Stream processing

Anne-Marie Kermarrec

EPFL

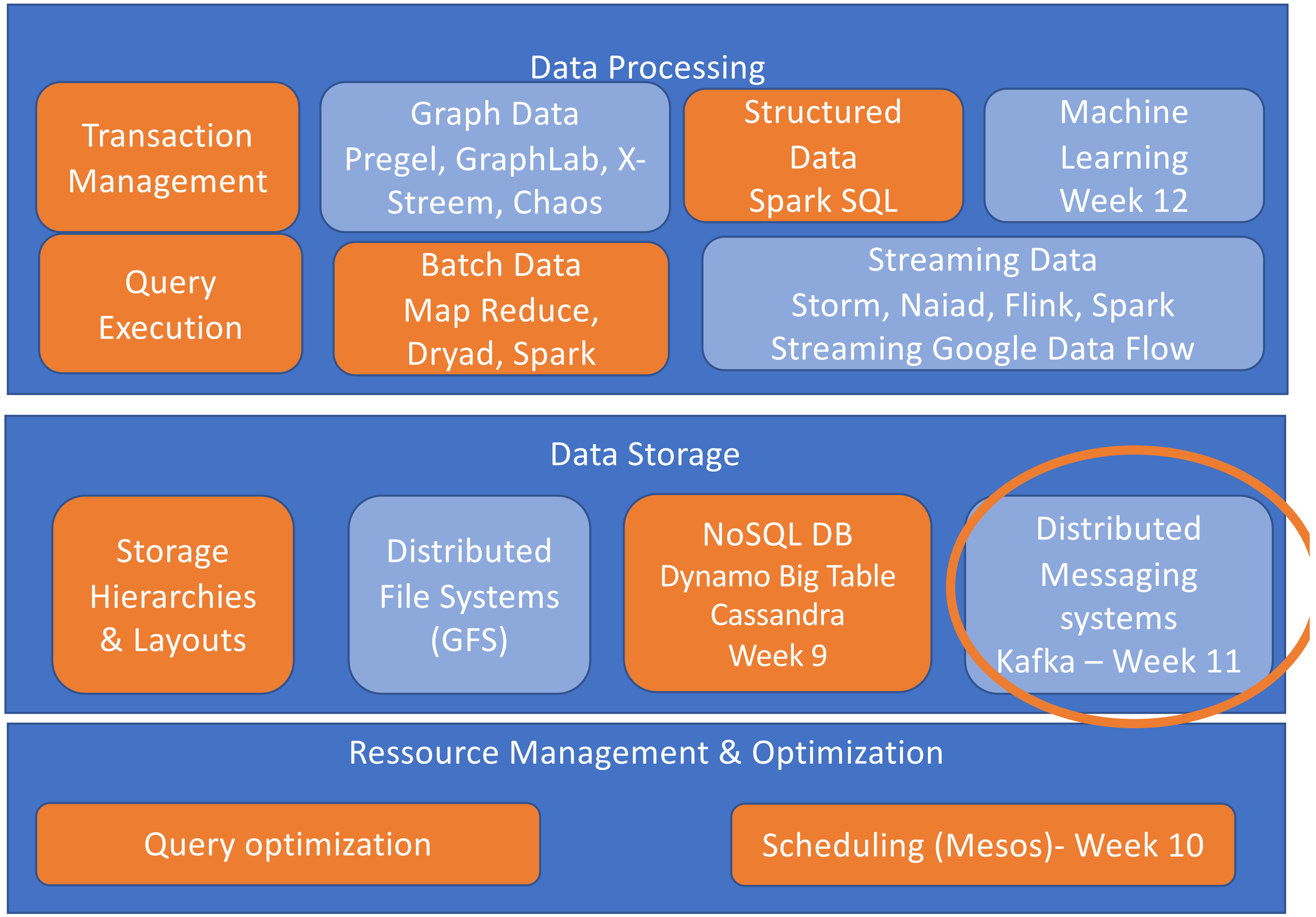
Where are we?

Gossip Protocols
Week 7

Consistency protocols
CAP Theorem
Week 9

Distributed/decentralized systems
Week 8-12

Data science software stack


Data Processing
Transaction Management
Graph Data Pregel, GraphLab, X-Streem, Chaos
Structured Data Spark SQL
Machine Learning Week 12
Query Execution
Batch Data Map Reduce, Dryad, Spark
Streaming Data Storm, Naiad, Flink, Spark Streaming Google Data Flow
Data Storage
Storage Hierarchies & Layouts
Distributed File Systems (GFS)
NoSQL DB Dynamo Big Table Cassandra Week 9
Distributed Messaging systems Kafka – Week 11
Ressource Management & Optimization
Query optimization
Scheduling (Mesos)- Week 10

Stream Processing

- Stream processing continuously incorporates new data to compute a result.
- The input data is unbounded.
  - A series of events, no predetermined beginning or end.
- User applications can compute various queries over this stream of events.

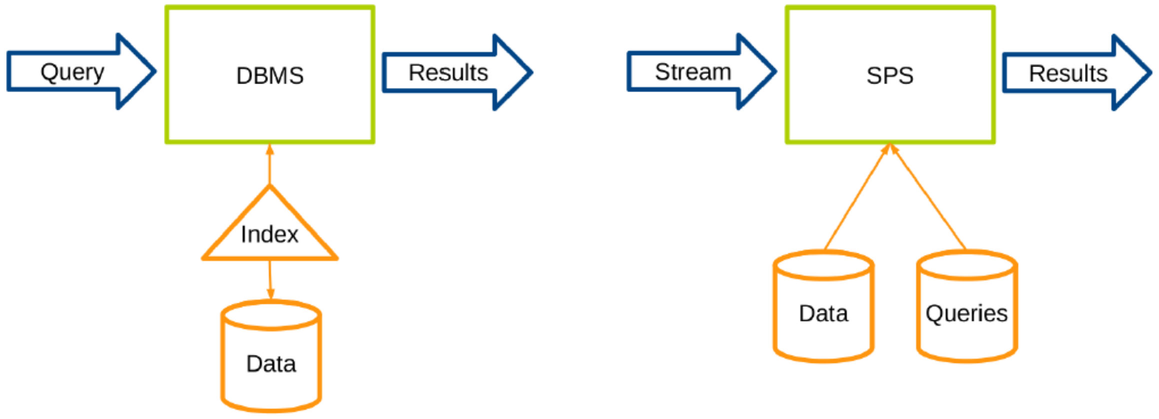
Use cases

- Fraud detection systems
- Trading systems (examine price changes and execute trades)
- Military and intelligence systems
- Advertizement systems and recommenders
  - Data analytics
  - Recommenders
  - …

- Real-time analytics
  - Rate of certain events e.g. tracking a running count of each type of event, or aggregating them into hourly windows
  - Compute rolling average of a value
  - Compare current statistics

Stream Processing versus DBMS

- Database Management Systems (DBMS): data-at-rest analytics
  - Store and index data before processing it.
  - Process data only when explicitly asked by the users
- Stream Processing Systems (SPS): data-in-motion analytics
  - Processing information as it flows, without (or with) storing them persistently.



EPFL

Stream processing versus batch processing

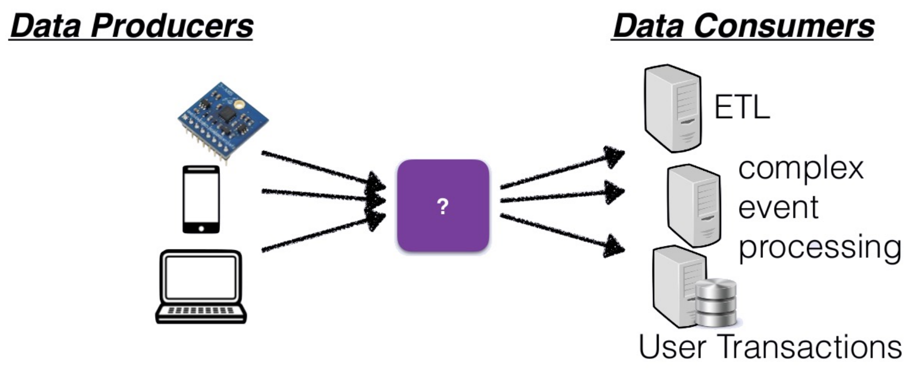
- Advantages of stream processing
  - Near real-time results
  - Do not need to accumulate data for processing
  - Streaming operators typically require less memory
- Disadvantages of stream processing
  - Some operators are harder to implement with streaming
  - Stream algorithms are often approximations

Example

- Recommender system
  - Every time someone loads a page; a *viewed page event* generates several events.
- That may lead to any of the following:
  - Store the message in Cassandra/MongoDB for future analysis
  - Count page views and update a dashboard
  - Trigger an alert if a page view fails
  - Send an email notification to another user
  - Compute analytics
  - Compute recommendations

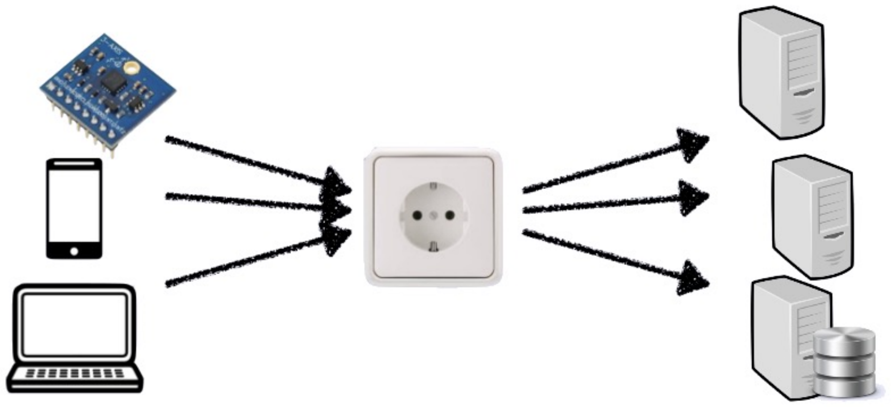
Messaging System

- Disseminate streams of events from various producers to various consumers.
- Messaging system is an approach to notify consumers about new events.
- Messaging systems
  - Direct messaging
  - Pub/Sub systems



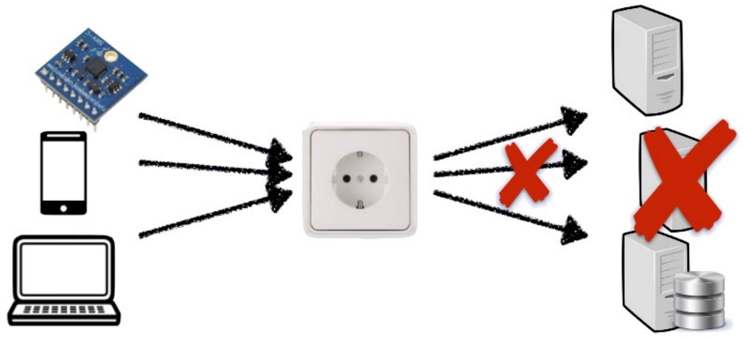
Direct Messaging

- Necessary in latency critical applications (e.g., remote surgery).
- A producer sends a message containing the event, which is pushed to consumers.
- Both consumers and producers have to be online at the same time.

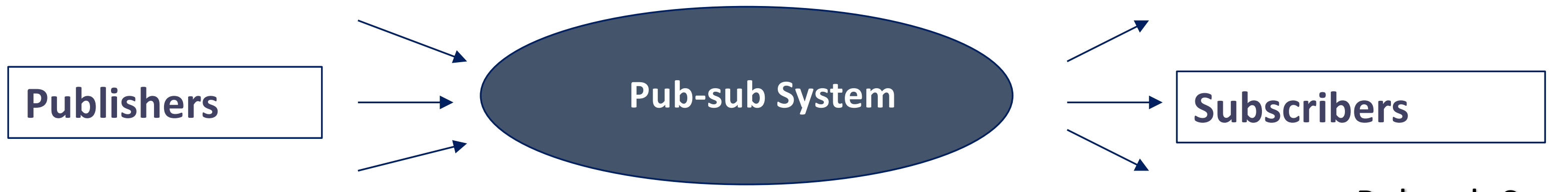


Direct Messaging

- Issues when consumer crashes or temporarily goes offline.
- Producers may send messages faster than the consumers can process.
  - Dropping messages
  - Backpressure
- Message brokers can log events to process at a later time.

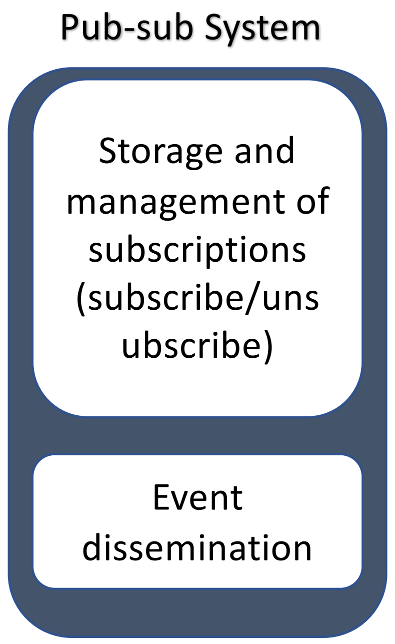


Publish-subscribe systems

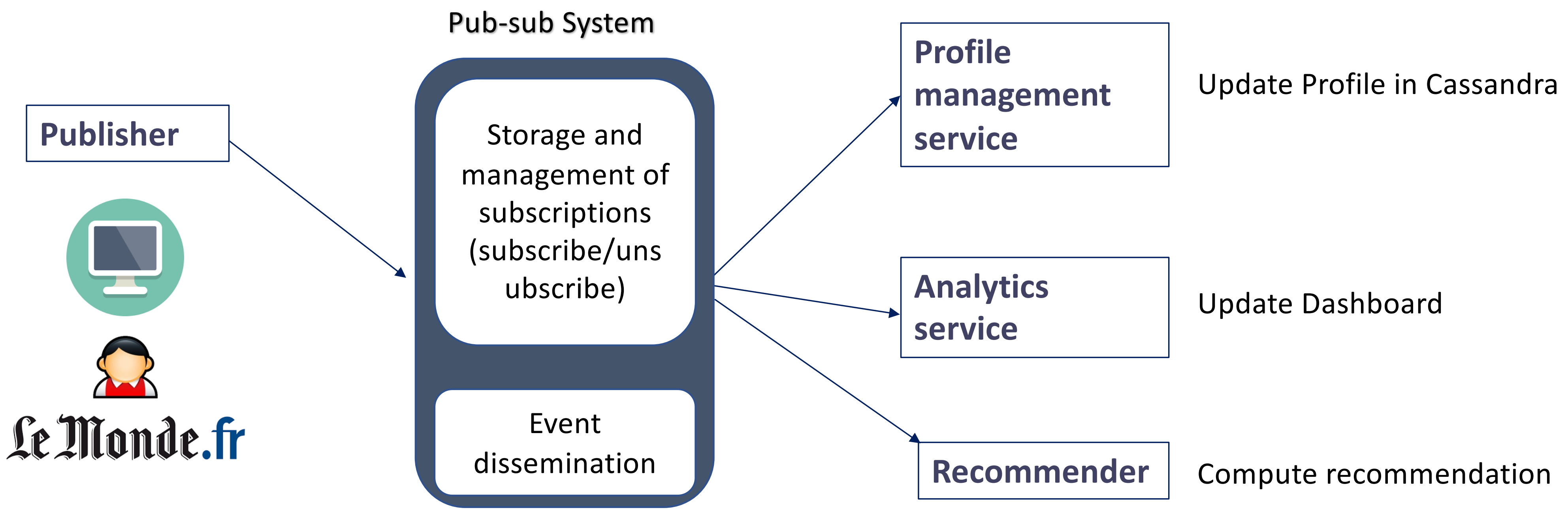


- Asynchronous (loosely coupled) event notification system
- A set of Subscribers/consumers register their interest (subscriptions)
- A set of Publishers/producers issue some events (events)
- Publish-subscribe system
  1. Manages users subscriptions
  2. Matches published events against subscriptions
  3. Disseminate events to matching subscribers (and no others)

- Flexible and seamless messaging substrate for applications


Pub-sub System
Storage and management of subscriptions (subscribe/unsubscribe)
Event dissemination

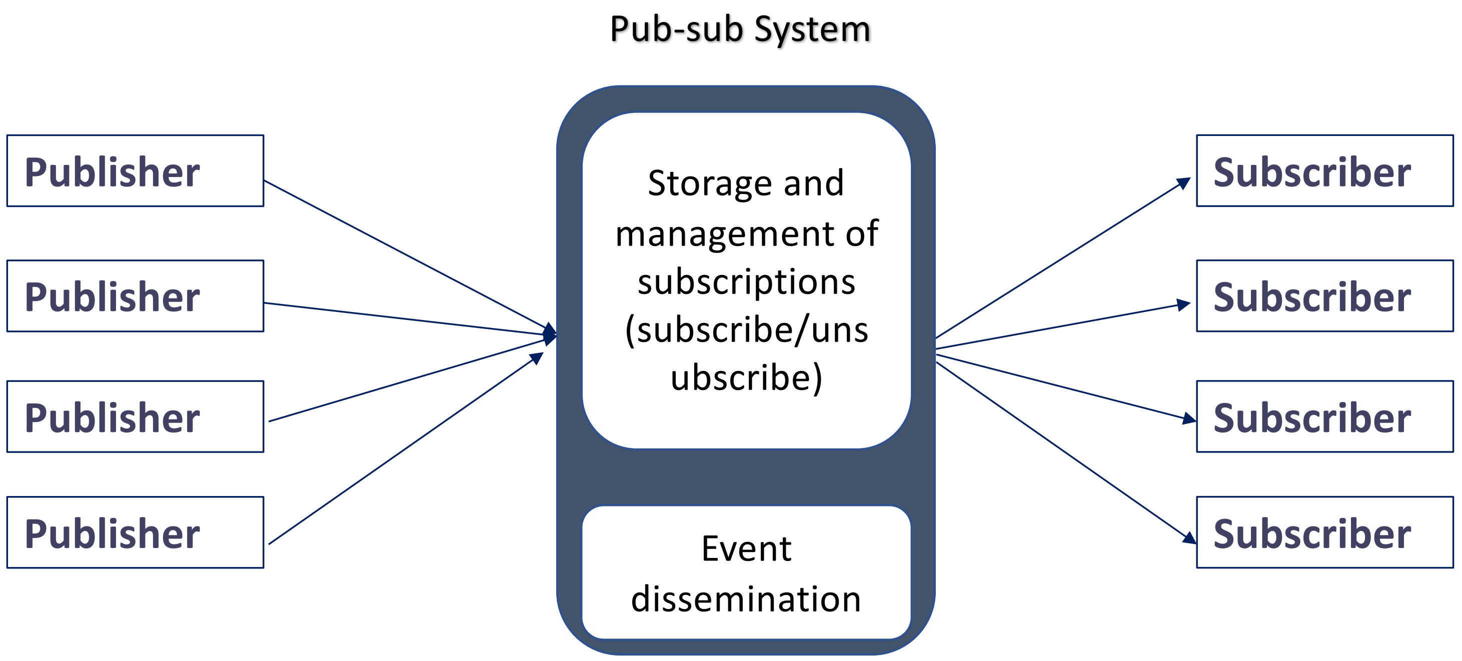
Example



Prominent way of disseminating information

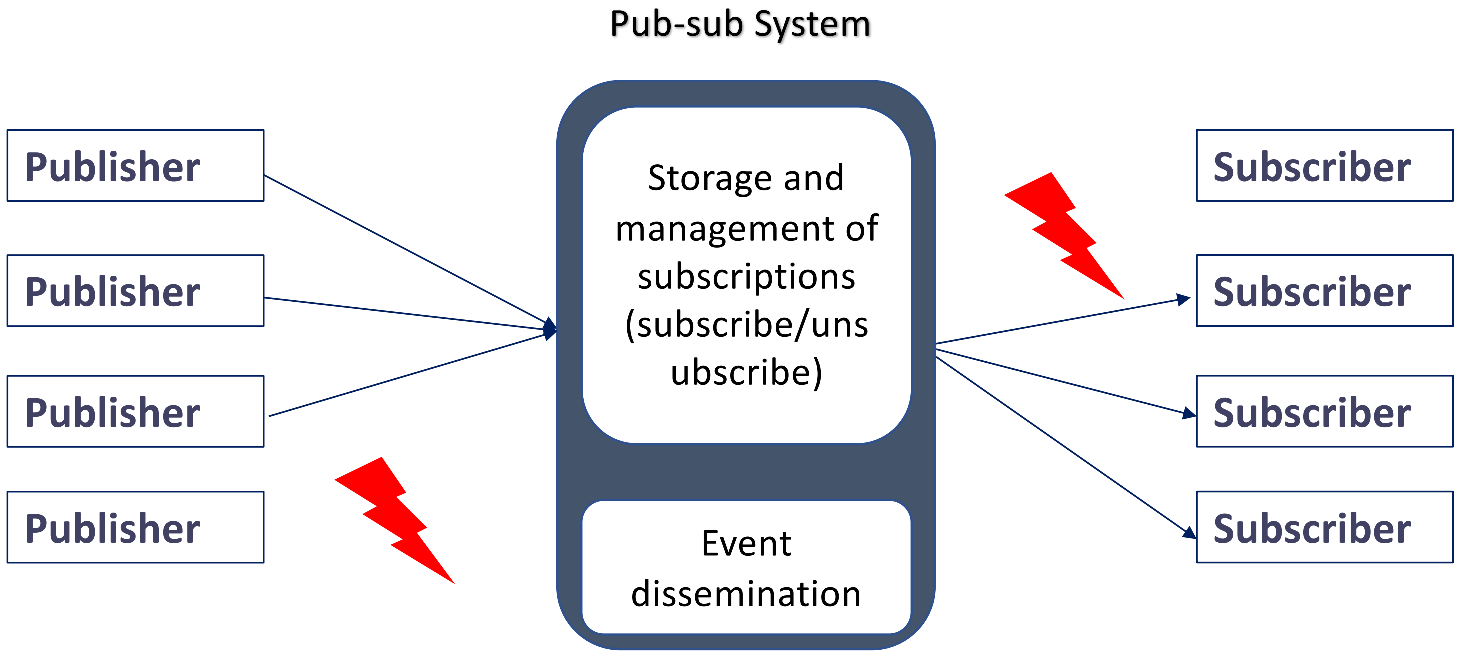
- Social networks
- RSS feeds
- Recommendation systems

Decoupling in time, **space** and synchronization

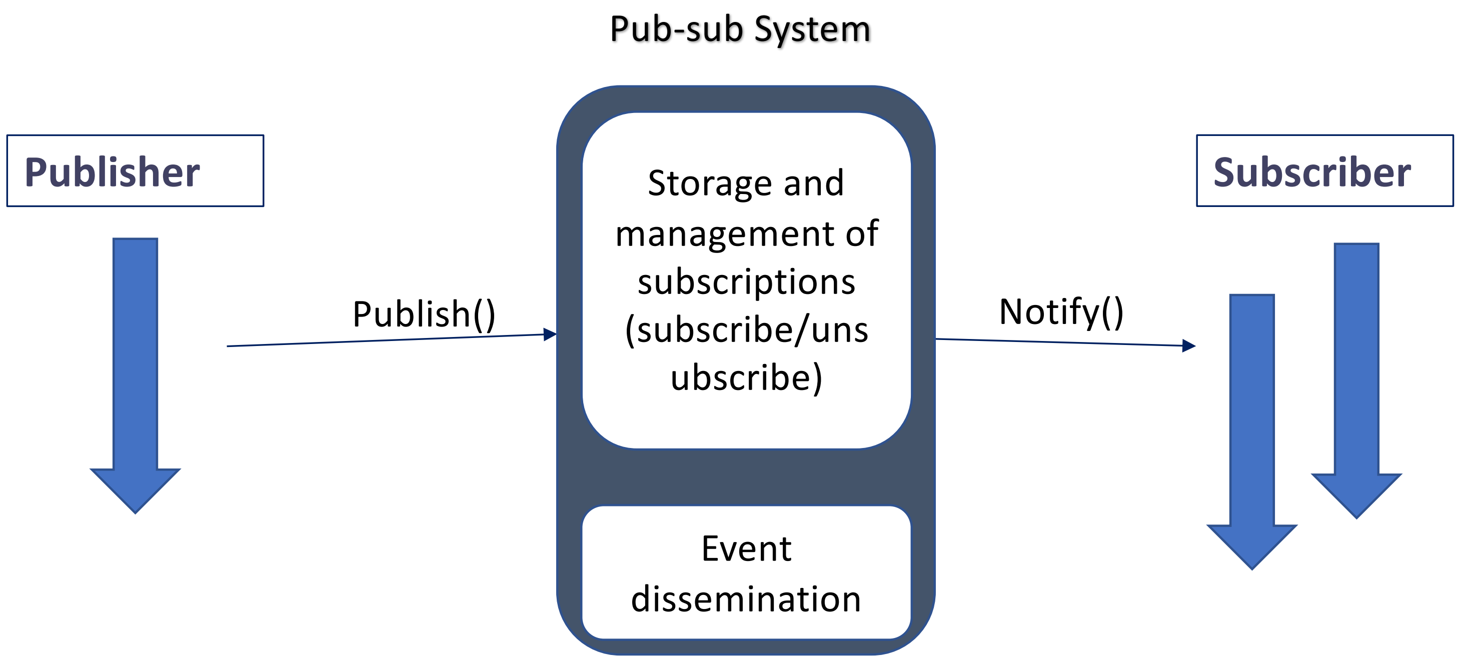


the interacting parties do not need to know each other.

Decoupling in **time**, space and synchronization



Decoupling in time, space and synchronization



Synchronization decoupling: producers are not blocked upon publication, subscribers are asynchronously notified

Pub-sub systems: expressiveness

- Differences in subscription expressiveness
- Topic-based ~ Application-level multicast

```
topic=houses_sales
```

- Content-based
  - Attribute-based

```
s1=(city=Rennes) (capacity=2_Bedrooms)
```

  - Range queries

```
s1=(city=Rennes || Saint Malo) &&
(capacity=3_Bedrooms || price < 500,000 EUR)
```

Pub/Sub architecture

- Centralized Broker model
  - Consists of multiple publishers and multiple subscribers and centralized broker
  - Subscribers/Publishers will contact 1 broker, and do not need to have knowledge about others.
  - E.g. CORBA event services, JMS, JEDI etc…

EPFL

Distributed/decentralized architectures

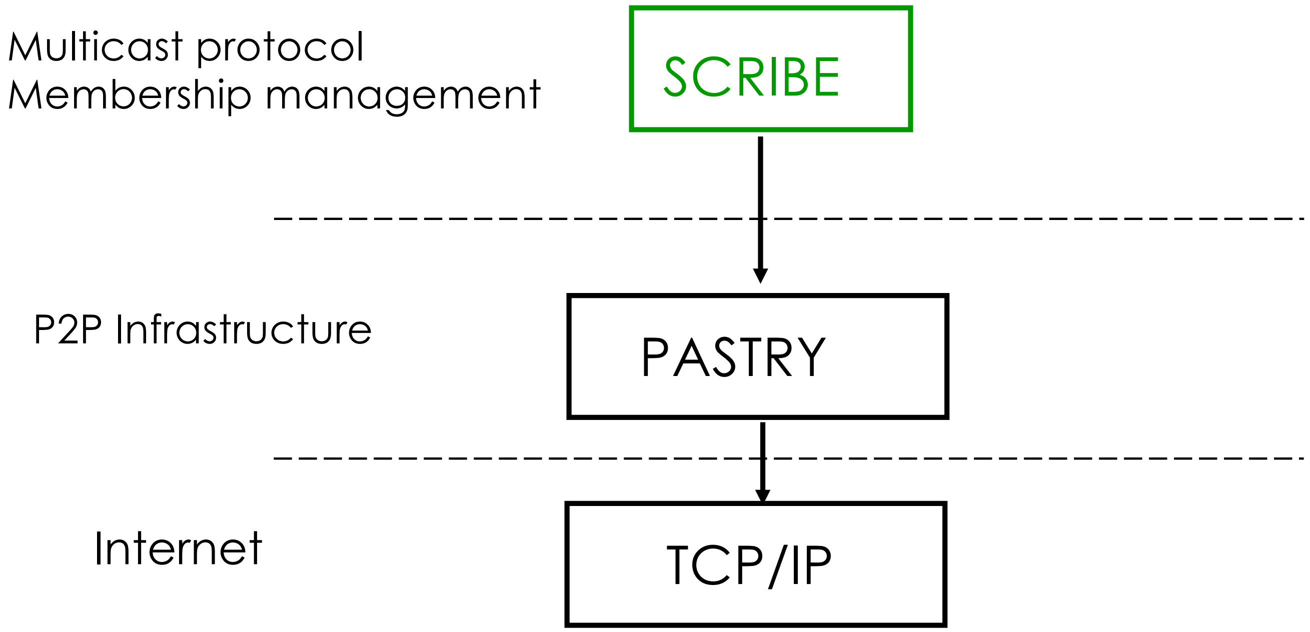
- Distributed model
  - A set of nodes act as brokers (Siena, Kafka)
- Decentralized model
  - Each node can be publisher, subscriber or broker.
  - DHT are employed to locate nodes in the network.
  - E.g. Java distributed event service/Tera

- Topic-based pub-sub: Equivalent/related to application-level multicast (ALM)

ALM on structured overlay networks

- Overlay network used for group naming and group localization
- Flooding-based multicast [CAN multicast]:
  - Creation of a specific network for each group
  - Message flooded along the overlay links
- Tree-based multicast [Bayeux, Scribe]
  - Creation of a tree per group
  - Flooding along the tree branches

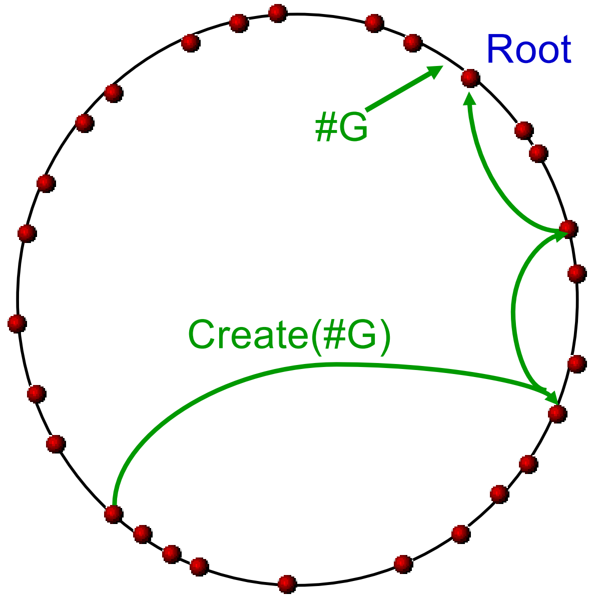
Scribe


Multicast protocol
Membership management
SCRIBE
P2P Infrastructure
PASTRY
Internet
TCP/IP

Scribe: design

- Goals
  - Group creation
  - Membership maintenance
  - Messages dissemination within a group
- Construction of a multicast tree on top of a Pastry-like infrastructure
  - Creation of a tree per group
  - The tree root is the peer hosting the key associated to that group
  - The tree is formed as the union of routes from every member to the root
  - *Reverse path forwarding*
  - Messages flooded along the tree branches

Scribe: group (topic) creation

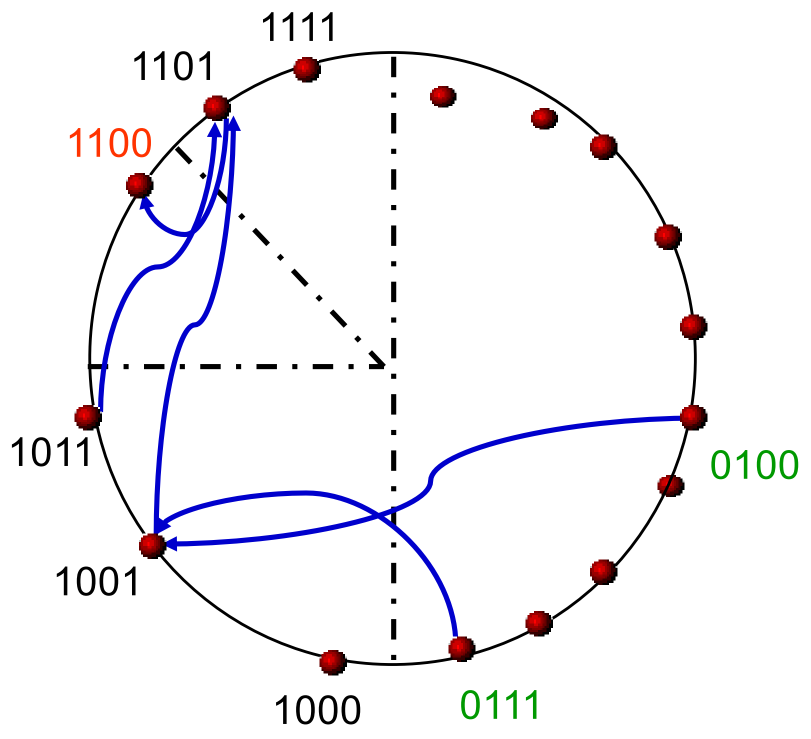


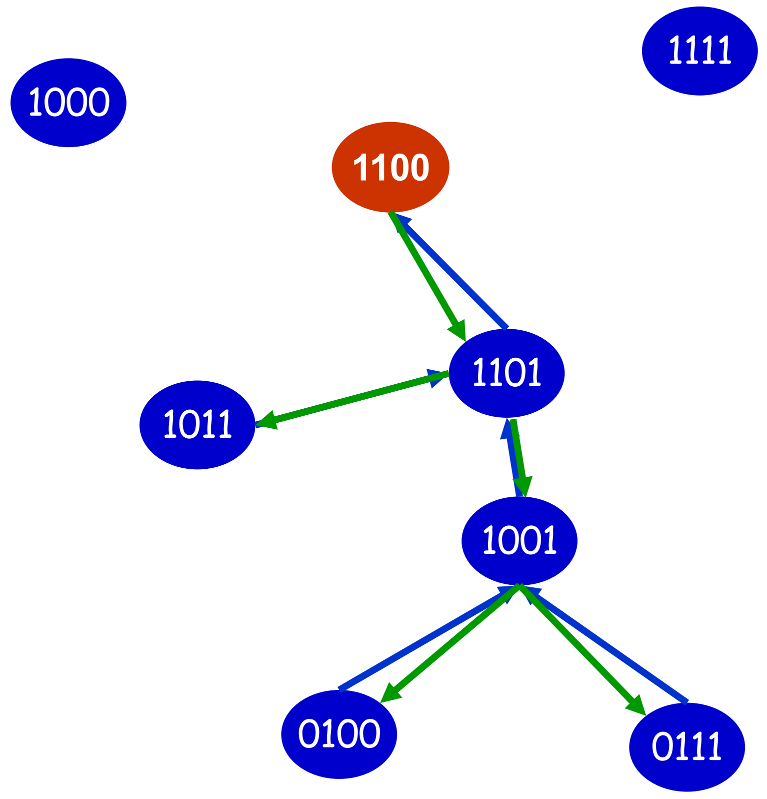
- Each group is assigned an identifier *groupId = Hash(name)*
- Multicast tree root : node which nodeId is the numerically closest to the groupId
- **Create(group):** P2P routing using the *groupeId* as the key

One tree per topic

- **join(group)**: message sent through Pastry using *groupeId* as the key
- **Multicast tree**: union of Pastry routes from the root to each group
  - Low latency: leverages Pastry proximity routing
  - Low network link stress: most packets are replicated low in the tree

Scribe : join(group)



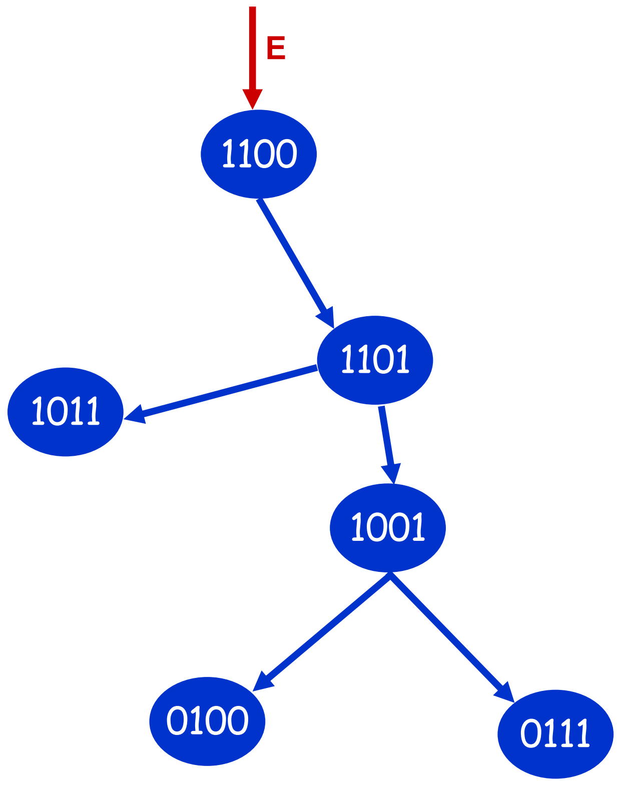


Scribe: message dissemination

Multicast(group, m)

- Routing through Pastry to the root key=*groupeId*

- Flooding along the tree branches from the root to the leaves

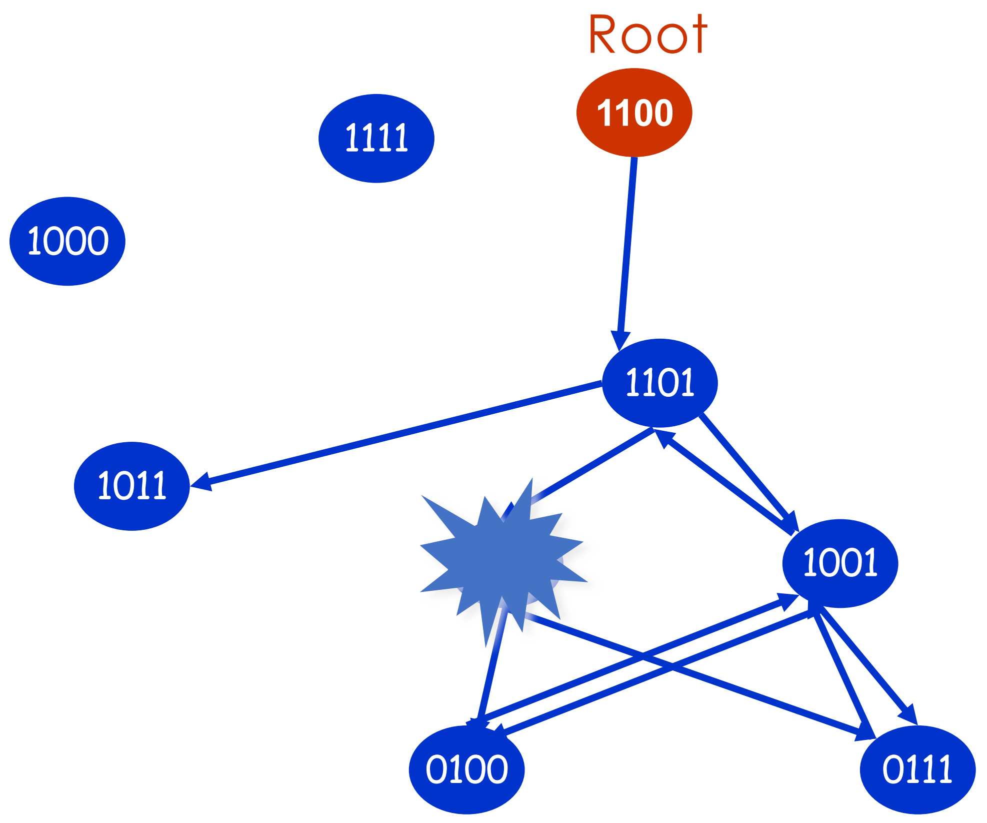


EPFL

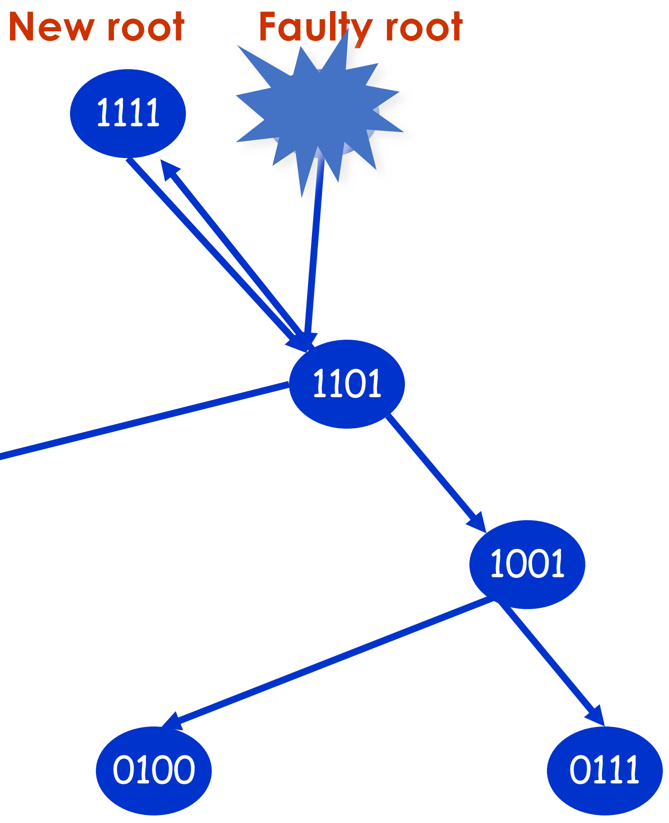
Reliability

- « best effort » reliability guarantee
  - Tree maintenance when failures are detected
  - Stronger guarantee may also be implemented
- Node failure
  - Parents periodically send heartbeat messages to their descendants in the tree
  - When such messages are missed, nodes join the group again
- Local reconfiguration
- Pastry routes around failures

Tree maintenance



Tree maintenance



EPFL

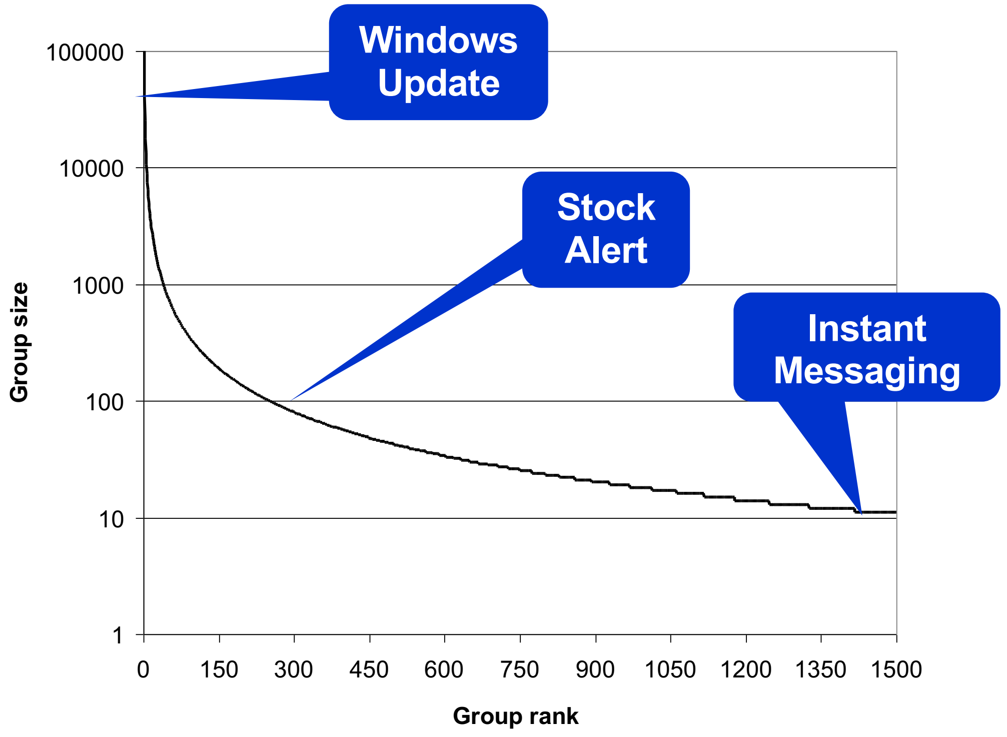
Load balancing

- Specific algorithm to limit the load on each node
    - Size of forwarding tables
- Specific algorithm to remove the forwarders-only peers from the tree
    - small-size groups

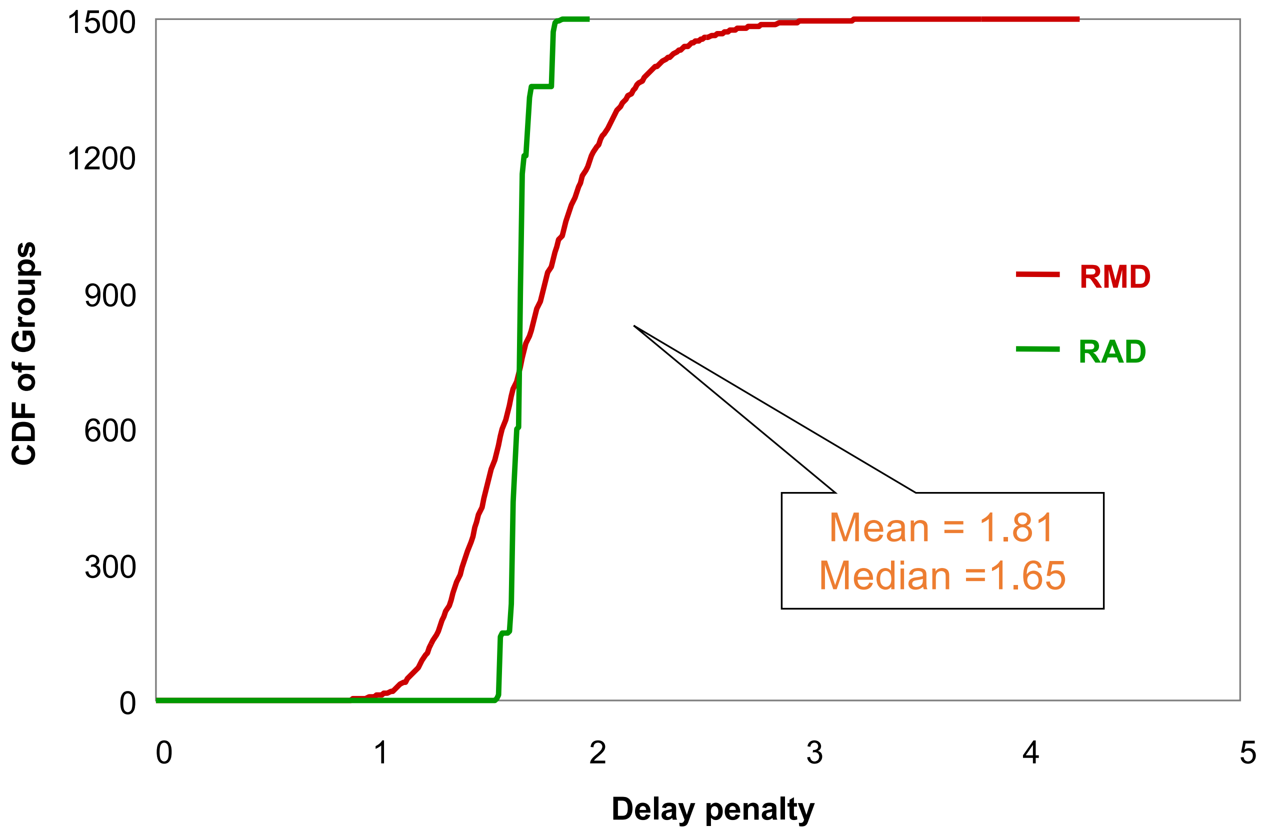
Scribe performance

- Discrete event simulator
- Evaluation metrics
  - Relative delay penalty
    - RMD: max $\text{delay}_{\text{app-mcast}}$ / max $\text{delay}_{\text{ip-mcast}}$
    - RAD: avg $\text{delay}_{\text{app-mcast}}$ / avg $\text{delay}_{\text{ip-mcast}}$
  - Stress on each network link
  - Load on each node
    - Number of entries in the routing table
    - Number of entries in the forwarding tables
- Experimental set-up
  - Georgia Tech *Transit-stub* model *(5050 core routers*)
  - 100 000 nodes chosen at random among 500 000
  - Zipf distribution for 1500 groups
  - Bandwidth not modeled

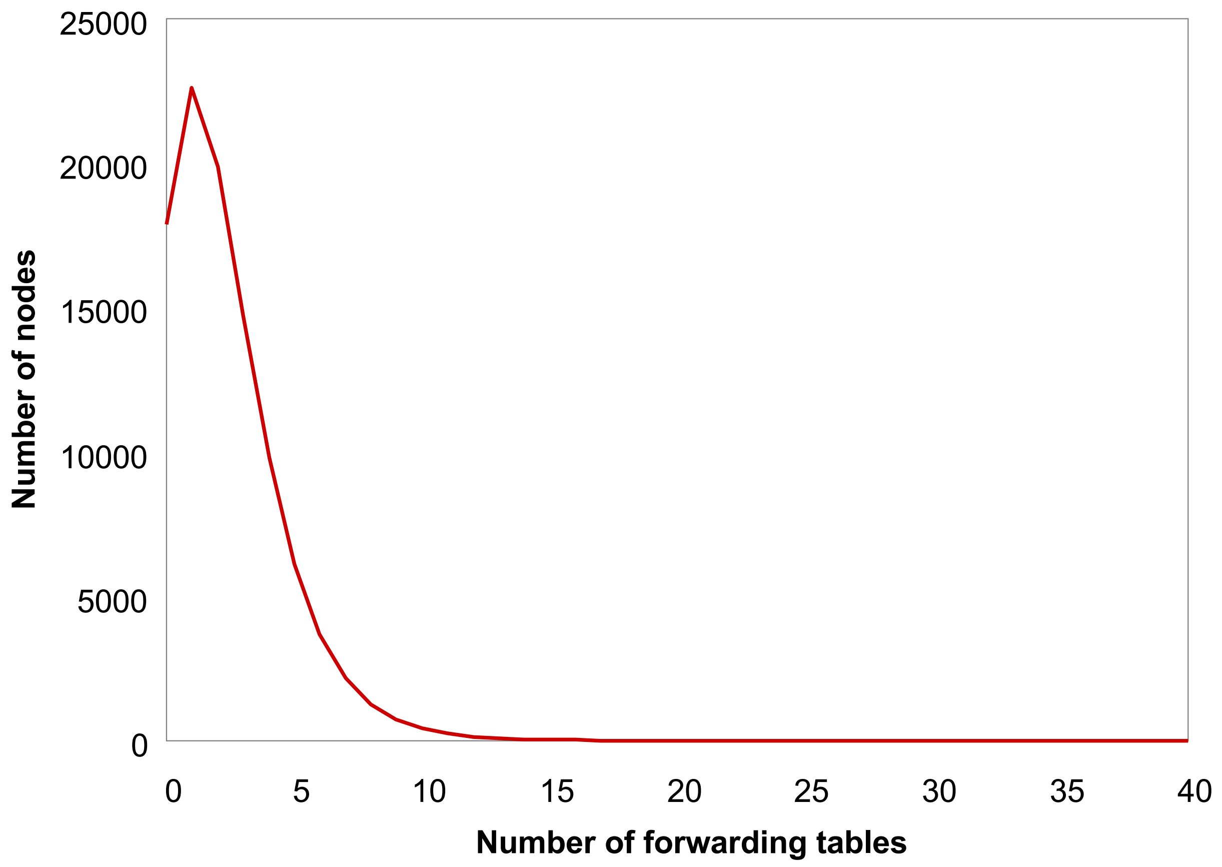
Group distribution



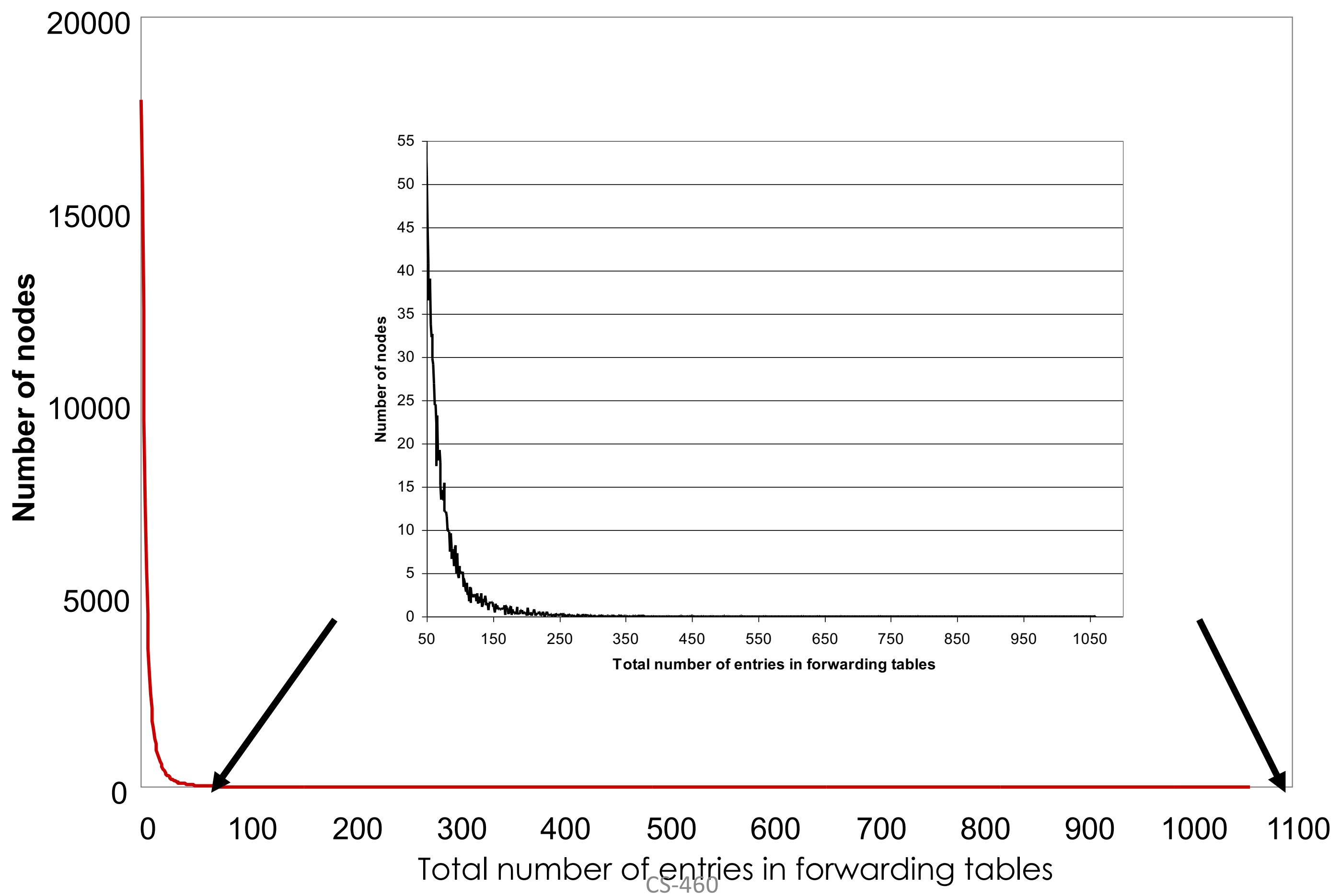
Delay/IP


1500
1200
900
600
300
0
0
1
2
3
4
5
CDF of Groups
Delay penalty
RMD
RAD
Mean = 1.81
Median =1.65

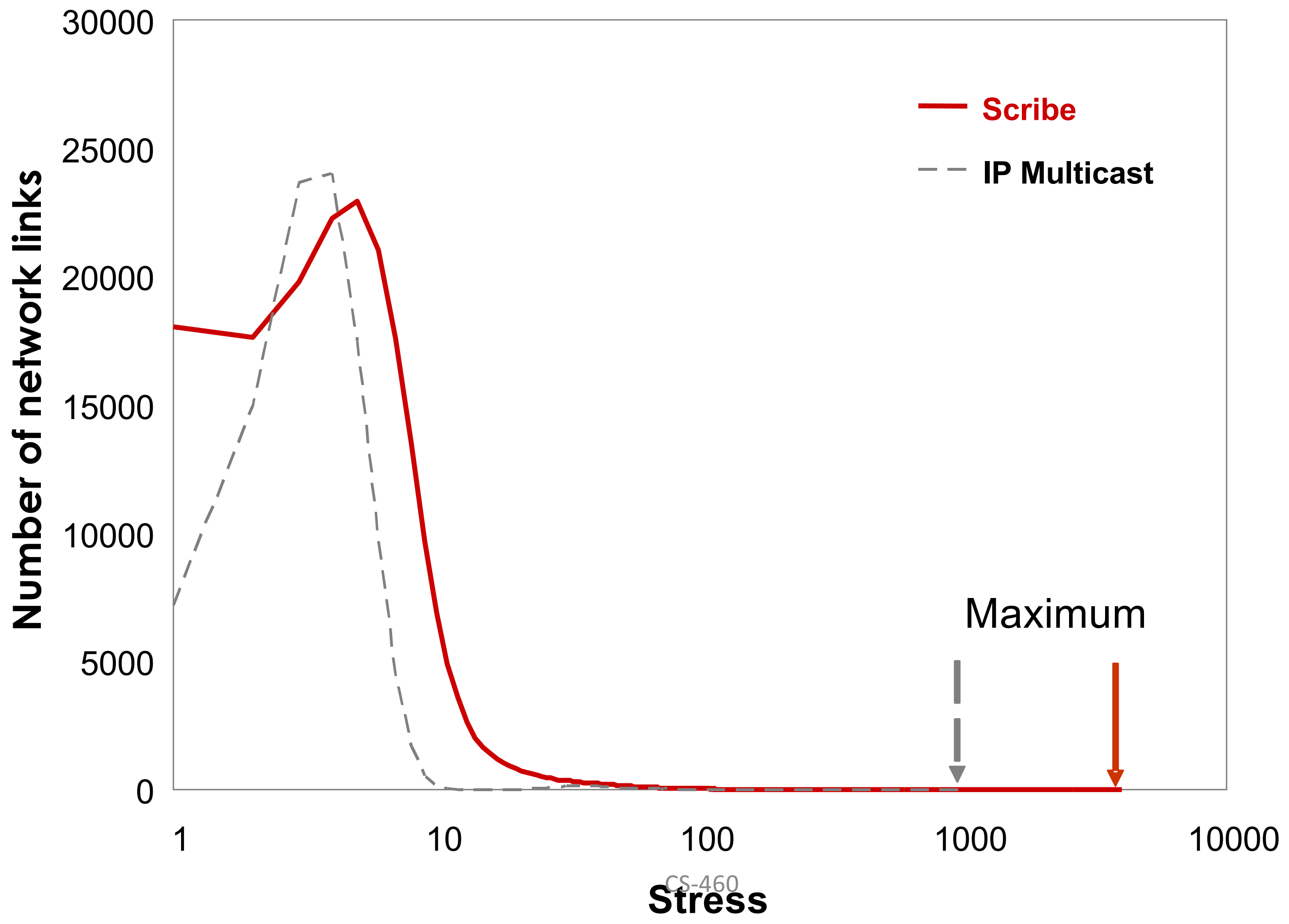
Load balancing



Load balancing



Network load



Summary

- Generic P2P infrastructures
  - Good support for large-scale distributed applications
  - ALM Infrastructure
- Scribe exhibits good performances/IP multicast
  - Large size groups
  - Large number of groups
  - Good load-balancing properties

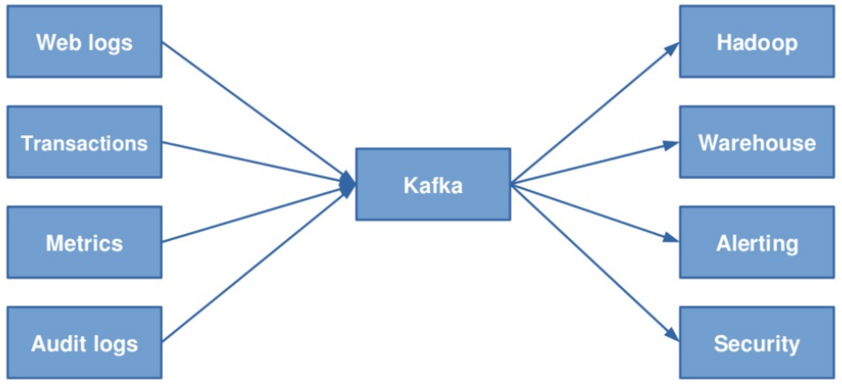
kafka

Kafka: A distributed messaging system for log processing

Developed by LinkedIn, now Apache, written in Scala and Java

Kafka

- Kafka is a distributed, topic-based, partitioned, replicated commit log service.
- Fast
- Scalable
- Durable
- Distributed
- A log-based message broker
- Distributed stream processing software



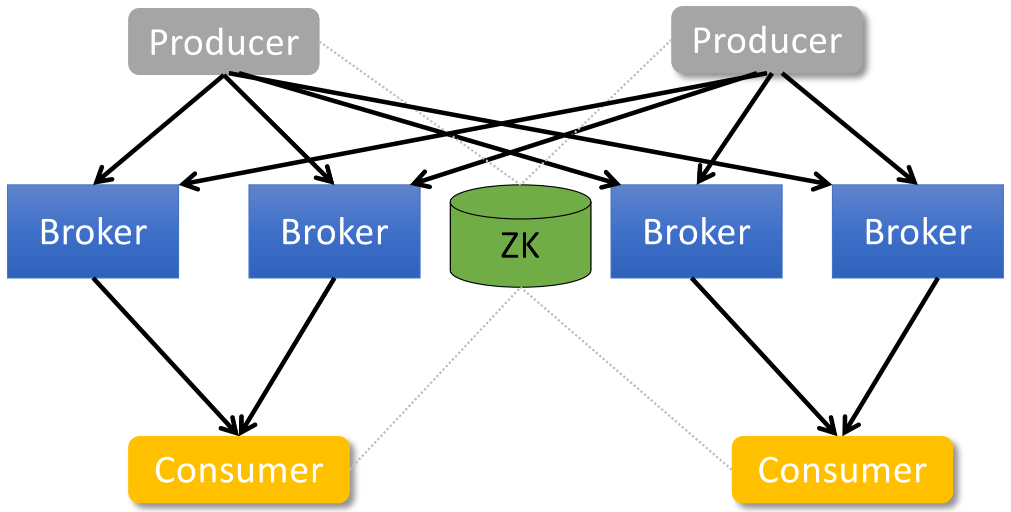
Kafka adoption and use cases

- Widely spread across industries such as healthcare, finance, retail, and manufacturing (˜50 000 companies in 2025).
- **LinkedIn:** activity streams, operational metrics, data bus
  - 400 nodes, 18k topics, 220B msg/day (peak 3.2M msg/s)
- **Tesla:** stream processing to handle trillions of IoT events daily, uses Kafka to ingest, process, and analyze data from its vehicle fleet in real time
- **Netflix**: real-time monitoring and event processing
- **X**: as part of their Storm real-time data pipelines
- **Spotify**: log delivery (from 4h down to 10s), Hadoop
- Airbnb, Cisco, Gnip, InfoChimps, Ooyala, Square, Uber, JPMorgan,…
- Mediego☺

Kafka in a nutshell

- **Producers** write data to **brokers**.
- **Consumers** read data from **brokers (Pull model)**
- Distributed, run in a cluster

- The data
  - Data is stored in **topics**.
  - **Topics** are split into **partitions**, which are **replicated**.



Partitioned Logs

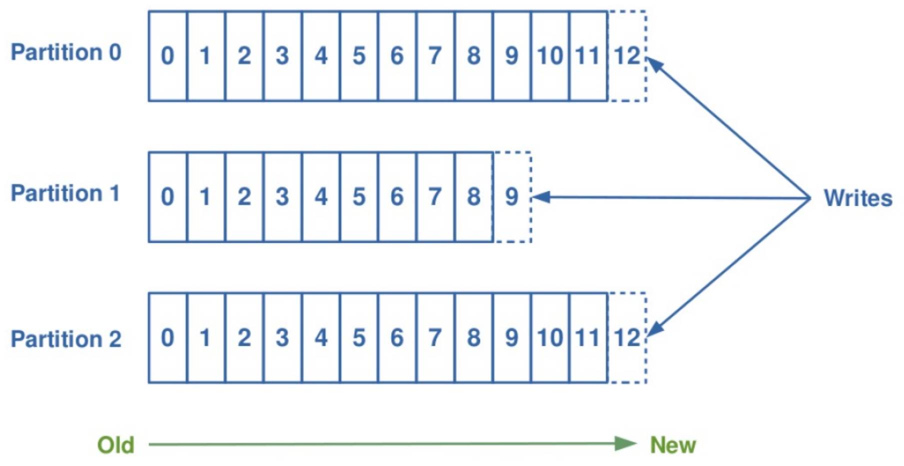
- In typical message brokers, once a message is consumed, it is deleted.
- Log-based message brokers durably store all events in a sequential log.
- A log is an append-only sequence of records on disk.
- A producer sends a message by appending it to the end of the log.
- A consumer receives messages by reading the log sequentially.

Partitioned Logs

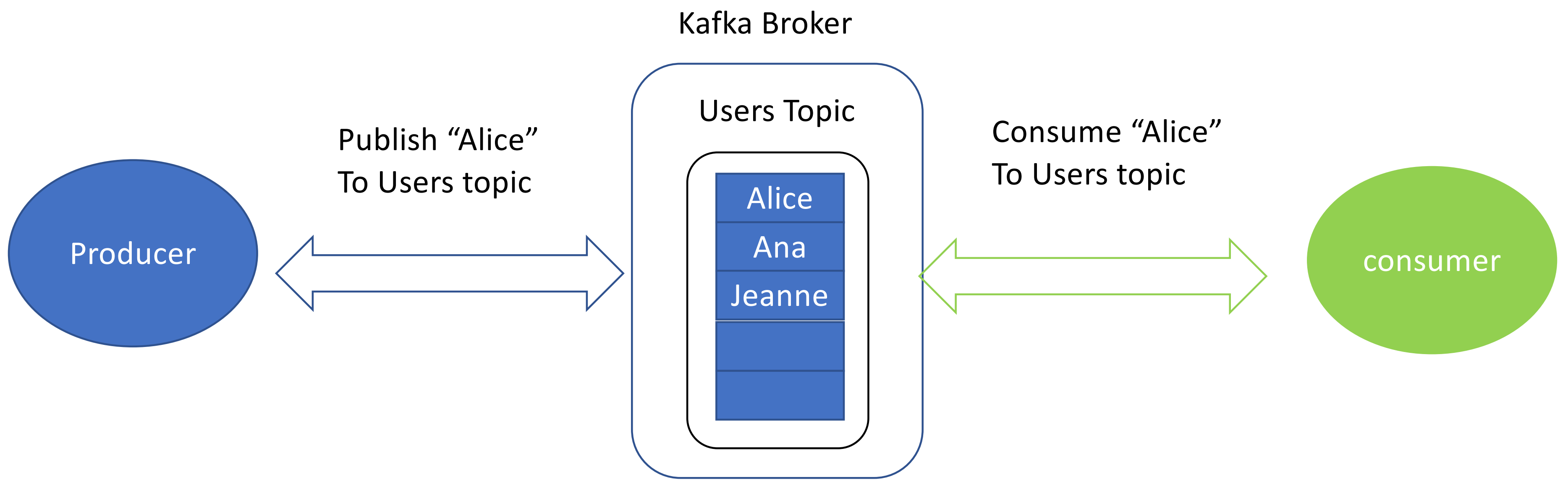
- To scale up the system, logs can be partitioned hosted on different machines.
- Each partition can be read and written independently of others.
- A topic is a group of partitions that all carry messages of the same type.
- Within each partition, the broker assigns a monotonically increasing sequence number (offset) to every message
- No ordering guarantee across partitions.

Topics

- Topics are queues: a stream of messages of a particular type
- Each message is assigned a sequential id called an offset (no overhead related to maintaining explicit message id)
- Topics are logical collections of partitions (the physical files).
  - Ordered
  - Append only
  - Immutable

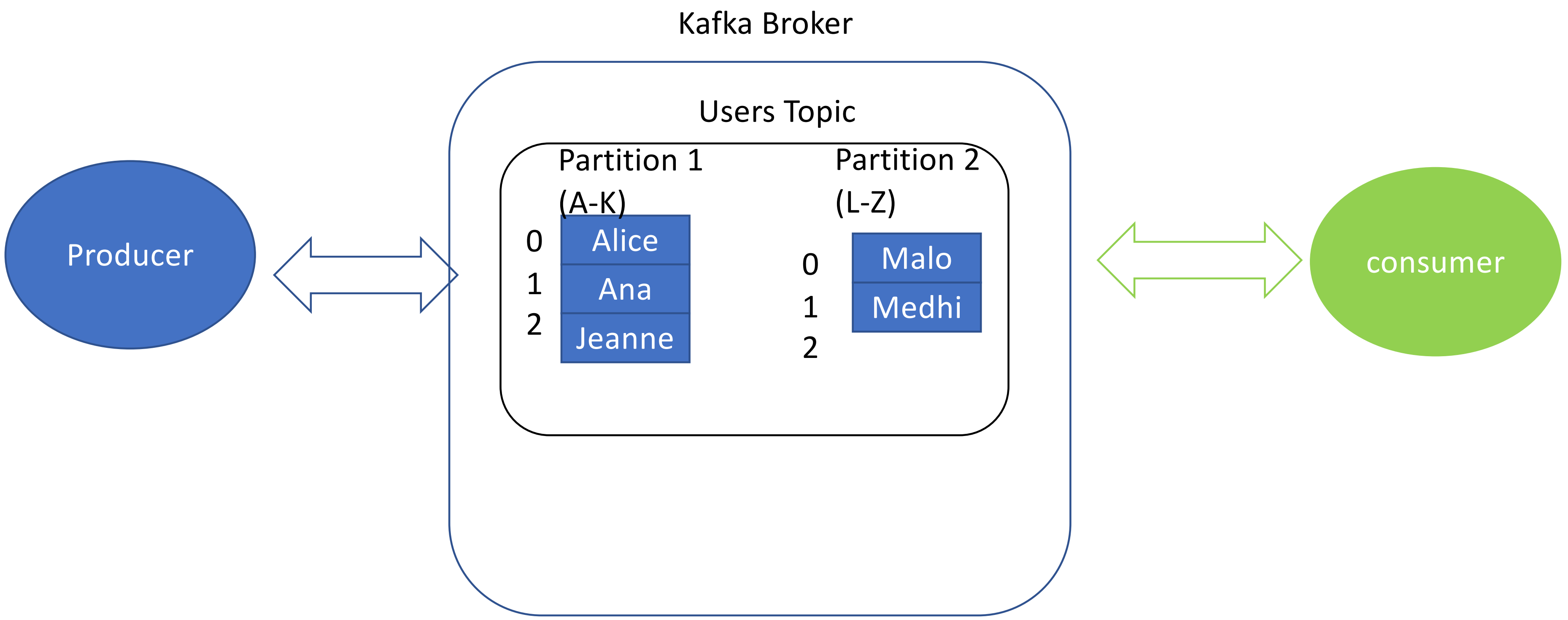


Kafka Producer



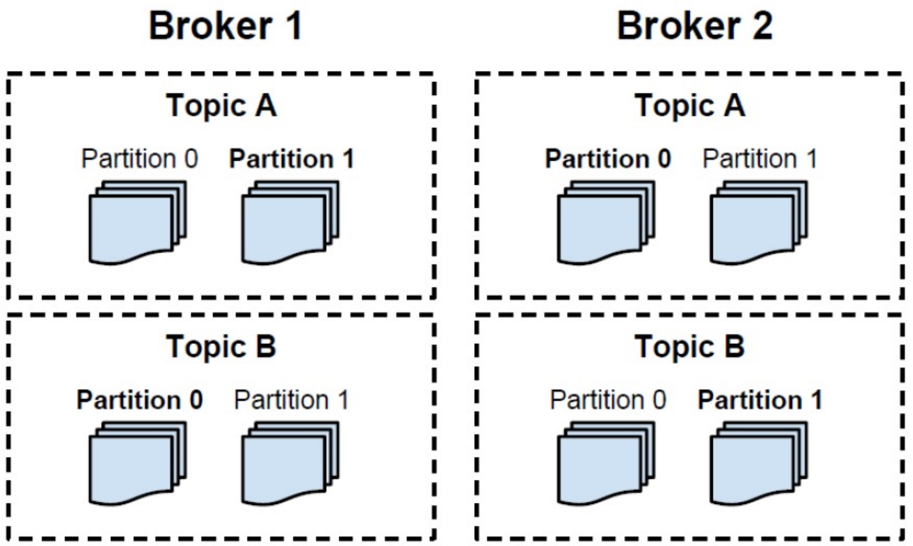
Topic, Position
Append only

Kafka Partitions

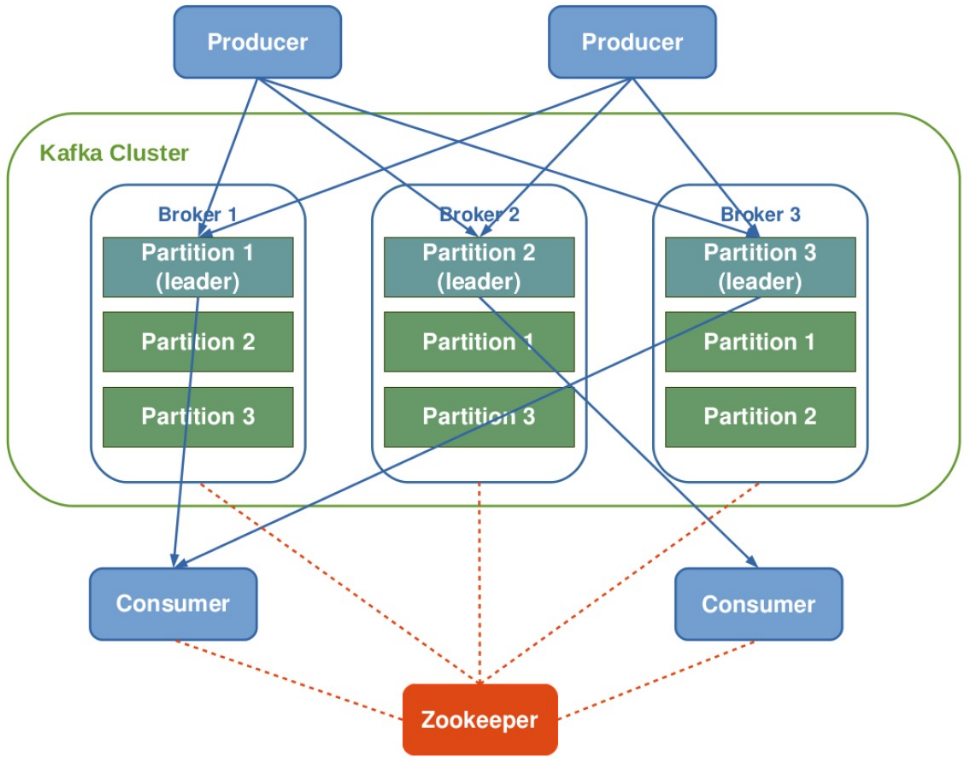


Partitions

- Partitions of a topic are replicated: fault-tolerance
- A broker contains some of the partitions for a topic: load-balancing
- One broker is the leader of a partition: all writes and reads must go to the leader.



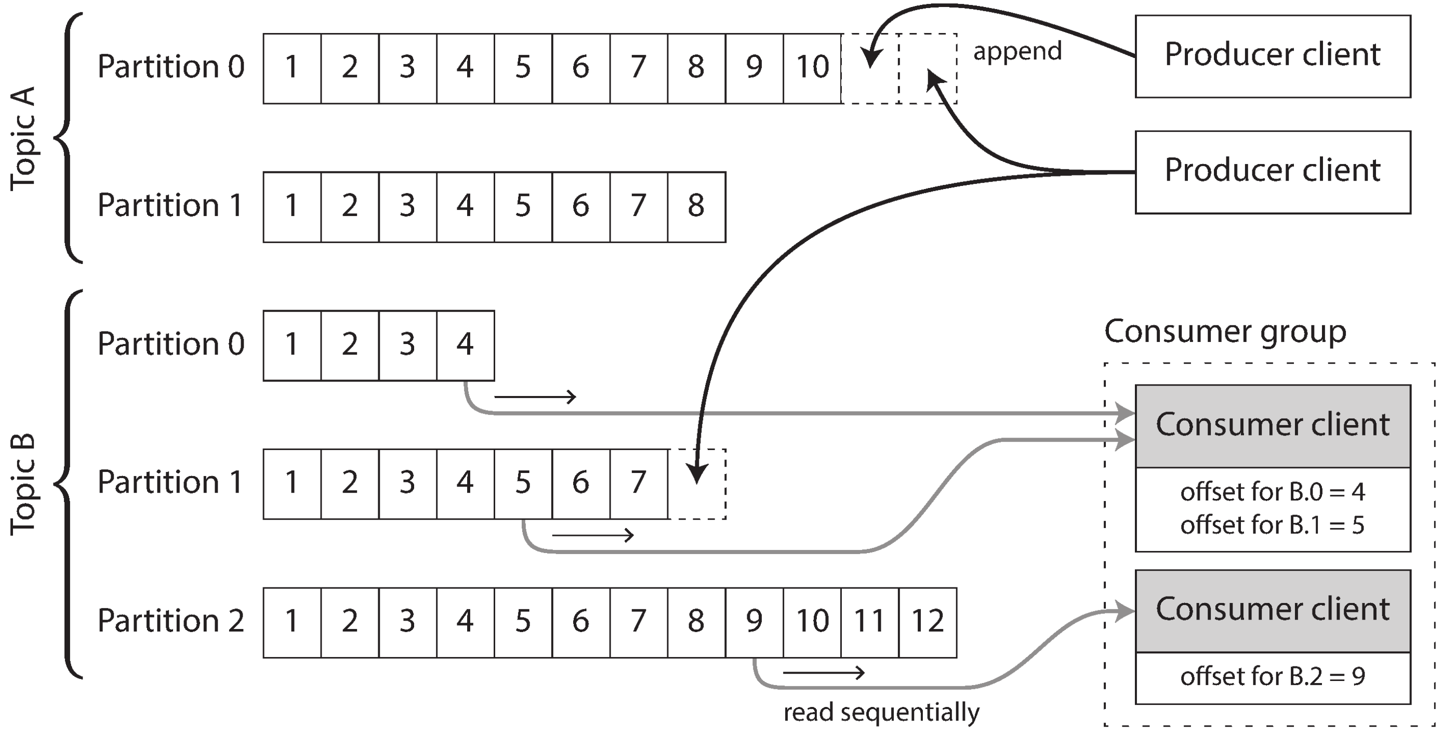
Kafka architecture



Consumer groups

- A consumer group: one or more consumers that jointly consume a set of subscribed topics
  - each message is delivered to only one of the consumers within the group.
  - at any given time, all messages from one partition are consumed only by a single consumer within each consumer group
    - Avoids synchronization

Partitioned Logs

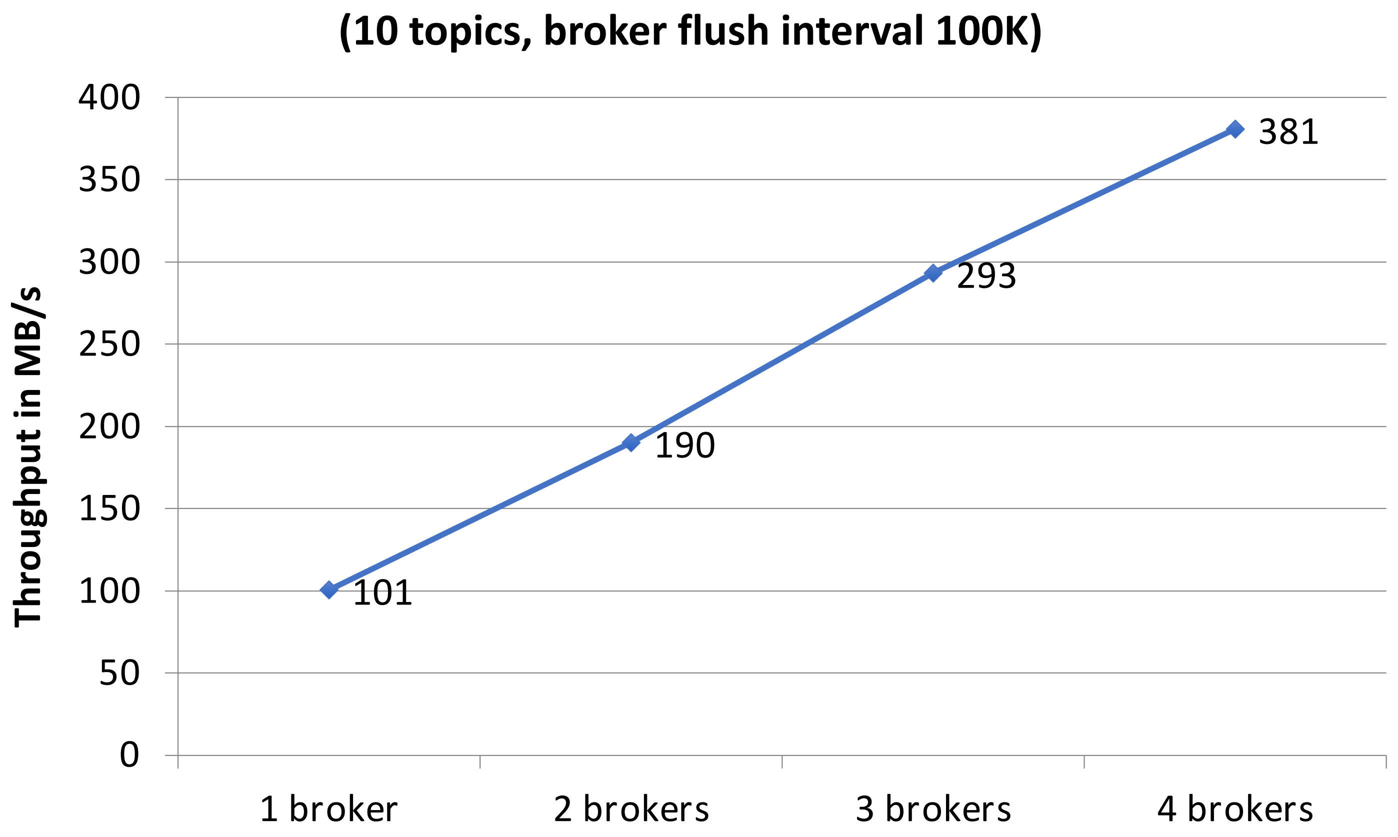


State and guarantees

- State
  - Brokers are stateless: no metadata for consumers-producers in brokers.
  - Consumers are responsible for keeping track of offsets.
  - Messages in queues expire based on pre-configured time periods (e.g., once a day).
  - Side benefit : A consumer can deliberately rewind back to an old offset and re-consume data.
- Delivery guarantees
  - Kafka guarantees that messages from a single partition are delivered to a consumer in order.
    - There is no guarantee on the ordering of messages coming from different partitions.
    - Kafka only guarantees at-least-once delivery (the client needs to check for duplicate)
- Kafka uses Zookeeper (up to 2025) for the following tasks:
  - Detecting the addition and the removal of brokers and consumers.
  - Keeping track of the consumed offset of each partition.

EPFL

Scalability



Kafka

- Simple and efficient (hight throughput)
- Persistent storage
- Pull-based pub-sub system
- Widely used in industry

References

The many faces of publish/subscribe. Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, Anne-Marie Kermarrec. ACM Computing Surveys June 2003.

XL peer-to-peer pub/sub systems. Anne-Marie Kermarrec & Peter Triantafillou. ACM Computing Surveys Nov. 2013.

Kafka: A distributed messaging system for log processing. J. Kreps et al. NetDB, 2011

Spark: The Definitive Guide. M. Zaharia et al., O'Reilly Media, 2018 - Chapter 20

Fundamentals of stream processing: application design, systems and analytics. H. Andrade et al., Cambridge University Press, 2014 - Chapter 1-5, 7, 9

High-availability algorithms for distributed stream processing. J. Hwang et al., ICDE 2005

References (ALM)

- M. Castro, P. Druschel, A-M. Kermarrec and A. Rowstron, "**SCRIBE: A large-scale and decentralised application-level multicast infrastructure**", IEEE Journal on Selected Areas in Communication (JSAC), Vol. 20, No, 8, October 2002.
- M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron and A. Singh, "**SplitStream: High-bandwidth multicast in a cooperative environment**", SOSP'03, Lake Bolton, New York, October, 2003.
- Shelley Q. Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy Katz John Kubiatowicz « ***Bayeux: An Architecture for Scalable and Fault-tolerant Wide-area Data Disseminatio****n* »Eleventh International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2001)
- Sylvia Ratnasamy, Mark Handley, Richard Karp, Scott Shenker « **Application-level Multicast using Content-Addressable Networks** » (2001) Lecture Notes in Computer Science, NGC 2001 London.
- D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. « ***Bullet: High bandwidth data dissemination using an overlay mesh*** ». In 19th ACM Symposium on Operating Systems Principles, October 2003.