Motivation

Anwendungsszenarien

Lebenszyklus eines Informationssystems

Firma mit Daten über Produkte, Verläufe, Kunden, Angestellte, . . .

- Kleine Firma (Startup)
- Größere Firma (Wachstum, Erfolg, . . .)
- Noch größere Firma (mehr Niederlassungen, mehr Daten, verschiedene Orte,. . .)

Was nun?

- zentrale Datenbank (Data Warehouse)
- Verteilte Architektur
- Replikation
- Cloud computing

Universität Trier

Firmenzusammenschlüsse und -käufe

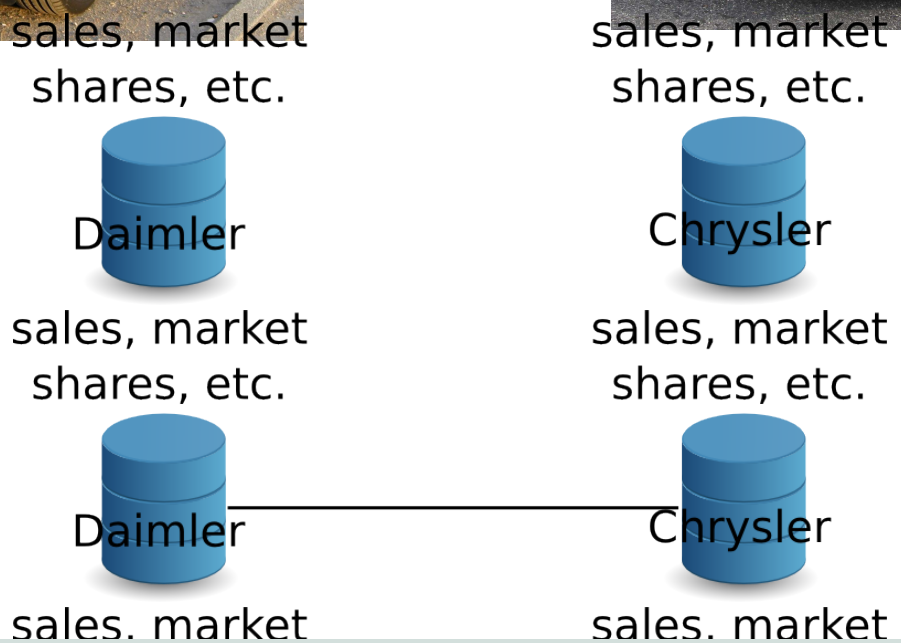
Daimler-Benz und Chrysler

- 1998: Daimler-Benz AG verschmilzt mit Chrysler Corporation (DaimlerChrysler AG)
- 2007: Chrysler wird verkauft (Daimler AG)

Daimler (Mercedes CL)

Chrysler (300C)

Quelle: Wikipedia



Relationale Datenbanken

- viele nützliche Eigenschaften: z.B. deklarative Anfragen, Konsistenzerhaltung, Datenunabhängigkeit, Normalisierung, ...
- aber auch Nachteile: skalieren nicht gut mit steigender Last, komplexe Eigenschaften oft nicht benötigt, Datenmodell passt nicht überall

Daher seit ca. 10-20 Jahren: **NoSQL-Datenbanken**

- kein relationales Datenmodell, sondern irgendein anderes: (JSON-, XML-)Dokumente, Key-Value-Paare, Graphen, Objekte, ...
- verteilte Architektur, sehr gut skalierbar
- oft keine deklarative Anfragesprache, sondern einfaches API
- schwache Konsistenzmodelle, verteilte Kopien nicht immer gleich
- aber: potentiell sehr hohe Performance

Heute typischerweise Hybridsysteme: "NoSQL" interpretiert als "Not-only-SQL"

Big Data Analytics

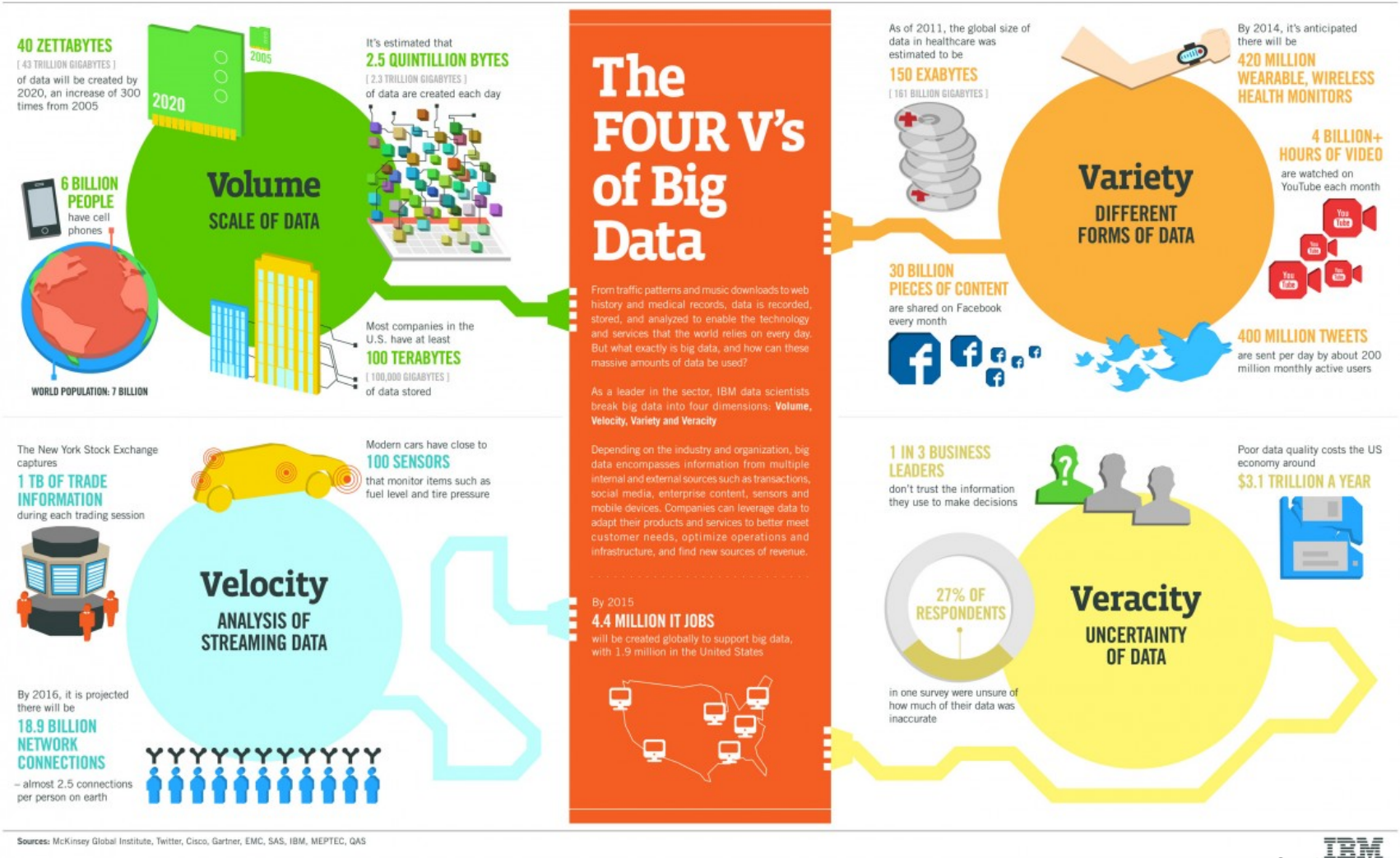
Große Datenmengen werden erzeugt und sollen zu neuem Wissen verarbeitet werden

- Zugriffslogs von Webservern
- Clicklogs von Suchmaschinen
- Verkaufshistorie (insgesamt oder individuell)
- Verbindungen und Interaktionen in sozialen Netzen
- Bewegungsdaten (insgesamt oder individuell)
- ...

Zentrale Systeme wären mit solchen Datenmengen überfordert

Beispiel: 10TB Daten von einer Platte mit 100MB/s lesen dauert *27 Stunden*

Big Data: 4 Vs



Quelle: IBM

Gründe für verteiltes Datenmanagement

Gründe für verteiltes Datenmanagement

Gründe für die Verwendung von verteilten Datensystemen und verteilter Anfrageverarbeitung [Kossmann, 2000]:

- Kosten und Skalierbarkeit
  Mainframes sind schwierig zu erweitern
- Replikation zur Verbesserung der Verfügbarkeit
  Durch mehrere Kopien der Daten kann die Verarbeitung (zu einem gewissen Grad) fortgesetzt werden, wenn ein Server ausfällt
- Integration verschiedener Softwaremodule
  Kein einzelnes Softwarepaket kann alle Bedürfnisse einer Firma erfüllen
- Integration von Legacy-Systemen
  Alte Legacy-Systeme müssen mit modernen Systemen koexistieren

Gründe für verteiltes Datenmanagement

Gründe für die Verwendung von verteilten Datensystemen und verteilter Anfrageverarbeitung:

- Neue Anwendungen
  Neue Anwendungen setzen verstärkt auf verteilte Technologien, z.B. elektronischer Handel, computer-supported collaborative work (CSCW)
- Zwänge des Marktes
  Firmen müssen ihr Geschäft reorganisieren, z.B. Pizza-Lieferdienste

Universität Trier

Gründe für verteiltes Datenmanagement

Warum Verteilung?

Verteilte Verarbeitung passt besser zur Organisationsstruktur von Firmen, die heute oft bereits verteilt ist.

Ziel von verteiltem Datenmanagement

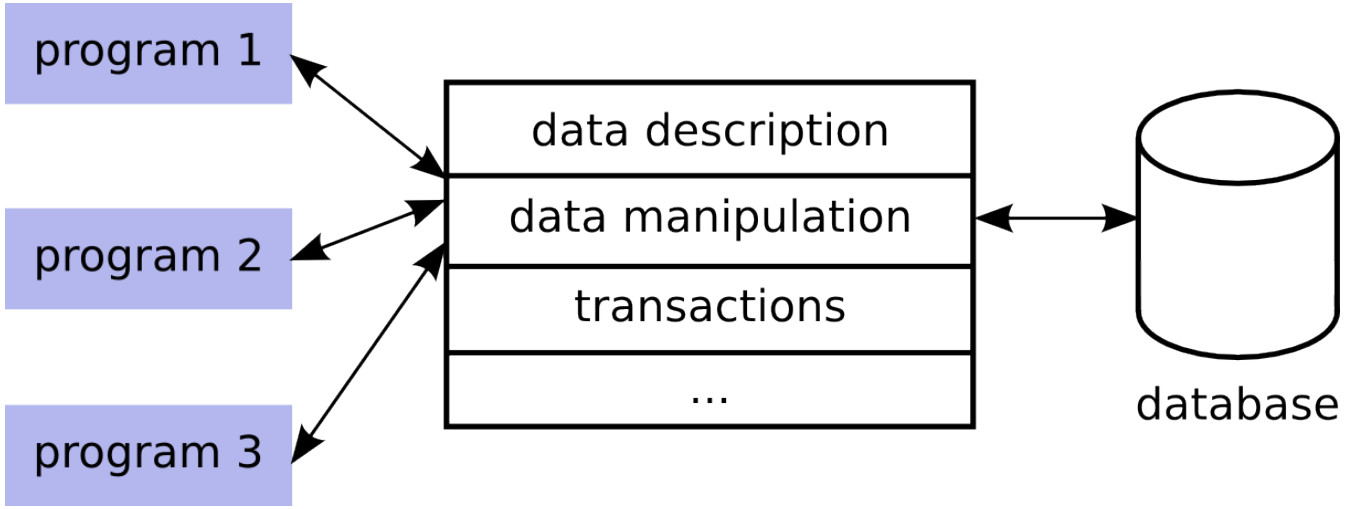
Ausführen von Anfragen so effizient wie möglich
Minimierung von Antwortzeit und Verzögerungen der Anwendung

Was ist verteiltes Datenmanagement?

Universität Trier

Zentrales vs. verteiltes Datenmanagement

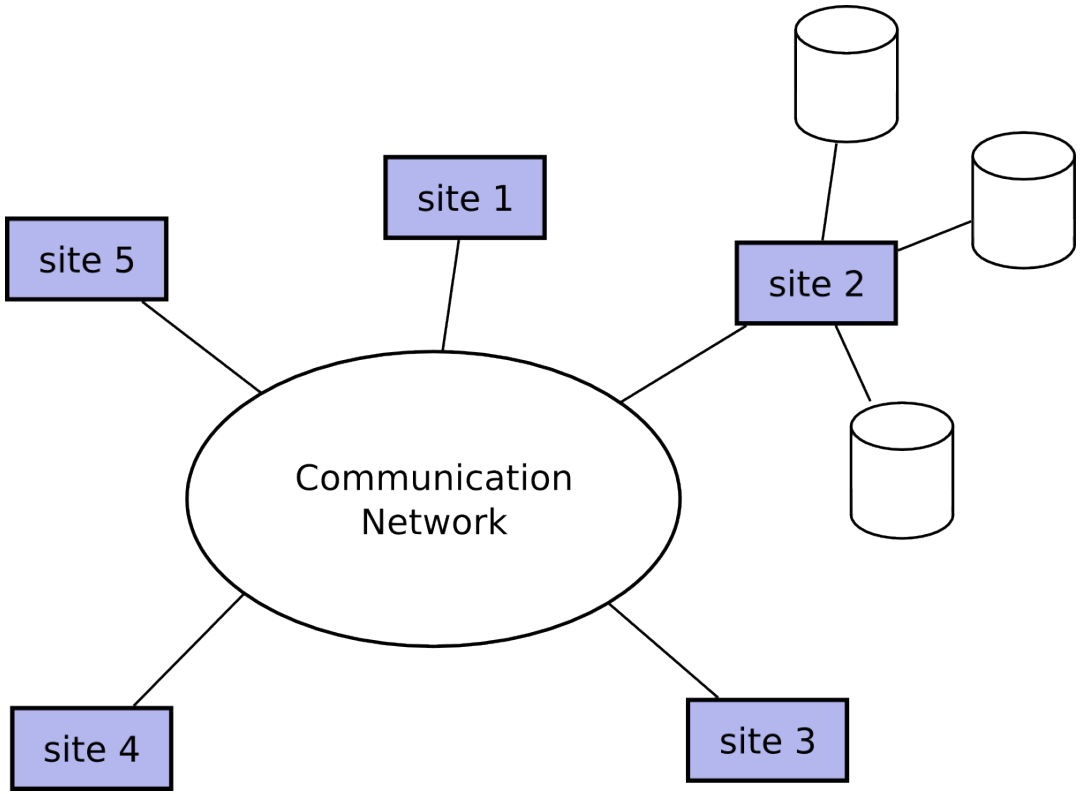
Zentrales Datenmanagement [Özsu Valduriez, 2011]:



Heutige Anforderungen:

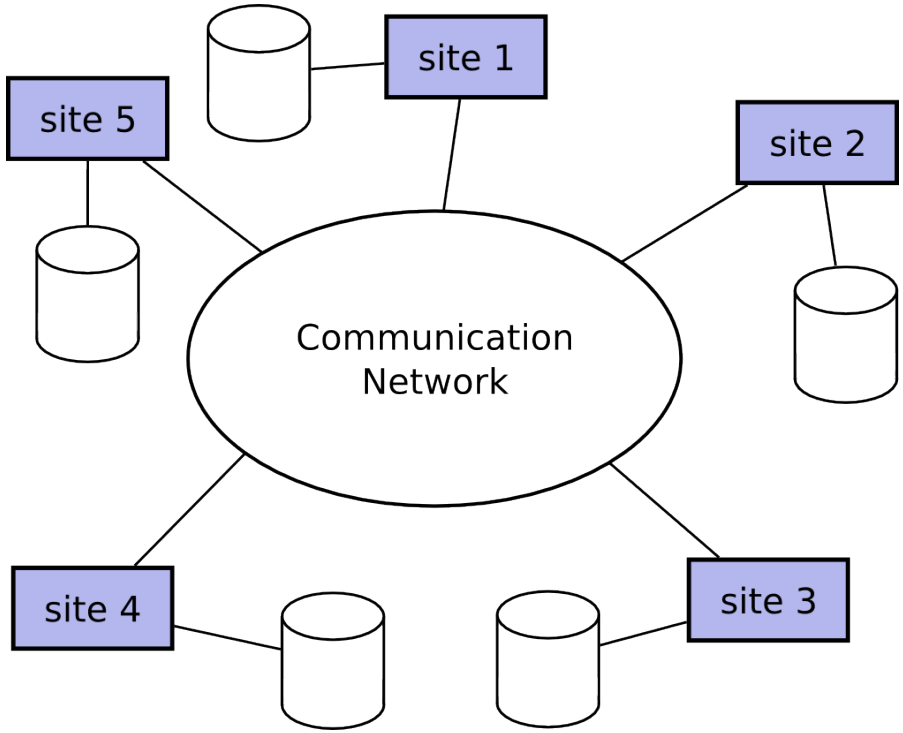
- Unterstützung dezentraler Datenorganisation
- Verfügbarkeit
- Performanz
- Skalierbarkeit unter wechselnder Last

Datenmanagement in einem Netz



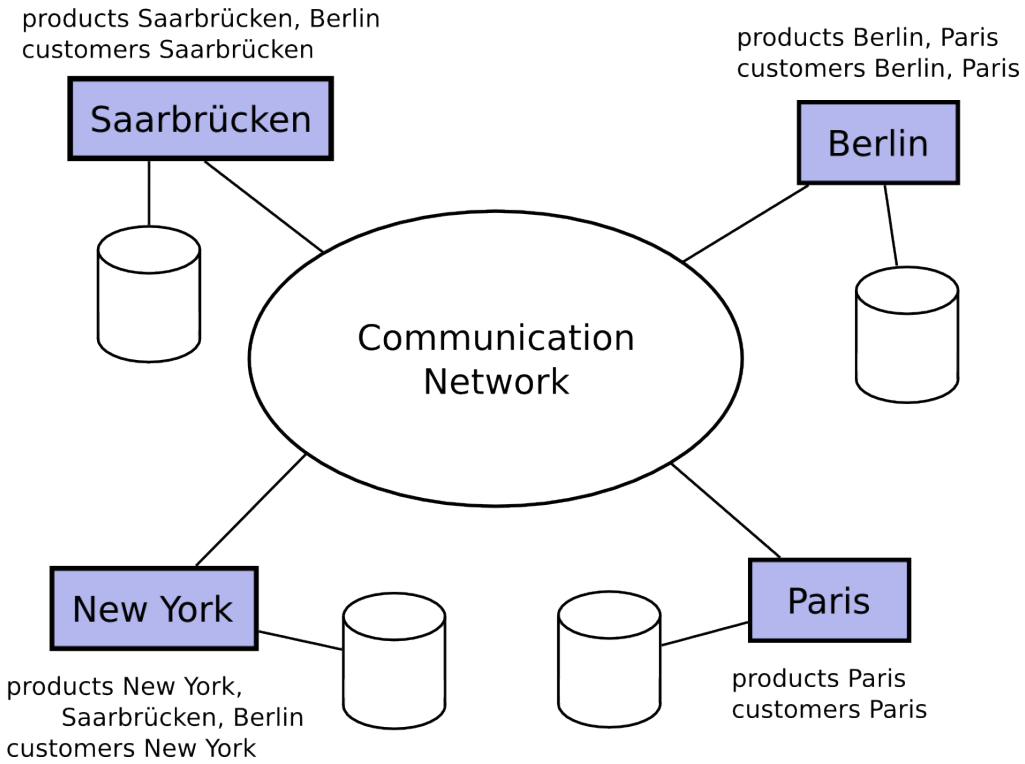
Hier handelt es sich nicht um ein *verteiltes Datenbanksystem*, sondern um eine *zentrale Datenbank in einem Netz*

Verteiltes Datenbanksystem



In einem *verteilten Datenbanksystem* sind die Daten über eine Reihe von Rechnern (Sites) verteilt.

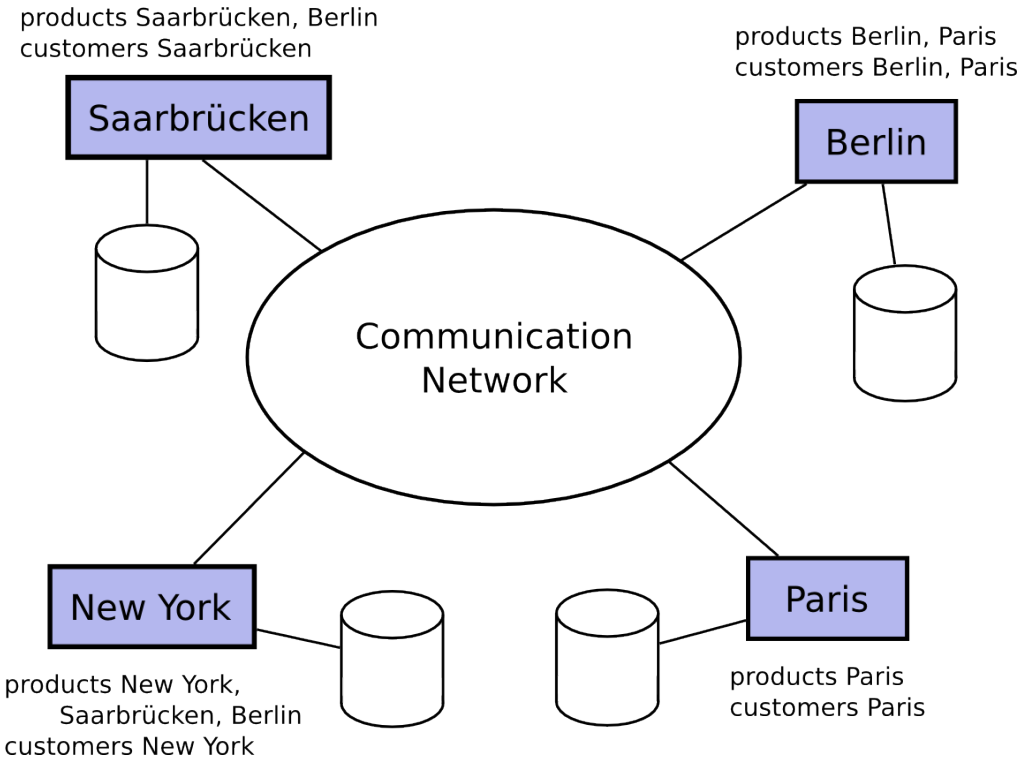
Beispiel: Verteiltes Datenbanksystem



Szenario:

- Firma mit Niederlassungen in Saarbrücken, Berlin, New York und Paris
- Datenbank mit Produkten und Kunden

Beispiel: Verteiltes Datenbanksystem



Beispielanfrage:

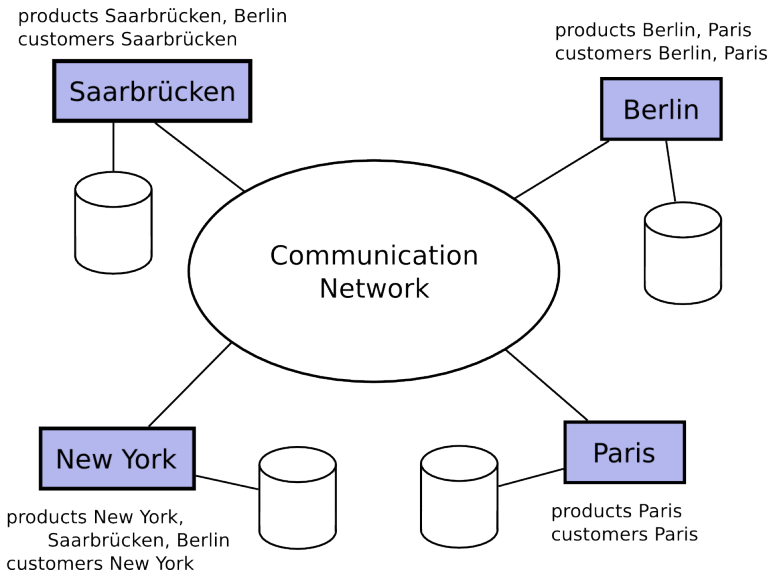
- Finde die Namen aller Kunden und der Produkte, die sie im letzten Monat gekauft haben

Anfrageverarbeitung:

- Join über die Relationen Produkte und Kunden

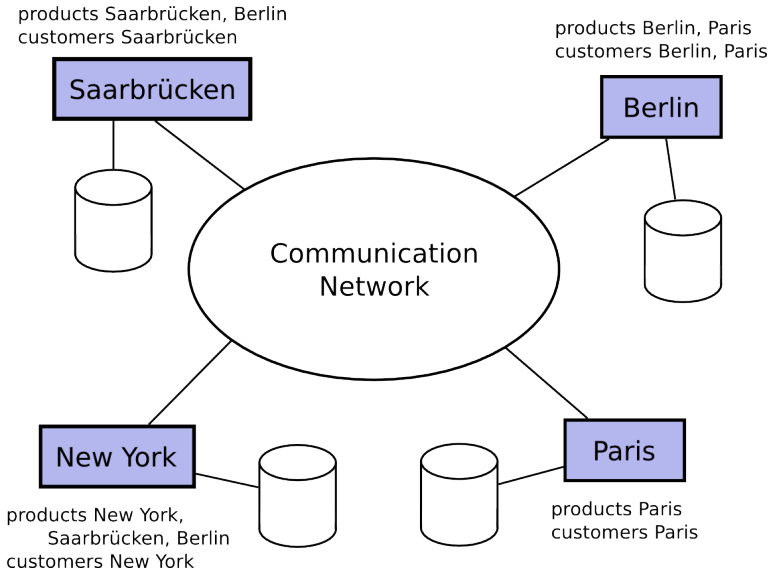
Wichtige Aspekte verteilter Anfrageverarbeitung

- *Fragmentierung*: Daten von Kunden und Produkten in Saarbrücken werden in der lokalen Datenbank gespeichert.
  - Daten werden in der Nähe des Ortes gespeichert, wo sie verwendet werden
  - Queries müssen aufgeteilt werden
- *Replikation*: Aus Performancegründen werden einige Daten repliziert



Wichtige Aspekte verteilter Anfrageverarbeitung . . .

- *Parallelität*: Mehrere Anfragen werden zur gleichen Zeit ausgeführt (Inter-Anfrage-Parallelität), Teile einer Anfrage werden parallel ausgeführt (Intra-Anfrage-Parallelität)
- *Transparenz*: Benutzer sollten die Fragmentierung, den Speicherort, oder die Replikation von Daten nicht beachten müssen



Definitionen und Charakteristika

Definition 1.1 (Verteilte Datenbank)

Eine *verteilte Datenbank* ist eine Sammlung von mehreren, logisch zusammenhängenden Datenbanken, die über ein Netz verbunden sind.

Definition 1.2 (Verteiltes Datenbankmanagementsystem)

Ein *verteiltes Datenbankmanagementsystem* (VDBMS oder DDBMS) ist eine Software, die die Daten in einer verteilten Datenbank verwaltet und die Verteiltung transparent für die Benutzer macht.

Bemerkung 1.3

Diese Definitionen sind unabhängig von Datenmodell und Anfragesprache. Sie gelten also für relationale Systeme genauso wie für NoSQL-Systeme, die oft ein nichtrelationales Datenmodell verwenden.

Verteilte Datenbanksysteme unterscheiden sich in . . .

- Grad der Kopplung
  lose Kopplung oder starke Kopplung
- Verbindungsstrukturen
  Punkt-zu-Punkt Kommunikation oder gemeinsamer Kommunikationskanal
- Abhängigkeit der Komponenten
  Abhängigkeit der Verarbeitungskomponenten
- Synchronisation zwischen den Komponenten
  Synchrone oder asynchrone Verarbeitung

Arten verteilten Datenmanagements

Andere Arten verteilter Daten (bzw. verteilten Datenmanagements):

- Peer-to-Peer und Filesharing
- Cloud Computing
- Web Services und das *Deep* Web
- Semantic Web
- Big Data Analytics

Herausforderungen verteilter Systeme

Irrtümer der verteilten Datenverarbeitung

Die folgenden Annahmen werden oft beim Entwurf und bei der Implementierung von verteilten Systemen gemacht, aber sie sind *falsch*.

1. Das Netz ist ausfallsicher.
2. Die Latenz ist gleich null.
3. Die Bandbreite ist unbegrenzt.
4. Das Netz ist sicher.
5. Die Netztopologie ändert sich nicht.
6. Es gibt nur einen Administrator.
7. Die Übertragungskosten sind null.
8. Das Netzwerk ist homogen (von James Gosling, Sun, 1996).
9. Der Ort (eines Datums) ist irrelevant (von Harry J. Foxwell, Oracle, 2009).

Die ersten sieben Einträge wurden 1994 von Peter Deutsch (Sun) formuliert.

Hardware-Ausfälle

Hardwareausfälle passieren ständig. Algorithmen und Infrastrukturen müssen also damit umgehen können.

Beispiel 1.4

Vereinfachende Annahme: Die Wahrscheinlichkeit, dass ein Rechner heute ausfällt, ist $P = \frac{1}{365}$.
Dann ist die Wahrscheinlichkeit, dass von n Rechnern heute mindestens einer ausfällt $1 - (1 - P)^n$

- $n = 1$: $0,0027$
- $n = 10$: 0.02706
- $n = 100$: 0.239
- $n = 1000$: 0.9356
- $n = 10000$: ≈ 1.0

Dramatisch unterschiedliche Zugriffszeiten

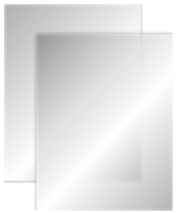
- L1 cache reference 0.5 ns
- L2 cache reference 7 ns
- Main memory reference 100 ns
- Compress 1K bytes with Zippy 10,000 ns
- Send 2K bytes over 1 Gbps network 20,000 ns
- Read 1 MB sequentially from memory 250,000 ns
- Round trip within same datacenter 500,000 ns
- Disk seek 10,000,000 ns
- Read 1 MB sequentially from network 10,000,000 ns
- Read 1 MB sequentially from disk 30,000,000 ns
- Send packet CA->Netherlands->CA 150,000,000 ns

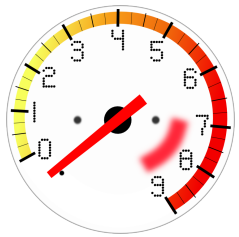
Jeff Dean (Google),
http://research.google.com/people/jeff/stanford-295-talk.pdf

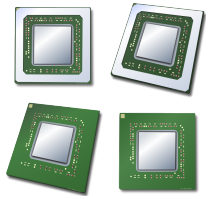
Versprechen verteilter Datenbanksysteme

Versprechen verteilter Datenbanksysteme

- Transparentes Datenmanagement
- Verlässlichkeit
- Verbesserte Performance
- Einfache Erweiterbarkeit







Transparentes Datenmanagement

Transparente Systeme verstecken Implementierungsdetails

- Datenunabhängigkeit (Datentransparenz – physisch, logisch)
  - Anwendungen müssen bei Änderungen der Definition und Organisation der Daten nicht geändert werden, und umgekehrt
  - Logische Datenunabhängigkeit (Änderungen der Schemadefinition)
    Anwendung läuft unverändert, wenn neue Attribute zu einer Relation hinzugefügt werden)
  - Physische Datenunabhängigkeit (Änderungen an der physischen Datenorganisation)
    Verstecken der physischen Datenorganisation (Relationen, Indexe)

Transparentes Management verteilter und replizierter Daten ...

- Netztransparenz (auch Verteilungstransparenz genannt)
  - Existenz des Netzes wird versteckt
  - Anwendungen sehen das verteilte DBS wie ein zentrales DBS
  - Ort der Daten wird versteckt
- Replikationstransparenz
  - Verwaltung von Kopien (Replikaten) von entfernt gelagerten Daten (Performance, Verlässlichkeit, Verfügbarkeit)
  - Existenz von Kopien wird versteckt

Transparentes Management verteilter und replizierter Daten ...

- Fragmentierungstransparenz
  - Aufteilen einer Relation in kleinere Fragmente (Performance, Verlässlichkeit, Verfügbarkeit)
  - Horizontale Fragmentierung (tupelweise Fragmentierung) und vertikale Fragmentierung (attributweise Fragmentierung)
  - Existenz von Fragmenten wird versteckt, Anfragen werden an die globale (unfragmentierte) Relation gestellt

Wer ist für Transparenz verantwortlich?

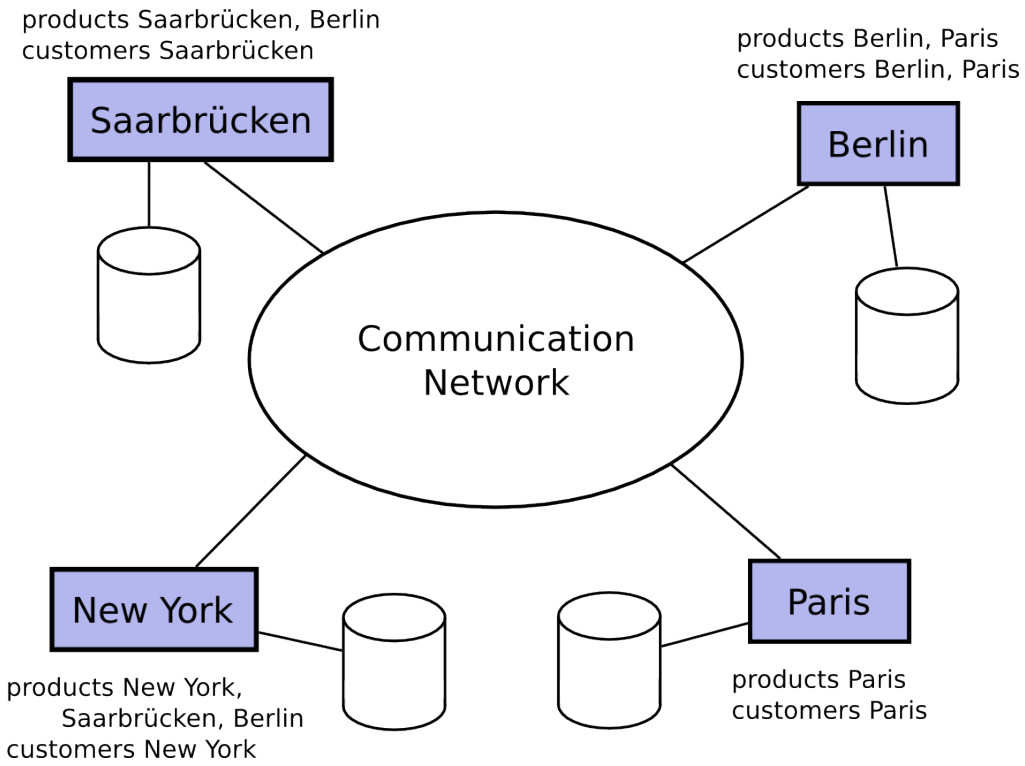
- Anwendung
  - Anwendungen oder Anwendungsmodule sind auf verteilte Art implementiert
  - Kommunikation und Datenaustausch durch standardisierte Protokolle (RPC, CORBA, HTTP, ...)
- Betriebssystem
  - Verantwortlich für Netztransparenz, z.B. auf dem Level des Filesystems (NFS) oder der Prokotolle
- Datenbanksystem
  - Transparenter Zugriff auf Daten auf nichtlokalen Datenbankinstanzen
  - Erfordert Aufteilung von Anfragen, Transaktionskontrolle, Replikation

Verlässlichkeit

Verlässlichkeit durch verteilte Transaktionen

- Kompensation des Ausfalls von Rechnern durch Kopien (Replikate) von Daten auf entfernten Rechnern
- Verteilte Transaktionen garantieren, dass
  - eine Sequenz von Datenbankoperationen als atomare Einheit durchgeführt wird, und
  - ein konsistenter Datenbankzustand in einen anderen konsistenten Datenbankzustand überführt wird, selbst wenn mehrere Transaktionen parallel ausgeführt werden.
  - Beispiel (Produkte/Kunden-Szenario):
    Erhöhe die Preise aller Produkte um 5%
- Höherer Aufwand für Änderungen bis hin zu Stillstand bei Rechnerausfall

Beispiel: Verteiltes Datenbanksystem



Szenario:

- Firma mit Niederlassungen in Saarbrücken, Berlin, New York und Paris
- Datenbank mit Produkten und Kunden

Performance

Verbesserte Performance

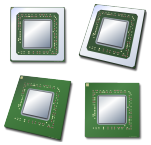
- Fragmentierung der konzeptuellen Datenbank, so dass Daten möglichst nahe an dem Ort ihrer Benutzung gespeichert werden
  $\rightarrow$ Reduktion von Transferkosten und Verzögerungen
- Inhärenter Parallelismus von verteilten Systemen
  - Inter-Anfrage-Parallelismus: Ausführung mehrerer Anfragen zur gleichen Zeit
  - Intra-Anfrage-Parallelismus: Parallele Ausführung von Teilanfragen, die auf verschiedene Teile der verteilten Datenbank zugreifen, auf verschiedenen Rechnern

Verbesserte Performance . . .

- Reine Lesezugriffe oder auch Änderungen
  - Anfragedatenbank (für Adhoc-Anfragen) und Produktionsdatenbank (für Änderungen)
    - Regelmäßiges Kopieren der Produktionsdatenbank in die Anfragedatenbank
  - Nur Lesezugriff während der regulären Arbeitszeiten, Updates werden zwischengespeichert und nachts als Batch eingespielt
  - Beispiel, wo das (fast) so ist: Porta vs. Stud.IP

Systemerweiterung

Einfachere Systemerweiterung



- Notwendigkeit, die Datenbank zu vergrößern und/oder die Bearbeitungszeit von Anfragen zu verringern
- Erweiterung des Rechnerverbundes durch zusätzlichen Speicher und/oder Verarbeitungskapazität
- System aus kleineren Computern ist oft billiger als einzelne große Maschine gleicher Kapazität

Herausforderungen

Herausforderungen

- Verteiltes Datenbankdesign
  Fragmentierung, Replikation und Verteilung
- Verteilte Anfrageausführung
  Möglichst kosteneffizientes Ausführen einer Anfrage über das Netz
- Verteilte Concurrency Control
  Synchronisation paralleler Zugriffe, so dass Integrität gewährleistet wird
- Zuverlässigkeit des verteilten DBMS
  Sicherstellen von Konsistenz, Erkennen und Beheben von Ausfällen
- Heterogene Datenbanken
  Übersetzung zwischen Datenbanksystemen - Datenmodelle und Sprachen

Universität Trier

Systemarchitektur

Klassische Standardarchitekturen

Verteilte Datenbanksysteme

- Verteiltes Informationssystem
  Anwendungen kommunizieren zum Datenaustausch
- DBMS in einem verteilten Filesystem
  Betriebssystem versteckt die Verteilung
- Verteiltes Datenbanksystem
  Verteilung wird vollständig vom verteilten Datenbanksystem verwaltet

Parallele Datenbanksysteme

- Datenverarbeitung durch parallele Computer (mehrere Prozessoren/Kerne, Spezialhardware)
- Bessere Performance durch mehrere Verarbeitungseinheiten
- Beispiel:
  - Sequentielles Lesen einer 100GB Relation benötigt ca. 17 Minuten
  - Paralleles Lesen von 10 Platten in 10 Threads dauert nur 1:40 Minuten

Heterogene Datenbanksysteme

- Ziel:
  - Integration existierender unabhängiger Datenbanken (Legacy-Systeme)
- Integrierter Zugriff:
  - Globale Anfragen, Beziehungen zwischen Objekten verschiedener Datenbanken, globale Integrität
- Probleme:
  - Heterogenität auf verschiedenen Ebenen: System, Datenmodell, Schema
- Anwendungsbeispiel:
  - Integration von Web-Quellen (z.B. Mediator-Systeme)

Moderne Standardarchitekturen

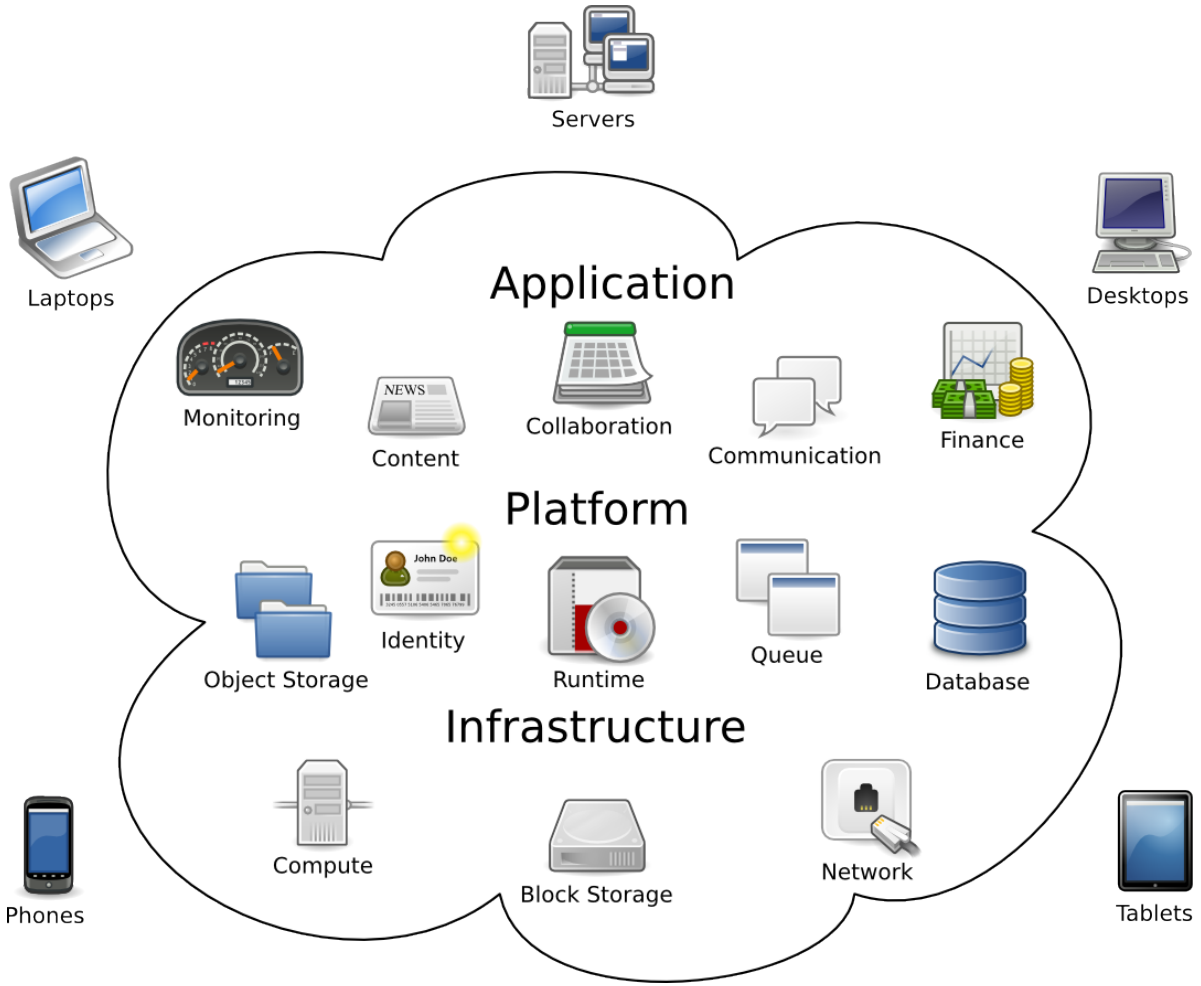
Peer-to-Peer-Systeme

- Definition:
  - Peer-to-Peer (P2P) Netze haben keine spezialisierten Server
  - Alle (oder mindestens die meisten) Peers (Rechner) speichern Daten und erlauben den Zugriff darauf
  - Begrenzte Information über das Netz
    - Peers kennen nur ihre direkten Nachbarn
    - kein globales Wissen
    - keine zentrale Koordination
- Beispiele:
  - Napster, Gnutella, Freenet, BitTorrent
  - Verteilte Speicherung und Verwaltung von Daten (z.B. MP3-Dateien)
  - Suche über zentralen Server (Napster) oder verteilt (Gnutella)
  - Verteilung von Dateien und Downloads

Grid-Systeme

- Ziel:
  - Kombination der Resourcen vieler Computer in einem Netz
  - Große Datenmengen
  - Probleme, deren Lösung langwierige Berechnungen erfordert, z.B. Erdbebenvorhersage, Wettervorhersage, Klimamodelle
- Architektur
  - Cluster von (oft tausenden) lose gekoppelten Computern
  - Grid Middleware (z.B. Globus Toolkit, gLite, UNICORE) bietet einfachen Zugriff auf Grid-Resourcen, teilt das Problem in kleine Stücke auf, und weist sie Rechnern zu

Cloud computing



http://en.wikipedia.org/wiki/Cloud_computing

Charakteristiken

- Kunden gehört die physikalische Infrastruktur nicht
- Kunden benutzen Resourcen und zahlen dafür
- Einzelner Zugriffspunkt für alle Rechenbedürfnisse von Kunden
- Kunden müssen sich nicht um Lastspitzen kümmern
- Kommerzielle Angebote müssen Qualitätsvereinbarungen (Quality of Service, QoS) mit Kunden erfüllen und schließen in der Regel Servicevereinbarungen (Service Level Agreements, SLA) mit ein.

Klassifikation klassischer verteilter Datenbanksysteme

Definition 1.5 (Mehrrechner-Datenbanksystem [Rahm, 1994])

Ein *Mehrrechner-Datenbanksystem* ist ein Datenbanksystem, das mehrere Rechner oder Datenbankinstanzen zur Verarbeitung von Datenbankoperationen verwendet.

Bemerkung 1.6

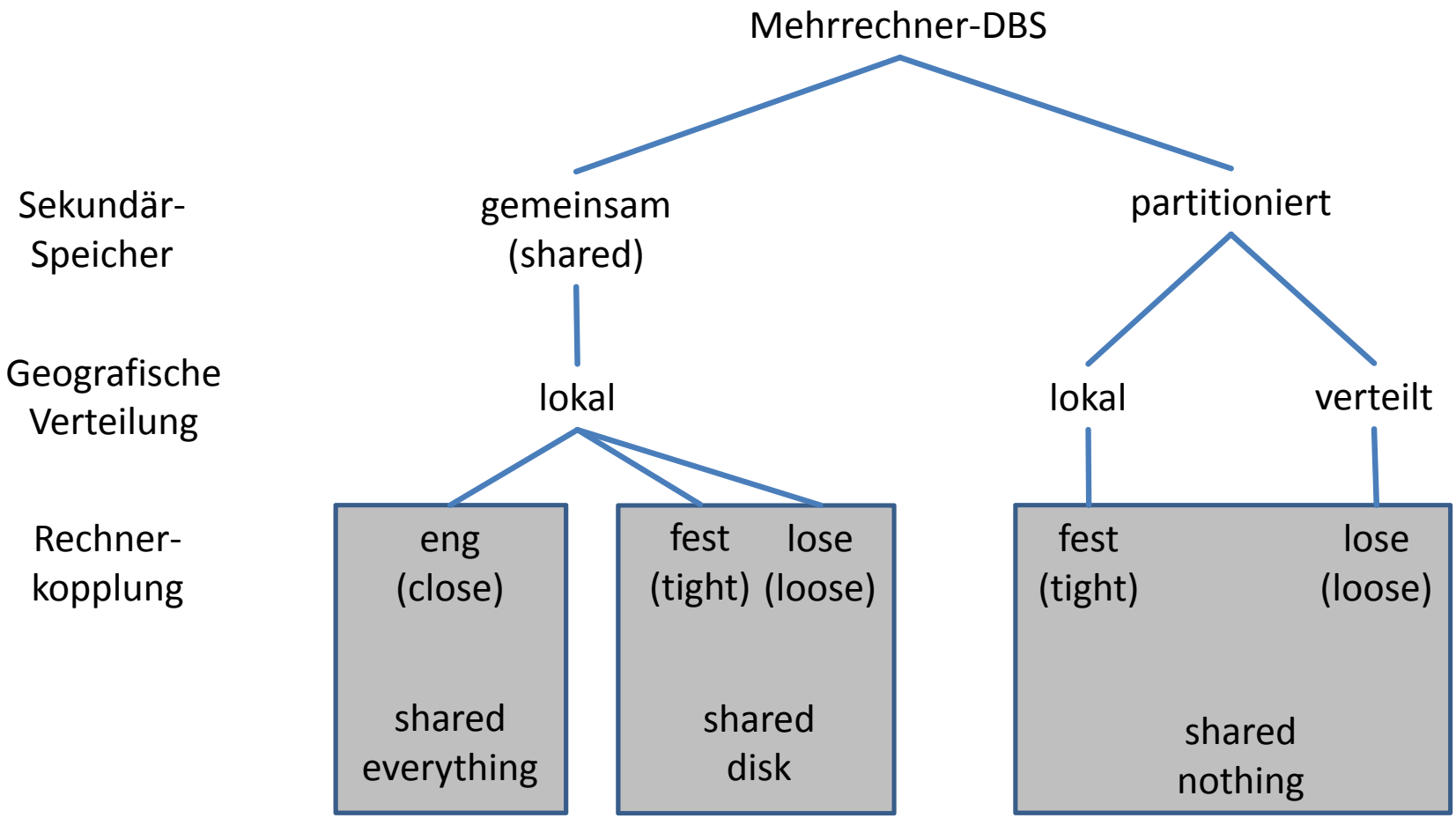
Gelegentlich werden solche System auch als "Mehrprozessor-Systeme" bezeichnet.

Klassifikationskriterien [Rahm, 1994]

- Funktionale Äquivalenz und Spezialisierung
- Zugriff auf Sekundärspeicher
- Geografische (örtliche) Verteilung
- Kopplung der Rechner (eng, fest, lose)
- Integrierte oder föderierte Architektur
- Zentrale oder dezentrale Koordinierung

Bemerkung 1.7

Wir nehmen an, dass jeder Rechner die gleiche Funktionalität bereitstellt.



Zugriff auf Sekundärspeicher

Partitionierter Zugriff

- Voller Zugriff auf lokale Partition
- Zugriff auf nicht-lokale Partitionen erfordert Kommunikation mit dem Partitionseigentümer ("owner site")

gemeinsamer (shared) Zugriff

- Voller Zugriff auf alle Partitionen (gesamte Datenbank)
- Erfordert aufwendige Synchronisation

Geographische Verteilung

Lokal (Datenbank-Cluster)

- sehr effiziente Inter-Rechner-Kommunikation
- Fehlertoleranz
- erlaubt dynamische Verteilung und Lastbalancierung
- geringer administrativer Aufwand

Verteilt (verteiltes DBS in Wide-Area-Netz)

- Unterstützung von verteilten Organisationsstrukturen
- Fehlertoleranz im Fall von katastrophalen Fehlern

Kopplung der Rechner

Enge Kopplung (close coupling)

- CPUs eines Rechners teilen sich den Hauptspeicher
- Effiziente Kooperation
- Lastbalancierung oft bereits durch das Betriebssystem
- Probleme: Zuverlässigkeit, Kohärenzkontrolle (CPU-Cache), nur für relativ kleine Zahl von Prozessoren/Kernen möglich

Kopplung der Rechner

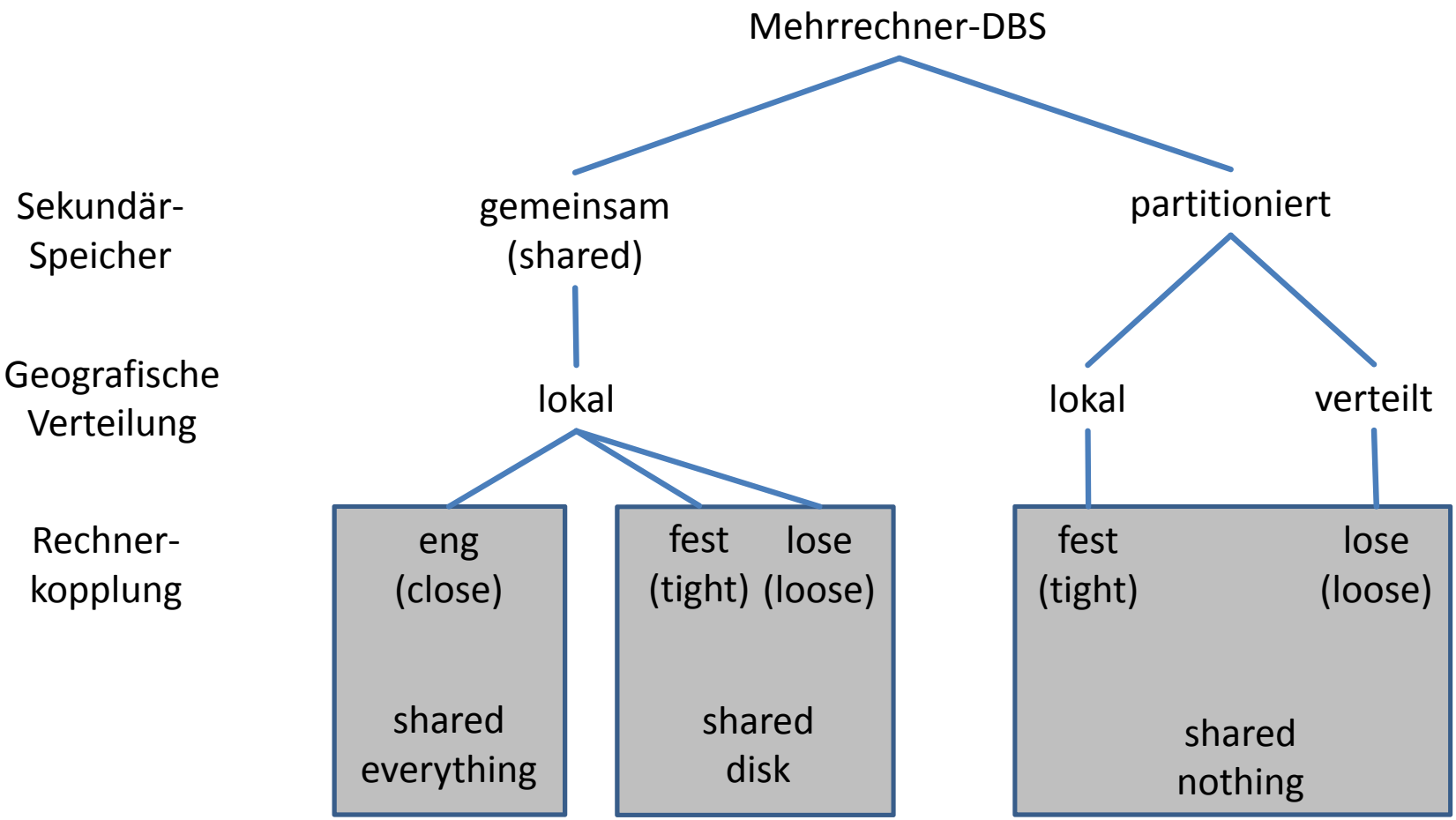
Lose Kopplung (loose coupling)

- Unabhängige Standorte ("sites") mit eigenem Hauptspeicher und eigenen Prozessoren
- Vorteile: Isolation von Fehlern, Skalierbarkeit
- Nachteile: teure kommunikation über das Netz, teure Operationen, schwierige Lastbalancierung

Feste Kopplung (tight coupling)

- Kombination von enger und loser Kopplung
- Neben eigenem Hauptspeicher haben Sites auch gemeinsamen schnellen (Sekundär-)Speicher
- Verwaltet vom Betriebssystem

Klassen


Mehrrechner-DBS
Sekundär-Speicher
gemeinsam (shared)
partitioniert
Geografische Verteilung
lokal
lokal
verteilt
Rechner-kopplung
eng (close)
shared everything
fest (tight)
lose (loose)
shared disk
fest (tight)
lose (loose)
shared nothing

"Shared Everything"-Systeme

Mehrere Rechner teilen sich Hauptspeicher und Sekundärspeicher

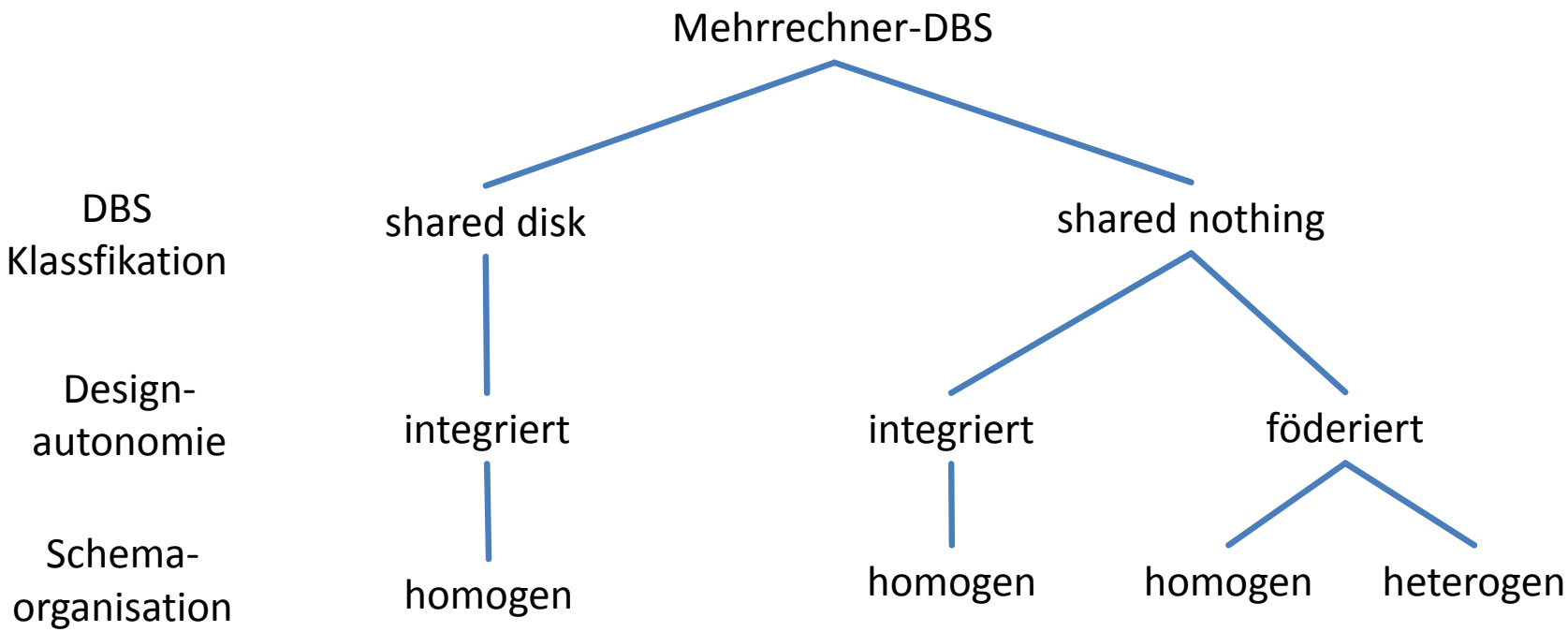
"Shared Disk"-Systeme

Mehrere Rechner - jeder mit eigenem Speicher - teilen sich gemeinsamen Sekundärspeicher

"Shared Nothing"-Systeme

Weder Hauptspeicher noch Sekundärspeicher werden zwischen den Rechnern geteilt

Heterogenität und klassische verteilte Datenbanksysteme



Integrierte vs. föderierte Architektur

Integrierte Architektur

- Gemeinsame Datenbank für alle Sites (ein konzeptuelles Schema)
- hohe Verteilungstransparenz (Zugriff auf globale DB über das lokale DBMS)
- erfordert Kooperation der DBMS - schränkt Autonomie der Sites ein
- Top-down Ansatz

Föderierte Architektur

- Sites mit eigener lokaler DB und eigenem lokalem Schema
- Globales integriertes Schema (Schemaintegration)
- Höherer Autonomiegrad der Sites
- Bottom-up Ansatz

Näheres in der Vorlesung über Datenintegration

Klassifikation von P2P-Systemen

Universität Trier

Zentrale vs. dezentrale Koordination

Zentrale Koordination

- Jeder Knoten hat eine globale Sicht (entweder direkt oder über einen Masterknoten)
- Zentraler Koordinator:
  - Dient als Initiator von Anfragen und Transaktionen
  - Kennt alle betroffenen Sites
- Gewährleistung von DB-Eigenschaften wie ACID, Vollständigkeit von Ergebnissen, . . .
- Typische Anwendung: Verteilte und parallele Datenbanken
  - Hohe Verfügbarkeit, beschränkte Skalierbarkeit

Zentrale vs. dezentrale Koordination ...

Dezentrale Koordination

- Kein globales Wissen (Schema, Verteilung der Daten, etc.)
- Keine globale Sicht auf das Gesamtsystem
- Sites sind relativ autonom: Globales Verhalten ist das Ergebnis lokaler Interaktionen
- Typische Anwendung: P2P

Zentrale vs. dezentrale Koordinierung

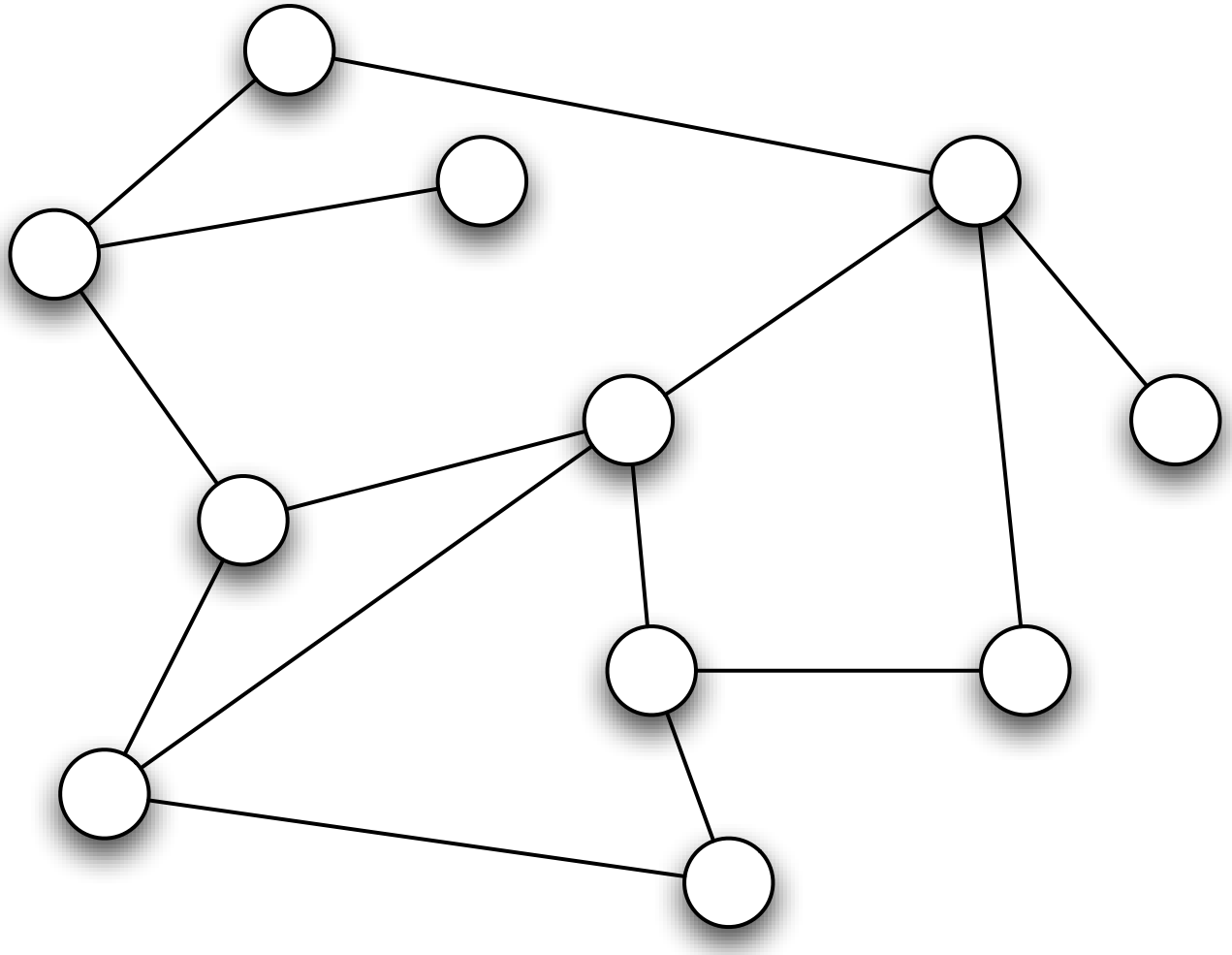
Grenzen von klassischen verteilten Datenbanksystemen

- Globales Schema muss bekannt sein
- Teilnehmer müssen bekannt sein
- Teilnehmer müssen antworten (also verfügbar sein)
- Hinzukommende bzw. wegfallende Teilnehmer erfordern Änderungen am globalen Schema

Offene Fragen

- Skalierbarkeit hin zu Internet-Scale ($>$200 Millionen Sites)?
- Robustheit gegen Ausfälle, Attacken, Zensur, etc.

Peer-to-Peer-Systeme



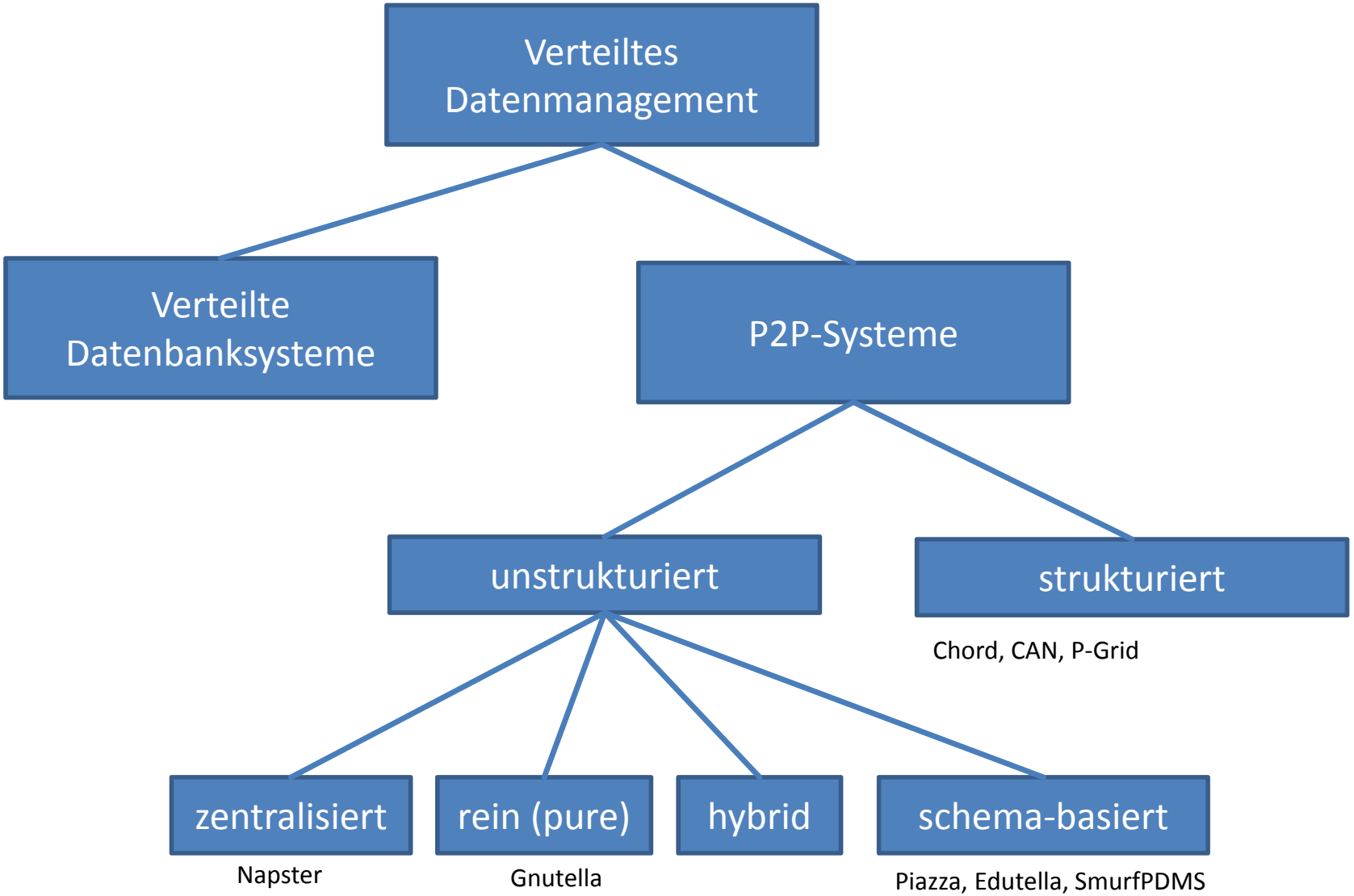
Peer-to-Peer-Systeme . . .

- Mehrere P2P Projekte, Dienste, Systeme
- Durchbruch wegen Filesharing
- Heute
  - Neue Anwendungen: Instant Messaging, Rechen-/Datengrids, Kollaboration, Web Caching, Metadatenmanagement
  - Neue Konzepte: Verteilte Datenstrukturen, Routing, . . .
- Trends
  - Selbstorganisierende Systeme
  - Metadatenmanagement
  - Sicherheit, Trust

Charakteristika

- Selbstorganisation
  globales Verhalten ist das Ergebnis lokaler Interaktion
- Dezentrale Kontrolle
  keine zentrale Koordination, keine zentrale Datenbank, kein Peer kennt das gesamte System
- Symmetrische Kommunikation
  Im wesentlichen gleiche und autonome Peers, verfügbare Daten und Dienste können von allen Peers genutzt werden

Peer-to-Peer-Systeme . . .



Anwendungen

- Faire Verwaltung von großen öffentlichen Datenmengen (Metadaten für das Semantic Web, Web Service Verzeichnisse, Gendaten,. . .)
- verteilte Indexstrukturen
- Kooperatives Speichern/Spiegeln von Daten
  - Speichern von Daten für Sites, die nur zeitweise online sind
  - Dezentralisiertes Caching oder Replikation von Datensammlungen, Software, etc.

Zusammenfassung

Zusammenfassung

- Motivation für verteiltes Datenmanagement
- Transparenz und andere Versprechen verteilter Datenbanksysteme
- Systemarchitekturen

Universität Trier

Literatur

[Özsu Valduriez, 2011] M. Tamer Özsu, P. Valduriez.
Principles of Distributed Database Systems.
Third Edition, Springer, 2011.

[Rahm, 1994] E. Rahm.
Mehrrechner-Datenbanksysteme.
Addison-Wesley, Bonn, 1994.

[Rahm Saake Sattler, 2015] . Rahm, G. Saake, K.-U. Sattler.
Verteiltes und Paralleles Datenmanagement. Von verteilten Datenbanken zu Big Data und Cloud
Springer Vieweg, 2015.

[Kossmann, 2000] D. Kossmann.
The State of the Art in Distributed Query Processing,
ACM Computing Surveys,
Vol. 32, No. 4, 2000, S. 422-469.