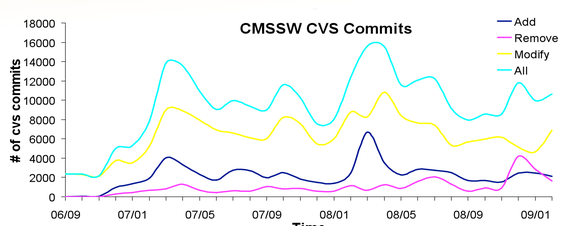
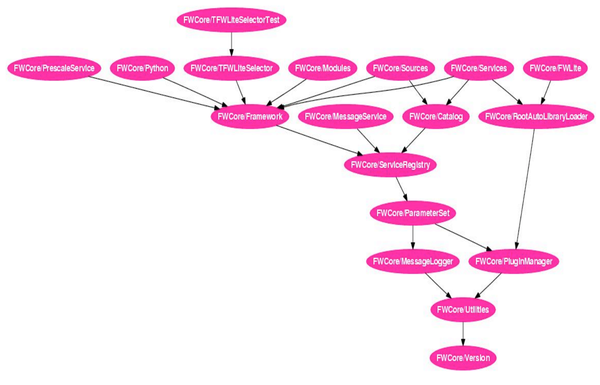
# Optimization of the CMS software build and distribution system

Giulio Eulisse, Shahzad Muzaffar
Northeastern University, Boston
On behalf of the CMS Offline and Computing Projects

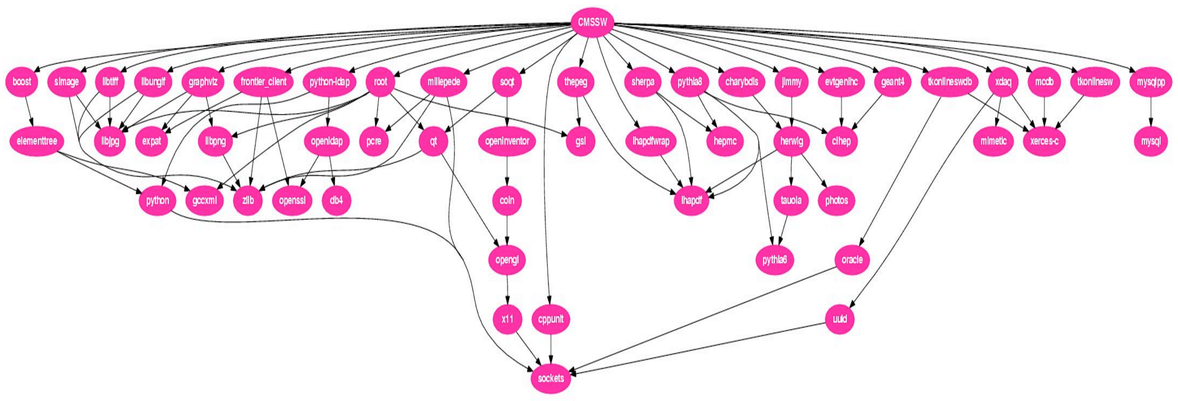
## Introduction

Optimal build, release and distribution of CMS software, which is actively developed by hundreds of developers all over the world, is quite a challenge. Over 2400 shared libraries, plugins, and executables are generated out of two million lines of *CMSSW* (CMS Software) code, which is divided in 1100 individual *Packages* organized in 100 *Sub-Systems*. Its dependency on more than hundred external software packages make its build and distribution more complex.

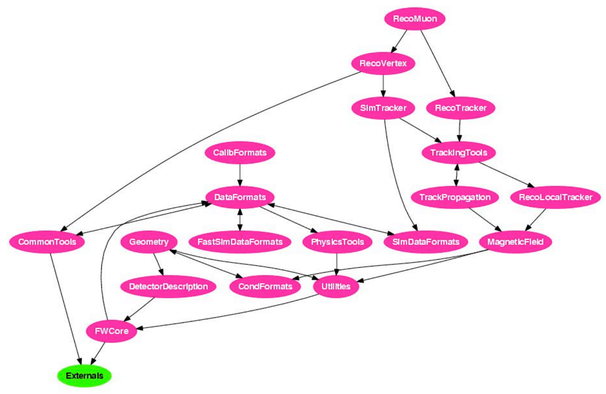



CMSSW Direct Dependencies:
Packages of FWCore Subsystem
(Restricted to FWCore only)




CMSSW External Tools Dependencies




CMSSW Direct Dependencies
SubSystem level Dependency of RecoMuon

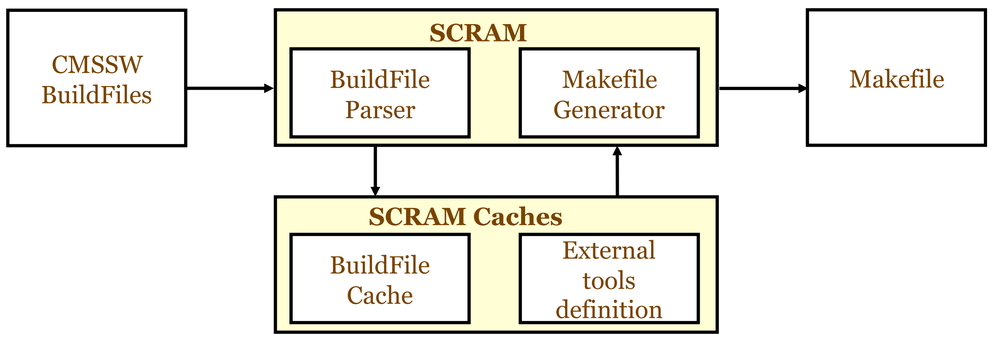
## Objectives

- Hide all the complex dependency information from software developers
- Provide an easy and fast way so that developers can work on few packages without the need of rebuilding every thing
- Easy and consistent way of distributing CMS software

## Problems

CMS uses *SCRAM* to achieve the first two objectives. *SCRAM* transforms the user defined build rules into *gmake* rules. Developers define dependencies in simple text file (*BuildFile*) and *SCRAM* takes care of the rest.

For software distribution, CMS has been using *APT* repository. *PKGTOOLS* are used to build the *RPM*s for *CMSSW* and its externals.




### CMS software build issues

Increasing number of CMS packages and dependency on large number of external tools made *SCRAM* very slow.

- In developer's area with few packages *SCRAM* overhead was more than the compilation time
- Memory usage of *SCRAM* went over 1GB
- *SCRAM* generated internal caches grow over 100MB
- Generated *Makefile* went over 75MB
- Build rules were not allowing to even build code in parallel

### External tools build and distribution issues

- External tools were built one at a time so it was taking too much time
- Often downloading a *CMSSW* version brought in multiple versions of same external tool
- For simple patches, full *CMSSW* release was to build and distribute

## Solutions

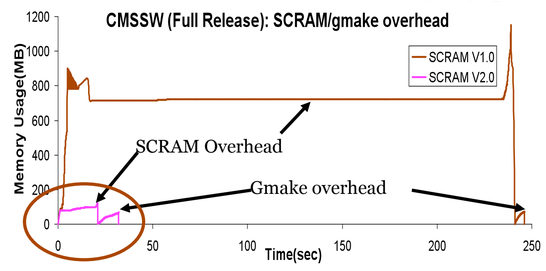
Instead of looking in to new tools and migrating whole CMS community to learn new tools, we looked for the cause of the problems and fixed them.

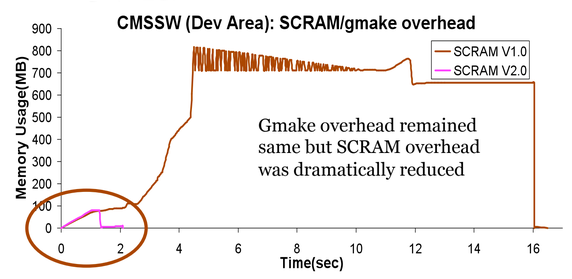
### Dependency checking

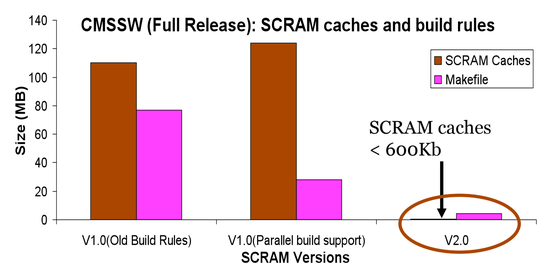
*SCRAM* was taking most of its time resolving packages and externals tools dependencies and reading its large internal cache files. As *gmake* is much faster in keeping track of dependencies so why bother doing it in *PERL*.

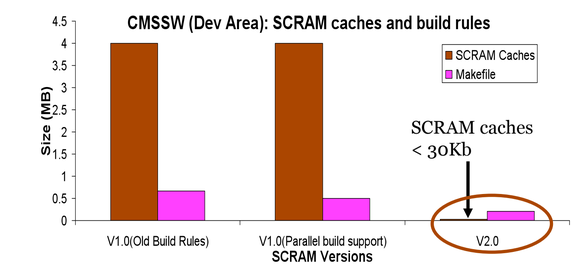
Reducing *SCRAM* internal cache size and moving all the dependency tracking work into *gmake* fixed many things. *SCRAM* overhead was dramatically reduced (much smaller processing time, memory usage and disk space).

## Solutions



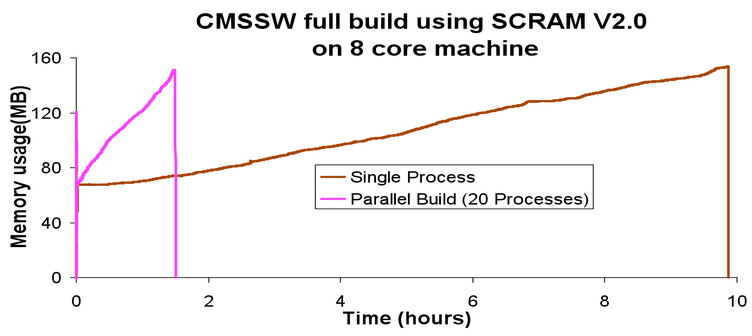




### Parallel compilation to reduce build time

In order to reduce build time, CMS build rules were modified to allow parallel build (*gmake -j n*). This allowed us to make use of multi core machines, which resulted in much faster builds.




New *CMSSW* build rules allowed us to build few big shared libraries instead of thousands of small ones and help us identifying issue like

- A lot of *C++* templates code being replicated in many small library/plugin
- Copying of files between different packages resulted in same symbols defined in many libraries which mean unpredictable runtime behavior.

### External tools build and distribution

New *PKGTOOLS* and spec files improved many aspects of the distribution

- Build things in parallel which do not depend of each other. This resulted in much faster build of externals tools (couple of hours to build all externals)
- Always rebuild a tool for which any base level tool is modified. This avoided the distribution of multiple version of same tool
- *CMSSW* patch release setup to only build and distribute a limited number of *CMSSW* packages
- Automatically create external tool definition files for *SCRAM*

## Results

- Developers have more time to spend on their code instead of waiting for compilation to finish
- Multiple *Integration Builds* per day for all supported platforms and different *CMSSW* release series
- Couple of hours to build, release and distribute a full *CMSSW* release
- Distribution size reduction mean fast download and deployment of CMSSW software at remote sites

Shahzad.Muzaffar@cern.ch