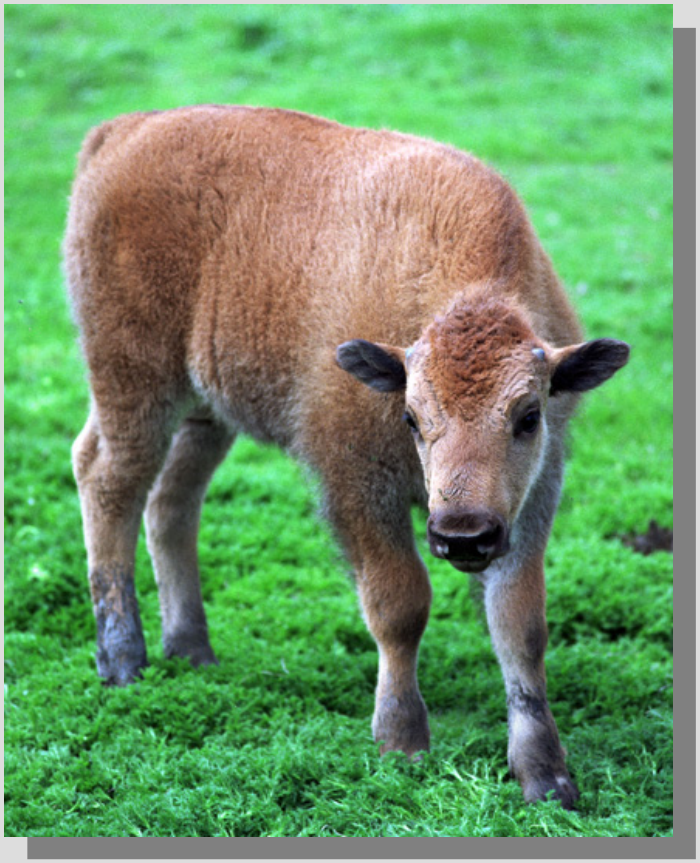
Introduction to the CAF Architecture

Elliot Lipeles (UCSD)

- Overview of the Idea
- Bird's eye view of the Condor implementation
- The life of a job
- CAF-SAM Interface
- Monitoring
- Priority Scheme



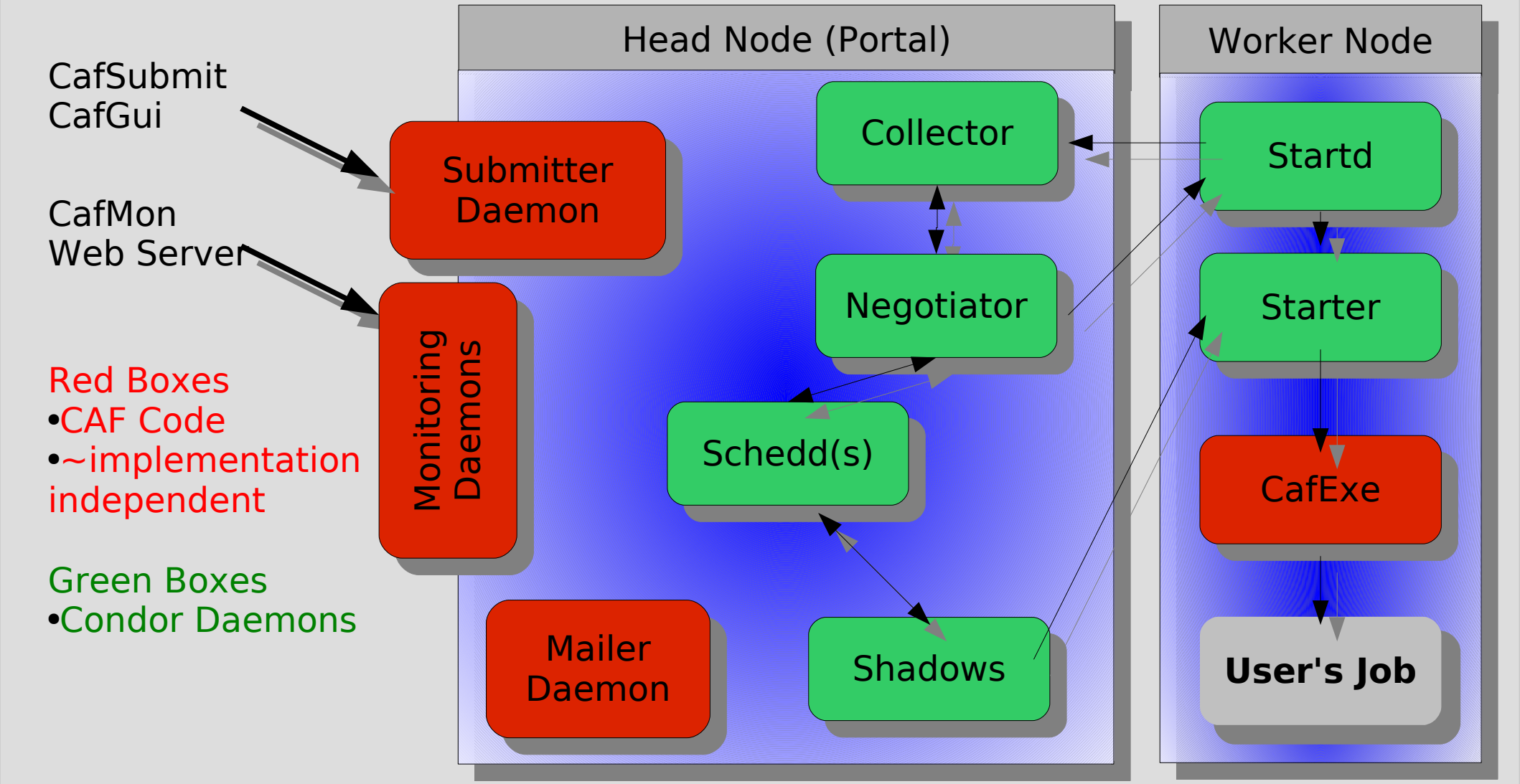
The CAF Idea

- Batch systems are complex and multifarious
- CDF users have other things to do than learn about them
- Simple interface: GUI + command line
  - No work to create a parallel job
  - Easy monitoring
- Lightweight implementation
  - Modular
  - Python daemons can be easily modified
- Flexible system has allowed us to innovate without making users relearn

Separation of Interface and Implementation

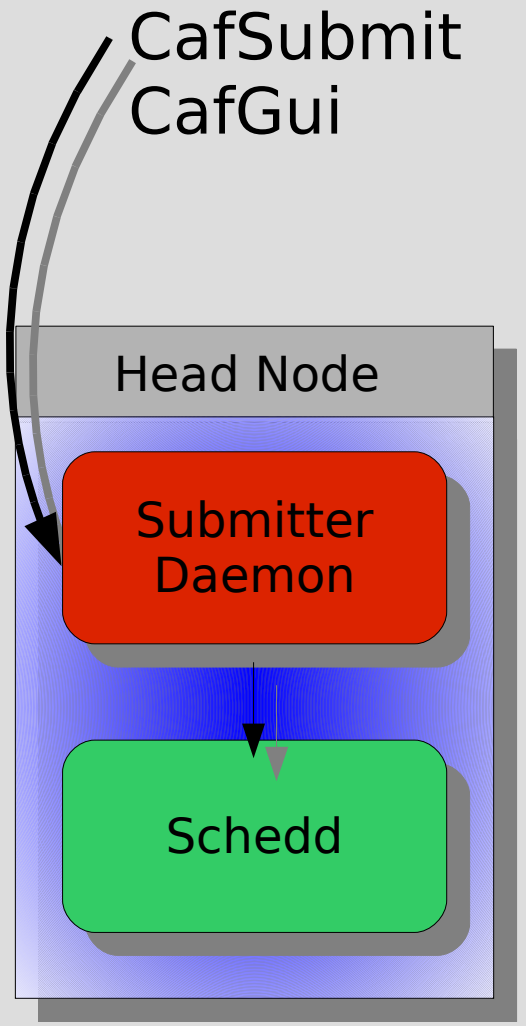
- CAF architecture separates user interface from implementation:
  - FBSNG (no longer used)
  - Condor
    - overview in this talk
    - Scaling, glide-ins, wide area glide-ins other talks
  - LCG (other talk)
- Same client tools used for all of the above

Condor Implementation: Bird's Eye View



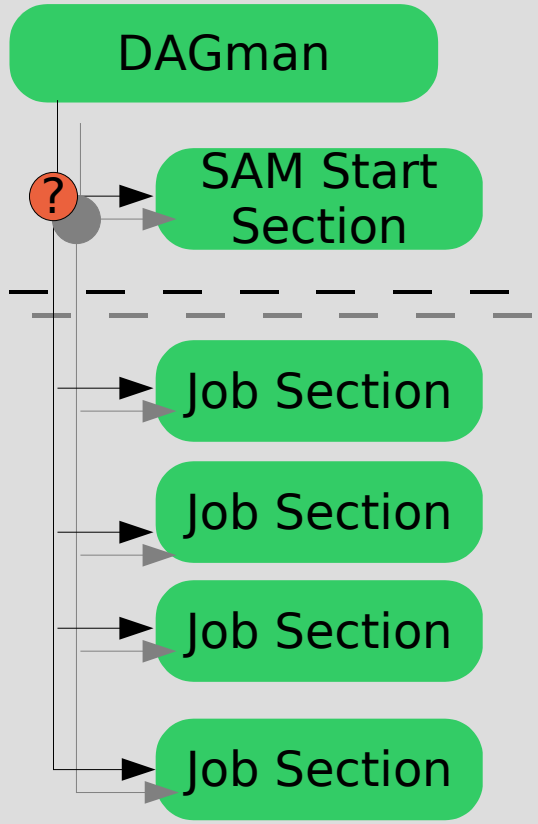
Job Submission

- CafSubmit/CafGui sends tarball + parameters to Submitter
  - via a kerberized python socket
- Submitter writes files describing job configuration
  - DAG=Direct Acyclic Graph(next slide)
    - Implements parallelism
  - ClassAds
    - Descriptions of each section of job
    - CAF sections are separate Condor jobs
- Submitter submits ClassAd for DAGman to schedd



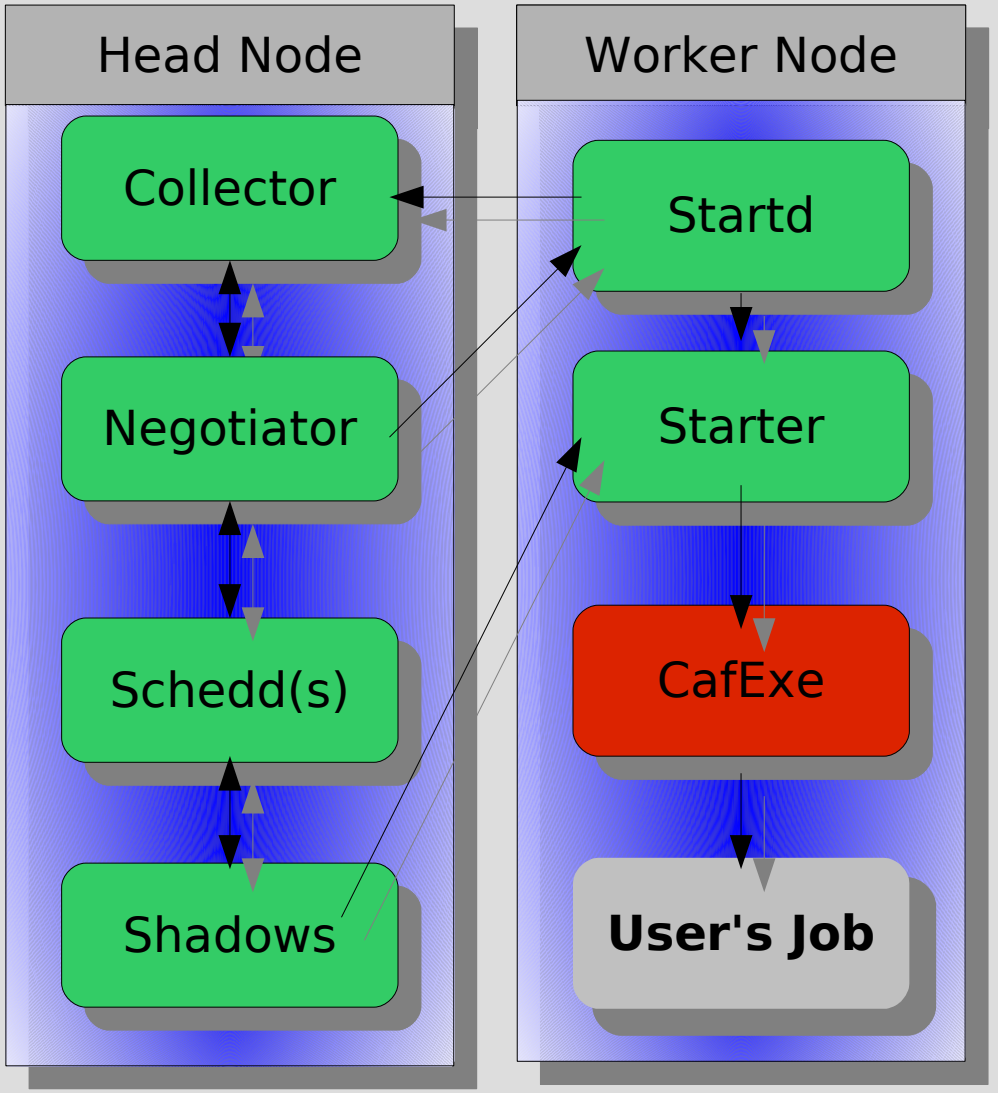
DAGMan

- A Condor “Scheduler Universe” job
  - This is jobs that manage other jobs
  - Runs on head node
- DAGman submits individual job sections automatically
  - Submission can be conditional
    - E.g. SAM start section first, other sections only if start works
- DAGman job finishes when all sections are done



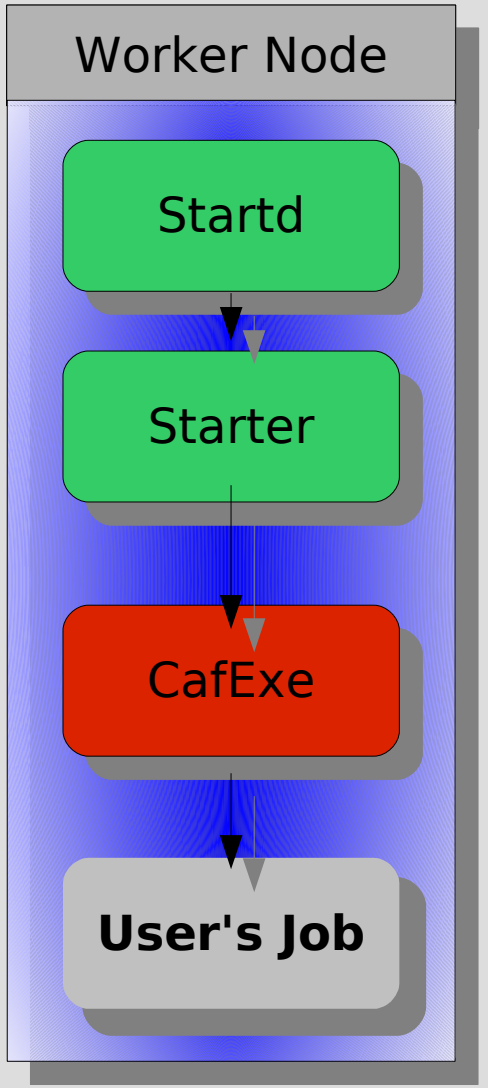
Condors Daemon Overview

- Collector has list of Virtual Machines (VMs)
- Schedd has list of jobs
- Negotiator matches jobs to VMs
- Negotiator sends “claim” to Startd on worker
- Startd starts Starter
- Shadow sends job info to Starter
- Starter starts CafExe
- CafExe starts user's job



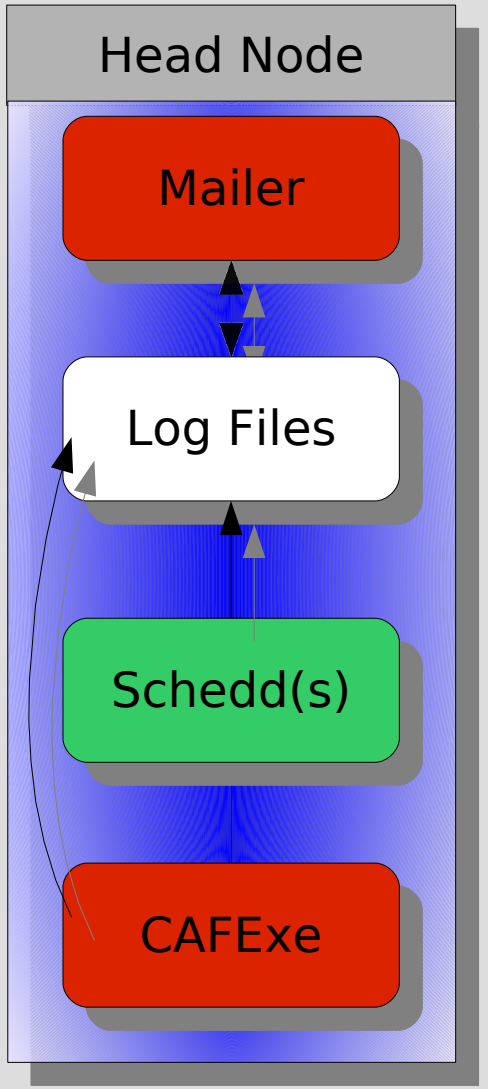
CafExe: CAF job wrapper

- The executable submitted to Condor
- Starts the actual user code
- Unpacks input tarball
- Configures the environment
- Monitors the process
- Builds the output tarball
- Transfers out the tarball
  - Lots of protocols tried
  - Falls back to lost+found ICAF area
- Adding some interface with SAM to improve the book keeping



The End Game

- When section exits, Condor transfers back CafExe logs files to head node
- Submitter keeps a file with a list of all running jobs and who gets the mail
- Mailer daemon periodically reads the job's log file to see if it's done
- Mailer builds gets jobs information from log files, builds and sends email
- Same library that does this parsing is used in the other monitoring activities



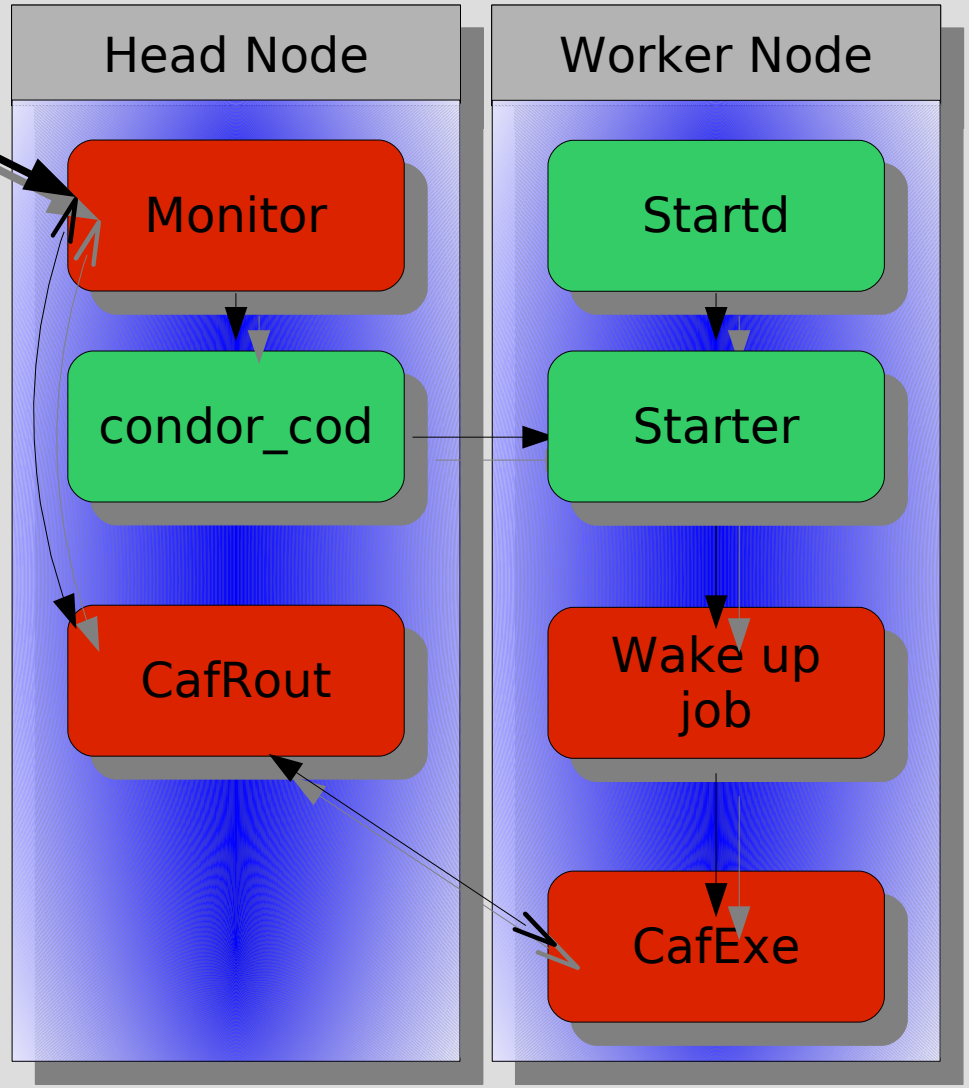
CAF-SAM Interface

- CafSubmit/CafGui
  - Check validity of dataset before submission
  - Warn user if large number of files from tape
- Start Section -> Starts a project
- CafExe sets environment variables for user
- End Section -> Stops the project
- Mailer
  - Makes sure project is stopped
  - Gets project summary (“sam dump project”)
- Being added
  - Start consumer process for user
  - Set flag for each file in project: CAF output successfully delivered

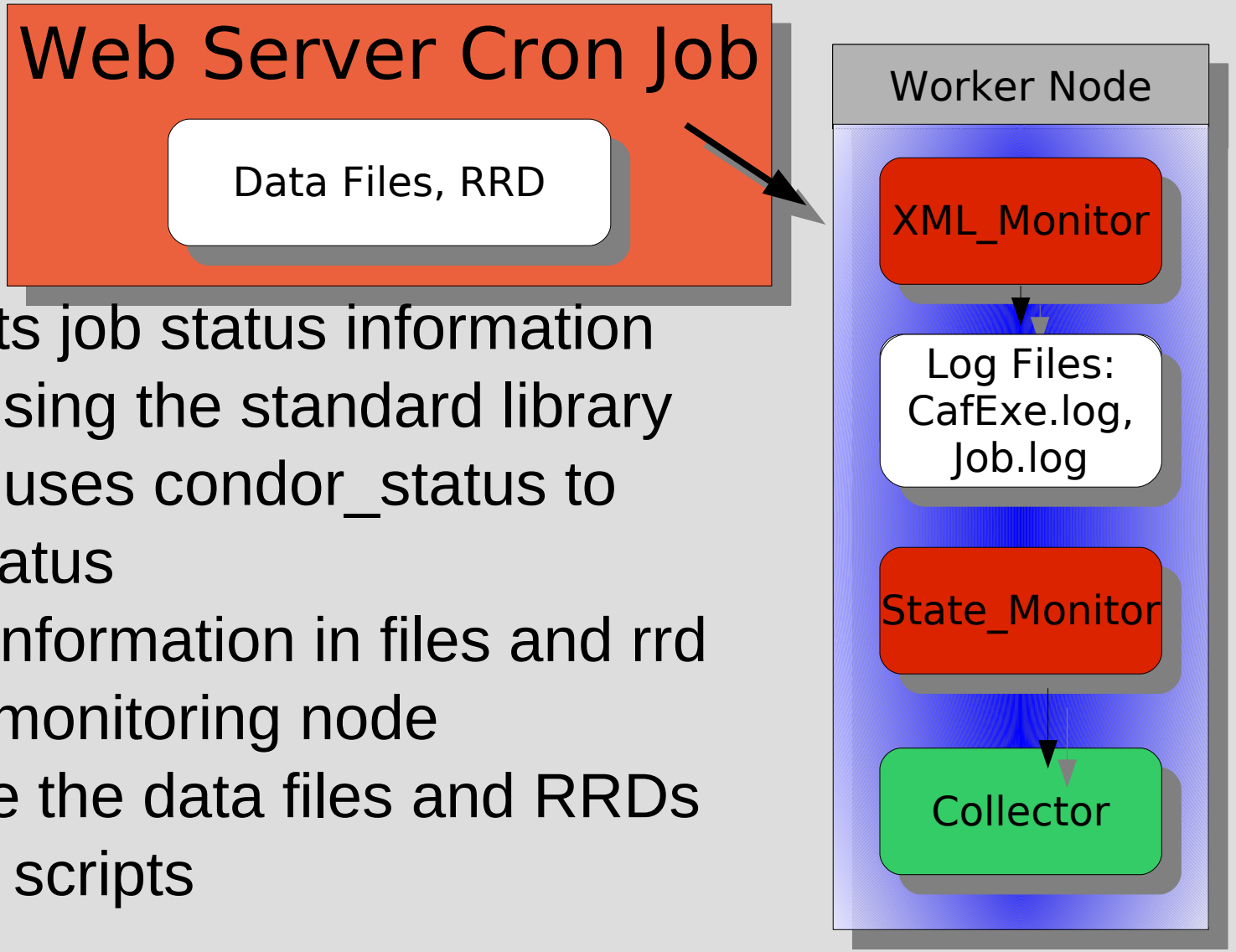
Interactive Job Monitoring

CafMon

- Establishes connection with CafRout
- Uses Condor's Computing On Demand (COD) system
  - Uses the monitoring libraries to find VM
- Monitor wakes up CafExe using COD
- CafExe calls back CafRout
- Now connection runs from CafMon to CafExe

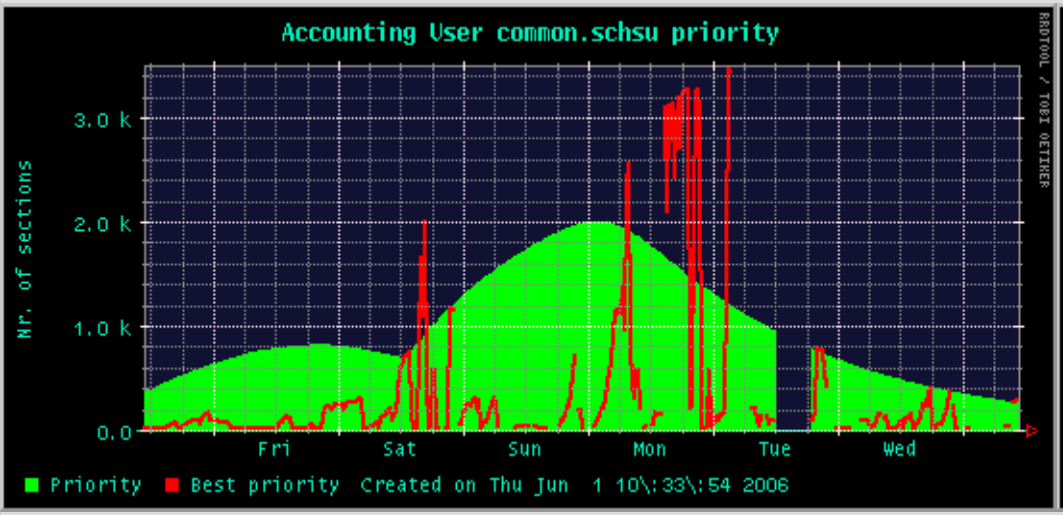


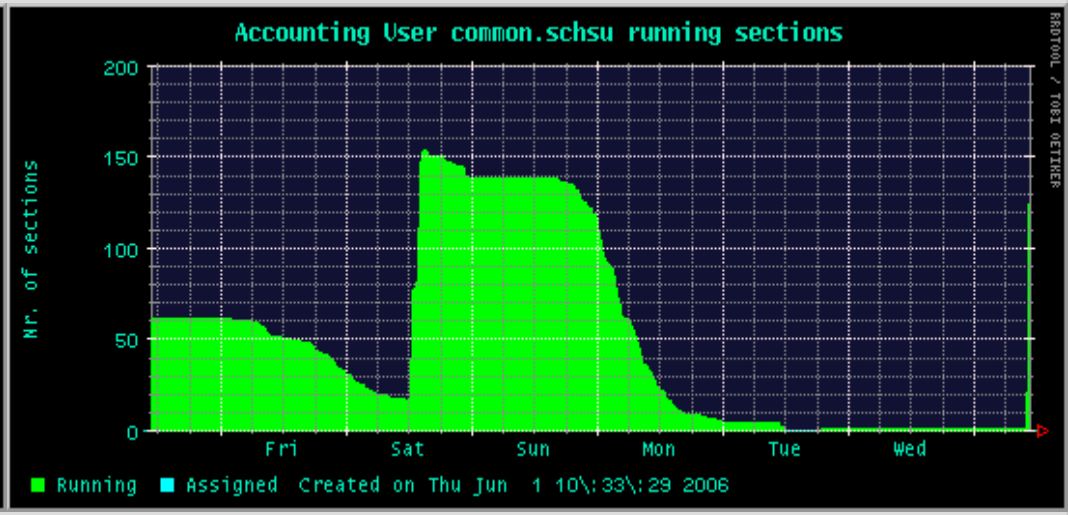
Passive Web-based Monitoring



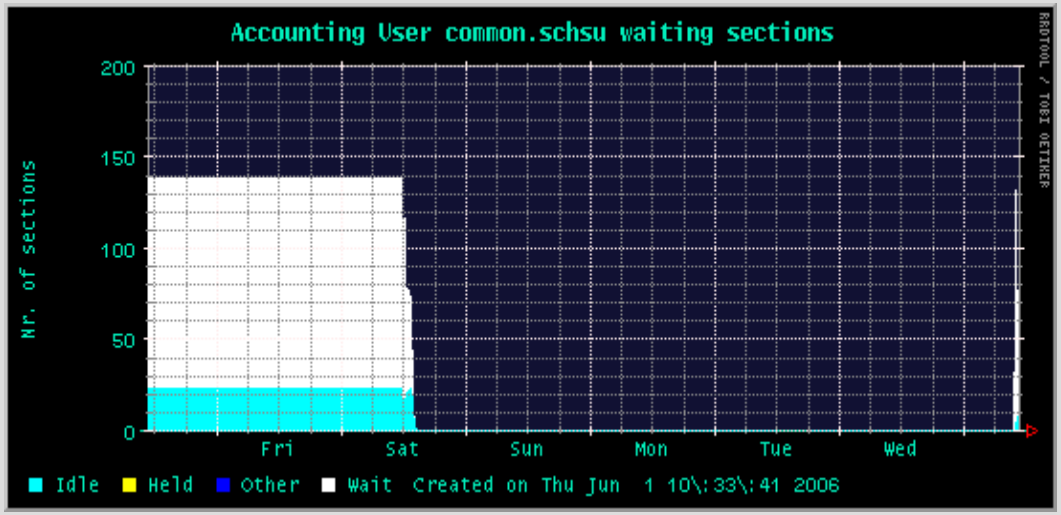
- Monitor extracts job status information from log files using the standard library
- State_Monitor uses condor_status to collect node status
- Cron jobs put information in files and rrd databases on monitoring node
- Web pages are the data files and RRDs served by CGI scripts

Condor's Priority Scheme

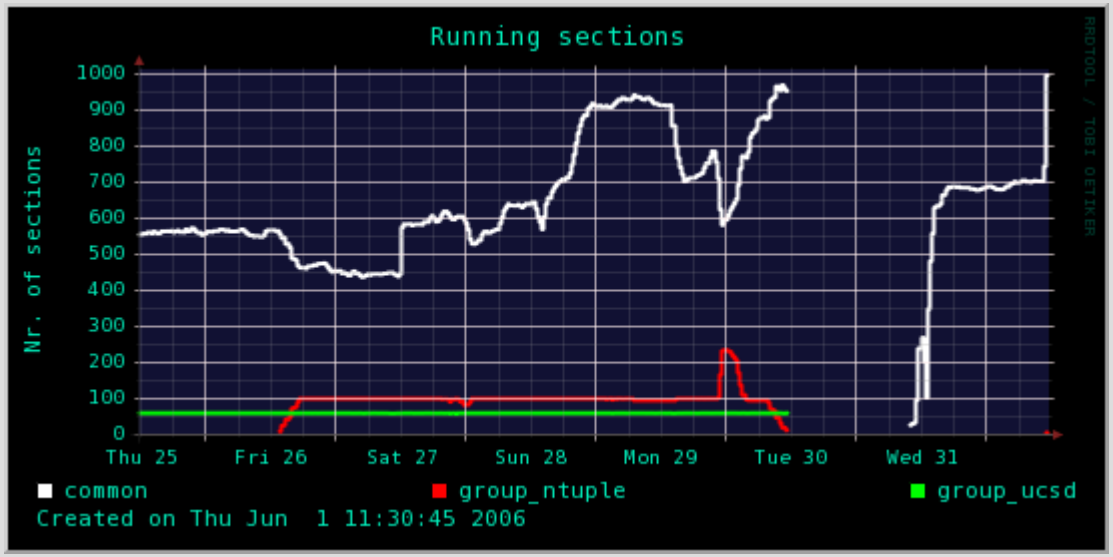




- Priority rise with CPU consumption
- Priority decreases exponentially (1 day lifetime)
- Best Priority Wins



Condor's Accounting Group Priority Scheme



- Groups have quotas VMs
- If a group needs a VM and is under its quota, it gets the next VM available
  - If multiple groups need VMs round-robin
- If there are free VMs groups can exceed their quotas
- Allows experiment make priority decisions

Summary

- CAF architecture separates user interface from implementation
- Condor implementation has been successfully in production for more than two years
- Under the hood there is lots of work going on but users usually don't have to know!