Developing Rolls

Anoop Rajendra

Two Questions

- How do you reliably add and configure (complex) software in a cluster environment
- How do you capture changes to your system, and replicate it?

What are Rolls?

- Software components that make up a Rocks system
- Mechanism for delivery of packages and configuration
- Rolls are the atomic unit in Rocks
- Rolls are how you should be getting your software onto a Rocks cluster

Purpose of Rolls

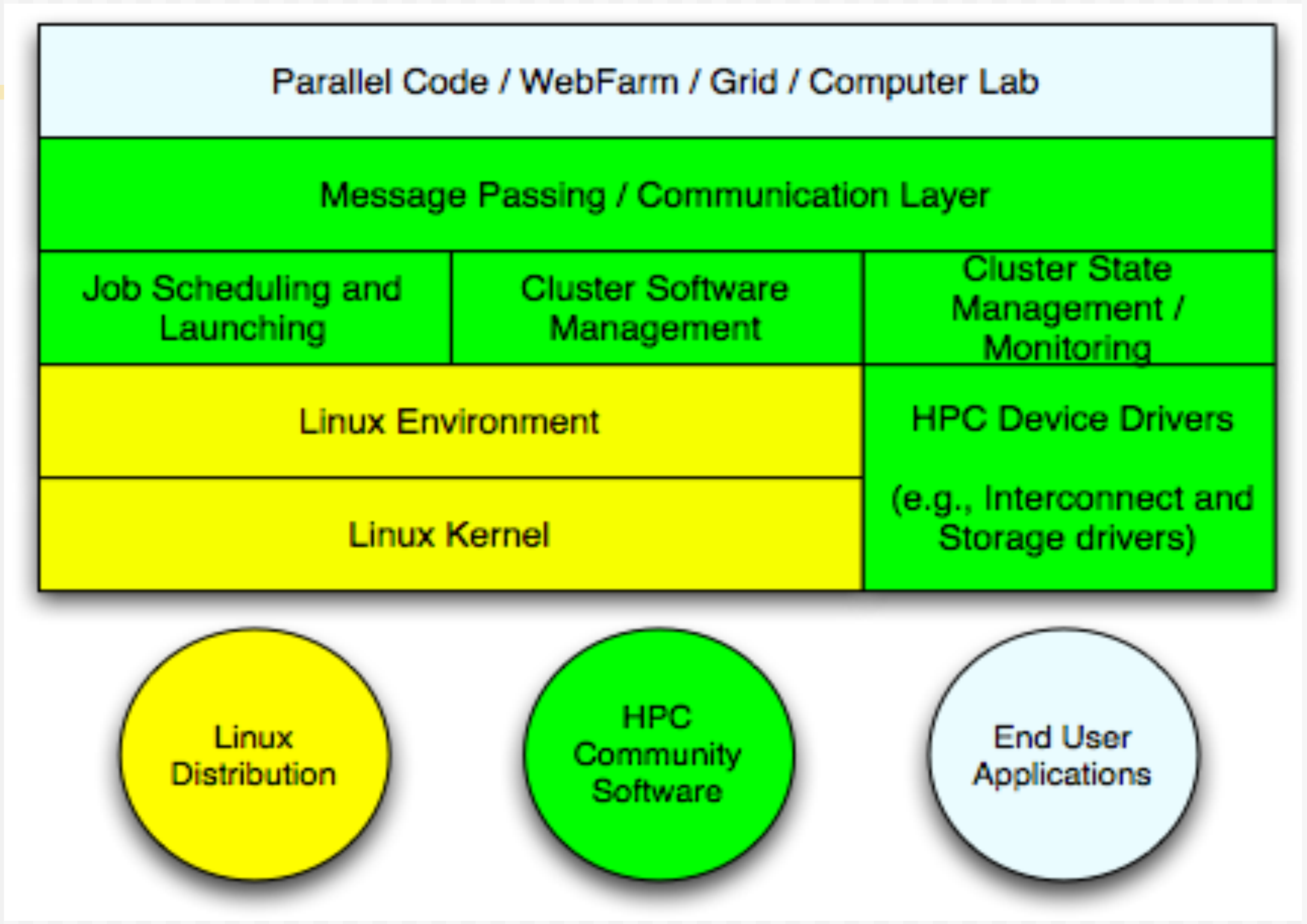
- Capture expert knowledge and automate it.
- Enable others to extend the system to provide completely new functionality
- Make the clustered system reliable and reproducible
- Backup of your software infrastructure

Rocks Philosophy

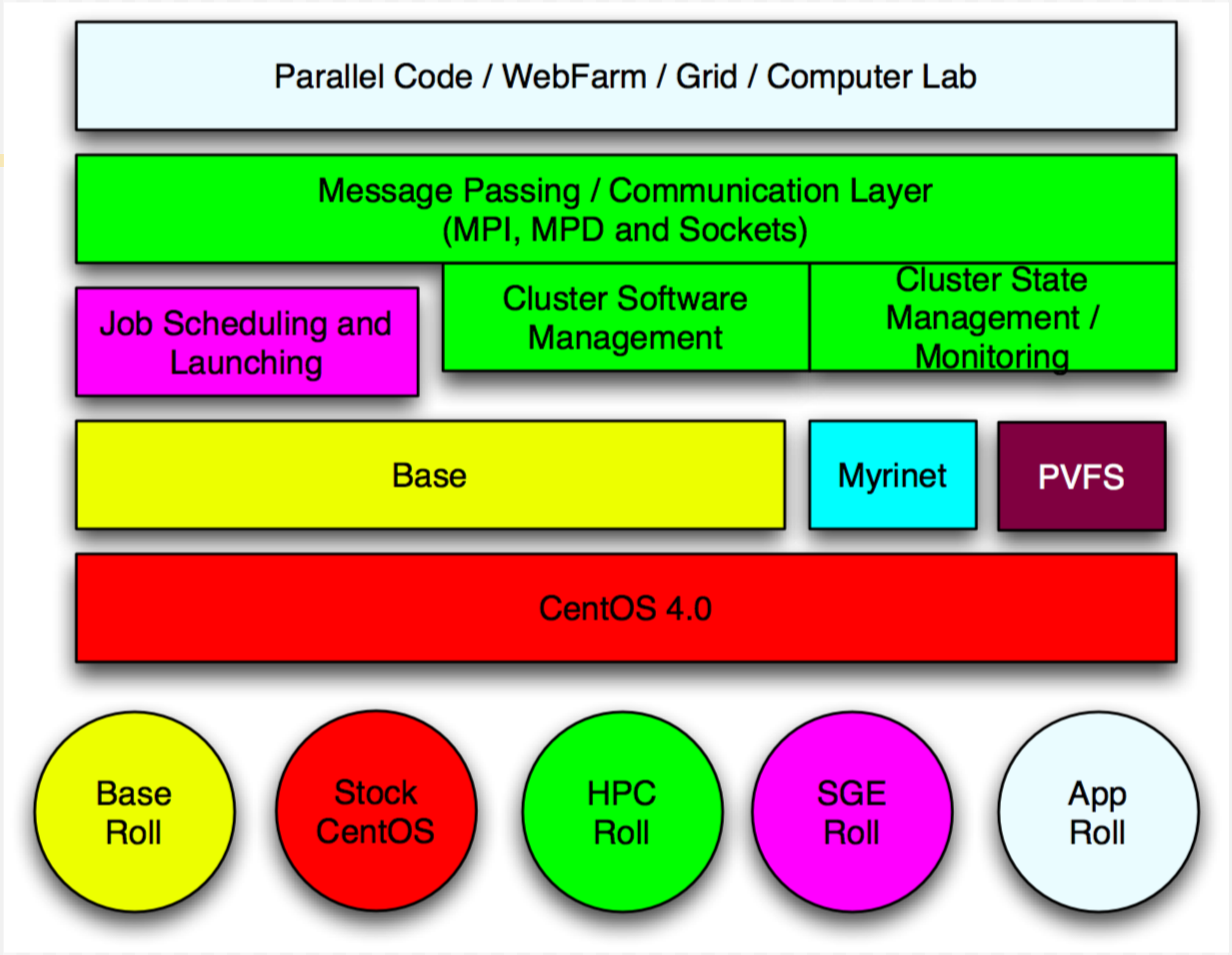
- We've developed a "cluster compiler"
  - Source code + preprocessor + linker
  - XML framework + XML parser + kickstart (Jumpstart for Solaris) file generator
- Think about "programming your cluster"
  - Not "administering your cluster"

Context of a Roll

Normal RedHat Distribution


Parallel Code / WebFarm / Grid / Computer Lab
Message Passing / Communication Layer
Job Scheduling and Launching
Cluster Software Management
Cluster State Management / Monitoring
Linux Environment
Linux Kernel
HPC Device Drivers
(e.g., Interconnect and Storage drivers)
Linux Distribution
HPC Community Software
End User Applications

Distribution Based on Rolls


Parallel Code / WebFarm / Grid / Computer Lab
Message Passing / Communication Layer (MPI, MPD and Sockets)
Job Scheduling and Launching
Cluster Software Management
Cluster State Management / Monitoring
Base
Myrinet
PVFS
CentOS 4.0
Base Roll
Stock CentOS
HPC Roll
SGE Roll
App Roll

What’s inside a Roll?

- Binaries - RPM format
- Configuration data
- Installation Map

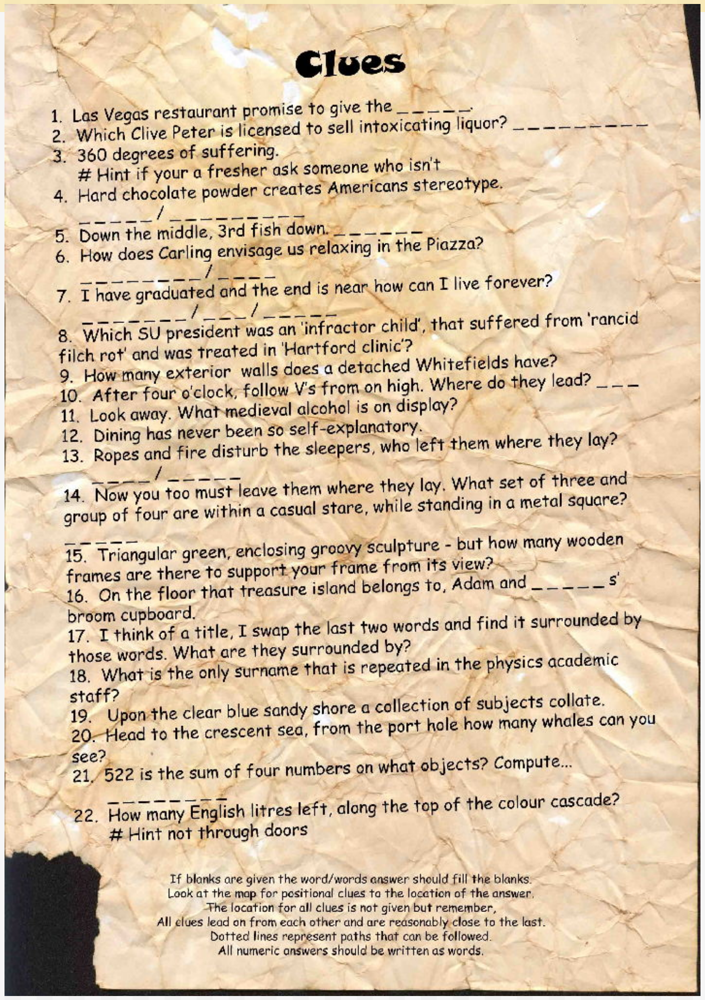
Treasure Hunting

- The treasure you seek is a fully installed and configured cluster
- What are the things you'll need

Map

Tells you the path you need to take

Instruction set



Clues

1. Las Vegas restaurant promise to give the _ _ _ _ _.
2. Which Clive Peter is licensed to sell intoxicating liquor? _ _ _ _ _ _ _ _ _ _
3. 360 degrees of suffering.
   # Hint if your a fresher ask someone who isn't
4. Hard chocolate powder creates Americans stereotype.
   _ _ _ _ _ _ / _ _ _ _ _ _ _ _ _ _
5. Down the middle, 3rd fish down. _ _ _ _ _ _
6. How does Carling envisage us relaxing in the Piazza?
   _ _ _ _ _ _ _ _ _ / _ _ _ _
7. I have graduated and the end is near how can I live forever?
   _ _ _ _ _ _ _ _ / _ _ _ / _ _ _ _ _
8. Which SU president was an 'infractor child', that suffered from 'rancid filch rot' and was treated in 'Hartford clinic'?
9. How many exterior walls does a detached Whitefields have?
10. After four o'clock, follow V's from on high. Where do they lead? _ _ _
11. Look away. What medieval alcohol is on display?
12. Dining has never been so self-explanatory.
13. Ropes and fire disturb the sleepers, who left them where they lay?
    _ _ _ _ / _ _ _ _ _
14. Now you too must leave them where they lay. What set of three and group of four are within a casual stare, while standing in a metal square?
    _ _ _ _ _
15. Triangular green, enclosing groovy sculpture - but how many wooden frames are there to support your frame from its view?
16. On the floor that treasure island belongs to, Adam and _ _ _ _ _ s' broom cupboard.
17. I think of a title, I swap the last two words and find it surrounded by those words. What are they surrounded by?
18. What is the only surname that is repeated in the physics academic staff?
19. Upon the clear blue sandy shore a collection of subjects collate.
20. Head to the crescent sea, from the port hole how many whales can you see?
21. 522 is the sum of four numbers on what objects? Compute...
    _ _ _ _ _ _ _ _ _
22. How many English litres left, along the top of the colour cascade?
    # Hint not through doors

If blanks are given the word/words answer should fill the blanks.
Look at the map for positional clues to the location of the answer.
The location for all clues is not given but remember,
All clues lead on from each other and are reasonably close to the last.
Dotted lines represent paths that can be followed.
All numeric answers should be written as words.

What to do at each point in the map ?

Resources

- Small trinkets you pick up along the way
  - Just to keep
  - Or to use later

Translate that to a Roll

- Graph file ⇔ Installation Map ⇔ Map
- Node Files ⇔ Configuration Data ⇔ Instruction set
- RPM/Binaries ⇔ Resources

Making your own Roll

Creating a Roll Directory

- On a Rocks 5.3 system

```
# cd /export/site-roll/rocks/src/roll
# rocks create new roll valgrind version=3.5 color=orange
# find valgrind/ -type f
valgrind/src/sunos.mk
valgrind/src/linux.mk
valgrind/src/valgrind/Makefile
valgrind/src/valgrind/version.mk
valgrind/src/Makefile
valgrind/nodes/valgrind.xml
valgrind/Makefile
valgrind/graphs/default/valgrind.xml
valgrind/version.mk
```

Part I: Packages

Packages

- Rolls require packages to be in native OS format
  - RPM for Linux
  - PKG for Solaris

Packages - Advantages

- Inspect software with native OS tools
- Can install “by hand” using OS tools
- Tracking is easy – The system knows about the package

Packages - Disadvantages

- You have to make your software into a package
  - This only seems hard
- Package Mechanisms can sometimes cause odd behavior
  - Solaris PKG does not like “_”
  - RPMS can have quirks

Our Philosophy on Packages

- We use packages as a transport
- No configuration is done in the package %post section
  - This is what the Rocks node files are used for
- Stay away from explicitly creating “spec” files
- Make is your friend (ours too)

Make requirements

- For Linux, we support building only a frontend node
- For Solaris, we support Solaris Development appliance
- Faith
  - There is large set of included make rules that allow us to quickly package software
  - You have to trust what the system is doing.

Different Ways For Packaging From Source

- Build software by hand, then point

  ```
  # rocks create package
  ```

  at the directory
- Build an RPM Spec file
- Use the Rocks-supplied Make Infrastructure

Valgrind – A Working Example

- Using the valgrind example.

```
# cd  valgrind/src/valgrind
# wget -q http://valgrind.org/downloads/valgrind-3.5.0.tar.bz2
```

- Edit version.mk to read

```
# cat version.mk
PKGROOT         = /opt/valgrind
NAME            = valgrind
VERSION         = 3.5.0
RELEASE         = 1
TARBALL_POSTFIX         = tar.bz2
```

- The only thing that needed changing was the last line from “.tgz” to “tar.bz2”

Valgrind – A Working example

- Now inspect the Makefile
- Three lines make all the difference in the world

```
REDHAT.ROOT = $(CURDIR)/../../

-include $(ROCKSROOT)/etc/Rules.mk
include Rules.mk
```

- Never ever change these lines
  - (Unless you're doing something real fancy)
    - (which you shouldn't be in the first place)

Valgrind – A working example

- A really simple Makefile
- Three targets
  - build, install, clean
- "build" runs "configure; make"
- "install" runs "make install"
- "clean" can be to cleanup after yourself in the build directory
    - strictly optional but recommended

Valgrind – A working example

- One small change to Makefile
- Since tarball is “`tar.bz2`” change

“`gunzip –c`” to “`bzcat`”

- Run

```
# gmake pkg
```

- “gmake” compiles on both Linux and SunOS
- “pkg” creates RPM on Linux and PKG on SunOS

Do it!

```
[root@aurora valgrind]# ls
graphs  Makefile  nodes  src  version.mk
[root@aurora valgrind]# cd src/valgrind/
[root@aurora valgrind]# gmake pkg 1>build.log 2>&1 </dev/null &
[1] 18813
[root@aurora valgrind]# ls
_arch         Makefile   rocks-version.mk       Rules-linux.mk    Rules-
   scripts.mk       valgrind.spec.mk
build.log     _os        Rules-install.mk       Rules.mk
   valgrind-3.5.0.tar.bz2  version.mk
_distribution  python.mk  Rules-linux-centos.mk  Rules-rcfiles.mk
   valgrind.spec
[root@aurora valgrind]# cd ../..
[root@aurora valgrind]# ls
BUILD  graphs  Makefile  nodes  RPMS  SOURCES  SPECS  src  SRPMS  version.mk
```

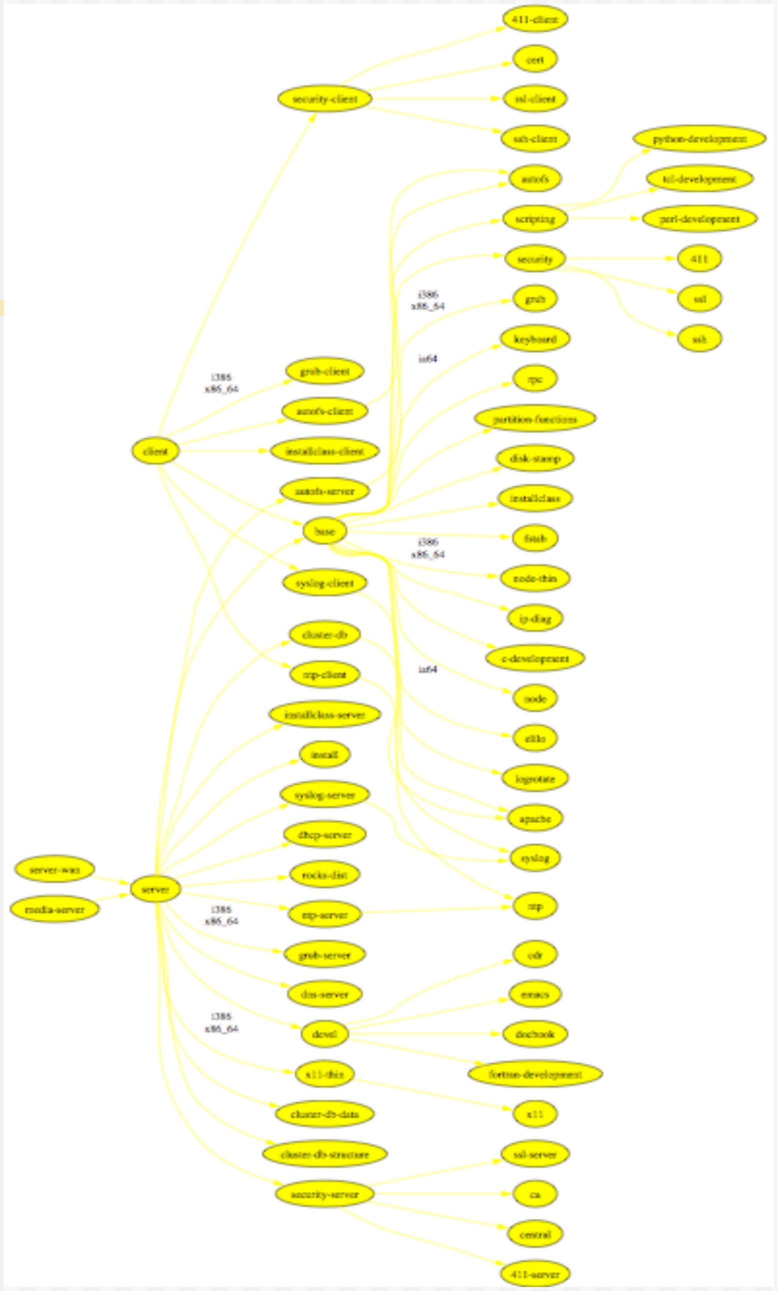
```
# find RPMS/ -type f
RPMS/x86_64/valgrind-3.5.0-1.x86_64.rpm
[root@aurora valgrind]# rpm -qip RPMS/x86_64/valgrind-3.5.0-1.x86_64.rpm
Name        : valgrind                    Relocations: (not relocatable)
Version     : 3.5.0                            Vendor: Rocks Clusters
Release     : 1                            Build Date: Thu 03 Jun 2010
   09:25:24 AM PDT
Install Date: (not installed)              Build Host:
   aurora.rocksclusters.org
Group       : System Environment/Base      Source RPM:
   valgrind-3.5.0-1.src.rpm
Size        : 57700775                        License: University of
   California
Signature   : (none)
Summary     : Tool for finding memory management bugs in programs
Description :
Valgrind is a tool to help you find memory-management problems in your
programs. When a program is run under Valgrind's supervision, all
reads and writes of memory are checked, and calls to
malloc/new/free/delete are intercepted. As a result, Valgrind can
detect a lot of problems that are otherwise very hard to
find/diagnose.
```

Part II: The Map

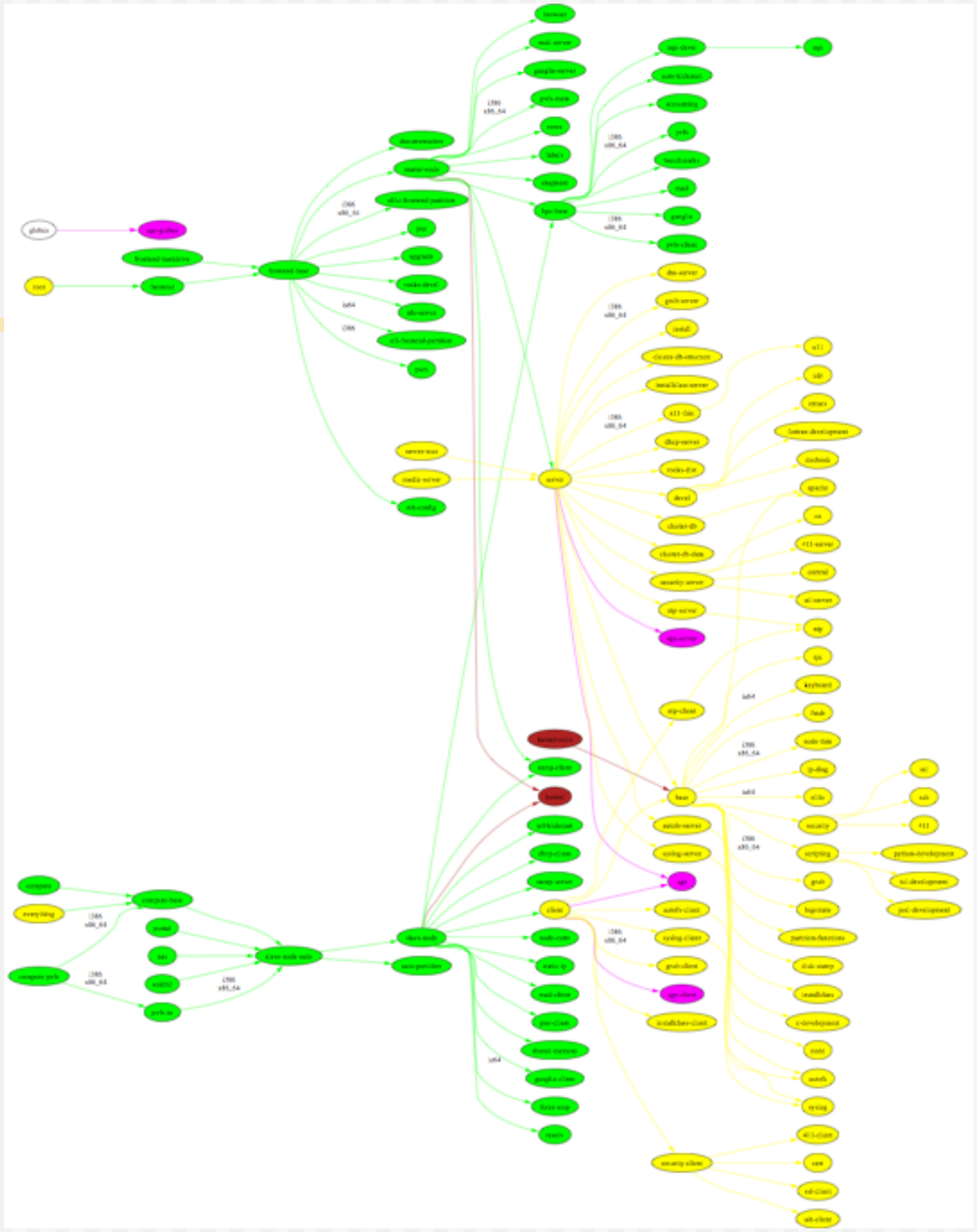
Install Rocks Base Graph

Basic Instructions that define all Rocks Appliances

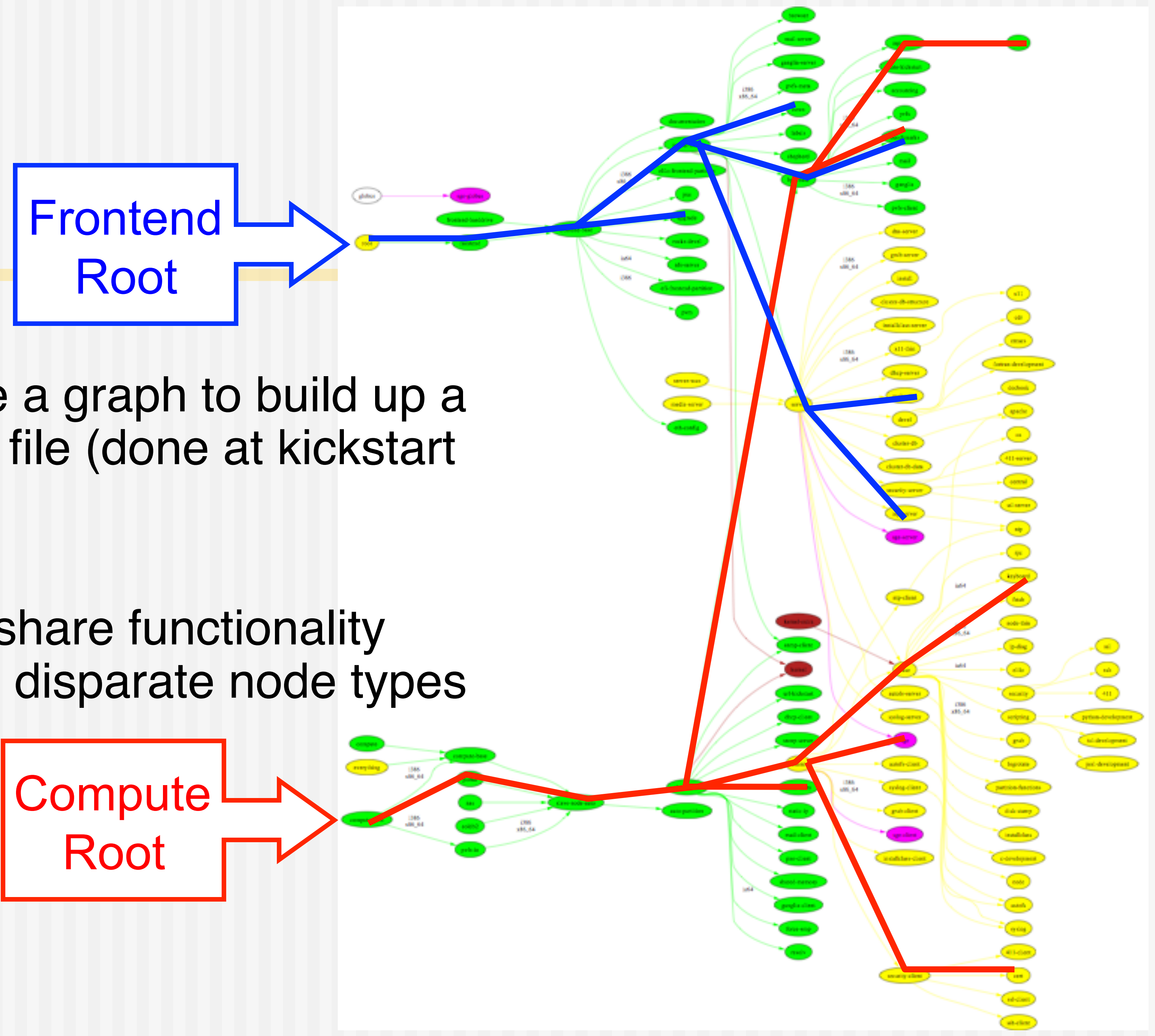
Rolls have packages and graphs



Base + Rolls

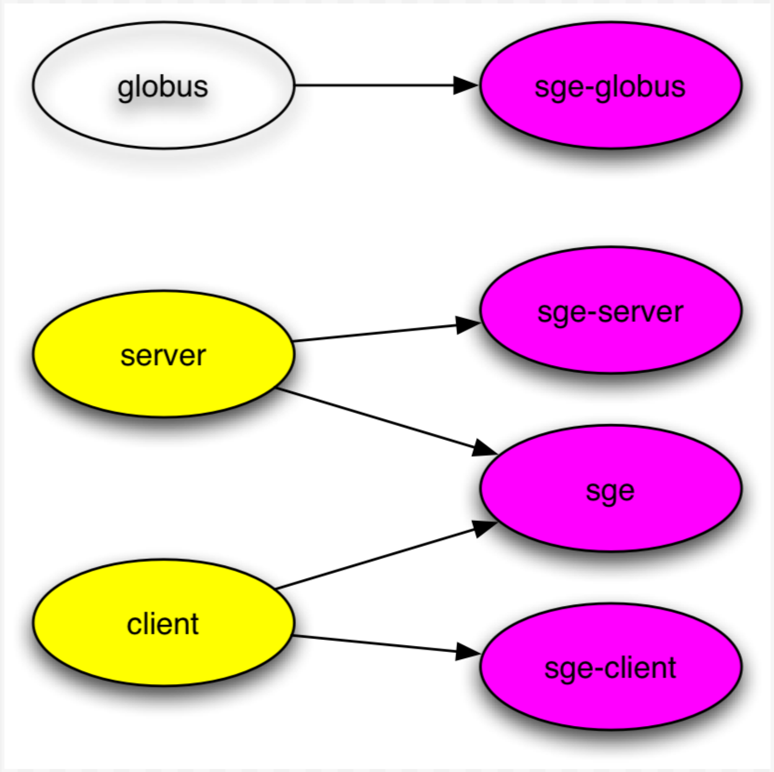


- Traverse a graph to build up a kickstart file (done at kickstart time)
- Flexible
- Easy to share functionality between disparate node types



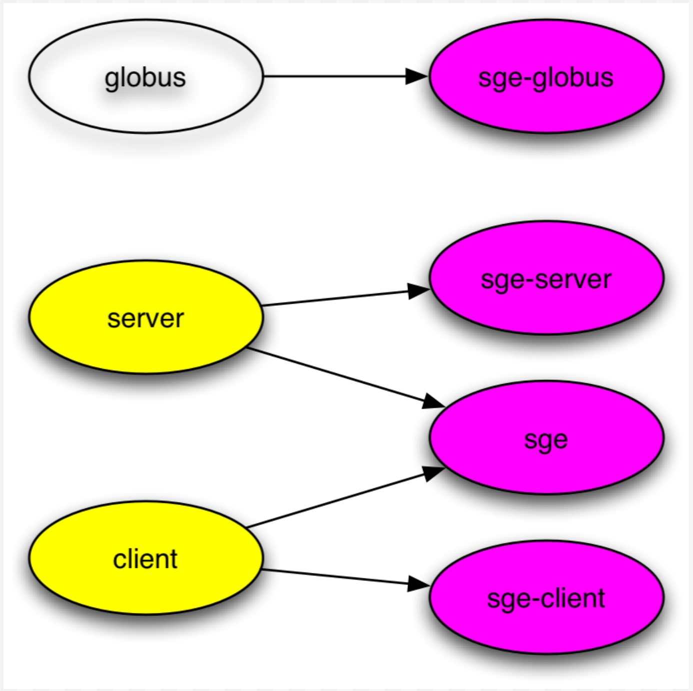
Use Graph Structure to Dissect Distribution

- 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



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



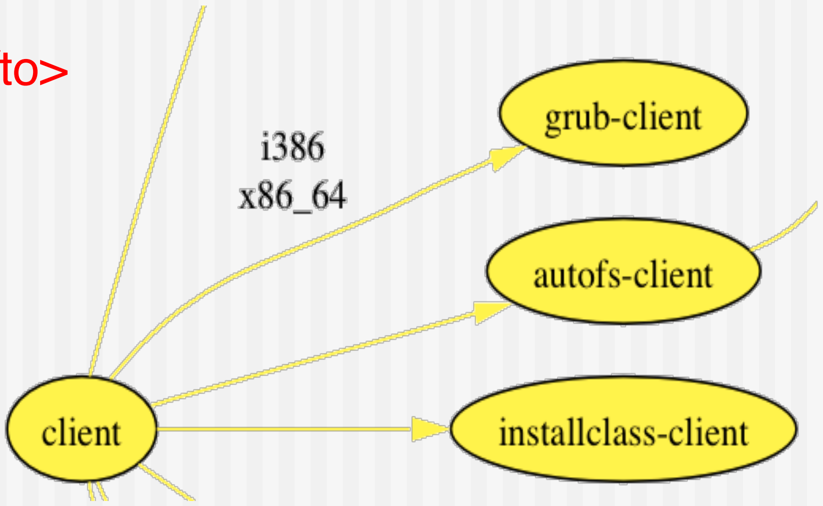
Graph Edges: <edge>

- <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.
- In 5.3 edges can have conditionals based on attributes

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>

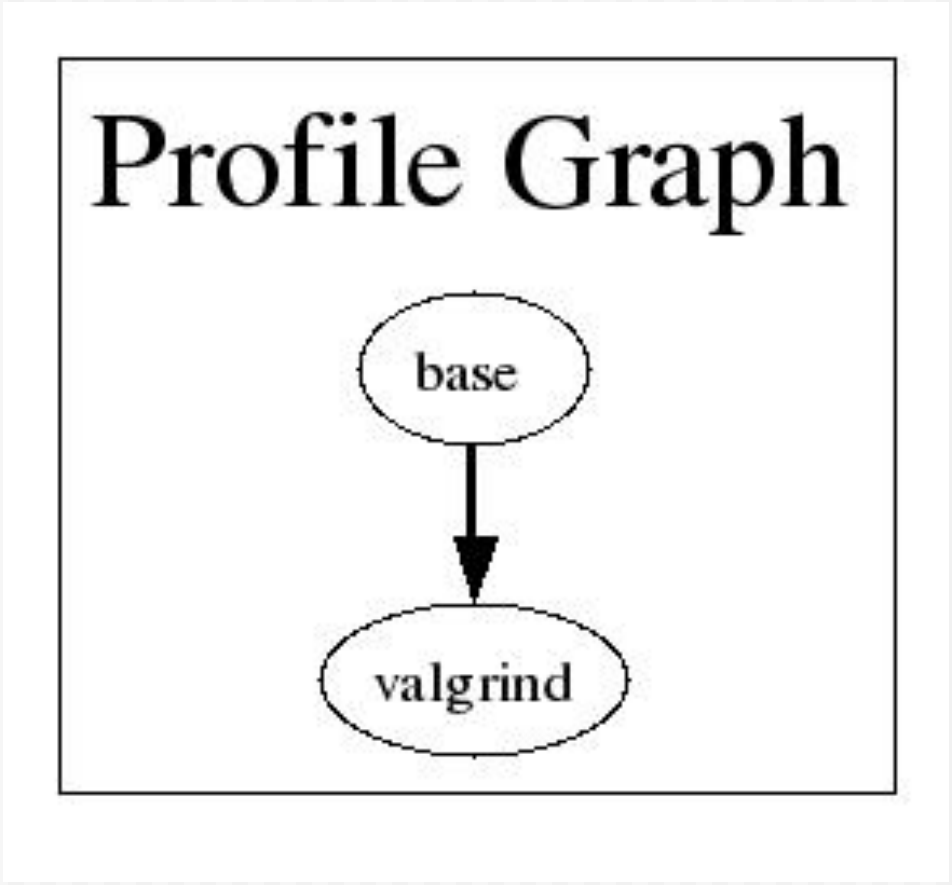
- <order> attributes
  - head
    - Required. The name of a node whose <post> section will appear BEFORE in the kickstart file.
  - tail
    - Required. The name of a node whose <post> section will appear AFTER in the kickstart file.
      - <order head="grub" tail="grub-server"/>
  - arch
    - Optional.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

Valgrind Example: Connecting into the graph

```
# vi graphs/default/valgrind.xml ( and add:)
        <edge from="base">
              <to>valgrind</to>
         </edge>
```

This tells us that Valgrind should be on all appliances.

Valgrind – A working Example



Part III: Instruction set

Node XML Files

- We use XML files to define the **nodes** in the graph
  - What packages to install
  - What to do at <post> installation

- We also use XML files to define the **graph structure**

<package> Tag

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

Convention

- <roll>-server.xml
  - Things you install and configure only on Frontends
- <roll>-client.xml
  - Things you install and configure only on “client” nodes (eg. Compute, NAS, VM-containers, …)
- <roll>-common.xml
  - Things installed everywhere

Where the art is: <post>

- Package Creation ranges from trivial to not-so-trivial
- Defining where packages go, some on this appliance, some on that. Straightforward
- But, the post section …

Nodes Post Section

- Scripts have minimal $PATH (/bin, /usr/bin)
- Error reporting is minimal
  - Follow “Day in the Trenches” presentation by Greg Bruno
- 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">
  - Arbitrary Attributes - In 5.3 most tags can have conditionals based on attributes

Post Example: PXE config

for an x86_64 machine:

```
<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>
…
```

```
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…
```

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">

Rocks release:
<eval>
date +"%d-%b-%Y"
echo ""
cat /etc/rocks-release
</eval>

</file>
```

```
…RCS checkin commands...
cat > /etc/hi << ‘EOF’

Rocks release:
13-May-2005

Rocks release 4.2.1 (Cydonia)

EOF
…RCS cleanup commands…
```

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>
```

Valgrind – A working example

- In our case, it’s so simple we don’t have to change the node file
- Inspect the node file, in any case

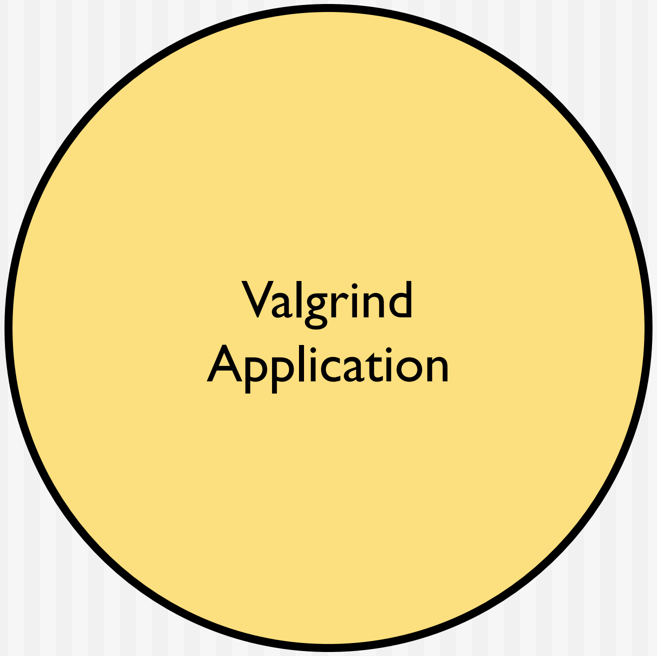
```
<kickstart>
   <package>valgrind</package>
   <package>roll-valgrind-usersguide</package>
</kickstart>
```

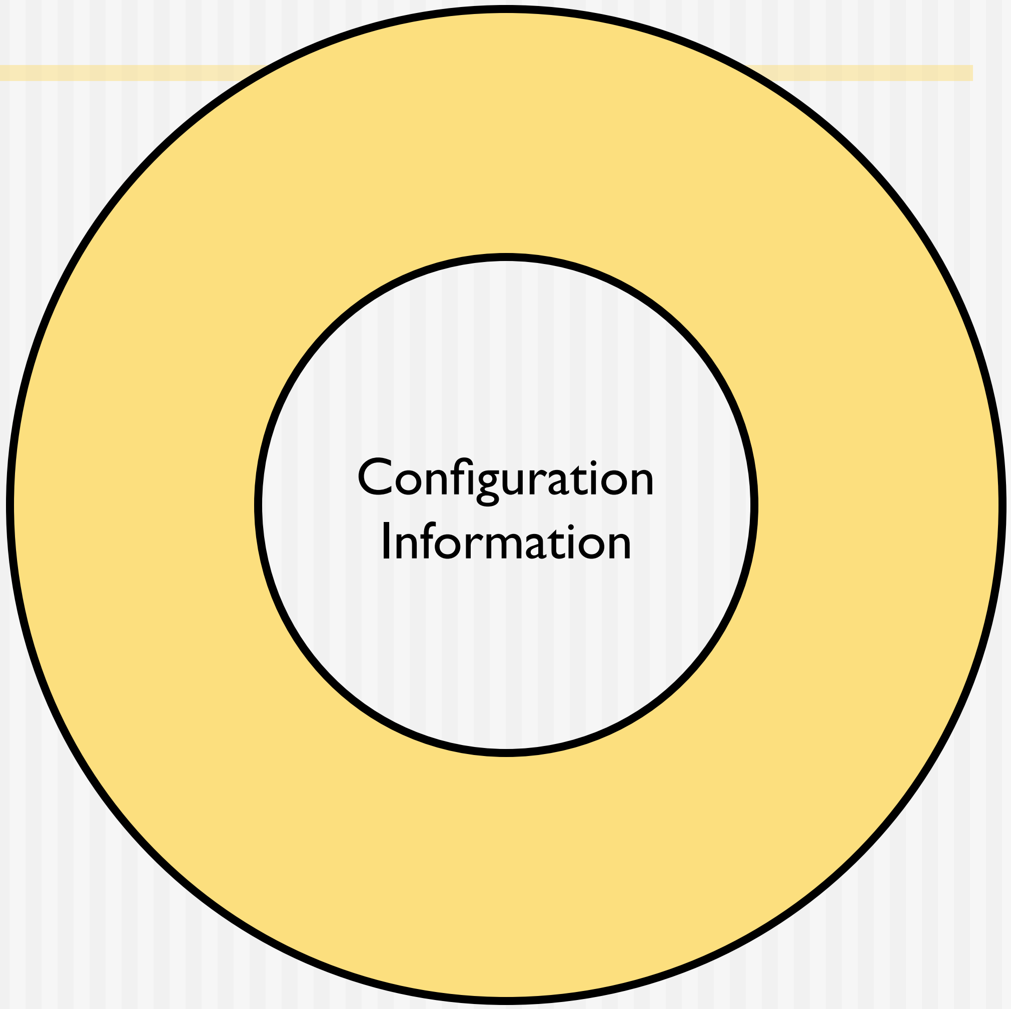
Roll Building

```
#  gmake roll 1>build.log 2>&1 </dev/null &
[1] 3627
[root@aurora valgrind]#
[root@aurora valgrind]#
[1]+  Done                    gmake roll > build.log 2>&1 < /dev/null
[root@aurora valgrind]# ls valgrind-3.5-0.x86_64.disk1.iso
valgrind-3.5-0.x86_64.disk1.iso
[root@aurora valgrind]# rocks add roll valgrind-3.5-0.x86_64.disk1.iso
Copying valgrind to Rolls.....39934 blocks
[root@aurora valgrind]#
```

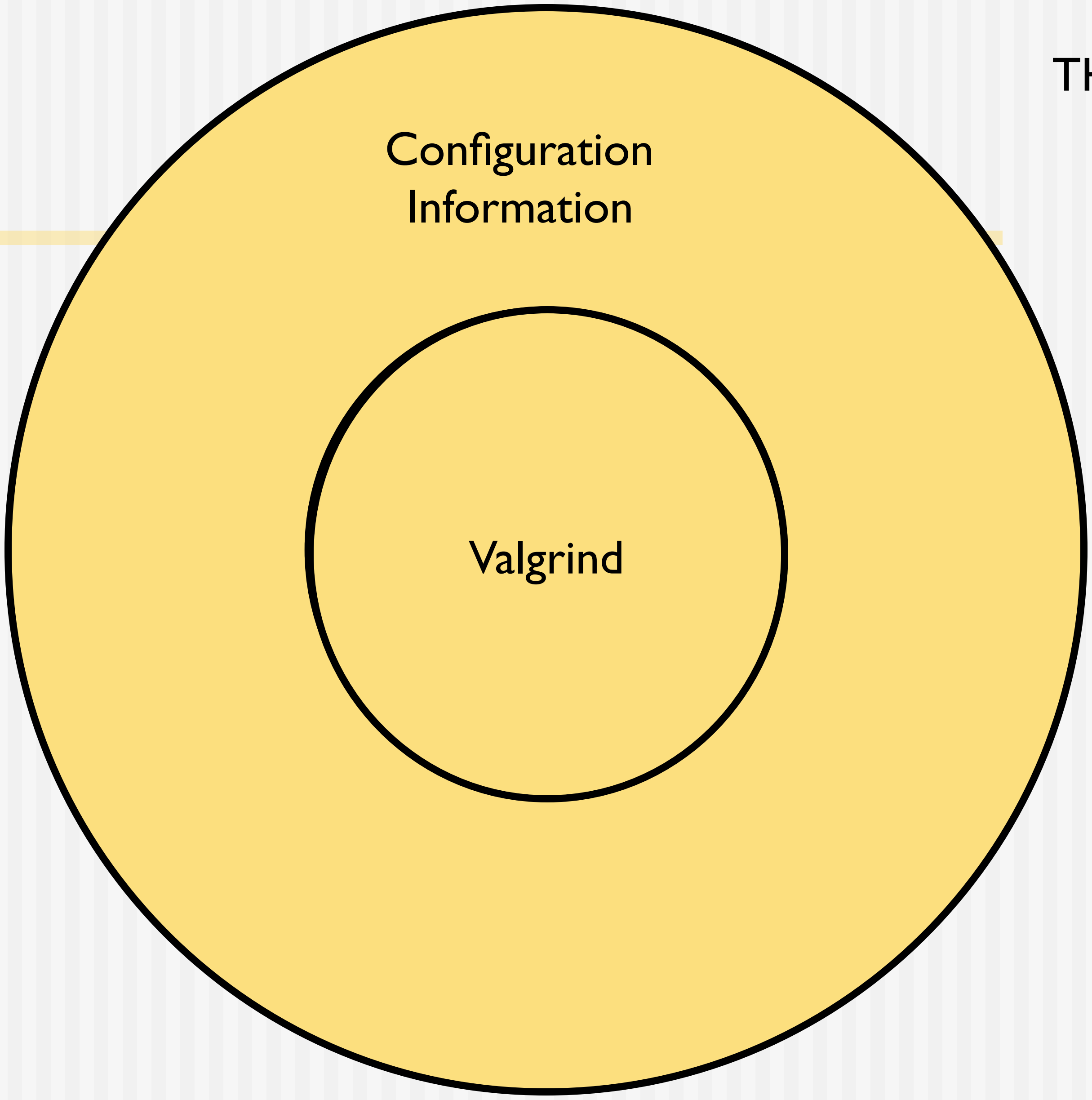
- And you’re done

To Re-iterate

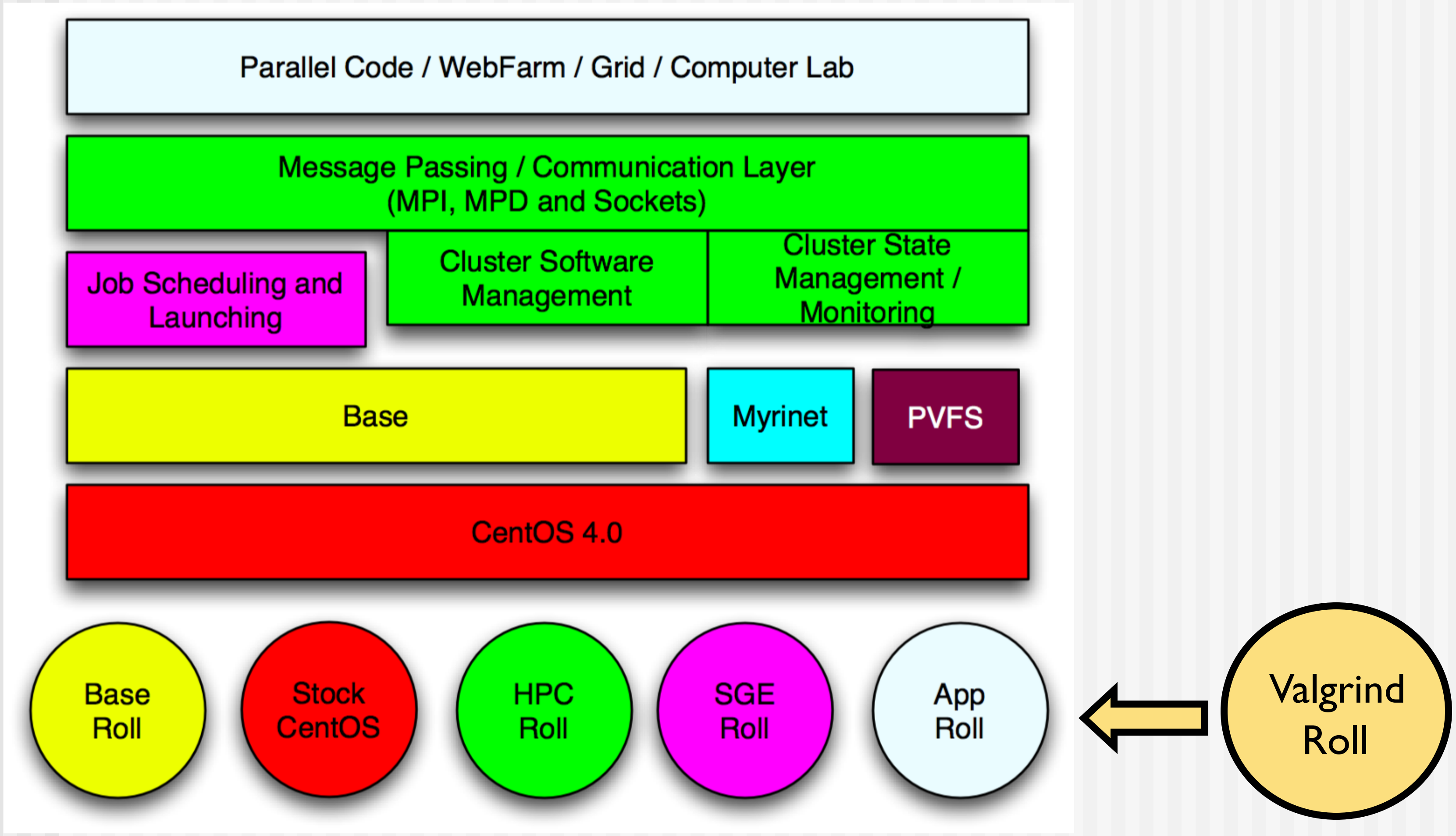



Configuration
Information

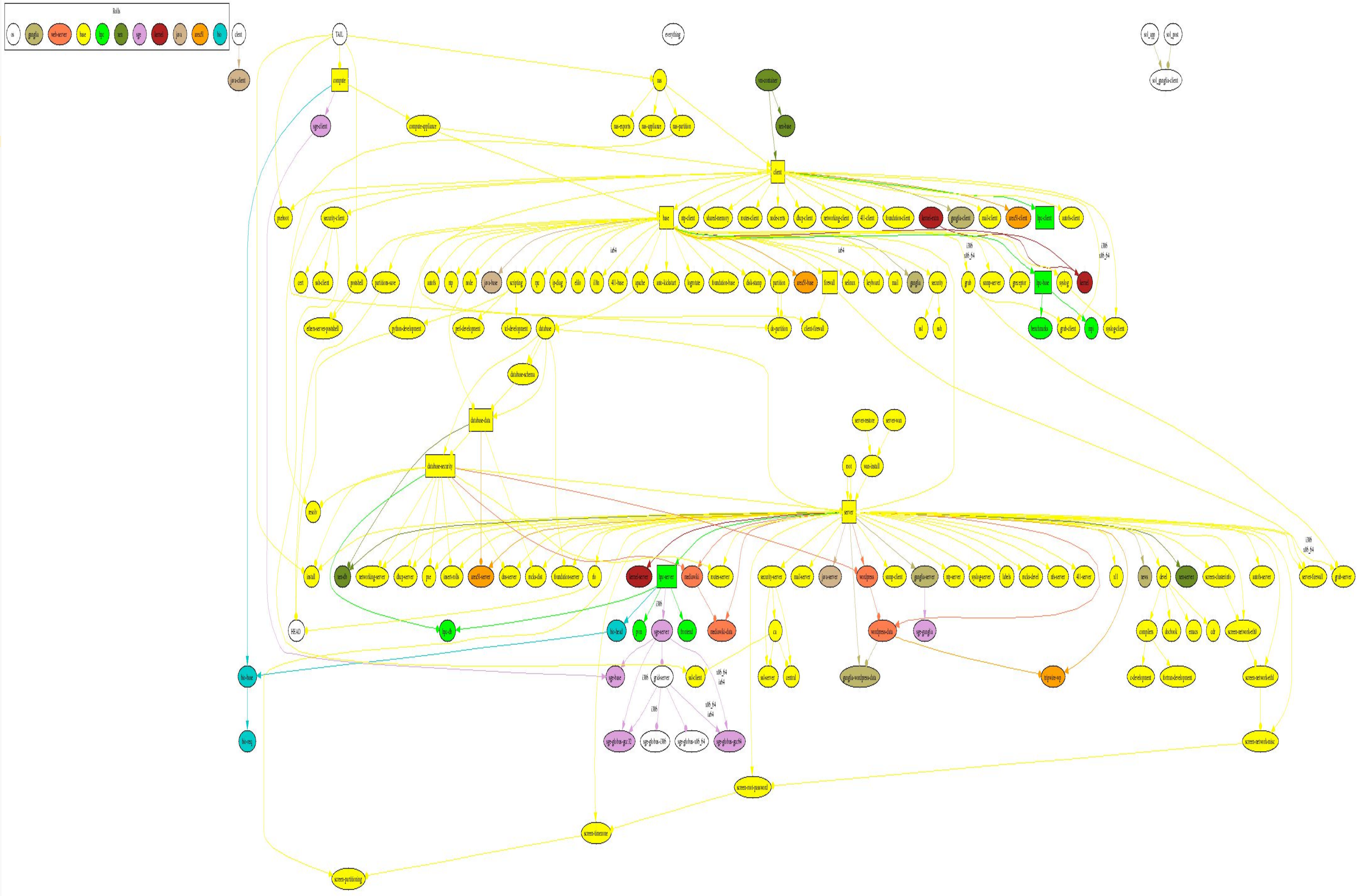
THE ROLL



How does it fit in?


Parallel Code / WebFarm / Grid / Computer Lab
Message Passing / Communication Layer (MPI, MPD and Sockets)
Job Scheduling and Launching
Cluster Software Management
Cluster State Management / Monitoring
Base
Myrinet
PVFS
CentOS 4.0
Base Roll
Stock CentOS
HPC Roll
SGE Roll
App Roll
Valgrind Roll

How does it fit in?



How does all this help you?

- Roll mechanism is the recommended way of deploying software on a Rocks cluster
- It fits into the framework of Rocks
- It's reproducible
- It scales

Summary

- Look at the Rocks Rolls for examples.
- Rolls are not difficult, Understanding what is going on under the covers helps demystify
- Some software is more challenging than others
- Test. Test. Test.