

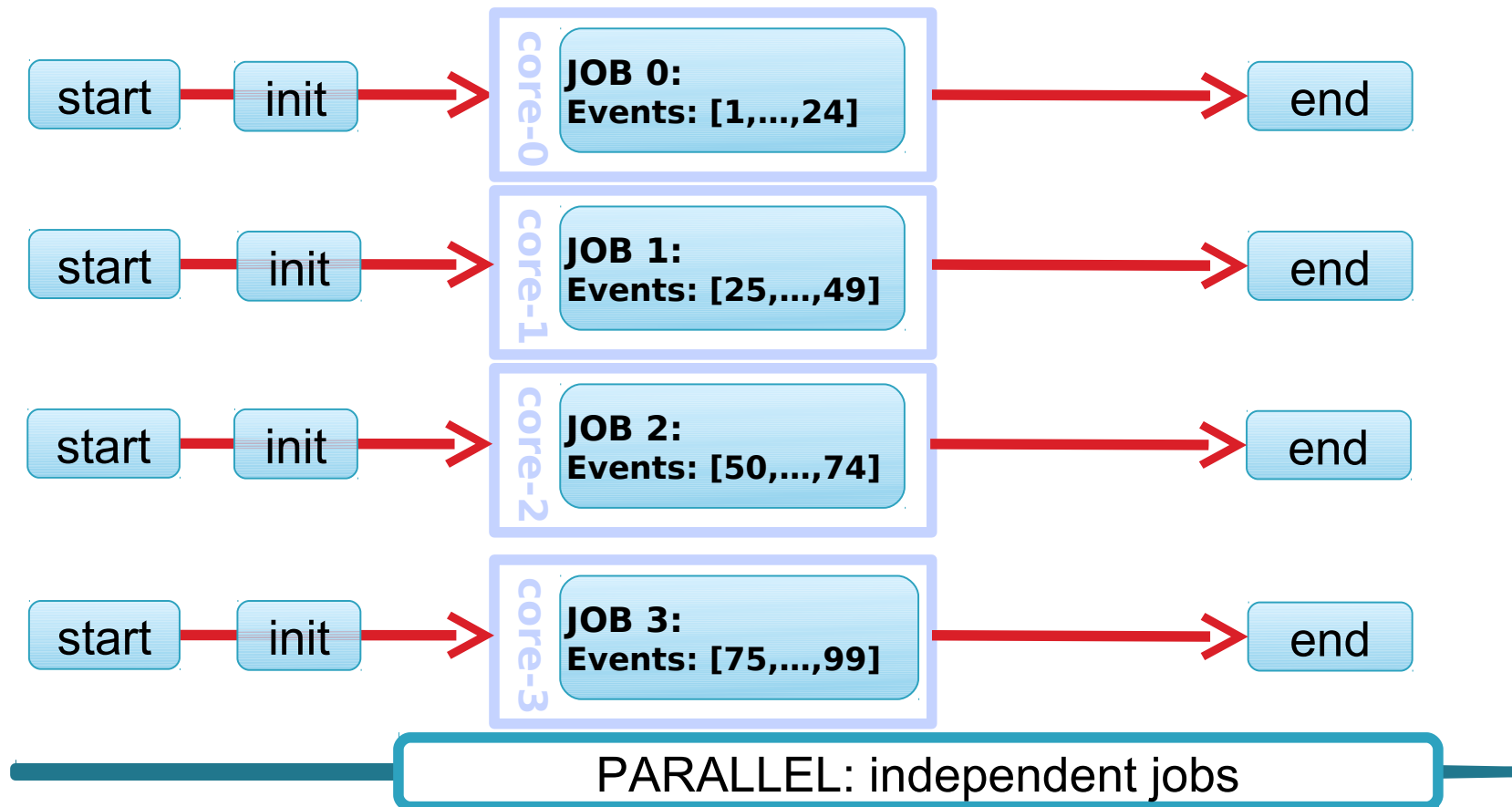
---

# **AthenaMP and the US ATLAS Facility**

**Paolo Calafiura**

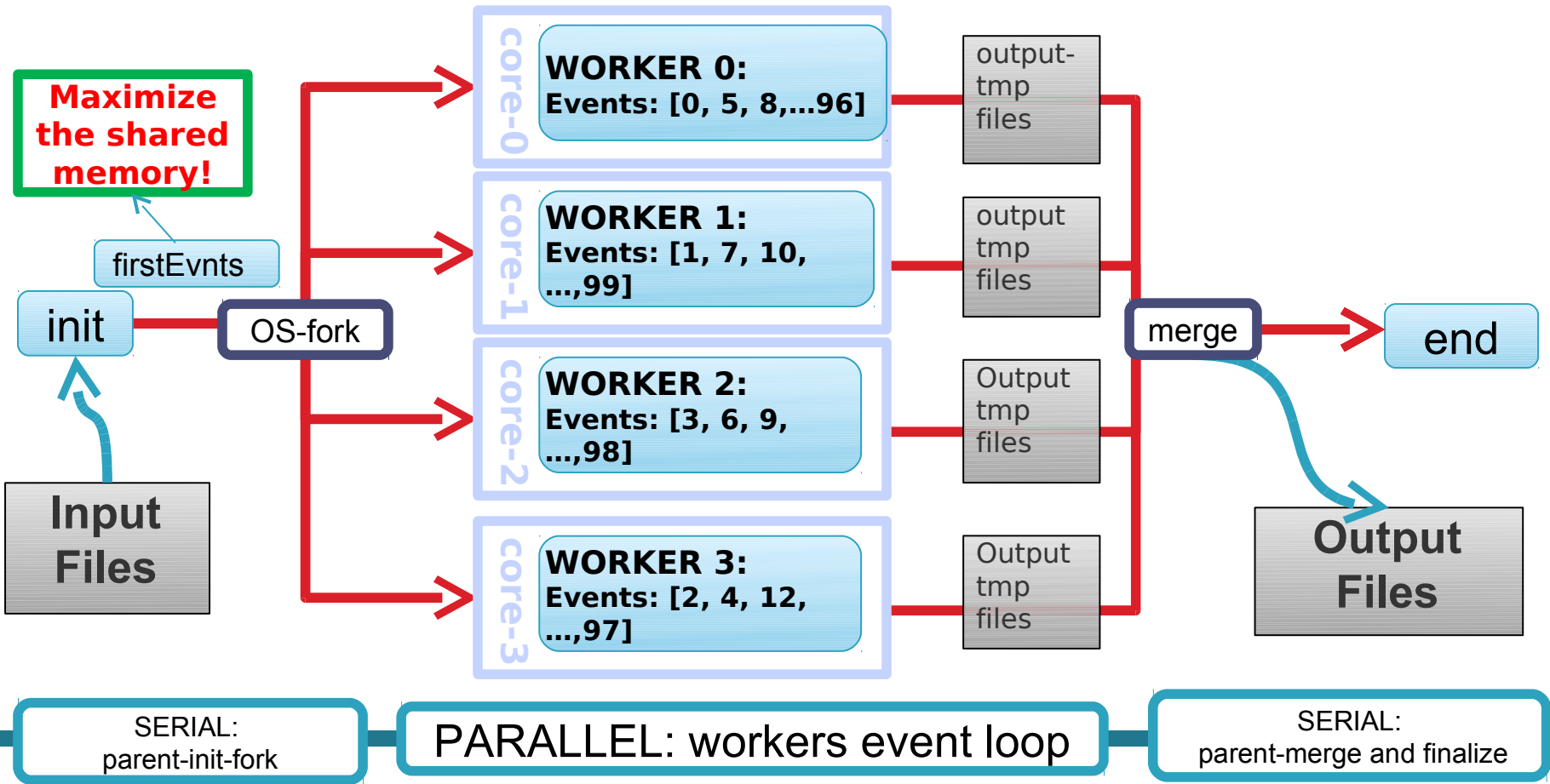
# Today: athenaMJ

```
for i in range(4):  
    $> Athena.py -c "EvtMax=25; SkipEvents=$i*25" Job0.py
```



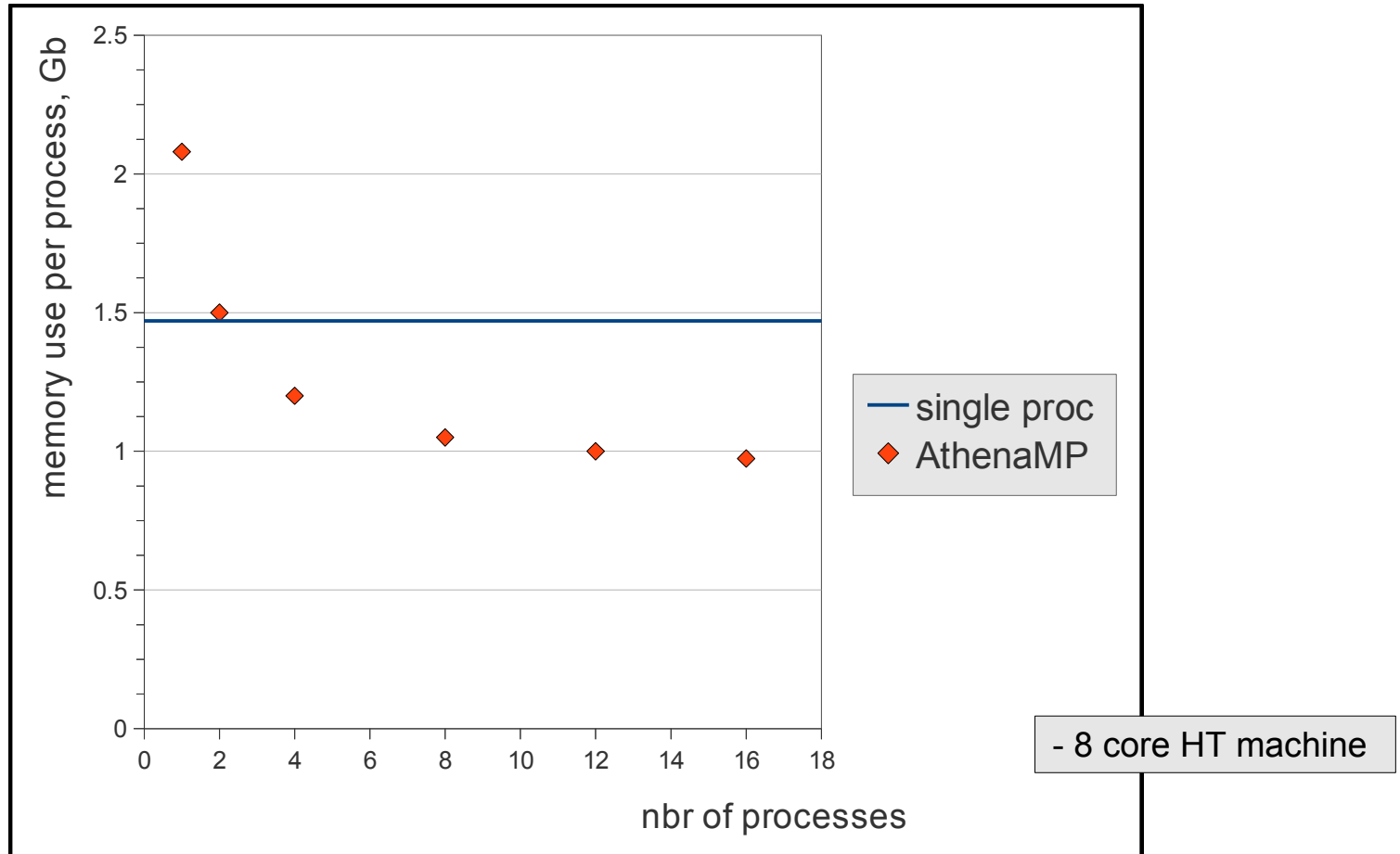
# Event Level Parallelism with AthenaMP

```
> Athena.py --nprocs=4 -c EvtMax=100 Jobo.py
```



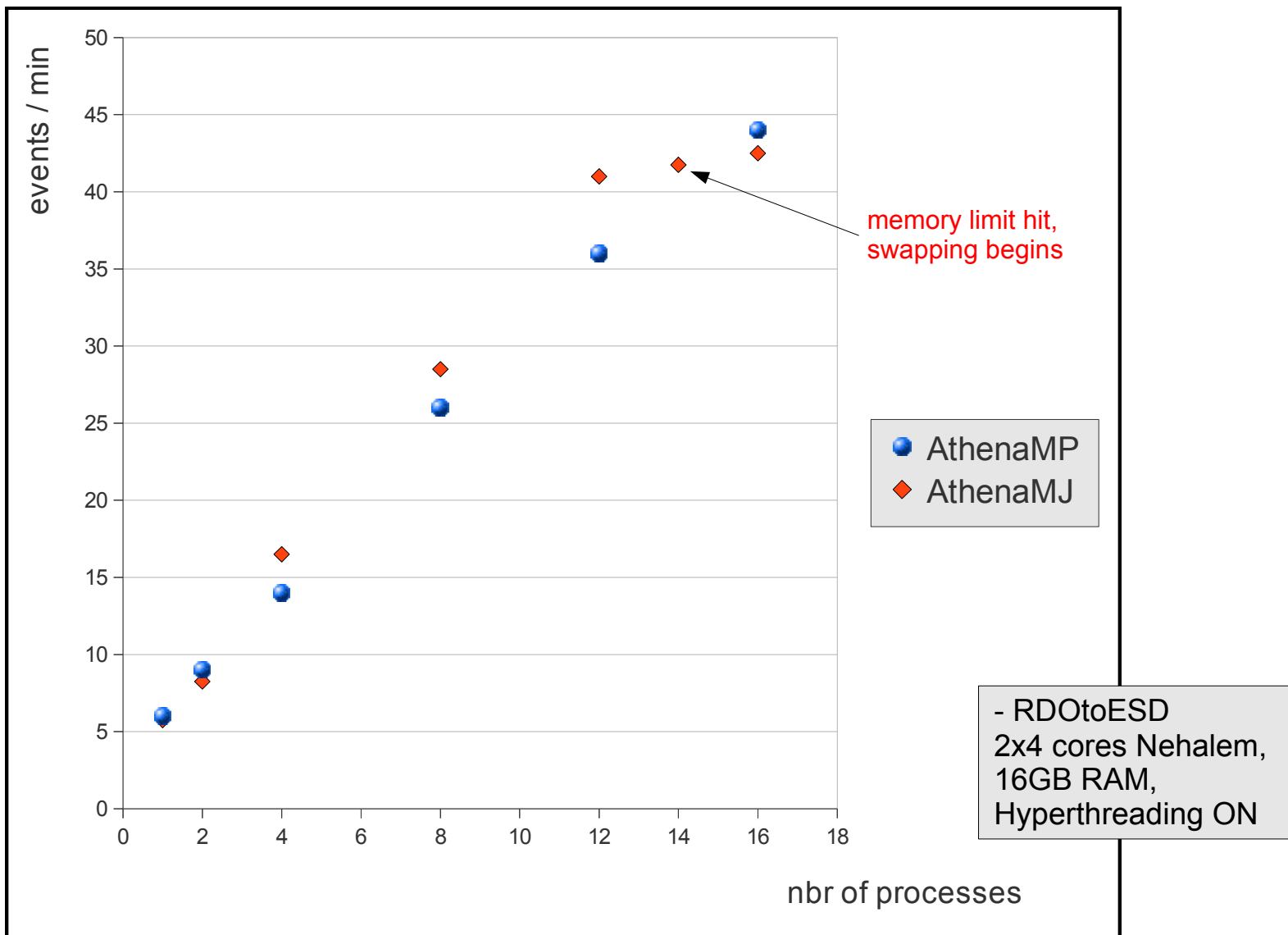
# Why AthenaMP?

- Main goal is to reduce overall memory footprint
- Use Linux fork() to share memory automatically



**AthenaMP ~0.5 Gb physical memory saved per process**

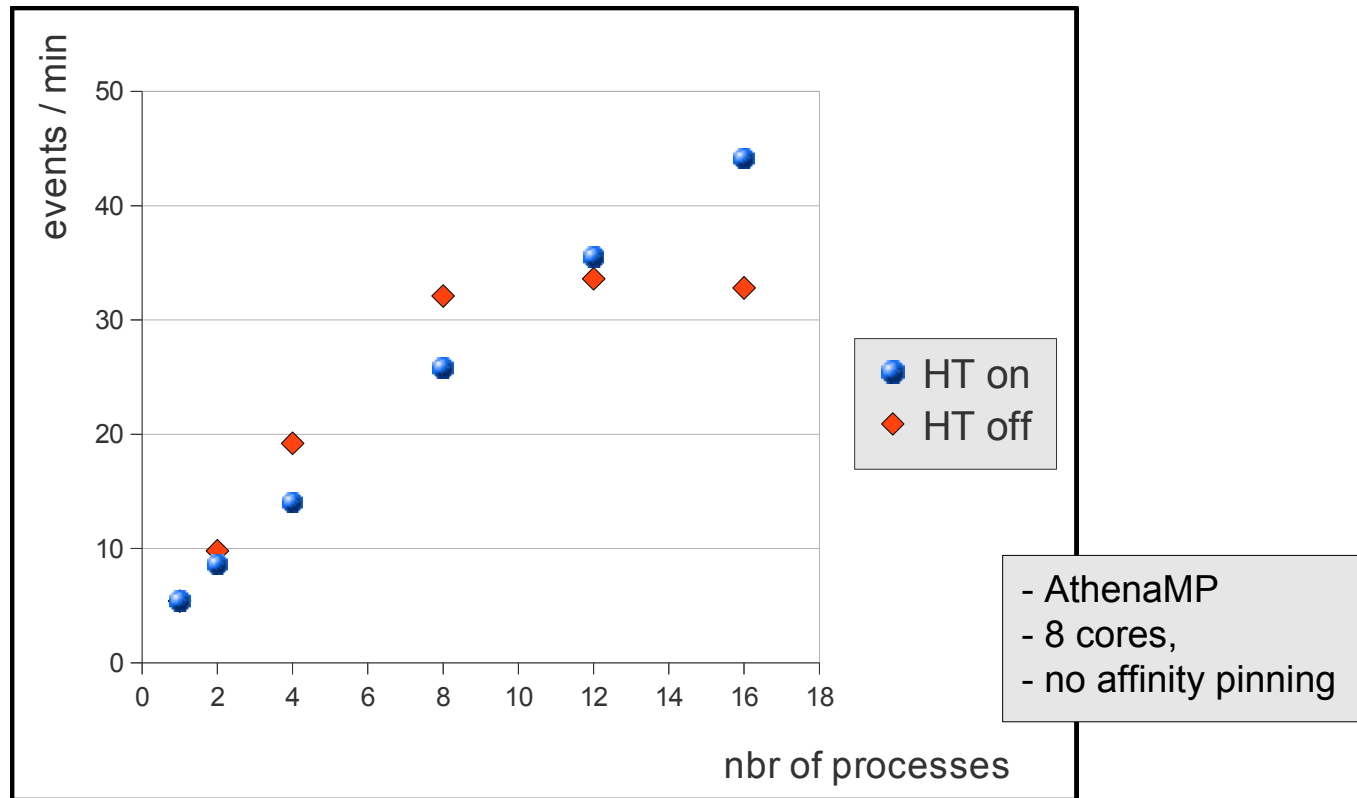
# ~Same Event Throughput



# Hyper-Threading

Nature of instruction pipeline allows instructions to be interleaved

- OS sees “virtual cores” as just another processor
- Only effective until one of the threads saturates a shared resource and stalls



**HT increases event throughput by 25%**

# athenaMP Testing

---

- Running small-scale nightly tests in 16.x.0 since (at least) January
  - 11 BStoESD and RDOtoESD jobs, both athena, and Reco trfs
    - No known athenaMP-related problem
    - No physics validation yet
    - No real test of metadata handling yet
    - Only basic testing of POOL “fast-merge” mechanism (Peter VG, used to speed-up merging of event output files)
  - Just added G4 tests
    - Random number management changes sequencing used so far
- Last week a large scale test (1600 events, 8 workers) run out of memory!
  - TBC/investigated

# Production Testing and Validation

---

- Backporting athenaMP tags AtlasProduction-16.5.0.X
- Learning how to work with whole-node queues
  - Testing at CERN, OSCER, and, in the future, Edinburgh
  - Teething problems (e.g. whole-node jobs not starting in crowded queues)
- Not being able to run successfully yet (new cache 16.5.0.3 coming out today)
- Must complete before 17.x-based productions start



# What Next?

---

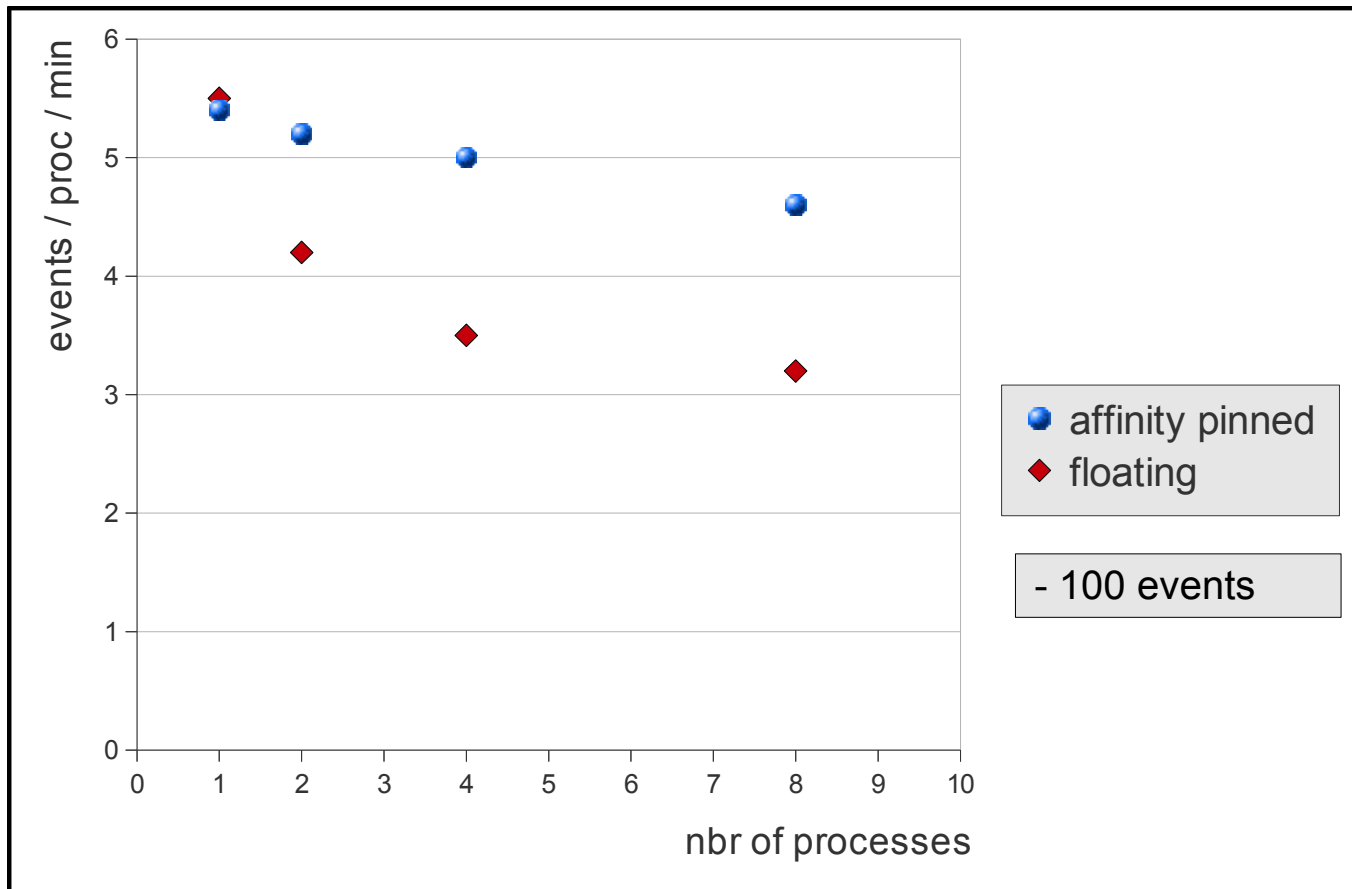
- File Merging Framework
  - Generalize what we are doing with POOL fastmerge
    - Of interest beyond athenaMP
- I/O processes
  - Event source
    - Read centrally data from disk. Deflate once, do not duplicate buffers
  - Event sink
    - Merge events on the fly, same memory, no fastmerge postprocessing
  - Predicated on ability to exchange events or data objects in bytestream(-like) format
- Sub-event parallelization
  - Specialized workers (~4 event): tracking, calos, monitoring, etc
  - Reduce memory footprint of each worker
  - Increase memory locality

---

# Extra

# CPU Affinity

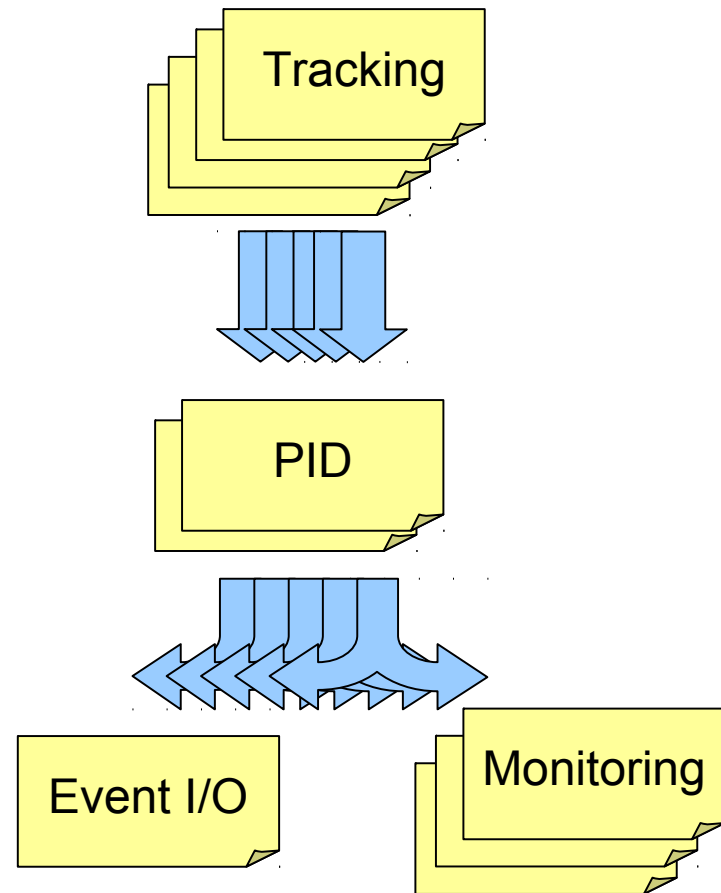
- Linux schedulers will move processes from core to core during the course of a job
- We can prevent this by pinning a process to a core via its affinity, and gain 20%



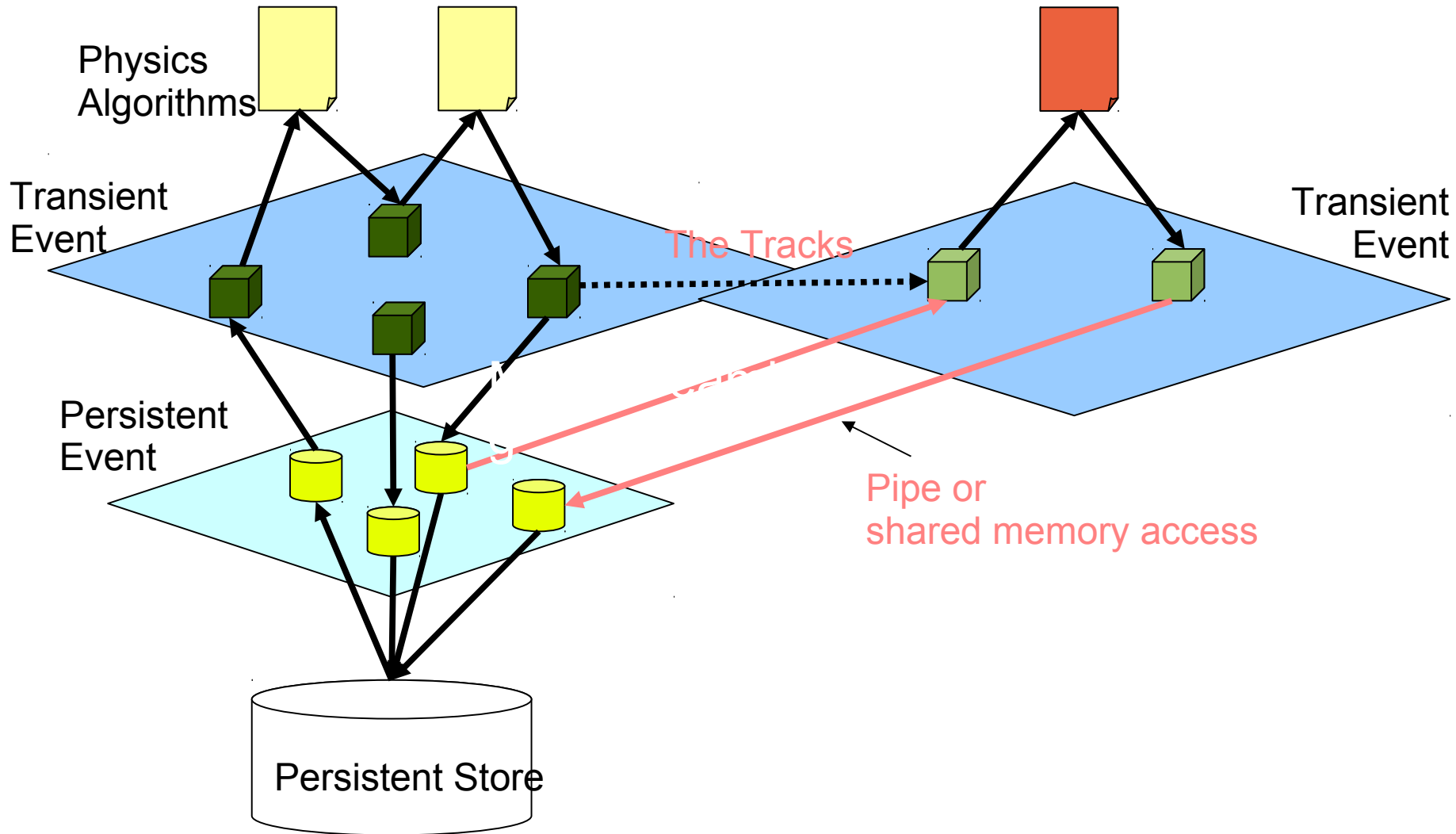
# Beyond Event Parallel

Many-core is pushing us to go task-parallel

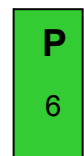
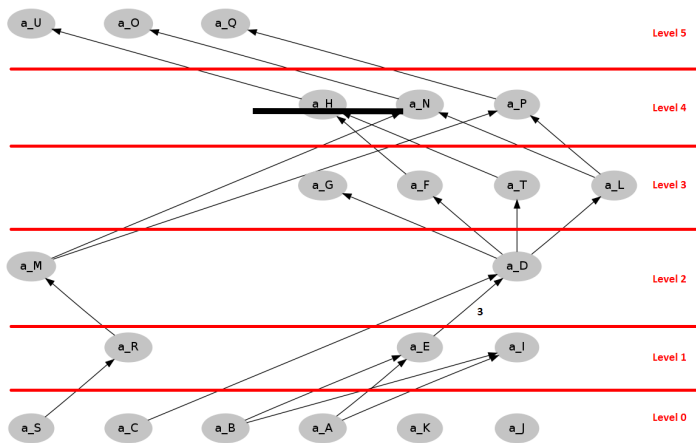
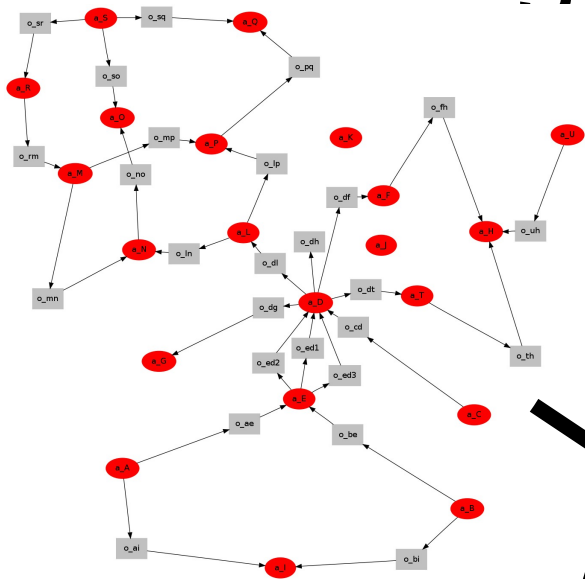
- Smaller processes
- Improved memory locality
- Potentially better caching of asynchronous I/O and other stalls



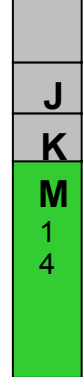
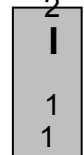
# Communication among sub-tasks



# Automated Sub-event Parallelism through Dataflow Analysis



Core 0



Core 1

Challenging optimization:

- Minimize memory traffic, load-balance cores