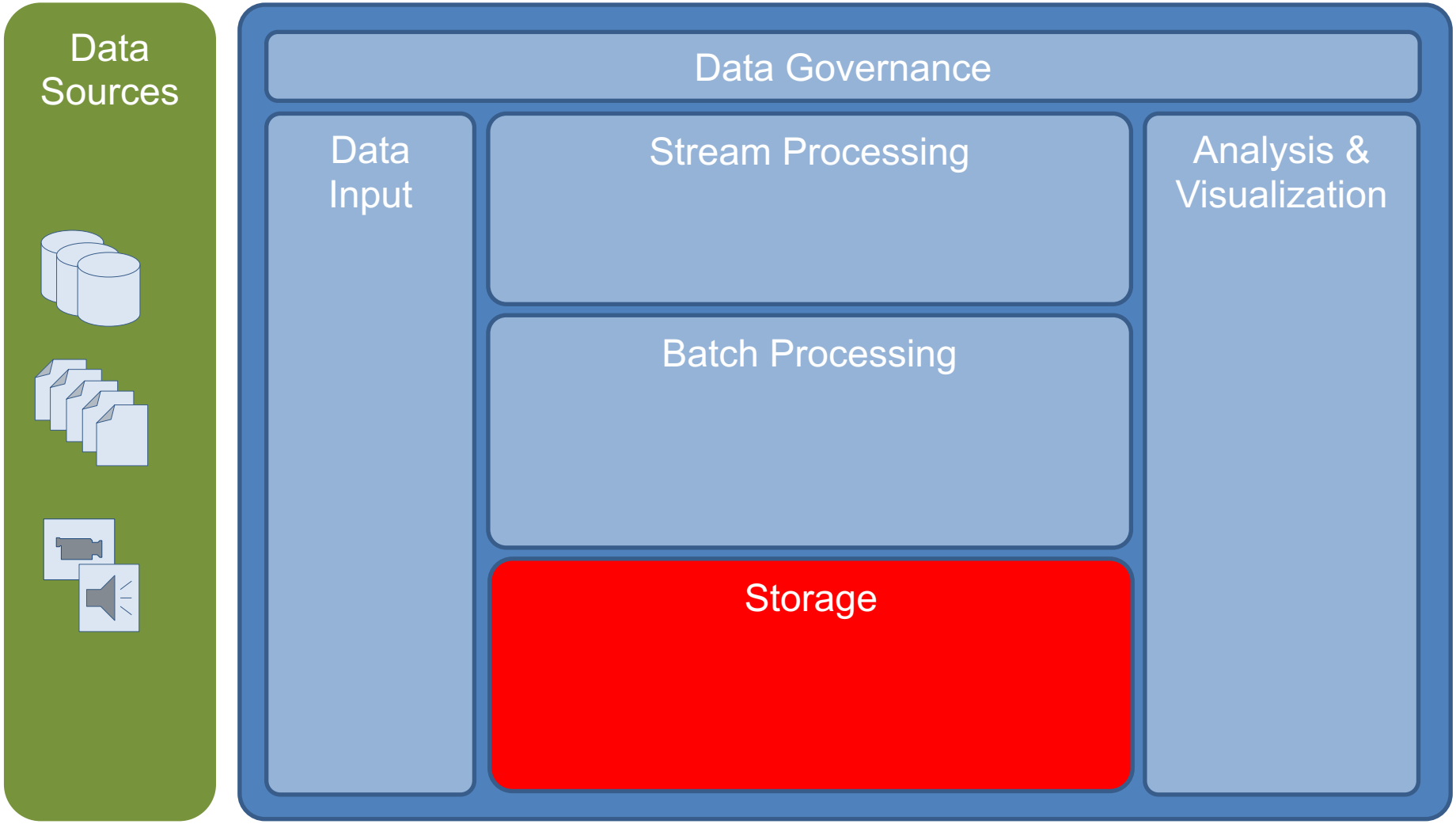
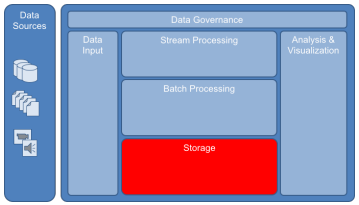
Big Data

Storage

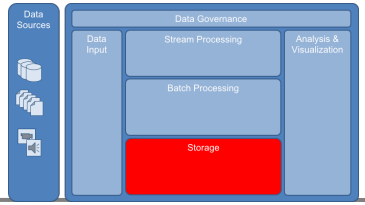
Big-Data Referenzarchitektur



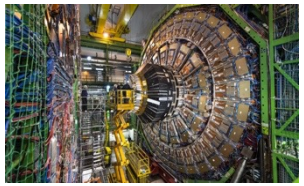


DATA STORAGE

Die Herausforderung



Volume

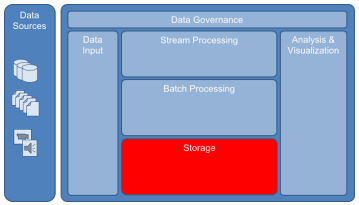


Velocity

Variety

- Speicherlösungen müssen einfach skalierbar sein um große Datenvolumen aufnehmen zu können
- Sie müssen performant sein
- Sie müssen unterschiedliche Datenformate unterstützen

Skalierbarkeit



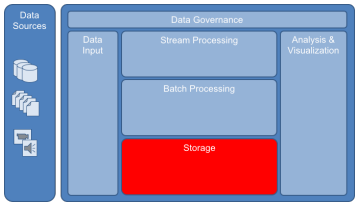
Vertikal (Scale Up):

- Steigerung der Leistung eines Rechners durch Hinzufügen von zusätzlichem RAM, mehr / schnellere CPUs etc.
- Nicht von Software abhängig
- Hat eine natürliche Grenze

Horizontal (Scale Out)

- Hinzufügen zusätzlicher Knoten
- Software muss geeignet sein
- Dann im Prinzip unbegrenzt

Speicherarten



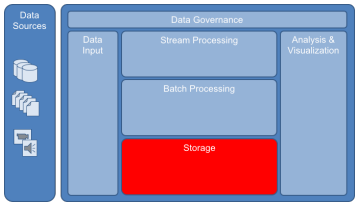
´Small Data´

- Filesystem:
  - speichert und organisiert Daten auf einem Speichermedium (Festplatte, Memorystick etc.),
  - liefert logische Sicht als Baumstruktur o.Ä.
  - Beispiele (NTFS, ext4, APFS)

Big Data

- Distributed File System
  - Kann große Files verteilt über mehrere Cluster-Nodes speichern
  - Files erscheinen in der logischen Sicht wie lokal
  - Beispiele Hadoop Distributed File System (HDFS, Google File System (GFS)

Speicherarten



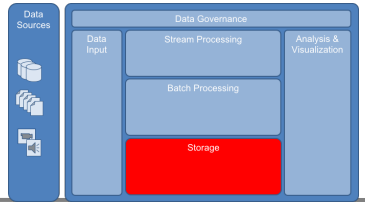
´Small Data´

- Relationale DB:
  - speichert Daten normalisiert
  - Relationales Datenmodell
  - Datenaufruf mittels SQL
  - Beispiele: Oracle Database, (Microsoft) SQLServer, (SAP) HANA, (IBM) DB2

Big Data

- Auch RDBMS
- NoSQL (not only SQL)
  - Nicht-relationale Datenbank
  - Kann auch semi- und unstructured data aufnehmen
  - Hoch skalierbar, kann z.B. auf HDFS aufsetzen
  - Datenaufruf mittels API und diversen Query Languages (XQuery, SPARQL, SQL etc.)
  - Beispiele: (Hadoop) Hbase, (Apache) Cassandra, Google BigTable, (AWS) SimpleDB

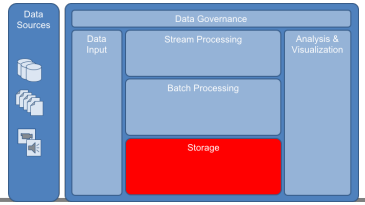
Skalierbarkeit von Speicher



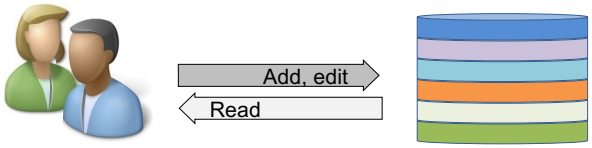
Wie verteile ich Daten auf mehrere Speicherknoten?

1. Sharding (verteilen)
2. Replication (vervielfältigen)
3. Kombination von 1 und 2

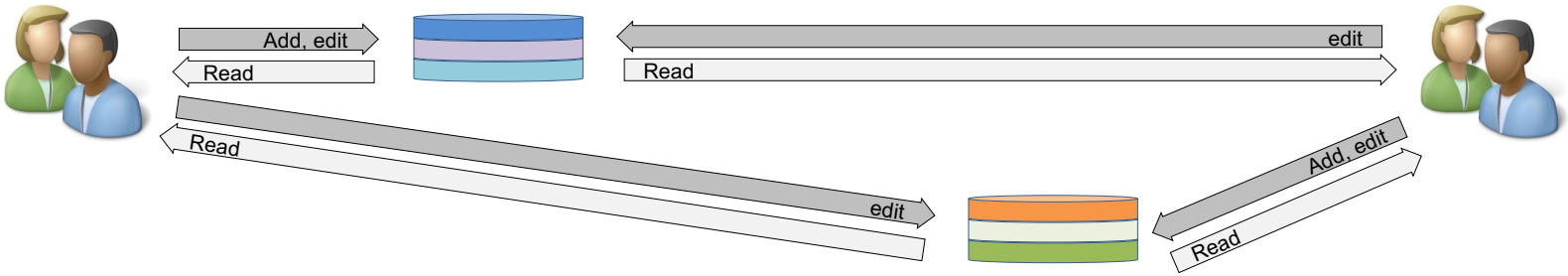
Sharding



Nicht verteilt:

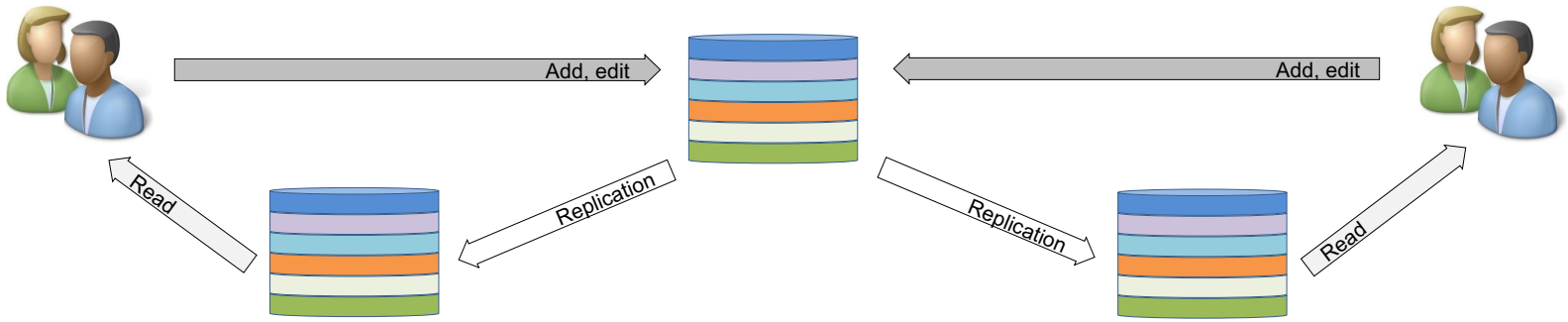


Sharding:



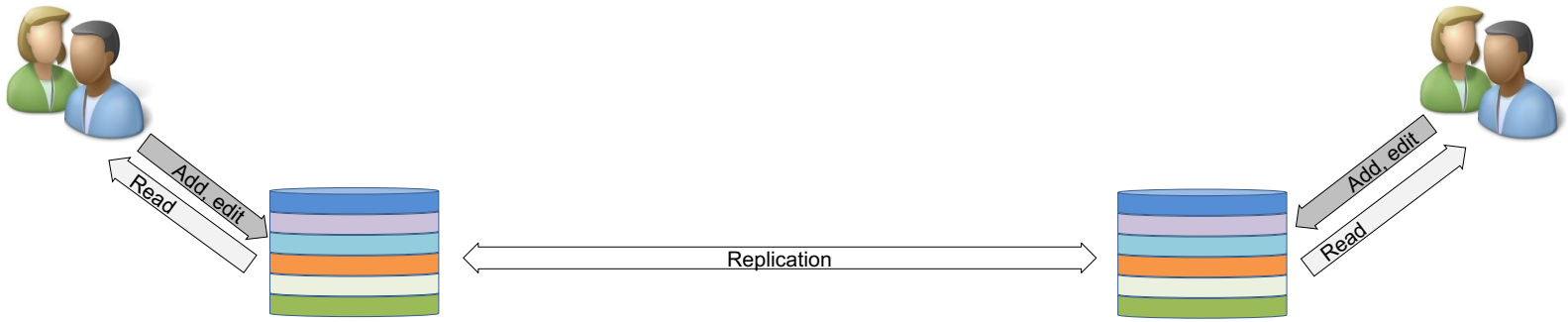
Replikation

Primary-Replica Replication*:



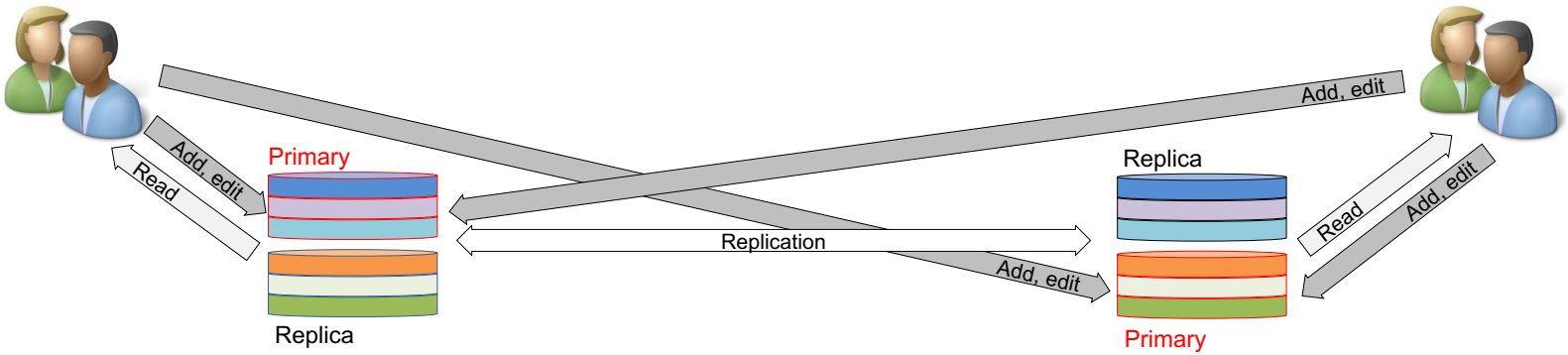
*: “Master-Slave” sollte man nicht mehr verwenden, allerdings gibt es noch keine Begriffe, die sich wirklich durchgesetzt haben:
https://www.theserverside.com/opinion/Master-slave-terminology-alternatives-you-can-use-right-now
https://www.washingtonpost.com/opinions/2020/06/12/tech-industry-has-an-ugly-master-slave-problem/

Peer-to-Peer Replication:

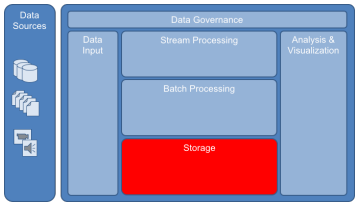


Kombination Sharding - Replication

…eine Möglichkeit:



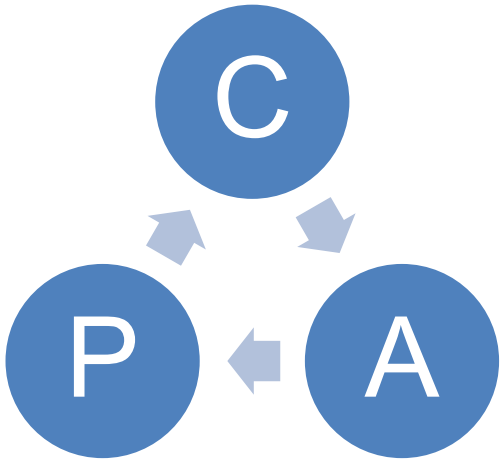
Man kann nicht alles haben: CAP



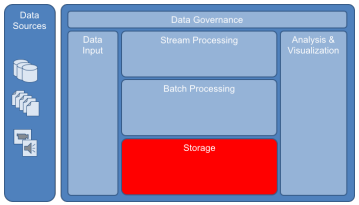
CAP-Theorem (Brewer´s Theorem)

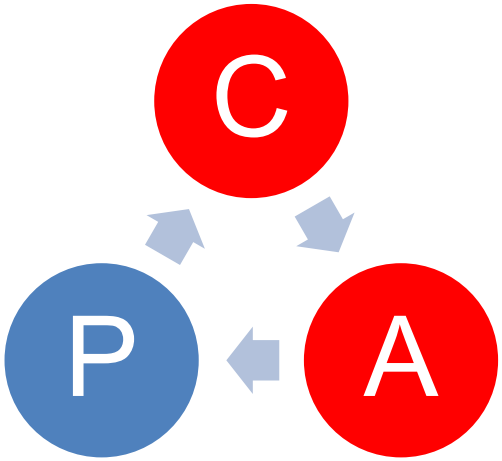
Bei einem verteilten Datenbanksystem, das auf einem Cluster läuft, kann man nur 2 der folgenden Eigenschaften erfüllen:

- **Consistency:** Man kann von jedem Knoten des Clusters lesen und bekommt immer das gleiche Ergebnis
- **Availability:** Die Knoten und die darauf befindlichen Daten sind immer erreichbar und abrufbar (Lesen und Schreiben)
- **Partition Tolerance:** Das System funktioniert auch noch, wenn wenn die Verbindung zu einzelnen Knoten unterbrochen ist



Man kann nicht alles haben: CAP

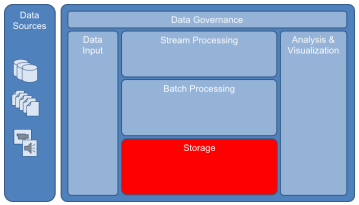


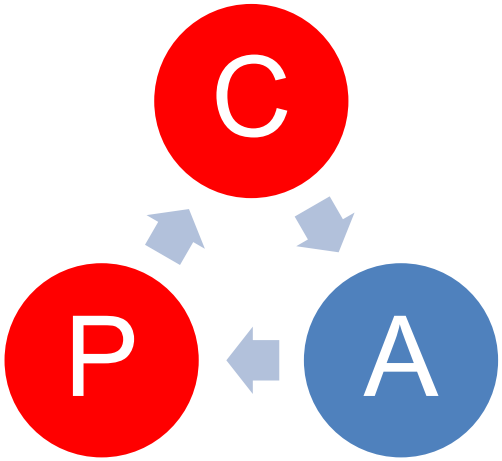


Wenn Consistency und Availability benötigt werden, müssen die Knoten Informationen austauschen können

⇒ Partition Tolerance ist nicht gegeben

Man kann nicht alles haben: CAP

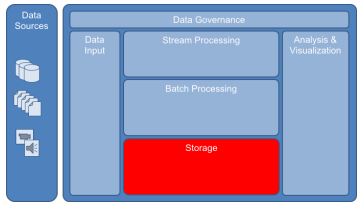


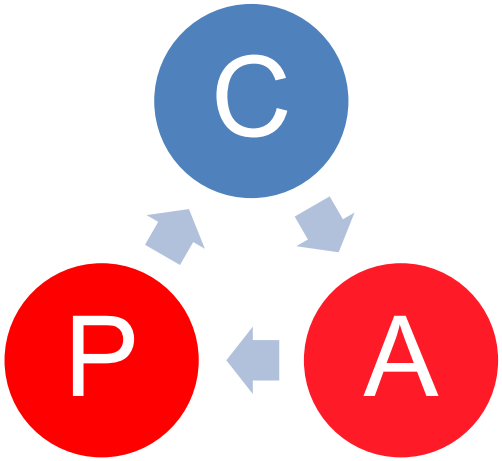


Wenn Consistency und Partition Tolerance benötigt werden, müssen die Knoten Zeit haben, um ihren Inhalt zu synchronisieren. Während dieser Zeit ist der Service nicht verfügbar.

⇒ Availability ist nicht gegeben

Man kann nicht alles haben: CAP

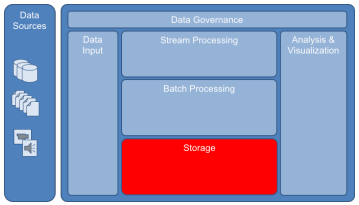




Wenn Availibiltiy und Partition Tolerance benötigt werden, kann es vorkommen, dass neue Inhalte noch nicht zwischen den Knoten synchronisiert sind. Das Resultat wären unterschiedliche Ergebnisse zwischen den einzelnen Knoten

⇒ Consistency ist nicht gegeben

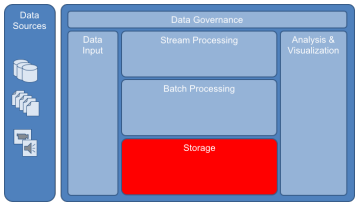
ACID



Datenbank –Designprinzip für Transaction Management

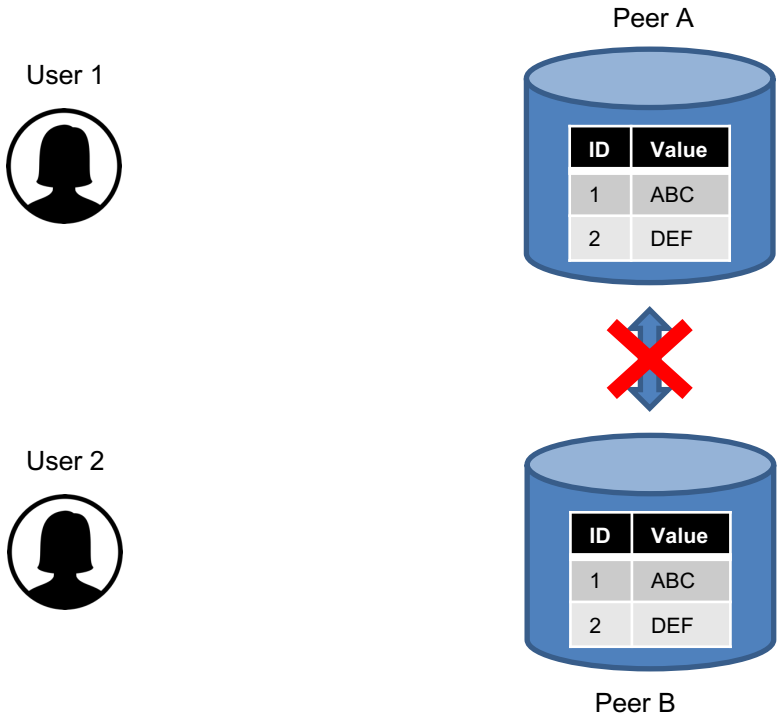
- Atomicity: eine Transaktion ist immer als ganzes erfolgreich oder nicht, d.h. wenn es zwischendrin einen Fehler gibt, findet ein Rollback statt.
- Consistency: alle Daten werden vorher validiert, d.h. nur Daten, die dem Datenbank-Schema und seinen Einschränkungen entsprechen, werden gespeichert. Die Datenbank als ganzes ist stets konsistent.
- Isolation: das Ergebnis einer Transaktion ist nicht sichtbar für andere bevor die Transaktion nicht vollständig durchgeführt ist. Andere Transaktion auf den selben Objekten sind solange nicht möglich.
- Durability: wenn eine Transaktion commited wurde, dann ist sie auch bei einem Systemfehler permanent.

BASE

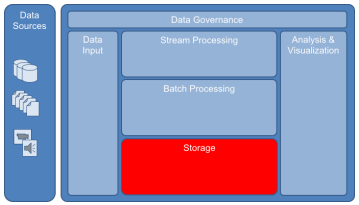


Mögliches Design-Prinzip für eine verteilte Datenbank

- Basically Available: Die Datenbank gibt immer Daten oder eine Fehlermeldung zurück
- Soft State: Es kann aber sein, dass die Daten, wenn Sie gelesen werden inkonsistent sind.
- Eventual Consistency: Bei einem 2. Lesen kann das Ergebnis anders sein, da die Daten inzwischen synchronisiert wurden

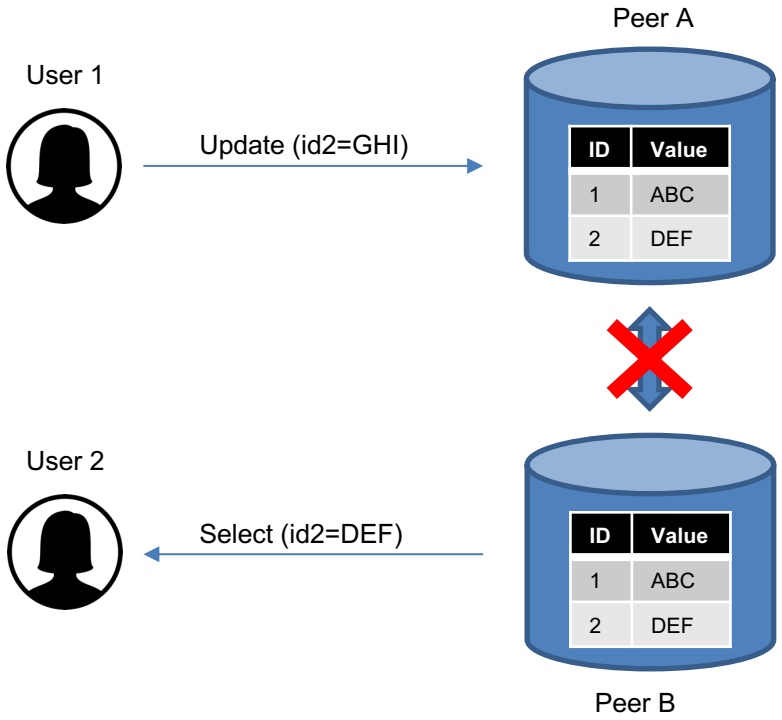


BASE

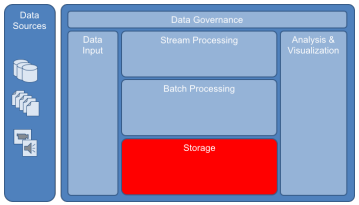


Mögliches Design-Prinzip für eine verteilte Datenbank

- Basically Available: Die Datenbank gibt immer Daten oder eine Fehlermeldung zurück
- Soft State: Es kann aber sein, dass die Daten, wenn Sie gelesen werden inkonsistent sind.
- Eventual Consistency: Bei einem 2. Lesen kann das Ergebnis anders sein, da die Daten inzwischen synchronisiert wurden

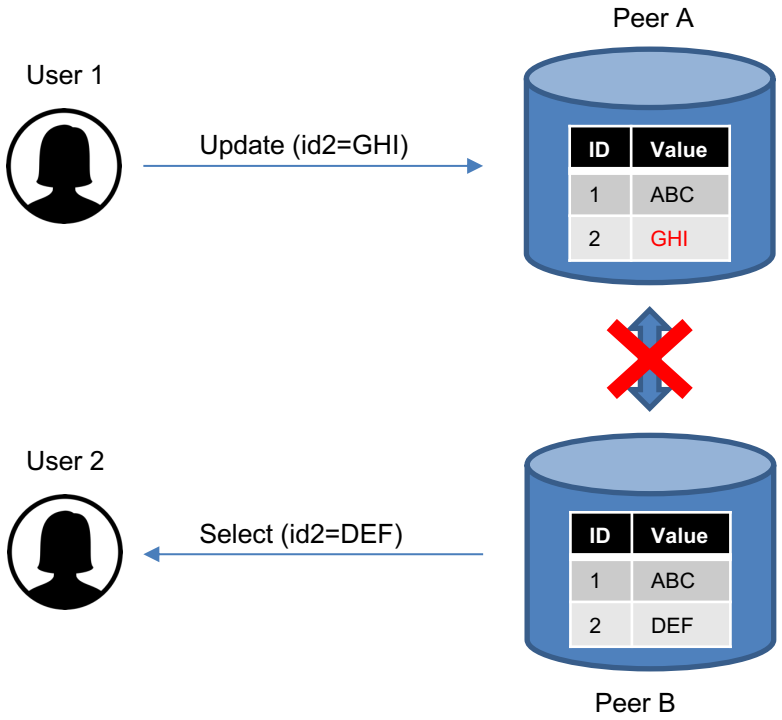


BASE

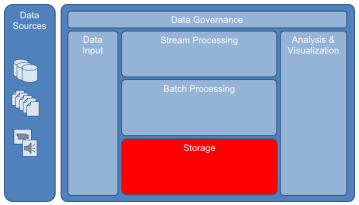


Mögliches Design-Prinzip für eine verteilte Datenbank

- Basically Available: Die Datenbank gibt immer Daten oder eine Fehlermeldung zurück
- Soft State: Es kann aber sein, dass die Daten, wenn Sie gelesen werden inkonsistent sind.
- Eventual Consistency: Bei einem 2. Lesen kann das Ergebnis anders sein, da die Daten inzwischen synchronisiert wurden

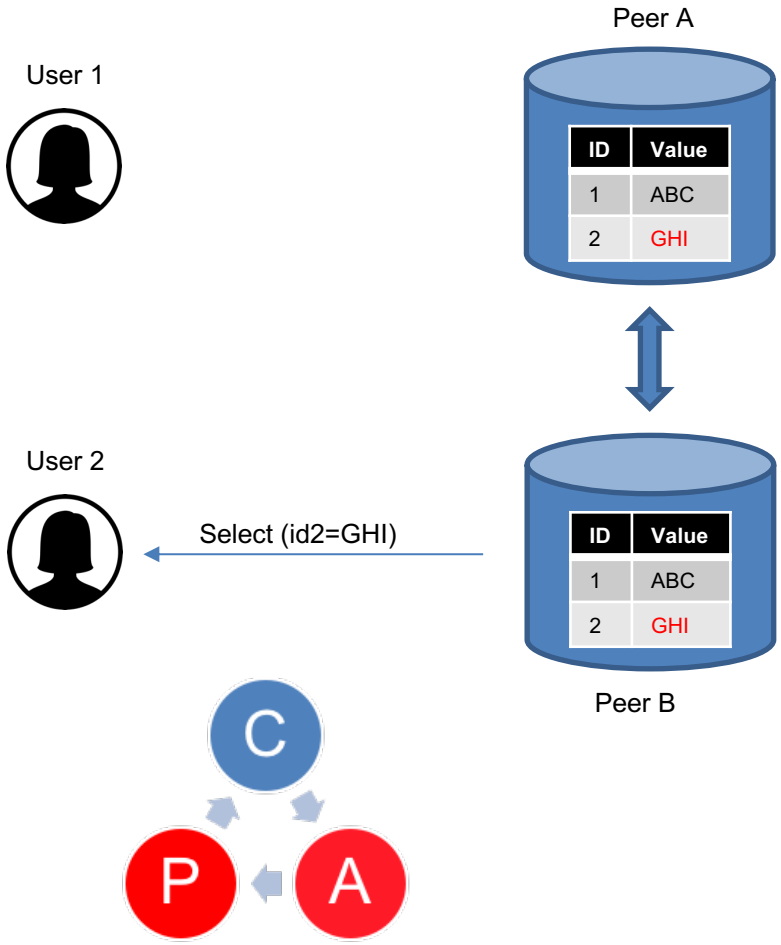


BASE

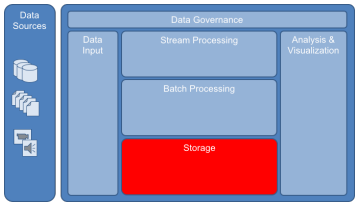


Mögliches Design-Prinzip für eine verteilte Datenbank

- Basically Available: Die Datenbank gibt immer Daten oder eine Fehlermeldung zurück
- Soft State: Es kann aber sein, dass die Daten, wenn Sie gelesen werden inkonsistent sind.
- Eventual Consistency: Bei einem 2. Lesen kann das Ergebnis anders sein, da die Daten inzwischen synchronisiert wurden

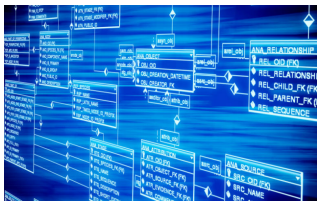


Speicheroptionen



On-Disk

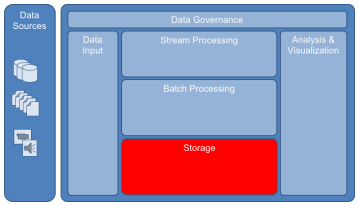
Verteiltes File System (DFS)



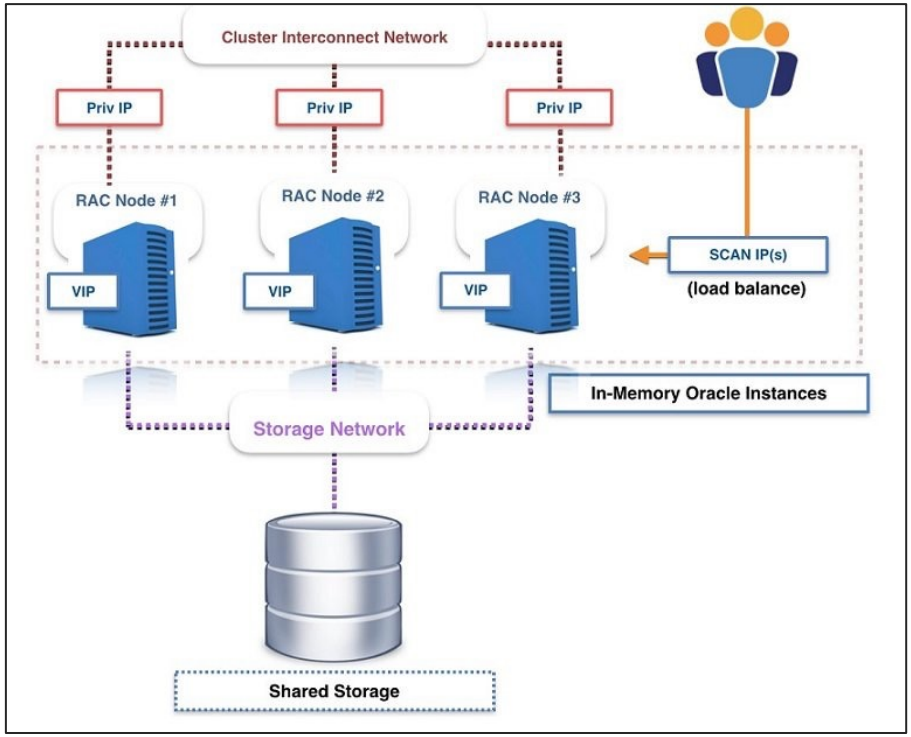
Relationale Datenbank (RDBMS)

NoSQL Datenbank

RDBMS für Big Data

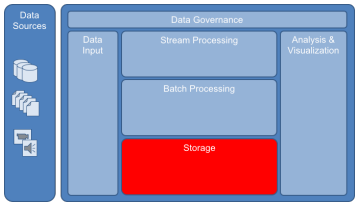


- Gut für Transaktionen
  - von relativ kleinen Datenmengen (strukturiert!),
  - die random read/write benötigen
- ACID-compliant
- Laufen oft auf einem Knoten, werden vertikal skaliert
- Manche können auf Clustern betrieben werden, benötigen dann Shared Storage
- Replikation ist gut möglich, Sharding muss auf Applikationslevel geschehen



Oft nicht ideal für Big Data!

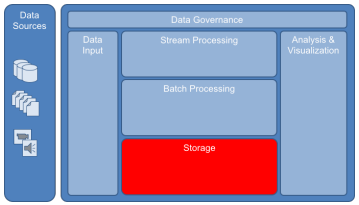
NoSQL für Big Data



- Daten können schemalos und nicht normalisiert sein
- Designed für
  - horizontale Skalierung
  - Sharding und Replikation
  - High Availability
- Oft OpenSource
- Flexible APIs/Query Languages
- BASE not ACID
- Unterschiedliche Typen
  - Key-Value
  - Document based
  - Column-family
  - Graph

mongoDB
APACHE HBASE
redis
cassandra
amazon DynamoDB
neo4j
CouchDB relax

Datenbanktypen – Key-Value



- Daten werden als Paare von einem Schlüssel (Key) und beliebigen Werten (Value) gespeichert. Der Wert kann unterschiedliche Formate haben (text, videos...)
- Sehr schnell und gut skalierbar
- Die Applikation muss den Key kennen, um auf die Daten zuzugreifen
- Es sind nur *insert*, *select*, *delete* Befehle möglich
- Beispiele: Amazon Dynamo, Redis

Key	Value
325	Max Mustermann, 12.3.1998, Weil am Rhein, Reiermann und Söhne
121	<CustomerID>121214</CustomerID><Name>...White</Name><Birthday>...986</Birthday><Address>Baslerstrasse_22,_Loerrach</Address>
531	1100010101000111000110010 1010101000100100100010010 0010010001111000101011010001000110001001...
351	0101011101001100111001110110011101000...01001 1100100111100...001...

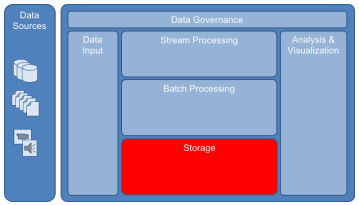
Text

XML

Video

Bild

Datenbanktypen – Document DB

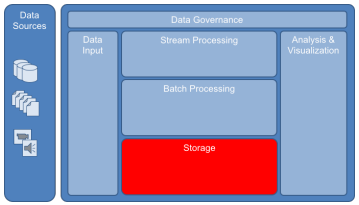


- Ähnlich wie Key-Value aber der Wert ist ein Dokument mit strukturierter Information (z.B. XML)
- Struktur der Dokumente kann unterschiedlich sein, Daten sind „Self-describing“
- Neben *insert*, *select*, *delete* sind *update* Befehle möglich, auch auf Teile des Dokuments
- Gut für semistrukturierte Daten mit wechselndem Schema
- Beispiele: MongoDB, CouchDB

Key	Value
325	{cu
121	{cu
531	{cu
351	{cu

```
{
   custID:1324234,
   custName:Mueller,
   invoiceID:2888389,
   Date:04092020,
   Items:[
      {itemID:23,quantity:2},
      {itemID:12,quantity:5},
      {itemID:43,quantity:1}
      ]
}
```

Datenbanktypen – Column Family DB



- Daten werden ähnlich wie bei RDBMS in Tabellen abgelegt
- Diese werden aber spaltenweise (in Column Families) gespeichert
- Schneller Zugriff auf Werte einer Spalte
- Gut wenn random read/write Zugang benötigt wird und die Daten eine gewisse Struktur haben
- Beispiele: Hbase, Amazon SimpleDB

Row Key	Column Family ID	Column Family Adresse
A10	ID:Nachname „Müller“ ID:Vorname „Hans“ ID:Email “hm134@gmx.de“	Adresse:Straße „Basler Straße“ Adresse:Nummer „4“ Adresse:Ort „Lörrach“
A11	ID:Email “zt111@gmx.de“	Adresse:Straße „Talweg“ Adresse:Nummer „6“ Adresse:Ort „Lörrach“
A12	ID:UserID “dhsten123“	Adresse:Straße „Lange Straße“ Adresse:Nummer „7“ Adresse:Ort „Weil“
A13	ID:Nachname „Schmidt“ ID:Vorname „Franz“	Adresse:Ort „Lörrach“

Column

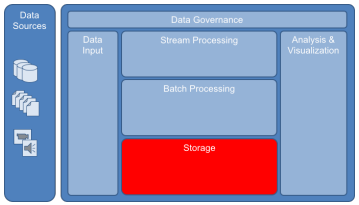
Column

Row Keys müssen zugewiesen werden

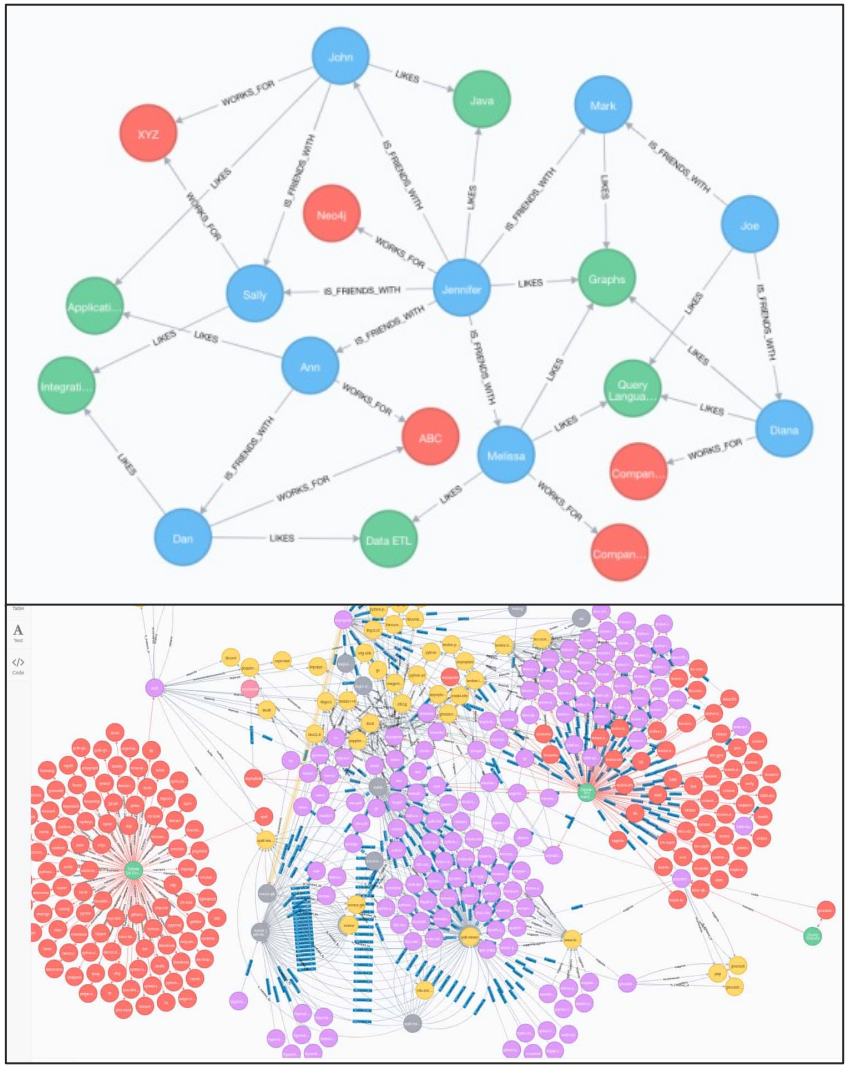
Column Family

Column Family

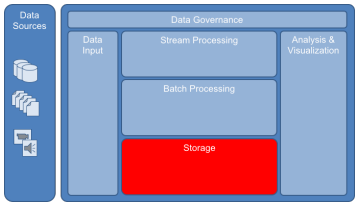
Datenbanktypen – Graph DB



- Schwerpunkt auf den Beziehungen der Entitäten
- Entitäten werden als Knoten (nodes) gespeichert und haben Attribute, die als Key-Value-Paare abgelegt werden
- Beziehungen werden als Kanten (Edges) gespeichert, die auch Attribute haben
- Gut zum Speichern von Beziehung und Analysieren von Mustern
- Beispiele: Neo4J, OrientDB, Infinite Graph



Speicheroptionen

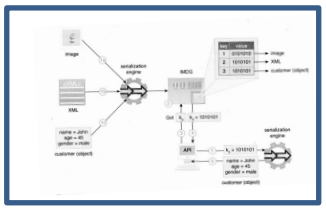


On-Disk

- Verteiltes File System (DFS)
- Relationale Datenbank (RDBMS)
- NoSQL Datenbank

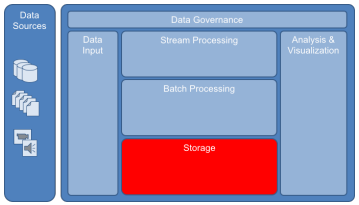
In-Memory

IM Data Bases



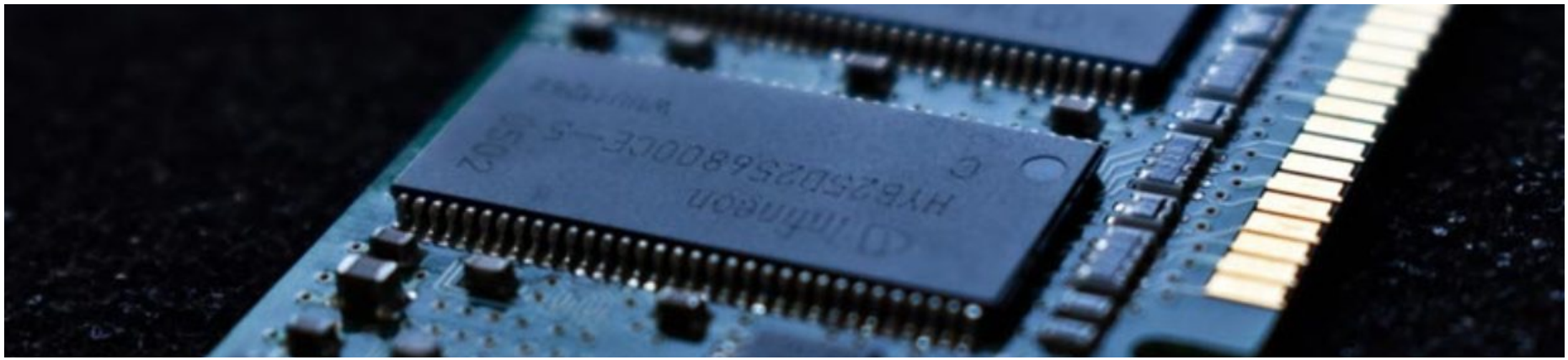
Grid Storage

In-Memory vs. On-Disk

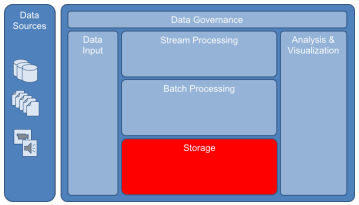


IM benutzt RAM als Speichermedium

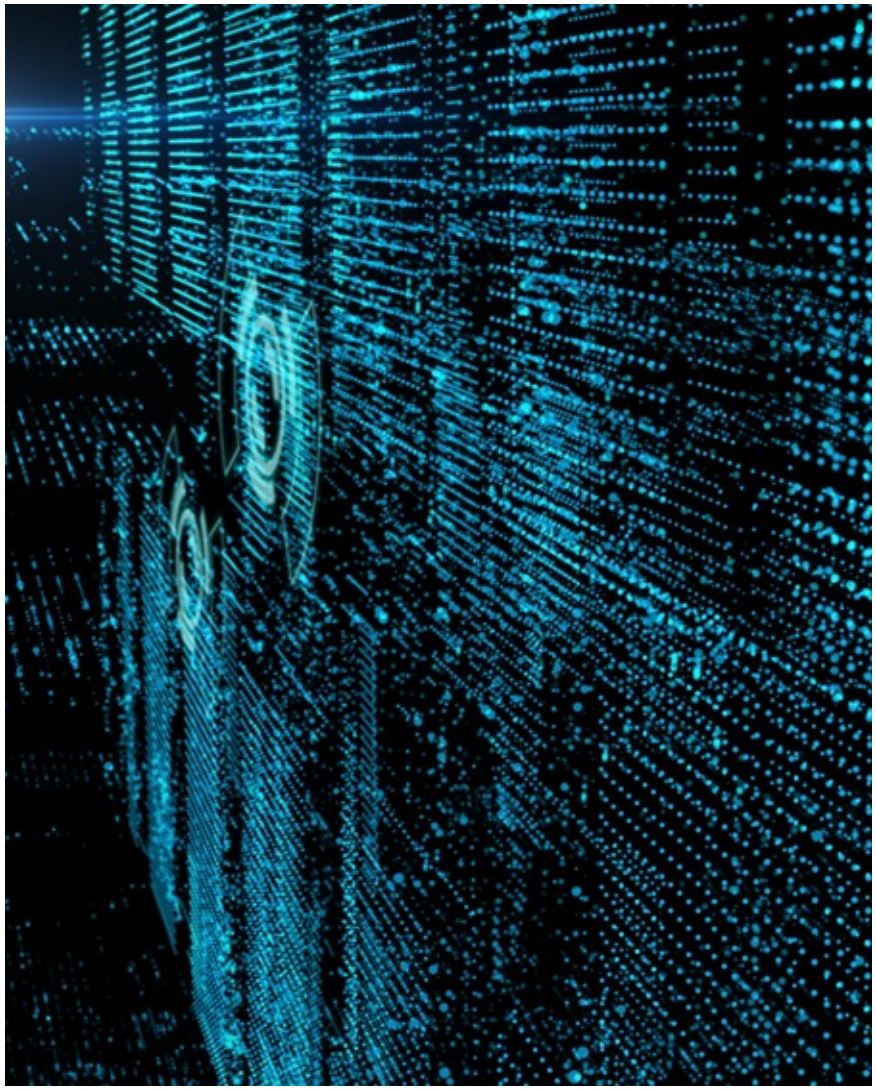
+ Geringere Latenz und schnellere Datentransferzeiten als Festplatten
+ Kann im Cluster gut horizontal skaliert werden
+ Gut für „Velocity“ Aspekt von Big Data und Real-time Analytics
– Hohe Kosten im Vergleich zu Festplatten
– „Durability“ (was passiert mit den Daten bei einem Systemunterbruch) muss gesondert adressiert werden



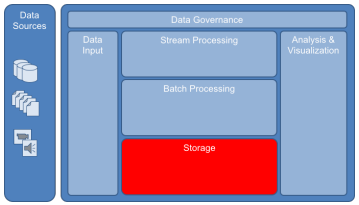
In-Memory Data Bases (IMDB)



- Bessere Performance
- Kann relationale oder NoSQL DB sein
- Je nach Technologie horizontal oder vertikal skalierbar
- Massnahmen für Durablity im Fall von Systemausfällen nötig
  - Transaction Logs und Snapshots werden auf HD gespeichert
  - Replikation
  - Kombination mit on-disk DB
- Event notification durch continuous querries möglich



In-Memory Data Grids



Lesen Sie das entsprechende Kapitel aus dem Lehrbuch und beantworten Sie die Fragen in Moodle