TELEMETRY ONBOARDING

Roberto A. Vitillo, Mark Reid

1. What is Unified Telemetry?
2. Executive dashboards
3. Telemetry dashboards
4. Adding a probe
5. What is a ping
6. Data pipeline
7. Experiments
8. Offline processing
9. Stats
10. Privacy Policy
11. Data Collection Policy

WHAT IS UNIFIED TELEMETRY?

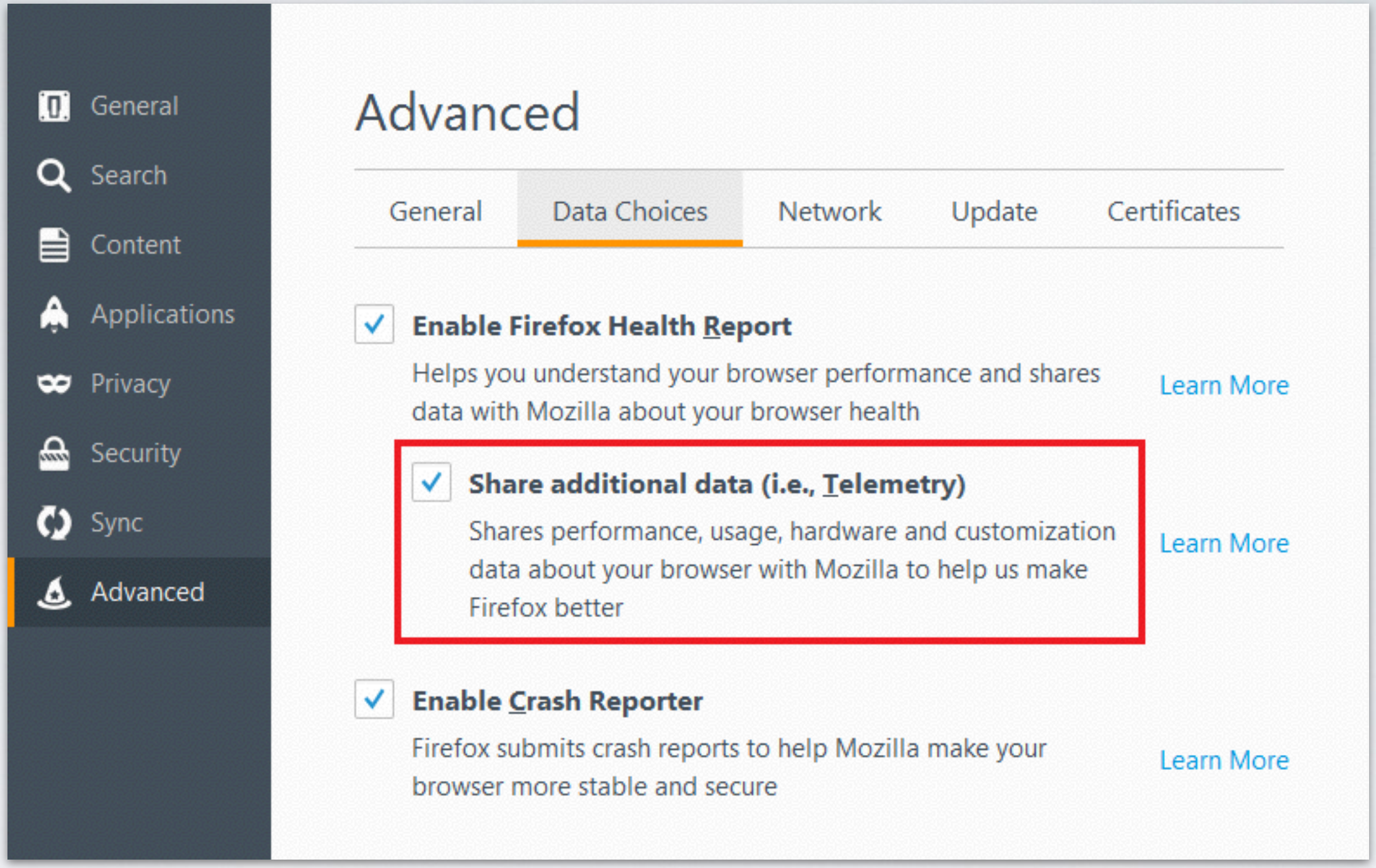
- Measures how Firefox behaves in the real world [1].
- Collects non-personal information about performance, hardware, usage, errors, and customizations
- The data, used in aggregate,
  - allows to identify new issues and regressions
  - allows to conduct longitudinal studies and experiments

[1] https://wiki.mozilla.org/Unified_Telemetry

UNIFIED?

- Classic Telemetry [1], designed to send performance and usage data
  - enabled by default on pre-release channels and disabled by default on release
- Firefox Health Report [2] is a data source intended to be representative and support longitudinal study.
  - enabled by default on release and pre-release

[1] https://wiki.mozilla.org/Telemetry
[2] https://wiki.mozilla.org/Firefox_Health_Report



Add the following definitions to your mozconfig file:

```
export MOZILLA_OFFICIAL=1
export MOZ_TELEMETRY_REPORTING=1
```

Telemetry Data

about:telemetry

Search

Telemetry Data

This page shows the information about performance, hardware, usage and customisations collected by Telemetry. This information is submitted to Mozilla to help improve Mozilla Firefox.

FHR data upload is enabled. Change
Extended Telemetry recording is enabled. Change

Ping data source:
Current ping data
Archived ping data

Show subsession data

General Data *Click to toggle section*

Environment Data *Click to toggle section*

Telemetry Log *Click to toggle section*

Slow SQL Statements *(No data collected)*

Browser Hangs *Click to toggle section*

Thread Hangs *(No data collected)*

Histograms *Click to toggle section*

Keyed Histograms *Click to toggle section*

Simple Measurements *Click to toggle section*

Late Writes *Click to toggle section*

System Information *Click to toggle section*

Add-on Details *Click to toggle section*

Histograms Collected by Add-ons *(No data collected)*

REFERENCES

- https://wiki.mozilla.org/Unified_Telemetry
- https://wiki.mozilla.org/Telemetry
- https://wiki.mozilla.org/Firefox_Health_Report

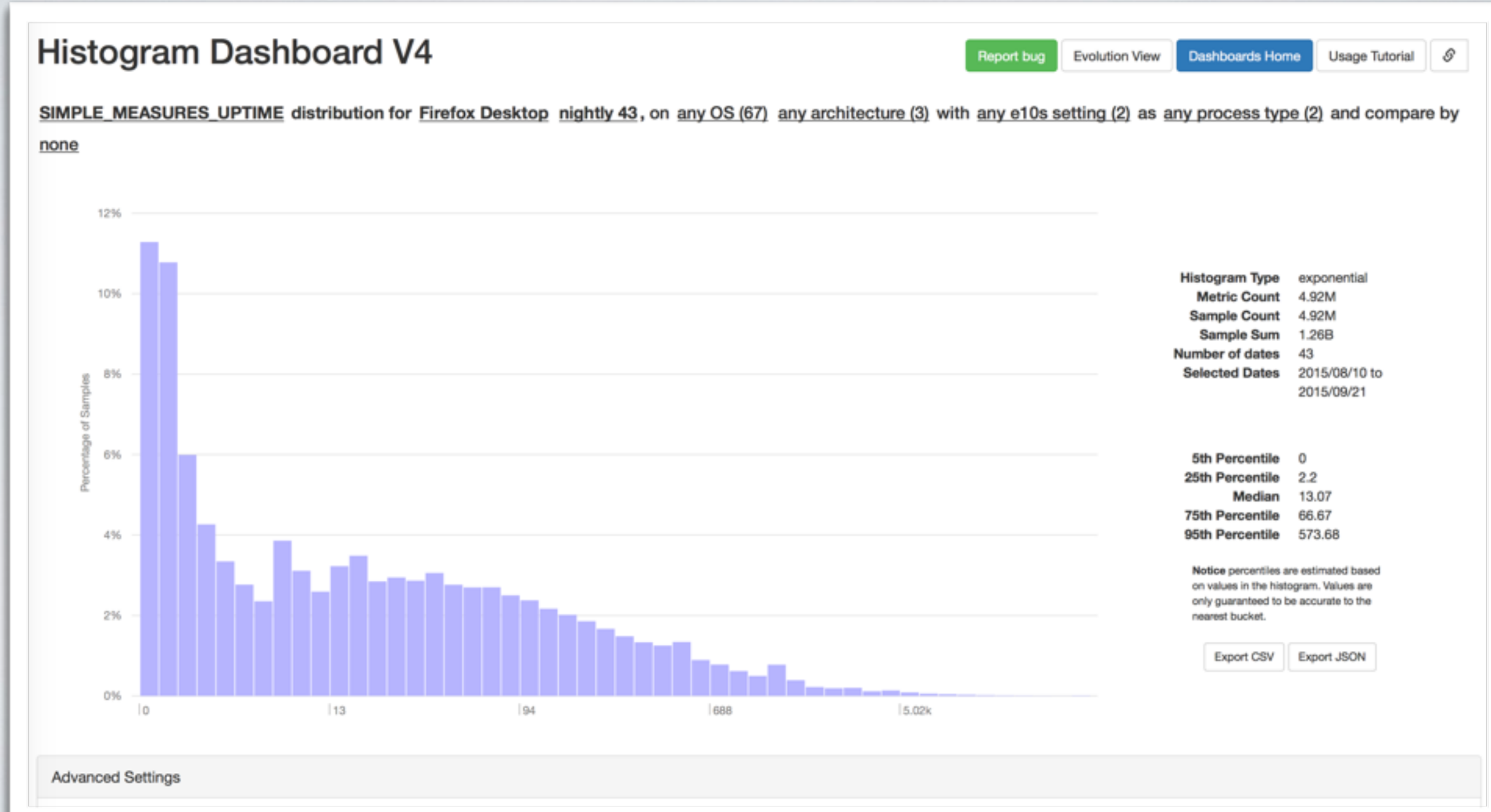
EXECUTIVE DASHBOARDS

CONFIDENTIAL — Only for staff and contributors under NDA — Do not share

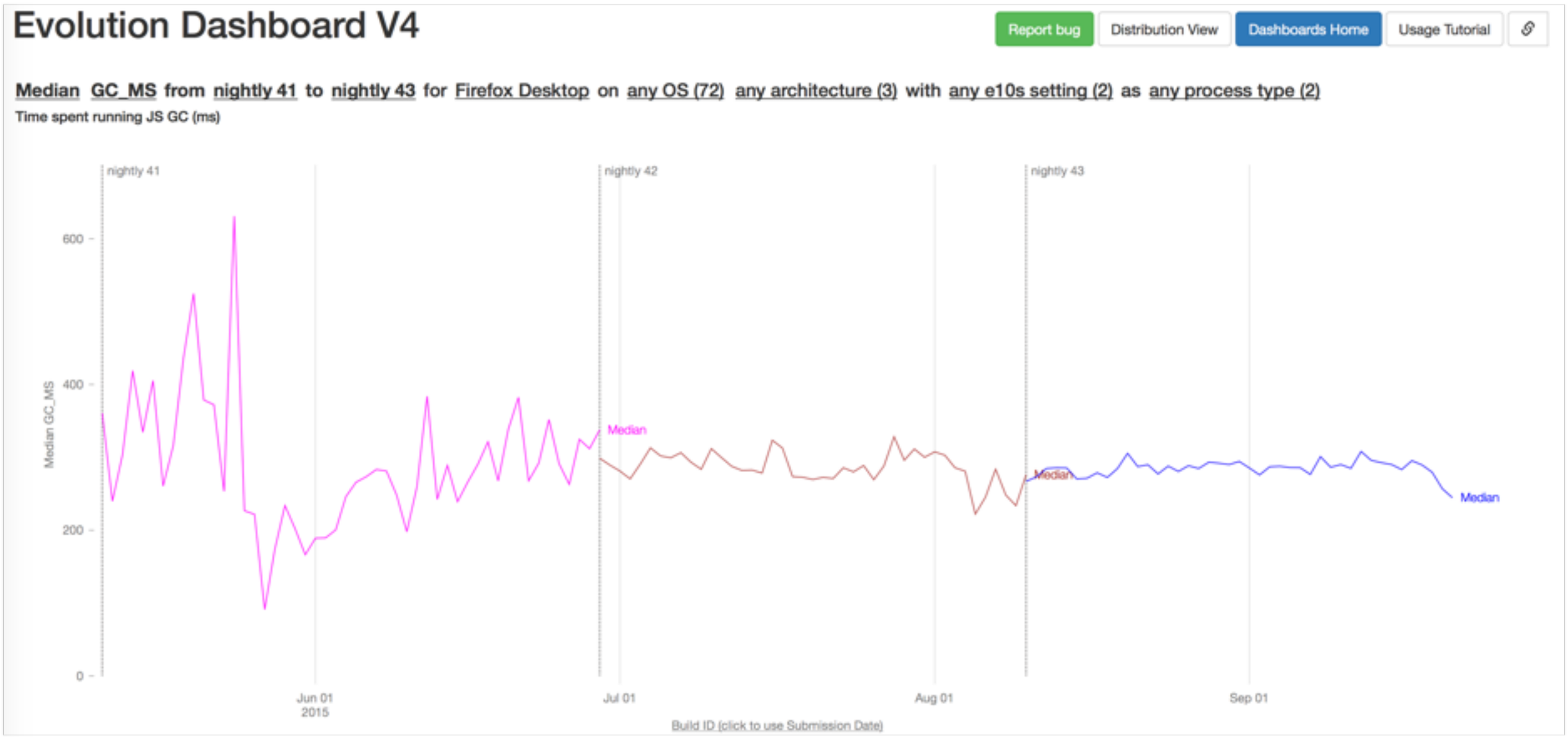
Mozilla Summary Dashboards

https://metrics.services.mozilla.com/

TELEMETRY DASHBOARDS



https://telemetry.mozilla.org/new-pipeline/dist.html



https://telemetry.mozilla.org/new-pipeline/evo.html

Aggregates can be accessed through a javascript library…

Examples

Getting versions in a range:

```
Telemetry.init(function() {
    var versions = Telemetry.getVersions("nightly/40", "nightly/42");
    console.log("Versions between nightly 40 to nightly 42 (inclusive):\n" + versions.join("\n"));
});
```

Getting a list of measures:

```
Telemetry.init(function() {
    Telemetry.getFilterOptions("nightly", "42", function(filterOptions) {
        console.log("Available measures:\n" + filterOptions.metric.join("\n"));
    });
});
```

Getting the dates for which there are submissions for GC_MS on nightly 42 on Windows:

```
Telemetry.init(function() {
    Telemetry.getEvolution("nightly", "42", "GC_MS", {os: "Windows_NT"}, true, function(evolutionMap) {
        console.log("Available dates:\n" + evolutionMap[""].dates().join("\n"));
    });
});
```

https://github.com/mozilla/telemetry-dashboard/blob/master/v2/doc.md

…or directly using the HTTP endpoints

API

Aggregates are made available through a HTTP API. There are two kinds of aggregates: per submission date and per build-id. To access the aggregates use the `aggregates_by/build_id/` and `aggregates_by/submission_date/` prefix respectively.

The following examples are based on build-id aggregates.

Get available channels:

```
curl -X GET http://SERVICE/aggregates_by/build_id/channels/
["nightly","beta","aurora"]
```

Get a list of options for the available dimensions on a given channel and version:

```
curl -X GET "http://SERVICE/filters/?channel=nightly&version=42"
{"metric":["A11Y_CONSUMERS","A11Y_IATABLE_USAGE_FLAG",...],
 "application":["Fennec","Firefox"],
 ...}
```

Get a list of available build-ids for a given channel:

```
curl -X GET "http://SERVICE/aggregates_by/build_id/channels/nightly/dates/"
[{"date":"20150630","version":"42"}, {"date":"20150629","version":"42"}]
```

https://github.com/vitillo/python_mozaggregator/blob/master/README.md

For example: https://aggregates.telemetry.mozilla.org/aggregates_by/submission_date/channels/nightly/?version=42&dates=20150709&metric=GC_MS&application=Firefox&child=false

Telemetry alerts

Login

Histogram Regression Detector

Telemetry Main-thread IO

Unpacked Add-on File Scan

Filter Metrics By:

Select Metric:

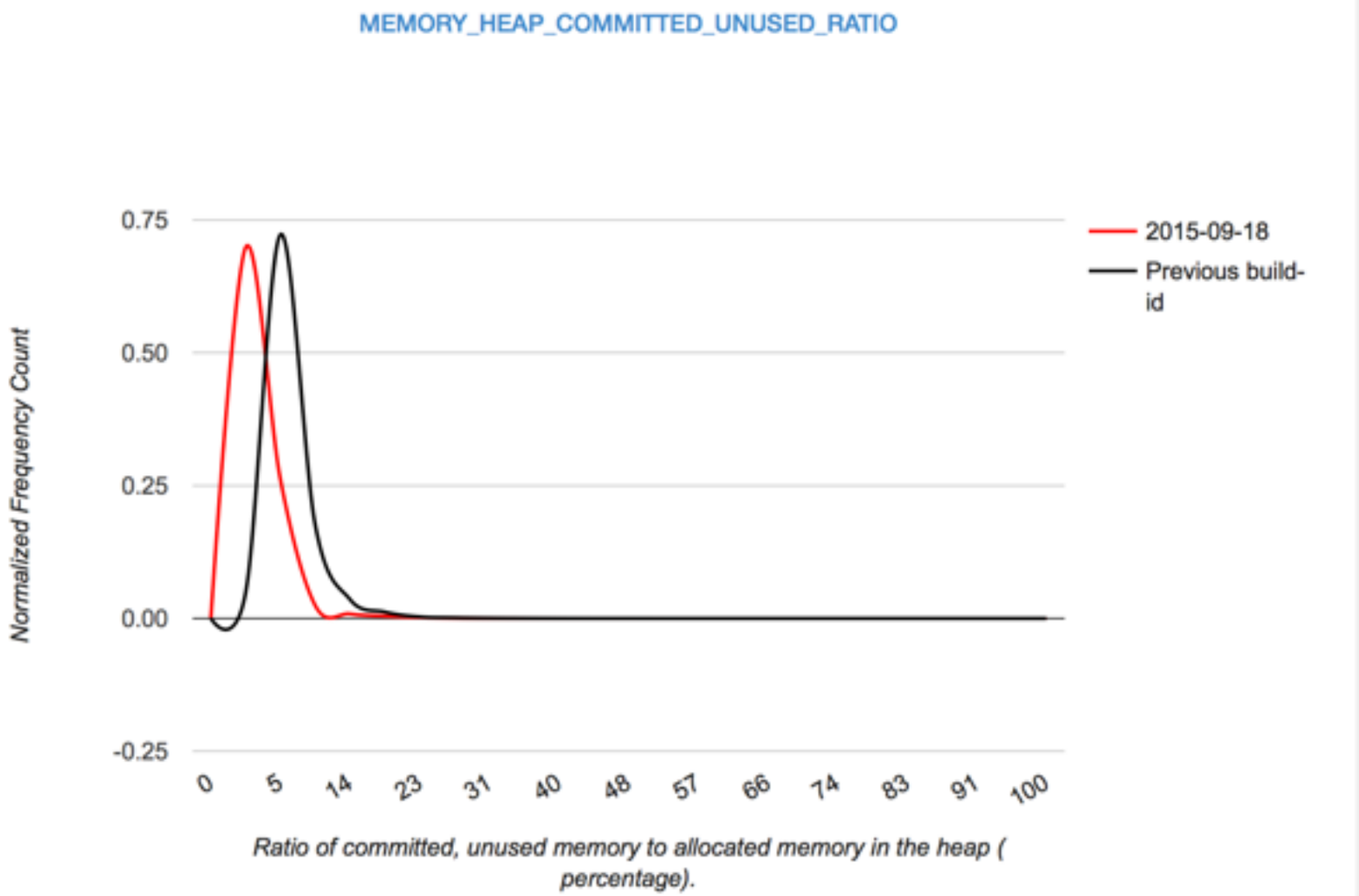
From:

2015-08-22

To:

2015-09-22

Failure to load alerts, reason: Not Found



http://alerts.telemetry.mozilla.org/index.html

Alert details: http://alerts.telemetry.mozilla.org/index.html#/detectors/1/metrics/677/alerts/?from=2015-09-18&to=2015-09-18

Changeset for 20150918030202: https://hg.mozilla.org/mozilla-central/pushloghtml?fromchange=e7d613b3bcfe1e865378bfac37de64560d1234ec&tochange=11dc79e232110ba6de5179e46dfbda77b52a88c3

REFERENCES

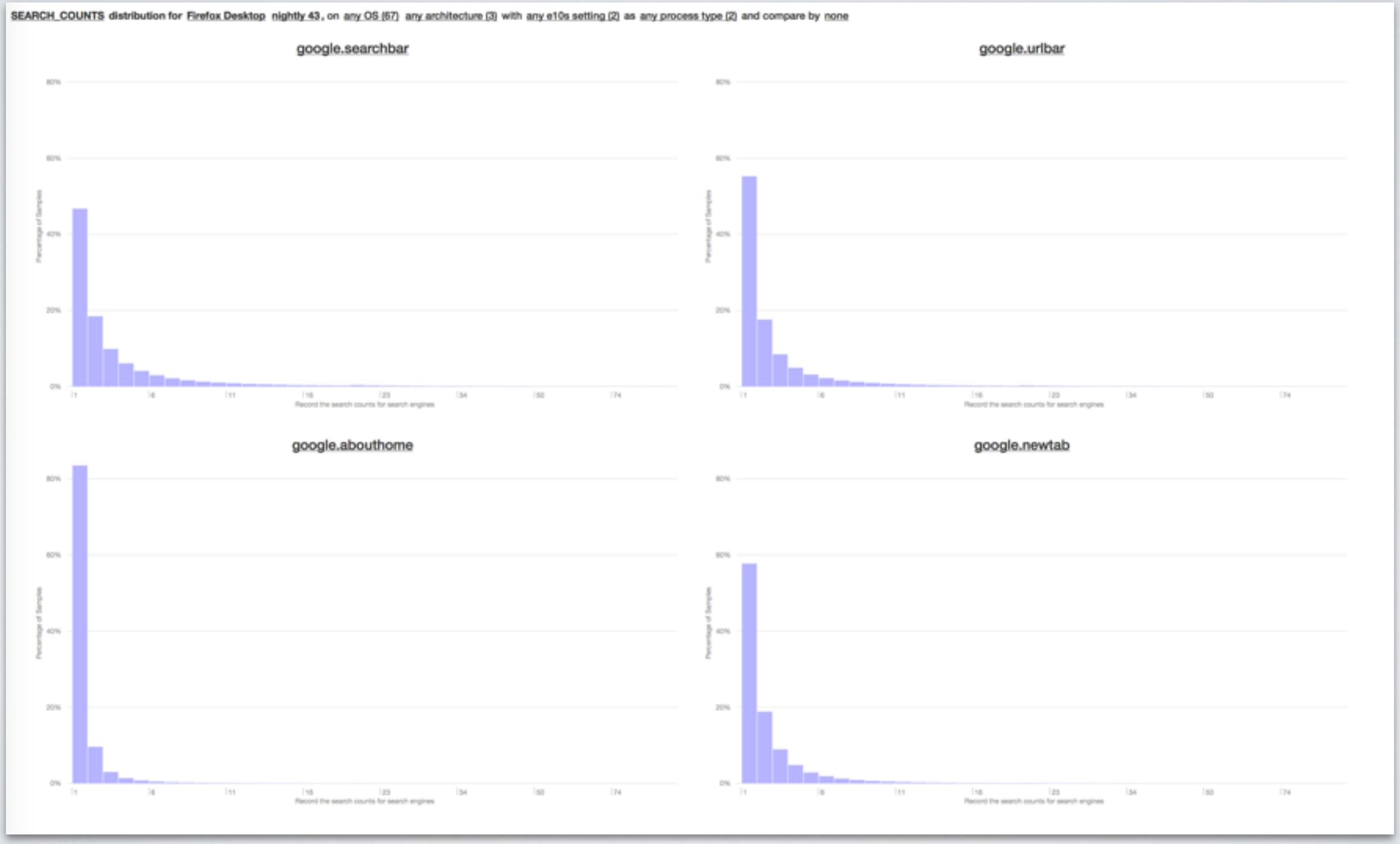
- http://robertovitillo.com/2015/07/02/telemetry-metrics-roll-ups/
- https://anthony-zhang.me/blog/telemetry-demystified/
- http://robertovitillo.com/2014/07/28/regression-detection-for-telemetry-histograms/
- https://telemetry.mozilla.org/
- https://metrics.services.mozilla.com/

ADDING A PROBE

- Telemetry histograms are the preferred way to track numeric measurements such as timings
- There are various types of histograms [1]:
  - flag, e.g. FXA_CONFIGURED
  - boolean, e.g. E10S_WINDOW
  - count, e.g. CONTENT_DOCUMENTS_DESTROYED
  - enumerated, e.g. DEVICE_RESET_REASON
  - linear, e.g. GC_MAX_PAUSE_MS
  - exponential, e.g. GC_MARK_MS

[1] https://developer.mozilla.org/en-US/docs/Mozilla/Performance/Adding_a_new_Telemetry_probe

Keyed histograms are collections of one of the histogram types, indexed by a string key, e.g. SEARCH_COUNTS



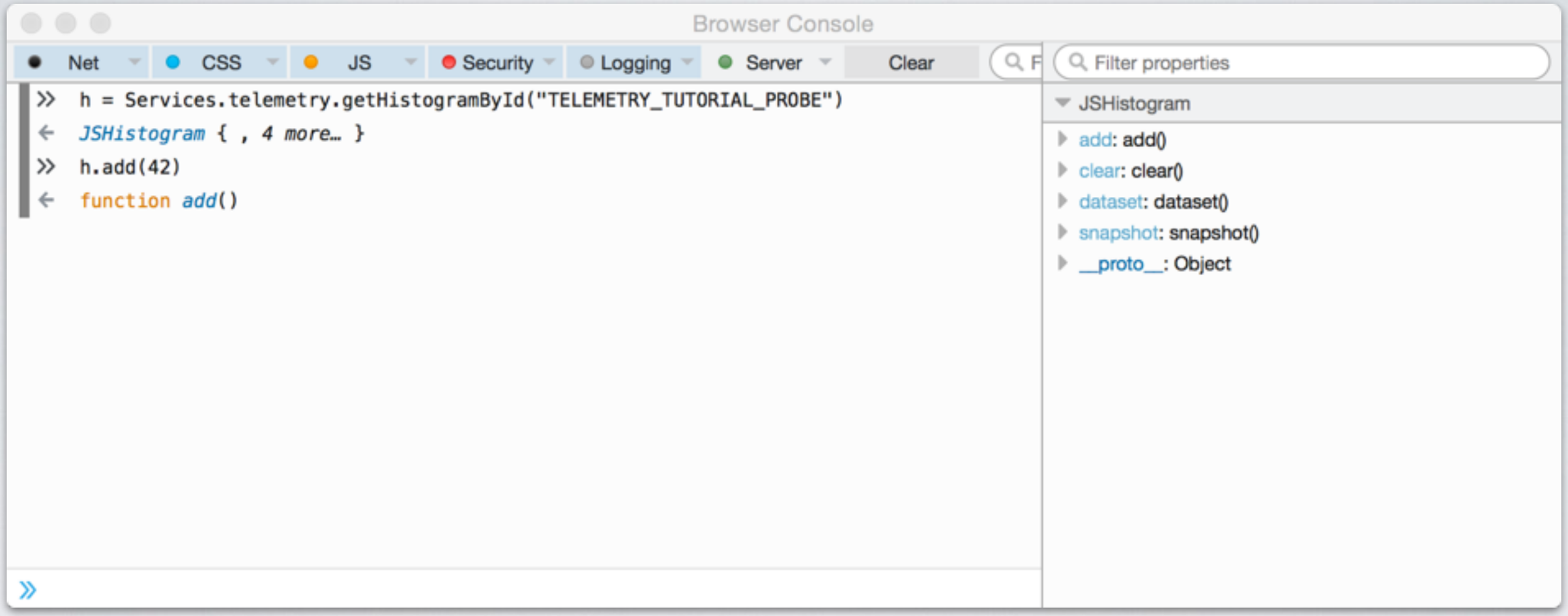
Declaring a Histogram

toolkit/components/telemetry/Histograms.json

```
"TELEMETRY_TUTORIAL_PROBE": {
    "alert_emails": ["rvitillo@mozilla.com"],
    "expires_in_version": "50",
    "kind": "exponential",
    "high": "10000",
    "n_buckets": 50,
    "description": "Telemetry tutorial probe (ms)"
}
```

./mach build toolkit/components/telemetry

Accumulating data



REFERENCES

- https://developer.mozilla.org/en-US/docs/Mozilla/Performance/Adding_a_new_Telemetry_probe

WHAT IS A PING?

- A Telemetry ping [1] is the data that we send to our Telemetry servers.

- Stored on the client as a JSON object

- Contains common information to all pings and a payload specific to a certain ping types.

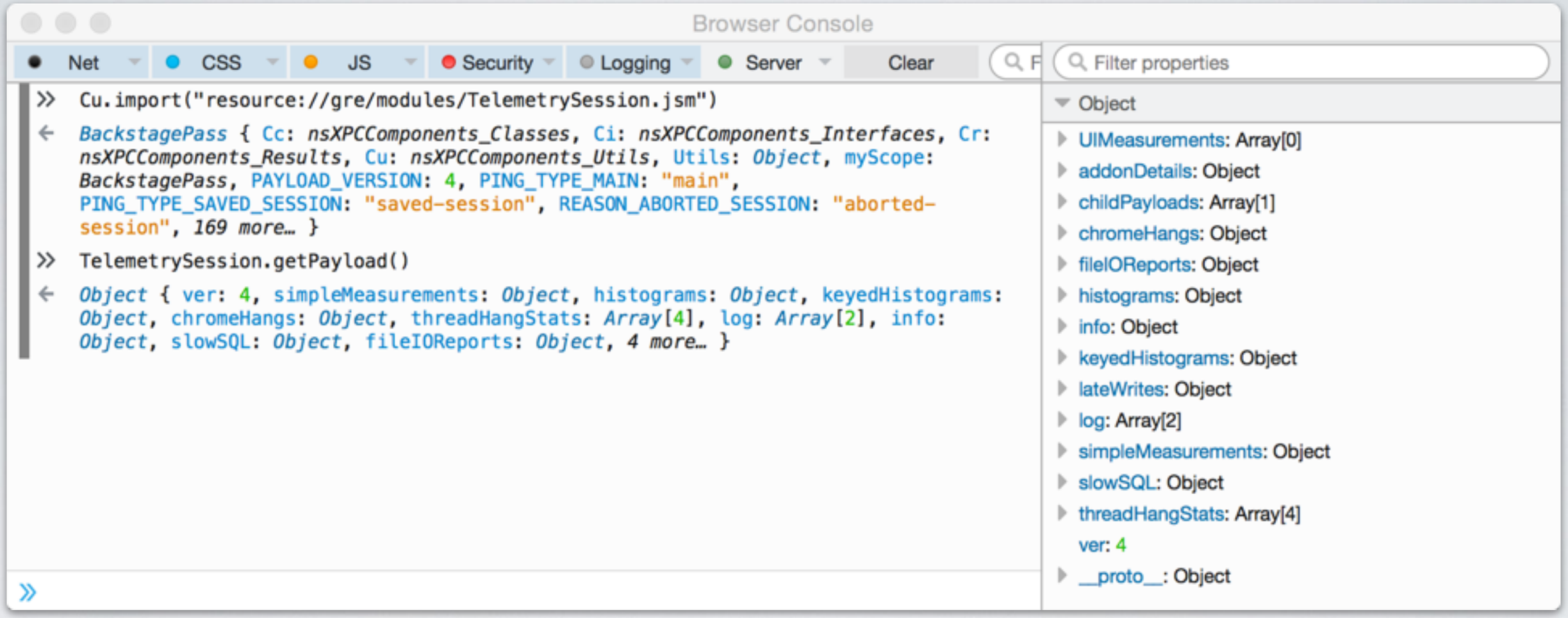
[1] https://ci.mozilla.org/job/mozilla-central-docs/Tree_Documentation/toolkit/components/telemetry/telemetry/pings.html

Ping Types

- Main ping, contains most of the measurements that are used to track the performance and health of Firefox in the wild
  - a ping is triggered by different scenarios, which is documented by the reason field (e.g. shutdown)
  - reasons lead to a session split, initiating a new sub-session; important measurements are reset for those subsessions
- Crash ping, captured after the main Firefox process crashes

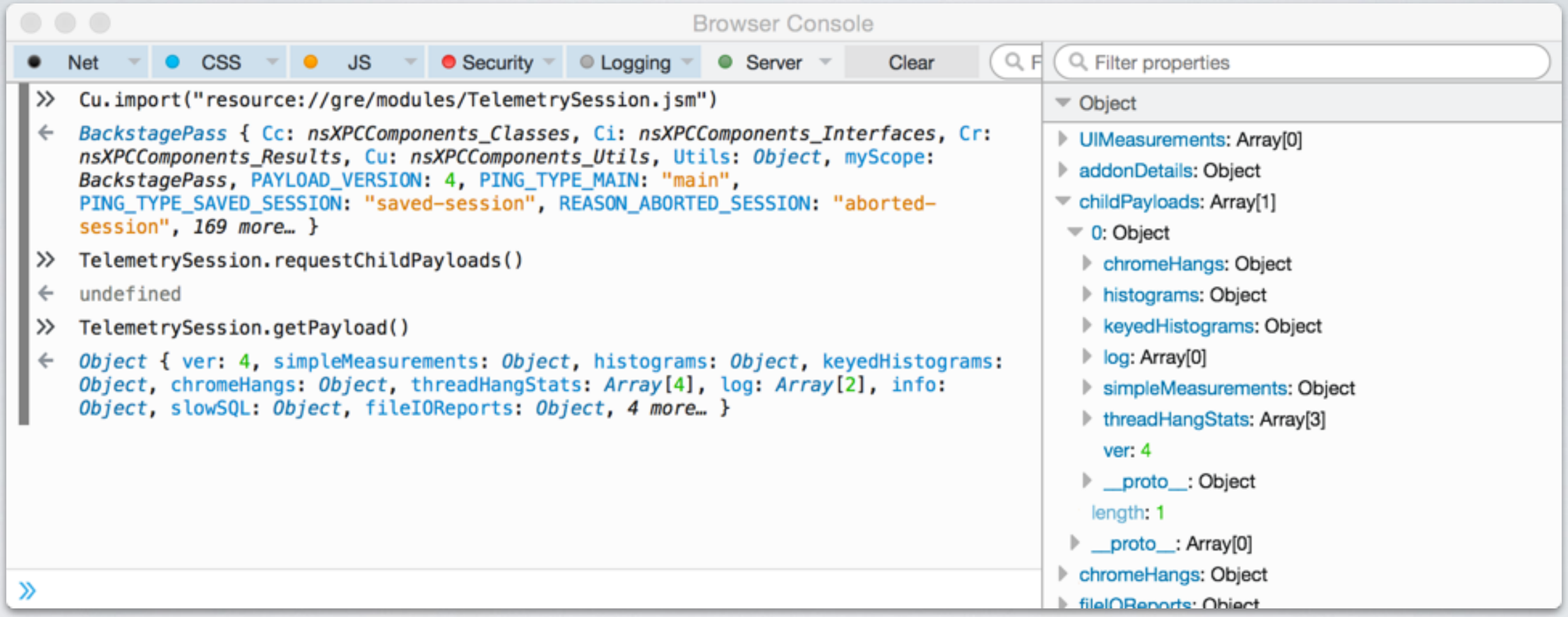
```
{
  type: <string>, // "main", "activation", "deletion", "saved-session", ...
  id: <UUID>, // a UUID that identifies this ping
  creationDate: <ISO date>, // the date the ping was generated
  version: <number>, // the version of the ping format, currently 4

  application: {
    architecture: <string>, // build architecture, e.g. x86
    buildId: <string>, // "20141126041045"
    name: <string>, // "Firefox"
    version: <string>, // "35.0"
    vendor: <string>, // "Mozilla"
    platformVersion: <string>, // "35.0"
    xpcomAbi: <string>, // e.g. "x86-msvc"
    channel: <string>, // "beta"
  },

  clientId: <UUID>, // optional
  environment: { ... }, // optional, not all pings contain the environment
  payload: { ... }, // the actual payload data for this ping type
}
```

Main Ping Payload



Cu.import("resource://gre/modules/TelemetrySession.jsm")

E10s Caveat


Browser Console
Net
CSS
JS
Security
Logging
Server
Clear
Filter properties
Cu.import("resource://gre/modules/TelemetrySession.jsm")
BackstagePass { Cc: nsXPCComponents_Classes, Ci: nsXPCComponents_Interfaces, Cr: nsXPCComponents_Results, Cu: nsXPCComponents_Utils, Utils: Object, myScope: BackstagePass, PAYLOAD_VERSION: 4, PING_TYPE_MAIN: "main", PING_TYPE_SAVED_SESSION: "saved-session", REASON_ABORTED_SESSION: "aborted-session", 169 more… }
TelemetrySession.requestChildPayloads()
undefined
TelemetrySession.getPayload()
Object { ver: 4, simpleMeasurements: Object, histograms: Object, keyedHistograms: Object, chromeHangs: Object, threadHangStats: Array[4], log: Array[2], info: Object, slowSQL: Object, fileIOReports: Object, 4 more… }
Object
UIMeasurements: Array[0]
addonDetails: Object
childPayloads: Array[1]
0: Object
chromeHangs: Object
histograms: Object
keyedHistograms: Object
log: Array[0]
simpleMeasurements: Object
threadHangStats: Array[3]
ver: 4
__proto__: Object
length: 1
__proto__: Array[0]
chromeHangs: Object

Environment

- Consists of data that is expected to be characteristic for performance and other behaviour and not expected to change too often (e.g. number of CPU cores)

- Changes to most of these data points are detected and lead to a session split in the "main" ping

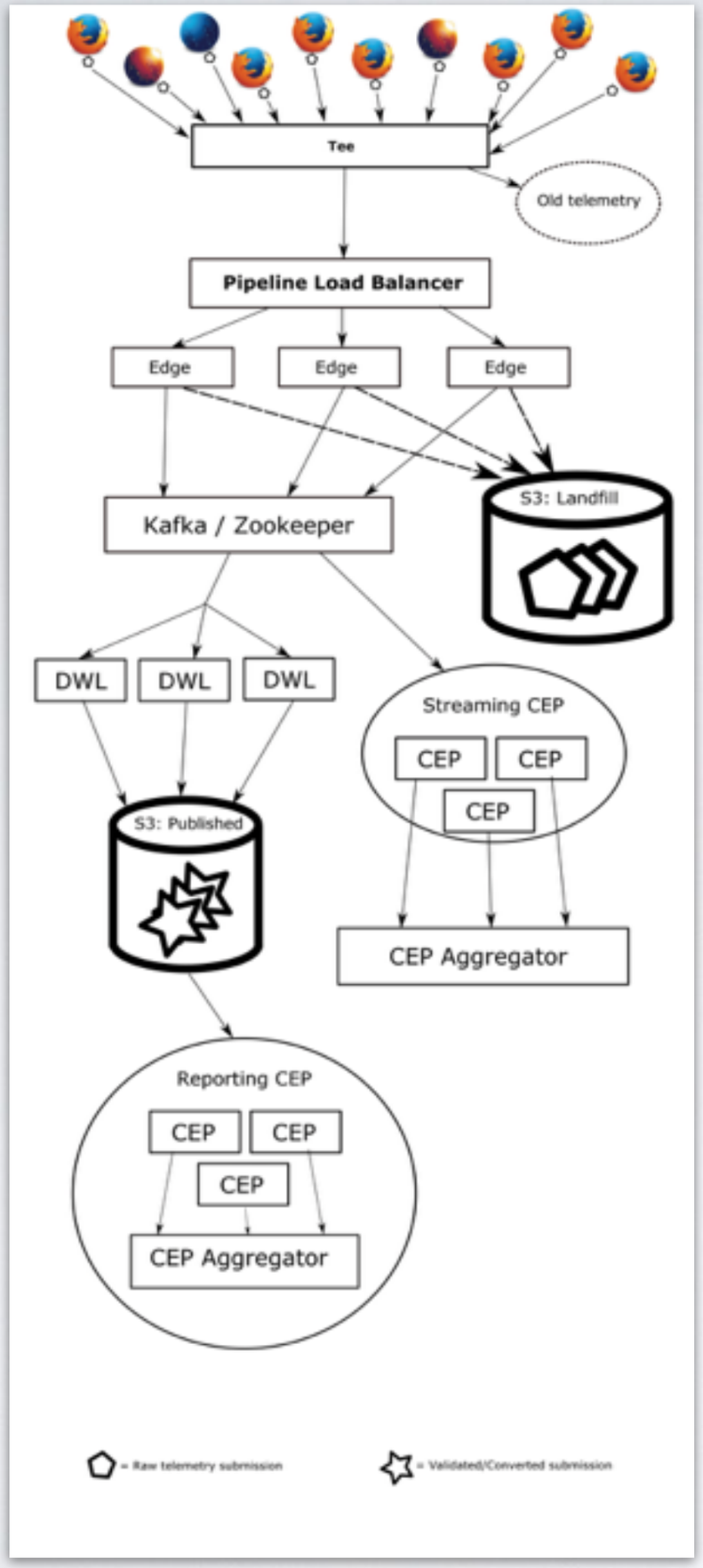
REFERENCES

- https://ci.mozilla.org/job/mozilla-central-docs/Tree_Documentation/toolkit/components/telemetry/telemetry/pings.html

DATA PIPELINE

- The data pipeline ingests, transforms, stores and analyzes data
- The endpoint is an HTTP server that:
  - listens for POST/PUT from Firefox
  - does some decoding / preprocessing
  - makes the data available for streaming analyses (Heka)
  - archives the data to S3 for further offline analyses (Spark)

[1] https://wiki.mozilla.org/CloudServices/DataPipeline



CEP = Complex Event Processor (basically streaming analysis or reporting)

DWL = Data Warehouse Loader

Landfill = Shorthand for "write-mostly store for backup / recovery purposes"

REFERENCES

- https://wiki.mozilla.org/CloudServices/DataPipeline
- https://github.com/mozilla-services/data-pipeline
- https://github.com/mozilla-services/heka
- https://github.com/apache/spark/

Break Time!

Regroup in a few minutes

OFFLINE PROCESSING

IPYTHON, PANDAS & SPARK

https://github.com/vitillo/telemetry-onboarding/tree/master/notebooks

EXPERIMENTS

- Telemetry Experiments is a feature that allows Firefox to automatically download and run specially-designed restartless addons based on certain conditions

- Currently only available in desktop versions of Firefox

- Experiments can be targeted at various populations by specifying conditions in an experiment manifest

- Before building an experiment, contact a data-collection peer; additional privacy or security review might be required

- Product approval is required to run an experiment

[1] https://wiki.mozilla.org/Telemetry/Experiments

1. Setup staging server [1], which Firefox is going to contact to install the current experiment

2. Write restart-less add-on [2]

   - addon.js, when startup() is called, it must manually inject its user interface and other behavior into the application; when shutdown() is called, it must remove anything that it has added to the application

   - install.rdf, used to determine information about an add-on as it is being installed (e.g. author, version, etc.)

3. manifest.json, the experiment manifest which targets a specific population

[1] https://wiki.mozilla.org/QA/Telemetry/Installing_Staging_Server

[2] https://developer.mozilla.org/en-US/Add-ons/Bootstrapped_extensions

Flags required for testing

- experiments.force-sample-value = “0.0”
- experiments.logging.level = 0
- experiments.manifest.cert.checkAttributes = false
- experiments.manifest.uri = “http://localhost:8000/firefox-manifest.json”
- xpinstall.signatures.required = false

```
let {classes: Cc, interfaces: Ci, utils: Cu} = Components;

Cu.import("resource:///modules/experiments/Experiments.jsm");
Cu.import("resource://gre/modules/Task.jsm");
Cu.import("resource://gre/modules/Preferences.jsm");

var gStarted = false;

const kSELF_ID = "flash-protectedmode-beta35@experiments.mozilla.org";

function startup() {
  // Seems startup() function is launched twice after install, we're
  // unsure why so far. We only want it to run once.
  if (gStarted) {
    return;
  }
  gStarted = true;

  Task.spawn(function*() {
    let branch = yield Experiments.instance().getExperimentBranch(kSELF_ID);
    switch (branch) {
      case "control":
        return;
      case null:
        let r = (Math.random() >= 0.5);
        if (!r) {
          yield Experiments.instance().setExperimentBranch(kSELF_ID, "control");
          return;
        }
        yield Experiments.instance().setExperimentBranch(kSELF_ID, "experiment");
        // FALL THROUGH
      case "experiment":
        let defaultPrefs = new Preferences({defaultBranch: true});
        defaultPrefs.set("dom.ipc.plugins.flash.disable-protected-mode", true);
        return;
      default:
        throw new Error("Unexpected experiment branch: " + branch);
    }
  }).then(
    function() {
    },
    function(e) {
      Cu.reportError("Got error during bootstrap startup: " + e);
    });
}

function shutdown() {
  let defaultPrefs = new Preferences({defaultBranch: true});
  defaultPrefs.set("dom.ipc.plugins.flash.disable-protected-mode", false);
}
```

https://bugzilla.mozilla.org/show_bug.cgi?id=1110215

```
{
  "publish"      : true,
  "priority"     : 2,
  "name"         : "Flash Protected-Mode Testing",
  "description" : "Measuring the effect of Flash protected mode on crashes, hangs, and other browser jank.",
  "info"         : "<p><a href=\"https://bugzilla.mozilla.org/show_bug.cgi?id=1110215\">Related bug</a></p>",
  "manifest"     : {
    "id"                : "flash-protectedmode-beta35@experiments.mozilla.org",
    "startTime"         : 1418601600,
    "endTime"           : 1421280000,
    "maxActiveSeconds" : 2764800,
    "appName"           : ["Firefox"],
    "channel"           : ["beta"],
    "os"                : ["WINNT"],
    "minVersion"        : "35.0",
    "minBuildID"        : "20141215000000",
    "maxVersion"        : "37.*",
    "sample"            : 0.1,
    "disabled"          : true
  }
}
```

http://hg.mozilla.org/webtools/telemetry-experiment-server/file/tip/experiments/flash-protectedmode-beta/manifest.json

Bug 1111791 - Telemetry report: effect of the Flash protected-mode experiment

https://bugzilla.mozilla.org/show_bug.cgi?id=1111791

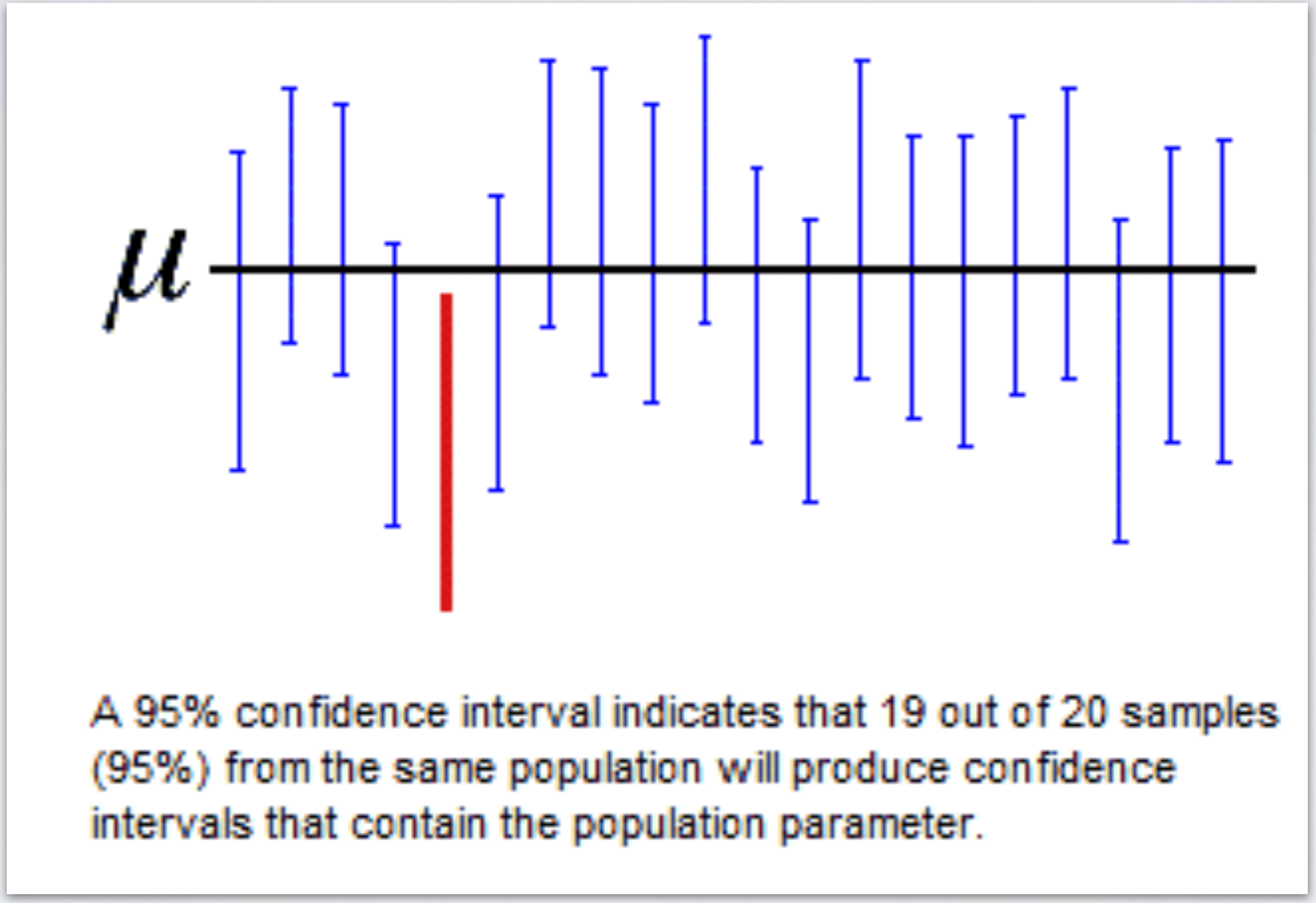
REFERENCES

- https://wiki.mozilla.org/Telemetry/Experiments
- https://developer.mozilla.org/en-US/Add-ons/Bootstrapped_extensions
- https://wiki.mozilla.org/QA/Telemetry#Telemetry_Experiments.2FFHR_Documentation
- http://codefirefox.com/video/install-telmetry-experiment
- http://hg.mozilla.org/webtools/telemetry-experiment-server/file/tip/experiments
- https://bugzilla.mozilla.org/show_bug.cgi?id=1110215
- https://bugzilla.mozilla.org/show_bug.cgi?id=1111791

STATS

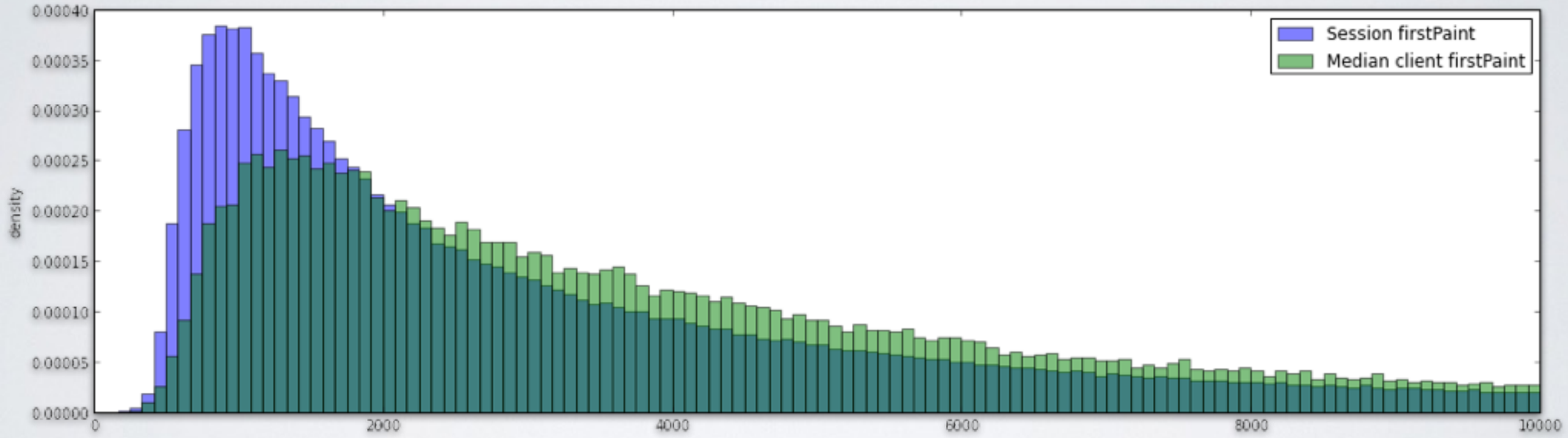
USE REPRESENTATIVE SAMPLES

USE SUFFICIENT DATA



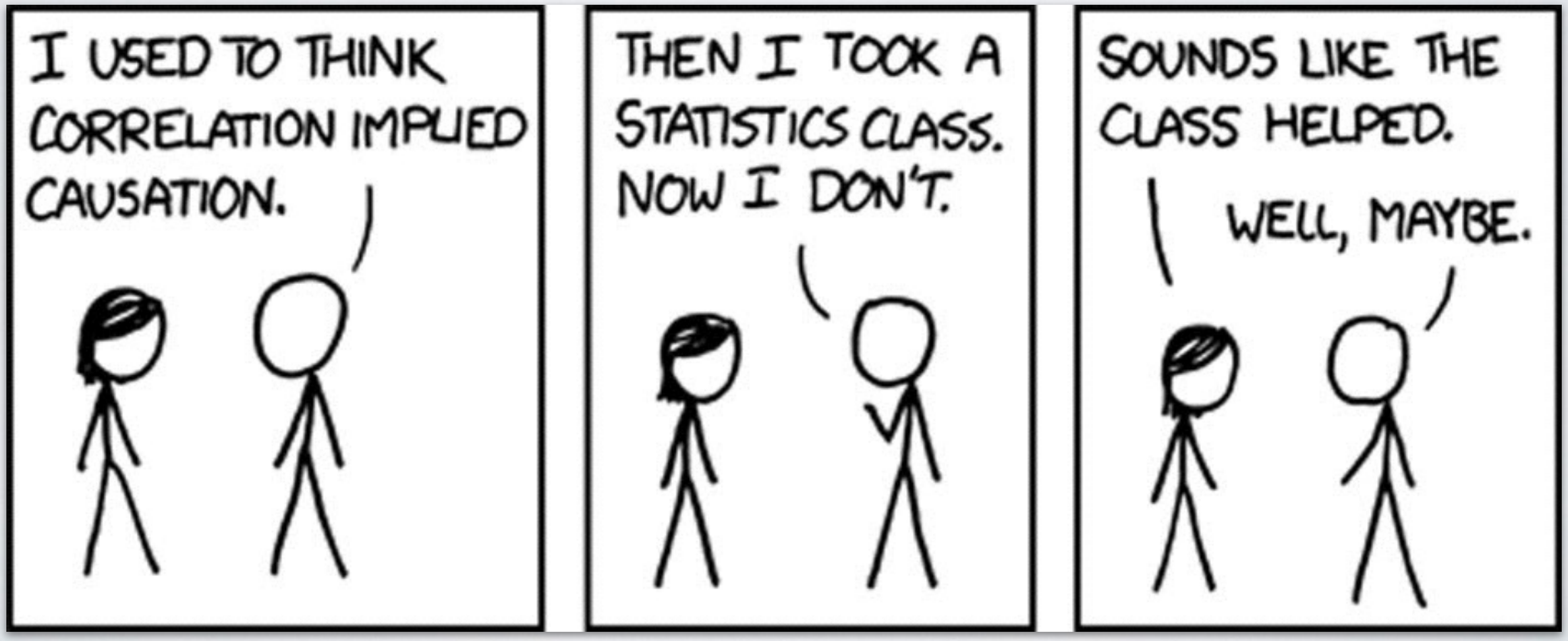
http://robertovitillo.com/2015/03/15/confidence-intervals-and-hypothesis-tests-for-engineers/

BEWARE OF PSEUDOREPLICATION



http://robertovitillo.com/2014/12/19/clientid-in-telemetry-submissions/

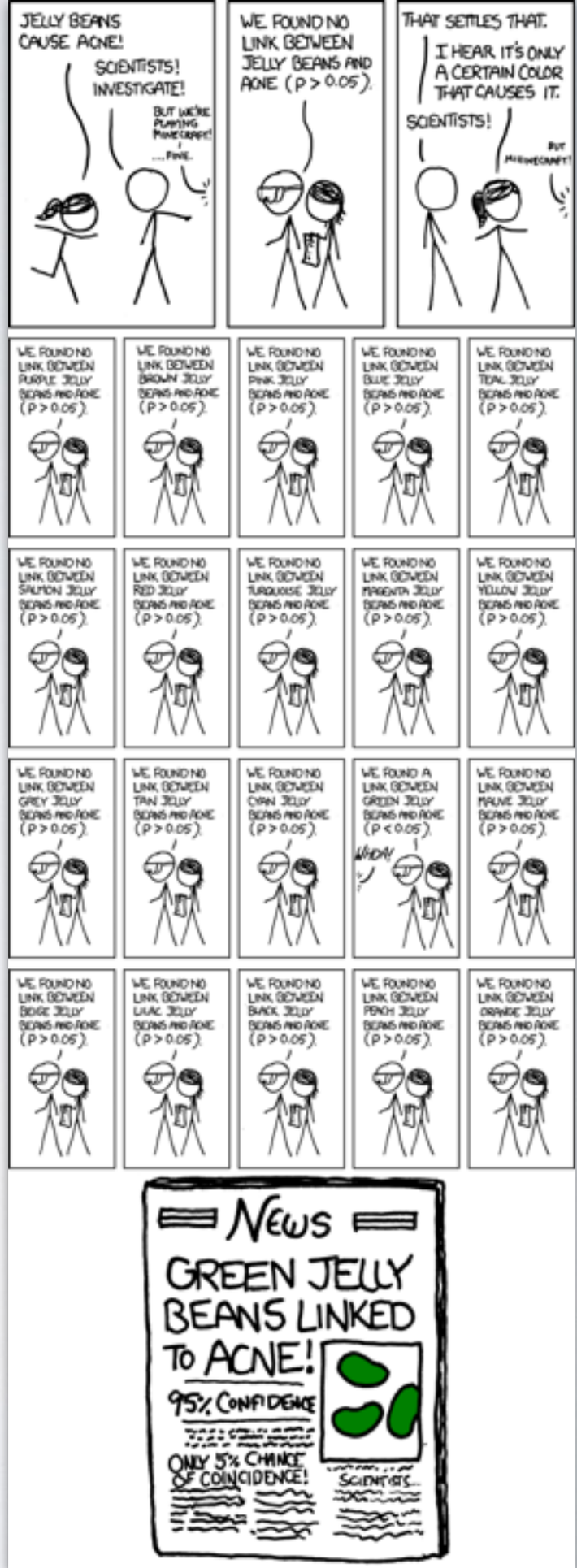
CORRELATION IS NOT CAUSATION



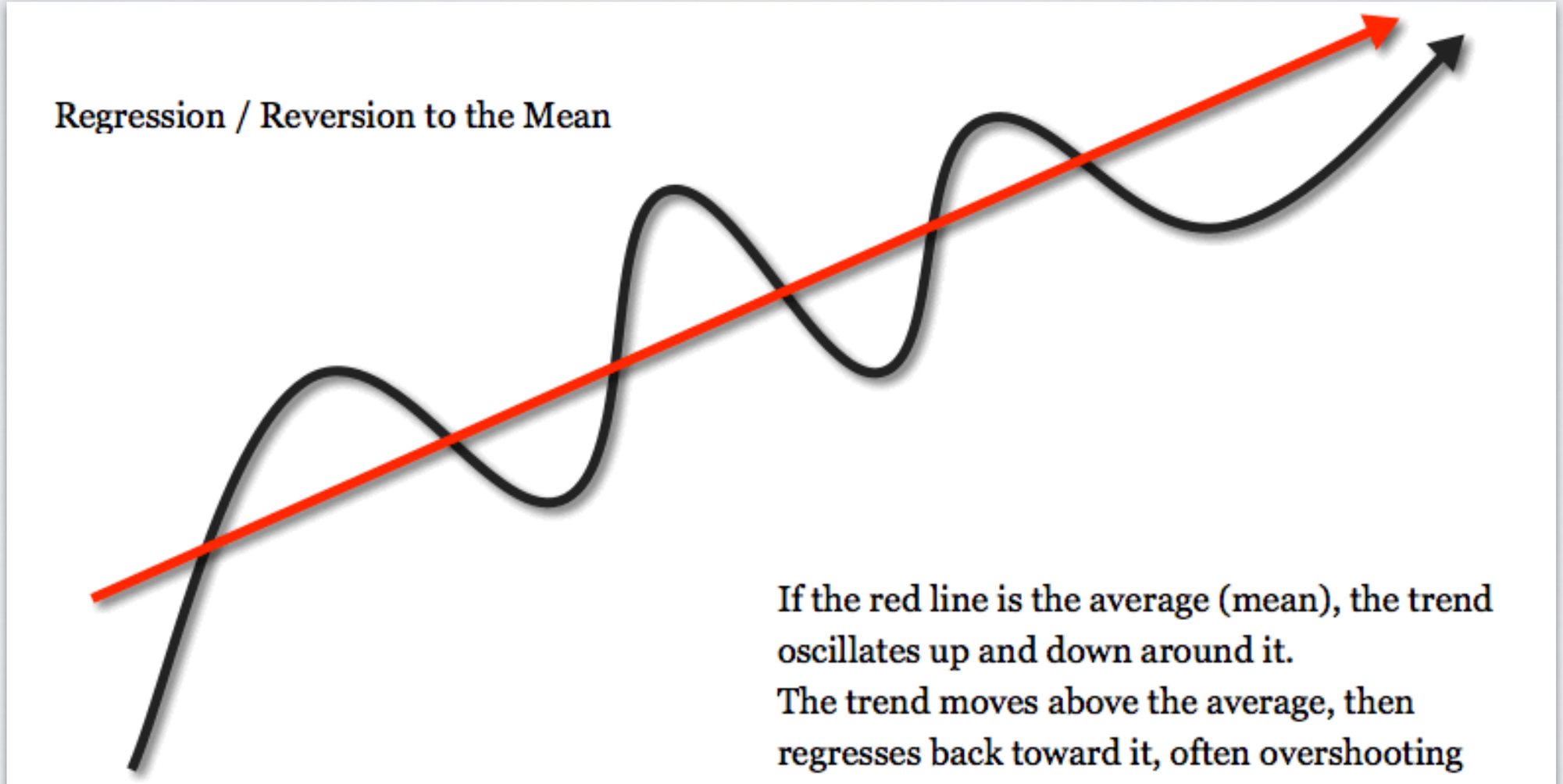
USE THE RIGHT AVERAGE

- Mean: add up the values, and divide by the number of values
- Median: the median is the 50th percentile; half the values are higher than the median, and half are lower
- Geometric mean: compute the logarithm of all values, compute the mean of the logarithms, and then take the antilog
    - It is a better measure of central tendency when data follow a lognormal distribution (long tail).

CONTROL THE FALSE DISCOVERY RATE



REGRESSION TOWARD THE MEAN



MAKE YOUR ANALYSES REPRODUCIBLE

In [1]:
```
import ujson as json
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import plotly.plotly as py

from moztelemetry import get_pings, get_pings_properties, get_one_ping_per_client, get_clients_history

%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

In [7]:
```
pings = get_pings(sc, app="Firefox", channel="release", submission_date="20150928", build_id="20150917150946")
```

In [8]:
```
def telemetry_enabled(ping):
    return ping.get("environment", {}).get("settings", {}).get("telemetryEnabled", False)
```

In [*]:
```
pings.count()
```

Out[9]:
```
1242836
```

REFERENCES

- http://www.statisticsdonewrong.com/data-analysis.html

- http://www.amazon.com/How-Lie-Statistics-Darrell-Huff/dp/0393310728

- http://www.slideshare.net/RobertoAgostinoVitil/all-you-need-to-know-about-statistics

PRIVACY POLICY

1. No surprises: use and share information in a way that is transparent and benefits the user

2. User control: develop products and advocate for best practices that put users in control of their data and online experiences

3. Limited data: collect what we need, de-identify where we can and delete when no longer necessary

4. Sensible settings: design for a thoughtful balance of safety and user experience

5. Defense in depths: maintain multi-layered security controls and practices, many of which are publicly verifiable

https://www.mozilla.org/en-US/privacy/

DATA COLLECTION POLICY

- When proposing a new measurement or data system, consider the requirements and the necessary data properties, e.g:
  - is it necessary to take a measurement from all users? Or is it sufficient to measure only prerelease users?
  - is it desirable to track data changes over time? With what frequency? With what latency?
- For every new measurement, even a simple new Telemetry probe, please request approval by setting the feedback flag for the module owner or a peer.
  - Owner: Benjamin Smedberg
  - Peers: Vladan Djeric, Ally Naagktgeboren

https://wiki.mozilla.org/Firefox/Data_Collection