
# An Introduction to High-Throughput Computing

## Monday morning, 9:45am

Scot Kronenfeld <kronenfe@cs.wisc.edu>

OSG Software Team Member

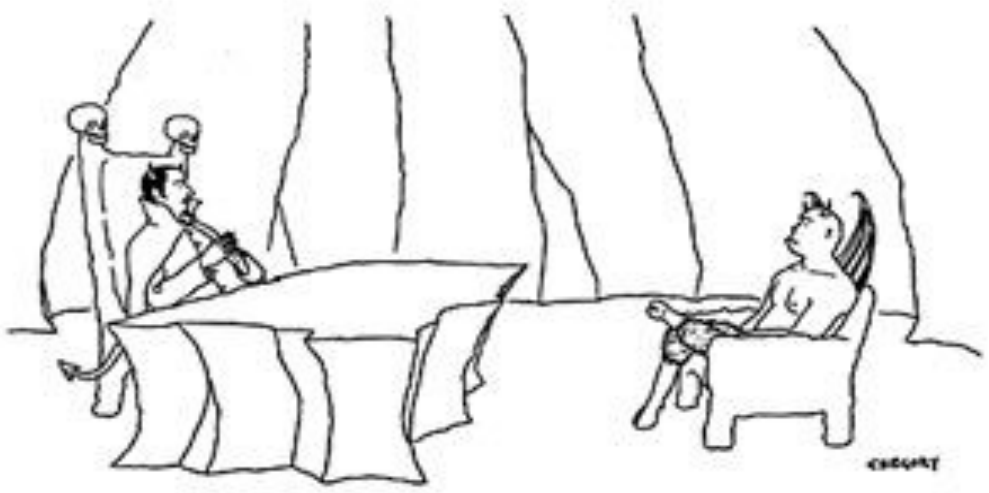
University of Wisconsin-Madison


# Who Am I?

- Open Science Grid Software Team Member
- With Condor for 5+ years

# Overview of day

- Lectures alternating with exercises
  - Emphasis on lots of exercises
  - Hopefully overcome PowerPoint fatigue & help you understand better



*"I need someone well versed in the art of torture—do you know PowerPoint?"*

Open Science Grid

# Some thoughts on the exercises

- It's okay to move ahead on exercises if you have time
- It's okay to take longer on them if you need to
- If you move along quickly, try the "On Your Own" sections and "Challenges"

Open Science Grid

# Most important!

- Please ask me questions!
  - …during the lectures
  - …during the exercises
  - …during the breaks
  - …during the meals
  - …over dinner
  - …the rest of the week
- If I don't know, I'll find the right person to answer your question.

Open Science Grid

# Before we start

- Sometime today, do the exercise on getting a certificate.
  - It is **required** for upcoming exercises
  - It will be easiest if you do it today

Open Science Grid

# Goals for this session

- Understand basics of high-throughput computing
- Understand the basics of Condor
- Run a basic Condor job

Open Science Grid

# The setup: You have a problem

- Your science computing is complex!
  - Monte carlo, image analysis, genetic algorithm, simulation…
- It will take a year to get the results on your laptop, but the conference is in a week.
- What do you do?


# Option 1: Use a "supercomputer" aka High Performance Computing (HPC)

- "Clearly, I need the best, fastest computer to help me out"
- Maybe you do…
  - Do you have a highly parallel program?
    - i.e. individual modules must communicate
  - Do you require the fastest network/disk/memory?
- Are you willing to:
  - Port your code to a special environment?
  - Request and wait for an allocation?


# Option 2: Use lots of commodity computers

- Instead of the fastest computer, lots of individual computers
- May not be fastest network/disk/memory, but you have a lot of them
- Job can be broken down into separate, independent pieces
  - If I give you more computers, you run more jobs
  - You care more about total quantity of results than instantaneous speed of computation
- This is **high-throughput computing**

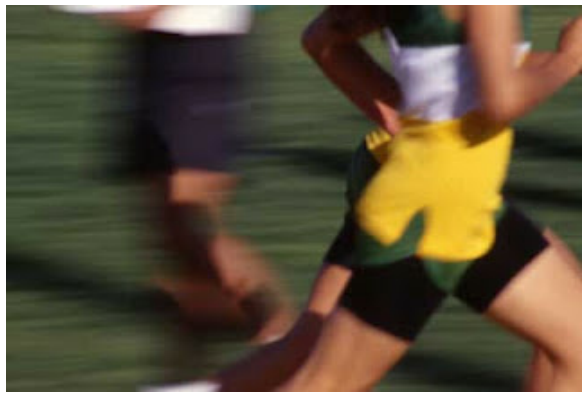
# What is high-throughput computing? (HTC)

- An approach to distributed computing that focuses on long-term throughput, not instantaneous computing power
  - We don't care about operations per second
  - We care about operations per year
- Implications:
  - Focus on reliability
  - Use all available resources
    - That slow four-year old cluster down the hall? Use it!

Open Science Grid

# Think about a race

- Assume you can run a four minute mile
- Does that mean you can run a 104 minute marathon?
- The challenges in sustained computation are different than achieving peak in computation speed
  - Our focus is sustained computation



Open Science Grid

# HTC is not for all problems

- Do you need real time results?
- Do you need to minimize latency to results?
- Do you have very parallel code with lots of communication between modules?

- You might need HPC instead

Open Science Grid

# An example problem: BLAST

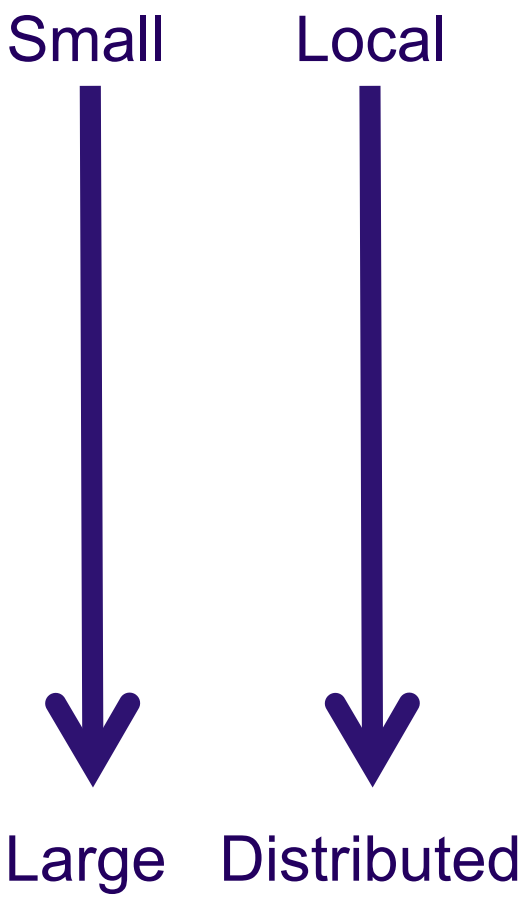
- A scientist has:
  - Question: Does a protein sequence occur in other organisms?
  - Data: lots of protein sequences from various organisms
  - Parameters: how to search the database.
- More throughput means
  - More protein sequences queried
  - Larger/more protein data bases examined
  - More parameter variation
- We'll try out BLAST later today

Open Science Grid

# Why is HTC hard?

- The HTC system has to keep track of:
  - Individual tasks  (a.k.a. jobs) & their inputs
  - Computers that are available
- The system has to recover from failures
  - There will be failures! Distributed computers means more chances for failures.
- You have to share computers
    - Sharing can be within an organization, or between orgs
    - So you have to worry about security
    - And you have to worry about policies on how you share
- If you use a lot of computers, you have to handle variety:
  - Different kinds of computers (arch, OS, speed, etc..)
  - Different kinds of storage (access methodology, size, speed, etc…)
  - Different networks interacting (network problems are hard to debug!)

# Let's take one step at a time




- Can you run one job on one computer?
- Can you run one job on another local computer?
- Can you run 10 jobs on a set of local computers?
- Can you run 1 job on a remote computer?
- Can you run 10 jobs at a remote site?
- Can you run a mix of jobs here and remotely?

This is the path we'll take in the school this week

Open Science Grid

# Discussion

- For 5 minutes, talk to a neighbor: If you want to run one job in a local cluster of computers:
  1) What do you (the user) need to provide so a single job can be run?
  2) What does the system need to provide so your single job can be run?
     - Think of this as a set of processes: what needs happen when the job is given? A "process" could be a computer process, or just an abstract task.


# Scot's answer: What does the user provide?

- A "headless job"
  - Not interactive/no GUI: how could you interact with 1000 simultaneous jobs?
- A set of input files
- A set of output files
- A set of parameters (command-line arguments)
- Requirements:
  - Ex: My job requires at least 2GB of RAM
  - Ex: My job requires Linux
- Control/Policy:
  - Ex: Send me email when the job is done
  - Ex: Job 2 is more important than Job 1
  - Ex: Kill my job if it runs for more than 6 hours


# Scot's answer: What does the system provide?

- Methods to:
  - Submit/Cancel job
  - Check on state of job
  - Check on state of available computers
- Processes to:
  - Reliably track set of submitted jobs
  - Reliably track set of available computers
  - Decide which job runs on which computer
  - Manage a single computer
  - Start up a single job


# Surprise! Condor does this (and more)

- Methods to:
  - Submit/Cancel job. `condor_submit/condor_rm`
  - Check on state of job. `condor_q`
  - Check on state of avail. computers. `condor_status`
- Processes to:
  - Reliably track set of submitted jobs. `schedd`
  - Reliably track set of avail. computers. `collector`
  - Decide which job runs on where. `negotiator`
  - Manage a single computer `startd`
  - Start up a single job `starter`


# But not only Condor

- You can use other systems:
  - PBS/Torque
  - Oracle Grid Engine (né Sun Grid Engine)
  - LSF
  - ...
- But I won't cover them.
  - Our expertise is with Condor
  - Our bias is with Condor
- What should you learn at the school?
  - How do you think about HTC?
  - How can you do your science with HTC?
  - … For now, learn it with Condor, but you can apply it to other systems.


# A brief introduction to Condor

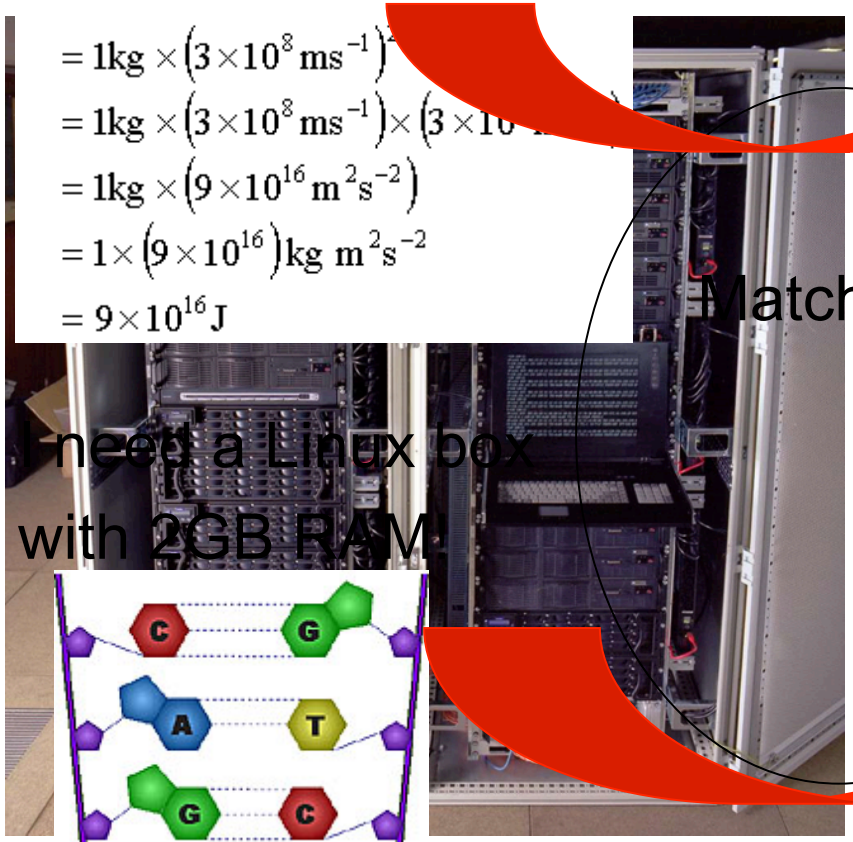
- Please note, we will only scratch the surface of Condor:
  - We won't cover MPI, Master-Worker, advanced policies, site administration, security mechanisms, Condor-C, submission to other batch systems, virtual machines, cron, high-availability, computing on demand, …
- Why?
  - The goal is *to introduce you to HTC* by using Condor, not to make you Condor experts.

# ...And Gabber Takes Computers...

I need a Mac!

Desktop Computers

$$E = mc^2$$
$$= 1\text{kg} \times \left(3\times10^{8}\,\text{ms}^{-1}\right)^{2}$$
$$= 1\text{kg} \times \left(3\times10^{8}\,\text{ms}^{-1}\right) \times \left(3\times10^{8}\,\text{ms}^{-1}\right)$$
$$= 1\text{kg} \times \left(9\times10^{16}\,\text{m}^{2}\text{s}^{-2}\right)$$
$$= 1\times\left(9\times10^{16}\right)\text{kg m}^{2}\text{s}^{-2}$$
$$= 9\times10^{16}\,\text{J}$$



I need a Linux box with 2GB RAM!

Open Science Grid

# Quick Terminology

- Cluster: A dedicated set of computers not for interactive use
- Pool: A collection of computers used by Condor
  - May be dedicated
  - May be interactive
- Remember:
  - Condor can manage a cluster in a machine room
  - Condor can use desktop computers
  - Condor can access remote computers
  - HTC uses for all available resources


# Matchmaking

- Matchmaking is fundamental to Condor
- Matchmaking is two-way
  - Job describes what it requires:

    I need Linux && 8 GB of RAM
  - Machine describes what it requires:

    I will only run jobs from the Physics department
- Matchmaking allows preferences
  - I need Linux, and I prefer machines with more memory but will run on any machine you provide me

# Why Two-way Matching?

- Condor conceptually divides people into three groups:
  - Job submitters
  - Computer owners
  - Pool (cluster) administrator

} May or may not be the same people

- All three of these groups have preferences

Open Science Grid

# ClassAds

- ClassAds state facts
  - My job's executable is analysis.exe
  - My machine's load average is 5.6
- ClassAds state preferences
  - I require a computer with Linux
- ClassAds are extensible
  - They say whatever you want them to say

Open Science Grid

# Example ClassAd

```
MyType         = "Job"          <-- String
TargetType     = "Machine"
ClusterId      = 1377           <-- Number
Owner          = "roy"
Cmd            = "analysis.exe"
Requirements =                  <-- Expression
    (Arch == "INTEL")
&& (OpSys == "LINUX")
&& (Disk >= DiskUsage)
&& ((Memory * 1024)>=ImageSize)
…
```

Open Science Grid

# Schema-free ClassAds

- Condor imposes some schema
  - Owner is a string, ClusterID is a number…
- But users can extend it however they like, for jobs or machines
  - `AnalysisJobType = "simulation"`
  - `HasJava_1_4 = TRUE`
  - `ShoeLength = 7`
- Matchmaking can use these attributes
  - `Requirements = OpSys == "LINUX"`
    `&& HasJava_1_4 == TRUE`

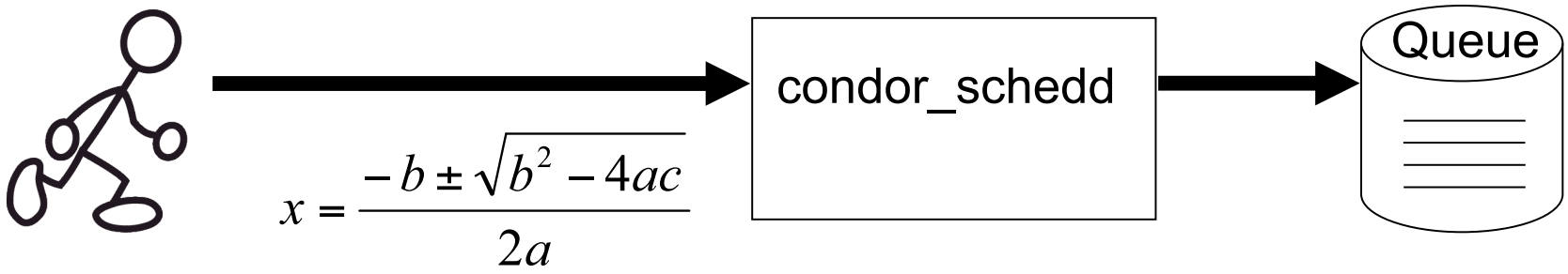
Open Science Grid

# Don't worry

- You won't write ClassAds (usually)
  - You'll create a simple *submit file*
  - Condor will write the ClassAd
  - You can extend the ClassAd if you want to
- You won't write requirements (usually)
  - Condor writes them for you
  - You can extend them
  - In some environments (see glideins tomorrow) you provide attributes instead of requirements expressions

# Submitting jobs: condor_schedd

- Users submit jobs from a computer
  - Jobs described as ClassAds
  - Each submission computer has a queue
  - Queues are not centralized
  - Submission computer watches over queue
  - Can have multiple submission computers
  - Submission handled by *condor_schedd*



# Advertising computers

- Machine owners describe computers
  - Configuration file extends ClassAd
  - ClassAd has dynamic features
    - Load Average
    - Free Memory
    - …
  - ClassAds are sent to Matchmaker by *condor_startd* on computer

ClassAd

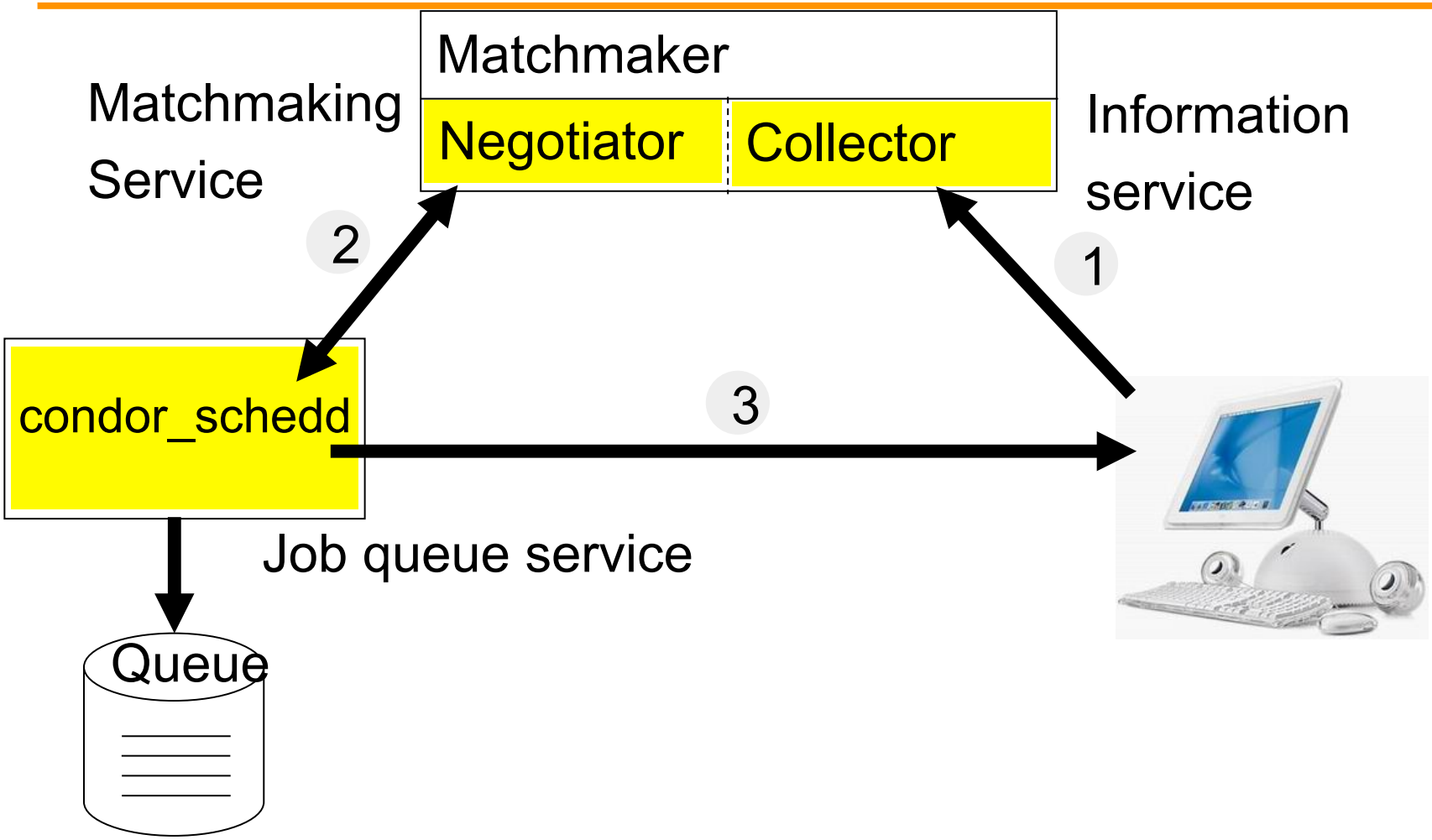
```
Type = "Machine"
Requirements = "…"
```

Matchmaker (Collector)

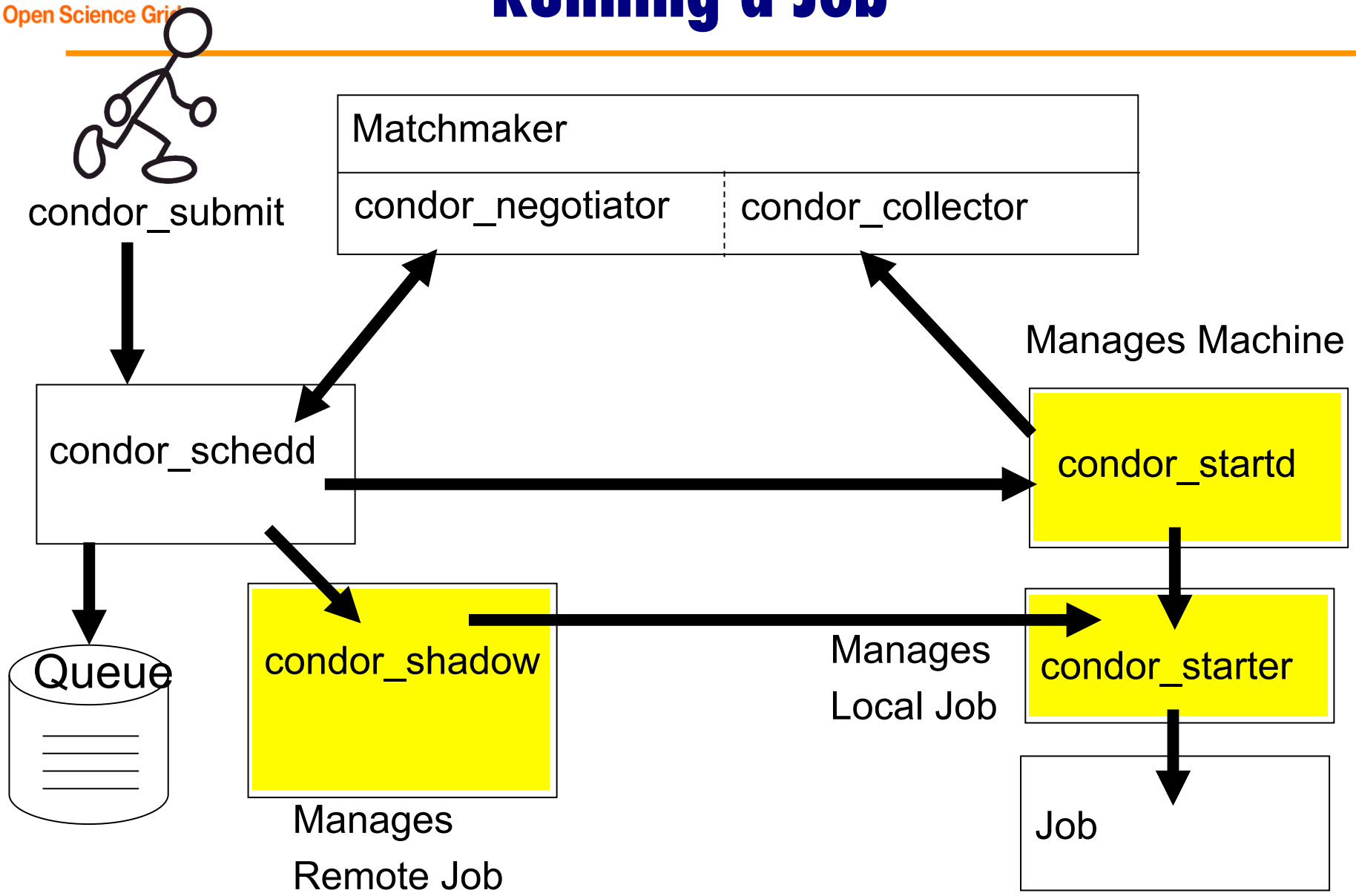

# Matchmaking

- Negotiator collects list of computers
- Negotiator contacts each schedd
  - What jobs do you have to run?
- Negotiator compares each job to each computer
  - Evaluate requirements of job & machine
  - If both are satisfied, there is a match
- Upon match, schedd contacts execution computer to run job

# Matchmaking diagram



# Running a Job




# Condor processes

| Process | Function |
|---|---|
| Master | Takes care of other processes |
| Collector | Stores ClassAds |
| Negotiator | Performs Matchmaking |
| Schedd | Manages job queue |
| Shadow | Manages job (submit side) |
| Startd | Manages computer |
| Starter | Manages job (execution side) |


# If you forget most of these remember two (for other lectures)

| Process | Function |
|---|---|
| Master | Takes care of other processes |
| Collector | Stores ClassAds |
| Negotiator | Performs Matchmaking |
| **Schedd** | **Manages job queue** |
| Shadow | Manages job (submit side) |
| **Startd** | **Manages computer** |
| Starter | Manages job (execution side) |

Open Science Grid

# Some notes

- One negotiator/collector per pool
  (Perhaps extras for reliability)
- Can have many schedds (submitters)
- Can have many startds (computers)
- A machine can play multiple roles
  - E.g. Can have both schedd & startd


# Today's Condor setup

- One submit computer
  - One schedd/queue for everyone
  - osg-ss-submit.chtc.wisc.edu
- Will use local Condor pool
  - Have dedicated subset
- Tomorrow we will expand the setup

# That was a whirlwind tour!

- Enough with the theory: let's use Condor!

- Goal: Check out our installation, run some basic jobs.

Open Science Grid

# Questions?

- Questions? Comments?
  - Feel free to ask me questions later:

    Scot Kronenfeld <kronenfe@cs.wisc.edu>
- Upcoming sessions
  - Now – 11:15
    - Hands-on exercises
  - 11:15 – 11:30
    - Break
  - 11:30 – 12:15
    - More!