University of Applied Sciences
HOCHSCHULE EMDEN·LEER Networked Systems

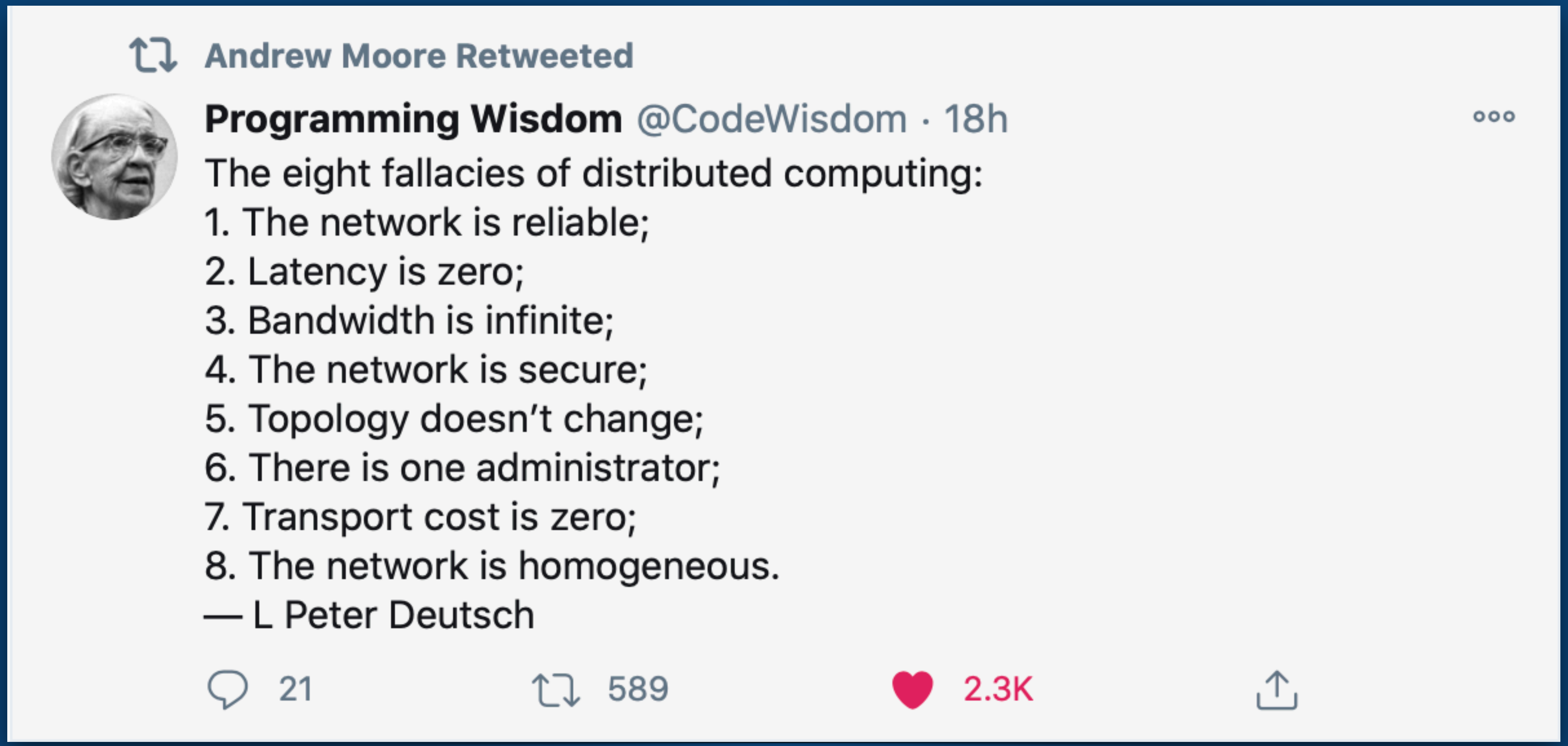
PICCOLO

Information-Centric Dataflow

Re-Imagining Reactive Distributed Computing

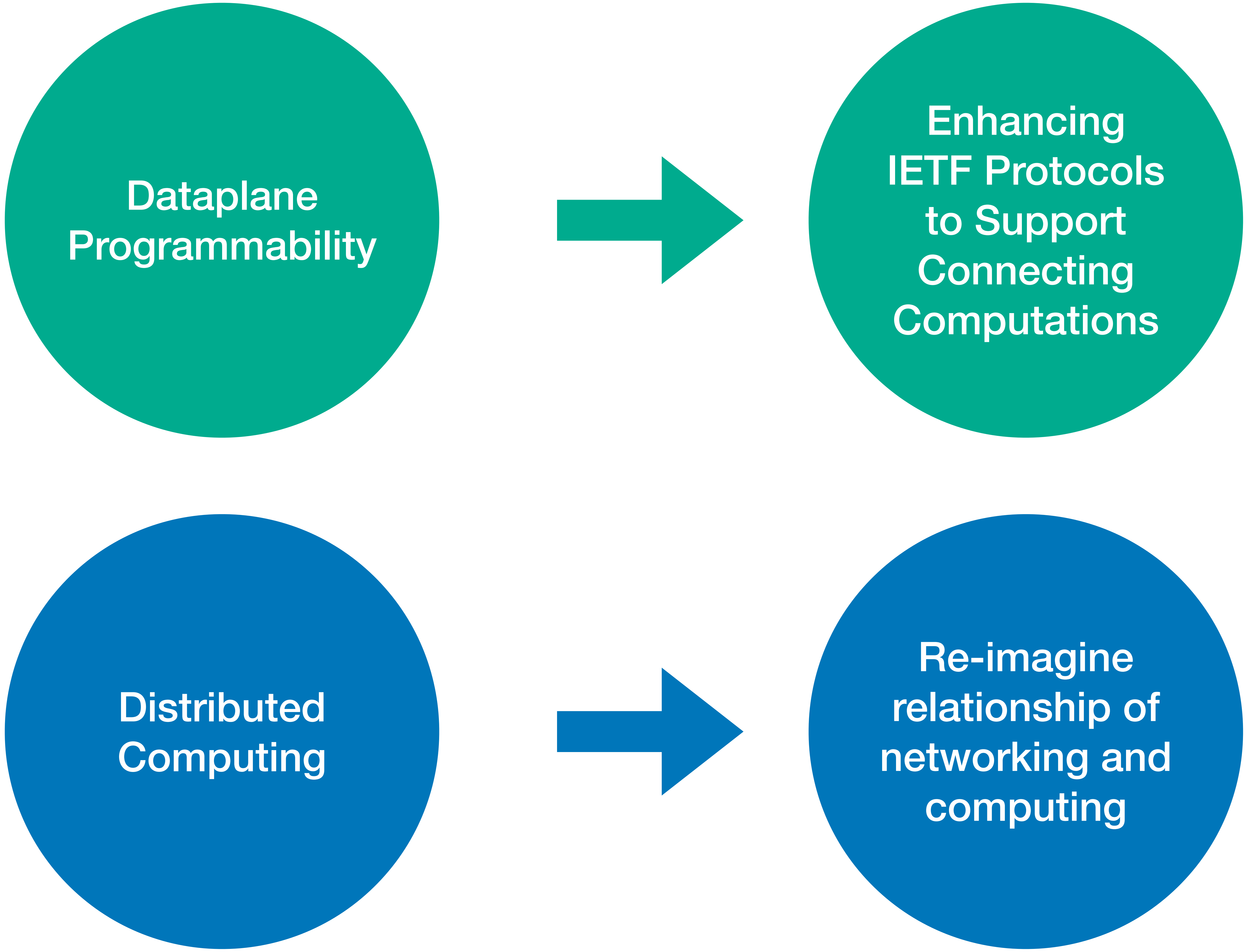
Dirk Kutscher, Laura Al Wardani, T M Rayhan Gias

IRTF COINRG Meeting at IETF-112
2021-11-11



COIN

My Perspective: Two strands



This work

ACM ICN-2021

Vision: Information-Centric Dataflow – Re-Imagining Reactive Distributed Computing

Dirk Kutscher
University of Applied Sciences Emden/Leer
Emden, Germany
Dirk.Kutscher@hs-emden-leer.de

Laura Al Wardani
University of Applied Sciences Emden/Leer
Emden, Germany
laura.al.wardani@hs-emden-leer.de

T M Rayhan Gias
University of Applied Sciences Emden/Leer
Emden, Germany
rayhan.gias@hs-emden-leer.de

ABSTRACT

This paper describes an Information-Centric Dataflow system that is based on name-based access to computation results, NDN PSync dataset synchronization for enabling consuming compute functions to learn about updates and for coordinating the set of compute functions in a distributed Dataflow pipeline. We describe how relevant Dataflow concepts can be mapped to ICN and how data-sharing, data availability and scalability can be improved compared to state-of-the-art systems. We also provide a specification of an application-independent namespace design and report on our experience with a first prototype implementation.

CCS CONCEPTS

• **Networks** → **Network architectures**; **Network protocols**; **Application layer protocols**;

KEYWORDS

Information-Centric Networking, Distributed Computing, Dataflow

1 INTRODUCTION

The Dataflow paradigm is a popular distributed computing abstraction that is leveraged by several popular data processing frameworks such as Apache Flink [12] and Google Dataflow [4]. Fundamentally, Dataflow is based on the concept of asynchronous messaging between computing nodes, where data controls program execution, i.e., computations are triggered by incoming data and associated conditions. This typically leads to very modular system architectures that enable re-use, re-composition, and parallel execution naturally. Most of the popular distributed processing frameworks today are implemented as overlays, i.e., they allow for instantiating computations and for inter-connecting them, for example by creating and maintaining communication channels between nodes such as system processes and microservices.

We claim that the connection-based approach incurs several architectural problems and inefficiencies, for example: application logic is concerned with receiving and producing data as a result of computation processes but **connections imply transport endpoint addresses that are typically not congruent**. This typically implies a mapping or orchestration system. One key goal for Dataflow systems is to enable parallel execution, i.e., one computation is run in parallel, which also affects the communication relationships with upstream producers and downstream consumers. For example, when parallelizing a computation step, it typically implies that each instance is consuming a partition of the inputs instead of all the inputs. **An indirection- and connection-based approach makes it harder to configure (and especially to dynamically re-configure) such dataflow graphs**.

In some variants of Dataflow, for example stream processing, it can be attractive if one computation output can be consumed by multiple downstream functions. Connection-based overlays typically require **duplicating the data for each such connection**, incurring significant overheads. In large-scale scenarios, the computation functions may be distributed to multiple hosts that are inter-connected in a network. Orchestrators may have visibility into compute resource availability but typically have to treat the TCP/IP network as a blackbox. As a result, the actual **data flow is locked into a set of overlay connections that do not necessarily follow optimal paths**, i.e., the communication flows are incongruent with the logical data flows.

In this paper, we present *IceFlow* – an Information-Centric Dataflow system approach that supports traditional Dataflow with Information-Centric principles and that can be used as a drop-in replacement for existing Dataflow-based frameworks. IceFlow's objectives are: (*i*) reducing complexity in Dataflow systems by removing connection-based overlays and corresponding orchestration requirements; (*ii*) enabling efficient communication by reducing data duplication; and (*iii*) enabling additional improvements through more direct communication and caching in the network.

IceFlow is employing access to authenticated data in the network as per CCNx/NDN-based ICN for the communication between computation functions and provides additional features such as flow-control, partitions for data streaming, and a window concept for synchronizing computations in streaming pipelines. The contributions of this paper are: (*i*) an ICN naming scheme for Dataflow; (*ii*) a concept for receiver-driven flow control in IceFlow-based Dataflow systems and, (*iii*) for dealing with parallel processing in IceFlow-based Dataflow systems; and (*iv*) a prototype implementation.

The rest of the paper is structured as follows: section 2 presents fundamental Dataflow concepts and a problem statement. Section 3 presents IceFlow's design. We report on first implementation experience in section 4, discuss related work in section 5 and conclude this vision paper with a discussion in section 6.

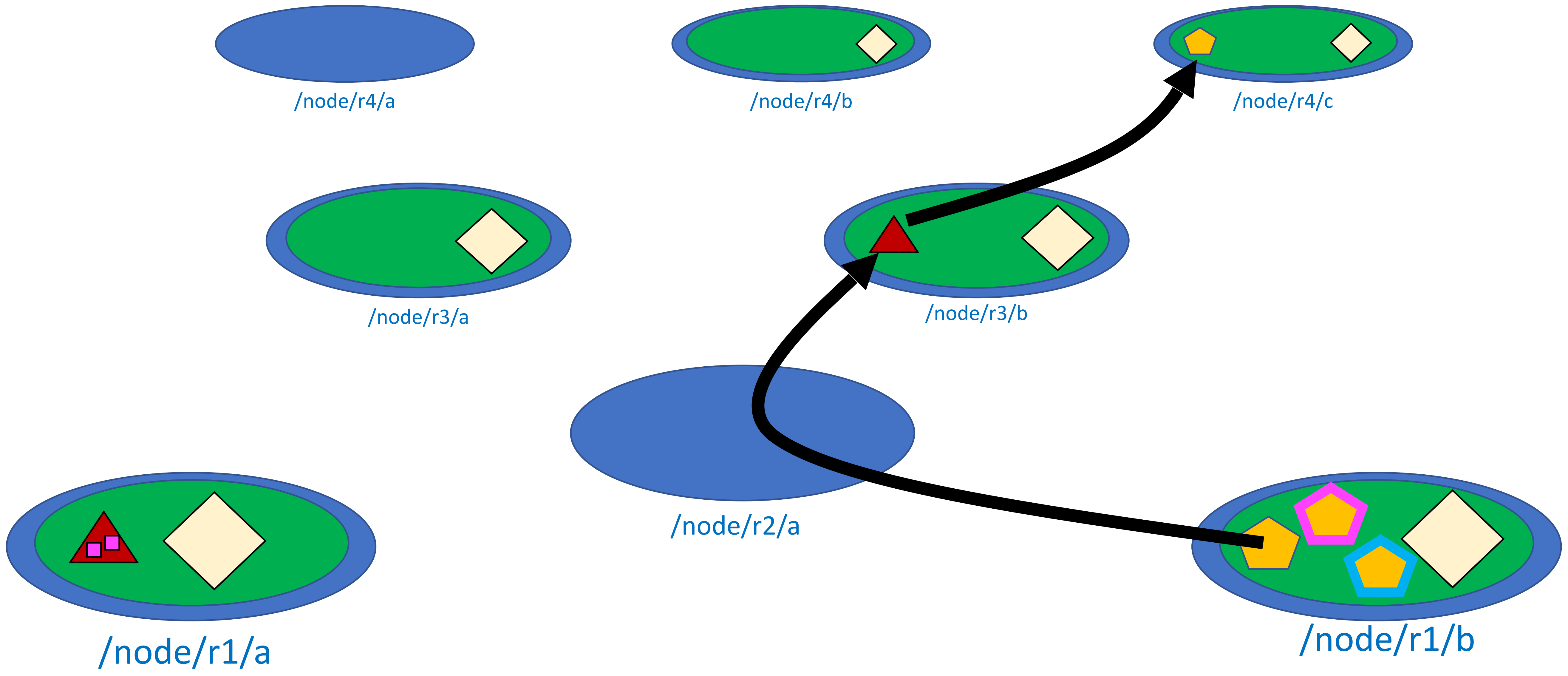
ICN '21, September 22–24, 2021, Paris, France

ACM ISBN 978-1-4503-8460-5/21/09…$15.00
https://doi.org/10.1145/3460417.3482975

Distributed Computing

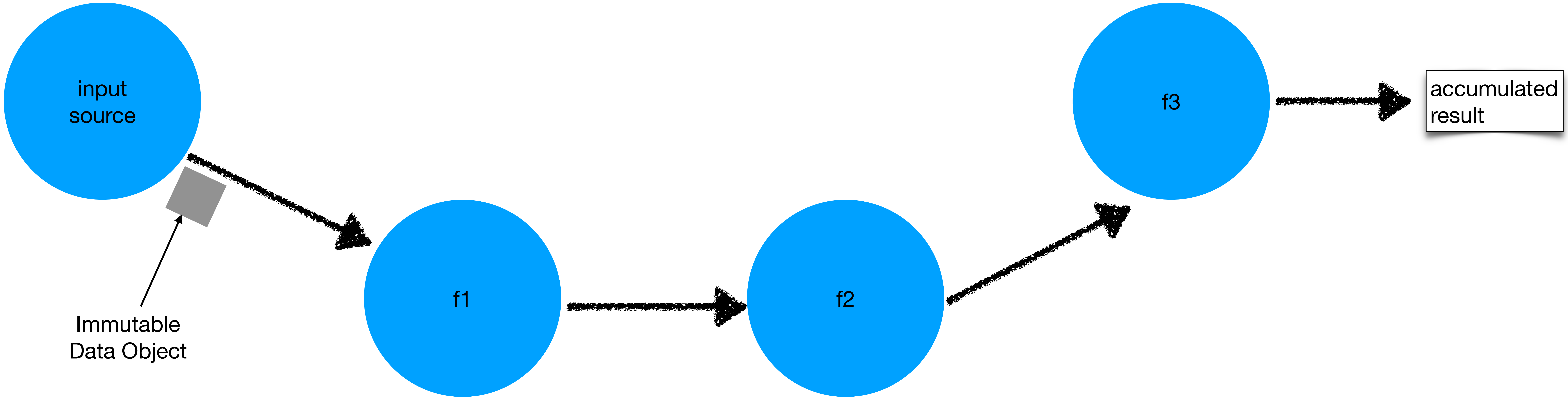
Many Different Types of Interactions

- Message passing
- Remote Method Invocation
- Dataset synchronization
- Key-value store



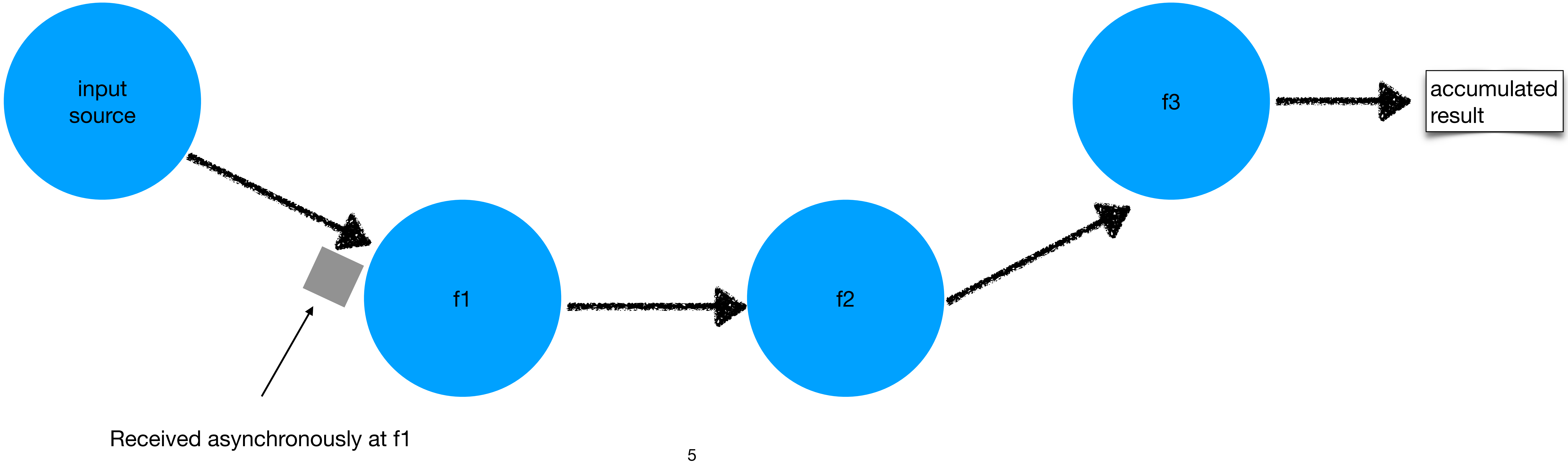
Dataflow

Structured Distributed Data Processing



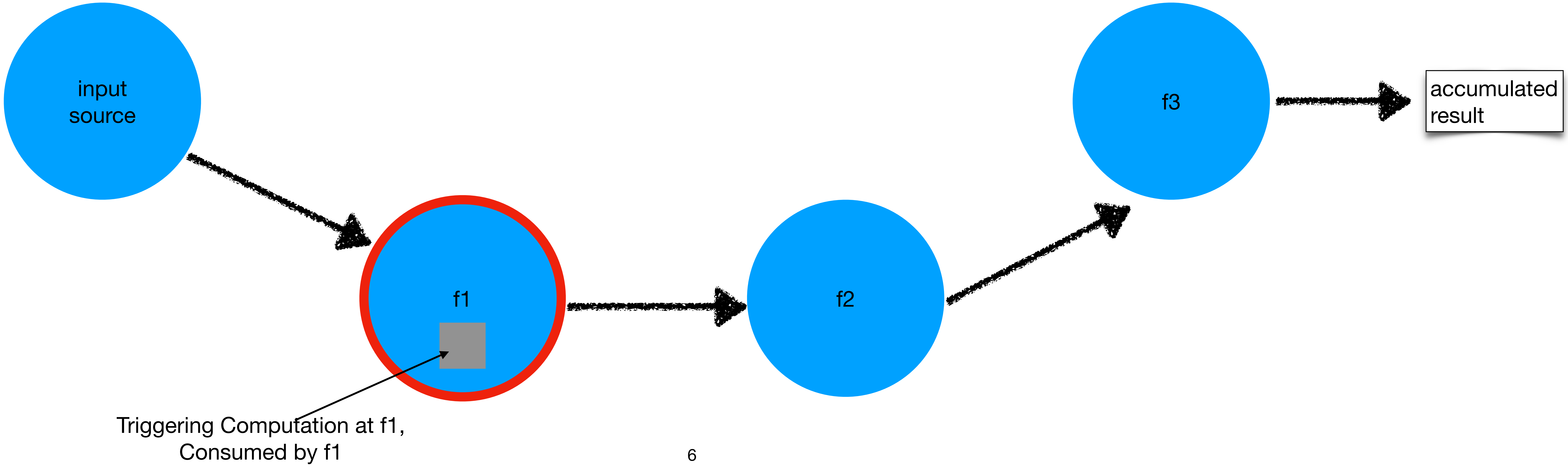
Dataflow

Structured Distributed Data Processing



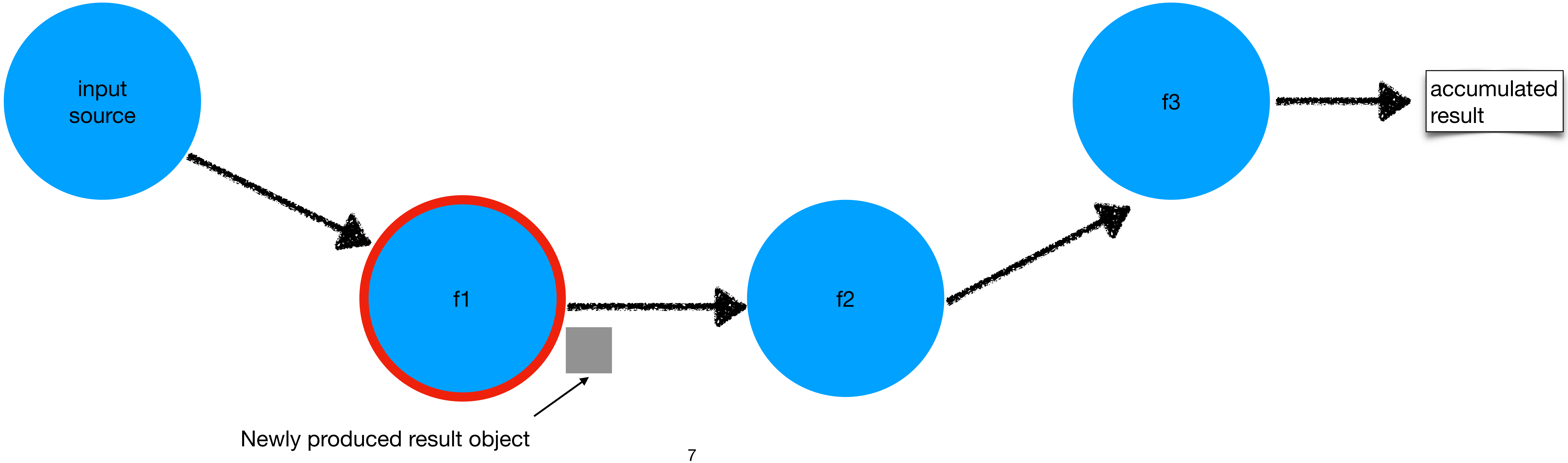
Dataflow

Structured Distributed Data Processing



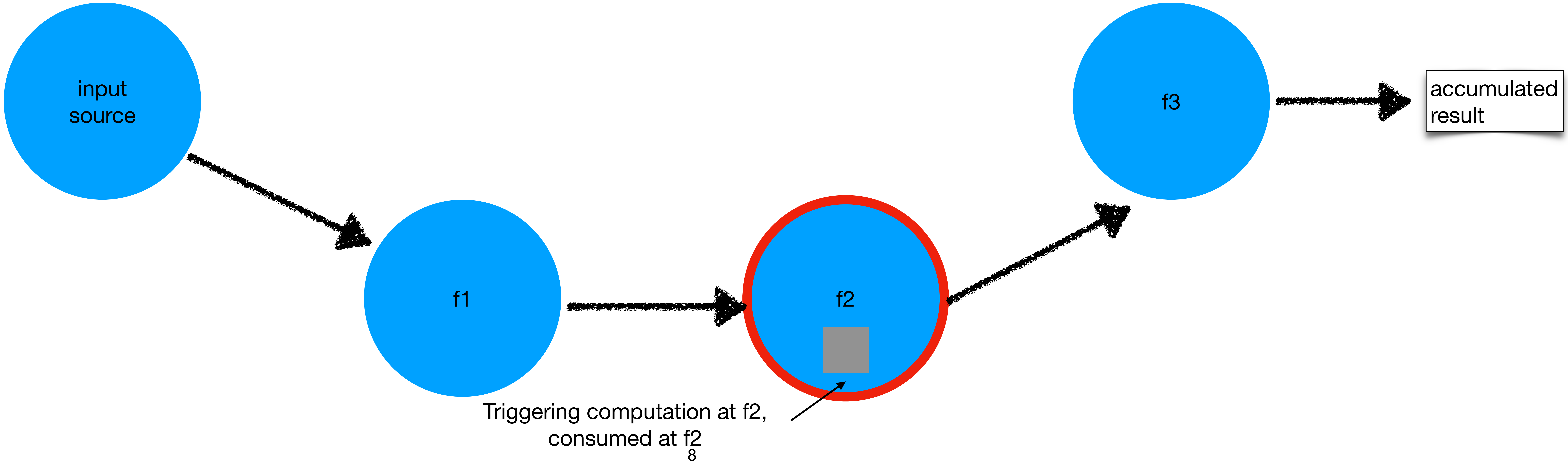
Dataflow

Structured Distributed Data Processing



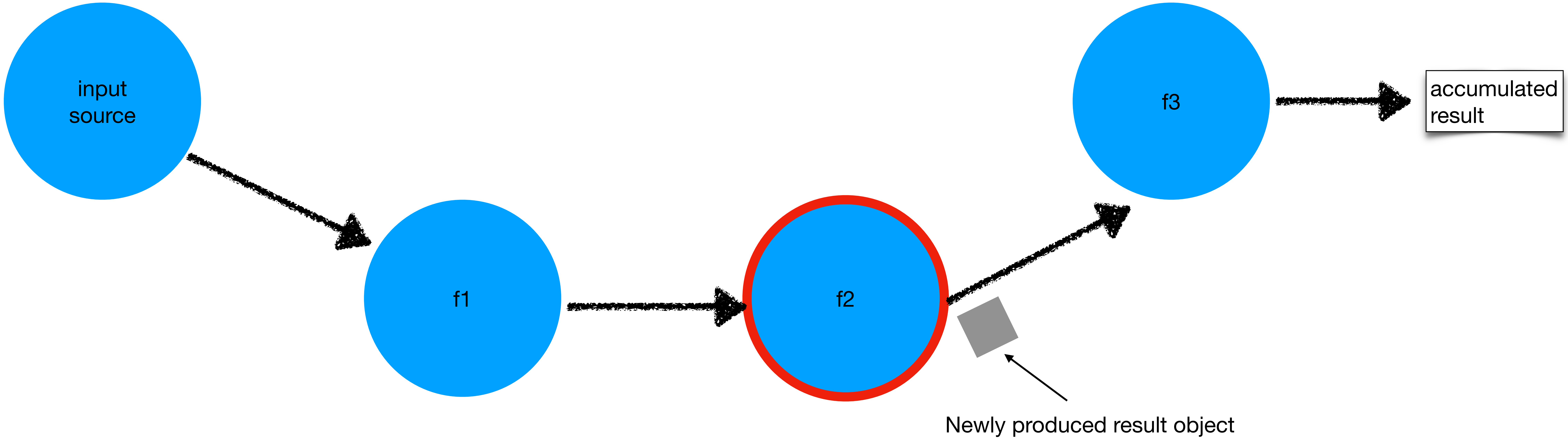
Dataflow

Structured Distributed Data Processing



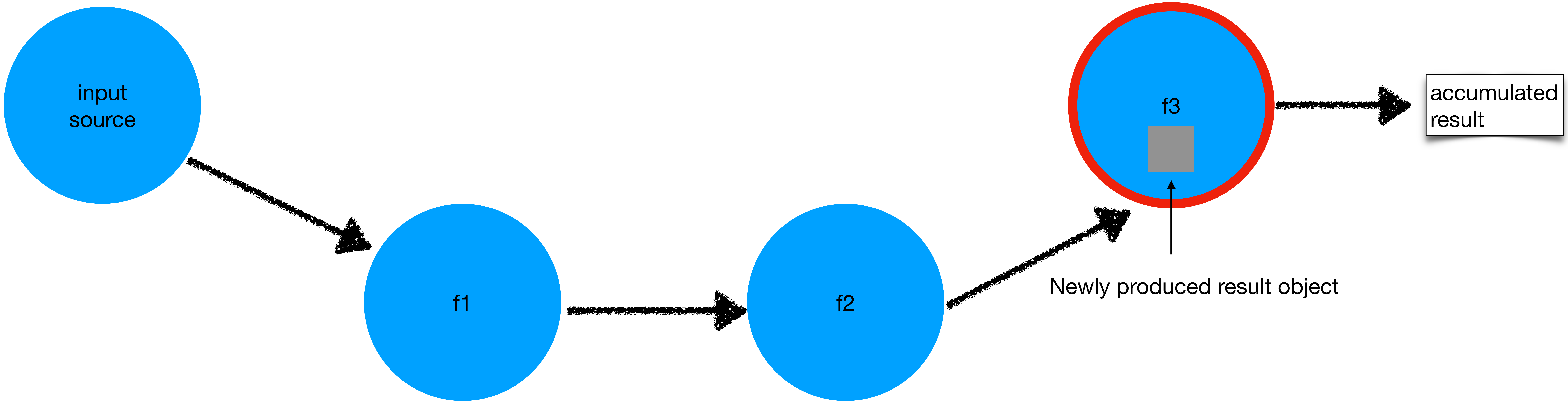
Dataflow

Structured Distributed Data Processing



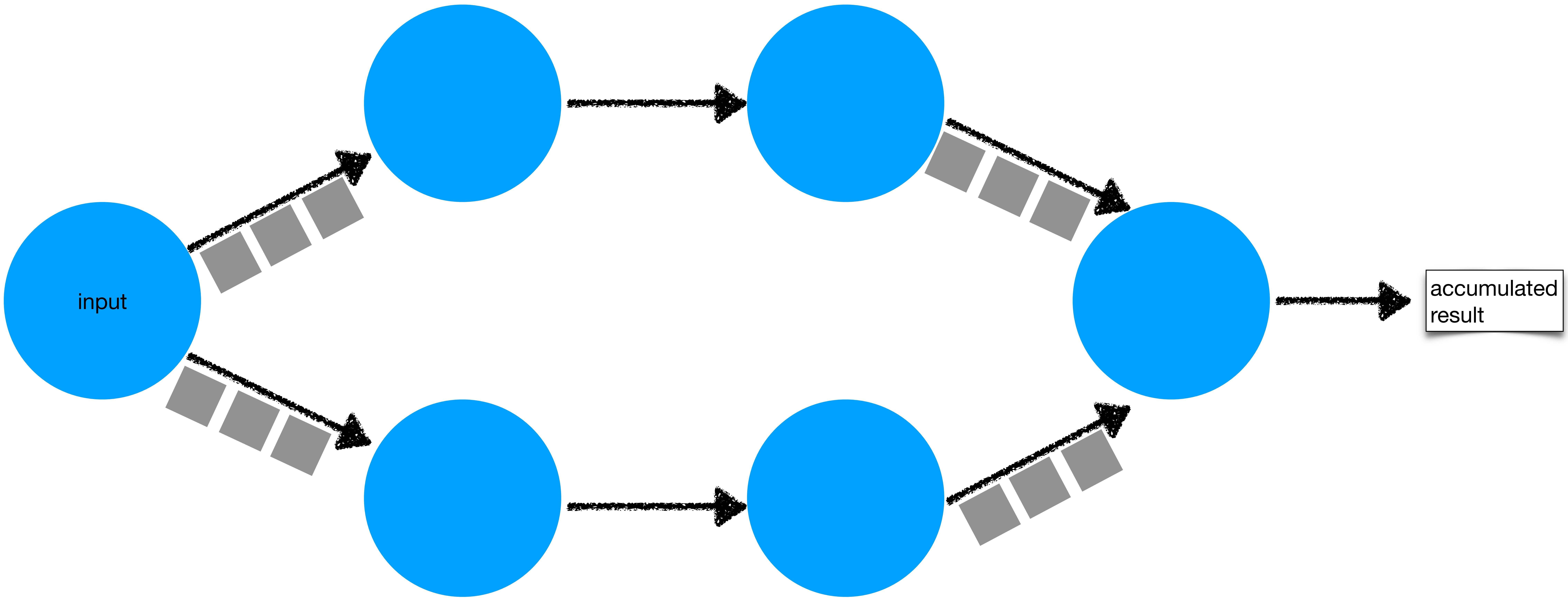
Dataflow

Structured Distributed Data Processing



Dataflow

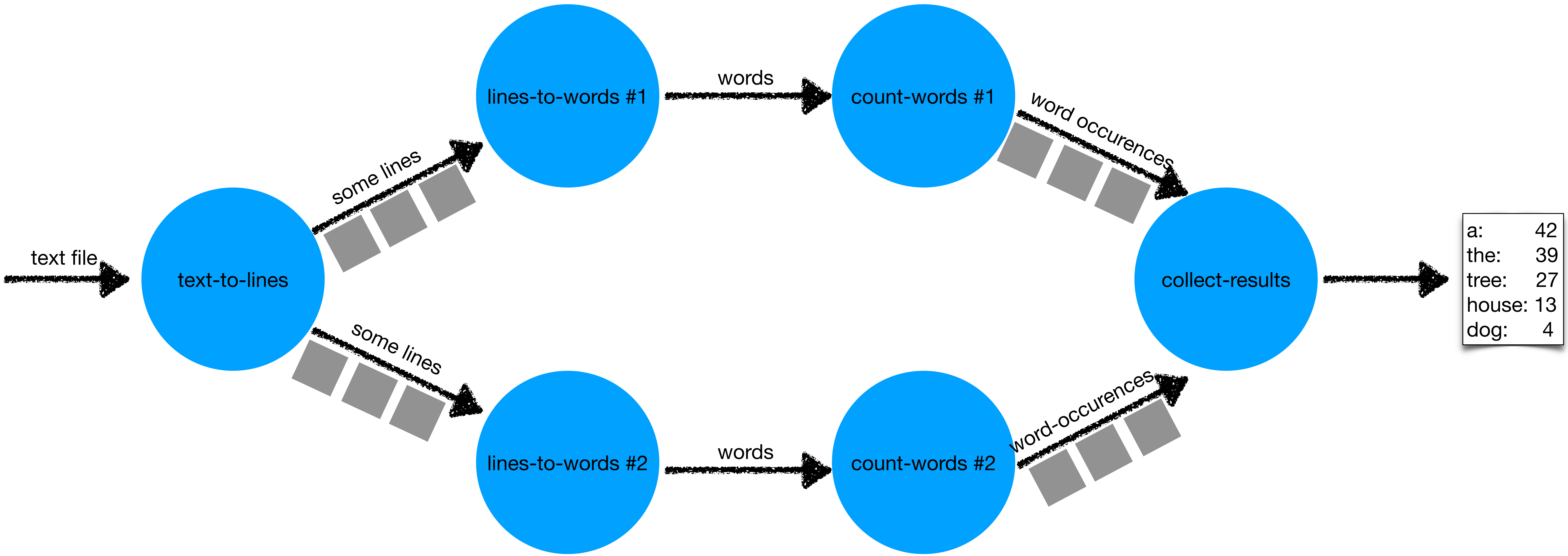
Structured Distributed Data Processing



Dataflow

Poster Child Example: word-count

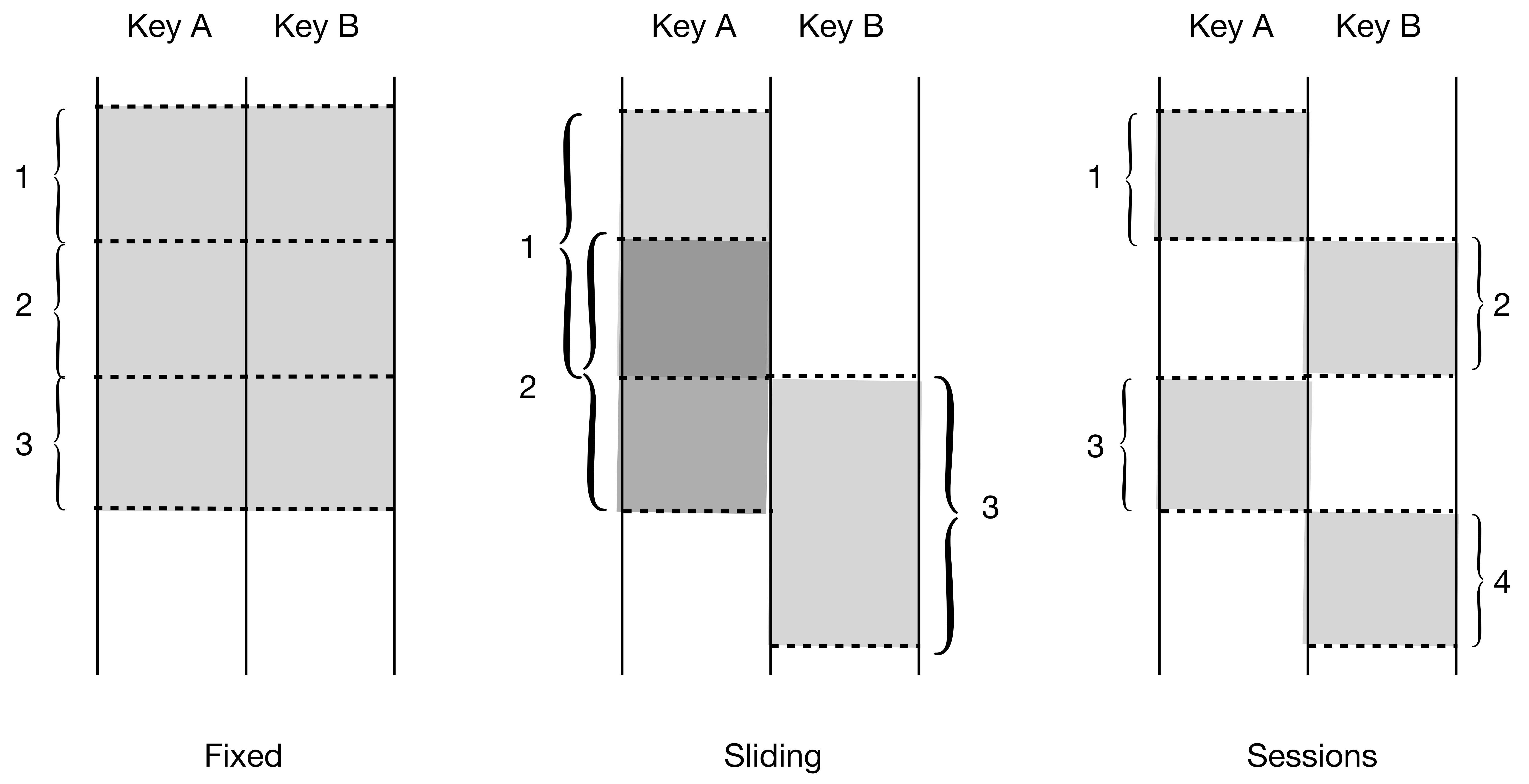
```
text-to-lines -> lines-to-words -> word-count ->
    ↪ collect-results
```



Dataflow Concepts

Batch & Stream Processing

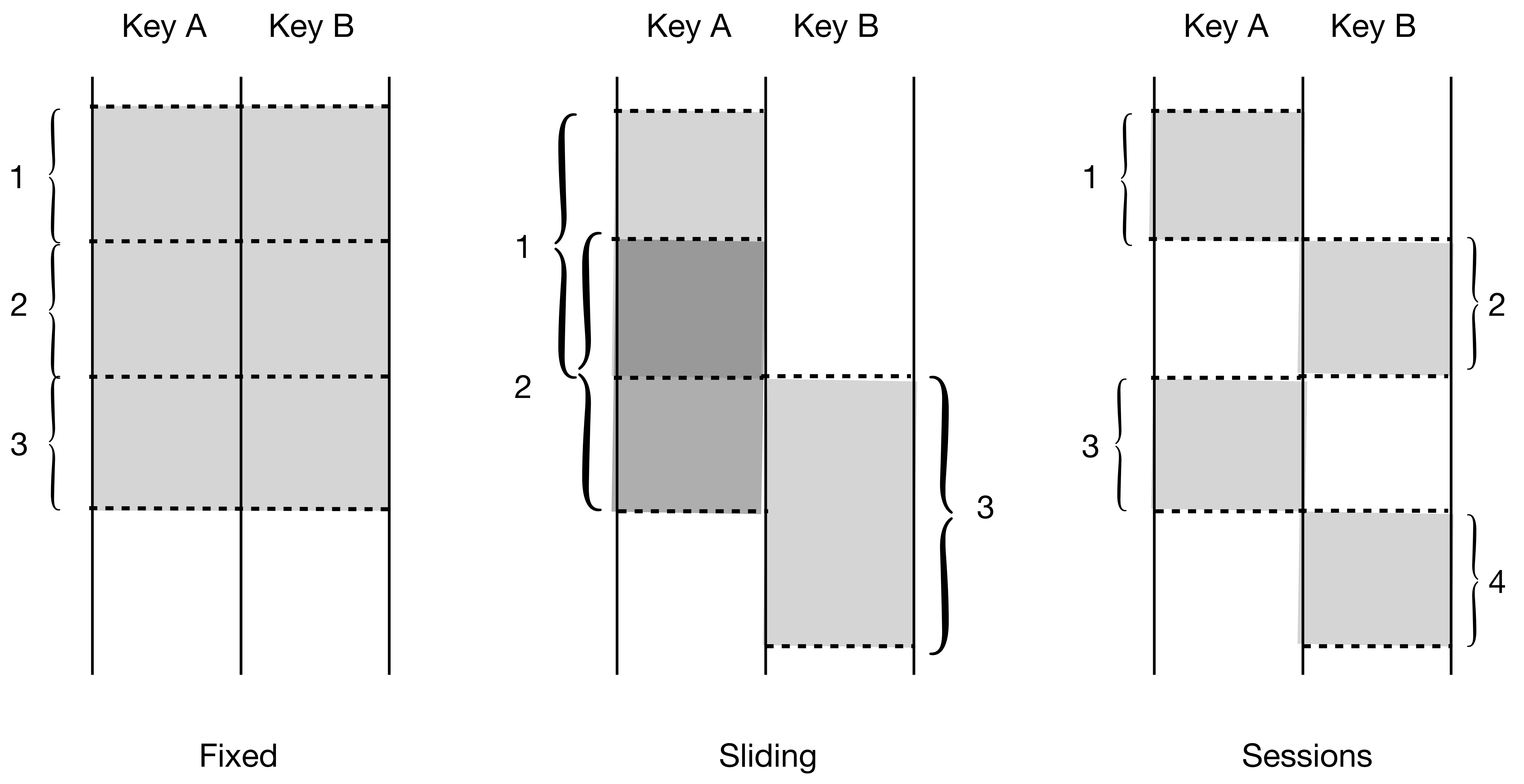
- Data objects as asynchronous events
  - Stream processing: each data object processed independently (unbounded)
  - Batch processing: grouping of data objects (bounded)



Dataflow Concepts

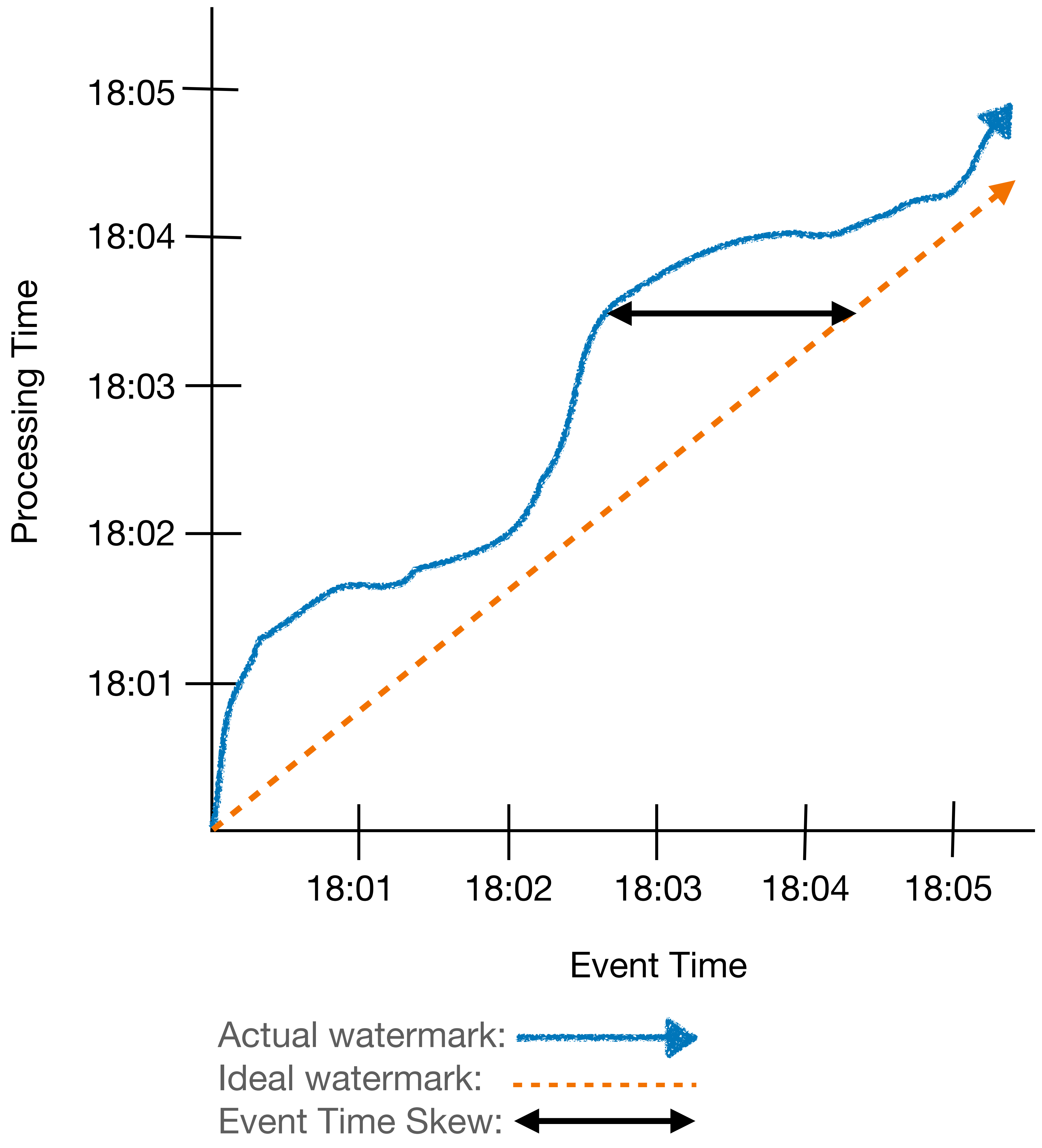
Windowing

- Slicing data sets for processing as a group (aggregation)
- One data item can be assign to more than one group
- Directing data to specific consumers



Dataflow Concepts
Timing

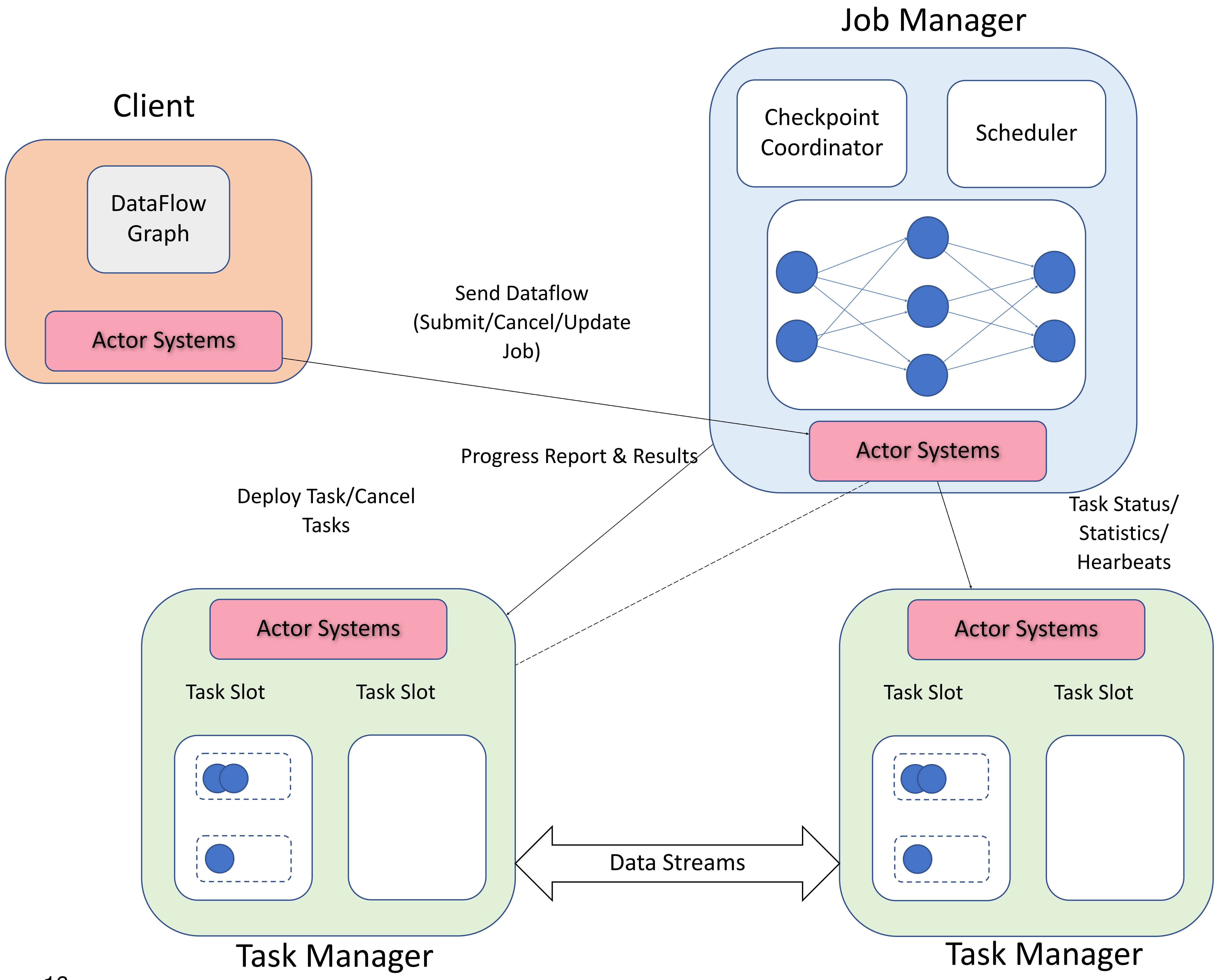
- Elastic data processing
- Asynchronous sourcing
- Unpredictable transport and processing delays
- Ideally: processing matches production rate
- Task of a Dataflow system: adjust processing graph to production rate and "real-time requirements"



Dataflow

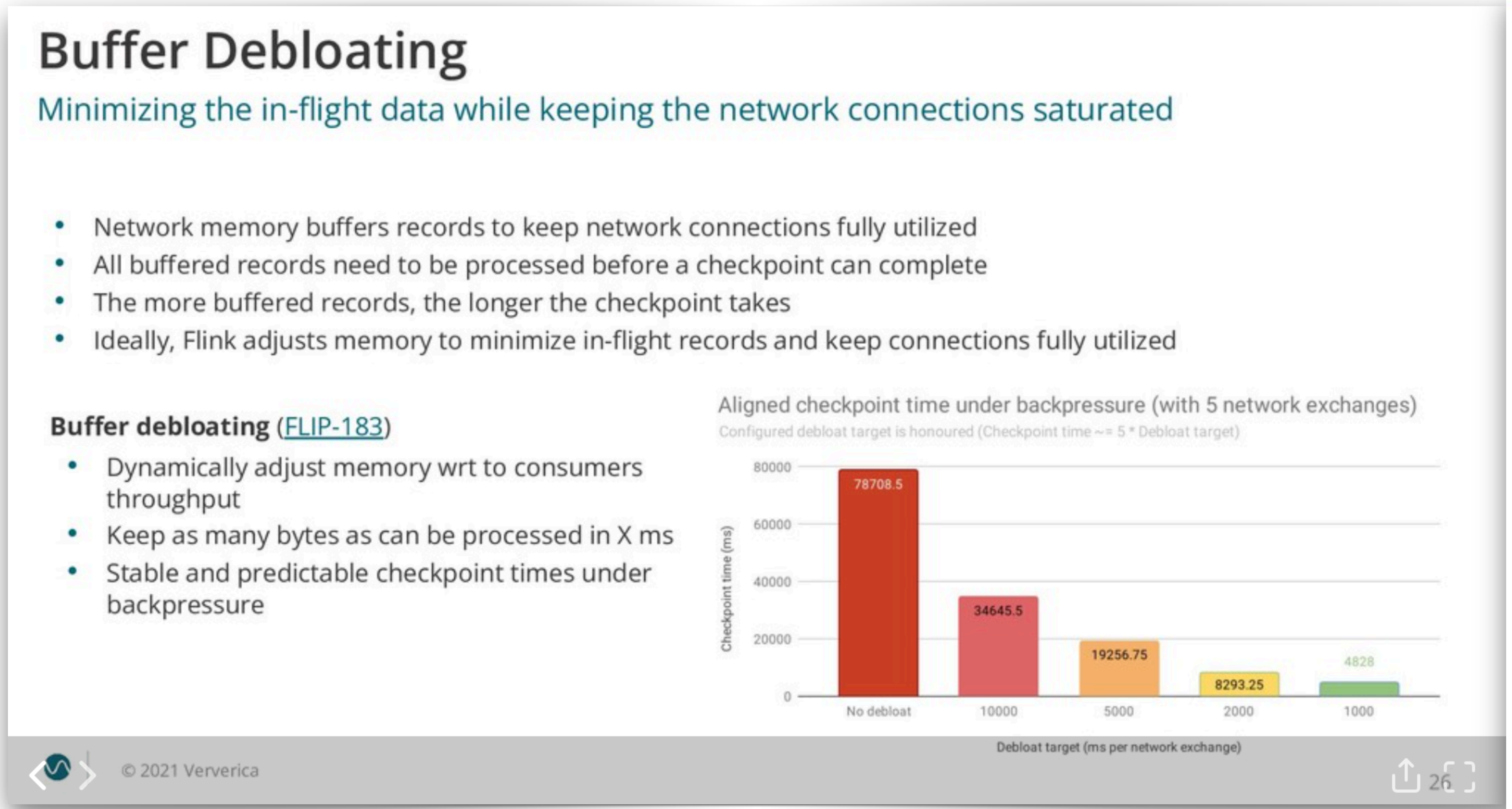
Mainstream Implementations

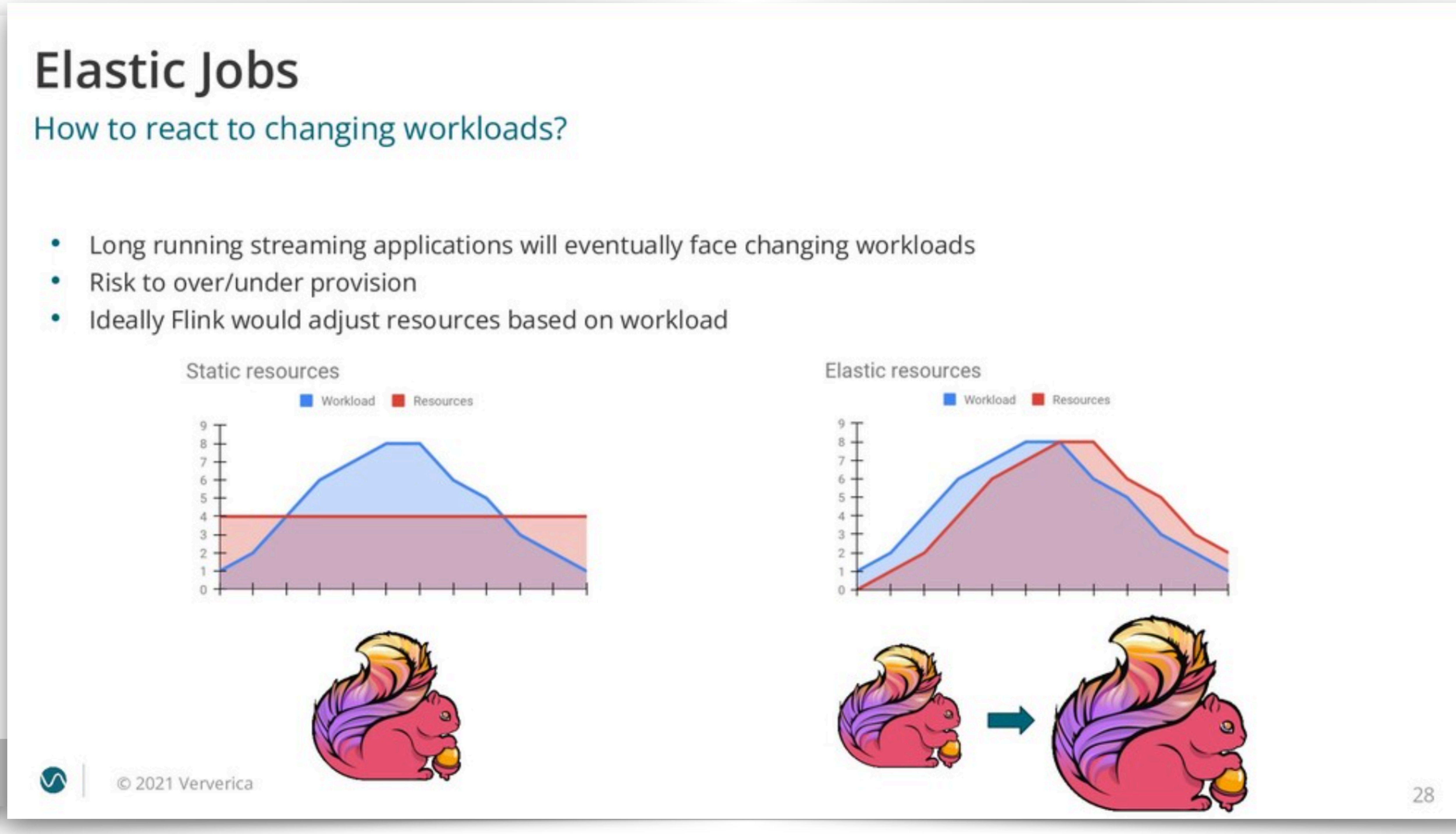
- **Apache BEAM**
  - Unified programming model for data processing pipelines
- **Dataflow runners**
  - Execution environments for Dataflow applications
  - Apache Flink, Samza, Spark
  - Google Cloud Dataflow



Recent Additions to Flink

Announced at Flink Forward 2021

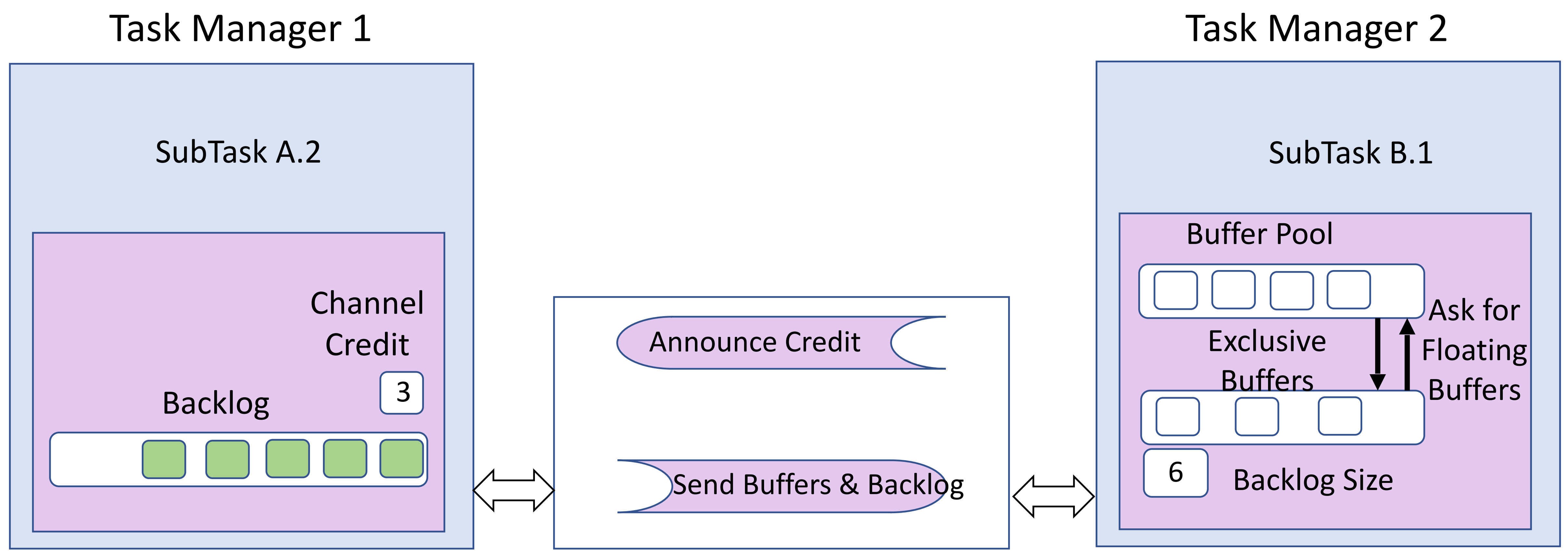




Dataflow

Transport and Back Pressure

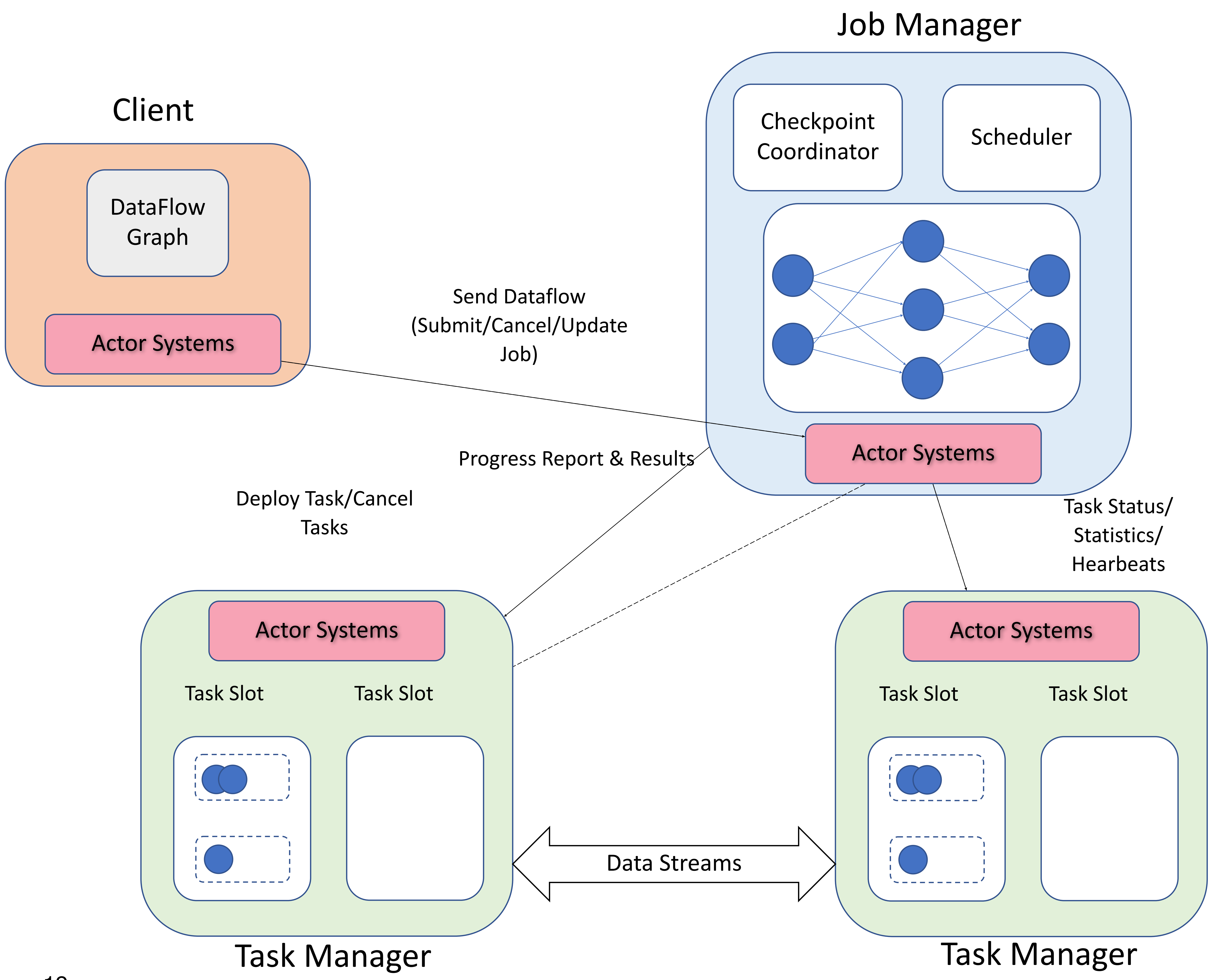
- Example: Apache Flink
- Connections connect task managers, not tasks
- Need to regulate upstream processing rates



Problem Statement

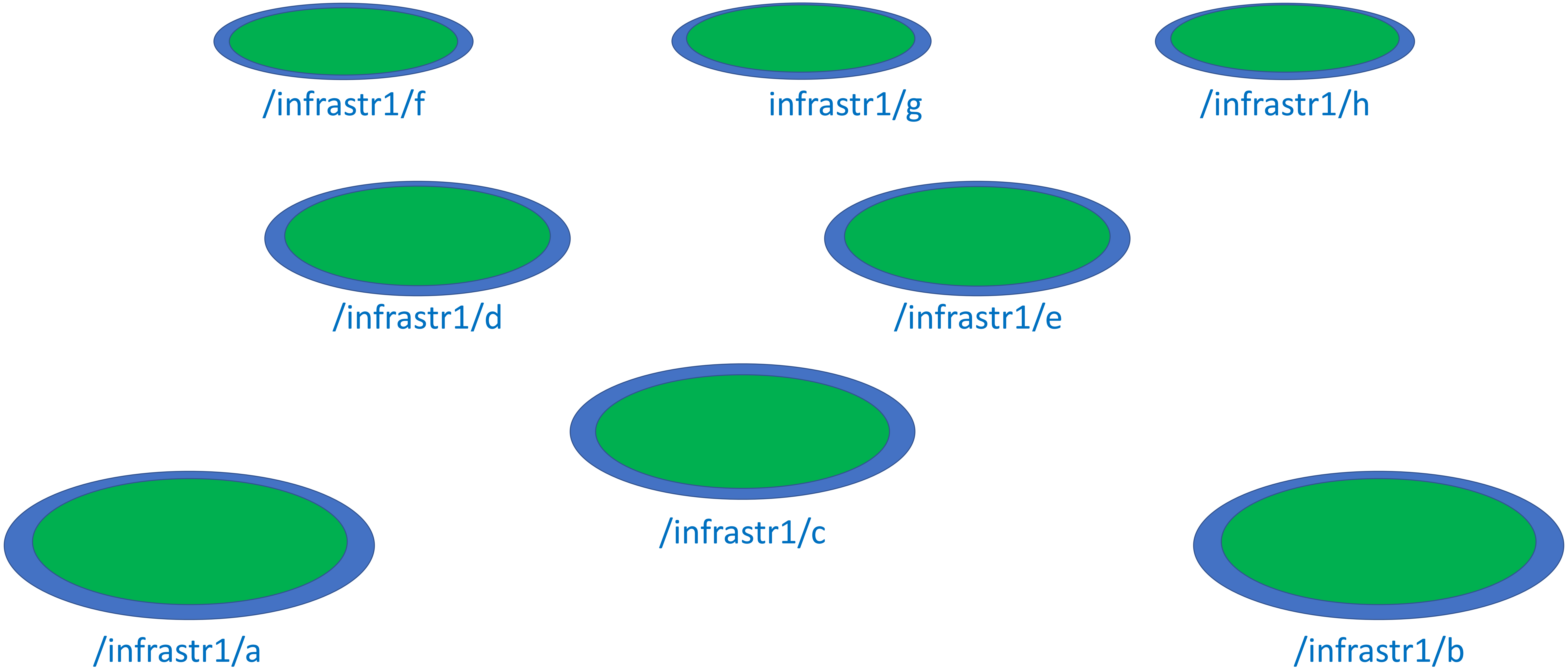
Overlays, Pipes, Address Mappings, Orchestration

- **Overlays do not match the inherent logic of processing immutable data objects**
  - Data is locked into connections
  - Connections are virtual channels between IP hosts
  - Orchestrator required to track resources, maintain mappings of task relationships to connections between hosts
- **Elastic Dataflow requires agile function instantiation, flow graph updates etc.**
- **Performance is a function of upstream data rates, network throughput, processing speed**
  - Limited visibility into root causes of performance problems at orchestrator



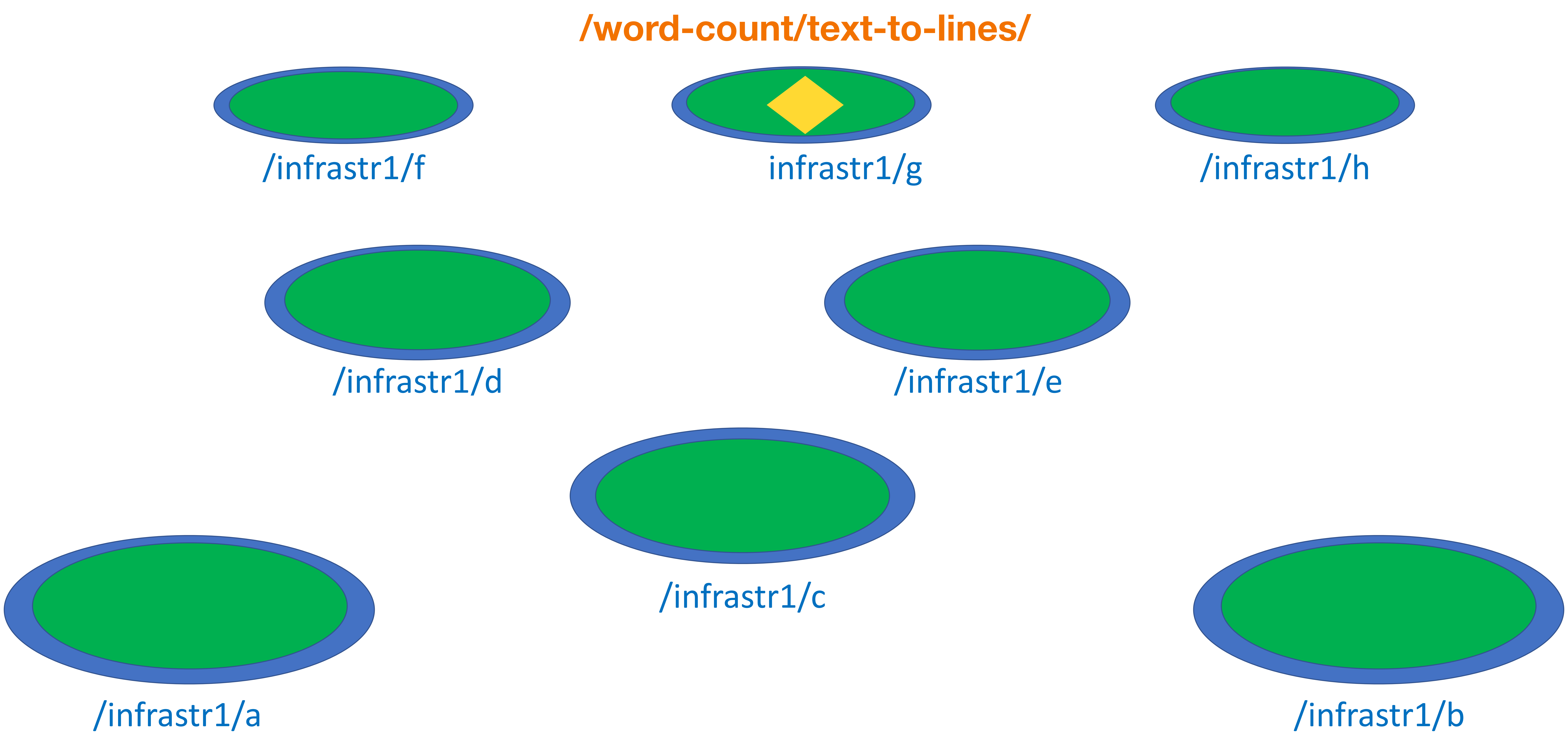
IceFlow

Information-Centric Dataflow



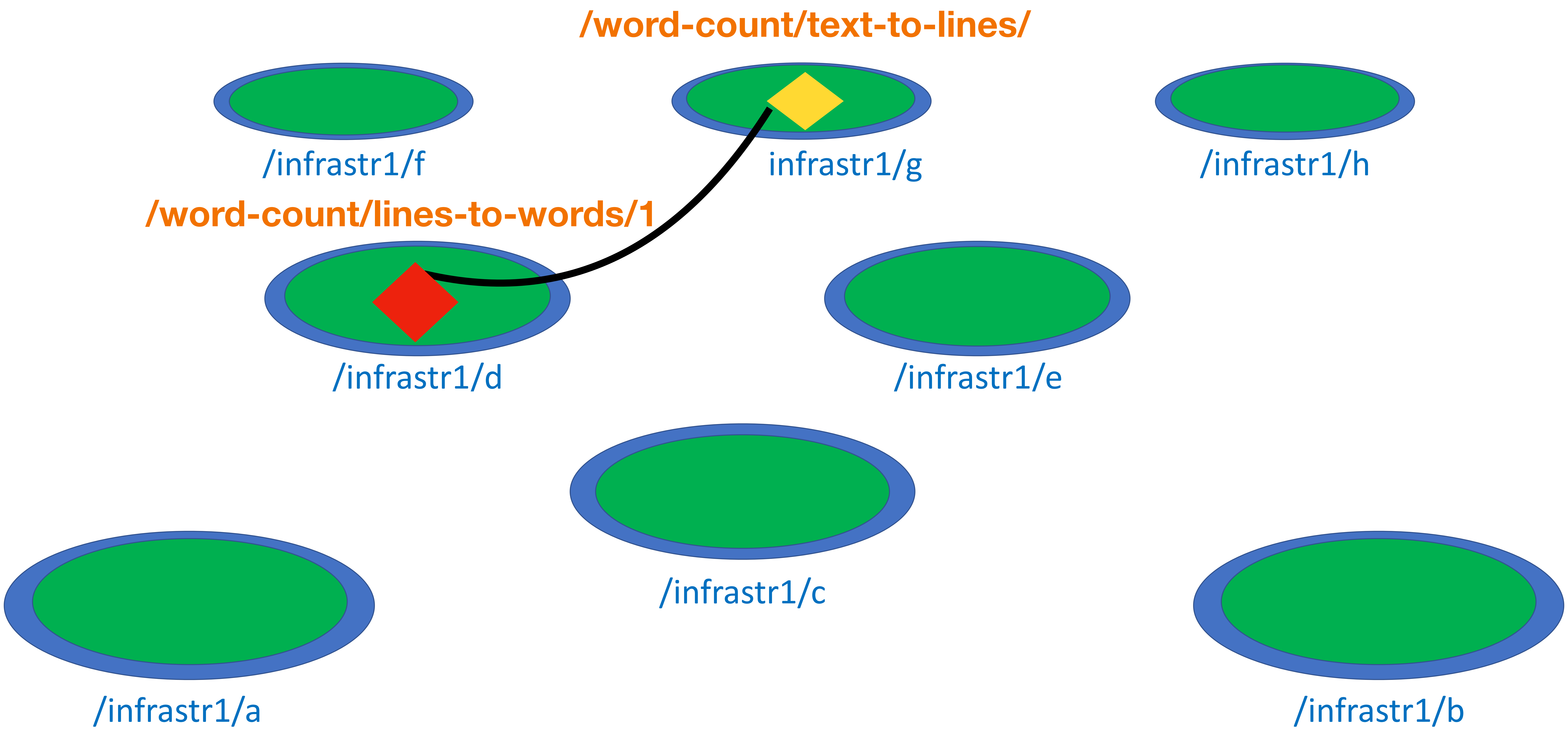
IceFlow

Information-Centric Dataflow



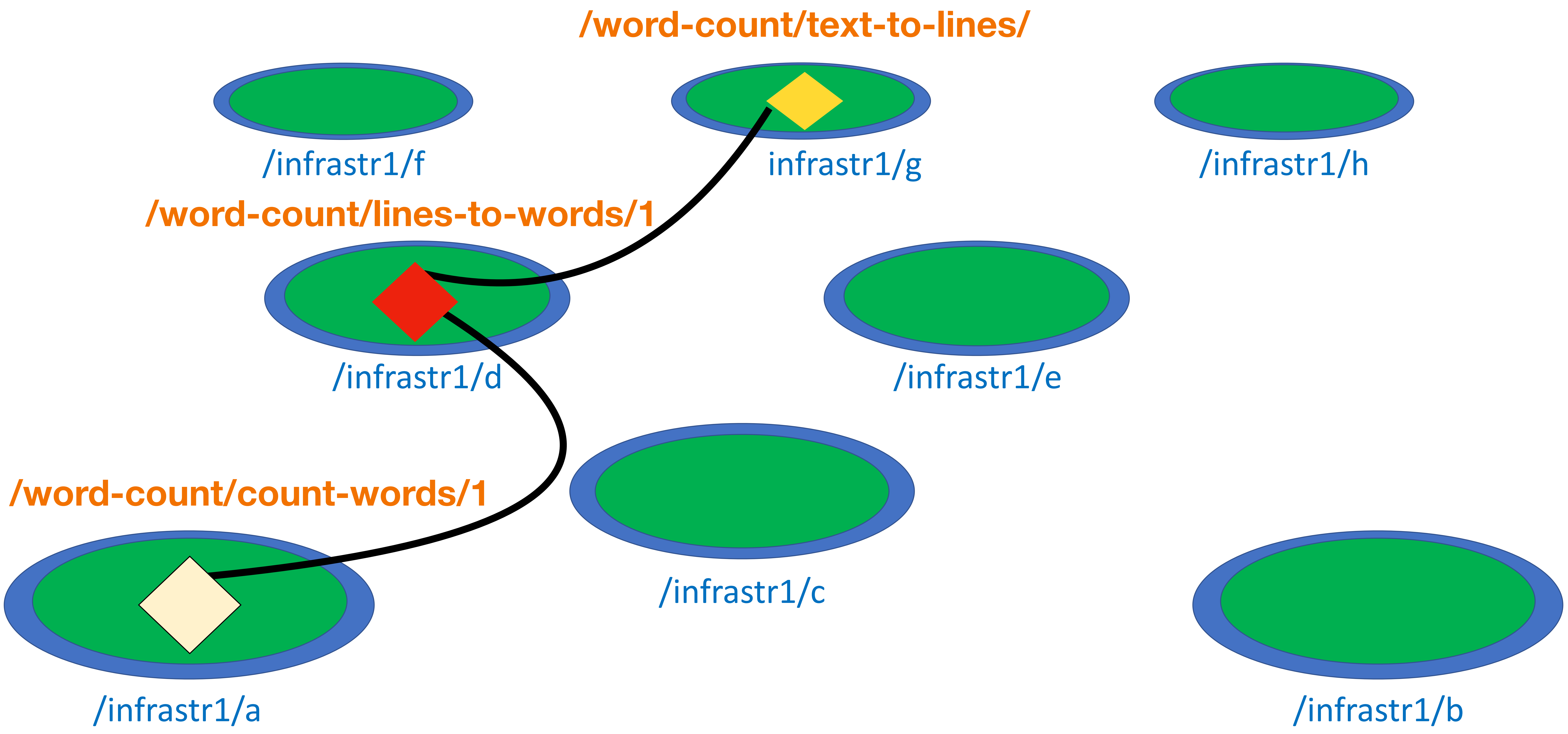
IceFlow

Information-Centric Dataflow



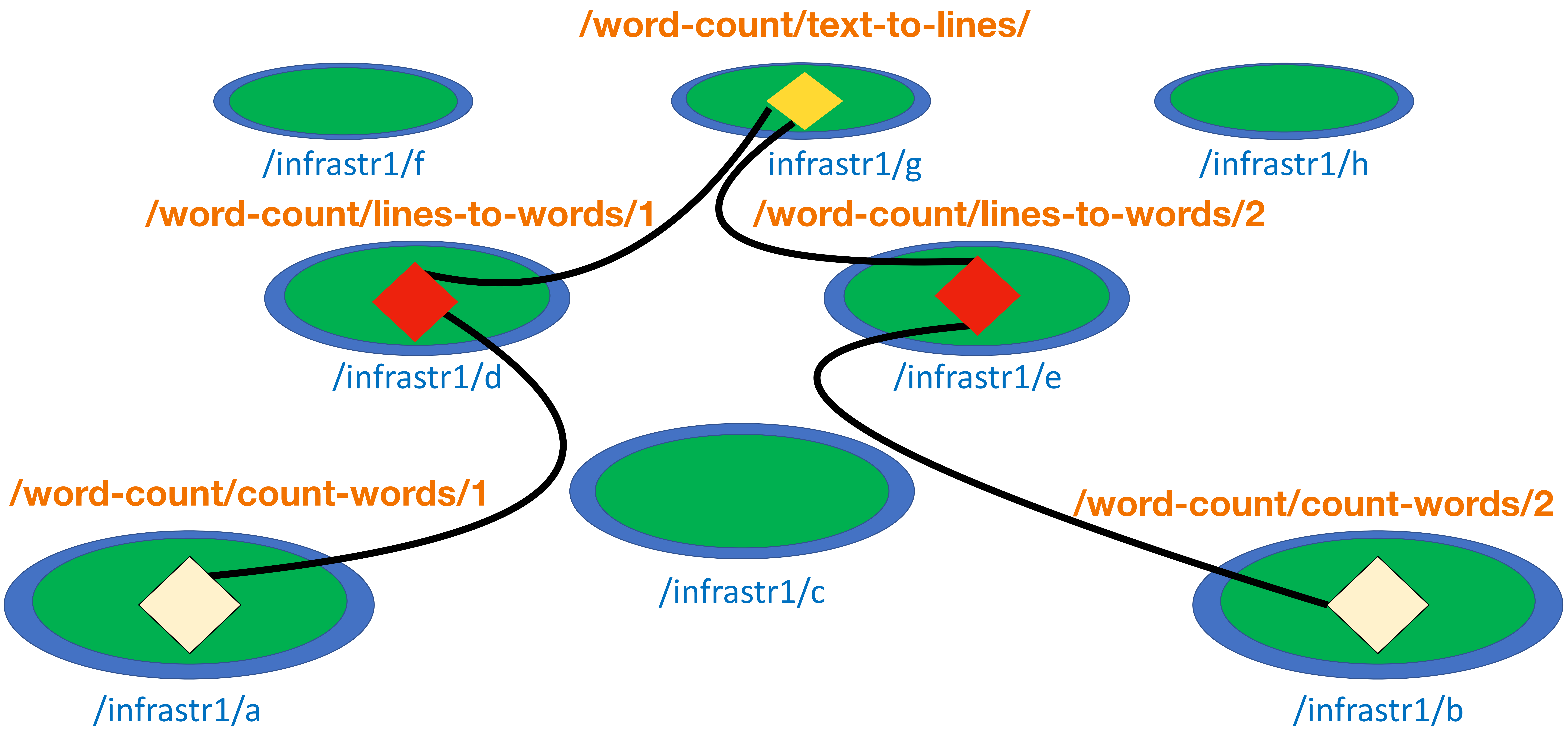
IceFlow

Information-Centric Dataflow



IceFlow

Information-Centric Dataflow



IceFlow

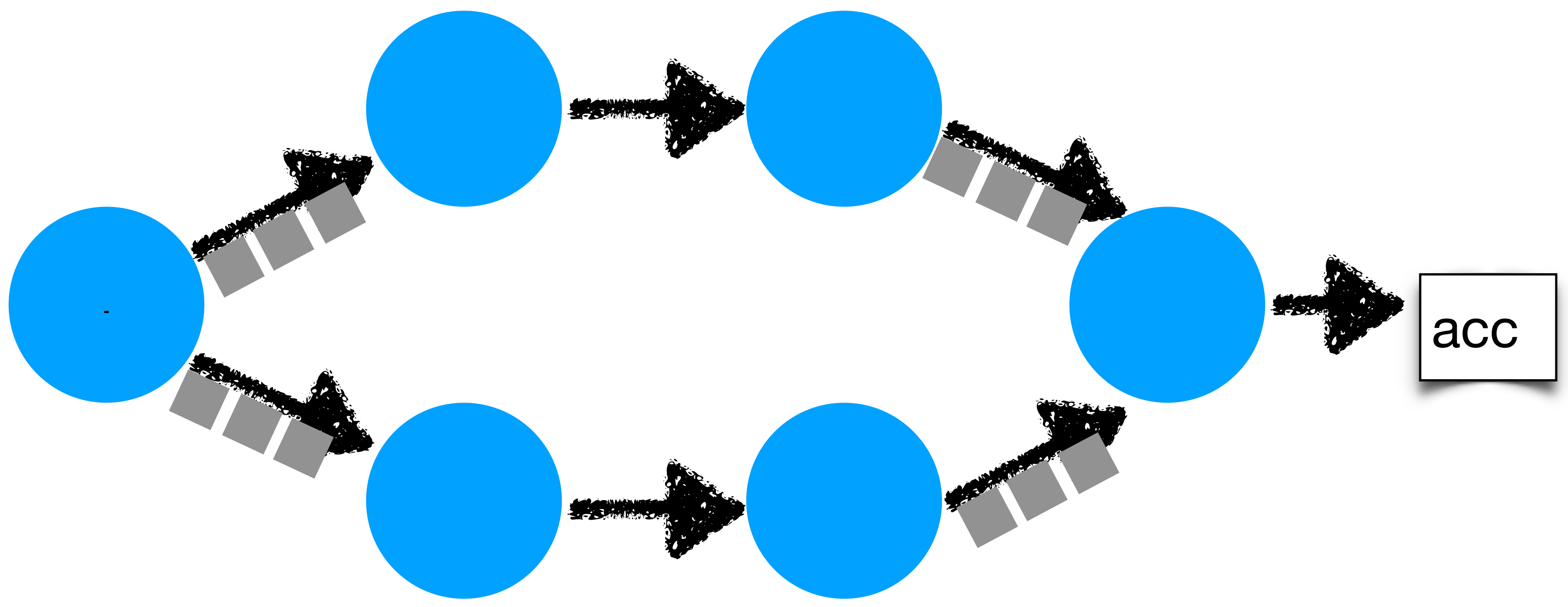
Concepts

- **Just Names**
  - For infrastructure
  - And for actors
- **Computation results as Named Data Objects**
  - Usual ICN properties...
- **Asynchronous data production**
  - Consumer has to know when data is available
- **Flow control**
  - Some coupling between consumers and producers
- **Garbage collection**
  - producers may be resource-constrained
  - cannot keep data forever

```
/[app]/[actor]/[instance]/data/[partition]/[object]
```

app	the name of the application
actor	the name of a Dataflow actor
instance	actor instance number
partition	monotonically increasing partition number to structure data objects on the producer's side
object	monotonically increasing sequence number

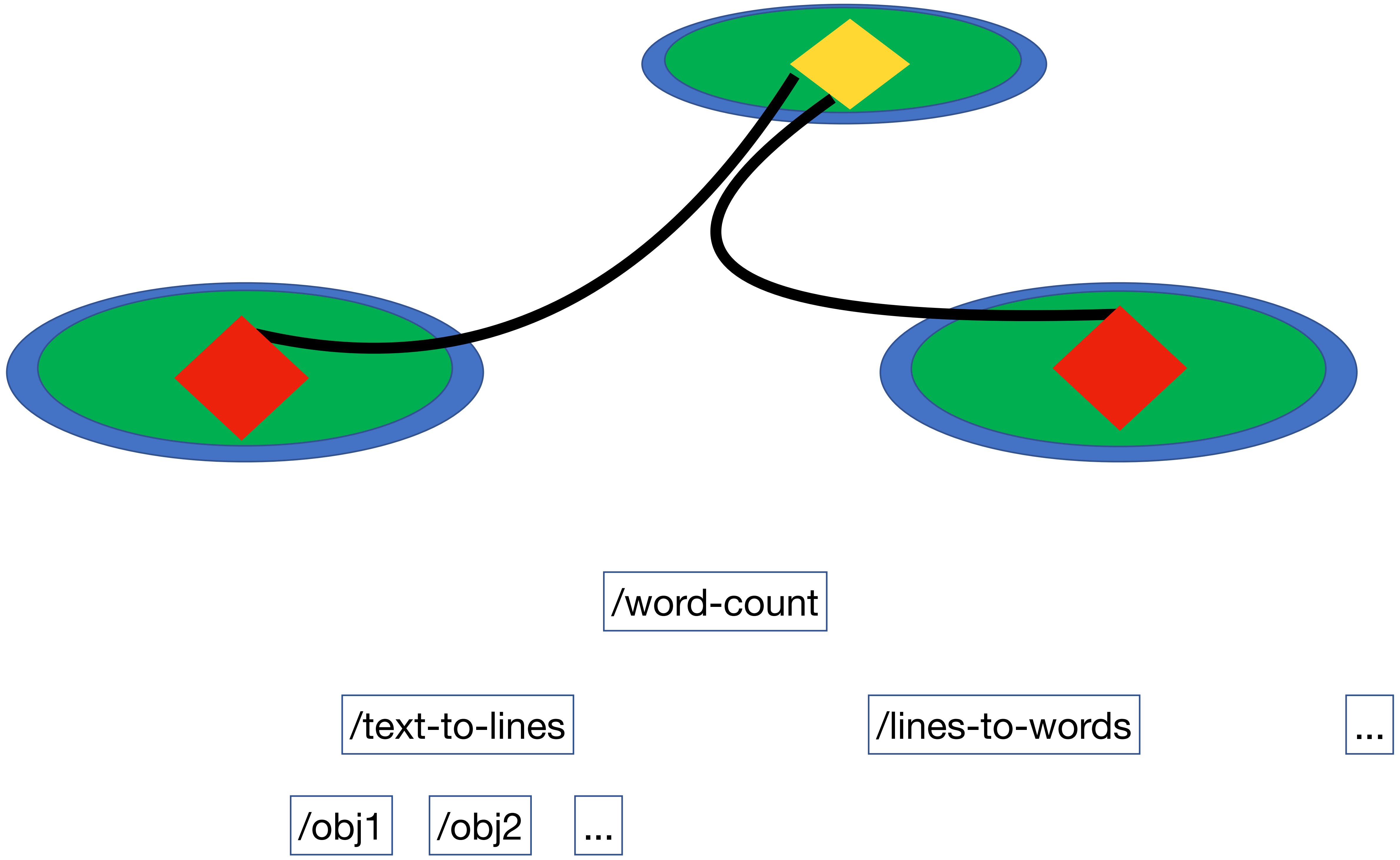
```
/word-count/text-to-lines/1/data/1/1
/word-count/lines-to-words/2/data/3/27
```



IceFlow Operation

Dataset Synchronization

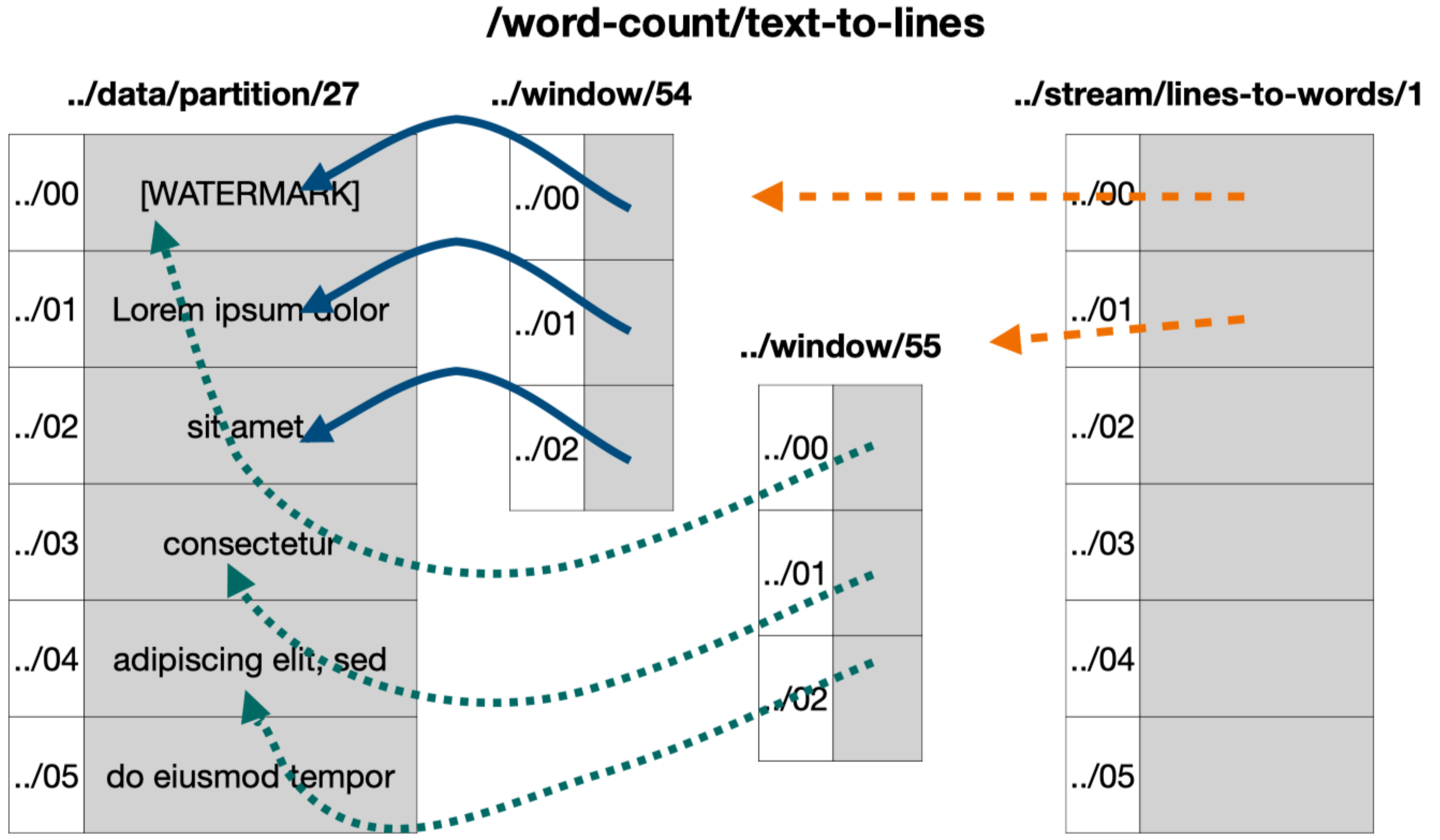
- Producers produce data under a known prefix
  - Consumers subscribe to prefix
  - And learn update new input data
- Ideally: one prefix for whole application ("word-count")
  - Everyone could learn about all data in the app context
  - For practical reasons: need indirection
  - One prefix per consumer group



IceFlow

Windows and Result Sharing

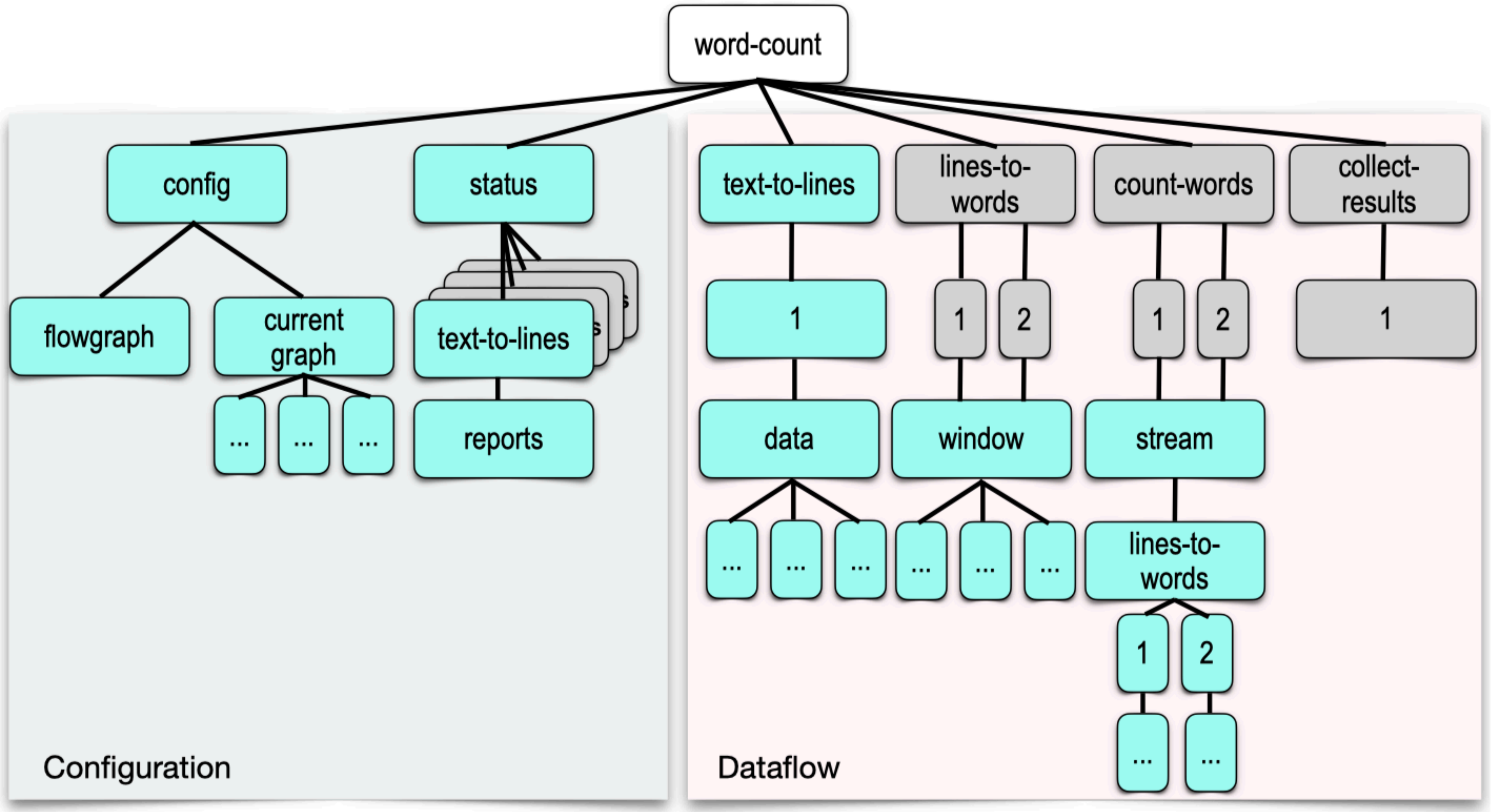
- Need more flexibility to re-use computation results in different contexts
    - Group data objects in windows
    - Group windows under per-consumer name prefixes



IceFlow

Dataflow data and configuration

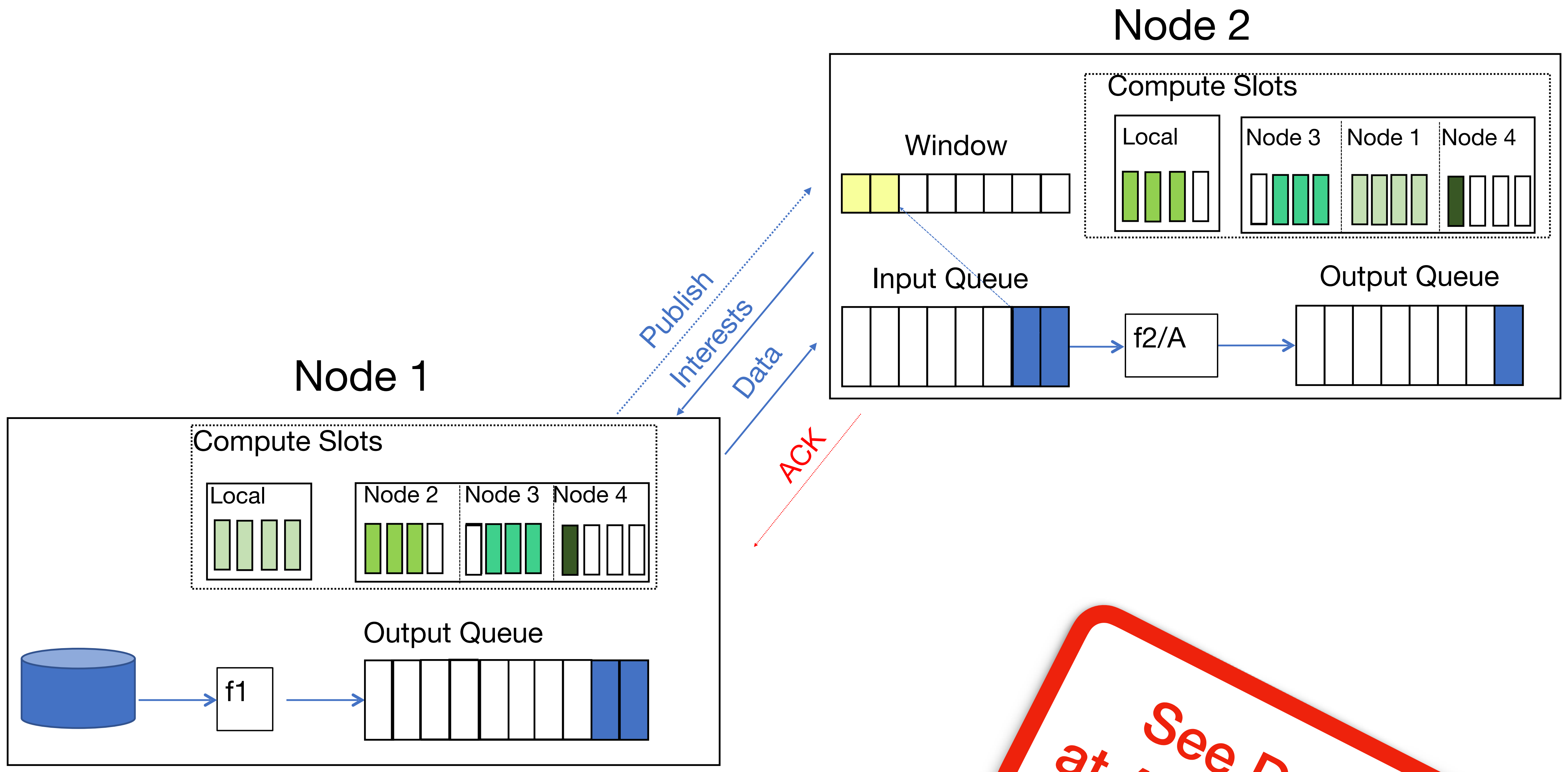
- Need additional shared information
  - Static application flowgraph
  - Actual current dynamic flowgraph
- Also: loose coupling between consumers and producers
  - Consumers reports: what windows have been processed
  - So that producer can advance
- Result: share namespace with Dataflow data and configuration info
  - Some config info represented in CRDTs (like in CFN)



IceFlow

Resource Management

- IceFlow can be smarter than receiver-driven AIMD
    - No need to fetch data that cannot be processed at throughput speed
    - "Receive Window"
- Producers should not overrun consumers
    - Output queue occupancy...
    - When consistently full: trigger scale-out



IceFlow

Insights So Far

- **Todays Dataflow systems are powering many data science applications**
- Overlay approach
  - Usual address mapping and virtual circuit issues
  - Limited data sharing
  - Centralized orchestration
- **Real opportunity for redesigning distributed data processing with ICN**
  - Elegant name-based approach: no mappings, no resolution – just data
  - Direct sharing of computation results
  - Potentially better visibility into network performance
- **Dataset synchronization in principle the right approach**
  - NDN Psync performance not great in experiments (NFD)
  - Also requires multicast forwarding strategy
- **Additional mechanisms needed**
  - Name-based routing (NLSR should be fine)
  - Failure recovery
- **Take-aways for COIN**
  - IceFlow an example for new protocol work
  - Breaking up overlays
  - Here: Dataflow – other interaction classes next?

PICCOLO

piccolo-project.org

This project receives funding from the German Federal Ministry for Ecomomic Affairs and Energy (BMWi) within the "Development of Digital Technologies" framework programme and is managed by the "Digital Technologies and Applications" project agency of the German Aerospace Center (DLR) in Bonn, Germany.

Federal Ministry for Economic Affairs and Energy

DLR Deutsches Zentrum für Luft- und Raumfahrt German Aerospace Center