A scalable and asynchronous detector simulation system based on ALFA (FairMQ) — DRAFT 25.6.18

Sandro Wenzel (CERN), for the ALICE collaboration

see Ivana's talk for complementary approaches

The ALICE simulation environment: From Run2 to Run3

SimEngines {Geant4, Geant3, FLUKA}

Virtual Monte Carlo Layer

ALIROOT

Detector Description (TGeo)
Physics Modelling (Hits)

Services: MagField, IO, VMCApplication, Logging, EventLoop, etc.

Run2

The ALICE simulation environment: From Run2 to Run3

SimEngines {Geant4, Geant3, FLUKA}

Virtual Monte Carlo Layer

ALIROOT

Detector Description (TGeo)
Physics Modelling (Hits)

Services: MagField, IO, VMCApplication, Logging, EventLoop, etc.

Run2

ALICE-O2

Detector Description (TGeo)
Physics Modelling (Hits)

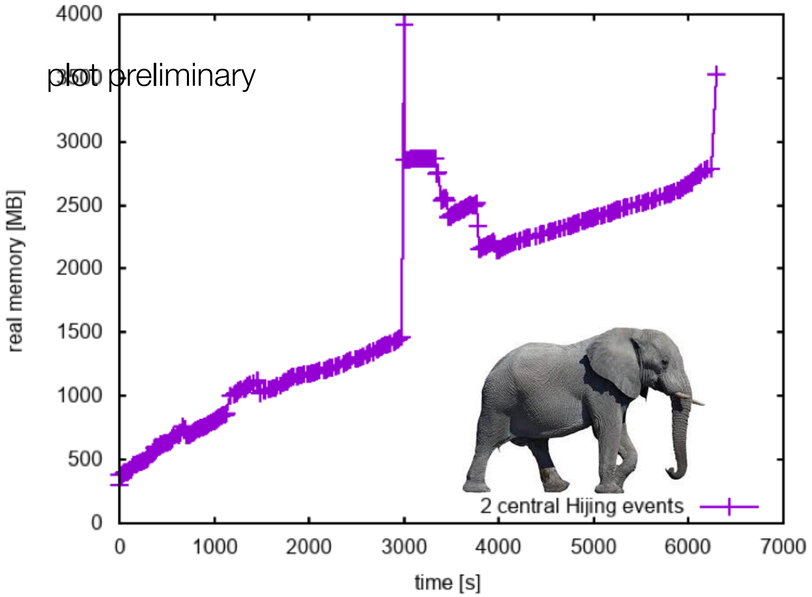
FairRoot **FairMQ**

Run3

“Keep the big picture”; “Make in-house code smaller”; “Potentially benefit from new developments (e.g., FairMQ)”

Motivation: The ALICE simulation scale

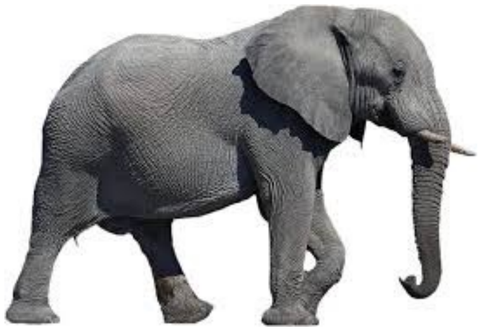
- **Simulating Pb-Pb collision** can be very demanding
  - may have up to 100k primaries in the collision to transport
- **Heavyweight resource utilization** using standard single-core event based simulation (FairRoot)
  - **~O(GBs) of memory / event**
  - **may be ~O(h) of CPU time / event**



- **Consequences:**
  - typically bad for scheduling and efficiently using given resource (packing problem)
  - prevents access to (opportunistic) HPC
  - sub-optimal user experience

The goal : Improve on this situation

- **Goal:** A simulation system running on anything from laptop to many-core and HPC facilities
  - get **result faster** (if resource available) for any given event
  - be able to **utilize smallest opportunistic resources**
  - **support VMC** (not relying on a particular simulation engine)
  - **user convenience** (same events; same output file regardless where and how run)
- **Ingredients put forward here:**
  - **independent actors** based on heterogenous multi-processing and message passing
  - **event splitting** and **collaborative simulation parallelism**



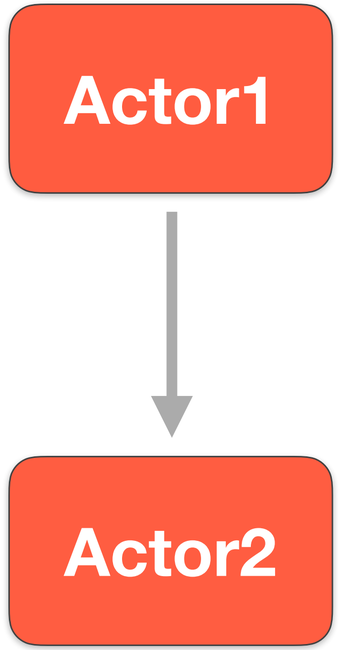
How? Use FairMQ as foundation

ALFA talk

DPL talk

- **FairMQ** = Fair(MessageQue) an abstract messaging library
- **Enables systems of heterogenous actors** which can be used to have
  - **asynchronous** and **parallel computing**
  - easy **scalability** from single node to complex cluster
- Does not replace multi-threading but complements it
- **Easy to use** … in particular to strip apart existing applications

```
Send(channel, message);
```




```
OnData(Kernel);

// callback for incoming message
void Kernel(FairMQMessage message) {
  // process message
}
```

Ingredient 1: Separation of concern with FairMQ

- **Break existing monolithic simulation** into **few actors with specialized concern**
  - deployed on same or different nodes
- **We gain:**
  - **Concurrent** event generation, particle transport and IO

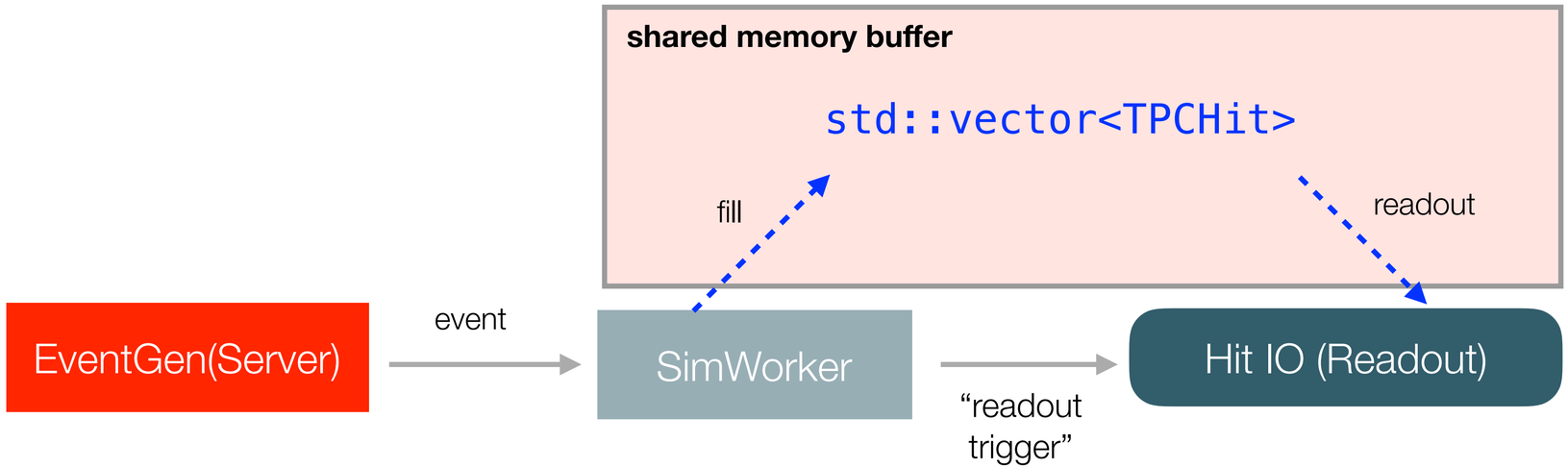
- **Is it easy? … Absolutely!**
  - Actors and communication setup in a few lines of code with FairMQ
  - C++ object exchange trivial thanks to ROOT serialisation (in principle no special care needed)
  - few days of work starting from existing code

Fast communication / shared memory

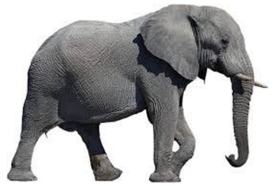
see posters of Sergey and Dennis

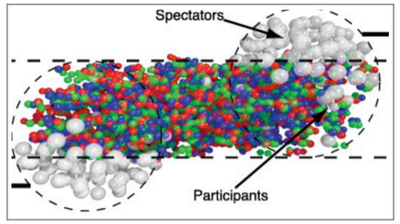
- **Some overhead when sending C++ structures**
  - need to copy/serialize/deserialize
- **Fast alternatives** such as shared memory transport within one node are **possible and supported by FairMQ**

- **Implemented a scheme similar to data acquisition:**
  - The sim workers directly put their C++ hits data into a shared buffer
  - Readout process streams directly from shared memory
  - Not a single copy; not a single pack/unpack; …. but some need for syncronization



Ingredient 2: Event splitting



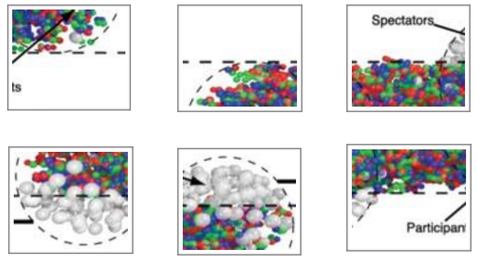


full event

EventGen(Server) → **SimWorker**

sub-events

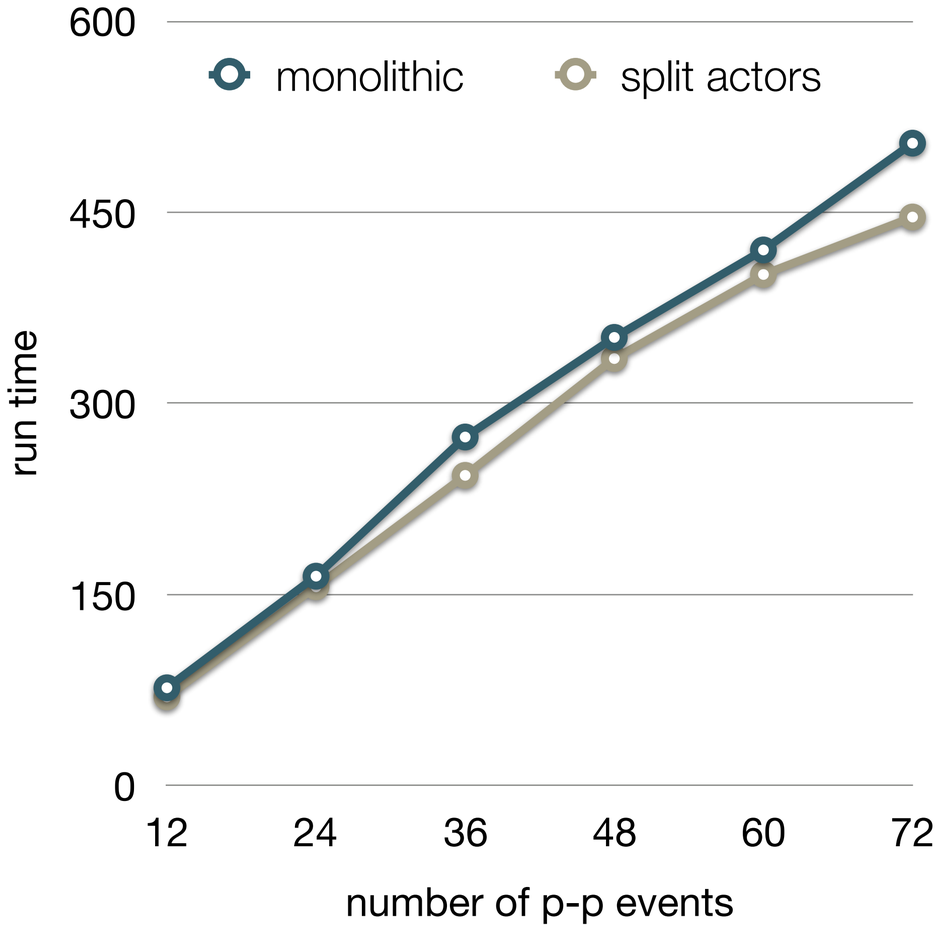
(one at a time)



- **Event-Splitting:**
  - provide less resource demanding work items
- **What do we gain?**
  - lower memory profile; **potential for collaboration**; fit work to small time windows
- **Is it easy?**
  - yes, since primaries are independent
  - done also by ATLAS in G4
  - might just complicate bookkeeping (MC truth)

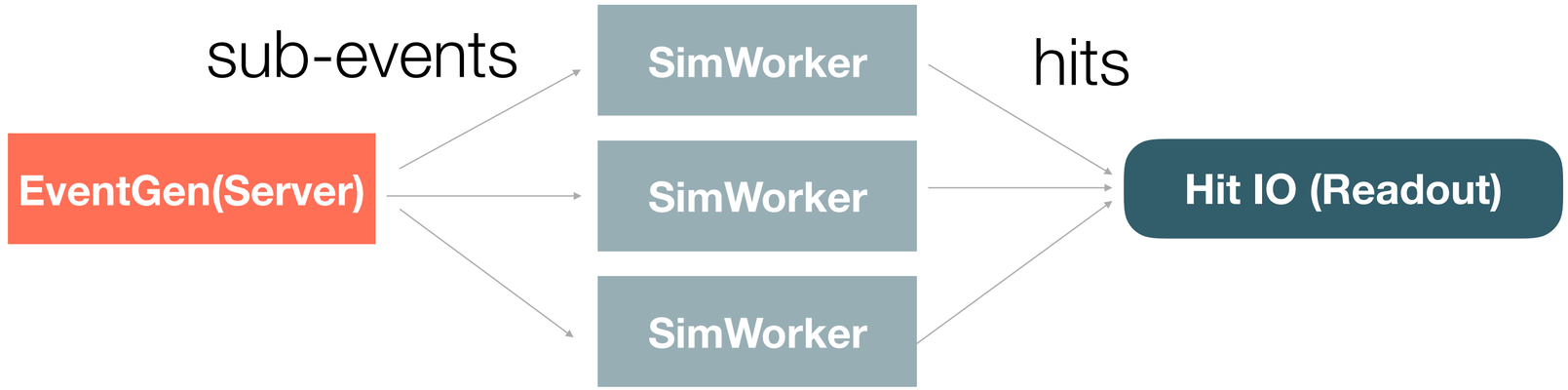
Benchmark single worker

- Is this **beneficial straight away** or is there an **overhead**?
- **Works well** with typical 14TeV p-p events:
  - gain few percent from split asynchronous components
  - gain higher than transport cost
  - essentially no additional gain from shared memory in this scenario
- A different test with large Pb-Pb also shows **no negative impact due to sub-event splitting.**



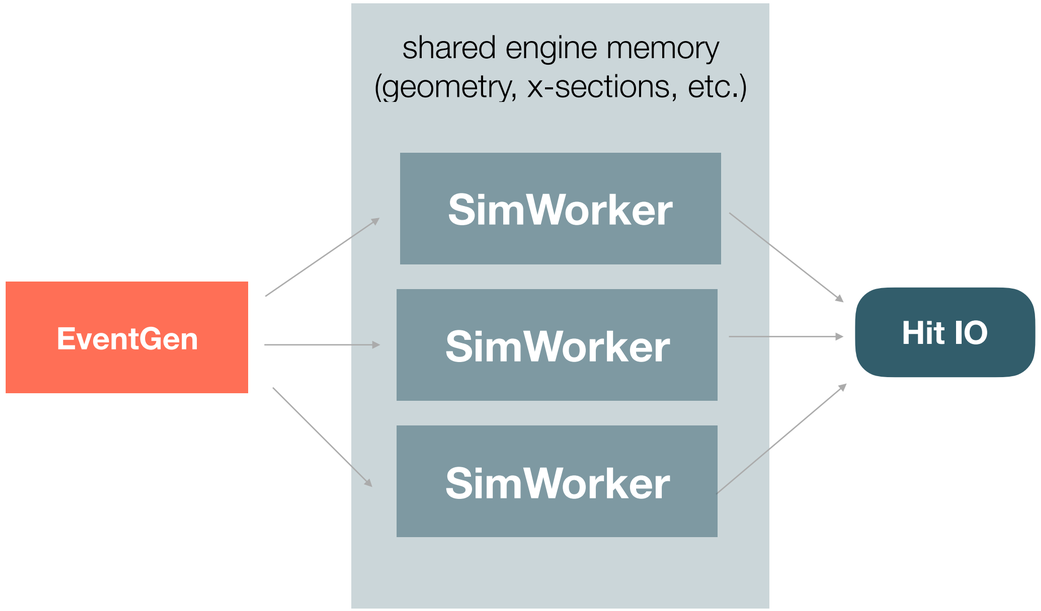
Ingredient 3: Collaborative Parallel Simulation

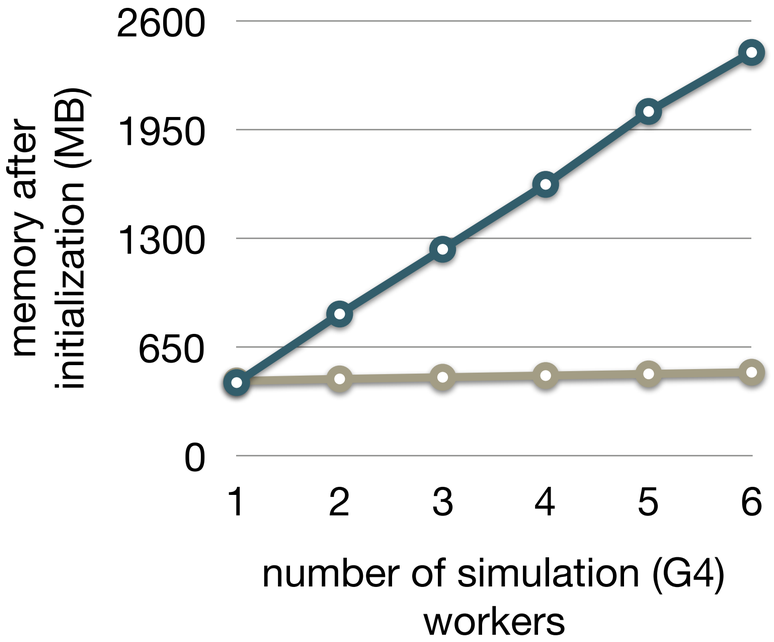
- Based on the previous steps, the idea is to **fan out the number of simulation workers**
- Target a diamond workflow for user-experience (but easy to change)
- **Scenario, where workers collaborate on transport/simulation**
  - given set of events
  - even on same event via (sub-events)



A scaling study : Memory

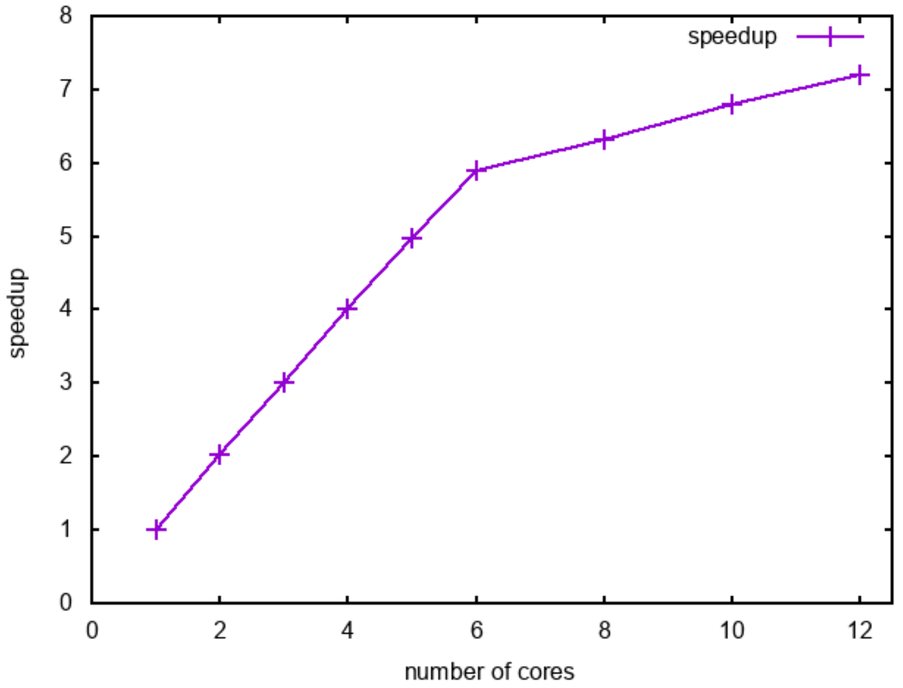
- **Isn't there a memory problem with too many workers?**
  - after all a single G4 worker has ~500MB of initial memory usage (geometry, x-sections etc)
- **Absolutely not!**
  - **implemented mechanism** based on **late forking** letting all simulation workers share the same simulation setup (geometry, x-sections) … "copy-on-write"
  - **works exceptionally well:** Tests with G4 shows that each additional worker is essentially free





A scaling study : Speedup

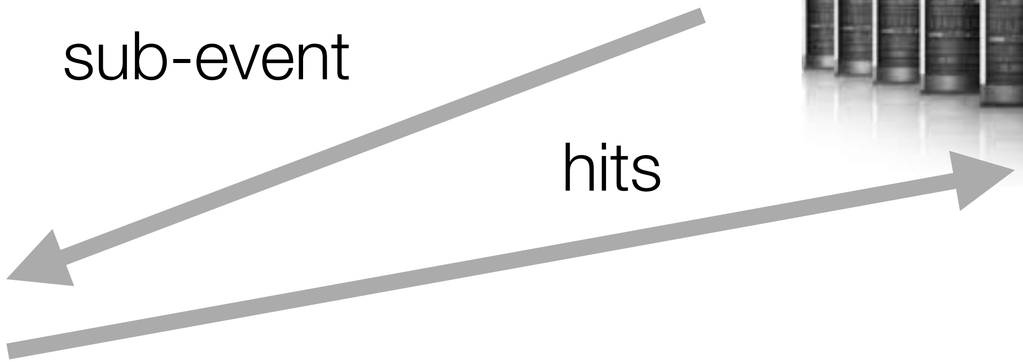
- **So what about the speedup?**
- Tested system collaborating on 1 large Pb-Pb event (60K primaries) as a function of the number of workers
- **Very good scaling** up to number of physical cores …
- … big Pb-Pb **events are now accessible in a few minutes**
- Together with good memory behaviour, this enables **scheduling on (opportunistic) HPC resources**



Preliminary; more (systematics) tests on larger systems in progress

Elastic (and Volunteer) Computing

- Architecture is the typical foundation for **elastic computing which we get as free lunch (really!!).**
- The number of workers can dynamically increase/decrease **at any time**.
- **Volunteer workers** can attach **anywhere** to a running production.
- Probably not a primary goal but nice to have.




```
“o2sim_join -eventserverIP xxx -hitmergerIP yyy"
```

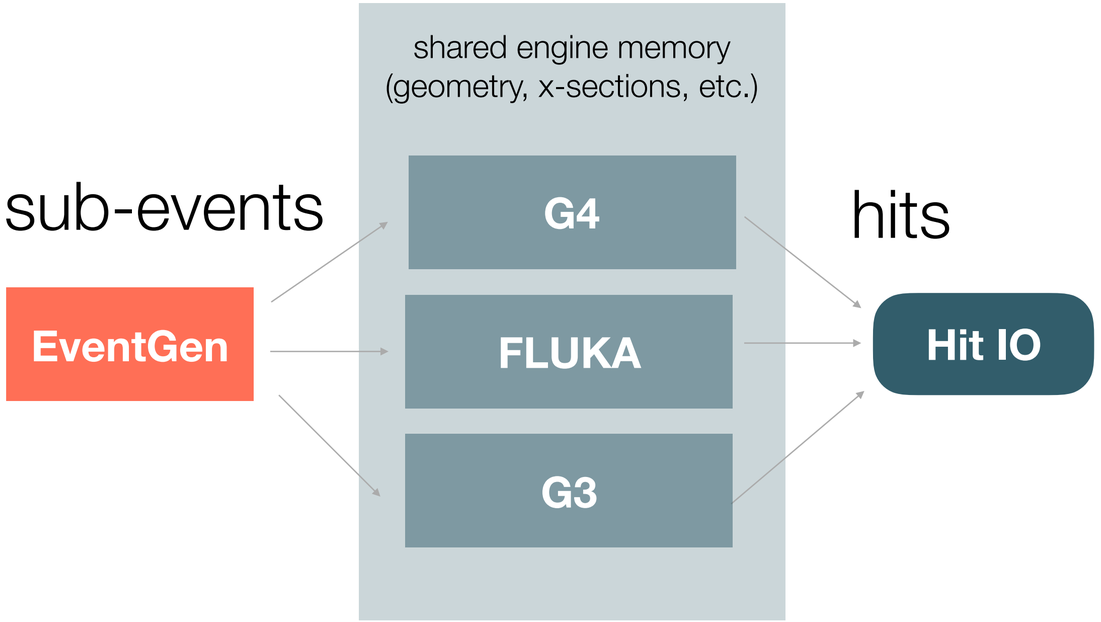
The final picture

- **In the novel ALICE simulation based on FairMQ …**
- **… we parallelized Geant4, Geant3 and FLUKA** at the same time
- … **may collaborate** on a **single event**
- **… are HPC ready**
- … do event generation, transport and IO **asynchronously**
- … **provide user convenience**: single merged file, single source of events
- … **have elasticity**
- … **are agile** (can change deployment easily and scale across nodes)
- Can also combine with complementary approaches such as internal multi-threading of G4

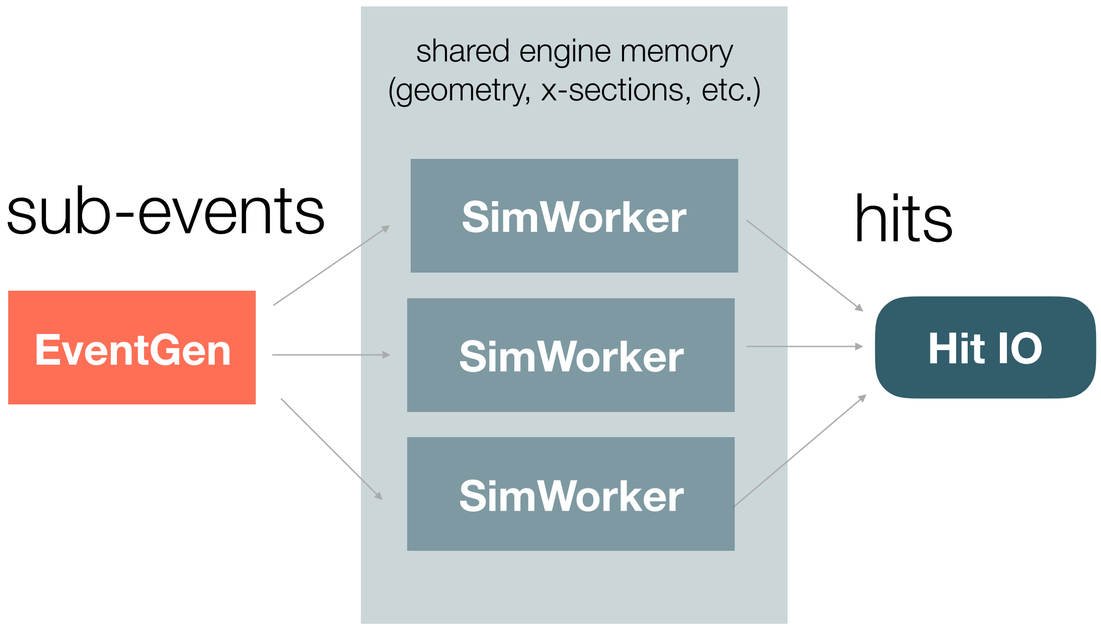
BACKUP SECTION

Heterogeneous Simulation

- An interesting extension is heterogenous computing: simulation workers can take different flavours
  - G3 / G4 / Fluka collaborate on same event
    - (e.g., as a function of particle type, energy)
  - Attach workers doing fast simulation kernels.
- ToBe developed.

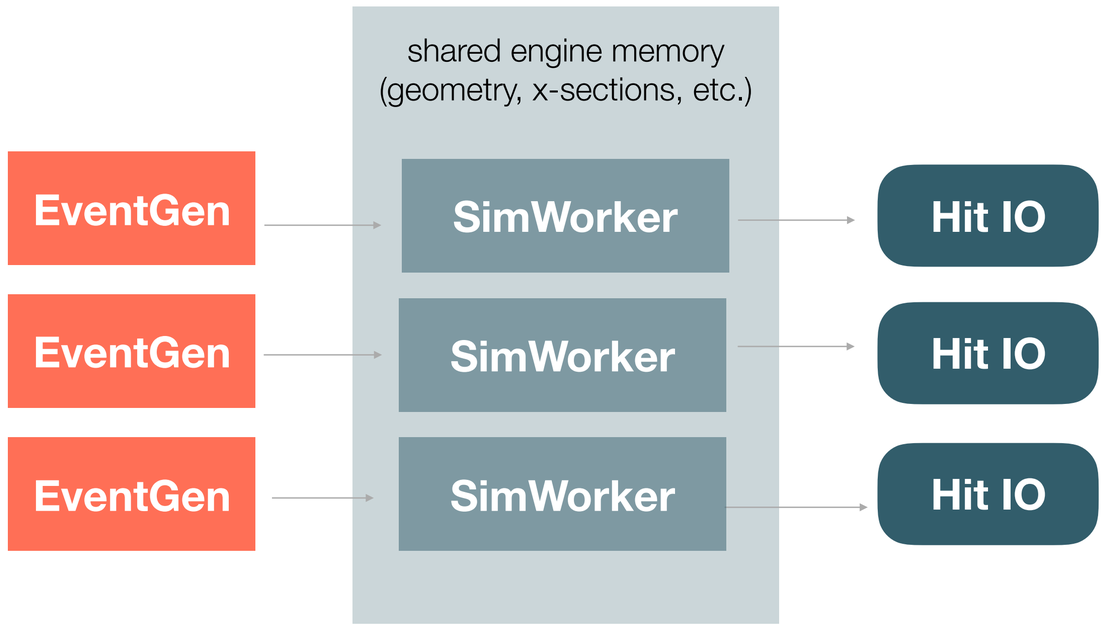


Collaborative Parallel Workers or Trivial Parallelism?



collaborative; user friendly; better to transport large events quickly

easy switch



more traditional; might be better in certain situations such as small events

Deployment tests with DDS?

Coupling to data processing

- We can directly couple simulation to DPL data processing

Integrated fast/full simulation framework