Current Status of Geant4 MultiThreading
– How it is designed and implemented
– How to convert Geant4 to Geant4MT

Xin Dong and Gene Cooperma
High Performance Computing Lab
College of Computer and Information Science
Northeastern University
Boston, Massachusetts 02115
USA
{gene,xindong}@ccs.neu.edu

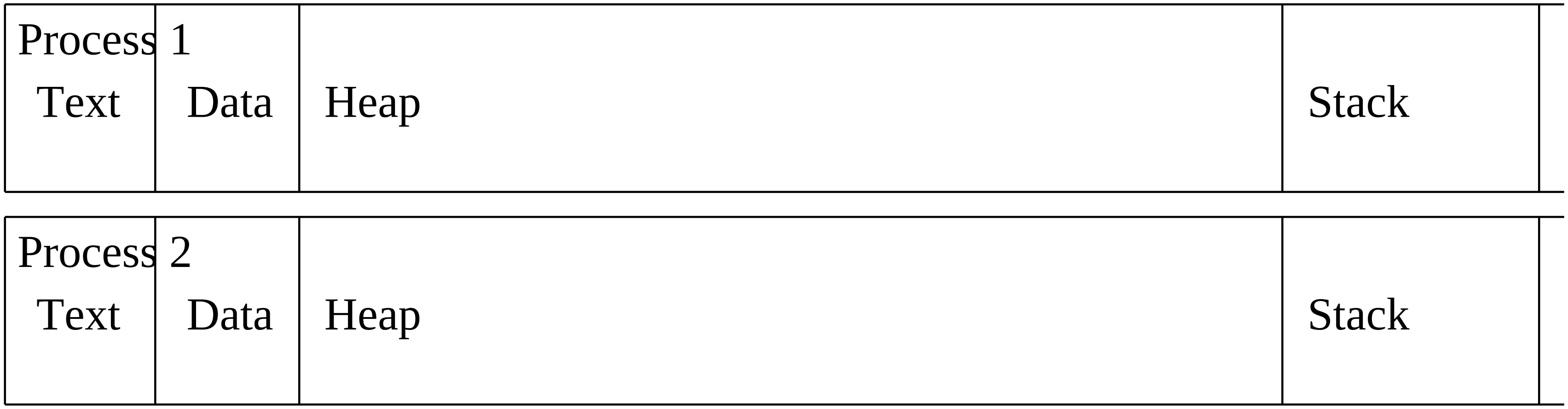
Geant4 MultiThreading Overview

Geant4 MultiThreading (Geant4MT)

- adopt the same event-level parallelism as the prior distributed memory parallelization has done
- replace k independent copies of the Geant4 process with an equivalent single process with k threads
- uses the many-core machine in a memory-efficient scalable manner
- modify both the source code of the Geant4 kernel and the source code of Geant4 applications
  - the code modification for thread safety
  - the code modification for memory footprint reduction
  - the code for the worker thread initialization
  - the thread private malloc library
  - the thread safe CLHEP interface
  - the parallelization frame code for applications

Geant4MT Thread Safety

Replace the following two Geant4 processes



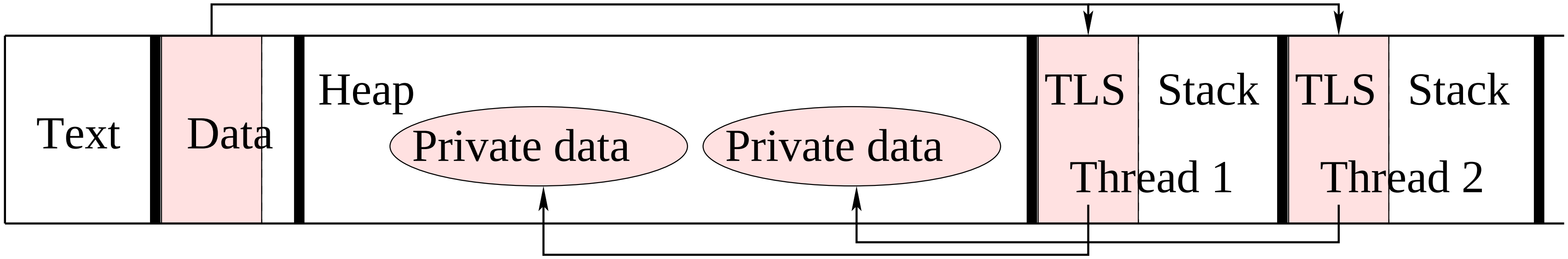
Process 1

Text Data Heap Stack

Process 2

Text Data Heap Stack

with one process with two Geant4 threads

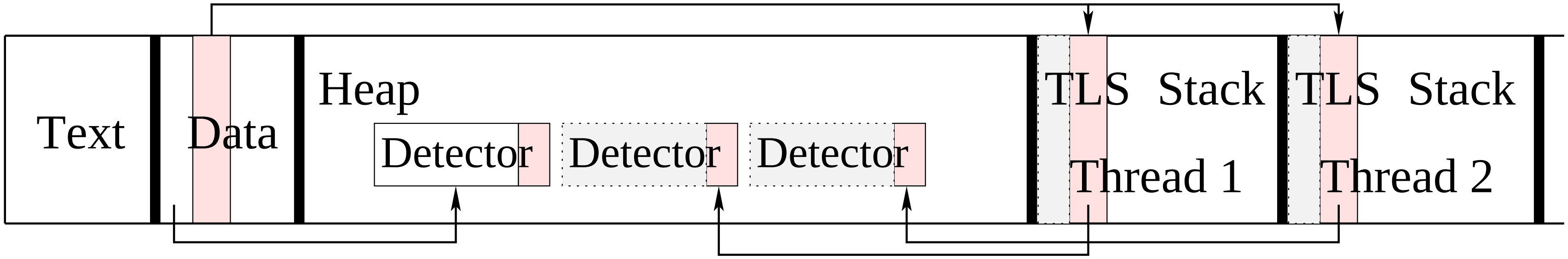


Text Data Heap Private data Private data TLS Stack Thread 1 TLS Stack Thread 2

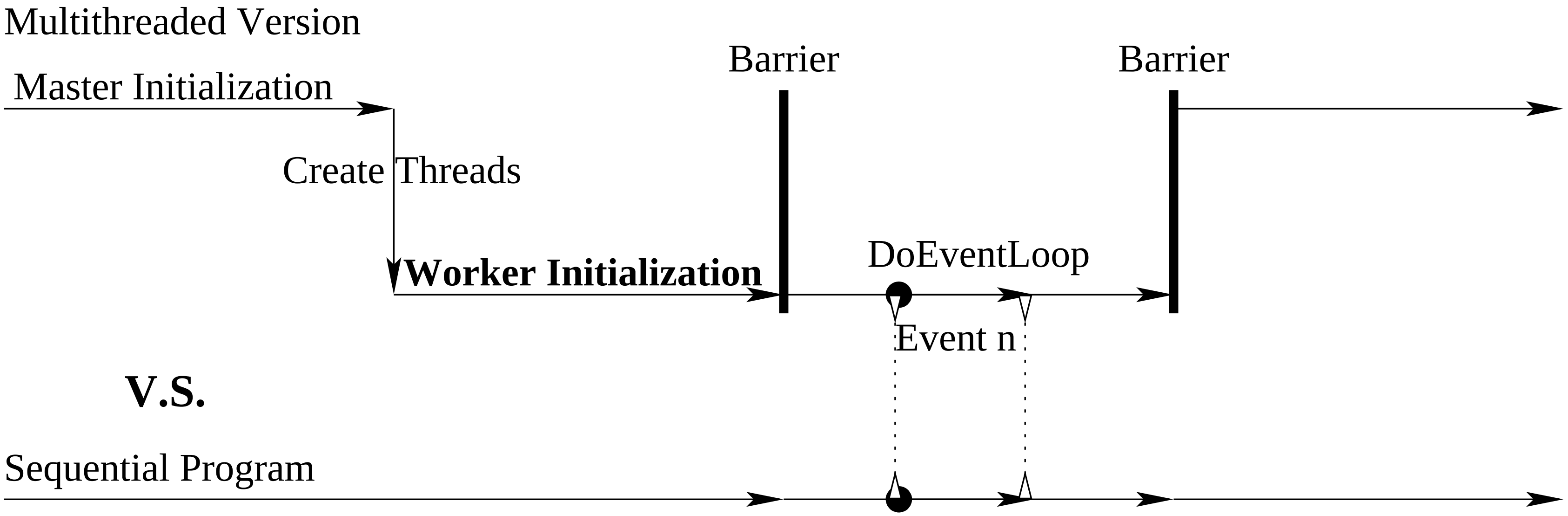
Geant4 detector is replicated by each thread. This leads to a thread-safe usage of C++ STL.

Geant4MT Memory Footprint Reduction

Implement the following data model

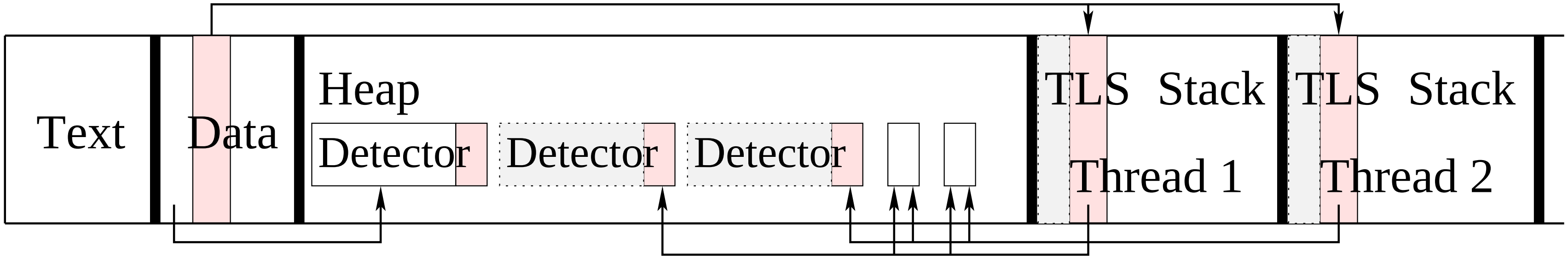


Because some detector data structure is changed, initialization must be changed correspondingly for threads.



Malloc: Central Heap Performance Bottleneck

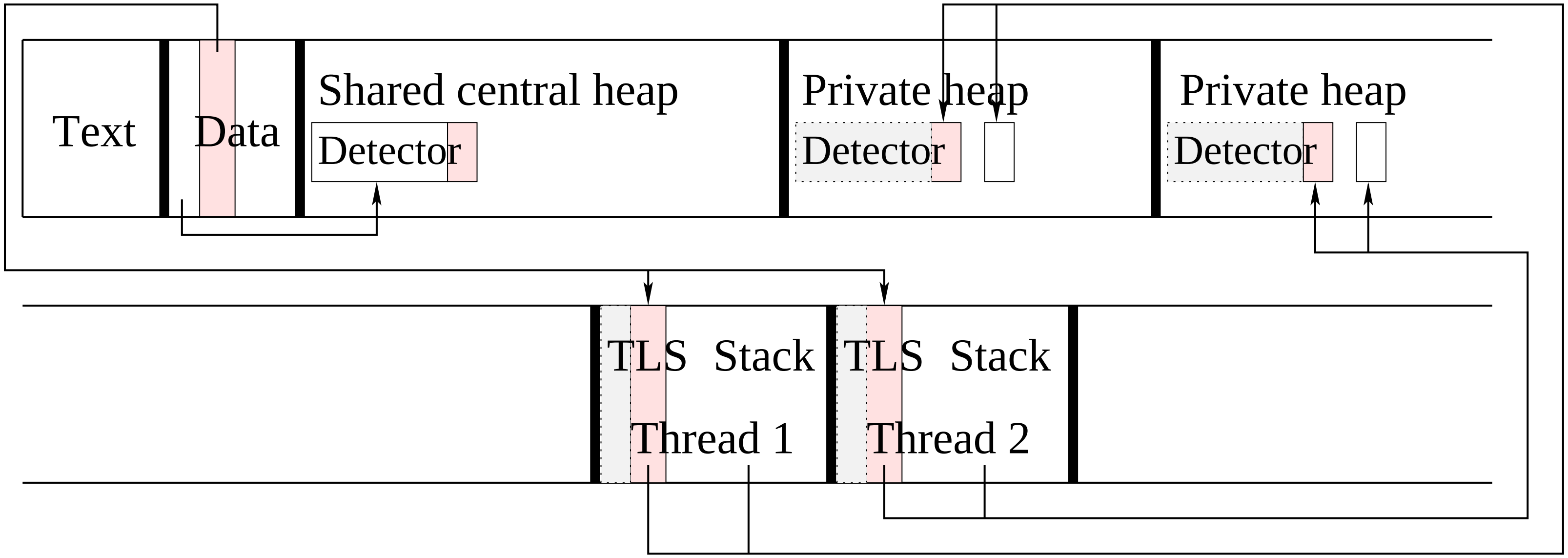
Even if memory allocation/deallocation consists of 10 to 20 instructions, their cost is not negligible for thread-level parallelism.



- memory chunks are maintained using a “boundary tag” method
  - allocation/deallocation generates random accesses to memory address space and more cache misses
- POSIX standard requires memory allocator to be thread safe
  - locks/unlocks in addition to cache coherence misses
- C++ string and STL containers implementation
  - intensive dynamic memory allocations and deallocations

Thread Private Allocator (TPMalloc)

Make the malloc state (arena) thread local and force each worker thread to mmap a large thread private region.



If a thread allocates memory, then the same thread will free it.

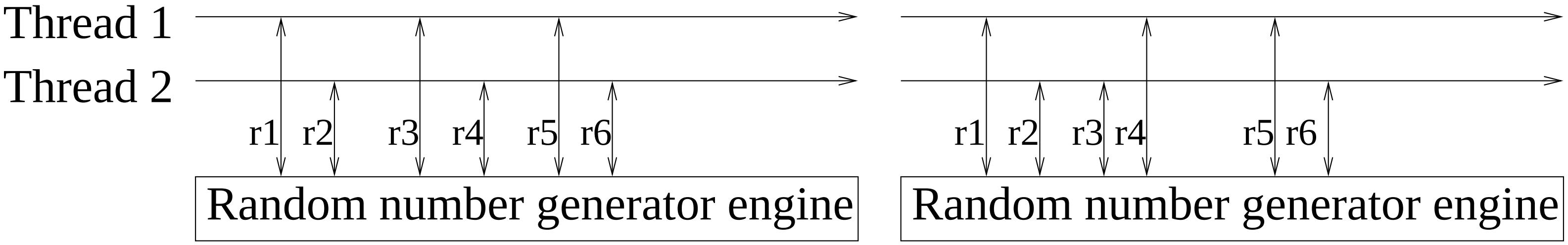
For the simulation phase when a huge amount of navigation history data is dynamically allocated.

Those history data is used temporarily and freed by the same thread.

Segregated thread private regions in the heap and completely lock-free

Thread Safe CLHEP Interface

If Geant4 threads invoke the same random number generator engine, then reproducibility is not guaranteed.



Thread 1

Thread 2

r1 r2 r3 r4 r5 r6

Random number generator engine

r1 r2 r3 r4 r5 r6

Random number generator engine

Case 1

Case 2

Case1: thread 1 got r1, r3, r5; thread 2 got r2, r4, r6

Case2: thread 1 got r1, r4, r5; thread 2 got r2, r3, r6

Since the CLHEP static interface is not stateless, G4MTHepRandom is implemented for Geant4MT to achieve reproducibility

- A multithreaded HepRandom class used as a per thread singleton
- The parent class for distribution classes leveraged from CLHEP

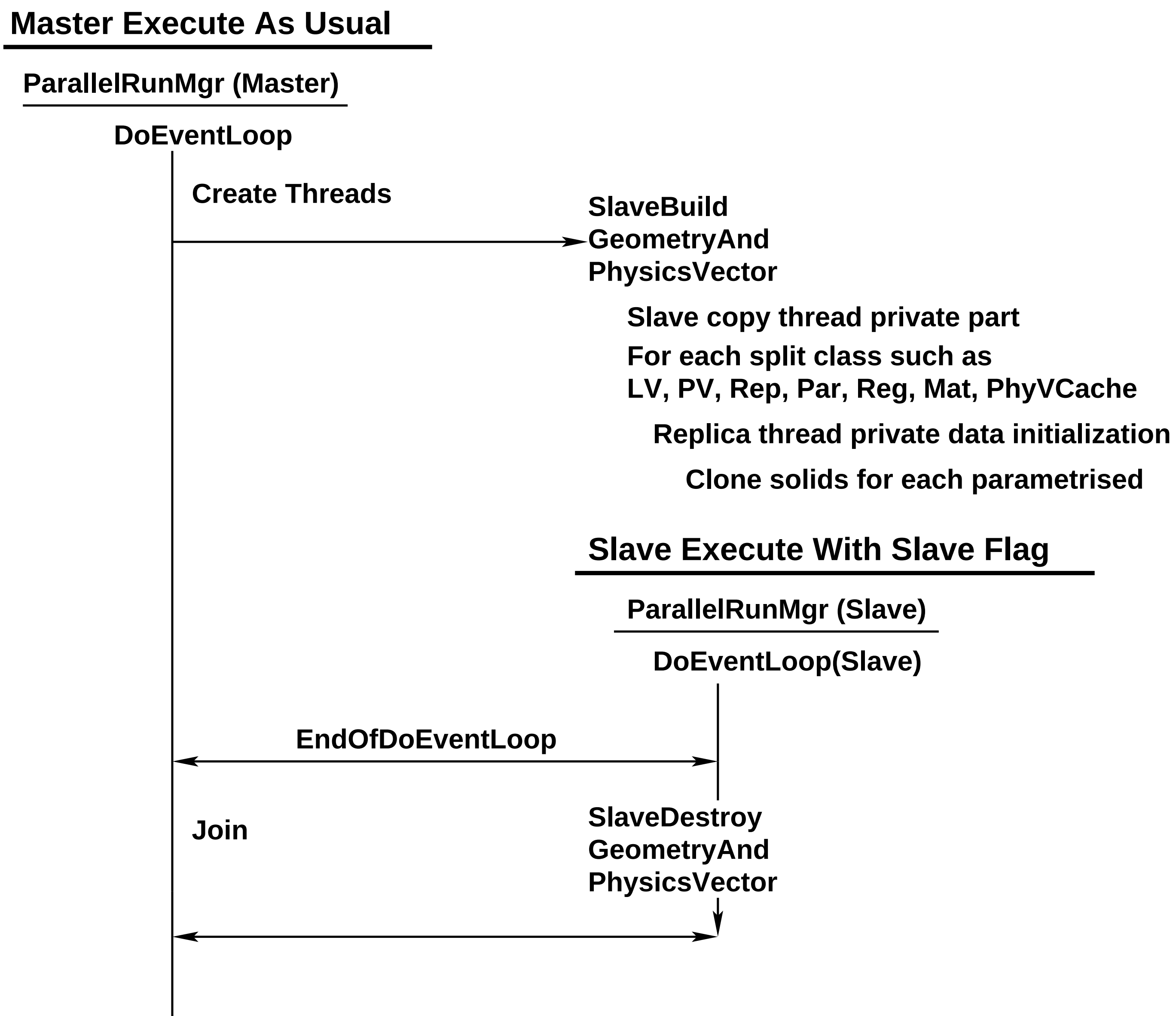
This change allows the Geant4MT to compile against the original CLHEP maintained outside of the Geant4 kernel.

Parallelization Frame Code for Applications

Geant4 applications are multithreaded in a fashion similar to the ParGeant4 for distributed memory clusters.

- A new main function and a thread function as wrappers
- Some minor change in the real application main function to coordinate master phase and worker phase initialization
- A parallel run manager and some modification in the DoEventLoop function to spawn worker threads
- User-defined organization for the parallel simulation of events and the aggregation for simulation results
- A child class for the class G4coutDestination, which has one per thread instance to redirect the output to a thread private file. This instance is associated to G4coutbuf and G4cerrbuf for output demangle.
- Debugging tools for errors introduced by the Geant4MT: incorrectly initialized worker threads; and data race generated by writing to some shared data.

Geant4MT Threads Life Time



Master Execute As Usual

ParallelRunMgr (Master)

DoEventLoop

Create Threads

SlaveBuild GeometryAnd PhysicsVector

Slave copy thread private part

For each split class such as LV, PV, Rep, Par, Reg, Mat, PhyVCache

Replica thread private data initialization

Clone solids for each parametrised

Slave Execute With Slave Flag

ParallelRunMgr (Slave)

DoEventLoop(Slave)

EndOfDoEventLoop

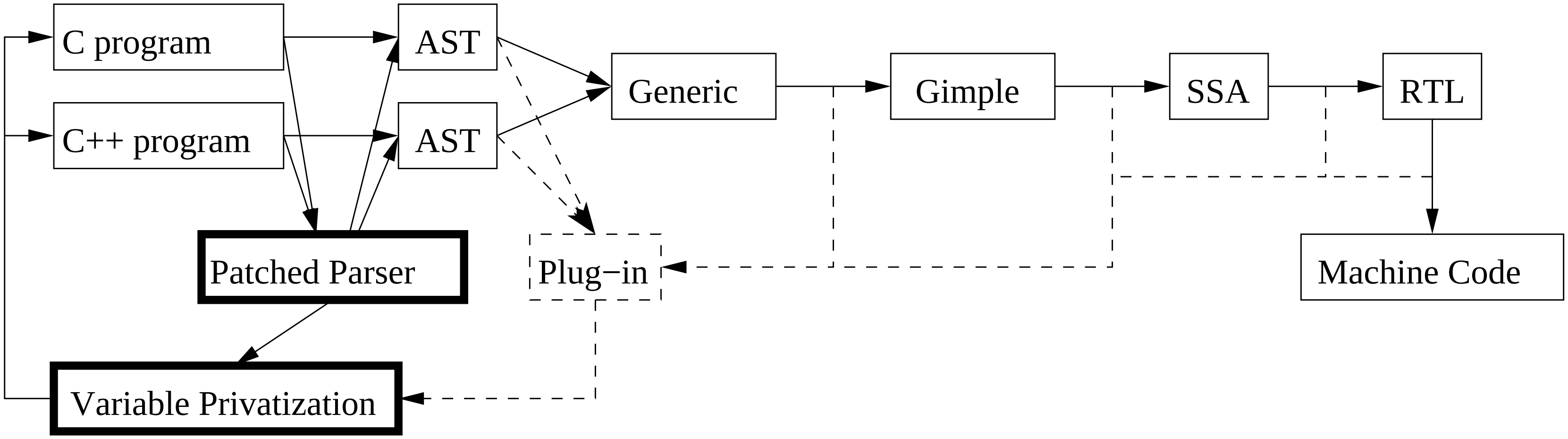
Join

SlaveDestroy GeometryAnd PhysicsVector

Geant4MT Tools for Implementation Support

- Transformation for Thread Safety (TTS)
  1. make each global or static variable thread-local
  2. independent threads lead to absolute thread-safety: any thread can call any function. No data race!
- Transformation for Memory Reduction (TMR)
  1. *relatively read-only data*: written to during its initialization and read-only during the computation of each task.
  2. share relatively read-only data, and replicate other data
- Debugging Tools
  1. compare the original program with the multi-threaded version
  2. runtime correctness: to serialize updates to shared data
- Malloc Non-standard Extension using a Thread-Private Heap (TPMalloc)
- Avoidance of Cache Coherence Bottlenecks

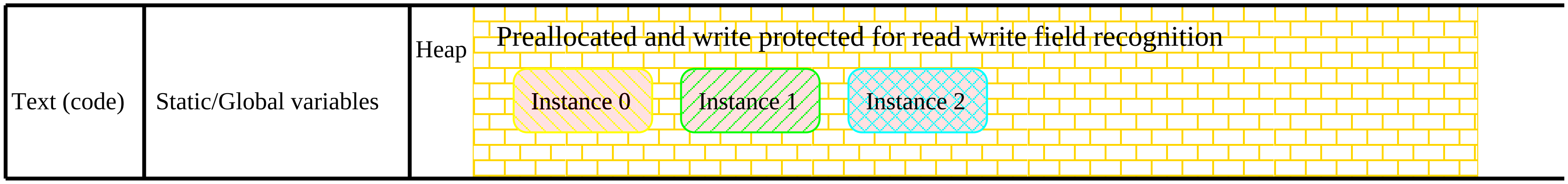
TTS Architecture



- Patch some code in C++ parser to recognize: global declarations and corresponding extern declarations; and static declarations
- Variable privatization is implemented via the ANSI C/C++ keyword `__thread` (since C99)
- LLVM Clang compiler supports plug-ins very well, which leads to a portable solution for the maintenance of TTS transformed program

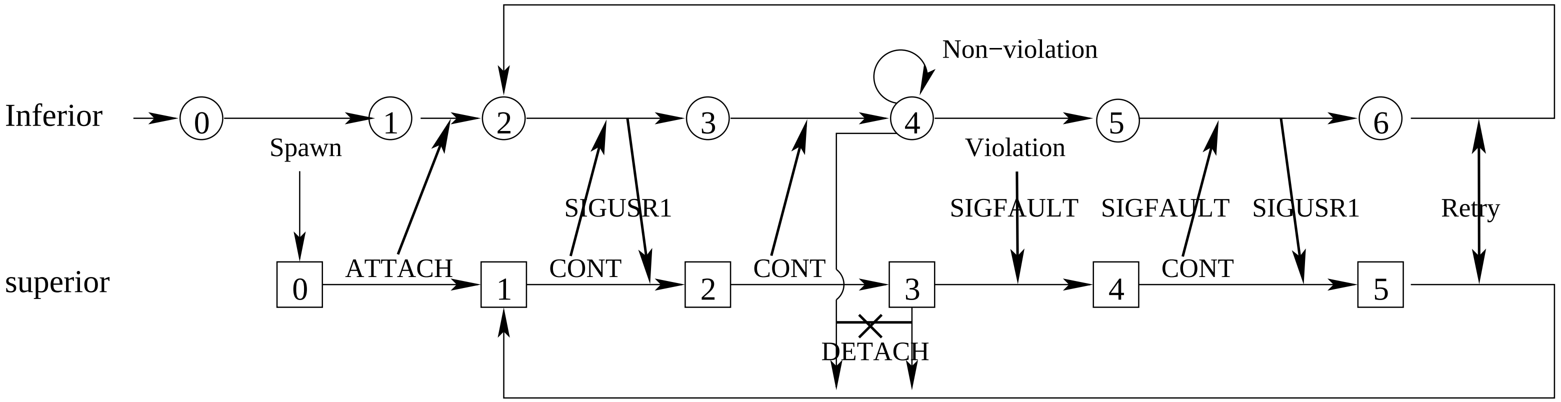
Transformation for Memory Reduction (TMR)

Is a large array of object instances relatively read only?



Put all sharable instances into a pre-allocated region in the heap via

- overloading the “new” method and the “delete” method



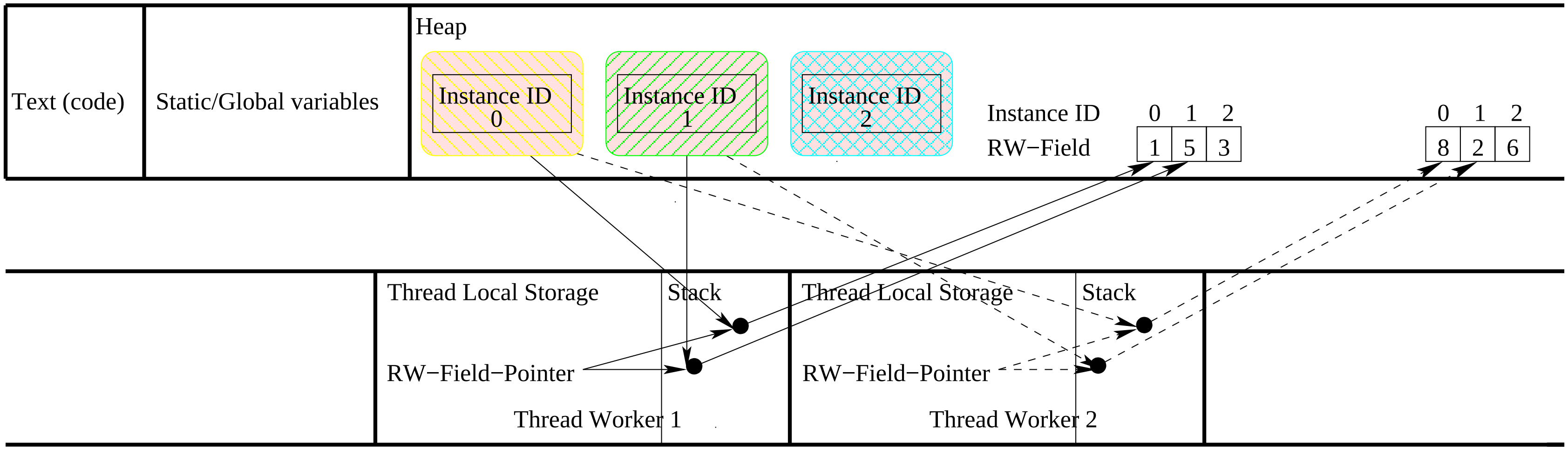
The superior takes advantage of memory write-protection and directs the execution of the inferior: remove “w”; catch segfault; re-enable “w” and retry the instruction.

TMR Implementation Example

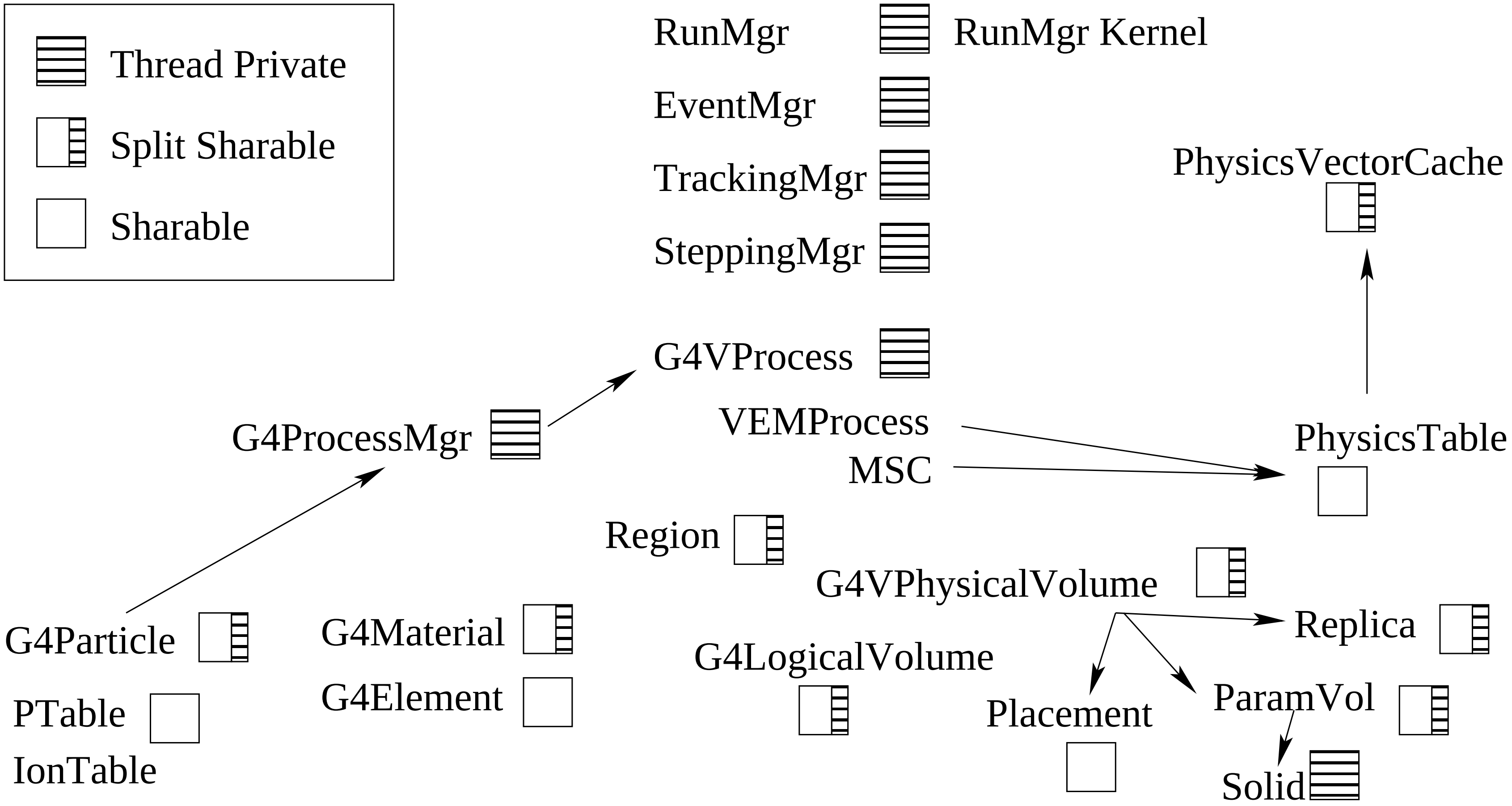
If those object instances are relatively read-only, just share them. Otherwise, reorganize the data structure as follows.

class volume
{
    RW_t rw;
    RD_t rd;
}
__thread vector<volume> store;*
	__thread RW_t *rw_array;
	class volume {
	    int instanceID;
	    RD_t rd;}
	#define rw (rw_array[instanceID])
	vector<volume> store;*

Corresponding Data Model



Geant4MT Sharable Classes



▤ Thread Private

◫ Split Sharable

□ Sharable

RunMgr ▤ RunMgr Kernel

EventMgr ▤

TrackingMgr ▤

SteppingMgr ▤

G4VProcess ▤

VEMProcess

MSC

PhysicsTable □

PhysicsVectorCache ◫

G4ProcessMgr ▤

Region ◫

G4VPhysicalVolume ◫

Replica ◫

ParamVol ◫

Placement □

Solid ▤

G4LogicalVolume ◫

G4Particle ◫

PTable □

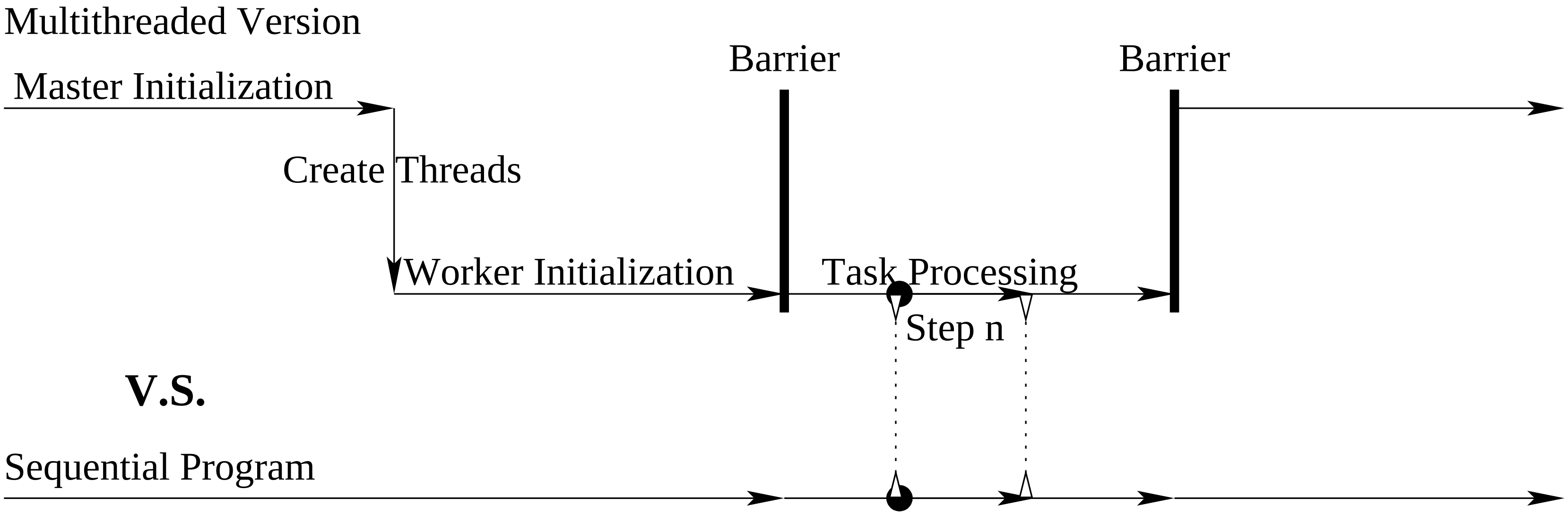
IonTable

G4Material ◫

G4Element □

Bisimulation: Debug for Single-Thread Correctness

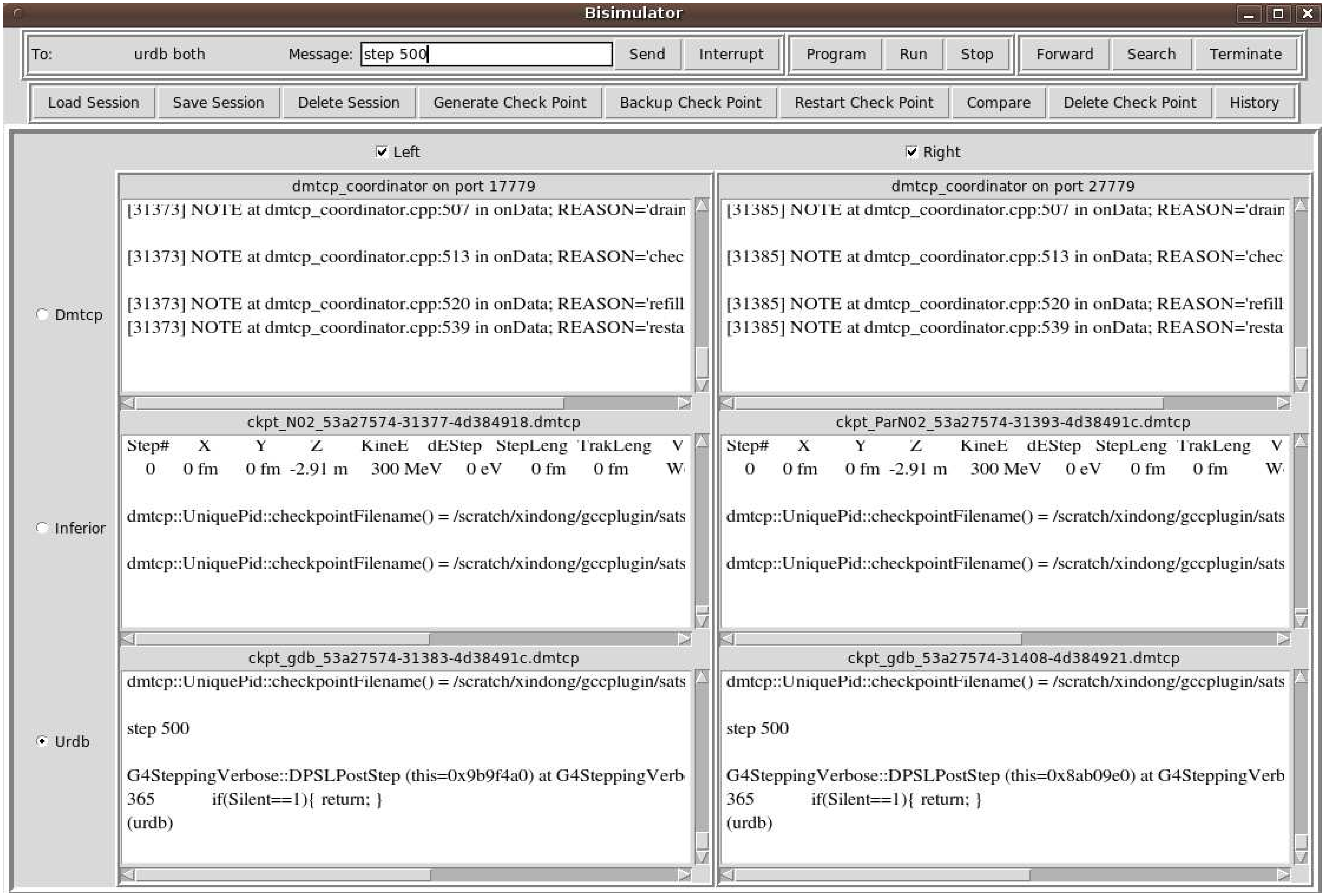
Because some classes are split, initialization must be changed correspondingly for threads. Bugs could be introduced into the initialization code.



In a similar spirit to proof by bisimulation, debug by comparison

- The comparison verifies the original program and the multi-threaded version against whether they behave identically.
- The problematic instruction that leads to a difference is the clue for the root cause of the bug.

Checkpointed Debug Sessions for Bisimulator



dmtcp_coordinator on port 17779

```
[31373] NOTE at dmtcp_coordinator.cpp:507 in onData; REASON='drain

[31373] NOTE at dmtcp_coordinator.cpp:513 in onData; REASON='chec

[31373] NOTE at dmtcp_coordinator.cpp:520 in onData; REASON='refill
[31373] NOTE at dmtcp_coordinator.cpp:539 in onData; REASON='resta
```

dmtcp_coordinator on port 27779

```
[31385] NOTE at dmtcp_coordinator.cpp:507 in onData; REASON='drain

[31385] NOTE at dmtcp_coordinator.cpp:513 in onData; REASON='chec

[31385] NOTE at dmtcp_coordinator.cpp:520 in onData; REASON='refill
[31385] NOTE at dmtcp_coordinator.cpp:539 in onData; REASON='resta
```

ckpt_N02_53a27574-31377-4d384918.dmtcp

```
Step#    X      Y       Z       KineE    dEStep  StepLeng  TrakLeng   V
  0    0 fm   0 fm  -2.91 m   300 MeV    0 eV     0 fm      0 fm     W

dmtcp::UniquePid::checkpointFilename() = /scratch/xindong/gccplugin/sats

dmtcp::UniquePid::checkpointFilename() = /scratch/xindong/gccplugin/sats
```

ckpt_ParN02_53a27574-31393-4d38491c.dmtcp

```
Step#    X      Y       Z       KineE    dEStep  StepLeng  TrakLeng   V
  0    0 fm   0 fm  -2.91 m   300 MeV    0 eV     0 fm      0 fm     W

dmtcp::UniquePid::checkpointFilename() = /scratch/xindong/gccplugin/sats

dmtcp::UniquePid::checkpointFilename() = /scratch/xindong/gccplugin/sats
```

ckpt_gdb_53a27574-31383-4d38491c.dmtcp

```
dmtcp::UniquePid::checkpointFilename() = /scratch/xindong/gccplugin/sats

step 500

G4SteppingVerbose::DPSLPostStep (this=0x9b9f4a0) at G4SteppingVerb
365         if(Silent==1){ return; }
(urdb)
```

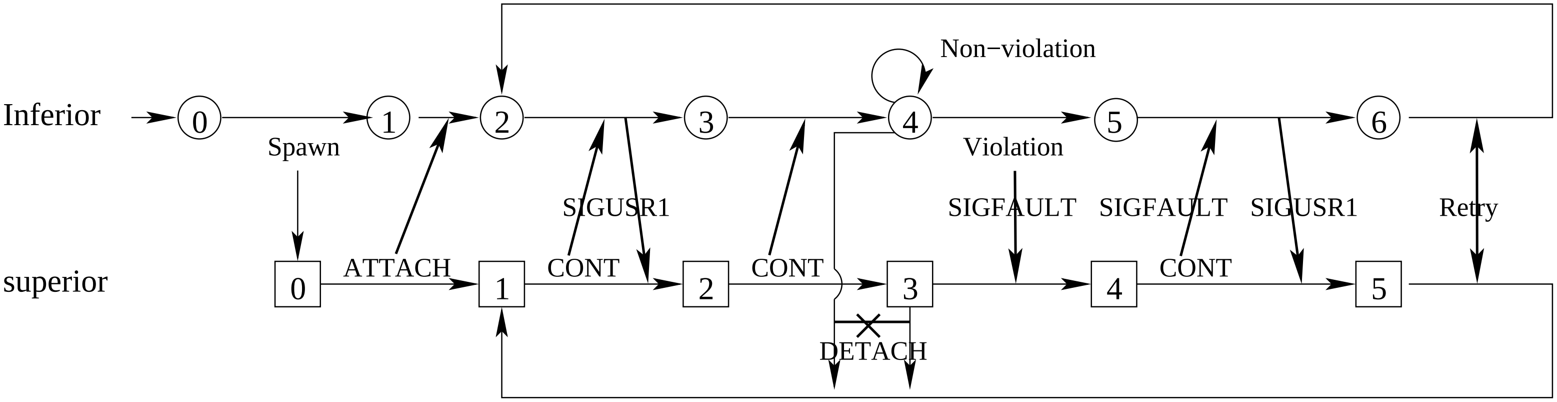
ckpt_gdb_53a27574-31408-4d384921.dmtcp

```
dmtcp::UniquePid::checkpointFilename() = /scratch/xindong/gccplugin/sats

step 500

G4SteppingVerbose::DPSLPostStep (this=0x8ab09e0) at G4SteppingVerb
365         if(Silent==1){ return; }
(urdb)
```

Run-time: Correctness Guarantee

A deeper error: shared “relatively read-only” is changed by task processing

- Run-time correctness to serialize unintended updates to shared data
  - Independence of tasks: Geant4MT’s task-oriented nature
  - Memory write-protection



  - Data Race Coordinator (DRC) recovery policy
    - ∗ thread that updates to shared “relatively read-only” is suspended
    - ∗ remaining threads finish their current task and become quiescent
    - ∗ quiescent threads wait upon the suspended thread to finish

Geant4 MultiThreading – History

- ParGeant4 for cluster: master/worker, event-level, seed per event ...
- October 2007, Geant4 parallelization for many-core CPUs
- January 2008, thread safe Geant4.9.0 via manually changing
- March 2008, transformation tools for thread safety (TTS)
- December 2008, transformation for sharing detector data (TMR)
- April 2009, Geant4MT for Geant4.9.1: performance bottleneck
- July 2009, Geant4MT for Geant4.9.2: performance bottleneck
- October 2009, performance improvement: TPMalloc
- December 2009, performance improvement: Thread Private Output
- March 2010, Debugging tool Bisimulation
- September 2010, Geant4MT for Geant4.9.4.b01
- December 2010, Geant4MT for Geant4.9.4
- March 2011, Geant4MT for Geant4.9.4.p01

Geant4MT for Next Releases I

Generally a three-day work for each release:

- Install a sequential version of the new release
  - use patched gcc 4.2.2 and install to another directory -geant4 (keep the source clear)
  - turn on debug mode -g and enable GDML
- TMR patch: If patched successfully, shared classes and corresponding initialization code are not changed much.
- delete /tmp/geant4* and recompile
- TTS
  - ./elsa-2005.08.22bG4MT/elsa/geant4mtphase2 /tmp/geant4static
  - ./elsa-2005.08.22bG4MT/elsa/geant4mtphase2 /tmp/geant4global
  - ./elsa-2005.08.22bG4MT/elsa/geant4mtphase2 /tmp/geant4extern
  - ./elsa-2005.08.22bG4MT/elsa/geant4mtphase2

Geant4MT for Next Releases II

- Combine the change in the source directory and the destination directory
  - cp G4Integrator.icc
    from ./geant4/src/geant4/source/global/HEPNumerics/include
    to ./geant4.9.4.b01/source/global/HEPNumerics/include
  - cp G4ReferenceCountedHandle.hh
    from ./geant4/src/geant4/source/global/management/include
    to ./geant4.9.4.b01/source/global/management/include
- TTS may introduce some statements into methods where the transformed static member is not really used. It derives lots of warnings. To eliminate compiler warnings:
  - change /geant4/src/geant4/config/architecture.gmk
    ...
    $CPPFLAGS+ =>> /tmp/Geant4MTWarning\,2 \; > \; \&\,1$
  - recompile
  - use a script to delete those useless statements following warnings

Geant4MT for Next Releases III

- Patch the CLHEP thread safe interface
  RandFlat::shoot G4RandFlat
  RandGamma::shoot G4RandGamma
  RandBit::shootBit G4RandBit
  RandExponential::shoot G4RandExponential
  RandFlat::shootArray G4RandFlatArray
  RandFlat::shootInt G4RandFlatInt
- Change global/HEPRandom/include/Randomize.hh
  #define G4RandFlat G4MTRandFlat::shoot
  #define G4RandGamma G4MTRandGamma::shoot
  #define G4RandBit G4MTRandBit::shootBit
  #define G4RandExponential G4MTRandExponential::shoot
  #define G4RandFlatArray G4MTRandFlat::shootArray
  #define G4RandFlatInt G4MTRandFlat::shootInt
- Compile applications and test: Parallel A01, B01, ParN02, ParScorer
- Anything wrong, change and retry. Finally, go back to the second step.

What is Patch I

Sharable Class	Private Data Manager Template	Initial 0 ?	Additional Processing
G4LogicalVolume	G4MTTransitory	No	
G4VPhysicalVolume	G4MTTransitory	No	If Replics, SlaveG4PVReplica() If Parameterised, Solid clone SlaveG4LogicalVolume(clone)
G4PVReplica	G4MTTransitory	No	
G4ParticleDefinition	G4MTTransitoryParticle	Yes	
G4Region	G4MTTransitory	Yes	
G4Material	G4MTTransitory	No	Per Instance SlaveG4Material()
G4PhysicsVectorCache	G4MTTransitoryPhysicsVector	Yes	
Thread private fields are indexed by instanceID. Fields and instanceID grow only. Even if an instance is deleted, its instanceID and thread private fields are still there. Only ions and physics vectors are allowed to be created within DoEventLoop. For the reason above, their template is different.			
Template methods: 1. AddNew(): grow the array for private fields 2. SlaveCopy(): memory copy of the array for private fields from the master 3. AllocateSlave(): per object offset[i].initialize() 4. FreeSlave(): deallocate the array for private fields			

What is Patch II

Initialize physics processes created for the worker

- Change the SetPhysics method for the original G4RunManagerKernel class
  - Skip ConstructParticles
  - Process manager will be reinitialized, because it is thread private

Initialize physics tables for the class G4VEnergyLossProcess or the class G4VMultipleScattering

- Change three methods AddProcessManager, BuildPhysicsTable and PreparePhysicsTable
  1. master thread keeps a shadow process manager pointer firstProcess
  2. worker thread calls SlaveBuildPhysicsTable(firstProcess).
  3. otherwise (*pVector)[j]->BuildPhysicsTable(*particle);

What is Patch III

```
diff -Naur source/geometry/navigation/src/G4Navigator.cc
patched/source/geometry/navigation/src/G4Navigator.cc

- static G4double fAccuracyForWarning = kCarTolerance,
- fAccuracyForException = 1000*kCarTolerance;
+ static G4double fAccuracyForWarning = kCarTolerance;
+ static G4double fAccuracyForException = 1000*kCarTolerance;
```

```
diff -Naur source/graphics_reps/src/BooleanProcessor.src
patched/source/graphics_reps/src/BooleanProcessor.src

- static void set_shift(int); //G.Barrand
+ static void set_shift(int a_shift); //G.Barrand
```

What is Patch IV

```
diff -Naur source/processes/cuts/include/G4ProductionCuts.hh
patched/source/processes/cuts/include/G4ProductionCuts.hh

- static const G4ParticleDefinition* gammaDef;
- static const G4ParticleDefinition* electDef;
- static const G4ParticleDefinition* positDef;
+ static G4ParticleDefinition* gammaDef;
+ static G4ParticleDefinition* electDef;
+ static G4ParticleDefinition* positDef;

- static const G4ParticleDefinition* protonDef; // for proton
+ static G4ParticleDefinition* protonDef; // for proton
```

Questions

Thank You.