HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

DESIGNING Data-Intensive Applications
The big ideas behind reliable, scalable & maintainable systems
RELIABILITY
SCALABILITY
MAINTAINABILITY
Tolerating hardware & software faults
Human error
Measuring load & performance
Latency percentiles, throughput
Operability, simplicity & evolvability
Chapter 1. Reliable, Scalable, and Maintainable Applications
Chapter 2. Data Models and Query Languages
Chapter 3. Storage and Retrieval
Chapter 4. Encoding and Evolution
Chapter 5. Replication
Chapter 6: Partitioning
Chapter 7. Transactions
Chapter 8. The Trouble with Distributed Systems
Chapter 9. Consistency and Consensus
Chapter 10. Batch Processing
Chapter 11. Stream Processing
Chapter 12. The Future of Data Systems
OCEAN OF DISTRIBUTED DATA
SEA OF STORAGE & RETRIEVAL
SEA OF DERIVED DATA
COAST OF TEXTUAL ENCODINGS
GULF OF BINARY ENCODINGS
DISTRIBUTED SYSTEMS
BAY OF CAUSALITY
UNCHARTED TERRITORY
W
E
N
S

Distributed Data Management
Lecture Summary

Thorsten Papenbrock

F-2.04, Campus II

Hasso Plattner Institut

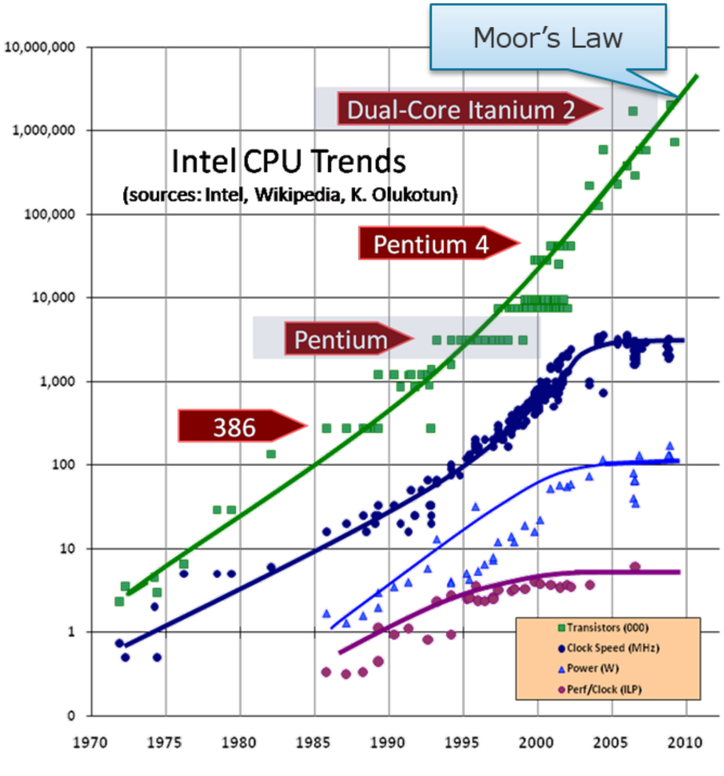
O'REILLY
Martin Kleppmann: *Designing Data-Intensive Applications*
O'Reilly Media, 2017
ISBN 978-1-4493-7332-0

Overview

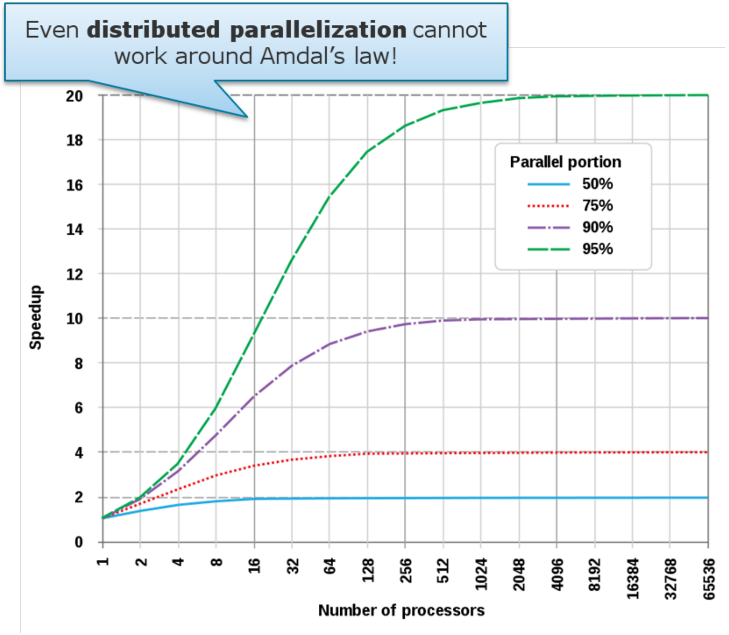
Topics DDM

1. Introduction
2. Foundations
3. Encoding & Communication
4. Akka Actor Programming
5. Data Models & Query Languages
6. Storage & Retrieval
7. Replication
8. Partitioning
9. Distributed Systems
10. Consistency & Consensus
11. Transactions
12. Batch Processing
13. Spark Batch Processing
14. Stream Processing
15. Distributed DBMS
16. Distributed Query Optimization

Some Important Topics
1 Introduction



Moor's Law



Even **distributed parallelization** cannot work around Amdal's law!

$$Speedup(s) = \frac{1}{(1-p)+\frac{p}{s}}$$

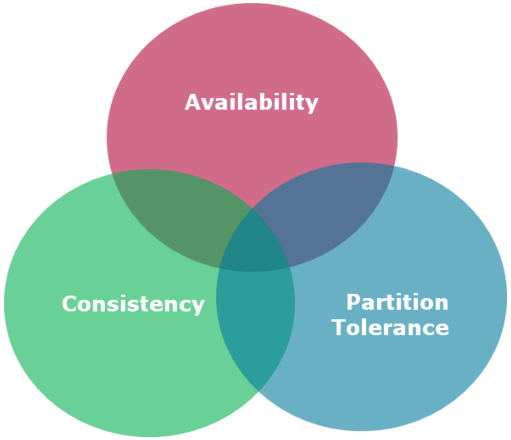
Some Important Topics

2 Foundations

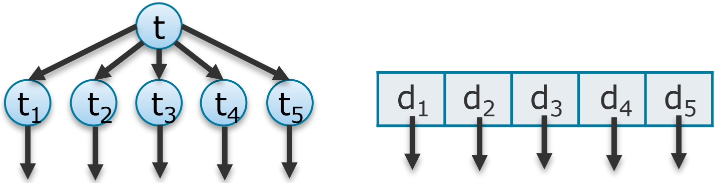
Reliability

- = *fault-tolerance*: fault/defect → may cause → error → may **not** cause → failure

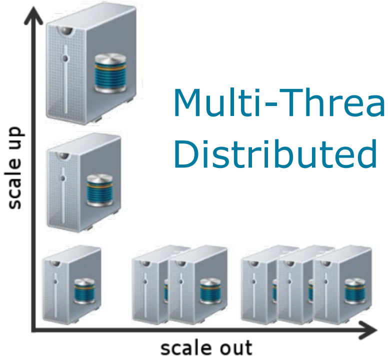
ACID & CAP & BASE



Task-Parallelism vs. Data-Parallelism

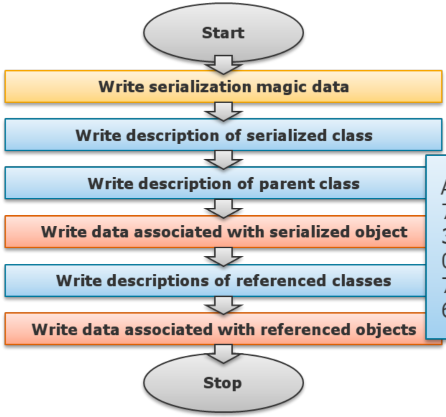


Multi-Threading vs. Distributed Computing



Some Important Topics
3 Encoding & Communication

Java Serialization



Start

Write serialization magic data

Write description of serialized class

Write description of parent class

Write data associated with serialized object

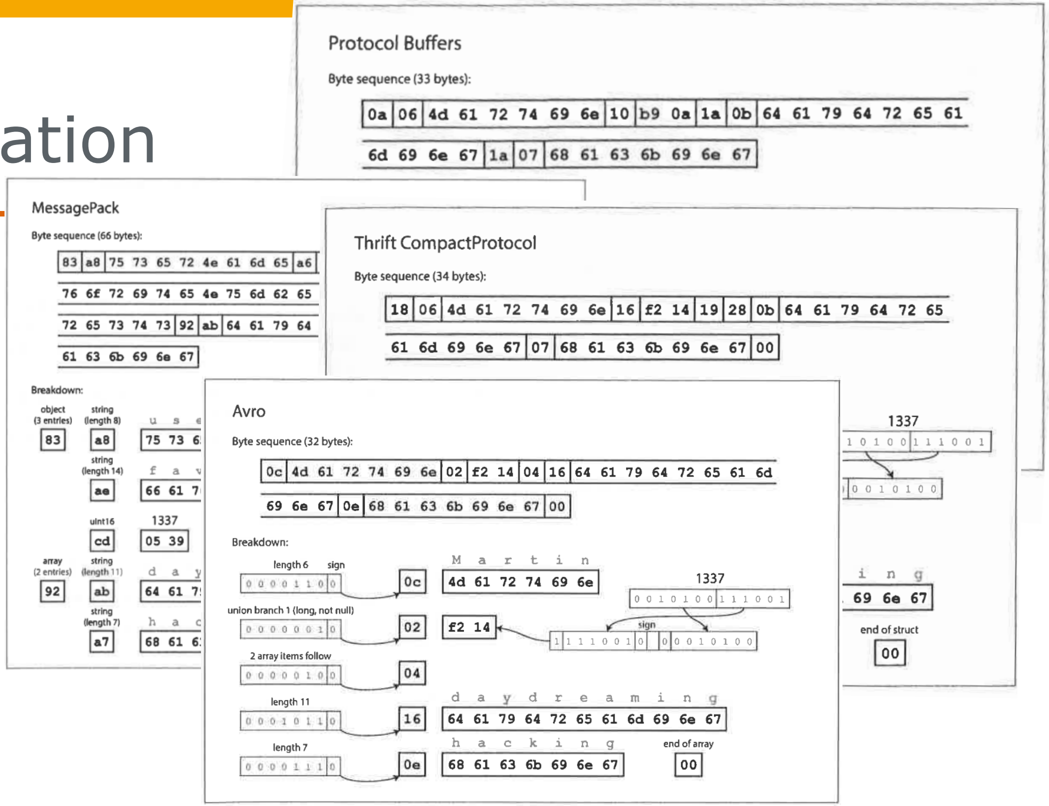
Write descriptions of referenced classes

Write data associated with referenced objects

Stop

```
class TestSerial implements Serializable {
  public byte version = 100;
  public byte count = 0;
}
```

```
AC ED 00 05 73 72 00 0A 53 65
72 69 61 6C 54 65 73 74 A0 0C
34 00 FE B1 DD F9 02 00 02 42
00 05 63 6F 75 6E 74 42 00 07
76 65 72 73 69 6F 6E 78 70 00
64
```



Protocol Buffers

Byte sequence (33 bytes):

```
0a 06 4d 61 72 74 69 6e 10 b9 0a 1a 0b 64 61 79 64 72 65 61
6d 69 6e 67 1a 07 68 61 63 6b 69 6e 67
```

MessagePack

Byte sequence (66 bytes):

```
83 a8 75 73 65 72 4e 61 6d 65 a6
76 6f 72 69 74 65 4e 75 6d 62 65
72 65 73 74 73 92 ab 64 61 79 64
61 63 6b 69 6e 67
```

Breakdown:

Thrift CompactProtocol

Byte sequence (34 bytes):

```
18 06 4d 61 72 74 69 6e 16 f2 14 19 28 0b 64 61 79 64 72 65
61 6d 69 6e 67 07 68 61 63 6b 69 6e 67 00
```

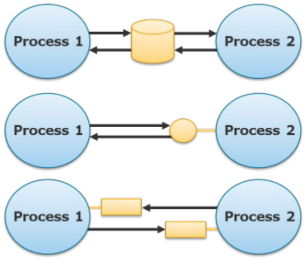
Avro

Byte sequence (32 bytes):

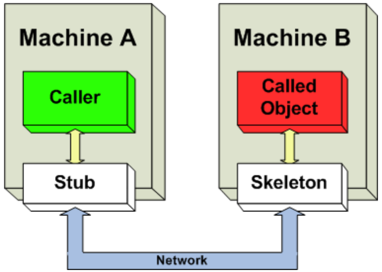
```
0c 4d 61 72 74 69 6e 02 f2 14 04 16 64 61 79 64 72 65 61 6d
69 6e 67 0e 68 61 63 6b 69 6e 67 00
```

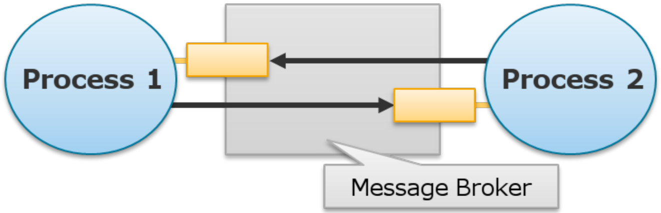
Breakdown:

Dataflow Models



RPCs



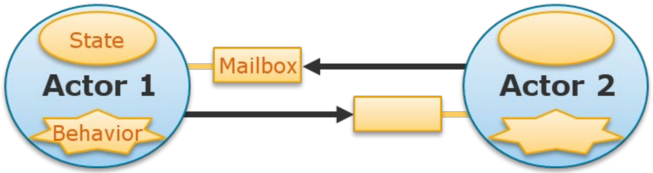


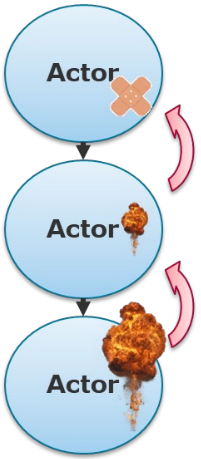
ThorstenPapenbrock
Slide **5**

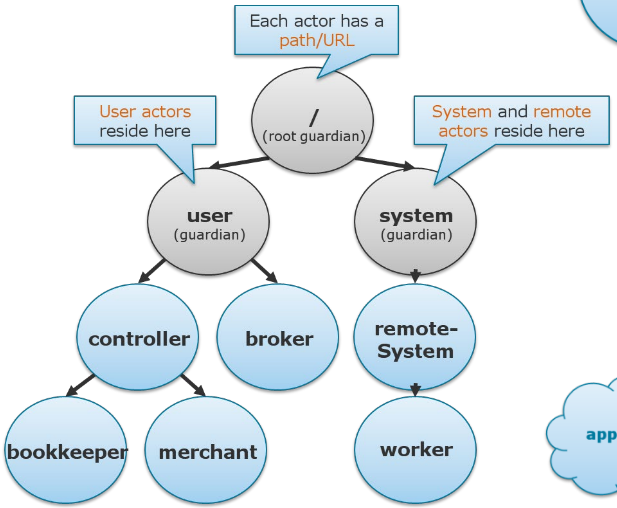
Some Important Topics
4 Akka

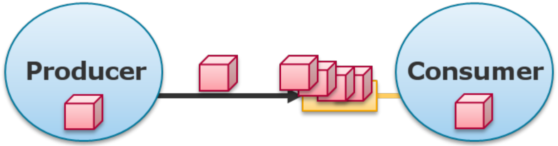
Actor Model

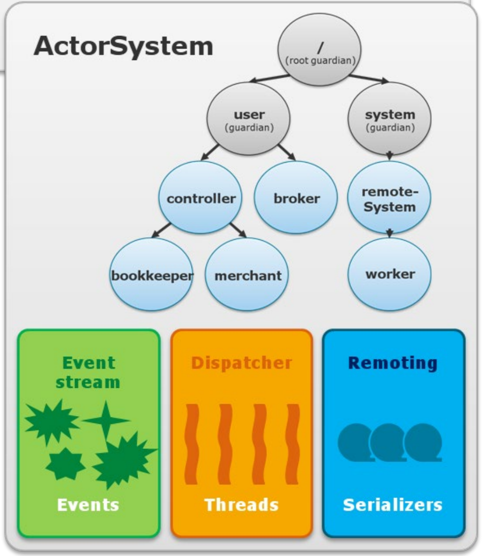
```
public class Worker extends AbstractActor {
  @Override
  public Receive createReceive() {
    return receiveBuilder()
      .match(String.class, s -> this.sender().tell("Hello!", this.self()))
      .match(Integer.class, i -> this.sender().tell(i * i, this.self()))
      .match(Doube.class, d -> this.sender().tell(d > 0 ? d : 0, this.self()))
      .match(MyMessage.class, s -> this.sender().tell(new YourMessage(), this.self()))
      .matchAny(object -> System.out.println("Could not understand received message"))
      .build();
  }
}
```

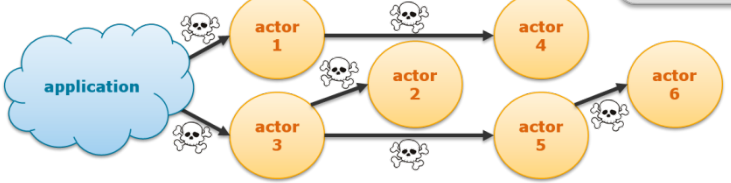












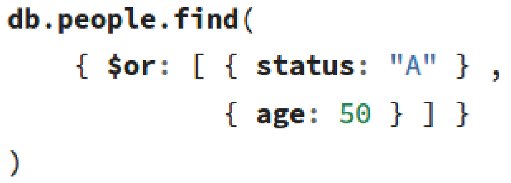
Some Important Topics

5 Data Models & Query Languages

SPARQL

```
SELECT ?locationName
WHERE {
    ?hpi :name “HPI gGmbH” .
    ?hpi :location ?locationName .
}
```

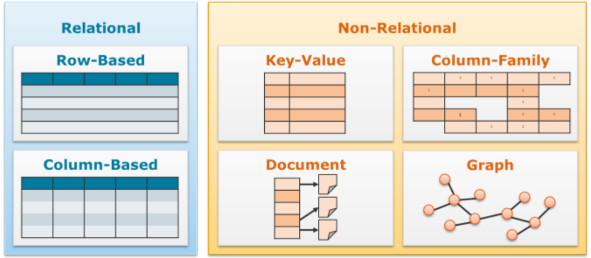
MongoDB API



```
db.people.find(
    { $or: [ { status: "A" } ,
             { age: 50 } ] }
)
```

SQL

```
SELECT *
FROM PC PC1, PC PC2
WHERE PC1.speed = PC2.speed
AND PC1.ram = PC2.ram
AND PC1.model < PC2.model;
```



Redis

```
SET hello “hello world”
GET hello
→ “hello world”
```

Cipher

```
MATCH (me {name:"T. Papenbrock "})
MATCH (expert)-[:KNOWS]->(db:Database {name:"Neo4j"})
MATCH path = shortestPath( (me)-[:FRIEND*..5]-(expert) )
RETURN db, expert, path
```

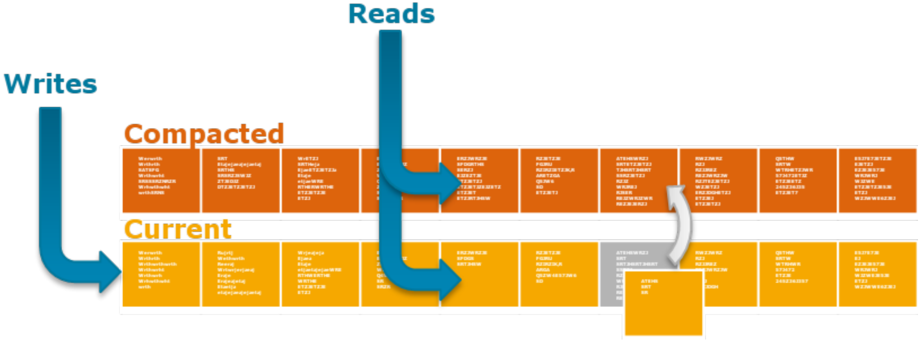
CQL

```
SELECT *
FROM myTable
WHERE myField > 5000
AND myField < 100000
ALLOW FILTERING;
```

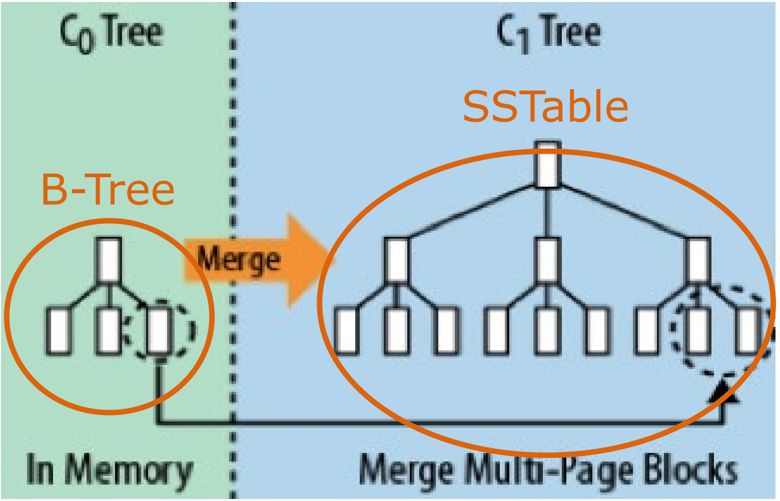
Some Important Topics

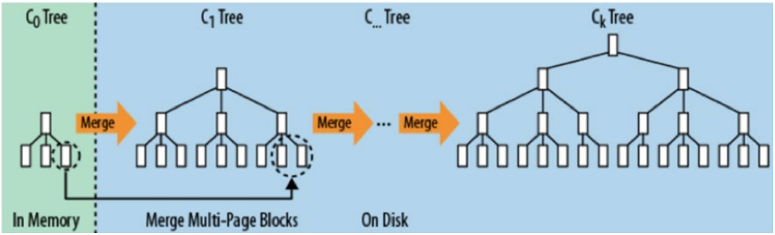
6 Storage & Retrieval

Segmentation



LSM-Trees with B-trees and SSTables

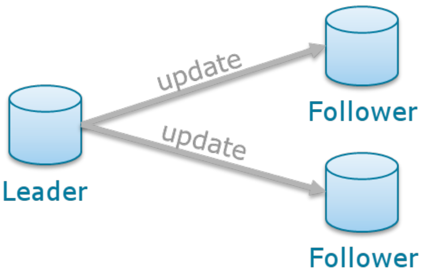




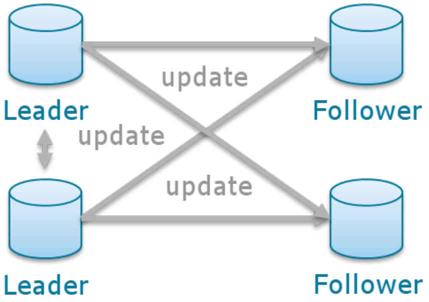
Some Important Topics

7 Replication

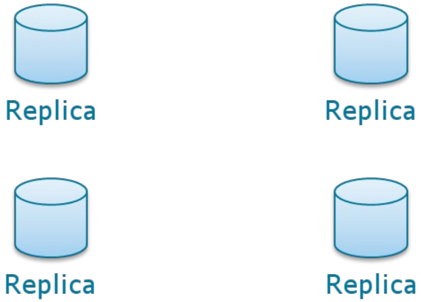
Single-Leader Replication

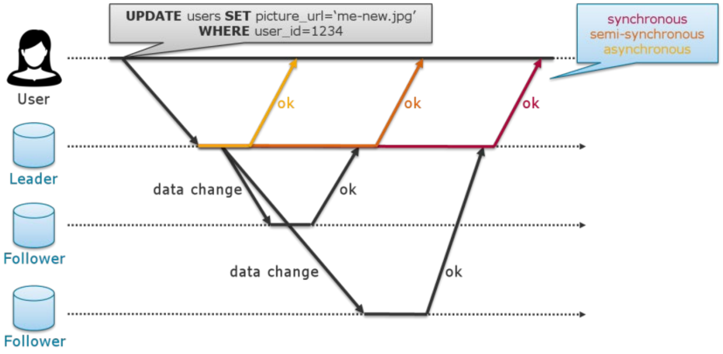


Multi-Leader Replication



Leaderless Replication





```
UPDATE users SET picture_url='me-new.jpg'
WHERE user_id=1234
```

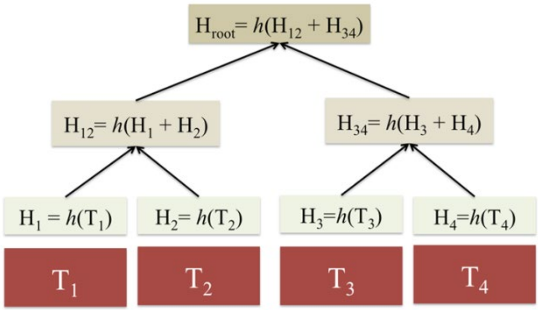
Quorum

- quorum (w,r)

Quorum Consistency

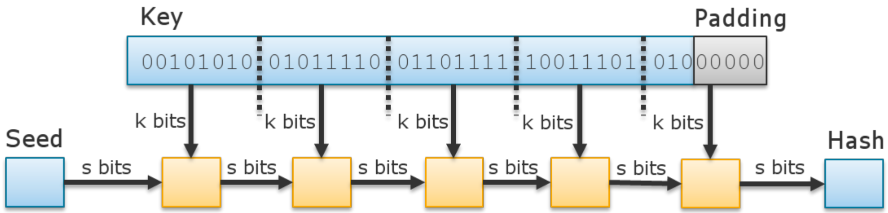
- w + r > n

Gossip & Merkle Trees

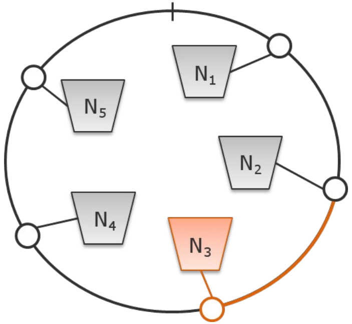


8 Partitioning

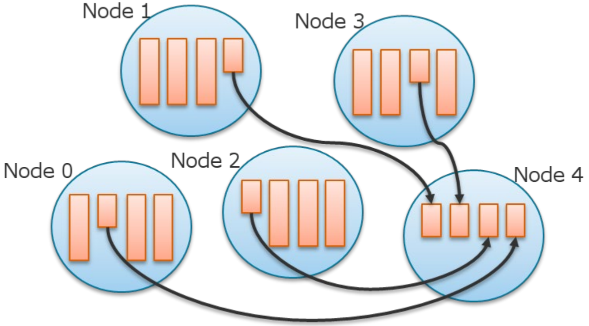
Range Partitioning by Hash of Key



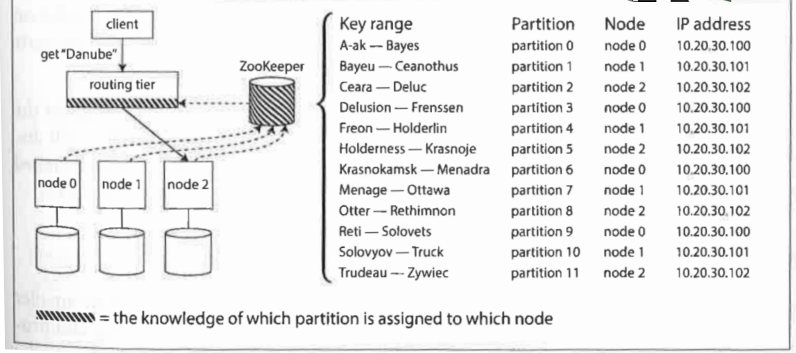
Consistent Hashing



Rebalancing Partitions



Partition-Lookup

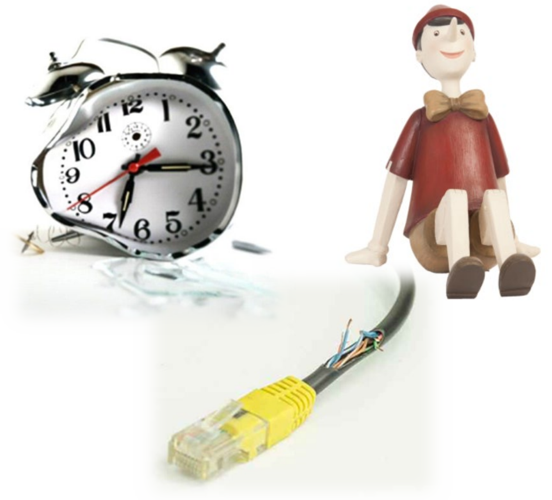


Key range	Partition	Node	IP address
A-ak — Bayes	partition 0	node 0	10.20.30.100
Bayeu — Ceanothus	partition 1	node 1	10.20.30.101
Ceara — Deluc	partition 2	node 2	10.20.30.102
Delusion — Frenssen	partition 3	node 0	10.20.30.100
Freon — Holderlin	partition 4	node 1	10.20.30.101
Holderness — Krasnoje	partition 5	node 2	10.20.30.102
Krasnokamsk — Menadra	partition 6	node 0	10.20.30.100
Menage — Ottawa	partition 7	node 1	10.20.30.101
Otter — Rethimnon	partition 8	node 2	10.20.30.102
Reti — Solovets	partition 9	node 0	10.20.30.100
Solovyov — Truck	partition 10	node 1	10.20.30.101
Trudeau — Zywiec	partition 11	node 2	10.20.30.102

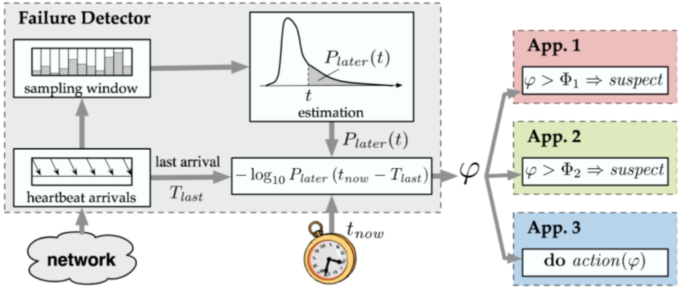
▨ = the knowledge of which partition is assigned to which node

Some Important Topics

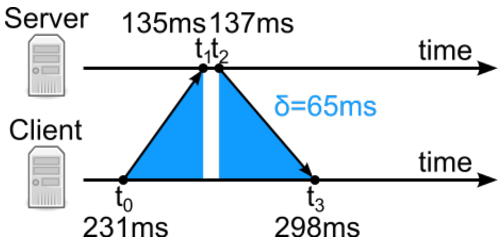
9 Distributed Systems



The φ accrual failure detector

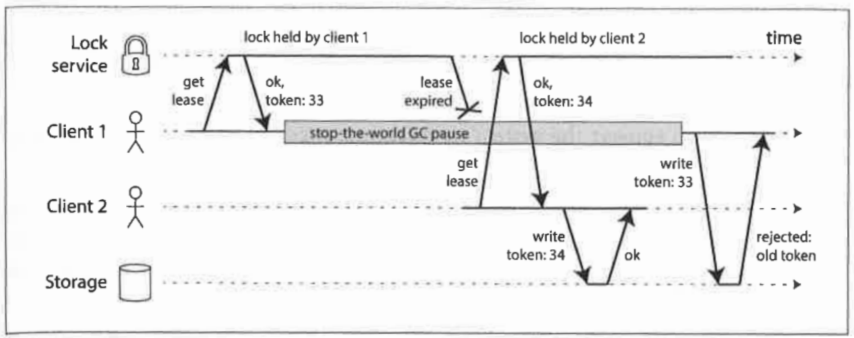


The network time protocol (NTP)



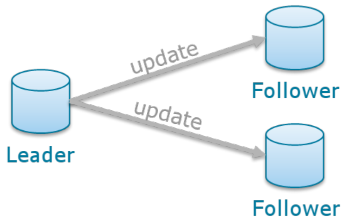
$$\theta = \frac{(t_1 - t_0) + (t_2 - t_3)}{2}$$

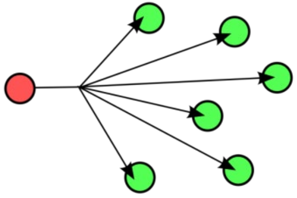
Leases

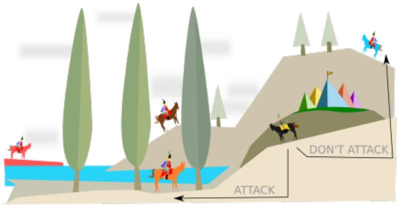


10 Consistency & Consensus

Linearizability ⟺ Total Order Broadcast ⟺ Consensus

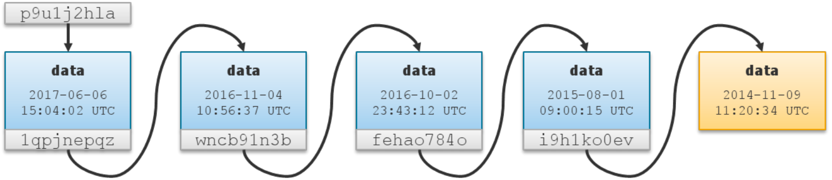




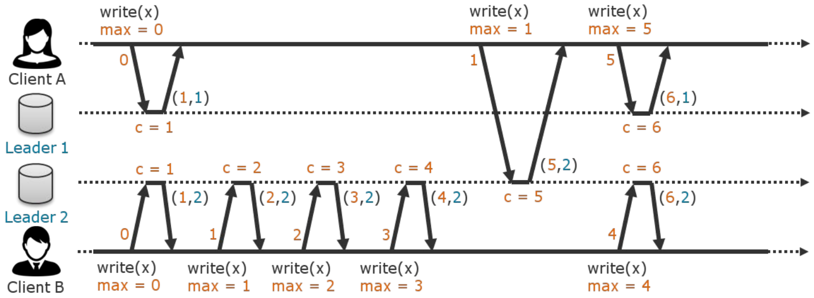


Leader Election

Blockchain

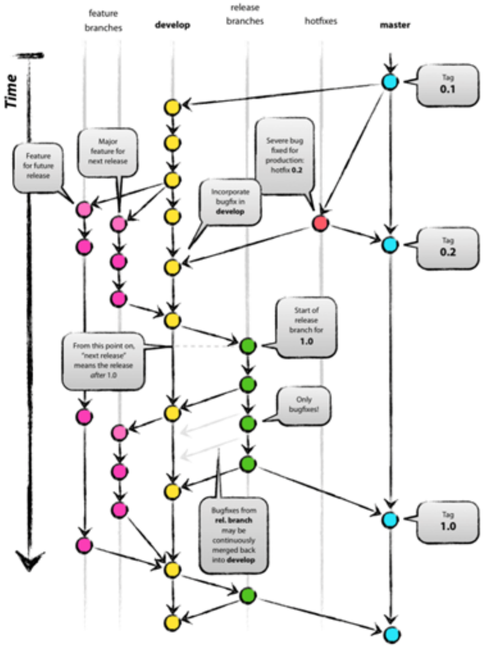


Ordering with Lamport timestamps

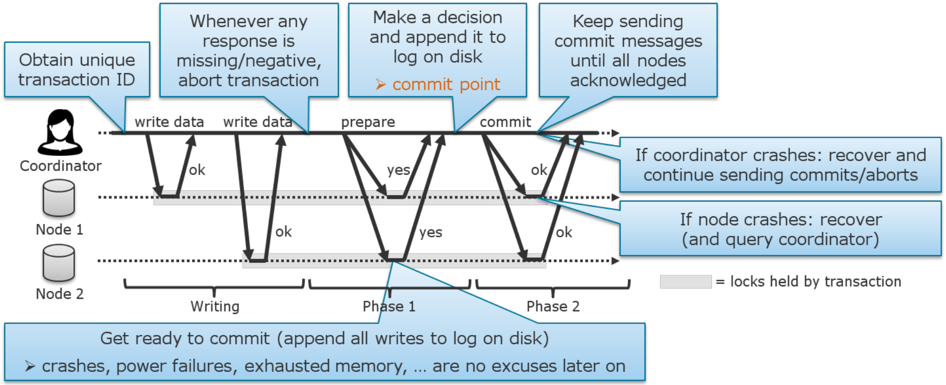


Some Important Topics
11 Transactions

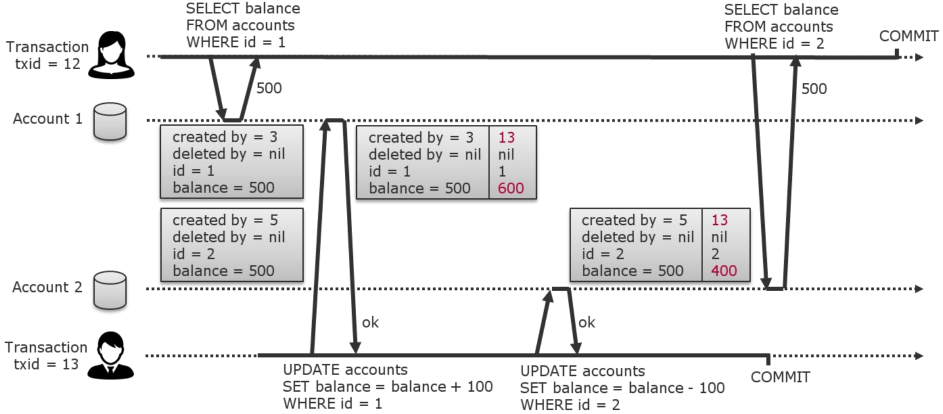
Causal Ordering


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

Two-Phase Commit (2PC)


Obtain unique transaction ID
Whenever any response is missing/negative, abort transaction
Make a decision and append it to log on disk
➢ commit point
Keep sending commit messages until all nodes acknowledged
Coordinator
Node 1
Node 2
write data
write data
prepare
commit
ok
ok
yes
yes
ok
ok
If coordinator crashes: recover and continue sending commits/aborts
If node crashes: recover (and query coordinator)
= locks held by transaction
Writing
Phase 1
Phase 2
Get ready to commit (append all writes to log on disk)
➢ crashes, power failures, exhausted memory, … are no excuses later on

Snapshot Isolation via MVCC

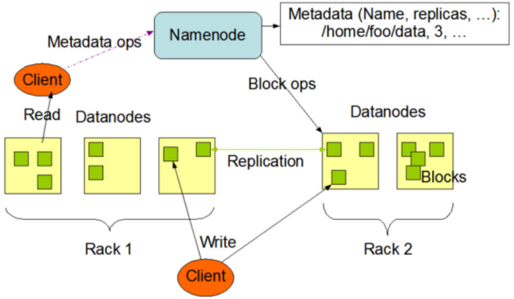


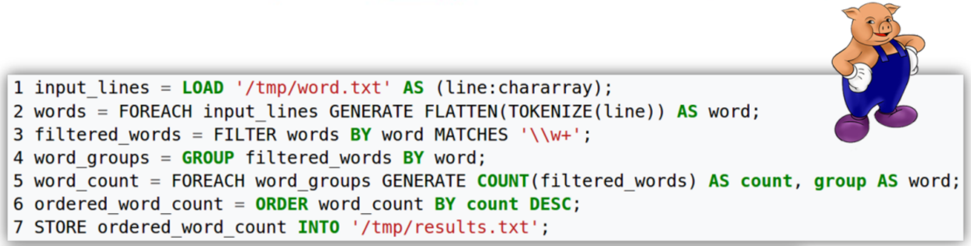
Some Important Topics

12 Batch Processing

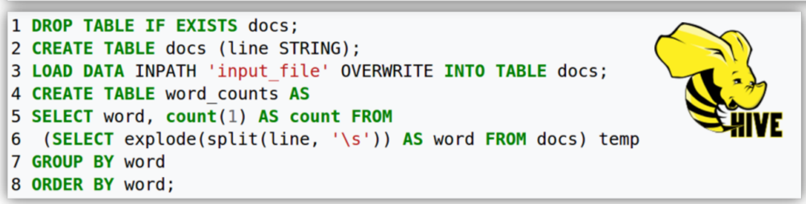
Spark Vs Flink

HDFS



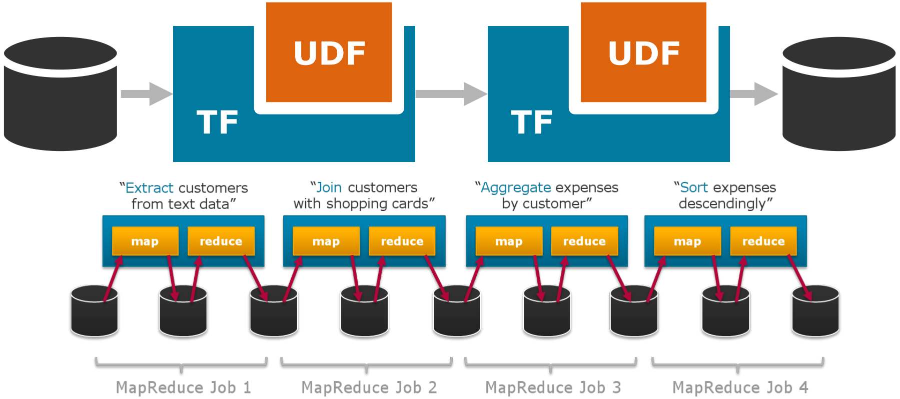


```
input_lines = LOAD '/tmp/word.txt' AS (line:chararray);
words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;
filtered_words = FILTER words BY word MATCHES '\\w+';
word_groups = GROUP filtered_words BY word;
word_count = FOREACH word_groups GENERATE COUNT(filtered_words) AS count, group AS word;
ordered_word_count = ORDER word_count BY count DESC;
STORE ordered_word_count INTO '/tmp/results.txt';
```

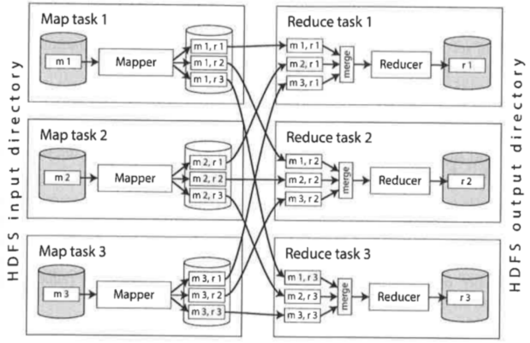


```
DROP TABLE IF EXISTS docs;
CREATE TABLE docs (line STRING);
LOAD DATA INPATH 'input_file' OVERWRITE INTO TABLE docs;
CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
 (SELECT explode(split(line, '\s')) AS word FROM docs) temp
GROUP BY word
ORDER BY word;
```

Transformation Pipelines


UDF
TF
UDF
TF
"Extract customers from text data"
"Join customers with shopping cards"
"Aggregate expenses by customer"
"Sort expenses descendingly"
map
reduce
map
reduce
map
reduce
map
reduce
MapReduce Job 1
MapReduce Job 2
MapReduce Job 3
MapReduce Job 4

MapReduce



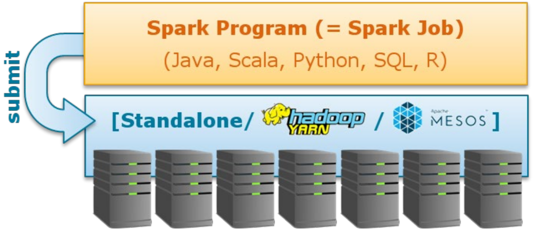
Distributed Data Management

Lecture Summary

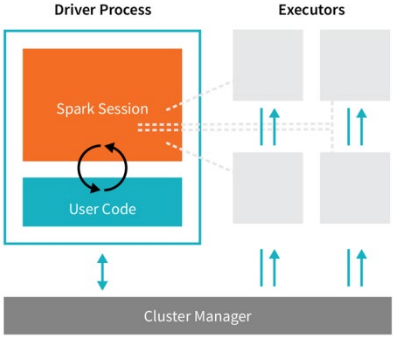
ThorstenPapenbrock
Slide **14**

Some Important Topics

13 Spark

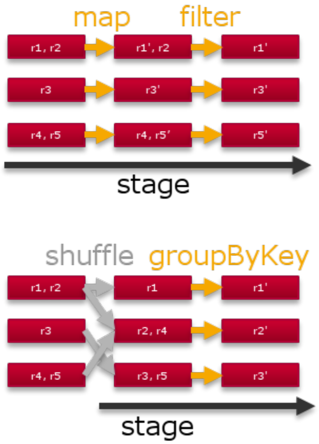


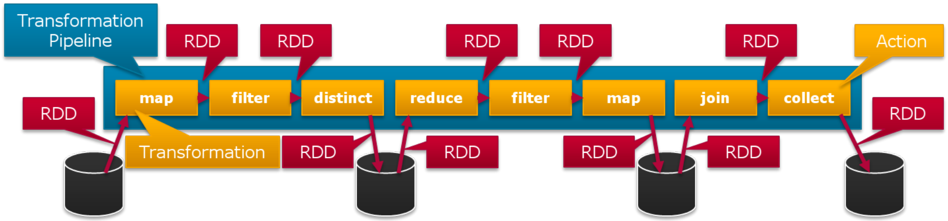
Spark Program (= Spark Job)
(Java, Scala, Python, SQL, R)
submit
[Standalone/ hadoop YARN / Apache MESOS]

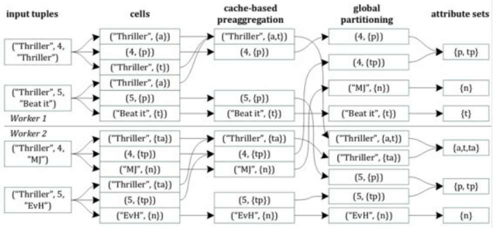


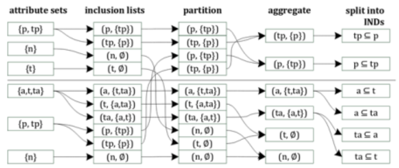
```
val sum = data.as[String]
  .filter(value => value == null)
  .flatMap(value => value.split("\\s+"))
  .map(value => (value,1))
  .reduceByKey(_+_)
  .collect()
```

```
val result = flightData
  .groupBy("DESTINATION")
  .sum("FLIGHTS")
  .sort(desc("sum(FLIGHTS)"))
  .select(
    col("DESTINATION"),
    col("sum(FLIGHTS)").as("sum"))
  .collect()
```









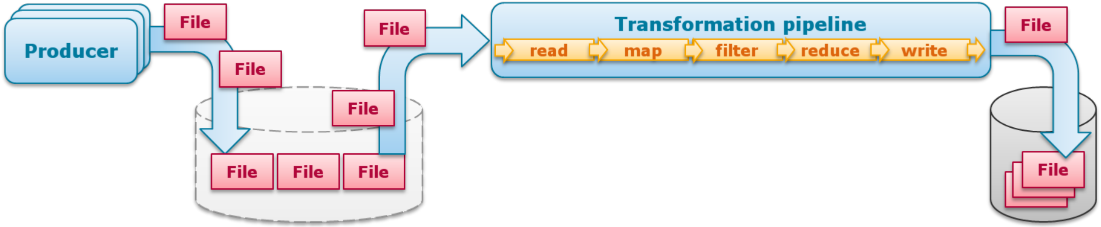
Distributed Data Management

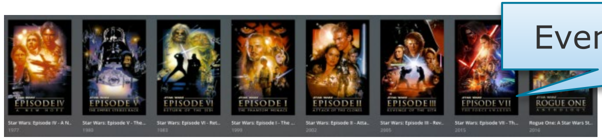
Lecture Summary

ThorstenPapenbrock
Slide **15**

Some Important Topics
14 Stream Processing

Data Streams





Event Time vs. Processing Time

CQL

```
SELECT count(*)
FROM Requests R [PARTITION BY R.client_id
                 ROWS 10 PRECEDING
                 WHERE R.domain = 'stanford.edu']
WHERE R.url LIKE 'http://cs.stanford.edu/%'
```

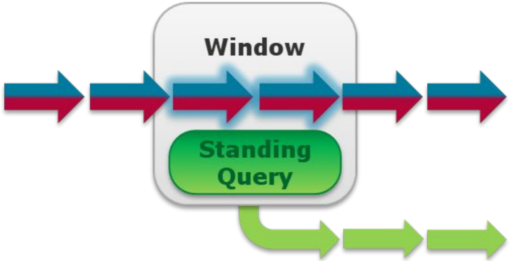
STORM

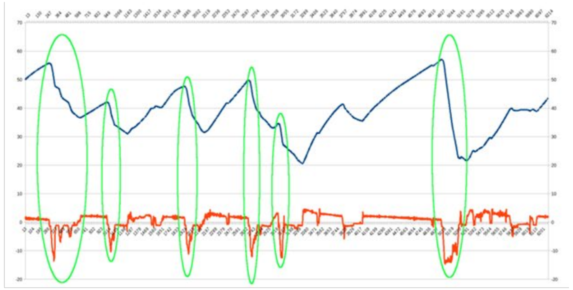
Spark

Flink

```
val env = StreamExecutionEnvironment.getExecutionEnvironment

val text = env.socketTextStream("localhost", 4242, '\n')

val windowCounts = text
  .flatMap { w => w.split("\\s") }
  .map { w => WordWithCount(w, 1) }
  .keyBy("word")
  .timeWindow(Time.seconds(5), Time.seconds(1))
  .sum("count")

windowCounts.print().setParallelism(1)

env.execute("Socket Window WordCount")

case class WordWithCount(word: String, count: Long)
```

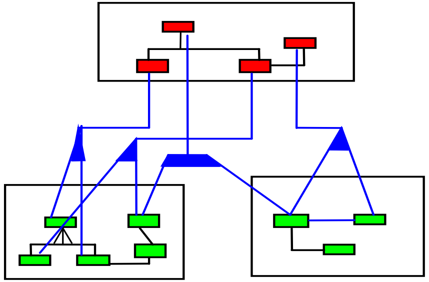
Windowing (Tumbling, Hopping, Sliding, Session)



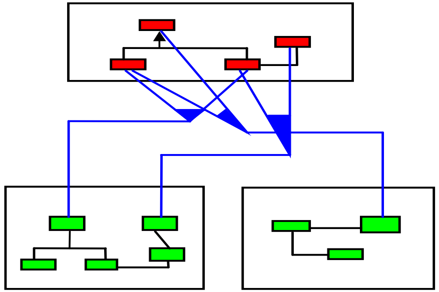


Some Important Topics
15 Distributed DBMS

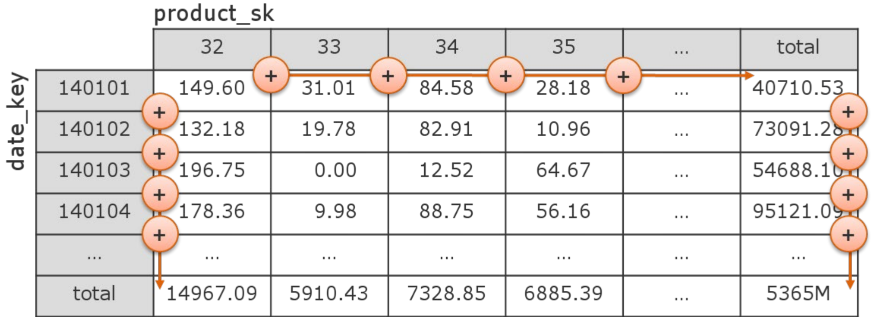
Global as View



Local as View



Data Cubes

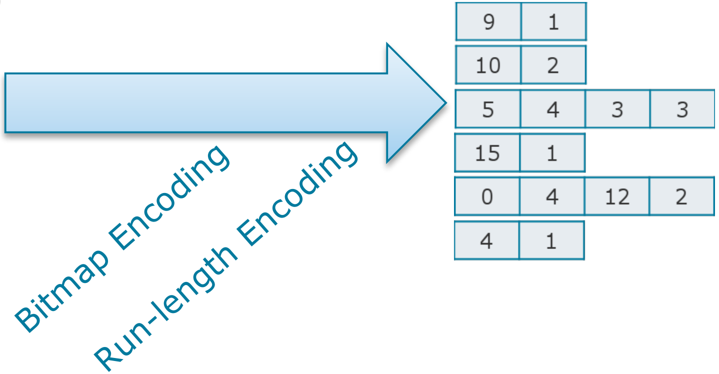


product_sk
date_key

	32	33	34	35	…	total
140101	149.60	31.01	84.58	28.18	…	40710.53
140102	132.18	19.78	82.91	10.96	…	73091.28
140103	196.75	0.00	12.52	64.67	…	54688.10
140104	178.36	9.98	88.75	56.16	…	95121.09
…	…	…	…	…	…	…
total	14967.09	5910.43	7328.85	6885.39	…	5365M

Column Store Compression (see Parquet file format)

date_key	product_sk	store_sk	promotion_sk	customer_sk	quantity	net_price	discount_price
140102	69	4	NULL	NULL	1	13.99	13.99
140102	69	5	19	NULL	3	14.99	9.99
140102	69	5	NULL	191	1	14.99	14.99
140102	74	3	23	202	5	0.99	0.89
140103	31	2	NULL	NULL	1	2.49	2.49
140103	31	3	NULL	NULL	3	14.99	9.99
140103	31	3	21	123	1	49.99	39.99
140103	31	8	NULL	233	1	0.99	0.99
file 1	file 2	file 3	file 4	file 5	file 6	file 7	file 8

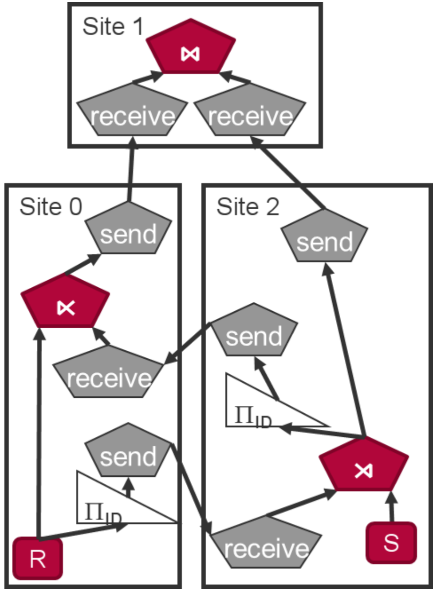


9	1		
10	2		
5	4	3	3
15	1		
0	4	12	2
4	1		

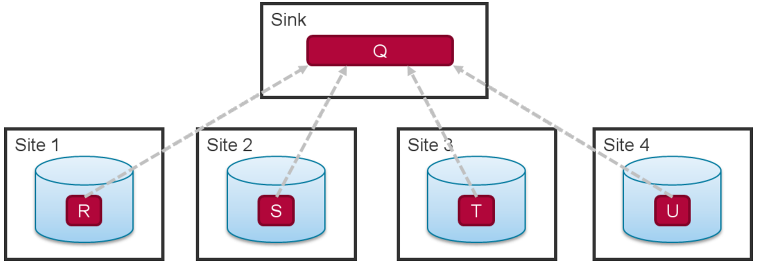
Some Important Topics

16 Distributed Query Optimization

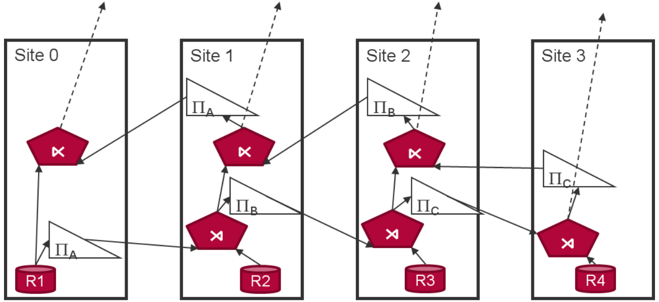
Distributed Joins



Distributed Query Execution



Distributed Join & Full Reducer



Vorwärts
R2‘ = R2 ∝ R1
R3‘ = R3 ∝ R2‘
R4‘ = R4 ∝ R3‘

Rückwärts
R3‘‘ = R3‘ ∝ R4‘
R2‘‘ = R2‘ ∝
R3‘‘
R1‘‘ = R1 ∝ R2‘‘

Semi-Join/
Reducer

Distributed Data Management

Lecture Summary

ThorstenPapenbrock

Slide **18**

What to remove from the exam menu?

Overview

Topics DDM++

17. Services and Containerization
18. Cloud-based Data Systems
19. Further Details
20. Distributed Algorithms
21. Mining Data Streams

17 Services and Containerization

Akka Cluster (Recap)

- Connects ActorSystem nodes in a cluster into one distributed system
- Has no control over …
  - resource allocation
    ActorSystems use whatever JVM resources they are started with.
  - node scaling
    ActorSystems are automatically tied together but they are started from the outside world.
  - resource isolation
    ActorSystems on the same host may compete for resources; all actors in one ActorSystem share the same resources.

17 Services and Containerization

Batch & Stream Processing Frameworks (Recap)

- Connect nodes in a cluster into one distributed system
- Perform cluster-wide resource management
- Restrict the programming to …
  - non-interactive but data-driven applications
    Transformation pipelines do not wait for user input or have observable side effects for users.
  - non-branching data analytics or data transformation applications
    Transformation pipelines do not support complex, branching application logic.
  - non-dynamic step-by-step applications
    Transformation pipelines are static sequences of standard operations.

17 Services and Containerization

Kubernetes

- Connects nodes in a cluster into one distributed system
- Performs cluster-wide resource management
- Restricts the programming only slightly

“Kubernetes (k8s) is an open-source system for automating deployment, scaling, and management of containerized applications.”

https://kubernetes.io

Topics DDM++

17 Services and Containerization

Kubernetes

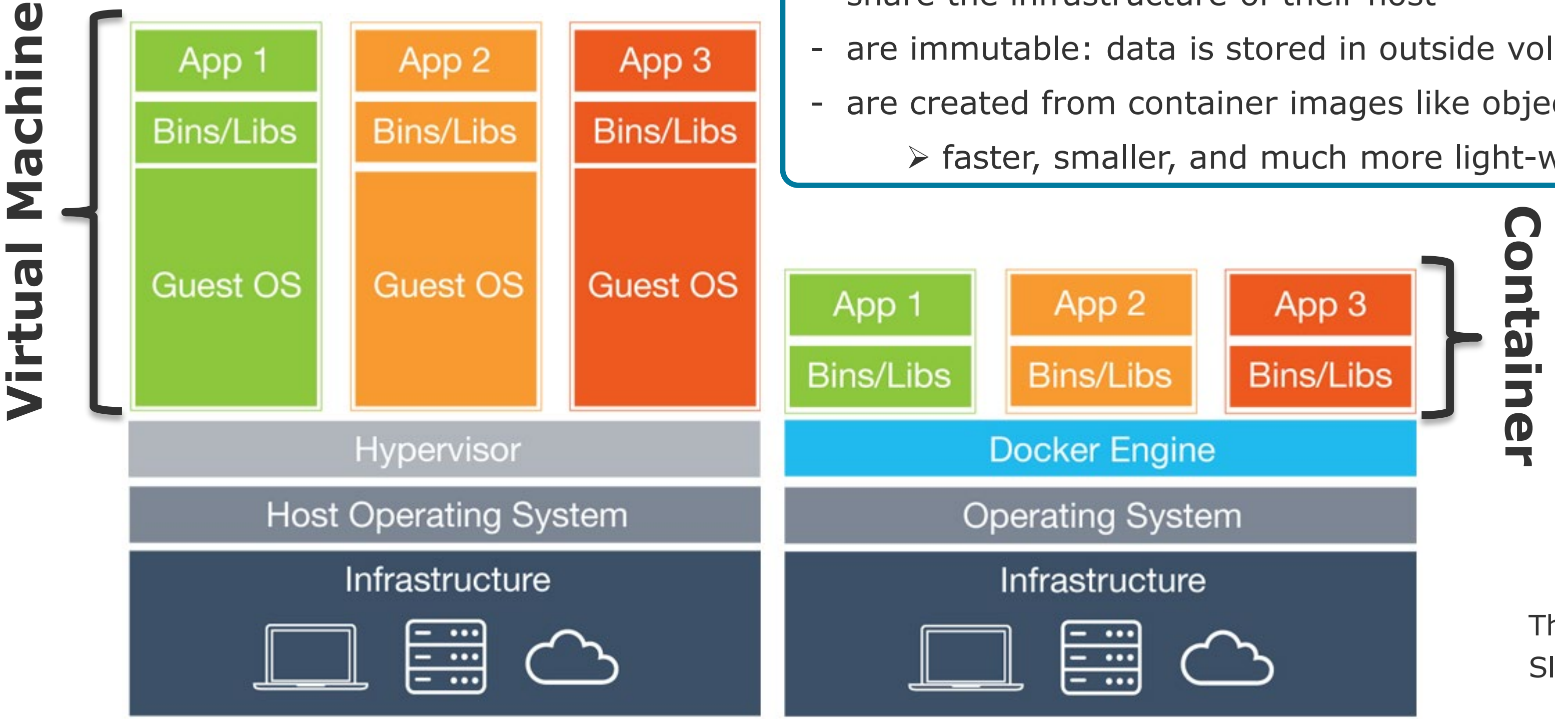
- Can be thought of as
  - a) a container platform.
  - b) a microservices platform.
  - c) a portable cloud platform.

17 Services and Containerization

Container (Docker)

Container

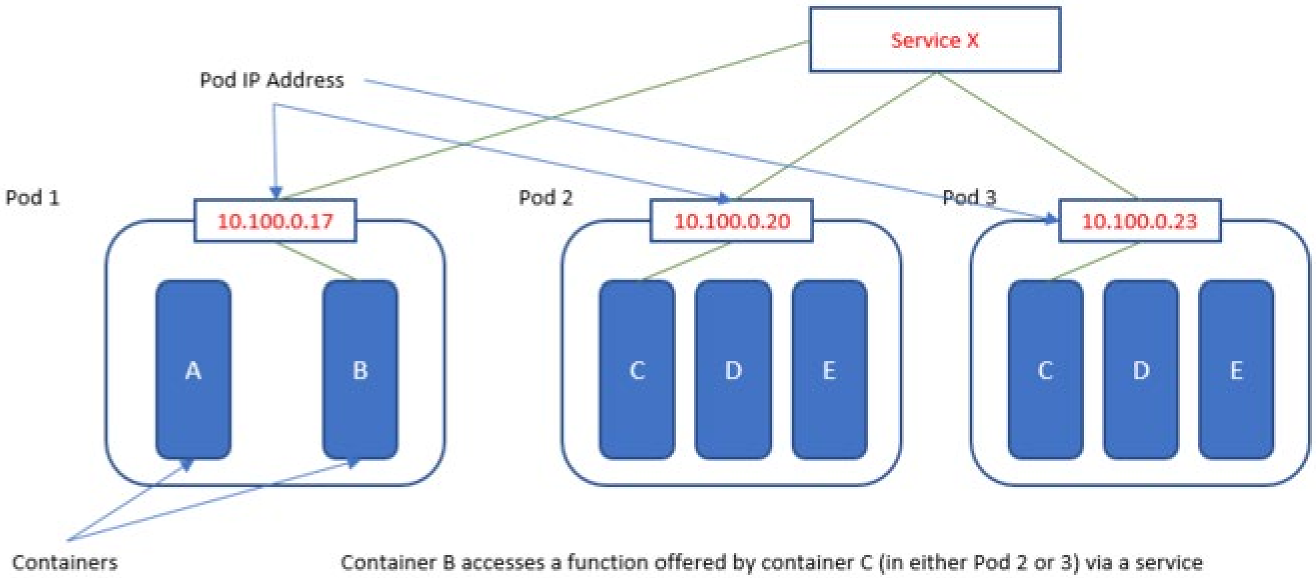
- share the infrastructure of their host
- are immutable: data is stored in outside volumes
- are created from container images like objects from classes
  - ➢ faster, smaller, and much more light-weight than VMs



Topics DDM++

17 Services and Containerization

Kubernetes



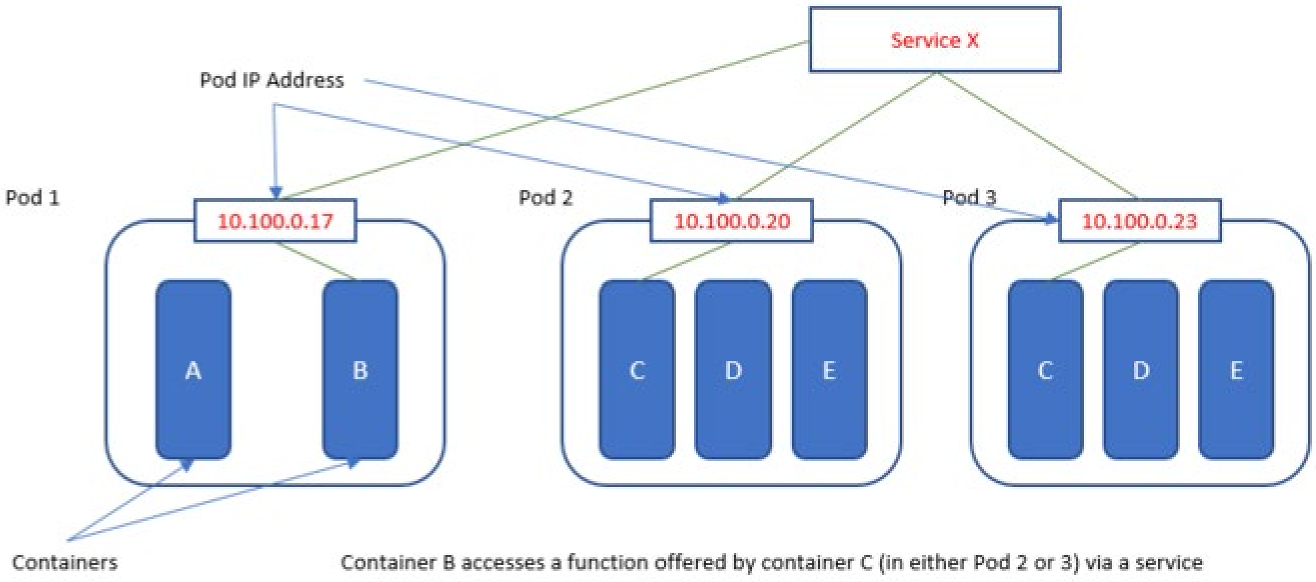
Container B accesses a function offered by container C (in either Pod 2 or 3) via a service

Container

- an application written in any programming language
- implements and encapsulates some functionality
- brings its own dependencies

17 Services and Containerization

Kubernetes



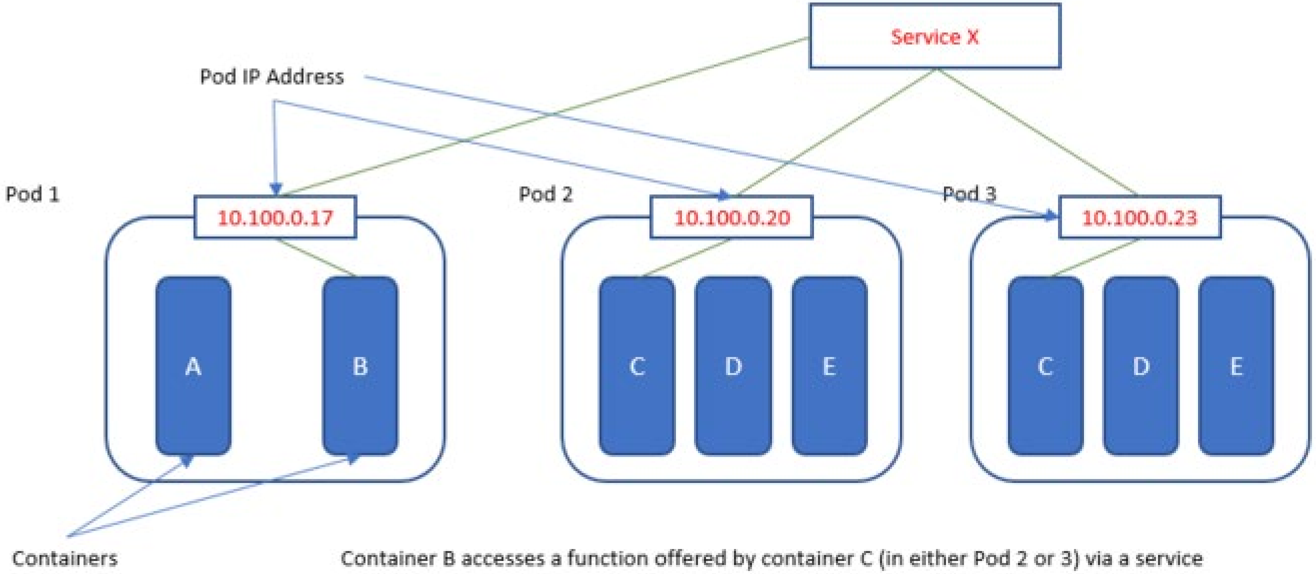
Container B accesses a function offered by container C (in either Pod 2 or 3) via a service

Pod

- a group of containers tied to some pool of resources
- the smallest scheduling unit in Kubernetes
- isolated from other pods

17 Services and Containerization

Kubernetes



Container B accesses a function offered by container C (in either Pod 2 or 3) via a service

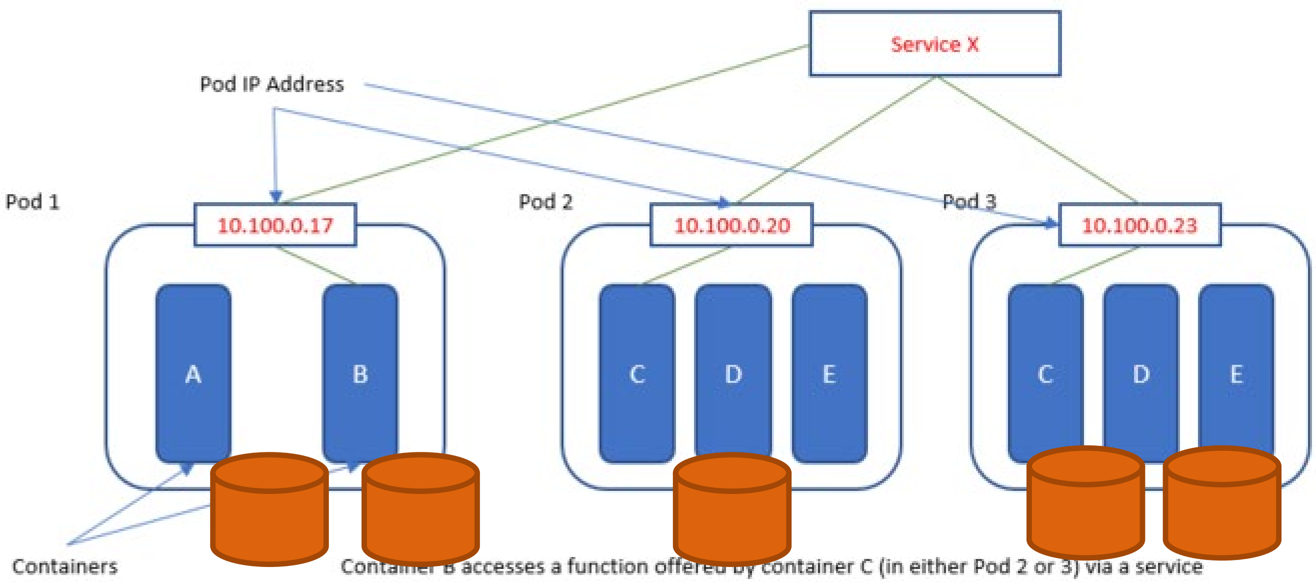
Service

- a set of pods that work together to achieve a greater task
- i.e. the orchestration of some container functions into one service endpoint
- public elements that can be looked-up in the cluster

Topics DDM++

17 Services and Containerization

Kubernetes



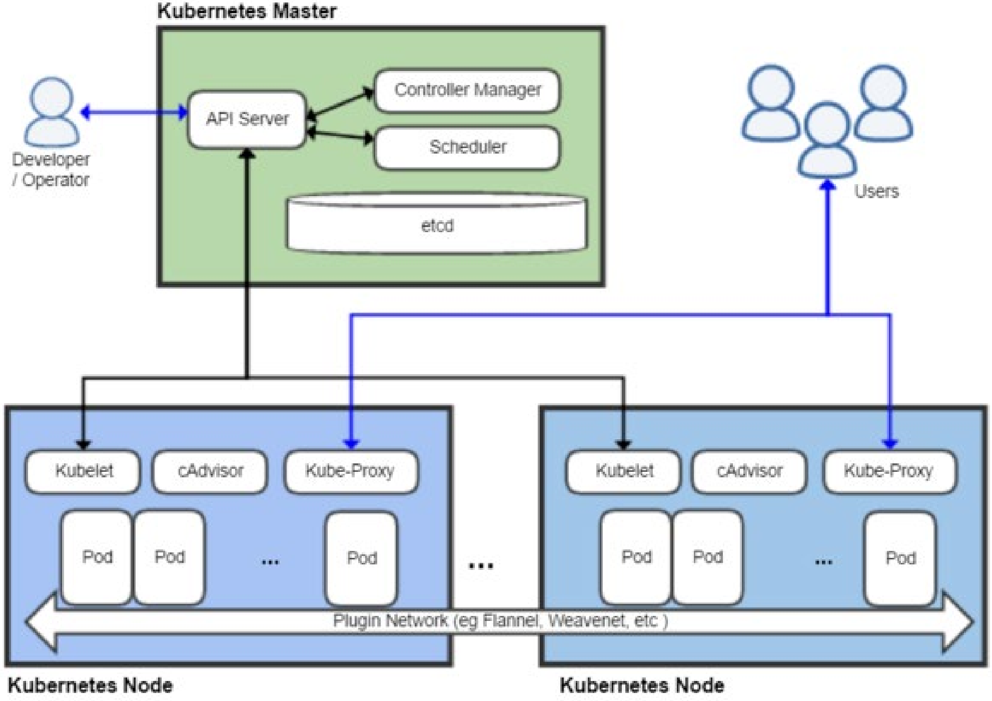
Service X
Pod IP Address
Pod 1
10.100.0.17
Pod 2
10.100.0.20
Pod 3
10.100.0.23
A
B
C
D
E
C
D
E
Containers
Container B accesses a function offered by container C (in either Pod 2 or 3) via a service

Volumes

- objects describing persistent storage
- can be shared by the containers of one pod

17 Services and Containerization

Kubernetes



API Server

- REST interface for cluster configuration (workloads and containers)

Controller Manager

- creates/deletes Pods w.r.t. some target configuration

Scheduler

- dynamic Pod scheduling on the available cluster nodes based on resource-requirements and -availability

etcd

- service discovery and cluster management (see ZooKeeper)

Kubelet

- manages and monitors all Pods on one cluster node

17 Services and Containerization

Kubernetes vs. Akka – Similarities

- Both use many same programming patterns (scheduler, router, master-worker, proxies, singletons, …)
- Both can implement batch- and stream-processing pipelines (map, reduce, join, filter … transformations as actors/Pods)
- Both provide means for dynamic scaling (creating and deleting actors/Pods based on current load)
- Both support branching logic (actors/containers decide freely: if A do this; if B do that)
- Both provide isolation for state and computation (private data in actors/containers and private resources in ActorSystems/Pods)

Topics DDM++

17 Services and Containerization

Kubernetes vs. Akka – Differences

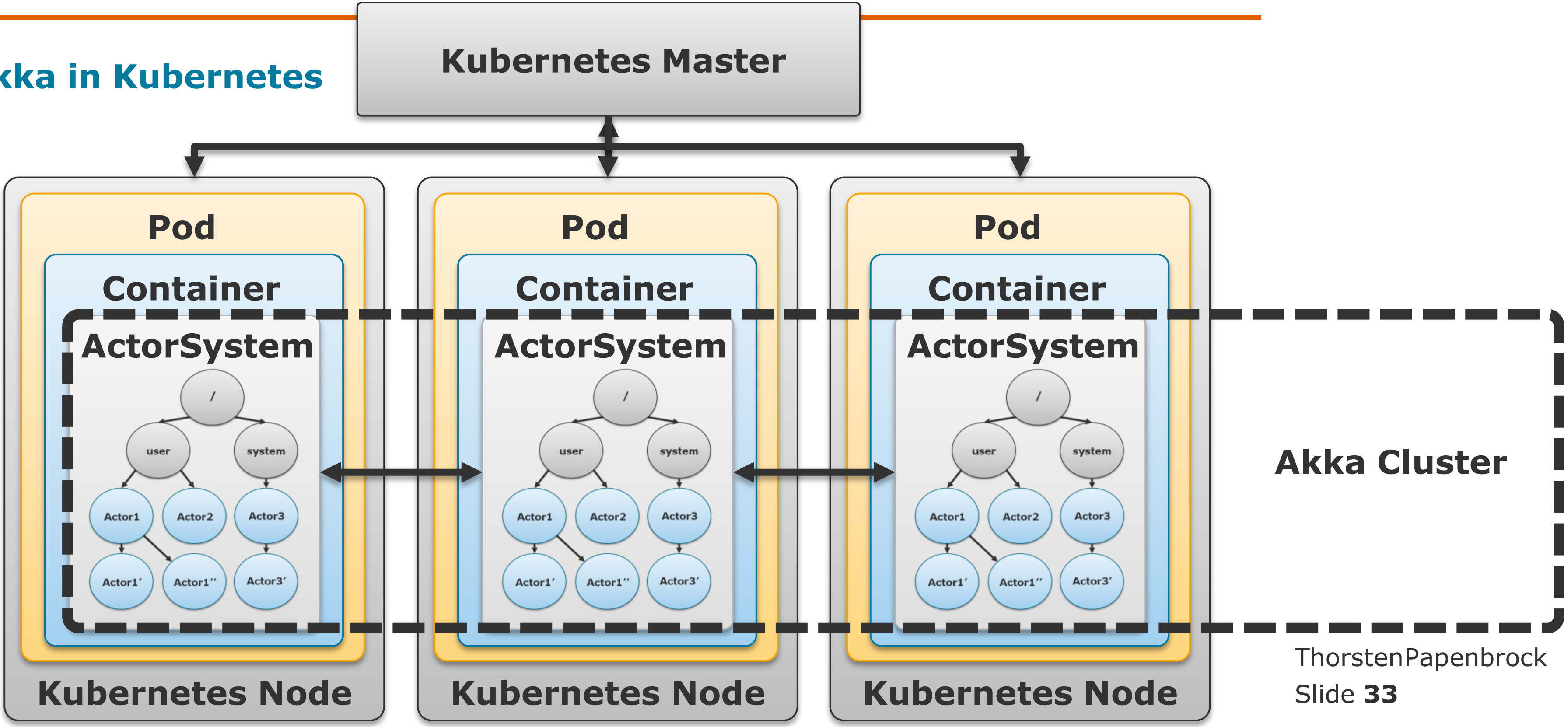
- Akka is more a programming framework while Kubernetes is an orchestration framework for programs (programming vs. configuration)
- Akka:
  - light-weight, bound to the JVM
  - difficult resource management
  - fully asynchronous messaging
- Kubernetes:
  - heavy-weight, code-agnostic due to containerization
  - powerful resource management
  - synchronous service calls

for distributed applications

for distributed systems

17 Services and Containerization

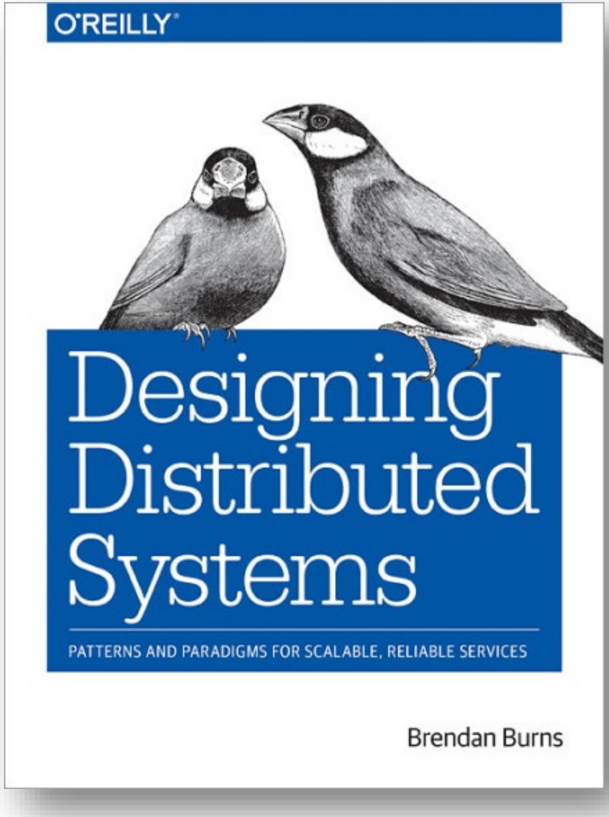
Akka in Kubernetes



17 Services and Containerization

Kubernetes further reading

- Official website and documentation
  https://kubernetes.io
- Wikipedia
  https://en.wikipedia.org/wiki/Kubernetes
- Book
  Designing Distributed Systems



O'REILLY

Designing Distributed Systems

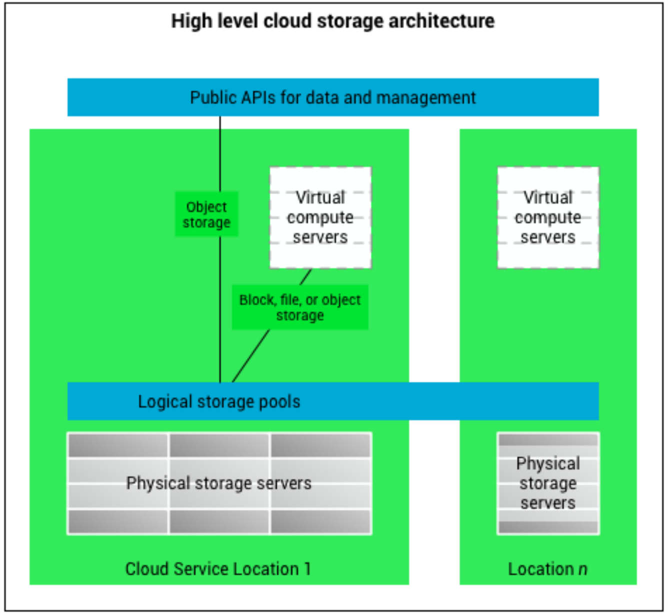
PATTERNS AND PARADIGMS FOR SCALABLE, RELIABLE SERVICES

Brendan Burns

18 Cloud-based Data Systems

Cloud-based Data Systems

- Physical storage servers
  - Partitioning: Each server persists some partitions of the data.
  - Replication: Partitions are replicated to several servers.
  - Dynamic: The number of storage servers may dynamically adjust to the amount of data.
- Virtual compute servers
  - Perform computations on the data (join, filter, sort, …)
  - Created on-demand and possibly close to the data
  - Dynamic: The number of compute servers my dynamically adjust to the query load of the system.



Topics DDM++

18 Cloud-based Data Systems

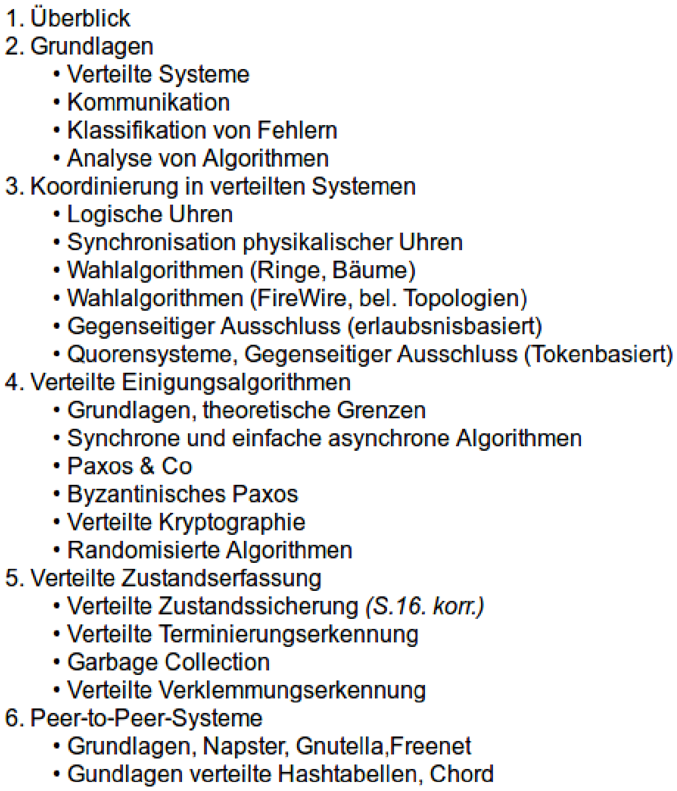
Cloud-based Data Systems

- Challenges
  - Computation and data co-placement
  - Multi-tenancy data in one data system

- Examples
  - Amazon S3
  - Oracle Cloud Storage
  - Microsoft Azure Storage
  - Openstack Swift
  - EMC Atmos
  - EMC ECS
  - Hitachi Content Platform

Topics DDM++

19 Further Details on Distributed Systems



1. Überblick
2. Grundlagen
   - Verteilte Systeme
   - Kommunikation
   - Klassifikation von Fehlern
   - Analyse von Algorithmen
3. Koordinierung in verteilten Systemen
   - Logische Uhren
   - Synchronisation physikalischer Uhren
   - Wahlalgorithmen (Ringe, Bäume)
   - Wahlalgorithmen (FireWire, bel. Topologien)
   - Gegenseitiger Ausschluss (erlaubsnisbasiert)
   - Quorensysteme, Gegenseitiger Ausschluss (Tokenbasiert)
4. Verteilte Einigungsalgorithmen
   - Grundlagen, theoretische Grenzen
   - Synchrone und einfache asynchrone Algorithmen
   - Paxos & Co
   - Byzantinisches Paxos
   - Verteilte Kryptographie
   - Randomisierte Algorithmen
5. Verteilte Zustandserfassung
   - Verteilte Zustandssicherung *(S.16. korr.)*
   - Verteilte Terminierungserkennung
   - Garbage Collection
   - Verteilte Verklemmungserkennung
6. Peer-to-Peer-Systeme
   - Grundlagen, Napster, Gnutella,Freenet
   - Gundlagen verteilte Hashtabellen, Chord

https://www4.cs.fau.de/Lehre/WS03/V_VA/Skript

- Modelle verteilter Berechnungen
- Raum-Zeit Diagrammen
- Virtuelle Zeit; logische Uhren und Kausalität
- Wellenalgorithmen
- Verteilte und parallele Graphtraversierung
- Berechnung konsistenter Schnappschüsse
- Election und Symmetriebrechung
- Verteilte Terminierung
- Garbage-Collection in verteilten Systemen
- Beobachten verteilter Systeme
- Berechnung globaler Prädikate

https://vs.inf.ethz.ch/edu/WS0405/VA

Topics DDM++

20 Distributed Algorithms

Sorting

(e.g. distributed merge sort)

Clustering

(e.g. distributed k-means)

Graph Traversal

(e.g. Bulk Synchronous Parallel model)

Machine Learning

(e.g. ML in Spark and Flink)

Data Mining

(e.g. distributed page rank)

Topics DDM++

21 Mining Data Streams

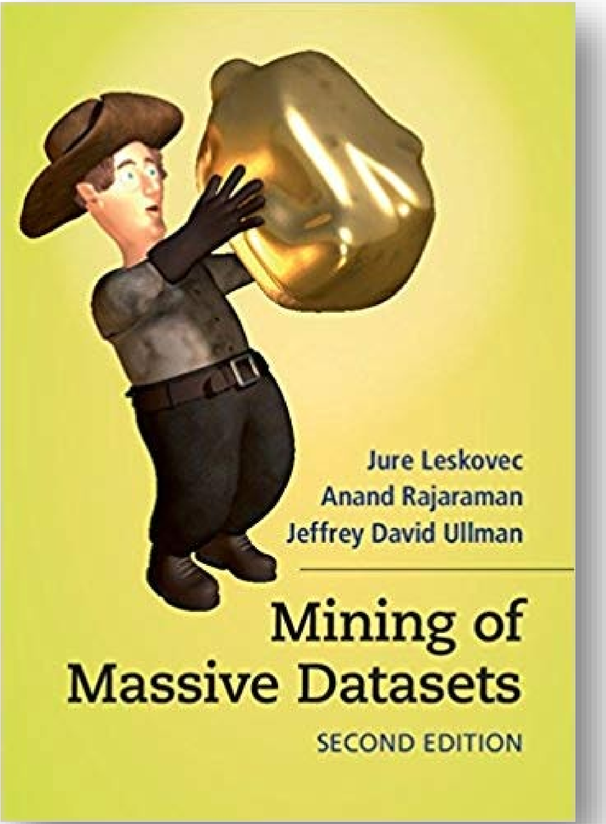
Sampling
(e.g. representative sampling window)

Filtering
(e.g. Bloomfilter)

Counting
(e.g. HyperLogLog)

Aggregation
(e.g. windowing)

Popular elements search
(e.g. decaying windows)



Overview

Next Semester

Seminar:

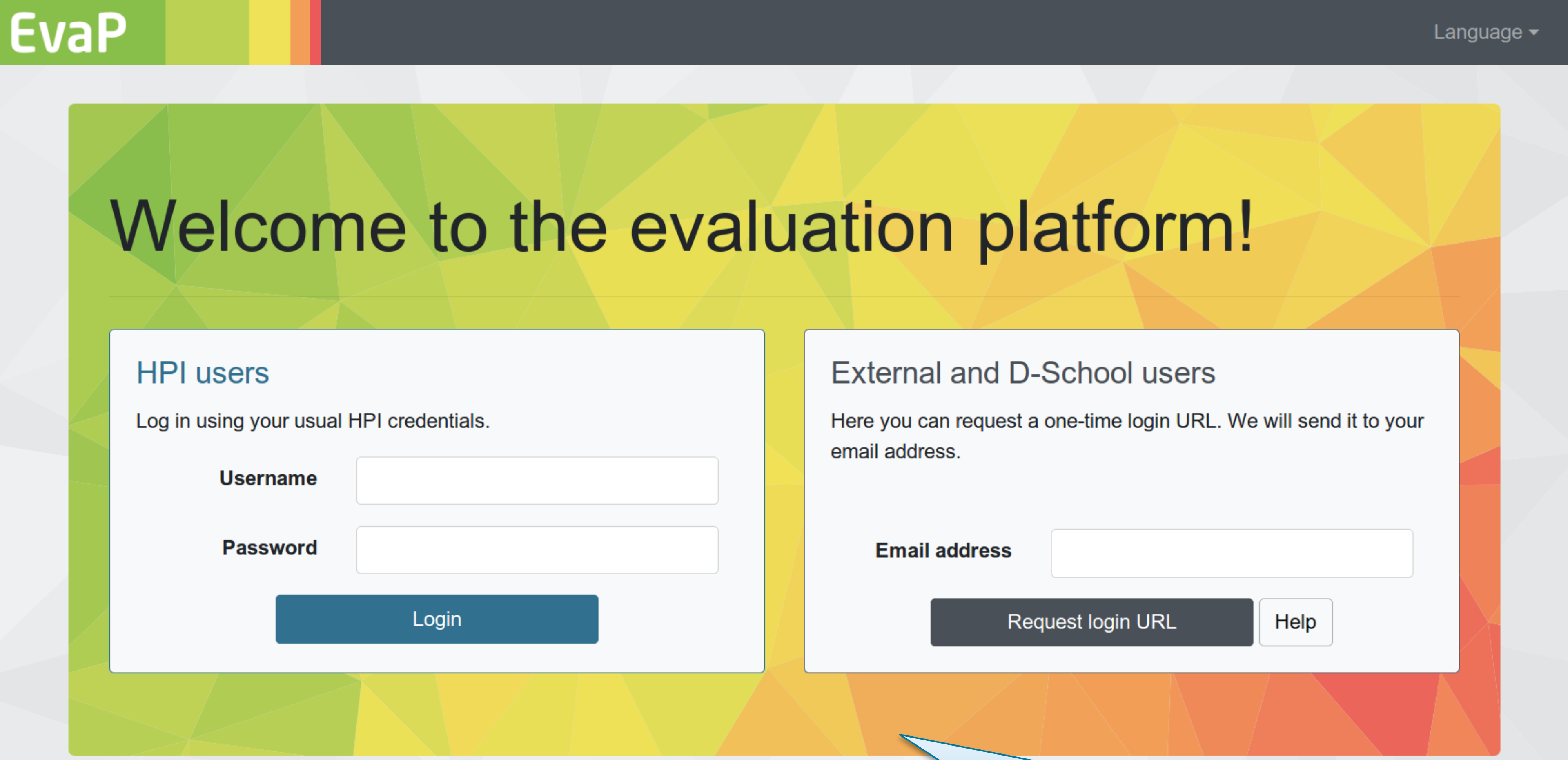
Sustainable Machine Learning on Edge Device Clusters

- Data Preparation
- Data Cleaning
- Data Profiling
- Model Training
- ➢ On three clusters: PI & computer & server

Open positions:

Student Assistant

- DDM 2020 Tutor
- Project Metanome
- Project <?>



Welcome to the evaluation platform!

HPI users

Log in using your usual HPI credentials.

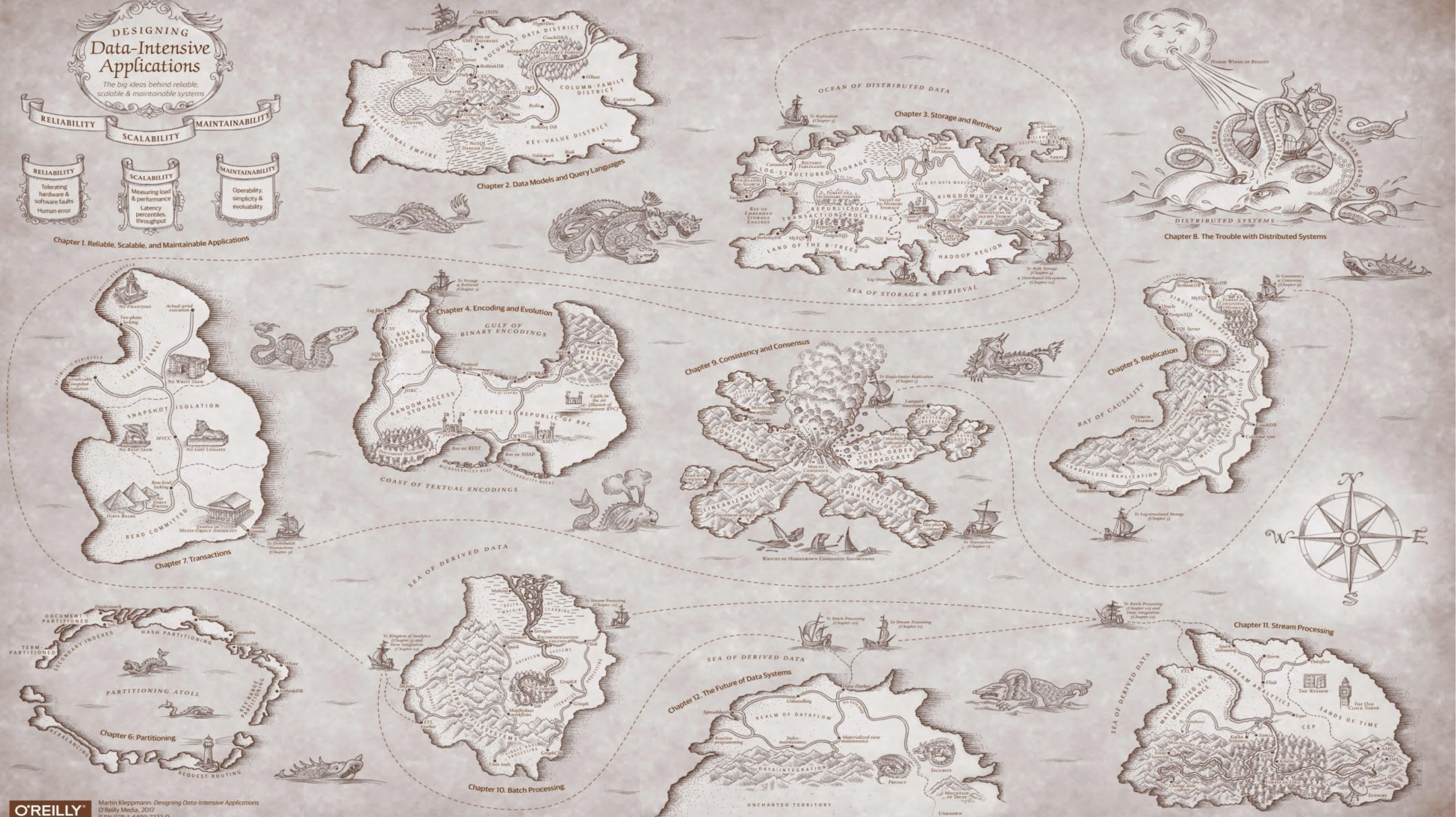
External and D-School users

Here you can request a one-time login URL. We will send it to your email address.

https://evaluierung.hpi.uni-potsdam.de/

DESIGNING Data-Intensive Applications

The big ideas behind reliable, scalable & maintainable systems



RELIABILITY · SCALABILITY · MAINTAINABILITY

RELIABILITY
Tolerating hardware & software faults
Human error

SCALABILITY
Measuring load & performance
Latency percentiles, throughput

MAINTAINABILITY
Operability, simplicity & evolvability

Chapter 1. Reliable, Scalable, and Maintainable Applications

Chapter 2. Data Models and Query Languages

Chapter 3. Storage and Retrieval

Chapter 8. The Trouble with Distributed Systems

Chapter 4. Encoding and Evolution

Chapter 7. Transactions

Chapter 9. Consistency and Consensus

Chapter 5. Replication

Chapter 6: Partitioning

Chapter 10. Batch Processing

Chapter 12. The Future of Data Systems

Chapter 11. Stream Processing

Martin Kleppmann: *Designing Data-Intensive Applications*
O'Reilly Media, 2017
ISBN 978-1-4493-7332-0