

Multithreaded Simulation for ATLAS: Challenges and Validation Strategy

CHEP 2019

Adelaide

5th November 2019

Marilena Bandieramonte

John Derek Chapman

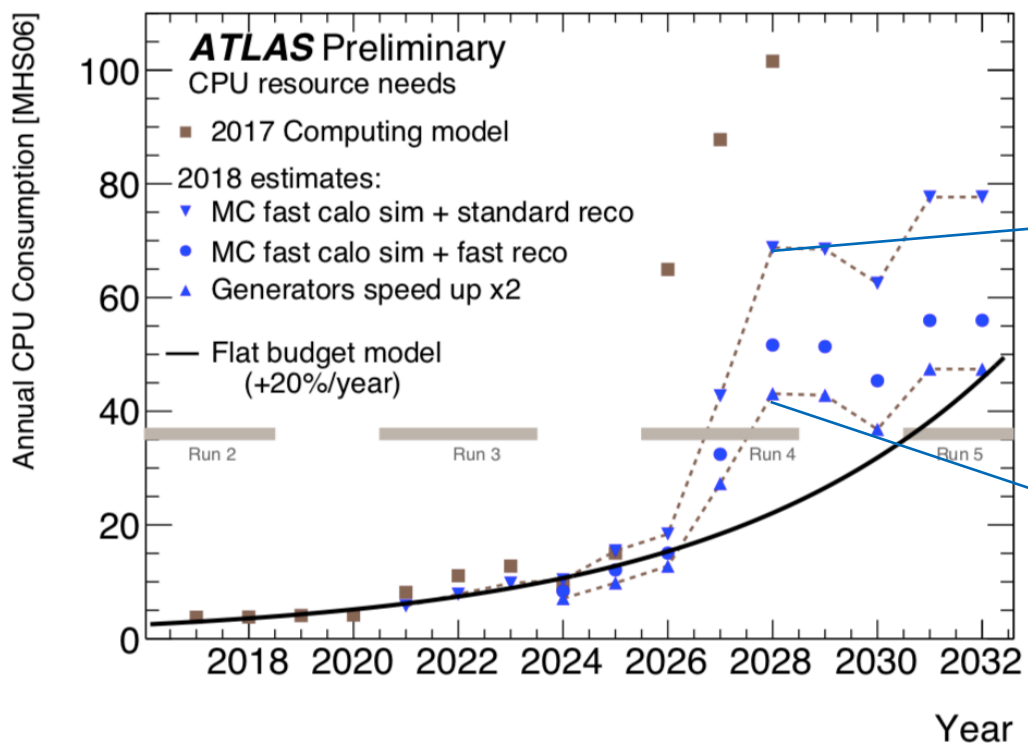
Heather Gray

Miha Muskinja

Yu Him Justin Chiu

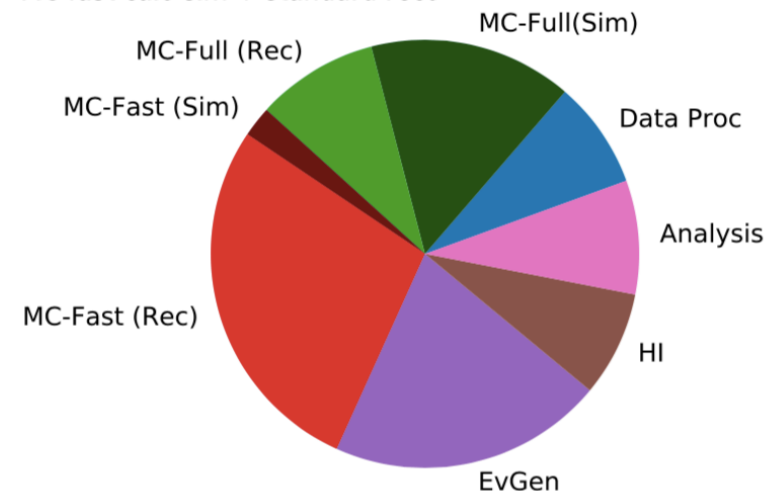
on behalf of the ATLAS collaboration



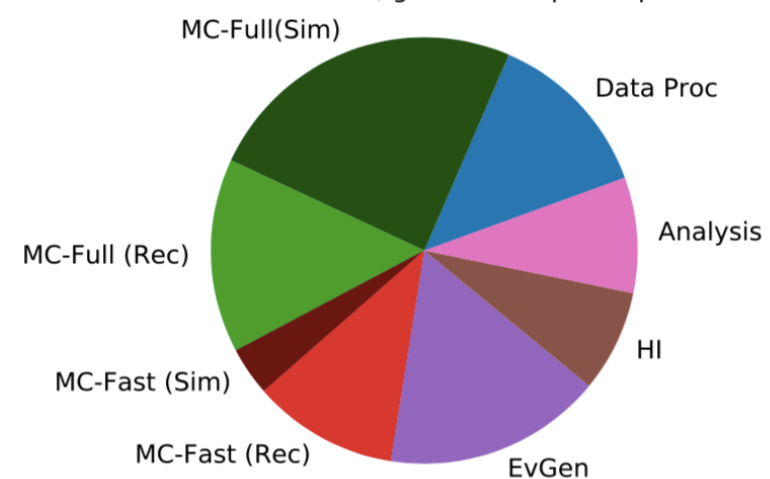


[ATLAS public]

ATLAS Preliminary. 2028 CPU resource needs
MC fast calo sim + standard reco



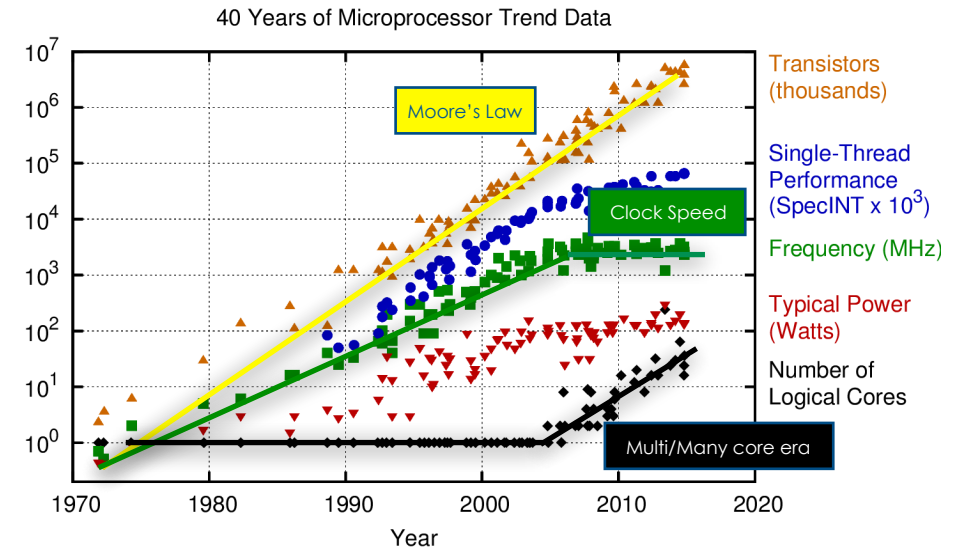
ATLAS Preliminary. 2028 CPU resource needs
MC fast calo sim + fast reco, generators speed up x2



- In Run3, we plan to run at least 50% of simulation with fast techniques (we aim to reach ~75%), but full **Geant4 simulation** will be heavily used regardless
- In Run 4, Full Simulation is expected to be the largest CPU consumers (20-25%)
 - Together with FastSim and FastReco it amounts to ~40% of all expected CPU consumption.
- Any **performance optimizations** of ATLAS simulation have a big impact on the overall picture.

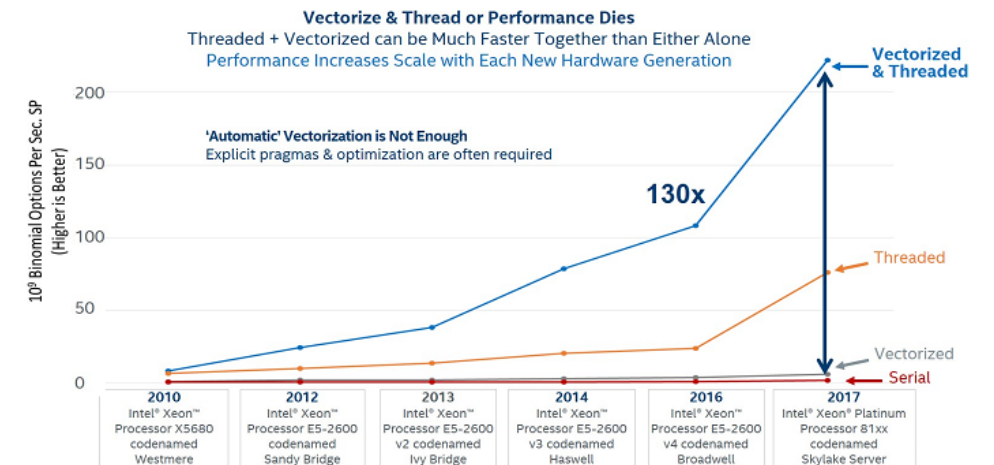
Why multi-threading?

- Evolution trend of faster single-threaded CPU performance broken **more than 10 years ago**
- Increase of **CPU cores** and more execution units to overcome stagnation in CPU Clock Speed
 - low power core sharing a pool of memory
- **We need a 'Multi-Threaded design (AthenaMT)'** to run effectively on modern architectures and profit from multi-core designs
 - MT approach is critical for heterogeneous architectures (e.g. GPU HPCs)
 - This approach will scale better than the existing multi-processor approach (AthenaMP) especially on the architectures that are foreseen to be used in the next LHC runs
- Production ready MT simulation is considered **CRITICAL** for Run 3 and **BLOCKER** for Run 4
 - to exploit the HL-LHC successfully
- What about **vectorization**?



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

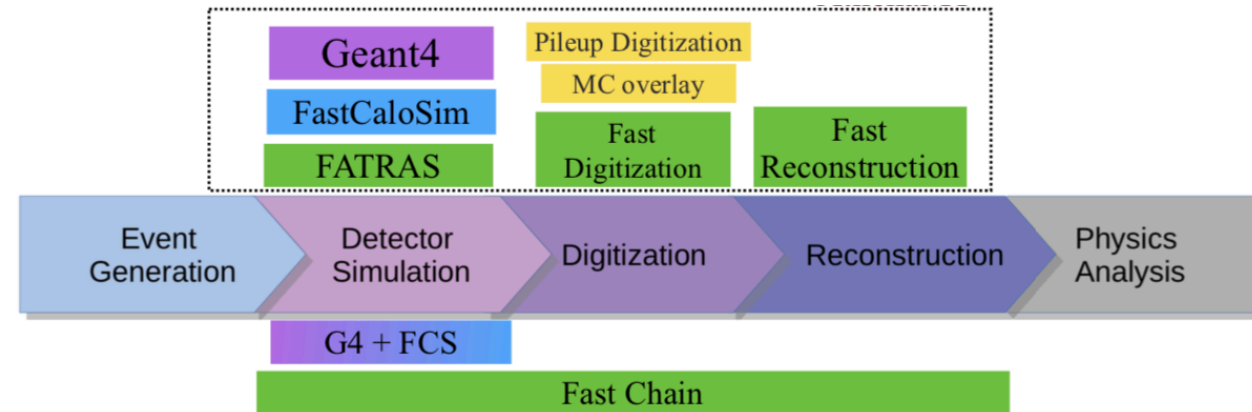
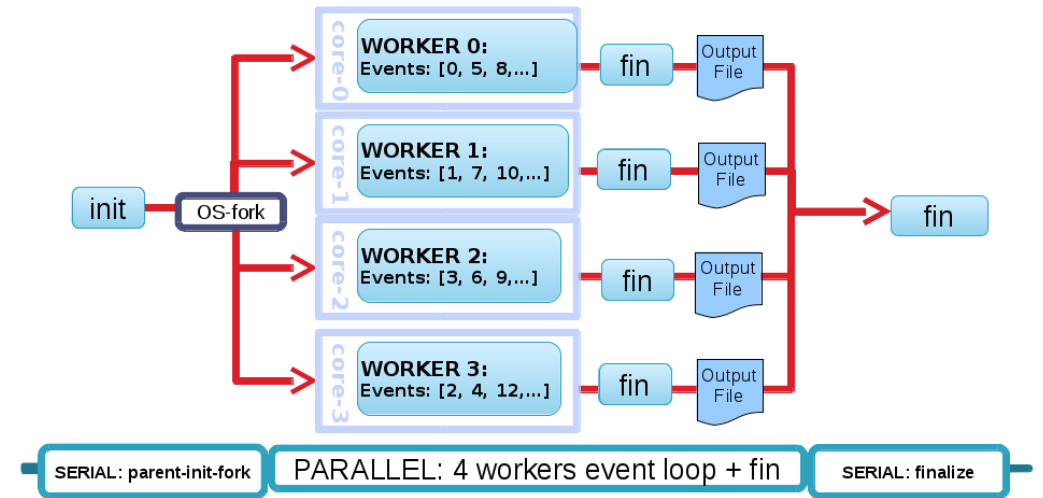
History of Intel chip introductions by clock speed and number of transistors



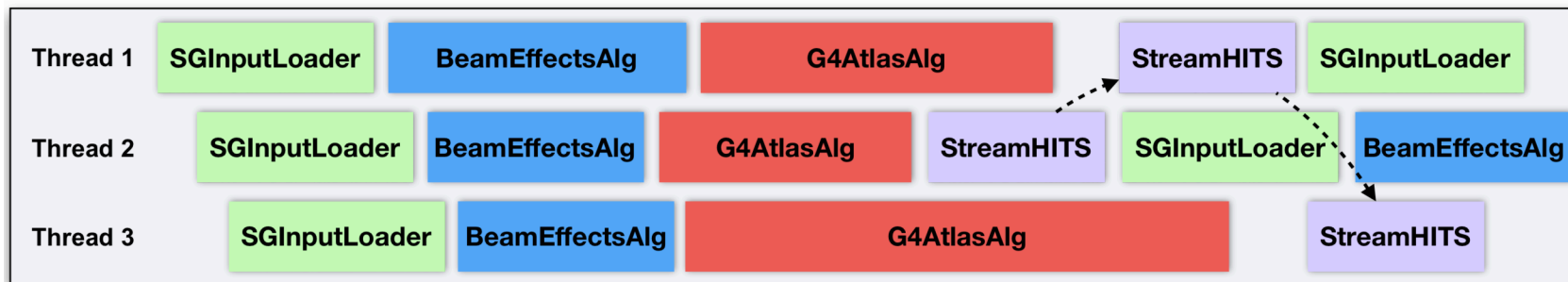
Why multi-threading?

- The amount of **Monte-Carlo** that can be produced already **limits many physics analyses** and this will get **worse** with the increased luminosity expected
- The current model, **AthenaMP**, relies on Linux's *copy-on-write* mechanism for sharing memory pages between forks:
 - won't scale for Run-3 and beyond
- Ongoing effort to migrate **ATLAS** computing model to **multi-threaded AthenaMT**
 - Finer-grained *task parallelism*, minimised memory footprint
 - Only `execute()` is concurrent
 - *Scheduler-driven*, by dependency graph
- **Simulation, Digitization and Reconstruction** moving to MT paradigm using the AthenaMT/GaudiHive infrastructure.
 - Better scaling in terms of memory footprint (leverage new architectures)
 - Easy the investigation of heterogeneous computing architectures (e.g. use GPUs, FPGAs etc)

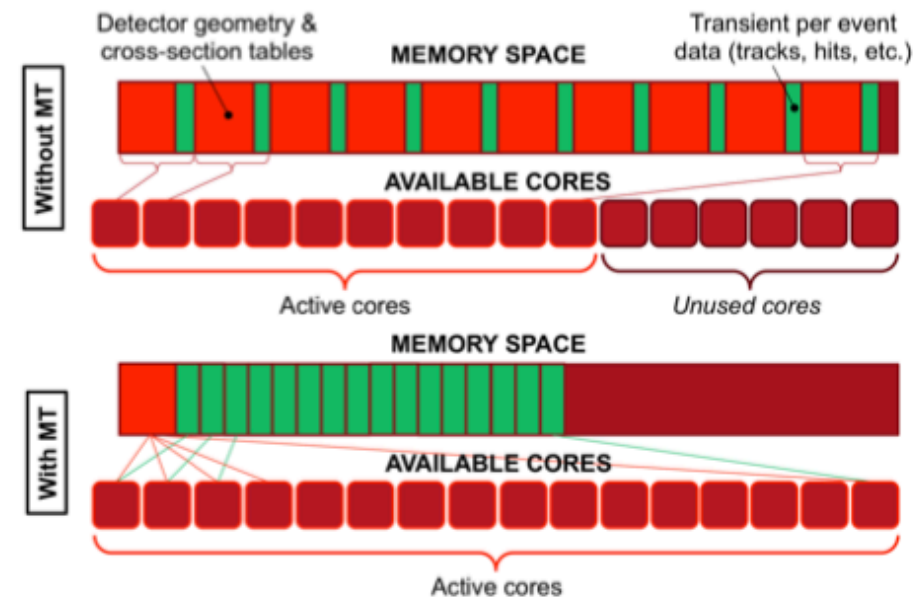
Schematic View of ATLAS AthenaMP



- **AthenaMT** is based on **GaudiHive**, a multi-threaded, concurrent-execution extension to Gaudi :
 - Concurrency model based on **Intel® Threading Building Blocks library (TBB)**
 - Computation is broken down into **tasks (building blocks)** that can run in parallel
 - Scheduling is driven by data-flow
 - Events processed in multiple threads
- **Geant4** has its own approach to parallel processing
 - Master-slave concurrency model, using **pthread**s
 - Provides event-level parallelism
 - Thread safety achieved using **thread-local storage**
 - Main Geant4MT components must be thread-local
- **GaudiHive** provides **task locality, not thread locality**
 - Cannot easily pin a Gaudi component to a specific thread
 - Must decouple the Gaudi components from the Geant4 core functionality
 - Initialization is very tricky: G4 requires that thread-local objects are initialized in their threads at the right time

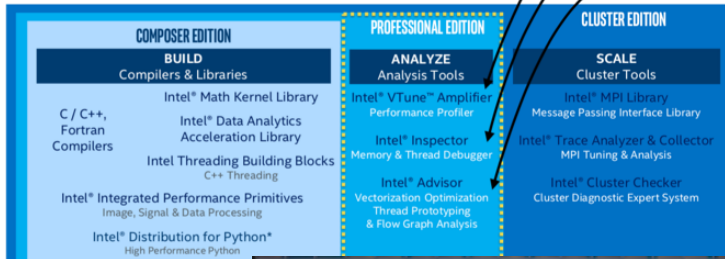


- **Geant4MT** has been successfully integrated in **AthenaMT** outside of the **Integrated Simulation Framework (ISF)**
 - Inter-event rather than intra-event parallelism:
 - memory saving coming from sharing geometry and cross-section tables between threads
- **Segfaults** during execution or finalization of MT jobs, due to the way **TBB** starts new threads:
 - During **execution** of a MT job:
 - **TBB can spawn new threads** even after initialization is complete
 - The simulation was aborted because the geometry was released after the initialization but it is always needed to initialize new threads
 - When **finalizing** a MT job:
 - TBB creates extra-threads that are not caught by the ThreadPoolSvc -> no call to G4ThreadInitTool::initThread
 - Crashes when G4ThreadInitTool::terminateThread is called for those threads



Parallel Studio XE

What's Inside Intel® Parallel Studio XE
Comprehensive Software Development Tool Suite



Overall Performance
Deadlocks etc.
Vectorization etc.



```
[mbandier@p05614910w96644 g4mt_tests]$ inspxe-cl -help
Intel(R) Inspector Command Line tool
Copyright (C) 2009-2018 Intel Corporation. All rights reserved.

Usage: inspxe-cl <-action> [-action-option] [-global-option] [--] target [target options]]

Type 'inspxe-cl -help <action>' for help on a specific action.

Available actions:
collect
collect-with
command
convert-suppression-file
create-breakpoints
create-suppression-file
export
finalize
help
import
knob-list
merge-states
report
version

[mbandier@p05614910w96644 g4mt_tests]$ inspxe-cl -help collect
-c, -collect=<string>          Choose an analysis type.

Usage: inspxe-cl -collect=<string> [-action-option] [-global-option] [--] <target> [<target options>]

<string> gives the analysis type to perform on <target>. Use -knob-list to
list the knobs for an analysis type. Available analysis types:

Name  Description
m11   Detect Leaks
m12   Detect Memory Problems
m13   Locate Memory Problems
t11   Detect Deadlocks
t12   Detect Deadlocks and Data Races
t13   Locate Deadlocks and Data Races

Action options:
-app-working-dir=<string>    Specify a directory where the application will be
```

```
[mbandier@p05614910w96644 g4mt_tests]$ inspxe-cl -collect ti2 -knob stack-depth=16 -- python /build2/mb/build/install/AthSimulation/22.0.0/Instal
lArea/x86_64-slc6-gcc62-opt/bin/athena.py "--threads=4" "runargs.AtlasG4Tf.py" "SimuJobTransforms/skeleton.EVGENToHIT_MC12.py"
```

- Collection of **data races** detected in AthenaMT simulation with **Intel Inspector**:

ATLASSIM-3991	Data race 1 - read/write		OPEN
ATLASSIM-3992	Data race 2 - read/write		OPEN
ATLASSIM-3993	Data race 3 - read/write		OPEN
 ATLASSIM-3994	Data race 4 - read/write		RESOLVED
ATLASSIM-3995	Data race 5 - read/write		OPEN
ATLASSIM-3996	Data race 6 - read/write		OPEN
 ATLASSIM-3997	Data race 7 - read/write		RESOLVED
ATLASSIM-3998	Data race 8 - read/write		OPEN
ATLASSIM-3999	Data race 9 - read/write		OPEN
ATLASSIM-4005	Data race 10 - read/write		OPEN
ATLASSIM-4015	Data race 11 - write/write - libRIO.so and AtlasFieldSvc		OPEN
ATLASSIM-4016	Data race 12 - write/write - libG4AtlasAlgLib and libGaudiCoreSvc		OPEN

Problem:

```
static const G4String tileVolumeString("Tile");
```

Was not thread-safe, substituted with:

```
static const char * const tileVolumeString = "Tile" ;
```

That is initialised before the first call

Description

Data race of read-write type:

```
Description: Read
Source: char_traits.h:262
Function: compare
Module: libstdc++.so.6
Variable: tileVolumeString

Call Stack:
libstdc++.so.6!compare - char_traits.h:262
libTileGeoG4SDLib.so!find - basic_string.h:2025
libTileGeoG4SDLib.so!ProcessHits - TileGeoG4SD.cc:61
libG4tracking.so!Hit - G4VSensitiveDetector.hh:122
libG4tracking.so!ProcessOneTrack - G4TrackingManager.cc:126
libG4event.so!DoProcessing - G4EventManager.cc:185
libG4AtlasAlgLib.so!ProcessEvent - G4AtlasWorkerRunManager.cxx:179
libG4AtlasAlgLib.so!execute - G4AtlasAlg.cxx:325
libGaudiPythonLib.so!execute - Algorithm.h:51
libGaudiKernel.so!sysExecute - Algorithm.cpp:496
libGaudiHive.so!execute - AlgoExecutionTask.cpp:60
```

```
Description: Write
Source: char_traits.h:290
Function: copy
Module: libTileGeoG4SDLib.so
Variable: tileVolumeString

Code snippet:
288     if (__n == 0)
289         return __s1;
>290     return static_cast<char_type*>(__builtin_memcpy(__s1, __s2, __n));
291     }
292
```

```
Call Stack:
libTileGeoG4SDLib.so!copy - char_traits.h:290
libTileGeoG4SDLib.so!ZN8G4StringC4EPKc - G4String.icc:39
libTileGeoG4SDLib.so!ProcessHits - TileGeoG4SD.cc:61
libG4tracking.so!Hit - G4VSensitiveDetector.hh:122
libG4tracking.so!ProcessOneTrack - G4TrackingManager.cc:126
libG4event.so!DoProcessing - G4EventManager.cc:185
libG4AtlasAlgLib.so!ProcessEvent - G4AtlasWorkerRunManager.cxx:179
libG4AtlasAlgLib.so!execute - G4AtlasAlg.cxx:325
libGaudiPythonLib.so!execute - Algorithm.h:51
```


- Collection of **lock hierarchy violations** detected in AthenaMT simulation with **Intel Inspector**:
- It happens when two threads are trying to access and lock two critical sections in a different order. Possible deadlock.

Thread 1:

```
Description: Lock owned
Source: gthr-default.h:748
Function: __gthread_mutex_lock
Module: libAthAllocators.so
Variable: block allocated at ArenaBase.cxx:148
```

Code snippet:

```
746 {
747     if (__gthread_active_p ())
>748         return __gthrw_(pthread_mutex_lock) (__mutex);
749     else
750         return 0;
```

Call Stack:

```
libAthAllocators.so!__gthread_mutex_lock - gthr-default.h:748
libAthAllocators.so!allocator - ArenaBase.icc:40
libGeneratorObjectsTPCnv.so!_ZN2SG21ArenaHandleBaseAllocTINS_18ArenaPoolA
libGeneratorObjectsAthenaPoolPoolCnv.so!createTransient - TPConverter.icc
libGeneratorObjectsAthenaPoolPoolCnv.so!PoolToDataObject - T_AthenaPoolCu
libAthenaPoolCnvSvcLib.so!createObj - AthenaPoolConverter.cxx:68
libAthenaBaseComps.so!makeCall - AthCnvSvc.cxx:565
libAthenaBaseComps.so!createObj - AthCnvSvc.cxx:261
```

Thread 2:

```
Description: Lock owned
Source: gthr-default.h:748
Function: __gthread_mutex_lock
Module: libAthAllocators.so
Variable: block allocated at ArenaBase.cxx:30
Code snippet:
```

```
746 {
747     if (__gthread_active_p ())
>748     return __gthrw_(pthread_mutex_lock) (__mutex);
749     else
750     return 0;
```

Click to edit

Call Stack:

```
libAthAllocators.so!__gthread_mutex_lock - gthr-default.h:748
libStoreGateLib.so!clearStore - SGImplSvc.cxx:309
libStoreGateLib.so!clearStore - SGHiveMgrSvc.cxx:49
libAthenaServices.so!clearWBSlot - AthenaHiveEventLoopMgr.cxx:1310
libAthenaServices.so!drainScheduler - AthenaHiveEventLoopMgr.cxx:1260
libAthenaServices.so!nextEvent - AthenaHiveEventLoopMgr.cxx:850
libAthenaServices.so!executeRun - AthenaHiveEventLoopMgr.cxx:764
```

- Differences in the LAr Hits affecting ***very rarely*** the energy:

Ex:

```
Py:diff-root      INFO comparing [22] leaves over entries...
000.LArHitContainer_p2_LArHitEMB.m_energy.6891 3484255171L -> 1336771523L => diff= [22.27205722%]
Py:diff-root      INFO Found [630943] identical leaves
Py:diff-root      INFO Found [1] different leaves
```

- [LArG4SimpleSD](#) instances are contained in [SDWrapper](#), which has a common hit collection container for all SDs.
- The [SDWrapper](#) is a derived [G4VSensitiveDetector](#), and instances of [SDWrapper](#) are contained in a [thread ID-keyed map](#).
- Problem was likely localized to [ProcessHits](#).

```
LArCalorimeter/LArG4/LArG4Barrel/src/LArBarrelCalibrationCalculator.cxx
LArCalorimeter/LArG4/LArG4Barrel/src/LArBarrelPresamplerCalculator.cxx
```

Strategy for MT debugging

Simulation with 10 ttbar events,
sequential mode vs *MT* with 5 threads

Strategy for MT debugging

Simulation with 10 ttbar events,
sequential mode vs *MT* with 5 threads

Increment the
messaging level
to DEBUG (~7GB
for 10 ttbar evts)

Strategy for MT debugging

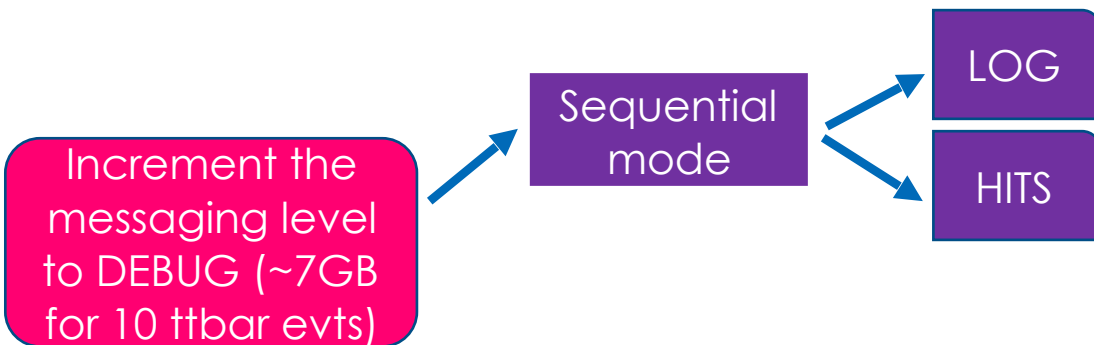
Simulation with 10 ttbar events,
sequential mode vs *MT* with 5 threads

Increment the
messaging level
to DEBUG (~7GB
for 10 ttbar evts)

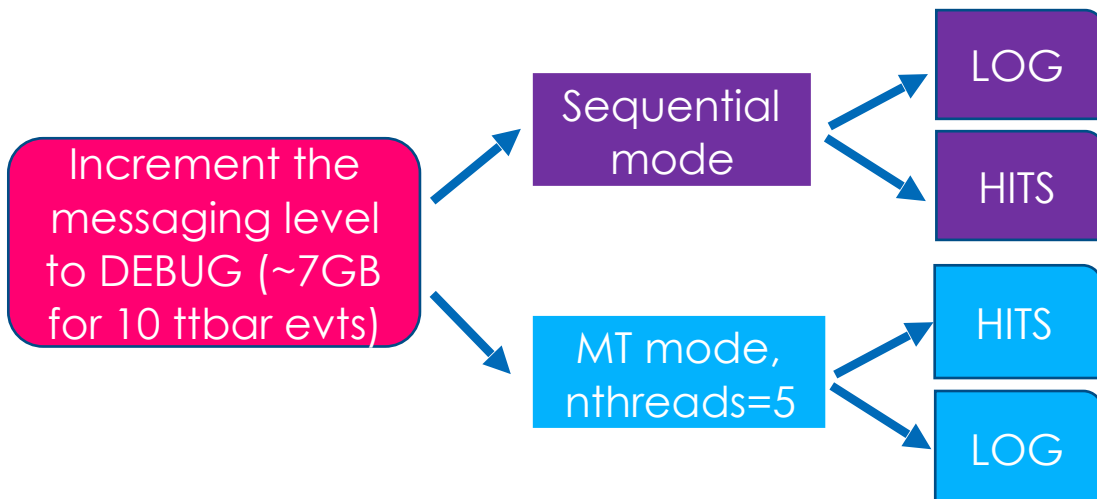


Sequential
mode

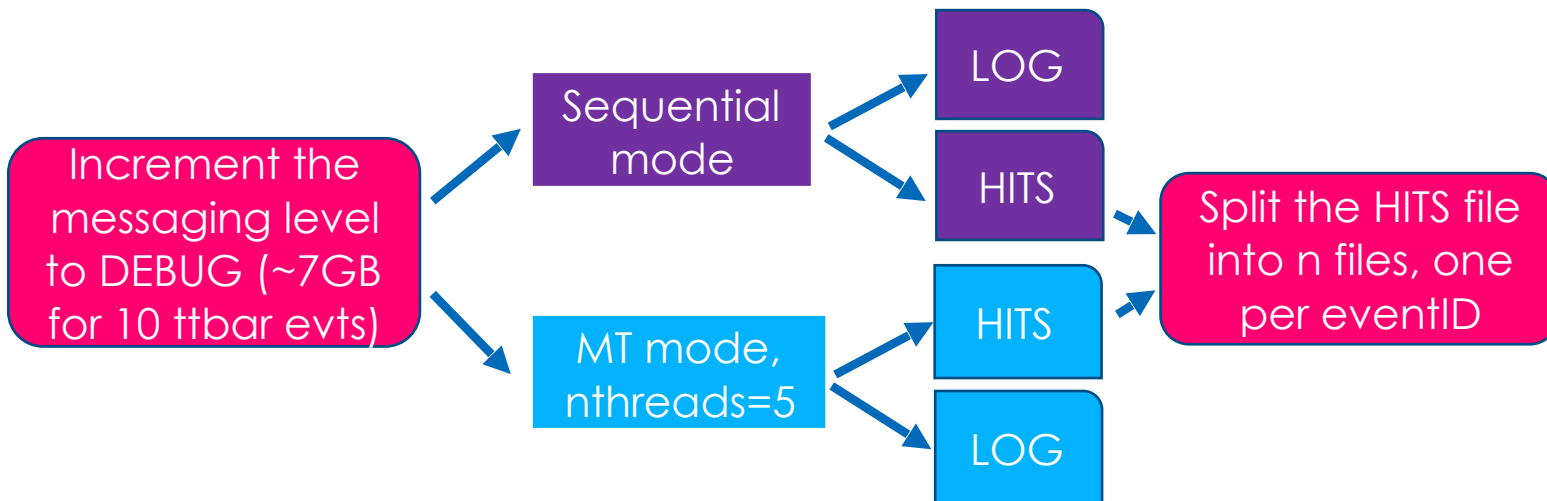
Simulation with 10 ttbar events,
sequential mode vs *MT* with 5 threads



Simulation with 10 ttbar events,
sequential mode vs *MT* with 5 threads

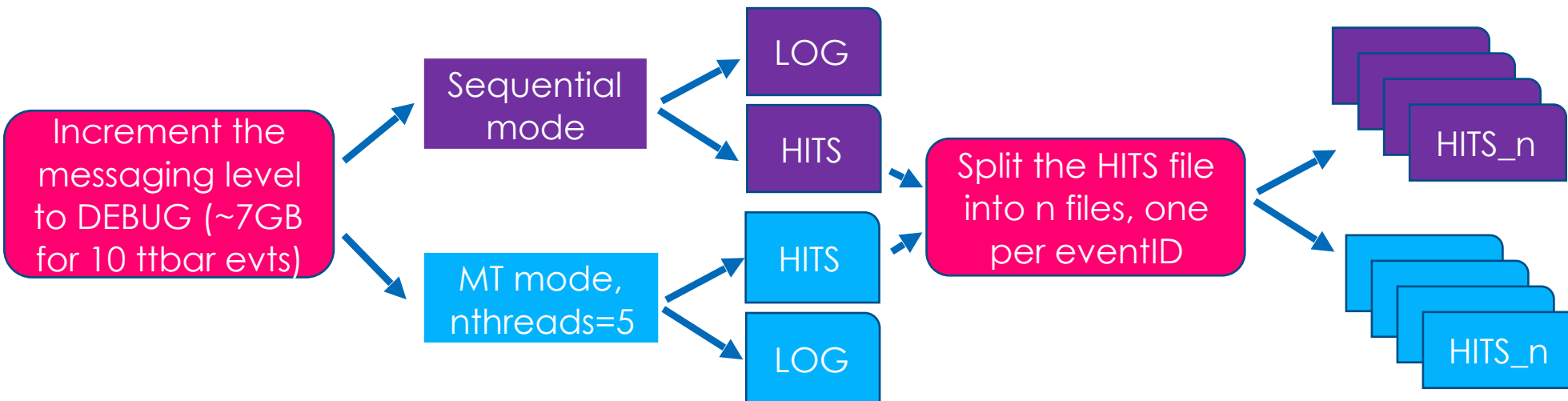


Simulation with 10 tbar events,
sequential mode vs *MT* with 5 threads

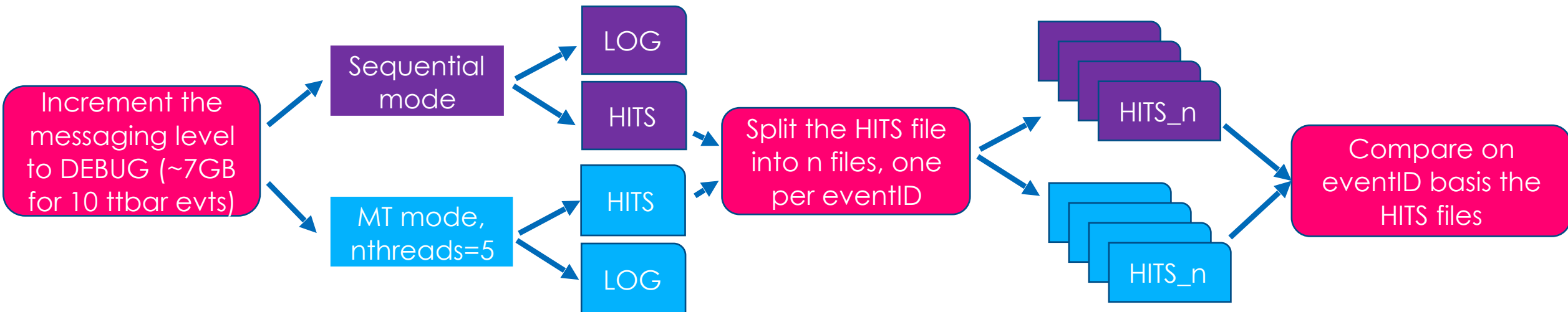


Strategy for MT debugging

Simulation with 10 tbar events,
sequential mode vs *MT* with 5 threads

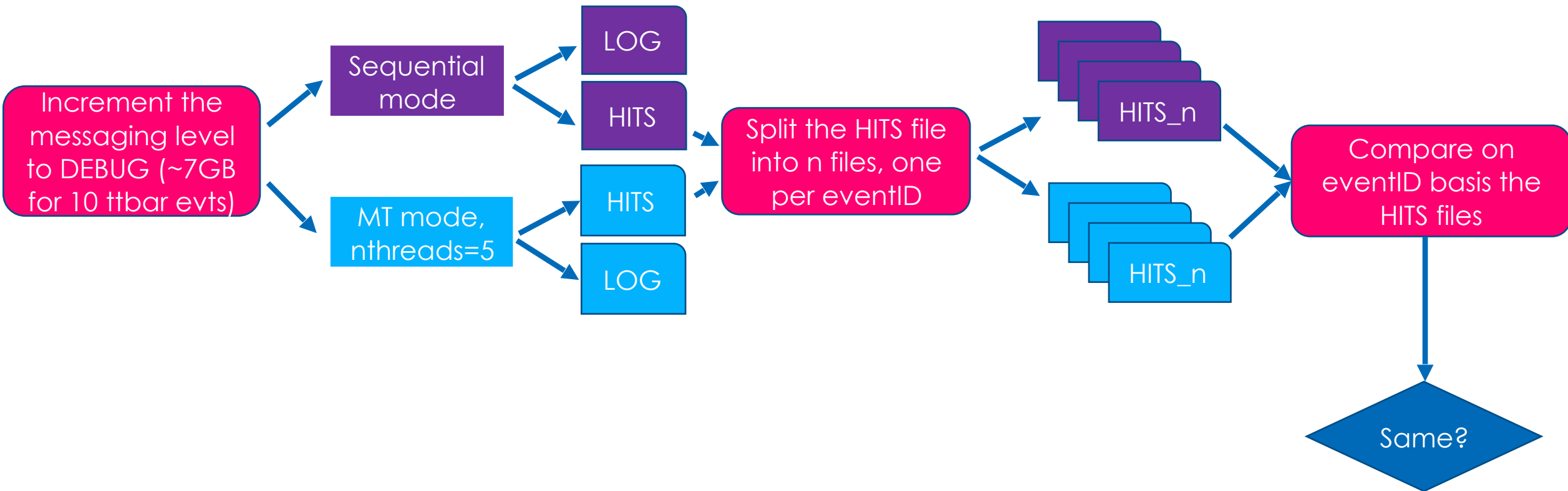


Simulation with 10 tbar events,
sequential mode vs *MT* with 5 threads



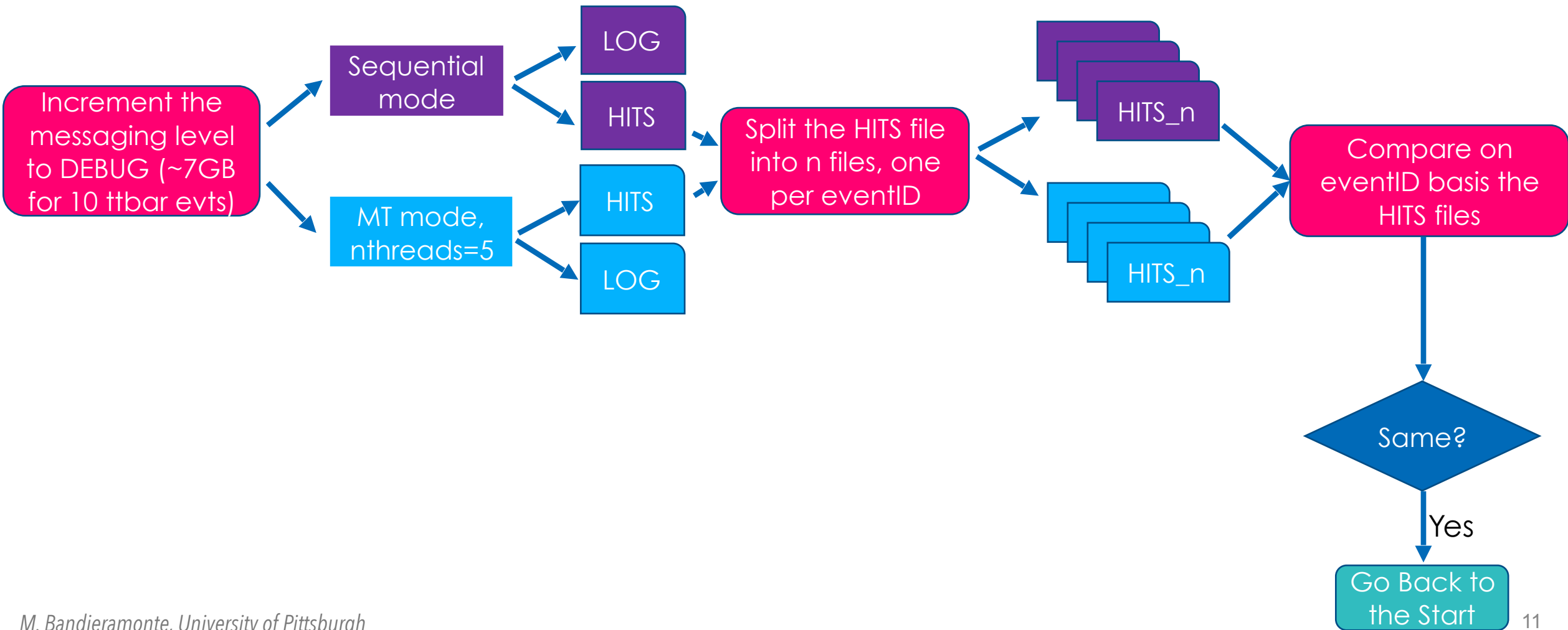
Strategy for MT debugging

Simulation with 10 tbar events,
sequential mode vs *MT* with 5 threads



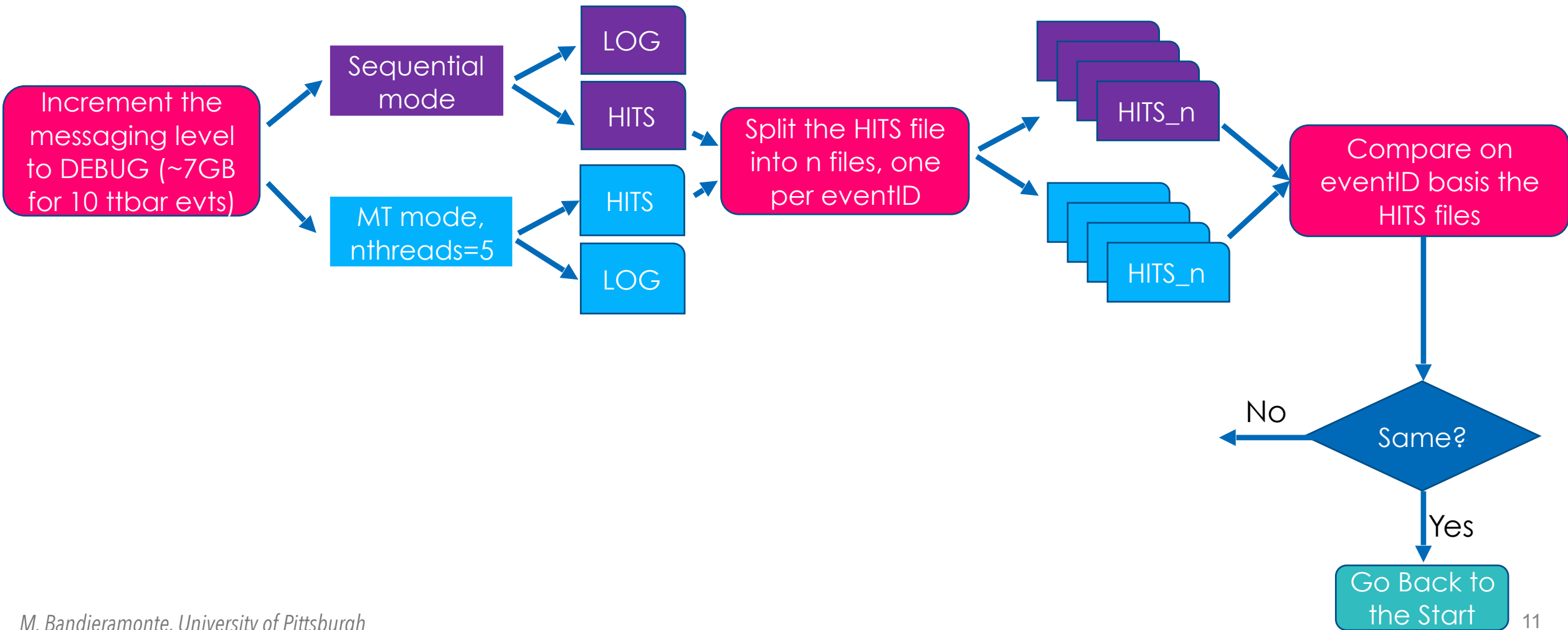
Strategy for MT debugging

Simulation with 10 tbar events,
sequential mode vs *MT* with 5 threads



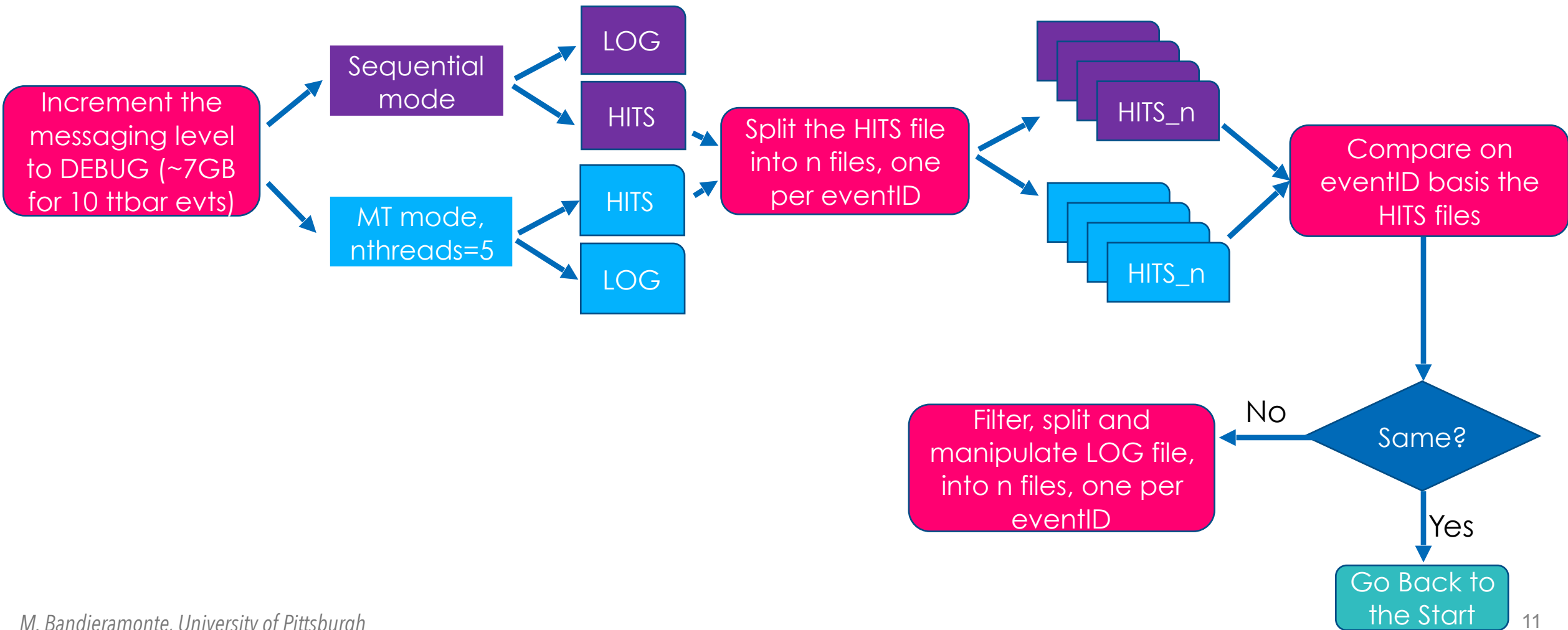
Strategy for MT debugging

Simulation with 10 tbar events,
sequential mode vs *MT* with 5 threads



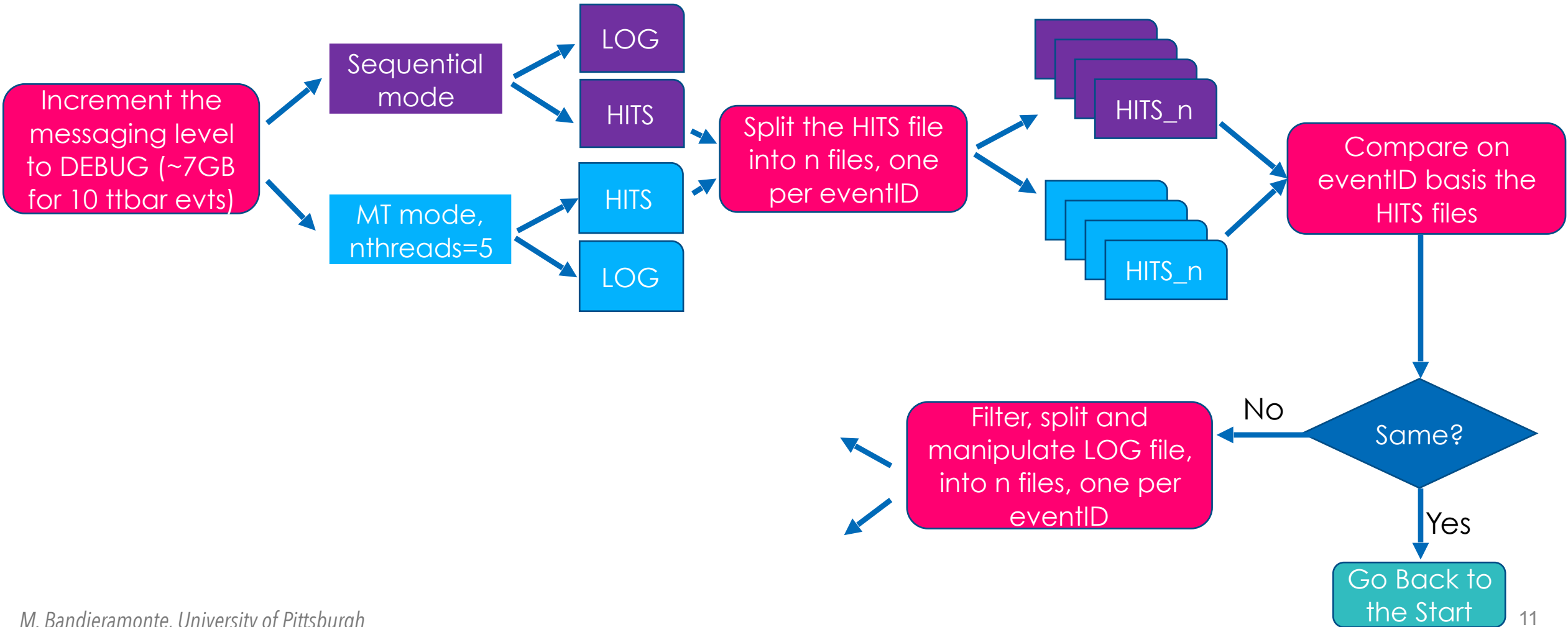
Strategy for MT debugging

Simulation with 10 tbar events,
sequential mode vs *MT* with 5 threads



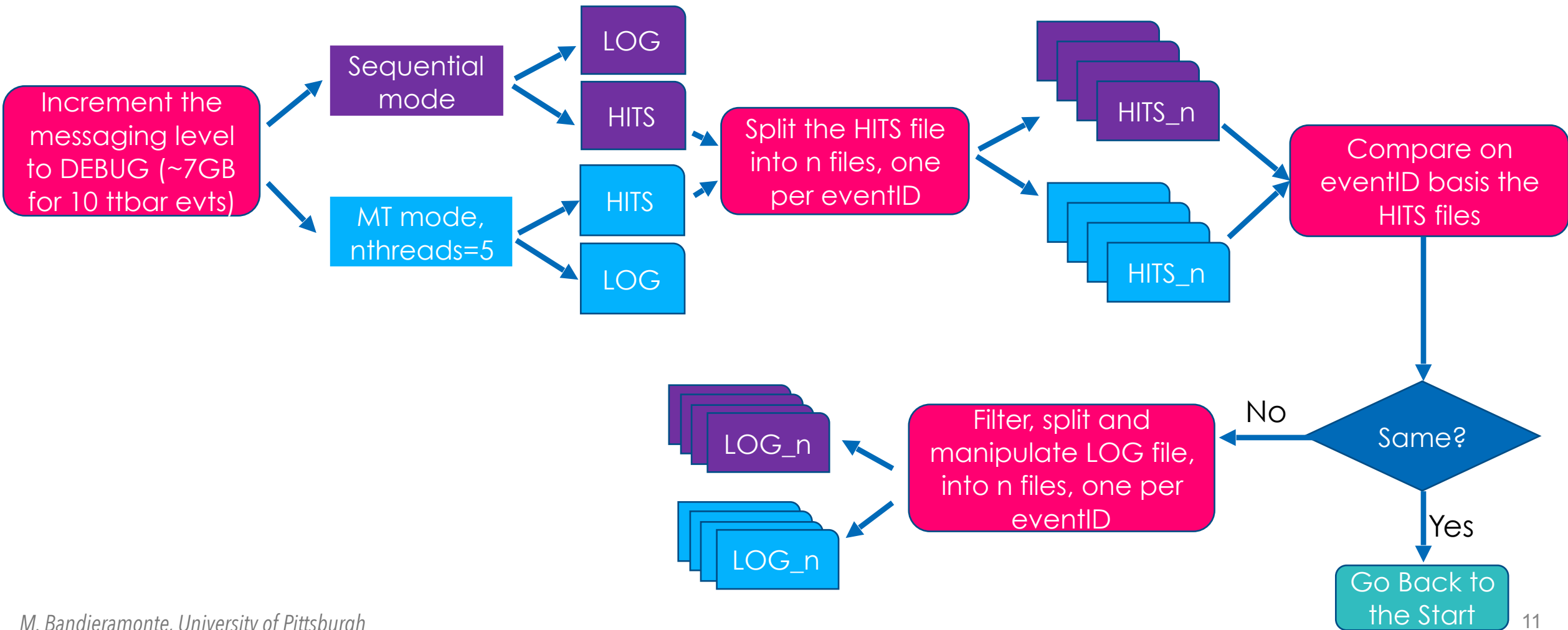
Strategy for MT debugging

Simulation with 10 tbar events,
sequential mode vs *MT* with 5 threads



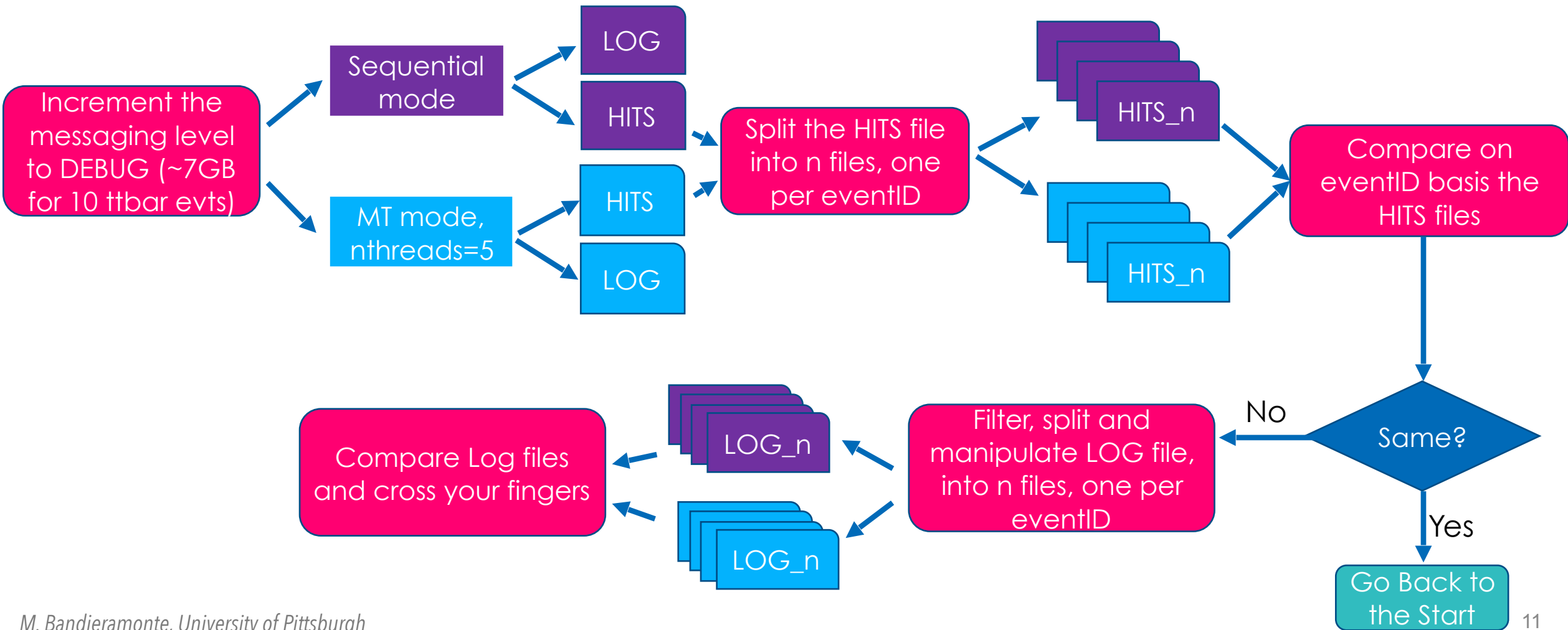
Strategy for MT debugging

Simulation with 10 tbar events,
sequential mode vs *MT* with 5 threads



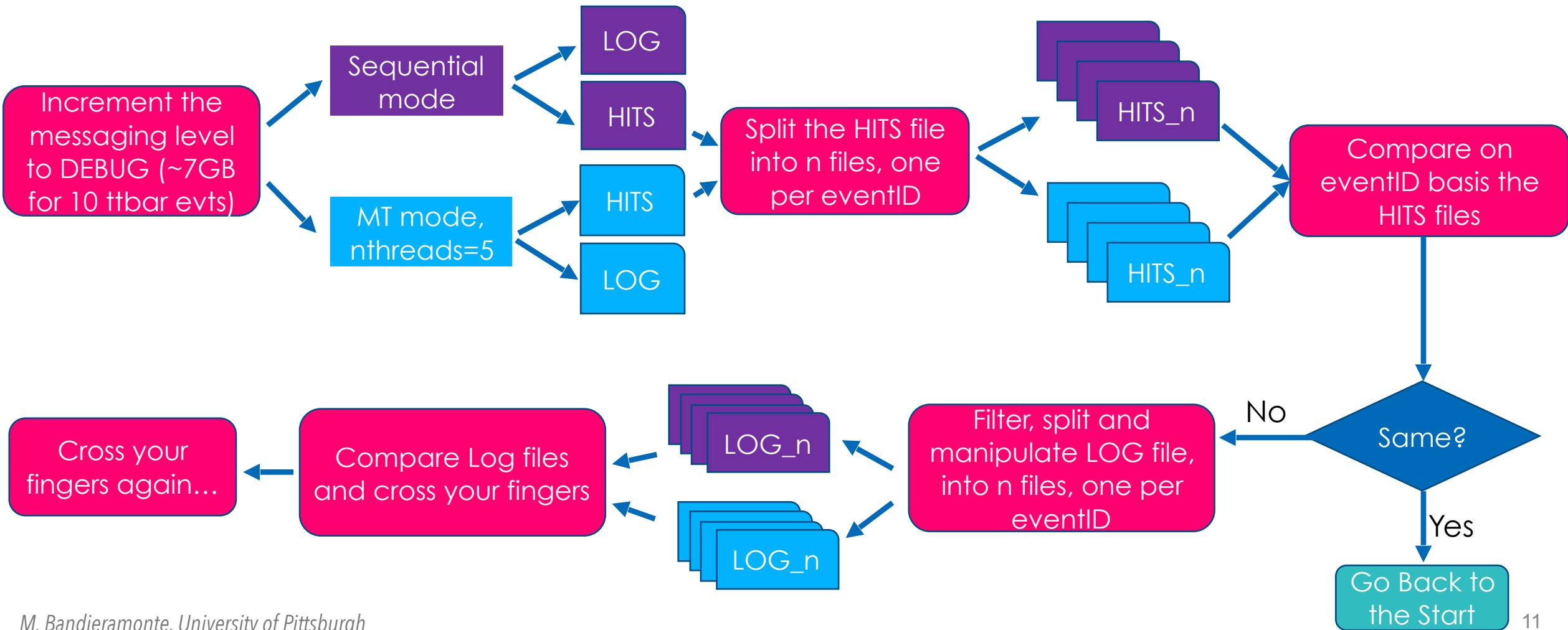
Strategy for MT debugging

Simulation with 10 tbar events,
sequential mode vs *MT* with 5 threads



Strategy for MT debugging

Simulation with 10 tbar events,
sequential mode vs *MT* with 5 threads



Strategy for MT debugging

Simulation with 10 ttbar events,
sequential mode vs *MT* with 5 threads

EMBPresamplerCalculator

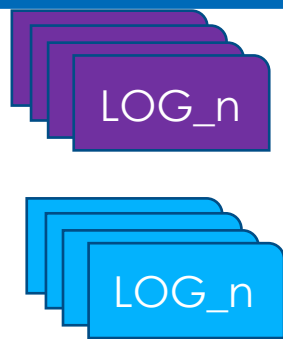
```

10673650 EMBPresamplerCa... DEBUG module,x0,y0,current0 from map 5 0.541834 0.234809 [-1.03946-] {+1.08644+}
10673651 EMBPresamplerCa... DEBUG Energy for sub step 0.0246629
10673652 EMBPresamplerCa... DEBUG set current map for module 5
10673653 EMBPresamplerCa... DEBUG module,x0,y0,current0 from map 5 0.523319 0.204478 [-1.00652-] {+1.04875+}
10673654 EMBPresamplerCa... DEBUG Energy for sub step 0.0246629
10673655 EMBPresamplerCa... DEBUG set current map for module 5
10673656 EMBPresamplerCa... DEBUG module,x0,y0,current0 from map 5 0.504805 0.174147 [-0.973578-] {+1.01106+}
10673657 EMBPresamplerCa... DEBUG Energy for sub step 0.0246629
10673658 EMBPresamplerCa... DEBUG set current map for module 5
10673659 EMBPresamplerCa... DEBUG module,x0,y0,current0 from map 5 0.48629 0.143817 [-0.932291-] {+0.970216+}
10673660 EMBPresamplerCa... DEBUG Hit Energy/Time [-0.175662-] {+0.179722+} 87.253
  
```

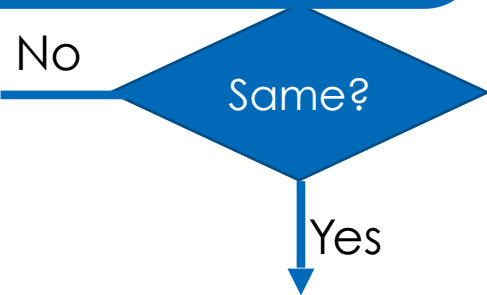
Incr
 mess
 to DE
 for 10

Cross your
 fingers again...

Compare Log files
 and cross your fingers



Filter, split and
 manipulate LOG file,
 into n files, one per
 eventID



Go Back to
 the Start

The problem & the fix

PsMap is a singleton and **SetMap ()** method was not thread-safe:

- set the current “Current map” in its private member **m_curr**
- store the module in **m_module**.

LArBarrelPresamplerCalculator.cxx

```
m_psmap->SetMap(imodule);  
m_psmap->Map()-> GetAll(x0,y0,&gap,&current0,&current1,&current2);
```

PsMap.cxx

```
void PsMap::SetMap(int module)  
{  
    if (m_module==module) return;  
    m_module=module;  
    [...]  
    if (m_theMap.find(code) != m_theMap.end())  
        m_curr = m_theMap[code];  
    else {  
        m_curr=0;  
    }  
}
```

- **! Data race** in the *LArBarrelPresamplerCalculator*:
 - **SetMap** could be called by another thread before the current values were obtained

The problem & the fix

PsMap is a singleton and **SetMap ()** method was not thread-safe:

- set the current “Current map” in its private member **m_curr**
- store the module in **m_module**.

LArBarrelPresamplerCalculator.cxx

```
CurrMap* cm = m_psmmap->GetMap(imodule);
```

PsMap.cxx

```
CurrMap* PsMap::GetMap(int module) const
{
    [...]
    auto it = m_theMap.find(code);
    if (it != m_theMap.end())
        return it->second;
    else {
        return nullptr;
    }
}
```

- **! Data race** in the *LArBarrelPresamplerCalculator*:
 - **SetMap** could be called by another thread before the current values were obtained
- Remove **SetMap ()** function and **m_curr** and **m_module** members
- Implement **CurrMap* GetMap(imodule) const** method

The problem & the fix

PsMap is a singleton and **SetMap ()** method was not thread-safe:

- set the current “Current map” in its private member **m_curr**
- store the module in **m_module**.

LArBarrelPresamplerCalculator.cxx

```
CurrMap* cm = m_psmmap->GetMap(imodule);
```

PsMap.cxx

```
CurrMap* PsMap::GetMap(int module) const
{
    [...]
    auto it = m_theMap.find(code);
    if (it != m_theMap.end())
        return it->second;
    else {
        return nullptr;
    }
}
```

Tested with 100 ttbar with MT 10 threads → Solved!

- **! Data race** in the *LArBarrelPresamplerCalculator*:
 - **SetMap** could be called by another thread before the current values were obtained
- Remove **SetMap ()** function and **m_curr** and **m_module** members
- Implement **CurrMap* GetMap(imodule) const** method

- **Recent progress highlights:**

- **Output** validation:

- **Fixed:** thread-unsafety causing difference in HITS of **LAr sensitive detector** (~1-2%)
- **Fixed:** thread-unsafety causing difference in HITS of **Tile sensitive detector** (~1-5%)
- **Fixed:** simulation with **CaloCalibrationHit** (~50% of Dead material hits)

We can now run reliably full **single-threaded** and **multi-threaded** simulations, results are fully consistent **(read identical) !** physics validation in progress.

- Confirmed reproducibility of simulation with **SUSY/Exotics G4Extensions** enabled:

- We currently have six packages which add support for additional particles and physics processes to Geant4

- **Charginos** - Stable and Decaying Charginos – OK ✓
- **Gauginos** - OK ✓
- **Neutralinos** - Decaying - OK ✓
- **Monopoles** - OK ✓
- **Quirks** - Postponed - lack of samples (no associated physics analysis) ✗
- **RHadrons** - waiting for samples ✗
- **Sleptons** - Stable, Decaying taus, Decaying light - OK ✓

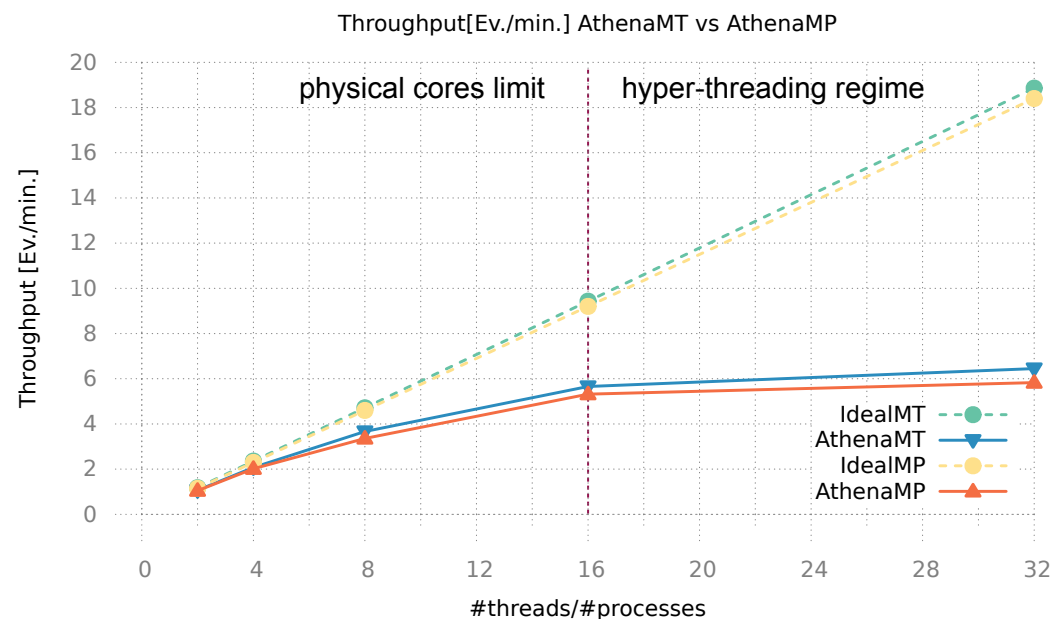
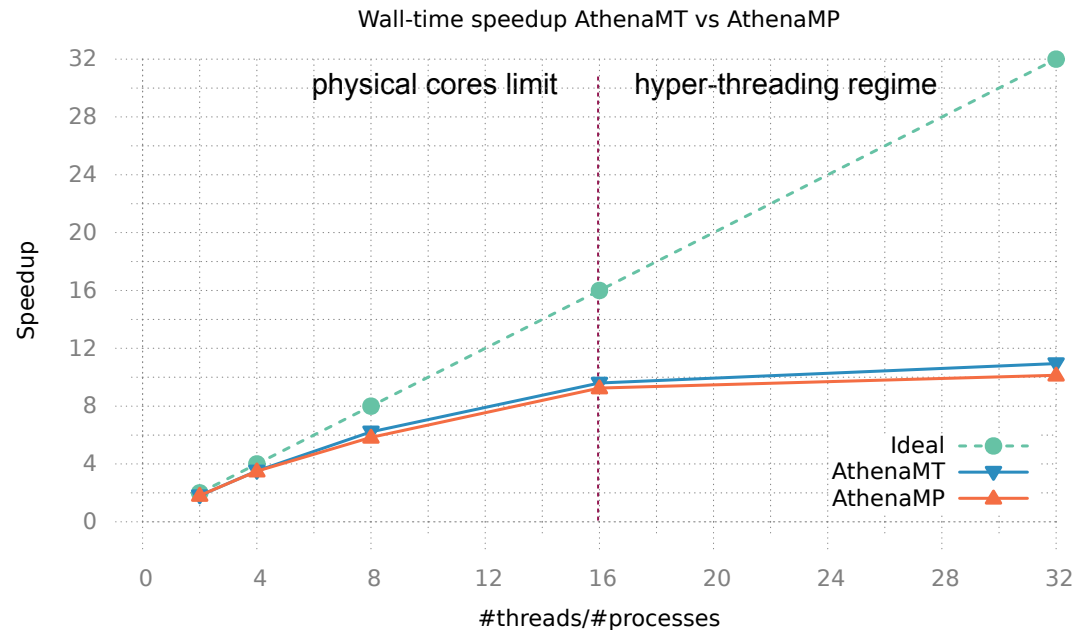
Architecture: x86_64
 CPU op-mode(s): 32-bit, 64-bit
 Byte Order: Little Endian
 CPU(s): 32
 On-line CPU(s) list: 0-31
 Thread(s) per core: 2
 Core(s) per socket: 8
 Socket(s): 2
 NUMA node(s): 2
 Model: 79
 Model name: Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz
 Test on 100 ttbar events, with prom
 Athena, r2019-09-30T2130, master

results are AVG of 5 separate runs (from 1-32 threads/processes) – the machine was quiet all the time (me as only user)

$$\text{AthenaMT Speedup}_{\text{th}_n} = \text{Wall-time}_{\text{th}_1} / \text{Wall-time}_{\text{th}_n}$$

$$\text{AthenaMP Speedup}_{\text{proc}_n} = \text{Wall-time}_{\text{proc}_1} / \text{Wall-time}_{\text{proc}_n}$$

Wall-Time [min.]	1 thread/process
AthenaMT	169.6733333
AthenaMP	173.9166667



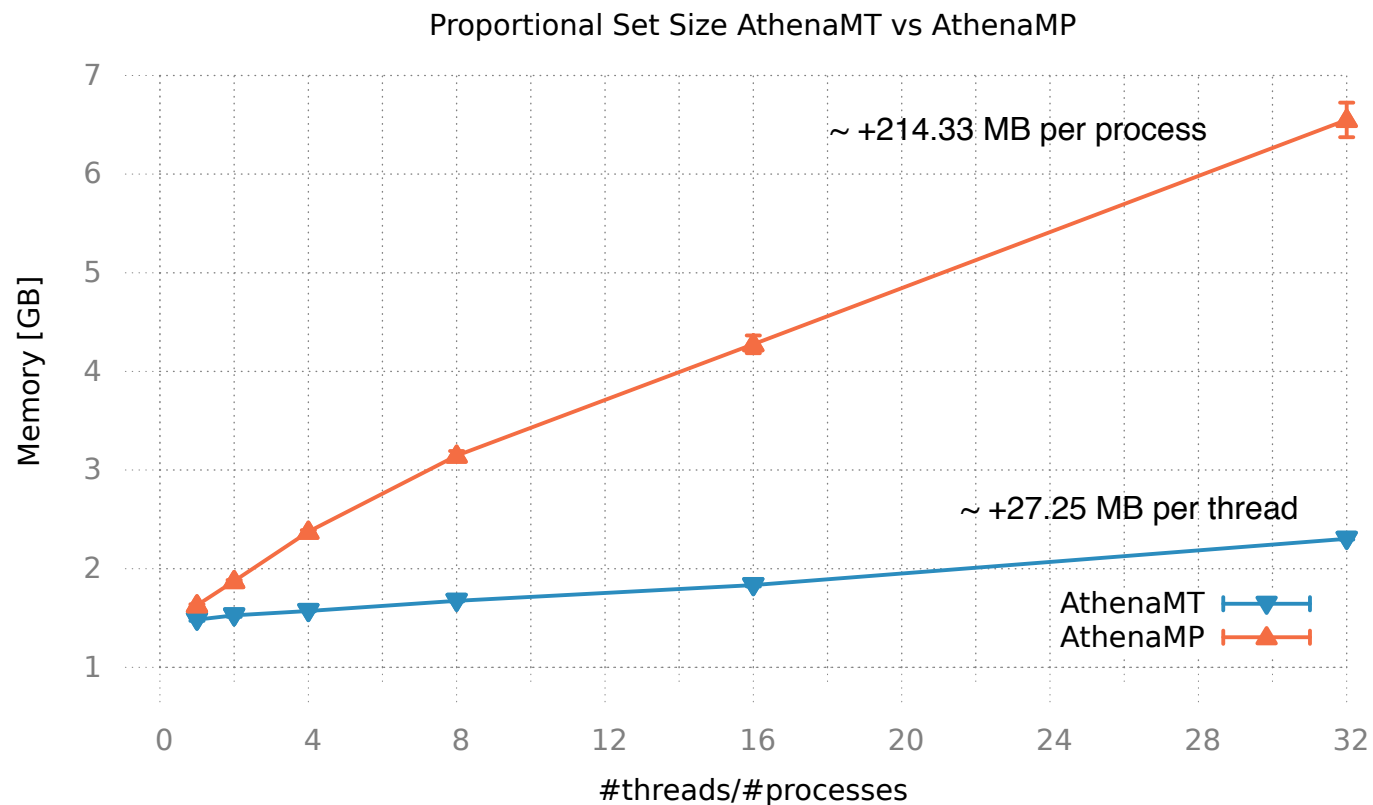

```

Architecture:      x86_64
CPU op-mode(s):   32-bit, 64-bit
Byte Order:       Little Endian
CPU(s):           32
On-line CPU(s) list: 0-31
Thread(s) per core: 2
Core(s) per socket: 8
Socket(s):        2
NUMA node(s):    2
Model:            79
Model name:       Intel(R) Xeon(R)
                   CPU E5-2620 v4 @ 2.10GHz
    
```

Test on 100 ttbar events, with prom
Athena, r2019-09-30T2130, master

results are AVG of 5 separate runs (from 1-32 threads/processes) – the machine was quiet all the time (me as only user)

PSS[GB]	1 thread/process
AthenaMT	1.482771301
AthenaMP	1.628312683



The Proportional Set Size (PSS) is the portion of main memory occupied by a process and is composed by the private memory of that process plus the proportion of shared memory with one or more other processes

- **The Athena Multi-threaded simulation with Geant4MT is fully functional**
 - **Outside of ISF:**
 - The G4 single threaded vs multi-threaded output has been confirmed to be identical
 - 100k grid test were ran with 8 cores without reported issues (physics validation in progress)
 - **Inside ISF:**
 - After revising the Geant4 initialisation steps in MT mode, simulation runs correctly in multi-threaded mode with 1 thread and the output has been validated
 - Next steps
 - *Solve the thread-unsafely issues and assure that G4MT simulation works with more than one thread and that the results are reproducible*

Thanks for your attention!

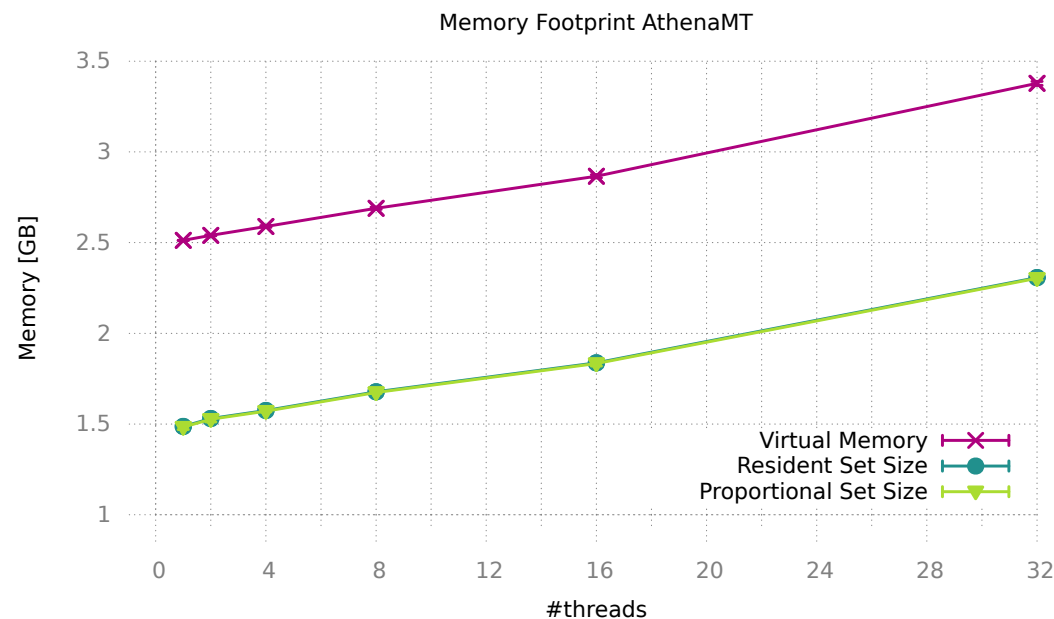
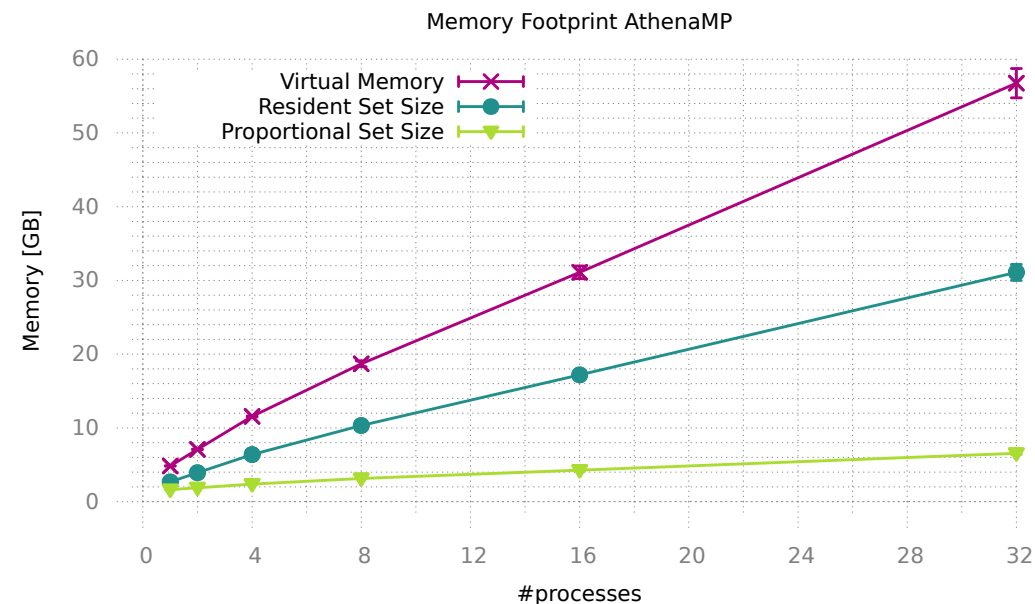
Marilena Bandieramonte
marilena.bandieramonte@cern.ch

Backup slides

Architecture: x86_64
 CPU op-mode(s): 32-bit, 64-bit
 Byte Order: Little Endian
 CPU(s): 32
 On-line CPU(s) list: 0-31
 Thread(s) per core: 2
 Core(s) per socket: 8
 Socket(s): 2
 NUMA node(s): 2
 Model: 79
 Model name: Intel(R) Xeon(R)
 CPU E5-2620 v4 @ 2.10GHz

Test on 100 ttbar events, with prom
 Athena, r2019-09-30T2130, master

results are AVG of 5 separate runs (from 1-32
 threads/processes) – the machine was quiet all the
 time (me as only user)



AthenaMT vs AthenaMP benchmarks

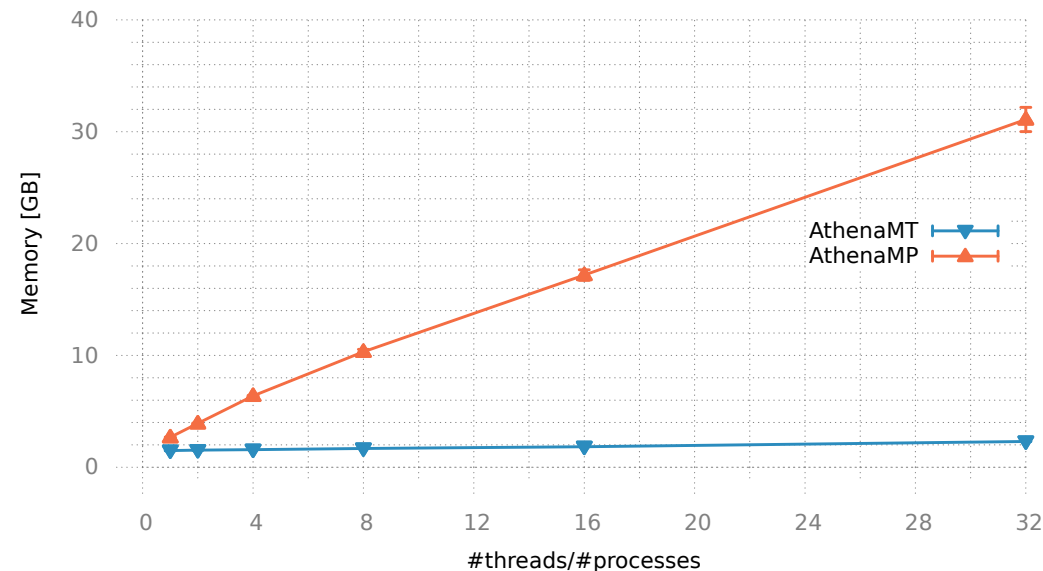
```

Architecture:      x86_64
CPU op-mode(s):   32-bit, 64-bit
Byte Order:       Little Endian
CPU(s):           32
On-line CPU(s) list: 0-31
Thread(s) per core: 2
Core(s) per socket: 8
Socket(s):        2
NUMA node(s):    2
Model:            79
Model name:       Intel(R) Xeon(R)
                   CPU E5-2620 v4 @ 2.10GHz
  
```

Test on 100 ttbar events, with prom
 Athena, r2019-09-30T2130, master

results are AVG of 5 separate runs (from 1-32
 threads/processes) – the machine was quiet all the
 time (me as only user)

Resident Set Size AthenaMT vs AthenaMP



Virtual Memory AthenaMT vs AthenaMP

