# MERLIN computation needs for the HL-LHC upgrade.

J. Molson

University of Manchester, Cockcroft Institute

16th January, 2013

# What is Merlin?

- C++ Accelerator physics library
- Provides a set of useful functions for accelerator modelling
- Initially used to simulate ground motion in the ILC BDS and linac
- Later the ILC damping rings
- Written by Nick Walker et al (DESY)
- Now adapted for large scale proton collimation simulations by Manchester and Huddersfield
- Three main sections of the library:
- Accelerator lattice loading/creation and storage
- Tracker
- Physics processes
- Modular design - easy to modify and extend

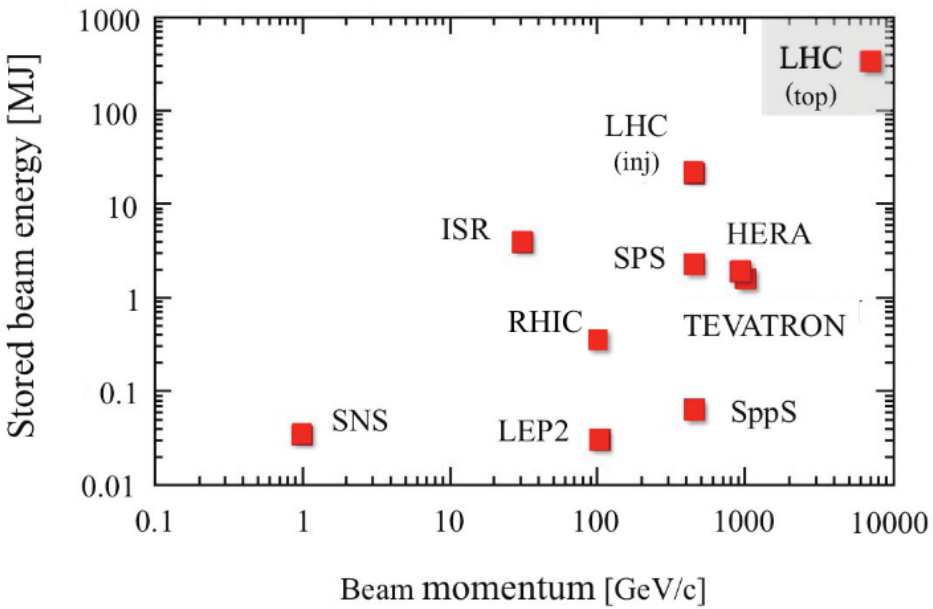
# Physics processes

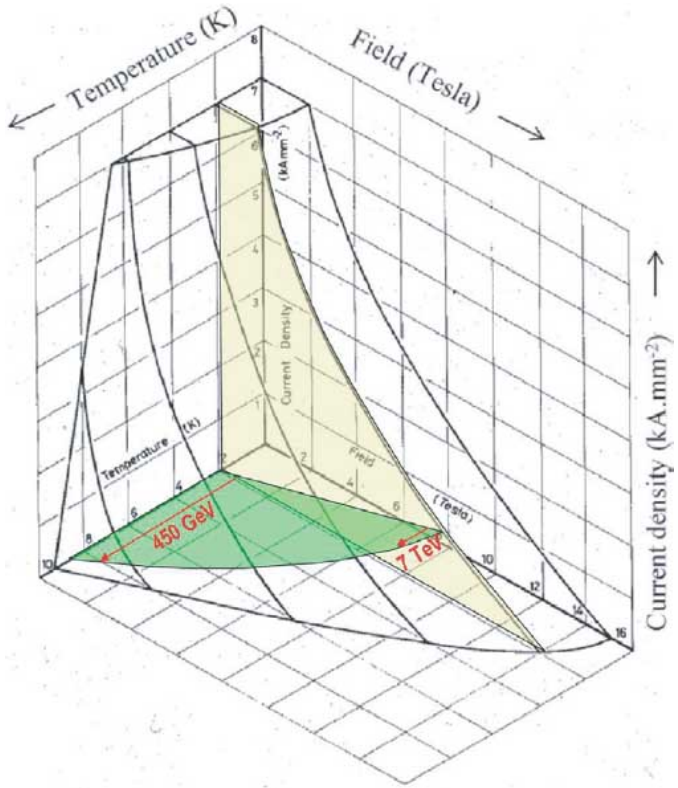
- Additional physics on top of tracking to be applied at selected elements and positions
- Can be enabled or disabled as required - processes are attached to trackers
- Examples: Synchrotron radiation, collimation, wakefields, etc
- Easy to create, template examples exist
- Trackers manage stepping within processes - inputs are the AcceleratorComponent and bunch

# The Large Hadron Collider (LHC)

- 7 TeV proton-proton synchrotron, 26.65km length
- Beams collide at 4 experimental regions - (ATLAS, ALICE, CMS, LHCb)
- 2 collimation regions
- Additional regions for RF, and the beam dump
- Injection at 450 GeV, ramp to up 7000 GeV (Currently running at 4000 GeV)
- Superconducting magnet system, 1.9K, 8.33T dipoles
- High stored beam energy!

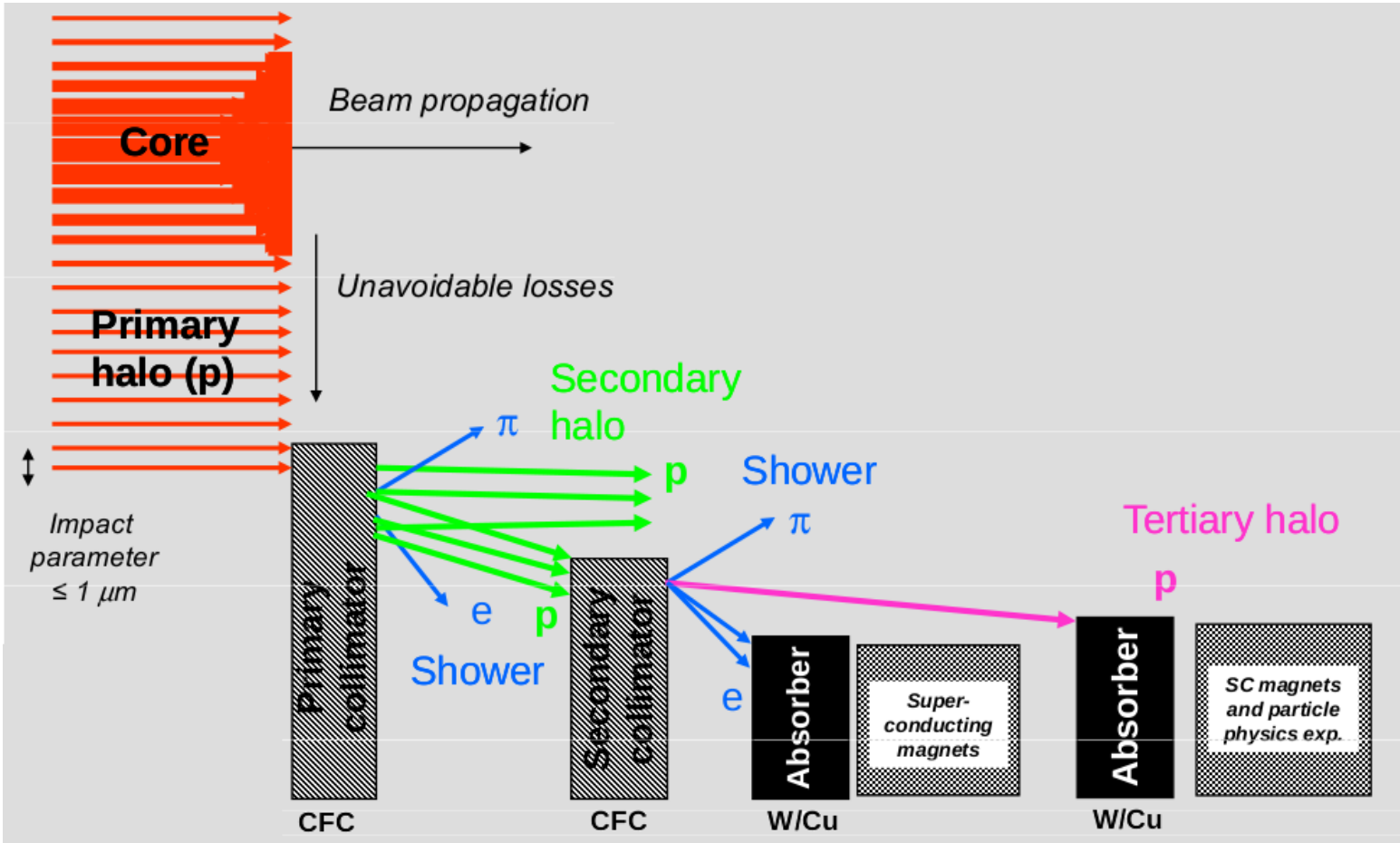
# Why do we need to collimate



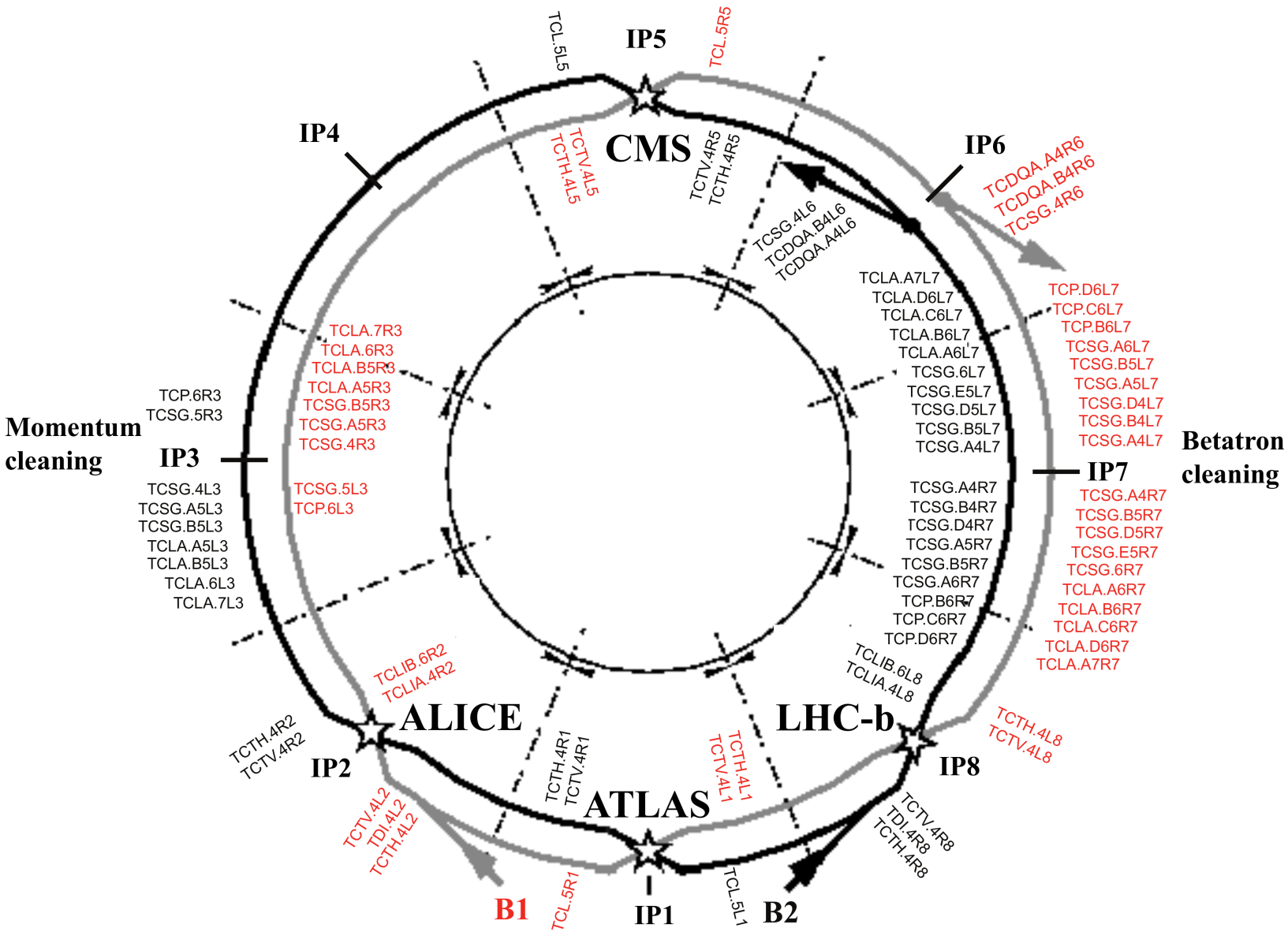




- 360MJ stored beam energy.
- $4.5 mW/cm^3$ will quench a magnet at top energy!

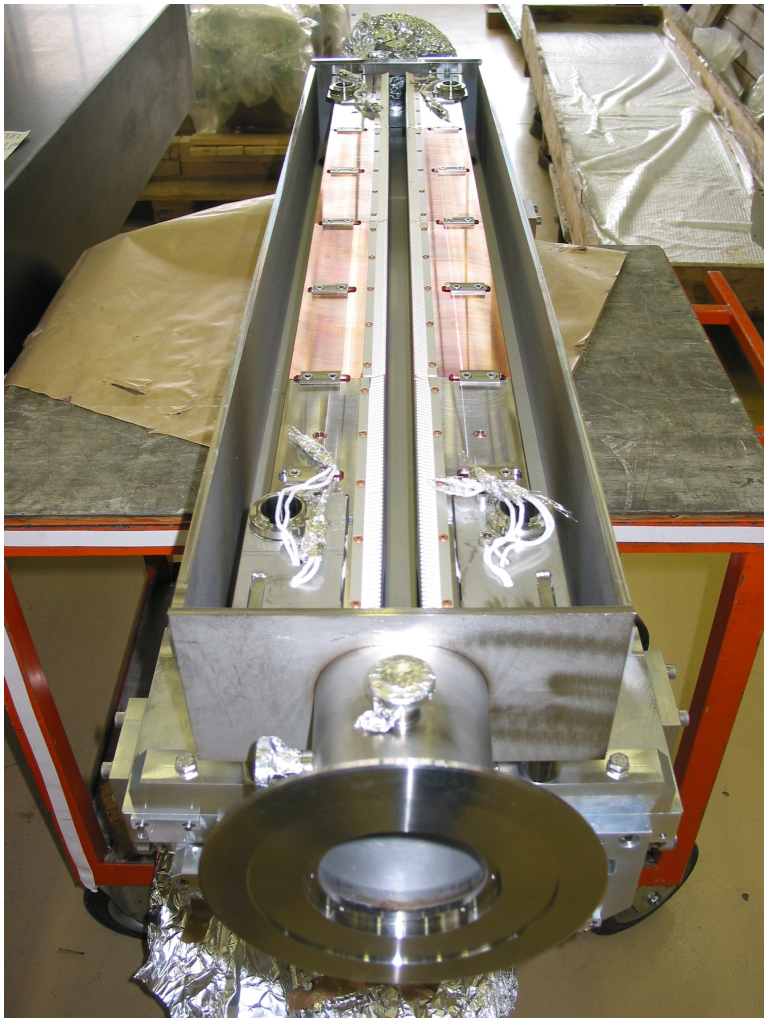
# Collimation



# Collimation Layout



# Collimator Images



# Simulation running

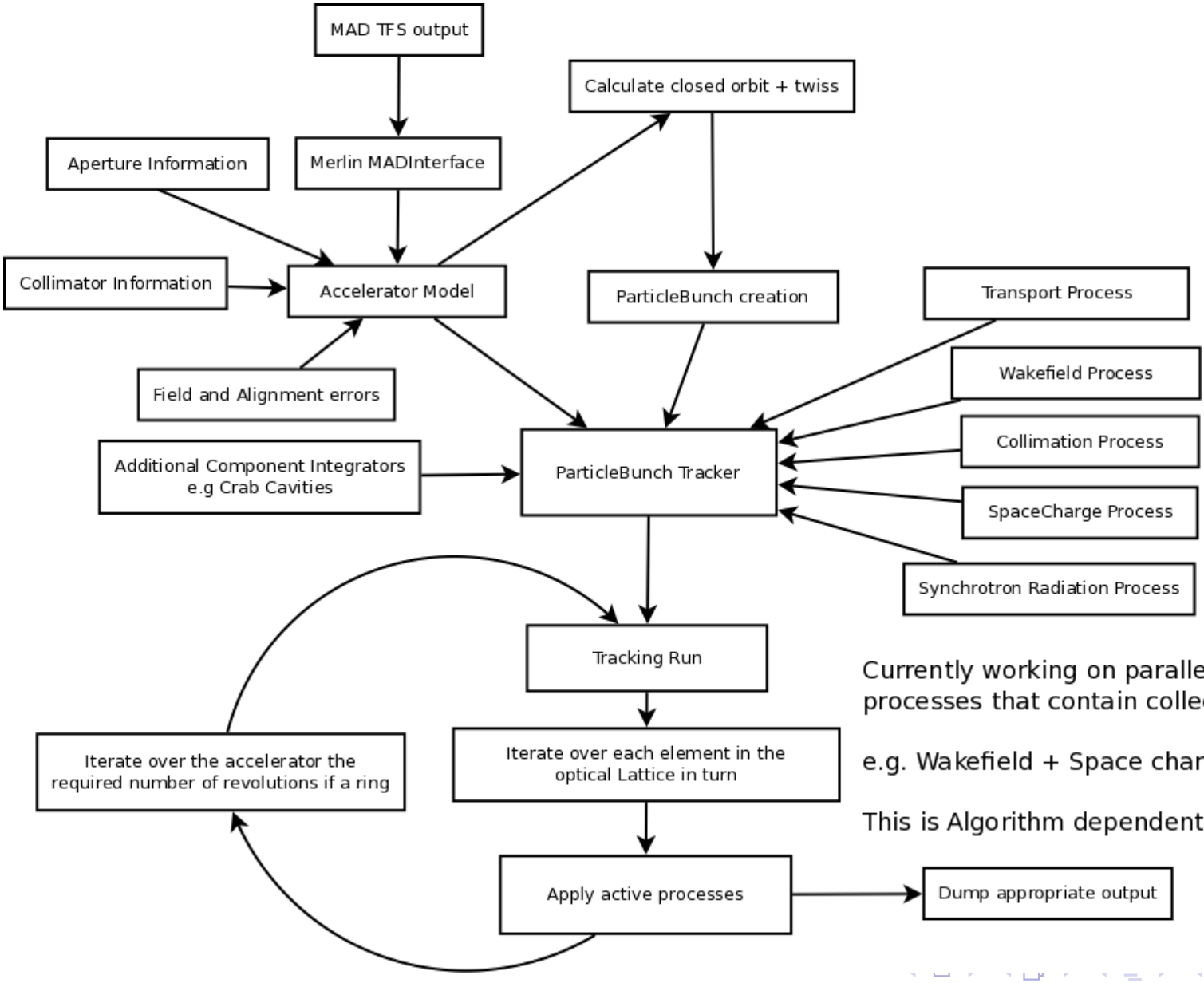
- Tracking and collimation is independent on a per-particle basis, so do not need any parallel computers - just lots of CPU hours.
- Currently run on grid systems and lxbatch in addition to local machines.
- Typical run involves $\sim$1000 cores for $\sim$hours - we want to simulate billions of particles.
- Want to probe down to low loss levels - e.g. to find possible areas that could suffer from radiation damage.
- And then re-run on different optics and collimator configurations.
- Simulation will expand to fill all available computing resources.

# OpenMP

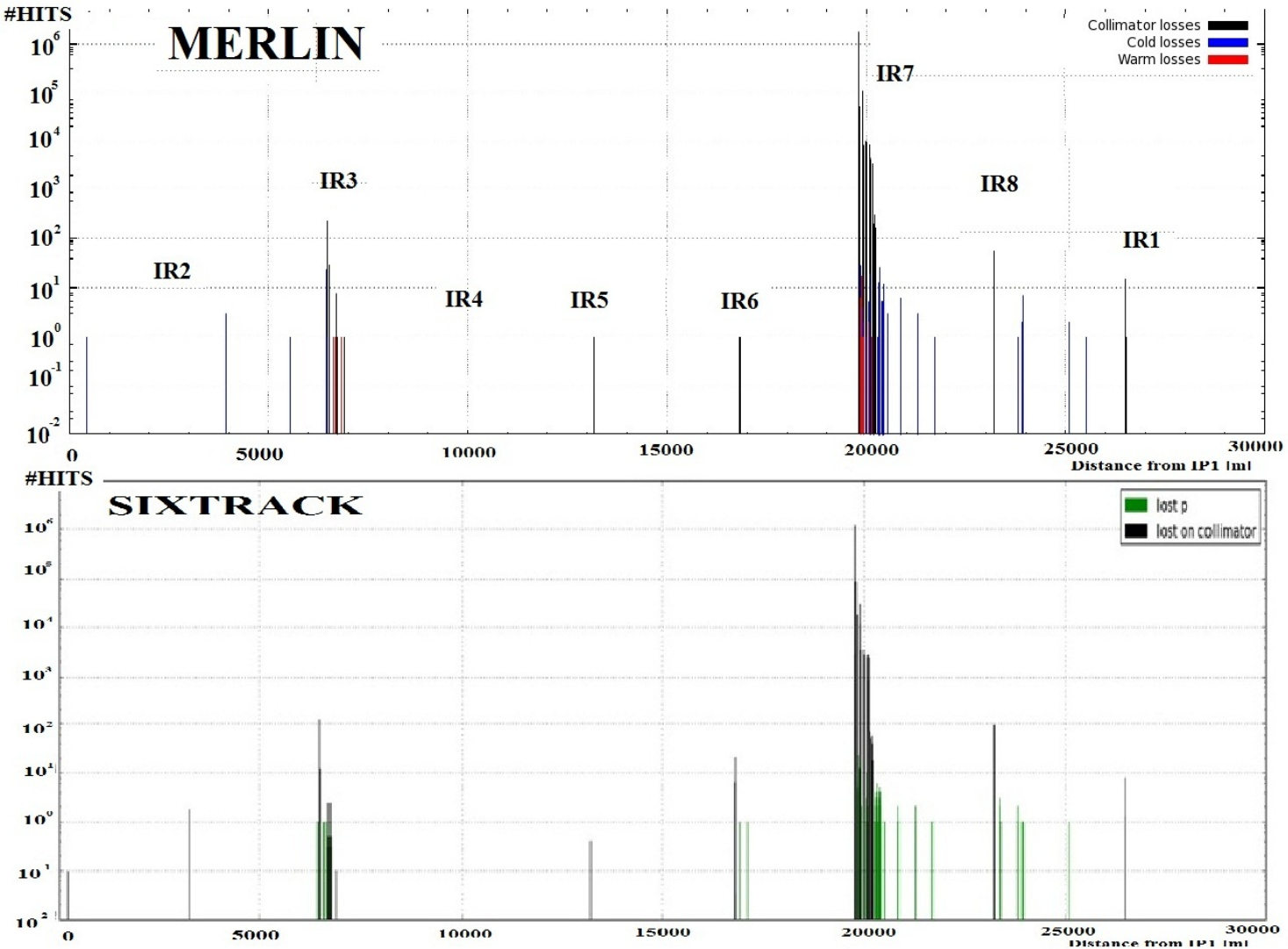
- Minimal work can be done to get this to run in "parallel".
- Use OpenMP in a loop over all particles in the bunch.

```
#pragma omp parallel for
for(size_t i = 0; i<bunch.size(); i++)
{
amap->Apply(bunch.GetParticles()[i]);
}
```

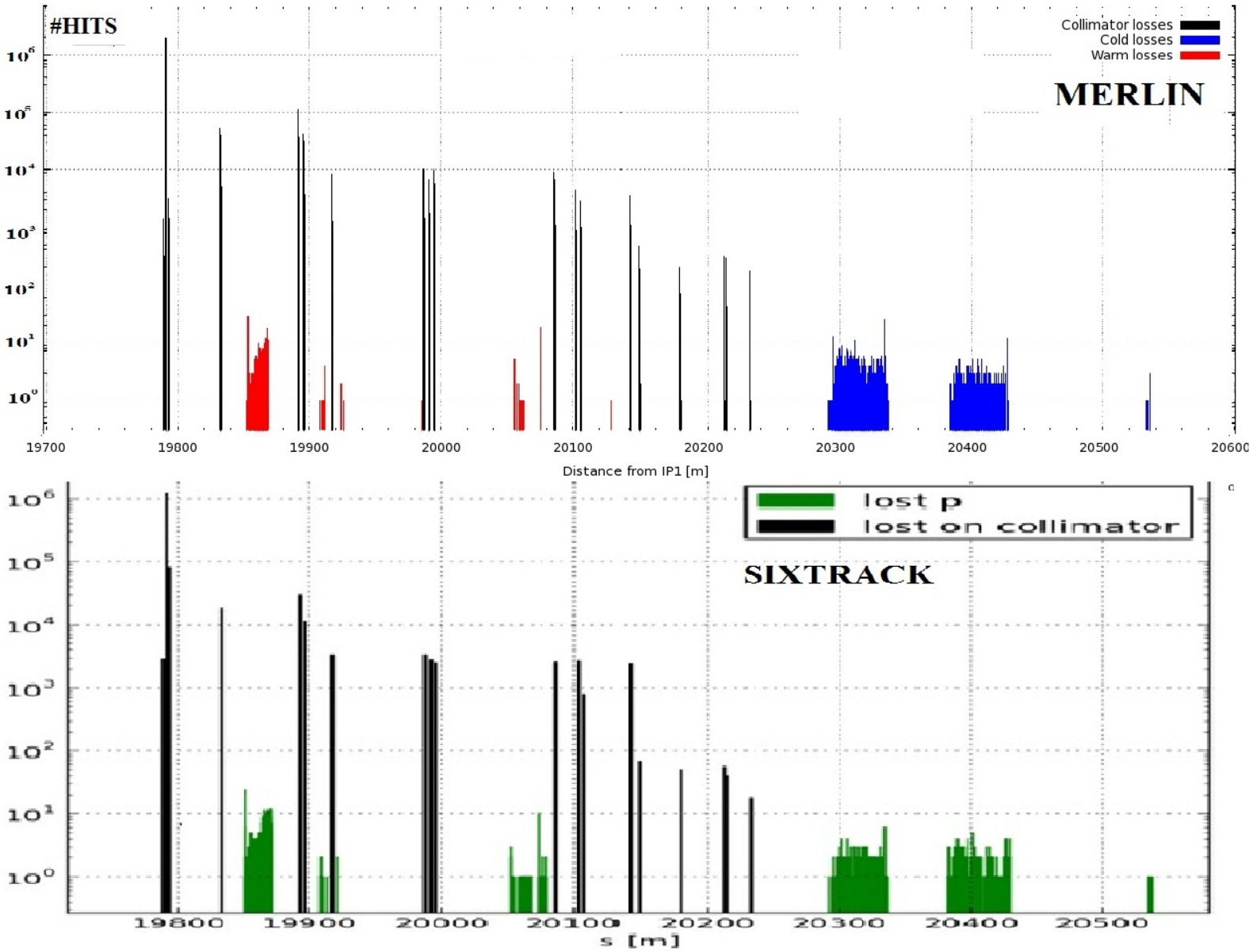
# Example run


MAD TFS output
Calculate closed orbit + twiss
Aperture Information
Merlin MADInterface
Collimator Information
Accelerator Model
ParticleBunch creation
Transport Process
Wakefield Process
Field and Alignment errors
Collimation Process
Additional Component Integrators
e.g Crab Cavities
ParticleBunch Tracker
SpaceCharge Process
Synchrotron Radiation Process
Tracking Run
Currently working on parallelising processes that contain collective effects
Iterate over the accelerator the required number of revolutions if a ring
Iterate over each element in the optical Lattice in turn
e.g. Wakefield + Space charge effects
This is Algorithm dependent.
Apply active processes
Dump appropriate output

# Loss map results comparison (Sixtrack plots from LHC collimation group (R. Bruce))



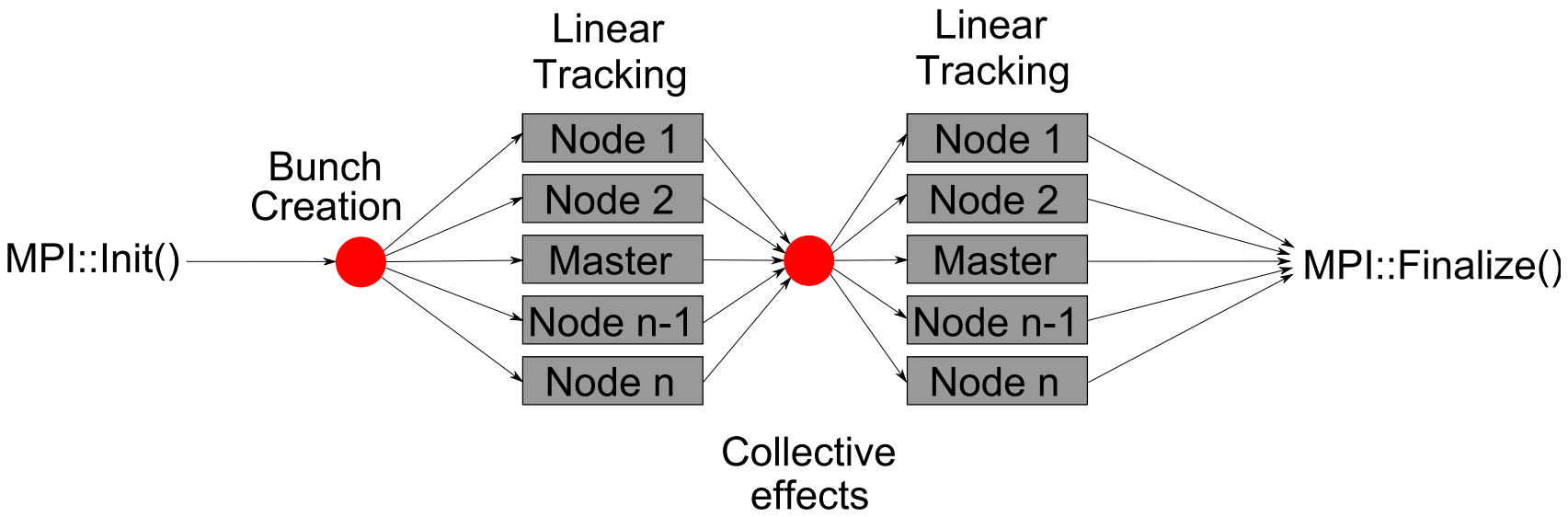
# Loss map Results IR7



# Parallel running

- Wish to run large simulations - very cpu heavy - use MPI
- Must use multiple physical machines with interconnects
- Run multiple copies of the same binary that can communicate with each other
- Tracking, collimation, etc, are all independent on a per-particle basis, do not need any knowledge about other particles
- Collective effects such as space charge and wakefields do require this information
- Functions exist such as parallel bunch moment calculations (mean, standard deviation) in addition to the ability to move particles between computers
- Parallel running is implemented at a per process algorithm level

# Parallel running



# Naive example

Given some variable x, we wish to calculate the mean: $\bar{x} = \frac{1}{n} \sum_{n=1}^{n} x$

- Sum all the x values on a single process.
- Share the sum values between all processes.
- Share the number of x values per process between all processes (n).
- Sum all x values and sum all n values.
- Divide to get the mean.

# Psedocode

```
//Some array of values
doubleInputArray[NMAX];
double sum = 0;
for(size_t i = 0; i < NMAX; i++)
{
sum += InputArray[i];
}

Allreduce(MPI_IN_PLACE, &sum, 1, MPI_DOUBLE, MPI_SUM);
Allreduce(MPI_IN_PLACE, &NMAX, 1, MPI_DOUBLE, MPI_SUM);

double mean = sum/NMAX:
```

# Example: collimator resistive wall wakes

- Collimator jaws are placed very close to the beam ($\leq 5\sigma$).
- Resistive wall effect leads to beam emittance growth.
- Good conducting materials aren't always good for collimation.
- Want new collimation layouts that can use different "novel" materials that allow collimators to sit further from the beam core.

# Algorithm

- Slice the bunch longitudinally into n slices - each node must agree on where to do the slicing from the mean and standard deviation of the bunch distribution.
- Calculate the kicks on each node due to the charge contribution from each slice.
- Sum the kicks from each slice over the full simulation and distribute to all nodes.
- Apply the wakefield using the distributed contributions.

# Conclusions

- We are developing the code Merlin to operate with proton machines for high energy collimation simulations.
- Simulations of the HL-LHC upgrade at CERN are progressing well.
- Different physics problems require different computing methods.
- We will always need more computing power.
- Would like to have access to a xeon phi to test.
- Always looking for new victims/code users.