First experiences with G4MT prototype

Andrea Dotti (andrea.dotti@cern.ch)

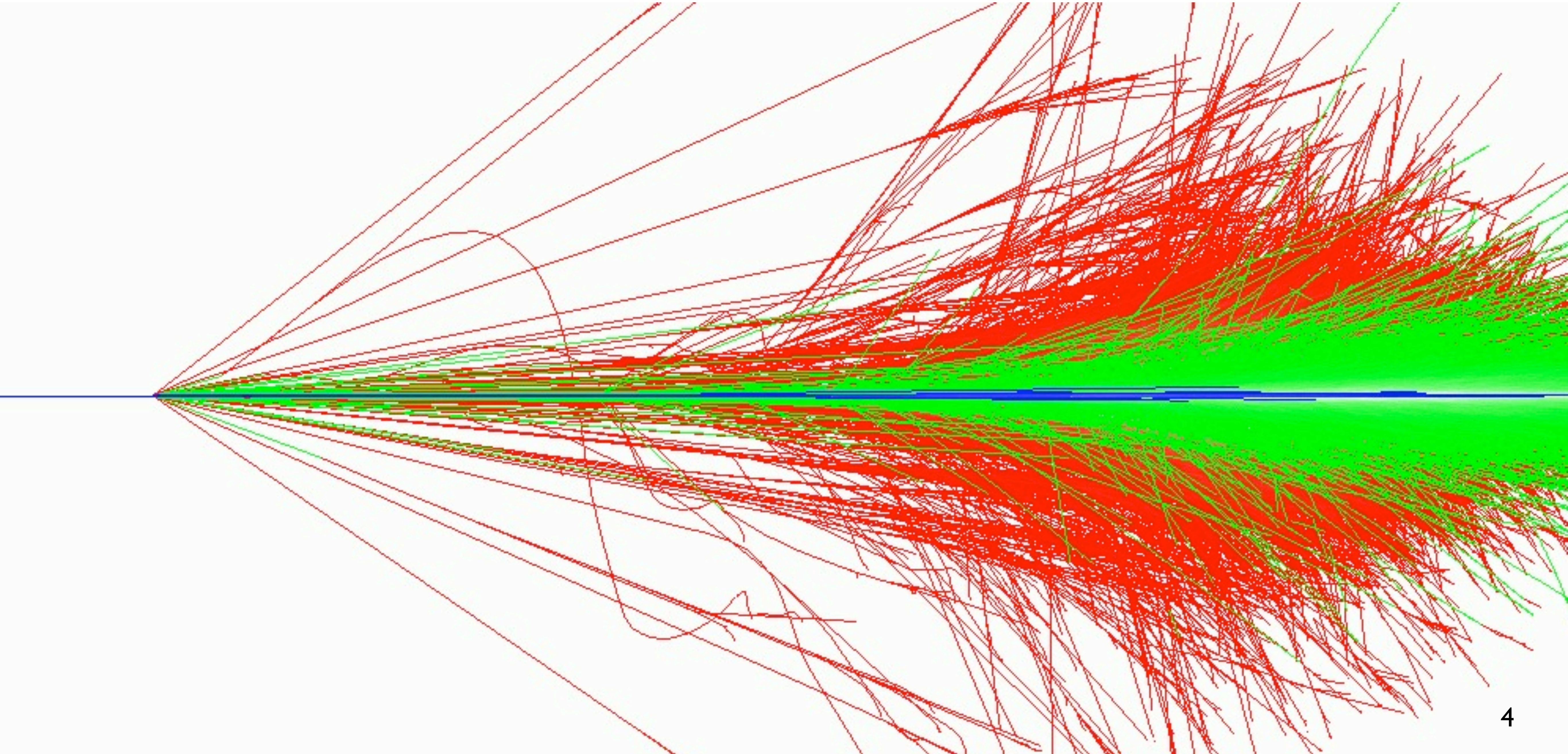
Goals

- Large experiments are investigating task-based parallelism for their software framework
  - **TBB** looks particularly promising
- Geant4-MT capabilities will be embedded in Geant4 Version 10 (Dec. 2013)
- **We need to be sure that G4MT can be used in these (parallel) frameworks**
- We need to answer few questions:
  - **Is G4MT "compatible"** with such frameworks?
  - Are **changes to G4 code** needed?
  - Can we provide a simple TBB-based application as an **example**?

Outlook

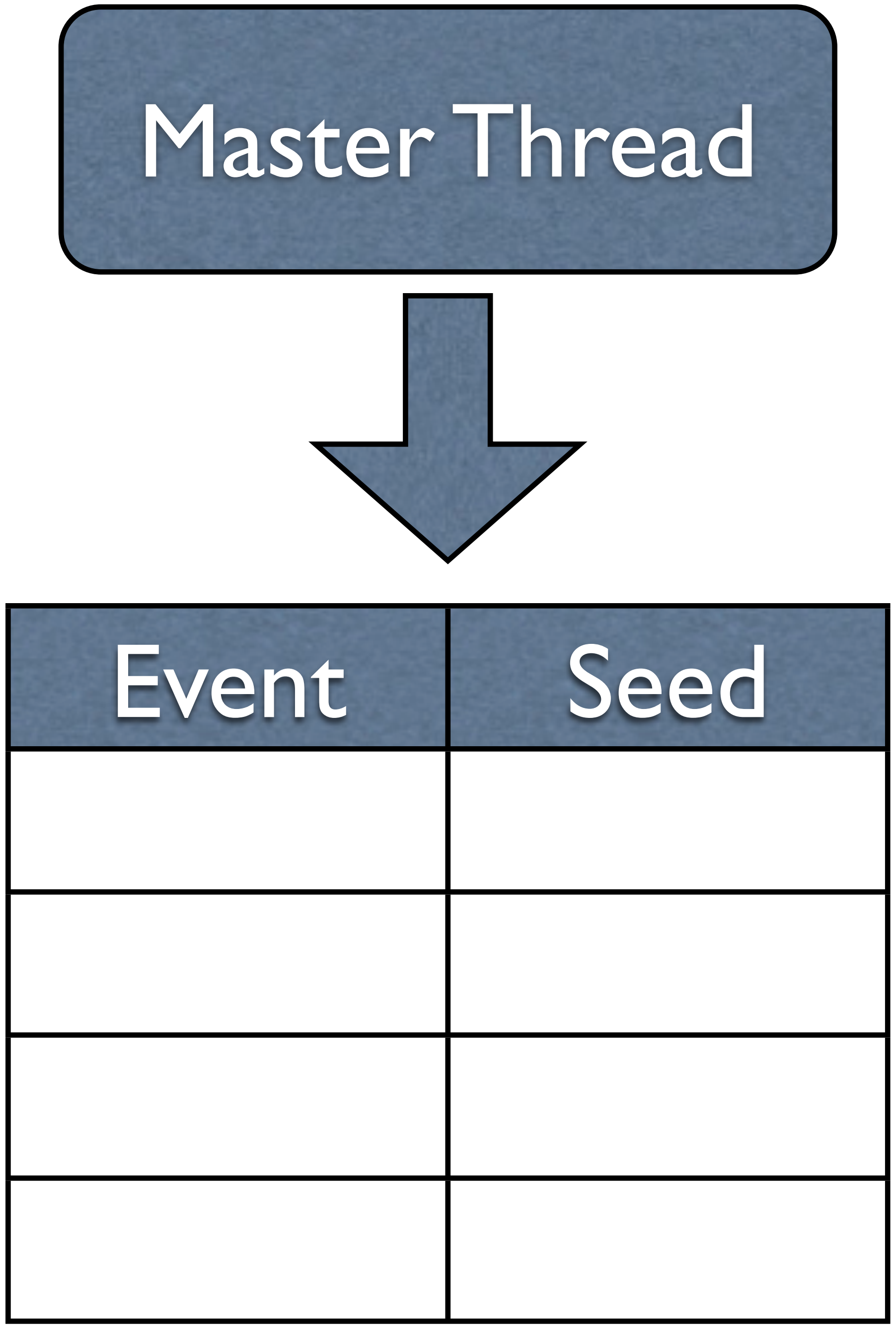
- Introduction
  - G4MT working model: a reminder
- First experiences with TBB
  - Requirements
  - Some details
  - A simple TBB based application (ParN02tbb)
- Conclusions

G4MT: reminder



- G4MT is a effort in two directions:
  - Make the relevant classes in Geant4 thread-safe
  - Provide a G4MTRunManager that implements event-level parallelism
    - Simple applications can use directly G4MTRunManager
    - Complex ones will do as they always did: write/subclass their own run-manager
- G4MT developed with easiness of porting as a guiding principle
  - Porting of a simple application should takes few hours

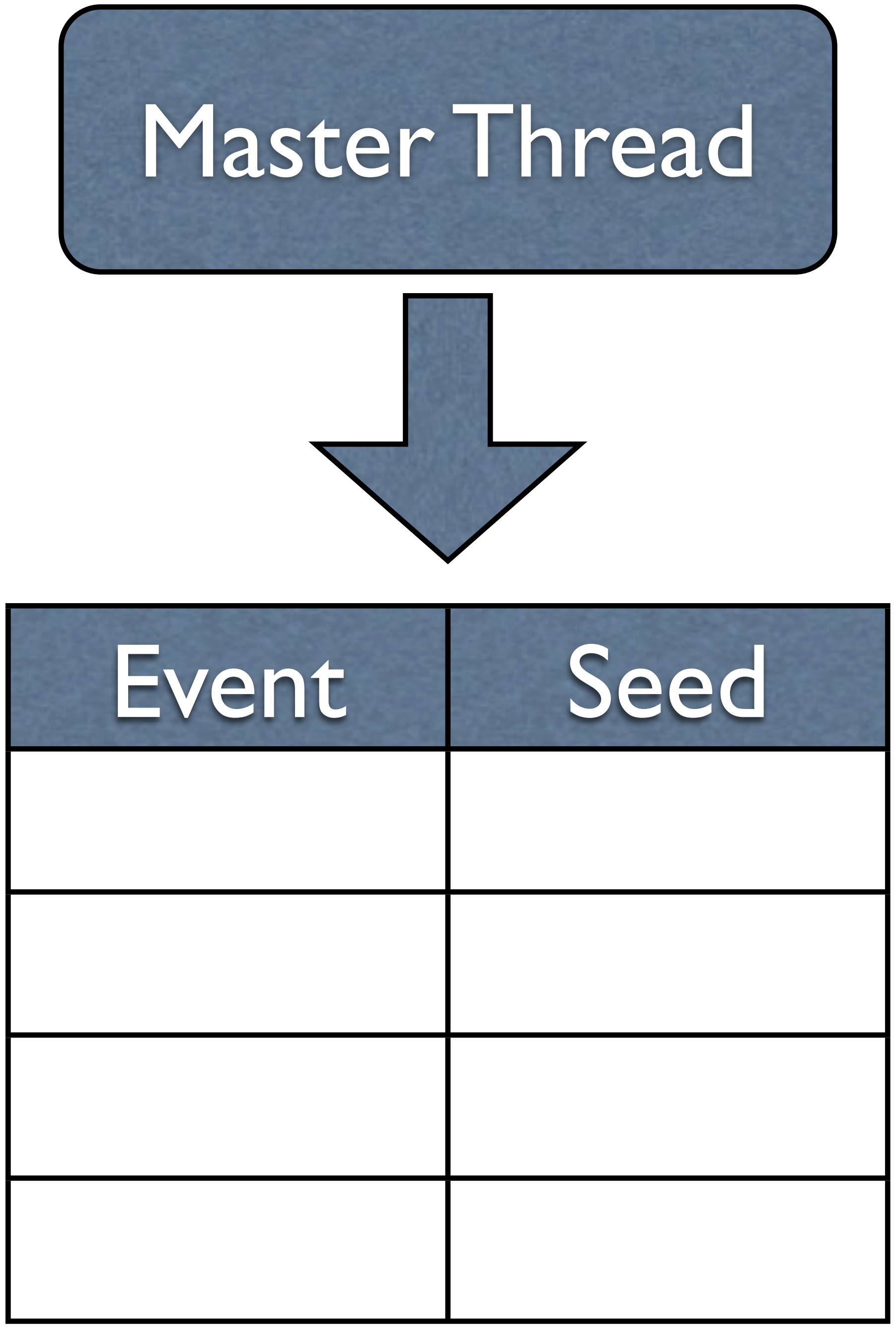
G4MTRunManager workflow



Master Thread

Event	Seed

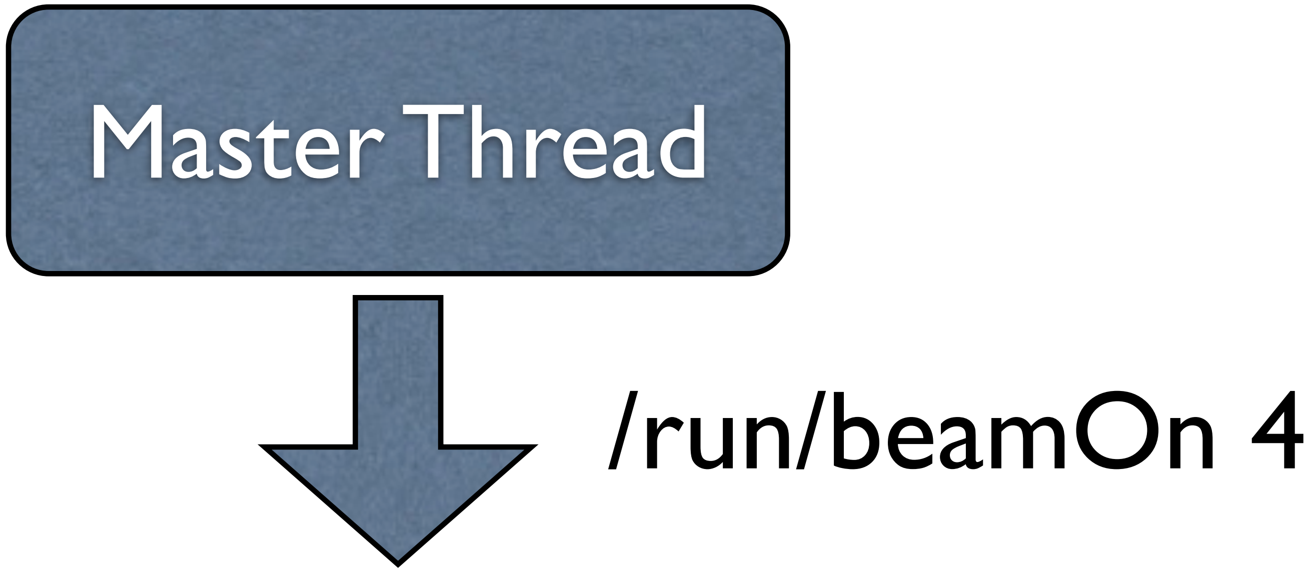
Master thread holds
a shared array that maps:
EventNum ↦ Event Random Seed



Master Thread

Event	Seed

Job is started:
Geometry and physics are built
G4 kernel is initialized

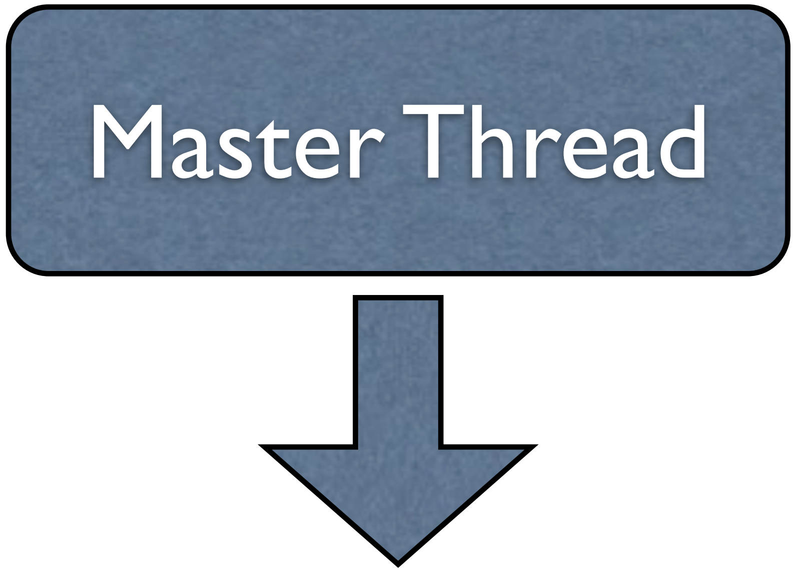


Master Thread

/run/beamOn 4

Event	Seed
0	123456
1	876532
2	666534
3	876473

A pseudo-event loop is started: the seeds array is filled.
Note: this is guarantees reproducibility
See A. Ribon's http://goo.gl/rDMxg for a discussion on reproducibility



Master Thread

Event	Seed
0	123456
1	876532
2	666534
3	876473

Worker threads are spawned:
Each one initialize a worker version of the run manager

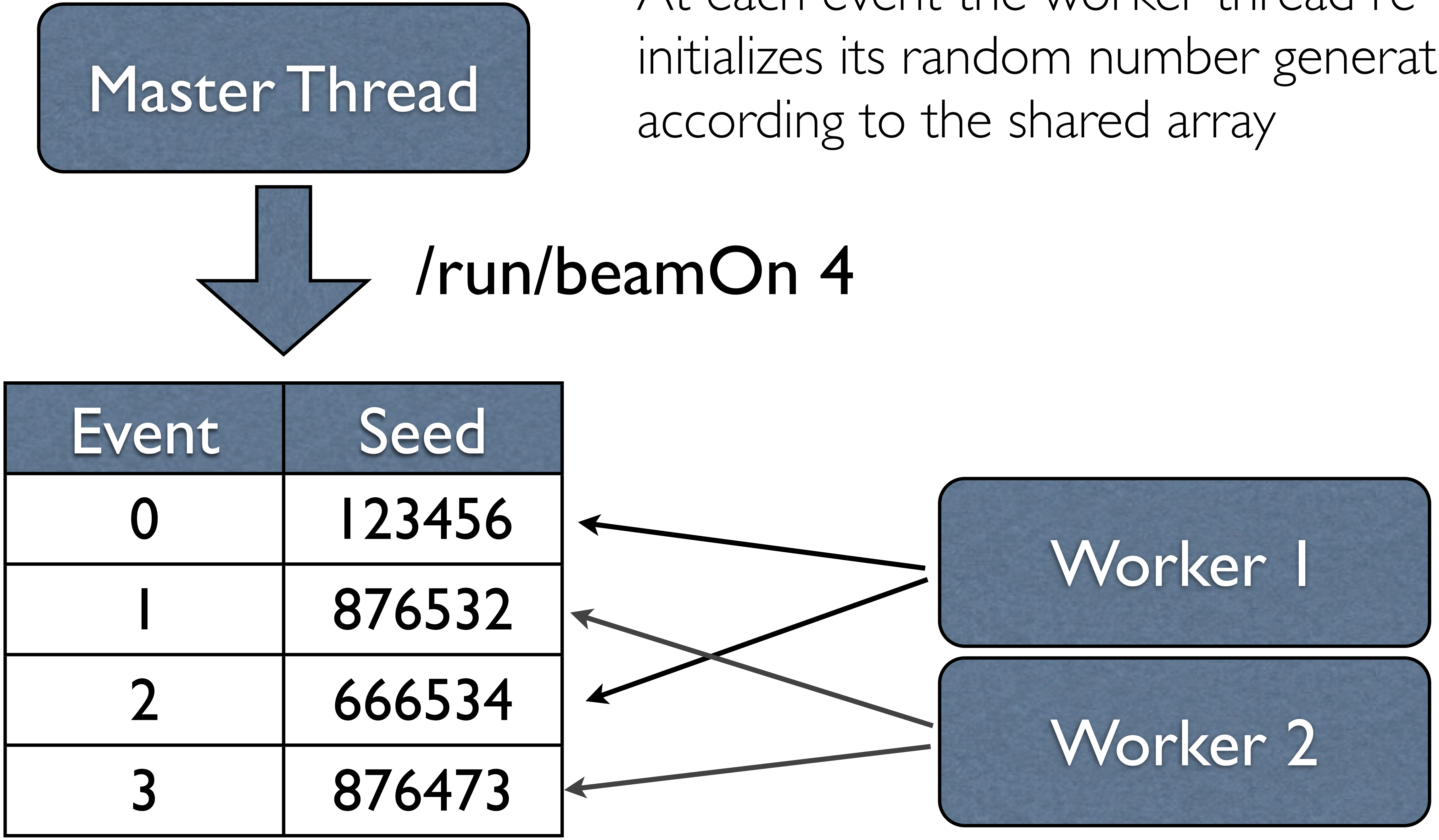
Read-only parts of geometry and physics are shared, read-write parts are copied

Worker 1

Worker 2

To guarantee maximum portability:
POSIX threads

Events are processed in round robin way. At each event the worker thread re-initializes its random number generator according to the shared array



Master Thread

/run/beamOn 4

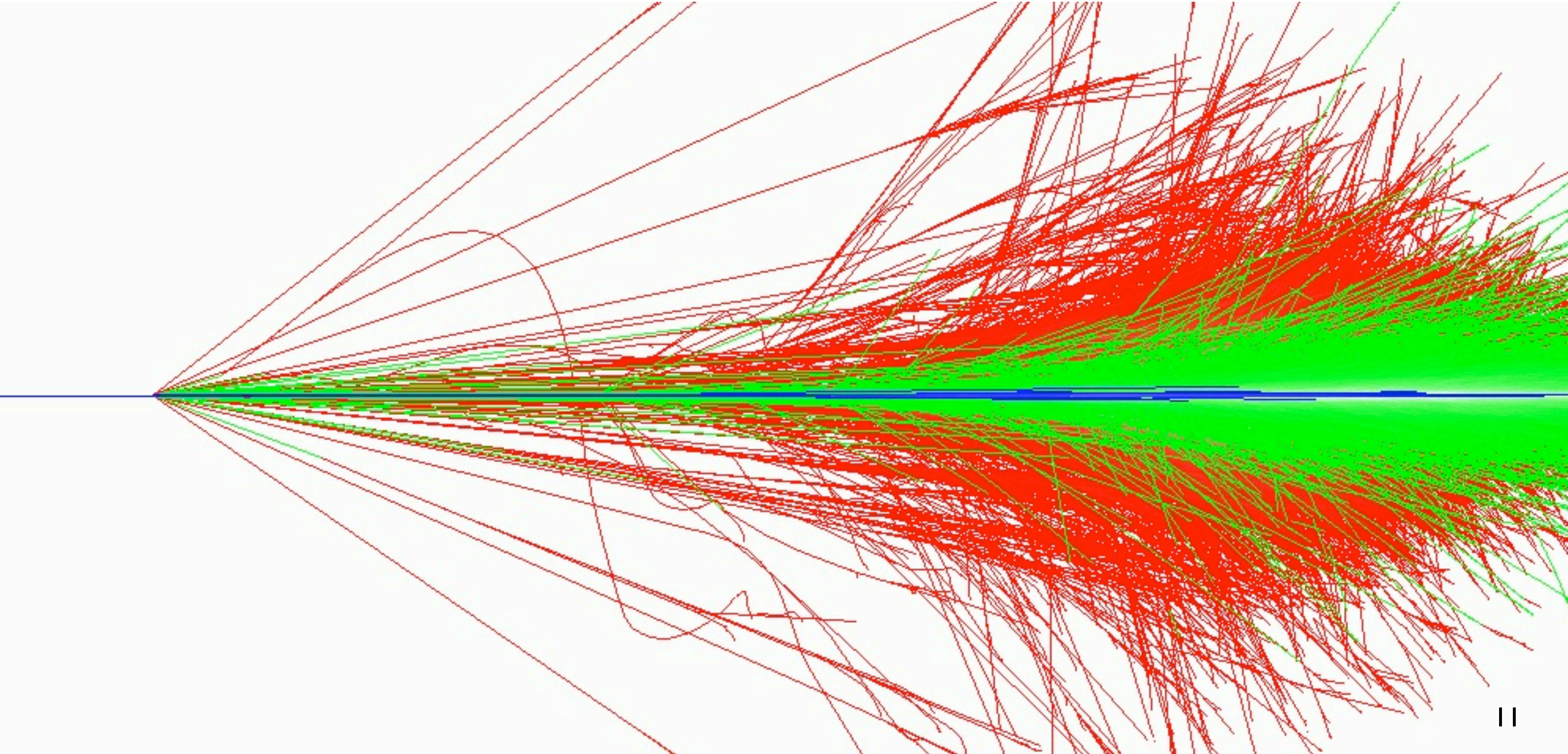
Event	Seed
0	123456
1	876532
2	666534
3	876473

Worker 1

Worker 2

Use of TBB

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



Requirements

- **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

My Implementation

- I opted for the simplest solution:
  - Derive from G4RunManager and re-implement DoEventLoop
- **Similarly to large experiment frameworks:**
  - Take control of G4 re-implementing G4RunManager
- **Similarly to G4MT: perform a preliminary event loop**
  - Simulation of one event is a single tbb::task
  - Create a list of tasks initialized with predetermined random seeds
  - Obtaining an association: task ↔event

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

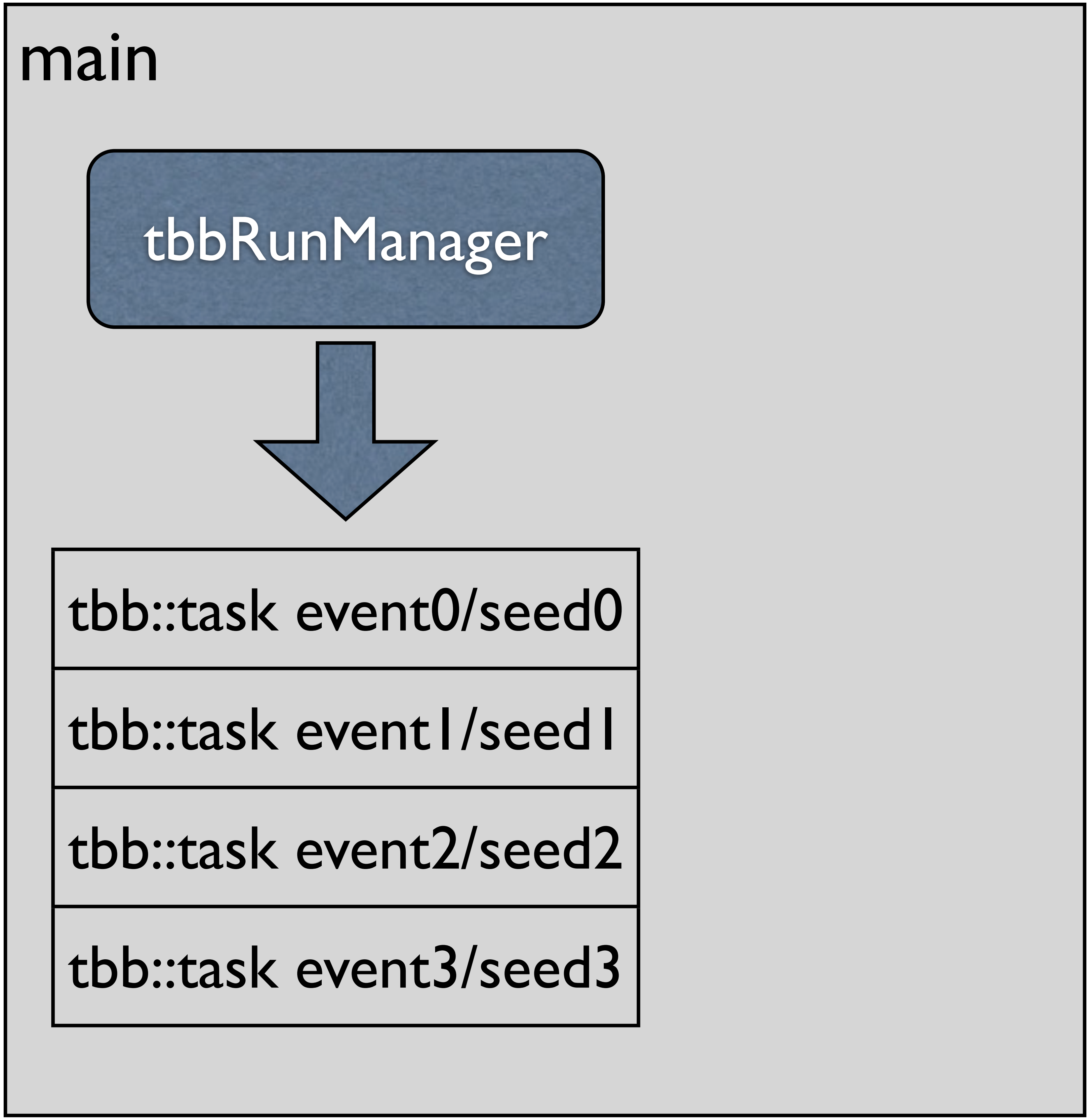
- The derived RunManager **has no control on threads**
  - However each thread needs a (private) instance of G4 kernel (physics, SD, user actions, ...) and access to shared resources (geometry, physics tables)
- **Consequence**: before doing simulation work each tbb::task checks if current thread has an already initialized “context”, if not it sets up things correctly (this “context” will be re-used by any other task running on this thread)

Some pseudo-code

```
class G4RunManager {
private:
  static __thread G4RunManager* instance;
public:
  static G4RunManager* GetRunManager() {return instance;}
  [...]
};
```

Current G4MT

```
tbb::task* MyTask::execute() {
  if ( G4RunManager::GetRunManager() == NULL ) {
     tbbRunManager = new tbbRunManager();
     tbbRunManager->InitializeWorker();
  }
  G4Random::setRandomSeed( ... );
  G4RunManager::GetRunManager()->DoOneEvent();
  return NULL;
}
```



main

tbbRunManager

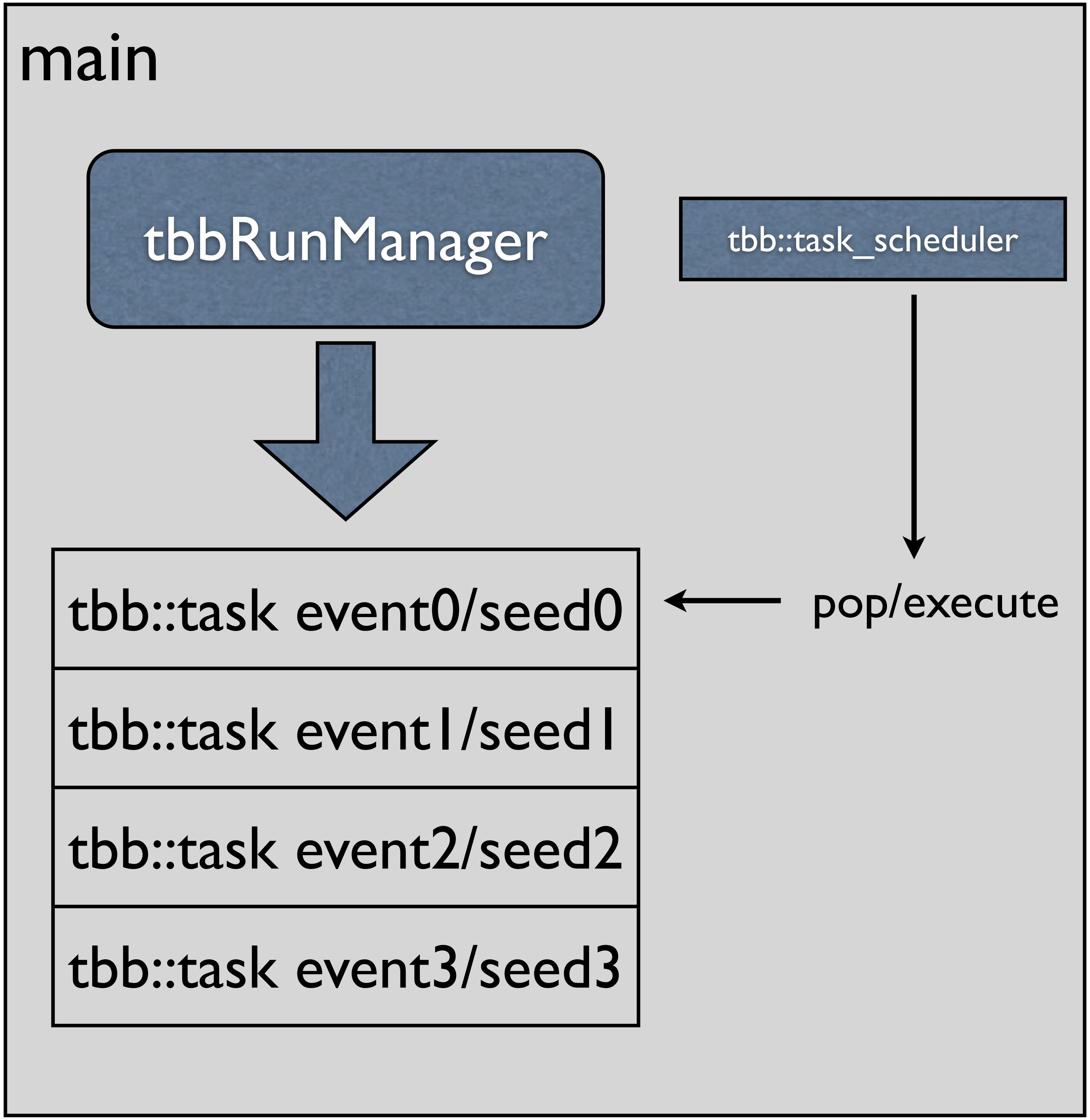
tbb::task event0/seed0

tbb::task event1/seed1

tbb::task event2/seed2

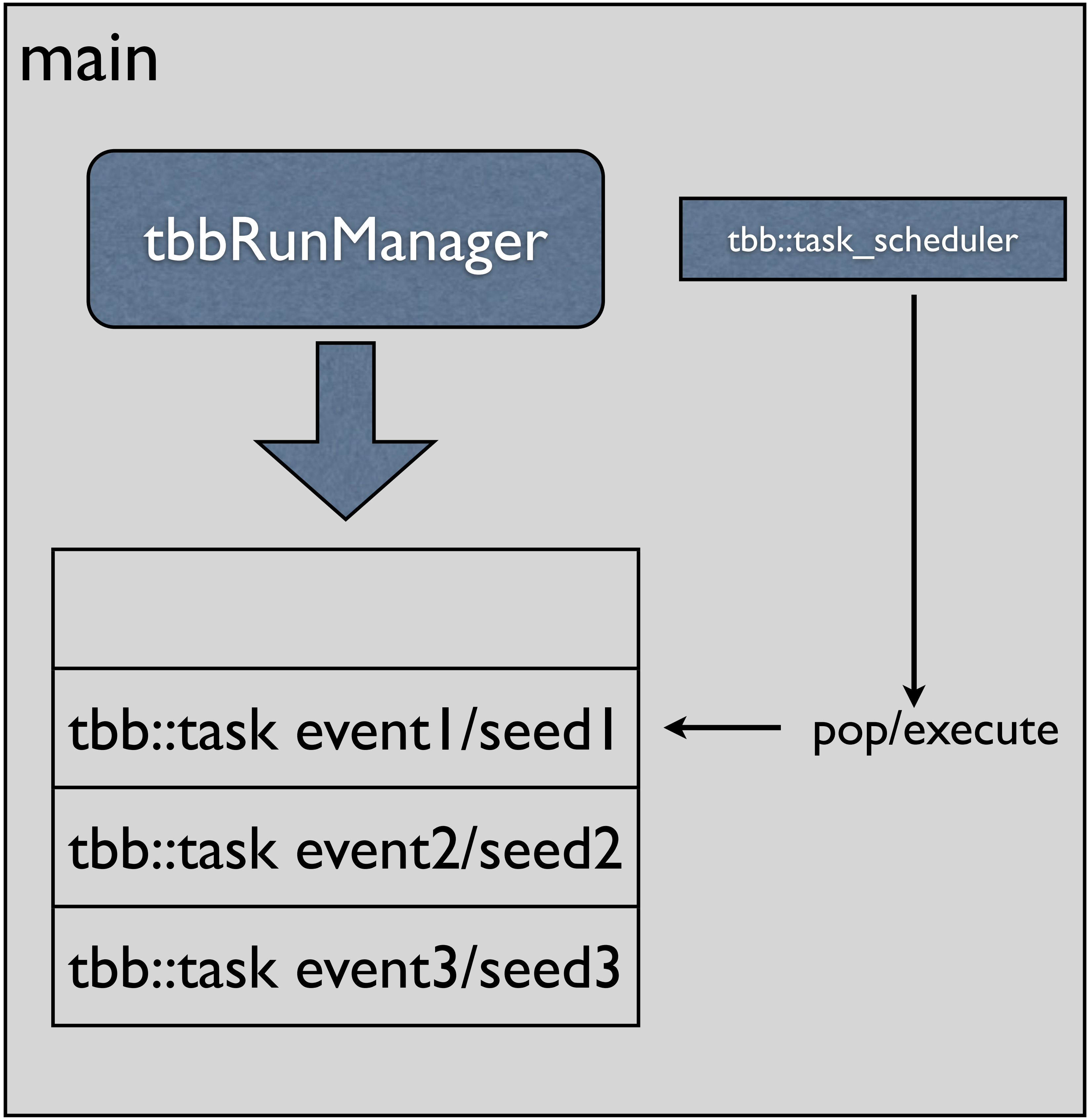
tbb::task event3/seed3

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



tbb::task event0/seed0
tbb::task event1/seed1
tbb::task event2/seed2
tbb::task event3/seed3

Let's assume now we have only 1 thread



main

tbbRunManager

tbb::task_scheduler

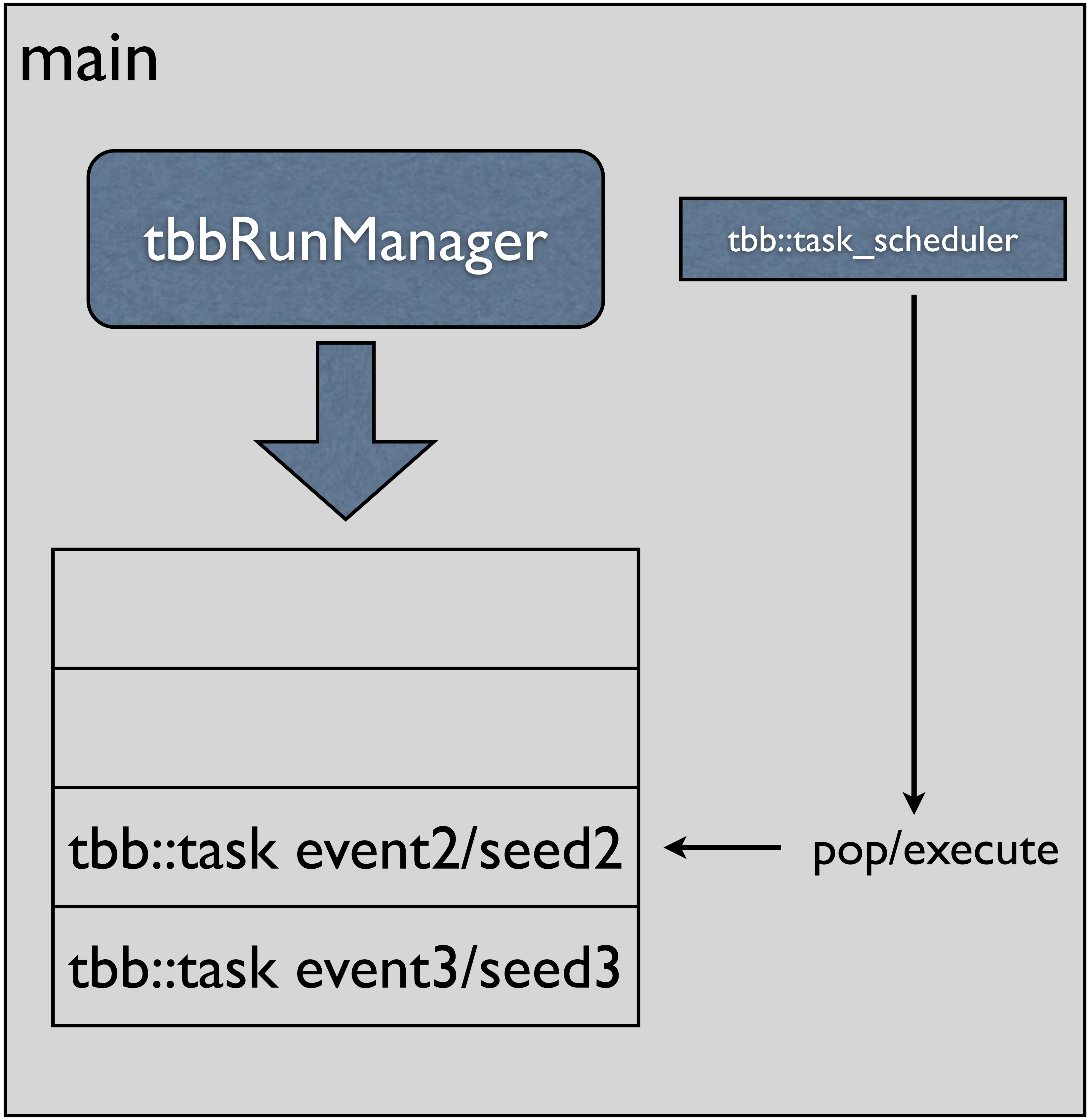
tbb::task event1/seed1

tbb::task event2/seed2

tbb::task event3/seed3

pop/execute

Let's assume now we have only 1 thread



main

tbbRunManager

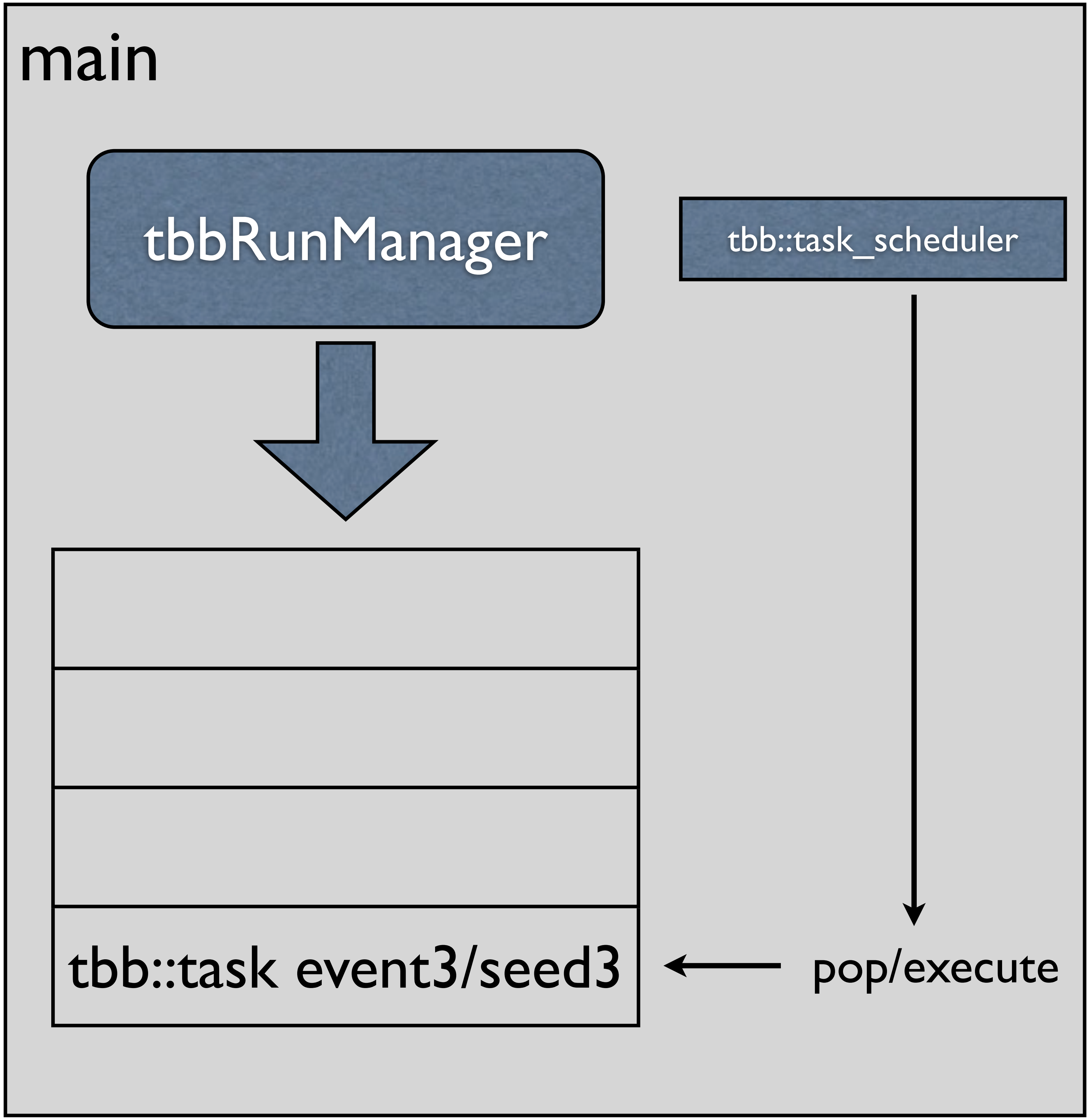
tbb::task_scheduler

tbb::task event2/seed2

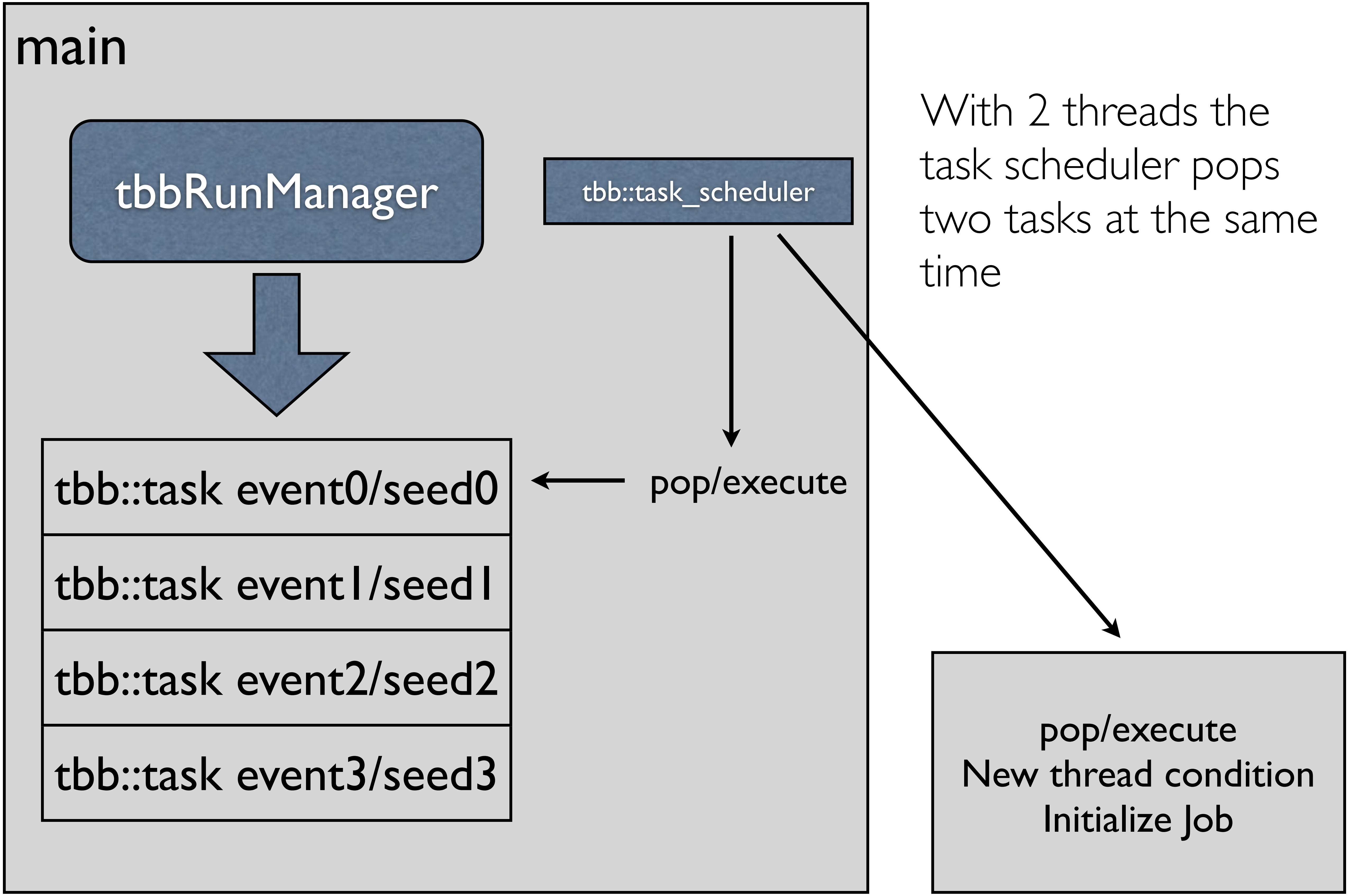
tbb::task event3/seed3

pop/execute

Let’s assume now we have only 1 thread



Let's assume now we have only 1 thread



main

tbbRunManager

tbb::task_scheduler

tbb::task event0/seed0

tbb::task event1/seed1

tbb::task event2/seed2

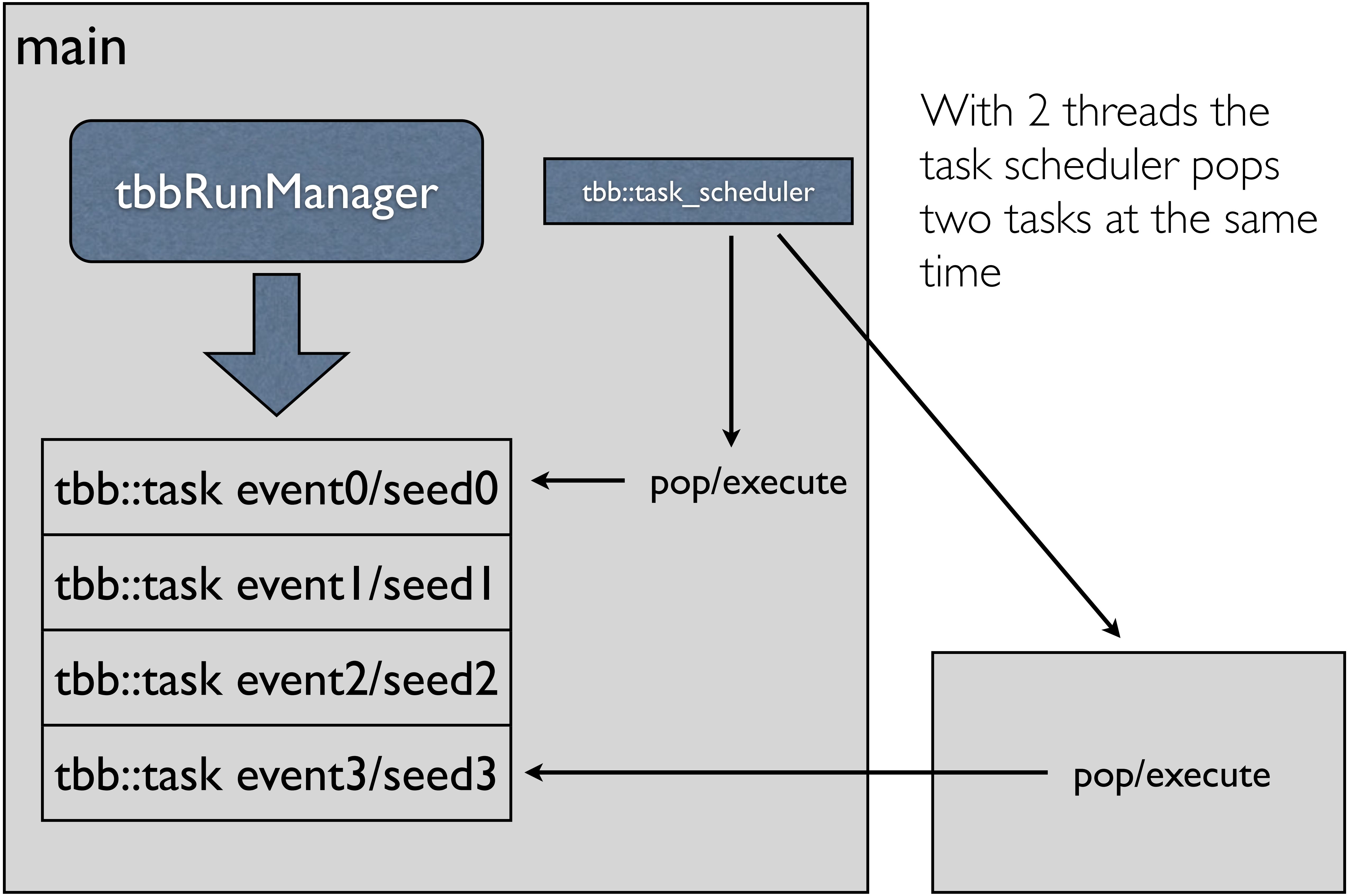
tbb::task event3/seed3

pop/execute

pop/execute
New thread condition
Initialize Job

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

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



main

tbbRunManager

tbb::task_scheduler

tbb::task event0/seed0

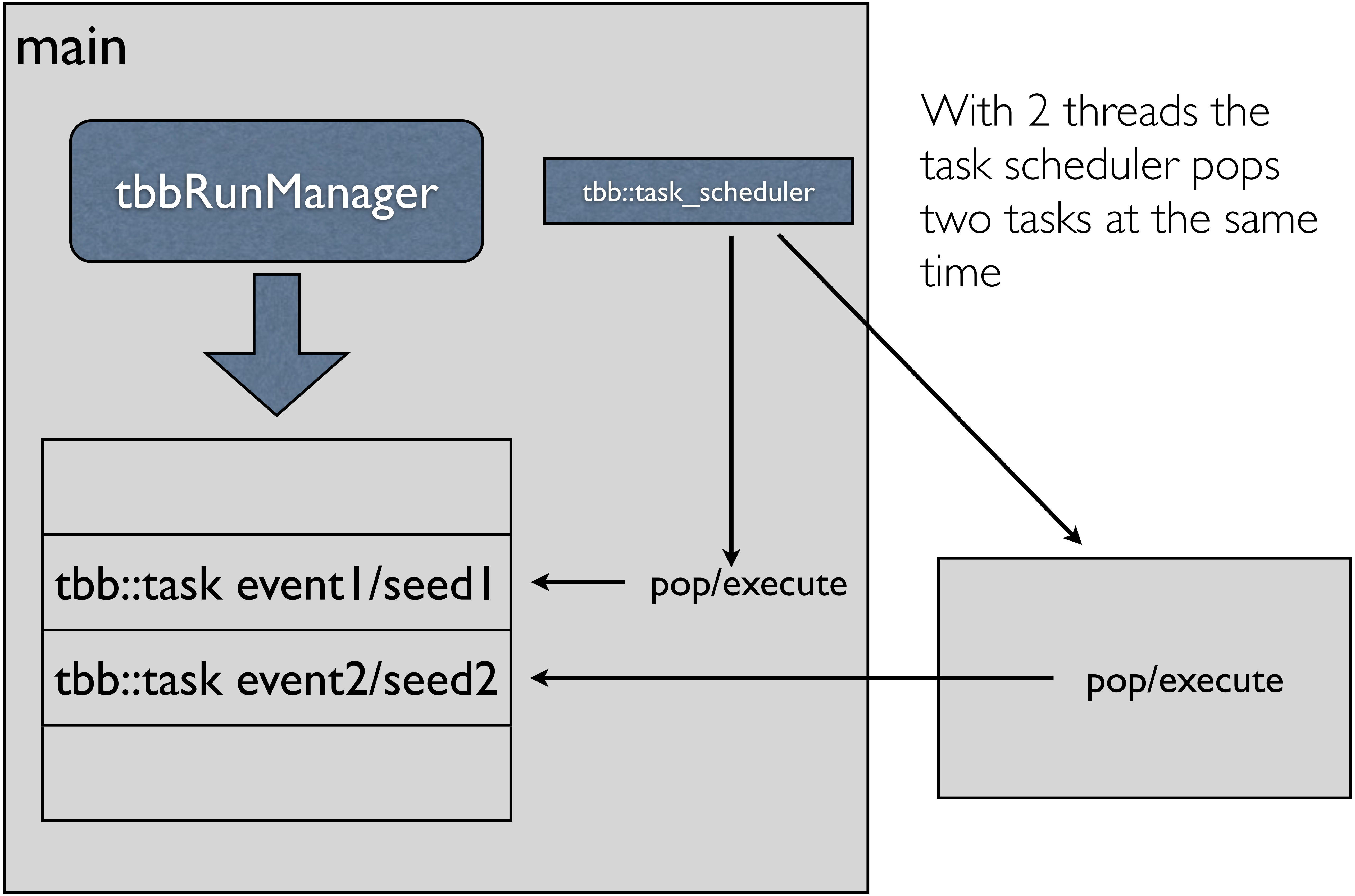
tbb::task event1/seed1

tbb::task event2/seed2

tbb::task event3/seed3

pop/execute

pop/execute

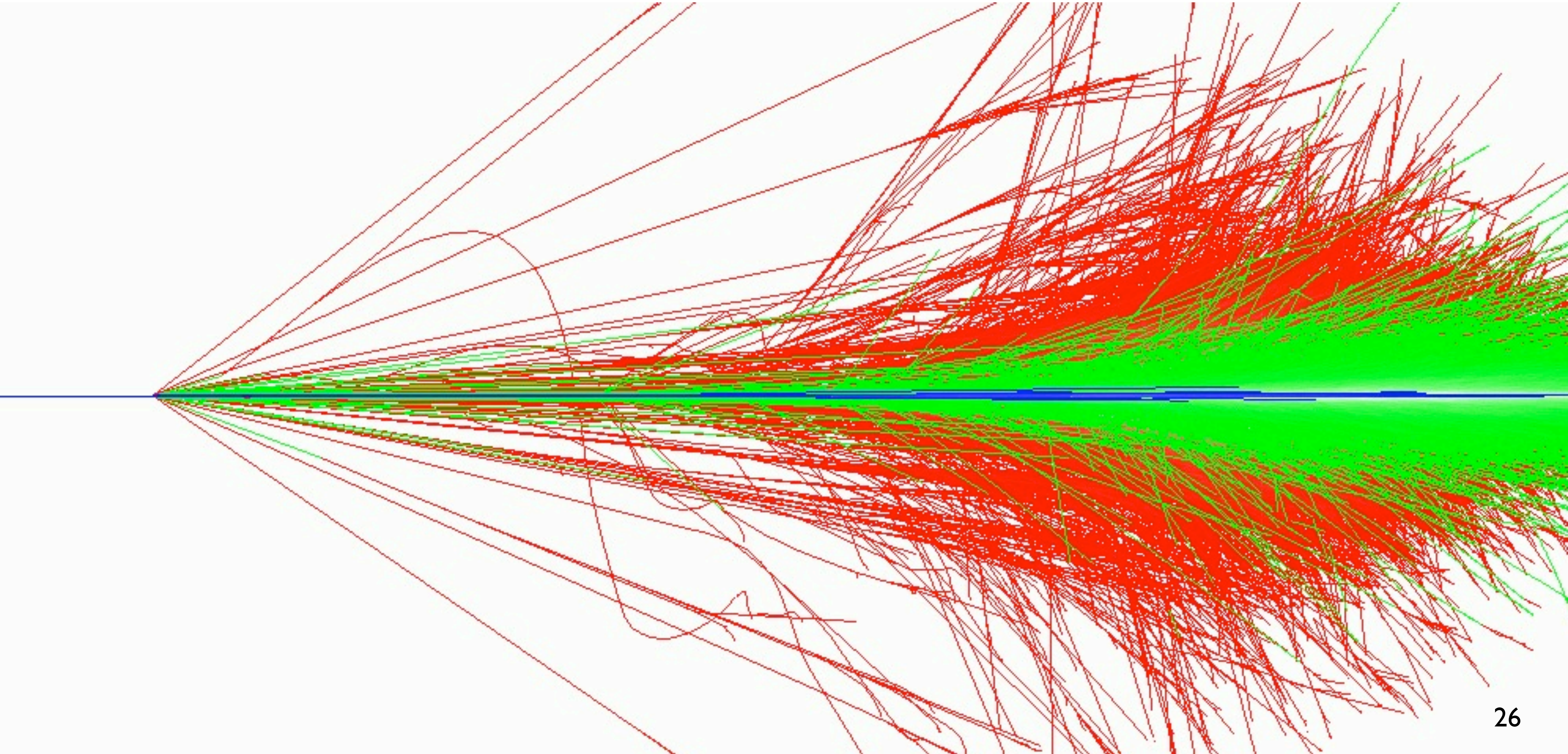


With 2 threads the task scheduler pops two tasks at the same time

Differences between TBB and G4MT

- Strategy is very similar with two differences
- It is G4MTRunManager to spawn/control threads, it's not the case with this example
- 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**

Conclusions



- Created a simple G4MT application with TBB
  - Events are tasks that can be executed in parallel
- No **code change** in G4MT was needed
- Few new classes were developed to "glue" 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

Other activities

- This exercise was part of a larger activity to investigate G4MT capabilities:
  - Porting to G4MT to **MacOSX** : **done** (at least for clang 3.1)
  - Porting of an application that includes **analysis code**: **done**
  - **Study reproducibility** (verify that G4MT gives same results as G4 when using same random seeds): **done** (need to be re-done with 9.6 and increase "strength" of test)
  - Porting to new **Intel Xeon Phi**: first preliminary porting **done**
  - Study **scaling** on Intel Xeon Phi: **ongoing**
  - **Provide example** based on TBB: **to do**