acts Propagator module

supported by

EP-R&D Programme on Technologies for Future Experiments

iris hep

Bundesministerium für Bildung und Forschung

cooperations

HSF HEP Software Foundation

intel

CERN openlab

A. Salzburger (CERN) for the ACTS project

@SaltyBurger

An example of code rationalisation

Extrapolator

~ 360 lines interface in IExtrapolator.h
~ 4700 lines of code in Extrapolator.cxx

Propagator

document the code!

```
/// @brief Propagate track parameters - User method
///
/// This function performs the propagation of the track parameters according
/// to the internal implementation object until at least one abort condition
/// is fulfilled, the destination surface is hit or the maximum number of
/// steps/path length as given in the propagation options is reached.
///
/// @tparam parameters_t Type of initial track parameters to propagate
/// @tparam surface_t Type of target surface
/// @tparam action_list_t Type list of actions
/// @tparam aborter_list_t Type list of abort conditions
/// @tparam propagator_options_t Type of the propagator options
///
/// @param [in] start Initial track parameters to propagate
/// @param [in] target Target surface of to propagate to
/// @param [in] options Propagation options
///
/// @return Propagation result containing the propagation status, final
///         track parameters, and output of actions (if they produce any)
template <typename parameters_t, typename propagator_options_t,
          typename target_aborter_t = SurfaceReached,
          typename path_aborter_t = PathLimitReached>
Result<action_list_t_result_t<
    BoundTrackParameters, typename propagator_options_t::action_list_type>>
propagate(const parameters_t& start, const Surface& target,
          const propagator_options_t& options) const;
```

~ 2 public interface methods, same functionality

Code modernisation

```
// projection of direction onto normal vector of reference frame
double PC = pVector[4] * C[0] + pVector[5] * C[1] + pVector[6] *
double Bn = 1. / PC;

double Bx2 = -A[2] * pVector[29];
double Bx3 = A[1] * pVector[38] - A[2] * pVector[37];

double By2 = A[2] * pVector[28];
double By3 = A[2] * pVector[36] - A[0] * pVector[38];

double Bz2 = A[0] * pVector[29] - A[1] * pVector[28];
double Bz3 = A[0] * pVector[37] - A[1] * pVector[36];

double B2 = B[0] * Bx2 + B[1] * By2 + B[2] * Bz2;
double B3 = B[0] * Bx3 + B[1] * By3 + B[2] * Bz3;
```

AtlasStepper, transcript in ACTS project

```
void covarianceTransport(Covariance& covarianceMatrix, Jacobian& jacobian,
                         FreeMatrix& transportJacobian, FreeVector& derivatives,
                         BoundToFreeMatrix& jacobianLocalToGlobal,
                         const Vector3D& direction) {
  // Build the full jacobian
  jacobianLocalToGlobal = transportJacobian * jacobianLocalToGlobal;
  const FreeToBoundMatrix jacToLocal =
      surfaceDerivative(direction, jacobianLocalToGlobal, derivatives);
  const Jacobian jacFull = jacToLocal * jacobianLocalToGlobal;

  // Apply the actual covariance transport
  covarianceMatrix = jacFull * covarianceMatrix * jacFull.transpose();

  // Reinitialize jacobian components
  reinitializeJacobians(transportJacobian, derivatives, jacobianLocalToGlobal,
                        direction);

  // Store The global and bound jacobian (duplication for the moment)
  jacobian = jacFull;
}
```

EigenStepper, covariance transport

Propagator infrastructure

```
Propagator<Navigator, Stepper>
```

Mathematical module that ensures transport of parameters & covariances through magnetic field (if present)

Module that provides next candidate surfaces:

- Detector surfaces
- Layer approach surfaces
- Boundary surfaces

new layer initiates new detector surface candidates

new boundary surface (= new volume) initiate new layer candidates

Proposal:

Drop the layer concept and simplify to surfaces and volumes only, see tomorrow & detray talks.

Propagation flow

Stepping loop

(until max number of steps)

`Navigator::status:`

On a current surface?

Walk through actor list

Walk through aborter list

`Navigator::target`

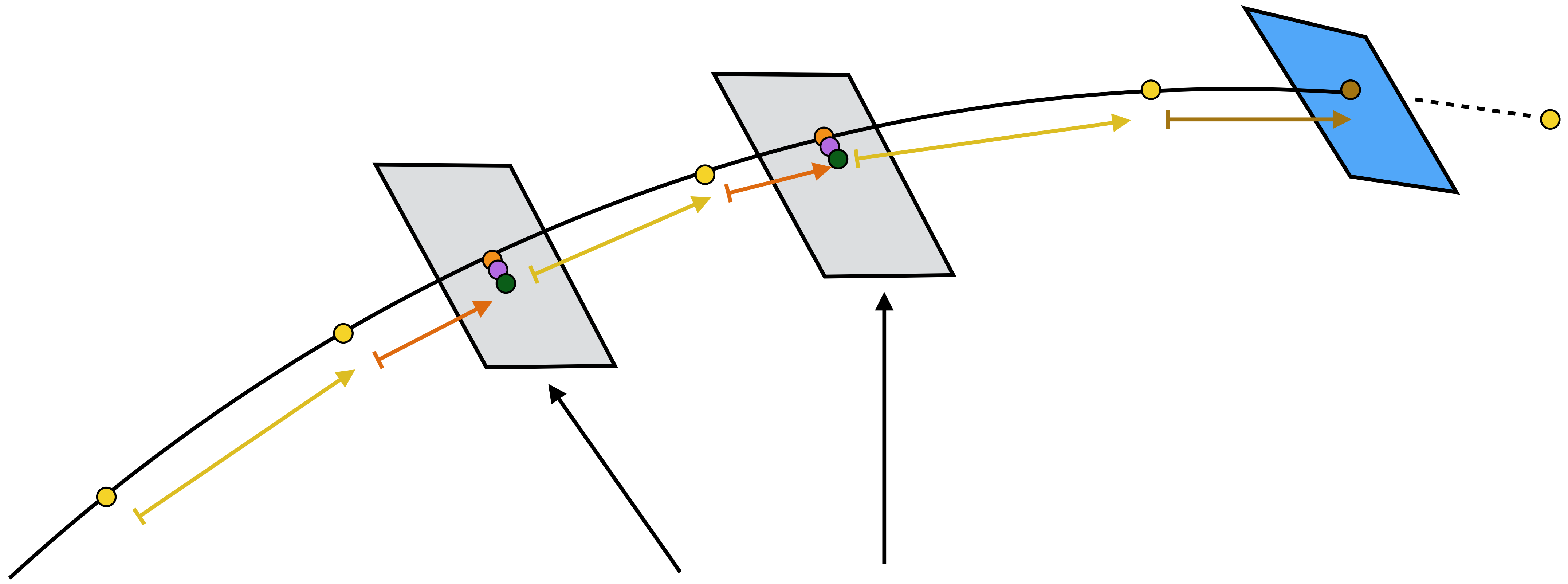
Next candidate surface

```
// Pre-Stepping: abort condition check
if (!state.options.abortList(result, state, m_stepper)) {
  // Pre-Stepping: target setting
  m_navigator.target(state, m_stepper);
  // Stepping loop
  ACTS_VERBOSE("Starting stepping loop.");

  terminatedNormally = false;  // priming error condition

  // Propagation loop : stepping
  for (; result.steps < state.options.maxSteps; ++result.steps) {
    // Perform a propagation step - it takes the propagation state
    Result<double> res = m_stepper.step(state);
    if (res.ok()) {
      // Accumulate the path length
      double s = *res;
      result.pathLength += s;
      ACTS_VERBOSE("Step with size = " << s << " performed");
    } else {
      ACTS_ERROR("Step failed with " << res.error() << ": "
                                     << res.error().message());
      // pass error to caller
      return res.error();
    }
    // Post-stepping:
    // navigator status call - action list - aborter list - target call
    m_navigator.status(state, m_stepper);
    state.options.actionList(state, m_stepper, result);
    if (state.options.abortList(result, state, m_stepper)) {
      terminatedNormally = true;
      break;
    }
    m_navigator.target(state, m_stepper);
  }
} else {
  ACTS_VERBOSE("Propagation terminated without going into stepping loop.");
}
```

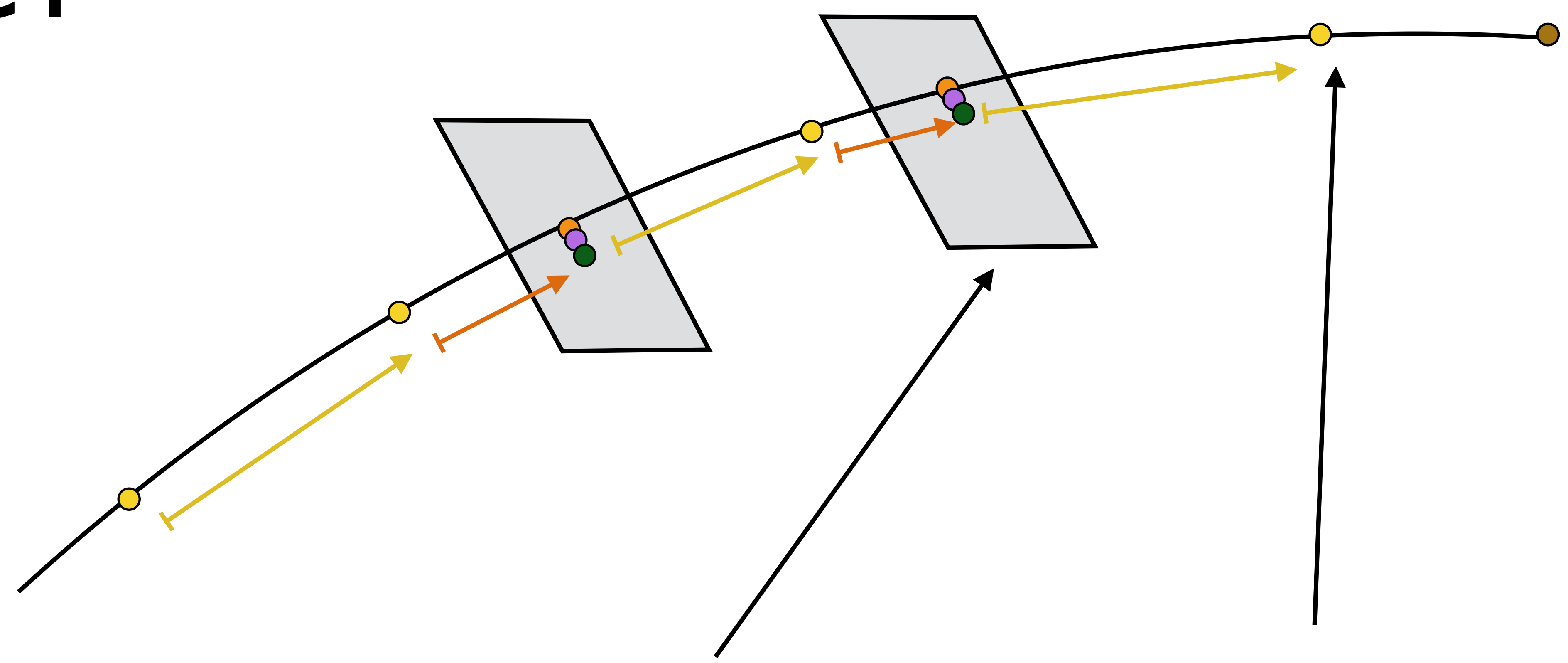
Free/bound state propagation: `Stepper`



`BoundParameters` to `BoundParameters`

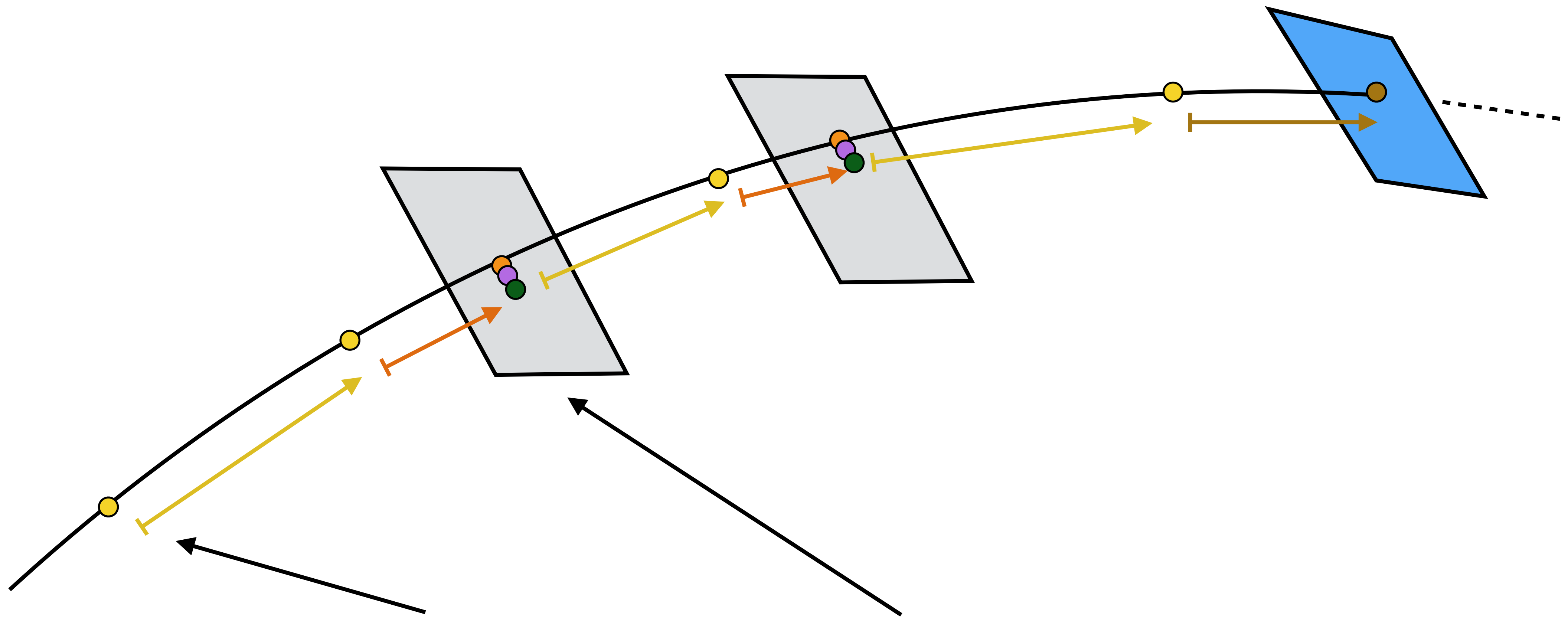
`CurvilinearParameters` to `CurvilinearParameters`

```
return  { BoundParameters, JacobianBoundToBound, ActsScalar };
          dim: 6           dim: 6x6
```



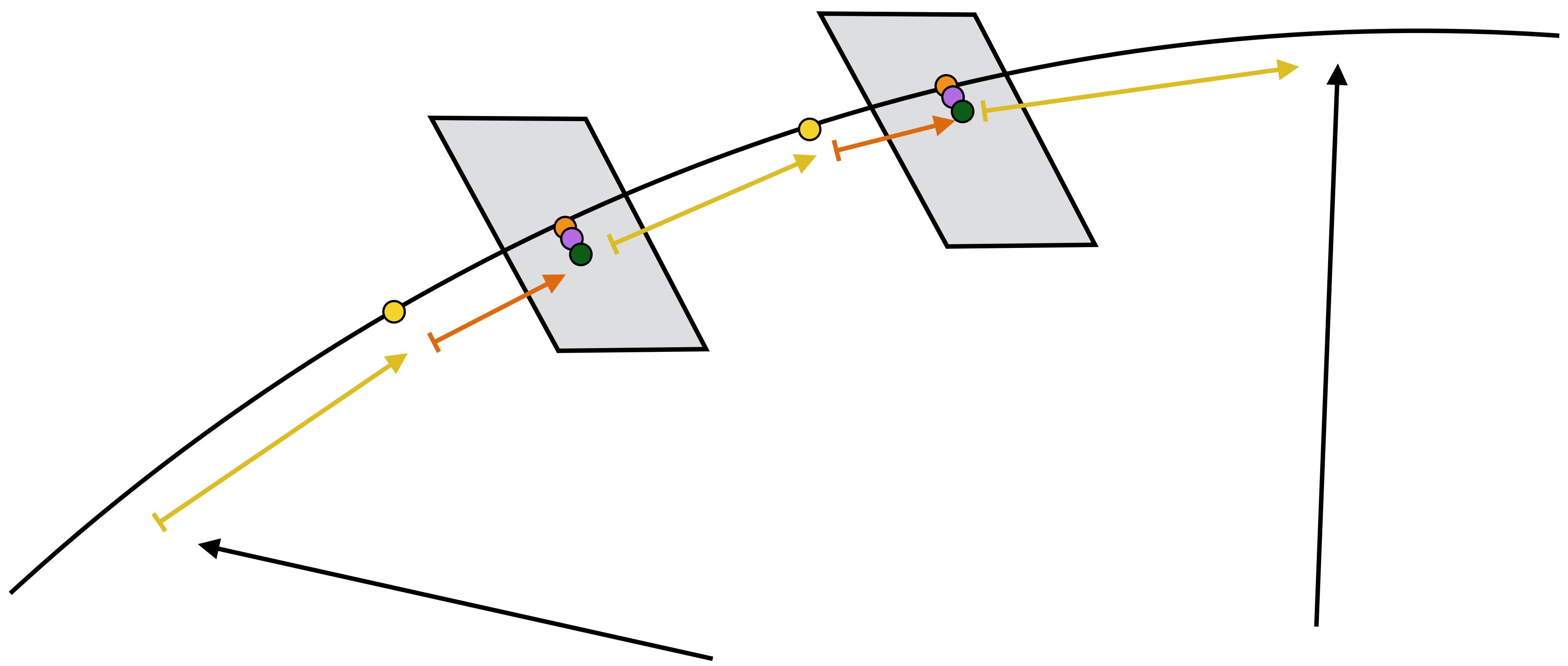
`BoundParameters` to `FreeParameters`

```
return  { FreeParameters, JacobianBoundToFree, ActsScalar };
          dim: 8          dim: 6x8
```



`FreeParameters` to `BoundParameters`

```
return  { BoundParameters, JacobianFreeToBound, ActsScalar };
          dim: 6           dim: 8x6
```



`FreeParameters` to `FreeParameters`

```
return  { FreeParameters, JacobianFreeToFree, ActsScalar };
          dim: 8          dim: 8x8
```

State propagation: `Stepper`

Standard stepper is 4th order Runge-Kutte-Nystrøm integrator

$$k_1 = f(t_n, y_n)$$

$$k_2 = f\left(t_n + \frac{h}{2}, y_n + h\frac{k_1}{2}\right)$$

$$k_3 = f\left(t_n + \frac{h}{2}, y_n + h\frac{k_2}{2}\right)$$

$$k_4 = f\left(t_n + h, y_n + hk_3\right).$$

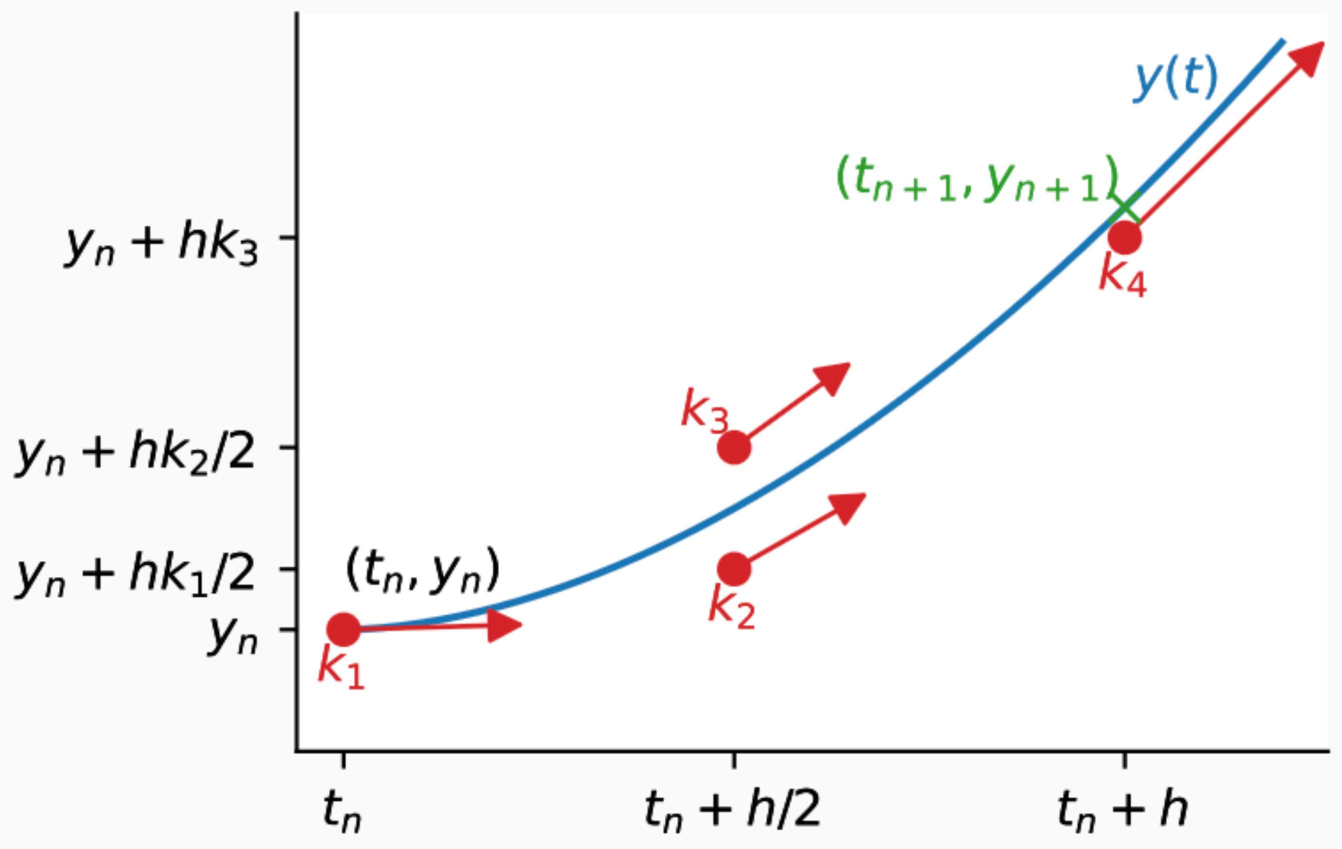


Fig. 5 *Illustration of the RKN method approximating a first order differential equation. Shown is the calculation of an estimate y_{n+1} at $t_{n+1} = t_n + h$, based on the current step (t_n, y_n). Shown are the four distinct points at which function $y(t)$ is evaluated, and which are blended to form the estimate.*

Two implementations:

- AtlasStepper, transcript in ACTS project
- EigenStepper, covariance transport

+ Dedicated covariance transport module based on auto-differatiation.

Free/bound state propagation

Point of Attention

Change to 8x8 internally did show some performance degradation

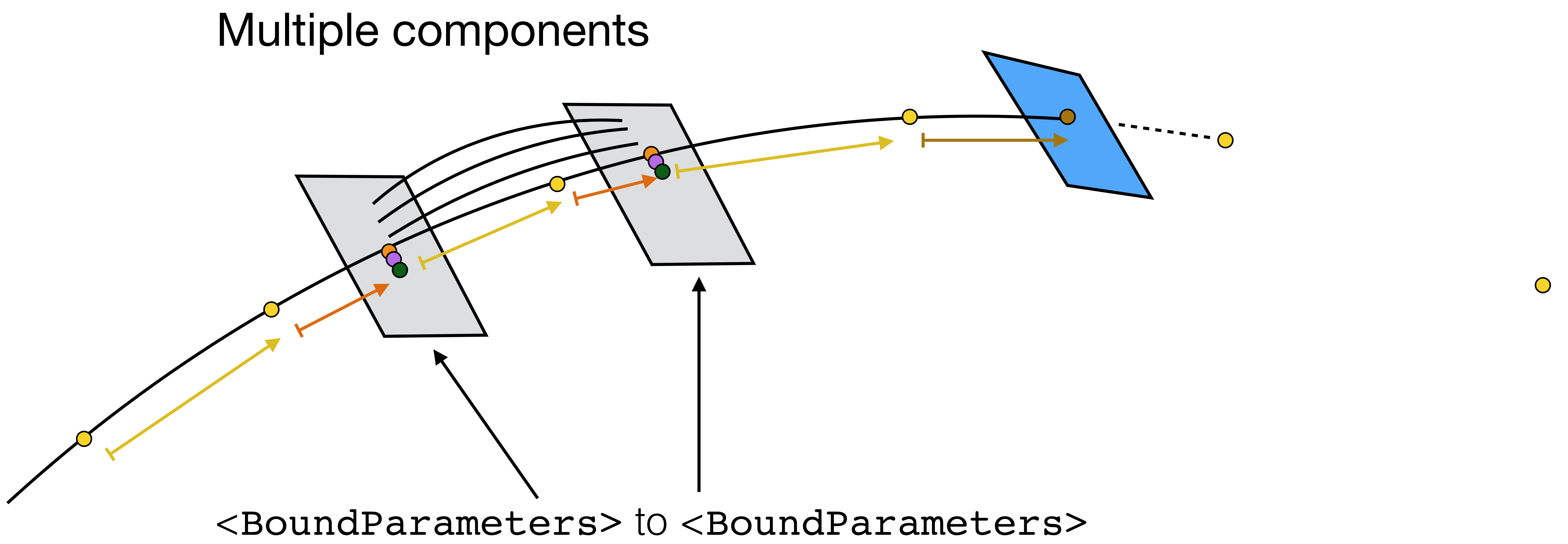
- Chance for some customised code in place

Similarity Function Scaling with Increasing Matrix Sizes (Using Google Benchmark)

Function Name	6x6 Time(ns)	7x7 TIme(ns)	8x8 Time(ns)	9x9 Time(ns)	10x10 Time(ns)
matrixMultEigen	37.4	94.5	244	303	368
matrixMultFastLoopTheTriple	49.6	119	163	234	311
matrixMultFastLoopTripleInit	45.6	116	152	229	311
matrixMultFastLoopTripleRef	50.5	121	166	238	313
matrixMultFastLoopTripleRefInit	45.6	117	154	230	310
sMatrixSimilarity	59.1	112	151	362	483

Work by Scott Hurley this summer on matrix multiplication profiling

GSF: MultiStepper



Gaussian sum filter (multi variant Kalman filter) needs the propagation of several components for each measurement surface

- Enabled by a dedicated stepper, that allows to transport multiple components in parallel
- All components follow the same navigation stream (otherwise impossible for the fit to converge)
- Demonstrator of the success of component design choice
  - Fits into the current design by exchanging the stepper with a multi component stepper
- See talk by Benjamin on GSF

Actor & Aborter design

Propagator is compile-time extendable via `Actor/Aborter` mechanism

- function call takes templated Options class and hence changes return type appropriately
- has proven to have a high flexibility
- needs configuration at compile time


```cpp
// The step length logger for testing & end of world aborter
using MaterialInteractor = Acts::MaterialInteractor;
using SteppingLogger = Acts::detail::SteppingLogger;
using EndOfWorld = Acts::EndOfWorldReached;

// Action list and abort list
using ActionList = Acts::ActionList<SteppingLogger, MaterialInteractor>;
using AbortList = Acts::AbortList<EndOfWorld>;
using PropagatorOptions =
    Acts::DenseStepperPropagatorOptions<ActionList, AbortList>;

PropagatorOptions options(context.geoContext, context.magFieldContext,
                          Acts::LoggerWrapper{logger()});
options.pathLimit = pathLength;
```

Point of Attention

Clients like ATLAS have a large configuration space to cover

- A pre-compiled instance of every setup?
- A switch on/off action for Actors (some have a `sterile` flag already) ?