ETL Architecture For Quick Iteration

Gil Shklarski
VP of Technology

FLATIRON

Who I am

VP of Technology at Flatiron Health

I used to be a scalable backends guy in Facebook and Microsoft

Now I mostly engineer teams and people...

At Flatiron, we organize the world's oncology treatment information and make it useful for patients, physicians, life science companies, and researchers.

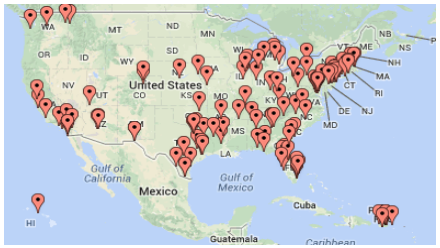
Building the world's largest cancer database

The best cancer research happens in clinical trials

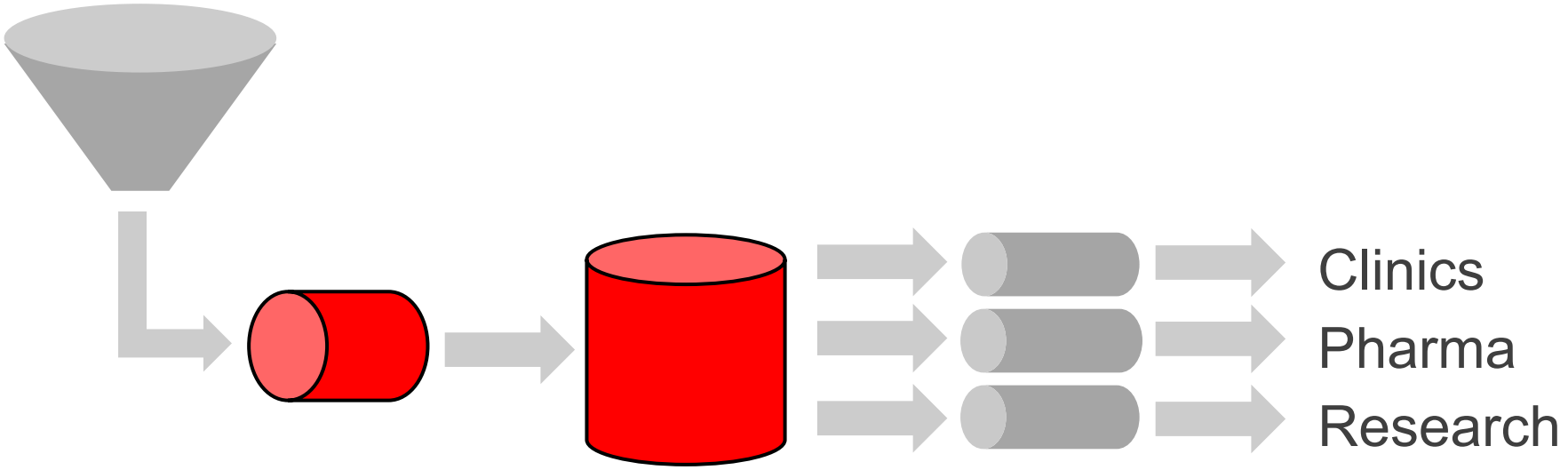
Only 4% of US cancer patients participate in clinical trials (expensive, hard to enroll)

Data detailing the treatment outcomes of 24 out of 25 patients stays siloed and is basically lost

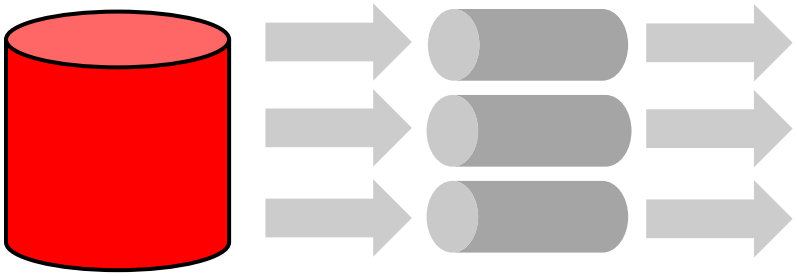
Building the world's largest cancer database

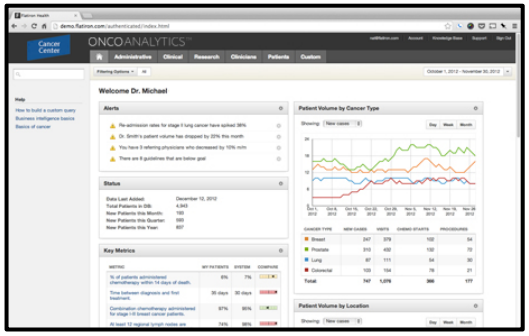


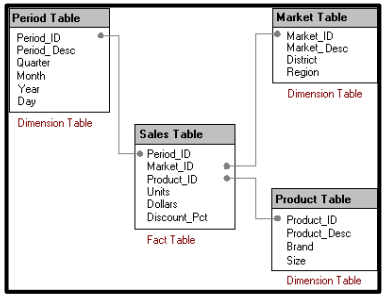
Largest real-world oncology data source, with ~1/5 of incident cases in the US

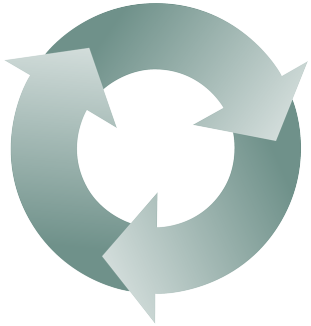


Building the world's largest cancer database





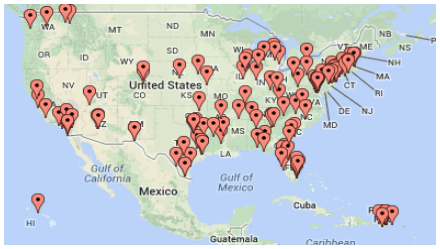




Agenda

- ~~Introduction~~
- Problem definition:
  - Task
  - Challenges
    - Data
    - Organizational
- Our approach:
  - What we built to address that
  - Challenges
  - Evolution
- Summary

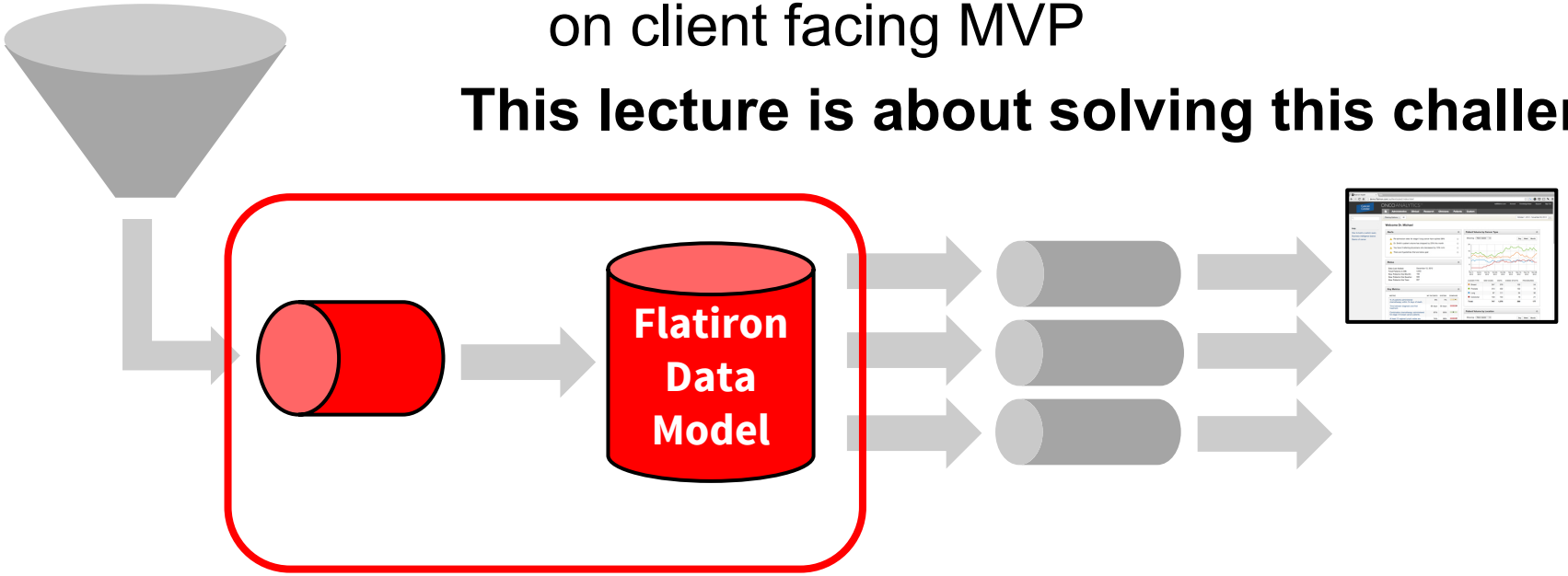
The task



Business requirement (early 2013):

- Get data from as many clinics as possible into a single data model.
- Optimize for quick client feedback: pivoting on client facing MVP

This lecture is about solving this challenge



Data Challenges:
Oncology Treatment Data 101

Oncology Clinic

Electronic **H**ealth **R**ecords:

- Visits, Diagnosis, Labs, Drugs
- Notes, scans, reports

Practice **M**anagement:

- Schedule, Charges
- Transactions, Insurance

Oncology Treatment Data 101

If you have seen one hospital IT setup, you have seen one hospital IT setup

- Source system heterogeneity
- Clinical workflow heterogeneity
- Interfaces heterogeneity
- A lot of legacy data

Oncology Treatment Data 101

2220	Blood Serum Albumin	g/dL
QD25001600	ALBUMIN/GLOBULIN RATIO QD	(calc)
QD25001400	ALBUMIN QD	g/dL
QD50058600	ALBUMIN	%
QD50055700	ALBUMIN	g/dL
CL3215104	Albumin % (EPR)	%
LC001081	ALBUMIN, SERUM (001081)	g/dL
LC003718	Albumin, U	%
LC001488	Albumin	g/dL
LC133751	Albumin, U	%
CL3215162	Albumin%, Urine	%
CL3215160	Albumin, Urine	mg/24hr
3234	ALBUMIN SS	g/dL
LC133686	Albumin, U	%
QD50060710	MICROALBUMIN	mg/dL
QD50061100	MICROALBUMIN/CREATININE RATIO, RANDOM URINE	mcg/mg creat
QD85991610	ALBUMIN	relative %
50058600	ALBUMIN UPEP RAND	%
CL3210074	ALBUMIN LEVEL	g/dL
QD86008211	ALBUMIN/GLOBULIN RATIO	(calc)
LC149520	Albumin	g/dL
QD45069600	PREALBUMIN	mg/dL
QD900415245	ALBUMIN, SERUM	mg/dl
QD900429745	ALBUMIN	g/dL
CL3215124	Albumin Electrophoresis	g/dL
LC016931	Prealbumin	mg/dL
QD50060800	MICROALBUMIN, 24 HOUR UR	mg/24 h
QD50060900	MICROALBUMIN, 24 HOUR UR	mcg/min
QD85994821	ALBUMIN,SERUM	g/dL
CL3213320	PREALBUMIN	mg/dL
QD85995225	PROTEIN ELECTROPHORESIS ALBUMIN	g/dL

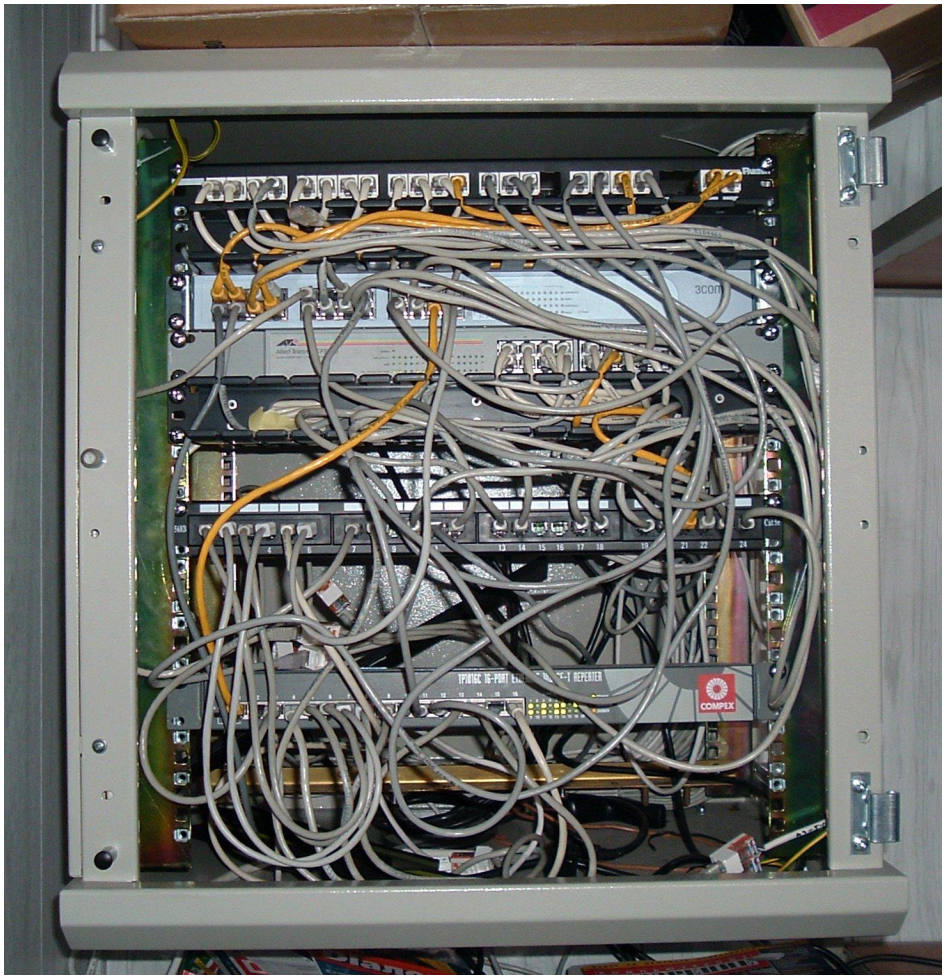
1751-7	**Albumin [Mass/volume] in Serum or Plasma**	**g/dL**

FLATIRON

Data Challenges: Summary

- Dirty, high dimensional, incomplete, heterogenous data
- Not: web scale log-streams
- Not: huge images, genomic sequence data

Give me the tech already!

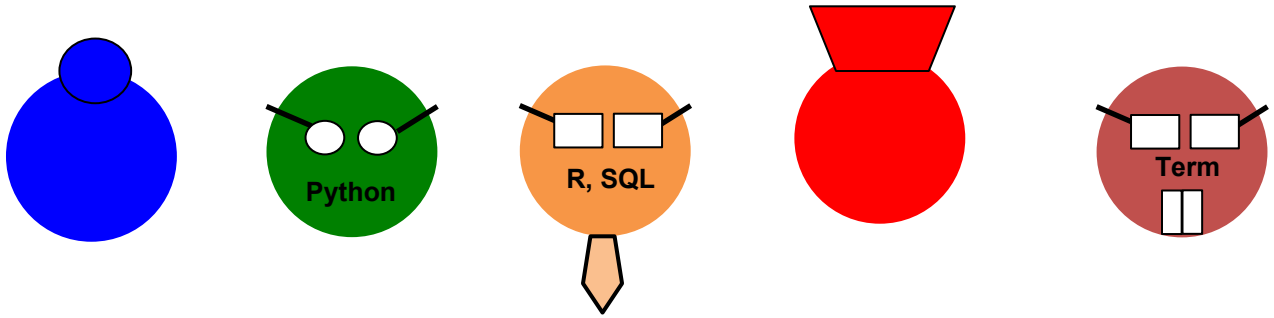


Not just yet...

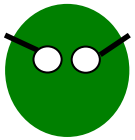
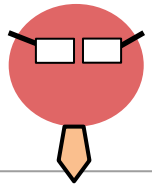
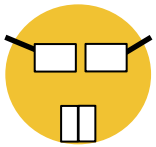
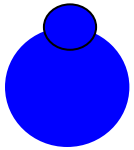
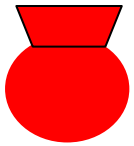
Understanding our organizational challenge is key for describing the architecture we built

Organizational Challenge

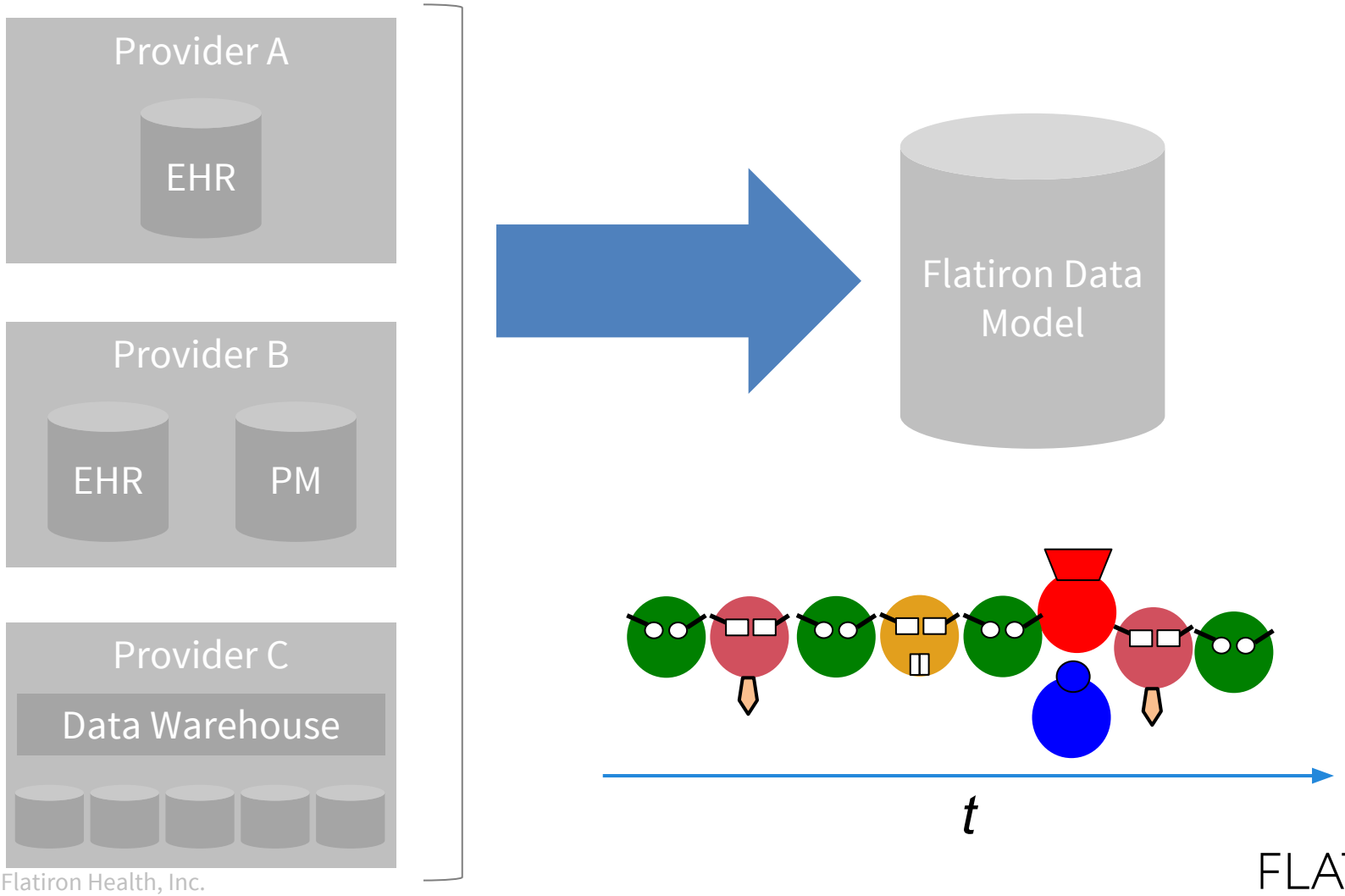
An oncologist, an engineer, a medical analyst, a nurse, and an informaticist walk into a bar…



Organizational Challenge

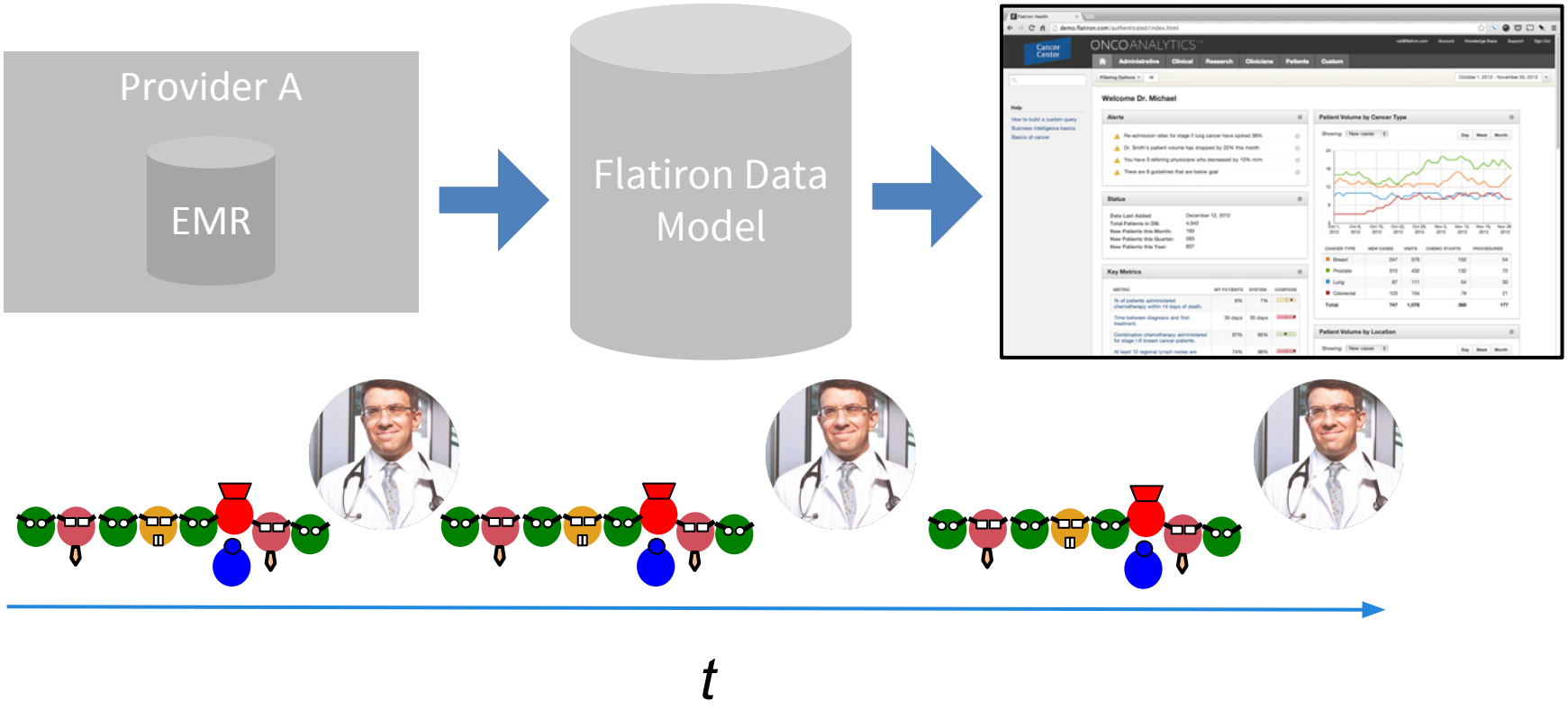
	Software engineer	Build systems, algorithms, implement business logic	Python, JS, SQL,...
	Data Analyst	Expert of clinics and EHR/PM databases	SQL, R,...
	Medical Informaticist	Expert of medical data representation, terminologies	Proprietary tools
	Flatiron Nurse	Expert of treatment workflows	English
	Flatiron Oncologist	Understands the disease and treatment	English

Organizational Challenge



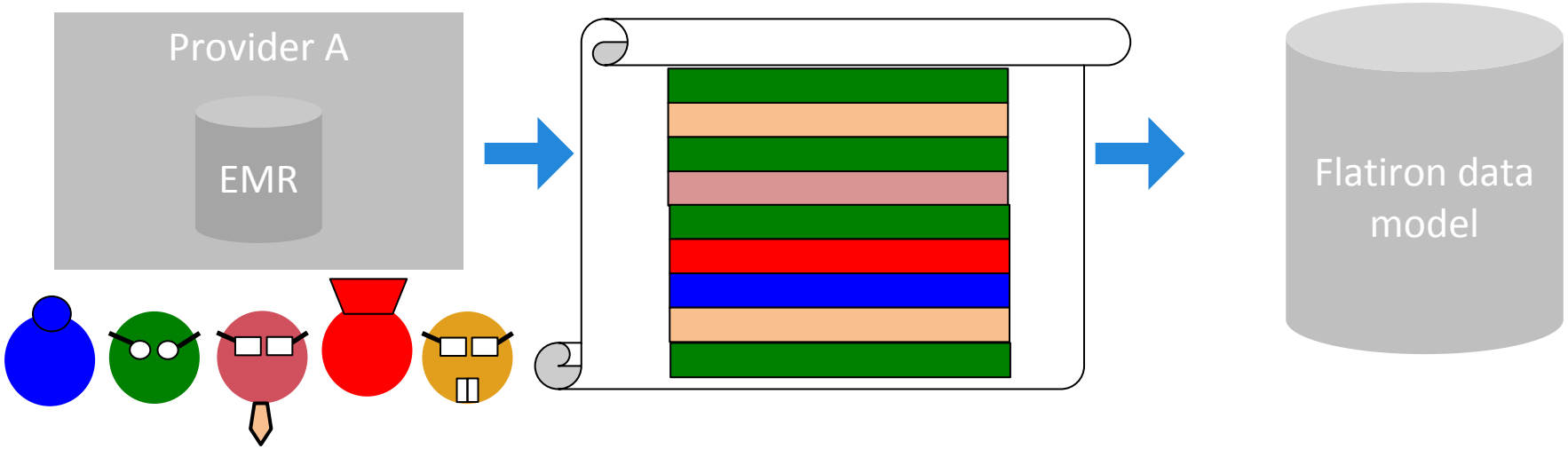
FLATIRON

Organizational Challenge



FLATIRON

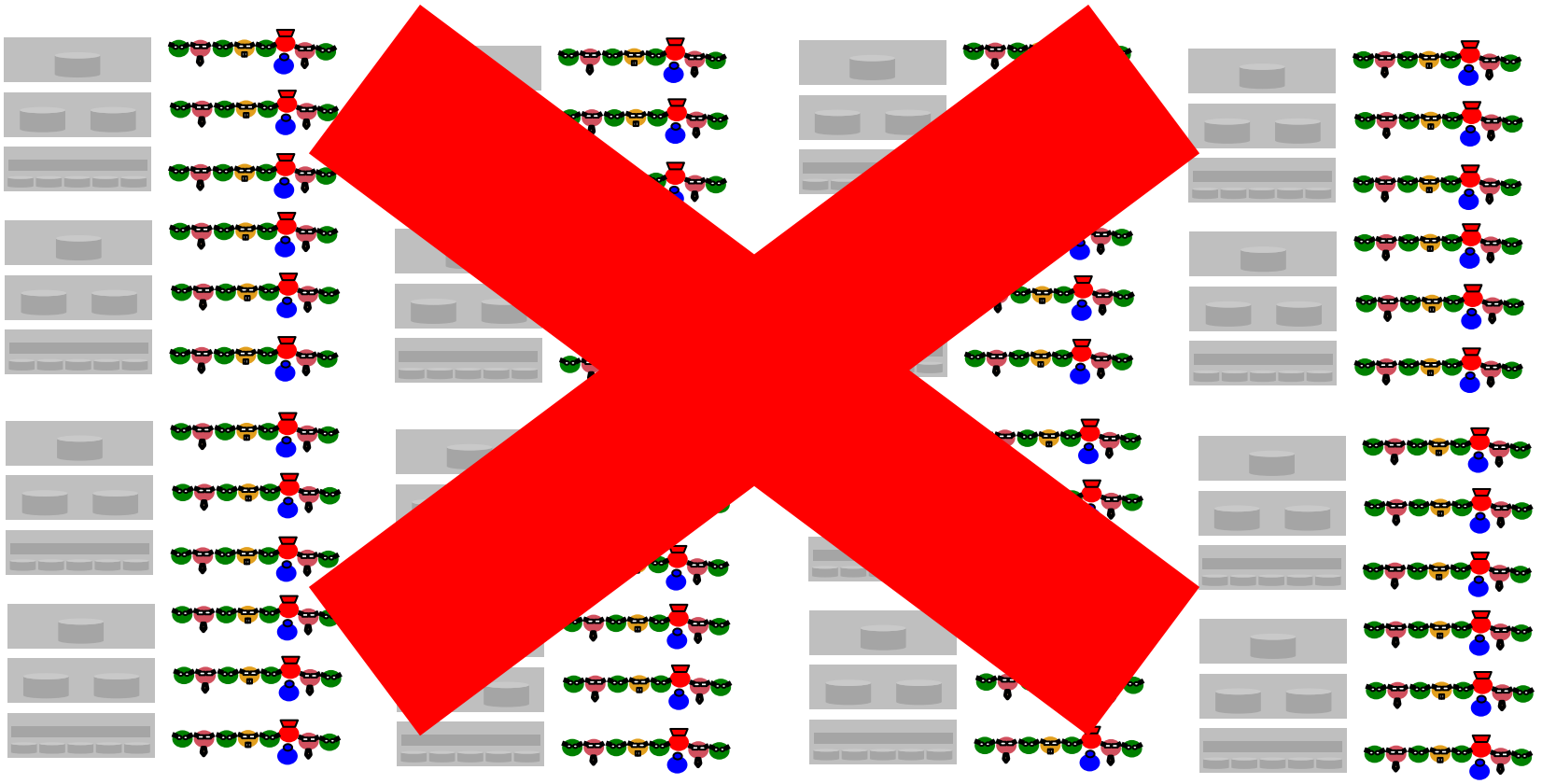
Organizational Challenge



Conway's law (1968):
organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations...

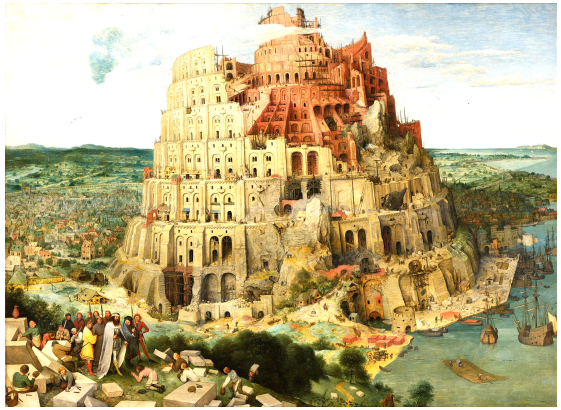
FLATIRON

Organizational Challenge



FLATIRON

Organizational Challenge: Summary



Integrating a clinic requires multidisciplinary expertise

Hand-off between disciplines required a lot of back-and forth

Initial attempt produced codebase difficult to understand, hard to reuse, hard to maintain

Clash between agile process and unpredictable external feedback

Agenda

- ~~Introduction~~
- ~~Problem definition:~~
  - ~~Task~~
  - ~~Challenges~~
    - ~~Data~~
    - ~~Organizational~~
- Our approach:
  - What we built to address that
  - Challenges
  - Evolution
- Summary

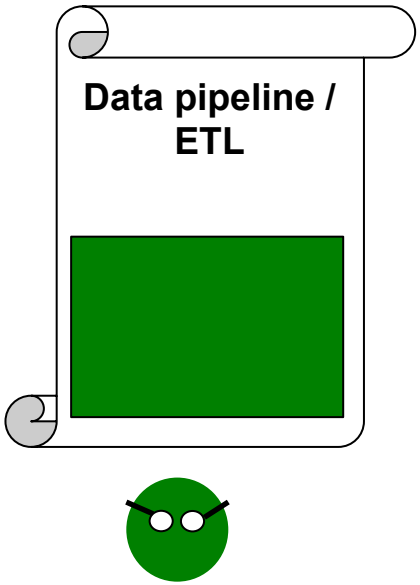
Our Data Integration requirements

- Optimize for latency to client feedback:
  - Limit multidisciplinary handoffs
  - Optimize experimentation/invention process

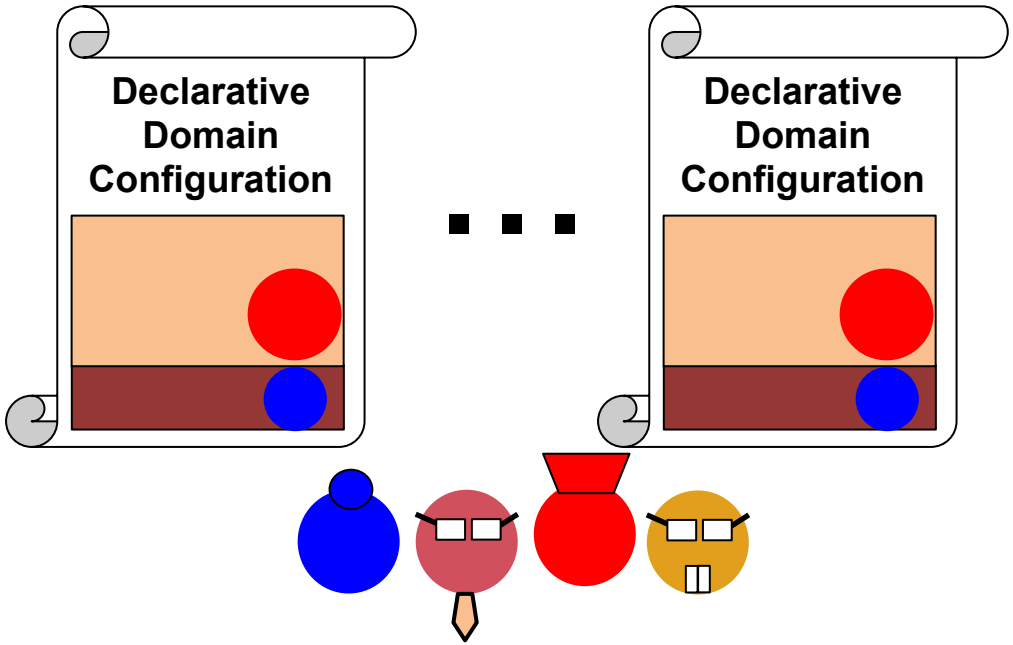
- Support healthcare data sources:
  - make it easy to add client specific logic
  - make it easy to reuse logic
  - make it secure (protect health information)

1st realization - ETL framework

Framework:



Per client:

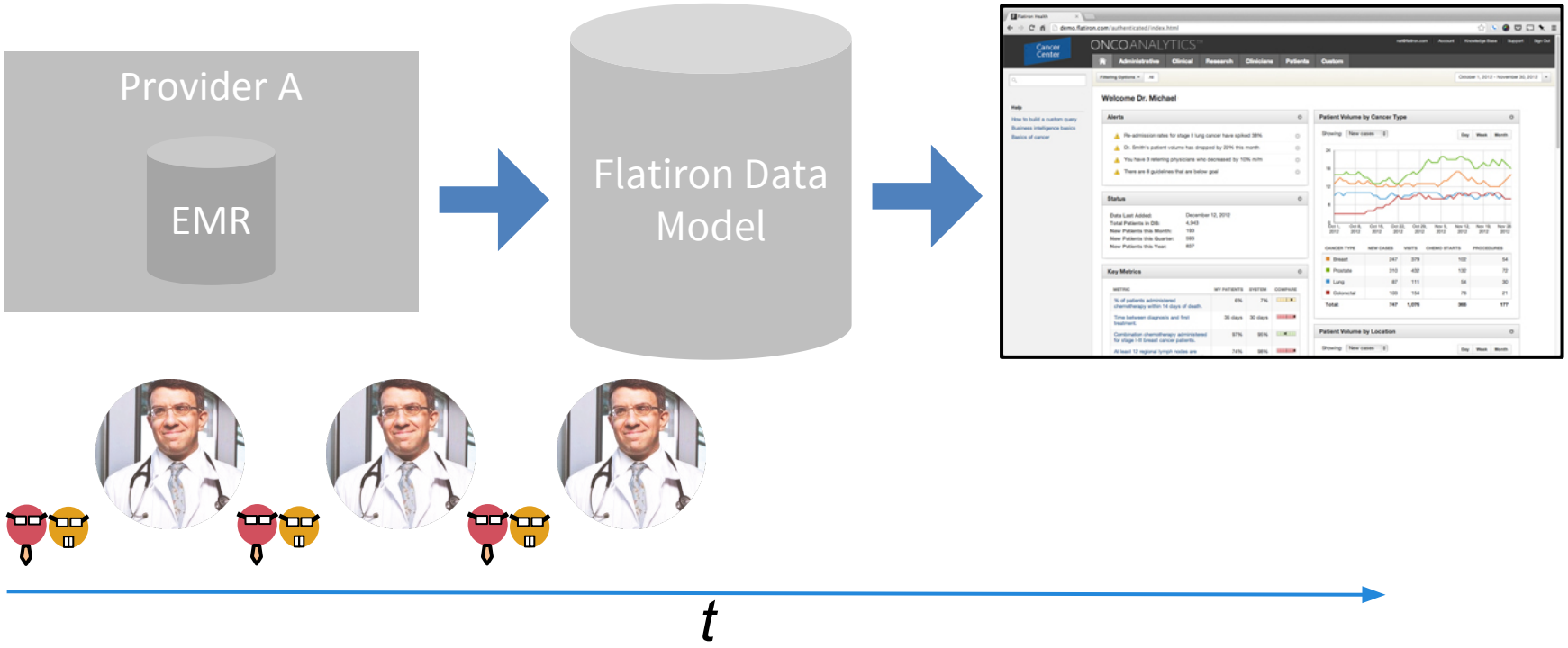


Separation of concerns: an analyst can be fully productive with SQL alone (or R or ...).

No more business logic handoffs, redundant knowledge transfer lag, and its debugging

FLATIRON

1st realization - ETL framework



	Data Analyst
	Medical Informaticist

FLATIRON

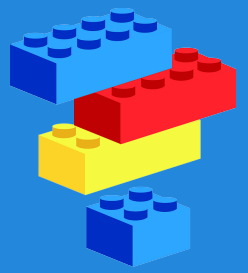
Our Data Integration requirements

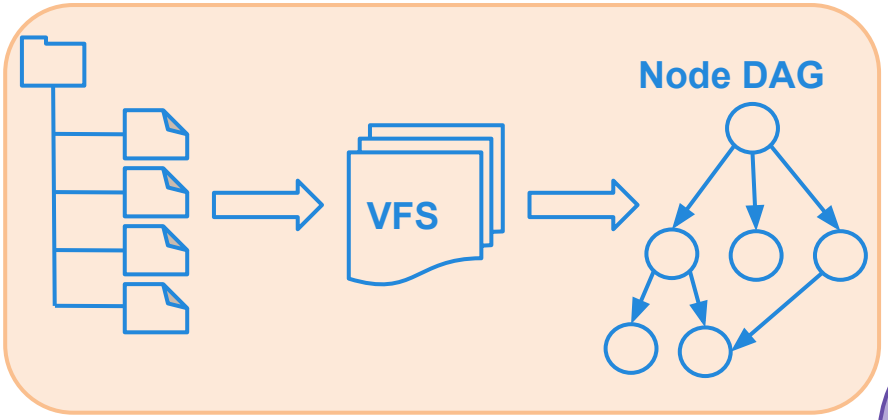
- Optimize for latency to client feedback:
  - ~~Limit multidisciplinary handoffs~~
  - Optimize experimentation/invention process

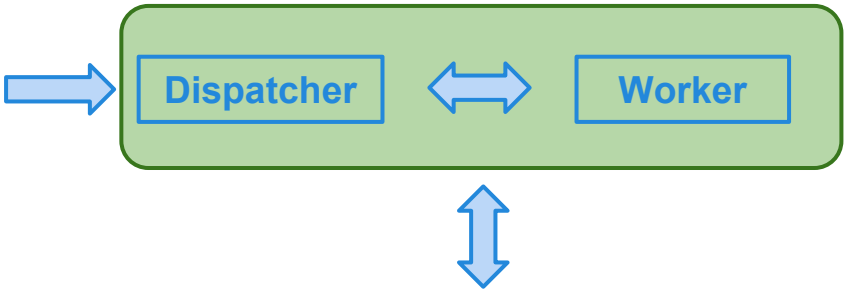
- Support healthcare data sources:
  - make it easy to add client specific logic
  - make it easy to reuse logic
  - secure protected health information

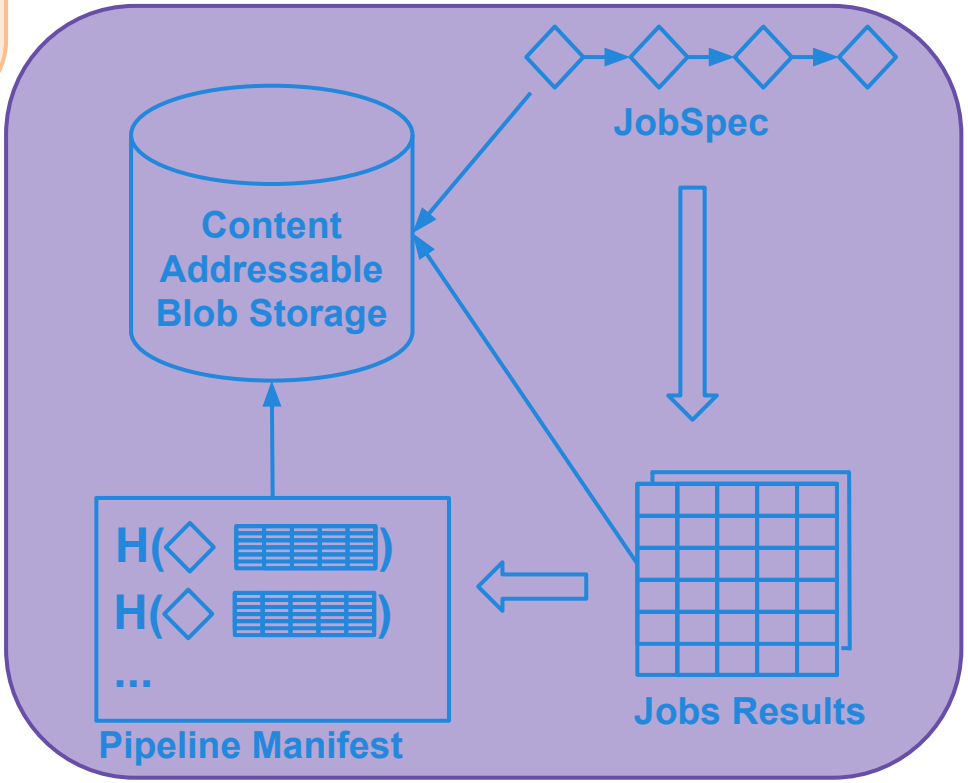
Build vs. Buy ?

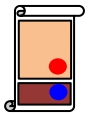
Our ETL Framework - High level architecture



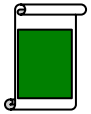




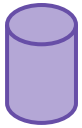




ETL Configuration



Runtime



Results / execution artifacts

ETL Representation

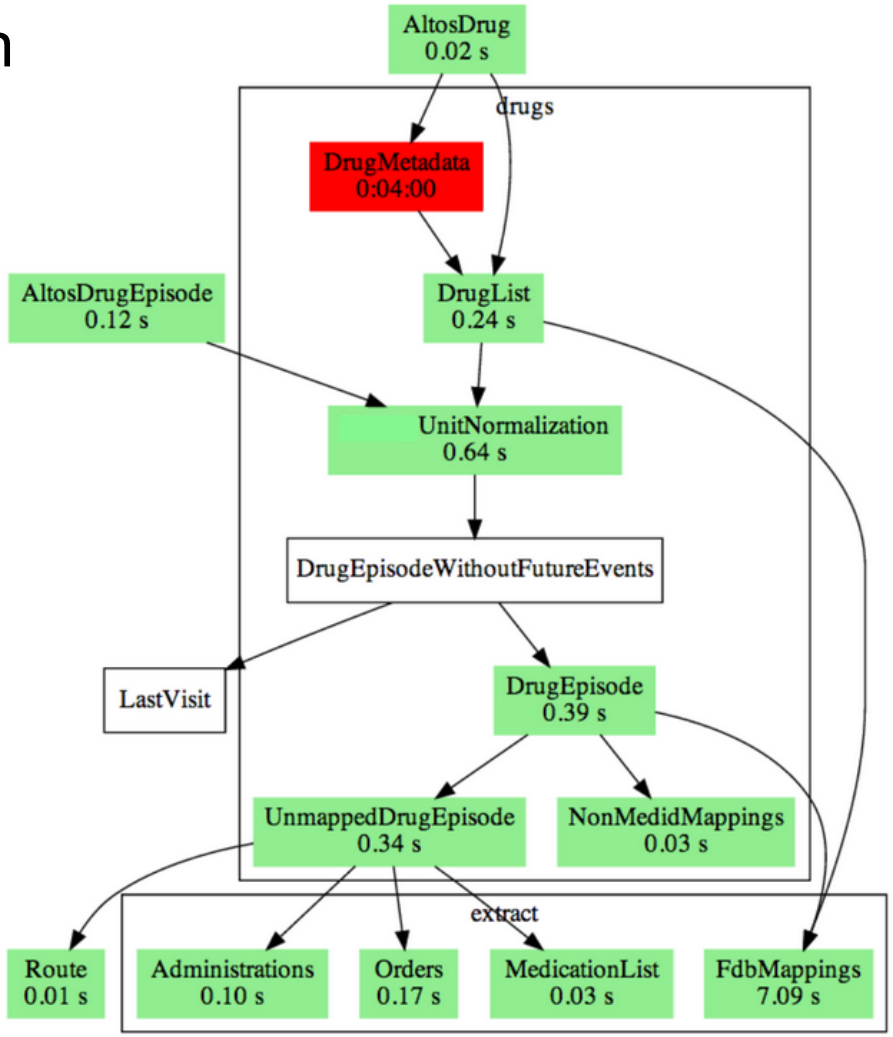
Complex transformation broken to a series of simpler transformation represented as directed acyclic graph.

Each node:

Input: A set of tables (nodes)

Transformation: A simple SQL, Python, R, Bash transform

Output: A single table



ETL Representation

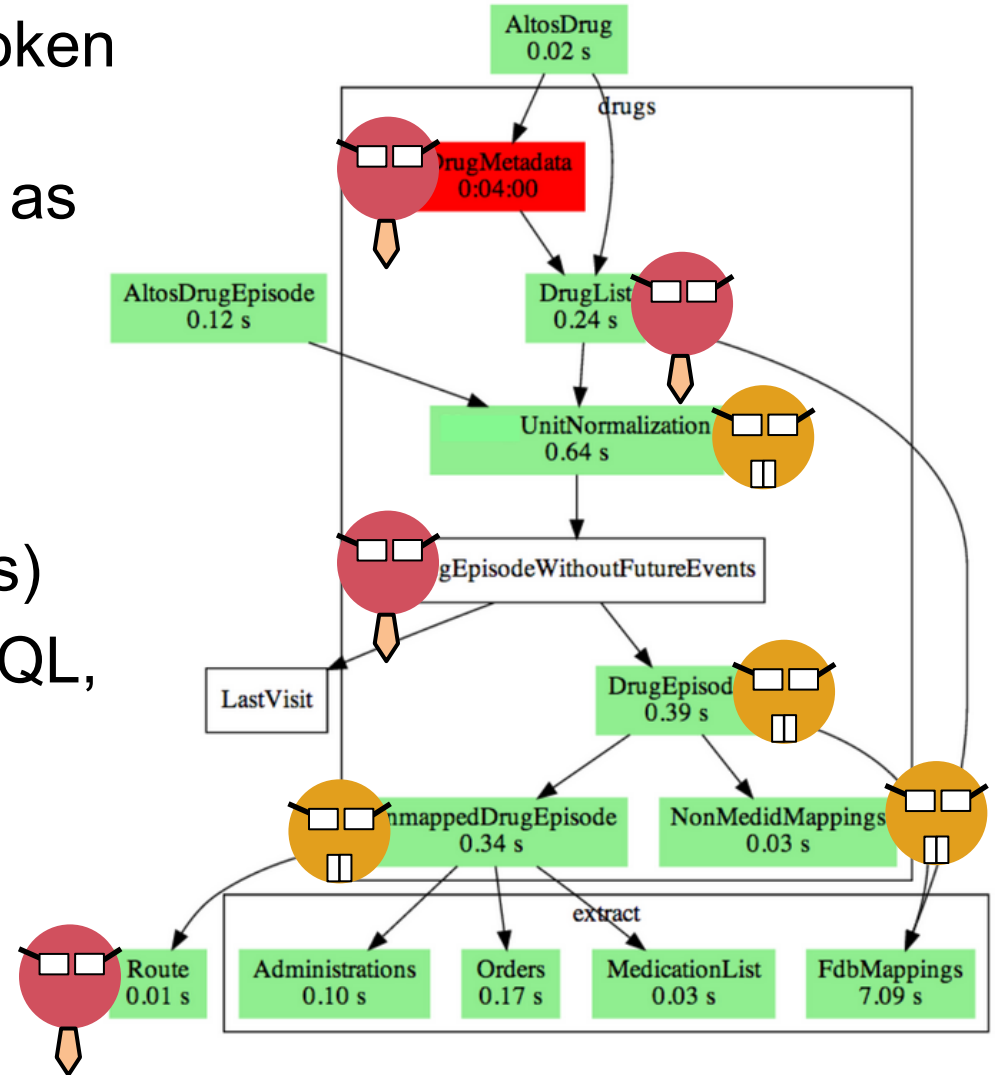
Complex transformation broken to a series of simpler transformation represented as directed acyclic graph.

Each node:

Input: A set of tables (nodes)

Transformation: A simple SQL, Python, R, Bash transform

Output: A single table

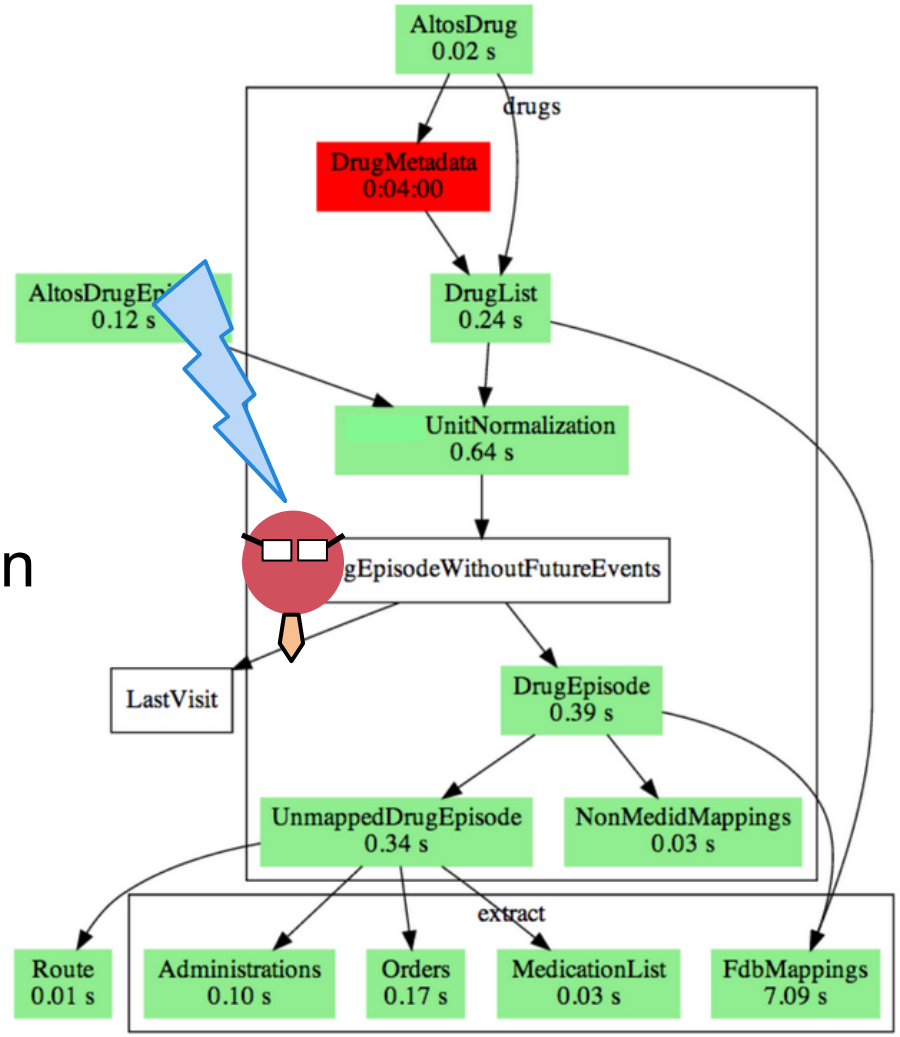


ETL Representation

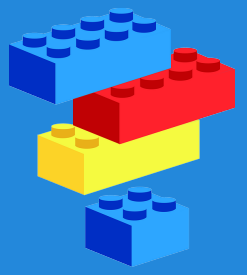
Nodes can input and output CSVs.

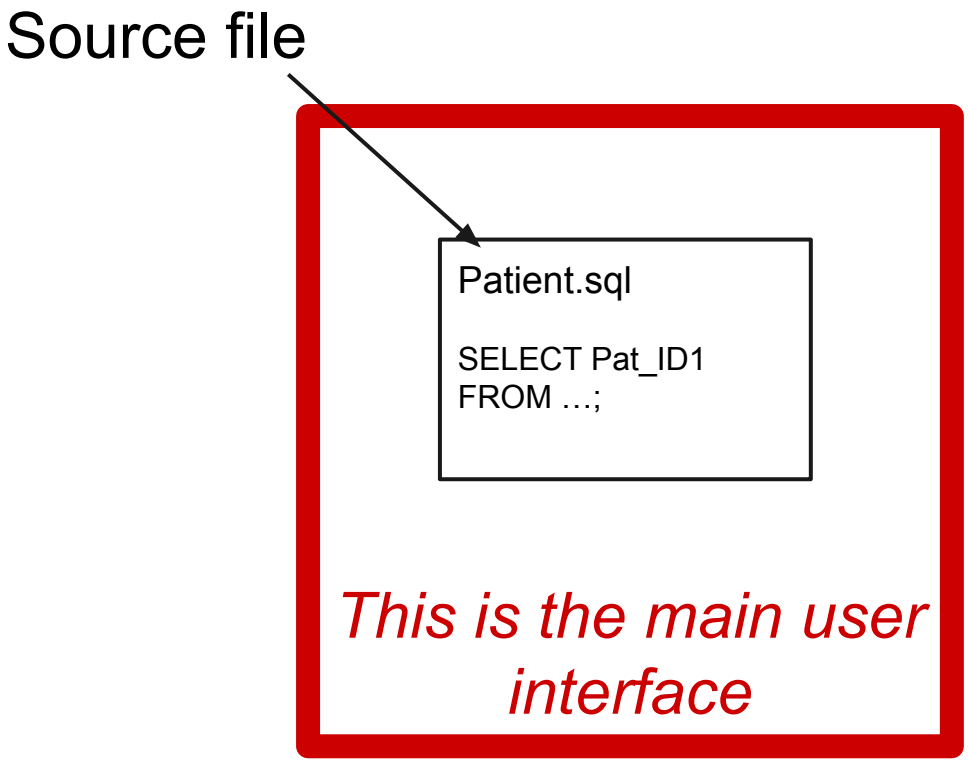
ETL graph representation is agnostic to specific languages.

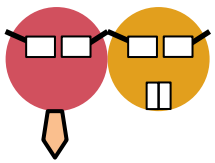
Adding a transformation node in any language is as easy as adding several lines in the framework's code.



In practice

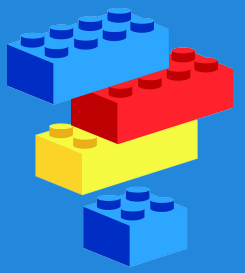


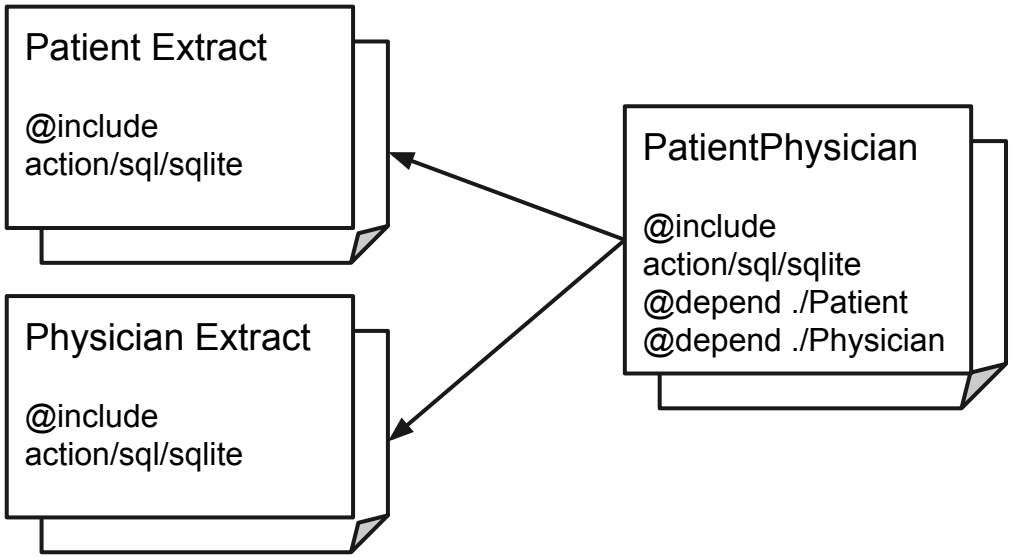


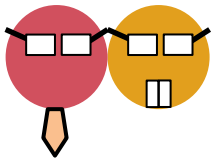


FLATIRON

In practice - dependencies

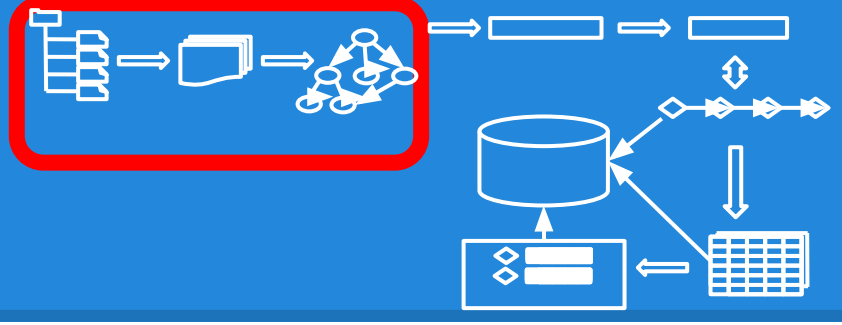




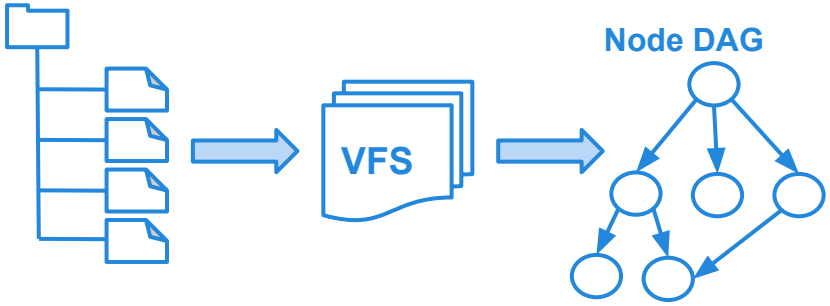


FLATIRON

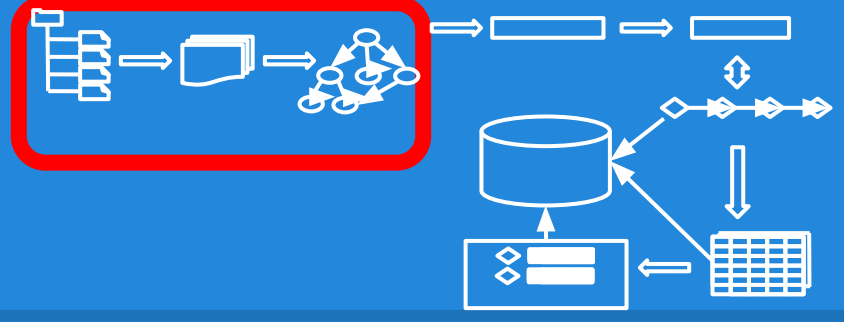
ETL Representation



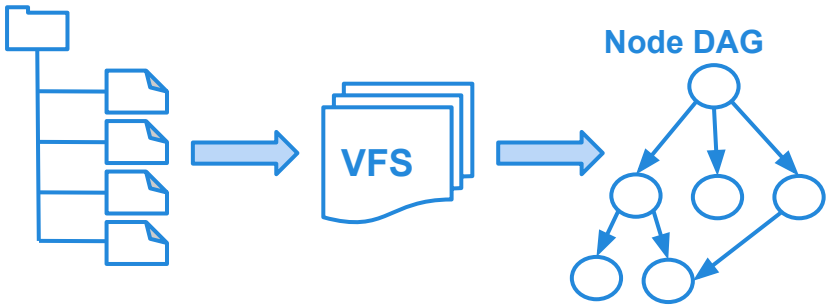
- Text representation
  - Each independent pipeline is a directory
  - Each transformation is a file
  - Directives in file define dependencies
- Git
  - Versioning
  - Branch == Experiment
  - Easy branching...



ETL Representation

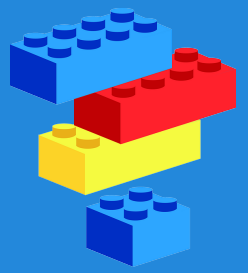


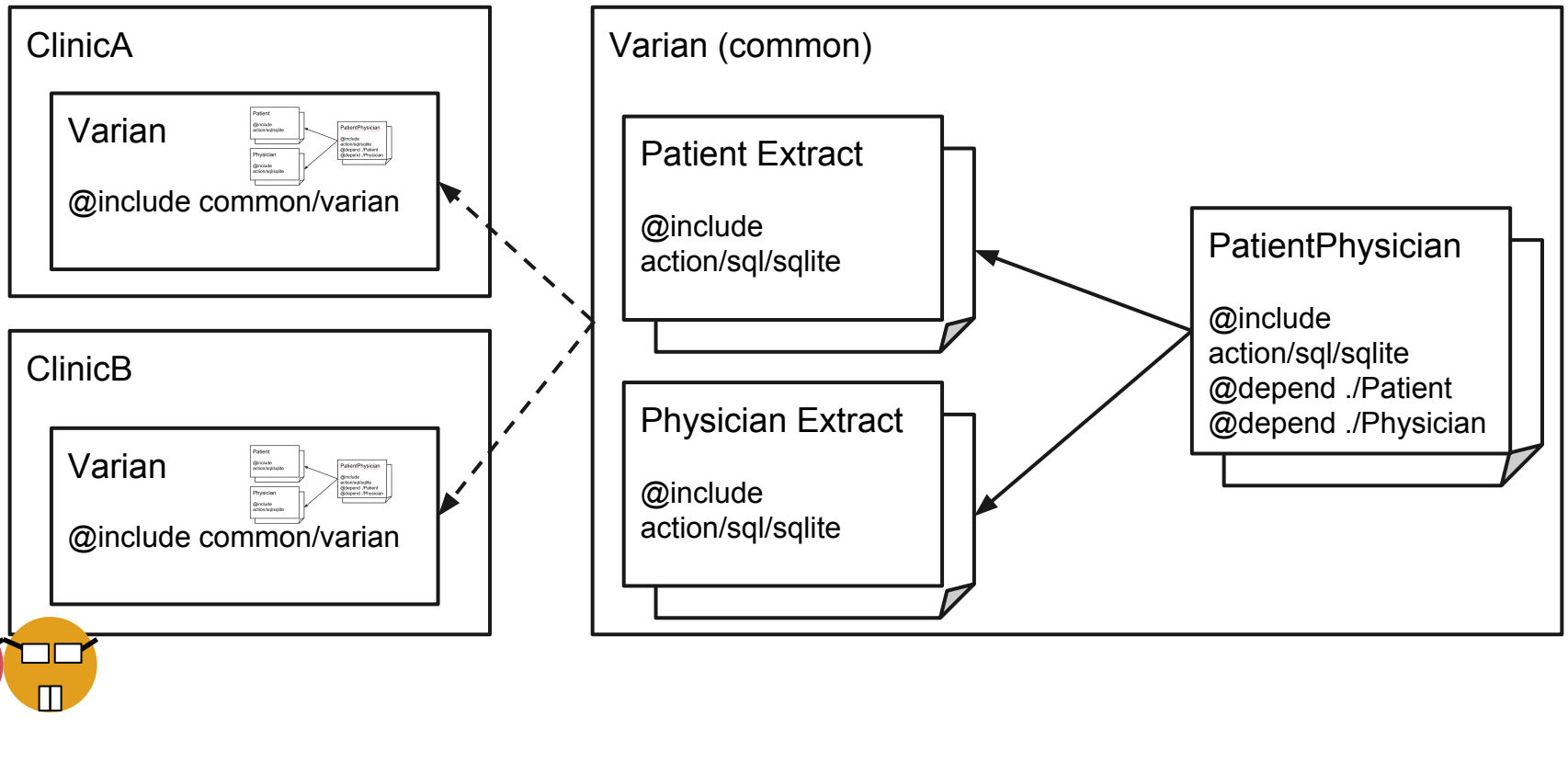
- VFS enables richness of directives



- Introduction of OO concepts and meta programming world to the ETL world
  - Inheritance and polymorphism of one more groups of transformations
  - Templating (parameter substitution)
- Adding a new clinic
  - In many cases, as easy as a set of include directives, overriding several nodes

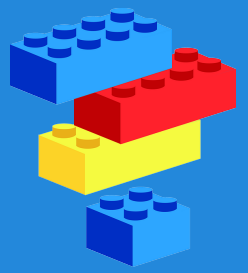
In practice - node reuse

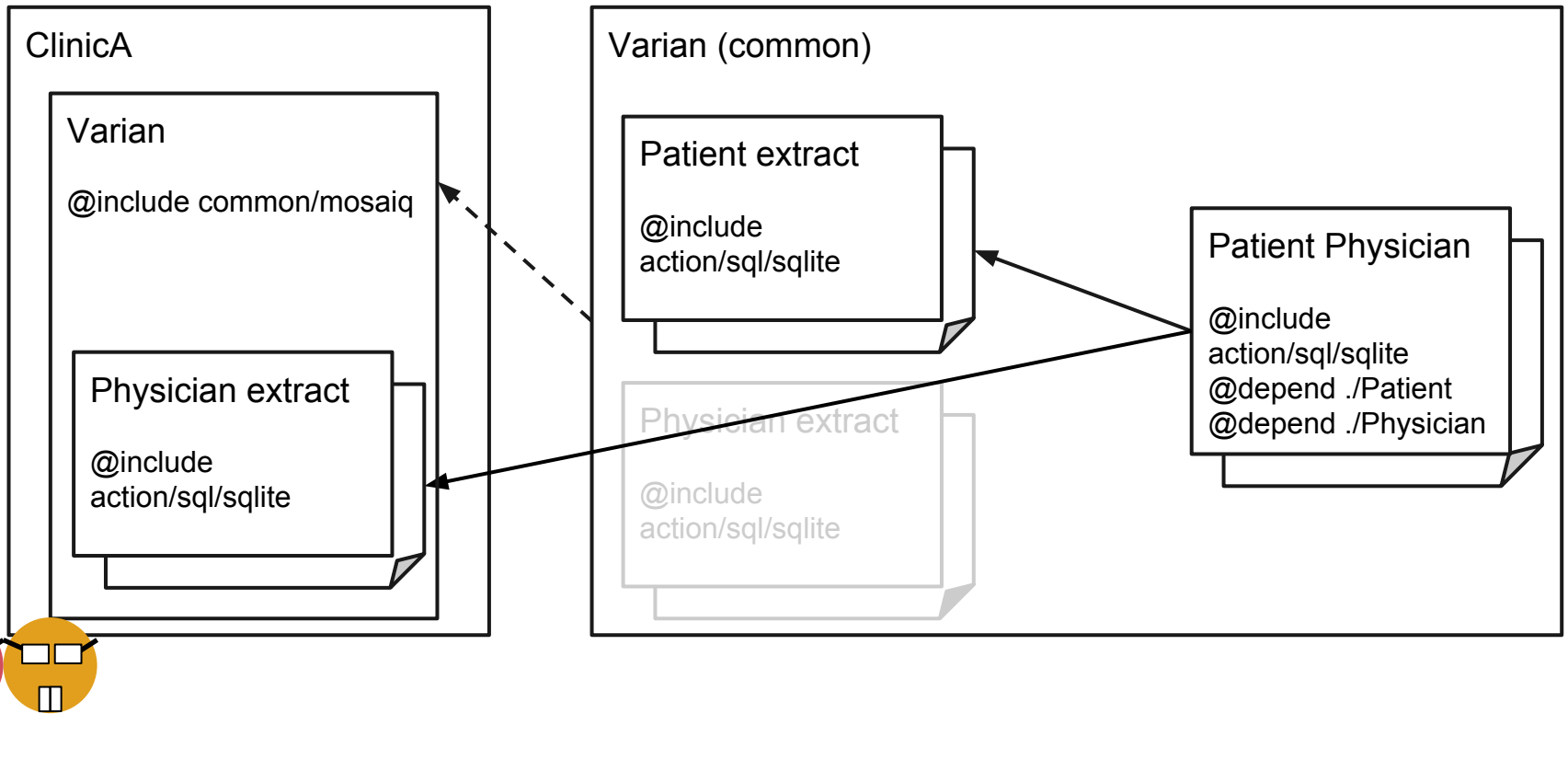




FLATIRON

In practice - node override





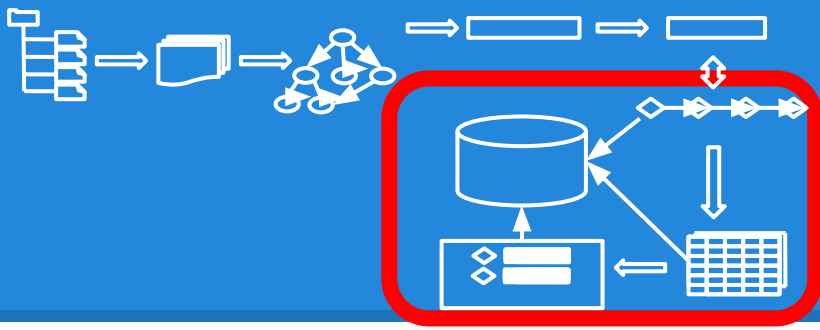
FLATIRON

Our Data Integration requirements

- Optimize for latency to client feedback:
  - ~~Limit multidisciplinary handoffs~~
  - Optimize experimentation/invention process

- Support healthcare data sources:
  - ~~make it easy to add client specific logic~~
  - ~~make it easy to reuse logic~~
  - secure protected health information

Execution artifacts caching



- Our node transformations are deterministic functions:

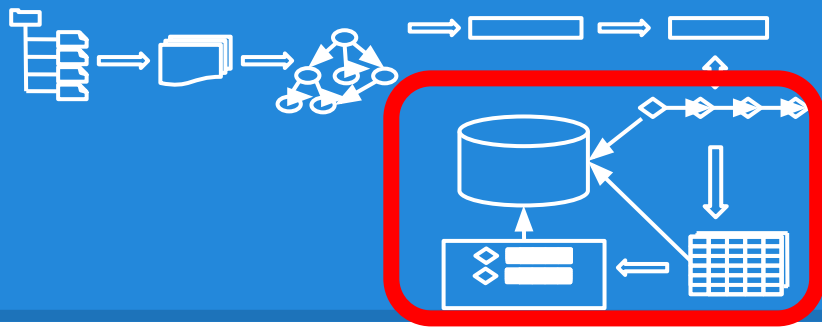
  f(input tables, node code) = output table

- We can cache results using the following key:

  Hash(input tables, node code)

- We cache results in an immutable centralized key value store (currently S3)

Execution artifacts caching



A: AltosDrug.sql

```
@include action/sql/sqlite
@depend ./DrugMetadata
@depend ./DrugList

SELECT C1, C2
JOIN …
….
```

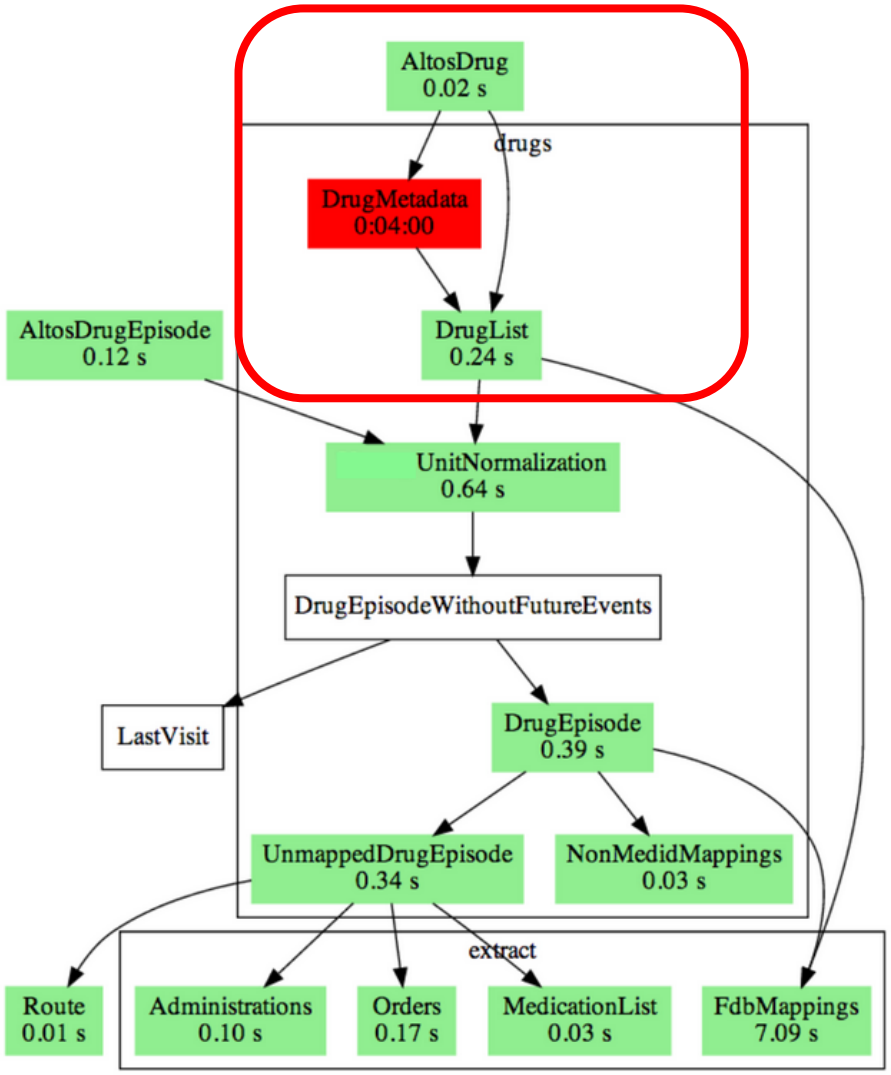
B: DrugList.csv

```
1,Tylenol,100mg,..
2, B12, 50mg,...
3, Carboplatin, 10ml
...
```

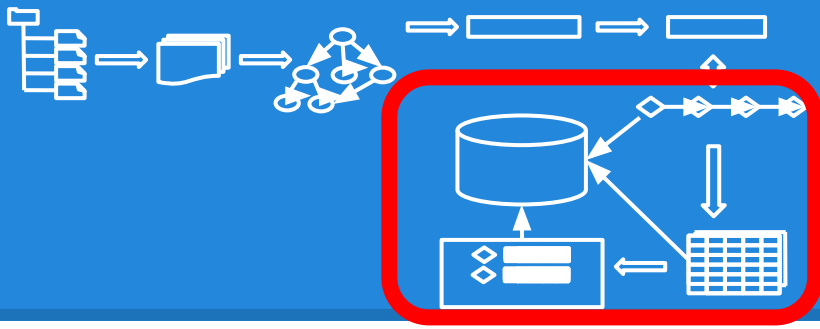
C: DrugMetadata.csv

```
Tylenol,acetaminophen,...
B12, cyanocobalamin,...
Abraxane, paclitaxel,...
Onxol, paclitaxel,...
….
```

Key = SHA256(A||B||C)



Execution artifacts caching



Production

A: AltosDrug.sql

```
@include action/sql/sqlite
@depend ./DrugMetadata
@depend ./DrugList

SELECT C1, C2
JOIN …
….
```

B: DrugList.csv

```
1,Tylenol,100mg,..
2, B12, 50mg,...
3, Carboplatin, 10ml
...
```

C: DrugMetadata.csv

```
Tylenol,acetaminophen,...
B12, cyanocobalamin,...
Abraxane, paclitaxel,...
Onxol, paclitaxel,...
….
```

Key = SHA256(A||B||C)

Experiment

A': AltosDrug.sql

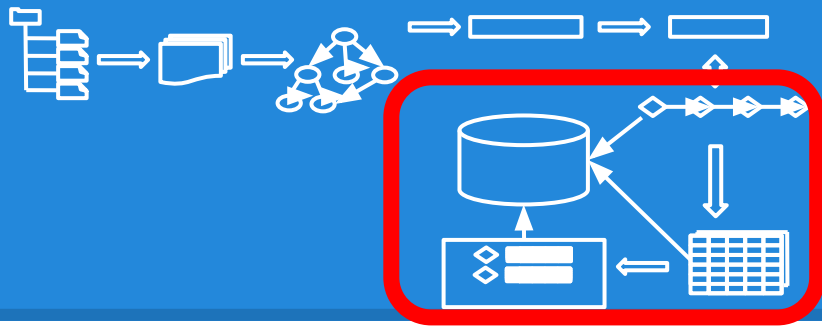
```
@include action/sql/sqlite
@depend ./DrugMetadata
@depend ./DrugList

SELECT C1, C3
JOIN …
….
```

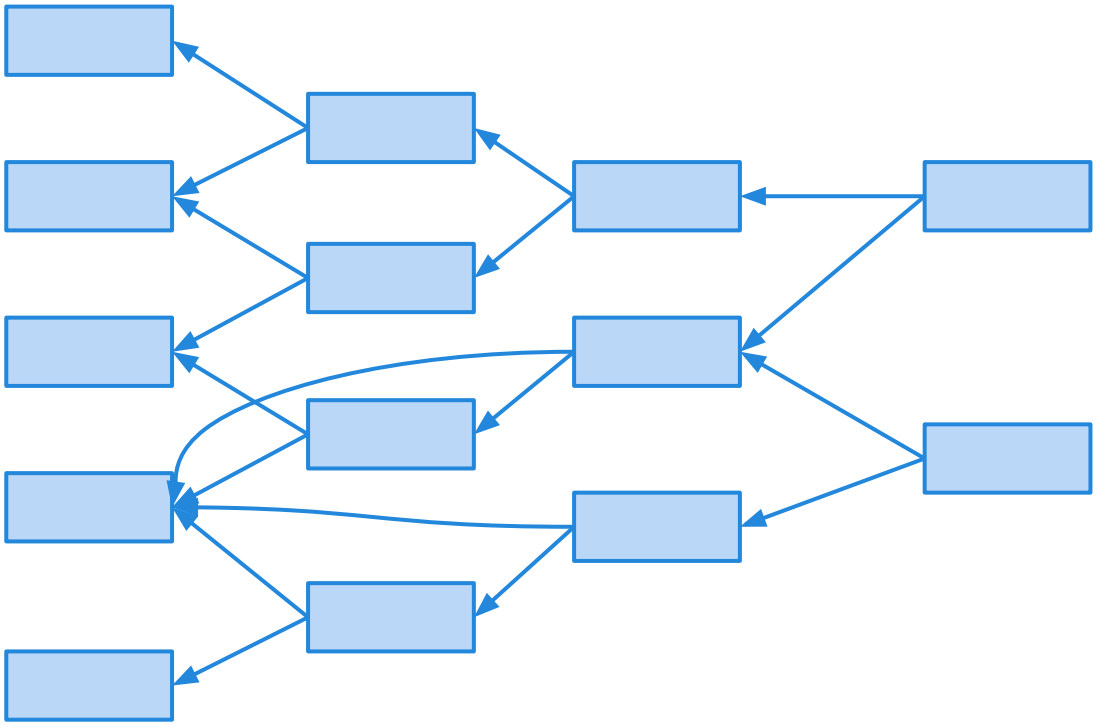
Key' = SHA256(A' ||B||C)

FLATIRON

Cache in action - 2nd example

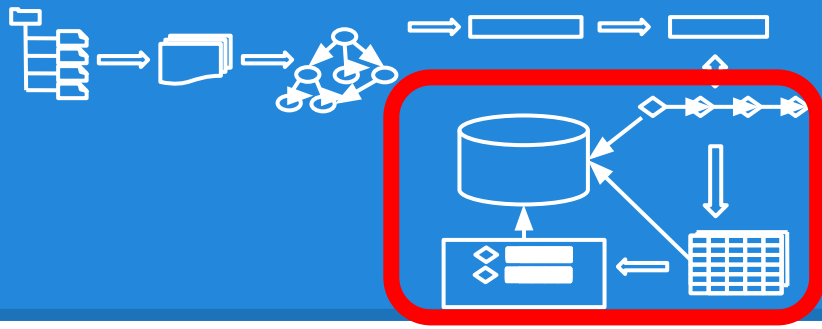


Latest production run results

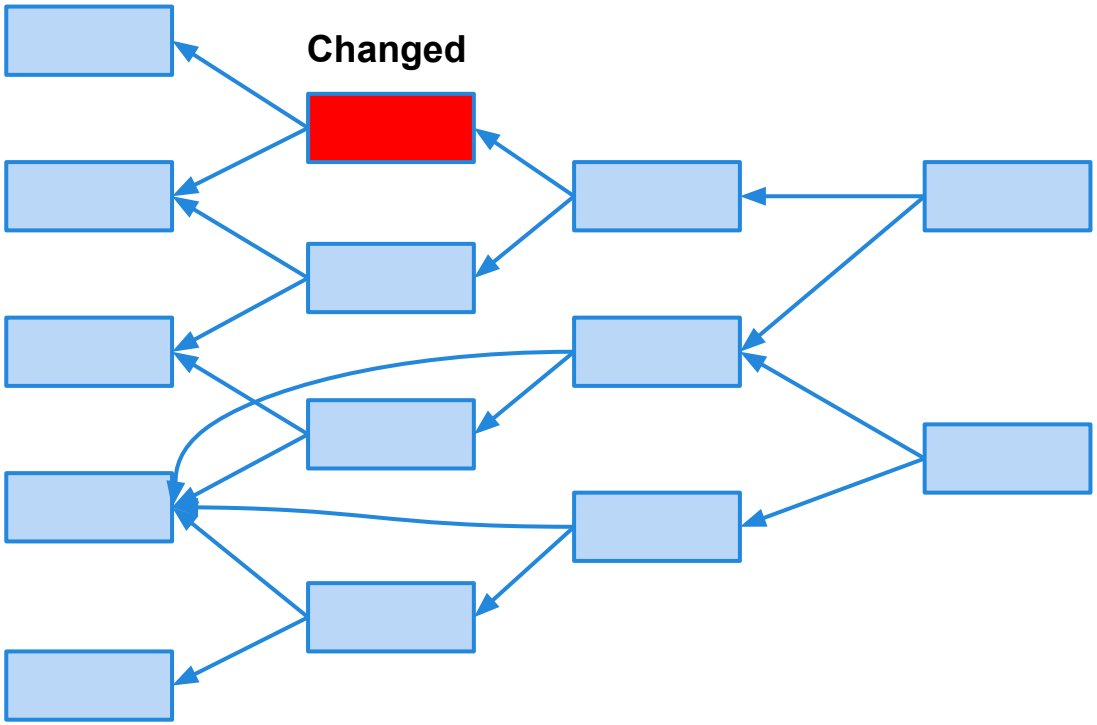


FLATIRON

Cache in action - 2nd example

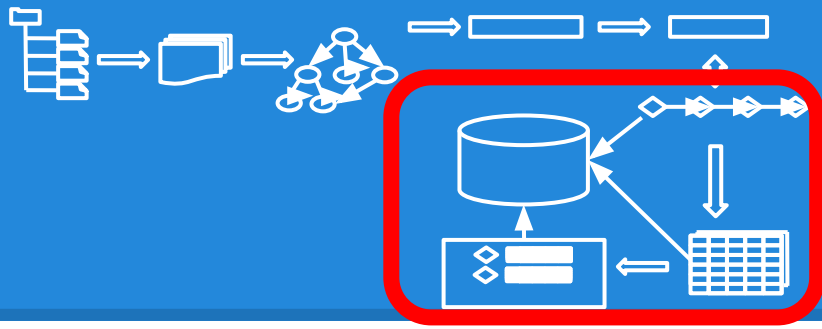


One transformation file changed

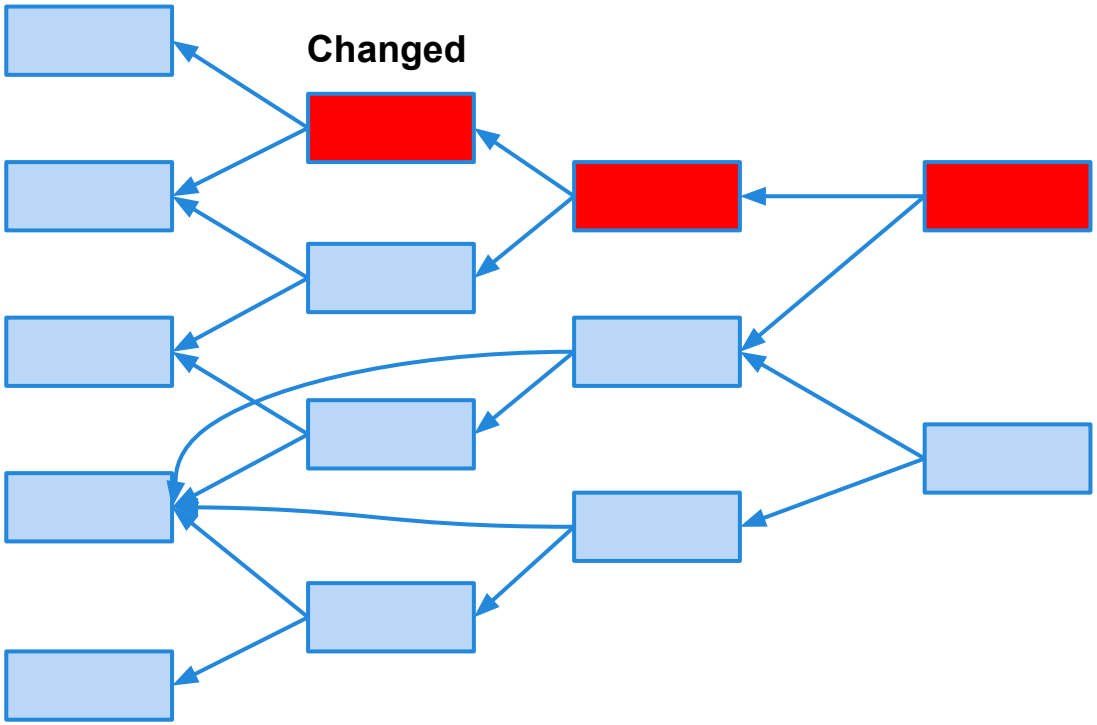


FLATIRON

Cache in action - 2nd example



Limited subgraph needs recompute



FLATIRON

Quick experimentation

- Surgically changing a very complex pipeline is fast
  - Branching is easy (as any other code branch in Git)
  - Overriding a node is easy (think OO)
  - Need to recompute only one path of the DAG, all other freshest data is ready

- Our data analysts can experiment independently:
  - Need to know SQL/R code
  - Need to know how to commit code to Git

Continuous data ingestion

- For specific pipelines (ETL graphs)
  - Whenever we check-in a new node (or a node update) we recompute the pipeline
  - Cache of intermediates makes this efficient
- Basically a continuous integration test
- Allows testing and development with most recent data
- Data artifacts are computed, stored, and ready for reuse - “Warms the cache”

Our Data Integration requirements

- Optimize for latency to client feedback:
  - ~~Limit multidisciplinary handoffs~~
  - ~~Optimize experimentation/invention process~~

- Support healthcare data sources:
  - ~~make it easy to add client specific logic~~
  - ~~make it easy to reuse logic~~
  - secure protected health information

Security for Protected Health Information

- Exchanging cache keys (secure hashes) instead of actual data
  - Central storage - central governance
  - Immutable storage - supports auditing and data provenance requirements.
    - For each pipeline run we store a manifest which lists all the artifacts for that run.

Bottom Line

Challenges

- Homegrown tool
  - Rebuilding the wings of an airplane while flying
  - Training engineers and non-engineers on a homegrown tool
  - Lack of a development environment (everything beyond text tools)
- The tool is only one part of the equation:
  - Tool is used in use cases not planned for (great, but fails in new ways)
  - Production == Experimentation
    - Empowerment of non-engineers
    - A lot of ‘engineering skills’ to teach

Evolution

- Building a startup does not end with experimentation
  - Tool built in our ‘invent’ phase.
  - We successfully supported the business…
  - Now, this aspect of the company switched to the ‘scaling’ phase
- Business requirement expanded from “integrate as quickly as you can” to “do that, and have a stable nightly pipeline”

Evolution

- We built a lot around this ETL framework:
  - Pipelines scheduling system
  - Running multiple pipelines including running stable (old) versions of ETL nodes to guarantee fresh data
- Ongoing work:
  - Mature testing framework
  - Optimization for large data sets / streaming (incremental ETL). Started to become an issue two years in.
  - Various aspects of robustness

Summary

- Architectural concepts contributing to quick ETL experimentation:
  - Text based branching for ETL configuration (Git FTW)
  - Introduction of OO concepts to ETL
  - Caching based on hash of code and inputs
  - Continuous data ingestion

Summary

- You can write code with an Oncologist in the room

- Technology is as useful as its compatibility with:
  - People operating it:
    - Making specialists independent was key
  - Business requirement: pivoting product lines vs Performance / Robustness

Special Thanks

KEEP CALM

AND

KICK CANCER'S ASS

FLATIRON

Shameless plug: We are hiring!

Drop me a note at gil@flatiron.com or visit http://flatiron.com