Argonne NATIONAL LABORATORY

I/O challenges for HEP applications on multi-core processors

An ATLAS Perspective

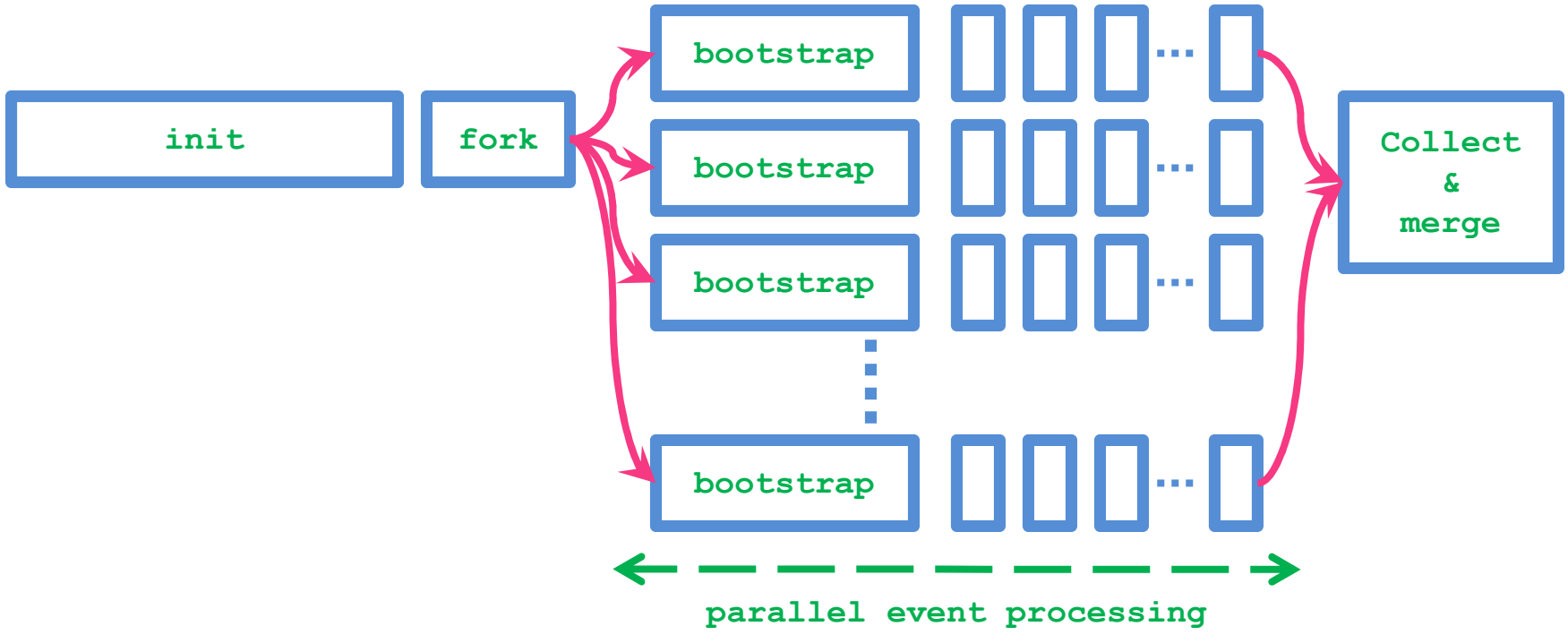
Mini-Workshop on multi-core joint project

Peter van Gemmeren (ANL)

U.S. DEPARTMENT OF ENERGY

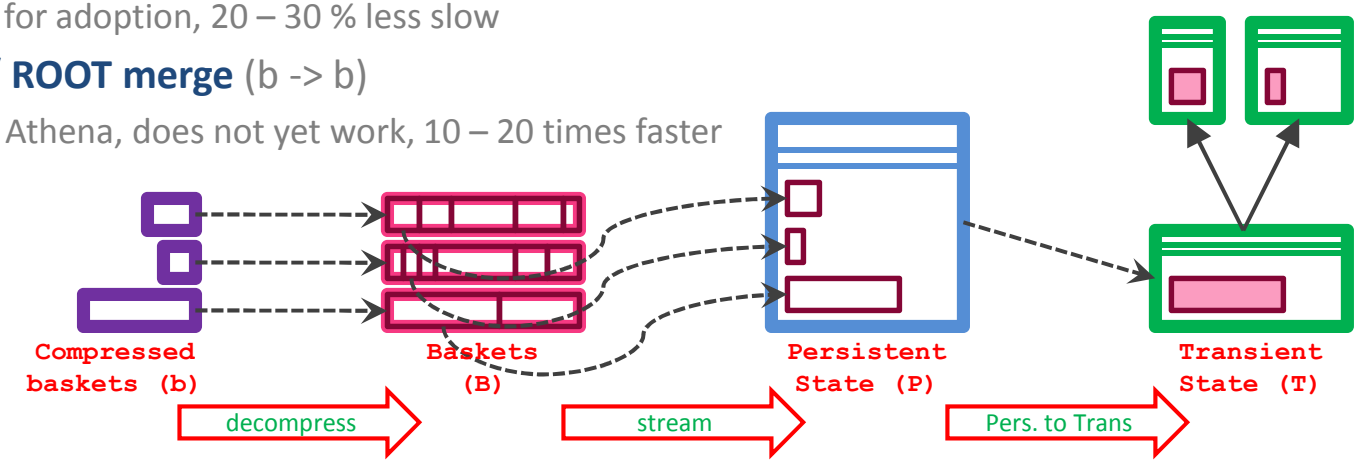
AthenaMP: The Process point of view

- **AthenaMP** is the ALTAS multi-core framework
- Diagram taken from Sebastien Binet (thanks).



Merge

- In **AthenaMP** each worker node produces it **own output file**, which need to be **merged** after all worker are done.
  - Done in **serial**, can take significant amount of wall clock time.
- For ATLAS, which uses transient / persistent separation and data stored in POOL / ROOT there are 3 different merge level:
  - **Athena merge** (b -> B -> P -> T -> T -> P -> B -> b)
    - Currently used, slow
  - **Direct persistent to persistent copy in Athena** (b -> B -> P -> P -> B -> b)
    - Waiting for adoption, 20 – 30 % less slow
  - **Fast POOL / ROOT merge** (b -> b)
    - Outside Athena, does not yet work, 10 – 20 times faster



Full Athena merge
(b -> B -> P -> T -> T -> P -> B -> b)

- Runs **inside Athena Event Loop**:
  - Uncompress ROOT baskets, rebuild persistent objects.
  - Convert persistent to transient objects
  - Do nothing with transient objects
  - Convert transient to persistent objects
  - Stream persistent objects into ROOT baskets and compress
- Produces Output File which is very similar to one produced by a single core processing all input events.
  - **Handles Metadata propagation** using full Athena framework.
    - However, there may be metadata which is irreproducible from many incomplete jobs (e.g., lumiblock information)
  - **Re-optimizes persistent Data Objects**
    - E.g.: The file has one GUID which is used in all the DataHeader (and read only once for all events)
  - **Re-optimizes ROOT baskets**
    - Small baskets are combined and re-compressed.

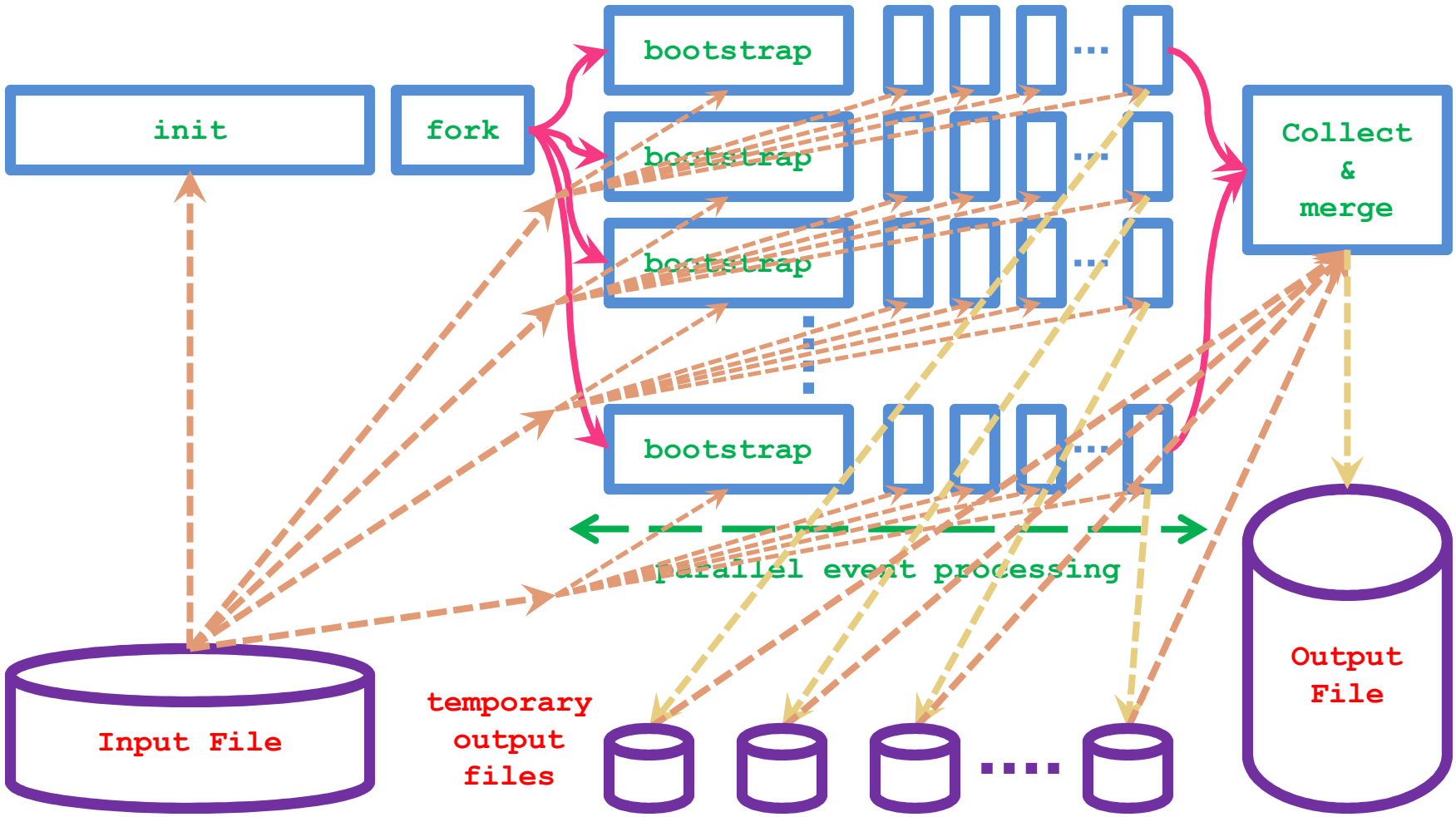
Direct persistent to persistent copy in Athena (b -> B -> P -> P -> B -> b)

- **Same as full Athena**, but **without** P -> T and T -> P
  - Uncompress ROOT baskets, rebuild persistent objects.
  - Do nothing with persistent objects
  - Stream persistent objects into ROOT baskets and compress
- Can do P -> T and T -> P for selected objects
- Creates **new DataHeader** and re-optimizes its persistent representation (as in full athena)
- Produces Output File which is very similar to one produced by a single core processing all input events.
  - Handles Metadata propagation using full Athena framework.
  - Re-optimizes **selected** persistent Data Objects
  - Re-optimizes ROOT baskets

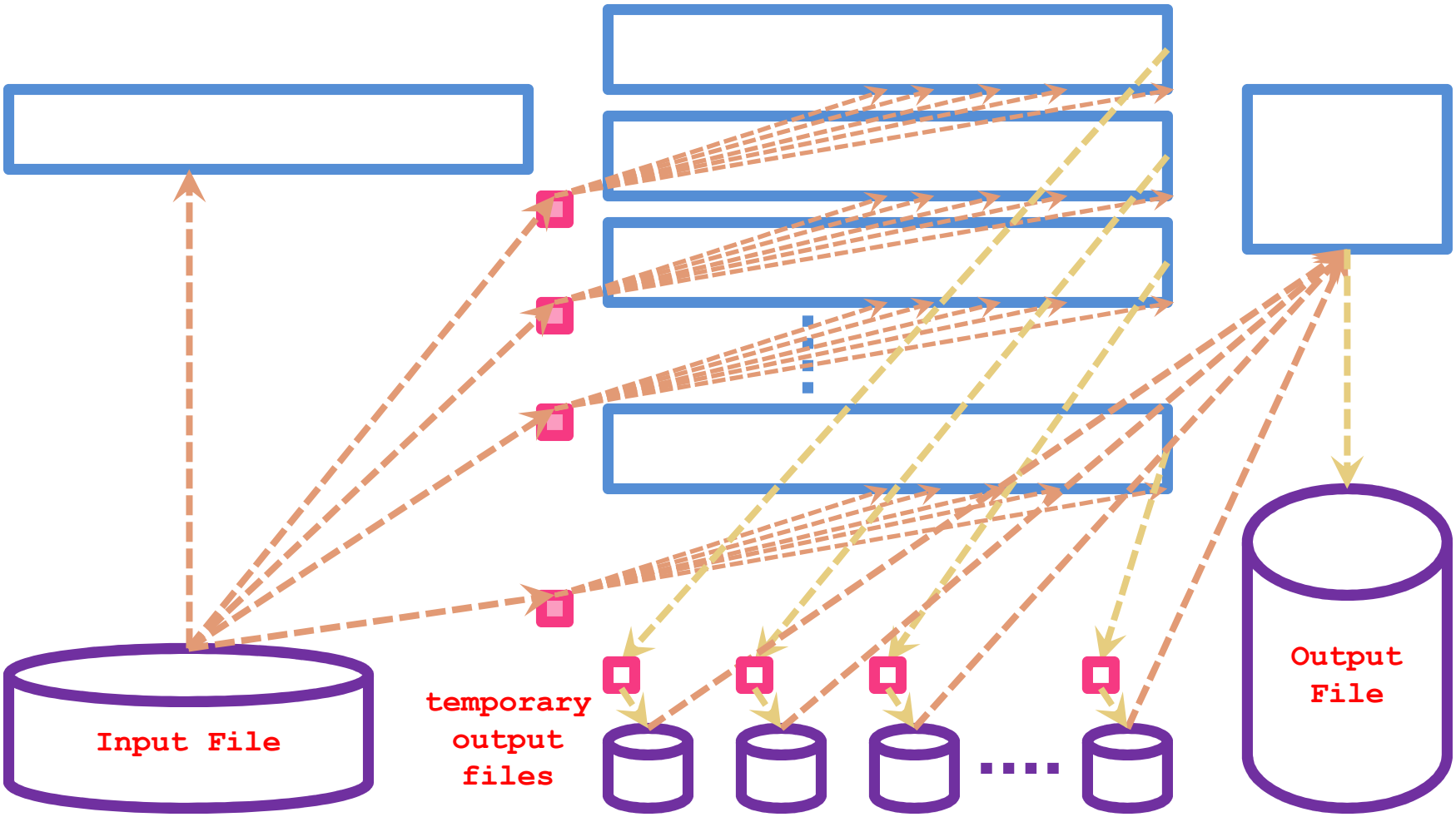
Fast POOL / ROOT merge (b -> b)

- Run as **separate POOL application**, not using the Athena framework
- Does not **currently** work for ATLAS events, using external tokens
  - Recent (CVS head -> POOL nightly) modifications by Markus Frank and myself promise to enable POOL fast merge for ATLAS events
- Even when POOL fast merge works, there are **inherent limitations**:
  - Can't **Summarize metadata**
    - Metadata records have to be summarized on every read.
  - **Navigational references** are not updated
    - Utilize POOL redirection, which may cost time.
  - **Re-optimize storage layout** (-> non-optimal compression, slower read speed for some objects)
    - In principle this could be overcome, by having ROOT uncompress, combine and recompress the buffer (b -> B -> B -> b). However this will slow down the merge significantly.

AthenaMP: The I/O point of view



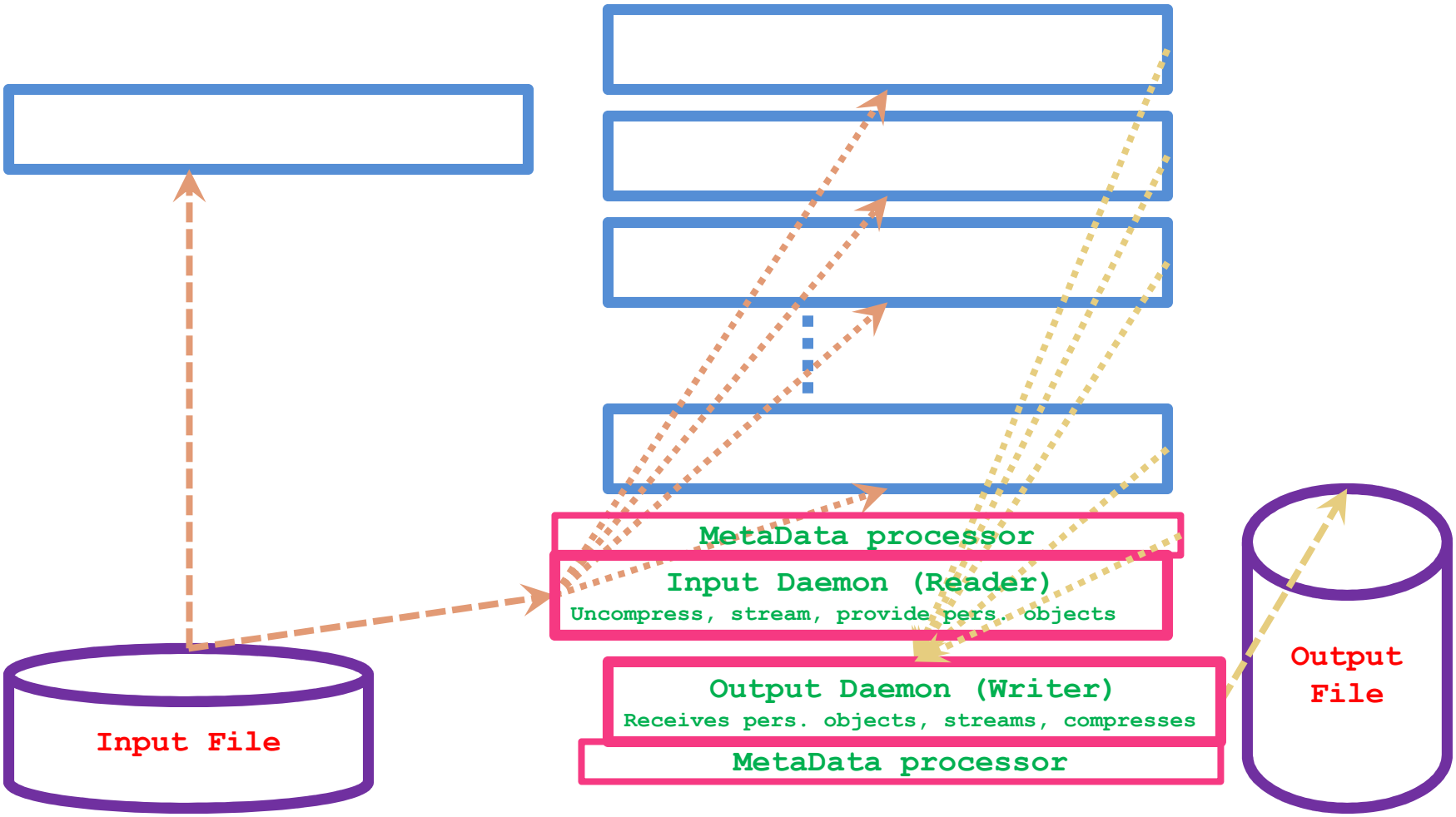
AthenaMP: The I/O point of view, to be fair



So what's wrong with that?

- Read data: ⇢ A process (initialization, event execute,…) reads part of the input file (e.g., to retrieve one event).
  - All worker use the same input file, which in general is to large to be cached in memory.
  - Multiple access may mean **poor read performance**, especially if events are not consecutive.
- Uncompress / Stream ROOT baskets: ■ Each reader (i.e. worker) will retrieve its event data, which means reading multiple ROOT baskets, uncompressing them and streaming them into persistent objects.
  - ROOT baskets contain object member of several events, so multiple worker may use the same baskets and each of them will uncompress them independently:
    - **Wastes CPU time** (multiple uncompress of the same data)
    - **Wastes memory** (multiple copies of the same Basket, probably not shared)
- Write data: ⇢ Each process writes its own output file.
- Compress / Stream to ROOT baskets: ■ Writer compress data separately.
  - Suboptimal compression factor (which will cost storage and CPU time at subsequent reads)
    - This can be 'healed' later by using Athena merge.
  - **Wastes memory** (each worker needs its own set of output buffer)

AthenaMP -> ~GaudiParallel: The Scatter/Gather point of view



AthenaMP -> ~GaudiParallel:

- **GaudiParallel** follows **MPI Scatter / Gather** scheme
  - Reduces I/O, total memory, CPU & Wall Clock time and even disk space
  - I/O uses new Reader / Writer classes
    - Serialize whole event from Transient Data Store and send it to worker process
- **Athena**
  - Uses StoreGate (flat data store) instead of Gaudi Transient Data Store (hierarchical).
    - StoreGate (and ATLAS persistency) support C++ classes with all their capabilities / complexity
  - Implements Transient / Persistent separation in persistency framework.
    - Could use t-p conversion as locus for scatter / gather to allow **on demand retrieval of objects**
      - Rather than sending complete events, the persistent state of objects can be send.
      - Persistent state representations are **much simpler** than their transient counterparts.
  - Have dedicated **processing of in-file metadata** attached to Reader and/or Writer
- To Be Addressed:
  - Event ordering
    - Multi-core processed file may not preserve the event ordering of input files.
  - Removing any duplicate events

Summary

- ATLAS uses **AthenaMP** to utilize multi-core processors
  - Works today
  - Very process focused
    - I/O more of an afterthought
      - Currently limited to merging output files
    - I/O becoming bottleneck:
      - Even for reconstruction up to 20 – 30 % wall clock time in merge
      - Increases memory foot print by several 100 MB per core
- Program of work needed to parallelize I/O for multi-core processor
  - Several design alternatives should be investigated
  - Emerging I/O capabilities in GaudiParallel should be used for Athena (if it makes sense).
    - Of course differences between Gaudi and Athena mean there is work to do on our end
- Design of multi-core I/O needs to be highly configurable to allow performance tuning.
  - Expect multi-core I/O performance tuning to be a significant effort
    - Similar to single core I/O tuning or (multi-core) memory/CPU optimization.