


- 
- GPU Cherenkov -
 - Process Sequence -
- and others

GPUs Why?



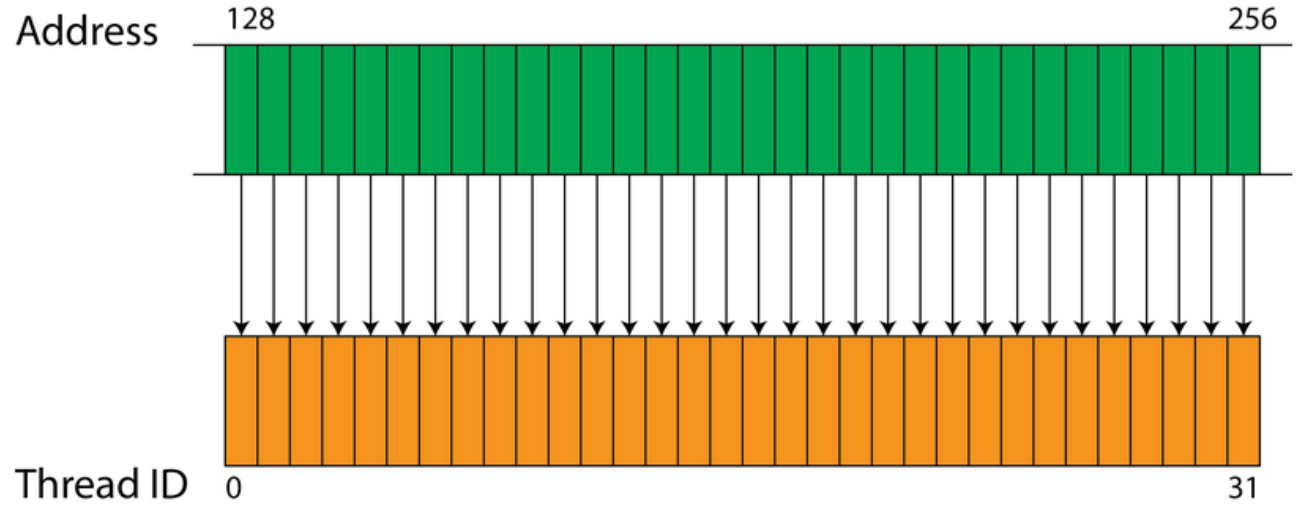
A100 108x SMS



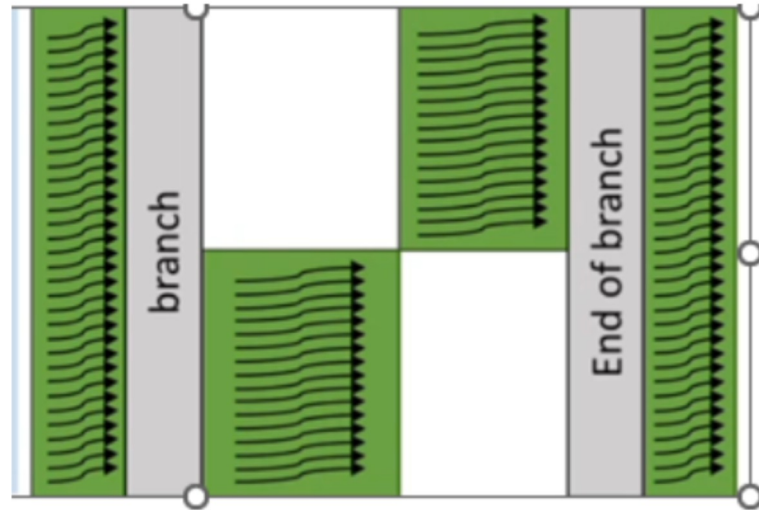
132x SMS H100

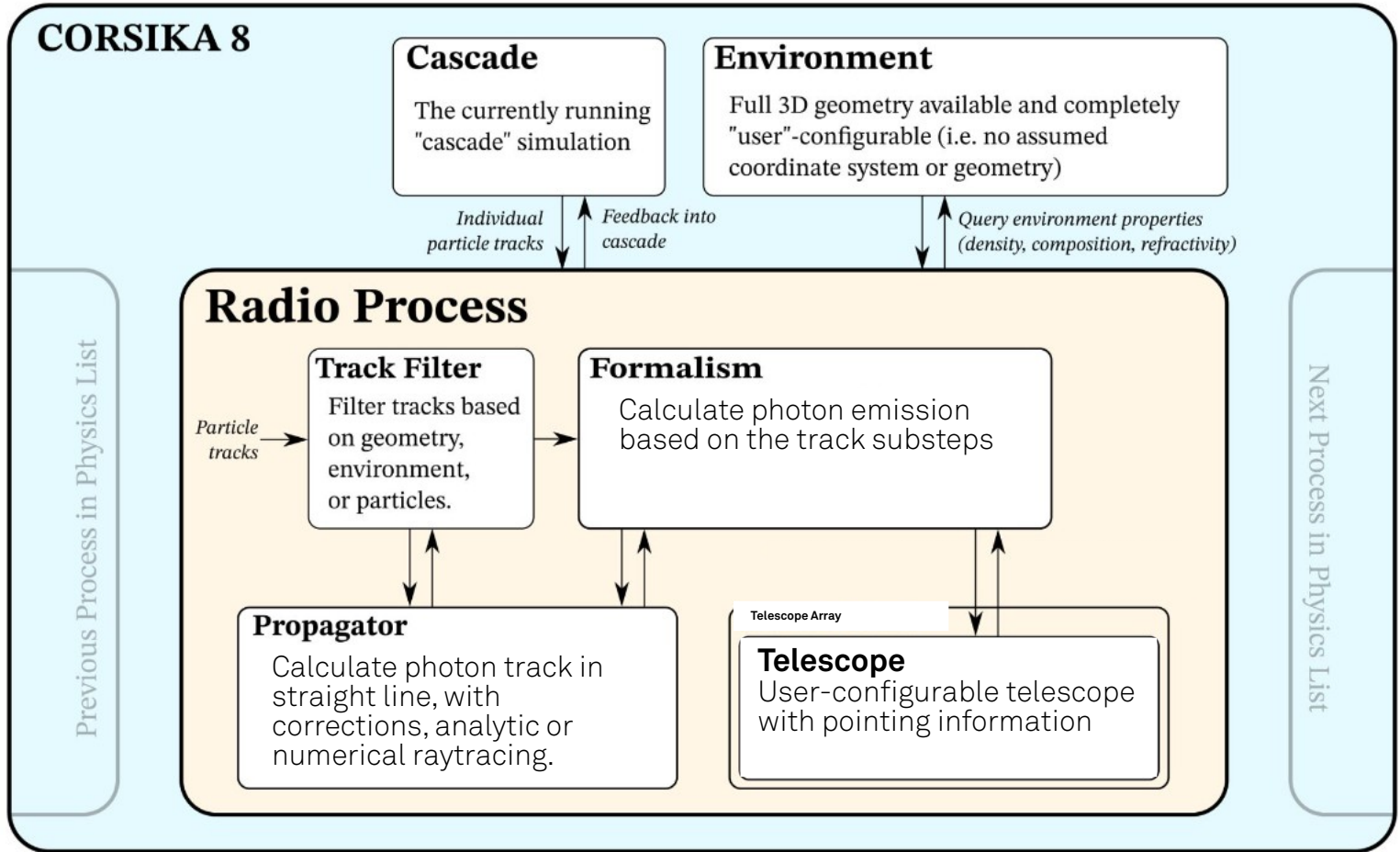
GPUs How?

Memory Coalescing



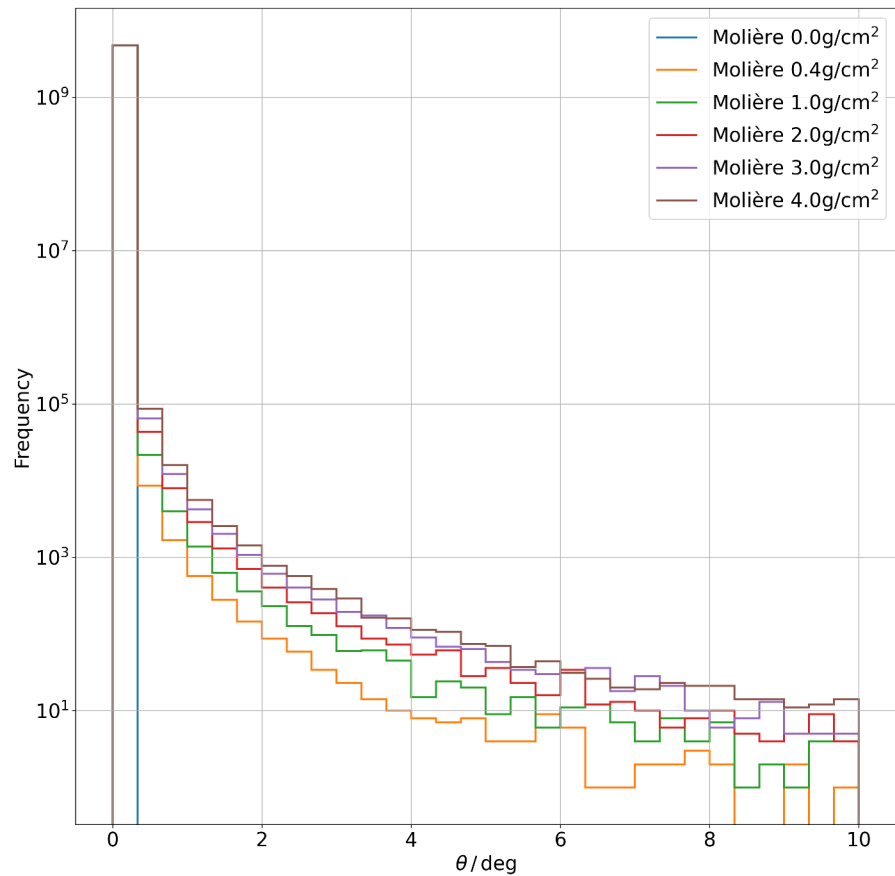
Warp Divergence





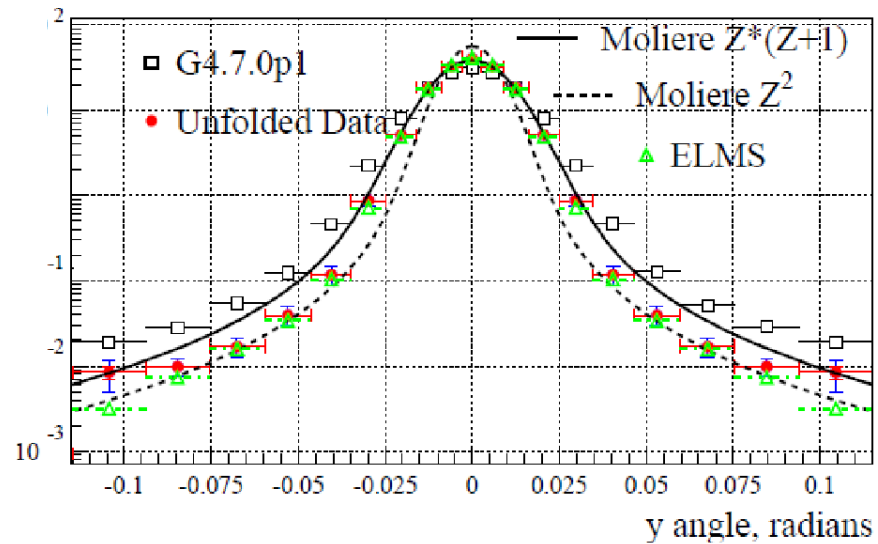
- 1) Collect particle substeps (linear traces) and reduce to coordinate, direction, velocity and time
- 2) Transfer simplified particles to GPU
- 3) At least 10 particles per AU → 320 per Warp to allow for effective latency hiding
- 4) Store surviving particles in local shared memory (locally & fast), go back to 3) if number is small Avoids warp divergence
- 5) Generate photons (position, direction, Wavelength, Time) and write to memory
- 6) “Iterate” through photons and calculate straight line impact on horizontal plan, afterwards apply correction. Clip with array boundary's and store in shared memory.
- 7) 1 Telescope per warp, N Photons → Check for hit and store in global memory for download on host machine

Substepping



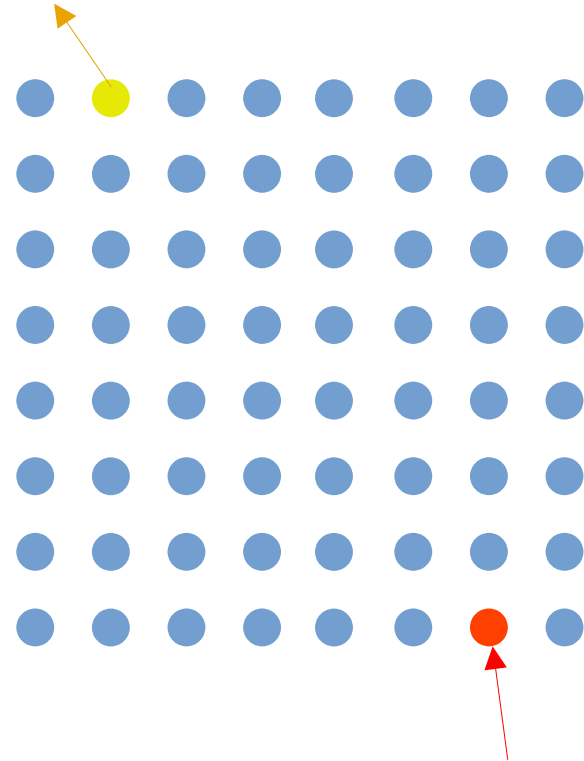
$$\vec{F} = q\vec{E} + q\vec{v} \times \vec{B}$$

Electric force
Magnetic force



Substepping Idea

- Weight edges with probability / $(1-p)$, take path with highest/lowest probability for path interpolation.
Alternative: Potential field and gradient-descent
- Sample random path or optimal path?



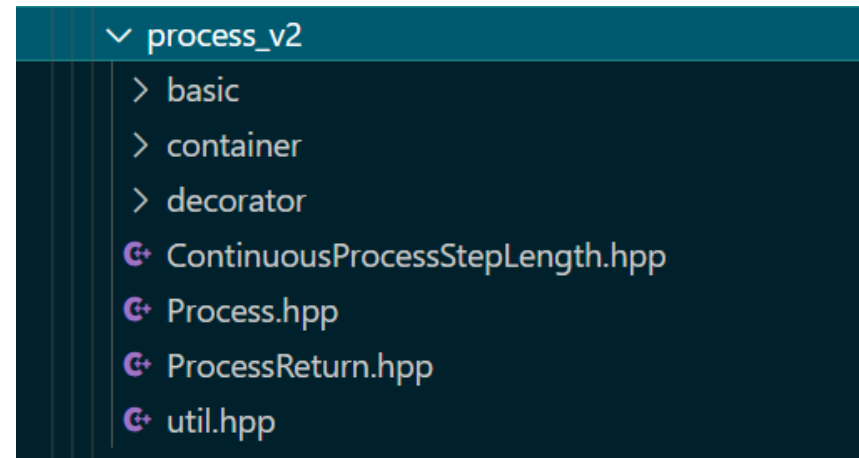


Now something different

Process Sequence V2

Changes

- Clearer Structure and divide between base classes, storage classes and modifications
- Removed several levels of templates and meta templates



Changes

```
You, last month | 1 author (You)
namespace corsika {
    You, last month | 1 author (You)
    template <ConceptProcess... TProcesses>
    class ProcessSequence;

    You, last month | 1 author (You)
    template <ConceptProcess TProcess, ConceptProcess... TSequence>
    class ProcessSequence<TProcess, TSequence...> : public ProcessSequence<TSequence...> {
    private:
        using process_type = typename std::decay_t<TProcess>;
        using sequence_type = ProcessSequence<TSequence...>;

        TProcess process_;
    };
};
```

- Clearer Structure and divide between base classes, storage classes and modifications
- Removed several levels of templates and meta templates
- ProcessSequence is now a variadic template, which removes code duplication

- All currently used templated arguments are easility known by the stack → move over to shared class template Tstack and derive information's from it
This allows the user of virtual functions and make modules callable from outside!
- For this Stack rework
- Cleanup unused/dead code (separate issues #595)
- Change interfaces to

```
You, 5 days ago | 1 author (You)
struct DecayProcess : virtual BaseProcess<DecayProcess> {
public:
    template <typename TView>
    void doDecay(TView& view);

    template <typename TParticle>
    TimeType getLifetime(TParticle& particle);
```

Stack V1

- Currently Stack and data to store are heavily interlinked
- Data is stored by information → Array of positions, Array or direction, ... (Unconfirmed) Caching problem, every information require separate load
- Information stored and information required by modules are completely separate, not meeting requirements fails with loooooong template errors.

Stack V2

- Separation between functionality and Data
- Data should be driven by selected modules e.g. automatically include history information if required

No LTO / IPO (Link Time / Inter procedural Optimization) → ODR (One Definition Rule) issues not caught → ABI (Application Binary Interface) issues not caught

```
cmake_minimum_required(VERSION 3.9.4)

include(CheckIPOSupported)
check_ipo_supported(RESULT supported OUTPUT error)

add_executable(example Example.cpp)

if( supported )
    message(STATUS "IPO / LTO enabled")
    set_property(TARGET example PROPERTY INTERPROCEDURAL_OPTIMIZATION TRUE)
else()
    message(STATUS "IPO / LTO not supported: <${error}>")
endif()
```

Function Parameter order arbitrary

- clang-tidy bugprone-easily-swappable-parameters

```
struct Range {
    TType min;
    TType max;
} range_;

public:

// Construct with {.min=, .max=}
RangeCheck(const Range range)
    : range_(range) {}
```

```
module_control.addOption(
    control::ControlOption<double>("test3", "test4", 1.0)
        .setConstraint(control::checks::RangeCheck<double>({.min=0.0, .max=1.0}))
        .setConstraint(control::checks::NeedsCheck<double>(option1));
```

But C++ 20