Cascading

www.cascading.org

info@cascading.org

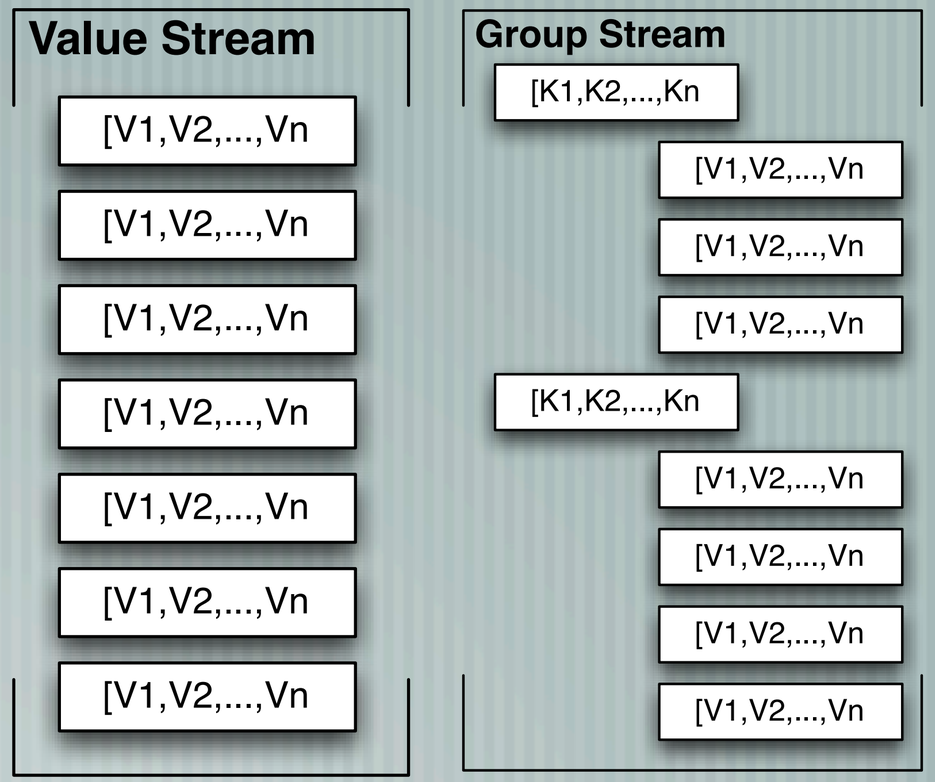
Design Goals

- Make large processing jobs more transparent
- Reusable processing components independent of resources
- Incremental “data” builds
- Simplify testing of processes
- Scriptable from higher level languages (Groovy, JRuby, Jython, etc)

Cascading Introduction

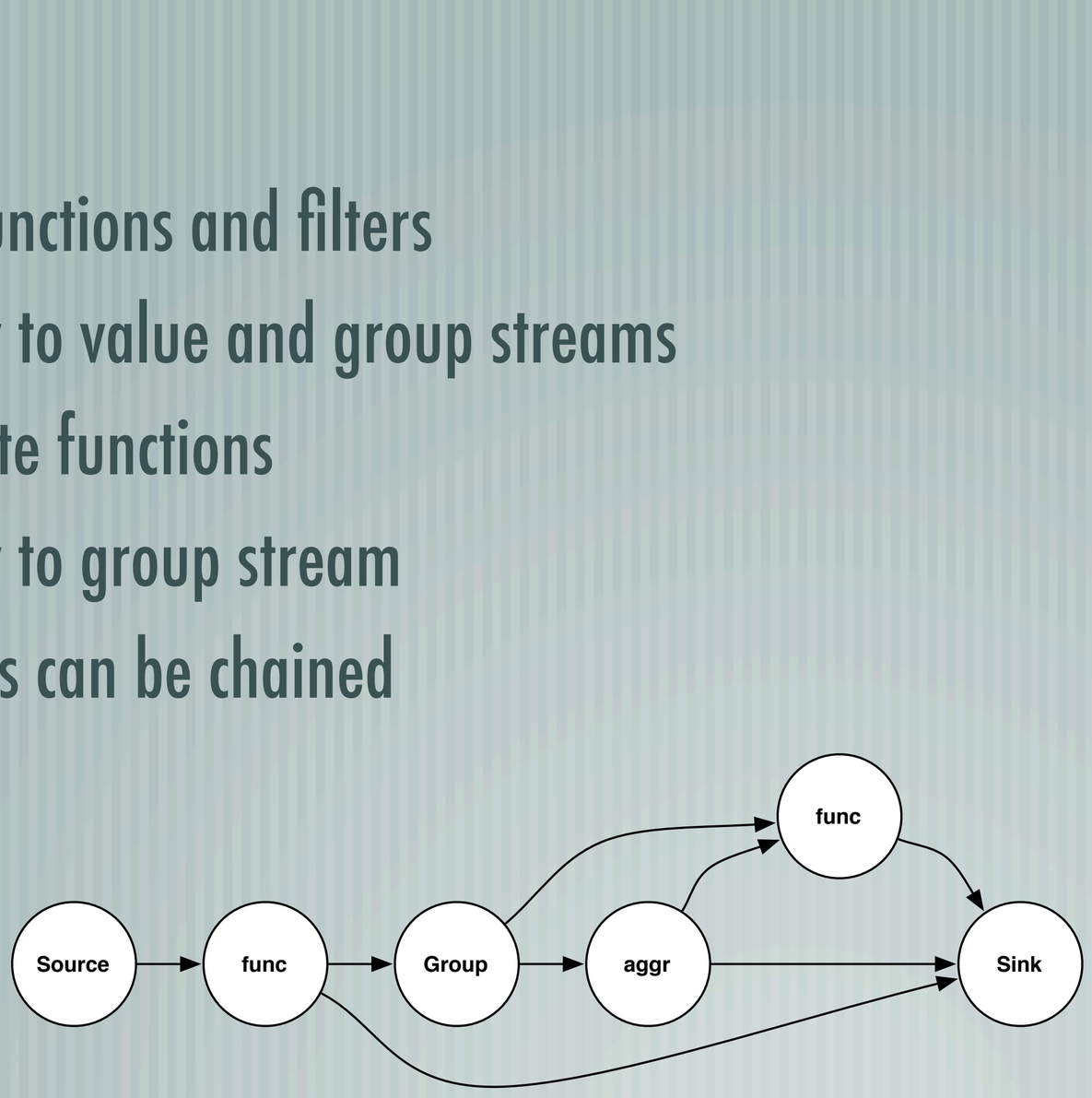
Tuple Streams

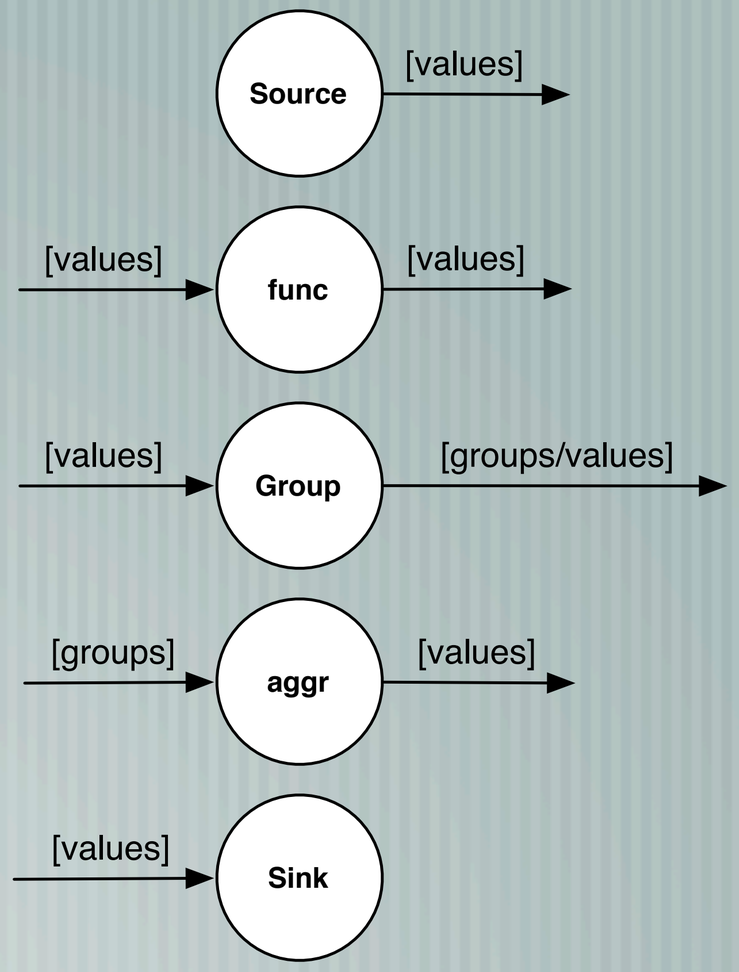
- Tuple
  - A set of ordered data ["John", "Doe", 39]
- Value Stream
  - Just tuples
- Group Stream
  - Tuples groups by a key



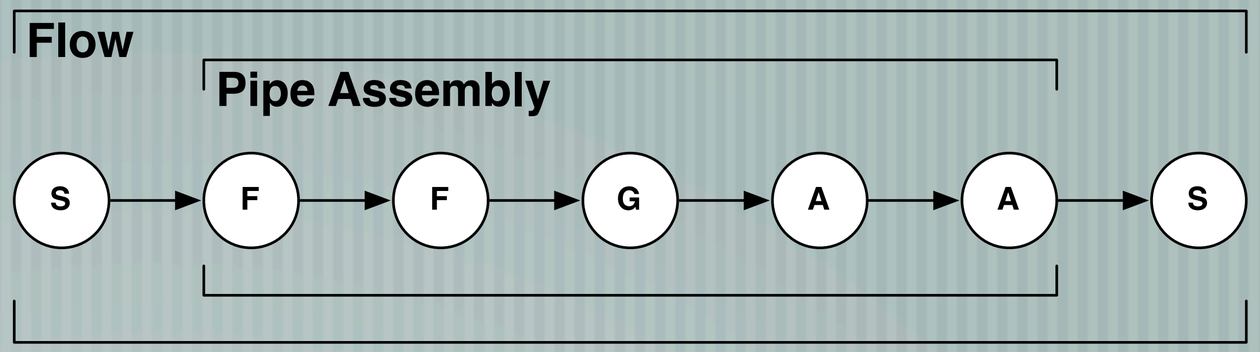
Tuple Streams

- Scalar functions and filters
  - Apply to value and group streams
- Aggregate functions
  - Apply to group stream
- Functions can be chained

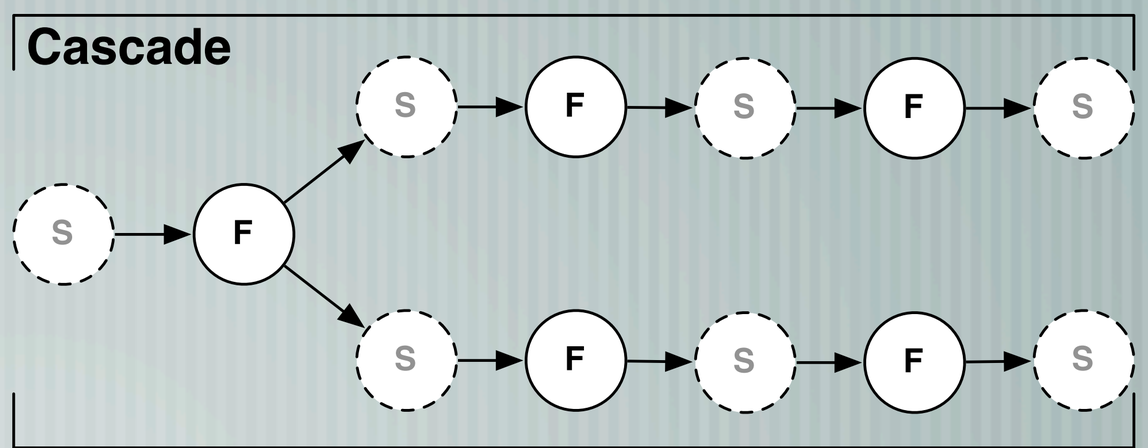




Stream Processing

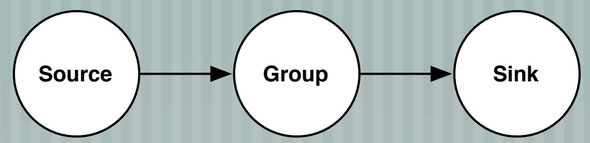


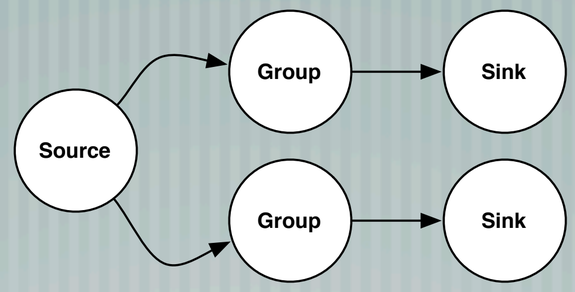
- Pipe Assemblies
  - A chain of scalar functions, groupings, aggregate functions
  - Reusable, independent of data source/sink
- Flows
  - Assemblies plus sources and sinks
- Cascades
  - A collection of Flows

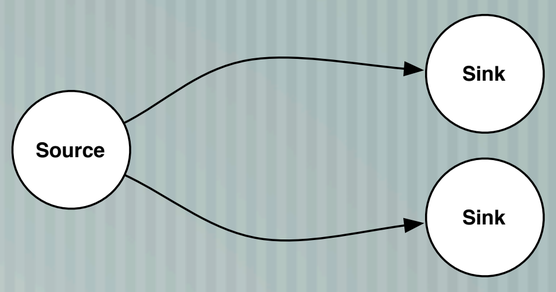


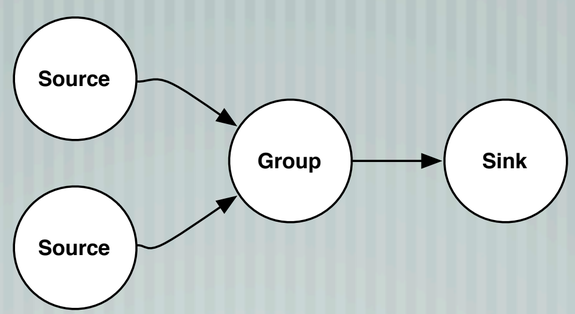
Processing Patterns

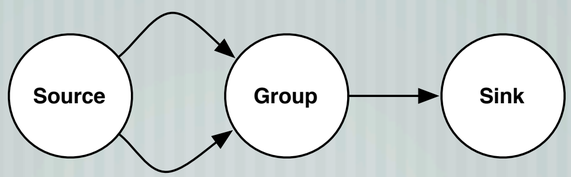
- Chain
- Splits
- Joins
- Cross



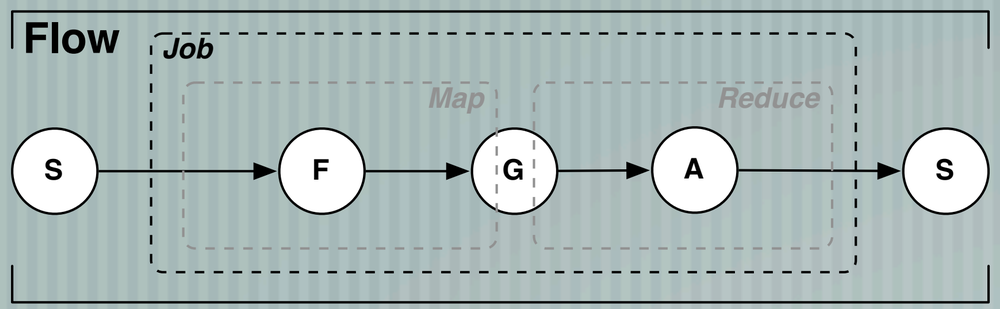


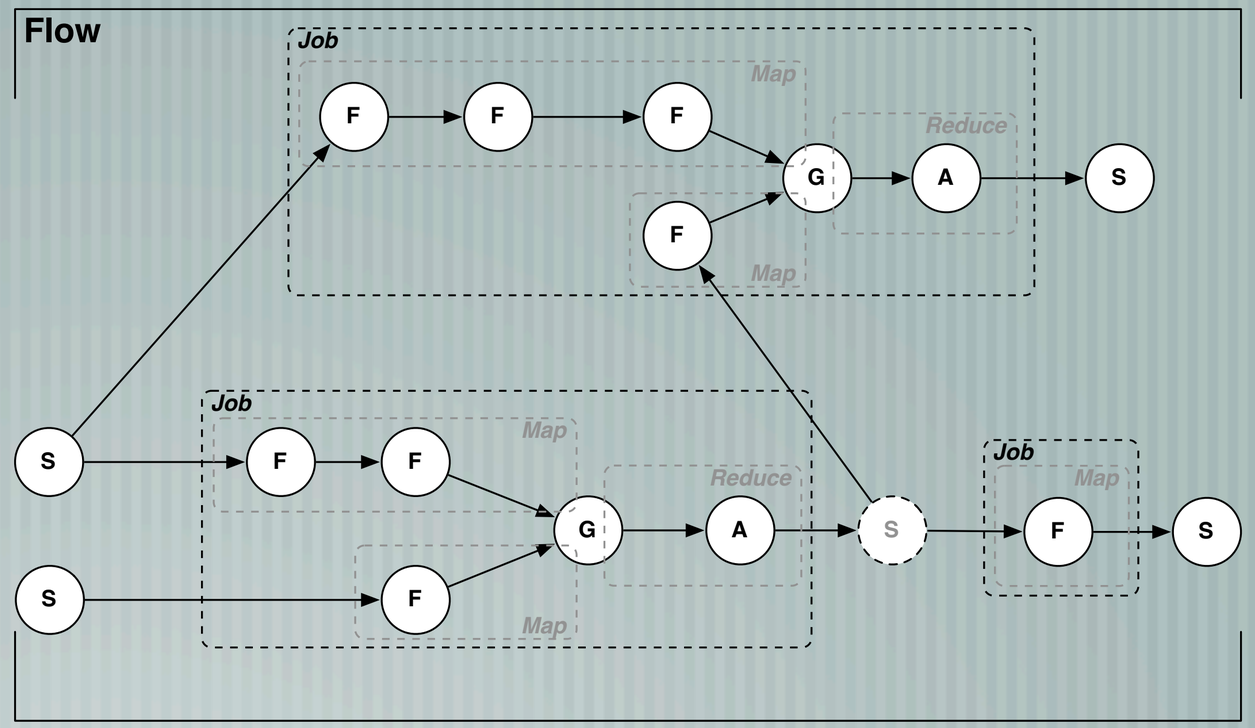






MapReduce Planner





- Flows are logical ‘units of work’
- Flows ‘compiled’ into MR Jobs
- Intermediate files are created (and destroyed) to join Jobs

Topological Scheduler

- Flows walk MapReduce Jobs in dependency order
- Cascades walk Flows in dependency order
- Independent Jobs and Flows are scheduled to run concurrently
- Listeners can react to element events (notify completion or failures)
- Only stale data-sets are rebuilt (configurable)

Scripting - Groovy

```
Flow flow = builder.flow("wordcount")
  {
    source(input, scheme: text()) // input is filename of raw text document

    tokenize(/[.,]*\s+/) // output new tuple for each split, result replaces stream by default
    group() // group on stream
    count() // count values in group, creates 'count' field by default
    group(["count"], reverse: true) // group/sort on 'count', reverse the sort order

    sink(output)
  }

flow.complete() // execute, block till completed
```

System Integration

- FileSystems (unique to Cascading)
  - Raw file S3 reading/writing (MD5)
  - Raw file HTTP reading (MD5)
  - Zip files
  - Can bypass native Hadoop ‘collectors’
- Event notification via listeners (XMPP/SQS/Zookeeper notifications)
- Groovy scripting for easier local shell/file operations (wget, scp, etc)

Cascading API & Internals

Core Concepts

- Taps and Schemes
- Tuples and Fields
- Pipes and PipeAssemblies
- Each and Every Operators
- Groups
- Flows, FlowSteps, and FlowConnectors
- Cascades, and CascadeConnectors, optional

Taps and Schemes

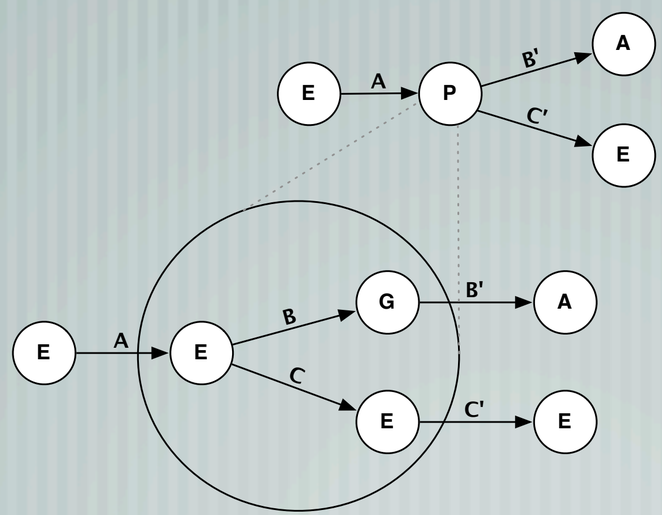
- Taps, abstract out where and how a data resources is accessed
  - hdfs, http, local, S3, etc
- Taps, used as Tuple (data) stream sinks, sources, or both
- Schemes, define what a resource is made of
  - text lines, SequenceFile, CSV, etc

Tuples and Fields

- Tuples are the ‘records’, read from Tap sources, written to Tap sinks
- Fields are the ‘column names’, sourced from Schemes
- Tuple class, an ordered collection of Comparable values
  - (“a string”, 1.0, new SomeComparableWritable())
- Fields class, a list of field names, absolute or relative positions
  - (“total”, 3, -1) // fields ‘total’, 4th position, last position

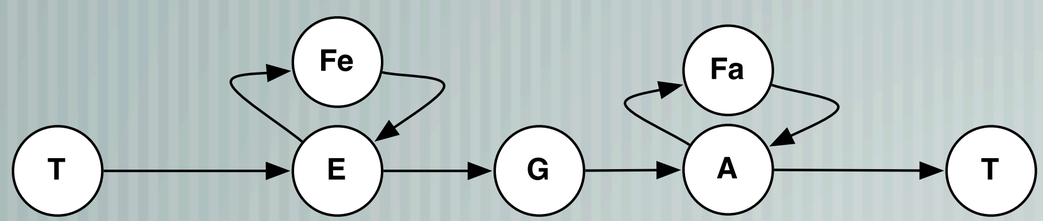
Pipes and PipeAssemblies

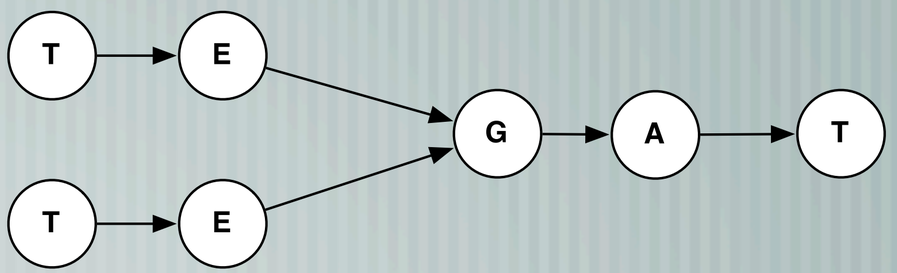
- Tuple streams pass through Pipes to be processed
- Pipes, apply functions, filters, and aggregators to the Tuple stream
- Pipe instances are chained together into assemblies
- Reusable assemblies are subclasses of class PipeAssembly



Group Class and Subclasses

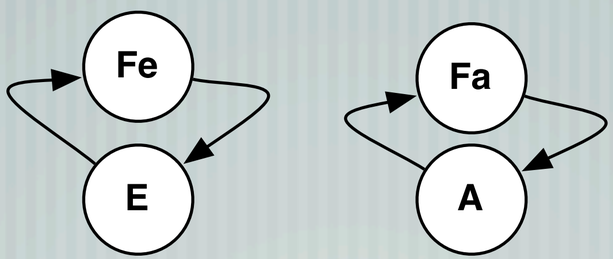
- Group, subclass of Pipe, groups the Tuple stream on given fields
- GroupBy and CoGroup subclass Group
- GroupBy groups and sorts
- CoGroup performs joins





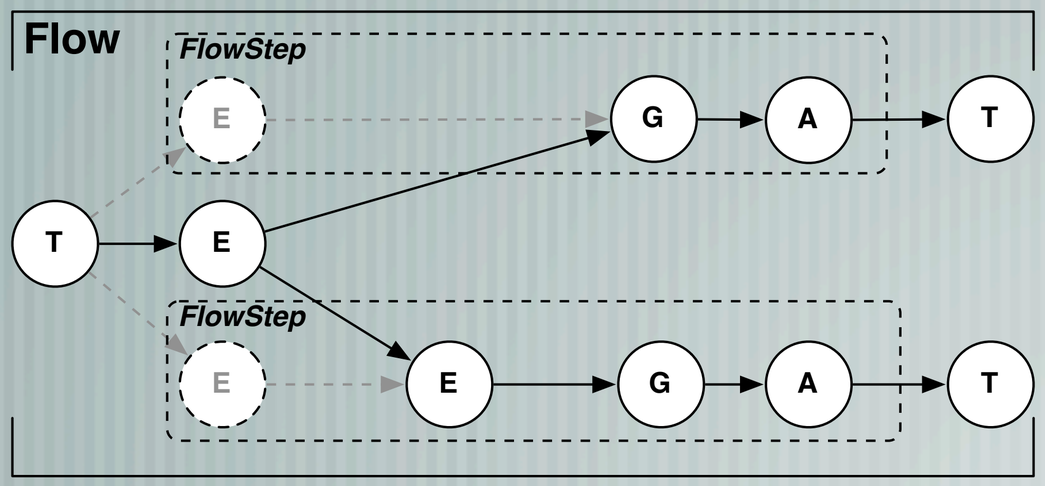
Each and Every Classes

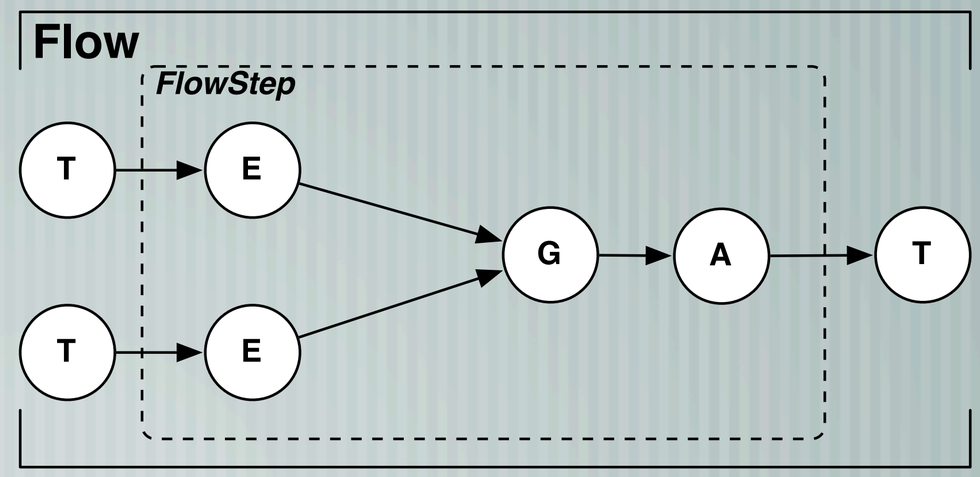
- Each, subclass of Pipe, applies Functions and Filters to each Tuple instance
  - (a,b,c) -> Each(func()) -> (a,b,c,d)
- Every, subclass of Pipe, applies Aggregators to every Tuple group
  - (a: b,c) -> Every(agg()) -> (a,d: b,c)



Flows and FlowConnectors

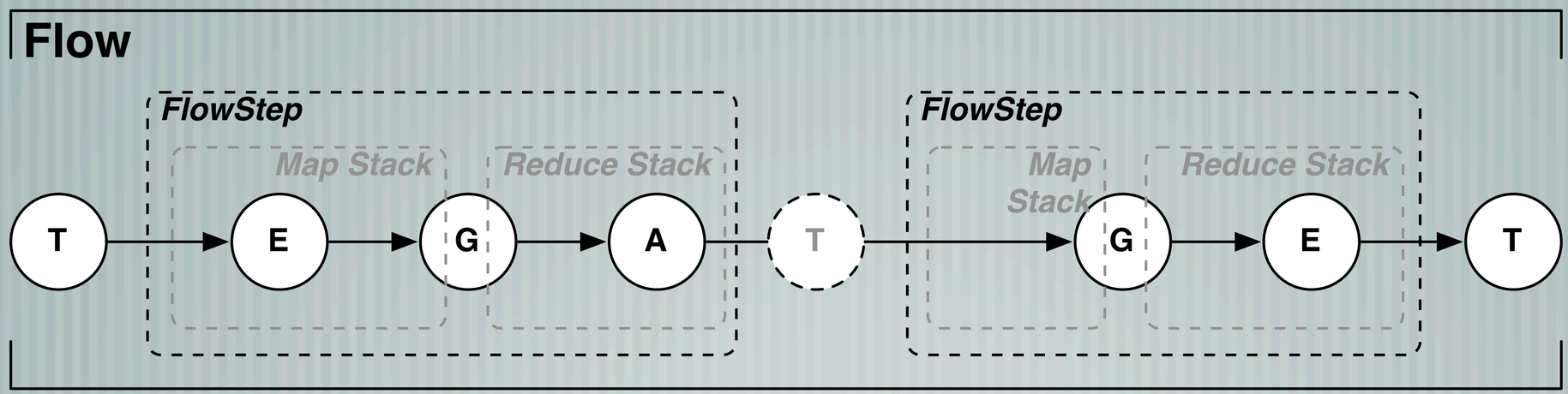
- Flows encapsulate assemblies and sink and source Taps
- FlowConnectors connect assemblies and Taps into Flows





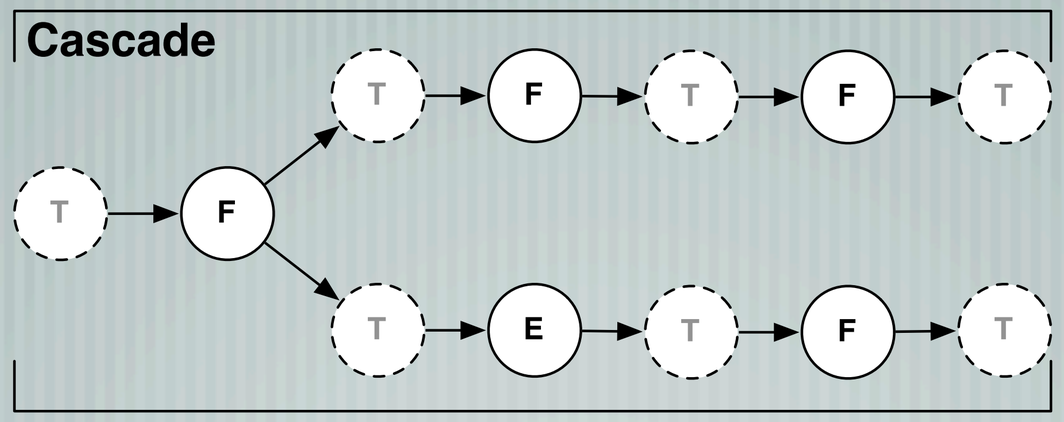
FlowSteps and FlowConnectors

- Internally, FlowConnectors ‘compile’ assemblies into FlowSteps
- FlowSteps are MapReduce jobs, which are executed in Topo order
- Temporary files are created to link FlowSteps



Cascades and CascadeConnectors

- Are optional
- Cascades bind Flows together via shared Taps
- CascadeConnectors connect Flows
- Flows are executed in Topo order



Syntax

- Each(previous, argSelector, function/filter, resultSelector)
- Every(previous, argSelector, aggregator, resultSelector)
- GroupBy(previous, groupSelector, sortSelector)
- CoGroup(joinN, joiner, declaredFields)
- Function(numArgs, declaredFields, ...)
- Filter (numArgs, ...)
- Aggregator(numArgs, declaredFields, ...)