ddsflow - DDS-based workflow engine

Svetlana Shasharina*, Nanbor Wang, and James Matykiewicz

Tech-X Corporation

sveta@txcorp.com

STSci Sep 15-16, 2011

Background

- Collaboration with FNAL on data processing workflows and monitoring of workflows (originally for JDEM and LQCD). FNAL drove the requirements and are shaping the workflow language
  - Erik Gottschalk
  - Jim Kowalkowski
  - Marc Paterno
- Work is funded by Phase II SBIR from HEP/DOE
- Very much a work in progress
- Goals:
  - Identify the type of targeted workflows
  - Find ways to express them
  - See if Data Distribution Service is suitable for implementation
  - Develop a working system

Tech-X Corporation

- Small company, founded in 1994, 70 people, Boulder CO,
- Funding: 1/3 SBIR, 1/3 commercial, 1/3 federal grants
- Collaboration with DOE labs and NASA missions: exploration of new technologies and providing industry-standard solutions
- Distributed middleware
  - Data Distribution Service
  - Grid/Globus
  - CORBA
  - Web Services
- Physics simulations
- Mission critical systems (real time and embedded systems)
- Visualization (remote and application specific)
- Scientific workflows
- GPU

TECH-X CORPORATION

Common features of scientific workflows we are trying to address

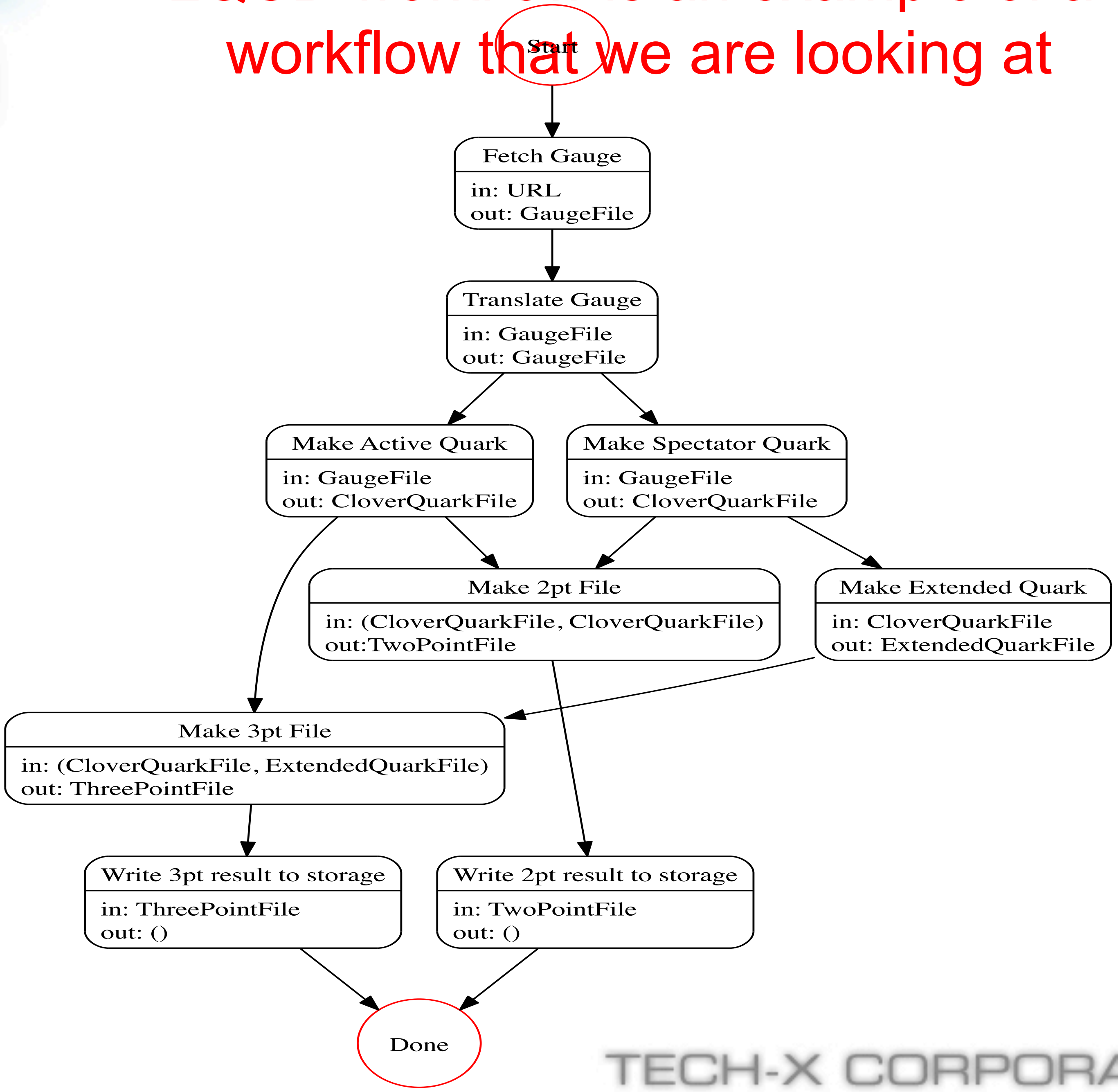
- Data driven
  - Actions triggered by data availability
- Dynamic
  - Resources allocated as needed dynamically and flow is changed depending on outcome and number of outputs
- Data management of transient data (without the need to specify locations and names)
- Expression for workflow should be editable: text/language based
  - GUI could be useful to set up a simple prototype or show the flow but one should be able to drop into an editable mode to set up things like loops, edit configurations
  - GUIs do not work well in distributed environments)

Workflow types can be defined using workflow patterns http://www.workflowpatterns.com/patterns/control/

Examples of control patterns

- Sequence
- And-split or parallel split (called map for identical threads)
- And-merge or synchronization (called reduce for identical threads)
- Or-split (exclusive)
- Or-merge (exclusive)

LQCD workflow is an example of a workflow that we are looking at

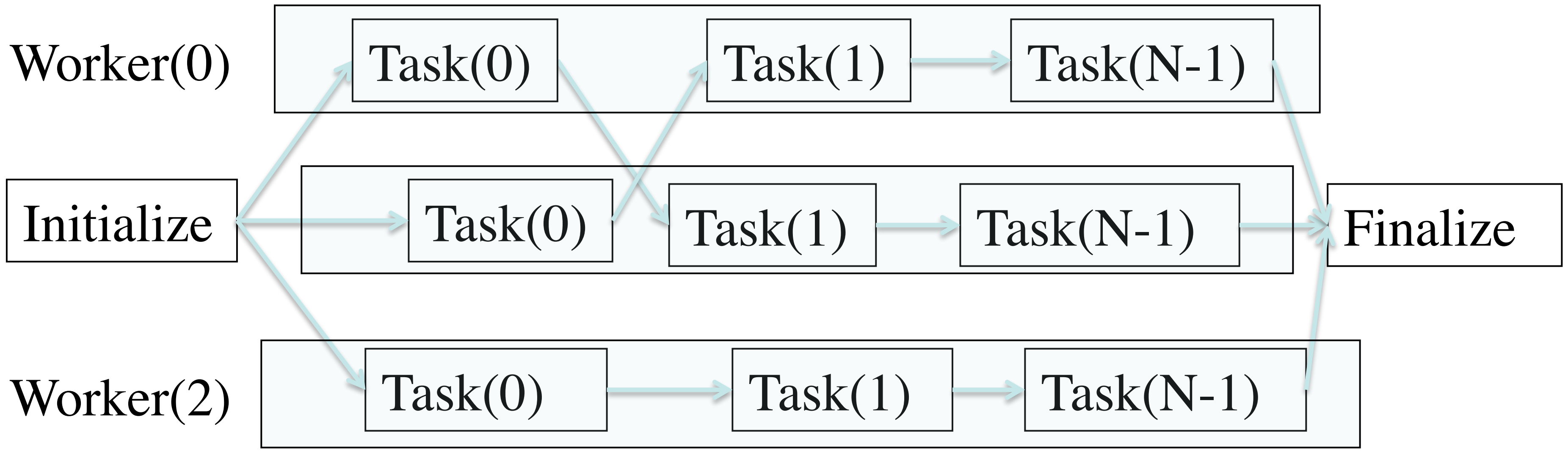


TECH-X CORPORATION

ddsflow: using control functions

- Actor - a software component that wraps an application:
  - Has “in” and “out” ports so that the connections could be verified for correct data exchange
  - Ports: file type (no names as connection between actors is specified and we have just one in and out port at the moment)
  - Paired with configuration
- Composite actor is created using the chosen control functions (this is the main difference from condor approach)
- On the initial stage we decided to limit the scope by using the following control functions (and extend as we go)
  - Sequence
  - Map (recursive) and and-split
  - Reduce and and-merge
- We use xml for workflow description (actors and their composition) and have an evolving xsd schema (need to map to PDL!)

ddsflow implements map and reduce (for now)



Tasks can be continued by different working processes: data can be passed between them (the Worker(1) performs Task(1) using data from Worker (0)). We might need to change that as data might not be on shared file system

Task-ordering within a sequence is enforced by the framework. However, sequences can proceed independent of each others.

Functional requirements of ddsflow

- Initialization phase (we are ready to process data after that)
  - Parse the workflow description and verify connectivity by checking in and out ports
  - Pair actors with configurations
  - Create workspace
  - Allocate initial memory and resources and instantiate all objects including tasks available at time = 0
- Operations phase (we are processing data)
  - Dynamically manage resources to accommodate the system dynamics and change the number of streams and tasks depending on outputs in previous tasks
  - Terminate if needed
  - Manage intermediate data
    - Create temporary output files with default file names and extensions derived from the file type
    - Clean them as step using them as output is over
  - Be able to kill the process
- Termination phase
  - Clean up memory
  - Shut down all processes.

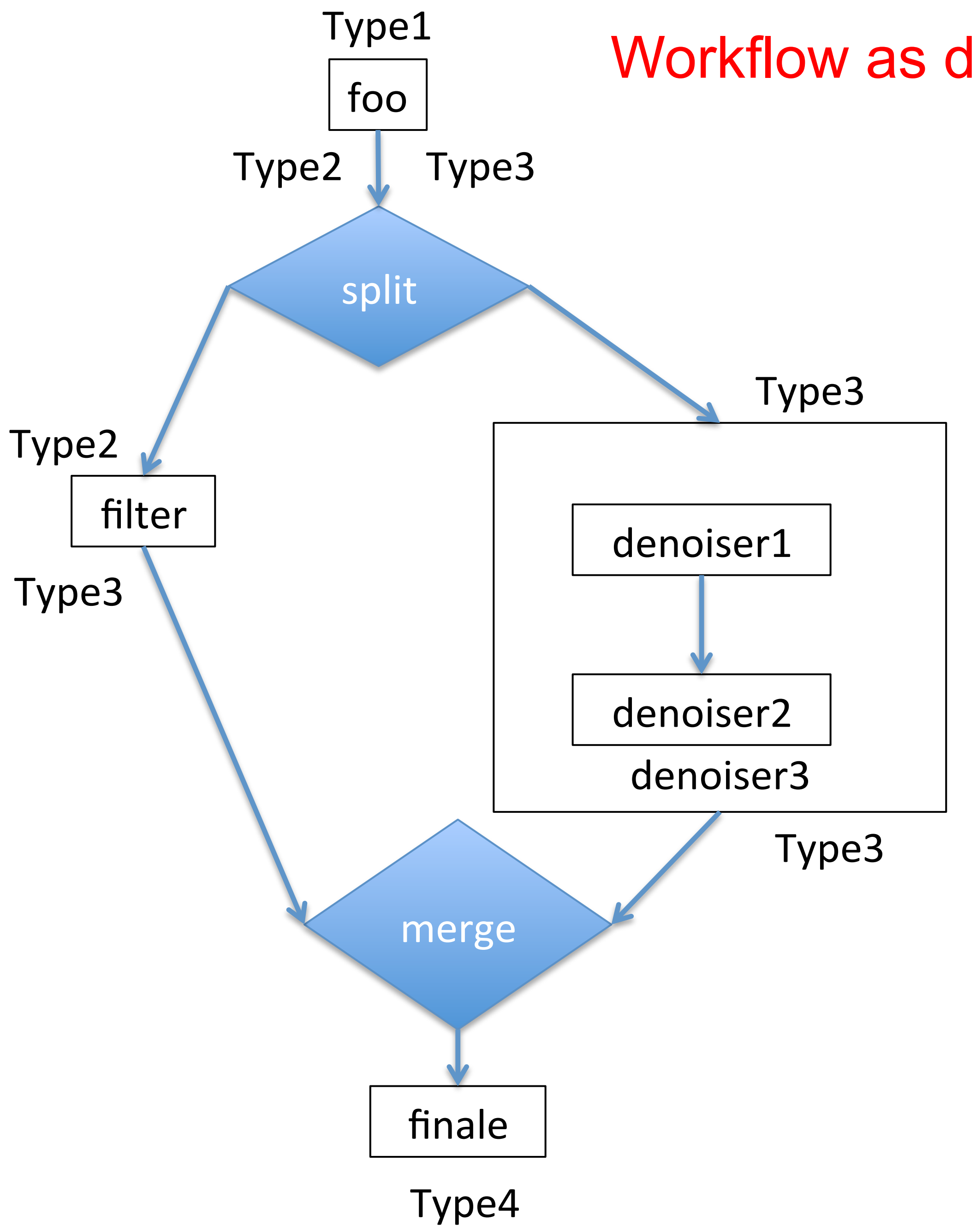
TECH-X CORPORATION

Simple actor: executable, in and out data types, configuration

```
<actor>
      <id>foo</id>
      <exec>foo.sh</exec>
      <inport>
        <type>Type1</type>
      </inport>
      <outport>
        <type>[Type2, Type3]</type>
      </outport>
      <config>foo.cfg</config>
  </actor>
```

Composite actors: control function and actors to operate on

```
<compositeActor>
  <type> sequence </type>
  <id> denoiser3 </id>
  <actors> denoiser1, denoiser2 </actors>
</compositeActor>
<compositeActor>
  <type> split </type>
  <id>splitter</id>
  <actors> filter, denoiser3</actors>
</compositeActor>
<compositeActor>
  <type> merge</type>
   <id>merger</id>
   <actors>splitter</actors>
</compositeActor>
<workflowExp> // Not sure about this?
  <actorId> fullworkflow </actorId>
    <indata> inputfilenames </indata>
    <outdata> outputfilenames </outdata>
<workflowExpr>
```

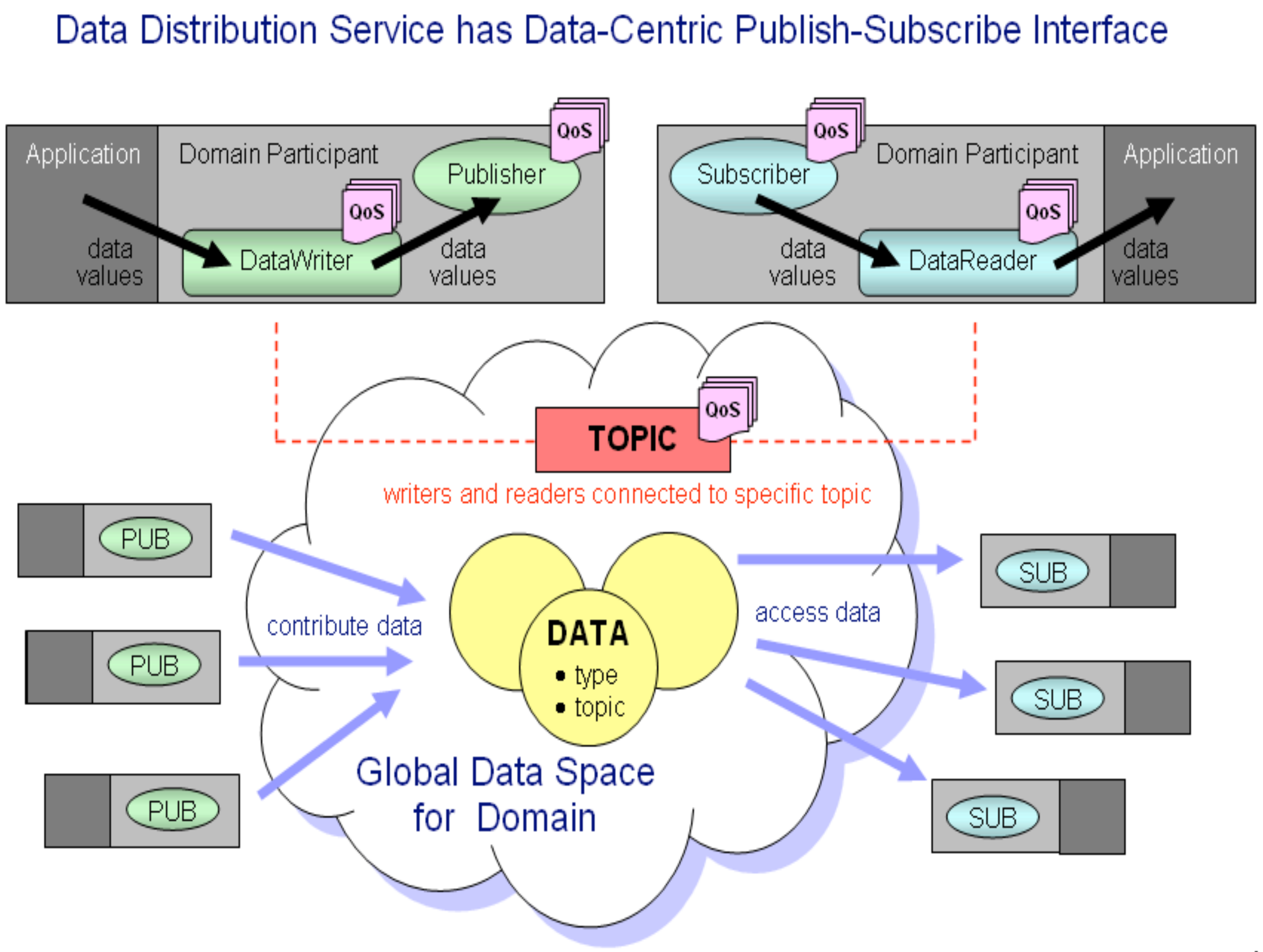


Workflow as detailed above

ddsfow uses uses Data Distribution Service (DDS)

- OMG standard for real time publish-subscribe systems
- Free implementations are available and good
- Rich quality-of-service features (resource and time aware, time sequences): to get just want to you want, how much you want and when you want
- Works over WAN
- Security support
- C++ and Java binding
- Tested in DOD applications (military ships, air traffic control)
- Topics – events with data (can be used to exchange data in memory, we tried FITS data)

DDS decouples participants and delivers information of interest



ddflow elements

- Initializer
  - Splits data, possibly creates workers and workspaces, publishes (for all initial work tickets with correct specification of the workflow
- Finalizer
  - Cleans and shuts down
- TupleSpaceServer (blackboard)
  - Changes status in task tickets in accordance with the workflow order: once a worker reports that task n done, a ticket for task n-1 with <n-1 in-data> = <n out-data> is changed to ready and worker topic is published (with the worker id next in the queue). Once a worker reports that is doing this work, the status is changed to running etc.
  - Has a queue of idle workers in the order they reported that they are done
  - Recording each state provides full provenance
- Workers
  - Work on any assigned ticket (number of workers is not related to number of tasks)
  - Publish their status
  - Listen to task assignment (matching its id to the one in the worker ticket)

ddsflow plans

- Extend implementation to more control functions and recursive maps (now all ~works for map-reduce systems)
- Test if we can implement given examples (LQCD, for example)
- Add error and status information management
- Performance studies
- Investigate use of functional languages to define workflows
- Intervention based on data quality or other status events

Functional languages provide building blocks to create composite actors

- Functional language treat y = f(x) as a function definition, not assignment. Thus y is a functor (function that can be treated as an object). Predefined methods (for example, map) allow easy creating of composite actors. FLs allow
  - Unbounded stream instead of one argument so a function gets applied to a list (x, followed by remainder)
  - Lazy evaluation: happens only when the elements are available (a must if you are dealing with streams or in dynamic flows when we do not know the number of outcomes in a step)
- Concerns
  - Do we need the full featured a full programming language? (functional languages are not “simple”)

Some functions under consideration (FNAL suggestion)

- Map applies a function to every element of a list:

map (f, [x_0, x_1, …, x_{N-1}]) = [f (x_0), …, f (x_{N-1})]

- Reduce collapses a list into a single value by applying a binary operator * cumulatively from left to right:

reduce (*, [x_0, x_1, x_2, x_3])=((x_0*x_1)*x_2)*x_3.

- Scan accumulates all the intermediate results of reduce operation:

scan (*, [x_0, x_1, $x_{2, \ldots}$]) = [x_0, x_0*x_1, x_0*x_1)*x_2, …].

- Filter selects elements of the list for which a function applied returns true:

filter (f, [x_0, x_1, …, x_{N-1}])= [x_i, …] for all i for which f (x_i) = true.

- Zip merges two lists into one with elements that are tuples from the original lists:

zip [a,b,c...] [z,y,x...] = [(a,z), (b,y), (c,x)...]

- Unzip takes a list of tuples, and breaks them apart into a tuple of lists

unzip [(a,b),(c,d)] = ([a,c], [b,d])

- Cycle is the same as map but until some condition is true.

How simple map (no reduce) workflow would look like

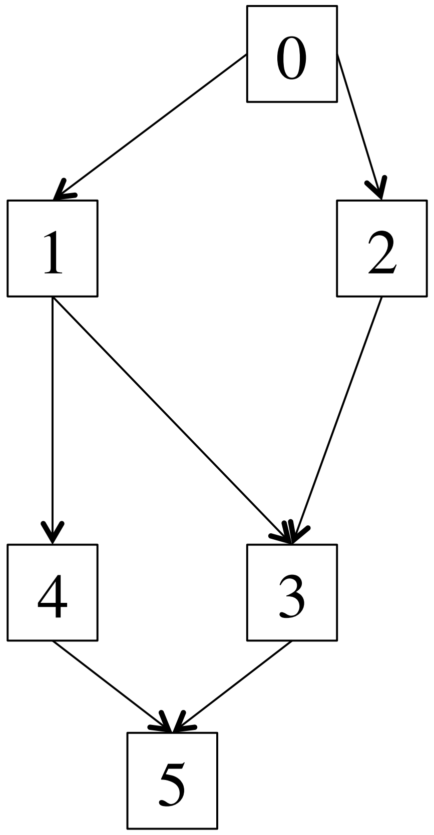
```
ss_split :: Spectrum -> [Spectrum]
ss_generate :: Spectrum -> Image
ss_combine :: [Image] -> Image
spectra = ss_split input_spectrum
images = map ss_generate spectra
result = ss_combine images
OR
my_workflow :: Spectrum -> Image
my_workflow is = ss_combine (map ss_generate (ss_split is))
```

Message

- Requirements or ddsflow were developed by working with scientists to identify their needs
- We found that the requirement are very similar in LQCD and JDEM
- The key is to figure out what kind of workflows are needed and how people would like to interact with the workflow system
- Next step for us is to develop a tool for creating such workflows from scratch, or clone and modify them
  - The tool should be easy to use

Extra slides

LQCD features DAGS and we need to describe them using control functions. At some point: DAG parser?



$$\begin{tikzcd}
 & 0 \arrow[dl] \arrow[dr] & \\
1 \arrow[d] \arrow[dr] & & 2 \arrow[dl] \\
4 \arrow[dr] & 3 \arrow[d] & \\
 & 5 &
\end{tikzcd}$$