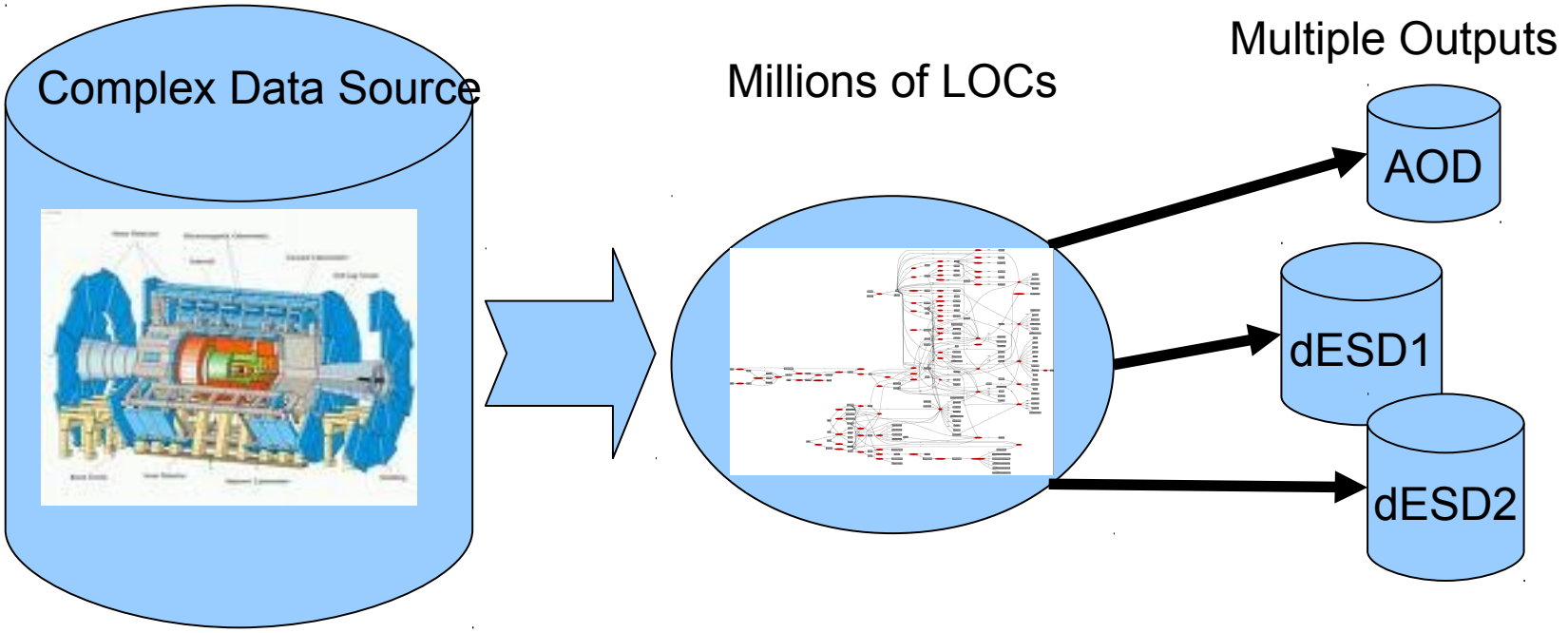
Multicore Computing in ATLAS

Paolo Calafiura

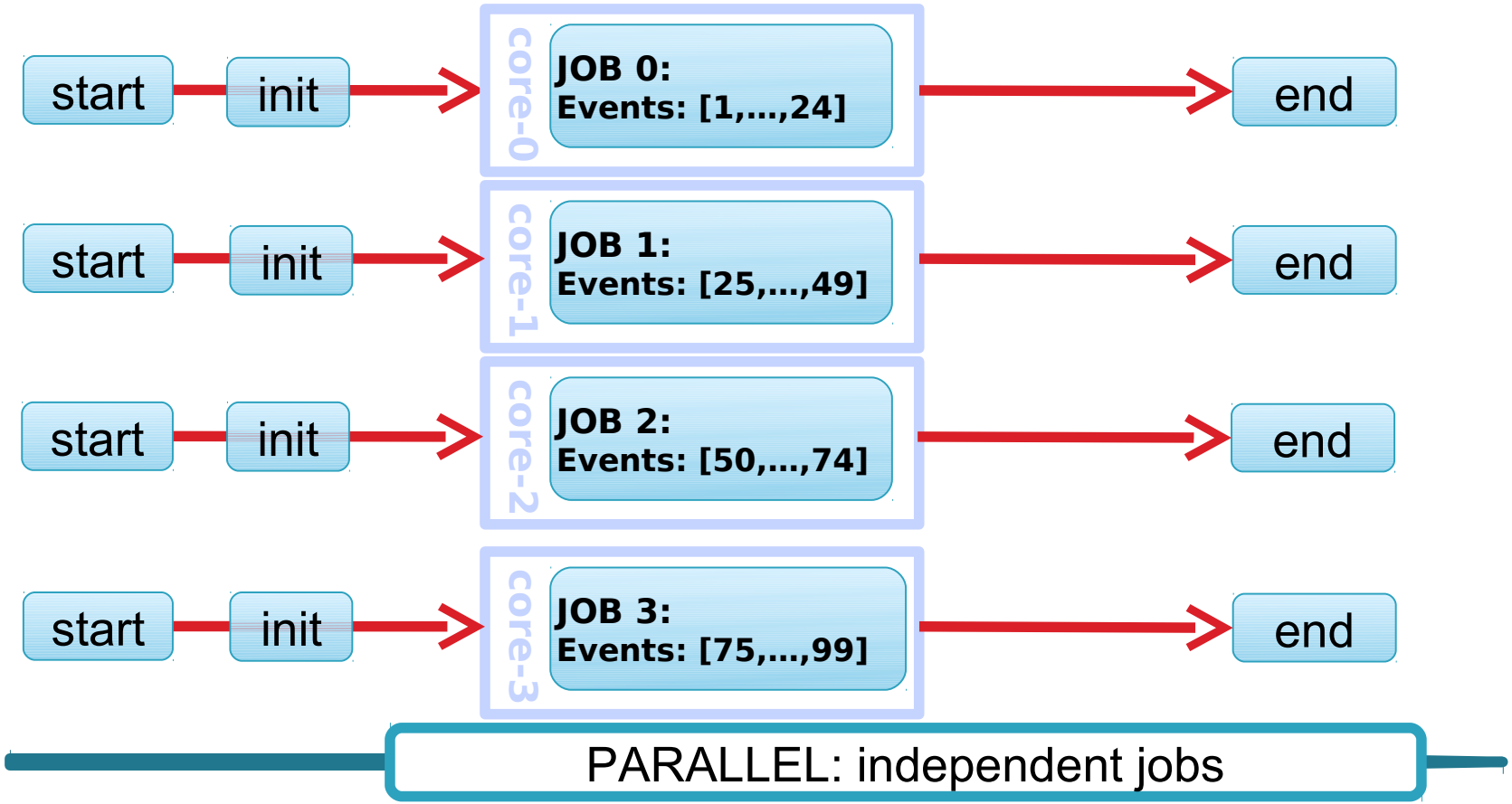
Why Embarrassingly Parallel?



- Our computational units are many, small, and very regular (not nearly enough black holes in our billions of events)
- **Low data bandwidth** (0.1-10MB/s)

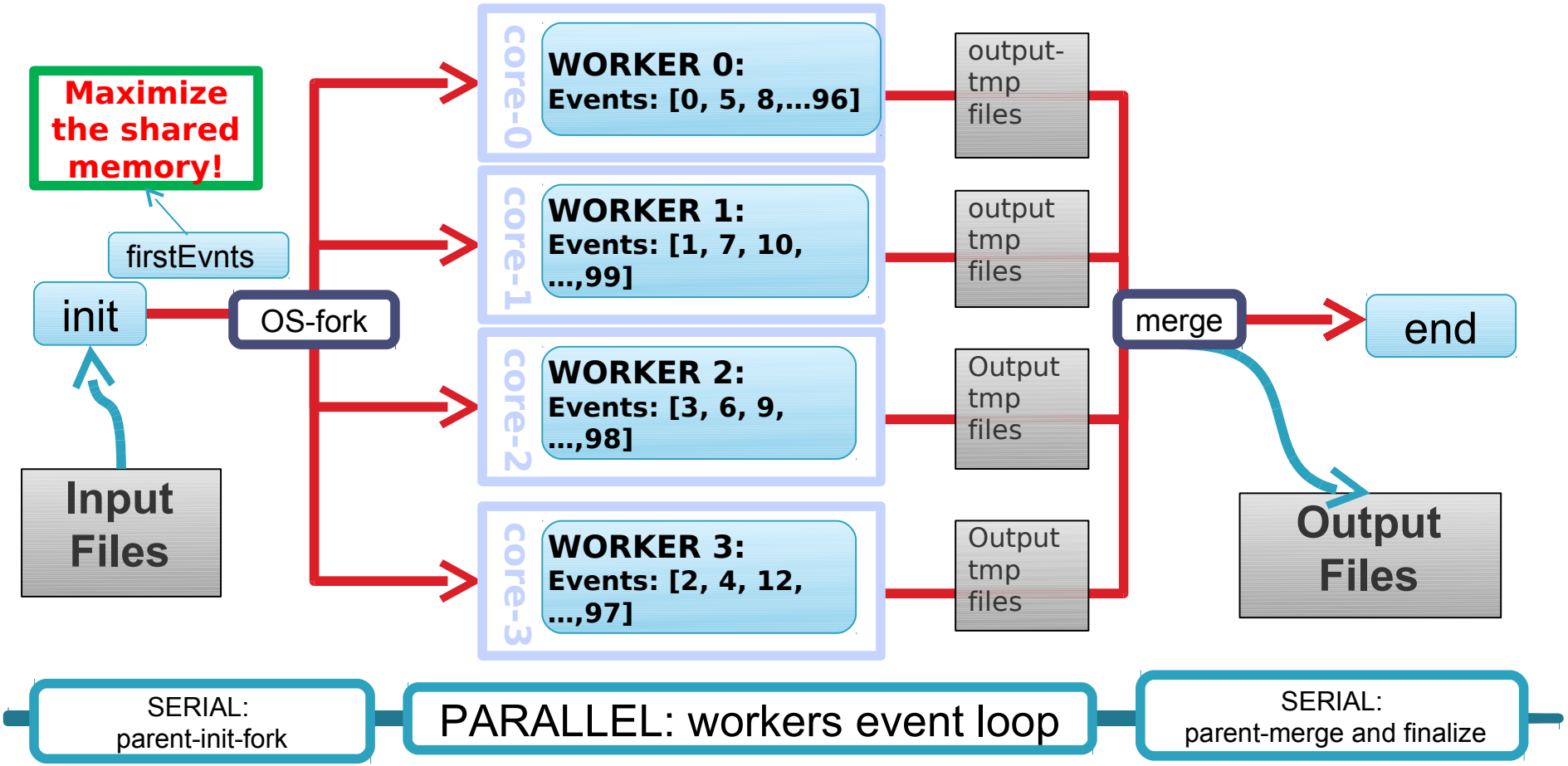
Today multi-job approach: athenaMJ

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

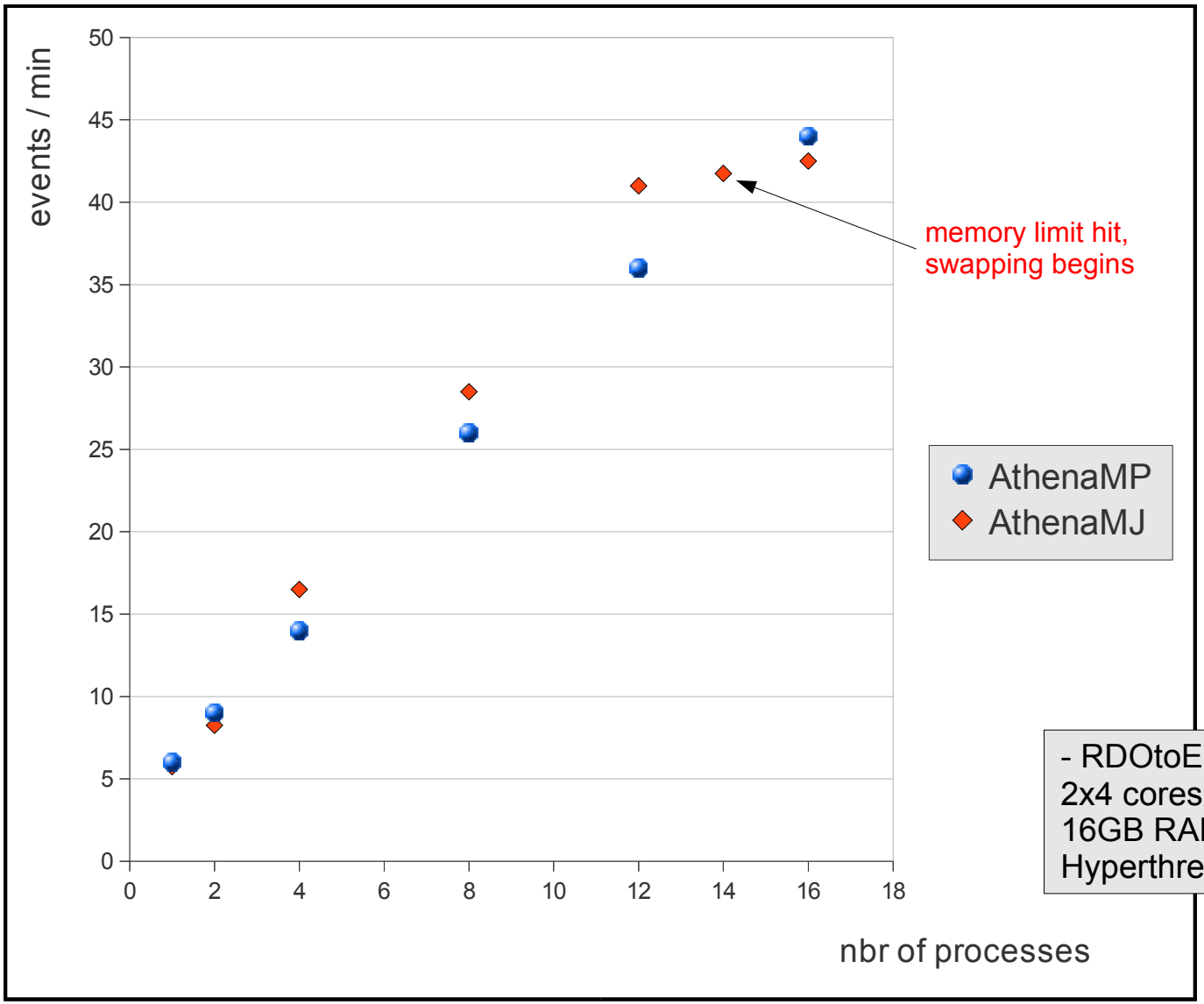


Event Level Parallelism with AthenaMP

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

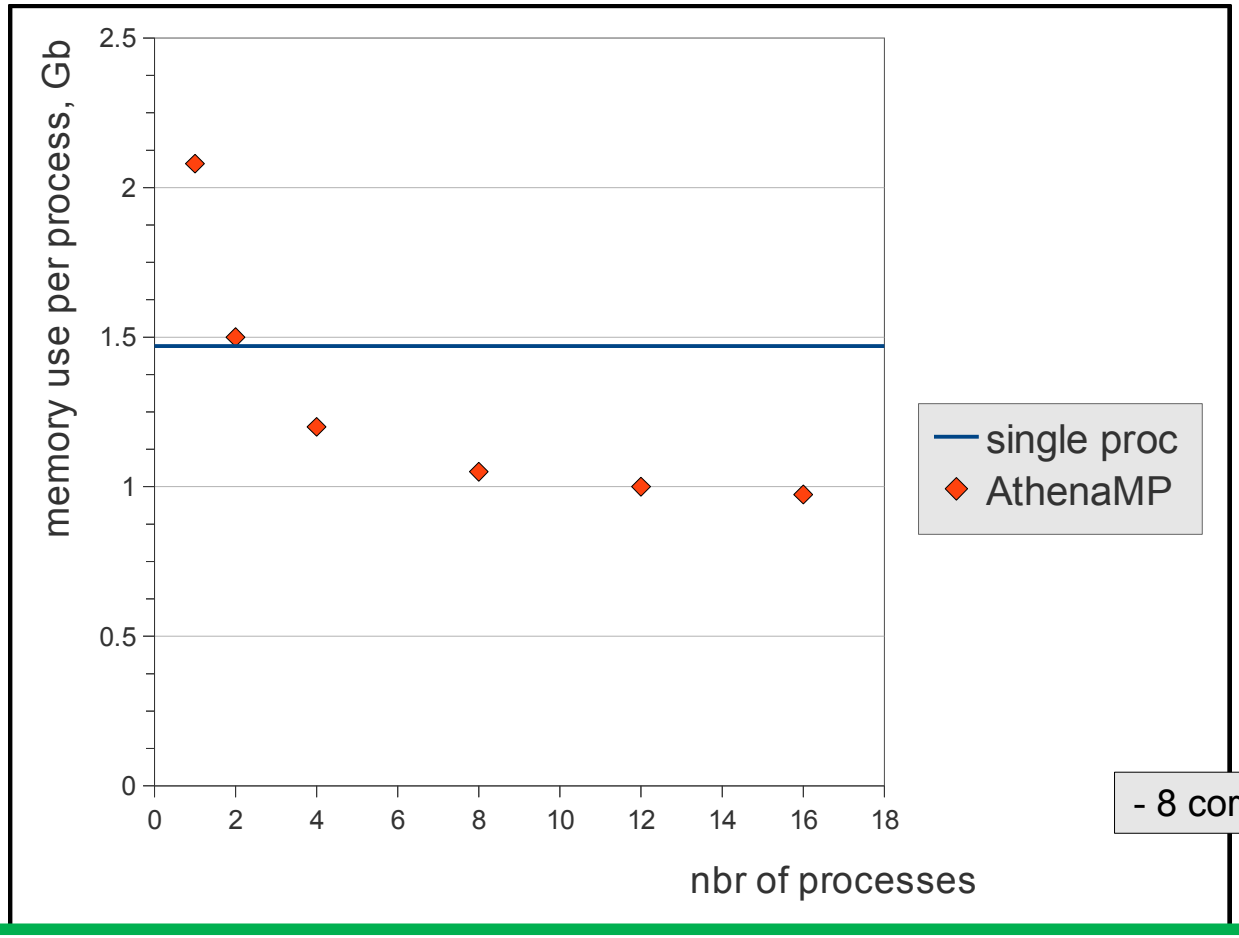


~Same Event Throughput



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

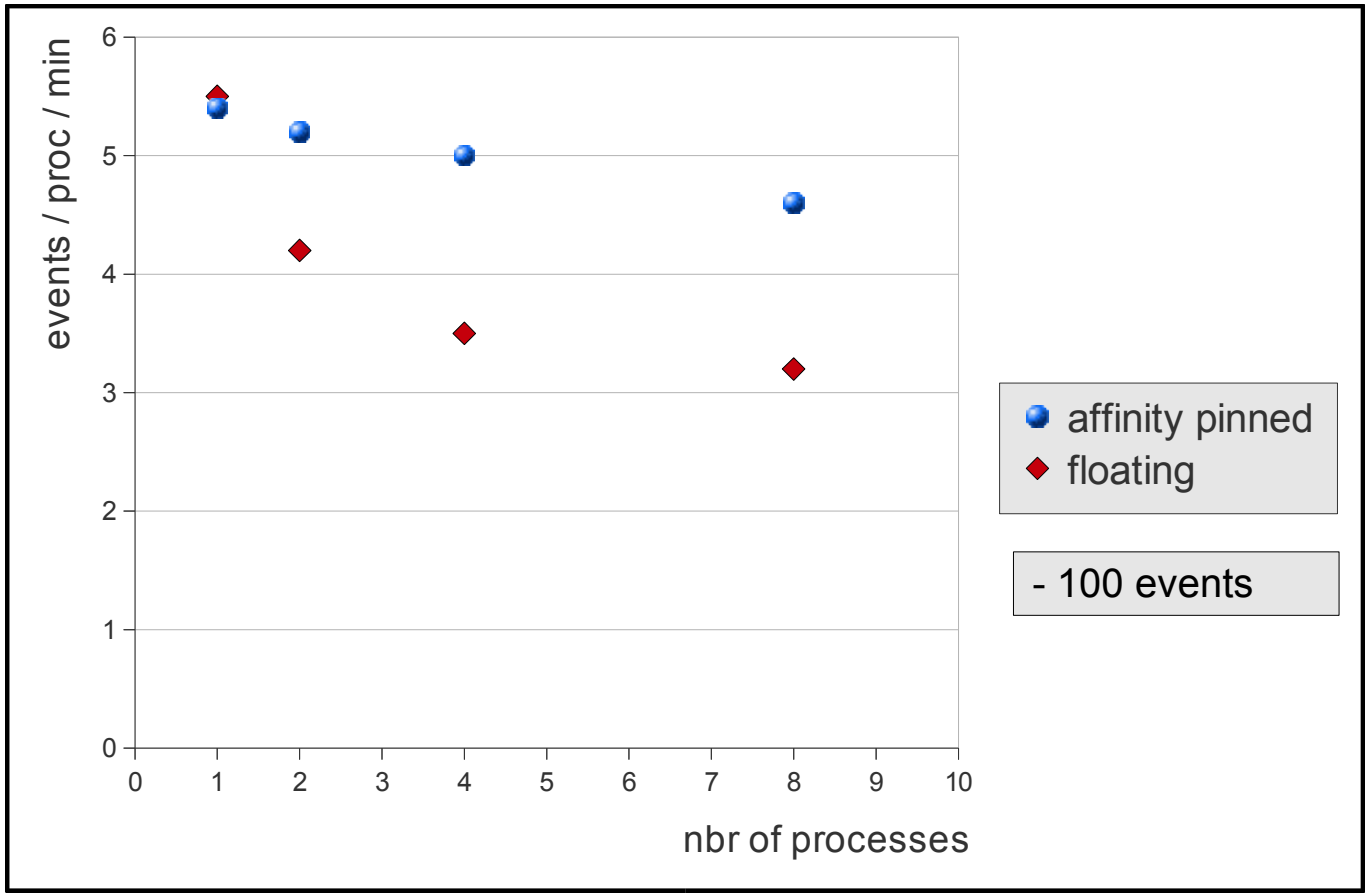
Why Multiprocessing?

Forking has annoying issues like memory unsharing, file merging, why not go multi-thread?

- ATLAS HLT started with GaudiMT in ~2002
  - Same event farming approach, threading code well-hidden
  - Never got beyond proof-of-concept
    - Too much code to port, too many developers to educate
    - Hardly an ATLAS issue: examples of MLOCs applications running reliably in MT are few and far between....
- Even with multi-processing took us four years to go from Scott's prototype to (almost) production-quality

Multicore Tweaks: CPU Affinity

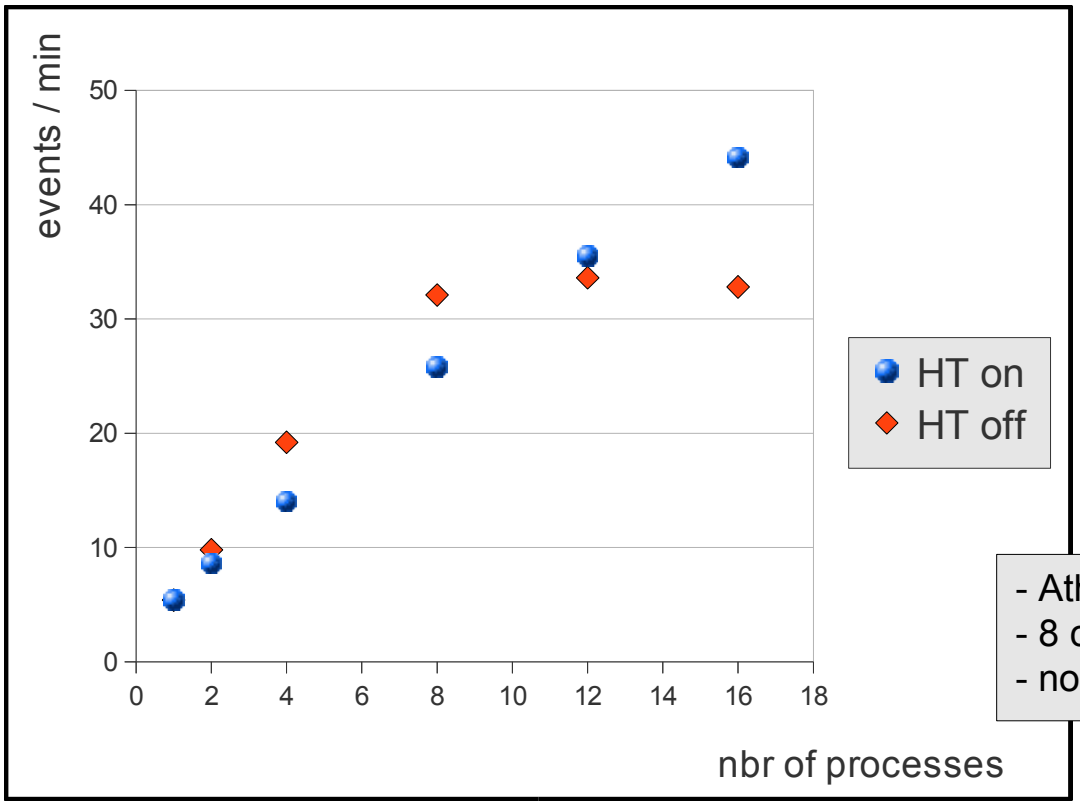
- Linux schedulers will move processes from core to core during the course of a job
- We can prevent this by pinning a process to a core via its affinity, and gain 20%



Multicore Tweaks: 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 in Production

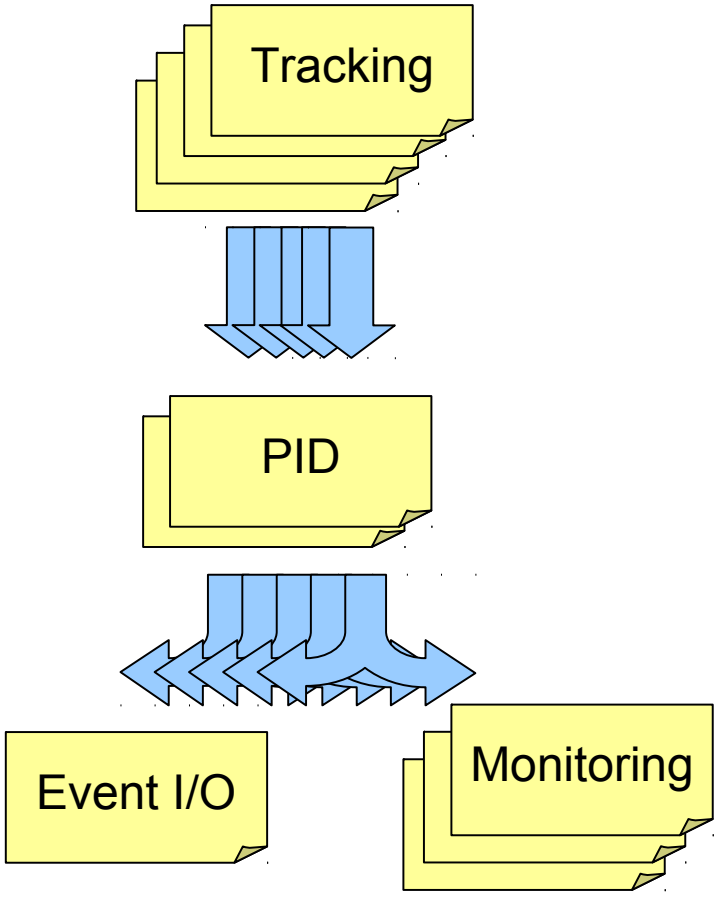
- Six months of focused effort, not quite there yet
  - Great improvements since Vakho started working on this full time last month.
- Details in Andy's talk

- Need of a File Merging Framework
  - Generalize what we are doing with POOL fastmerge
    - Of interest beyond athenaMP

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

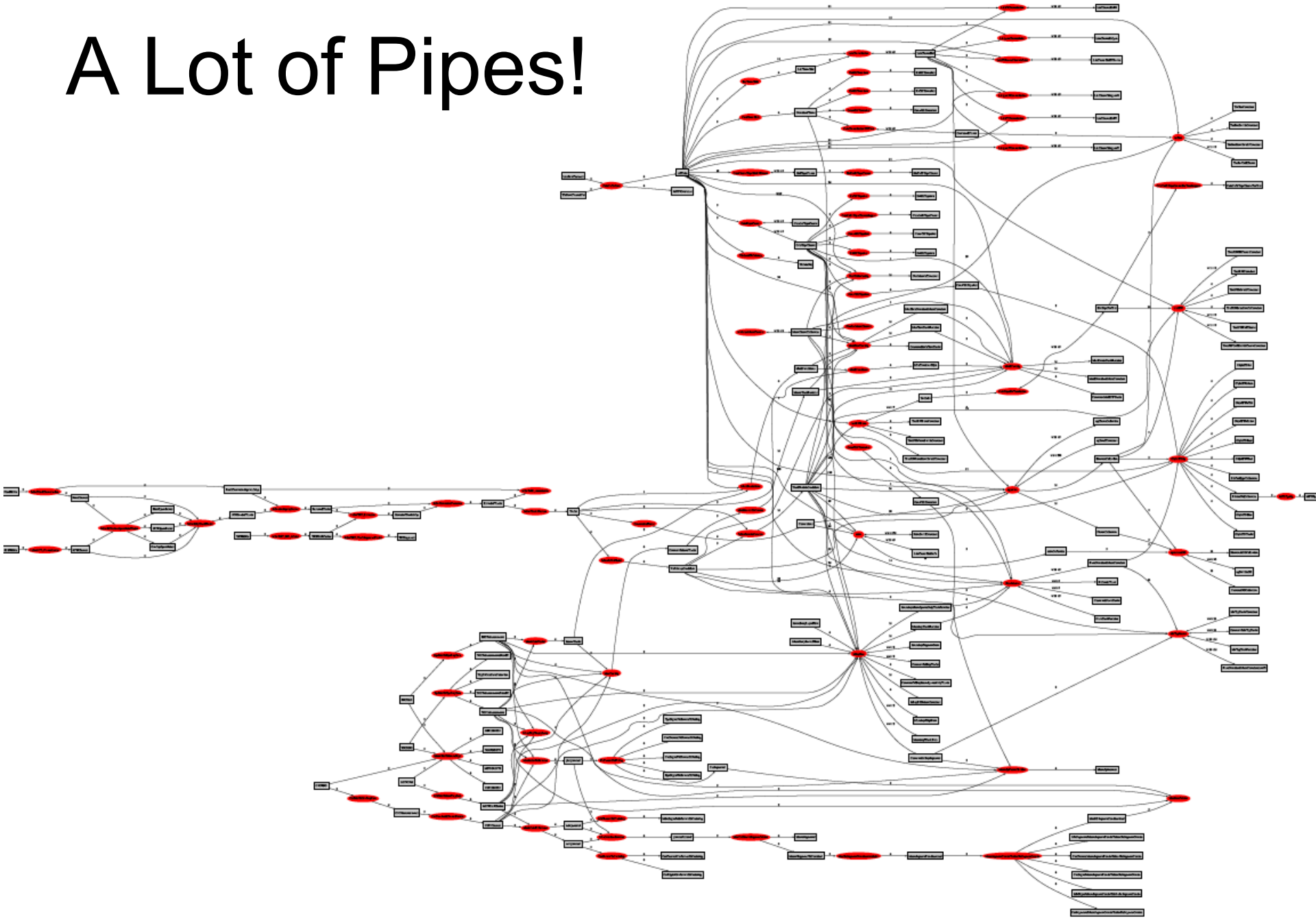


Dedicated I/O Processes

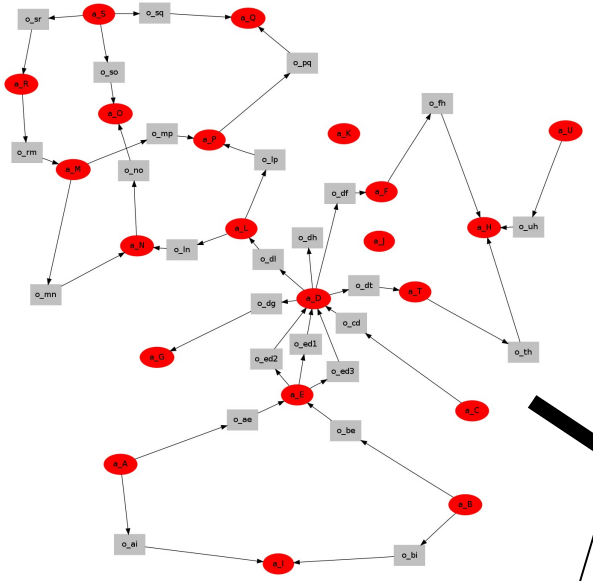
- Event source
  - Read centrally data from disk. Deflate once, do not duplicate buffers
- Event sink
  - Merge events on the fly, same memory, no fastmerge postprocessing
- Event worker
  - As before, minus all I/O dependencies (e.g. dictionaries)
- Predicated on ability to exchange events or data objects in bytestream(-like) format
- Most promising sub-event parallelization.
  Work will start this summer (next week!)

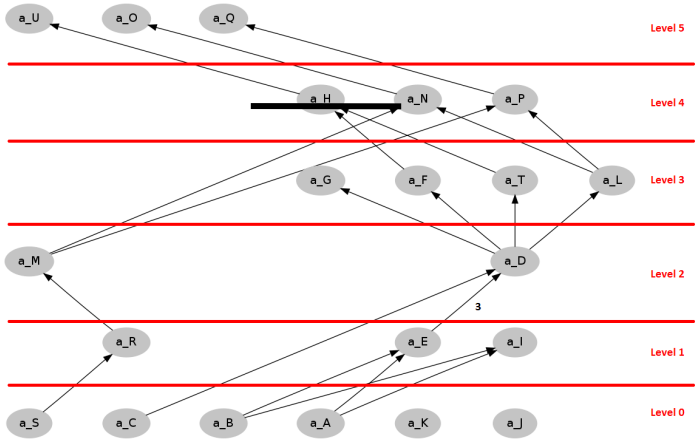
Multi-staged, Pipelined Processing

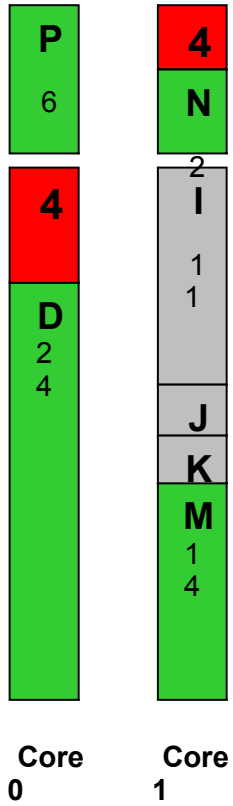
A Lot of Pipes!



Automated Sub-event Pipelining through Dataflow Analysis







Challenging optimization:

- Minimize memory traffic, load-balance cores
- Prototype results promising (2.5x parallel speedup)

Summary

Go parallel to save memory resources

Basic event task farm will be used in production this summer

- Famous last words...

Sub-event parallelization should improve memory locality and disk access patterns

Extra

Communication among sub-tasks