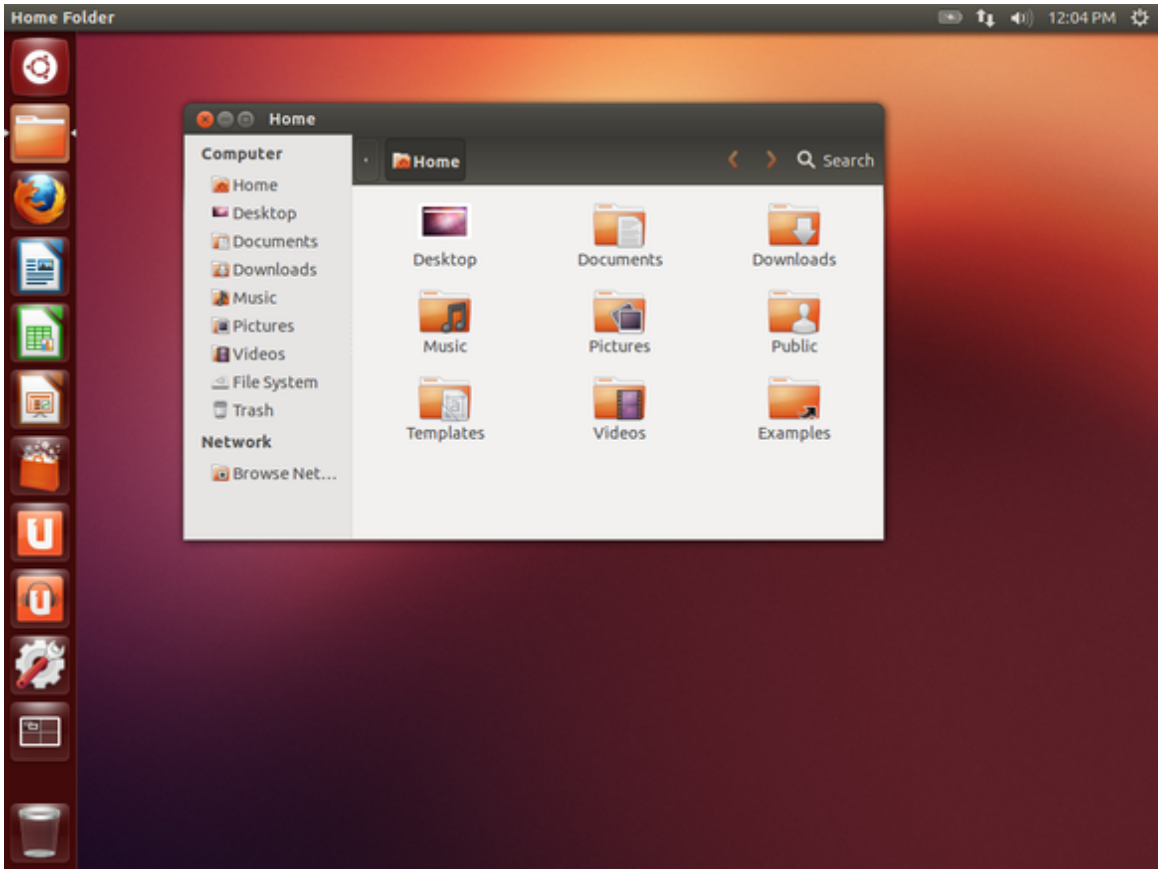
Snap your App!

Ted Gould
ted@canonical.com
@tedjgould
Ubucon
November 19th, 2015

> “ Ubuntu Core provides transactional updates with rigorous application isolation. This is the smallest, safest Ubuntu ever, on devices and on the cloud. We’re excited to unleash a new wave of developer innovation with snappy Ubuntu Core! ”
>
> Mark Shuttleworth, founder of Ubuntu and Canonical.

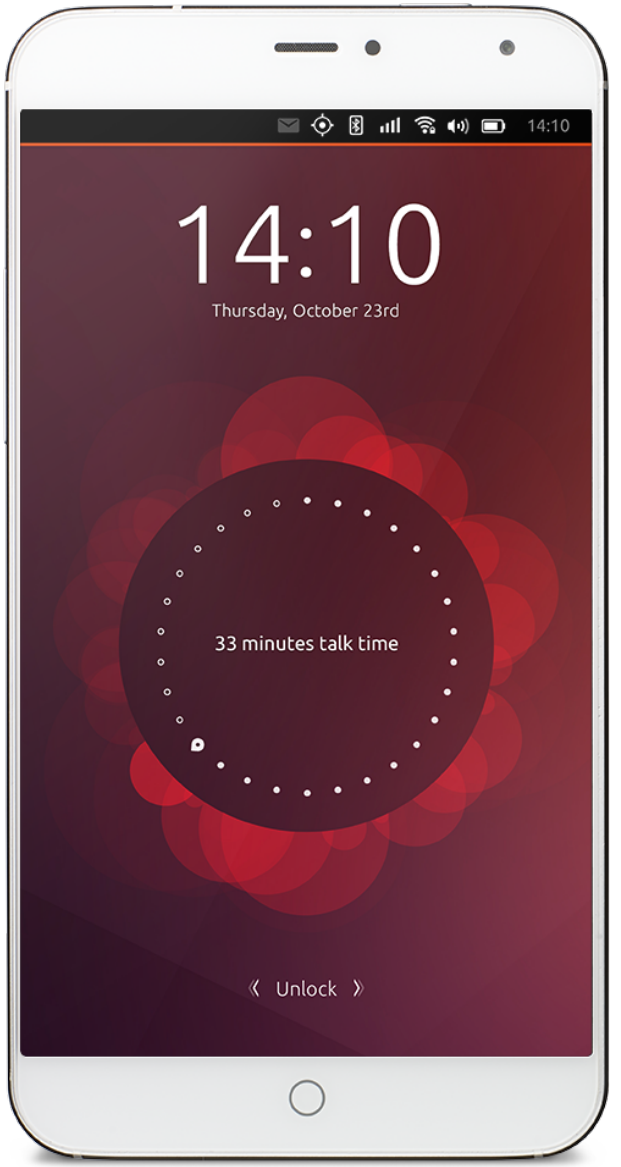
Ubuntu

- Debian based (package format and policies)
- Regular Releases
- Updated via Apt



Ubuntu Phone

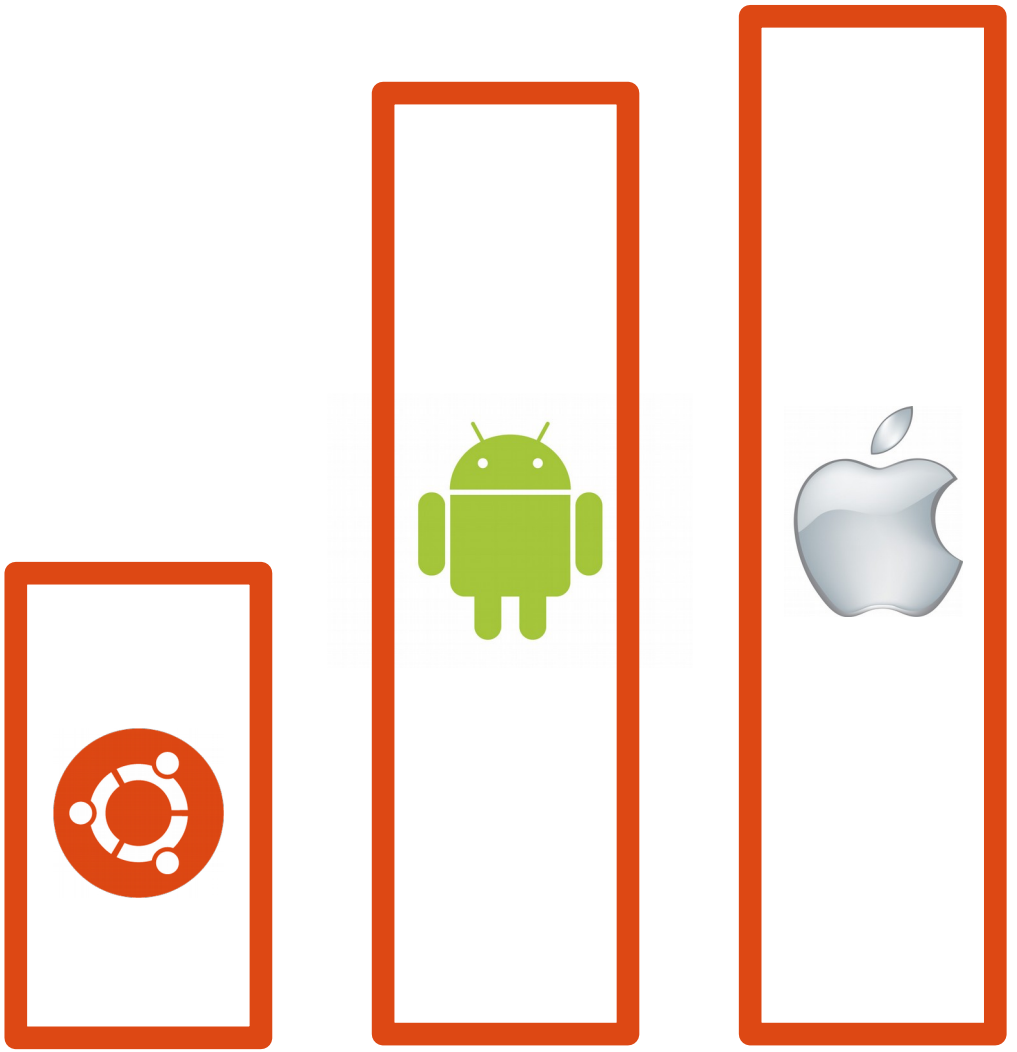
- Click Applications
- System Image Updates



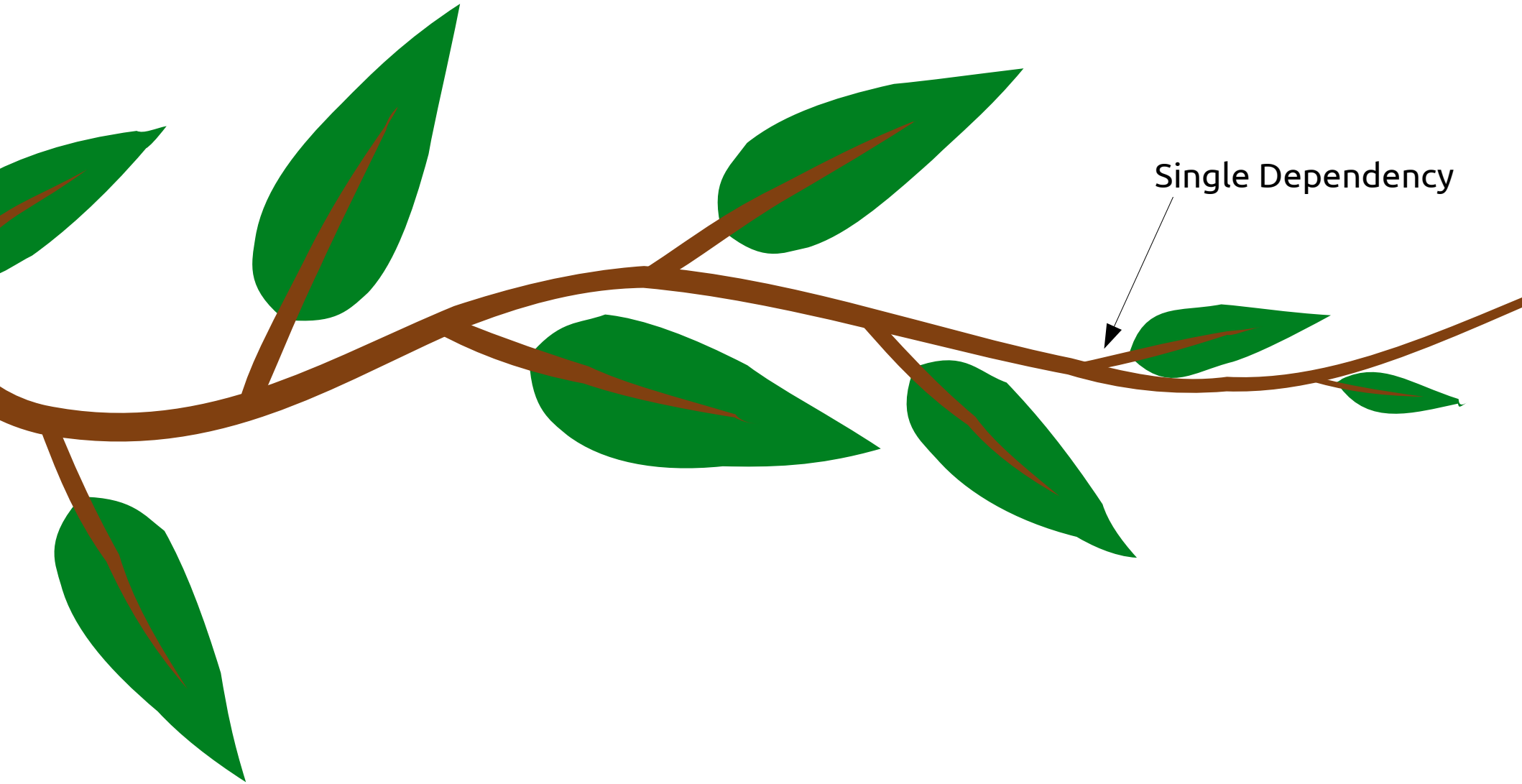
Why Click?

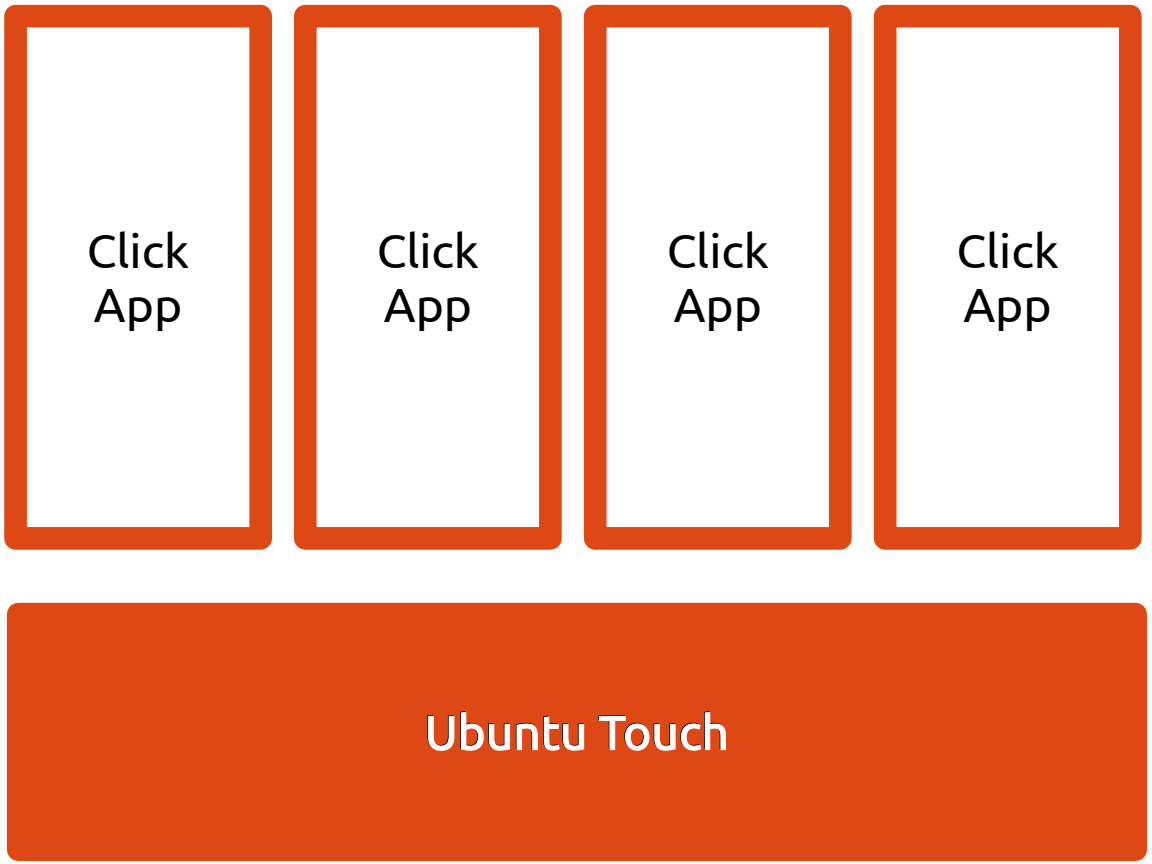
Millions

Thousands



Leaves are Simpler

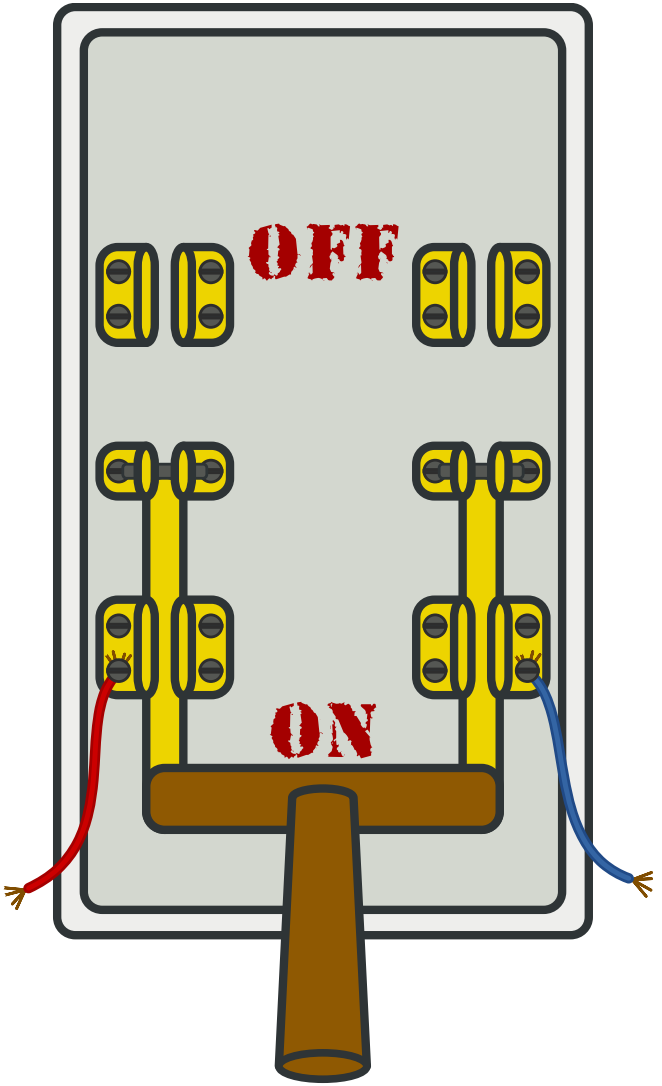



Click App
Click App
Click App
Click App
Ubuntu Touch

Reviewers Don't Scale

Why Image Based Updates?

- Binary: Works or it doesn't
- Provides opportunity to rollback
- Simple testable states



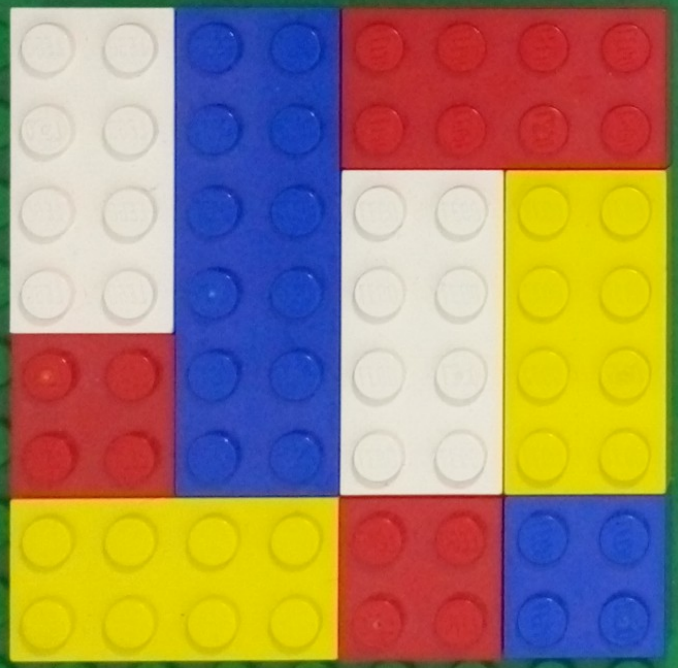
“Let's Take Ubuntu Phone all the way to 11!”

— Us, acting cooler than we actually are

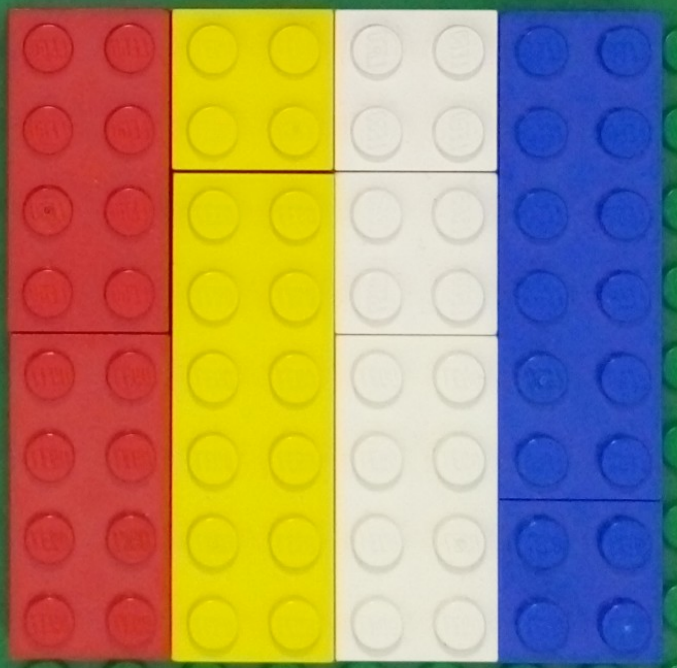
Snappy Ubuntu Core

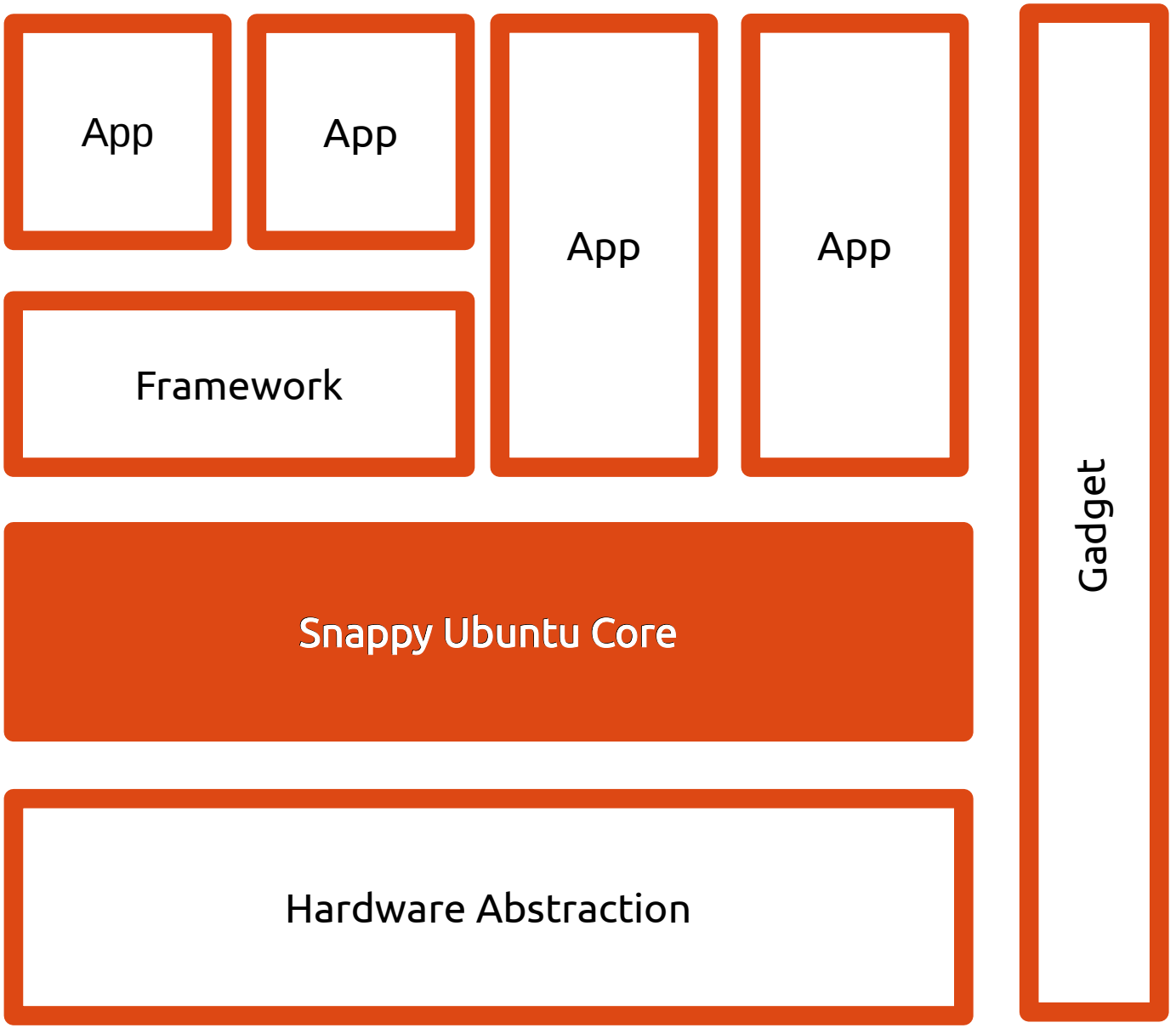
- Transactional Updates
- Snap based Apps
- Snap frameworks
- OS snap

Deb-based System



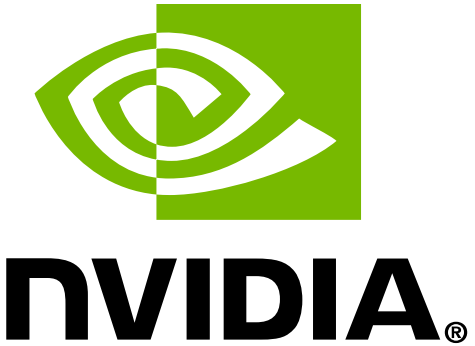
Snap-based System




App
App
App
App
Framework
Gadget
Snappy Ubuntu Core
Hardware Abstraction

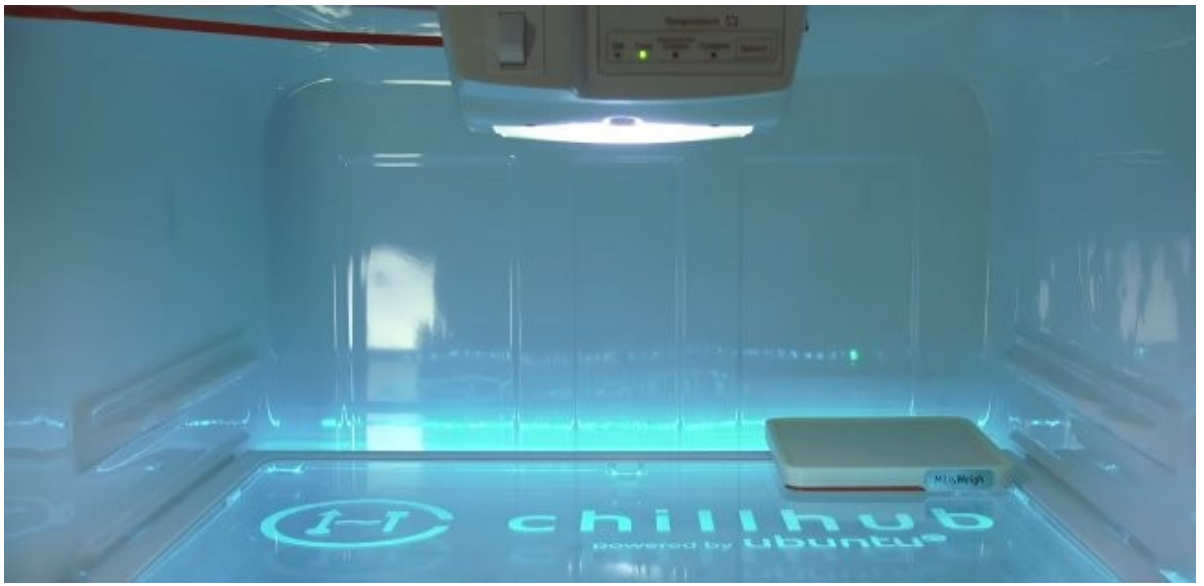
Hardware Abstraction

- Provided by board vendors
- Allows for custom drivers and config



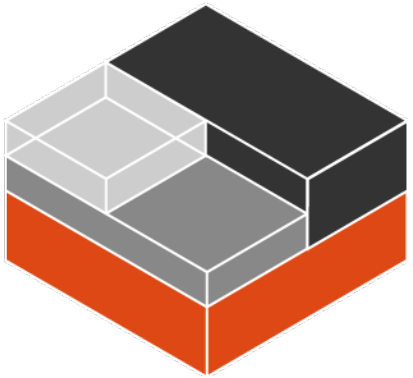
“Gadget” Snap

- Per-device configuration
- Snaps to install
- Permissions
- Branding



Framework Snaps

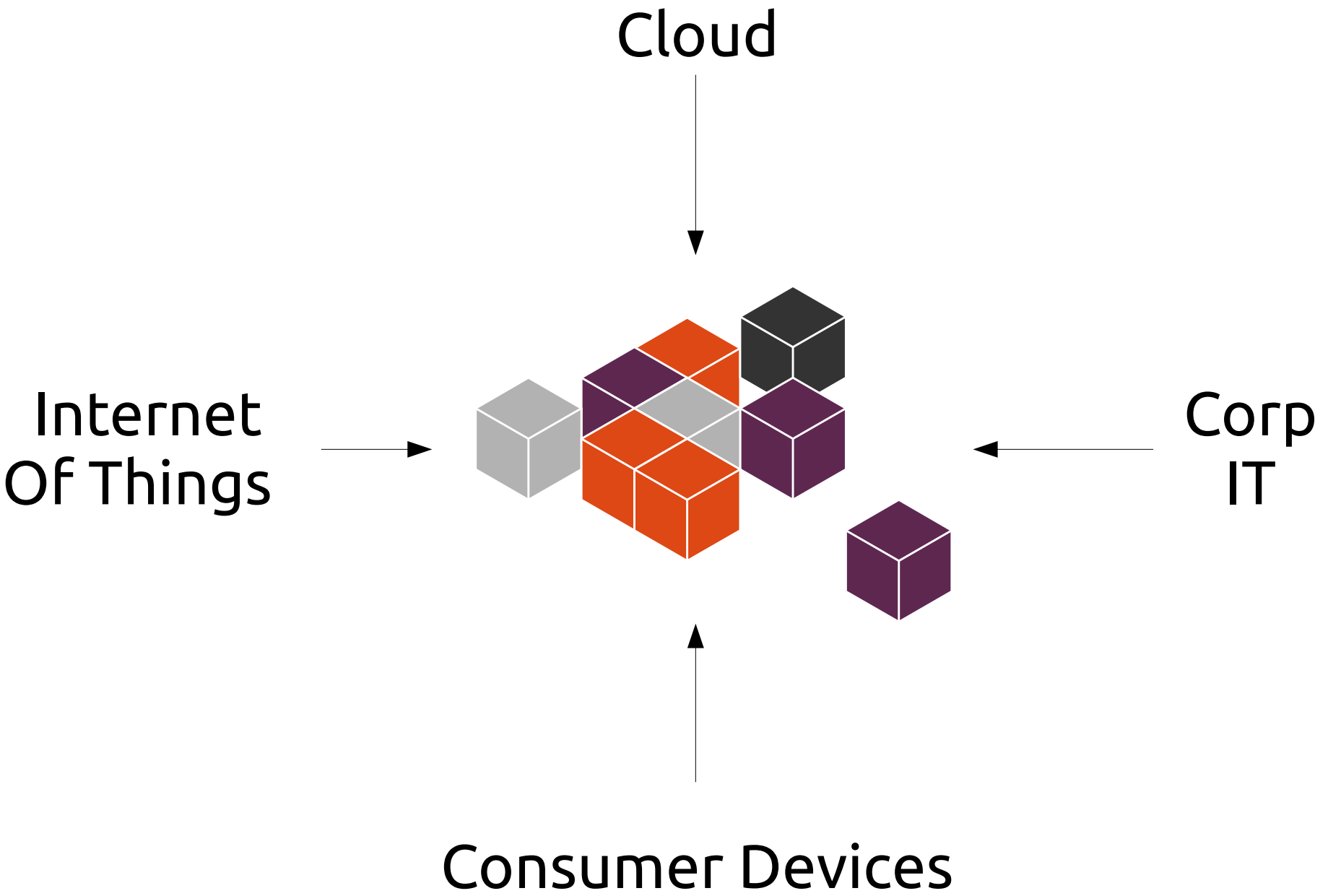
- Provide shared services
- Mediate resources
- IPC to Apps



Mir System
Compositor

Why snap? (not click)

- Click v2.0
- Click only for leaf nodes
- Support for OS and framework snaps
- Lower level components


Cloud
Internet
Of Things
Corp
IT
Consumer Devices

Trend: Internet of Things

- Smaller computers
- Internet connectivity
- Full OS resources
- Must be reliable

Trend: Consumer Devices

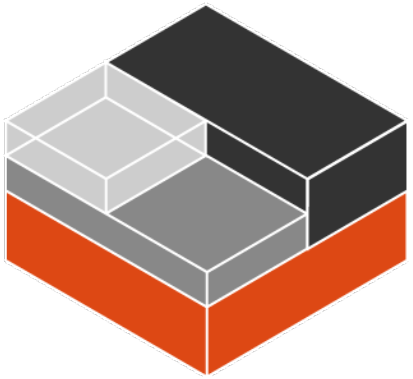
- More complex interactions
- Bullet-proof experience
- Complex security situations

Trend: Corporate/Education IT

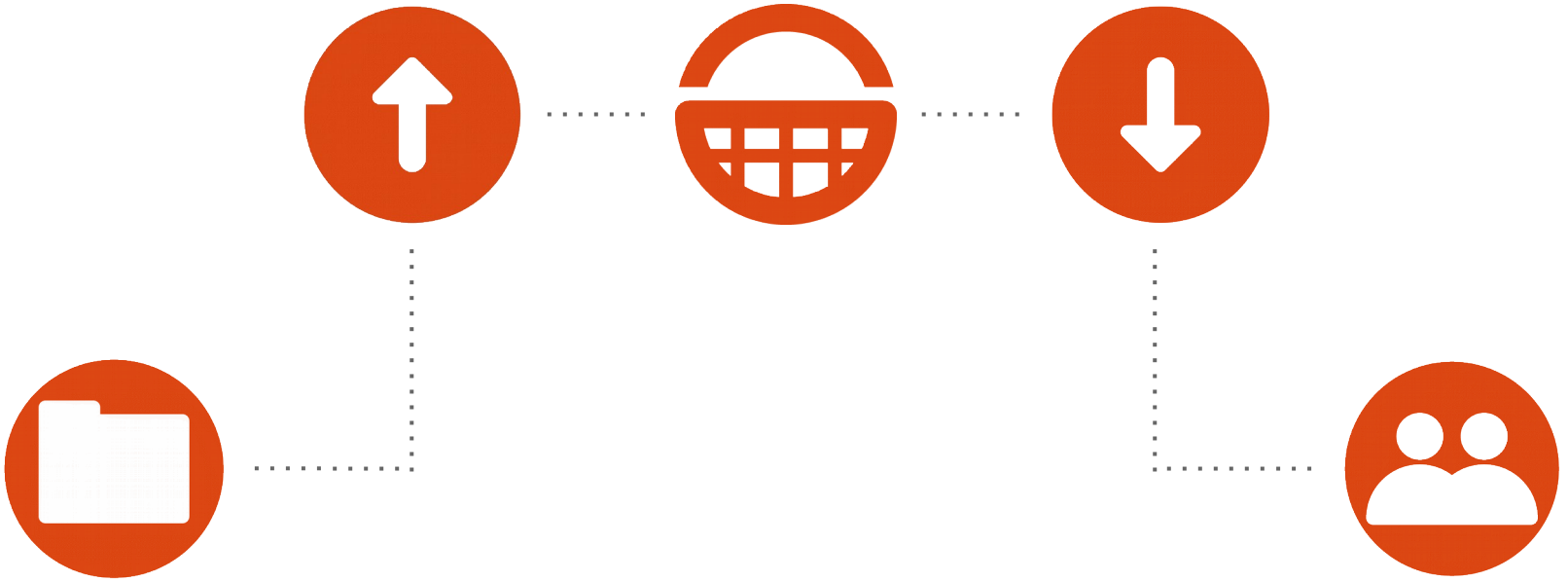
- Verifiable Images
- Upgrades don't cause downtime
- Confined apps and app permissions

Trend: Cloud and Containers

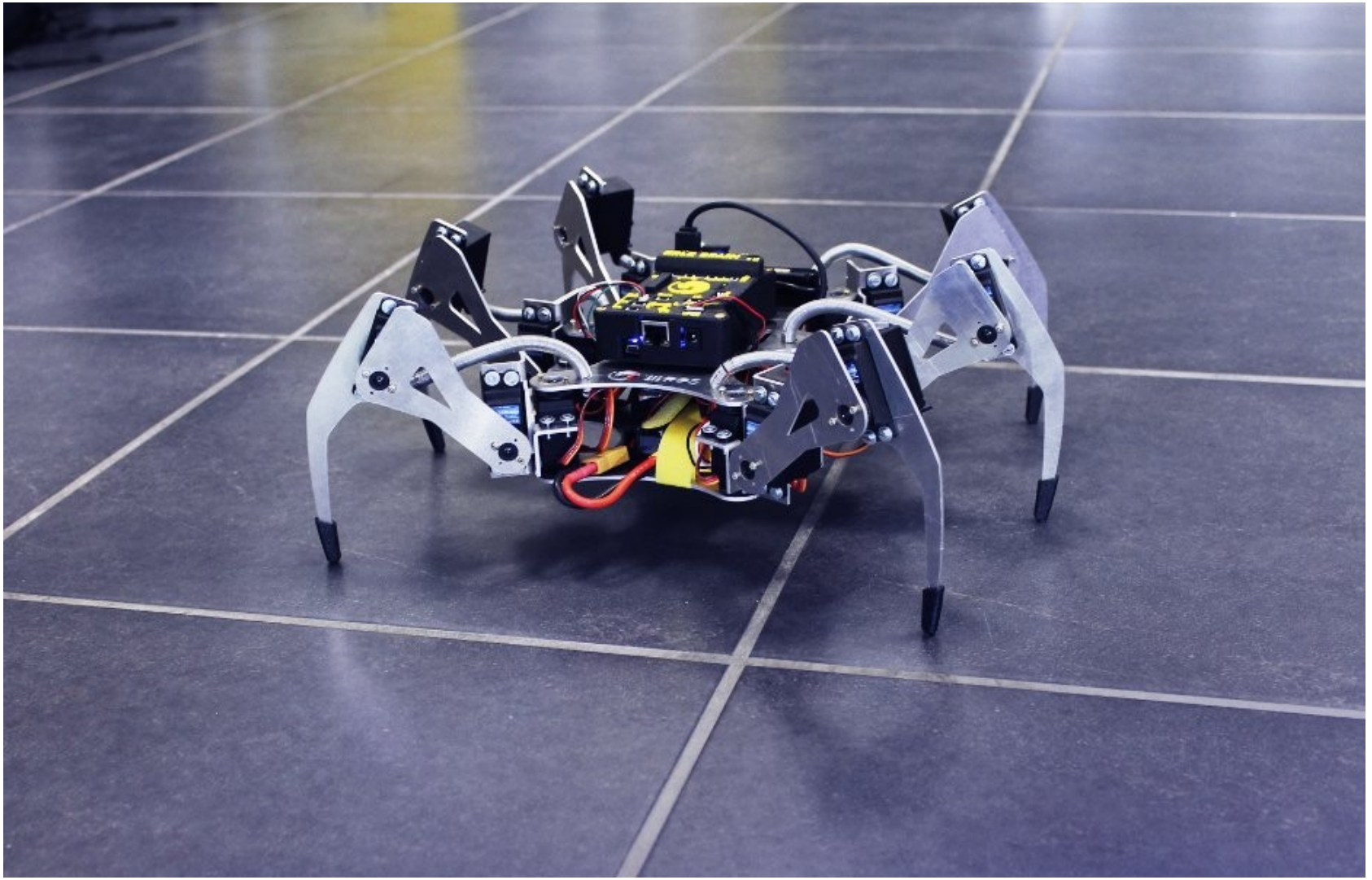
- Small base OS
- Unit of specialized code
- Tested as a unit



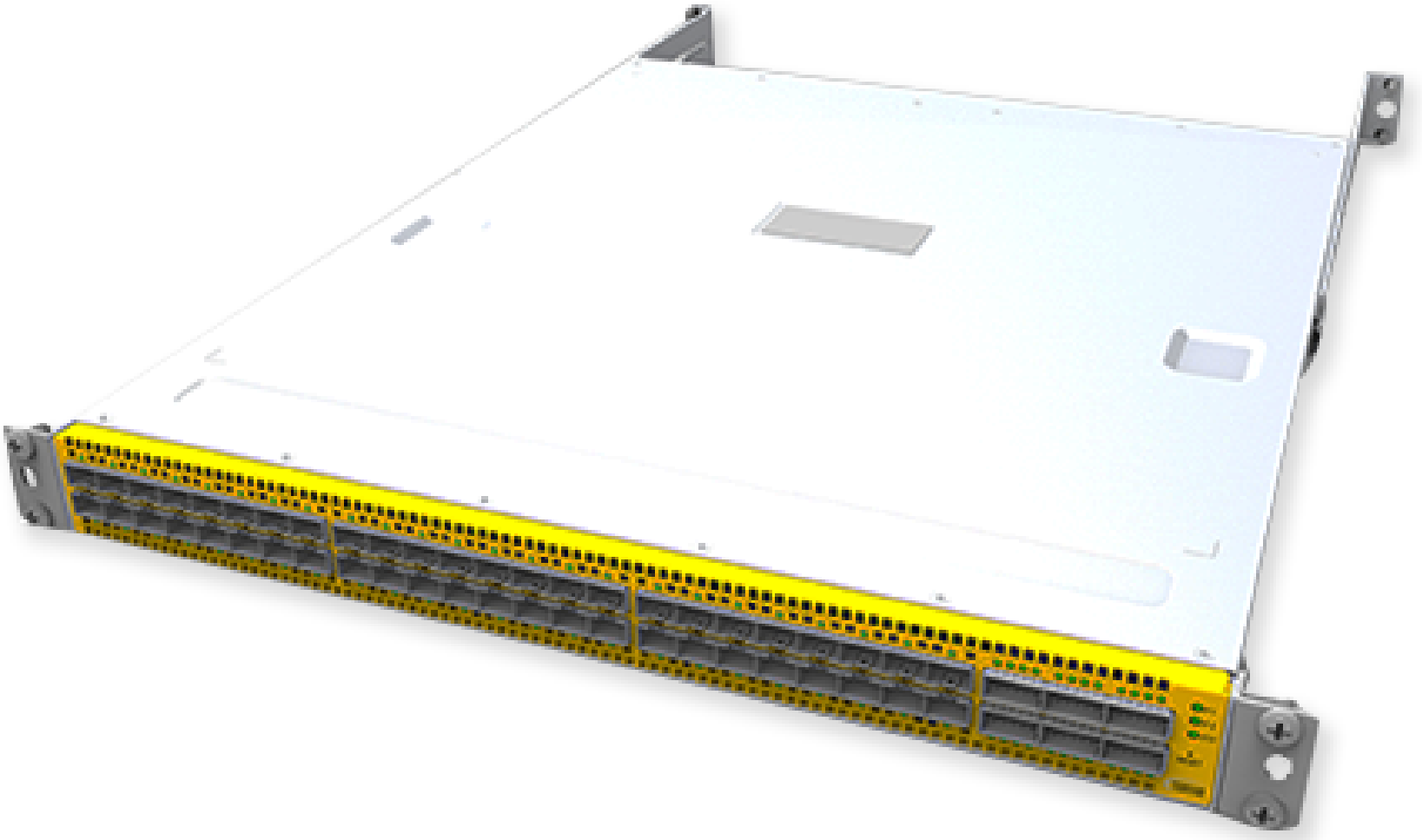
Snappy Store



Store Story: Erle Robotics

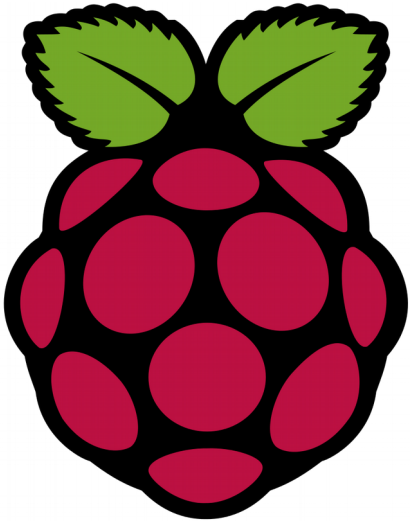


Store Store: Networking



Try it today!

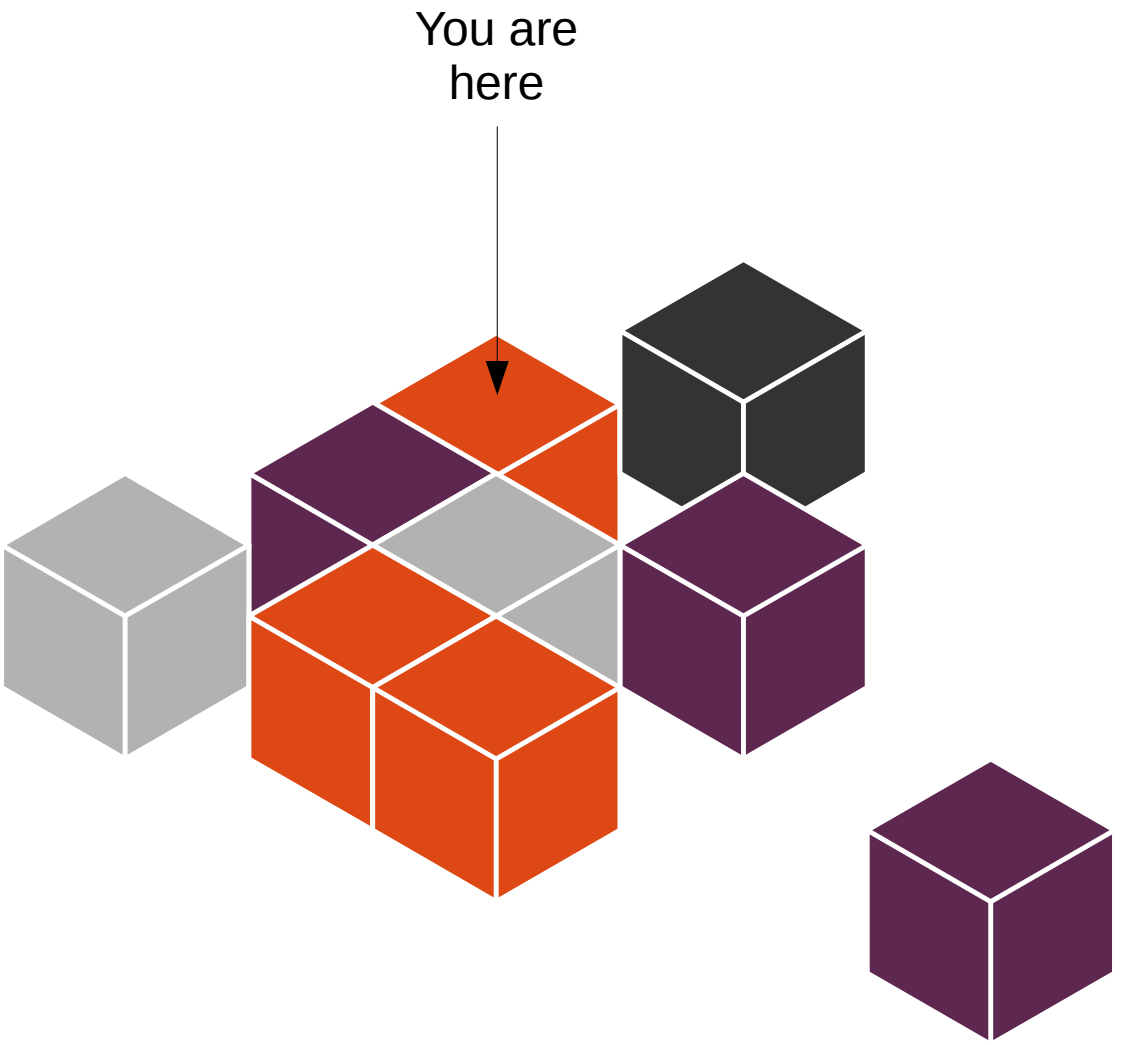
Microsoft Azure



http://ubuntu.com/snappy

What is a snap?

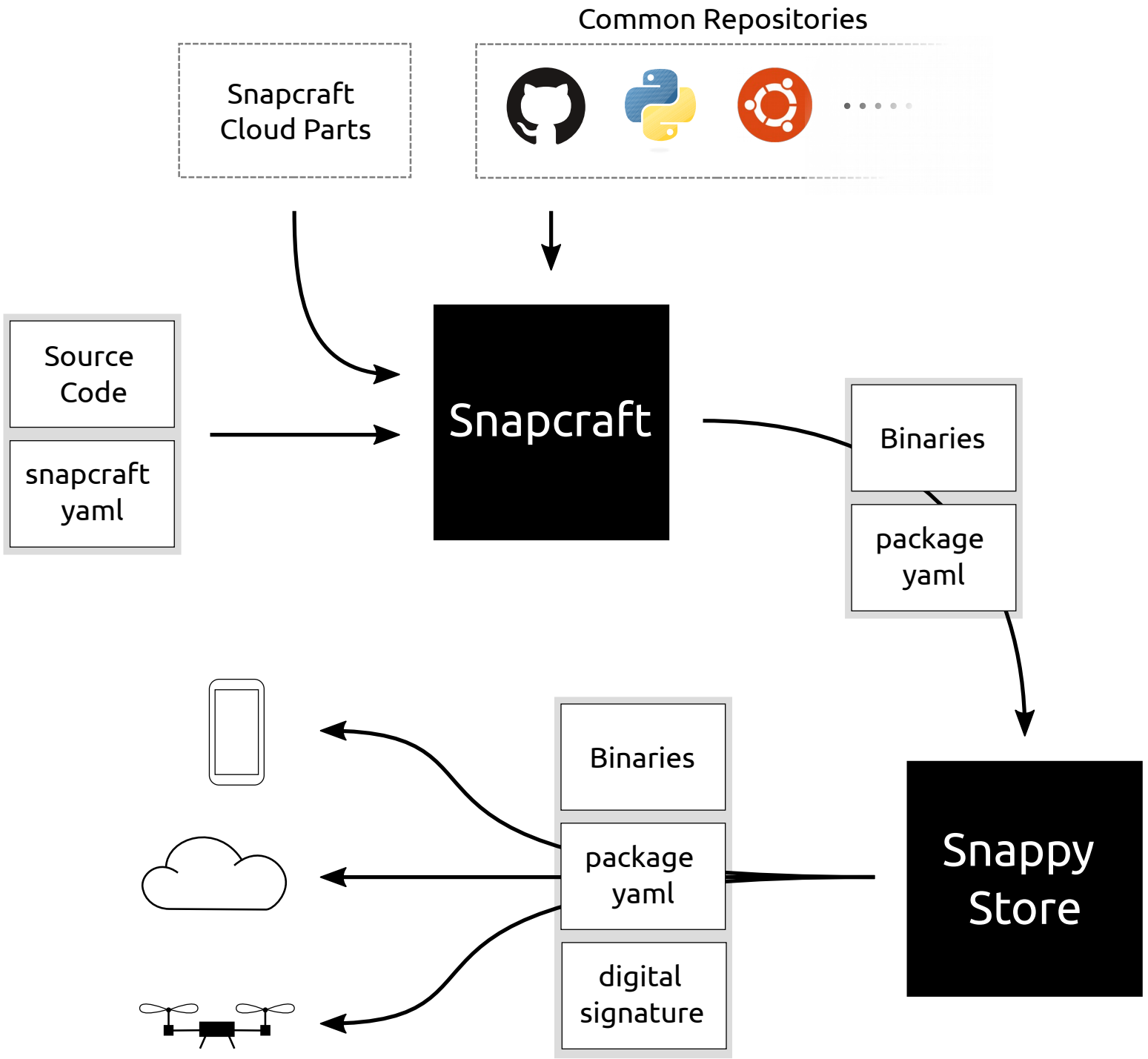
- A self-contained bundle
- Includes dependencies
- Works with frameworks



Snap Directories

Directory	Writable?	Description
/apps/<app-name>/<version>/	No	Read-only files, libraries, resource files, and other binary data shipped with the app.
/var/lib/apps/<app-name>/<version>/	Yes	Writable files, configuration or other data that is not specific to any user. This directory needs to be created by the application right now and will be created by the snappy tool soon.
/var/lib/apps/<app-name>/<old-versions>	No	Read-only for the apps, backup purpose.
/home/user/apps/<app-name>/<version>/	Yes	Writable, configuration or other data specific to the calling user. The app needs to create them right now. They will be created by the snappy tool soon.
/home/user/apps/<app-name>/<old-versions>/	No	Read-only for the configuration or other specific data for the calling user, backup purpose. This is not created by snappy right now but will be soon.

https://developer.ubuntu.com/en/snappy/guides/filesystem-layout/


Common Repositories
Snapcraft
Cloud Parts
Source
Code
snapcraft
yaml
Snapcraft
Binaries
package
yaml
Binaries
package
yaml
digital
signature
Snappy
Store

Block Inception

snapcraft.yaml

```
name: photoviewer
version: 0.2
vendor: Ted Gould <ted@canonical.com>
frameworks: [mir]
summary: Photoviewer from Flickr tags

binaries:
  photoviewer:
    exec: qmlscene main.qml --
    caps:
      - mir_client
      - network-client
```

```
parts:
  qml:
    plugin: qml
  photoviewer:
    plugin: copy
    files:
      main.qml: main.qml
      PhotoViewerCore: PhotoViewerCore
```

snapcraft.yaml

```
name: photoviewer
version: 0.2
vendor: Ted Gould <ted@canonical.com>
frameworks: [mir]
summary: Photoviewer from Flickr tags

binaries:
  photoviewer:
    exec: qmlscene main.qml --
    caps:
      - mir_client
      - network-client

parts:
  qml:
    plugin: qml
  photoviewer:
    plugin: copy
    files:
      main.qml: main.qml
      PhotoViewerCore: PhotoViewerCore
```

Package Information

snapcraft.yaml

```
name: photoviewer
version: 0.2
vendor: Ted Gould <ted@canonical.com>
frameworks: [mir]
summary: Photoviewer from Flickr tags

binaries:
  photoviewer:
    exec: qmlscene main.qml --
    caps:
      - mir_client
      - network-client
```

System Integration

```
parts:
  qml:
    plugin: qml
  photoviewer:
    plugin: copy
    files:
      main.qml: main.qml
      PhotoViewerCore: PhotoViewerCore
```

snapcraft.yaml

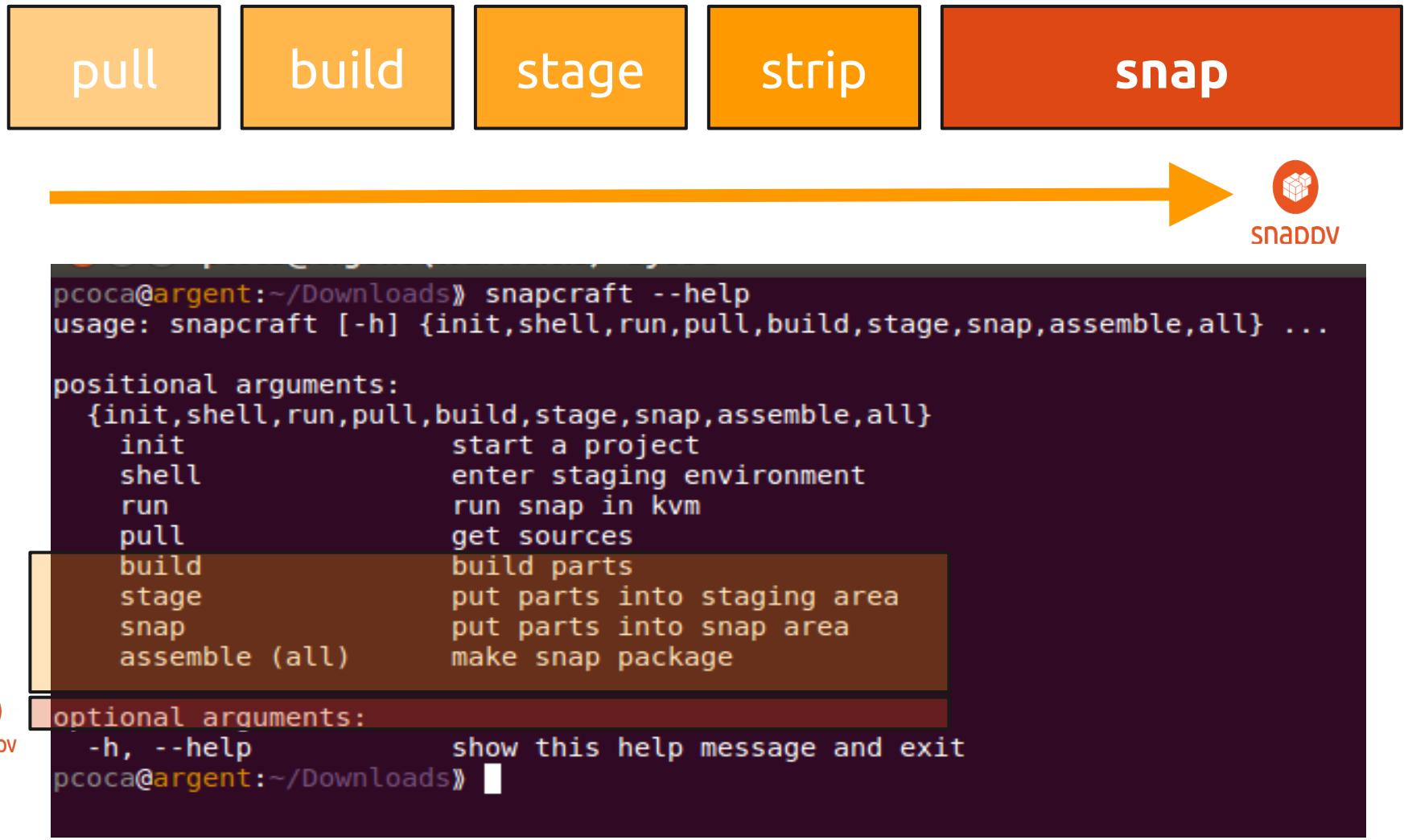
```
name: photoviewer
version: 0.2
vendor: Ted Gould <ted@canonical.com>
frameworks: [mir]
summary: Photoviewer from Flickr tags

binaries:
  photoviewer:
    exec: qmlscene main.qml --
    caps:
      - mir_client
      - network-client
```

Build Instructions

```
parts:
  qml:
    plugin: qml
  photoviewer:
    plugin: copy
    files:
      main.qml: main.qml
      PhotoViewerCore: PhotoViewerCore
```

Snapcraft lifecycle

This lifecycle diagram depicts the whole snapcraft process, that is composed by several steps with reflection in the snapcraft utility pertinent commands.



Snapcraft lifecycle

The pull phase takes care of the downloading / cloning of the remote files needed for this part.

Snapcraft will create a parts/ directory with sub-directories for each part that contains the downloaded content.

i.e: **`parts/part-name/src`**

This step will download content, e.g. checkout a git repository or download a binary component like the Java SDK.

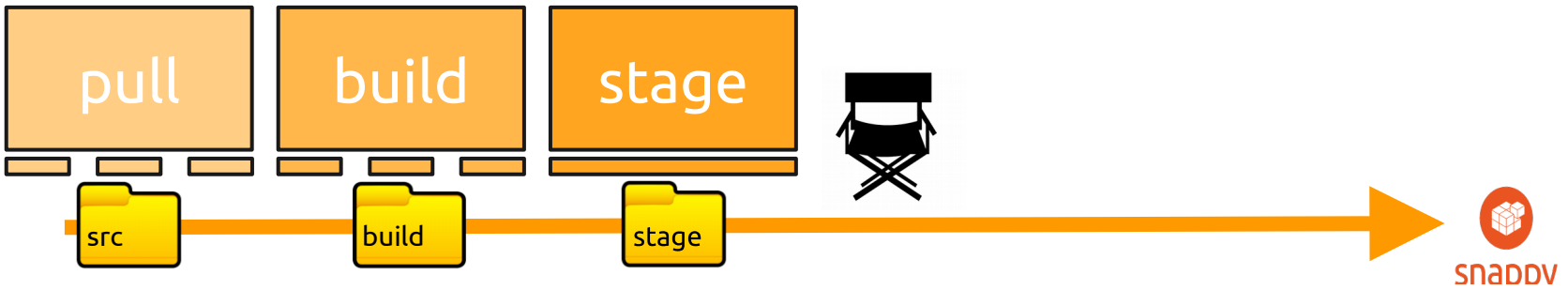
Snapcraft lifecycle

The build phase builds the parts of the downloaded code.

The next step is that each part is built in its parts/part-name/build directory and installs itself into parts/part-name/install.

Snapcraft lifecycle

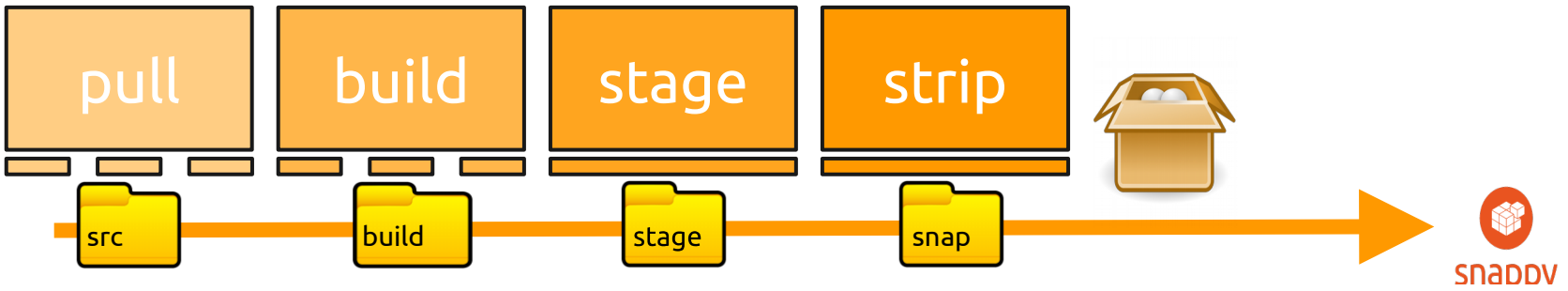
The stage phase copies the installed files into a user-visible stage/folder. All parts share the same file layout.



the parts are combined into a single directory tree that is called the "staging area". It can be found under the ./stage directory.

Snapcraft lifecycle

The strip phase copies the files in stage, minus any filtered/excluded files into the user-visible snap/folder. It also creates any additional package metadata.

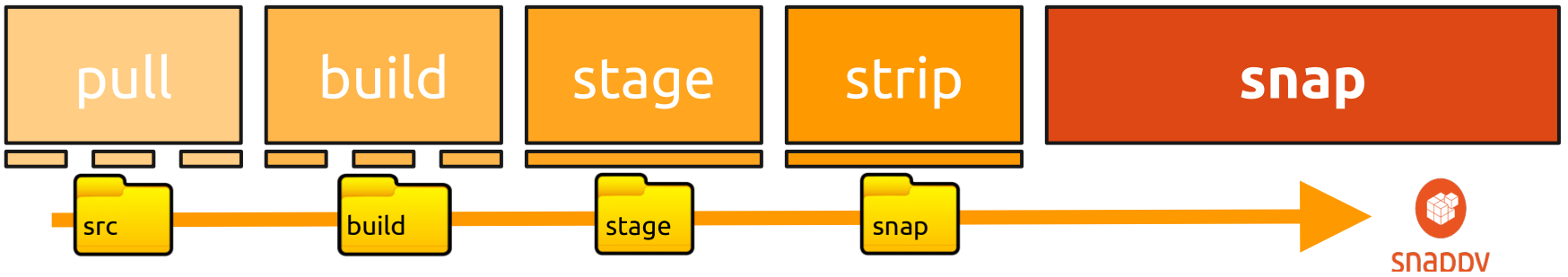


The snap step moves the data into a ./snap directory. It contains only the content that will be put into the final snap package (unlike the staging area which may include some development files).

This ./snap directory is useful for inspecting what is going into your snap and to make any final post-processing on snapcraft's output. The metadata info about the project will also now be placed in ./snap/meta.

Snapcraft lifecycle

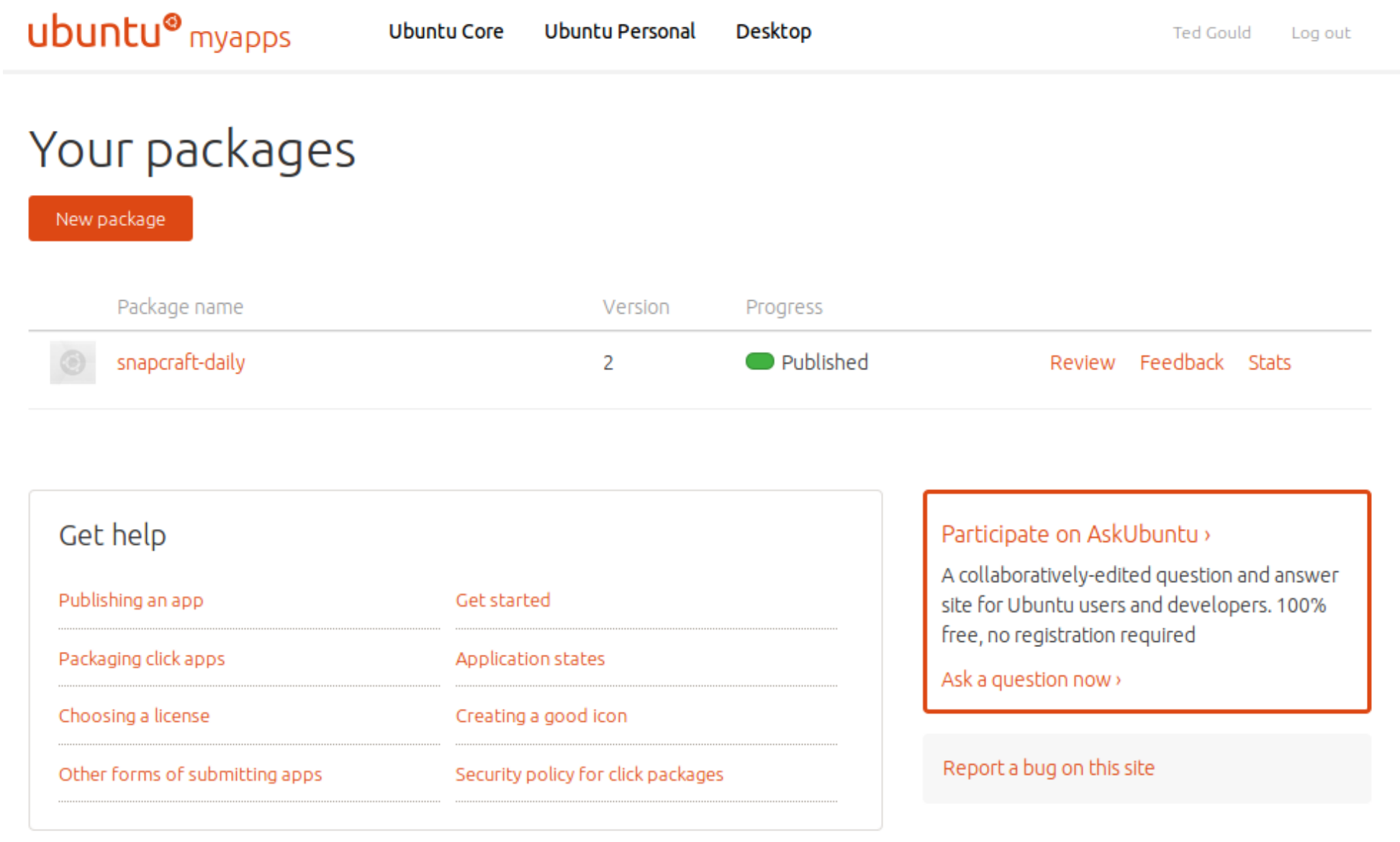
The snap phase wraps the needed files into a .snap file with the snap packaging format.



The final step builds a snap package out of the snap directory.

This .snap file can be uploaded to the Ubuntu store and published directly to snappy Ubuntu Core users.

Upload to Store



Further Reading

- http://developer.ubuntu.com — Information on all things developing for Ubuntu, from phone API docs to Snappy config file formats.
- http://myapps.developer.ubuntu.com — Store to upload apps to
- http://ubuntu.com/snappy — Information on Snappy, suitable for non-developers
- http://askubuntu.com/ — Stack Exchange to ask and eventually answer questions about Ubuntu, including Snappy and Snapcraft

Questions Please

Ted Gould
ted@canonical.com
@tedjgould