Moving CMS to a Container-based Infrastructure

Brian Bockelman

MAGIC Meeting, 4 April 2018

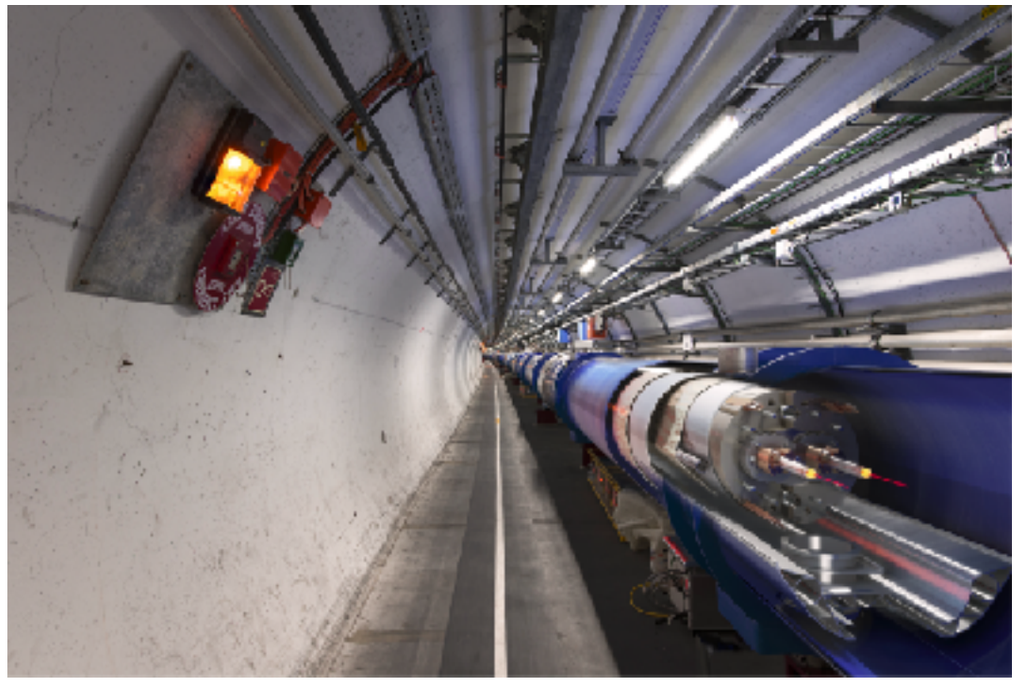
Note:

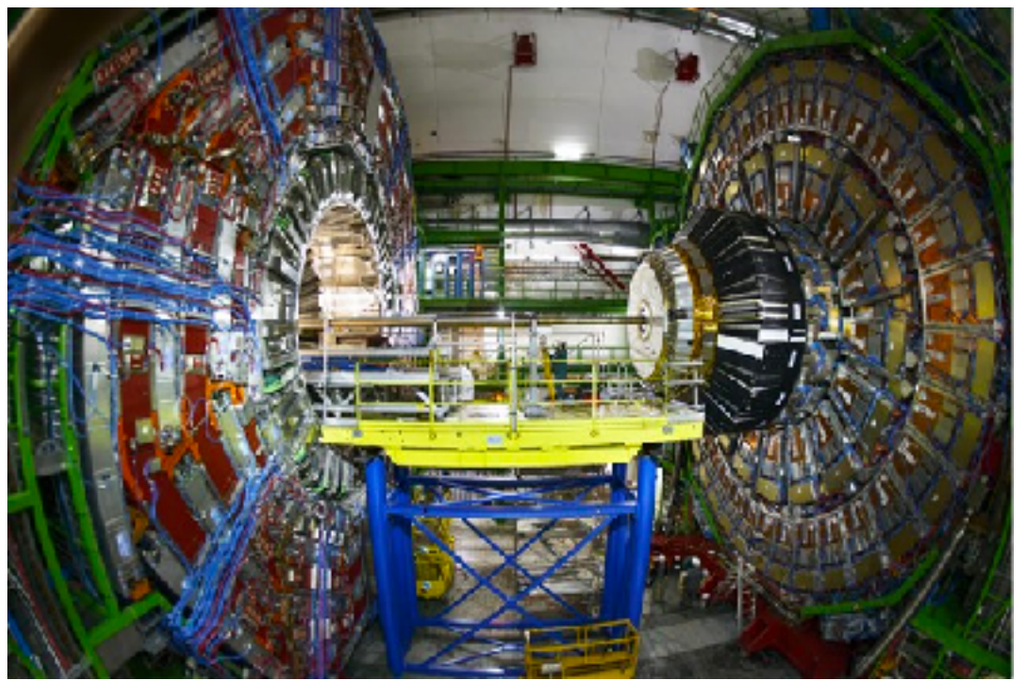
Examining a narrow corner of a broad area.

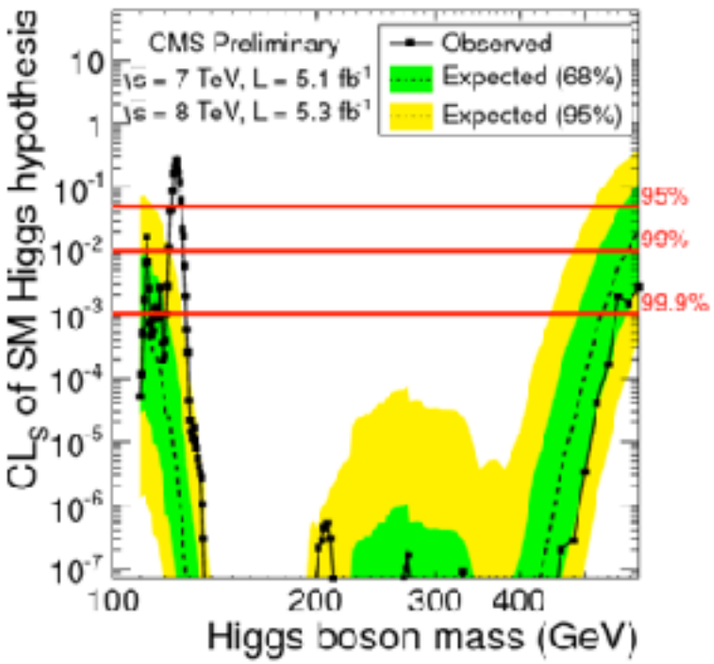
Apologies if your favorite part isn't covered!
We can bring it up in the discussion afterward…

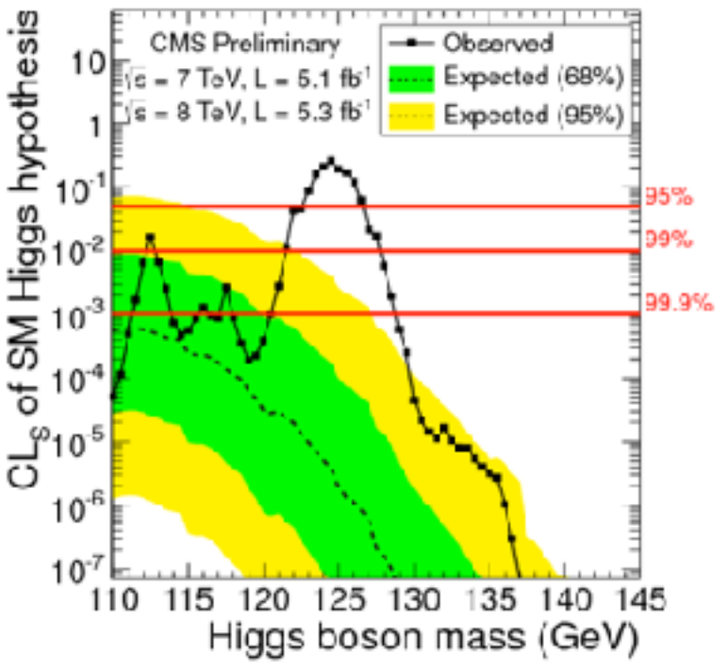
Setting the stage: CMS Science

- CMS is one of the general purpose particle detectors on the Large Hadron Collider (LHC), located outside Geneva, Switzerland.
- When running, proton bunches cross at a rate of 40MHz; this year, we'll have an average of 40 collisions per crossing.
- Provides physicists in the CMS Collaboration (~4,000 individuals) the data necessary to probe fundamental questions about the universe.
    - Much of the discovery process breaks down into a statistics exercise: counting observed collisions of a certain type versus those explainable by known physics.



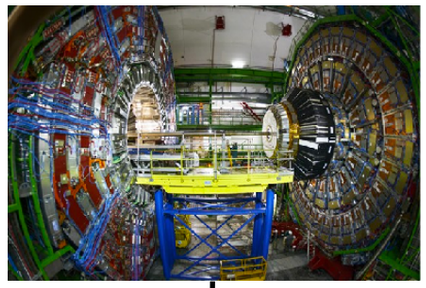






Setting the Stage: CMS Computing

- Perhaps unsurprisingly, one of the most complex physics endeavors results in a multifaceted computing challenge:
  - We need highly-tuned, high-performance computing hardware near the detector to quickly filter out >99.99% of the data resulting from uninteresting events.
  - The remaining ~0.001% events have more potential physics value, enough to be analyzed in depth. The raw readouts must be "reconstructed" into physics objects.
  - Bad news: these ~0.001% of events are still billions of events, requiring hundreds of millions of computing hours to reconstruct.
  - Worse news: for every event kept by the detector, we need to simulate 2 events to understand the background statistics. Simulated events take twice as much CPU as "real" events.
- *But we're not done yet*: the events must be processed by users in order to perform their scientific analyses. We also have to ~yearly reconstruct the entire dataset as our algorithms and calibrations improve.



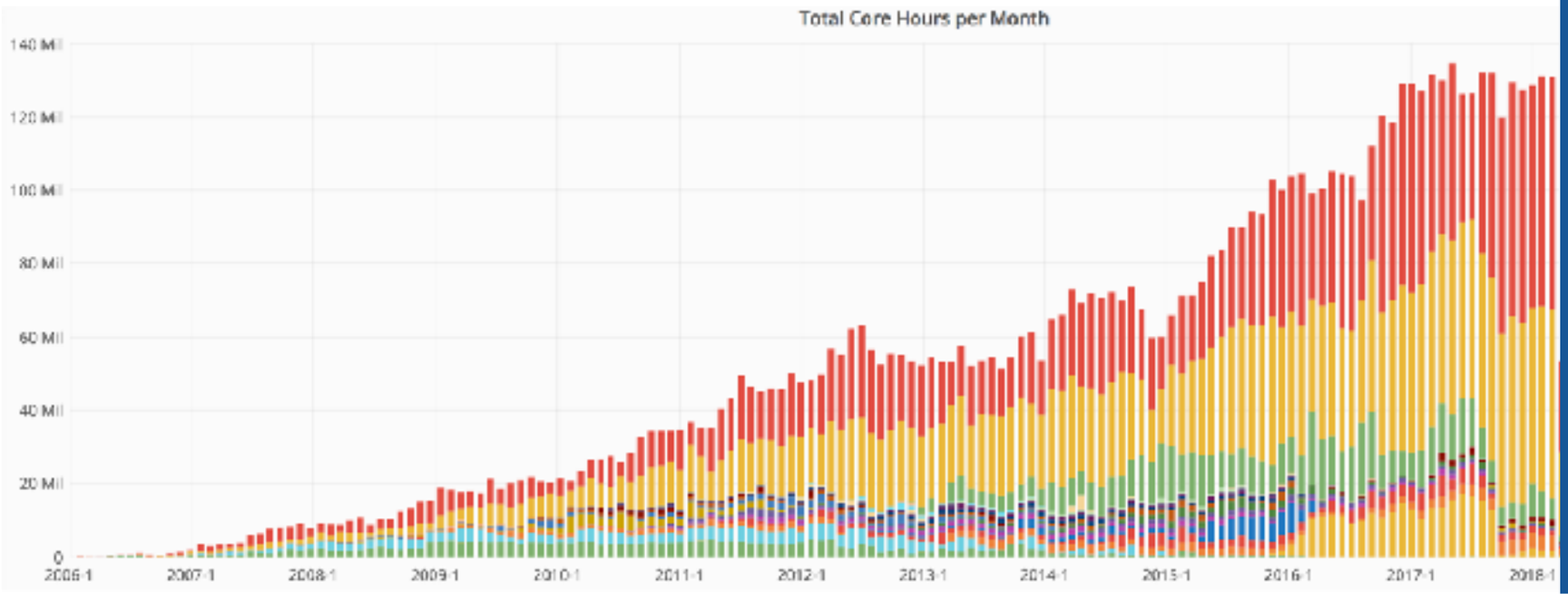
In the end, this is a multi-billion-CPU-hour affair!

Setting the Stage: CMS Computing

- The HEP community has traditionally turned to distributed computing to tackle their computing challenges:
    - Cannot find enough resources at a single site to tackle the entire problem.
    - Each event can be processed independently of the others -> no need for tightly-coupled communication.
    - Allows us to quickly leverage any available resources, regardless of where they sit.
- This is traditionally expressed as a *high throughput computing* problem: “how many events can be processed per month?”, not “how many events per second?”

Distributed High Throughput Computing (DHTC)

- Within the US, OSG provides a common platform and operation services to execute DHTC-style workflows.
  - While ATLAS and CMS dominate the raw hours on OSG, it's a much wider community effort.
- A common platform helps avoid reinvention and allows easier sharing and technology transfer. Singularity is a great example of this.



In the last 24 Hours	
235,000	Jobs
4,814,000	CPU Hours
5,360,000	Transfers
532	TB Transfers
In the last 30 Days	
8,812,000	Jobs
135,992,000	CPU Hours
199,988,000	Transfers
21,878	TB Transfers
In the last 12 Months	
135,380,000	Jobs
1,593,281,000	CPU Hours
2,300,466,000	Transfers
218,000	TB Transfers

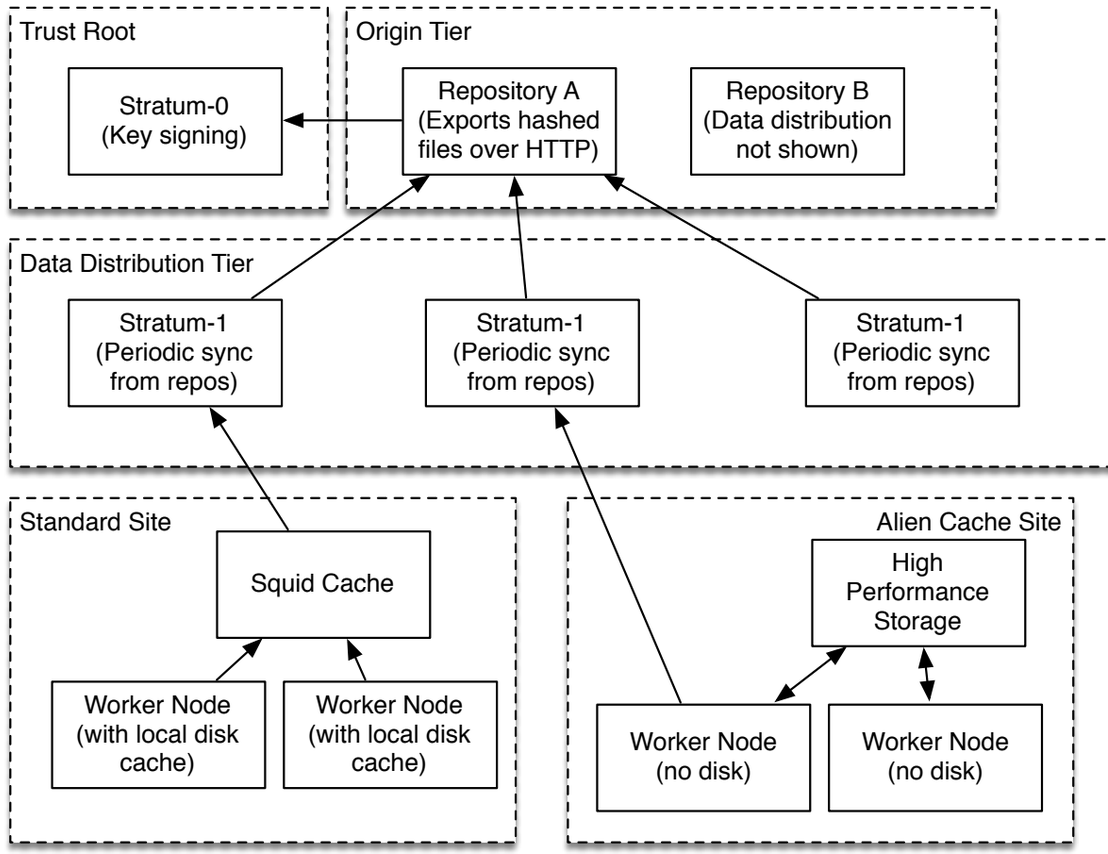
OSG delivered across 125 sites

Challenges: Portable Environments

- The CMS experiment needs a wide variety of software packages to support its activities. Counting “lines of code” is tricky, but millions of lines of code exist for CMS software, building on top of tens of millions LoC of HEP community software.
    - Smaller than a Linux distribution - but similar in complexity.
- We need all this software to function consistently on a wide variety of clusters, each with a subtly different OS environment.
- Solution #0: Install our software at every site by hand.
    - Man-power intensive, nearly impossible to synchronize installs across all sites, impossible to do any sort of integrity checking.

Portable Environments: CVMFS

- The CERN VM File System (CVMFS) is a FUSE-based filesystem that provides a consistent, global read-only shared filesystem.
    - Metadata and data is transported over HTTP; utilizes a hierarchy of HTTP servers and caches to efficiently deliver LHC's software to the worker nodes:
- **Good news**: provides efficient, reliable access to tens of millions of files / 10's of TB of data.
- **Bad news**: it still is a custom filesystem; adoption is limited outside sites traditionally used by the HEP community.
- CVMFS tackles the software distribution problem, but not the problems with uniform OS environments!



Challenge: Isolated Execution

- CMS - and most of HEP - uses batch resources in a “pilot model” or “overlay model” or “resource allocation model”:
    - A “pilot job” is sent to the batch system.
    - On start (“resource allocation”), the pilot validates the environment and starts a connection to a global pool of resources.
    - Pilot downloads and executes the “payload jobs”, which are the actual user’s scientific tasks.
- Sounds strange for batch system admins! Easier to understand if you think of the batch system as a miniature “cloud” and the pilot as a request for a VM.
    - However, *unlike* VMs the pilot is not root but a regular batch system user.
    - **Basic Question**: How do we isolate the pilot process and credentials

Solution approach: Singularity

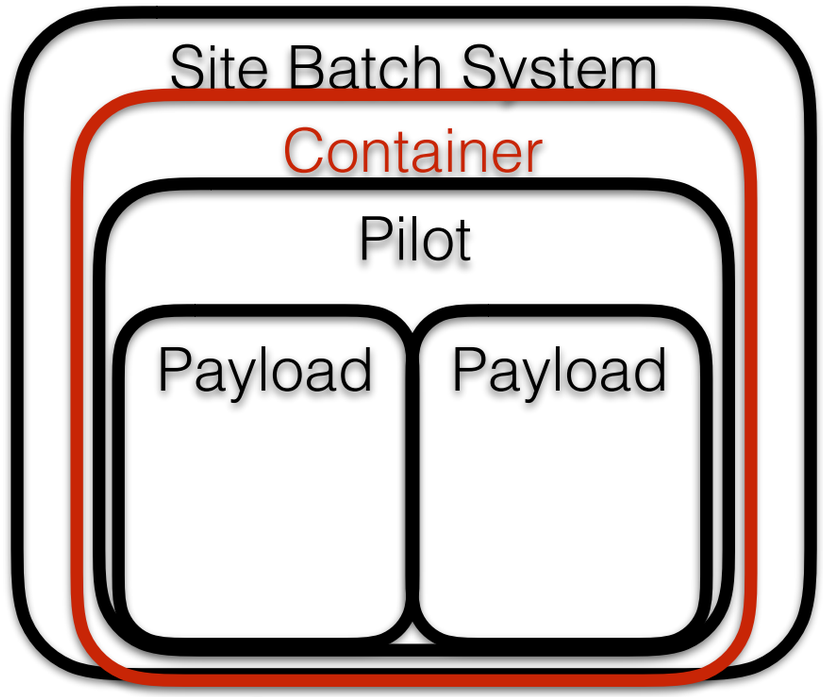
- Singularity is a container solution tailored for the HPC use case.
  - It allows for a portable of OS runtime environments.
  - It can provide isolation needed by CMS.
- Simple isolation: Singularity does not do resource management (i.e., limiting memory use), leaving that to the batch system.
- Operations: No daemons, no UID switching; **no edits to config file needed**. “Install RPM and done.”
- Goal: User has no additional privileges by being inside container. E.g., disables all setuid binaries inside the container.

http://singularity.lbl.gov

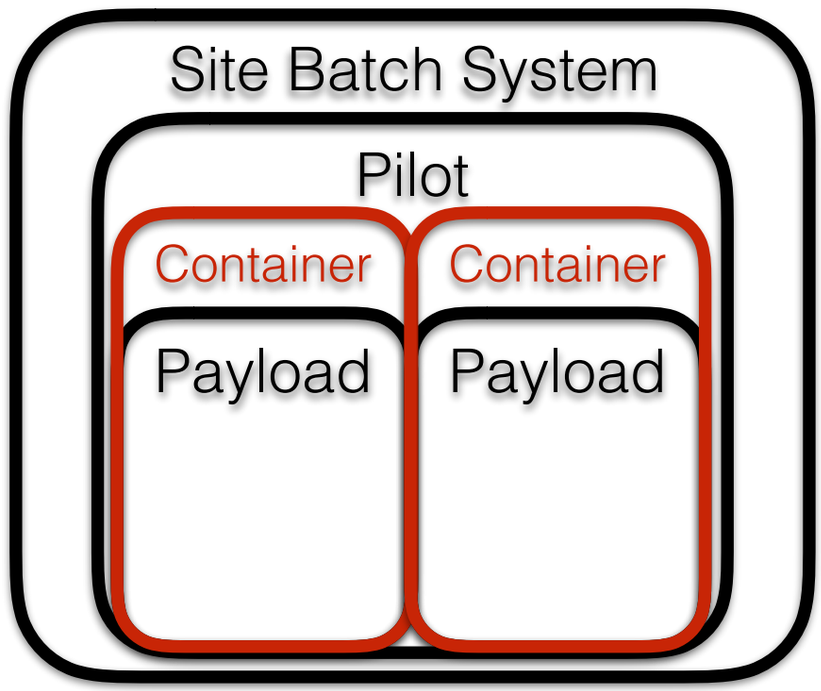
Who is in a container?

- When should we invoke Singularity? Before the pilot or afterward?
- Three options when using containers:
  - A: Batch system starts pilot inside a container.
  - B: Pilot starts each payload inside its own container.
  - C: Combine A and B.
- Option A does not meet our isolation goals. **Option B and C does**.

Option A:



Option B:



View From the Worker Node - Sysadmin

```
slurmstepd: [8295392]
 \_ /bin/bash /var/spool/slurmd/job8295392/slurm_script
     \_ /bin/bash /var/lib/globus/condor-ce/spool/5263/0/cluster4115263.proc0
         \_ /bin/bash /scratch/glide_kmuqIk/main/condor_startup.sh glidein_co
             \_ /scratch/glide_kmuqIk/main/condor/sbin/condor_master -f -pidf
                 \_ condor_procd -A /scratch/glide_kmuqIk/log/procd_address -
                 \_ condor_startd -f
                     \_ condor_starter -f login02.osgconnect.net
                         \_ /util/opt/singularity/2.2.hcc-c0d435a/gcc/4.4.7/l
                             \_ /util/opt/singularity/2.2.hcc-c0d435a/gcc/4.4
                                 \_ /util/opt/singularity/2.2.hcc-c0d435a/gcc
                                     \_ /bin/bash /srv/condor_exec.exe
                                         \_ pegasus-kickstart -n job-wrapper.
                                             \_ /bin/bash ./job-wrapper.sh 10
                                                 \_ /usr/bin/time -f corsika:
                                                     \_ /bin/bash ./execute_c
                                                         \_ ./corsika75000Lin
```

Site Batch System

Pilot

Singularity

User Payload

View From the Worker Node - Pilot Job

Pilot

```
\_ /bin/bash /scratch/glide_kmuqIk/main/condor_startup.sh glidein_co
    \_ /scratch/glide_kmuqIk/main/condor/sbin/condor_master -f -pidf
        \_ condor_procd -A /scratch/glide_kmuqIk/log/procd_address -
        \_ condor_startd -f
            \_ condor_starter -f login02.osgconnect.net
```

Singularity

```
\_ /util/opt/singularity/2.2.hcc-c0d435a/gcc/4.4.7/l
    \_ /util/opt/singularity/2.2.hcc-c0d435a/gcc/4.4
        \_ /util/opt/singularity/2.2.hcc-c0d435a/gcc
```

User Payload

```
\_ /bin/bash /srv/condor_exec.exe
    \_ pegasus-kickstart -n job-wrapper.
        \_ /bin/bash ./job-wrapper.sh 10
            \_ /usr/bin/time -f corsika:
                \_ /bin/bash ./execute_c
                    \_ ./corsika75000Lin
```

View From the Worker Node - User job

Inside the container, the user's executable has no ability to interact with either the site batch system or the pilot job (or, importantly, its files).

User Payload

```
\_ /bin/bash /srv/condor_exec.exe
    \_ pegasus-kickstart -n job-wrapper.
        \_ /bin/bash ./job-wrapper.sh 10
            \_ /usr/bin/time -f corsika:
                \_ /bin/bash ./execute_c
                    \_ ./corsika75000Lin
```

Singularity and CMS

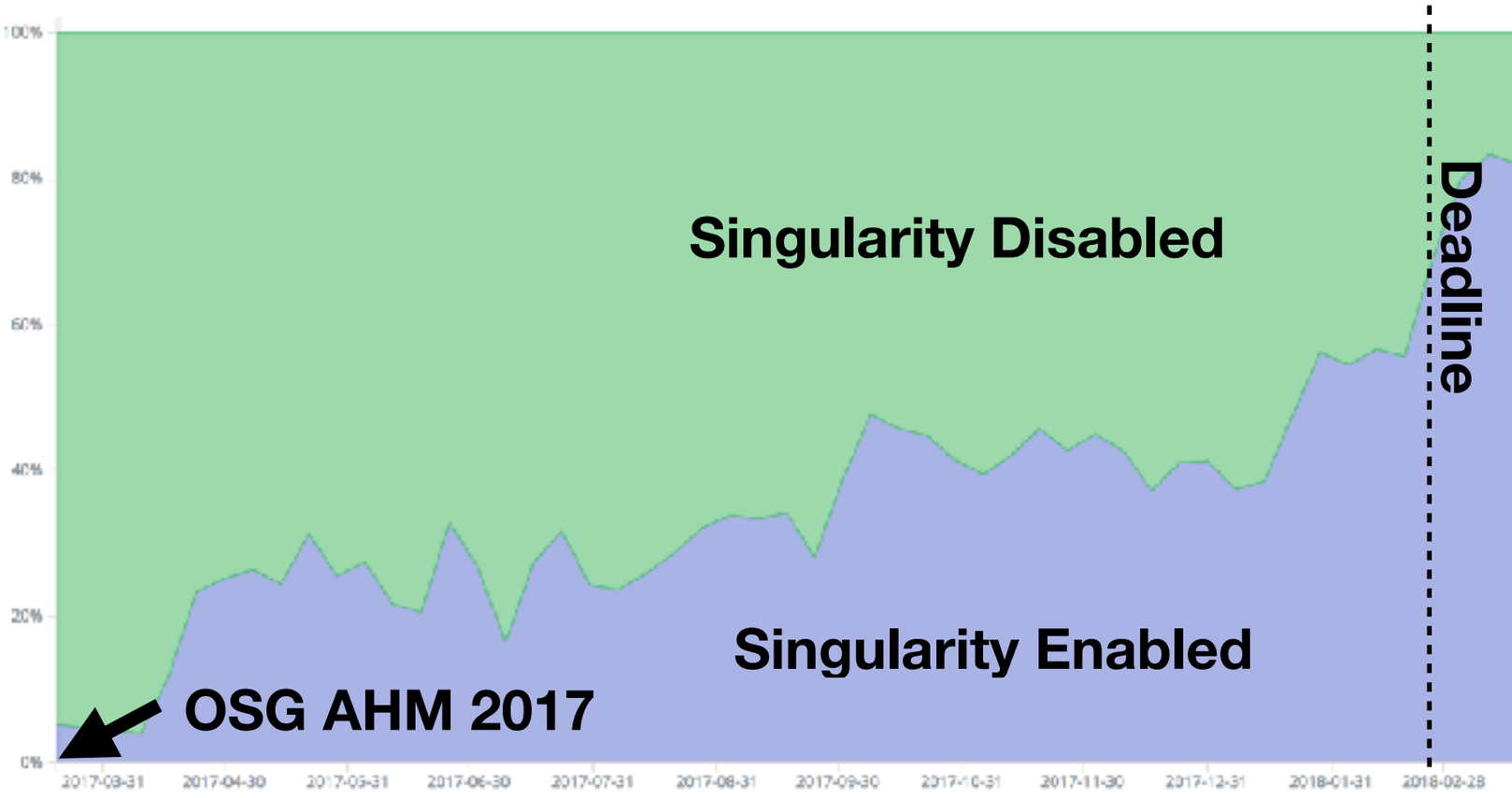
- While we've wanted to leverage containers for years (HTCondor / container integration predates Docker!), none fit into our environment as the execution is happening as an **unprivileged user inside a batch system** on a variety of Linux distributions found in research computing.
    - Other solutions required much newer kernel features, root-level privilege, or integrated poorly with batch systems.
- Immediately saw the potential of Singularity for our community.
    - It only had minor features missing. Started contributing to the project summer 2016; contributions to enable our community peaked in 2017.
    - By version 2.2 in early 2017, we were able to run our workflows inside Singularity.

Singularity Adoption

- In early CY2017, OSG developed the ability to run Singularity-based jobs inside HTCondor:
    - Singularity invoked by “job wrapper” that transforms Unix environment and calculates the correct image and internal mount points.
    - The images are distributed utilizing CVMFS as an “unpacked” directory. Allow us to benefit from the same distribution network, deduplication, and partial download capabilities as CVMFS.
- Only minor changes needed to translate OSG approach to CMS: mostly around locating the container and CMS configuration files.
- By our March 2017 USCMS meeting, one site (Nebraska) had switched production jobs to run inside Singularity.

Singularity Adoption

- Starting April 2017, any CMS site could opt-in into container use.
    - Slowly rolled out across interested sites; mostly OSG initially.
- Multiple months of debugging the “long tail” of tricky environment issues.
- In November 2017, CMS decided to base its 2018 data-taking release on RHEL7 and require the use of Singularity.
- In January 2017, enabled the use of unprivileged Singularity for sites with newer kernels.
- March 2017 was the internal deadline for sites to enable Singularity.
- 90% adoption hit by end of March.



As of April 2018,

126 million

containers launched. Up to 1M / day!

Pitfalls

- **Shared filesystems**: When data is mounted as a shared filesystem on the worker node, it can be tricky to expose this correctly within a container
    - Traditionally Unix filesystem access control is UID-centric, but here all users have the same UID.
    - Do we need “containers for filesystems”?
- **Broad range of Linux kernels**: Singularity supports “ancient” kernel versions, some of which are quite buggy with respect to container features. We’ve had to integrate quite a few workarounds…
- **Validating runtime environment**: How do we determine whether Singularity is setup and working? Container image is correct?
    - Ad-hoc process, mostly based on failure modes discovered by experience.
    - Handling failures becomes tricky: new failure modes can cause tens of thousands of failed jobs before being understood.
    - It’s difficult/impossible to answer “can I spawn a new process?” in Linux: **containers are an order magnitude worse!**

Lessons Learned

- Transition was surprisingly fast: **containers solve significant issues for our community and sites were happy to oblige.**

- We benefited greatly from already tightly controlling the user environment. Very few assumptions were made about what was available at the site - particularly with respect to shared filesystems - making containerization simpler.

- Container developers are enthusiastic, wonderful, and productive — but not superhumans!

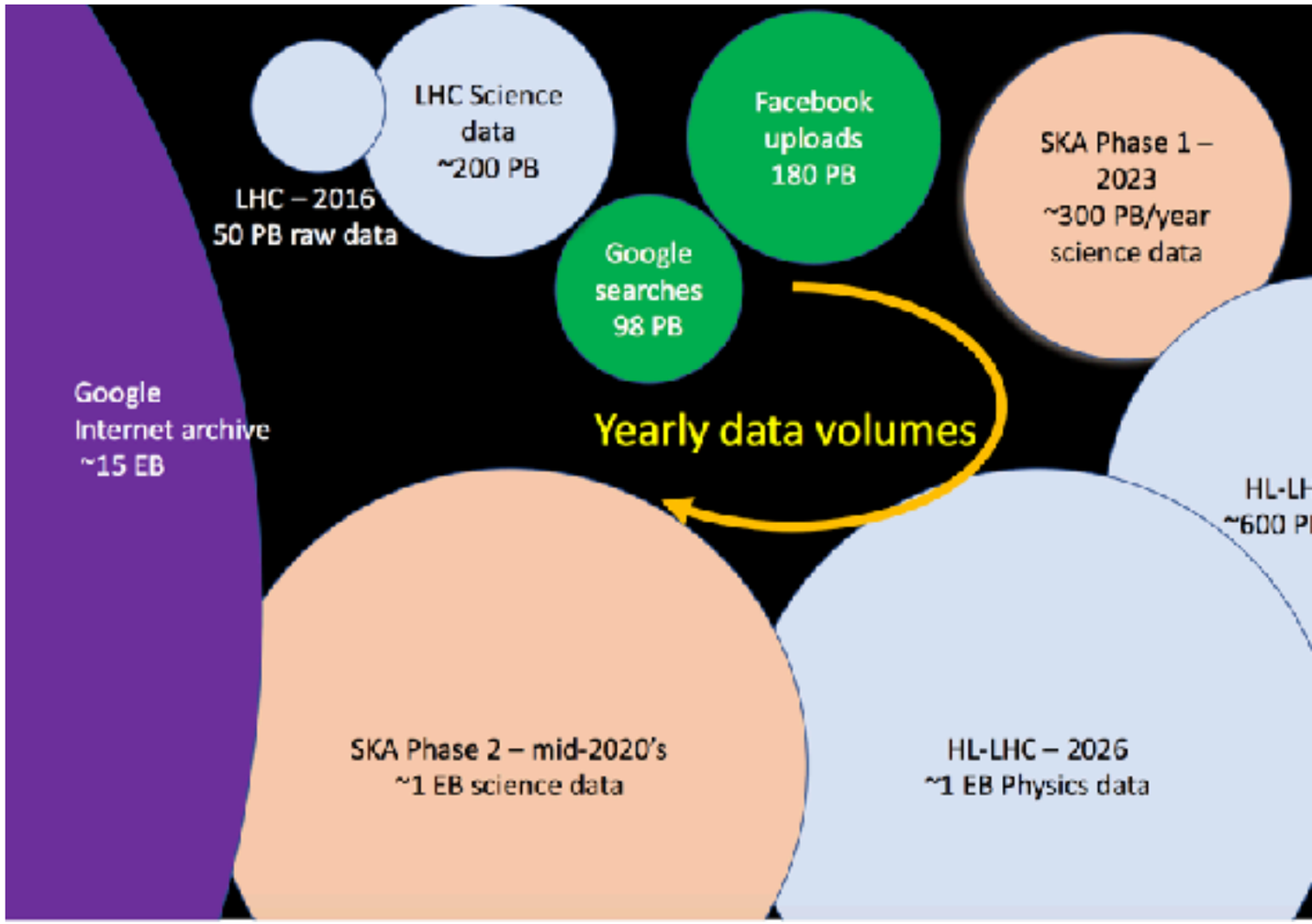
    - Sometimes things break; we must understand the various points of failure and be ready for it at the higher layer.

Looking forward: Going all-in on Containers

- CMS remains a strange user of Singularity: we care about **portable OS environment and isolation**, but had a pre-existing domain-specific solution for **portable applications**.
    - The domain-specific solution (CVMFS) works *wonderfully* on resources that commonly support HEP, but limit our ability to utilize a wider range of resources.
- Investigating the ability to move our software stack into the container itself.
    - Size remains a problem as each release is 10GB and there may be 50 in-use releases.
    - May require us to manage N containers instead of today's single container.

Looking forward: Tackling the HL-LHC era

- If nothing is done, the High-Luminosity LHC (HL-LHC) in the mid-2020's will cause a two-order-magnitude increase in computing challenges.
    - The community is in the process of rallying together to "bend the resource curve" back into our budgets.
    - One strategy will be to better utilize a wider diversity of resources beyond traditional HEP resources.
- Containers provide a powerful tool in this direction: while we've converted to using them at a large scale, **we still have significant challenges / opportunities remaining!**



"Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Networking and Information Technology Research and Development Program."

The Networking and Information Technology Research and Development (NITRD) Program

Mailing Address: NCO/NITRD, 2415 Eisenhower Avenue, Alexandria, VA 22314

Physical Address: 490 L'Enfant Plaza SW, Suite 8001, Washington, DC 20024, USA Tel: 202-459-9674, Fax: 202-459-9673, Email: nco@nitrd.gov, Website: https://www.nitrd.gov