GPU @ ALICE

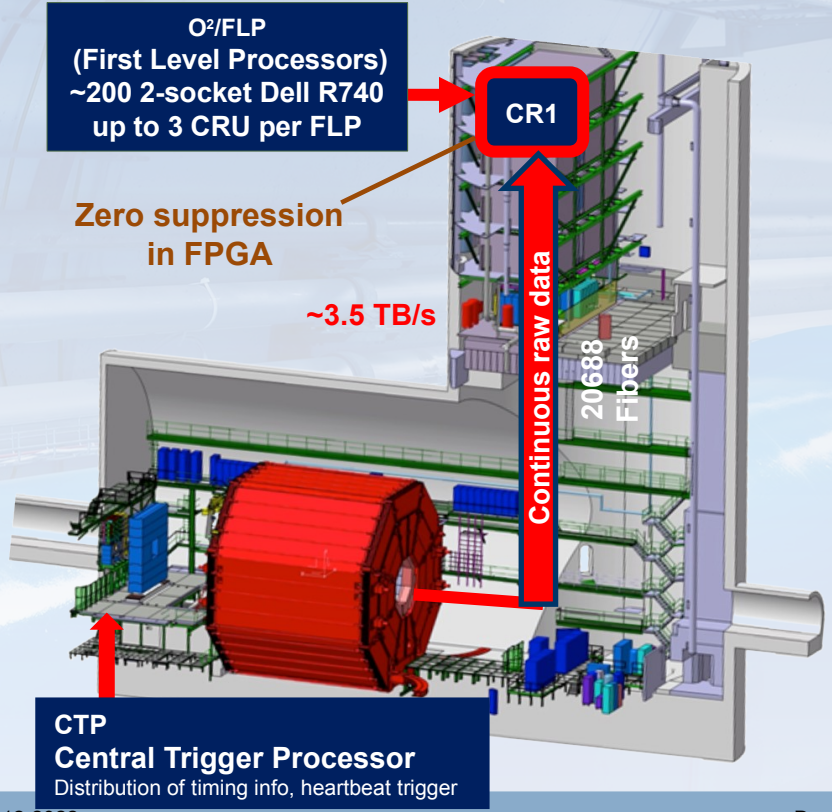
David Rohr

7.12.2022

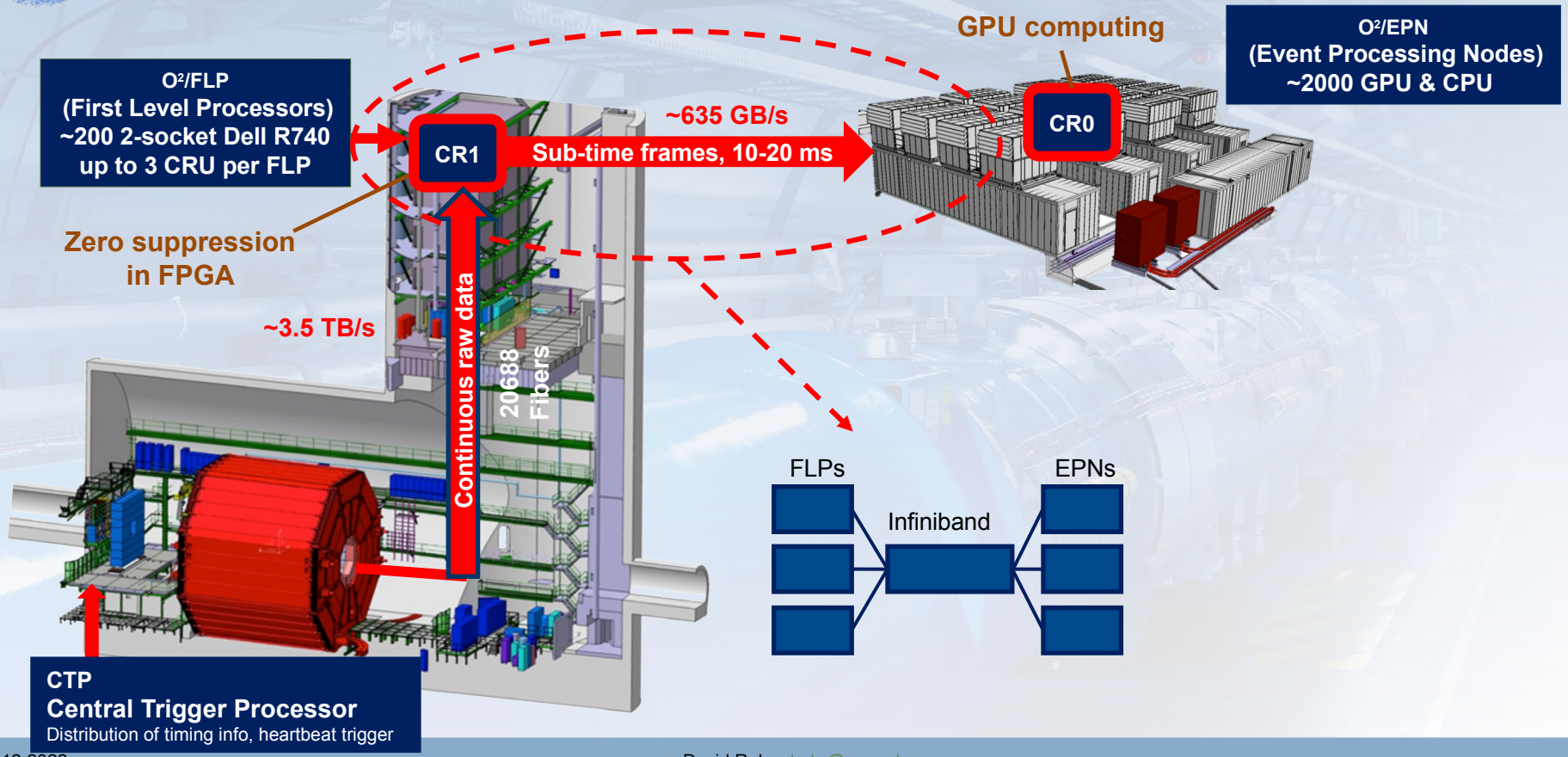
Anual ALICE USA Computing Project Fall Meeting

drohr@cern.ch

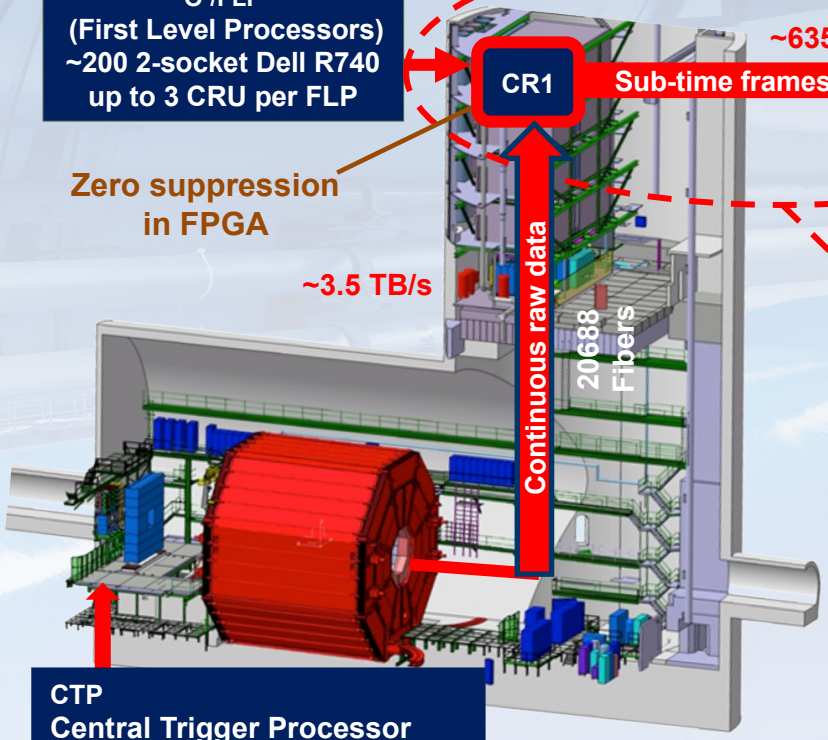
ALICE Data Flow in Run 3



ALICE Data Flow in Run 3



ALICE Data Flow in Run 3



O²/FLP
(First Level Processors)
~200 2-socket Dell R740
up to 3 CRU per FLP

CR1

~635 GB/s
Sub-time frames, 10-20 ms

GPU computing

CR0

O²/EPN
(Event Processing Nodes)
~2000 GPU & CPU

Zero suppression
in FPGA

~3.5 TB/s

Continuous raw data

20688
Fibers

~100 GB/s
CTF: Compressed time frames

Calibration data

80 PB
disk storage, 360GB/s
(~25% redundancy)

~70 % CTF

~30 % CTF

Tier 0
archival

Tier 1
archival

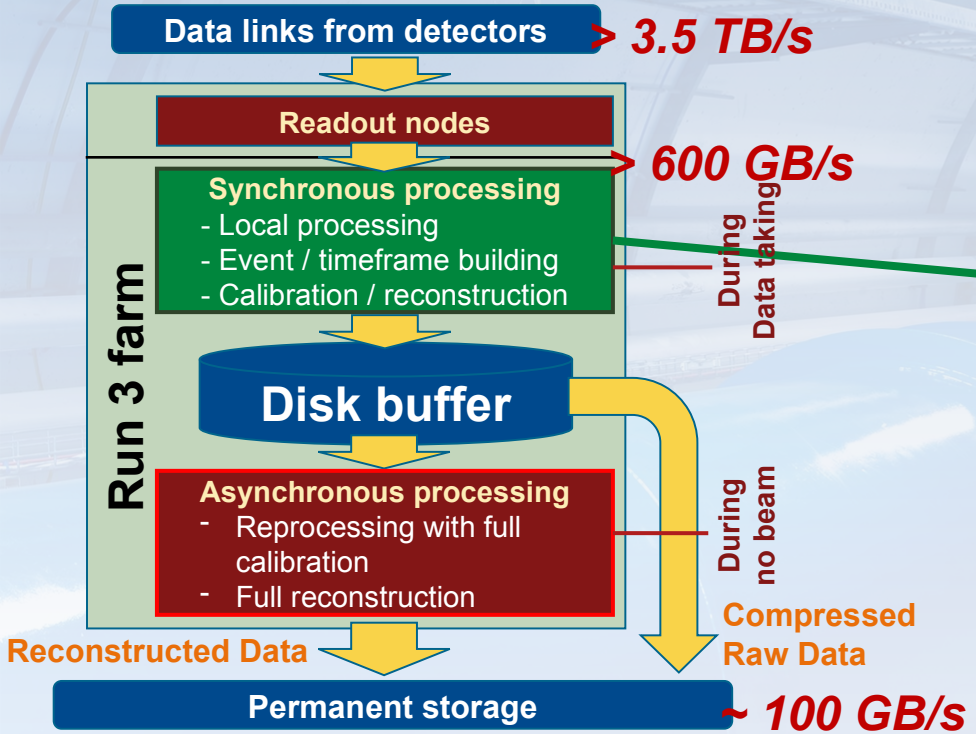
FLPs

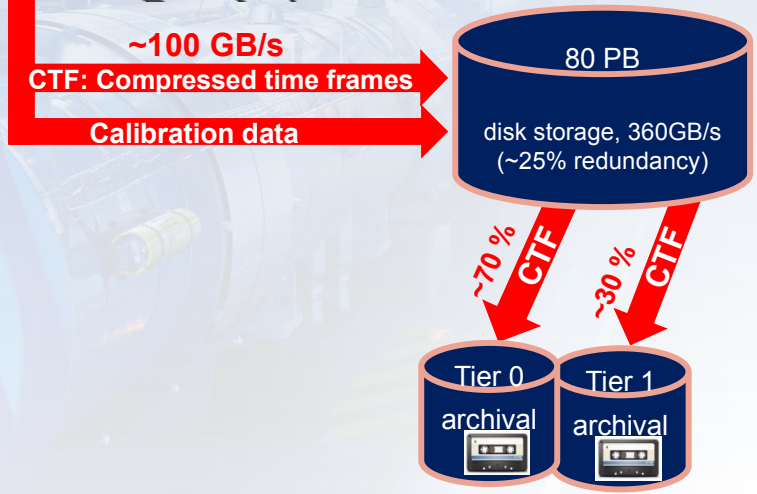
Infiniband

EPNs

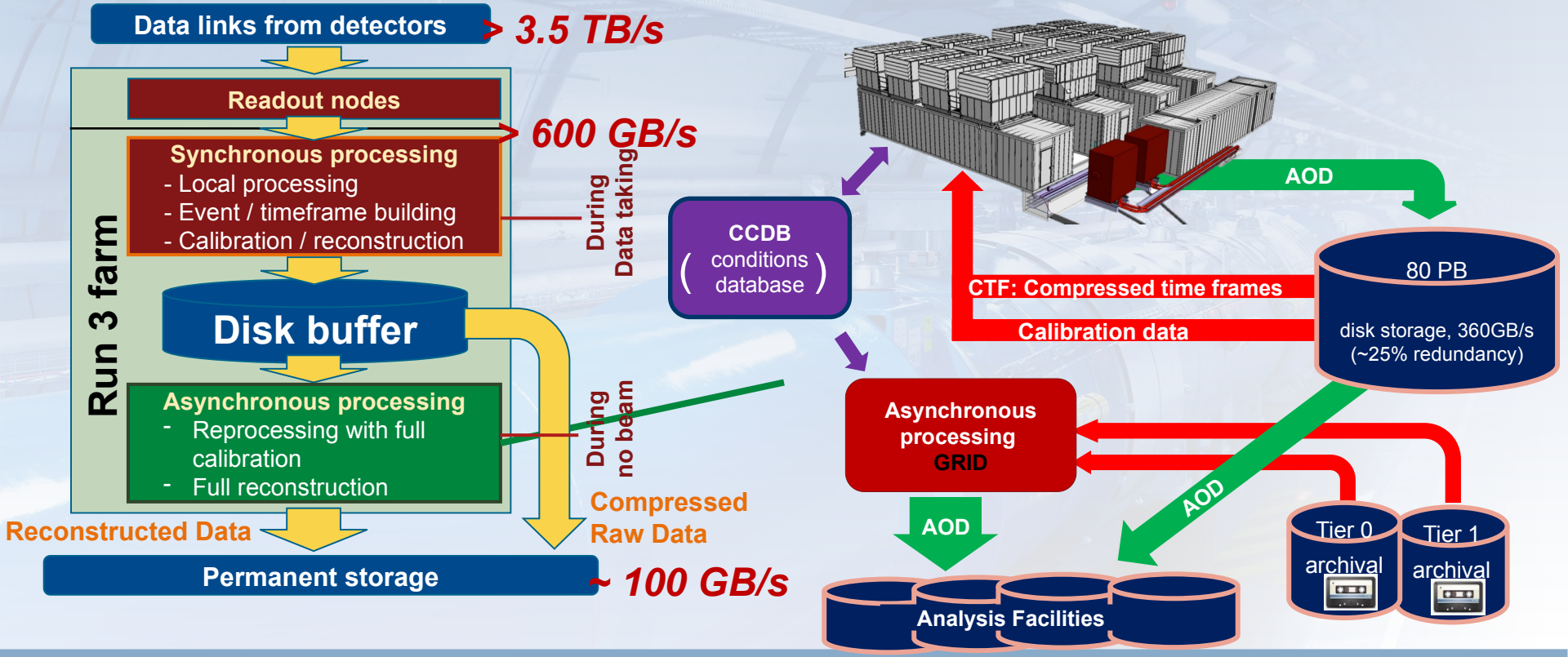
CTP
Central Trigger Processor
Distribution of timing info, heartbeat trigger

Synchronous and Asynchronous Reconstruction

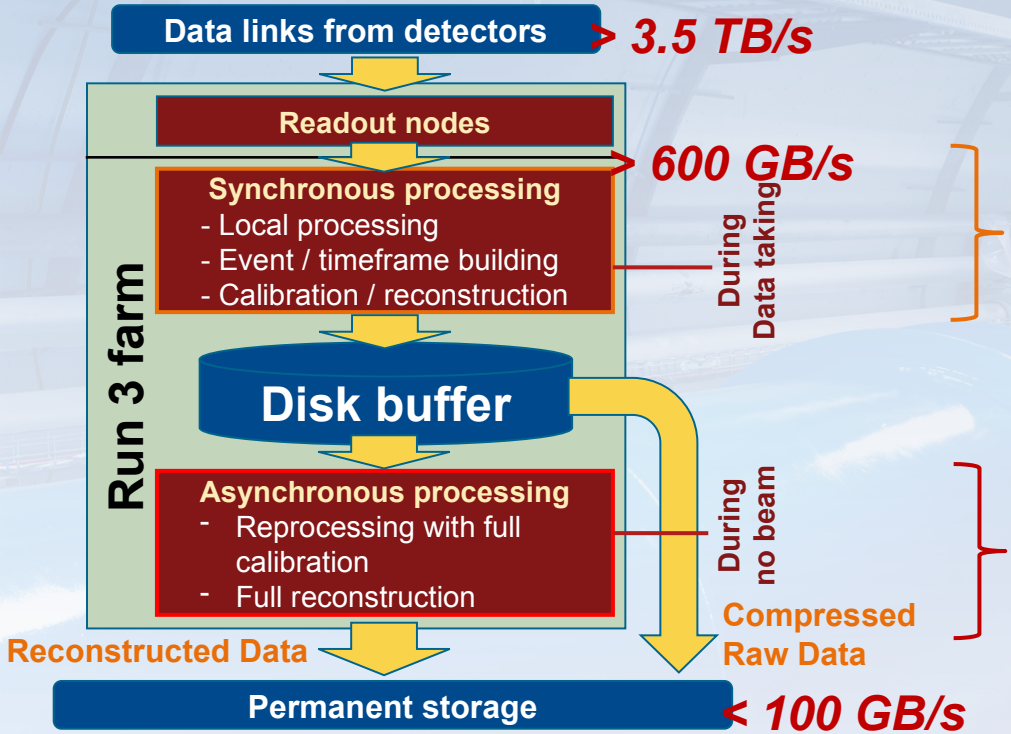




Synchronous and Asynchronous Reconstruction



Synchronous and Asynchronous Reconstruction



- Calibration: Tracking for ITS / TPC / TRD / TOF for ~1% of tracks.
- Data compression: track-model compression requires full TPC tracking for all collisions.
  - → TPC tracking dominant workload during synchronous reconstruction.
  - → Well suited to run on **GPUs**, EPN farm designed for best TPC clusterization / tracking / compression performance.

- No clear single computational hot-spot.
- TPC reconstruction important but not dominant.
  - Actually faster than in the synchronous phase: no clusterization / no compression / less hits after hit removal in synchronous phase overcompensates the slowdown of more elaborate fits.
- Full reconstruction for all other detectors.

→ More heterogeneous workload.

Baseline / optimistic scenario for GPU processing

- **EPNs make massive use of GPUs to speed up the real time TPC processing (bulk of synchronous reconstruction).**
- **Aiming to use the GPUs as well as possible also in the asynchronous reconstruction.**
- **GPU processing developed for 2 scenarios:**

Baseline GPU solution
(fully available, used 2022):
TPC + part of ITS tracking on GPU

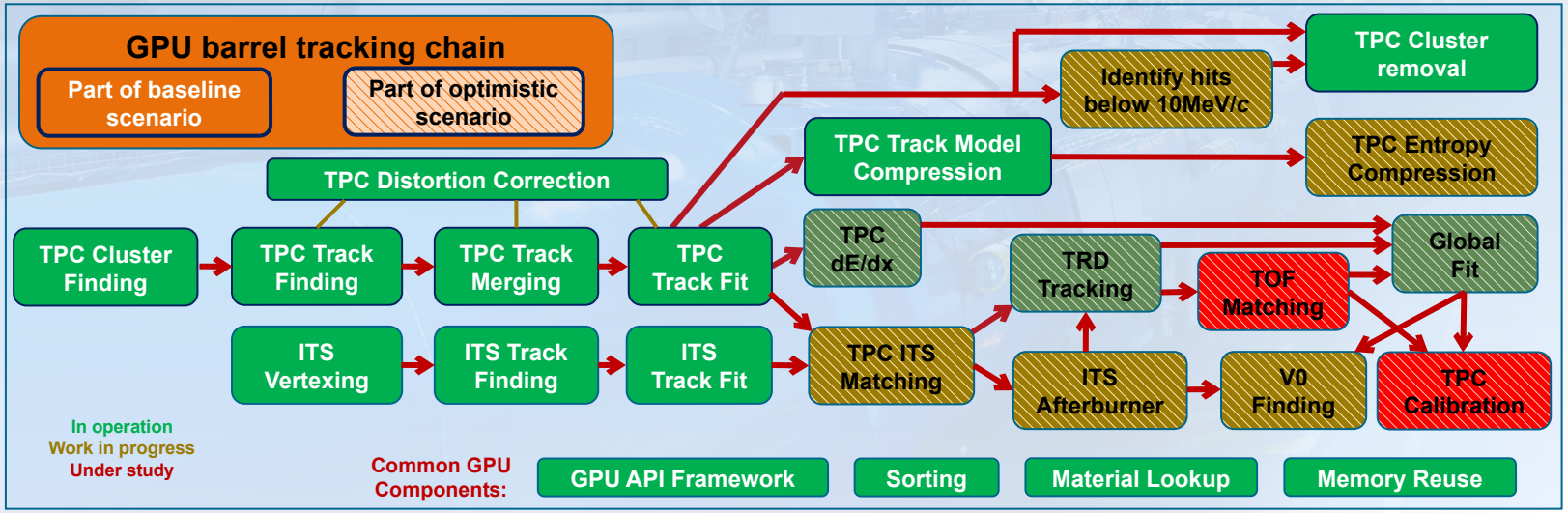
- This is the mandatory part to keep step with the data taking during the synchronous reconstruction.
  - Aiming for ~20% margin.
- Caching the raw data is impossible, i.e. if we are not fast enough here, we need to reduce the interaction rate.

Optimistic GPU solution
(what we are aiming for eventually):
Run full barrel tracking on GPU

- This aims to make the best use of the GPUs also in the asynchronous phase.
- Does not affect the synchronous processing much, though could offload slightly more steps to the GPU as well..
- If we cannot use the GPUs for a large part of the asynchronous reconstruction on the EPN, the processing would be CPU bound while the GPUs would be idle.

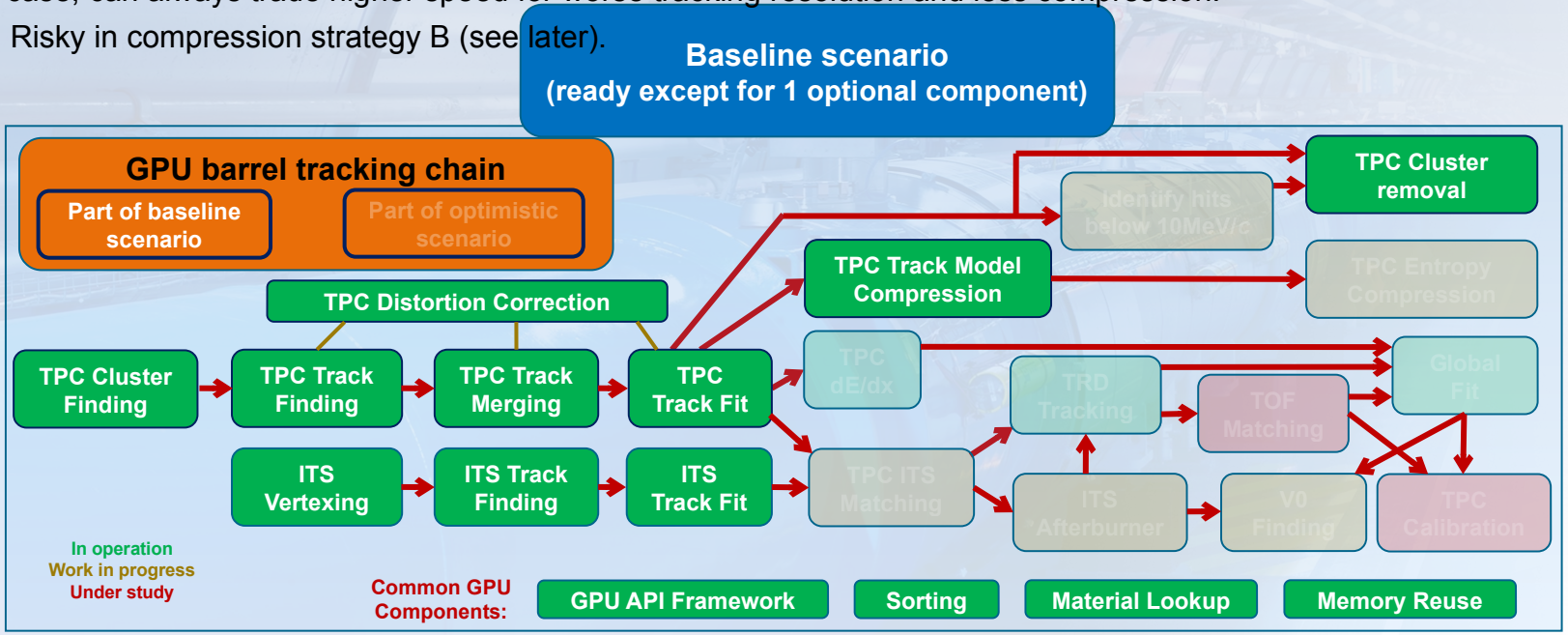
Central Barrel Global Tracking Chain

- Overview of reconstruction steps considered for GPU-offload:
  - Mandatory **baseline scenario** includes everything that must run on the GPU during synchronous reconstruction.
  - **Optimistic scenario** includes everything related to the barrel tracking.



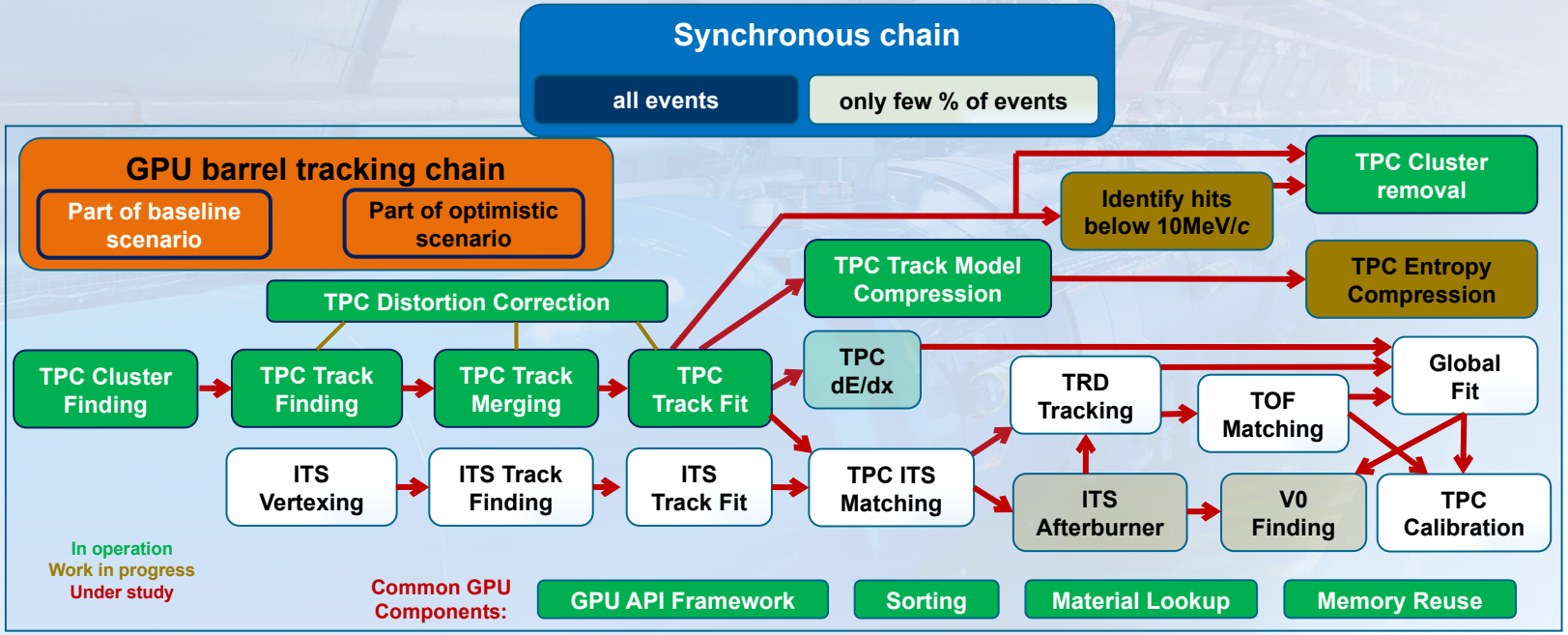
Central Barrel Global Tracking Chain

- Baseline scenario fully implemented (module some improvements e.g. distrotion correction).
  - Not mandatory to speed up the synchronous GPU code further, but we should try nonetheless.
  - If we add / improve reconstruction steps, we have to speed it up accordingly to remain in the 2000 GPU budget.
  - Worst case, can always trade higher speed for worse tracking resolution and less compression.
    - Risky in compression strategy B (see later).



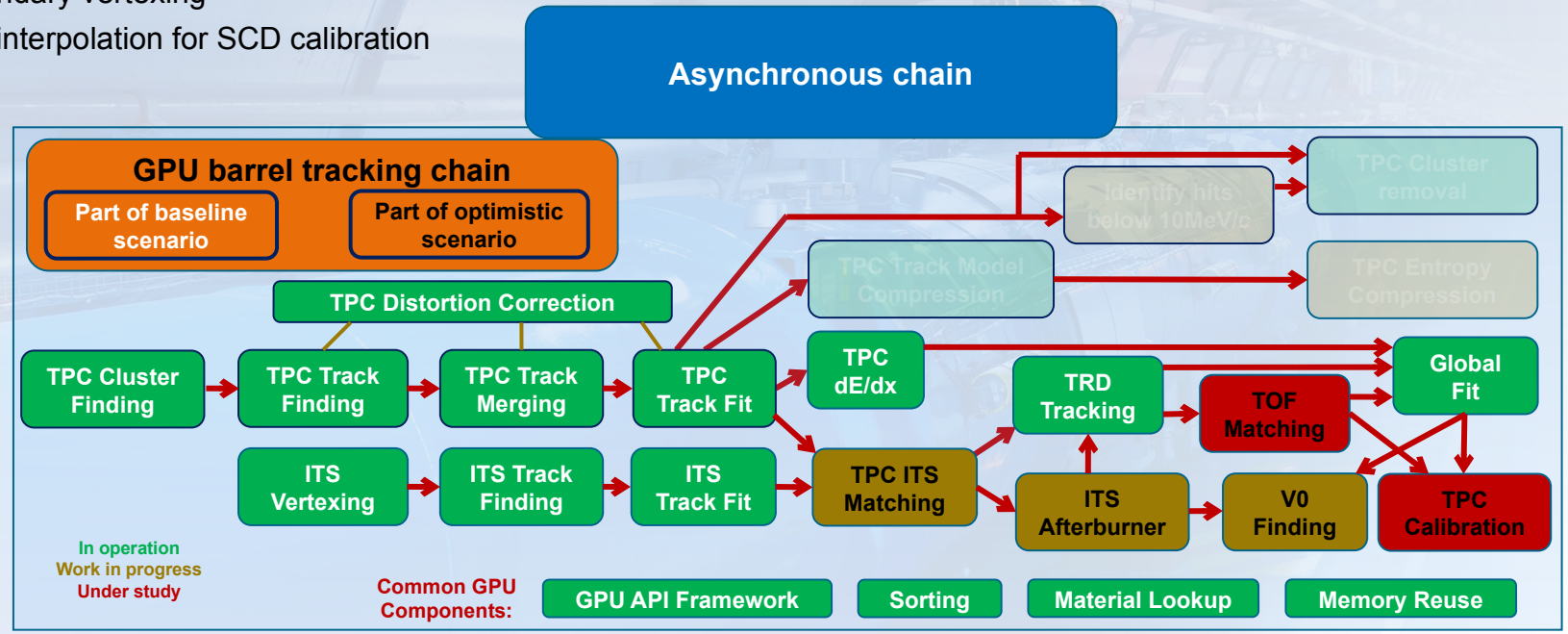
Central Barrel Global Tracking Chain

- Baseline scenario fully implemented (module some improvements e.g. distrotion correction).
  - 2 optional parts still being investigated for sync. reco on GPU: TPC entropy encoding / Looper identification < 10 MeV.



Central Barrel Global Tracking Chain

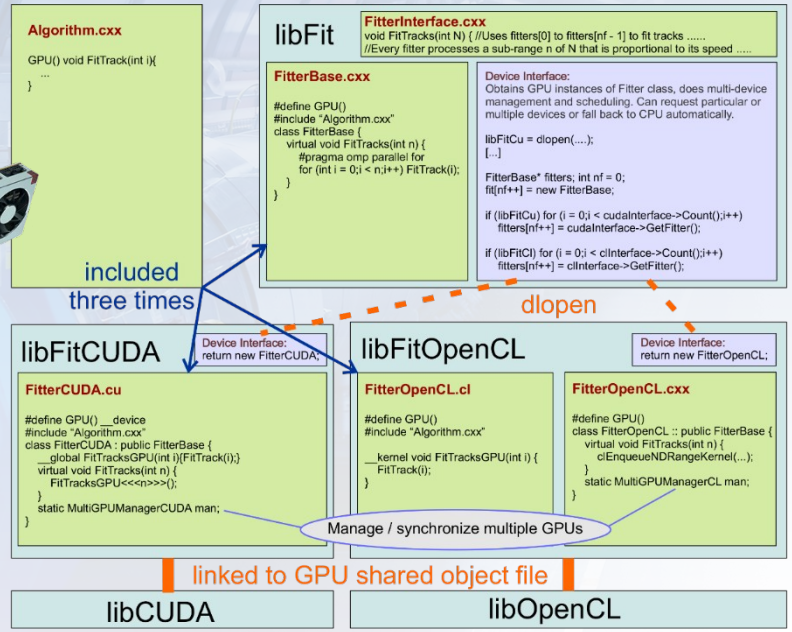
- Several steps missing in asynchronous reconstruction:
  - Matching to ITS
  - Matching to TOF
  - Secondary vertexing
  - TPC interpolation for SCD calibration



Compatibility with different GPU Frameworks

- Generic common C++ Code compatible to CUDA, OpenCL, HIP, and CPU (with pure C++, OpenMP, or OpenCL).
  - OpenCL needs clang compiler (ARM or AMD ROCm) or AMD extensions (TPC track finding only on Run 2 GPUs and CPU for testing)
  - Certain worthwhile algorithms have a vectorized code branch for CPU using the Vc library
  - All GPU code swapped out in dedicated libraries, same software binaries run on GPU-enabled and CPU servers

- Screening different platforms for best price / performance.
  (including some non-competitive platforms for c… …atio…
  - **CPUs** (**AMD Zen**, **Intel Skylake**)
    C++ backend with **OpenMP**, AMD **OCL**
  - **AMD GPUs**
    (**S9000** with **OpenCL 1.2**, **MI50** / **Radeon 7** / **Navi** with **HIP** / **OCL 2.x**)
  - **NVIDIA GPUs**
    (**RTX 2080** / **RTX 2080 Ti** / **Tesla T4** with **CUDA**)
  - **ARM Mali GPU** with **OCL 2.x**
    (Tested on dev-board with Mali G52)



Tuning of asynchronous pp reco on EPNs with GPU

- **Benchmarks of asynchronous reconstruction, pp collisions (real data), 650 kHz inellastic rate.**
- **Looked at 3 scenarios:**
  1. 8-core GRID node, CPU only: Allocated 4 physical cores (8 virtual cores) and 32 GB of RAM (4 per virtual core) on an EPN - closest to grid setup.
     - Note: GRID guarantees only 16 GB, but currently no workflow that can efficiently process TFs with 16 GB of memory.
  2. 1/8th of an EPN, or Single-GPU: 8 physical cores (16 virtual cores), 64 GB of memory, 1 MI50 GPU – **running today on EPNs**.
  3. ½ of an EPN / 1 NUMA domain: 32 physical cores (64 virtual cores), 256 GB of memory, 4 MI50 GPUs – closest to synchronous processing.

Setup	Seconds per TF
8 core, no GPU, unoptimized, 1 TF in flight, 32 GB	84.75
8 core, no GPU, optimized, 3 TF in flight, 64 GB	50.21
16 core + GPU, unoptimized, 1 TF in flight, 64 GB	71.25
16 core + GPU, optimized, 8 TF in flight, 64 GB	11.23
4 * (16 core + GPU, optimized, 8 TF in flight, 64 GB)	3.55 = (14.20 / 4)
64 core + 4 GPU, 24 TF in flight, 256 GB, partially optimized	3.45

CPU only

1/8th EPN

1/2 only

- Slowdown when running 4 independent 1-GPU workflows in one NUMA domain due to competition for resources: 14.20s vs. 11.23s
- Using the 1/2 EPN setup, we currently gain ~3%: 3.45s vs. 3.55s, but not yet fully optimized, and still causes some framework trouble.

Tuning of asynchronous Pb-Pb reco on EPNs

- Pb-Pb processing needs more memory than pp due to larger time frames.
- 1 GPU workflow not efficient, since we would need more memory. The 1NUMA domain workflow has synergy effects and is thus not limited by memory.
- Tuned only 2 cases:
  - the CPU-only workflow without memory constraint (for reference)
  - the 1 NUMA domain workflow.

- Tested on MC Pb-Pb data (since 2022 Pb-Pb real data is too low IR for realistic measurements).

Setup	Seconds per TF
8 core, no GPU, optimized, 3 TF in flight, 96 GB	200
64 core + 4 GPU, 24 TF in flight, 256 GB, partially optimized	25

  - Note: Running 8 * 8-core workflow in one NUMA domain does not speed up the throughput 8-fold, since they compete for resources such as memory.
  - GPU Pb-Pb async workflow currently heavily impacted by still poor performance of some CPU-bound algorithms. Average GPU load is only ~20%. Thus little improvement from GPU usage.

- Note: the 200s is very close to the conservative estimate we have been using for the resource estimates for Pb-Pb processing, so asynchronous processing of 2022 Pb-Pb data at 50 kHz should have worked nicely.

GPU / CPU fraction of workload (Pb-Pb 50 kHz MC)

- The table below shows the relative compute time (linux cpu time) of the processing steps running on the processor.
  - The synchronous reconstruction is fully dominated by the TPC (99%) which already fully runs on the GPU, some more processes might follow.
  - Basically no margin to offload mory synchronous reconstruction step to the GPU – or if we did, it wouldn't change anything.

Synchronous processing

Processing step	% of time
TPC Processing	99.37 %
EMCAL Processing	0.20 %
ITS Processing	0.10 %
TPC Entropy Coder	0.10 %
ITS-TPC Matching	0.09 %
MFT Processing	0.02 %
TOF Processing	0.01 %
TOF Global Matching	0.01 %
PHOS / CPV Entropy Coder	0.01 %
ITS Entropy Coder	0.01 %
FIT Entropy Coder	0.01 %
TOF Entropy Coder	0.01 %
MFT Entropy Coder	0.01 %
TPC Calibration residual extraction	0.01 %
TOF Processing	0.01 %

Running on GPU in baseline scenario **Running on GPU in optimistic scenario**

GPU / CPU fraction of workload (650 kHz pp)

- Same table for asynchronous reconstruction.
- Compute time much more wide-spread: TPC is only ~60%.
- Imperative to offload more steps onto the GPU for good EPN usage

Asynchronous processing

Process	Compute Time [%]
TPC Tracking	61.64
ITS TPC Matching	6.13
MCH Clusterization	6.13
TPC Entropy Decoder	4.65
ITS Tracking	4.16
TOF Matching	4.12
TRD Tracking	3.95
MCH Tracking	2.02
AOD Production	0.88
QC	4.00
Rest	2.32

- Currently (**baseline scenario**) **61.6%** of the asynchronous workflow on the GPU.
- When the GPU offloading effort for TPC + ITS + TRD + TOF is finished (**optimistic scenario**), we will have **85%** on the GPU in this setup.
  - Already on GPU
  - To be offloaded
- For reference: **90%** of EPN compute power in the GPU (assuming GPU speedup for other detectors is similar as for TPC).

Summary / Requirements to run on GPUs

- ALICE O2 code is agnostic wrt. GPU vendors, though we don't test all models on a regular basis.
- In principle, we should be able to run on all GPUs that support AMD ROCm, NVIDIA CUDA, OpenCL 3.0 with C++ extensions (basically only ARM), and in the future we might have a look at Intel One API, but not yet.
- Prcatically, that means AMD MI*** and NVIDIA.

- The code is mostly optimized for the MI50 GPU, but we have tested that MI100 and MI210 run with acceptable performance, and most NVIDIA GPUs (A100, V100, RTX2080, RTX3090) run with good performance.
- GPU optimization is mostly for the sync reco, but we are cathing up with async.

- Requirement to run: For processing 50 kHz Pb-Pb data (nominal Pb-Pb rate, this is what we should aim for):
  - GPU with 32 GB
  - 64 GB of host memory per GPU
- For lower interaction rate, e.g. 500 kHz pp
  - GPU with 8 GB (although not ideal)
  - 40 GB of host memory per GPU

- Note that our current builds on CVMFS support only AMD MI50 and NVIDIA RTX3090. If we want to support different architectures, we either have to create additional builds for them, or update our current builds such that they can support multiple GPU architectures per backend.

Summary / Requirements to run on GPUs

- From the performance perspective,:
  - at the moment (60% of workload on the GPU), 1 GPU per 8 physicsl CPU cores is ideal.
  - When we have finalized the optimistic scenario, 3 or 4 GPUs per 8 physical CPU cores should become ideal.
- Our GPU code needs RHEL8 or compatible (or current Ubuntu, whatsoever, but not RHEL7!)
- On the EPN farm, we use the EL8 (rocky 8.x) singularity container from CVMFS
- Once GPUs are available, the same software from CVMFS can run on the GPUs, or on the CPUs only.
  - Controlled only via variable from the JDL.
  - If the GPU is different from the EPN model, the workflow might require some tuning, since the relation between GPU performance and CPU performance will be different.
- On the EPNs, we use the slurm scheduler and the standard ROCR_VISIBLE_DEVICES and CUDA_VISIBLE_DEVICES to steer which GPU to use. But in principle any scheduler that exports these variables should work (in case the server has more than 1 GPU).
- Finally, the “usual” stuff needs to be set up for the job submission, i.e. the ALICE user must be in video group, GPU must be available from singularity (with –rocm flag for us), etc.