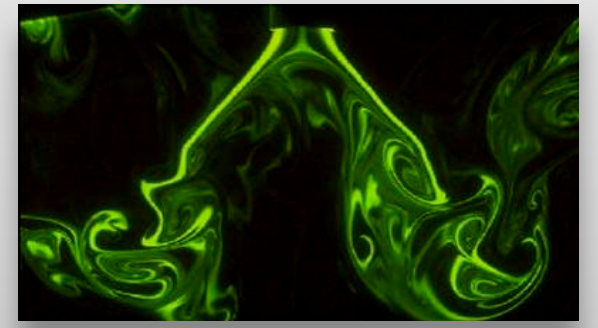
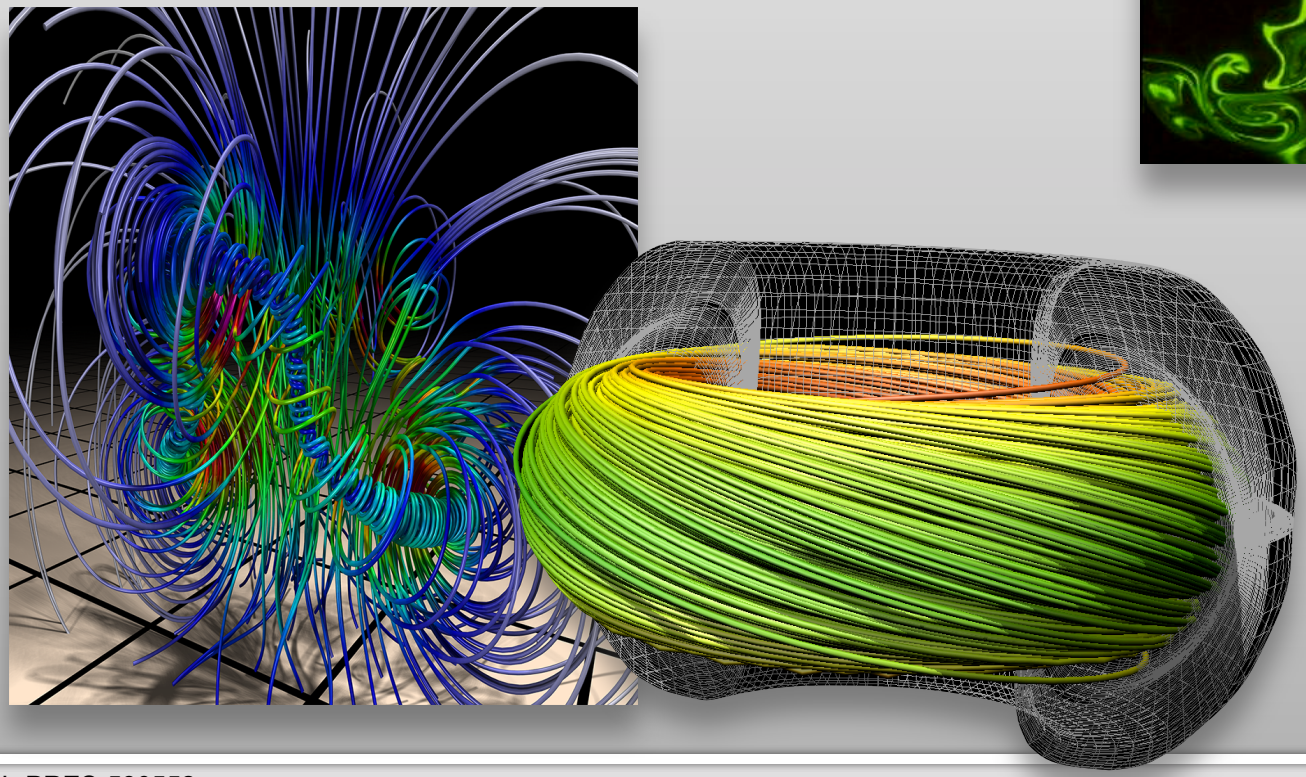
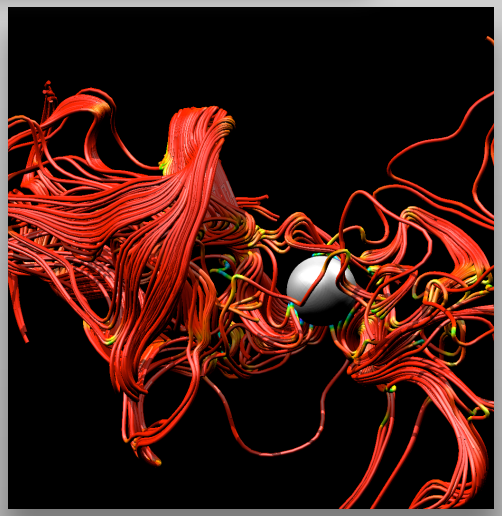
Streamlines

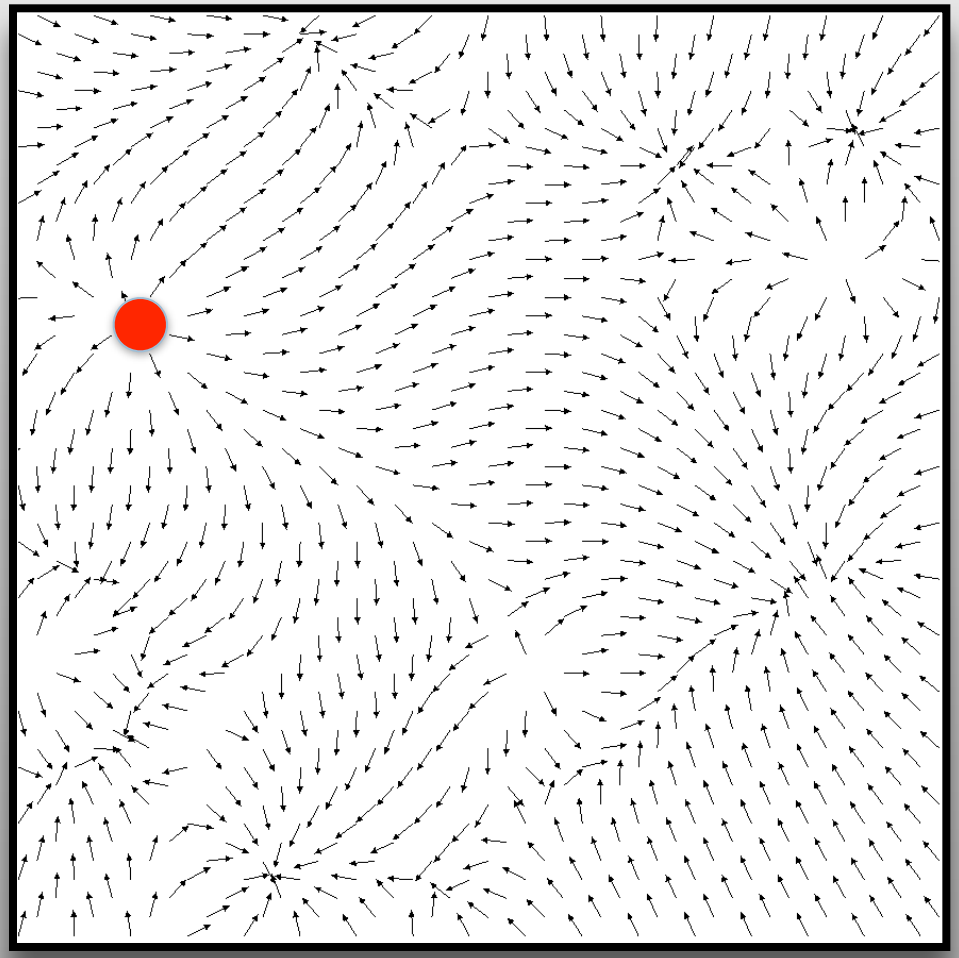
- Streamlines are an effective way to visualize flow
- One example from a larger class: Particle Based Visualization



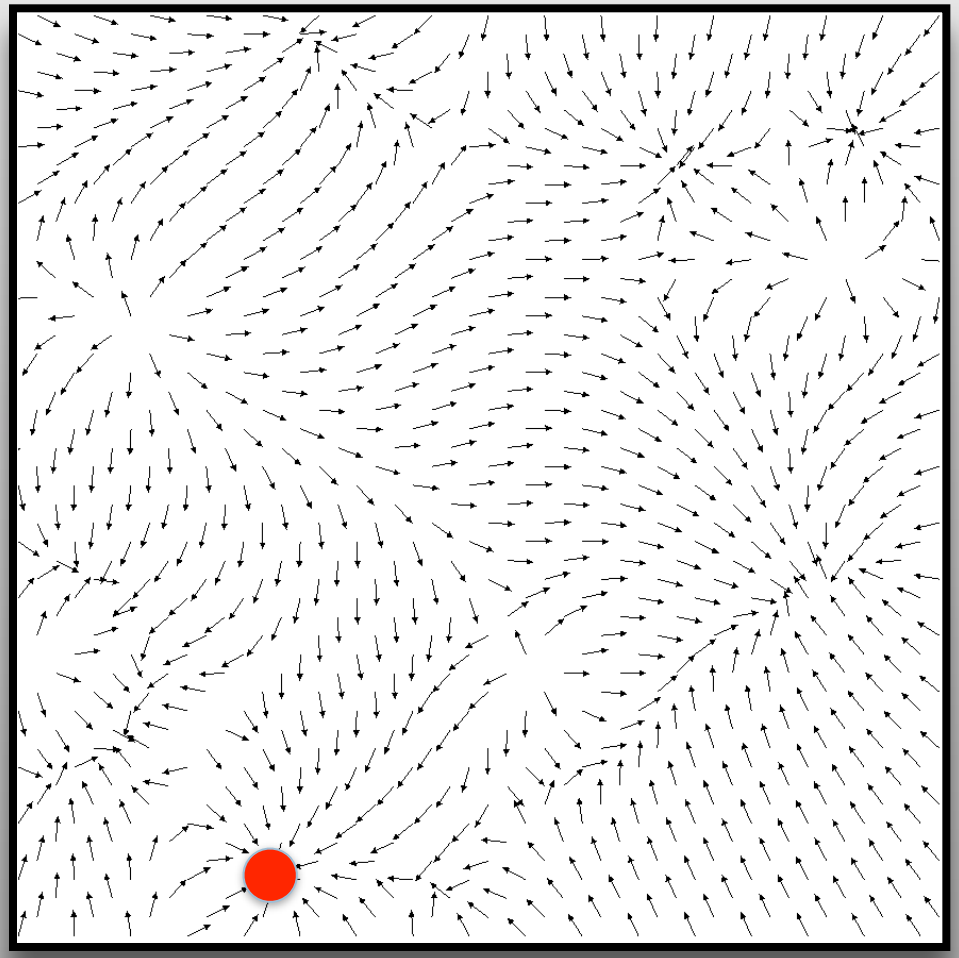




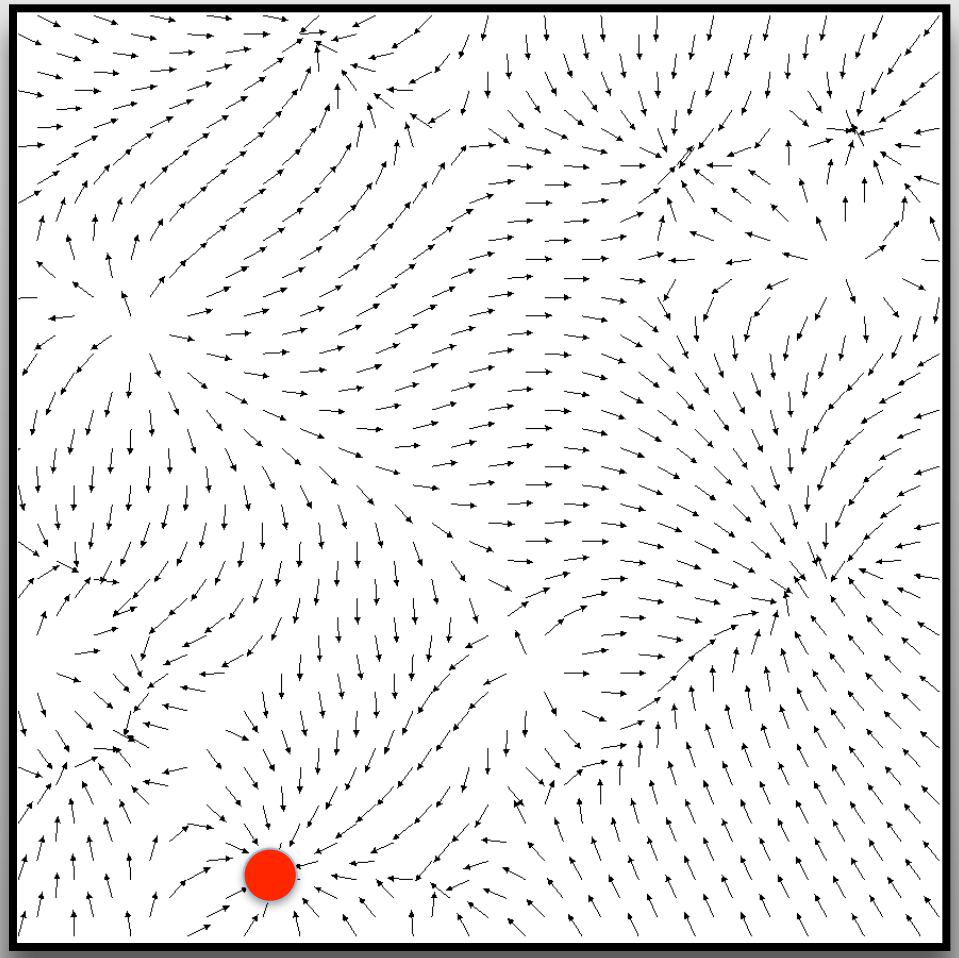
Particle advection is a foundational visualization algorithm



Particle advection is a foundational visualization algorithm

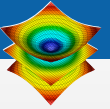


Particle advection is a foundational visualization algorithm



Particle advection creates integral curves

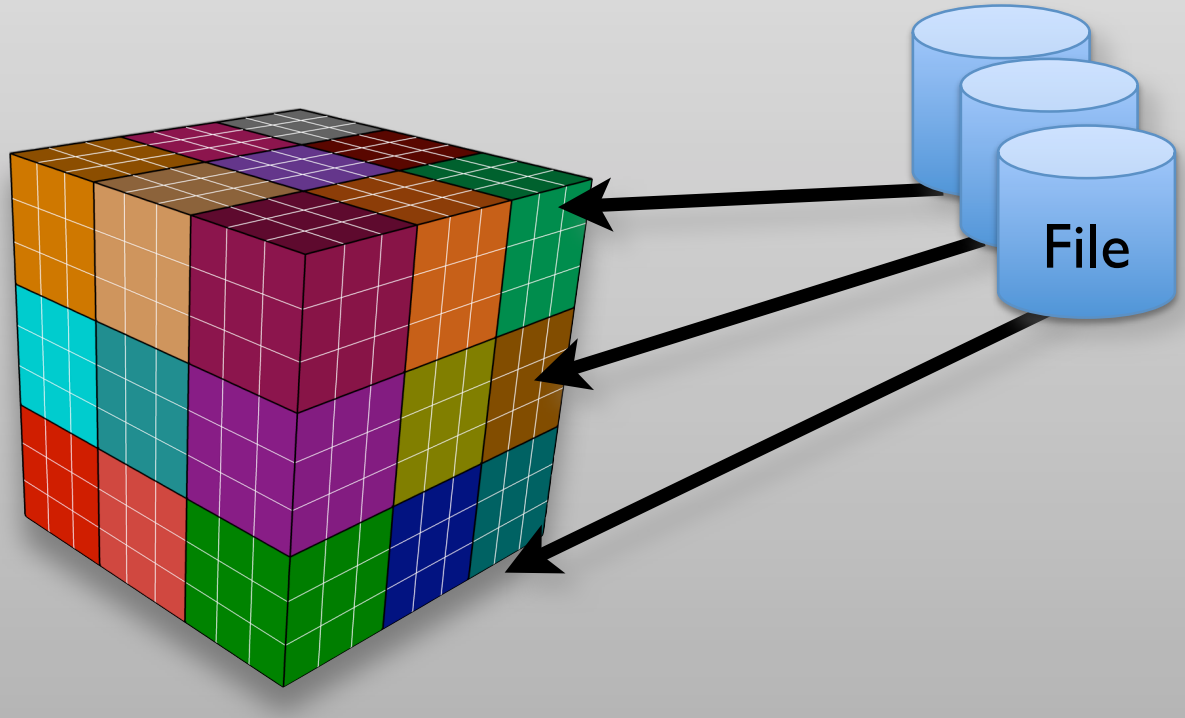
$$S'(t) = v(t, S(t)) \quad S(t_0) := x_0$$



Barriers to efficient particle advection

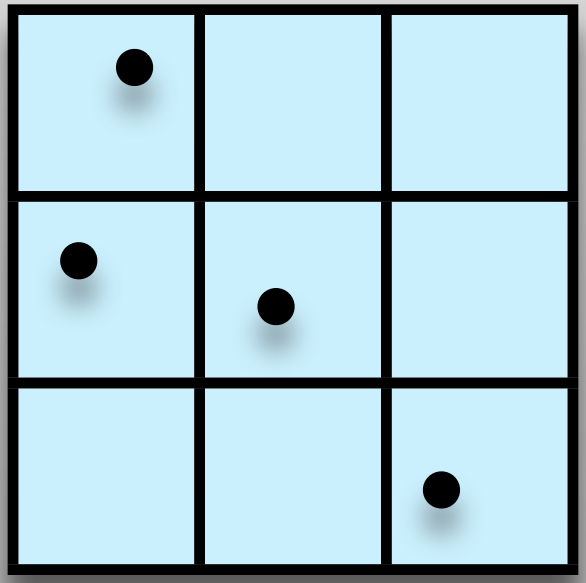
Barriers to efficient particle advection

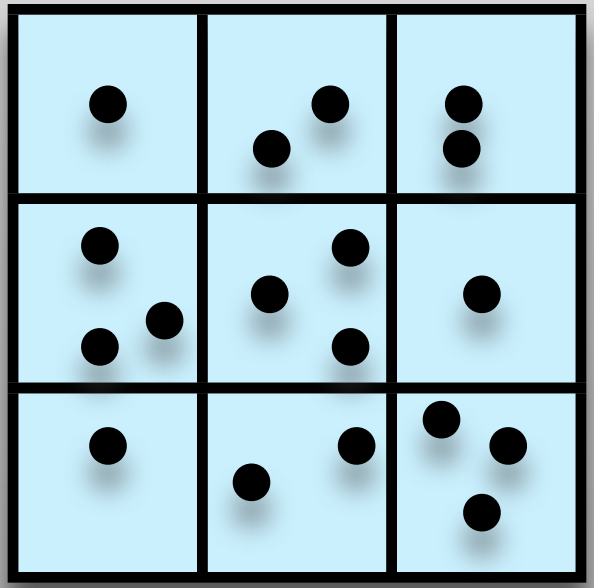
- Dataset size



Barriers to efficient particle advection

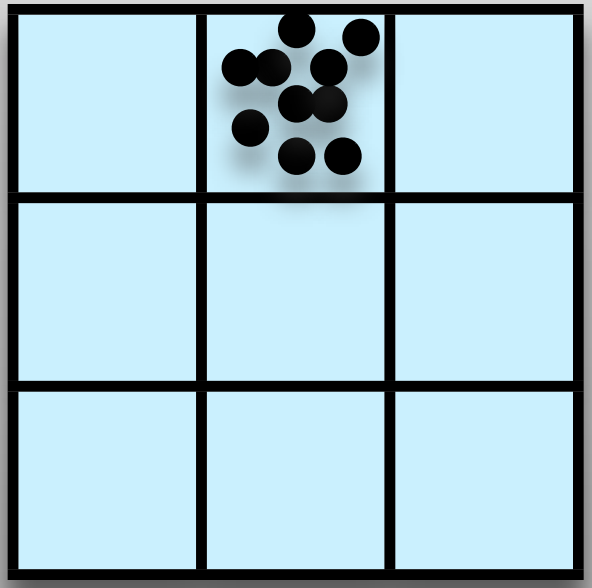
- Dataset size
- Seed count

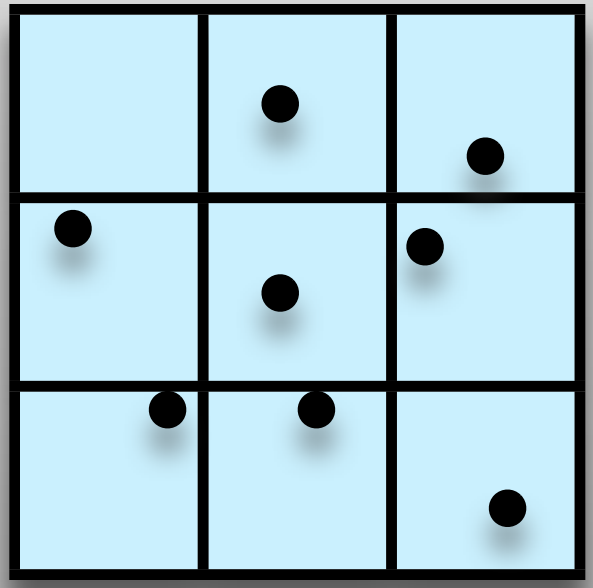




Barriers to efficient particle advection

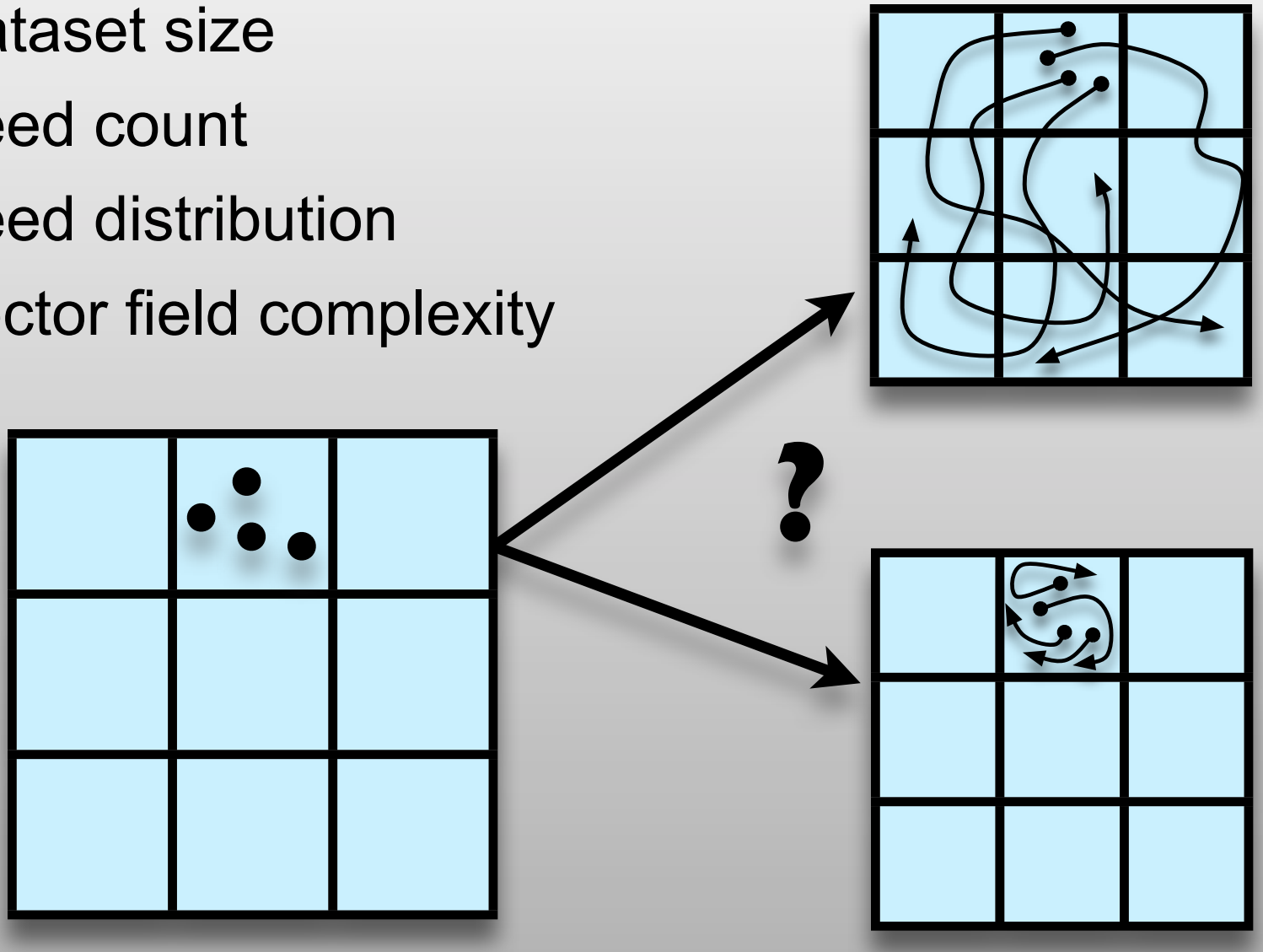
- Dataset size
- Seed count
- Seed distribution

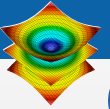




Barriers to efficient particle advection

- Dataset size
- Seed count
- Seed distribution
- Vector field complexity

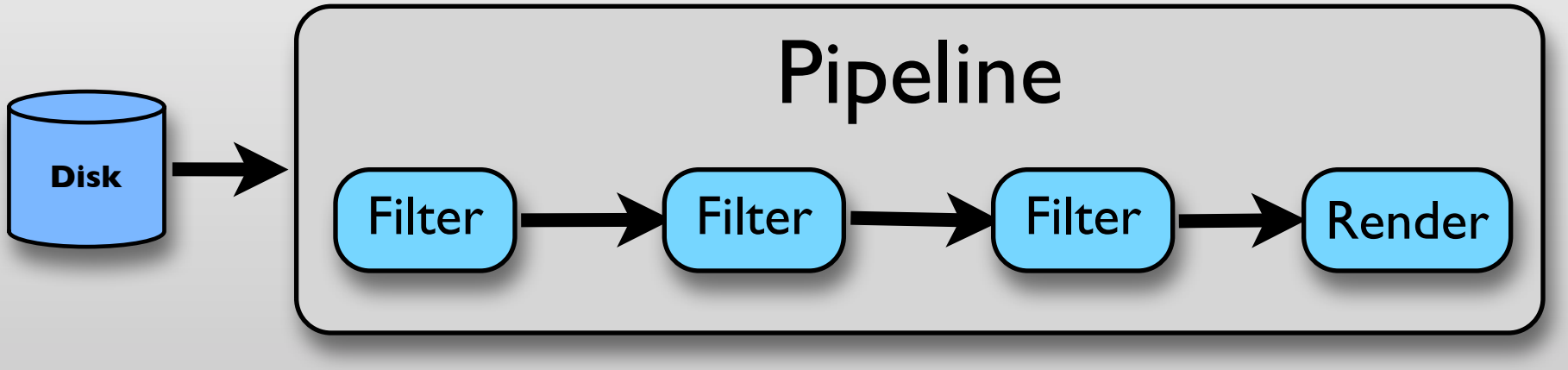




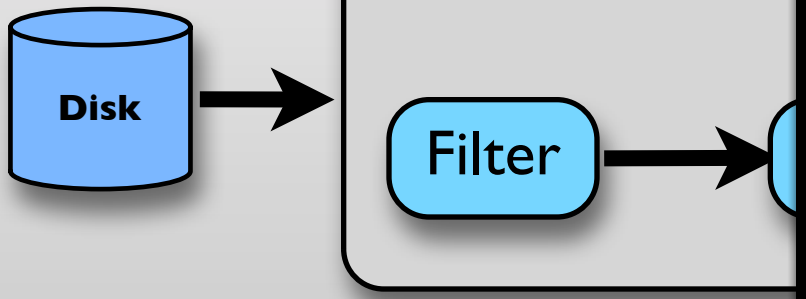
Goals

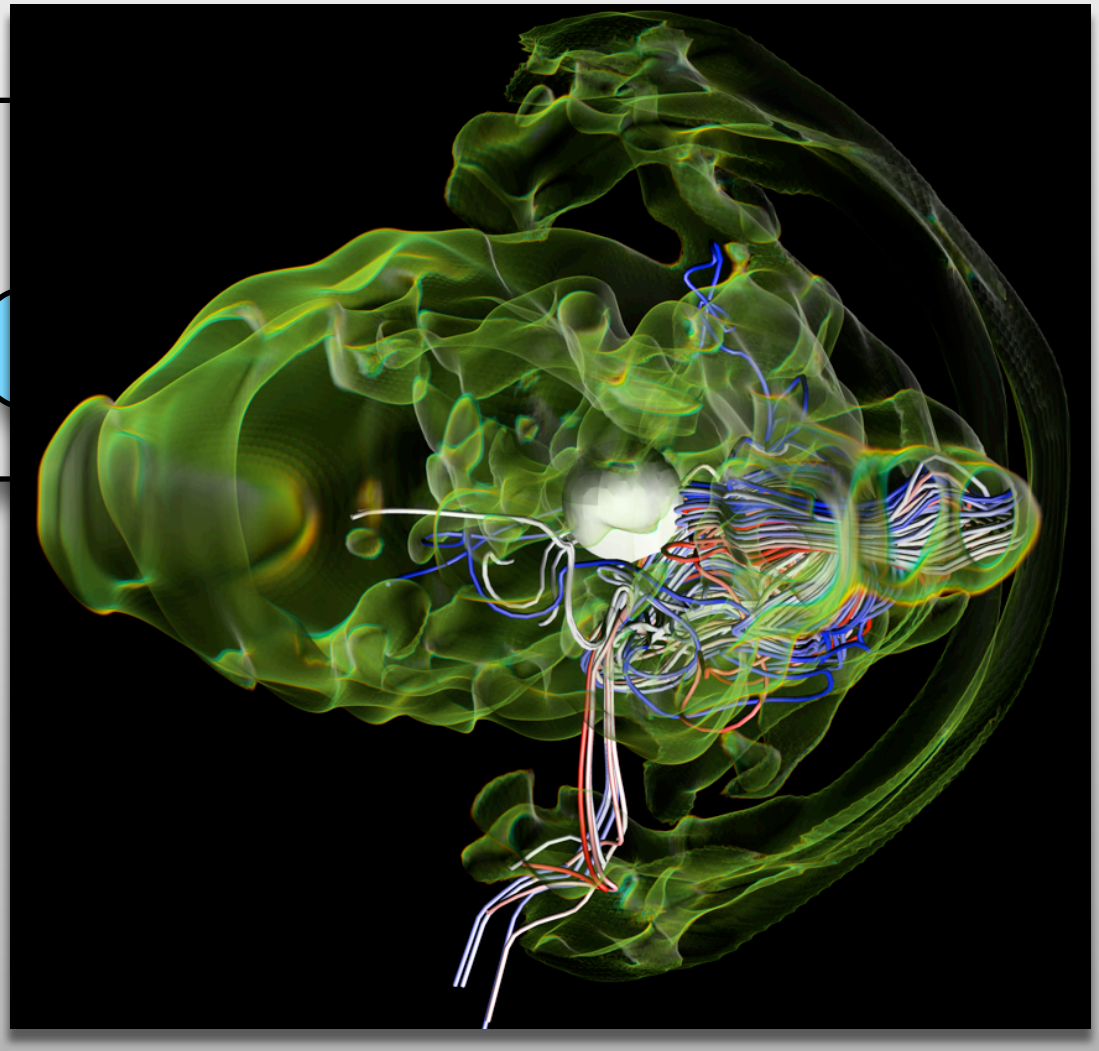
- This is really hard to do.
- Come up with a system that makes it easy to do
- Efficient across a broad set of uses cases
- Support use cases requiring 1, to millions of particles
  - Efficiency at **all** levels: computation, communication, I/O, memory
  - Flexible and extensible
  - Fit within existing large data analysis tools

Data flow network

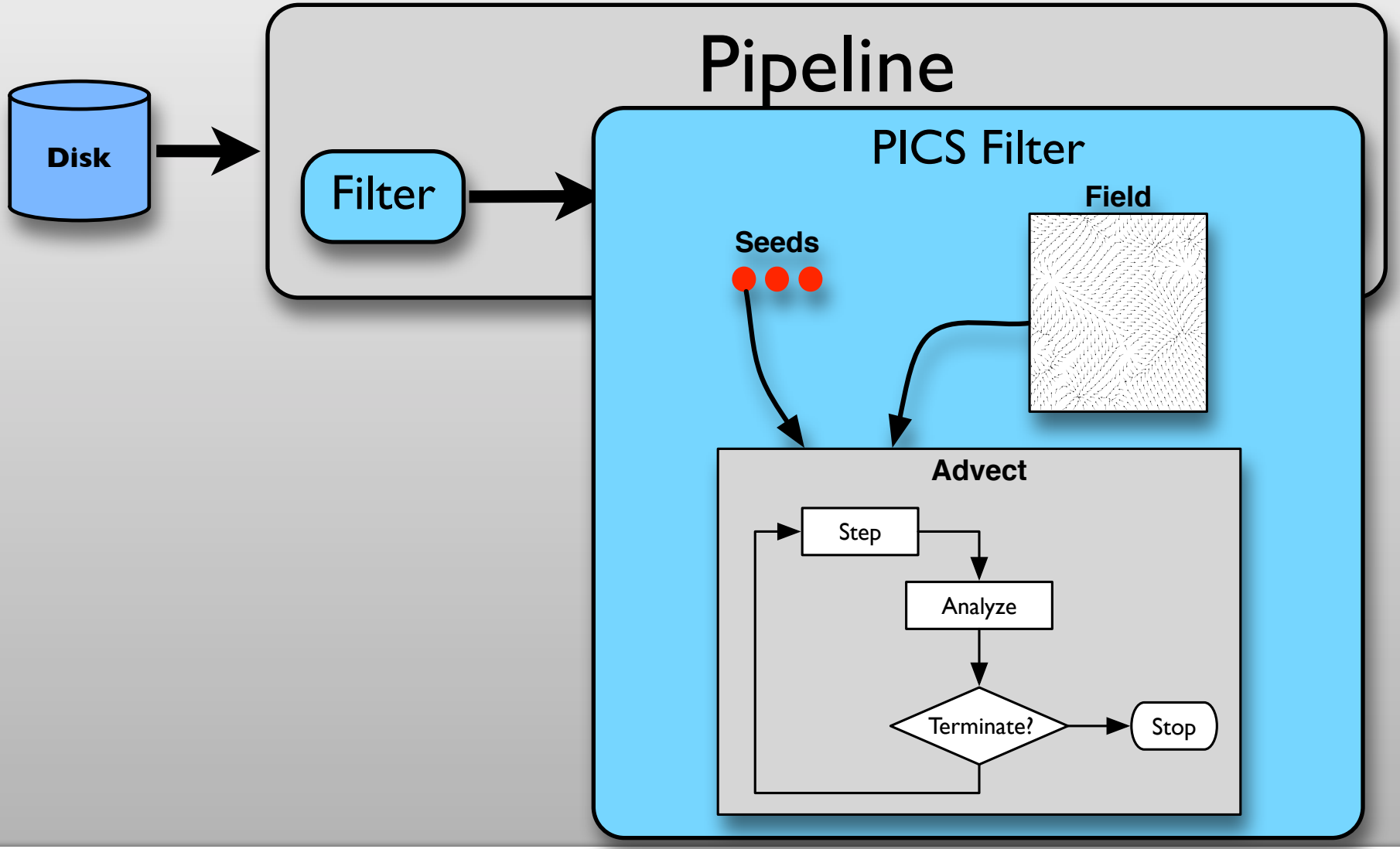


Data flow network

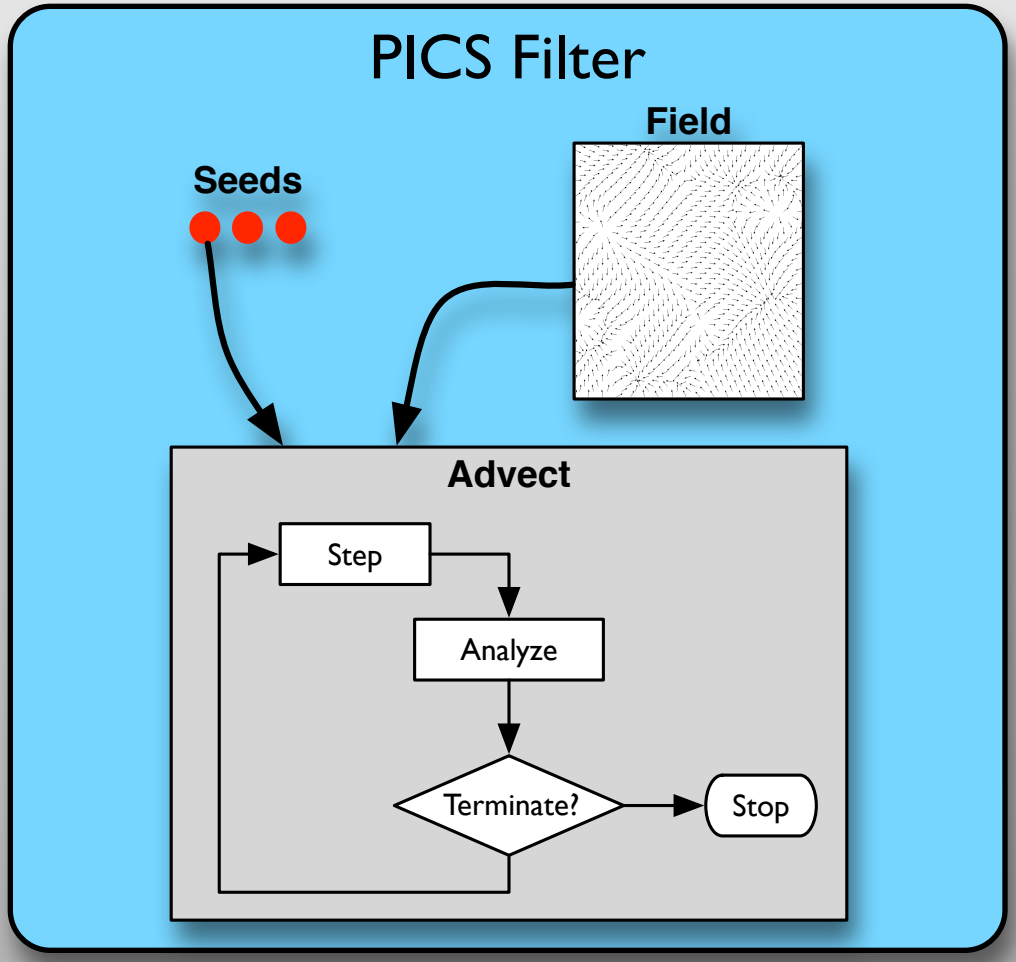




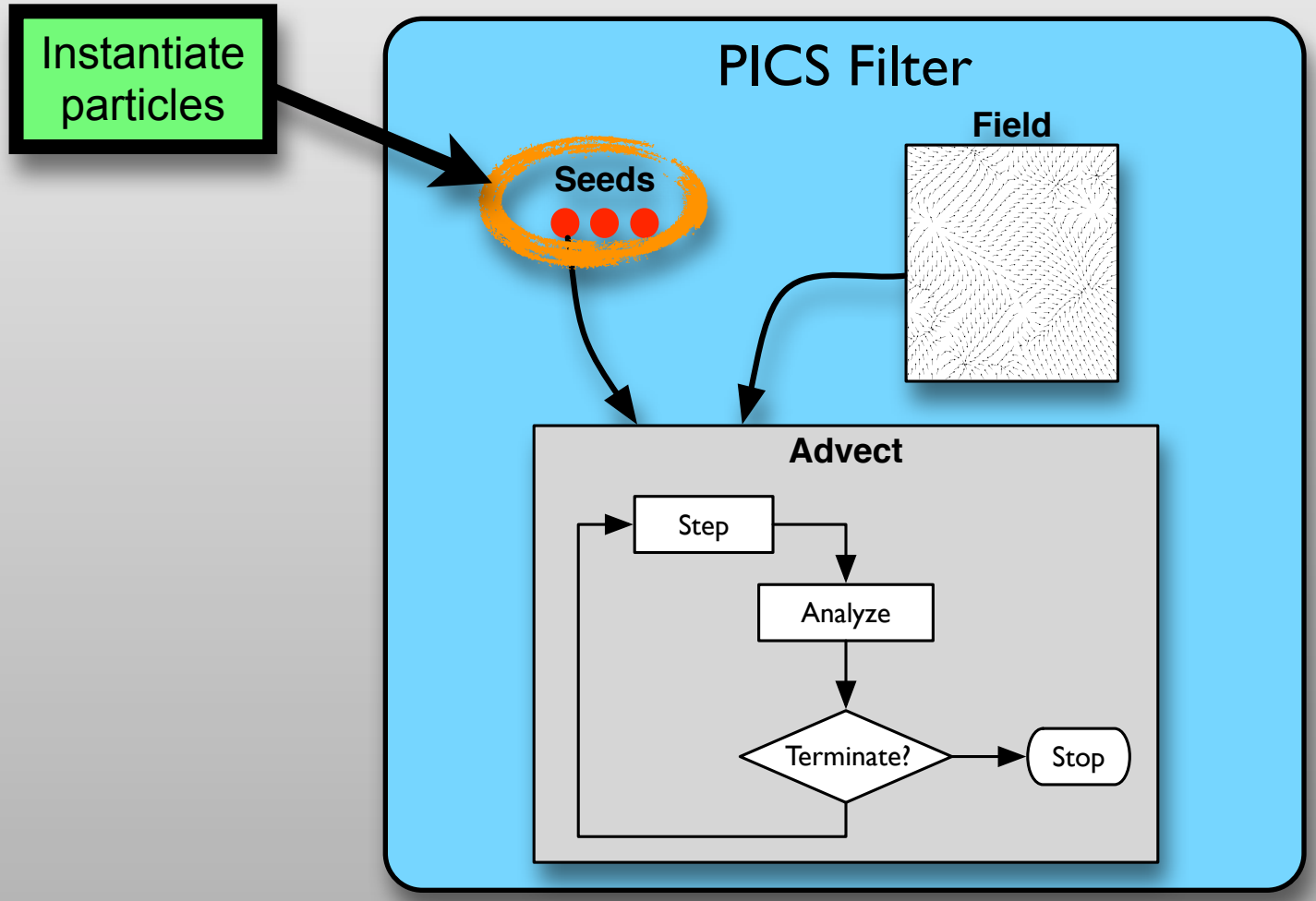
Data flow network



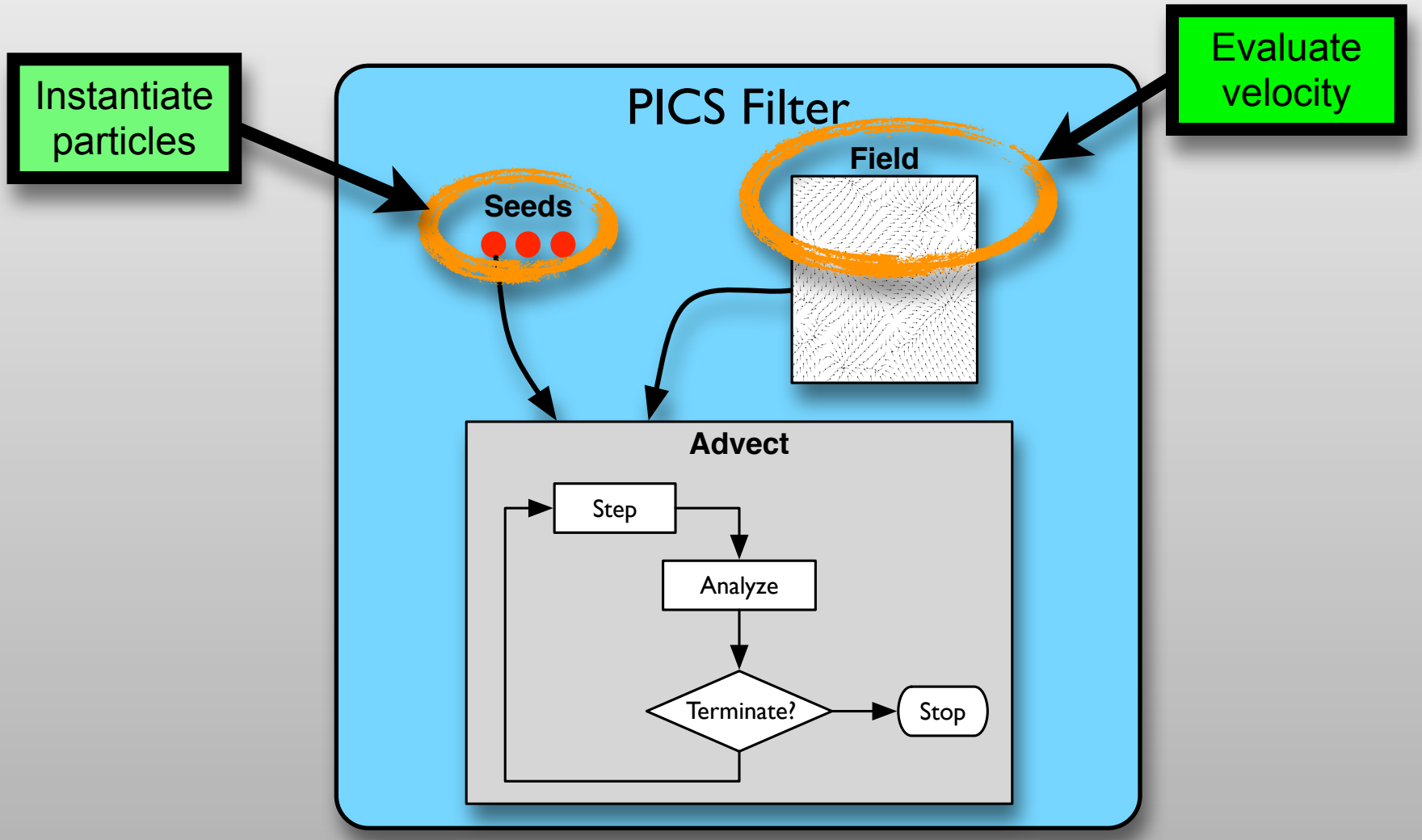
Data flow network



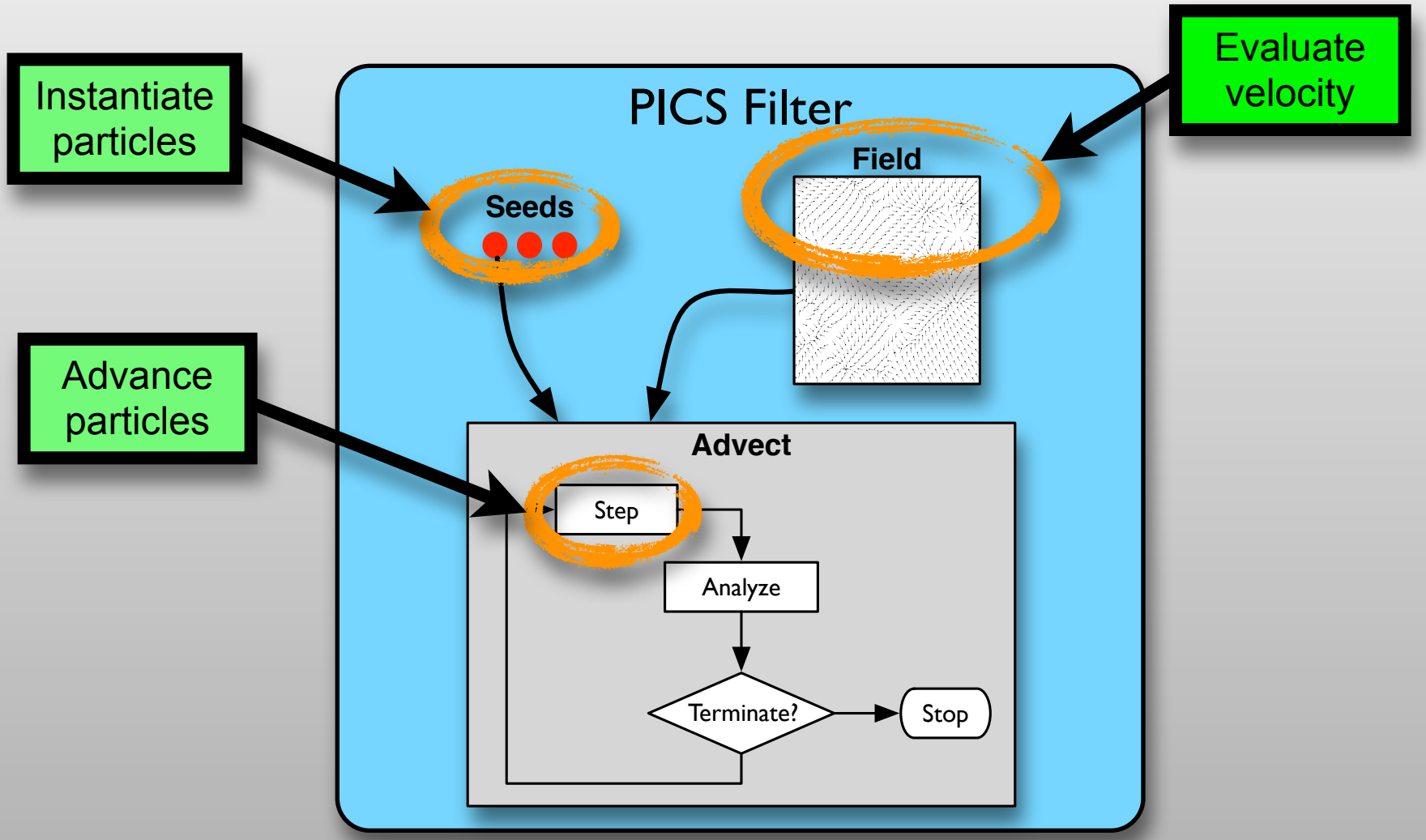
Data flow network



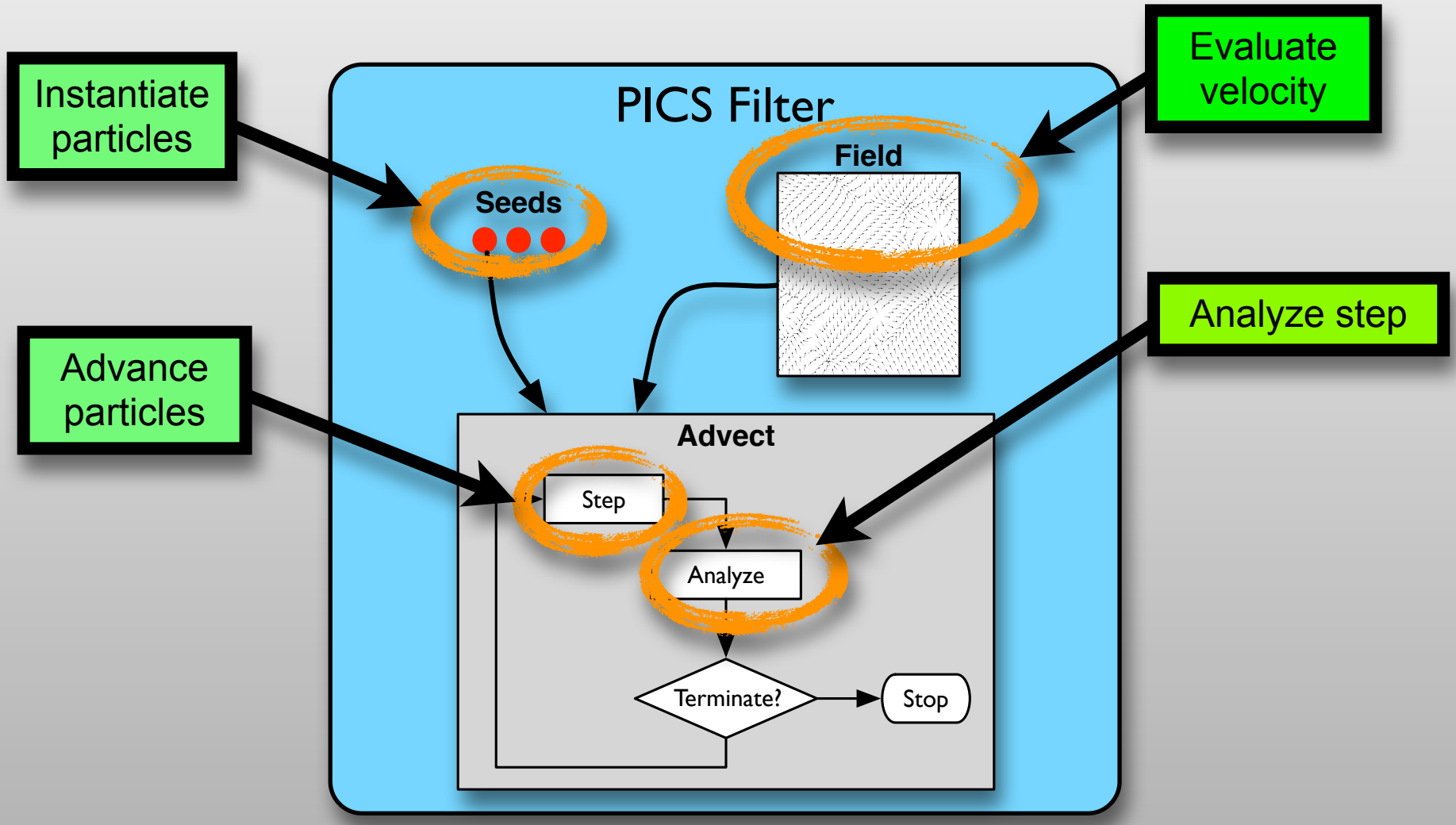
Data flow network



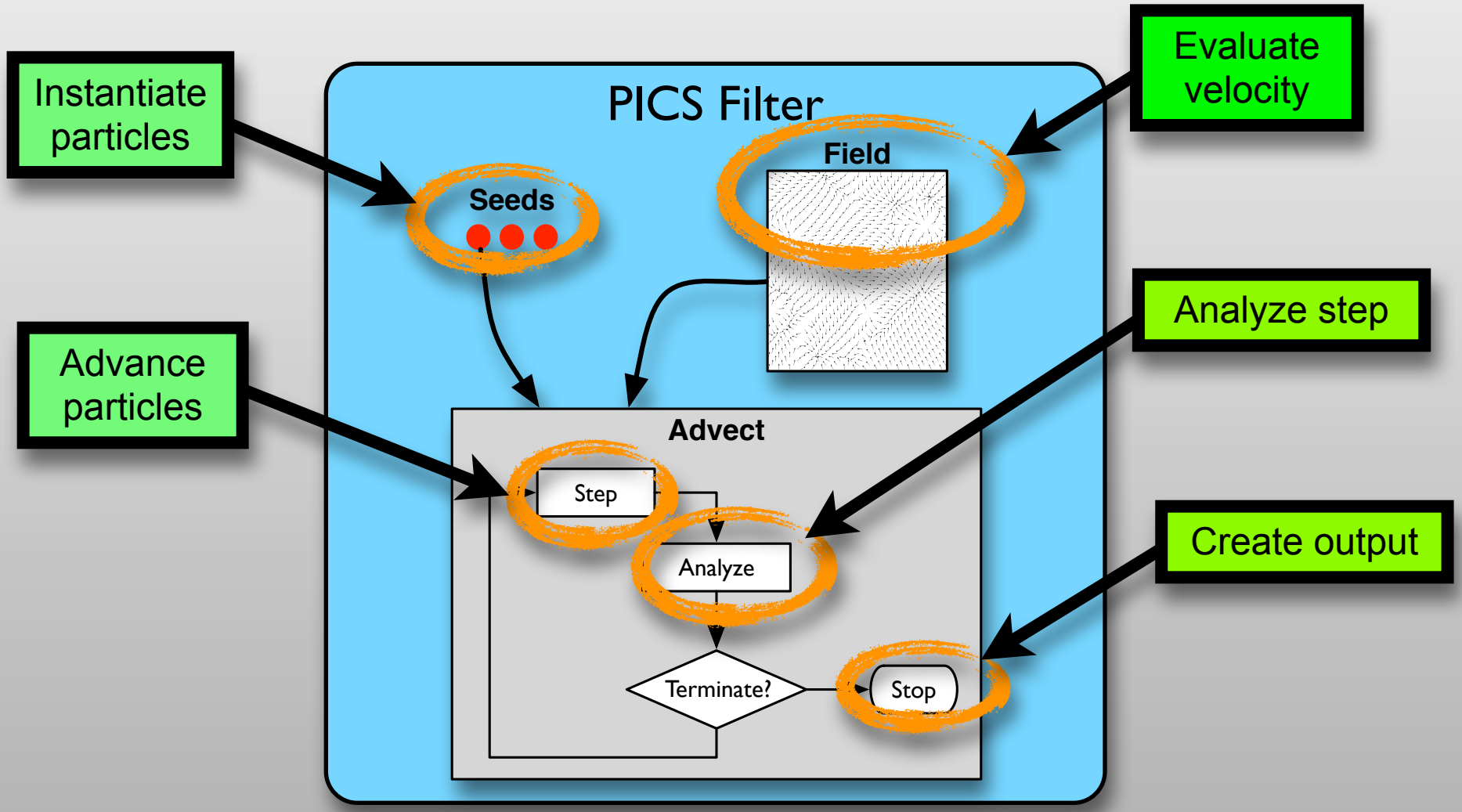
Data flow network



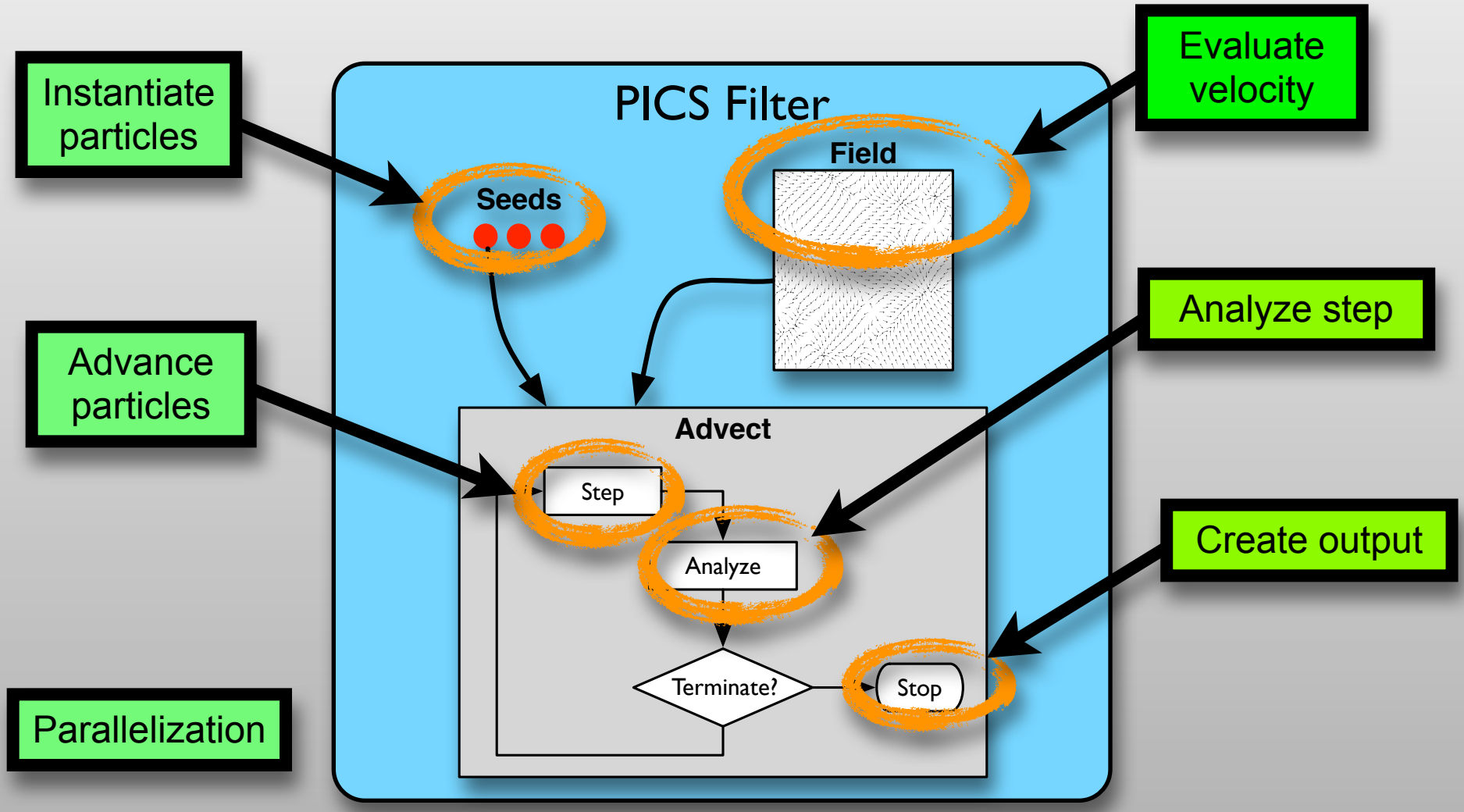
Data flow network



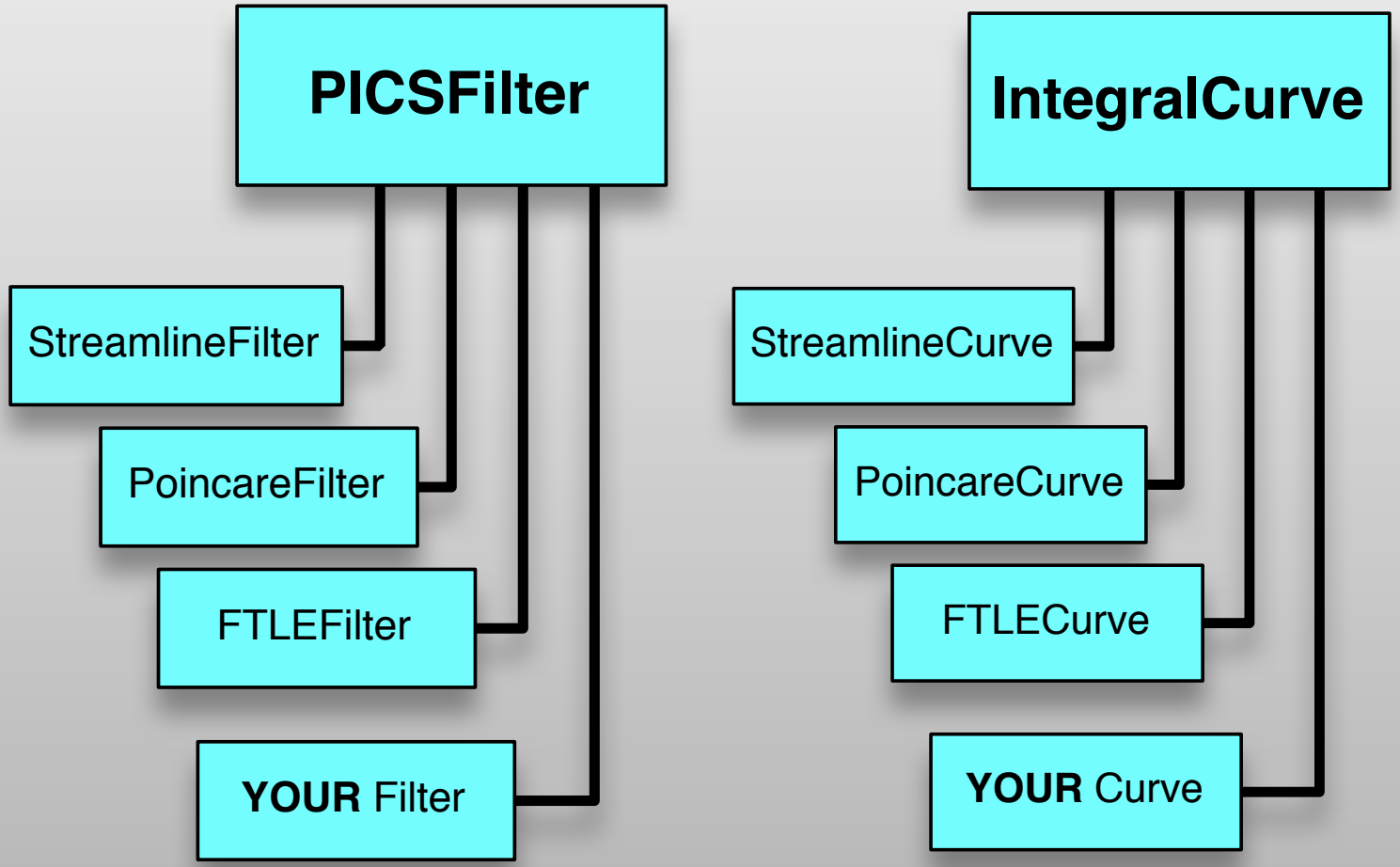
Data flow network



Data flow network

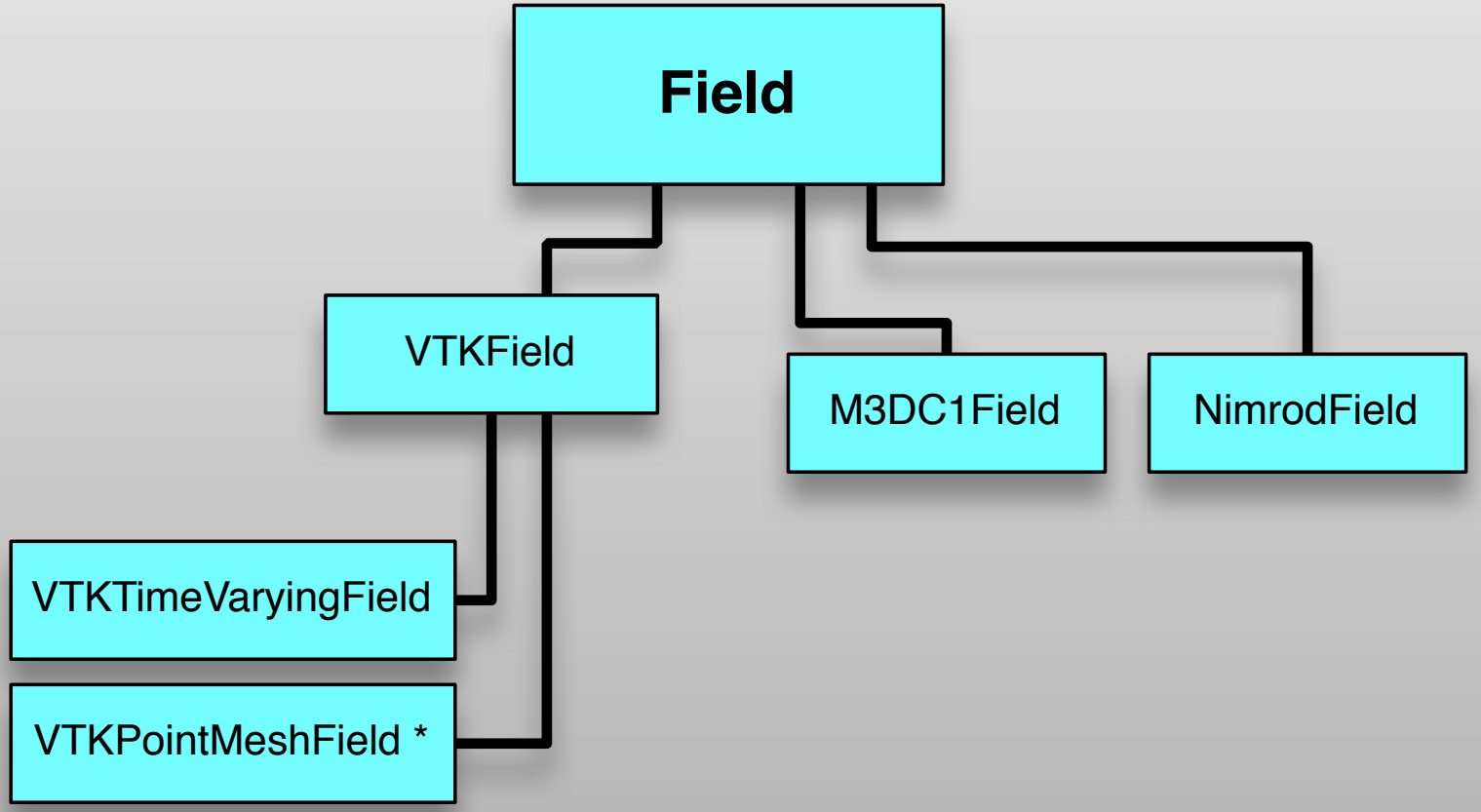


Inheritance Hierarchy



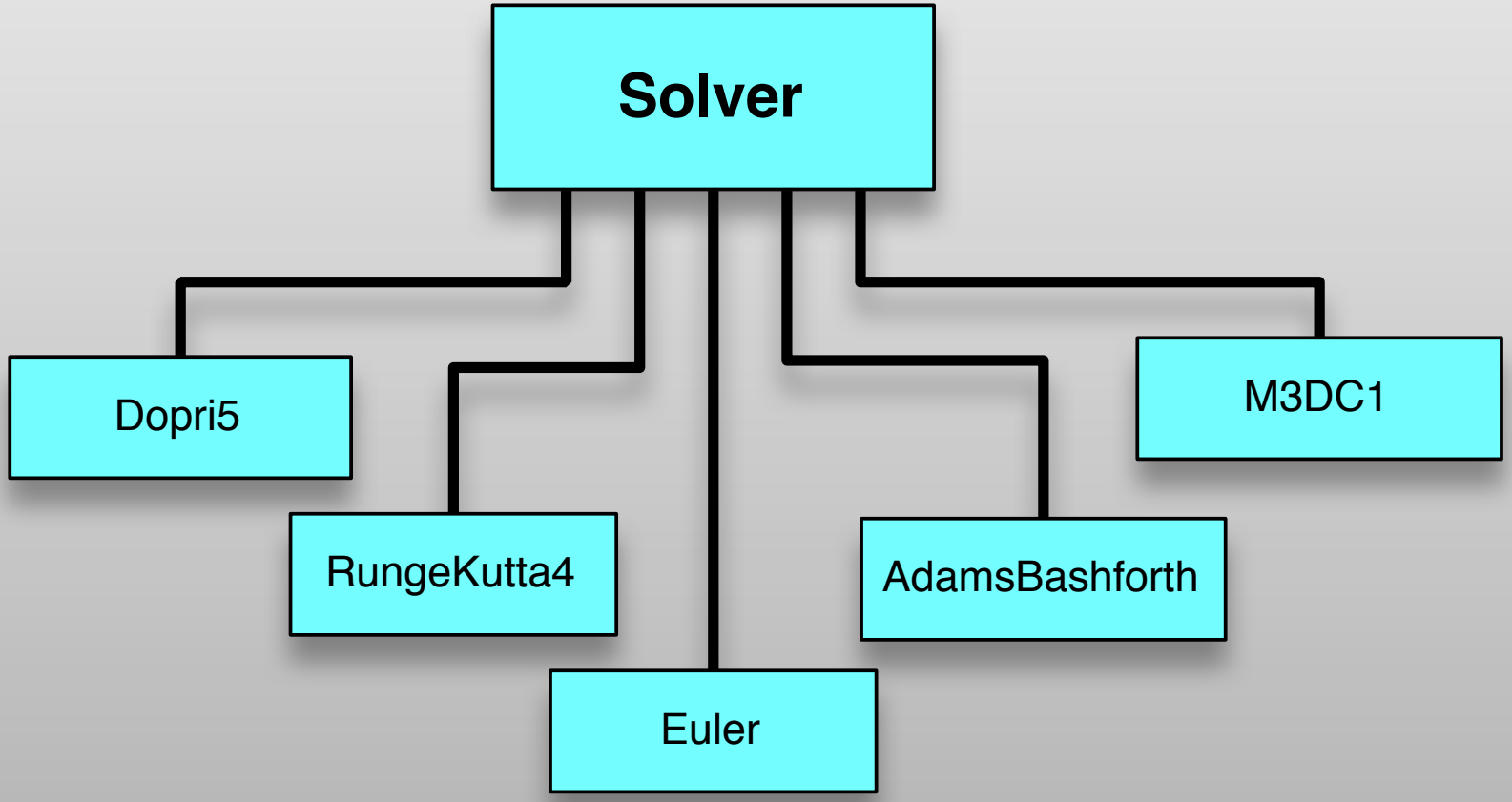
Field evaluation

vector Field::Evaluate(loc, time) = 0;



Particle Advection

location Solver::Step(location, time, field) = 0;



Particle path analysis

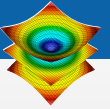
bool IntegralCurve::AnalyzeStep() = 0;

- User defined action for each step
- Return **true** for termination

Streamline
- Store step
- Store scalar

FTLE
- Store first step
- Store last step

Poincare
- Check for intersections
- Topological analysis to see if more seeds needed

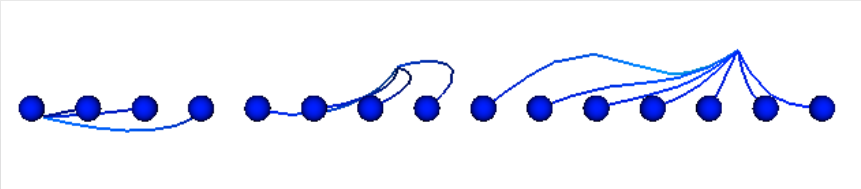


Initial locations

vector<locations> PICS::GetInitialLocations() = 0;

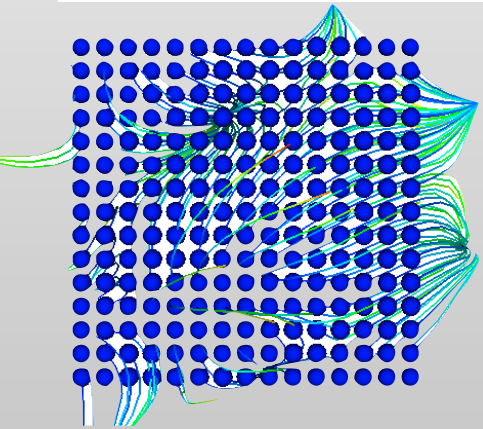
Initial locations

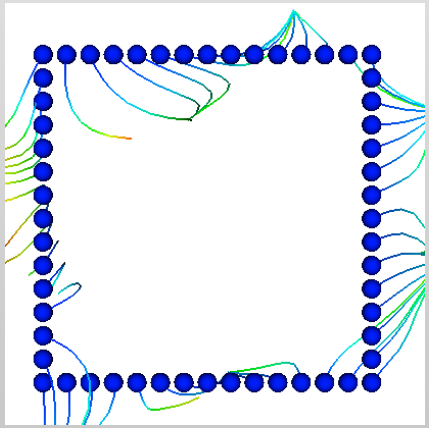
vector<locations> PICS::GetInitialLocations() = 0;



Initial locations

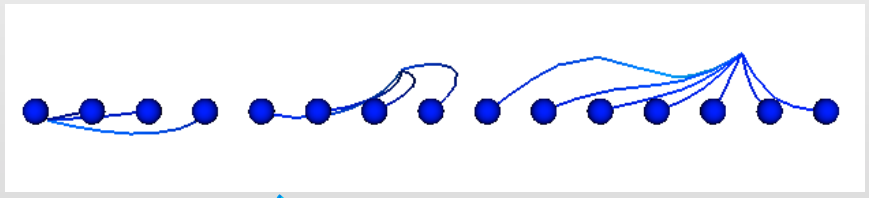
vector<locations> PICS::GetInitialLocations() = 0;

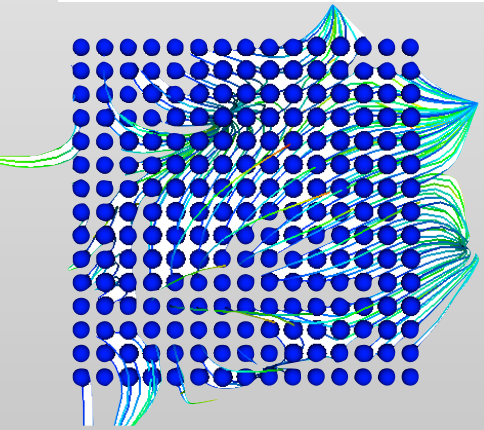


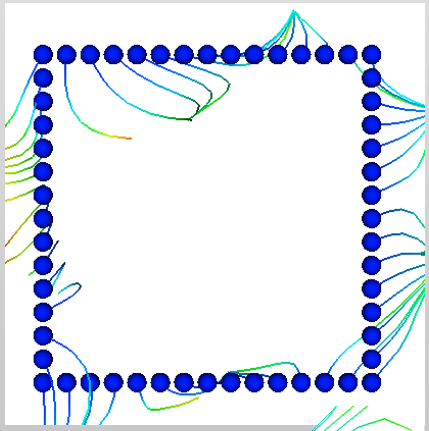


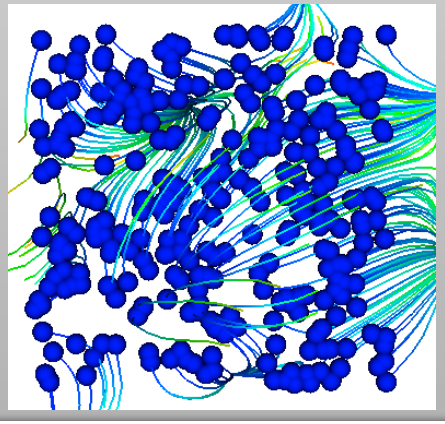
Initial locations

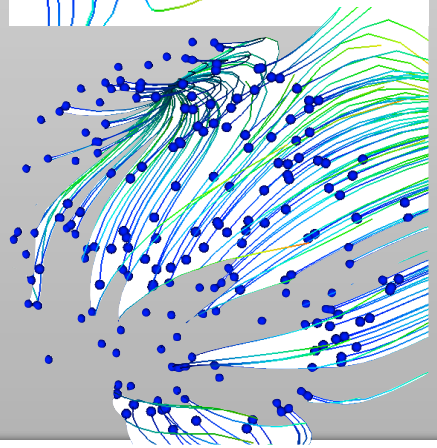
vector<locations> PICS::GetInitialLocations() = 0;





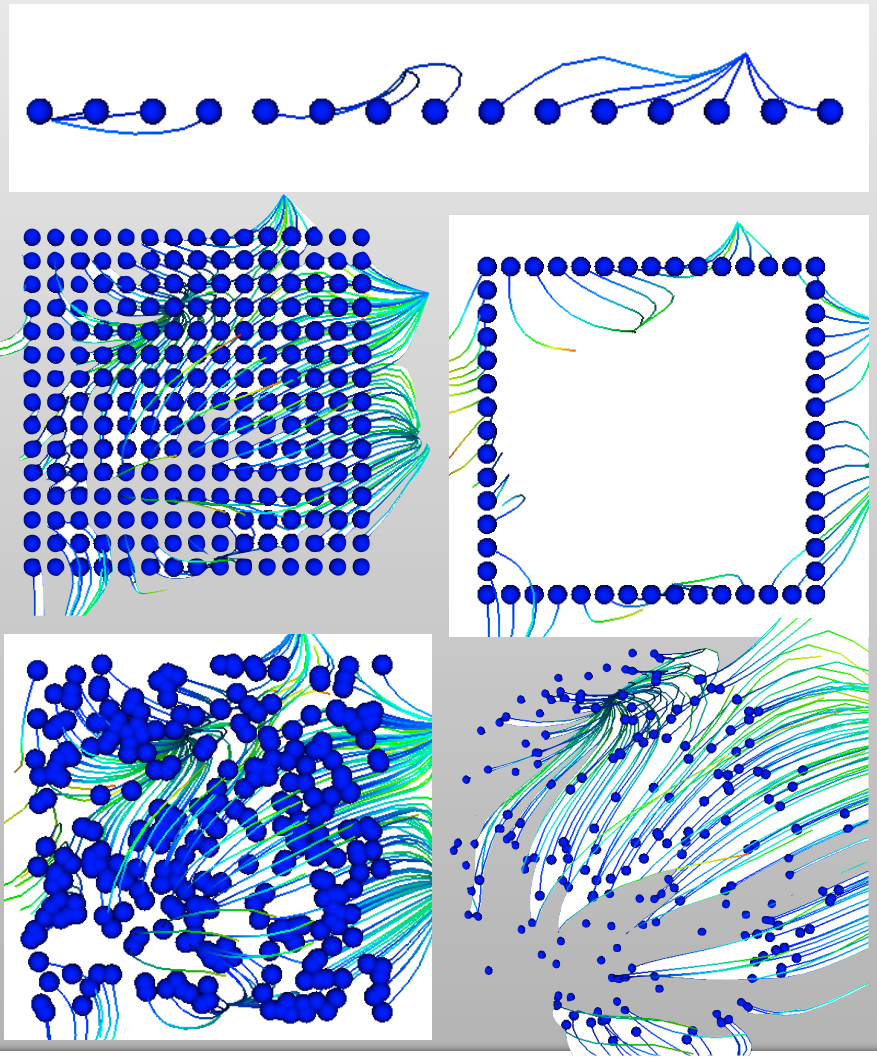


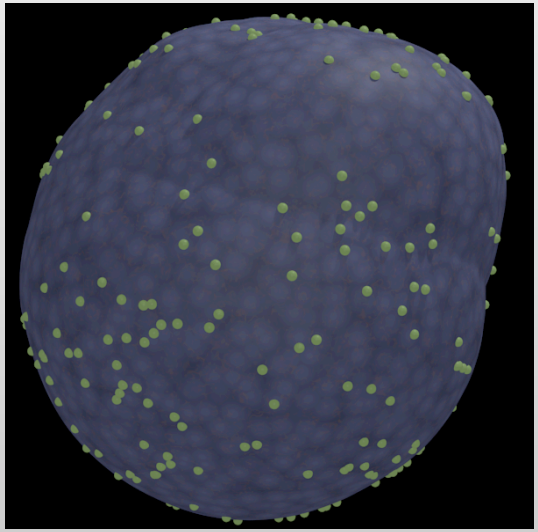




Initial locations

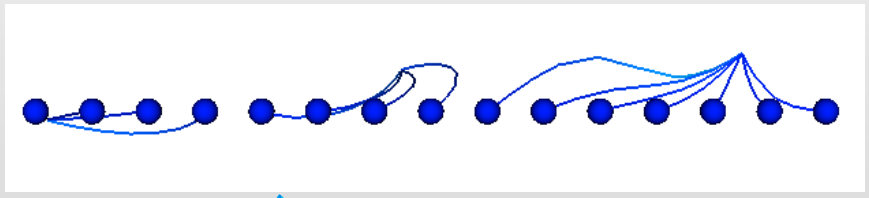
vector<locations> PICS::GetInitialLocations() = 0;

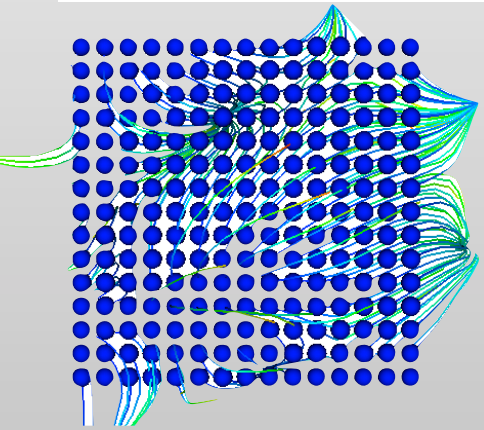


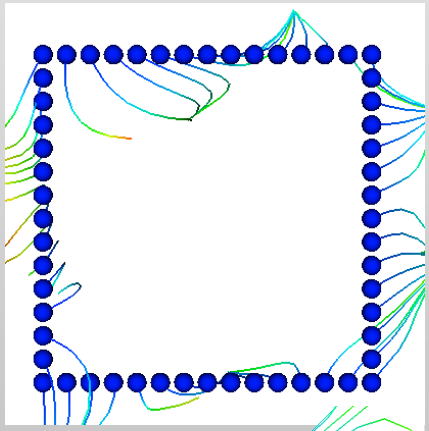


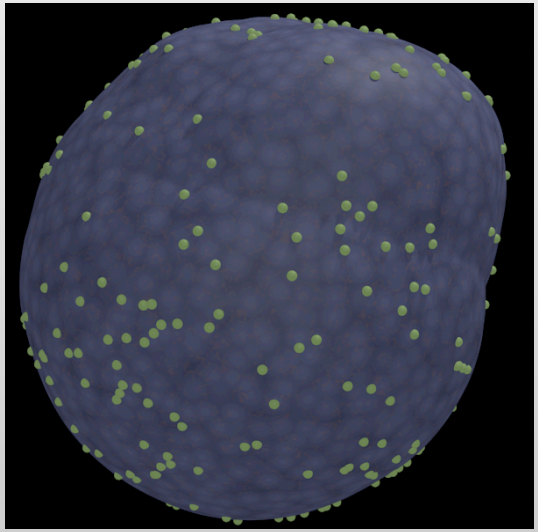
Initial locations

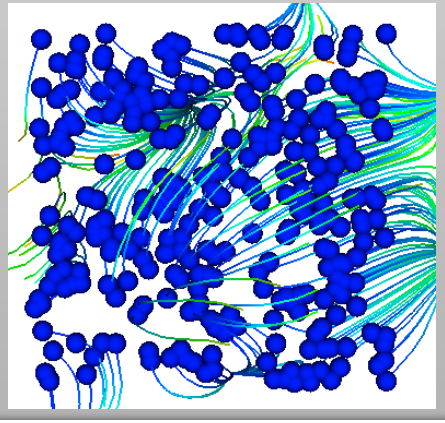
vector<locations> PICS::GetInitialLocations() = 0;

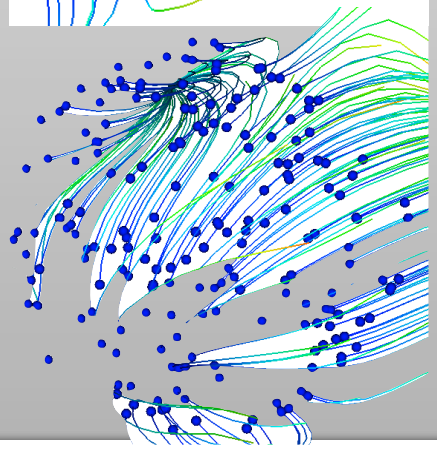


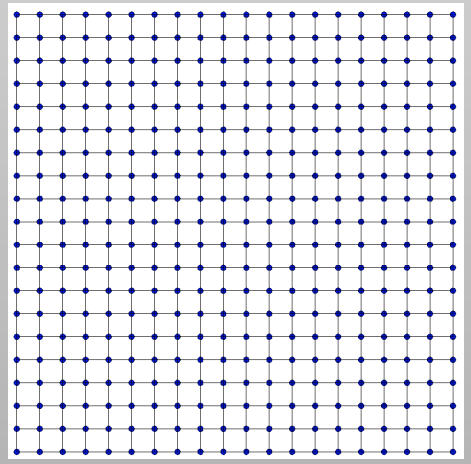


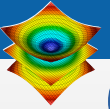










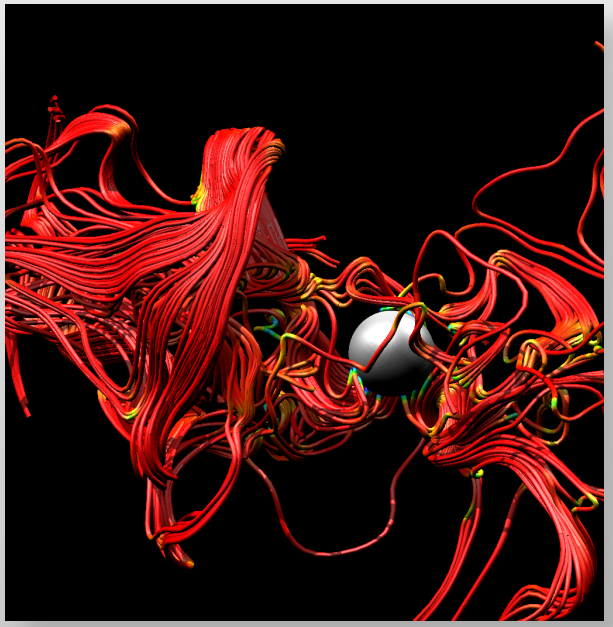


Create output

PICS:Filter::CreateOutput() = 0;

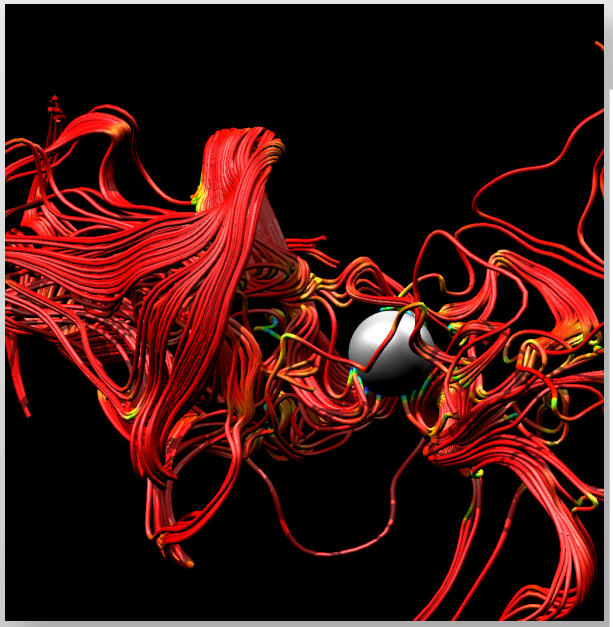
Create output

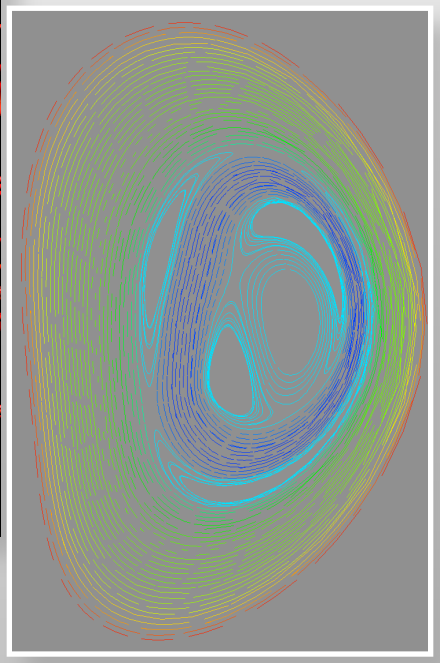
PICS:Filter::CreateOutput() = 0;



Create output

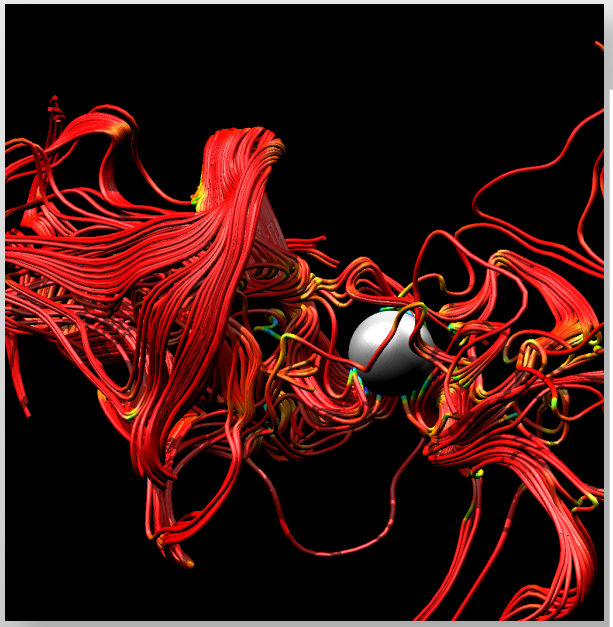
PICS:Filter::CreateOutput() = 0;

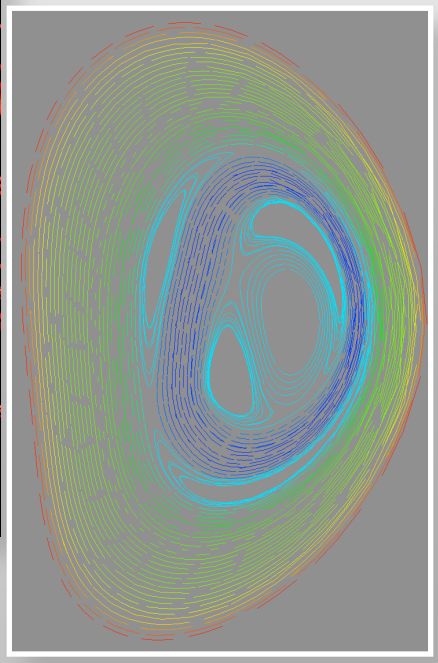


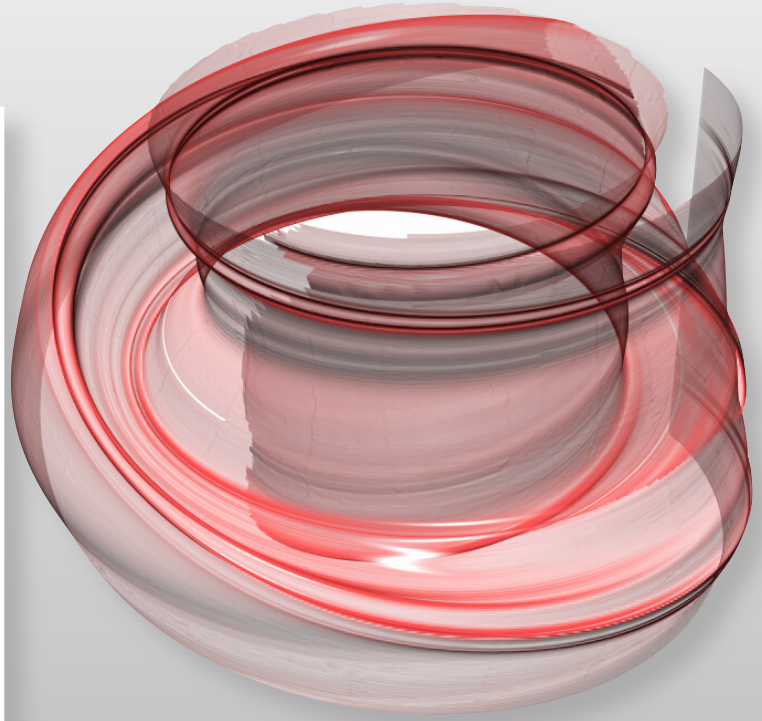


Create output

PICS:Filter::CreateOutput() = 0;

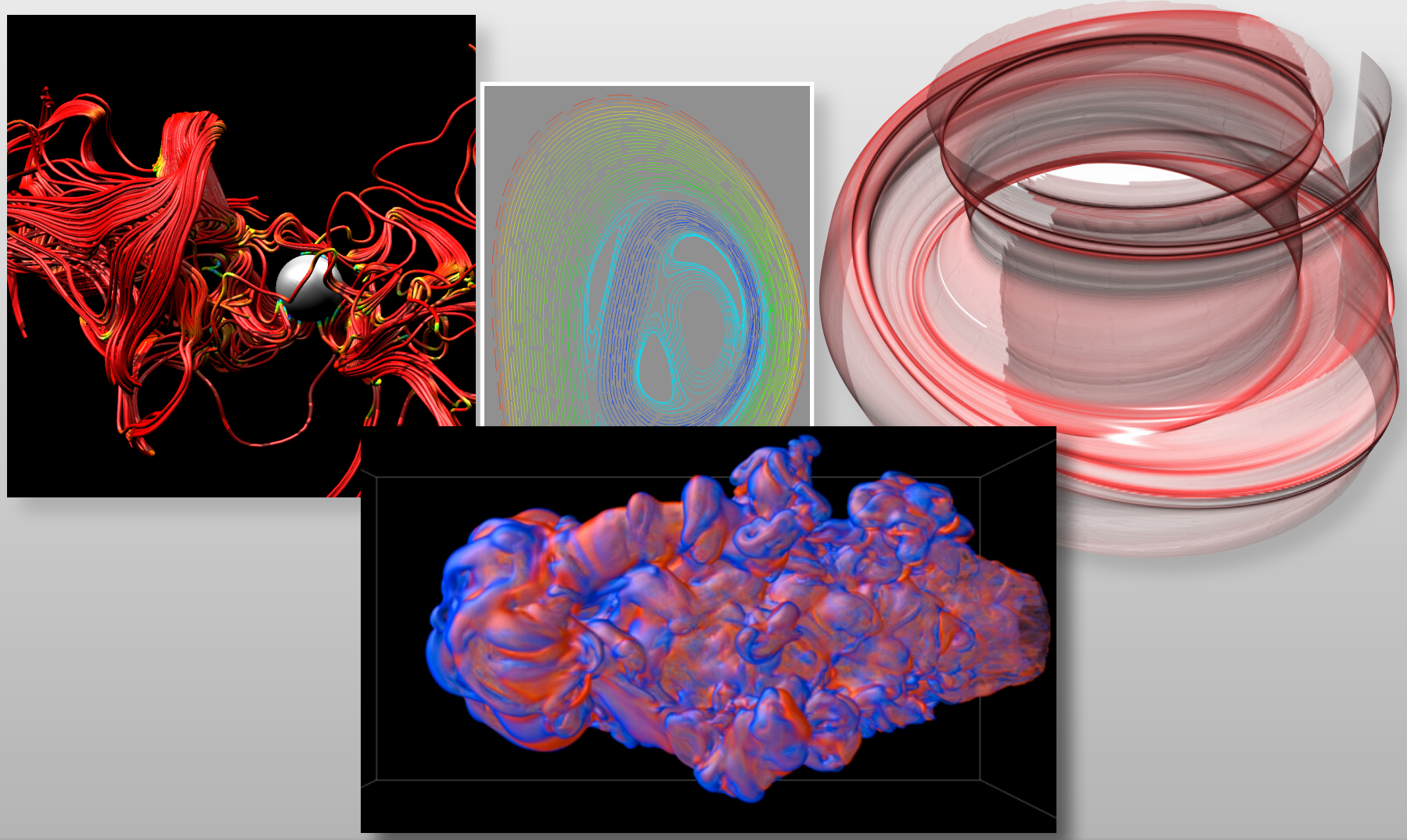






Create output

PICS:Filter::CreateOutput() = 0;



Parallelization