

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



Fannie Mae (Public, NYSE:FNM) Watch this stock

1.06
-0.02 (-1.85%)

Range	1.06 - 1.08	P/E	-
52 week	0.35 - 2.13	Div/yield	-
Open	1.08	EPS	-14.63
Vol / Avg.	6.75M/37.17M	Shares	1.11B
Mkt cap	1.19B	Beta	2.99

Real-time: 11:26AM EST
 NYSE real-time data - Disclaimer

Compare: Dow S&P 500 FRE [more »](#)

Zoom: [1d](#) [5d](#) [1m](#) [3m](#) [6m](#) [YTD](#) [1y](#) [5y](#) [10y](#) [Max](#)

Jan 22, 2009 - Jan 21, 2010 +0.33 (44.59%)

Government Bailout

Move large amounts of data through several computational steps



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

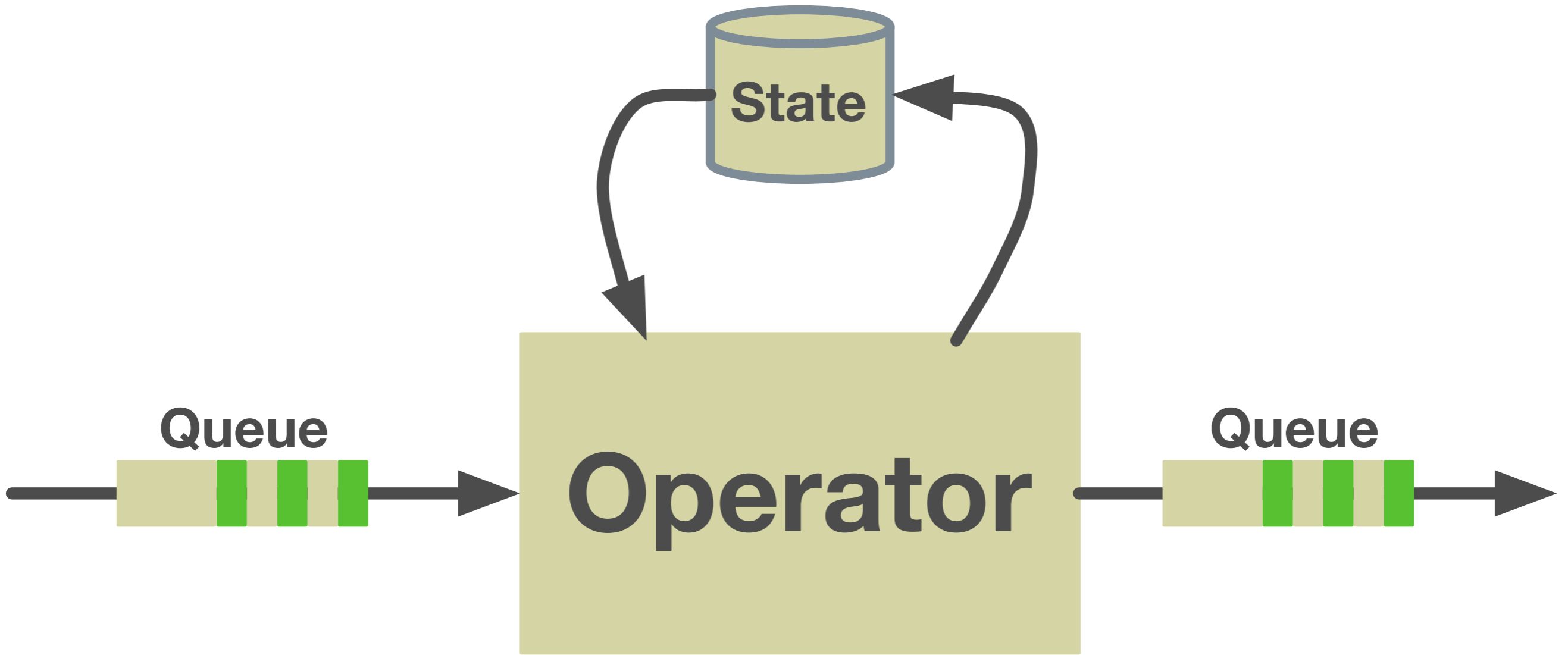
 Generality: Translating StreamIt, CQL, and Sawzall to Brooklet

 Utility: Optimizing Brooklet to Brooklet

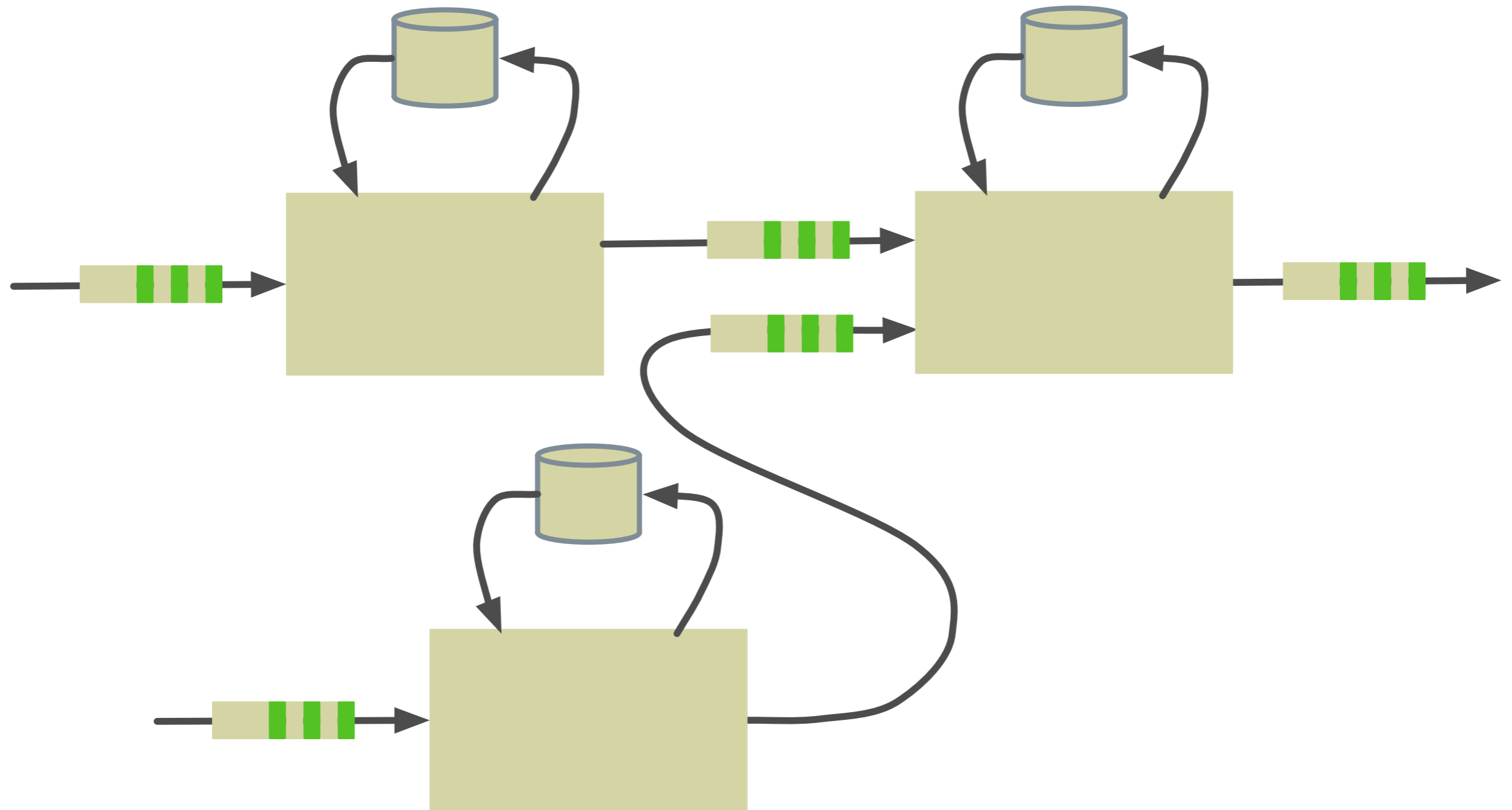
 Outlook and Conclusions



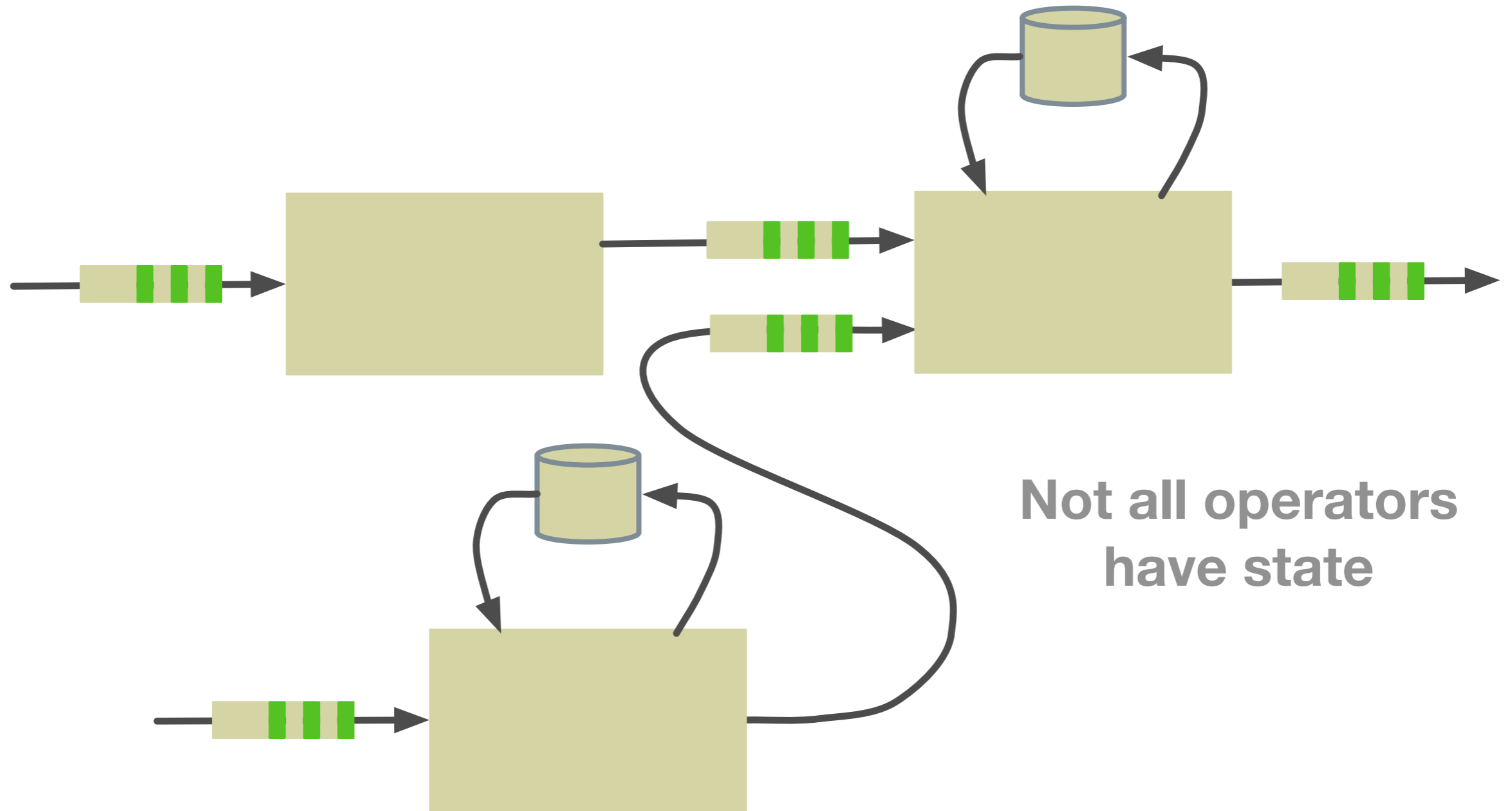
Elements of a Streaming App



Elements of a Streaming App

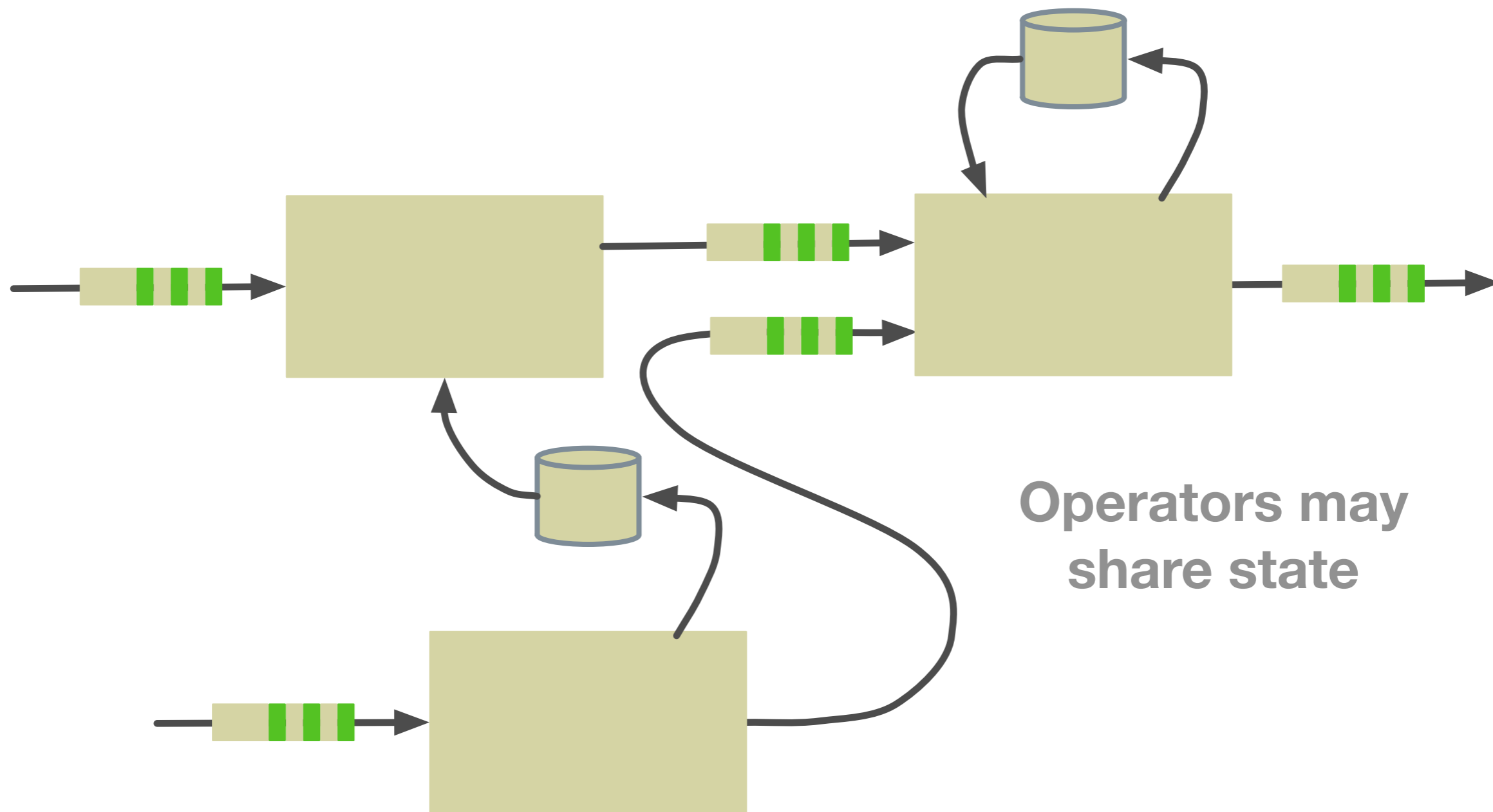


Elements of a Streaming App



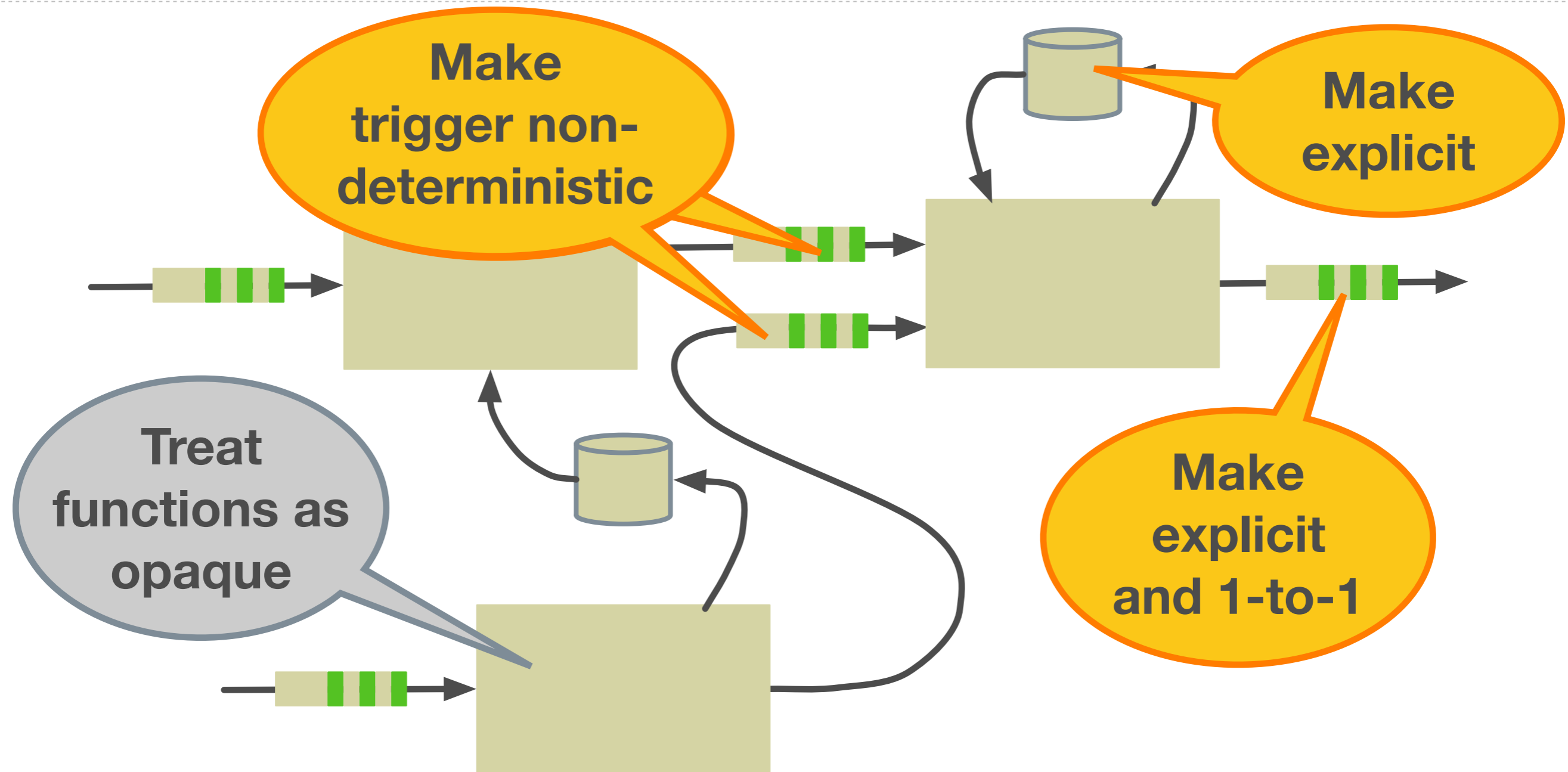
Not all operators have state

Elements of a Streaming App

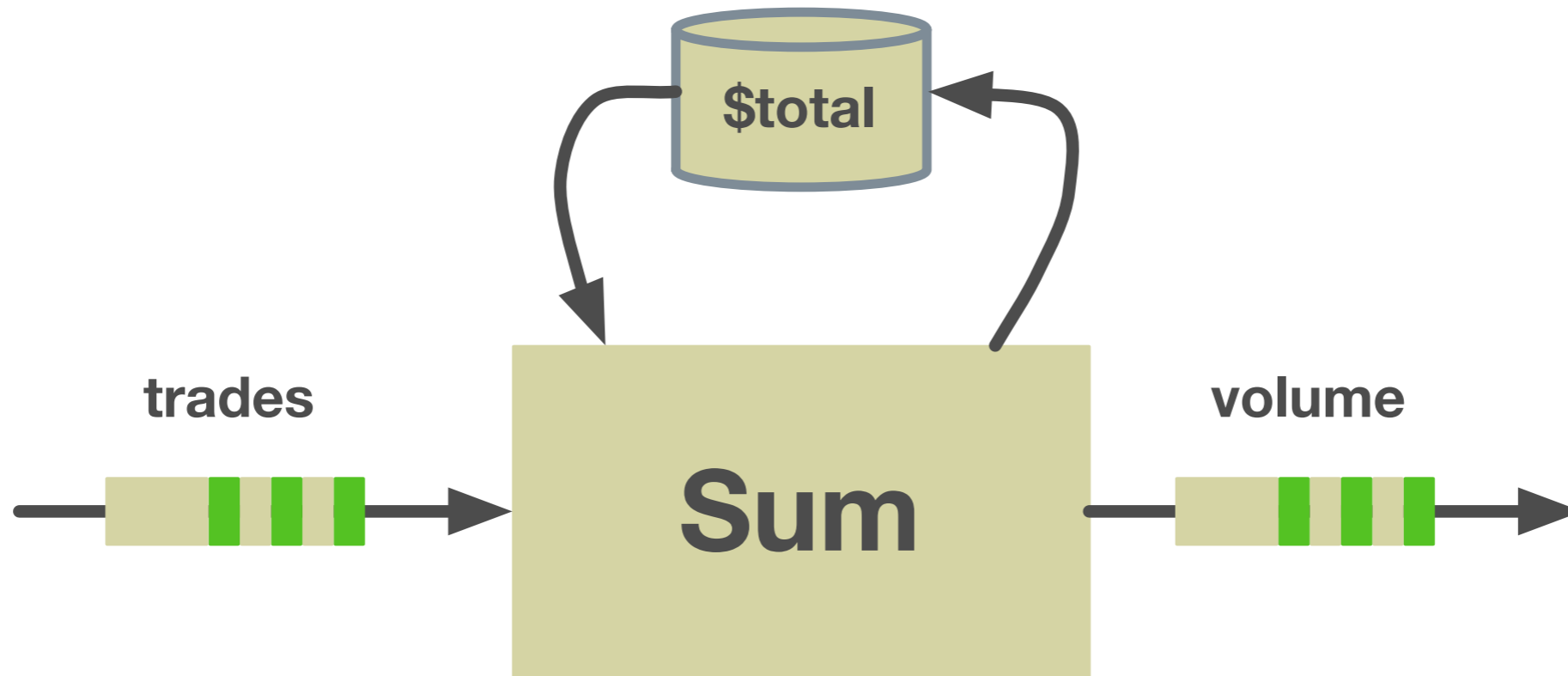


Operators may share state

Requirements for Calculus

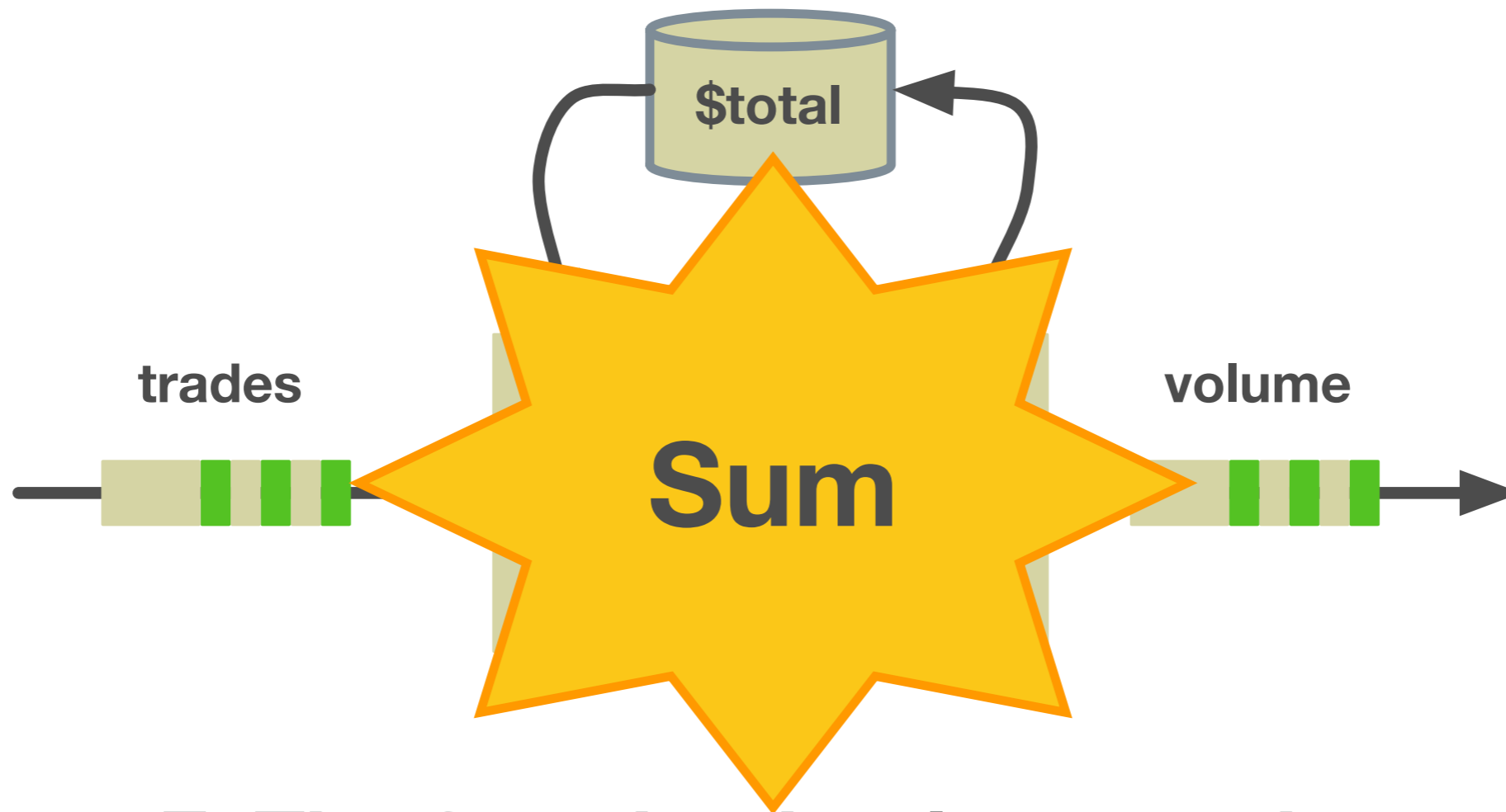


Brooklet Syntax



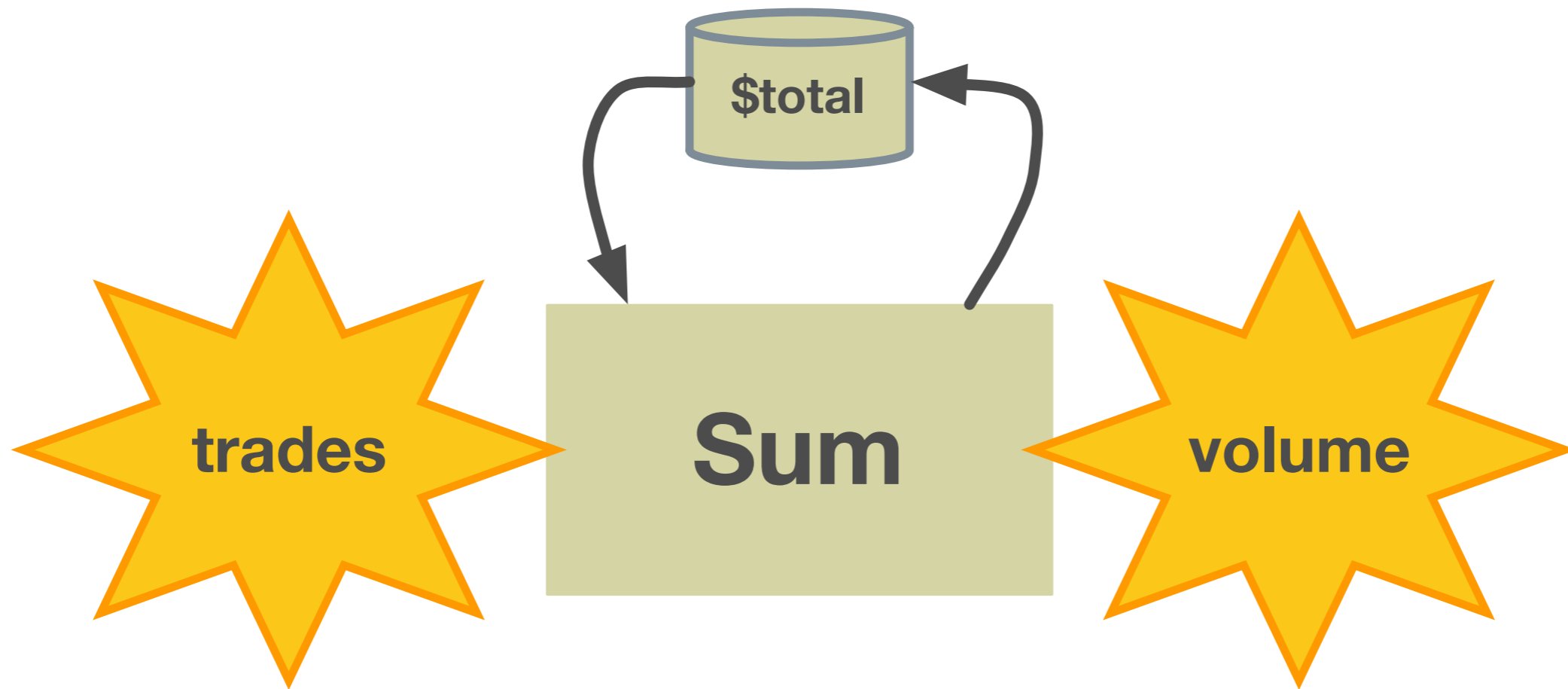
(volume, \$total) ← Sum(trades, \$total)

Function Environment



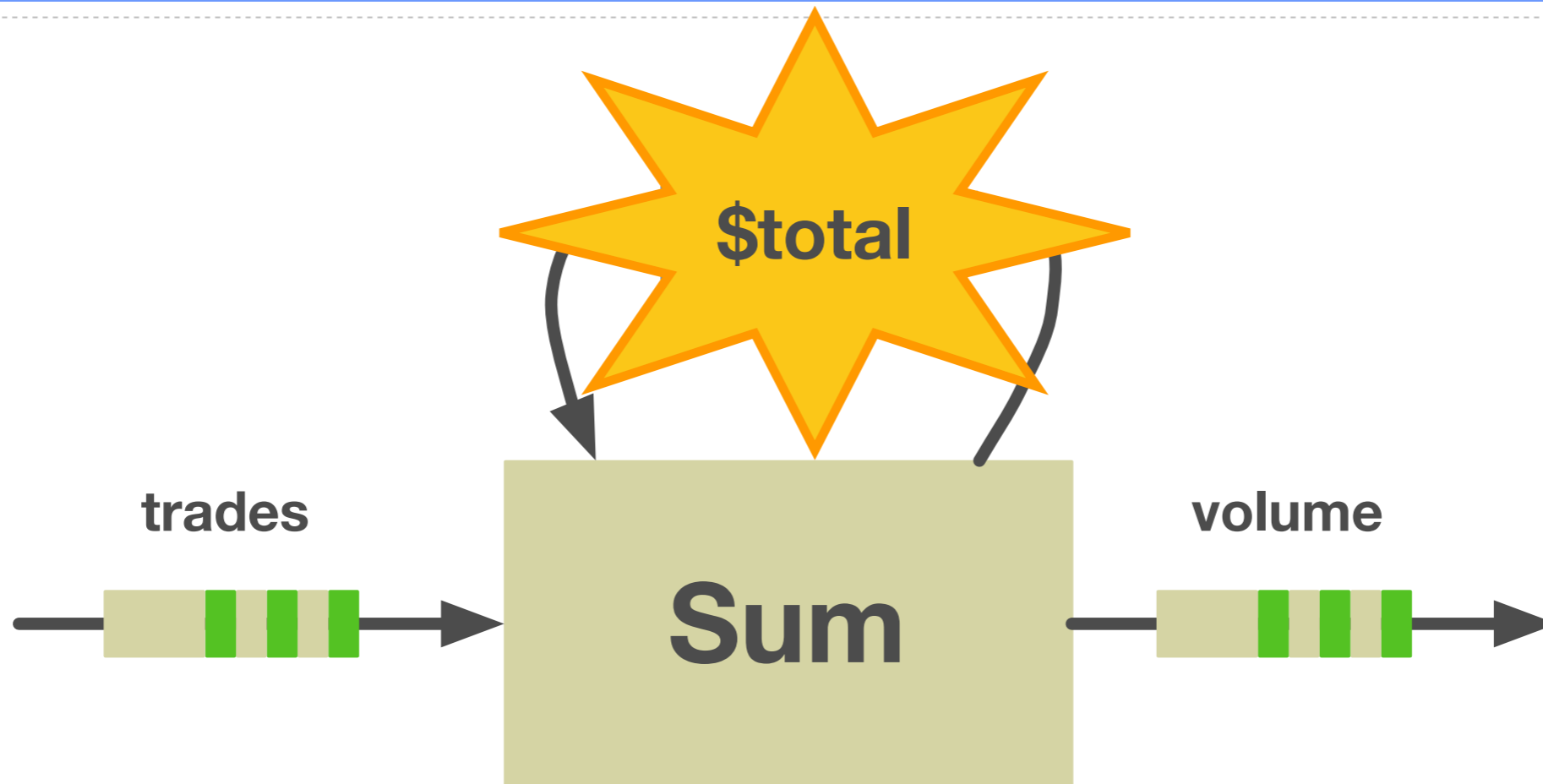
F: The function implementations

Queue Store



Q: The contents of the queues

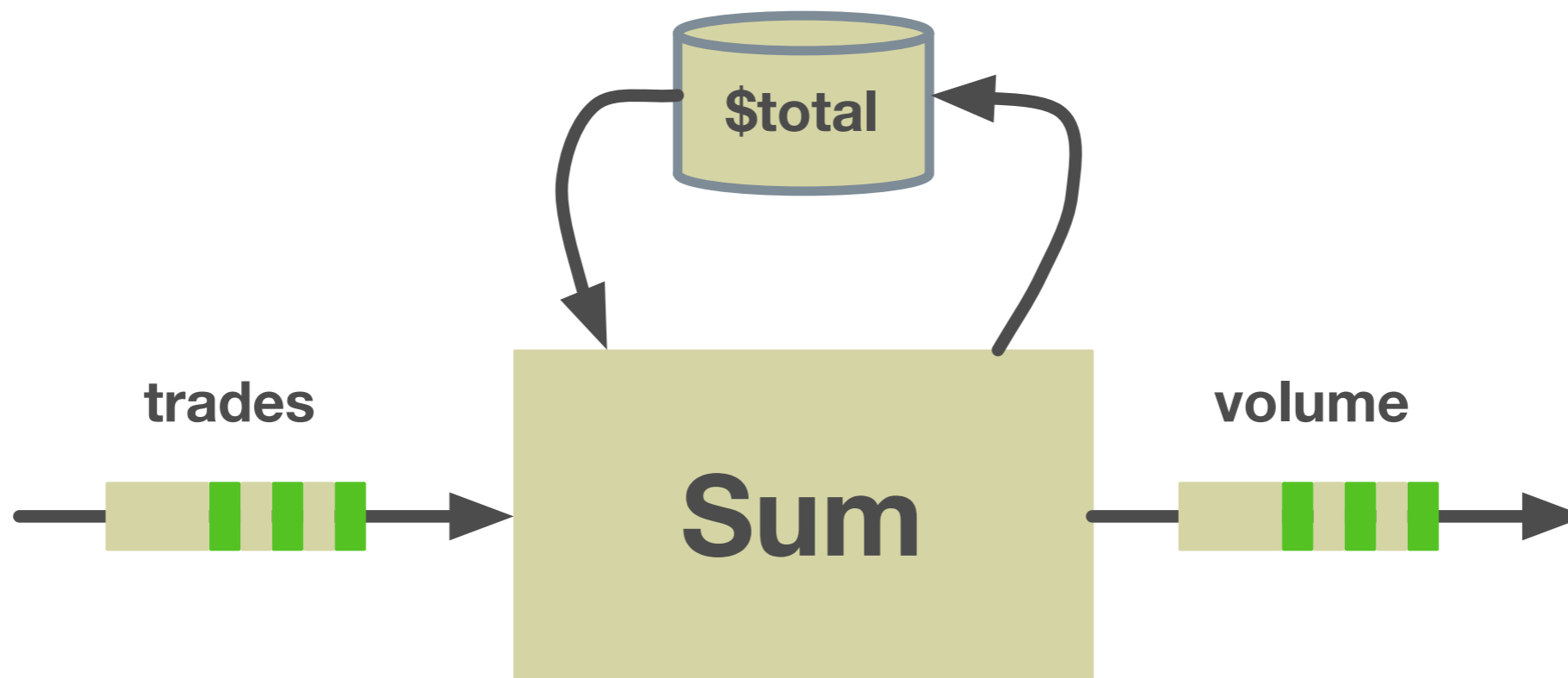
Variable Store



V: The contents of the variables

Brooklet

Operational Semantics



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

Complete Calculus

Brooklet syntax:

$P_b ::= out\ in\ \bar{op}$	Brooklet program
$out ::= output\ \bar{q};$	Output declaration
$in ::= input\ \bar{q};$	Input declaration
$op ::= (\bar{q}, \bar{v}) \leftarrow f(\bar{q}, \bar{v});$	Operator
$q ::= id$	Queue identifier
$v ::= \$id$	Variable identifier
$f ::= id$	Function identifier

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);

```

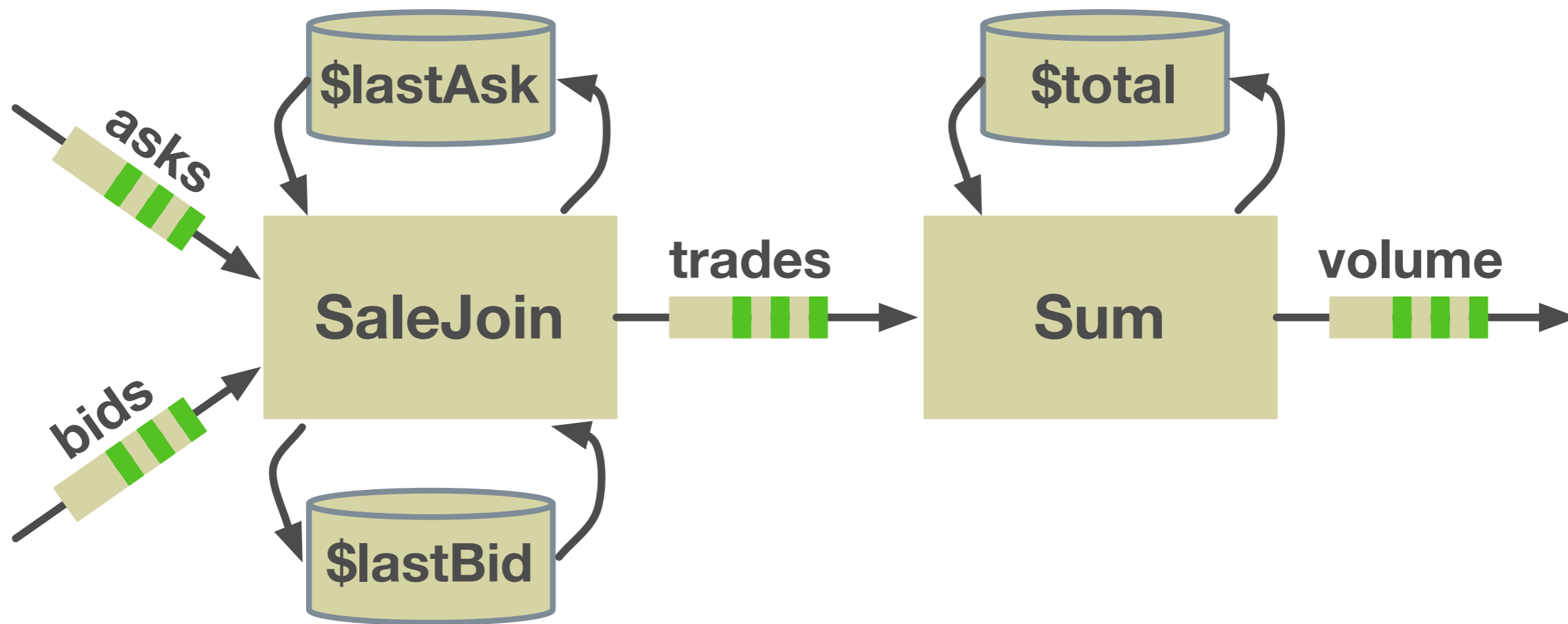
Brooklet semantics: $F_b \vdash \langle V, Q \rangle \longrightarrow \langle V', Q' \rangle$

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

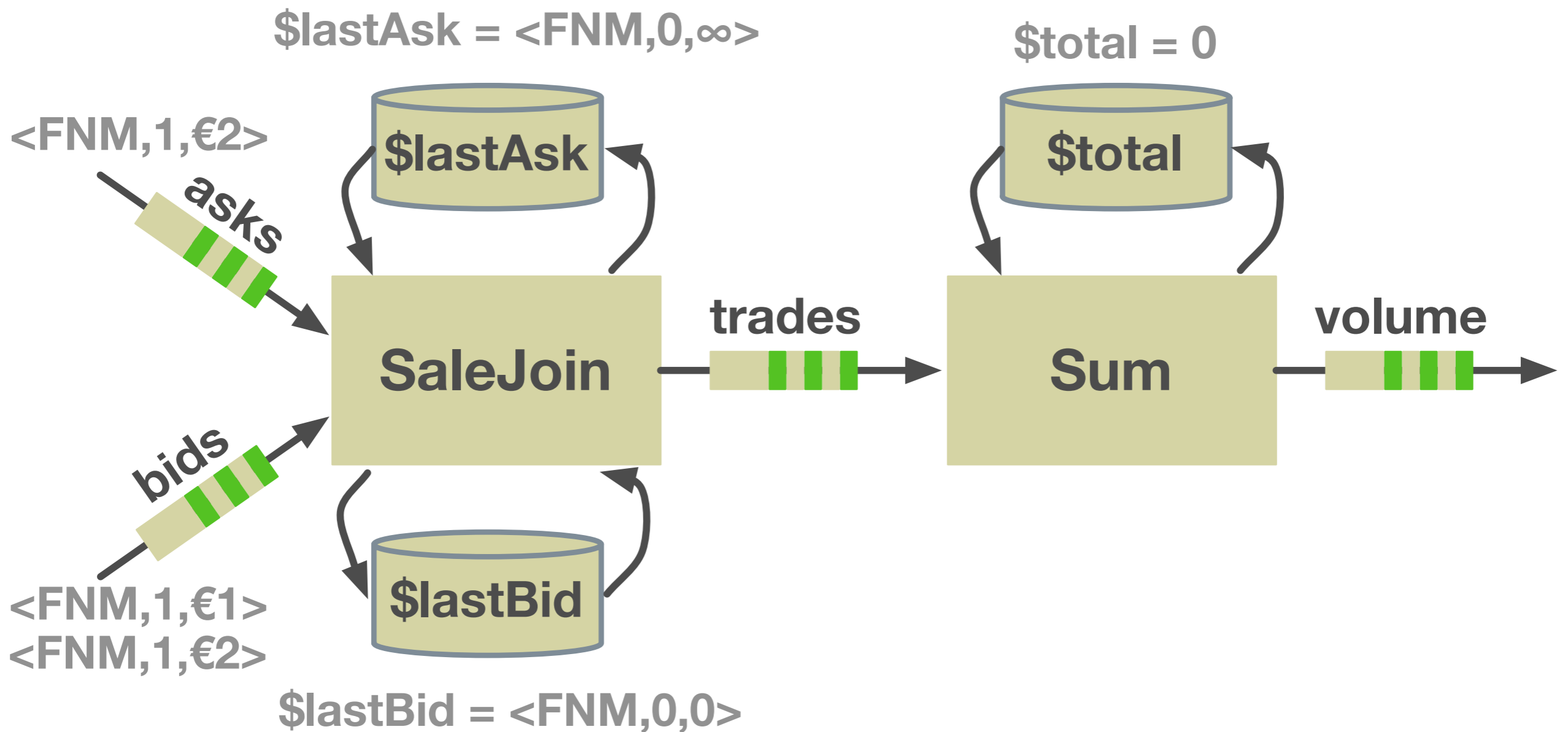
$$\begin{array}{c}
 op = (_, \bar{v}) \leftarrow f(_, _); \\
 \hline
 updateV(op, V, \bar{d}) = [\bar{v} \mapsto \bar{d}]V \quad (\text{E-UPDATEV})
 \end{array}$$

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

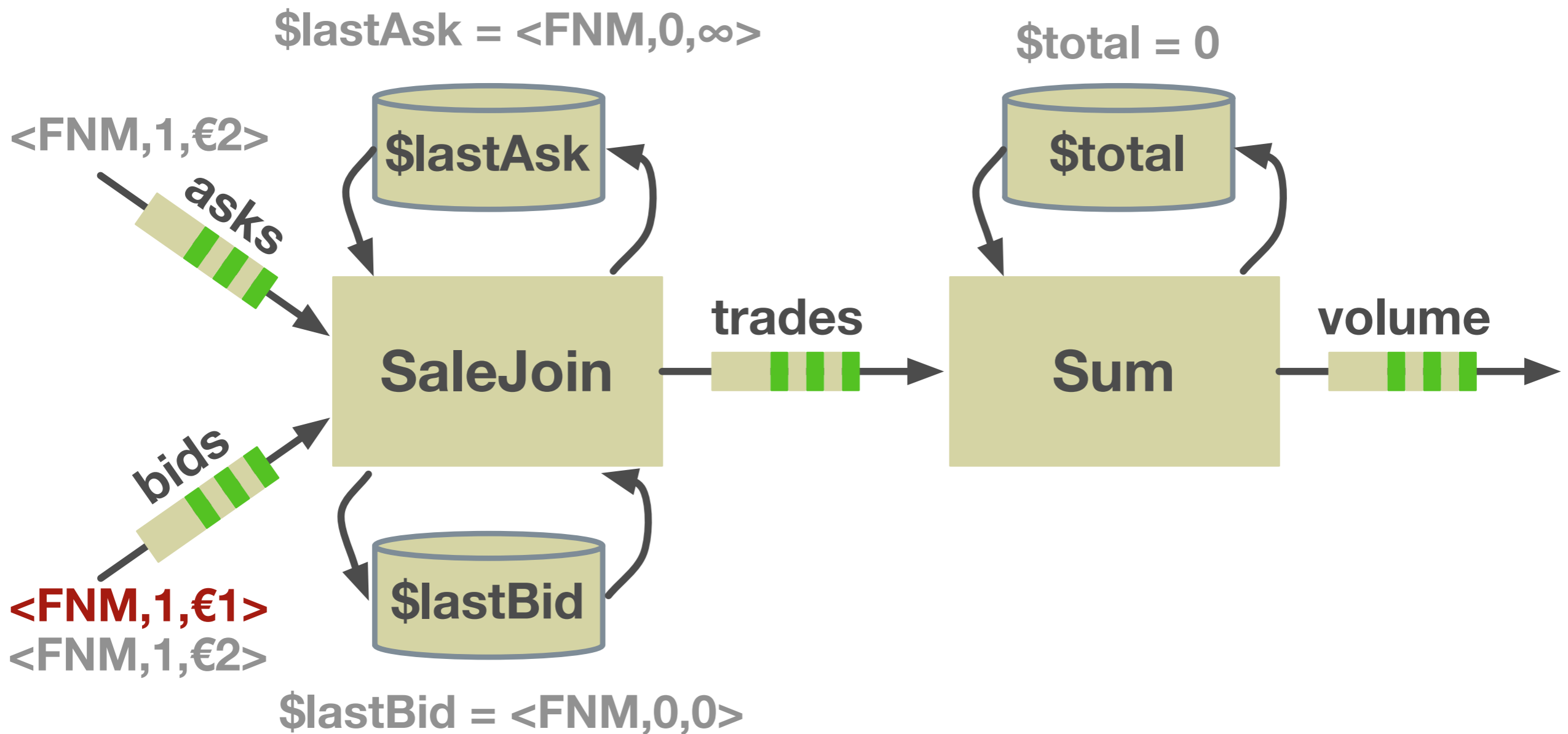
Example: A Fannie Mae Bid/Ask Join



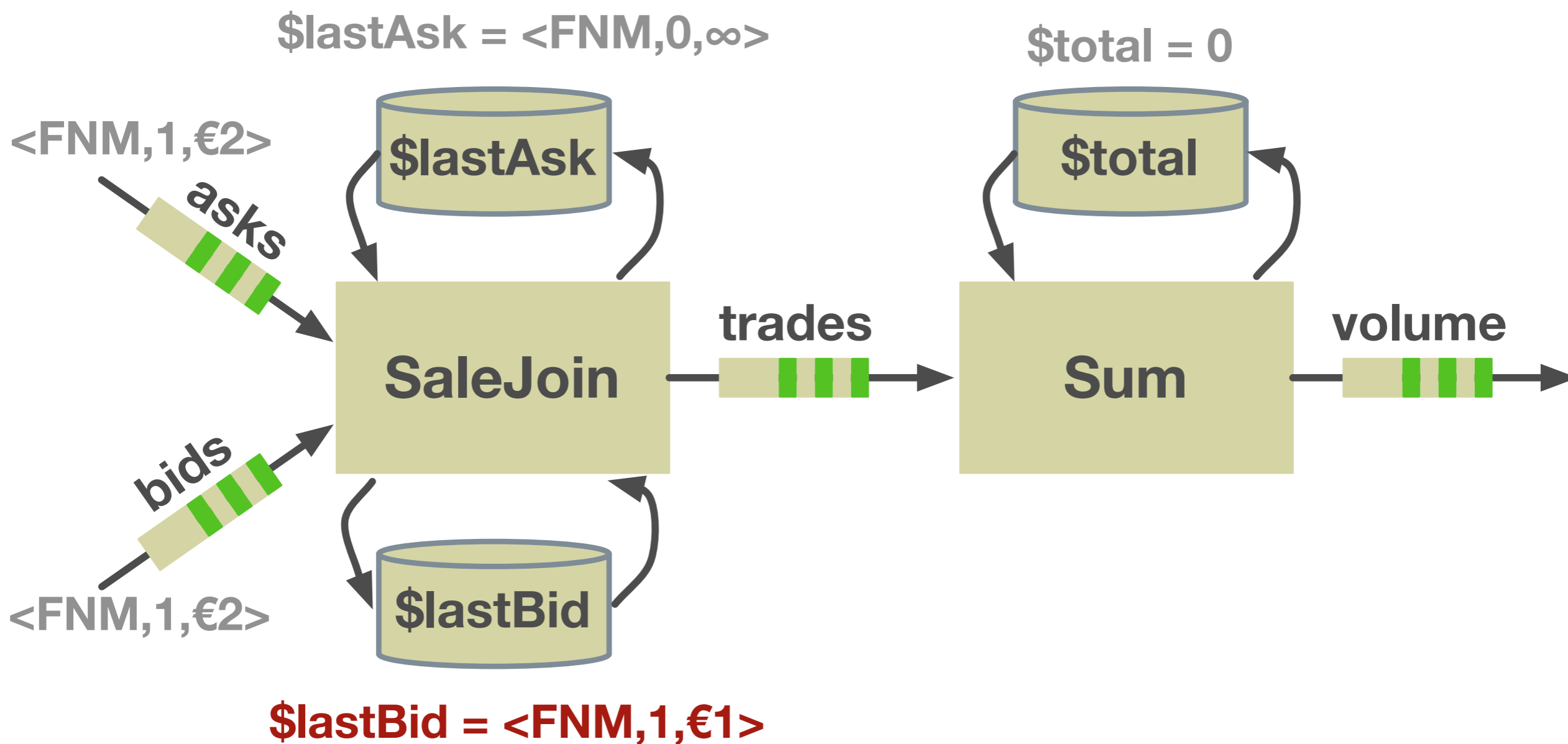
Example: A Fannie Mae Bid/Ask Join



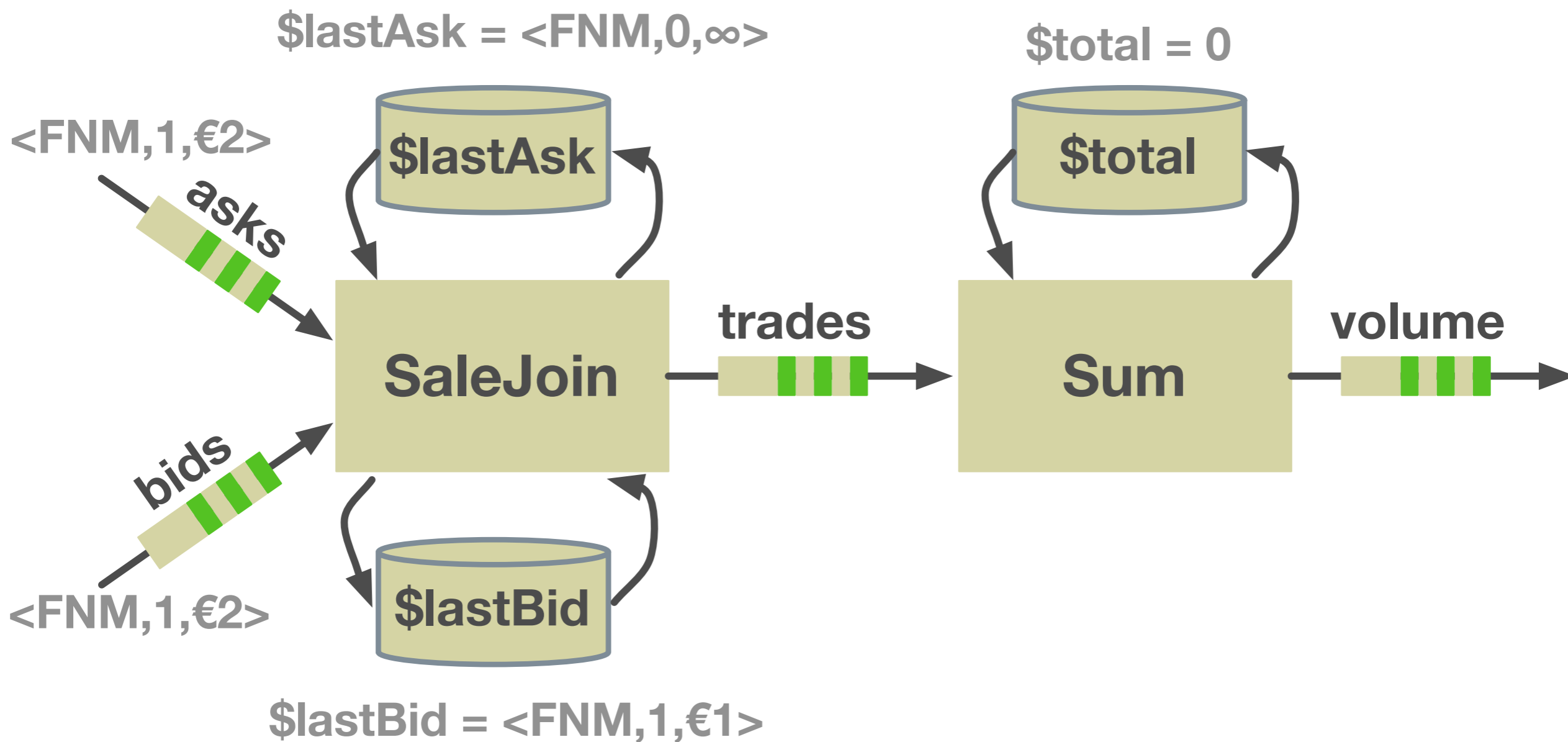
Example: A Fannie Mae Bid/Ask Join



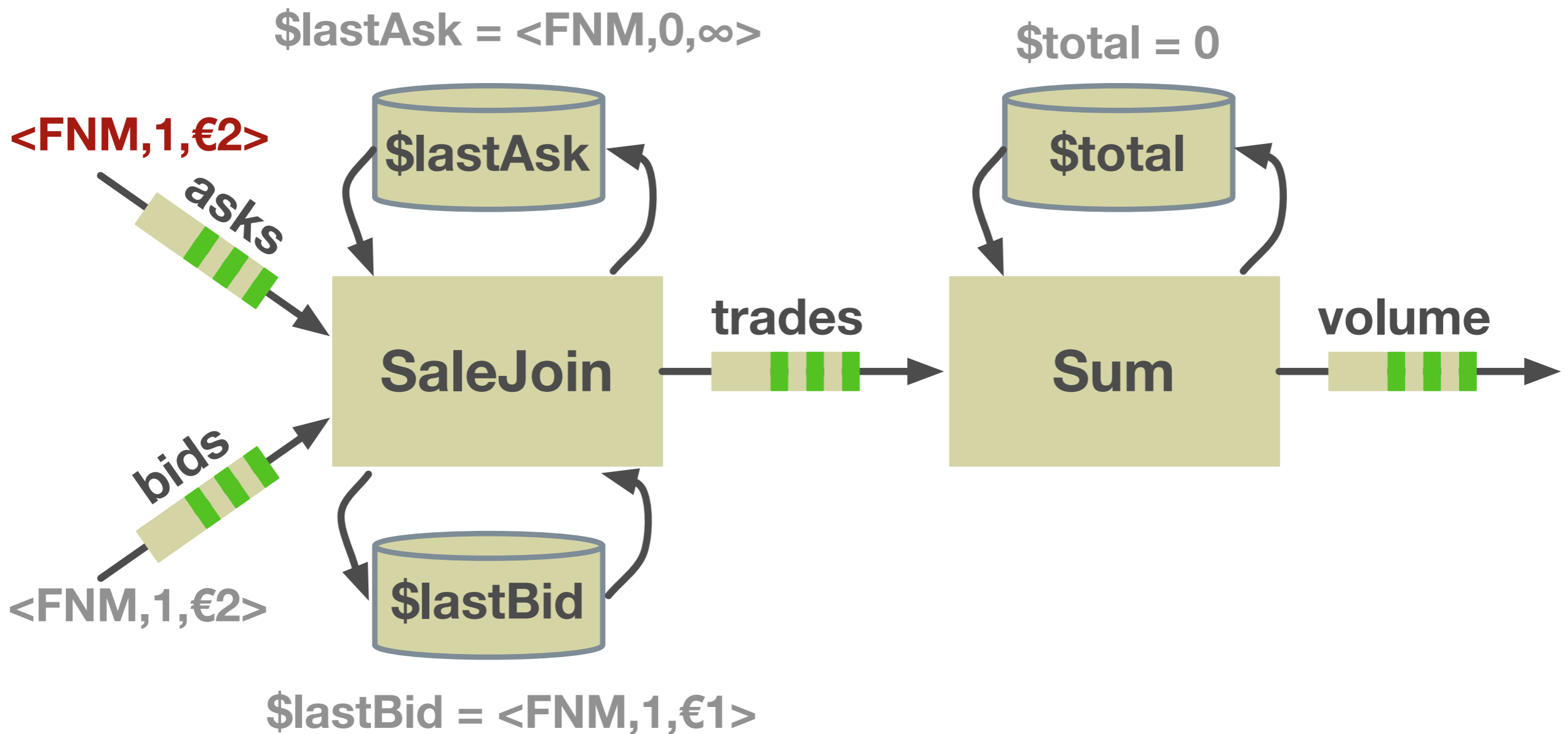
Example: A Fannie Mae Bid/Ask Join



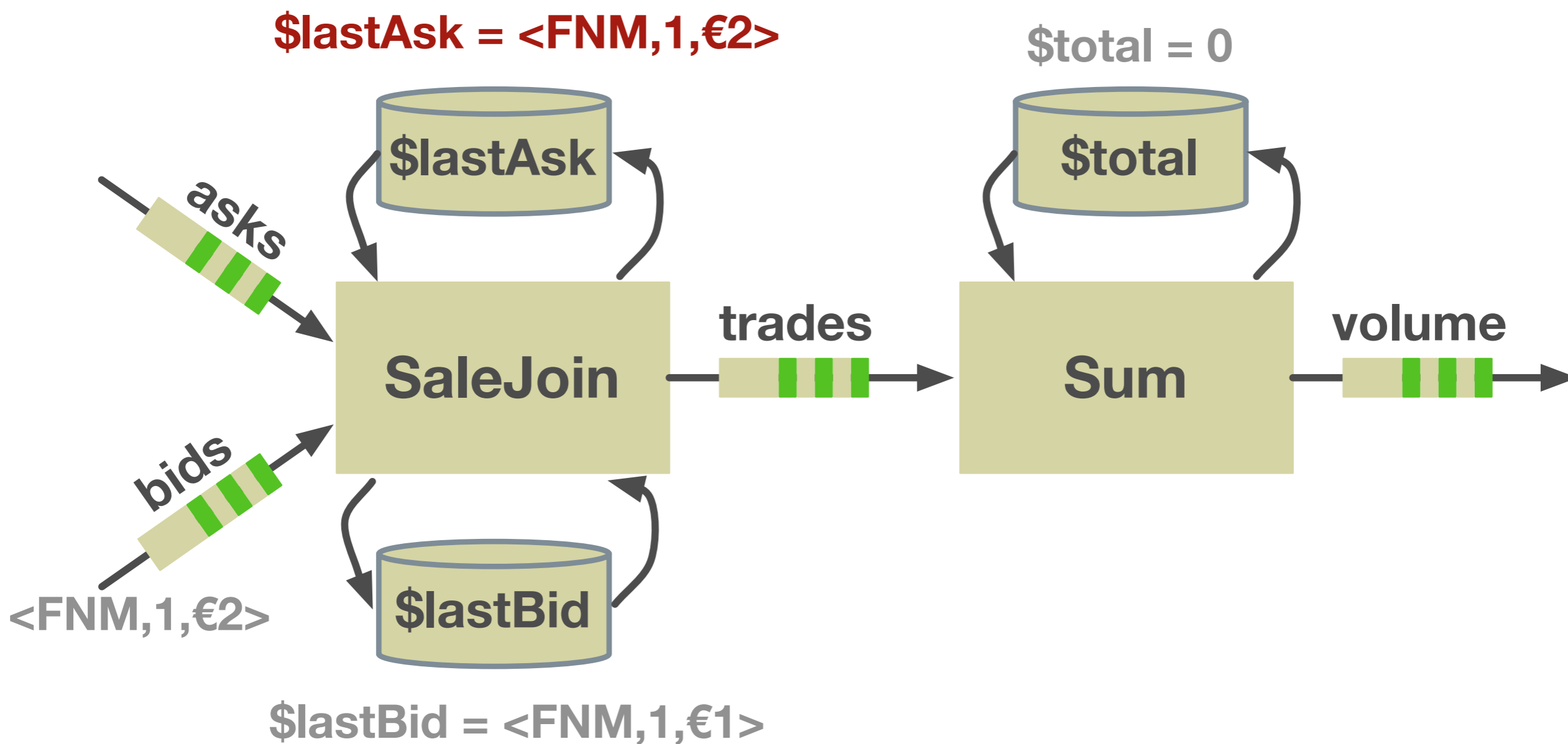
Example: A Fannie Mae Bid/Ask Join



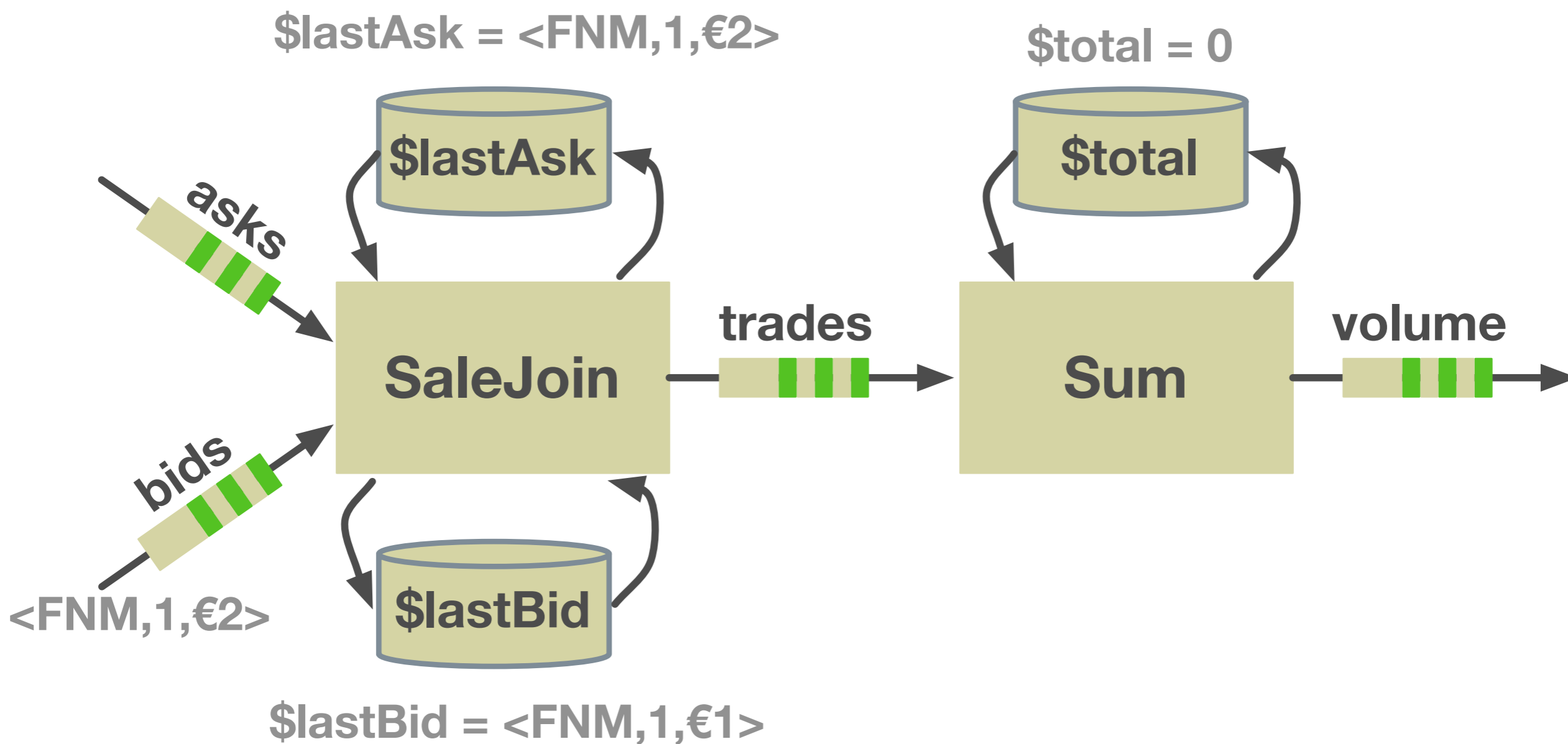
Example: A Fannie Mae Bid/Ask Join



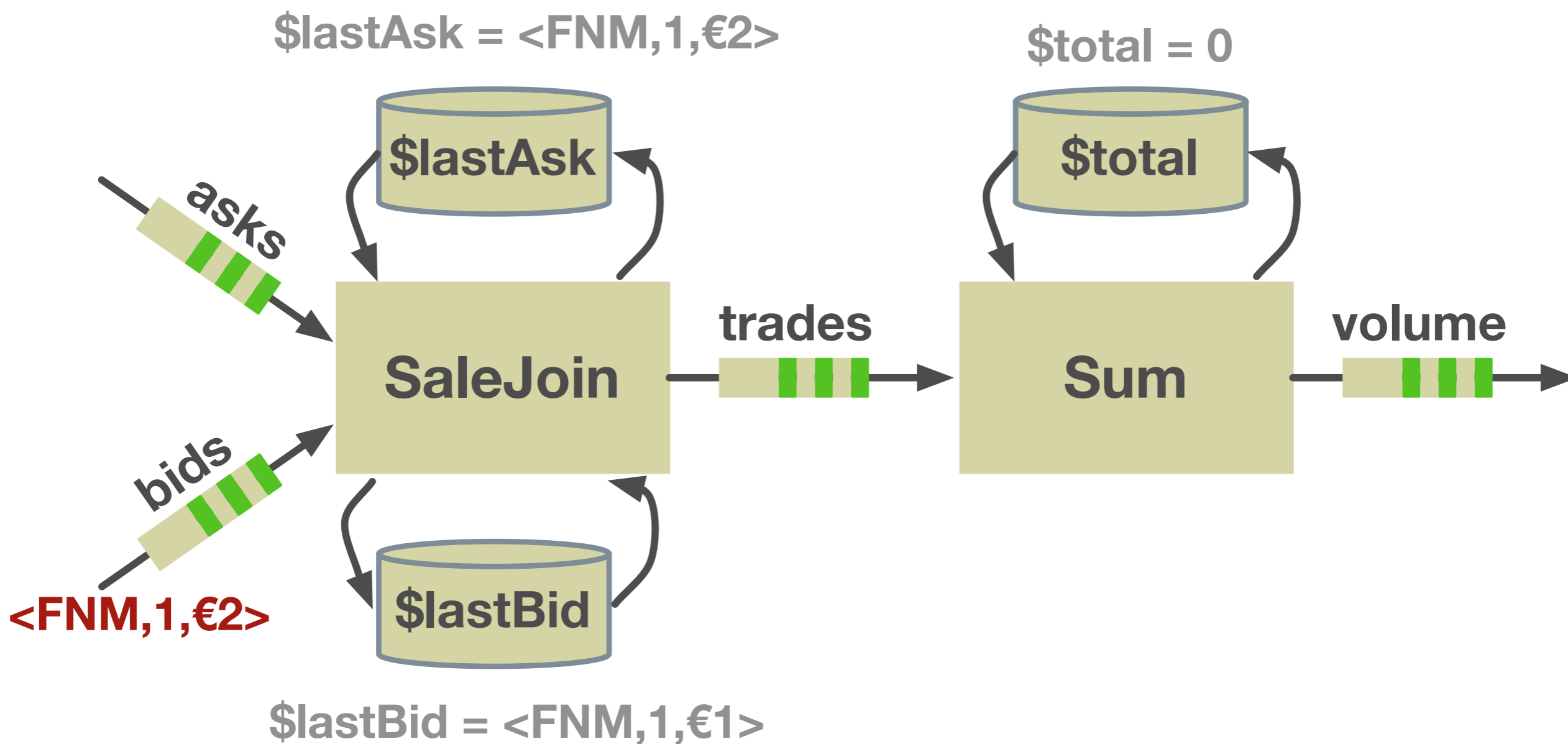
Example: A Fannie Mae Bid/Ask Join



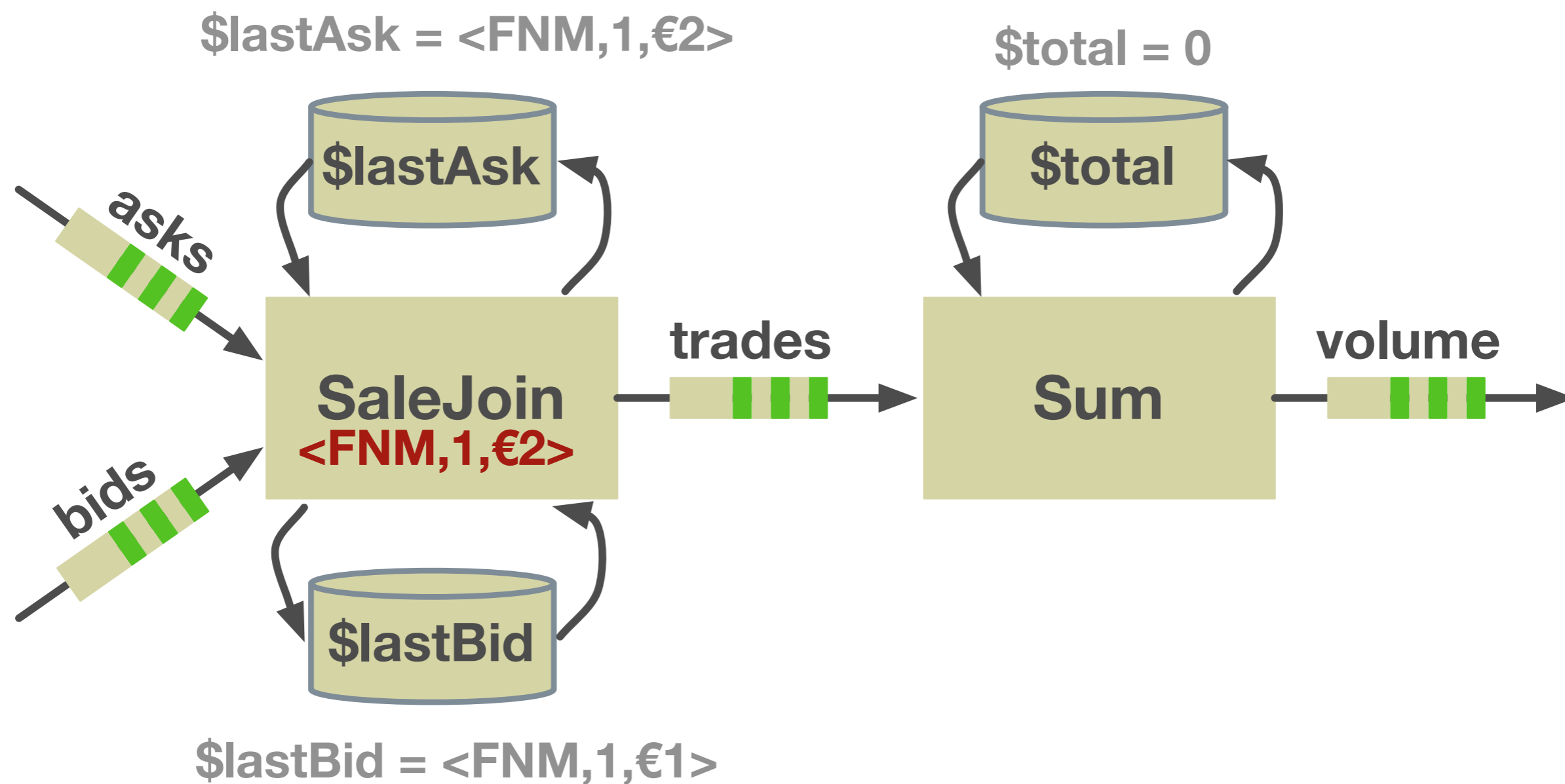
Example: A Fannie Mae Bid/Ask Join



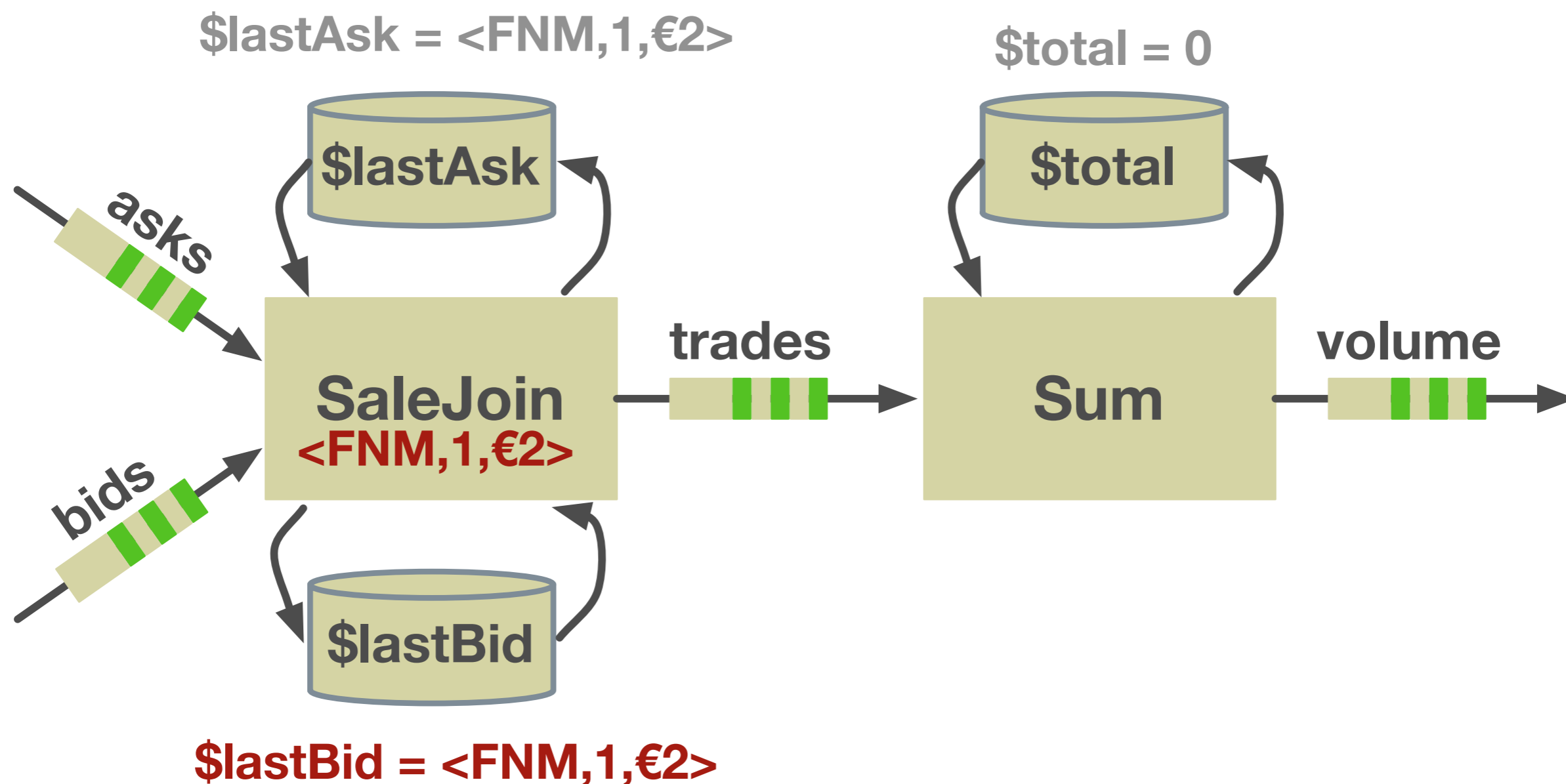
Example: A Fannie Mae Bid/Ask Join



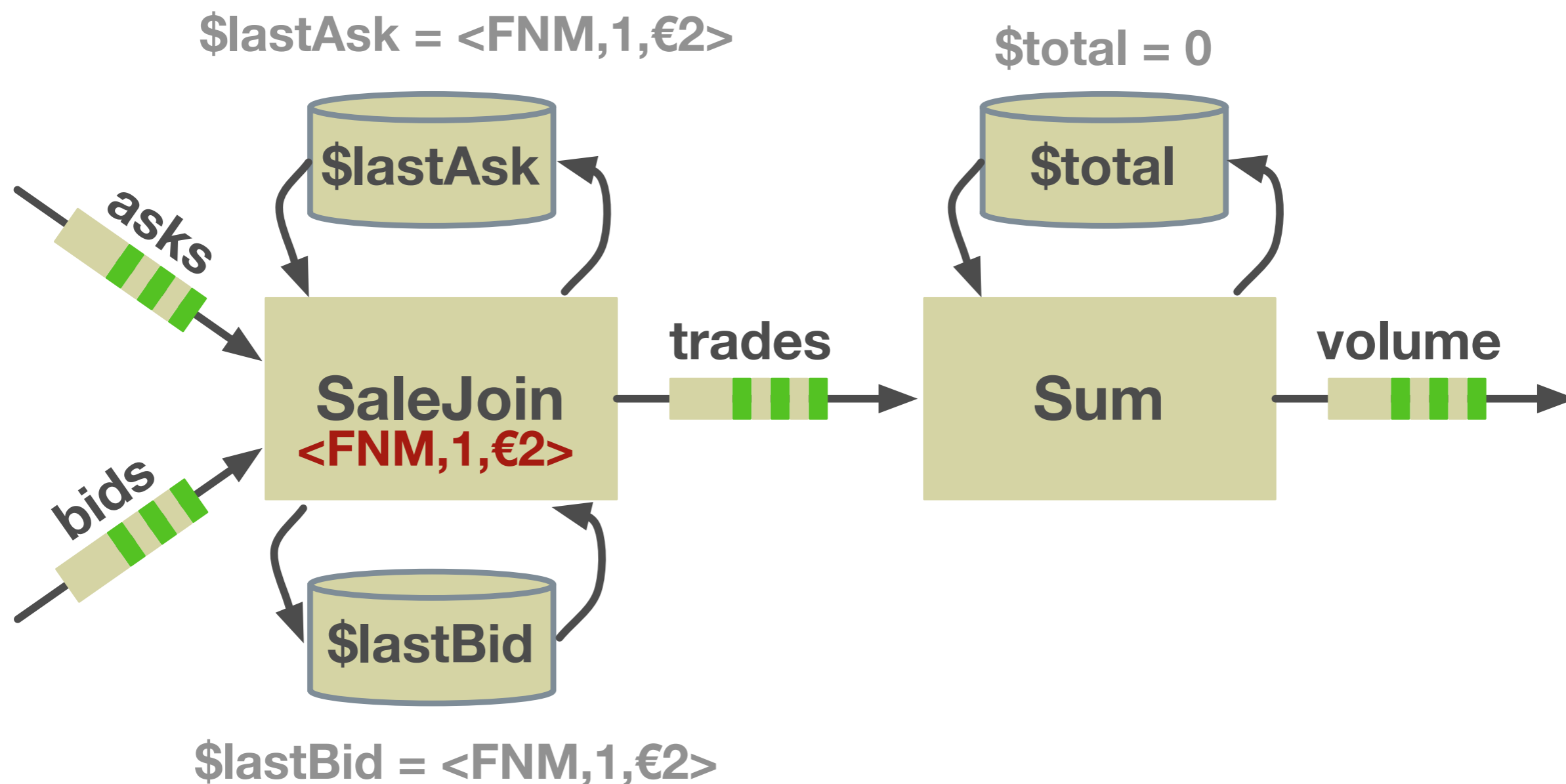
Example: A Fannie Mae Bid/Ask Join



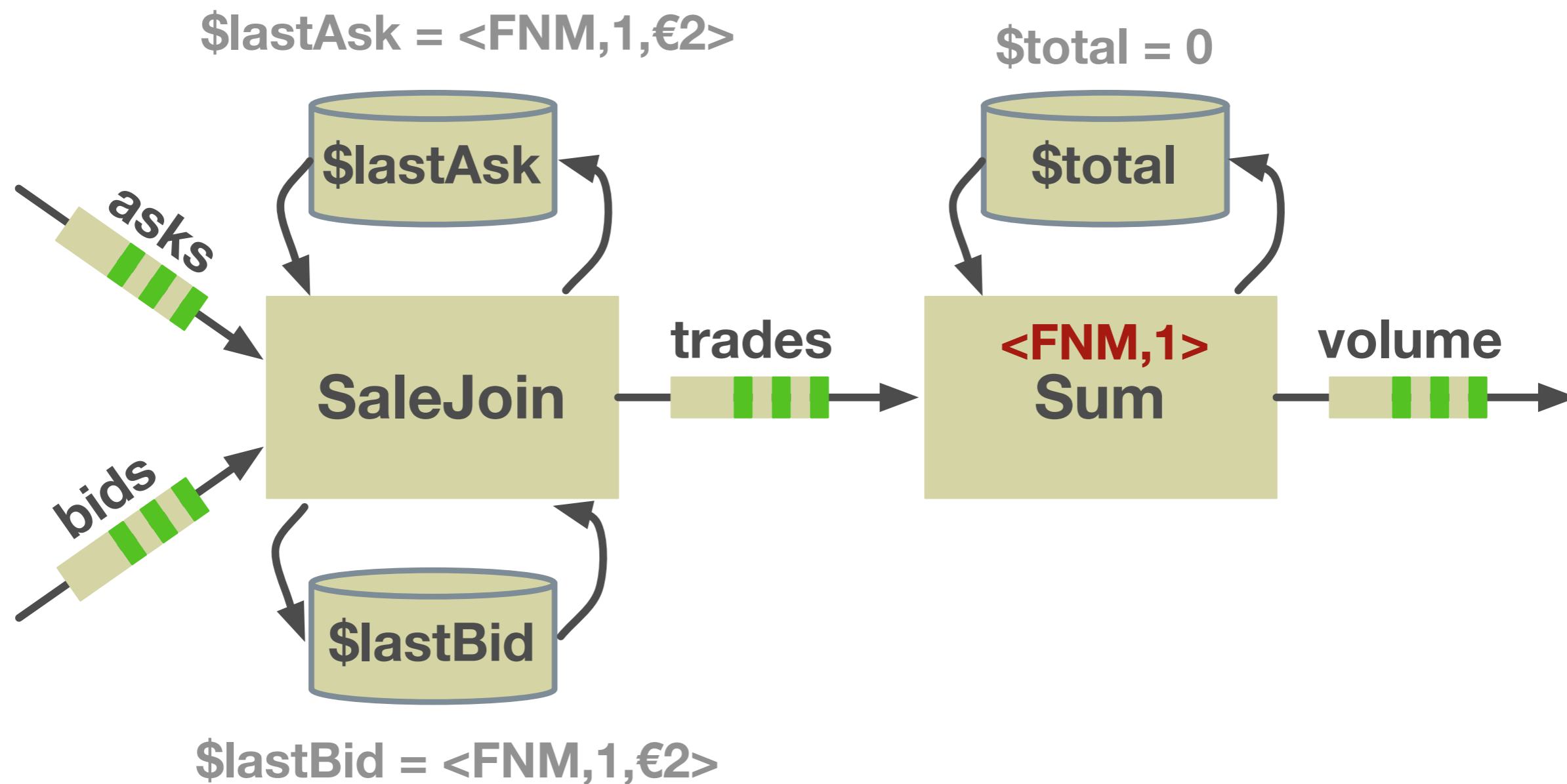
Example: A Fannie Mae Bid/Ask Join



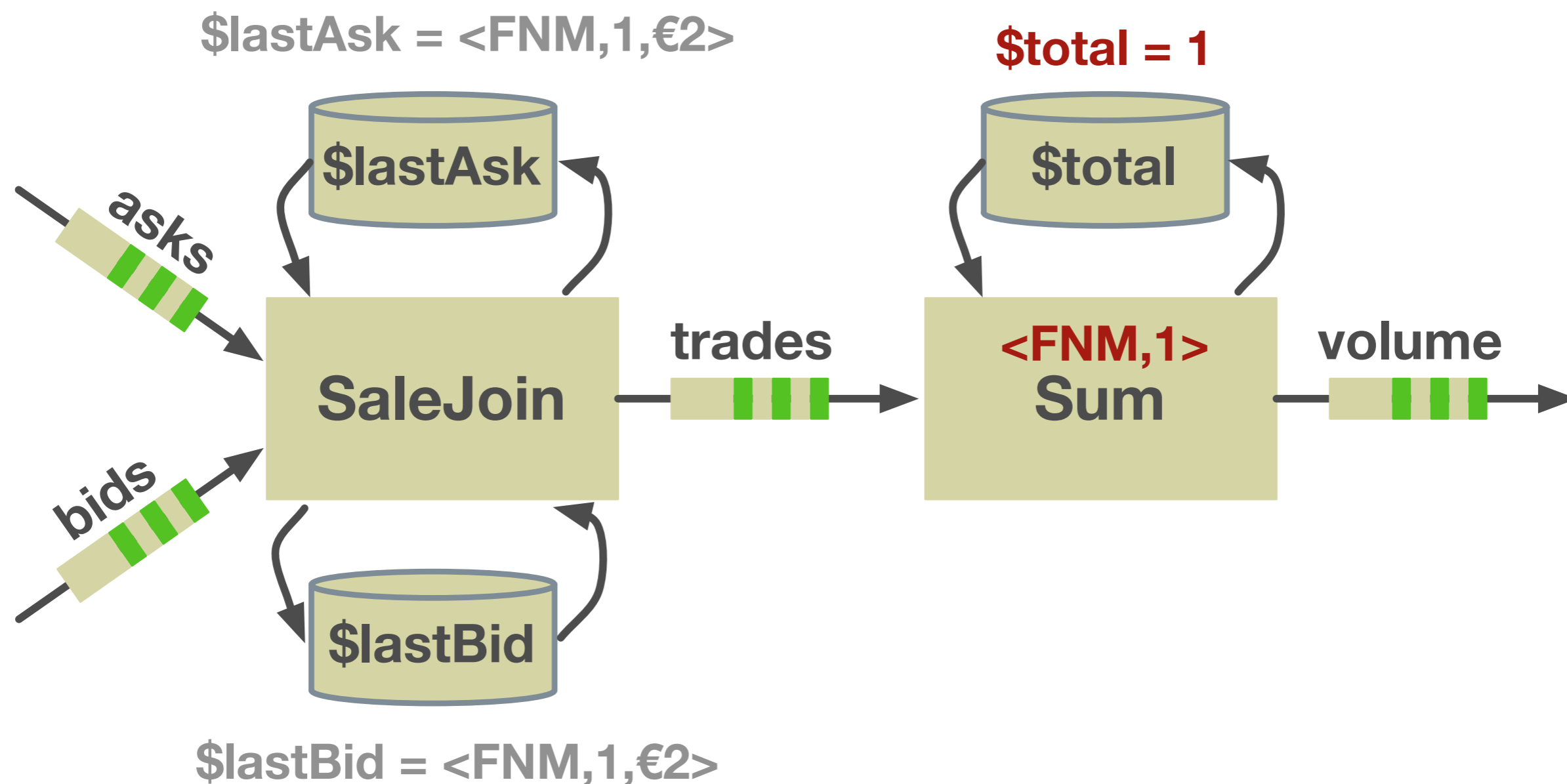
Example: A Fannie Mae Bid/Ask Join



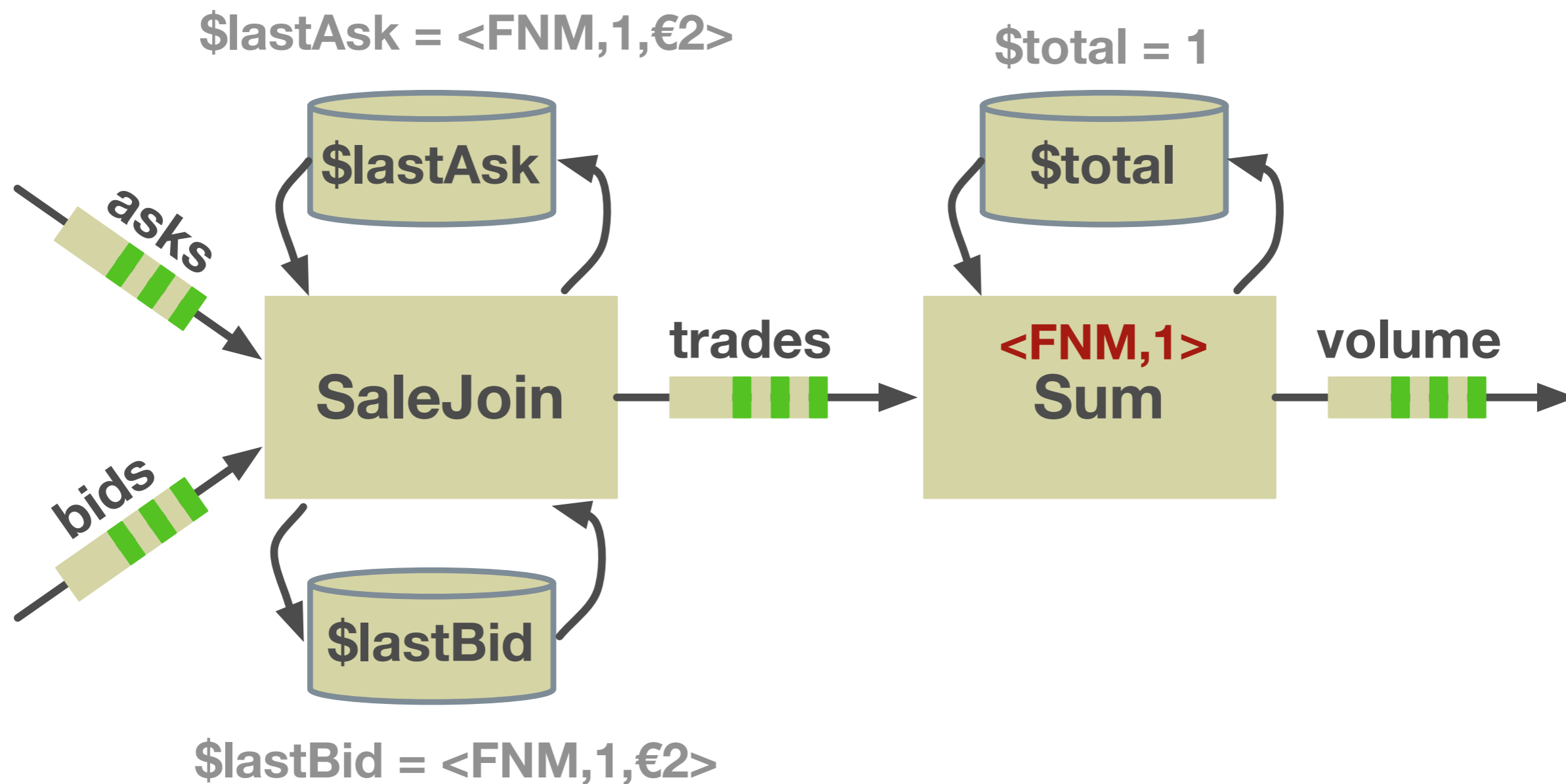
Example: A Fannie Mae Bid/Ask Join



Example: A Fannie Mae Bid/Ask Join



Example: A Fannie Mae Bid/Ask Join

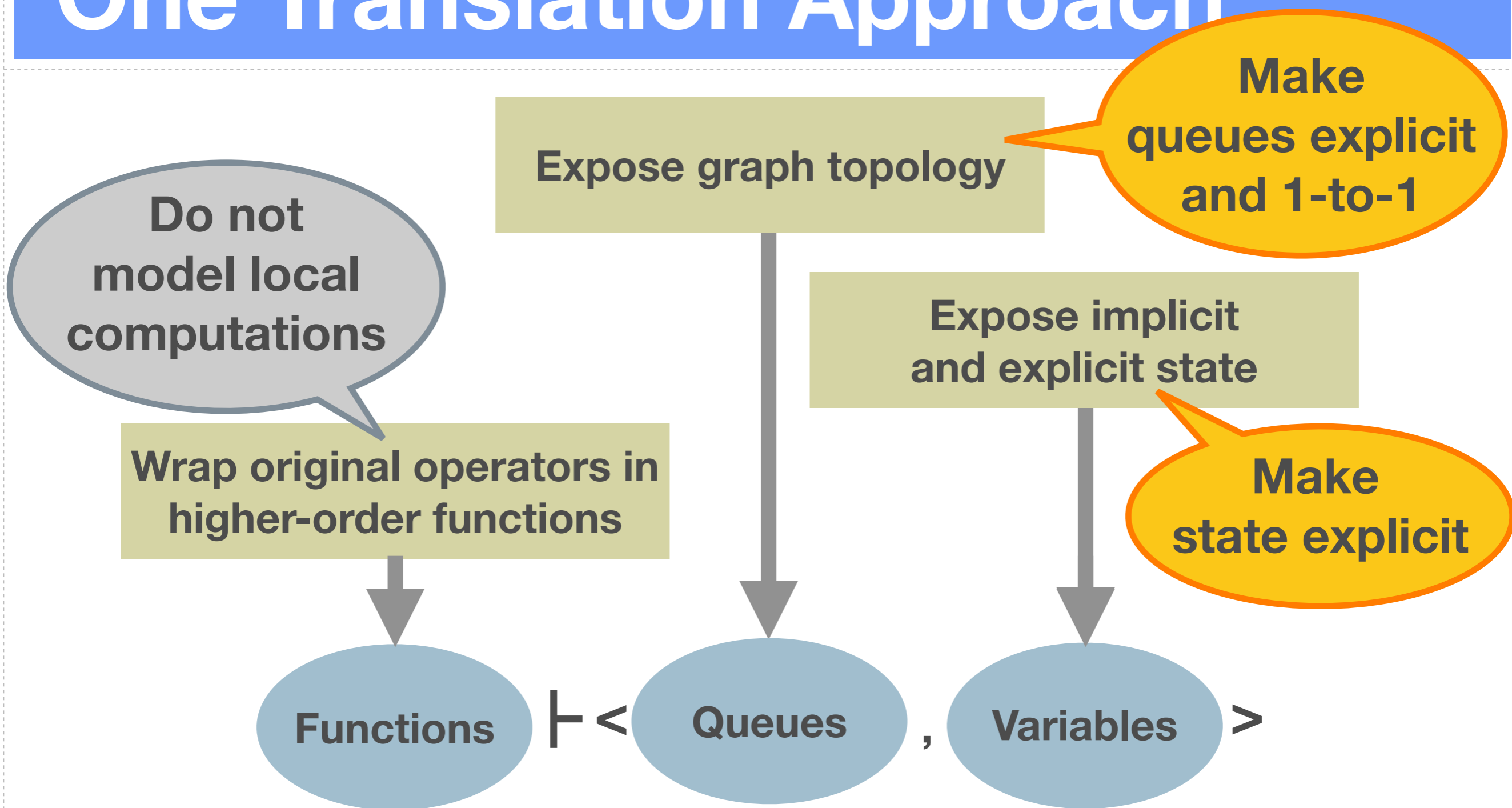


Translations

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



CQL, StreamIt, Sawzall: One Translation Approach

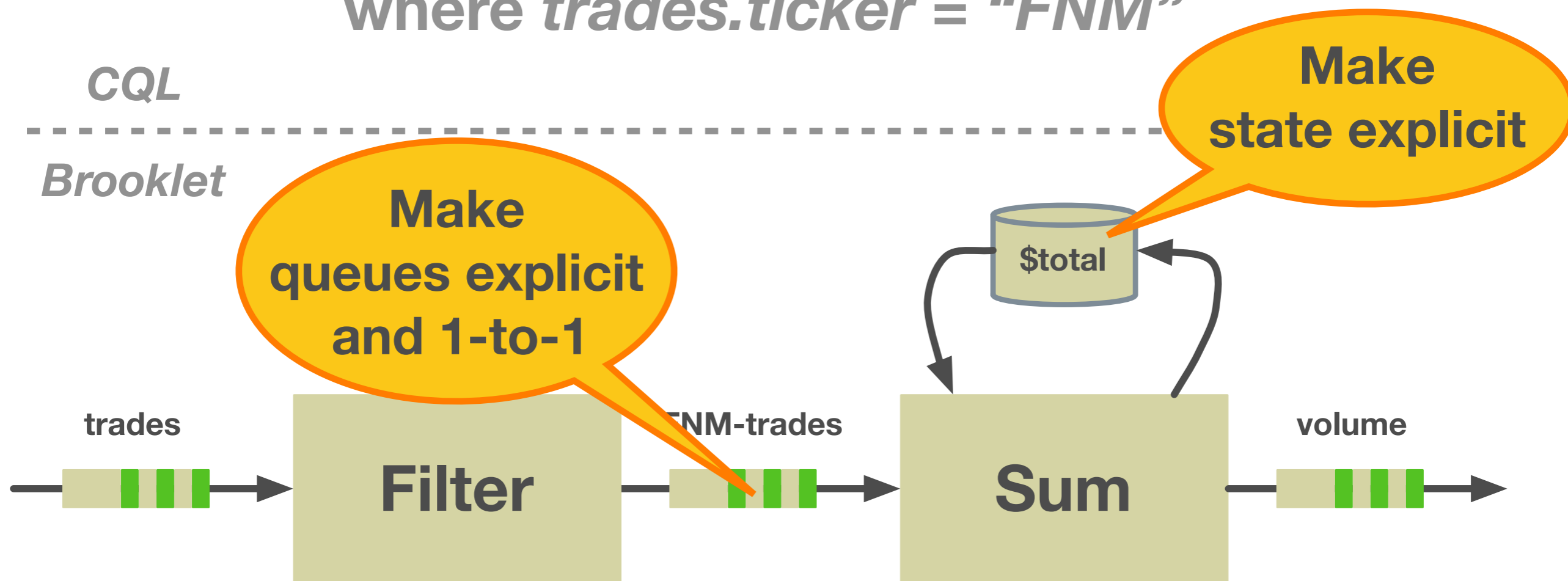


Example: CQL to Brooklet

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

CQL

Brooklet



Example: CQL to Brooklet

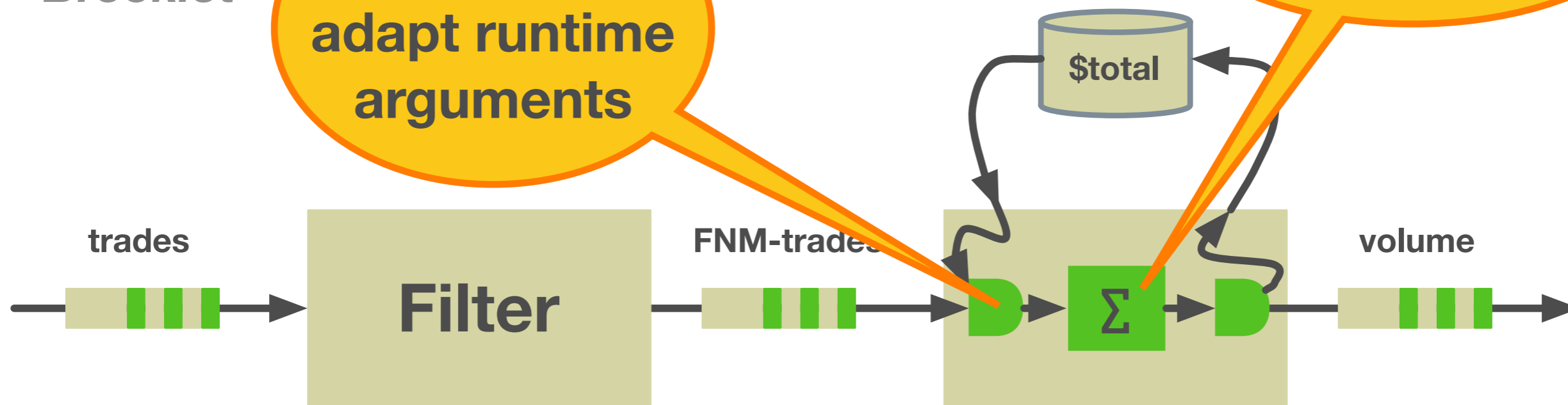
*select Sum(shares) from trades
where trades.ticker = "FNM"*

CQL

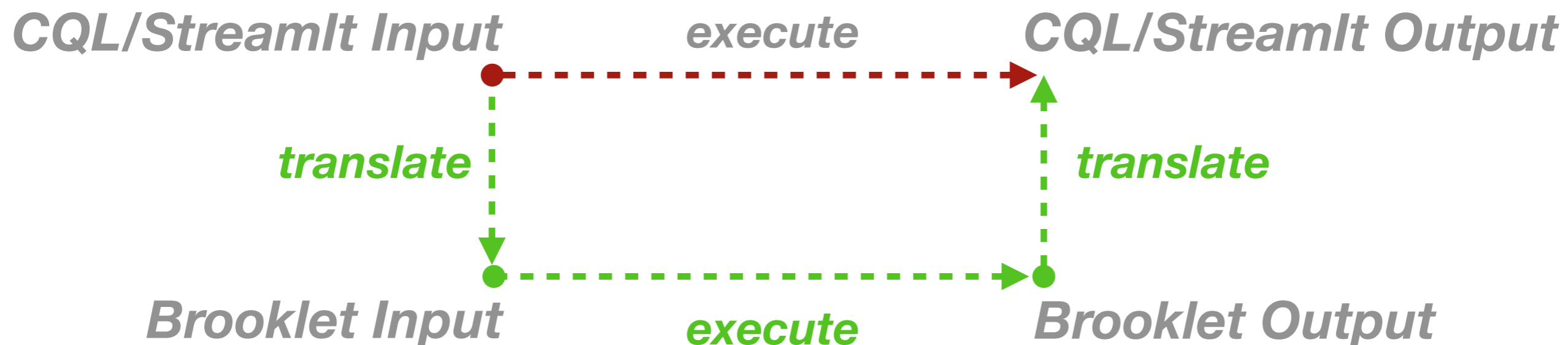
Brooklet

Dynamically adapt runtime arguments

Statically bind the original function



Translation Correctness Theorem



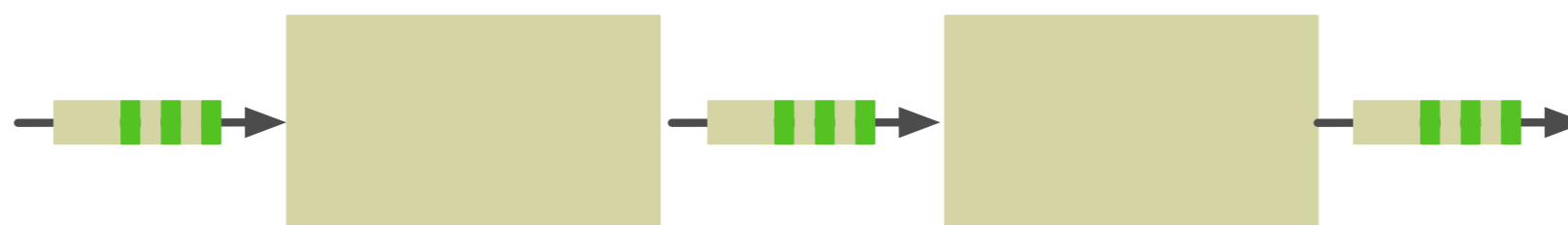
- ⬢ Results under CQL and StreamIt semantics are the same as the results under Brooklet semantics after translation
- ⬢ First formal semantics for Sawzall

Optimizations

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



Operator Fusion: Eliminate Queueing Delays



before



after



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

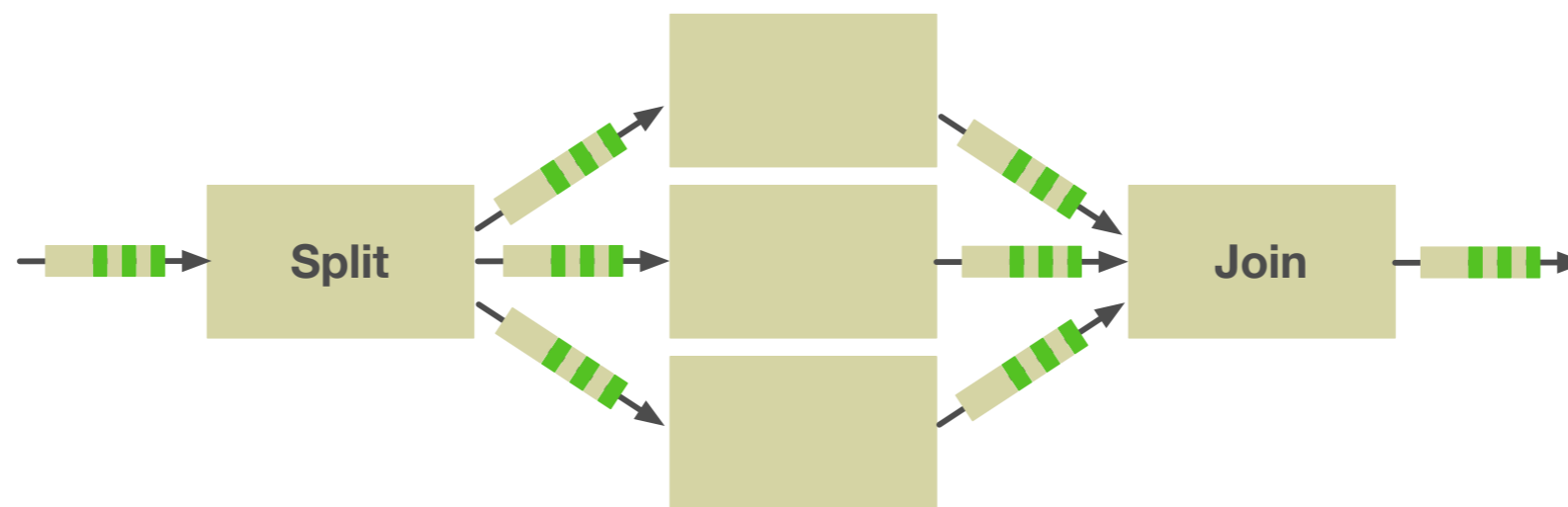
Operator Fission: Process More Data in Parallel



before

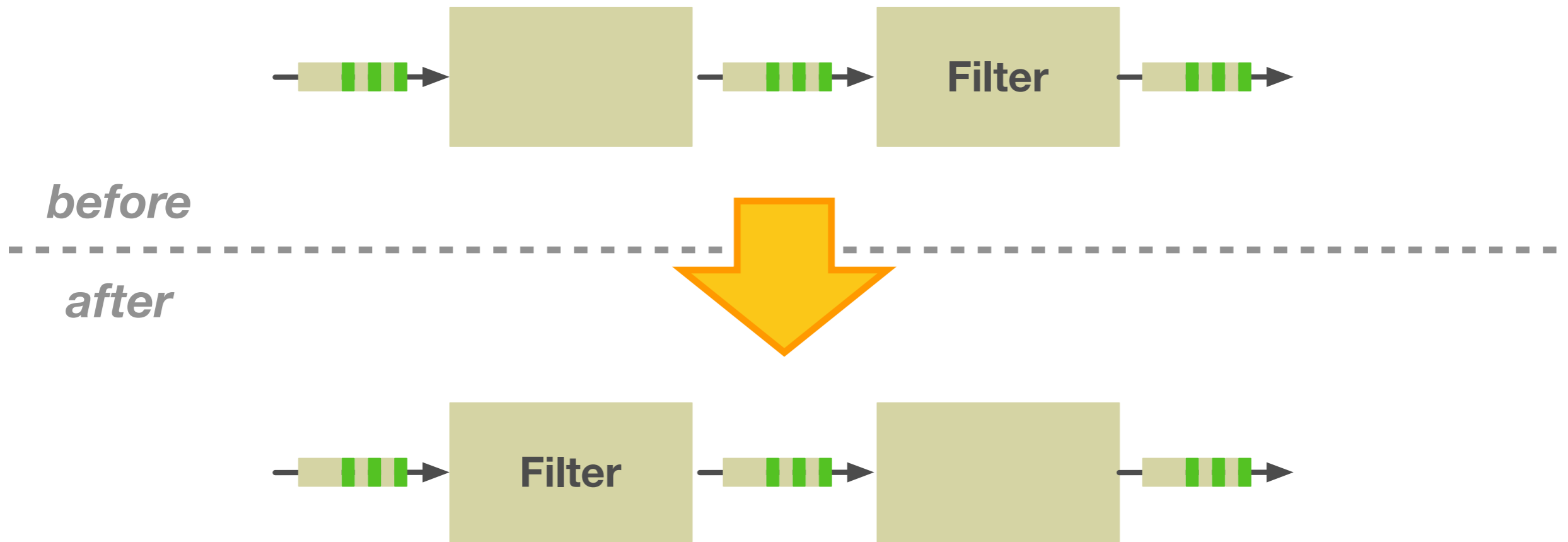


after



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

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: $\llbracket F_c, P_c \rrbracket_c^p = \langle F_b, P_b \rangle$
 $\llbracket F_c, SName \rrbracket_c^p = \emptyset, \text{output } SName; \text{input } SName; \bullet$
 (T_c^p-SNAME)

$\llbracket F_c, RName \rrbracket_c^p = \emptyset, \text{output } RName; \text{input } RName; \bullet$
 (T_c^p-RNAME)

$F_b, \text{output } q_o; \text{input } \bar{q}; \overline{op} = \llbracket F_c, P_{cs} \rrbracket_c^p$
 $q'_o = \text{freshId}() \quad v = \text{freshId}()$
 $F'_b = [S2R \mapsto \text{wrapS2R}(F_c(S2R))]F_b$
 $\overline{op}' = \overline{op}, (q'_o, v) \leftarrow S2R(q_o, v);$

$\llbracket F_c, S2R(P_{cs}) \rrbracket_c^p = F'_b, \text{output } q'_o; \text{input } \bar{q}; \overline{op}'$
 (T_c^p-S2R)

$F_b, \text{output } q_o; \text{input } \bar{q}; \overline{op} = \llbracket F_c, P_{cr} \rrbracket_c^p$
 $q'_o = \text{freshId}() \quad v = \text{freshId}()$
 $F'_b = [R2S \mapsto \text{wrapR2S}(F_c(R2S))]F_b$
 $\overline{op}' = \overline{op}, (q'_o, v) \leftarrow R2S(q_o, v);$

$\llbracket F_c, R2S(P_{cr}) \rrbracket_c^p = F'_b, \text{output } q'_o; \text{input } \bar{q}; \overline{op}'$
 (T_c^p-R2S)

$\overline{F_b, \text{output } q_o; \text{input } \bar{q}; \overline{op}} = \overline{\llbracket F_c, P_{cr} \rrbracket_c^p}$
 $n = |P_{cr}| \quad q'_o = \text{freshId}() \quad \bar{q}' = \bar{q}_1, \dots, \bar{q}_n$
 $\forall i \in 1 \dots n : v_i = \text{freshId}() \quad \overline{op}' = \overline{op}_1, \dots, \overline{op}_n$
 $F'_b = [R2R \mapsto \text{wrapR2R}(F_c(R2R))](\cup F_b)$
 $\overline{op}'' = \overline{op}', (q'_o, \bar{v}) \leftarrow R2R(\bar{q}_o, \bar{v});$

$\llbracket F_c, R2R(\overline{P_{cr}}) \rrbracket_c^p = F'_b, \text{output } q'_o; \text{input } \bar{q}'; \overline{op}''$
 (T_c^p-R2R)

CQL operator wrappers:

$\frac{\sigma, \tau = d_q \quad s = d_v}{s' = s \cup \{\langle e, \tau \rangle : e \in \sigma\} \quad \sigma' = f(s', \tau)}$
 $\frac{}{\text{wrapS2R}(f)(d_q, -, d_v) = \langle \sigma', \tau \rangle, s'}$
 (W_c-S2R)

$\frac{\sigma, \tau = d_q \quad \sigma' = d_v \quad \sigma'' = f(\sigma, \sigma')}{\text{wrapR2S}(f)(d_q, -, d_v) = \langle \sigma'', \tau \rangle, \sigma}$
 (W_c-R2S)

$\frac{\sigma, \tau = d_q \quad d'_i = d_i \cup \{\langle \sigma, \tau \rangle\}}{\forall j \neq i \in 1 \dots n : d'_j = d_j}$
 $\frac{\exists j \in 1 \dots n : \nexists \sigma : \langle \sigma, \tau \rangle \in d_j}{\text{wrapR2R}(f)(d_q, i, \bar{d}) = \bullet, \bar{d}'}$
 (W_c-R2R-WAIT)

$\frac{\sigma, \tau = d_q \quad d'_i = d_i \cup \{\langle \sigma, \tau \rangle\}}{\forall j \neq i \in 1 \dots n : d'_j = d_j}$
 $\frac{\forall j \in 1 \dots n : \sigma_j = \text{aux}(d_j, \tau)}{\text{wrapR2R}(f)(d_q, i, \bar{d}) = \langle f(\bar{\sigma}), \tau \rangle, \bar{d}'}$
 (W_c-R2R-READY)

$\frac{\langle \sigma, \tau \rangle \in d}{\text{aux}(d, \tau) = \sigma}$
 (W_c-R2R-AUX)

Operator Fission

$$op = (q_{out}) \leftarrow f(q_{in});$$

$$\forall i \in 1 \dots n : q_i = \text{freshId}() \quad \forall i \in 1 \dots n : q'_i = \text{freshId}()$$

$$F'_b, op_s = \llbracket \emptyset, \text{split roundrobin}, \bar{q}, q_{in} \rrbracket_s^p$$

$$\forall i \in 1 \dots n : op_i = (q'_i) \leftarrow f(q_i);$$

$$F''_b, op_j = \llbracket \emptyset, \text{join roundrobin}, q_{out}, \bar{q}' \rrbracket_s^p$$

$$\langle F_b, op \rangle \longrightarrow_{split}^N \langle F_b \cup F'_b \cup F''_b, op_s \ \overline{op} \ op_j \rangle$$