VectorFlow

A. Gheata

24th Geant4 Collaboration Meeting

23-27 September 2019

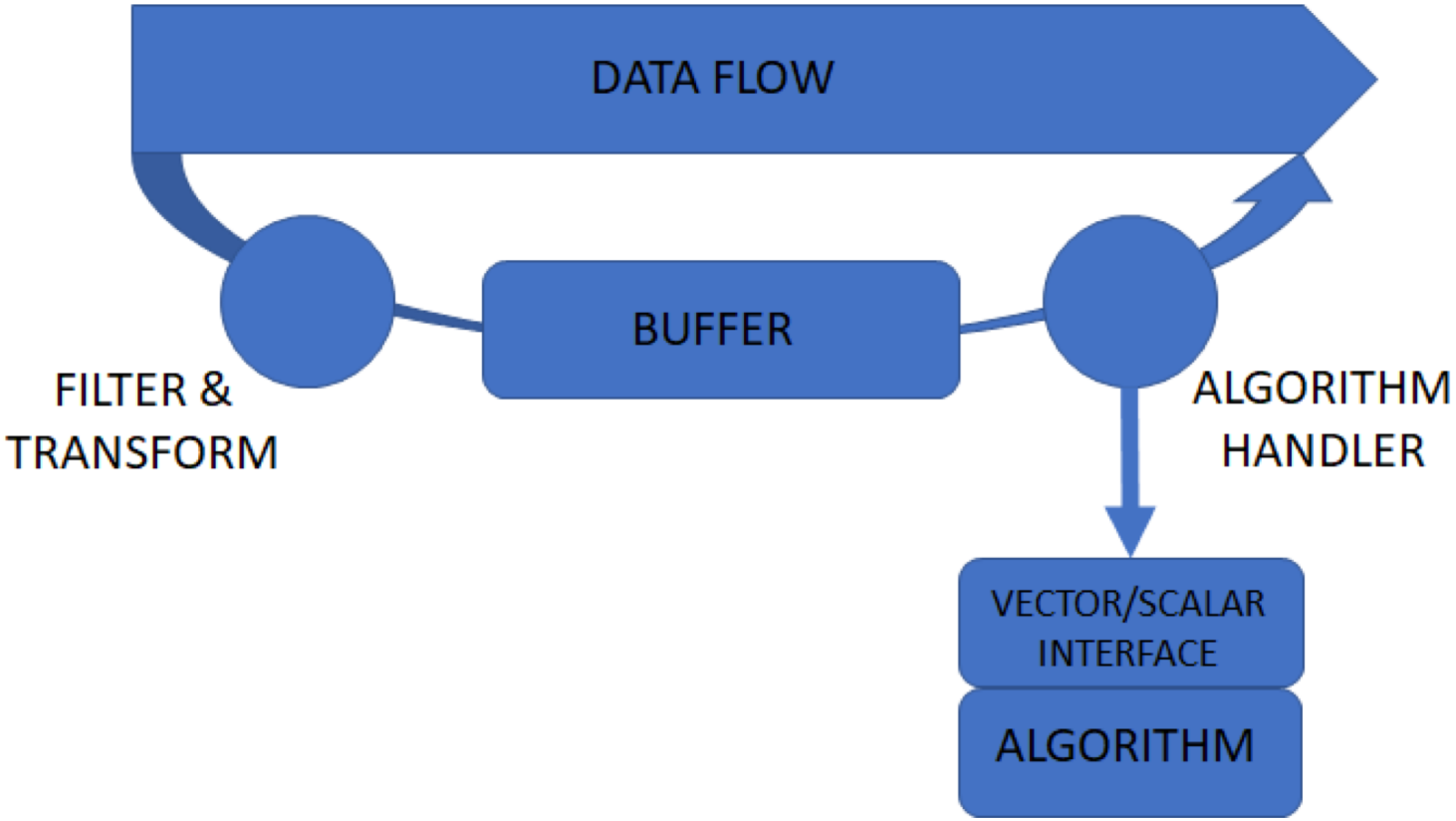
Jefferson Lab

The idea: vector components in a scalar workflow

A way to generalize vectorization by passing vectors of data to functions rather than rely on inner loops.

A vector adapter?

- Idea originating from GeantV workflow, but generalized as templated API usable in any workflow
- Do not provide an implementation, but rather a recipe and examples on how to do it
- Using VecCore as underlying vectorization library

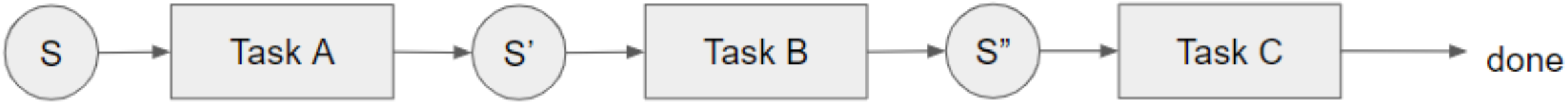


How it works

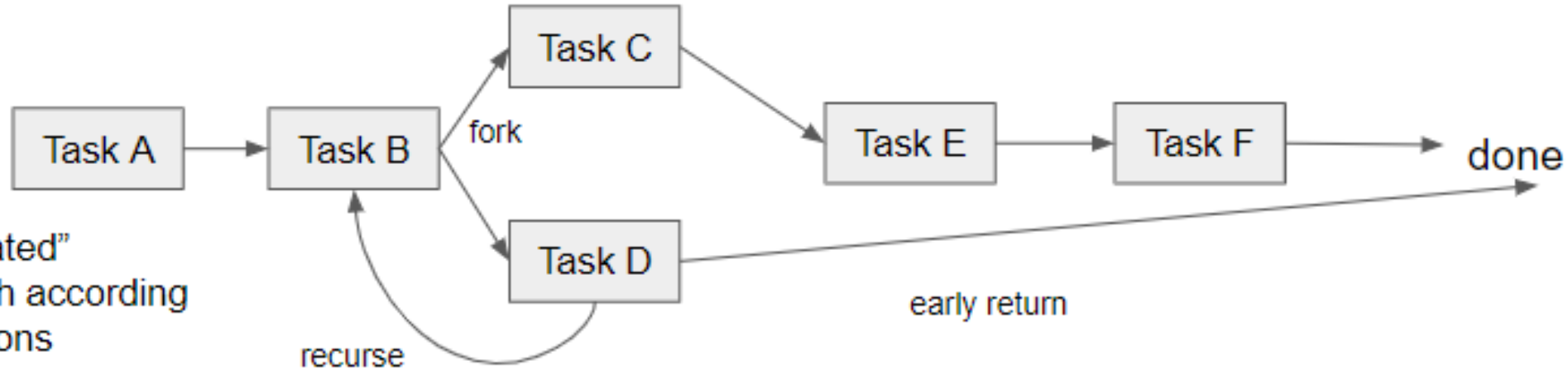
- A ***FILTER*** step to select scalar states to be processed.
- An ***ACCUMULATION*** step buffers the scalar states holding the data of interest
- A ***GATHER*** step in aligned memory to prepare the data needed for SIMD processing.
- A ***VECTORIZATION*** step, which integrates with VecCore services.
- A ***SCATTER*** step to re-integrate the output into the framework data flow.

Supported workflows

Pipeline flow:



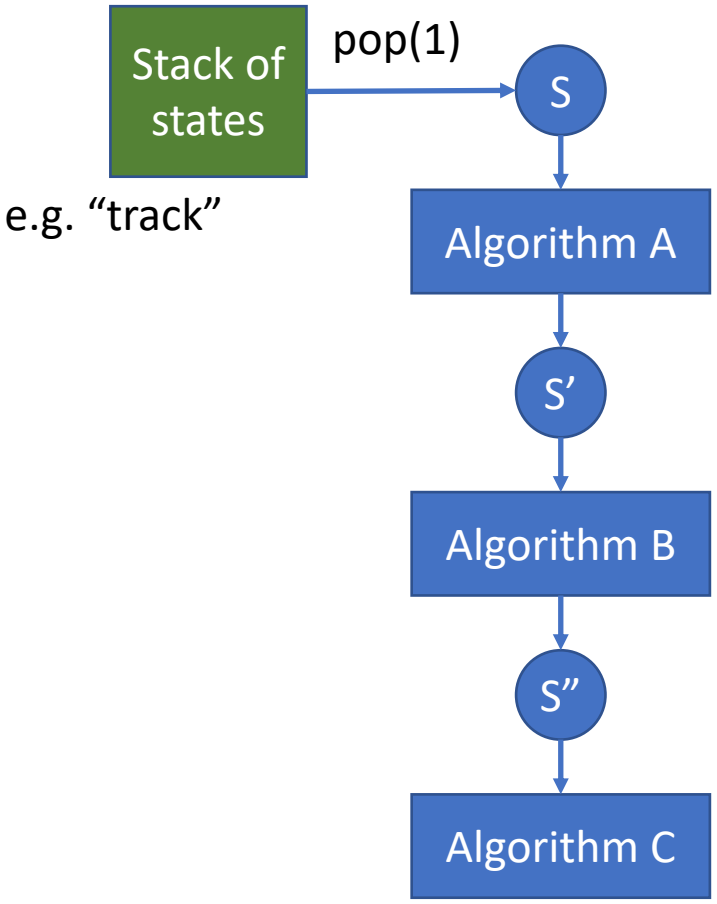
Complex flow:



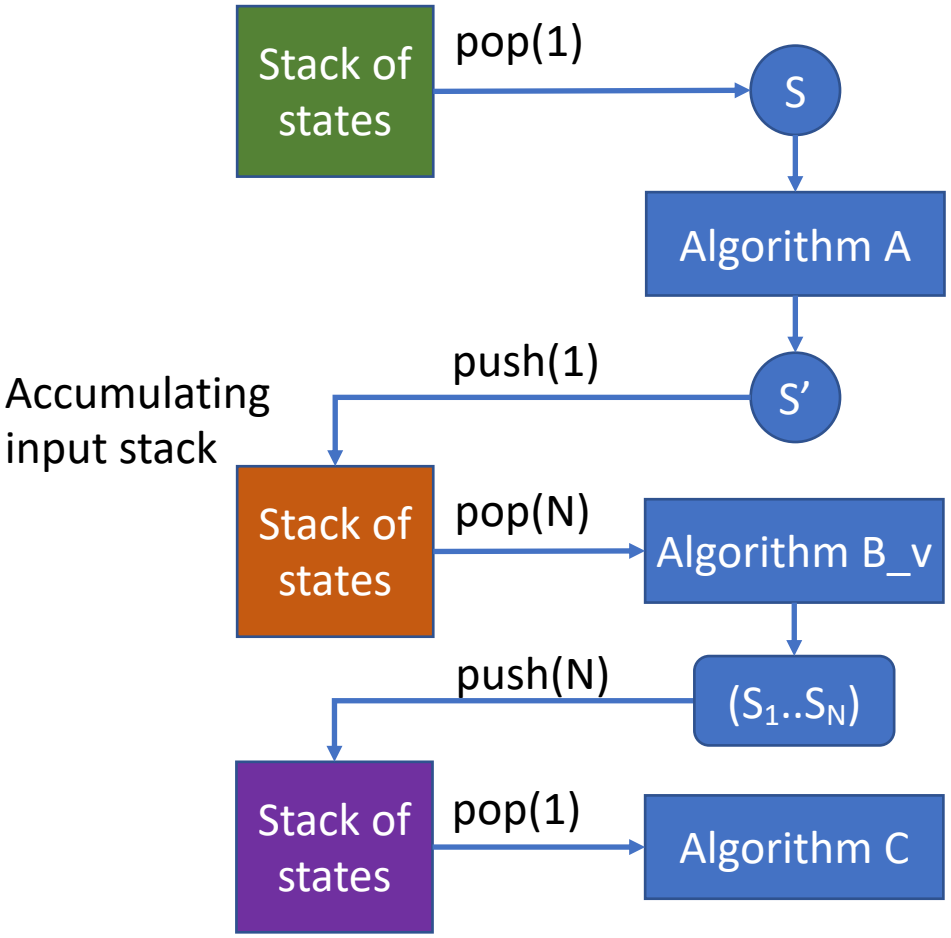
State is "propagated" through the graph according to runtime decisions

- Different flows can be inter-connected
- The data management introduces copy overheads, vectorization has to worth it

Flow re-integration



e.g. "track"



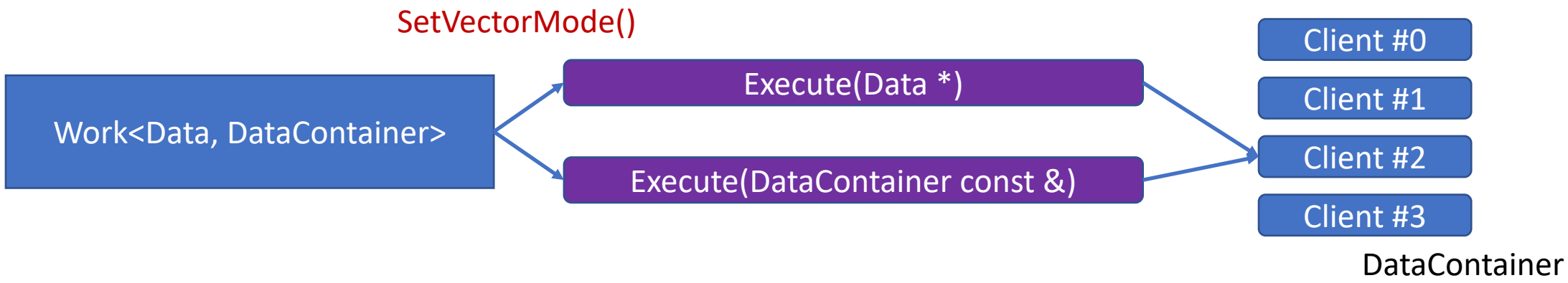
Pop also from "secondary" stacks

Implementation

- Implemented VectorFlow interfaces in the context of GSoC 2019
  - Student: Arturo Garza Rodriguez
  - Mentors: G. Amadio and A. Gheata
- New lightweight library now available: https://github.com/agheata/vectorflow
  - Simple design around the concept of “Work” that can have a scalar and a vectorized implementation
- **In particular the goal is to test this for components in the Geant4 workflow**

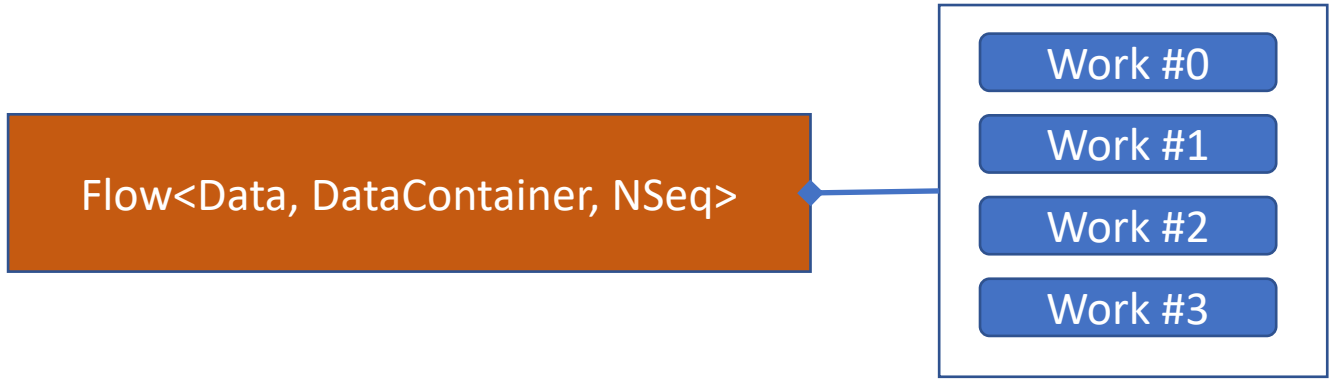
Work interface: Base class that provides interfaces to scalar and vector work.

- Interface to arbitrary user algorithm providing abstract interfaces for scalar and vector “Execute” methods
- Templated on user-defined data and container
- Work can have “clients” and and can dispatch the processed data to any of those

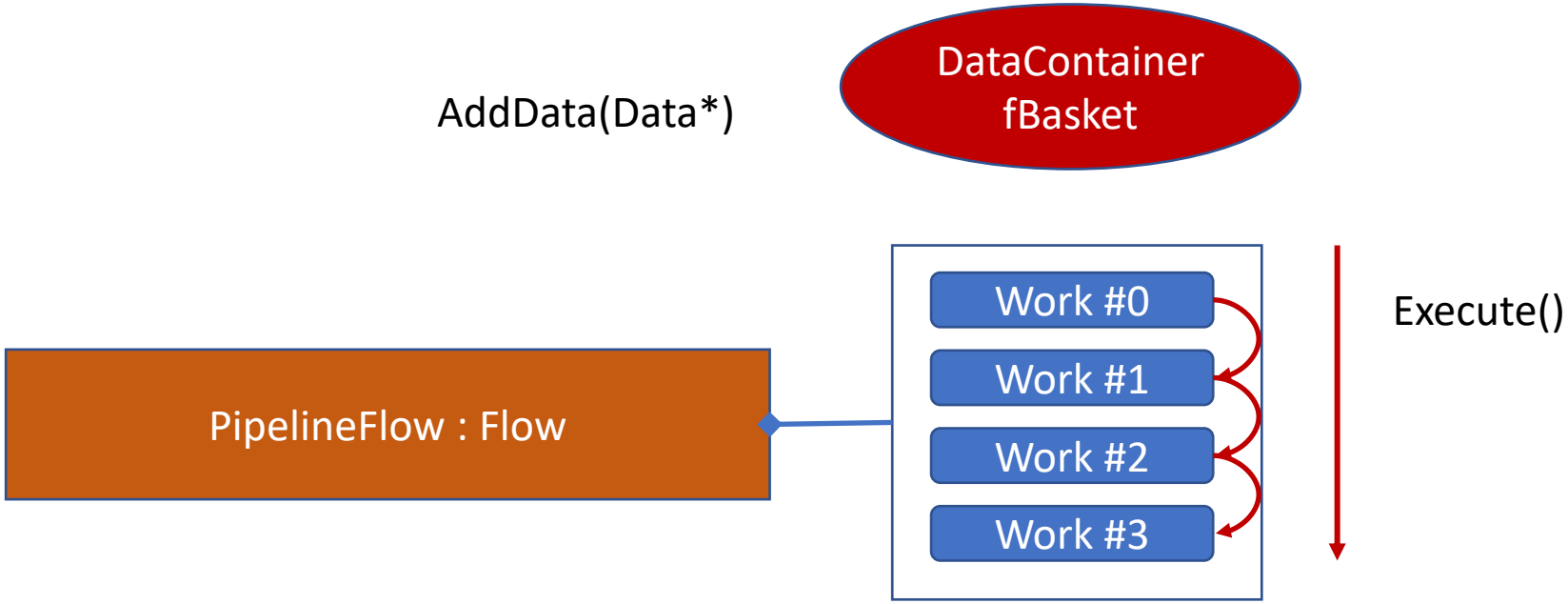


FLOW interface - Base class that provides support to the different types of flows

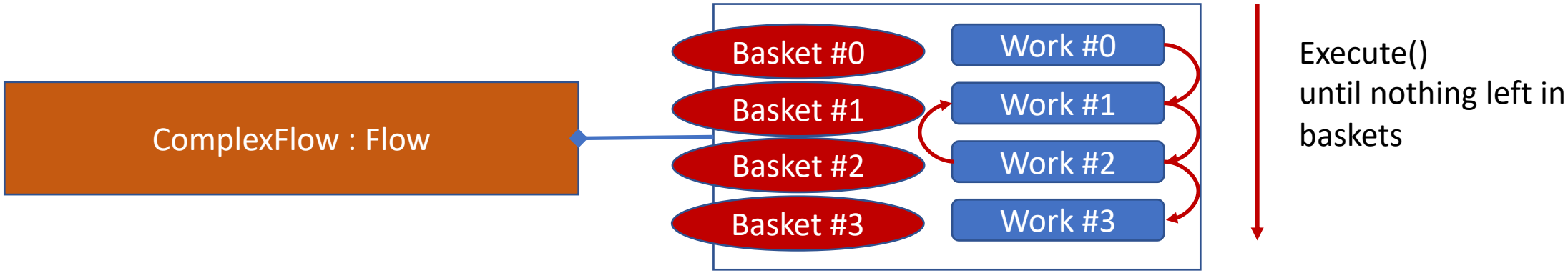
- Add each stage to the workflow via *AddWork*.
- Set each stage to be executed either in scalar or vector mode via *SetVectorMode*.
- Two flows to support any variant type of simulation workflows were proposed: the PIPELINE flow & the COMPLEX flow.



Pipeline flow

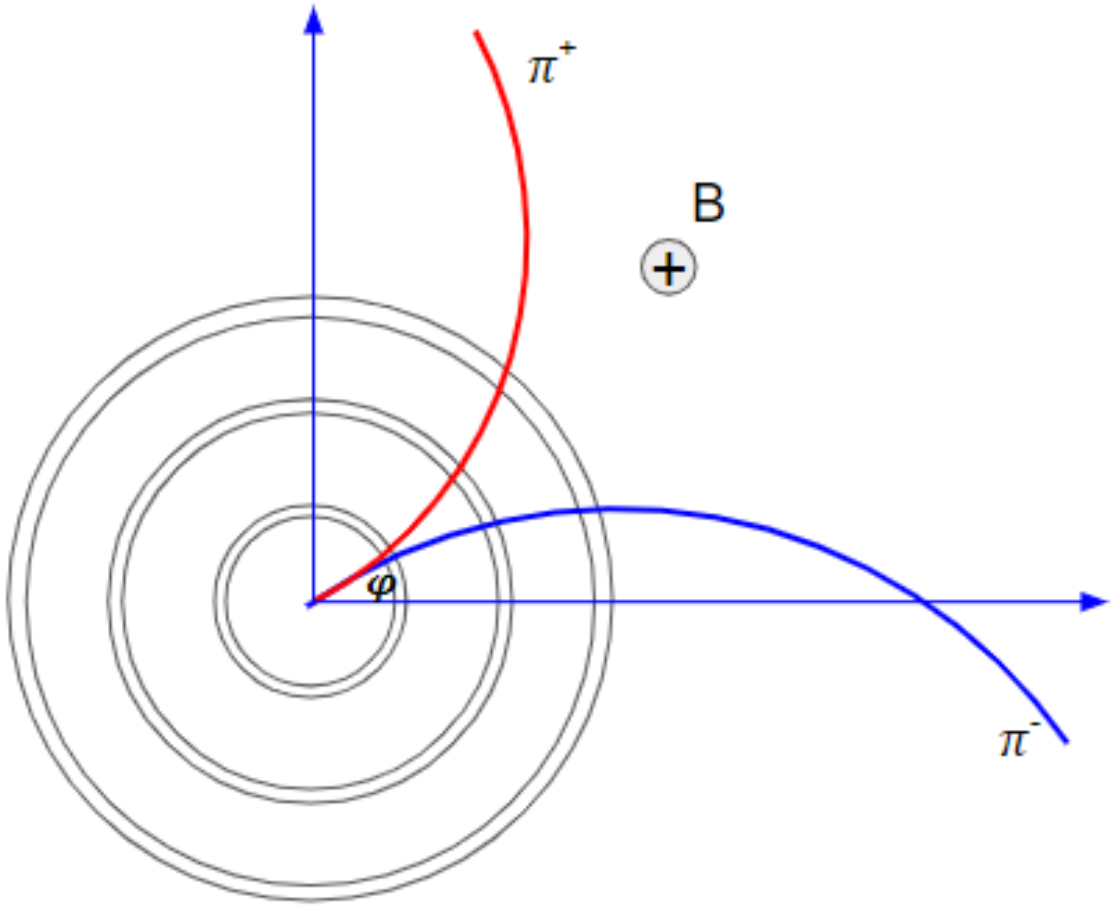


Complex flow



EXAMPLE

- COMPLEX FLOW: **Generate + Propagate**
- Each layer is translated to a **VectorFlow task** that propagates a **vector** of tracks (previously **gathered** in the correct format for SIMD processing).
- Each particle track can be propagated to an inner or outer layer in the **tube**, i.e. a task **dispatches** data to the other tasks' containers.
- If tracks are propagated outside the geometry, they are **scattered** back to the original data flow.
- The **flow** continues its execution until as long as there is still data in the buffers.



Up to ~2.0x speed-up in AVX2 processor, even though inherent overhead due to data transformations.

Prototyping VectorFlow with Geant4 components

- Requires stateless transport engine – ongoing work (see talk of WP)
  - Moving the state in the G4Track
    - Done already for managers (WP)
- Pause/resume for a given track w/o overhead
- Prototyping the vectorflow for the performance-critical components
  - Field propagation, MSC