User composes a stream-processing program in terms of “operators”: result is a (reverse) tree, with ≥1 source nodes and rooted in a single sink node.
StreamMap
Apply a function to transform each input event

\[ \lambda_{in} \rightarrow \text{function} \rightarrow \lambda_{out} = \lambda_{in} \]

We'll go through the 8 operators briefly and show how they may be represented in a Jackson network model.
The operator description are taken from the README at striot.org.
StreamScan
Apply a function to transform each input event, taking into account past history

\[ \lambda_{\text{in}} \rightarrow \lambda_{\text{out}} = \lambda_{\text{in}} \]
**StreamFilter**
Generates a stream containing only those events that meet a user-defined criteria

\[ \lambda_{\text{out}} = f \cdot \lambda_{\text{in}} \]

\[ \lambda_{\text{out}'} = (1 - f) \cdot \lambda_{\text{in}} \]

- \( f \) is the selectivity of the operator
- \( \lambda_{\text{out}'} \) is the rate of the discarded events
- To balance the network equations we've introduced a "phantom" output
StreamFilterAcc
Filter events, taking into account past results

\[ \lambda_{\text{out}} = f \cdot \lambda_{\text{in}} \]

\[ \lambda'_{\text{out}} = (1 - f) \cdot \lambda_{\text{in}} \]

\( f \) is the selectivity of the operator
\( \lambda'_{\text{out}} \) is the rate of the discarded events
StreamWindow chop
Create non-overlapping windows that are a fixed number (N) events in length

\[ \lambda_{in} \rightarrow \text{Window} \rightarrow \lambda_{out} = \lambda_{in}' = (N - 1) \]

The modelling of streamWindow depends upon the user-supplied "window maker" parameter. Here we discuss four example window makers.
StreamWindow chopTime

Create non-overlapping windows that are a fixed time (T) length

\[ \lambda_{in} \rightarrow \rightarrow \lambda_{out} = \lambda_{in} - \lambda_{out}' \]
StreamWindow sliding
Create overlapping windows that are a fixed number of events in length

\[ \lambda_{\text{in}} \rightarrow \text{Circle} \rightarrow \lambda_{\text{out}} = \lambda_{\text{in}} \]
StreamWindow slidingTime
Create overlapping windows that are a fixed time length

$\lambda_{\text{in}} \rightarrow \lambda_{\text{out}} = \lambda_{\text{in}}$
StreamExpand
Generate an output event for each element of a list in the input event

\[ \lambda_{\text{in}} \rightarrow \lambda_{\text{out}} = N \cdot \lambda_{\text{in}} \]

\[ \lambda_{\text{in}}' = (N - 1) \cdot \lambda_{\text{in}} \]

The dual of windowing
"N" is the average number of elements in a list
Here to balance the network equations we've introduced a "phantom" input
StreamMerge
Combine ≥2 streams by merging their events

\[ \lambda_{\text{out}} = \lambda_{\text{in}}^{a} + \lambda_{\text{in}}^{b} \]

Here representing two inputs but could be more
StreamJoin
Combine two streams by creating a tuple for each pair of input events

\[ \lambda^A_{in} \rightarrow \text{Tuple} \rightarrow \lambda_{out} = \lambda^A_{in} \]

\[ \lambda^B_{in} \rightarrow \text{Tuple} \rightarrow \lambda_{out'} = \lambda^A_{in} \]

This one is a work in progress

\( \lambda_{Ain} = \lambda_{Bin} \)
The design of Striot means we are able to rewrite a stream-processing program: rewrites preserve the functional behaviour but may change other aspects (e.g. performance)

Here we have replaced two serially-arranged “filter” nodes with a single “filter” which performs the work of both of the original operators.
Before fusion

\[ \lambda_{\text{in}} = f_a \cdot \lambda_{\text{in}} \]

\[ \lambda_{\text{b}} = f_a \cdot \lambda_{\text{in}} \]

\[ \lambda_{\text{out}} = f_b \cdot \lambda_{\text{b}} \]

This slide is busier than I'd like! The input and output arrival rates and the selectivity of each operator are known.
What are the properties of the new Server, performing the work of the two older Servers?
We are confident that we know the output rates and the filter selectivity: the product of the input selectivities

But what of the mean service rate?

Intuitively Since the new server is doing the work of the two original servers, it seems the mean average service time should be the sum of the mean average service times for the two servers. This does not model any efficiency gains (or losses)
Generalised question

“Can we represent a Jackson network as a single server and what are the properties of that server?”

I've seen the technique of applying Little's Theorem to a sub-part of a Jackson network, but Little's Theorem does not appear to help us determine the mean service rate because it is not defined in terms of service rates.