Introduction to Stream Processing

Outline

- Motivating Applications
  - Need for Stream Processing
  - Classes of Applications
  - Key Application Requirements
- Emergence of Stream Processing
  - Databases → Rule Engines → Stream Processing
- Stream Processing Systems
  - Examples
  - Anatomy of a Stream Processing System

Data Deluge

Every natural system and man-made system
is becoming interconnected, instrumented and intelligent

Utilities

Healthcare

Cities

Food

Manufacturing

Transportation

Oil & Gas

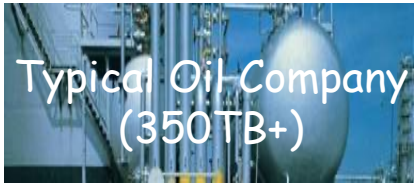
Safety

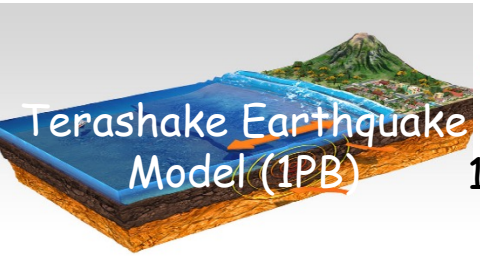
User Data:
Smart Phones
and Homes

Data Deluge

YouTube
500 hrs of videos/min, 2x in 5 months

95 Million Images/Day





120 PB Disk Storage

Total digital data created in 2012 **1.8 Zettabytes**. In 2017 **16.3**, in 2020 **44 ZettaBytes**! Not just numeric timeseries, mostly unstructured

Need for Streaming Analysis

- Ever increasing
  - Number of data sources
  - Volume of data
  - Variety of data
  - Velocity of data
- To utilize data
  - React fast – low latency
  - Analyze more data – high volume
  - Correlate different sources – variety
  - Apply complex, predictive analytics, and learning
  - Present live results and take action

Healthcare and Internet of Things

MiniMed Connect

4. integrate with other context information from phone (accelerometer, location, other vitals) and other devices, e.g. fitbit

1. Capture streaming data, and 24 hour snapshot

2. Pre-process, clean, transform and prepare for storage and summary

3. Provide authenticated access and summaries of streaming data, customized to end-user preferences

6. Apply models against streaming data to create real-time insights and provide the user with: Insulin Recommendations, Behavior Recommendations, and identified deviations and patterns

5. Develop models of Patient insulin regimen, Activity and wellness, and Patient Context

Cloud

Medtronic Store
(Supplies, Accessories…)

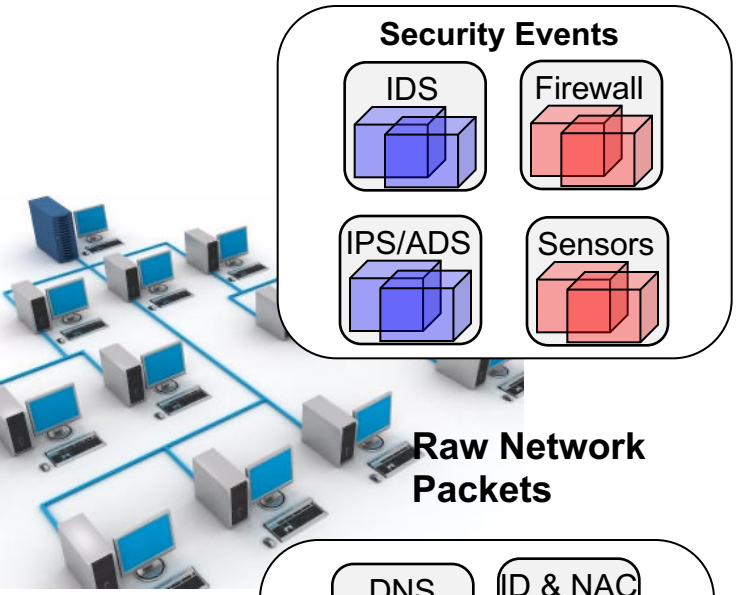
Doctor
prescriptions and Updates

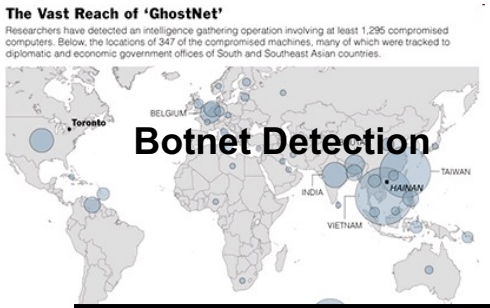
UnitedHealthcare
WELLPOINT
CIGNA
Aetna
HUMANA.

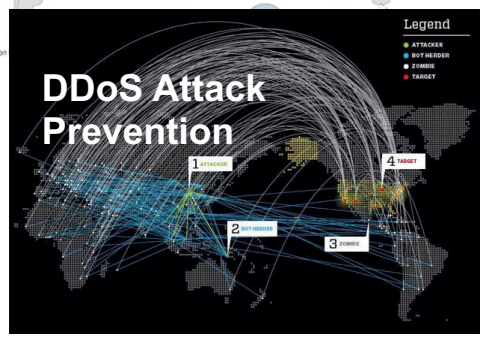
A Streaming Healthcare Application

- Streaming Timeseries Data
  - Heterogeneous (Health, Context, Location)
  - Multiple sources
  - Noise, Delay, Missing values, Unobserved Information
- Streaming Analysis
  - Continuous prediction, pattern mining, live alerting (hypo prediction)
  - Online learning
- Systems
  - Scaling, distributed processing, fault tolerance

Use Case: Cybersecurity







Automated Threat Analysis and Mitigation

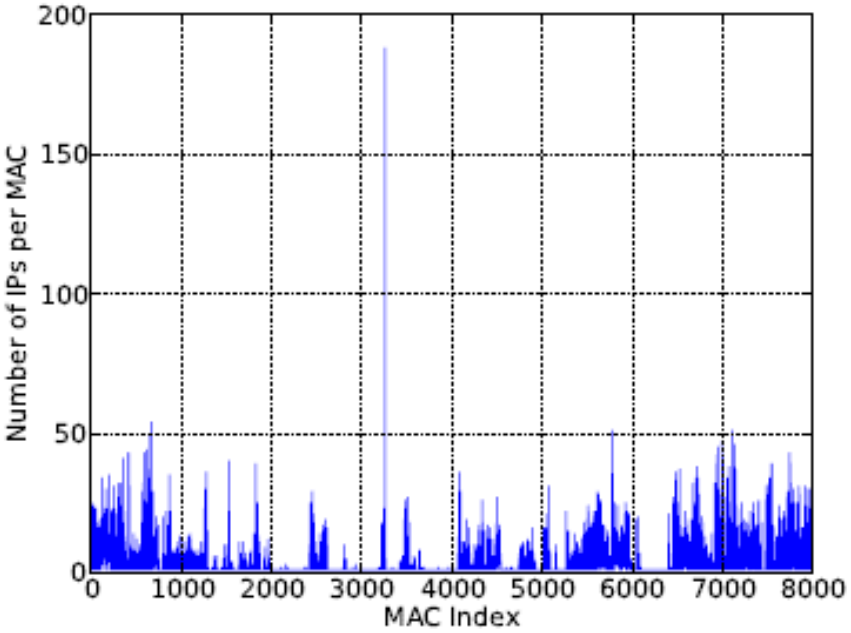
Discriminative network behavioral patterns for infection detection & prevention

Securing assets and information

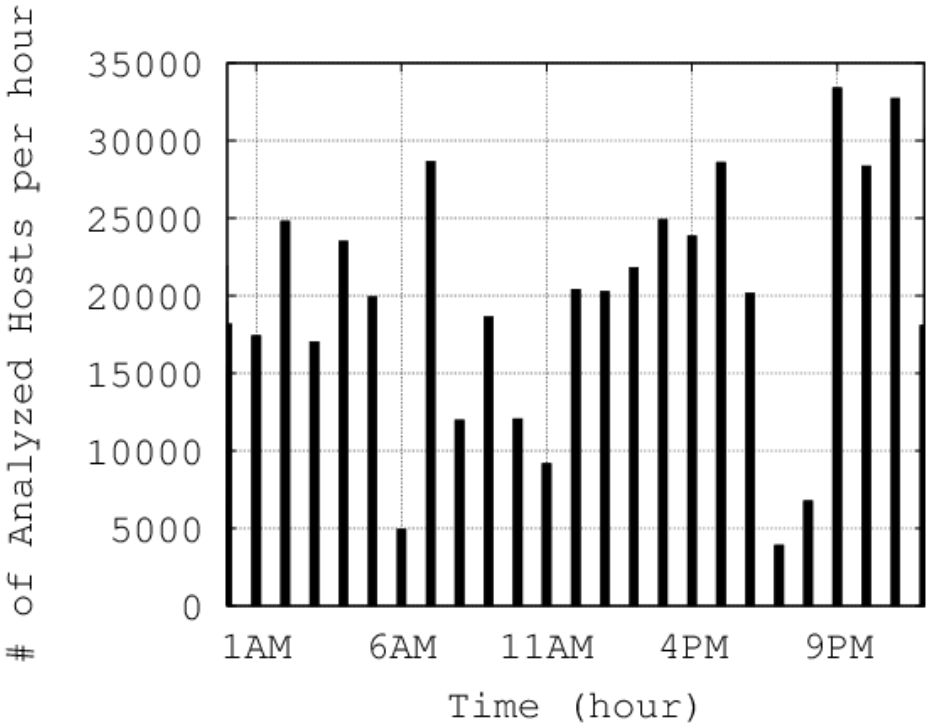
A Streaming Network Security Application

- Streaming Timeseries Data
  - Heterogeneous (Control Information, Raw Packets)
  - High throughput
  - Structured and Unstructured Content
- Streaming Analysis
  - Continuous threat monitoring, outlier detection
  - Online and continuous learning
- Systems
  - Scaling, distributed processing, fault tolerance

Network Dynamics

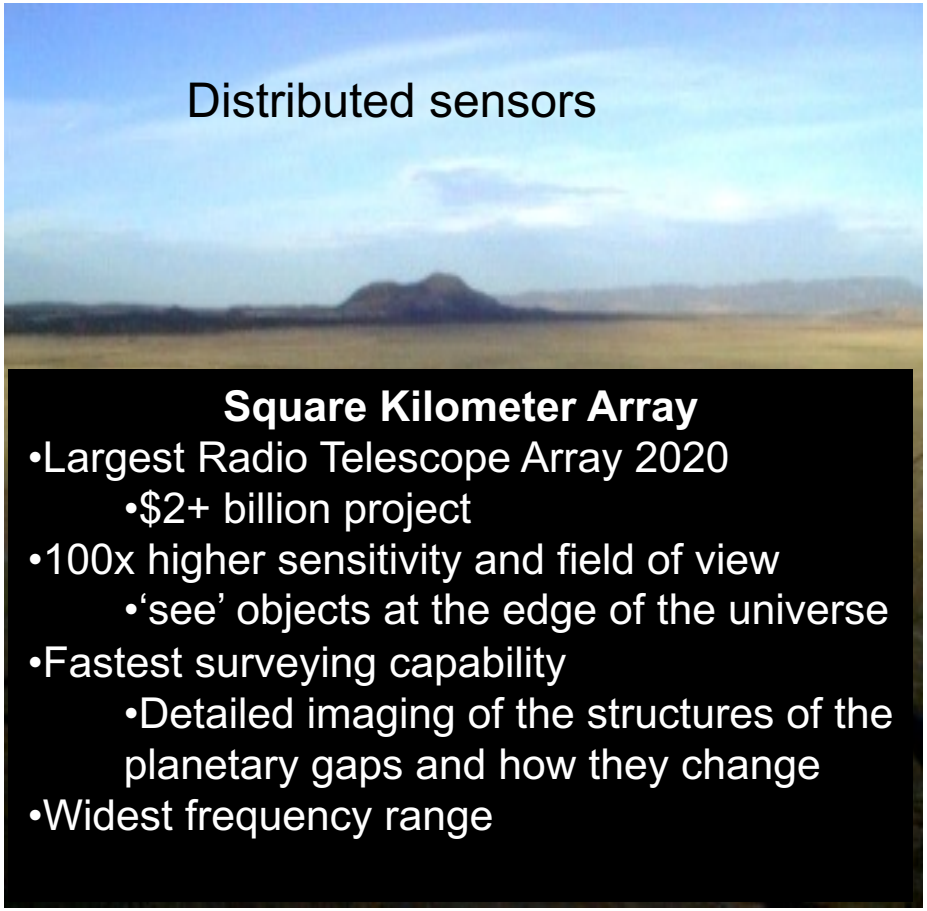


DHCP Churn: Dynamic Client Identity



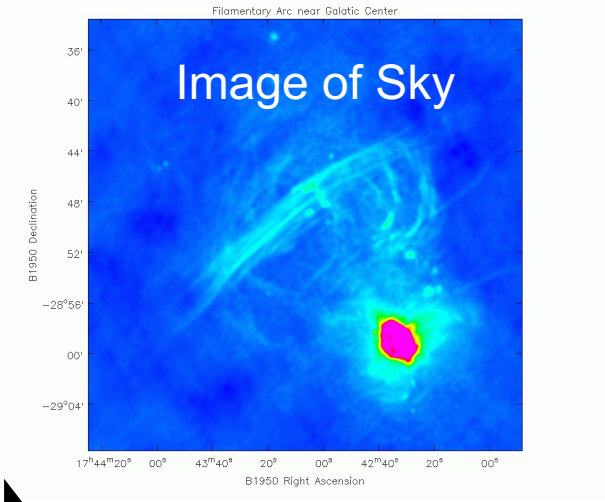
Active hosts per hour (1 day period)

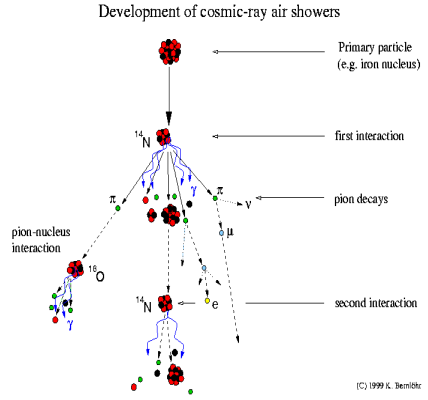
Use Case: Radio Astronomy



Square Kilometer Array

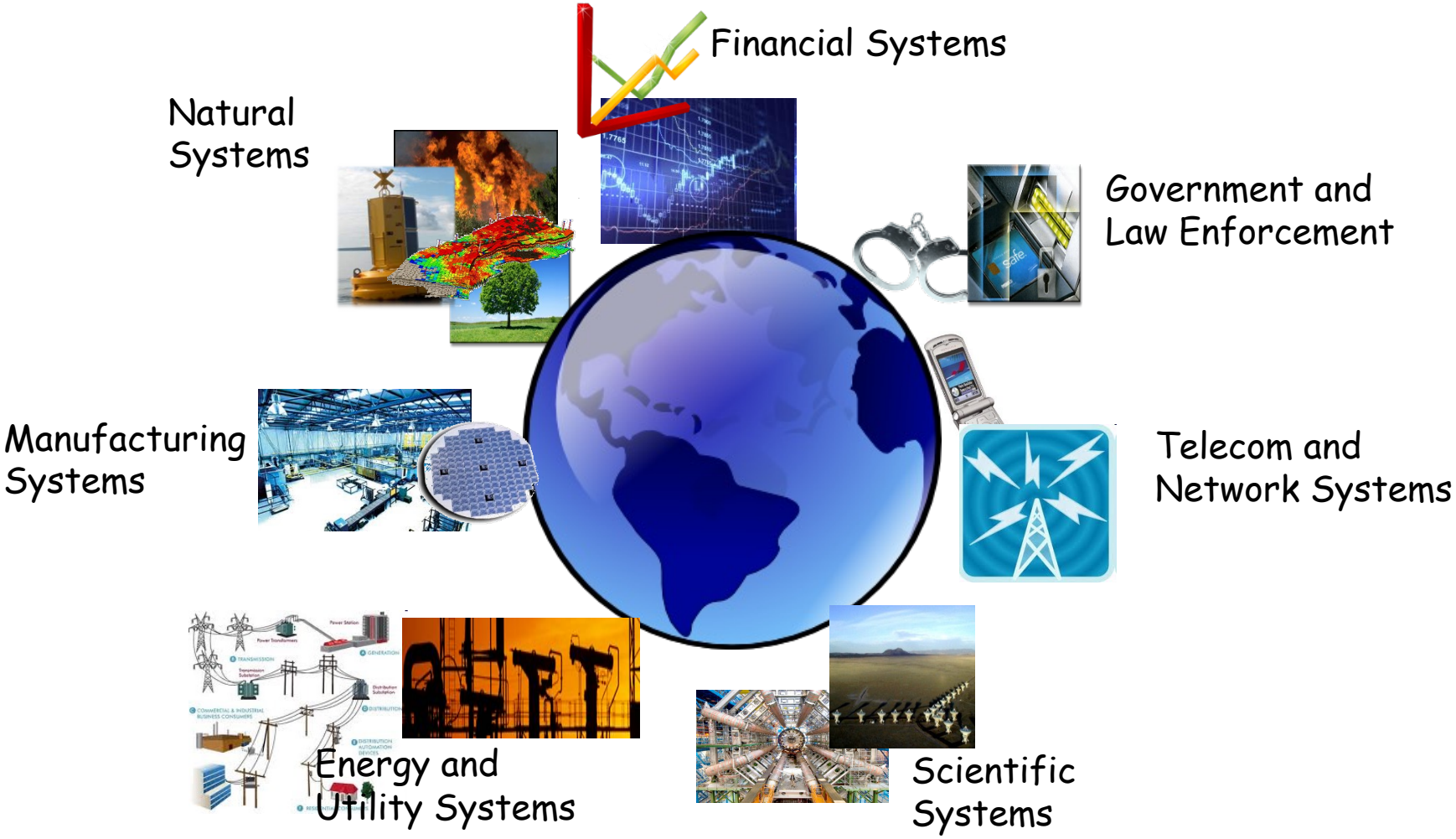
- Largest Radio Telescope Array 2020
  - $2+ billion project
- 100x higher sensitivity and field of view
  - ‘see’ objects at the edge of the universe
- Fastest surveying capability
  - Detailed imaging of the structures of the planetary gaps and how they change
- Widest frequency range





Transient Detection (Cosmic Rays)

Several Other Use Cases



Streaming Application Requirements

- Streaming Data
  - Wide Range of Data Rates: Manufacturing: 5-10 Mbps, Astronomy: ~x00 Gbps, Healthcare: ~x00 Kbps per patient
- Streaming Analysis
  - Hierarchical, open-ended, and long running analysis. Online learning
- Multimodal Data Analysis
  - Correlated primary sources, Structured or Unstructured
- Distributed Analysis
  - Decomposable into flowgraphs, distributed data, processing
- High Performance
  - Real-time, Low-latency, high throughput, scaling
- Dynamic and Adaptive Analysis
  - Source, data, analysis and resource variability
- Loss Tolerant Analysis
  - Highly noisy and lossy data, graceful degradation under failure

Classes of Stream Processing Apps

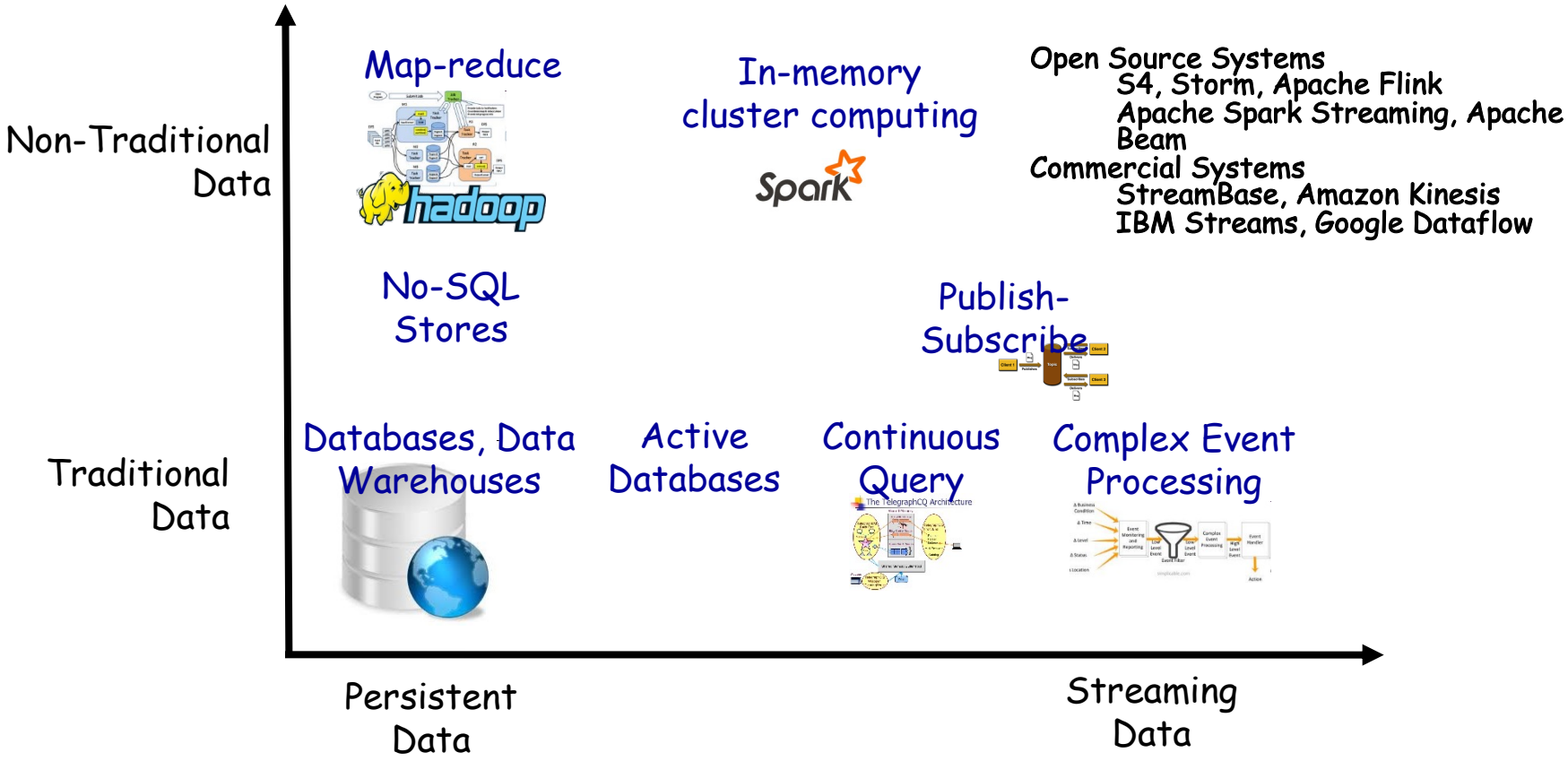
- Performance Driven
- Control and Decision Support
- Exploration Driven
- Historical Analysis and Simulation

Why Stream Processing Systems?

- Need a natural paradigm for handling streaming data applications
- Programming streaming applications
  - Language, constructs
  - IDE, Debugging, Visualization
- Running and scaling streaming applications
  - Runtime spanning distributed machines
- Tools
  - Toolkits, connectors, adapters, optimizers

- Makes it “easy” to develop, deploy, and manage streaming applications

Towards Stream Processing Systems



Towards Stream Processing Systems

System	Streaming Analysis	Distributed Processing	High Performance and Scalability	Unstructured Analysis	Fault Tolerance	Adaptive Analysis
Databases (DB2, Oracle, MySQL)	No	Yes	Partly	No	Yes	No
Parallel Processing (PVM, MPI, OpenMP)	No	Yes	Yes	Yes	Yes	No
Active Databases (Ode, HiPac, Samos)	Partly	Partly	No	No	Yes	Yes
Continuous Query Systems (NiagaraCQ, OpenCQ)	Partly	Partly	No	No	Yes	Yes
Pub-Sub Systems (Gryphon, Siena, Padres)	Yes	Yes	No	No	Yes	Partly
CEP Systems (Esper, SASE, IBM WBE, Tibco BE, Oracle CEP)	Yes	Partly	Partly	No	Yes	Partly
Map-Reduce (Hadoop)	No	Yes	Yes	Yes	Yes	No

Evolution towards Stream Processing Systems

- Distributed Computing Systems
- Data Management Systems
  - Databases and Data Warehouse
- Information Flow Processing Systems
  - Active Databases
  - Continuous Query Systems
  - Publish-Subscribe Systems
  - Complex Event Processing Systems
- Stream Processing Systems

Distributed Computing Systems

- **Distributed systems**
  - Multiple computers interacting through a computer network to support a single application or a family of interconnected applications – i.e., a ***distributed application***

- **Why Distributed Systems?**
  - Applications require coordinated use of several computers
    - Data generated in one location and needed in another location
  - ***Cost***: more cost-efficient than single high-end computer
  - **Reliability**: increased reliability without single point of failure
  - **Scalability/Management**: easier to expand and manage

Distributed Systems – A Brief History

- Concurrent processes
  - Inter-process communication by ***message-passing*** in operating system architectures studied in 1960s
- ARPANET: late 1960s
- Local-area networks
  - Ethernet in 1970s
- ARPANET e-mail early 1970s
- Usenet and FidoNet from 1980s
  - distributed discussion systems

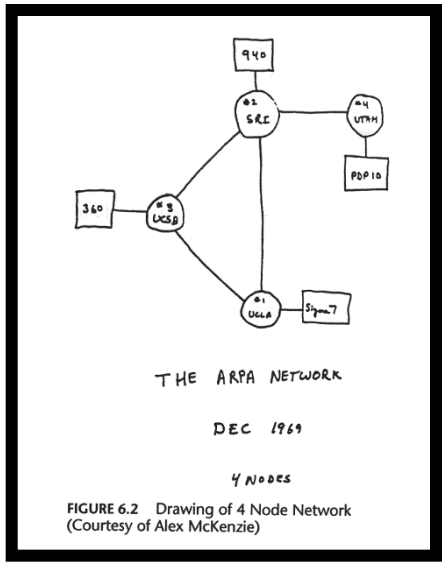


FIGURE 6.2 Drawing of 4 Node Network (Courtesy of Alex McKenzie)

Distributed Systems

- Multiple issues (versus sequential programs)
  - Algorithmic issues
    - Non-trivial design of parallel algorithms
  - "Mechanical" issues
    - Running, debugging, optimizing applications

- Multiple paradigms developed
  - Algorithmic issues
    - Still on our own, in most cases… ☺
    - Certain classes of problems (in computational physics, finance, visualization) have well-defined and friendly programming environments with pre-programmed efficient algorithms underneath them
  - "Mechanical" issues
    - Addressed by different programming environments
    - Language and management constructs
    - Two popular infrastructures: PVM and MPI

Distributed Systems – PVM

- Parallel Virtual Machine (PVM): software for parallel networking of loosely coupled computers
  - Network of machines can be used as a single distributed parallel processor:
  - Run-time environment and library for message-passing, Task and resource management, Fault notification
  - Functions for ***manually*** parallelizing an existing source program, or for writing new parallel/distributed programs.
- Process-based computation: Unit of parallelism is a ***task***
  - Independent sequential thread of control with communication and computation
  - Multiple tasks may execute on a single processor
- Explicit message-passing model
  - Collections of computational tasks
  - Parallelization strategy: data-, task-, or hybrid decomposition
  - Tasks cooperate by explicitly sending and receiving messages
- Heterogeneity support
  - supports heterogeneity in terms of machines, networks, and applications
- Multiprocessor support
  - Native message-passing facilities on multiprocessors
  - Vendors often supply their own optimized PVM for their systems

```
#include “pvm3.h”
main() {
  int ptid, msgtag;
  char buf[100];
  ptid = pvm_parent();
  strcpy(buf, "hello, world from ");
  gethostname(buf + strlen(buf), 64);
  msgtag = 1;
  pvm_initsend(PvmDataDefault);
  pvm_pkstr(buf);
  pvm_send(ptid, msgtag);
  pvm_exit();
}
```

Distributed Systems – MPI

- Message Passing Interface (MPI) ***specification***
  - Allows many computers and processors to work jointly for an application
  - Used in computer clusters and supercomputers.
  - dominant model for high-performance computing
  - Supports heterogeneous architectures
- Interface provides
  - Virtual topology management
  - Synchronization facilities
  - Communication functionality
- Library functions
  - Point-to-point rendezvous-type send/receive operations
  - Broadcast messages
  - Choice of Cartesian or graph-like logical process topology
  - Data (send/receive) between process pairs
  - Combining partial results of computations (gathering and reduction operations)
  - Synchronizing nodes (barrier operation)
  - Obtaining network-related information

```
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
MPI_Comm_rank(MPI_COMM_WORLD,&myid);

if(myid == 0) {
    printf("%d: We have %d processors\n", myid, numprocs);
    for(i=1;i<numprocs;i++)
    {
      sprintf(buff, "Hello %d! ", i);
      MPI_Send(buff, BUFSIZE, MPI_CHAR, i, TAG, MPI_COMM_WORLD);
    }
    for(i=1;i<numprocs;i++)
    {
      MPI_Recv(buff, BUFSIZE, MPI_CHAR, i, TAG, MPI_COMM_WORLD, &stat);
      printf("%d: %s\n", myid, buff);
    }
 }
 else {
    /* receive from rank 0: */
    MPI_Recv(buff, BUFSIZE, MPI_CHAR, 0, TAG, MPI_COMM_WORLD, &stat);
    sprintf(idstr, "Processor %d ", myid);
    strcat(buff, idstr);
    strcat(buff, "reporting for duty\n");
    /* send to rank 0: */
    MPI_Send(buff, BUFSIZE, MPI_CHAR, 0, TAG, MPI_COMM_WORLD);
 }
 MPI_Finalize();
```

Other Distributed Computing Trends

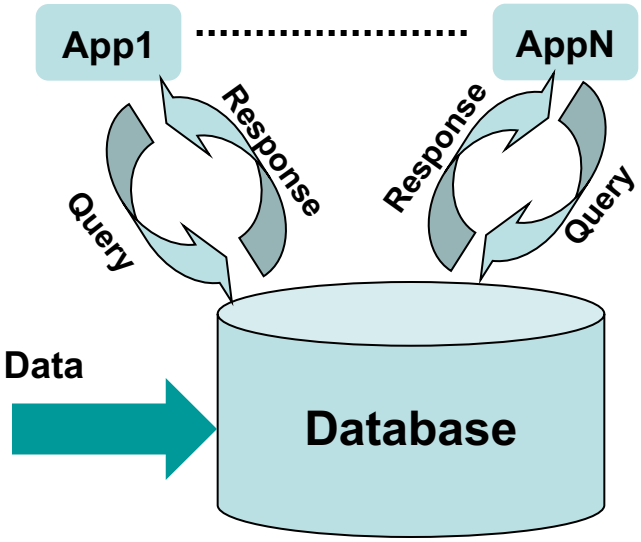
- **Hardware**
  - Multi-core
    - A single chip, multiple “independent” cores
  - Distributed/Hybrid infrastructure
    - Clusters, virtual machines, hybrid configurations, accelerators (GPUs, FPGAs, TPUs)
  - Inter-operation
    - Complex architectures of inter-connected hardware platforms
- **Software**
  - Large-scale inter-operation issues
    - Web-based, middleware-based, databases, solvers, etc…
    - CORBA, DCOM, RPC
  - Integration
    - Business intelligence frameworks
    - Web frontend, databases, component-based integration
    - Different parts written using different languages

Data Management Systems: Databases

Traditional Computing

Historical fact finding with data-at-rest

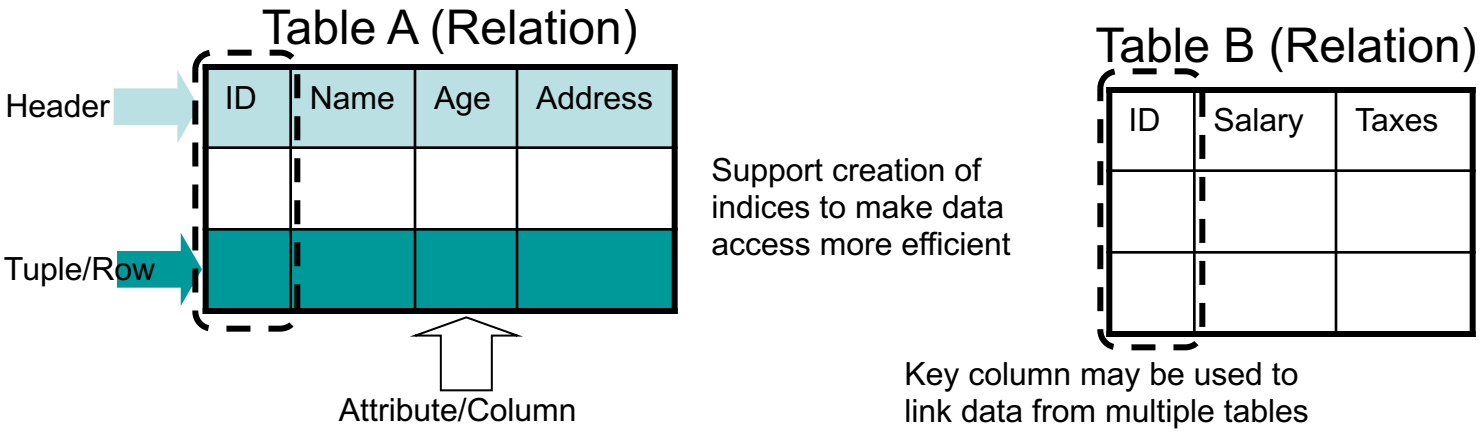
- Batch paradigm, pull model
- Query-driven: submits queries to static data
- Relies on Databases, Data Warehouses



Traditional Computing with Databases

- Databases
  - Software based containers to collect and store information
  - Support retrieval, updates, addition, and removal of data
  - Multiple models: Network, Hierarchical, Relational
- Relational Databases
  - Use a relational model (schema)
    - Provides declarative method to specify data and queries (predicates)
    - Database management takes care of underlying data structures and retrieval mechanisms
  - Most popular types of databases
  - e.g. IBM DB2, Oracle Database, Microsoft SQL Server, PostgreSQL, MySQL

Traditional Computing with Databases



- Queries against database specified in Structured Query Language (SQL)
- Operations
  - Union – combine tuples of two relations (remove duplicates)
  - Intersection – find common tuples
  - Difference – set difference
  - Cartesian product
  - Select – subset of tuples from relation
  - Project – Select with duplicates removed (group by, distinct)
  - Join – tuples from two relations are merged based on common attribute (key)
  - Division – inverse of cartesian product (use tuples from one relation to partition tuples in another relation)
- Stored Procedures: Executable code stored in the database to customize operations

Traditional Computing with Databases

- Relational databases used widely
  - Financial records, Manufacturing, Personnel
- Several analytic tools developed
  - Online Transaction Processing (OLTP)
    - Manage transaction oriented applications
  - Online Analytical Processing (OLAP)
    - Extensions with multi-dimensional databases and efficient data aggregates to answer complex queries efficiently
  - Data Mining Tools
    - SAS, SPSS, Weka
  - Visualization and Report Generation Tools
    - Cognos

NoSQL Stores: Semi-Structured Data

- Provide data storage outside tabular form
- Scale better than SQL Databases
  - Sacrifice Atomicity Consistency Isolation Durability (ACID) guarantees for eventual-consistency
- Can store semi-structured data
- Often provide SQL-like languages for query
- Several Types
  - Column: Accumulo, Hbase, Cassandra
  - Document: CouchDB, Cosmos DB, MongoDB,
  - Key Value: Redis, BerkeleyDB, Zookeeper
  - Graph: Apache Graph, Neo4J

Databases and Streaming Data (?)

- Data Ingest
  - Cannot easily handle unstructured data types and proprietary formats, including audio, video, multimedia, graph structures etc.
  - Notion of time and data order not natural
- Data Analysis
  - Mining tools available, but hard to extend and customize
  - Limited support for long-running queries
- Performance
  - Cannot keep up with data rates, analysis requirements, latency
  - Cannot scale to store the potentially infinite amount of stream data
- Ease of Use
  - Requires using SQL or C++/Java extensions to invoke SQL
- Solutions based on traditional computing face severe bottlenecks
  - Applications often run hours to days behind data
  - Majority of collected data is never analyzed

From Databases towards Streaming

- Active databases (e.g. triggers in commercial Databases)
- ECA (event-condition-action) rules
  - capture events, the conditions surrounding these events, and the actions to be triggered when the conditions are met
  - Implemented as SQL triggers
- Closed
  - Only operate on events within database
  - Ode, HiPac
- Open
  - External event sources are allowed
  - Samos, Snoop
- Disadvantages
  - Data always needs to be stored before processing

Continuous Query Systems

- Continuous queries
  - Standing queries that run until explicit termination
  - Trigger, Query, Stop Condition
  - "monitor the price of 10 megapixel (MP) digital cameras in next two months and notify me when one with a price less than $100 becomes available"
- Evaluated continuously over changing data
  - XML based file input
- Push-based model (answers sent to user)
  - Rule based analytics
- NiagaraCQ, OpenCQ
- Precursors to Stream Processing
  - Run data through queries ☺

Publish/Subscribe Systems

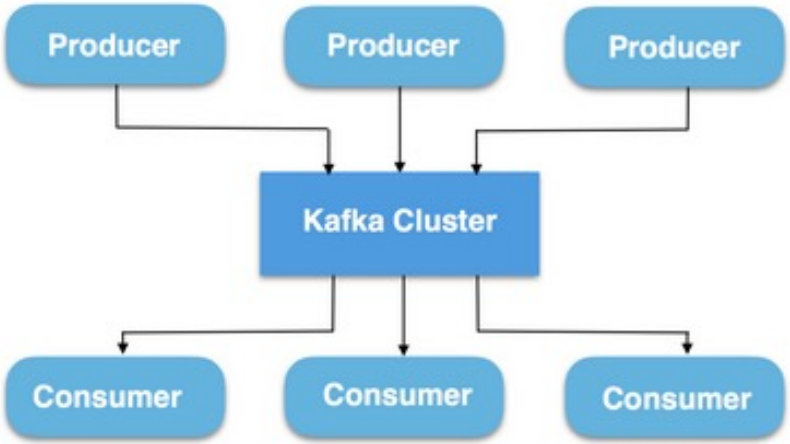
- Asynchronous messaging systems
  - Senders publish messages (characterized into classes) without sending to specific receivers
  - Receivers subscribe to different classes to receive messages of interest
  - Intermediate message broker forwards messages
- Decoupling of publisher and subscriber
  - Improved scalability
  - Dynamic network topology
- Message oriented middleware solutions
  - Message queue, Java Message Service
- Message filtering
  - Topic based: Logical channels, all messages in topic received
  - Content based: Message filtered based on attribute
  - Hybrid possible

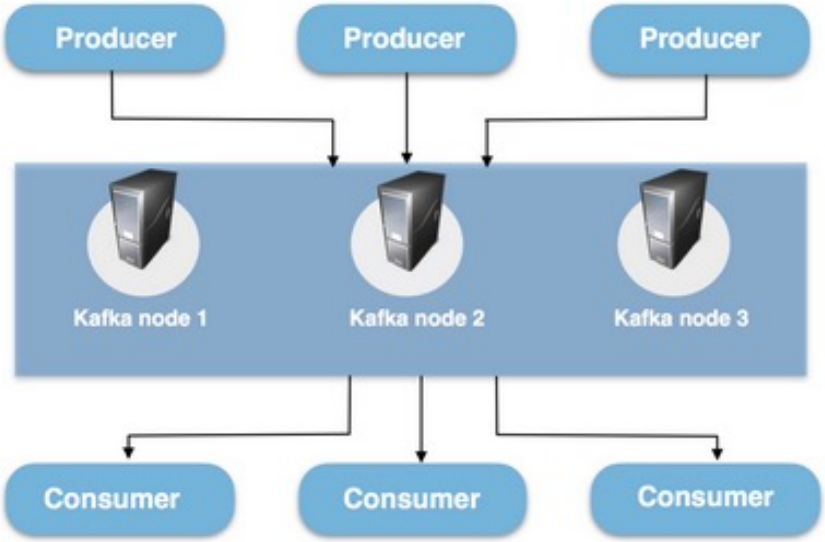
Publish/Subscribe Systems

- Systems
  - Gryphon, Siena - U. Colorado, Padres - U. Toronto
- Extensions into Enterprise Service Bus
  - SOAP, Web-services etc.
- Disadvantages
  - Cannot guarantee delivery of data
  - Instabilities in throughput
    - Bursts of data followed by long periods of silence
  - Scalability issues
  - Slowdowns as number of applications increase
  - Lack of security

Publish/Subscribe Systems

- Apache Kafka: distributed commit log service that functions much like a publish/subscribe messaging system
  - Improved throughput,
  - built-in partitioning,
  - replication and fault tolerance



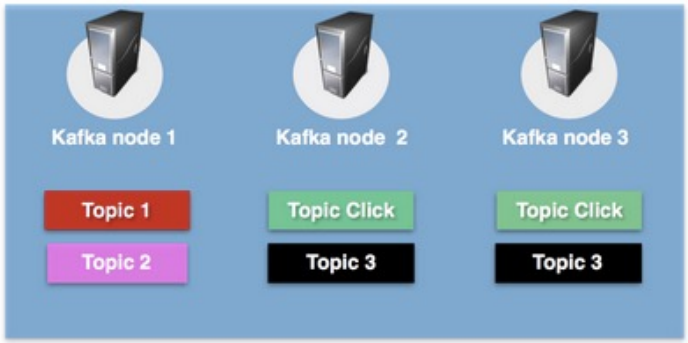


https://www.cloudkarafka.com/blog/2016-11-30-part1-kafka-for-beginners-what-is-apache-kafka.html

Publish/Subscribe Systems: Kafka

Kafka Topic

A **Topic** is a category/feed name to which messages are stored and published. Messages are byte arrays that can store any object in any format. As said before, all Kafka messages are organized into topics. If you wish to send a message you send it to a specific topic and if you wish to read a message you read it from a specific topic. Producer applications write data to topics and consumer applications read from topics. Messages published to the cluster will stay in the cluster until a configurable retention period has passed by. Kafka retains all messages for a set amount of time, and therefore, consumers are responsible to track their location.

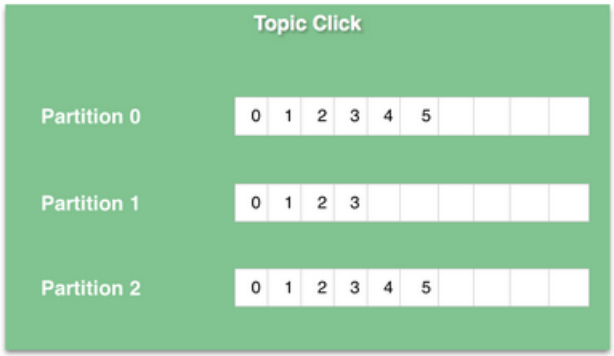


https://www.cloudkarafka.com/blog/2016-11-30-part1-kafka-for-beginners-what-is-apache-kafka.html

Publish/Subscribe Systems: Kafka

Kafka topic partition

Kafka topics are divided into a number of partitions, which contains messages in an unchangeable sequence. Each message in a partition is assigned and identified by its unique **offset.** A topic can also have multiple partition logs like the *click-topic* has in the image to the right. This allows for multiple consumers to read from a topic in parallel.



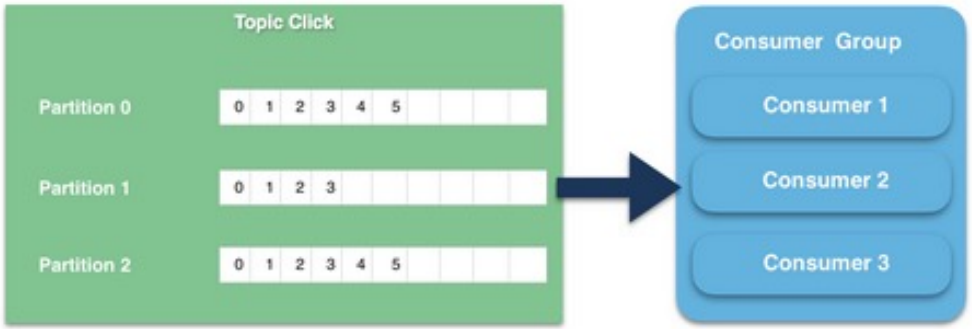
In Kafka, replication is implemented at the partition level. The redundant unit of a topic partition is called a replica. Each partition usually has one or more replicas meaning that partitions contain messages that are replicated over a few Kafka brokers in the cluster. As we can see in the pictures - the *click-topic* is replicated to Kafka node 2 and Kafka node 3.

https://www.cloudkarafka.com/blog/2016-11-30-part1-kafka-for-beginners-what-is-apache-kafka.html

Publish/Subscribe Systems: Kafka

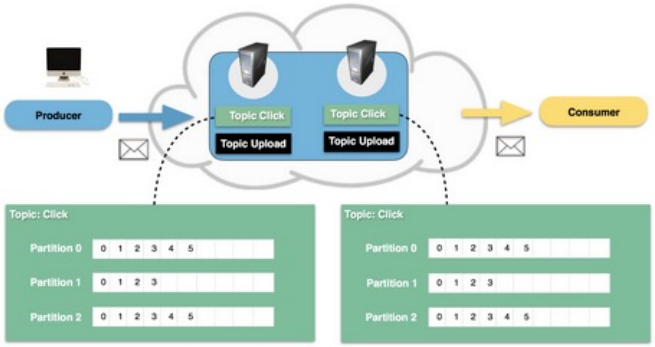
Consumers and consumer groups

Consumers can read messages starting from a specific offset and are allowed to read from any offset point they choose. This allows consumers to join the cluster at any point in time.



https://www.cloudkarafka.com/blog/2016-11-30-part1-kafka-for-beginners-what-is-apache-kafka.html

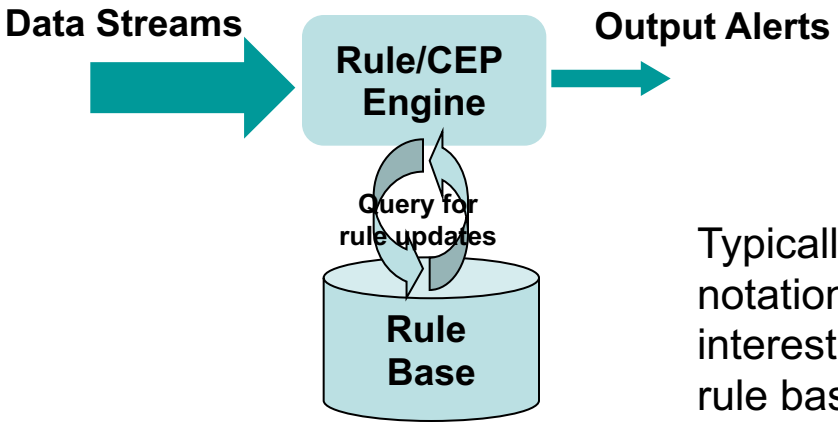
Publish/Subscribe Systems: Kafka



1. *A user with user-id 0 clicks on a button on the website.*
2. *The web application publishes a message to partition 0 in topic "click".*
3. *The message is appended to its commit log and the message offset is incremented.*
4. *The consumer can pull messages from the click-topic and show monitoring usage in real-time, or it can replay previously consumed messages by setting the offset to an earlier one.*

https://www.cloudkarafka.com/blog/2016-11-30-part1-kafka-for-beginners-what-is-apache-kafka.html

Complex Event Processing Engines



Typically accept condition/action pairs using “if-then” notation and watch input stream for condition of interest. Enforce collection of rules/patterns stored in rule base.

Rule firing leads to alerts, as well as other rules becoming active

CEP Systems: Rules/Patterns

- Temporal sequences
  - event *A* following event *B*
- Negation
  - event *A* not appearing in a sequence
- Kleene closure
  - event *A* appearing repeatedly a number of times
  - Regular expression based search
- Support for time windows

Example Rules

Intrusion Detection Rules

GeneralBruteForceTwentyInTwoUniqDST: Alert anytime that a single *destination IP* targeted with **20** or more events in "*generalbruteforce"* group *within a* **2** *minute window*

HttpScanFiveInThreeUniqSRCBootstrapRule: Alert anytime that any single *source ip* produces more than **5** *unique event names* in the "*httpscan" group* within a **3** *minute window*

NotificationNeededForP2PRule: Alert each time *an event in "p2p" group* is detected

- Expert defined rules on data streams
- Simple rules
  - Comprehensibility and Ease of Construction
  - User Validation
- Typical Analytics
  - Capture event/group arrival rates, temporal relationships, filter and composite conditions

Telco Mediation Rules

```
RULE TR33_A{
  (CALL_RECORD_TYPE==1) || (CALL_RECORD_TYPE==3) ||
  (CALL_RECORD_TYPE==5) || (CALL_RECORD_TYPE == 15)
=>
  CALL_DIRECTION = "O";
}
RULE TR33_B{
  (CALL_RECORD_TYPE==2) || (CALL_RECORD_TYPE==4) ||
  (CALL_RECORD_TYPE==7) || (CALL_RECORD_TYPE == 14)
=>
  CALL_DIRECTION = "I";
}
RULE TR33_C{
  (CALL_RECORD_TYPE == 0)
=>
  CALL_DIRECTION = "T";
}
RULE TR33_D{
  VAR X = lookup(incomingRoute, "DIM_MCS_ROUTE","ROUTE")
  (CALL_RECORD_TYPE == 0) &&  (incomingRoute != NULL) && (X)
=>
  CALL_DIRECTION = "R";
}
RULE TR33_E{
  VAR X = lookup(incomingRoute,"DIM_MCS_ROUTE","ROUTE")
  VAR Y = lookup(outgoingRoute,"DIM_MCS_ROUTE","ROUTE")
  (CALL_RECORD_TYPE == 0) && (incomingRoute != NULL) &&
  (outgoingRoute != NULL) && X && Y
=>
  CALL_DIRECTION = "T"; }
```

CEP Engines for Streaming Data

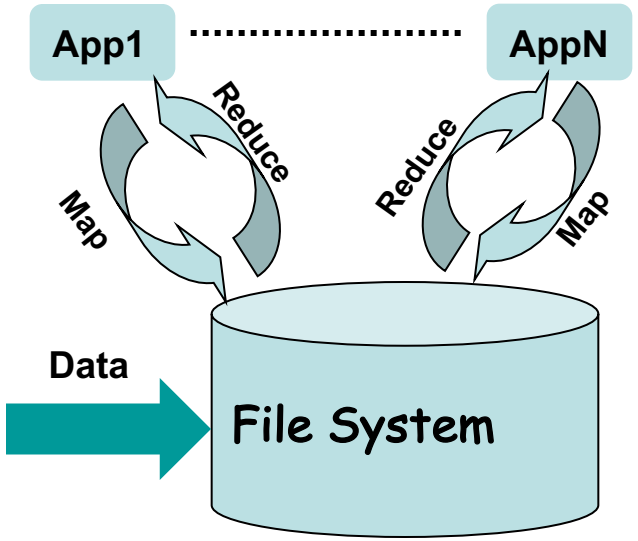
- Several popular engines
  - IBM ODM, Oracle CEP, Tibco Business Events
- Data Ingest
  - Cannot easily handle unstructured data types and proprietary formats, including audio, video, multimedia, graph structures etc.
  - Cannot easily handle data from heterogeneous and distributed sources
- Data Analysis
  - Limited to simple rule languages, need to be extended to support stream processing requirements
- Performance
  - Cannot handle very large data rates and low latency
- Ease of use
  - Low barrier to usage – several UIs available

Other Systems

- Supervisory Control And Data Acquisition (SCADA) systems
- Extract/Transform/Load (ETL) systems

Map-Reduce Framework

Applications partitioned into Map/Reduce and implemented on distributed compute infrastructure

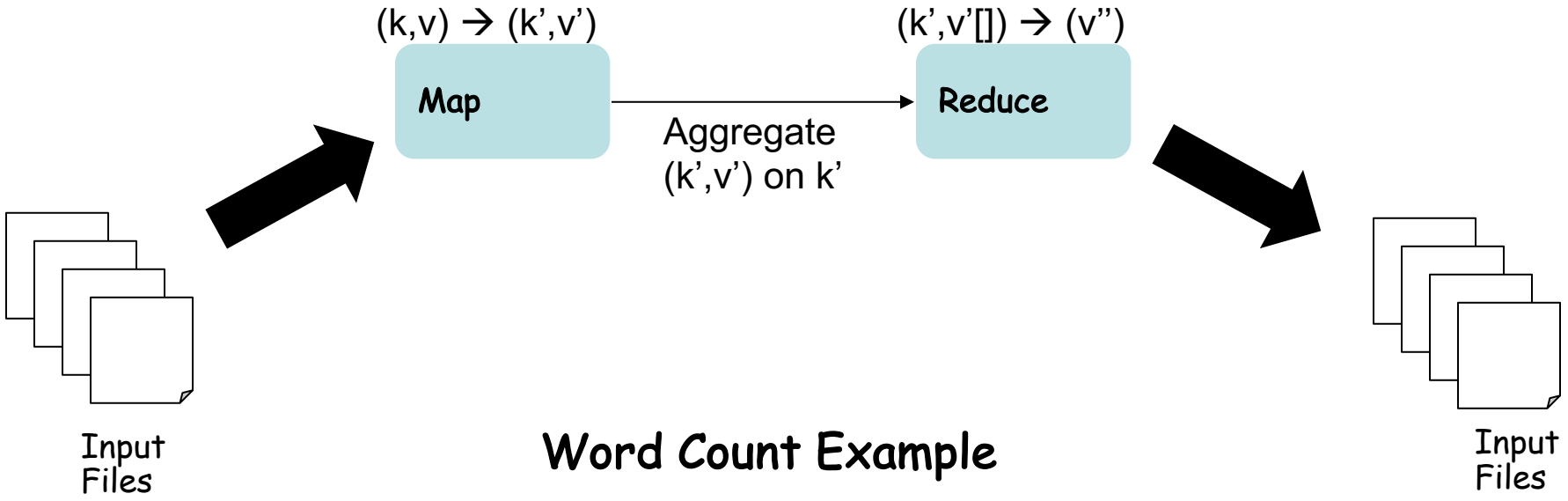


Google FS, Hadoop FS, IBM GPFS, Parascale FS, MapR

- Programming model (from Google) for large-scale distributed 'batch' processing
  - Powerful paradigm but restricted to certain classes of problems
  - Extensible for different applications
- Execution environment
  - Exploits parallelism (pipelined and data)
  - Resilience to failures and jitter

Inspired by Map/Reduce paradigm in functional programming (e.g. Lisp).

Map-Reduce Programming Model

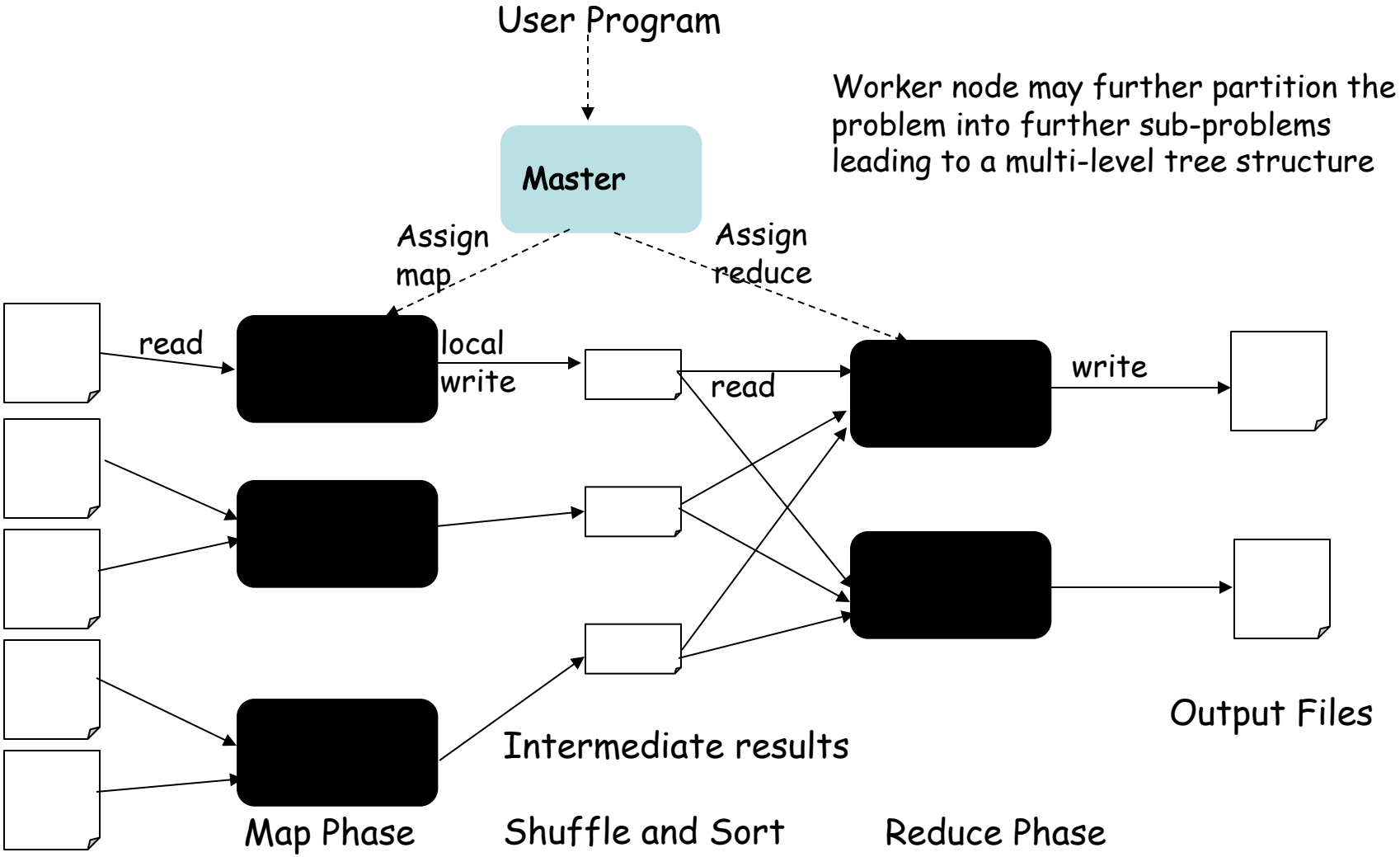


Word Count Example

```
void map(String name, String
document):
  // name: document name
  // document: document contents
  for each word w in document:
    EmitIntermediate(w, "1");
```

```
void reduce(String word, Iterator
partialCounts):
  // word: a word
  // partialCounts: a list of
aggregated partial counts
  int sum = 0;
  for each pc in partialCounts:
    sum += ParseInt(pc);
  Emit(word, AsString(sum));
```

Map-Reduce Execution Model



Using Map-Reduce Programming Model

- Input Reader
  - Divide input into appropriate size splits and generate kev/value pairs
- Mapper
- Partition
  - Allocate each mapper result to appropriate reducer
- Compare
  - Used to sort/shuffle the intermediate results for improved efficiency
- Reducer
- Output Writer
  - Write results to output file system

- Google's use of Map-Reduce*
  - Clustering of news items for Google News, Processing of satellite imagery, Statistical machine translation, Large-scale machine learning

*J. Zhao, J. Pjesivac-Grbovic, "MapReduce The Programming Model and Practice", SigMetrics 2009.

Hadoop

- Apache Implementation of the Map-Reduce Framework
  - Open source Java implementation
  - Yahoo! significant contributor
- Hadoop File System
  - Distributed, Scalable, Portable Java based filesystem
- Job Tracker
  - Partitions work out to individual task trackers
  - Monitors task trackers via heartbeat
- Task Tracker
  - Spawns processes for the job
- Scheduler
  - FIFO, Fair (Facebook), Capacity (Yahoo!)
- Fault Tolerance
  - Limited fault tolerance

Machine Learning Map-Reduce: Mahout

- Mahout Open Source Learning Library
  - Apache
- Supports development of scalable machine learning libraries on Hadoop
- User-contributed (somewhat large community)
- Two modes of operation
  - single-mode and map-reduce
- Command-line based analogous to Hadoop

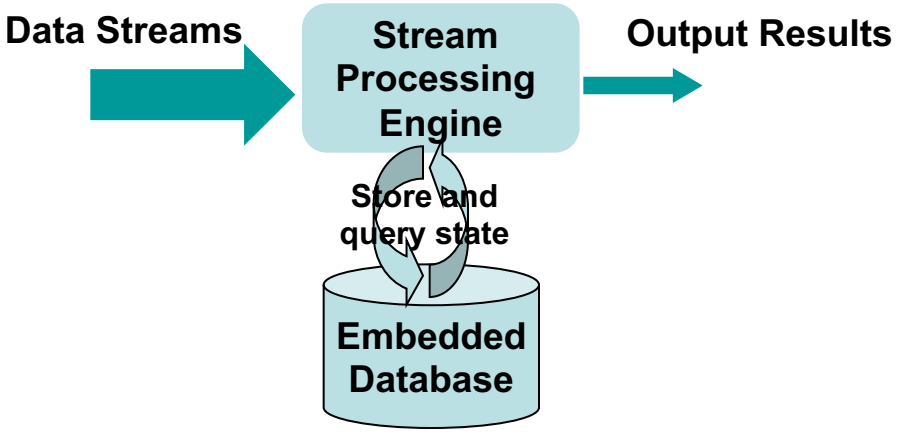
```
$MAHOUT_HOME/bin/mahout kmeans –I <in> -o <out> -k 50
```

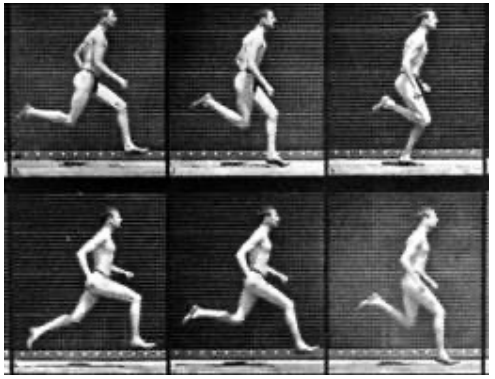
- API:

```
KMeansDriver.runjob(Pathinput,Pathoutput…)
```

- Supported Algorithms
  - Classification (Logistic Regression, Naïve Bayes, C-Bayes, Random Forest)
  - Clustering (Kmeans, fuzzy kmeans, spectral clustering, Canopy, Fuzzy K-means, dirichlet)
  - Frequent Pattern Mining (Pattern and itemset mining)
  - Dimensionality Reduction (SVD)
  - High performance collection (opted from CERN Colt library)
  - Math library

Stream Processing Systems



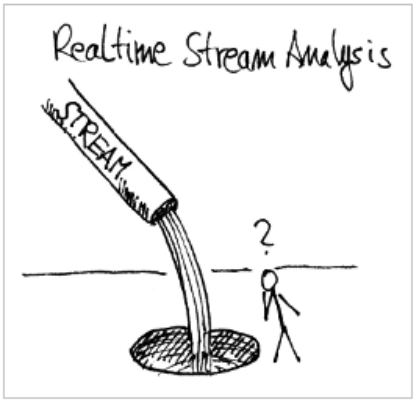


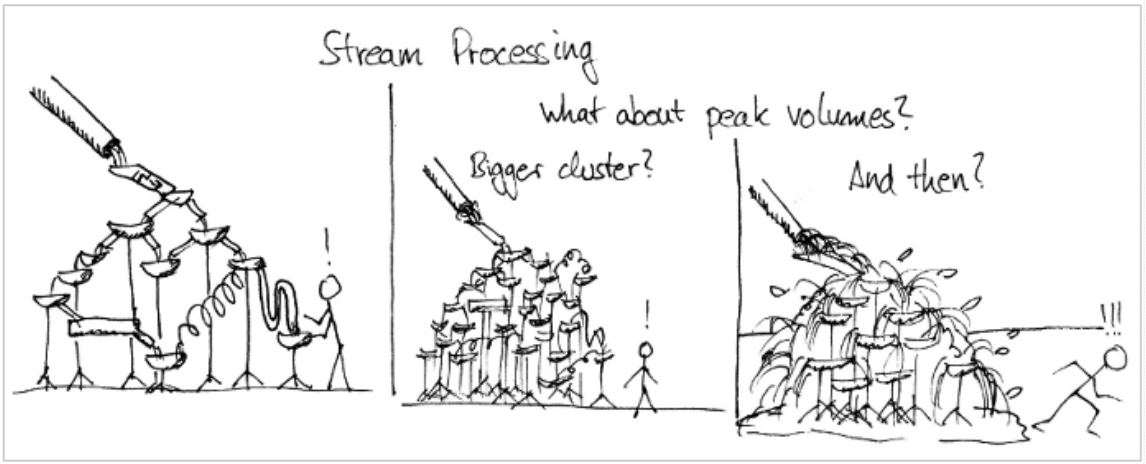
Real time analysis of data-in-motion

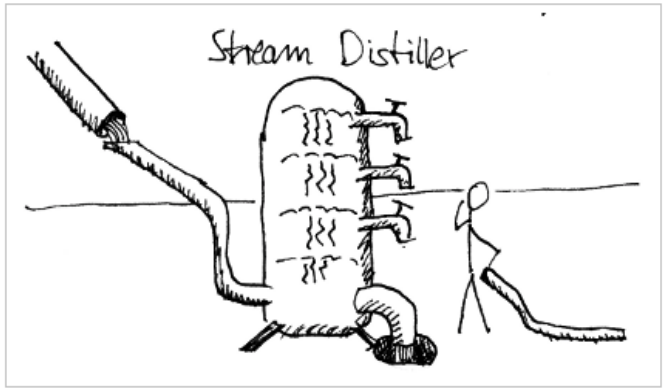
Perform arbitrarily complex processing on data as it streams through. Use embedded database to store data summaries, history, state information etc.

Unlike DBMS long-running queries through which data flows to get filtered appropriately

Databases versus Stream Processing

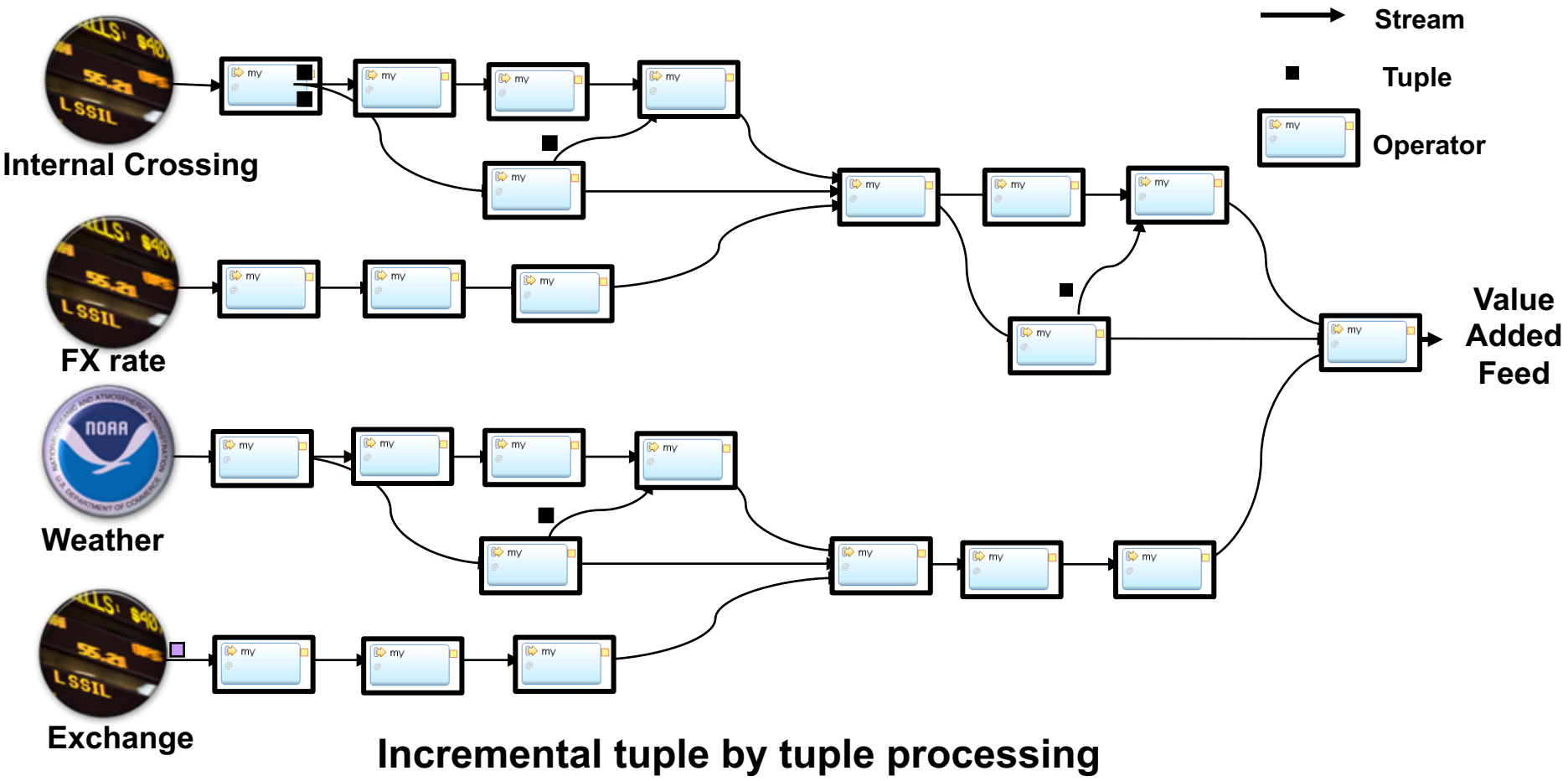




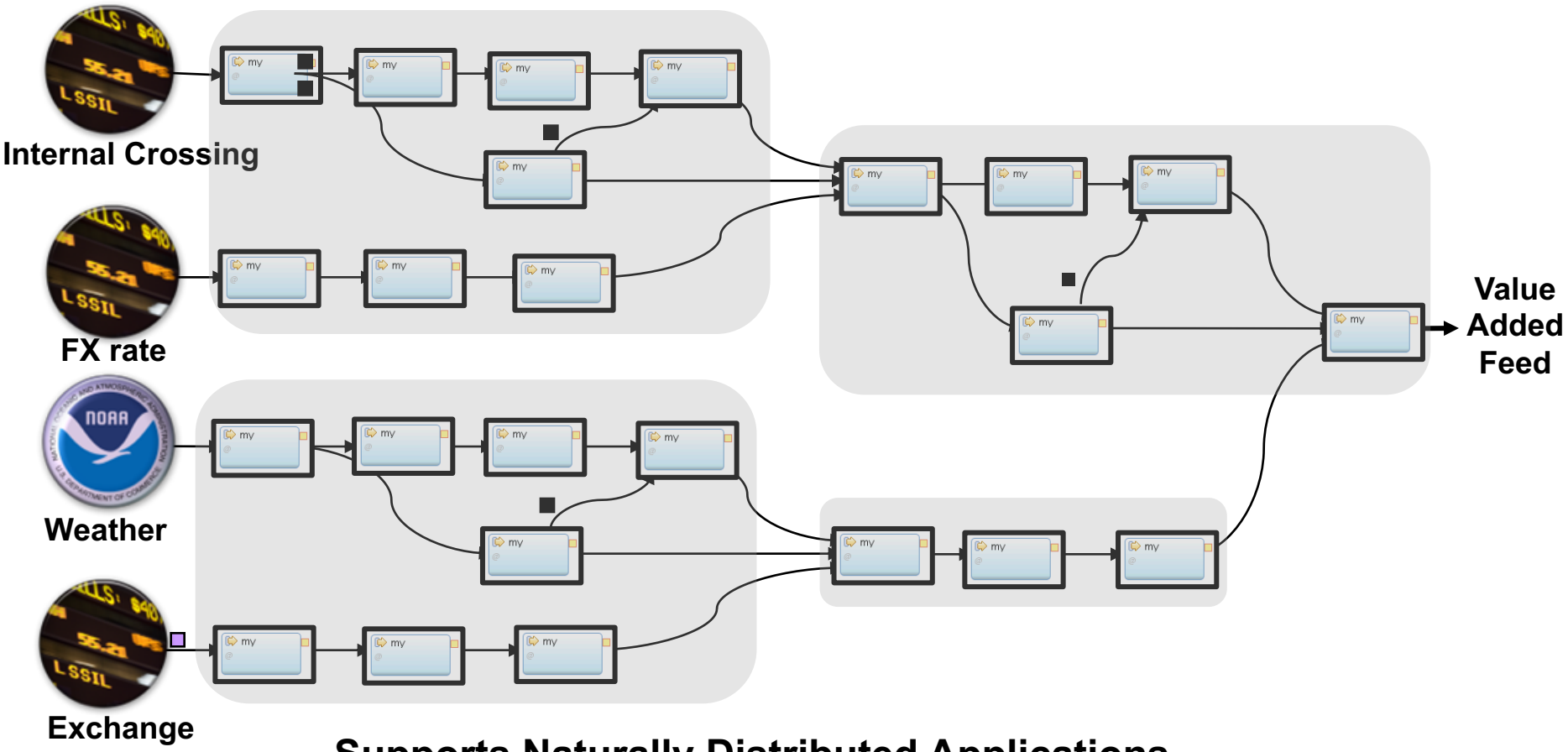


Thanks to B. Gedik

Introduction to Stream Processing



Introduction to Stream Processing



Stream Processing System Principles

- Keep Data Moving
  - Process message in stream without requiring storage
  - Use asynchronous, non-polling based communication
- Handle Stream Imperfections
  - Out of order, missing, delayed, noisy data
- Generate predictable outcomes
  - Reproducible results
- Handle both streaming as well as stored data
  - Efficient storage, access and modification of state information
- Support data safety and availability
  - High availability and resilience to loss – when needed
- Partition and scale applications
  - Distribute processing to achieve incremental scaling
- Process and respond instantaneously
  - Optimized and high performance execution engine
- Query using high level languages
  - Do not rely purely on C++/Java to minimize development time

Stream Processing Systems

- Programming Model
  - Flowgraph composition
  - Development Environment
- Runtime
  - Distribution
  - Data transport
  - Scheduling
  - Fault Tolerance
- Toolkits
  - Connectors
  - Analytics
  - Stream Relational

Stream Processing Systems

- Academic Systems (now retired)
  - TelegraphCQ – Berkeley
  - Aurora & Borealis – Brown
  - STREAM – Stanford
- Industrial Prototypes
  - Gigascope – AT&T
  - System S – IBM Research
- Open Source Systems
  - S4 – Yahoo S4!
  - Storm – Twitter, Apache
  - Apache Flink
  - **Apache Spark Streaming**
  - **Apache Beam**
- Commercial Systems
  - StreamBase – StreamBase Systems
  - Amazon Kinesis
  - IBM Streams
  - Google Dataflow

Stream and Signal Processing

- Sensing and Aggregation of Data from Diverse Sources
  - Task Driven Sensing
  - Robust and Error Resilient Data Gathering
    - Distributed compression, data reduction, processing
- Managing confidence, uncertainty and noise
  - Missing samples, delayed samples, non-time aligned
  - Algorithms and system services to support analytics/application
- Resource Constrained Online Analytics
  - Resource adaptive filtering, tracking, feature extraction, compression
  - Complexity scalable mining and online learning
  - Distributed analysis

Going Forward

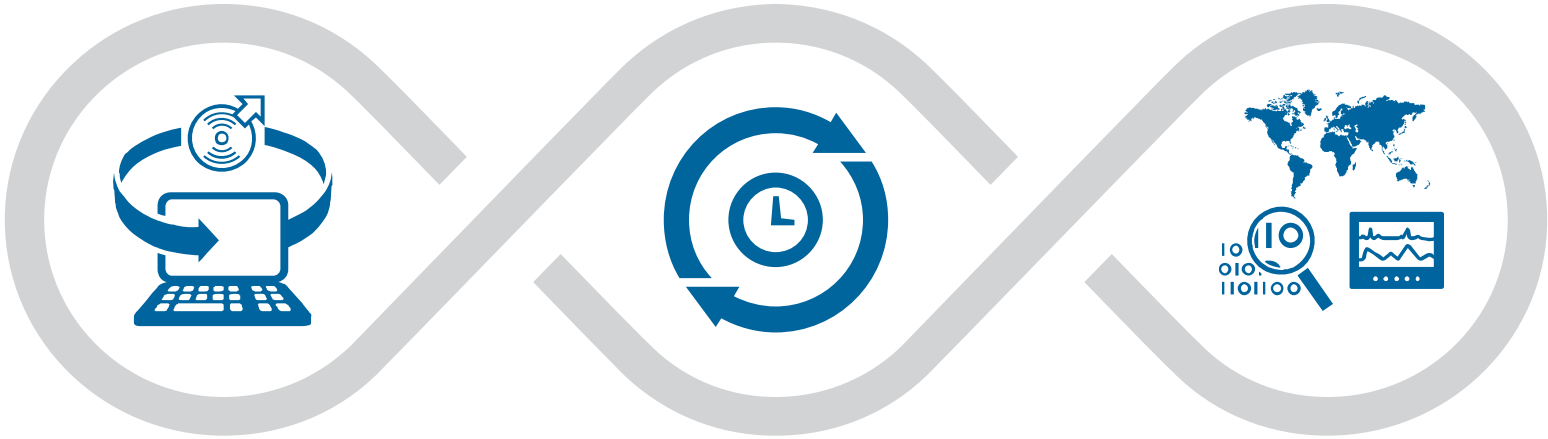
- Examples of two Stream Processing Systems
  - Apache Spark Streaming, Apache Beam
- Fundamentals of stream processing systems
  - Plumbing and architecture of systems
  - Programming stream processing applications
  - Design principles of application development
- Streaming Data Analysis
  - Data Preprocessing, Data Analysis
- Focus on exploring space of stream processing
  - Student seminars and projects

Anatomy of A Stream Processing System

Integrated Development Environment

Scale-Out Runtime

Analytic Toolkits

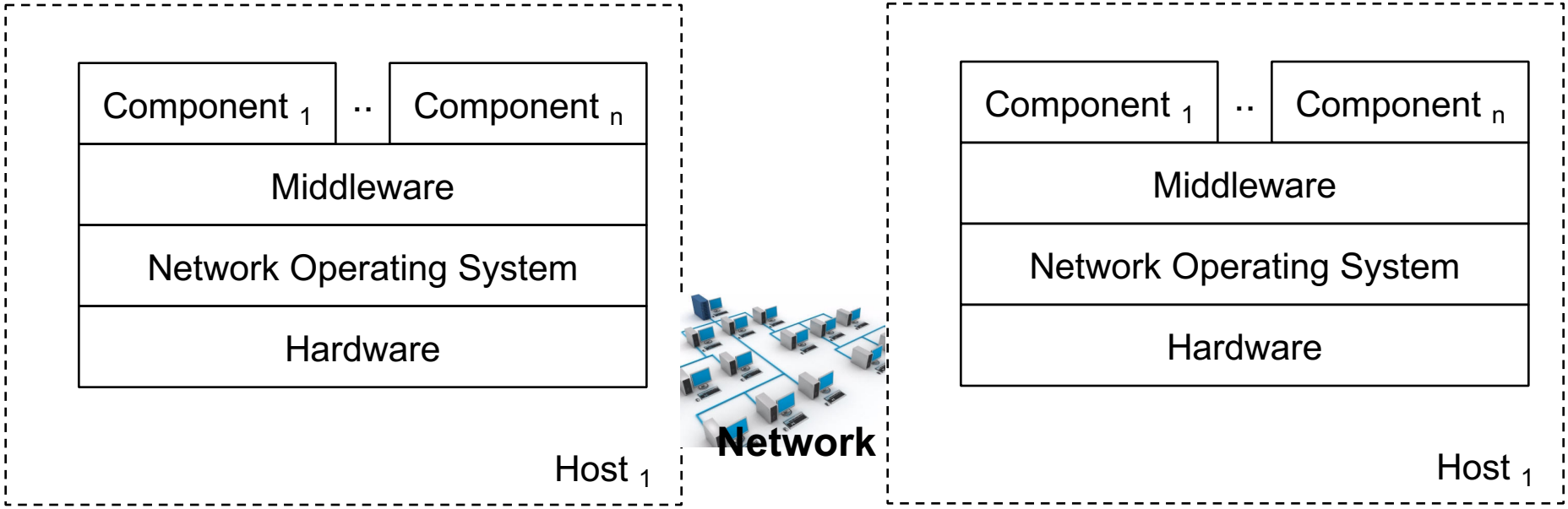


Development and Management

Flexibility and Scalability

Functional and Optimized

Middleware



- *Middleware*: computer software that provides services to software applications beyond those available from the operating system
- *Stream Processing Middleware*: distributed system that provides an execution substrate for stream processing applications

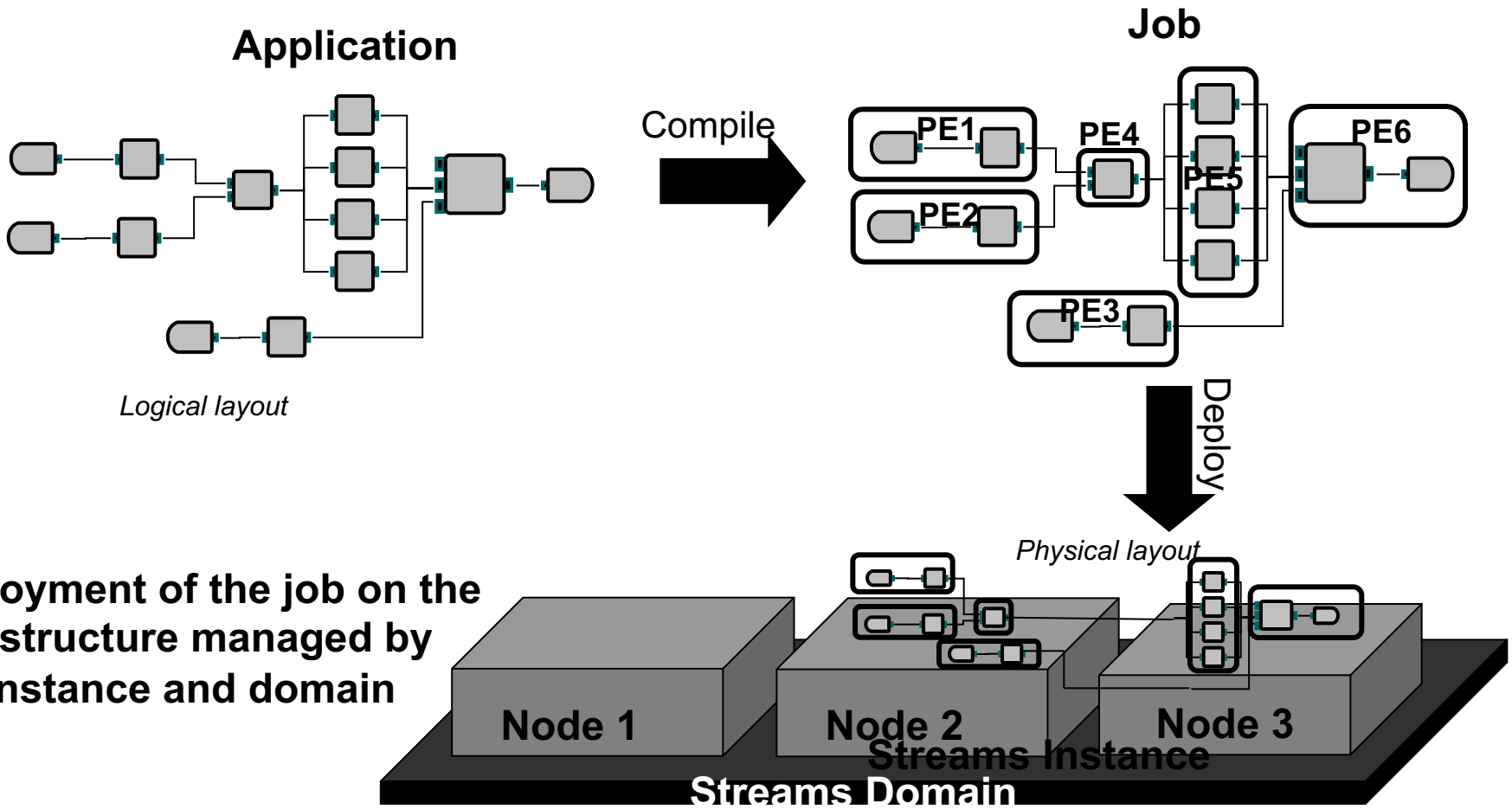
Middleware Support

- Access transparency
  - Local and remote components accessible via same operations
- Location transparency
  - Components locatable via name, independent of location
- Concurrency transparency
  - Components/objects can be run in parallel
- Migration transparency
  - Allows movement of components without affecting other components
- Replication transparency
  - Allows multiple instances of objects for improved reliability (mirrored web pages)
- Failure transparency
  - Components designed while accounting for failure of other services

Conceptual Entities of Interest

- *Domain*
  - Grouping of resources for common management
- *Instance* of a stream processing middleware
  - runs a number of applications/jobs within domain
- *User*
  - Application developer, application analyst, or system administrator
- *Application*
  - A logical data flow graph
- *Job*
  - Instantiated version of the application
- *Processing Element (PE)*
  - Corresponds to an OS process
  - Execution container for a sub-graph

Conceptual Entities of Interest



Deployment of the job on the infrastructure managed by the instance and domain

Stream Processing Middleware Services

- Distribution
  - Plumbing related to moving processes around
- Resource management
  - Placement, scheduling, load shedding
- Job management
  - Starting/stopping/editing jobs, handling dynamic connections
- System monitoring
  - Health, state, performance of the system
- Security
- Logging

- Data Transport
  - Inter-PE, intra-PE, intra-host, inter-host
- Debugging
- Visualization

Instance Components

Component Name	Acronym	Executable Name
Streams Application Manager	SAM	`streams-sam`
Streams Resource Manager	SRM	`streams-srm`
Scheduler	SCH	`streams-sch`
Name Service	NS	`dnameserver`
Authentication and Authorization Service	AAS	`streams-aas`
Streams Web Server	SWS	`streams-sws`

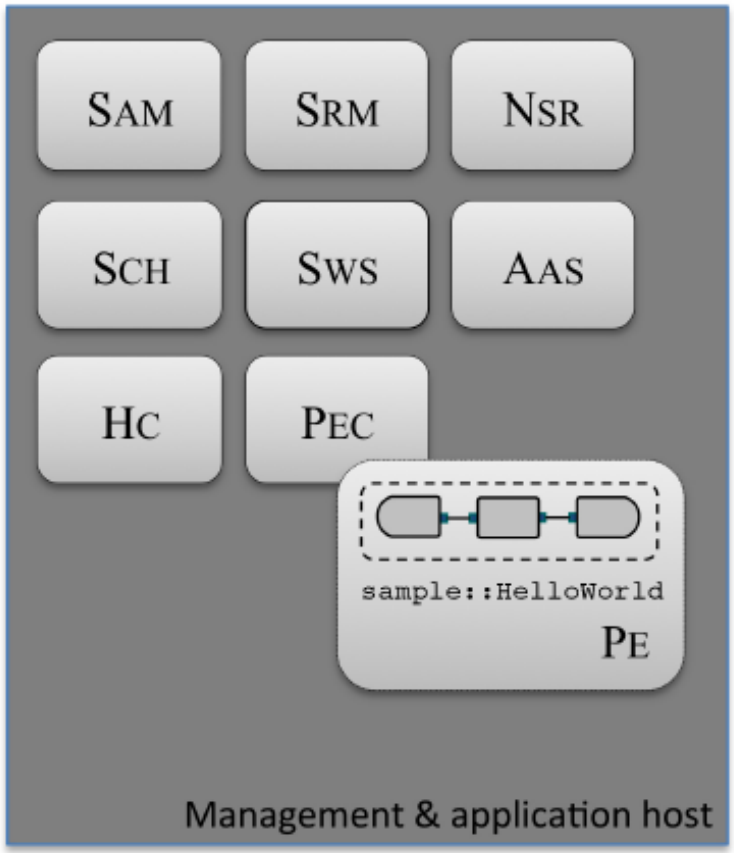
Domain Services

Table 8.1 InfoSphere Streams centralized components.

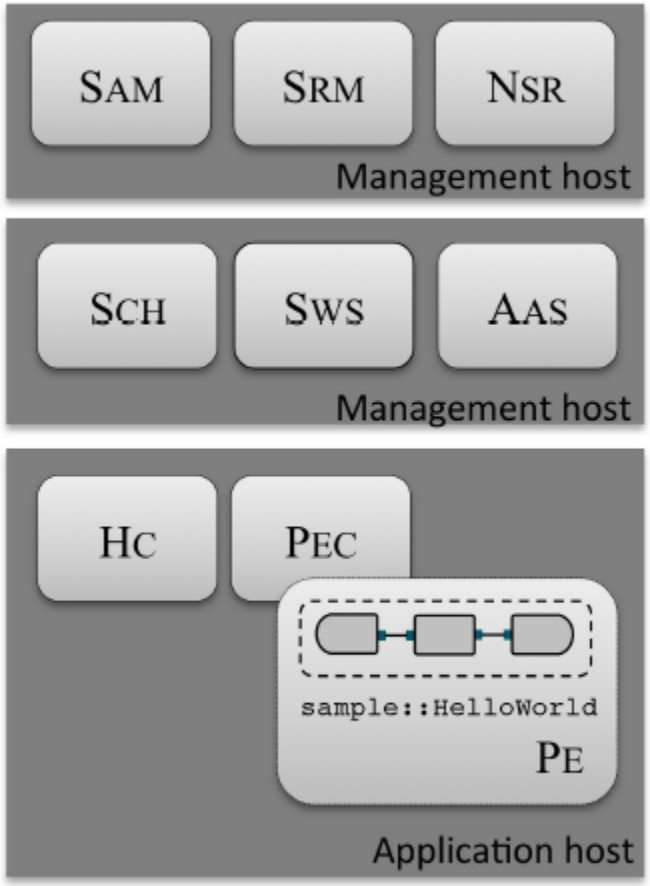
Component Name	Acronym	Executable Name
Host Controller	HC	`streams-hc`
Processing Element Container	PEC	`streams-pec`

Table 8.2 InfoSphere Streams distributed components.

Middleware Components



Single Node Instance



Multiple Node Instance

Application and Job Management

- A user submits an application to the runtime instance
- Runtime checks security credentials to ensure a job can be instantiated by the user for this instance
- A job with an id is created from the application
- The scheduler is tasked with placing the PEs (typically formed at compile-time) on the hosts of the instance
  - Considering current load of the system
  - Considering the placement constraints
- The PEs are then moved to the hosts and instantiated there
- The connections between the PEs are established via the transport
- Additional workflows:
  - Job cancellation
  - Dynamic connection management
  - PE relocations, restarts

Resource Management

- *Resource manager* needs to track
  - Resource health
    - Is the host alive?
  - Resource amounts
    - How much CPU is available on the host?
- Health is often tracked using liveness pings
  - Check if a resources is alive by sending a periodic ping
  - If not results for a long time, assume its dead
- Resource amounts are tracked using metrics
  - Collect usage metrics periodically to track availability

Application Monitoring

- Application health
  - Is the PE healthy?
    - Yes, no, partially
  - What state is the PE in
    - Initialization, connection establishment, running, etc.
- Application performance
  - Throughput
  - Backpressure (queue sizes or blocking time)
  - Latency (very application specific)
- Custom metrics

Scheduling

- Compile time and Runtime Scheduling
- There could be a few different goals
  - Load balancing
    - Hosts have similar load on them
    - This is more important when the system is highly loaded
  - Application performance
    - Throughput is typically a major concern
- Some difficult issues:
  - Input rates are not known
    - Can predict based on history
    - Seasonality is typical
  - Operator costs/selectivities are not known
    - Profiling runs to measure operator characteristics

Scheduling

- Provide placement
  - Given job/application constraints
    - `partitionColocation, partitionExlocation, partitionIsolation`
  - Given current state of the compute infrastructure
- Steps
  - Check constraints – to see if placement feasible
  - Solve bipartite matching problem (PEs→hosts)
  - Incrementally learn resource consumption of each PE
  - Load balancing approach: Place in decreasing order of PE complexity on nodes with increasing levels of load
    - i.e. place most expensive PE on least loaded host
  - Periodically re-optimize

Fault Tolerance

- Fault tolerance of the middleware
  - Runtime components save state to database periodically
    - Includes commands, responses with unique IDs
    - Ensures commands executed *at most once* on recovery
  - Service requests use fault tolerant communication with client-server architecture
  - Critical middleware state changes infrequently
    - compared to application state and data
- Application fault-tolerance with data loss
  - Restart failed PEs (with some bound)
  - Relocate PEs on failed hosts
  - Managing state

Application Fault Tolerance

- Active replication
  - Run multiple copies, switch over when you detect failure in one
  - Switch over is a problem for non-deterministic apps
    - Time based windows, arbitrary fan-in
  - Disadvantage: n times resources for n-fold tolerance
    - Given it has little impact on performance, this may be a good choice resource-wise as well, if throughput is the main challenge
- Checkpointing with source data persistence
  - Get a consistent checkpoint, log all source data that is not yet part of a checkpoint
  - Disadvantage
    - costly recovery process
    - operators need to know how to checkpoint their state

Other Services

- Name Service
  - directory for storing references to objects
  - object is location of a component represented as a Common Object Request Broker Architecture (CORBA) Interoperable Object Reference (IOR)
  - Users submit lookup requests for a human-readable symbolic name
  - Can remove and register objects
  - Objects can include data transport endpoints (more later)
- Authentication and Authorization Service
  - provides authorization checks
  - inter-component authentication
  - includes a repository of Access Control Lists (for different entities)
  - Can *entity* perform operation on *object*?
    - E.g. can the user `user` cancel job `2011`

Visualization

Commercial Systems: Streams and Dataflow

- Compile time topology visualization
- Run time topology visualization
  - State
  - Health
  - Metrics
  - Logical to physical mapping
- Data visualization
  - Live charts
  - Dashboards

References and Reading

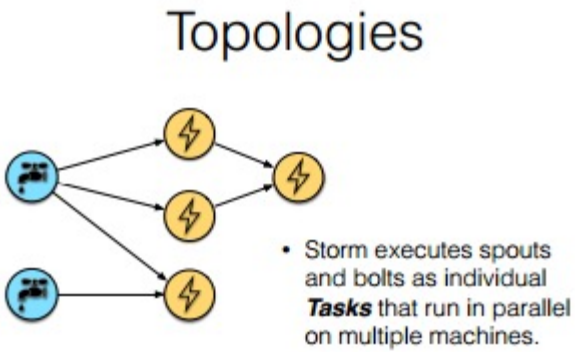
- Chapter 1 of book
- Eight Requirements of Stream Processing: http://www.cs.brown.edu/research/db/publications/8rulesSigRec.pdf
- Aurora: http://www.cs.brown.edu/research/aurora/
- Borealis: http://www.cs.brown.edu/research/borealis/public/
- Telegraph CQ: http://telegraph.cs.berkeley.edu/
- Stanford Stream (inactive): http://infolab.stanford.edu/stream/
- Streamit: http://publications.csail.mit.edu/lcs/pubs/pdf/MIT-LCS-TM-620.pdf
- Streambase: http://www.streambase.com/
- Apama: http://web.progress.com/en/apama/index.html
- IBM System S/InfoSphere Streams: http://www-01.ibm.com/software/data/infosphere/streams/
- Linear Road Benchmark: http://www.cs.brandeis.edu/~linearroad/

Backup

Apache Storm

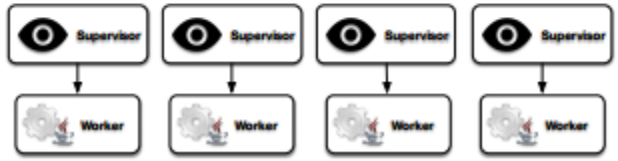
- A tuple: an immutable set of key / value pairs
- Streams: an unbounded sequence of tuples
- Spouts: source of streams..
- Bolts: receive tuples and do stuff – emit other tuples, read / write data store, perform computation

- Routing tuples between tasks:
  - Shuffle (random)
  - LocalOrshuffle
  - FieldGroupings

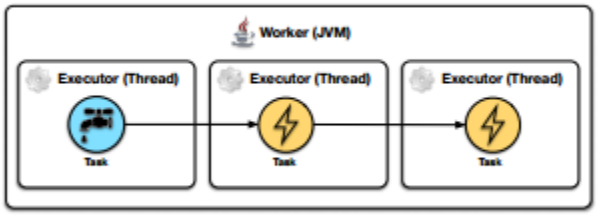


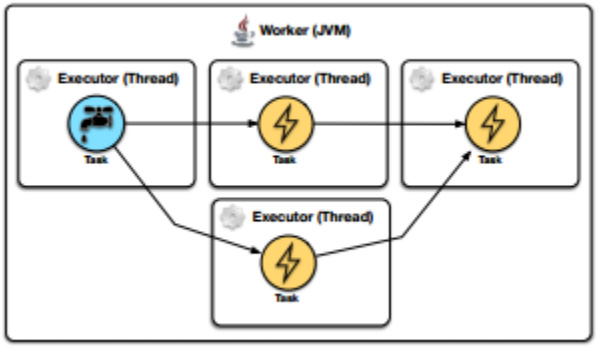
Apache Storm: Topology, Parallelism

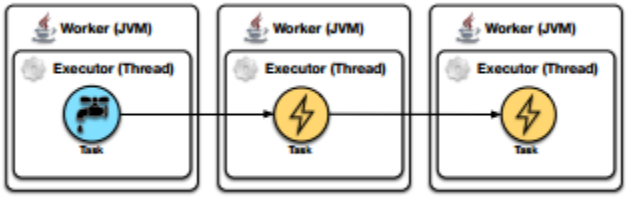
Topology Deployment



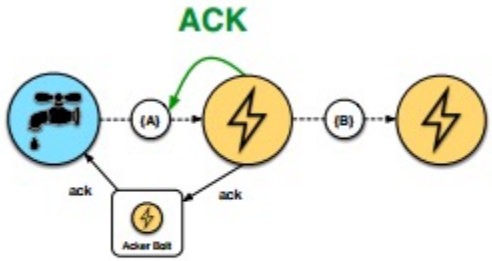
- Built with resilient components
- Parallelism via threads, JVMs





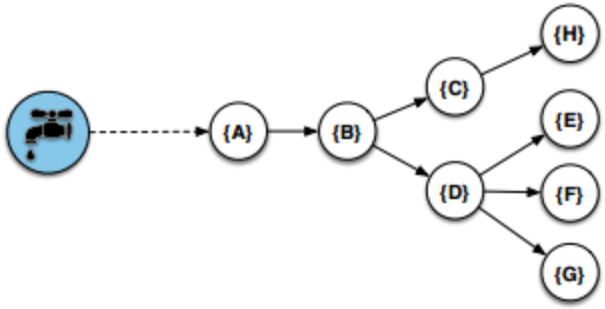


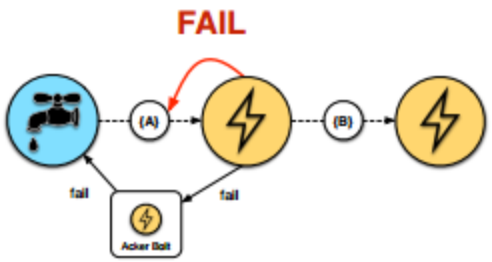
Apache Storm: Reliable Processing



Bolts may emit Tuples ***Anchored*** to one received.
Tuple "B" is a descendant of Tuple "A"

Acks are delivered via a system-level bolt



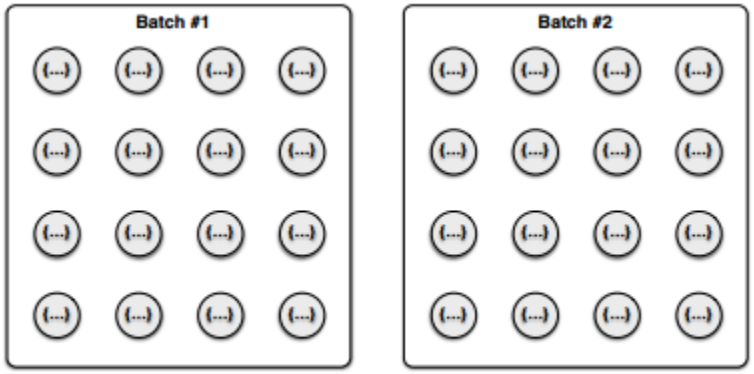


Apache Storm Deployment Models

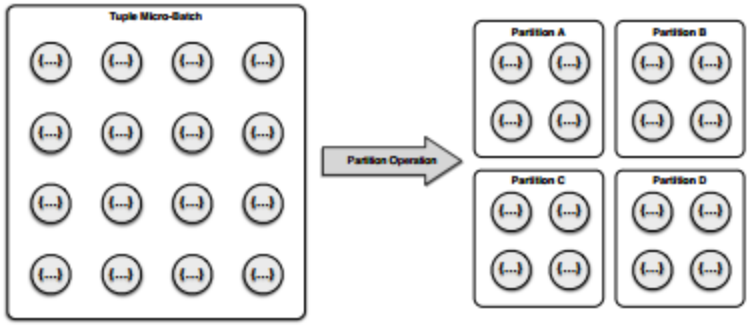
- Core Storm (spouts and Bolts)
  - One a time
  - Low latency
  - Operates on tuple streams

- Trident (streams and operations)
  - Micro-batch
  - Higher throughput
  - Operates on streams of tuple batches and partitions

Apache Storm Trident: Micro-batching

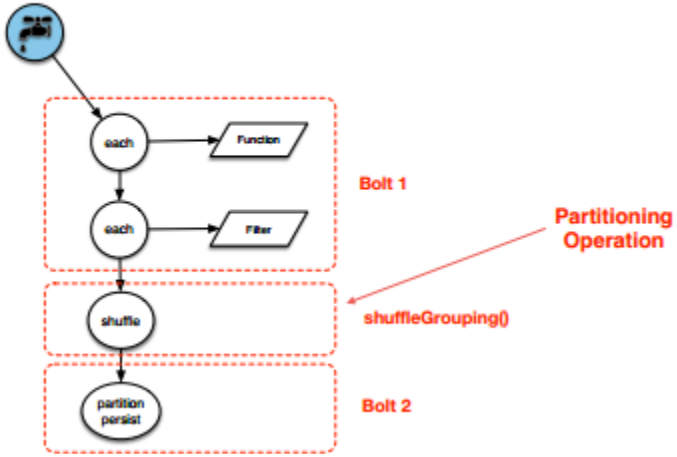


Trident Batches are ***Ordered***

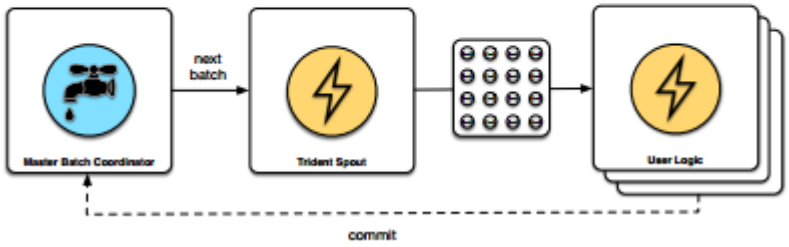


Trident Batches can be ***Partitioned***

Trident Topologies



Trident Batch Coordination



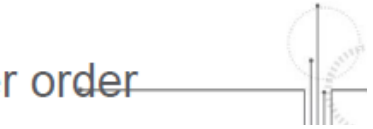
Apache Storm

- In use by several companies

- However, recent development slow
  - Other open source active projects (Spark Streaming)
- More information
  - https://storm.apache.org/

Amazon Kinesis: Core concepts

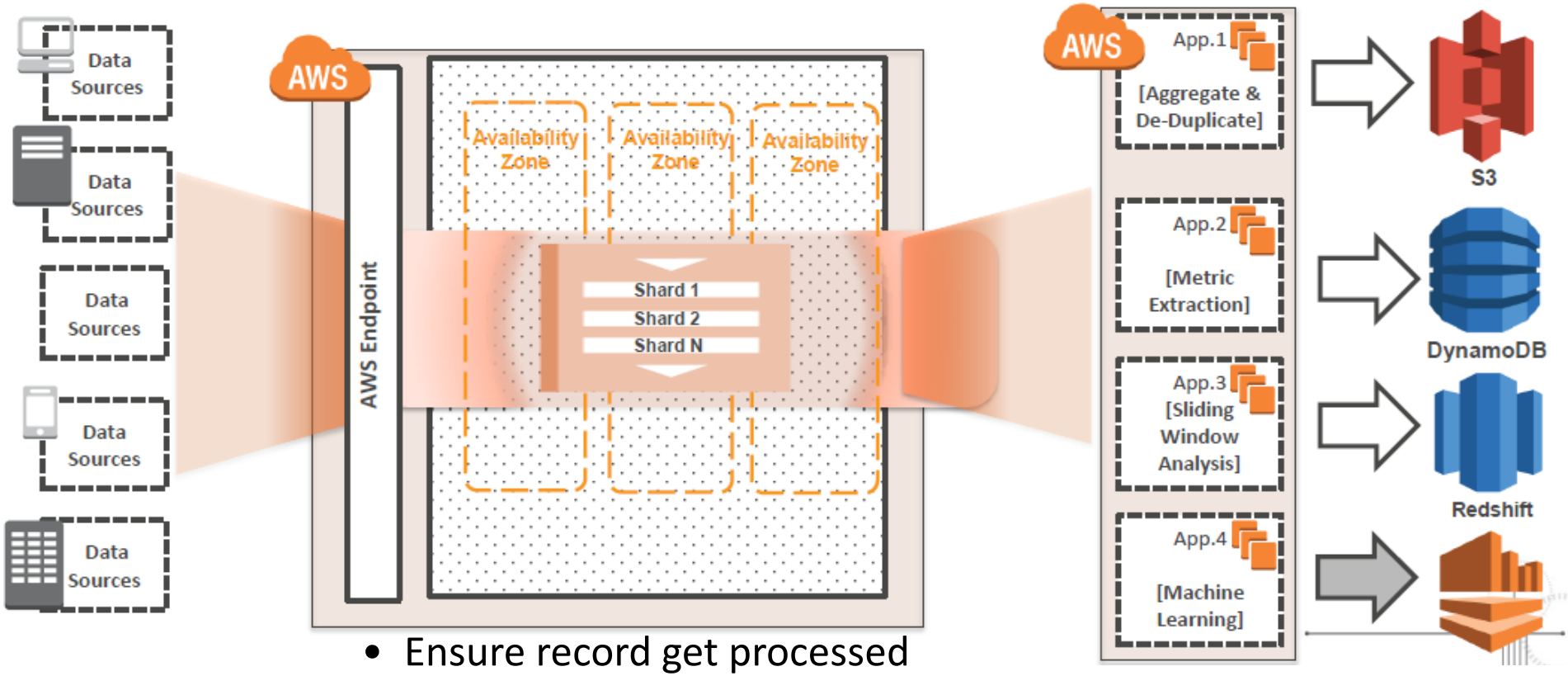
- **Data record** ~ tweet
- **Stream** ~ all tweets (the Twitter Firehose)
- **Partition key** ~ Twitter topic (every tweet record belongs to exactly one of these)
- **Shard** ~ all the data records belonging to a set of Twitter topics that will get grouped together
- **Sequence number** ~ each data record gets one assigned when first ingested.
- **Worker** ~ processes the records of a shard in sequence number order



"A common use is the real-time aggregation of data followed by loading the aggregate data into a data warehouse or map-reduce cluster"

Amazon Kinesis

Service for Real-Time Big Data Ingestion that Enables Continuous Processing

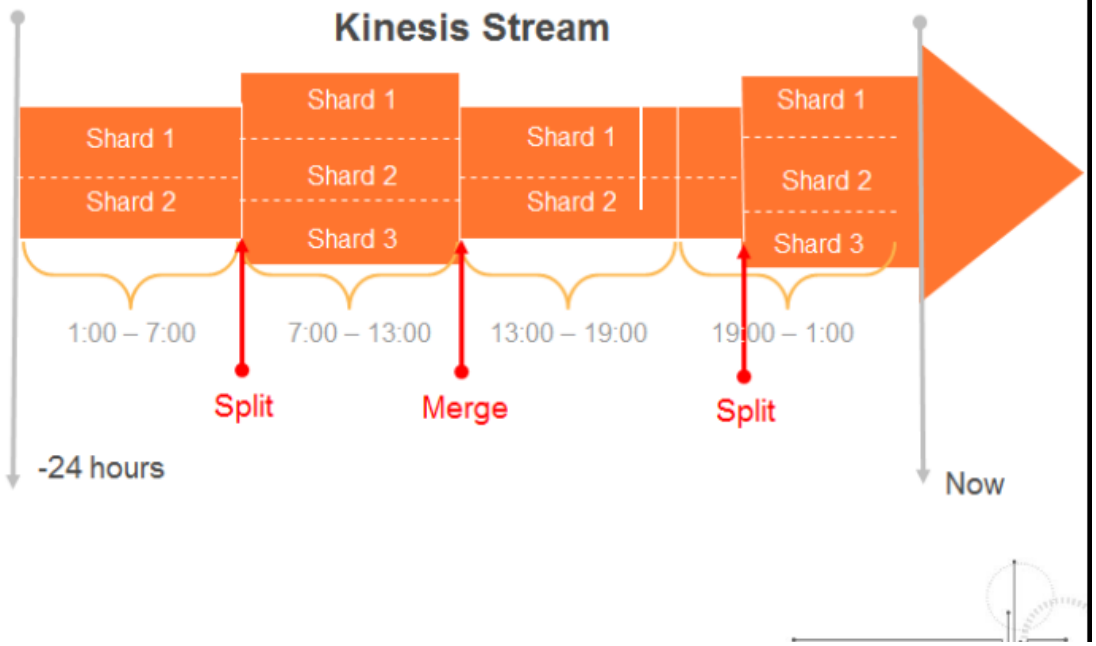


- Ensure record get processed
- Ensure records are distributed
- Match workers to shards..

Amazon Kinesis

Kinesis Stream: Managed ability to capture and store data

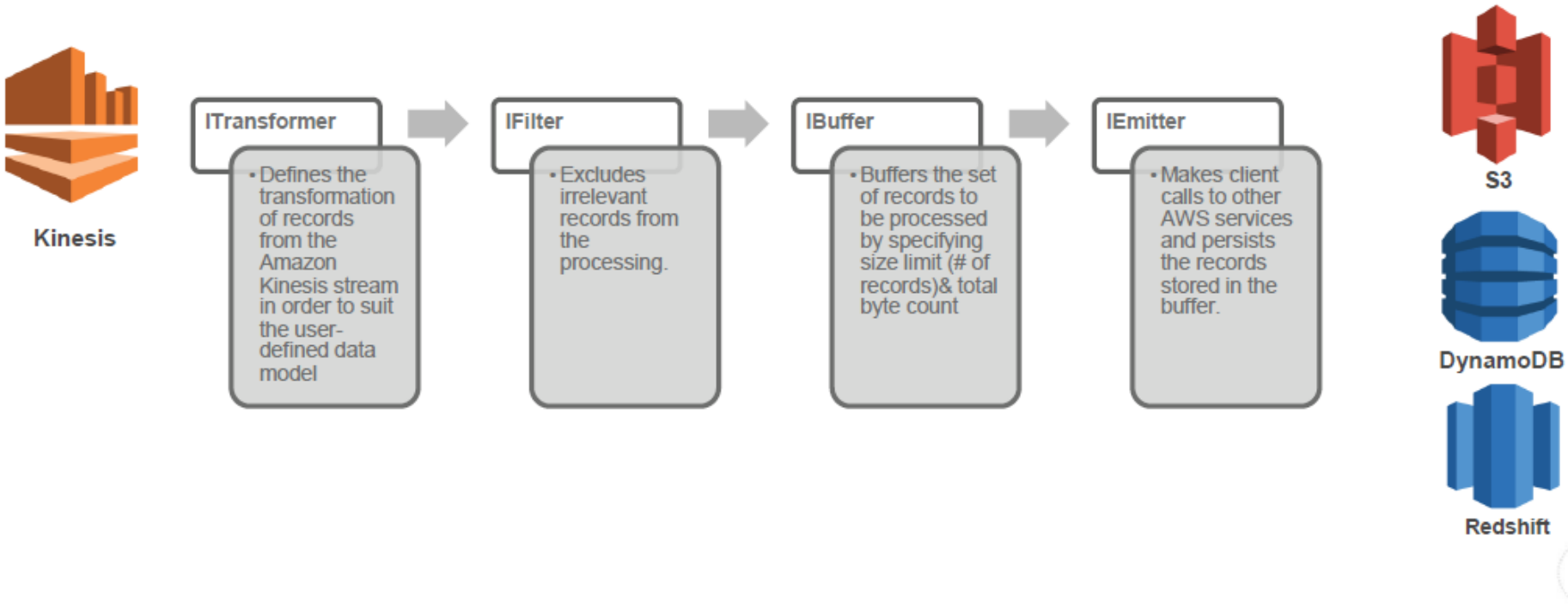
- Streams contain **Shards**
- Each Shard ingests data up to 1MB/sec, and up to 1000 TPS
- Each Shard emits up to 2 MB/sec
- All data is stored for **24 hours**
- **Scale** Kinesis streams by adding or removing Shards
- **Replay** data inside of 24Hr. Window



Amazon Kinesis

Amazon Kinesis Connector Library

Customizable, Open Source Apps to Connect Kinesis with S3, Redshift, DynamoDB



Amazon Kinesis

- Targeted towards special classes of applications
  - Built in connectors for several open data streams, e.g. Twitter
- Cloud deployment model
  - Streaming data to cloud has security, throughput limitations
- Micro-batch processing model
  - Performance and latency limitation
  - User control and programming flexibility limited
- More information
  - http://aws.amazon.com/kinesis/