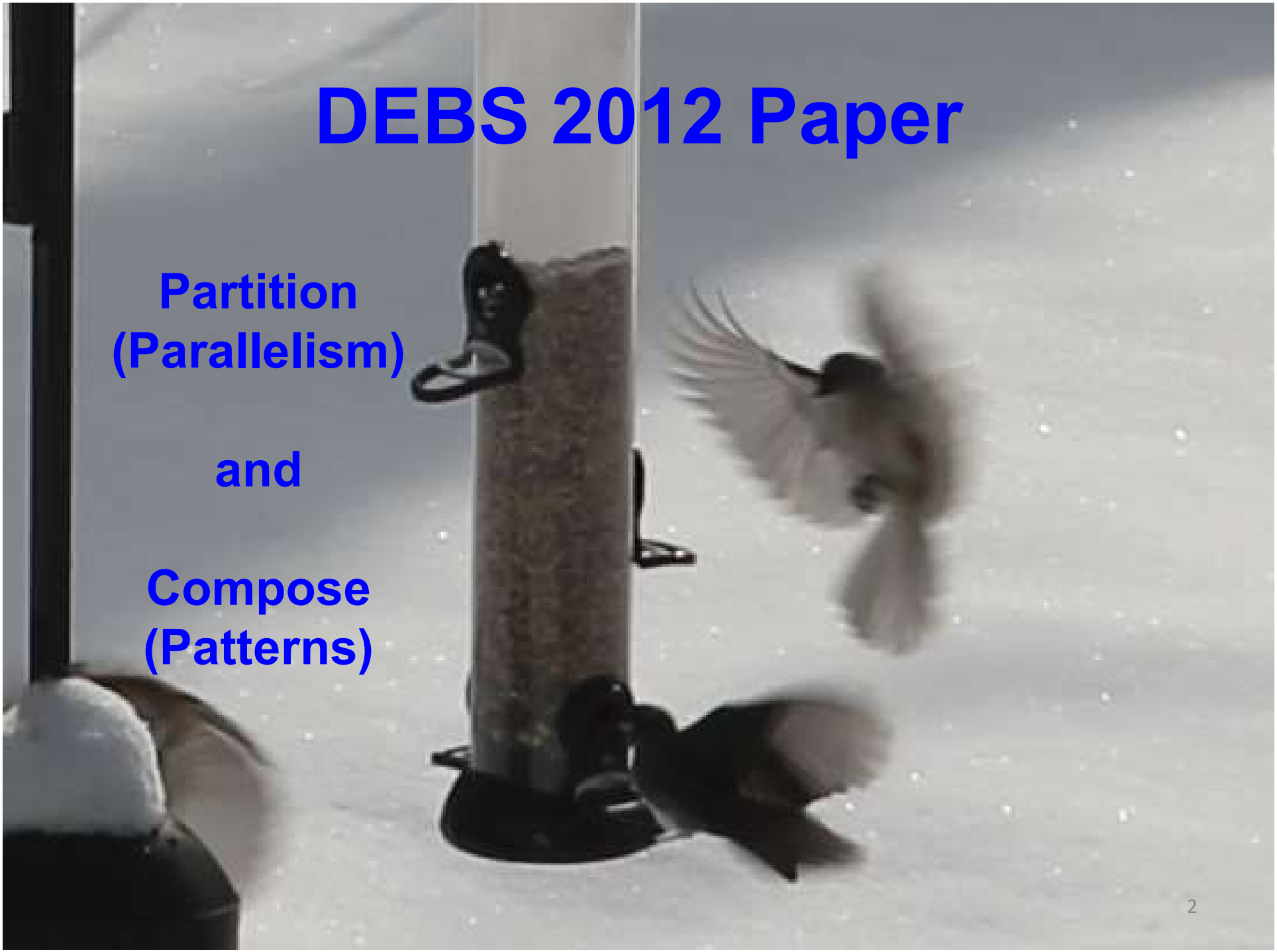
Partition and Compose: Parallel Complex Event Processing (Ten Years Later)

Martin Hirzel, IBM Research

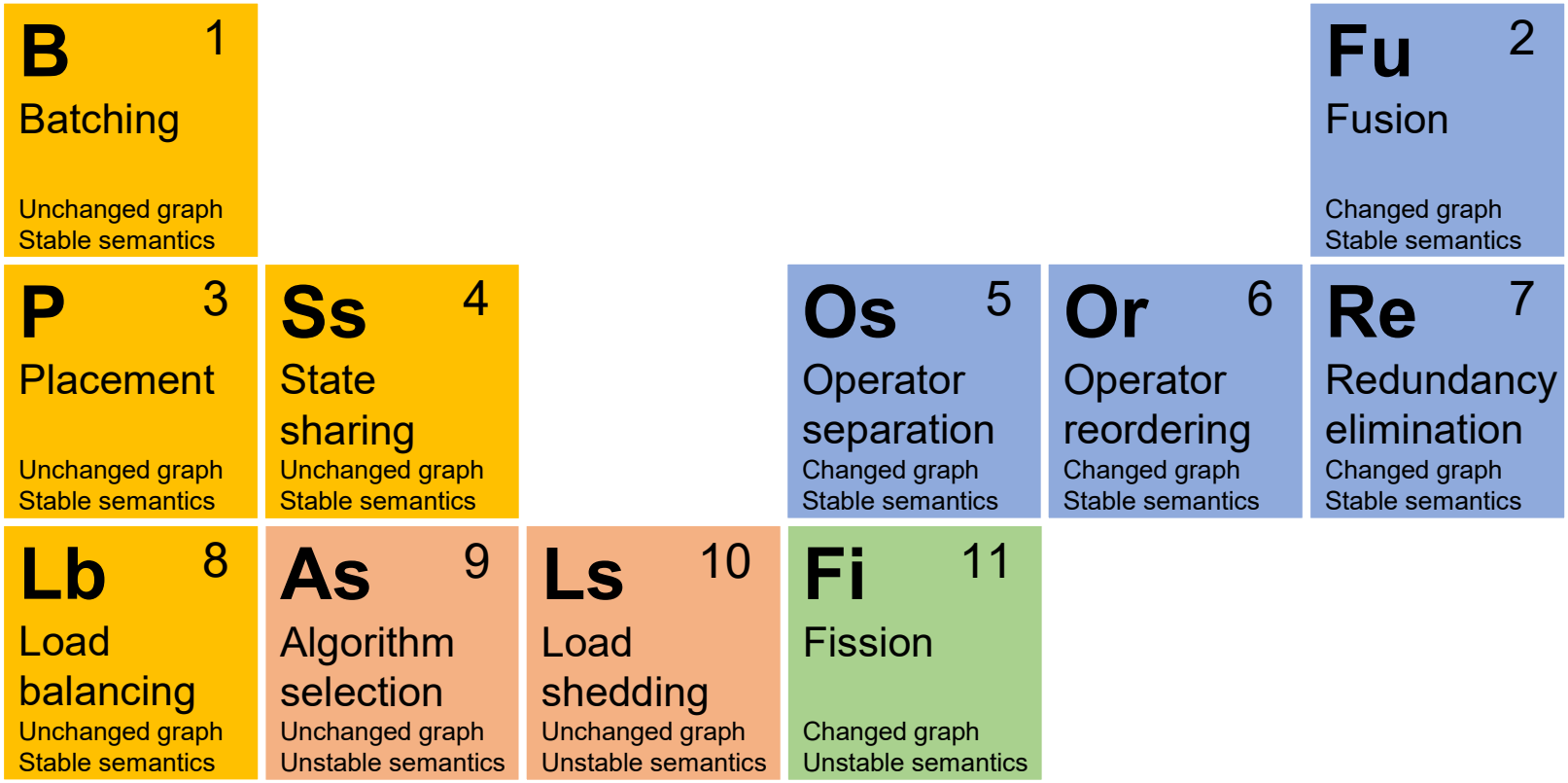
Thursday, 30 June 2022

DEBS


DEBS 2012 Paper
Partition
(Parallelism)
and
Compose
(Patterns)


Feeding 2 Birds from 1 Feeder
Partition (Parallelism)
and
Compose (Patterns)
Speed
Ease of Coding

Stream Processing Optimizations



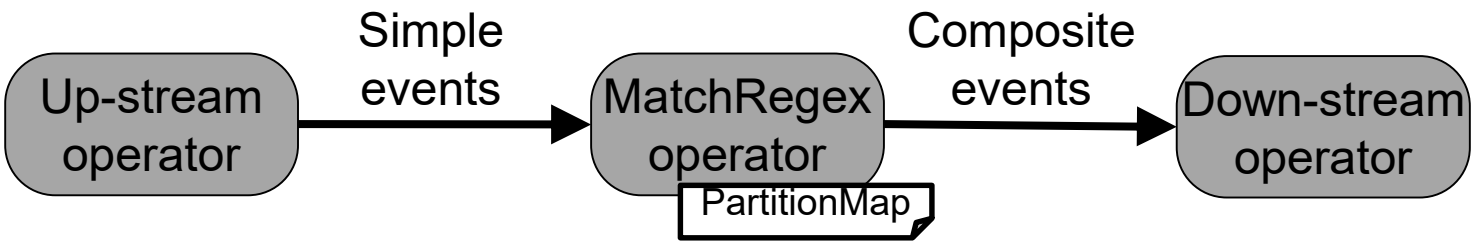
➔ ***DEBS 2013 Tutorial and CSUR 2014 article***

Fission for Parallelism

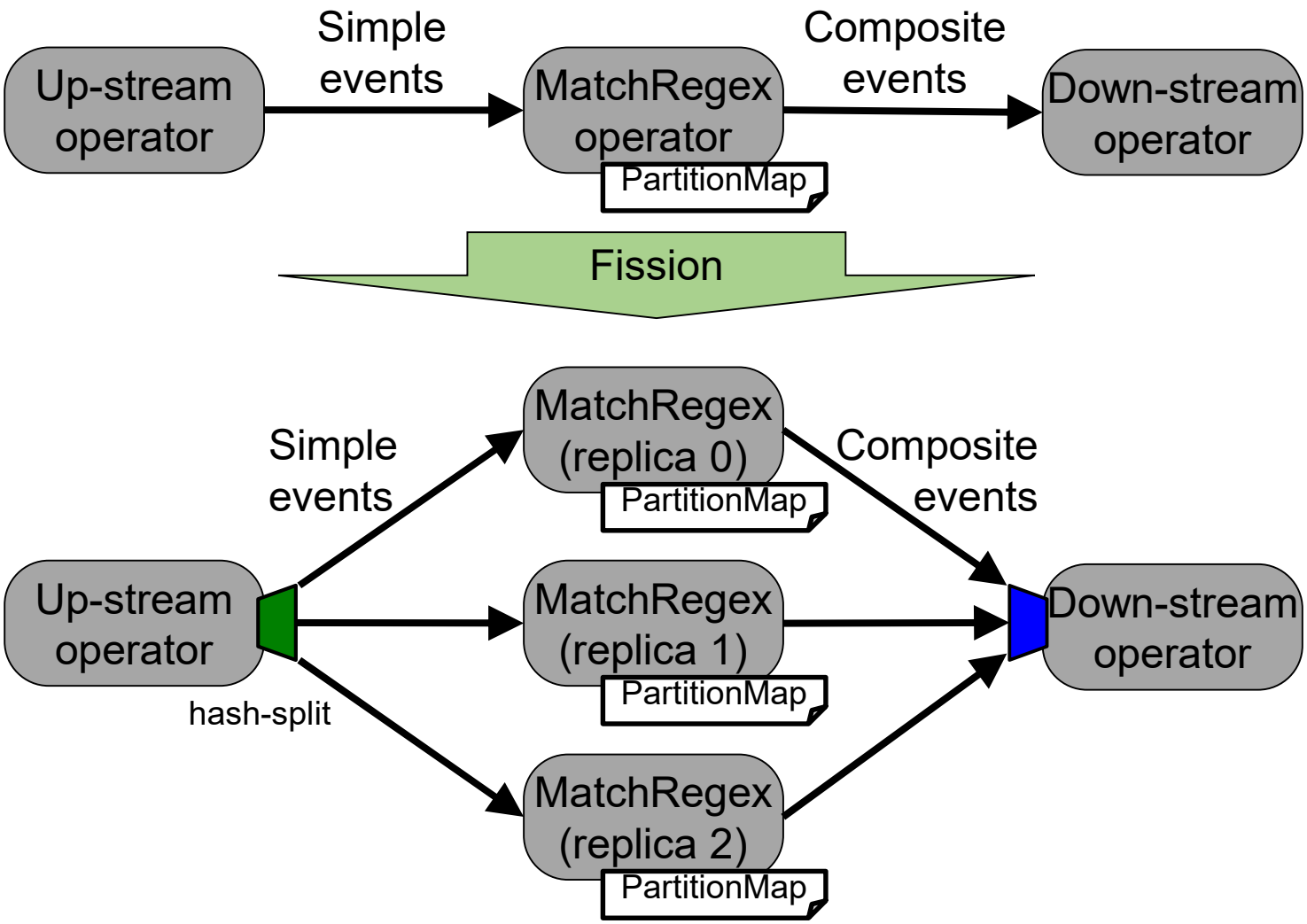
B 1 Batching Unchanged graph Stable semantics					**Fu** 2 Fusion Changed graph Stable semantics
P 3 Placement Unchanged graph Stable semantics	**Ss** 4 State sharing Unchanged graph Stable semantics		**Os** 5 Operator separation Changed graph Stable semantics	**Or** 6 Operator reordering Changed graph Stable semantics	**Re** 7 Redundancy elimination Changed graph Stable semantics
Lb 8 Load balancing Unchanged graph Stable semantics	**As** 9 Algorithm selection Unchanged graph Unstable semantics	**Ls** 10 Load shedding Unchanged graph Unstable semantics	**Fi** 11 Fission Changed graph Unstable semantics		

➔ ***partitioning for stable semantics***

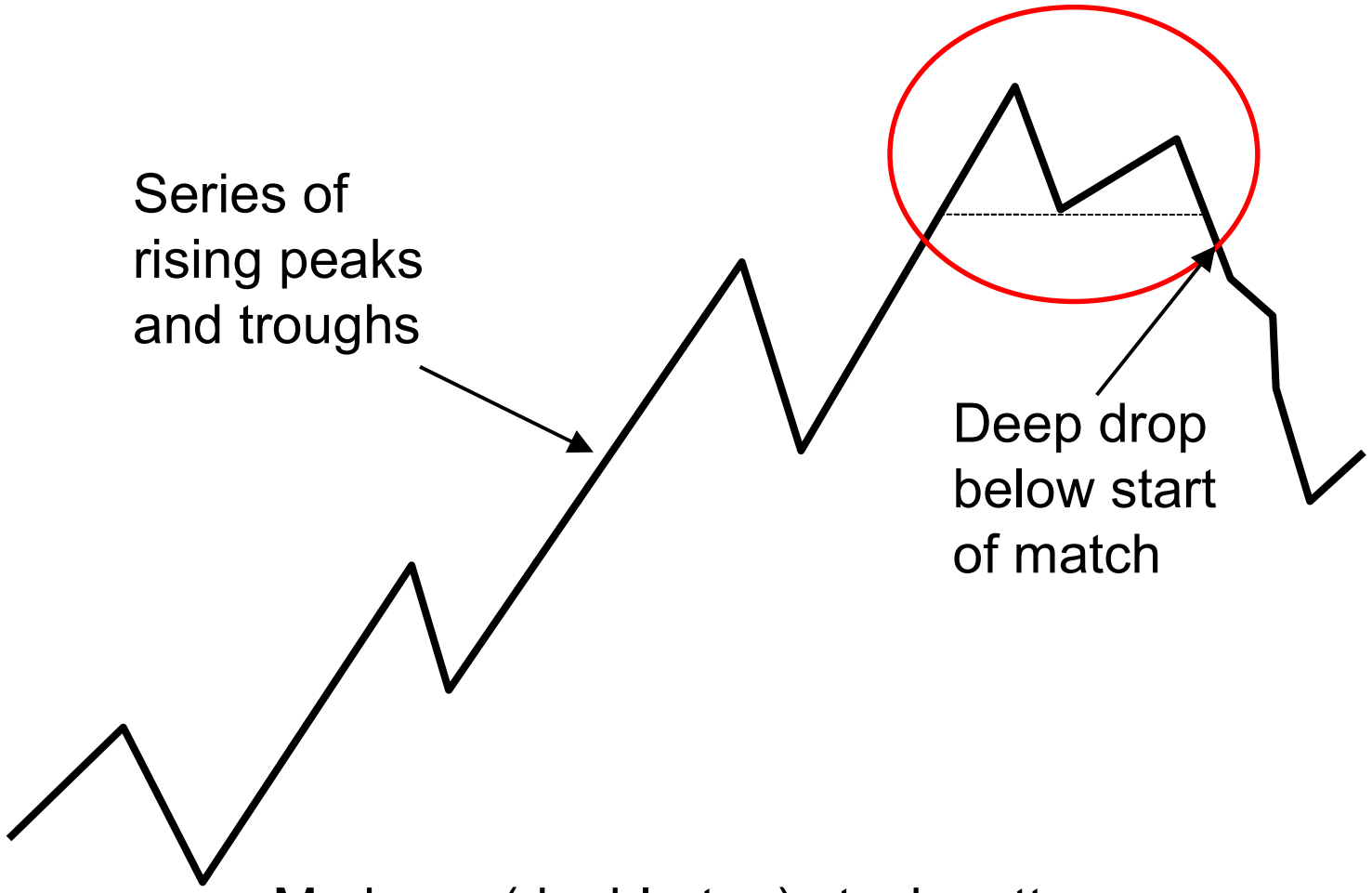
Partition Parallelism



Partition Parallelism



Scenario: Financial analysis

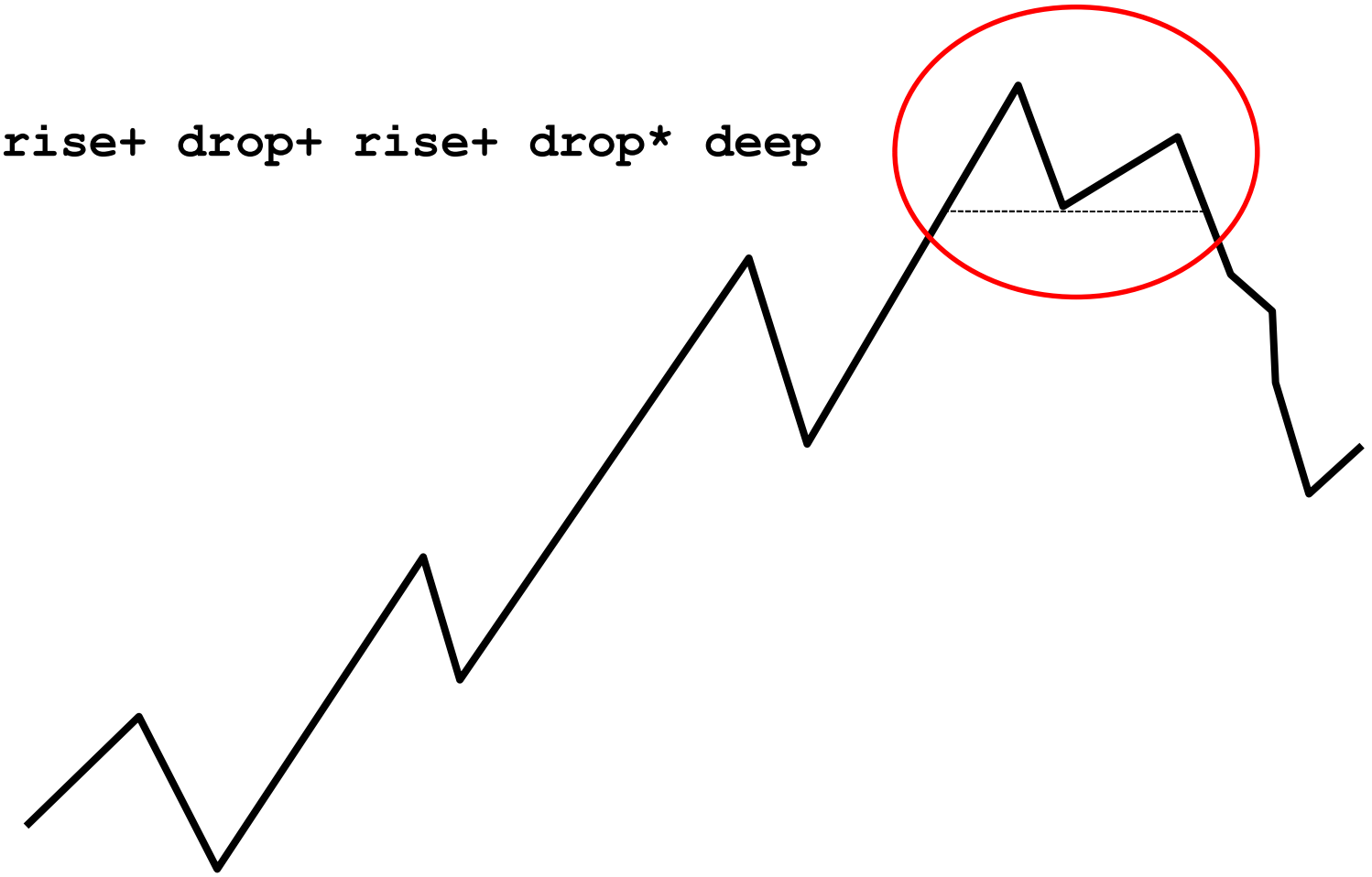


M-shape (double-top) stock pattern

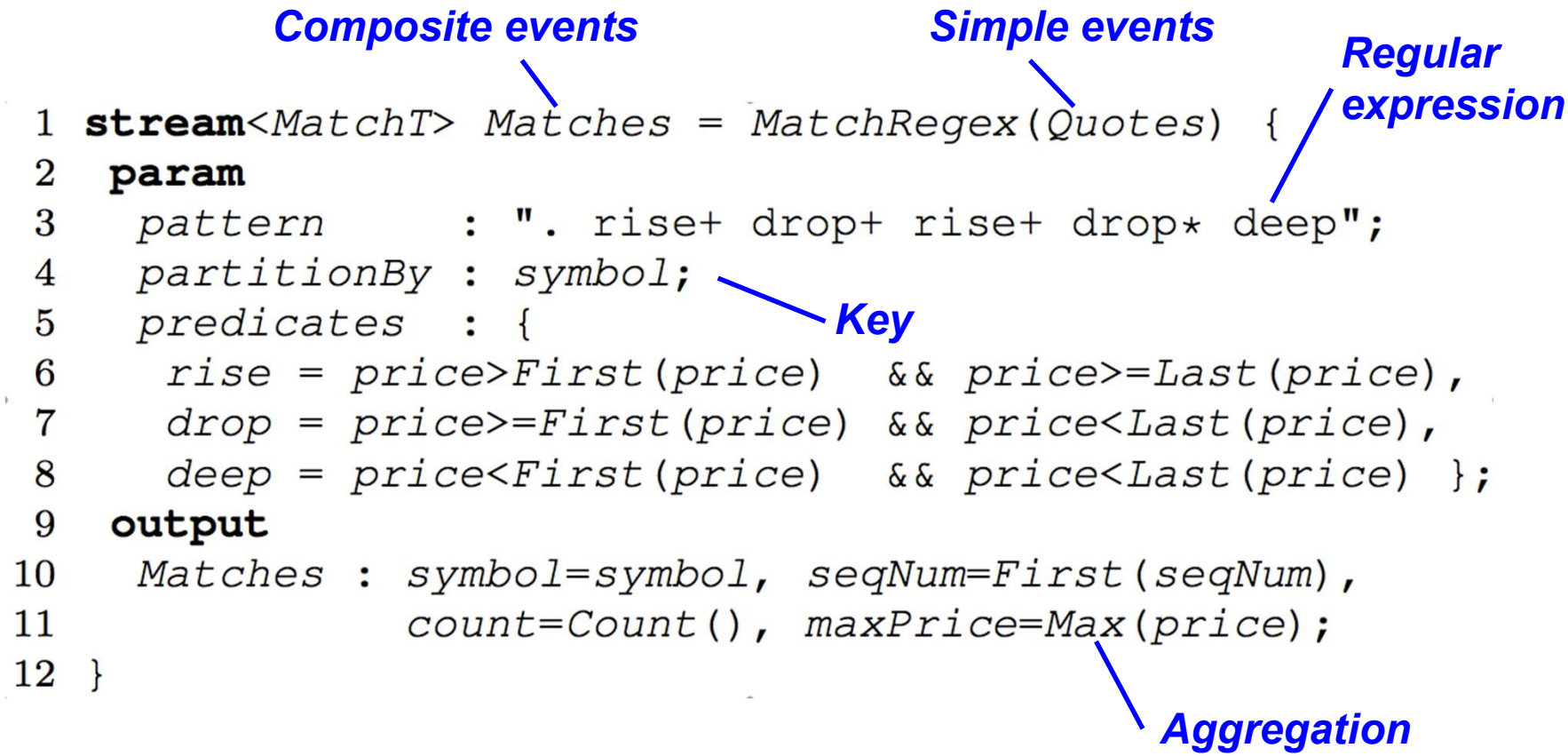
Source: http://www.cs.cornell.edu/bigreddata/cayuga/

Regular Expression

```
. rise+ drop+ rise+ drop* deep
```



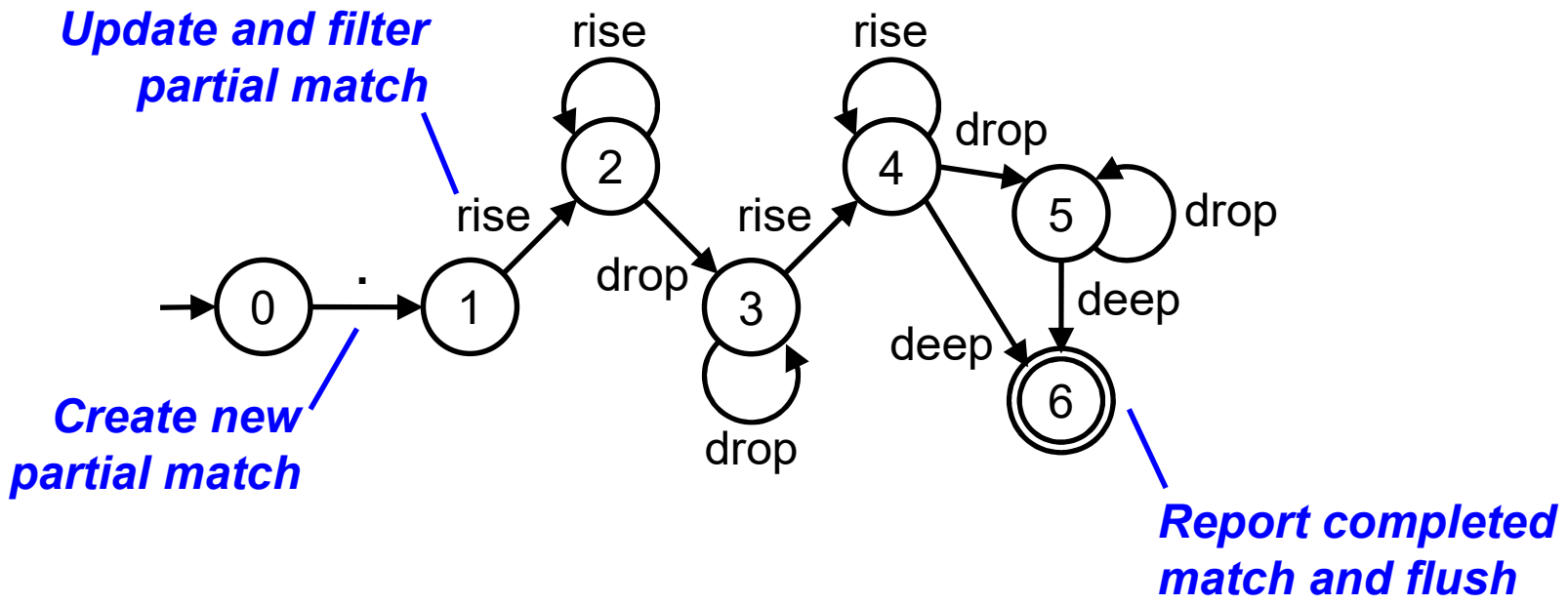
M-Shape pattern in SPL



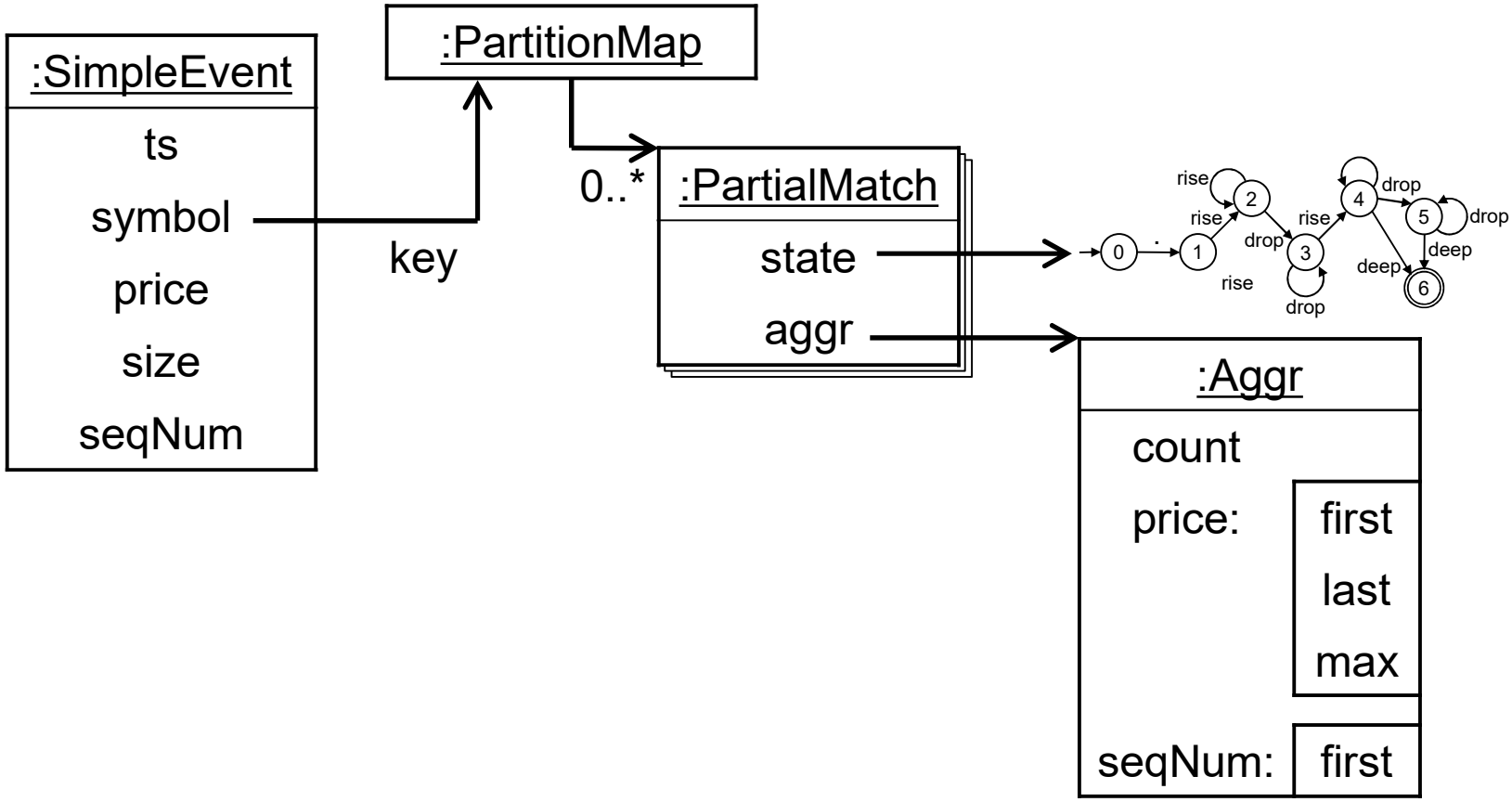

```
1  stream<MatchT> Matches = MatchRegex(Quotes) {
2   param
3    pattern     : ". rise+ drop+ rise+ drop* deep";
4    partitionBy : symbol;
5    predicates  : {
6     rise = price>First(price)  && price>=Last(price),
7     drop = price>=First(price) && price<Last(price),
8     deep = price<First(price)  && price<Last(price) };
9   output
10   Matches : symbol=symbol, seqNum=First(seqNum),
11             count=Count(), maxPrice=Max(price);
12 }
```

Automaton

```
. rise+ drop+ rise+ drop* deep
```

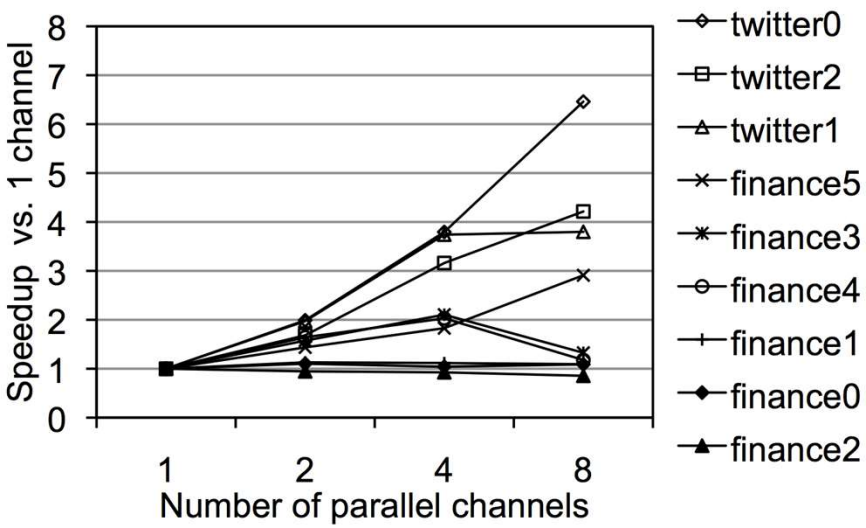


Data Structures

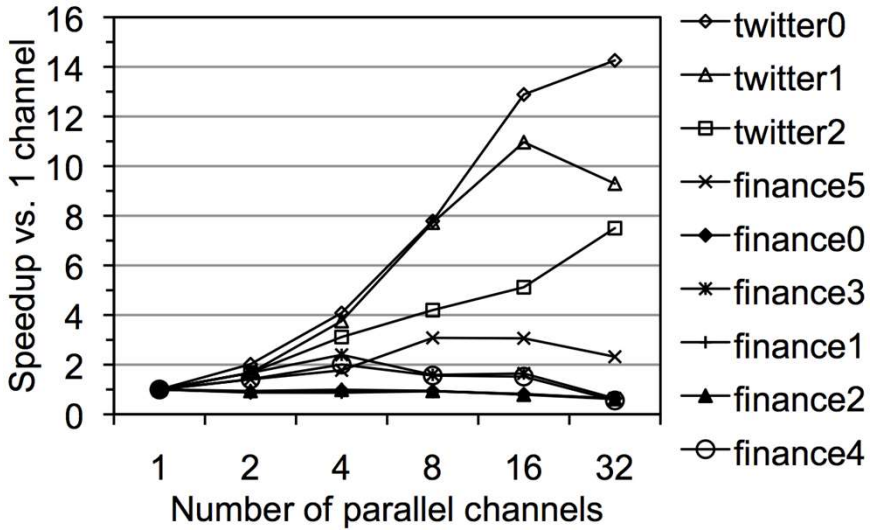


Speedups

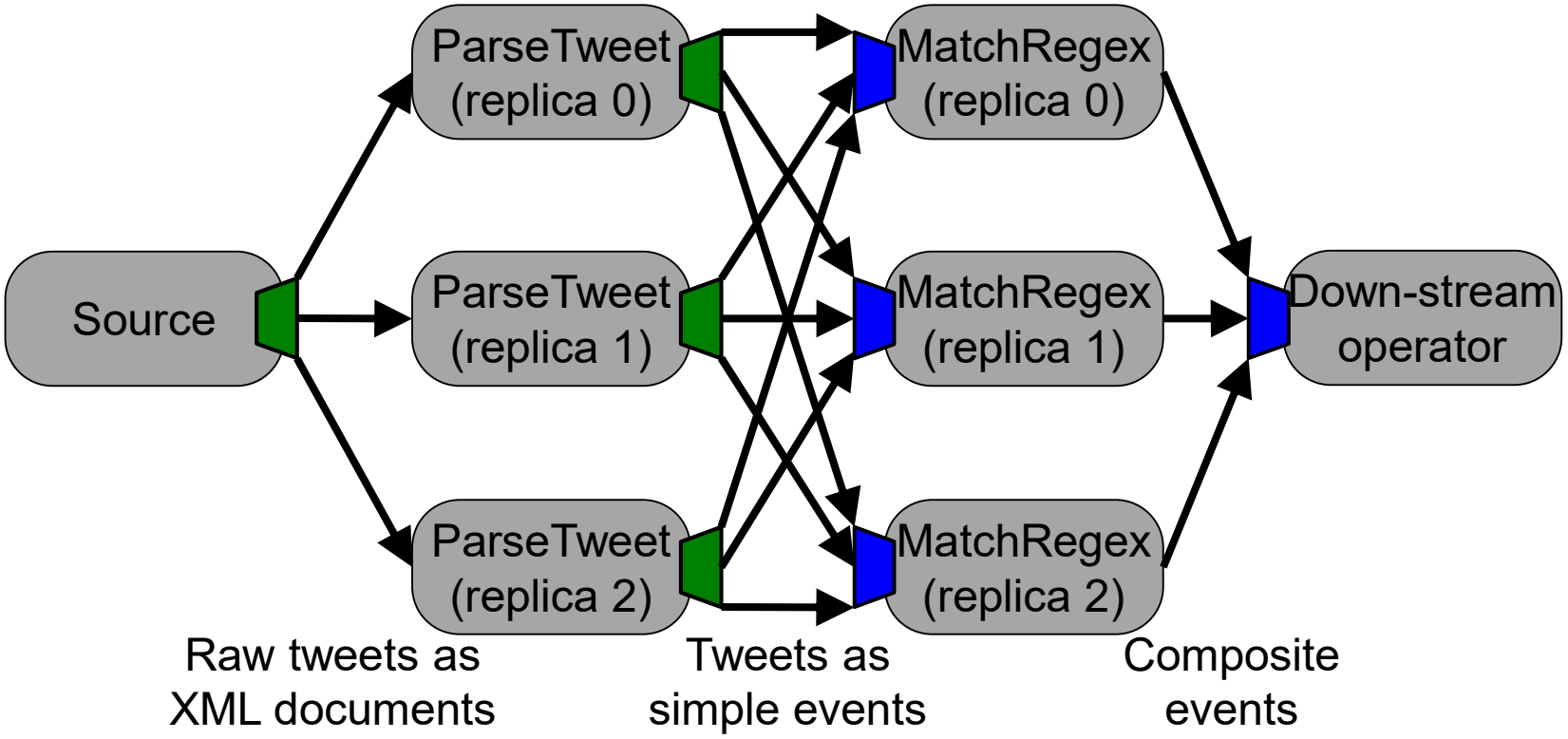
1 Machine x 8 Cores



4 Machines x 8 Cores = 32

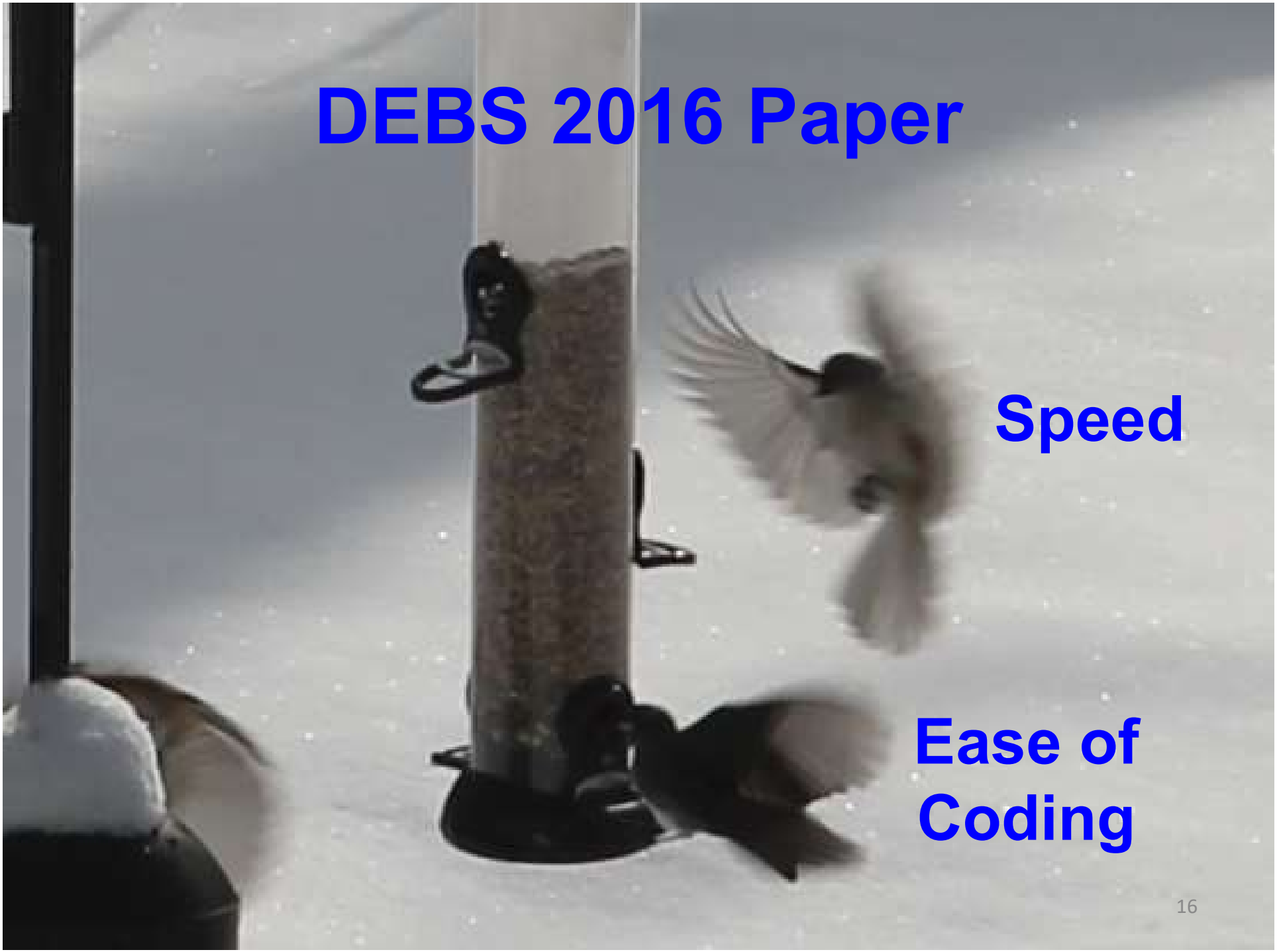


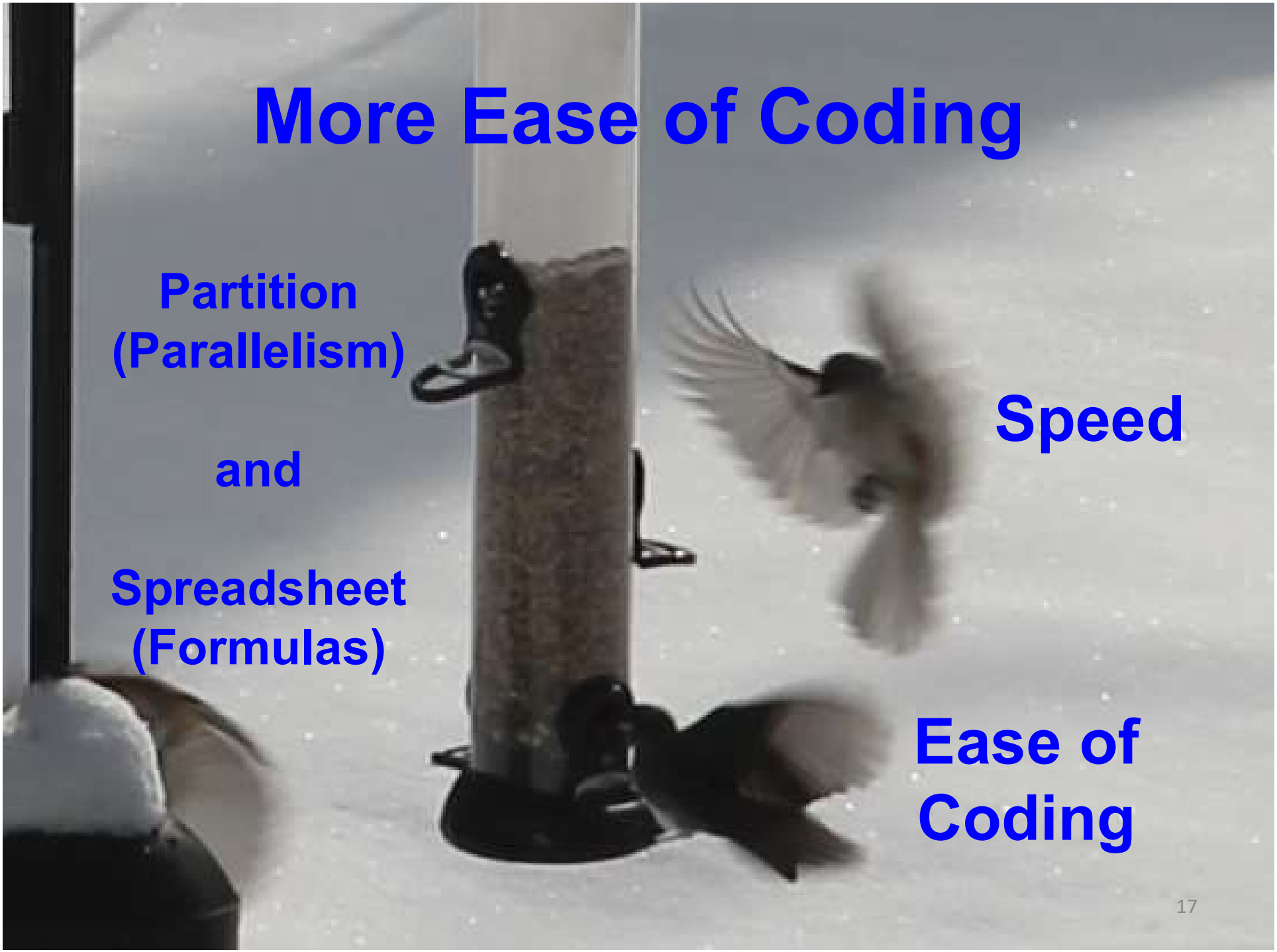
Shuffle in twitter02 and twitter03



Productization

- In IBM Streams product since 2012
- Library operator (no core runtime change)


DEBS 2016 Paper
Speed
Ease of Coding


More Ease of Coding
Partition
(Parallelism)
and
Spreadsheet
(Formulas)
Speed
Ease of
Coding

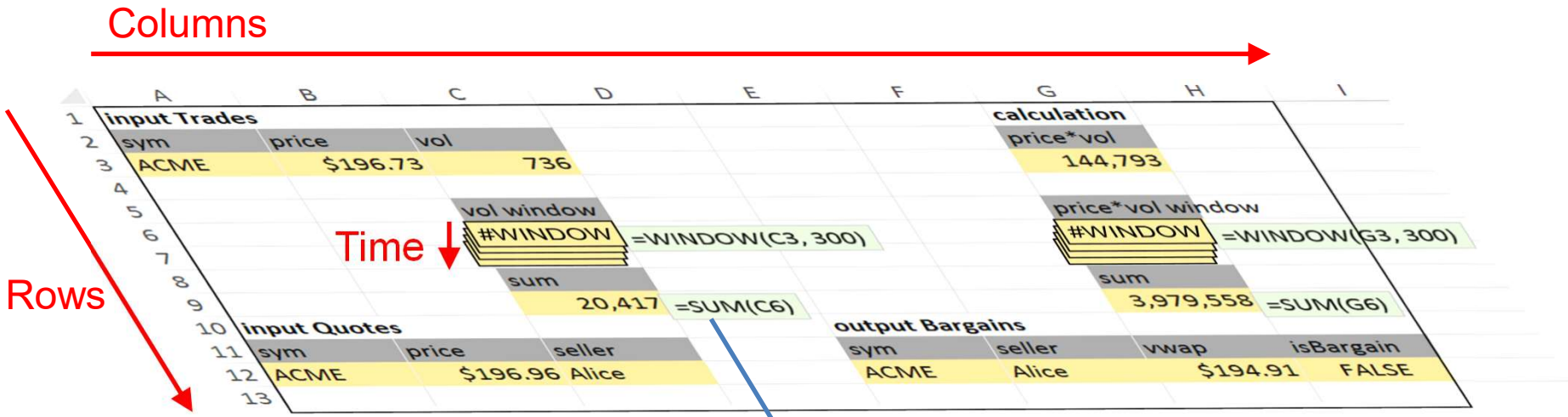
Streaming Spreadsheets

	A	B	C	D	E	F	G	H
1	**input Trades**				**line chart**		**calculation**	
2	sym	price	vol				price*vol	
3	ACME	$199.07	842				167,617	=B3*C3
4	ACME	$198.99	718				142,875	
5	ACME	$197.70	773				152,822	
6	ACME	$198.84	496				98,625	
7	ACME	$198.16	424				84,020	
8	ACME	$199.15	737				146,774	
9	ACME	$198.71	664				131,943	
10	ACME	$196.73	736				144,793	=B10*C10
11			sum				sum	
12			5,390	=SUM(C3:C10)			1,069,469	=SUM(G3:G10)
13	**input Quotes**				**output Bargains**			
14	sym	price	seller		sym	seller	vwap	isBargain
15	ACME	$196.96	Alice		ACME	Alice	$198.42	TRUE
					=A15	=C15	=G12/C12	=B15<G15

Scrolling

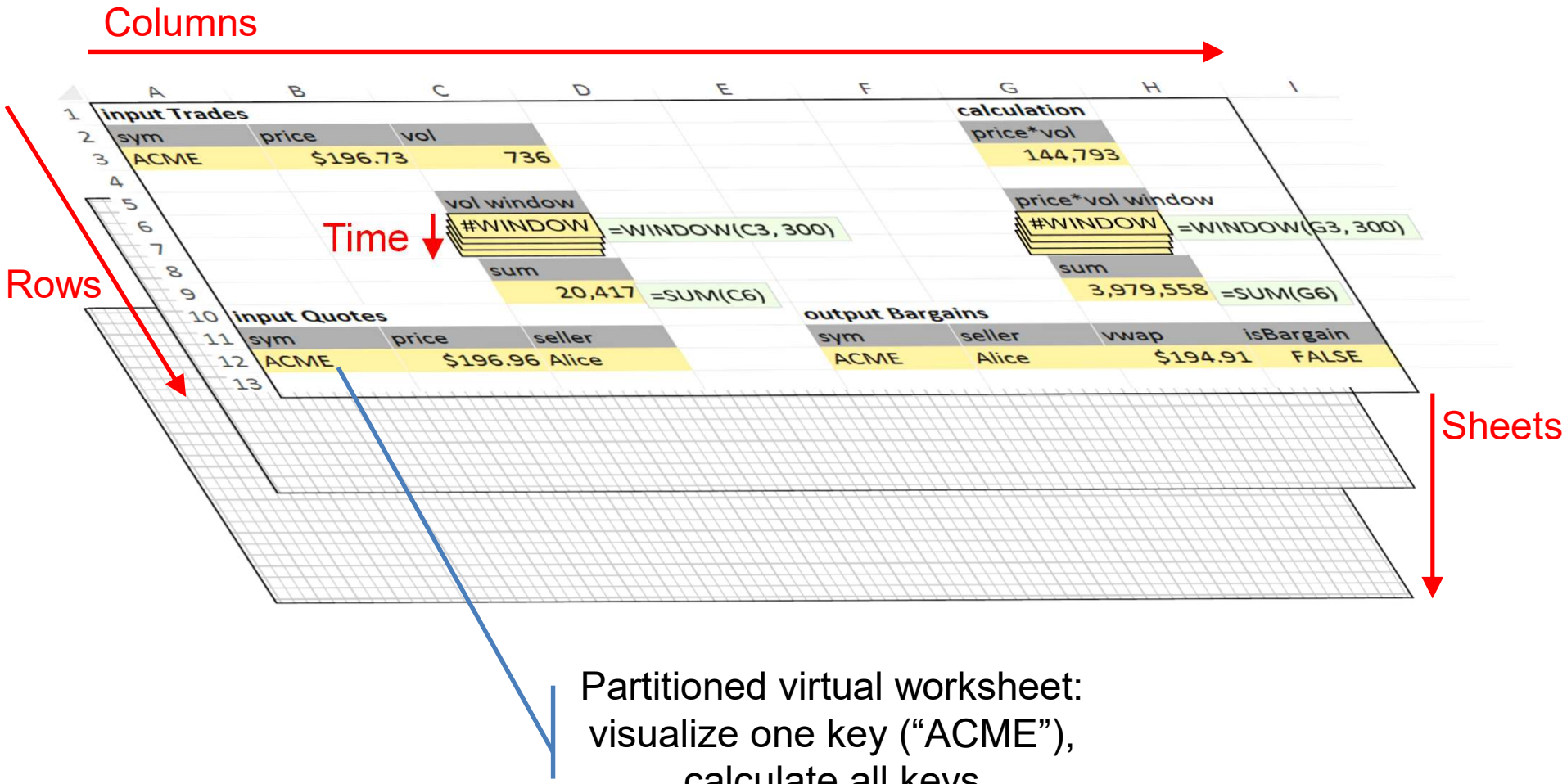
Scrolling

Time-Based Windows



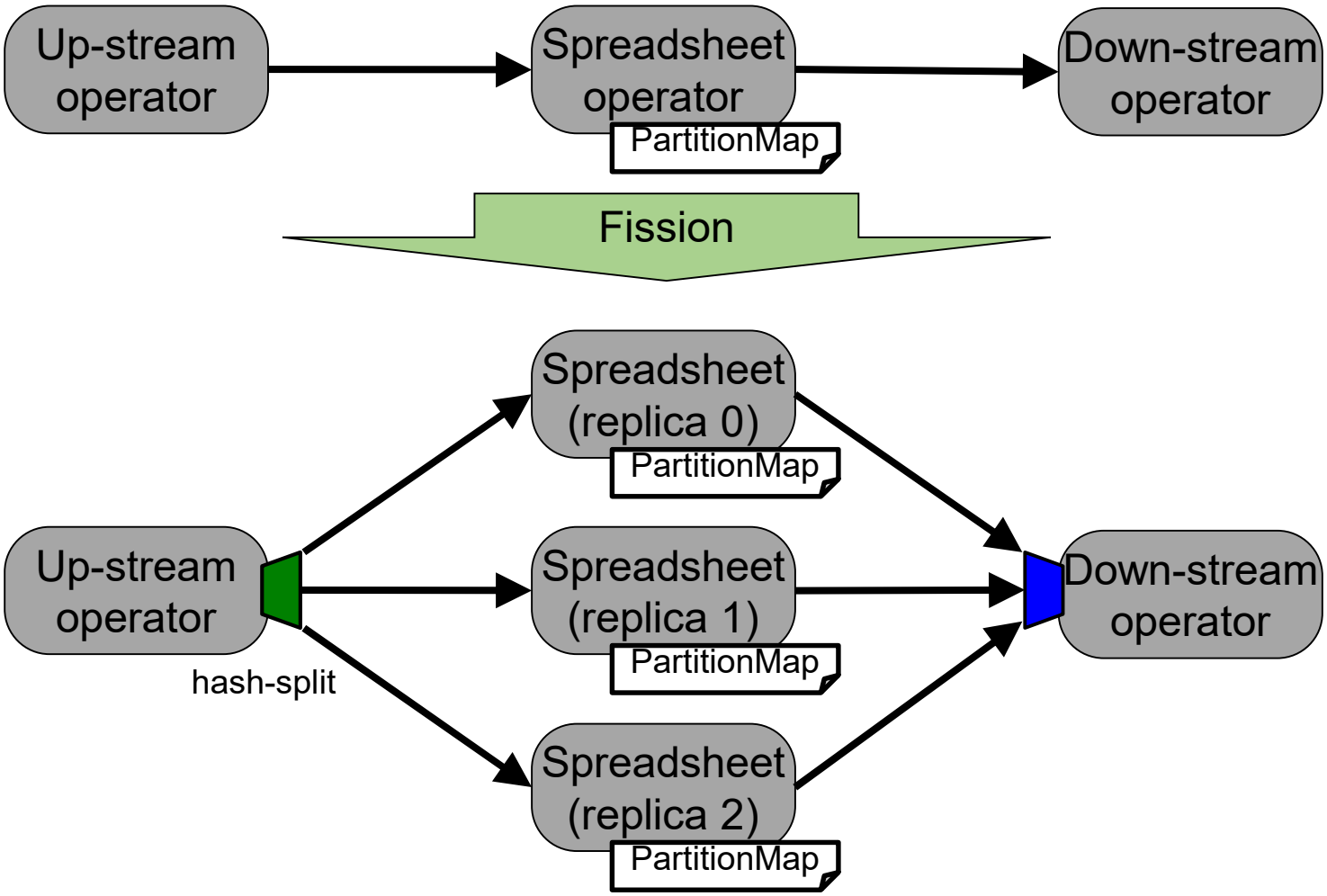
Variable-sized windows:
occupy one cell,
aggregate all elements

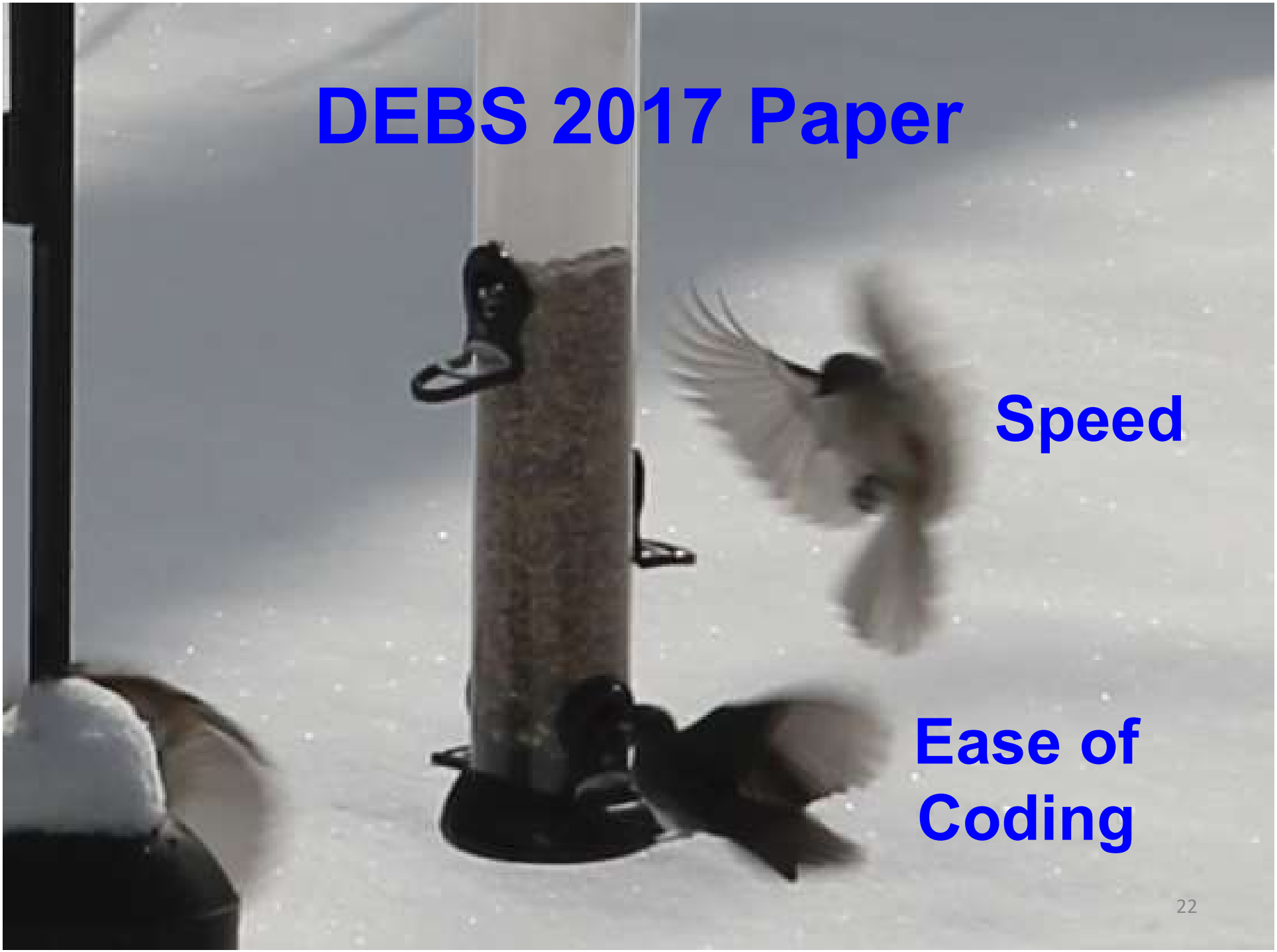
Partitioned Virtual Worksheets

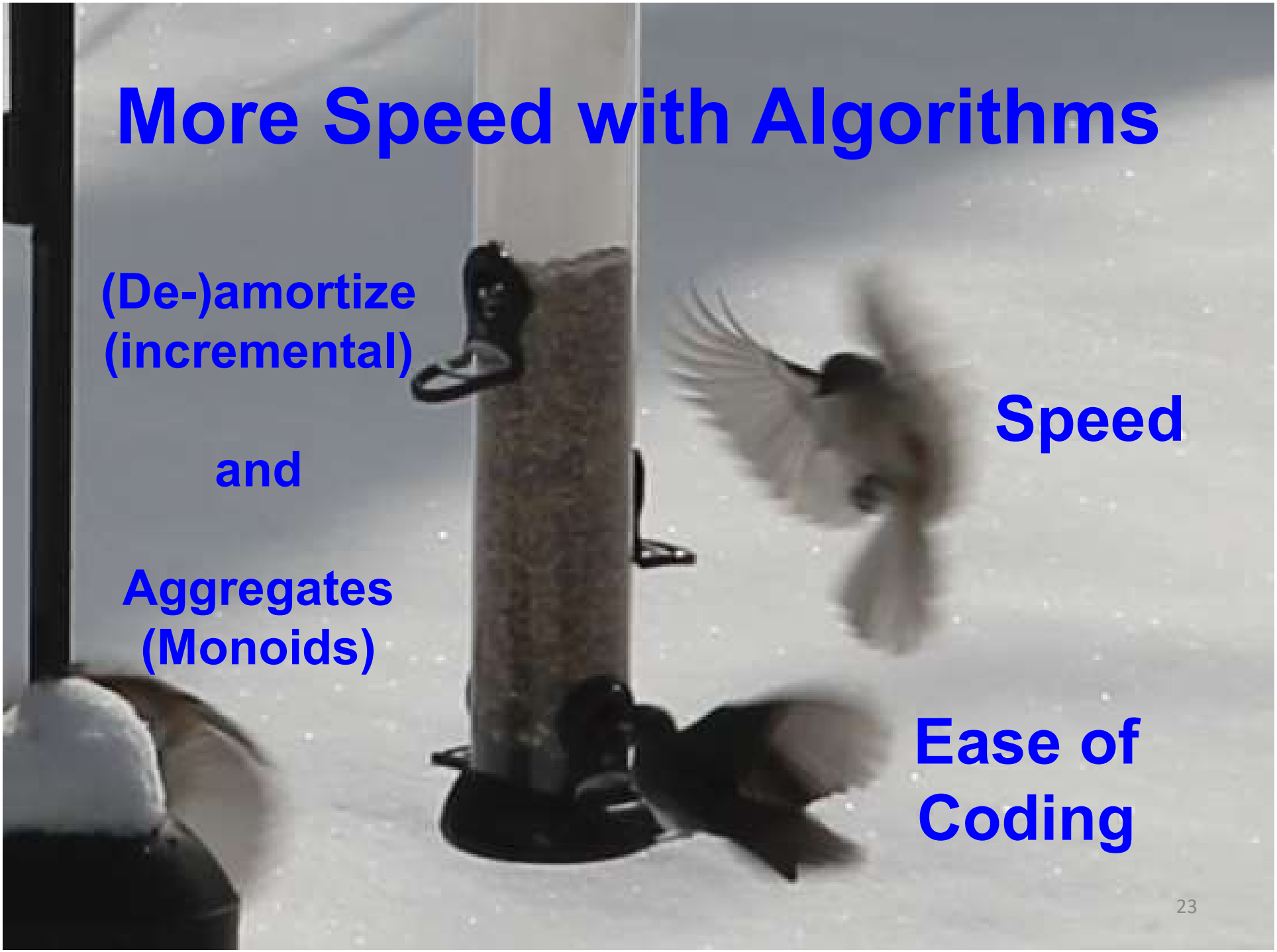


Partitioned virtual worksheet:
visualize one key (“ACME”),
calculate all keys

Partition Parallelism, Again




DEBS 2017 Paper
Speed
Ease of Coding

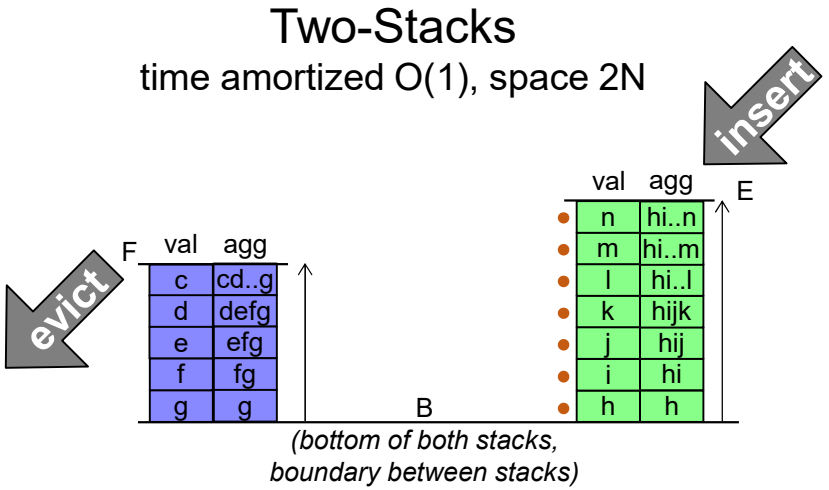

More Speed with Algorithms
(De-)amortize
(incremental)
and
Aggregates
(Monoids)
Speed
Ease of
Coding

Algorithm Selection for Incremental Aggregation

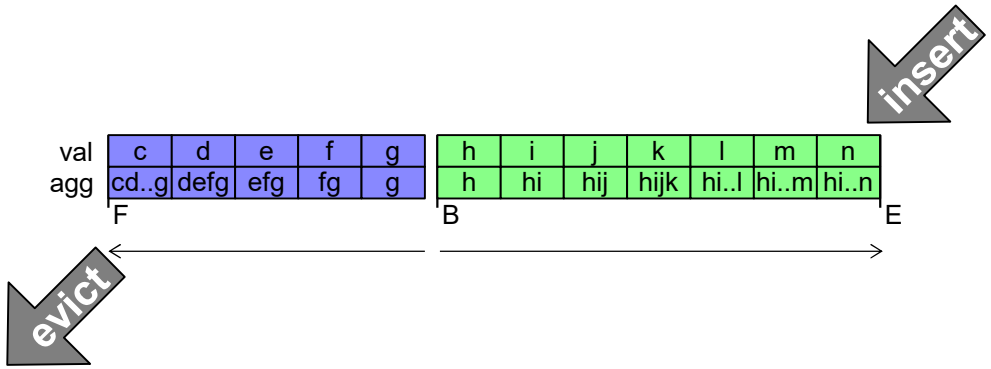
B 1 Batching Unchanged graph Stable semantics					**Fu** 2 Fusion Changed graph Stable semantics
P 3 Placement Unchanged graph Stable semantics	**Ss** 4 State sharing Unchanged graph Stable semantics		**Os** 5 Operator separation Changed graph Stable semantics	**Or** 6 Operator reordering Changed graph Stable semantics	**Re** 7 Redundancy elimination Changed graph Stable semantics
Lb 8 Load balancing Unchanged graph Stable semantics	**As** 9 Algorithm selection Unchanged graph Unstable semantics	**Ls** 10 Load shedding Unchanged graph Unstable semantics	**Fi** 11 Fission Changed graph Unstable semantics		

➔ ***exact (not approximate) for stable semantics***

Constant-Time Aggregation

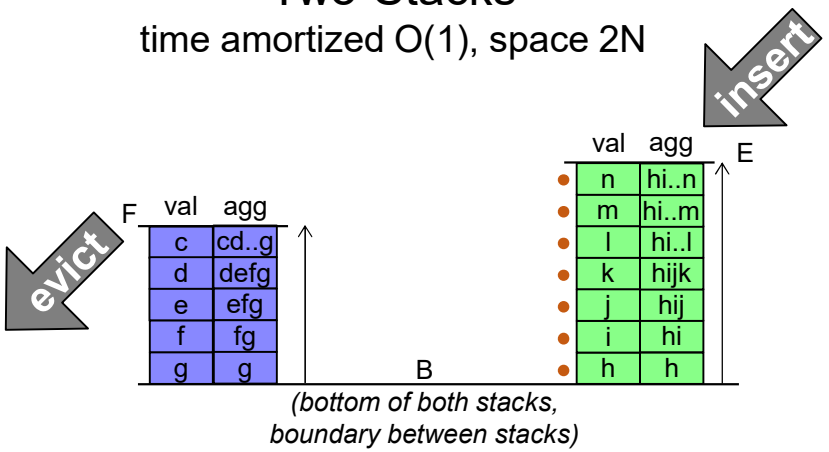


Emulate 2 Stacks with 1 Queue



Two-Stacks

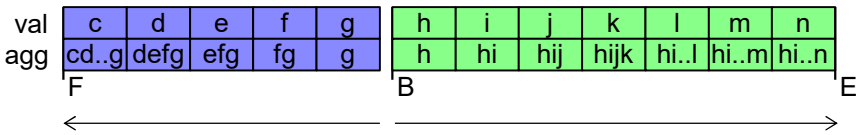
time amortized O(1), space 2N



De-Amortize Flip Operation

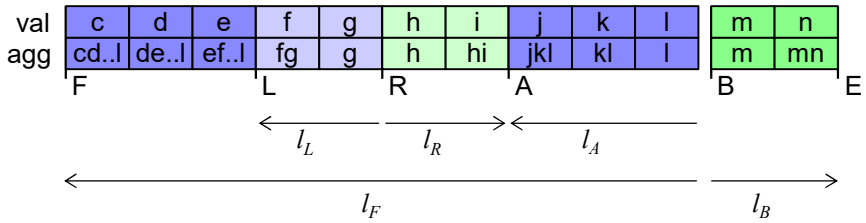
Two-Stacks

time amortized O(1), space 2N



DABA

time worst-case O(1), space 2N

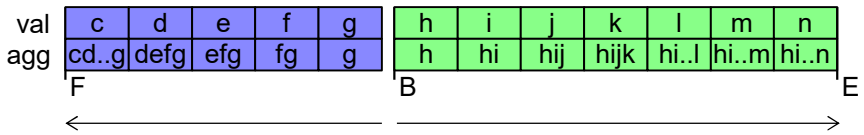


➔ ***DEBS 2017 paper***

Go Lite on Space

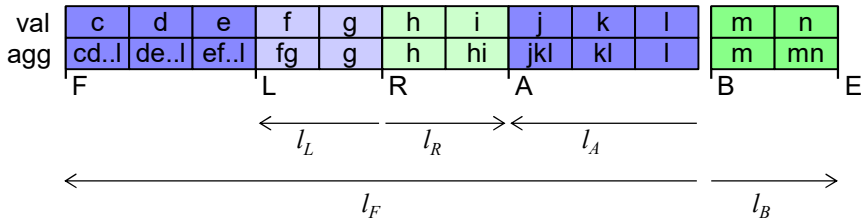
Two-Stacks

time amortized O(1), space 2N



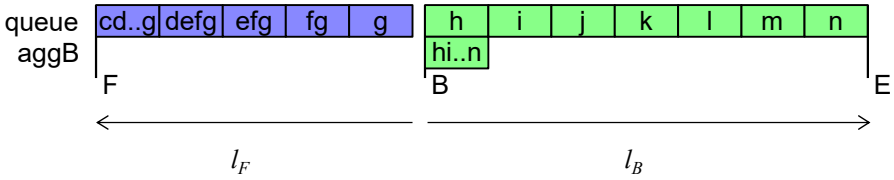
DABA

time worst-case O(1), space 2N



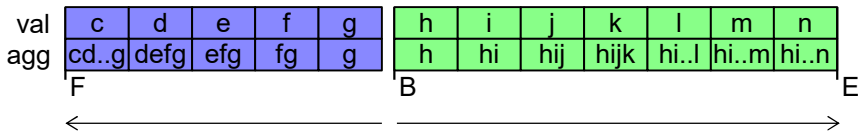
Two-Stacks Lite

time amortized O(1), space N+1

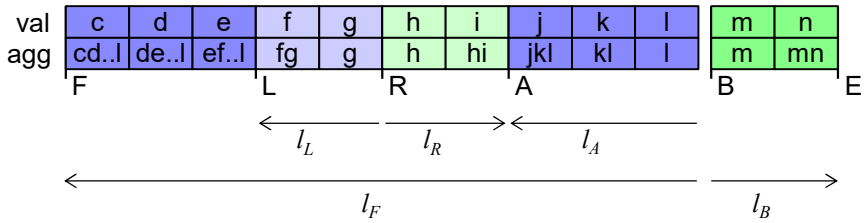


Both De-Amortized and Lite

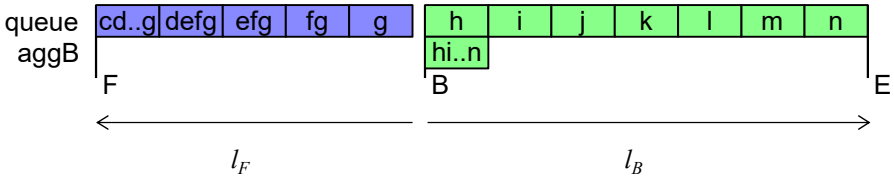
Two-Stacks
time amortized O(1), space 2N



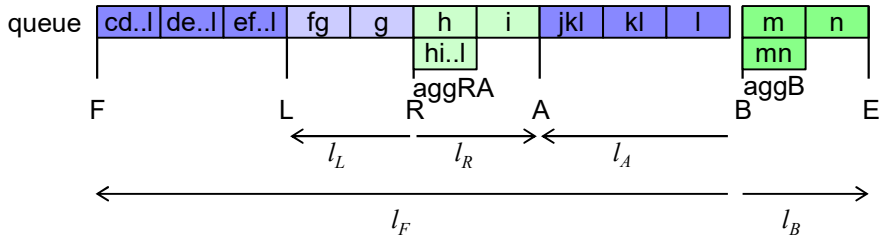
DABA
time worst-case O(1), space 2N



Two-Stacks Lite
time amortized O(1), space N+1



DABA Lite
time worst-case O(1), space N+2



➔ ***VLDB Journal 2021 paper***

Open-Source Repository

github.com/ibm/sliding-window-aggregators

To Learn More

- Tutorial: Stream Processing Optimizations.
  Scott Schneider, Buğra Gedik, Martin Hirzel. DEBS 2013.
- A Catalog of Stream Processing Optimizations. Martin Hirzel, Robert Soulé, Scott Schneider, Buğra Gedik, Robert Grimm. CSUR 2014.
- Spreadsheets for Stream Processing with Unbounded Windows and Partitions. Martin Hirzel, Rodric Rabbah, Philippe Suter, Olivier Tardieu, Mandana Vaziri. DEBS 2016.
- Low-Latency Sliding-Window Aggregation in Worst-Case Constant Time. Kanat Tangwongsan, Martin Hirzel, Scott Schneider. DEBS 2017.
- In-Order Sliding-Window Aggregation in Worst-Case Constant Time. Kanat Tangwongsan, Martin Hirzel, Scott Schneider. VLDB Journal 2021.

Thank You, Collaborators!

- Tutorial: Stream Processing Optimizations. Scott Schneider, Buğra Gedik, Martin Hirzel. DEBS 2013.
- A Catalog of Stream Processing Optimizations. Martin Hirzel, Robert Soulé, Scott Schneider, Buğra Gedik, Robert Grimm. CSUR 2014.
- Spreadsheets for Stream Processing with Unbounded Windows and Partitions. Martin Hirzel, Rodric Rabbah, Philippe Suter, Olivier Tardieu, Mandana Vaziri. DEBS 2016.
- Low-Latency Sliding-Window Aggregation in Worst-Case Constant Time. Kanat Tangwongsan, Martin Hirzel, Scott Schneider. DEBS 2017.
- In-Order Sliding-Window Aggregation in Worst-Case Constant Time. Kanat Tangwongsan, Martin Hirzel, Scott Schneider. VLDB Journal 2021.

Thank You, DEBS Community!

- Tutorial: Stream Processing Optimizations. Scott Schneider, Buğra Gedik, Martin Hirzel. DEBS 2013.
- A Catalog of Stream Processing Optimizations. Martin Hirzel, Robert Soulé, Scott Schneider, Buğra Gedik, Robert Grimm. CSUR 2014.
- Spreadsheets for Stream Processing with Unbounded Windows and Partitions. Martin Hirzel, Rodric Rabbah, Philippe Suter, Olivier Tardieu, Mandana Vaziri. DEBS 2016.
- Low-Latency Sliding-Window Aggregation in Worst-Case Constant Time. Kanat Tangwongsan, Martin Hirzel, Scott Schneider. DEBS 2017.
- In-Order Sliding-Window Aggregation in Worst-Case Constant Time. Kanat Tangwongsan, Martin Hirzel, Scott Schneider. VLDB Journal 2021.

Take-Home Messages

- Parallelism itself is a not a goal, but just one of several possible optimizations
- Partitioning is key to stable parallelism
- Aggregation is foundational to most event and stream programming models

Backup

Data sets …

Name	Type	Key	# Keys	# Events	Logical time
finance	Trade	symbol	390	10,000,000	2 h 01 min
twitter	Tweet	author	6,142	200,000	36 h 40 min

```
type Trade = tuple<
  timestamp ts, rstring symbol,
  uint32 price, uint32 size, uint32 seqNum>;
type Tweet = tuple<
  uint64 id, timestamp ts, rstring author,
  rstring content>;
```

… and benchmarks

Name	Pattern	Description	Selectivity	Topology
finance0	`largeSize priceRise+ priceDrop`	Large trade followed by peak	1.71 %	*MatchRegex* only
finance1	`. rise+ drop+ rise+ drop* deep`	M-shape (double top)	0.31 %	*MatchRegex* only
finance2	`. rise* riseEnd flat* flatEnd`	Rise then flat with time window	2.72 %	*MatchRegex* only
finance3	`divergence`	Price substantially above VWAP	0.03 %	*VWAP* → *MatchRegex*
finance4	`hi gap* lo`	Max of hi smaller than min of lo	5.15 %	*MinMax* → *MatchRegex*
finance5	`. notTooLong* largeIncrease`	Large increase with time window	0.04 %	*MatchRegex* only
twitter0	(None)	Parse tweet only, no matching	100.00 %	*ParseTweet* only
twitter1	`. sameTags+ sameTags5th`	Five tweets with identical tags	14.07 %	*ParseTweet* → *MatchRegex*
twitter2	`.+ disjointTags`	Different first vs. last tags	2.15 %	*ParseTweet* → *MatchRegex*