EL CAPITAN

El Capitan Systems Getting Started Guide March 2026 Respin with Latest Information

Updated March 23, 2026 based on slides originally presented August 4-5, 2025

Ramesh Pankajakshan , Ryan Day, Scott Futral, John Gyllenhaal

Livermore Computing (LC)

Prepared by LLNL under Contract DE-AC52-07NA27344.

Subpresentations in This Guide

11 slides | El Capitan Systems Architecture | Ramesh Pankajakshan

14 slides | Flux for El Capitan Systems | Ryan Day

30 slides | Building and Running Successfully | John Gyllenhaal

El Capitan Architecture

Ramesh Pankajakshan
Lead, El Capitan COE

August 4-5, 2025 (No changes March 2026)

LLNL-PRES-2007699
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

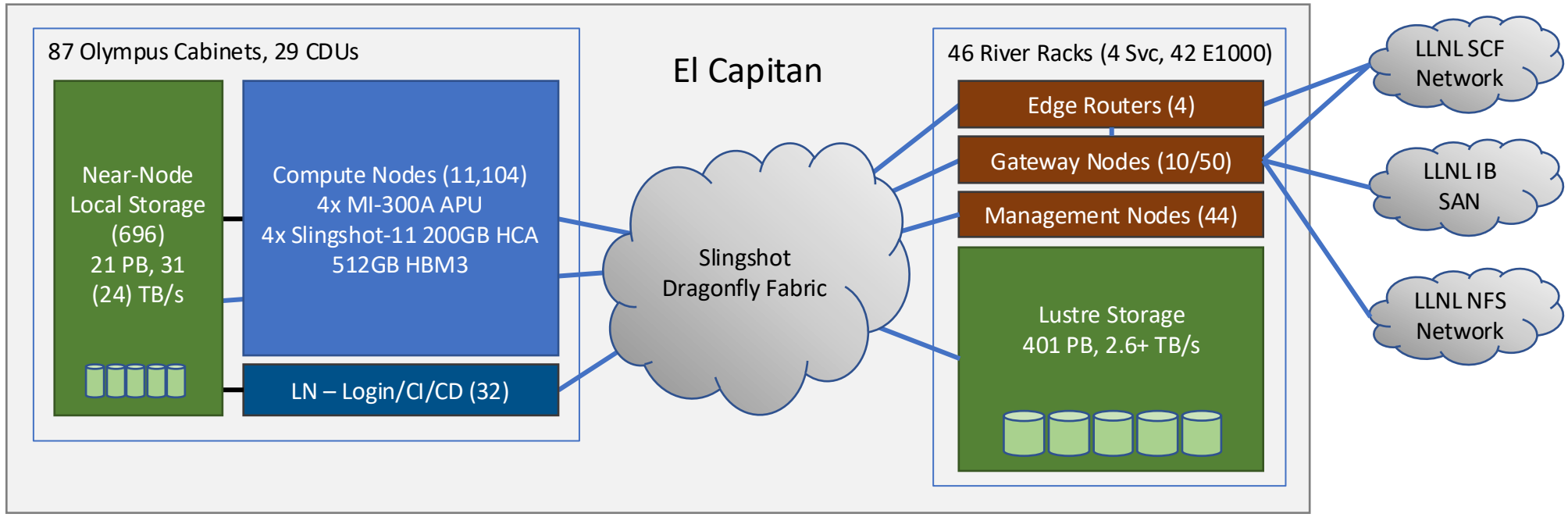
Lawrence Livermore National Laboratory

HPE has delivered a highly capable AMD GPU-accelerated system

- El Capitan will meet its stockpile stewardship simulation mission
- System will feature:
  - Peak 2.8 DP exaflops
  - Peak power 34.8MW
    - HPL 29.6MW
  - AMD MI300A APU - 3D chiplet design w/AMD CDNA 3 GPU, “Zen 4” CPU, cache memory and HBM chiplets
  - Slingshot interconnect
- HPE provides several critical innovations
  - HPE and LLNL have worked with ORNL jointly on non-recurring engineering (NRE) activities
  - MI300A, world’s first data center APU directly addresses multiple challenges
  - Uses TOSS software stack, enhanced with HPE software
  - El Capitan includes an innovative near node local storage solution (Rabbit)

Late binding of the processor solution has ensured El Capitan provides the best possible value

El Capitan Architecture



AMD INSTINCT™ MI300A: The world's first data center APU

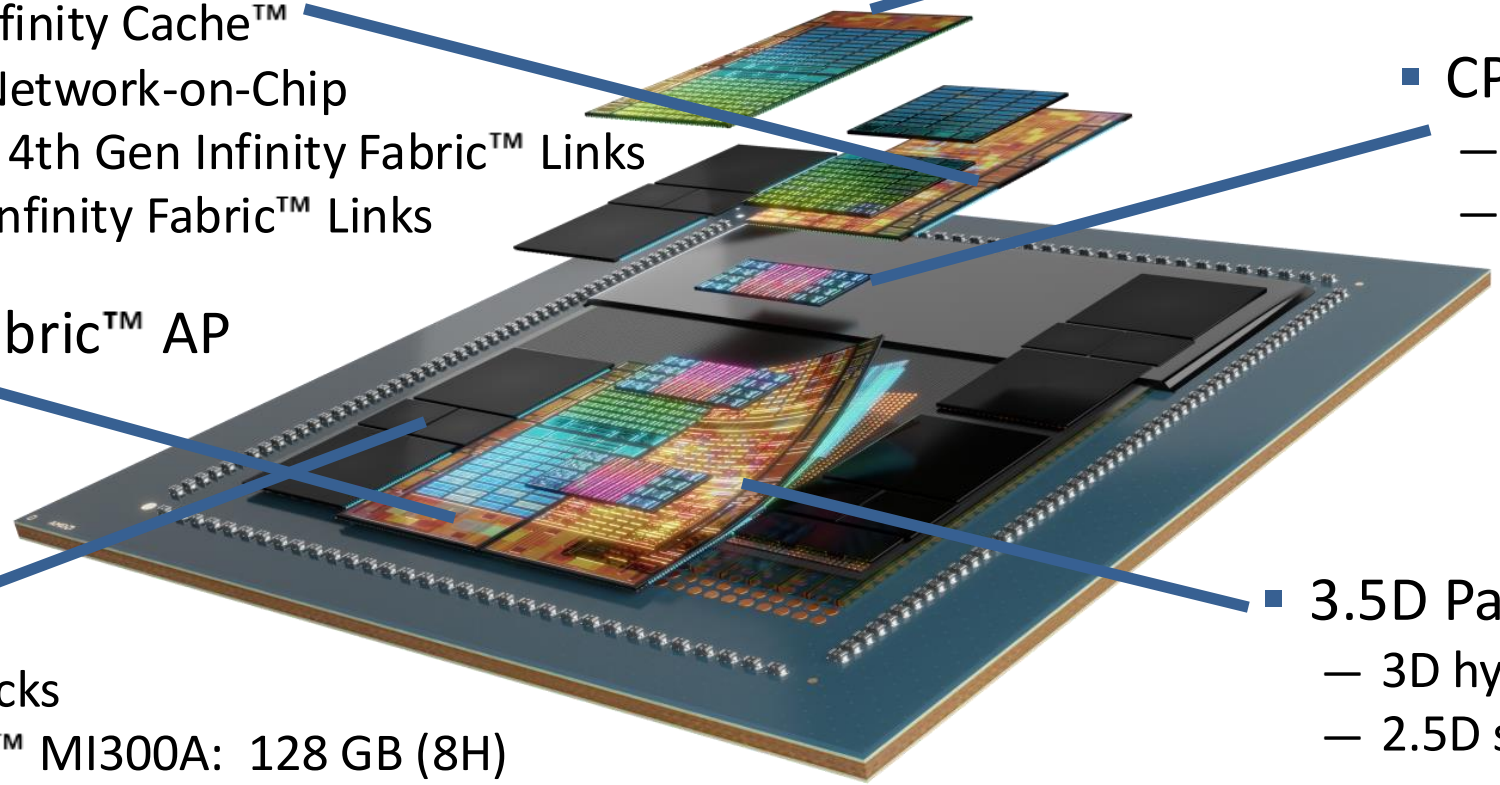
- 4th Gen AMD Infinity Architecture:
  AMD CDNA™ 3 and EPYC™ CPU "Zen 4" together
  - CPU and GPU cores share a unified on-package pool of memory
- Groundbreaking 3D packaging
  - CPU | GPU | Cache | HBM
  - 24 Zen4 cores, 146B transistors, 128GB HBM3
- Designed for leadership memory bandwidth and application latency
- APU architecture designed for power savings
  - compared to discrete implementation

> 8X Expected AI Training Performance vs. MI250X

Preliminary data and projections, subject to change

AMD Instinct™ MI300 Modular Chiplet Package

- I/O Die (IOD)
  - 128 Channel HBM3 Interface
  - 256MB AMD Infinity Cache™
  - Infinity Fabric Network-on-Chip
  - 4 x16 PCIe® 5 + 4th Gen Infinity Fabric™ Links
  - 4 x16 4th Gen Infinity Fabric™ Links
- AMD Infinity Fabric™ AP Interconnect
- HBM3
  - 8 physical stacks
  - AMD Instinct™ MI300A: 128 GB (8H)
  - AMD Instinct™ MI300X: 192 GB (12H)
- Accelerator Complex Die (XCD)
  - 6 x 38 AMD CDNA™ 3 Compute Units
- CPU Complex Die (CCD)
  - 3 x 8 "Zen 4" Cores
  - Replaced by additional XCDs on MI300X
- 3.5D Package
  - 3D hybrid bonding
  - 2.5D silicon interposer



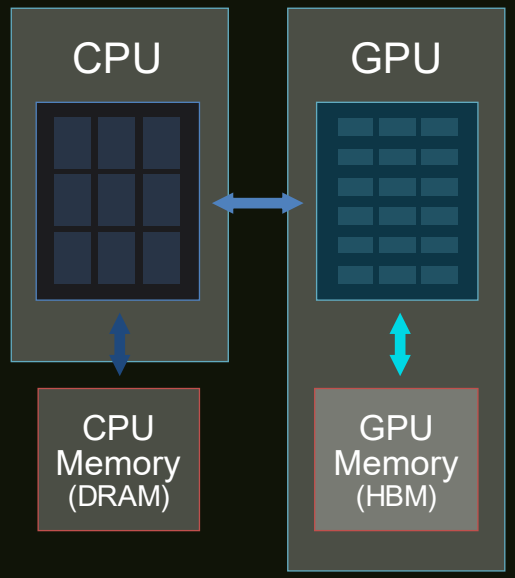
3D CPU+GPU integration for next-level efficiency

EL CAPITAN
ADVANCED SIMULATION & COMPUTING

AMD CDNA™ 2 Coherent Memory Architecture → **AMD CDNA™ 3 Unified Memory APU Architecture**

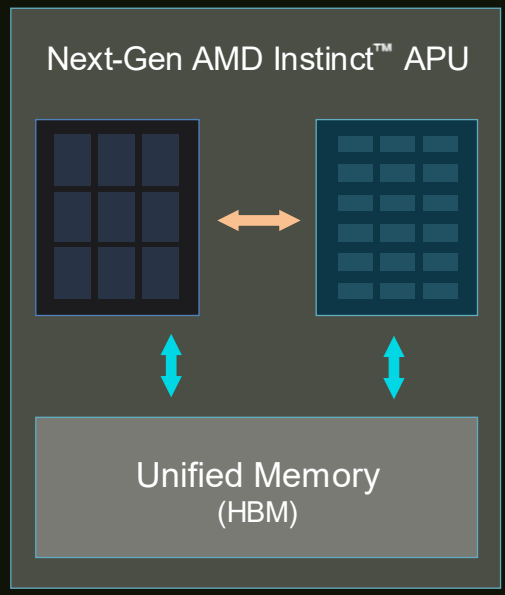
AMD Instinct™ MI250 Accelerator

- Simplifies programming
- Low overhead 3rd Gen Infinity interconnect
- Industry standard modular design

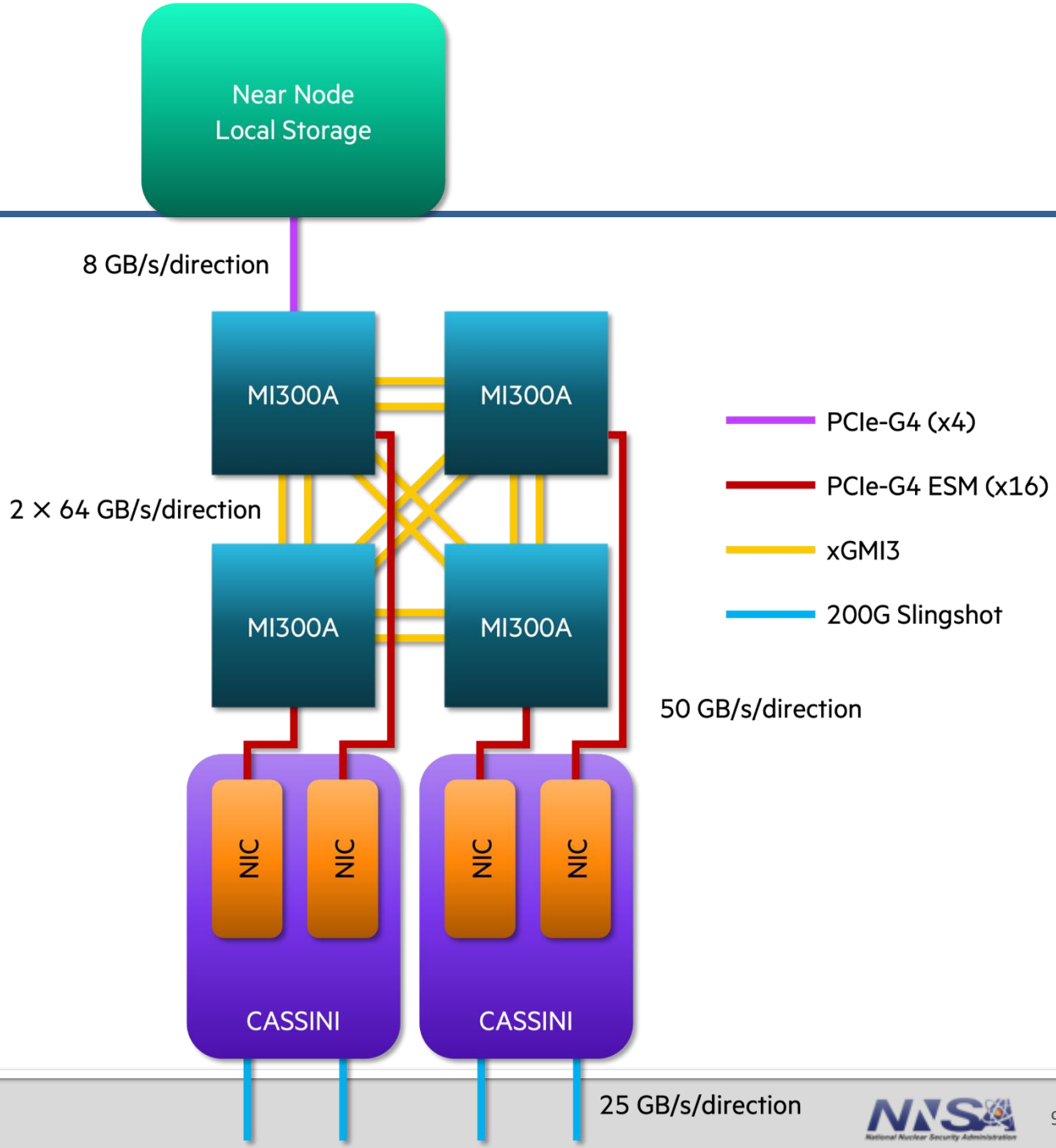


AMD Instinct™ MI300 Accelerator

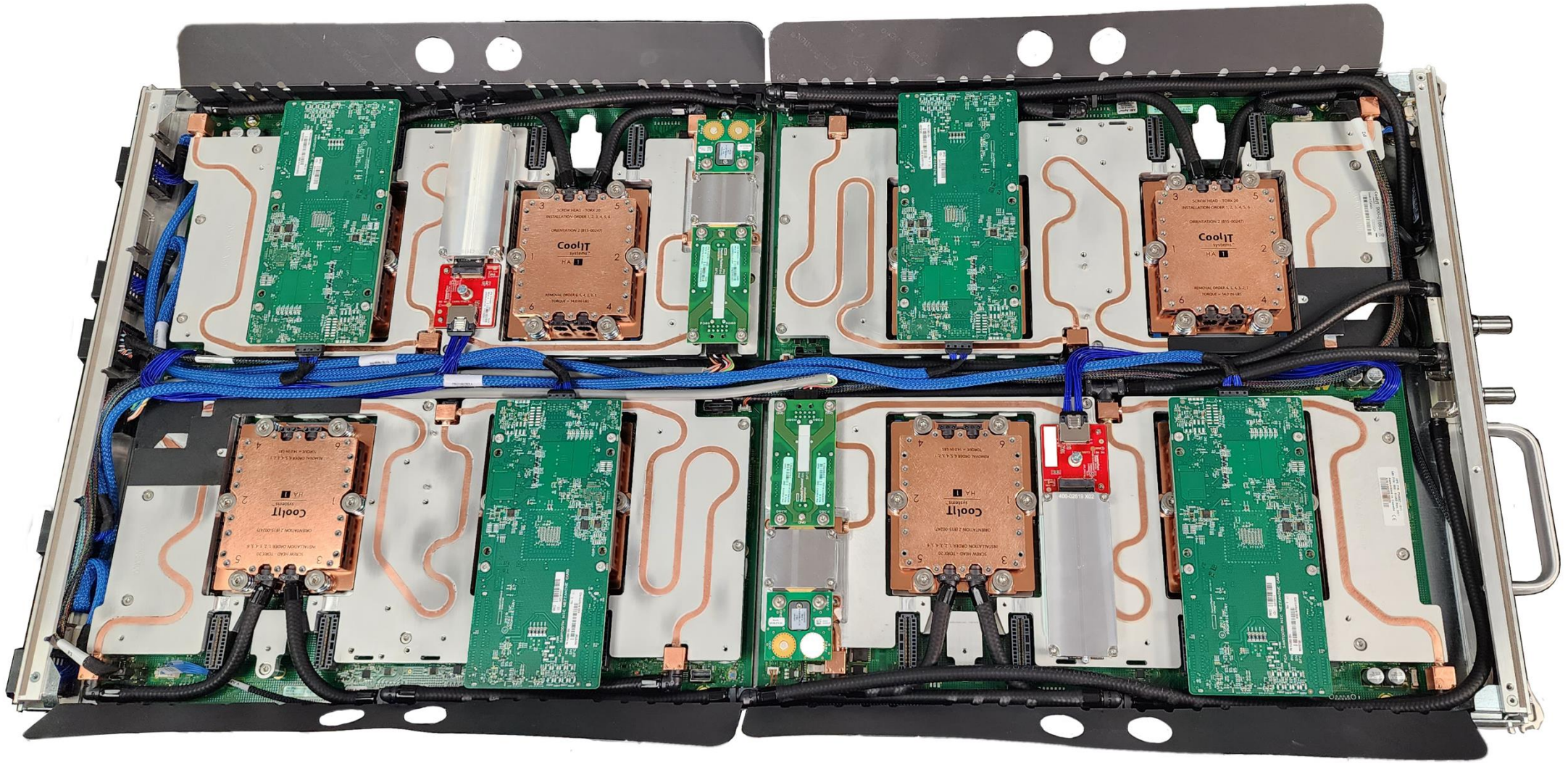
- Eliminates redundant memory copies
- High bandwidth, low latency communication
- Low TCO with unified memory APU package



Node Architecture

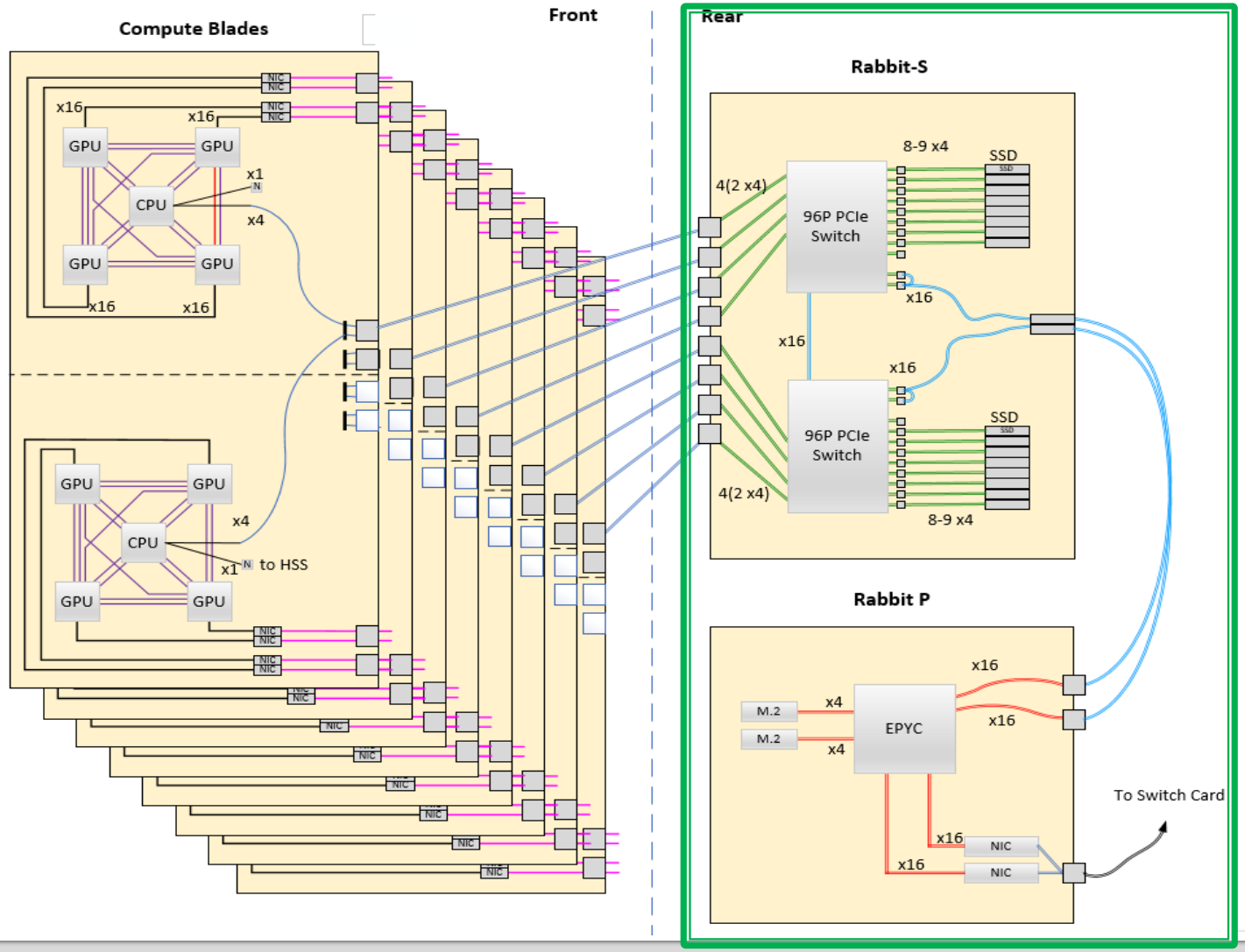


MI300A at HPE: The HPE compute blade



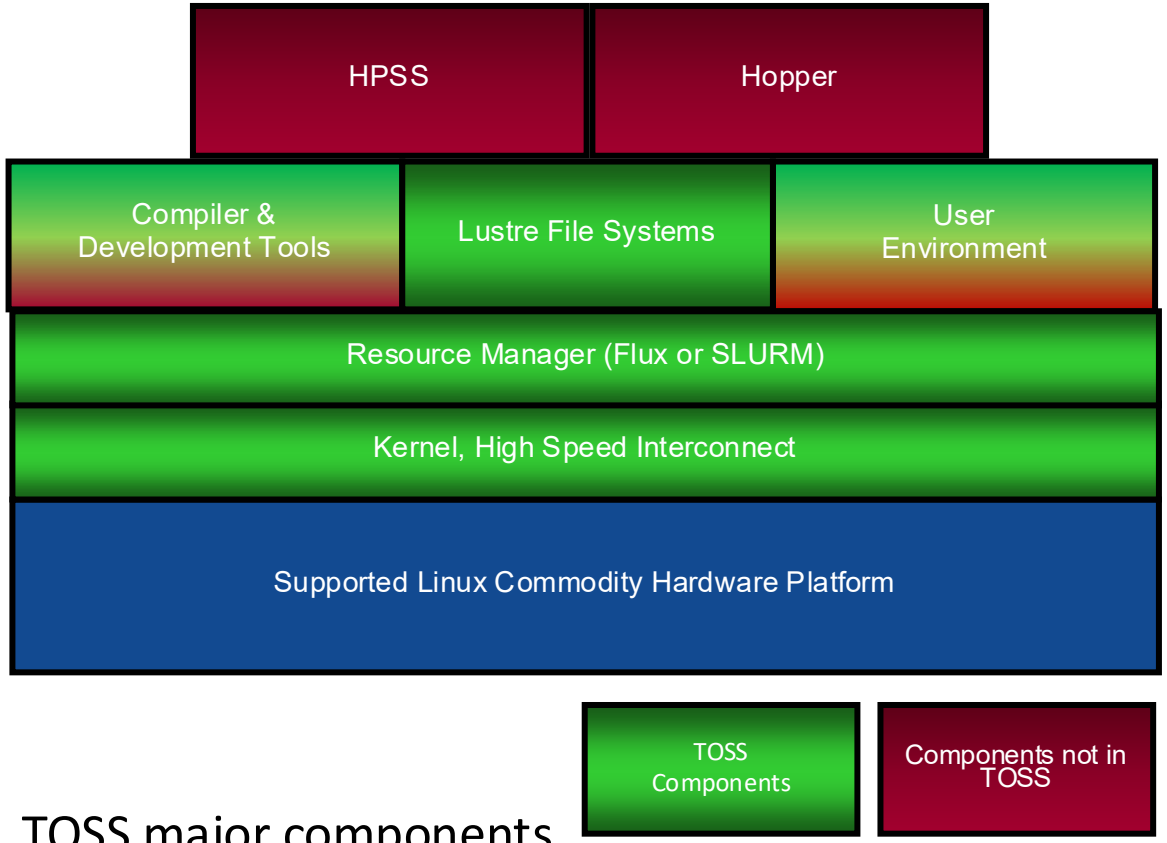
Rabbit modules are a 4U near node local storage solution

- All in one solution: Rabbit- 4U
  - Houses 16 SSDs that attach to Rabbit-S board
  - Locates Storage Processor (AMD Epyc CPU) on Rabbit-P board
- Compute blades direct attached to Rabbit-S through bulkhead cables
- Rabbit-S to Rabbit-P board connections are internal (no external cables)

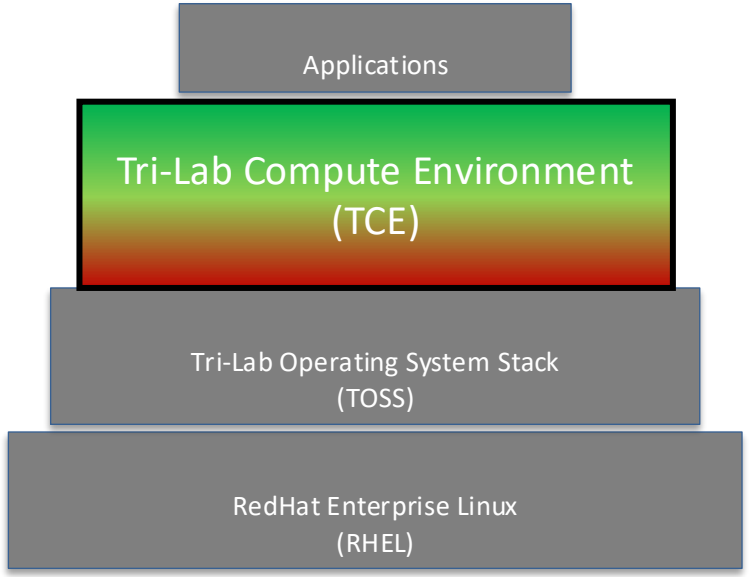


Rabbit 4U design provides easy access to SSDs

El Capitan will be the first ATS to use TOSS and TCE in production



- TOSS major components
  - The OS – LLNL's Linux distribution based on RHEL
  - Resource Manager (SLURM or Flux)
  - Lustre



- The Tri-Lab Compute Environment (TCE) is an application development environment (DE)
  - Compilers (Intel, PGI, GNU, …)
  - MPI (MVAPICH, OpenMPI, …)
  - Debuggers (TotalView, Allinea)
  - Performance Tools

Using Flux on ElCap and Tuo

CORAL2 roadshow

Updated 3/23/26. Originally presented 8/4-5/2025

Main updates: Spindle debugging tweaks and --amd-gpumode={CPX|TPX|SPX}

Ryan Day, LC Resource Management
Livermore Computing

Prepared by LLNL under Contract DE-AC52-07NA27344.

Overview

- Using Flux like Slurm.
- Flux differences from Slurm.
- Priority, limits, queues, etc on CORAL2 systems.
- CORAL2 specific features.
- Where to find out more.

You can use Flux like Slurm

- Most Slurm commands and flags have an equivalent Flux command

Slurm command	Flux command
`srun`	`flux run | flux submit`
`sbatch`	`flux batch`
`salloc`	`flux alloc`
`squeue`	`flux jobs -A`
`scancel`	`flux cancel`

Slurm flag	Flux flag
`-N`	`-N`
`-n`	`-n`
`-t`	`-t`
`-A`	`--bank`
`-p`	`-q`

https://hpc.llnl.gov/banks-jobs/running-jobs/batch-system-cross-reference-guides

We have wrappers for many Slurm commands

```
[day36@tuolumne2151:~]$ which srun
/usr/global/tools/flux_wrappers/bin/srun

[day36@tuolumne2151:~]$ ls /usr/global/tools/flux_wrappers/bin/
bankinfo  mshare       sbatch showq     squeue  sxterm
checkjob  quickreport  sbcast sinfo     srun    utilizationreport
jobinfo   salloc       scancel slurm2flux  suflux

[day36@tuolumne2151:~]$ salloc -N1 -p pdebug -t 15 -v
# running: flux --parent alloc --nodes=1  --queue=pdebug  --time-limit=900s  --verbose
jobid: f22WbqzS3gJj
flux-job: f22WbqzS3gJj started                                          00:00:03
[day36@tuolumne1015:~]$
```

Flux differences: `--parent` and dependent jobs

```
[day36@tuolumne2151:~]$ flux alloc -N 1 -q pdebug
flux-job: f22WqgCeuuf5 started                                          00:00:04

[day36@tuolumne1012:~]$ flux jobs -A
       JOBID USER     NAME       ST NTASKS NNODES     TIME INFO

[day36@tuolumne1012:~]$ flux --parent jobs -A
       JOBID QUEUE    USER     NAME       ST NTASKS NNODES     TIME INFO
f22SL6vpUBxT pbatch   jones289 ./train_s+  S      4      4      12h eta:2.044h
f22SL79RN5tT pbatch   jones289 ./train_s+  S      4      4      12h
…

[day36@tuolumne1012:~]$ flux --parent batch --dependency=afterany:$(flux getattr jobid) ...
```

Flux differences: `--exclusive` and mpibind

```
[day36@dane7:~]$ salloc -N1 -p pdebug --exclusive srun -n2 numactl -s | grep physcpu
…
physcpubind: 168 169 170 171 172 173 174 175 176 177 178 …
physcpubind: 112 113 114 115 116 117 118 119 120 121 122 …
[day36@dane7:~]$ salloc -N1 -p pdebug --exclusive srun -n2 --exclusive numactl -s | grep physcpu
…
physcpubind: 56 168
physcpubind: 0 112

[day36@tuolumne2151:~]$ flux alloc -N1 -q pdebug --exclusive flux run -n2 numactl -s | grep physcpu
physcpubind: 95
physcpubind: 94

[day36@tuolumne2151:~]$ flux alloc -N1 -q pdebug --exclusive flux run -N1 -n2 --exclusive numactl -s | \
grep physcpu
physcpubind: 1 2 3 4 5 6 7 9 10 …
physcpubind: 49 50 51 52 53 54 55 57 58 …
```

Mpibind assigns both of these tasks to the same GPU

Mpibind assigns these tasks to different GPUs

Mpibind and system cores (MPIBIND_RESTRICT keeps tasks off of system cores)

CURRENTLY:

```
[day36@tuolumne2151:~]$ flux alloc -N1 -q pdebug
flux-job: f22WyYbNQe9M started                                  00:00:04

[day36@tuolumne1012:~]$ echo $MPIBIND_RESTRICT
1-7,9-15,17-23,25-31,33-39,41-47,49-55,57-63,65-71,73-79,81-87,89-95,…

[day36@tuolumne1012:~]$ flux run -N1 -n2 --exclusive numactl -s | grep physcpu
physcpubind: 1 2 3 4 5 6 7 9 10 11 12 13 14 15 17 18 19 20 21 22 23 …
physcpubind: 49 50 51 52 53 54 55 57 58 59 60 61 62 63 65 66 67 68 69 70 71 …

[day36@tuolumne1012:~]$ MPIBIND_RESTRICT= flux run -N1 -n2 --exclusive numactl -s | \
grep physcpu
physcpubind: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 …
physcpubind: 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 …
```

Still pending March 2026, in future: Flux will understand user cores natively

Other useful Flux commands: queued and completed jobs

```
[day36@tuolumne2151:~]$ flux jobs -A
       JOBID QUEUE    USER     NAME       ST NTASKS NNODES     TIME INFO
f22SL7ZdLMm1 pbatch   jones289 ./train_s+  S      4      4      12h eta:1.664h
…
f228wkgYA4vT pbatch   shin9    submit/es+  D      8      8       1d depends:after-finish=f228wkW1MApP
f22X2AZS4poq pdebug   mast2    flux        R      1      1   2.265m tuolumne1026
…

[day36@tuolumne2151:~]$ flux jobs -A -o deps
       JOBID QUEUE    NAME       URG PRI          STATE    DEPENDENCIES
f22SL7ZdLMm1 pbatch   ./train_s+ 16  99724        SCHED
…

[day36@tuolumne2151:~]$ flux jobs -a -o endreason
       JOBID QUEUE    USER     NAME       ST   T_INACTIVE INACTIVE-REASON
f22WyYbNQe9M pdebug   day36    flux       CD  Jun18 11:11 Exit 0
f22WtaJY2ka3 pdebug   day36    flux        F  Jun18 10:53 Exit 1
f22WpvMnMktw pdebug   day36    flux       CA  Jun18 10:45 Canceled: interrupted by ctrl-C
```

Job priority and standby

Flux uses FairTree (https://slurm.schedmd.com/fair_tree.html) for fairshare priority. View banks and users with `flux account` or `bankinfo`:

```
[day36@tuolumne2151:~]$ bankinfo -T root -v
Name                    Shares  Norm Shares         Usage   Norm Usage   Level FS   Priority  Type
ROOT                         1     0.000000  1076809737.3     0.000000          --         --  Bank
 parent                 331000     0.114221   277095302.8     0.257330          --         --  Bank
  childA                   373     0.000129           0.0     0.000000          --         --  Bank
  childB                  1000     0.000345       36436.5     0.000034          --         --  Bank
…
 overhead                    1     0.000000   661799186.3     0.614592          --         --  Bank
  guests                     1     0.000000   626133217.7     0.581471          --         --  Bank
  lc                         1     0.000000    25017080.6     0.023233          --         --  Bank
  standby                    1     0.000000    10648888.0     0.009889          --         --  Bank
```

Queues, limits, and DATs

Please observe any social limits in `news job.lim.CLUSTER`. You can list queues and nodes with `flux queue list` and `flux resource list`. You can submit jobs to any `enabled` queue, but they will only run if the queue is also `started`.

The normal queues are:

pbatch: 24 hour time limit, max job is 4096 nodes (elcap) or 256 nodes (tuo).

pdebug: 1 hour time limit, intended for interactive debugging, visualization, and other inherently interactive work. Do not use more than half the pdebug nodes during work hours. Do not use pdebug to run batch jobs. Do not chain jobs to run one after the other. Individuals who misuse the pdebug queue in this or any similar manner may be denied access to running jobs in the pdebug queue.

Other queues you may encounter:

plarge: intended for larger scale debug runs on elcap. Generally started Thursdays by request.

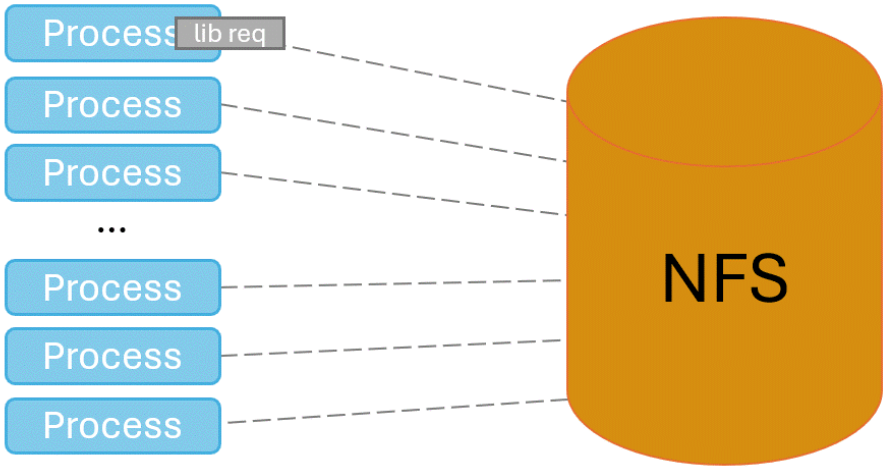
pdat_MMDD: created for specific users for Dedicated Application Times (DATs).

pall: Mostly used for system admin testing, but may be used by full system DATs.

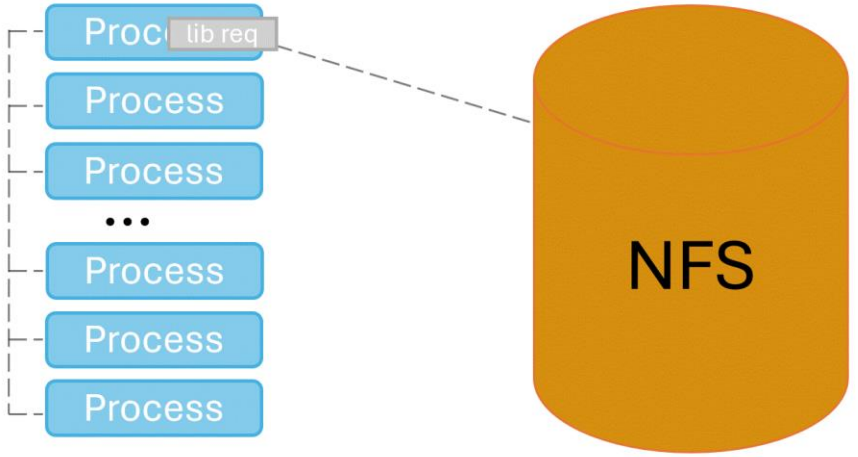
pci: very small queue for Gitlab CI jobs.

CORAL2 Specific Capabilities: Spindle

Spindle improves application start-up scalability by staging libraries and python.



Without Spindle application processes overwhelm the shared file system loading libraries at start-up.



With Spindle application processes coordinate library loading for faster start-up.

CORAL2 Specific Capabilities: Spindle

Spindle is on by default on TUOLUMNE and ELCAP and is critical for preventing apps from crushing file system. However, spindle may break your application until spindle is tuned for it (expecially pyTorch):

If your app breaks mysteriously on Tuolumne or Elcap, trying temporarily trying fastload instead or turning off spindle:

```
fastload flux run …                  // Recommend trying fastload instead of spindle first
```

-or-

```
flux run -o fastload …               // Alt fastload invocation through flux
```

-or-

```
export SPINDLE_FLUXOPT=off           // Do not run more than 16 nodes with Spindle off
```

-or-

```
flux run -o spindle.level=off …  // Alt spindle off, keep under 16 nodes
```

Please email the LC Hotline (lc-hotline@llnl.gov) and ccing John Gyllenhaal (gyllenhaal1@llnl.gov) if using fastload or turning off spindle solves a crash or mysterious error in your application on Tuolumne or ELCAP so we can patch Spindle!

Make spindle more efficient for many jobs in an allocation with:

```
export SPINDLE_PATH= /collab/usr/global/tools/spindle/toss_4_x86_64_ib_cray_live
$SPINDLE_PATH/bin/spindle --start-session
  flux run ...
  flux run ...
  flux submit ...
  #wait for job completion
$SPINDLE_PATH/bin/spindle --end-session
```

CORAL2 specific options: --amd-gpumode=CPX|TPX|SPX

The MI300A GPUs on ElCap and Tuo can be made to appear as multiple logical GPUs with TPX (12) and CPX (24) mode:

```
[day36@tuolumne2151:~]$ flux alloc -N1 -q pdebug --amd-gpumode=CPX
flux-job: f22XDH7AQwa7 start                                                    00:00:04
[day36@tuolumne1006:~]$ /opt/rocm-6.2.4/bin/rocm-smi
========================================= ROCm System Management Interface =========================
====================================================== Concise Info ================================
Device  Node  IDs                Temp        Power     Partitions          …
              (DID,     GUID)  (Junction)  (Socket)  (Mem, Compute, ID)
============================================================================================
0       4     0x74a0,   6167   47.0°C      129.0W    NPS1, CPX, 0        …
1       5     0x74a0,   40982  47.0°C      129.0W    NPS1, CPX, 1        …
…
22      26    0x74a0,   24562  46.0°C      76.0W     NPS1, CPX, 1        …
23      27    0x74a0,   55282  46.0°C      76.0W     NPS1, CPX, 2        …
============================================================================================
================================================ End of ROCm SMI Log ========================
```

Learn more

- https://hpc.llnl.gov/documentation/user-guides/using-el-capitan-systems
- https://hpc-tutorials.llnl.gov/flux/
- https://hpc.llnl.gov/banks-jobs/running-jobs
- https://flux-framework.readthedocs.io/en/latest/
- `flux help`
- `flux *command* help` / `man flux *command*`
- lc-hotline@llnl.gov / 925-422-4531

Questions?

Building and Running Successfully on El Capitan Systems

El Capitan Systems Getting Started Workshop

Originally presented August 4-5, 2025

Updated March 23, 2026 - See next three slides for change summary

John Gyllenhaal
Livermore Computing (LC)

Prepared by LLNL under Contract DE-AC52-07NA27344.

March 2026 change summary since Aug 2025 (1/3)

- Stable/recommended: cce/20.0.0 or rocm/6.4.3 + cray-mpich/9.0.1 (must use 6.4.3 for key hang fixes)
  - cce/20.0.0a-magic (or cce/20.0.0 with rocm/6.4.3)
  - rocmcc/6.4.3-magic or rocmcc/6.4.3-cce-20.0.0a-magic or rocmcc/6.4.3-cce-20.0.2-magic
  - cray-mpich/9.0.1 // Compatible with rocm/6.4.3 but not rocm/7*, most performance and fixes with rocm 6.4.*
  - cce/20.0.2-magic (or cce/20.0.2 with rocm/6.4.3) // Interface changes, recommend only if need LLNL fixes below
    - (ELCAP-133) Unable to use runtime variable in openmp thread_limit(var) clause
    - (ELCAP-621) CCE 19 Ftn gives incorrect results with O2 or O3
    - (ELCAP-641) ftn openmp target compile rejects loops with equivalenced arrays
    - (ELCAP-899) CCE 20 fortran link time increase compared to CCE 19
- New/forward-looking: cce/21.0.0 or rocm/7.2.1 + cray-mpich/9.1.0 (7.2.1 installed 3/31/26)
  - cce/21.0.0a-magic (or cce/21.0.0 with rocm/7.2.1)
  - rocmcc/7.2.1-magic or rocmcc/7.2.1-cce-21.0.0a-magic
  - Can also use rocm/7.2.0 now but many fixes in 7.2.1
  - cray-mpich/9.1.0 // Compatible with rocm/7* but not rocm/6*, more hang fixes
  - cray-mpich/9.1.0 only visible if rocm/7-compatible compiler module is loaded!
- New default HSA_ENABLE_IPC_MODE_LEGACY=1 required to make 9.1.0 GTL work with rocm/7.2.*

March 2026 change summary since Aug 2025 (2/3)

- SHS 13 libfabric and libcxi now default for magic wrappers for cray-mpich/9.0.1 & 9.1.0
  - For executables relinked after 3/18/26 magic wrapper update. No change for cray-mpich/8*
  - /usr/lib64/libfabric found to be much faster and stable for at least one application
  - Linking with --oldfabricdefault restores pre-3/18/26 RPATH behavior (use of old libfabric 2.1 w/new cxi)
  - Linking with --shs13 pins to exactly current SHS 13 libraries but not recommended unless needed
  - Linking with --shs2.1 pins to exactly old SHS 2.1 libcxi and libfabric libraries (not recommended)
- New CPX & TPX flux alloc options: --amd-gpumode=CPX or --amd-gpumode=TPX
  - SC25 study found better L2 cache hit rate in CPX mode due to higher data locality in stencil loops
- New pyTorch launcher tool: https://github.com/LBANN/HPC-launcher
- stat-cl -magic <jobid>, stat-gui -magic <jobid> makes STAT easy on running jobs
- python/3.13.2 new python default with wrapper to enable rocm tools
- hipMalloced buf in bcast and other collectives without GTL triggers ‘cxil_map: write error’
  - Workaround use GTL with MPICH_GPU_SUPPORT_ENABLED=1

March 2026 change summary since Aug 2025 (3/3)

- Current flux batch script recommendations to facilitate debugging
  - First error out of code most likely indicates real problem
    - Most flux messages afterwards are just from your application dying, not the cause
  - Separate stderr file in flux batch script makes finding first error much easier

    #flux:--output='output_{{name}}.{{id}}.out'

    #flux:--error='output_{{name}}.{{id}}.err'
  - Run check_elcap_nodes in batch script first for quick (< 15 secs) node diagnostics
  - Run on_hang_stat_and_kill <options> before main application to debug hangs
    - Run on_hang_stat_and_kill -h for full instructions and examples on CORAL2 machine
    - Also have on_hang_kill and on_hang_stat available
- rocmcc/6.4.3-magic supports LLNL-specific --auto-asan option for C/C++
  - Need --auto-asan for both compiling and linking
  - Recommend setting these two env variables when running instrumented asan executable
    - export HSA_XNACK=1
    - export ASAN_OPTIONS="halt_on_error=0:detect_leaks=0"

Goals and Scope

- Present concise and actionable rules of thumb for using El Capitan
  - Present key concepts and commands for compiling applications and running on El Cap systems
  - Make you aware of the existence of several complex technical details that may impact your application
    - If your application is impacted, will indicate how to reach out to get focused help
- Aiming for biggest impact with a “short” presentation
  - Presenting the key points of dozens of hours of NDA presentations over the last few years
  - Will not dive into all the juicy technical details or be able provide full answers to questions
    - The “real” answers to “why” are often quite complex, confusing, time consuming, and requiring NDAs
- No NDA required for this talk’s info
  - Often limits discussion of the technical details that drive the rules of thumb presented
- Snapshot in time
  - Systems still evolving as new and improved functionality comes online (UPDATED March 2026)

Where to get help and key documentation for today's talks

- Point of contact for getting help or asking questions
  - LC Hotline: lc-hotline@llnl.gov or 925-422-4531
- El Capitan user guides (mi300a's: elcap, tuolumne or tuo, rzadams mi250x's/EAS: rzvernal, tioga, tenaya)
  - https://hpc.llnl.gov/documentation/user-guides/using-el-capitan-systems
- Run and scheduling jobs with Flux
  - https://hpc-tutorials.llnl.gov/flux/
  - https://hpc.llnl.gov/banks-jobs/running-jobs/batch-system-cross-reference-guides
- Accessing LC systems smoothly with ssh
  - https://hpc.llnl.gov/documentation/user-guides/accessing-lc-systems
- Accelerating PyTorch, TensorFlow, or any ML code using RCCL (internal modules, files, websites)
  - New recommended resource for launching PyTorch on LC: https://github.com/LBANN/HPC-launcher
  - Use either module 'rccl/working-env' or 'rccl/fast-env-slows-mpi' to be compatible with system's current MPI
  - Sample build script /collab/usr/global/tools/rccl/toss_4_x86_64_ib_cray/rocm-6.2.1/buildme
  - https://lc.llnl.gov/confluence/display/LC/2023/06/02/RCCL+performance+on+Tioga#RCCLperformanceonTioga-InstalltheAWS-OFI-RCCLplugin
  - https://lc.llnl.gov/confluence/display/LC/Distributed+PyTorch+on+CORAL+2+systems#DistributedPyTorchonCORAL2systems-Extra:InstallAWS-OFI-RCCLplugin

Accessing El Capitan Systems: Ssh Key Hints and Gotchas

- Elcap is on the Secure (SCF) classified network and Tri-labs users should use cross-realm authentication
- RZadams is in the Restricted Zone (RZ) and Tri-lab users should use IHPC cross-realm authentication
- Tuolumne is in the Collaboration Zone (CZ) and you can ssh tuo.llnl.gov or ssh tuolumne.llnl.gov without VPN
- You cannot use ssh keys to connect to LC supercomputers from outside LC (in any zone)
  - Must use a two-factor authentication or tri-lab cross-realm authentication to get into an LC supercomputer
  - https://dev.llnl.gov/security-access/ssh/ has ssh config files and info that may help with ssh proxy setups
    - Can proxy through initial connection (often oslic for CZ, rzslic for RZ, and cslic for SCF) for additional connections
    - Ian Lee's .ssh/config files are FOR PLACEMENT ON YOUR WORKSTATION ONLY (!) NOT ON LC SUPERCOMPUTERS
    - Placing in your LC .ssh/config file very common mistake that causes unnecessary authentications and makes it fragile
- We require you to use 4k bits (or larger) rsa ssh keys (without passphrase) on LC
  - ssh-keygen -t rsa -b 4096 -N ""
    - LC has common home directories, so empty passphrase does not reduce security and is approved for use at LC
  - Will enable you to ssh between LC machines without password or passphrase
- Do not put LC private keys on different zones (don't share CZ, RZ, or SCF private keys)
  - Do not copy your CZ LC .ssh/id_rsa file outside the CZ or RZ key outside of the RZ!

High-level El Cap MI300A APU-based Node Overview

- Four sockets per node
  - Each socket contains one powerful GPU, 21 user cores (+ 3 system) (2 HW threads/core), and 128 GB HBM3 memory
    - Roughly 120GB memory per socket available to users (OS and system processes use the rest)
  - Each node has 4 GPUs, 84 user cores and 12 system cores and 512GB HBM3 memory
  - GPUs, CPUs, OS, ram disks (i.e., /tmp) all share same HBM3 ”GPU” memory (no DRAM) and unified address space
  - Four NUMA domains per node
    - Best practice: 4 MPI ranks per node, each using 1 GPU with memory and cores from same socket via binding
  - Each GPU can now be split into 3 (TPX mode) or 6 (CPX mode) virtual GPUs per socket at node allocation time but support is still evolving
- Binding is critical, everything must be on same socket for good performance
  - Microbenchmarks run up to 30X slower if GPU accessing memory from different socket
    - Although you can actually access all 480GB avail memory from one GPU, it will be slow!
    - Using hipMalloc for most or all memory allocations gives great binding and page size settings automatically
  - Running 3X+ slower than expected is usually due to bad binding, often due to missing flux run option --exclusive (-x)
    - More binding, page size, and running with flux guidance in later slides
- May be able to share a single GPU (or virtual GPU) with up to 4 MPI tasks with caveats (and luck)
  - 2X hardware-based GPU sharing usually works by default, env variables can often allow efficient 4X GPU sharing
    - If you need to share GPUs (i.e., UQ of tiny runs), reach out for more details on what to try (not covered more in this overview)
  - Performance falls off cliff if GPU falls back to software-based context switching of GPU
    - May fall off performance cliff for other reasons. 10% or less overhead expected for working-well cases.

SPX mode and the new TPX / CPX GPU Modes

- SPX mode (1 big GPU per socket), 4 GPUs/node is the default mode
  - Combines 6 GPU chiplets and uses one chiplet's scheduling hardware to drive GPU
  - Maximizes memory available per GPU (~120 GBs) and very powerful GPUs
  - Only mode is MPI currently certified by HPE to work (some hangs in other modes)
- CPX mode (6 virtual GPUs per socket, 1 per chiplet) yields 24 GPUs/node
  - flux alloc … --amd-gpumode=CPX
  - Every virtual GPU uses own chiplet's scheduling hardware but HBM spread across 4 stacks
  - Reports for 15-50% performance gains for small inputs that fit in tiny memory/GPU
    - SC25 study found better L2 cache hit rate due to higher data locality to be cause of performance gains
- TPX mode (3 virtual GPUs per socket) yields 12 GPUs/node
  - flux alloc … --amd-gpumode= TPX
  - Original focus but CPX mode appears to yield most performance benefit

Maximizing Application Performance under the Flux Scheduler

- Pro tip: Always use **--exclusive** (or **-x**) with flux run and don't specify other constraints
  - Only specify --nodes=# (or -N #) and --ntasks=# (or -n #) or --tasks-per-node=#
  - Do **NOT** explicitly specify --cores-per-task=# (or -c #) or --gpus-per-task=# (or --g #) for HPC runs
    - Those -c and -g options are for packing nodes for UQ or regression testing, may yield poor performing bindings
  - **--exclusive** (or **-x**) tells flux to optimally divide node resources between tasks using mpibind
    - In general, need 4 (or multiple of 4) tasks per node for optimal performance due to 4 sockets
    - -o mpibind=verbose tells mpibind to print bindings (.e.g.: mpibind: task 3 nths 21 gpus 3 cpus 73-79,81-87,89-95)
    - mpibind sets env variable ROCR_VISIBLE_DEVICES=3 in order to map GPU 0 request by app to GPU 3
- **84** (of 96) cores dedicated to user processes, 12 cores reserved for system and lustre
  - Those 84 cores to bind to currently specified by MPIBIND_RESTRICT (must be set to scale well):
    - **MPIBIND_RESTRICT**=1-7,9-15,17-23,25-31,33-39,41-47,49-55,57-63,65-71,73-79,81-87,89-95,97-103,105-111,113-119,121-127,129-135,137-143,145-151,153-159,161-167,169-175,177-183,185-191
    - To actually be able to use all 96 cores, unset MPIBIND_RESTRICT (but will have a lot of noise at scale)
  - New flux allocation-based mechanism to totally hide system cores by default in the works!
    - Having flux hide the system cores at allocation time restores expected flux behavior to some flux options
- The 'srun' wrapper for flux automatically adds --exclusive for you
  - srun -N 4 --ntasks-per-node=4 …

Always Allocate Interactive Compute Nodes for Compiling, Testing, and Debugging

- Always allocate a pdebug (or pbatch or pdev) compute node for compiling and testing your code
  - Please do not run big compiles or application runs on the login nodes!
    - Crashing or OOMing login nodes can kill all running jobs and app runs can get login node GPUs into bad states
  - Other Tri-lab supercomputer centers have the opposite policy, but this is how our center was designed
    - We understand that this may be the different than other supercomputer sites policies
- The 'pdebug' pool is only for compiling/testing/debugging, not production work
  - Use as few nodes as feasible (no more than half total) and do not block queue with large requests or chains
    - Use 'pbatch' allocations if need to debug or test large node count jobs or long running jobs
- Please do not game the system
  - We rely on social contracts and good neighbor polices not strong technical controls
    - Continuing to abuse login nodes, file systems, batch queues, etc., after being warned not to has gotten users banned
  - Hogging pdebug nodes with obvious production work most common form of abuse, do not abuse pdebug!
    - We know those idle nodes look tempting, but you are preventing others from getting their work done
- Need help or system seems hung, contact LC Hotline lc-hotline@llnl.gov or 925-422-4531
  - Please feel free to ask how you get can something done "the right way"

How to avoid “bad” nodes for your code

- What to do if some nodes appear problematic for your application?
  - Please(!) report it to the LC Hotline and tell us the symptoms so we can see if it needs to be replaced
  - Tell flux to NOT schedule your allocations on those node(s) with --requires=-host:name1,name2,…
    - Note the - before the -host. This indicates do not use those hosts.
- Batch and interactive examples if tuolumne1005,tuolumne1007 in pdebug appear bad
  - flux batch --queue=pdebug --requires=-host:tuolumne1005,tuolumne1007
  - flux --parent alloc --nodes=2 --queue=pdebug --time-limit=1h --requires=-host:tuolumne1005,tuolumne1007
- You can also request specific nodes (=host, no - before host) but please use sparingly
  - Will delay launching of your allocation and can be hard to tell why delay in flux queries
  - These constraints puts higher load on flux scheduler, so not want queue flooded with these request
  - You must specify exactly all the nodes desired to run on with --requires=host:name1,name2,…
    - flux --parent alloc --nodes=2 --queue=pdebug --time-limit=1h --requires=host:tuolumne1006,tuolumne1008
- Currently no way to specify nodes to avoid or use with salloc/sbatch “slurm” wrappers

How to choose between CC, crayCC, and "LC" -magic compiler wrappers

- cc/CC/ftn interface provides "CRAY" magic with everything specified by Cray module files
  - Automatically add -lxpmem, -lmpi_gtl_hsa, libsci, etc. based on modules loaded
  - Useful for those that value and are familiar with the traditional cray interface
  - Best practice: Have same Cray and rocm modules loaded when compiling and running executable
- New craycc/crayCC/crayftn and mpicc/mpicxx/mpifort provides module-less Cray environment
  - Common interface but YOU must link in -lxpmem, -lmpi_gtl_hsa, libsci, etc. explicitly for best performance
  - Enables cmake, autotools, spack in cray environment but need to add extra link options for performance
  - Best practice: Add RPATHs to your all your libraries for consistent running of executable
- The -magic wrappers provide "LC" magic where the environment is totally ignored
  - Load compiler with -magic extension (cce/20.0.0a-magic or rocmcc/6.4.3-magic) to get LC magic
    - mpicc/mpicxx/mpifort automatically switch to magic versions with -magic compiler loaded
  - Auto adds RPATHs to compilers and rocm and adds -lxpmem -lmpi_gtl_hsa (but not libsci or Cray libraries)
  - Best practice: Use full path to compiler in build system (should have -magic in path)
    - CXX= /usr/tce/packages/cray-mpich/cray-mpich-9.0.1-rocmcc-6.4.3-magic/bin/mpicxx
- NOTE: Beta compiler/mpi/rocm versions are deleted once release versions become available

C++ HIP compiler examples, no Fortran HIP support (mi300a's architecture name is gfx942)

- AMD HIP compilation (with LC -magic)
  - ml rocmcc/6.4.3-magic cray-mpich/9.0.1
  - mpiamdclang++ -D__HIP_PLATFORM_AMD__ -I/opt/rocm-6.4.3/include -O3 -g --offload-arch=gfx942 -std=c++11 --rocm-path=/opt/rocm-6.4.3 -x hip -mllvm -amdgpu-early-inline-all=true -mllvm -amdgpu-function-calls=false -fhip-new-launch-api --driver-mode=g++ rush_larsen_gpu_hip_mpi.cc -o rush_larsen_gpu_hip_mpi
  - /usr/tce/packages/cray-mpich/cray-mpich-9.0.1-rocmcc-6.4.3-magic/bin/mpiamdclang++

- CCE HIP compilation (with LC -magic)
  - ml cce/20.0.0a-magic cray-mpich/9.0.1
  - mpicrayCC -D__HIP_PLATFORM_AMD__ -I/opt/rocm-6.4.3/include -O3 -g --cuda-gpu-arch=gfx942 -std=c++11 --rocm-path=/opt/rocm-6.4.3 -x hip rush_larsen_gpu_hip_mpi.cc -o rush_larsen_gpu_hip_mpi
  - /usr/tce/packages/cray-mpich/cray-mpich-9.0.1-cce-20.0.0a-magic/bin/mpicrayCC

OpenMP GPU compiler examples, C++ and Fortran (mi300a's architecture name is gfx942)

- AMD C++ OpenMP Offloading
  - ml rocmcc/6.4.3-magic cray-mpich/9.0.1
  - mpiamdclang++ -O3 -g -fopenmp -fopenmp-targets=amdgcn-amd-amdhsa -Xopenmp-target=amdgcn-amd-amdhsa -march=gfx942 rush_larsen_gpu_omp_mpi.cc -o rush_larsen_gpu_omp_mpi
  - /usr/tce/packages/cray-mpich/cray-mpich-9.0.1-rocmcc-6.4.3-magic/bin/mpiamdclang++
- CCE Fortran OpenMP Offloading
  - ml cce/20.0.0a-magic cray-mpich/9.0.1
  - mpicrayftn -O3 -g -fopenmp -haccel=amd_gfx942 rush_larsen_gpu_omp_mpi_fort.F90 -o rush_larsen_gpu_omp_mpi_fort
  - /usr/tce/packages/cray-mpich/cray-mpich-9.0.1-cce-20.0.0a-magic/bin/mpicrayftn
- Mixed AMD C++ and Cray Fortran magic module
  - ml rocmcc/6.4.3-cce-20.0.0a-magic

Turning on GPU-aware MPI and using xpmem can significantly increase MPI performance

- **Starting with cray-mpich/8.1.30 (July 2024), -magic compilers automatically add libraries**
  - **-lxpmem -L/opt/cray/pe/mpich/9.0.1/gtl/lib -lmpi_gtl_hsa -Wl,-rpath,/opt/cray/pe/mpich/9.0.1/gtl/lib**
  - **-lxpmem doubles bandwidth on node using cpu-level MPI**
  - **Can turn off with --no-xpmem and/or --no-gtl if suspect libraries causing issues**
- **If not using -magic compilers, recommend add to link line (here for cray-mpich/9.0.1)**
  - **-lxpmem -L/opt/cray/pe/mpich/9.0.1/gtl/lib -lmpi_gtl_hsa -Wl,-rpath,/opt/cray/pe/mpich/9.0.1/gtl/lib**
- **Turn on GPU-aware MPI by setting env variable MPICH_GPU_SUPPORT_ENABLED=1**
  - **Must have GTL library linked in for this to work (punts otherwise)**
  - **Yields another 2X+ on-node memory bandwidth increase over xpmem**
  - **However freeing GPU MPI buffers can cause memory growth (see later slide for mitigations)**
- **ABI used by GTL changed with rocm/6.0 and with rocm/7.0**
  - **rocm/5.7.1 requires use of older cray-mpich/8.1.27**
  - **ABI stable after rocm/6.0 for rocm/6.* but enhanced for cray-mpich/8.1.33 in rocm/6.4.***
  - **Use cray-mpich/9.0.1 with rocm/6.4.3 and cray-mpich/9.1.0 with rocm/7.2.1 (or 7.2.0)**

Why don't we load the rocm module by default?

- **Having a ROCM_PATH set to a different rocm than your application uses can cause errors!**
  - **AMD designed ROCM_PATH to allow plugging in debug rocm .so files at runtime**
    - **So setting ROCM_PATH to a different rocm can cause your application to mix two different rocm library sets**
  - **Everyone else (cce, cmake, etc) uses ROCM_PATH to find which rocm you are using**
  - **Having a conflicting ROCM_PATH can break you both at compile time and run time!**
- **The -magic compilers add link line options to make executable ignore ROCM_PATH**
  - **-L/opt/rocm-6.4.3/lib -Wl,-rpath,/opt/rocm-6.4.3/lib -lamdhip64 -lhsakmt -lhsa-runtime64 -lamd_comgr**
  - **We recommend you add something like the above to your link line (for the correct rocm, of course)**
  - **This is key for published executables where users may have a different rocm loaded**
- **Major ABI change in rocm/7.0.0 that MPI & CCE does not support until March 2026**
  - **cce/21.0.0 and cray-mpich/9.1.0 from March 2026 requires rocm/7 (7.2.1 recommended)**

PyTorch, TensorFlow, or using RCCL, need to use AWS Plugin and RCCL module to set env vars

- RCCL (ML-optimized comm) using TCP sockets is very slow on CORAL2's network
  - AWS's open-source RCCL Plugin (librccl-net.so) uses libfabric to greatly accelerates RCCL on slingshot
  - Currently you need to build it for your application if using RCCL/PyTorch/TensorFlow (see below)
    - LD_LIBRARY_PATH needs to point to the directory that contains the plugin library
  - You MUST also currently either load a rccl module or set env variables to make compatible with slingshot
    - 'ml rccl/working-env' currently sets one env variable to prevent libfabric conflict between rccl and MPI that causes hangs
    - 'ml rccl/fast-env-slows-mpi` currently sets 8 tuning parameters that accelerates rccl at the expense of MPI performance
    - Planned future upgrade of using 'kdreg2' with libfabric is expected to eliminate the need for loading rccl/working-env
- Instructions for building and using AWS RCCL plugin available
  - A very useful sample build script: /collab/usr/global/tools/rccl/toss_4_x86_64_ib_cray/rocm-6.2.1/buildme
  - New pyTorch launcher tool: https://github.com/LBANN/HPC-launcher
  - Example of build, testing, and verifying RCCL on Tioga (remember to use --exclusive or -x on Tuolumne)
    - https://lc.llnl.gov/confluence/display/LC/2023/06/02/RCCL+performance+on+Tioga#RCCLperformanceonTioga-InstalltheAWS-OFI-RCCLplugin
  - PyTorch specific instructions:
    - https://lc.llnl.gov/confluence/display/LC/Distributed+PyTorch+on+CORAL+2+systems#DistributedPyTorchonCORAL2systems-Extra:InstallAWS-OFI-RCCLplugin

Applications must reduce file system load to something manageable! Tools like Spindle can help!

- Linux and python .so search algorithm at scale causes denial-of-service attack on file systems
  - Without mitigation, can take hours to start executable and makes LC filesystems unresponsive for everyone
  - Rapid small-scale runs of many python launches can cause same file systems issues
    - 4 nodes running 10 executable invocations per core per second used 99.9% of file system resources until mitigated
  - Please do not initialized CONDA, python, or spack in your .bashrc file. Can cause huge file system load!
  - Please build for just the one GPU. Building for both mi300a and mi250x can double file sizes.
- Main symptom is slow launch times (or sysadmins killing your job and contacting you)
  - With Spindle, full El Cap scale launches typically take less than 2 minutes
  - If it is taking more than 2 minutes to launch your application, please reach out to us!
- Spindle uses advanced algorithms at launch time to mitigate file system load (caches directory contents)
  - Spindle in 'medium' mode is invoked flux run by default on Elcap and Tuolumne but not any other machines
  - Tuning is sometimes needed to make compatible with new applications (breaks POSIX standard to reduce load)
    - If small scale testing with SPINDLE_FLUXOPT=disable or fastload flux run reveals functionality change (i.e., fixes issue) contact us for help!
    - This is the first thing to try if get unexpected segfaults or get errors that files or libraries cannot be found
  - But you must use another method to mitigate file system load if you turn spindle off at scale! Contact us for help!
    - Options include 'fastload' (previous automatic mitigation technique) and 'build_libcache' and other Spindle modes.
- NOTE: Putting large executables in lustre may also be helpful even with Spindle at scale

Where (and where not) to write and save data

- Don't write data to workspace (/usr/WS*) or home (/g/g*) directories!
  - Write your data to /p/lustre# (/p/lustre4 on elcap and /p/lustre5 on tuolumne)
  - Very easy to crush NFS servers with parallel reads and writes and they have limited quotas
    - Hammering the small quotas on those systems is also very hard on the systems
- Lustre supports large default quotas and more can be requested
  - Elcap 100TB (space) /10M (inodes) Tier 1 quota (default, form for requesting more)
  - Tuolumne 50TB (space) /5M (inodes) Tier 1 quota (default, form for requesting more)
  - Not backed up, so don't put only copy of key files on lustre
- Rabbits provide node-local temporary ssd storage (deleted when job completes)
  - /l/ssd (200GB current size). Documentation provides rabbit options and data movement examples
  - https://flux-framework.readthedocs.io/projects/flux-coral2/en/latest/guide/rabbit.html
    - #FLUX: #DW jobdw type=xfs capacity=1TiB name=xfsproject sets $DW_JOB_xfsproject to 1TiB local space
- Tape archive (htar, ftp storage) can provide long storage for key data
  - 300TB default quota for tape storage currently

Early access testing allowed us to identify and fix key issues before production use

- OOM killer priorities dialed in by flux since mid May 2025
  - Running out of memory now kills user tasks first and no longer breaking the nodes (most of the time)
    - GPU memory usage is not currently visible to kernel, so flux now paints user tasks as first to kill
  - No longer need to sysadmins to reboot and bring back up 1000s of nodes a day
  - Longer term work on making GPU memory visible to OS and cgroups continues
- Unrecoverable Memory Faults now successfully causes retirement of ‘suspect’ memory
  - Until mid May 2025, complex interactions prevented ‘suspect’ memory from actually being retired
    - This continued reuse of ‘suspect’ memory dramatically negatively impacted mean time to failure
  - Each memory block can now only get one unrecoverable error before being taken out of use
  - Too much retired memory triggers node memory replacement to prevent noticeable shrinkage
- GPU-aware MPI unexpected memory growths tracked down
  - Software fixes for most issues in cray-mpich 8.1.33, 9.0.1, and 9.1.0 (next slide)
- These are examples of why early access shakeout periods key for stable production usage

Solutions delivered in cray-mpich/8.1.33, 9.0.1 and 9.1.0 (9.1.0 has more fixes but requires rocm/7.*)

- rocm/6.4.3 allows execution of applications without read-bit set
  - One non-readable application currently also requires ROCM_PATH=/opt/rocm-6.4.3
- cray-mpich/9.0.1 supports new IPC signal cache management algorithm
  - Freeing GPU MPI buffers can cause memory growth with current IPC cache algorithm
    - OS cannot reuse freed memory because other GPUs nave mapped it into their address space
    - Workarounds: Don't free GPU mpi buffers or reuse freed memory via memory pools not the OS
  - New algorithm 'signals' other GPUs on same node to unmap memory so can be freed
    - Disabled by default since new. Must set GTL_DISABLE_HSA_IPC_SIGNAL_CACHE=0 to enable
    - Requires rocm/6.4.* since relies on new rocm signal ABI implemented starting in rocm/6.4.0
  - Strongly recommend rocm/6.4.3 with cray-mpich/9.0.1
- cce/20.0.0 supports rocm/6.4.* natively
  - Most cce/19.0.0 + rocm/6.4 issues tied to setting DEBUG_HIP_7_PREVIEW=1

hipMalloc, HSA_XNACK=1, and 2mb pages options

- GPU performs best with 2mb pages and requires pages touched by GPU to be mapped into GPU
  - Use hipMalloc when possible to allocate memory, get 2mb pages mapped in GPU that can also be used on CPU
    - Doing this will greatly simplify your life and maximize GPU performance
    - Do not need any of the more complicated options below if use hipMalloc for all GPU memory allocations
- CPUs and GPUs share memory but cpu-based allocators not auto mapped to GPU
  - setting env variable HSA_XNACK=1 will page-fault in CPU pages into GPU (with slight overhead)
  - CPU default 4k page size will cause 15% or more performance overhead on GPU due to GPU TLB size
- Enabling transparent huge pages makes most > 2mb CPU allocations have 2mb pages
  - Must be enabled at compute node allocation time
    - flux alloc --setattr=thp=always -N1
    - salloc -N1 --thp=always
  - Recommended starting point if allocating memory on CPU to be used on GPU (need HSA_XNACK=1 also)
- Linking in -lhugetlbfs and enabling in allocation can put < 2mb CPU allocations in 2mb pages
  - Must be enabled at compute node allocation time and can coexist with thp
    - flux --parent alloc --setattr=hugepages=460G --setattr=thp=always -N 1
    - salloc --hugepages=460G --thp=always -N 1
    - Must also set env variable HUGETLB_MORECORE=yes (and need HSA_XNACK=1 also)
    - Some users have reported a limit of 430G yields less error messages and we may have a new version coming soon to make things better

Use on_hang_stat_and_kill to handle unexpected hangs and debug them (on_hang_kill, on_hang_stat also available)

- Usage: on_hang_stat_and_kill <options> <'files_to_watch'>

  Running with no arguments defaults to:
  on_hang_stat_and_kill --thresh 20.0 '{{stdout}} {{stderr}}'

  Runs stat-cl and flux cancel on all running jobs in allocation if <'files_to_watch'> are not updated for <thresh> minutes.

  'files_to_watch' defaults to '{{stdout}} {{stderr}}' and must be quoted!
  Supports custom mustache templates {{stdout}} {{output}} {{stderr}} {{error}}
  flux mustache templates {{id}} {{jobid}} {{name}} and
  shell globbing (i.e. '*{{id}}*.out') and watching directories (i.e. '.')

  Example usage inside flux batch script (auto backgrounds watchdog):
  on_hang_stat_and_kill <options_if_defaults_not_sufficient>
  flux run ... a.out ...

  Options (also accepts --thresh=value and - instead of --):
  --thresh minutes // Trigger if files not updated in <minutes> (20.0 default)
  --delay minutes // Delay first check (0.0 default) to allow for slow startup

- Backgrounds itself, so you do not need to (but ok if you do)
- Users with long startup but fast cycles may want something like: on_hang_stat_and_kill --delay=45.0 --thresh=5.0

Example flux batch script usage of on_hang_stat_and_kill (NOTE: Separate stderr file makes finding first error much easier!)

- example flux script: test16.flux (works in slurm batch script emulation also)

```
#!/bin/sh
# Example Flux batch script
#flux: --queue=pdev
#flux: --job-name=test16
#flux: --output='output_{{name}}.{{id}}.out'
#flux: --error='output_{{name}}.{{id}}.err'
#flux: -N 2              # Request 2 nodes
#flux: -t 60             # Wall time limit (minutes)

on_hang_stat_and_kill
flux run -x -N2 -n8 ./fake_sim 10 1 2
```

- Example submission: flux batch test16.flux
  f39NSjZ4dy6o
- View all five .dot files generated from stat-cl with stat-view: stat-view .../stat_results/app_name.jobid.dumpid/*.dot
- We currently log all on_hang script hang triggers (no sensitive info saved)

Example on_hang_stat_and_kill --thresh=.2 output for batch
If no within 0.2 minutes, stats and kills

- cat output_test16.f39NSjZ4dy6o.out

  on_hang_stat_and_kill 2025-11-06 10:04:54: Started watching 'output_test16.f39NSjZ4dy6o.out output_test16.f39NSjZ4dy6o.err' in f39NSjZ4dy6o 0.003h idle trigger

  Fake_sim start 10 cycles 1 seconds per cycle hang 2000 seconds in cycle 2 at Thu 06 Nov 2025 10:04:57
  Cycle 1 sleeping 1 seconds at Thu 06 Nov 2025 10:04:57
  Cycle 2 HANGING for 2000 seconds at Thu 06 Nov 2025 10:04:58
  HANG DETECTED 2025-11-06 10:05:14 on_hang_stat_and_kill rzadams f39NSjZ4dy6o at 0.009h 2 nodes 0.157h remains 0.003h idle output_test16.f39NSjZ4dy6o.out

  on_hang_stat_and_kill 2025-11-06 10:05:14: Running STAT on running jobs in allocation using 'stat-cl -magic all'
  Running stat-cl --jobid f39NSjZ4dy6o --attach on f38JoGST gyllen fake_sim R 8 2 19.77s rzadams[1076,1085] at Thu Nov 6 10:05:15 PST 2025
  STAT started at 2025-11-06-10:05:16
  Results written to /usr/WS2/gyllen/debug/on_hang_trigger/stat_results/fake_sim.f39NSjZ4dy6o.0000
  DONE with stat-cl -magic, ran stat-cl on 1 job in f39NSjZ4dy6o Thu Nov 6 10:05:20 PST 2025

  on_hang_stat_and_kill 2025-11-06 10:05:20: Running flux cancel to KILL all running jobs in allocation
  on_hang_stat_and_kill 2025-11-06 10:05:20: Running flux cancel f38JoGST

  on_hang_stat_and_kill 2025-11-06 10:05:20: DONE and exiting!

Updated stat-cl and stat-gui -magic with full flux TARGET (jobid/subid1/...) support deployed 10/30/25

- Run 'stat-cl -magic -h ' to see what new options -magic supports (flux help here):

```
Usage: stat-cl -magic <options> jobid|all          <-- passthru_args>
       stat-cl -magic <options> jobid/subjobid     <-- passthru_args>
       stat-cl -magic <options> subjobid           <-- passthru_args>
       stat-cl -magic <options> TARGET             <-- passthru_args>

<jobid> can refer to a flux allocation or a run inside the current allocation
<jobid/subjobid> specifies a specific subjobid in an specific allocation
<subjobid> will trigger depth 1 search of running allocations for subjobid
Any flux <TARGET> (jobid/subid1/subid2/...) of any depth can now be used
Use 'all' as a shortcut to run stat on all currently running jobs/allocations

--dir <dir>      // Puts stat_results in <dir> instead of cwd of each job
 -v | --verbose   // Print all the flux commands run
 -h | --help      // This usage message
 -- passthru_args // All args after -- are passed thru to stat-cl

Runs stat-cl on the jobid passed in, doing all the machinations
necessary for stat to work (instead of the user doing them).

If run on an flux allocation, runs 'flux proxy' and stats all running jobs.
In the allocation, does the flux job attach' to enable stat-cl.
Adds parent flux allocation jobid to stat_results directory name for tracking.
Use flux path-like TARGET (jobid/subid1/...) to specify exactly what you want.
```

Example stat-cl -magic jobid on an allocation

- flux jobs

```
JOBID          QUEUE   USER    NAME  ST NTASKS NNODES   TIME    INFO
f2Sod2T846kf pdebug  gyllen  flux   R      1      1   2.499m  tuolumne1024
```

- stat-cl -magic f2Sod2T846kf

```
Attaching to allocation f2Sod2T846kf with flux proxy --nohup to stat-cl all running jobs
Running stat-cl --jobid f2Sod2T846kf  --attach on ffsuCevK gyllen fake_sim R 4 1 1.197m tuolumne1024 at Wed
Oct 22 09:40:12 PDT 2025
STAT started at 2025-10-22-09:40:13
Results written to /usr/WS2/gyllen/debug/stat-jobid/stat_results/spindle_bootstrap.f2Sod2T846kf.0000

DONE with stat-cl -magic, ran stat-cl on 1 job in f2Sod2T846kf Wed Oct 22 09:40:15 PDT 2025
```

- Spindle replaces executable name with spindle_bootstrap in stat_results on ELCAP and TUO
- Use stat-view on all five .dot files: stat-view stat_results/spindle_bootstrap.f2Sod2T846kf.0000/*.dot

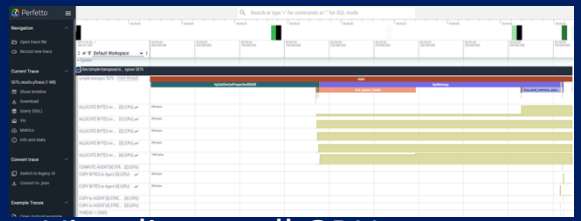
Performance Tools

Rocprofv3/rocprof-sys/rocprof-compute – Best for GPU deep dives, not scaling

ROCPROFV3 HSA_API SUMMARY:

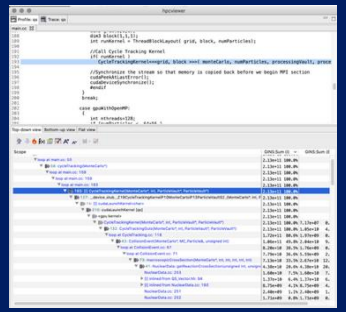
NAME	DOMAIN	CALLS	DURATION (nsec)	AVE
hsa_queue_create	HSA_API	4	28007621	
hsa_amd_memory_async_copy_on_engine	HSA_API	24	55617052	
hsa_amd_memory_pool_allocate	HSA_API	67	26428438	
hsa_amd_memory_pool_free	HSA_API	72	5176173	
hsa_executable_freeze	HSA_API	2	964125	
hsa_signal_wait_scacquire	HSA_API	26	853122	
hsa_executable_load_agent_code_object	HSA_API	2	616175	
hsa_amd_agents_allow_access	HSA_API	35	430680	

Print GPU metrics on command line

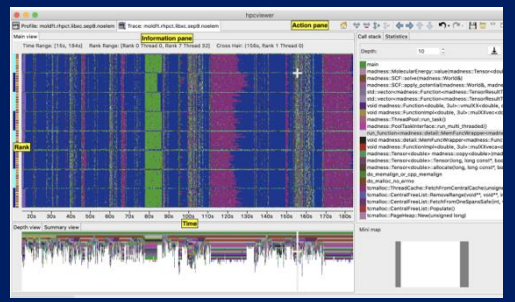


Visualize small GPU traces on with perfetto
https://lc.llnl.gov/perfetto or https://rzlc.llnl.gov/perfetto

HPCToolkit – Whole application profiling with good GPU support



Show CPU+GPU metrics on code regions



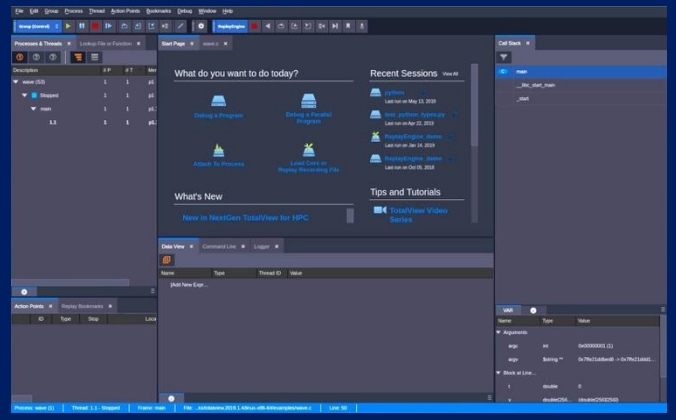
Visualize CPU+GPU traces across nodes

HPE Perftools (was CrayPAT) – HPC-focused profiling and tracing

Caliper – Application integrated profiling

Debugging Tools

TotalView – Visual HPC and GPU debugger



rocgdb – Serial GPU debugger from AMD based on GDB

gdb4hpc – Parallel GPU debugger from HPE based on GDB

ASAN – Compiler-based memory correctness tools

STAT – Core dump and stack trace analysis

Questions?


Lawrence Livermore National Laboratory
ENERGY
NNSA
Hewlett Packard Enterprise
AMD
Red Hat
EL CAPITAN
TUOLUMNE