Joining Infinity — Windowless Stream Processing with Flink

Sanjar Akhmedov, Software Engineer, ResearchGate

ResearchGate is a social network for scientists.

It started when two researchers discovered first-hand that collaborating with a friend or colleague on the other side of the world was no easy task.


Connect the world of science.
Make research open to all.

We have, and are continuing to change how scientific knowledge is *shared* and *discovered.*

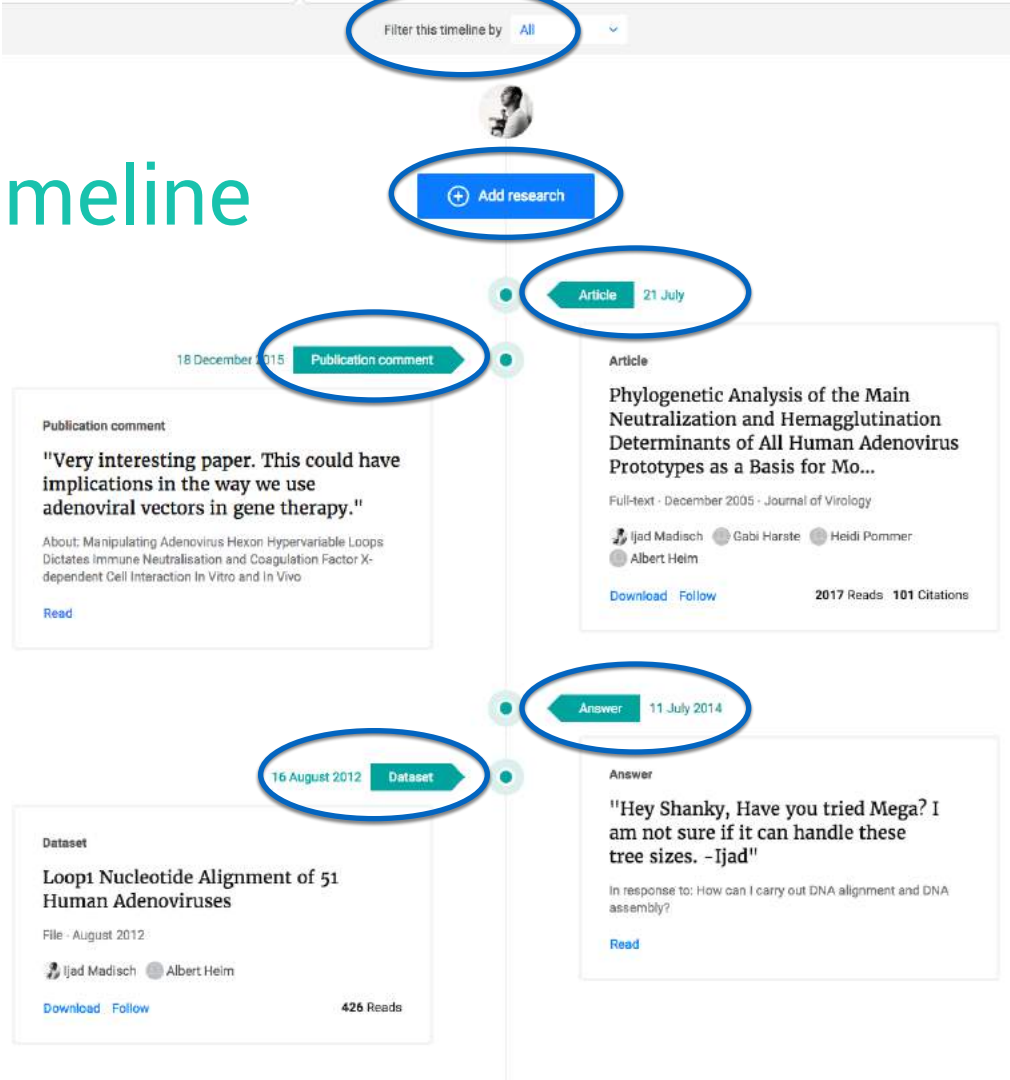
10,000,000

Members

102,000,000

Publications

Feature: Research Timeline



Feature: Research Timeline

Article

Phylogenetic Analysis of the Main Neutralization and Hemagglutination Determinants of All Human Adenovirus Prototypes as a Basis for Mo...

Full-text · December 2005 · Journal of Virology

Ijad Madisch Gabi Harste Heidi Pommer Albert Heim

Download Follow 2017 Reads 101 Citations

Filter this timeline by All

Add research

Article 21 July

Article

Phylogenetic Analysis of the Main Neutralization and Hemagglutination Determinants of All Human Adenovirus Prototypes as a Basis for Mo...

Full-text · December 2005 · Journal of Virology

Ijad Madisch Gabi Harste Heidi Pommer Albert Heim

Download Follow 2017 Reads 101 Citations

18 December 2015 Publication comment

Publication comment

"Very interesting paper. This could have implications in the way we use adenoviral vectors in gene therapy."

About: Manipulating Adenovirus Hexon Hypervariable Loops Dictates Immune Neutralisation and Coagulation Factor X-dependent Cell Interaction In Vitro and In Vivo

Read

Answer 11 July 2014

Answer

"Hey Shanky, Have you tried Mega? I am not sure if it can handle these tree sizes. -Ijad"

In response to: How can I carry out DNA alignment and DNA assembly?

Read

16 August 2012 Dataset

Dataset

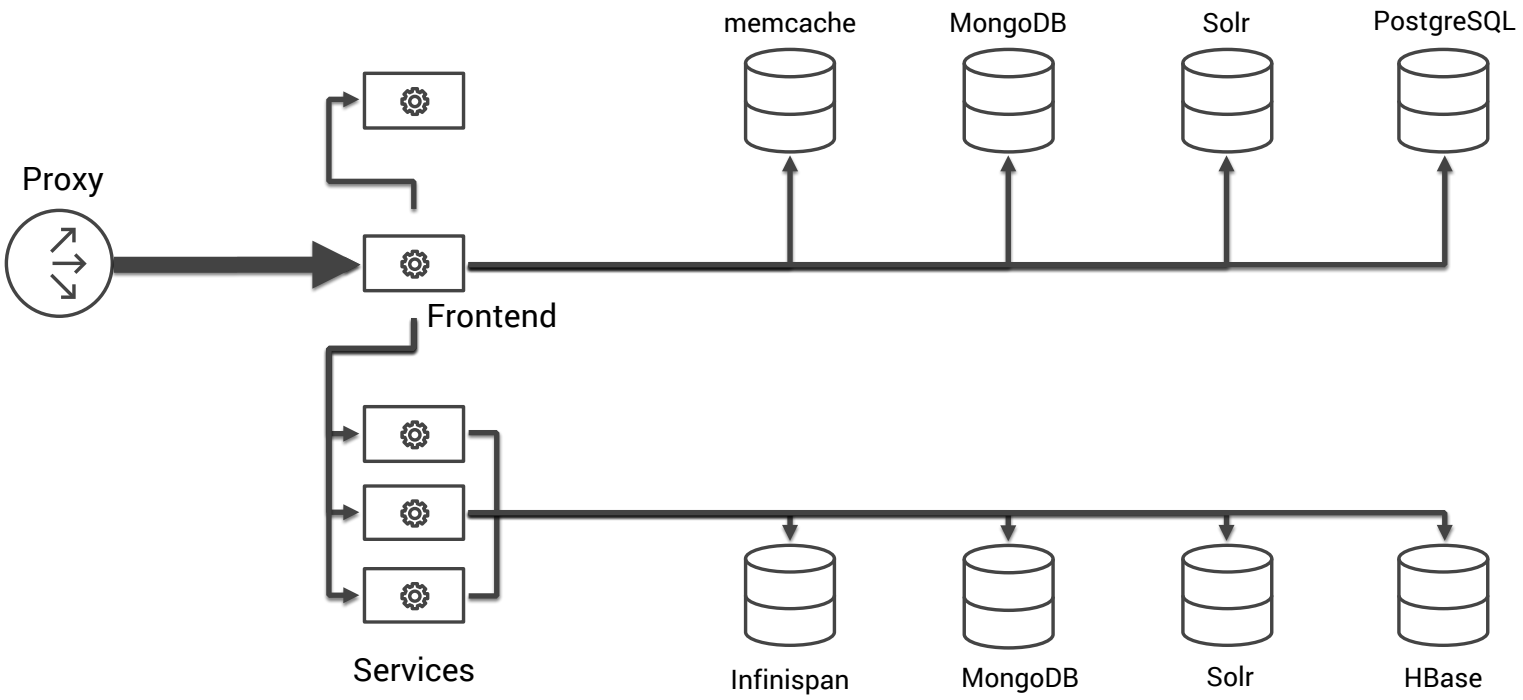
Loop1 Nucleotide Alignment of 51 Human Adenoviruses

File · August 2012

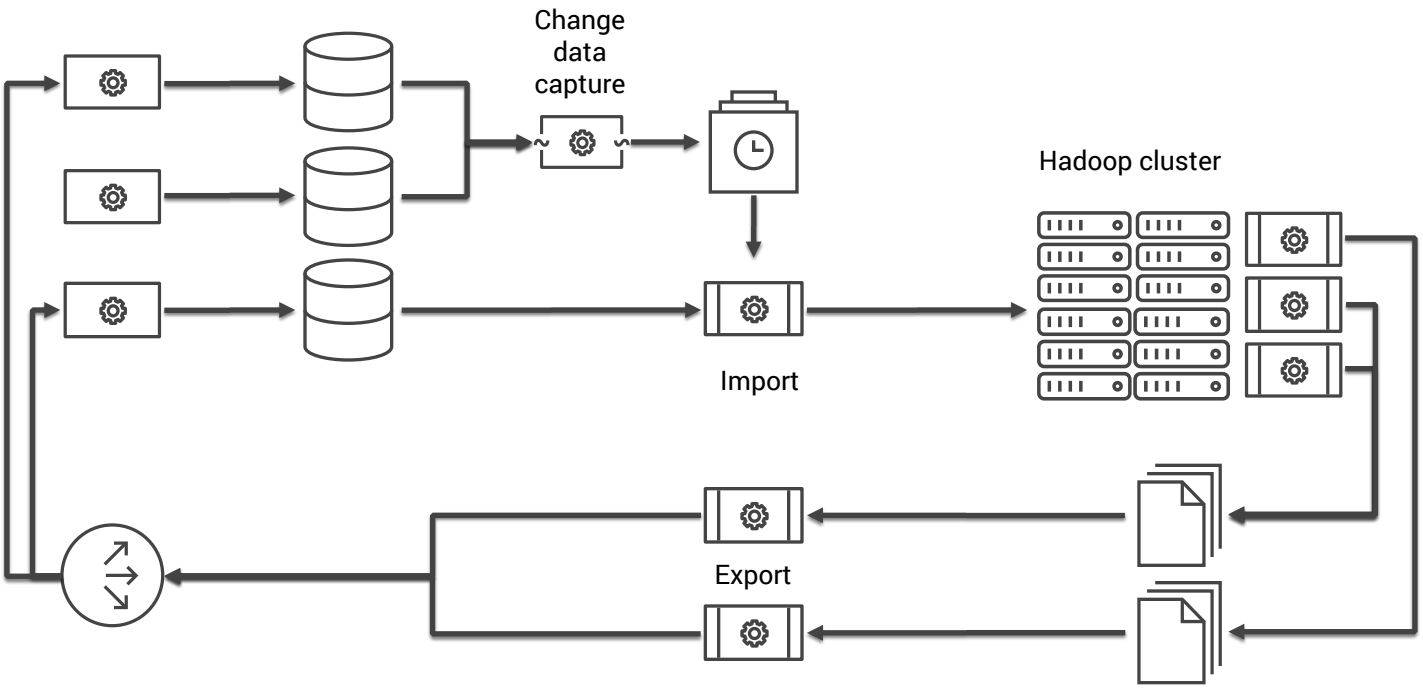
Ijad Madisch Albert Heim

Download Follow 426 Reads

Diverse data sources



Big data pipeline



Data Model

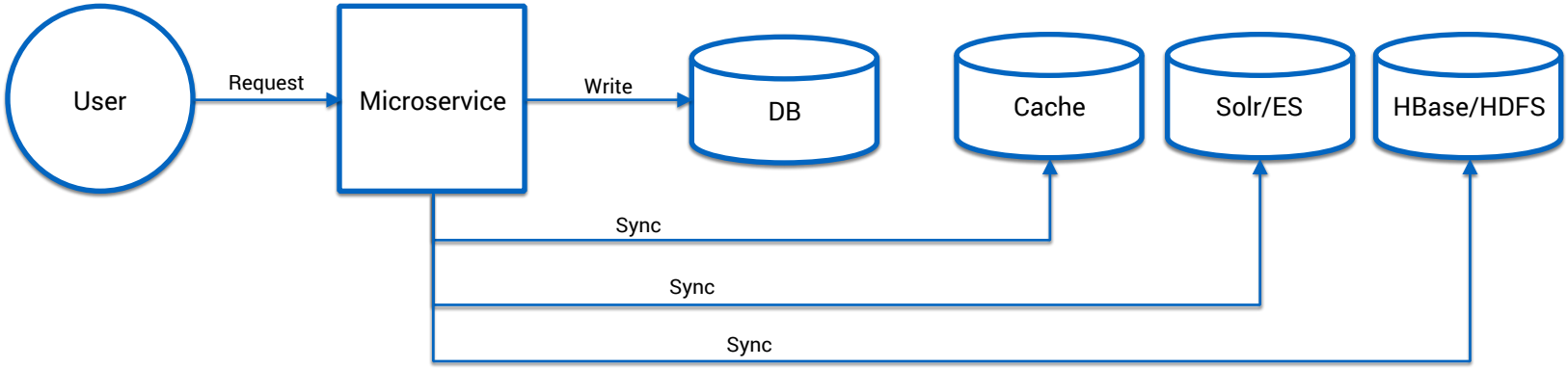
Hypothetical SQL

```sql
CREATE TABLE accounts (
  id                 SERIAL PRIMARY KEY,
  claimed_author_ids INTEGER[]
);

CREATE TABLE publications (
  id         SERIAL PRIMARY KEY,
  author_ids INTEGER[]
);

CREATE MATERIALIZED VIEW account_publications
REFRESH FAST ON COMMIT
AS
  SELECT
    accounts.id     AS account_id,
    publications.id AS publication_id
  FROM accounts
    JOIN publications
      ON ANY (accounts.claimed_author_ids) = ANY (publications.author_ids);
```

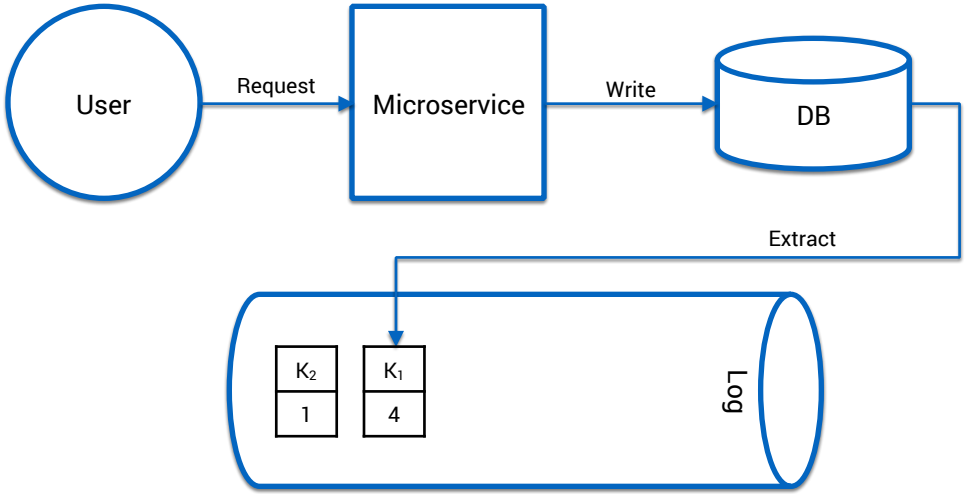
Challenges

- Data sources are **distributed** across different DBs
- Dataset **doesn't fit in memory** on a single machine
- Join process must be **fault tolerant**
- **Deploy** changes fast
- Up-to-date join result in **near real-time**
- Join result must be **accurate**

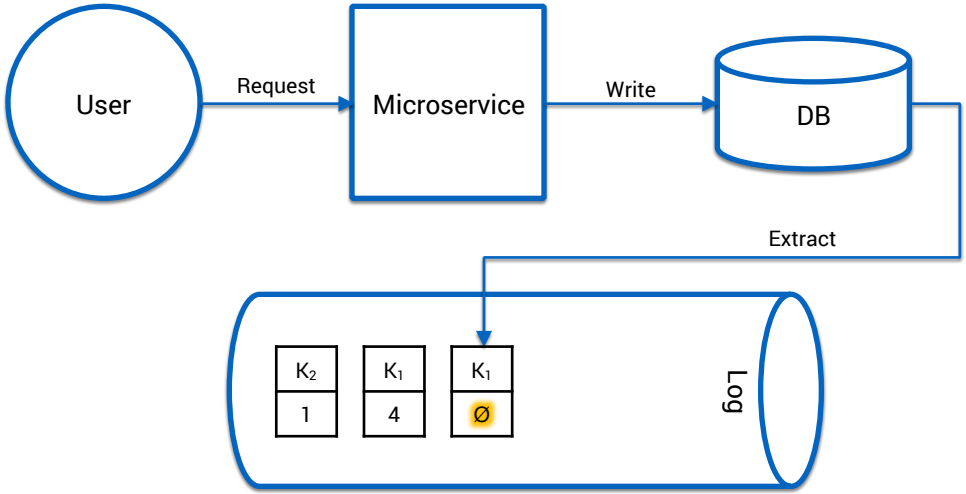
Change data capture (CDC)



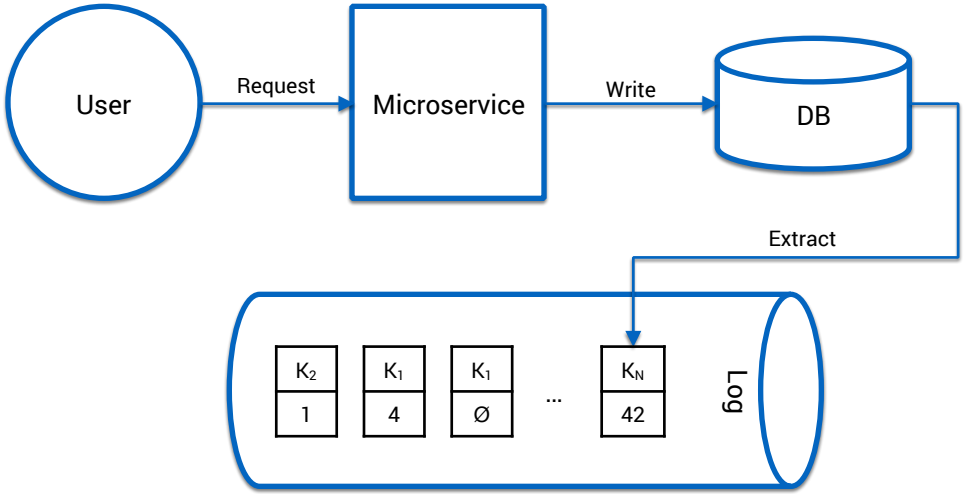
Change data capture (CDC)



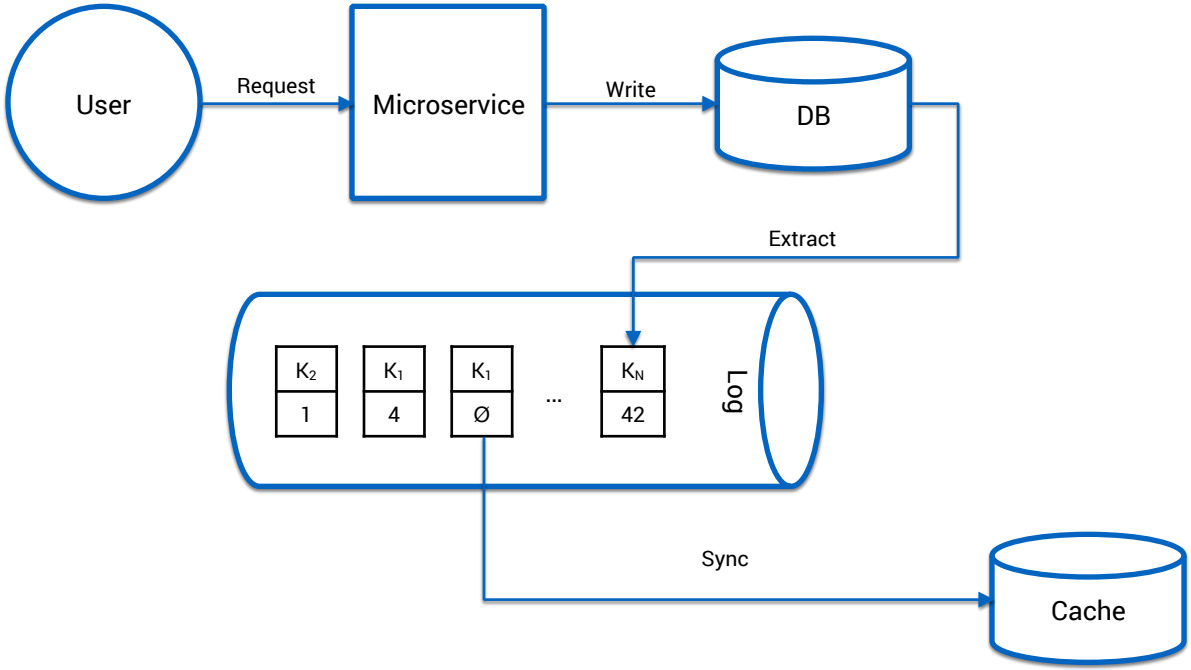
Change data capture (CDC)



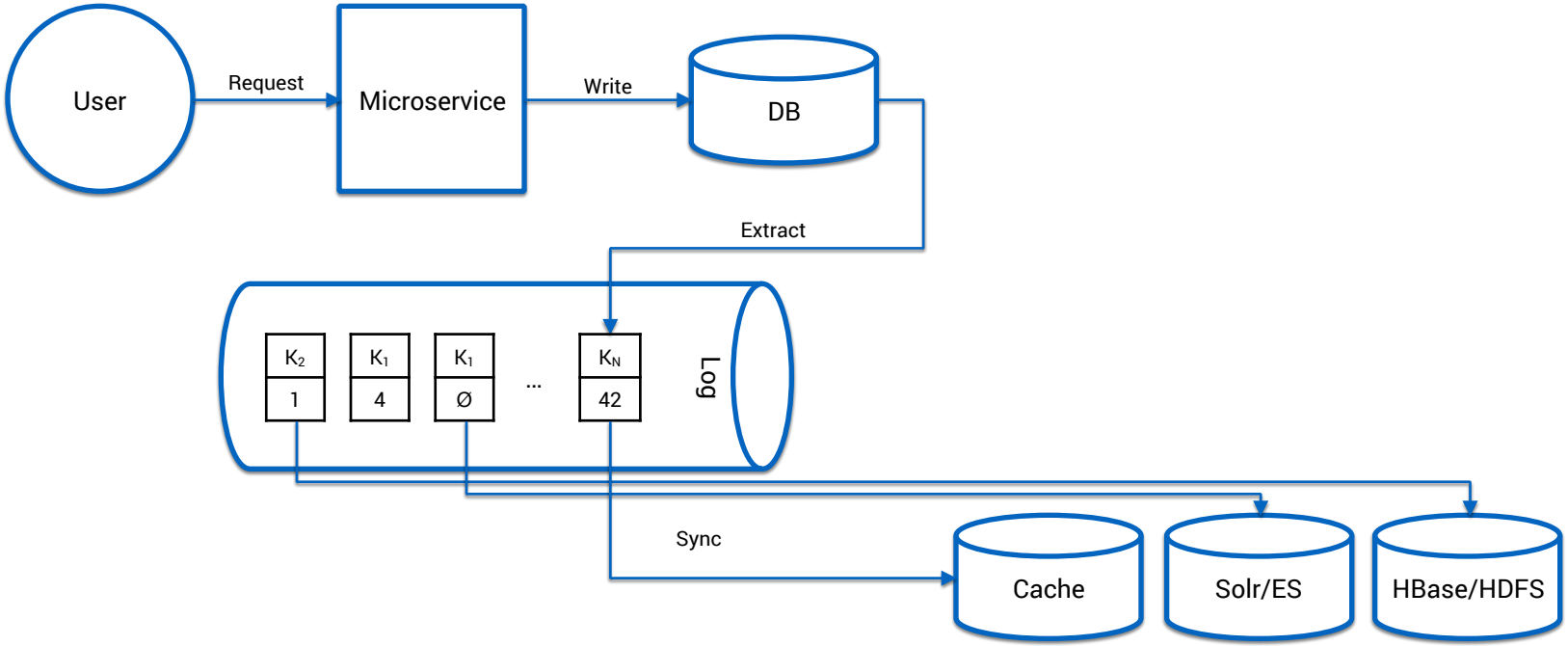
Change data capture (CDC)



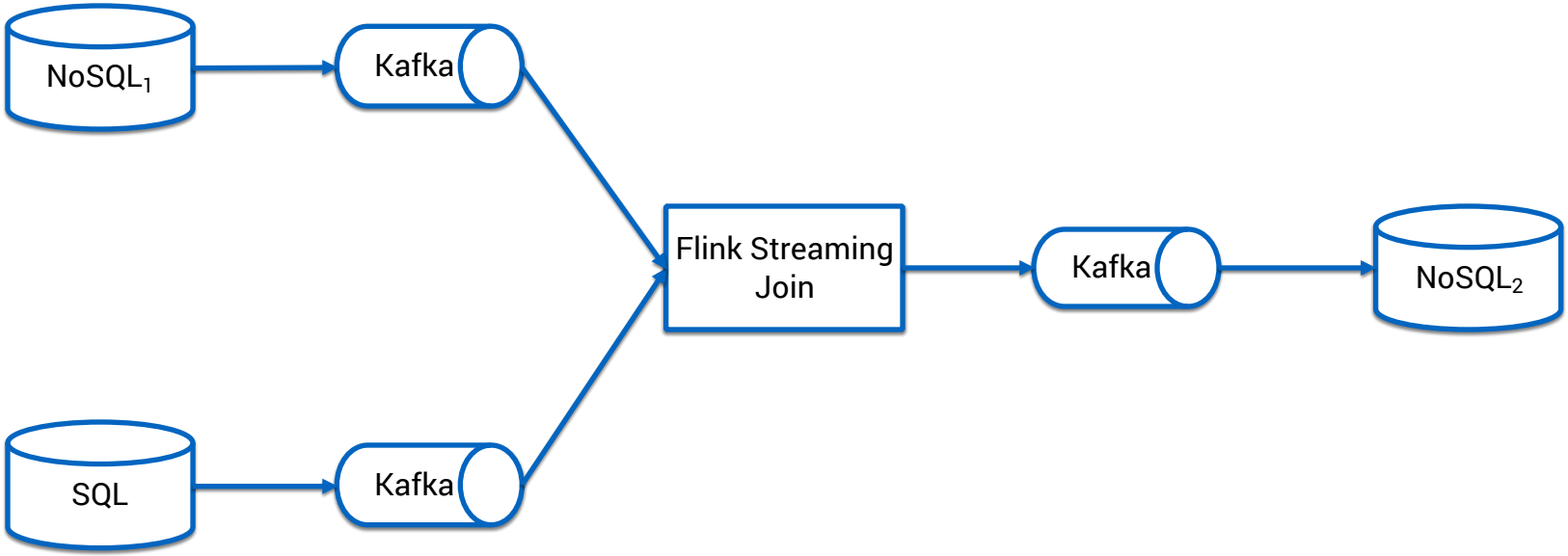
Change data capture (CDC)



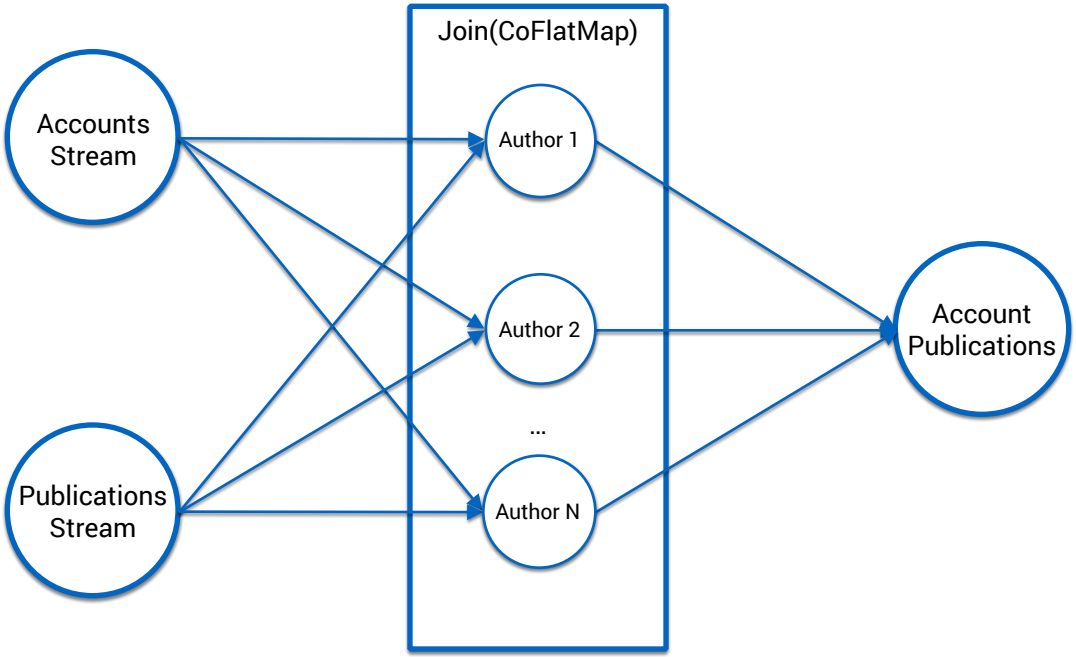
Change data capture (CDC)



Join two CDC streams into one



Flink job topology



Prototype implementation

```
DataStream<Account> accounts = kafkaTopic("accounts");
DataStream<Publication> publications = kafkaTopic("publications");
DataStream<AccountPublication> result = accounts.connect(publications)
  .keyBy("claimedAuthorId", "publicationAuthorId")
  .flatMap(new RichCoFlatMapFunction<Account, Publication, AccountPublication>() {

    transient ValueState<String> authorAccount;
    transient ValueState<String> authorPublication;

    public void open(Configuration parameters) throws Exception {
      authorAccount = getRuntimeContext().getState(new ValueStateDescriptor<>("authorAccount", String.class, null));
      authorPublication = getRuntimeContext().getState(new ValueStateDescriptor<>("authorPublication", String.class, null));
    }

    public void flatMap1(Account account, Collector<AccountPublication> out) throws Exception {
      authorAccount.update(account.id);
      if (authorPublication.value() != null) {
        out.collect(new AccountPublication(authorAccount.value(), authorPublication.value()));
      }
    }

    public void flatMap2(Publication publication, Collector<AccountPublication> out) throws Exception {
      authorPublication.update(publication.id);
      if (authorAccount.value() != null) {
        out.collect(new AccountPublication(authorAccount.value(), authorPublication.value()));
      }
    }
  });
```

Prototype implementation

```
DataStream<Account> accounts = kafkaTopic("accounts");
DataStream<Publication> publications = kafkaTopic("publications");
DataStream<AccountPublication> result = accounts.connect(publications)
  .keyBy("claimedAuthorId", "publicationAuthorId")
  .flatMap(new RichCoFlatMapFunction<Account, Publication, AccountPublication>() {

    transient ValueState<String> authorAccount;
    transient ValueState<String> authorPublication;

    public void open(Configuration parameters) throws Exception {
      authorAccount = getRuntimeContext().getState(new ValueStateDescriptor<>("authorAccount", String.class, null));
      authorPublication = getRuntimeContext().getState(new ValueStateDescriptor<>("authorPublication", String.class, null));
    }

    public void flatMap1(Account account, Collector<AccountPublication> out) throws Exception {
      authorAccount.update(account.id);
      if (authorPublication.value() != null) {
        out.collect(new AccountPublication(authorAccount.value(), authorPublication.value()));
      }
    }

    public void flatMap2(Publication publication, Collector<AccountPublication> out) throws Exception {
      authorPublication.update(publication.id);
      if (authorAccount.value() != null) {
        out.collect(new AccountPublication(authorAccount.value(), authorPublication.value()));
      }
    }
  });
```

Prototype implementation

```
DataStream<Account> accounts = kafkaTopic("accounts");
DataStream<Publication> publications = kafkaTopic("publications");
DataStream<AccountPublication> result = accounts.connect(publications)
  .keyBy("claimedAuthorId", "publicationAuthorId")
  .flatMap(new RichCoFlatMapFunction<Account, Publication, AccountPublication>() {

    transient ValueState<String> authorAccount;
    transient ValueState<String> authorPublication;

    public void open(Configuration parameters) throws Exception {
      authorAccount = getRuntimeContext().getState(new ValueStateDescriptor<>("authorAccount", String.class, null));
      authorPublication = getRuntimeContext().getState(new ValueStateDescriptor<>("authorPublication", String.class, null));
    }

    public void flatMap1(Account account, Collector<AccountPublication> out) throws Exception {
      authorAccount.update(account.id);
      if (authorPublication.value() != null) {
        out.collect(new AccountPublication(authorAccount.value(), authorPublication.value()));
      }
    }

    public void flatMap2(Publication publication, Collector<AccountPublication> out) throws Exception {
      authorPublication.update(publication.id);
      if (authorAccount.value() != null) {
        out.collect(new AccountPublication(authorAccount.value(), authorPublication.value()));
      }
    }
  });
```

Prototype implementation

```
DataStream<Account> accounts = kafkaTopic("accounts");
DataStream<Publication> publications = kafkaTopic("publications");
DataStream<AccountPublication> result = accounts.connect(publications)
  .keyBy("claimedAuthorId", "publicationAuthorId")
  .flatMap(new RichCoFlatMapFunction<Account, Publication, AccountPublication>() {

    transient ValueState<String> authorAccount;
    transient ValueState<String> authorPublication;

    public void open(Configuration parameters) throws Exception {
      authorAccount = getRuntimeContext().getState(new ValueStateDescriptor<>("authorAccount", String.class, null));
      authorPublication = getRuntimeContext().getState(new ValueStateDescriptor<>("authorPublication", String.class, null));
    }

    public void flatMap1(Account account, Collector<AccountPublication> out) throws Exception {
      authorAccount.update(account.id);
      if (authorPublication.value() != null) {
        out.collect(new AccountPublication(authorAccount.value(), authorPublication.value()));
      }
    }

    public void flatMap2(Publication publication, Collector<AccountPublication> out) throws Exception {
      authorPublication.update(publication.id);
      if (authorAccount.value() != null) {
        out.collect(new AccountPublication(authorAccount.value(), authorPublication.value()));
      }
    }
  });
```

Prototype implementation

```
DataStream<Account> accounts = kafkaTopic("accounts");
DataStream<Publication> publications = kafkaTopic("publications");
DataStream<AccountPublication> result = accounts.connect(publications)
  .keyBy("claimedAuthorId", "publicationAuthorId")
  .flatMap(new RichCoFlatMapFunction<Account, Publication, AccountPublication>() {

    transient ValueState<String> authorAccount;
    transient ValueState<String> authorPublication;

    public void open(Configuration parameters) throws Exception {
      authorAccount = getRuntimeContext().getState(new ValueStateDescriptor<>("authorAccount", String.class, null));
      authorPublication = getRuntimeContext().getState(new ValueStateDescriptor<>("authorPublication", String.class, null));
    }

    public void flatMap1(Account account, Collector<AccountPublication> out) throws Exception {
      authorAccount.update(account.id);
      if (authorPublication.value() != null) {
        out.collect(new AccountPublication(authorAccount.value(), authorPublication.value()));
      }
    }

    public void flatMap2(Publication publication, Collector<AccountPublication> out) throws Exception {
      authorPublication.update(publication.id);
      if (authorAccount.value() != null) {
        out.collect(new AccountPublication(authorAccount.value(), authorPublication.value()));
      }
    }
  });
```

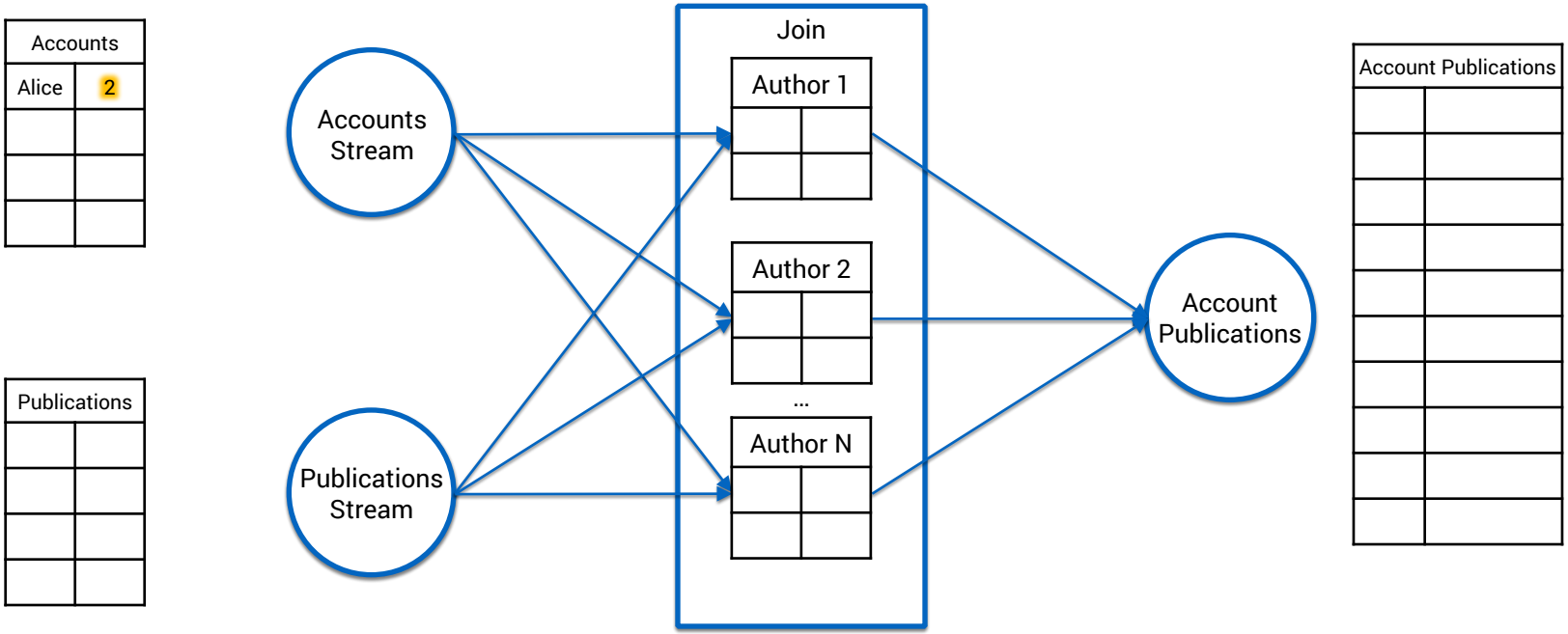
Prototype implementation

```
DataStream<Account> accounts = kafkaTopic("accounts");
DataStream<Publication> publications = kafkaTopic("publications");
DataStream<AccountPublication> result = accounts.connect(publications)
  .keyBy("claimedAuthorId", "publicationAuthorId")
  .flatMap(new RichCoFlatMapFunction<Account, Publication, AccountPublication>() {

    transient ValueState<String> authorAccount;
    transient ValueState<String> authorPublication;

    public void open(Configuration parameters) throws Exception {
      authorAccount = getRuntimeContext().getState(new ValueStateDescriptor<>("authorAccount", String.class, null));
      authorPublication = getRuntimeContext().getState(new ValueStateDescriptor<>("authorPublication", String.class, null));
    }

    public void flatMap1(Account account, Collector<AccountPublication> out) throws Exception {
      authorAccount.update(account.id);
      if (authorPublication.value() != null) {
        out.collect(new AccountPublication(authorAccount.value(), authorPublication.value()));
      }
    }

    public void flatMap2(Publication publication, Collector<AccountPublication> out) throws Exception {
      authorPublication.update(publication.id);
      if (authorAccount.value() != null) {
        out.collect(new AccountPublication(authorAccount.value(), authorPublication.value()));
      }
    }
  });
```

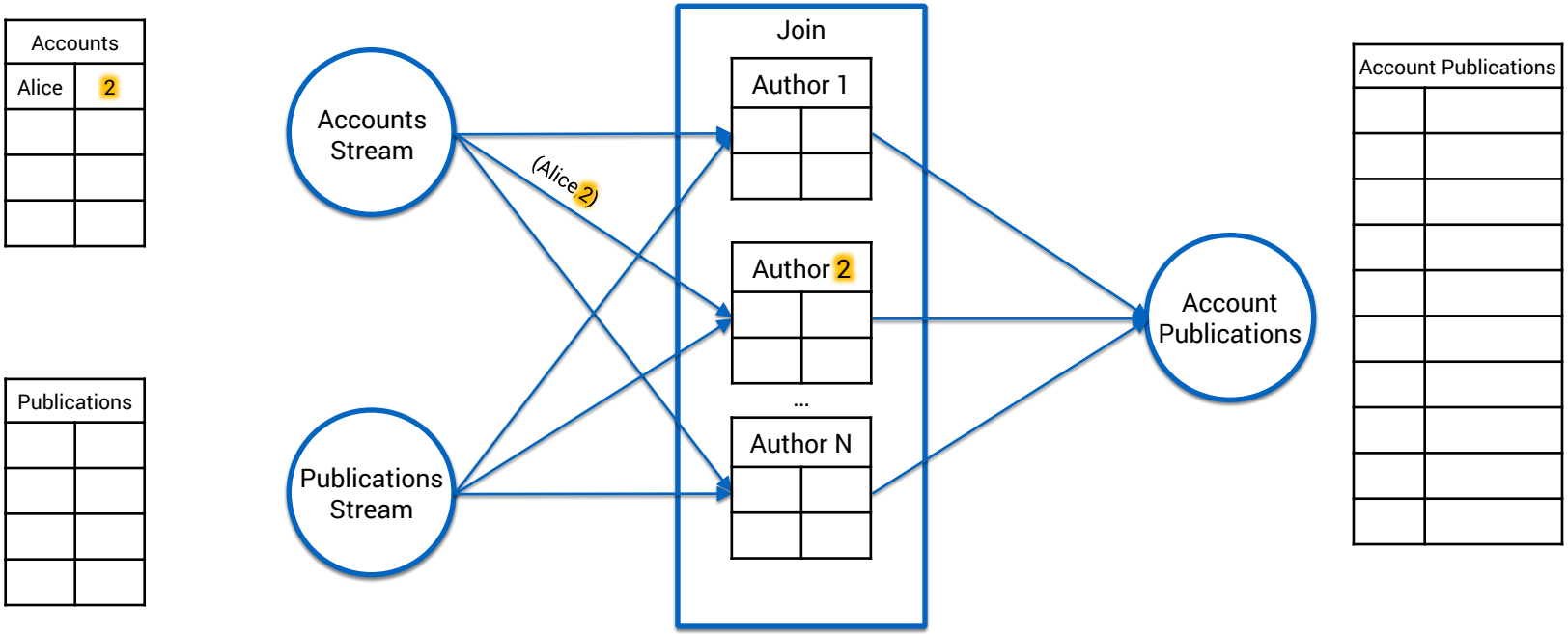
Prototype implementation

```
DataStream<Account> accounts = kafkaTopic("accounts");
DataStream<Publication> publications = kafkaTopic("publications");
DataStream<AccountPublication> result = accounts.connect(publications)
  .keyBy("claimedAuthorId", "publicationAuthorId")
  .flatMap(new RichCoFlatMapFunction<Account, Publication, AccountPublication>() {

    transient ValueState<String> authorAccount;
    transient ValueState<String> authorPublication;

    public void open(Configuration parameters) throws Exception {
      authorAccount = getRuntimeContext().getState(new ValueStateDescriptor<>("authorAccount", String.class, null));
      authorPublication = getRuntimeContext().getState(new ValueStateDescriptor<>("authorPublication", String.class, null));
    }

    public void flatMap1(Account account, Collector<AccountPublication> out) throws Exception {
      authorAccount.update(account.id);
      if (authorPublication.value() != null) {
        out.collect(new AccountPublication(authorAccount.value(), authorPublication.value()));
      }
    }

    public void flatMap2(Publication publication, Collector<AccountPublication> out) throws Exception {
      authorPublication.update(publication.id);
      if (authorAccount.value() != null) {
        out.collect(new AccountPublication(authorAccount.value(), authorPublication.value()));
      }
    }
  });
```

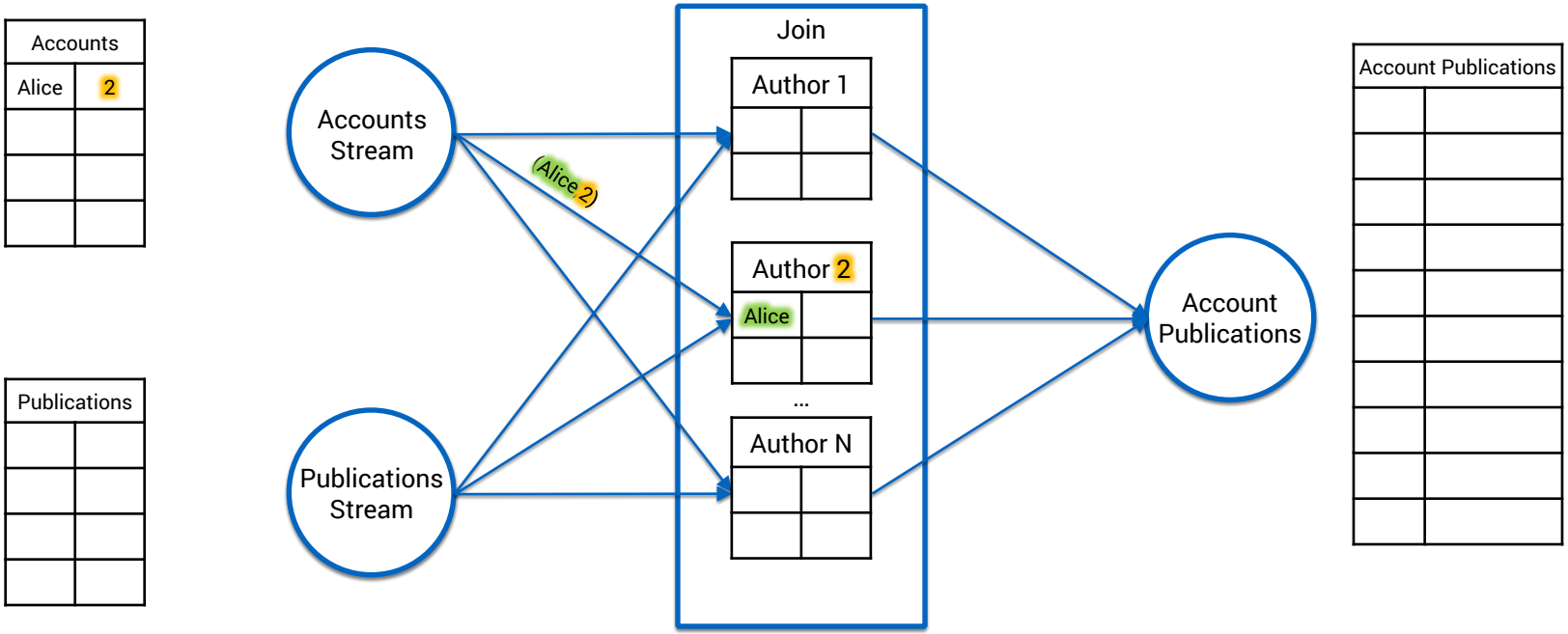
Example dataflow



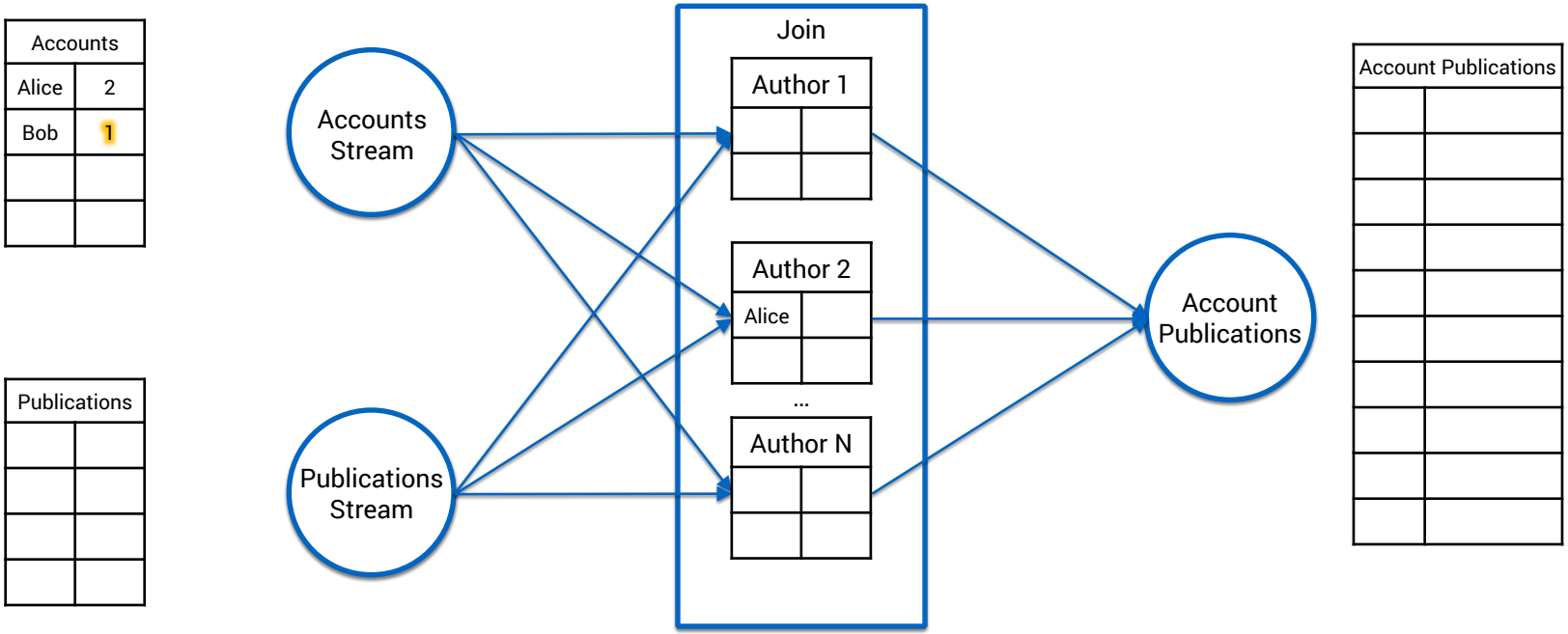
Example dataflow



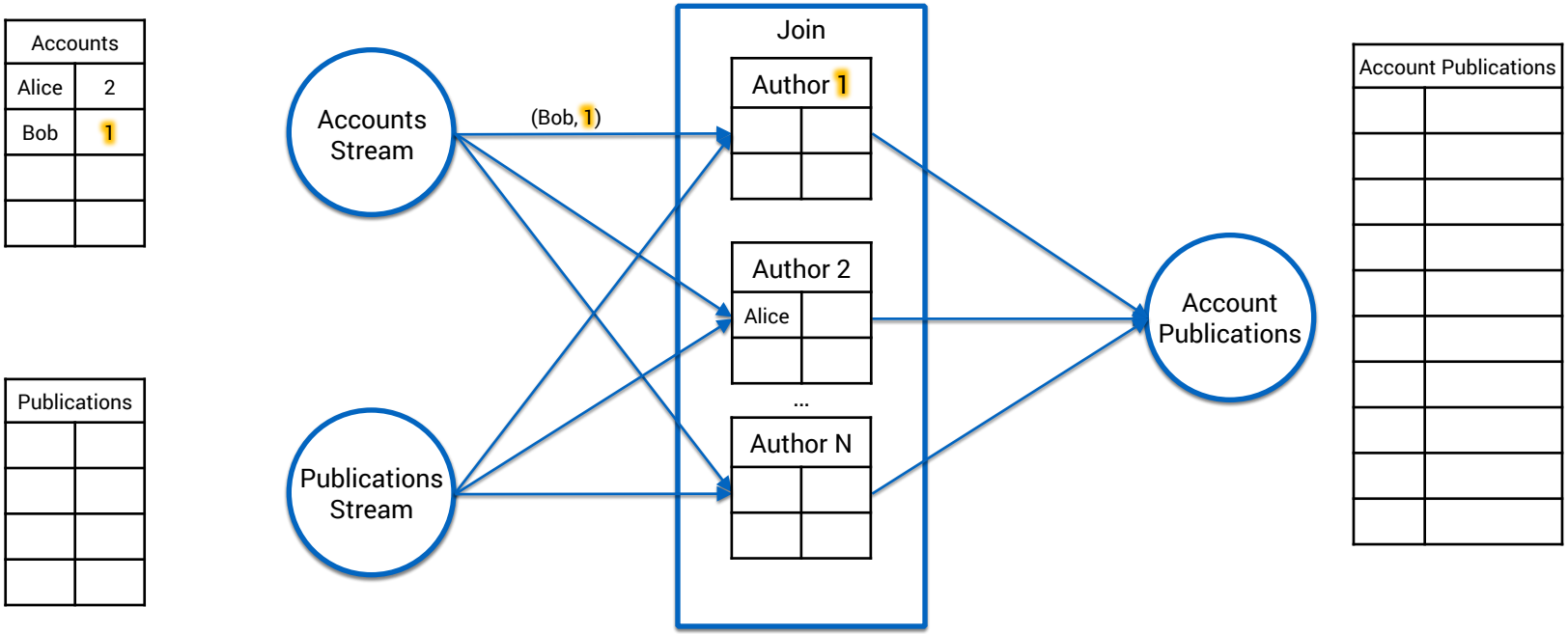
Example dataflow



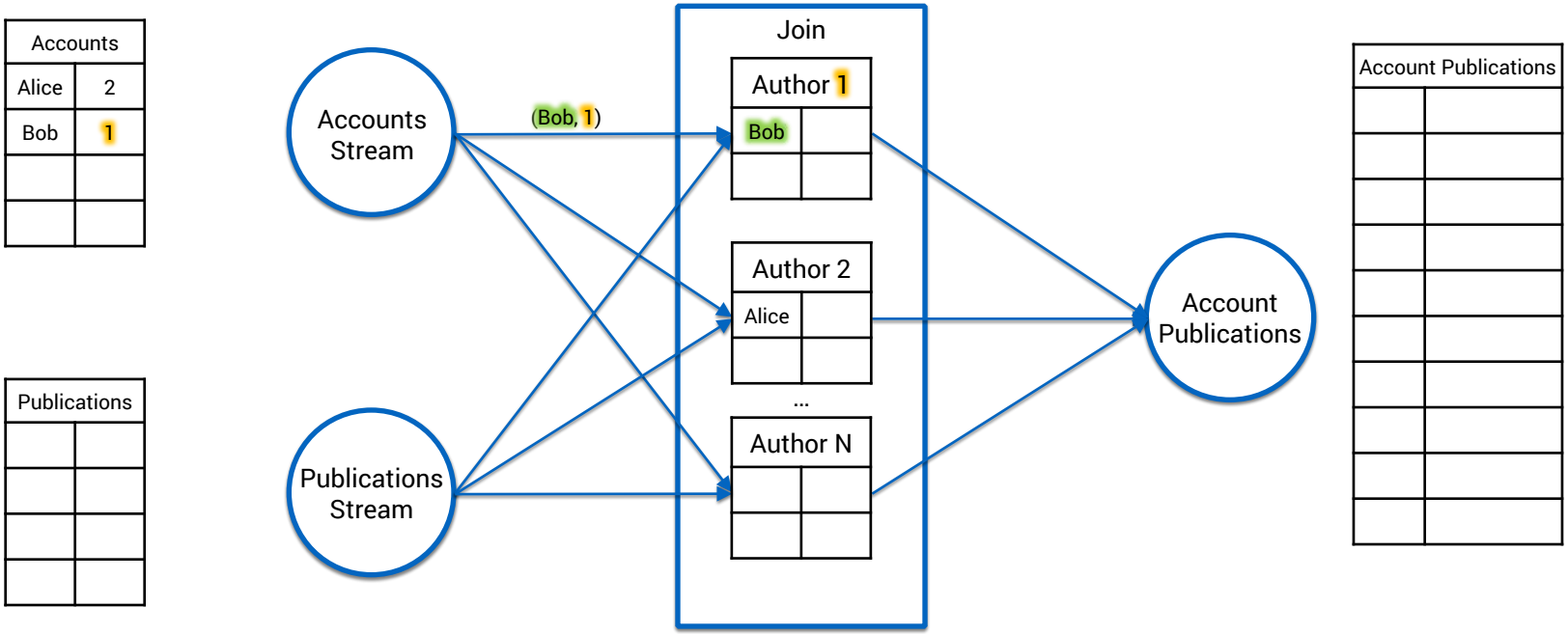
Example dataflow



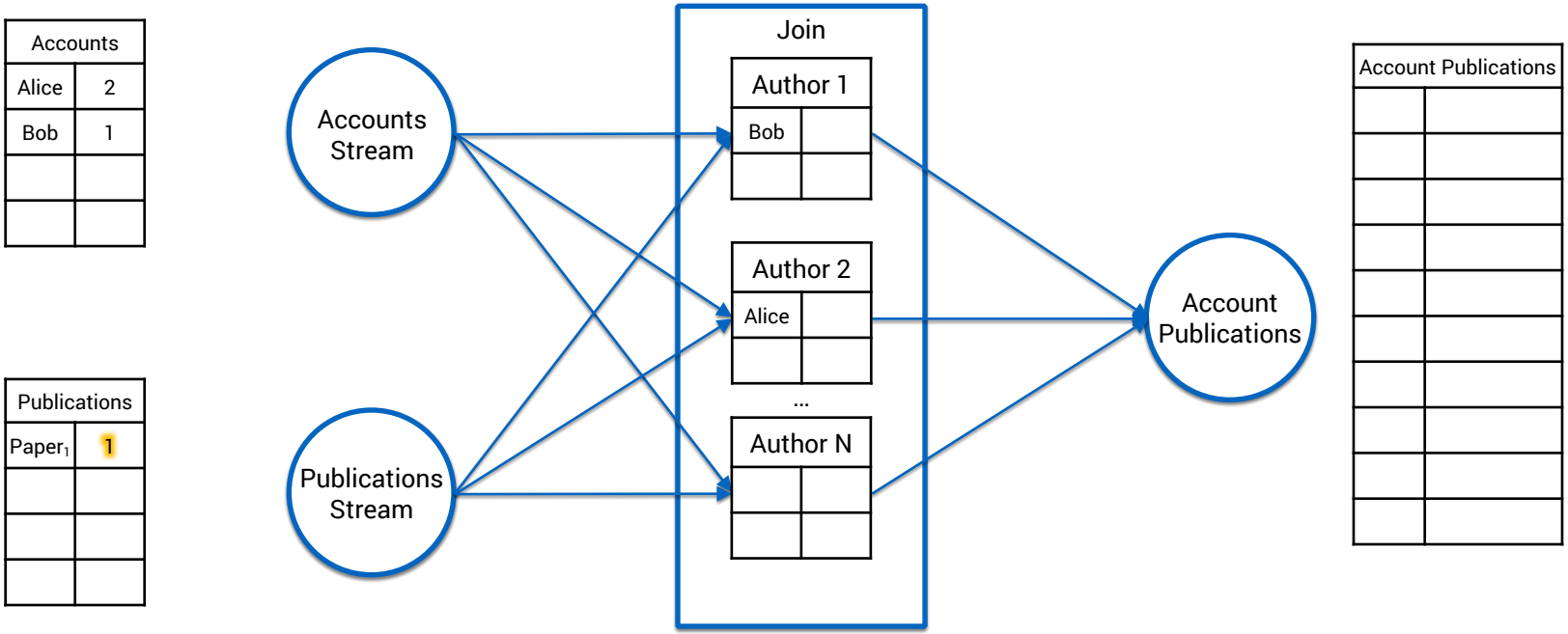
Example dataflow



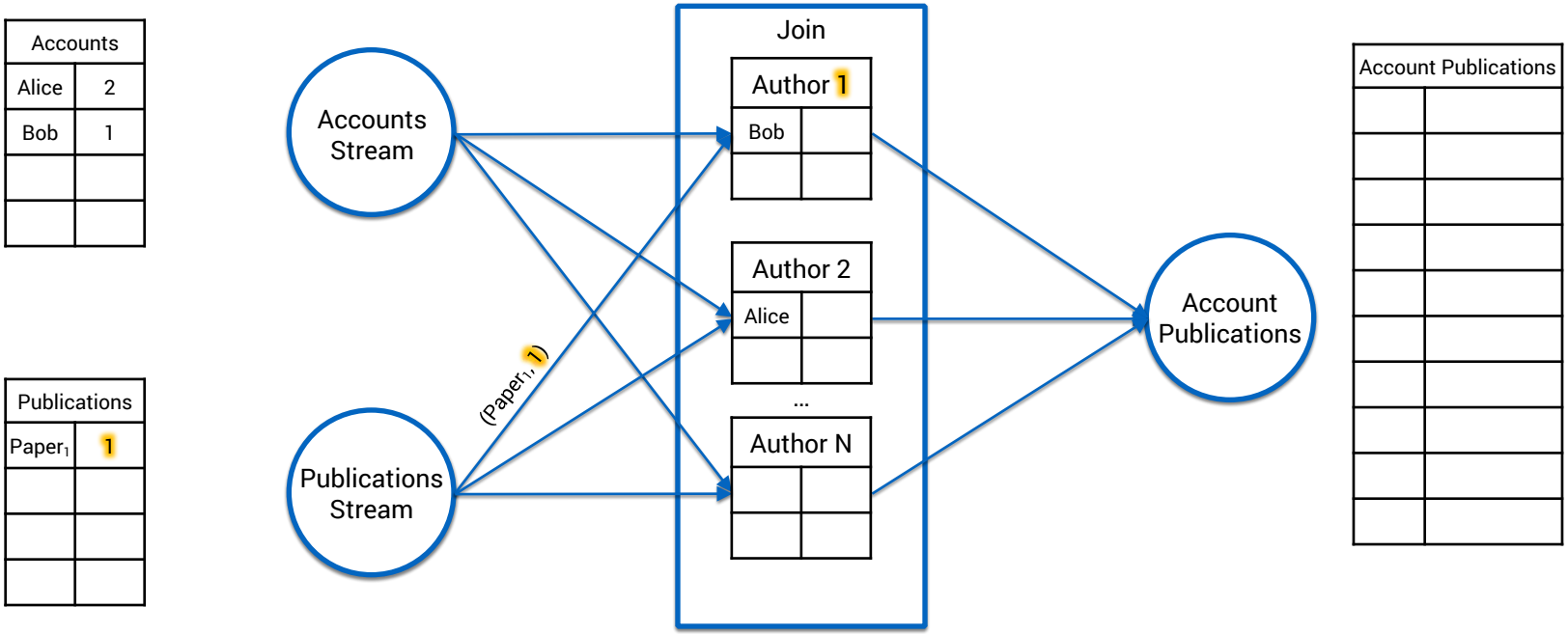
Example dataflow



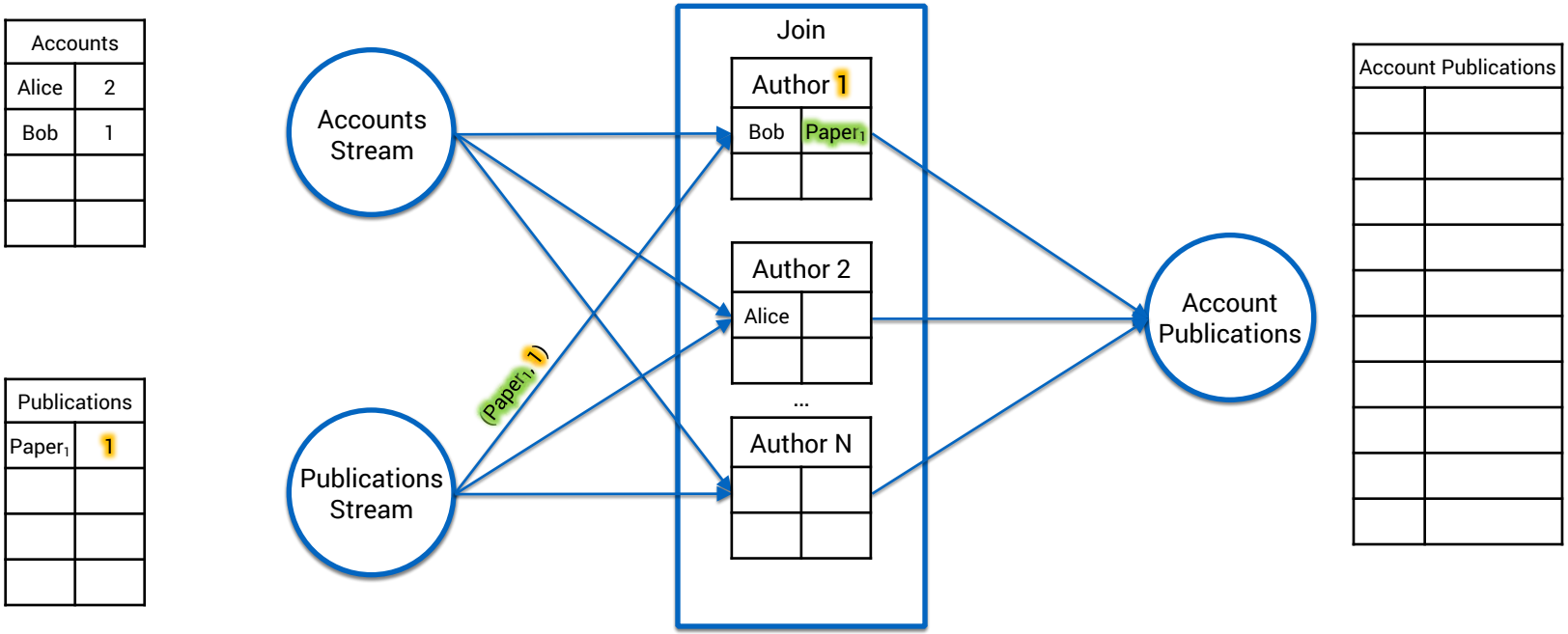
Example dataflow



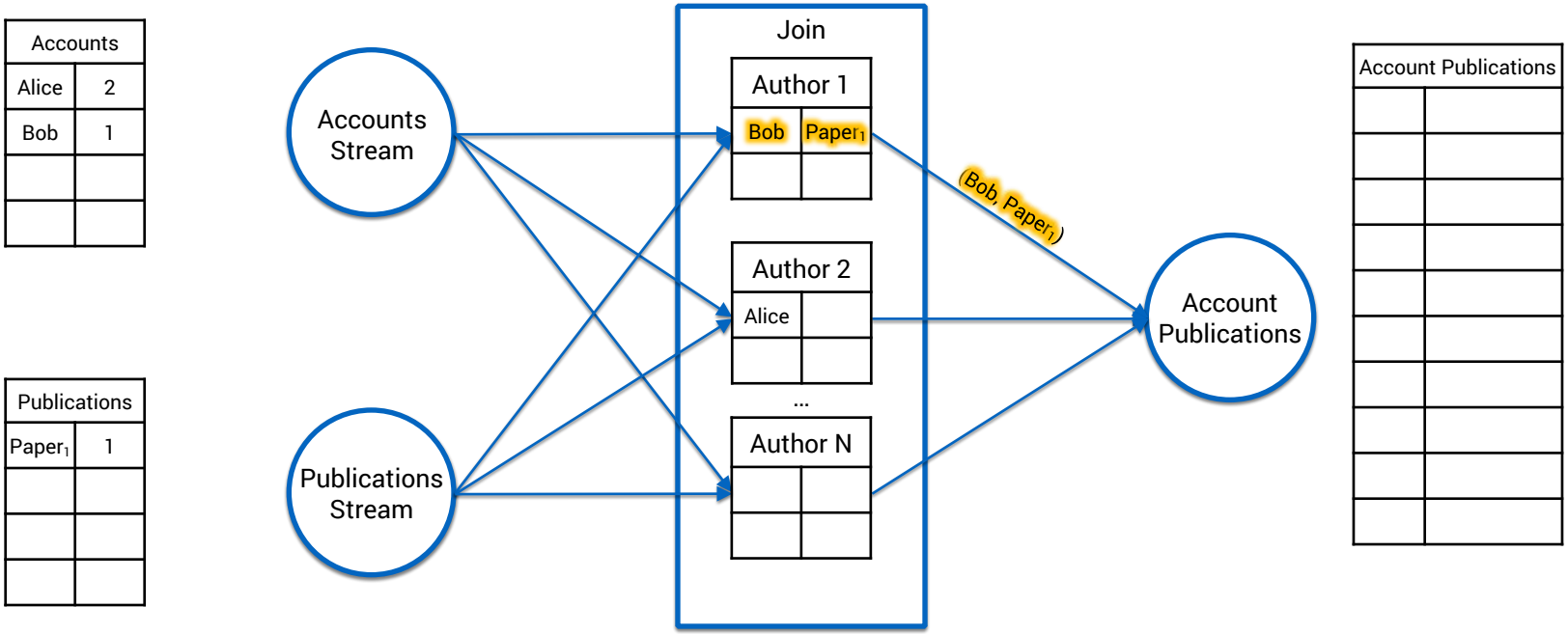
Example dataflow



Example dataflow



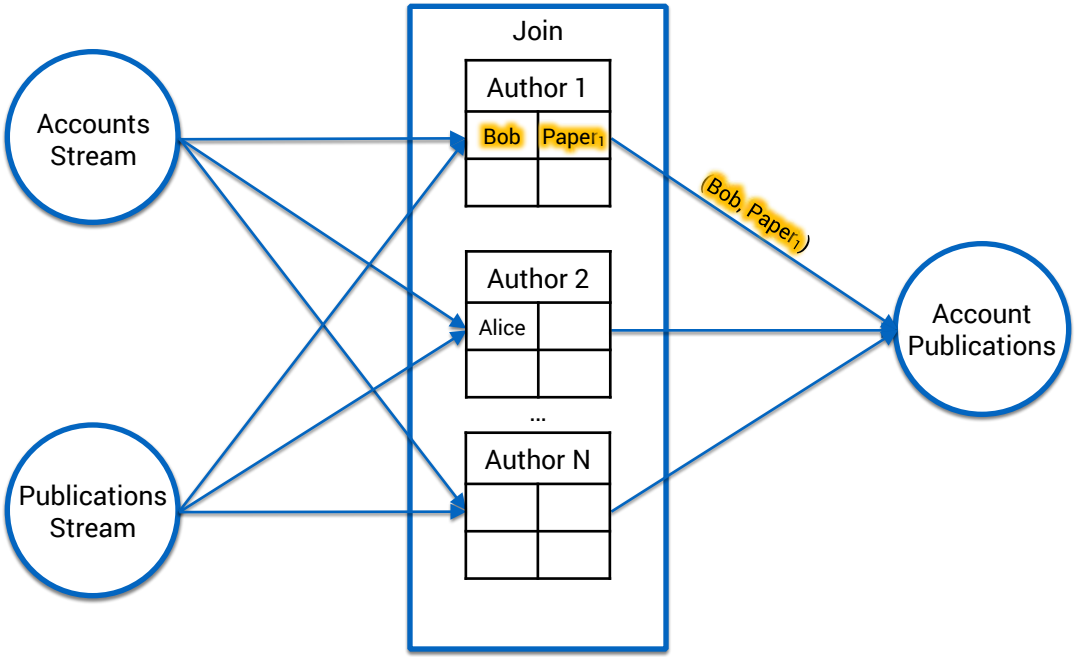
Example dataflow



Example dataflow

Accounts	
Alice	2
Bob	1

Publications	
$Paper_1$	1



Account Publications	
K_1	(Bob, $Paper_1$)

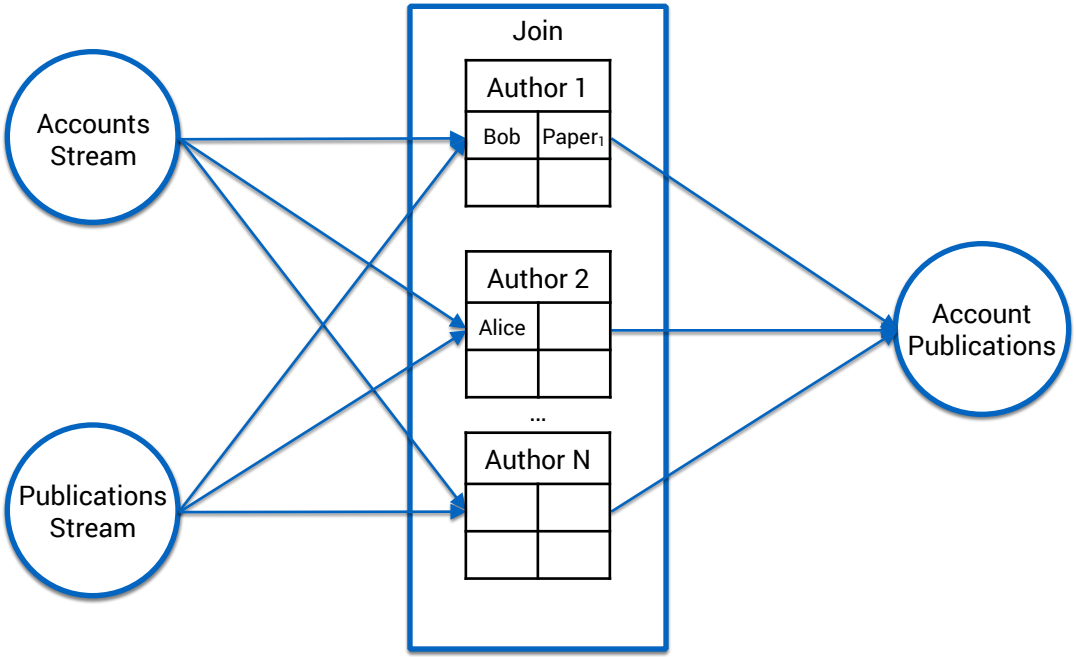
Challenges

- ✔ Data sources are **distributed** across different DBs
- ✔ Dataset **doesn't fit in memory** on a single machine
- ✔ Join process must be **fault tolerant**
- ✔ **Deploy** changes fast
- ✔ Up-to-date join result in **near real-time**
- ? Join result must be **accurate**

$Paper_1$ gets deleted

Accounts	
Alice	2
Bob	1

Publications	
$Paper_1$	1
$Paper_1$	Ø

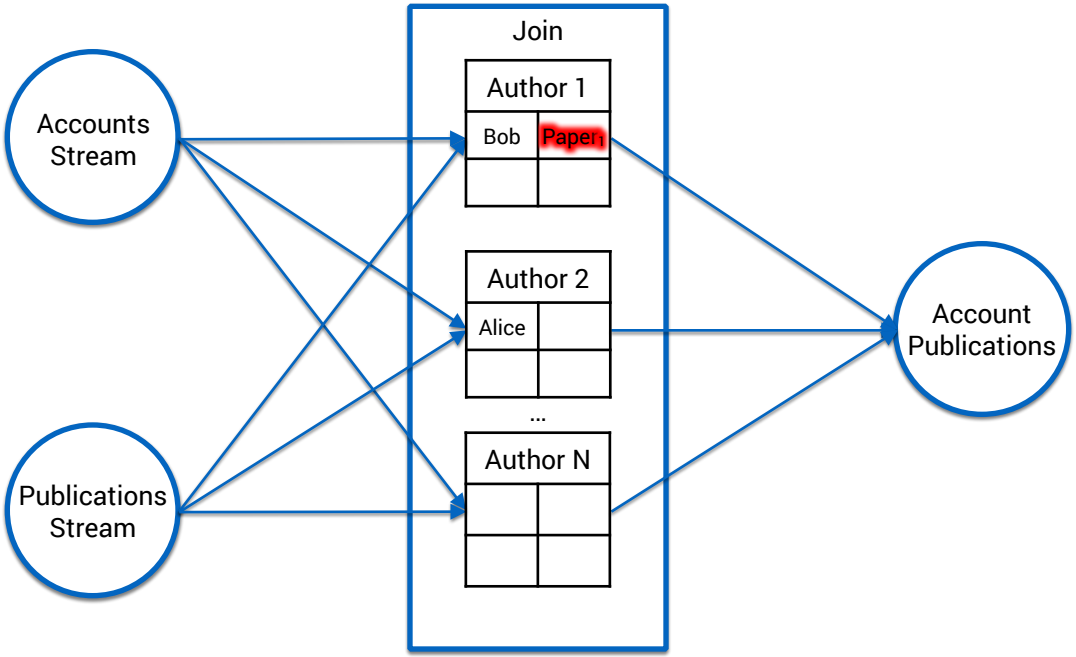


Account Publications	
K_1	(Bob, $Paper_1$)

$Paper_1$ gets deleted

Accounts	
Alice	2
Bob	1

Publications	
$Paper_1$	1
$Paper_1$	Ø

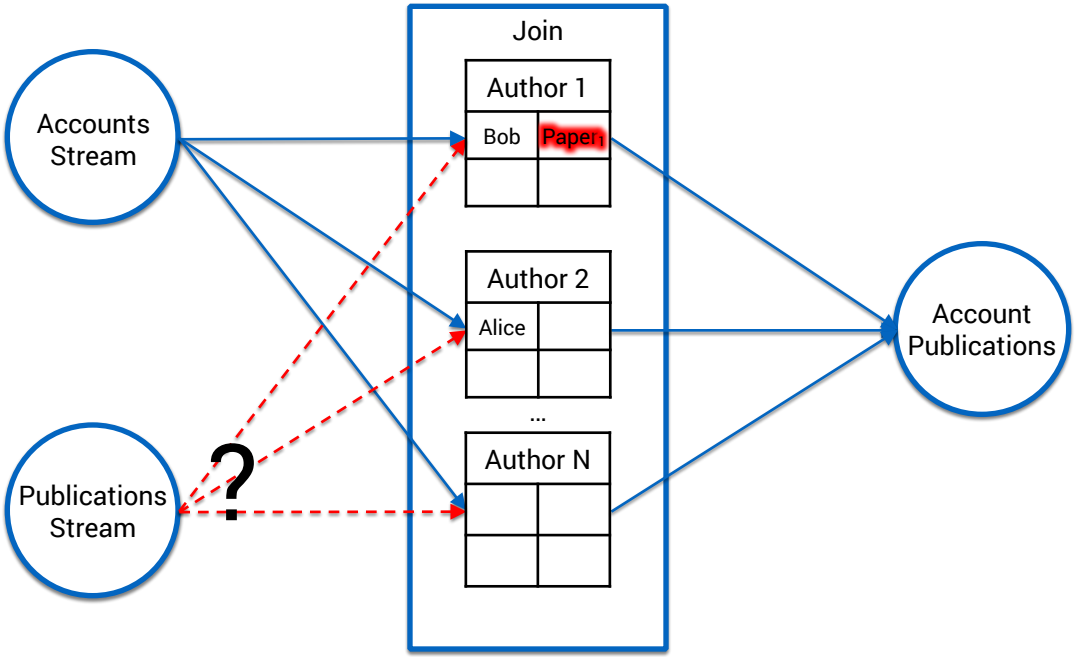


Account Publications	
K_1	(Bob, $Paper_1$)

$Paper_1$ gets deleted

Accounts	
Alice	2
Bob	1

Publications	
$Paper_1$	1
$Paper_1$	Ø

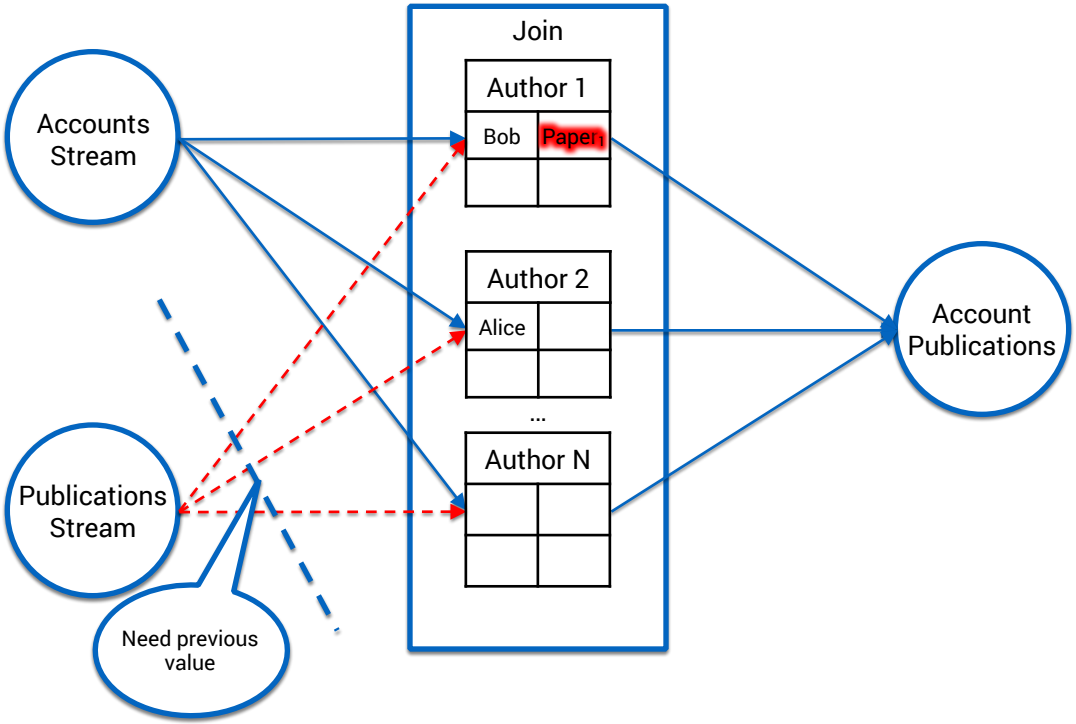


Account Publications	
K_1	(Bob, $Paper_1$)

$Paper_1$ gets deleted

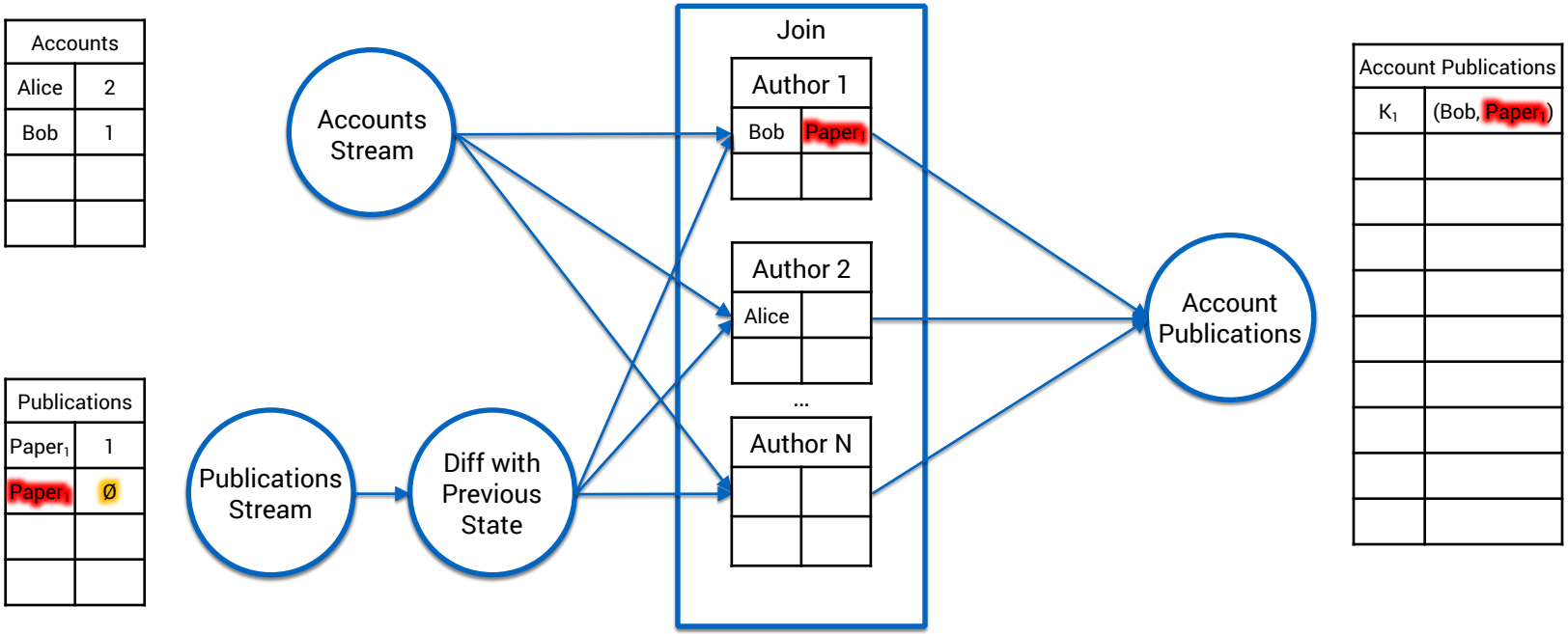
Accounts	
Alice	2
Bob	1

Publications	
$Paper_1$	1
$Paper_1$	Ø

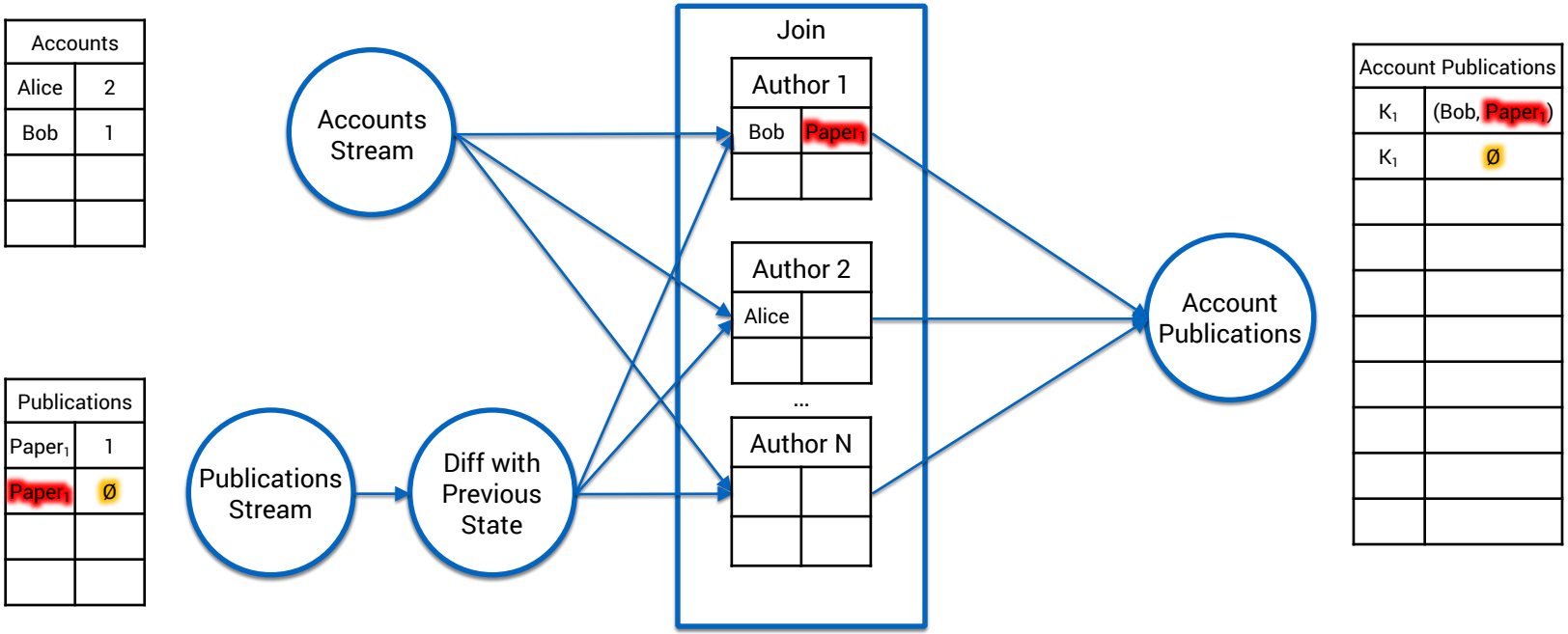


Account Publications	
K_1	(Bob, $Paper_1$)

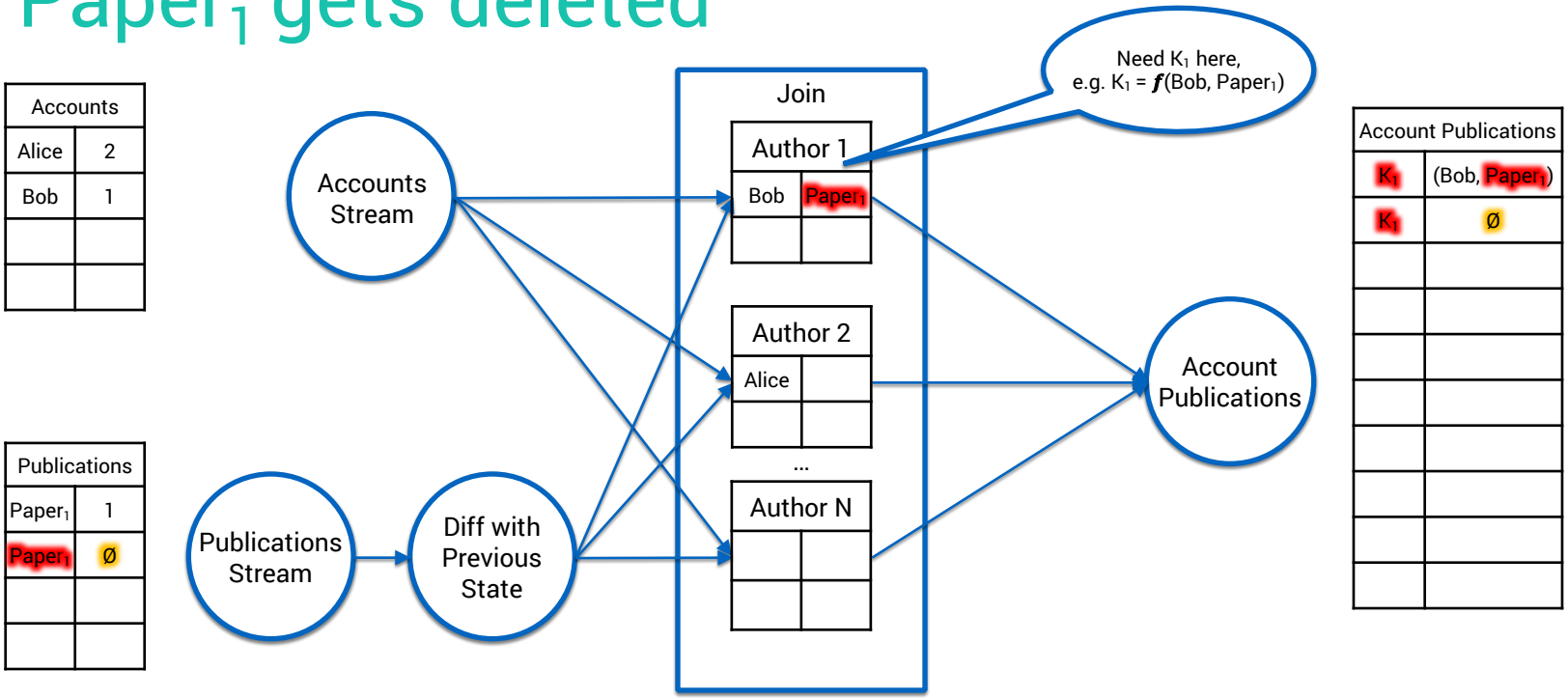
$Paper_1$ gets deleted



$Paper_1$ gets deleted



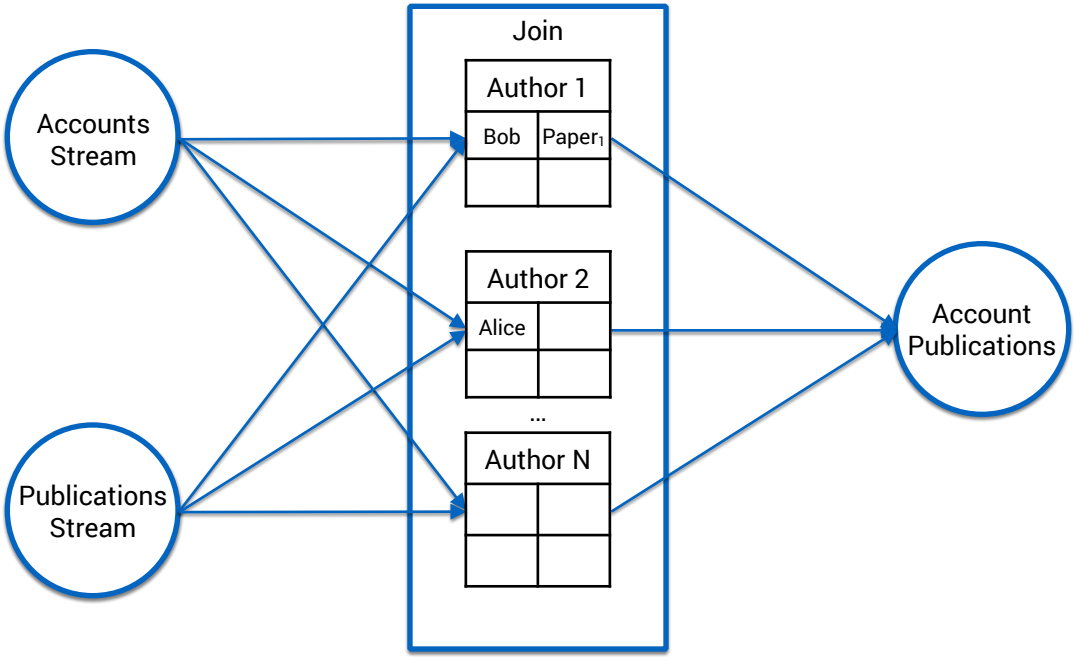
$Paper_1$ gets deleted



$Paper_1$ gets updated

Accounts	
Alice	2
Bob	1

Publications	
$Paper_1$	1
$Paper_1$	2

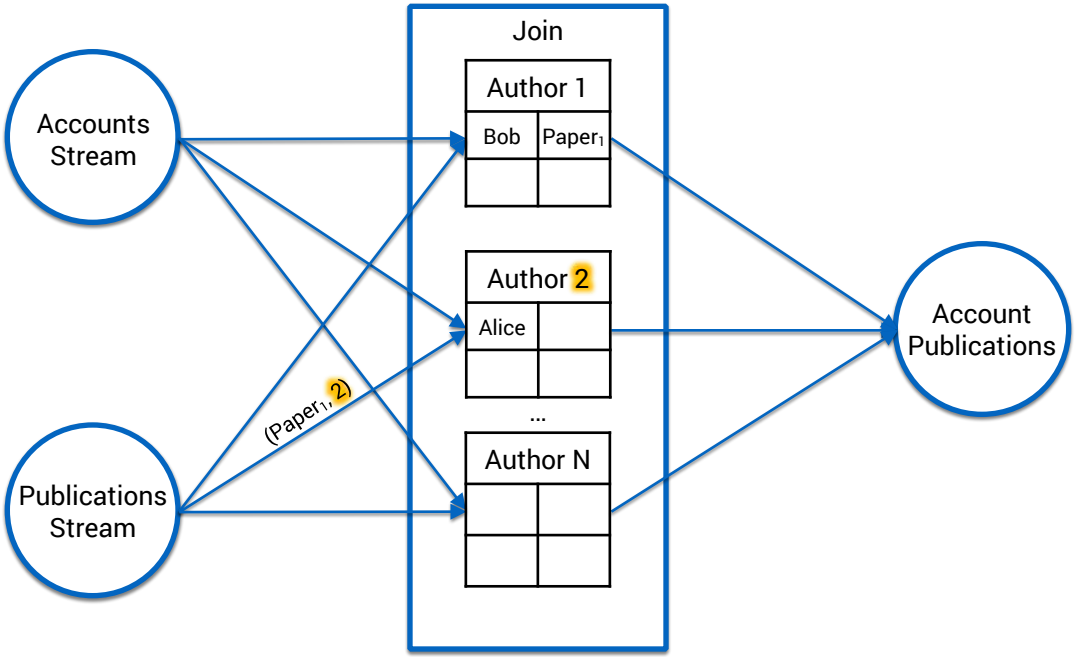


Account Publications	
K_1	(Bob, $Paper_1$)

$Paper_1$ gets updated

Accounts	
Alice	2
Bob	1

Publications	
$Paper_1$	1
$Paper_1$	2

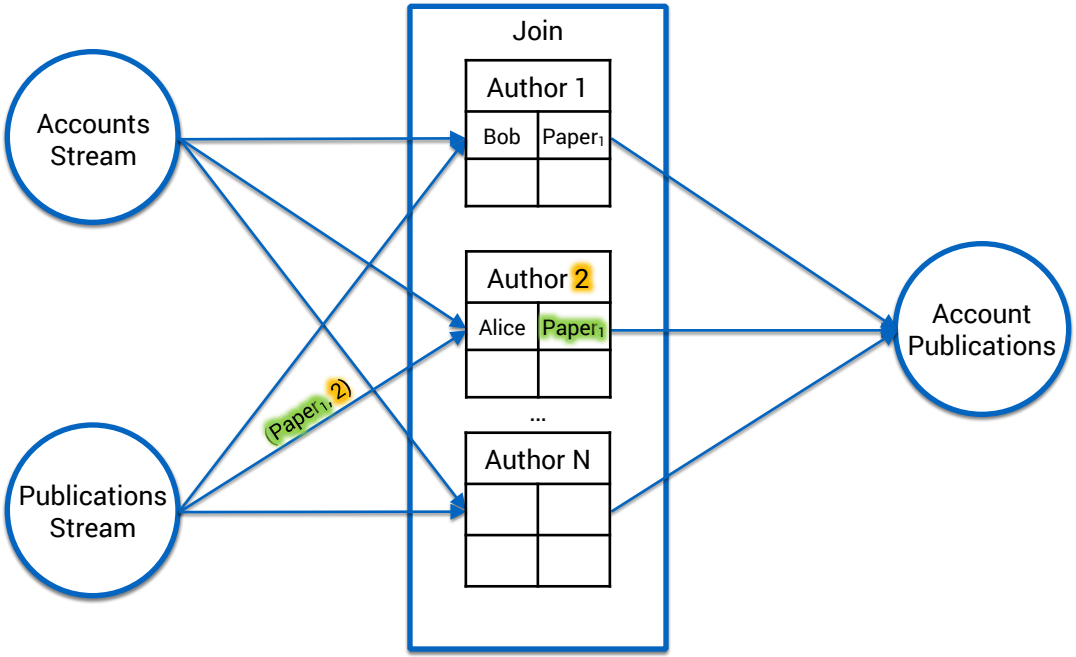


Account Publications	
K_1	(Bob, $Paper_1$)

$Paper_1$ gets updated

Accounts	
Alice	2
Bob	1

Publications	
$Paper_1$	1
$Paper_1$	2

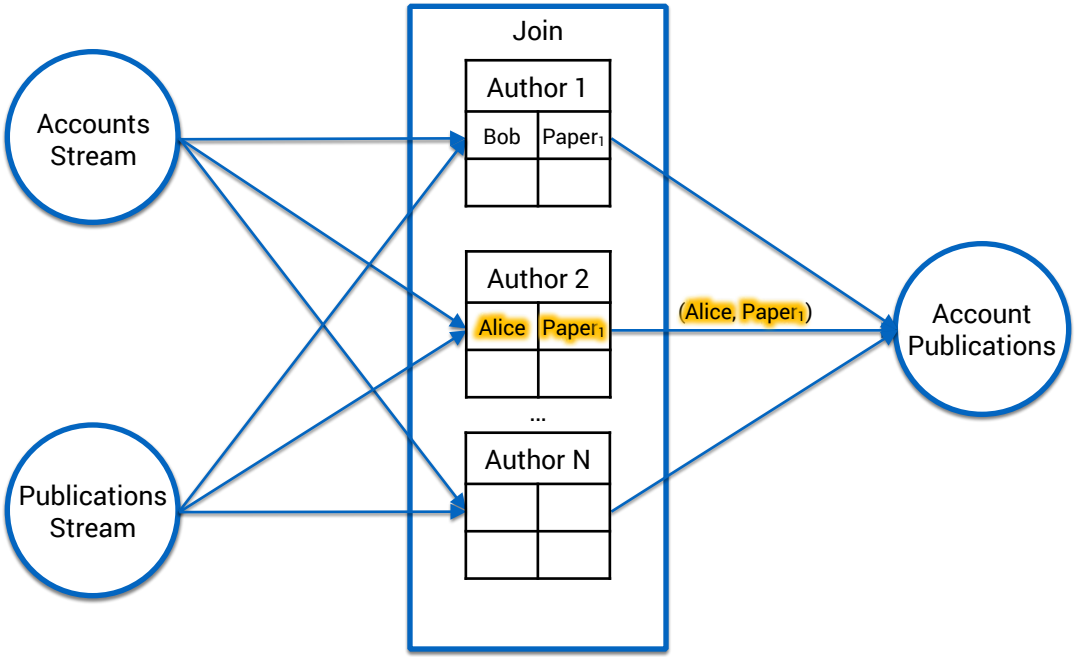


Account Publications	
K_1	(Bob, $Paper_1$)

$Paper_1$ gets updated

Accounts	
Alice	2
Bob	1

Publications	
$Paper_1$	1
$Paper_1$	2

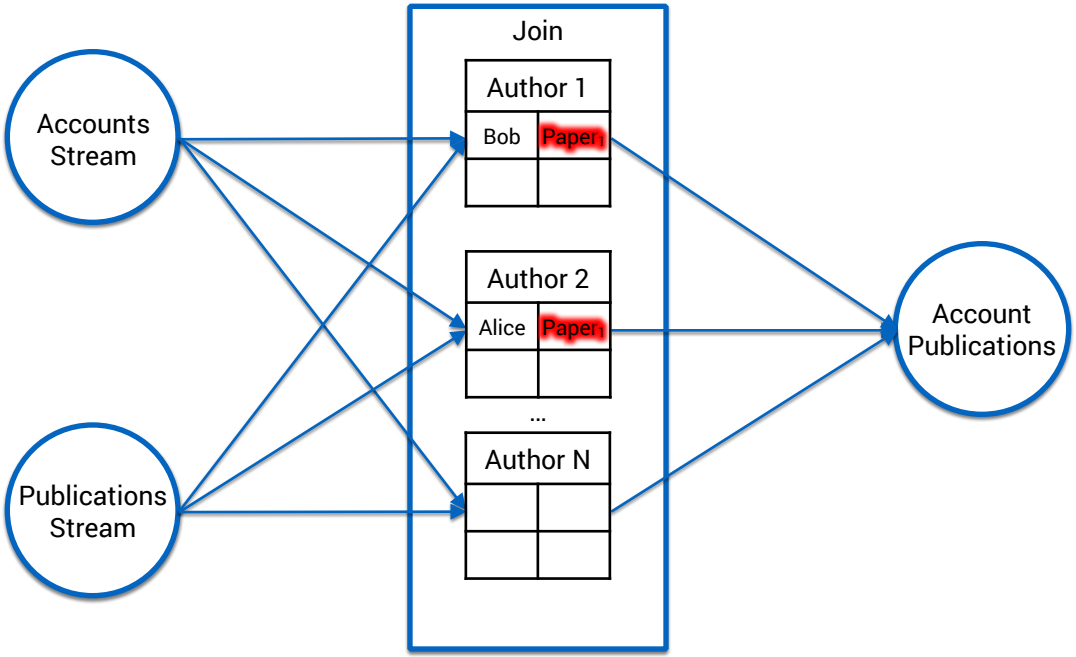


Account Publications	
K_1	(Bob, $Paper_1$)
	(Alice, $Paper_1$)

$Paper_1$ gets updated

Accounts	
Alice	2
Bob	1

Publications	
$Paper_1$	1
$Paper_1$	2

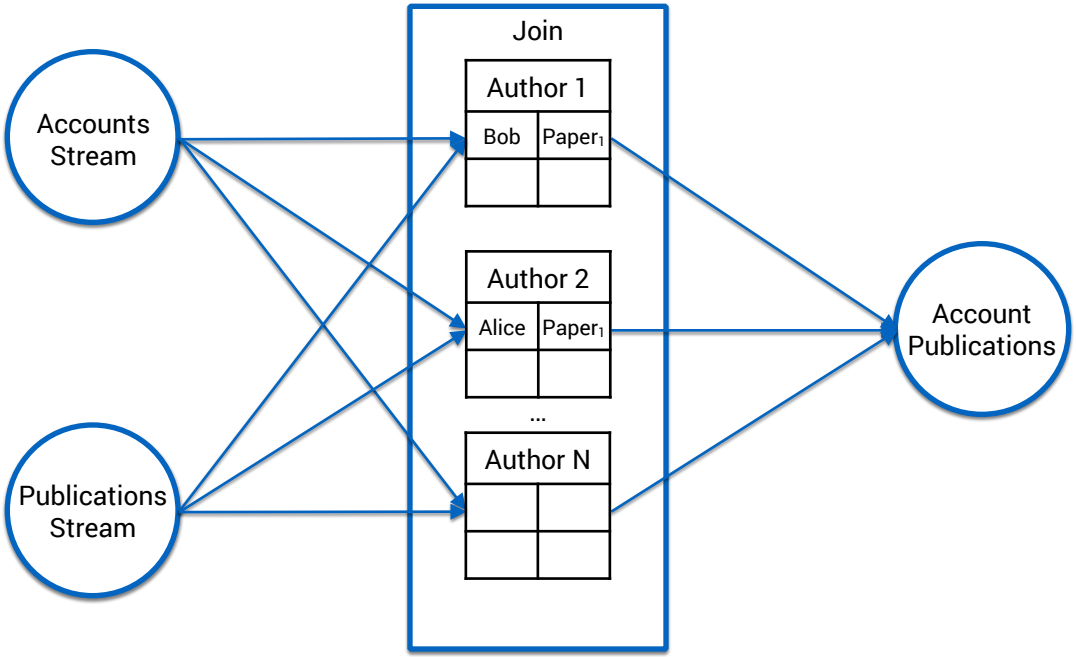


Account Publications	
K_1	(Bob, $Paper_1$)
??	(Alice, $Paper_1$)

Alice claims $Paper_1$ via different author

Accounts	
Alice	2
Bob	1

Publications	
$Paper_1$	1
$Paper_1$	(1, 2)

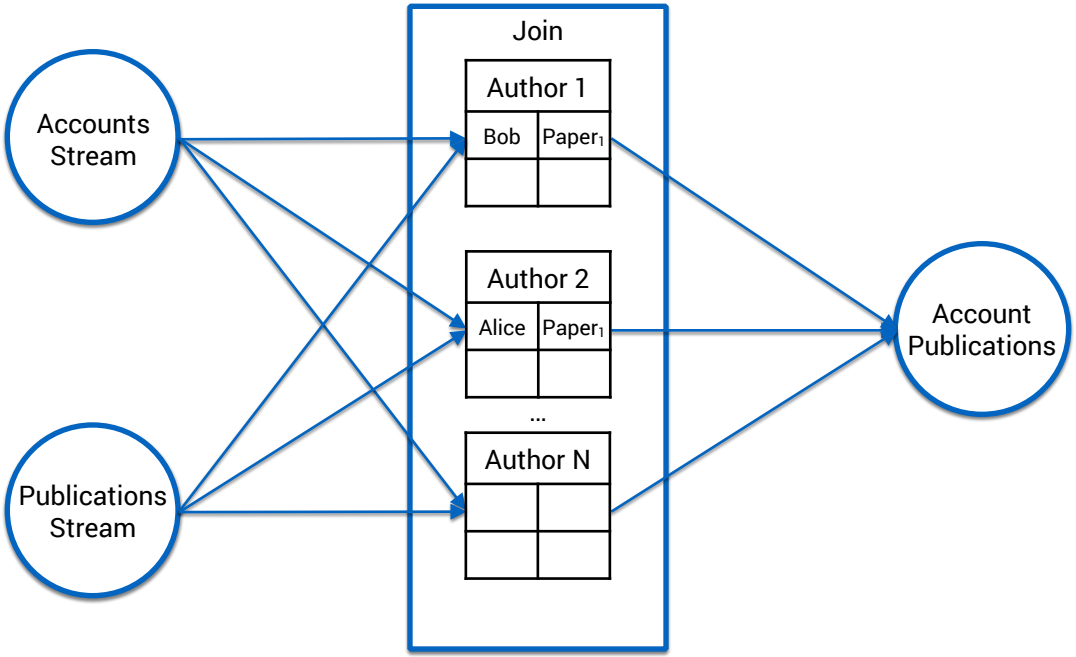


Account Publications	
K_1	(Bob, $Paper_1$)
K_2	(Alice, $Paper_1$)

Alice claims $Paper_1$ via different author

Accounts	
Alice	2
Bob	1
Bob	Ø

Publications	
$Paper_1$	1
$Paper_1$	(1, 2)

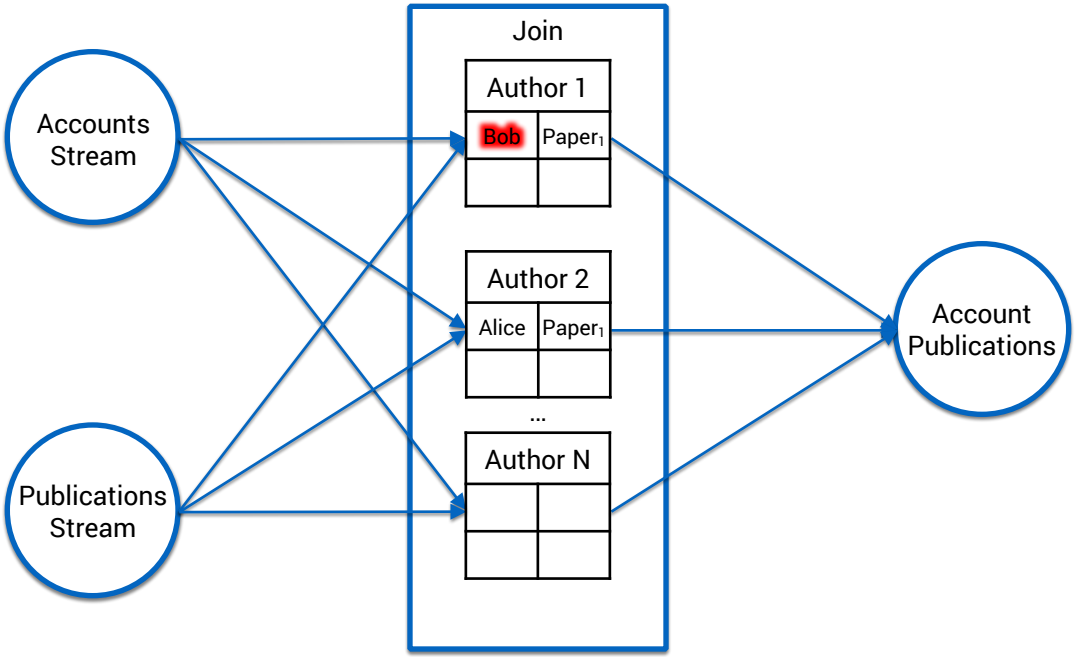


Account Publications	
K_1	(Bob, $Paper_1$)
K_2	(Alice, $Paper_1$)

Alice claims $Paper_1$ via different author

Accounts	
Alice	2
Bob	1
Bob	Ø

Publications	
$Paper_1$	1
$Paper_1$	(1, 2)

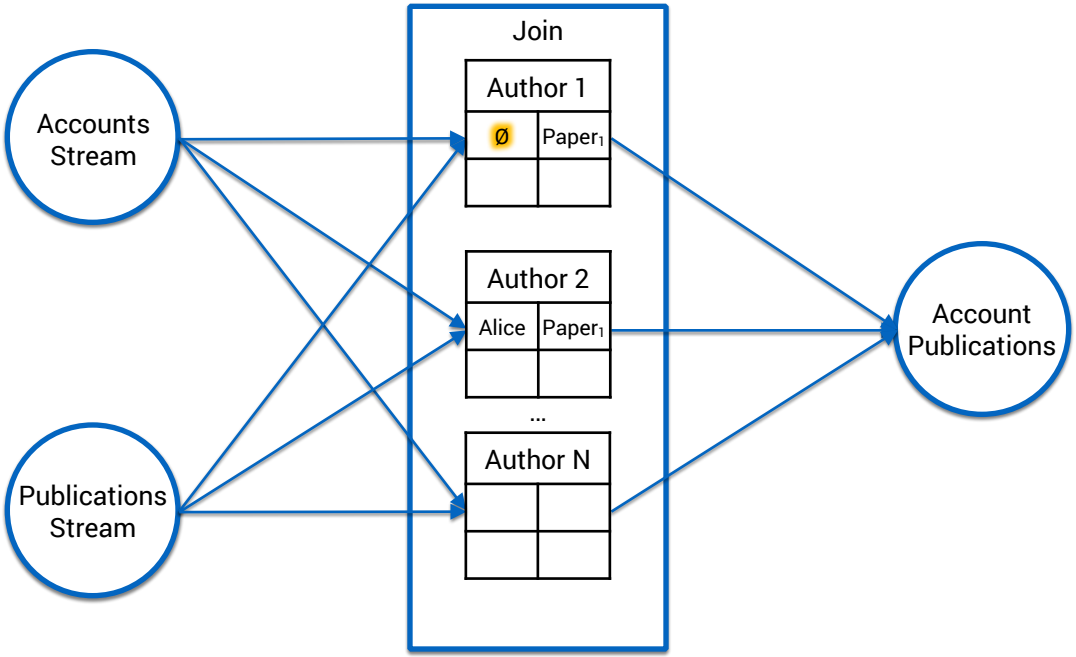


Account Publications	
K_1	(Bob, $Paper_1$)
K_2	(Alice, $Paper_1$)

Alice claims $Paper_1$ via different author

Accounts	
Alice	2
Bob	1
Bob	Ø

Publications	
$Paper_1$	1
$Paper_1$	(1, 2)

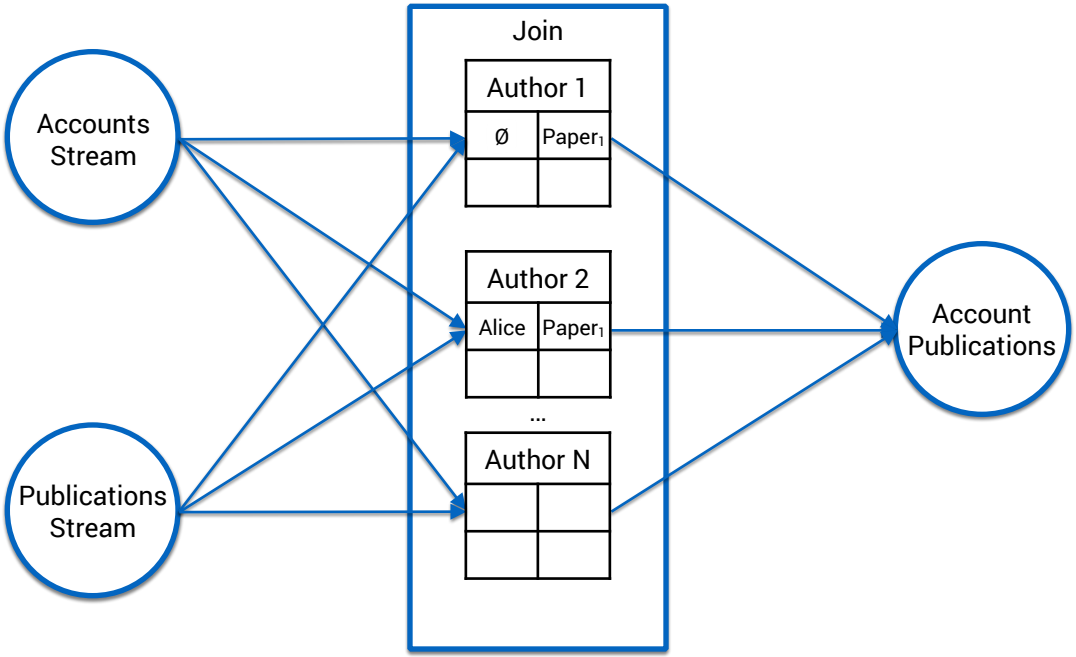


Account Publications	
K_1	(Bob, $Paper_1$)
K_2	(Alice, $Paper_1$)
K_1	Ø

Alice claims $Paper_1$ via different author

Accounts	
Alice	2
Bob	1
Bob	Ø

Publications	
$Paper_1$	1
$Paper_1$	(1, 2)

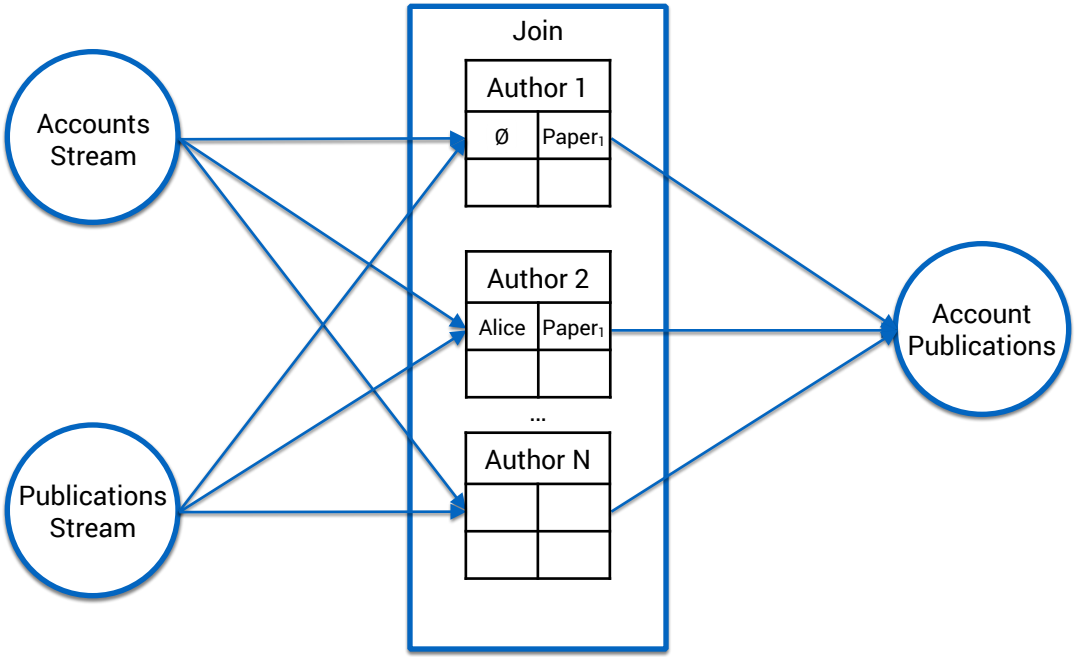


Account Publications	
K_1	(Bob, $Paper_1$)
K_2	(Alice, $Paper_1$)
K_1	Ø

Alice claims $Paper_1$ via different author

Accounts	
Alice	2
Bob	1
Bob	Ø
Alice	1

Publications	
$Paper_1$	1
$Paper_1$	(1, 2)

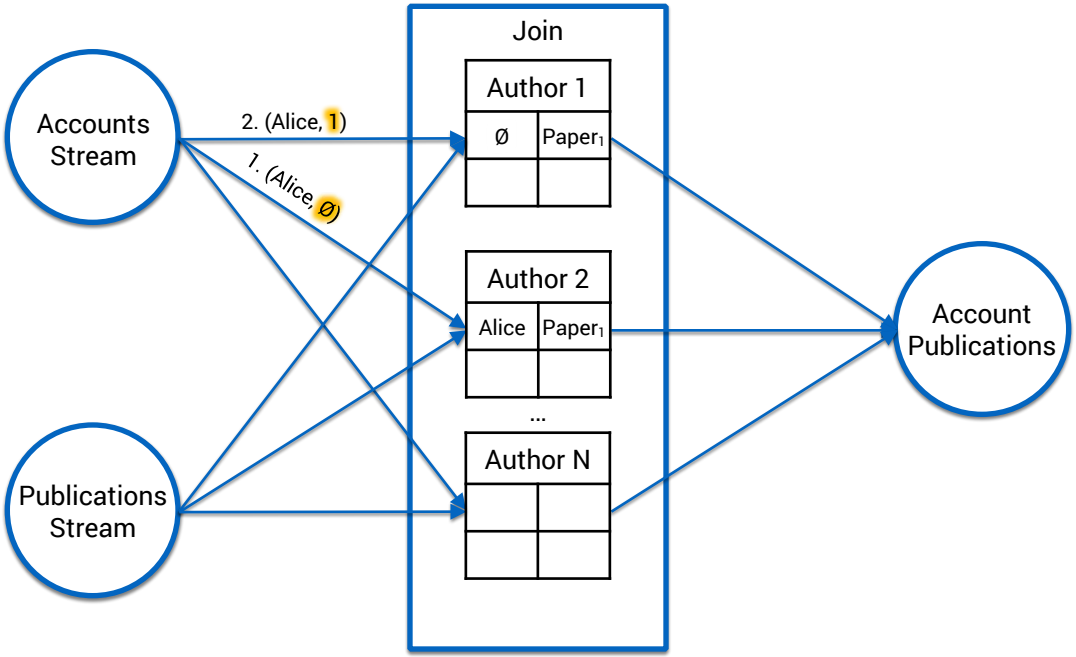


Account Publications	
K_1	(Bob, $Paper_1$)
K_2	(Alice, $Paper_1$)
K_1	Ø

Alice claims $Paper_1$ via different author

Accounts	
Alice	2
Bob	1
Bob	Ø
Alice	1

Publications	
$Paper_1$	1
$Paper_1$	(1, 2)

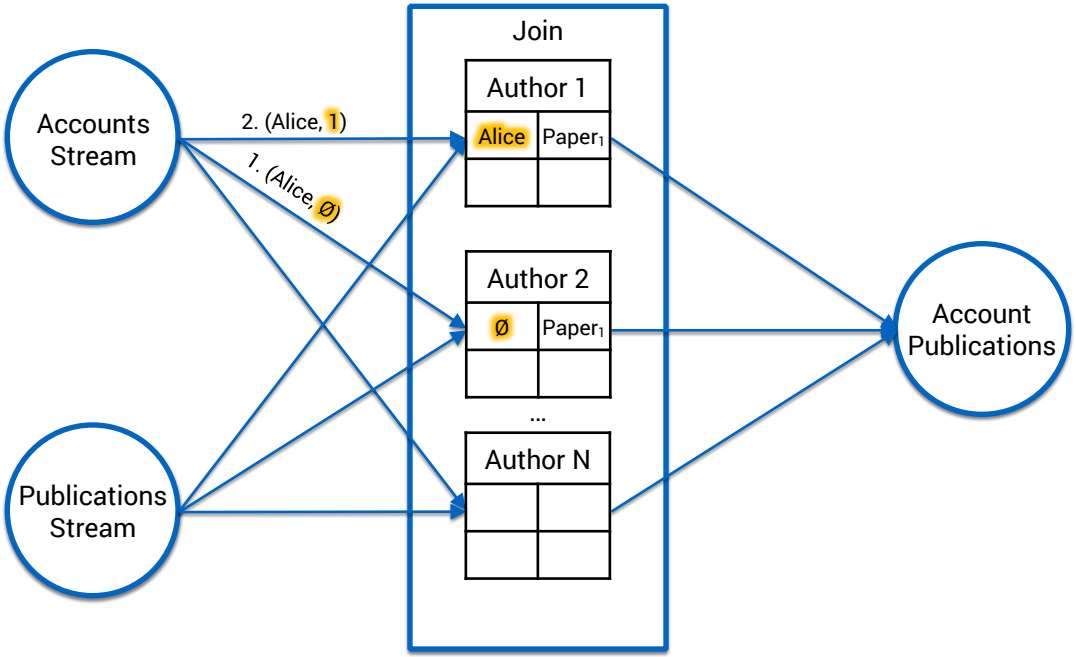


Account Publications	
K_1	(Bob, $Paper_1$)
K_2	(Alice, $Paper_1$)
K_1	Ø

Alice claims $Paper_1$ via different author

Accounts	
Alice	2
Bob	1
Bob	Ø
Alice	1

Publications	
$Paper_1$	1
$Paper_1$	(1, 2)

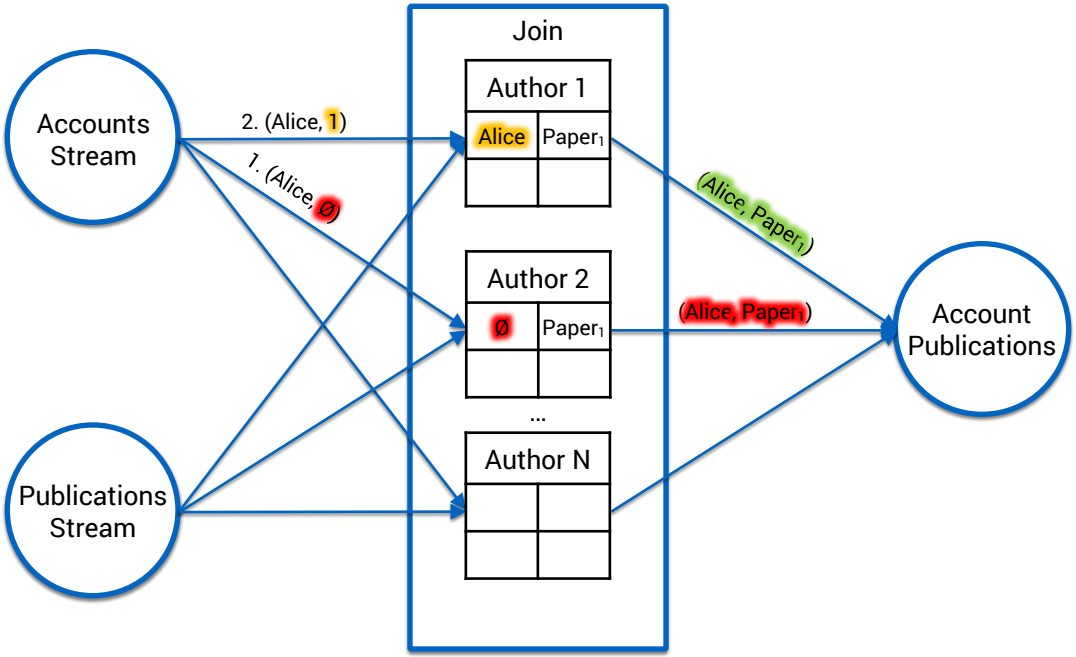


Account Publications	
K_1	(Bob, $Paper_1$)
K_2	(Alice, $Paper_1$)
K_1	Ø

Alice claims $Paper_1$ via different author

Accounts	
Alice	2
Bob	1
Bob	Ø
Alice	1

Publications	
$Paper_1$	1
$Paper_1$	(1, 2)

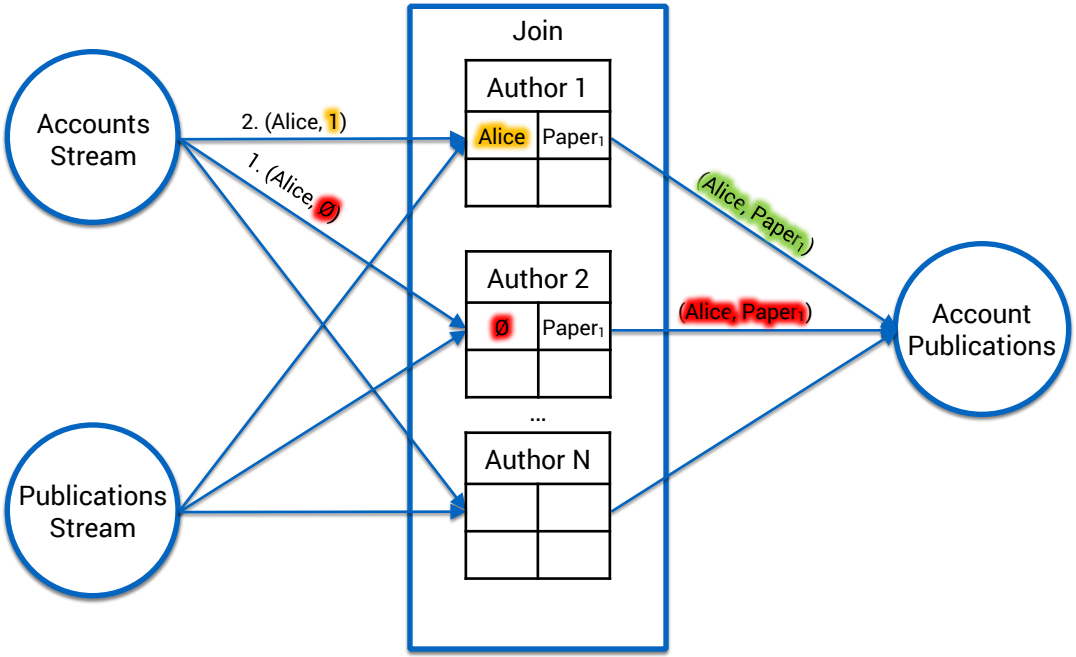


Account Publications	
K_1	(Bob, $Paper_1$)
K_2	(Alice, $Paper_1$)
K_1	Ø

Alice claims $Paper_1$ via different author

Accounts	
Alice	2
Bob	1
Bob	Ø
Alice	1

Publications	
$Paper_1$	1
$Paper_1$	(1, 2)

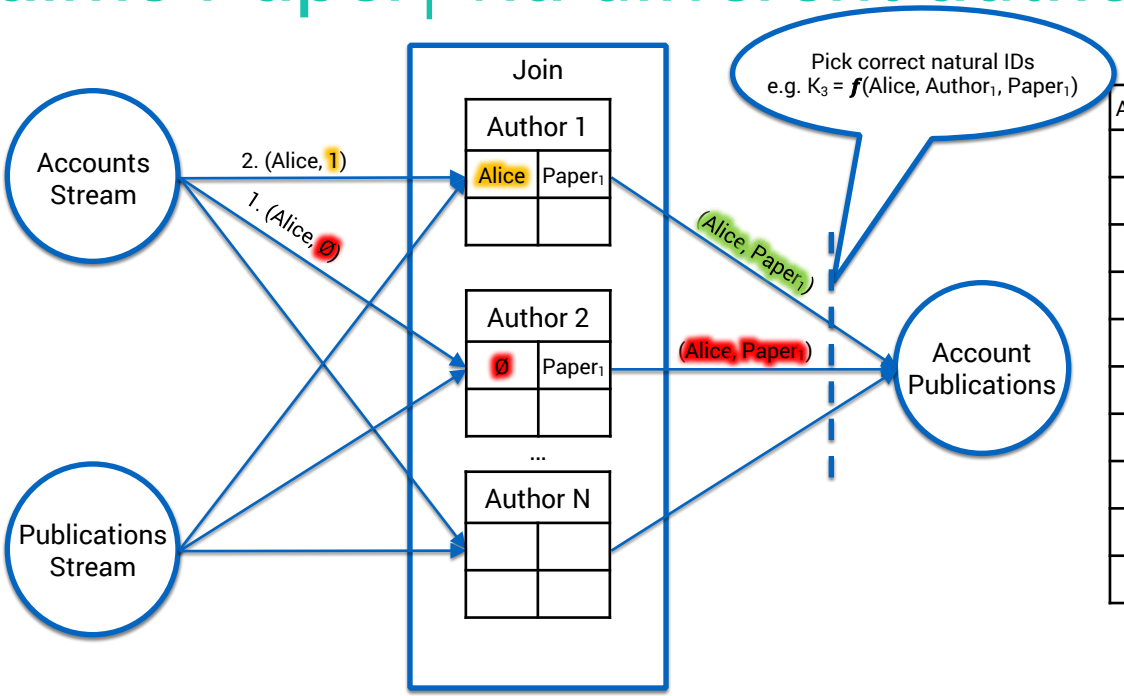


Account Publications	
K_1	(Bob, $Paper_1$)
K_2	(Alice, $Paper_1$)
K_1	Ø
K_2	(Alice, $Paper_1$)
K_2	Ø

Alice claims $Paper_1$ via different author

Accounts	
Alice	2
Bob	1
Bob	Ø
Alice	1

Publications	
$Paper_1$	1
$Paper_1$	(1, 2)

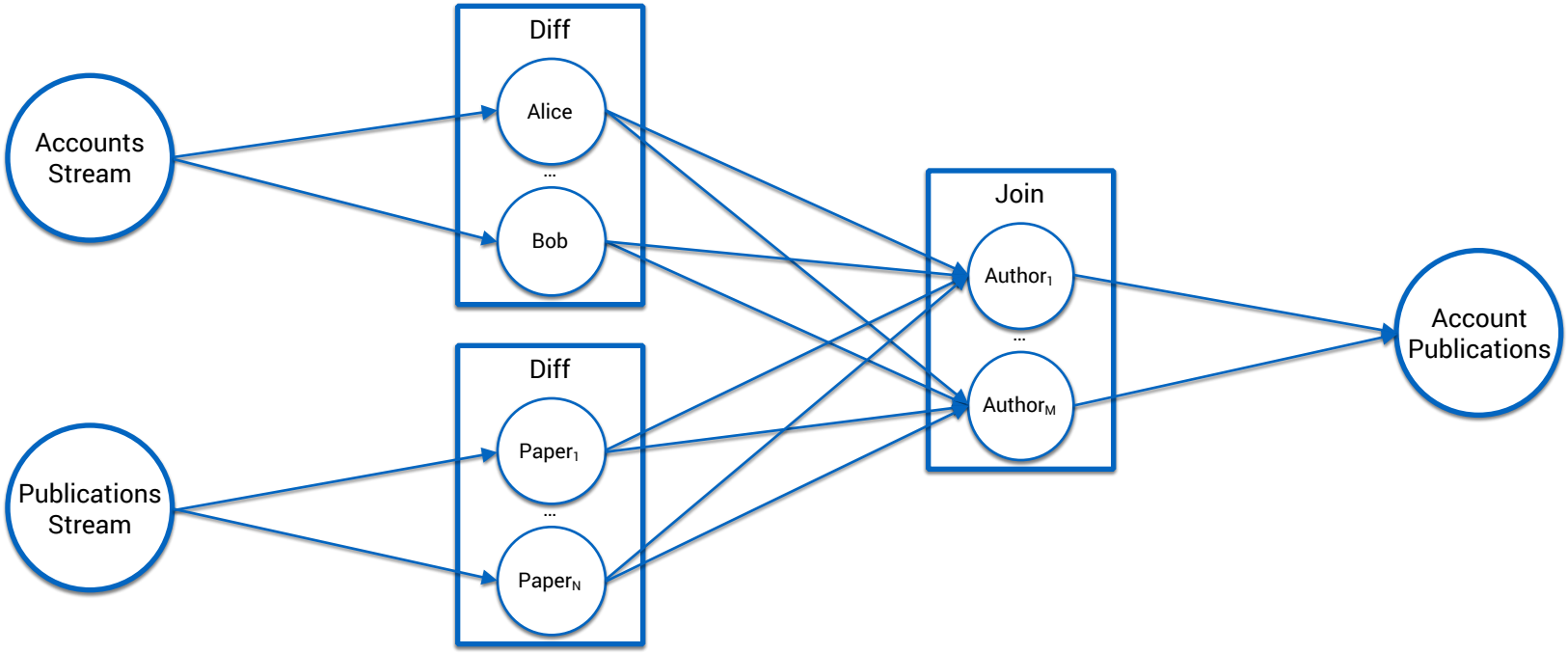


Account Publications	
K_1	(Bob, $Paper_1$)
K_2	(Alice, $Paper_1$)
K_1	Ø
K_3	(Alice, $Paper_1$)
K_2	Ø

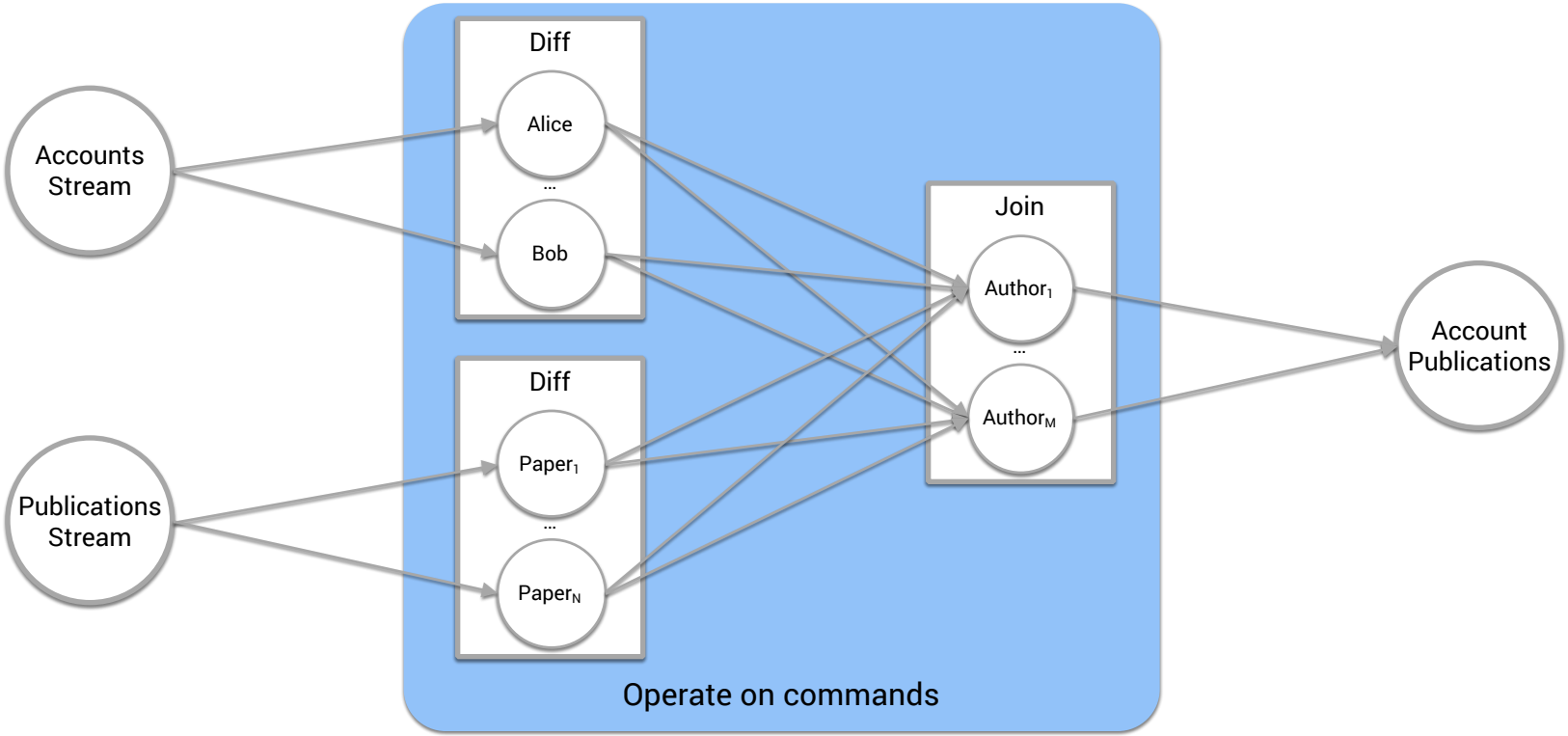
How to solve deletes and updates

- Keep **previous element state** to update previous join result
- Stream elements are not domain entities but **commands** such as delete or upsert
- Joined stream must have **natural IDs** to propagate deletes and updates

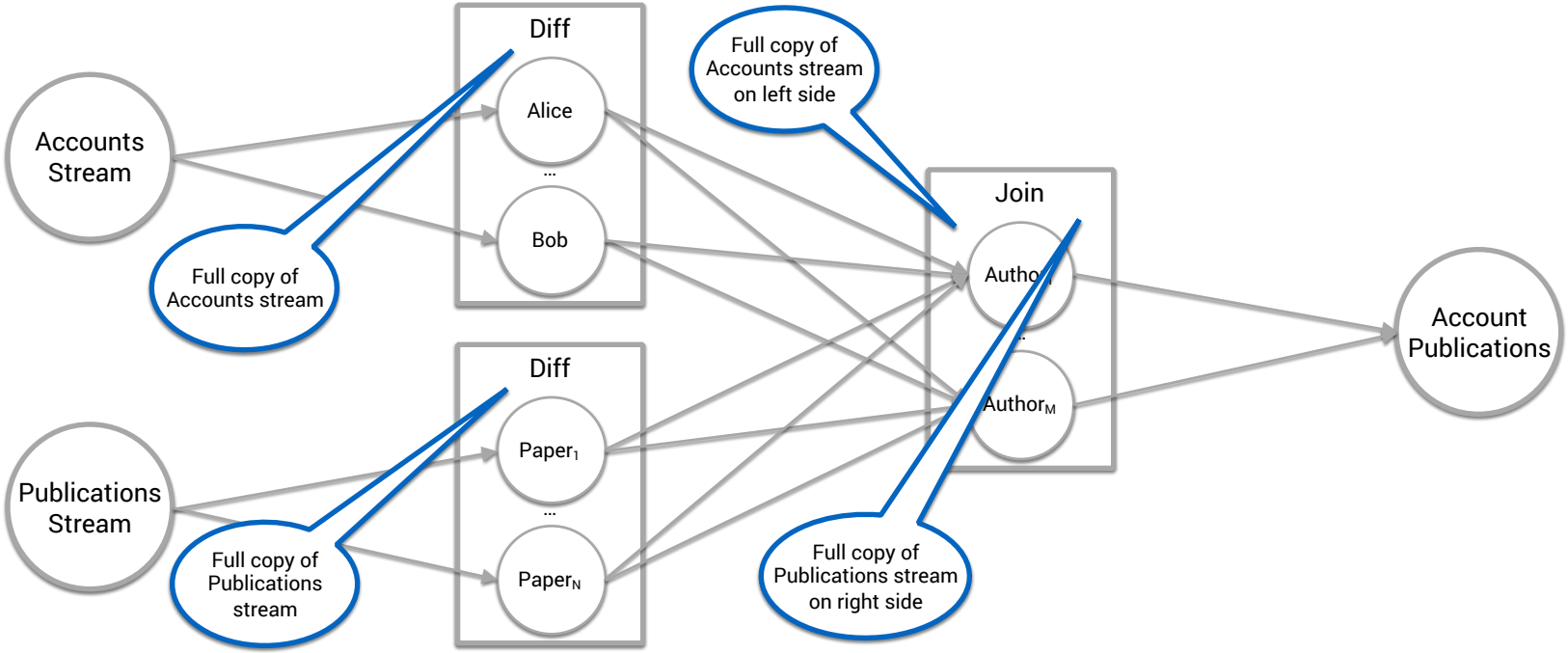
Generic join graph



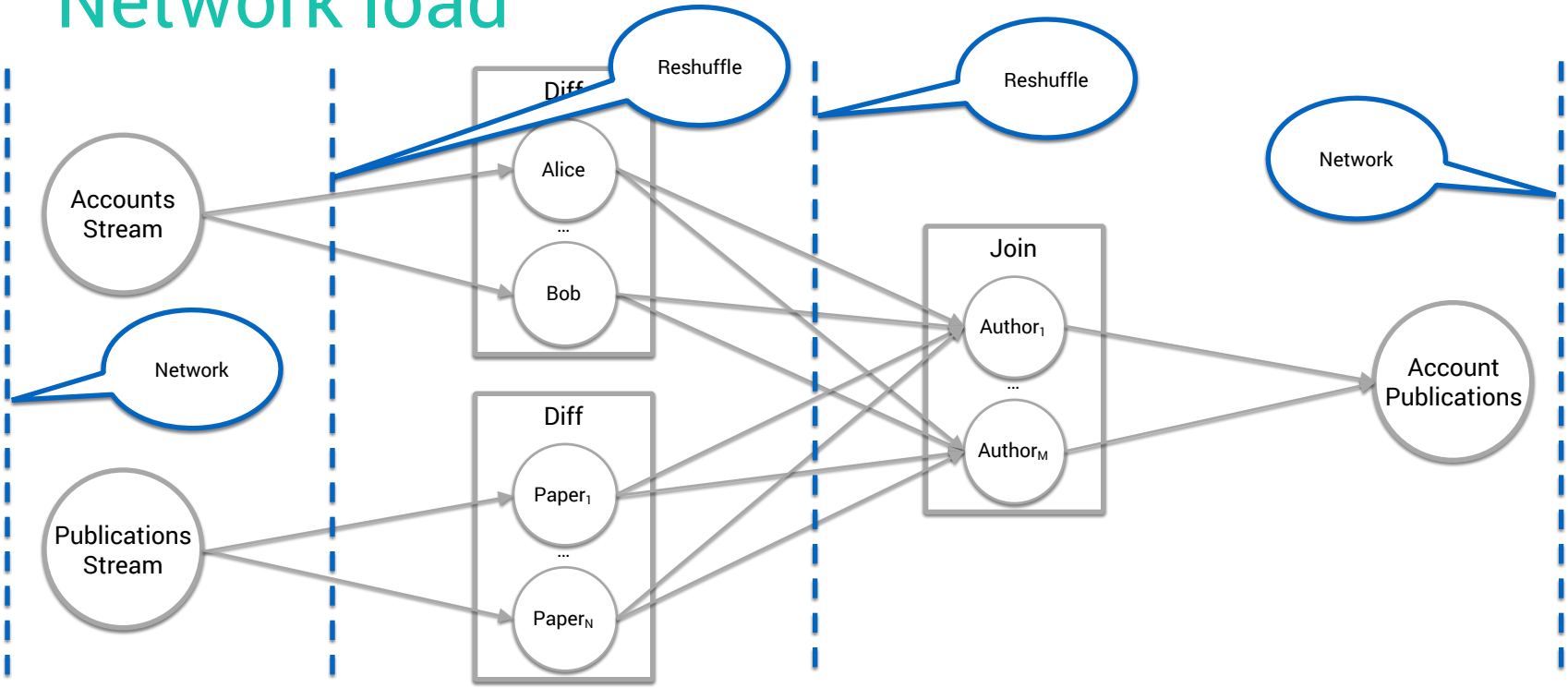
Generic join graph



Memory requirements



Network load



Resource considerations

- In addition to handling Kafka traffic we need to **reshuffle all data twice** over the network

- We need to **keep two full copies** of each joined stream in memory

Questions

We are hiring - www.researchgate.net/careers

ResearchGate

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris pharetra interdum felis, sit amet aliquet mauris. Proin non fermentum sem. Vivamus a ligula vel arcu convallis porttitor.

Background

General properties

Human adenoviruses (HAdV) are causing a broad spectrum of diseases. One of the most severe forms of adenovirus infection is a disseminated disease resulting in significant morbidity and mortality. Several reports in recent years have identified HAdV-31 from species A (HAdV-A31) as a cause of disseminated disease in children following haematopoetic stem cell transplantation (hSCT) and liver transplantation. We sequenced and analyzed the complete genome of the HAdV-A31 prototype strain to uncover unique sequence motifs associated with its high virulence. Moreover, we sequenced coding regions[5] known to be essential for tropism and virulence (early transcription units E1A, E3, E4, the fiber knob and the penton base) of HAdV-A31 clinical isolates from patients with