Building a Data Provenance Tool with GroovyFX

Agenda

- Data Provenance
  - Overview - what is it/why do you care
  - Tool Demo
- Client Case Study
  - Functional Programming techniques
  - Code walk-thru
- GroovyFX
  - Structuring classes
  - Code walk-thru

Background

- 25+ years software engineering
  - C, C++, Java, Groovy, Perl (paid)
  - Groovy Eclipse Plugin
  - Lisp, Racket, Clojure, Haskell (hobby)
- Groovy since 2005
  - mostly large, legacy system rewrites
  - multi-year, multi-million dollar projects
  - systems/processes/algorithms too big to fit in my head
  - *this talk applies to these kinds of projects*

Data Provenance - Definition

Data Provenance captures where data came from and how it's been manipulated.

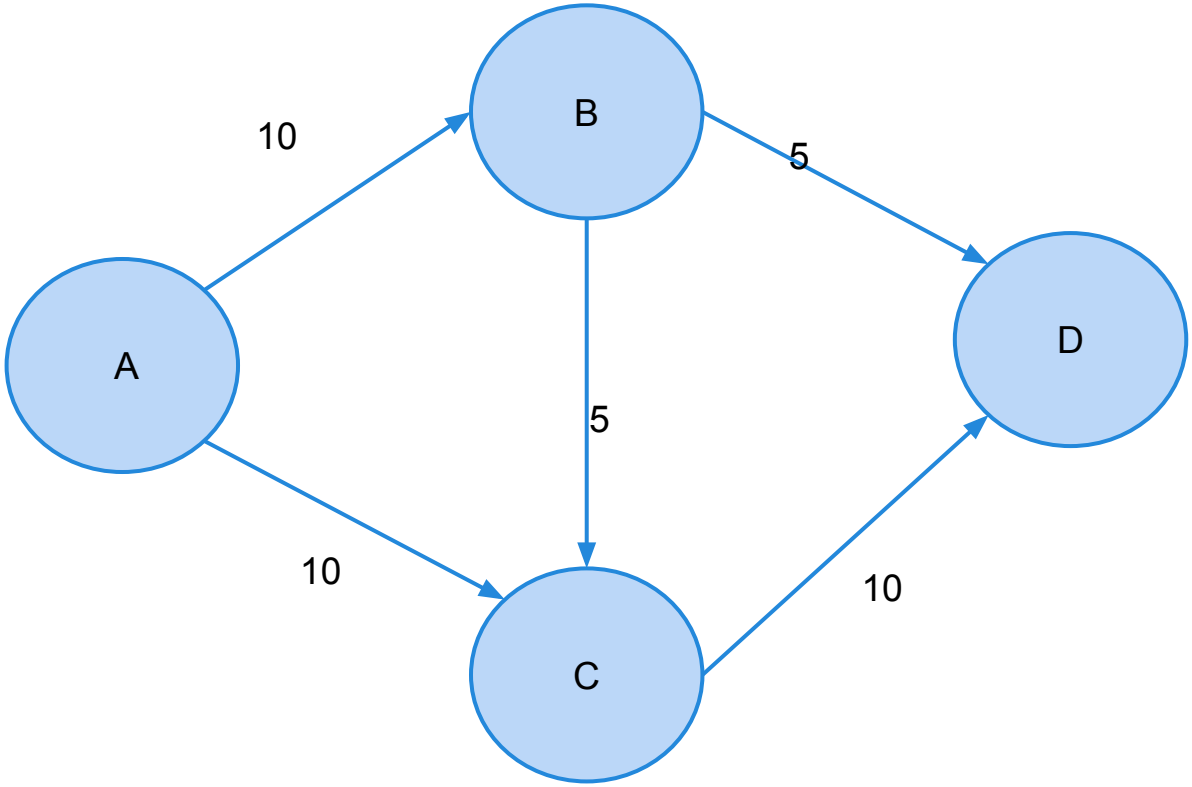
- the answer needs to be correct
  - *what the business needs*
- the answer needs to be explainable
  - *what the business wants*

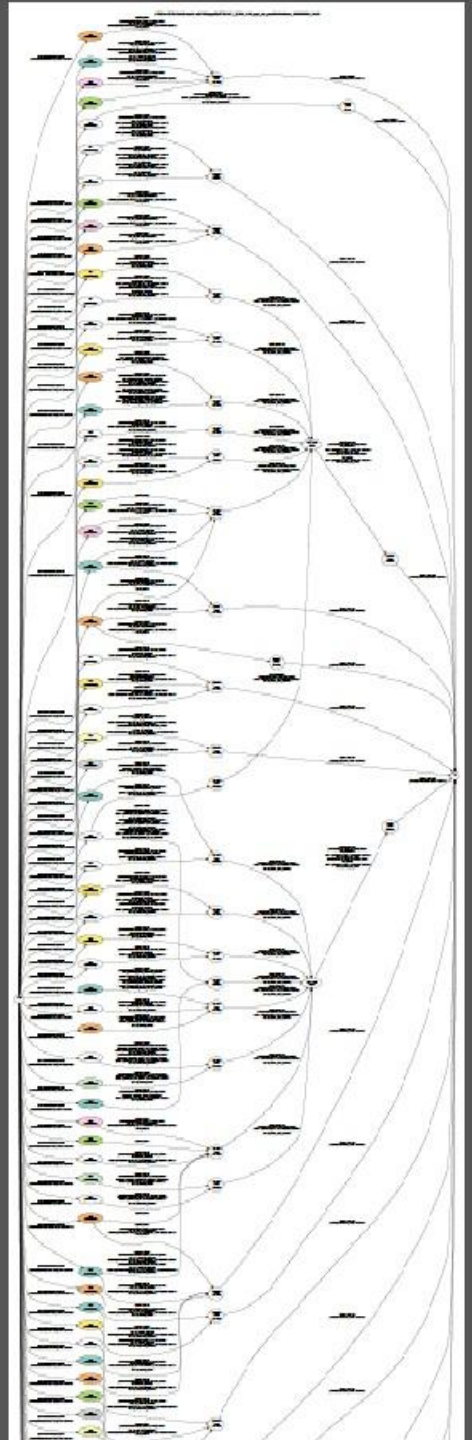
Example of Large System Rewrite

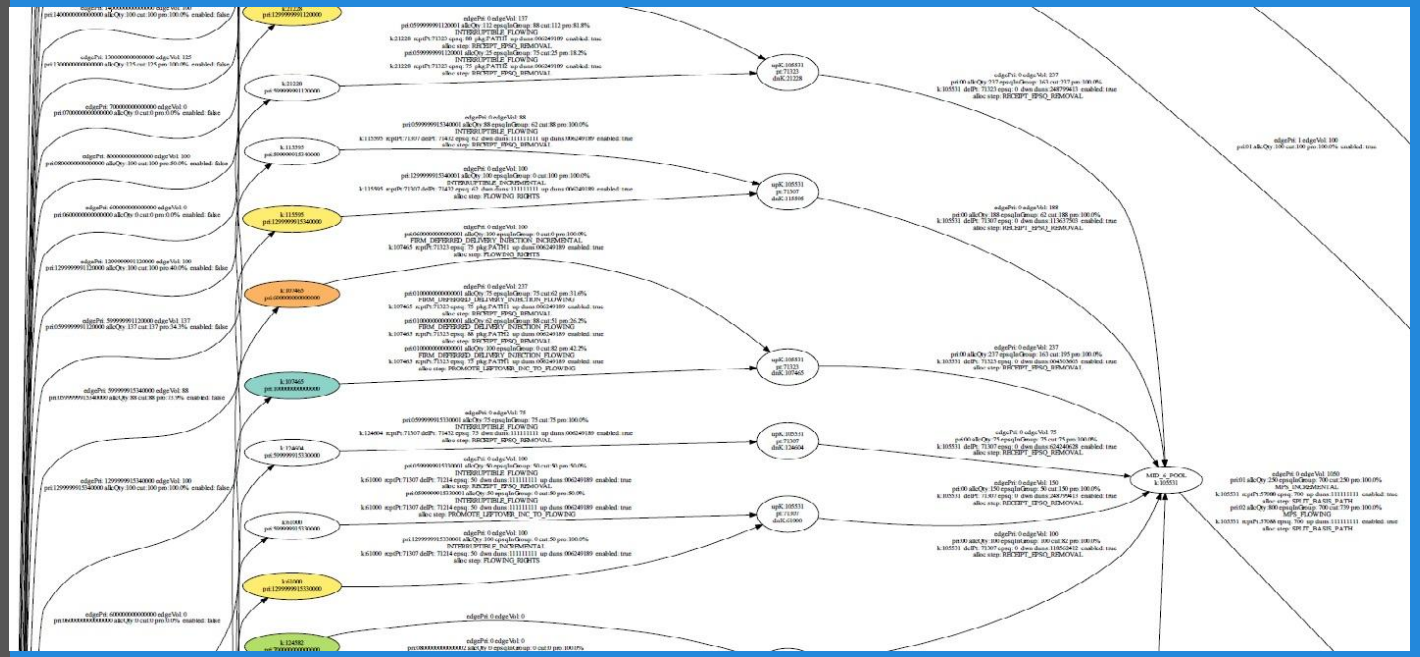
Prioritizing orders on a natural gas pipeline

- FERC requirements
  - fairness across contracts
- Client requirements
  - optimize utilization of the pipeline
  - fairness within contracts, user priorities
  - thousands of orders every day
    - orders can change in the middle of the day
    - sync with other pipelines several times a day
- Tool Demo

Maximize the Flow







Sample data of a test network. The networks are large, with many constraint points, lots of orders.

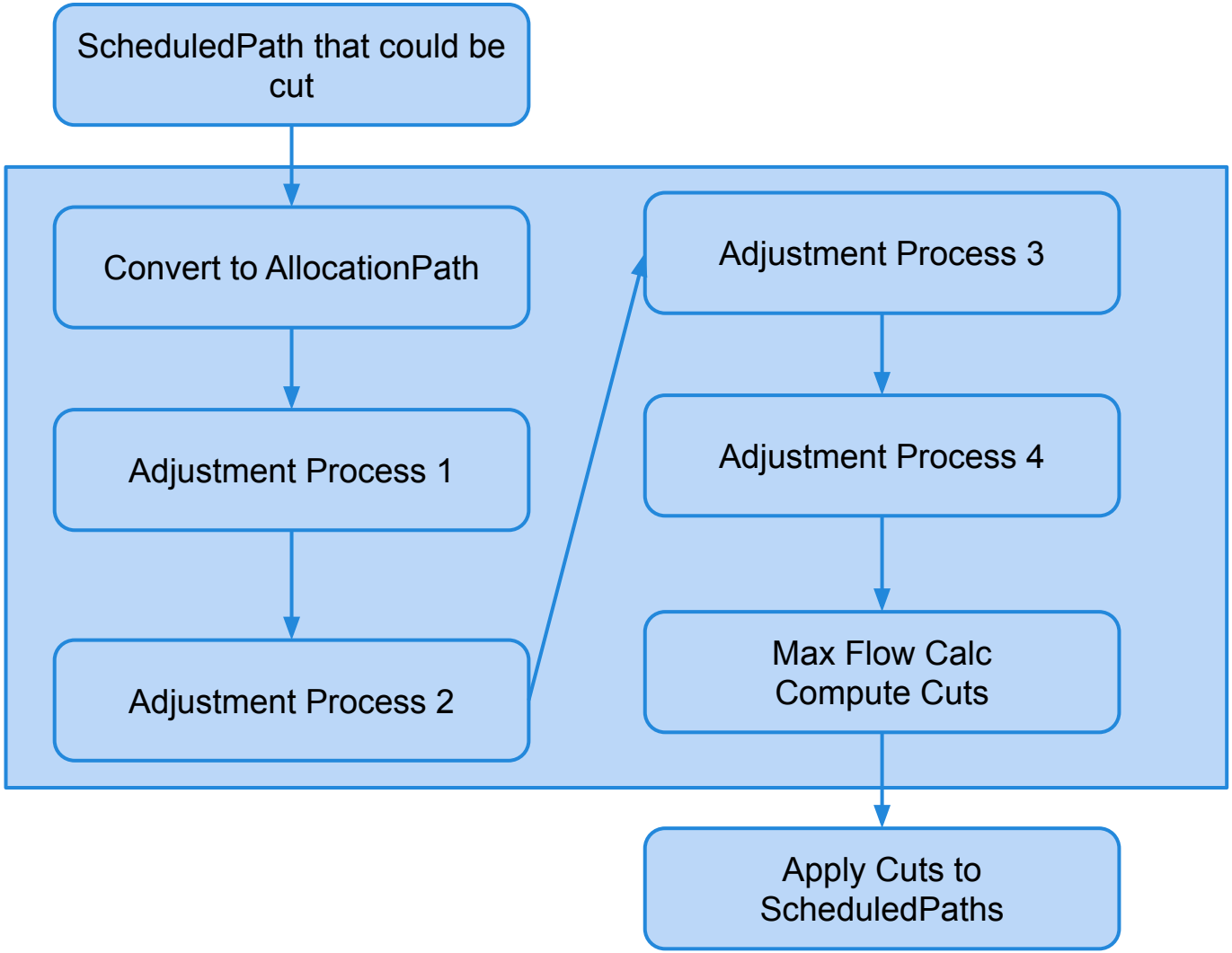
Why data provenance?

A human can't "at a glance" look at the network and tell you why an order was cut. Network is too large and there are too many business rules.

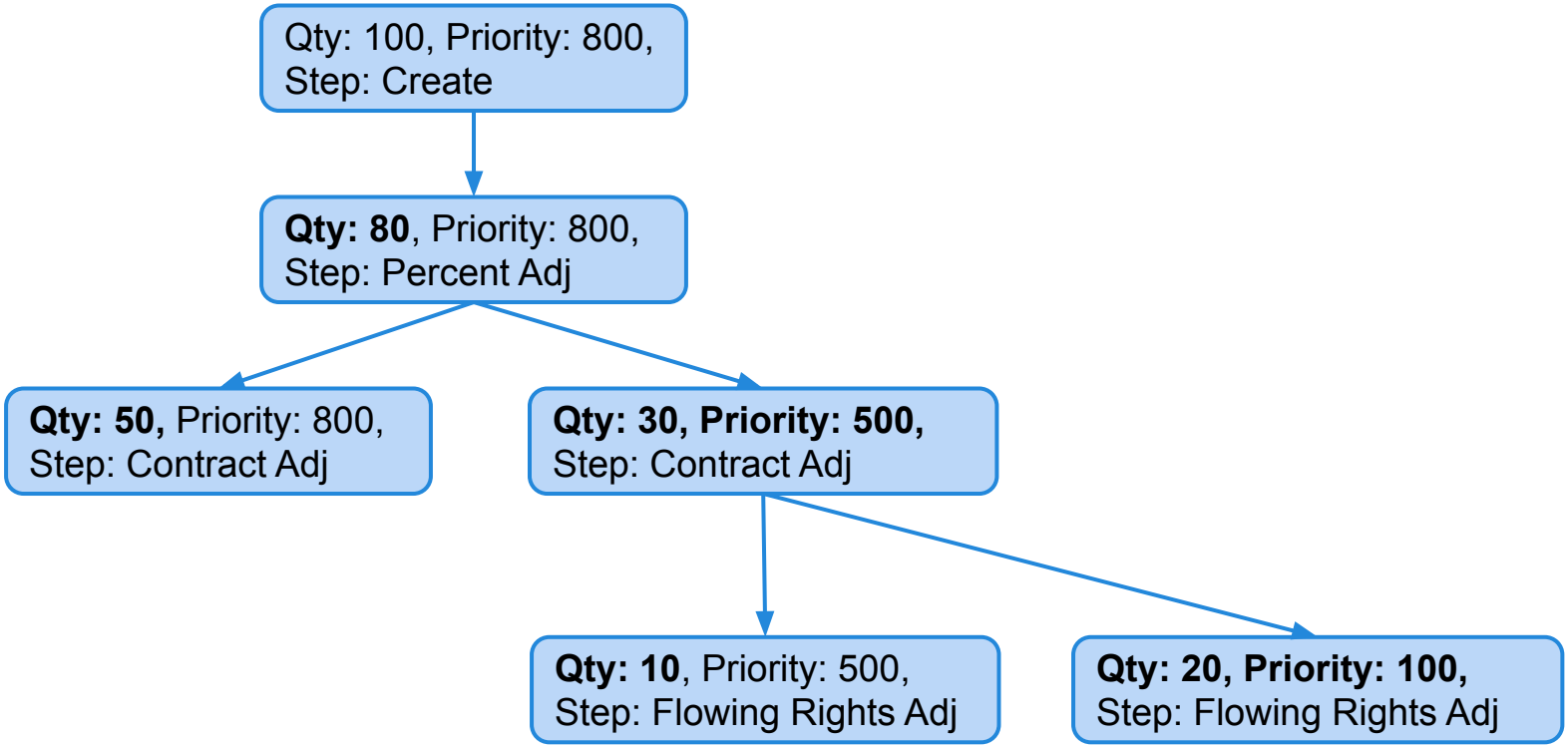
Data Provenance Implementation

- Original order ScheduledPath
  - receipt point, delivery point, contract(priority), quantity
- Convert to a new object, an AllocationPath with
  - cuttable quantity
  - numerical priority
  - allocation step
  - parent AllocationPath

Allocation Process



Example of the data



Data Provenance - The Code

- Functional approach
  - data classes are just data structures
  - 'function' classes are stateless
    - Spring singletons with injected stateless singletons
    - All required data for a computation is retrieved upstream, passed into the functions
- each step is complicated and lives in its own class

Data Provenance - The Code

- API into public method
  - takes a collection of AllocationPaths
  - returns a new collection of AllocationPaths
- Allocation Step classes
  - always create a new AllocationStep anytime priority or quantity changes
  - heavy use of collection classes, closures help thinking about steps as set transformations >*very prominent in functional programming*

Groovy Code Walk-thru

- Functional programming is a lot of map-filter-reduce
- Our code uses lots of findAll, collect, flatten (collectMany), collectEntries
- Orchestration class calls services that return updated collection of paths

GroovyFX

Model View Controller

- View
  - GroovyFX builders
- Model
  - backing model for GUI controls
  - data to be displayed
- Controller
  - widget event handlers
  - menu actions

View

- *main* file
  - calls GroovyFX.start closure, which initiates JavaFX startup
- mostly made up of GroovyFX builder code
- instantiates and wires up:
  - presentation model
  - handlers

Model

One class which ties together:

- backing data for the GUI
- the real data being read in by the tool

We use wrapper classes around the real data to capture GUI state i.e. isVisible

Controllers

Controllers are GUI event handlers, one class per type of event

- File Open
- Search Button
- Tree Item Selection

Instantiated from the attachHandlers() method in the main class

Demo / Code Walk-thru

- File Open -> FileEventHandler
- Select path -> PathSelectionHandler
- Search -> SearchButtonHandler

Resources

- "Transparent Accountable Data Mining: New Strategies for Privacy Protection" by Gerald Sussman et al., 2006
- Dierk Koenig's GroovyFX blogs
- Internet search 'Model View Presenter'
- "Tree View with a Data Source" blog: http://myjavafx.blogspot.com/2012/03/treeview-with-data-source.html