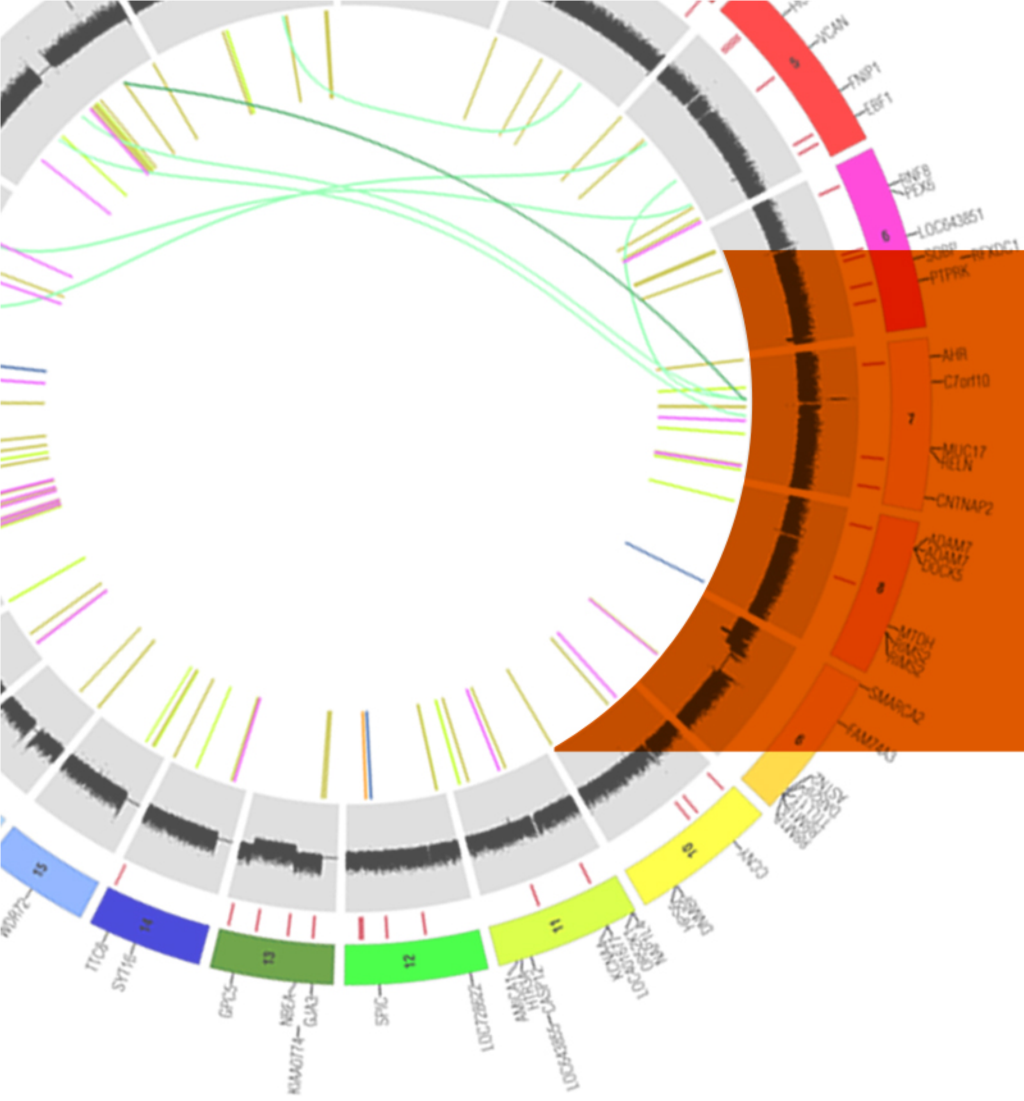


The Cancer Genome Atlas

Gettting MAFs into Firehose:

Enhancing Automation and Transparency

David I. Heiman

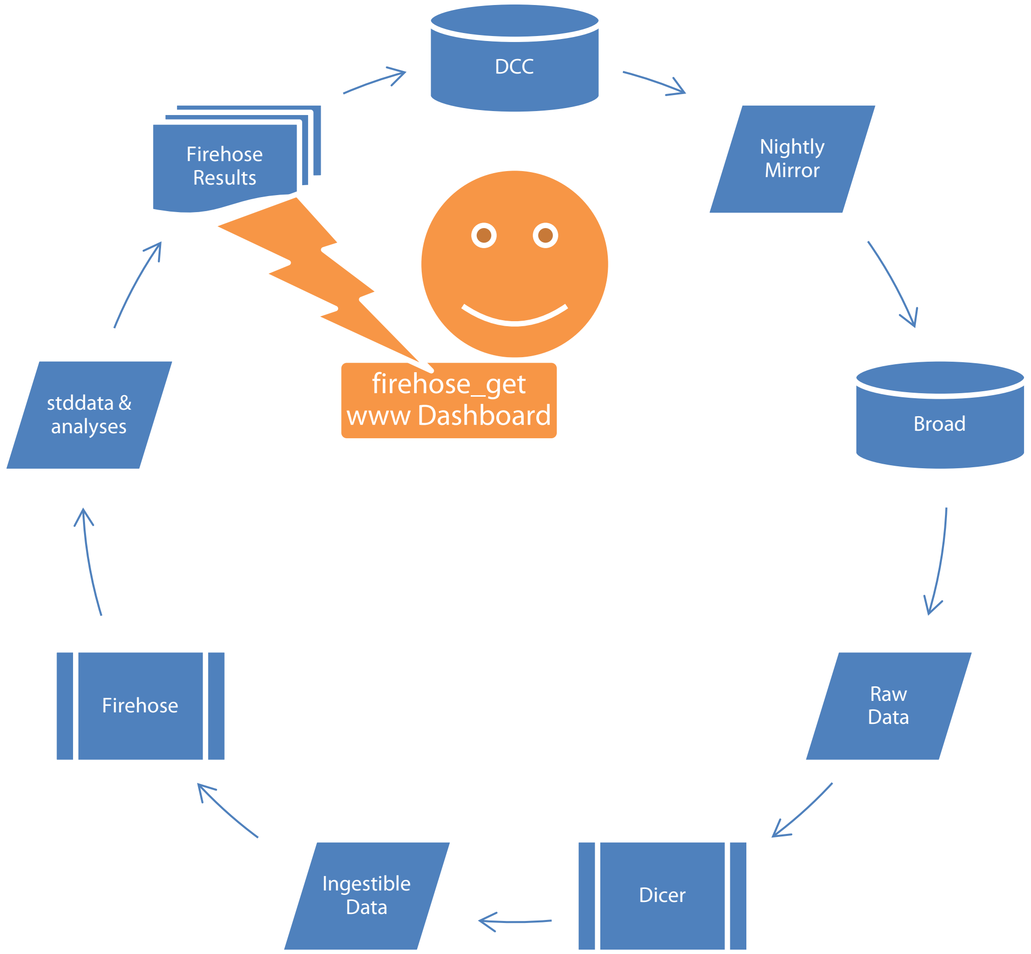
TCGA Informatics WG call

Thursday, March 21, 2013

FIREHOSE
Broad GDAC

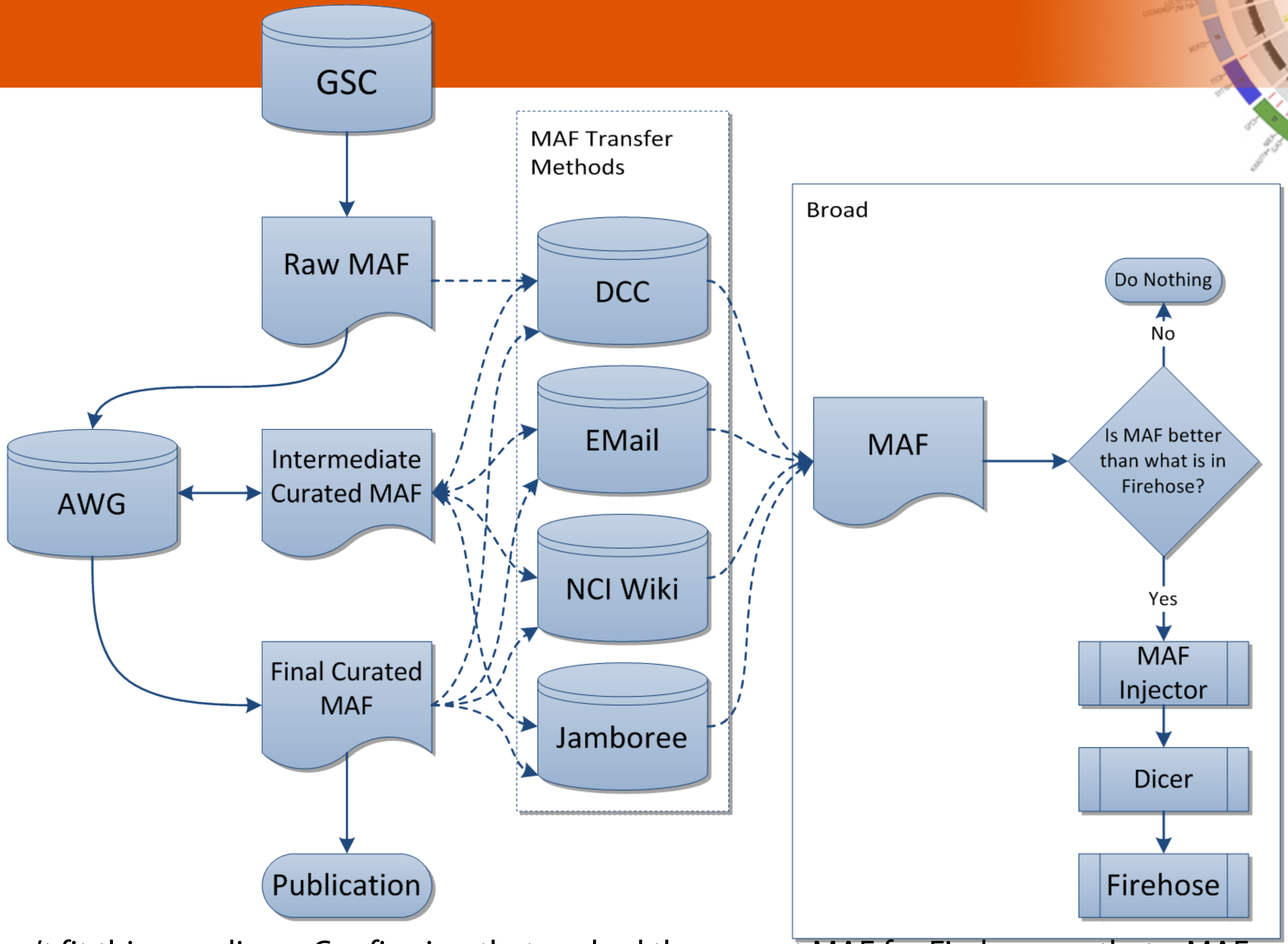
Standard Flow of GDAC Firehose Data

The mostly automated standard Firehose workflow. We maintain a mirror of the DCC locally, dice it into Firehose-ingestible data, and perform our Firehose runs on chosen dicings.



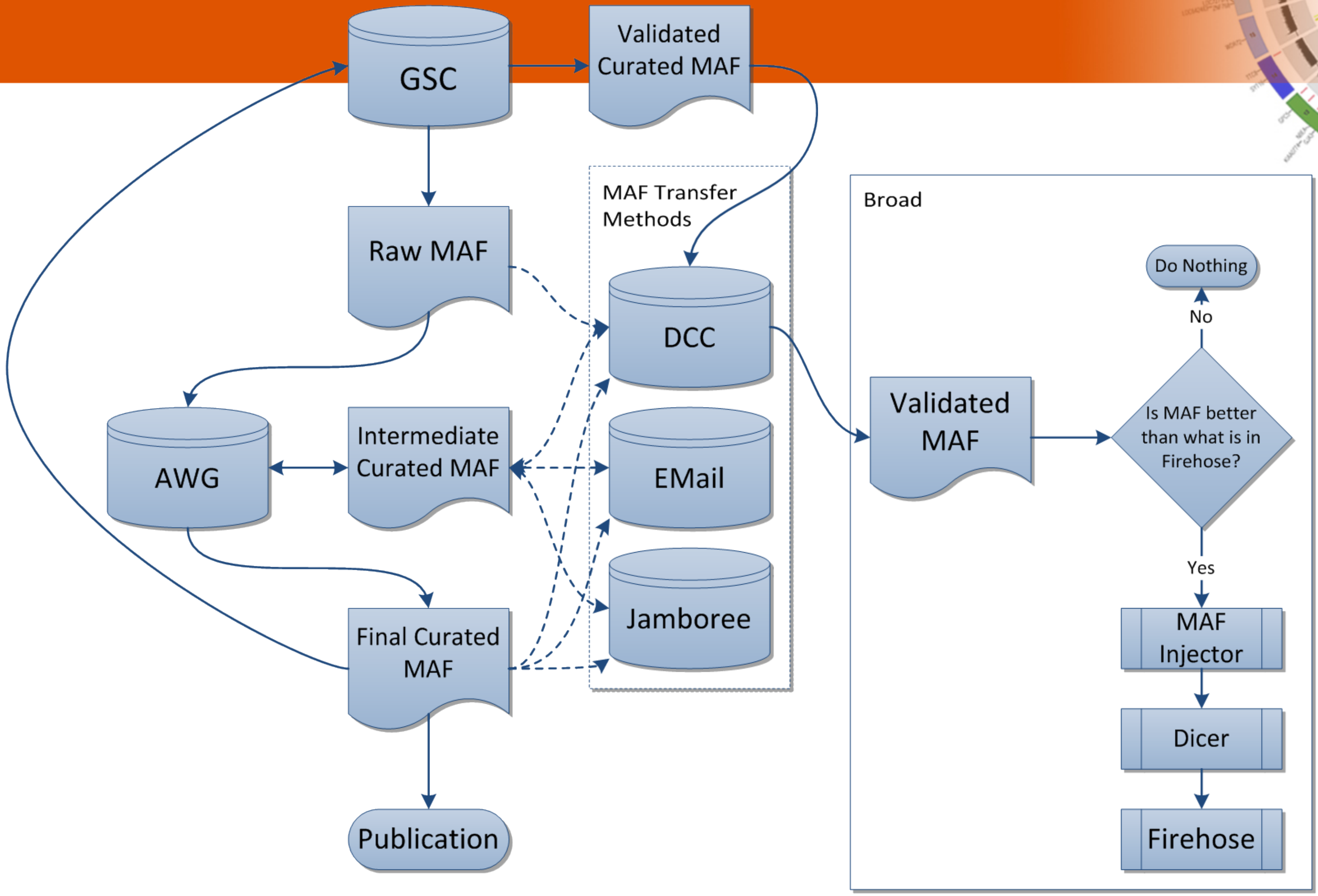
The Cancer Genome Atlas

Old MAF Data Flow



MAFs clearly don't fit this paradigm. Confirming that we had the correct MAF for Firehose, or that a MAF was even available could be difficult. There are probably still a few MAFs in Firehose that are intermediate curated MAFs rather than final ones. There was little direct communication about the availability of MAFs, we simply had to watch everything. Things obviously fell through the cracks.

New MAF Data Flow



New rules, and enforcement thereof, have helped enormously to streamline the overall process. We now require that any MAF we ingest into Firehose MUST exist outside of it in a publically accessible fashion.

MAF Processing Requires Manual Intervention

- Location of coverage files (WIGs) is still not standardized
  - Should be included with every submission as part of the MAF archive, but in some cases they're only on the Jamboree site, or possibly in a previous submission, or may not exist at all, in which case we have to use fake ones.
- "New" does not always mean "Best"
  - We prioritize AWG curated MAFs over uncurated
    - But, this could cause issues with new data generated for a tumor with a defunct AWG – how can we release viable new data if it hasn't been curated?
    - Broad GSC submits uncurated MAFs to the DCC
  - DCC has been reprocessing v2.2 MAFs into v2.3, increasing the revision number and changing timestamps

MAF Processing Requires Manual Intervention

- Inconsistent versioning:
  - Format: <Serial Index>.<Revision>.0
    - Serial Index:
      - WashU: Increments after each Data Freeze by the AWG, all uploaded MAFs are AWG-curated
      - Broad: 0 indicates Auto-GSC generated MAF, 1 indicates AWG-curated MAF
      - Baylor: ?
    - Revision:
      - WashU: Changes since last Data Freeze (resets with increment of Serial Index)
      - Broad: Current data revision (archive version 1.n.0 based on archive version 0.n.0)
      - Baylor: ?
      - DCC: May increment after any auto-processing (e.g. converting from v2.2 to v2.3)
  - Broad's versioning system became a policy last year, so earlier data doesn't follow it. For COAD, the Illumina MAFs submitted by Baylor contain completely different samples between 1.1.0 and 1.2.0.

How We Obtain MAFs: Old Way

- It was like the wild west
- We had someone watching the AWG and TCGA MAF Files wiki pages at the NCI Wiki
- Some MAFs didn't exist in either location – they could be passed around by email, or located at the Jamboree site.
- What it often came down to was someone would notice we didn't have a current MAF, contact us, and we put it in, updating the TCGA MAF Dashboard Google Doc as we went along.

Issues with the old way

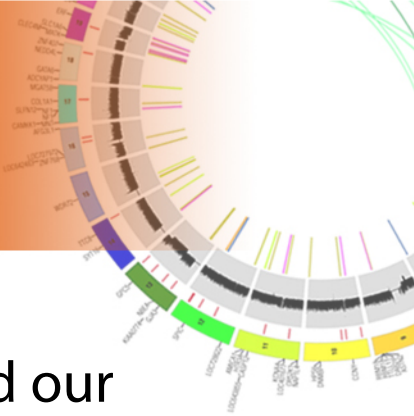
- Extremely difficult to maintain consistency
- Not all of these MAFs were accessible to anyone outside of the TCGA, making Firehose the only source in several cases.
  - Firehose should not be the only way to obtain these MAFs – our purview is to process and provide data snapshots of publicly available TCGA data.
- It’s turned out that in several cases, what we thought was the final AWG-curated MAF was really an intermediate one.
- We took unvalidated MAFs, and had to run special cleaning processes on them, customizing and recustomizing on a regular basis

How We Obtain MAFs: New Way

Policy Changes Help Enforce Transparency

- Starting this year, we aim to *ONLY* accept new MAFs available from the DCC.
  - No need to clean a validated MAF
- All MAFs we have must be publicly available outside of Firehose (currently working on syncing colorectal and ovarian MAFs to the publications)
- Enhance the TCGA MAF Dashboard to have greater detail on the publicly available DCC MAFs, as well as what is in Firehose

Steps Toward Automation



- New methods that parse our local copy of the DCC and our overlay of processed files to generate a detailed summary of current MAFs.
  - The new DCC Snapshot and updated Firehose Ingested tables on the TCGA MAF Dashboard reflect this output.
- Methods to do basic comparisons of MAFs that are version-agnostic

Parsing Available MAFs

```python
# Returns an array of archives/file paths determined by data_func
def get_archive_data(self, data_func, public=True, tumor_glob='*',
                     center_type_glob='*', center_glob='*',
                     technology_glob='*', data_type_glob='*',
                     archive_glob='*', file_glob=''):
    '''Required argument is a data choosing function (several are supplied
    by this class).  Optionally, the user may define more specific globs to
    pinpoint their search.  The path glob resolves to:
        <ROOT>/<tumor_glob>/<center_type_glob>/<center_glob>/<technology_glob>/<data_type_glob>
    ROOT is determined by the public flag.  Default is public.
    file_glob can be used to look for specific files.  e.g '*.maf.*' may be
    used to only return paths to maf files.  If unset, only paths to
    archives are returned.
    Returns a list of paths to the chosen data'''
    path_glob = self.public_root if public else self.secure_root
    path_glob = os.path.join(path_glob, tumor_glob, center_type_glob,
                             center_glob, technology_glob, data_type_glob)
    data_paths = []
    for path in glob.iglob(path_glob):
        for archive in data_func(path, archive_glob):
            data_glob = os.path.join(archive, file_glob)
            data = glob.glob(data_glob)
            # Sometimes empty directories exist. They are ignored, but maybe
            # they should cause a warning?
            if data:
                data_paths += data
    return data_paths
```

```python
# Most recent archive by serial index, then revision
@staticmethod
def max_archive_by_serial_index(path, archive_glob):
    '''Takes as arguments the path to the archives and a glob to specify
    which archives to examine.  Yields the path of the most recent archive
    determined first by serial index, then by revision to break ties.'''
    serial_index = -1
    revision = -1
    data_path = ''
    for archive in glob.iglob(os.path.join(path, archive_glob)):
        archive_ary = archive.split('.')
        cur_si = int(archive_ary[-3])
        cur_rev = int(archive_ary[-2])
        if (cur_si, cur_rev) > (serial_index, revision):
            serial_index = cur_si
            revision = cur_rev
            data_path = archive
    if data_path:
        yield data_path
```

```python
# Most recent revision for each serial index
@staticmethod
def max_archive_per_serial_index(path, archive_glob):
    '''Takes as arguments the path to the archives and a glob to specify
    which archives to examine.  Returns the paths of the most recent
    archives for each serial index'''
    serial_indices = {}
    for archive in glob.iglob(os.path.join(path, archive_glob)):
        serial_index, revision = archive.split('.')[-3:-1]
        if not (serial_indices.has_key(serial_index) and
                int(serial_indices[serial_index].split('.')[-2]) >= int(revision))
            serial_indices[serial_index] = archive
    return serial_indices.itervalues()
```

Parsing Available MAFs

```
def mirror_per_serial_index(self, public=True, tumor_glob='*',
                            center_glob='*', technology_glob='*'):
    '''Returns the file paths for the most recent revisions of each serial
    index in the mirror'''
    return self.mirror.get_archive_data(self.mirror.max_archive_per_serial_index,
                                        public, tumor_glob,
                                        self.center_type_glob, center_glob,
                                        technology_glob,
                                        self.mirror_data_type_glob,
                                        self.archive_glob, self.file_glob)

def mirror_by_serial_index(self, public=True, tumor_glob='*',
                           center_glob='*', technology_glob='*'):
    '''Returns the file paths for the most recent revisions of the most
    recent serial indices in the mirror'''
    return self.mirror.get_archive_data(self.mirror.max_archive_by_serial_index,
                                        public, tumor_glob,
                                        self.center_type_glob, center_glob,
                                        technology_glob,
                                        self.mirror_data_type_glob,
                                        self.archive_glob, self.file_glob)
```

Summarizing Available MAFs

```
@staticmethod
def path_summary(path):
    '''Returns a dict of tumor type, center, and archive version'''
    path.rstrip(os.path.sep)
    inc = 1 if os.path.isdir(path) else 0
    path_split = path.split(os.path.sep)
    tumor = path_split[-7 + inc]
    center = path_split[-5 + inc]
    archive = '.'.join(path_split[-2 + inc].split('.')[-3:])
    return {'Tumor Type': tumor,
            'Center': center,
            'Archive Version': archive}
```

```
def maf_summary(self, maf, status=None):
    '''Returns a dict of file and mutation stats'''
    tumor_ct, normal_ct, mutations, cols = self.maf_counts(maf, status)
    size, last_mod = os.stat(maf)[6:9:2]
    return {'File Name': os.path.basename(maf),
            'Tumor Samples': str(tumor_ct),
            'Normal Samples': str(normal_ct),
            'Mutations': str(mutations),
            'Columns': str(cols),
            'Size': CommonFunctions.sizeof_fmt(size),
            'Last Modified':
                time.strftime('%d %b %Y', time.localtime(last_mod)),
            'Location': maf,
            'md5sum': CommonFunctions.md5sum(maf)}
```

```
def maf_summary(self, maf, status=None):
    return dict(self.maf_finder.path_summary(maf), **self.maf_tools.maf_summary(maf, status))

def mirror_maf_summary(self, maf, status=None, ext_locs=True):
    summary = self.maf_summary(maf, status)
    if ext_locs:
        return tuple(self._mirror_maf_summary_generator(summary))
    return tuple(summary[header] for header in self.mirror_summary_headers)

def _mirror_maf_summary_generator(self, summary):
    for header in self.mirror_summary_headers:
        if header == 'Location':
            yield summary[header].replace(self.maf_finder.mirror.location, 'https:/', 1)
        else:
            yield summary[header]
```

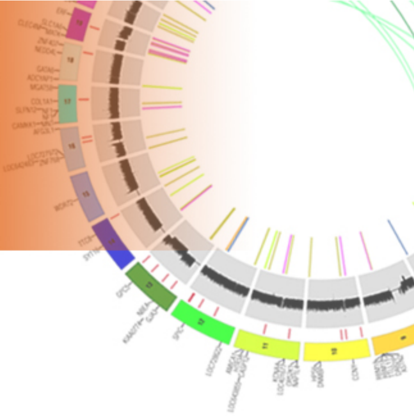
Comparing MAFs

```python
def info_generator(self, data, header_indices):
    for key in header_indices.iterkeys():
        if key != self.stat_header:
            if key != self.tumor_barcode_header:
                yield data[header_indices[key]]
            else:
                yield self.maf_indiv(header_indices, data)

def maf_indiv(self, indices, data):
    return data[indices[self.tumor_barcode_header]][5:12]

def build_maf_set(self, filename, status=None):
    header_indices = { head : -1 for head in self.cmp_headers }
    header_indices[self.tumor_barcode_header] = -1
    header_indices[self.stat_header] = -1
    header = True
    maf_set = set()
    maf = open(filename, 'r')
    for line in maf:
        if not line.startswith('#'):
            if header:
                data = line.lower().rstrip('\n').split('\t')
                for key in header_indices.iterkeys():
                    header_indices[key] = data.index(key.lower())
                    header = False
                else:
                    data = line.rstrip('\n').split('\t')
                    if not status or data[header_indices[self.stat_header]].lower() == status:
                        maf_set.add(tuple(self.info_generator(data, header_indices)))
    return maf_set
```

Future Changes



- Implement Confluence API to generate tables
  - Migrate most of dashboard to gdac.broadinstitute.org
- Run table generation as a cron job after nightly mirror/dice
  - Include change notification
- We need more stringent enforcement of MAF standards TCGA-wide in order to implement full automation like we have with other data types. Otherwise, there will continue to be curation issues. Specifically, we should have:
  - Matching WIGs
  - Working SDRFs
  - Standardized Versioning
  - Enforced AWG submission