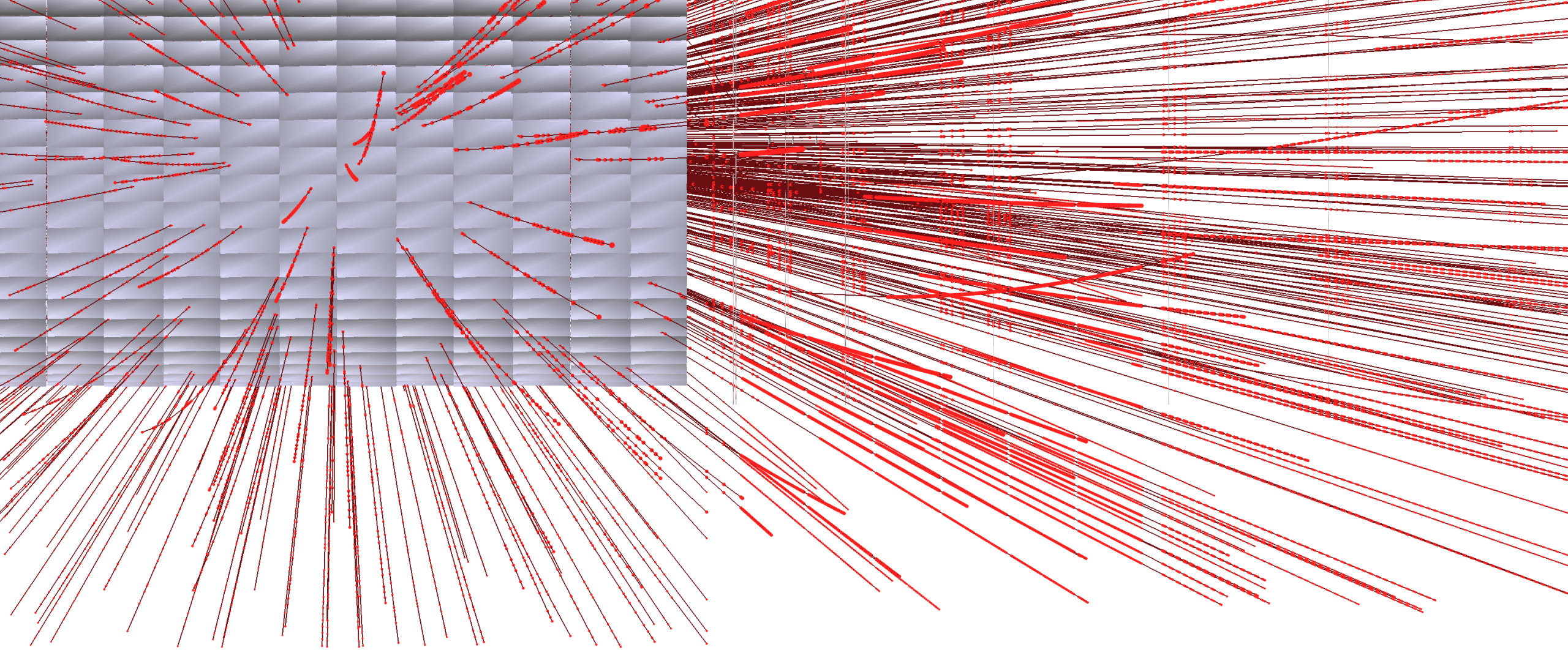
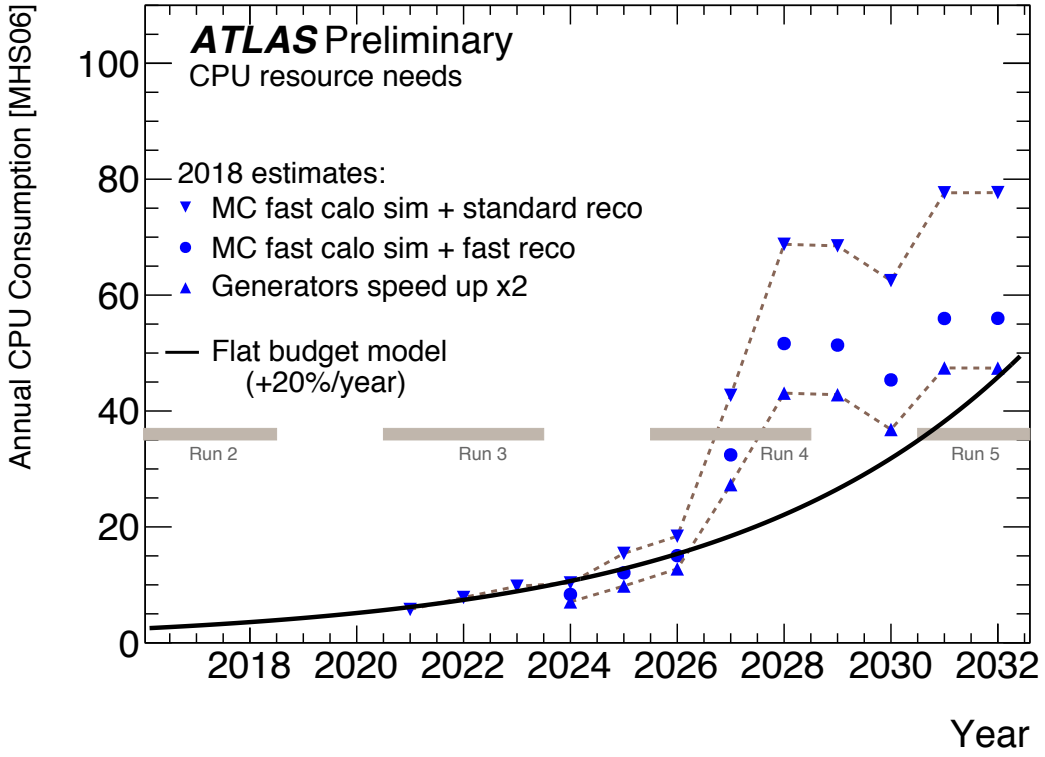


Acts in multi-threaded environment

Paul Gessinger - 01/15/2019 - Tracking workshop for HEP - LBNL

What's the problem, where are we?

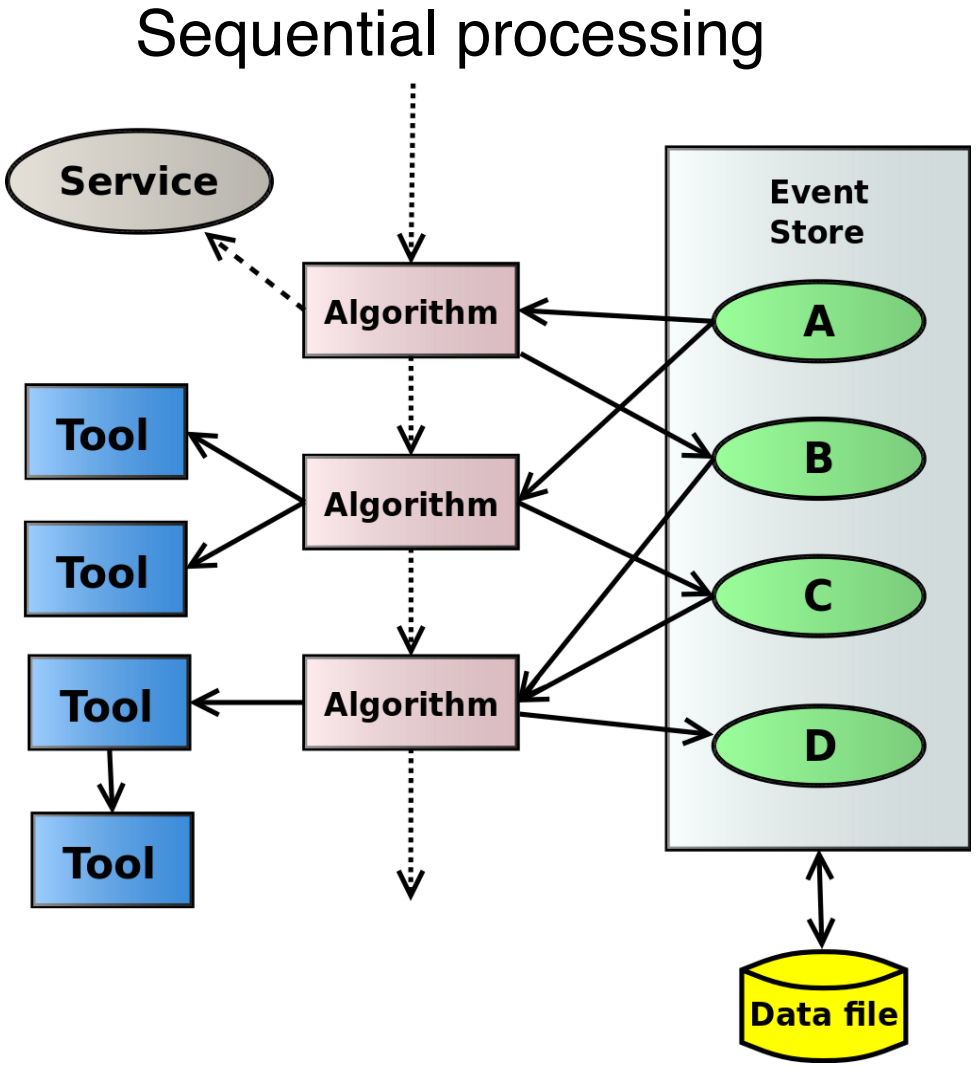
- Track reconstruction is (often) the most CPU-intensive part of event reconstruction (ATLAS: ≈80%)
- CPU resources are limited!
- Scaling with pile-up is not encouraging
- Parallelization can help!
- But: just executing on more CPU(-cores might be problematic: $\frac{memory}{\#CPUs}$ will decrease
- Other approaches can help to saturate more CPUs on same amount of memory



Parallelization in ATLAS

ATLAS reconstruction

- ATLAS reconstruction comprises multiple domains, (e.g. ID, muons, calorimeters, jets) with lots of interdependencies
- Time spent by domain:
  - ID: (Pixel, SCT, TRT) ≈ 11s/event
  - Everything else: ≈ 15×1.5s (15 = no. of domains)
- ID reconstruction is **clearly** the place to optimize!
- ATLAS software is based on Gaudi
  - *Algorithms* process events, data flows through input and output collections
  - Can use *Tools* to offload some of their work
  - *Services* (singletons) can be accessed from both



[1]

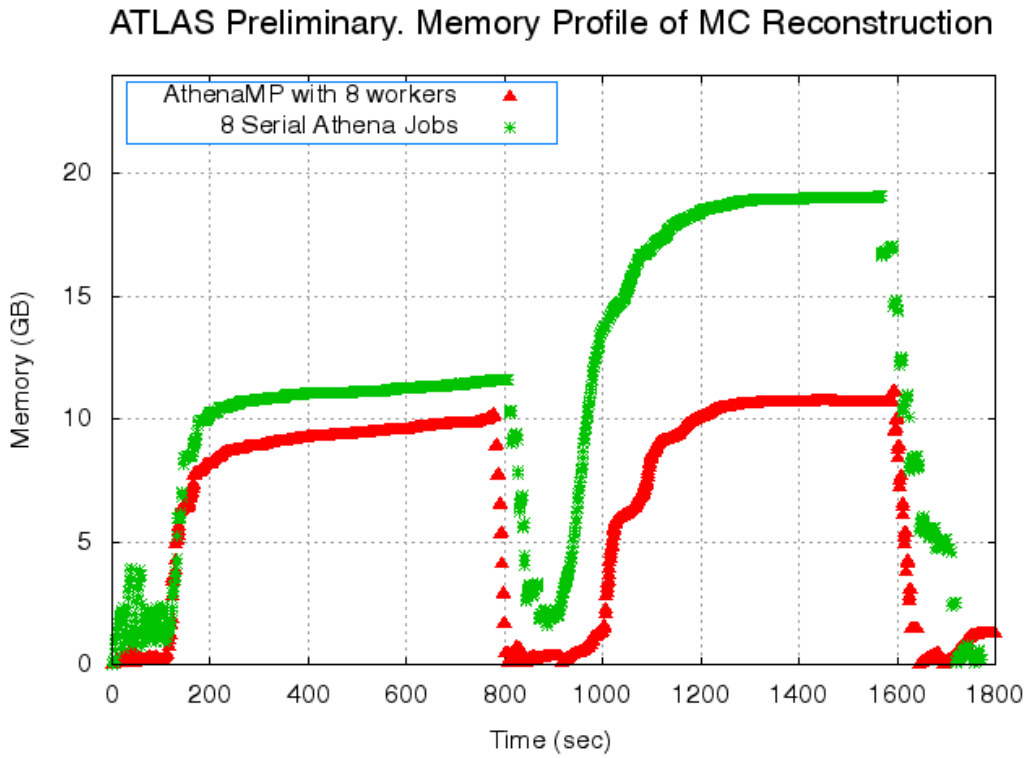
[1]: Scott Snyder

Run separate jobs

- Simplest idea: just run multiple instances of the whole software
- Almost “trivial” to implement
- But:
  - Can only parallelize on event ranges
  - Duplicates the **entire** software stack in memory!

Multi processing

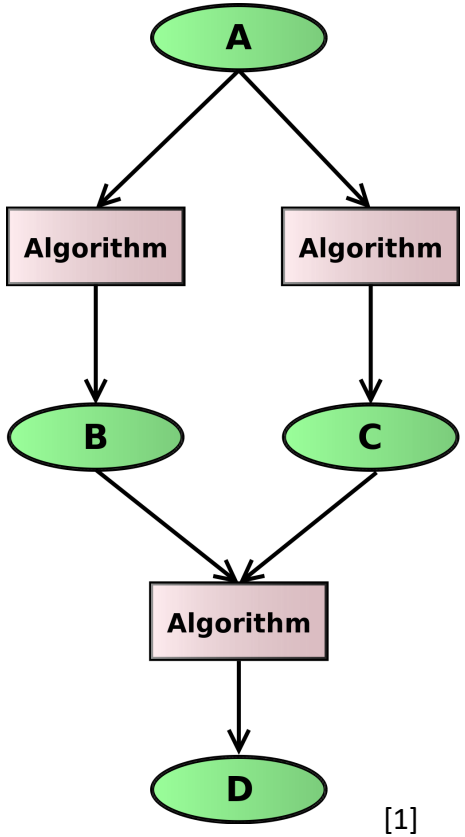
- Run 2: parallelization with multiple processes, but forked
- Requires little change to actual code
- Start processing first event, then fork the process to all CPUs
- Copy-on-write allows easy sharing of memory
- Memory savings might not be enough for Run 3 and beyond



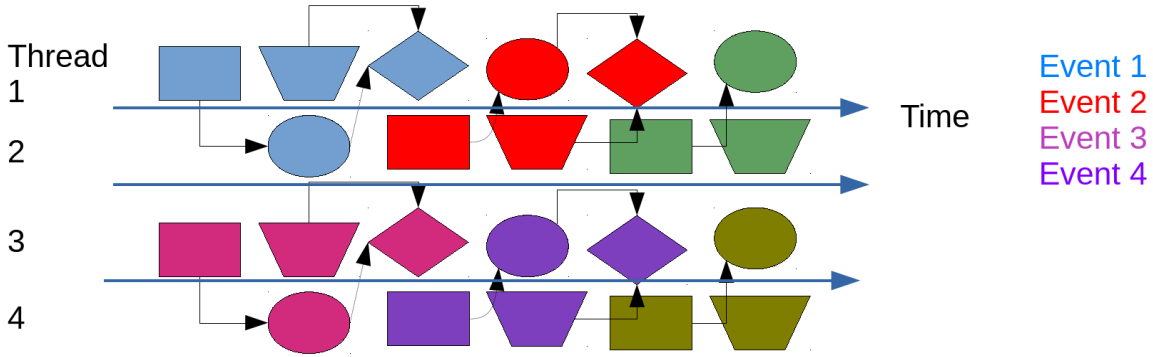
[1]: Scott Snyder

Multi-threading

- AthenaMT event-processing framework (in development)
- Allow parallelization at algorithm level (scheduler figures out data dependencies)
  - **Inter- and intra-event** parallelism possible
- Ideally: algorithms only instantiated once, invoked for every event
  - Keeps memory footprint low
  - All tools need to be thread-safe
  - Most importantly: no mutable state
- **However:** most of ID chain is sequential



[1]



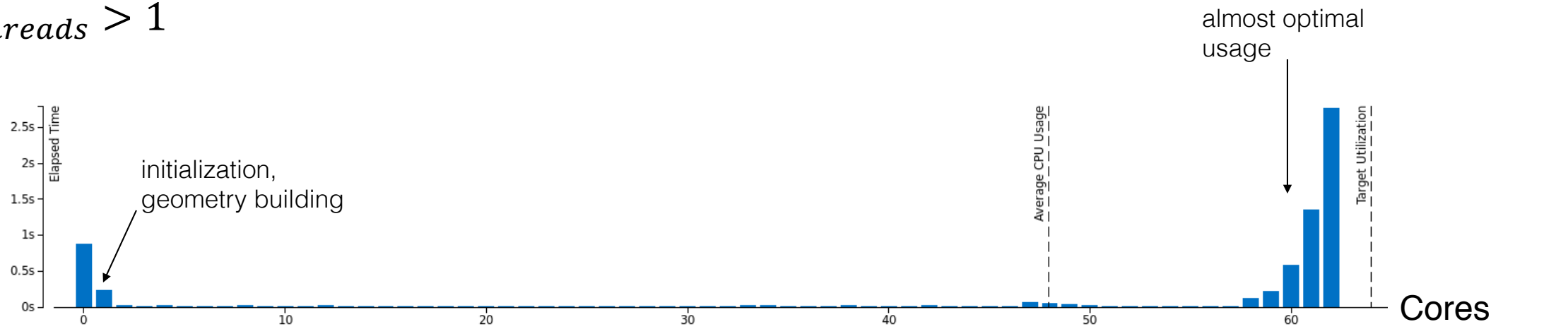
[1]: Scott Snyder

Enter: Acts

Toolkit containing thread-safe algorithms and utilities

Acts and Multi-Threading

- Acts is designed to be thread-safe:
  - Local configuration and state
  - Immutable data everywhere (e. g. geometry)
- Acts does **not** provide infrastructure for parallelization
- Parallelization is implemented by experiment software
- The test framework[1] contains a TBB[2] based event-by-event parallel loop
  - CI job[3] tests results from demo particle extrapolation is identical, for $n_{threads} = 1$ and $n_{threads} > 1$



[1]: https://gitlab.cern.ch/acts/acts-framework, [2]: https://www.threadingbuildingblocks.org [3]: https://gitlab.cern.ch/acts/acts-framework/-/jobs/3043263

Design choices for multi-threading

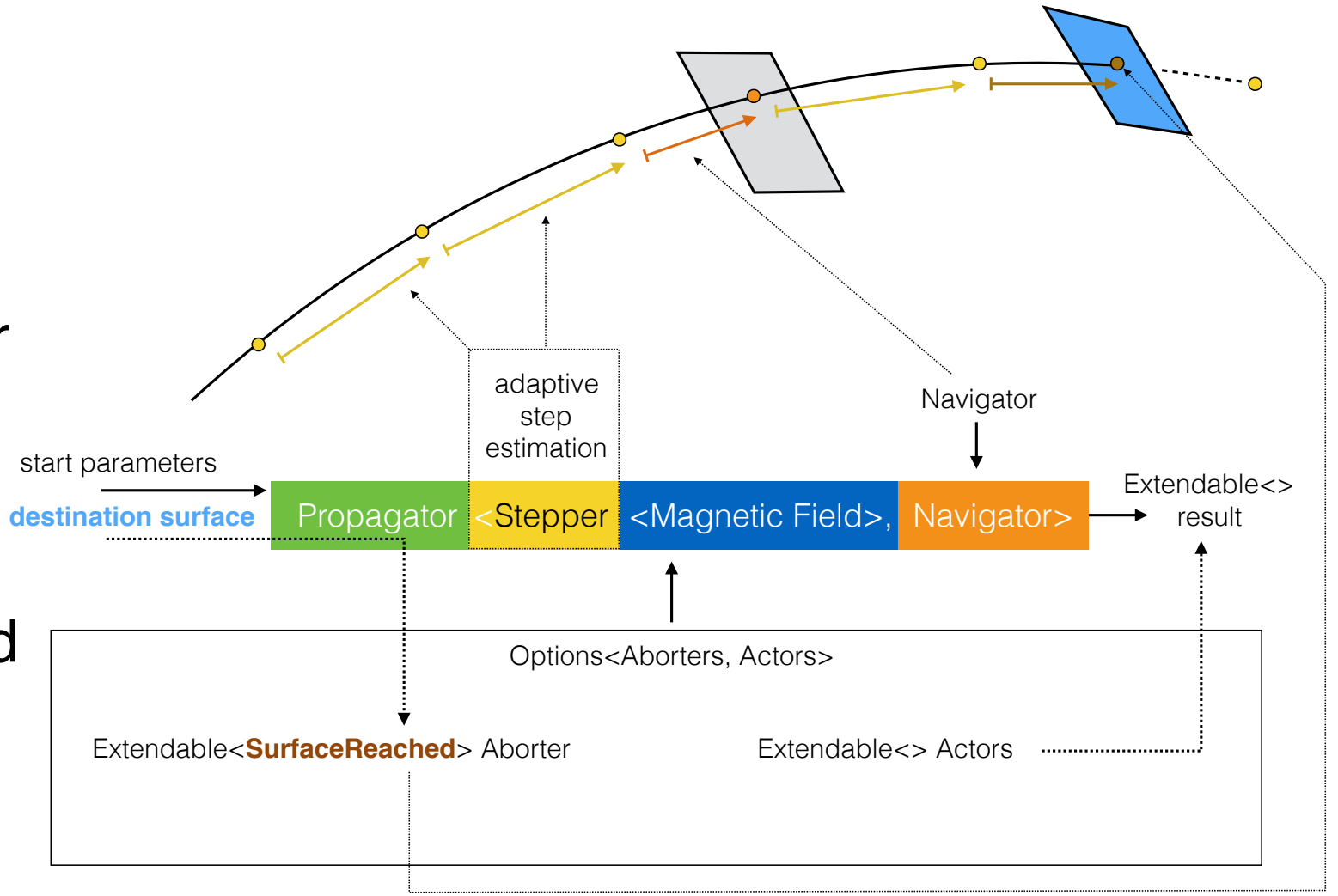
- Mutable state is limited to **thread-local** storage, passed around explicitly

```
OptionsType eOptions;
// setup options ...
StateType state(start, eOptions);


// Perform the actual propagation & check its outcome
if (propagate_impl(result, state) != Status::IN_PROGRESS) {
  result.status = Status::FAILURE;
} else {
  // Convert into return type and fill the result object
  m_stepper.convert(state.stepping, result);
  result.status = Status::SUCCESS;
}
return result;
```

- Geometry is considered **immutable** after creation (technically, we const-cast it currently during closure, but we want to get rid of that)

Implementation Propagator and KalmanFilter

- Acts propagation is based on existing ATLAS implementation
- Redesigned interface, eliminated all mutable state
- Switched to Eigen library for math (much more readable)
- **Designed to be flexible and extensible!**
- KalmanFilter is implemented on top of the Propagator



Implementation Propagator and KalmanFilter

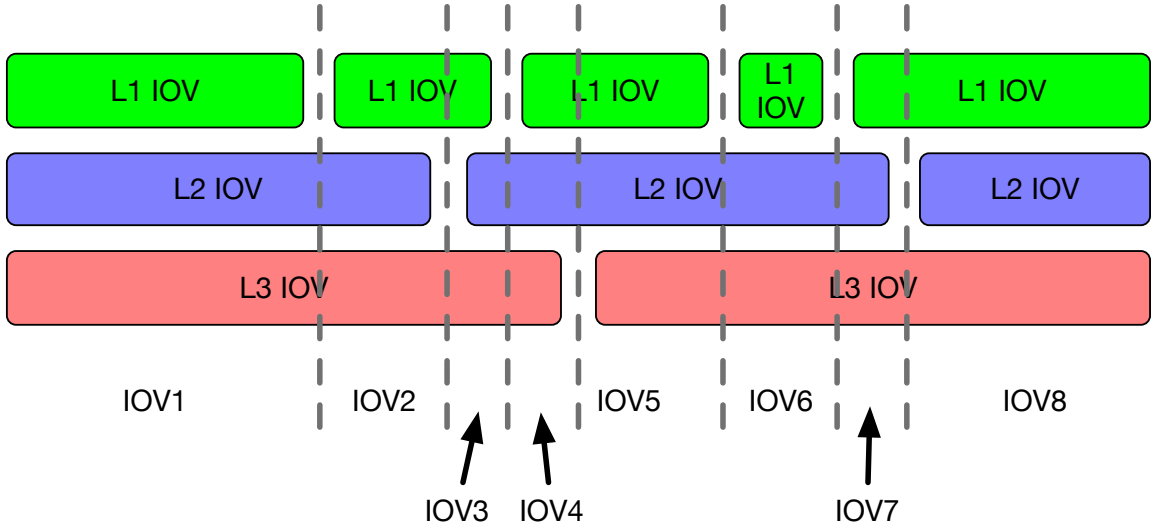
- `KalmanFilter` contains `Propagator` object
- Call to `fit()` sets up propagation and attaches the `KalmanActor`
- `KalmanActor` implements logic:
  - Initialize
  - Forward filter
  - Backward smoothing
- Delegates to separate *calibrator*, *updator* and *smoother*
- Can be set up once, then invoked from many threads

```
template <typename input_measurements_t,
          typename parameters_t,
          typename surface_t>
auto
fit(input_measurements_t measurements,
    const parameters_t&  sParameters,
    const surface_t*     rSurface = nullptr) const
{
  // Bring the measurements into Acts style
  auto trackStates = m_inputConverter(measurements);

  // Create the ActionList and AbortList
  using KalmanActor  = Actor<decltype(trackStates)>;
  using KalmanResult = typename KalmanActor::result_type;
  using Actors       = ActionList<KalmanActor>;
  using Aborters     = AbortList<>;

  // Create relevant options for the propagation options
  PropagatorOptions<Actors, Aborters> kalmanOptions;
  // Catch the actor and set the measurements
  auto& kalmanActor = kalmanOptions.actionList.template get<KalmanActor>();
  kalmanActor.trackStates   = std::move(trackStates);
  kalmanActor.targetSurface = rSurface;

  // Run the fitter
  const auto& result
      = m_propagator.template propagate(sParameters, kalmanOptions);

  /// Get the result of the fit
  auto kalmanResult = result.template get<KalmanResult>();

  // Return the converted Track
  return m_outputConverter(std::move(kalmanResult));
}
```

Challenges and problems

Conditions and concurrency

- Conditions: parameters recorded during data taking
- Vary between events (Interval of Validity, IOV)
  - Time-dependent detector properties (wire-sagging, bending, temperatures, …)
  - E.g. **alignment** of sensitive surfaces can change over time (fitted empirically to account for it)
- **Problem**: we consider the geometry constant!
- One solution would be: re-create the geometry at IOV boundaries

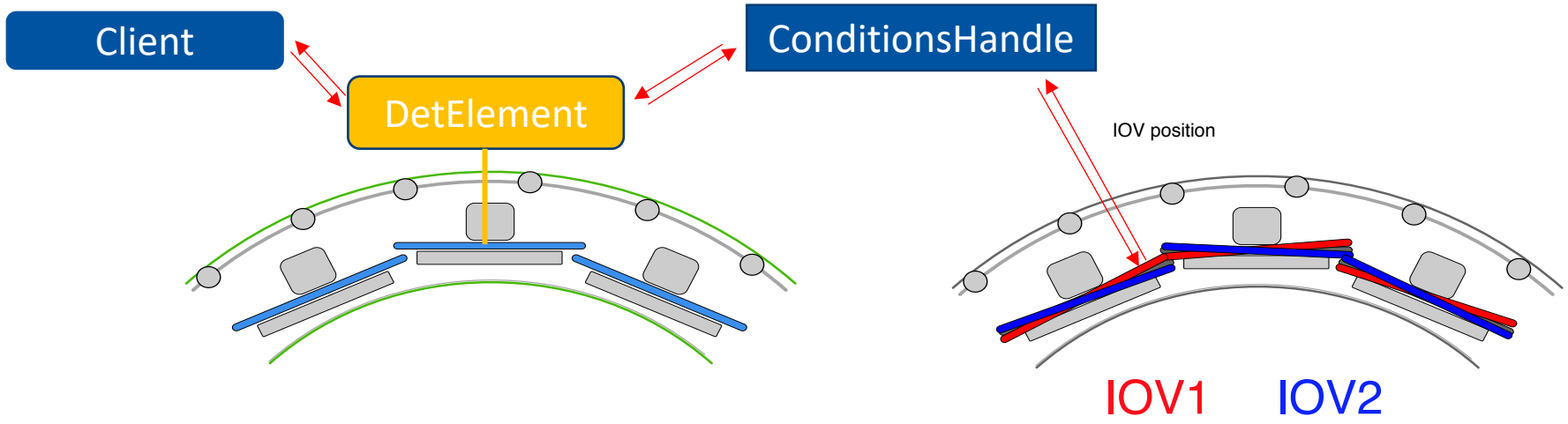


Conditions and concurrency

- Previously: sequential event processing
  - Conditions are accessed through services
  - When event processing reaches an IOV boundary, all services are notified
  - Services update their conditions caches
  - Event processing continues
- **Problem**: this does not work with multiple threads: multiple IOVs can be in flight at the same time
  - n_{IOV} might be small for unfiltered data, but could be large for highly selected samples
- Solution in ATLAS: remove conditions services, enter: **Conditions algorithms**
  - Conditions algorithm makes conditions data available
  - Scheduler makes sure they are scheduled before conditions are required.
- **Problem**: we would potentially re-build the geometry quite a lot!

Conditions: alignment

- What information do we need for the alignment? Module positions (transforms)
- Tracking geometry consists of surfaces, are connected to a (experiment specific) detector element
- Transform comes from that detector element
- Detector elements **need to be aware** of alignment / IOV!



Conditions: alignment

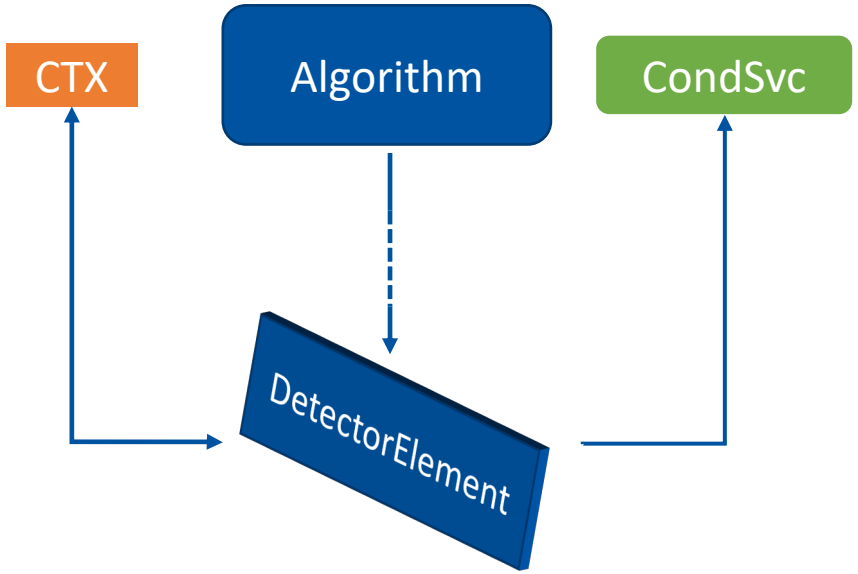
- However: multiple alignments can be in flight at the same time!
  - Detector elements needs to know from which event they are accessed
- ATLAS: Conditions service provides conditions objects for an **event context**
  - Algorithms are explicitly passed the event context:

```
StatusCode
ActsExtrapolationAlg::execute_r(const EventContext& ctx)
const {
  // ...
}
```

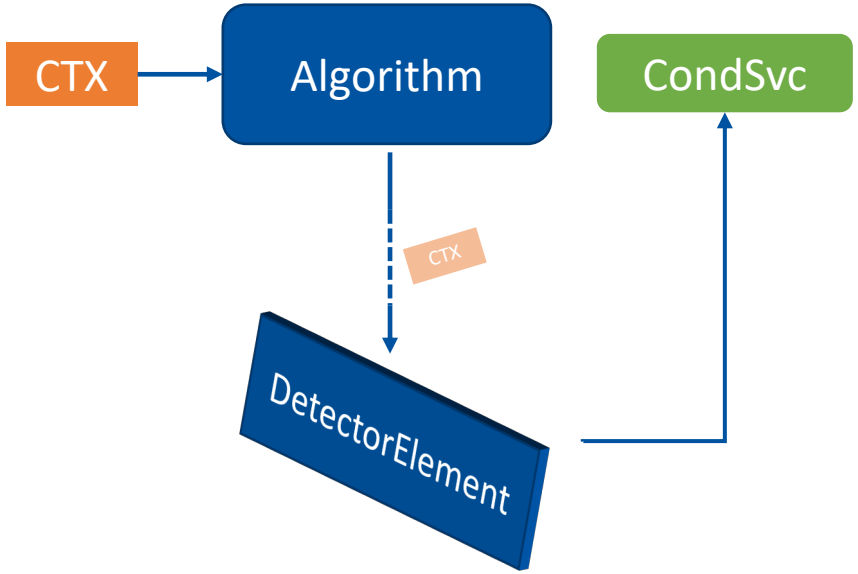
- **Problem**: call chain from algorithm down to individual surfaces is **very** deep and has many paths.

Conditions: alignment

- The **magic** solution:
  - Event context can be accessed from *thread-local static variable* (set up by the scheduler)
  - Detector element can implicitly figure out which IOV it is being accessed from!
  - (this is what was implemented so far)

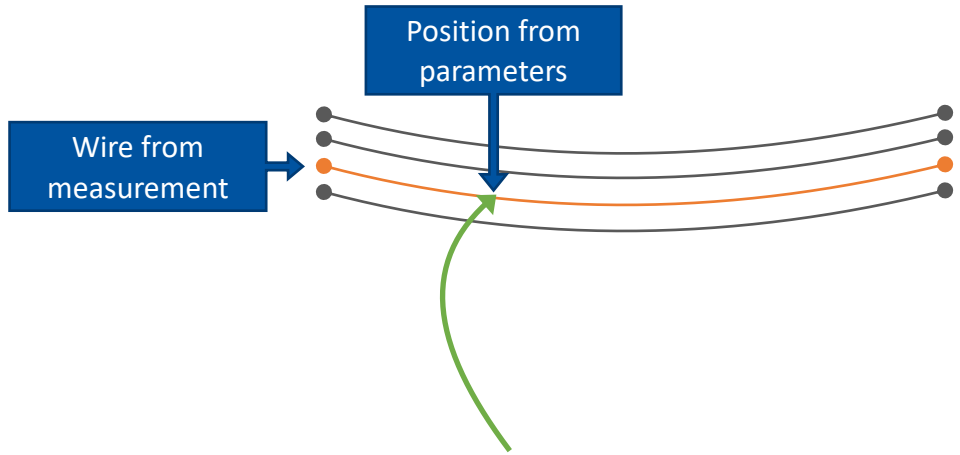


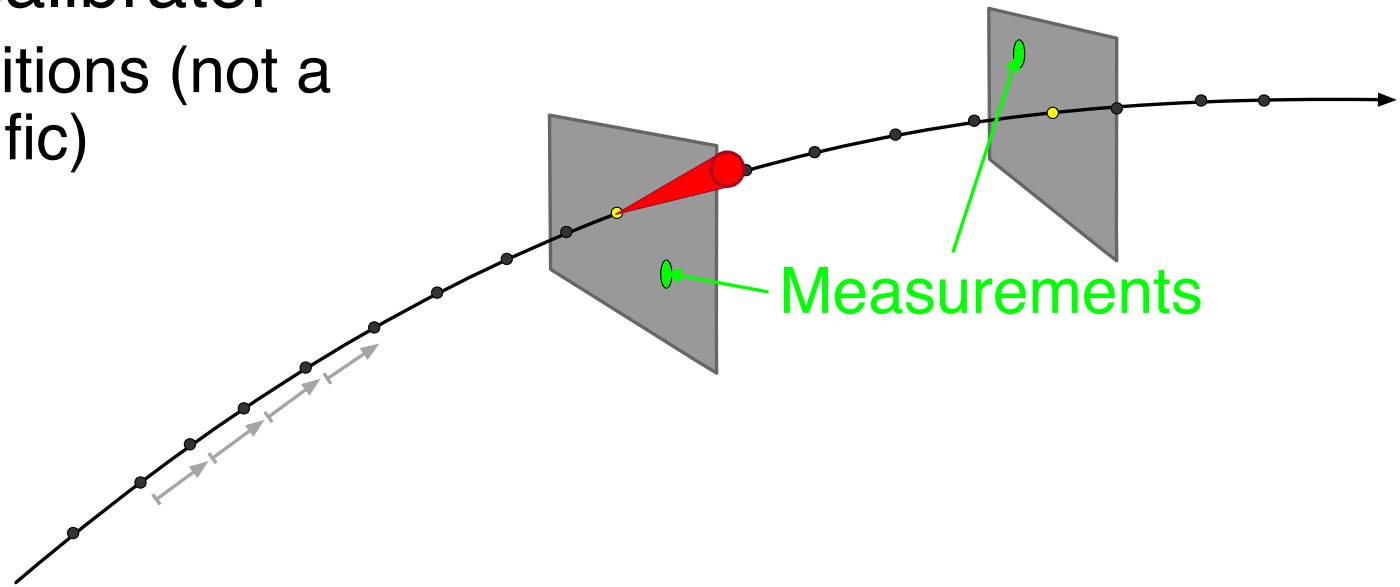
- The **clean** solution:
  - **Every** call chain down to sensitive surfaces needs to pass along the context
  - Type-erase context object so it can be experiment-agnostic
  - (we will most likely switch to this solution)



Conditions for calibration

- Calibration is performed during fitting
  - Idea: raw measurement supplemented with full track parameter prediction
  - Typical example: wire-sagging: wire identified by measurement, sagging calculated at predicted position
- The *updator* delegates to the calibrator
  - Calibrator needs access to conditions (not a problem, since experiment specific)

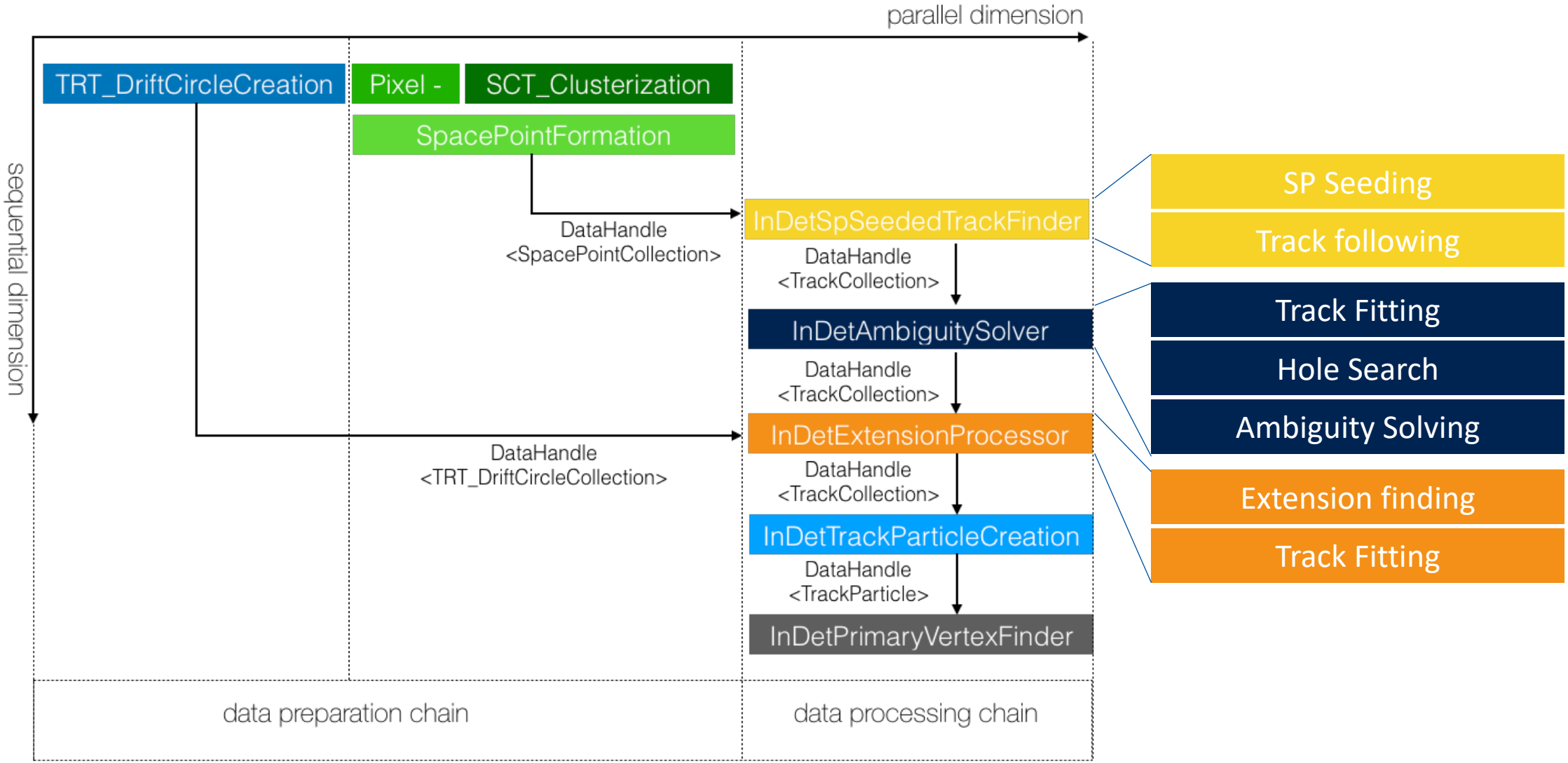




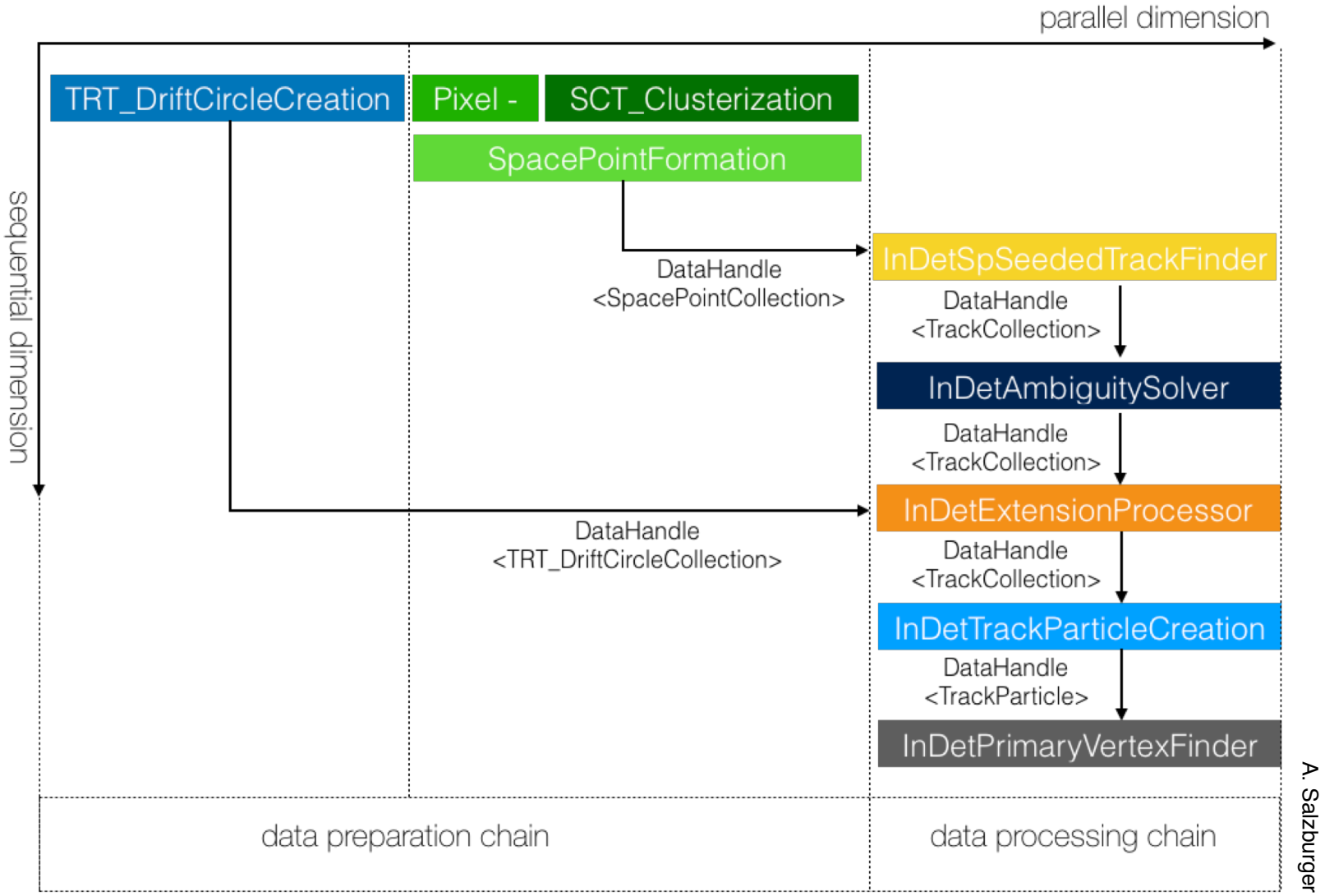
Opportunities

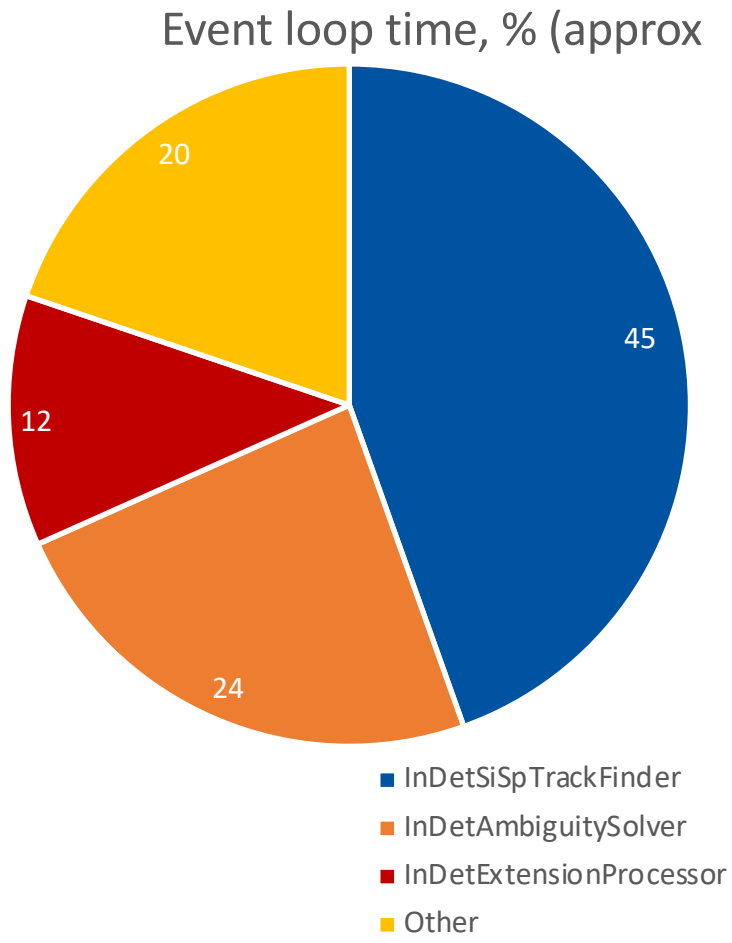
From ATLAS' perspective

Algorithms in ID reconstruction



Time spent in ATLAS reconstruction





Algorithm	Concurrent Cell	Number of cells	Tests	Rel. CPU of ID	Comments
Cluster creation	per module	O(1000)	**no tests exist**	O(5%)	output merging step necessary
SpacePoint creation	per space point	O(10000)	**no tests exist**	O(<1%)	output merging step necessary
SpacePoint seeded track finding	detector region	O(10-100)	GPU based test version from 2011	O(50%)	**overlaps are dangerous**
Ambiguity Solving	fitting: per track ambiguity: per tracks through shared modules	O(100-1000)	**no tests exist**	O(20%)	**book keeping of hits is shared**
TRT extension	per track	O(100)	**no tests exist**	O(10%)	may need update for DataHandle

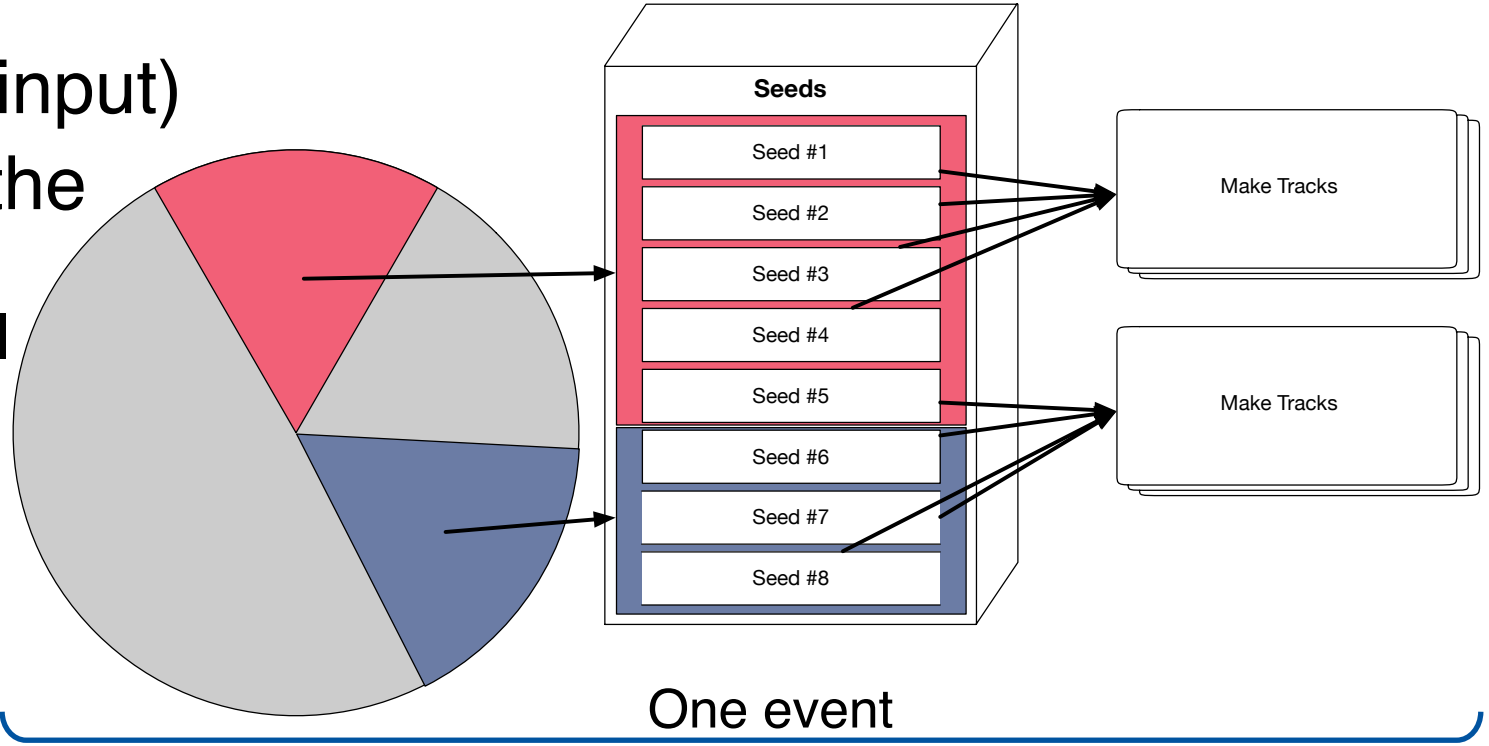
A. Salzburger

Parallelization of ID reco chain

- The largest parts of the ID reco chain are sequential
  - *InDetSpSeededTrackFinder* needs to run before Ambiguity solving, and so on
- Inputs (drift circle creation, clusterization) can be parallelized
  - Need to be careful not to have runaway preemptive pre-processing and starve main algorithm
- If the large algorithms can be made thread-safe: event-by-event parallelism possible
  - If CPUs can be saturate like this: **fine, we're done!**
  - If not? **Finer granularity!**
- In many cases: parallelism by geometric regions requires careful treatment of boundaries (ambiguity solving)

Parallelization of ID reco chain

- E.g. produce seeds parallelized, batch and spawn tasks to do track following
- Acts seed finder implementation can parallelize on middle space point bins!
- Feed into track finding (which runs sequential per input)
- Tune chunk sizes to strike the right balance
  - No effort so far beyond initial tests
- Should allow scaling parallelism as necessary



Summary

- Multi-threaded infrastructure is well underway in ATLAS
- Acts provides components that can be deployed in concurrent environments
- Details of unit-of-parallelization is left to the experiment to decide