DQM OPTIMIZATION

Vojtěch Šimetka

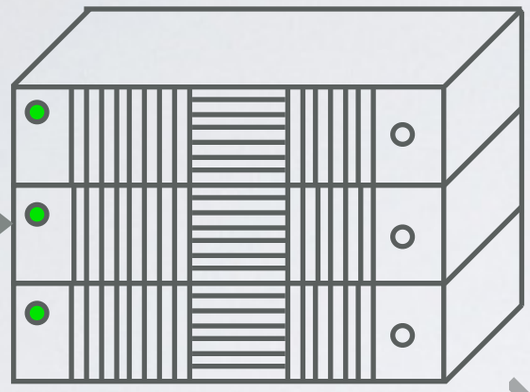
Supervisor: Federico de Guio
In collaboration with: Vincenzo Innocente
Marco Rovere

OUTLINE

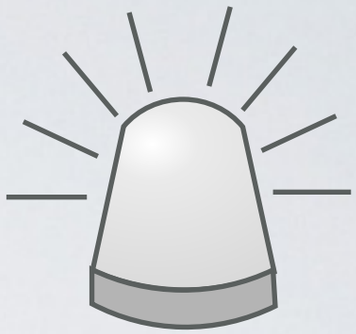
- Introduction to DQM and CMSSW
- Problems and goals
- Tools
- Results
- Good practices
- Conclusion

DQM
Online

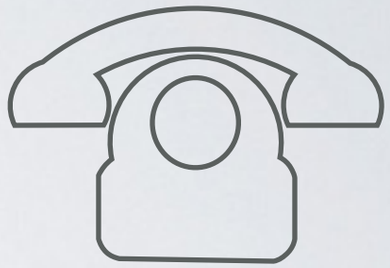
CMS
detector
data



Cluster

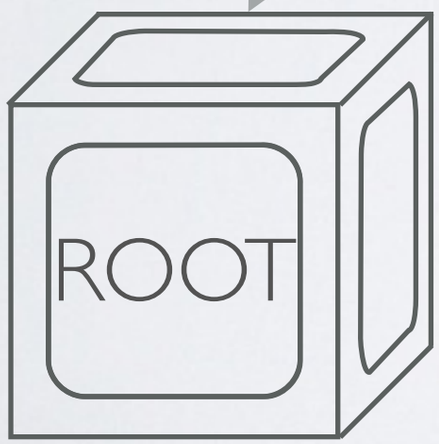


Alert



DQM
on call

DQM GUI



Archive

Online shift

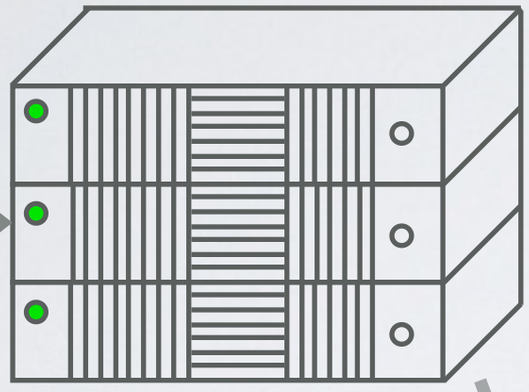
DQM

Offline

RECO

MC

data

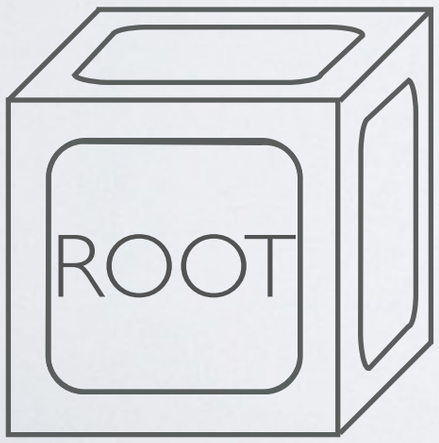


Computing infrastructure

DQM GUI

Certified runs

ROOT



Archive

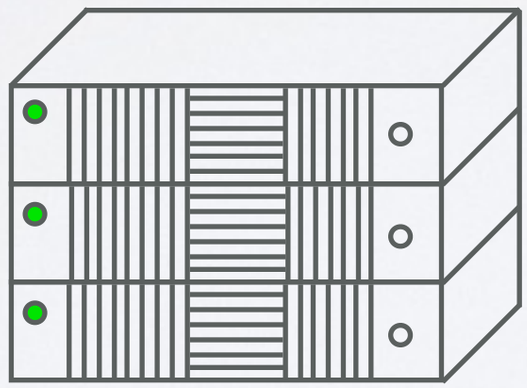
Offline review

THE GOALS

Why optimize?

A lot of data

Limited amount of processing power



2.5 GB
live memory
per job

Other limits and requirements:

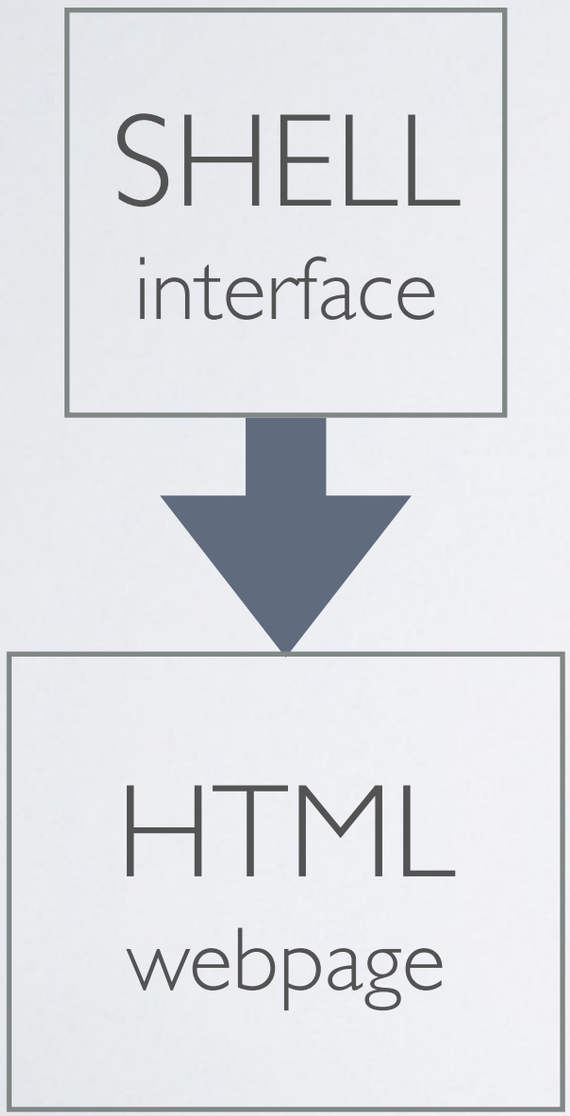
- LHC upgrade brings higher pileup
- Processing time
- Constantly running
- Release validation
- …

PROFILING TOOLS

- IgProf - memory and performance profiler
- Valgrind tools (Memcheck, Cachegrind and Massif) memory profilers
- Other tools: ps, time, gnuplot …
- runTheMatrix.py - provides workflows that are run by offline DQM

PROFILING INTERFACE

```
./profiler.sh [options] step_number WF_1 … WF_N
```



- Interface that runs in sequence selected profiling tools with one command
- Profiles standard runTheMatrix WFs
- Will be available and used by DQM team
- Customizations:
  - Pileup and bunch crossing time spacing
  - Event count

Output and description at: https://vsimetka.web.cern.ch

OUTPUT WEBPAGE

CMSSW version

Record parameters

Profiling logs

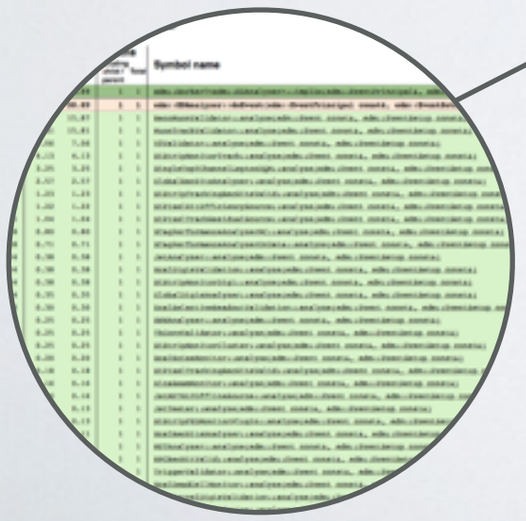
Version: CMSSW_7_2_0_pre1

Workflow	Step	BX	Pileup	Event Count	CPU Time	Memory total	Memory live	Memory live peak	Memory usage	Valgrind leaks	Cache miss and hit	Massif output	Workflow steps	Logs
20.0	3	50	70	10	show	show	show	show	show	show	show	show	show	show

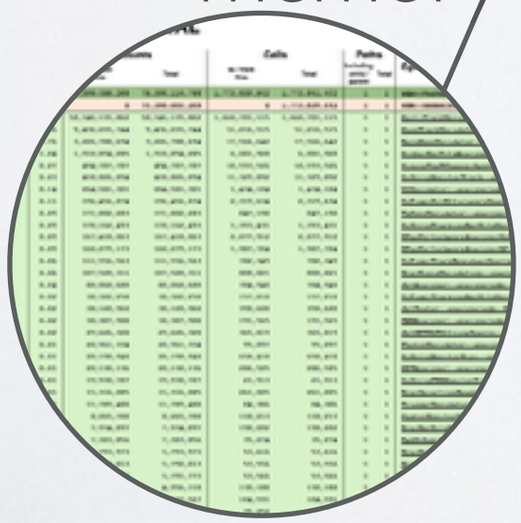
Version: CMSSW_7_2_0_pre2

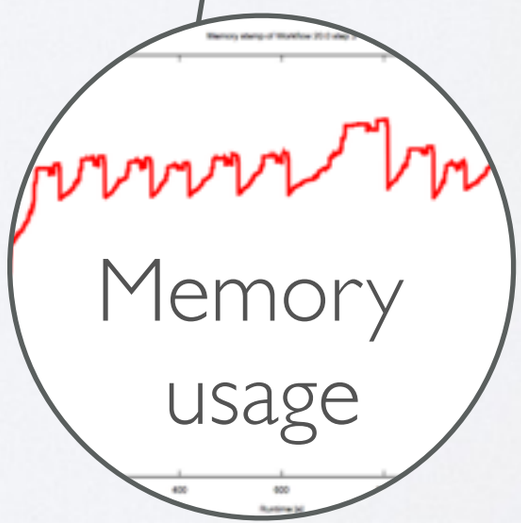
Workflow	Step	BX	Pileup	Event Count	CPU Time	Memory total	Memory live	Memory live peak	Memory usage	Valgrind leaks	Cache miss and hit	Massif output	Workflow steps	Logs
20.0	3	50	0	1000	show	show	show	show	show	N/A	N/A	N/A	show	show
20.0	3	50	70	10	show	show	show	show	show	N/A	N/A	N/A	show	show

IgProf performance



IgProf memory





Memory usage

Valgrind Cachegrind Massif

Steps commands

RESULTS

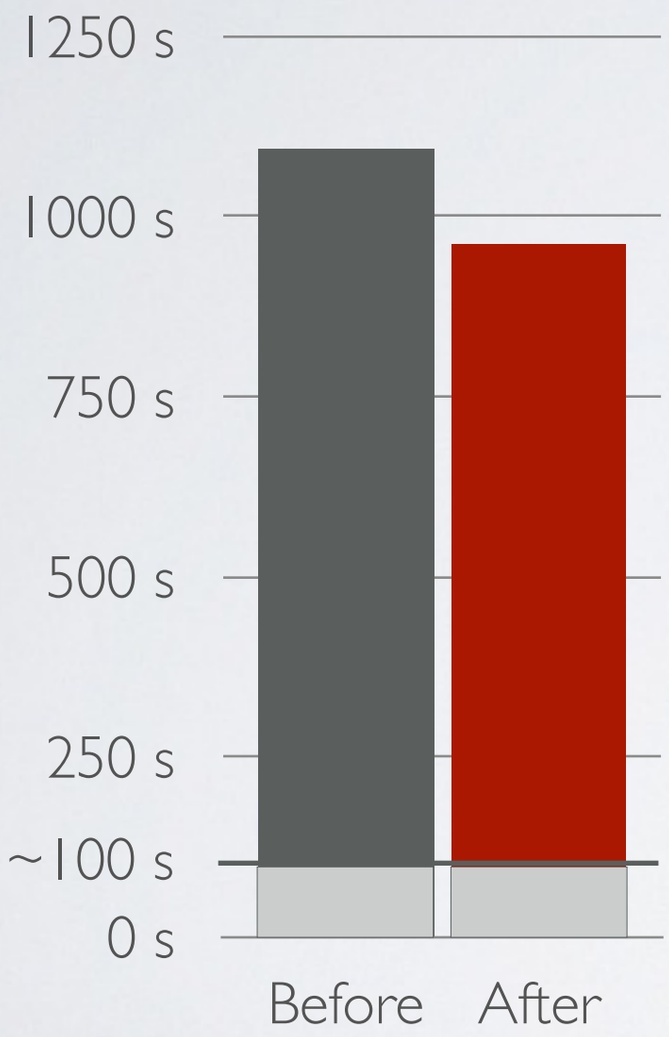
COMPARISON SETUP

- Before: CMSSW 7.2.0 - pre2 as a baseline
- After: CMSSW 7.2.0 - pre3 which includes changes we made over summer

- Workflow 20.0 (Single Muon with Transverse Momentum of 10GeV)
- step 3 (Reconstruction + DQM + Validation)
- 10 events
- Pileup 70, bunch crossing time spacing: 50ns
- Same ROOT input and configuration file for both releases

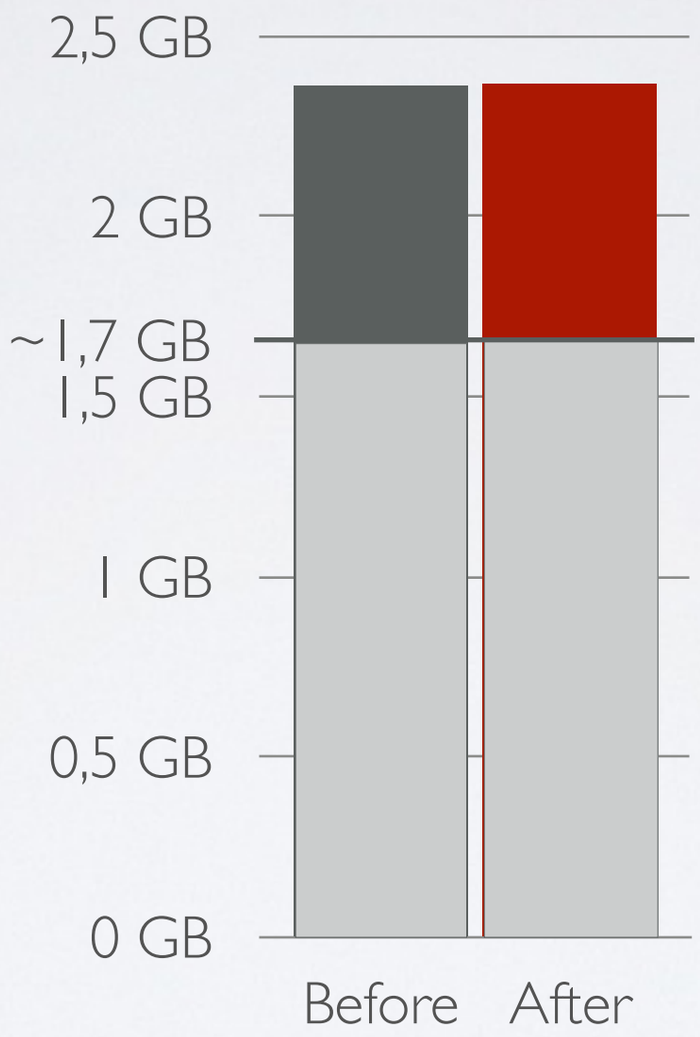
OPTIMIZATION

CPU Time



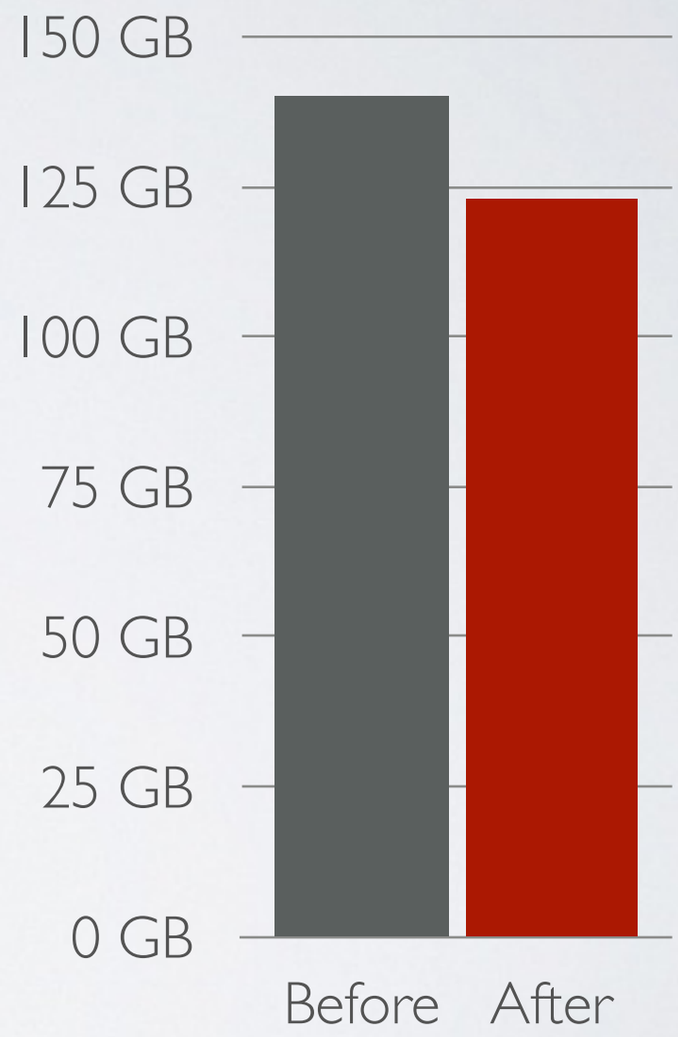
Reduction: 13,2 %

Peak Live Memory



0 %

Total Memory

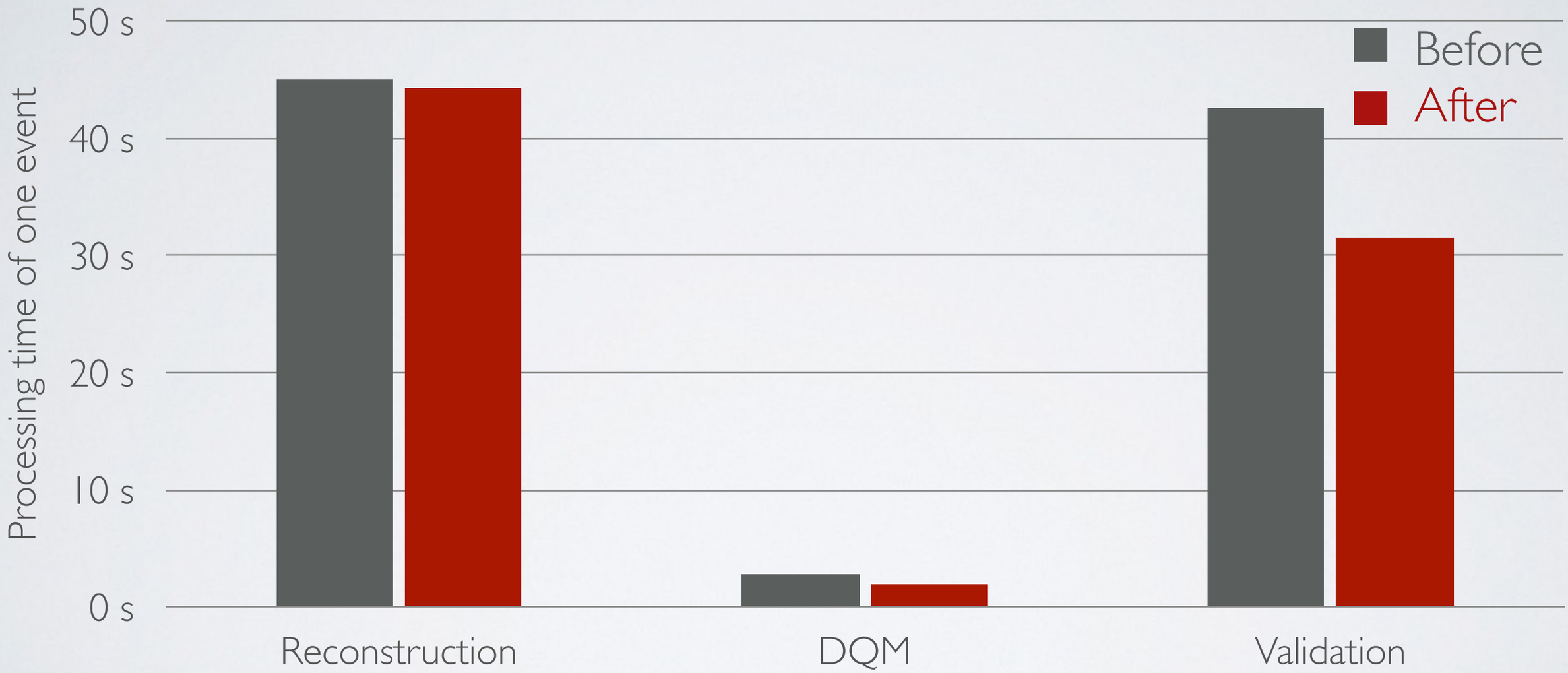


12,1 %

Impact of loading libraries, not event processing

OPTIMIZATION

CPU time per sub-step per event



Reduction: 26 %

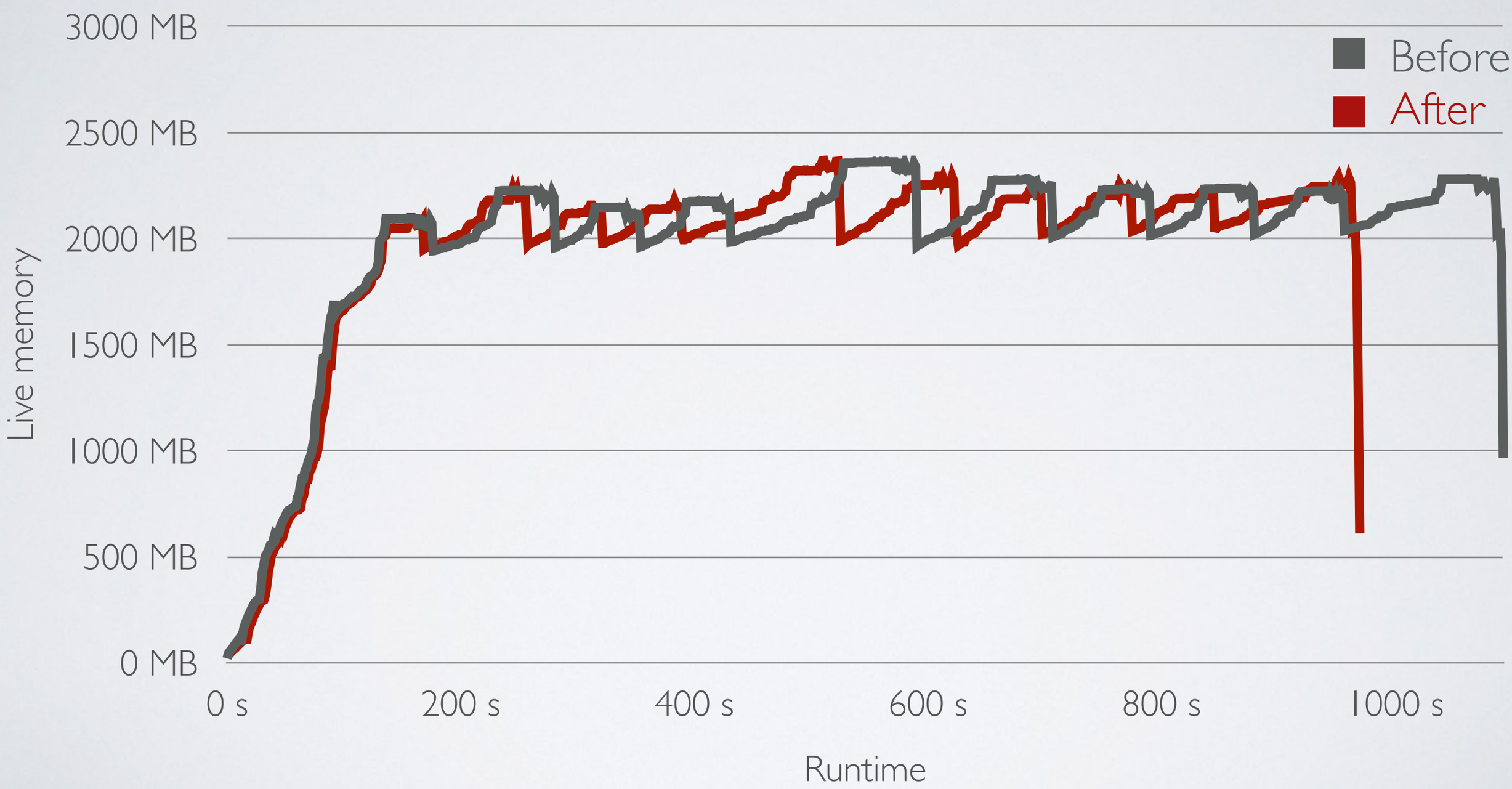
OPTIMIZATION

Modules overview

	Before		After	
	Time	Total Memory	Time	Total Memory
MultiTrackValidator	298 s	31 GB	179 s	14 GB
RecoMuonValidator	18 s	9 GB	16 s	9 GB
MuonTrackValidator	17 s	3 GB	16 s	3 GB
SiStripMonitorTrack	13 s	0 GB	4 s	0 GB

OPTIMIZATION

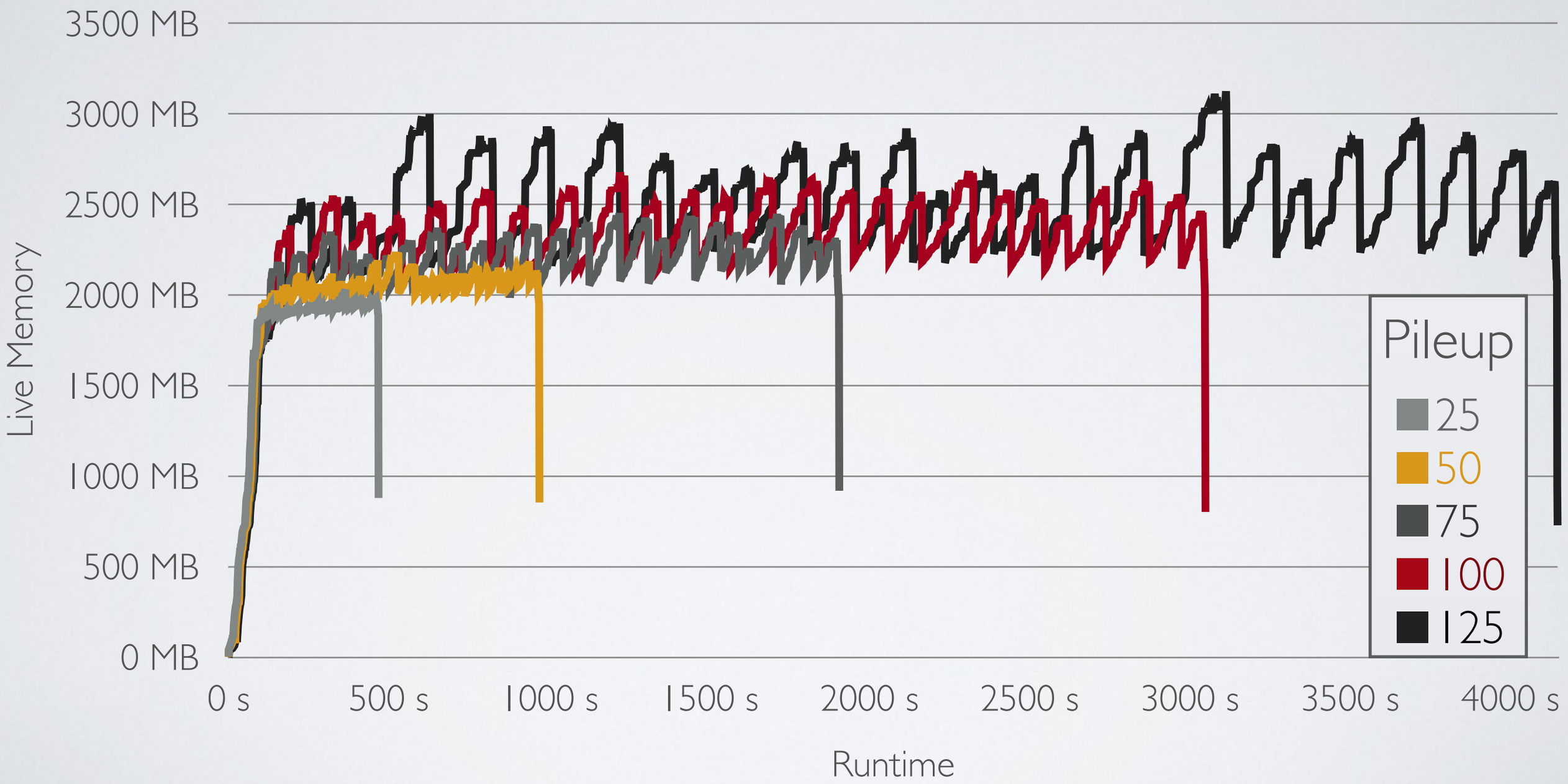
Live memory



Machine load dependent measurement

PILEUP INFLUENCE

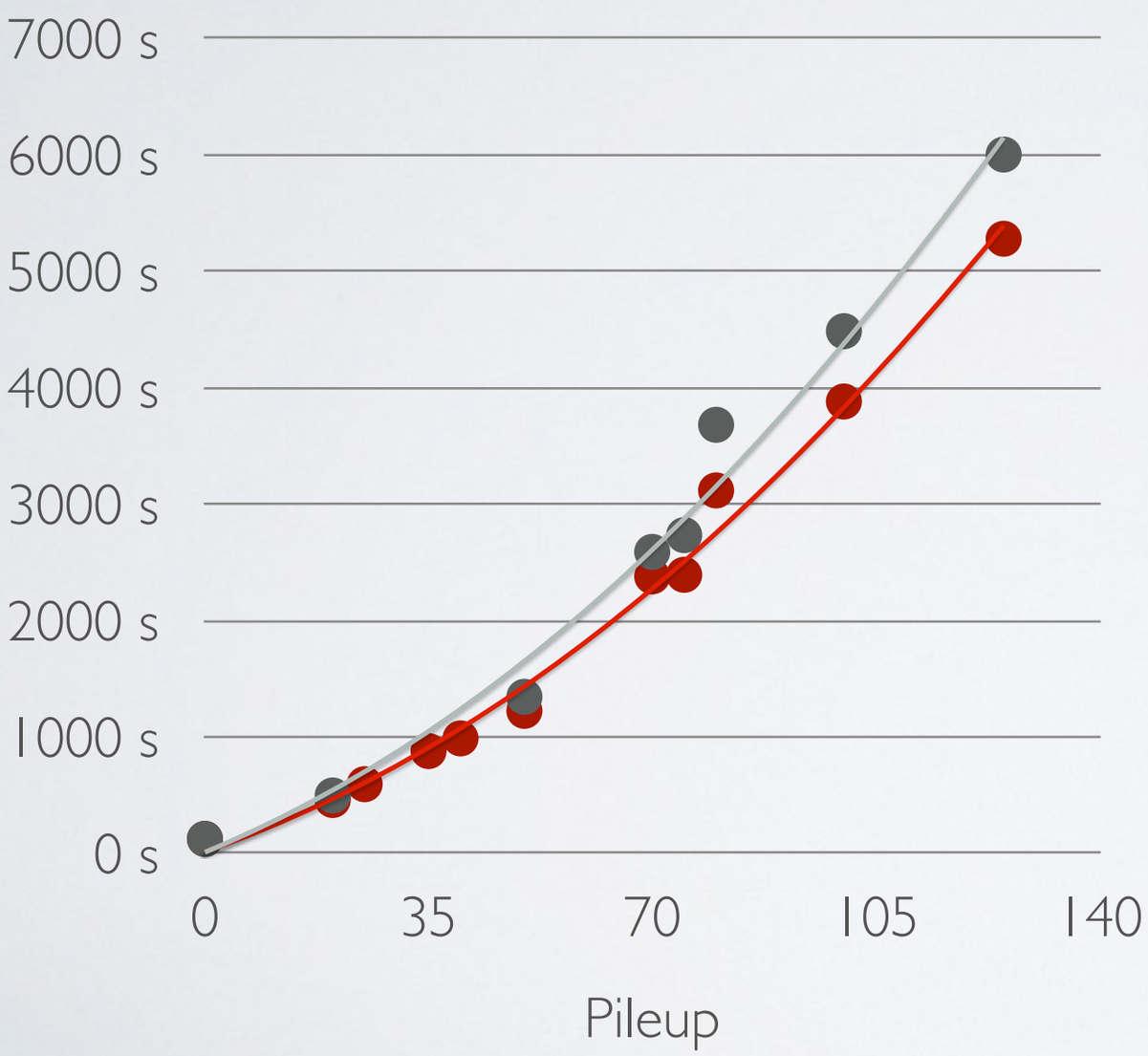
Live memory



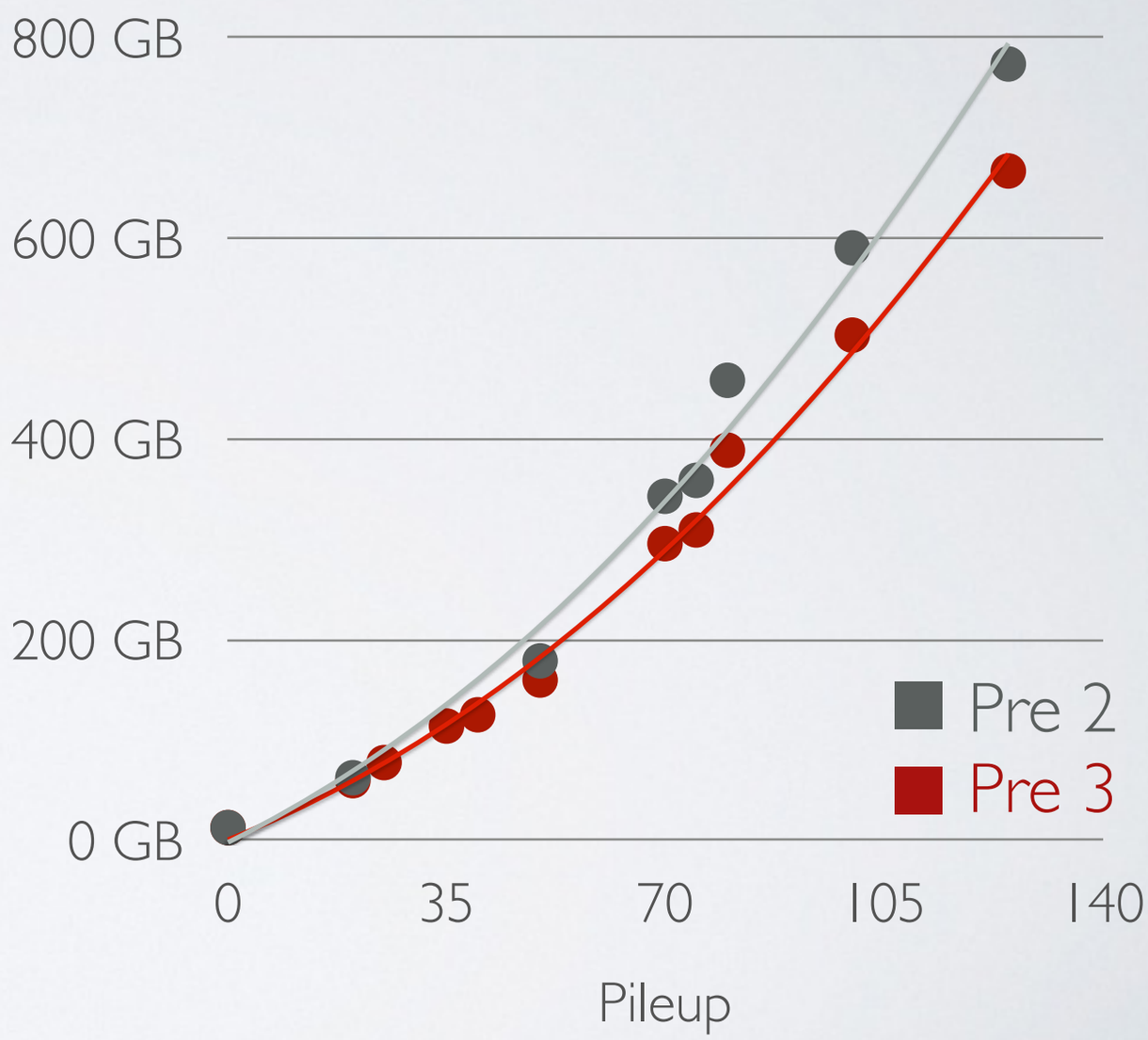
data from runTheMatrix.py workflow 20.0 step 3 with 25 events BX 50 on CMSSW 7.2.0 - pre3

PILEUP INFLUENCE

CPU Time



Total Memory



data from runTheMatrix.py workflow 20.0 step 3 with 25 events BX 50

GOOD PRACTICES - C++

Matching collections

```
collection_a // N elements
collection_b // M elements with unique identifier
```

Naive approach (vector):

```
for (auto const & a : collection_a)
{
    for (auto const & b : collection_b)
    {
        if (*a == *b)
            statement;
    }
}
```

Better approach (hashmap):

```
for (auto const & a : collection_a)
{
    if (collection_b[a->key])
        statement;
}
```

Time:

N*M ✖

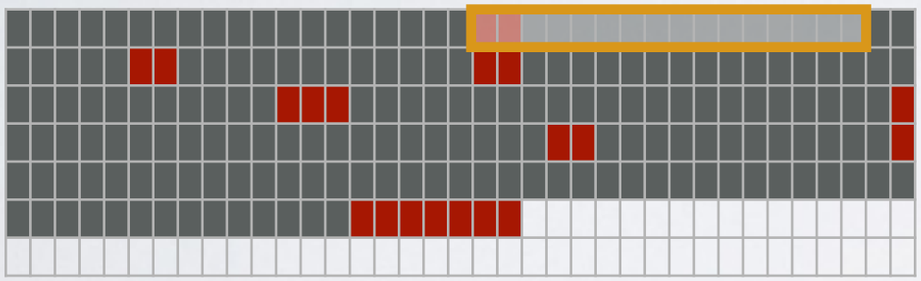
N*constant ✔

Use correct container for your data http://www.cplusplus.com/reference/stl/

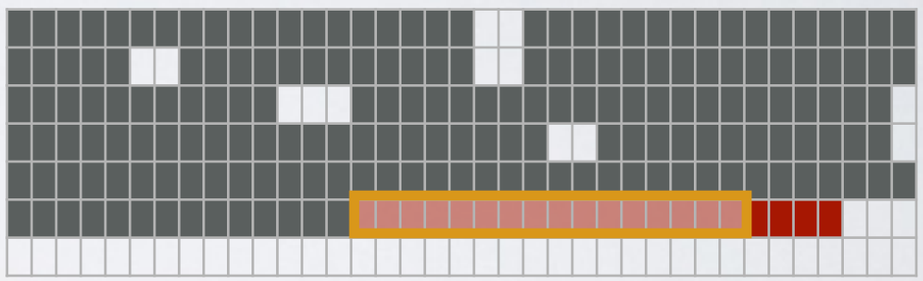
GOOD PRACTICES - C++

Locality of reference

`vector.push_back(&object)`

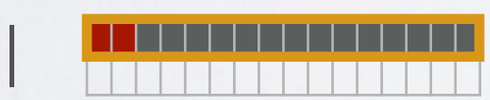


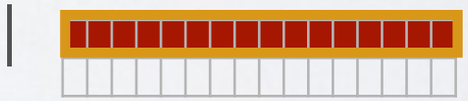
`vector.push_back(object)`



RAM
[~50 ns]

`foreach object do something`





Cache
[~1 ns]

✖

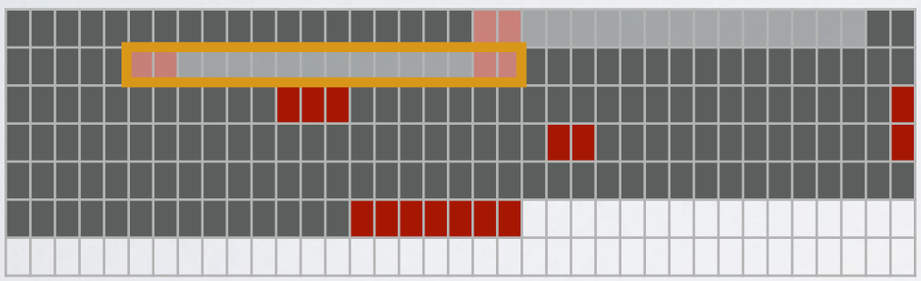
depends on size of the objects and algorithm

✔

GOOD PRACTICES - C++

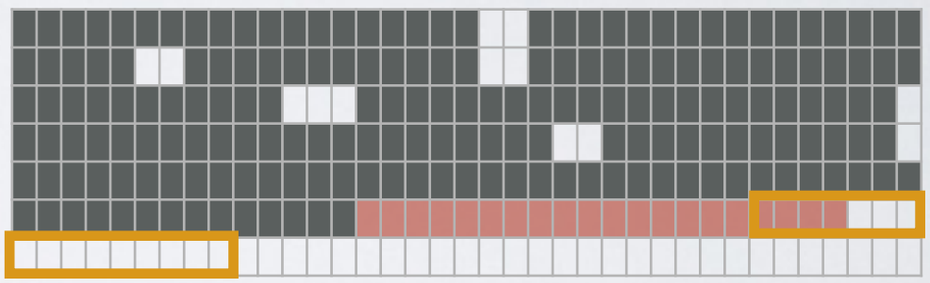
Locality of reference

`vector.push_back(&object)`

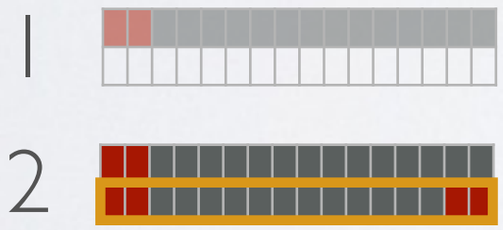


RAM
[~50 ns]

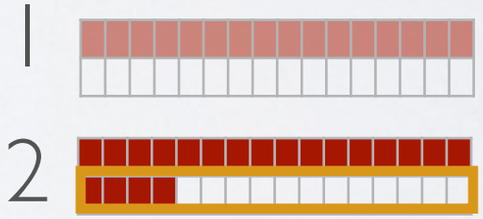
`vector.push_back(object)`



`foreach object do something`



Cache
[~1 ns]

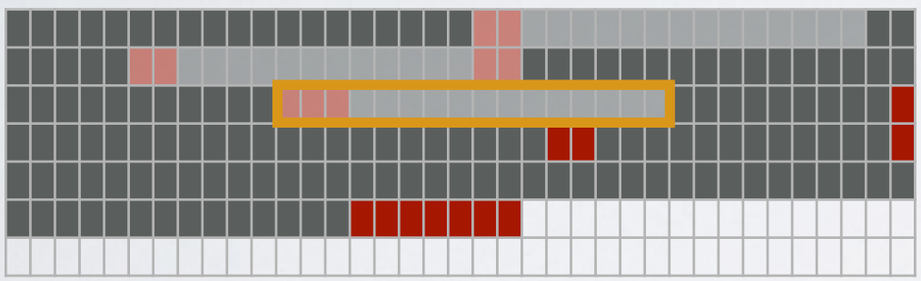


depends on size of the objects and algorithm

GOOD PRACTICES - C++

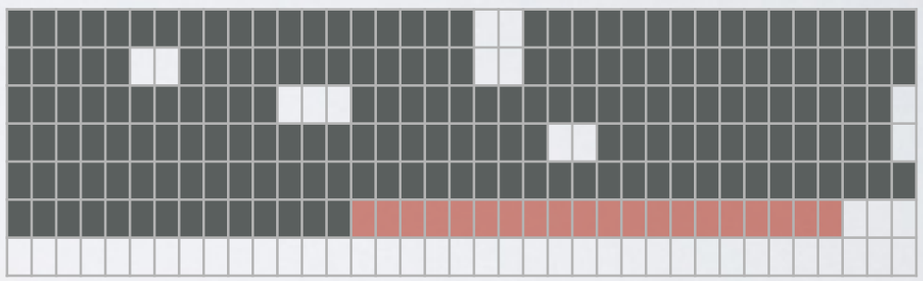
Locality of reference

`vector.push_back(&object)`

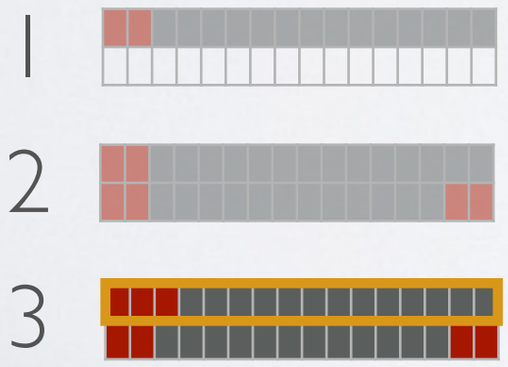


RAM
[~50 ns]

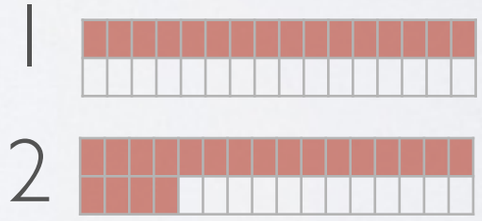
`vector.push_back(object)`



`foreach object do something`



Cache
[~1 ns]

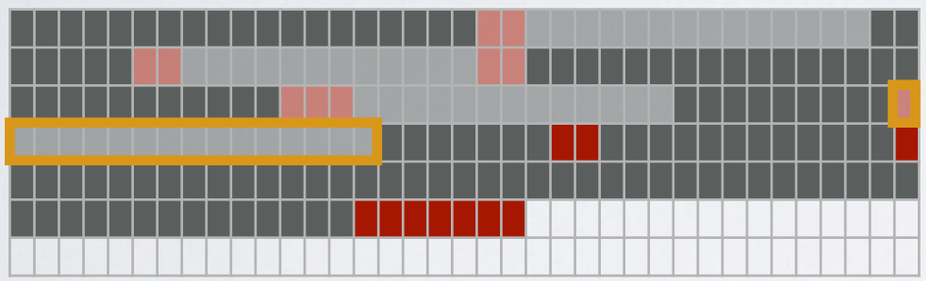


depends on size of the objects and algorithm

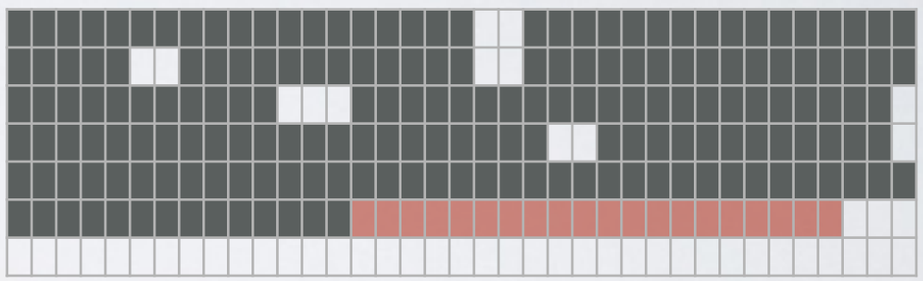
GOOD PRACTICES - C++

Locality of reference

`vector.push_back(&object)`

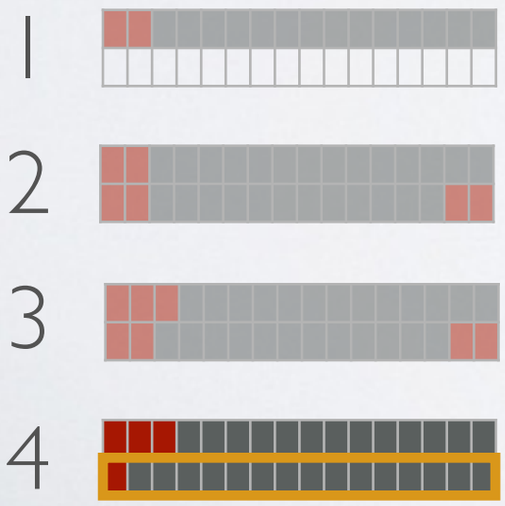


`vector.push_back(object)`

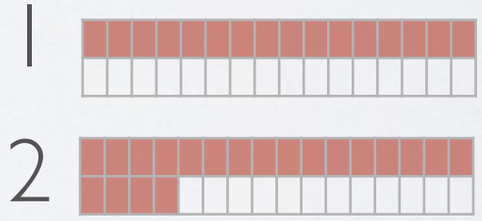


RAM
[~50 ns]

`foreach object do something`



Cache
[~1 ns]



depends on size of the objects and algorithm

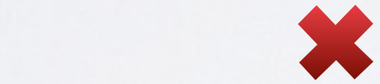
GOOD PRACTICES - C++

Memory allocation

```
new object_1 … object_N

process(object_1 … object_N)

delete object_1 … object_N
```



```
new object_1
process(object_1)
delete object_1

…

new object_N
process(object_N)
delete object_N
```

Live memory

Swap = HDD [~10 ms]
SSD [~75 µs]

Max RAM

RAM [~ 50 ns]

GOOD PRACTICES - C++

Short-circuit evaluation

```
boolean a(){…} true in 90%
boolean b(){…} true in 50%
boolean c(){…} true in 30%
boolean d(){…} true in  1%
```

```
if (a() && b() && c() && d())
{
    …
}
```

```
if (d() && c() && b() && a())
{
    …
}
```

Number of functions evaluated:

1+0.9+(0.9*0.5)+(0.9*0.5*0.3)

= 2.485 ✖

1+0.01+(0.01*0.3)+(0.01*0.3*0.5)

= 1.0145 ✔

GOOD PRACTICES - C++

Indentation and documentation

```
int func(int x)
{
if (x<0) return -1;
       if (x==0 || x==1)
return 1;
    return x*func(x-1)
}
```

```
/**
 * @brief Calculate factorial of n
 * @param n Number of series
 * @return Factorial of n or -1
 *         on error
 */
int factorial(int n)
{
    if (n<0)
        return -1;

    if (n==0 || n==1)
        return 1;

    return n*factorial(n-1)
}
```

Doxygen comments allows automatic generation of programmers documentation

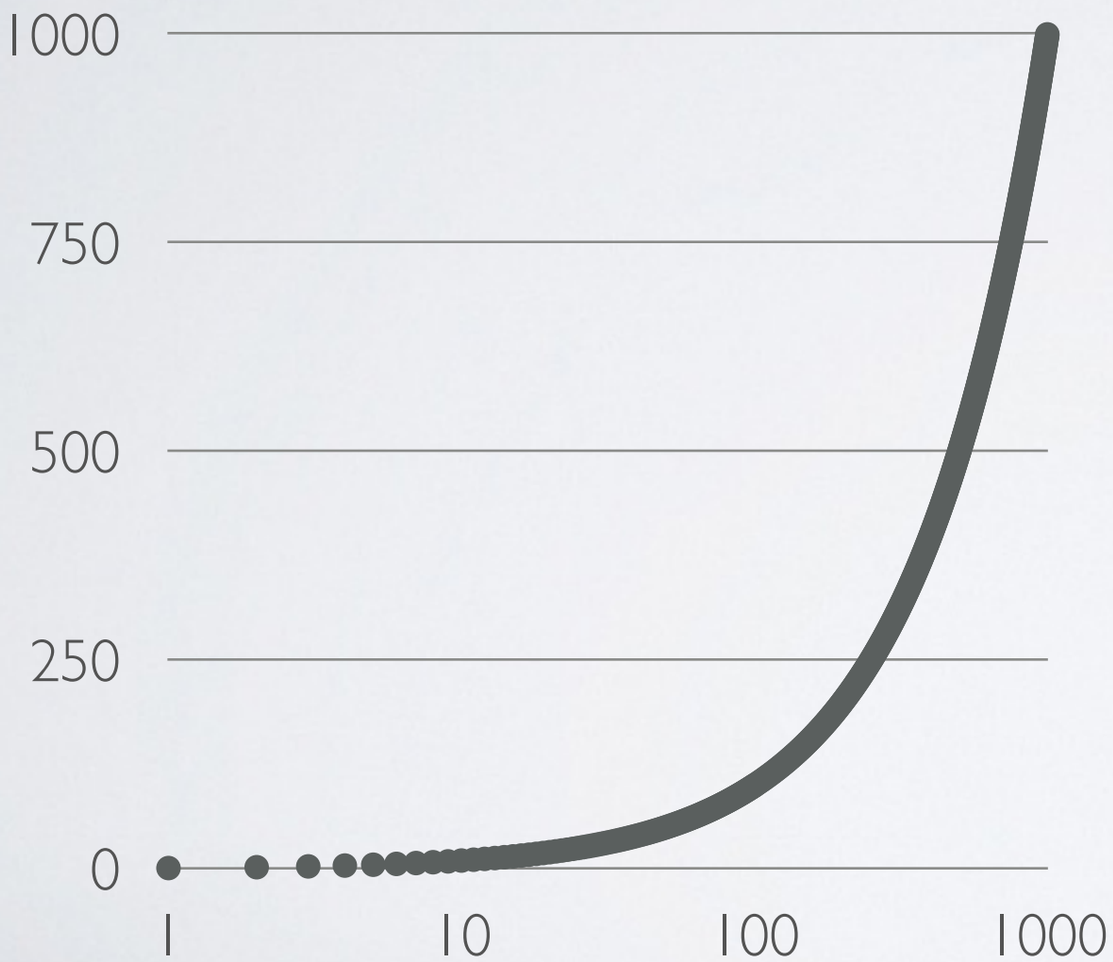
GOOD PRACTICES - C++

- Use std smart-pointers for managing allocated objects (`std::unique_ptr`)
- Avoid using pointers in collections for small objects (and allocation in general)
- Use correct container for your data http://www.cplusplus.com/reference/stl/
- Clear your containers and delete objects when you don't need them anymore. Try to create containers and objects in the innermost possible scope, if possible create them on the stack not on the heap
- Correctly indent and comment your code, use Doxygen style comments to explain what classes, functions, their parameters and other structures are used for so programmers documentation can be produced
- + CMSSW/Documentation/CodingRules and Google C++ Style Guide

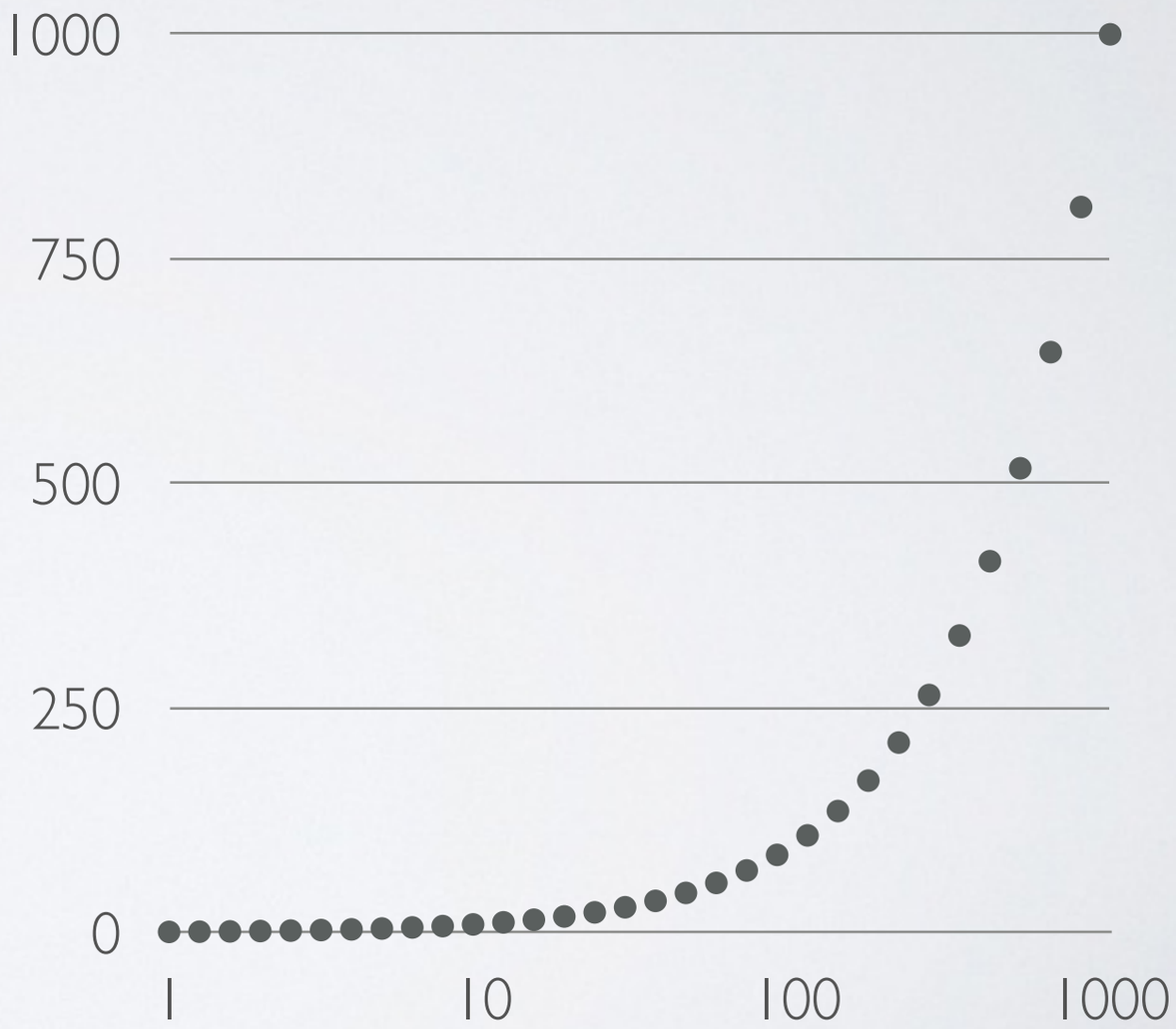
GOOD PRACTICES - DQM

Logarithmical binning

1000 bins ✖

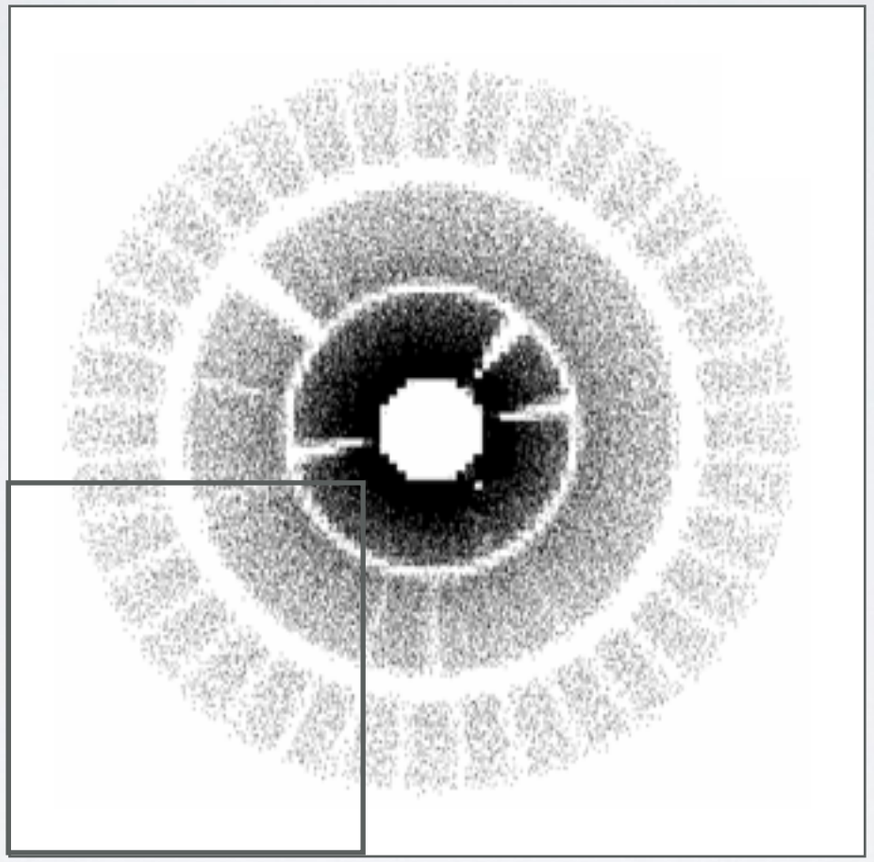


33 bins ✔



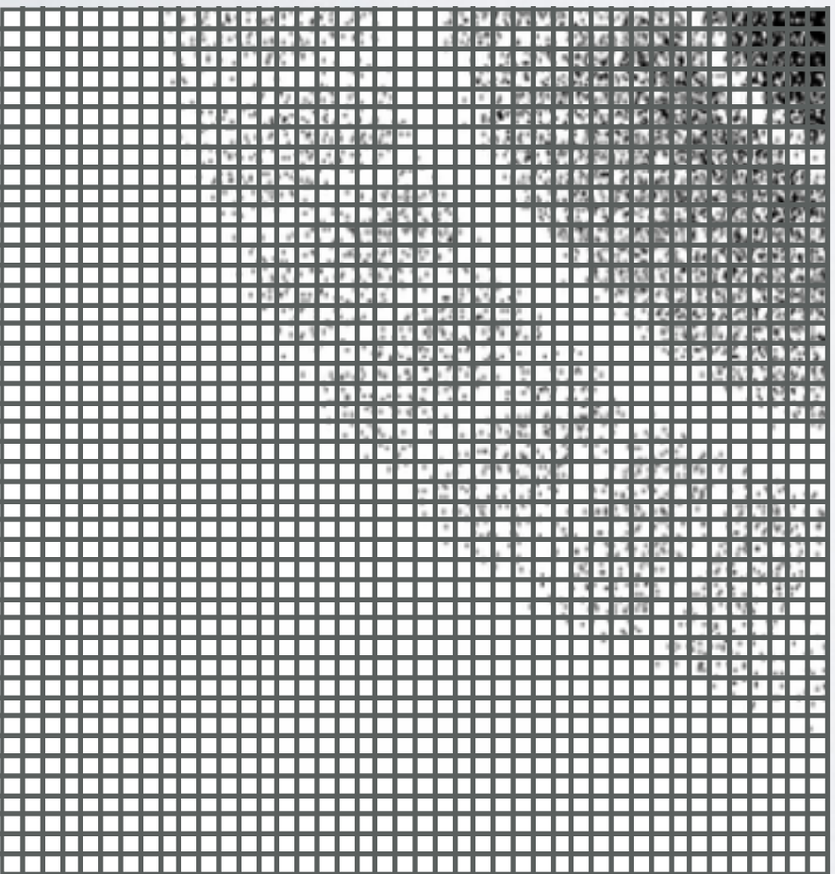
GOOD PRACTICES - DQM

Smart binning



✖

2 890 000 bins

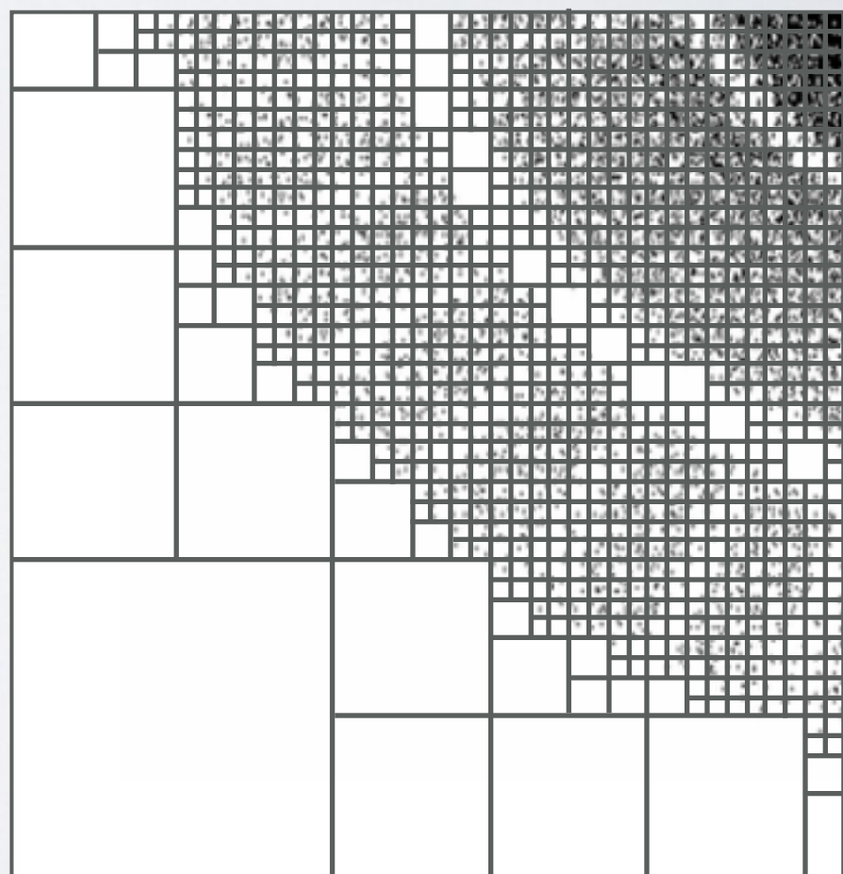


~45%

less memory

✔

~1 600 000 bins



CONCLUSION

- New easy to use tool for profiling runTheMatrix workflows, displaying and archiving reports
- Improved Reconstruction + DQM + Validation step performance by 13.2% and total memory consumption by 12.1%
  - Validation sub-step performance improved by 26%
  - Allows to process extreme scenarios with high pileup
- Acquired new knowledge:
  - profiling tools, collaborating, git, UNIX and Shell…
  - better understanding of physics problems, CMSSW and detector

REFERENCE

1. Atanas Batinkov, *DQM poster*, 2013, Seoul, Korea
2. Federico de Guio, *The Compact Muon Solenoid Experiment*, 2013, CHEP2013 Computing in High Energy Physics 2013
3. Atanas Batinkov, Marco Rovere, Laura Borrello, Federico De Guio, Dan Duggan, Salvatore Di Guida, *The CMS Data Quality Monitoring Software: Experience and Future Improvements*, 2013
4. IgProf, igprof.org
5. Valgrind tools (Memcheck, Cachegrind and Massif), valgrind.org
6. runTheMatrix, https://twiki.cern.ch/twiki/bin/view/CMSPublic/CompOpsRelValWMAOperations
7. C++ collections, www.cplusplus.com/reference/stl/
8. Locality of reference, http://en.wikipedia.org/wiki/Locality_of_reference
9. Short-circuit evaluation, http://en.wikipedia.org/wiki/Short-circuit_evaluation
10. Doxygen, www.stack.nl/~dimitri/doxygen/
11. CMSSW Doxygen documentation, https://cmssdt.cern.ch/SDT/doxygen/CMSSW_7_1_3/doc/html/index.html
12. CMSSW coding rules, https://github.com/cms-sw/cmssw/tree/CMSSW_7_2_X/Documentation/CodingRules
13. Google C++ style guide, google-styleguide.googlecode.com/svn/trunk/cppguide.xml