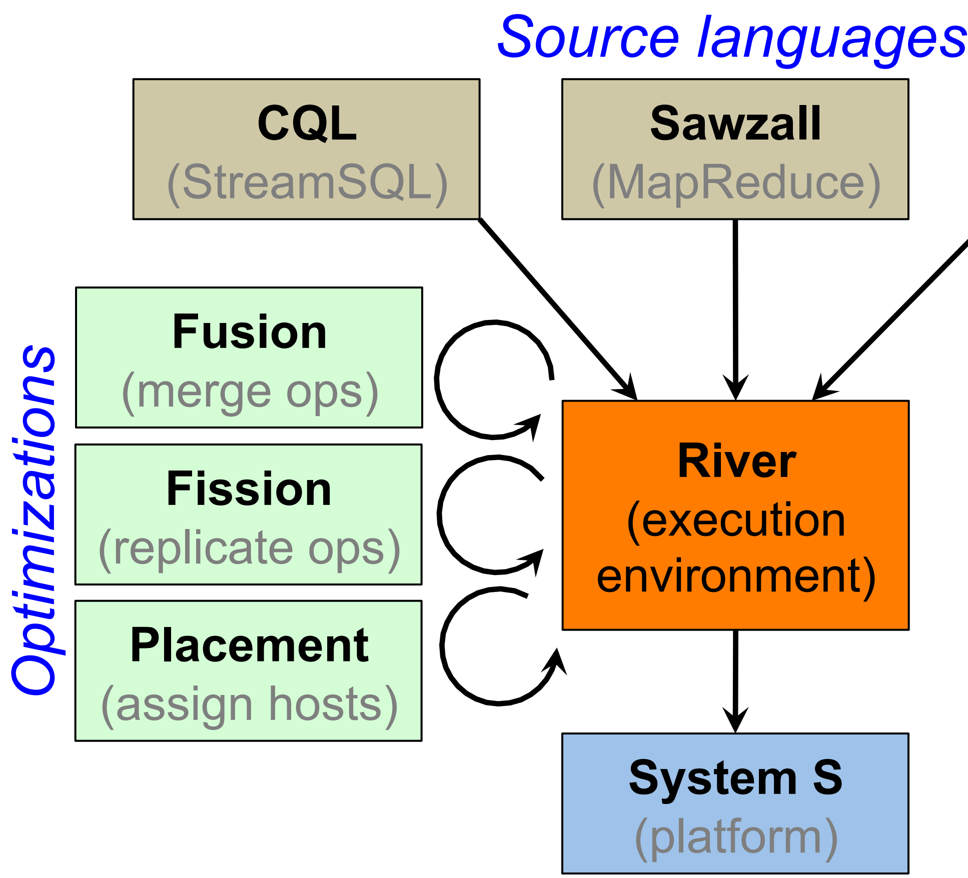
From a Calculus to an Execution Environment for Stream Processing

Robert Soulé Martin Hirzel Buğra Gedik Robert Grimm

Cornell University IBM Research Bilkent University New York University

DEBS 2012

… to an Execution Environment

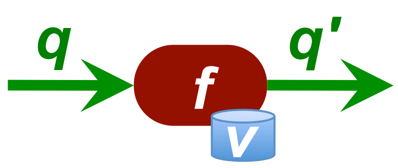


Benefits of execution environment:

- *Language portability*
- *Optimization reuse*

From a Calculus …

- Calculus = formal language + semantics
  - Stream calculus, Soulé et al. [ESOP'10]
- Graph language:
  - Stream operators with functions (***F***)
  - Queues (***Q***)
  - Variables (***V***)



- Semantics:
  - Small-step
  - Operational
  - Sequence of "operator firings"
    $F \vdash \langle Q_1, V_1 \rangle$
    $\rightarrow_b \langle Q_2, V_2 \rangle$
    $\rightarrow_b^* \ldots$

Benefits of Calculus: Translation Correctness Proofs

Input Execute Output

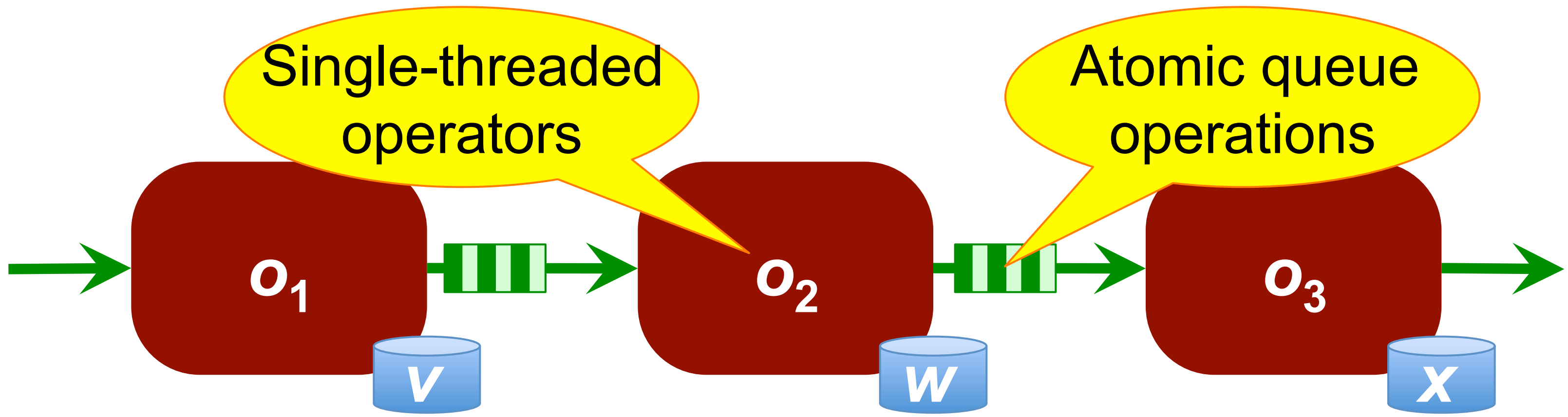
$$
\begin{tikzcd}
F_s, P_s, I_s \arrow[rr, "\longrightarrow_s^*"] \arrow[d, "\text{Translate}"', "{[\![\cdot]\!]_s^{p,i}}"] & & O_s \\
F_b, P_b, I_b \arrow[rr, "\longrightarrow_b^*"'] & & O_b \arrow[u, "{[\![\cdot]\!]_s^{o}}"', "\text{Translate}"]
\end{tikzcd}
$$

Input Execute Output

From Abstractions to the Real World

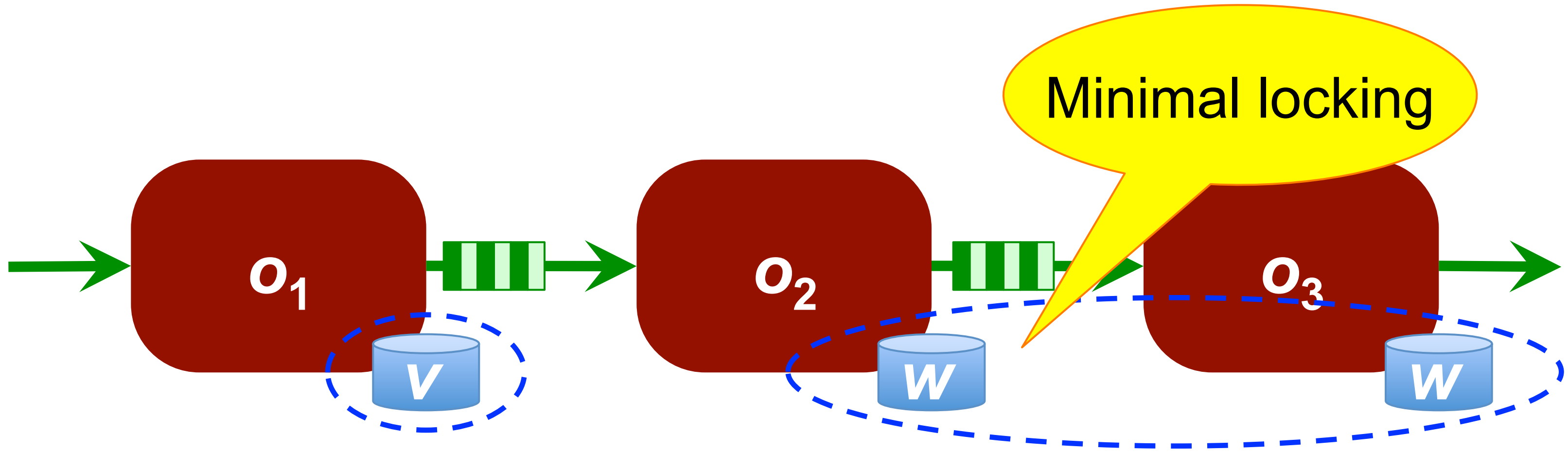
Brooklet calculus	River execution environment
Sequence of atomic steps	Operators execute concurrently
Pure functions, state threaded through invocations	Stateful functions, protected with automatic locking
Non-deterministic execution	Restricted execution: bounded queues and back-pressure
Opaque functions	Function implementations
No physical platform, independent from runtime	Abstract representation of platform, e.g. placement
Finite execution	Indefinite execution

Concurrent Execution
Case 1: No Shared State



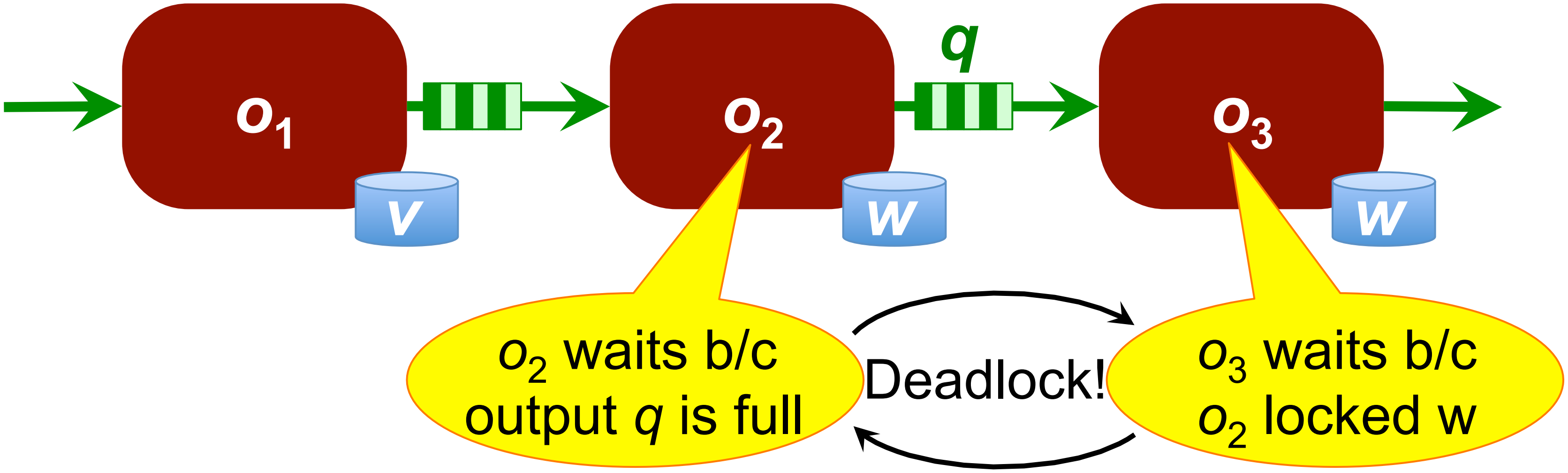
- Brooklet operators fire one at a time
- River operators fire concurrently
- For both, data must be available

Concurrent Execution
Case 2: With Shared State



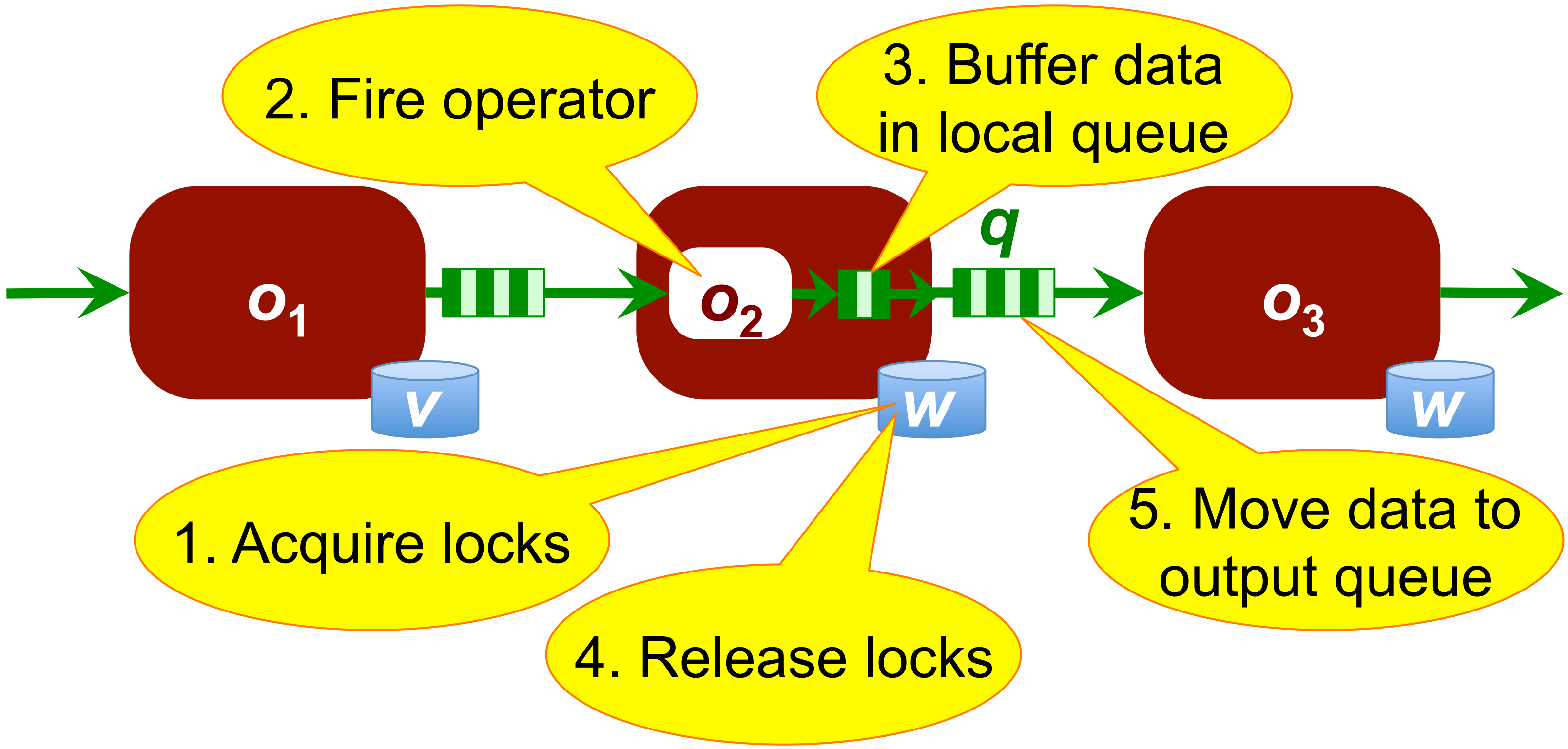
- Locks form equivalence classes over shared variables
- Every shared variable is protected by one lock
- Shared variables in the same class protected by same lock
- Locks acquired/released in standard order

Restricted Execution
Bounded Queues



- Naïve approach:
  block when output queue is full

Restricted Execution
Safe Back-Pressure



- Our approach: only block on output queue when not holding locks on variables

Applications of an Execution Environment

- Easier to develop source languages
  - Implementation language
  - Language modules
  - Operator templates
- Possible to reuse optimizations
  - Annotations provide additional information between source and intermediate language

Function Implementations and Translations

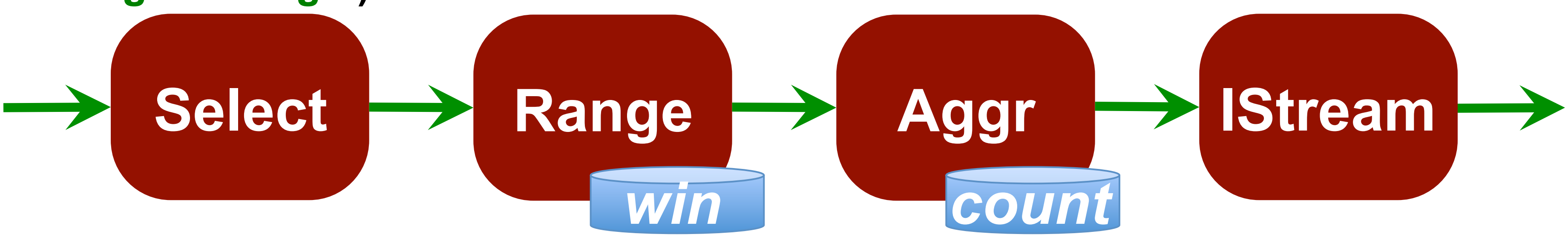
logs : {origin : string; target : string} stream;
hits : {origin : string; count : int} stream =
select istream(origin, count(origin))
from logs[range 300]
where origin != target

Expose operators, communication, and state

Pre-existing operator templates

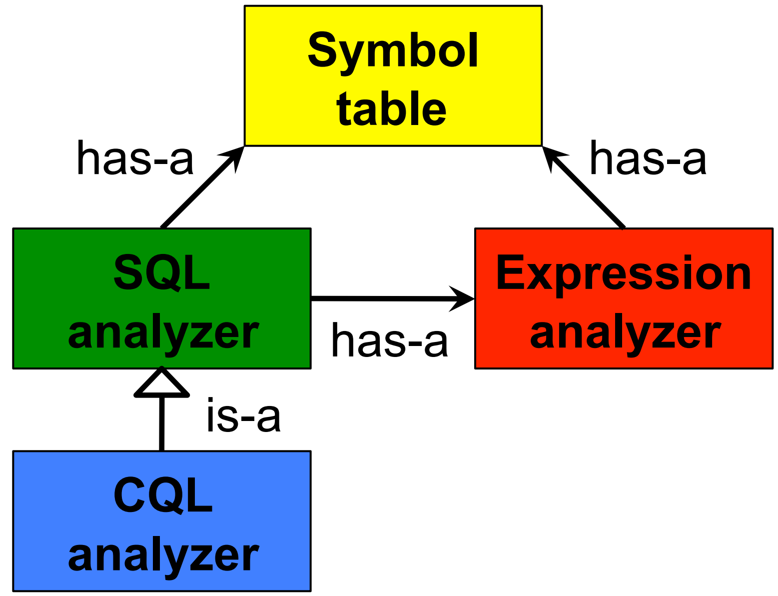
Bag.filter (fun x -> #expr)

Bag.filter (fun x ->
origin != target)



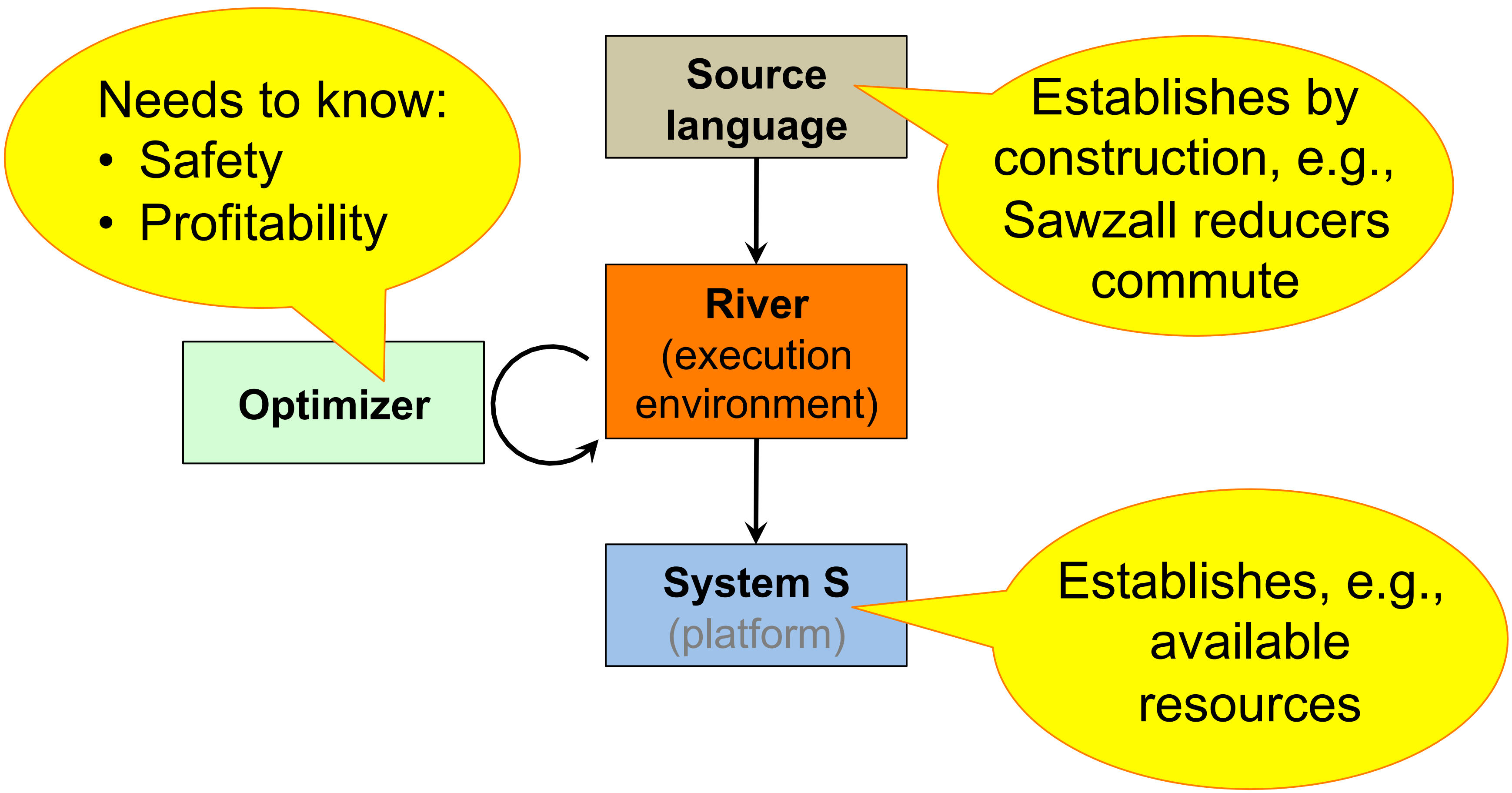
Translation Support: Pluggable Compiler Modules

```
select istream(*)
   from quotes[now], history
   where quotes.ask<=history.low
      and quotes.ticker=history.ticker
```



CQL = SQL + Streaming + Expressions

Optimization Support: Extensible Annotations



Optimization Support: Current Annotations

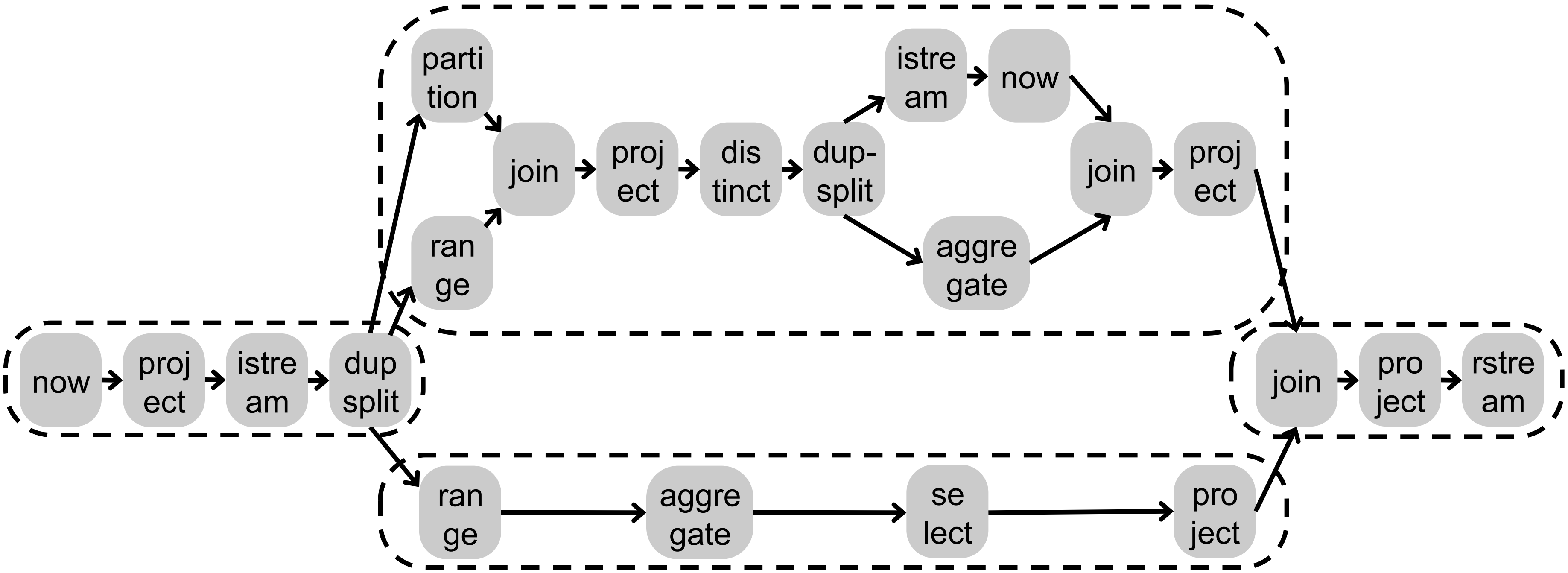
Annotation	Description	Optimization
@Fuse(ID)	Fuse operators with same ID in the same process	Fusion
@Parallel()	Perform fission on an operator	Fission
@Commutative()	An operator's function is commutative	Fission
@Keys($k_1,\ldots,k_n$)	An operator's state is partitionable by fields $k_1,\ldots,k_n$	Fission
@Group(ID)	Place operators with same ID on the same machine	Placement

Evaluation

- Four benchmark applications
  - CQL linear road
  - StreamIt FM radio
  - Sawzall web log analyzer (batch)
  - CQL web log analyzer (continuous)
- Three optimizations
  - Placement
  - Fission
  - Fusion

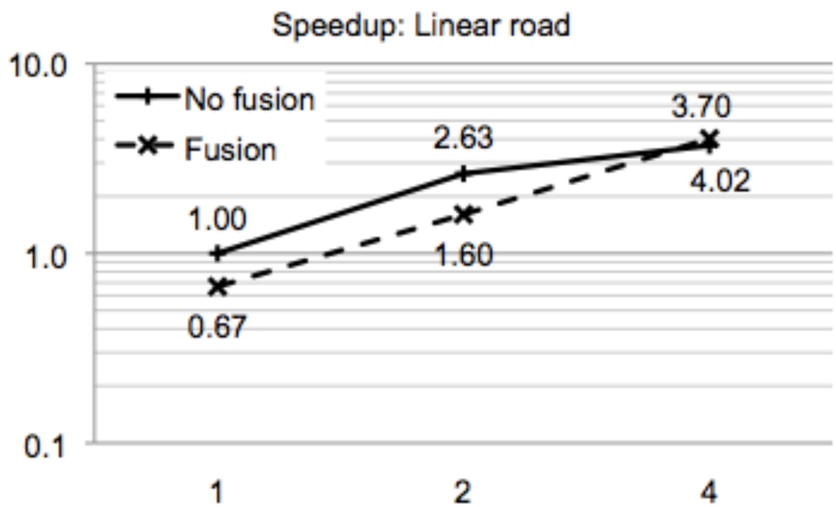
Distributed Linear Road

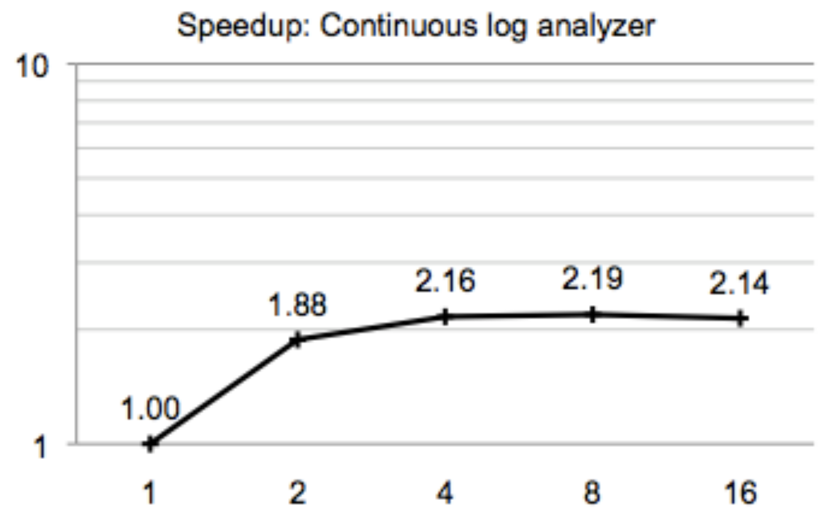
(simplified version from Arasu/Babu/Widom [VLDBJ'06])



First distributed CQL implementation

CQL: Placement, Fusion, Fission

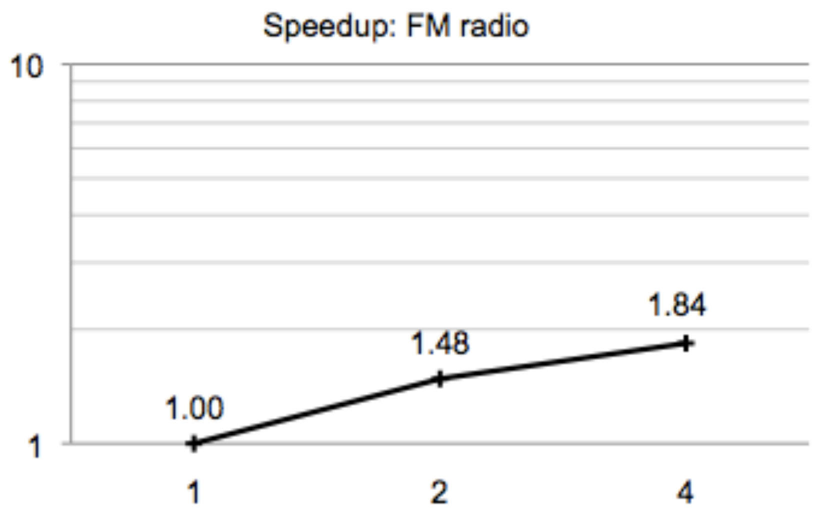


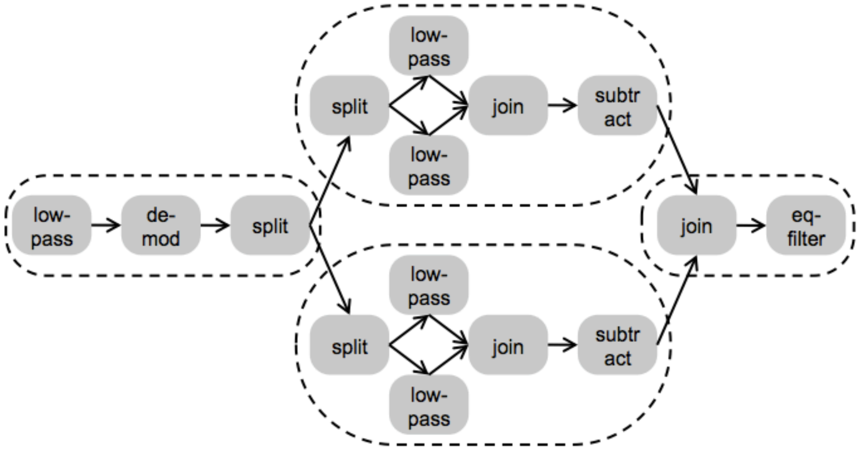


- *Placement + Fusion*
  → 4x speedup on 4 machines

- *Fission*
  → 2x speedup on 16 machines
- *Insufficient work per operator*

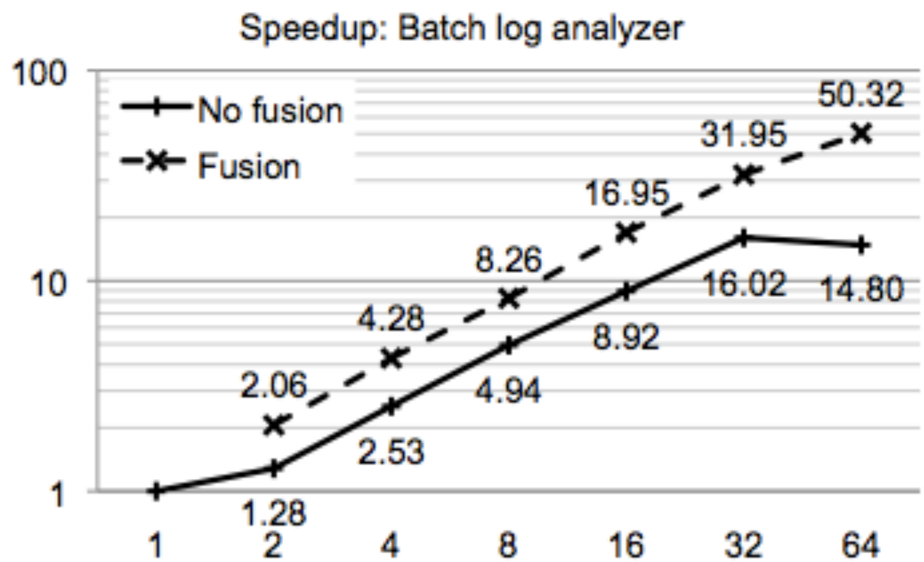
StreamIt: Placement





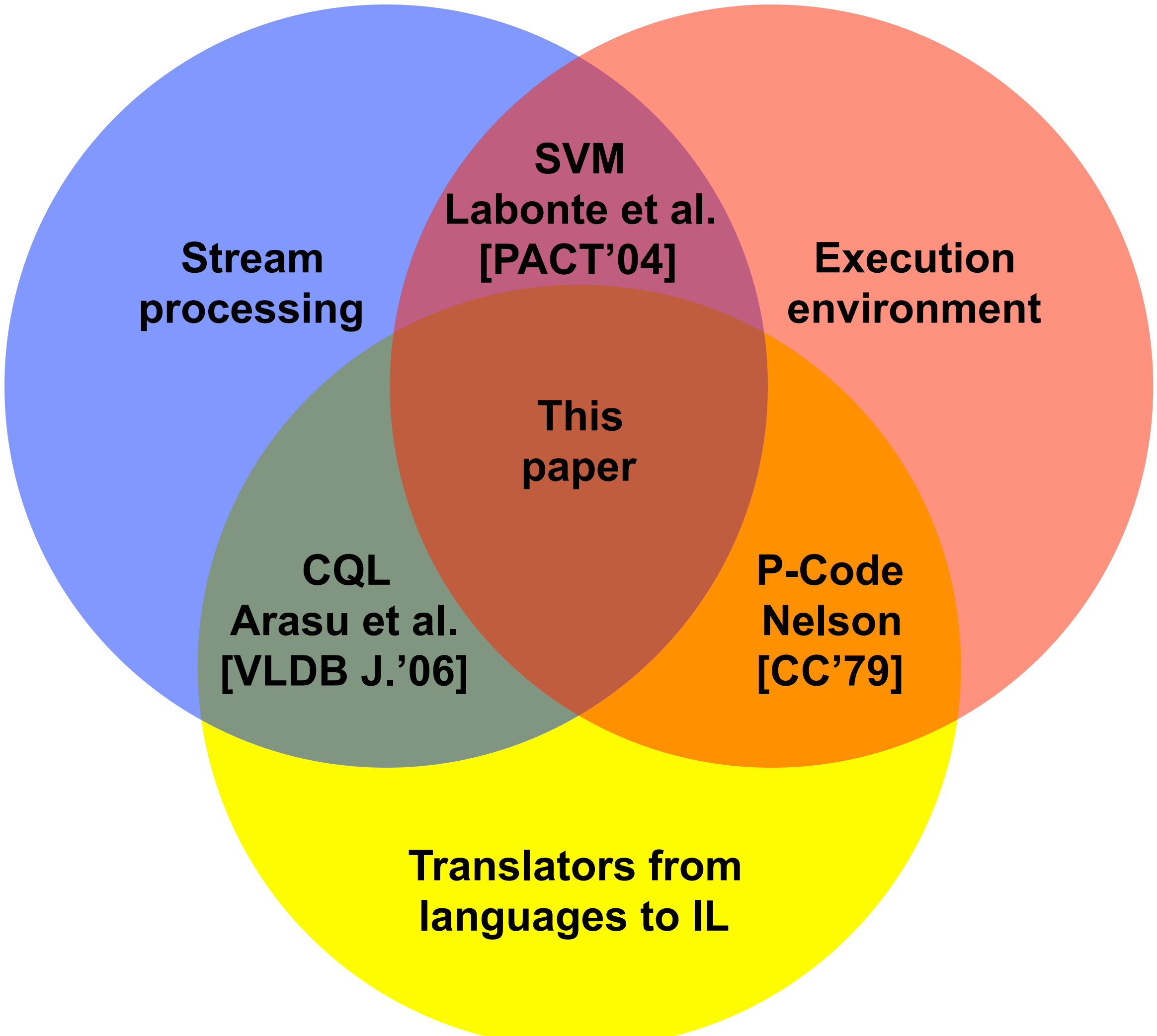
- *Optimization reuse → 1.8x speedup on 4 machines*

Sawzall (MapReduce on River) Fission + Fusion



- *Same fission optimizer for Sawzall as for CQL*
- *8.92x speedup on 16 machines, 14.80x on 64 cores*
- *With fusion, 50.32x on 64 cores*

Related Work



Conclusions

- River, execution environment for streaming
- Semantics specified by formal calculus
  - Brooklet, Soulé et al. [ESOP'10]
- 3 source languages, 3 optimizations
  - First distributed CQL
  - Language compiler module reuse
  - Optimization enabled by annotations
- Encourages innovation in stream processing
- http://www.cs.nyu.edu/brooklet/