

# Low-Latency Sliding-Window Aggregation in Worst-Case Constant Time

Kanat Tangwongsan

Mahidol University International College

[kanat.tan@mahidol.edu](mailto:kanat.tan@mahidol.edu)

Martin Hirzel

IBM Research

[hirzel@us.ibm.com](mailto:hirzel@us.ibm.com)

Scott Schneider

IBM Research

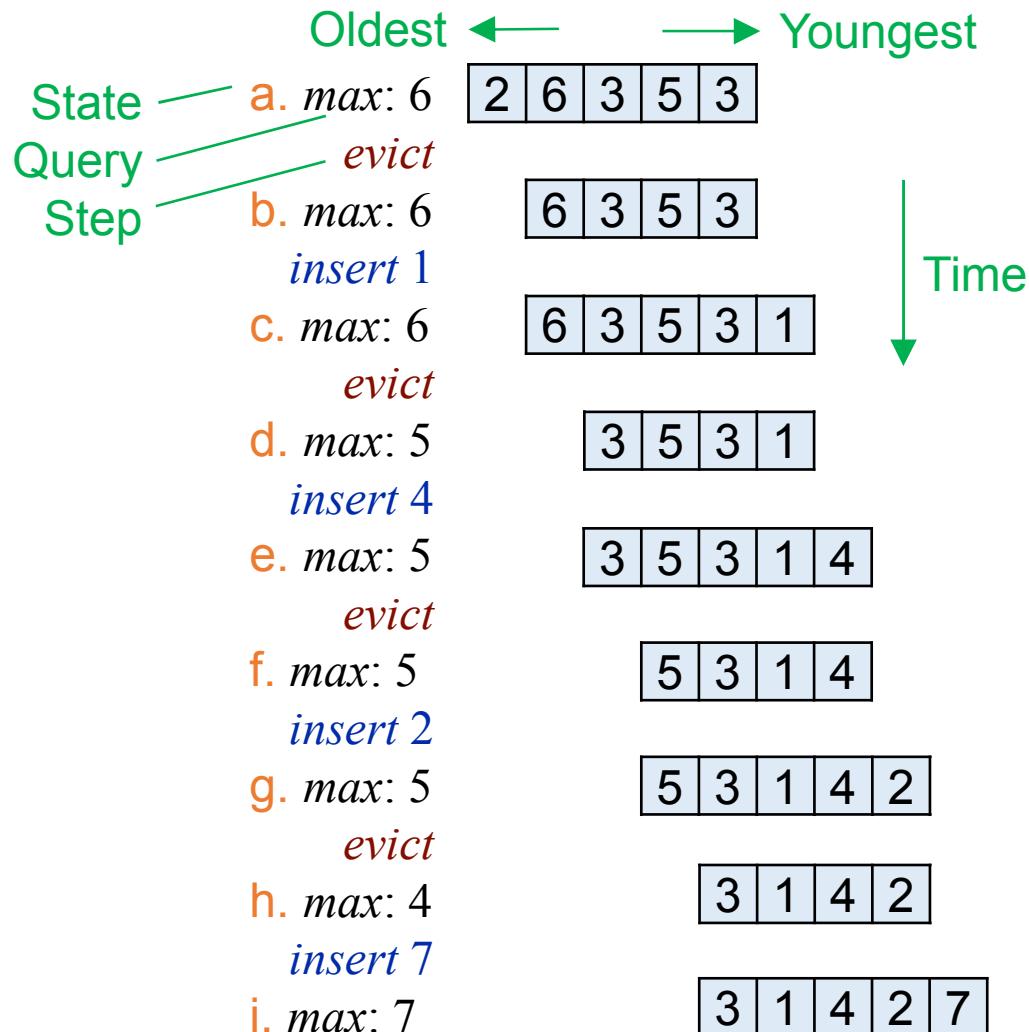
[scott.a.s@us.ibm.com](mailto:scott.a.s@us.ibm.com)

DEBS 2017

# Title Explained

What	Why
Low-Latency	QoS or Real-time
Sliding-Window	Recent $\simeq$ relevant
Aggregation	Stream “reduce”
Worst-Case	No latency spikes
Constant Time	$O(1)$ better than $O(\log n)$

# Running Example



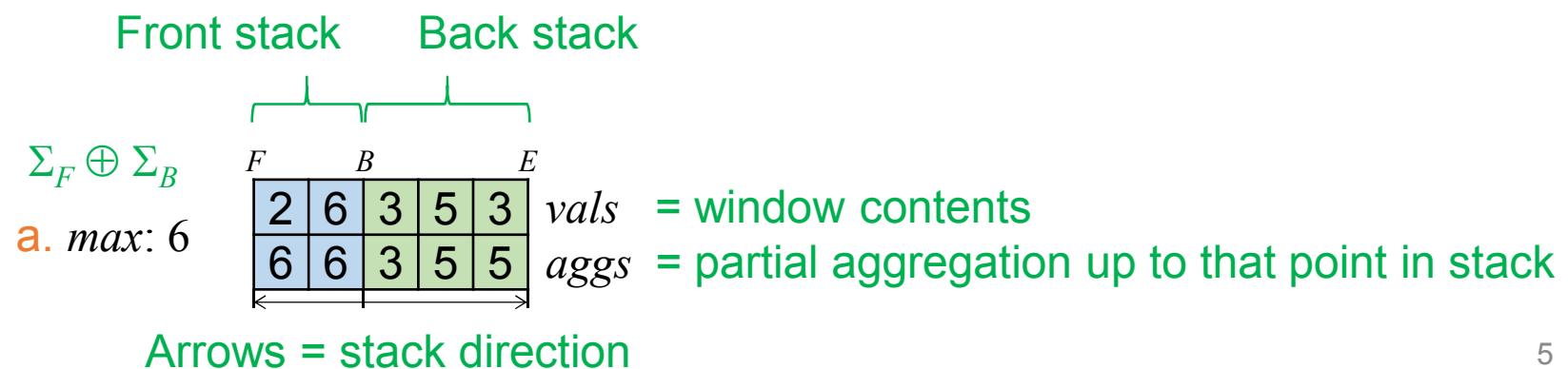
In general:

- Any associative aggregation operation (not just max ⇒ sum, geoMean, Bloom, ...)
- Any interleaving of insert and evict (not just alternating ⇒ variable-sized windows)

# Two-Stacks Algorithm

What	Why
Low-Latency	QoS or Real-Time
Sliding-Window	Recent $\simeq$ relevant
Aggregation	Stream “reduce”
<del>Amortized Worst Case</del>	<del>Some</del> No latency spikes
Constant Time	$O(1)$ better than $O(\log n)$

# Two-Stacks Example



# Two-Stacks Example

a. max: 6

*evict*

b. max: 6

F	B		E	
2	6	3	5	3
6	6	3	5	5

vals  
aggs

F	B		E	
6	3	5	3	
6	3	5	5	

*F.pop()*  
*// O(1) time*

# Two-Stacks Example

a.  $\max: 6$

*evict*

b.  $\max: 6$

*insert 1*

c.  $\max: 6$

	F	B		E	
vals	2	6	3	5	3
aggs	6	6	3	5	5

$\leftarrow \rightarrow$

	F	B		E	
vals	6	3	5	3	
aggs	6	3	5	5	

$\leftarrow \rightarrow$

	F	B		E	
vals	6	3	5	3	1
aggs	6	3	5	5	5

$\leftarrow \rightarrow$

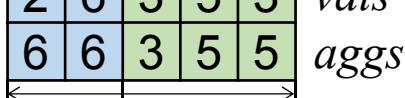
*B.push( $v, \Sigma_B \oplus v$ )*  
*//  $O(1)$  time*

# Two-Stacks Example

a. max: 6

F	B		E	
2	6	3	5	3
6	6	3	5	5

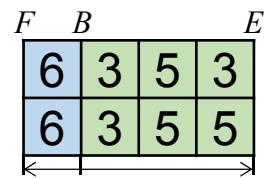
vals  
aggs



*evict*

b. max: 6

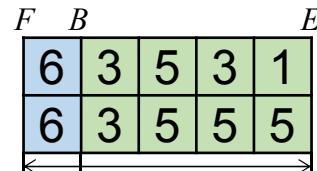
F	B		E	
6	3	5	3	
6	3	5	5	



*insert 1*

c. max: 6

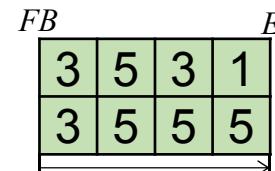
F	B		E	
6	3	5	3	1
6	3	5	5	5



*evict*

d. max: 5

FB		E	
3	5	3	1
3	5	5	5



# Two-Stacks Example

	F	B	E	
a. max: 6	2	6	3	5
	6	6	3	5

vals  
aggs

evict

	F	B	E	
b. max: 6	6	3	5	3
	6	3	5	5

insert 1

	F	B	E	
c. max: 6	6	3	5	3
	6	3	5	5

evict

	FB	E	
d. max: 5	3	5	3
	3	5	5

insert 4

	FB	E	
e. max: 5	3	5	3
	3	5	5

# Two-Stacks Example

b. max: 6

F	B		E
6	3	5	3
6	3	5	5

insert 1

c. max: 6

F	B		E
6	3	5	3
6	3	5	5

evict

d. max: 5

FB			E
3	5	3	1
3	5	5	5

insert 4

e. max: 5

FB			E
3	5	3	1
3	5	5	5

flip

f1.

F			BE
3	5	3	1
5	5	4	4

**if**  $F.isEmpty()$

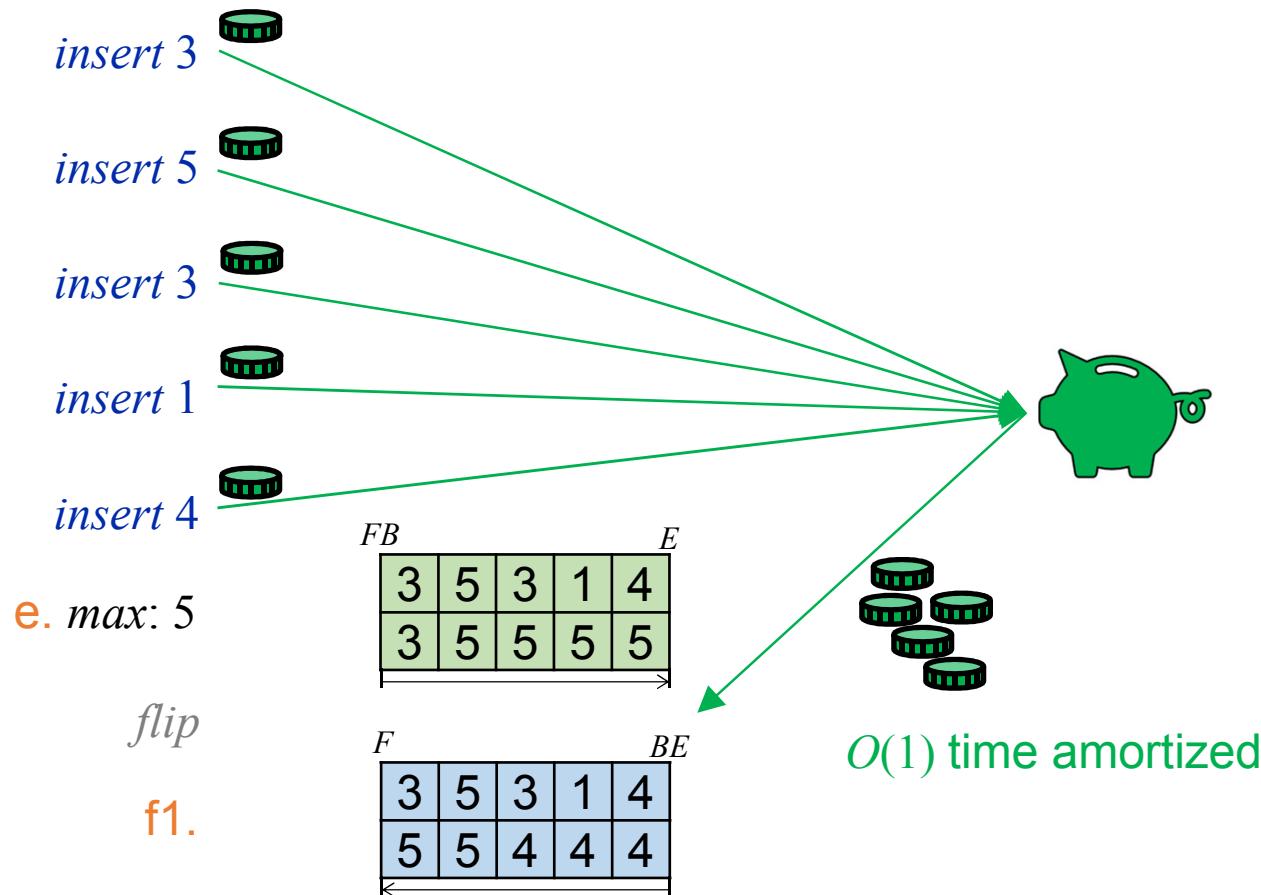
**while not**  $B.isEmpty()$

$F.push(B.top().val, B.top().val \oplus \Sigma_F)$

$B.pop()$

$// O(n)$  time, latency spike

# Banker's Amortized Analysis



# Two-Stacks Example

c. max: 6

F	B		E
6	3	5	3
6	3	5	5

*evict*

d. max: 5

FB		E
3	5	3
3	5	5

*insert 4*

e. max: 5

FB		E
3	5	3
3	5	5

*flip*

f1.

*evict*

f. max: 5

F		BE
3	5	3
5	5	4

F		BE
5	3	1
5	4	4

# Two-Stacks Example

d. max: 5

FB				E
3	5	3	1	
3	5	5	5	

insert 4

e. max: 5

FB				E
3	5	3	1	4
3	5	5	5	5

flip

f1.

F				BE
3	5	3	1	4
5	5	4	4	4

evict

f. max: 5

F				BE
5	3	1	4	
5	4	4	4	

insert 2

g. max: 5

F			B	E
5	3	1	4	2
5	4	4	4	2

# Two-Stacks Example

e. max: 5

<i>FB</i>				<i>E</i>
3	5	3	1	4
3	5	5	5	5

*flip*

f1.

<i>F</i>				<i>BE</i>
3	5	3	1	4
5	5	4	4	4

*evict*

f. max: 5

<i>F</i>				<i>BE</i>
5	3	1	4	
5	4	4	4	

*insert 2*

g. max: 5

<i>F</i>			<i>B</i>	<i>E</i>
5	3	1	4	2
5	4	4	4	2

*evict*

h. max: 4

<i>F</i>			<i>B</i>	<i>E</i>
3	1	4	2	
4	4	4	2	

# Two-Stacks Example

f1.

F				BE
3	5	3	1	4
5	5	4	4	4

*evict*

f. max: 5

F				BE
5	3	1	4	
5	4	4	4	

*insert 2*

g. max: 5

F			B	E
5	3	1	4	2
5	4	4	4	2

*evict*

h. max: 4

F			B	E
3	1	4	2	
4	4	4	2	

*insert 7*

i. max: 7

F			B	E
3	1	4	2	7
4	4	4	2	7

# Beyond Two-Stacks

Two-Stacks is  $O(1)$ , but only amortized.

Want  $O(1)$  worst-case to prevent latency spikes.

# Beyond Two-Stacks

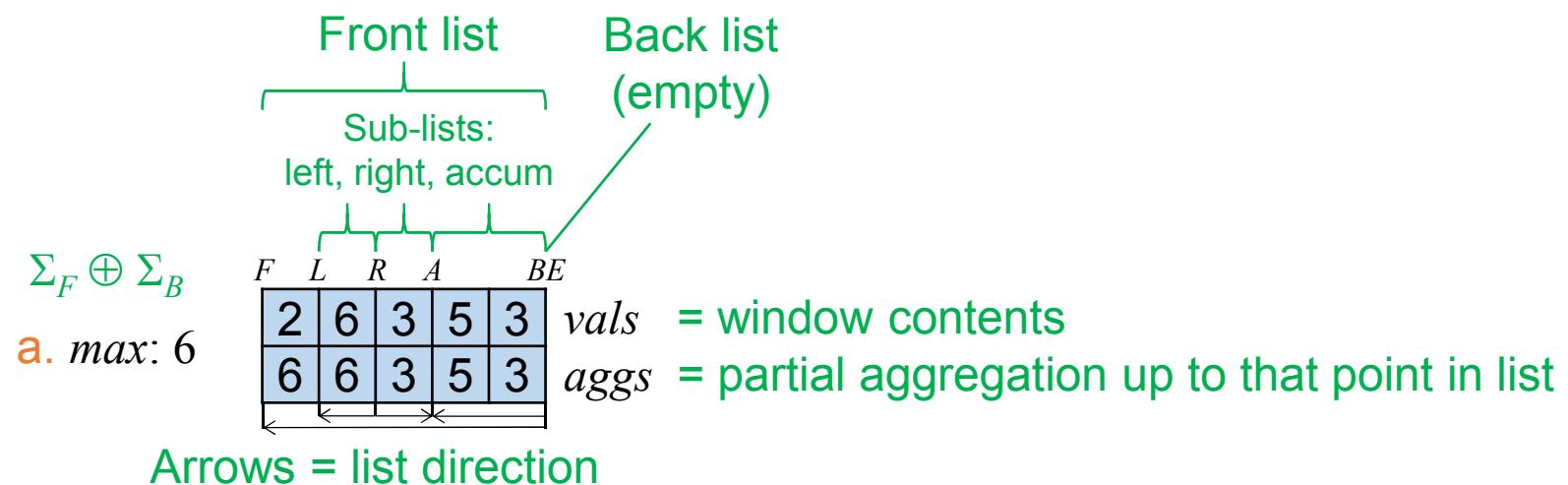
Two-Stacks is  $O(1)$ , but only amortized.

Want  $O(1)$  worst-case to prevent latency spikes.

Okasaki [JFP'95]: FIFO queue in  
 $O(1)$  worst-case, but no aggregation.

DABA (De-Amortized Banker's Aggregator)  
inspired by Okasaki's, supports aggregation.

# DABA Example



# DABA Example

a. max: 6

*evict*

b1.

F	L	R	A	BE
2	6	3	5	3
6	6	3	5	3

*vals*

*aggs*

```
// evict  
vals.popFront()  
aggs.popFront()
```

FL	R	A	BE
6	3	5	3
6	3	5	3

# DABA Example

a. max: 6

*evict*

b1.

*shrink*

b. max: 6

F	L	R	A	BE	
2	6	3	5	3	<i>vals</i>
6	6	3	5	3	<i>aggs</i>

F	R	A	BE	
6	3	5	3	
6	3	5	3	

F	LRA	BE	
6	3	5	3
6	5	5	3

```
// evict
vals.popFront()
aggs.popFront()
// fixup → shrink case
aggs[L] ←  $\Sigma_L \oplus \Sigma_R \oplus \Sigma_A$ 
L ← L + 1
aggs[A - 1] ← vals[A - 1]  $\oplus \Sigma_A$ 
A ← A - 1
// O(1) time
```

# DABA Example

a. max: 6

*evict*

b1.

*shrink*

b. max: 6

*insert 1*

c1.

F	L	R	A	BE	vals
2	6	3	5	3	
6	6	3	5	3	aggs

F	LRA	BE
6	3	5
6	3	5

F	LRA	BE
6	3	5
6	5	5

F	LRA	B	E
6	3	5	3
6	5	5	3

// insert

*vals.pushBack(v)*

*aggs.pushBack( $\Sigma_B \oplus v$ )*

# DABA Example

a. max: 6

*evict*

F	L	R	A	BE
2	6	3	5	3
6	6	3	5	3

b1.

*shrink*

b. max: 6

*insert 1*

c1.

*shift*

c. max: 6

F	LRA	BE	
6	3	5	3
6	3	5	3

F	LRA	BE	
6	3	5	3
6	5	5	3

F	LRA	B	E	
6	3	5	3	1
6	5	5	3	1

F	LRA	B	E	
6	3	5	3	1
6	5	5	3	1

```
// insert
vals.pushBack(v)
aggs.pushBack( $\Sigma_B \oplus v$ )
// fixup → shift case
A ← A + 1
R ← R + 1
L ← L + 1
// O(1) time
```

# DABA Example

b1.

<i>F</i>	<i>LRA</i>	<i>B</i>	<i>E</i>
6	3	5	3
6	3	5	3

*shrink*

b. *max: 6*

<i>F</i>	<i>LRA</i>	<i>B</i>	<i>E</i>
6	3	5	3
<b>6</b>	<b>5</b>	5	3

*insert 1*

c1.

<i>F</i>	<i>LRA</i>	<i>B</i>	<i>E</i>	
6	3	5	3	1
6	5	5	3	<b>1</b>

*shift*

c. *max: 6*

<i>F</i>	<i>LRA</i>	<i>B</i>	<i>E</i>	
6	3	5	3	1
6	5	5	3	<b>1</b>

*evict*

*shift*

d. *max: 5*

<i>F</i>	<i>LRA</i>	<i>B</i>	<i>E</i>
3	5	3	1
5	5	3	1

# DABA Example

b. max: 6

*insert 1*

c1.

*shift*

c. max: 6

*evict*

*shift*

d. max: 5

*insert 4*

*shift*

e. max: 5

F	LRA	BE	
6	3	5	3
6	5	5	3

F	LRA	B	E
6	3	5	3
6	5	5	3

F	LRA	B	E
6	3	5	3
6	5	5	3

F	LRA	B	E
3	5	3	1
5	5	3	1

F	LRA	B	E
3	5	3	1
5	5	3	1

# DABA Example

c1.	<table border="1"><tr><td>F</td><td>LRA</td><td>B</td><td>E</td></tr><tr><td>6</td><td>3</td><td>5</td><td>3</td><td>1</td></tr><tr><td>6</td><td>5</td><td>5</td><td>3</td><td>1</td></tr></table>	F	LRA	B	E	6	3	5	3	1	6	5	5	3	1
F	LRA	B	E												
6	3	5	3	1											
6	5	5	3	1											
<i>shift</i>	<table border="1"><tr><td>F</td><td>LRA</td><td>B</td><td>E</td></tr><tr><td>6</td><td>3</td><td>5</td><td>3</td><td>1</td></tr><tr><td>6</td><td>5</td><td>5</td><td>3</td><td>1</td></tr></table>	F	LRA	B	E	6	3	5	3	1	6	5	5	3	1
F	LRA	B	E												
6	3	5	3	1											
6	5	5	3	1											
c. max: 6	<table border="1"><tr><td>F</td><td>LRA</td><td>B</td><td>E</td></tr><tr><td>6</td><td>3</td><td>5</td><td>3</td><td>1</td></tr><tr><td>6</td><td>5</td><td>5</td><td>3</td><td>1</td></tr></table>	F	LRA	B	E	6	3	5	3	1	6	5	5	3	1
F	LRA	B	E												
6	3	5	3	1											
6	5	5	3	1											
<i>evict</i>	<table border="1"><tr><td>F</td><td>LRA</td><td>B</td><td>E</td></tr><tr><td>6</td><td>3</td><td>5</td><td>3</td><td>1</td></tr><tr><td>6</td><td>5</td><td>5</td><td>3</td><td>1</td></tr></table>	F	LRA	B	E	6	3	5	3	1	6	5	5	3	1
F	LRA	B	E												
6	3	5	3	1											
6	5	5	3	1											
d. max: 5	<table border="1"><tr><td>F</td><td>LRA</td><td>B</td><td>E</td></tr><tr><td>3</td><td>5</td><td>3</td><td>1</td><td>1</td></tr><tr><td>5</td><td>5</td><td>3</td><td>1</td><td>1</td></tr></table>	F	LRA	B	E	3	5	3	1	1	5	5	3	1	1
F	LRA	B	E												
3	5	3	1	1											
5	5	3	1	1											
<i>insert 4</i>	<table border="1"><tr><td>F</td><td>LRA</td><td>B</td><td>E</td></tr><tr><td>3</td><td>5</td><td>3</td><td>1</td><td>1</td></tr><tr><td>5</td><td>5</td><td>3</td><td>1</td><td>1</td></tr></table>	F	LRA	B	E	3	5	3	1	1	5	5	3	1	1
F	LRA	B	E												
3	5	3	1	1											
5	5	3	1	1											
<i>shift</i>	<table border="1"><tr><td>F</td><td>LRAB</td><td>E</td></tr><tr><td>3</td><td>5</td><td>3</td><td>1</td><td>4</td></tr><tr><td>5</td><td>5</td><td>3</td><td>1</td><td>4</td></tr></table>	F	LRAB	E	3	5	3	1	4	5	5	3	1	4	
F	LRAB	E													
3	5	3	1	4											
5	5	3	1	4											
e. max: 5	<table border="1"><tr><td>F</td><td>LRAB</td><td>E</td></tr><tr><td>3</td><td>5</td><td>3</td><td>1</td><td>4</td></tr><tr><td>5</td><td>5</td><td>3</td><td>1</td><td>4</td></tr></table>	F	LRAB	E	3	5	3	1	4	5	5	3	1	4	
F	LRAB	E													
3	5	3	1	4											
5	5	3	1	4											
<i>evict</i>	<table border="1"><tr><td>F</td><td>LRAB</td><td>E</td></tr><tr><td>5</td><td>3</td><td>1</td><td>4</td><td>1</td></tr><tr><td>5</td><td>3</td><td>1</td><td>4</td><td>1</td></tr></table>	F	LRAB	E	5	3	1	4	1	5	3	1	4	1	
F	LRAB	E													
5	3	1	4	1											
5	3	1	4	1											
f1.	<table border="1"><tr><td>F</td><td>LRAB</td><td>E</td></tr><tr><td>5</td><td>3</td><td>1</td><td>4</td><td>1</td></tr><tr><td>5</td><td>3</td><td>1</td><td>4</td><td>1</td></tr></table>	F	LRAB	E	5	3	1	4	1	5	3	1	4	1	
F	LRAB	E													
5	3	1	4	1											
5	3	1	4	1											

```
// evict  
vals.popFront()  
aggs.popFront()
```

# DABA Example

c. max: 6

*evict*

*shift*

d. max: 5

*insert 4*

*shift*

e. max: 5

*evict*

f1.

*flip*

f2.

F	LR A	B	E
6	3	5	3
6	5	5	3

F	LR A	B	E
3	5	3	1
5	5	3	1

F	LRAB	E
3	5	3
5	5	3

F	LRAB	E
5	3	1
5	3	1

FL	R	ABE
5	3	1
5	3	1

// evict

*vals.popFront()*

*aggs.popFront()*

// fixup → flip case

*L ← F*

*A ← E*

*B ← E*

# DABA Example

d. max: 5

*insert 4*

*shift*

e. max: 5

*evict*

f1.

*flip*

f2.

*shrink*

f. max: 5

F	LR	A	B	E
3	5	3	1	
5	5	3	1	

F	LRAB	E
3	5	3
5	5	3

F	LRAB	E
5	3	1
5	3	1

FL	R	ABE
5	3	1
5	3	1

F	L	R	A	BE
5	3	1	4	
5	3	1	4	

// evict

*vals.popFront()*

*aggs.popFront()*

// fixup → flip case

$L \leftarrow F$

$A \leftarrow E$

$B \leftarrow E$

// fixup → shrink case

$aggs[L] \leftarrow \Sigma_L \oplus \Sigma_R \oplus \Sigma_A$

$L \leftarrow L + 1$

$aggs[A - 1] \leftarrow vals[A - 1] \oplus \Sigma_A$

$A \leftarrow A - 1$

//  $O(1)$  time

# DABA Example

e. max: 5

*evict*

F	LRAB				E
3	5	3	1	4	
5	5	3	1	4	

f1.

*flip*

F	LRAB				E
5	3	1	4		
5	3	1	4		

f2.

*shrink*

FL	R	ABE		
5	3	1	4	
5	3	1	4	

f. max: 5

*insert 2*

*shrink*

g. max: 5

F	L	R	A	BE
5	3	1	4	
5	3	1	4	

F	LRA				B	E
5	3	1	4	2		
5	4	4	4	2		

# DABA Example

f1.

*flip*

F	LRAB	E
5	3	1
5	3	1

f2.

*shrink*

FL	R	ABE
5	3	1
5	3	1

f. *max: 5*

*insert 2*

*shrink*

g. *max: 5*

*evict*

*shift*

h. *max: 4*

F	L	R	A	BE
5	3	1	4	
5	3	1	4	

F	LR A	B	E
5	3	1	4
5	4	4	2

F	LRA	B	E
3	1	4	2
4	4	4	2

# DABA Example

f2.

*shrink*

f. max: 5

*insert 2*

*shrink*

g. max: 5

*evict*

*shift*

h. max: 4

*insert 7*

*shift*

i. max: 7

FL	R	ABE
5	3	1 4
5	3	1 4

F	L	R	A	BE
5	3	1	4	
5	3	1	4	

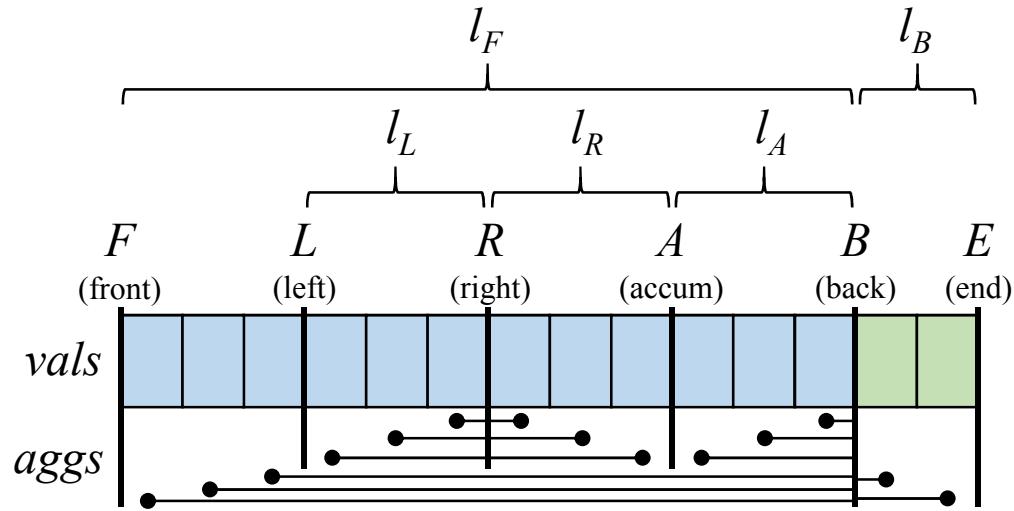
F	LRA	B	E
5	3	1 4	2
5	4	4 4	2

F	LRA	B	E
3	1	4 2	
4	4	4	2

F	LRAB	E
3	1	4 2 7
4	4	4 2 7

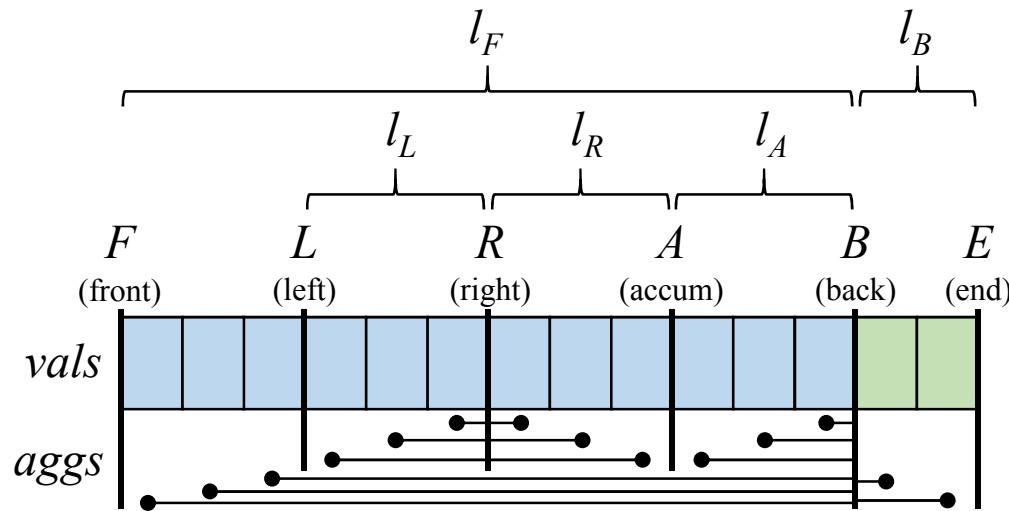
# Aggregation Invariants

(Answering queries correctly)



# Size Invariants

## (Getting work done on time)



$$(|l_F| = 0 \quad \text{and} \quad |l_B| = 0)$$

*or*

$$(|l_L| = |l_R| \quad \text{and} \quad |l_L| + |l_R| + |l_A| + 1 = |l_F| - |l_B|)$$

# Related Work

Algorithm	Time	Space	Invertible	FIFO
Subtract on evict	worst $O(1)$	$O(n)$	needed	no
Recalculate from scratch	worst $O(n)$	$O(n)$	no	no
Tree-based (e.g. reactive)	amzd. $O(\log n)$	$O(n)$	no	no
Two-stacks	amzd. $O(1)$	$O(n)$	no	needed
DABA	worst $O(1)$	$O(n)$	no	needed

# Results

**Latencies** [cycles],  $n = 2^{14}$ , average  $\pm$  standard deviation

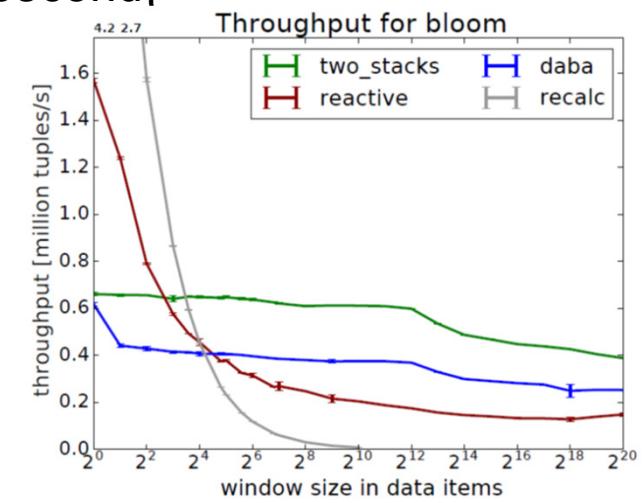
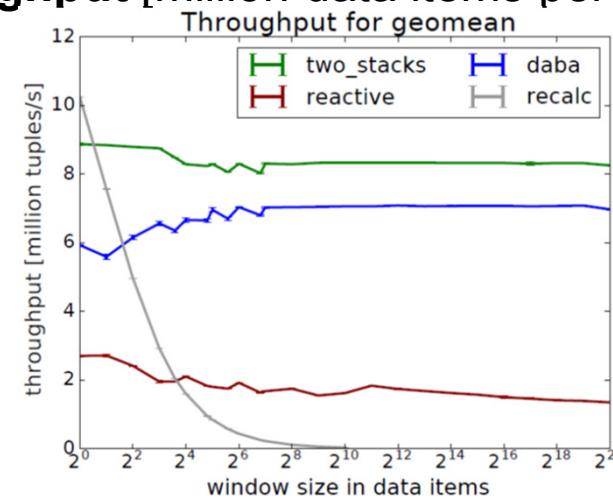
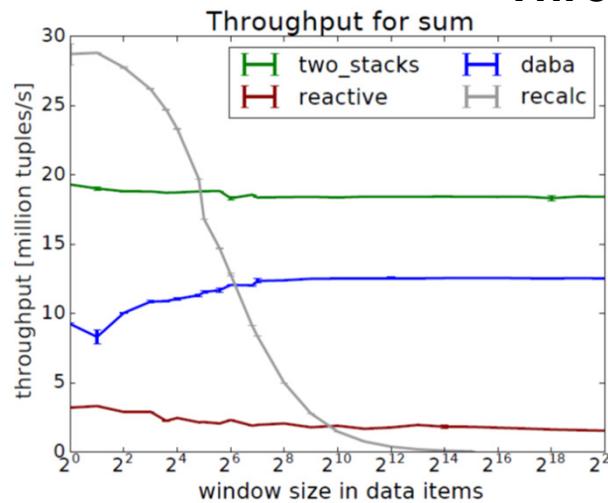
Operator	Two-Stacks	DABA
Sum	$127 \pm 2,980$	$193 \pm 113$
Max	$125 \pm 2,991$	$200 \pm 122$
GeoMean	$299 \pm 3,777$	$366 \pm 168$
Bloom	$5,532 \pm 298,982$	$9,021 \pm 4,289$

# Results

**Latencies [cycles],  $n = 2^{14}$ , average  $\pm$  standard deviation**

Operator	Two-Stacks	DABA
Sum	$127 \pm 2,980$	$193 \pm 113$
Max	$125 \pm 2,991$	$200 \pm 122$
GeoMean	$299 \pm 3,777$	$366 \pm 168$
Bloom	$5,532 \pm 298,982$	$9,021 \pm 4,289$

**Throughput [million data items per second]**



# Conclusions

DABA aggregates FIFO sliding windows  
in  $O(1)$  worst-case time and  $O(n)$  space

DABA works for any associative  
binary aggregation operator  $\oplus$

Orthogonal optimizations (not in this talk):  
data parallelism; coarse-grained sliding