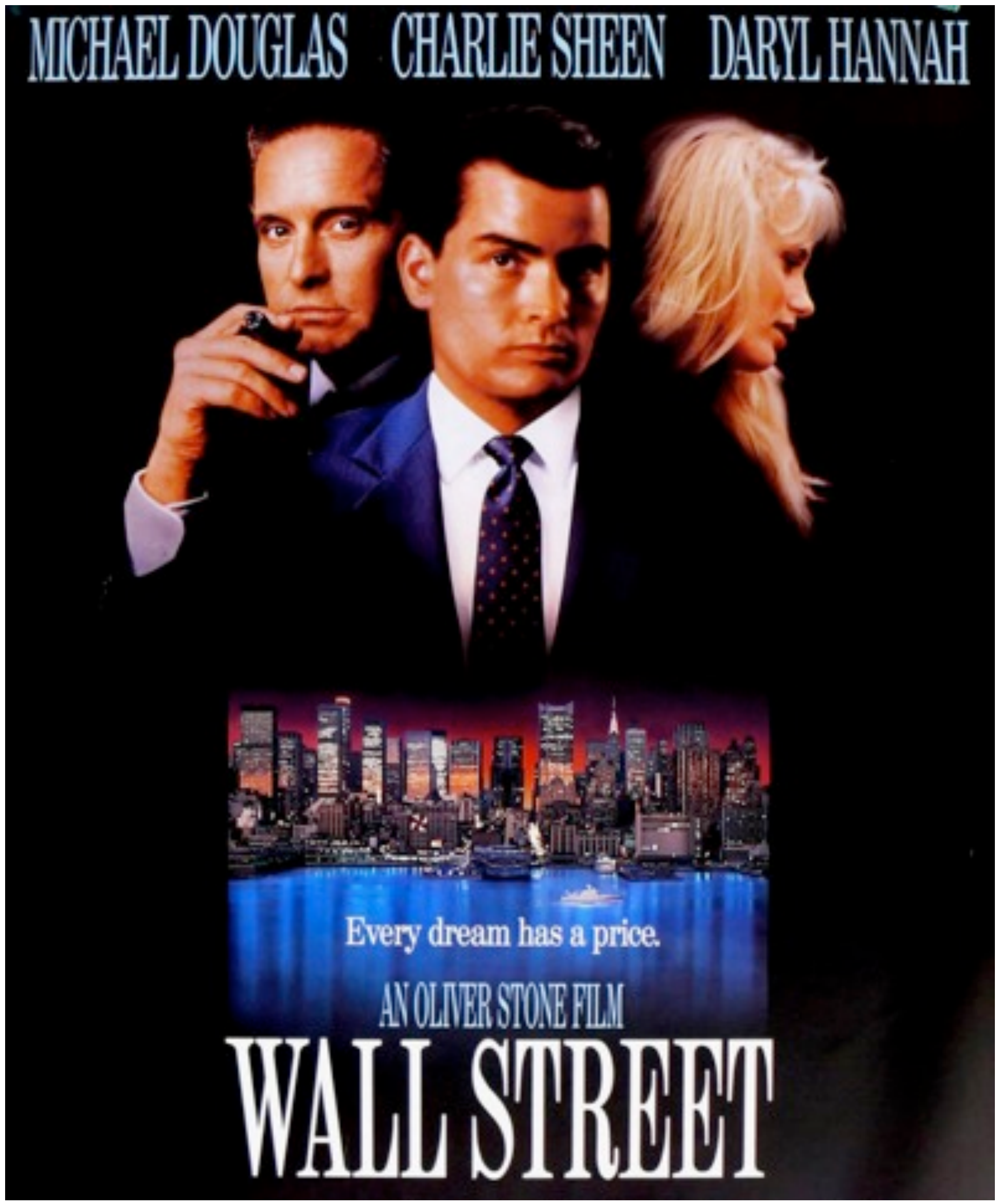
# A Universal Calculus for Stream Processing Languages

**Robert Soulé**, Martin Hirzel, Robert Grimm, Buğra Gedik, Henrique Andrade, Vibhore Kumar, and Kun-Lung Wu

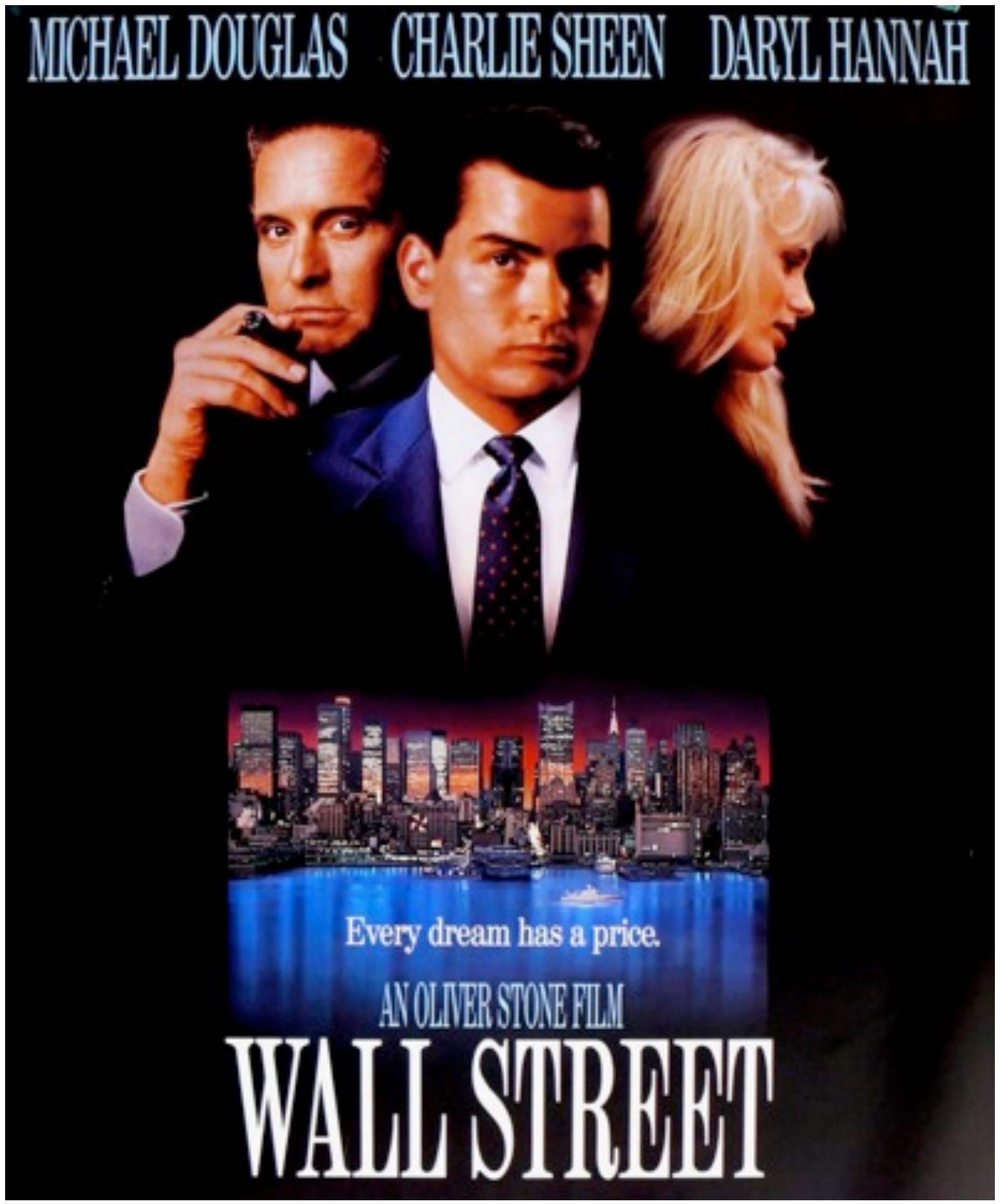
New York University and IBM Research

# Stream Processing Is Everywhere

# Stream Processing Is Everywhere

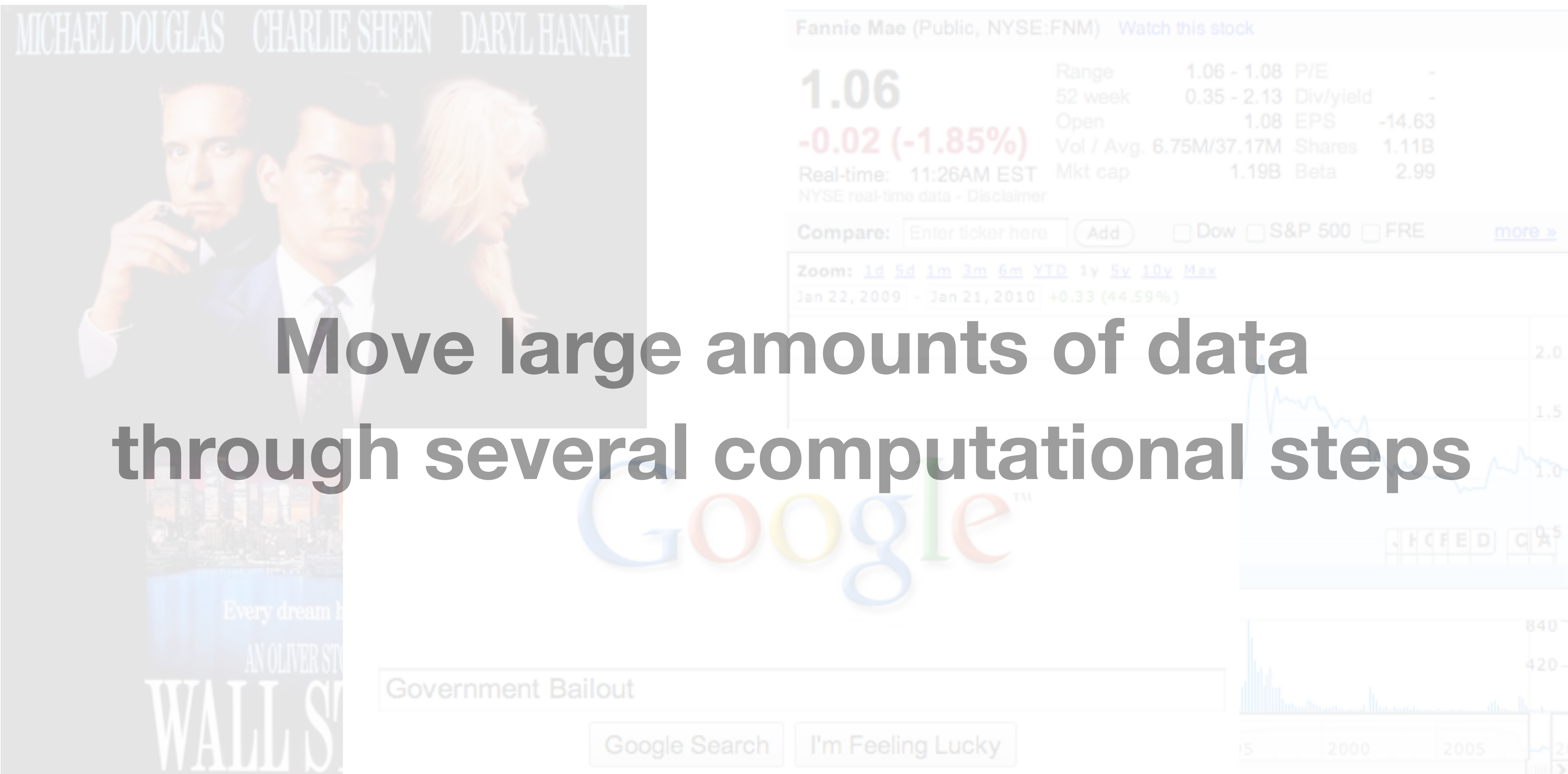


# Stream Processing Is Everywhere



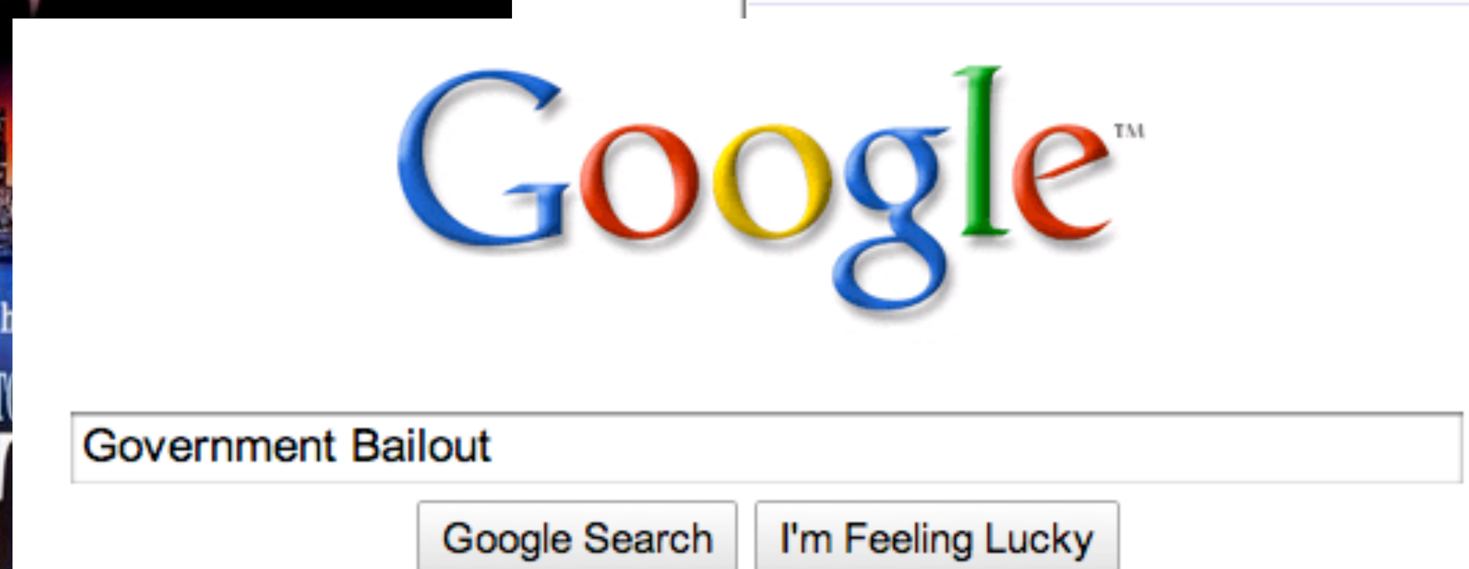

# Stream Processing Is Everywhere

# Stream Processing Is Everywhere

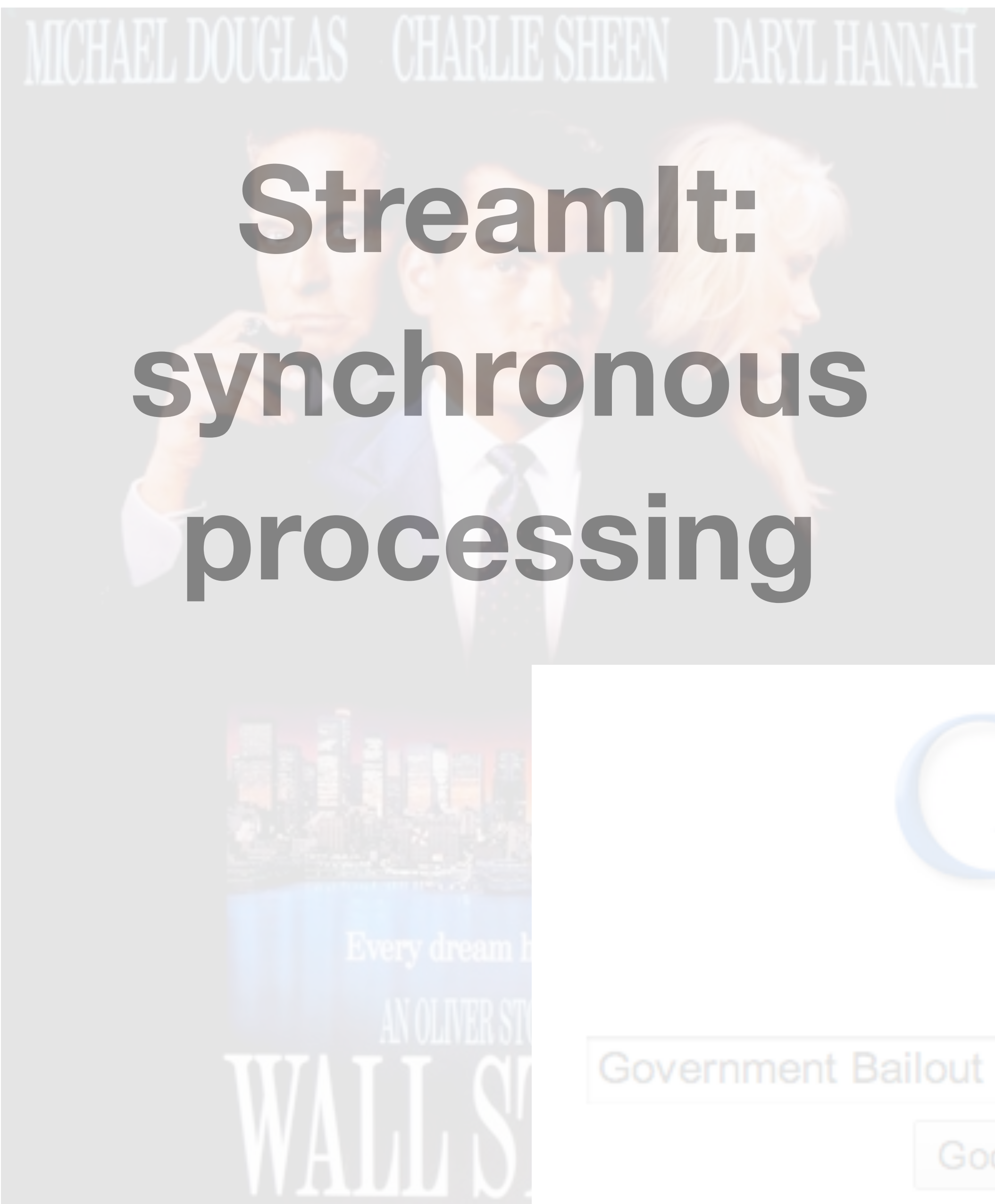



**Move large amounts of data through several computational steps**

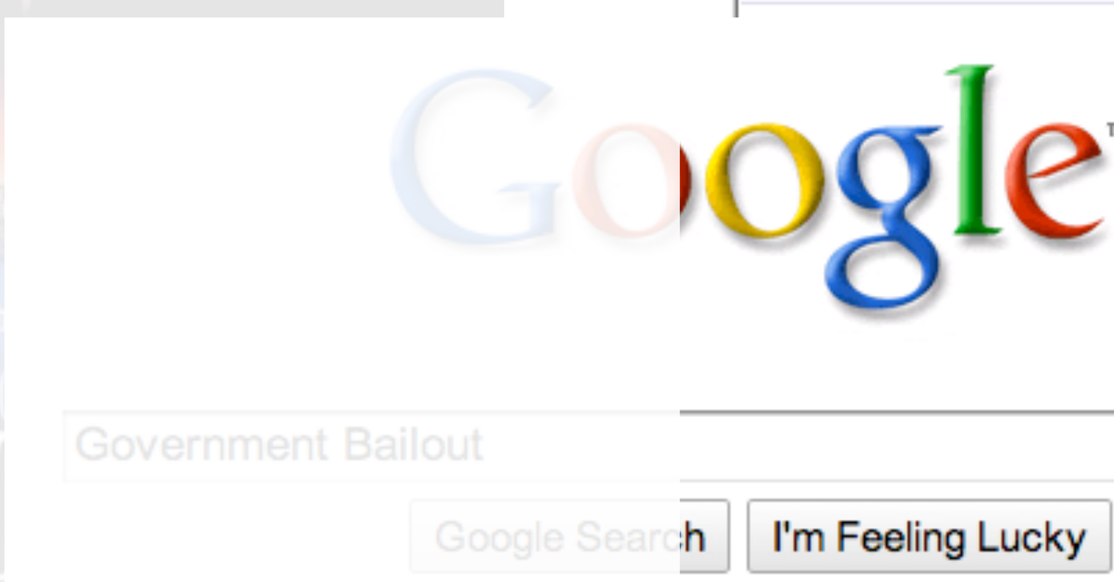
# Stream Processing Has Many Flavors

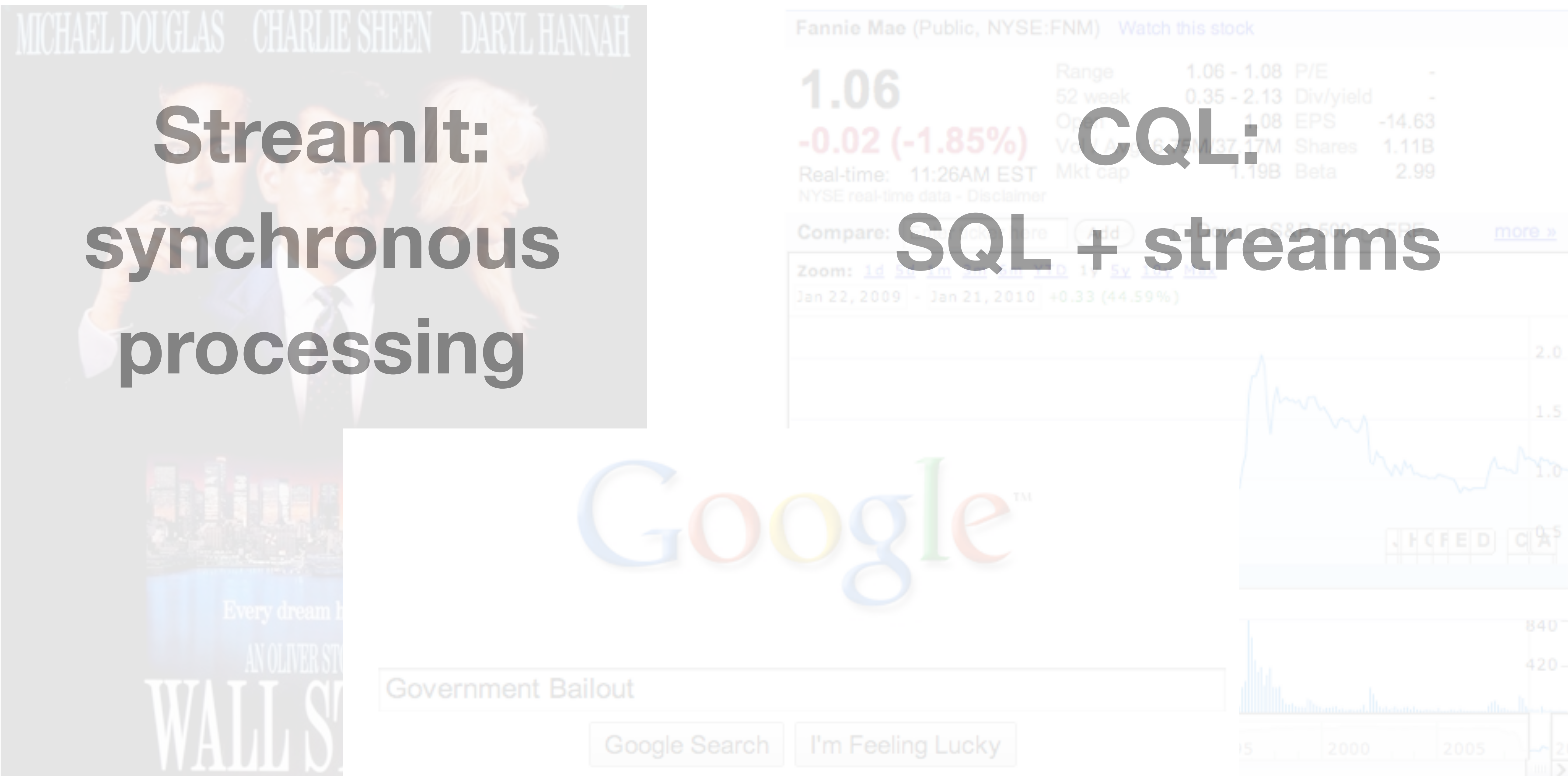

# Stream Processing Has Many Flavors




StreamIt: synchronous processing

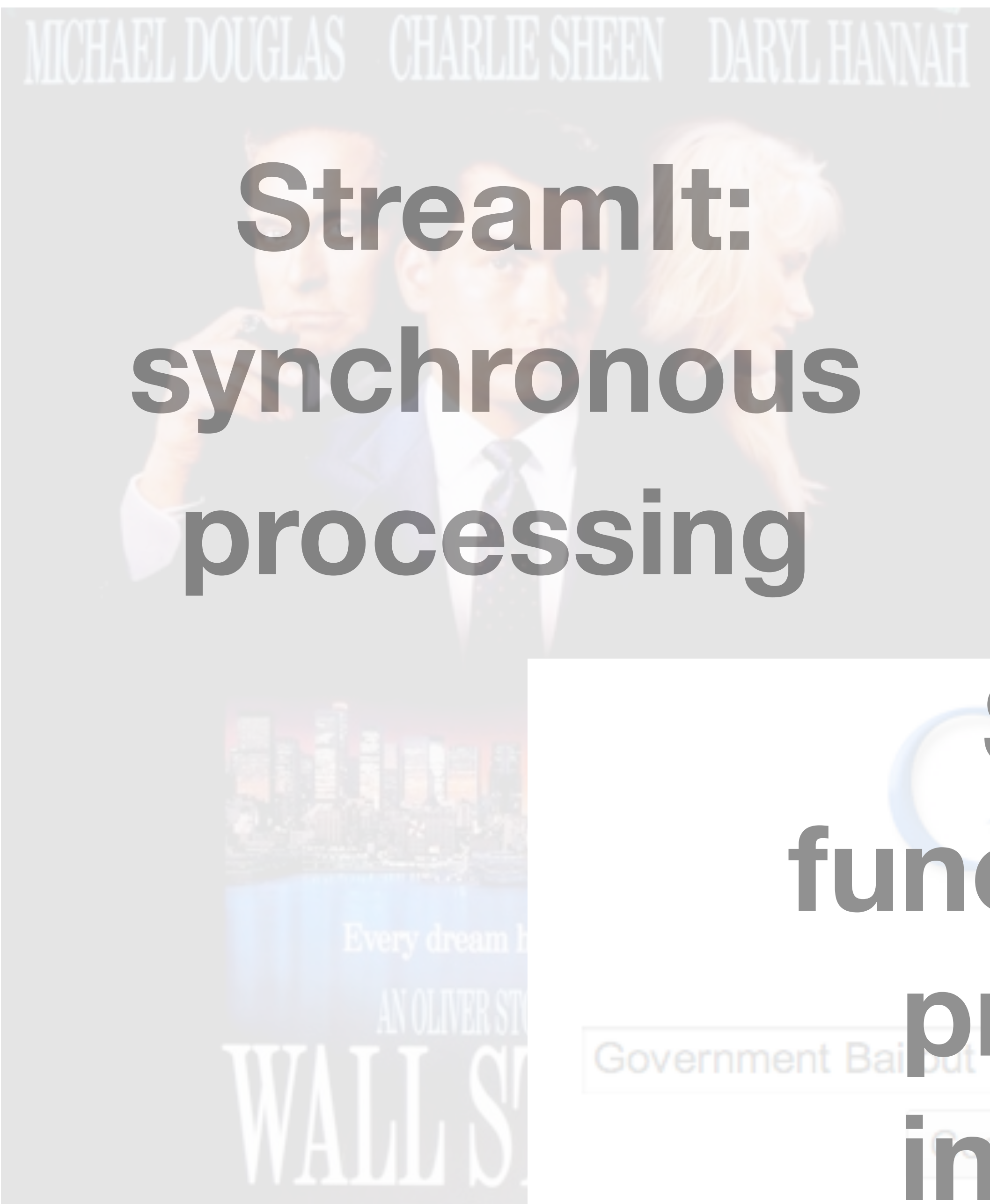
# Stream Processing Has Many Flavors

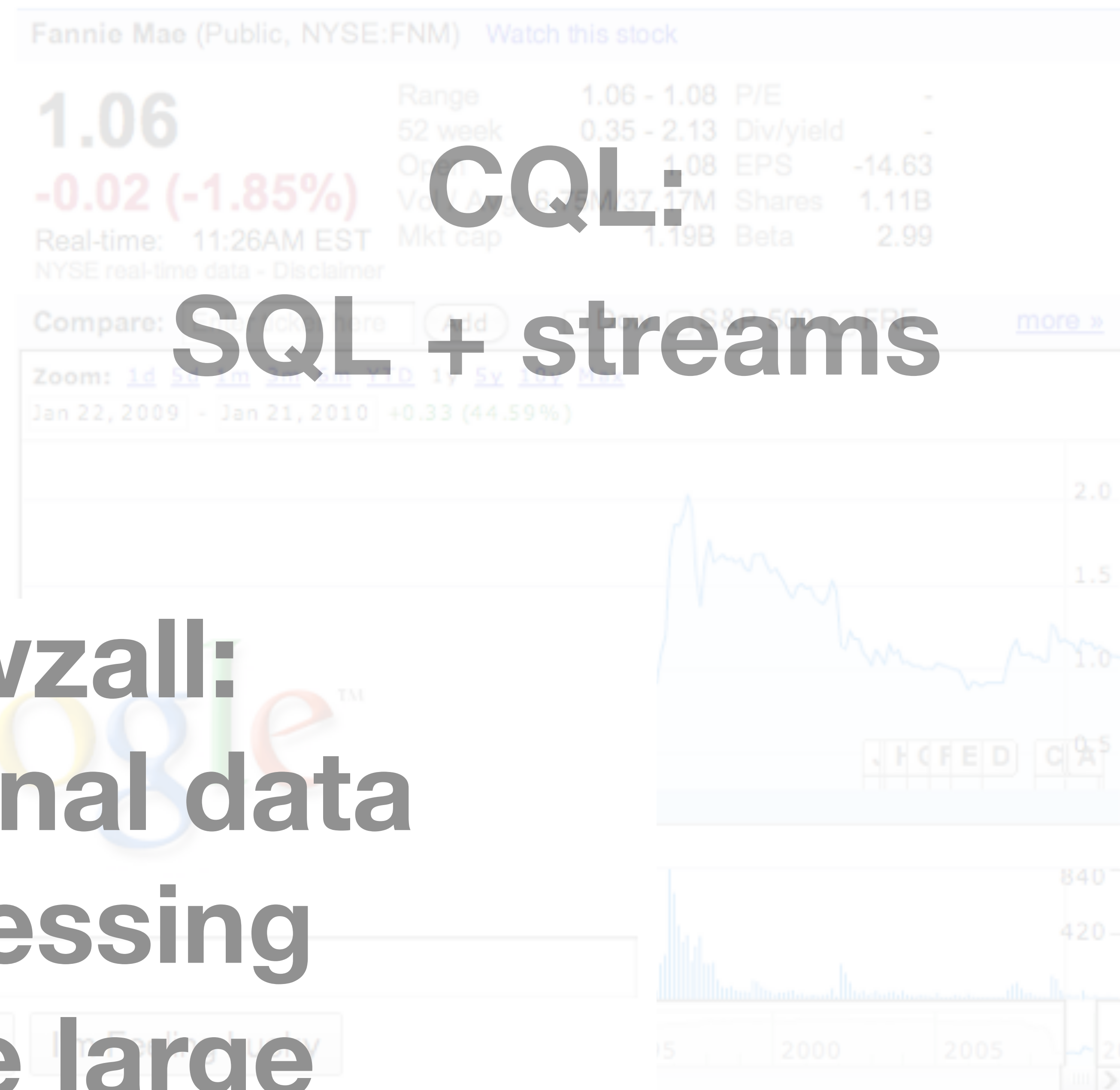



**StreamIt: synchronous processing**

**CQL: SQL + streams**

# Stream Processing Has Many Flavors



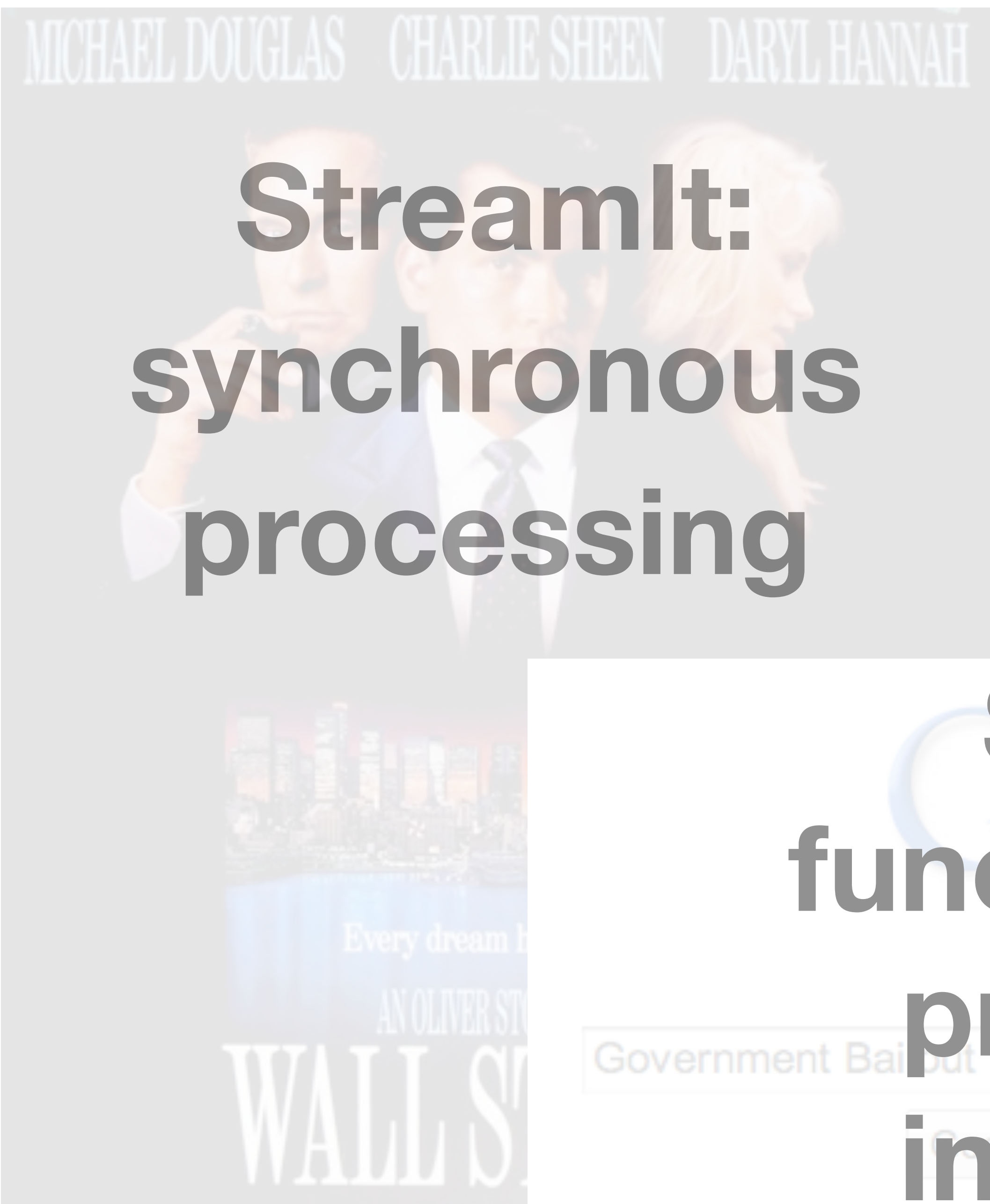




StreamIt: synchronous processing

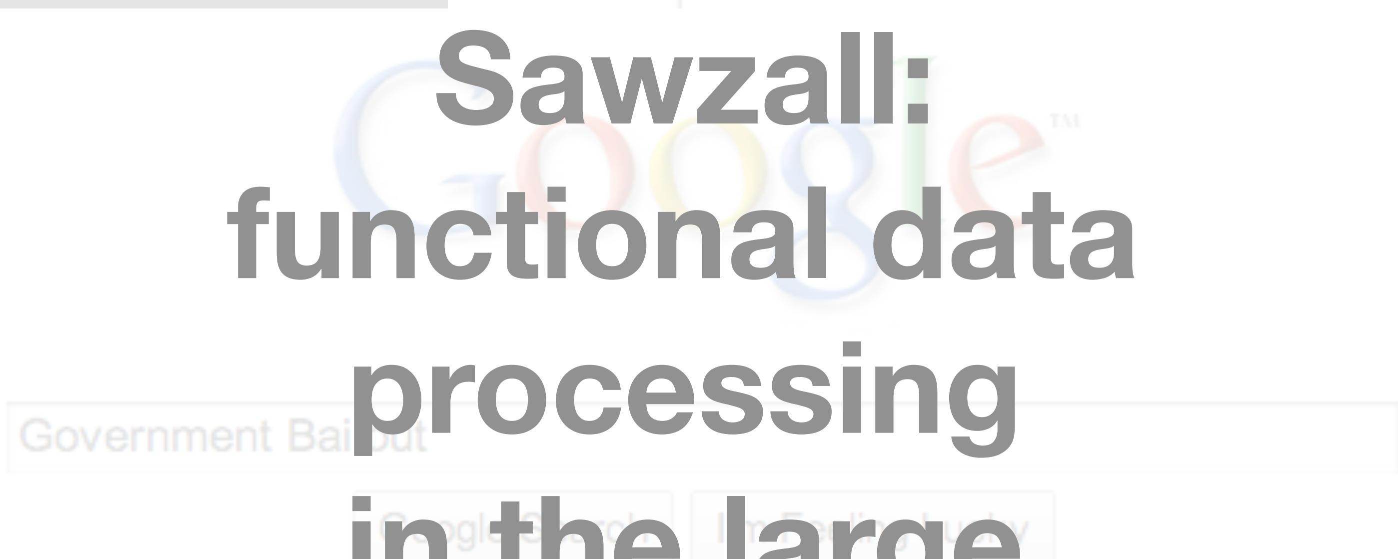
CQL: SQL + streams

Sawzall: functional data processing in the large

# Stream Processing Has Many Implementations








**StreamIt: synchronous processing**

**CQL: SQL + streams**

**Sawzall: functional data processing in the large**

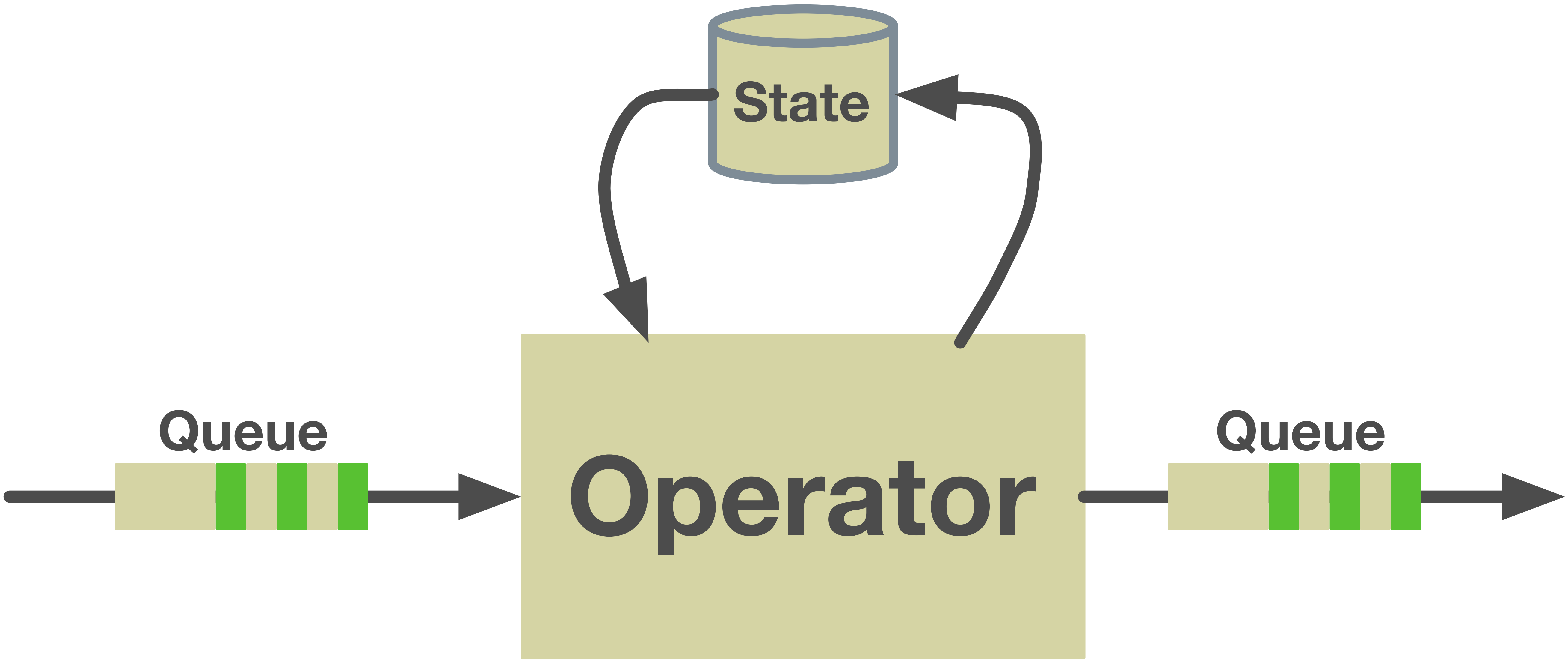
# Variety Breeds Confusion

- We want to understand and compare streaming languages
  - What is their expressiveness?
  - How to optimize the data processing steps?
  - How to scale the overall applications? Especially across clusters?
- Enter our universal calculus: Brooklet
  - Formal foundation for answering the above questions
  - Provably correct optimizations and translations

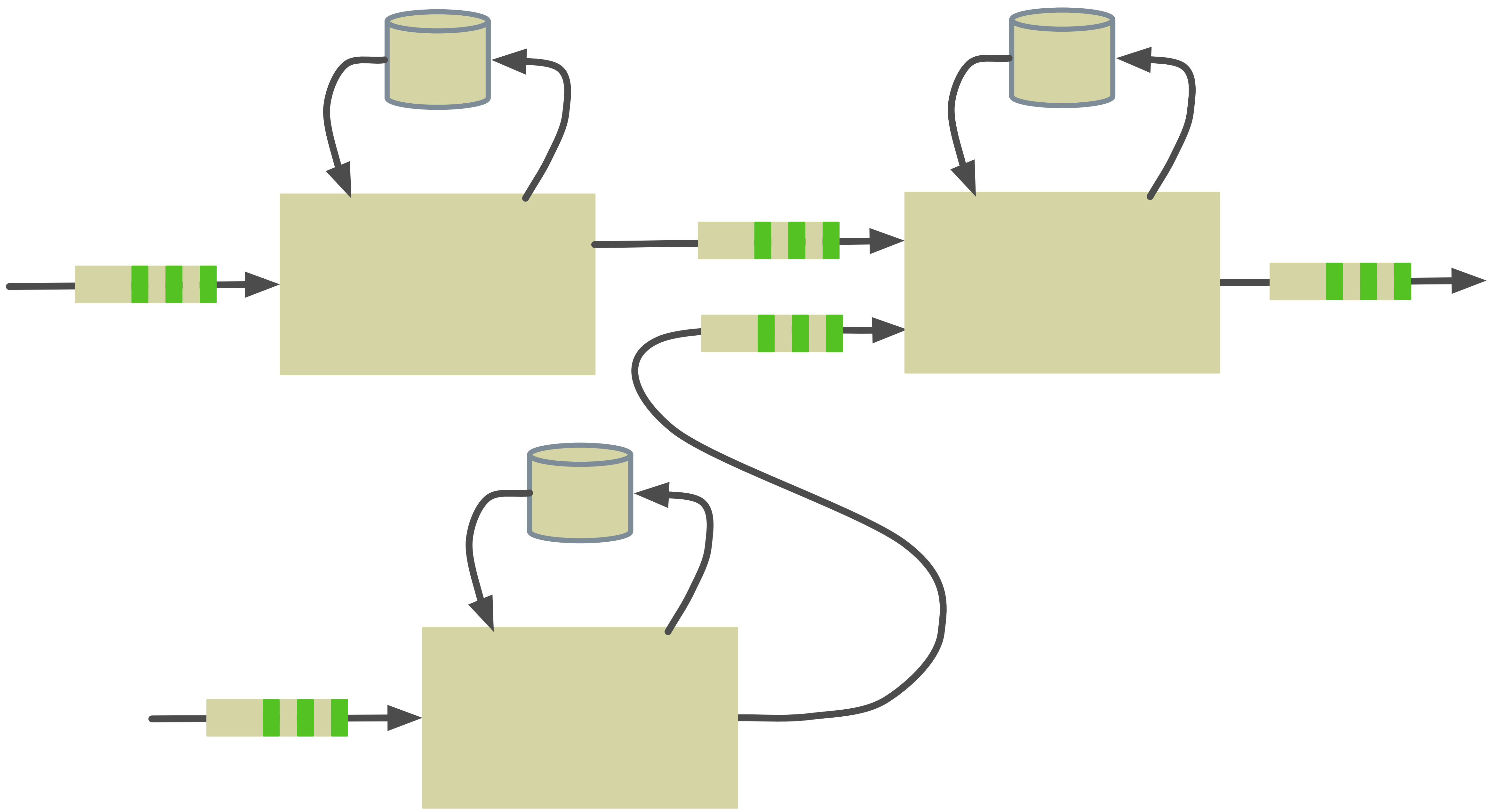
# Outline of This Talk

- Motivation
- **Requirements**
- The Brooklet Core Calculus
- Generality: Translating StreamIt, CQL, and Sawzall to Brooklet
- Utility: Optimizing Brooklet to Brooklet
- Outlook and Conclusions

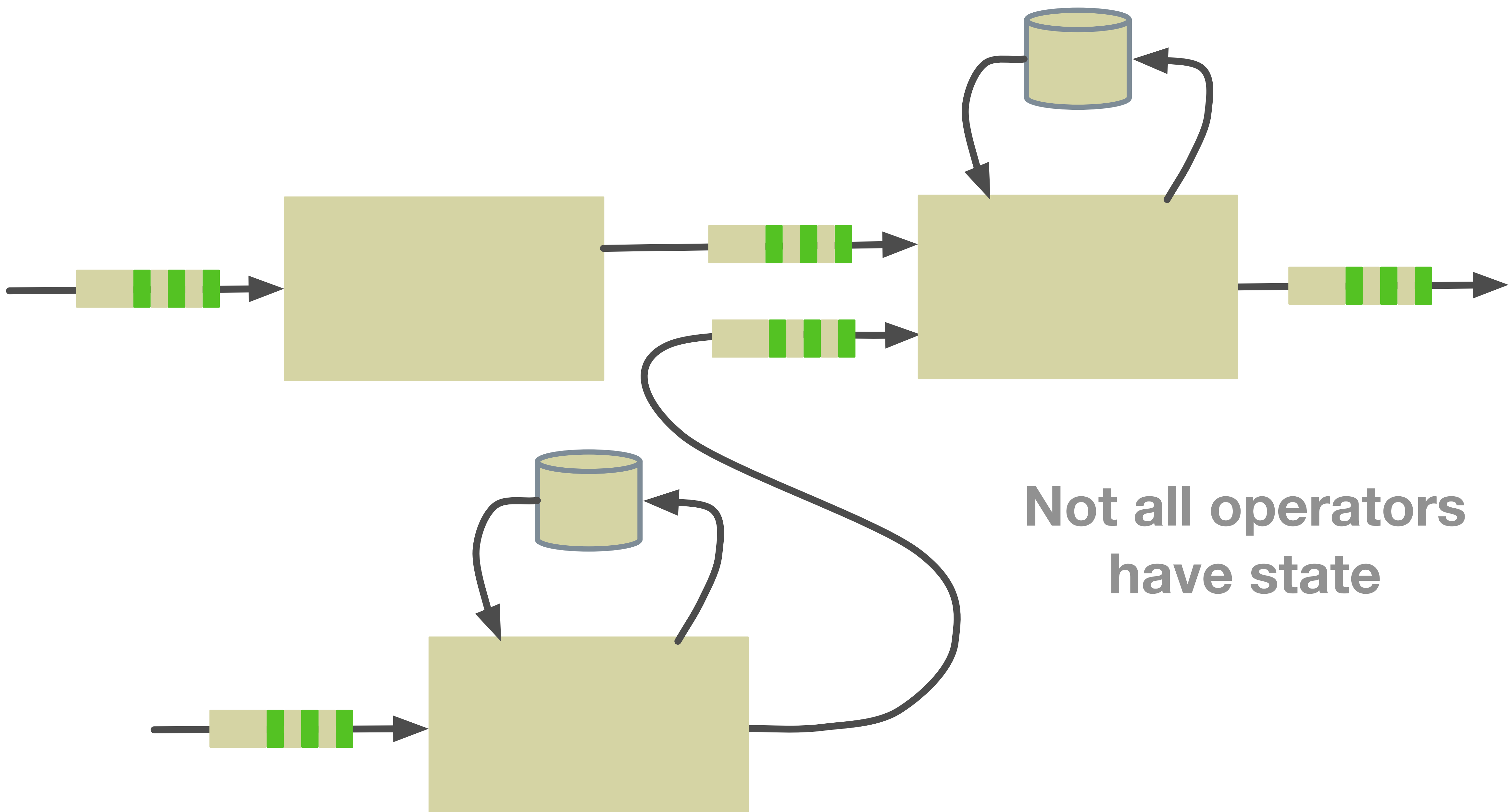
# Elements of a Streaming App



# Elements of a Streaming App

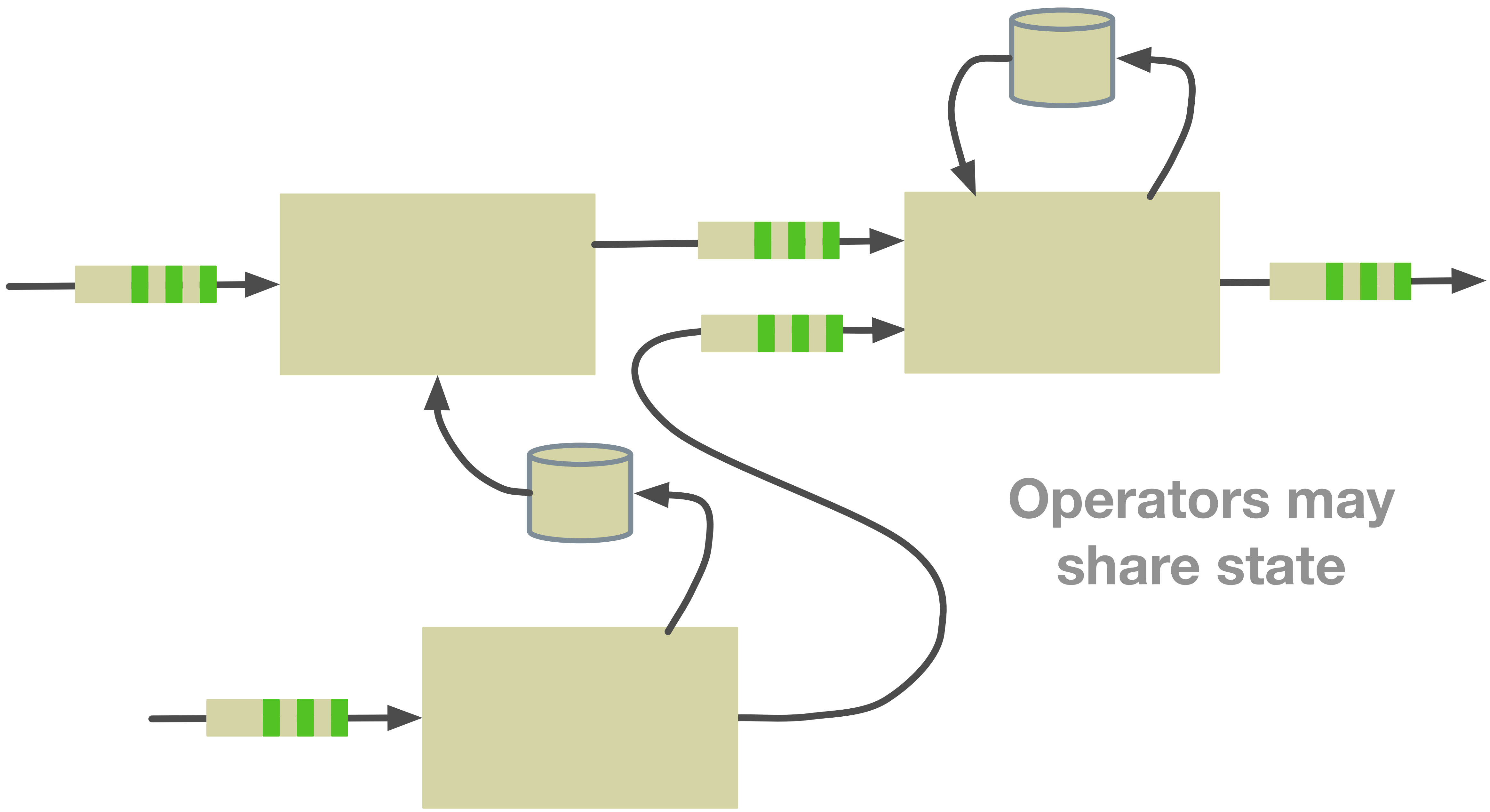


# Elements of a Streaming App



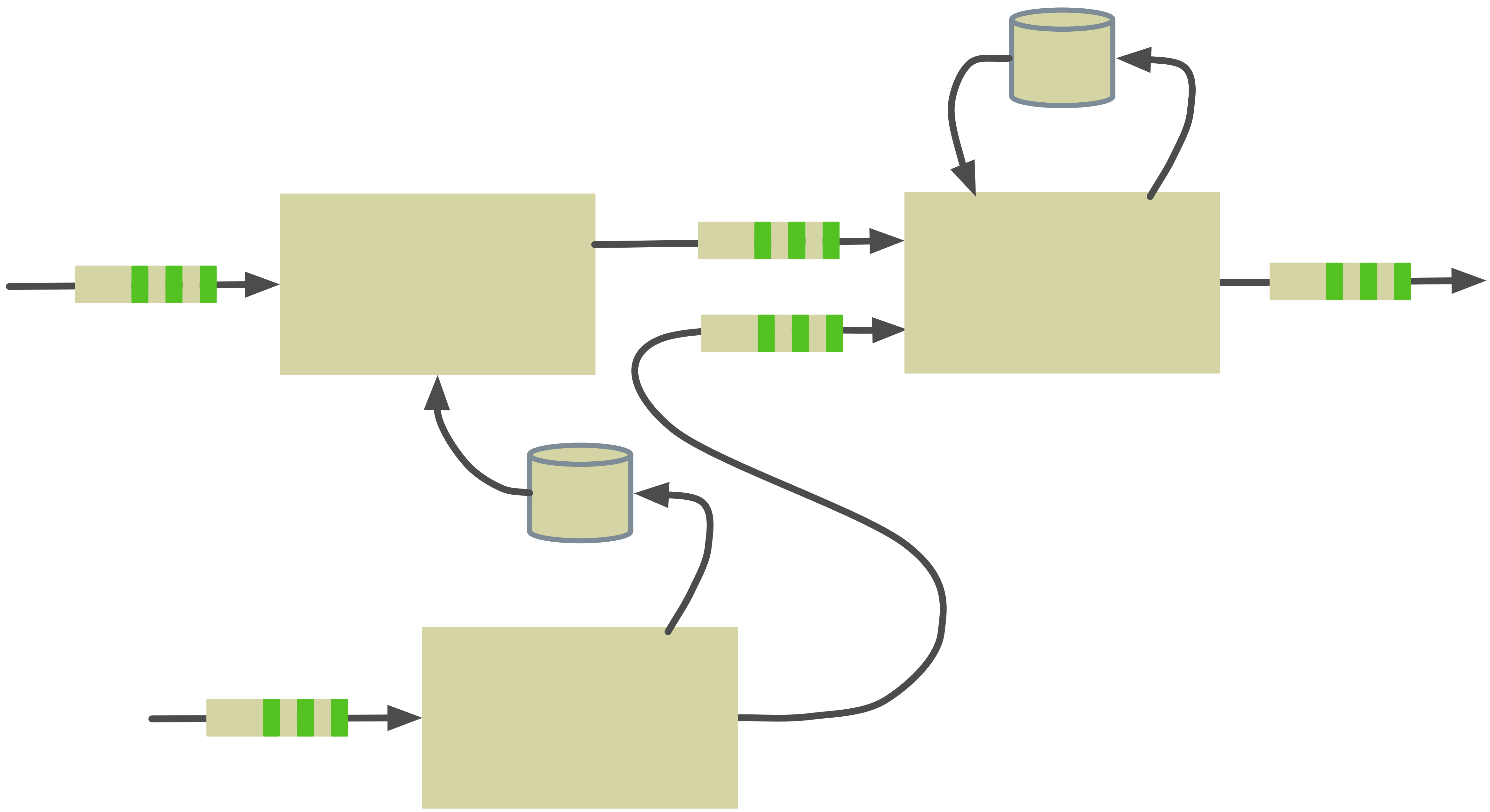
Not all operators have state

# Elements of a Streaming App

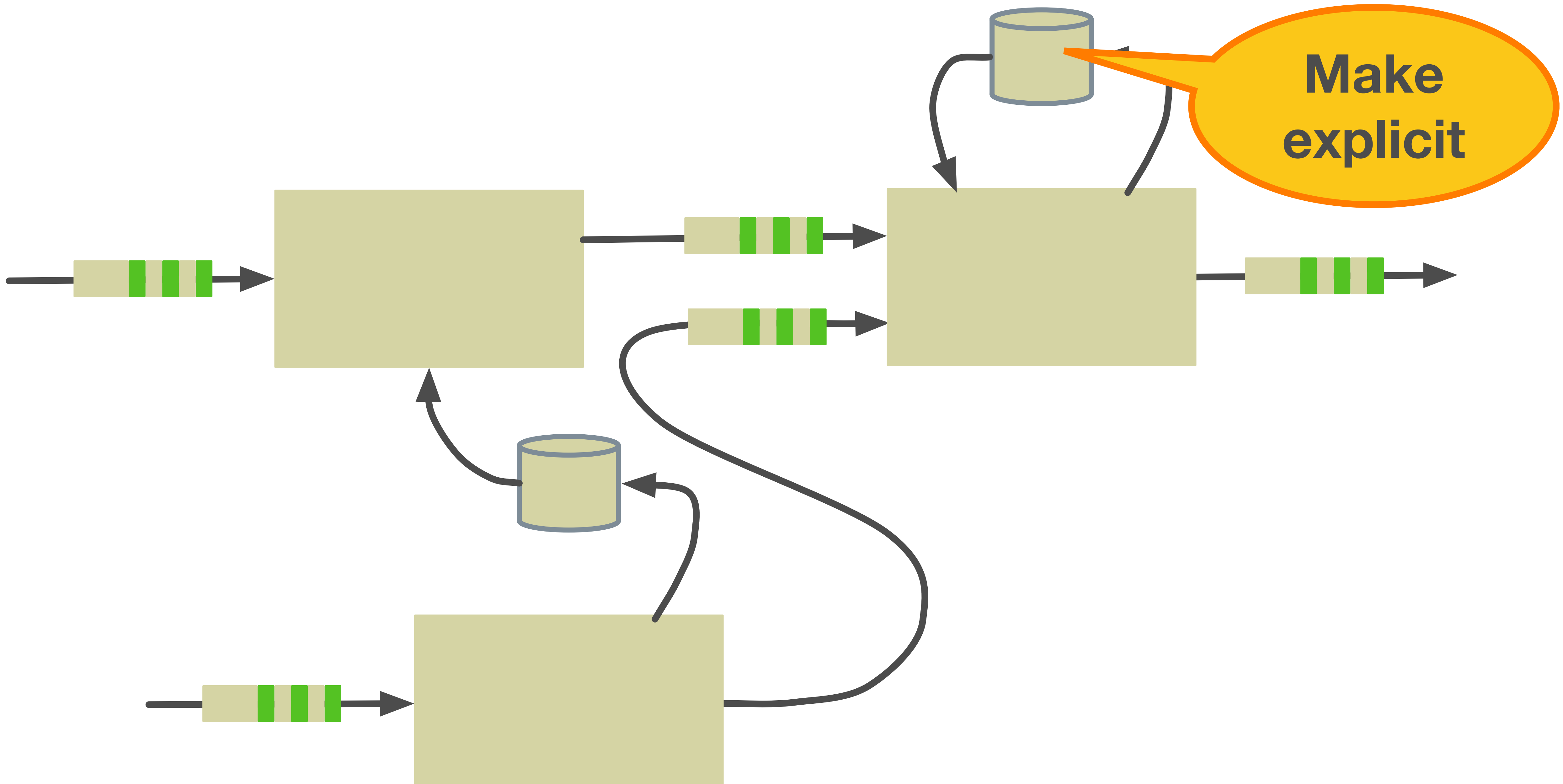


Operators may share state

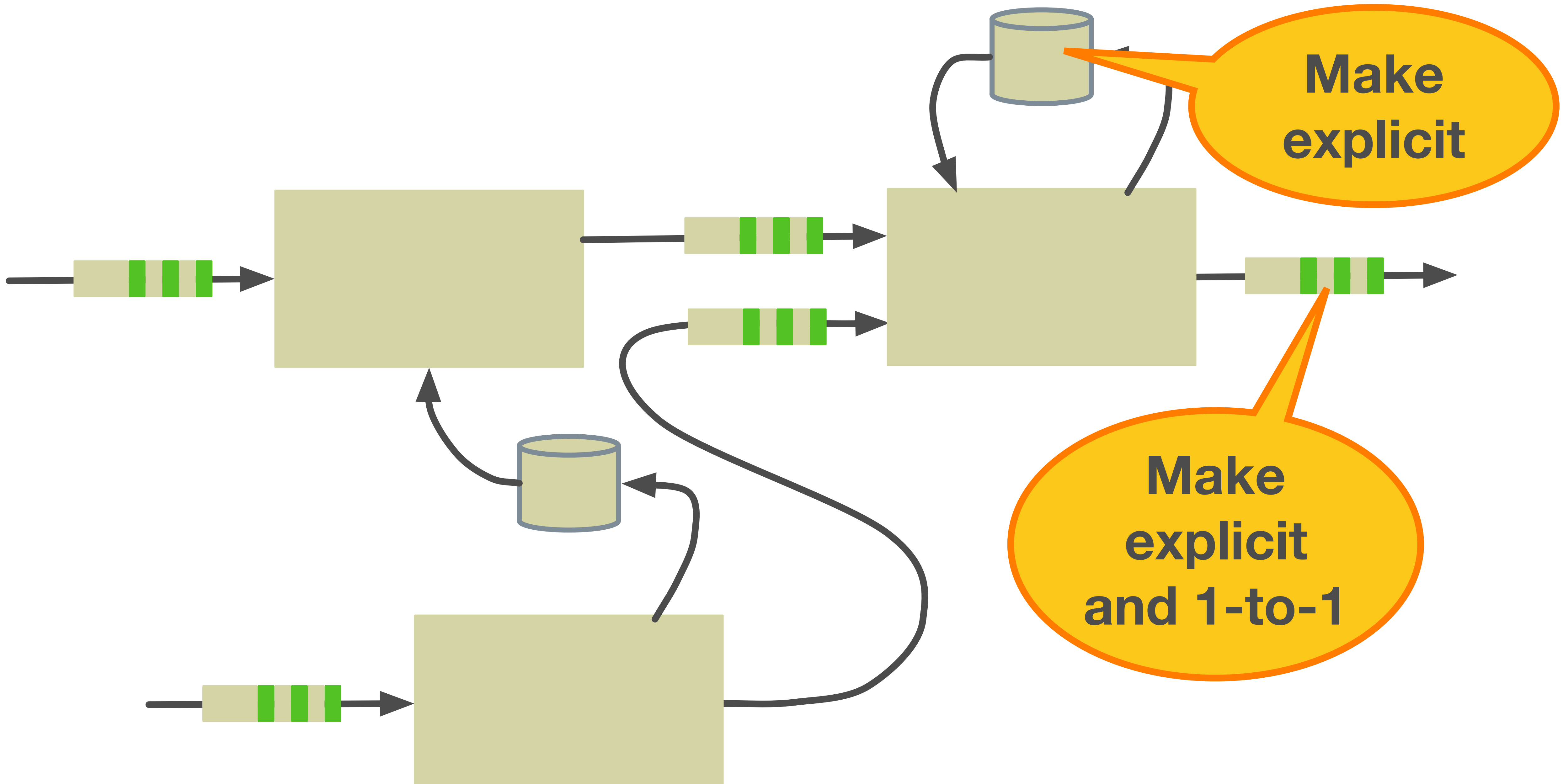
# Requirements for Calculus



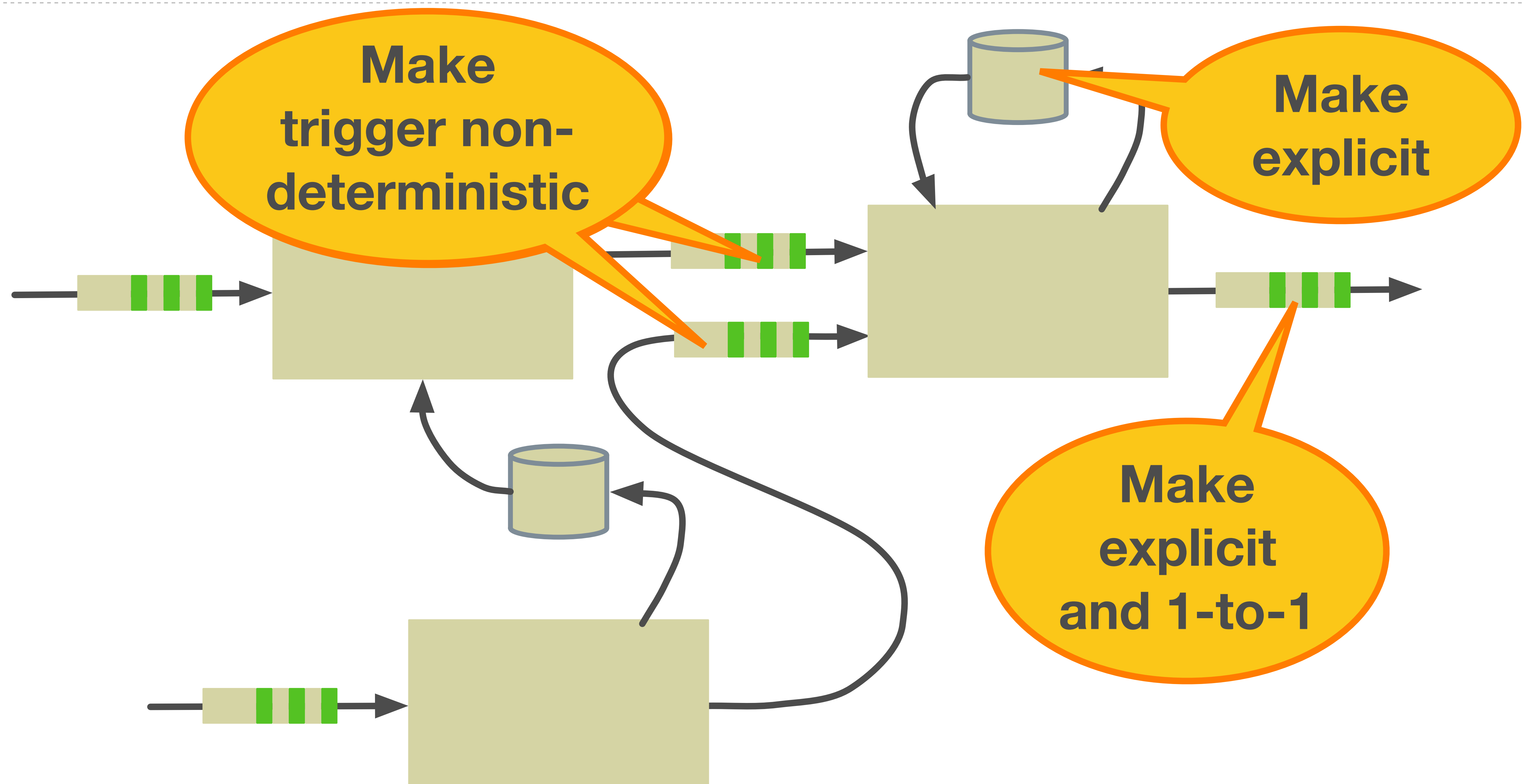
# Requirements for Calculus



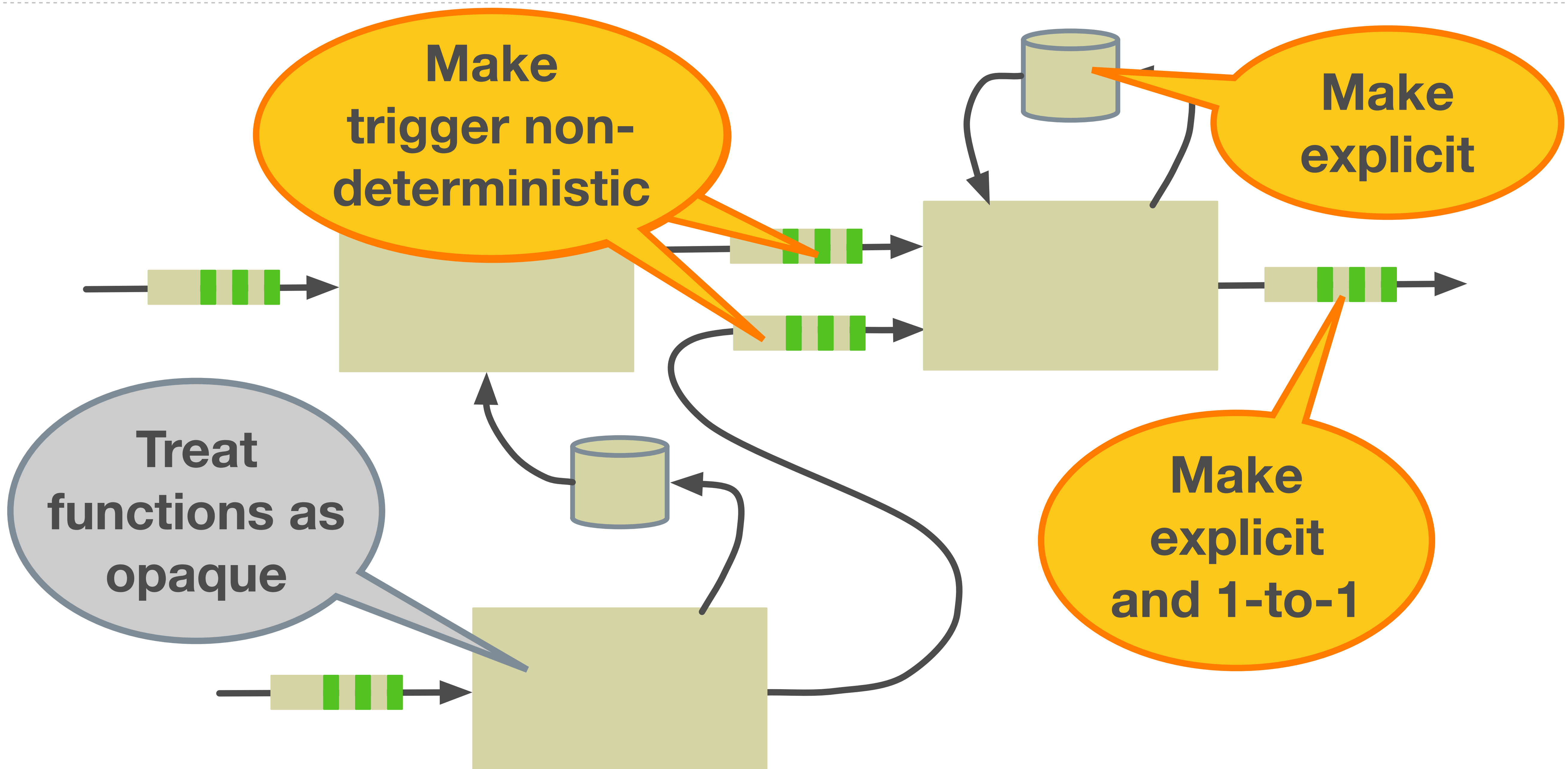
# Requirements for Calculus



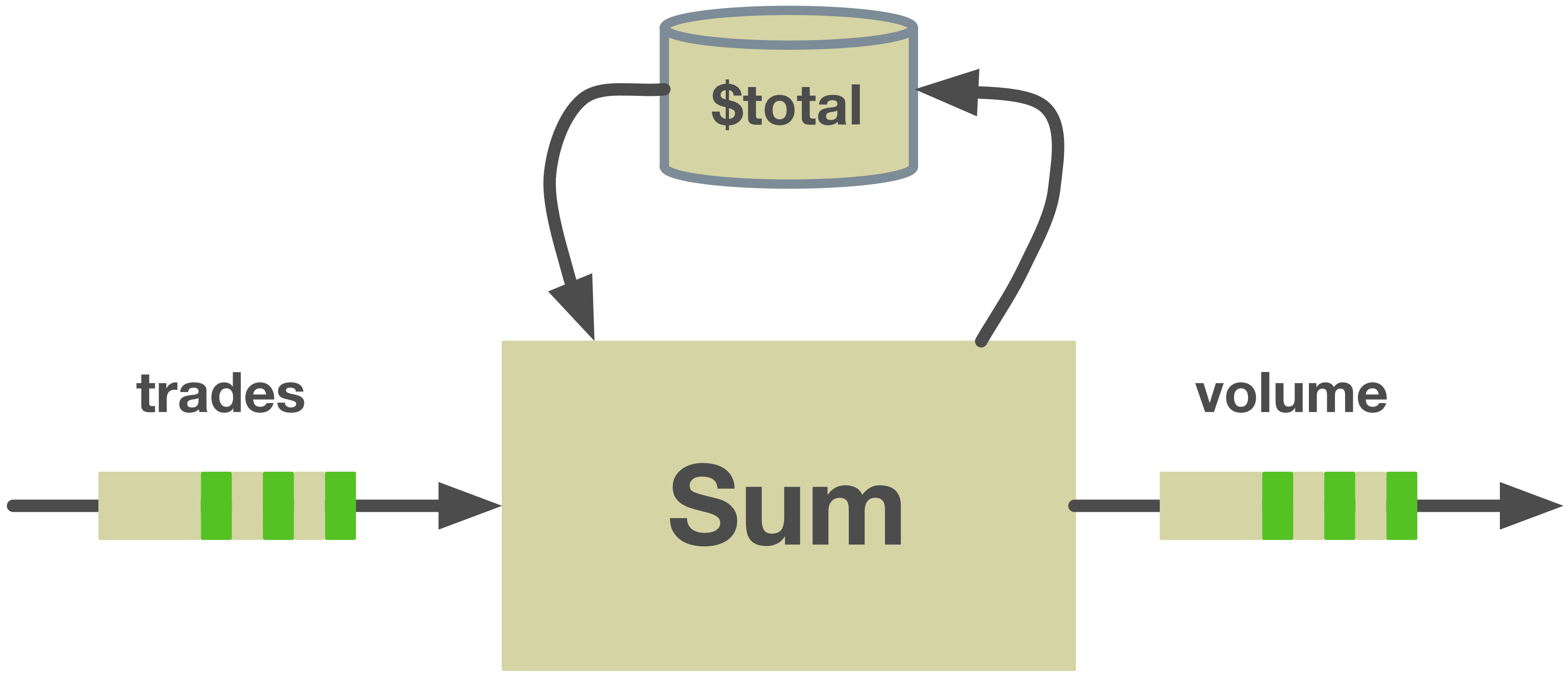
# Requirements for Calculus



# Requirements for Calculus

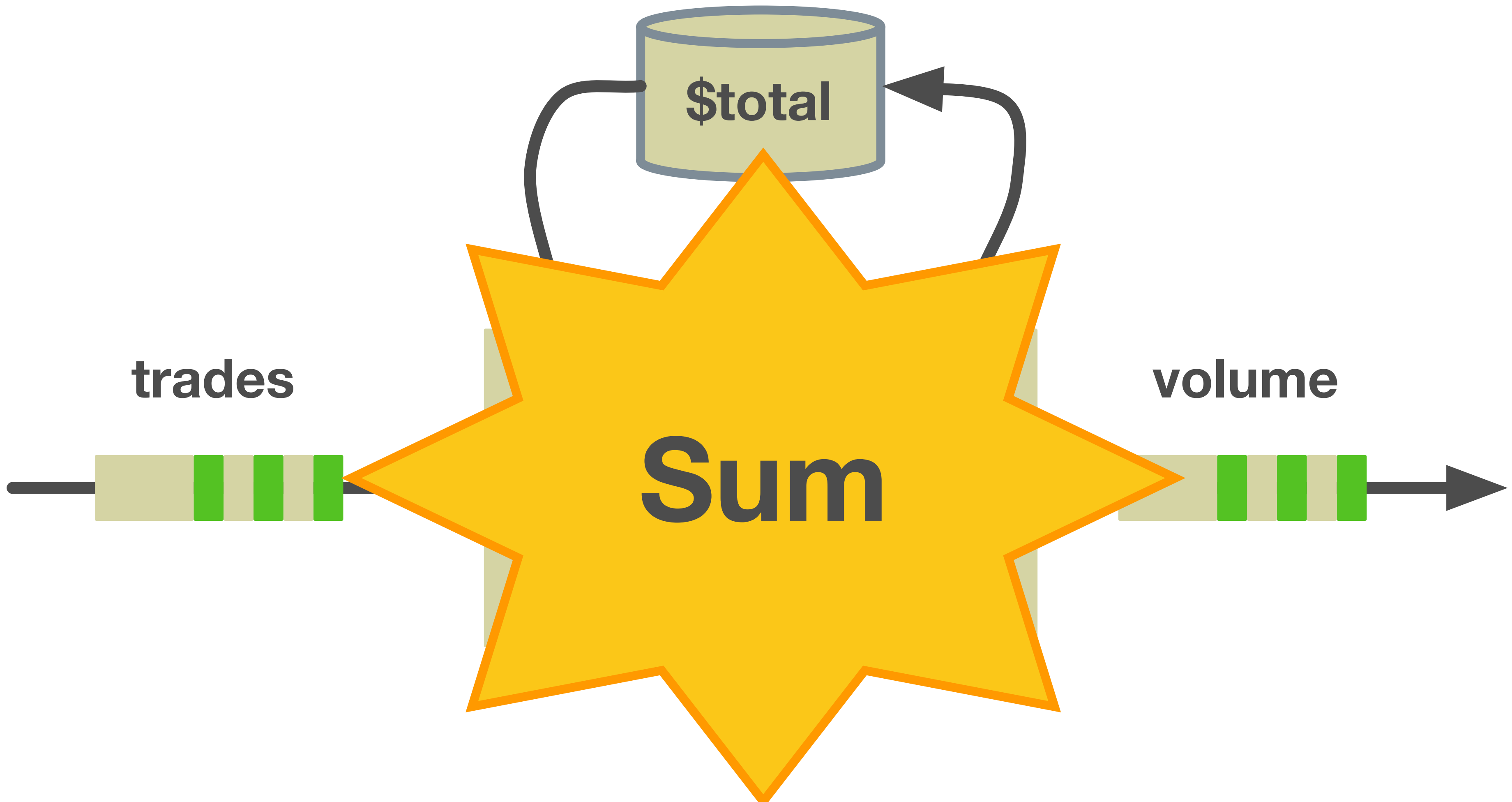


# Brooklet Syntax



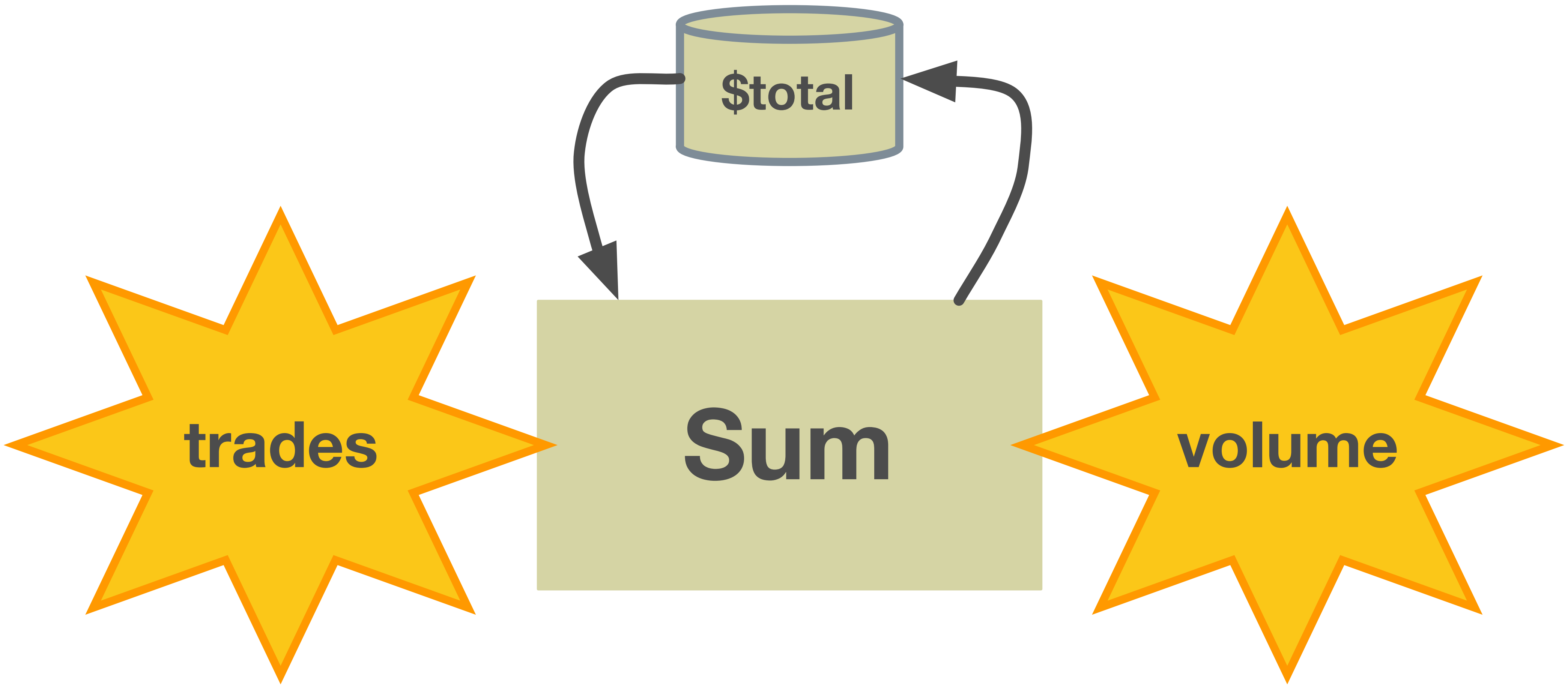

(volume, $total) ← Sum(trades, $total)

# Function Environment



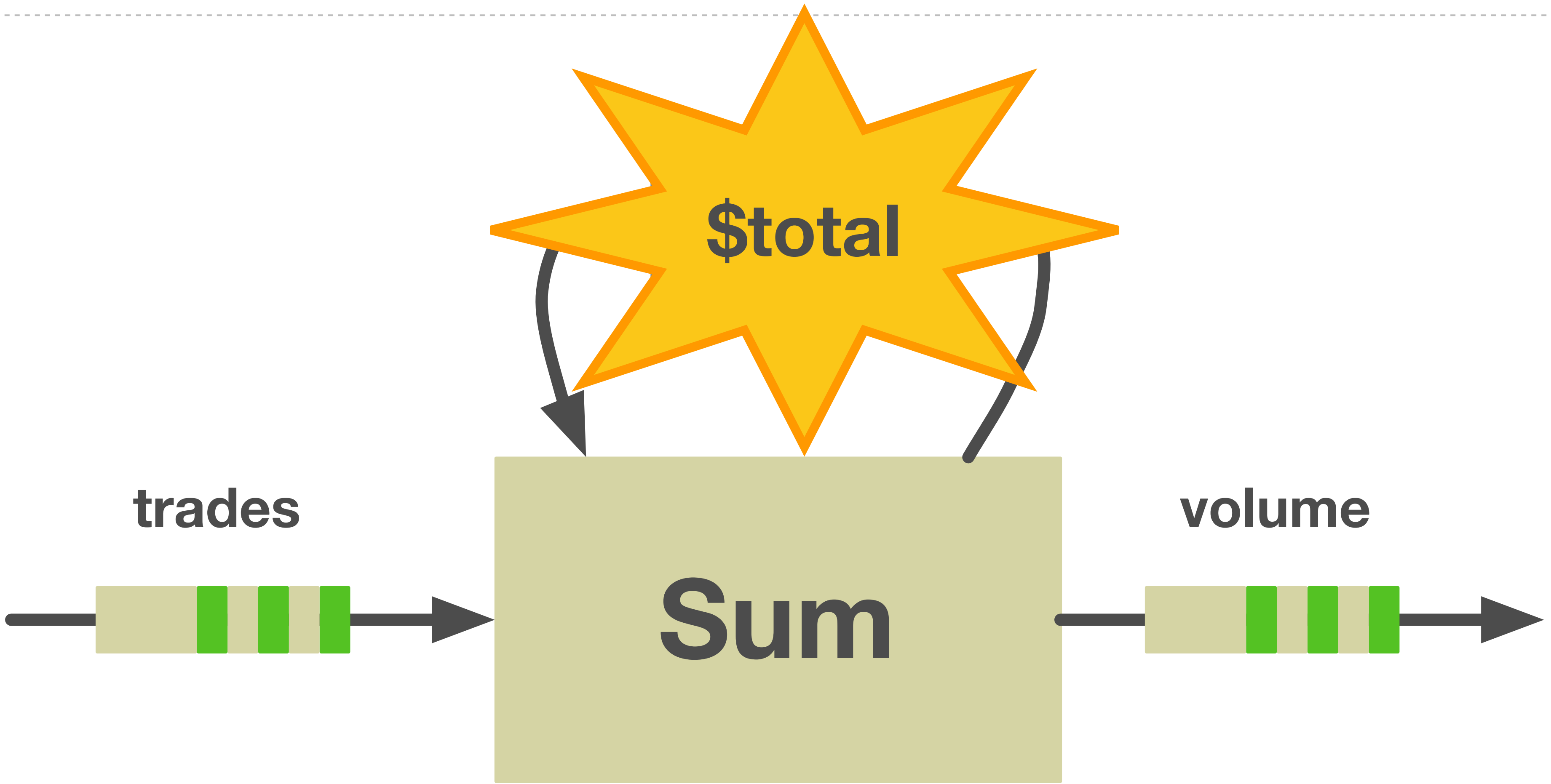

F: The function implementations

# Queue Store



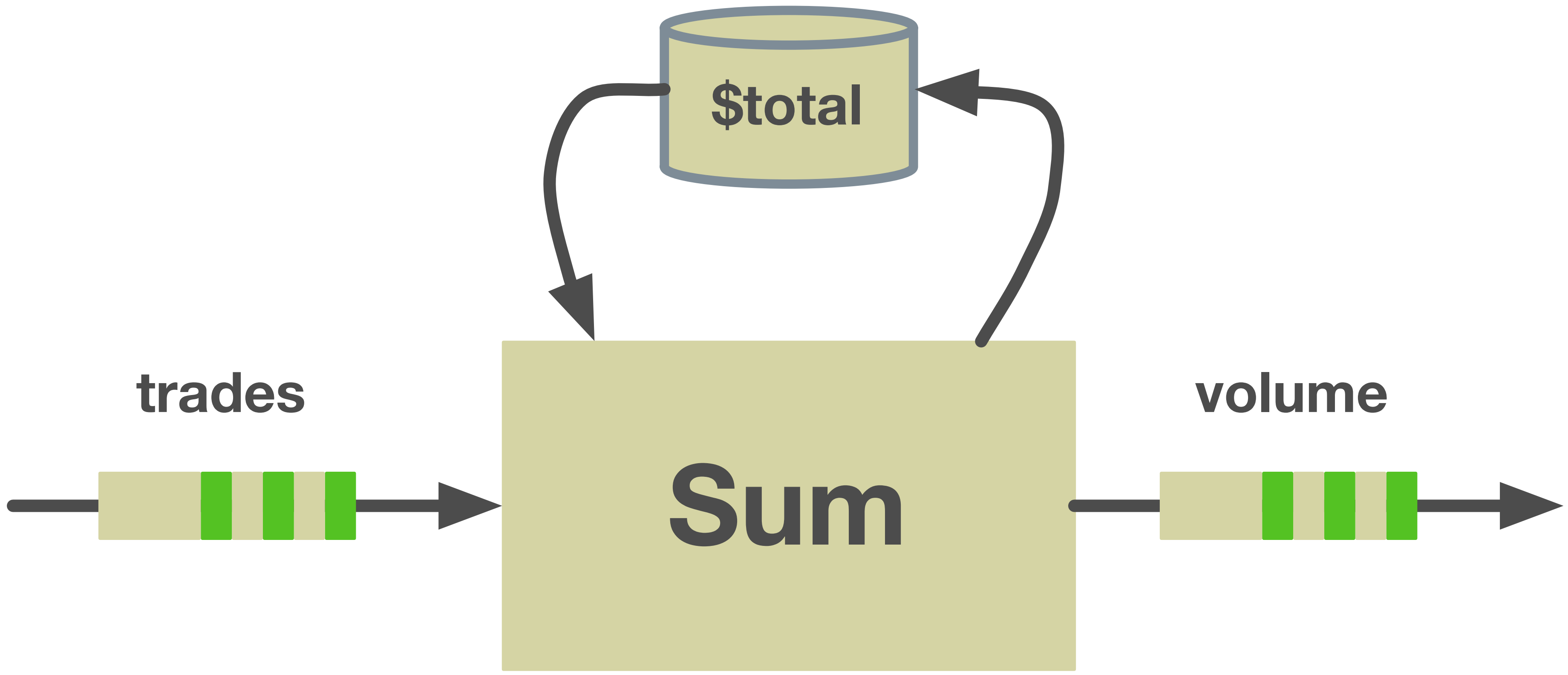

Q: The contents of the queues

# Variable Store




V: The contents of the variables

# Brooklet Operational Semantics




$$F \vdash \langle Q, V\rangle \rightarrow \langle Q', V'\rangle$$

# Complete Calculus

**Brooklet syntax:**

$$\begin{array}{llr}
P_b & ::= \; out \;\; in \;\; \overline{op} & \textit{Brooklet program} \\
out & ::= \; \texttt{output} \; \overline{q} \; \texttt{;} & \textit{Output declaration} \\
in & ::= \; \texttt{input} \; \overline{q} \; \texttt{;} & \textit{Input declaration} \\
op & ::= \; \texttt{(} \; \overline{q}, \overline{v} \; \texttt{)} \leftarrow f \; \texttt{(} \; \overline{q}, \overline{v} \; \texttt{);} & \textit{Operator} \\
q & ::= \; id & \textit{Queue identifier} \\
v & ::= \; \texttt{\$} \; id & \textit{Variable identifier} \\
f & ::= \; id & \textit{Function identifier}
\end{array}$$

**Brooklet example:** IBM market maker.

```
output result;
input bids, asks;
(ibmBids) ⟵ SelectIBM(bids);
(ibmAsks) ⟵ SelectIBM(asks);
($lastAsk)⟵ Window(ibmAsks);
(ibmSales)⟵ SaleJoin(ibmBids,$lastAsk);
(result,$cnt) ⟵ Count(ibmSales,$cnt);
```

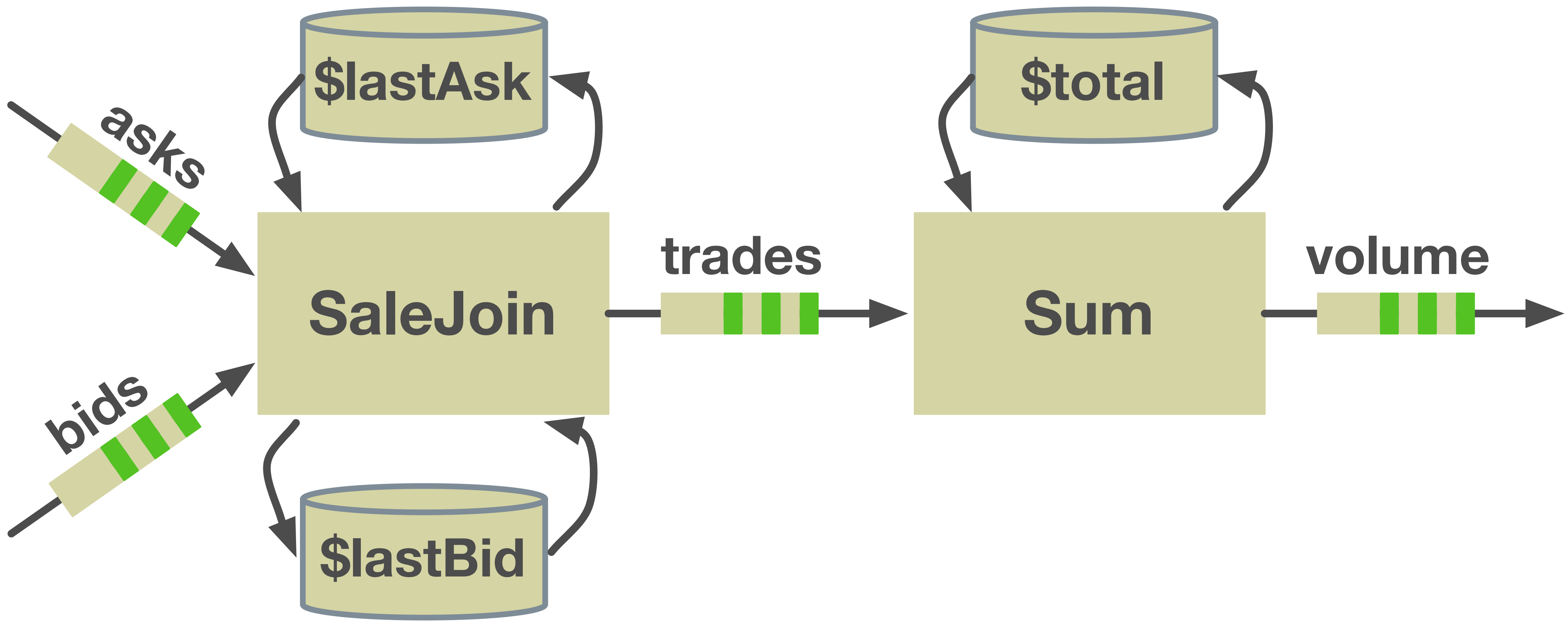
**Brooklet semantics:** $F_b \vdash \langle V, Q \rangle \longrightarrow \langle V', Q' \rangle$

$$\frac{\begin{array}{c} d, b = Q(q_i) \\ op = (\_,\_) \leftarrow f\,(\overline{q}, \overline{v}); \\ (\overline{b}', \overline{d}') = F_b(f)(d, i, V(\overline{v})) \\ V' = updateV(op, V, \overline{d}') \\ Q' = updateQ(op, Q, q_i, \overline{b}') \end{array}}{F_b \vdash \langle V, Q \rangle \longrightarrow \langle V', Q' \rangle} \quad \text{(E-FIREQUEUE)}$$

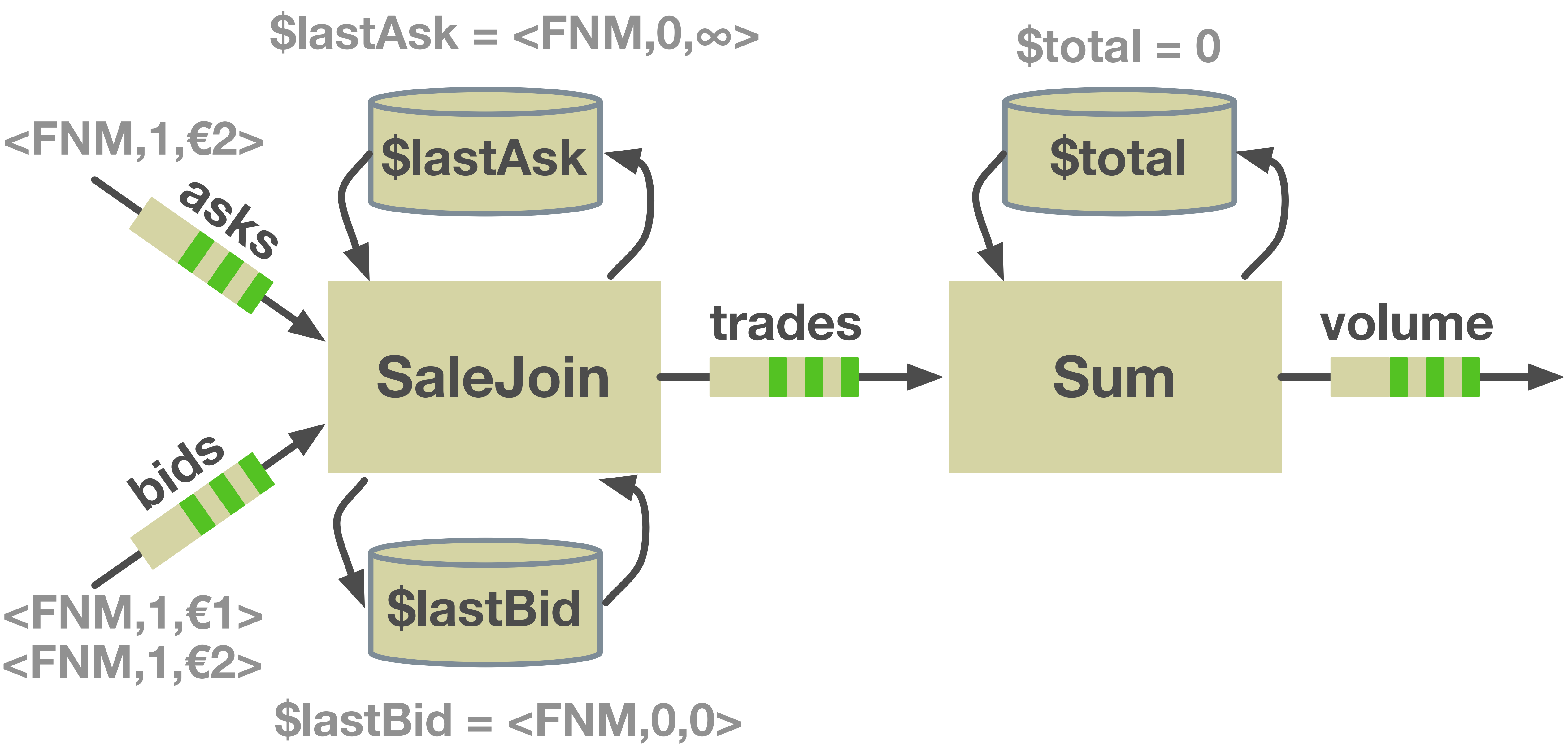
$$\frac{op = (\_, \overline{v}) \leftarrow f\,(\_,\_);}{updateV(op, V, \overline{d}) = [\overline{v} \mapsto \overline{d}]V} \quad \text{(E-UPDATEV)}$$

$$\frac{\begin{array}{c} op = (\overline{q}, \_) \leftarrow f\,(\_,\_); \\ d_f, b_f = Q(q_f) \\ Q' = [q_f \mapsto b_f]Q \\ Q'' = [\forall q_i \in \overline{q} : q_i \mapsto Q(q_i), b_i]Q' \end{array}}{updateQ(op, Q, q_f, \overline{b}) = Q''} \quad \text{(E-UPDATEQ)}$$

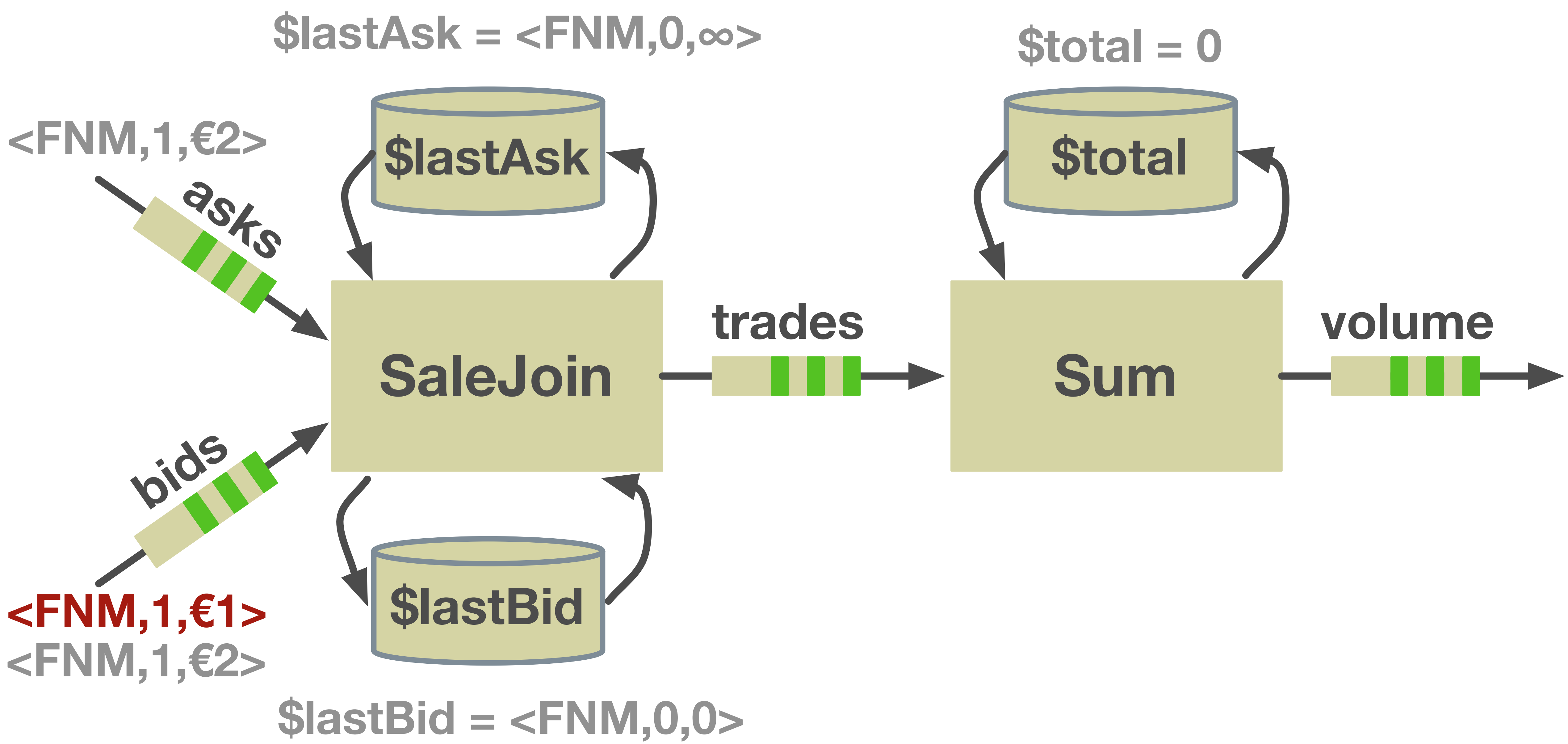
# Example:
# A Fannie Mae Bid/Ask Join



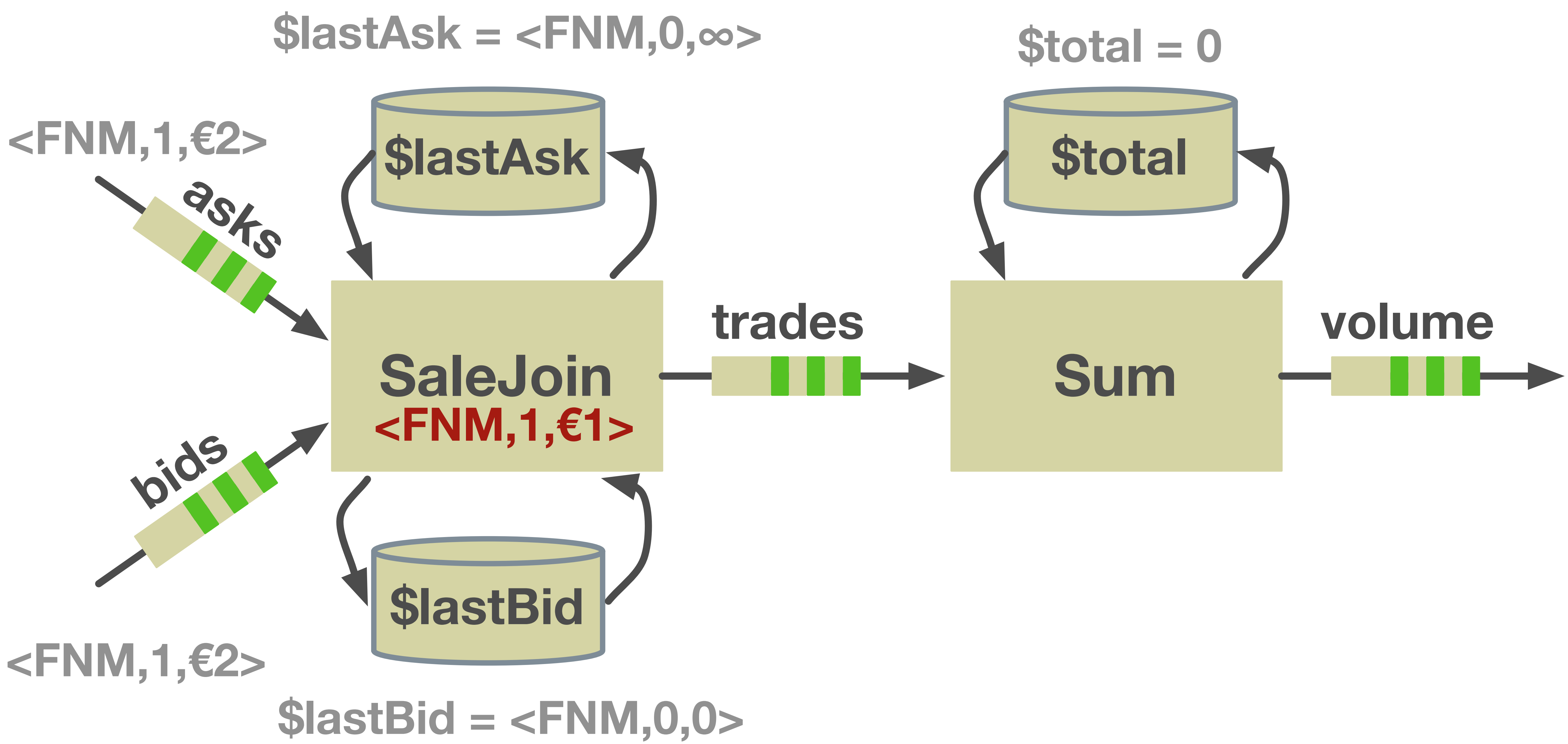
# Example:
# A Fannie Mae Bid/Ask Join



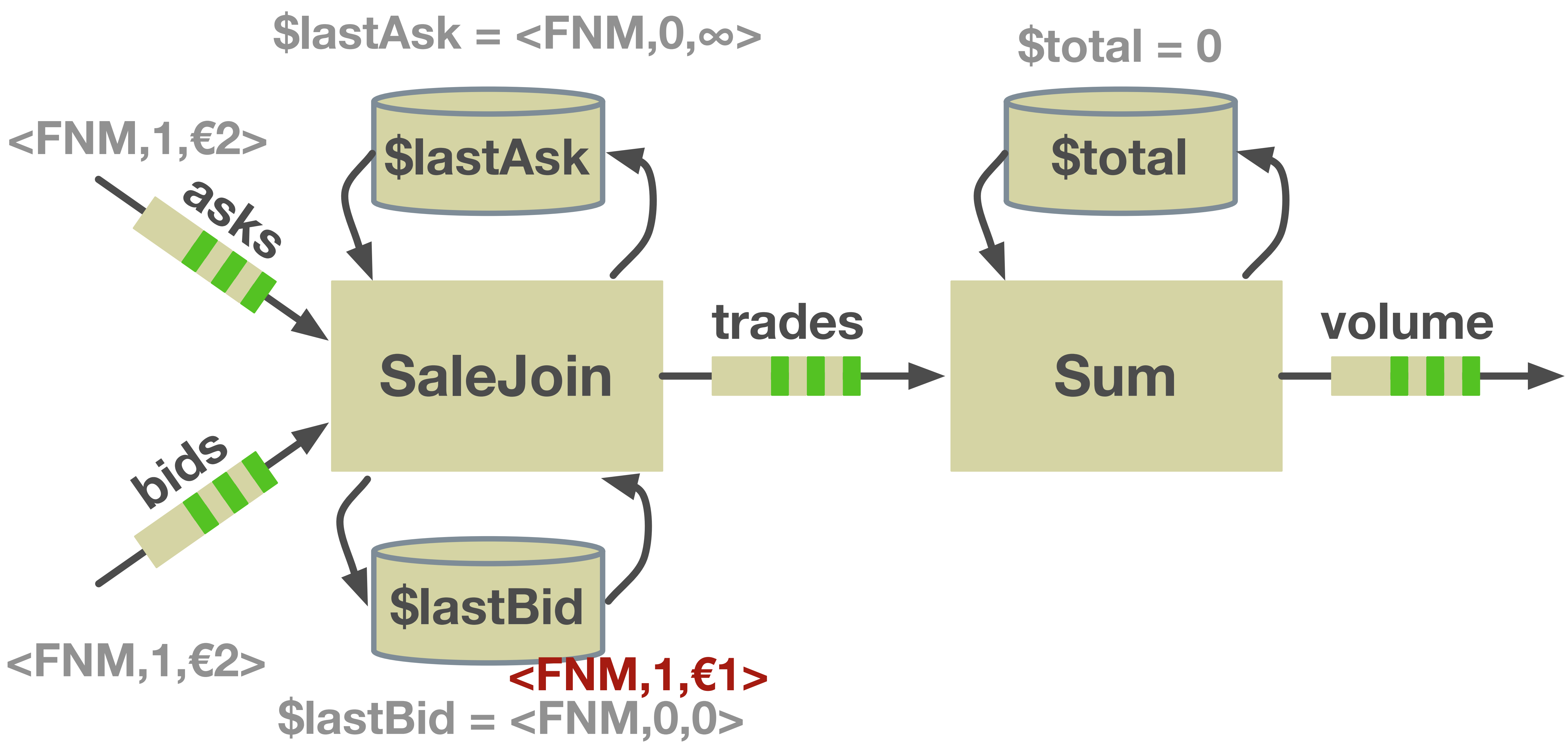
# Example:
# A Fannie Mae Bid/Ask Join



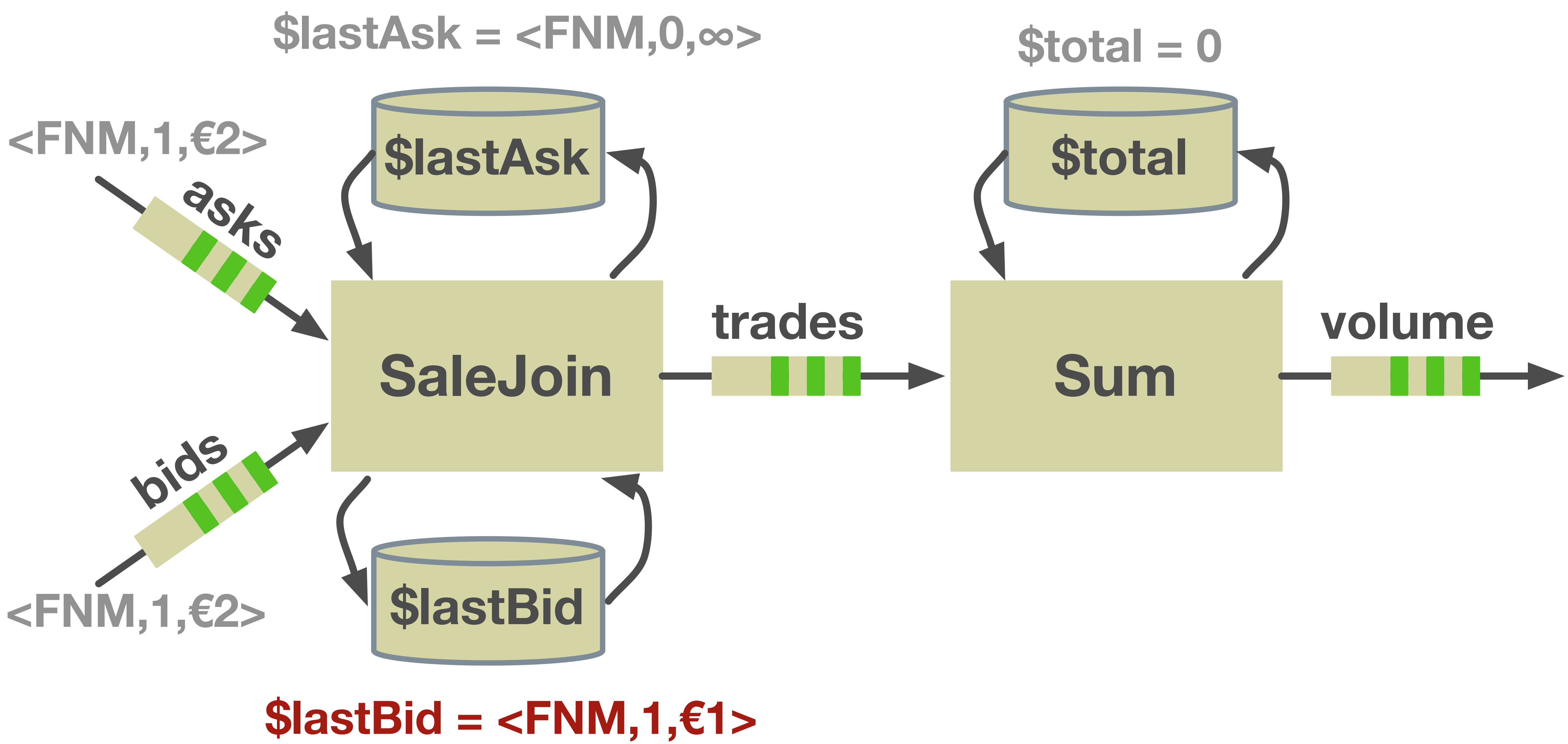
# Example:
# A Fannie Mae Bid/Ask Join



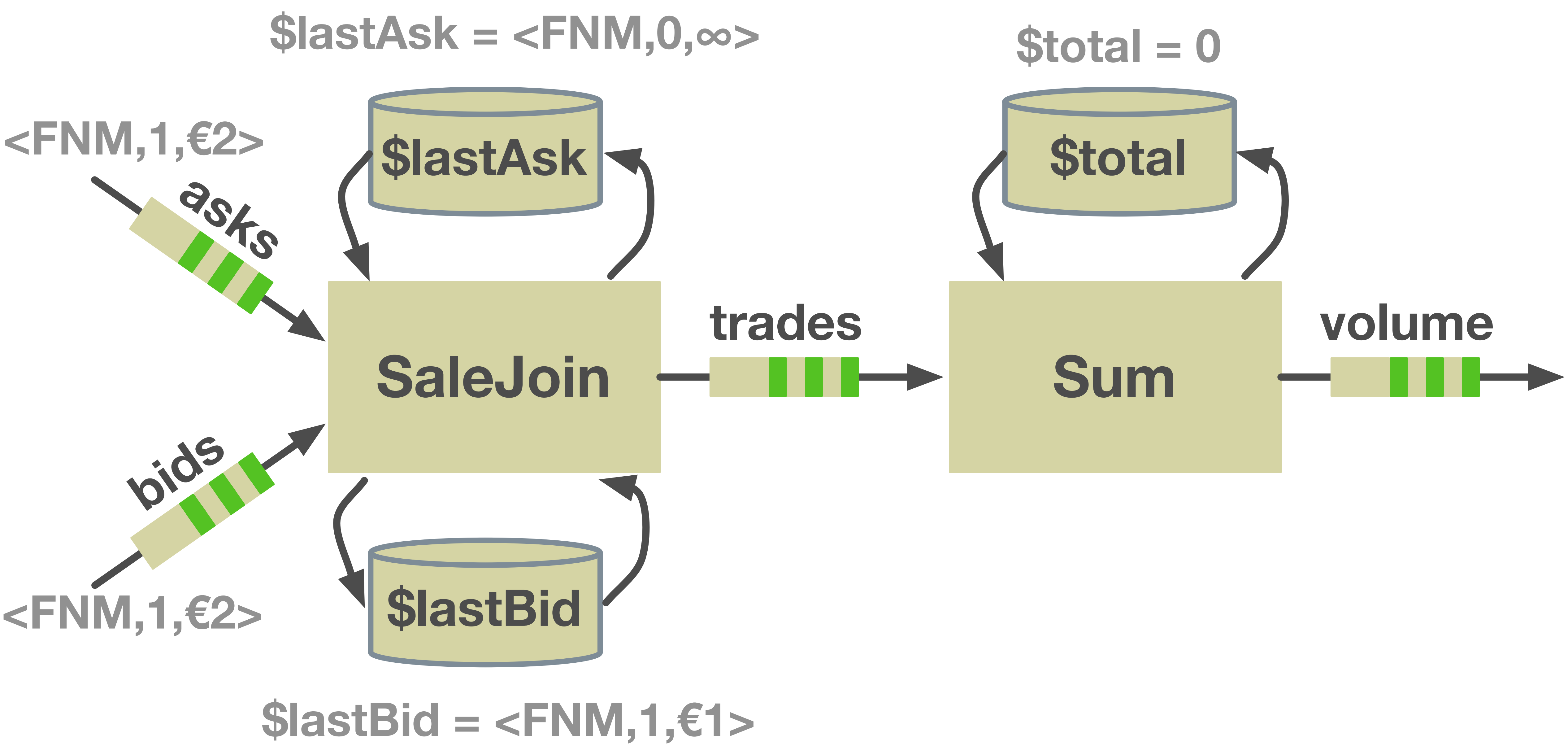
# Example: A Fannie Mae Bid/Ask Join



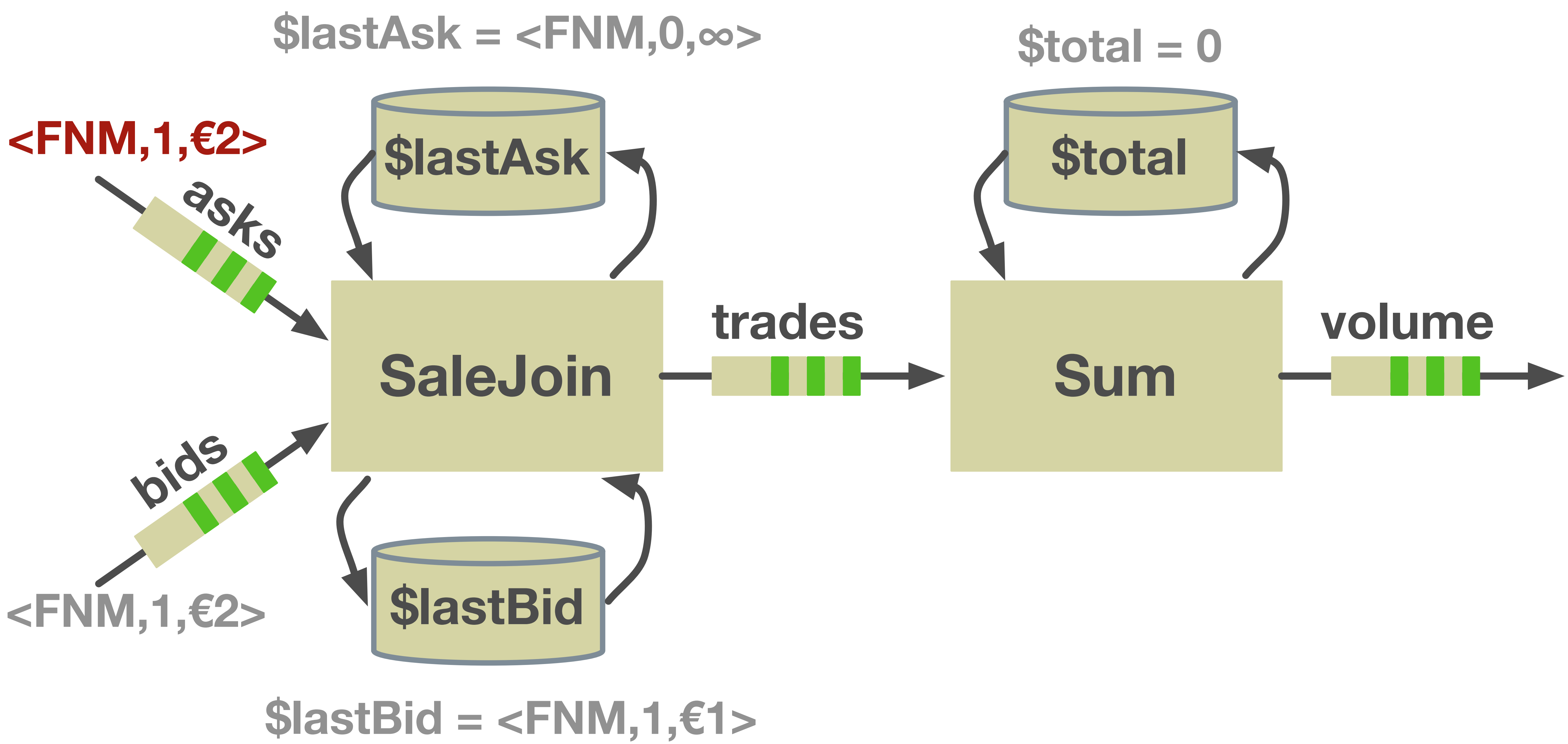
# Example:
# A Fannie Mae Bid/Ask Join



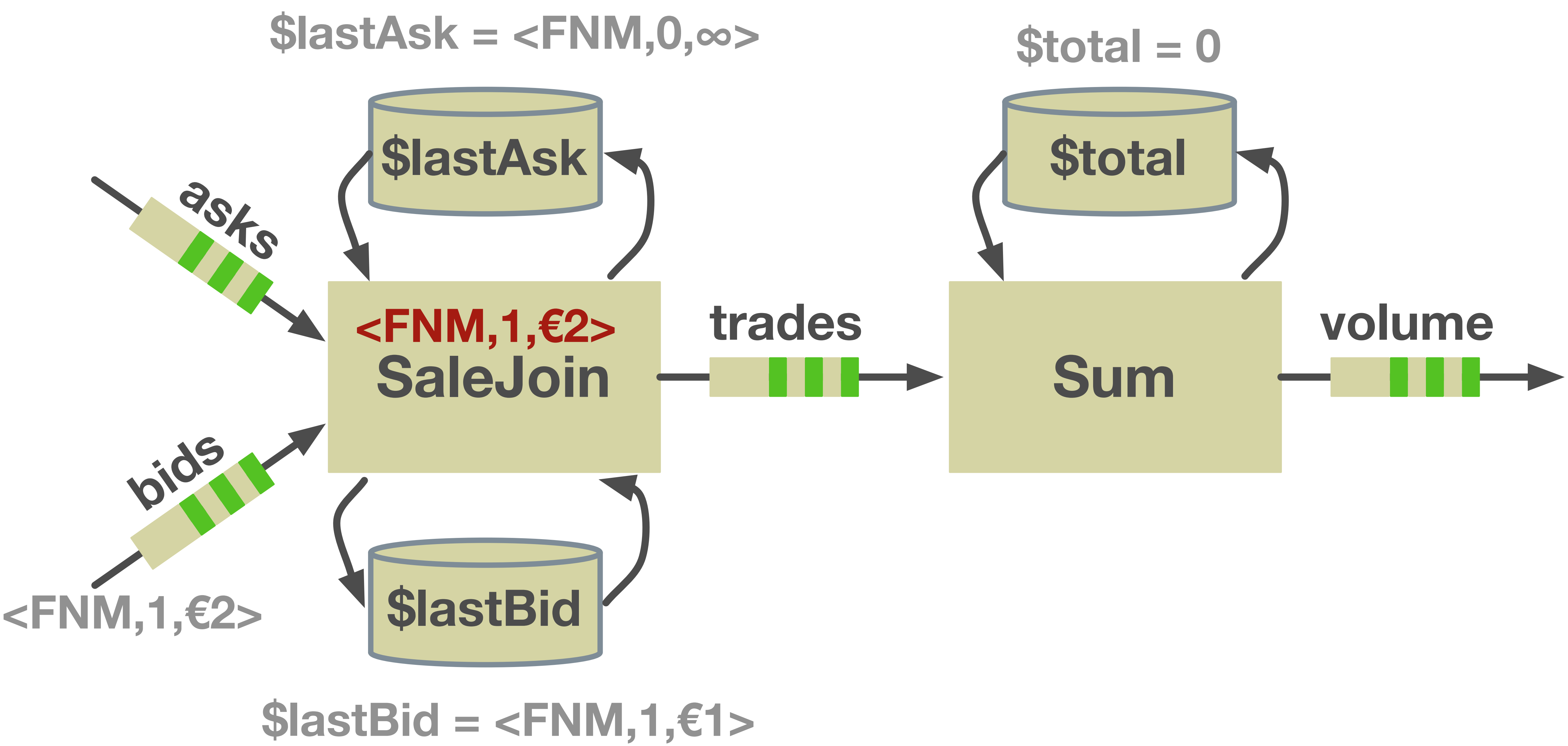
# Example: A Fannie Mae Bid/Ask Join



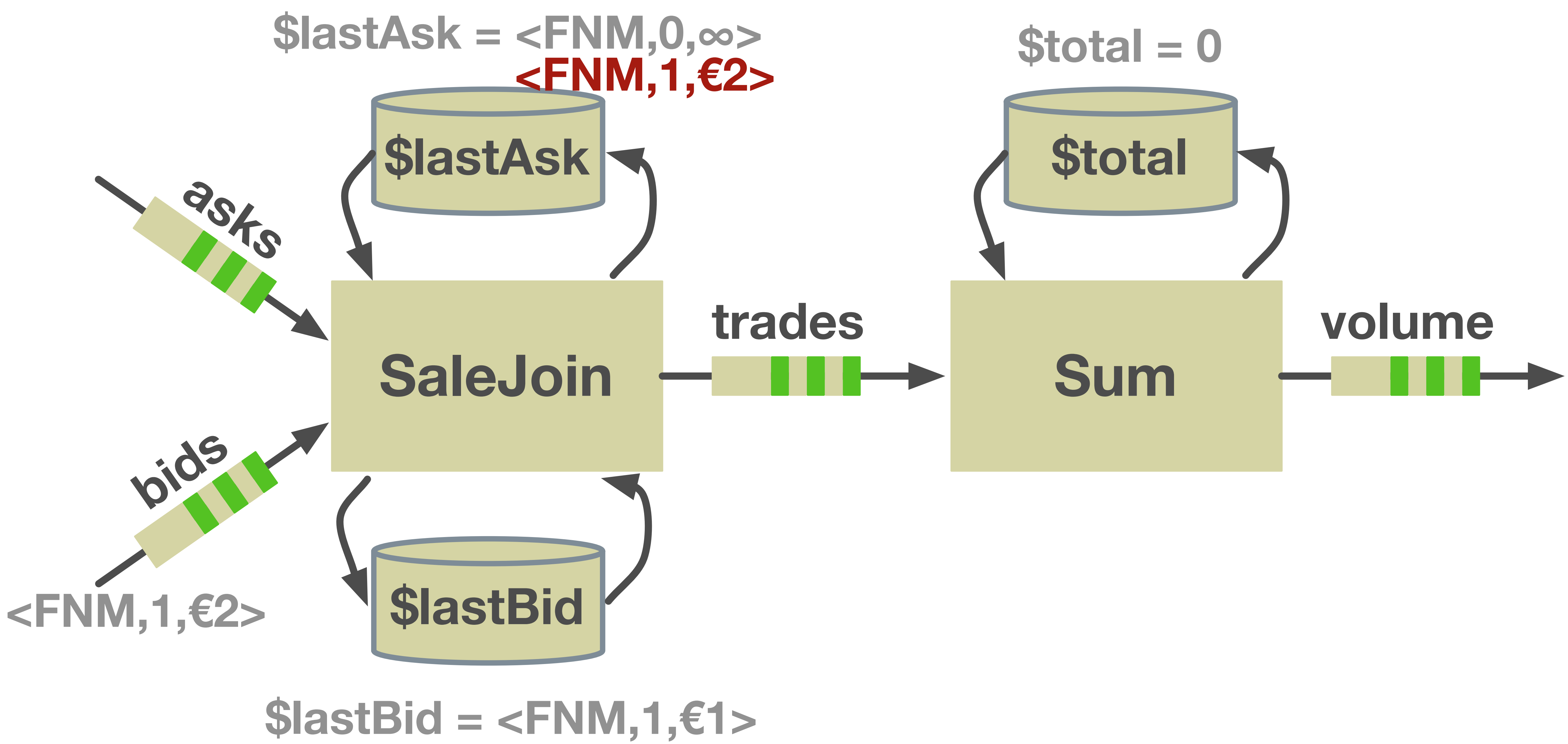
# Example: A Fannie Mae Bid/Ask Join



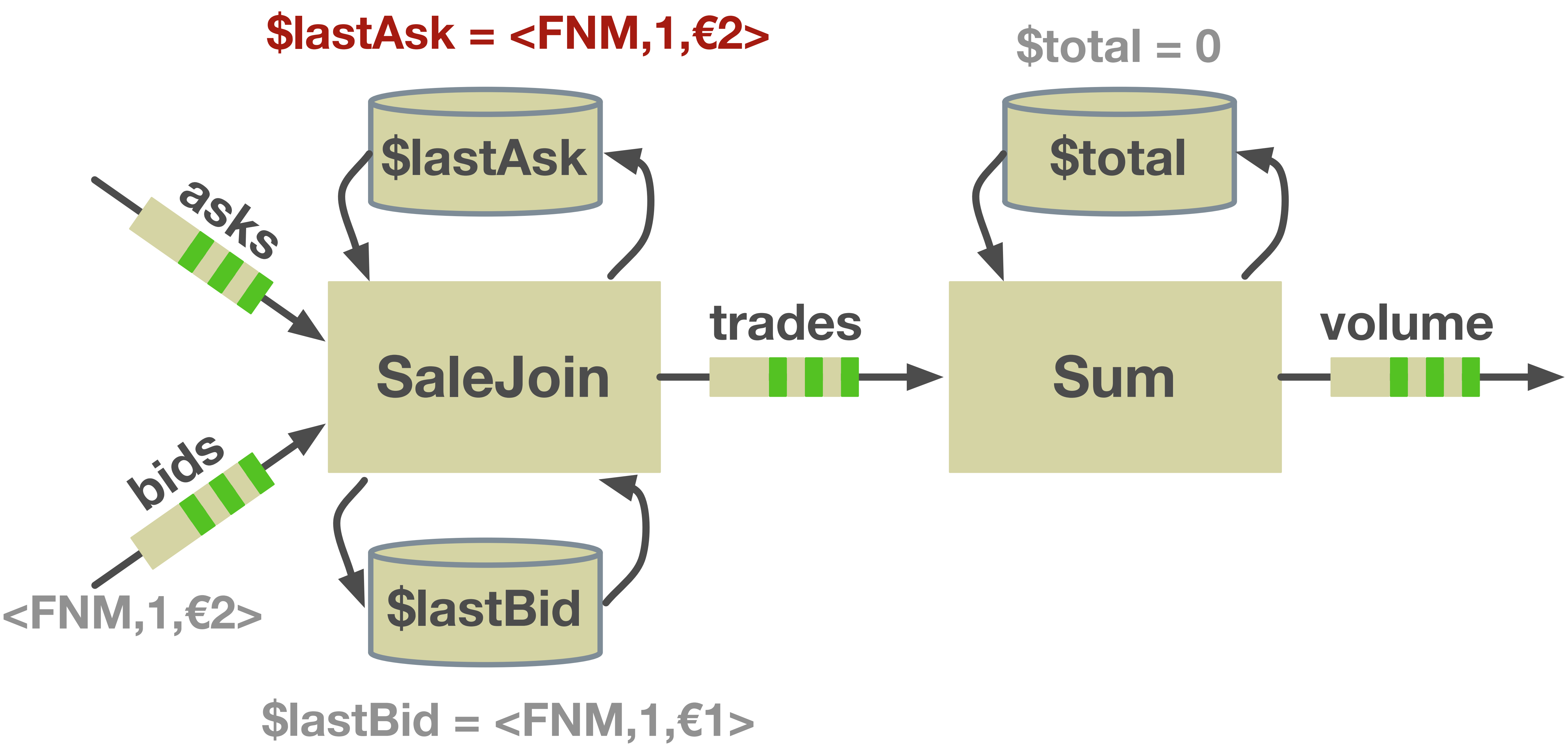
# Example:
# A Fannie Mae Bid/Ask Join



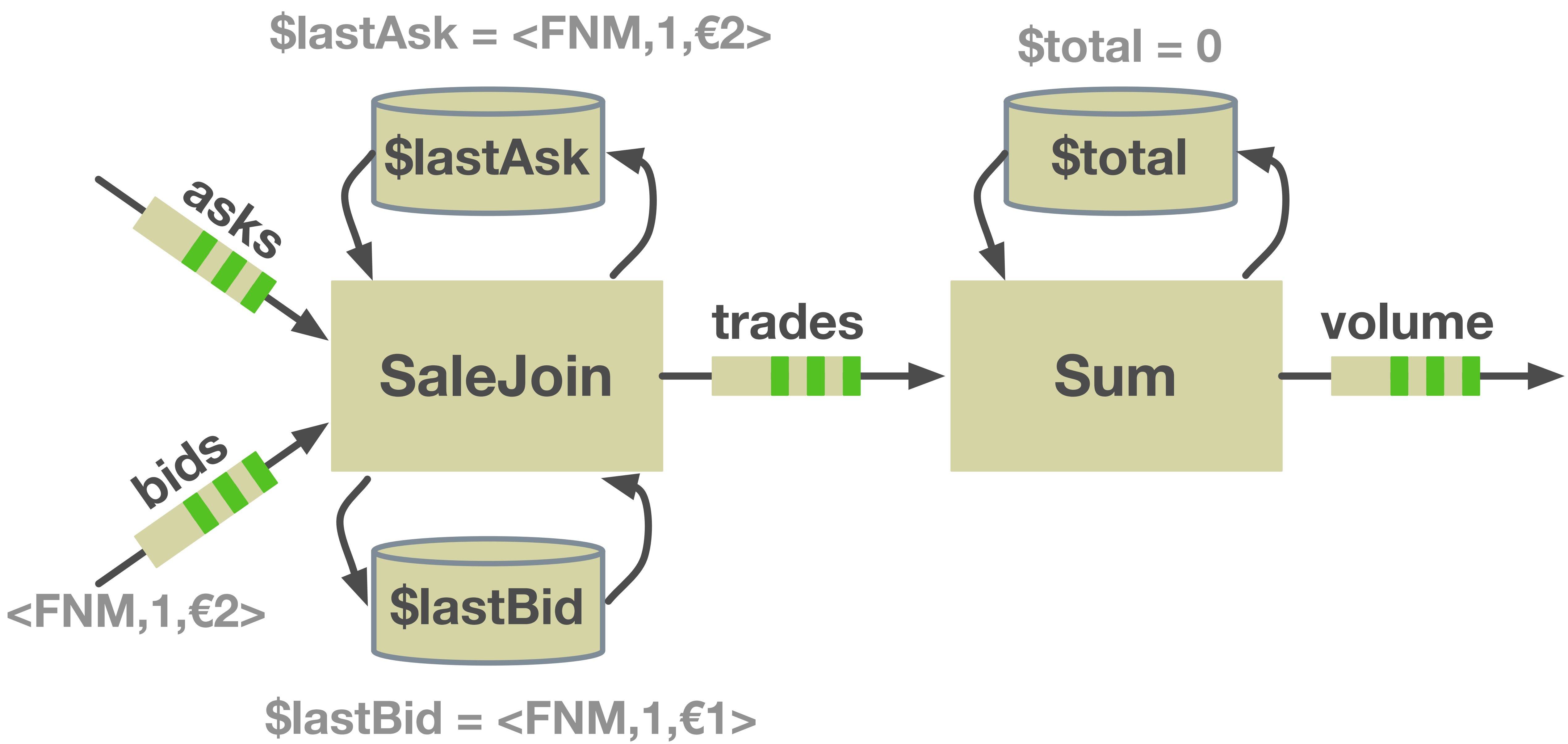
# Example: A Fannie Mae Bid/Ask Join



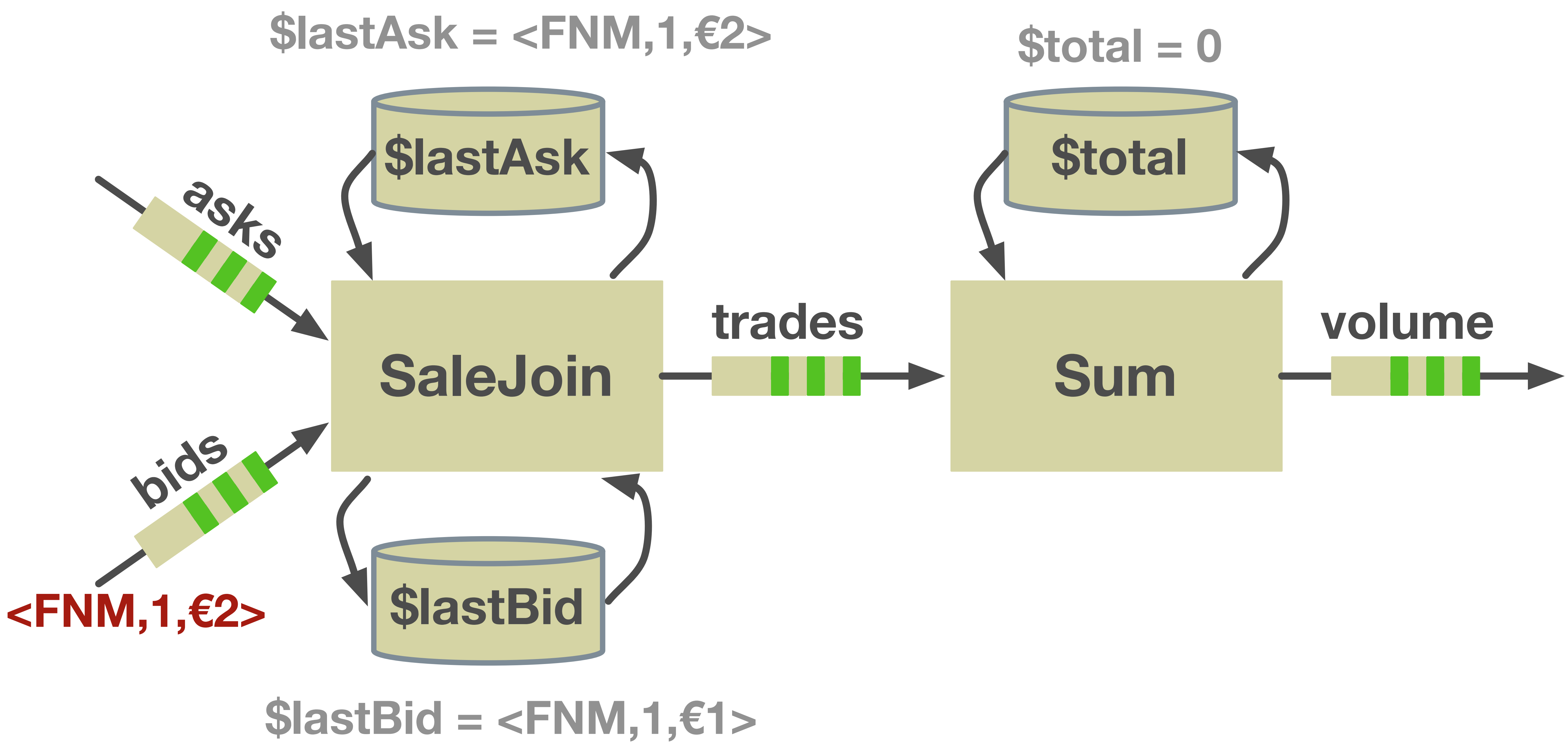
# Example:
# A Fannie Mae Bid/Ask Join



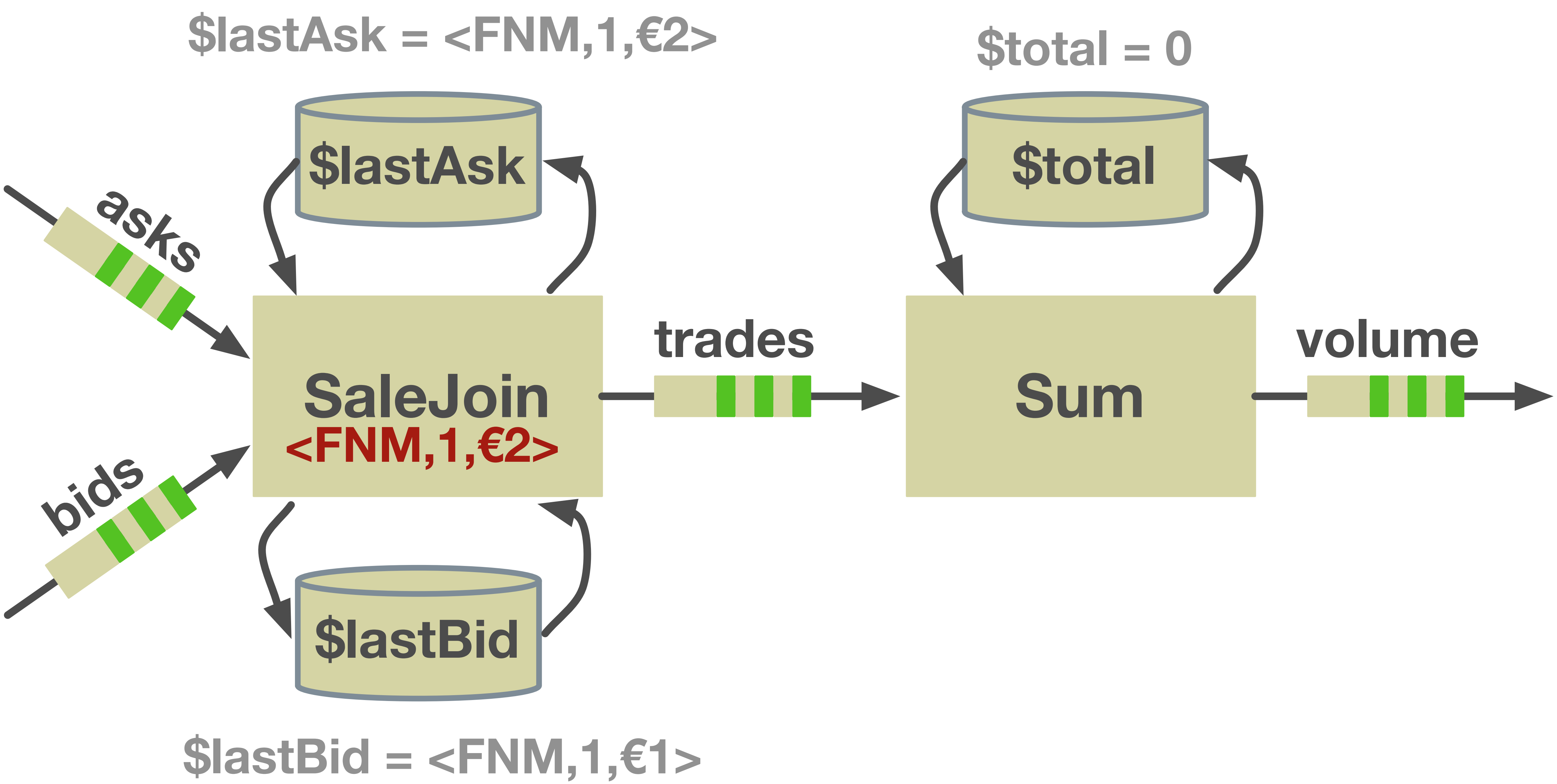
# Example:
# A Fannie Mae Bid/Ask Join



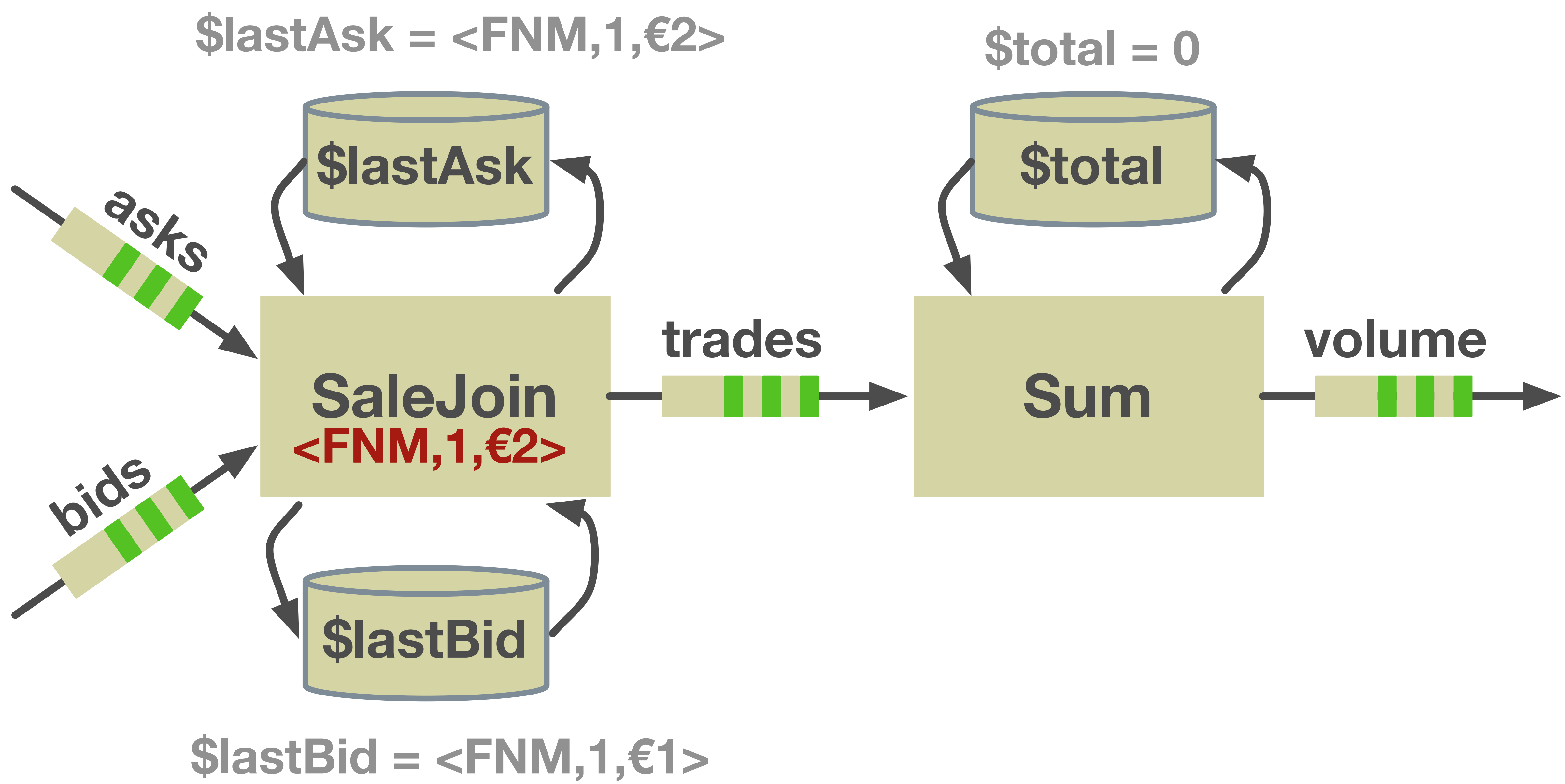
# Example:
# A Fannie Mae Bid/Ask Join



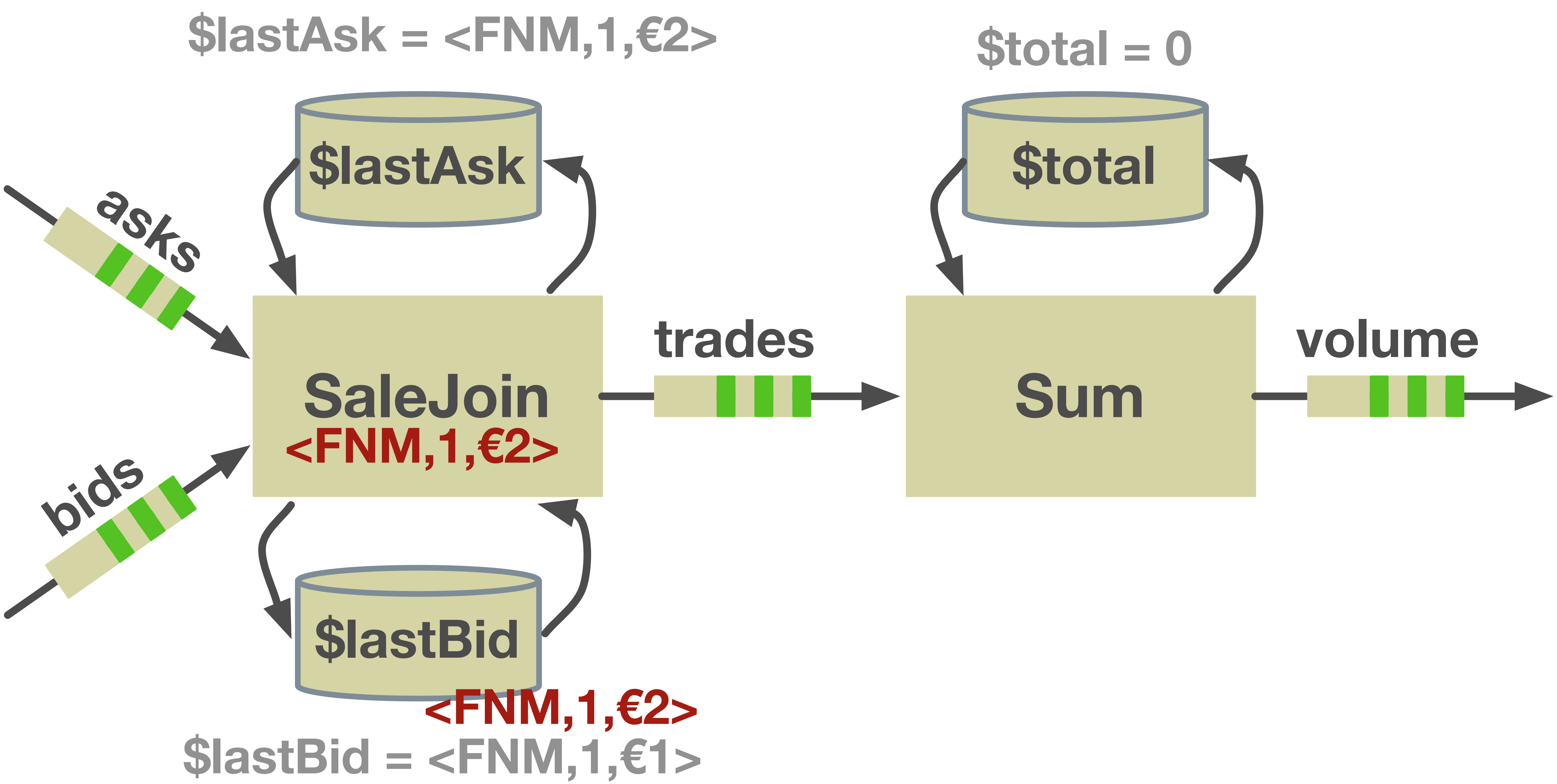
# Example:
# A Fannie Mae Bid/Ask Join



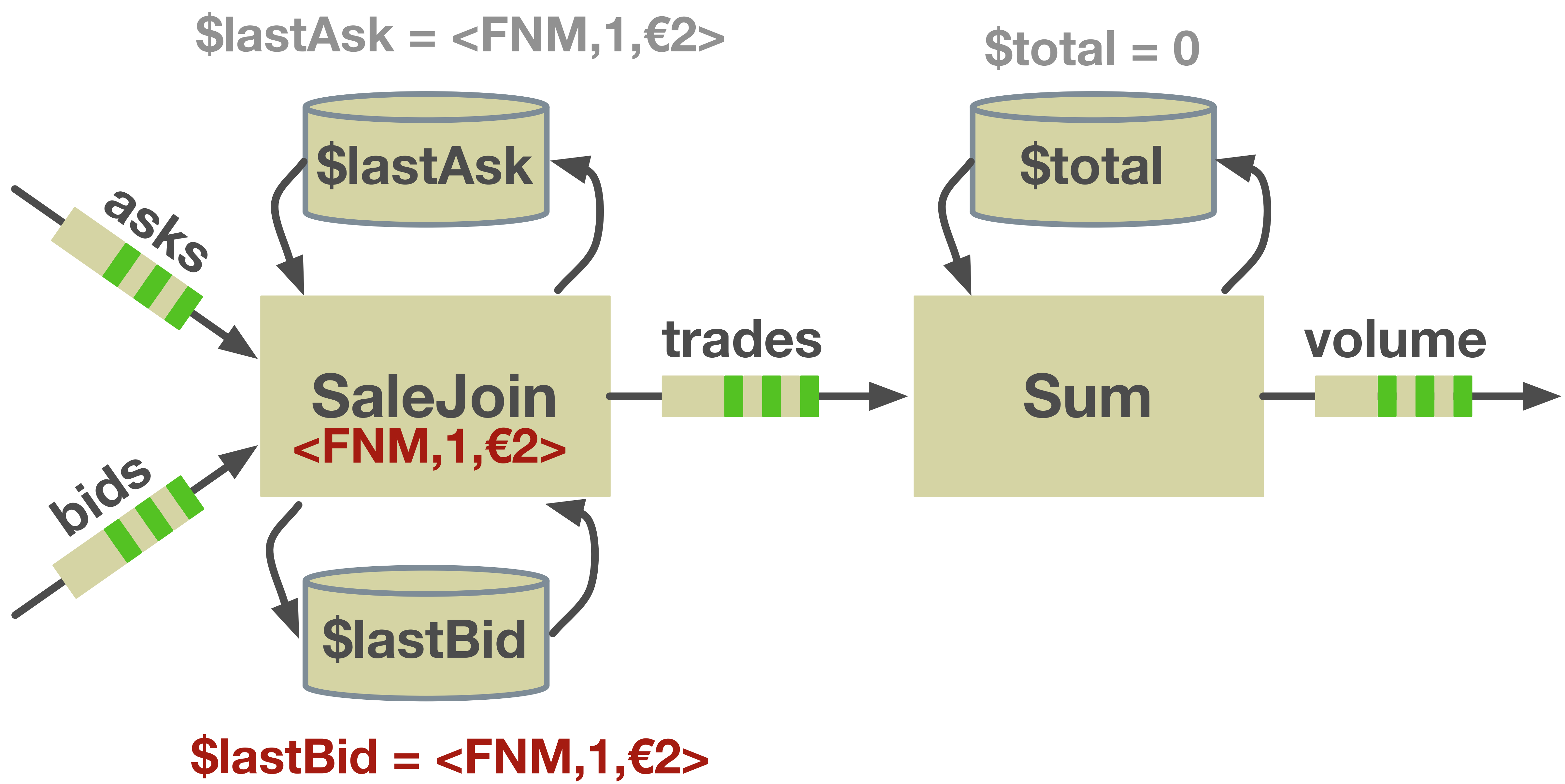
# Example: A Fannie Mae Bid/Ask Join



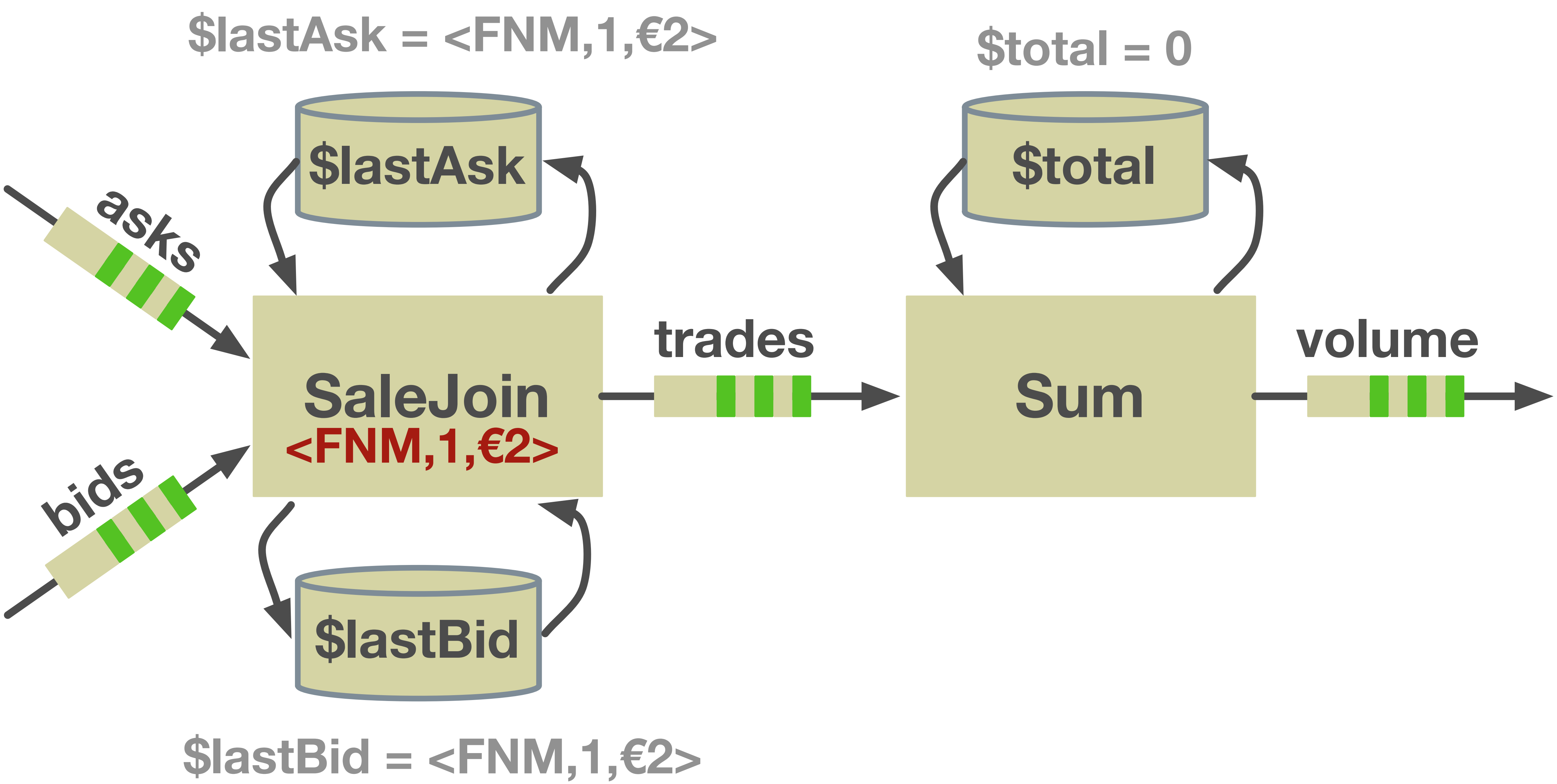
# Example:
# A Fannie Mae Bid/Ask Join



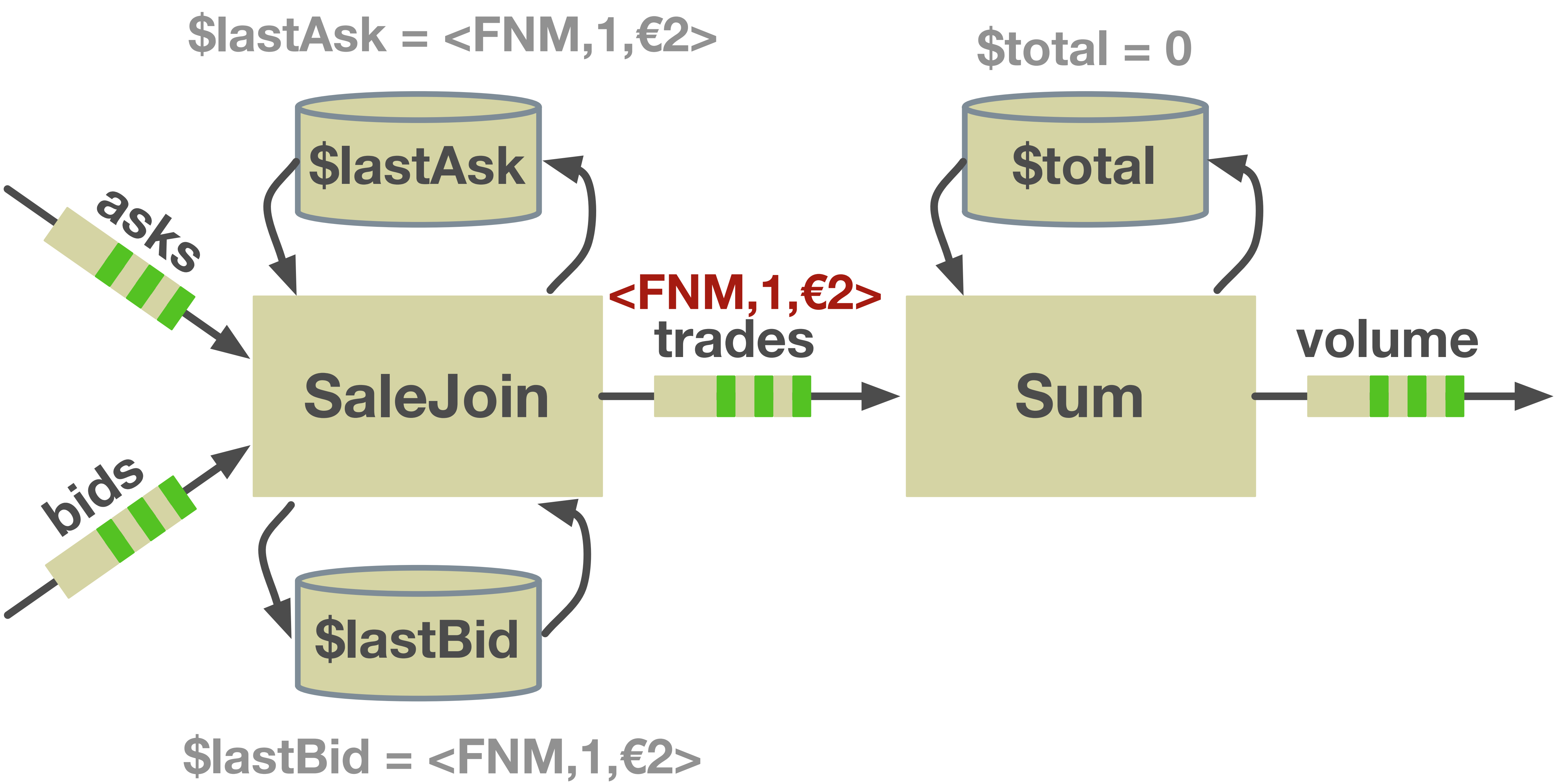
# Example: A Fannie Mae Bid/Ask Join



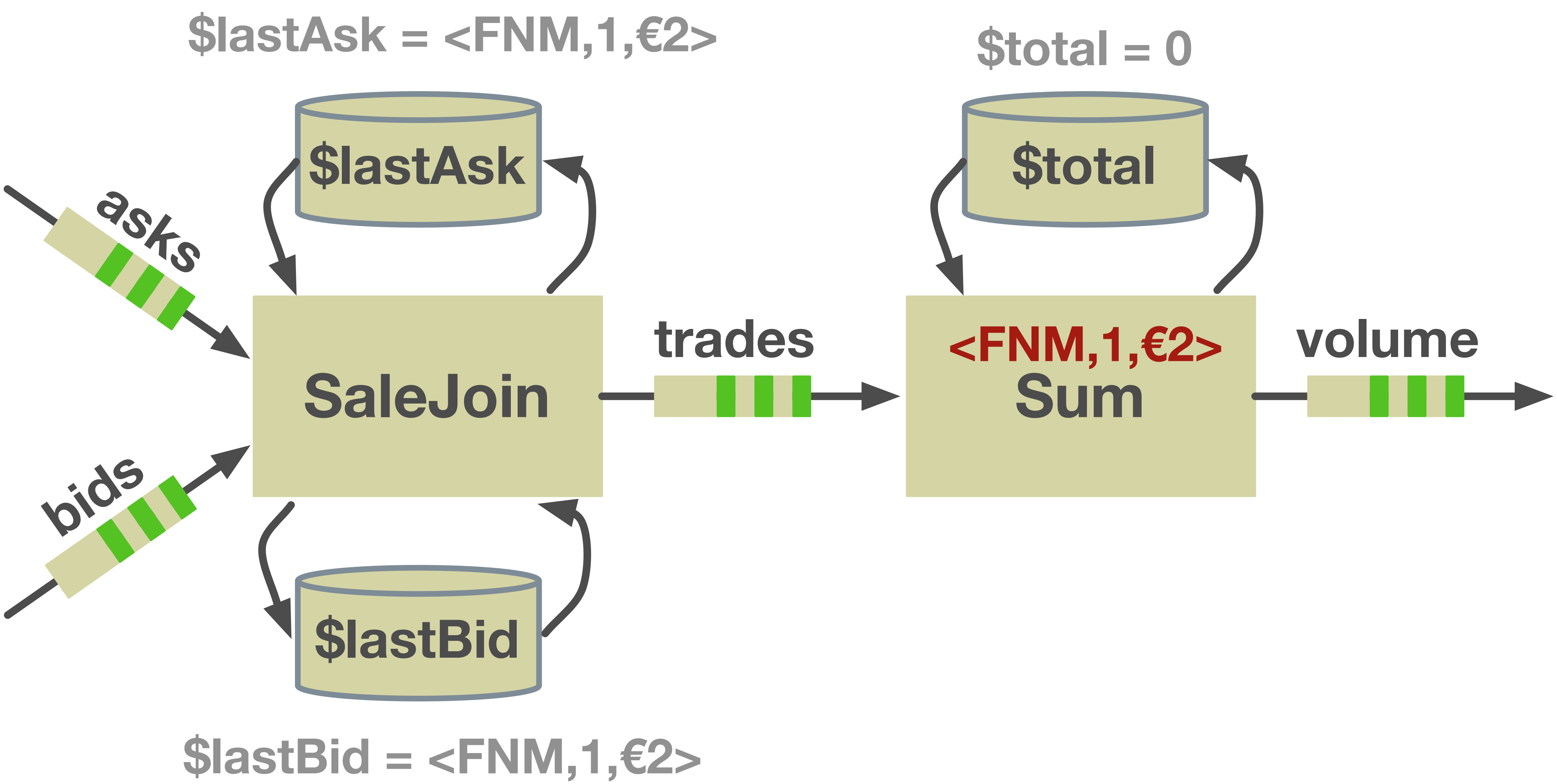
# Example: A Fannie Mae Bid/Ask Join



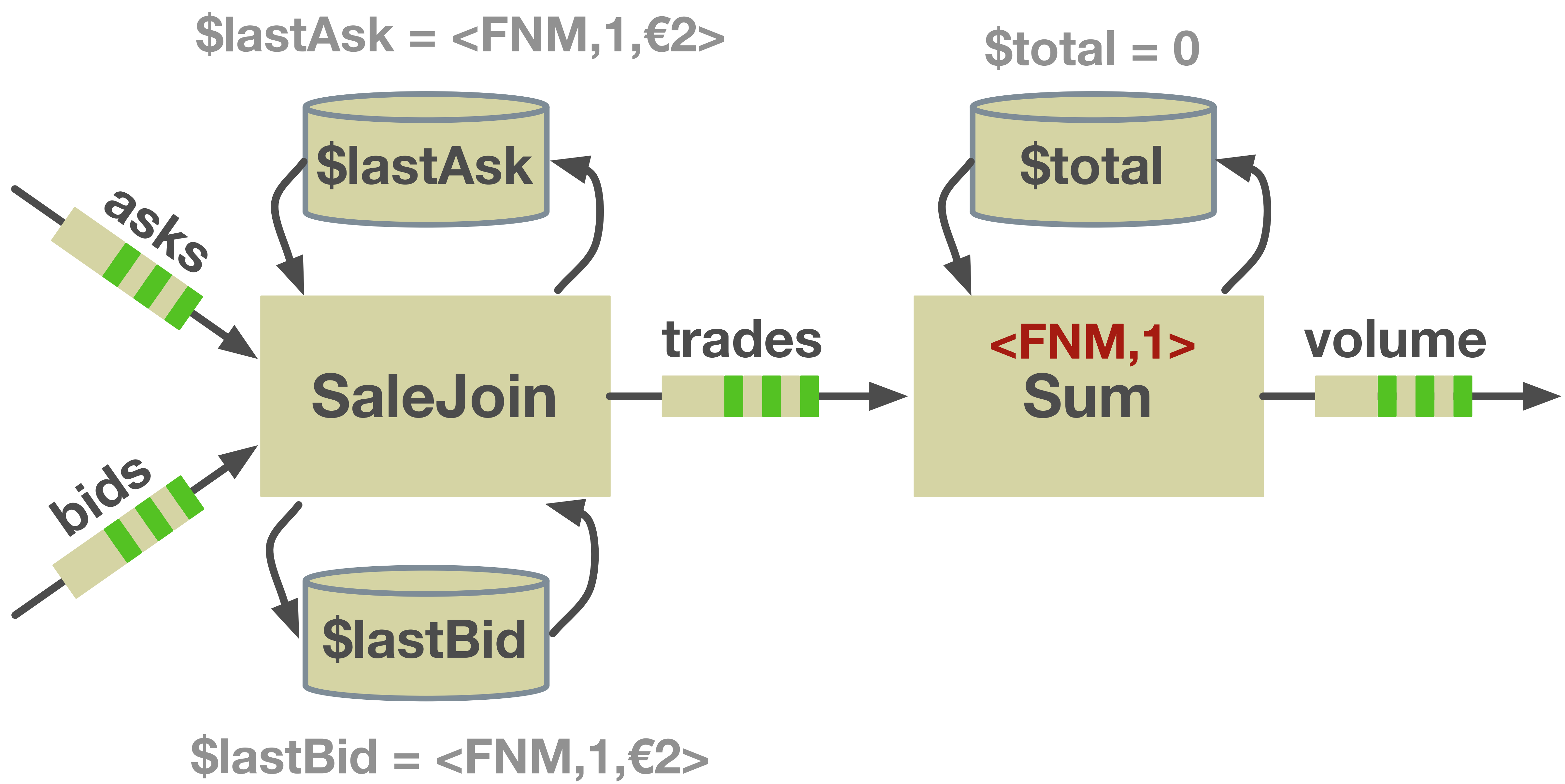
# Example: A Fannie Mae Bid/Ask Join



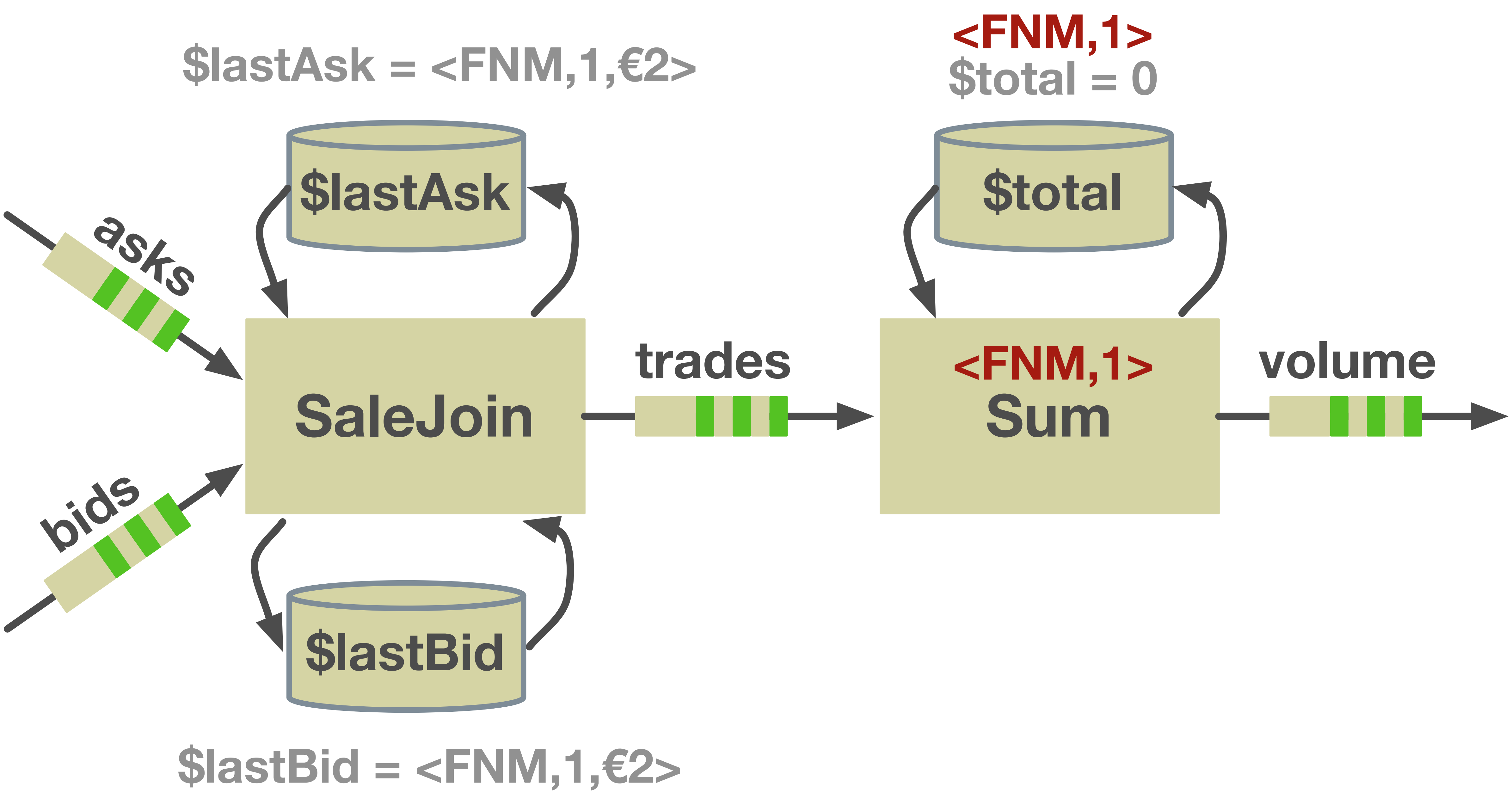
# Example:
# A Fannie Mae Bid/Ask Join



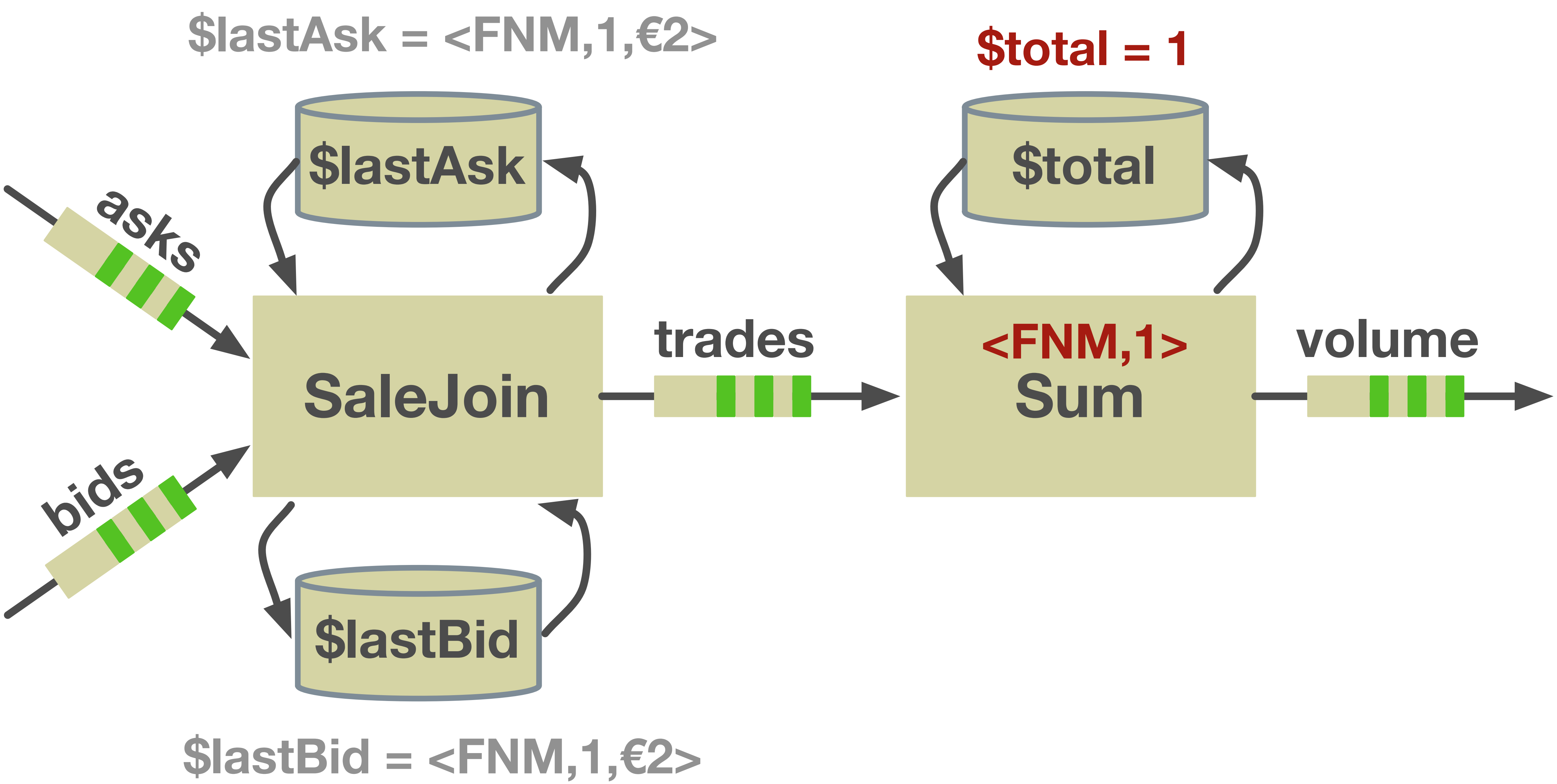
# Example: A Fannie Mae Bid/Ask Join



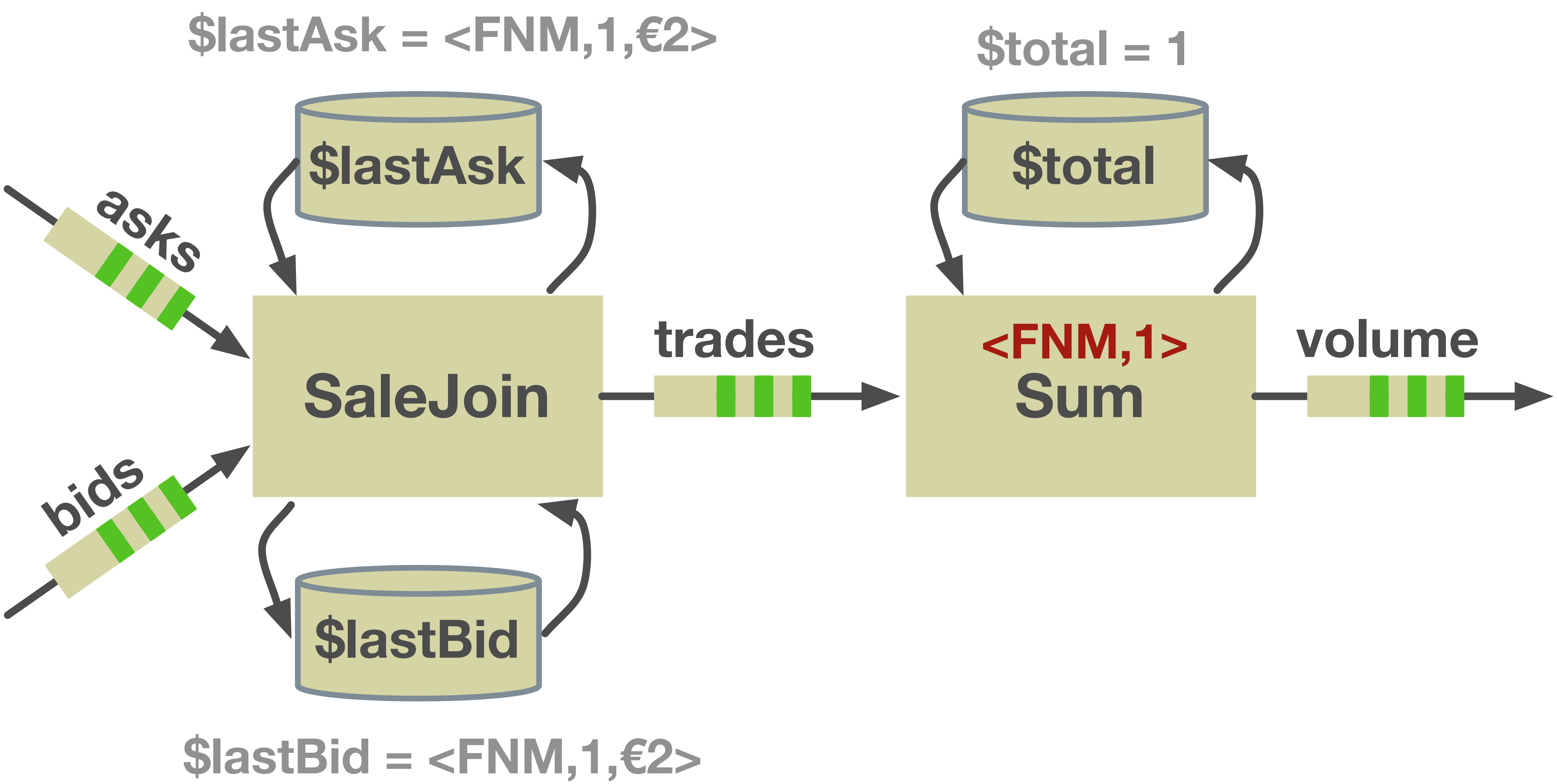
# Example:
# A Fannie Mae Bid/Ask Join



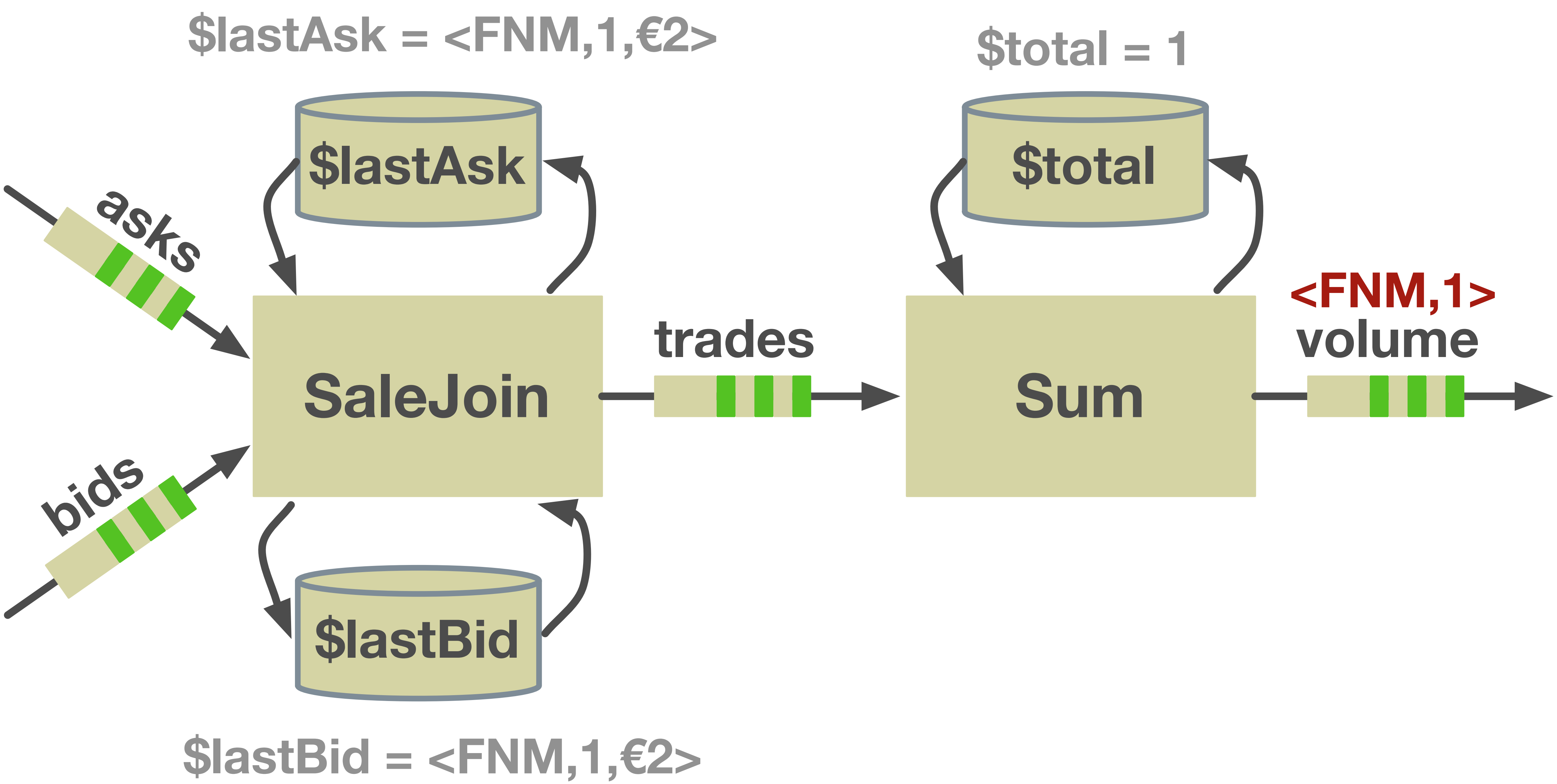
# Example:
# A Fannie Mae Bid/Ask Join



# Example:
# A Fannie Mae Bid/Ask Join



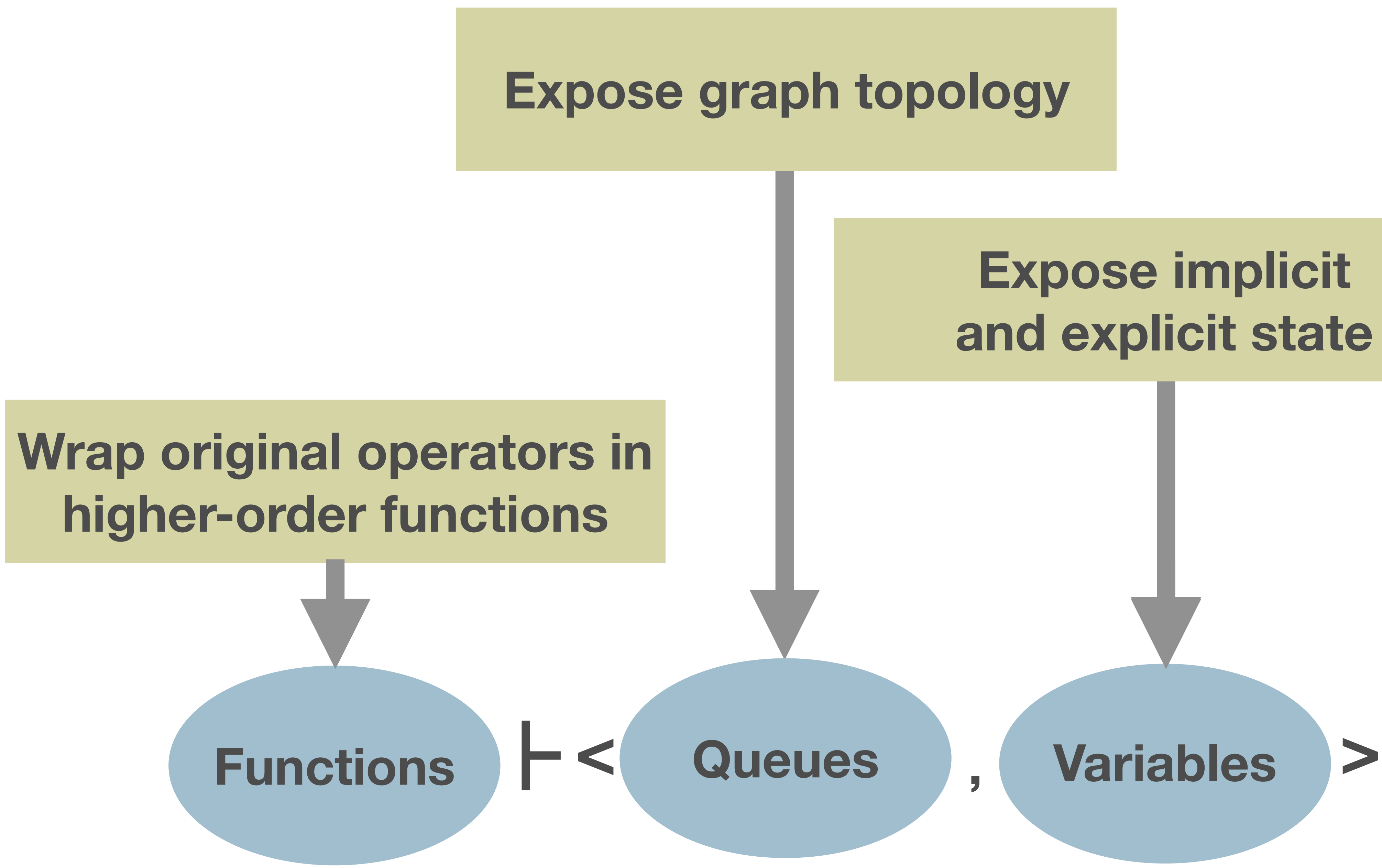
# Example: A Fannie Mae Bid/Ask Join



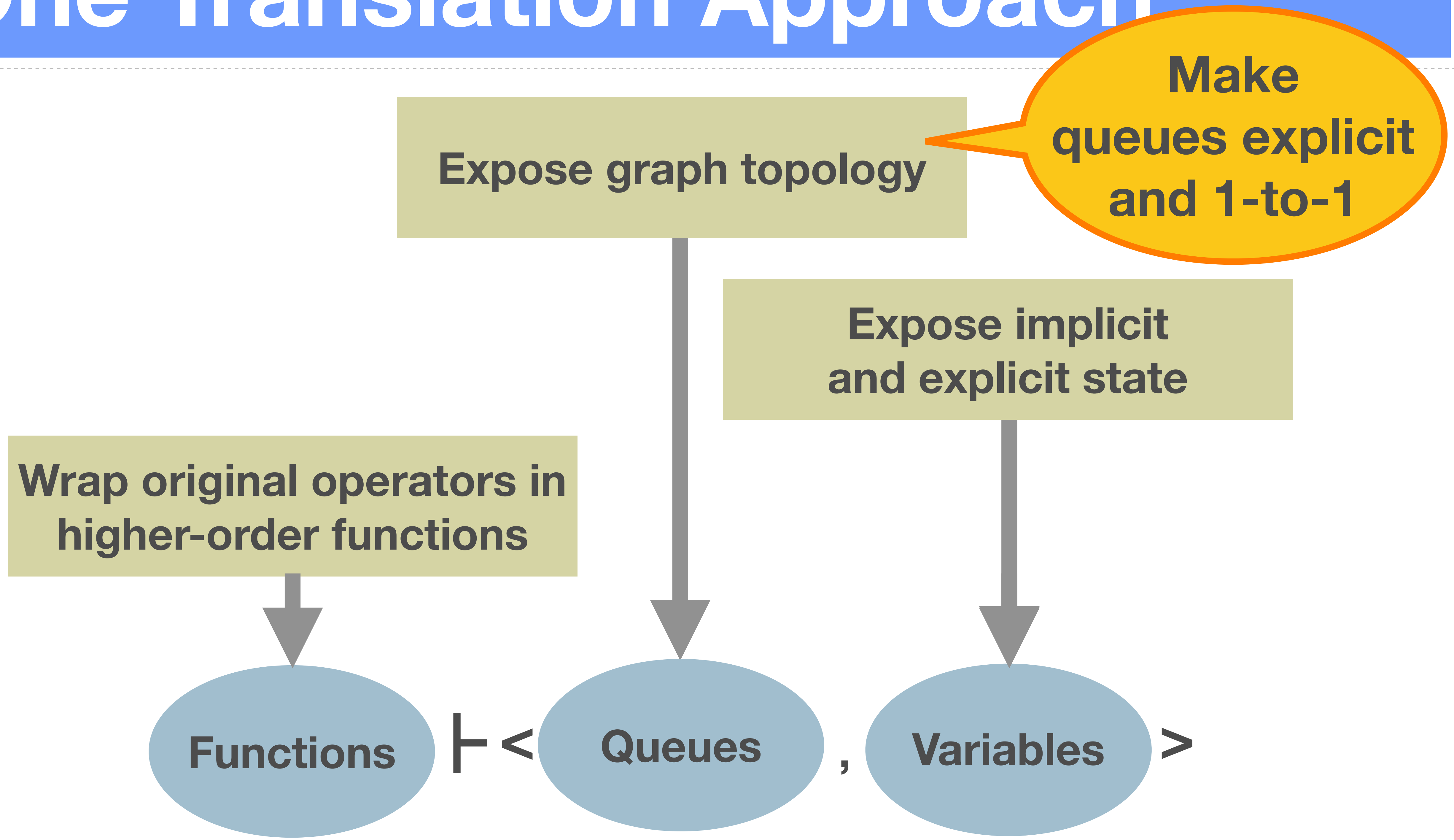
# Translations

**Demonstrating Brooklet's generality
by translating three rather diverse streaming languages**

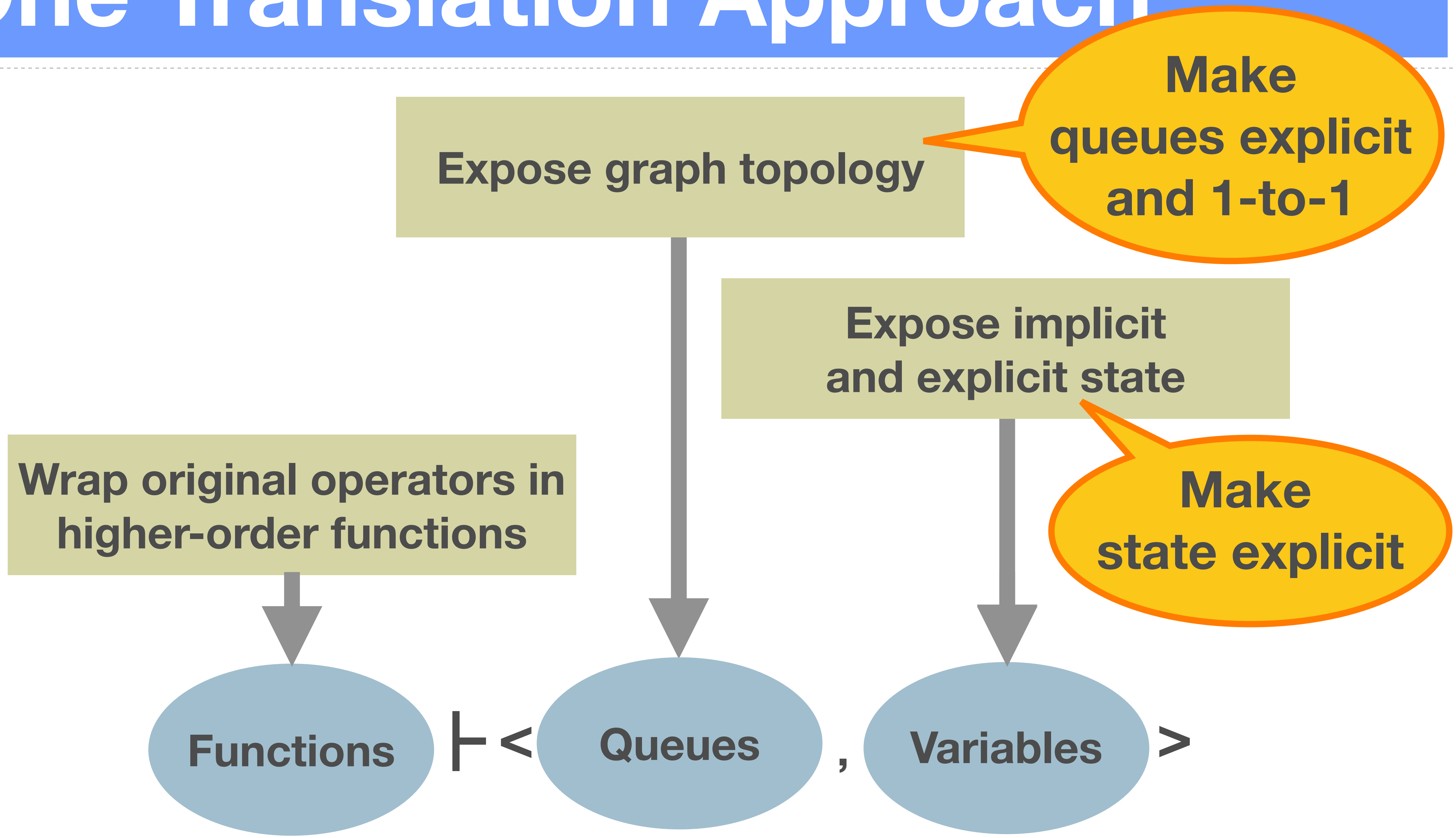
# CQL, StreamIt, Sawzall: One Translation Approach



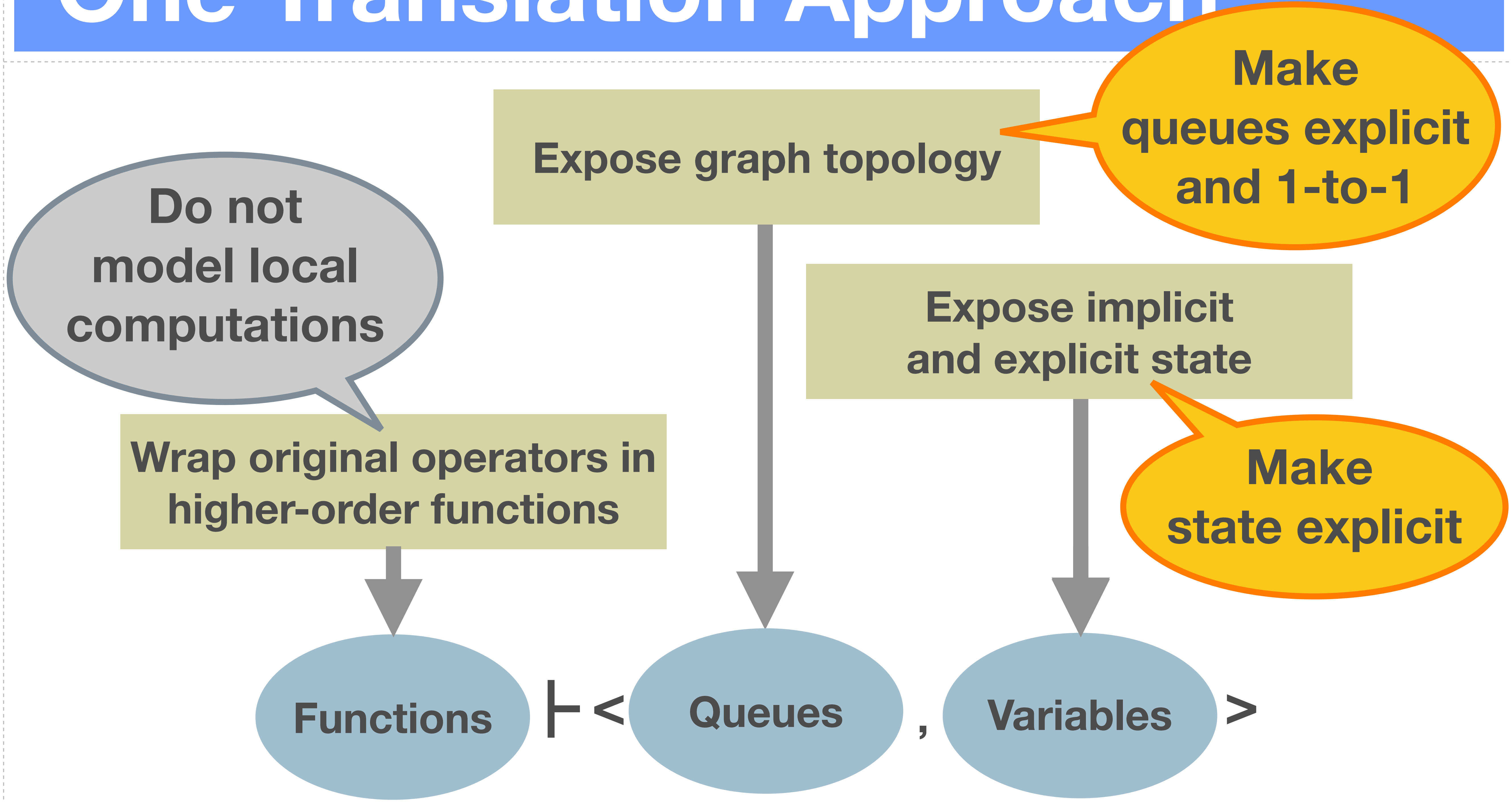
# CQL, StreamIt, Sawzall: One Translation Approach



# CQL, StreamIt, Sawzall: One Translation Approach



# CQL, StreamIt, Sawzall: One Translation Approach



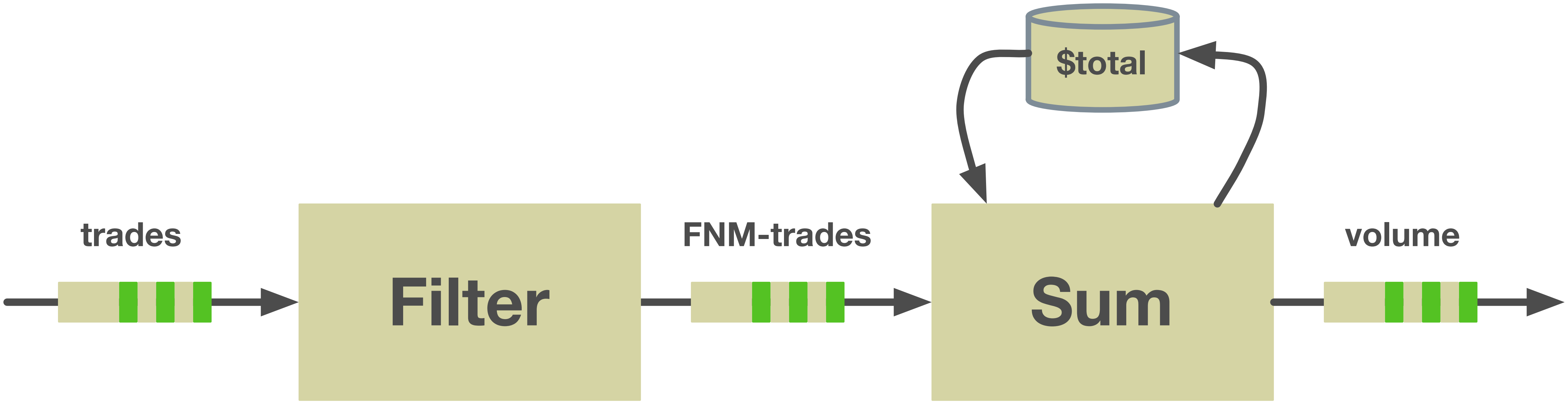
# Example: CQL to Brooklet

select *Sum(shares)* from *trades*
where *trades.ticker* = *"FNM"*

*CQL*

---

*Brooklet*

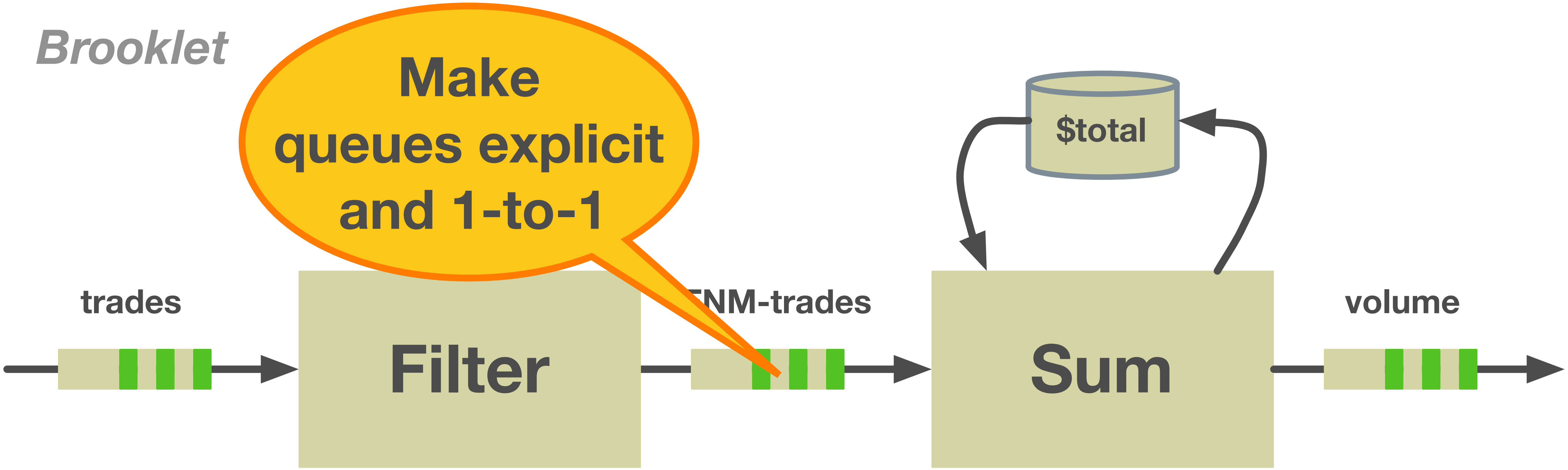


# Example: CQL to Brooklet

select *Sum(shares)* from *trades*
where *trades.ticker* = *"FNM"*

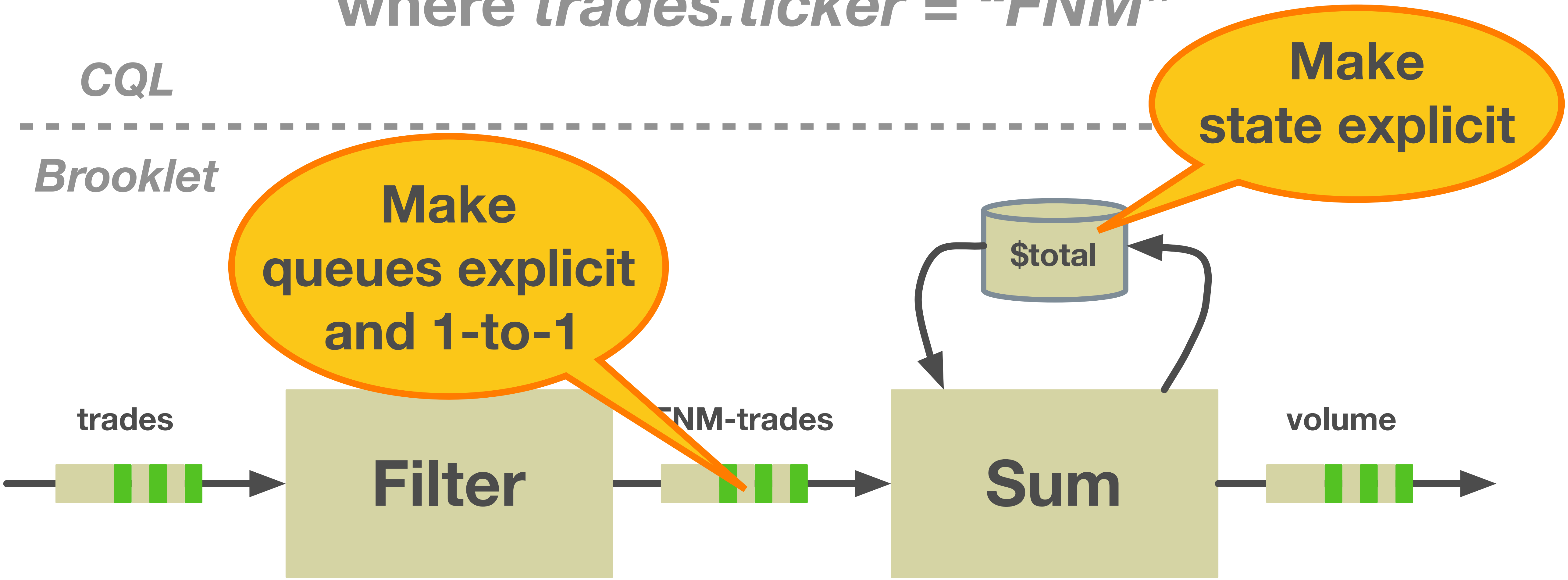
*CQL*

*Brooklet*



# Example: CQL to Brooklet

select *Sum(shares)* from *trades*
where *trades.ticker* = *"FNM"*



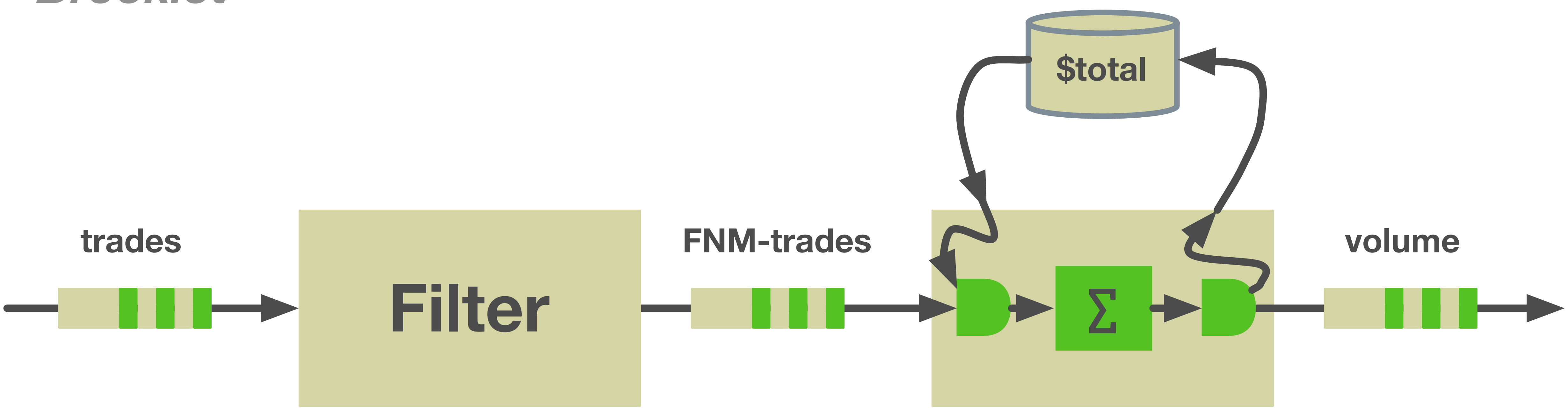
# Example: CQL to Brooklet

select *Sum(shares)* from *trades*
where *trades.ticker* = *"FNM"*

*CQL*

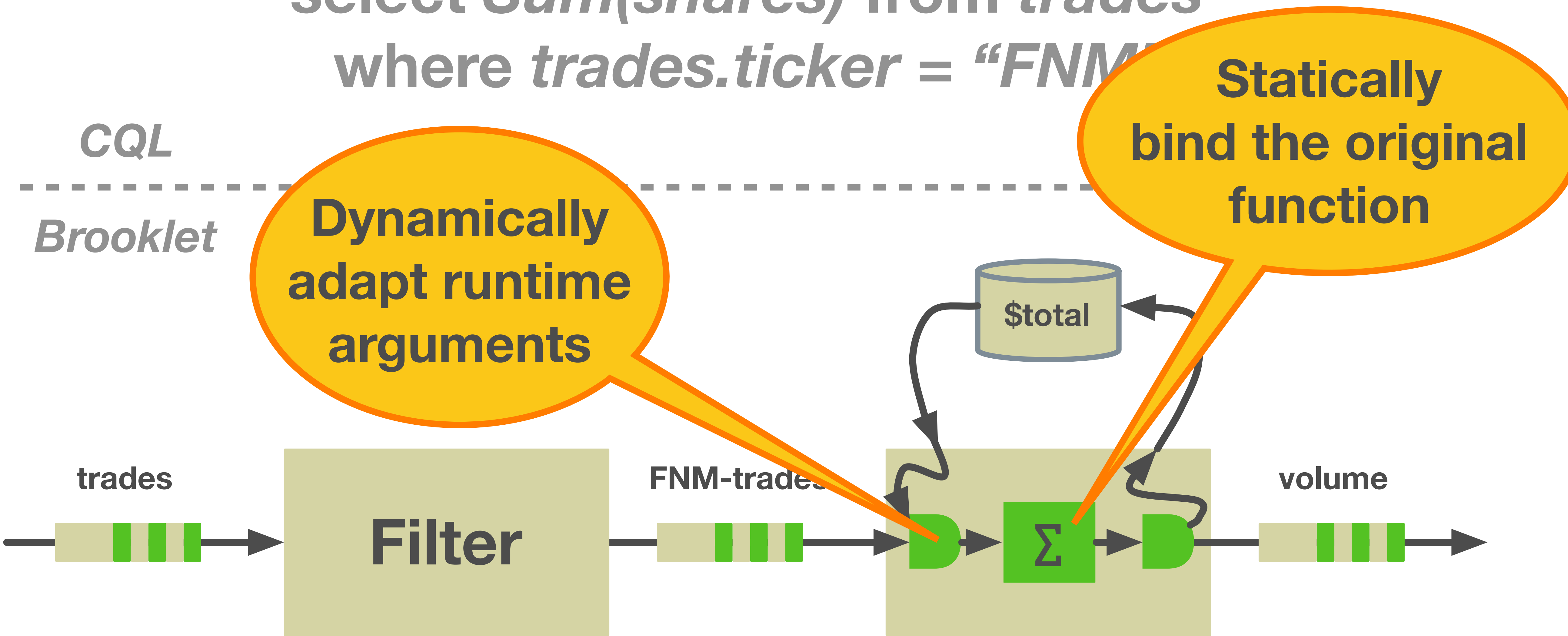
---

*Brooklet*

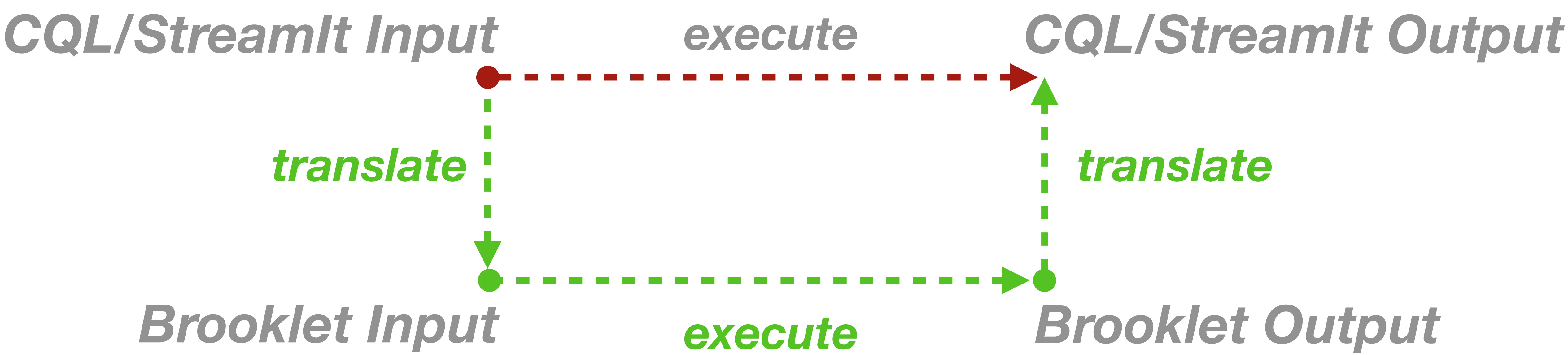


# Example: CQL to Brooklet

**select *Sum(shares)* from *trades***
**where *trades.ticker* = *“FNM*…**



# Translation Correctness Theorem

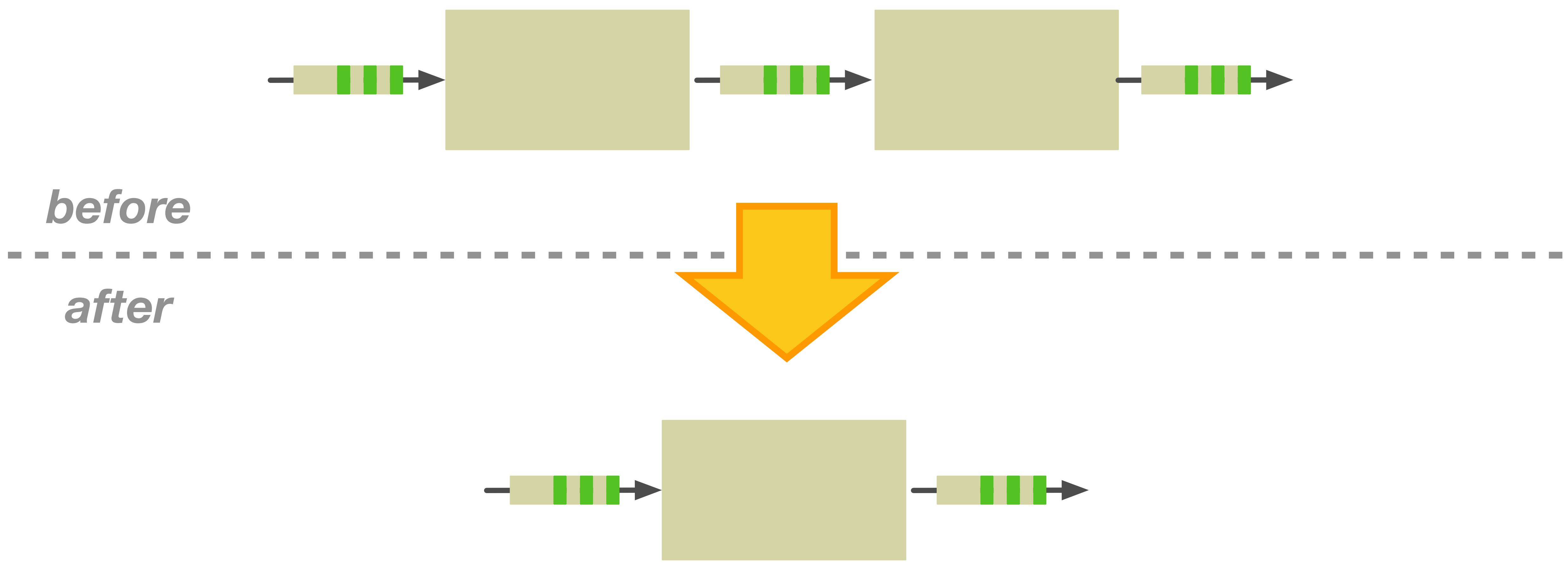



- Results under CQL and StreamIt semantics are the same as the results under Brooklet semantics after translation
- First formal semantics for Sawzall

# Optimizations

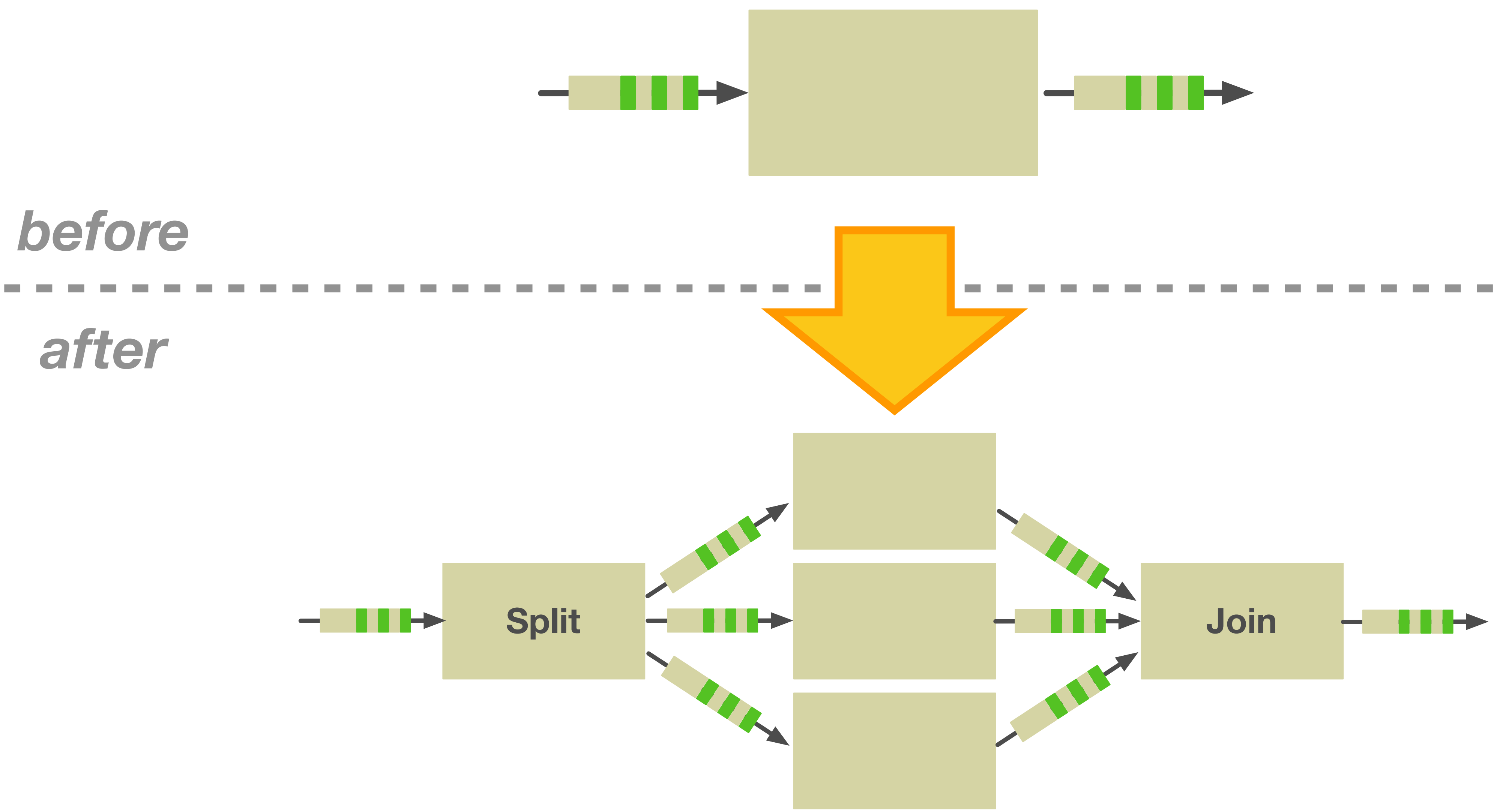
**Demonstrating Brooklet's utility by realizing three essential optimizations**

# Operator Fusion: Eliminate Queueing Delays



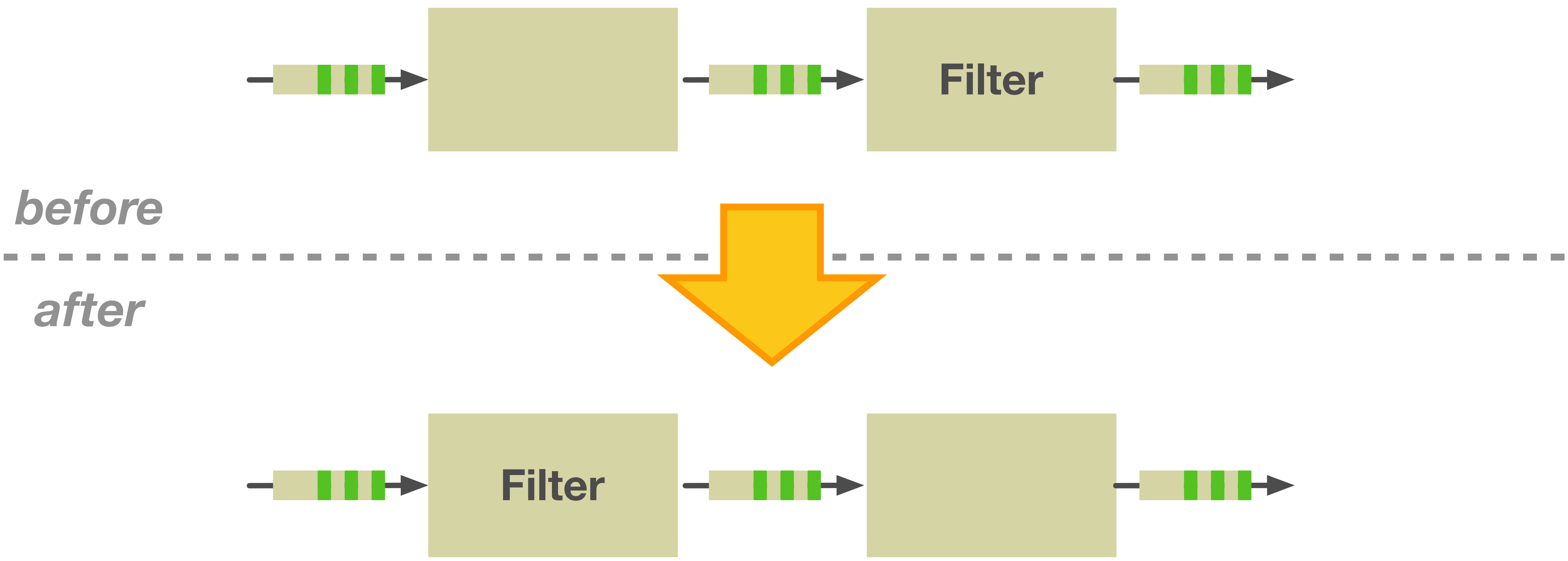

**Look for connected operators, whose state isn't used anywhere else**

# Operator Fission: Process More Data in Parallel




## Look for stateless operators

# Operator Reordering: Filter Data Early




Look for operators whose read/write sets don't overlap [Ghelli et al., SIGMOD 08]

# Outlook

# Outlook

- More optimizations
  - dynamic operator placement, load balancing, subquery sharing, eliminating spurious synchronization

# Outlook

- More optimizations
  - dynamic operator placement, load balancing, subquery sharing, eliminating spurious synchronization
- Richer extended calculus
  - Types, verify progress, time constraints

# Outlook

- More optimizations
  - dynamic operator placement, load balancing, subquery sharing, eliminating spurious synchronization
- Richer extended calculus
  - Types, verify progress, time constraints
- Common execution platform
  - Practical challenges: data types, library of operators, serialization, process management, error handling

# Conclusions

- Streaming is everywhere
  - Media, finance, web applications
- Need a calculus to understand (distributed) implementations
  - Minimal, non-deterministic, makes state and communication explicit
- Provide a formal and practical foundation for stream programming
  - Mappings from CQL, StreamIt, and Sawzall
  - Formalizing of Fusion, Fission, and Reordering

http://cs.nyu.edu/brooklet

# CQL Translation Rules

**CQL program translation:** $[\![ F_c, P_c ]\!]_c^p = \langle F_b, P_b \rangle$

$$[\![ F_c, SName ]\!]_c^p = \emptyset, \texttt{output}\,SName\texttt{;input}\,SName\texttt{;}\bullet \qquad (\mathrm{T}_c^p\text{-SName})$$

$$[\![ F_c, RName ]\!]_c^p = \emptyset, \texttt{output}\,RName\texttt{;input}\,RName\texttt{;}\bullet \qquad (\mathrm{T}_c^p\text{-RName})$$

$$\frac{\begin{array}{c} F_b, \texttt{output}\ q_o\texttt{;}\ \texttt{input}\ \overline{q}\texttt{;}\ \overline{op} = [\![ F_c, P_{cs} ]\!]_c^p \\ q_o' = freshId() \qquad v = freshId() \\ F_b' = [S2R \mapsto wrapS2R(F_c(S2R))]F_b \\ \overline{op}' = \overline{op}, (q_o', v) \leftarrow S2R(q_o, v)\texttt{;} \end{array}}{[\![ F_c, S2R(P_{cs}) ]\!]_c^p = F_b', \texttt{output}\ q_o'\texttt{;}\ \texttt{input}\ \overline{q}\texttt{;}\ \overline{op}'} \qquad (\mathrm{T}_c^p\text{-S2R})$$

$$\frac{\begin{array}{c} F_b, \texttt{output}\ q_o\texttt{;}\ \texttt{input}\ \overline{q}\texttt{;}\ \overline{op} = [\![ F_c, P_{cr} ]\!]_c^p \\ q_o' = freshId() \qquad v = freshId() \\ F_b' = [R2S \mapsto wrapR2S(F_c(R2S))]F_b \\ \overline{op}' = \overline{op}, (q_o', v) \leftarrow R2S(q_o, v)\texttt{;} \end{array}}{[\![ F_c, R2S(P_{cr}) ]\!]_c^p = F_b', \texttt{output}\ q_o'\texttt{;}\ \texttt{input}\ \overline{q}\texttt{;}\ \overline{op}'} \qquad (\mathrm{T}_c^p\text{-R2S})$$

$$\frac{\begin{array}{c} \overline{F_b, \texttt{output}\ q_o\texttt{;}\ \texttt{input}\ \overline{q}\texttt{;}\ \overline{op}} = \overline{[\![ F_c, P_{cr} ]\!]_c^p} \\ n = |\overline{P_{cr}}| \qquad q_o' = freshId() \qquad \overline{q}' = \overline{q}_1, \ldots, \overline{q}_n \\ \forall i \in 1 \ldots n : v_i = freshId() \qquad \overline{op}' = \overline{op}_1, \ldots, \overline{op}_n \\ F_b' = [R2R \mapsto wrapR2R(F_c(R2R))](\cup \overline{F_b}) \\ \overline{op}'' = \overline{op}', (q_o', \overline{v}) \leftarrow R2R(\overline{q_o}, \overline{v})\texttt{;} \end{array}}{[\![ F_c, R2R(\overline{P_{cr}}) ]\!]_c^p = F_b', \texttt{output}\ q_o'\texttt{;input}\ \overline{q}'\texttt{;}\overline{op}''} \qquad (\mathrm{T}_c^p\text{-R2R})$$

**CQL operator wrappers:**

$$\frac{\begin{array}{c} \sigma, \tau = d_q \qquad s = d_v \\ s' = s \cup \{\langle e, \tau \rangle : e \in \sigma\} \qquad \sigma' = f(s', \tau) \end{array}}{wrapS2R(f)(d_q, \_, d_v) = \langle \sigma', \tau \rangle, s'} \qquad (\mathrm{W}_c\text{-S2R})$$

$$\frac{\sigma, \tau = d_q \qquad \sigma' = d_v \qquad \sigma'' = f(\sigma, \sigma')}{wrapR2S(f)(d_q, \_, d_v) = \langle \sigma'', \tau \rangle, \sigma} \qquad (\mathrm{W}_c\text{-R2S})$$

$$\frac{\begin{array}{c} \sigma, \tau = d_q \qquad d_i' = d_i \cup \{\langle \sigma, \tau \rangle\} \\ \forall j \neq i \in 1 \ldots n : d_j' = d_j \\ \exists j \in 1 \ldots n : \nexists \sigma : \langle \sigma, \tau \rangle \in d_j \end{array}}{wrapR2R(f)(d_q, i, \overline{d}) = \bullet, \overline{d}'} \qquad (\mathrm{W}_c\text{-R2R-Wait})$$

$$\frac{\begin{array}{c} \sigma, \tau = d_q \qquad d_i' = d_i \cup \{\langle \sigma, \tau \rangle\} \\ \forall j \neq i \in 1 \ldots n : d_j' = d_j \\ \forall j \in 1 \ldots n : \sigma_j = aux(d_j, \tau) \end{array}}{wrapR2R(f)(d_q, i, \overline{d}) = \langle f(\overline{\sigma}), \tau \rangle, \overline{d}'} \qquad (\mathrm{W}_c\text{-R2R-Ready})$$

$$\frac{\langle \sigma, \tau \rangle \in d}{aux(d, \tau) = \sigma} \qquad (\mathrm{W}_c\text{-R2R-Aux})$$

# Operator Fission

$$\frac{\begin{array}{c} op = (q_{out}) \leftarrow f(q_{in}); \\ \forall i \in 1 \ldots n : q_i = \mathit{freshId}() \qquad \forall i \in 1 \ldots n : q'_i = \mathit{freshId}() \\ F'_b, op_s = [\![\, \emptyset, \texttt{split roundrobin}, \overline{q}, q_{in} \,]\!]^p_s \\ \forall i \in 1 \ldots n : op_i = (q'_i) \leftarrow f(q_i); \\ F''_b, op_j = [\![\, \emptyset, \texttt{join roundrobin}, q_{out}, \overline{q}' \,]\!]^p_s \end{array}}{\langle F_b, op \rangle \longrightarrow^N_{split} \langle F_b \cup F'_b \cup F''_b, op_s\ \overline{op}\ op_j \rangle}$$