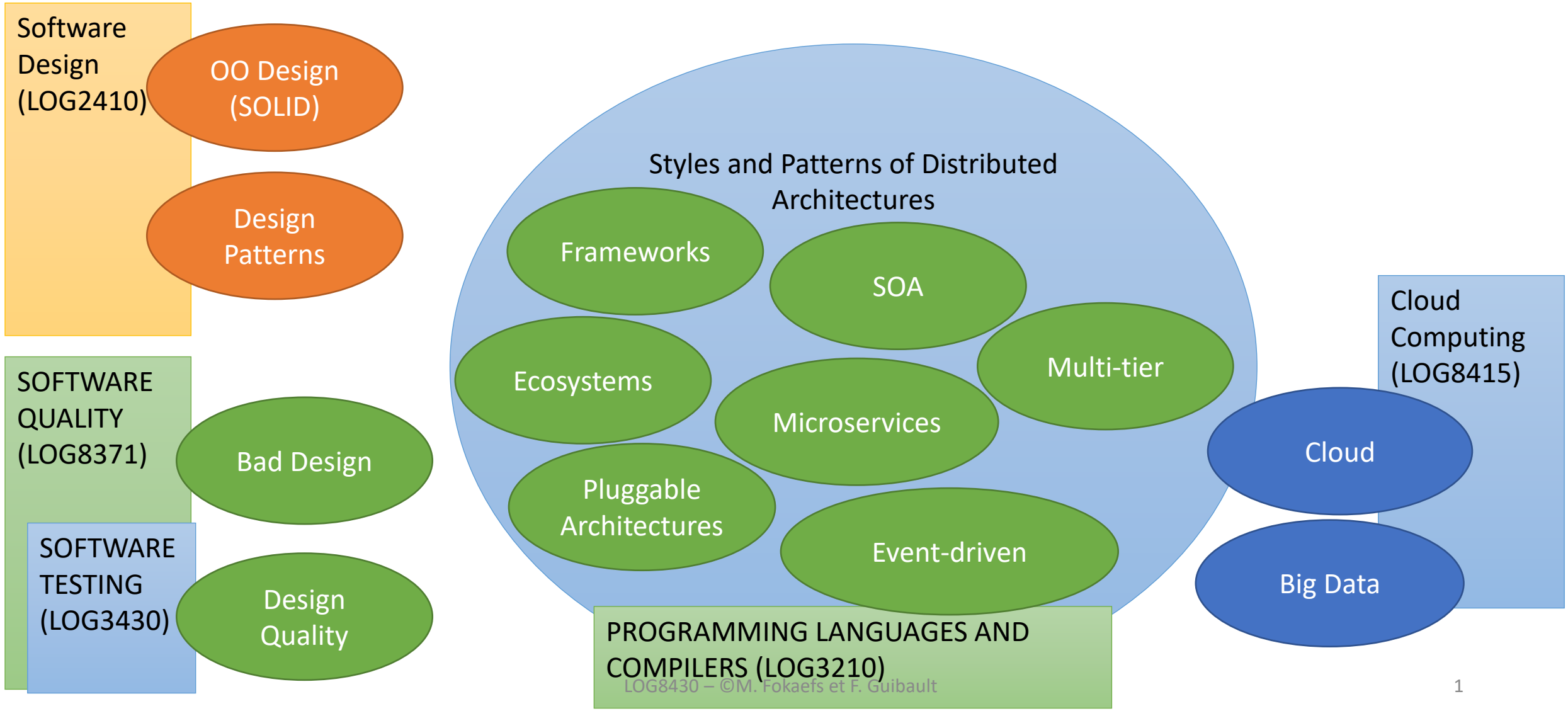
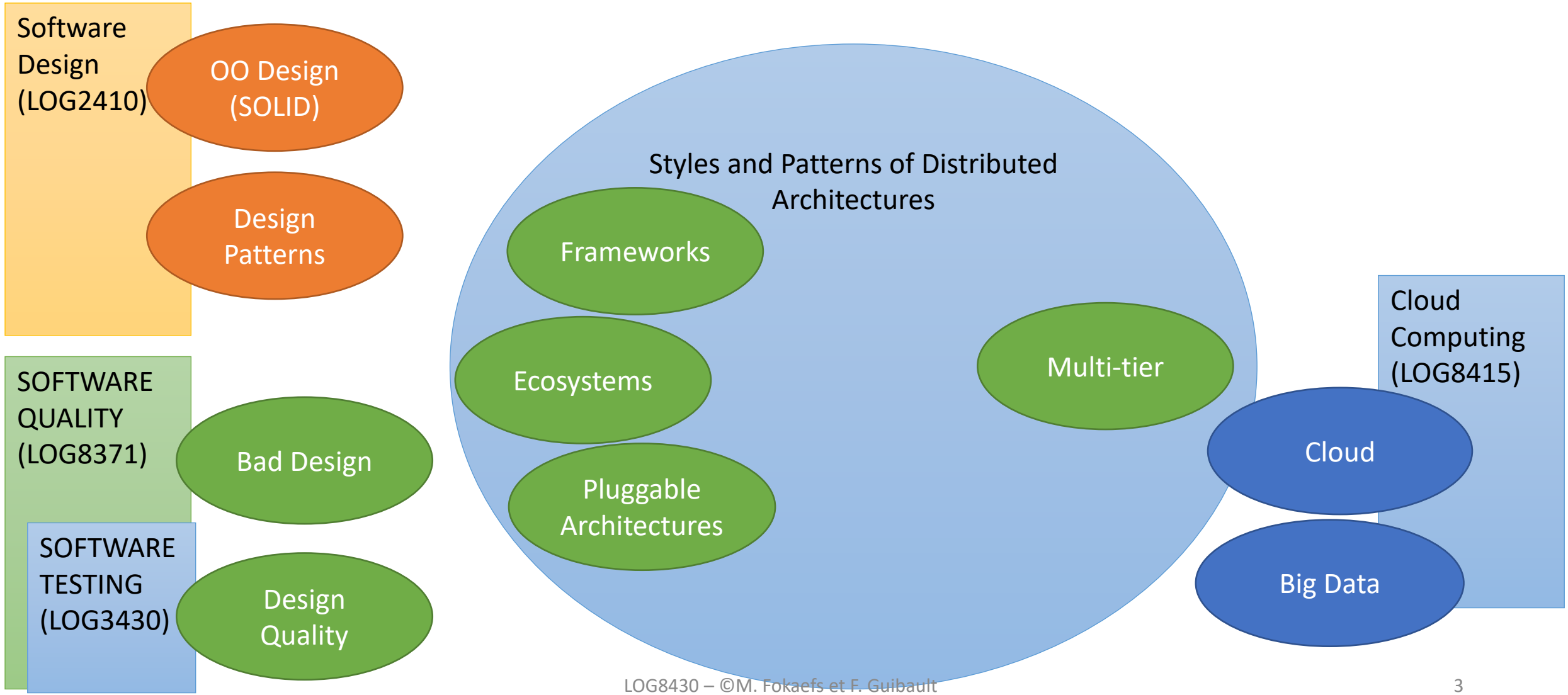
Course Map

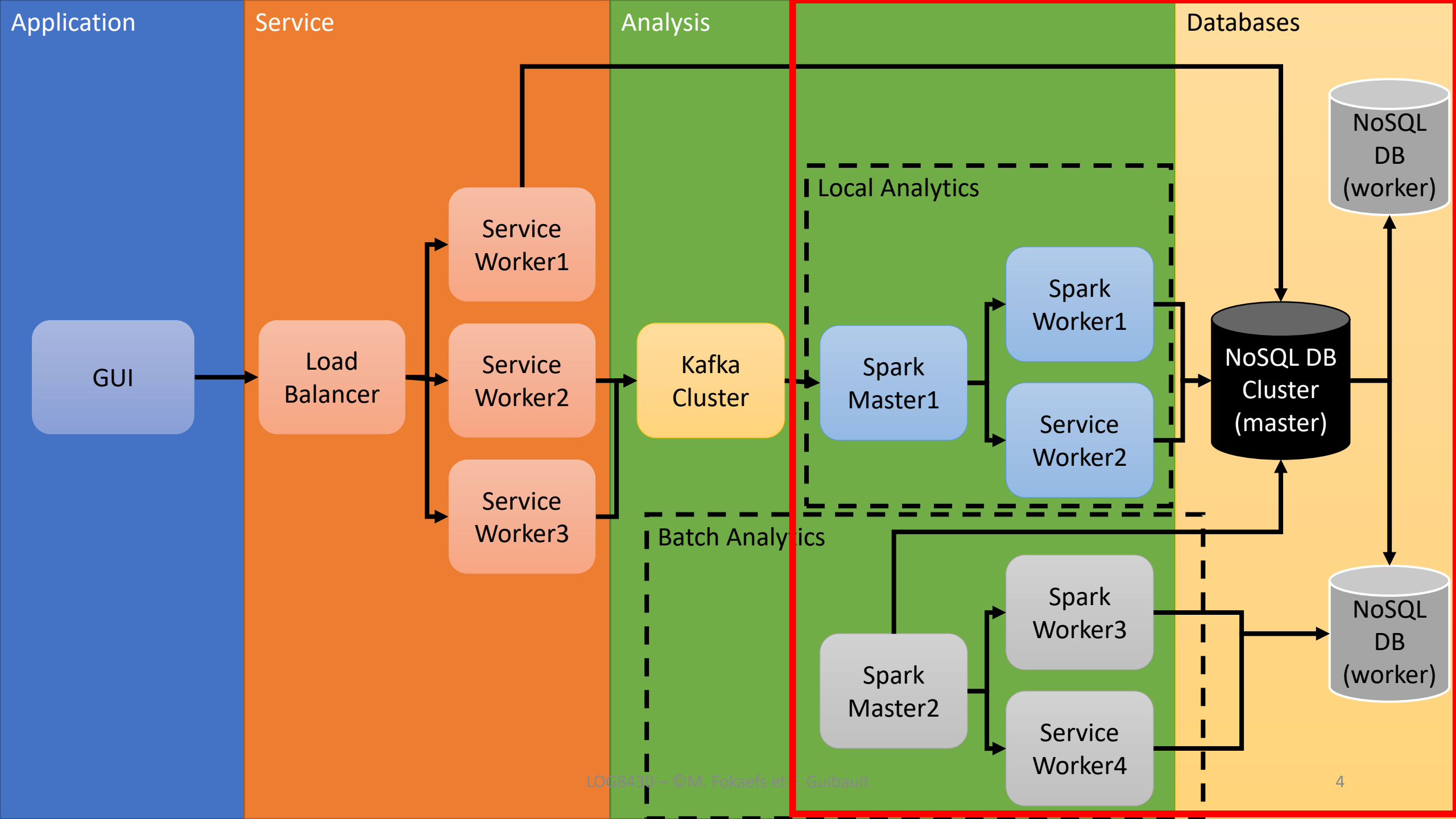

Software Design (LOG2410)
OO Design (SOLID)
Design Patterns
SOFTWARE QUALITY (LOG8371)
Bad Design
SOFTWARE TESTING (LOG3430)
Design Quality
Styles and Patterns of Distributed Architectures
Frameworks
SOA
Ecosystems
Multi-tier
Microservices
Pluggable Architectures
Event-driven
PROGRAMMING LANGUAGES AND COMPILERS (LOG3210)
Cloud Computing (LOG8415)
Cloud
Big Data

LOG8430E: Big Data Architectures

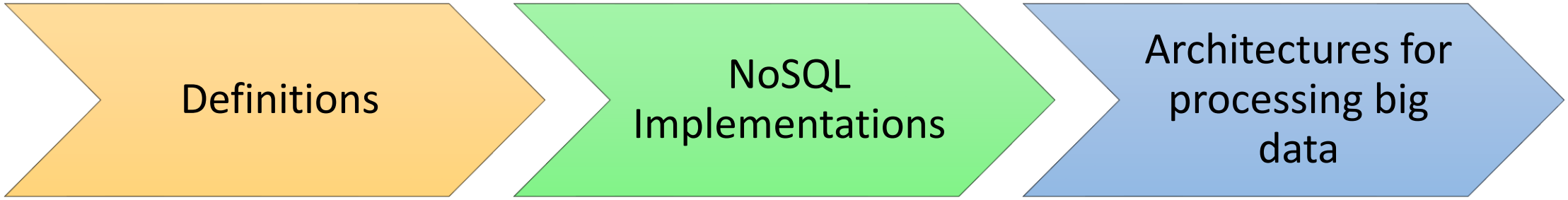
Database systems and architectures and data processing architectures

Today


Software Design (LOG2410)
OO Design (SOLID)
Design Patterns
SOFTWARE QUALITY (LOG8371)
Bad Design
SOFTWARE TESTING (LOG3430)
Design Quality
Styles and Patterns of Distributed Architectures
Frameworks
Ecosystems
Pluggable Architectures
Multi-tier
Cloud Computing (LOG8415)
Cloud
Big Data


Application
Service
Analysis
Databases
GUI
Load Balancer
Service Worker1
Service Worker2
Service Worker3
Kafka Cluster
Local Analytics
Spark Master1
Spark Worker1
Service Worker2
NoSQL DB Cluster (master)
NoSQL DB (worker)
Batch Analytics
Spark Master2
Spark Worker3
Service Worker4
NoSQL DB (worker)

Software architectures for data



- NoSQL
- Distributed databases
- The CAP theorem
- BASE and ACID properties

- Wide-column
- Document-based
- Key-value
- Graph

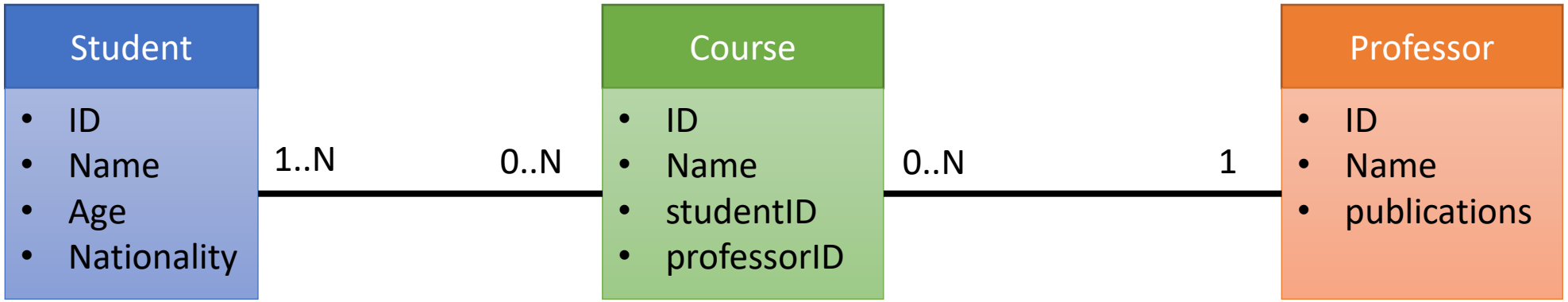
- Lambda architecture
- Kappa architecture

Question

- We want to organize a university. We have students with their attributes (ID, name, nationality, etc.). We also have professors with their attributes (ID, name, publications, etc.). Finally, we have courses that are attended by students are taught by professors.
- Design a system or a data model (an E-R diagram).

Question

- We want to organize a university. We have students with their attributes (ID, name, nationality, etc.). We also have professors with their attributes (ID, name, publications, etc.). Finally, we have courses that are attended by students are taught by professors.
- Design a system or a data model (an E-R diagram).



Problem

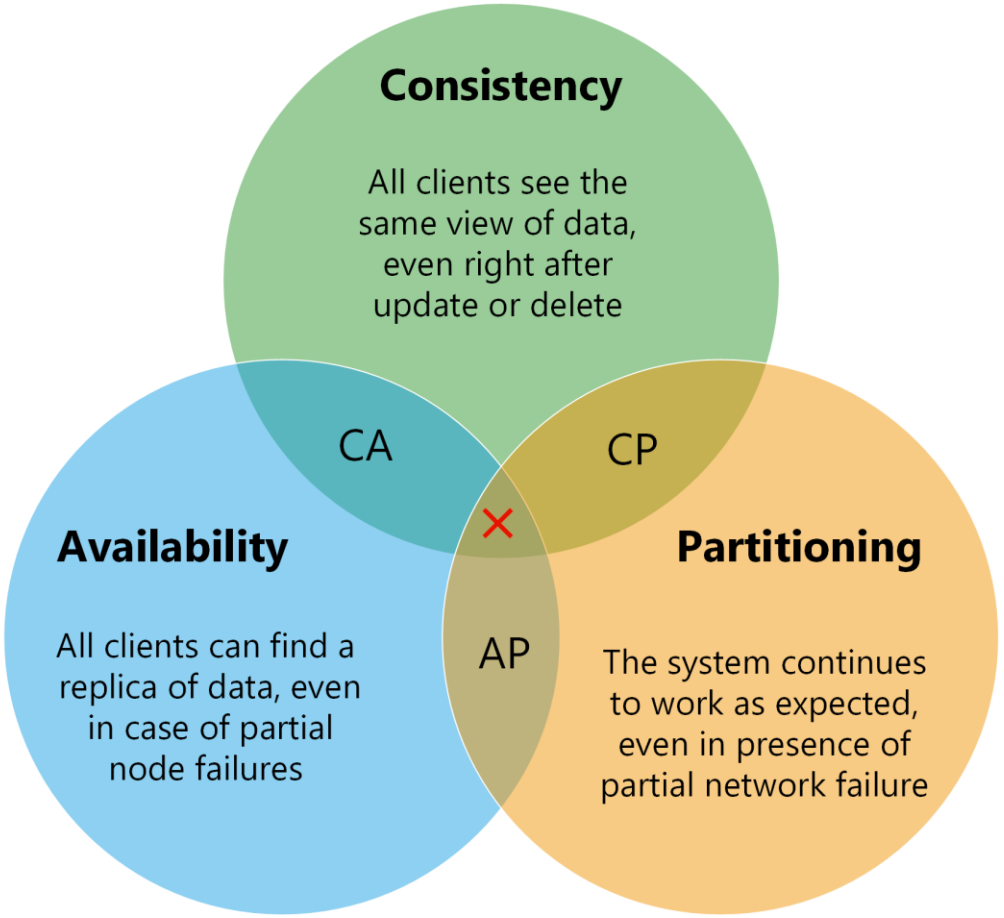
- Now, do not think about students and professors, but think about stars, planets and planet systems.
- There are not just a few objects, but billions.
- Also, we have more types of objects and more complex relationships.
- By exploring the universe, we can discover many more.
- Is it sufficient to use a relational database (RDB)?
- We need a more powerful infrastructure to support the volume and the complexity of data and more flexible to accommodate new types of data.

Distributed databases

- The primary objective of the distributed databases is the management of great quantities of data while guaranteeing an (almost) absolute availability.
- For this reason, the system is installed on multiple servers and the data are partitioned and shared among the nodes.
- The system uses duplication, replication and data partitioning.
  - Replication is used to update the data on all nodes. It is a long and complex procedure.
  - For the duplication, there is a master that contains the database. This master duplicates the database and it ensures that the other nodes have the data. The user can only change the master.
  - The partitions exist between multiple nodes, but the relations between them are well defined. So, it is possible to reconstruct the entire database.
- The distribution and the transactions are opaque to the users.

The CAP theorem

- In favor of availability and performance, distributed systems often sacrifice consistency.
- **C**onsistency: All clients see the same view of the data, even after an update.
- **A**vailability: All clients can obtain a copy of the data.
- **P**artition Tolerance: The system continues to work, even after the failure of a node.
- So, we need to make a decision over consistency vs availability.
- …or maybe not… We will see!



ACID vs BASE

- No this is not a chemistry course!
- The traditional systems (relational) follow the ACID properties for their transactions:
  - **Atomic**: All operations of a transaction succeed or the entire transaction is cancelled.
  - **Consistent**: At the end of a transaction, the database is structurally correct.
  - **Isolated**: The transactions are not concurrent. The legal accesses to data are governed by the database, so that the transactions seem to be executed sequentially.
  - **Durable**: The results of the application of a transaction are permanent, even in case of failure.
- The distributed systems can follow the ACID properties, but more often they follow the BASE properties:
  - **Basic Availability**: The database seems to function most of the time.
  - **Soft-state**: The databases do not need to be consistent at writing, no more than the replicas need to be consistent with each other.
  - **Eventual Consistency**: The databases present a consistency at a future moment (for example, idly at read time).

Implementations of distributed databases: NoSQL

- NoSQL databases have become very popular in the era of big data.
- They expose neither SQL interfaces nor normalized interfaces.
  - This increases the efficiency and the flexibility in the definition of queries.
- They also use multiple types of data organisations.
  - This increases a lot the efficiency of the system.
- But, isn’t it possible to have distributed relational databases?
  - Yes, there are many such systems: MySQL Cluster, IBM Db2
  - They offer performance advantages, but they are different than NoSQL systems.

NoSQL: Types of data organisation

- Key-value
- Document
- Graph
- Wide Column

Key-value

- The data is organized in pairs of keys and values.
- The values follow the notion of objects, as in the case of object-oriented programming.
  - They can be complex enough.
- The principal difference between KV and RDB is that the former does not have a schema that defines the structure of the values.
- So, we can have different attributes or missing attributes between the objects of the same type.
  - This offers increased flexibility.
- The data can be partitioned and shared among the nodes by using the keys.

KV: Properties, advantages, implementations

- These systems are very efficient for simple applications.
- It is possible to store all data in memory, which increases efficiency.
  - It is still possible to use a persistent storage, i.e., the disk.
  - It is also possible to use a hybrid method.
- They can support complex data by offering some efficiency in searching values based on the keys.
- Implementations
  - BerkeleyDB
  - Redis
  - Memcached
  - Riak
  - Voldemort

Document

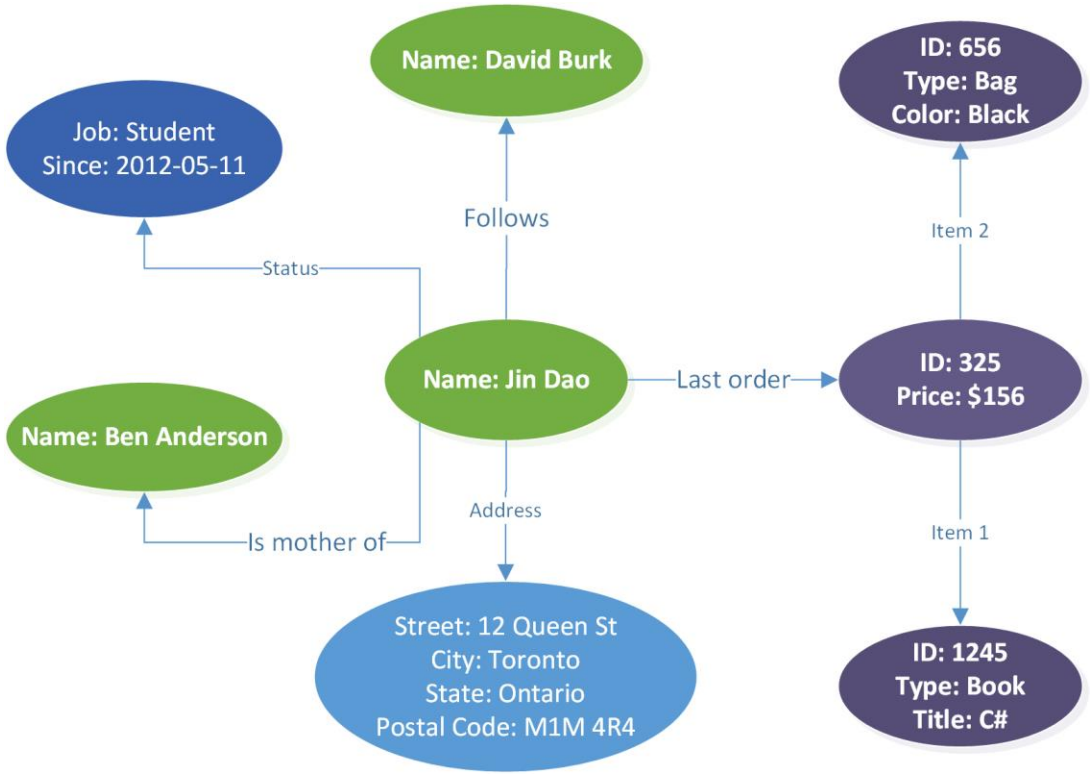
- These systems treat data as files.
- The files have a specific encoding, e.g., JSON, XML, etc.
- We can ask for documents with the help of keys.
- The documents can be of various sizes and complexities.

Document: Properties, advantages, implementations

- These systems are appropriate for applications, where data is documents (of course!), but also when we have data having a variable structure.
- Contrary to KV, where values are opaque, here, we can ask for documents as a function of their attributes.
- We can also define matadata to facilitate the retrieval of files.
- In this case, the keys are the primary indices and we can define secondary indices in the form of values of attributes or of metadata.
- Implementations
  - CouchDB
  - MongoDB
  - RavenDB

Graph

- In this systems, the data is represented as graphs.
- They respond to the need to combine a graphic model of the data with the relational tables in RDB.
- The nodes represent the entities and the edges the relationships. The nodes can have attributes.
- They allow to store any relation in a practical and efficient manner.

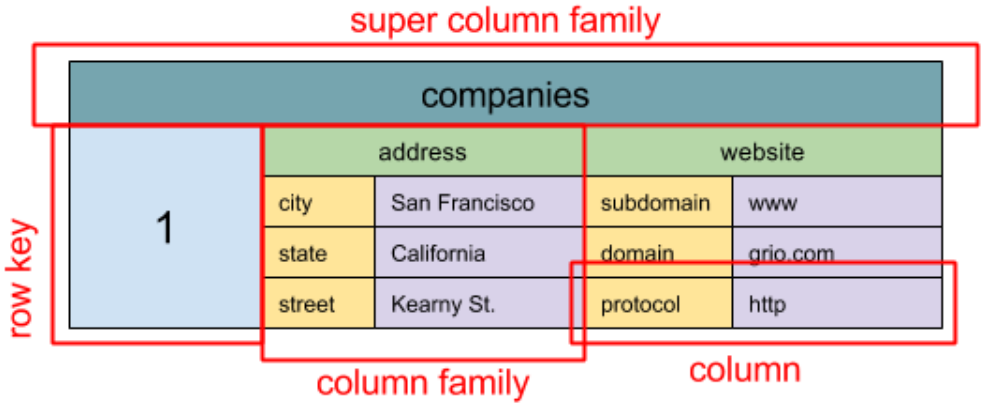


Graph: Properties, advantages, implementations

- The graphs are present everywhere!
- Often, we need to store graph data, while preserving their form as a graph.
- There are multiple algorithms to traverse or analyse the graphs that are already efficient.
- Certain queries are not possible with traditional query languages.
- The power and the capacity of the system depend on the underlying implementation.
  - There are systems that are implemented on top of RDB or KV.
- Implementations
  - Neo4j
  - OrientDB
  - Titan

Wide-column

- They are the systems the closest to relational databases, at least on a conceptual level.
- They conserve the concepts of tables, rows and columns.
  - This gives a sense of a schema
- The relational concepts form an interface for clients.
  - On an implementation level, the systems are based on distributed file systems.
- The definition of columns is very flexible.
  - We group multiple attributes in column families
  - We can omit certain columns.



Wide-column: Properties, advantages, implementations

- This type of DB facilitates the migration of relational databases towards distributed systems.
- The additional flexibility offered by the column families improve the maintainability of data and the management of variable data.
- They can support enormous data quantities.
- Wide-column systems are designed to function with distributed systems for big data analyses, such as MapReduce and Hadoop. They are designed to offer:
  - High availability and security.
  - Fast read and write.
- Implementations
  - HBase
  - Accumulo
  - Cassandra

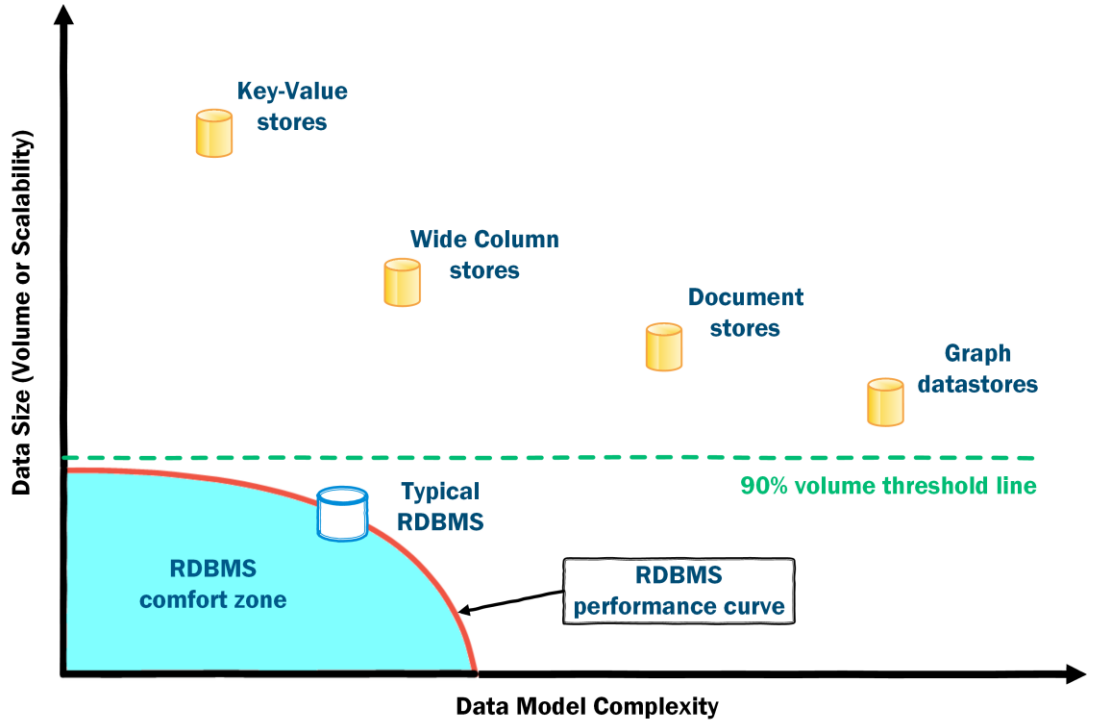
How to select a NoSQL solution?
According to the type of the application

- Key-value
  - To put data in cache memory
  - When we have sequential data (lists, queues, etc.)
  - To support publish/subscribe systems.
  - To maintain a state.
- Document
  - With very complex and nested data structures.
  - When we have JavaScript clients.
- Graph
  - When the relationships between data are very complex.
  - To work with hierarchies and taxonomies.
- Wide-column
  - When we have unstructured data, that does not change much and that needs to be stored for a long time.
  - When we need scalable storage.

How to select the NoSQL solution?

According to the properties of the data

- Data model and access pattern.
- Requirements of queries
- Non-functional properties (performance, security, scalability etc.)



Processing architectures

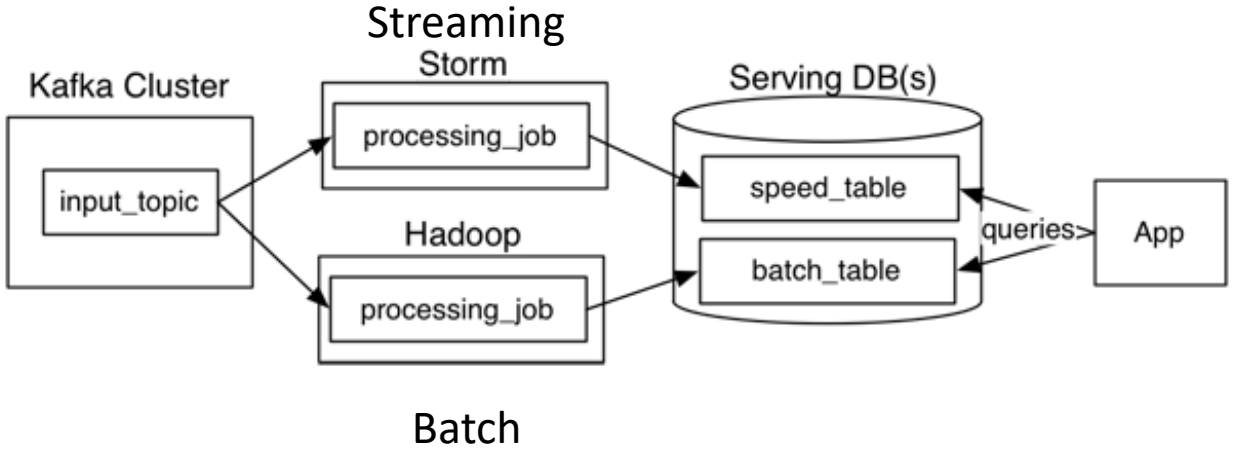
- Lambda architecture
  - “How to beat the CAP theorem”, October 2011
- Kappa architecture
  - “Questioning the Lambda Architecture”, July 2014

Lambda Architecture

- Remember! According to the CAP theorem, we cannot guarantee both availability and consistency at the same time.
  - We consider that partition tolerance is guaranteed by default.
- If we guarantee the availability, what will be the problem with consistency?
  - It is costly and slow to update all replicas of enormous data quantities.
- But, we could reduce…
  - The quantity of data?
  - Or the number of replicas?
- We could divide data in volatile and immutable.

Lambda Architecture

- For volatile data, we can guarantee availability.
  - We can keep few replicas and apply incremental analyses on smaller quantities of data.
  - We analyse the data in real-time (streaming).
- For immutable data, we can guarantee consistency.
  - By using the results of real-time analytics, we render the data immutable.
  - We apply more complex and costly analyses on these data (batch processing).
  - We favor accuracy over performance.



Lambda Architecture

Advantages

- We can guarantee that certain data will be consistent and other available.
- We guarantee the performance of real-time analyses.
- We guarantee the accuracy of the analyses on the majority of data.
- A scalable and fault tolerant architecture.
- A nice balance between performance and reliability.

Disadvantages

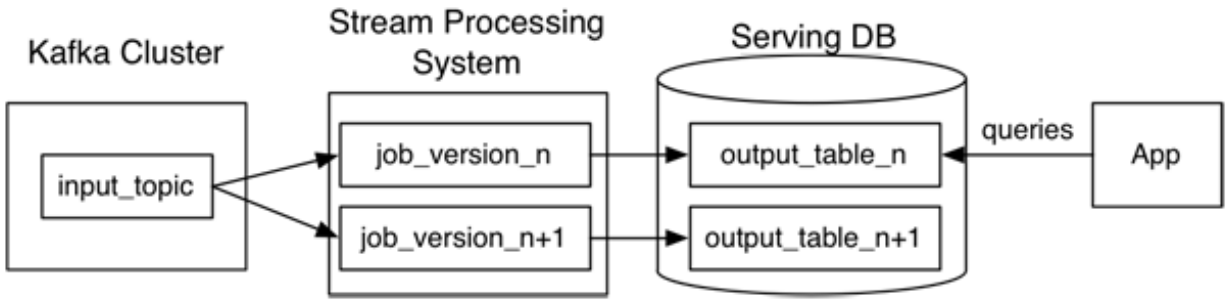
- Coding can be a complicated process.
- The data model can be difficult to migrate or maintain.
- The repetition of data processing (streaming and batch) creates a performance overhead.

Kappa Architecture

- The reprocessing of each data stream is not always necessary.
- The processing may be necessary only when the logic of the analysis changes.
- It is possible to predict and plan the reprocessing.
- There are applications that do not require the persistent storage of data for a long time.

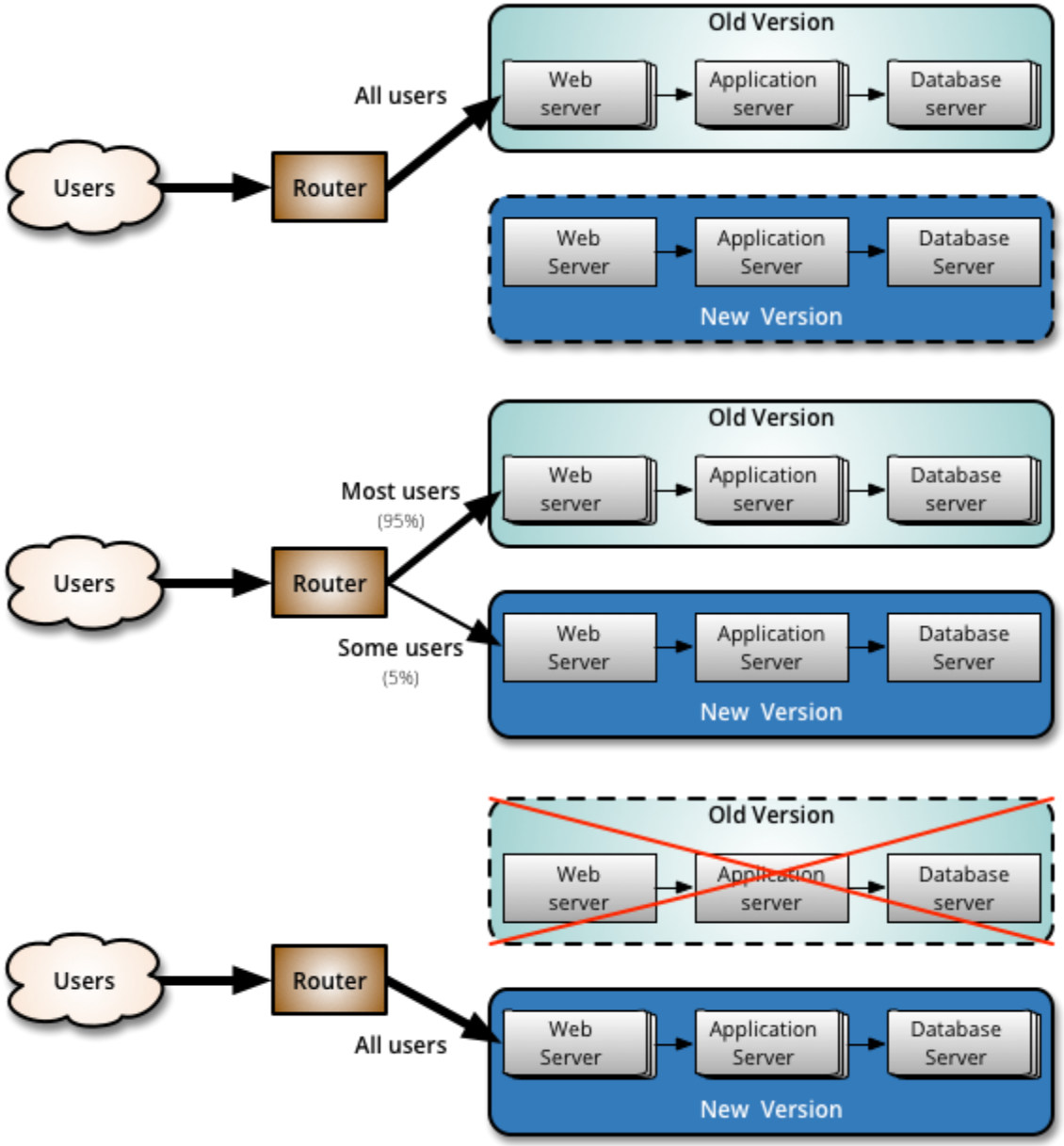
Kappa Architecture

- We keep just the level of streaming.
- We use message-oriented systems to manage the input of data and facilitate reprocessing.
- The solution offers an increased flexibility with respect to maintenance and evolution of analysis algorithms.
  - We can change the algorithm and reprocess the data with the new analysis.



Intermission: Canary Deployment

- We can replace an algorithm or a service with a new version at real-time.
- Deploy the two versions together.
- Gradually, redirect the requests towards the new version.
- When all requests are served by the new version, stop the old version.



Kappa Architecture

Advantages

- Increased flexibility for coding.
- Increased simplicity for the data model.
- Ease of development of online learning systems that do not require persistent storage.
- Use of memory for increased efficiency.

Disadvantages

- Reduced fault tolerance and recoverability.

How to select a processing architecture?

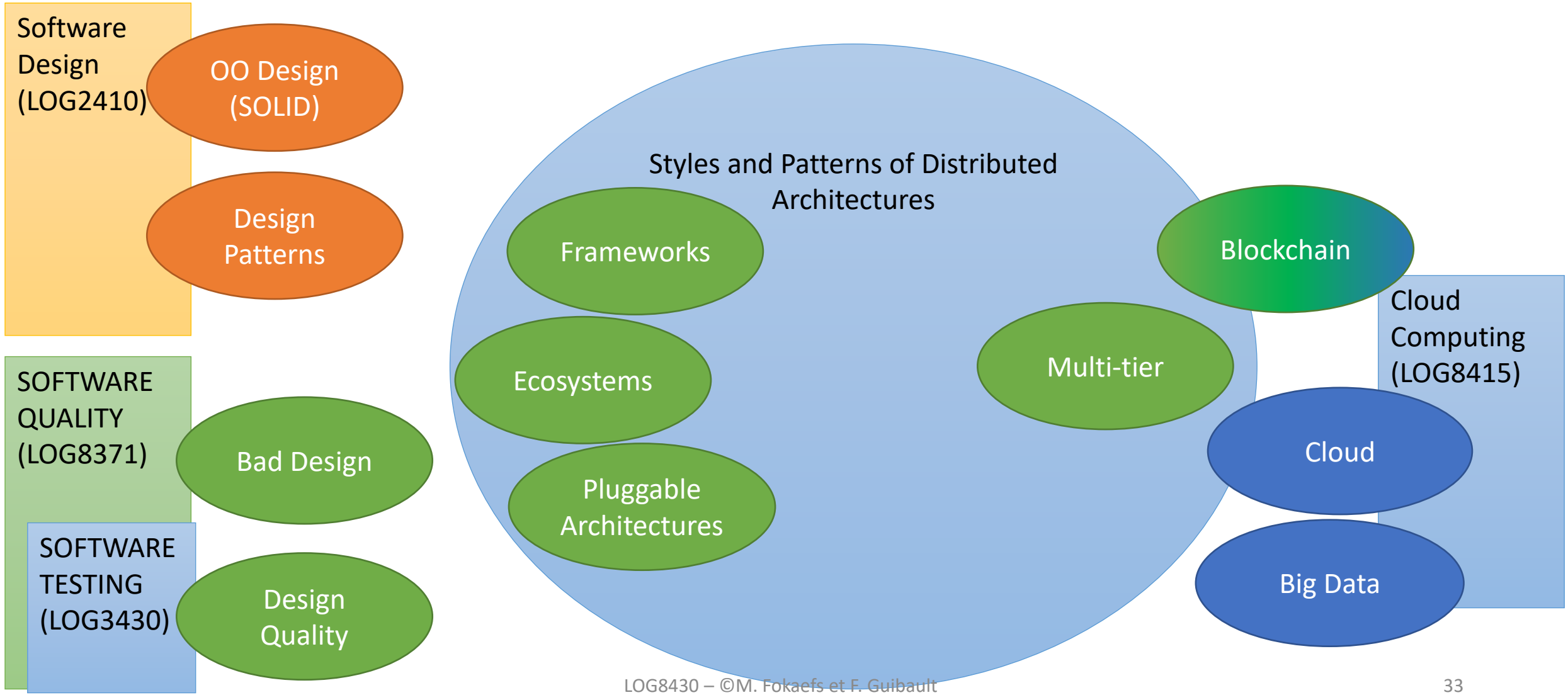
Lambda

- User requests need to be served on an ad-hoc basis using the immutable data storage.
- Fast responses are necessary and the system needs to manage various updates in the form of new data streams.
- No stored data should be deleted and this should allow the addition of updated and new data in the database.

Kappa

- Several events or queries on the data are queued to be processed based on storage in a distributed file system.
- The order of events or queries is not predetermined. The stream processing platforms can interact with the database at all moments.
- A need for resiliency and high availability because terabytes of storage management is required for each node of the system to support replication.

Next time


Software Design (LOG2410)
OO Design (SOLID)
Design Patterns
SOFTWARE QUALITY (LOG8371)
Bad Design
SOFTWARE TESTING (LOG3430)
Design Quality
Styles and Patterns of Distributed Architectures
Frameworks
Ecosystems
Pluggable Architectures
Blockchain
Multi-tier
Cloud Computing (LOG8415)
Cloud
Big Data