CMS Offline Software

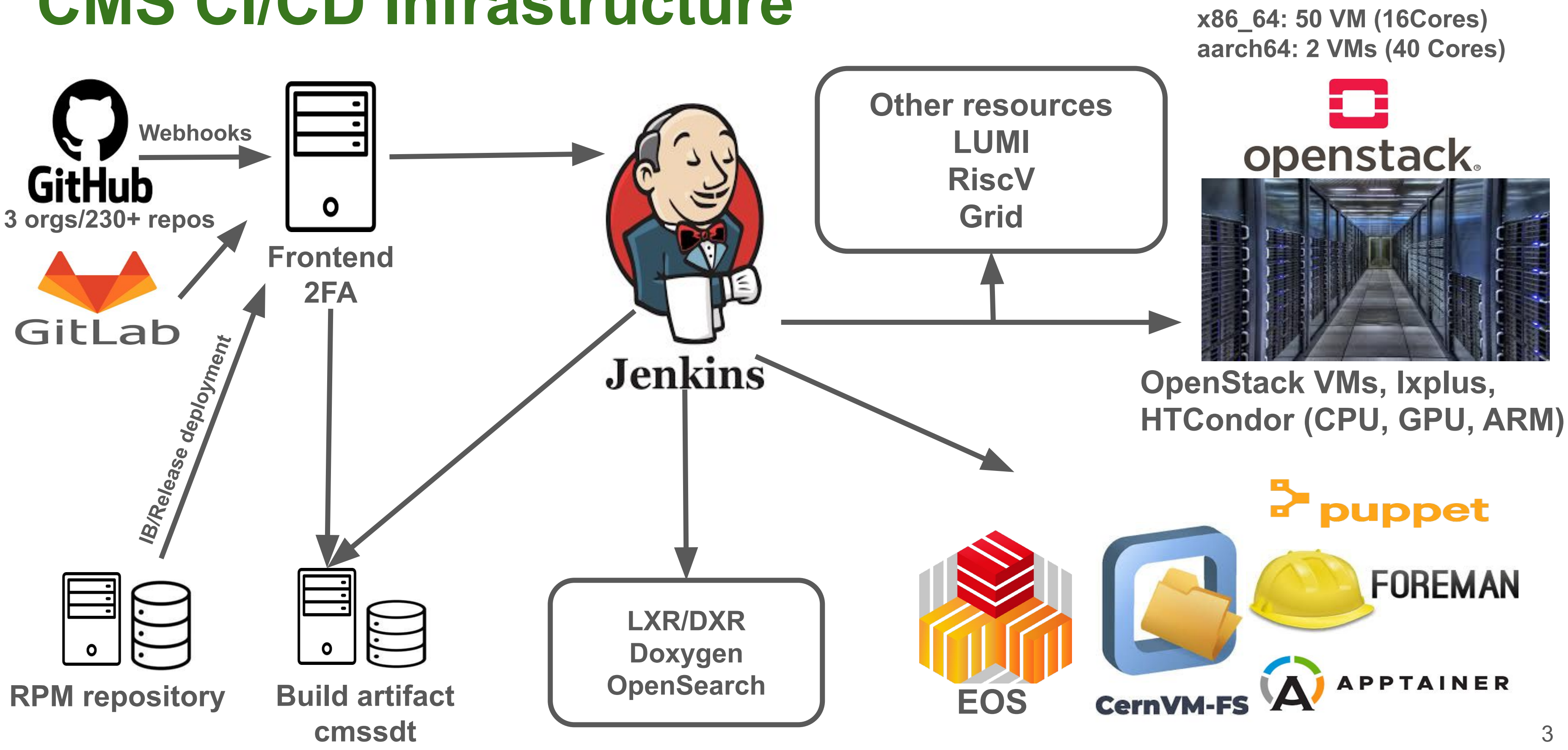
CI/CD System

Joint Experiment Meeting
06/02/2025

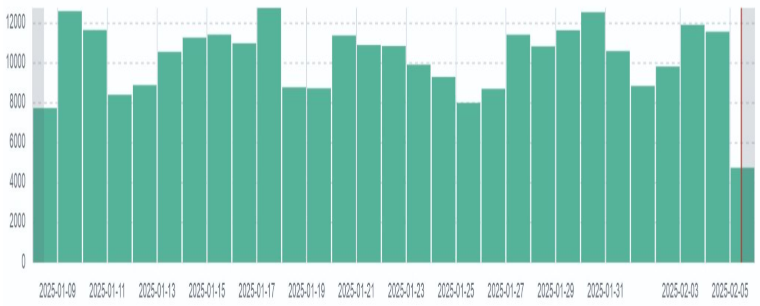
CMS Offline Software: CMSSW

- Hosted on github and has a large actively developed code base
  - Over 8M lines of code
    - 4+M C/C++, 1.6M python, 250K Fortran, 2.2M XML Geometry/data
  - Source code is organized in 1300 Packages
    - Each packages can build one public shared lib and multiple plugins/executables
- 650+ externals are needed at build/runtime
  - ROOT, Geant4, Tensorflow, PyTorch, ONNXRuntime, Cuda, ROCm, GCC, LLVM …
  - All externals are built and distributed in form of RPMs along with CMSSW releases
- 13 Open release cycles and 24 supported architectures
  - OS: slc6 - el9
  - Computer architectures: x86_64, aarch64, riscv64, recently dropped ppc64le support
  - Compilers: GCC 4.7 - GCC 14.2

CMS CI/CD Infrastructure



CMS’ Automation Server



- Jenkins is our automation server since 2013
  - 16 Cores/32GB OpenStack based VM
  - Accessible to CMS members via 2FA enabled front-end server
  - **Runs over 10K jobs/day with <1% failure rate**
    - A build is mark failed only if there are
      - Infrastructure issues (filesystem, network, github, VM misbehaving etc.)
        - Most of these failures are fixed after automatic retries
      - Bugs/Errors in the job itself
    - Failure in actual tests (Unit tests, Release validation, etc.) do not mark build as failed
      - Such failures are reported somewhere else: Github issues, PRs, IB dashboard etc.
- Nearly all projects are based on cms-bot scripts
  - All projects are based on freestyle general purpose Jenkins jobs
  - Job workflow is controlled by chaining Jenkins Projects (Upstream -> Downstream)

CMS Jenkins

```
hg_vocmssdt::jenkins::utils::setup_jenkins {'/build':
-     jenkins_version => '2.479.2',
+     jenkins_version => '2.479.3',
```

- ❖ Fully configured via Puppet
  - ➢ OpenStack/HTCondor based build agents are automatically added/removed
  - ➢ Updating Jenkins version is as simple as pushing the change to puppet
- ❖ Jenkins' configuration (jobs, nodes , secrets, plugins etc.) is backed up
  - ➢ Backup runs every 5 mins: Couple of minutes for full backup
- ❖ Jenkins backup is used during major migrations or testing new versions
  - ➢ Moving servers due to H/W upgrade or move to new OS
  - ➢ Testing Jenkins new major versions e.g. moving to Java 11 or Java17
- ❖ 99.99% uptime and maintenance requires <5% of time
  - ➢ Jenkins/Plugins versions update: 2-3 mins downtime
  - ➢ Major migrations/upgrades: 5-10 mins downtime

Why Jenkins?

- Clear winner when we migrated away from cron jobs in 2013
- Now a days one has a choice of Github actions, Gitlab CI/CD, Circle CI etc.
  - It really depends what are project’s requirements and complexity
  - Github Actions, Gitlab CI/CD are mostly good for organizations with few repositories
    - Sharing self hosted resources between different organizations is not possible unless one buys Enterprise account
      - [Github recommends to only use self-hosted runner for private repositories]
    - One needs to install and run technology specific software
      - Not all architectures are supported e.g. Github Actions self hosted runner software is not available for ppc64le and RiscV architectures OR CentOS 7 and earlier OS
- Number of jobs we run are way over Github actions Free plan limits

Jenkins’ Freestyle Projects

- Makes Jenkins’ management really easy
  - We know which plugins are used and where
    - Makes updating plugins versions much easier
      - In case of breaking changes we only need to test selected projects
    - Helps cleaning up unused/unmaintained plugins
  - Automation for retry of failed jobs is much easier
  - Only a small number plugins are required
    - ~90 plugins are installed in our production Jenkins instance
- Easy to find which build nodes/agents are in use
  - Each agent’s build history shows what jobs were run on it
  - This is not possible with pipeline projects unless one install extra plugins

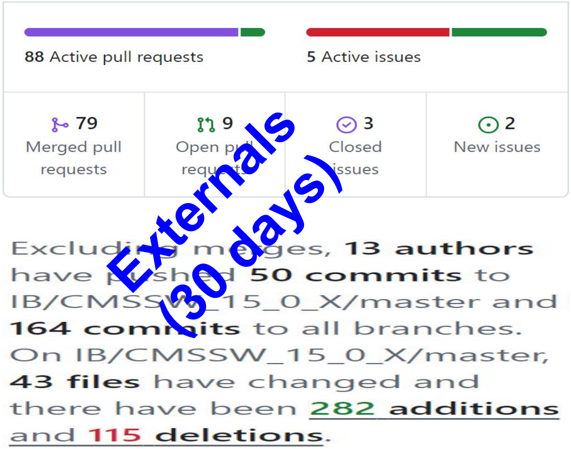
CMS Continuous Integration

- Based on github webhooks and freestyle Jenkins projects
  - Single Jenkins job to receive webhooks from all of our repositories (230+ from 3 github Orgs)
- We do not use Github Pull request plugin
  - Uses polling and consumes a lot of GH API calls especially when one has to manage hundreds of repositories
  - Security issues and also unmaintained
- No pipeline or multi-configuration projects
  - Require a lot dependent plugins
  - Make Jenkins management/updates really hard
  - Not easy to automate the retry of failed stages

```
Jenkinsfile (Declarative Pipeline)
pipeline {
  agent any
  stages {
    stage('Build') {
      steps { echo 'Building..'}
    }
    stage('Test') {
      steps { echo 'Testing..' }
    }
    stage('Deploy') {
      steps {echo 'Deploying....' }
    }
  }
}
```

CMS Continuous Integration…

- CMS Offline software monthly gets over 350 Pull Requests
  - Automated PR testing system allows us to integrate over 90% of these
- On avg. we run 20 Pull Request testing jobs/day
  - Each jobs can take 2-4 hours (depending on the change)
    - Run small subset of release validation tests (~220 out of 4.8K)
    - Unit tests
    - HLT test
    - Reconstruction/DQM comparisons
    - Static analysis
    - etc.



CMS Pull Request results

AddOn Tests	See Logs
Comparison failed	See failed
Comparison with the baseline	See Comparison Results
Compilation log	See Log
Compilation warnings summary	See Logs
DQM bin by bin comparison	See results
External Build Logs	See Log
External Build Stats	See Log
External tool conf	See log
Externals Checks	See log
Externals compilation	See Log
HLT Trigger comparison	See results
Matrix Tests Outputs	See Logs
Package dependency	See Log
Unit Tests	See Log ⚠ Errors Found
User Test materialBudgetTrackerPlots	See Log
max memory used comparison	See results

cmsbuild commented 12 hours ago — Member ···

+1

Size: This PR adds an extra 20KB to repository
Summary: https://cmssdt.cern.ch/SDT/jenkins-artifacts/pull-request-integration/PR-f5f3ab/44199/summary.html
COMMIT: `a1b23cd`
CMSSW: CMSSW_15_0_X_2025-02-04-1100/el8_amd64_gcc12
Additional Tests: GPU
User test area: For local testing, you can use `/cvmfs/cms-ci.cern.ch/week1/cms-sw/cmssw/47263/44199/install.sh` to create a dev area with all the needed externals and cmssw changes.

Comparison Summary

Summary:

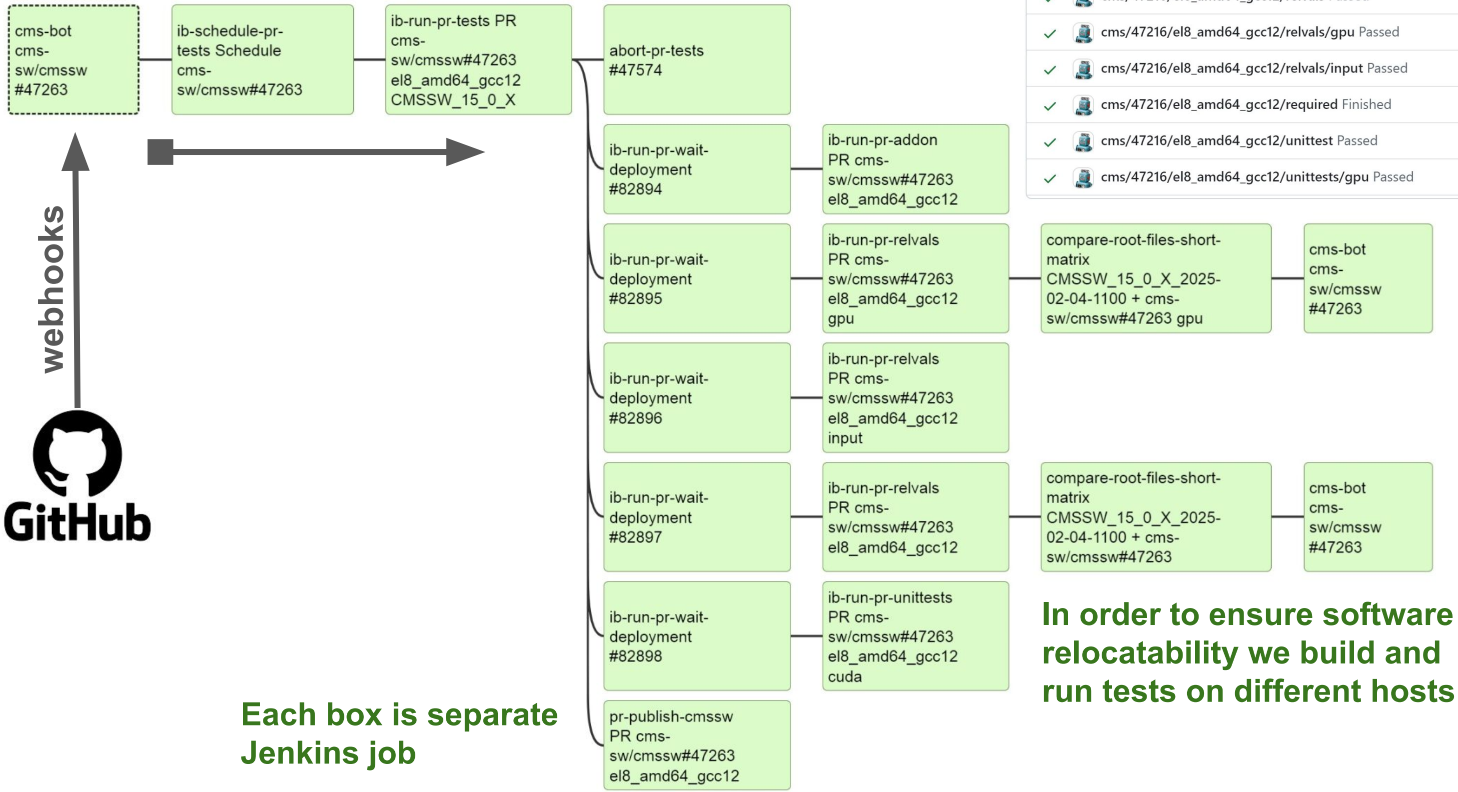
- You potentially added 78 lines to the logs
- Reco comparison results: 4 differences found in the comparisons
- DQMHistoTests: Total files compared: 50
- DQMHistoTests: Total histograms compared: 4016938

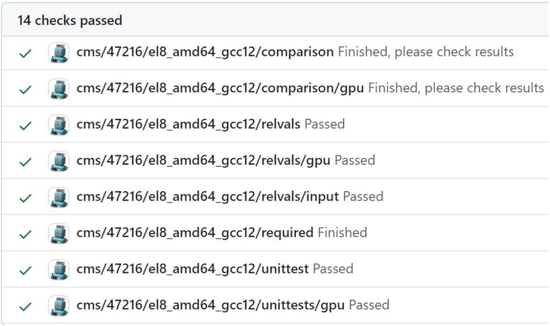
GPU Comparison Summary

Summary:

- No significant changes to the logs found
- Reco comparison results: 0 differences found in the comparisons
- DQMHistoTests: Total files compared: 7

Pull Request testing workflow





Each box is separate Jenkins job

In order to ensure software relocatability we build and run tests on different hosts

CMSSW Integration Build (IB)

- IBs are build every 12 hours
  - Force build full IB on Sunday for all open release cycles
  - Build full IB if externals packages are changed
  - Build patch/incremental IB if only CMSSW code is changed
  - Although Jenkins triggers 80+ IBs/day but on avg. 30 IBs/day are build
- IB are built for all Open release cycles/architectures
  - X86_64 IBs/releases of latest release cycle (15.0.X) are built for two micro-archs
    - x86-64-v3 (default) , x86-64-v2
    - Dynamically set runtime env based on the host
- Over 6K+ tests run for every IB
  - All tests run in parallel on different hosts. Majority of tests results are available with in couple of hours
    - Some long running tests can take 6+ hours
  - 40+ hours worth of tests are run for production architectures
- IBs are available on CVMFS for two weeks

CMS Integration Build

DEFAULT | UBSAN_X | TF_X | ROOT6_X | ROOT634_X | ROCM_X | RNTUPLE_X | PY312_X | NOOFAST_X | NONLTO_X | MULTIARCHS_X | MULTIARCHSV4_X | GPU_X | GEANT4_X | G4VECGEOM_X | DEVEL_X | DBG_X | CXXMODULE_X | CUDART_X | CLANG_X | ASAN_X

OS: el8 el9 CPU: aarch64 amd64 Compiler: gcc12 gcc13 gcc14

IB flavors/OS/archs/compilers for development release cycle

CMSSW_15_0_X_2025-02-03-2300

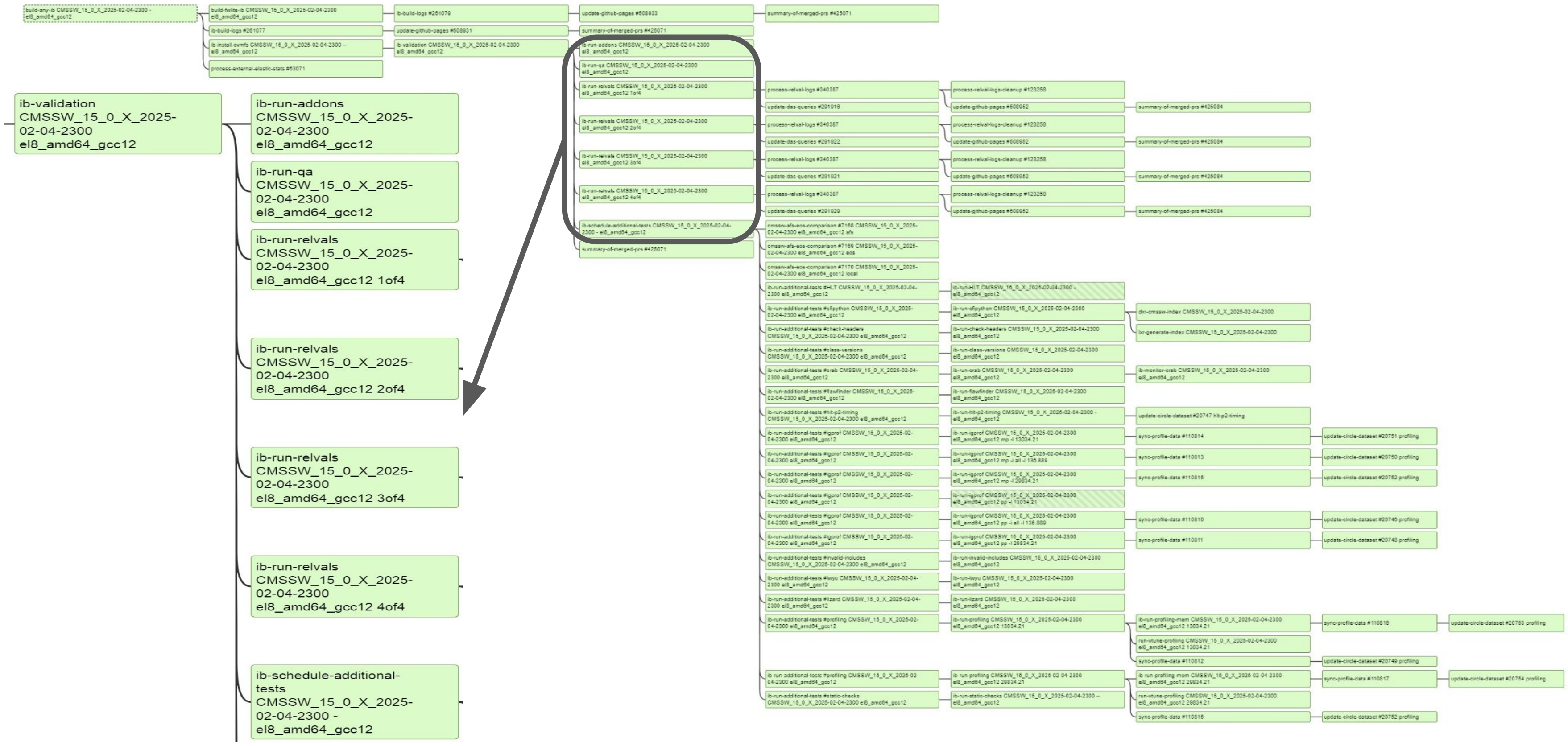
IB Tag | HLT | HLT Phase2 Timing | Class Versions | Clang Analyzer | Crab | Code complexity metrics | Flaw finder | IgProf | Vtune | Resources Piecharts | RECO event loop | RECO GPU module timings | Static Analyzer | SA thread unsafe modules | SA thread unsafe EventSetup products | Header consistency | FWLite

IB dashboard

	DEFAULT				
	el8 aarch64 gcc12 Full Build	el9 aarch64 gcc12 Full Build	el9 amd64 gcc12 Full Build	el8 amd64 gcc12 Full Build	el8 amd64 gcc13 Full Build
Builds	✓	✓	✓	✓	✓
Unit Tests	2	✓	✓	✓	✓
RelVals	4*	1*	4856	4856	4856
Other Tests	✓	✓	✓	✓	✓
Q/A	🔍	🔍	🔍	🔍	🔍

	UBSAN_X	ROOT6_X	RNTUPLE_X	NONLTO_X	MULTIARCHS_X	MULTIARCHSV4_X	GPU_X	DEVEL_X	CLANG_X	ASAN_X
	el8 amd64 gcc12 Full Build	el8 amd64 gcc12 Full Build	el8 amd64 gcc12 Full Build	el[illegible] amd64 gcc12 Full Build	el8 amd64 gcc12 Full Build	el9 amd64 gcc12 Full Build	el8 amd64 gcc12 Full Build	el8 amd64 gcc12 Full Build	el8 amd64 gcc12 Full Build	el8 amd64 gcc12 Full Build
Builds	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Unit Tests	?	✓	✓	✓	✓	✓	✓	1	1	?
RelVals		48[illegible]	4856	4788	50*	1970*	2	4856*	4856	11*
Other Tests	✓	✓	✓	✓	✓	✓		✓	✓	✓
Q/A	🔍	🔍	🔍	🔍	🔍	🔍	🔍	🔍	🔍	🔍

Integration Build workflow



CMS Offline Software

Build System (SCRAM)

SCRAM

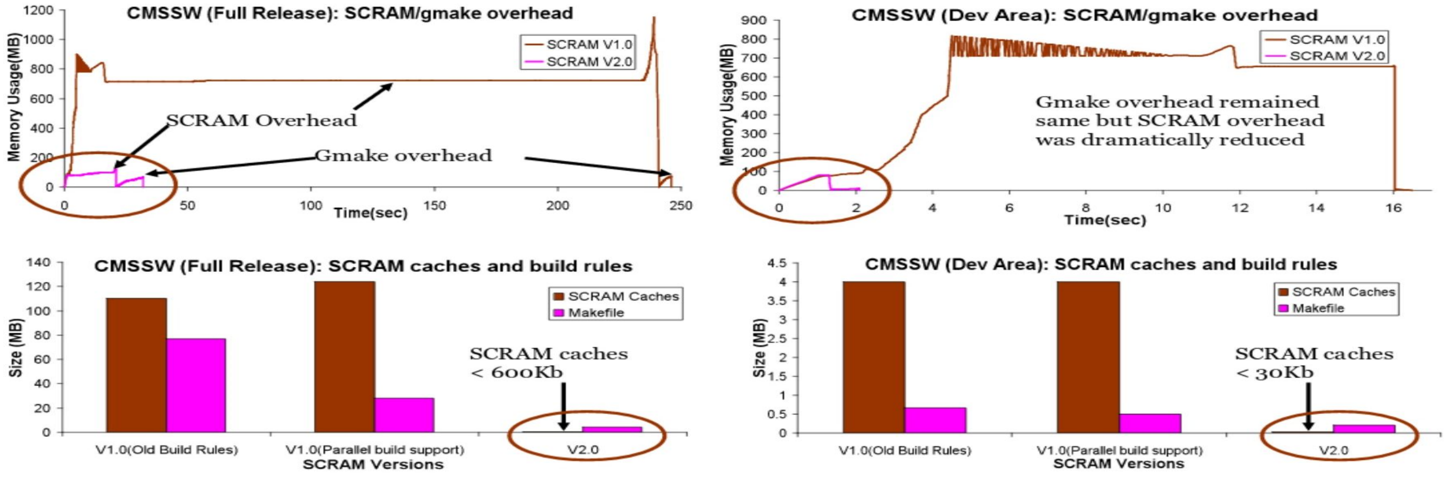
- ❖ Software Configuration Release and and Management tool
- ❖ Developed and used by CMS since 1998
  - ➢ In early/mid 2000’s, LCG projects like CORAL, POOL and SEAL also used it
- ❖ Just like CMake, it is build system configuration generator
  - ➢ SCRAM uses MAKE as backend
  - ➢ Converts user defined requirements from BuildFiles into MAKE rules
- ❖ As a project configuration manager its helps
  - ➢ Finding and using existing IBs/releases
  - ➢ Setup runtime environment
    - ■ Dynamically select the best env at runtime
  - ➢ Apply/Control site/project specific rules

```
<use name="tbb"/>
<use name="DataFormats/Common"/>
<use name="DataFormats/Provenance"/>
<use name="FWCore/ParameterSet"/>
<use name="FWCore/Utilities"/>
<export>
  <lib name="1"/>
</export>
```

Build shared library

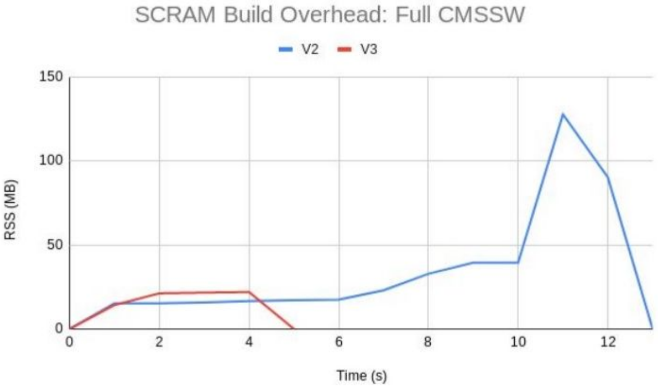
SCRAM: V1 vs V2

- ❖ Major rewrite was done in 2008 to improve its performance
  - ➢ Reduce code size: 35 PERL modules instead of 100+ in V1
  - ➢ Parallel builds support
  - ➢ Improve disk usage: Helped developers to develop on shared file-systems like AFS/EOS



SCRAM V3

❖ In order to reduce PERL dependency, in 2020, V3 was rewritten in PYTHON
  - ➢ Reduce code base: 5.5K instead of 13K
  - ➢ User interface remained same
  - ➢ For better tooling, used json format to store SCRAM's internal caches

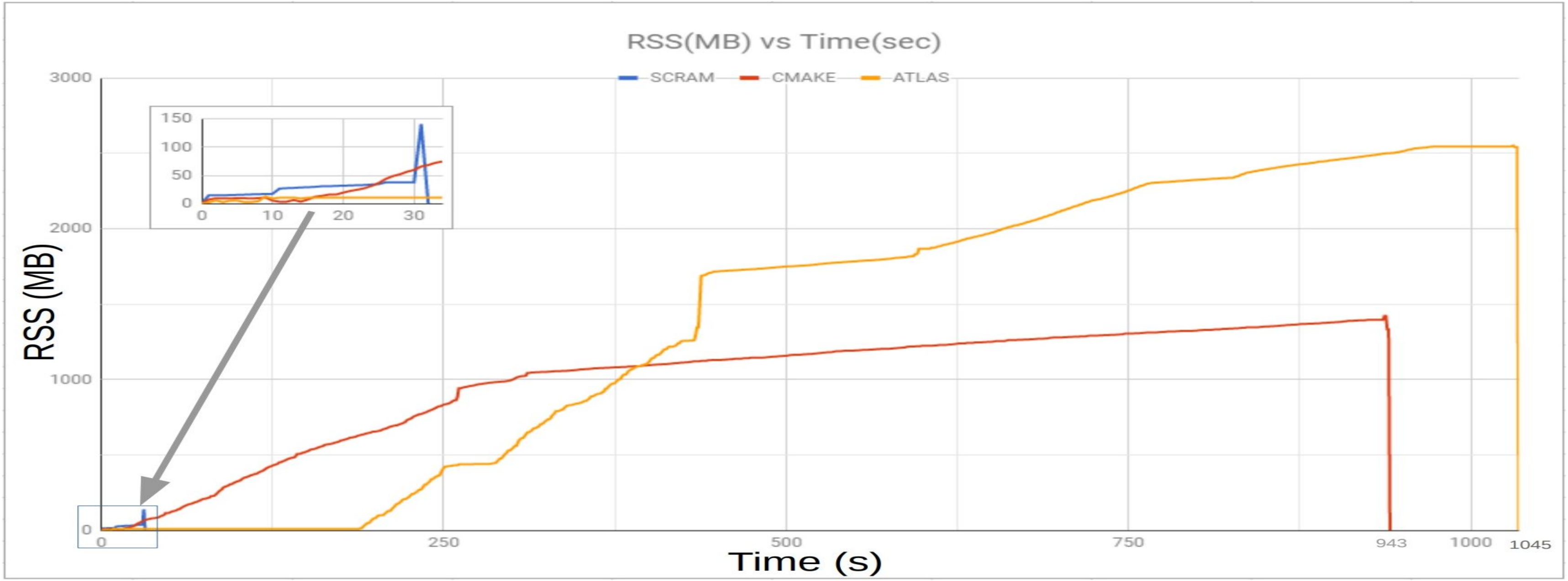


Why SCRAM

- Easy to use
  - Search available IB/releases: ***scram list***
  - Create developer area: ***scram -a el8_amd64_gcc12 project CMSSW_Version***
  - Build: ***scram build -j $(nproc)***
  - Setup runtime environment: ***eval `scram runtime -sh|-csh`***
  - Reset runtime environment: ***eval `scram unset -sh|-csh`***
- In 2018, we evaluated CMake but results were not promising
  - Auto converted BuildFiles to CMakeLists.txt
  - Converted SCRAM's tool-files to CMake's Find<Tool>.cmake
  - CMake configure step was 30+ times slow
    - 30+ times more disk usage
    - Generating a lot of small files per compilation unit
    - Not good for using it on shared file-systems AFS/EOS or Ceph volumes

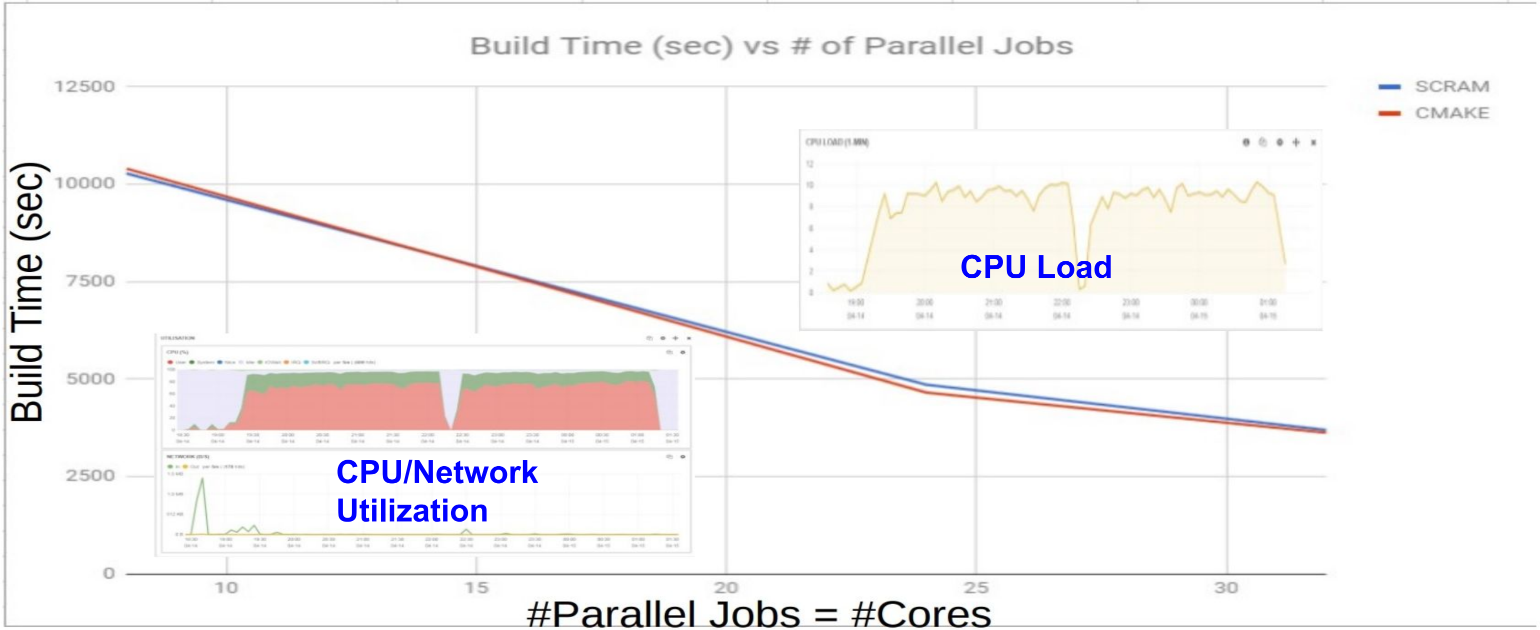
SCRAM(V2) vs CMAKE: Configuration step

	# Objects	# Libraries	# Binaries	#Rootmap PCM	DiskSpace (MB)
SCRAM	14984	2236	671	514	30
CMAKE	14832	2233	664	500	910



SCRAM(V2) vs CMAKE: Build time comparison

	SCRAM(sec)	CMAKE(sec)
08 Cores	10284	10404
24 Cores	4855	4656
32 Cores	3698	3627

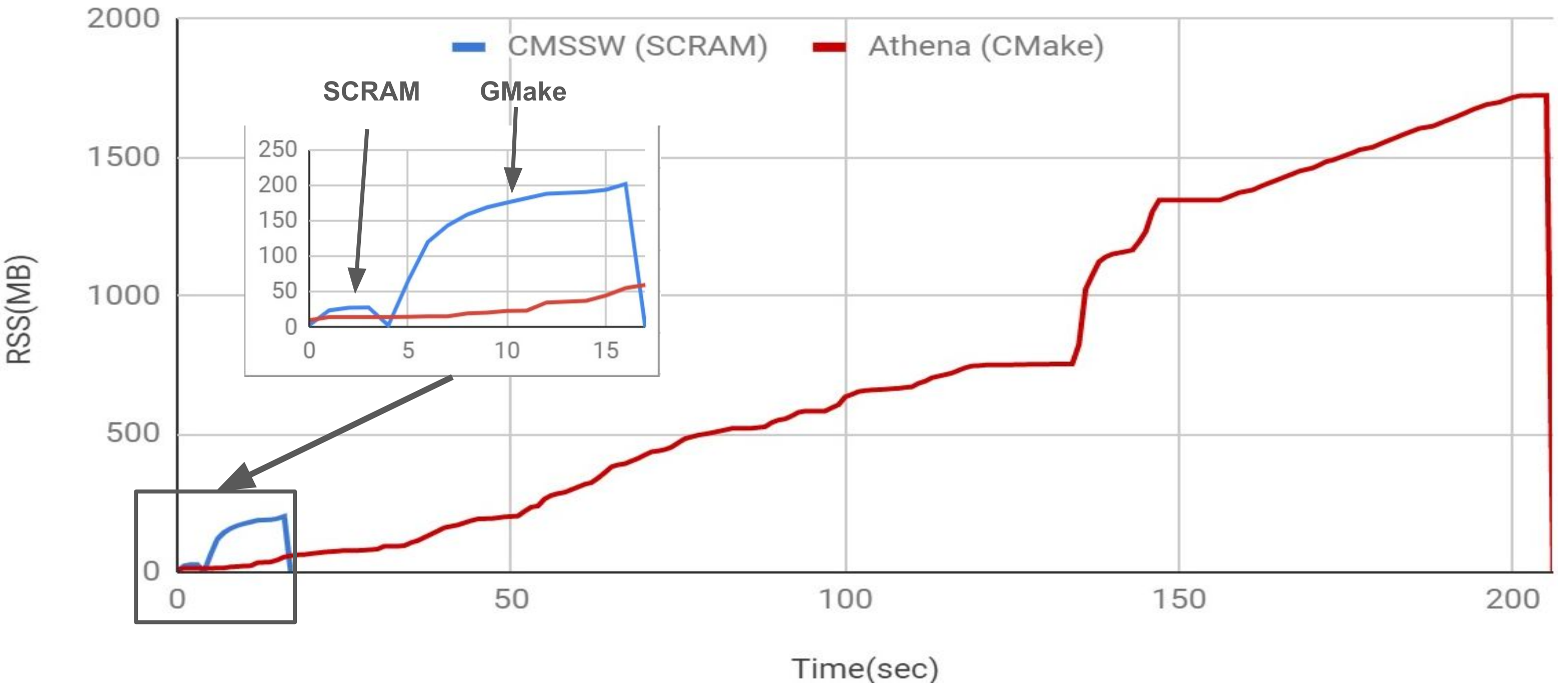


SCRAM(V3) vs CMAKE

- Comparing SCRAM performance with actual similar size CMake based project (ATHENA) shows that SCRAM still out perforces CMAKE
    - For comparison, I use Athena release/25.2.39 and CMSSW 15.0.X IB of 23rd JAN
    - Tests were done on a 16 core Openstack VM on local SSD
    - GCC 13 , AlmaLinux9

	Source Code	Total Targets	Objects files	Binary products	Configure time(s)	Build time(min)	Disk usage)MB)
CMSSW	4.1M	58K	18.5K	3600	4	85	60
ATHENA	3.3M	37K	17.2K	2683	205	150	500

SCRAM(V3) vs CMAKE



Summary

- ❖ CMS has a robust/scalable CI/CD system which has ensured high quality of Integration build and releases
  - ➢ Integration builds has the same quality of major release
  - ➢ Helped us test and integrate latest versions of externals with in a day

- ❖ SCRAM, though over 27 years old, but has not shown any aging
  - ➢ Easy to maintain: over 50% of PYTHON rewrite of V3 was done by a student