Introduction to Roll Development

Rocks-A-Palooza II

Rocks Philosophy

- We've developed a "cluster compiler"
  - XML framework + XML parser + kickstart file generator
  - Source code + preprocessor + linker
- Think about "programming your cluster"
  - Not "administering your cluster"

Goal of Rolls

- Develop a method to reliably install software on a frontend
- “User-customizable” frontends
- Two established approaches:
  - Add-on method
  - Rocks method

Add-on Method

1. User responsible for installing and configuring base software stack on a frontend
2. After the frontend installation, the user downloads ‘add-on’ packages
3. User installs and configures add-on packages
4. User installs compute nodes

Major issue with add-on method

- The state of the frontend before the add-on packages are added/configured is unknown

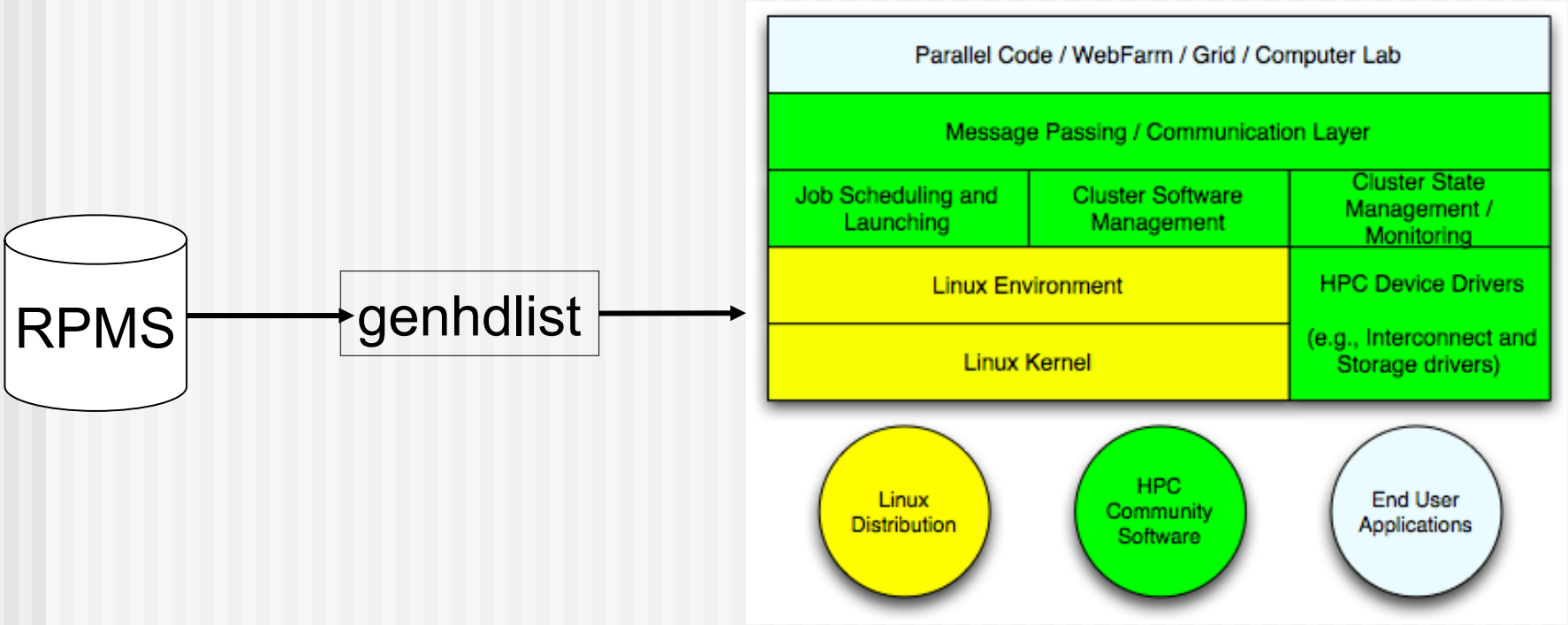
Rocks Method

- To address the major problem with the add-on method, we had the following idea:
  - All non-RedHat packages must be installed and configured in a controlled environment
- A controlled environment has a known state
- We chose the RedHat installation environment for the controlled environment

Goal of Rolls

- This led to modifying the standard RedHat installer in order to accept new packages and configuration
- A tricky proposition
  - A RedHat distribution is a monolithic entity
    - It's tightly-coupled
    - A program called "genhdlist" creates binary files (hdlist and hdlist2) that contain metadata about every RPM in the distribution
- To add/remove/change an RPM, you need to re-run genhdlist
  - Else, the RedHat install will not recognize the package
  - Or worse, it fails during package installation

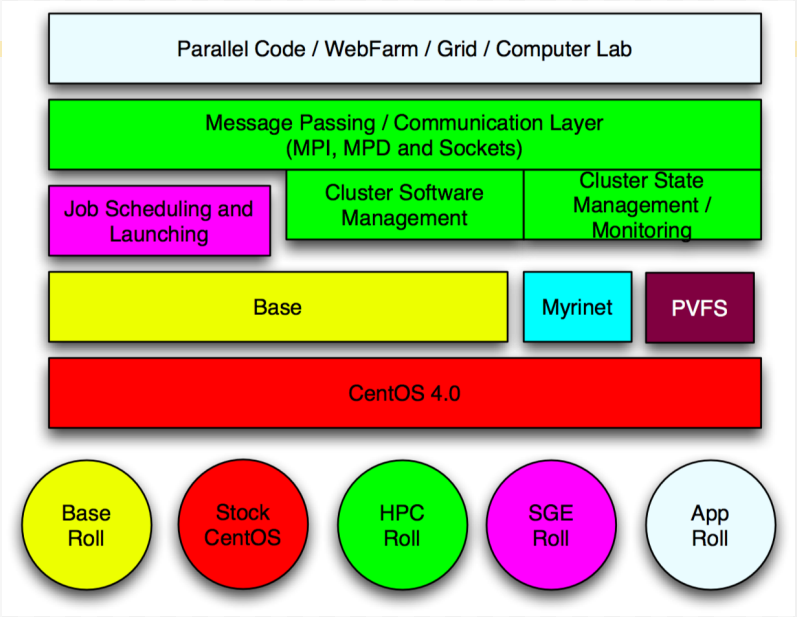
Monolithic Software Stack



Goal of Rolls

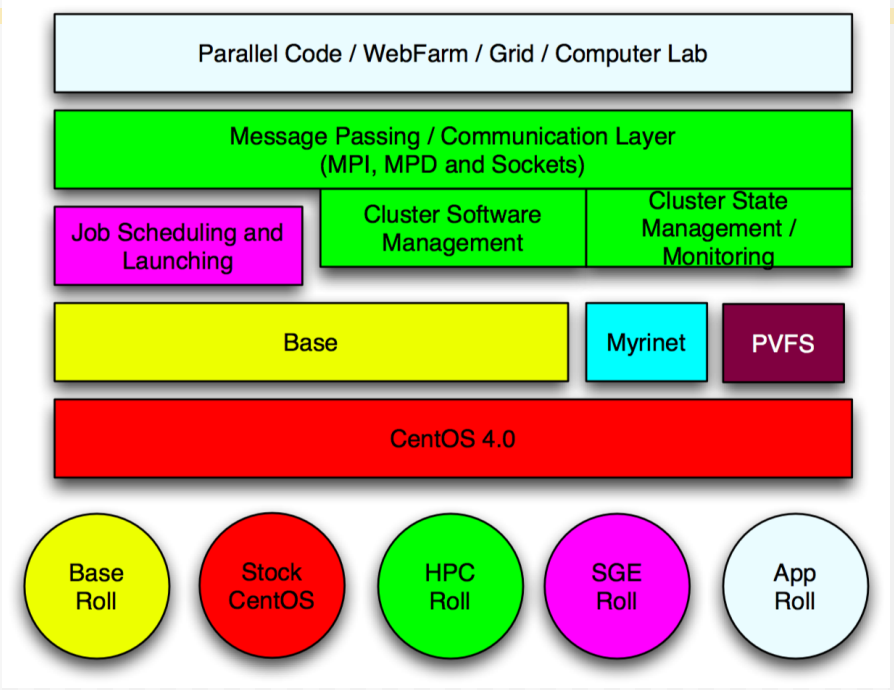
- Problem: To make the frontend user-customizable at installation time, we needed a mechanism that could accept new packages
- And, we still wanted to leverage the RedHat installer
  - We don't want to be in the installer business
- Solution: Our implementation makes the RedHat installer "think" it is just installing a monolithic RedHat distribution

Goal of Rolls



- How do you make all the packages above look like a monolithic distribution?
  - Easy! Just run “genhdlist” at release time!
- But, how do you do it when some of the above blocks are optional and/or unknown?
  - An “unknown” block is one produced after the release or by a third-party

Rolls Function and Value



- Function: Rolls extend/modify stock RedHat
- Value: Third parties can extend/modify Rocks
  - Because Rolls can be optional

The RedHat Installer

Anaconda: RedHat's Installer

- Open-source python-based installer
- Developed by RedHat
- (Somewhat) object-oriented
  - We extend when we can and insert "shims" when we can't

Anaconda: RedHat's Installer

- Key tasks:
  - Probe hardware
  - Ask users for site-specific values
    - E.g., IP addresses and passwords
  - Insert network and storage drivers
    - For network-based installations and to write packages down onto local disk
  - Install packages
    - RPMs
  - Configure services
    - Via shell scripts

Scripted Installation

- Anaconda achieves “lights-out” installation via kickstart mechanism
- It reads a “kickstart file”
  - Description of how to install a node
- One file composed of three key sections:
  - Main: general parameters
  - Packages: list of RPMs to install
  - Post: scripts to configure services

Kickstart File

- Main section

```
rootpw --iscrypted loijgoij5478fj2i9a
zerombr yes
bootloader --location=mbr
lang en_US
langsupport --default en_US
keyboard us
mouse genericps/2
text
install
reboot
timezone --utc America/Los_Angeles
part
```

Kickstart File

- Packages section

```
%packages --ignoredeps --ignoremissing
@Base
PyXML
atlas
autofs
bc
chkrootkit
contrib-pexpect
contrib-pvfs-config
contrib-python-openssl
```

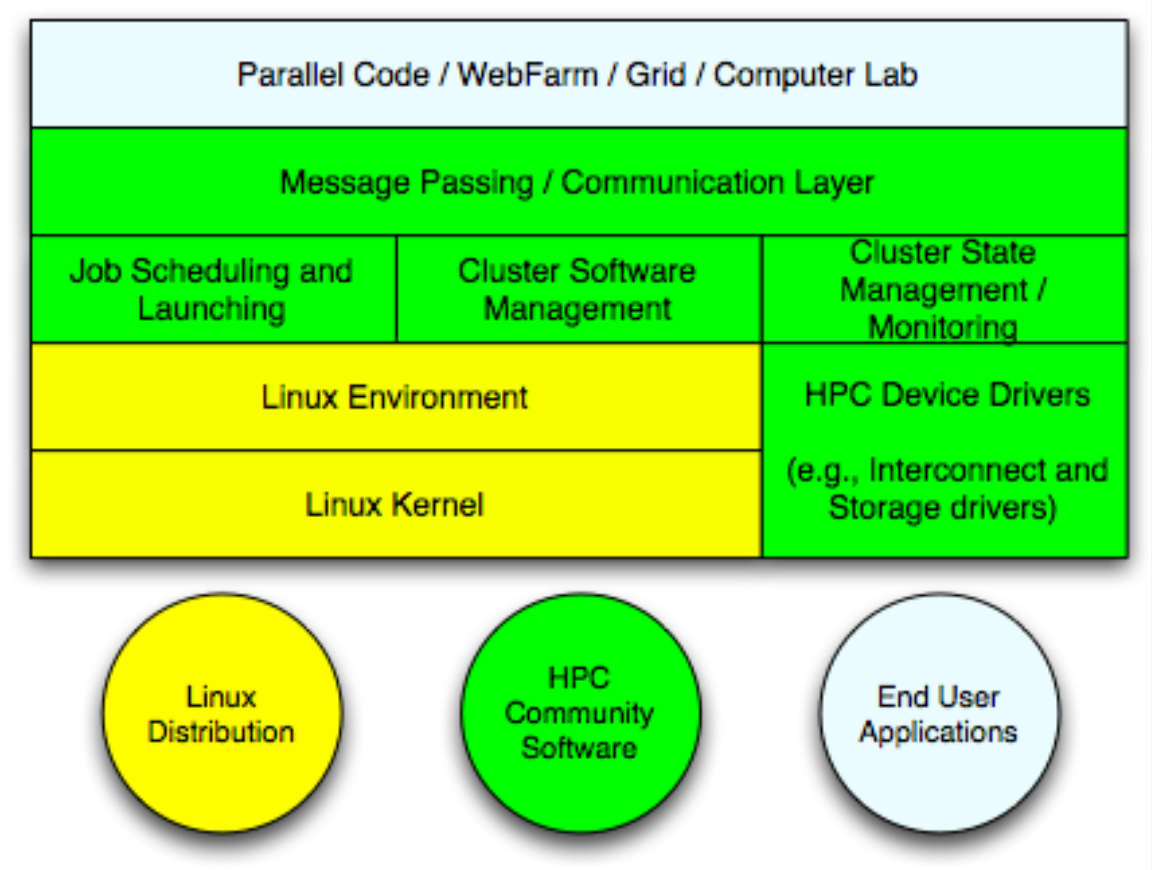
Kickstart File

- Post section

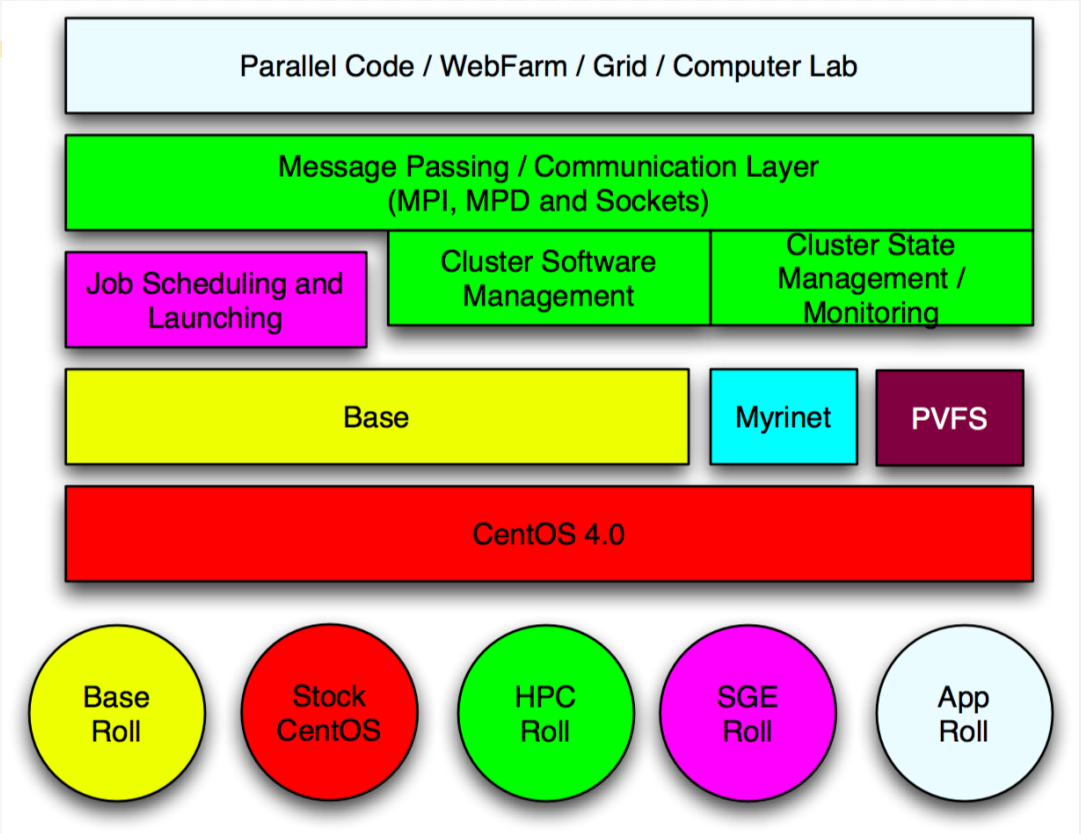
```
%post

cat > /etc/motd << 'EOF'
Rocks Compute Node
EOF
```

Rolls High-Level Description

Monolithic Software Stack



Rolls

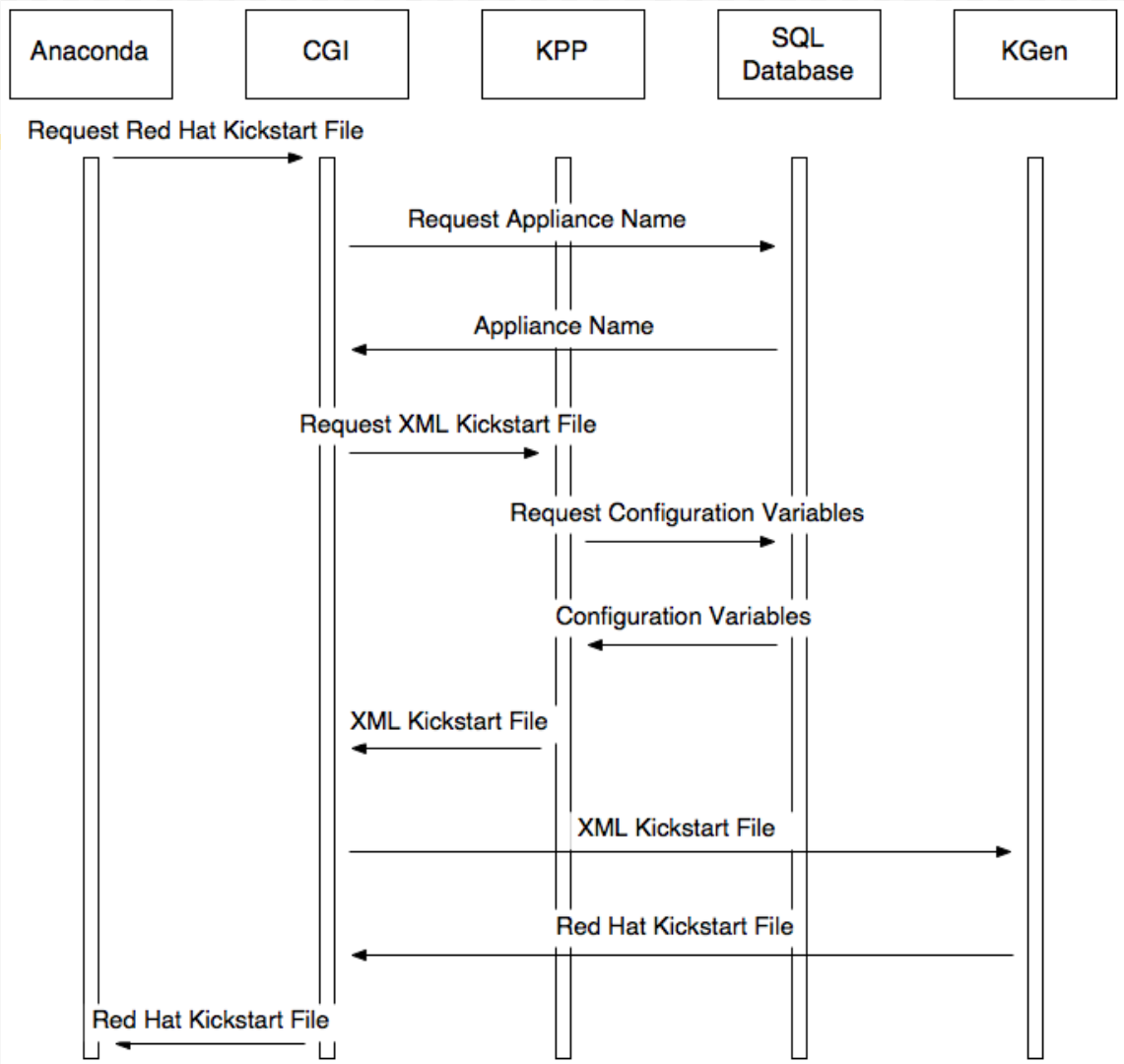


- Dissecting the monolithic software stack

Rolls

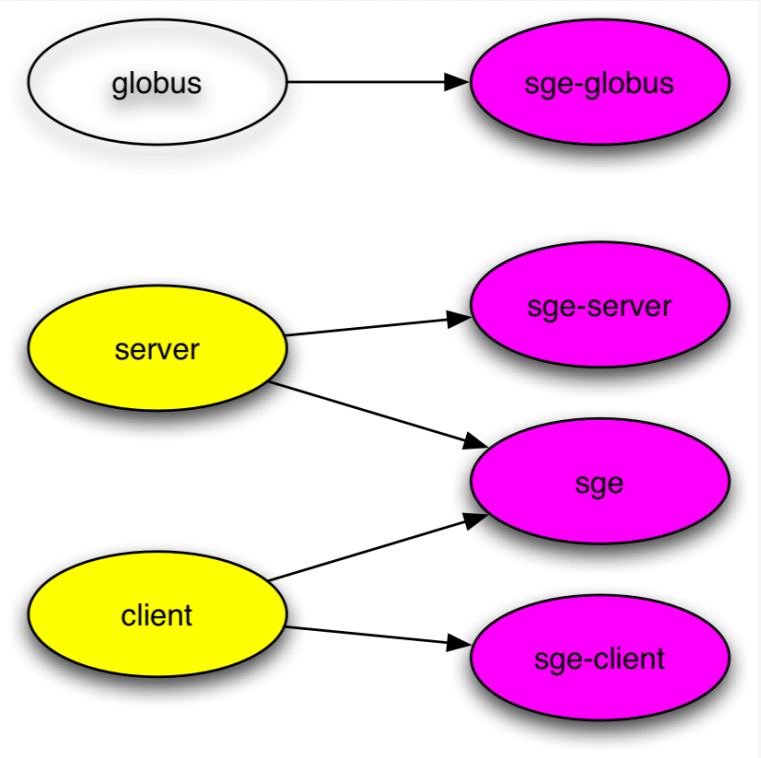
- Think of a roll as a “package” on a car

Getting A Kickstart File

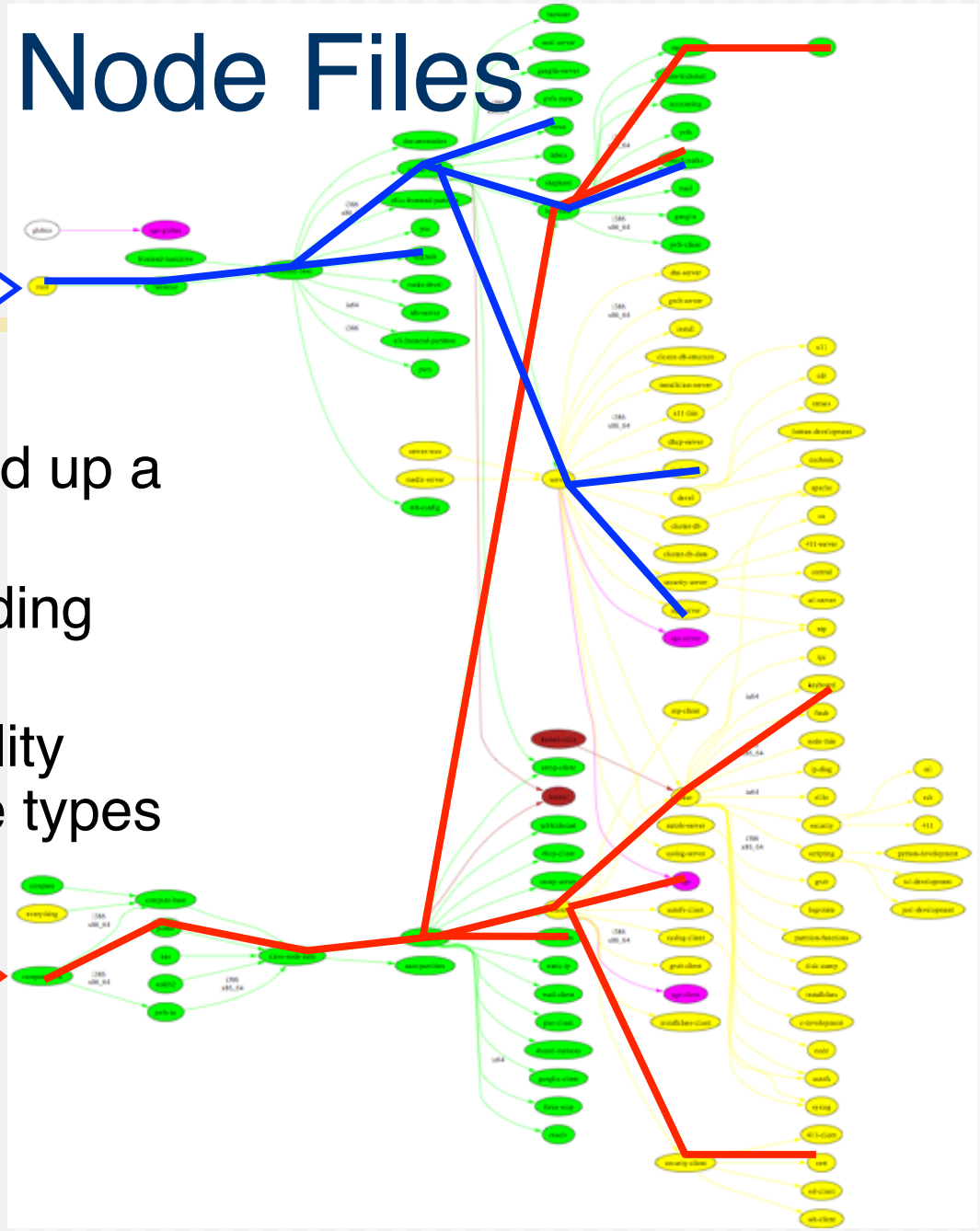


Use Graph Structure to Dissect Distribution

- Use ‘nodes’ and ‘edges’ to build a customized kickstart file
- Nodes contain portion of kickstart file
  - Can have a ‘main’, ‘package’ and ‘post’ section in node file
- Edges used to coalesce node files into one kickstart file



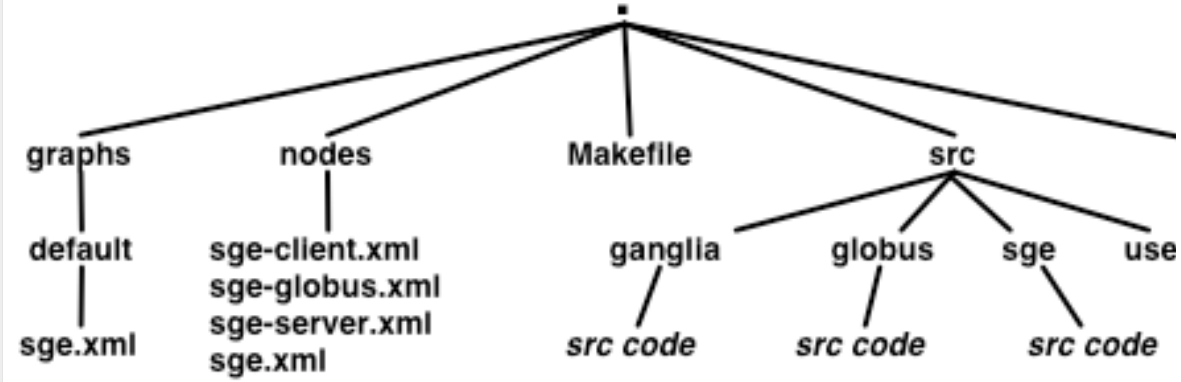
Coalescing Node Files

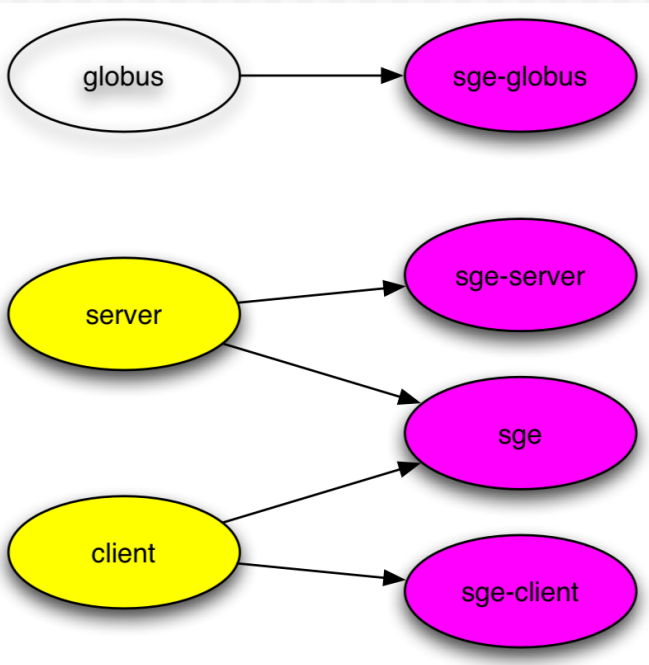


- Traverse a graph to build up a kickstart file
- Makes kickstart file building flexible
- Easy to share functionality between disparate node types

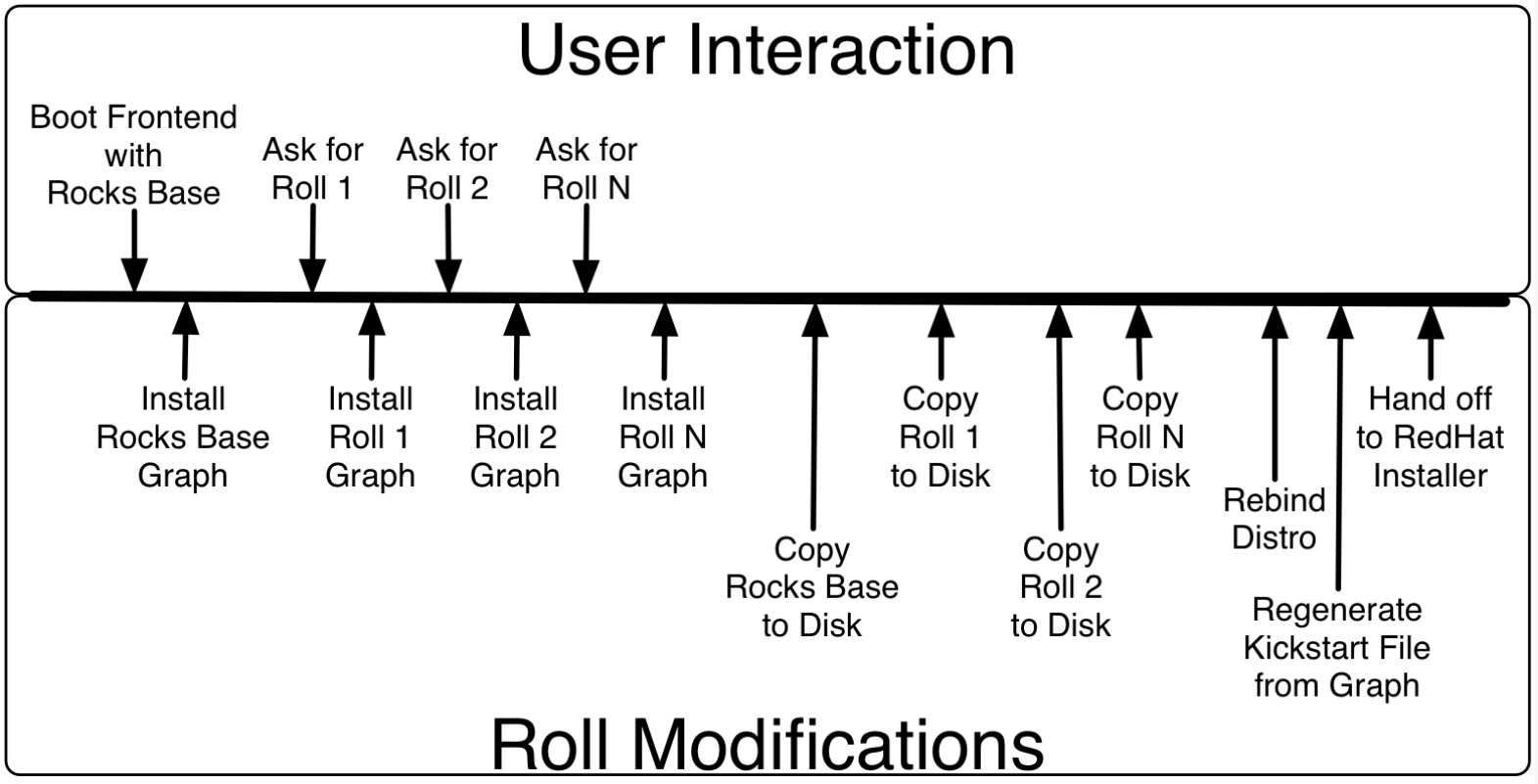
Why We Use A Graph

- A graph makes it easy to ‘splice’ in new nodes
- Each Roll contains its own nodes and splices them into the system graph file

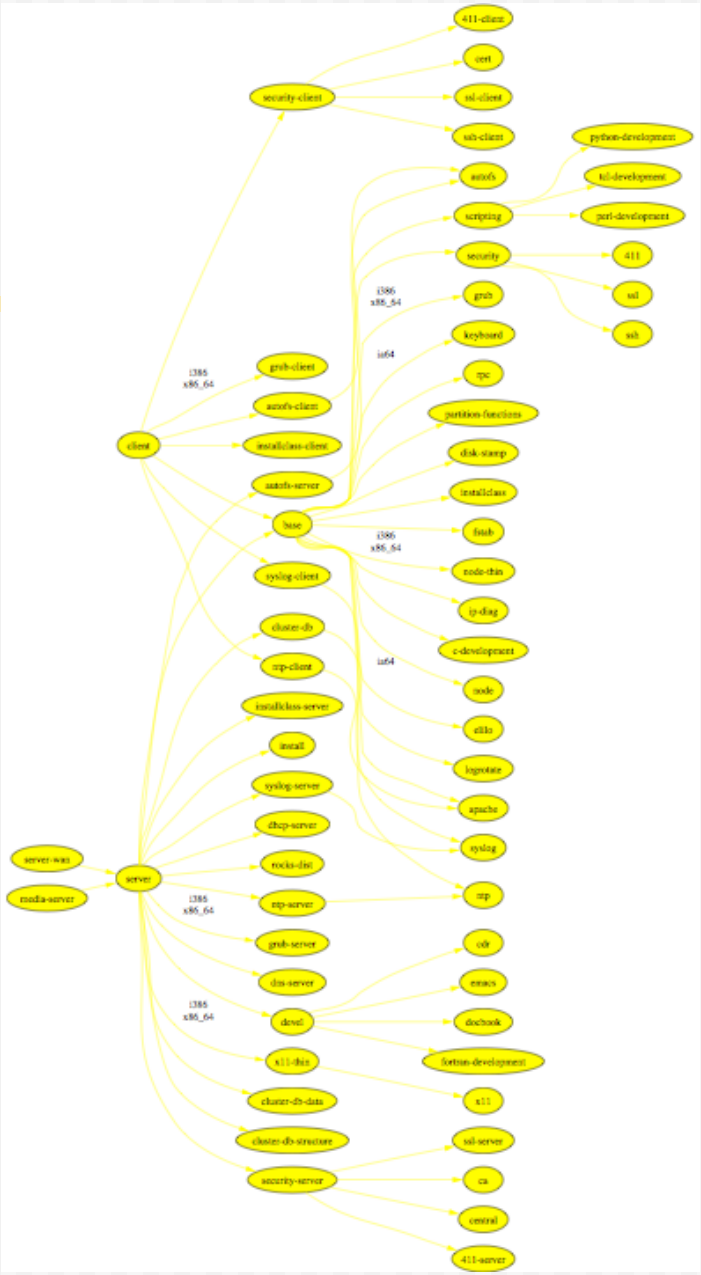




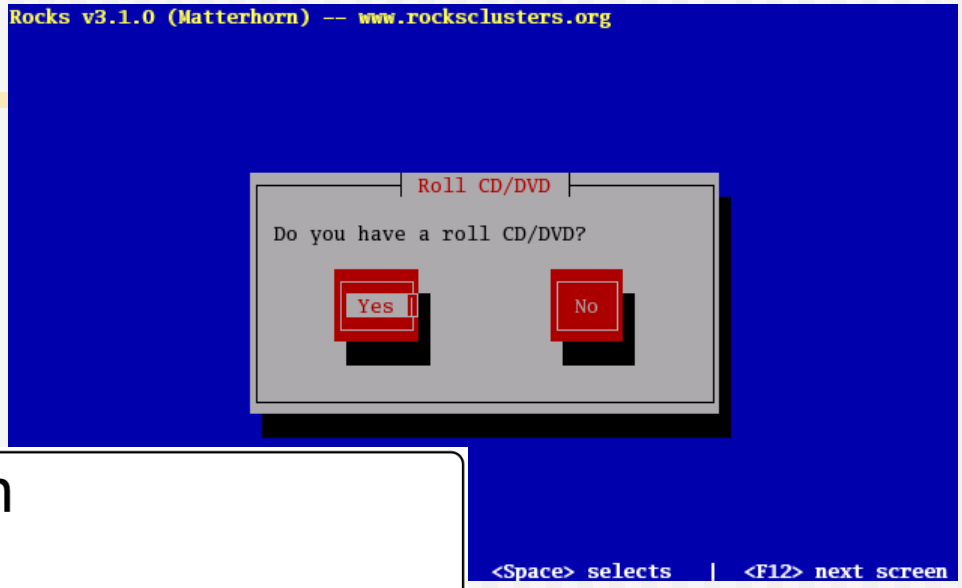
Rocks Extensions Installation Timeline

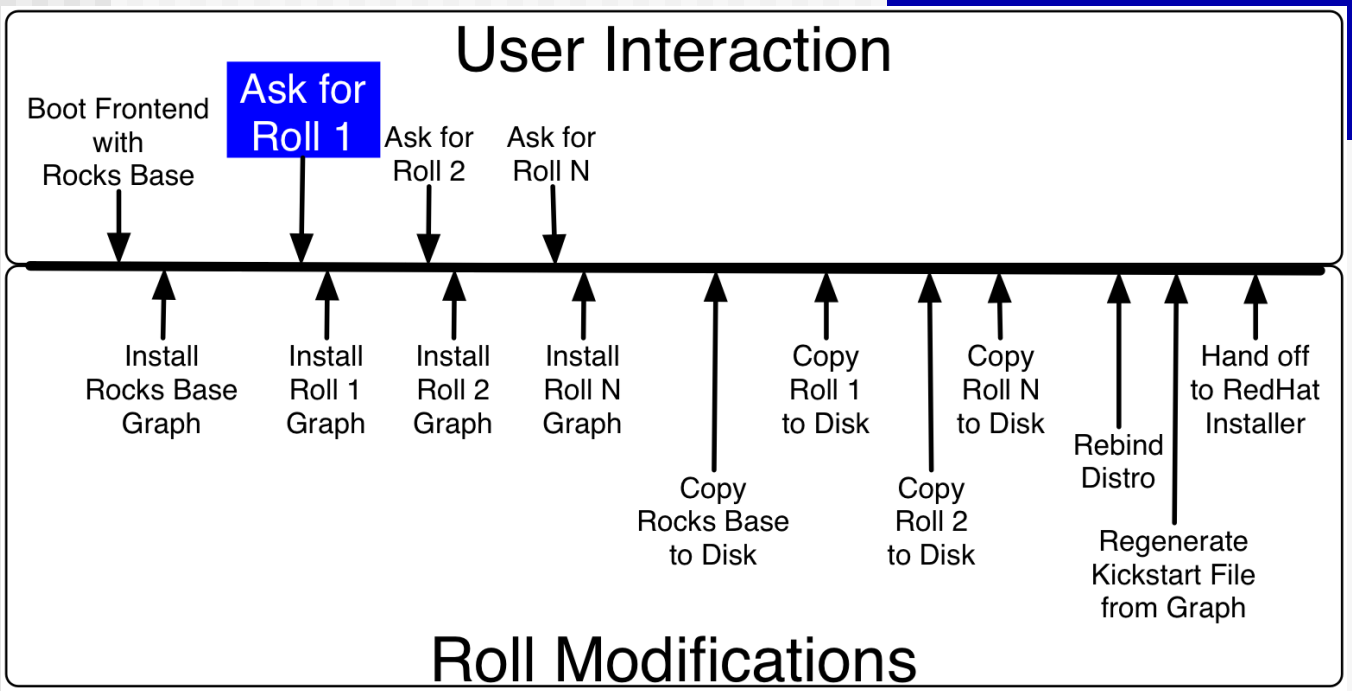


Install Rocks Base Graph

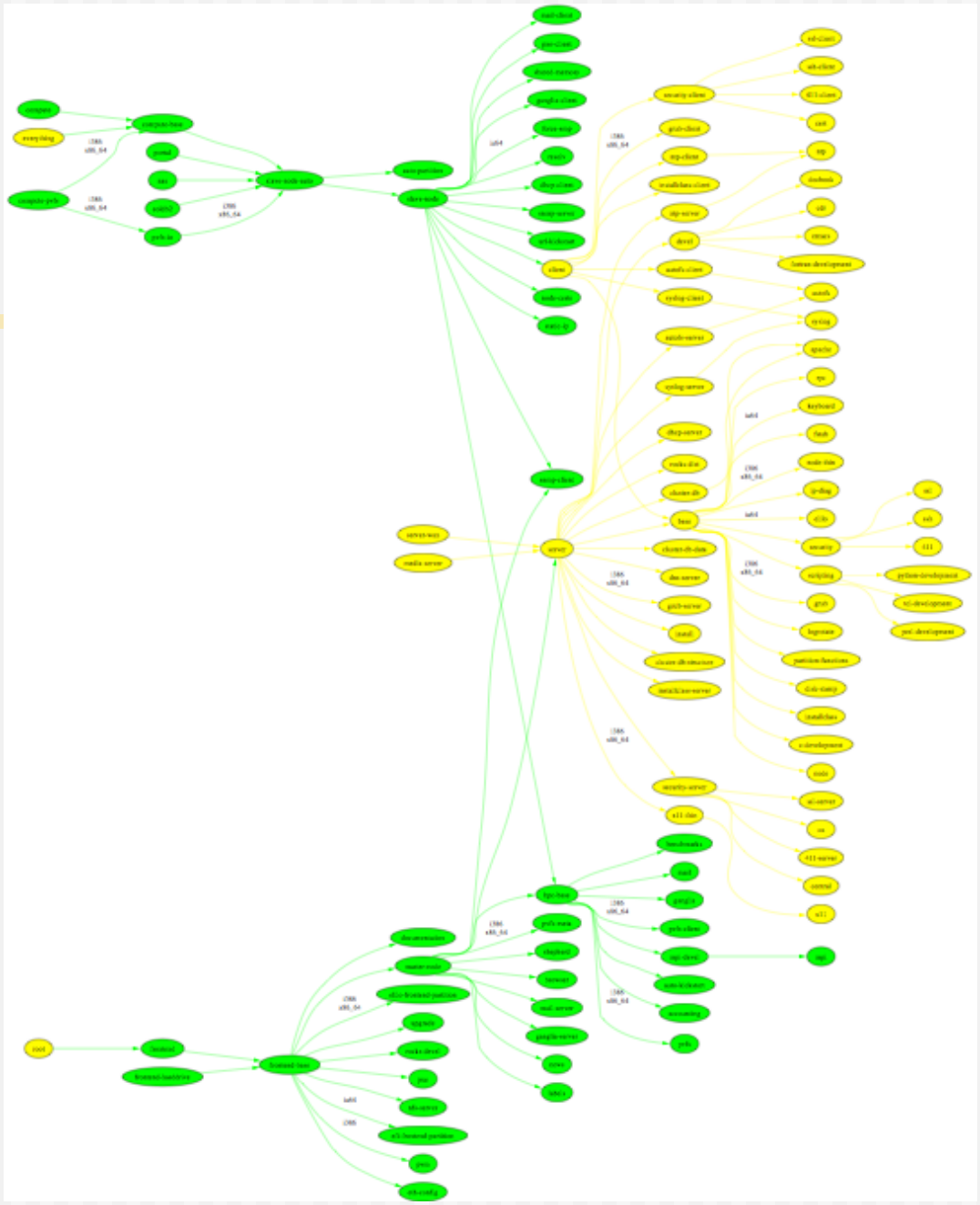


Anaconda Modified to Accept Rolls

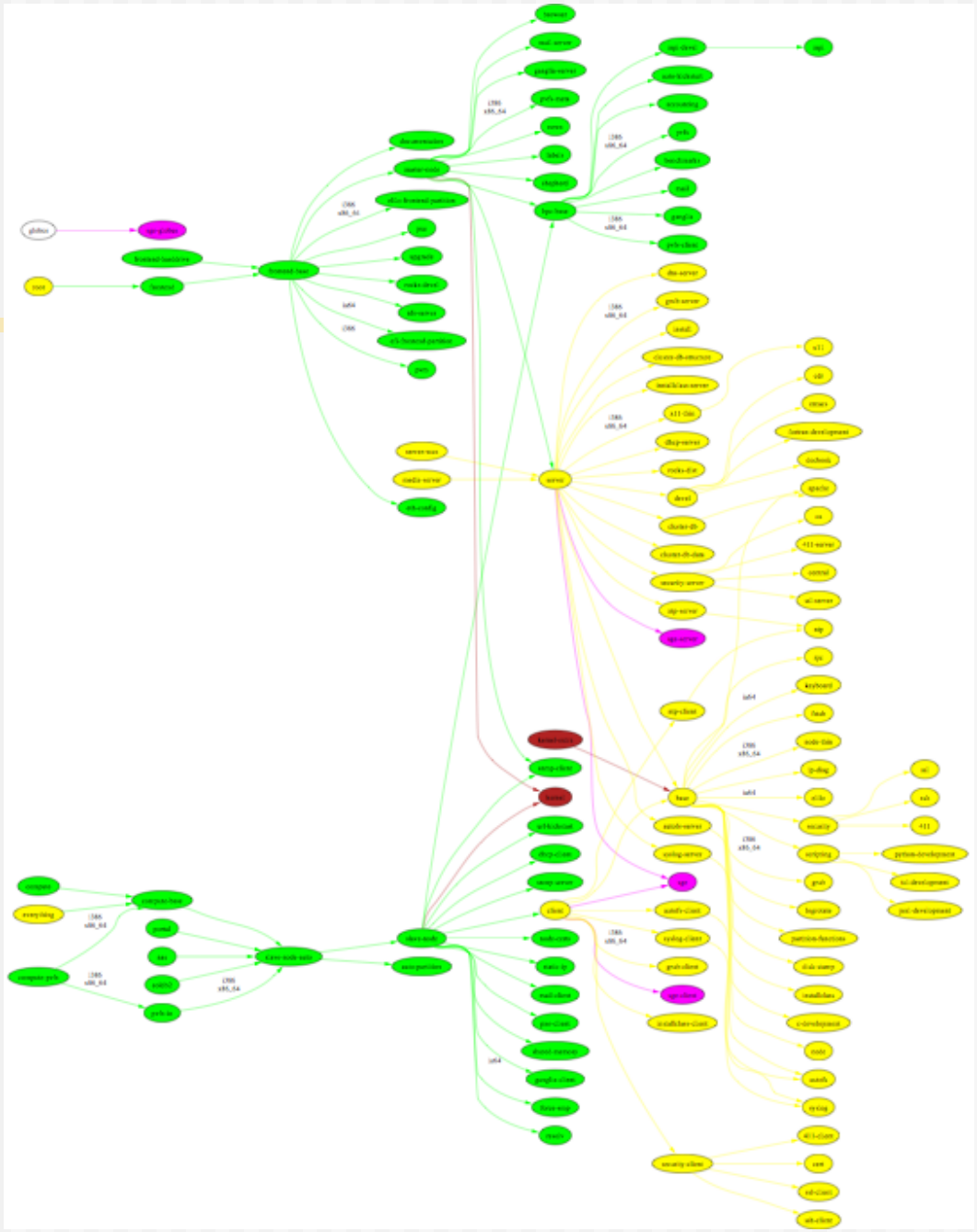




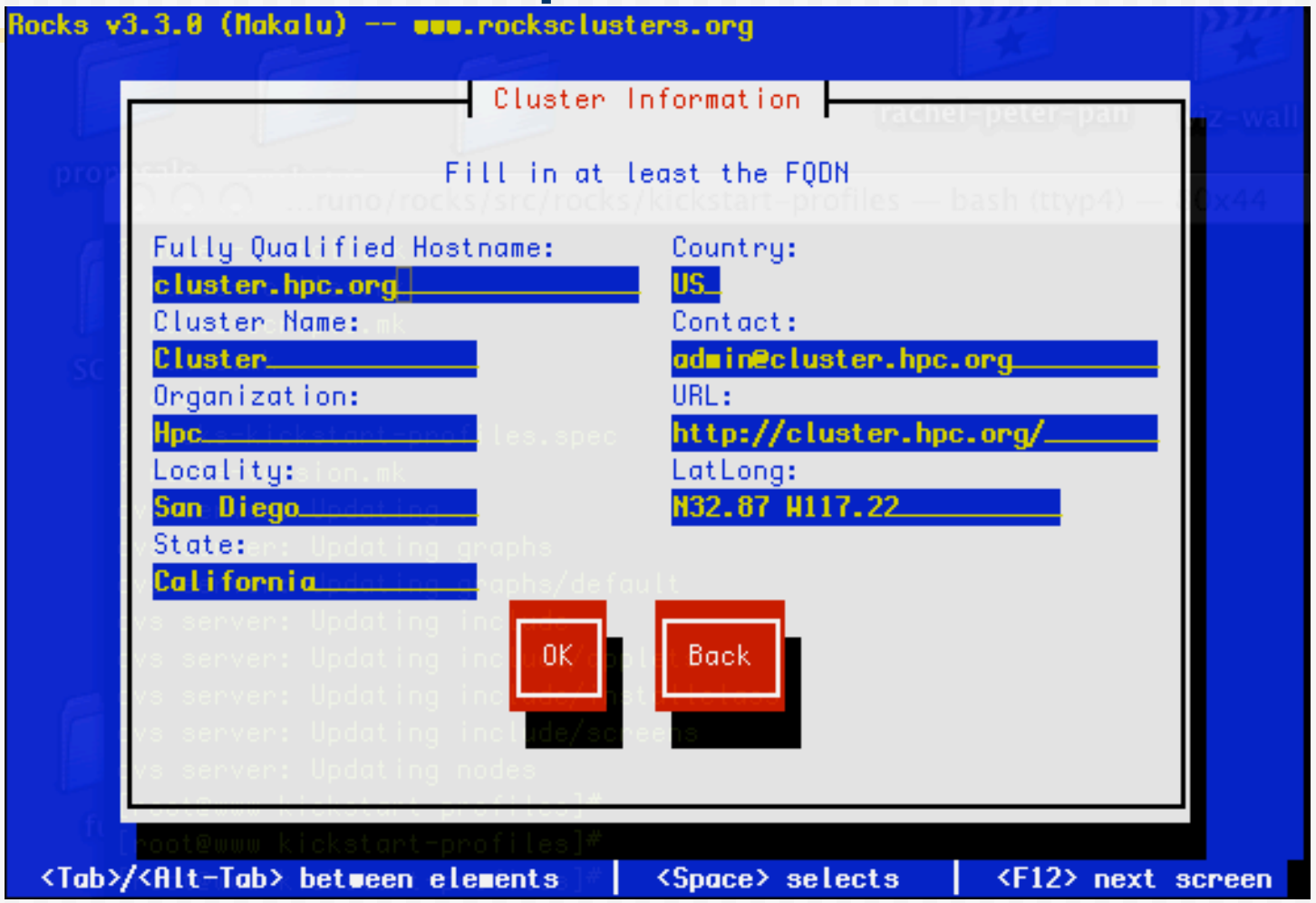
Install Roll Graph



Base + All Rolls



Anaconda Modified to Display New User Input Screens



Anaconda Modified to Display New User Input Screens

- How we do it:
  - Place a shim in Anaconda to call our screens instead of the 'betanag' RedHat screen

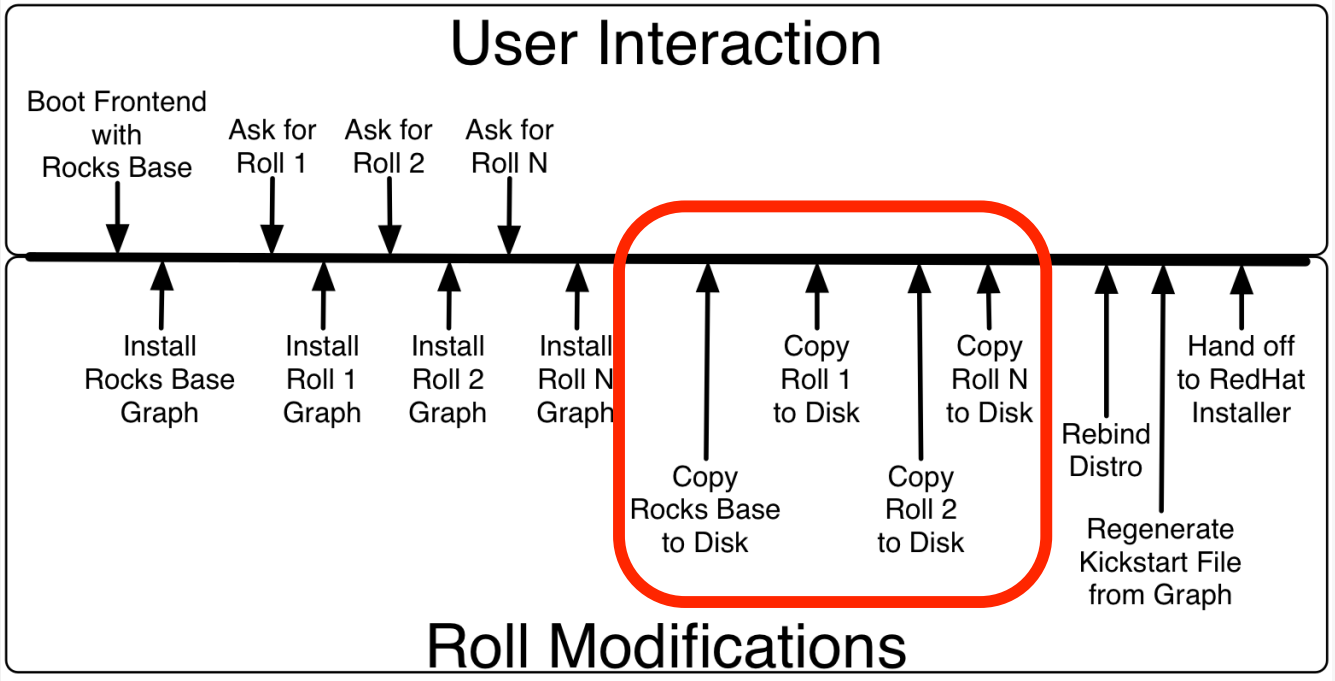
```
index = 0
for key in installSteps:
        if key[0] == "betanag":
                break
        index = index + 1

installSteps[index] = ("rockswindows", ("id.rocks", ))

# set list of user-defined windows
dispatch.skipStep("rockswindows", skip = 0)
stepToClasses["rockswindows"] = ("ksclass",
        tuple(rockswindows))
```

Anaconda Modified to Display New User Input Screens

- How you use it:
  - In your XML file:

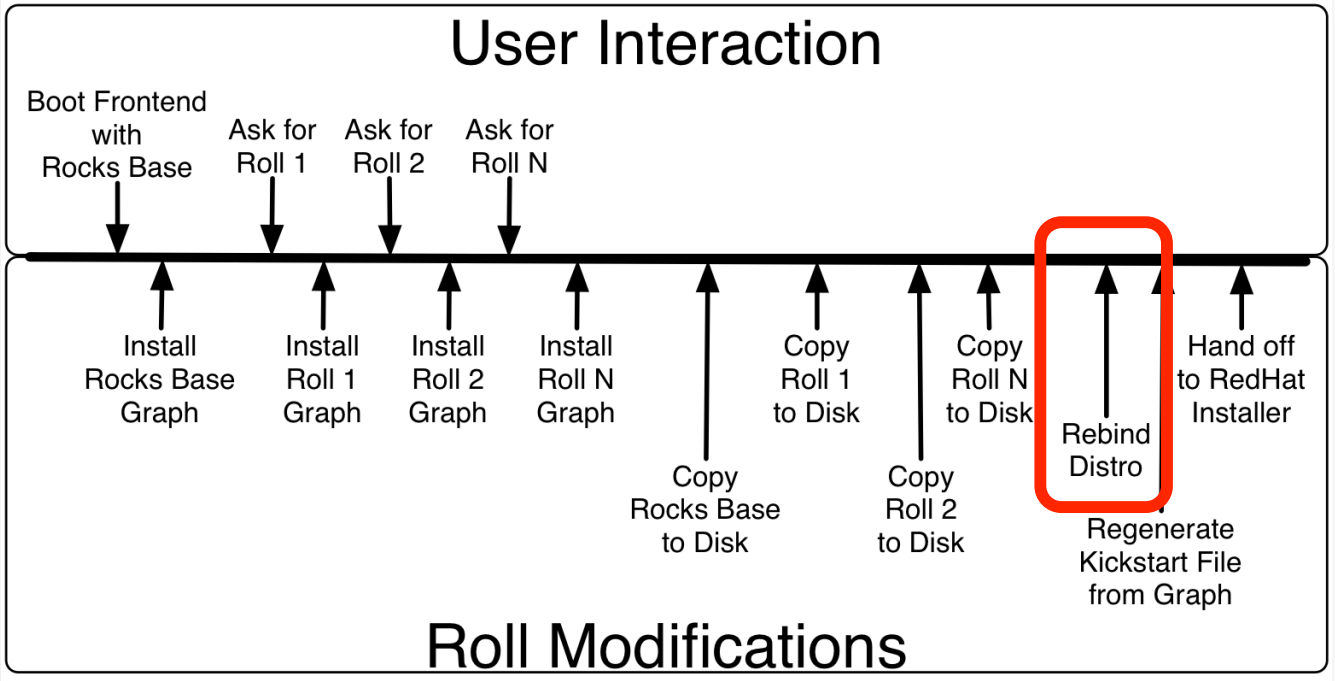
```
<installclass>

class CACLInfoWindow:

def __call__(self, screen, Info):

                bb = ButtonBar (screen, (TEXT_OK_BUTTON, ("Back","back")))
                toplevel = GridFormHelp (screen,
                        _("CACL Setup Information"), "CACLInfo", 1, 3)
                leftGrid = Grid(1,12)
                rightGrid = Grid(1,12)
                infoGrid = Grid(2,1)
                .
                .

addScreen("CACLInfoWindow")

</installclass>
```

Copy Media To Local Disk



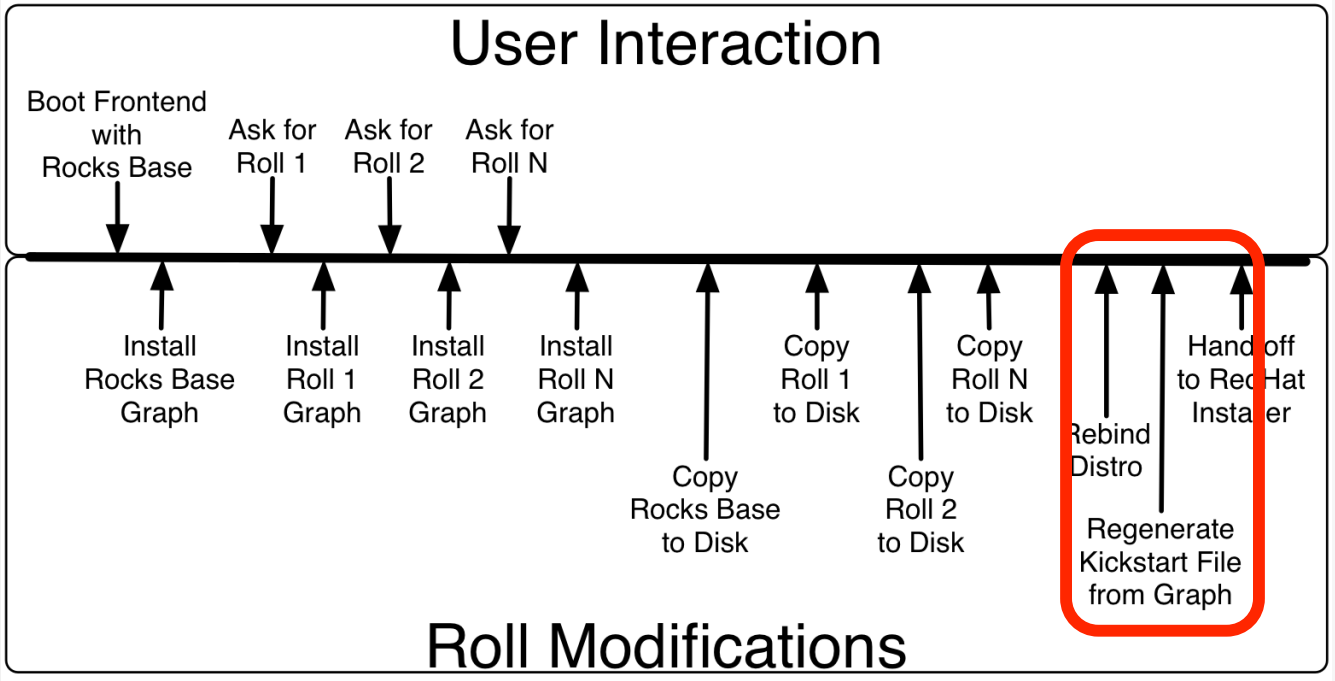
- Base and all user-supplied Rolls are copied to local disk
  - These packages are used to install compute nodes

Rebind Distro



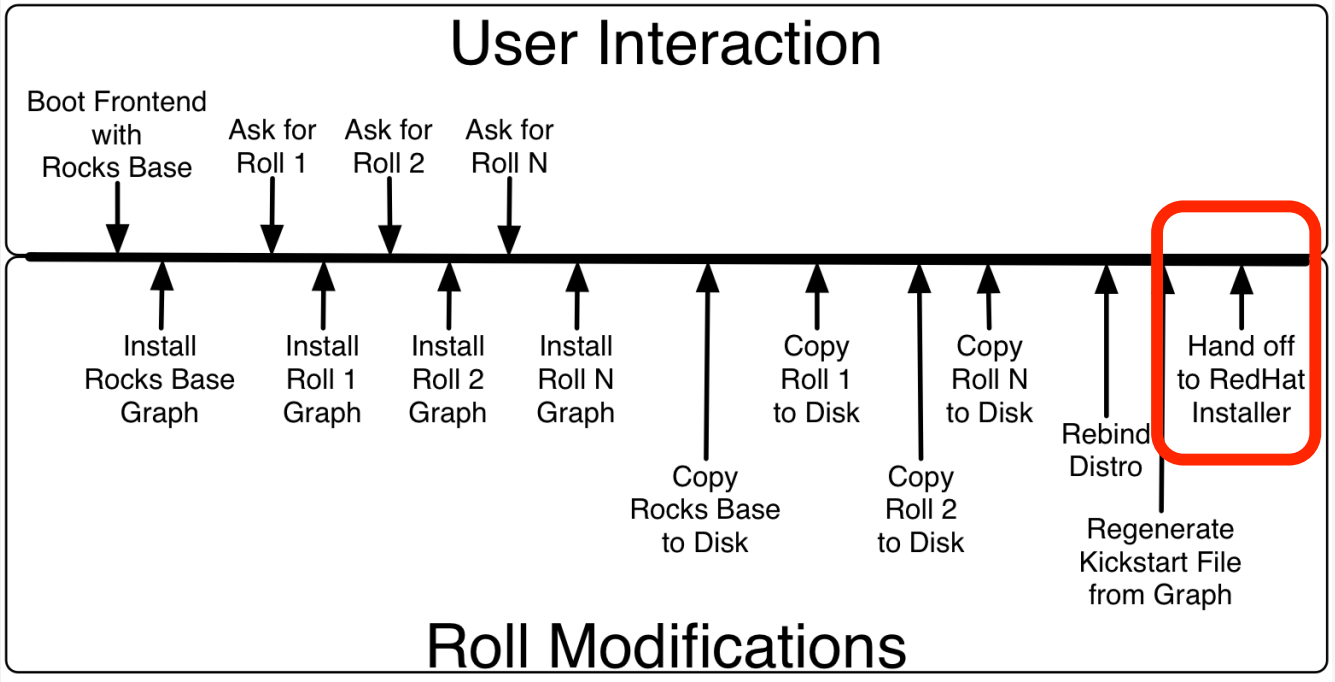
- Merge base with rolls into one RedHat-compliant distribution
  - This takes the dissected distro and tightly binds it
    - Note: We actually install the frontend off the local hard disk (not the CD media)

Rebuild the Kickstart File



- Traverse the final graph using the node ‘frontend’ as the root
  - Allows us to customize a frontend configuration at install time

Hand Off To RedHat



- Anaconda has no idea what hit it!
- The remainder of the installation looks like a standard RedHat installation (just with more packages and cluster-specific configuration)

Near Future

Rocks Futures

- Rocks 4.2
  - Graphical installer
  - ‘Restore’ Roll
    - Package files on the frontend into a roll
    - Used to configure a frontend without having to fill out the user-configuration screens
    - Also, quick way to restore to ‘known-good state’ or recover from a failed frontend
- Rocks 5.0
  - Base OS will be RHEL 5
    - Key technology in RHEL 5 is Xen

Roll Development Basics

Available Rolls for Rocks 4.1

- Rolls we provide
  - Area51
    - Security analysis tools
  - Condor
  - HPC
    - MPICH and cluster tools
  - Grid
    - Globus
  - SGE
  - Viz
  - Java

Available Rolls for Rocks 4.1

- Rolls produced by academic community
  - PBS/Maui
    - HPC group at University of Tromso, Norway
  - SCE - Scalable Computing Environment
    - University of Kasetsart, Thailand
  - APBS (Adaptive Poisson-Boltzmann Solver)
    - NBCR group, UCSD
  - MEME
    - NBCR group, UCSD
    - Tools for discovering and using protein and DNA sequence motifs
  - Ninf-G
    - AIST, Japan
    - RPC for the grid

Available Rolls for Rocks 4.1

- Rolls produced by commercial entities
  - Scalable Systems (Singapore)
    - RxC (Rocks web-based console)
    - Intel (Compilers, libraries and MPI environment)
    - Lustre Roll (available as beta release)
    - Bootstrapped the PVFS and SGE rolls
  - Quadrics
    - Interconnect Roll
  - Voltaire, SilverStorm, Topspin
    - IB Rolls
  - Myricom
    - Myrinet Roll
  - Scalable Informatics
    - ScalableInformatics Roll (cluster tools)
  - Absoft
    - Another Intel tools roll

Roll Contents

- RPMS
  - Your software.
  - Tasks:
    - Package bits into RPM
- Kickstart Graph
  - Your configuration.
  - Tasks:
    - Verify correct files exist after installation
    - Verify correct operation on frontend, computes.
    - Test, Test, Test

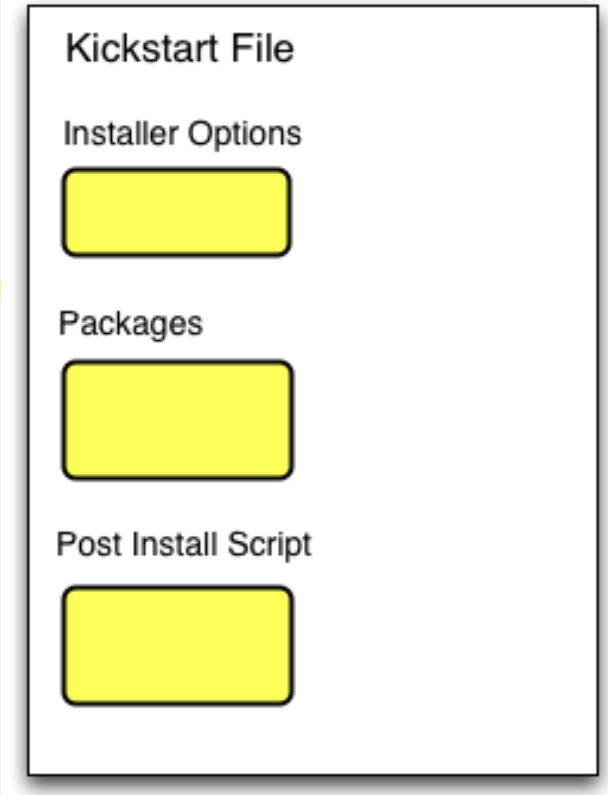
Rolls Codify Configuration for Cluster Services

- How do you configure NTP on compute nodes?
  - ntp-client.xml:

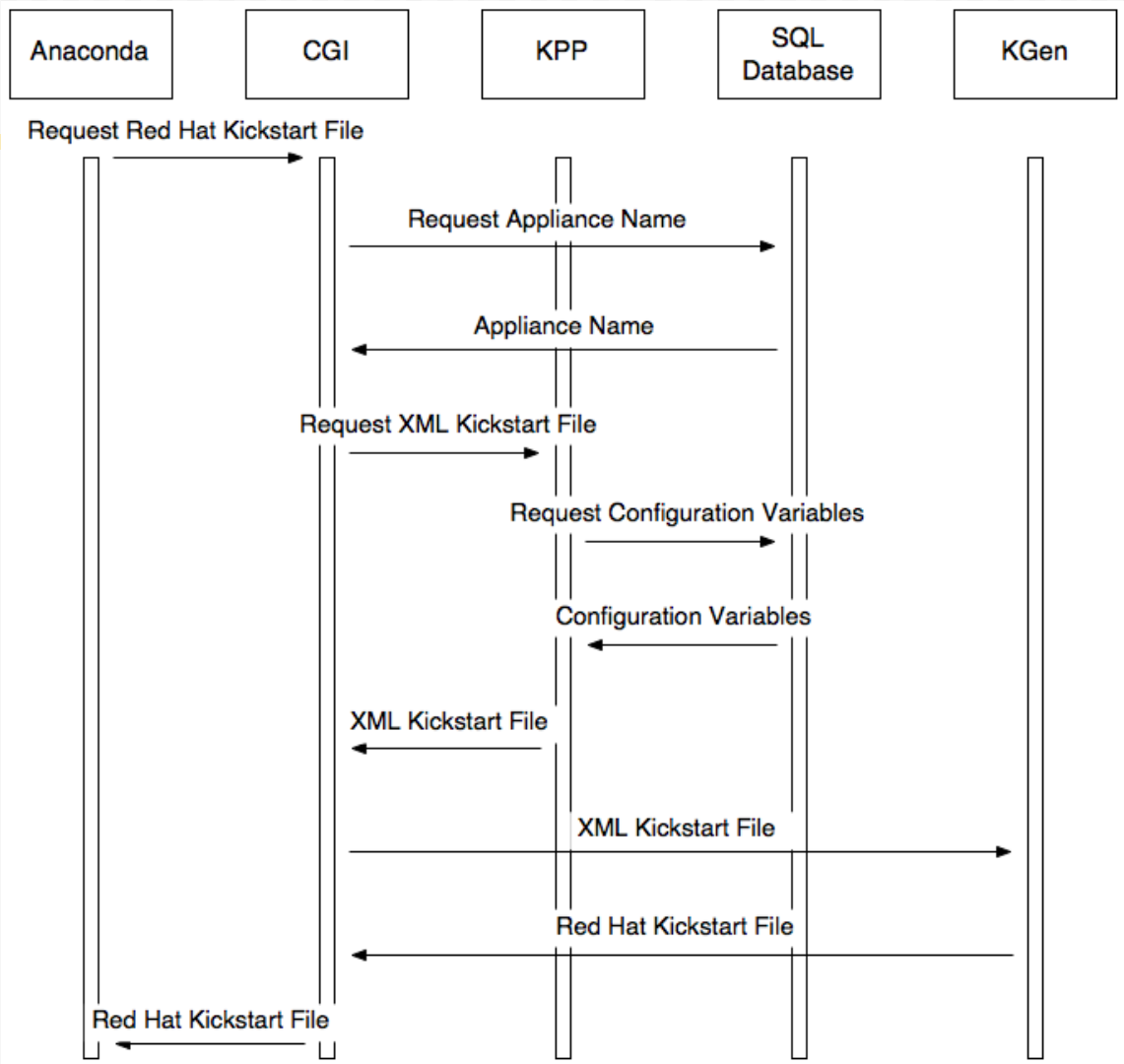
```
<post>

<!-- Configure NTP to use an external server -->

<file name="/etc/ntp.conf">
server <var name="Kickstart_PrivateNTPHost"/>
authenticate no
driftfile /var/lib/ntp/drift
</file>

<!-- Force the clock to be set to the server upon reboot -->

/bin/mkdir -p /etc/ntp

<file name="/etc/ntp/step-tickers">
<var name="Kickstart_PrivateNTPHost"/>
</file>

<!-- Force the clock to be set to the server right now -->

/usr/sbin/ntpdate <var name="Kickstart_PrivateNTPHost"/>
/sbin/hwclock --systohc
</post>
```

Kickstart File

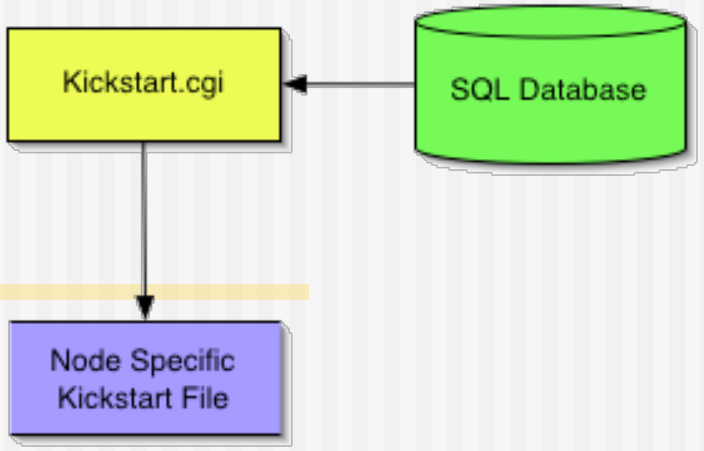
- RedHat's Kickstart: DNA of a node
  - Monolithic flat ASCII file
    - "Main": disk partitioning, timezone
    - "Packages": list of RPM names
    - "Post": shell scripts for config
  - No macro language
  - Requires forking based on site information and node type.



Getting A Kickstart File

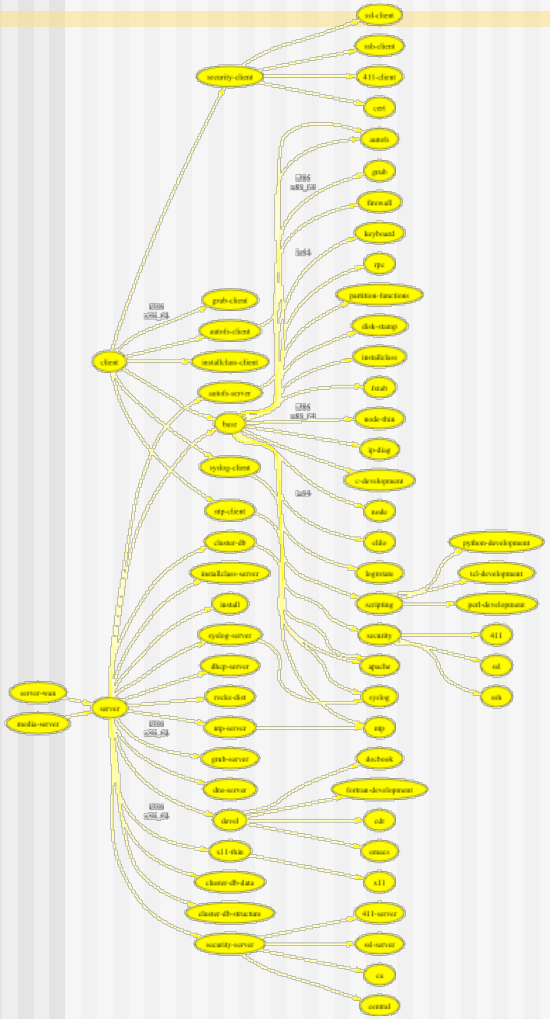


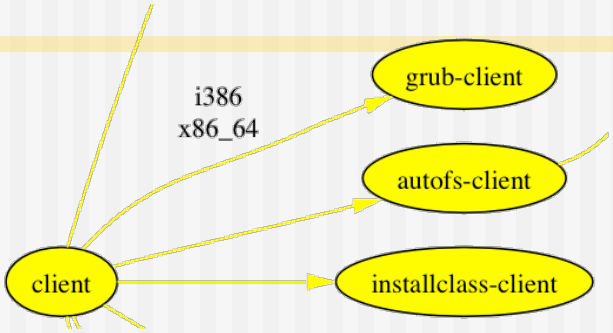
Kickstart File



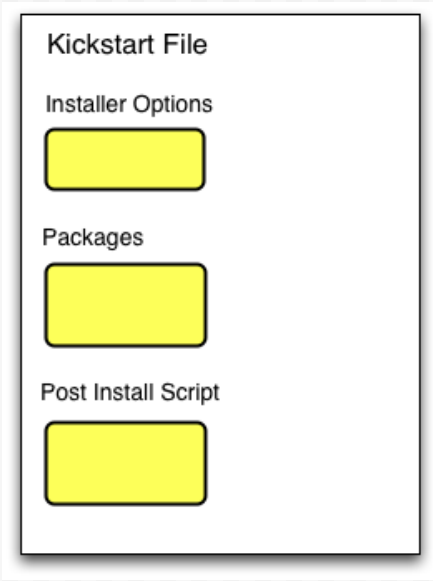
- Rocks XML Kickstart
  - Decompose a kickstart file into nodes and a graph
    - Graph specifies OO framework
    - Each node specifies a service and its configuration
  - SQL Database to help site configuration
  - “Compile” flat kickstart file from a web cgi script

Kickstart Graph for Kgen



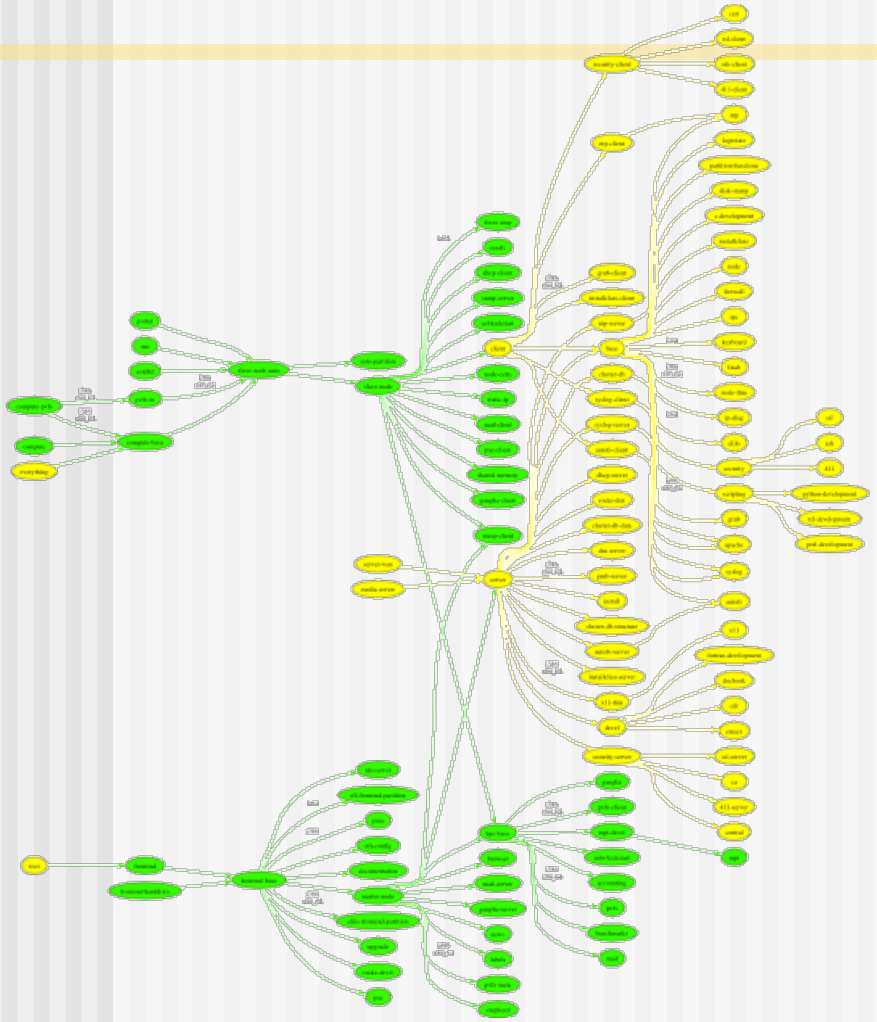


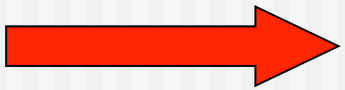
Compile (kgen)

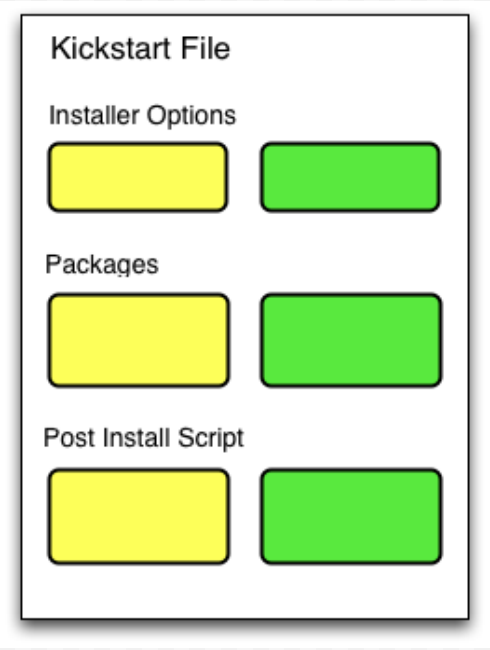


Sent to node (http)

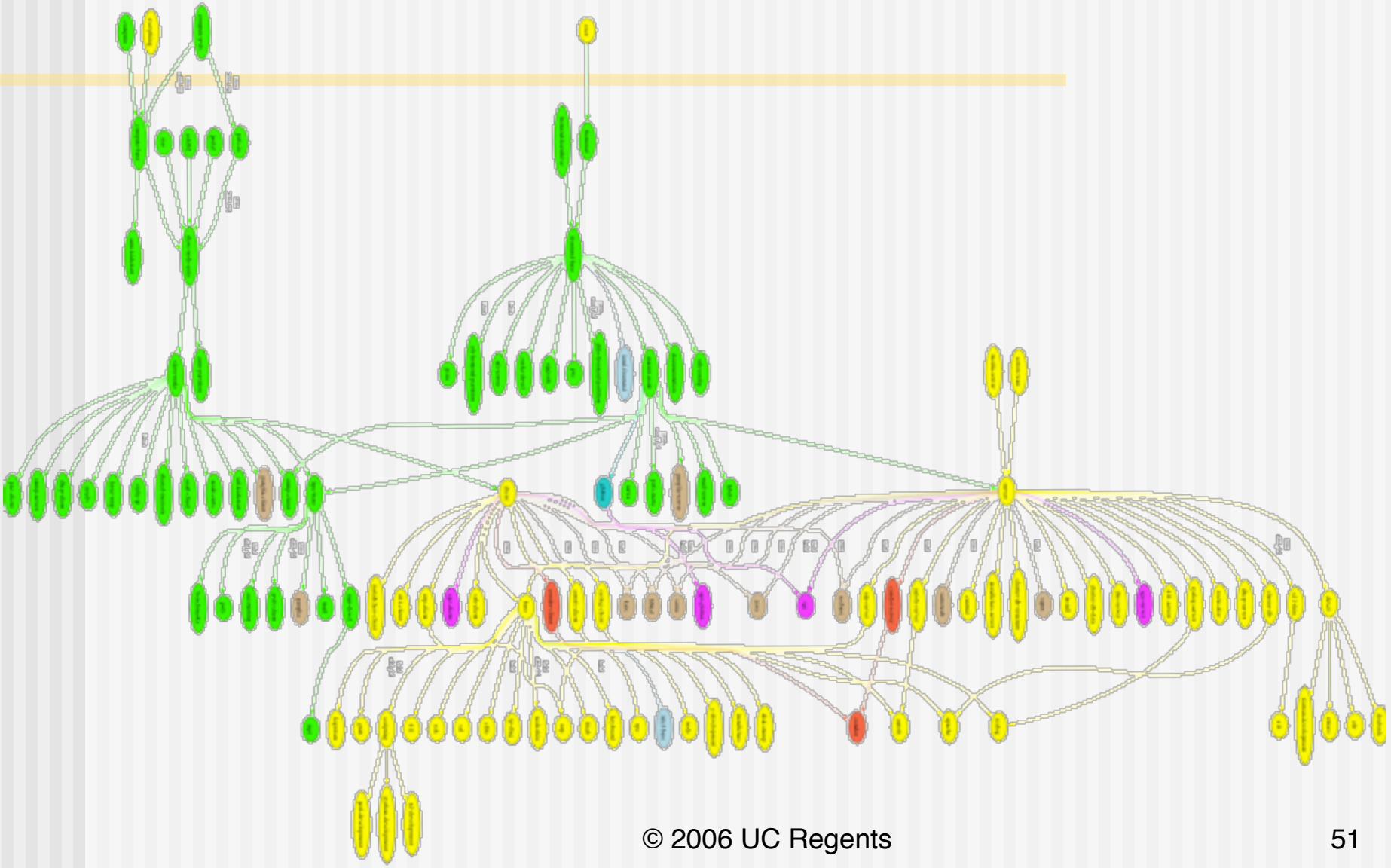
Kickstart Graph with Roll







Full Kickstart Graph



Kickstart XML Language

- Graph contains
  - Nodes
    - Rich language to help with configuration tasks
  - Edges
    - Simple. Defines node *MEMBERSHIP* in compiled kickstart files
  - Order
    - Simple syntax. Defines *POST SECTION ORDER* among nodes.

Example Roll: Sweetroll

- Will use a fictitious roll named “Sweetroll”

```
<?xml version="1.0" standalone="no"?>

<kickstart>
  <description>
The sweet roll.
  <description>
</kickstart>
```

Kickstart Nodes

- Altering Default Nodes
  - Can *replace* or *extend* default nodes in Roll
    - Extend: concatenate extend and default nodes
    - Replace: overwrite default node
  - *Discouraged use: Reserved for end users*
  - Extend by name: extend-[node].xml
    - sweetroll/nodes/extend-compute.xml
  - Replace by name: replace-[node].xml
    - sweetroll/nodes/replace-compute.xml

Kickstart Nodes

- Graph
  - Nodes
    - Rich language to help with configuration tasks
    - “Main” section
    - “Package” section
    - “Post” section
    - “Installclass” section
      - Used to modify Anaconda

Nodes XML Tools: <var>

- Get Variables from Database
  - `<var name="Kickstart_PrivateAddress"/>`
  - `<var name="Node_Hostname"/>`

```
10.1.1.1
compute-0-0
```

  - Can grab any value from the *app_globals* database table

<var> values from app_globals

←T→		ID	Membership	Service	Component	Value
Edit	Delete	6	0	Info	ClusterLatlong	N32.87 W117.22
Edit	Delete	16	0	Info	ClusterName	Onyx
Edit	Delete	30	0	Info	CertificateState	California
Edit	Delete	34	0	Info	CertificateOrganization	Rocksclusters
Edit	Delete	37	0	Info	CertificateLocality	San Diego
Edit	Delete	44	0	Info	CertificateCountry	US
Edit	Delete	45	0	Info	ClusterURL	http://onyx.rocksclusters.org/
Edit	Delete	50	0	Info	RocksRelease	Makalu
Edit	Delete	52	0	Info	RocksVersion	3.3.0
Edit	Delete	54	0	Info	ClusterContact	admin@onyx.rocksclusters.org
Edit	Delete	58	0	Info	Born	2005-02-23 14:30:13
Edit	Delete	1	0	Kickstart	PrivateKickstartBasedir	install
Edit	Delete	2	0	Kickstart	PartsizeRoot	6000
Edit	Delete	3	0	Kickstart	PublicAddress	198.202.88.74
Edit	Delete	4	0	Kickstart	PublicHostname	onyx.rocksclusters.org

- Combine “Service” and “Component”
  - For example, Kickstart_PublicAddress

Nodes XML Tools: <var>

- <var> attributes
  - name
    - Required. Format is “Service_Component”
    - Service and Component relate to column names in the app_global database table.
  - val
    - Optional. Sets the value of this variable
      - <var name=“Info_ClusterName” val=“Seinfeld”/>
  - ref
    - Optional. Set this variable equal to another
      - <var name=“Info_Weather” ref=“Info_Forecast”/>

Nodes XML Tools: <eval>

- Do processing on the frontend:
  - `<eval shell="bash">`
- To insert a fortune in the kickstart file:

```
<eval shell="bash">
/usr/games/fortune
</eval>
```

"Been through Hell?
Whaddya bring back for me?"
-- A. Brilliant

Nodes XML Tools: <eval>

- <eval> attributes
  - shell
    - Optional. The interpreter to use. Default “sh”
  - mode
    - Optional. Value is quote or xml. Default of quote specifies for kpp to escape any XML characters in output.
    - XML mode allows you to generate other tags:
      - <eval shell=“python” mode=“xml”>
        - import time
        - now = time.time()
        - print “<var name=‘Info_Now’ val=‘%s’/>” % now
      - </eval>

Nodes XML Tools: <eval>

- Inside <eval> variables are not accessed with <var>: use the environment instead.

```
<eval shell="sh">
echo "My NTP time server is
$Kickstart_PublicNTPHost"
echo "Got it?"
</eval>
```

My NTP time server is time.apple.com
Got it?

```
<eval shell="python">
import os
print "My NTP time server is",
 os.environ['Kickstart_PublicNTPHost']
print "Got it?"
</eval>
```

My NTP time server is time.apple.com
Got it?

Nodes XML Tools <include>

- Auto-quote XML characters in a file
  - **<include file="foo.py"/>**
- Quotes and includes file

  `sweetroll/include/foo.py`
- foo.py (native) → foo.py (quoted xml):

```
#!/usr/bin/python

import sys

def hi(s):
        print >> sys.stderr, s
```

```
#!/usr/bin/python

import sys

def hi(s):
        print &gt;&gt; sys.stderr, s
```

Nodes XML Tools: <include>

- <include> attributes
  - file
    - Required. The file to include (relative to “include/”) dir in roll src.
  - mode
    - Optional. Value is quote or xml. Default of quote specifies for kpp to escape any XML characters in file.
      - <include file=“my-favorite-things” mode=“quote”/>

Nodes XML Tools <file>

- Create a file on the system:
  - **<file name="/etc/hi-mom" mode="append">**
    - **How are you today?**
  - **</file>**
- Used extensively throughout Rocks post sections
  - Keeps track of alterations automatically via RCS.

```
<file name="/etc/hi" perms="444">
How are you today?
I am fine.
</file>
```

```
…RCS checkin commands...
cat > /etc/hi << 'EOF'
How are you today?
I am fine.
EOF
chmod 444 /etc/hi-mom
…RCS cleanup commands…
```

Nodes XML Tools: <file>

- <file> attributes
  - name
    - Required. The full path of the file to write.
  - mode
    - Optional. Value is “create” or “append”. Default is create.
  - owner
    - Optional. Value is “user.group”, can be numbers or names.
      - <file name=“/etc/hi” owner=“daemon.root”>
  - perms
    - Optional. The permissions of the file. Can be any valid “chmod” string.
      - <file name=“/etc/hi” perms=“a+x”>

Nodes XML Tools: <file>

- <file> attributes (continued)
  - vars
    - Optional. Value is “literal” or “expanded”. In literal (default), no variables or backticks in file contents are processed. In expanded, they work normally.
      - <file name=“/etc/hi” vars=“expanded”>
        - The current date is `date`
      - </file>
  - expr
    - Optional. Specifies a command (run on the frontend) whose output is placed in the file.
      - <file name=“/etc/hi” expr=“/opt/rocks/dbreport hi”/>

Fancy <file>: nested tags

```
<file name="/etc/hi">

Here is your fortune for today:
<eval>
date +"%d-%b-%Y"
echo ""
/usr/games/fortune
</eval>

</file>
```

…RCS checkin commands...
cat > /etc/hi << 'EOF'

Here is your fortune for today:
13-May-2005

"Been through Hell? Whaddya bring back for me?"
-- A. Brilliant

EOF
…RCS cleanup commands…

Nodes Main

- Used to specify basic configuration:
  - timezone
  - mouse, keyboard types
  - install language
- Used more rarely than other tags
- Rocks main tags are usually a straight translation:

```
<main>

 <timezone>America/Mission_Beach
 </timezone>

</main>
```

```
…
timezone America/Mission_Beach
…
rootpw --iscrypted sndk48shdlwis
mouse genericps/2
url --url http://10.1.1.1/install/rocks-dist/..
```

Nodes Main: Partitioning

- <main>
  - <part> / --size 8000 --ondisk hda </part>
  - <part> swap --size 1000 --ondisk hda </part>
  - <part> /mydata --size 1 --grow --ondisk hda </part>
- </main>

part / --size 8000 --ondisk hda
part swap --size 1000 --ondisk hda
part /mydata --size 1 --grow --ondisk hda

Nodes Packages

- **`<package>java</package>`**
  - Specifies an RPM package. Version is automatically determined: take the *newest* rpm on the system with the name ‘java’.
- **`<package arch="x86_64">java</package>`**
  - Only install this package on x86_64 architectures
- **`<package arch="i386,x86_64">java</package>`**

```
<package>newcastle</package>
<package>stone-pale</package>
<package>guinness</package>
```

```
%packages
newcastle
stone-pale
guinness
```

Nodes Packages

- RPMS are installed brute-force: no dependancy checking, always --force

Nodes Packages

- RPM name is a basename (not fullname of RPM)
  - For example, RPM name of package below is ‘kernel’

```
# rpm -qip /home/install/rocks-dist/lan/i386/RedHat/RPMS/kernel-2.6.9-22.EL.i686.rpm
Name        : kernel                       Relocations: (not relocatable)
Version     : 2.6.9                           Vendor: CentOS
Release     : 22.EL                        Build Date: Sun 09 Oct 2005 03:01:51 AM WET
Install Date: (not installed)              Build Host: louisa.home.local
Group       : System Environment/Kernel    Source RPM: kernel-2.6.9-22.EL.src.rpm
Size        : 25589794                        License: GPLv2
Signature   : DSA/SHA1, Sun 09 Oct 2005 10:44:40 AM WET, Key ID a53d0bab443e1821
Packager    : Johnny Hughes <johnny@centos.org>
Summary     : the linux kernel (the core of the linux operating system)
Description :
The kernel package contains the Linux kernel (vmlinuz), the core of any
Linux operating system
```

Nodes Post

- `<post>` for *Post-Install* configuration scripts
- Configuration scripts in <post> section run after *all* RPMs have been installed.
  - Useful: you have all your software available
  - Scripts run in “target” environment: /etc in <post> will be /etc on the final installed system
- Scripts are always non-interactive
  - No Human is driving

Nodes Post

ntp-client.xml

```
<post>

/bin/mkdir -p /etc/ntp
/usr/sbin/ntpdate <var name="Kickstart_PrivateNTPHost"/>
/sbin/hwclock --systohc

</post>
```

```
%post

/bin/mkdir -p /etc/ntp
/usr/sbin/ntpdate 10.1.1.1
/sbin/hwclock --systohc
```

Nodes Post Section

- Scripts have minimal $PATH (/bin, /usr/bin)
- Error reporting is minimal
  - Write to personal log file if you need debugging
- Not all services are up. Network is however.
  - Order tag is useful to place yourself favorably relative to other services
- Can have multiple <post> sections in a single node

Nodes XML Tools: <post>

- <post> attributes
  - arch
    - Optional. Specifies which architectures to apply package.
  - arg
    - Optional. Anaconda arguments to *%post*
      - --nochroot (rare): operate script in install environment, not target disk.
      - --interpreter: specifies script language
      - <post arg="--nochroot --interpreter /usr/bin/python">

Post Example: PXE config

```
<post arch="x86_64,i386">
mkdir -p /tftpboot/pxelinux/pxelinux.cfg

<file name="/tftpboot/pxe../default">
default ks
prompt 0
label ks
        kernel vmlinuz
        append ks inird=initrd.img……
</file>
</post>

<post arch="ia64">

<!-- Itaniums do PXE differently -->
…

</post>
```

for an x86_64 machine:

```
cat >> /root/install.log << 'EOF'
./nodes/pxe.xml: begin post section
EOF
mkdir -p /tftpboot/pxelinux/pxelinux.cfg

…RCS…
cat > /tftpboot/pxe../default << EOF
default ks
prompt 0
…
EOF
..RCS…
```

A Real Node file: ssh

```
<kickstart>
        <description>
        Enable SSH
        </description>

        <package>openssh/package>
        <package>openssh-clients</package>
        <package>openssh-server</package>
        <package>openssh-askpass</package>
<post>

<file name="/etc/ssh/ssh_config">
Host *
        CheckHostIP             no
        ForwardX11              yes
        ForwardAgent            yes
        StrictHostKeyChecking   no
        UsePrivilegedPort       no
        FallBackToRsh           no
        Protocol                1,2
</file>

chmod o+rx /root
mkdir /root/.ssh
chmod o+rx /root/.ssh

</post>
</kickstart>
```

Graph Edges

- <edge>
- Specifies *membership* in a kickstart file
  - To make a kickstart file for a compute node type:
    1. Take contents of “compute” xml node
    2. Follow all outgoing edges from “compute”
    3. Take all contents of child node
    4. Follow all its outgoing edges, etc, etc, etc
  - Edges between nodes listed in a “graph” file
    - sweetroll/graphs/default/sweetroll.xml
  - All graph files concatenated together

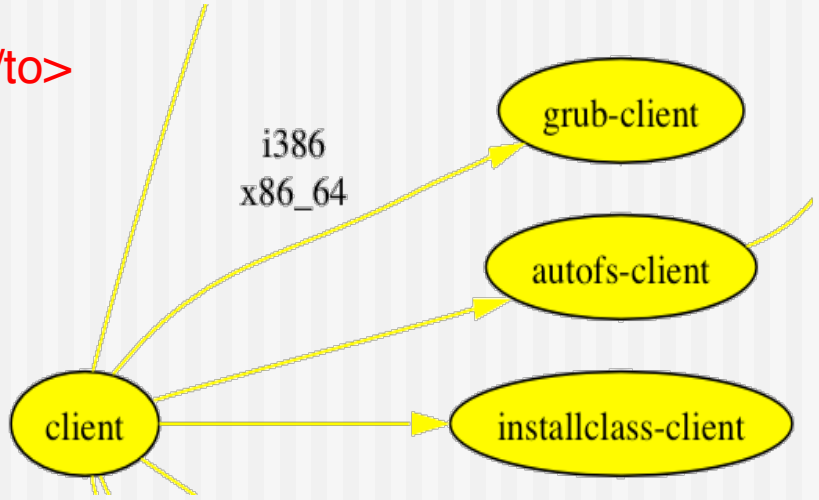
Graph Edges: <edge>

- <edge> attributes
  - from
    - Required. The name of a node at end of the edge
      - <edge from="base" to="autofs"/>
  - to
    - Required. The name of a node at the head of an edge
  - arch
    - Optional. Which architecture should follow this edge. Default is all.
  - gen
    - Optional. Which generator should follow this edge. Default is "kgen"

Graph Edges

<edge from="security-server" to="central"/>

<edge from="client">
    <to arch="i386,x86_64">grub-client</to>
    <to>autofs-client</to>
    <to>installclass-client</to>
</edge>



Graph Ordering

- Added recently to give us control over when node <post> sections are run
  - <order head="database">
    - <tail>database-schema</tail>
  - </order>
- *database* node appears before *database-schema* in all kickstart files.
- Special HEAD and TAIL nodes represent “first” and “last” (post sections that you want to run first/last)
  - <order head=“installclass” tail=“HEAD”/> BEFORE HEAD
  - <order head=“TAIL” tail=“postshell”/> AFTER TAIL

Graph Ordering: <order>

- <order> attributes
  - head
    - Required. The name of a node whose <post> section will appear BEFORE in the kickstart file.
  - tail
    - Required. The name of a node whose <post> section will appear AFTER in the kickstart file.
      - <order head="grub" tail="grub-server"/>
  - arch
    - Optional. Which architecture should follow this edge. Default is all.
  - gen
    - Optional. Which generator should follow this edge. Default is "kgen"

When Things Go Wrong

- Test your Kickstart Graph
  - Check XML syntax: xmllint
  - Make a kickstart file
    - Make kickstart file as a node will see it

    # dbreport kickstart compute-0-0 > /tmp/ks.cfg

  - Low level functionality test: kpp
    - Run the kickstart compilers by hand

When Things Go Wrong

- Test your Kickstart Graph
  - Check XML syntax: xmllint
    - # cd sweetroll/nodes
    - **# xmllint --noout sweetroll.xml**

```
<?xml version="1.0" standalone="no"?>

<kickstart>
  <description>
The sweet roll. This roll is just sweet!
  <description>
</kickstart>
```

```
# xmllint --noout sweetroll.xml

sweetroll.xml:7: parser error : Opening and
ending tag mismatch: description line 6 and
kickstart
</kickstart>
                ^
```

When Things Go Wrong

- Test your Kickstart Graph
  - Make a kickstart file
  - First, install Sweetroll “on-the-fly”:
    - # make roll; mount -o loop sweetroll-*.iso /mnt/cdrom
    - # rocks-dist copyroll; umount /mnt/cdrom
    - # cd /home/install; rocks-dist dist
    - # kroll sweetroll > /tmp/install-sweetroll.sh
    - # sh /tmp/install-sweetroll.sh

When Things Go Wrong

- Test your Kickstart Graph
  - With Sweetroll XML in place:

```
# dbreport kickstart compute-0-0 > /tmp/ks.cfg
```

  - Open /tmp/ks.cfg and look for the section:

```
cat >> /root/install.log << 'EOF'
./nodes/sweetroll.xml: begin post section
```

  - (We do this 10 times a day during release phase)
  - *Exactly the same as what a compute node actually sees during installation*

When Things Go Wrong

- Test your Kickstart Graph
  - Low level functionality test: kpp
    - Run the kickstart compilers by hand
      - For more difficult to diagnose problems
  - KPP is Kickstart Pre Processor: runs <eval>, <var>
  - KGEN is generator: turns XML into kickstart
    - # cd /home/install/rocks-dist/lan/x86_64/build
    - # kpp sweetroll
    - # kpp sweetroll | kgen

RPM Building

Building an RPM

- Generic RPMs are built with ‘spec’ file and ‘rpmbuild’
- It takes time to learn how to write a spec file
- Can use Rocks development source tree to create RPMs without having to make a spec file

Building an RPM

- We'll do the full procedure in the 'Cluster Management and Maintenance Lab'
- Short story
  - Checkout rocks development source tree
  - Make a new roll from a 'template' roll
  - Download the source tarball
  - Update a description file (version.mk)
  - Execute: make rpm
    - Assumes tarball adheres to 'configure, make, make install'

Loader Modifications

Loader Modifications

- The first program that runs during a RedHat install is a C program called “loader”
- Performs low-level setup
  - Loads drivers
  - Configures network
  - Downloads anaconda
  - Gets kickstart file

Loader Modifications

- Make HTTP the default install method
  - RedHat uses NFS as default
- Rationale
  - Installation is read-only, don't need a file system
  - HTTP traffic can be easily load balanced
  - Peer-to-peer networks use HTTP

Loader Modifications

- Robust kickstart file acquistion
  - 10 retries to get kickstart file
    - RedHat has only 1
  - NACK to throttle kickstart file acquistion
    - When load on frontend is high, the compute node is told to wait before next retry
- Rationale
  - The kickstart file is everything -- without it, a node is just a $2,000 paperweight
  - NACK feature is for supporting large cluster reinstallations

Loader Modifications

- Watchdog
  - If can't get kickstart file or if there is an error during the installation, reboot
    - This will restart the installation
    - RedHat just halts
- Rationale
  - Again, the kickstart file is everything

Loader Modifications

- Network-based frontend installations
  - In Rocks lingo: a “central” install
- Rationale
  - The “CD dance” during installation is not optimal
  - Needed to support grids of clusters from a central place
  - Huge benefit for development
    - Don’t have to burn CDs just to test code changes

Loader Modifications

- Secure kickstart
  - Added HTTPS support
- Rationale
  - Needed for support of network-based frontend installations (“central” installs)
    - Don’t want the root password for the frontend sent over the network in the clear!
  - Useful for compute nodes that are installed over a public network

Loader Modifications

- Support adding compute node to any ethernet interface
  - The first interface that receives a kickstart file, is anointed ‘eth0’
- Rationale
  - Email reduction
    - We got lots of email from people who plugged their ethernet cable into the “wrong” port
  - Even the Three Stooges can plug in the cables to the compute nodes!

Loader Modifications

- Bug Fixes
  - Added support for multiple CD drives
  - A couple stack overflow problems
- Rationale
  - Without the fixes, the installer halts