All Plumbing Needs an API

Andy Grover
agrover@redhat.com

github.com/agrover
@iamagrover
freenode, oftc irc: grover

August 29, 2012

In the beginning, there was the command-line

- Text streams, pipes, and text processing
- Sysadmins manage 1-10 machines
- Write scripts to automate repetitive work
  - sed/awk -> Perl -> Python & Ruby

Today...

- Virt & Cloud increase machine:admin ratio
  - now 100:1, 1000:1
- From “sysadmin scripts” to “IT automation”
- Tools must evolve from “scriptable” to “automatable”

Programs vs Scripts

- Fast, small
- Compiled
  - Not (usually) rebuilt or modified by admin
  - Configurable via cmdline opts or config file

- The sysadmin automates rote tasks
- Extremely specific to the environment
  - hardcoding is ok
- Gluing together or iterating
- Sysadmin knows the environment
- Sysadmin around to fix errors

Dynamic languages like Python & Ruby make it easy to start with a script, grow it into a reusable library, then into a framework

Even though it may be written in a “scripting” language, it should now be considered a “program”

Programs written like scripts!!!

- Programs that call system() or subprocess()
- Huge wrapper libraries around building cmdlines, parsing cmdline, regexps, and handling errors

- oVirt-vdsm
- OpenStack Nova
- udisks2
- ssm

- libvirt
- tgtd
- Anaconda

A program calling command line tools is the moral equivalent of web scraping

Why we should fix this:

Extra new process and subshell

Example 1

```
_execute('kill', '-9', pid, run_as_root=True)
```

- Should be using stdlib functions:

```
import os
import signal
os.kill(pid, signal.SIGKILL)
```

Easy to ignore exit code & stderr messages

Example 2

```
cmd = ('lvcreate', '-L', '%db' % size, '-n', lv, vg)
execute(*cmd, run_as_root=True, attempts=3)
```

Constructing the command-line and parsing the result

Example 3

```
out, _err = _execute('cat', '/proc/%d/cmdline' %
pid, check_exit_code=False)
```

Hidden dependencies are not expressed in package manager

Command-line executables must be present even when embedded platform

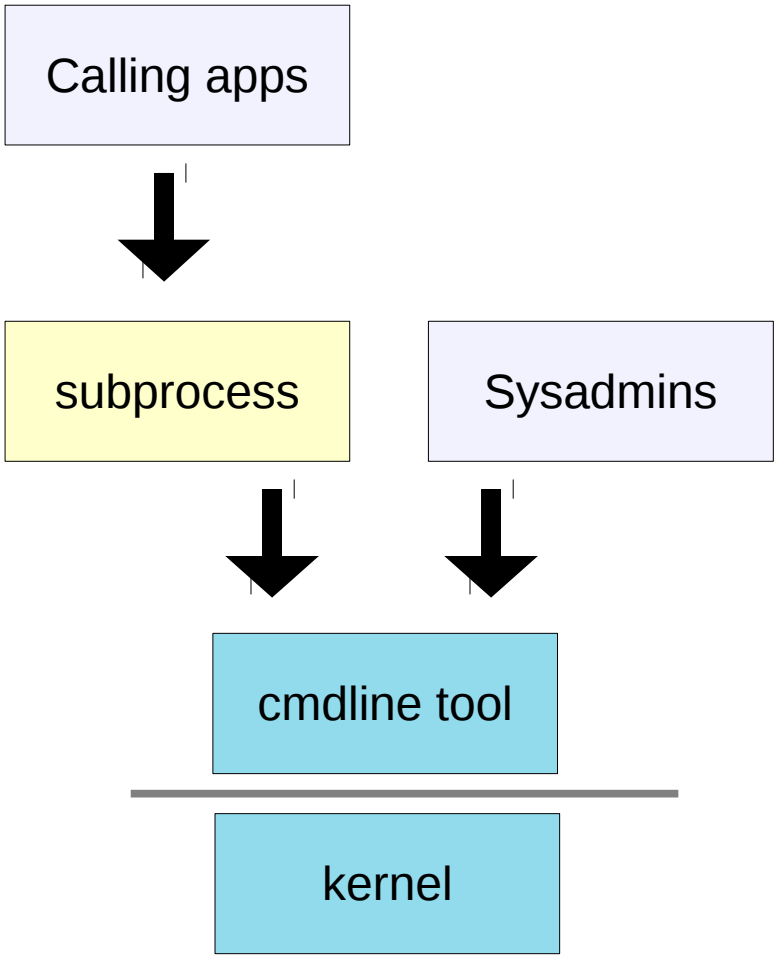
Different platforms may have different command-line option flags

Security?

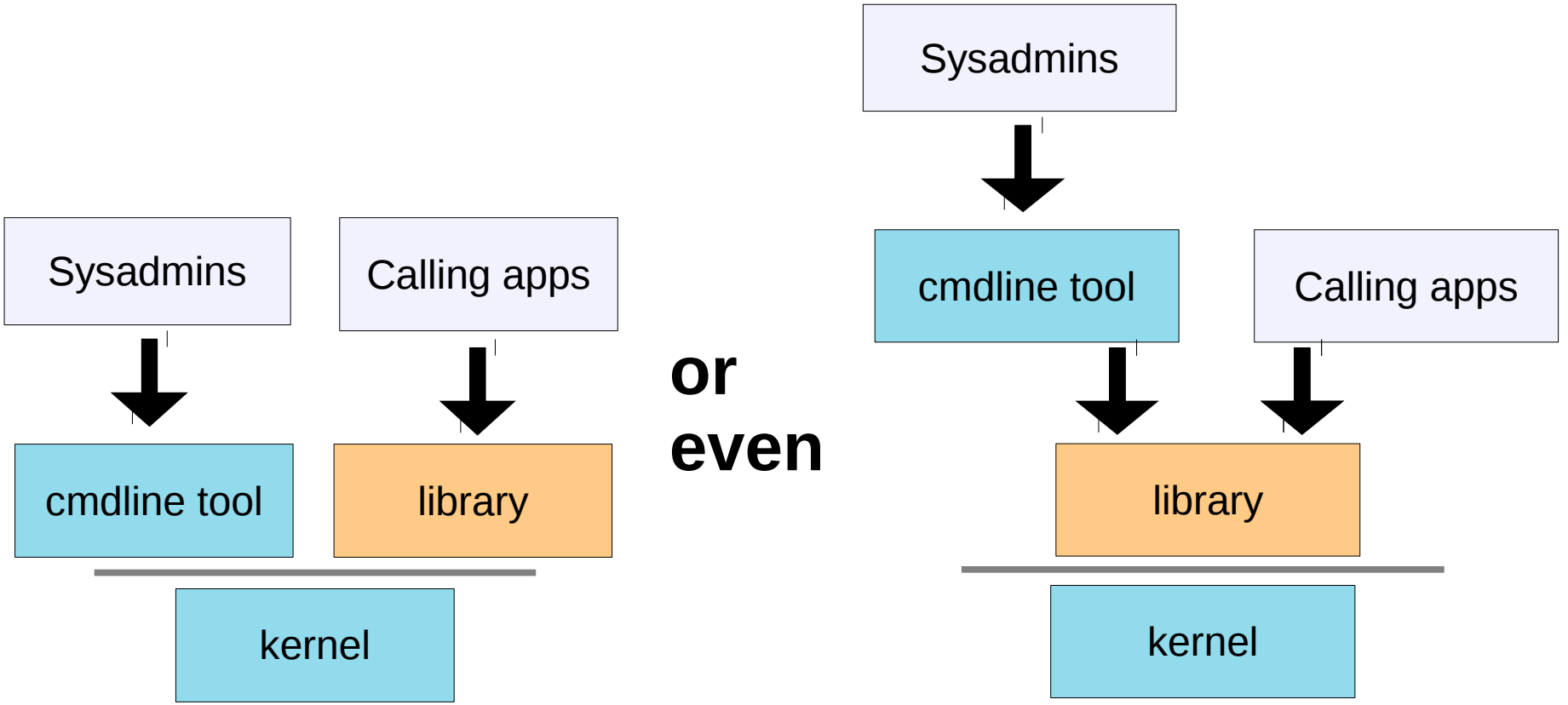
- Calling set[gu]id binaries
- Command-line injection opportunities
- No installed cmdline = less security surface

How to “libify” your code:

Starting with this...



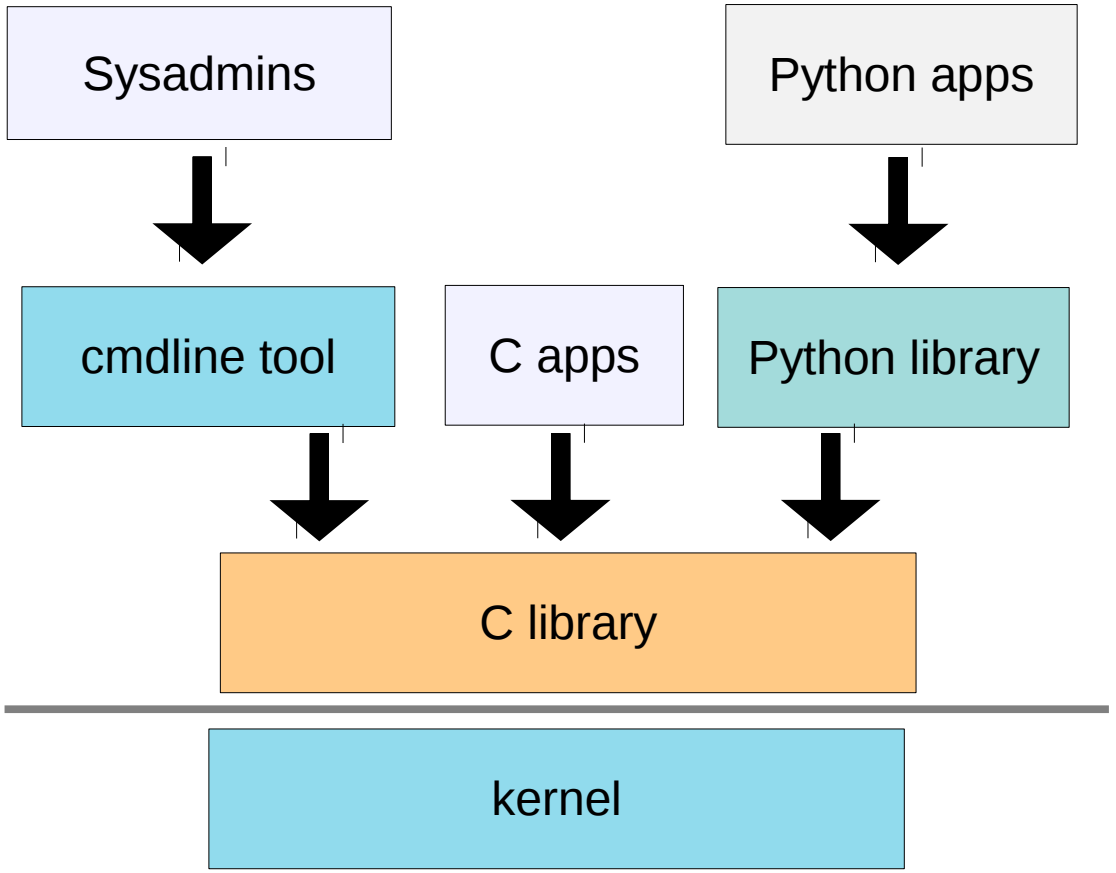
Add library and get:



Implementing the library

- Ideally C and Python
  - But C *or* Python also ok
- C lib maximizes cross-language accessibility
  - Somebody needs to write language wrappers
    - (Doesn't have to be you)
- Python lib helps majority of users
  - HLL features -> less user code
  - Other HLL language lib authors can use as a template

Leading to this



Implementing a C library/API

- Check out libabc!
  - http://0pointer.de/blog/projects/libabc.html
- Doesn't have to be 100% coverage
- Doesn't have to be perfect
  - libs have versioning
- Figure out memory allocation model
  - refs, handle, caller-allocated

Implementing a Python/C wrapper

- Call your C lib, convert to Python objects, expose Python objects and methods
- Some boilerplate
- Shamelessly copy (python-kmod, python-lvm)
- Add value with OO, lists, exceptions, docstrings, restricted featureset
- SWIG good, ctypes meh

Implementing a pure Python library

- Easiest
- No C needed when wrapping a configfs/sysfs interface
- Add value with OO, lists, exceptions, docstrings, restricted featureset
- Example for other HLL lib writers
  - Ruby, OCaml, Haskell, Java, etc.

Summary for potential library users

- Internalize that system() is a stopgap
- Gradual transition to APIs is possible
  - Use APIs that already exist (esp. in stdlib)
- Push for proper APIs in your language from your dependencies
  - Open bugs against your codebase
  - And theirs
    - Best bugs have patches attached

Summary for potential library authors

- Having an API will ultimately make your life easier
- Base APIs on how dependent projects are using your cmdline
- Push users to move to your new API
  - Open bugs against cmdline-using apps
    - Best bugs have patches attached
- Minimum Viable API
- APIs are not forever
- Use your API yourself

Some kernel/plumbing stuff that should have libs/APIs

- networking config
- storage config
- filesystem tools
- process accounting
- cgroups
- power management
- perf
- tracing
- iproute
- remote storage config
- more?

Thanks!

Questions and Discussion?

Backup

Summary

- Low-level tools need cmdline for sysadmin cmdline usage and basic scripting
- They ALSO need library interface so other programs can use it cleanly
- Without an API, programs resort to system()
- This is bad
- We should fix this
- It won't be too hard, and is worth the effort

Ousterhout's dichotomy

- Roughly: “systems programming languages are very different from scripting languages”
- Code in C == building blocks
  - “programs”
- Code in script == aids pasting blocks together in the UNIX tradition
  - “scripts”
- Where do Python and Ruby fit?