# Improving the analysis performance

**Andrei Gheata**

ALICE offline week

7 Nov 2013

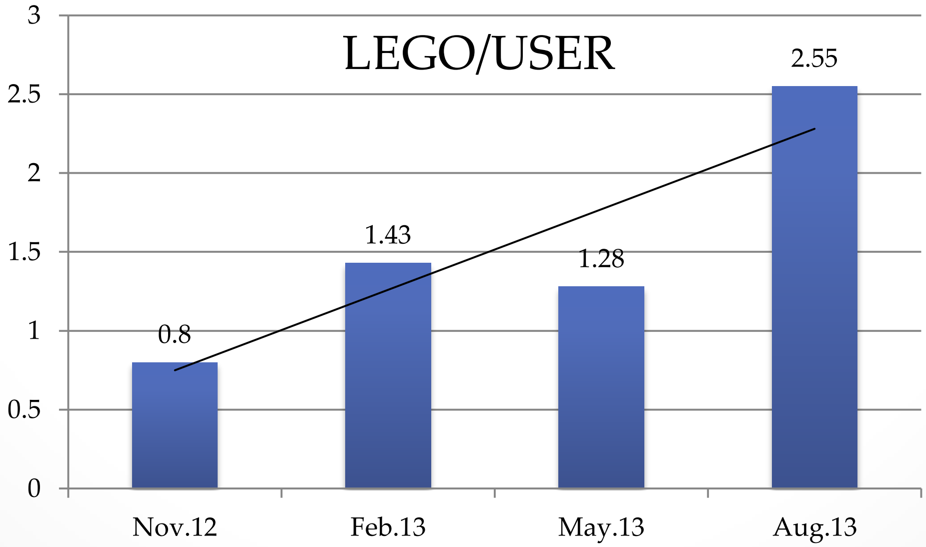
# Performance ingredients

1. Maximize efficiency: CPU/IO
   - Useful CPU cycles: deserialization is I/O accounted for as CPU
2. Minimize time to complete a given analysis
3. Maximize throughput
   - $N_{events}$/second/core for local data access

# 1. CPU/IO

# More CPU cycles

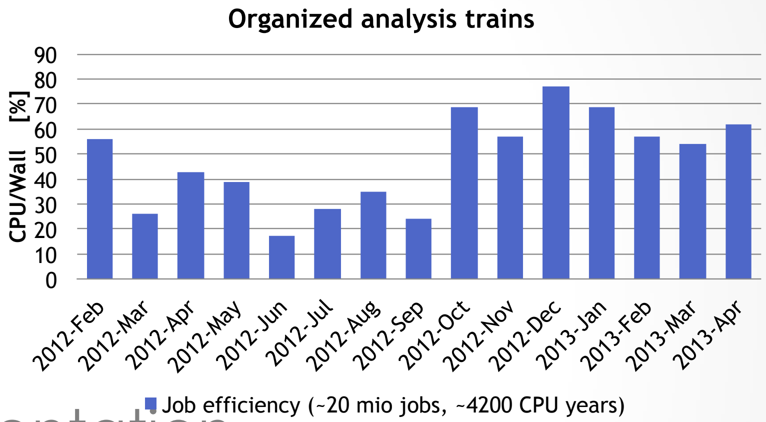
- Maximize CPU = organized analysis trains
  - At least one CPU intensive wagon gives others a free ride…
  - Many times possible in organized mode (LEGO), not the case for user distributed analysis
- Maximize participation in LEGO



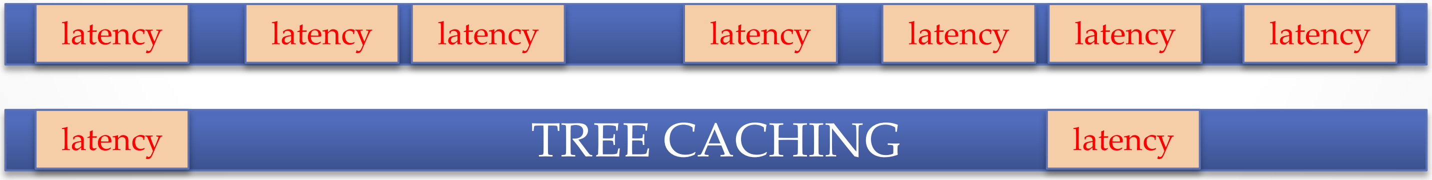
# Faster I/O

- Fragmented I/O introduces overheads
  - ~$N_{I/O\_req}$*latency
  - Killer in case of WAN file access

  (see old lessons)



- Tree caching reduces fragmentation


latency latency latency latency latency latency latency

latency TREE CACHING latency

  - Overheads still present for WAN access! Sharing WAN bandwidth does not scale as good as LAN at all sites.

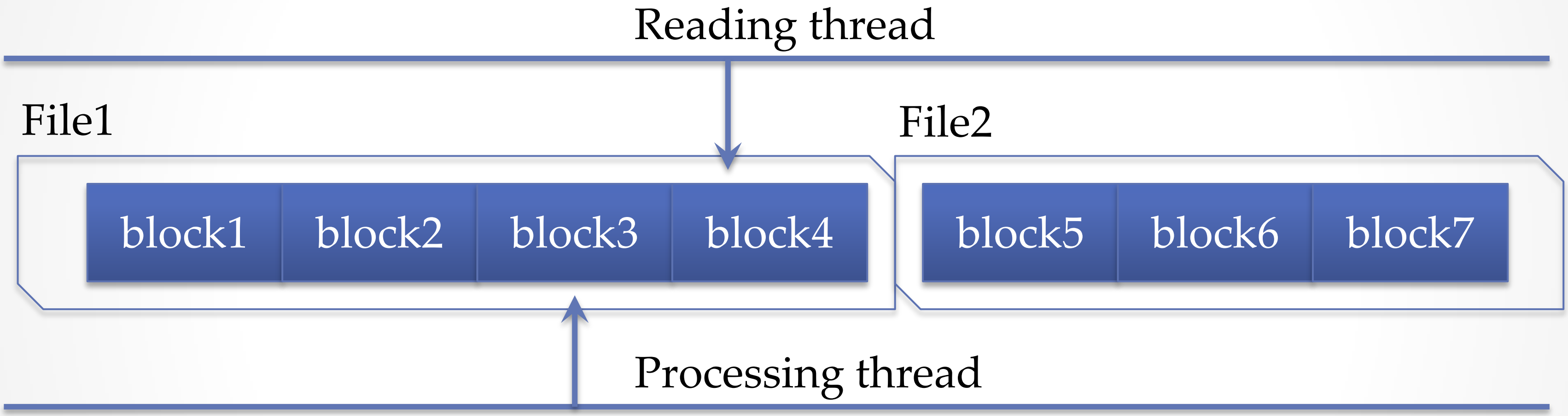
ALICE

# Maximizing local file access

- By smart file distribution and "basketizing" (splitting) per job - done
  - Locality is a feature by design
  - Algorithm fixed now after longstanding splitting issues
  - Some local file access may timeout, or jobs intentionally sent to free slots with non-local access
- Data migration?
  - Based on popularity services? Not terribly useful without a working forecast service…
  - Integrated with the job scheduling system?
- By using data prefetching
  - At low level (ROOT) – "copy during run" – to be tested at large scale
  - At workload management level (smart prefetching on proxies)?

# TFilePrefetch

- Separate thread reading ahead data blocks



- First testing round found a bug
  - The fix made it to ROOT v5-34-11 (not yet used in production)
  - To be tested with LEGO trains soon
- AliAnalysisManager::SetAsyncReading()
  - Job reading from CNAF with 70% efficiency becomes 82% !
  - Forward file opening not implemented -> non-smooth transitions between files

# Speeding-up deserialization

- "The event complexity highly contributes to DS time…"
  - …for the same amount of bytes read
  - Deserialization itself + ReadFromTree()
  - Remains to be measured
- Ongoing work: flattening *AliAODEvent* to 2 levels
  - Event + tracks, vertices, cascades, V0's, …
  - TFile::MakeFile, TFile::MakeClass as base for refactoring
  - Read event from AOD into new structure + write to new file
  - Compare reading speeds in the 2 cases
- For the future: new "thinned" AOD format serving 80% of analysis
  - The train model calls for "general" data -> contradicts with reducing size
  - To investigate: SOA for tracks, vertices, V0's, …

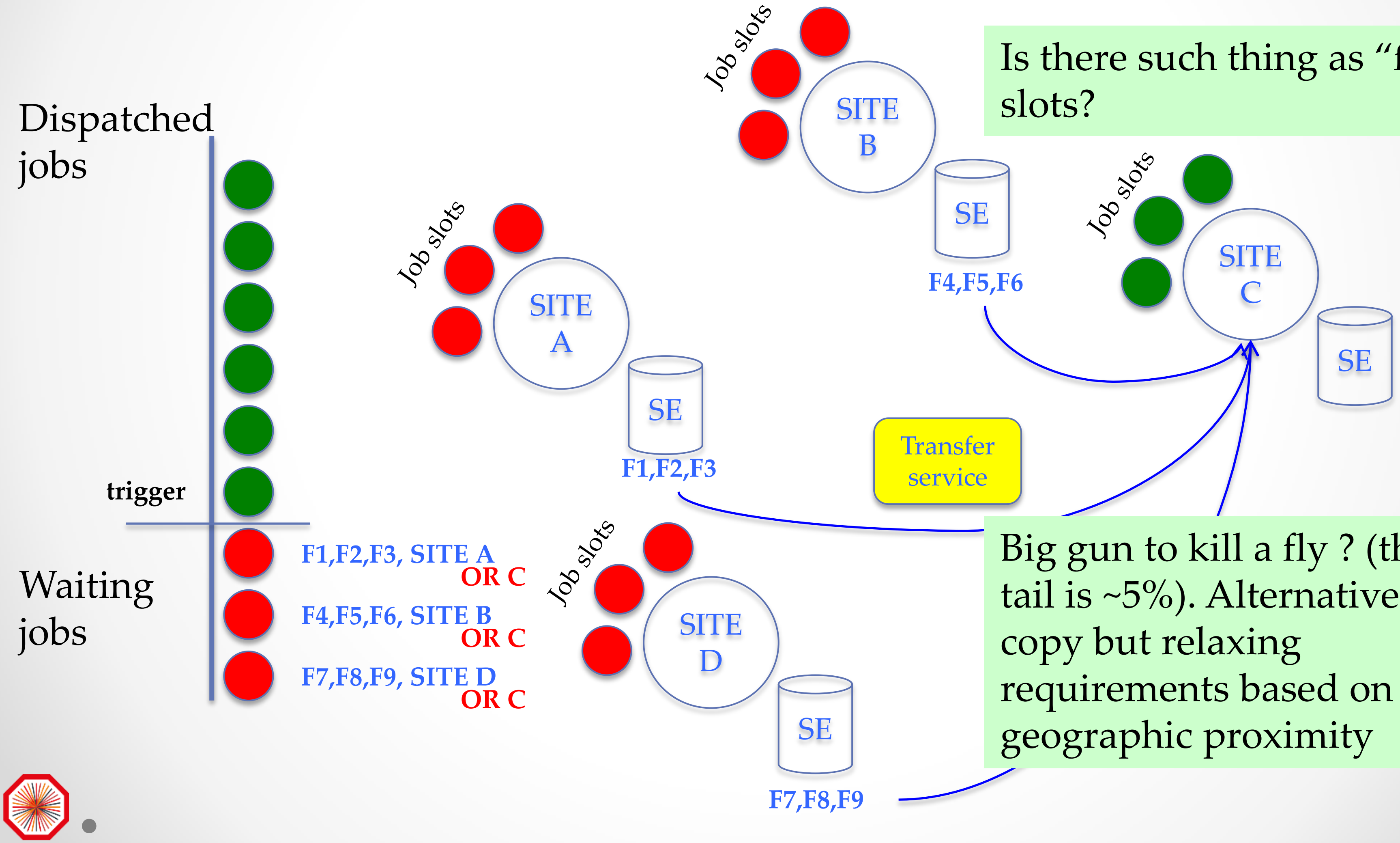
# Minimizing time to complete analysis

…

# Improving tails…

- Assess acceptable statistics loss cutting the tail
- Jobs in the queue not finding free slots where data is
  - Currently the site requirements are being released and jobs land on some free slot and access (almost) all files remotely -> efficiency price
  - One can also play with raising the priority for the tail jobs
- Prefetching jobs file lists on data proxies near free slots
  - Triggered by low watermark on remaining jobs or high watermark on waiting time
  - Change job requirements to match the data proxy
  - Better than local job file prefetching because can be done while jobs are waiting
- Just ideas to open the discussion…
  - Implementation would require a data transfer service and xrootd-based proxy caching
  - Can bring the efficiency up, but also reduce the time to finish the jobs

# Possible approaches



Is there such thing as “free slots?

Big gun to kill a fly ? (the tail is ~5%). Alternative: no copy but relaxing requirements based on geographic proximity

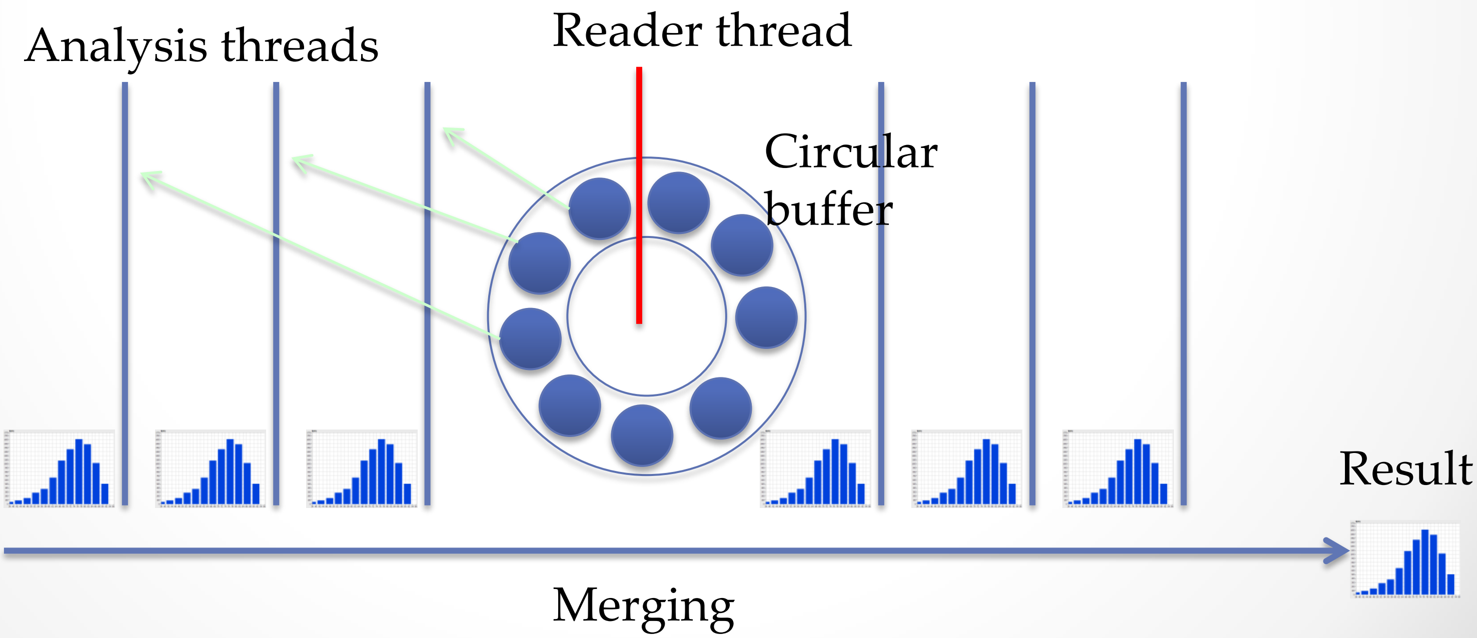
# Throughput/core

# Prerequisites

- Achieving micro parallelism
  - Vectors, instructions pipelining, ILP
  - Some performed by HW and compilers, other require explicit intervention
- Working on "parallel" data (i.e. vector-like)
  - Redesign of data structures AND algorithms
  - Data and code locality enforced at framework level, algorithm optimizations at user level
- Make use of coprocessors (GPGPU, Xeon Phi, ...)
  - Require parallelism besides vectorization (including at I/O level)
- "Decent" CPU usage
  - CPU bound tasks

# Does it worth?

- Nice exercise by Magnus Mager reshaping a three-prong vertexing analysis
  - Re-formatting input data and keeping minimal info, feeding threads from a circular data buffer, some AVX vectorization, custom histogramming
  - 500 MB/s processed from SSD



# Micro-parallelism path

- Data structures re-engineering: shrink and flatten, use SOA and alignment techniques runtime
- Concurrency in data management: reading, dispatching to workers
- Define work unit: e.g. vector of tracks, provide API with vector signatures
- Concurrent processing: thread safe user code, usage of vectorization, kernels

# Conclusions

- Analysis performance has multiple dimensions, we are addressing few
- Data management policy require improvements to extra reduce time to analysis completion while staying efficient
- File prefetching expected to bring efficiency up with extra 10%
- A long path in future towards micro-parallelism  in analysis