"It is wrong to suppose that if you can't measure it, you can't manage it - a costly myth"

W. Edwards Deeming

"If you can't measure it, you can't improve it"

Lord Kelvin

Technical Debt for Linux-based distributions: Estimating what you are missing

Linux Foundation Open Source Leadership Summit
Tahoe, CA (USA)
February 14th 2017

Jesus M. Gonzalez-Barahona (URJC & Bitergia)
Paul Sherwood (Codethink)
speakerdeck.com/bitergia

Outline

Some context

Why debt for distros

Approach

Current results

Next steps

Some context

/Jesus

My two hats:

Like five years ago I was having coffees with the gang of Bitergia founders

Involved in the company since then

bitergia.com

I work at Universidad Rey Juan Carlos...

...researching about software development

gsyc.es/~jgb

/Paul

Currently…

Codethink CEO
and shareholder

Consultant +
troubleshooter

Baserock contributor

Previously…

Teleca Founder

cmdline tools + VCS

Project Manager

"The Software
Commandments"

Why debt for distros

Context

(Paul's POV)

- Develop/integrate/test software
- Employ/fund others to do that too
- Offer teams to large customers

- Advise on business impacts of FOSS
- Recommend *using* FOSS
- See lots of projects *misusing* FOSS
  - EOL versions
  - Long local forks, not upstreamed

- Notice Year 1 practices hurt Year 2..Year 20
- Wonder why... maybe because
  - Year 1 metrics are obvious (developer costs vs delivery date)
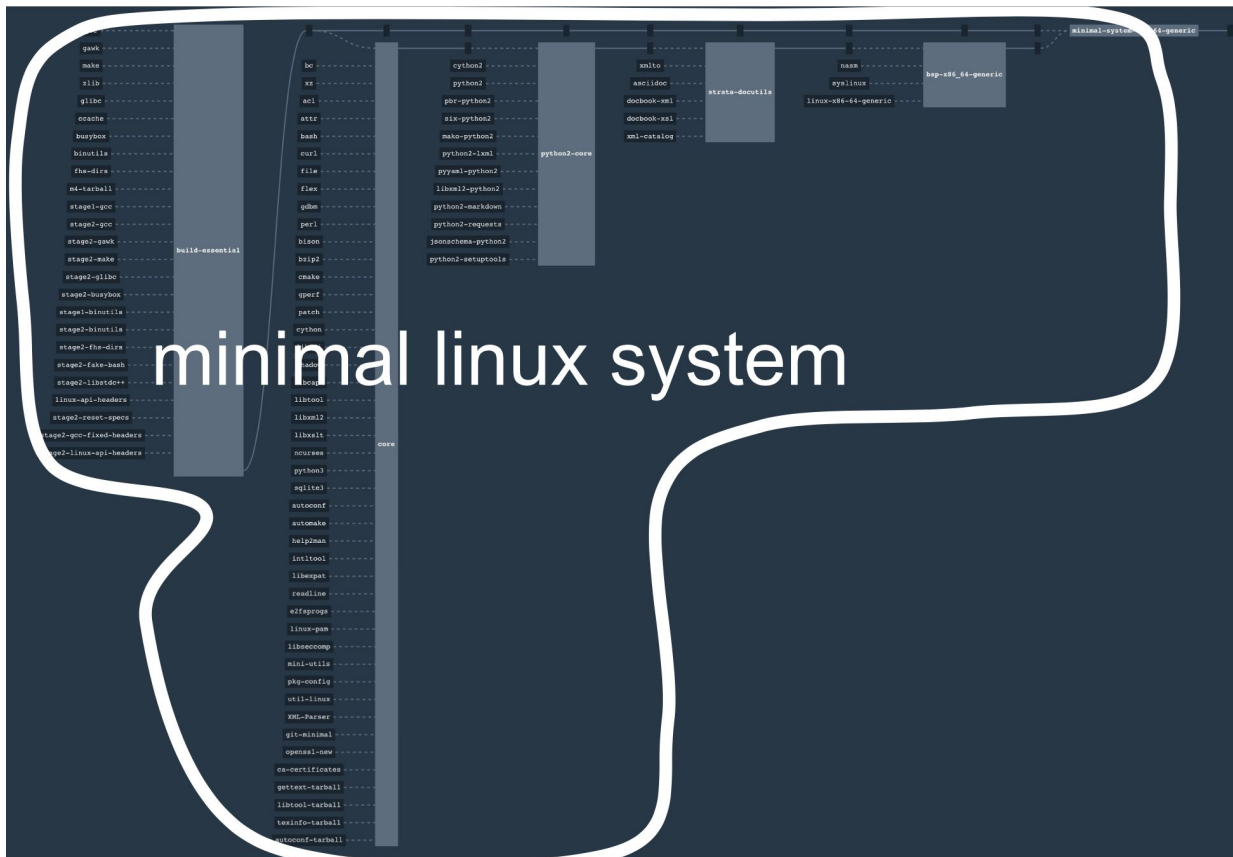 - Later metrics are a mystery...

Unanswered: when should we update?

Codethink

Bitergia

Unanswered: when should we update?
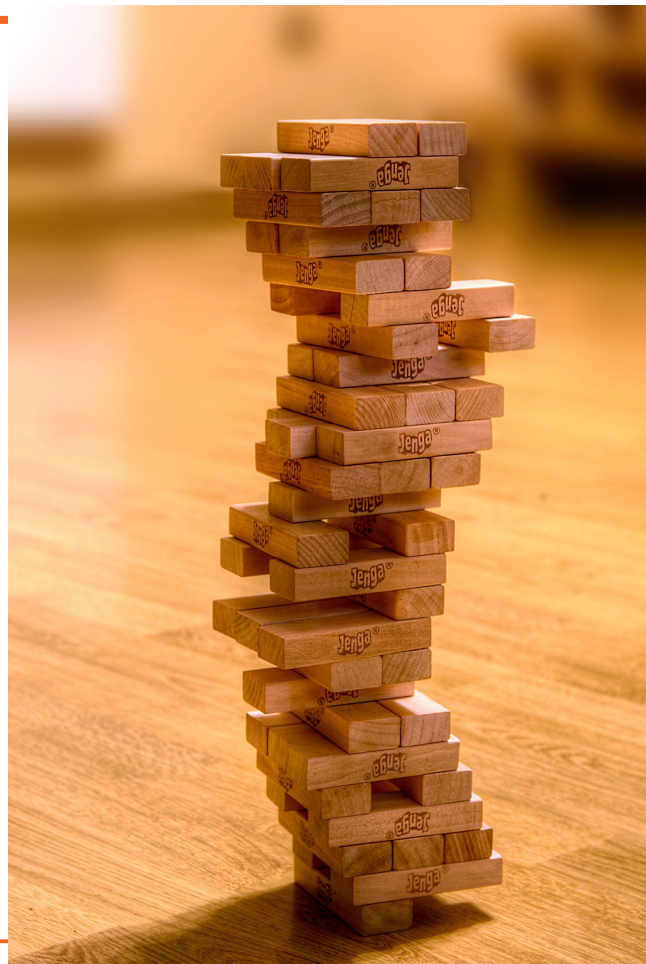
We're not talking about updating just a few components...

Typical IVI project approaching 1000...

Which ones do we need to upgrade?

How often do we need to re-decide?

minimal linux system

Production IVI System

pending
started
succeeded
failed
errored
aborted
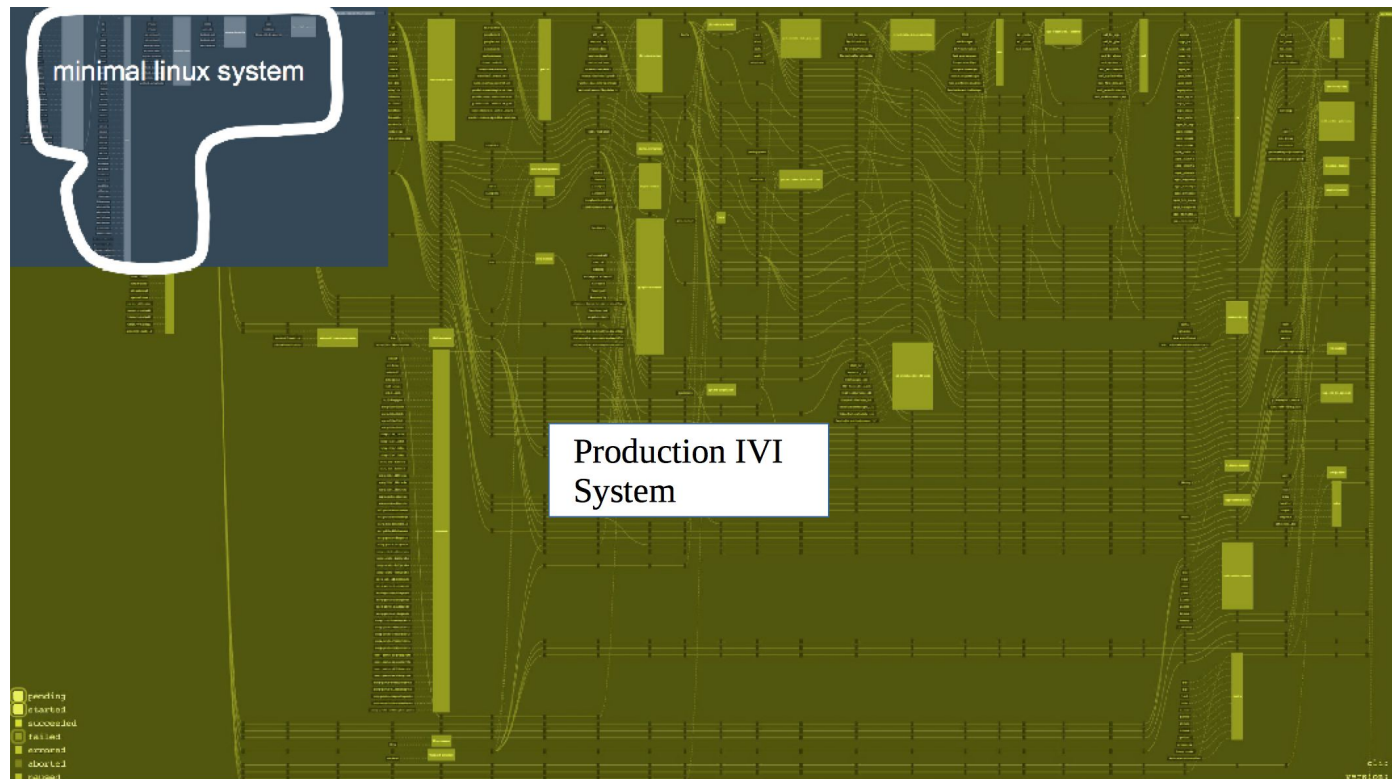
Example

- Project started on 3.8.x kernel in 2012
  - Plus custom drivers
- Went live three years later on same 3.8.x
  - Plus custom functionality
  - Plus thousands of fixes backported
- As the years go by
  - Developers move on - no-one understands the custom stuff
  - Cost of backporting increases
- New variants need new features (eg virtualization)
  - Cost of backporting from later kernels increases

Eventually one of the releases DEMANDS an update

Example continued

Insertions

Development

Maintenance
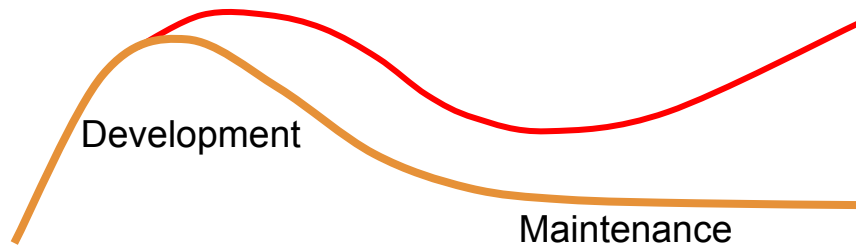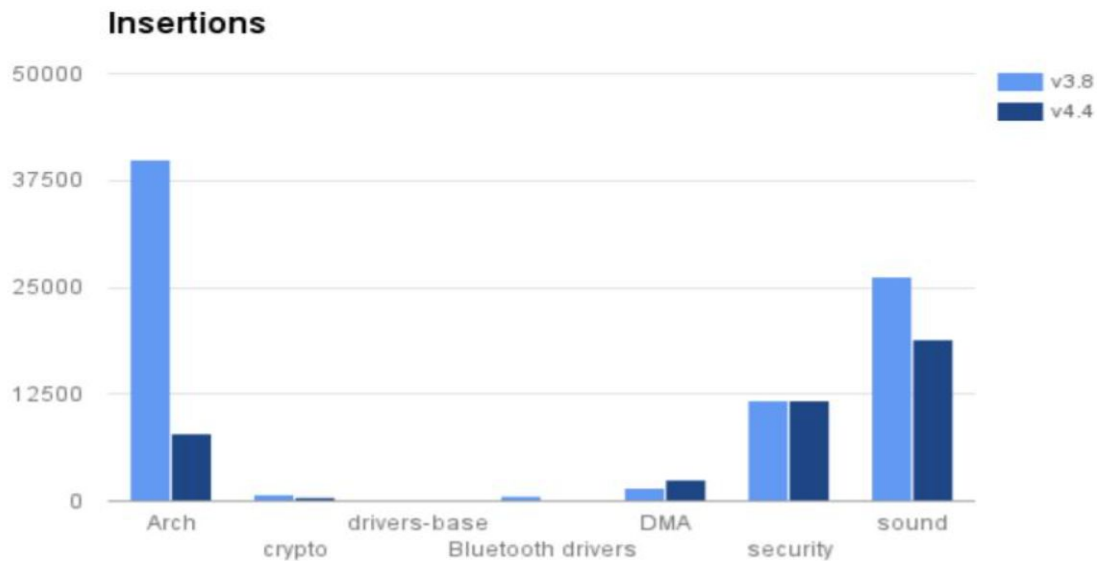
When to update

What you risk by upgrading

What you risk or lose by not upgrading

When to update

The balance may change suddenly over time

Rationale

- Technical debt is a popular concept
- … but not for third-party software
- … and not for FOSS

- Distros are large third-party software sets
- Distros update constantly
- Distro users often do not

- Cost of updating is perceived high
- Cost of not updating is unknown

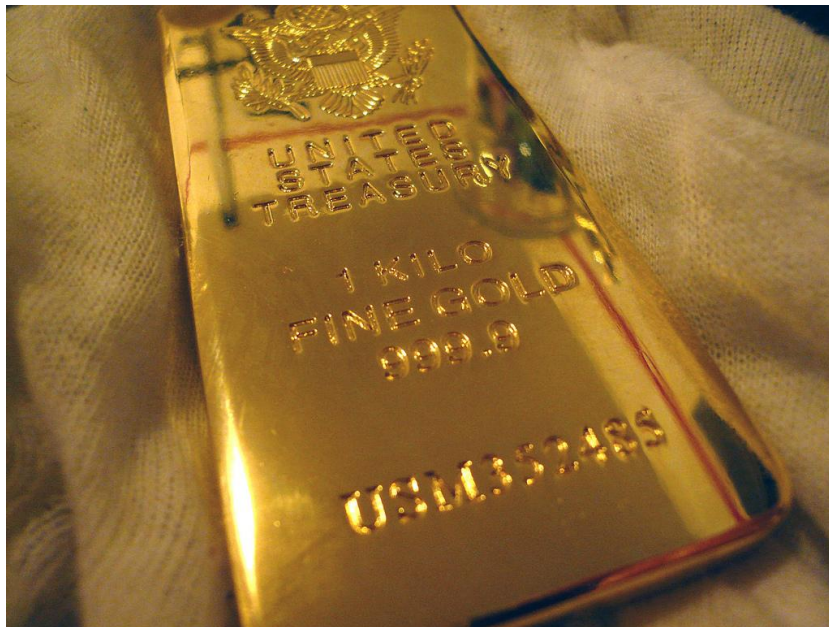Can we even **find** metrics for this?

Approach

What to measure?

- Delta vs mainline
- For individual components, and
- For whole stack:
  - distros
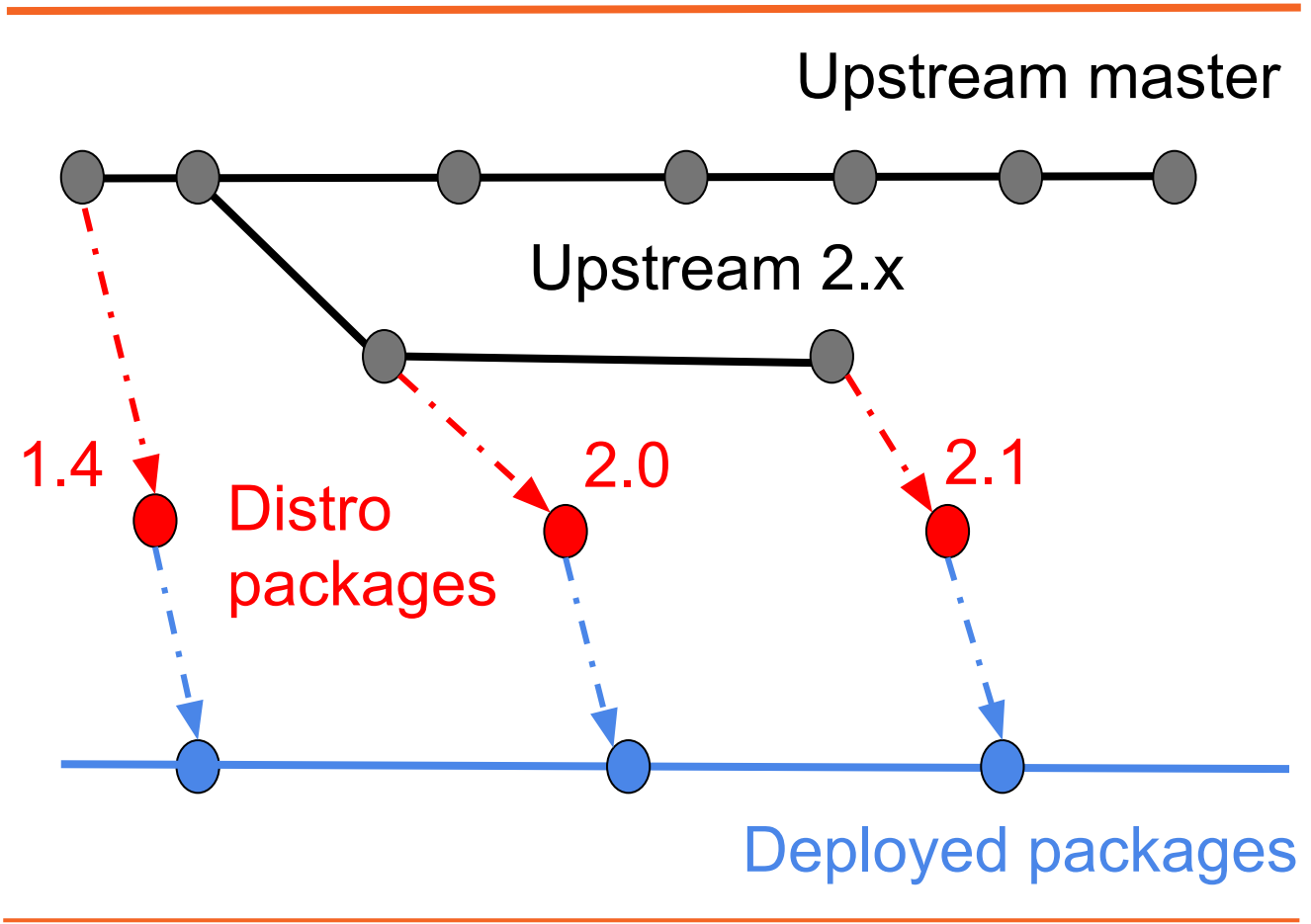 - custom assemblies/stacks

Defining "Gold standard"

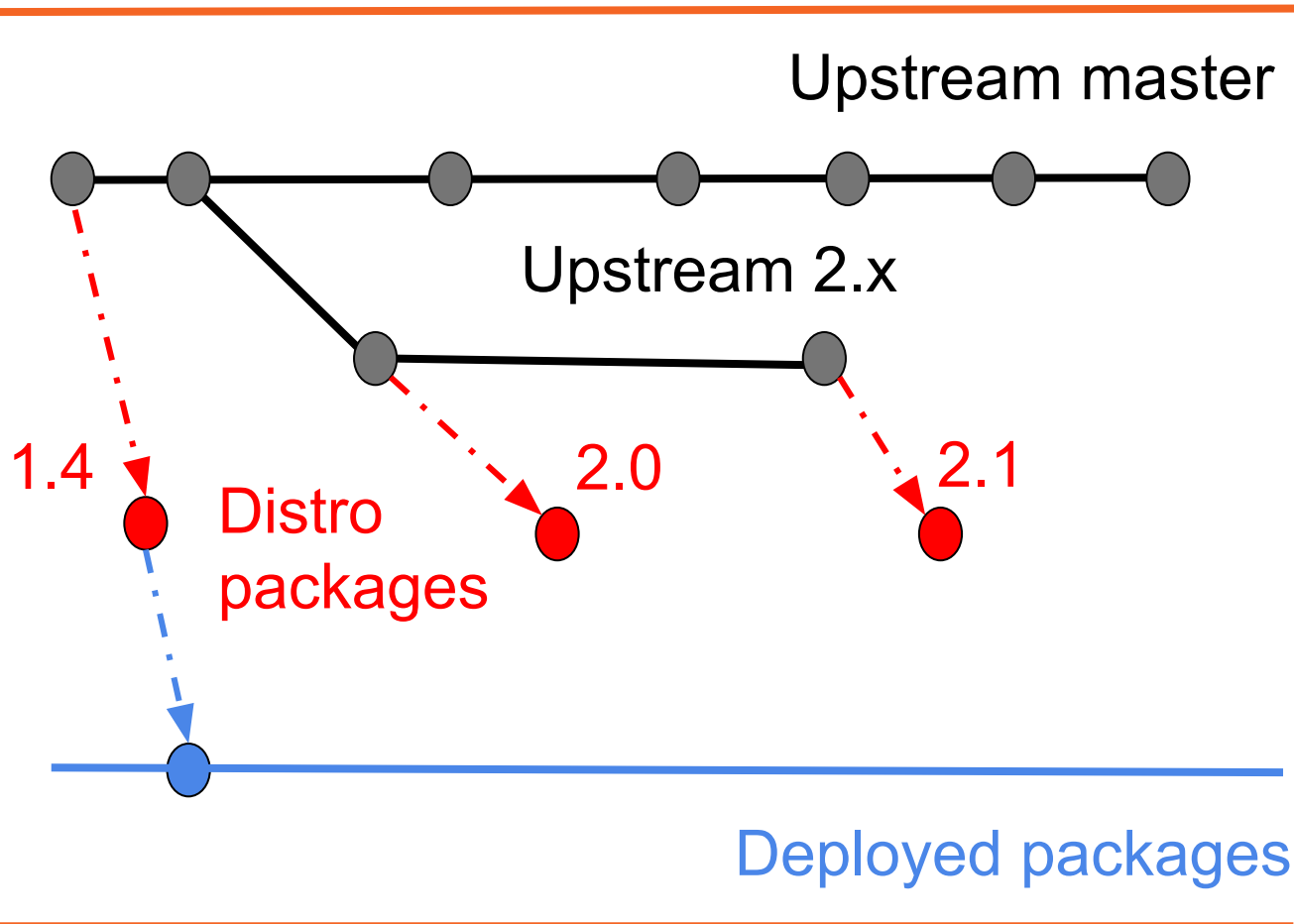# The different kinds of gold (examples)

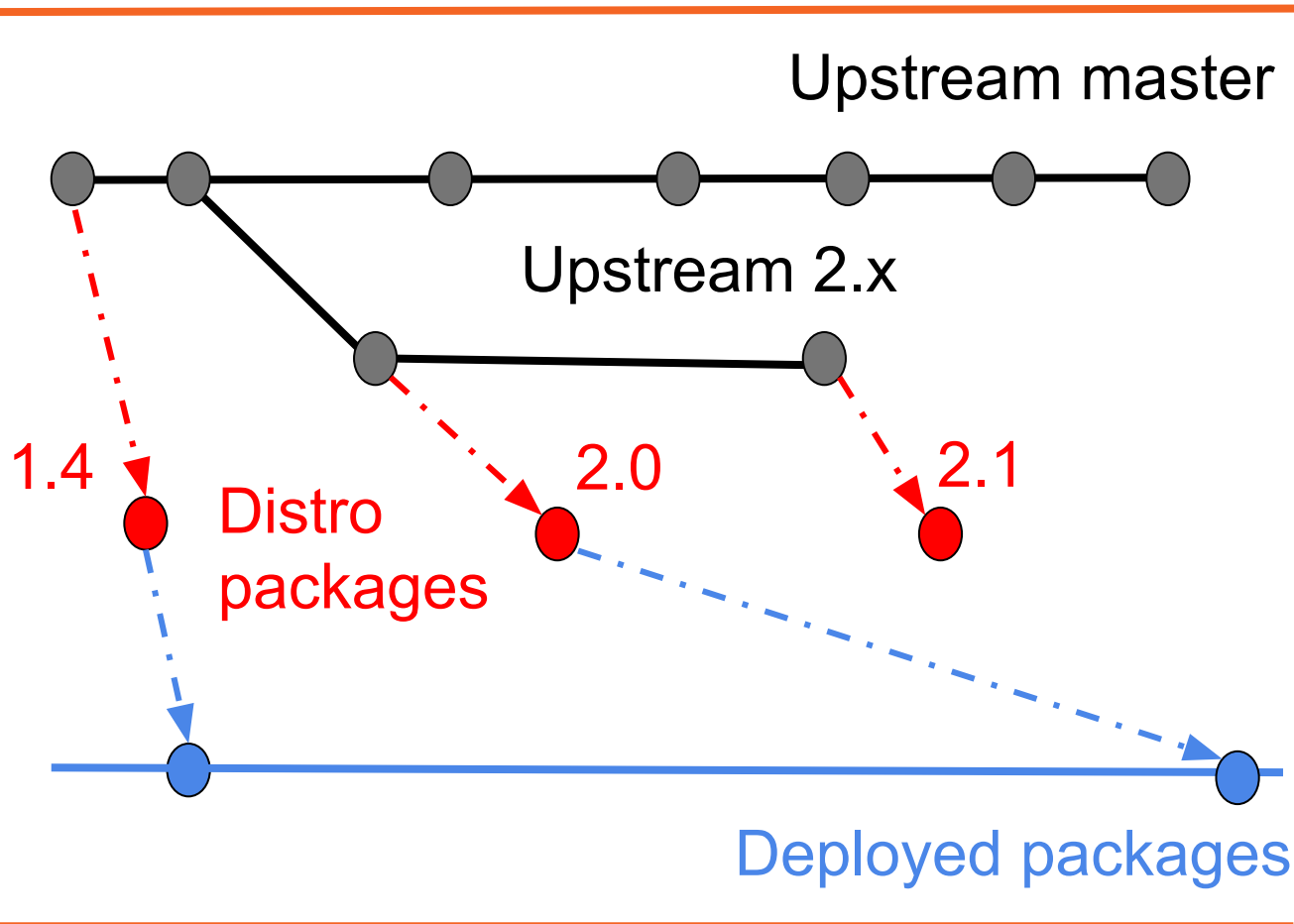| Goals	Scenarios	Candidates
Stability	Isolated system, frozen functionality	Debian stable
Functionality	Cloud application	Latest upstream
Security	Upgradable embedded	Stable upstream

Comparing with upstream

Upstream master

Upstream 2.x

1.4

2.0

2.1

Distro packages

Deployed packages

Codethink

Bitergia

Comparing with upstream (no updates)

Upstream master

Upstream 2.x

1.4

2.0

2.1

Distro packages

Deployed packages

Codethink

Bitergia

Comparing with upstream (late updates)

Upstream master

Upstream 2.x

1.4

2.0

2.1

Distro packages

Deployed packages

Comparing with upstream (new package)
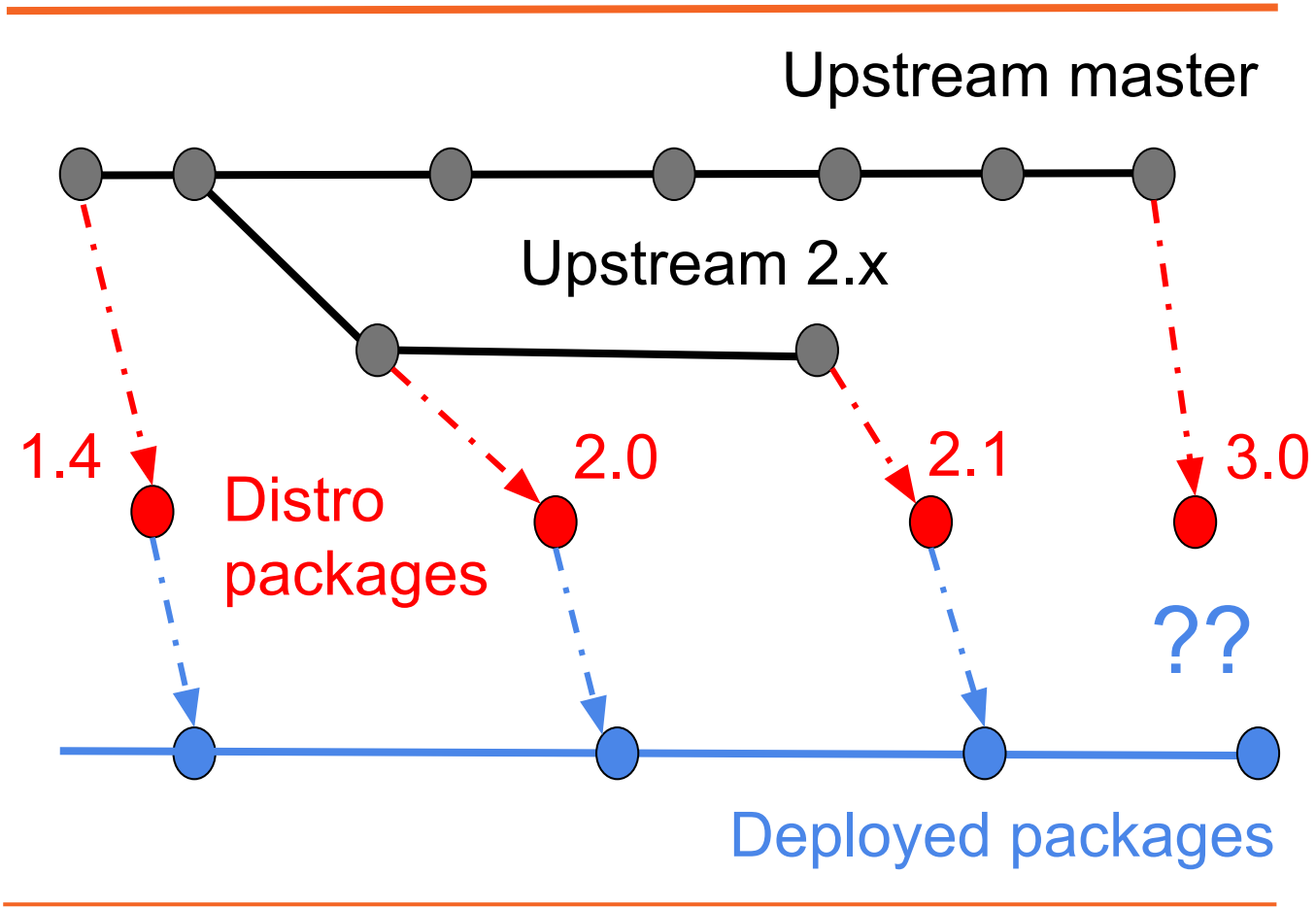
Upstream master

Upstream 2.x

1.4   Distro packages   2.0   2.1   3.0

??

Deployed packages

Compare "most likely upstream equivalent"

1.4   2.0   2.1   3.0

??
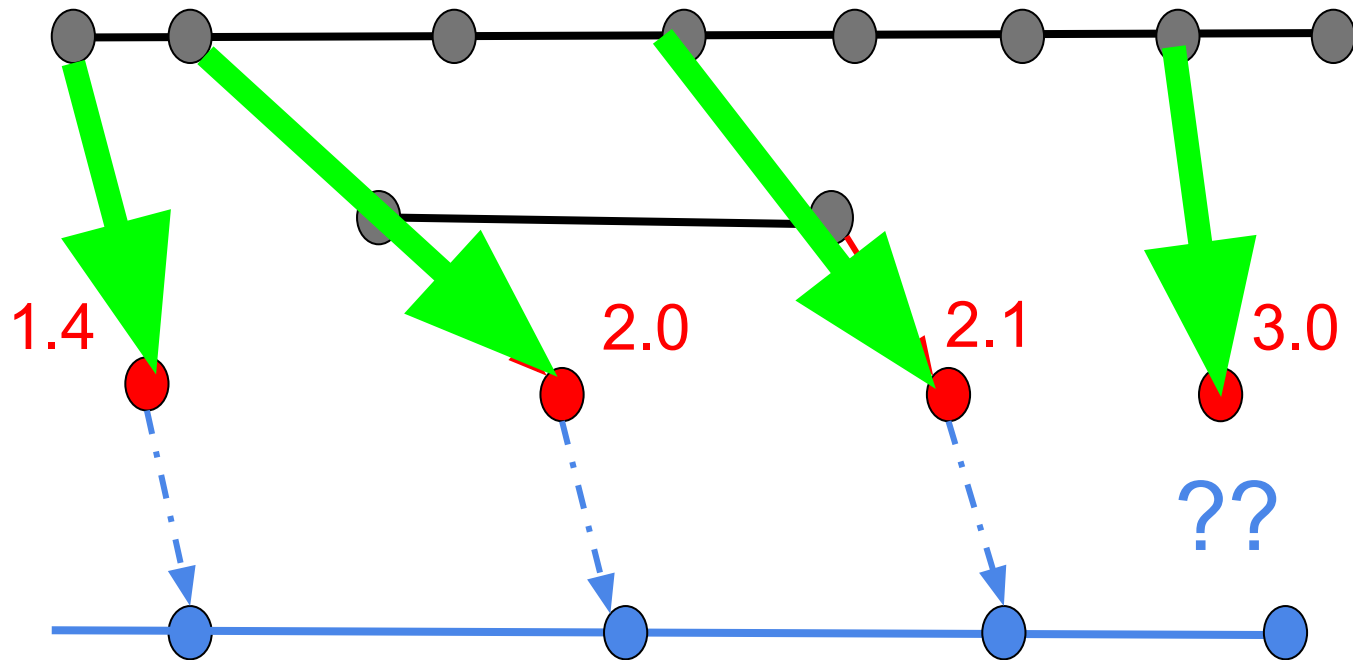
Compare "most likely upstream equivalent" with HEAD
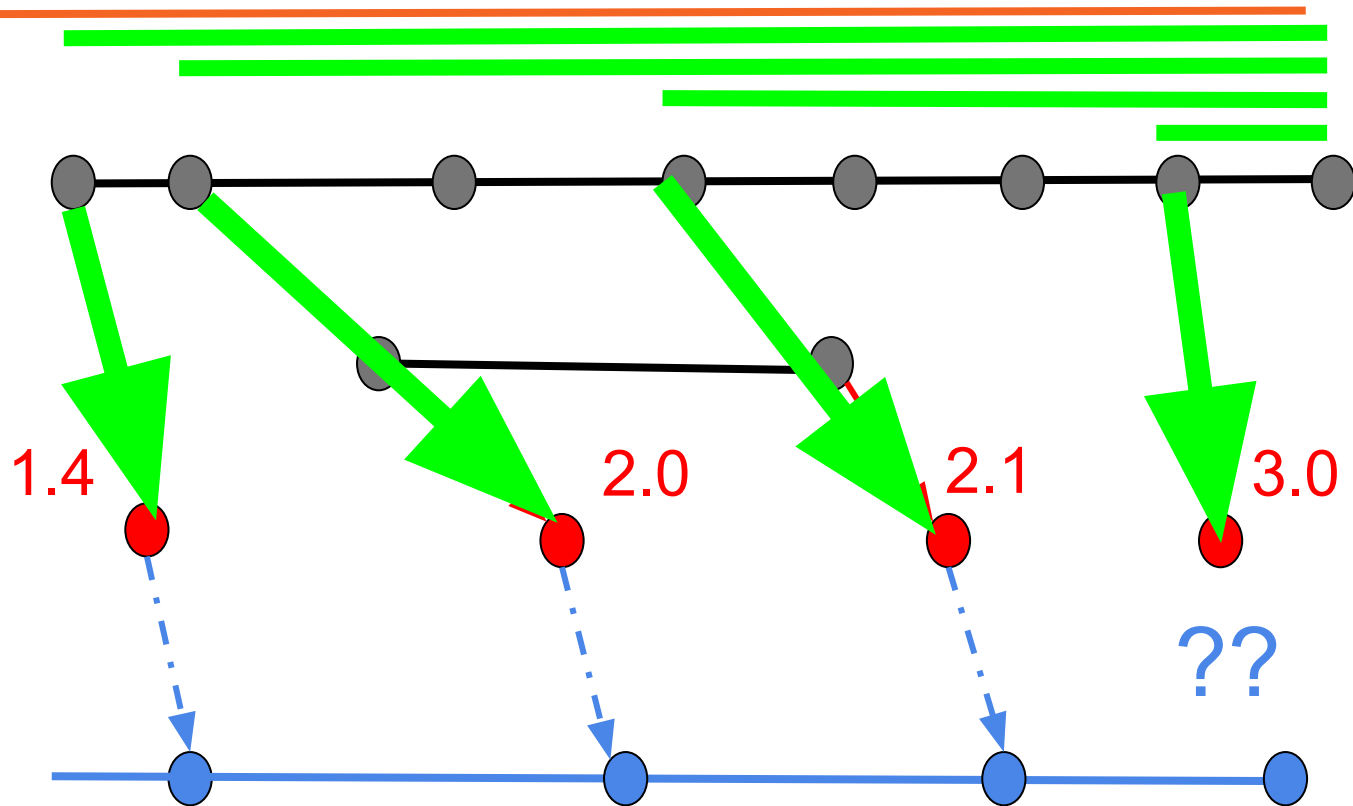
Difference is "technical lag" with "gold standard"

1.4   2.0   2.1   3.0

??
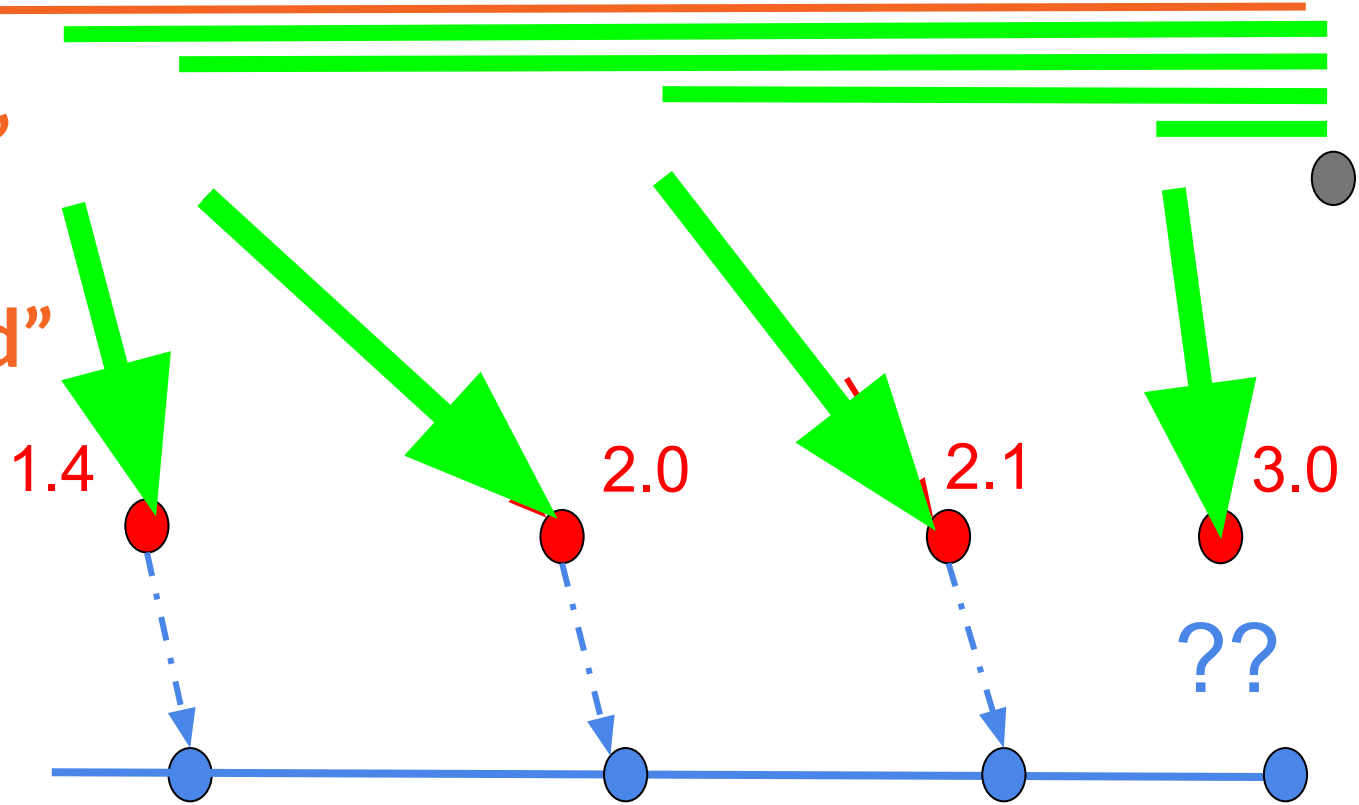
How to measure difference

1.4 2.0   2.1   3.0
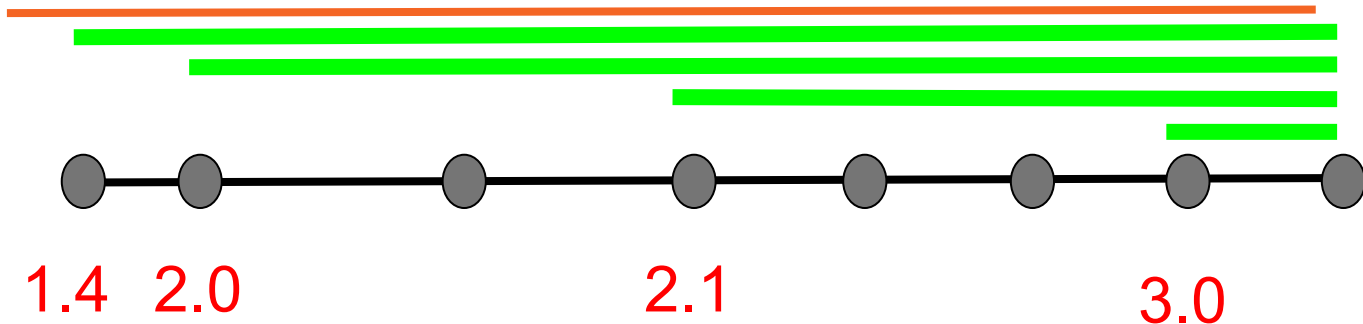
Lines of code
Number of functions, classes
Number of bugs fixed
Number of security bugs fixed
Number of issues closed
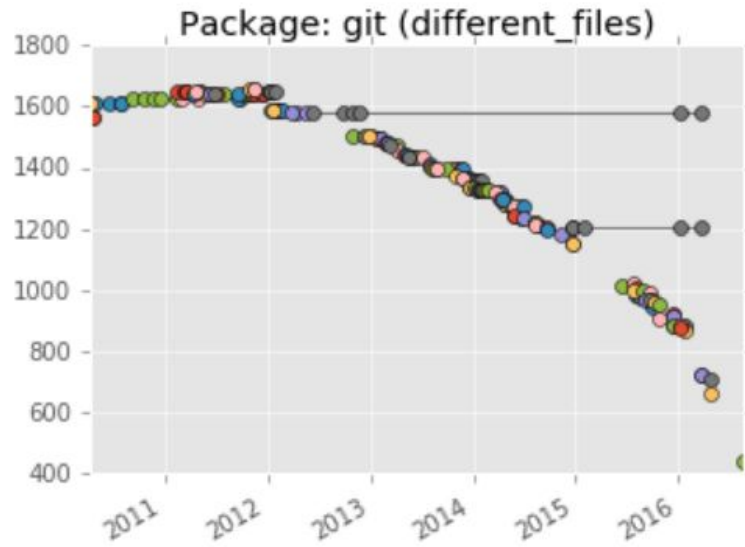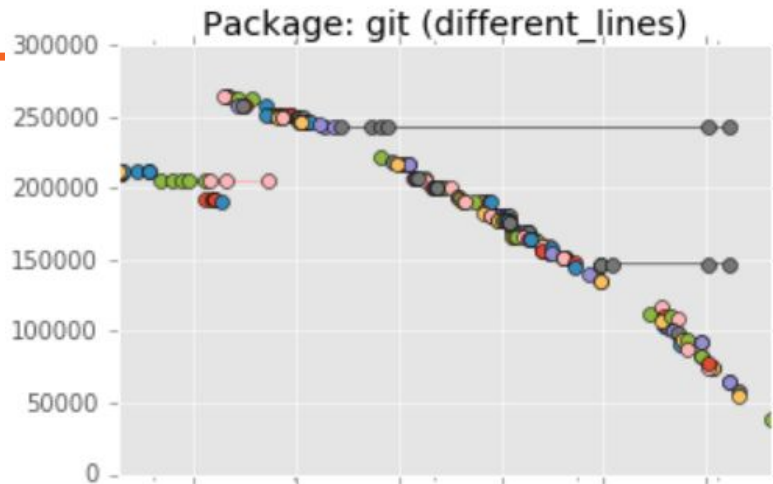Time for benchmark runs
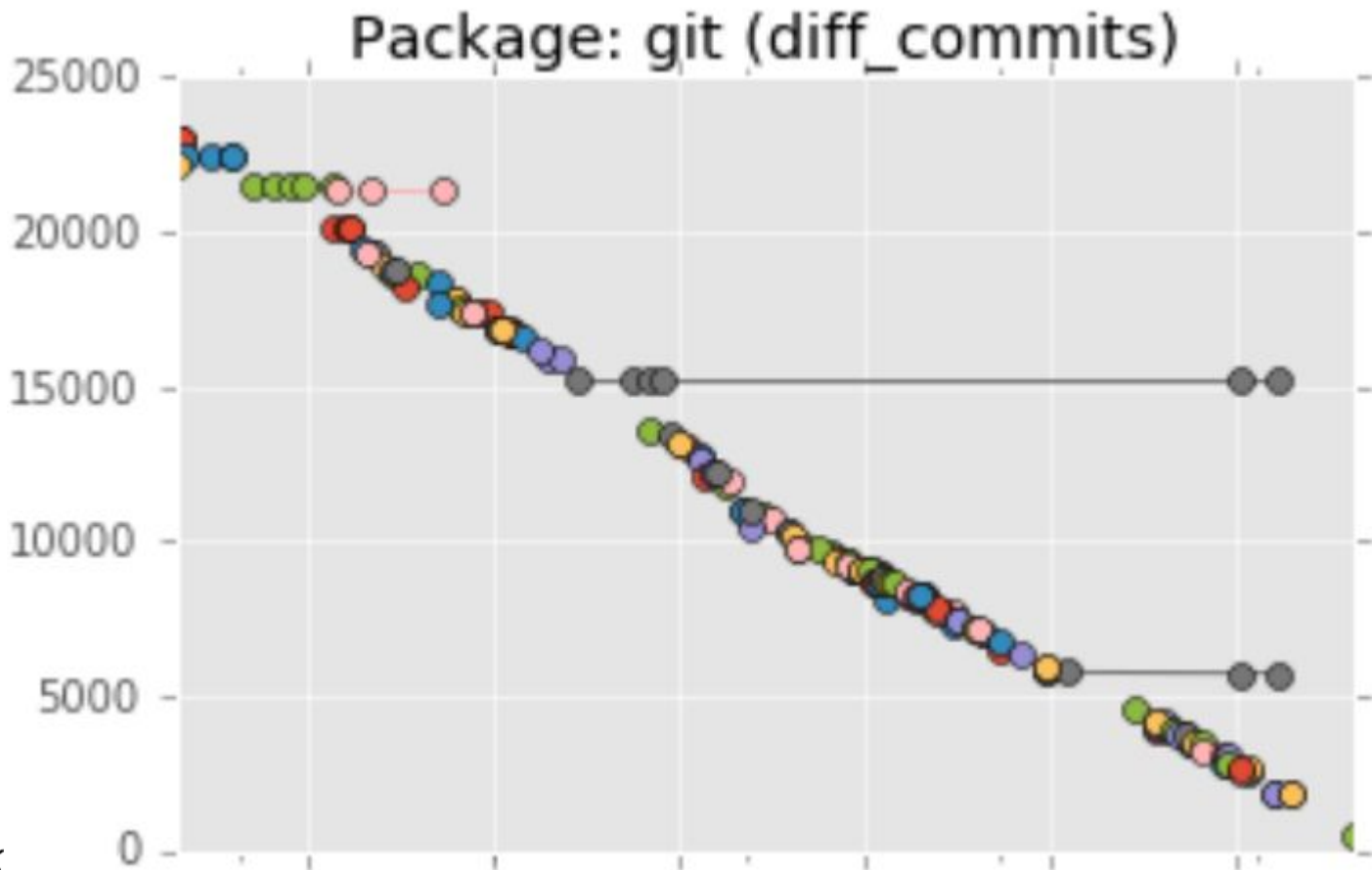Unit test coverage
Results in integration tests
...

Current results

Debian Git releases, lag in November (lines, files)

Debian Git releases, lag in Nov. (commits)

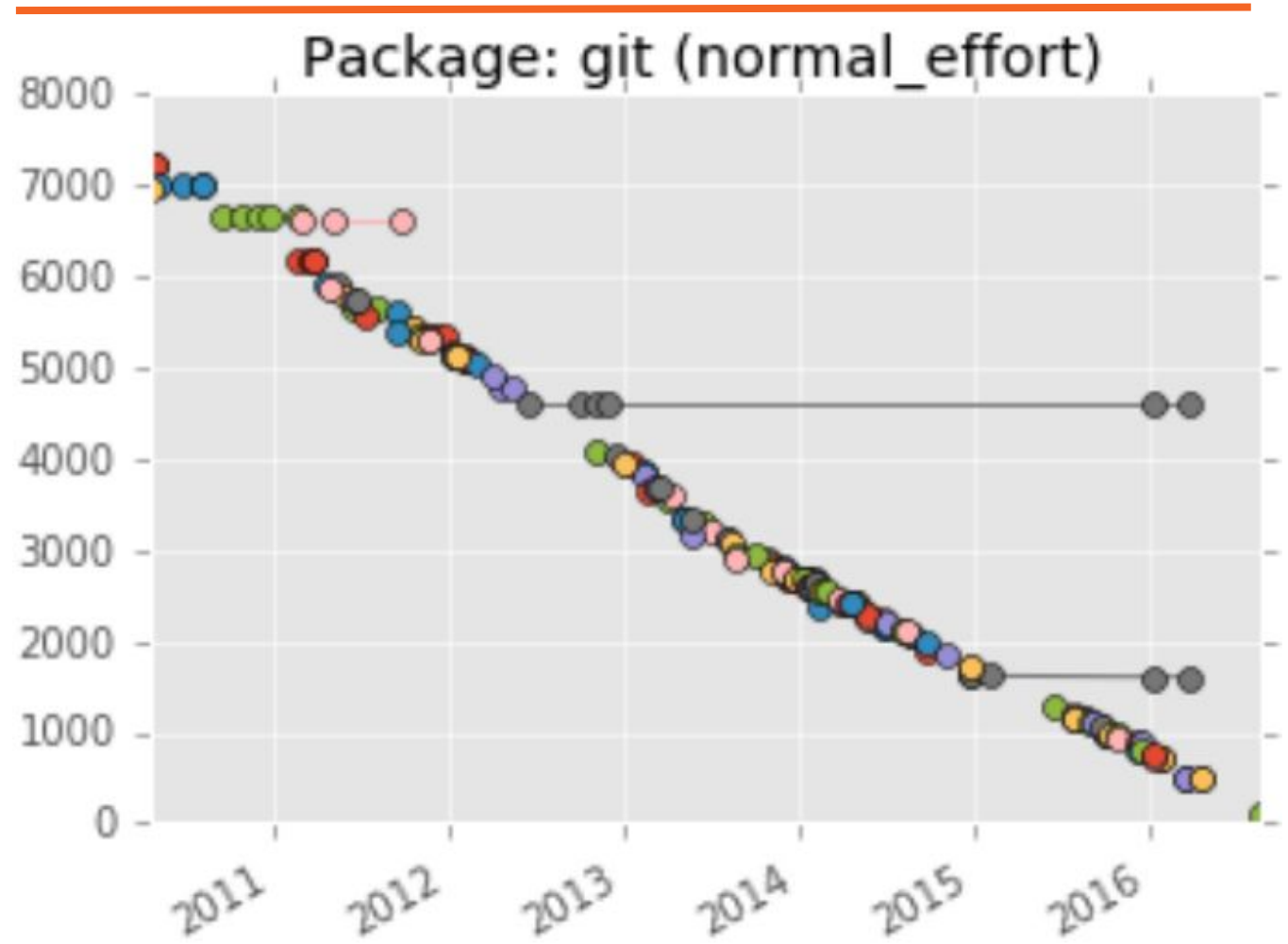Package: git (diff_commits)

Normalized effort (in days)

For each developer:
number of days with at least one commit

For a project:
sum for all developers

Debian Git releases, lag in Nov. (normalized effort)

Package: git (normal_effort)

Next steps

Application to many domains

Debian packages in a virtual machine

Python pip packages in a deployed container

JavaScript npm modules in a web app

Yocto packages in an embedded system

Definition of details, according to requirements

Different "golden standards"

Different metrics for lag

Different aggregations

Software for automated computation of lag per component (and dependencies?)

Credits

Images

"Gold", by Michael Mandiberg
CC Attribution-ShareAlike 2.0
https://flic.kr/p/6feTT2

https://commons.wikimedia.org/wiki/File:Jenga_distorted.jpg

"Gold philarmonic", by Eric Golub
CC Attribution-ShareAlike 2.0
https://flic.kr/p/7csHXG

https://commons.wikimedia.org/wiki/File:Jenga_distorted.jpg

"Plymouth", by Dennis Jarvis
CC Attribution-ShareAlike 2.0
https://flic.kr/p/5pqT72

"Balance scale", by winnifredxoxo at Flickr