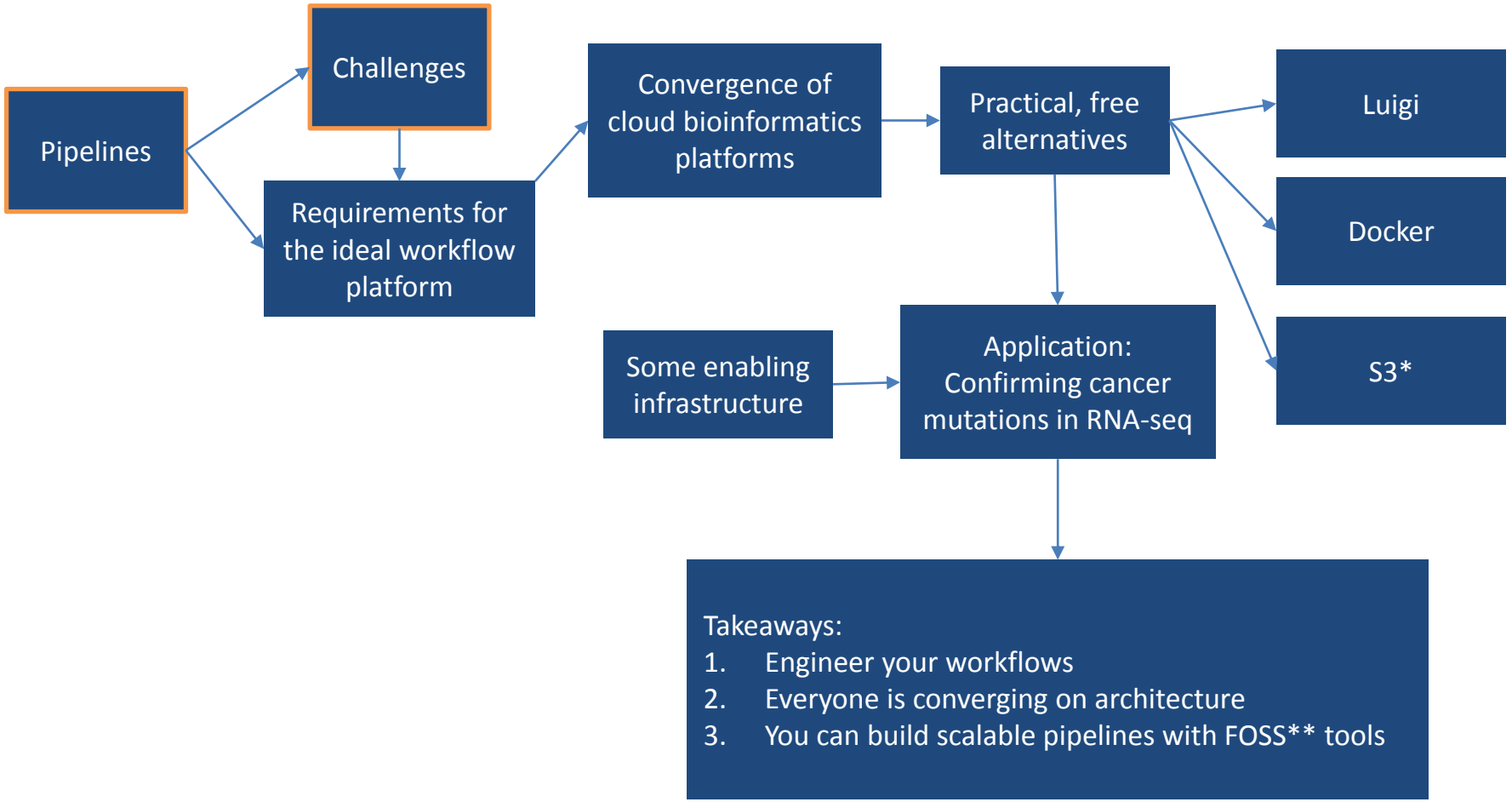
Building cloud-enabled cancer genomics workflows with Luigi and Docker

Jake Feala, PhD
Principal Scientist, Bioinformatics @ Caperna
Founder @ Outlier Bio

Building cloud-enabled cancer genomics workflows with Luigi and Docker

Jake Feala, PhD
Principal Scientist, Bioinformatics @ Caperna
Founder @ Outlier Bio

Outline

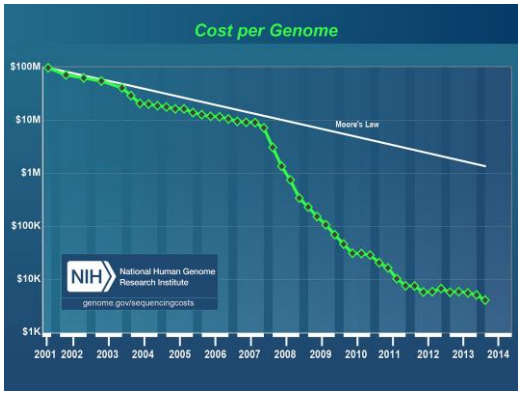


*Not technically free but nearly
**FOSS = Free and Open Source Software

data >> algorithms

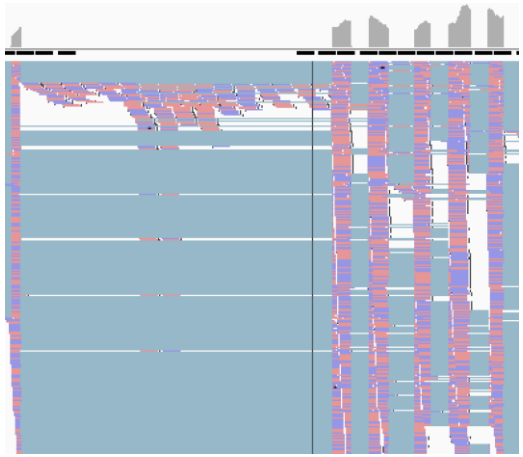
When in doubt, just sequence it!

Plummeting sequencing costs



+

Powerful insights

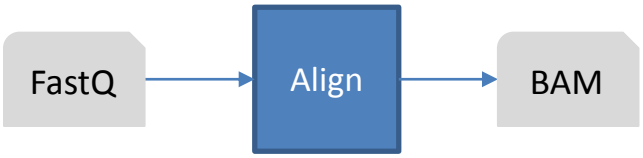


= **Mounds of data**

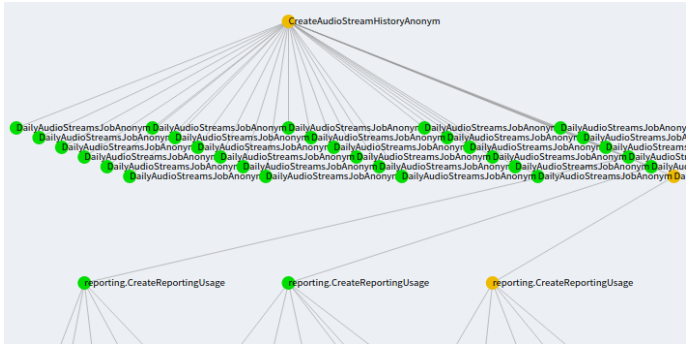
managing big genomics data is hard

Pipelines

What you show your boss



What you actually do



Build index

Align each lane

Merge SAM

SAM → BAM

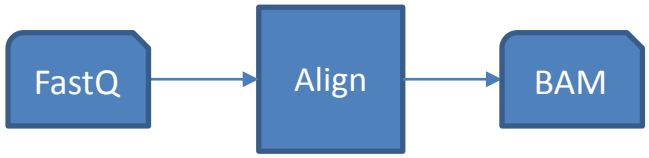
Sort BAM

Index BAM

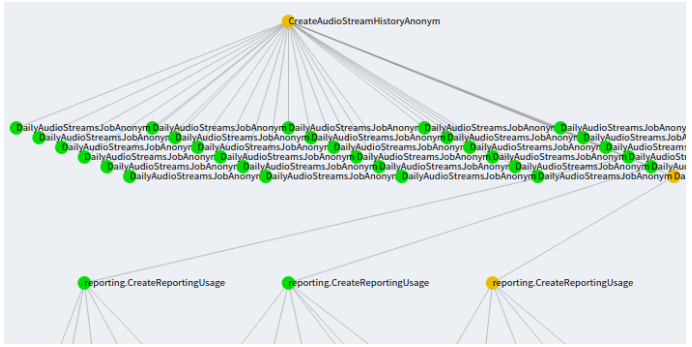
Update IGV registry

Pipelines

What you show your boss



What you actually do



Build index

Align each lane

Merge SAM

SAM → BAM

Sort BAM

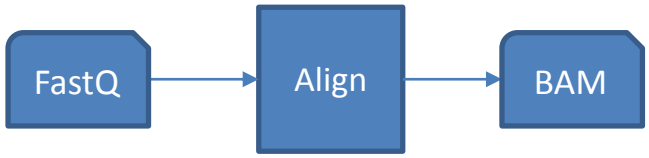
Index BAM

Update IGV registry

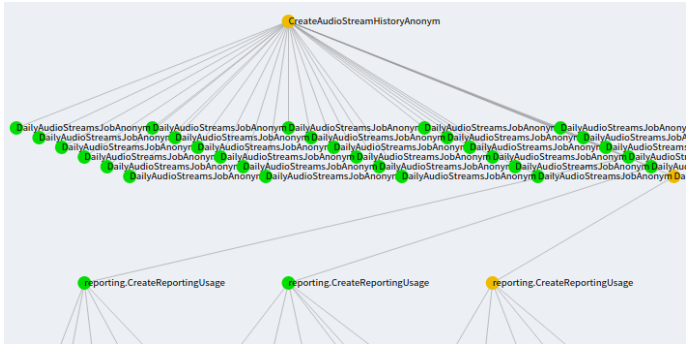
And sometimes….

Pipelines

What you show your boss



What you actually do



Build index

Align each lane

Merge SAM

SAM → BAM

Sort BAM

Index BAM

Update IGV registry

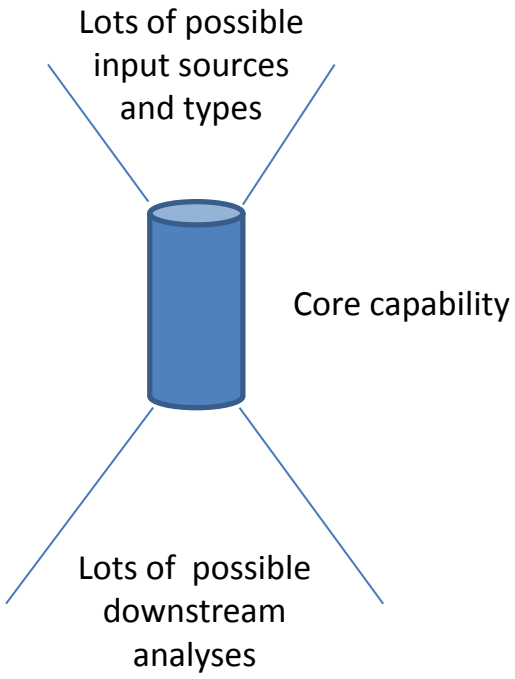
And sometimes….

And BTW these files are massive

```
 38G Mar  3 10:02 tumor.bam
9.7G Mar  2 14:23 Aligned.Sorted.bam
 23G Mar  2 14:24 Aligned_Recal.Sorted.bam
```

Capability = automated pipeline*

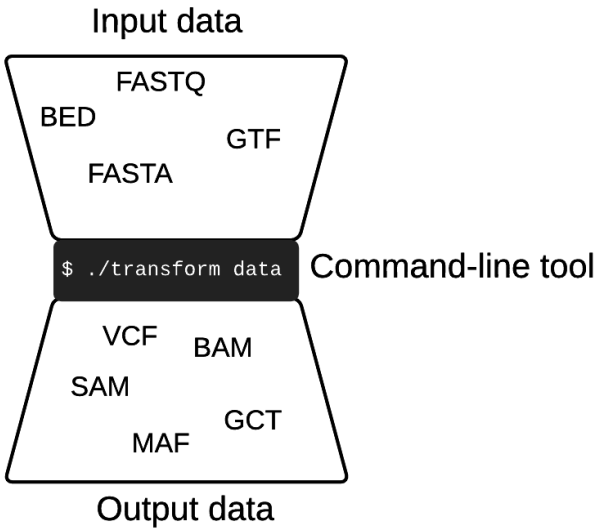
- **Pipelines are at the core of a bioinformatics group**
- Best way to show a capability is a complete pipeline
  - Any suitable input → quality-assured output
  - Sure, you may be able to do a certain analysis, but a working, tested pipeline proves it
- Lots of tools are available, but putting them together correctly into a pipeline is **hard**



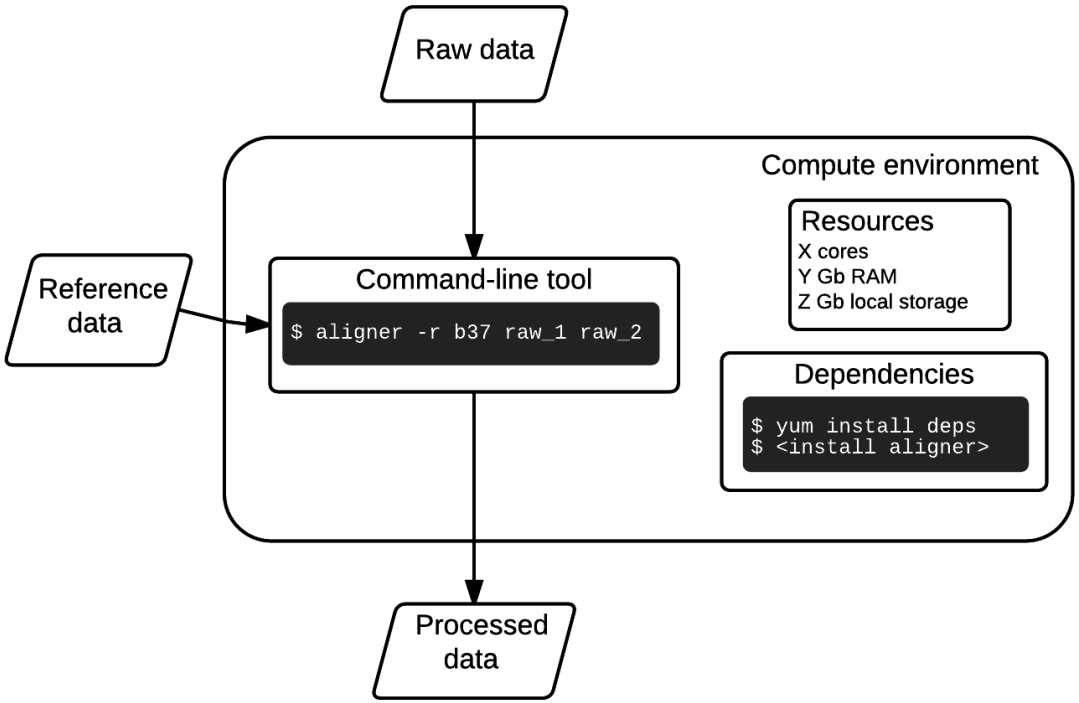
*"workflow" is synonymous with "pipeline" for the purposes of this talk

Linux commands and custom file formats are the “narrow waist” of bioinformatics

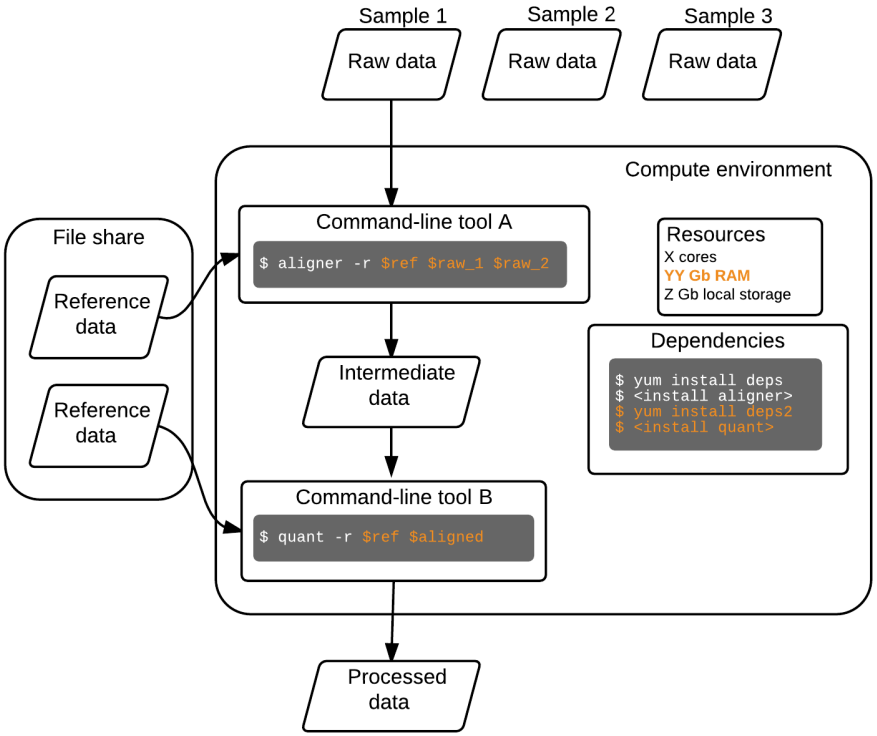
This constraint underlies the design of bioinformatics pipelines



This works great for tool developers…



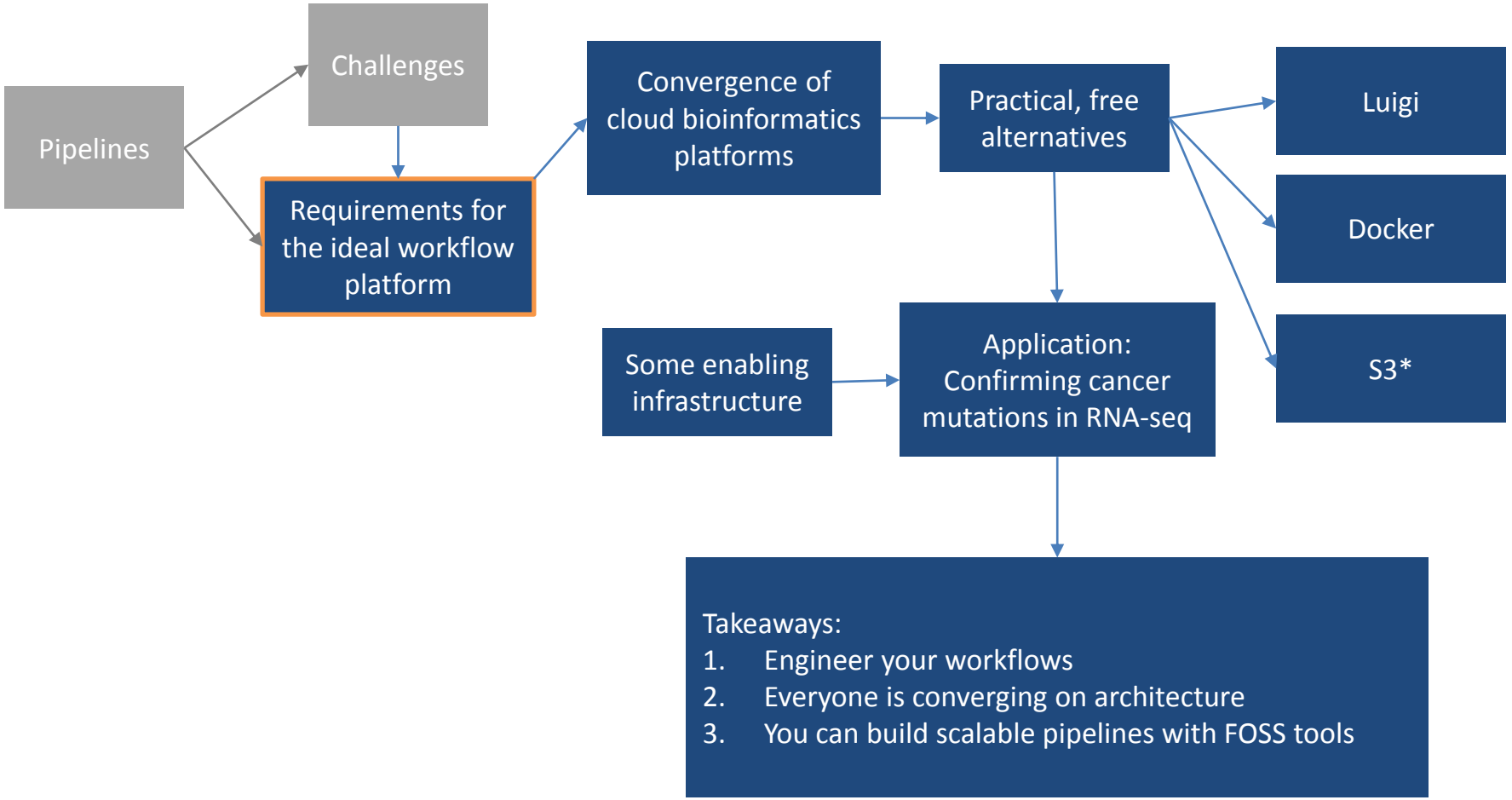
…but quickly gets complicated for users



Common homegrown solution:

- Bash scripts
- VMs (AMIs)
- StarCluster or similar
- SunGrid Engine or similar
- Parallelize by sample

Outline



*Not technically free but nearly
**FOSS = Free and Open Source Software

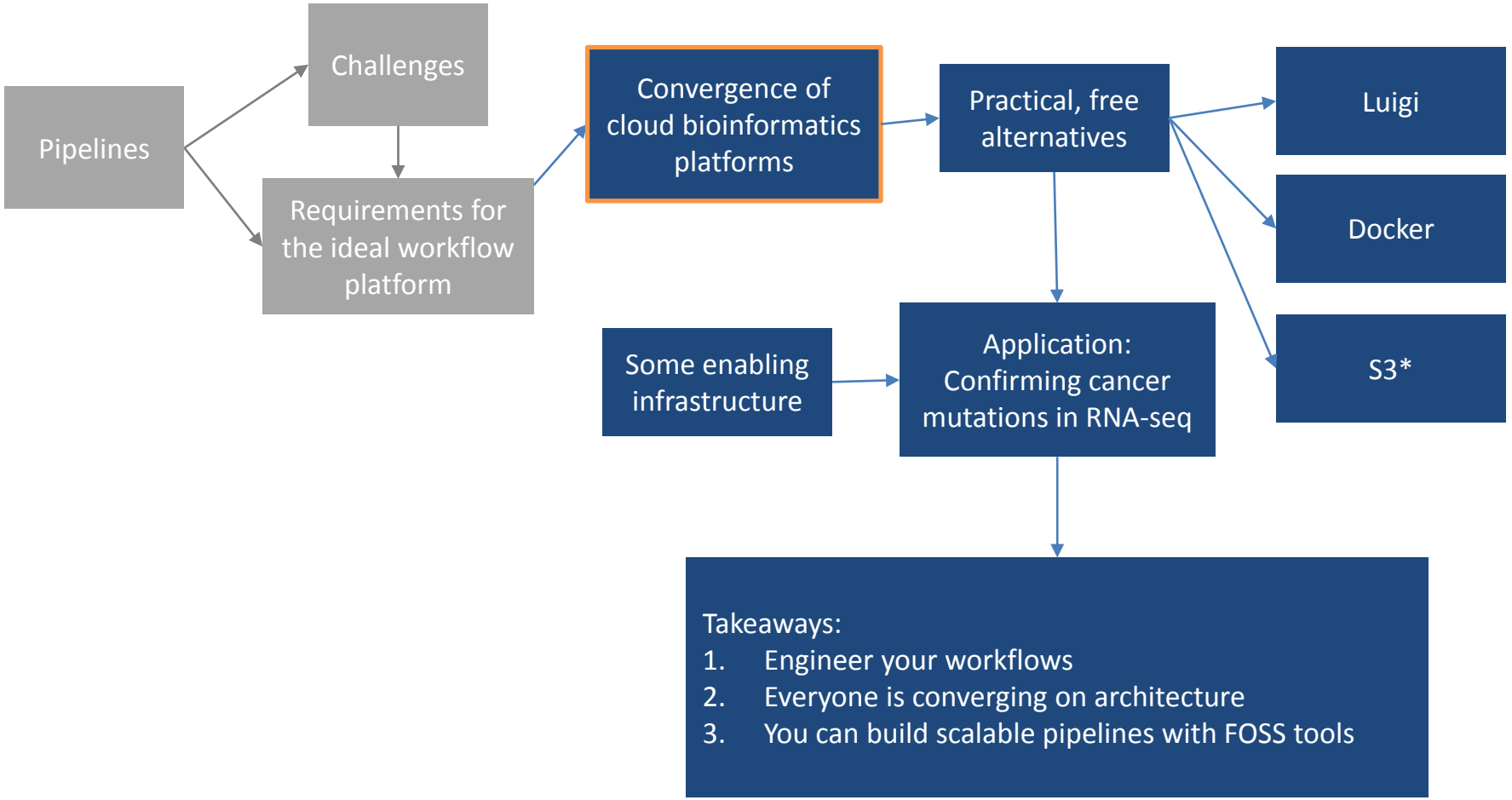
Limitations with commonly used technology

- Compute
  - **SGE** is flat, just a job scheduler, bash only, no dependencies
- Environment
  - **AMIs and VMs** are slow and heavyweight, no code, not good for ongoing dev
- Workflow
  - **Makefiles** have ugly syntax, static, inflexible
  - **GUIs and domain-specific systems** like Galaxy/Taverna are not easily programmable or general-purpose

Properties of the ideal pipeline system

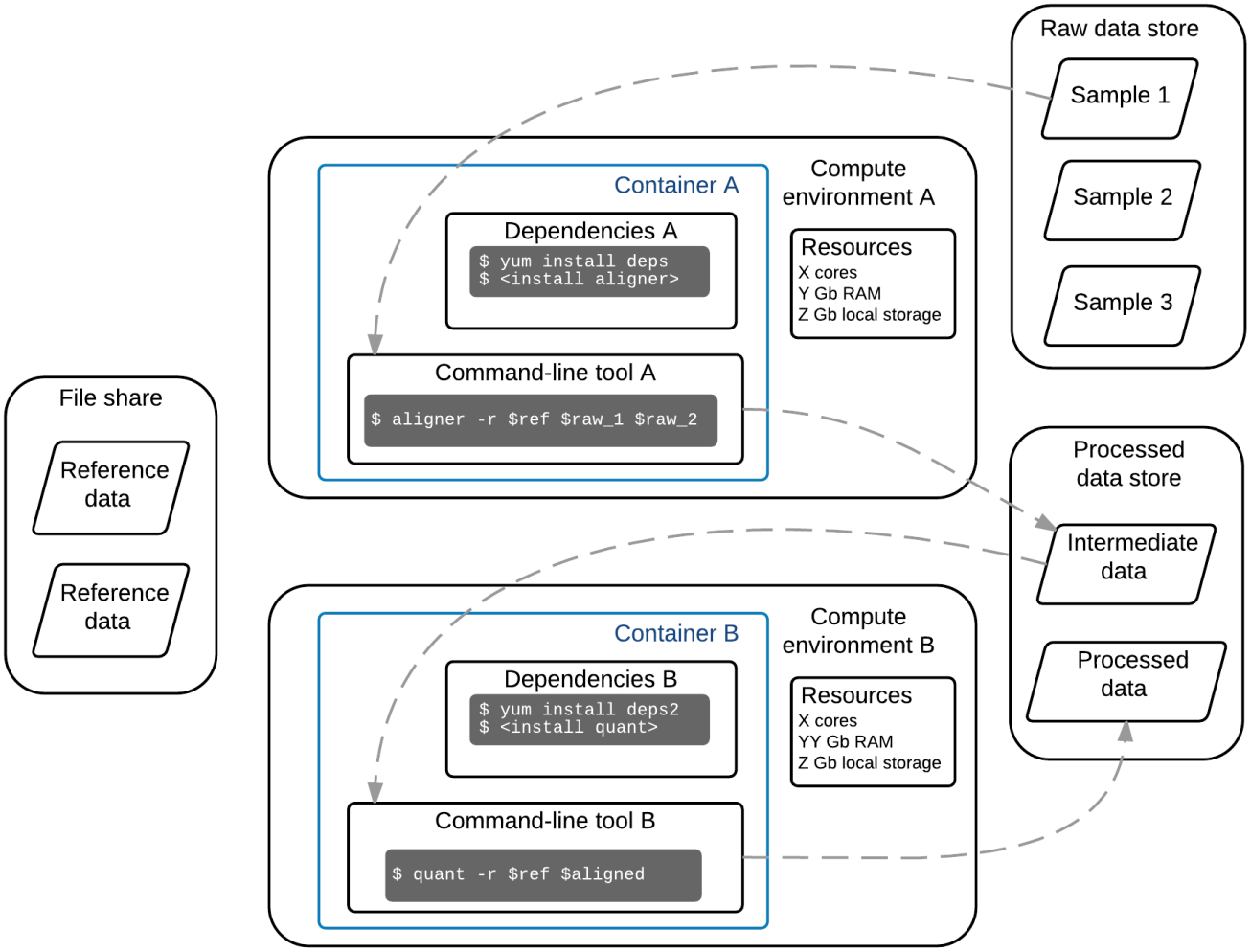
- General purpose: familiar language, can apply to any task
- Modular: any language, well-tested components with tight APIs
- Scalable: parallelize for free, independent of components
- Integrated: LIMS, metadata, viz, versioning, reporting
- Versioned: reproduce from snapshots in time
- Idempotent: resume from failure, guarantee outputs

Outline

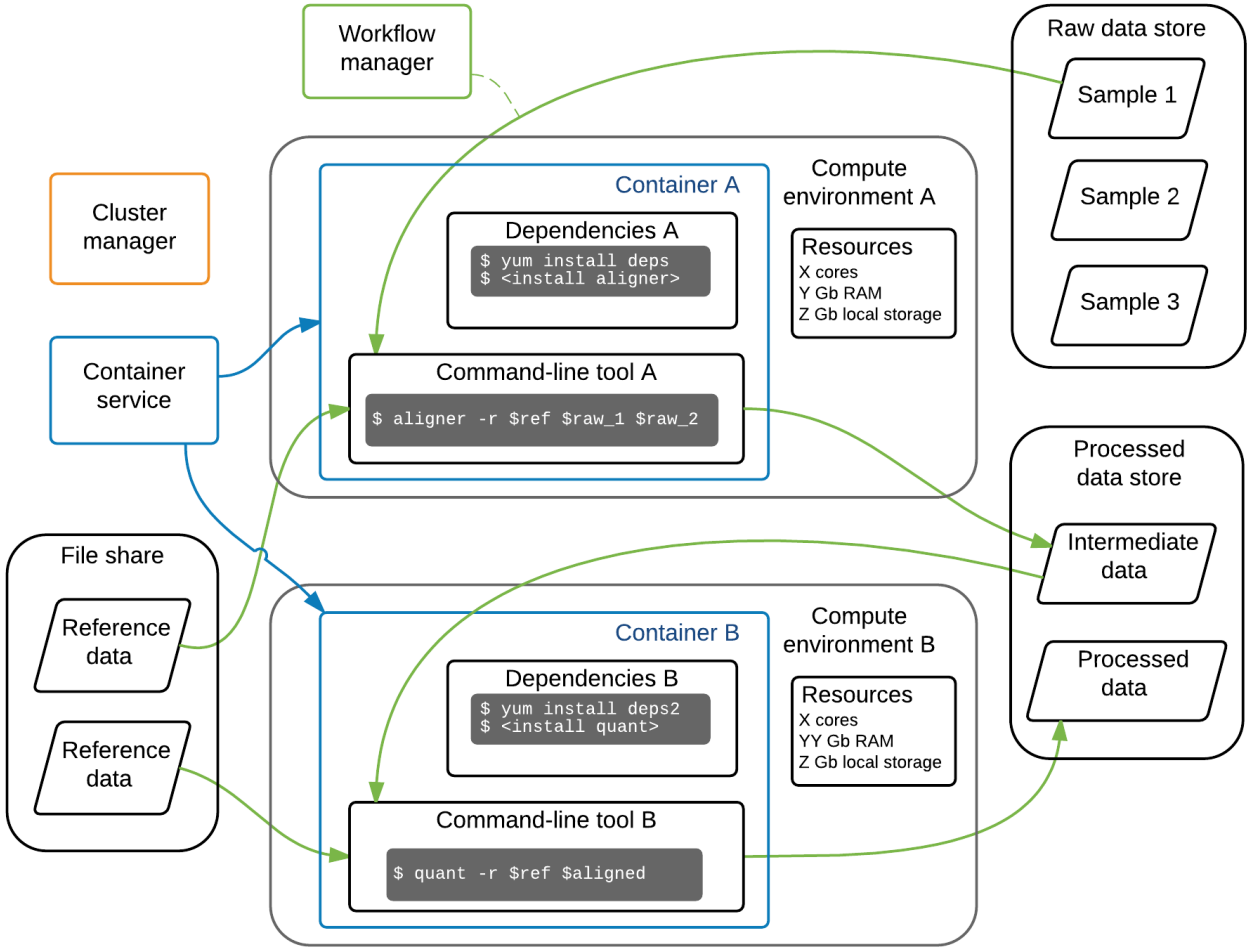


*Not technically free but nearly
**FOSS = Free and Open Source Software

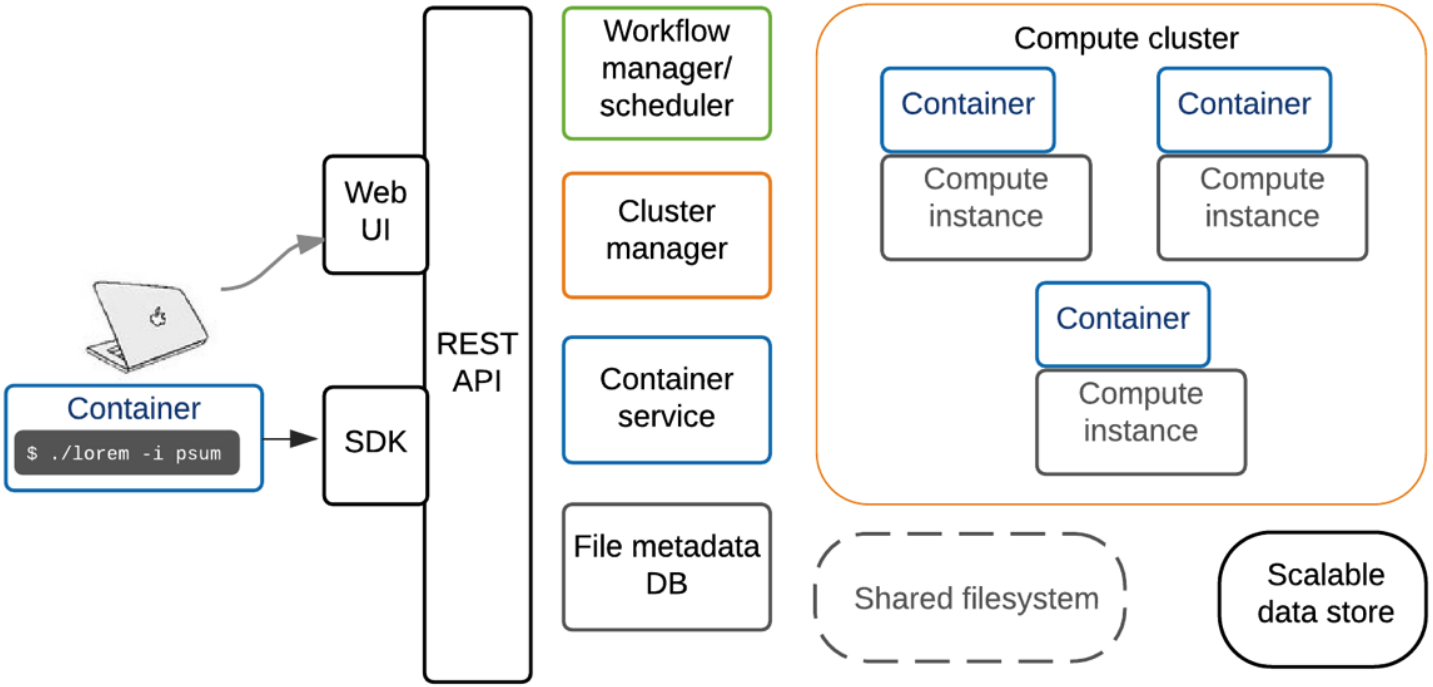
Emerging best practice is to containerize and decompose into atomic steps



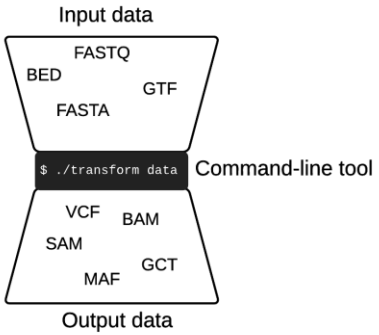
But atomicity and scaling bring added complexity

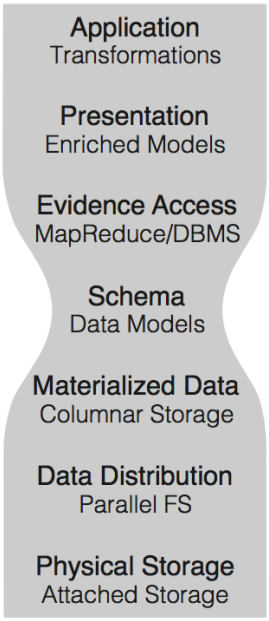


Everyone is doing this same basic architecture



That is, everyone except for ADAM/Spark. They are shifting the paradigm



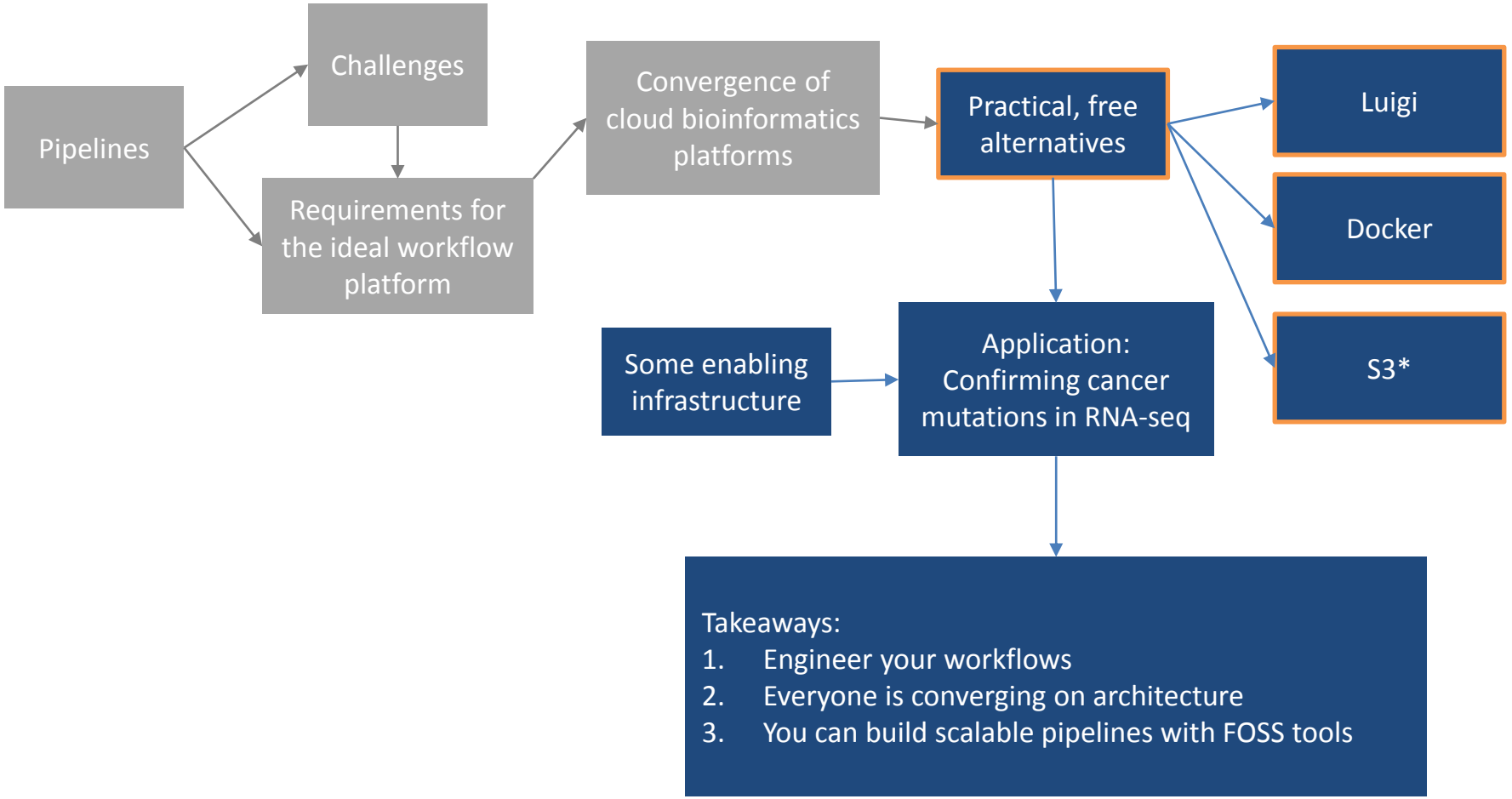


- Pipeline = sequence of transformations on a data model, not a string of shell commands
- Separate method from implementation
- Distribute records (i.e. alignments) for horizontal scaling
- Needs more community tools before adoption

Fig 1, Nothaft et al. (2015) Rethinking Data-Intensive Science Using Scalable Analytics Systems

https://bdgenomics.org

Outline



*Not technically free but nearly
**FOSS = Free and Open Source Software

A free alternative architecture

Components

- **Storage**: S3
- **Environment**: Docker
- **Workflows**: Luigi
- **Compute**: EC2 auto-scaling groups
- **Infrastructure**: to put the pieces together

Benefits

- Free (almost)
- Huge communities, battle tested
- Separate concerns, use best tool for each job

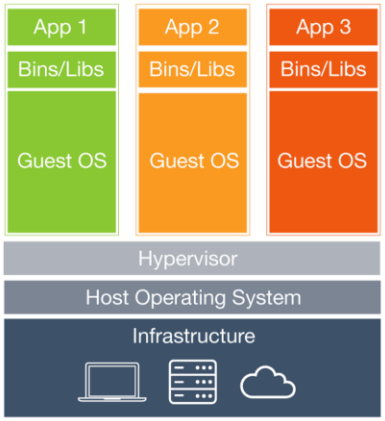
S3 and object storage

Steps:

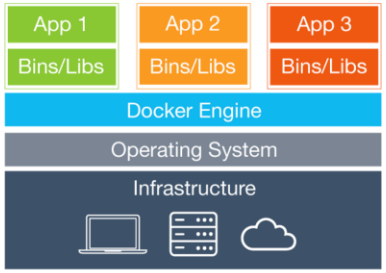
1. Learn how object store is different from a filesystem
2. Write a bit of extra code to manage transfers to/from S3
3. Never think about long-term storage again*

*within reason, if IT director is in the room.

Docker



Virtual Machines



Containers

- **Lightweight** environment management
- Specify environment as code (Dockerfile)
- **Portable** (laptop → cluster with same behavior)
- **Wide adoption** in tech
- **Gaining ground** in bioinformatics

From https://www.docker.com/what-docker

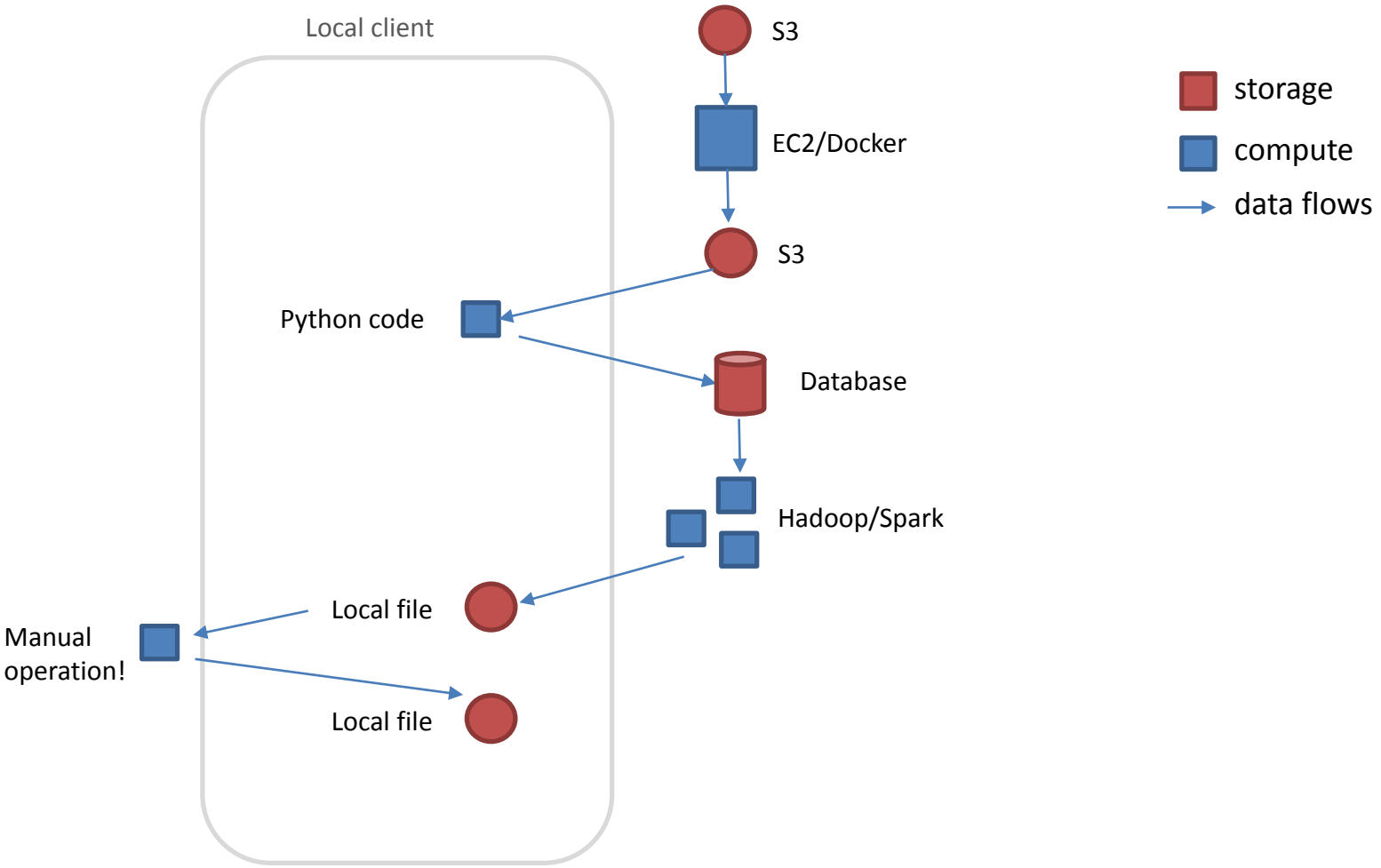
Luigi

- Like a Makefile but in pure Python*
- Executes **locally** => easy to debug
- OOP declarative framework with **Tasks and Targets**
  - Tasks are Python objects, they can do anything
  - Targets are Python objects, they can be anything
- **Idempotent** (resume where you left off) and **atomic** (no half-finished tasks)
- **Batteries included** (graph viz, CLI integration, S3, MySQL, Hadoop, Spark, Redshift, SGE, …

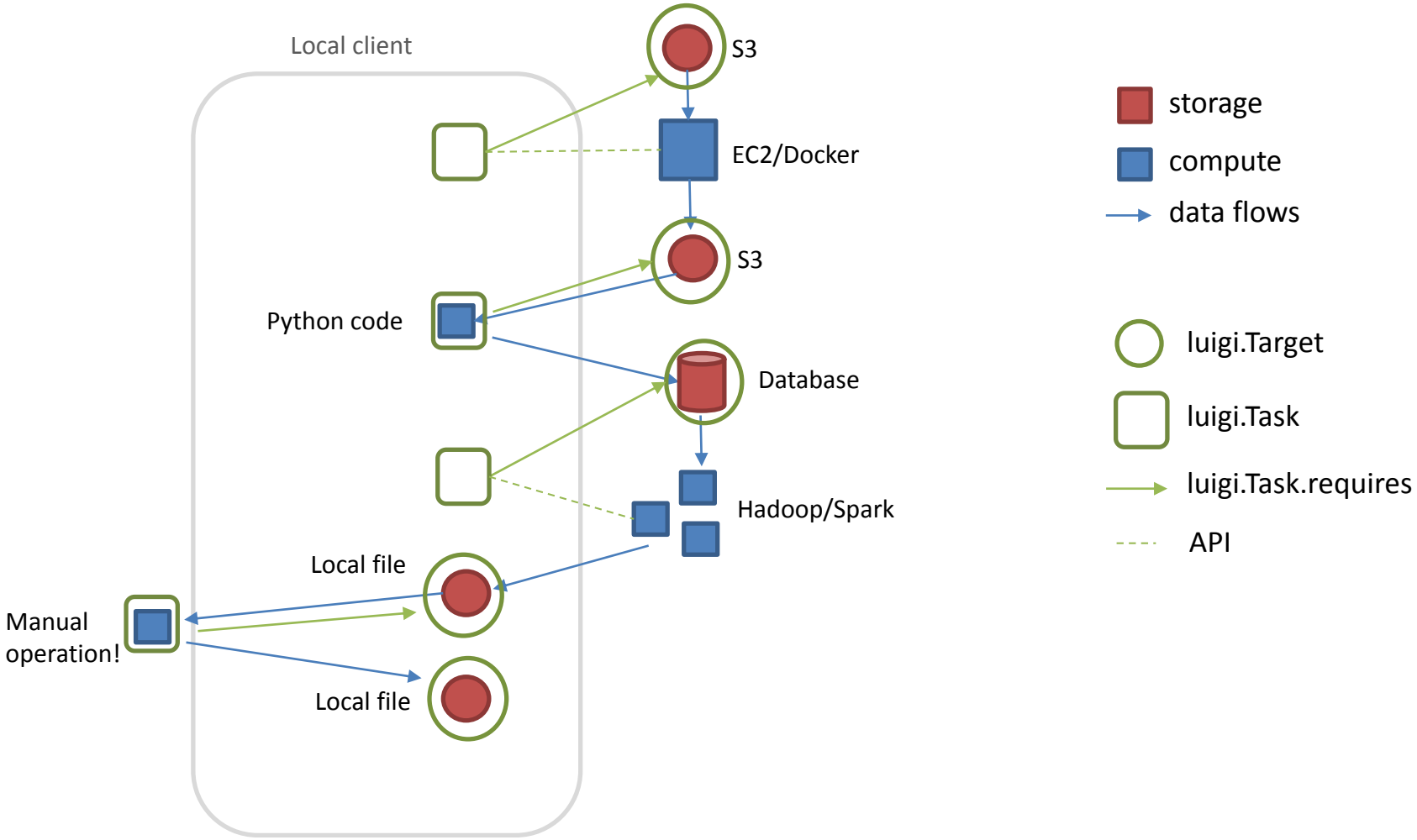
What's a Makefile?
- Build DAG of tasks
- Each task specifies dependencies and output
- Start with what you want to **build**, work up

*i.e., not another DSL

Luigi is flexible enough to do pretty much anything within a workflow



Luigi is flexible enough to do pretty much anything within a workflow



Anatomy of a Luigi Task

```python
class MyTask(luigi.Task):

    sample_id = luigi.Parameter()
    alg_param = luigi.IntParameter(default=42)

    def requires(self):
        return AnotherTask(sample_id=self.sample_id)

    def run(self):
        with self.input().open('w') as f:
            data = f.read()

        output = run_algorithm(data, param=self.alg_param)

        with self.output().open('w') as fw:
            fw.write(output)

    def output(self):
        filepath = os.path.join(DATA_DIR, self.sample_id, 'output.txt')
        return luigi.LocalTarget(filepath)
```

Anatomy of a Luigi Task

```python
class MyTask(luigi.Task):

    sample_id = luigi.Parameter()
    alg_param = luigi.IntParameter(default=42)

    def requires(self):
        return AnotherTask(sample_id=self.sample_id)

    def run(self):
        with self.input().open('w') as f:
            data = f.read()

        output = run_algorithm(data, param=self.alg_param)

        with self.output().open('w') as fw:
            fw.write(output)

    def output(self):
        filepath = os.path.join(DATA_DIR, self.sample_id, 'output.txt')
        return luigi.LocalTarget(filepath)
```

Declare your parameters (can be passed from CLI)

Read the input

Do the thing

Write the output

Build your filepaths in Task.output

Anatomy of a Luigi Task

```python
class MyTask(luigi.Task):

    sample_id = luigi.Parameter()
    alg_param = luigi.IntParameter(default=42)

    def requires(self):
        return AnotherTask(sample_id=self.sample_id)

    def run(self):
        with self.input().open('w') as f:
            data = f.read()

        output = run_algorithm(data, param=self.alg_param)

        with self.output().open('w') as fw:
            fw.write(output)

    def output(self):
        s3_path = os.path.join(DATA_BUCKET, self.sample_id, 'output.txt')
        return luigi.S3Target(s3_path)
```

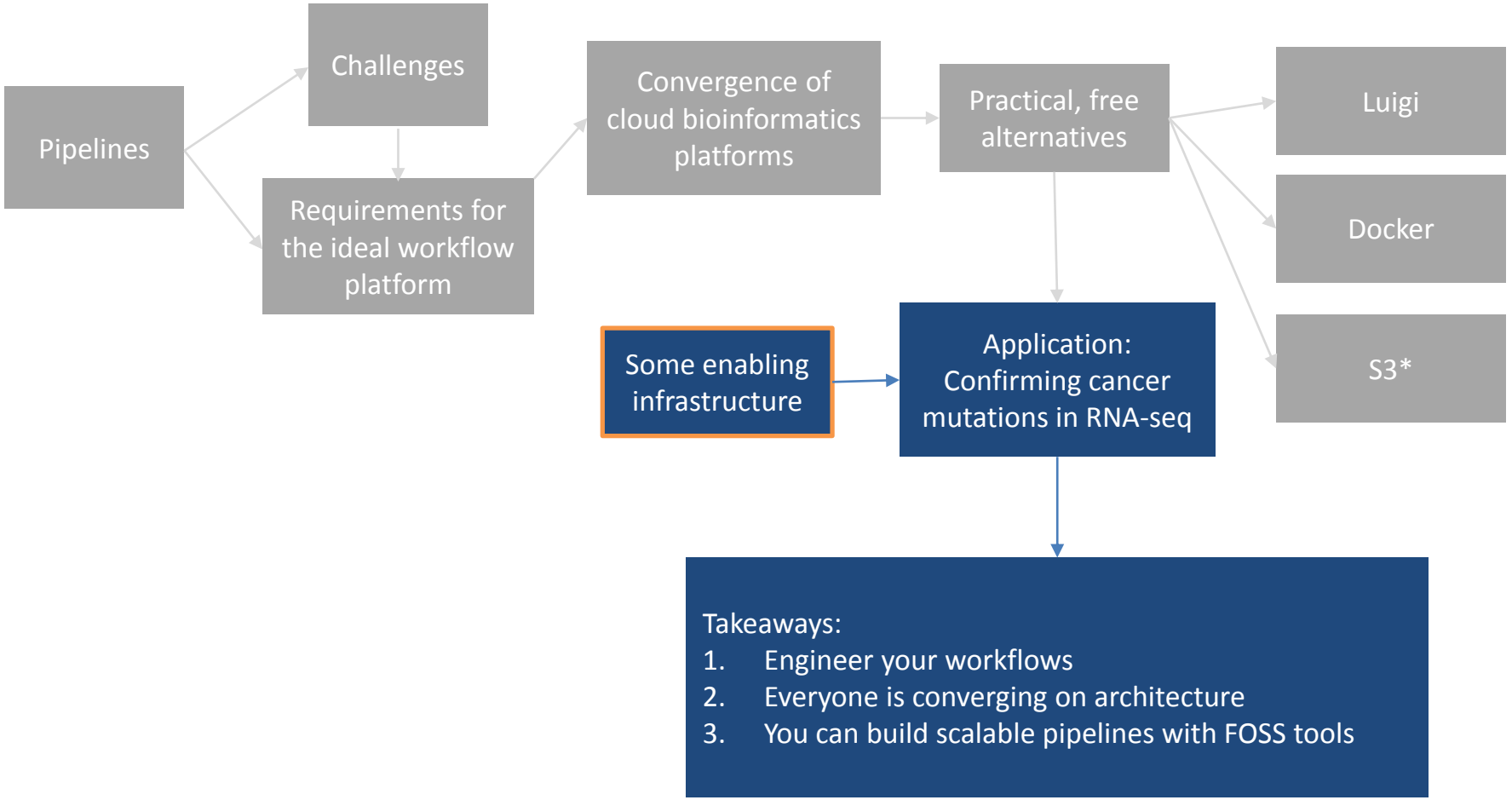
Now output to S3 instead

Anatomy of a Luigi Task

```
class MyS3DockerTask(luigi.Task):

    sample_id = luigi.Parameter()
    alg_param = luigi.IntParameter(default=42)

    def requires(self):
        return AnotherS3Task(sample_id=self.sample_id)

    def run(self):

        template = """
            aws s3 cp {input} /data/local_input
            docker pull myimage
            docker run myimage run-algorithm --param={param} /scratch/local_input > /scratch/local_output
            aws s3 cp local_output {output}
        """
        cmd = template.format(
            input=self.input().path, param=self.alg_param, output=self.output().path)
        subprocess.check_output(cmd)

    def output(self):
        s3_path = os.path.join(DATA_BUCKET, self.sample_id, 'output.txt')
        return luigi.S3Target(s3_path)
```

Run a Docker container instead

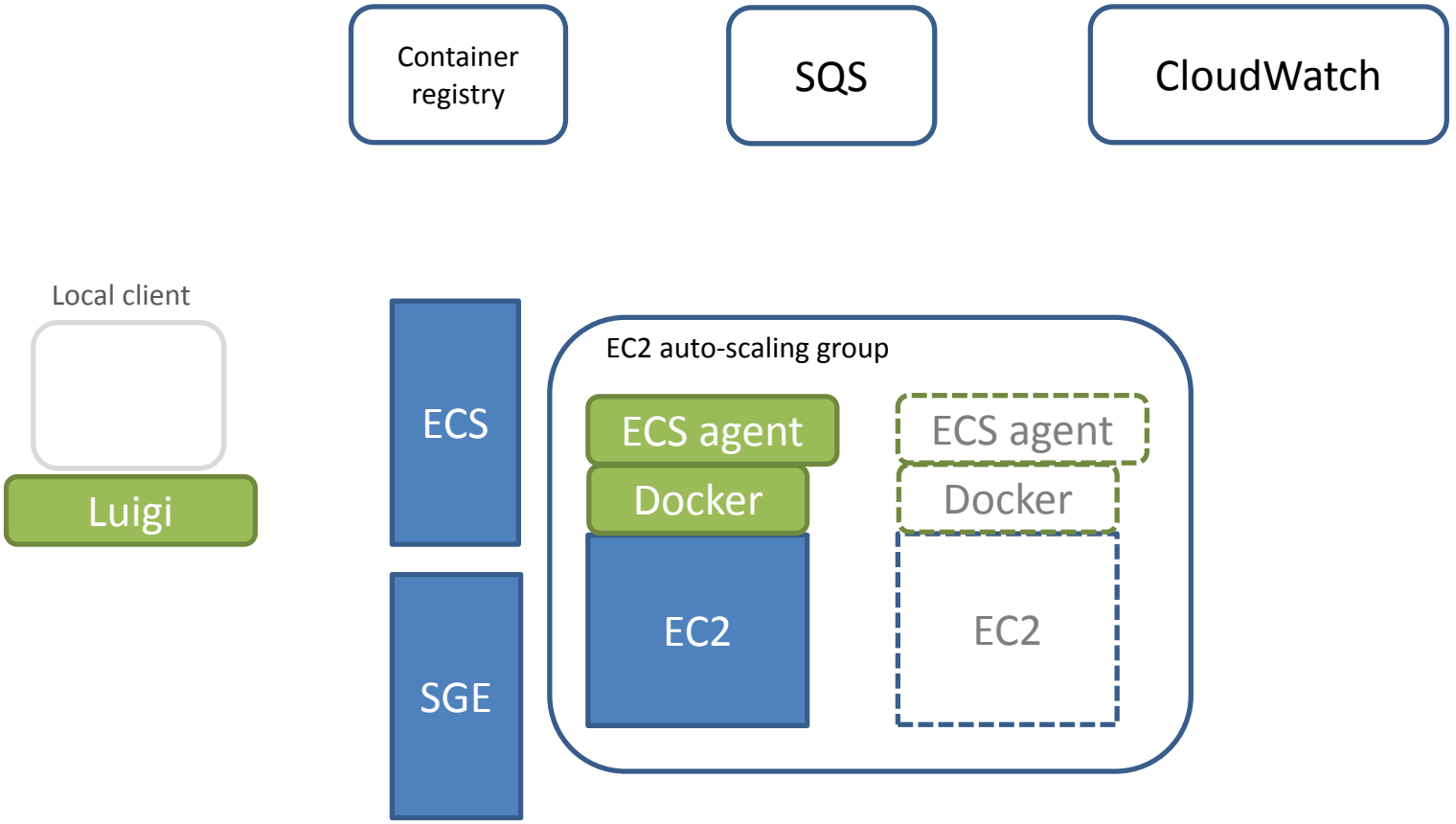
Outline



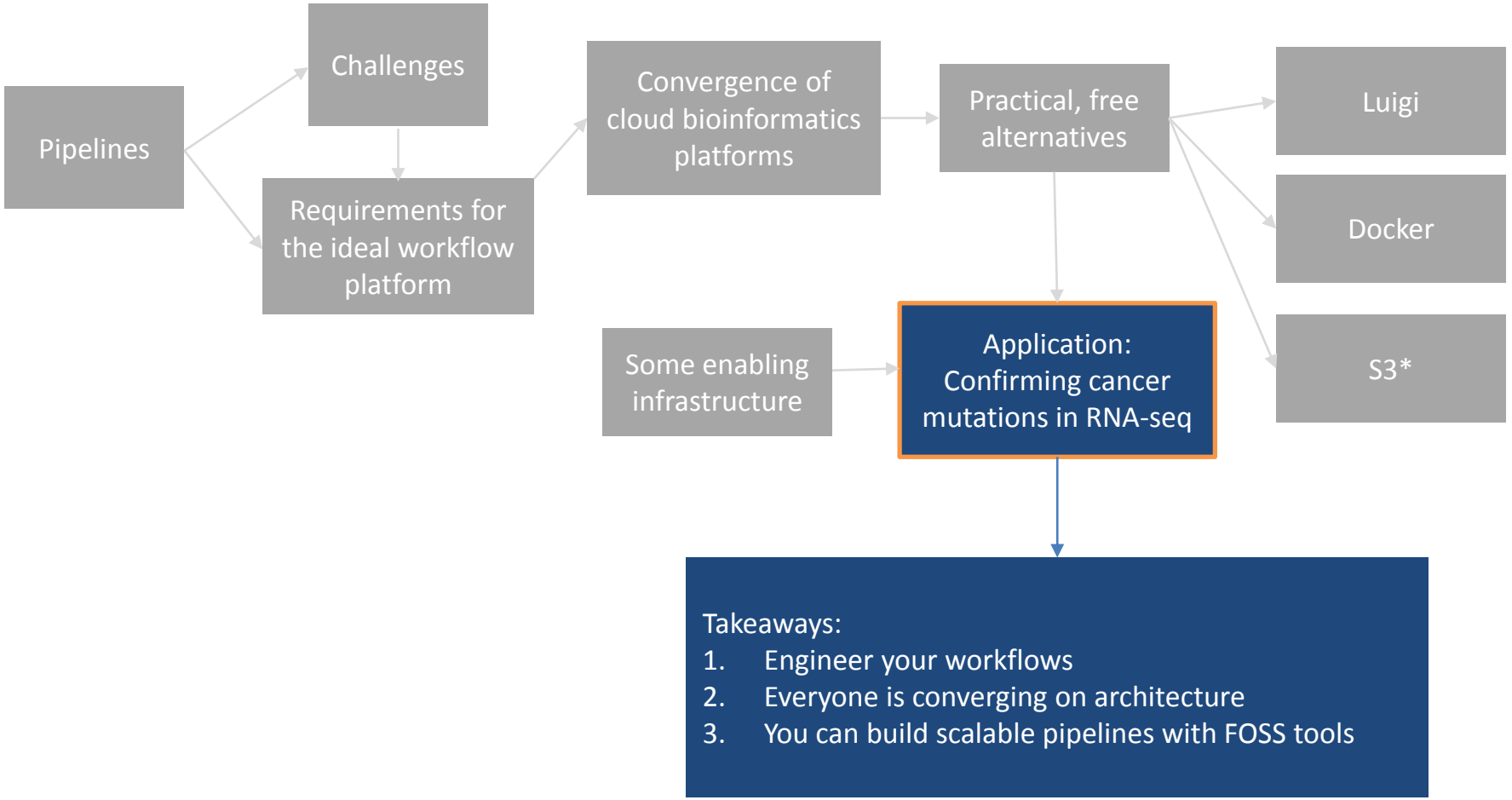
*Not technically free but nearly
**FOSS = Free and Open Source Software

Some assembly required

But lots of ways to do it!



Outline



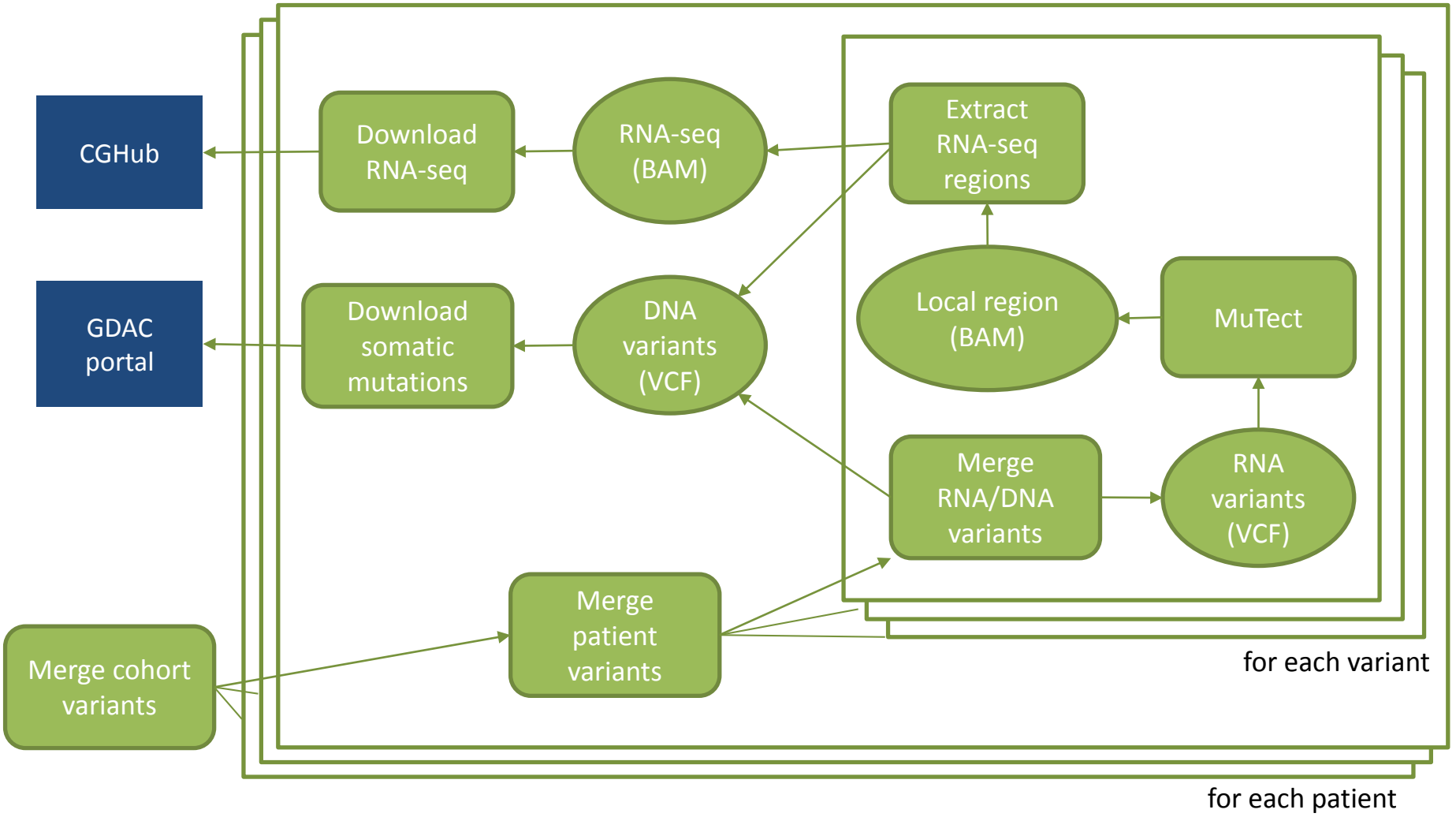
*Not technically free but nearly
**FOSS = Free and Open Source Software

An interesting* cancer genomics application

- Problem:
  - Calling somatic cancer mutations is difficult
    - Low allele frequencies
    - Complex variants
    - Purity, ploidy issues
  - RNA-seq can add confidence, but
    - TCGA does not provide RNA-seq variants
    - Manual review is slow
    - Automating and scaling requires a complex, custom workflow

*to me

Solution: RNA-seq mutation validation pipeline



Note: this workflow should soon be feasible with the NCI cloud pilot platforms

Dockerizing the apps

Base image ./Dockerfile

```
FROM centos:7.0.1406
MAINTAINER Jake Feala <jake@outlierbio.com>

# Yum and pip packages
RUN yum update -y && yum install -y bzip2 gcc gcc-c++ make tar wget
RUN curl "https://bootstrap.pypa.io/get-pip.py" -o "get-pip.py" && \
    python get-pip.py
RUN pip install --upgrade pip && pip install awscli boto3 click ipython

# Add shared utilities
ADD . /bio-it/
RUN cd /bio-it && pip install .

# Add minimal credentials for application user
ADD ./secrets/aws/* /root/.aws/
ENV AWS_PROFILE app_user
```

```
$ docker build –t outlierbio/bioit .
```

GTDownload ./bioit/apps/gtdownload/Dockerfile

```
FROM outlierbio/bioit

# Set up GeneTorrent
RUN cd /opt && \
  wget https://cghub.ucsc.edu/software/downloads/GeneTorrent/3.8.7/
  GeneTorrent-download-3.8.7-207-CentOS6.4.x86_64.tar.gz && \
  tar -xvf GeneTorrent-download-3.8.7-207-CentOS6.4.x86_64.tar.gz && \
  rm GeneTorrent-download-3.8.7-207-CentOS6.4.x86_64.tar.gz
ENV PATH $PATH:/opt/cghub/bin
```

```
$ cd bioit/apps/gtdownload
$ docker build –t outlierbio/gtdownload .
```

Samtools ./bioit/apps/samtools/Dockerfile

```
FROM outlierbio/bioit

RUN yum install -y libcurses zlib-devel

# Install samtools
ADD https://github.com/samtools/samtools/releases/download/1.3/samtools-1.3.tar.bz2 /
RUN tar xvjf samtools-1.3.tar.bz2 && \
  cd /samtools-1.3 && \
  make && \
  mv /samtools-1.3/samtools /usr/local/bin/
```

```
$ cd ../samtools
$ docker build –t outlierbio/samtools .
```

See https://github.com/outlierbio/bio-it for full project

Luigi-izing the workflow

```
class GTDownload(luigi.Task):

    analysis_id = luigi.Parameter()

    def run(self):
        cmd = '''docker run --rm -v /mnt/scratch:/scratch outlierbio/gtdownload \
                    "gtdownload -c {keypath} -p /scratch --max-children 8 -v {analysis_id} && \
                     aws s3 cp {src} {dst} --recursive"
                '''.format(keypath=CGHUB_PUBLIC_KEY, analysis_id=self.analysis_id,
                           src='/scratch/' + self.analysis_id, dst=self.output().path)
        out = subprocess.check_output(cmd, shell=True)

    def output(self):
        s3_path = 's3://outlierbio-data-raw/tcga/{analysis_id}/{analysis_id}.bam'.format(
            analysis_id=self.analysis_id)
        return luigi.s3.S3Target(s3_path)
```

See https://github.com/outlierbio/bio-it for full project

Luigi-izing the workflow

```python
class GTDownload(luigi.Task):

    analysis_id = luigi.Parameter()

    def run(self):
        cmd = '''docker run --rm -v /mnt/scratch:/scratch outlierbio/gtdownload \
                    "gtdownload -c {keypath} -p /scratch --max-children 8 -v {analysis_id} && \
        ou

    def ou
        s3

        re
```

```python
class ExtractRegion(luigi.Task):

    analysis_id = luigi.Parameter()
    region = luigi.Parameter()

    def requires(self):
        return GTDownload(analysis_id=self.analysis_id)

    def run(self):
        cmd = '''docker run --rm -v /mnt/scratch:/scratch outlierbio/samtools \
                    "aws s3 cp {src} {local} && \
                     samtools view -bh {local} {region} > {out} && \
                     aws s3 cp {out} {dst}"
                '''.format(src=self.input().path, local='/scratch/tmp.bam', region=self.region,
                           out='/scratch/out.bam', dst=self.output().path)
        out = subprocess.check_output(cmd, shell=True)

    def output(self):
        s3_path = 's3://outlierbio-data/tcga/{analysis_id}/{region}.bam'.format(
            analysis_id=self.analysis_id, region=self.region.replace(':', '_'))
        return luigi.s3.S3Target(s3_path)
```

See https://github.com/outlierbio/bio-it for full project

Luigi-izing the workflow

```
class GTDownload(luigi.Task):

    analysis_id = luigi.Parameter()

    def run(self):
        cmd = '''docker run --rm -v /mnt/scratch:/scratch outlierbio/gtdownload \
                    "gtdownload -c {keypath} -p /scratch --max-children 8 -v {analysis_id} && \
        ou

    def ou
        s3

        re
```

```
class ExtractRegion(luigi.Task):

    analysis_id = luigi.Parameter()
    region = luigi.Parameter()

    def requires(self):
        return GTDown

    def run(self):
        cmd = '''docke
                    "a
                     s
                     a
                ''' .forr

        out = subproce

    def output(self):
        s3_path = 's3
            analysis_
        return luigi.s
```

```
class RunSample(luigi.WrapperTask):

    sample_id = luigi.Parameter()

    def requires(self):
        for row in maf.iterrows():
            analysis_id = get_analysis_id(self.sample_id)
            region = row['Chromosome', 'Start']
            yield Mutect(analysis_id=analysis_id, region=region)

    def run(self):
        with self.input().open() as f:
            variants = list(vcf.VCFReader(f))

        maf['rna_af'] = maf['genome_change'].map(lambda gc: get_af(gc, variants))

        with self.output().open('w') as f:
            maf.to_csv(f, index=False)

    def output(self):
        return luigi.s3.S3Target(self.input().path.replace('.maf', 'rna_af.maf'))
```

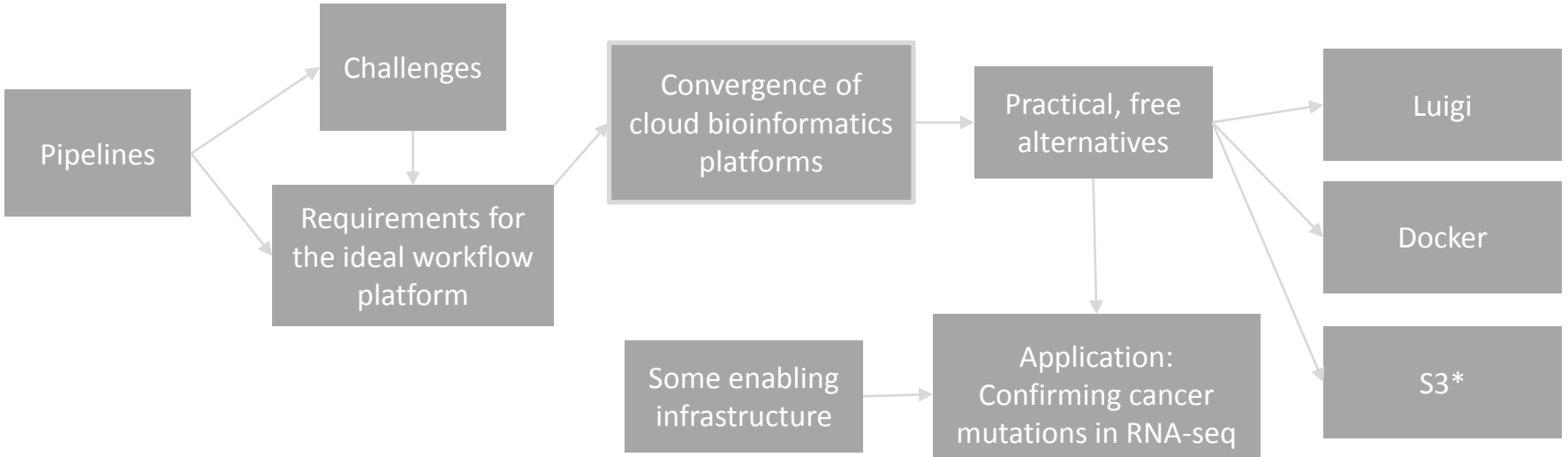
See https://github.com/outlierbio/bio-it for full project

Run it!

```
$ luigi --module bioit.validate_rnaseq RunSample --sample-id=TCGA-AB-2929
```

See https://github.com/outlierbio/bio-it for full project

Thanks! Questions?



Further reading

- My blog: https://medium.com/outlier-bio-blog
- GitHub repo for this talk: https://github.com/outlierbio/bioit
- Luigi: https://github.com/spotify/luigi
- Docker: https://docker.com
- Amazon AWS: http://lmgtfy.com/?q=aws

Takeaways:

1. Engineer your workflows
2. Everyone is converging on architecture
3. You can build scalable pipelines with FOSS tools

*Not technically free but nearly
**FOSS = Free and Open Source Software

More on reproducibility: 3 major components and some possible solutions

- Code
  - VCS
  - Notebooks
- Data
  - Metadata store
  - S3
  - Synapse
  - Versioned data APIs
- Environment
  - Package managers (pip –r)
  - VM
  - Docker

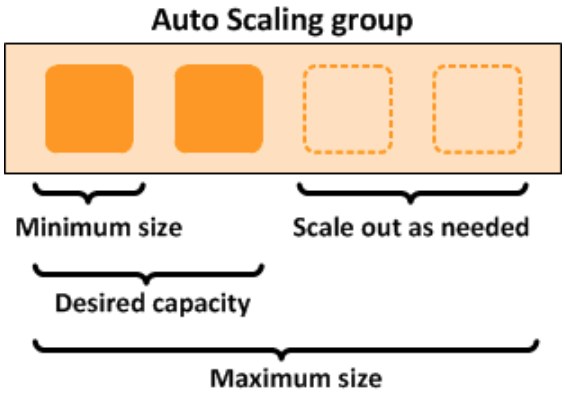
The slide about how this is kind of like functional programming

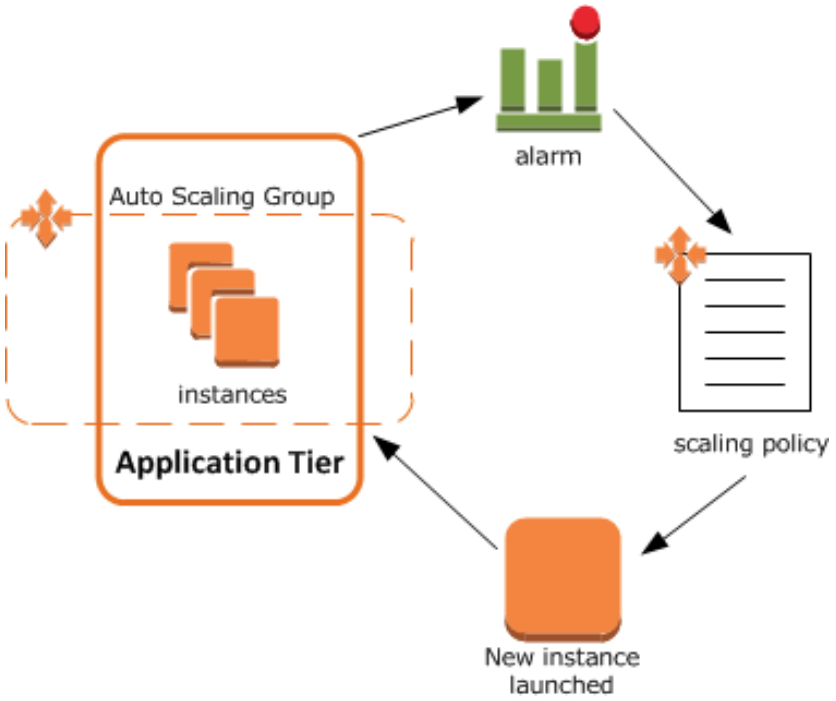
- No side effects (pure)
- Stateless
- Same output for every input
- Compose complex, scalable functionality from small components

Docker (user perspective)

- Specify environment as code (Dockerfile)
- A little like git
  - commits (each with uuid hash)
  - pull
  - push
- Isolated like a VM
- Run like an app
  - Takes arguments
  - CLI allows interactive use
  - Hop inside the container to debug

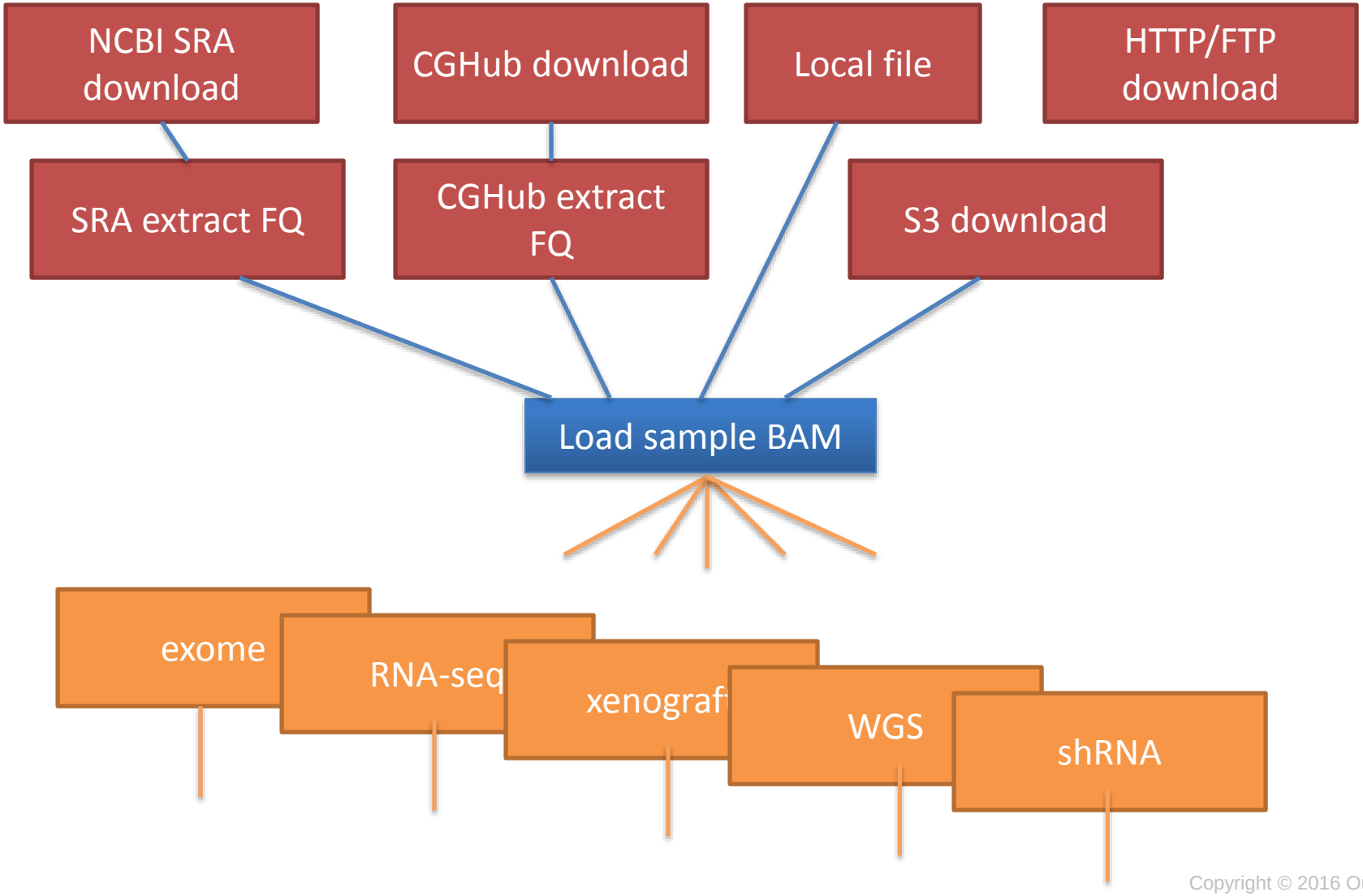
EC2 auto-scaling groups



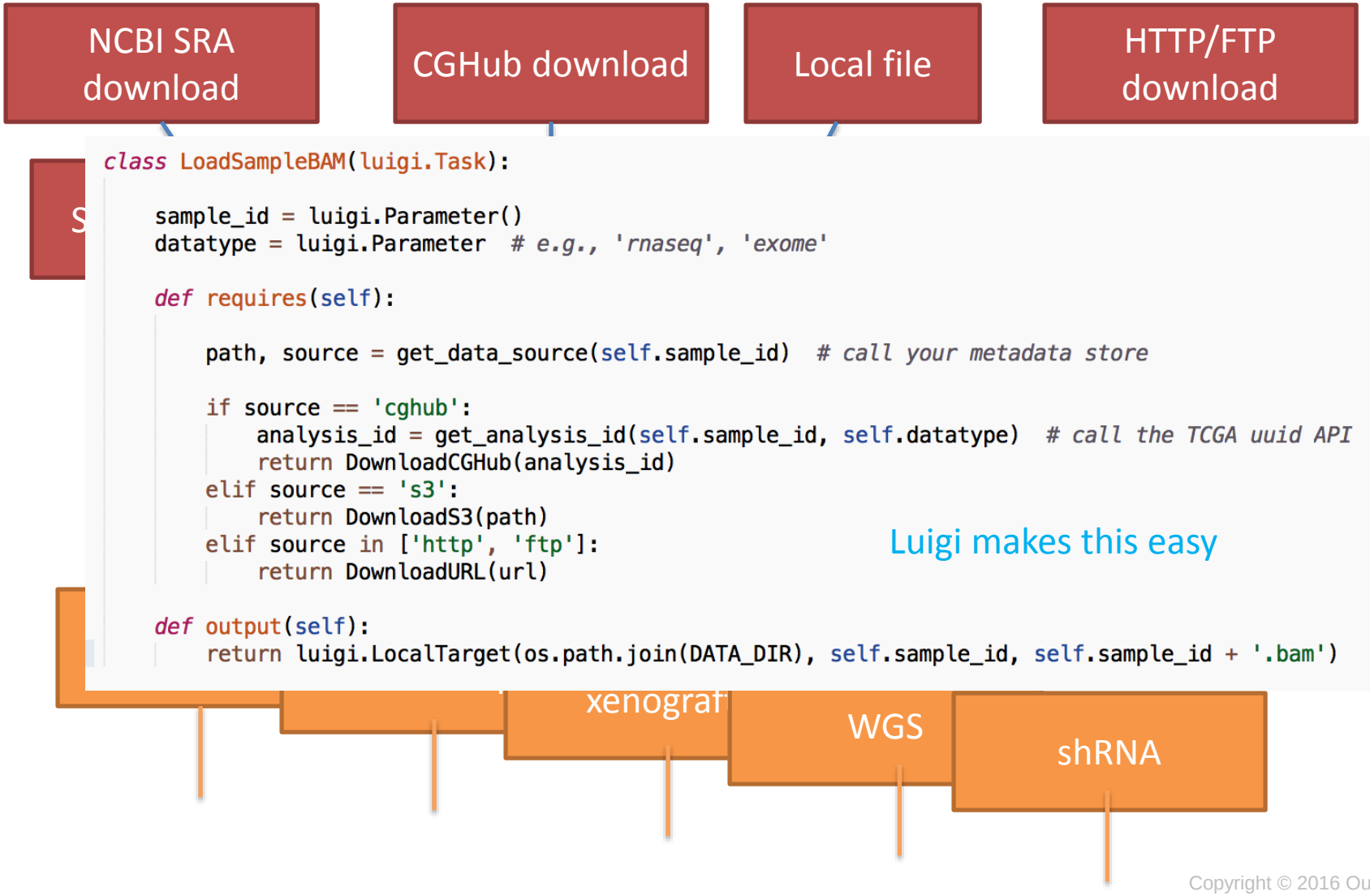


From http://docs.aws.amazon.com/AutoScaling/

Dispatching pattern allows fan-in from data sources, fan-out to pipelines



Dispatching pattern allows fan-in from data sources, fan-out to pipelines



Practical implementation notes

- Base Dockerfile with AWS credentials and helpers
- Simple metadata DB to map sample IDs → raw data locations
  - All other filepaths managed by pipeline code
  - Level-up: store pipeline versions, parameters, batches
- Small test dataset for constant integration tests (use Jenkins)

Challenges

- Filesystem: Still need a BIG, FAST local filesystem for NGS analysis

- Luigi has warts
  - Parameter passing
  - Delete files to re-run
  - Lots of code edits to rewire the DAG

5-year timeline (wishlist?)

- More distributed algorithms with ADAM/Spark
- Integration with Jupyter notebooks
- Open source template for spinning up a secure cloud infrastructure