SPARK-ROOT: First Looks at Performance with Spark.

Viktor Khristenko (Iowa), Jim Pivarski (Princeton University — DIANA), Luca Canali (CERN)

Outline

- Introduction
  - SPARK-ROOT
  - Intel’s Cluster
  - Data
- SPARK Execution
- SPARK Monitoring
- Procedure
- Queries Performed
- Results

SPARK-ROOT

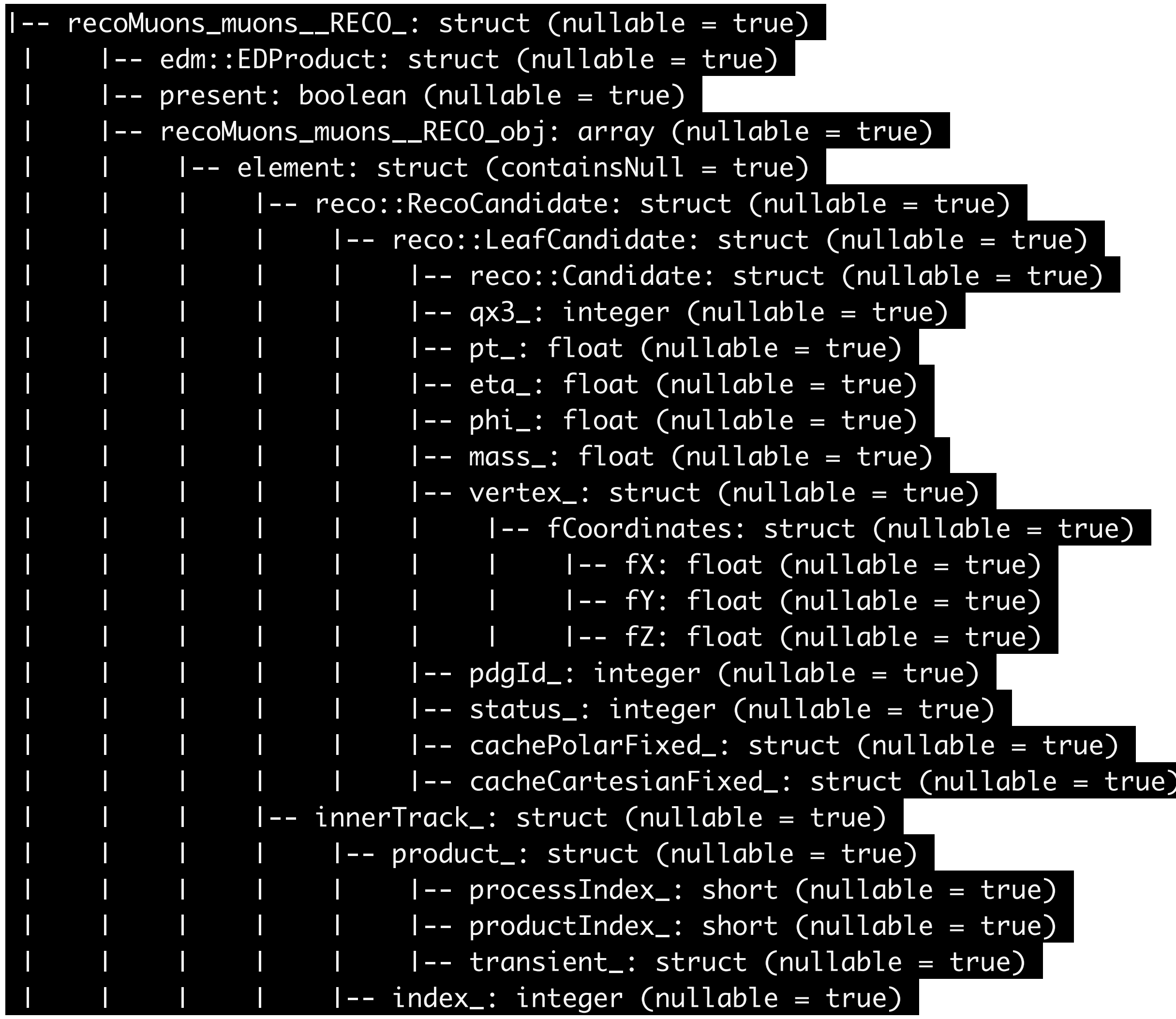
- ROOT **I**/O for JVM
- Usage with SPARK is just an example!
- on Maven - 0.1.9 latest keep incrementing!
  - http://search.maven.org/#search%7Cga%7C1%7Ca%3A%22spark-root_2.11%22
- https://github.com/diana-hep/spark-root
- https://github.com/vkhristenko/spark-root-applications
  - Monitoring/Definitions/Examples

Intel’s Cluster

- CERN IT-DB received a grant
- 14 machines
- 2x18 cores => 72 (2x36) threads max used (Spark’s —num-cores is actually threads!)
- spark-root got its first benchmarking/testing outside of CERN!

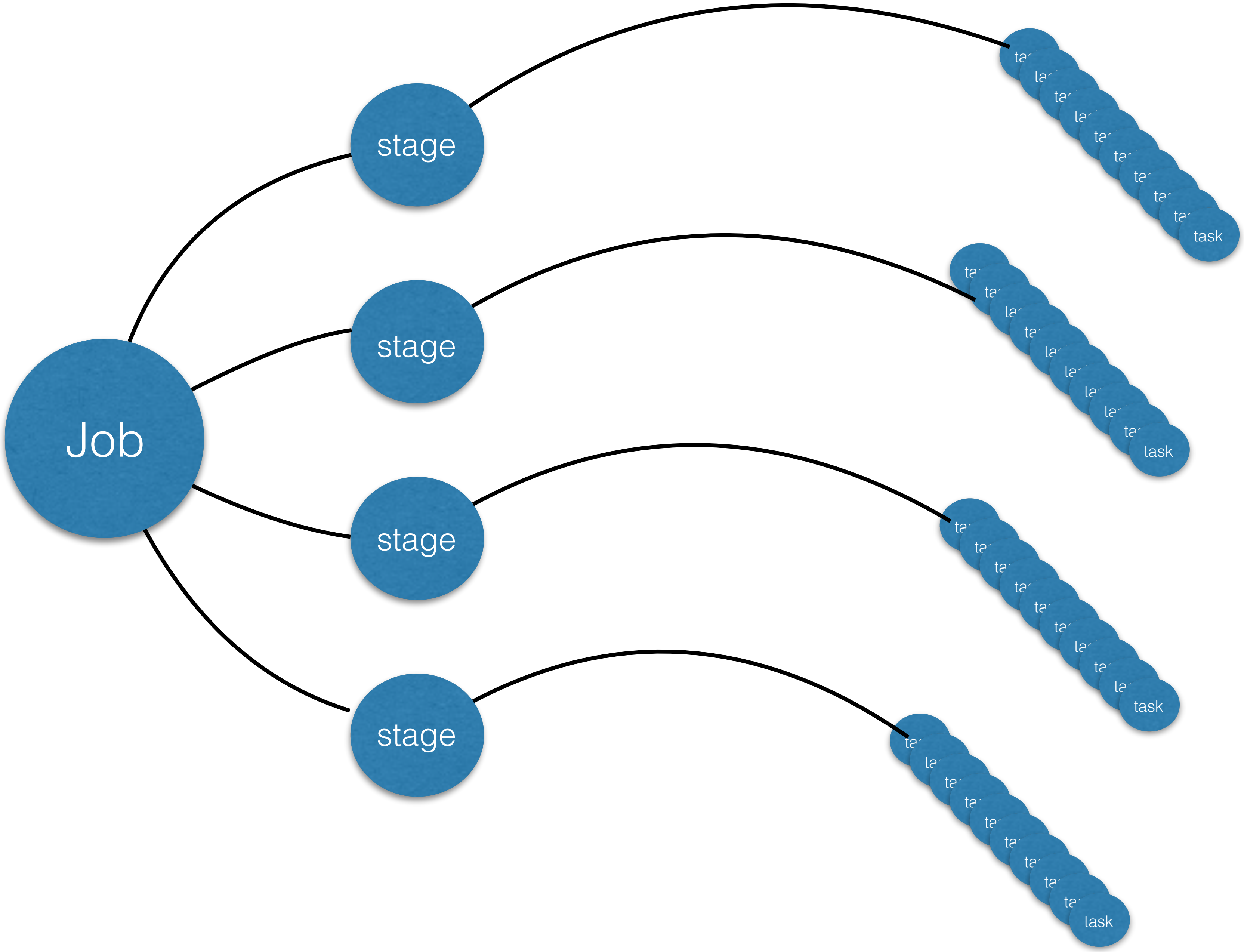
What Data?

- Muonia CMSSW AOD 2010
    - /MuOnia/Run2010B-Apr21ReReco-v1/AOD
    - http://opendata.cern.ch/record/10
- Total ~ 1.2TB
- Total files > 1000 (~1GB per file)



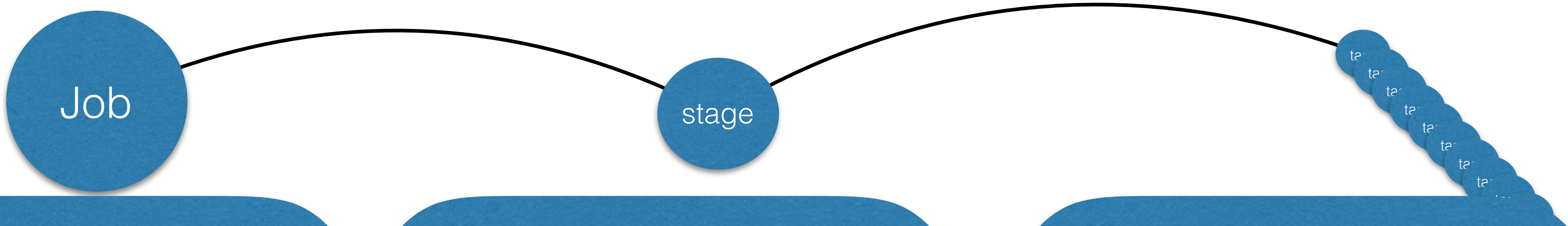
```
|-- recoMuons_muons__RECO_: struct (nullable = true)
 |    |-- edm::EDProduct: struct (nullable = true)
 |    |-- present: boolean (nullable = true)
 |    |-- recoMuons_muons__RECO_obj: array (nullable = true)
 |    |    |-- element: struct (containsNull = true)
 |    |    |    |-- reco::RecoCandidate: struct (nullable = true)
 |    |    |    |    |-- reco::LeafCandidate: struct (nullable = true)
 |    |    |    |    |    |-- reco::Candidate: struct (nullable = true)
 |    |    |    |    |    |-- qx3_: integer (nullable = true)
 |    |    |    |    |    |-- pt_: float (nullable = true)
 |    |    |    |    |    |-- eta_: float (nullable = true)
 |    |    |    |    |    |-- phi_: float (nullable = true)
 |    |    |    |    |    |-- mass_: float (nullable = true)
 |    |    |    |    |    |-- vertex_: struct (nullable = true)
 |    |    |    |    |    |    |-- fCoordinates: struct (nullable = true)
 |    |    |    |    |    |    |    |-- fX: float (nullable = true)
 |    |    |    |    |    |    |    |-- fY: float (nullable = true)
 |    |    |    |    |    |    |    |-- fZ: float (nullable = true)
 |    |    |    |    |    |-- pdgId_: integer (nullable = true)
 |    |    |    |    |    |-- status_: integer (nullable = true)
 |    |    |    |    |    |-- cachePolarFixed_: struct (nullable = true)
 |    |    |    |    |    |-- cacheCartesianFixed_: struct (nullable = true)
 |    |    |    |-- innerTrack_: struct (nullable = true)
 |    |    |    |    |-- product_: struct (nullable = true)
 |    |    |    |    |    |-- processIndex_: short (nullable = true)
 |    |    |    |    |    |-- productIndex_: short (nullable = true)
 |    |    |    |    |    |-- transient_: struct (nullable = true)
 |    |    |    |    |-- index_: integer (nullable = true)
```

SPARK Execution Model



1Query = 1 Job = N stages = stages.flatMap(_.tasks).length Tasks

SPARK Monitoring



- onJobStart/onJobEnd transitions
- job id
- job name/group
- startTime/endTime - same as timing the job!
- list of Stages

- onStageSubmitted/ onStageCompleted
- id/name
- submissionTime/ completionTime
- list of Tasks

- onTaskStart/onTaskEnd/ onTaskGettingResult
- id/host/executorId
- duration
- launchTime/finishTime/ gettingResultTime
- Metrics:
  - Exec DeserTime
  - Exec Deser CPU Time
  - Exec Run Time
  - Exec CPU Time
  - JVM GC Time
- bytes Read/Written
- ….

SPARK Monitoring Summary

- Job/Stage/Task Transitions are currently collected
- There are more transitions available!
- There is other monitoring info available (I/O like but limited). spark-root needs work on I/O functionality - with spark.sqlContext.read.root… can not __now__ see the I/O stats, but can with parquet…
- There is REST API -> JSON, however unreliable/depends on Cloudera Distribution used…. etc…
    - at least at this point…..

Procedure

- Use full 1.2TB of data
- Selected 5 type of queries: from df.count up to several lines long ones.
- launch spark with N executors M threads
- perform these 5 queries. each one is redone 3 times.
  - I’m aware of hashing - have to understand better these details. When it’s performed/when not…
- spark.stop! stop spark context
- redo the above steps varying number of executors range(5, 15, 1) keep threads=70
  - should’ve done 36 as well….
- redo the above steps varying number of threads range(20, 75, 5)
- important - each time I change the configuration (N execs, M threads) start/stop spark’s contexts

```
d.filter(_.muons.length >= 2)
.flatMap({e: Event => for (i <- 0 until e.muons.length; j <- 0 until e.muons.length) yield buildDiCandidate(e.muons(i), e.muons(j))})
.rdd.aggregate(emptyDiCandidate)(new Increment, new Combine)
```

Examples of Queries

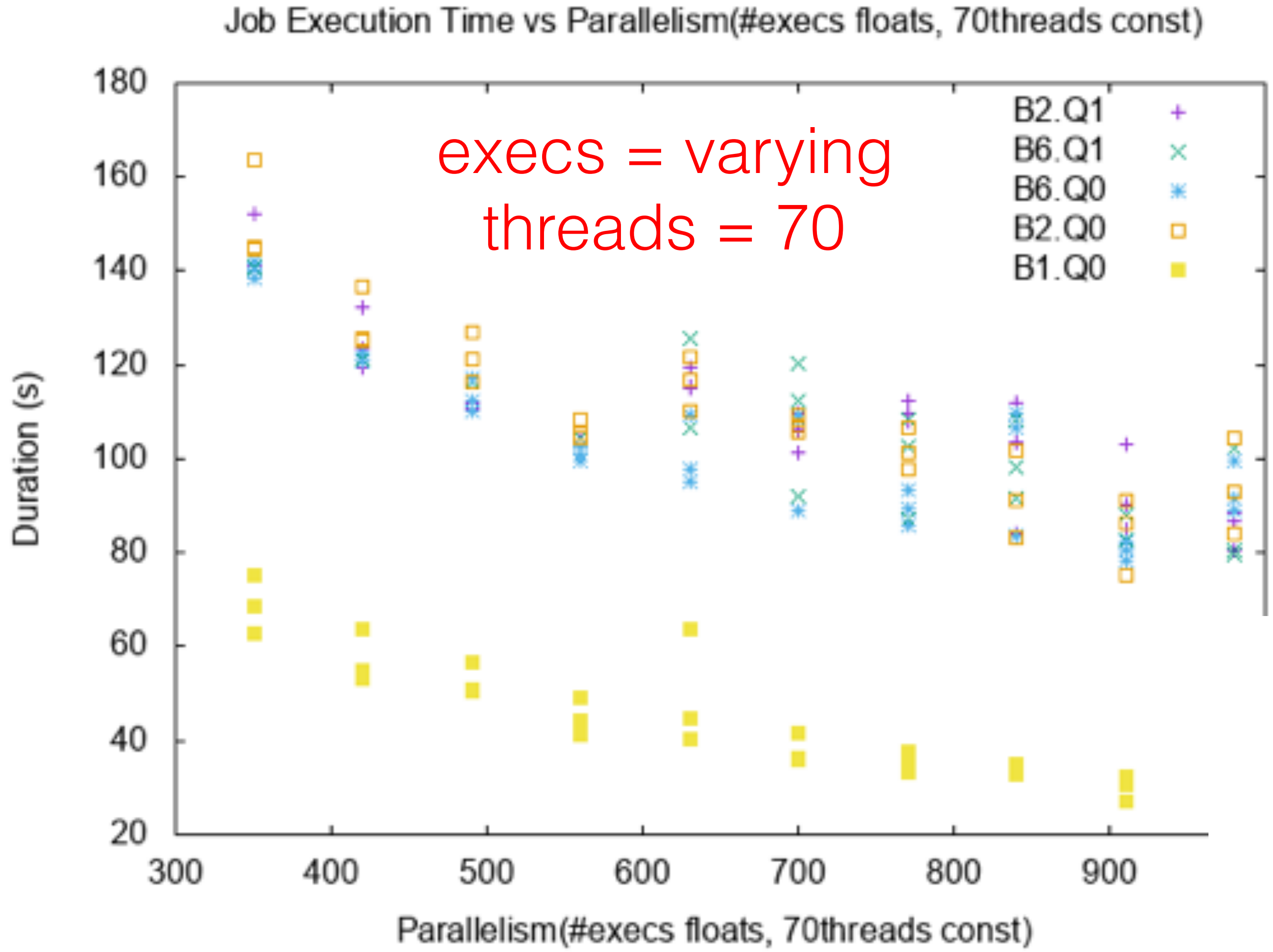
- Dataset[Row] - df.count - count #rows
- Dataset[Row] - select(column).flatMap(…).reduce(…)
- Dataset[Event] - ds.filter(_.muons.length >= 2)**.flatMap({e: Event => for (i <- 0 until e.muons.length; j <- 0 until e.muons.len gth) yield buildDiCandidate(e.muons(i), e.muons(j))}).rdd.aggregate(emptyDiCandidate)(new Increment, new Combine)**

histogrammar aggregation

dataset manipulations

Time per Job

36 phys cores!
36x14 = 504 total

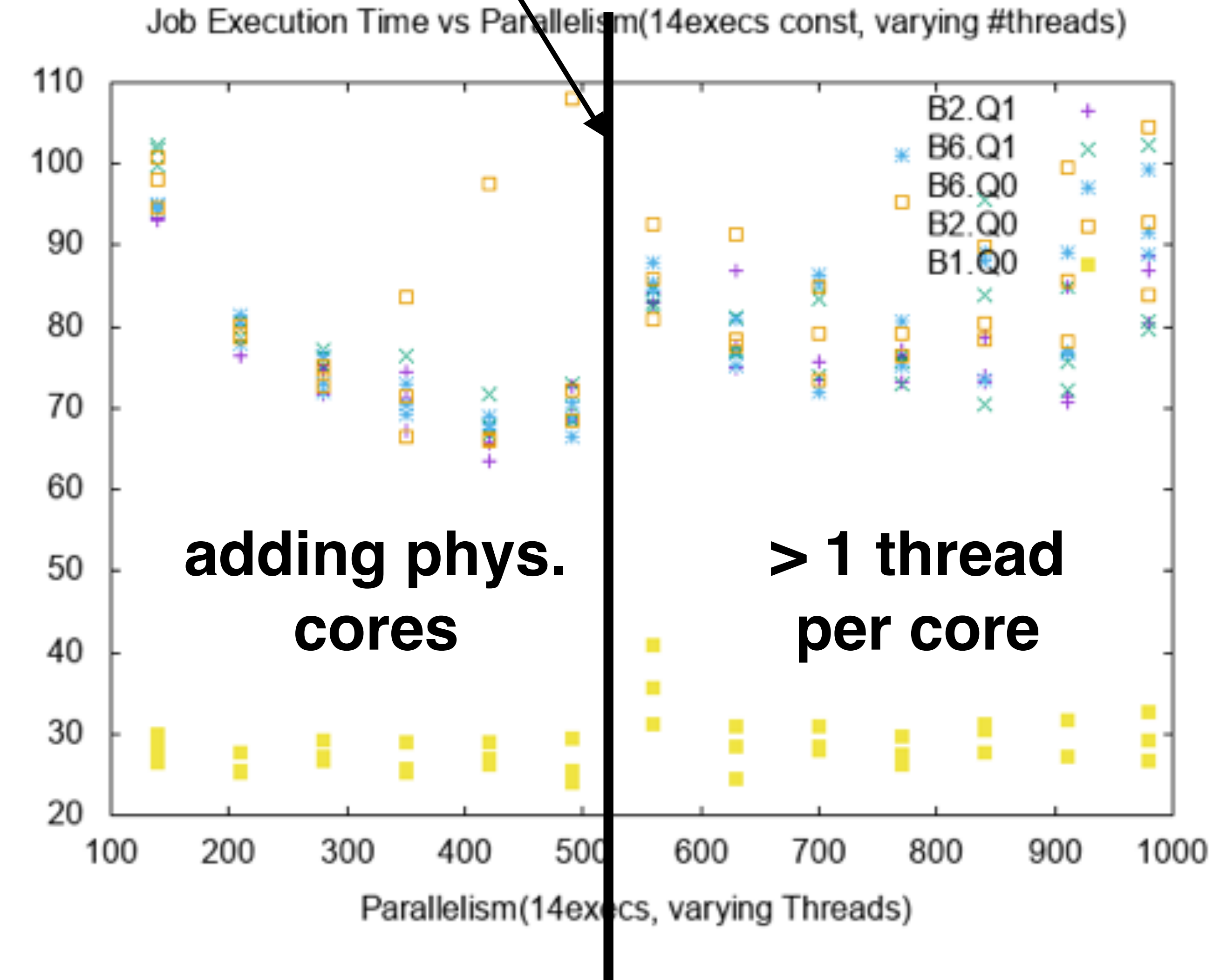


execs = varying
threads = 70

Here we adding more machines,
keeping #threads constant!

total 504
phys cores

execs = 14
threads = varying

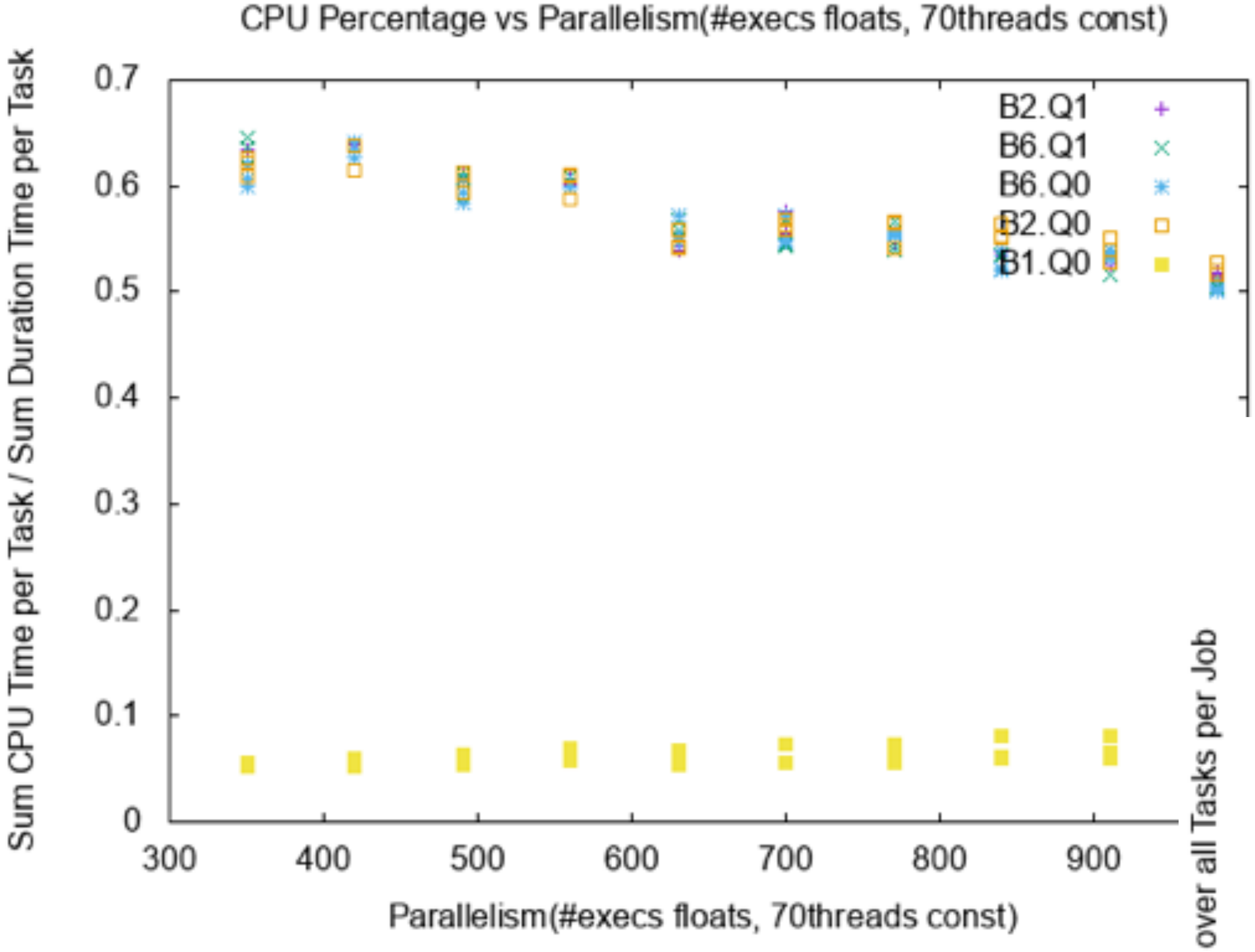


adding phys. cores

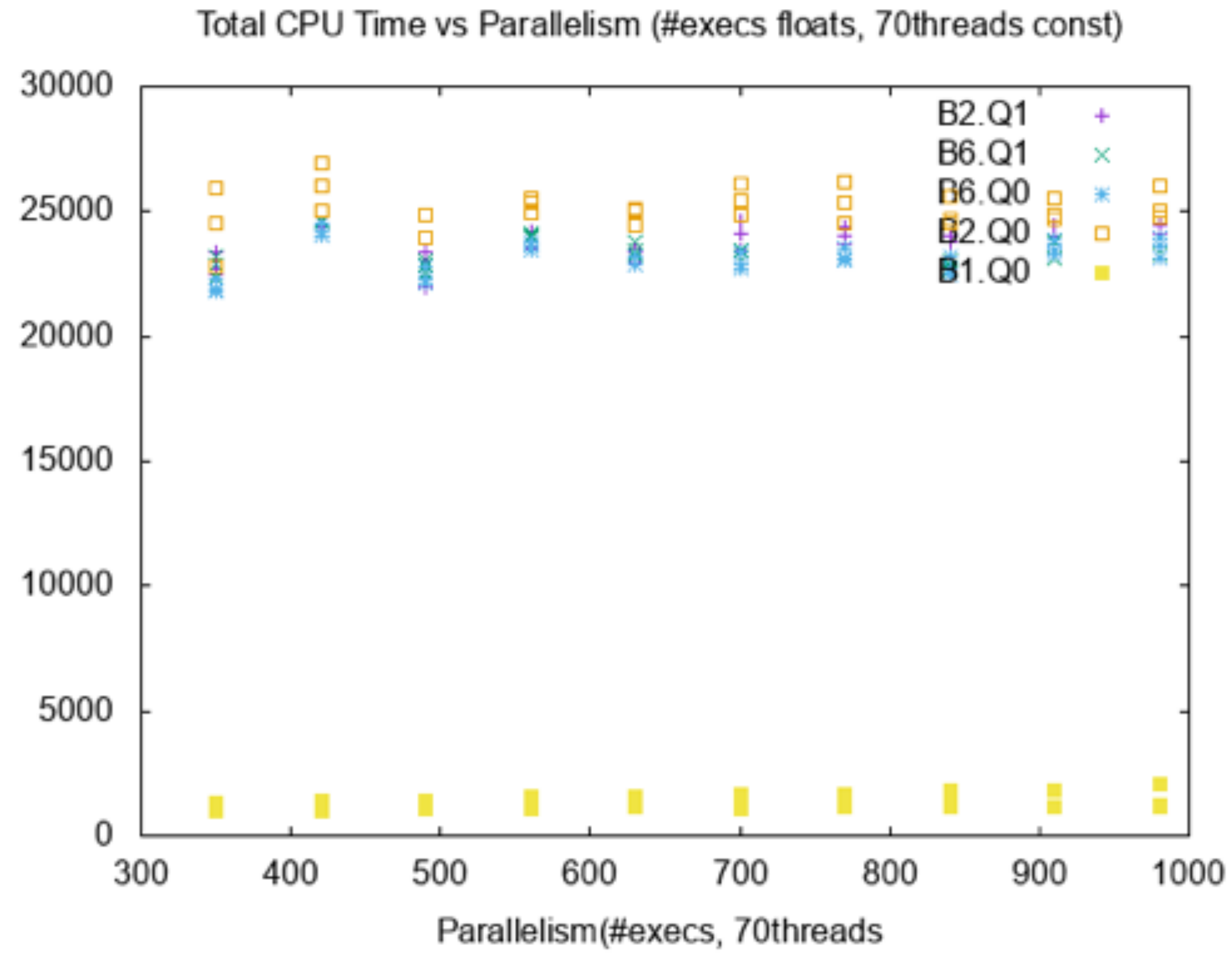
> 1 thread per core

CPU Usage

Ideally, CPU usage should be constant (per query) upon increasing the parallelism.

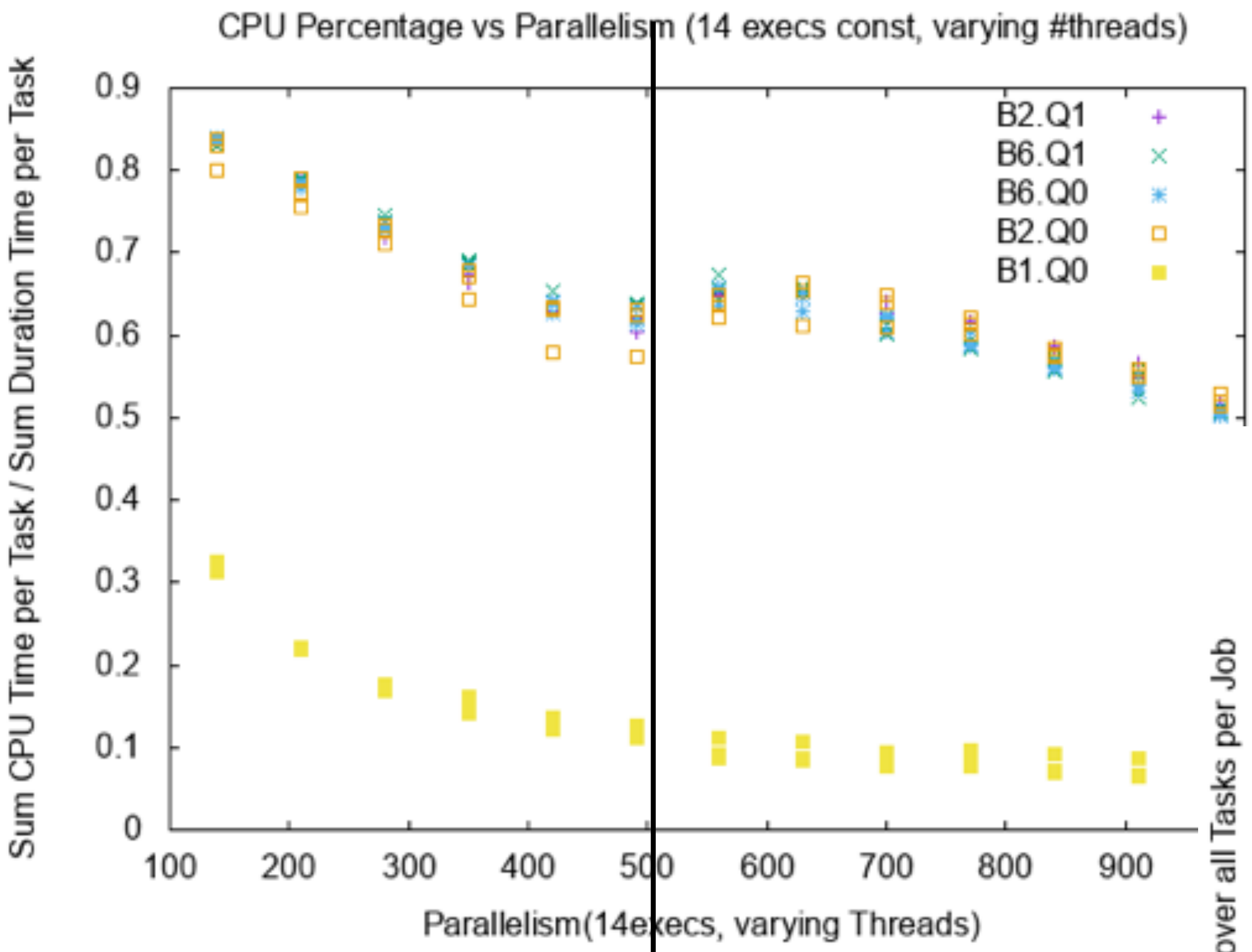


70 threads varying execs



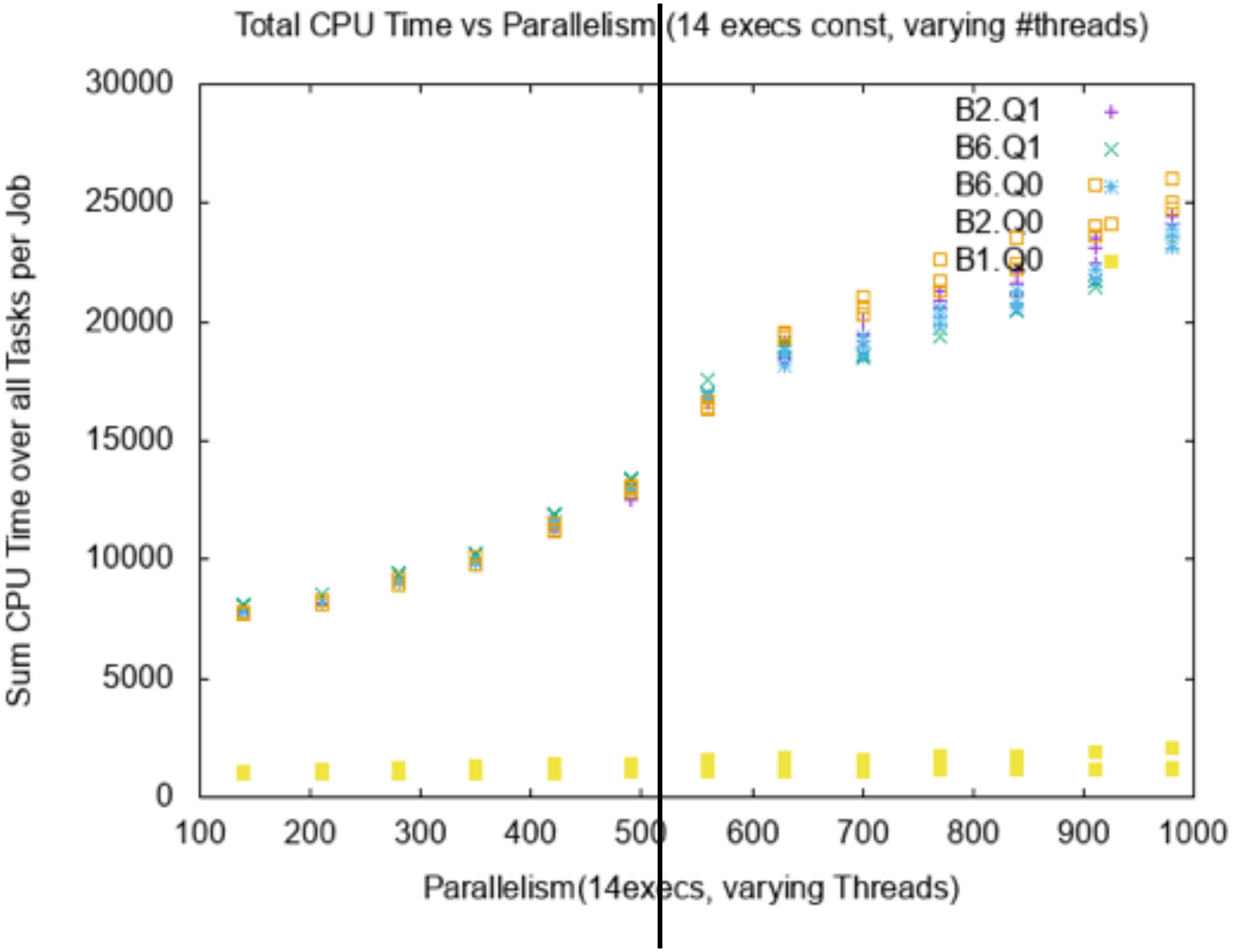
CPU Usage

Ideally, CPU usage should be constant (per query) upon increasing the parallelism.

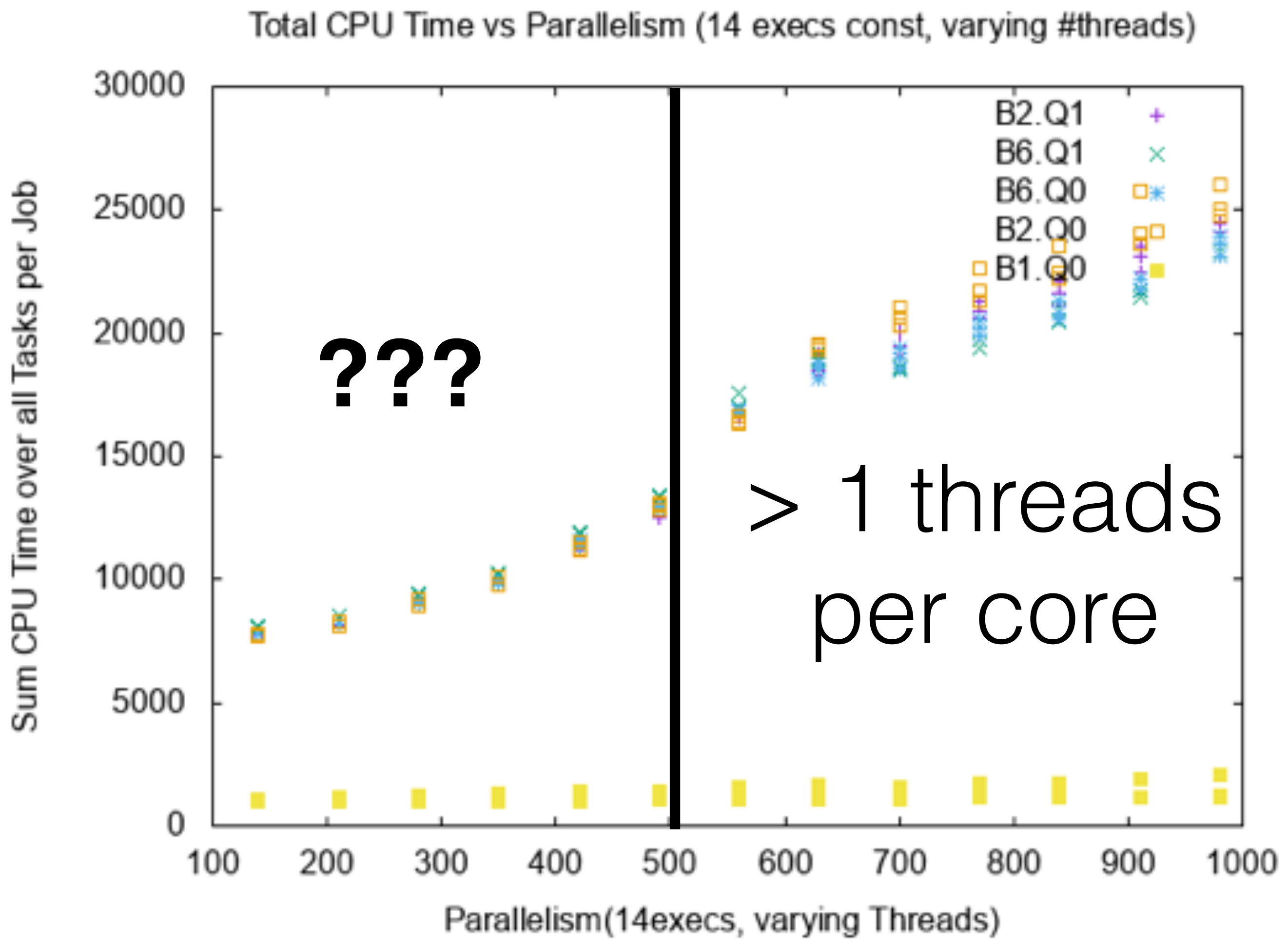


total 504
phys cores

14 execs
varying threads

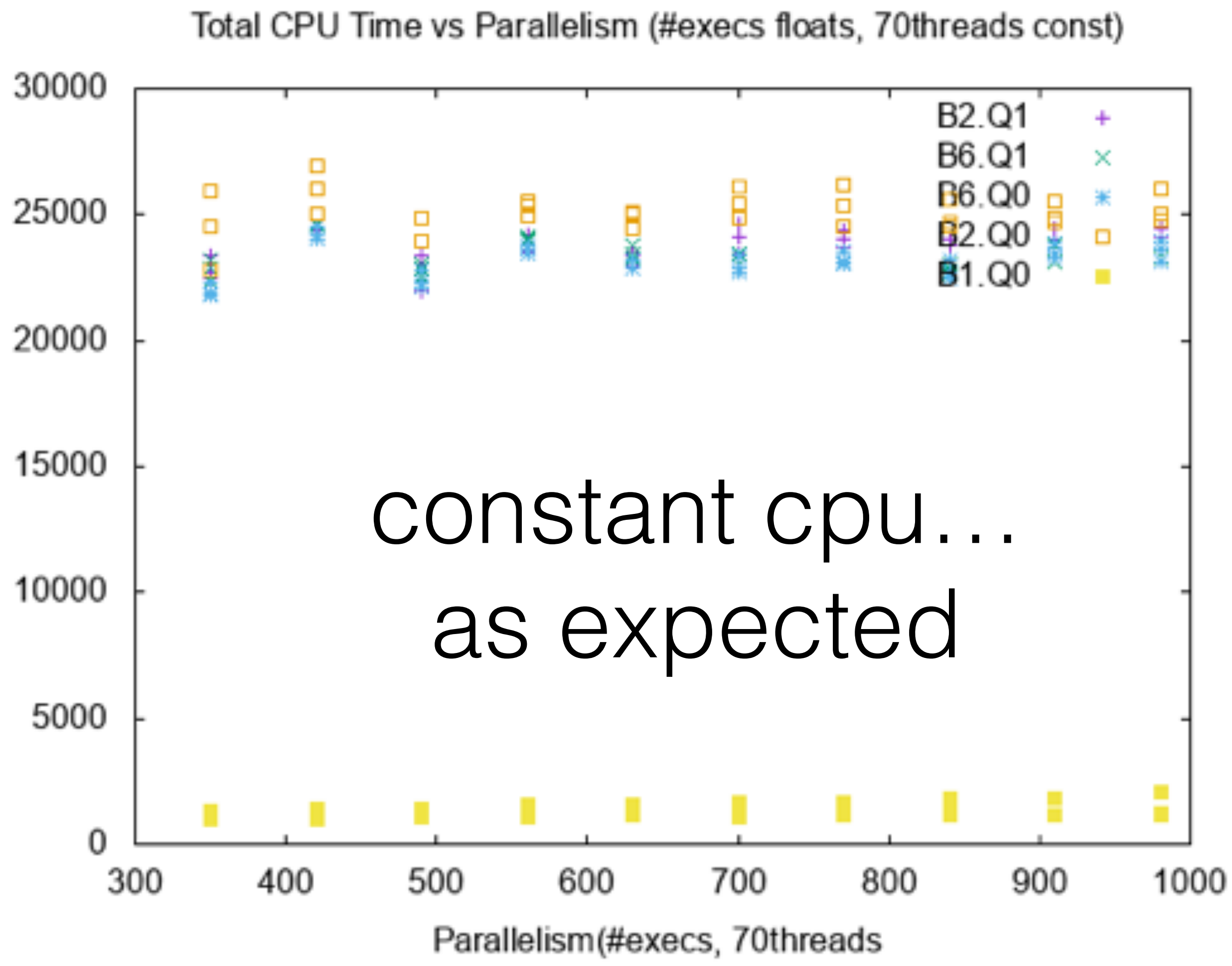


Trying to stitch pieces together.



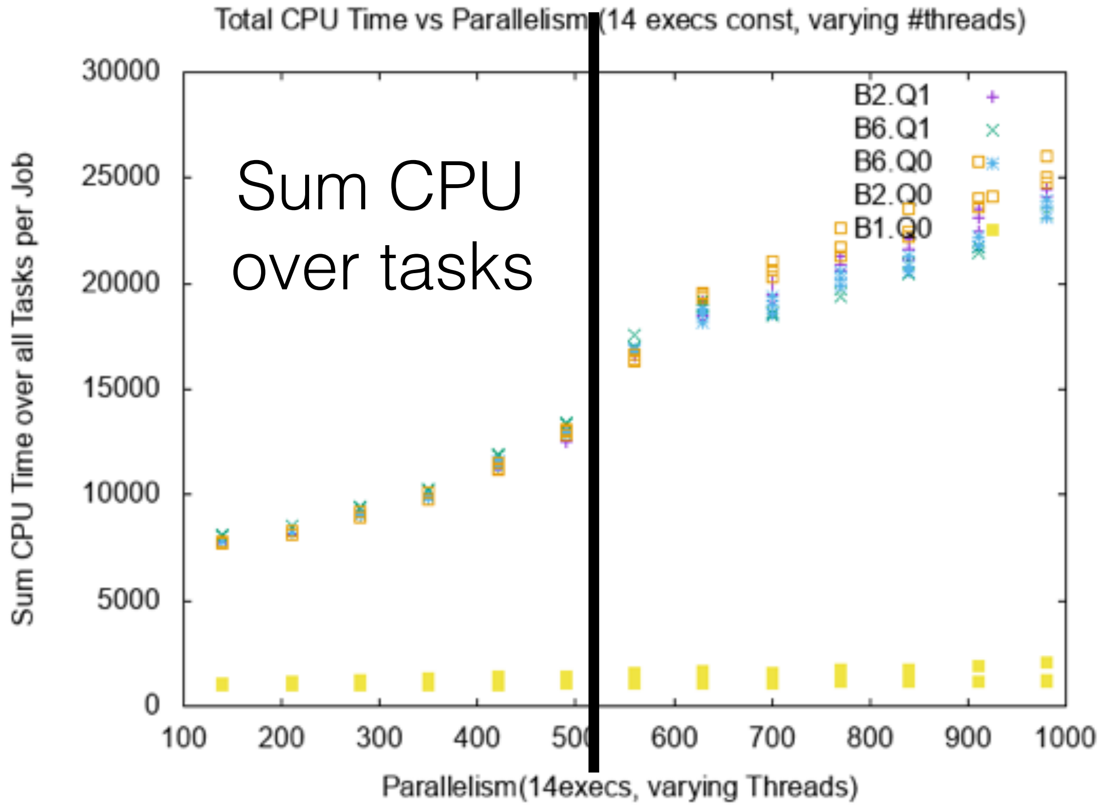
???

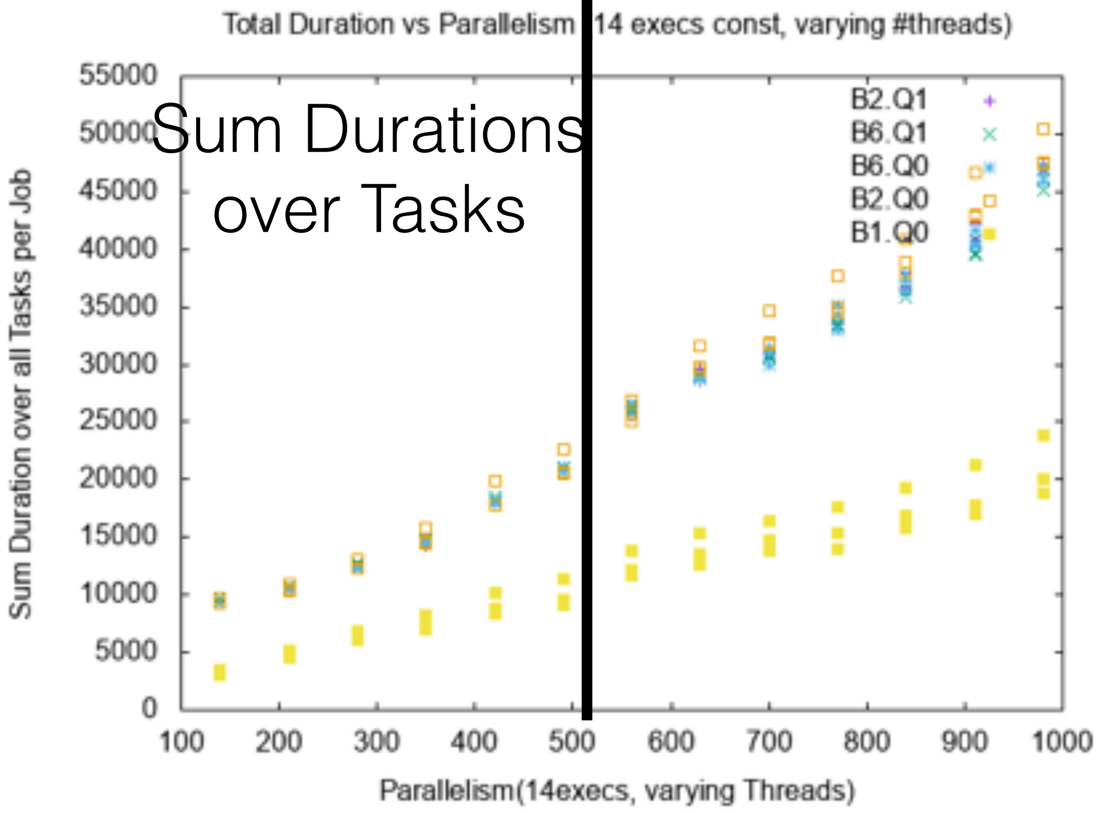
> 1 threads per core

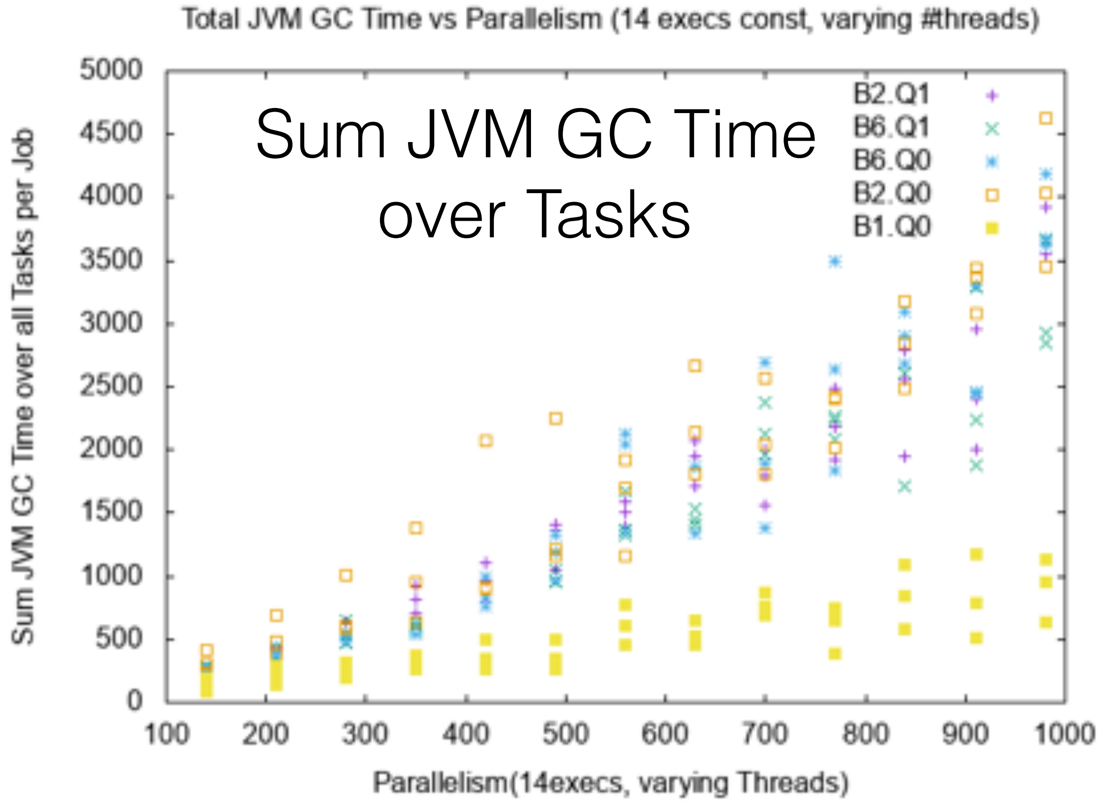


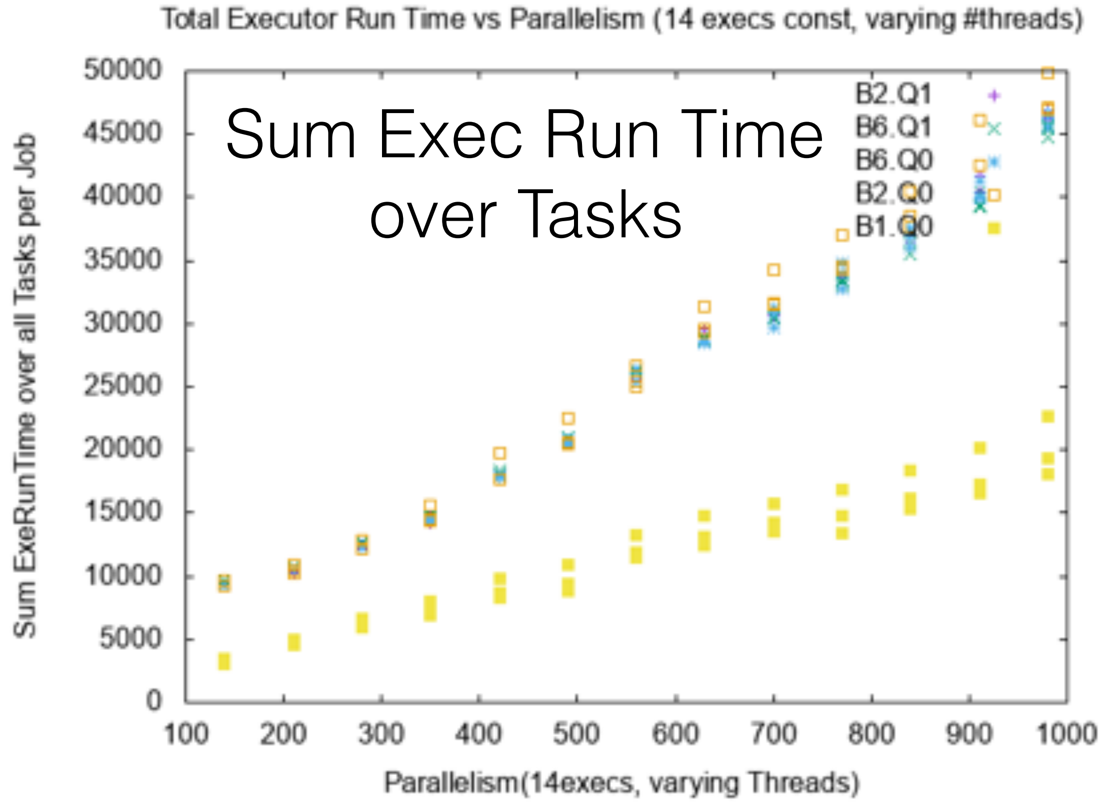
constant cpu… as expected

Other Monitorables


Sum CPU over tasks

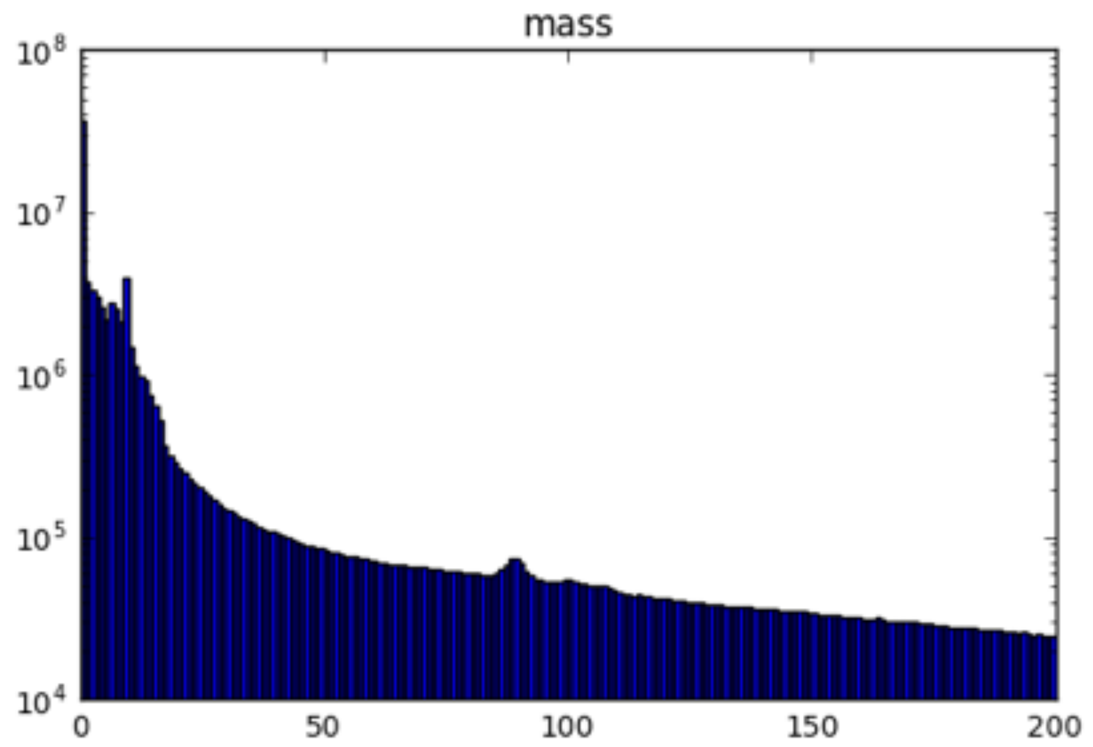

Sum Durations over Tasks

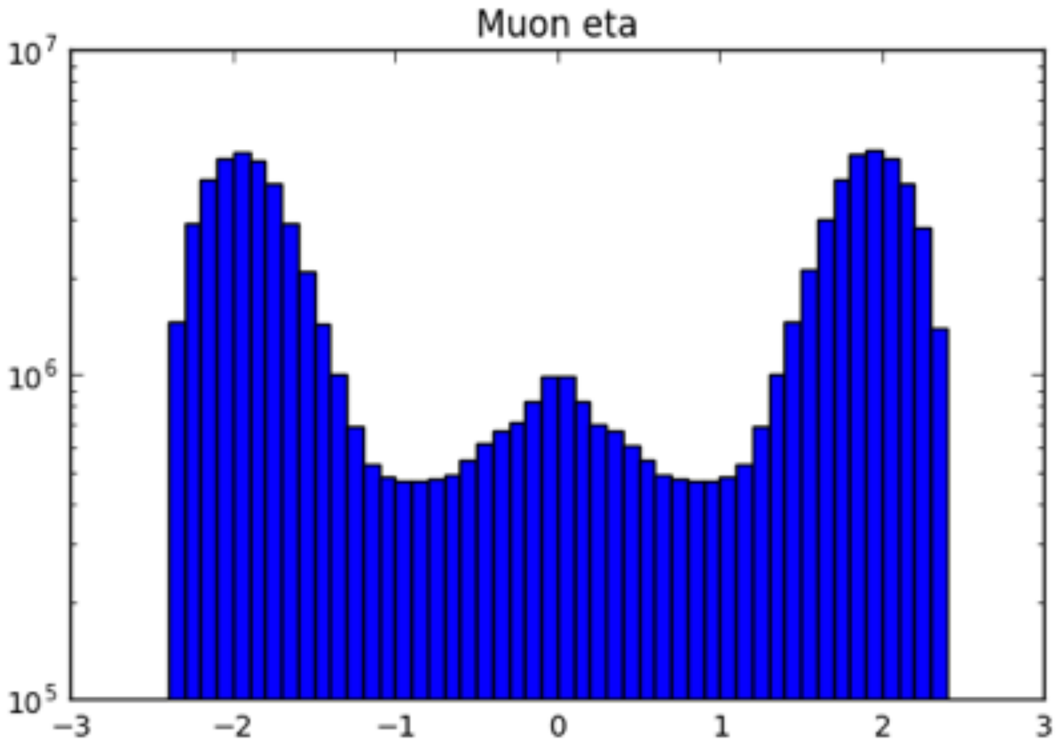

Sum JVM GC Time over Tasks

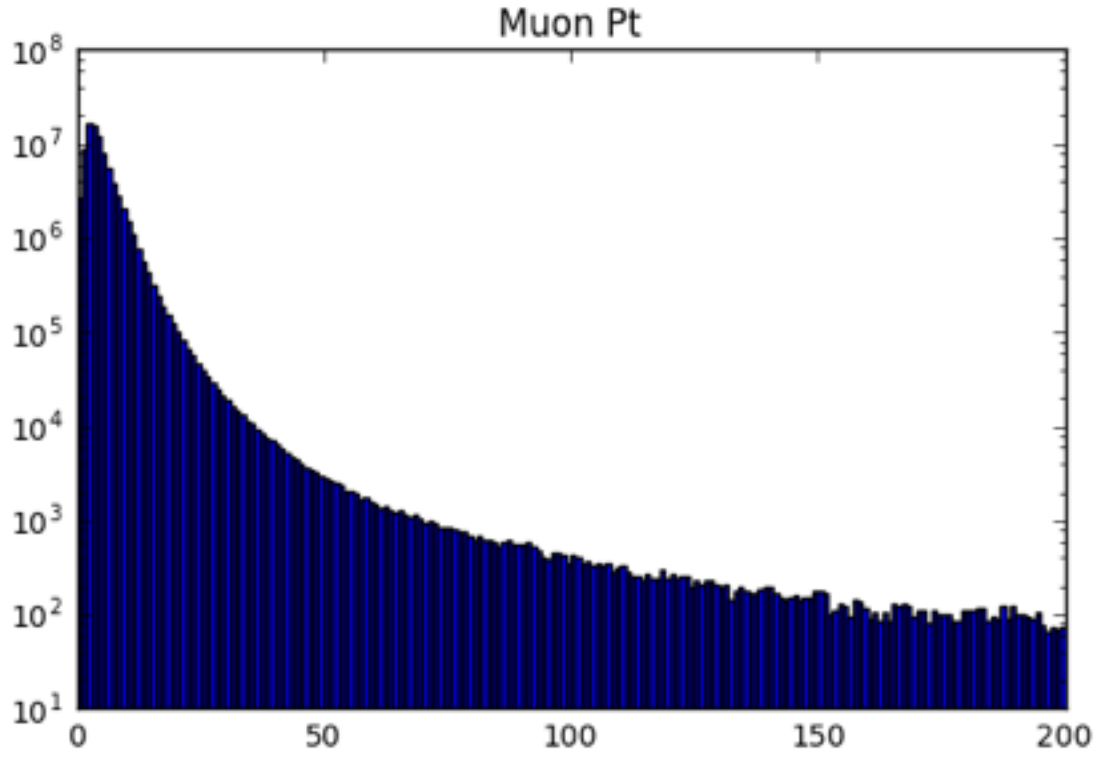

Sum Exec Run Time over Tasks

Special Examples

- Getting the to the dimuon mass + some cuts/filtering
- https://gist.github.com/vkhristenko/3bdd99716a81f2e65e1ef9bd419cb10e
- Employ spark-root + histogrammar + (ROOT/matplotlib)

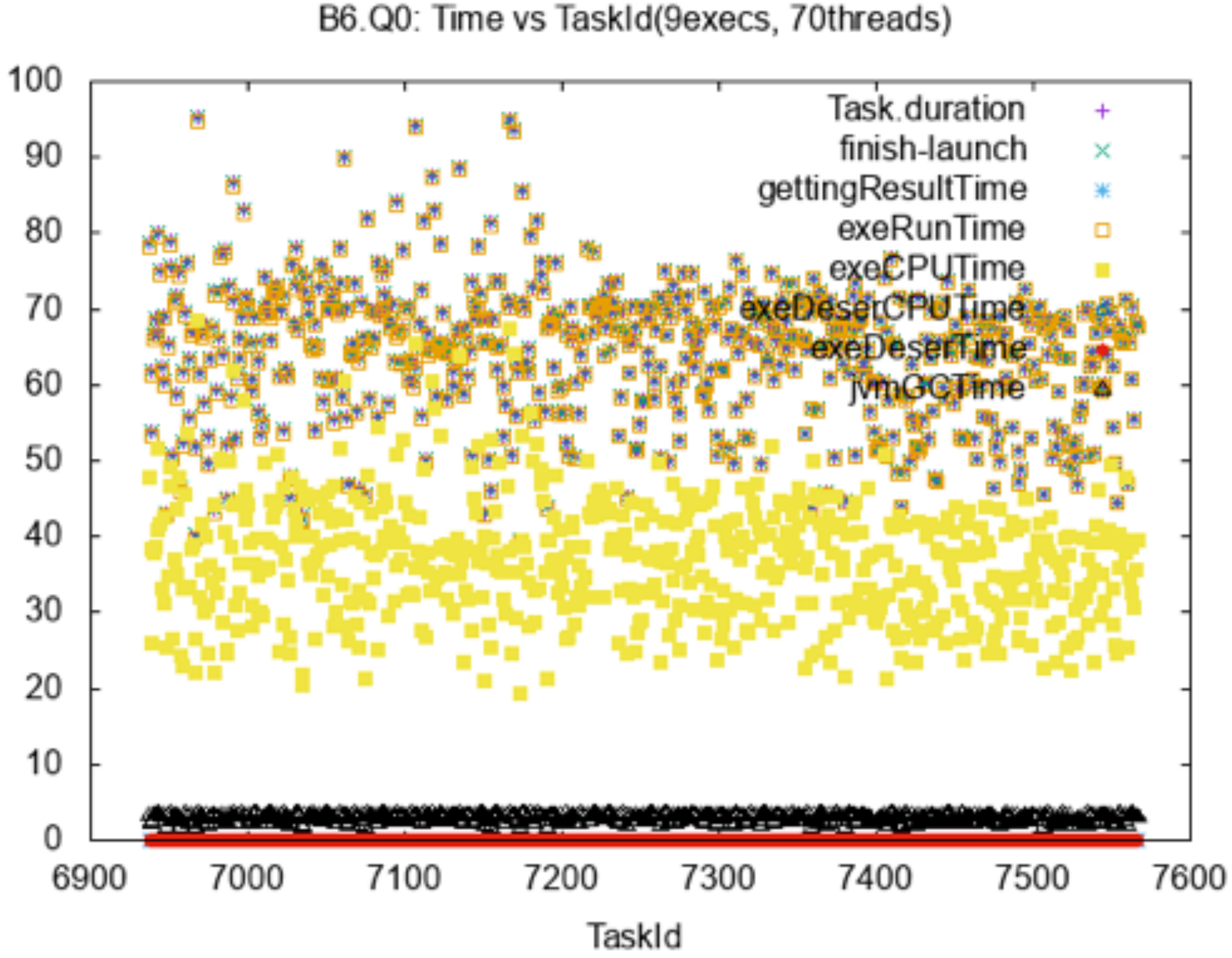


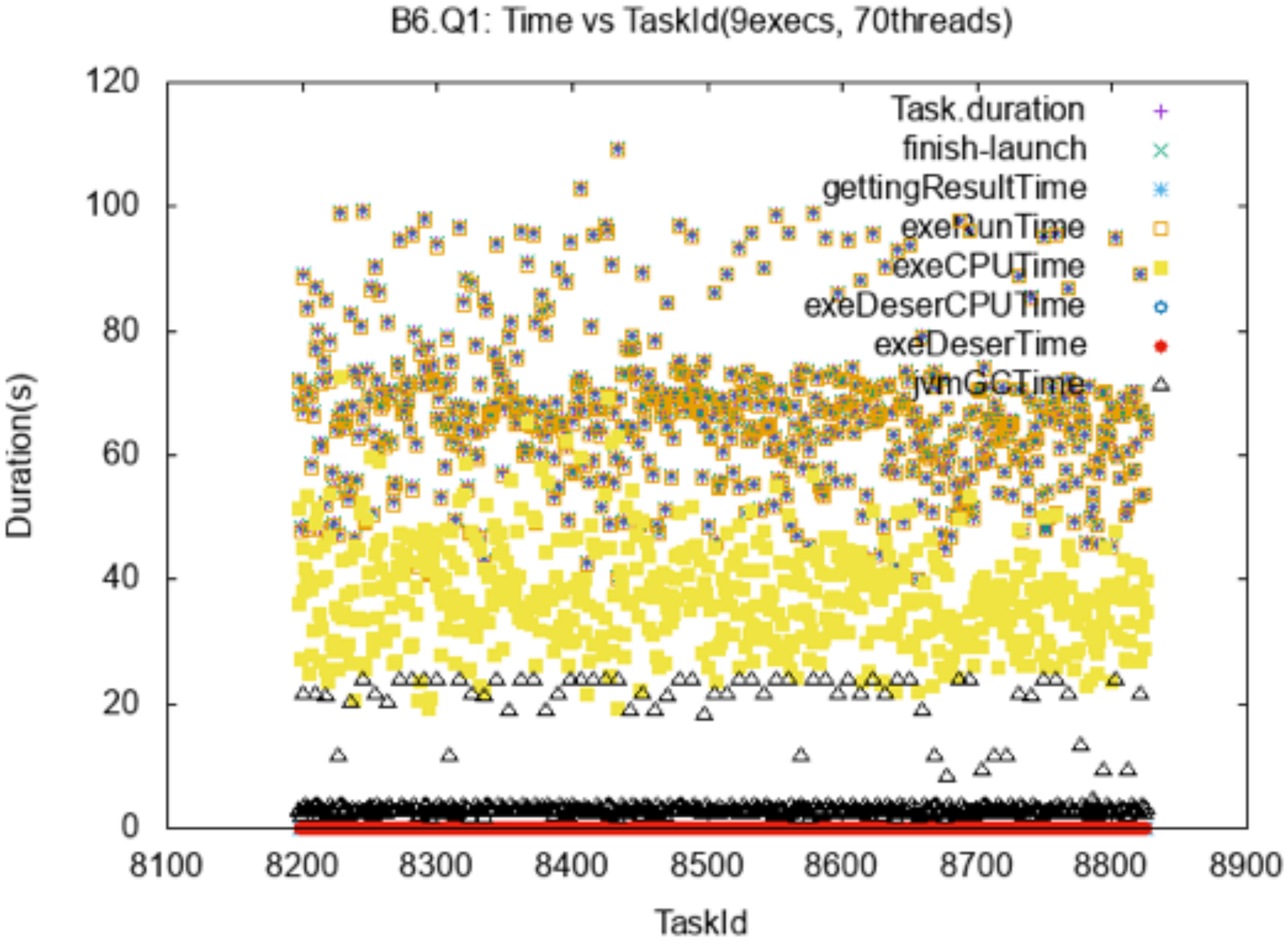




Duration vs Task Id

flatMap on just Muons, and aggregate with Histogram mar





filter (nMuons>2), build DiMuons, flatMap on DiMuons aggregate with Histogram mar

https://github.com/vkhristenko/spark-root-applications/blob/master/src/main/scala/org/dianahep/sparkrootapplications/benchmarks/AODPublicBenchmarkApp.scala for query details

Summary

- These are/is very preliminary results/report - **main idea is to learn/establish the ability to monitor what's going**
- Additional things we are looking at:
    - Business of each executor
    - Number of active tasks vs time