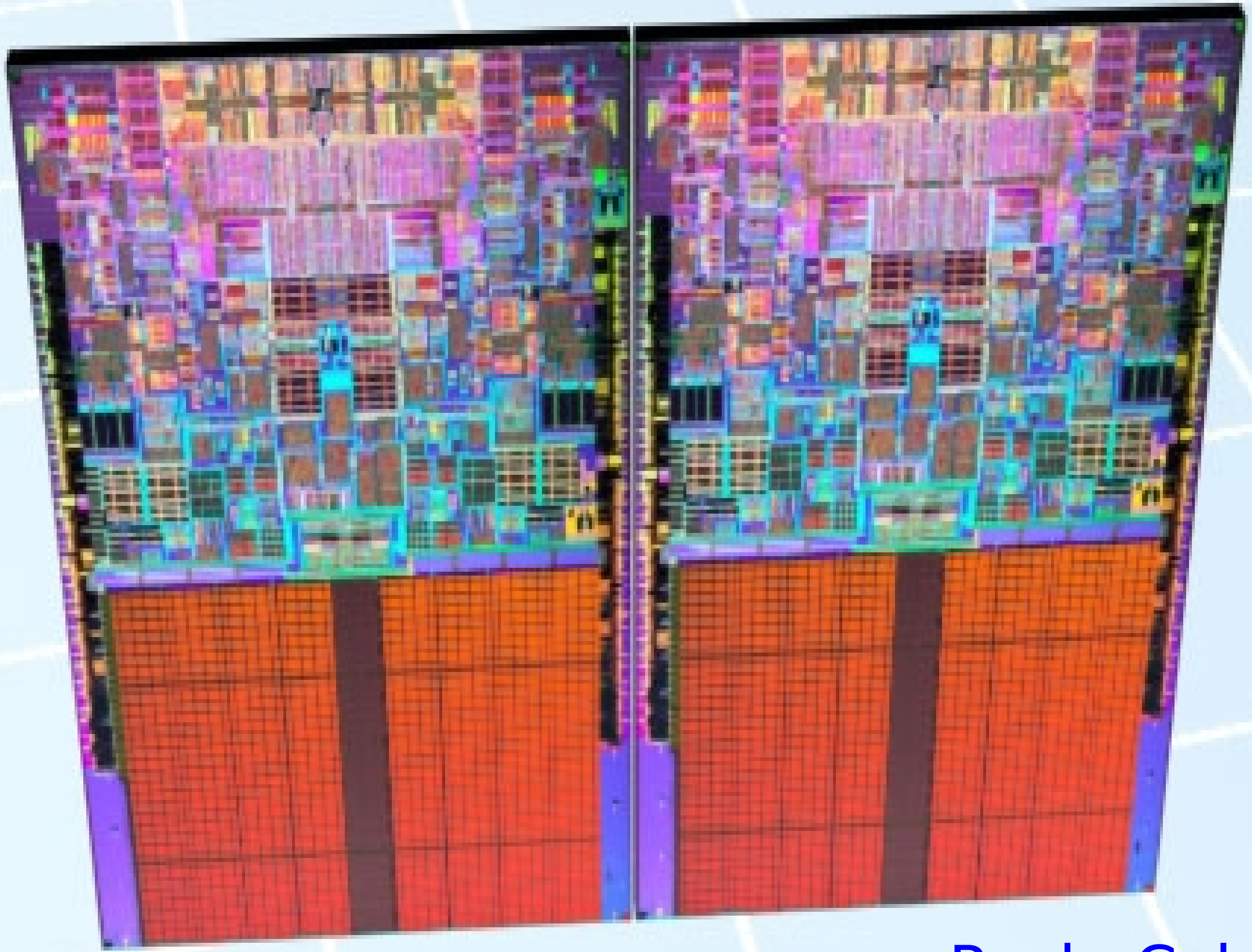
Core Performance



Paolo Calafiura

Lawrence Berkeley National Lab

CHEP 09 – March 28 2009

In this talk...

Efficient use of CPU and memory

- Performance Monitoring and Optimization

Unix programming environment

- i686, Linux, C/C++

“Rules of Optimization:

1. Don't do it

2. (for experts only) Don't do it yet “

(M. A. Jackson)

“More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason – including blind stupidity”

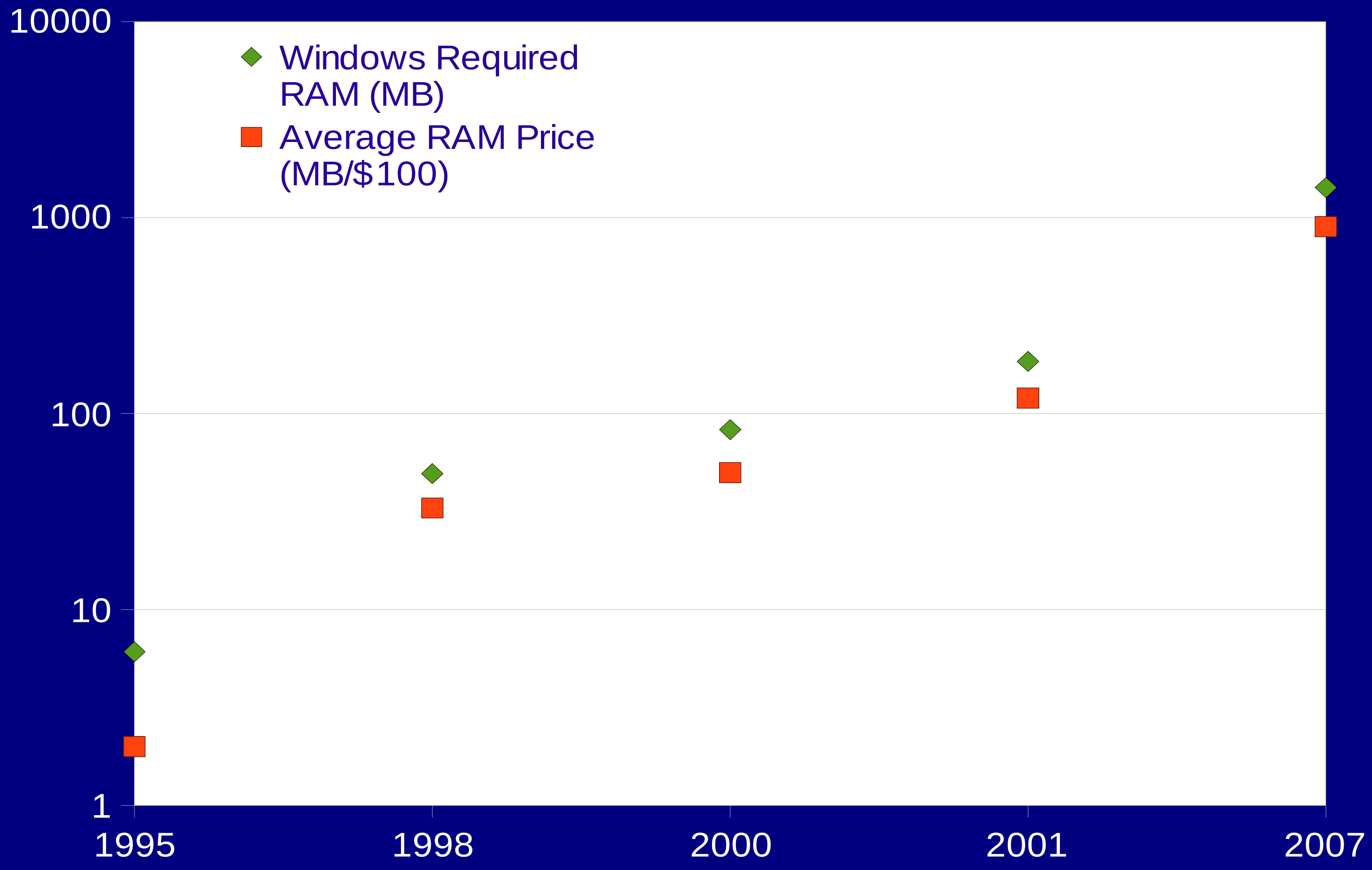
(W. A. Wulf)

“Preliminary optimization is the root of all evil”

(Donald Knuth)

Is Efficiency Effective?

Software Horror Vacui



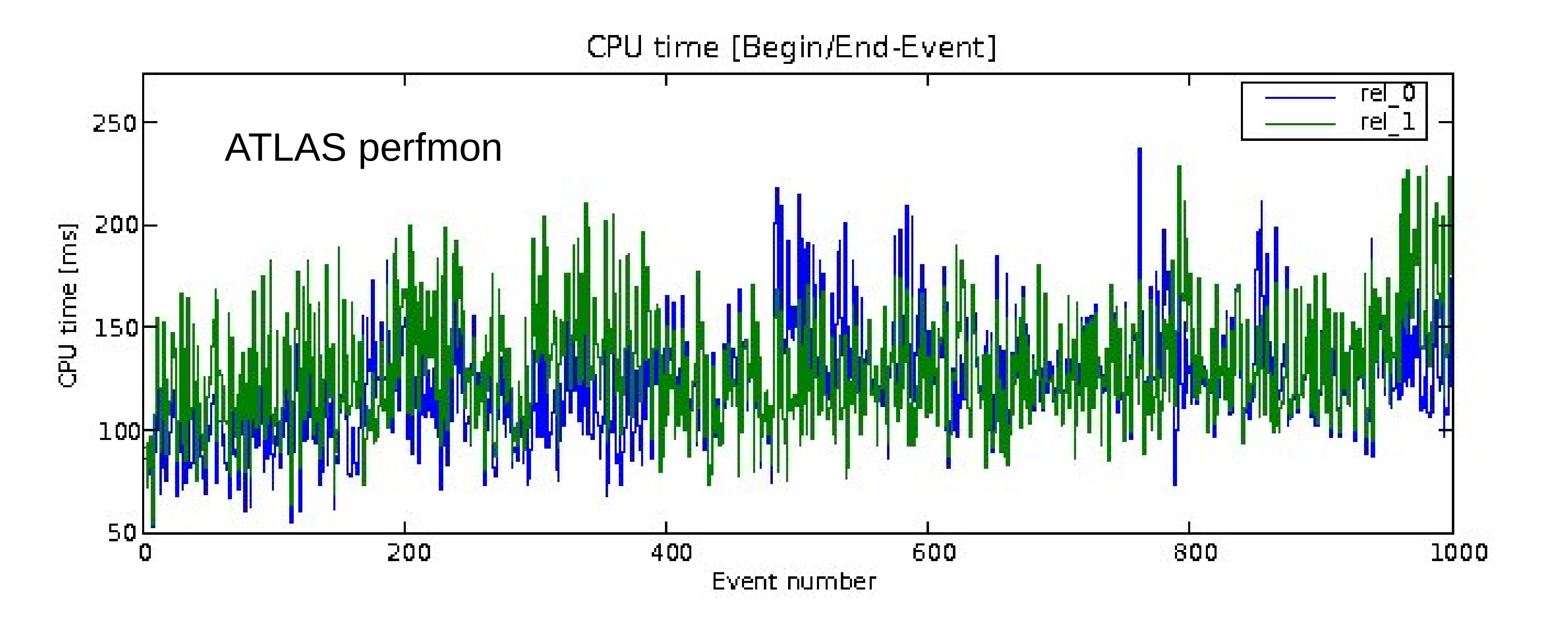
Hungrier than Vista

LHC reconstruction

pushing against 2GB/core
typical GRID node

Cai Guo-Qiang

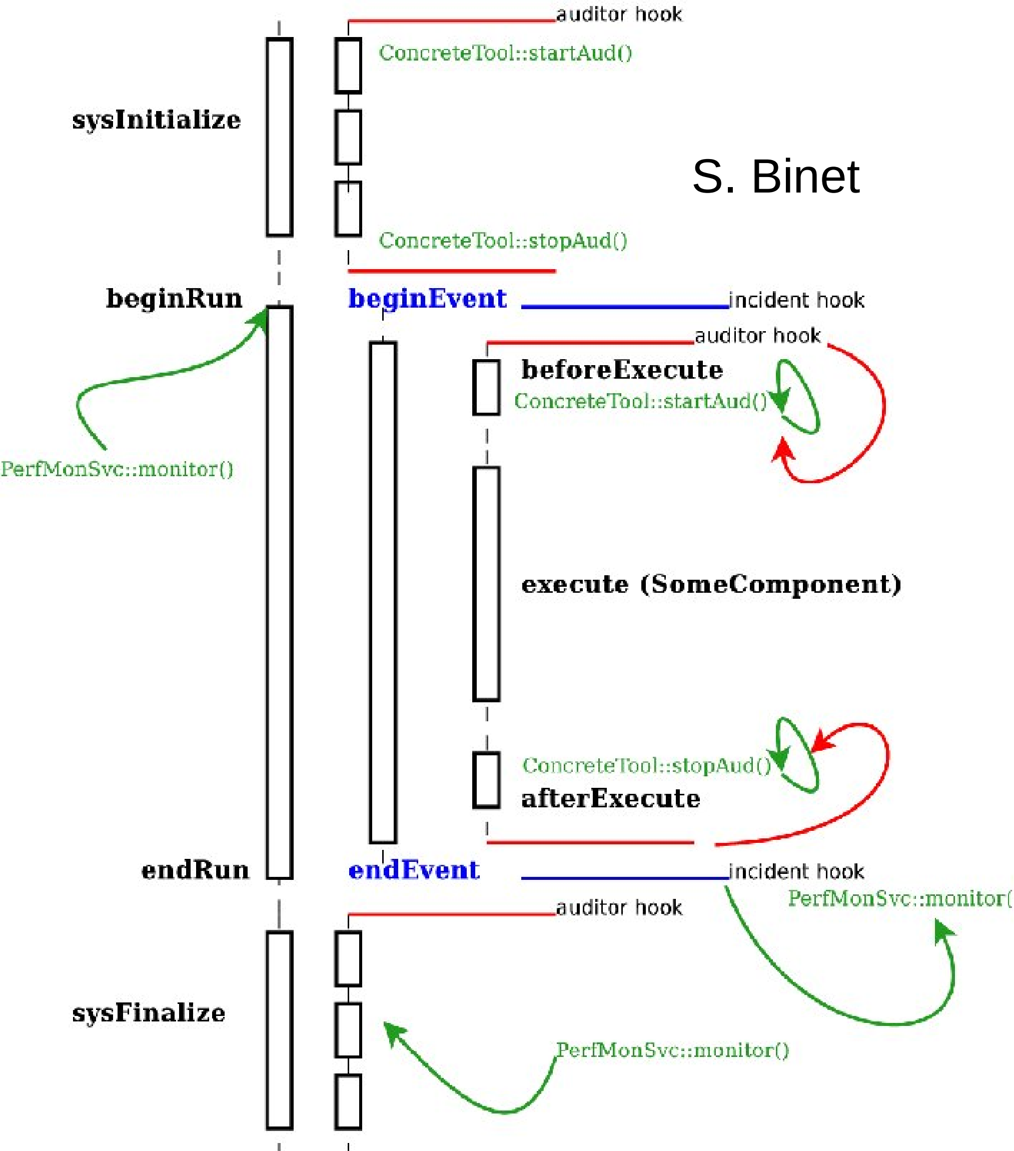
Performance Monitoring



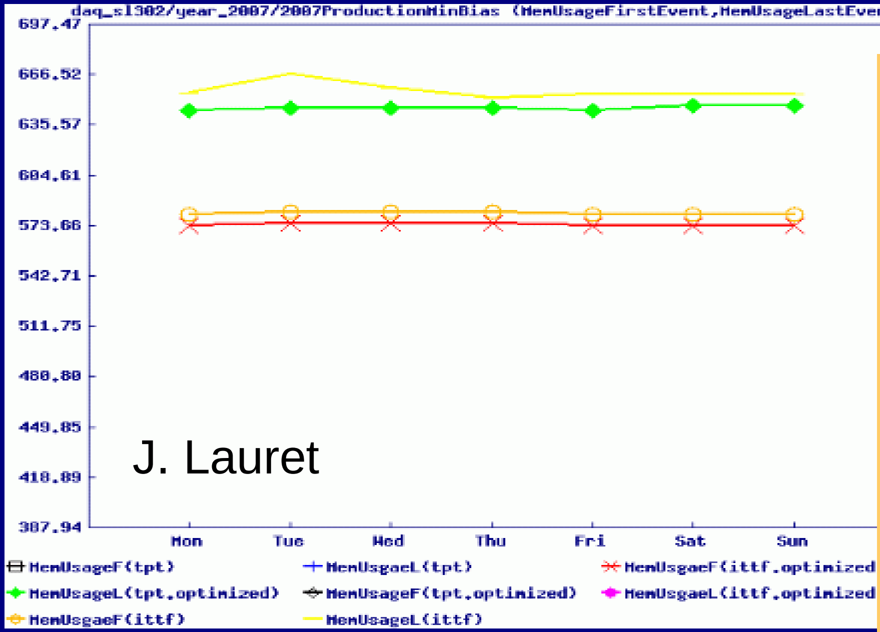
Information is the most valuable commodity
(Gordon Gekko)

Performance Monitoring in Practice

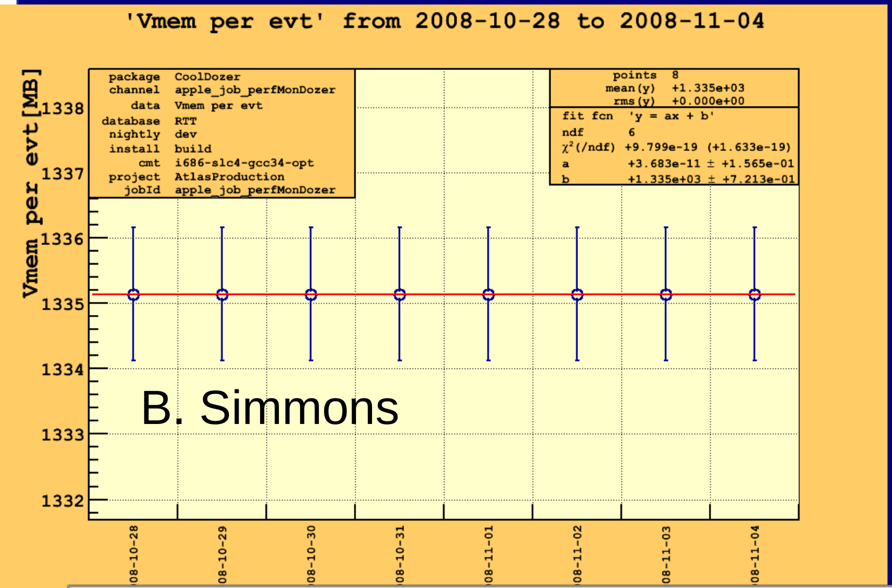
- Integrate into framework
- Collect data per component
  - CPU
    - user/sys/wall
  - Memory
    - VMEM, RSS, leaks, mallocs
  - Developer-defined



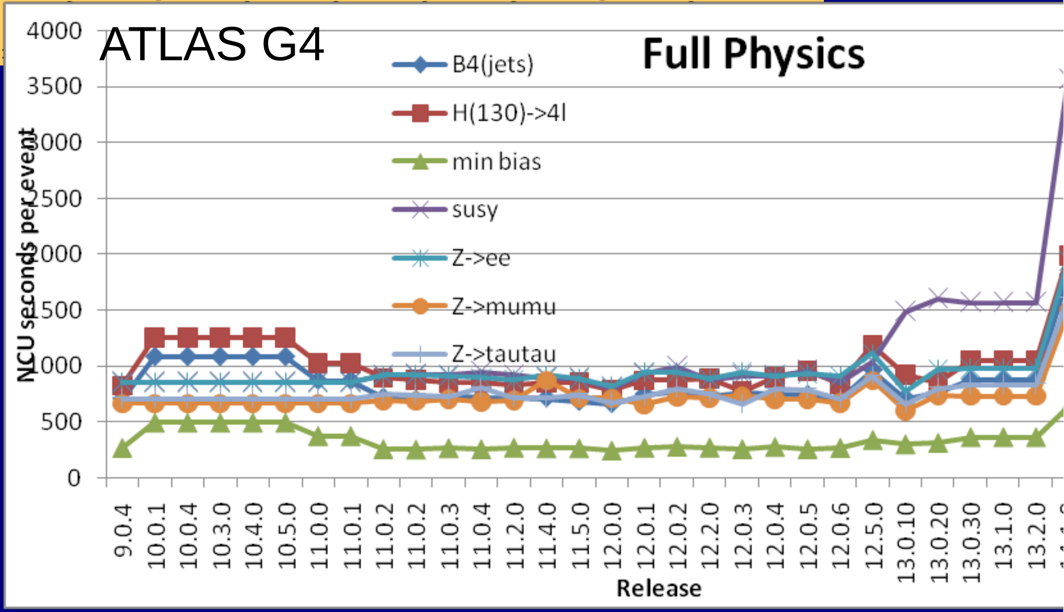
Track Performance Evolution



J. Lauret

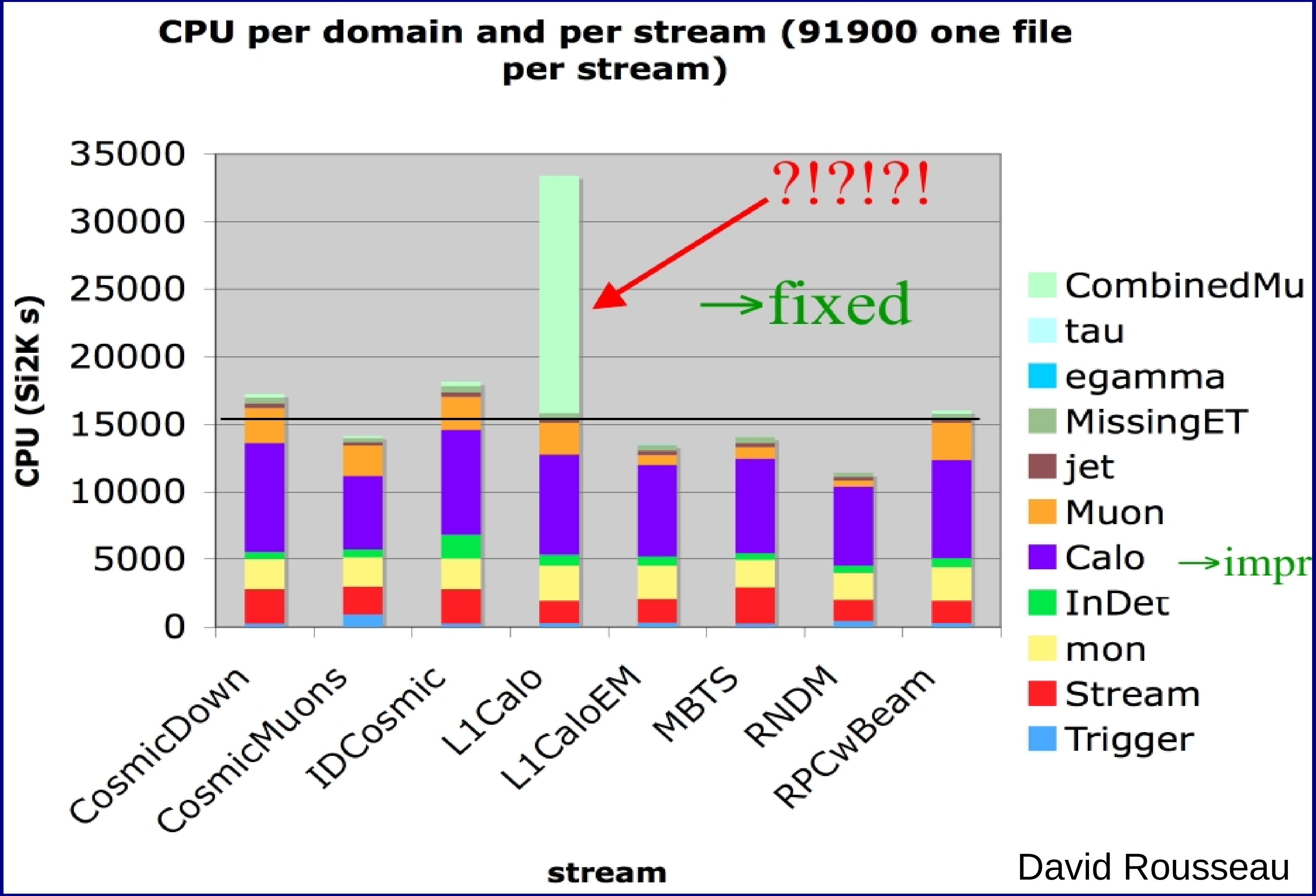


B. Simmons



Best tool for release coordinators to spot new issues

Actionable Information



Performance Monitoring for Managers

Start early, never stop

Monitor what runs in production, all of it

Track evolution, intervene immediately

Turn measurements into action items

Raise profile of exercise, put physicists in charge

Performance Optimization

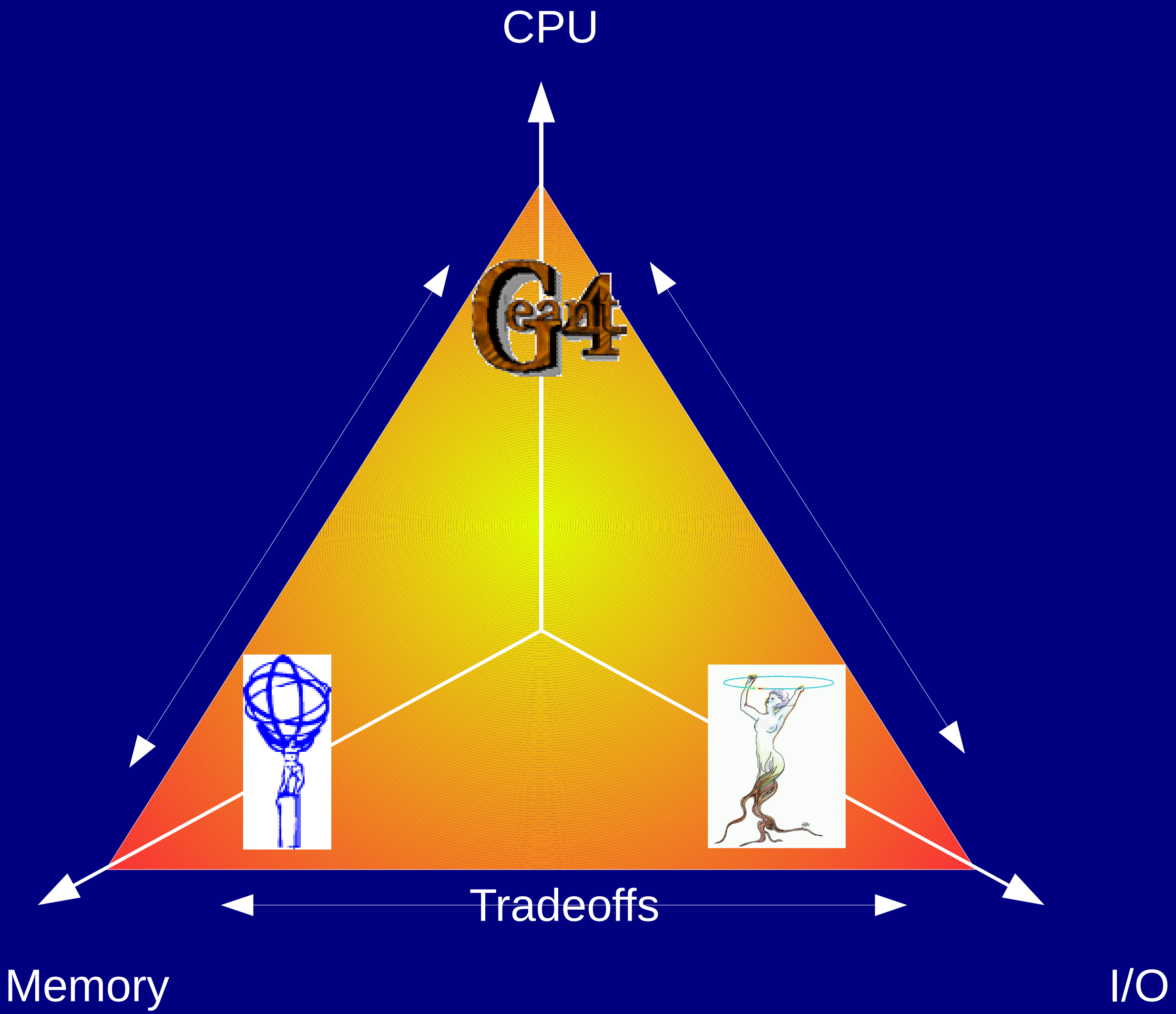
Monday Parallel Session 3

- Pete Elmer – CMS Software Performance
- Giulio Eulisse – HEP C++ meets reality

Thursday Parallel Session 12

- Andrzej Nowak – An update on perfmon and the struggle to get it into the Linux kernel

Optimize What?



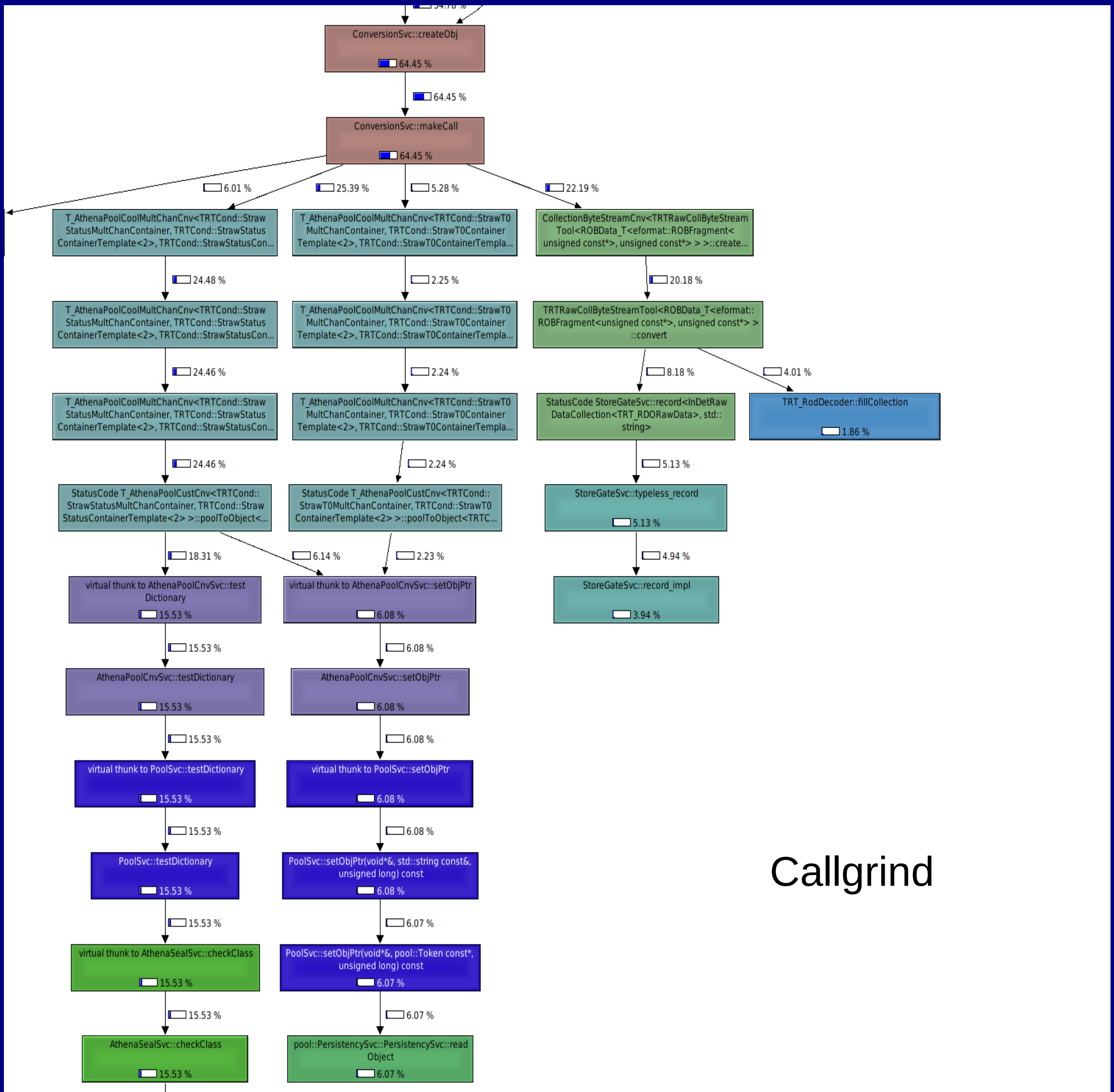
CPU Optimization

Tools provide

- Usage graph
- Line-by-line breakdown

First 10% “easy”

- pass by value
- temporaries
- sub-optimal containers



54.78 %
ConversionSvc::createObj
64.45 %
64.45 %
ConversionSvc::makeCall
64.45 %
6.01 %
25.39 %
5.28 %
22.19 %
T_AthenaPoolCoolMultChanCnv<TRTCond::StrawStatusMultChanContainer, TRTCond::StrawStatusContainerTemplate<2>, TRTCond::StrawStatusCon...
T_AthenaPoolCoolMultChanCnv<TRTCond::StrawT0MultChanContainer, TRTCond::StrawT0ContainerTemplate<2>, TRTCond::StrawT0ContainerTempla...
CollectionByteStreamCnv<TRTRawCollByteStreamTool<ROBData_T<eformat::ROBFragment<unsigned const*>, unsigned const*> > >::create...
24.48 %
2.25 %
20.18 %
T_AthenaPoolCoolMultChanCnv<TRTCond::StrawStatusMultChanContainer, TRTCond::StrawStatusContainerTemplate<2>, TRTCond::StrawStatusCon...
T_AthenaPoolCoolMultChanCnv<TRTCond::StrawT0MultChanContainer, TRTCond::StrawT0ContainerTemplate<2>, TRTCond::StrawT0ContainerTempla...
TRTRawCollByteStreamTool<ROBData_T<eformat::ROBFragment<unsigned const*>, unsigned const*> > ::convert
24.46 %
2.24 %
8.18 %
4.01 %
T_AthenaPoolCoolMultChanCnv<TRTCond::StrawStatusMultChanContainer, TRTCond::StrawStatusContainerTemplate<2>, TRTCond::StrawStatusCon...
T_AthenaPoolCoolMultChanCnv<TRTCond::StrawT0MultChanContainer, TRTCond::StrawT0ContainerTemplate<2>, TRTCond::StrawT0ContainerTempla...
StatusCode StoreGateSvc::record<InDetRawDataCollection<TRT_RDORawData>, std::string>
TRT_RodDecoder::fillCollection
1.86 %
24.46 %
2.24 %
5.13 %
StatusCode T_AthenaPoolCustCnv<TRTCond::StrawStatusMultChanContainer, TRTCond::StrawStatusContainerTemplate<2> >::poolToObject<...
StatusCode T_AthenaPoolCustCnv<TRTCond::StrawT0MultChanContainer, TRTCond::StrawT0ContainerTemplate<2> >::poolToObject<TRTC...
StoreGateSvc::typeless_record
5.13 %
18.31 %
6.14 %
2.23 %
4.94 %
virtual thunk to AthenaPoolCnvSvc::testDictionary
15.53 %
virtual thunk to AthenaPoolCnvSvc::setObjPtr
6.08 %
StoreGateSvc::record_impl
3.94 %
15.53 %
6.08 %
AthenaPoolCnvSvc::testDictionary
15.53 %
AthenaPoolCnvSvc::setObjPtr
6.08 %
15.53 %
6.08 %
virtual thunk to PoolSvc::testDictionary
15.53 %
virtual thunk to PoolSvc::setObjPtr
6.08 %
15.53 %
6.08 %
PoolSvc::testDictionary
15.53 %
PoolSvc::setObjPtr(void*&, std::string const&, unsigned long) const
6.08 %
15.53 %
6.07 %
virtual thunk to AthenaSealSvc::checkClass
15.53 %
PoolSvc::setObjPtr(void*&, pool::Token const*, unsigned long) const
6.07 %
15.53 %
6.07 %
AthenaSealSvc::checkClass
15.53 %
pool::PersistencySvc::PersistencySvc::readObject
6.07 %

Callgrind

CPU utilization

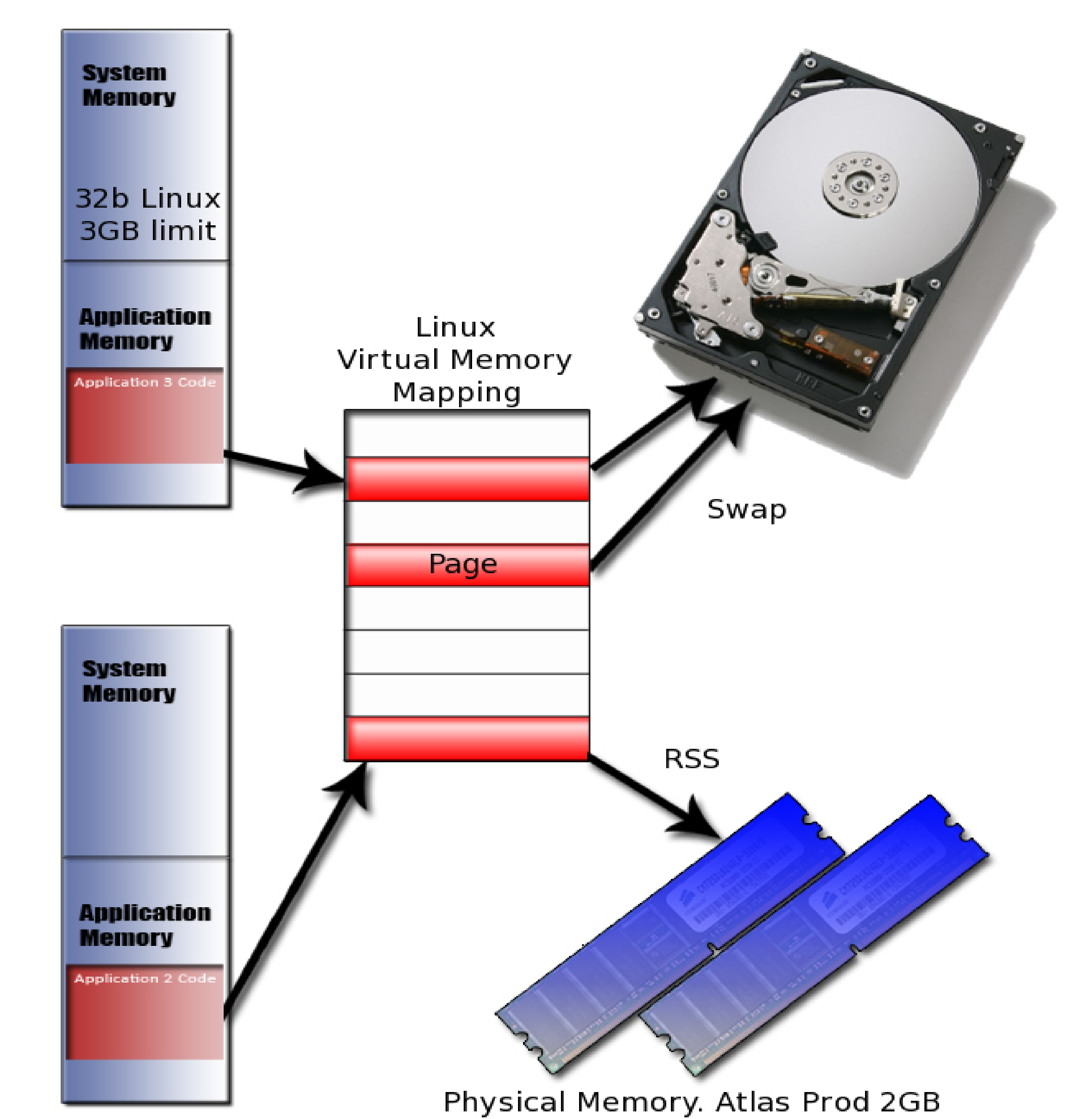
Three broad classes of changes which affect the CPU performance:

1. Physics algorithmic – the program output changes
   - Extra cuts, simulating extra or more detailed effects, "Do we run the extra trackfinder?"
2. Algorithmic, but technical – the program output does not change
   - Caching, lazy evaluation, removal of redundant calculations, data structures, etc.
3. Purely technical – the program output does not change
   - Changes related to specific issues in C++, the memory management, the operating system, the compilers and the hardware where are applications run

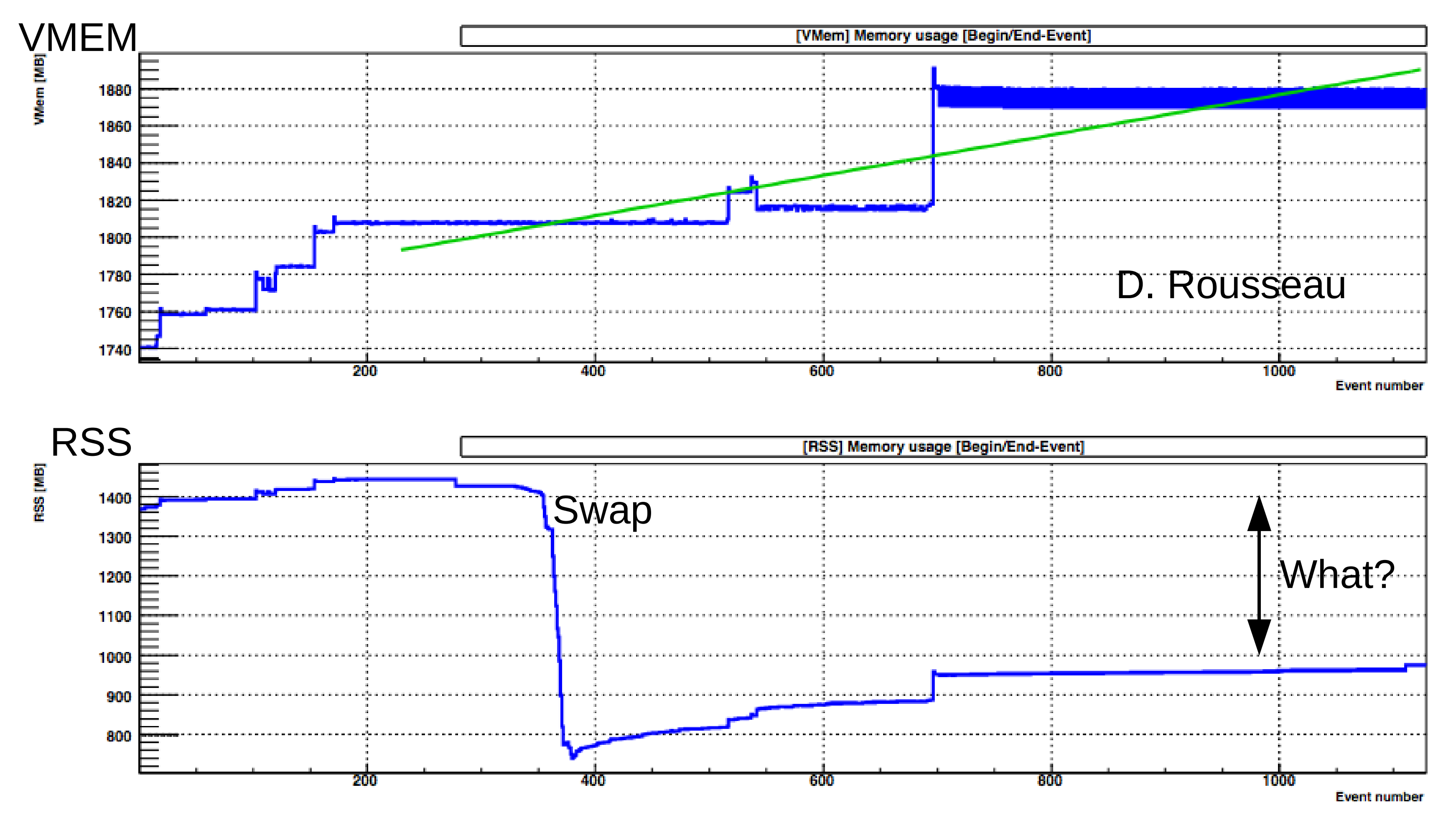
By far the largest gains/losses come from #1, of course. Improvements from #2 and #3 are "free beer" in that there are no trade-offs with physics (though there may be trade-offs between CPU use and memory, etc.)

Memory Optimization

Linux Memory Management

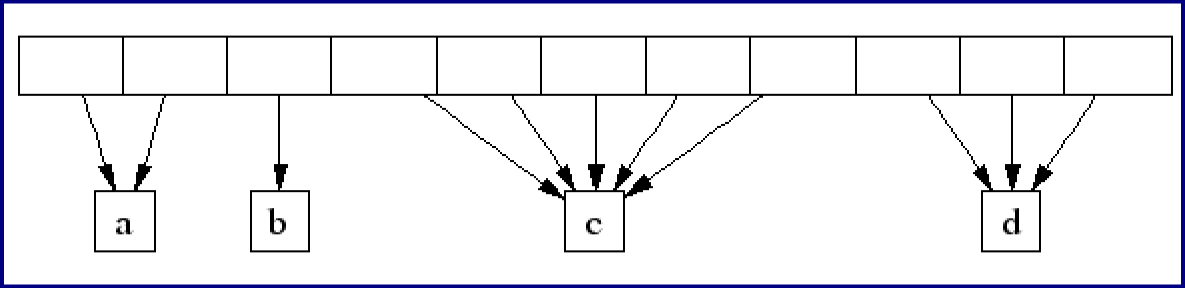


Swap in Action

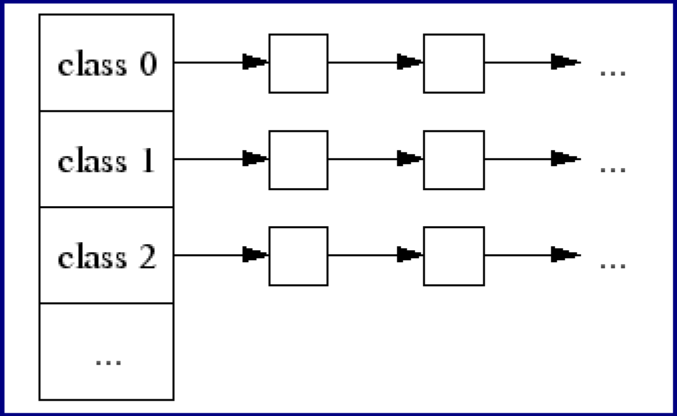


Event Number

Heap Allocation (tcmalloc)



Heap set of pages



Size object pools

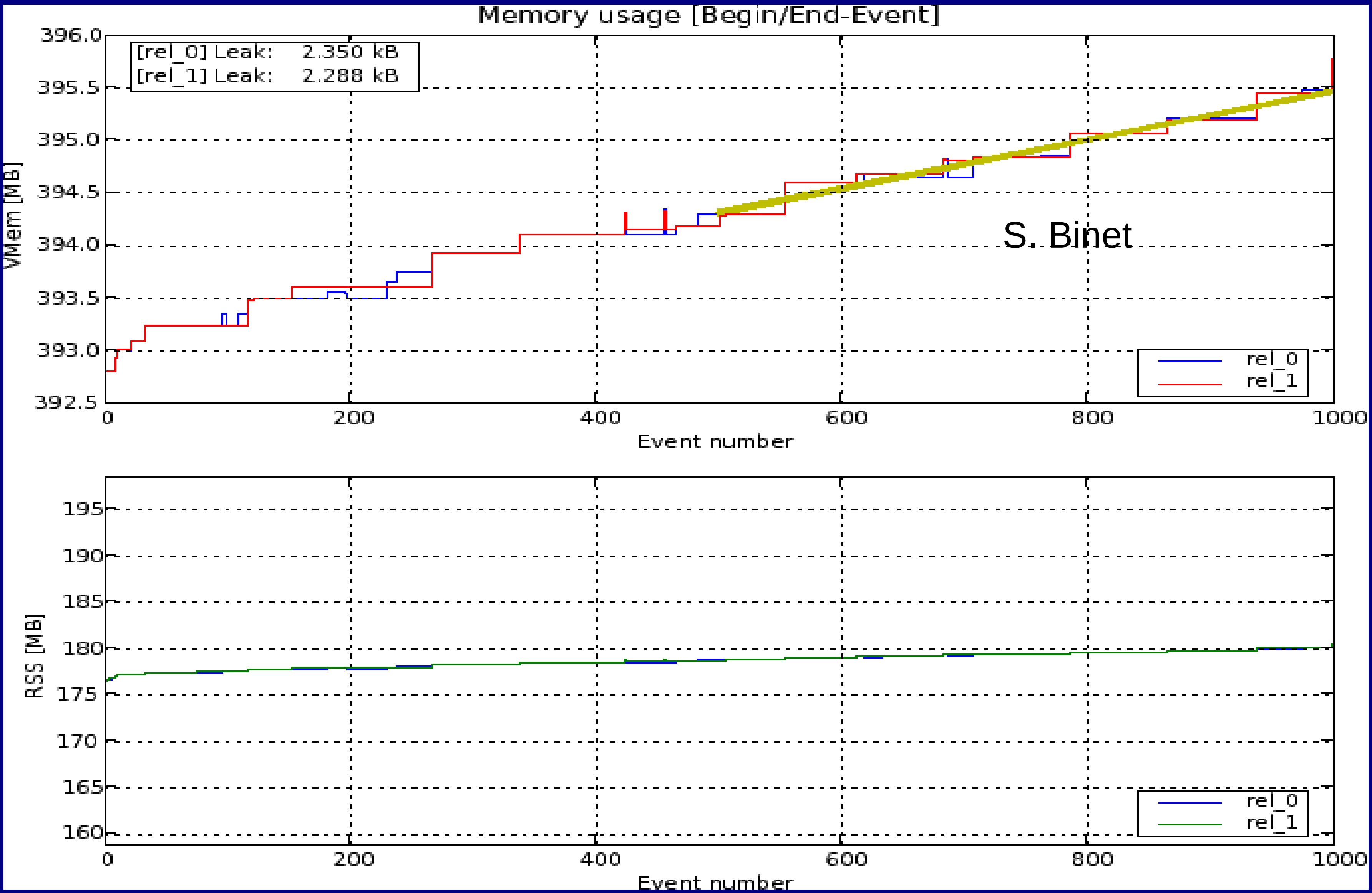
Allocation:

- Grab free entry from list

Deallocation

- Return slot to list
- Return memory “lazily”

Memory Leaks



S. Binet

Fighting Leaks: Valgrind

Interpreter tracks mallocs/frees

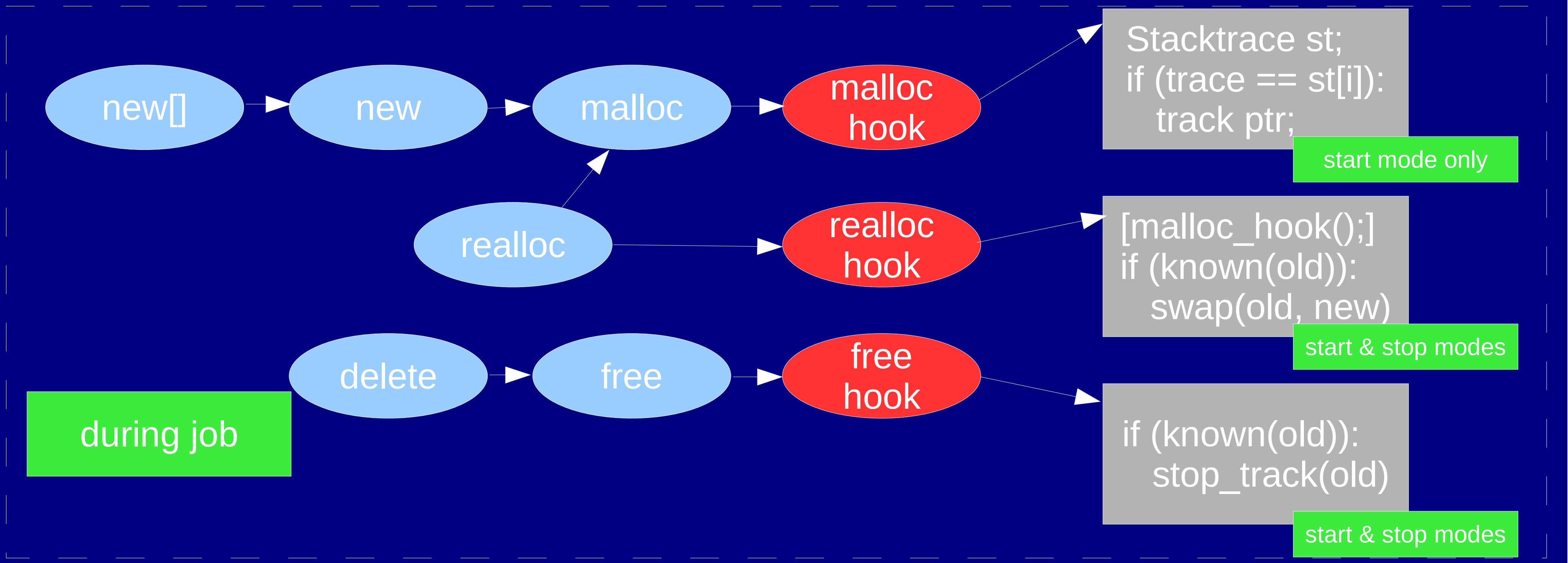
Reports leak candidates

```
==29199== 300 bytes in 15 blocks are definitely lost in loss record 1006 of 1301
==29199== at 0x3414B6D6: operator new(unsigned) (vg_replace_malloc.c:133)
==29199== by 0x3F79D4F0: Trk::TrackSummaryTool::createSummary(Trk::Track const&) (in /afs
dist/nightlies/rel/atlrel_1/Tracking/TrkTools/TrkTrackSummaryTool/TrkTrackSummaryTool-00-11-0
opt/libTrkTrackSummaryToolLib.so)
==29199== by 0x3F7BBDA0: InDet::PriVxTopAlg::m_preselect(Trk::Track const*) (in /afs/cern.ch
/nightlies/rel/atlrel_1/InnerDetector/InDetRecAlgs/InDetPriVxFinder/InDetPriVxFinder-01-00-01/i6
opt/libInDetPriVxFinder.so)
```

Slow(~10x), memory hungry(2-3x)

Lightweight: Hephaestus

Wim
Lavrijsen



```
Stacktrace st;
if (trace == st[i]):
   track ptr;
```

start mode only

```
[malloc_hook();]
if (known(old)):
   swap(old, new)
```

start & stop modes

```
if (known(old)):
   stop_track(old)
```

start & stop modes

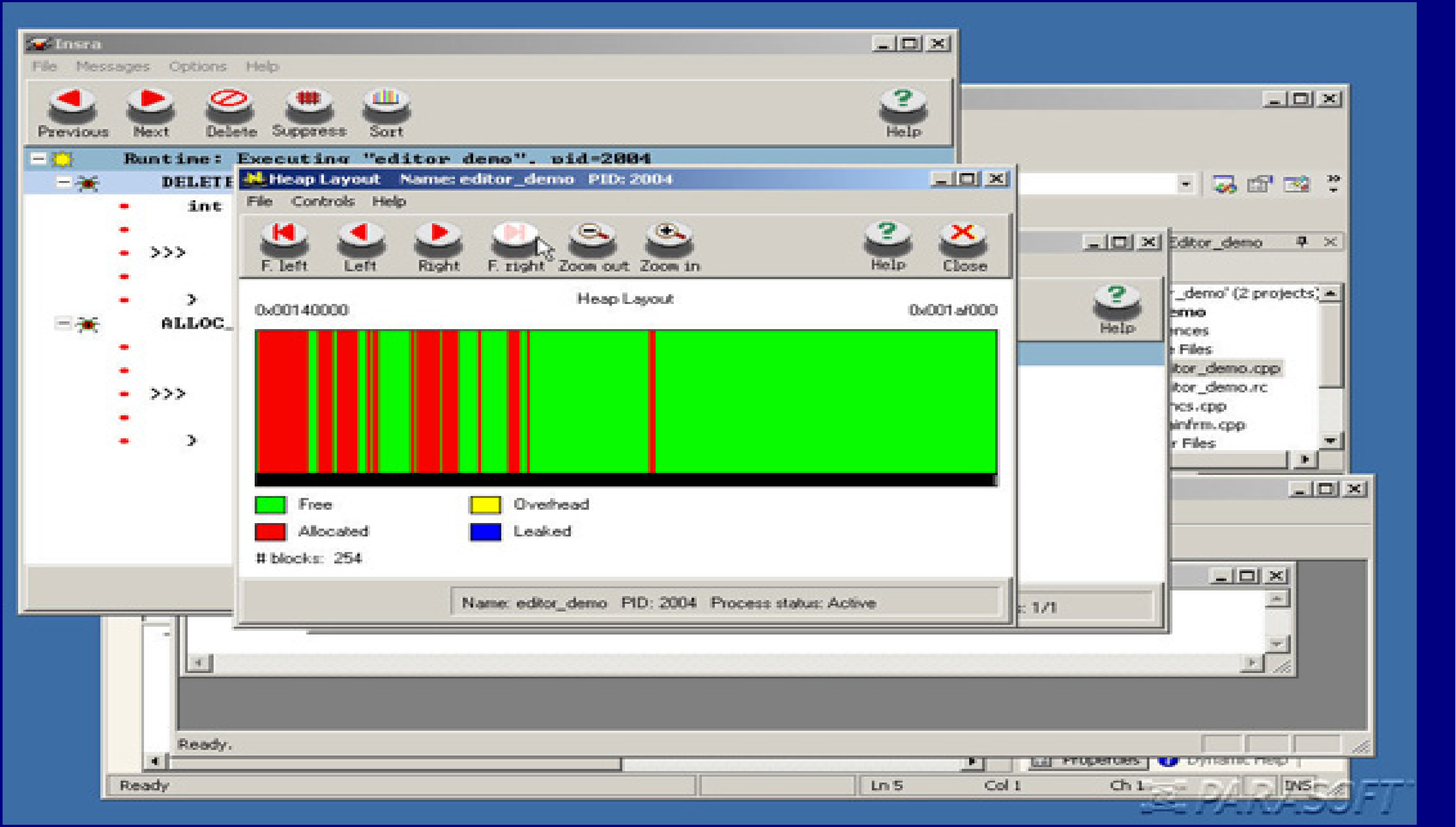
during job

```
leak detected, originating in:
CaloClusterBuilderSE::CreateImpactInCalo
        (Trk::Track const*) (28 bytes)
   0x8272ef2 CaloClusterBuilderSE::CreateImpactInCalo(Trk::Track
const*)
```

at job end

Requires preloading

Deluxe: insure++



Runtime: Executing "editor_demo", pid=2004

DELETE

int

>>>

>

ALLOC

>>>

>

🟩 Free

🟥 Allocated

🟨 Overhead

🟦 Leaked

# blocks: 254

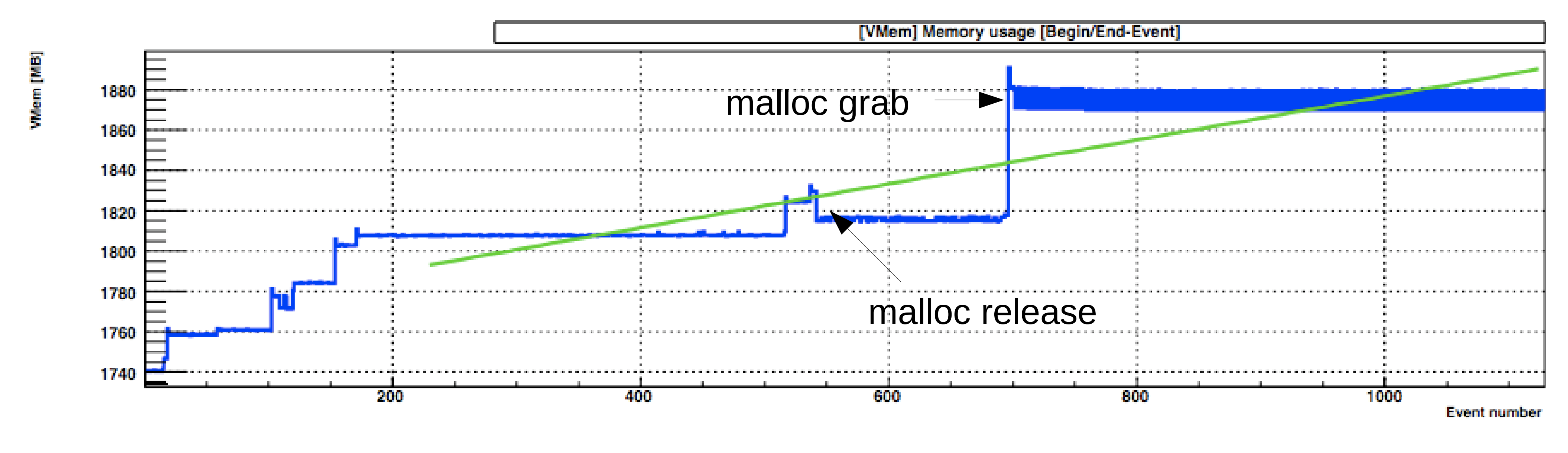
Name: editor_demo PID: 2004 Process status: Active

Ready

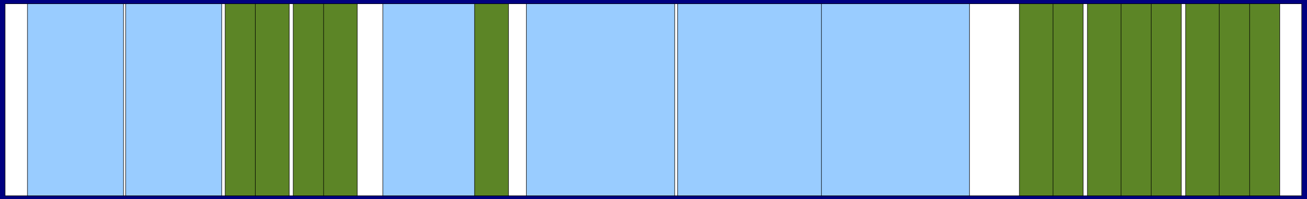
Ln 5 Col 1 Ch 1 INS

Requires special compilation

VMEM on the Rise

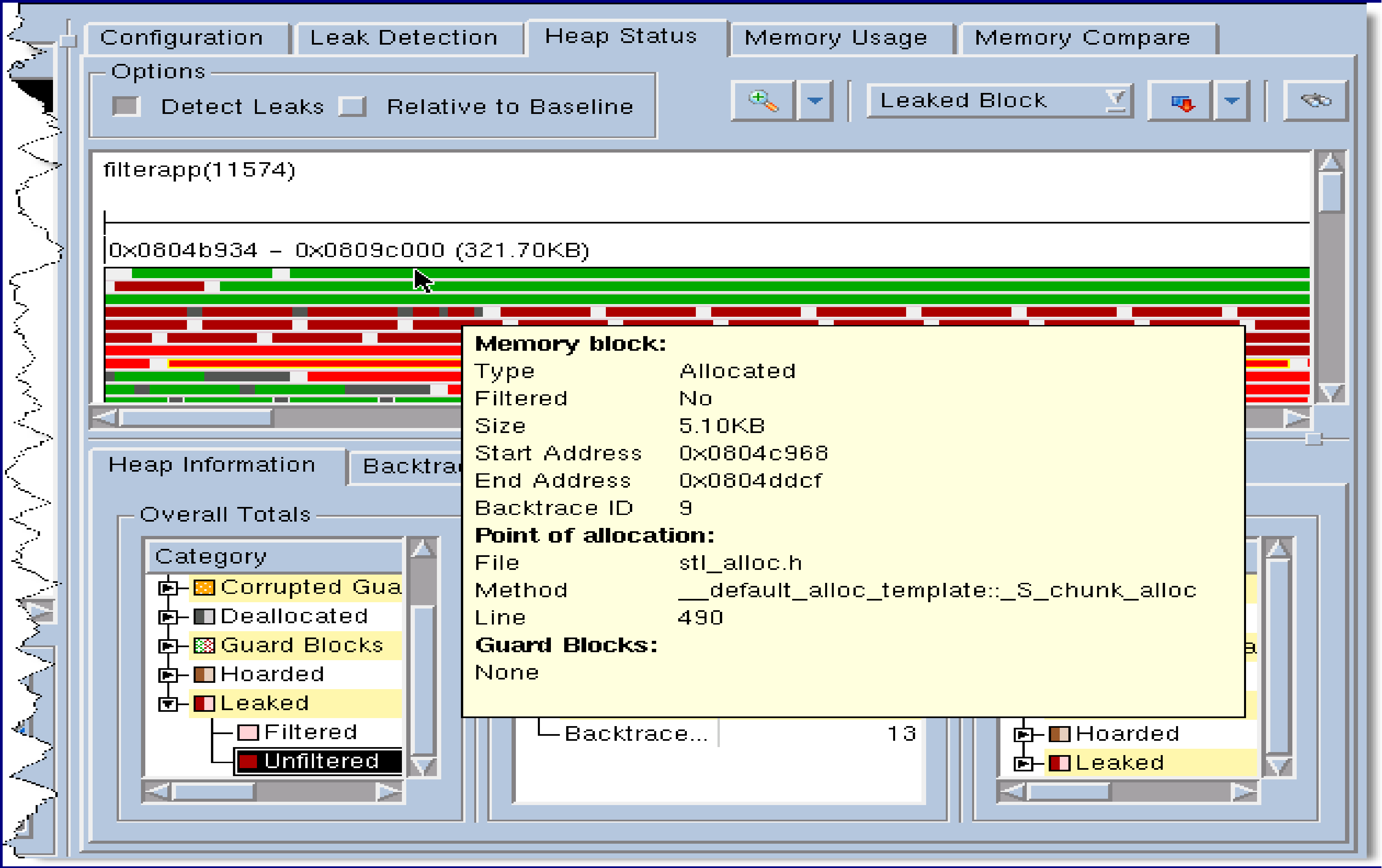


Heap Fragmentation



Possibly with different lifetimes

Visualizing Fragmentation (Totalview Memory Debugger)



filterapp(11574)

0x0804b934 – 0x0809c000 (321.70KB)

Memory block:

Type	Allocated
Filtered	No
Size	5.10KB
Start Address	0x0804c968
End Address	0x0804ddcf
Backtrace ID	9

Point of allocation:

File	stl_alloc.h
Method	__default_alloc_template::_S_chunk_alloc
Line	490

Guard Blocks:

None

Fighting Fragmentation

ATLAS ~10% memory lost to fragmentation

- Always reserve memory for containers
- Do not `new` small objects
  - If you have to (polymorphism), use a segregated allocator (e.g. memory pool)
- Do not mix long-lived and short-lived allocations
  - If you have to, consider using a private heaps

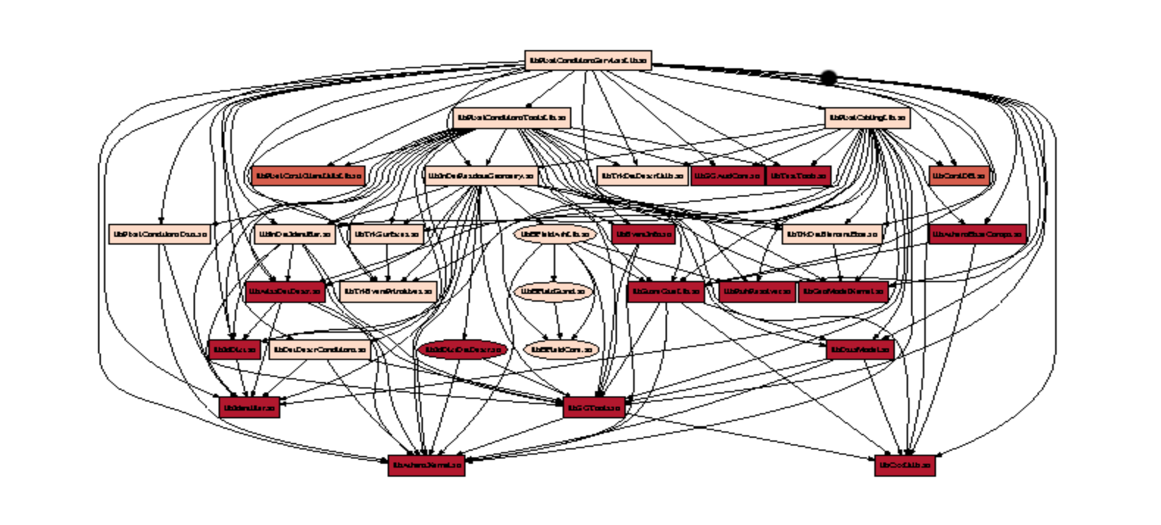
Fighting Library Bloat

Play with gcc symbols visibility

Reduce symbol duplications

- templates, statics

Optimize packaging studying dependencies



Summary

Monitor Core Performance

- Always
- Using the code that runs in production

Optimize Core Performance

- When you must
- Guided by tools
- Be aware of trade offs
- Know when to stop

Thanks

D. Brown, J. Lauret, B. Simmons, K. Ciba, M. Clemencic, M. Cattaneo, J. Apostolakis, F. Carminati, R. Brun, F. Rademakers, J. Kowalkowski, M. Paterno, S. Snyder, L. Sexton-Kennedy, D. Olson, C. Tull, S. Binet, D. Rousseau, W. Lavrijsen, M. Tatarkhanov, S. Jarp, A. Nowak, P. Elmer, Y. Yao, F. Winklmeier

Fork, COW, and vmem

Production systems must adapt to advanced memory optimization

- Athena/Gaudi MP use fork (and COW) to dramatically reduce real memory usage by aggressively sharing pages among evt workers
- Linux KSM promises to do this automatically
- We can run 4 RecExCommon instances in a 4GB machine without swapping

No use if jobs get killed based on (wrong) vmem statistics

Optimization in the Multi/Many-core Era

Scaling currently limited by I/O

Tilt CPU/Memory balance further

Cache conflicts increasingly important

Sharing memory pages crucial

- Production systems must adapt