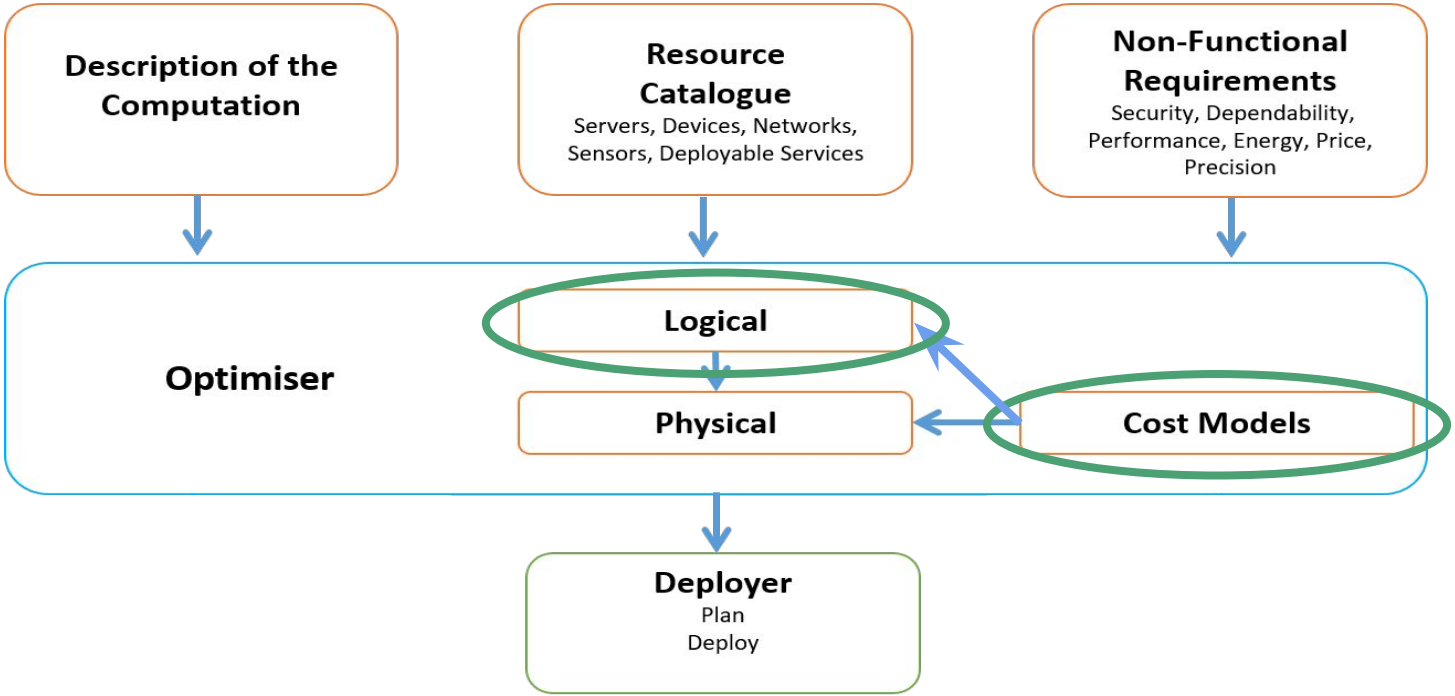
Picking a winner

cost models for evaluating stream-processing programs

Jonathan Dowland <jon.dowland@ncl.ac.uk>

UK Systems ‘21

StrIoT

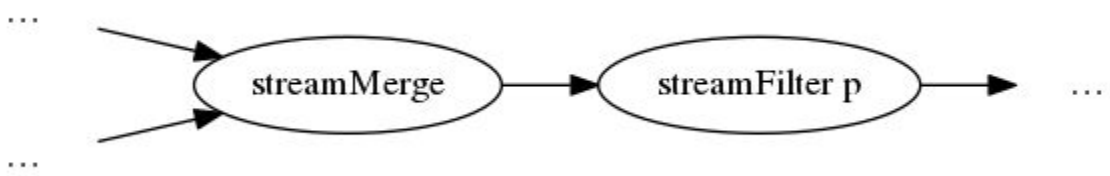


https://github.com/striot/striot

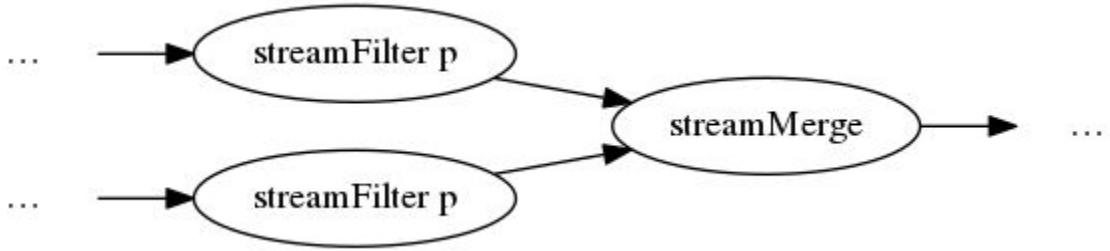
StrIoT operators

Filter	streamFilter	α → α
	streamFilterAcc	α → α
Map	streamMap	α → β
	streamScan	α → β
Window	streamWindow	α → [α]
	streamExpand	[α] → α
Combine	streamMerge	[α] → α
	streamJoin	α → β → (α,β)

Logical Optimiser: term-rewriting




```
streamFilter p (streamMerge [s1,s2…])
  = streamMerge [ streamFilter p s1,
                  streamFilter p s2, … ]
```



Rewrite rule implementation

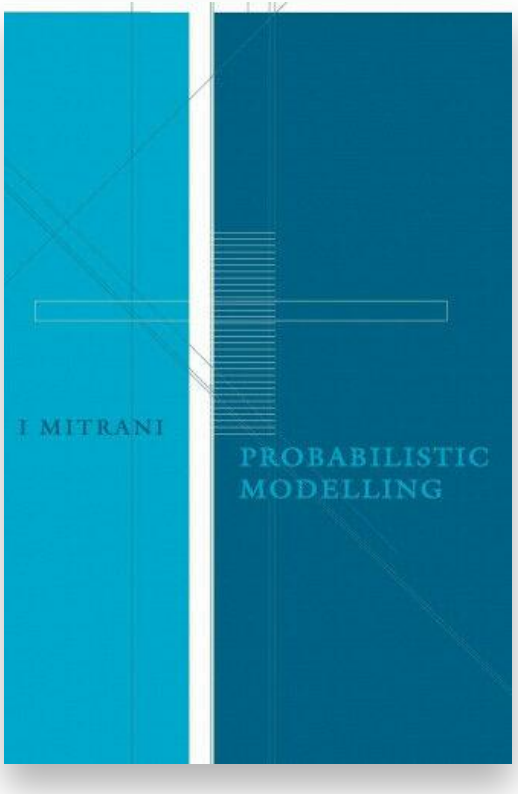
```
-- streamFilter f >>> streamFilter g = streamFilter (\x -> f x && g x)

filterFuse :: RewriteRule
filterFuse (Connect (Vertex a@(StreamVertex _ Filter (p:_) _ _))
                    (Vertex b@(StreamVertex _ Filter (q:_) _ _))) =

let c = a { parameters = [[| (\p q x -> p x && q x) $(p) $(q) |]] }

in Just (removeEdge c c . mergeVertices (`elem` [a,b]) c)
```

Cost models for evaluation

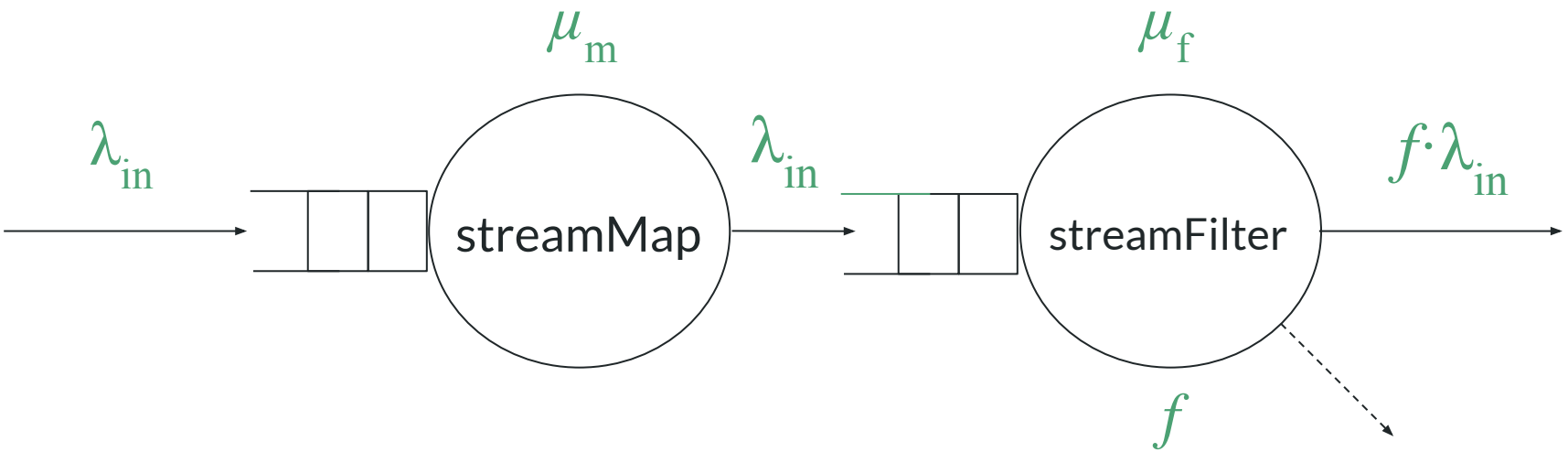
Queuing system model

Mitrani, I. (1997). *Probabilistic Modelling*. Cambridge: Cambridge University Press. doi:10.1017/CBO9781139173087.001

Utilisation (ρ) = arrival rate (λ) / service rate (μ)



Modelling StrIoT operators



$1 - f \cdot \lambda_{in}$

Encoding queueing theory properties

```
data StreamVertex = StreamVertex
{ vertexId :: Int
, operator :: StreamOperator
, parameters :: [ExpQ]
, intype :: String
, outtype :: String
                          }

data StreamOperator = Map | Filter
| Expand | Window | Merge | Join | Scan
| FilterAcc
| Source
| Sink deriving (Show,Ord,Eq)
```

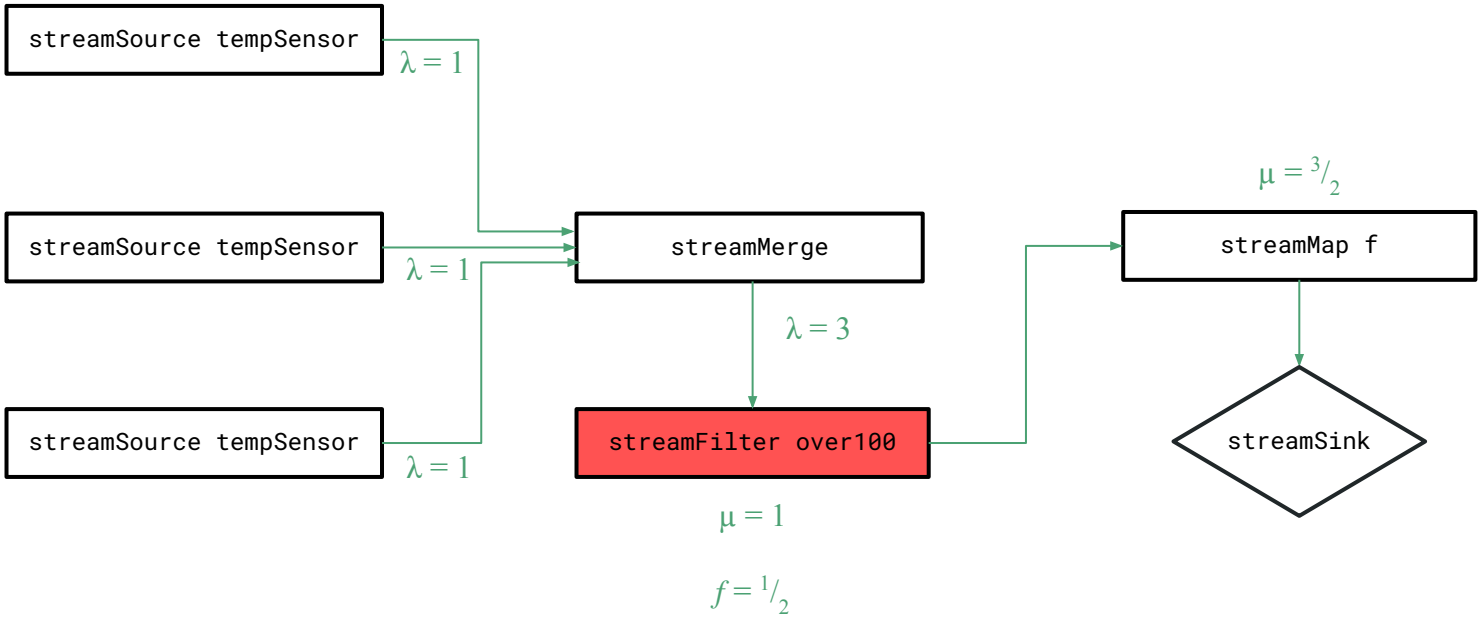
Re-write rules and queueing theory

```
-- streamFilter f >>> streamFilter g = streamFilter (\x -> f x && g x)

filterFuse :: RewriteRule
filterFuse (Connect (Vertex a@(StreamVertex _ (Filter sel1) (p:_) _ _ s1))
                    (Vertex b@(StreamVertex _ (Filter sel2) (q:_) _ _ s2))) =

let c = a { operator = Filter (sel1 * sel2)
          , parameters = [[| (\p q x -> p x && q x) $(p) $(q) |]]
          , serviceTime = s1 + (sel1 * s2) }

in Just (removeEdge c c . mergeVertices (`elem` [a,b]) c)
```

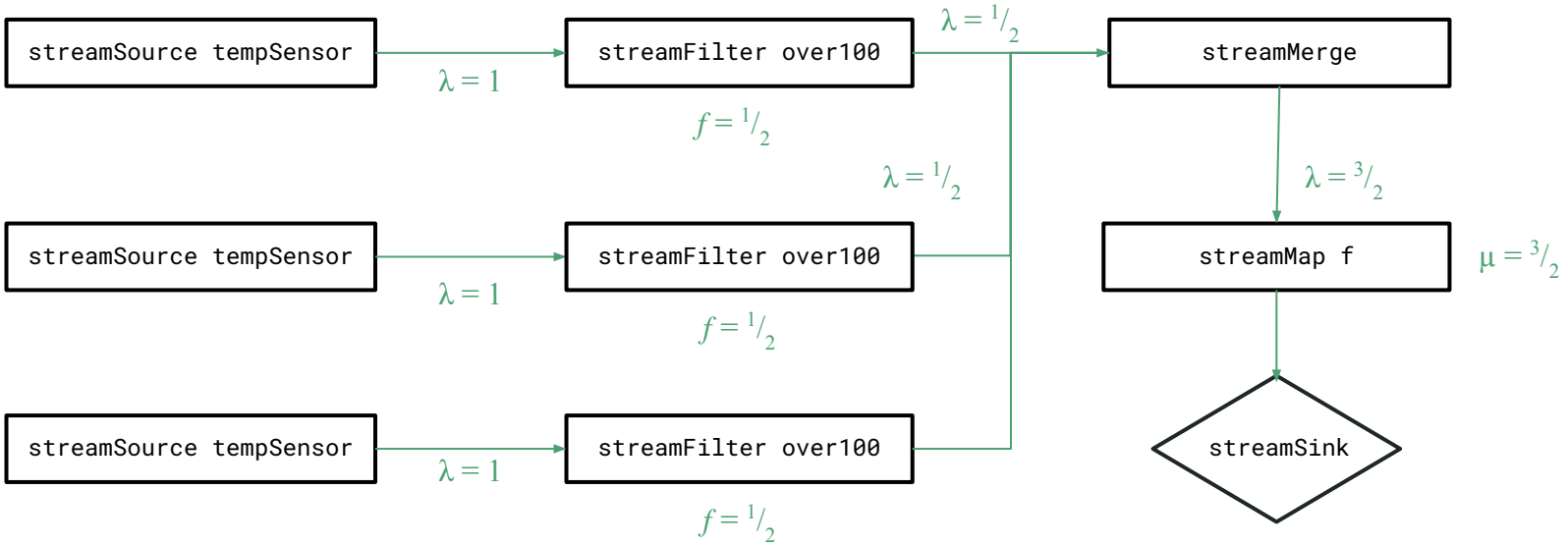
Example outcome #1 of 3

Reject over-utilised operators

Input program: over-utilised operator



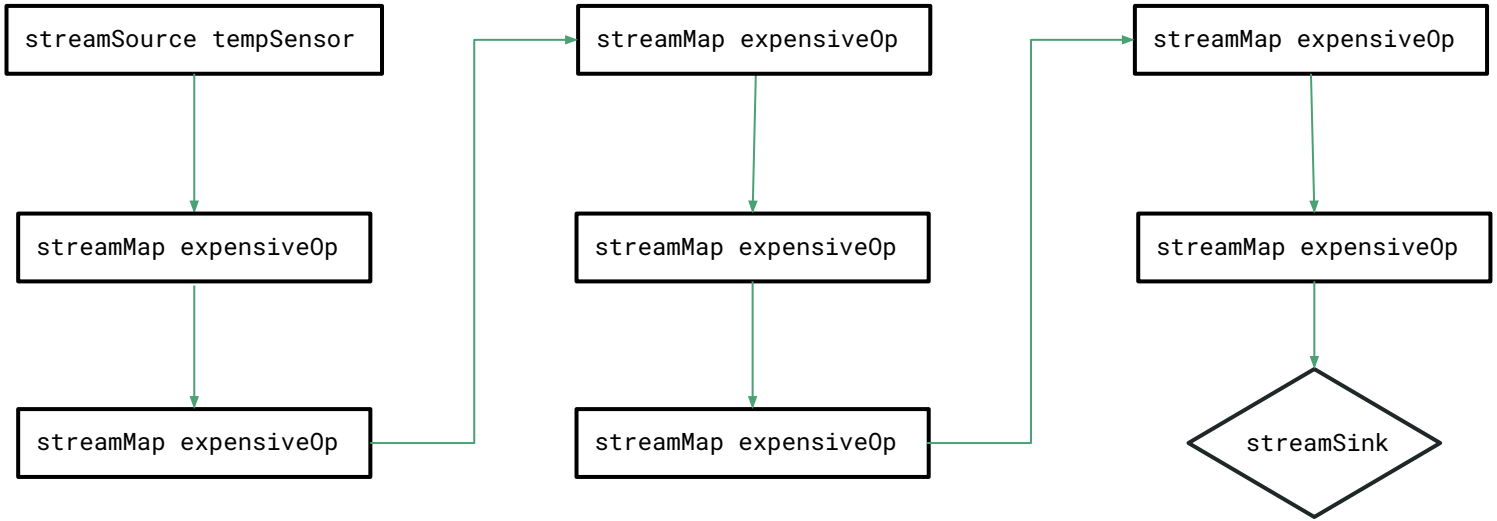
Re-written program: no over-utilised operators



Example outcome #2 of 3

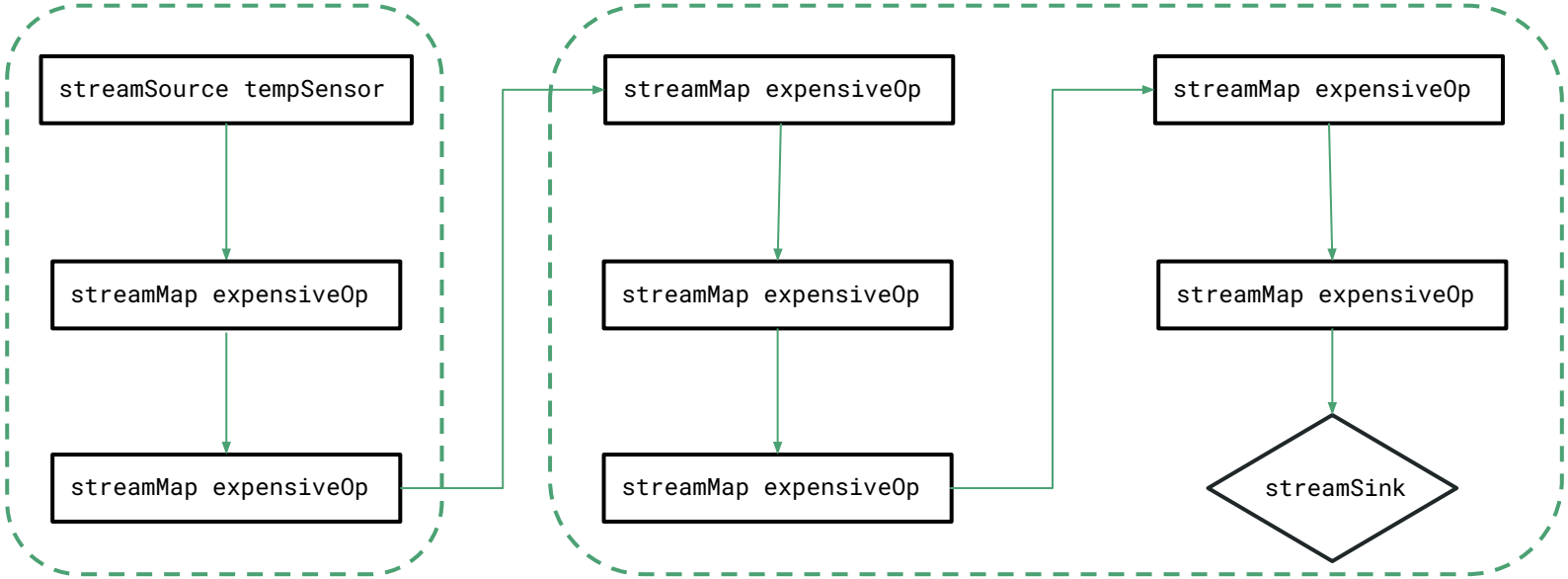
Discard plans with Nodes above a utilisation threshold

Input program



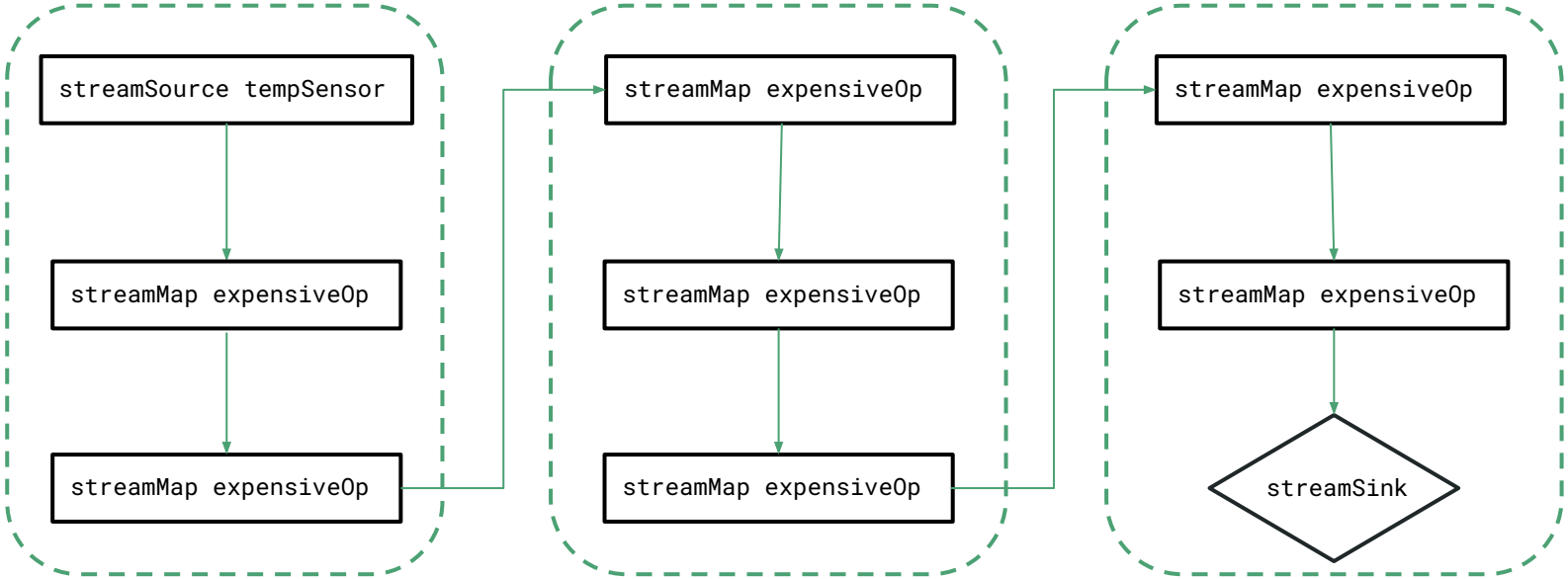
7 expensive operations (each $\rho = 1$)

Partition assignment (no max. Node Utilisation threshold)



2 Nodes

Partition assignment (max. Node Utilisation = 3)

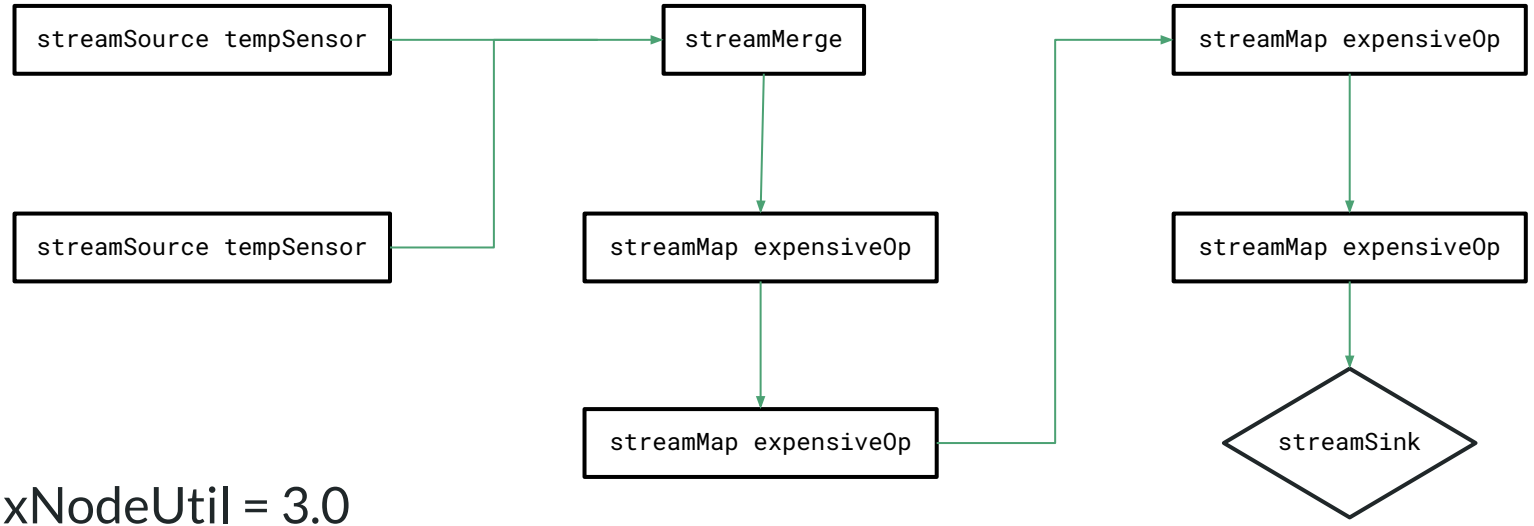


3 Nodes

Example outcome #3 of 3

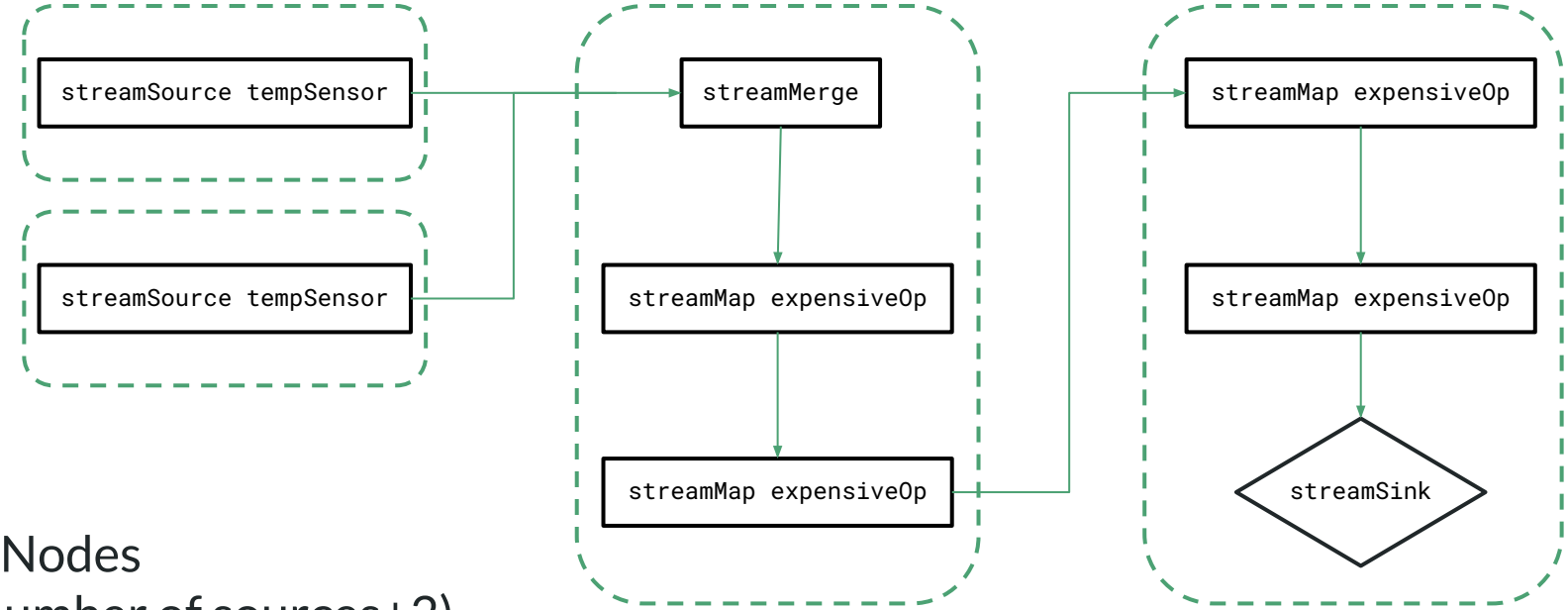
Reduce required Cloud nodes by increasing Edge utilisation

Input program



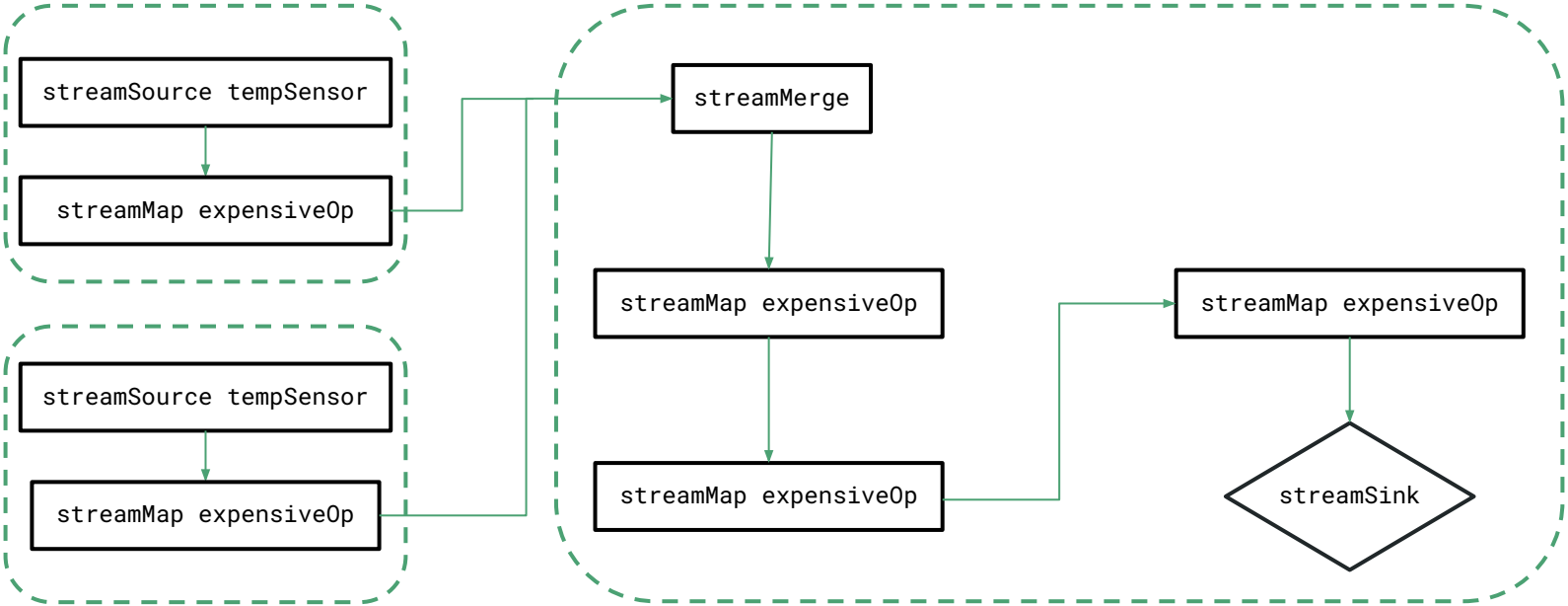
maxNodeUtil = 3.0

Partition assignment (input program)



4 Nodes
(number of sources+2)

Partition assignment (re-written program)



3 Nodes

Future work

- Heterogeneous nodes
  - (capabilities, limitations, costs...)
- Non-functional requirements
  - Bandwidth
- Further modelling work
- Operator semantics (streamWindow)
- quickSpec - machine-assisted law discovery

Thank you!
Q&A

Jonathan Dowland <jon.dowland@ncl.ac.uk>
UK Systems ‘21