MC event generation and analysis on HPC

Event simulation and analysis

- MC event generator + Rivet well established tool chain
- Rivet allows to quickly compare MC prediction with collider data
- This is important for
  - Validation of physics in MC event generator
  - Parameter scans of MC generator modelling/physics and optimisation w.r.t. collider data
- The workflow is mostly linear — MC simulates an event and Rivet analyses the event to fill histograms.
- Often high statistics accumulation necessary
- Traditionally done with batch-farms

DIY

- On HPC machines we want to exploit the parallel execution on many ranks to generate massive amounts of events as quickly as possible in a single program.
- Within Scidac we use the computing model of DIY.
- DIY takes care of all the MPI operations such that we can focus on writing flexlibe massively parallel programs.
- The atomic unit of DIY is called a block.
- In our application, each block contains an event generation object (Pythia8) and an analysis object (Rivet).
- A few foreach calls to functions operating on single blocks are sufficient.

```
// Set physics parameters
master.foreach([world, physics](Block* b, const diy::Master::ProxyWithLink& cp)
        {set_physics(b, cp, physics); });

// This runs the generator and rivet
master.foreach([world](Block* b, const diy::Master::ProxyWithLink& cp)
        {process_block(b, cp); });

// Reduction step --- sum histograms of all blocks
diy::reduce(master, assigner, partners, &reduceData<Block>);

// This is where the write out happens
master.foreach([world](Block* b, const diy::Master::ProxyWithLink& cp)
        { write_out(b, cp); });
```

Program sketch

- We run with 1 block per rank.
- With DIY we can efficiently run thousands of physics configurations with millions of simulated events each.