# Disk Usage of Jobs at the EP

## HTCondor European Workshop

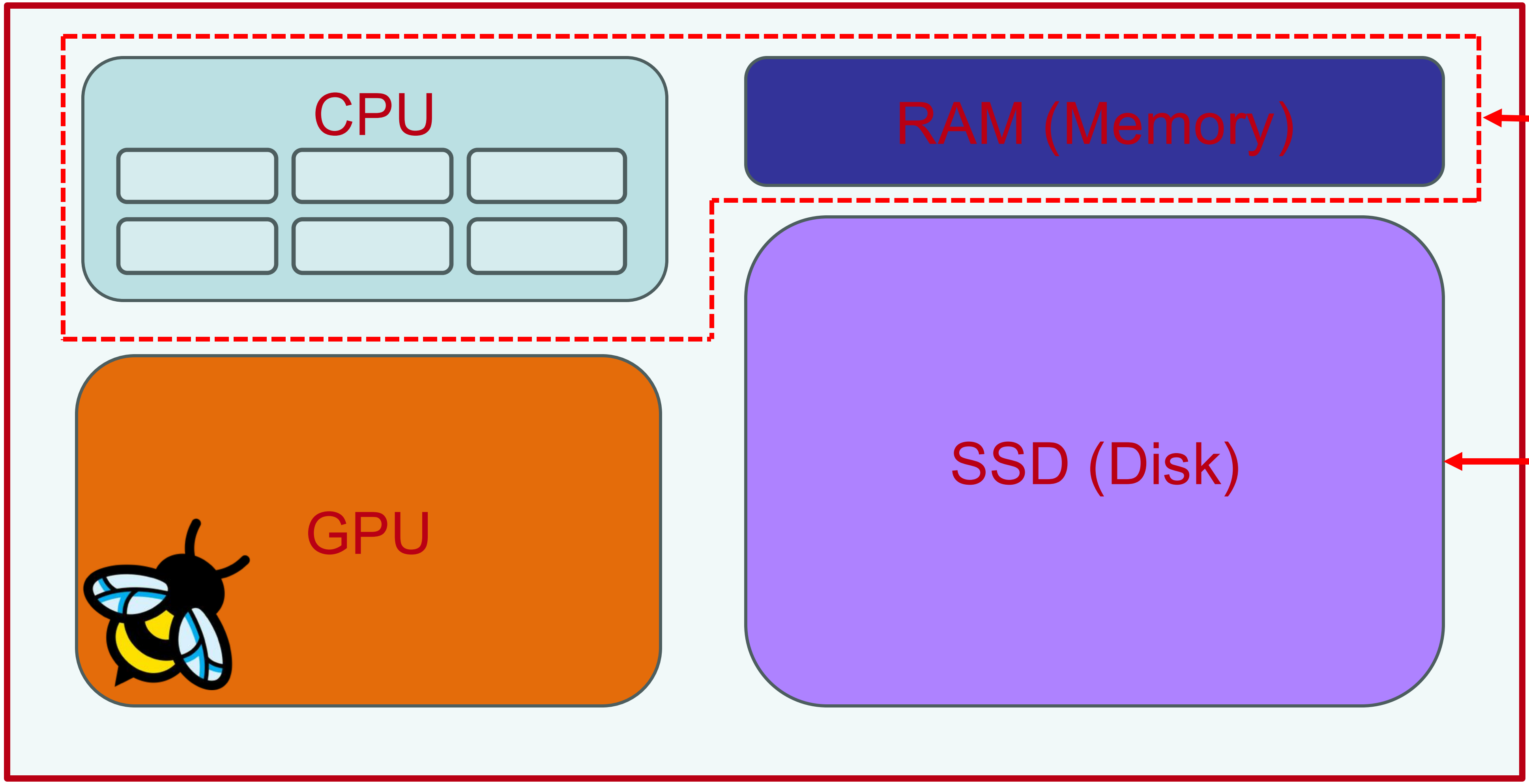
**Brian Bockelman**

MORGRIDGE
INSTITUTE FOR RESEARCH

# Jobs are Guests

- Jobs are being permitted to execute on your resources.
- Don’t want jobs to mess with other user jobs.
- Don’t want jobs to mess up the host machine.


# Resources at the EP




CPU & Memory enforced by cgroups

New! We can enforce disk usage!

# Basic Concept

- What mechanism exists to enforce disk usage?
  - Historical approach: Walk the scratch directory every 5 minutes, count files and add up bytes.

# Basic Concept

- What mechanism exists to enforce disk usage?
  - ~~Historical approach: Walk the scratch directory every 5 minutes, count files and add up bytes.~~ Too slow
  - Another idea: XFS project quotas, one per execute directory.

# Basic Concept

- What mechanism exists to enforce disk usage?
  - ~~Historical approach: Walk the scratch directory every 5 minutes, count files and add up bytes.~~ Too slow
  - ~~Another idea: XFS project quotas, one per execute directory.~~ XFS specific

# Basic Concept

- What mechanism exists to enforce disk usage?
  - ~~Historical approach: Walk the scratch directory every 5 minutes, count files and add up bytes.~~ Too slow
  - ~~Another idea: XFS project quotas, one per execute directory.~~ XFS specific
  - Carve out a filesystem per job. ✅

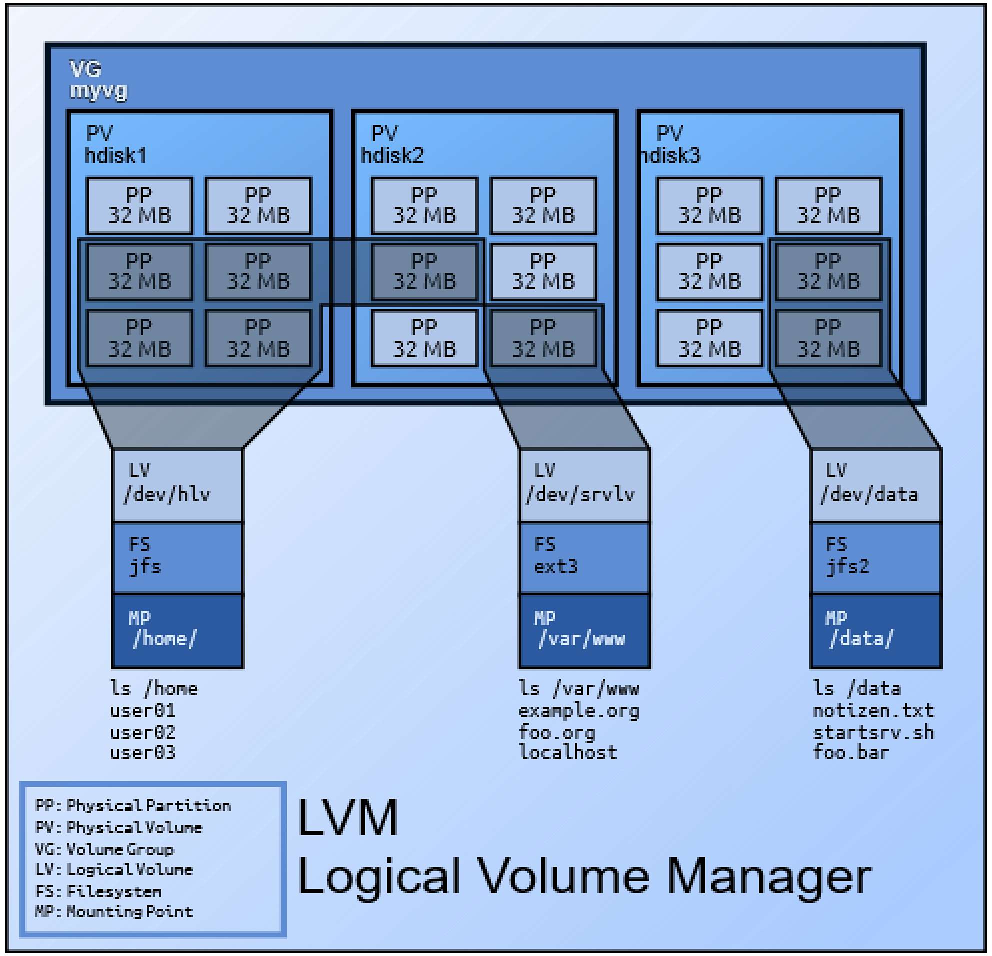
- What's nice about this?
  - Check job disk usage? "df"
  - Cleanup after the job? Unmount / delete the filesystem (quick!)
  - What happens when the job runs out of space? …. More later …

# How does the EP enforce disk usage?

- Using the Logical Volume Manager (LVM).
- The EP will create a unique ephemeral Logical Volume (LV) for each job the EP runs.

## Requirements

1. Linux OS
2. HTCondor running as root

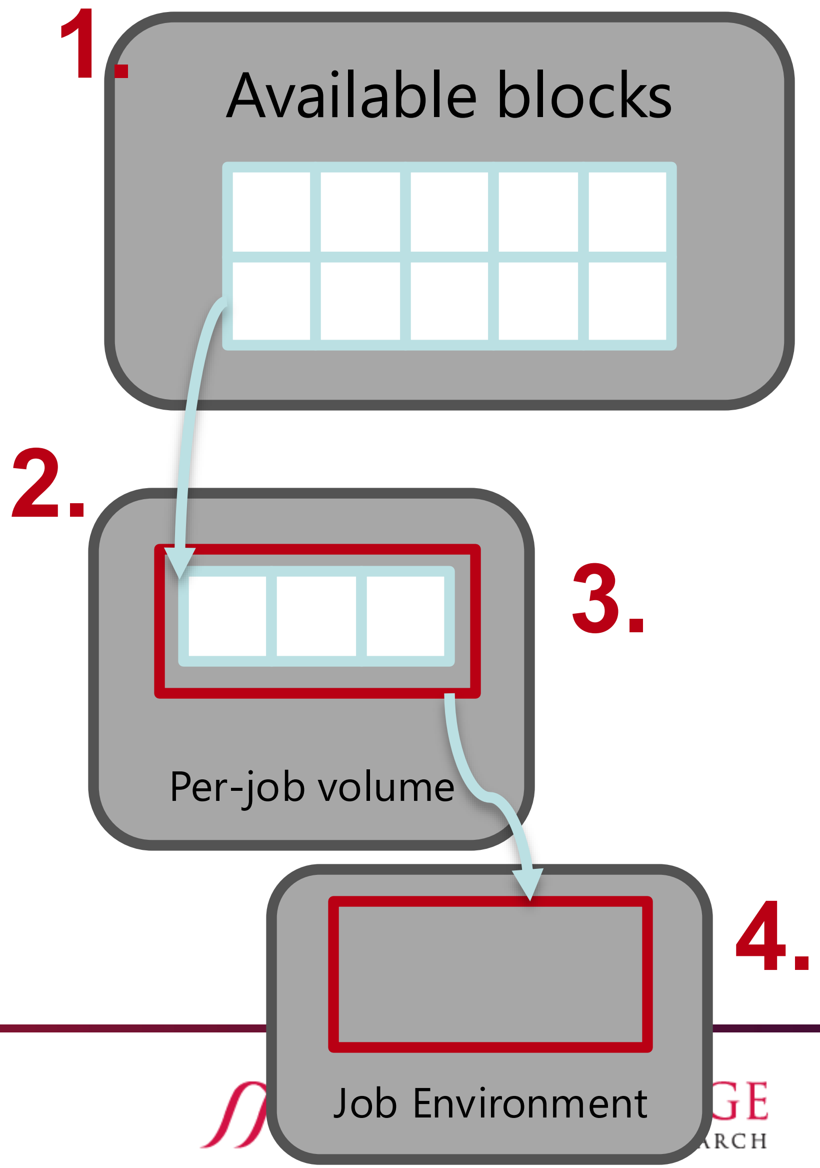



[Wikipedia - Various elements of the LVM]

# Logical Logical Volume Manager Logically

Here's how I think about the sequence:

1. The EP is given a raw block "device" it is the king of. The size of the raw device is the advertised "disk space".
2. When a claim is made for 10 GB, the starter will scoop out a 10GB of raw blocks into a "volume"
3. The starter then creates a filesystem in the volume.
4. The filesystem is mounted as the scratch directory inside the job's runtime environment.



# How to Enable Disk Enforcement?

- Step 1:

  Add the configuration STARTD_ENFORCE_DISK_LIMITS = True

- Step 2:

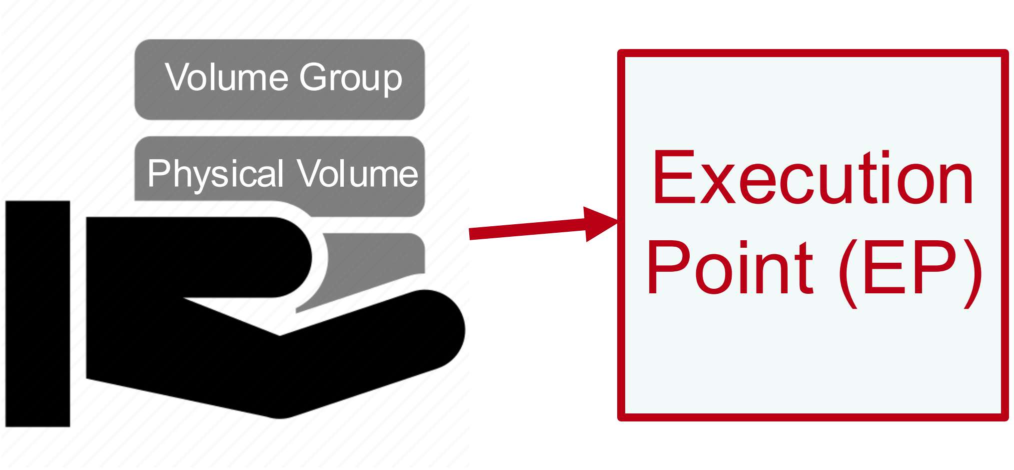
  Profit!

- Step 3:

  Wait, what if I don't have a block device? 🤔

# Setup LVM for the EP!

- Add the configuration STARTD_ENFORCE_DISK_LIMITS = True
- If you have a LVM setup with raw space,
  - Inform the EP of Volume Group name to use via LVM_VOLUME_GROUP_NAME
    - (No default; must be set)
- **Pitfall**: The EP assume it owns the entire volume group.
  - If other things use the VG, strange things will happen with disk space.



# Example Setup

Volume Group: condor_vg

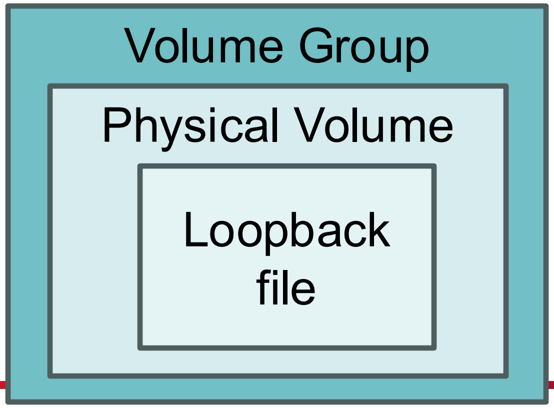
| Physical Volume | Physical Volume | Physical Volume |
|---|---|---|
| /dev/ssd1 | /dev/ssd2 | /dev/ssd3 |

```
# pvcreate /dev/ssd1 /dev/ssd2 /dev/ssd3
# vgcreate condor_vg /dev/ssd1
```

```
# Sample configuration
STARTD_ENFORCE_DISK_LIMITS = True
LVM_VOLUME_GROUP_NAME = condor_vg
```

# How to Enable Disk Enforcement?

- Add the configuration STARTD_ENFORCE_DISK_LIMITS = True
- If no block device is available, what next?
- Concept: **The Linux "loopback device"**. The EP will
  1. Create a file on disk of size "N GB". This file is our pretend "hard drive".
  2. Mount file as a loopback device. **Now, the OS sees a raw device**.
  3. Use the loopback device for the LVM volume group.
- Set the following configurations:
  - LVM_BACKING_FILE_SIZE_MB (Defaults to 10GB)
  - LVM_BACKING_FILE (Location on the filesystem)




**Note:** The EP needs a restart to enable or disable LVM enforcement.

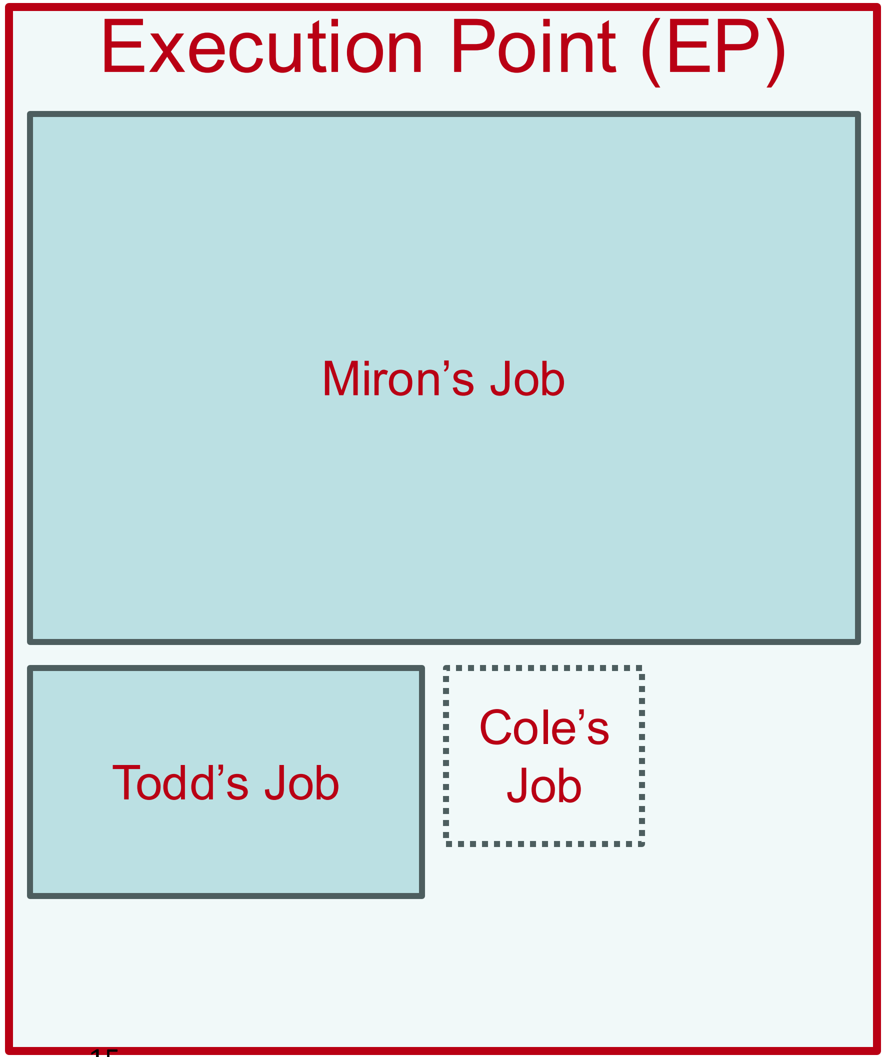
More in the HTCondor Manual
[HTCondor EP LVM Integration Documentation]

# Why should I bother with this?

# Benefit: Improved Job Isolation

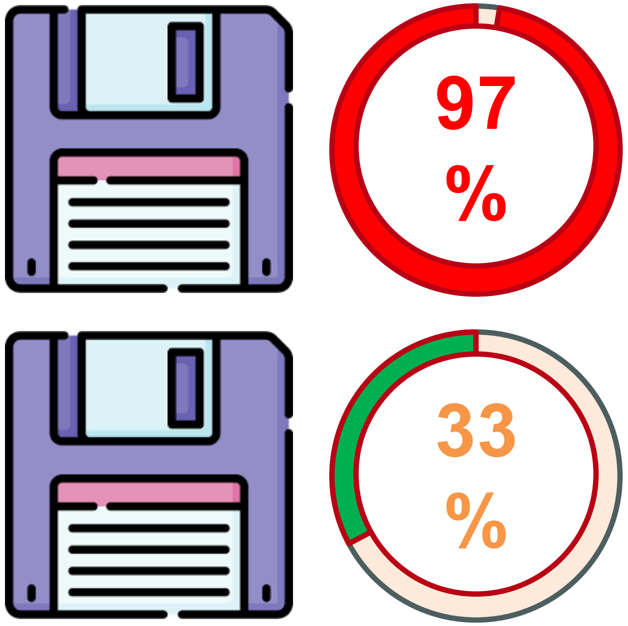
- Each job runs in its own filesystem.
  - The FS lives in the ephemeral LV created for the job.
  - Todd's job cannot use the space assigned to Cole's job.
- Make the FS only visible to the user job with LVM_HIDE_MOUNT
  - Note: This option does not play well with VM or Docker Universe Jobs!
  - Default is AUTO

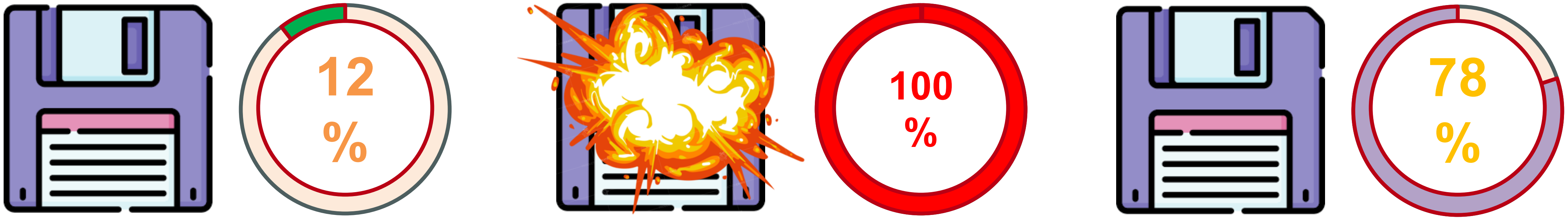
Configuration Knob - LVM_HIDE_MOUNT



# Benefit: Improved Space Management

- Jobs get what they ask for!
- A job can not run away and use more disk than requested.
- If a job uses all its assigned space, neither the host machine or other jobs will be affected.



# Benefit: Improved Reporting

If I could have your new disk requests ASAP that would be great

- HTCondor will monitor the LV usage to report over usage of disk.
- With a hard cap on disk space comes a greater risk of ENOSPC.
- Job put on hold with nice message rather than job appearing to randomly fail/crash.

Maybe!  Can anyone spot the race condition?

# Thin Vs Thick Provisioning

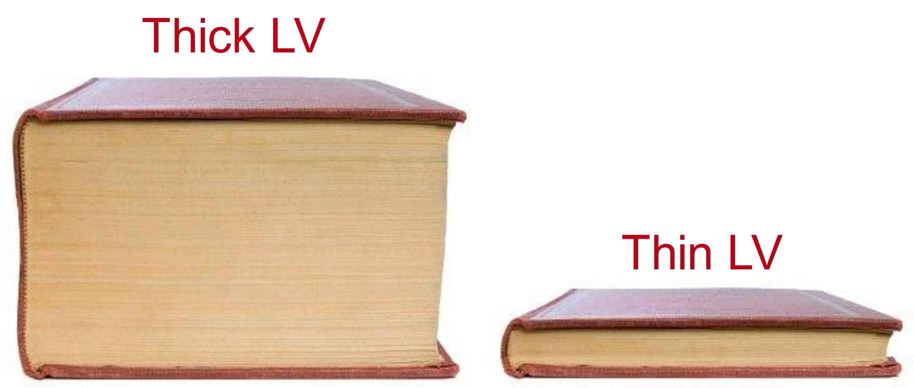
**Thin Provisioning**

Disk is provisioned as needed by writing to the filesystem.

**Thick Provisioning**

- Disk is provisioned at LV creation time
- Just requires a Volume Group

LVM_USE_THIN_PROVISIONING = true




**What happens when you're out of space?**

# Thin Vs Thick Provisioning

- Thick provisioning:
  - Job's system call gets an "out of space" response (ENOSPC)
    - … or job is sent a random SIGBUS signal and die.
  - **<u>You better hope the user wrote excellent error handling code!</u>**
    - (PS – they didn't)
- Thin provisioning:
  - Kernel pauses the operation until "something happens"
    - Something can be "process is killed, operation is cancelled" or "more blocks are created".
    - **Goal**: condor_startd notices the paused process, cleans it up, creates a nice error message for the user.
      - <u>Does not rely on user code being correct!</u>

# Thin Vs Thick Provisioning

- Thick provisioning:
  - Job's system call gets an "out of space" response (ENOSPC)
    - … or job is sent a random SIGBUS signal and die.
  - **You better hope the user wrote excellent error handling code!**
    - (PS – they didn't)
- Thin provisioning:
  - Kernel pauses the operation until "something happens"
    - Something can be "process is killed, operation is cancelled" or "more blocks are created".
    - **Goal**: condor_startd notices the paused process, cleans it up, creates a nice error message for the user.
      - Does not rely on user code being correct!
  - Except…

**Sometimes the startd also writes into the directory, deadlocking itself.**
**(Fixed with the nested scratch feature)**

# Benefit: More effective accounting

- No more pesky, slow traversal of the job's sandbox!
  - Currently an EP must traverse the entire job sandbox to report the job's disk usage (counting each file manually) and to cleanup after the job.
- With an LV we can do a simple query for disk usage.
- Once the job is gone, we can simply delete the LV in one step.

# Benefit: Data Encryption

- The EP can encrypt the job’s volume (via cryptsetup in the kernel)
  - Administrator can enable LV encryption for all jobs via configuration: ENCRYPT_EXECUTE_DIRECTORY
  - Users can request encryption with the a submit command: encrypt_execute_directory
  - The volume’s decryption key is immediately thrown away:
    - No way to decrypt working directory if the host crashes.  No one can steal the hard drive and read the contents.

- This replaces the prior approach using eCryptFS (deprecated, no longer shipped with EL starting at RHEL7)

# Parting Thoughts

- All resources must be managed!
  - Lack of effective disk management was long an irritant for HTCondor pool admins.
- HTCondor's solution approach relies on LVM.
  - Most sysadmins will prefer setting aside a raw block device on next reinstall.
  - Loopback device works in a pinch – less painful retrofit!
    - Can allow more clever setups – loopback on NFS?
- What should the next resource we manage?
  - Network anyone? …

# Questions?

This project is supported by the National Science Foundation under Cooperative Agreements OAC-2030508. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

MORGRIDGE
INSTITUTE FOR RESEARCH