Worspaces: Extending G4MT for task parallelism

John Apostolakis, Andrea Dotti

Andrea Dotti (andrea.dotti@cern.ch)

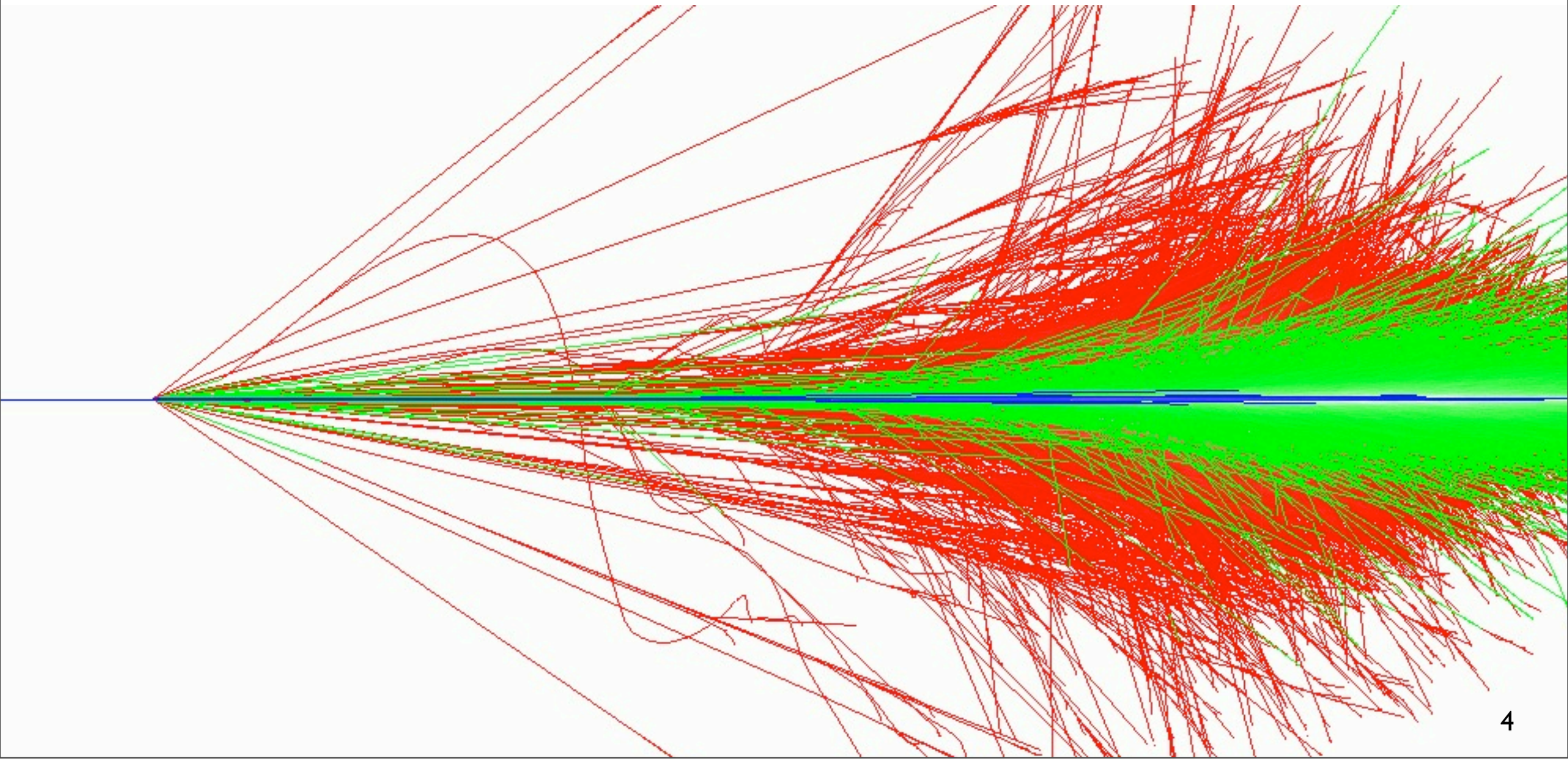
Goals

- Large experiments are investigating task-based parallelism for their software framework
  - **TBB** has been chosen as leading technology
- Geant4-MT is embedded in G4 Version 10 (Dec. 2013)
- Ensure G4MT is usable in experiment frameworks
  - Part 1: Viability - Andrea Dotti reported results (Dec 2012)
    - Is MT in **G4/MT "compatible"** with such frameworks?
    - What, if any, **changes to G4 code** needed?
    - Can we provide a simple TBB-based application as an **example**?
  - Part II: Using resources better (new)

Outlook

- Introduction
  - G4MT working model
- First experiences with TBB - reminder
  - Requirements
  - Some details
  - A simple TBB based application (ParN02tbb)
- Towards a second prototype
  - Limitations of first prototype
  - Analysis of G4MT - overcoming limitations
- Status and Plans

G4MT: reminder



Multi Thread

Master Thread

In the master thread a pseudo-event loop fills the seeds array.

/run/beamOn 4

Event	Seed
0	123456
1	876532
2	666534
3	876473

Worker threads spawned:
Shares Read-only part of geom. & phys.
Read-write parts are copied

Worker 1

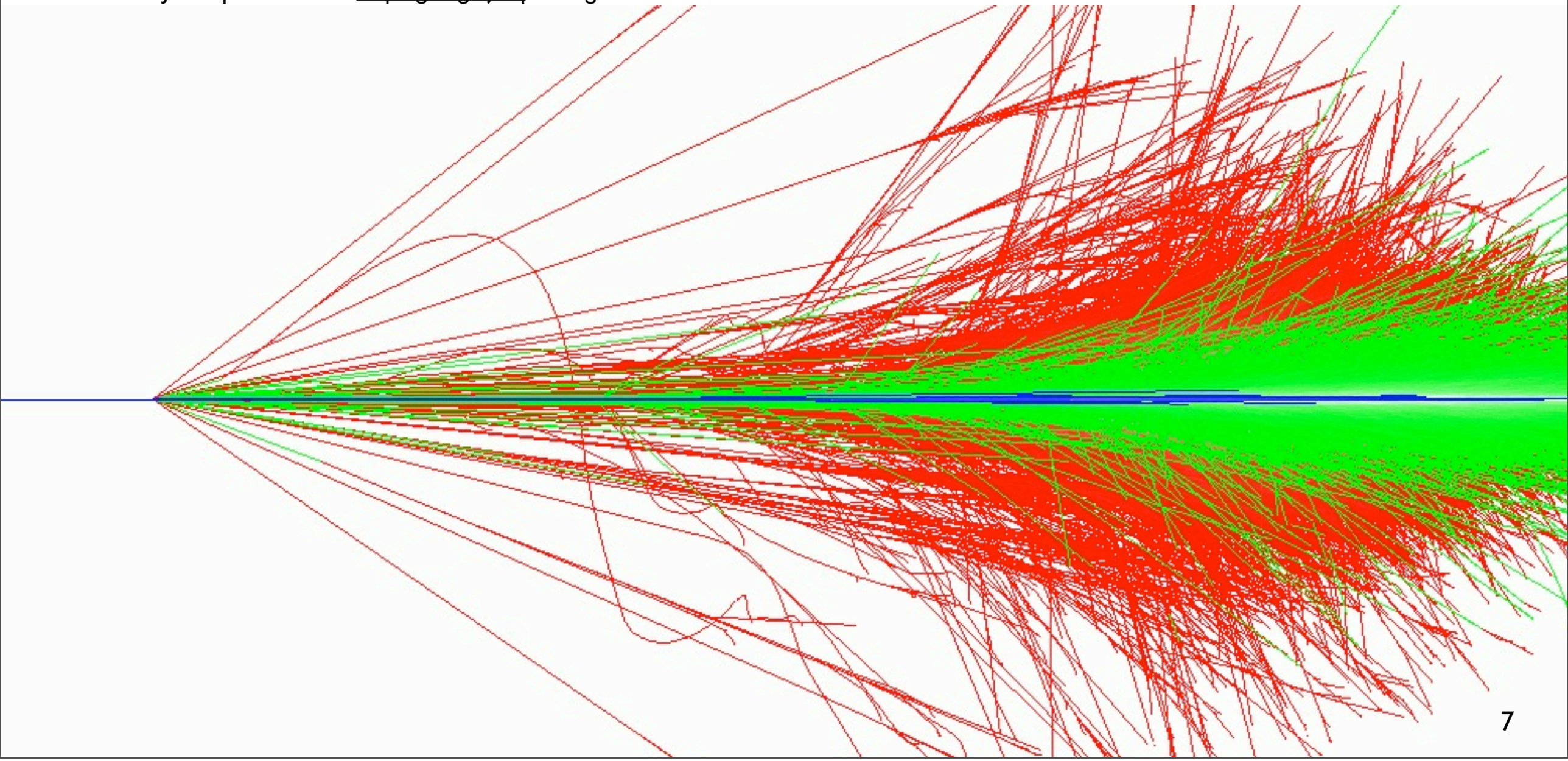
Worker 2

Static event decomposition: is round robin.

- Separate state for classes which have event-dependent state
- Split classes have part of state const, part changing:
  - Replica: id of current track (plan: to keep it only in Navigator state)
  - Logical volume: the solid, material is changed in Parameterised volume
  - Particle definition has pointer to Process Manager - which must point to different sets of processes for each worker (thread/task)
- Many classes are instantiated for each worker
  - All processes; Navigator; Classes for propagating in field

Use of TBB

See C. Jones presentation: http://goo.gl/fjRIq for a good introduction to TBB



Requirements for task/TBB

- **Do not change** G4 code
- All TBB specific code should be **external to G4**
  - As it is in experiment frameworks
- **Simplest solution:**
  - Sub-class G4RunManager to create a proxy to G4
  - Encapsulate in this new run manager all TBB specific code

First Implementation

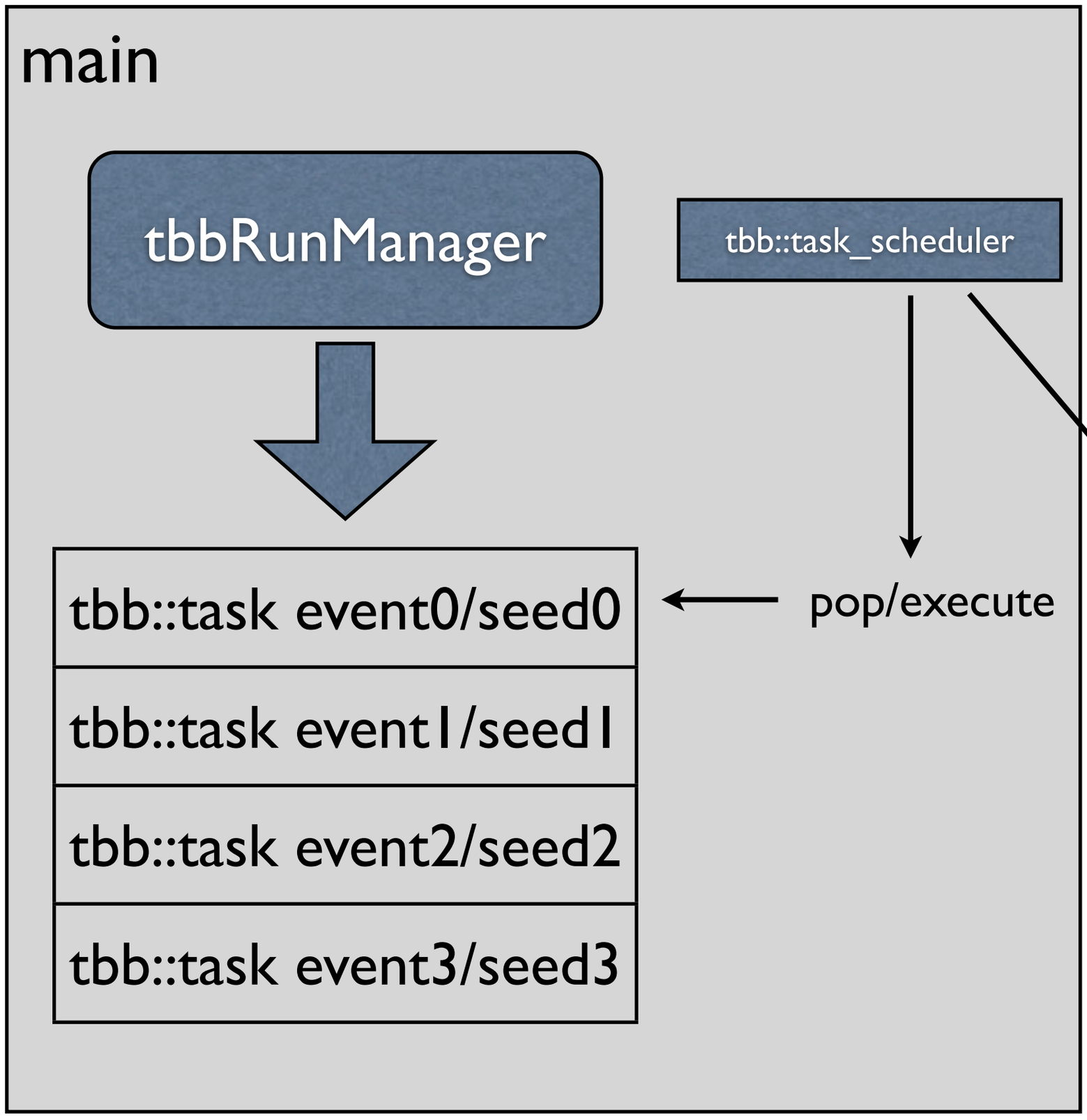
- Andrea Dotti opted for the simplest solution:
  - Derive from G4RunManager and re-implement DoEventLoop
- Similar to **large experiment frameworks:**
  - Take control of G4 re-implementing G4RunManager
- Similar to G4MT**: perform a preliminary pseudo-event loop**
  - Create a list of tasks initialized with predetermined random seeds
  - A single tbb::task simulate of one event
    - Maintaining the association (task ↔ event) simplifies the collection of hits: only one task creates hits for the event's hit collection.

Note

- User-specific RunManager
  - Incapsulates all tbb logic
- There is no need to modify any G4 class
- Starting from 9.6 G4RunManager will extend interface
  - Breaking down DoEventLoop
  - Achieving simpler control of run-workflow for derived run managers

An important point

- The derived RunManager **has no control on threads**
  - Each task/thread needs a private instance of G4 kernel (physics, SD, user actions, ...)
  - A task/thread also needs access to shared resources (geometry, physics tables)
- Consequence for **first version**:
  - each tbb::task checks if its thread has an initialized "context" = workspace
  - if needed it creates itself a "context"
  - the "context" will be re-used by any other task running on this thread.



Main function (main thread) creates the list of tbb::tasks

With 2 threads the task scheduler pops two tasks at the same time

pop/execute
New thread condition
Initialize Job

Differences between TBB and G4MT

- Strategy is very similar with two differences
- In MT the G4 class *G4MTRunManager spawns and control threads*
- With tasks the control of threads is by the task system
  - in this case TBB's runtime system
- In G4MT all threads are initialized at the same time before any event is simulated. It's not the case with this example
  - There is "lazy initialization" with TBB: **initialize context only when needed**

Differences between TBB and G4MT

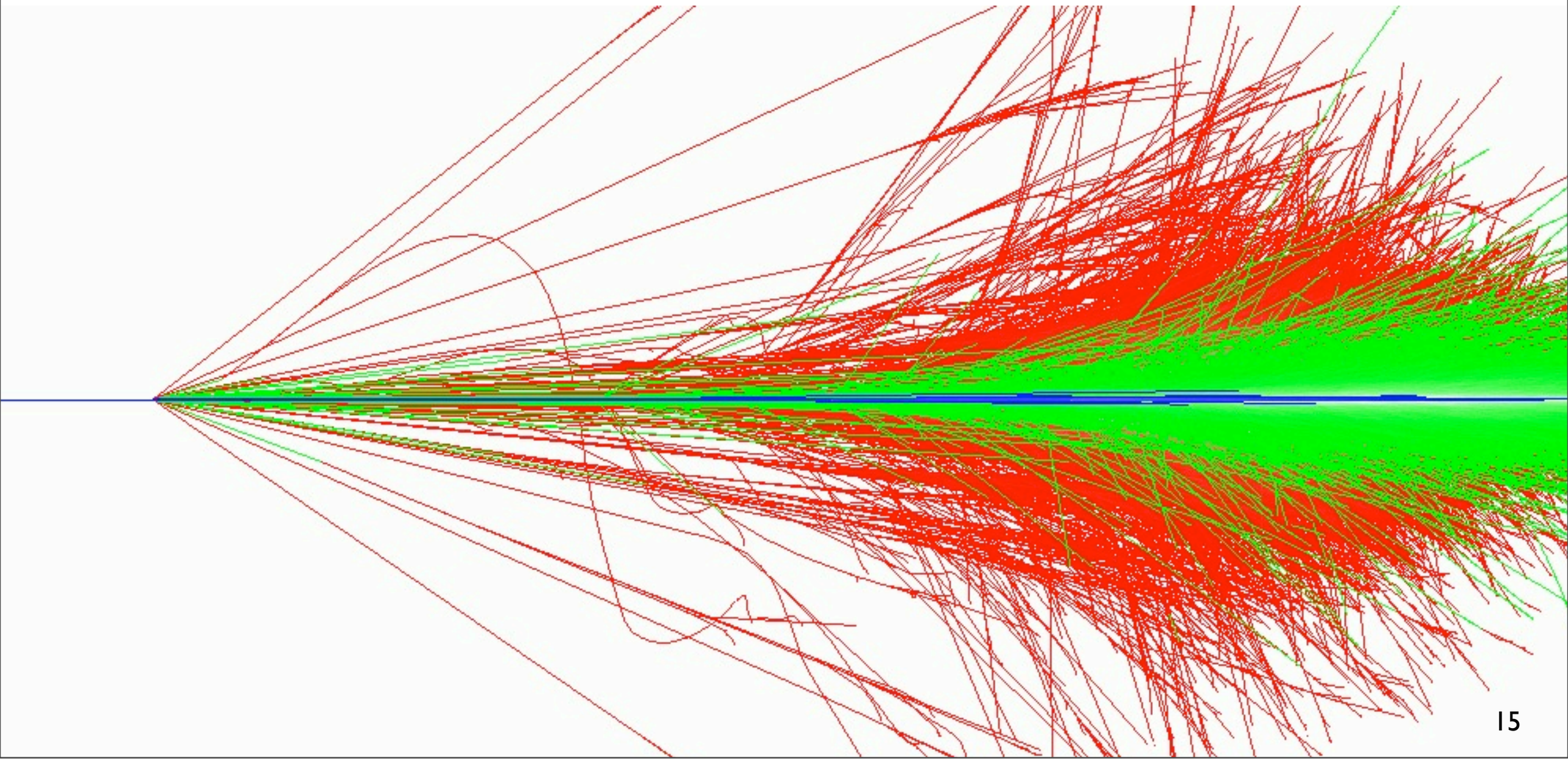
Multi-threading

- Explicit spawning and control of threads
  - G4MTRunManager
- All threads initialized at same time,
  - before any event is simulated.

Task-parallelism

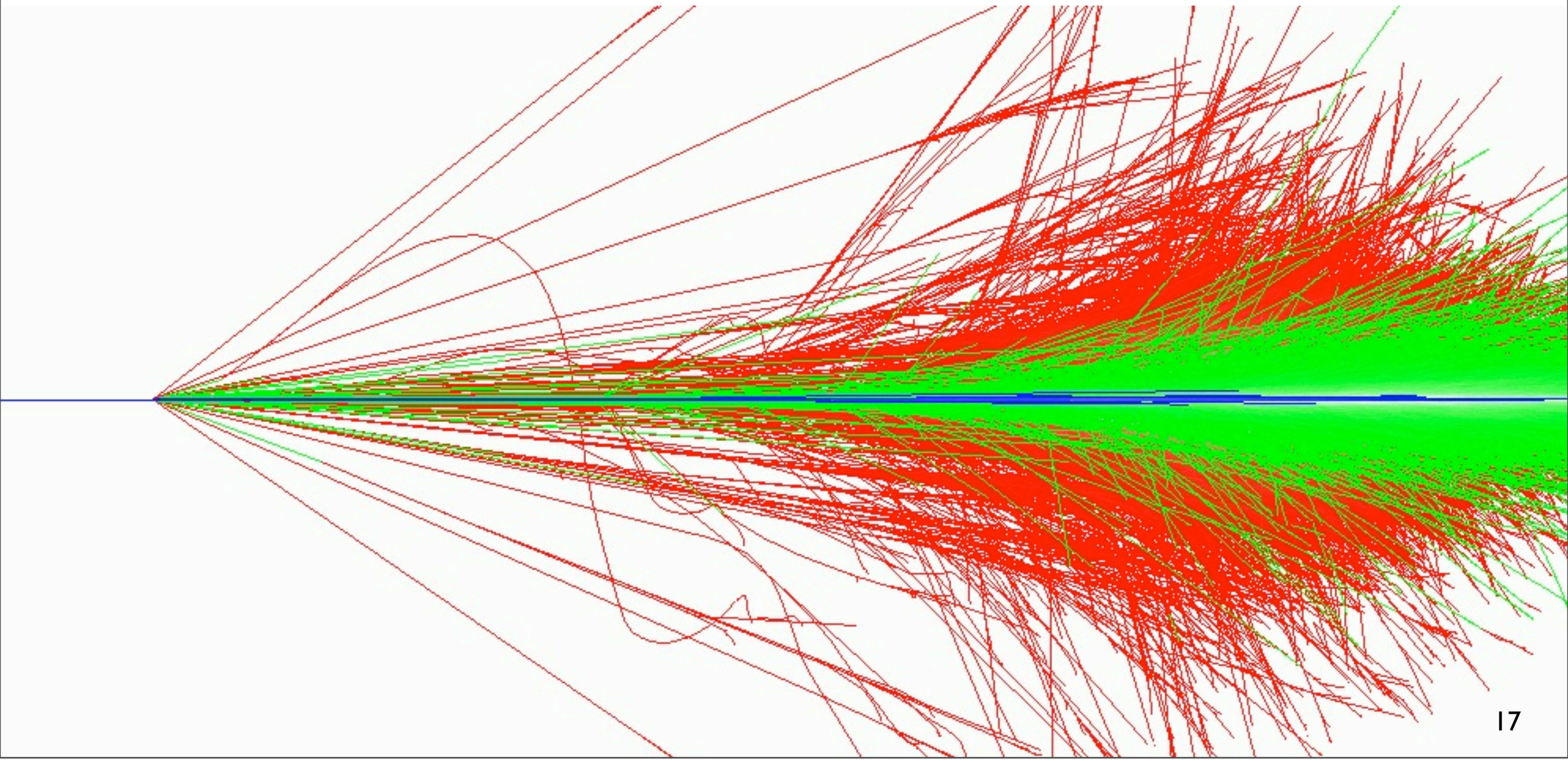
- Control is by the task system
  - e.g. TBB's runtime system
  - It creates and reuses thread
- Lazy initialization
  - initialize context only when needed
- Switching to a new task is fast

First conclusions (Dec.)



- Created a simple G4MT application with TBB
  - Events are tasks that can be executed in parallel
- No **code change** in G4MT was needed
- Developed few new classes "gluing" G4MT with TBB
- Having a bit of experience with G4MT *implementation* was **straightforward**
  - No major issues observed
  - The new G4RunManager interface should simplify developments further
- Testing and polishing of code needed
  - Including checks with large number of threads and scaling measurements
- **Aim to providing an official "TBB example"** in future G4MT prototype

Beyond first prototype



Goals

- Andrea's first prototype was based on Geant4MT 9.5p01
  - methods have been changed in development for Geant4 10.0-beta
  - designed evolved, methods renamed, ..
- First step: Update to development line
  - Make it work with recent reference tags (thus Geant4 10-beta)
- Second step: Improve - adapt better to experiment needs
  - Maintaining full compatibility with Geant4-MT
  - Adding methods to existing classes, and new classes

Limitations of version 1

- Conditions
  - Expect that the number of active threads is larger than the number of G4 sim tasks ongoing in parallel
- Limitations in the first version
  - Each active thread needs a separate work area for its private data, including its TLS data and split classes
  - The state must be initialized by each thread for the first event which it simulates
  - The amount of memory used by the simulation is larger than 'needed'
    - It is proportional to the total number of *threads* used in Geant4 processing - not the ones concurrently active
  - At the end of job, it is not possible to delete the G4/TBB objects.

Second TBB version

- To improve, need to reuse memory/objects
  - Idea: organise them into a 'workspace'
- A 'workspace' owns
  - the 'per-thread' state of 'split' object,
  - by extension it owns all objects created for them (state of replicas, the full set of physics processes)
- A running task must obtain/create a workspace
  - the number of workspaces will be proportional to the (maximum) number of simultaneous G4 jobs being executed.

Per thread state

- State of split objects
  - parameterised volumes, replicas and logical volumes
  - Physics Vector
  - particle (pointer to process manager)

- The objects dependent on split objects
  - full set of physics processes (from process manager)

- Stack of tracks, hits
  - allocated on the heap
  - already on per-thread banks of memory

Building 'version 2'

- Identify coherent part(s) of the 'state' to separate into workspace
  - Must contain all inter-dependent classes
    - Part 1: all geometry 'split' objects
    - Part 2: all particle split classes and processes
- Separate all existing references
- Add methods to create and manage 'workspaces'
  - Hook them into the TBB task initialisation / ending

Workspace use

- When a task starts
  - Obtain a Workspace: find an existing unused one, if available, or create one
  - Use the Workspace - hook it up: set up all Thread Local pointer areas to use it
- When a task is done
  - Unhook the Workspace - set your pointers to null
  - Return the Workspace to a 'Store' - a concurrent container of available workspaces

Issues and improvements

- Updated "G4/TBB" classes and helpers
  - to work with Geant4 9.6-ref05
  - fixed problem in example Novice N02 and ParN02tbb
- Improved CMake files - with help from Pere
  - Use Find.cmake. Works with Xcode (interim solution for DYLD libs)
- Inspected code for geometry
  - Identified issue with initialisation of parameterised volumes
  - Created code that reports case which is not supported
- Inspected code for physics lists, constructors
  - To identify creation sequence of physics process/model classes

Status of Workspaces

- Geometry Workspace
  - Combine initialization of all 'split' classes into one class
  - This class can be used both for vanilla MT and for TBB
  - Code is ready for commit - using it simplifies vanilla MT worker initialisation
- Other Workspaces
  - Any other libraries with split classes should become one or more Worskpaces
  - Simple extension - should not be time consuming to create
-

Example code

- The idea is to have two examples:
- Example I - Simple TBB or TaskParallel #1
  - Based on example B2b
  - Using simple TBB - no workspace management
- Example II - Advanced TBB or TaskParallel #2
  - Based on example B2B
  - Using 'advanced' TBB, using workspaces

Subsequent steps

- Things to check
  - memory use (Geant4 MT, MT+TBB) for realistic geometry
  - performance, scalability
  - ensure same results as sequential version (with same seeding)

- Repeatability
  - once it is reliable, on-demand/TBB is strong(er) check of repeatability
    - because ordering of events is potentially different in each run.

- Examine potential for sub-event parallelism
  - Need to deal with 'heterogenous' hit collection - by >1 tasks/threads
  - opportunity for better affinity of tracks, by particle type, volume, ..