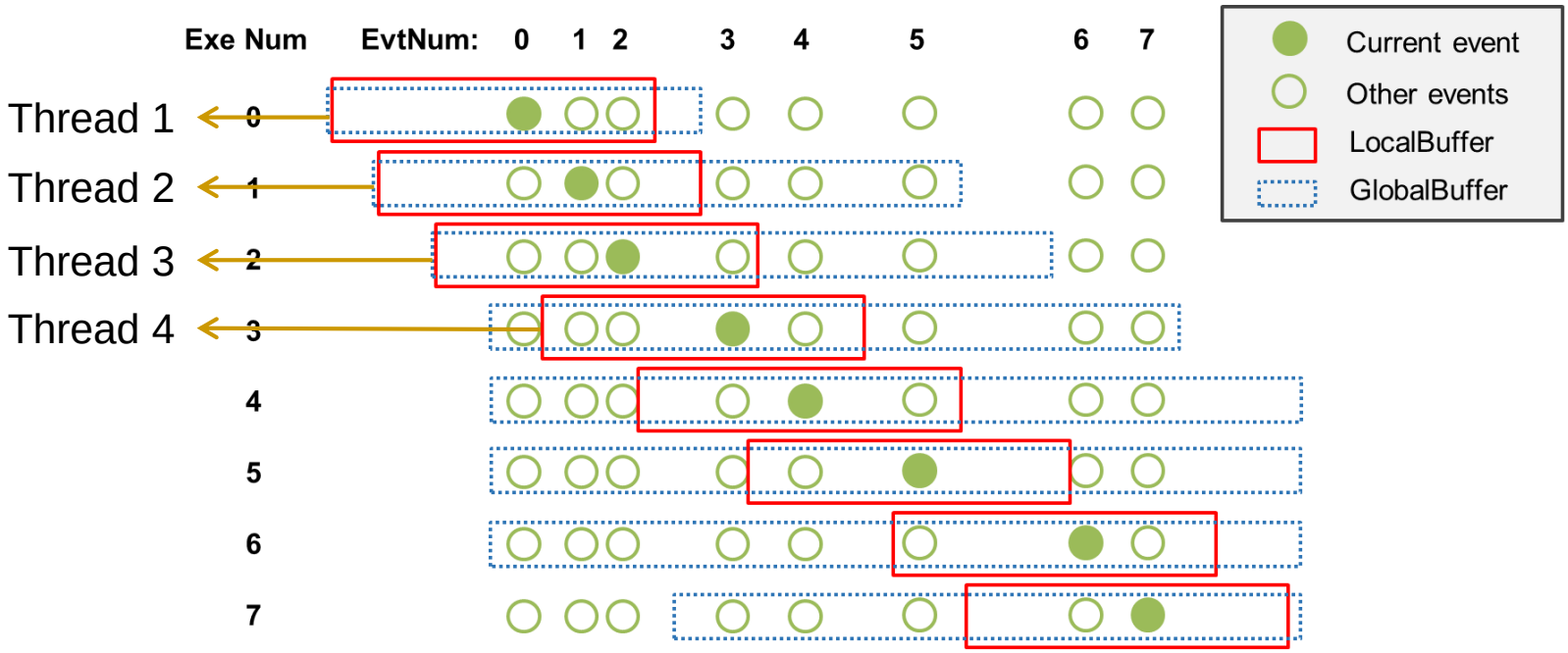
Data processing tasks are decoupled from I/O streams



Memory Management in Threads

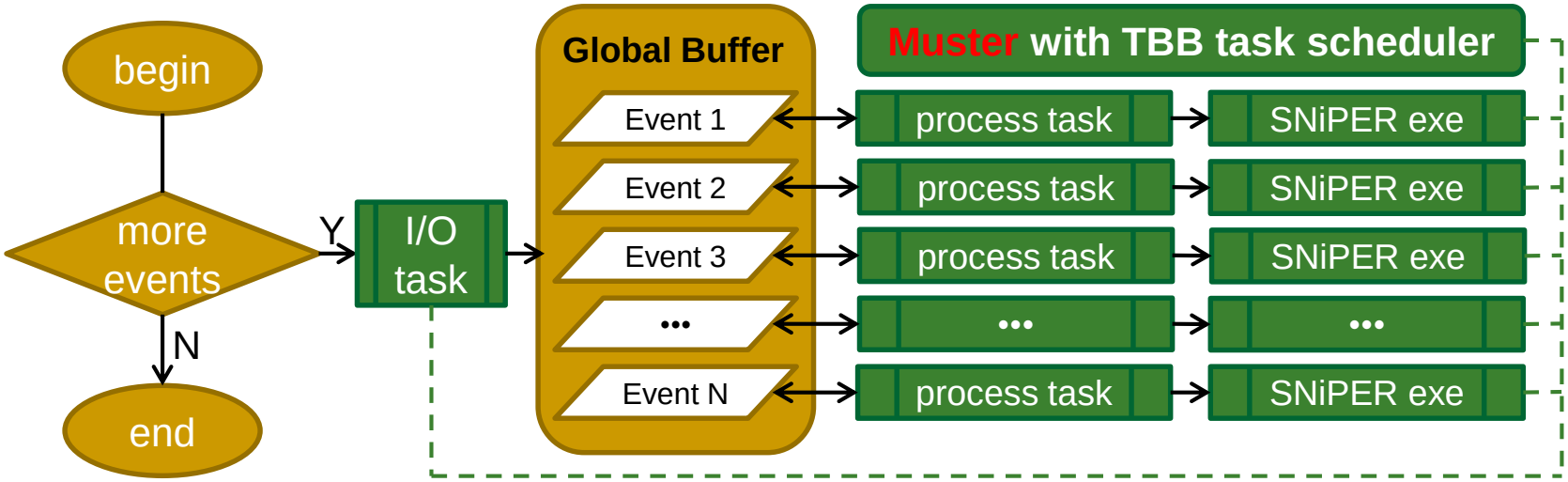
Global Buffer – bind to I/O streams, cache all events in memory, keep events in right order for output

Local Buffer – same as the serial version of DataBuffer



Execution with Muster

- Break the serial event loop into concurrent TBB tasks
- I/O is handled in separated TBB task
- There is no change for the processing procedure of each event in a TBB task
- Transparent to most users (algorithm developers)



Optimization with TBB features

- Recycling of TBB task objects
  - Event processing tasks can be recycled all the time
  - The number of processing tasks equal the number of threads
  - These tasks are allocated/released at the beginning/ending
- Scheduler Bypass
  - We are able to switch processing tasks and I/O task manually within rich context information

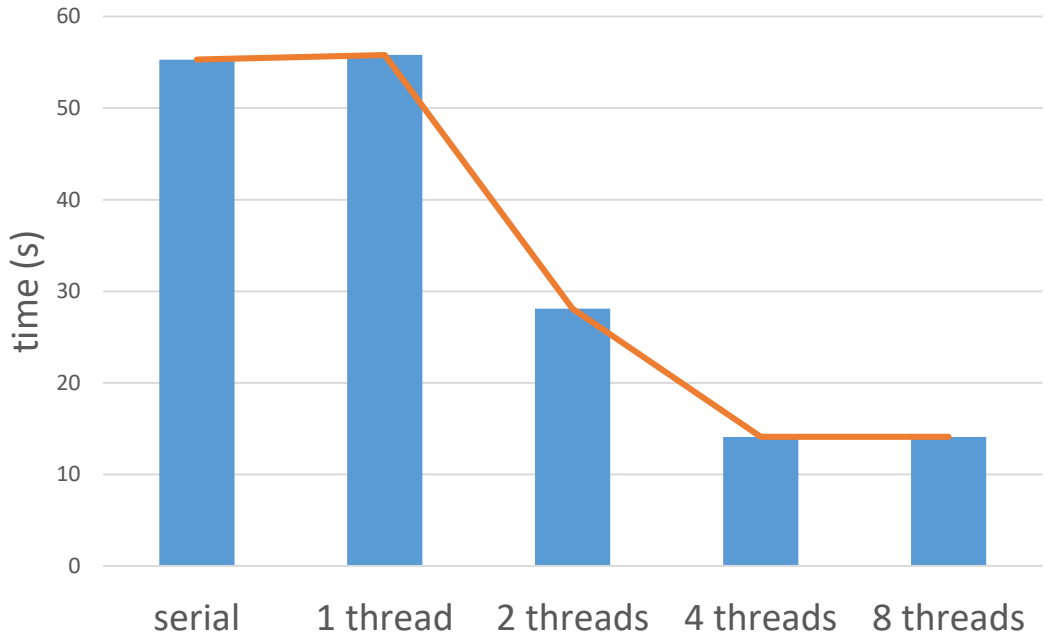
Current Status

- SNiPER Kernel
  - Revised potential conflicts in multi-threads
  - Keep APIs unchanged
- SNiPER Muster
  - Implemented as a non-invasive wrapper
  - Event level parallel processing with TBB tasks
- I/O problems, we are still using ROOT5 (will move to ROOT6)
  - Conflicts for concurrent reading and writing
  - Conflicts with our data lazy loading scheme

Performance Test

For the I/O issue, a dummy algorithm without I/O is used

4-cores CPU: Intel(R) Core(TM) i7-6700HQ



Summary

- SNiPER has been successfully applied to several non-collider experiments
- SNiPER Muster is implemented based on Intel TBB for multi-threaded parallel computing
  - More @ Poster #70 by Tao: Parallelized JUNO simulation software based on SNiPER
- We are pushing the I/O features according to our design

Thanks !