Incremental Techniques for Large-Scale Dynamic Query Processing

Tutorial

Part 2

Iman Elghandour[1] Ahmet Kara[2] Dan Olteanu[2] Stijn Vansummeren[1]

1 Université Libre de Bruxelles

2 University of Oxford

Fondation Wiener-Anspach

European Research Council
Established by the European Commission

Outline

- Part I: Introduction
- Part II: Main Algorithmic Ideas in Dynamic Query Processing: Traditional IVM and Recent Advances
- **Part III: Generalizations to Arbitrary Ring Structures**
- Part IV: Dynamic Query Processing in Big Data Frameworks
- Part V: Outlook

Generalisation to Arbitrary Ring Structures

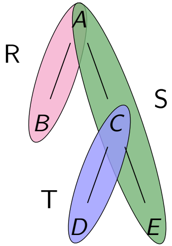
- The IVM machinery presented so far uses the ring over $\mathbb{Z}$
  $\Longrightarrow$ Tuples are mapped to integers.
  $\Longrightarrow$ Query operators use addition and multiplication.
- This ring suffices to compute and maintain the count aggregate.
- However, more complex analytical tasks might require other rings.
- Now, we present Factorized IVM (F-IVM) that allows for task-specific rings.
  $\Longrightarrow$ The machinery for maintenance remains the same.
  $\Longrightarrow$ Different applications are captured by different rings.

Incremental View Maintenance with Triple Lock Factorization Benefits.
Milos Nikolic and Dan Olteanu. SIGMOD 2018

Example: COUNT Aggregate

Compute COUNT over the natural join:
$R(a, b)$, $S(a, c, e)$, $T(c, d)$

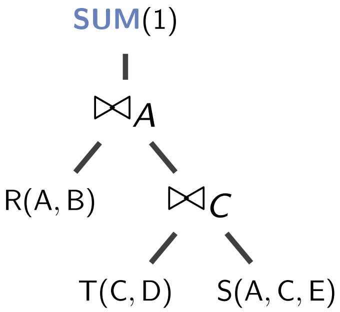
```
Q = SELECT SUM(1)
    FROM R NATURAL JOIN S
           NATURAL JOIN T
```



Example: COUNT Aggregate

Naïve: compute the join and then SUM(1)

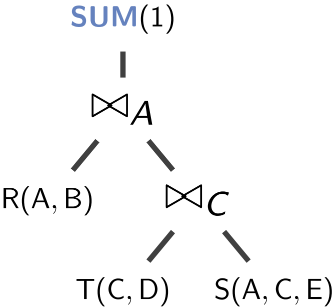
```
Q = SELECT SUM(1)
    FROM R NATURAL JOIN S
           NATURAL JOIN T
```



Example: COUNT Aggregate

Naïve: compute the join and then SUM(1)

```
Q = SELECT SUM(1)
    FROM R NATURAL JOIN S
           NATURAL JOIN T
```



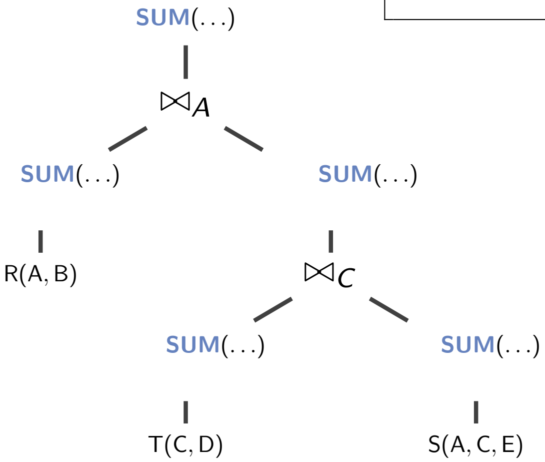
Let all relations be of size N

Computing Q takes $\mathcal{O}(N^3)$ time!

Example: COUNT Aggregate

Push SUM past joins
to eliminate variables

```
Q = SELECT SUM(1)
    FROM R NATURAL JOIN S
           NATURAL JOIN T
```



Example: COUNT Aggregate

Push SUM past joins
to eliminate variables

```
Q = SELECT SUM(1)
    FROM R NATURAL JOIN S
           NATURAL JOIN T
```

$Q = \textbf{SUM}(C_B * C_C)$

$\bowtie_A$

$V_R = A, \textbf{SUM}(1)$ as C_B
GROUP BY A

R(A, B)

$V_{ST} = A, \textbf{SUM}(C_D * C_E)$ as C_C
GROUP BY A

$\bowtie_C$

$V_T = C, \textbf{SUM}(1)$ as C_D
GROUP BY C

T(C, D)

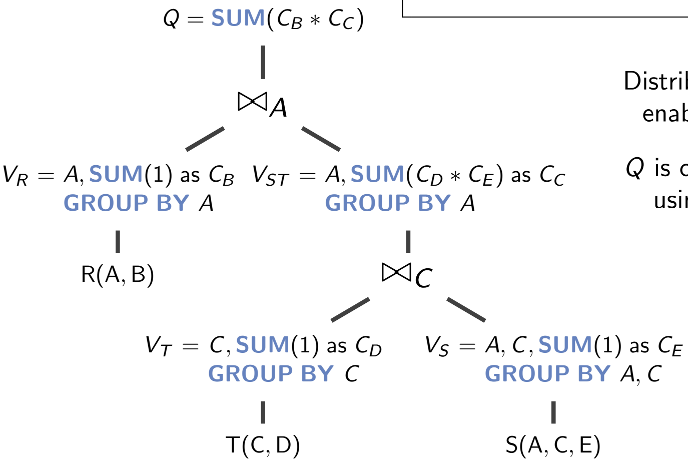
$V_S = A, C, \textbf{SUM}(1)$ as C_E
GROUP BY A, C

S(A, C, E)

Example: COUNT Aggregate

Push SUM past joins
to eliminate variables

```
Q = SELECT SUM(1)
    FROM R NATURAL JOIN S
           NATURAL JOIN T
```



Factorized evaluation
Distributivity of * over **SUM**
enables this query rewriting

Q is computed in $\mathcal{O}(N)$ time
using a hierarchy of views!

Example: SUM Aggregate

SUM over products of B and C

```
Q = SELECT SUM(B * C)
    FROM R NATURAL JOIN S
           NATURAL JOIN T
```

Example: SUM Aggregate

SUM over products of B and C

```
Q = SELECT SUM(B * C)
    FROM R NATURAL JOIN S
           NATURAL JOIN T
```

$Q = \textbf{SUM}(S_B * S_C)$

$\bowtie_A$

$V_R = A, \textbf{SUM}(B)$ as S_B **GROUP BY** A

$V_{ST} = A, \textbf{SUM}(S_D * S_E * C)$ as S_C **GROUP BY** A

R(A, B)

$\bowtie_C$

$V_T = C, \textbf{SUM}(1)$ as S_D **GROUP BY** C

$V_S = A, C, \textbf{SUM}(1)$ as S_E **GROUP BY** A, C

T(C, D)

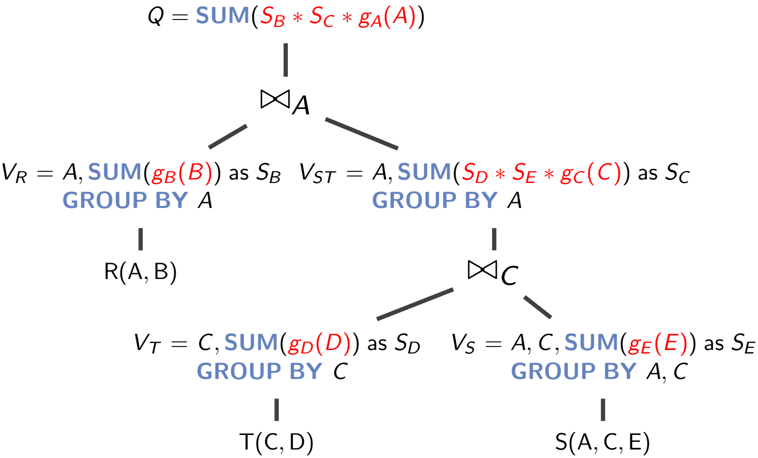
S(A, C, E)

Reuse counts of D and E when joining on C

Multiply by C only after joining on C

More General Example: SUM Aggregate

```
Q = SELECT SUM( gA(A) * gB(B) * gC(C) * gD(D) * gE(E) )
    FROM R NATURAL JOIN S NATURAL JOIN T
```



More General Example: SUM Aggregate

```
Q = SELECT SUM( gA(A) * gB(B) * gC(C) * gD(D) * gE(E) )
    FROM R NATURAL JOIN S NATURAL JOIN T
```

Join on & eliminate one variable at a time

$Q = \textbf{SUM}(S_B * S_C * g_A(A))$

$\bowtie_A$

$V_R = A, \textbf{SUM}(g_B(B))$ as S_B **GROUP BY** A

$V_{ST} = A, \textbf{SUM}(S_D * S_E * g_C(C))$ as S_C **GROUP BY** A

R(A, B)

$\bowtie_C$

$V_T = C, \textbf{SUM}(g_D(D))$ as S_D **GROUP BY** C

$V_S = A, C, \textbf{SUM}(g_E(E))$ as S_E **GROUP BY** A, C

eliminate D

T(C, D)

S(A, C, E)

More General Example: SUM Aggregate

```
Q = SELECT SUM( gA(A) * gB(B) * gC(C) * gD(D) * gE(E) )
    FROM R NATURAL JOIN S NATURAL JOIN T
```

Join on & eliminate one variable at a time

$Q = \textbf{SUM}(S_B * S_C * g_A(A))$

$\bowtie_A$

$V_R = A, \textbf{SUM}(g_B(B))$ as S_B **GROUP BY** A

R(A, B)

$V_{ST} = A, \textbf{SUM}(S_D * S_E * g_C(C))$ as S_C **GROUP BY** A

$\bowtie_C$

$V_T = C, \textbf{SUM}(g_D(D))$ as S_D **GROUP BY** C

eliminate D

T(C, D)

$V_S = A, C, \textbf{SUM}(g_E(E))$ as S_E **GROUP BY** A, C

eliminate E

S(A, C, E)

More General Example: SUM Aggregate

```
Q = SELECT SUM( gA(A) * gB(B) * gC(C) * gD(D) * gE(E) )
    FROM R NATURAL JOIN S NATURAL JOIN T
```

Join on & eliminate one variable at a time

$Q = \textbf{SUM}(S_B * S_C * g_A(A))$

$\bowtie_A$

$V_R = A, \textbf{SUM}(g_B(B))$ as S_B **GROUP BY** A

R(A, B)

$V_{ST} = A, \textbf{SUM}(S_D * S_E * g_C(C))$ as S_C **GROUP BY** A

eliminate C

$\bowtie_C$

$V_T = C, \textbf{SUM}(g_D(D))$ as S_D **GROUP BY** C

eliminate D

T(C, D)

$V_S = A, C, \textbf{SUM}(g_E(E))$ as S_E **GROUP BY** A, C

eliminate E

S(A, C, E)

More General Example: SUM Aggregate

```
Q = SELECT SUM( gA(A) * gB(B) * gC(C) * gD(D) * gE(E) )
    FROM R NATURAL JOIN S NATURAL JOIN T
```

Join on & eliminate one variable at a time

$Q = \textbf{SUM}(S_B * S_C * g_A(A))$

- $\bowtie_A$
  - $V_R = A, \textbf{SUM}(g_B(B))$ as S_B **GROUP BY** A
    - eliminate B
    - R(A, B)
  - $V_{ST} = A, \textbf{SUM}(S_D * S_E * g_C(C))$ as S_C **GROUP BY** A
    - eliminate C
    - $\bowtie_C$
      - $V_T = C, \textbf{SUM}(g_D(D))$ as S_D **GROUP BY** C
        - eliminate D
        - T(C, D)
      - $V_S = A, C, \textbf{SUM}(g_E(E))$ as S_E **GROUP BY** A, C
        - eliminate E
        - S(A, C, E)

More General Example: SUM Aggregate

```
Q = SELECT SUM( gA(A) * gB(B) * gC(C) * gD(D) * gE(E) )
    FROM R NATURAL JOIN S NATURAL JOIN T
```

Join on & eliminate one variable at a time

$Q = \textbf{SUM}(S_B * S_C * g_A(A))$

eliminate A

$\bowtie_A$

$V_R = A, \textbf{SUM}(g_B(B))$ as S_B **GROUP BY** A

eliminate B

R(A, B)

$V_{ST} = A, \textbf{SUM}(S_D * S_E * g_C(C))$ as S_C **GROUP BY** A

eliminate C

$\bowtie_C$

$V_T = C, \textbf{SUM}(g_D(D))$ as S_D **GROUP BY** C

eliminate D

T(C, D)

$V_S = A, C, \textbf{SUM}(g_E(E))$ as S_E **GROUP BY** A, C

eliminate E

S(A, C, E)

More General Example: SUM Aggregate

```
Q = SELECT SUM( gA(A) * gB(B) * gC(C) * gD(D) * gE(E) )
    FROM R NATURAL JOIN S NATURAL JOIN T
```

Imagine aggregate values are of type $\mathcal{R}$

$$g_X : \mathrm{Dom}(X) \to \mathcal{R}$$

Can we evaluate Q using the query plan from before?

Yes(!), but we need to:

- Define the binary operators $*$ and $+$ in $\mathcal{R}$
- Define zero in $\mathcal{R}$ (for initial values)
- Define one in $\mathcal{R}$ (e.g., if X is not used, $g_X(x) = 1$)
- Ensure distributivity of $*$ over $+$

Rings

- A ring $(\mathcal{R}, +, *, \mathbf{0}, \mathbf{1})$ is a set $\mathcal{R}$ with two binary operators:

Additive commutativity	$a + b = b + a$
Additive associativity	$(a + b) + c = a + (b + c)$
Additive identity	$\mathbf{0} + a = a + \mathbf{0} = a$
Additive inverse	$\exists - a \in \mathcal{R} : a + (-a) = (-a) + a = \mathbf{0}$
Multiplicative associativity	$(a * b) * c = a * (b * c)$
Multiplicative identity	$a * \mathbf{1} = \mathbf{1} * a = a$
Left and right distributivity	$a * (b + c) = a * b + a * c$ and $(a + b) * c = a * c + b * c$

Examples: $\mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}, \mathbb{R}^n$, matrix ring, polynomial ring

Factorized Ring Computation

- Relations are functions
  - mapping keys (tuples) to payloads (ring elements)

A	B	$\rightarrow$	R[A, B]
a_1	b_1	$\rightarrow$	r_1
a_2	b_1	$\rightarrow$	r_2

Finitely many tuples with non-zero payloads

r_1 and r_2 are elements from a ring

- Query language
  - Operations: union, join, and variable marginalization
  - More expressiveness via application-specific rings
- Query evaluation
  - using *view trees* as shown before

More General SUM Aggregate

```
Q = SELECT SUM( gA(A) * gB(B) * gC(C) * gD(D) * gE(E) )
    FROM R NATURAL JOIN S NATURAL JOIN T
```

Expressed in our framework:

$$Q = \underbrace{\bigoplus_A \bigoplus_B \bigoplus_C \bigoplus_D \bigoplus_E}_{\textit{variable marginalization}} \left(\underbrace{R[A,B] \otimes S[A,C,E] \otimes T[C,D]}_{\textit{natural joins}} \right)$$

Intuition: Relation payloads carry out the summation!

Marginalization of X applies g_X, sums payloads, projects away X

Join multiplies payloads of matching tuples

Query Operators

Relations R, S, and T with payloads from a ring $(\mathcal{R}, +, *, \mathbf{0}, \mathbf{1})$:

A	B	$\rightarrow$	R[A, B]
a_1	b_1	$\rightarrow$	r_1
a_2	b_1	$\rightarrow$	r_2

A	B	$\rightarrow$	S[A, B]
a_2	b_1	$\rightarrow$	s_1
a_3	b_2	$\rightarrow$	s_2

B	C	$\rightarrow$	T[B, C]
b_1	c_1	$\rightarrow$	t_1
b_2	c_2	$\rightarrow$	t_2

Query Operators

Relations R, S, and T with payloads from a ring $(\mathcal{R}, +, *, \mathbf{0}, \mathbf{1})$:

A	B	$\rightarrow$	R[A, B]
a_1	b_1	$\rightarrow$	r_1
a_2	b_1	$\rightarrow$	r_2

A	B	$\rightarrow$	S[A, B]
a_2	b_1	$\rightarrow$	s_1
a_3	b_2	$\rightarrow$	s_2

B	C	$\rightarrow$	T[B, C]
b_1	c_1	$\rightarrow$	t_1
b_2	c_2	$\rightarrow$	t_2

Union $\uplus$

A	B	$\rightarrow$	(R $\uplus$ S)[A, B]
a_1	b_1	$\rightarrow$	r_1
a_2	b_1	$\rightarrow$	$r_2 + s_1$
a_3	b_2	$\rightarrow$	s_2

Query Operators

Relations R, S, and T with payloads from a ring $(\mathcal{R}, +, *, \mathbf{0}, \mathbf{1})$:

A	B	$\rightarrow$	R[A, B]
a_1	b_1	$\rightarrow$	r_1
a_2	b_1	$\rightarrow$	r_2

A	B	$\rightarrow$	S[A, B]
a_2	b_1	$\rightarrow$	s_1
a_3	b_2	$\rightarrow$	s_2

B	C	$\rightarrow$	T[B, C]
b_1	c_1	$\rightarrow$	t_1
b_2	c_2	$\rightarrow$	t_2

Union $\uplus$

A	B	$\rightarrow$	(R $\uplus$ S)[A, B]
a_1	b_1	$\rightarrow$	r_1
a_2	b_1	$\rightarrow$	$r_2 + s_1$
a_3	b_2	$\rightarrow$	s_2

Query Operators

Relations R, S, and T with payloads from a ring $(\mathcal{R}, +, *, \mathbf{0}, \mathbf{1})$:

A	B	→	R[A, B]
a_1	b_1	→	r_1
a_2	b_1	→	r_2

A	B	→	S[A, B]
a_2	b_1	→	s_1
a_3	b_2	→	s_2

B	C	→	T[B, C]
b_1	c_1	→	t_1
b_2	c_2	→	t_2

Union $\uplus$

A	B	→	(R $\uplus$ S)[A, B]
a_1	b_1	→	r_1
a_2	b_1	→	$r_2 + s_1$
a_3	b_2	→	s_2

Join $\otimes$

A	B	C	→	((R $\uplus$ S) $\otimes$ T)[A, B, C]
a_1	b_1	c_1	→	$r_1 * t_1$
a_2	b_1	c_1	→	$(r_2 + s_1) * t_1$
a_3	b_2	c_2	→	$s_2 * t_2$

Query Operators

Relations R, S, and T with payloads from a ring $(\mathcal{R}, +, *, \mathbf{0}, \mathbf{1})$:

A	B	→	R[A, B]
a_1	b_1	→	r_1
a_2	b_1	→	r_2

A	B	→	S[A, B]
a_2	b_1	→	s_1
a_3	b_2	→	s_2

B	C	→	T[B, C]
b_1	c_1	→	t_1
b_2	c_2	→	t_2

Union $\uplus$

A	B	→	(R $\uplus$ S)[A, B]
a_1	b_1	→	r_1
a_2	b_1	→	$r_2 + s_1$
a_3	b_2	→	s_2

Join $\otimes$

A	B	C	→	((R $\uplus$ S) $\otimes$ T)[A, B, C]
a_1	b_1	c_1	→	$r_1 * t_1$
a_2	b_1	c_1	→	$(r_2 + s_1) * t_1$
a_3	b_2	c_2	→	$s_2 * t_2$

Query Operators

Relations R, S, and T with payloads from a ring $(\mathcal{R}, +, *, \mathbf{0}, \mathbf{1})$:

A	B	$\to$	R[A, B]
a_1	b_1	$\to$	r_1
a_2	b_1	$\to$	r_2

A	B	$\to$	S[A, B]
a_2	b_1	$\to$	s_1
a_3	b_2	$\to$	s_2

B	C	$\to$	T[B, C]
b_1	c_1	$\to$	t_1
b_2	c_2	$\to$	t_2

Union $\uplus$

A	B	$\to$	(R $\uplus$ S)[A, B]
a_1	b_1	$\to$	r_1
a_2	b_1	$\to$	$r_2 + s_1$
a_3	b_2	$\to$	s_2

Join $\otimes$

A	B	C	$\to$	((R $\uplus$ S) $\otimes$ T)[A, B, C]
a_1	b_1	c_1	$\to$	$r_1 * t_1$
a_2	b_1	c_1	$\to$	$(r_2 + s_1) * t_1$
a_3	b_2	c_2	$\to$	$s_2 * t_2$

Marginalization $\bigoplus_A$

for a given
$g_A : \text{Dom}(A) \to \mathcal{R}$

B	C	$\to$	($\bigoplus_A$(R $\uplus$ S) $\otimes$ T)[B, C]
b_1	c_1	$\to$	$r_1 * t_1 * g_A(a_1) + (r_2 + s_1) * t_1 * g_A(a_2)$
b_2	c_2	$\to$	$s_2 * t_2 * g_A(a_3)$

General Query Form

```
Q = SELECT X_1, ..., X_f , SUM( g_{f+1}(X_{f+1}) * ··· * g_m(X_m) )
    FROM R_1 NATURAL JOIN ... NATURAL JOIN R_n
    GROUP BY X_1, ..., X_f
```

Expressed as Functional Aggregate Query:

$$Q[X_1, \ldots, X_f] = \bigoplus_{X_{f+1}} \cdots \bigoplus_{X_m} \bigotimes_{i \in [n]} R_i[\mathcal{S}_i]$$

where:

- Relations $R_1, \ldots, R_n$ are defined over variables $X_1, \ldots, X_m$
- $X_1, \ldots, X_f$ are free variables
- R_i maps keys over schema $\mathcal{S}_i$ to payloads in a ring $(\mathcal{R}, +, *, \mathbf{0}, \mathbf{1})$
- Aggregations $\bigoplus_{X_{f+1}}, \ldots, \bigoplus_{X_m}$ use functions $g_{f+1}, \ldots, g_m$

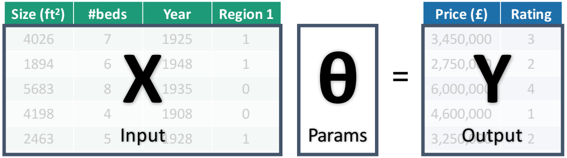
Applications

A host of problems are captured using task-specific rings

- Group-by aggregation over joins (we've seen this already)
- Gradient computation for learning regression models
- Representation of results of conjunctive queries
- Matrix chain multiplication
- ...

Learning Linear Regression Models

- Find model parameters Θ best satisfying:



- Iterative gradient computation:

$$\mathbf{\Theta}_{i+1} = \mathbf{\Theta}_i - \alpha\,\mathbf{X}^T(\mathbf{X}\,\mathbf{\Theta}_i - \mathbf{Y}) \quad \text{(repeat until convergence)}$$

- Matrices $\mathbf{X}^\mathbf{T}\,\mathbf{X}$ and $\mathbf{X}^\mathbf{T}\,\mathbf{Y}$ computed once for all iterations
  - Compute $SUM(X_i \cdot X_j)$, $SUM(X_i)$, and $SUM(1)$ for variables X_i and X_j
  - We assume that all variables are continuous

Learning Linear Regression Models over Joins

Compute $\mathbf{X}^{\mathsf{T}}\mathbf{X}$ where $\mathbf{X}$ is the join of the input relations

- Naïve: compute the join, then $\mathcal{O}(m^2)$ sums over the join result ($m = \#$query variables)

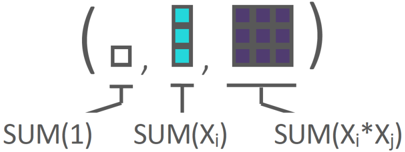
- Factorized: compute one optimized join-aggregate query
  - Using our running query

    $$\mathrm{Q} = \bigoplus_A \bigoplus_B \bigoplus_C \bigoplus_D \bigoplus_E (\, \mathrm{R}[A, B] \otimes \mathrm{S}[A, C, E] \otimes \mathrm{T}[C, D]\,)$$

    but a different payload ring and different functions g_X!

Linear Regression Ring

Set $\mathcal{R} = (\mathbb{Z}, \mathbb{R}^m, \mathbb{R}^{m\times m})$ of triples



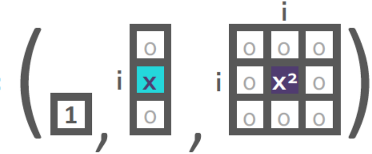
$$a +^{\mathcal{R}} b = (c_a + c_b, \mathbf{s}_a + \mathbf{s}_b, \mathbf{Q}_a + \mathbf{Q}_b)$$

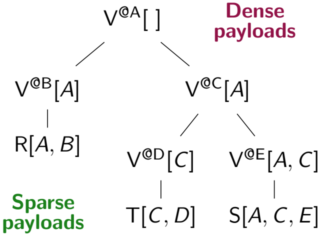
$$a *^{\mathcal{R}} b = (c_a c_b, c_b \mathbf{s}_a + c_a \mathbf{s}_b, c_b \mathbf{Q}_a + c_a \mathbf{Q}_b + \mathbf{s}_a \mathbf{s}_b^T + \mathbf{s}_b \mathbf{s}_a^T)$$

$$\mathbf{0} = (0, \mathbf{0}_{m\times 1}, \mathbf{0}_{m\times m})$$

$$\mathbf{1} = (1, \mathbf{0}_{m\times 1}, \mathbf{0}_{m\times m})$$

$g_{X_i}(x)$ for variable X_i





Relational Data Ring

- Set of relations over $\mathcal{R}$ with $\uplus$ and $\otimes$ forms a ring of relations
  - Relation $\mathbf{0}$ maps every tuple to $\mathbf{0} \in \mathcal{R}$
  - Relation $\mathbf{1}$ maps the empty tuple to $\mathbf{1} \in \mathcal{R}$, others to $\mathbf{0} \in \mathcal{R}$
- Payloads: Relations over $\mathcal{R} = \mathbb{Z}$ with the same schema!

A	B	$\rightarrow$	R[A, B]
a_1	b_1	$\rightarrow$	$\begin{array}{\|l} \text{C} \\ \hline c_1 \rightarrow 1 \\ c_2 \rightarrow 1 \end{array}$
a_2	b_1	$\rightarrow$	$\begin{array}{\|l} \text{C} \\ \hline c_3 \rightarrow 1 \end{array}$

Keep results of conjunctive queries in payloads

Evaluating Conjunctive Queries using Relational Payloads

- Consider the conjunctive query:

$$Q(A, B, C, D) = R(A, B), S(A, C, E), T(C, D)$$

- Compute Q using relations with relational payloads

$$\mathrm{Q} = \bigoplus_A \bigoplus_B \bigoplus_C \bigoplus_D \bigoplus_E (\, \mathrm{R}[A, B] \otimes \mathrm{S}[A, C, E] \otimes \mathrm{T}[C, D]\,)$$

- Lifting (aggregate) functions:

$$g_X(x) = \begin{cases} \begin{array}{|l} \mathrm{X} \\ \hline \mathrm{x} \to 1 \end{array} & \text{if } X \text{ is a free variable} \\ \\ \begin{array}{|l} \\ \hline () \to 1 \end{array} & \textit{otherwise} \end{cases}$$

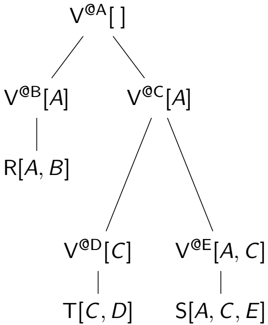
Listing Representation of Conjunctive Query Results

$$Q(A,B,C,D) = R(A,B), S(A,C,E), T(C,D)$$

A	B	→	R[A,B]
a_1	b_1	→	$() \to 1$
a_1	b_2	→	$() \to 1$
a_2	b_3	→	$() \to 1$
a_3	b_4	→	$() \to 1$

A	C	E	→	S[A,C,E]
a_1	c_1	e_1	→	$() \to 1$
a_1	c_1	e_2	→	$() \to 1$
a_1	c_2	e_3	→	$() \to 1$
a_2	c_2	e_4	→	$() \to 1$

C	D	→	T[C,D]
c_1	d_1	→	$() \to 1$
c_2	d_2	→	$() \to 1$
c_2	d_3	→	$() \to 1$
c_3	d_4	→	$() \to 1$



Listing Representation of Conjunctive Query Results

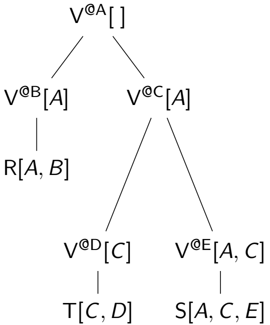
$$Q(A, B, C, D) = R(A, B), S(A, C, E), T(C, D)$$

A	B	→ R[A,B]
a_1	b_1	→ () → 1
a_1	b_2	→ () → 1
a_2	b_3	→ () → 1
a_3	b_4	→ () → 1

A	C	E	→ S[A,C,E]
a_1	c_1	e_1	→ () → 1
a_1	c_1	e_2	→ () → 1
a_1	c_2	e_3	→ () → 1
a_2	c_2	e_4	→ () → 1

C	D	→ T[C,D]
c_1	d_1	→ () → 1
c_2	d_2	→ () → 1
c_2	d_3	→ () → 1
c_3	d_4	→ () → 1

A	→ V@B[A]
a_1	→ B: $b_1 \to 1$, $b_2 \to 1$
a_2	→ B: $b_3 \to 1$
a_3	→ B: $b_4 \to 1$



Listing Representation of Conjunctive Query Results

$$Q(A,B,C,D) = R(A,B), S(A,C,E), T(C,D)$$

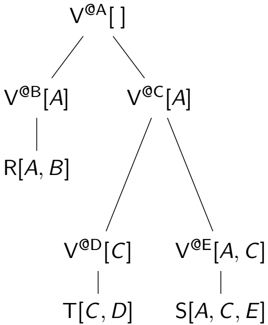
A	B	→ R[A,B]
a_1	b_1	→ $() \to 1$
a_1	b_2	→ $() \to 1$
a_2	b_3	→ $() \to 1$
a_3	b_4	→ $() \to 1$

A	C	E	→ S[A,C,E]
a_1	c_1	e_1	→ $() \to 1$
a_1	c_1	e_2	→ $() \to 1$
a_1	c_2	e_3	→ $() \to 1$
a_2	c_2	e_4	→ $() \to 1$

C	D	→ T[C,D]
c_1	d_1	→ $() \to 1$
c_2	d_2	→ $() \to 1$
c_2	d_3	→ $() \to 1$
c_3	d_4	→ $() \to 1$

A	→ V@B[A]
a_1	→ B $b_1 \to 1$ $b_2 \to 1$
a_2	→ B $b_3 \to 1$
a_3	→ B $b_4 \to 1$

C	→ V@D[C]
c_1	→ D $d_1 \to 1$
c_2	→ D $d_2 \to 1$ $d_3 \to 1$
c_3	→ D $d_4 \to 1$



Listing Representation of Conjunctive Query Results

$$Q(A,B,C,D) = R(A,B), S(A,C,E), T(C,D)$$

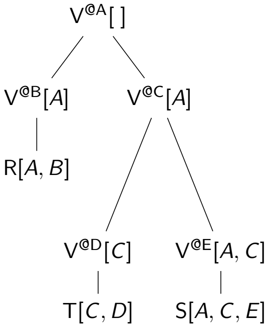
A	B	→ R[A,B]
a_1	b_1	→ $() \to 1$
a_1	b_2	→ $() \to 1$
a_2	b_3	→ $() \to 1$
a_3	b_4	→ $() \to 1$

A	C	E	→ S[A,C,E]
a_1	c_1	e_1	→ $() \to 1$
a_1	c_1	e_2	→ $() \to 1$
a_1	c_2	e_3	→ $() \to 1$
a_2	c_2	e_4	→ $() \to 1$

C	D	→ T[C,D]
c_1	d_1	→ $() \to 1$
c_2	d_2	→ $() \to 1$
c_2	d_3	→ $() \to 1$
c_3	d_4	→ $() \to 1$

A	→ $V^{@B}$[A]
a_1	→ B $b_1 \to 1$ $b_2 \to 1$
a_2	→ B $b_3 \to 1$
a_3	→ B $b_4 \to 1$

C	→ $V^{@D}$[C]
c_1	→ D $d_1 \to 1$
c_2	→ D $d_2 \to 1$ $d_3 \to 1$
c_3	→ D $d_4 \to 1$



A	C	→ $V^{@E}$[A,C]
a_1	c_1	→ $() \to 2$
a_1	c_2	→ $() \to 1$
a_2	c_2	→ $() \to 1$

Listing Representation of Conjunctive Query Results

$$Q(A,B,C,D) = R(A,B), S(A,C,E), T(C,D)$$

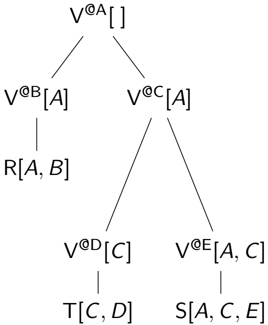
A	B	→ R[A,B]
a_1	b_1	→ $() \to 1$
a_1	b_2	→ $() \to 1$
a_2	b_3	→ $() \to 1$
a_3	b_4	→ $() \to 1$

A	C	E	→ S[A,C,E]
a_1	c_1	e_1	→ $() \to 1$
a_1	c_1	e_2	→ $() \to 1$
a_1	c_2	e_3	→ $() \to 1$
a_2	c_2	e_4	→ $() \to 1$

C	D	→ T[C,D]
c_1	d_1	→ $() \to 1$
c_2	d_2	→ $() \to 1$
c_2	d_3	→ $() \to 1$
c_3	d_4	→ $() \to 1$

A →	$V^{@B}$[A]: B
a_1 →	$b_1 \to 1$
	$b_2 \to 1$
a_2 →	$b_3 \to 1$
a_3 →	$b_4 \to 1$

C →	$V^{@D}$[C]: D
c_1 →	$d_1 \to 1$
c_2 →	$d_2 \to 1$
	$d_3 \to 1$
c_3 →	$d_4 \to 1$



A →	$V^{@C}$[A]: C D
a_1 →	$c_1\ d_1 \to 2$
	$c_2\ d_2 \to 1$
	$c_2\ d_3 \to 1$
a_2 →	$c_2\ d_2 \to 1$
	$c_2\ d_3 \to 1$

A	C	→ $V^{@E}$[A,C]
a_1	c_1	→ $() \to 2$
a_1	c_2	→ $() \to 1$
a_2	c_2	→ $() \to 1$

Listing Representation of Conjunctive Query Results

$$Q(A,B,C,D) = R(A,B), S(A,C,E), T(C,D)$$

A B → R[A,B]

A	B	→	R[A,B]
a_1	b_1	→	() → 1
a_1	b_2	→	() → 1
a_2	b_3	→	() → 1
a_3	b_4	→	() → 1

A C E → S[A,C,E]

A	C	E	→	S[A,C,E]
a_1	c_1	e_1	→	() → 1
a_1	c_1	e_2	→	() → 1
a_1	c_2	e_3	→	() → 1
a_2	c_2	e_4	→	() → 1

C D → T[C,D]

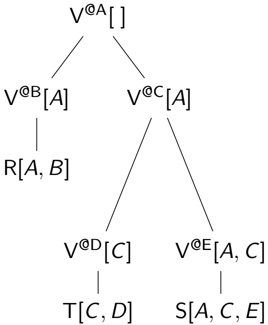
C	D	→	T[C,D]
c_1	d_1	→	() → 1
c_2	d_2	→	() → 1
c_2	d_3	→	() → 1
c_3	d_4	→	() → 1

A → $V^{@B}$[A]

A	→	B
a_1	→	b_1 → 1
		b_2 → 1
a_2	→	b_3 → 1
a_3	→	b_4 → 1

C → $V^{@D}$[C]

C	→	D
c_1	→	d_1 → 1
c_2	→	d_2 → 1
		d_3 → 1
c_3	→	d_4 → 1



() → $V^{@A}$[ ]

	→	A	B	C	D	
()	→	a_1	b_1	c_1	d_1	→ 2
		a_1	b_1	c_2	d_2	→ 1
		a_1	b_1	c_2	d_3	→ 1
		a_1	b_2	c_1	d_1	→ 2
		a_1	b_2	c_2	d_2	→ 1
		a_1	b_2	c_2	d_3	→ 1
		a_2	b_3	c_2	d_2	→ 1
		a_2	b_3	c_2	d_3	→ 1

A → $V^{@C}$[A]

A	→	C	D	
a_1	→	c_1	d_1	→ 2
		c_2	d_2	→ 1
		c_2	d_3	→ 1
a_2	→	c_2	d_2	→ 1
		c_2	d_3	→ 1

A C → $V^{@E}$[A,C]

A	C	→	$V^{@E}$[A,C]
a_1	c_1	→	() → 2
a_1	c_2	→	() → 1
a_2	c_2	→	() → 1

Factorized Representation of Conjunctive Query Results

$$Q(A,B,C,D) = R(A,B), S(A,C,E), T(C,D)$$

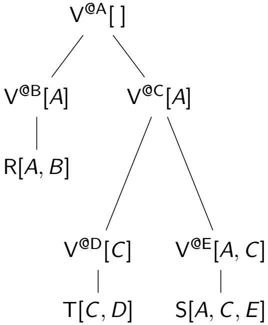
A	B	→ R[A,B]
a_1	b_1	→ () → 1
a_1	b_2	→ () → 1
a_2	b_3	→ () → 1
a_3	b_4	→ () → 1

A	C	E	→ S[A,C,E]
a_1	c_1	e_1	→ () → 1
a_1	c_1	e_2	→ () → 1
a_1	c_2	e_3	→ () → 1
a_2	c_2	e_4	→ () → 1

C	D	→ T[C,D]
c_1	d_1	→ () → 1
c_2	d_2	→ () → 1
c_2	d_3	→ () → 1
c_3	d_4	→ () → 1

A	→ V@B[A]
a_1 →	B: $b_1 \to 1$, $b_2 \to 1$
a_2 →	B: $b_3 \to 1$
a_3 →	B: $b_4 \to 1$

C	→ V@D[C]
c_1 →	D: $d_1 \to 1$
c_2 →	D: $d_2 \to 1$, $d_3 \to 1$
c_3 →	D: $d_4 \to 1$



A	C	→ V@E[A,C]
a_1	c_1	→ () → 2
a_1	c_2	→ () → 1
a_2	c_2	→ () → 1

Factorized Representation of Conjunctive Query Results

$$Q(A,B,C,D) = R(A,B), S(A,C,E), T(C,D)$$

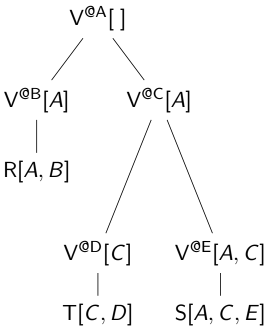
A	B	→ R[A,B]
a_1	b_1	→ () → 1
a_1	b_2	→ () → 1
a_2	b_3	→ () → 1
a_3	b_4	→ () → 1

A	C	E	→ S[A,C,E]
a_1	c_1	e_1	→ () → 1
a_1	c_1	e_2	→ () → 1
a_1	c_2	e_3	→ () → 1
a_2	c_2	e_4	→ () → 1

C	D	→ T[C,D]
c_1	d_1	→ () → 1
c_2	d_2	→ () → 1
c_2	d_3	→ () → 1
c_3	d_4	→ () → 1

A	→ $V^{@B}$[A]
a_1	→ B: $b_1 \to 1$, $b_2 \to 1$
a_2	→ B: $b_3 \to 1$
a_3	→ B: $b_4 \to 1$

C	→ $V^{@D}$[C]
c_1	→ D: $d_1 \to 1$
c_2	→ D: $d_2 \to 1$, $d_3 \to 1$
c_3	→ D: $d_4 \to 1$



A	→ $V^{@C}$[A]
a_1	→ C: $c_1 \to 2$, $c_2 \to 2$
a_2	→ C: $c_2 \to 2$

A	C	→ $V^{@E}$[A,C]
a_1	c_1	→ () → 2
a_1	c_2	→ () → 1
a_2	c_2	→ () → 1

Factorized Representation of Conjunctive Query Results

$$Q(A,B,C,D) = R(A,B), S(A,C,E), T(C,D)$$

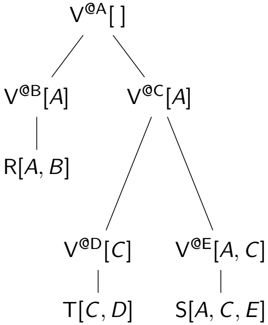
A	B	→ R[A,B]
a_1	b_1	→ $() \to 1$
a_1	b_2	→ $() \to 1$
a_2	b_3	→ $() \to 1$
a_3	b_4	→ $() \to 1$

A	C	E	→ S[A,C,E]
a_1	c_1	e_1	→ $() \to 1$
a_1	c_1	e_2	→ $() \to 1$
a_1	c_2	e_3	→ $() \to 1$
a_2	c_2	e_4	→ $() \to 1$

C	D	→ T[C,D]
c_1	d_1	→ $() \to 1$
c_2	d_2	→ $() \to 1$
c_2	d_3	→ $() \to 1$
c_3	d_4	→ $() \to 1$

A	→ V@B[A]
a_1 →	B: $b_1 \to 1$, $b_2 \to 1$
a_2 →	B: $b_3 \to 1$
a_3 →	B: $b_4 \to 1$

C	→ V@D[C]
c_1 →	D: $d_1 \to 1$
c_2 →	D: $d_2 \to 1$, $d_3 \to 1$
c_3 →	D: $d_4 \to 1$



()	→ V@A[ ]
() →	A: $a_1 \to 8$, $a_2 \to 2$

A	→ V@C[A]
a_1 →	C: $c_1 \to 2$, $c_2 \to 2$
a_2 →	C: $c_2 \to 2$

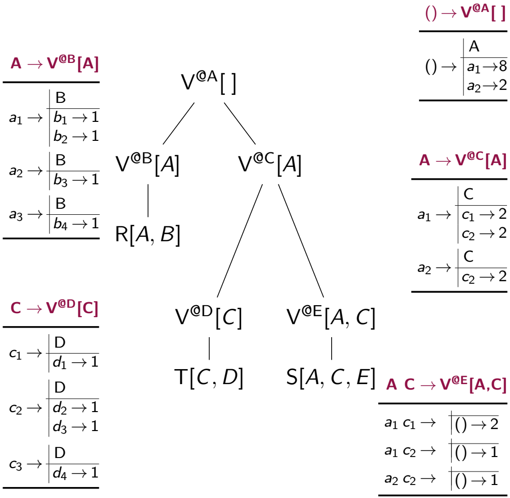
A	C	→ V@E[A,C]
a_1	c_1	→ $() \to 2$
a_1	c_2	→ $() \to 1$
a_2	c_2	→ $() \to 1$

Factorized Representation of Conjunctive Query Results

$$Q(A, B, C, D) = R(A, B), S(A, C, E), T(C, D)$$

Constant Delay Enumeration

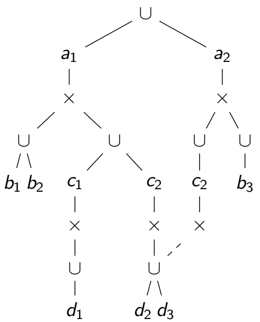
```
foreach a in V@A
  foreach b in V@B
    foreach c in V@C
      foreach d in V@D
        output (a,b,c,d)
```

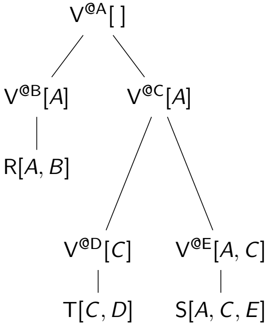


Factorized Representation of Conjunctive Query Results

$$Q(A, B, C, D) = R(A, B), S(A, C, E), T(C, D)$$

Factorized Join





$A \rightarrow V^{@B}[A]$

A	B
$a_1 \rightarrow$	$b_1 \rightarrow 1$
	$b_2 \rightarrow 1$
$a_2 \rightarrow$	$b_3 \rightarrow 1$
$a_3 \rightarrow$	$b_4 \rightarrow 1$

$C \rightarrow V^{@D}[C]$

C	D
$c_1 \rightarrow$	$d_1 \rightarrow 1$
$c_2 \rightarrow$	$d_2 \rightarrow 1$
	$d_3 \rightarrow 1$
$c_3 \rightarrow$	$d_4 \rightarrow 1$

$() \rightarrow V^{@A}[\,]$

()	A
$() \rightarrow$	$a_1 \rightarrow 8$
	$a_2 \rightarrow 2$

$A \rightarrow V^{@C}[A]$

A	C
$a_1 \rightarrow$	$c_1 \rightarrow 2$
	$c_2 \rightarrow 2$
$a_2 \rightarrow$	$c_2 \rightarrow 2$

$A\ C \rightarrow V^{@E}[A,C]$

A C	
$a_1\ c_1 \rightarrow$	$() \rightarrow 2$
$a_1\ c_2 \rightarrow$	$() \rightarrow 1$
$a_2\ c_2 \rightarrow$	$() \rightarrow 1$

Matrix Chain Multiplication

Input: Matrices $\boldsymbol{A}_i$ of size of $p_i \times p_{i+1}$ over some ring $\mathcal{R}$ ($i \in [n]$)

Compute: Product matrix $\boldsymbol{A}[x_1, x_{n+1}] = \sum_{x_2 \in [p_2]} \cdots \sum_{x_n \in p_n} \Pi_{i \in [n]} \boldsymbol{A}_i[x_i, x_{i+1}]$

Modeled in F-IVM:

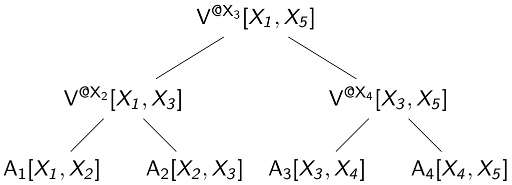
- Represent matrix $\boldsymbol{A}_i[x_i, x_{i+1}]$ by relation $\mathrm{A_i}[X_i, X_{i+1}]$ with $\mathrm{A_i}[a_i, a_{i+1}] = \boldsymbol{A}_i[a_i, a_{i+1}]$ (payloads are matrix entries)
- Express matrix multiplication by the query

$$\mathrm{A}[X_1, X_{n+1}] = \bigoplus_{X_2} \cdots \bigoplus_{X_n} \bigotimes_{i \in [n]} \mathrm{A_i}[X_i, X_{i+1}]$$

where each lifting function $g_{X_j}(X_j)$ maps any key to payload $\mathbf{1} \in \mathcal{R}$

Example: Product of Four Matrices

$$A[X_1, X_5] = \bigoplus_{X_2} \bigoplus_{X_3} \bigoplus_{X_4} \bigotimes_{i \in [4]} A_i[X_i, X_{i+1}]$$ (each A_i encodes a $p \times p$ matrix)

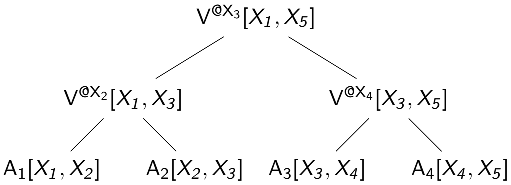


$$V^{@X_2} = \bigoplus_{X_2} A_1 \otimes A_2$$
$$V^{@X_4} = \bigoplus_{X_4} A_3 \otimes A_4$$
$$V^{@X_3} = \bigoplus_{X_3} V^{@X_2} \otimes V^{@X_4}$$

View computation time: $\mathcal{O}(p^3)$

Example: Product of Four Matrices

$$\mathrm{A}[X_1, X_5] = \bigoplus_{X_2} \bigoplus_{X_3} \bigoplus_{X_4} \bigotimes_{i\in[4]} \mathrm{A_i}[X_i, X_{i+1}]$$ (each A_i encodes a $p \times p$ matrix)



$$V^{@X_2} = \bigoplus_{X_2} A_1 \otimes A_2$$
$$V^{@X_4} = \bigoplus_{X_4} A_3 \otimes A_4$$
$$V^{@X_3} = \bigoplus_{X_3} V^{@X_2} \otimes V^{@X_4}$$

View computation time: $\mathcal{O}(p^3)$

Propagation of $\delta\mathrm{A}_2[X_2, X_3]$:

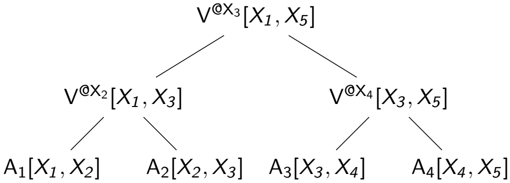
$$\delta\mathrm{V}^{@X_2}[X_1, X_3] = \bigoplus_{X_2} \delta\mathrm{A}_2[X_2, X_3] \otimes \mathrm{A}_2[X_2, X_3]$$ (time $\mathcal{O}(p)$)

$$\delta\mathrm{V}^{@X_3}[X_1, X_5] = \bigoplus_{X_3} \delta\mathrm{V}^{@X_2}[X_1, X_3] \otimes \mathrm{V}^{@X_4}[X_3, X_5]$$ (time $\mathcal{O}(p^2)$)

Further propagation of delta requires $\mathcal{O}(p^3)$ time.

Example: Product of Four Matrices

$$\mathrm{A}[X_1, X_5] = \bigoplus_{X_2} \bigoplus_{X_3} \bigoplus_{X_4} \bigotimes_{i\in[4]} \mathrm{A_i}[X_i, X_{i+1}] \text{ (each } A_i \text{ encodes a } p \times p \text{ matrix)}$$



$$V^{@X_2} = \bigoplus_{X_2} A_1 \otimes A_2$$
$$V^{@X_4} = \bigoplus_{X_4} A_3 \otimes A_4$$
$$V^{@X_3} = \bigoplus_{X_3} V^{@X_2} \otimes V^{@X_4}$$

View computation time: $\mathcal{O}(p^3)$

Propagation of a factorizable update $\delta\mathrm{A}_2[X_2, X_3] = \mathrm{u}[X_2] \otimes \mathrm{v}[X_3]$:

$$\delta\mathrm{V}^{@\mathrm{X}_2}[X_1, X_3] = \underbrace{\left(\bigoplus_{X_2} \mathrm{A}_1[X_1, X_2] \otimes \mathrm{u}[X_2]\right)}_{\mathrm{u}_2[X_1]} \otimes \mathrm{v}[X_3] \text{ (time } \mathcal{O}(p^2))$$

$$\delta\mathrm{V}^{@\mathrm{X}_3}[X_1, X_5] = \mathrm{u}_2[X_1] \otimes \left(\bigoplus_{X_3} \mathrm{v}[X_3] \otimes \mathrm{V}^{@\mathrm{X}_4}[X_3, X_5]\right) \text{ (time } \mathcal{O}(p^2))$$

Summary: Factorized IVM

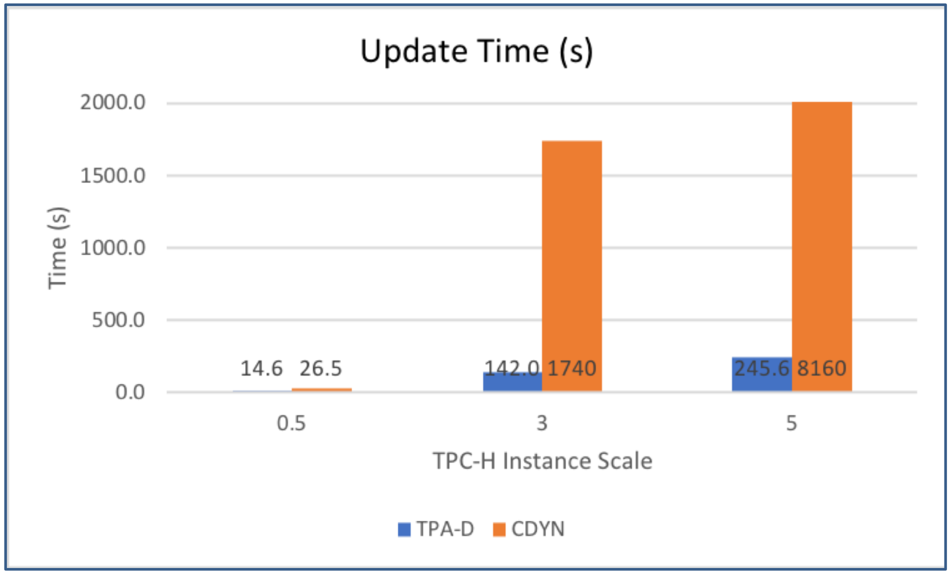
- Framework for unified IVM of in-database analytics
  - Captures many application scenarios via tasks-specific rings
- Based on 3 shades of factorization
  - Factorized query evaluation
  - Factorized representation of query results
  - Factorized updates
- Performance: Up to 2 OOM faster and 4 OOM less memory than state-of-the-art IVM techniques

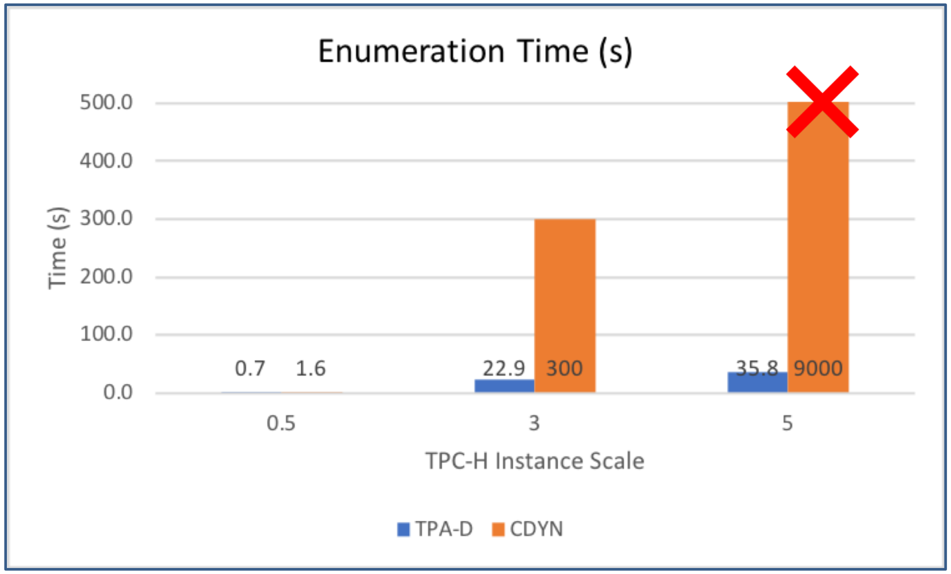
Outline

- Part I: Introduction
- Part II: Main Algorithmic Ideas in Dynamic Query Processing: Traditional IVM and Recent Advances
- Part III: Generalizations to Arbitrary Ring Structures
- **Part IV: Dynamic Query Processing in Big Data Frameworks**
- Part V: Outlook

Are Centralized Algorithms Scalable ?

Distributed Dynamic Yannakakis (TPA-D) vs Centralized Dynamic Yannakakis (CDYN)





Setup: 5 machines, 24GB RAM, 8 Cores / 16 Threads

0.5 = 2 million tuples; 3 = 20.5 million tuples; 5 = 34 million tuples

Why Distributed Streaming Frameworks?

- Efficiently process streams of big data
  - Data too big to fit into the memory of one machine
  - Centralized approaches are not efficient enough to process large data
- Add recovery to faults
  - Why ? Distributed Computations + Messages
  - Failure ➔ Too expensive to start recomputing from the beginning

Distributed Streaming Frameworks

STORM

Flink

Spark Streaming

HERON

Millwheel
Google Dataflow

Trill

samza

S-Store

PULSAR

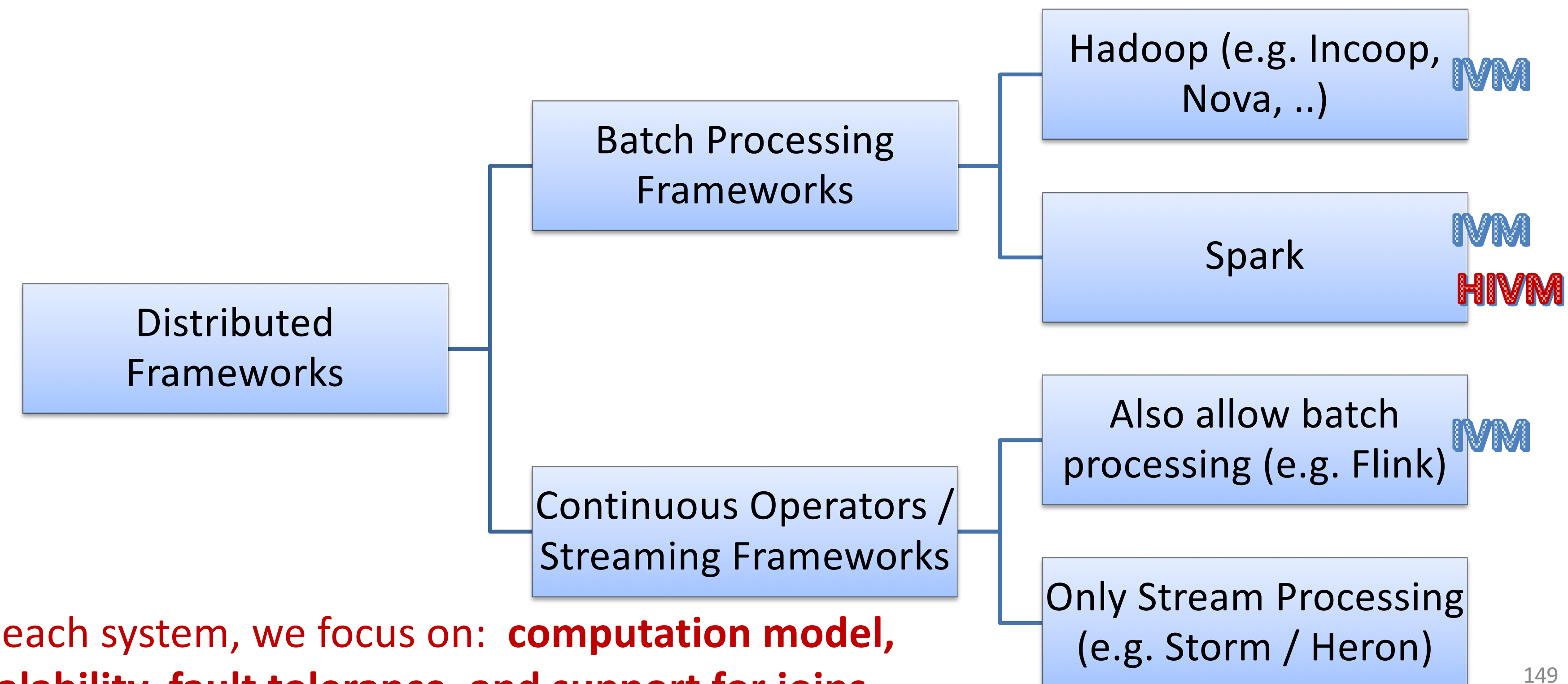
Main Objectives

- Supporting complex continuous queries ?
- Low latency
- Scalability
- Fault tolerance

(Low latency, Scalability, Fault tolerance) → Distributed streaming frameworks

Categories of Frameworks Supporting Incremental Processing of Big Data

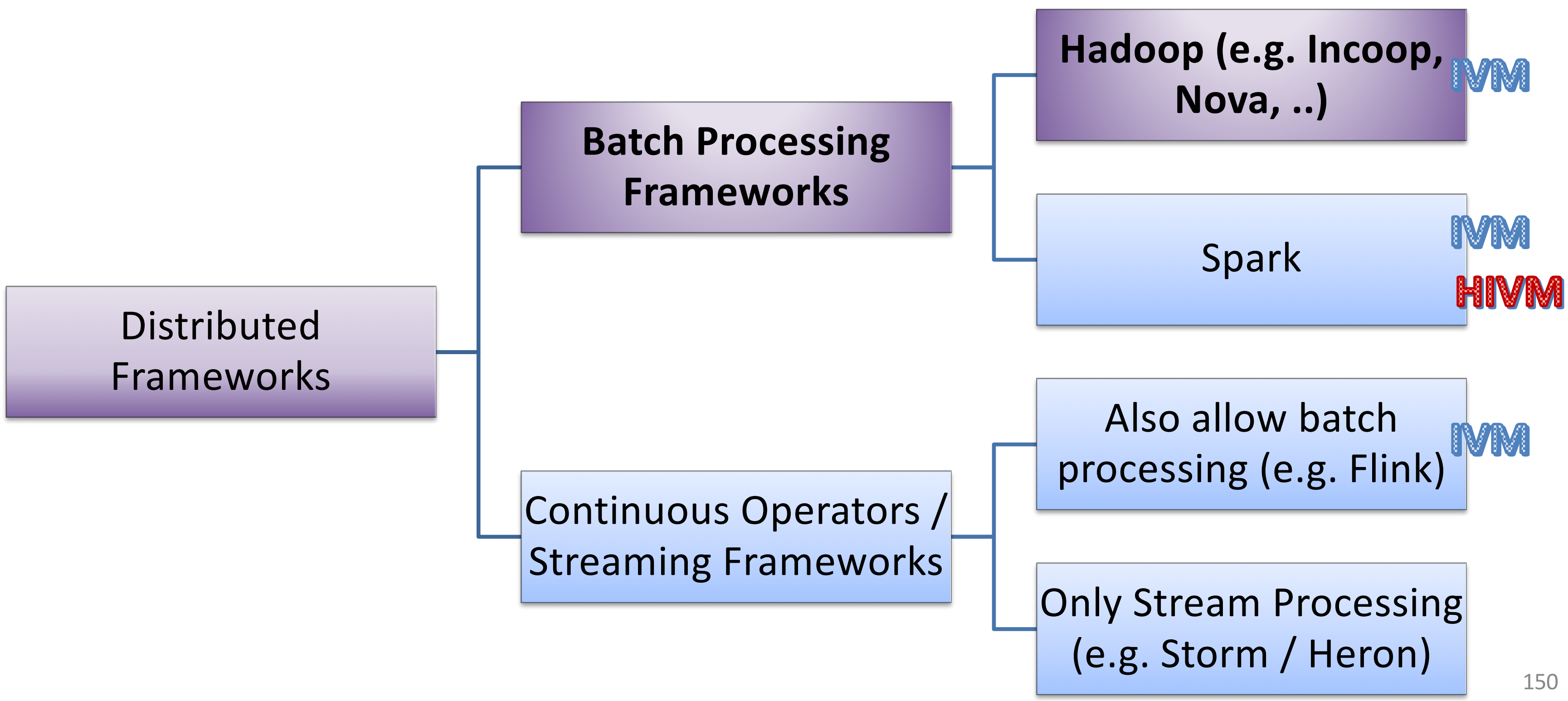
(Batch vs Stream Based Runtime Engine)



For each system, we focus on: **computation model, scalability, fault tolerance, and support for joins**

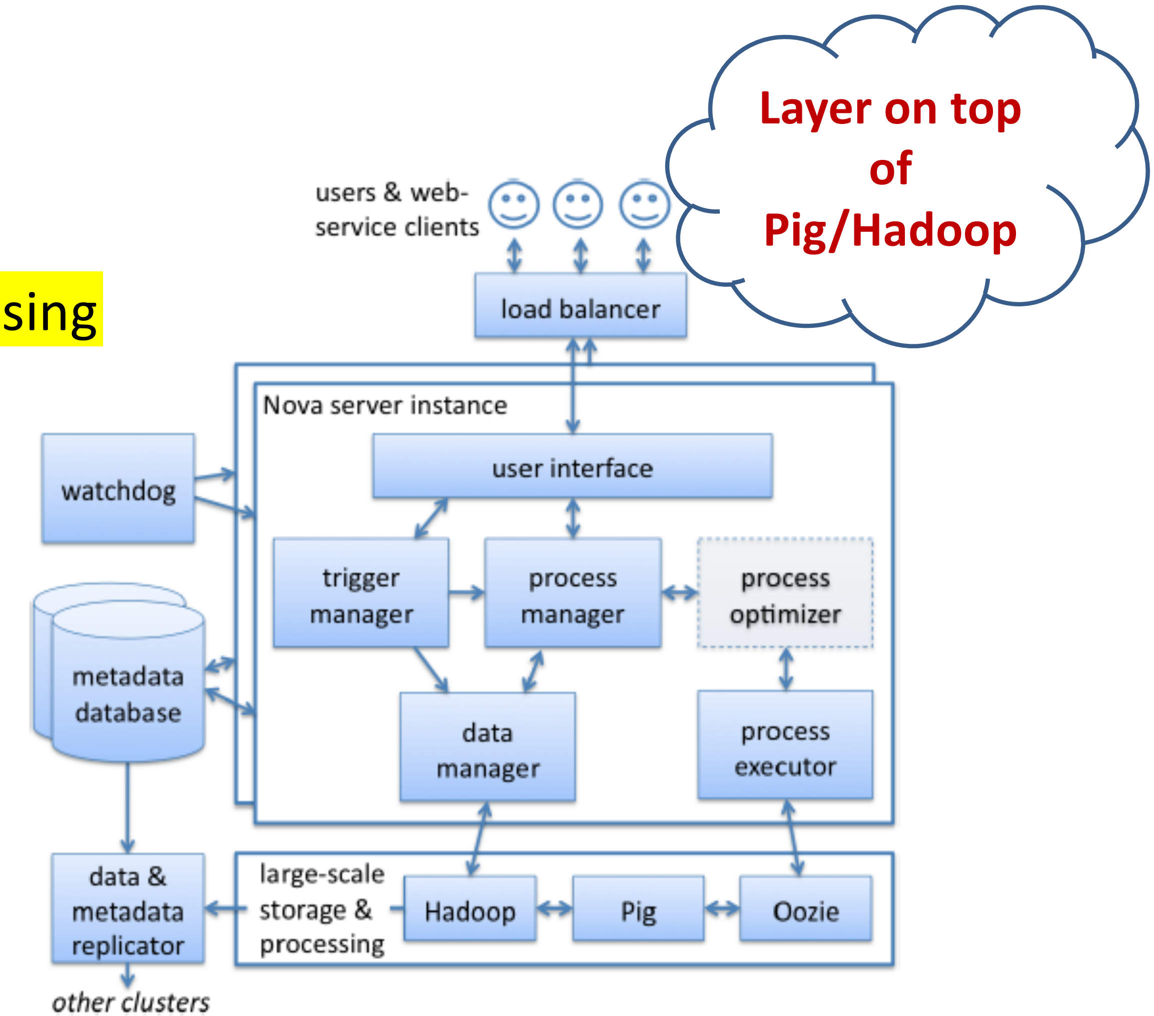
Categories of Frameworks Supporting Incremental Processing of Big Data

(Batch vs Stream Based Runtime Engine)



Nova [SIGMOD'11]

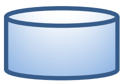
- Goals:
  - IVM • Dynamic query processing
  - Scheduling
  - Optimizations

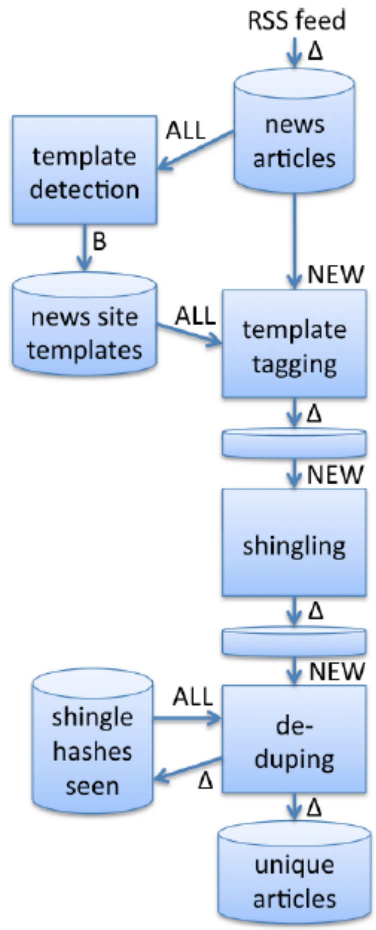


[SIGMOD'11] Christopher Olston et al. 2011. Nova: continuous pig/hadoop workflows. In Proc. ACM SIGMOD Int. Conf. on Management of Data. 1081–1090.

Example Nova Workflow

 Tasks

 Channels / Data Containers



- Data annotated with

All Read complete snapshot

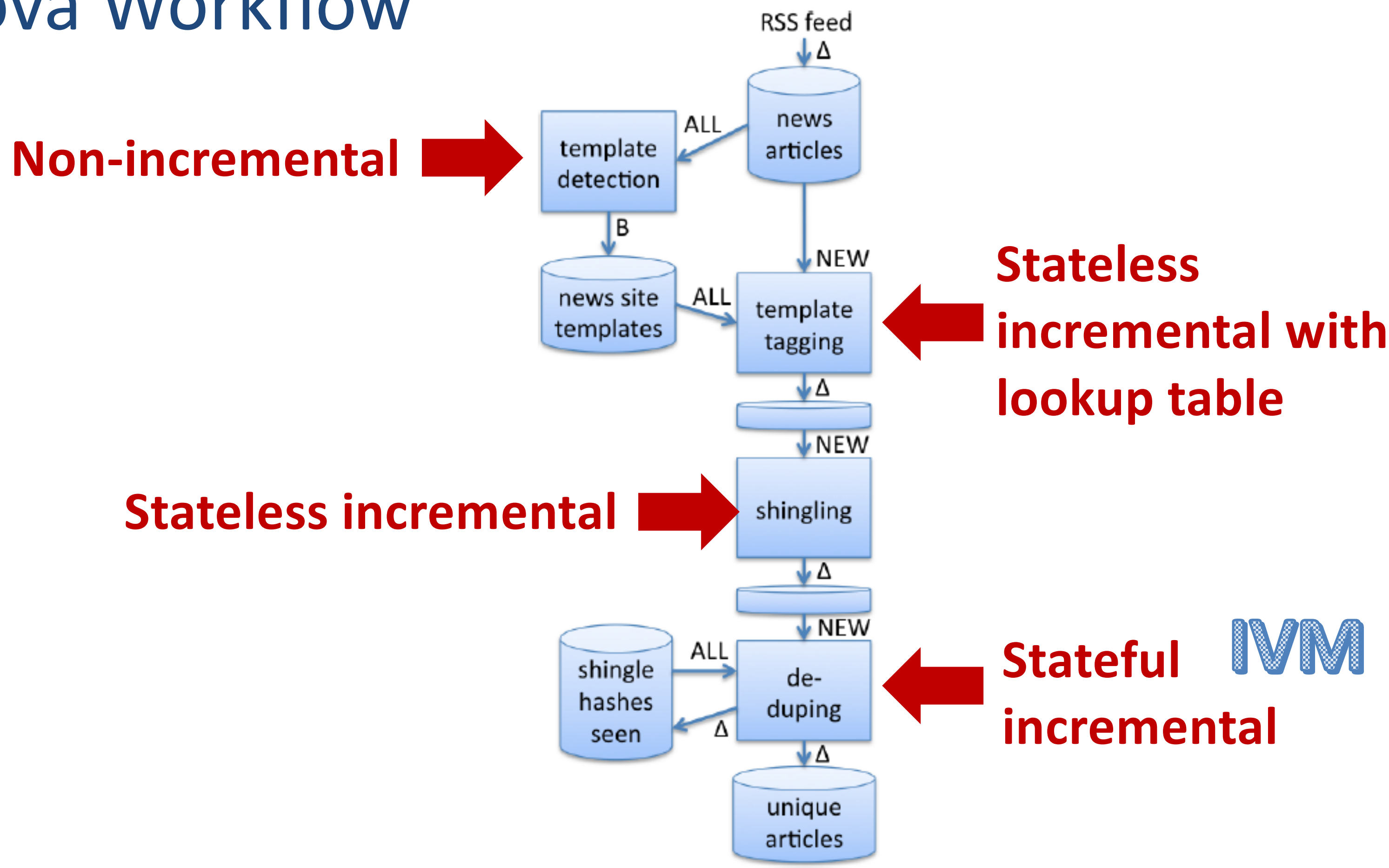
New Read NEW data

B Emit full snapshot

Δ Emit new data that augment existing

[SIGMOD'11] Christopher Olston et al. 2011. Nova: continuous pig/hadoop workflows. In Proc. ACM SIGMOD Int. Conf. on Management of Data. 1081–1090.

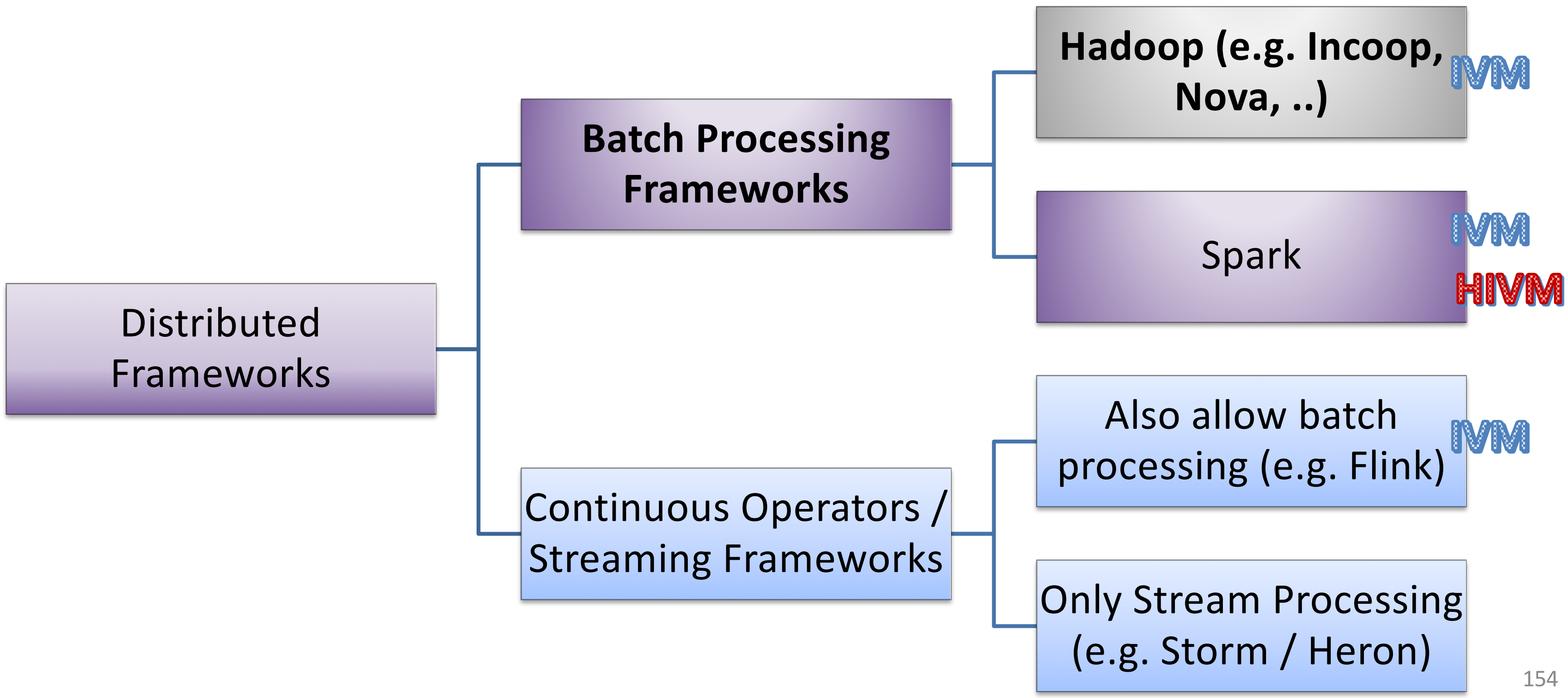
Example Nova Workflow



[SIGMOD’11] Christopher Olston et al. 2011. Nova: continuous pig/hadoop workflows. In Proc. ACM SIGMOD Int. Conf. on Management of Data. 1081–1090.

Categories of Frameworks Supporting Incremental Processing of Big Data

(Batch vs Stream Based Runtime Engine)



Spark Streaming [SOSP'13, SIGMOD'18]

Design objectives

- Process streams of large scale data
- Automatically handle faults and stragglers (**parallel recovery**)
- Integrate streaming with batch and interactive analysis

Credit: https://spark.apache.org/

[SOSP'13] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. 2013. Discretized Streams: Fault-tolerant Streaming Computation at Scale. In Proc. ACM Symp. on Operating Systems Principles (SOSP). 423–438.

[SIGMOD'18] Michael Armbrust et al. 2018. Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark. In Proc. ACM SIGMOD Int. Conf. on Management of Data. 601–613.

Spark Streaming – Computation Model

D-streams:

- Computations => short, stateless, deterministic tasks
- Streamed data => fault-tolerant data structures (RDDs)
- Recomputed deterministically
- Spark engine for processing

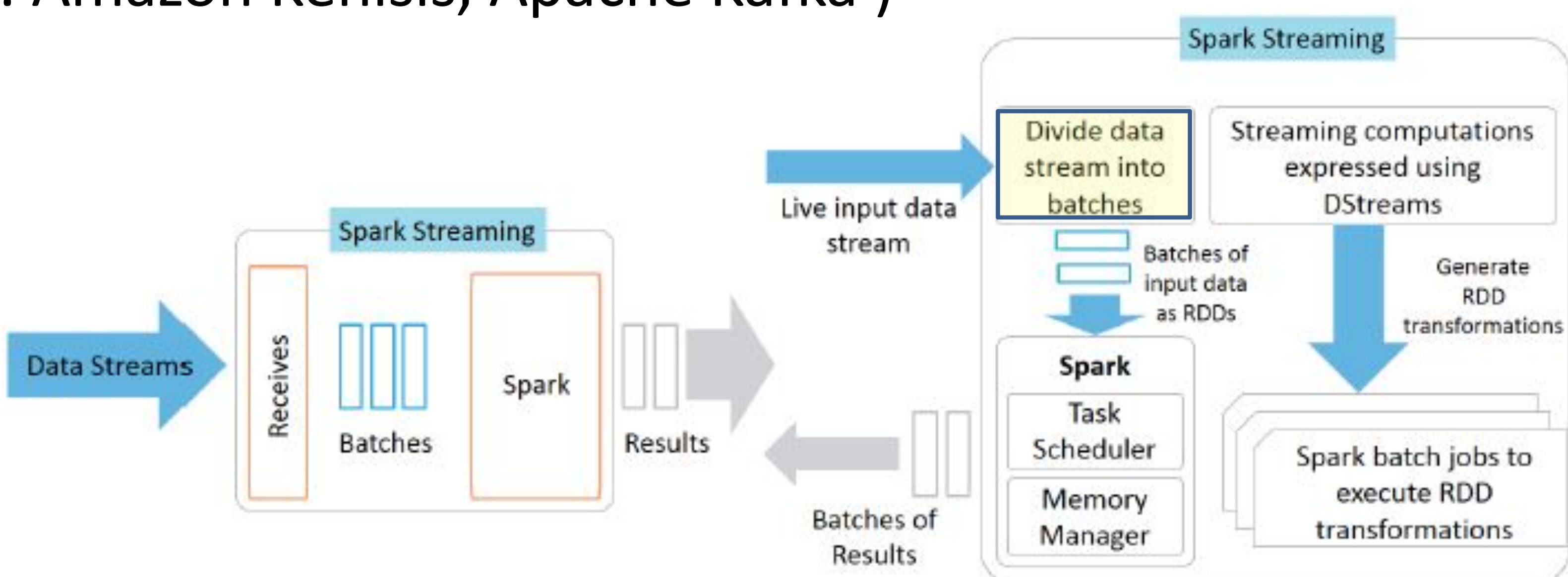
Credit: https://spark.apache.org/

Spark Streaming – Computation Model

Input

- Input should be re-playable

(e.g. Amazon Kenisis, Apache Kafka)

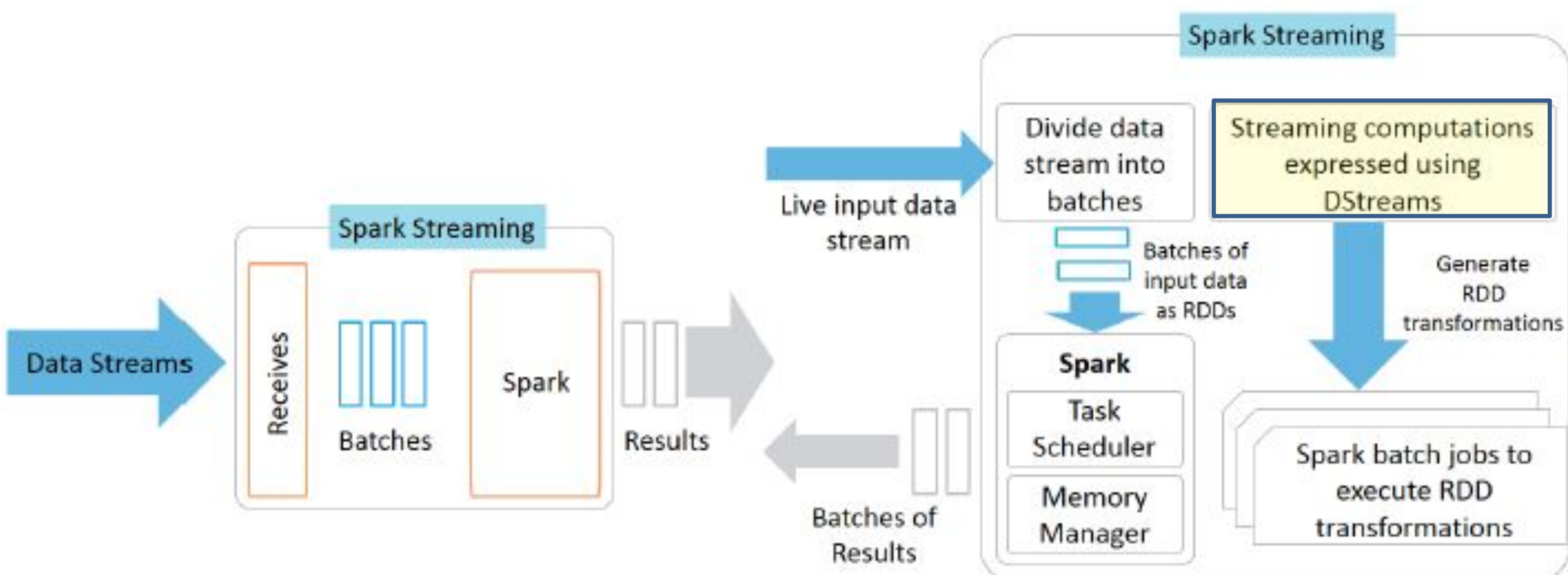


Credit: Michael Armbrust et al. 2018. Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark. In Proc. ACM SIGMOD Int. Conf. on Management of Data. 601–613.

Spark Streaming – Computation Model

Processing

- Time interval completes ➔ spark streaming generate a parallel job (RDD transformation) to operate on the data

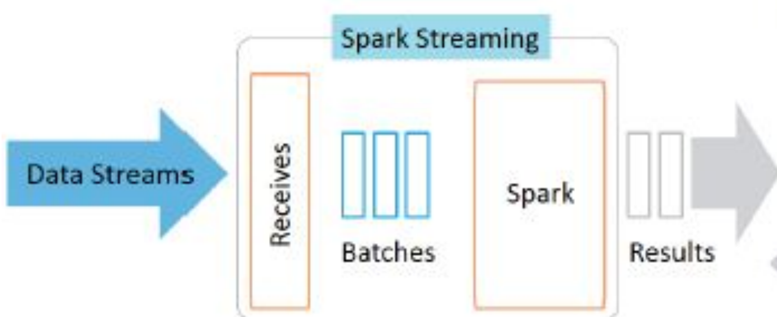


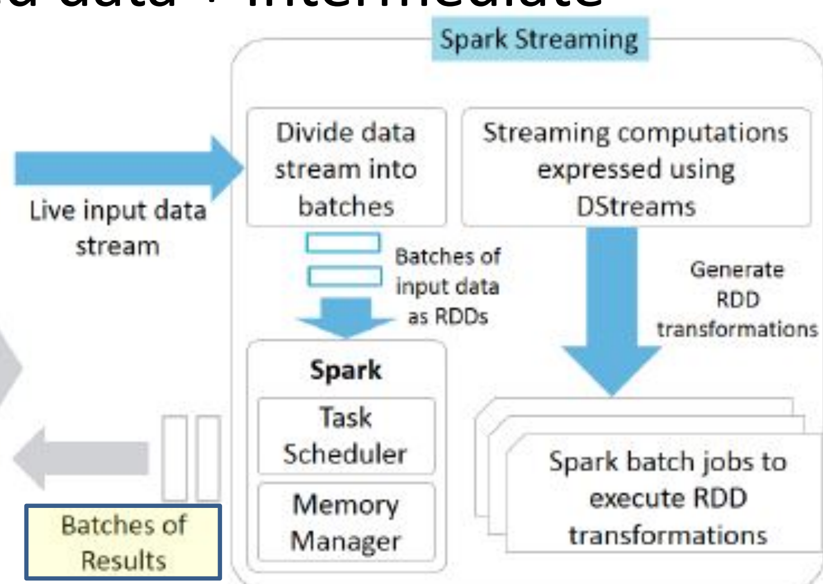
Credit: Michael Armbrust et al. 2018. Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark. In Proc. ACM SIGMOD Int. Conf. on Management of Data. 601–613.

Spark Streaming – Computation Model

Output

- Pushed to another system or stored as an RDD
- Next jobs operate on: Streamed data + Intermediate materialized data

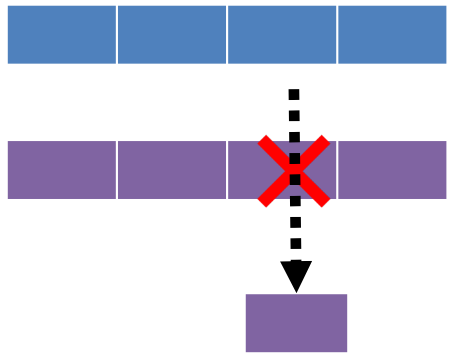




Credit: Michael Armbrust et al. 2018. Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark. In Proc. ACM SIGMOD Int. Conf. on Management of Data. 601–613.

Spark Streaming – Fault Tolerance

- Parallel recovery
  - Streams and intermediate data are stored as RDDs
  - Frequent checkpointing
  - Failure => only missing partitions are recomputed
- Straggler mitigation
  - Run speculative copies of slow tasks
  - Deterministic computations + Idempotent sinks



Spark Streaming – Incremental Processing of Queries

- Stateless ➔ fault tolerance
- Maintaining a State ➔ Arbitrary Stateful Operators

Define

```
// Define an update function that simply tracks the
// number of events for each key as its state, returns
// that as its result, and times out keys after 30 min.
def updateFunc(key: UserId, newValues: Iterator[Event],
               state: GroupState[Int]): Int = {
  val totalEvents = state.get() + newValues.size()
  state.update(totalEvents)
  state.setTimeoutDuration("30 min")
  return totalEvents
}
```

Use

```
// Use this update function on a stream, returning a
// new table lens that contains the session lengths.
lens = events.groupByKey(event => event.userId)
             .mapGroupsWithState(updateFunc)
```

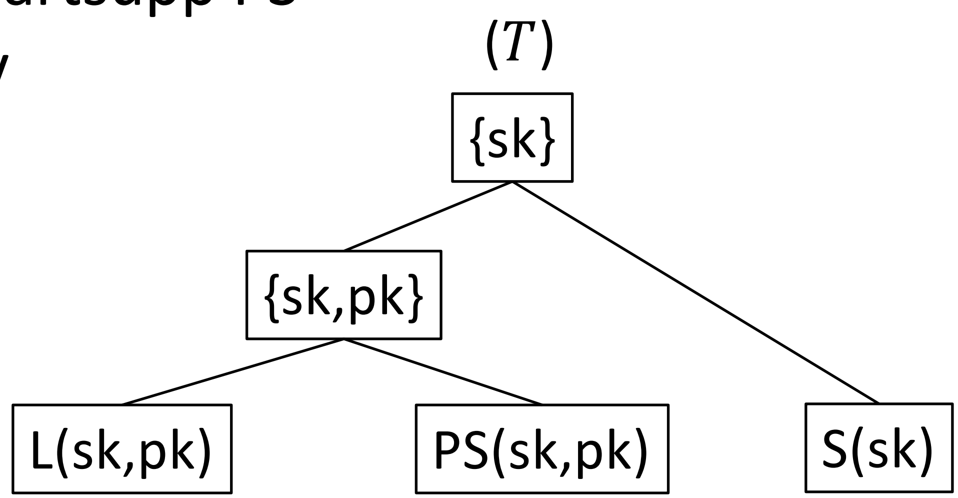
Spark Streaming – Support for Join Queries

- Allowed Joins:
  - Join between two data streams
  - Join between a data stream and a static dataset
- What about joining n datasets?
  - n-1 pair joins
  - Materialize intermediate join results
  - Dynamic query processing ?

Running Example

Query from TPC-H Benchmark:

SELECT *
FROM lineitem L, supplier S, partsupp PS
WHERE L.suppkey = S.suppkey
and L.suppkey = PS.suppkey
and L.partkey = PS.partkey



Example: Spark Implementation

```
val dsL = spark
  .readStream
  .format( source = "kafka")
  .option("kafka.bootstrap.servers", "localhost:9093")
  .option("subscribe", "li-topic")
  .load
  .select( col = "value",  cols = "timestamp")
  .as[(String,Timestamp)]
  .map(data => L(data._1, data._2))
  .withWatermark( eventTime = "lTimestamp",  delayThreshold = "10 seconds")
val dsPS = spark
  .readStream
  .format( source = "kafka")
  .option("kafka.bootstrap.servers", "localhost:9094")
  .option("subscribe", "ps-topic")
  .load
  .select( col = "value",  cols = "timestamp")
  .as[(String,Timestamp)]
  .map(data => PS(data._1, data._2))
  .withWatermark( eventTime = "psTimestamp",  delayThreshold = "10 seconds")
val dsS = spark
  .readStream
  .format( source = "kafka")
  .option("kafka.bootstrap.servers", "localhost:9095")
  .option("subscribe", "s-topic")
  .load
  .select( col = "value",  cols = "timestamp")
  .as[(String,Timestamp)]
  .map(data => S(data._1, data._2))
  .withWatermark( eventTime = "sTimestamp",  delayThreshold = "10 seconds")
```

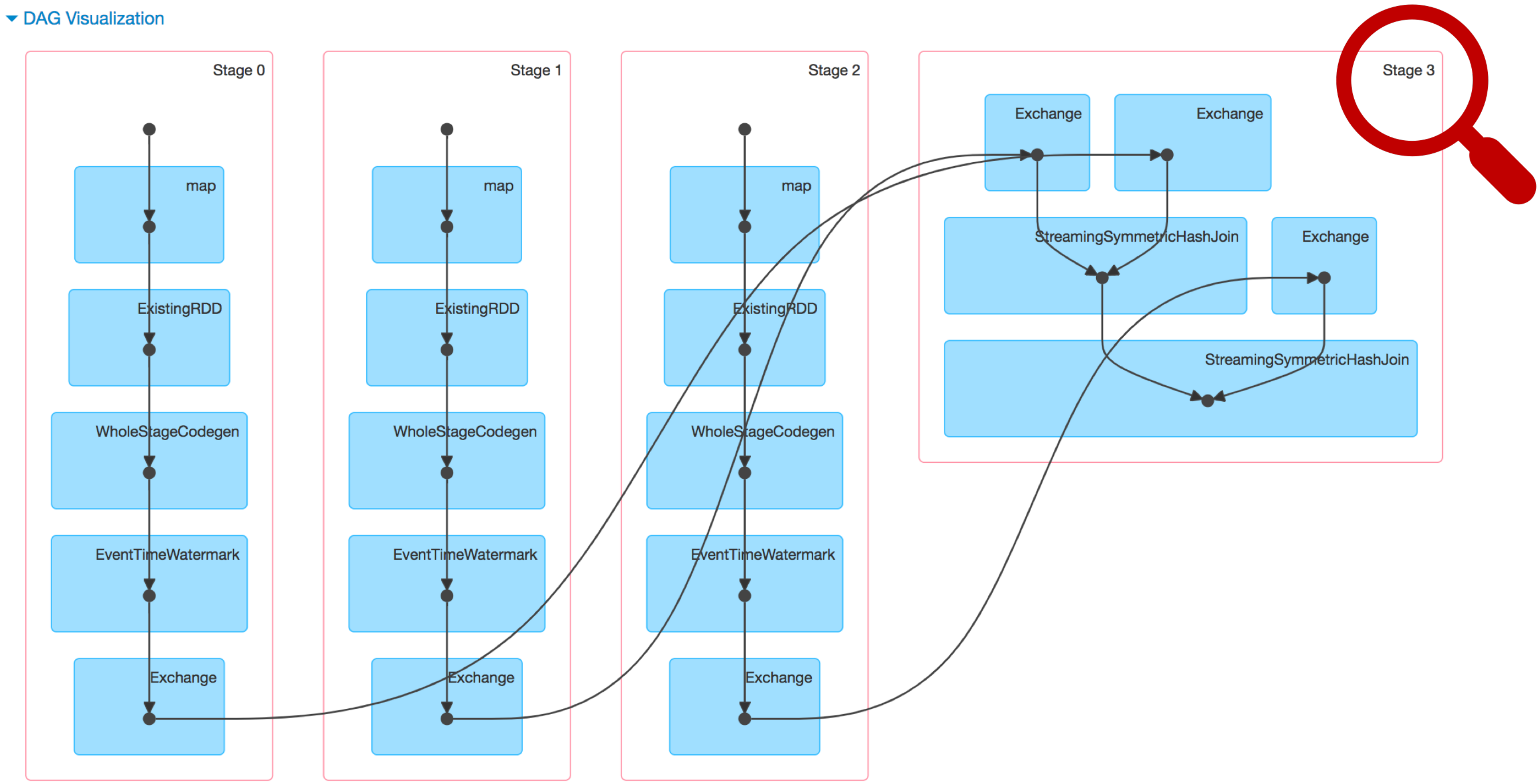
Define datasets + Define schema

Two step join

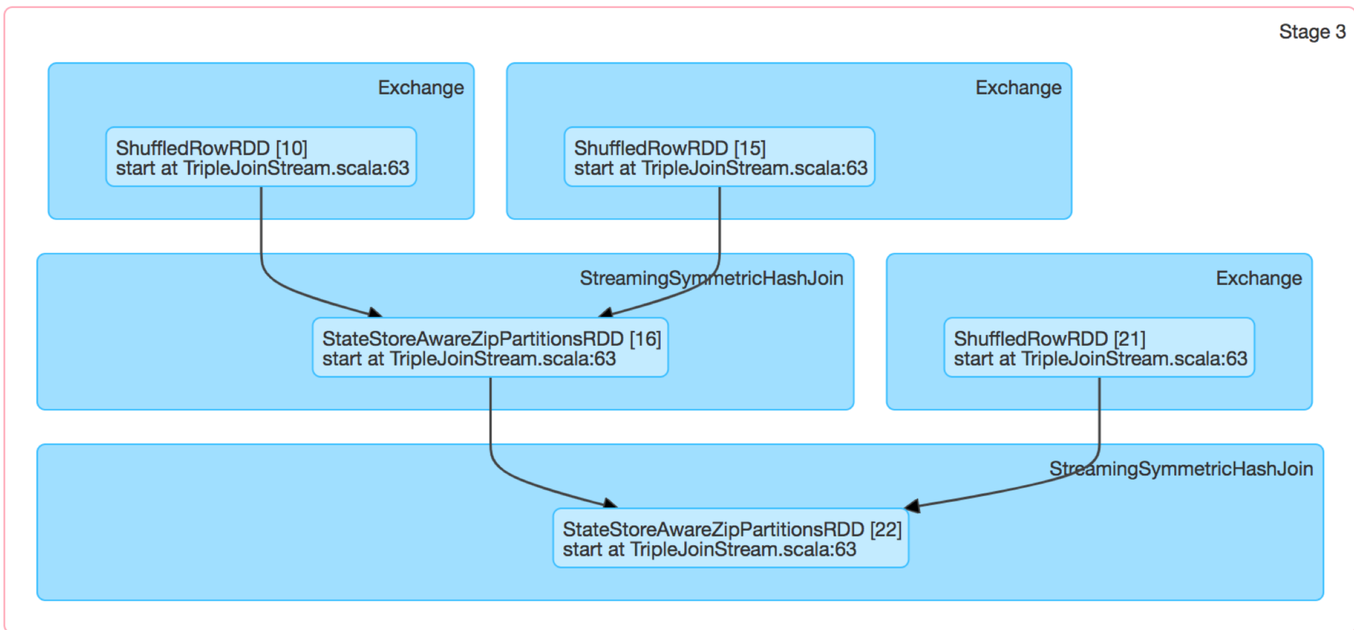
```
val tripleJoin =
  dsL
    .join(dsPS,expr( expr = """lSK = psSK"""),  joinType = "inner")
    .join(dsS, expr( expr = """lSK = sSK"""),  joinType = "inner")
```

Acknowledgment: Thanks to Omar Shahbaz Khan for preparing this code.

Example: Spark Execution Plan



Example: Spark Execution Plan



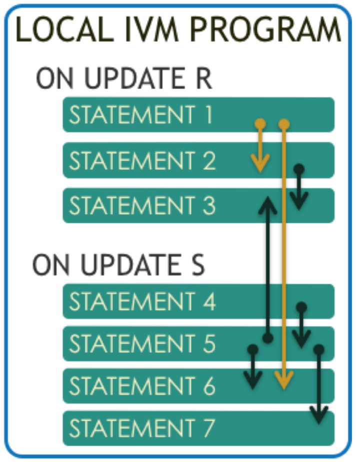
IVM ?

Distributed Higher-Order Incremental View Maintenance [SIGMOD’16]

TOASTER

HIVM + Spark Streaming

- On top of Spark: synchronous execution model
- Views ➔ local or distributed (partitioned)
- Parallel updates of views ➔ dependency among programs
- Batching updates (well in some of the cases !)



[SIGMOD’16] Milos Nikolic, Mohammad Dashti, and Christoph Koch. 2015. How to Win a Hot Dog Eating Contest: Distributed Incremental View Maintenance with Batch Updates. In Proc. ACM SIGMOD Int. Conf. on Management of Data. 511–526.

Distributed HIVM Solution

- Annotate each node in the query plan with **location tags**
  - LOCAL, PARTITIONED BY KEY, RANDOM
- Insert communication operations into query plans: **location transformers**

- **Holistic optimization** to minimize communication cost

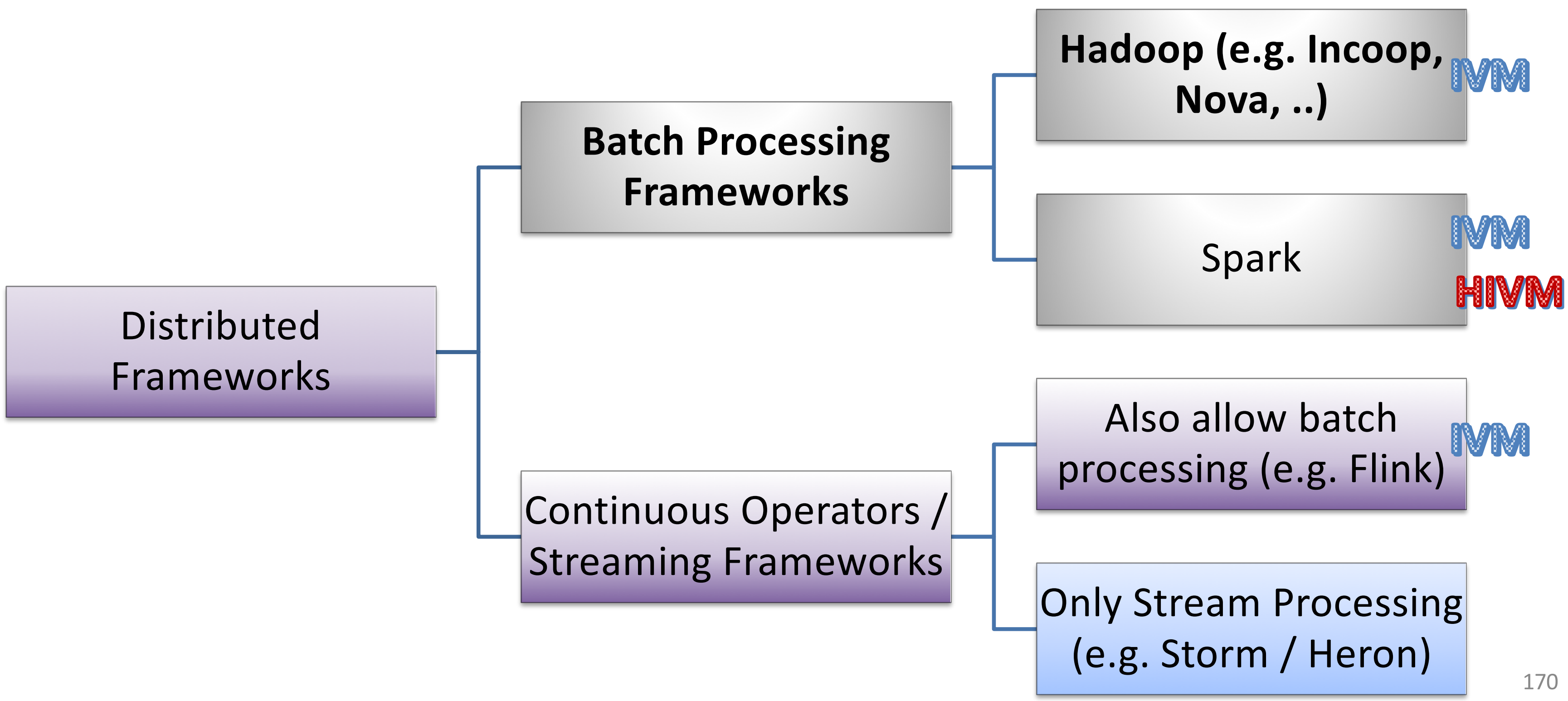
[SIGMOD'16] Milos Nikolic, Mohammad Dashti, and Christoph Koch. 2015. How to Win a Hot Dog Eating Contest: Distributed Incremental View Maintenance with Batch Updates. In Proc. ACM SIGMOD Int. Conf. on Management of Data. 511–526.

Distributed HIVM – Fault Tolerance

- Leverage Spark fault tolerance mechanism
- Periodic checkpointing in trigger program

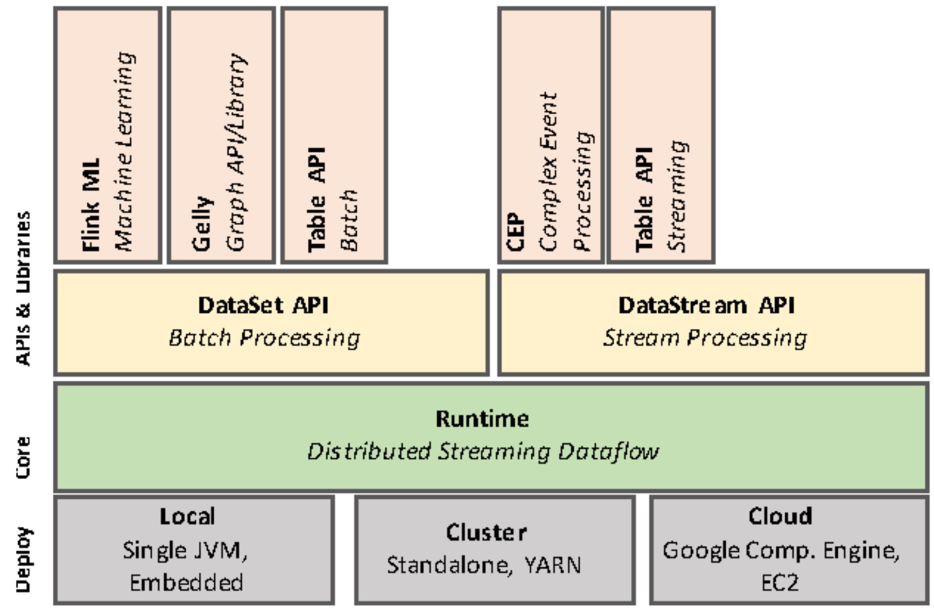
Categories of Frameworks Supporting Incremental Processing of Big Data

(Batch vs Stream Based Runtime Engine)



Flink [DE'15] (Based on Stratosphere[VLDBJ'14])

- No distinction between stream processing and batch processing
- However, core is a distributed streaming datflow engine



[DE'15] Carbone, Paris, et al. 2015. "Apache flink: Stream and batch processing in a single engine." *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36.4.

[VLDBJ'14] Alexander Alexandrov et al. 2014. The Stratosphere Platform for Big Data Analytics. The VLDB Journal 23, 6 (Dec. 2014), 939–964.

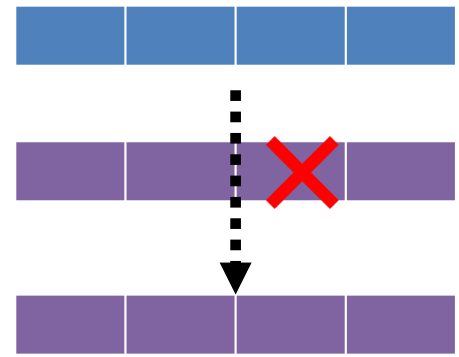
Flink – Computation Model

- Runtime program: DAG of stateful operators connected with data streams
  - **Stateful operators**: parallelized into one or more parallel instance (subtask)
  - **Streams**: partitioned into one or more stream partition (one per subtask)
- Static program ➔
  - finite stream
  - order of records is not important

Flink – Fault Tolerance

- Consistency guarantee: exactly-once-processing
- Input data: persistent + re-playable
- Frequent checkpointing
  - Allow partial re-execution
  - **Distributed consistent snapshots**:

    state of the operators + current position of the input stream
- Failure occurs ➔
  - Revert to latest snapshot
  - Redo the computation



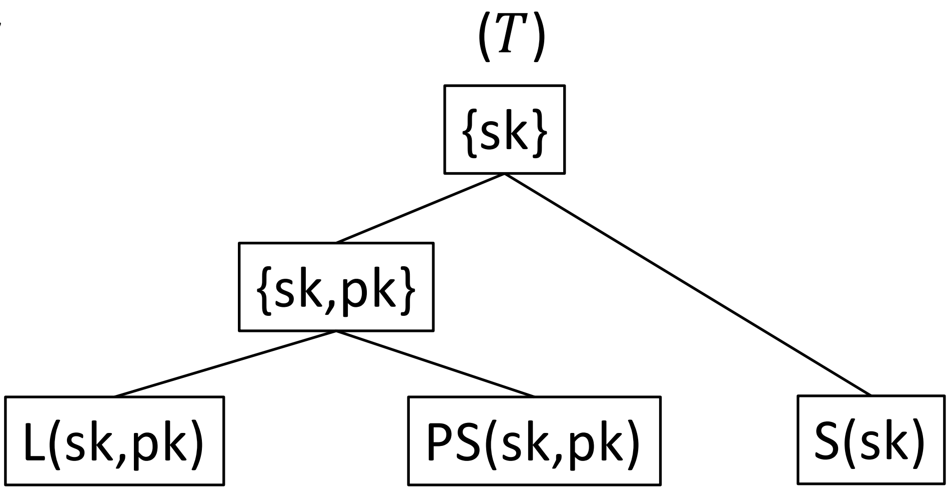
Flink – Support for Join Queries

- Join two streams
  - Window
  - Interval
- Join in Batch API allows custom join functions

Running Example

Query from TPC-H Benchmark:

SELECT *
FROM lineitem L, supplier S, partsupp PS
WHERE L.suppkey = S.suppkey
and L.suppkey = PS.suppkey
and L.partkey = PS.partkey



Example: Flink Implementation

Apache Flink

Two step join

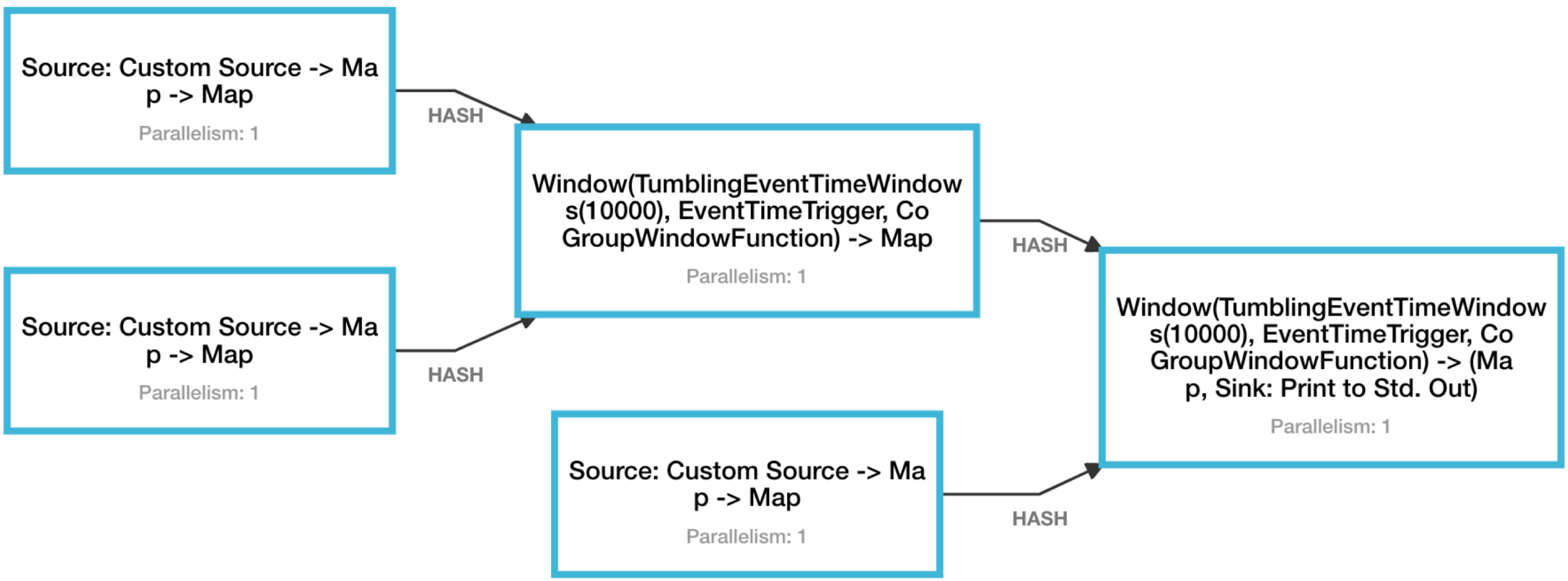
```
val doubleJoinStream =
  lStream
    .join(psStream)
    .where(l => l.lSK)
    .equalTo(p => p.psSK)
    .window(TumblingEventTimeWindows.of(Time.seconds( seconds = 10)))
    .apply((l,p) => (l,p))

doubleJoinStream.map(j => j._1.toString + " | " + j._2.toString).print

val tripleJoinStream =
  doubleJoinStream
    .join(sStream)
    .where(_._1.lSK)
    .equalTo(_.sSK)
    .window(TumblingEventTimeWindows.of(Time.seconds( seconds = 10))).apply((j,s) => (j._1,j._2,s))
```

Acknowledgment: Thanks to Omar Shahbaz Khan for preparing this code.

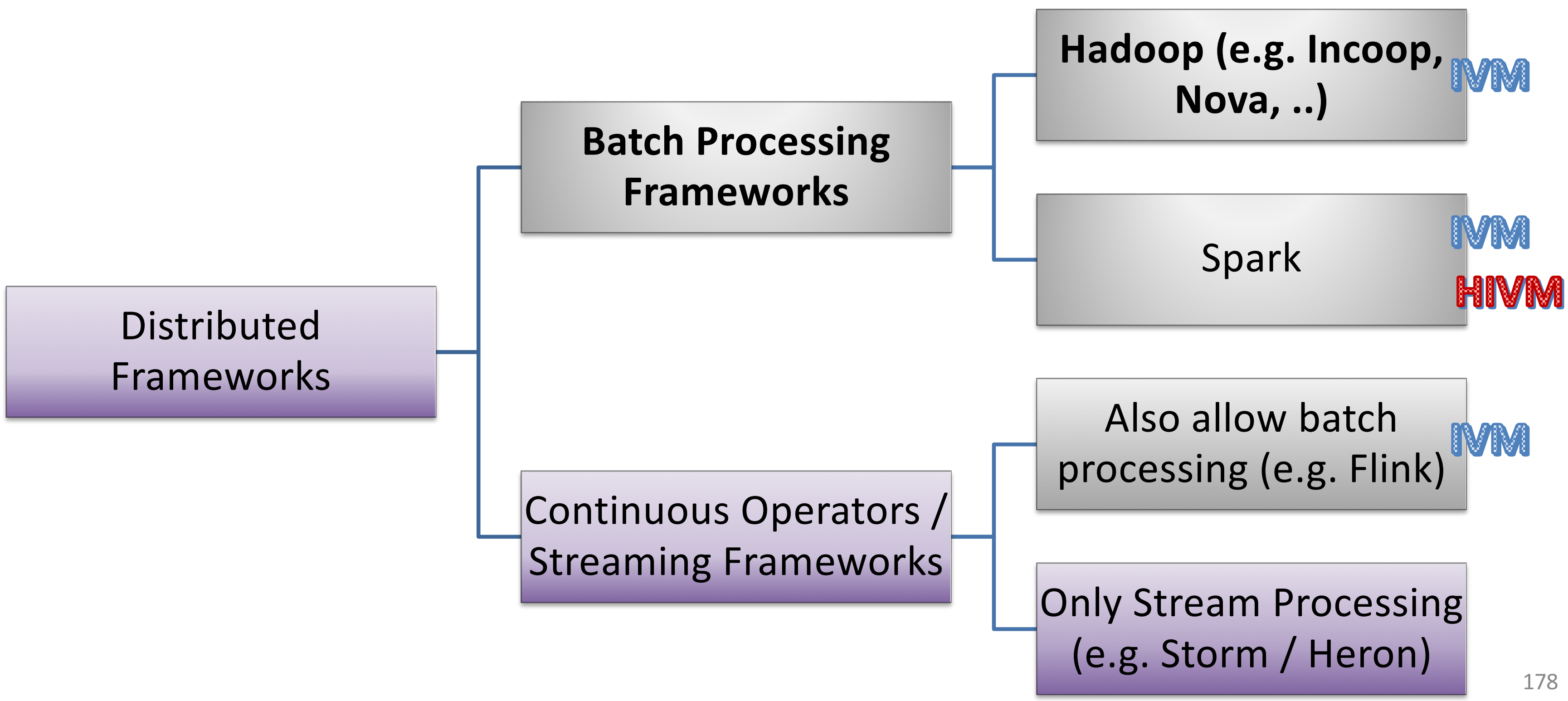
Example: Flink Execution Plan



IVM ?

Categories of Frameworks Supporting Incremental Processing of Big Data

(Batch vs Stream Based Runtime Engine)



Heron [SIGMOD'15, ICDE'17]

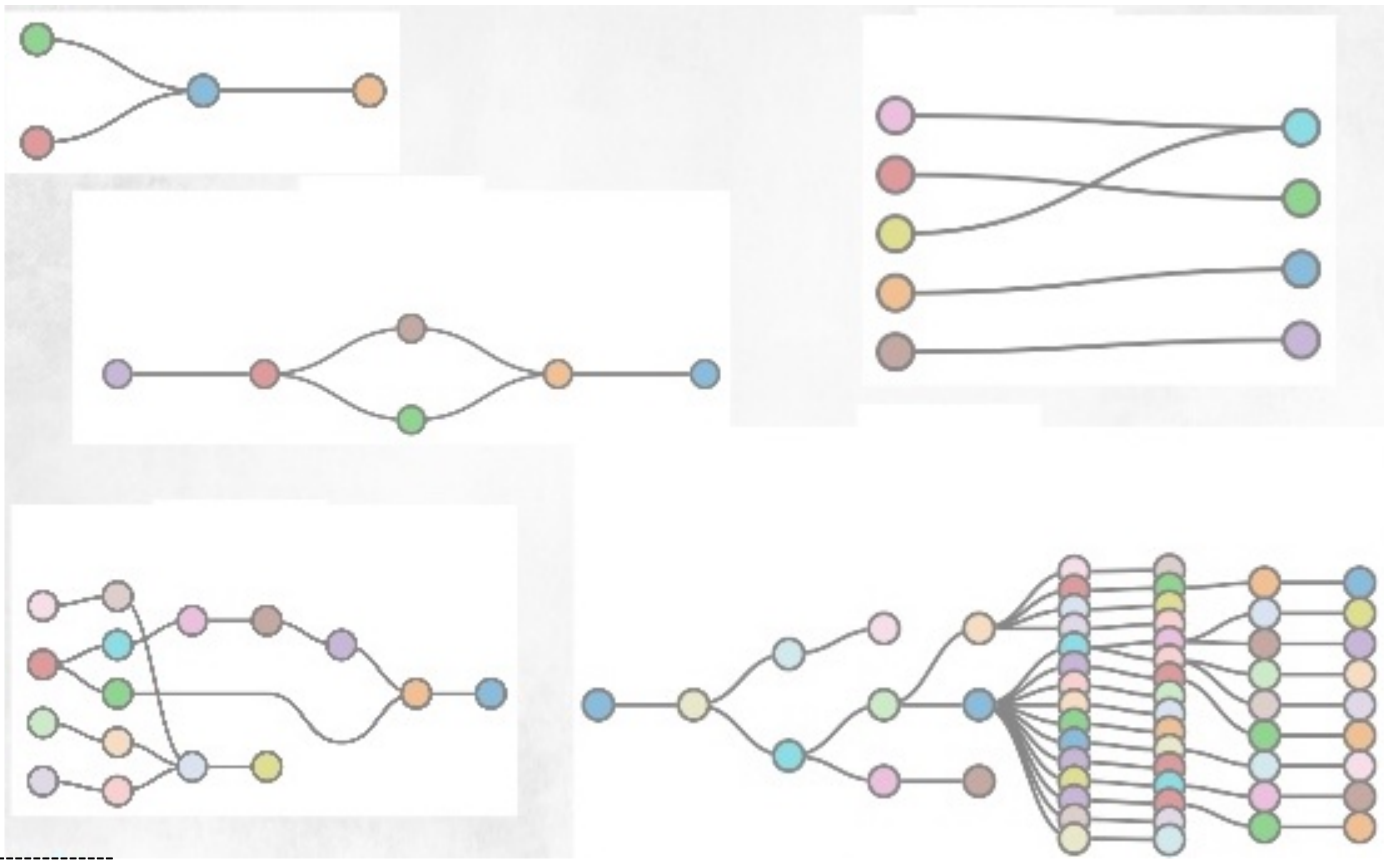
- Streaming engine
- Real-time performance for big data
- Based on + Same programming model of Storm [SIGMOD'14]

[SIGMOD'14] Ankit Toshniwal et al. 2014. Storm@Twitter. In Proc. ACM SIGMOD Int. Conf. on Management of Data. 147–156.
[SIGMOD'15] Sanjeev Kulkarni et al. 2015. Twitter Heron: Stream Processing at Scale. In Proc. ACM SIGMOD Int. Conf. on Management of Data. 239–250.
[ICDE'17] Fu, Maosong et al. 2017. Twitter Heron: Towards Extensible Streaming Engines. In Proc. *IEEE International Conference on Data Engineering (ICDE).* 1165-1172.

Heron – Computation Model

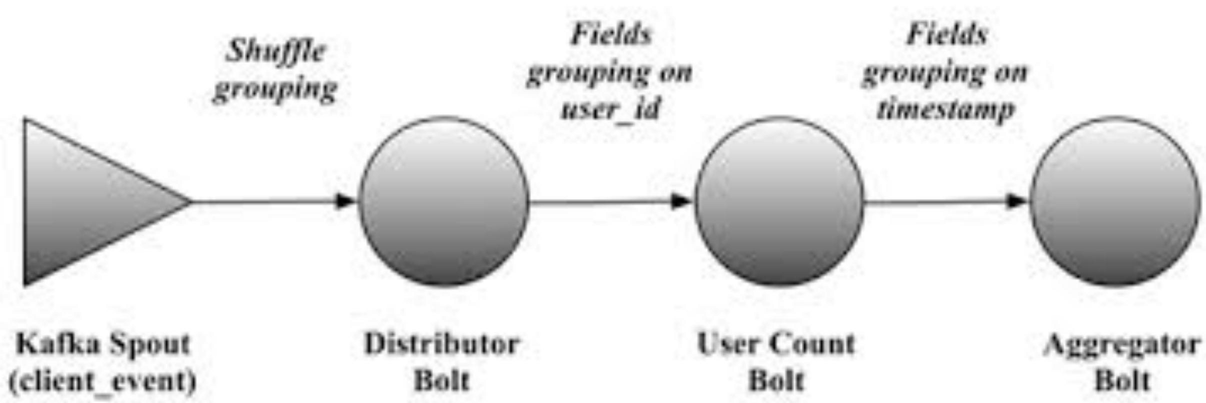
- Queries are represented as **topologies**, which are directed acyclic graphs of spouts and bolts
  - Spout = tuple sources for the topology (e.g. pull data from kafka)
  - Bolt = process data and pass them to next bolt(s)
- Programmer specifies:
  - The number of tasks created for each spout and bolt (degree of parallelism)
  - Data partitioning

Heron Example Topologies



Credit: Real Time Analytics: Algorithms and Systems by Arun Kejariwal (https://apache.github.io/incubator-heron/docs/resources/)

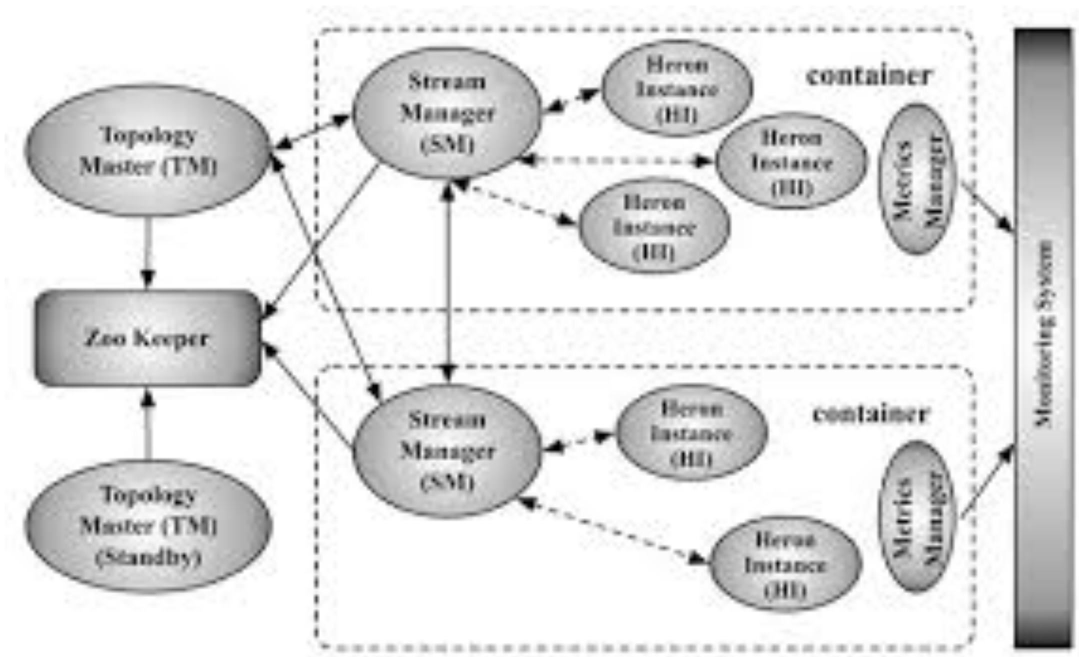
Example Topology : Real Time Active Users



Credit: Fu, Maosong et al. 2017. Twitter Heron: Towards Extensible Streaming Engines. In Proc. *IEEE International Conference on Data Engineering (ICDE).* 1165-1172.

Heron Topology Architecture

- Topologies are submitted to a scheduler “Apache Aurora”, which starts several containers:
  - Topology master
  - General container running:
    - Stream Manager,
    - Metric Manager, and
    - Heron Instances



Credit: Fu, Maosong et al. 2017. Twitter Heron: Towards Extensible Streaming Engines. In Proc. *IEEE International Conference on Data Engineering (ICDE).* 1165-1172.

Heron – Fault Tolerance: Tuples Processing Semantics

- At most once:
  - No tuple is processed more than once
  - Some tuples might be dropped (Not processed by the topology)
- At least once:
  - Each tuple is processed at least once (multiple times happens)
  - Add new bolt “acker” to track the processing of each tuple
  - Developer custom code for state recovery

Heron – Fault Tolerance: Workers

- Topology Master
  - Metadata kept in Zookeeper
  - A standby version is created upon startup in case master fails
- Failure scenarios
  - Death of a Stream Manager or Heron Instances: restarted from within the container
  - Container failure or Machine failure:
    - A new container is started
    - Failure recovery procedure of Stream Manager and Heron Instances

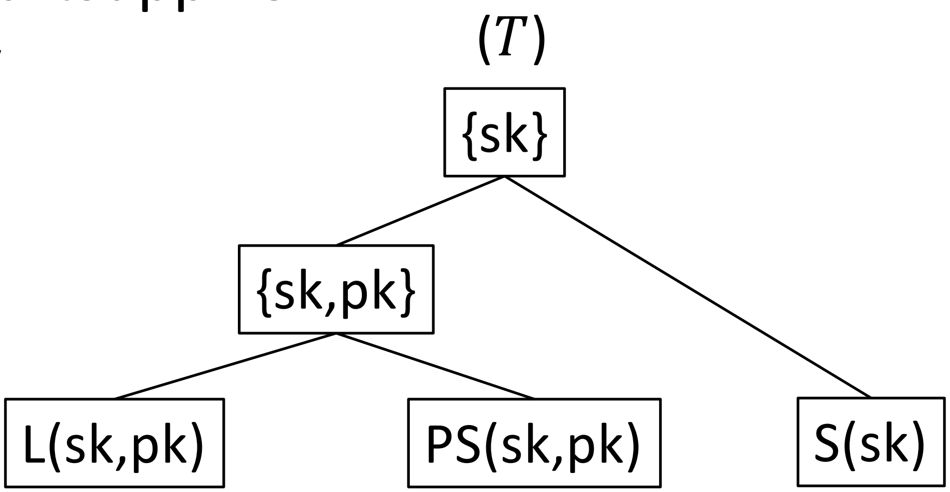
Heron – Support for Join Queries

- Storm SQL integration experimental feature ➔ does not support joins or aggregations
- Heron Streamlet API (beta) ➔ join operations of two streams
- Programmers write topology for applications
  - Advanced incremental view maintenance approach can be implemented as a topology❓

Running Example

Query from TPC-H Benchmark:

SELECT *
FROM lineitem L, supplier S, partsupp PS
WHERE L.suppkey = S.suppkey
and L.suppkey = PS.suppkey
and L.partkey = PS.partkey



Example: Heron/Storm Implementation

```
builder.setSpout( id = "li", lSpout, parallelism_hint = 1)
builder.setSpout( id = "ps", psSpout, parallelism_hint = 1)
builder.setSpout( id = "s", sSpout,  parallelism_hint = 1)

val firstStepJoinBolt = new JoinBolt( sourceId = "li",  fieldName = "lS_key")
  .join( newStream = "ps",  field = "psS_key",  priorStream = "li")
  .select( commaSeparatedKeys = "lS_key, l_obj, ps_obj")
  .withTumblingWindow(new Duration( value = 10, TimeUnit.SECONDS))

builder.setBolt( id = "firstStepJoin", firstStepJoinBolt,  parallelism_hint = 1)
    .fieldsGrouping( componentId = "li", new Fields( fields = "lS_key"))
    .fieldsGrouping( componentId = "ps", new Fields( fields = "psS_key"))

builder.setBolt( id = "liPsBolt", new DoubleJoinBolt).shuffleGrouping( componentId = "firstStepJoin")

val secondStepJoinBolt = new JoinBolt( sourceId = "liPsBolt",  fieldName = "key")
  .join( newStream = "s",  field = "s_key",  priorStream = "liPsBolt")
  .select( commaSeparatedKeys = "key, l_obj, ps_obj, s_obj")
  .withTumblingWindow(new Duration( value = 10, TimeUnit.SECONDS))

builder.setBolt( id = "secondStepJoin", secondStepJoinBolt)
  .fieldsGrouping( componentId = "liPsBolt", new Fields( fields = "key"))
  .fieldsGrouping( componentId = "s", new Fields( fields = "s_key"))

builder.setBolt( id = "liPsSBolt", new TripleJoinBolt).shuffleGrouping( componentId = "secondStepJoin")
```

Two step join

Acknowledgment: Thanks to Omar Shahbaz Khan for preparing this code.

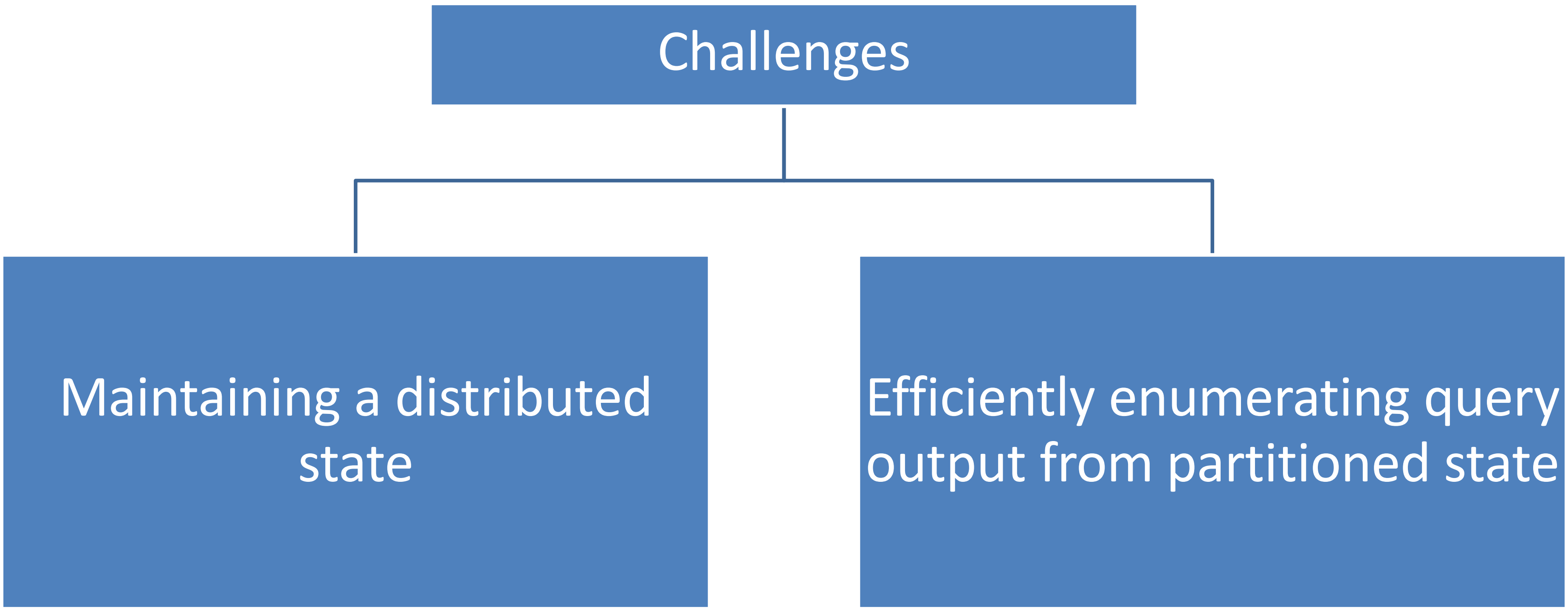
Distributed Streaming Frameworks: Summary

	Spark Streaming	Flink	Storm/Heron
Input stream Condition	Persistent + Re-playable	Persistent + Re-playable	No condition
Input stream	D-Streams	Tuples	Tuples
Computation model	Stateless Tasks	Topology of Stateful Operations	Topology of Stateful bolts
Runtime Engine	Batch Processing	Stream Processing	Stream Processing
Support Batching	✓	✓	✗
Fault tolerance	Exactly–once Parallel and partial Recovery	Exactly–once Parallel from Snapshot	At-most–once or At-least-once (programmer)

Outline

- Part I: Introduction
- Part II: Main Algorithmic Ideas in Dynamic Query Processing: Traditional IVM and Recent Advances
- Part III: Generalizations to Arbitrary Ring Structures
- Part IV: Dynamic Query Processing in Big Data Frameworks
- **Part V: Outlook**

Parallel and Distributed Dynamic Query Processing



Challenge: Maintaining Distributed State

- Query state can be:
  - Materialized views (e.g. IVM, HIVM)
  - Custom query representation (e.g. Dynamic Yannakakis, F-IVM)
- Assumptions:
  - Stored in memory → otherwise, promised performance is not guaranteed
  - Optimized for small memory footprint
  - Processing on single core
  - Streamed tuples trigger updates to this state

However ….

- Frameworks such as Spark:
  - Agnostic to data content ➔ shuffle data through network for join
  - Stateless operators

  Therefore
  - Distributed HIVM ➔

    custom partitioning + immutable RDDs periodically saved to HDFS
- Frameworks such as Storm:
  - Stateful operators
  - Implementing operators (bolts) that maintains intermediate state

Still Need to Address

- Reduce communication cost →

  partitioning; co-locating data

- Exploding state →

  repartitioning; rebalancing; spilling to disk

- Fault tolerance
  - Lost messages between workers
  - Recomputing failed partitions

  leverage existing frameworks; extra coding

Challenge: Enumerating Query Output from Partitioned State

- Consistency
  - Cause: simultaneous update to partitions of state

    maintain timestamps and track them

- Constant delay enumeration promised by Dynamic Yannakakis
  - Cannot be guaranteed: network messages to enumerate the output

    reduce messages between workers;

    new model to describe enumeration that takes into account the communication cost

Conclusion

- For many applications, it is essential to analyze fast evolving big data in real time
- Many algorithmic ideas (single core):
  - Delta queries
  - Storing intermediate results
  - Dealing with Skews
- Generalizing to complex aggregates
- Distributed streaming frameworks ➔ ill support of dynamic query processing
- New challenges: stateful operations yet scalable; consistency