Development of a Next Generation Concurrent Framework for the ATLAS Experiment

Paolo Calafiura, Walter Lampl, Charles Leggett,
David Malon, Graeme Stewart, Ben Wynne

for the ATLAS Collaboration

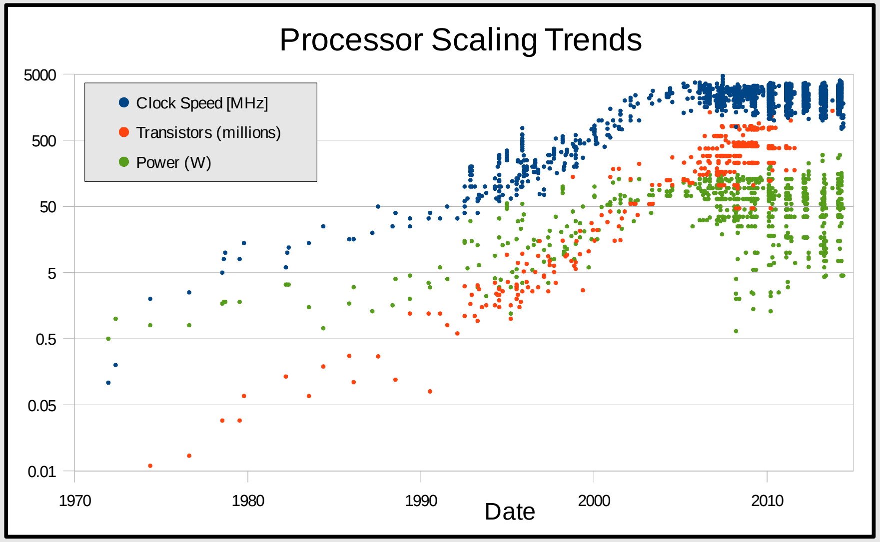
CHEP 2015

Motivation

- Current memory requirements for typical Reconstruction job:
  - **>3 GB** physical memory per job (continues to creep up)

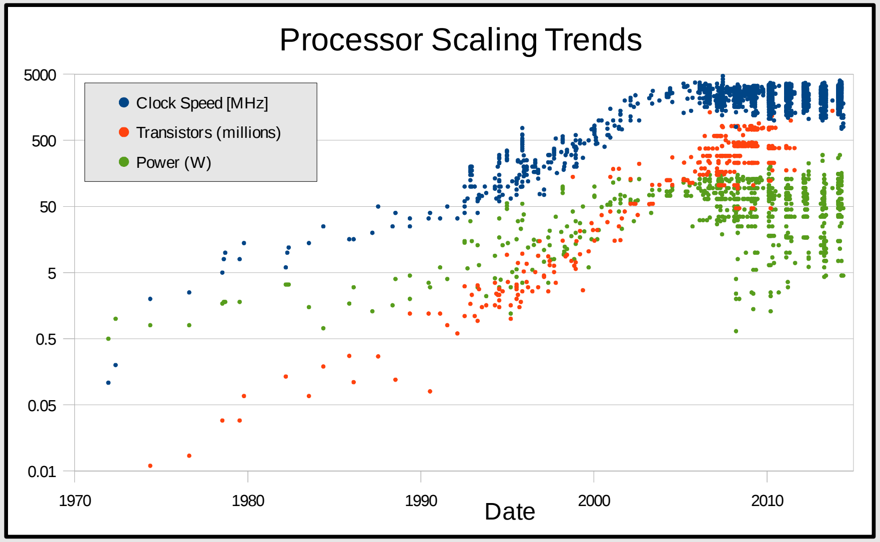
Motivation

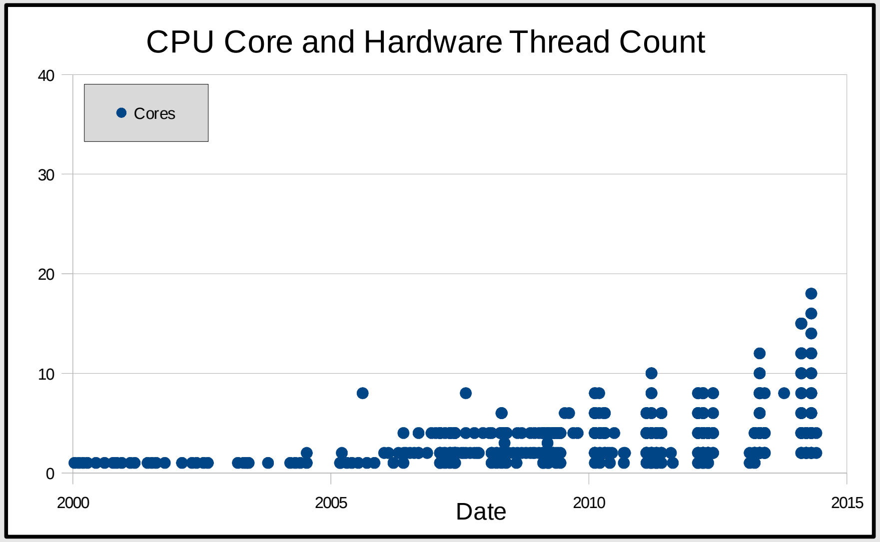
- Current memory requirements for typical Reconstruction job:
  - **>3 GB** physical memory per job (continues to creep up)



Motivation

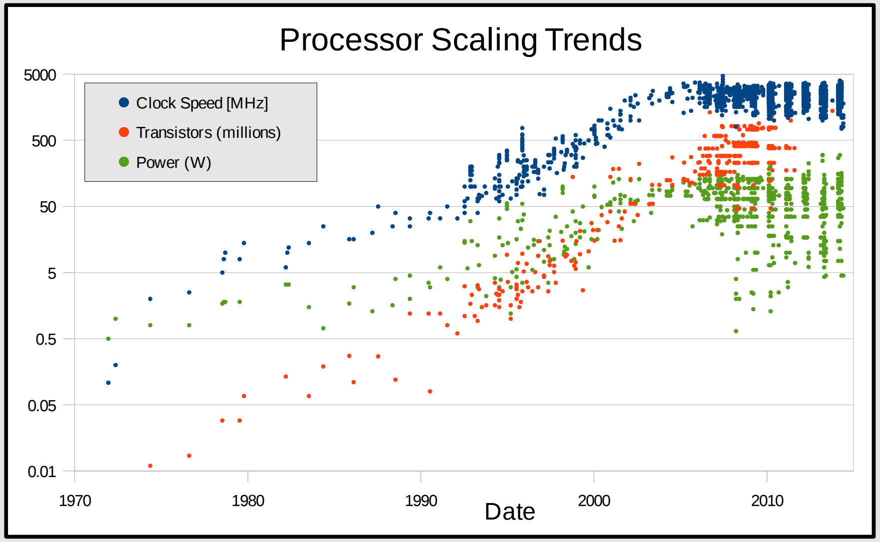
- Current memory requirements for typical Reconstruction job:
  - **>3 GB** physical memory per job (continues to creep up)

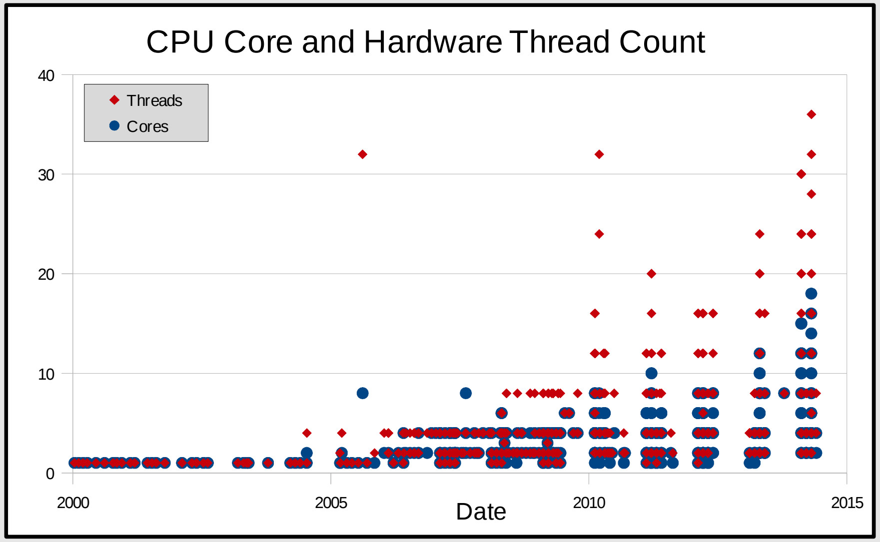




Motivation

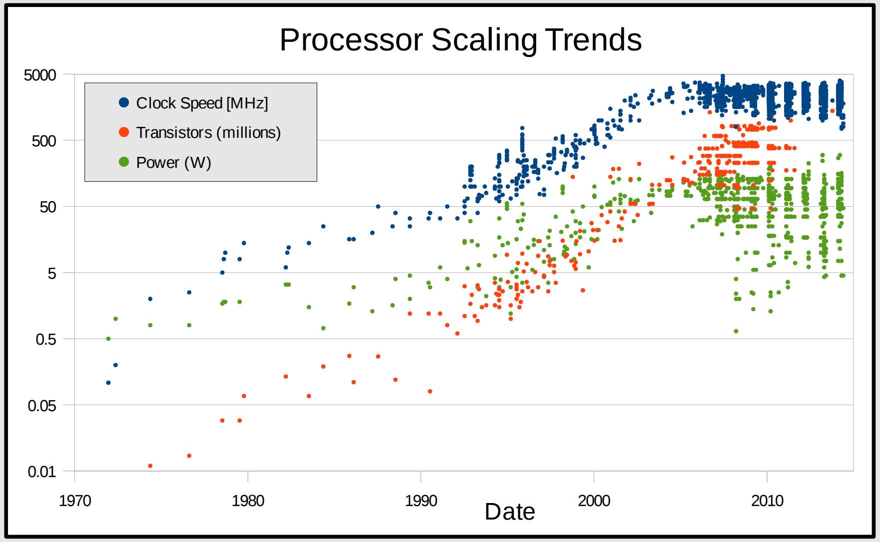
- Current memory requirements for typical Reconstruction job:
  - **>3 GB** physical memory per job (continues to creep up)

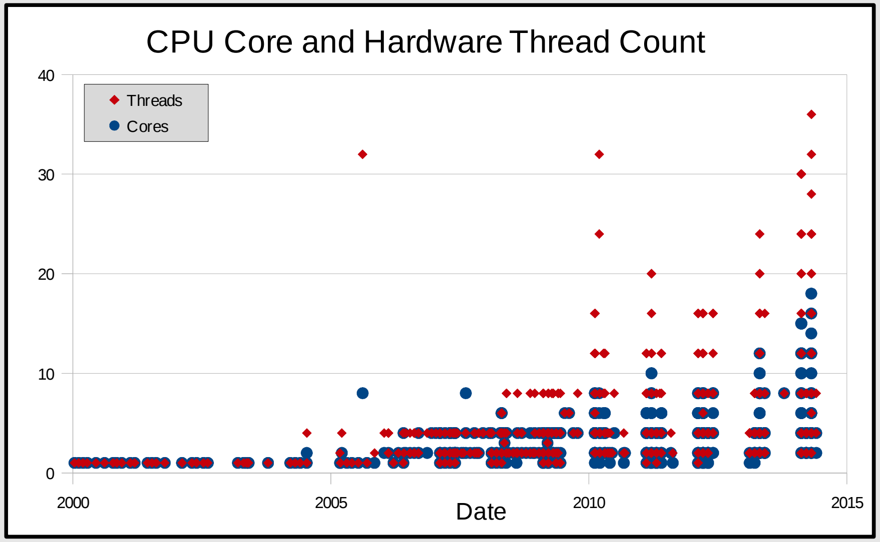


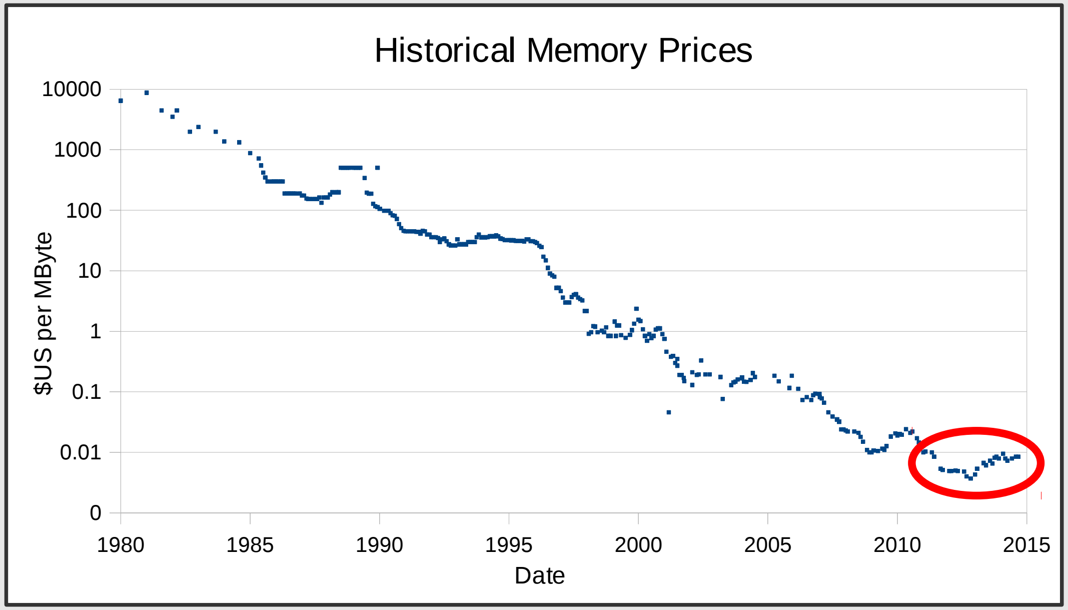


Motivation

- Current memory requirements for typical Reconstruction job:
  - **>3 GB** physical memory per job (continues to creep up)

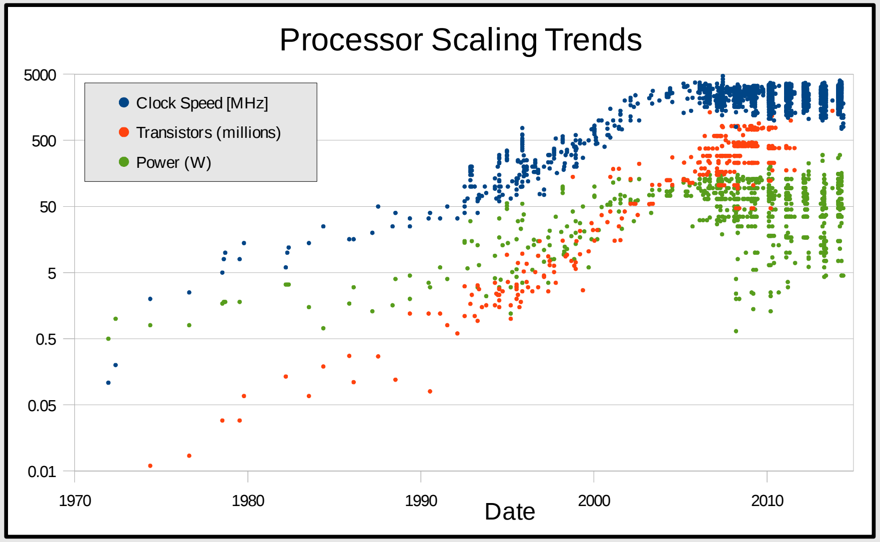


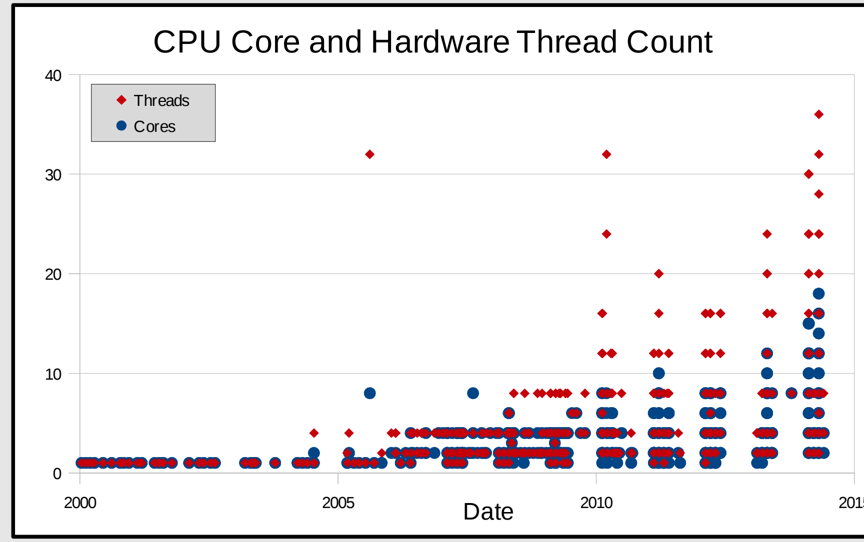


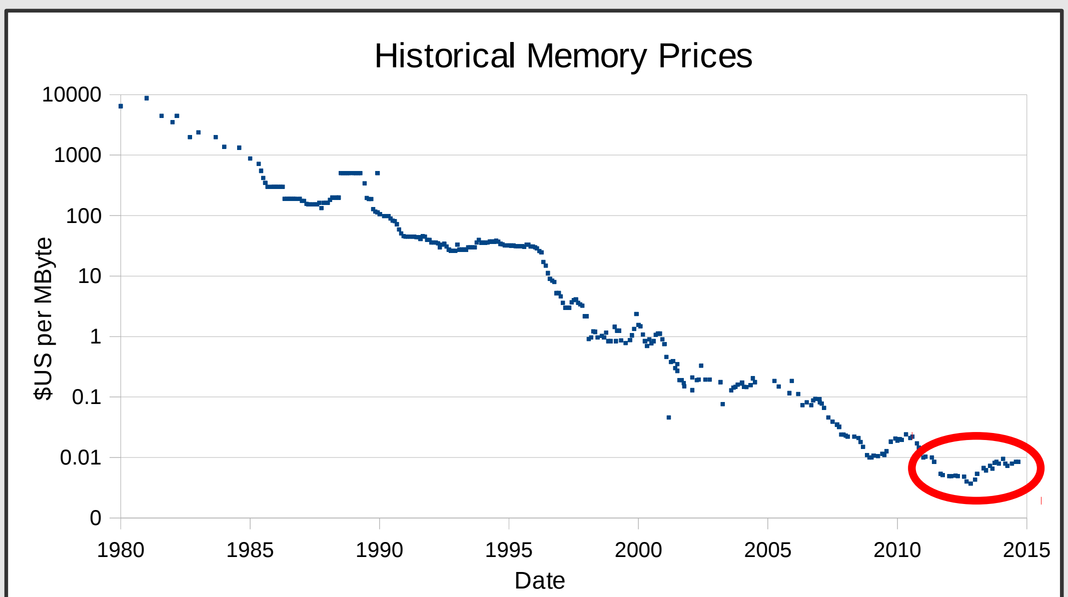


Motivation

- Current memory requirements for typical Reconstruction job:
  - **>3 GB** physical memory per job (continues to creep up)



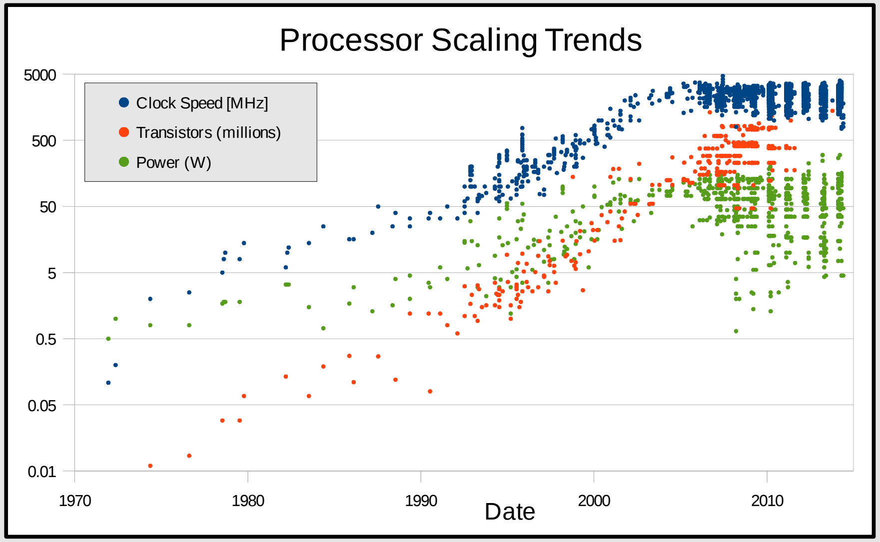


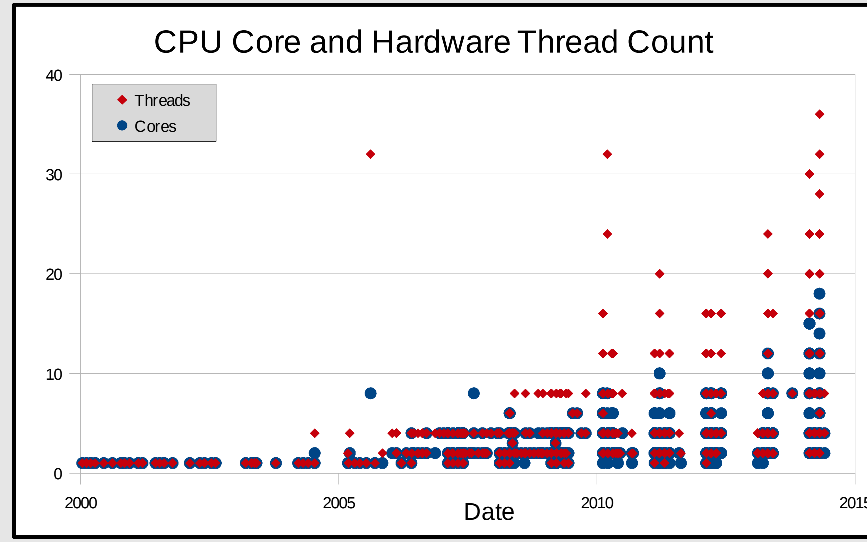


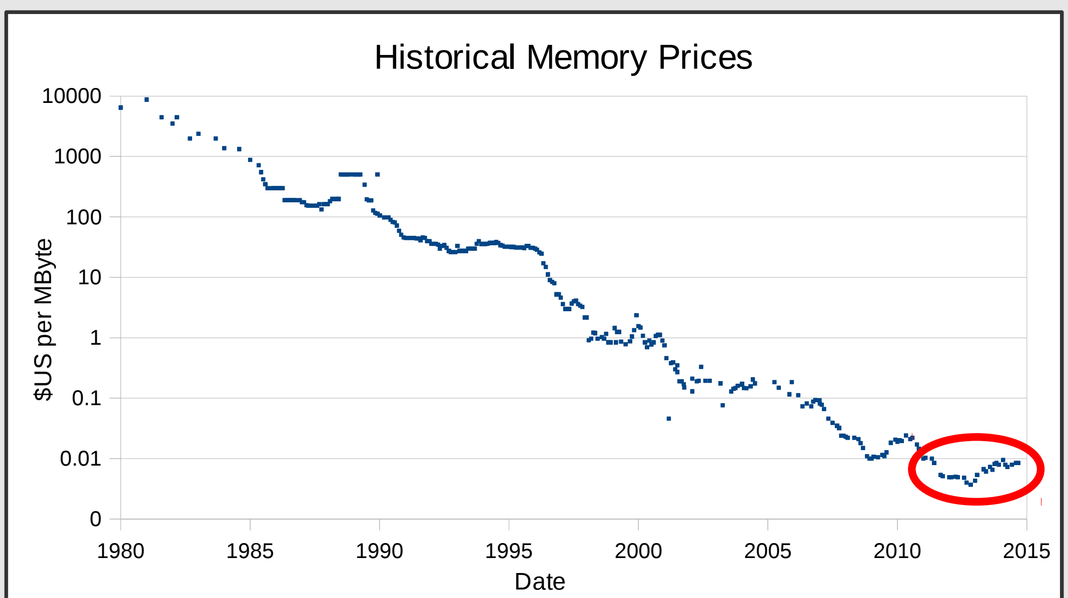
- 4 (cpu/node) x 18 (core/cpu) x 2 (threads/core) x 3.5 (GB/process) x 8 ($/GB) = **$4032**
  - **$5.6 Million** for 150k grid cores
  - future processors will have many more than 18 core/cpu

Motivation

- Current memory requirements for typical Reconstruction job:
  - ▸ **>3 GB** physical memory per job (continues to creep up)



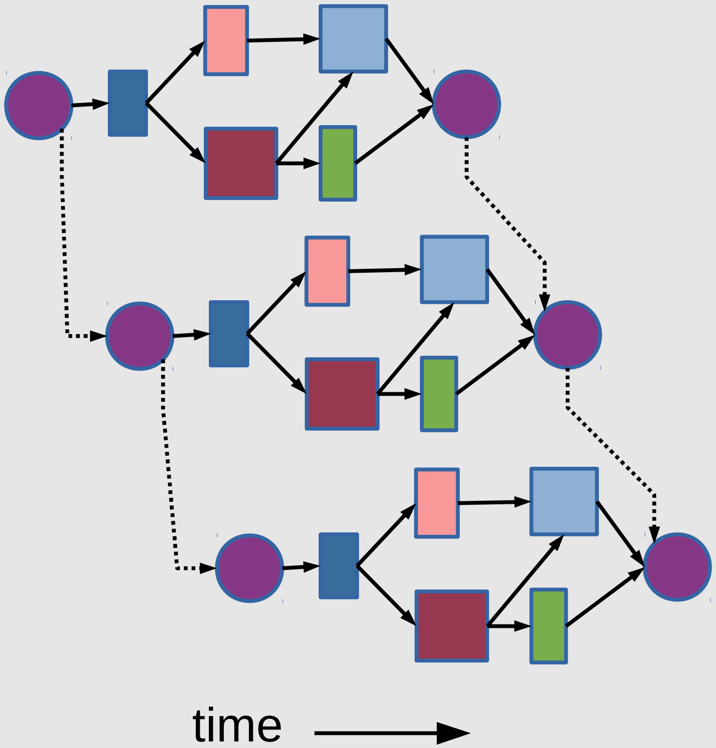




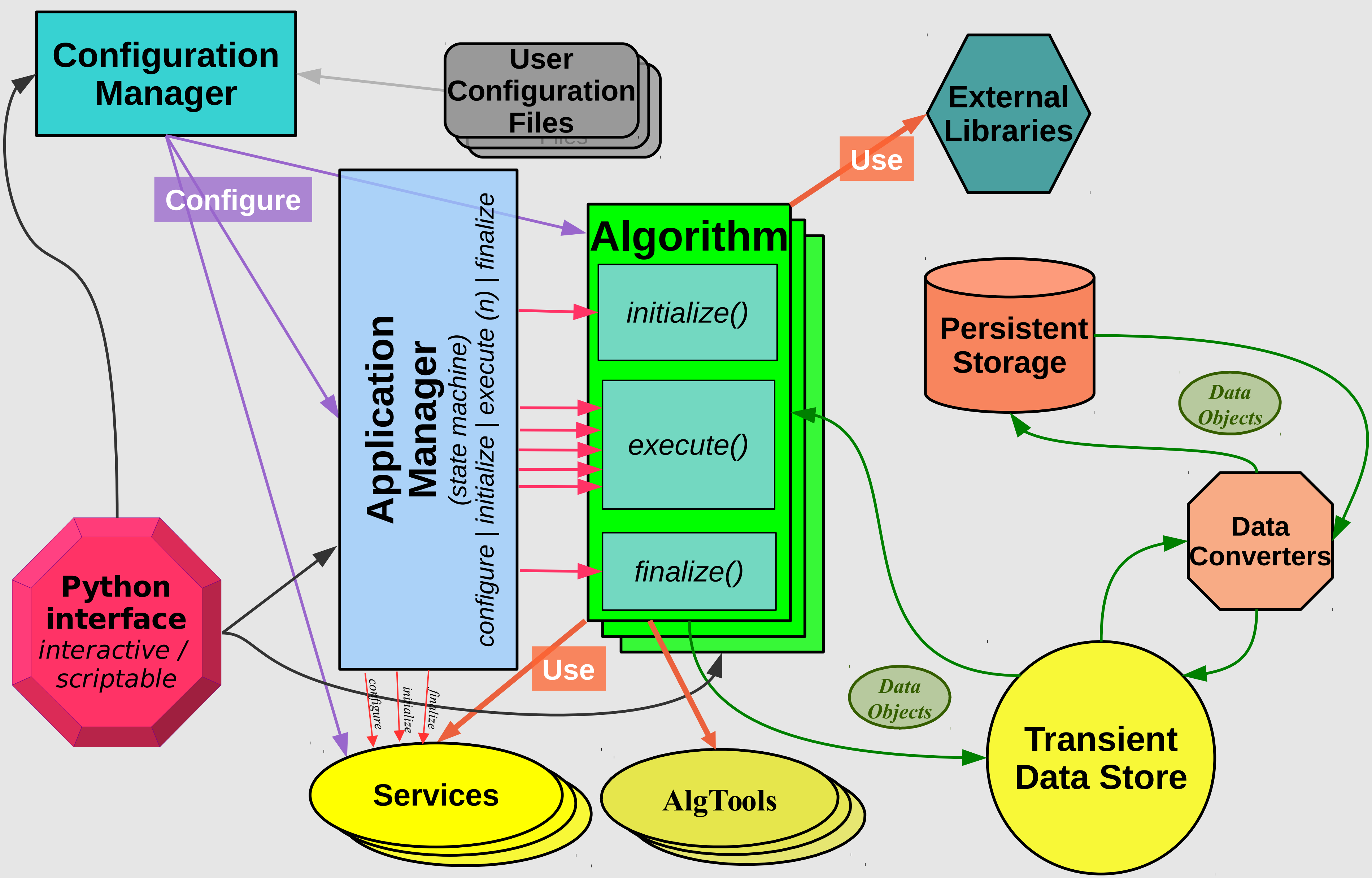
- 4 (cpu/node) x 18 (core/cpu) x 2 (threads/core) x 3.5 (GB/process) x 8 ($/GB) = **$4032**
  - ▸ **$5.6 Million** for 150k grid cores
  - ▸ future processors will have many more than 18 core/cpu
- Need to minimize memory usage while making full use of hardware
- AthenaMP's memory savings *via* COW won't save us in the long run

Gaudi Hive

- Gaudi Hive: multi-threaded, concurrent extension to Gaudi
  - since Athena is based on Gaudi, obvious first step in our evaluation of multi-threaded frameworks
- Data Flow driven
  - Algorithms declare their data dependencies
  - Scheduler automatically executes Algorithms as data becomes available.
- Multi-threaded
  - Algorithms process events in their own thread, from a shared thread pool.
- Pipelining: multiple algorithms and events can be executed simultaneously
  - some Algorithms are long, and produce data that many others need (eg track fitting). instead of waiting for it to finish, and idling processor, start a new event.
- Algorithm Cloning
  - multiple instances of the same Algorithm can exist, and be executed concurrently, each with different Event Context.
  - cloning is not obligatory, balancing memory usage with thread safety.

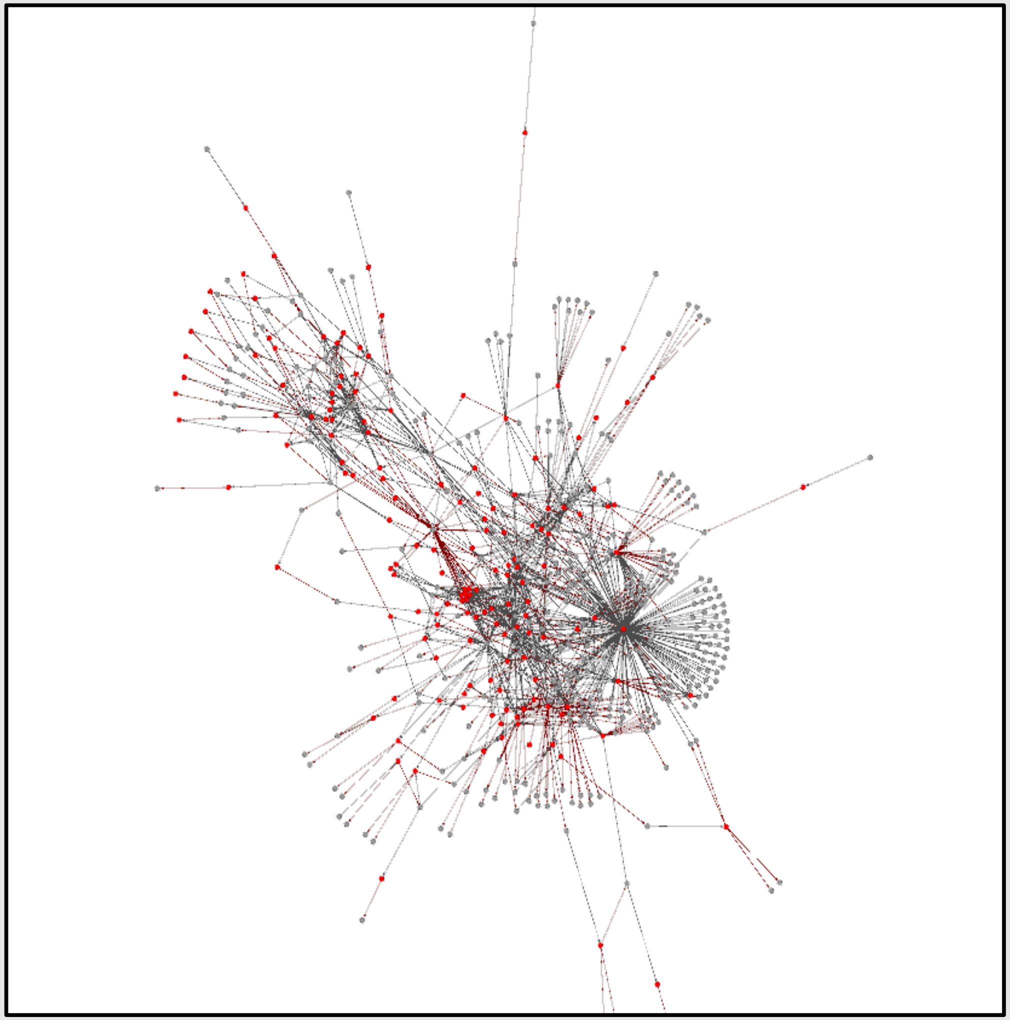


Gaudi Component Model



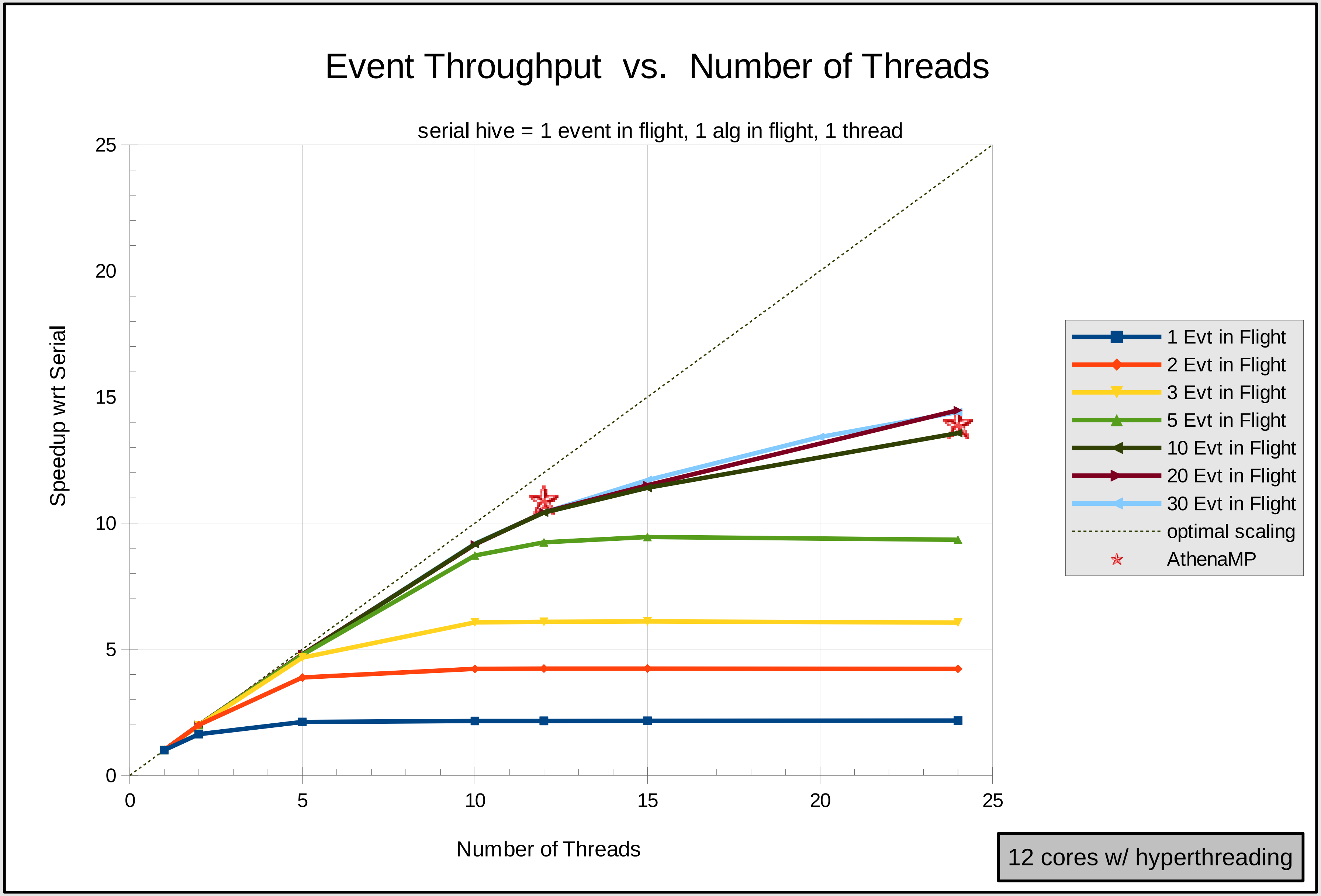
Stage I: Simulated Reconstruction

- Extract Algorithm/Data dependency graph, and timing data from running normal Atlas Reconstruction on *ttbar* event.
  - 161 Algs, 317 Data Objects



- Implemented a CPU Cruncher Algorithm to mimic CPU usage of each Reconstruction Algorithm using real timing data.
- Run through GaudiHive.
- Configuration parameters:
  - # of concurrent events
  - # Algs in flight (limit number of simultaneously executing Algorithms)
  - size of thread pool
  - cloning (multiple instances of each Algorithm)

Simulated Reconstruction Timing Results



- Disabling cloning on all but the 7 slowest Algorithms decreases maximum throughput by only 1.2%

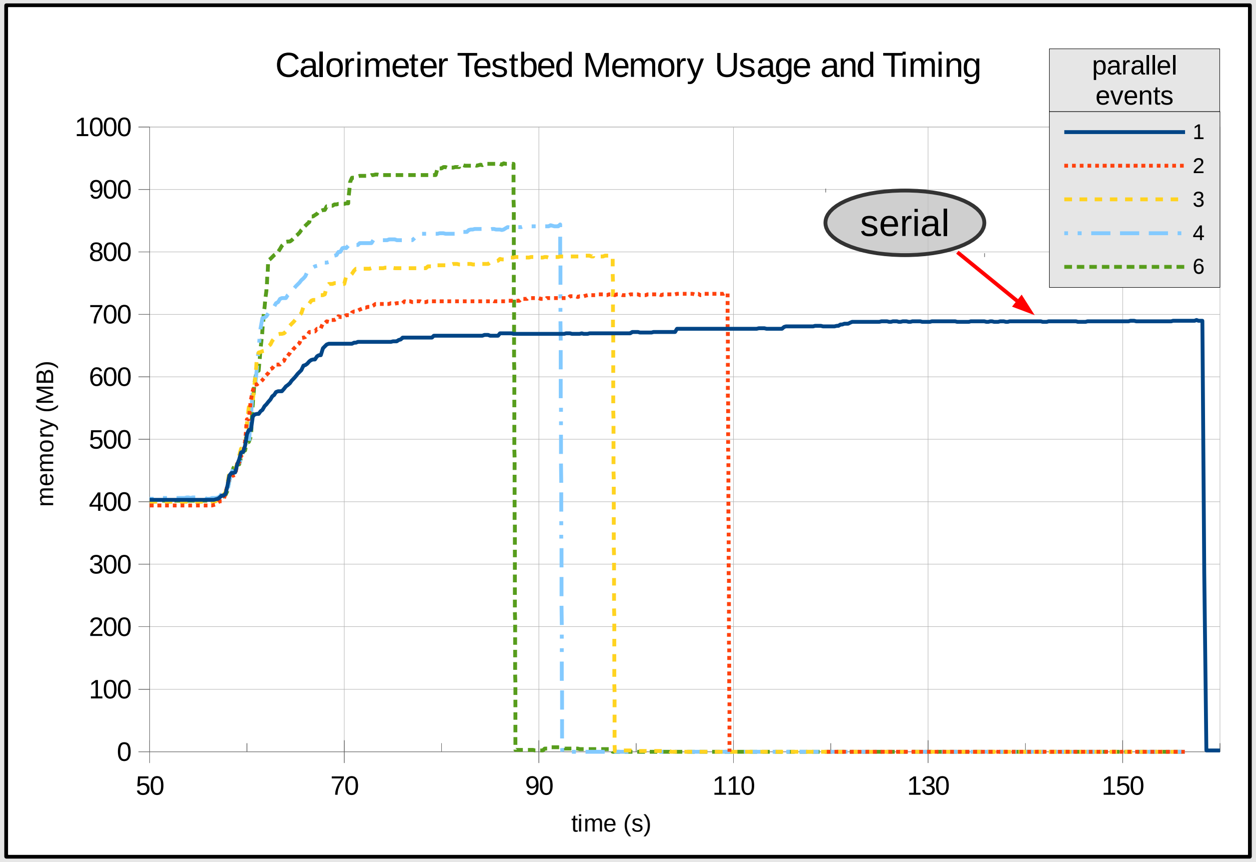
Stage II: Real Data Testbeds

- Run real data through real reconstruction code

- Select subsets of detector
  - Calorimeter (5 Algorithms, 16 Data Objects)
  - Inner Detector: SCT and Pixels (7 Algorithms, 19 DataObjects)

- Explore what needs to be modified in user code and framework to make it functional
  - framework extensions and incompatibilities
  - thread safety
    - fixing vs. locking
  - ATLAS general code design patterns
    - data access
    - inter-event communication back channels
    - shared and private Tool usage by Algorithms

Calorimeter Testbed Timing Results



parallel events	speedup wrt serial	speedup wrt n*Serial	memory ratio to serial	memory ratio to n*Serial
1	1.00	1.00	1.00	1.00
2	2.07	1.04	1.06	0.53
3	2.80	0.93	1.15	0.38
4	3.33	0.83	1.22	0.31
6	4.01	0.67	1.36	0.23

- Concurrency limited by small number of Algorithms in configuration, some of which could not be run concurrently for thread safety issues
- Best performance is with 6 concurrent events
  - **401%** event throughput, (ignoring startup to 1st event), and **36%** increase in memory consumption *vs* one serial job
  - **67**% event throughput, and **23%** of memory utilization of 6 serial jobs running concurrently

Stage II: Framework Experiments

- In order to minimize intervention in existing ATLAS software, can thread safety issues in user code be limited by modifying the framework?
  - multiple instances of Services and AlgTools (one per concurrent event)
  - distributing event specific asynchronous Incidents to "correct" clients
  - serializing / locking unsafe code
  - limit cloning of Algorithms

- YES, but at the expense of:
  - increased memory usage
  - lower performance due to reduced concurrency

Stage II: Testbed Conclusions

- Obvious benefits with speed/memory usage

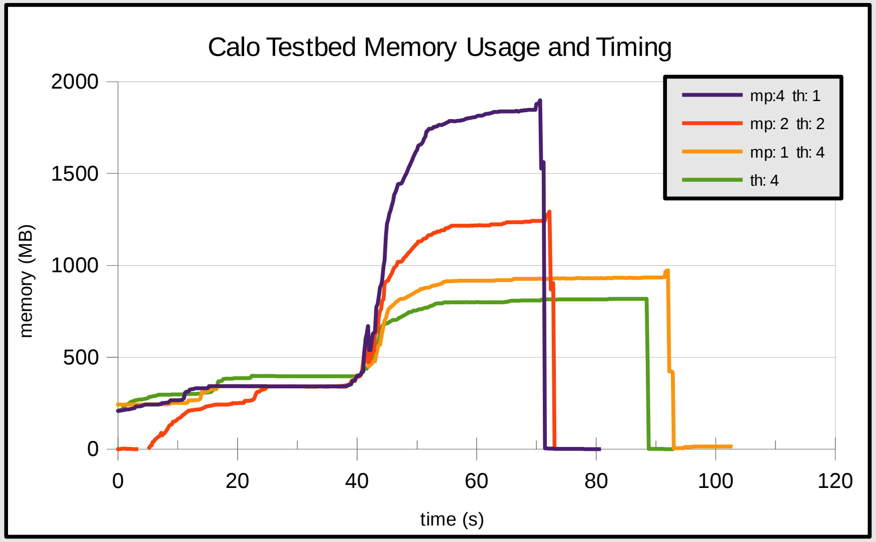
- Thread Safety Issues:
  - many shared Tools and Services are not thread safe or cache informations between events
  - global static variables
  - I/O must be serialized

- Software Pattern Issues:
  - memory pools
  - modification of data after registration in store
  - automatic data loading via proxies
  - Algorithms marshal many Tools, limiting concurrency

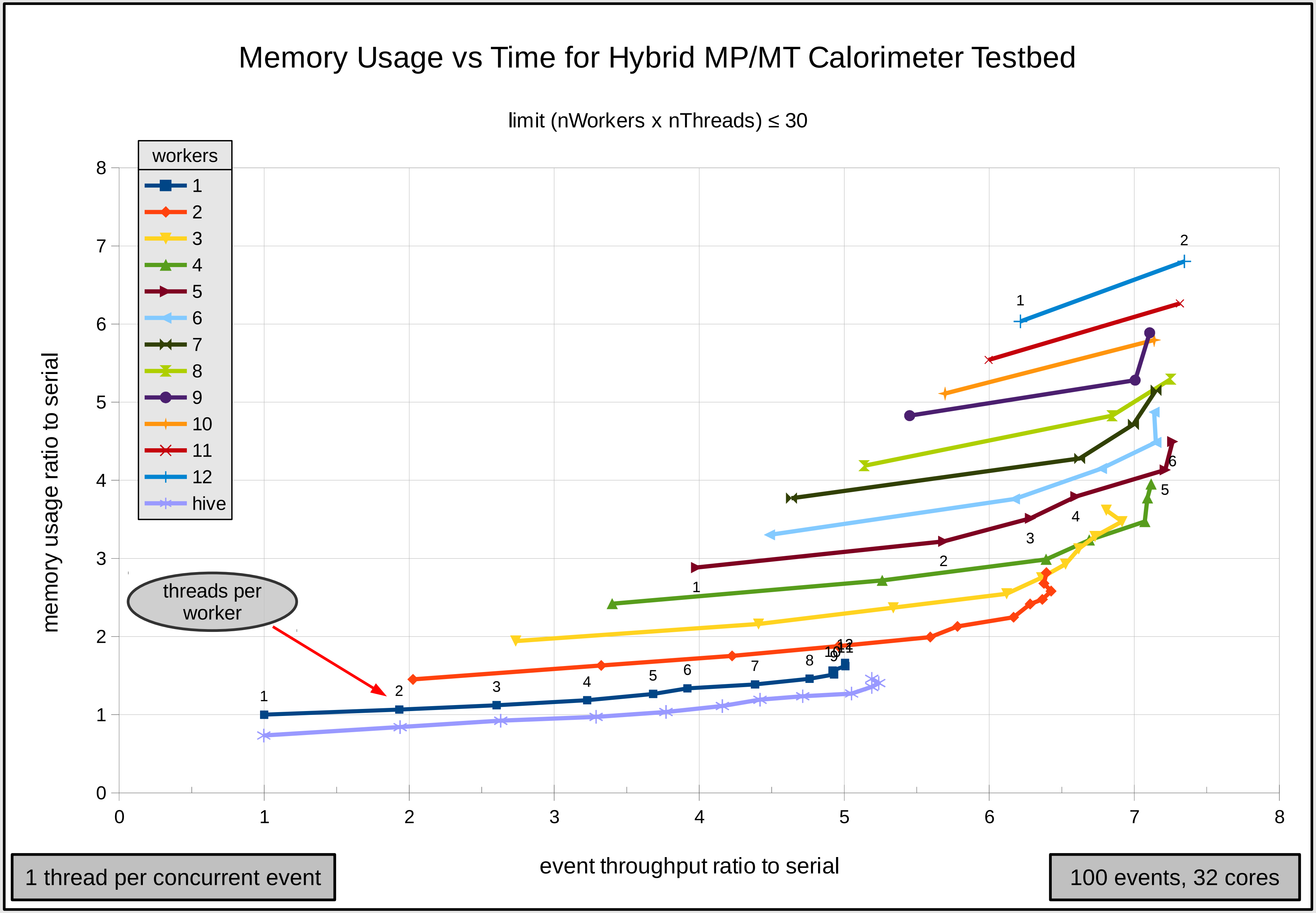
Stage III: Hybrid MP/MT

- Hybrid multiprocess / multi-threading framework implementation
  - events are distributed to worker processes *via* AthenaMP shared event queue
  - multi-threading within each worker
    - only one concurrent event, but multiple parallel algorithms in different threads
      - reduce thread safety issues
      - initial modeling on toy simulation shows ~8 parallel data paths for full reconstruction
    - full multi-threading, with multiple concurrent events and algorithms
- Maximizes processor utilization while reducing memory footprint compared with pure event level concurrency
  - *eg* 4 concurrent events:

	time /s	memory /MB
mp: 4 proc, 1 conc. event	32.8	1847
mp: 2 proc, 2 conc. events	34.4	1241
mp: 1 proc, 4 conc. events	53.6	935
hive: 4 concurrent events	47.8	817



Performance of MP/MT Hybrid



Stage IV: Geant4 Simulation

- Geant4 simulation is almost 1/2 of total ATLAS CPU budget
  - ideally suited for HPC
- Run simulation with multiple concurrent threaded events to do full ATLAS simulation
  - leverage thread safety in Geant4 v10
- Geant4 v10 can do event level parallelism *via* multi-threading
  - take control of event loop from G4, do it within Athena
- Issues: Geant4 v10 has very different method of user class initialization, separating thread local and global
  - will require a substantial rewrite of ATLAS G4 code
- Goal is to test production simulation apps by end of year

Summary

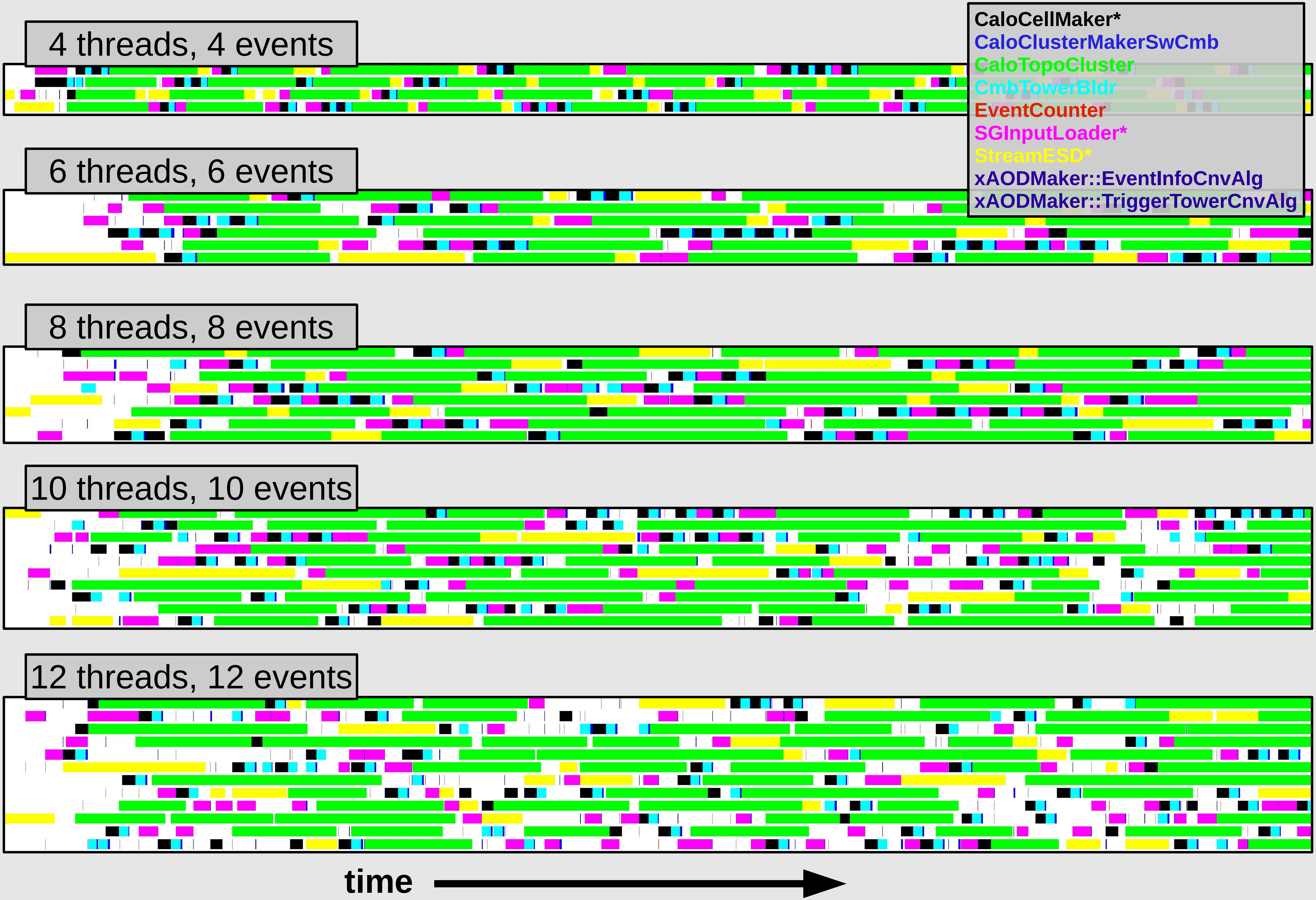
- Multi-threaded framework is not a drop in replacement
  - can't avoid thread safety issues by modifying framework instead of ATLAS code
    - reduced performance benefits
    - diminished memory savings
- Much user code can survive unscathed, after changing a few user patterns
  - no inter-event caching of data
  - thread safety desired but NOT required
- Core services will require full thread safety, and no inter-event data caching
  - non-trivial, requiring significant intervention
- Vast majority of these changes are backward compatible, and will likely make current serial code more efficient
- Envision **evolutionary** rather than **revolutionary** code changes

Extra

warning:

may cause visual discomfort

Alg Execution Timelines for Calo Testbed



Performance of MP/MT Hybrid