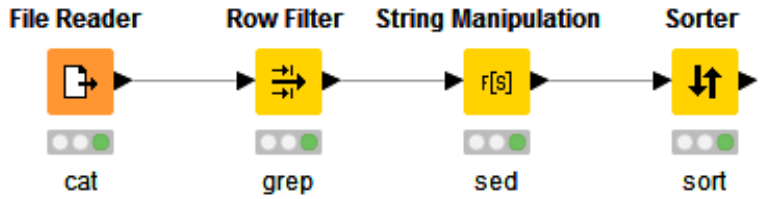
Streaming API

Streaming Overview

- Notion of (unix) pipes:




```
cat | grep "blah" | sed "s/foo/bar/g" | sort | …
```

- No blocking (nodes execute simultaneously)
- Processed rows are instantly provided as input to downstream node(s)
- I/O and memory only for few rows "in transit"
- Can be parallelized (also in the cloud)

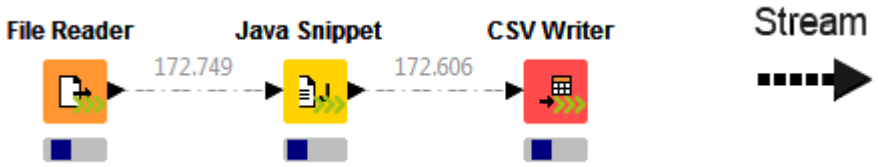
Streaming API – Status Quo

- Streaming API is in place (pending, i.e. still subject to change)
- Implemented by more than 100 nodes
- Simple Streaming Executor (Job Manager) in Labs
  Current limitations: no Loop- and Metanode-support
- Streaming Test Executor

- Available since v2.6 (July 2012)

Simple Streaming Job Manager – Facts

- All nodes execute concurrently
- Each node's execution in separate thread(s)
  → multiple threads/remote processes when node runs distributed (not implemented)
- Threads blocking when pulling data from input / pushing data to output



Implement Streaming API

- NodeModel as main entry point; it defines:
  - Role of its input ports: distributable/streamable
  - Role of its output ports: distributable
  - A StreamableOperator (a.k.a execute)
    - `runFinal(PortInput[] inputs, PortOutput[] outputs, ExecutionContext ctx)`
  - A MergeOperator – only if necessary, e.g.
    - input is distributable, output is not
    - passing warning messages from remote to local
- See online API

Implement Streaming API - StreamableOperator

```
return new StreamableOperator() {
        void runFinal(PortInput[] inputs, …) {
                RowInput rowInput = (RowInput) inputs[0];
                RowOutput rowOutput = (RowOutput)outputs[0];
                DataRow row;
                while((row = row.poll())!=null) {
                        rowOutput.push(row);
                }
                rowInput.close();
                rowOutput.close();
        }
}
```

- If input port *i* is STREAMABLE → **`inputs[i] instanceof RowInput`**, otherwise not!
- If at least one input port is DISTRIBUTABLE, all non-distributed **`inputs[i]==null!`**

Invocation Path

1) internals = NodeModel.createInitialStreamableOperatorInternals() - **master**
2) mergeOperator = NodeModel.createMergeOperator() (can be null) - **master**
3) while NodeModel.iterate (internals) - **master**
   - 3a) NodeModel.loadSettings(nodesettings) - **distributed** (factory for operator)
   - 3b) NodeModel.createStreamableOperator(PartitionInfo) - **distributed**
   - 3c) StreamableOperator.loadInternals(internals) - **distributed**
   - 3d) StreamableOperator.runIntermediate() - **distributed**
   - 3e) partialInternals= StreamableOperator.saveInternals() - **distributed**
   - 3f) internals = mergeOperator.mergeIntermediate (partialInternals) - **hierarchical**
     (merge intermediate results such as intermediate cluster centers or possible values)
4) NodeModel.computeFinalOutputSpecs(internals, upstreamNodeSpecs) - **master**
5) streamableOperator = NodeModel.createStreamableOperator() - **distributed**
6) streamableOperator.loadInternals(internals) - **distributed**
7) streamableOperator.runFinal(…) - **distributed**
   (such as assignments of input patterns to final cluster centers)
8) internals= mergeOperator.mergeFinal (partialInternals) - **hierarchical**
   (merge final results such as Bayesian Parameters or counts of patterns per cluster)
9) NodeModel.finishStreamableOperators(internals, RowOutput[]) - **master**
10) NodeModel.saveInternals - **master**
11) NodeModel.loadInternals - **client**

Invocation Path – Minimal Implementation Requirements

(only streaming, not distributed, without additional iterations)

1) internals = NodeModel.createInitialStreamableOperatorInternals() - **client (master??)**
2) mergeOperator = NodeModel.createMergeOperator() (can be null) - **master**
3) while NodeModel.iterate (internals) - **master**
   - 3a) NodeModel.loadSettings(nodesettings) - **distributed** (factory for operator)
   - 3b) NodeModel.createStreamableOperator(PartitionInfo) - **distributed**
   - 3c) StreamableOperator.loadInternals(internals) - **distributed**
   - 3d) StreamableOperator.runIntermediate() - **distributed**
   - 3e) partialInternals= StreamableOperator.saveInternals() - **distributed**
   - 3f) internals = mergeOperator.mergeIntermediate (partialInternals) - **hierarchical**
     (merge intermediate results such as intermediate cluster centers or possible values)
4) NodeModel.computeFinalOutputSpecs(internals, upstreamNodeSpecs) - **master**
5) streamableOperator = NodeModel.createStreamableOperator() - **distributed**
6) streamableOperator.loadInternals(internals) - **distributed**
7) streamableOperator.runFinal(…) - **distributed**
   (such as assignments of input patterns to final cluster centers)
8) internals= mergeOperator.mergeFinal (partialInternals) - **hierarchical**
   (merge final results such as Bayesian Parameters or counts of patterns per cluster)
9) NodeModel.finishStreamableOperators(internals, RowOutput[]) - **master**
10) NodeModel.saveInternals - **master**
11) NodeModel.loadInternals - **client**
    + getInputPortRoles() & getOutputPortRoles()

Streaming API - Examples

- Streamable Function: 1 in, 1 out, no rows added or removed (String Manipulation, Math Formula, …)
  - String2DateNodeModel.java
- Row Filter, Lag Column: Single Scan algorithms
  - RowFilterNodeModel.java
- More Complex Algorithms: k-means, Naïve Bayes
  - de.unikn.knime.stud.fillbrunn.distributedkmeans.DistributedKMeansNodeModel

Streamable API - TODOs

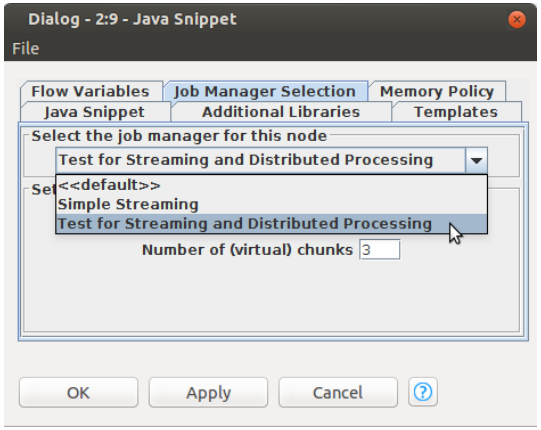
- Usability (= error handling)
- Make more nodes streamable and distributable
- Support loops and metanodes (Simple Streaming)
- Run nodes distributed
- …

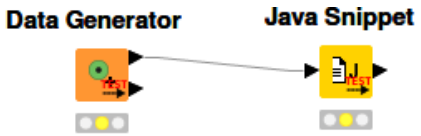
Streaming API - Pitfalls

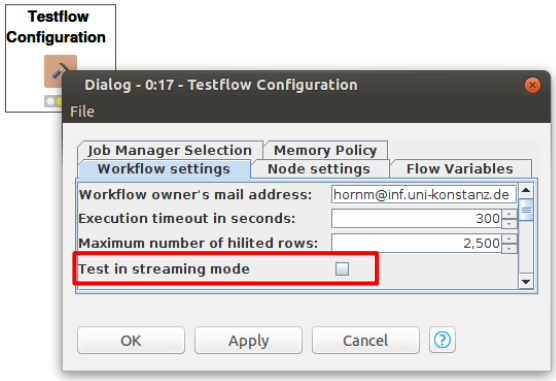
- NodeModel.configure() returns null if streaming API is implemented
- Missing getInportRole(), getOutportRole() implementations
- No RowOutput.close() or RowInput.close()
- If at least one port is distributed: all non-distributed ports have to be treated in NodeModel.finishStreamableExecution(…)

Streaming API - Testing

- Streaming Test Executor (comes with KNIME 3.2, part of the Testing-Extension)
- Checks (almost) all Streaming-API-methods (including distributed execution)
- Run as workflow test (additional test with the Test Streaming Executor set on ALL nodes)







Streaming API – Filter Streamable Nodes

By overriding createStreamableOperator(…) nodes are assumed to be ‘streamable’