Configuring and Managing Multiple Shasta Systems

Best Practices Developed During the Perlmutter Deployment

NERSC

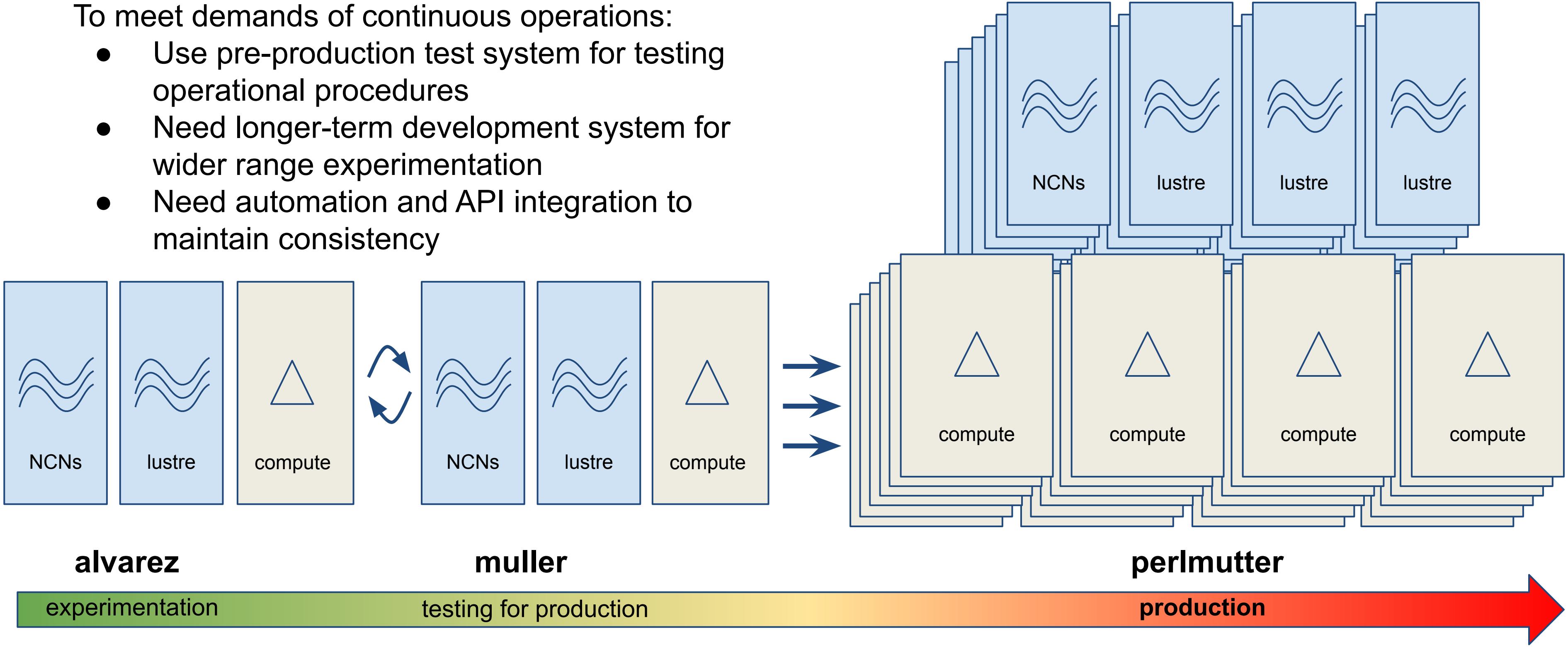
2022/05/04

Ershaad Basheer - NERSC
James Botts - NERSC
David Fox - NERSC
Aditi Gaur - NERSC
Doug Jacobsen - NERSC
Harold Longley - HPE
Eric Roman - NERSC
Chris Samuel - NERSC
John Stile - NERSC

HPE Cray-EX Systems at NERSC

To meet demands of continuous operations:

- Use pre-production test system for testing operational procedures
- Need longer-term development system for wider range experimentation
- Need automation and API integration to maintain consistency



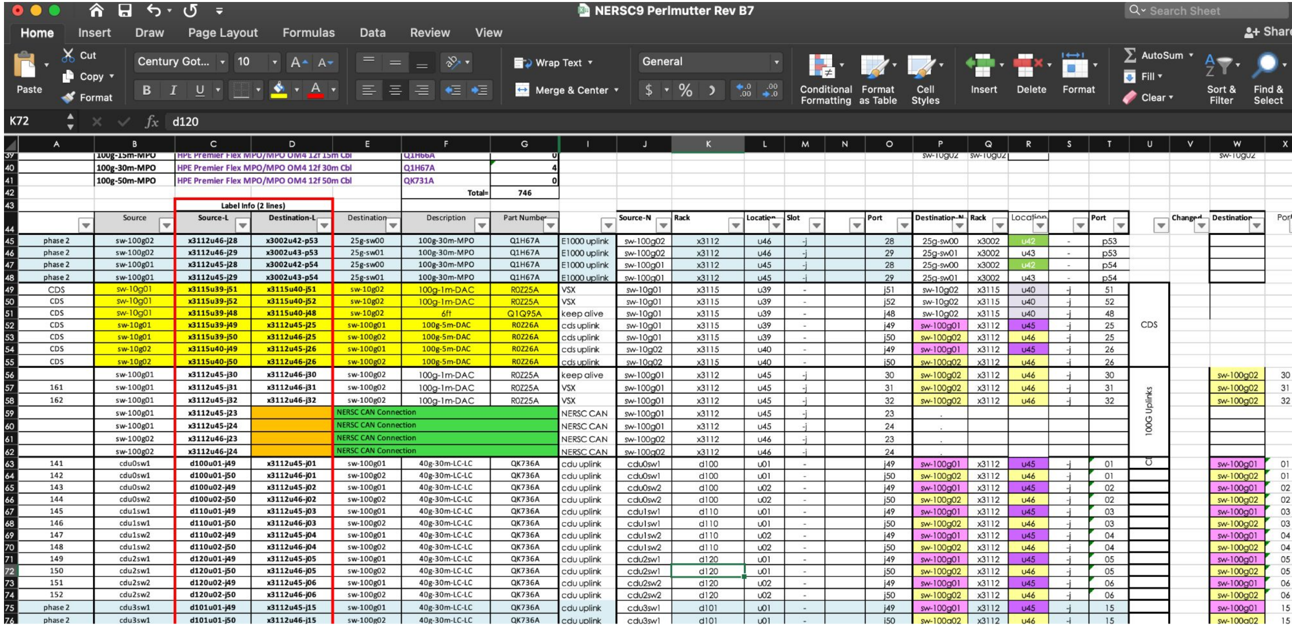
Don't forget perlmutter phase 2

Key Challenges for Deployment

- Hardware integration
  - SHCD (Shasta Hardware Configuration Document)
  - Validating physical cabling
  - Node testing
- Configuration Management (multiple systems!)
  - System configuration (gitea/CFS/IMS Recipes/BOS/BSS)
  - Secrets
  - HSN configuration
  - RPMs
  - Containers/Helm charts/Loftsman manifests for site-provided services
  - HPE-provided software and System Upgrades
- System Stabilization
  - DVS-Over-HSN
  - DVS/CPS Worker Node Separation

Hardware Integration: SHCD

- Must ensure that any all hardware changes are first in the SHCD
- Be sure the hardware configuration will support future modes of operation.
  - NERSC wants hsn0 interfaces on one set of switches and hsn1 on another in river racks to support system resiliency
- If the SHCD is wrong, nothing else on the system can be right



Best Practice: Get to know your SHCD, find ways to track and validate changes with your hardware support
Missing functionality: Ability to validate SHCD against documented CSM requirements

Hardware Integration: Validate Physical Cabling

Confirm that wiring in the system matches the SHCD

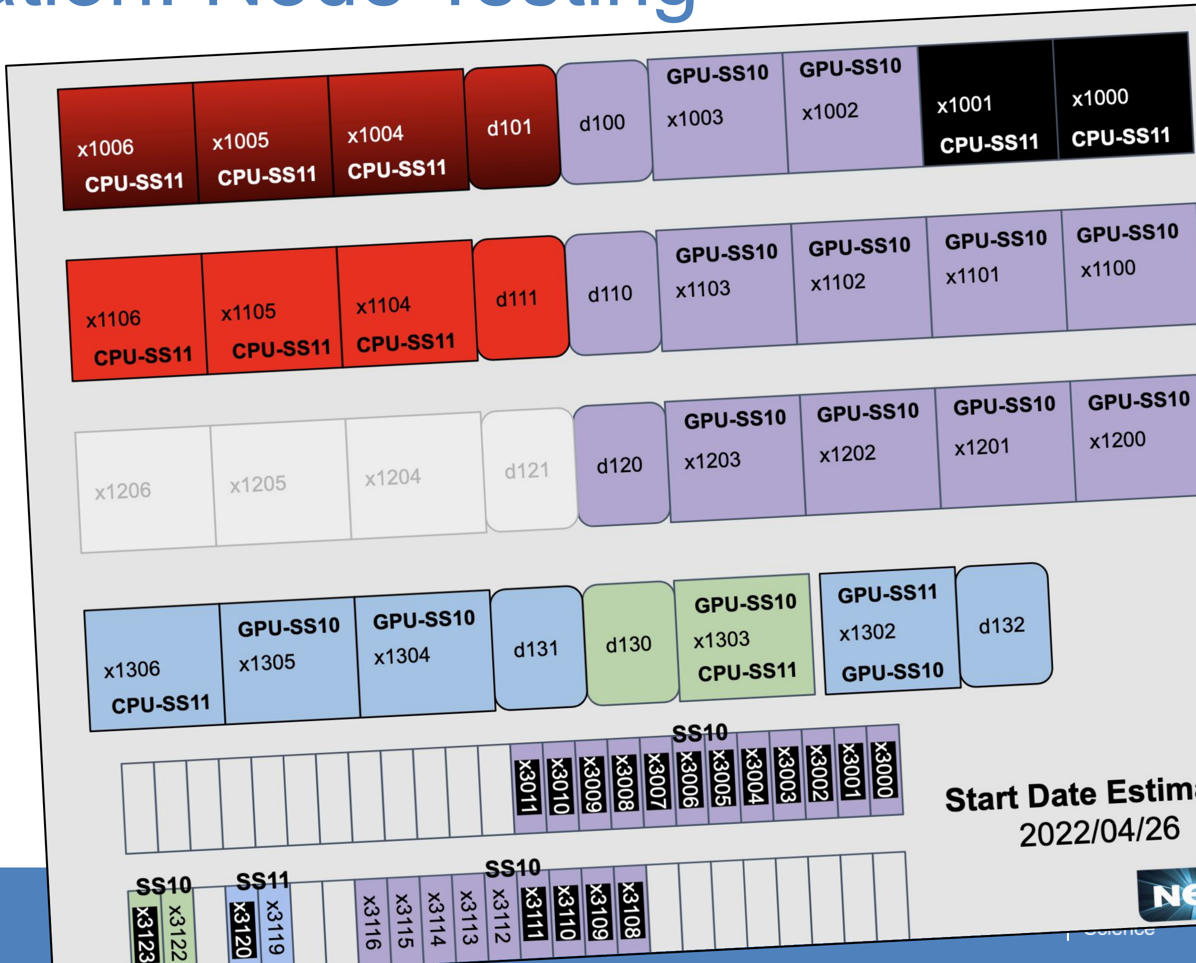
- For HSN, `cable_inventory` and LLDP plus analysis scripts (NERSC custom script shown) can help confirm things are where they belong

```
=== ANALYSIS ===
Incorrect island:  ['x1003c3r7j15', 'x3115c0r30j17', 'x1003c3r7j15', 'x3115c0r31j17']
Incorrect island:  ['x1201c1r1j15', 'x3000c0r44j19', 'x1201c1r1j15', 'x3000c0r45j19']
Incorrect island:  {'x3004c0r46j15', 'x3114c0r31j16', 'x3114c0r31j15', 'x3114c0r30j15',
'x3004c0r45j15'}
Incorrect island:  {'x3008c0r47j30', 'x3009c0r44j25', 'x3009c0r44j30', 'x3008c0r44j30'}
Incorrect island:  {'x3010c0r45j30', 'x3010c0r46j30', 'x3009c0r47j2', 'x3009c0r47j1'}
Incorrect island:  {'x3108c0r46j28', 'x3108c0r45j28', 'x3109c0r45j3', 'x3109c0r45j2'}
Incorrect island:  {'x3114c0r30j4', 'x3114c0r30j3', 'x3113c0r30j25', 'x3113c0r31j25'}
Incorrect island:  {'x3114c0r31j3', 'x3113c0r31j28', 'x3114c0r31j4', 'x3113c0r30j28'}
===========
Group of Incorrect connections:  ['x1003c3r7j15', 'x3115c0r30j17', 'x1003c3r7j15', 'x3115c0
Incorrect link ('x1003c3r7j15', 'x3115c0r30j17') related to missing links:
     ('x1003c3r7j15', 'x3115c0r31j17')
    Check if x3115c0r30j17 should move to x3115c0r31j17

===========
Group of Incorrect connections:  ['x1201c1r1j15', 'x3000c0r44j19', 'x1201c1r1j15', 'x3000c0
Incorrect link ('x1201c1r1j15', 'x3000c0r44j19') related to missing links:
     ('x1201c1r1j15', 'x3000c0r45j19')
    Check if x3000c0r44j19 should move to x3000c0r45j19

===========
Group of Incorrect connections:  {'x3004c0r46j15', 'x3114c0r31j16', 'x3114c0r31j15', 'x3114
'x3004c0r45j15'}
Incorrect link ('x3004c0r45j15', 'x3114c0r31j15') related to missing links:
     ('x3004c0r45j15', 'x3114c0r30j15')
     ('x3004c0r46j15', 'x3114c0r31j15')
```

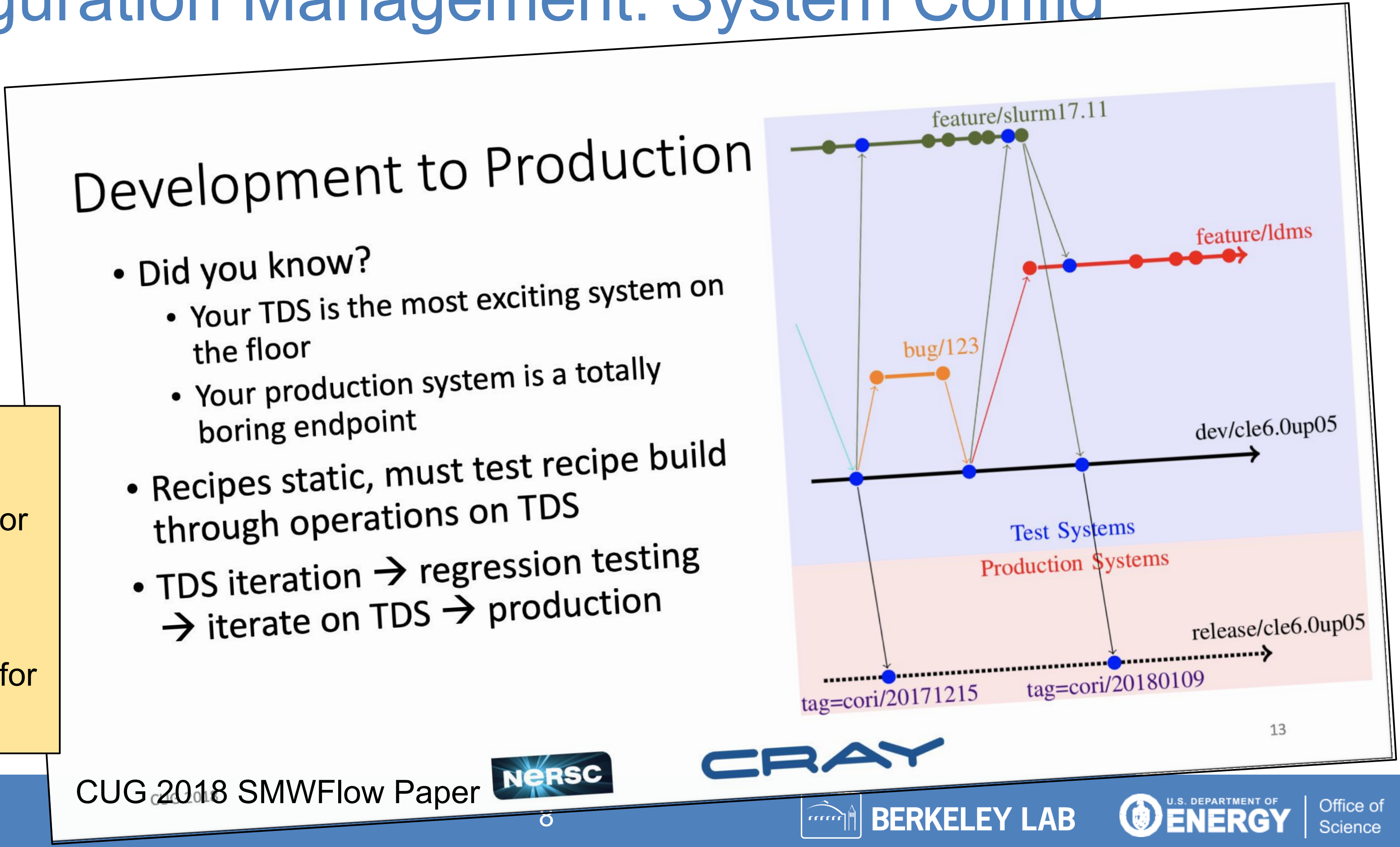
Hardware Integration: Node Testing

Stabilizing both the computes nodes and the network simultaneously is exceedingly difficult.

For the phase 2 system, NERSC is leveraging an HPCM test system to validate each row *before* it gets added to perlmutter.



Configuration Management: System Config

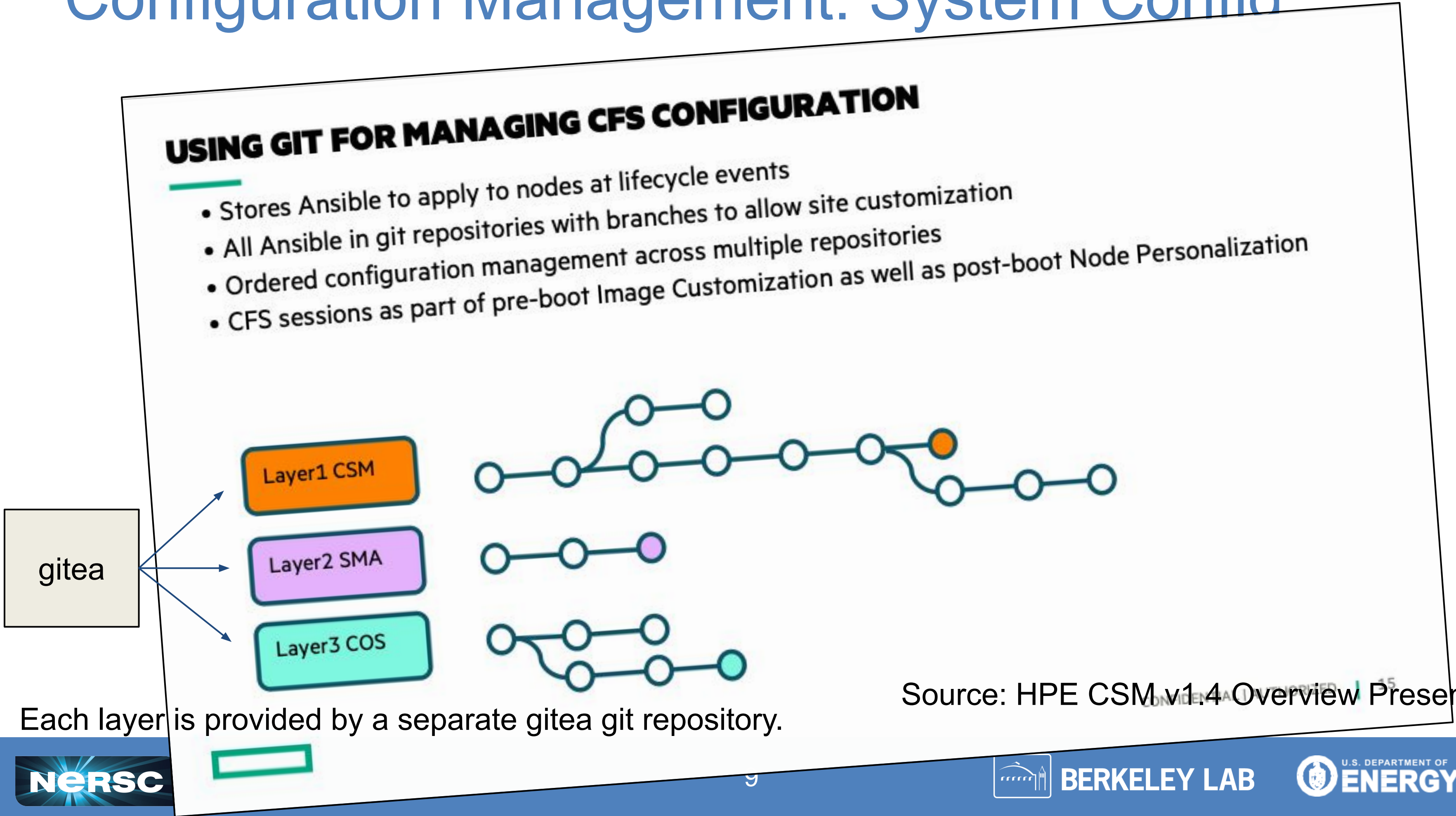


CUG 2018 SMWFlow Paper

Working multi-system, many-contributor administration model for XC.

How to do this for Cray EX?

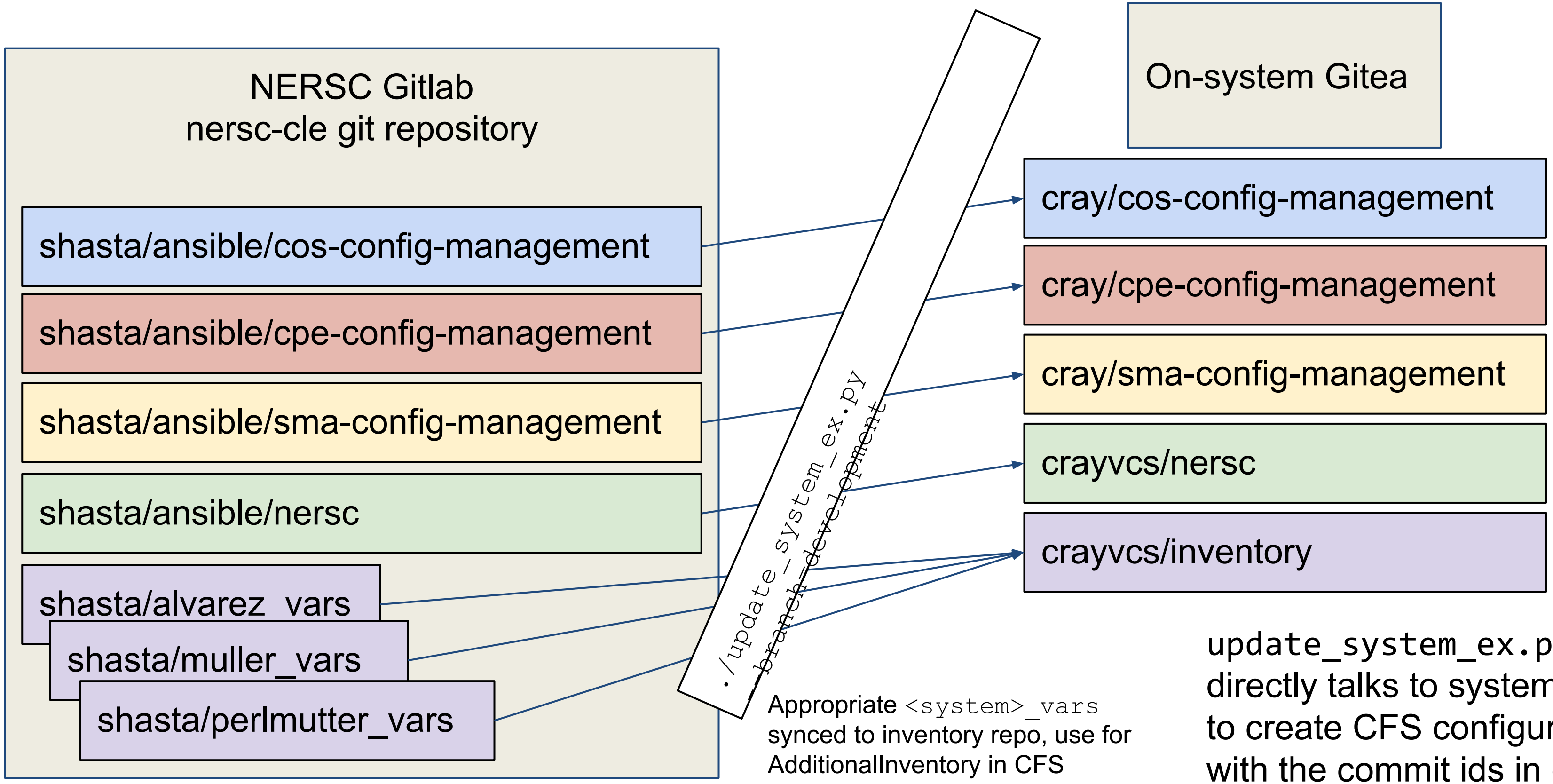
Configuration Management: System Config



Each layer is provided by a separate gitea git repository.

Source: HPE CSM v1.4 Overview Presentation

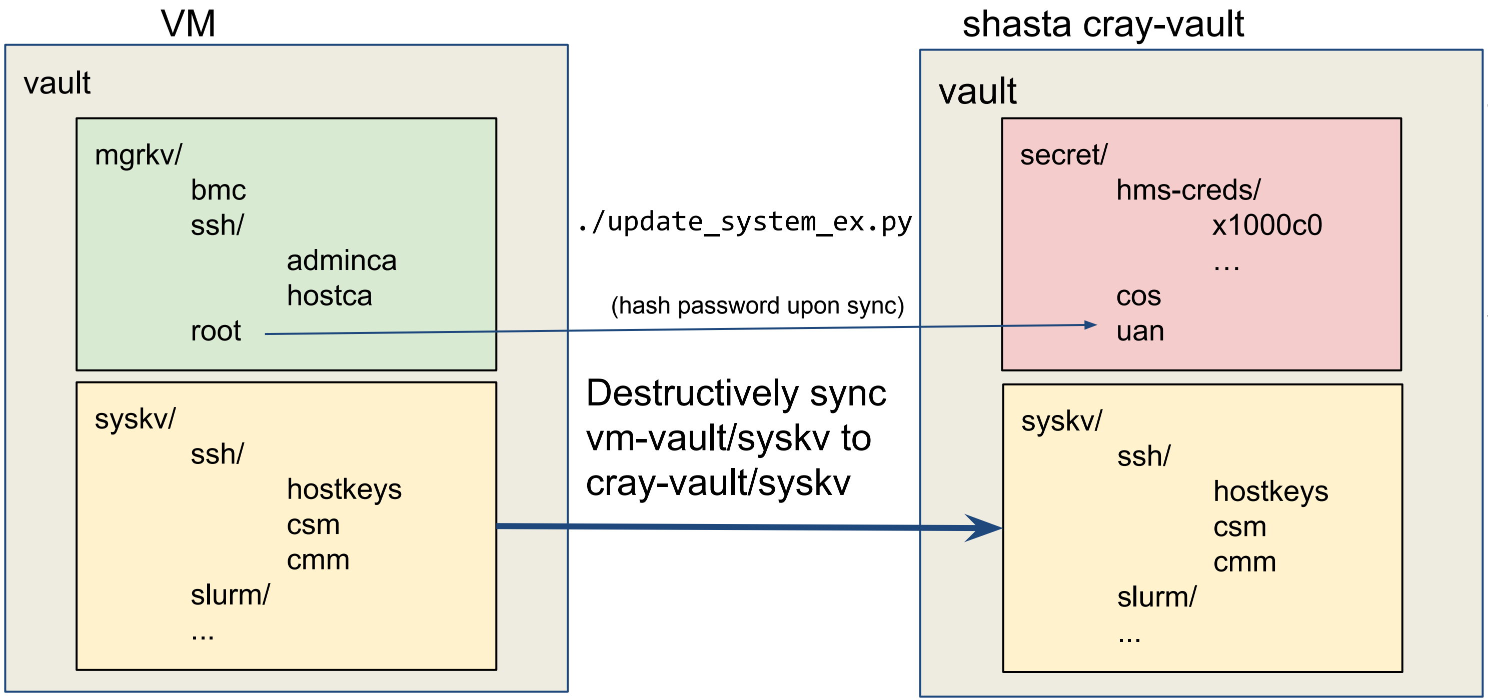
Configuration Management: System Config



Appropriate <system>_vars synced to inventory repo, use for AdditionalInventory in CFS

`update_system_ex.py` directly talks to system APIs to create CFS configurations with the commit ids in gitea.

Configuration Management - Secrets



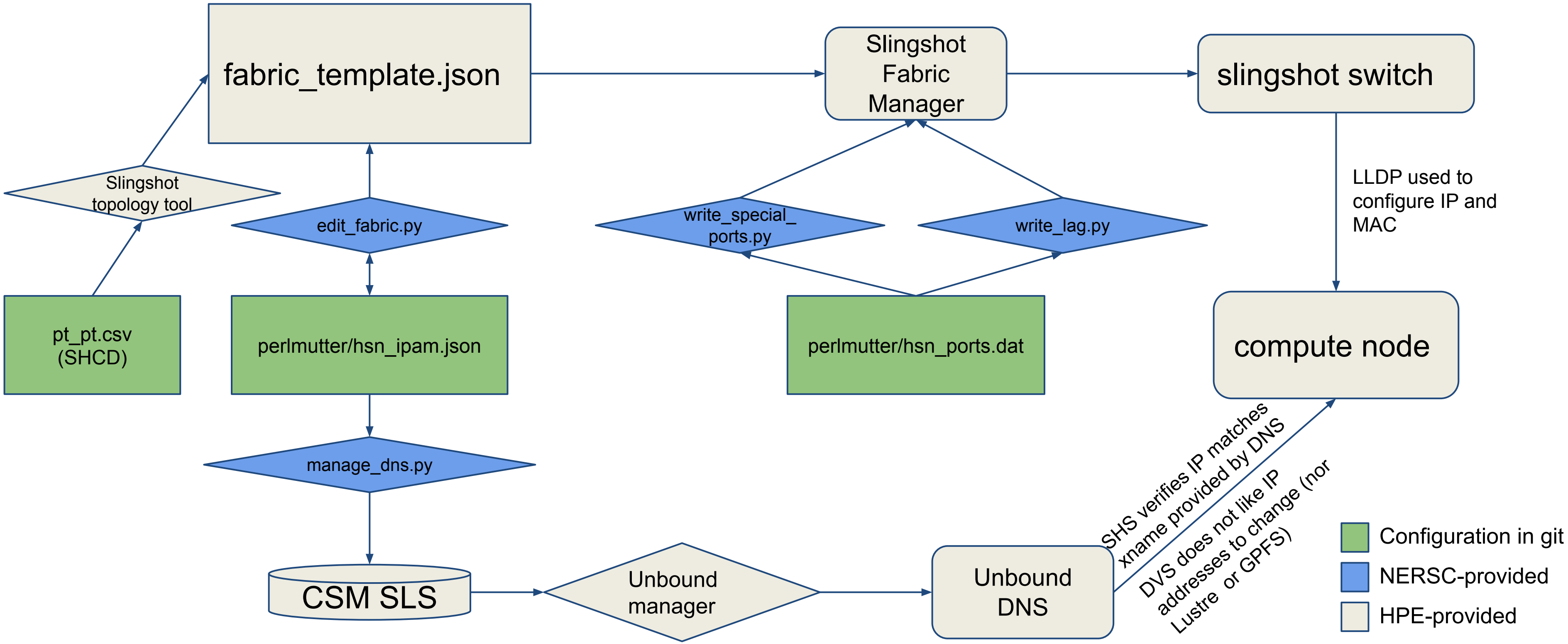
This requires policy changes by patching the Cray "vault/cray-vault" kubernetes object

We do not have an automated process for patching this and keeping it up-to-date

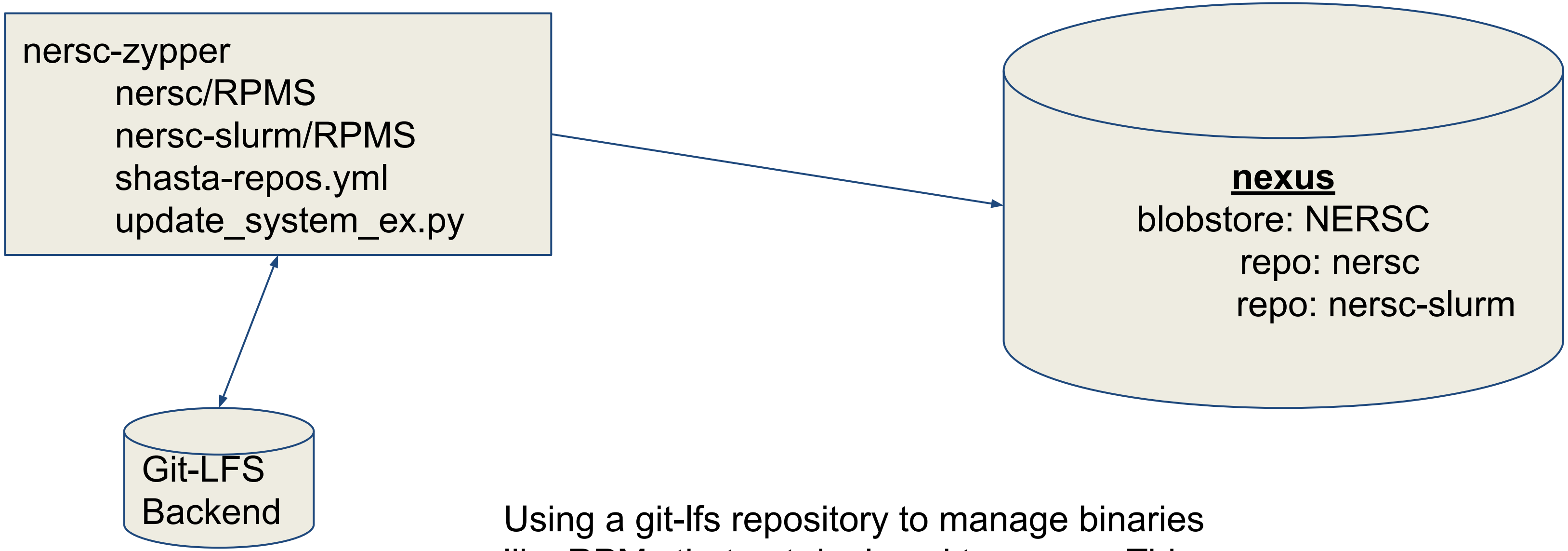
mgrkv has secrets that are only known on the manager VM, such as plaintext passwords, private keys for the host and admin CAs

syskv is sync'ed from the VM to cray-vault (using kubectl port forwarding) for use with CFS.
cray-vault also gets hashed passwords for deployment on the system

Configuration Management: HSN



Configuration Management: RPMs



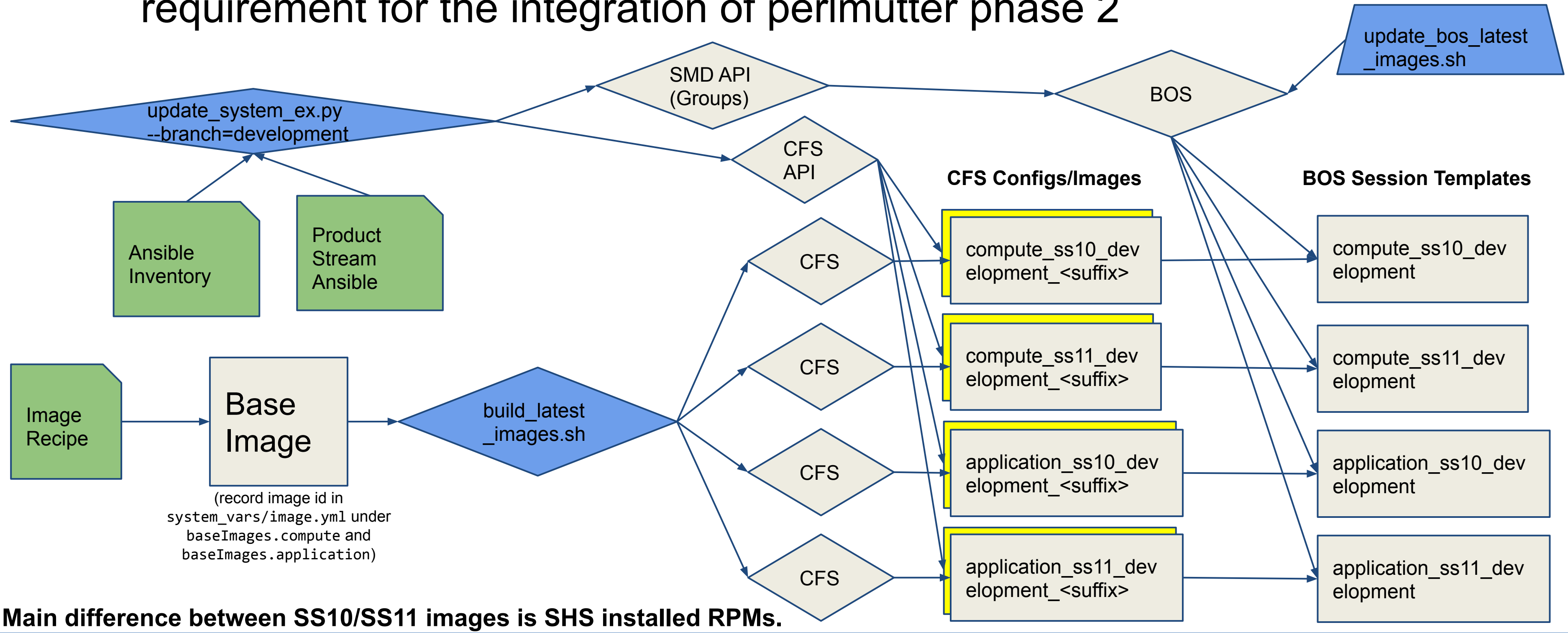
Using a git-lfs repository to manage binaries like RPMs that get deployed to nexus. This enables the same branching and collaborative testing we use for configuration files.

Configuration Management

- Helm Charts
  - NERSC-custom charts are stored in `nersc-cle` git repository
  - Deployed with Cray's `loftsman` from manager VM (leveraging end-user's privileges with kubernetes)
- Containers
  - Based on parse of nersc-cle charts, use skopeo to sync containers from external source, registry.nersc.gov, or VM-constructed container to on-system nexus
- customizations.yaml
  - Site/system overrides to helm charts. Presently not well managed because we need a process to generate sealed secrets from git sources (don't want to keep secrets, even encrypted in git)

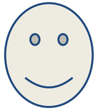
Managing SS10/SS11 Hybrid Systems

- This is explicitly *not* supported by HPE at this time. But is a necessary requirement for the integration of perlmutter phase 2



Main difference between SS10/SS11 images is SHS installed RPMs.

Configuration Management: Workflow



```
$ cd nersc-zypper
$ git checkout development
$ ./update_system_ex.py          # uploads RPMs to nexus
$ cd nersc-cle
$ git checkout development
$ ./update_system_ex.py --branch=development
     # record timestamp as `suffix
     # sync HSM groups for SS10/SS11 differentiation`
     # syncs secrets
     # writes feature/a ansible directories to dmjtest branches in gitea
     # generates CFS configuration objects, uploads using CFS API
          # compute-development-<suffix>
          # login-development-<suffix>
          # gateway-development-<suffix>
     # generates bos-sessiontemplates-<suffix> with unconfigured images
$ ./shasta/scripts/build_latest_images.sh development
$ cd bos-sessiontemplates-<suffix>
$ ../shasta/scripts/update_bos_latest_images.sh -i development -d development
     # generates and uploads usable BOS sessiontemplates
     # compute-development, login-development, gateway-development
$ cray bos session create --template-uuid compute-ss10-development --operation reboot
$ cray bos session create --template-uuid compute-ss11-development --operation reboot
```

Or use `prep_boot_config.py` to scalably rewrite BSS/CFS for all nodes and avoid bos completely. Enables dynamic rolling updates

Configuration: HPE Provided Software and Upgrades

- Installation of most CSM-compatible product streams have two phases:
  - Nexus artifact installation
    - Changes content in some common repositories non-destructively but content immediately is accessible and default for similar product lines (SLES repos vs SLES 22.01 repos)
  - Loftsman manifest deployment to configure the product
- NERSC does not have automation around the management of these products
  - the non-interactive `./install.sh` used *almost* across the board make automated data transfer/installation a clear future direction
- HPE Cray EX software recipe upgrade installation often start with CSM, then firmware (HFP), then COS and other products
  - This could result in three interruptions of each ncn-worker node
  - For major upgrades NERSC has usually found it's been fine to reorganize the update procedure to minimize the number of steps
    - The devil is in the details here, but there are many opportunities for automation and improving efficiency of the rolling update process

System Stabilization: DVS over HSN

- Switching DVS to HSN instead of NMN several benefits:
  - Nodes that have an incorrect HSN configuration fail to boot (making their problem obvious)
  - Use of HSN for DVS improves user experience because NMN bandwidth is limited
  - Increases options for backend LND (could use o2iblnd for SS10 or kkfilnd for SS11)
- In COS 2.0 and 2.1 this required patching the initrd to properly setup lnet and then dvs to add the needed lnds, and then modifying the `cos/uan_config_management` layers for configuration
- In COS 2.2 configuration in ansible inventory is all that is required

This was a **key** change during perlmutter deployment that led to major progress in the project

System Stabilization: CPS/DVS Worker Node Separation

- During perlmutter integration we had several instances of resource collisions and ncn-w* node crashes not tied directly to bugs in DVS, but contention with other critical workloads
- Perlmutter has 26 worker nodes, we chose to dedicate 8 of them to CPS/DVS by preventing management jobs to schedule on those nodes.
  - `cray cps deployment update --nodes=<list of nodes>`
  - `kubectl drain --delete-local-data --ignore-daemonsets <node one at a time>`
  - Leave them cordoned
- Estimate each worker node is capable of forwarding to 500 nodes
- NERSC's experience is that separating CPS/DVS from management pods has improved worker stability

What is the Impact?

- Focusing on *process-oriented* management that abstracts system details enables:
  - Scalable transfer of capability from small scale to test to large scale production
  - High fidelity transfer of features from test to production
  - Collaborative integration of difficult work on test systems with simple deployment to production
- Migrating HPE product stream code (ansible, recipes) into NERSC repository:
  - Enables rapid workarounds or improvements in HPE-provided code
  - Reinforces the separation of code vs data
  - Supports inclusion of HPE products in the development/deployment process

Conclusions

- Using the techniques in this presentation, NERSC was able to successfully deploy and maintain three Cray EX systems with minimal overlapped effort
- Insights gained using this work has enabled NERSC and HPE to develop a method of iteratively deploying perlmutter phase 2 with minimal disruption, despite all the hardware in the system being taken offline at some point during the process
- NERSC has demonstrated repeatedly the utility of having test systems to optimize deployment procedures and even start doing scalable test by moving production resources