
KubeCon
CloudNativeCon
North America 2024

KubeCon | CloudNativeCon

North America 2024

Resiliency for large-scale AI workloads

Abhijit Paithankar, Tech Lead Manager Nvidia
Arpit Singh, Senior Software Engineer Nvidia

Arpit Singh

Senior Software Engineer

NVIDIA

linkedin.com/in/arpitsardhana

Abhijit Paithankar

Tech Lead Manager

NVIDIA

linkedin.com/in/abhijit-paithankar

Poll

How many of you manage GPU clusters?

Agenda

1. Motivation for resilience in AI workloads
2. Understanding AI Workloads
3. Where do failures come from?
4. AI Infrastructure: Components, Bringup & Validation
5. Fault Tolerance - Building Blocks & Mechanisms
6. Monitoring
7. Propagation
8. Recovery strategies
9. Remediation
10. Scope for Enhancement

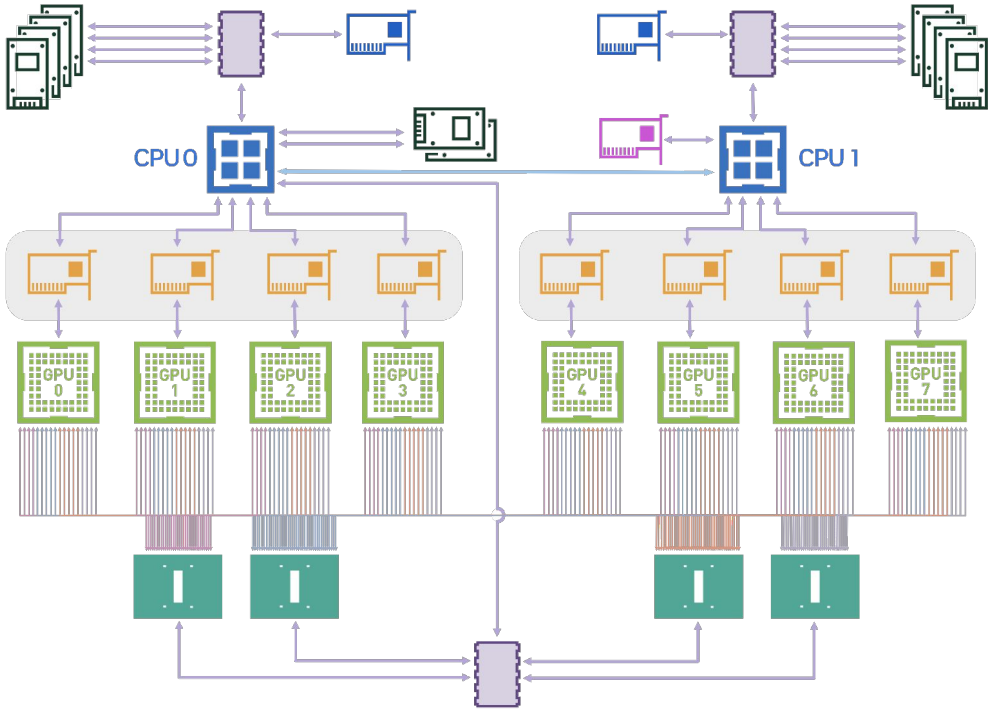
Agenda

1. Why resilience for AI training?
2. Understanding AI Workloads
3. Where do failures come from?
4. AI Infrastructure: Components, Bringup & Validation
5. Fault Tolerance - Building Blocks & Mechanisms
6. Monitoring
7. Propagation
8. Recovery strategies
9. Remediation
10. Scope for Enhancement

Why resilience?

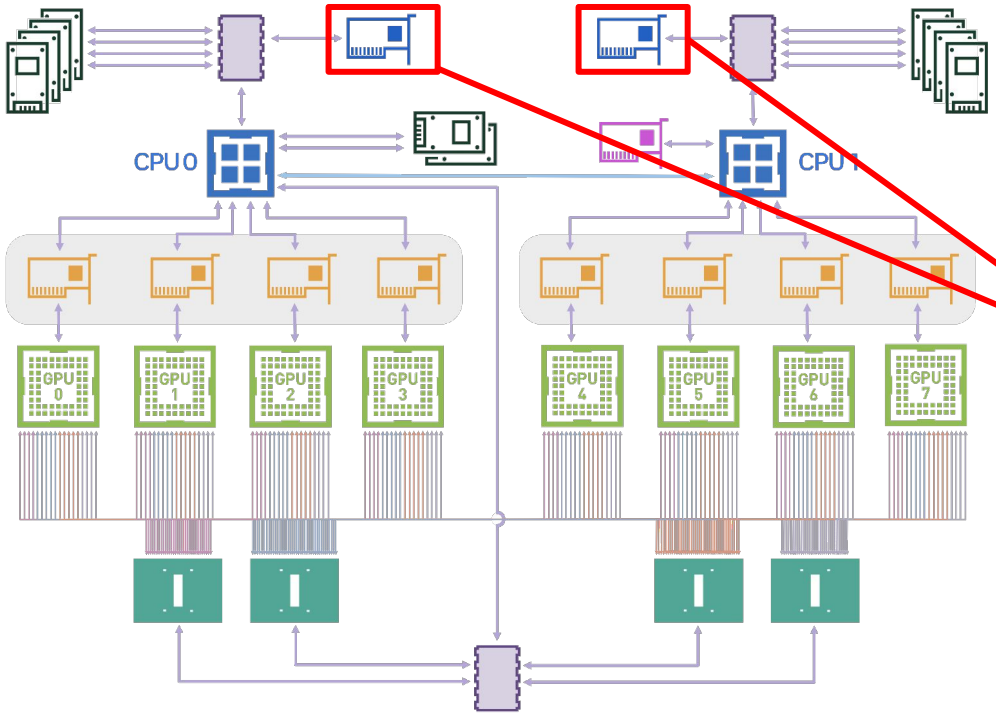
Complex compute hardware

Why resilience?



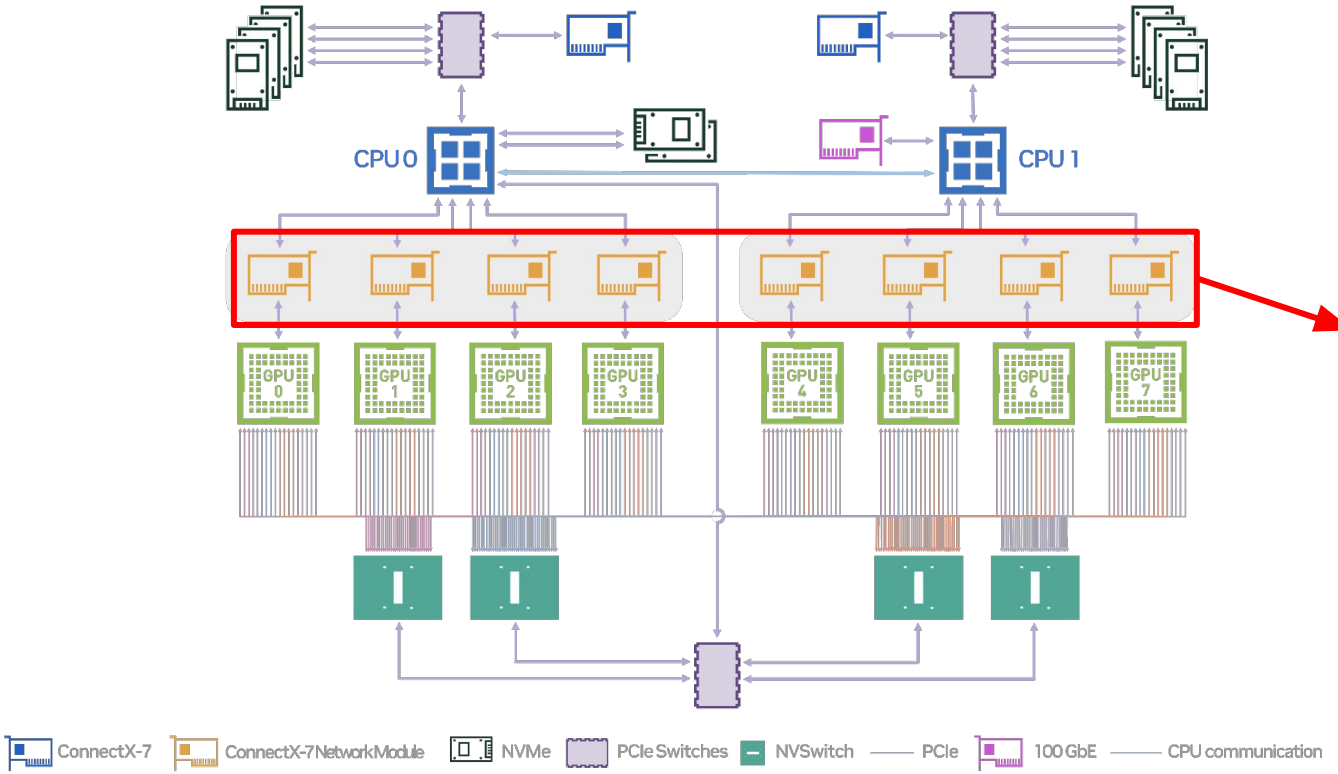
NVIDIA DGX H100/200 System Topology

Why resilience?



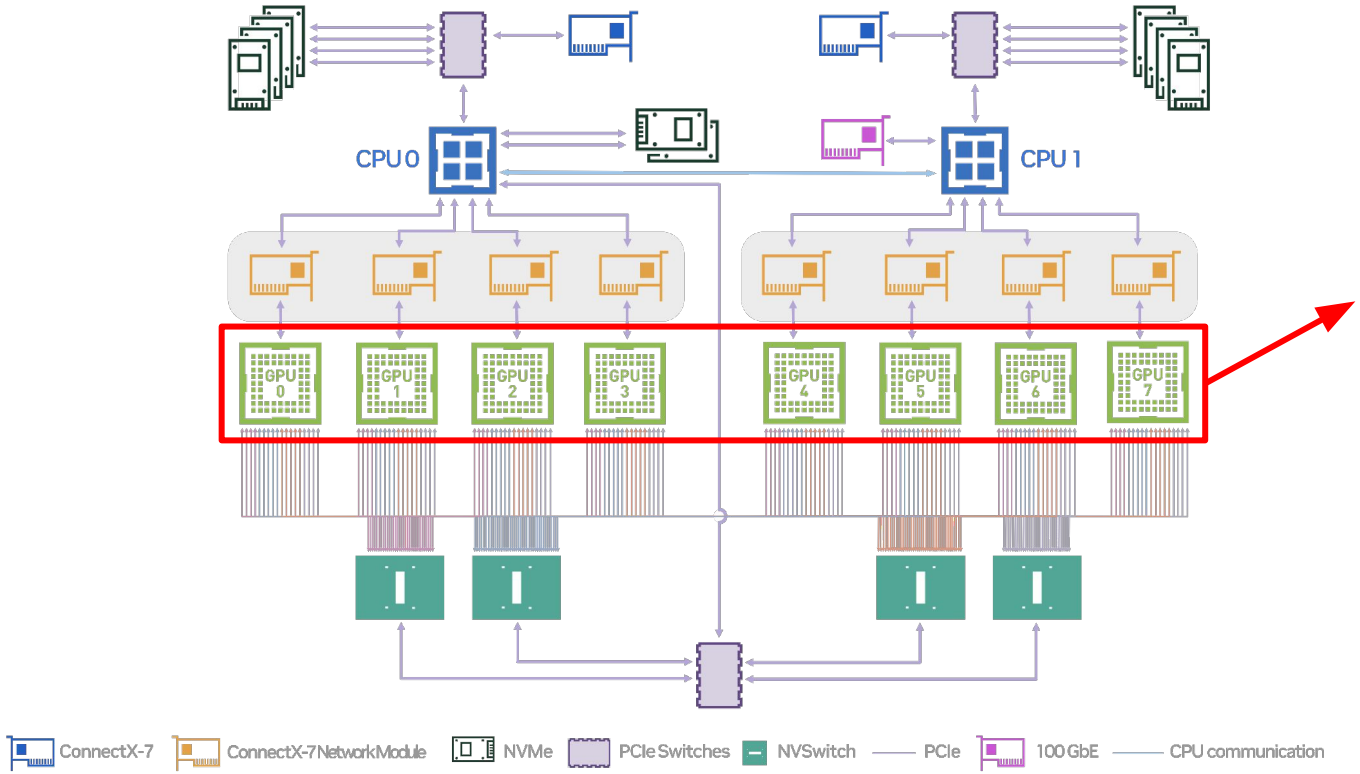
400Gb/s Ethernet cards connect to high performance parallel file system like **Lustre**

Why resilience?

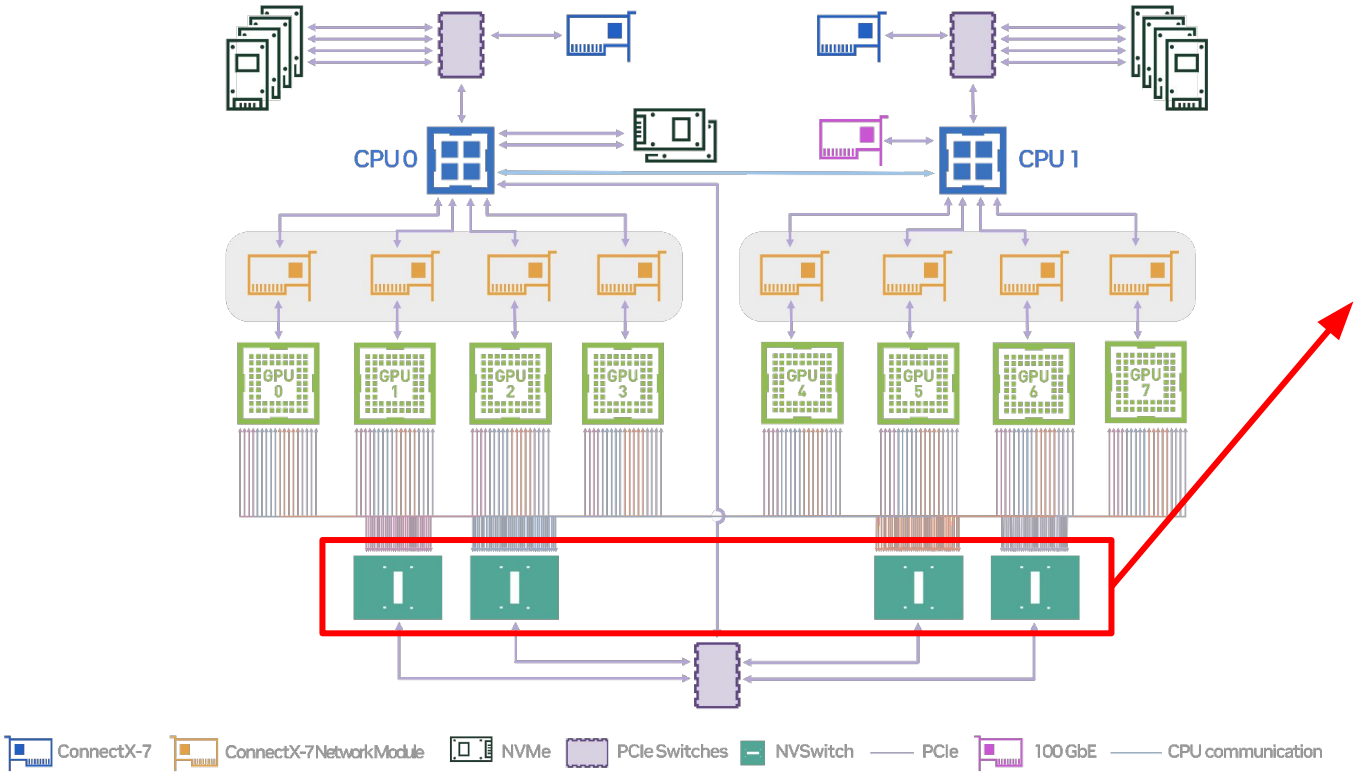


400Gb/s Infiniband cards to connect to **backend network**

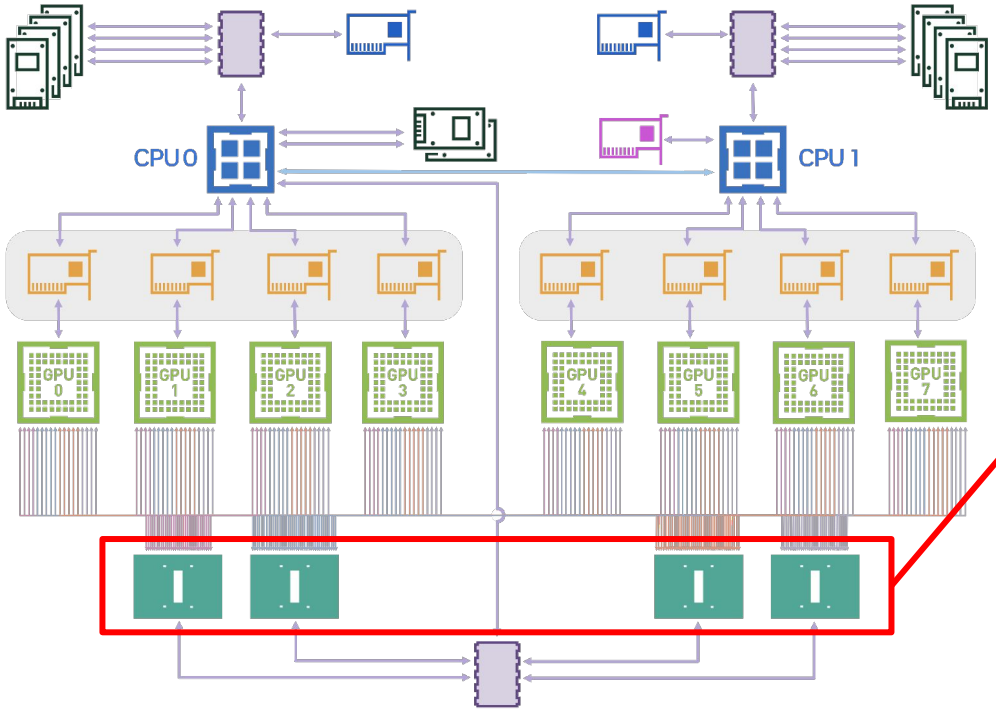
Why resilience?



Why resilience?



Why resilience?



NVSwitch for
900GB/s GPU-to-GPU

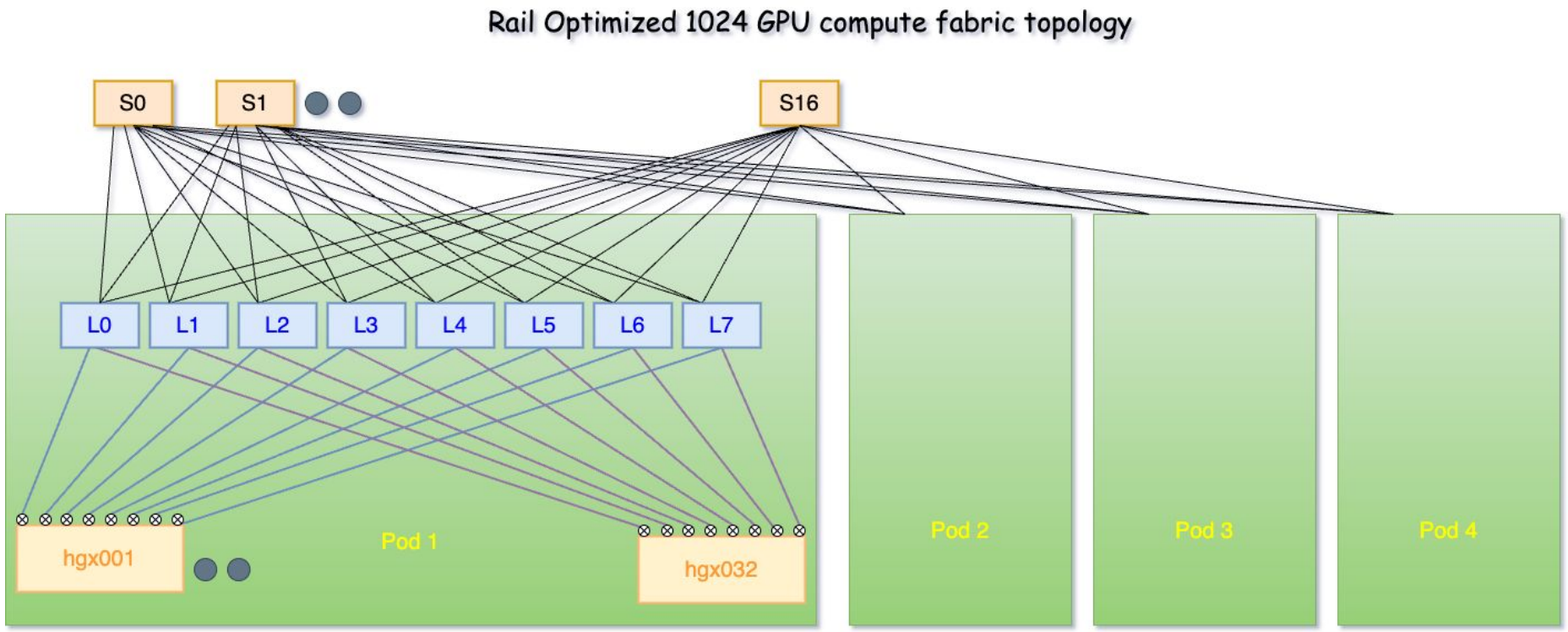
Yes, that's GigaBytes!

Why resilience?

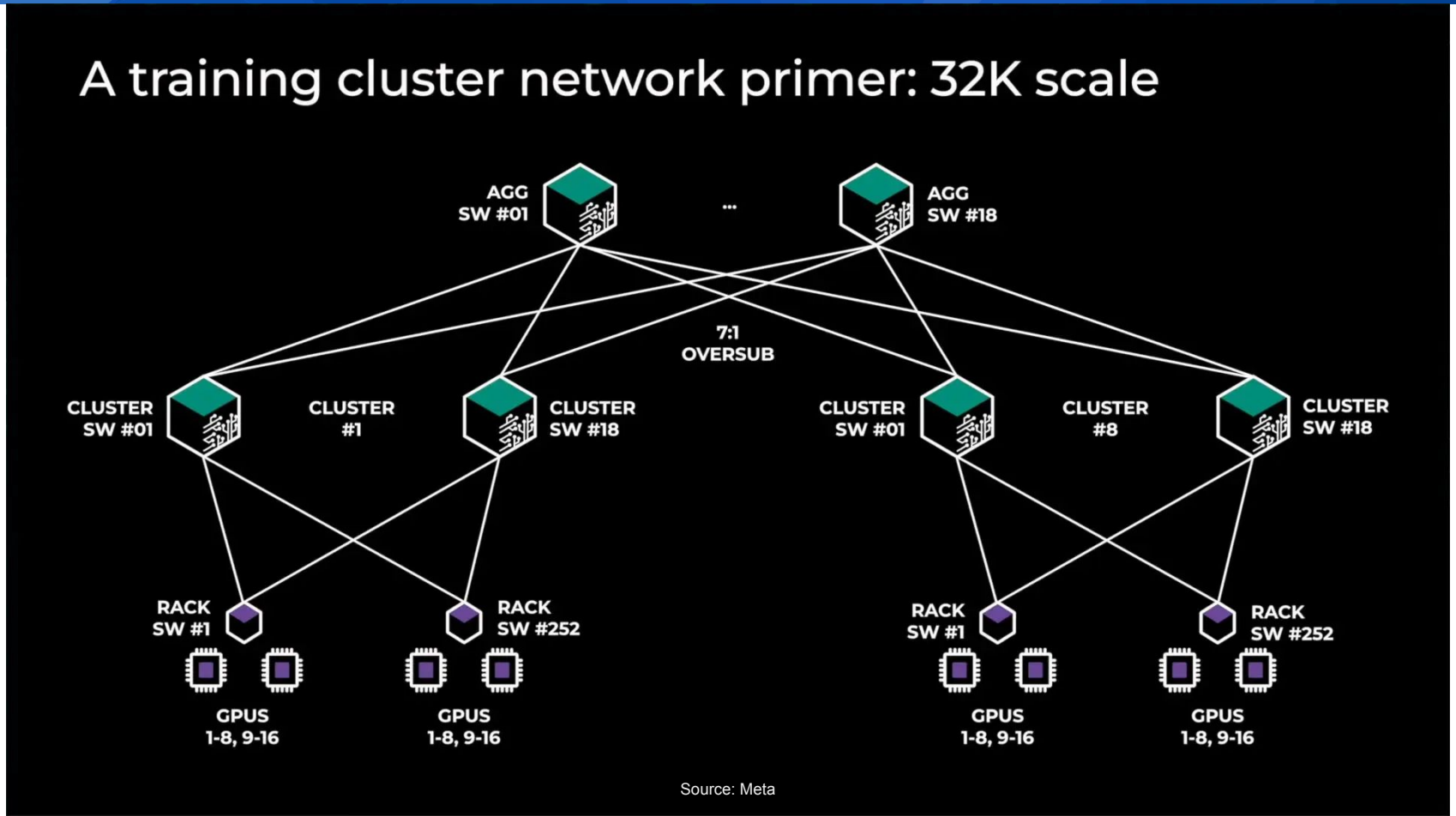
Large scale

Managing GPU clusters is way more difficult than managing CPU clusters

Why resilience?



Why resilience?

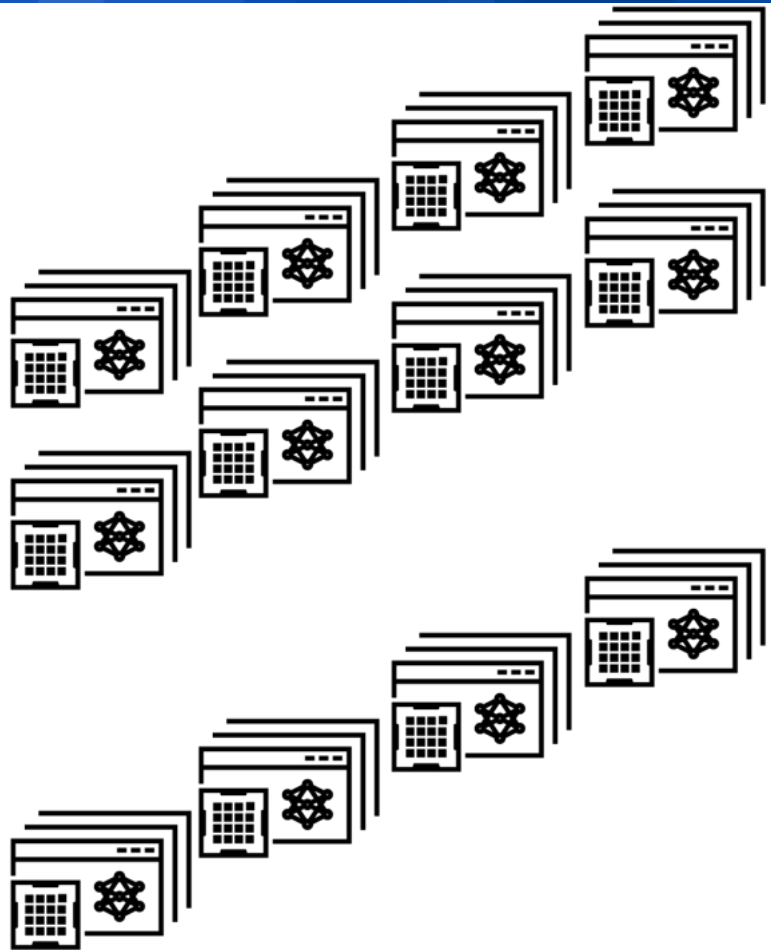


Why resilience?

Complex compute hardware

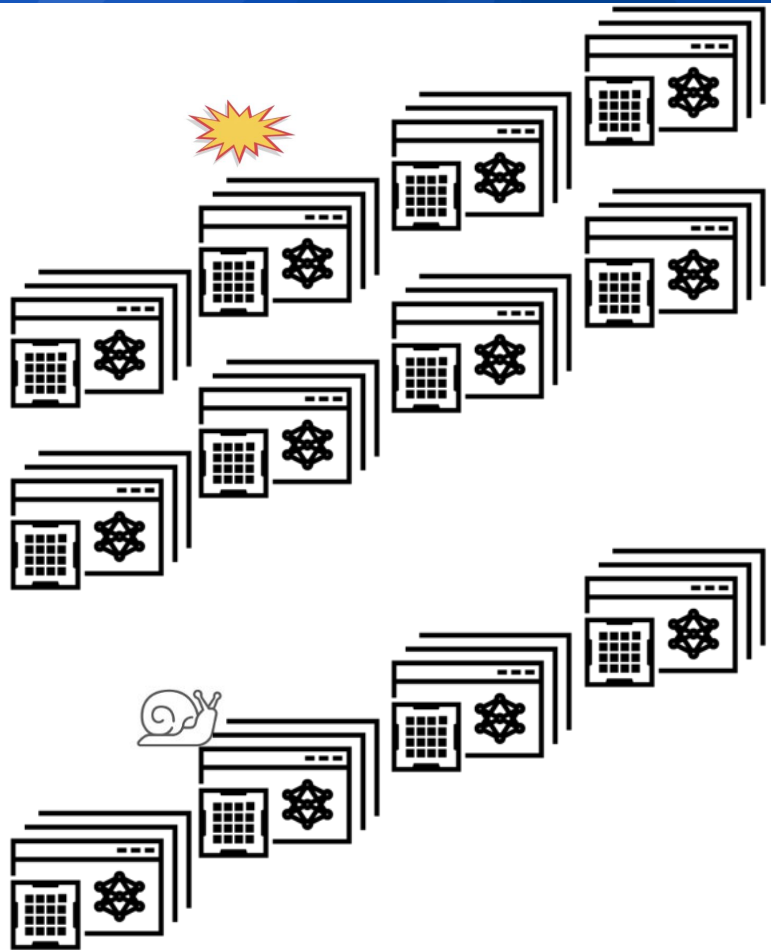
Large scale

The combined effect of these two is **synergistic**



Why resilience?

- Large scale training: Parallelized training jobs with 1K~100K GPUs
  - Billions to Trillions of parameter models
- Highly complex systems have multiple sources of failures:
  - Power, CPU, GPU, Memory, Network, Cable, Software, etc
- Some components may run at degraded performance, causing entire training to progress at slower rate
- For AI Training, failures are catastrophic due to
  - tightly coupled
  - synchronous workload
- Users expect “Fire and Forget” experience



Agenda

1. Motivation for resilience in AI workloads
2. AI workload characteristics
3. Where do failures come from?
4. AI Infrastructure: Components, Bringup & Validation
5. Fault Tolerance - Building Blocks & Mechanisms
6. Monitoring
7. Propagation
8. Recovery strategies
9. Remediation
10. Scope for Enhancement

AI workload characteristics

(Pre) Training

- Long running synchronous workloads
- High storage throughput for data read and write
- High traffic over backend network
- Large-scale, multi node and multi GPU
- Typically, a single training job is run on all nodes within the datacenter

Fine Tuning

- Short running synchronous workload < 24 hrs
- High traffic over backend network
- Frequent small writes
- Small scale, multi node and multi GPU
  - about 8 - 32 GPUs per job

Inference

- Request-response workload
- Batch input, streaming output
- Frontend network latency sensitive
- Employs load balancing and auto scaling
- Concurrent multiple instances
- Blast radius is limited to individual instances when failures occur

Simulation

- Long running medium scale workload
- High storage requirements
- Run at medium scale over multiple nodes and multiple GPUs
- Mixed mode GPU usage
  - Graphics mode
  - Compute mode

Agenda

1. Motivation for resilience in AI workloads
2. Understanding AI Workloads
3. Where do failures come from?
4. AI Infrastructure: Components, Bringup & Validation
5. Fault Tolerance - Building Blocks & Mechanisms
6. Monitoring
7. Propagation
8. Recovery strategies
9. Remediation
10. Scope for Enhancement

Where do failures come from?

Hard failures (require node reboot)

- Network/Fabric errors
  - NVLINK errors
  - External switch errors/failures
- GPU errors
  - ECC Uncorrectable
  - “Disappearing GPU”
  - Kernel driver bugs leading to wedged kernel/app
- System
  - PCIe related
  - Intermittent storage slowness or hang

Transient failures (recoverable)

- Network/Fabric errors
  - IB link flap
  - IB bandwidth degradation
- GPU errors
  - Thermal slowdowns
  - CUDA Errors
  - NaNs
- Remote storage performance degradation
- Application
  - Out Of Memory
  - Application crash/bugs

Where do failures come from?

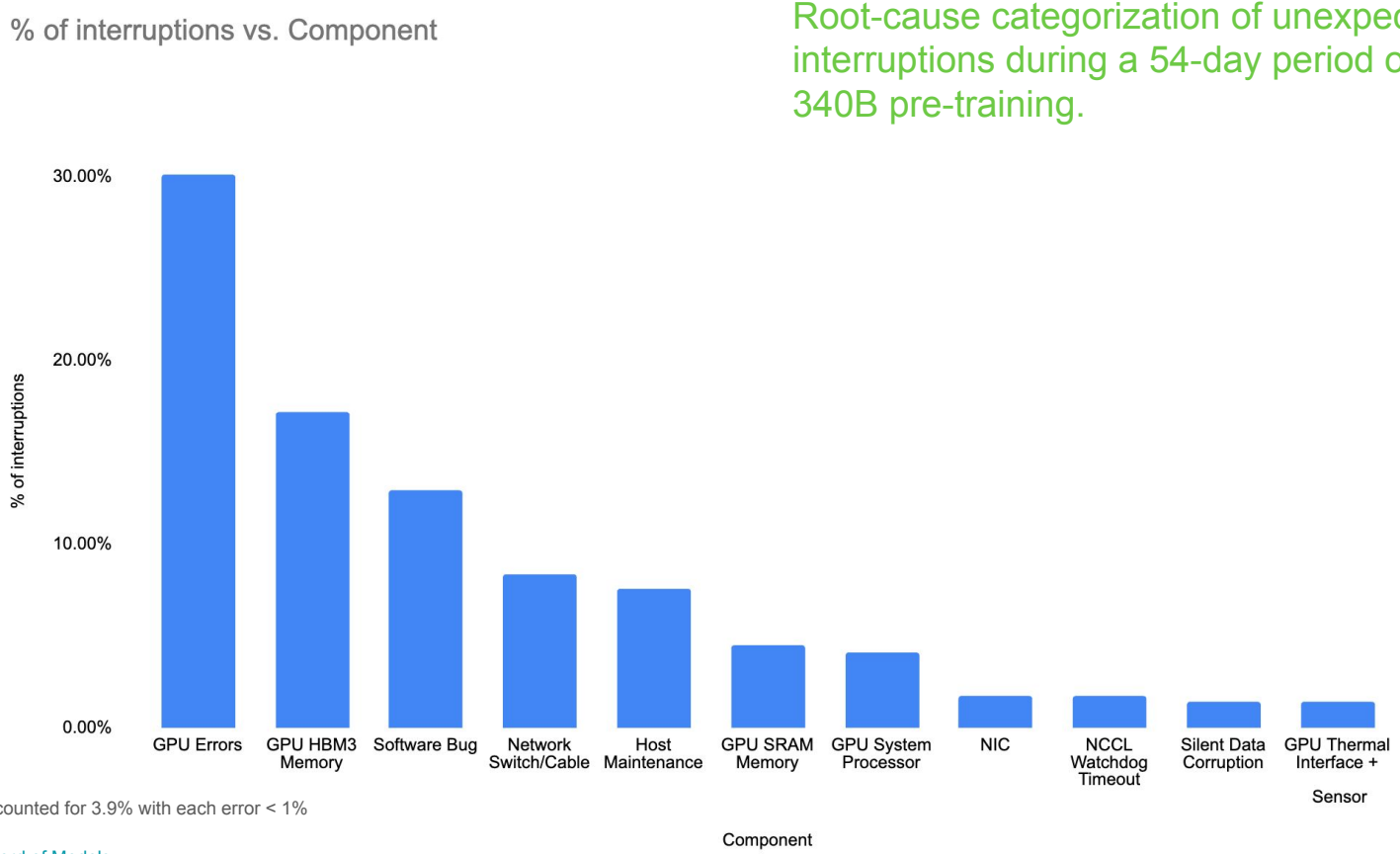
Other failures

- A faulty fan on a single node that was part of a 400-node cluster slowed down all jobs it was a part of

- An IB switch firmware bug causing performance degradation of backend network making the cluster unusable for multi node jobs. This was part of a routine firmware upgrade for the switch

Distribution of interruptions due to failures

Root-cause categorization of unexpected interruptions during a 54-day period of Llama 340B pre-training.

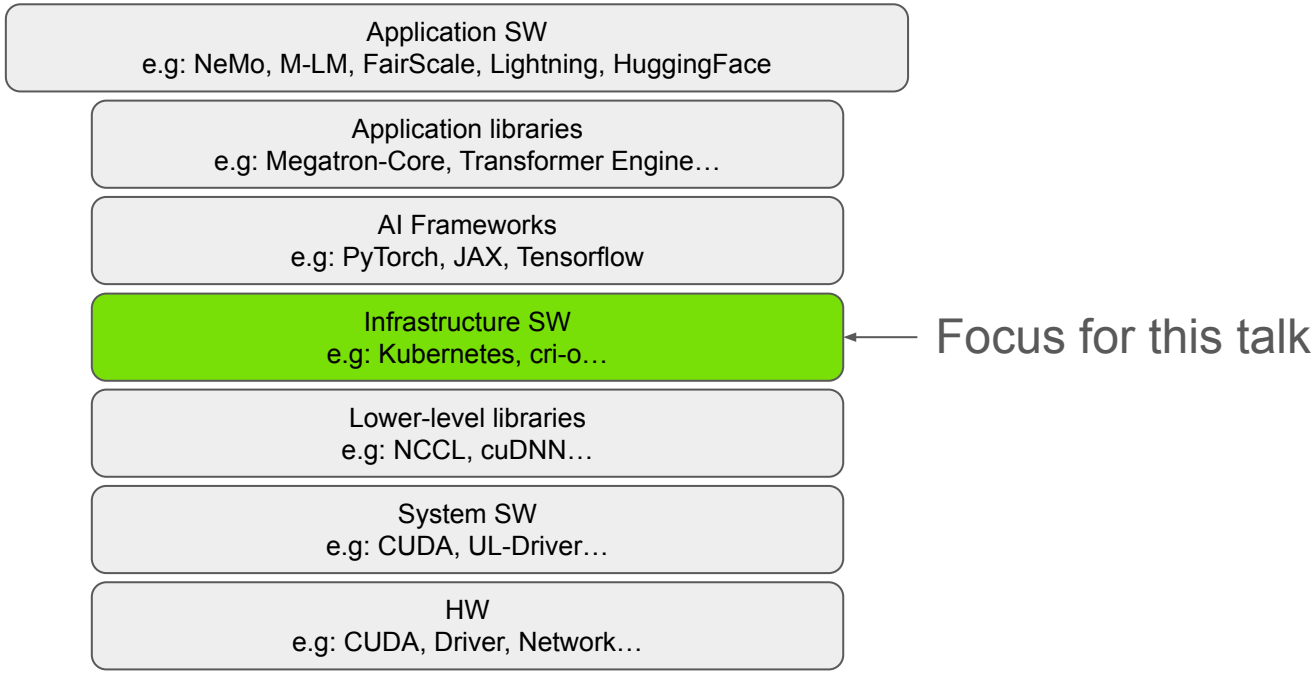


Rest of errors listed accounted for 3.9% with each error < 1%

Source: [The Llama 3 Herd of Models]

Focus for this talk

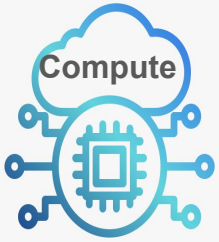
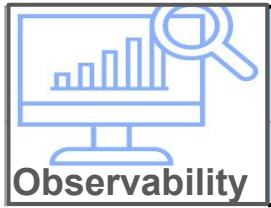
Our goal is to improve cluster utilization and throughput
Fault tolerance and resiliency is applicable in each layer of the training stack



Agenda

1. Motivation for resilience in AI workloads
2. Understanding AI Workloads
3. Where do failures come from?
4. AI Infrastructure - Components and Validation
5. Fault Tolerance - Building Blocks & Mechanisms
6. Monitoring
7. Propagation
8. Recovery strategies
9. Remediation
10. Scope for Enhancement

Components of the AI stack

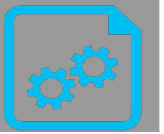
	GPU Operator	GPU mgmt, health status, metrics, drivers & configuration
	Scheduler	Job priority, gang scheduling, preemption support
	MPI Operator	To support MPI based multi node jobs
	Network Operator	RDMA network bring-up, IP address assignment, VF creation/deletion
	Ingress	To enable exec/attach in jobs
	Node local scratch space	Ephemeral, low latency high bandwidth space for user jobs
	Fast parallel remote storage	Secure Storage like lustre with support of RBAC to protect datasets
	Object Storage	For storing checkpoint
	Monitoring	Prometheus, Grafana and Logging solution Job Metrics aggregator for job related metrics
	Node health	Node Problem Detector.Example config: https://tinyurl.com/npd-gpu-node

Cluster Validation

Software checks	✓ Package verification ✓ Driver/library version consistency ✓ Software compatibility
Burn in checks	✓ GPU burn in tests ✓ Backend IB network burn in tests ✓ GPU burn in + IB burn in tests to stress GPU & network together
Performance	✓ Under load, validate all metrics are within required threshold ✓ A large number of small jobs in burst manner to test control plane robustness
End to end check	✓ Run a LLM training job & ensure that it runs within stipulated time
Tuning	✓ OS Tuning ✓ K8s client tuning



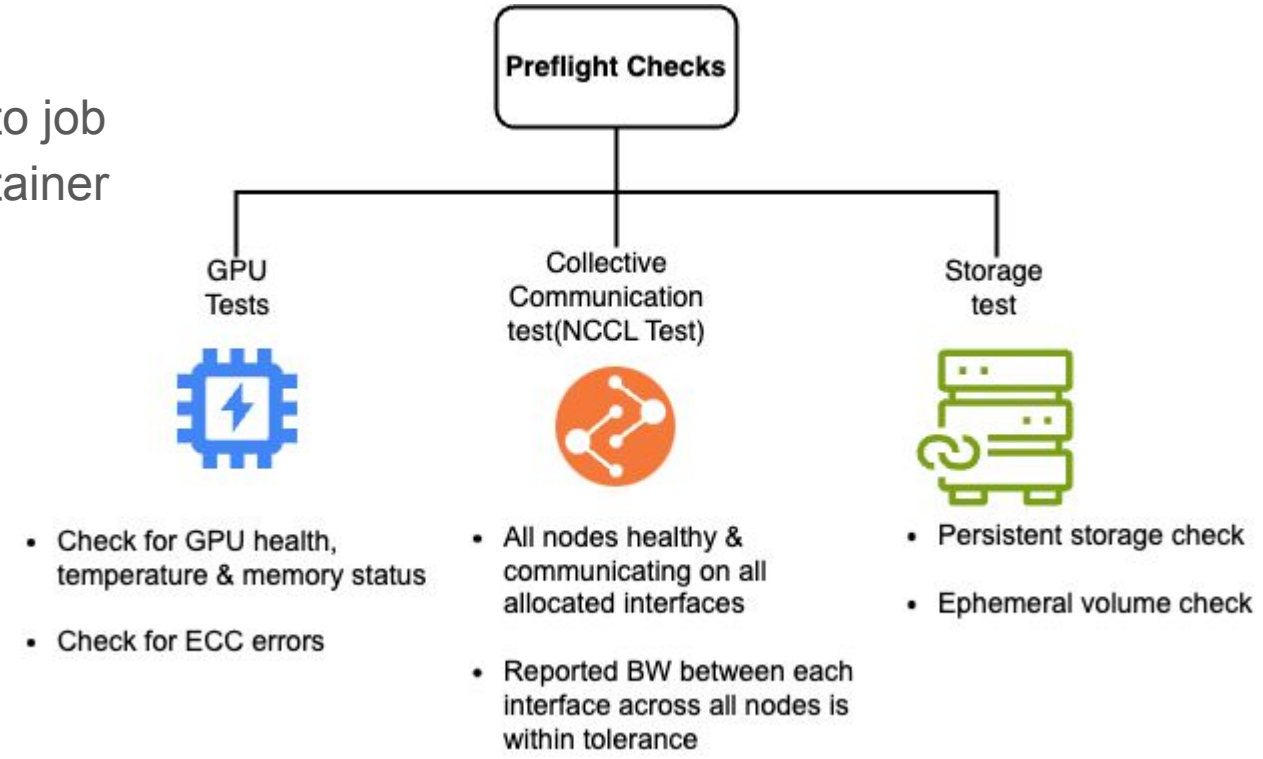




Job Preflight checks

Before job startup,

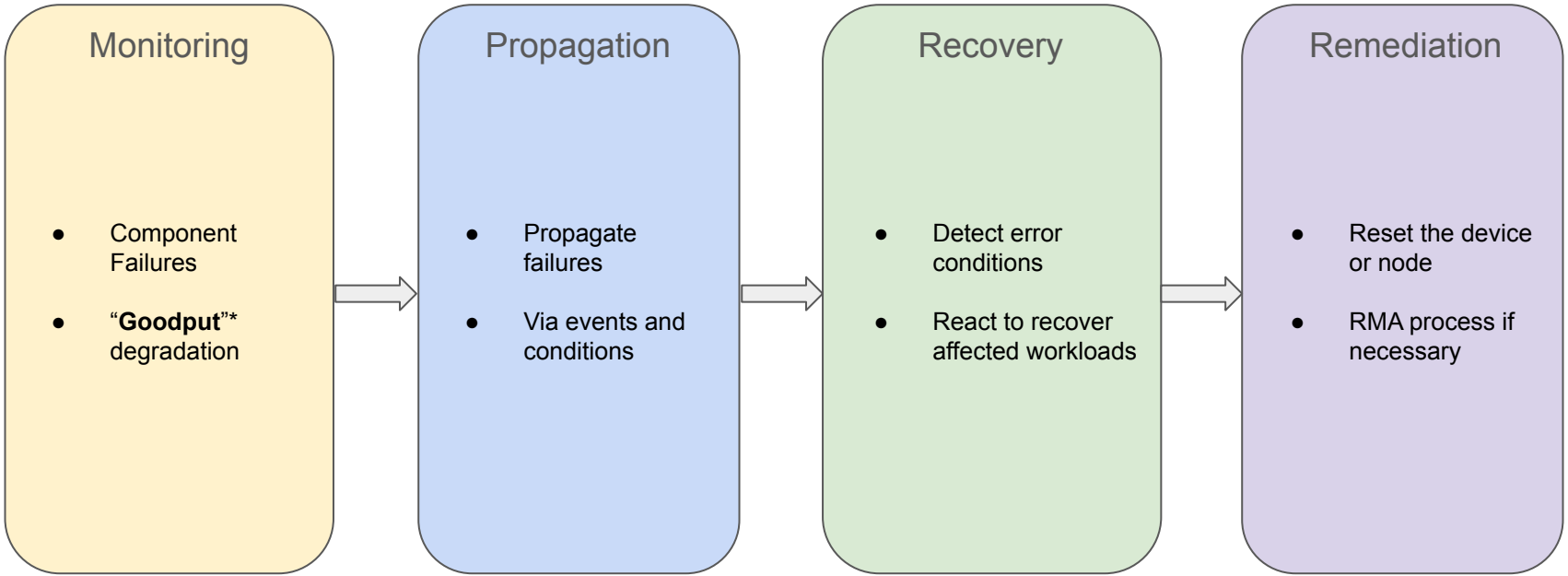
- Preflight check run
- On all nodes allocated to job
- Checks run as init-container



Agenda

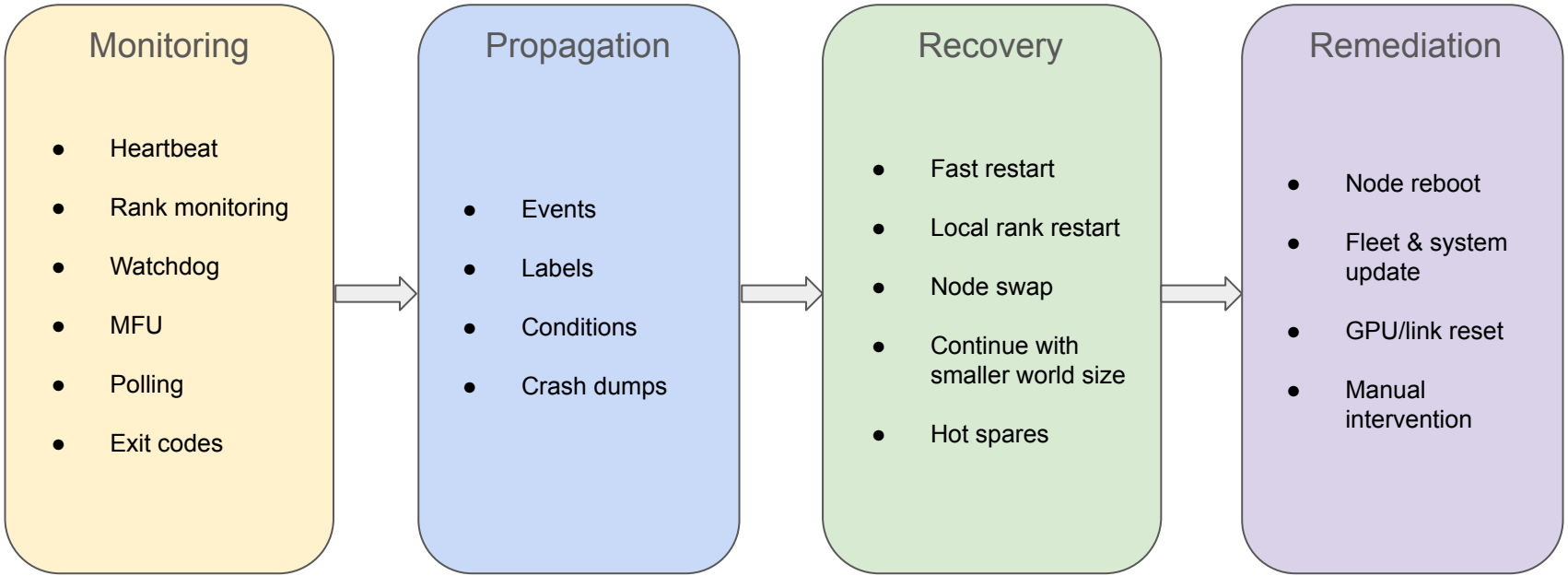
1. Motivation for resilience in AI workloads
2. Understanding AI Workloads
3. Where do failures come from?
4. AI Infrastructure: Components, Bringup & Validation
5. Fault tolerance - Building Blocks & Mechanisms
6. Monitoring
7. Propagation
8. Reaction strategies
9. Remediation
10. Scope for Enhancement

Building blocks: What?

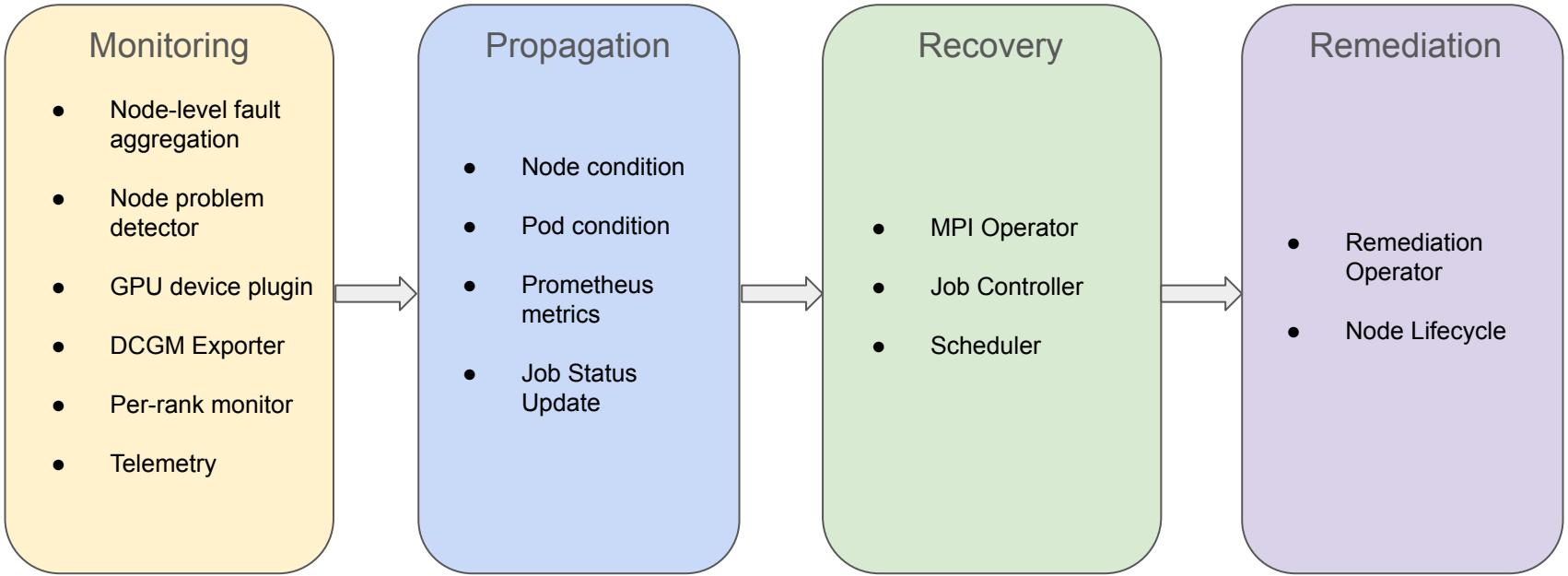


1. ***Goodput**: The time spent computing useful new steps over the elapsed time of the training job.* https://arxiv.org/abs/2312.11805

Building blocks: How?



Building blocks: Where?



Agenda

1. Motivation for resilience in AI workloads
2. Understanding AI Workloads
3. Where do failures come from?
4. AI Infrastructure: Components, Bringup & Validation
5. Fault Tolerance - Building Blocks & Mechanisms
6. Monitoring
7. Propagation
8. Recovery strategies
9. Remediation
10. Scope for Enhancement

Fault Monitoring

NVML	Low level GPU monitoring library.
DCGM	Suite of tools for GPU monitoring & diagnostics
Kernel logs	XID, SXID and GPU Driver errors
RDMA interfaces	Failure counters in sysfs
ML Job specific processes (eg: torch elastic processes)	Failure message & exit code parsing

Information about XID/SXID: https://docs.nvidia.com/deploy/xid-errors/index.html

NVML: Key Errors/Metrics Monitored

ECC Error Counts: Tracks *Correctable single-bit* and *Detectable double-bit errors* (current boot cycle & lifetime).

GPU Utilization: Reports current usage for GPU compute and memory interface.

Active Compute Processes:Lists active processes on the GPU, including *process name*/ *id*, & *allocated GPU memory*.

Clocks and P-State*: Provides max and current clock rates, alongside current performance state.

Temperature & Fan Speed: Reports GPU core temperature and fan speeds.

Power Management: Shows current power draw and power limits for supported GPUs.

Identification Details: Reports board serial numbers, PCI device IDs, and VBIOS/Inforom versions.

NVML Ref: developer.nvidia.com/management-library-nvml

*P-States range from P0 to P15, P0 being the highest performance/power state, and P15 being the lowest performance/power state. Each P-State maps to a performance level

NVML: Sample code


```
1   NVML nvml;
2
3   auto nvml_info = nvml.get_info();
4   std::cout << "driver version:" << "\t" << nvml_info.driver_version << "\n"
5             << "NVML version:"   << "\t" << nvml_info.nvml_version   << "\n"
6             << "\n";
7
8   NVMLDeviceManager device_manager{nvml};
9
10  std::cout << "\n"
11            << "devices_count:" << "\t" << device_manager.get_devices_count() << "\n"
12            << "devices: "      << "\n";
13
14  const auto devices_begin = device_manager.devices_begin();
15  const auto devices_end = device_manager.devices_end();
16
17  for (auto device = devices_begin; device != devices_end; ++device) {
18    const auto& info = (*device).get_info();
19    std::cout << "- device_index:"       << "\t\t"   << info.index << "\n"
20              << "  name:"              << "\t\t\t" << info.name  << "\n"
21              << "  fan_speed:"         << "\t\t"   << info.metrics.fan_speed          << "%"  << "\n"
22              << "  temperature:"       << "\t\t"   << info.metrics.temperature        << "C"  << "\n"
23              << "  power_usage:"       << "\t\t"   << info.metrics.power_usage        << "mW" << "\n"
24              << "  gpu_utilization:"   << "\t"     << info.metrics.gpu_utilization    << "%"  << "\n"
25              << "  memory_utilization:" << "\t"    << info.metrics.memory_utilization << "%"  << "\n";
26  }
```

https://medium.com/devoops-and-universe/monitoring-nvidia-gpus-cd174bf89311

DCGM: Key Metrics/Errors

CLOCK THROTTLE REASON*	DESCRIPTION
GPU_IDLE	Nothing is running on the GPU and the clocks are dropping to Idle state
SW_POWER_CAP	SW Power Scaling algorithm is reducing the clocks below requested clocks
HW_SLOWDOWN	HW Slowdown is an indicator of: • Temperature being too high • External Power Brake Assertion is triggered (e.g. by the system power supply) • Power draw is too high and Fast Trigger protection is reducing the clocks • May be also reported during PState or clock change
SW_THERMAL	SW Thermal Slowdown This is an indicator of one or more of the following: • Current GPU temperature > GPU Max Operating Temperature • Current memory temperature > Memory Max Operating Temperature
HW_THERMAL	Indicator of temperature being too high
HW_POWER_BRAKE	External Power Brake Assertion being triggered (e.g. by the system power supply)
DISPLAY_CLOCKS	GPU clocks are limited by current setting of Display clocks
CLOCKS_SETTING	GPU clocks are limited by current setting of applications clocks

*Each metric is prefixed with DCGM_CLOCKS_THROTTLE_REASON_
*DCGM API Guide: https://developer.download.nvidia.com/compute/DCGM/docs/1.56/DCGM_API_Reference_Guide.pdf

Kernel logs monitoring for XID errors and SXID errors

Overview of Error types:

- XID Errors: Indicate GPU hardware errors; often related to NVLink issues or GPU memory corruption.

```
# XID Error example
[Thu Sep 22 10:53:29 2022] NVRM: Xid (PCI:0000:b5:00): 74, pid='<unknown>', name=<unknown>, NVLink:
fatal error detected on link 6(0x10000, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0)
```

- SXID Errors: Primarily occur in NVSwitch environments.

```
# SXID Error example
[38889.018130] nvidia-nvswitch0: SXid (PCI:0000:07:00.0): 10001, Non-fatal, PRI WRITE SYSB error, instance=3, chiplet=1
[38889.022105] nvidia-nvswitch0: SXid (PCI:0000:07:00.0): 10001, Severity 0 Engine instance 03 Sub-engine instance 01
[38889.025580] nvidia-nvswitch0: SXid (PCI:0000:07:00.0): 10001, Data {0x01615018, 0x00000011, 0x4b409316, 0xbadf2016,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000}
```

Full list of XID and SXID errors: https://docs.nvidia.com/deploy/pdf/XID_Errors.pdf

Kernel logs monitoring for IB instability

- IB Link Flaps: Network link instability events; frequent occurrences may indicate hardware or configuration issues.

```
$ grep -i "ib.*lost carrier" $LOG_PATH | tail -n 1
```

- Ethernet Link flap detection

```
[xxx@yyyy ~]$ dmesg  -T | grep enp94s0
[Fri Jul 23 05:17:30 2021] IPv6: ADDRCONF(NETDEV_UP): enp94s0: link is not ready
[Fri Jul 23 05:17:30 2021] mlx5_core 0000:5e:00.0 enp94s0: Link down
[Fri Jul 23 05:17:30 2021] IPv6: ADDRCONF(NETDEV_UP): enp94s0: link is not ready
[Fri Jul 23 05:17:30 2021] mlx5_core 0000:5e:00.0 enp94s0: Link up
[Fri Jul 23 05:17:30 2021] IPv6: ADDRCONF(NETDEV_CHANGE): enp94s0: link becomes
ready
```

*Caveat: The specific strings to search in kernel logs depend on the environment and setup.
*Thanks to Lalit and Shinae for assistance in kernel log monitoring section

Network Fault monitoring counters

RDMA counters

a. Packet Sequence Errors

```
$ /sys/class/infiniband/mlx5_#/ports/1/hw_counters/packet_seq_err
```

b. Local ACK Timeout Errors

```
$ /sys/class/infiniband/mlx5_#/ports/1/hw_counters/local_ack_timeout_err
```

c. Link Flap Counters

```
$ /sys/class/infiniband/mlx5_#/ports/1/counters/link_downed

$ /sys/class/infiniband/mlx5_#/ports/1/counters/link_error_recovery
```

Represents port number

Network Fault troubleshooting tests

1. NCCL tests (https://github.com/NVIDIA/nccl-tests)

```
$ NCCL_SHM_DISABLE=1 NCCL_P2P_DISABLE=1 NCCL_DEBUG=INFO
/usr/local/bin/all_reduce_perf_mpi -b1G -e1G -f2 -t8 -g1
```

2. IB Tests (https://github.com/Mellanox/netop-tools)
   a. Device status

```
$ ibv_devinfo
$ ip -br a
```

```
$ mst status -v
$ ibstat
```

   b. Ping tests

```
$ ibping -S -C mlx5_1 -P 1
$ ibping -c 10000 -f -C mlx5_0 -P 1 -L 2
```

   c. Bandwidth test

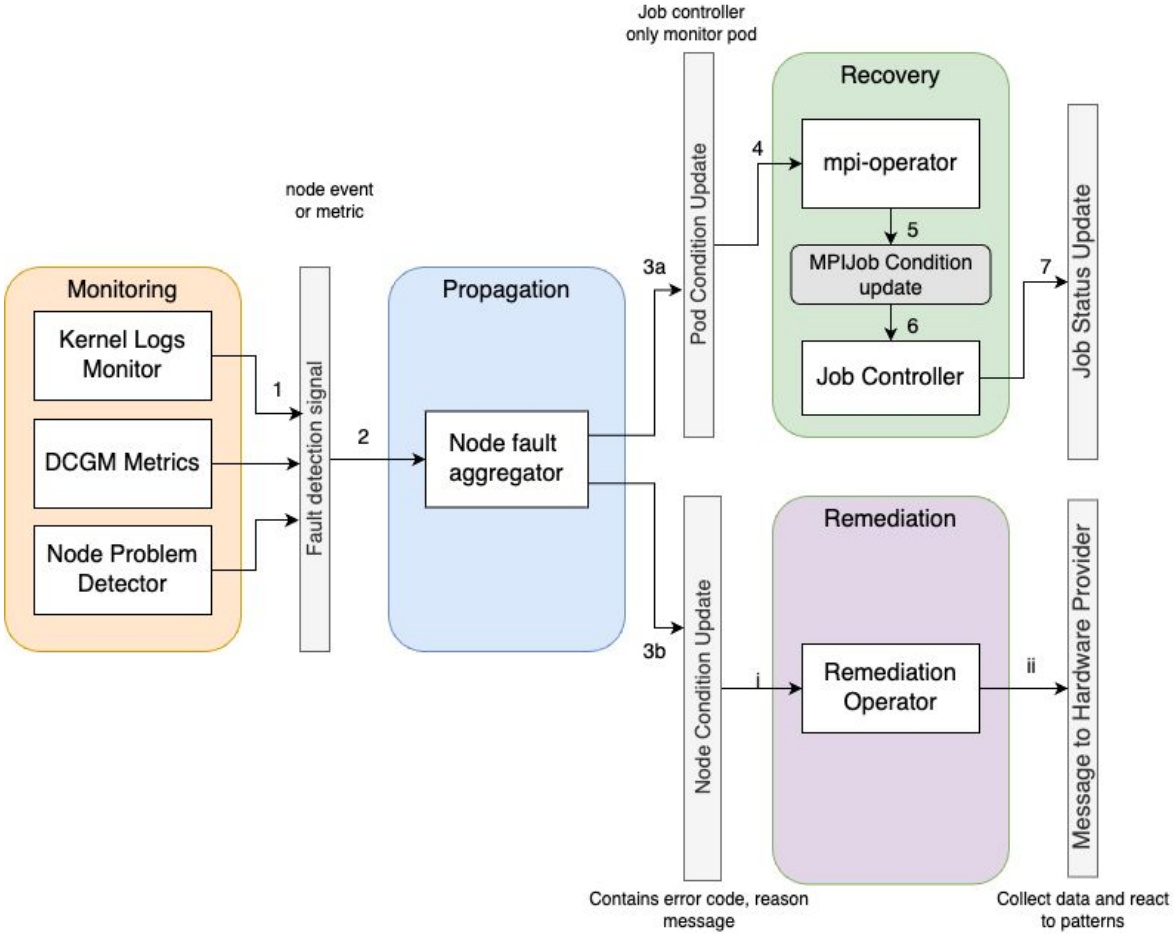
```
$ ib_write_bw -d mlx5_3
$ ib_write_bw -d mlx5_3 <ip address>
```

*Ensure necessary packages are installed; commands may vary with network configuration.
*Thanks to Tom for assistance in network fault troubleshooting section

Agenda

1. Motivation for resilience in AI workloads
2. Understanding AI Workloads
3. Where do failures come from?
4. AI Infrastructure: Components, Bringup & Validation
5. Fault Tolerance - Building Blocks & Mechanisms
6. Monitoring
7. Propagation
8. Recovery strategies
9. Remediation
10. Scope for Enhancement

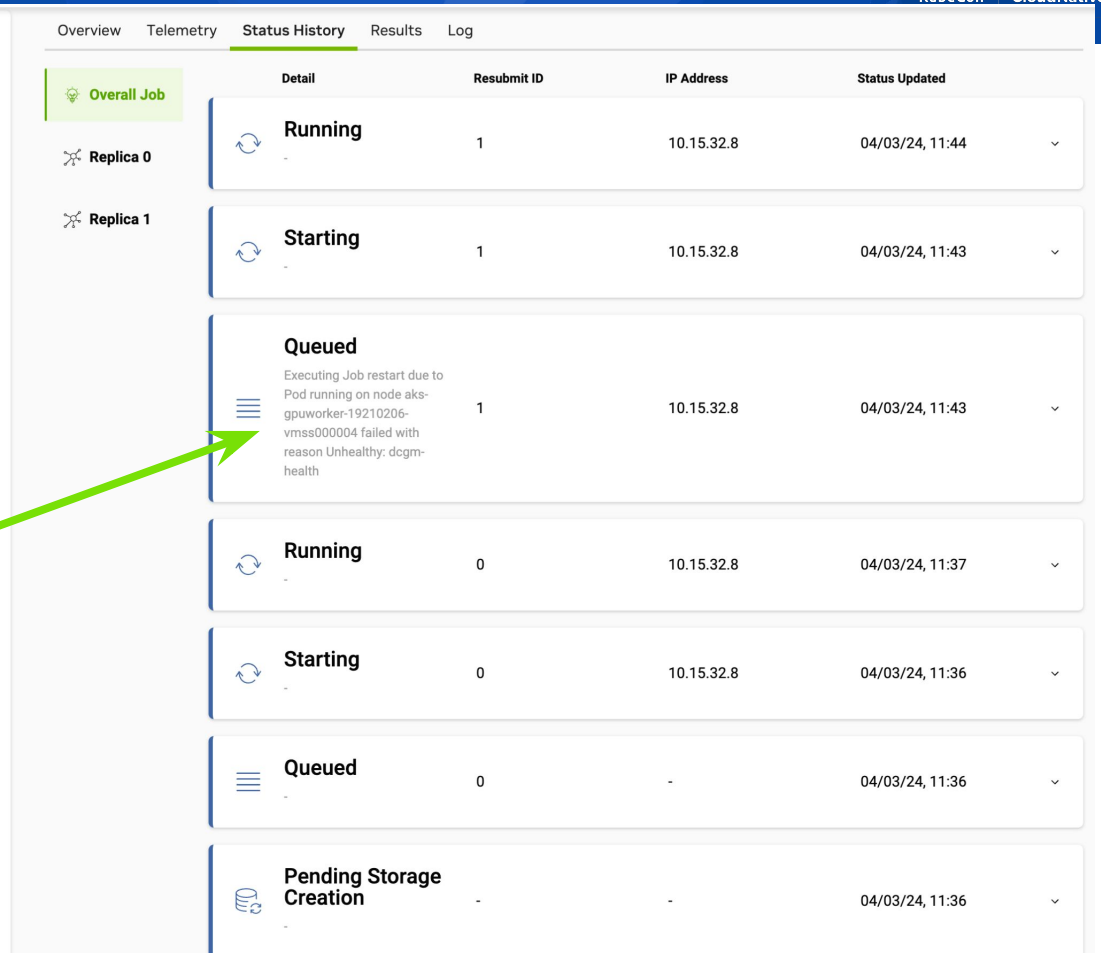
Fault Propagation



Job Status Updates

Job is re-queued after we detect an error on the host

Job status indicates error for user visibility



Agenda

1. Motivation for resilience in AI workloads
2. Understanding AI Workloads
3. Where do failures come from?
4. AI Infrastructure: Components, Bringup & Validation
5. Fault Tolerance - Building Blocks & Mechanisms
6. Monitoring
7. Propagation
8. Recovery strategies
9. Remediation
10. Scope for Enhancement

Recovery strategies

In job recovery

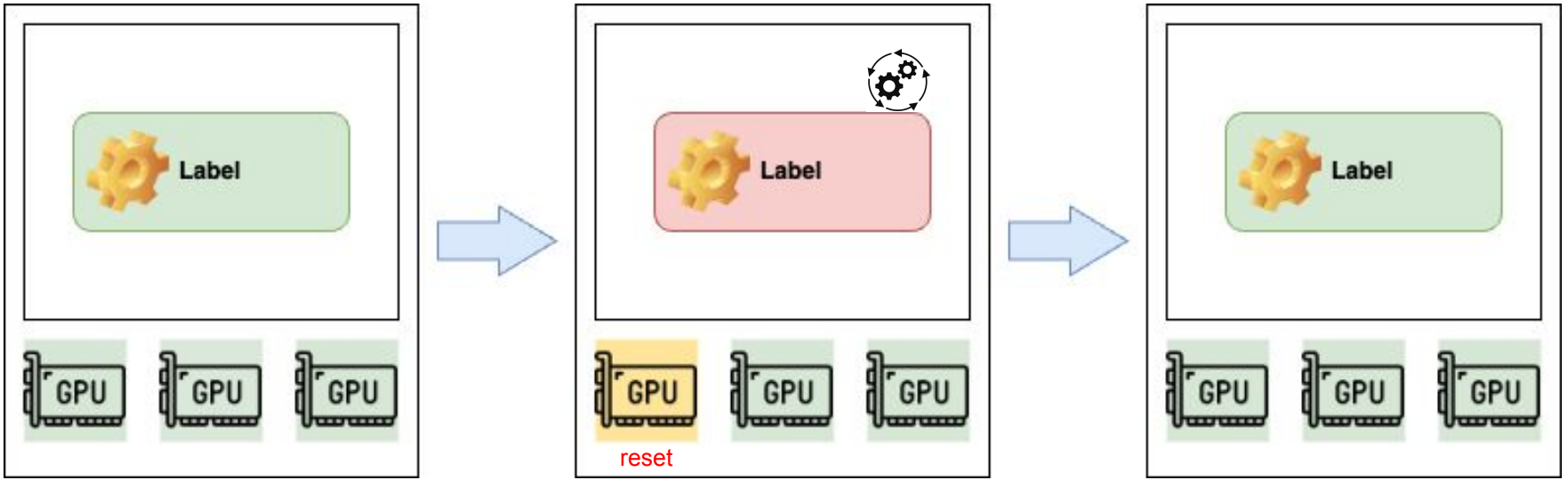
- In-Process restart
- In-Job restart
  - Reduced world size
  - Warm spares

In cluster recovery (job restart)

- Naive restarts
- Reduced world size
- Warm spares
- Job preemption

In-Process restart

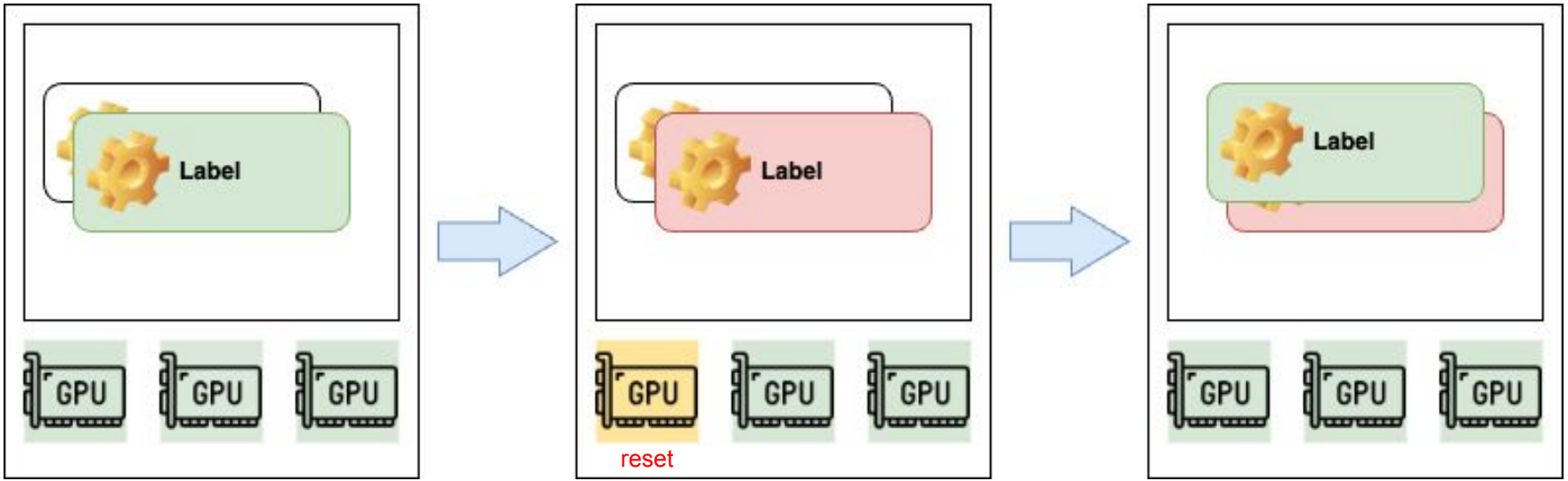
Automatic restart of training loop without process restart



In-Process restart

Automatic restart of training loop without process restart

Pre-create a stand-by process



In-Job restart strategies: Reduced world

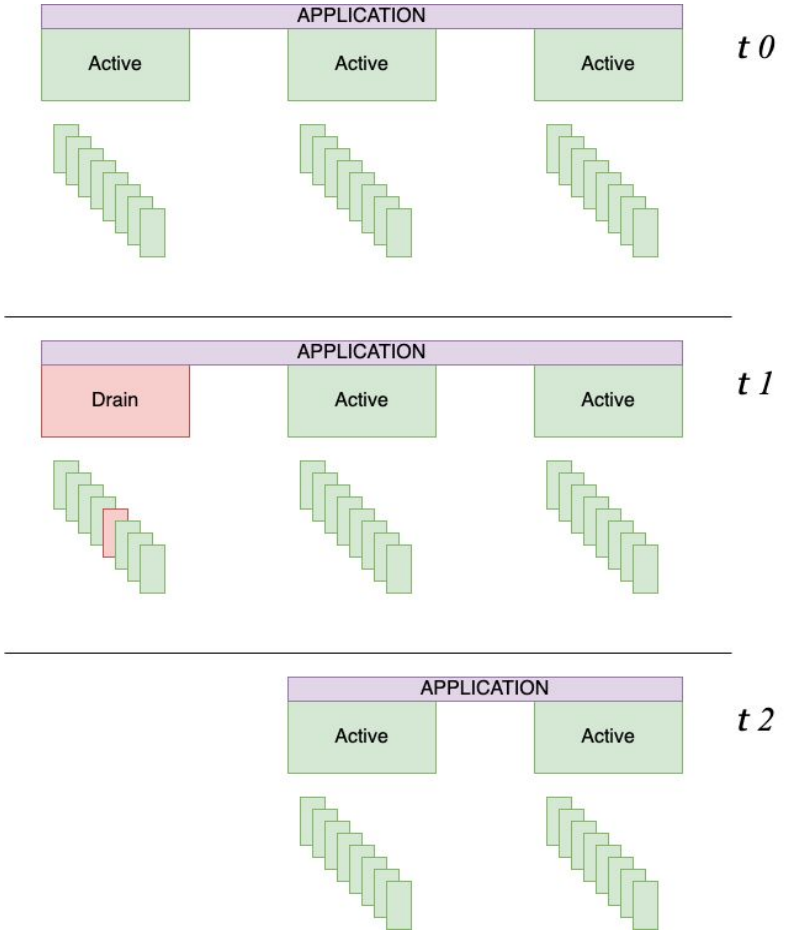
Automatic restart of training process without job restart when the failure is not recoverable.

Restart the processes only on healthy nodes

Loss of capacity results in lower application performance

Need support built into the application and DL frameworks

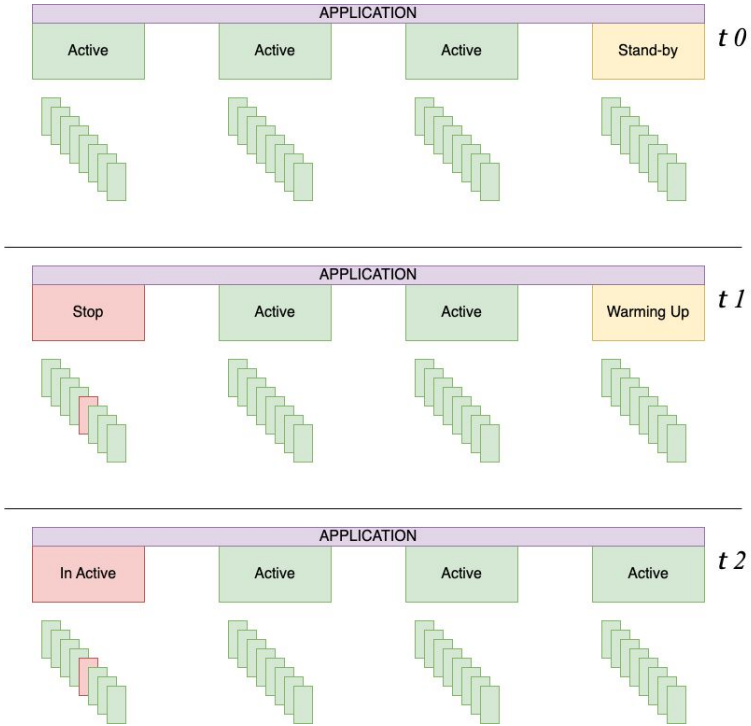
- Flexible checkpoints
- Compilation cache



In-Job restart strategies: Hot spare

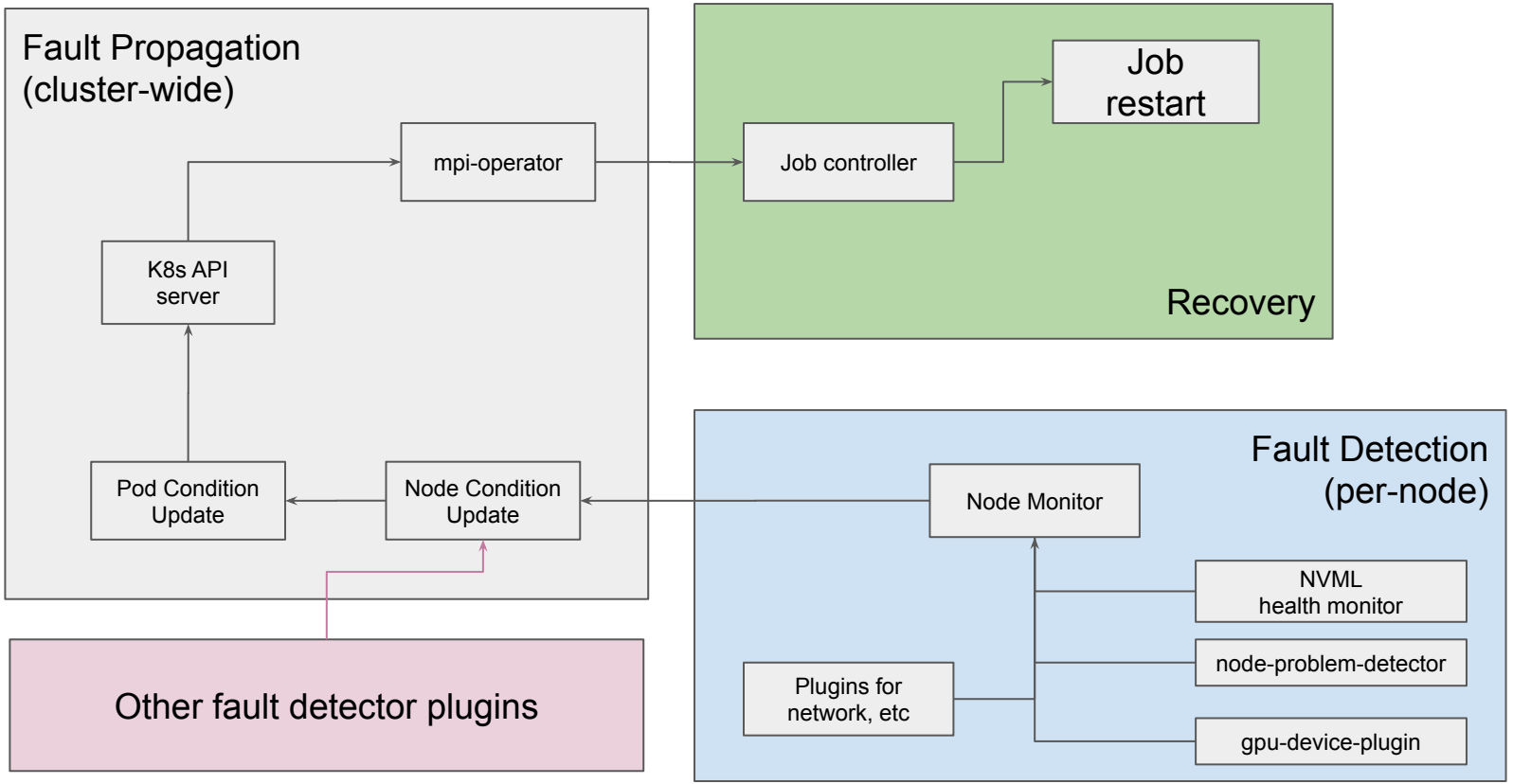
Always start the job with additional “spare” capacity

Warm sparing minimizes restart time, but locks up capacity that could be used for other workloads.



Job restart

Job restart workflow



Naive job restart

Naive restart leads to

- Higher job queue times
- Higher startup time

Optimizations

- **Queue time reduction**: Restarted job could be prioritized over other jobs
- **Startup time reduction**: Use the healthy nodes from the previous run to reuse cache

Job restart with reduced world

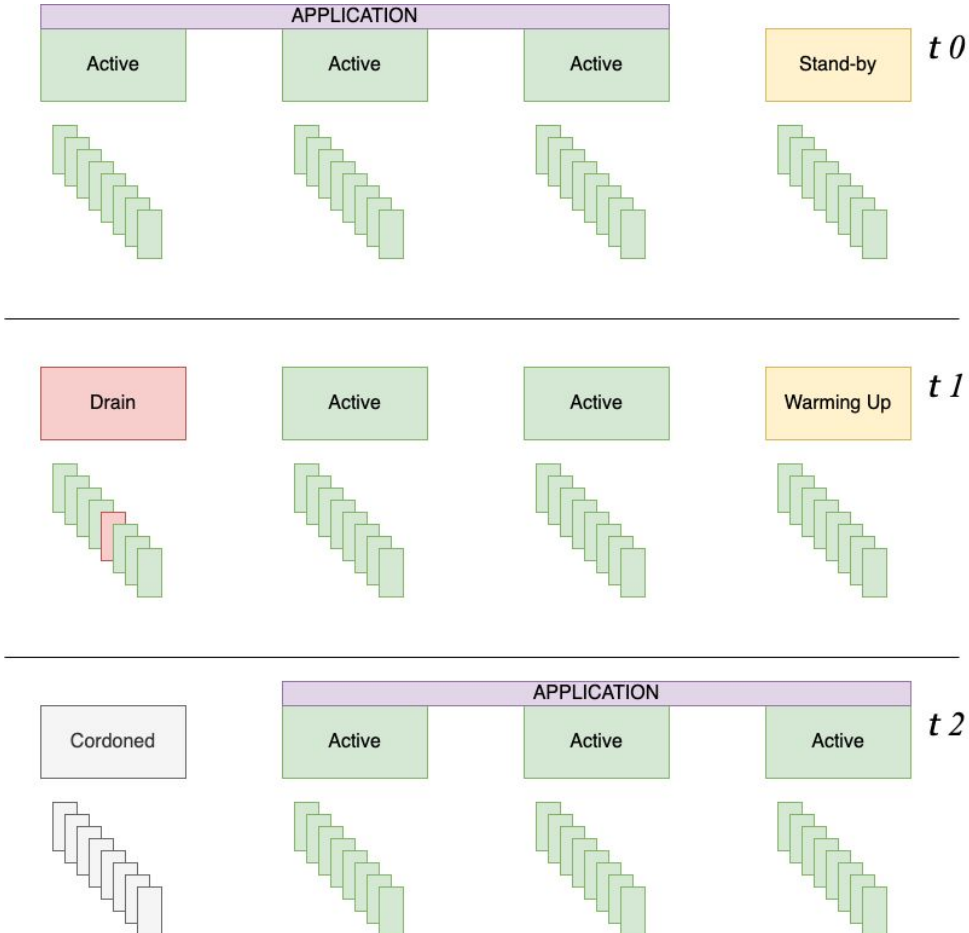
- Upon detecting a failure
  - Stop the job
  - Update the host list to exclude the faulty node for MPI compatibility
  - Relaunch the job with only healthy nodes.

Job restart with warm spares

- Few nodes are marked “stand by” in the cluster.
- On failure, the unhealthy node is cordoned
- One of the nodes from the stand-by pool is allocated by scheduler and job is restarted.

Stand-by nodes could preemptively cache data sets and container images of the running job.

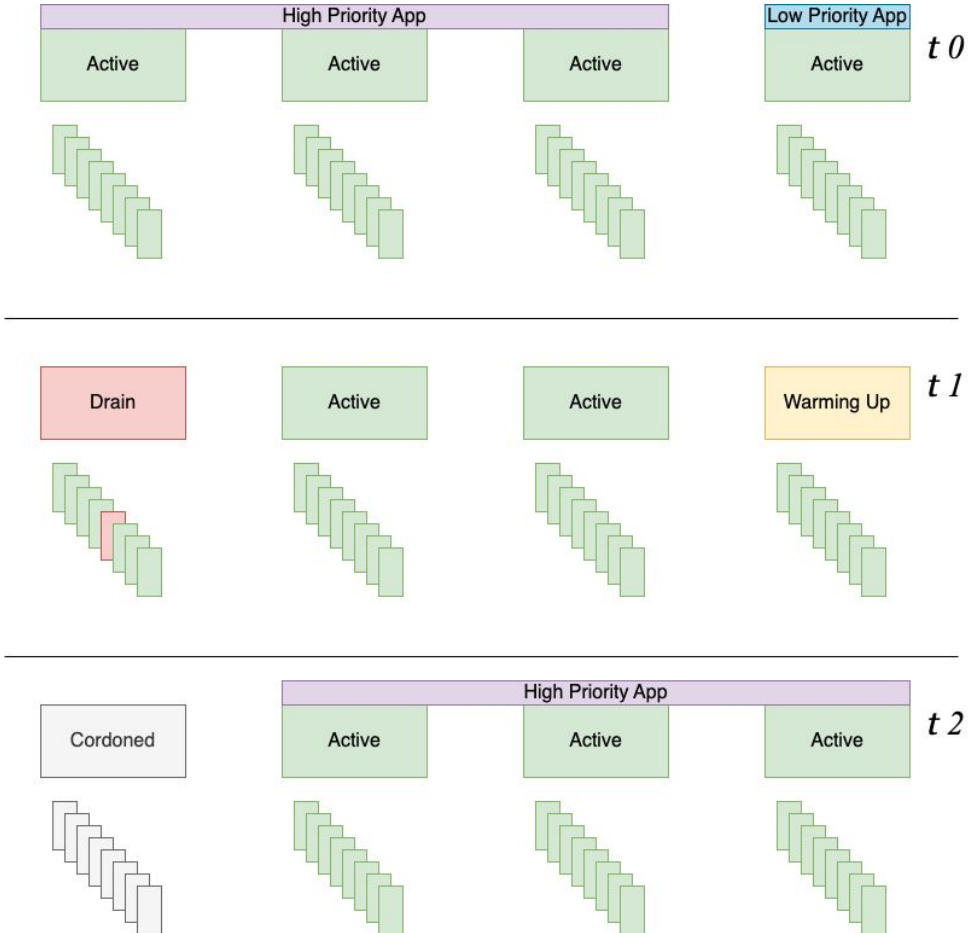
Idle nodes in the cluster are wasted resources



Job restart with scheduler preemption

- Some nodes in the cluster may run low-priority jobs to increase cluster utilization
- On node failure, the job is stopped.
- A low priority job preempted to free up capacity.
- High priority job continues with the new node.

Preemption needs scheduler support

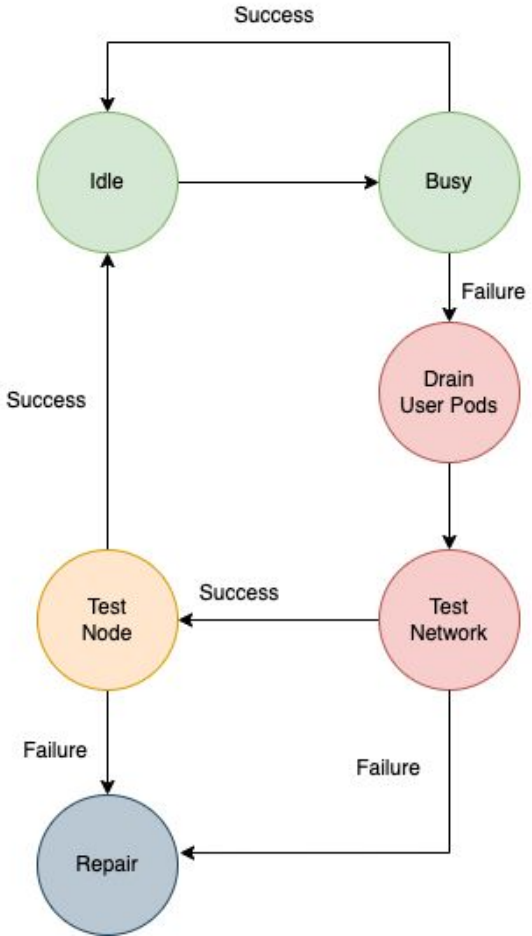


Agenda

1. Motivation for resilience in AI workloads
2. Understanding AI Workloads
3. Where do failures come from?
4. AI Infrastructure: Components, Bringup & Validation
5. Fault Tolerance - Building Blocks & Mechanisms
6. Monitoring
7. Propagation
8. Recovery strategies
9. Remediation
10. Scope for Enhancement

Node lifecycle

- It is often required to run some tests on a node after any failures to determine if the failure is transient or permanent
- A reboot can resolve many issues!
- Node hardware, including GPUs and network interfaces, connectivity with the backend IB fabric needs to be thoroughly tested before the node is allowed to join the cluster after a failure
- Frequent node errors lower the “node score” which could be used as scheduler input



Node Remediation Operations

Applicable Errors	Operation
Lost GPU (fall off the bus)	Sometimes kubectl can report a GPU as missing for benign XID errors. Check if NVML can detect the missing GPU. If not, reboot the node
GPU or IB nics overheating	Node RMA (most likely due to bad fans or improper cooling)
IB link loss	Check IB links if the link loss is temporary. If not, a node reboot might help
Lost storage access Storage access slowdown	Check CSI pods and restart them if necessary

NVIDIA GPU Debug Guidelines: https://docs.nvidia.com/deploy/gpu-debug-guidelines/index.html

Agenda

1. Motivation for resilience in AI workloads
2. Understanding AI Workloads
3. Where do failures come from?
4. AI Infrastructure: Components, Bringup & Validation
5. Fault Tolerance - Building Blocks & Mechanisms
6. Fault Monitoring
7. Fault Propagation
8. Fault Recovery strategies
9. Fault Remediation
10. Scope for Enhancement

Scope for Enhancement

1. Scale to 100K GPUs
   a. 12k to 15k worker nodes.
   b. Monitor health of 100K GPUs and 100K RDMA hardware devices
   c. Scale status and health updates

2. Scheduler features for AI workloads
   a. Gang scheduling with priority and preemption

3. Standardized Exit and Error Codes for apps
   a. Help controllers interpret & respond to application errors effectively
   b. Distinguish between infrastructure errors and application errors.

Questions?

Please scan the QR code to leave feedback on this session