



## Inhalt:

- *Hierarchische Datenbanken*
- *Netzartige Datenbanken*
- *Verteilte Datenbanken*
- *Objektorientierte Datenbanken*
- *No-SQL-Datenbanken*



*Hierarchische Datenbanken sind vermutlich die ältesten Datenbanken. Heute haben sie nur historische Bedeutung. Bei Bedarf kann man sie durch relationalen Datenbanken relativ einfach simulieren.*

*Datenbestand beinhaltet die Datensätze, die feste oder variable Struktur haben können. Die semantischen Beziehungen zwischen den Datensätzen sind fest programmäßig durch Verweise implementiert (nach der Art "Vorgänger-Nachfolger").*

*Vorteile:*

- *Theoretisch sehr schnelle Such-, Einfüge- und Änderungsvorgänge.*

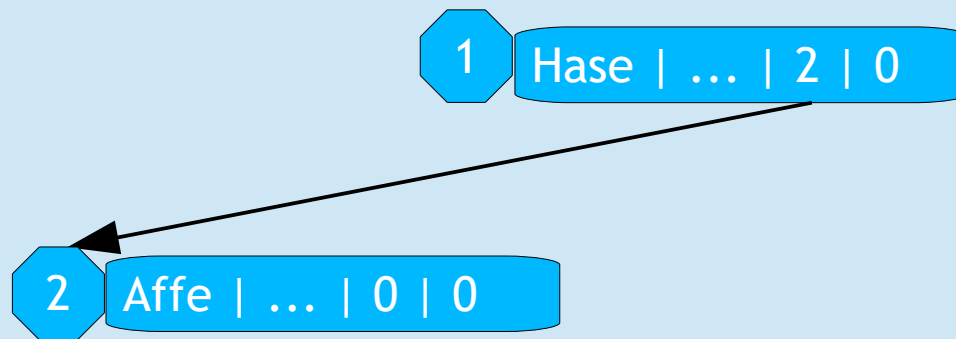
*Nachteile:*

- *Unflexible baumartige Struktur.*
- *Mit der Zeit werden die Zugriffe (wegen Einfüge- und Löschoperationen) langsamer, man muss den Baum ab und zu in Gleichgewicht bringen, was zeit- und speicheraufwendig ist.*



Hase

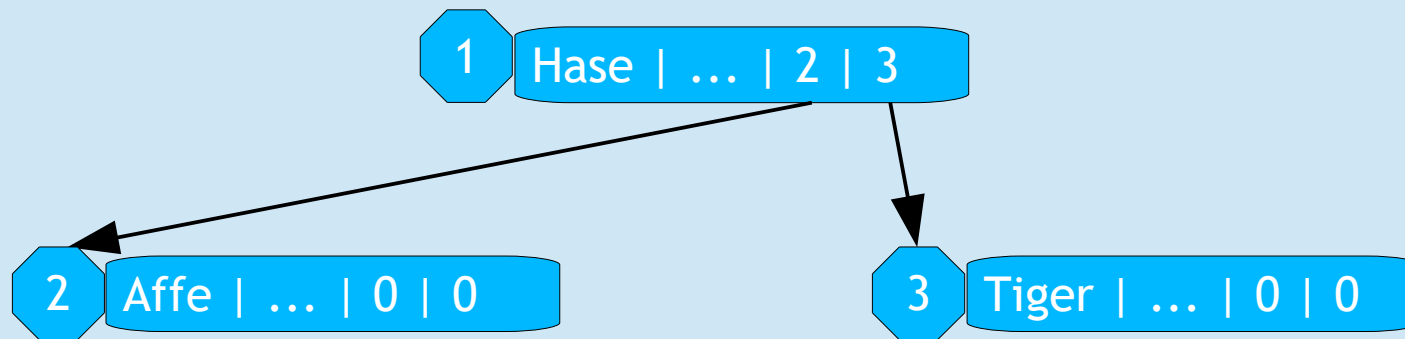
Alphabetische Ordnung



Hase

Affe

Alphabetische Ordnung

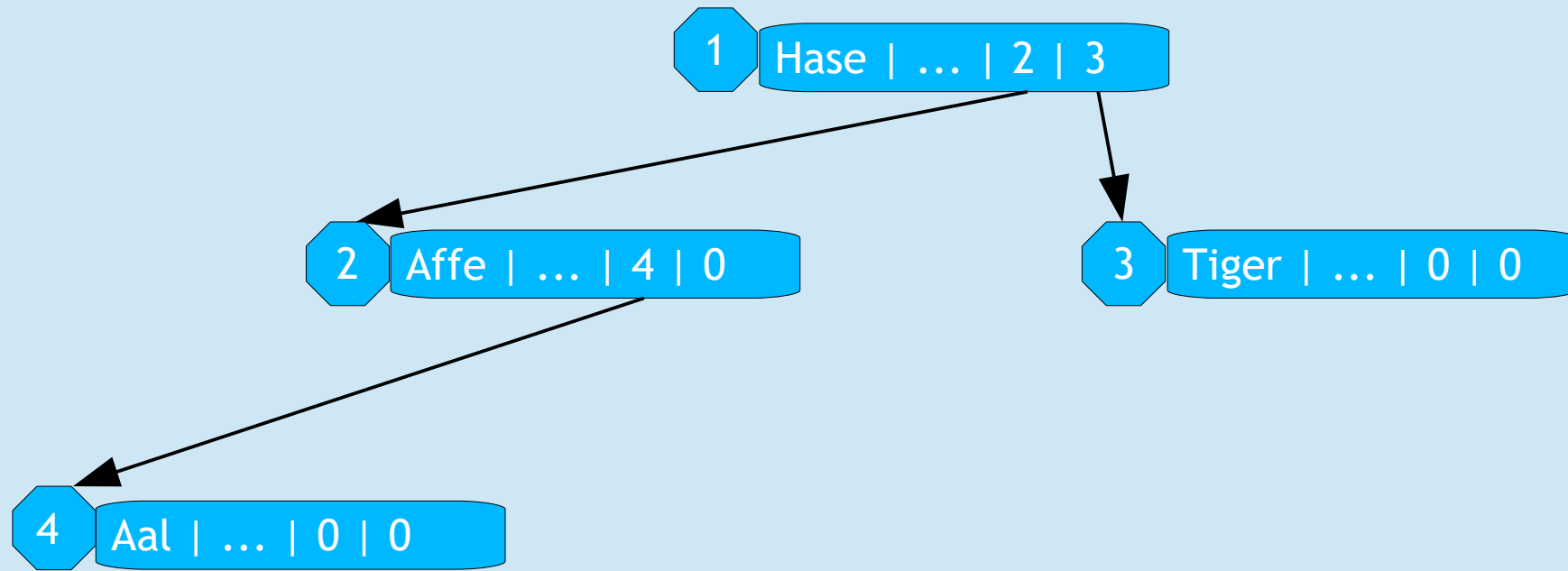


Alphabetische Ordnung

Hase

Affe

Tiger



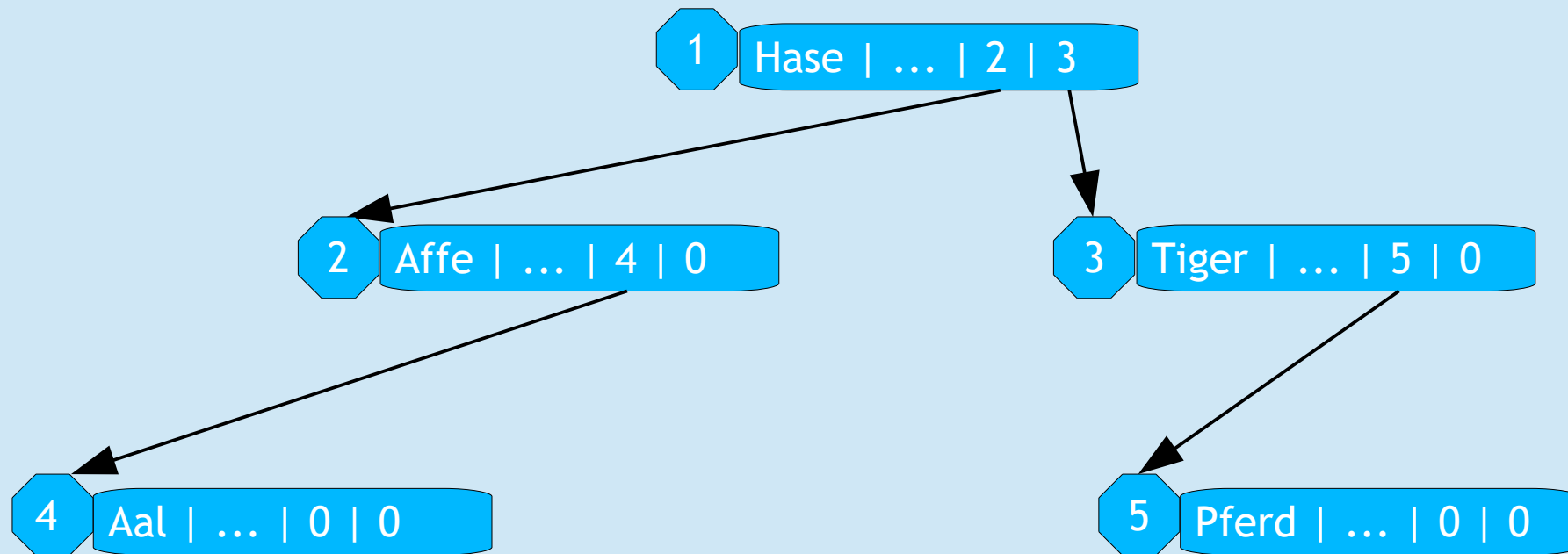
Alphabetische Ordnung

Hase

Affe

Tiger

Aal



Alphabetische Ordnung

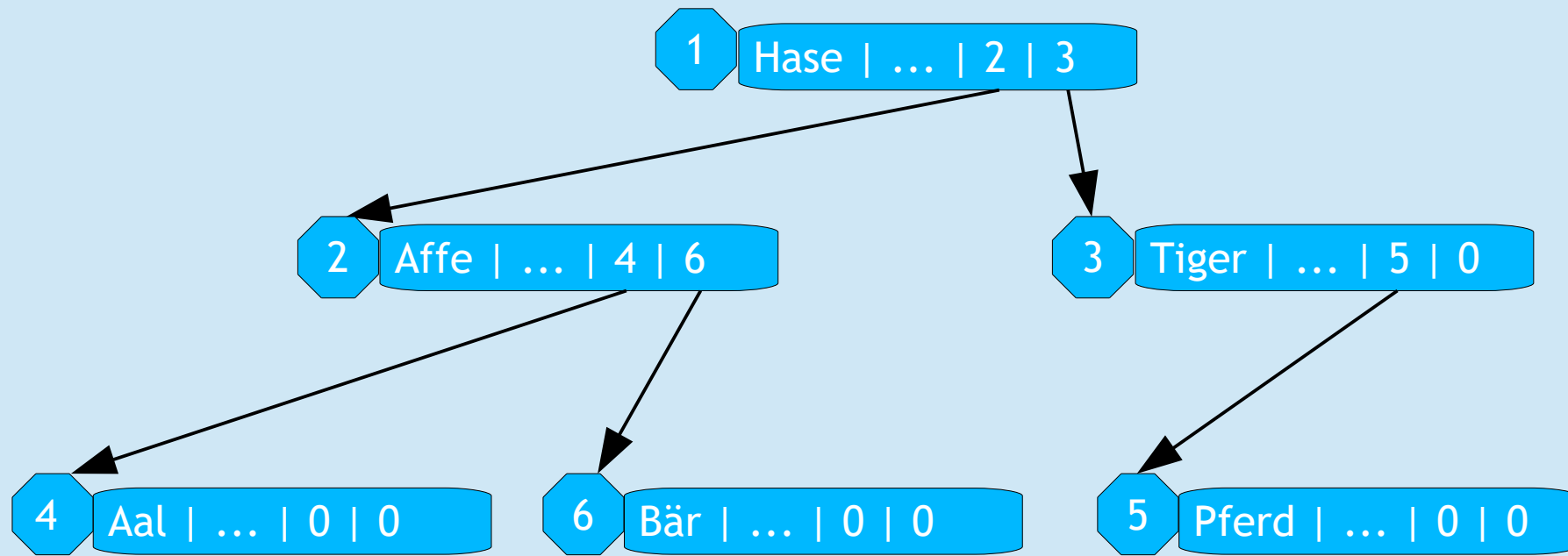
Hase

Affe

Tiger

Aal

Pferd



Alphabetische Ordnung

Hase

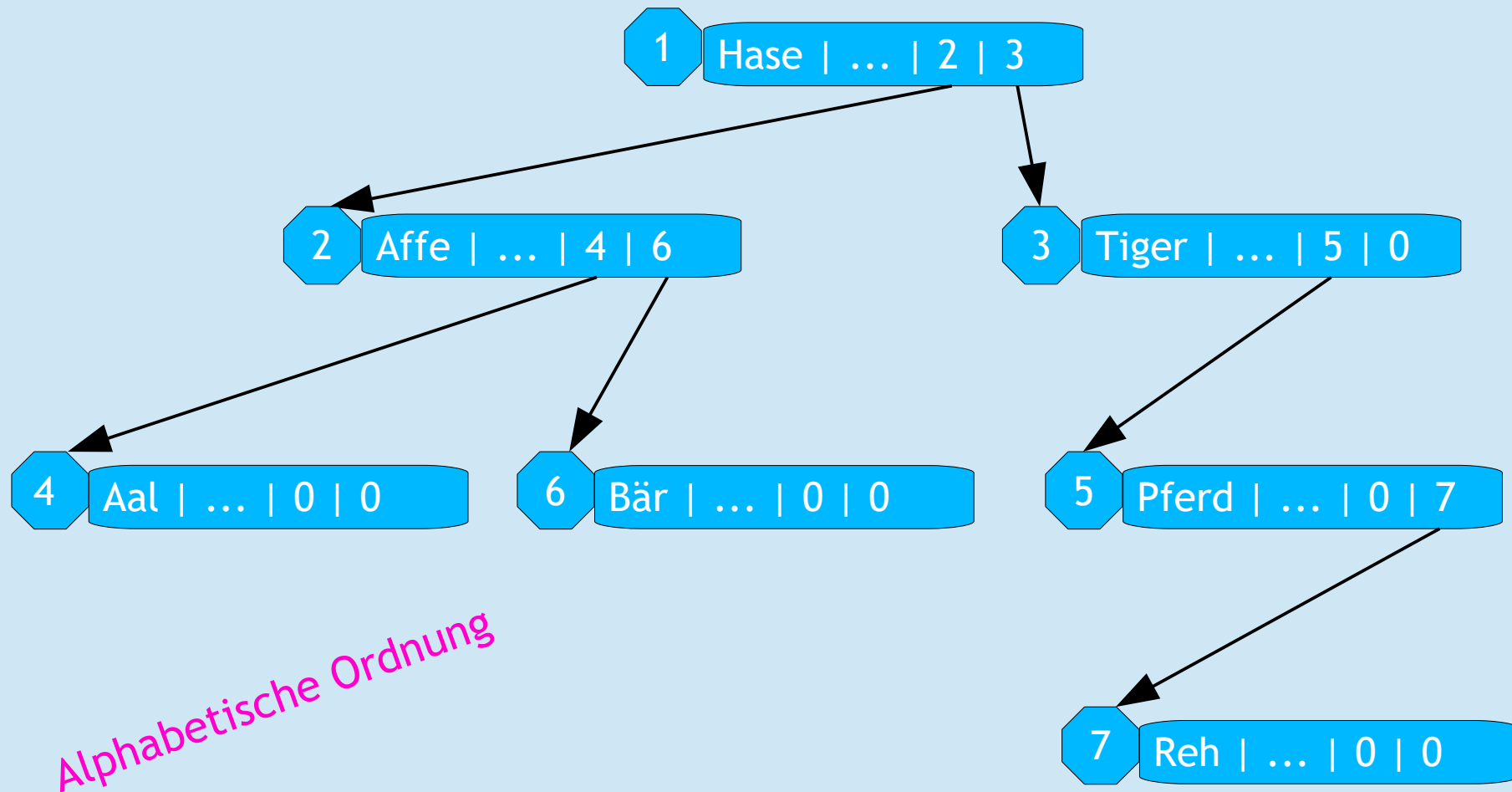
Affe

Tiger

Aal

Pferd

Bär



Hase

Affe

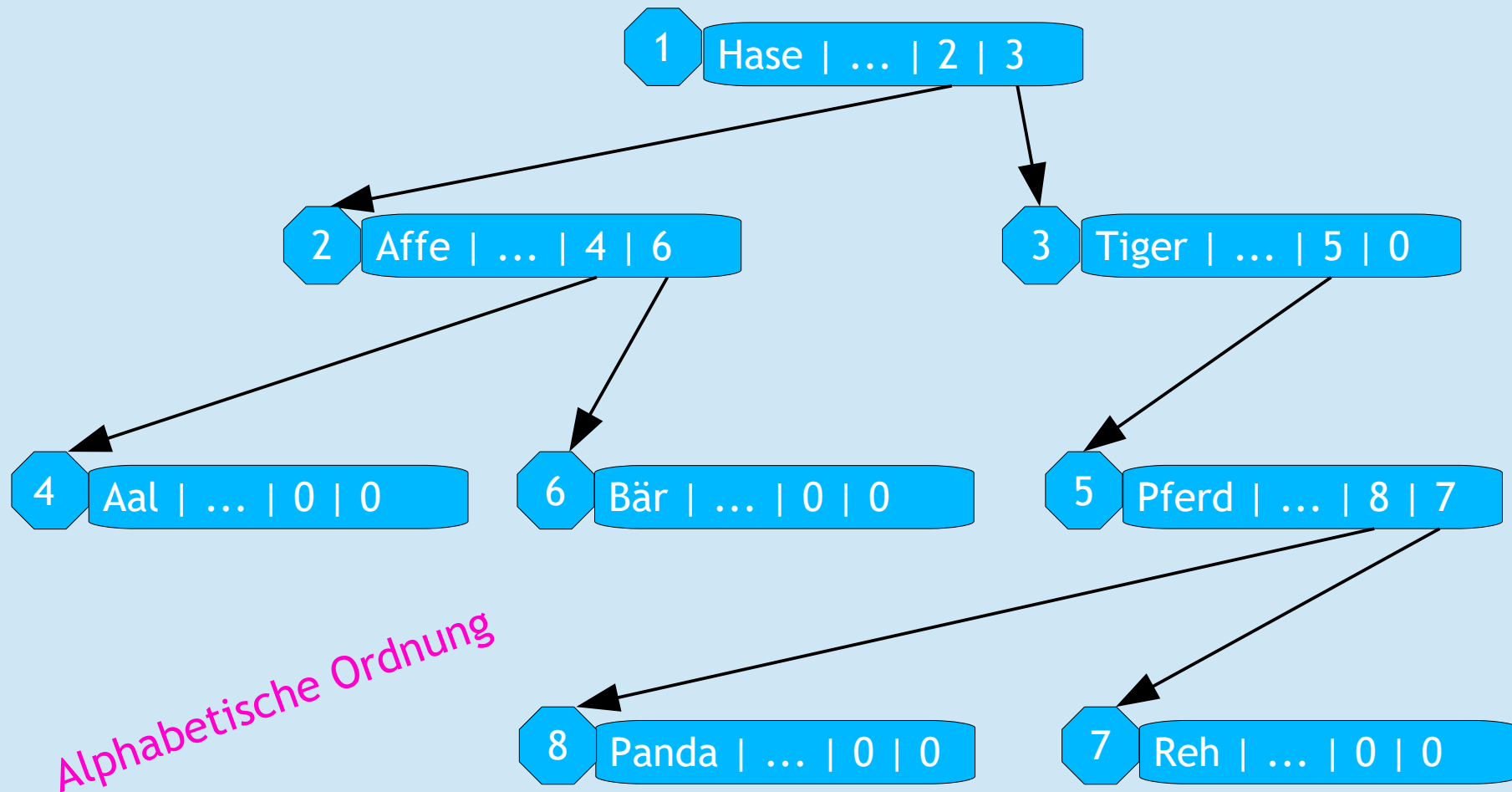
Tiger

Aal

Pferd

Bär

Reh



Hase

Affe

Tiger

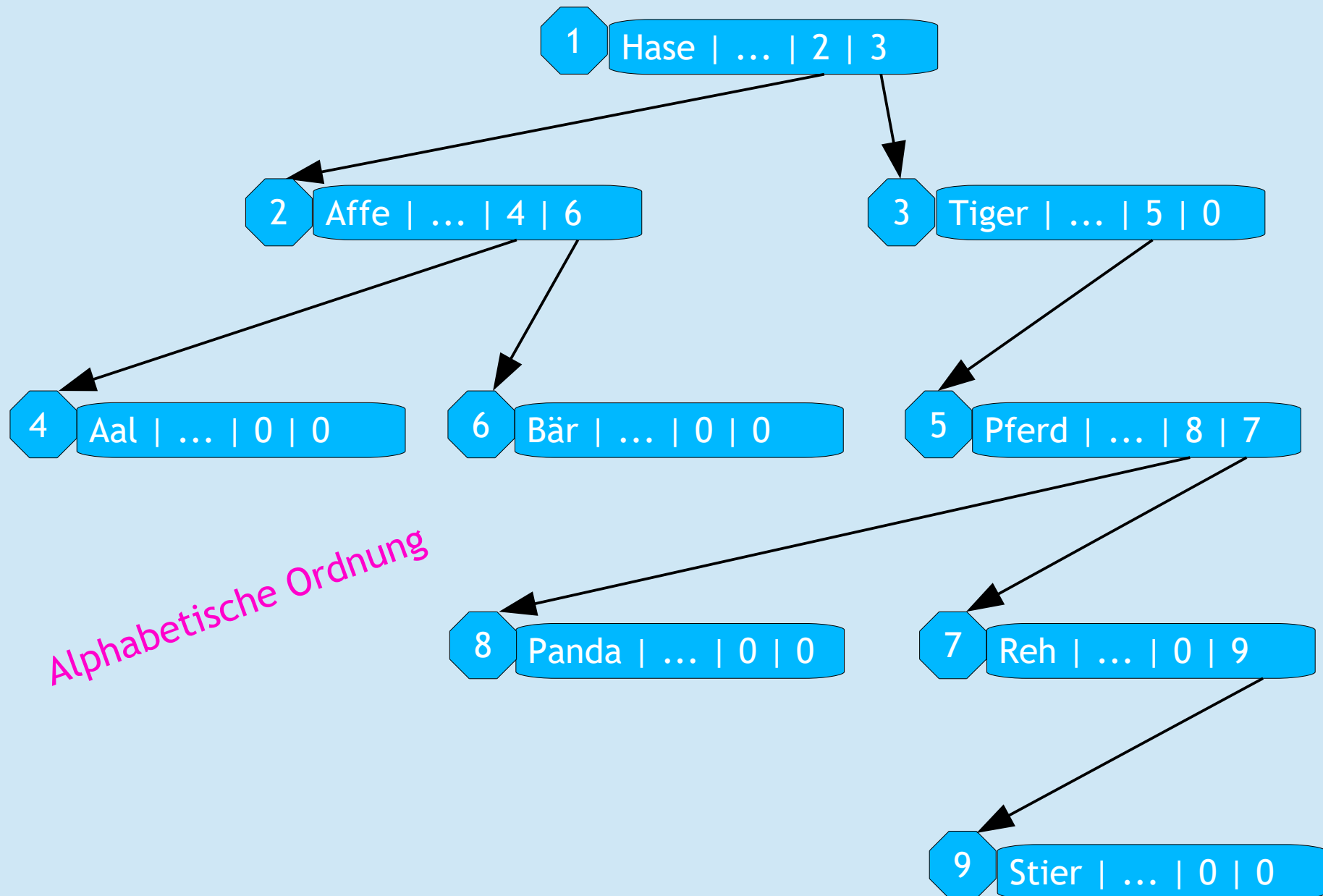
Aal

Pferd

Bär

Reh

Panda



Alphabetische Ordnung

Hase

Affe

Tiger

Aal

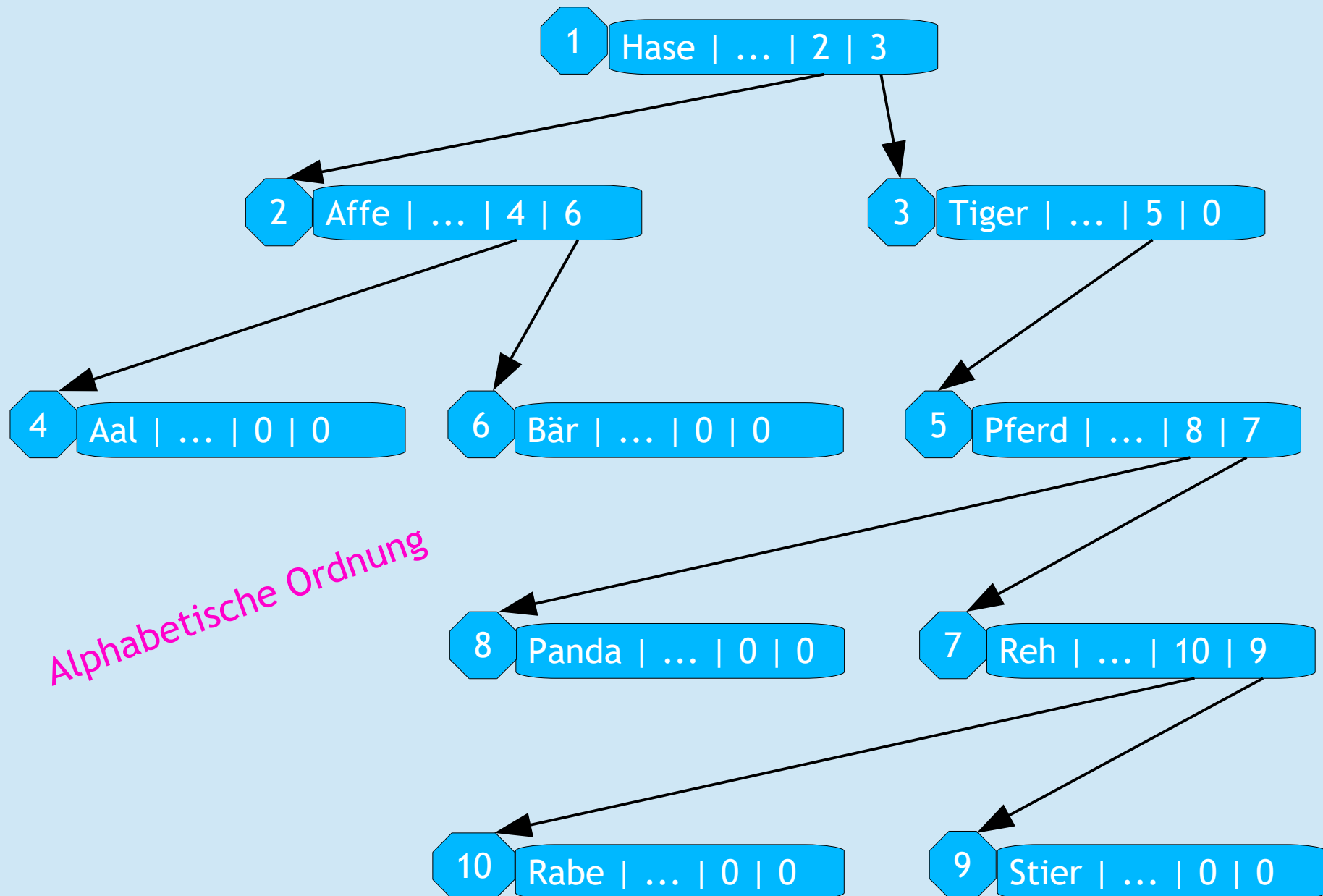
Pferd

Bär

Reh

Panda

Stier



Alphabetische Ordnung

Hase

Affe

Tiger

Aal

Pferd

Bär

Reh

Panda

Stier

Rabe



*Hierarchische Datenbanken passen sehr gut für die Abbildung solcher Anwendungsbereiche wie Dateisysteme mit Verzeichnissen und Dateien, LDAP, Internet-Domänen, Stücklisten.*

*Die Datensätze können nicht zwei, sondern viel mehr Verweise enthalten, somit kann man nicht nur binäre, sondern auch n-äre Bäume aufbauen.*

*Die einzelnen Datensätze können durchaus Verweise auf die anderen Typen der Strukturen enthalten. Somit können die Knoten von einem Baum auf die Bäume mit anders strukturierten Datensätzen verweisen. Die Beziehungen in diesen anderen Unterbäumen werden wieder als Programme realisiert. Solche Bäume/Unterbäume entsprechen den Entitätstypen.*

*Nicht hierarchische Beziehungen müssen anders implementiert werden, z.B. durch linear oder doppelt verkettete Listen, wobei jede Liste einem Knoten zugeordnet wird. Ein Baum selbst kann auch gleichzeitig eine verkettete Liste sein.*



*Die älteste, ständig weiterentwickelte und bis heute eingesetzte hierarchische Datenbank ist Informationsmanagement System von IBM (IMS).*

*Komponenten von IMS:*

- Datenbank selbst – Daten und Funktionalität;*
- IMS Explorer – Zugriff auf die Daten (anzeigen, editieren), Entwicklung der Anwendungen;*
- IMS SOAP Gateway – Kommunikation mit Web-Services mittels Simple Object Access Protocol unabhängig von Plattform;*
- IMS Java Connector – Entwicklung der eigenständigen IMS-Clients unter Java;*
- IMS Data Provider .NET – Entwicklung von .NET-Anwendungen mit SQL-Abfragen an IMS-Datenbank.*



*Netzartige Datenbanken entstehen aus den hierarchischen, indem man die Verweisfelder für Rückwärtsbewegung in den Datensatz einfügt. Somit stellen sie eine Verallgemeinerung der hierarchischen Datenbanken dar.*

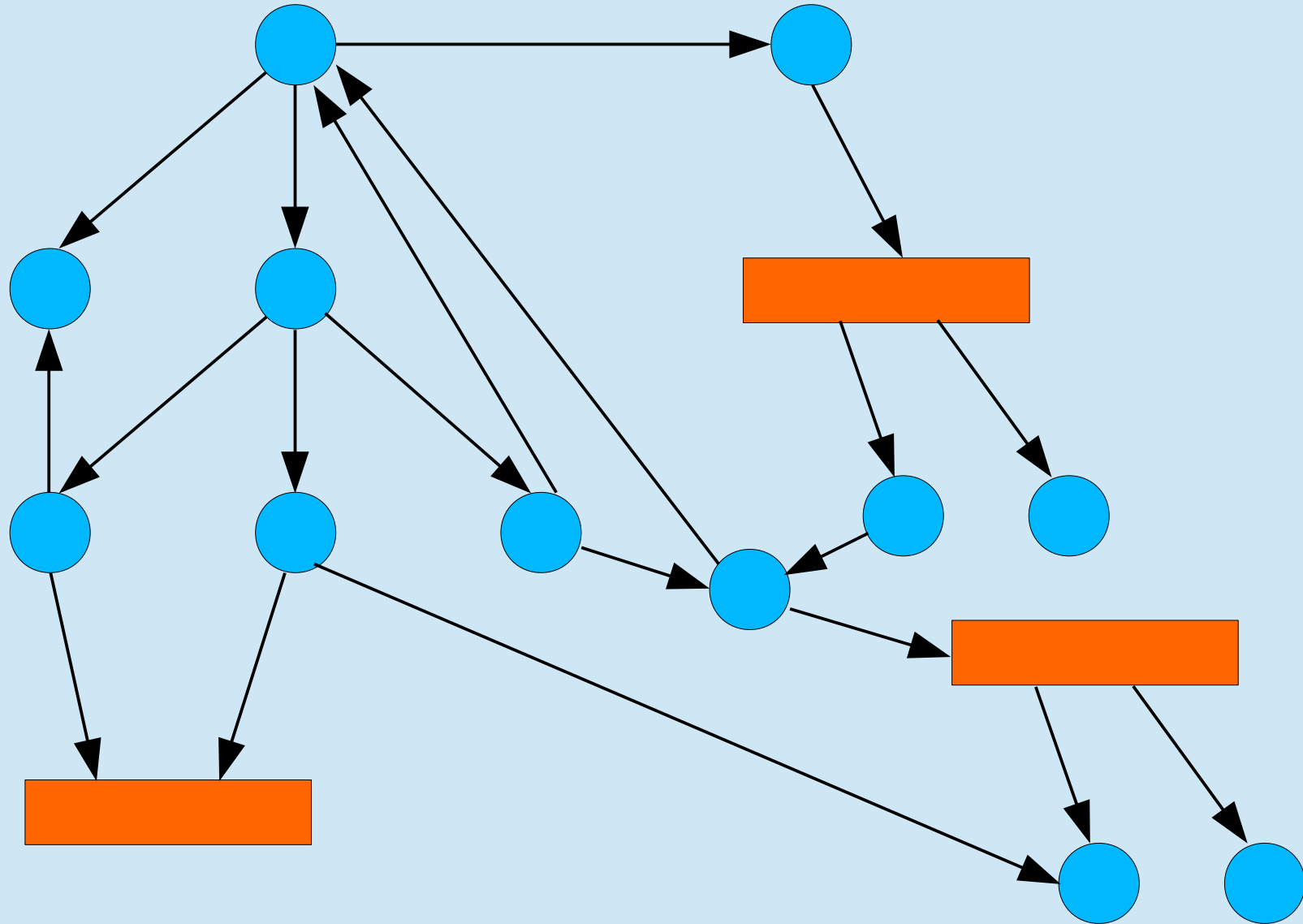
*Hier sind auch unterschiedliche Typen der Datensätze vertreten (unterschiedliche Entitätstypen). Durch besondere Art der Datensätzen werden auch die Beziehungen modelliert.*

*Vorteile:*

- Theoretisch sind vielfältige Verweise möglich.*

*Nachteile:*

- Unflexible netzartige Struktur.*
- Modellierung der Beziehungen ist eingeschränkt.*
- Saubere Trennung der Entitätstypen ist kaum möglich.*





*Netzartige Datenbanken passen für die Abbildung der netzartigen Strukturen, wie geografische Elemente, Makrostrukturen des Internets. Jeder Knoten kann mehrere Vorgänger und mehrere Nachfolger haben. Semantische Darstellung der Beziehungen wird auch meistens programmiert.*

*Ein bekannter Vertreter der netzartigen Datenbanken ist Universal Datenbank System von Siemens, (UDS).*

*Etwa 20 Jahre waren die netzartigen und hierarchischen Datenbanken dank ihren überragenden Leistungen im Einsatz, und dann wurden sie durch relationale verdrängt.*



## Definition.

Verteilte Datenbank ist ein Verbund von mehreren (meist relationalen) Datenbanken.

## Definition.

Verteilte Datenbank ist eine Zusammenfassung von Informationseinheiten, die sich auf mehreren Knoten (Computern) befinden, die über ein Netzwerk verbundenen sind.

Die Metadaten (Informationen über Komponenten dieses Verbundes, Zugriffsdaten auf diese Datenbanken) werden ebenfalls in einer Datenbank zusammengefasst (übergeordnete Datenbank).

Die Anwendungen greifen auf die verteilte Datenbank zu, und starten somit verteilte Transaktionen. Verteilte Transaktion besteht aus Transaktionen in den Komponenten-Datenbanken, aus Teil-Transaktionen. Bearbeitung der verteilten Transaktionen ist recht kompliziert, weil sie von Erfolg oder Mislungen der Teil-Transaktionen abhängig ist.

Horizontale Skalierung



## Vorteile der verteilten Datenbank:

- *optimale Darstellung der Unternehmensstruktur durch lokale Datenbestände;*
- *Unabhängigkeit der Teil-Datenbanken und deren Anwendungen von der anderen Knoten der verteilten Datenbank;*
- *Unabhängigkeit der verteilten Datenbank vom Ort;*
- *Plattform- und Netzwerkunabhängigkeit;*
- *ständiger Betrieb der verteilten Datenbank;*
- *Effizienz durch parallele Verarbeitung in den Teil-Datenbanken;*
- *Daten werden sowohl lokal als auch übergreifend verwendet;*
- *Transparenz der Anfragen und Anweisungen;*
- *Ergebnis- statt Sourcedaten-Transfer.*



## Nachteile der verteilten Datenbank:

- *Vorbereitungsaufwand – Konzepte, Planung, Koordination, Zugriffsmöglichkeiten;*
- *zusätzliche Administration der Datenbank mit Metadaten – Backup/Restore, Export/Import, Pflege der Konsistenz;*
- *aufwendige Entwicklung der Abfragen und Anweisungen;*
- *Abhängigkeit der Operationen von Lauffähigkeit der einzelnen Teil-Datenbanken.*



*Beim Entwurf einer verteilten Datenbank ist es so vorzugehen, wie auch bei konventionellen relationalen Datenbanken, es gibt aber noch zusätzliche Schritte:*

- *Fragmentierungsschema erstellen;*
- *Zuordnungsschema erstellen.*



*Bevor man Fragmentierungsschema erstellt, muss man ein allgemeines Schema haben, das die Metadaten (Feldname, Datentypen) aller Tabellen enthält.*

*Auf Basis der Zugriffsanforderungen auf die Daten werden Relationen in disjunkte Fragmente zerlegt:*

- *horizontale Zerlegung – mehrere Datensätze nach bestimmtem Kriterien werden zusammengefasst, z.B. es gibt oft Abfragen, die nur Professoren, oder nur Lehrbeauftragte, oder nur Mitarbeiter betreffen, entsprechend  
Status= 'Prof' oder Status= 'LB' oder Status= 'MA'*
- *vertikale Zerlegung – Attribute werden zusammengefasst, die nach einem bestimmten Muster abgefragt werden, z.B. Vorname und Nachname braucht man oft zusammen;*
- *kombinierte Zerlegung – horizontale und vertikale Zerlegung.*



## Anforderungen an Fragmentierung:

- *Vollständigkeit* – durch Zusammenlegung (Rekonstruktion) der Fragmente entsteht (logisch gesehen) die vollständige Datenbank, keine Teile sind verloren gegangen;
- *Disjunktion* – Fragmente überlappen sich nicht.

*Um diese Anforderungen zu erfüllen, muss man umfangreiche Kenntnisse über den Anwendungsbereich und über die geplanten/gewünschten Anfragen sowie Bearbeitungsanweisungen besitzen.*



*Bei der Zuordnung (Allocation) werden die erstellten Fragmenten auf die Knoten der verteilten Datenbank zugeteilt. Theoretisch muss diese Zuordnung redundanzfrei sein. Aus Sicherheits- oder Leistungsgründen wird die Replikation organisiert.*

*Auf jedem Knoten muss entsprechendes Teilschema des gesamten Schemas lokal implementiert werden.*

*Laut diesen Zuordnungen werden die Abfragen transparent zerlegt und an die Knoten verschickt. Im besten Fall arbeiten die Benutzer nur mit dem gesamten Schema und brauchen keinerlei Wissen über Fragmentierung oder Zerlegung. Selbstverständlich gibt es Ausnahmen.*

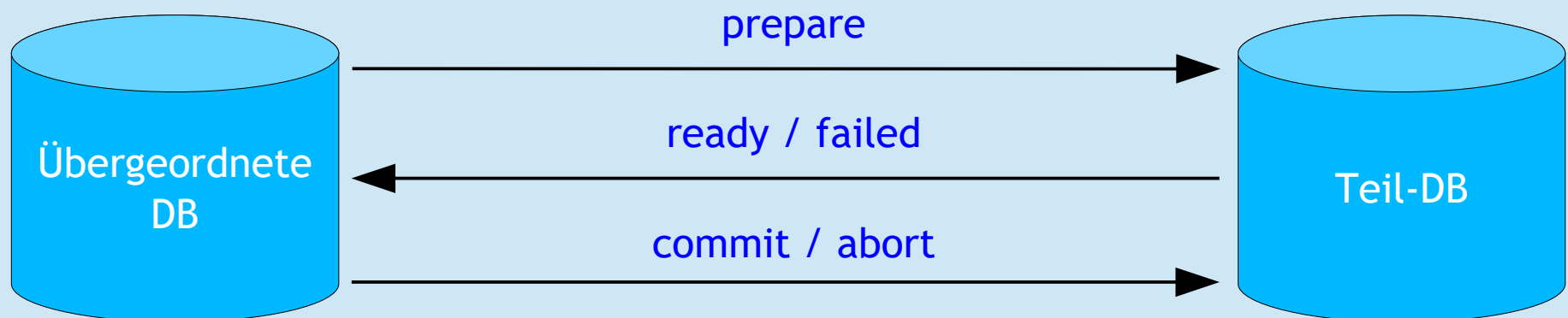


Die Datenbank (genau gesagt DBMS) muss fähig sein, eine globale Transaktion in Teil-Transaktionen zu zerlegen, an die zuständigen Knoten zu verschicken und die Antworten auszuwerten.

Globale Transaktion folgt dem allgemeinen Begriff – entweder werden alle Teil-Transaktionen erfolgreich durchgeführt, oder die Zustände aller Teil-Datenbanken werden zurückgesetzt.

Logischerweise muss jede Teil-Datenbank die UNDO- und REDO-Funktionalität beherrschen.

Die übergeordnete Datenbank kommuniziert mit den Teil-Datenbanken durch spezielle Nachrichten, um Transaktionen zu verwalten:





*Die Idee der Objektorientierung ist recht schwer mit den Datenbanken zu vereinbaren.*

*Die Idee der Objektorientierung besteht darin, die Daten und die dafür definierten Aktionen (Methoden) zusammen zu bringen. Jegliche Manipulation mit den Daten eines Objektes ist nur durch vordefinierte Methoden dieses Objektes erlaubt. Somit können die Objekte nicht willkürlich auf die Daten eines anderen Objektes zugreifen und sie verletzen (bewusst oder nicht bewusst). Die Beziehungen zwischen den Daten sind dabei nicht von Interesse.*

*Die Idee der Datenbanken besteht in Modellieren der Daten, d.h. die zentrale Stelle nehmen die Daten und deren Beziehungen zu einander. Dieses Modell muss von Aktionen (Programmen, Anwendungen, Methoden) unabhängig sein.*

*Es gibt aber viele Versuche, Objektorientierung und Datenbanken zusammen zu führen, – allerdings mit unterschiedlichem Erfolgsgrad.*



## Eigenschaften:

- *Die Objektdatenbanken zeigen sich vorteilhaft bei komplexen Anwendungen mit Klassenhierarchien unter Daten.*
- *Recht komplexe Objekten können gespeichert und abgerufen werden.*
- *Keine Normalisierung der Daten.*
- *Die Abfragen sind langsamer im Vergleich zu relationalen Datenbanken, auch wenn die Abfragen ziemlich einfach sind.*
- *Die Datenbanken sind zu Anwendungen allgemein nicht besonders kompatibel.*



*Eine aktuelle objektorientierte Datenbank, wie **db4o**, ist durch ihre geringe Speichergröße bekannt und passt am besten für die Java- und .NET-Anwendungen.*

*Diese Datenbank unterstützt keine bestimmte Abfragesprache wie SQL, sondern die Abfragen basieren auf den Strukturen der jeweiligen Programmiersprache. Das könnte gleichzeitig als Vor- oder auch als Nachteil verstanden werden. Die QBE (Query By Example) wird aber unterstützt.*

*Diese Datenbank unterstützt Transaktionen, und somit die Operationen COMMIT, ROLLBACK.*

*Ein sehr gutes Beispiel dieser Datenbank befindet sich da:*

*<https://www.theserverside.de/eine-kurze-einfuehrung-in-db4o/>*



*Oracle liefert einen passenden (bis zu einem bestimmten Grad) objektorientierten Ansatz innerhalb des relationalen Modells.*

*Es gibt Objekttypen, die Analog zu Klassen darstellen. Sie enthalten sowohl die Daten von unterschiedlichen Typen als auch Funktionen und Prozeduren. In PL/SQL kann man von diesen Objekttypen die Variablen oder Tabellen instanziiieren und damit objektorientierte Programme schreiben.*



*No-SQL-Datenbanken = not only SQL-Datenbanken.*

*Dieser Begriff betont, dass noch die anderen Datenbanken existieren, als relationale. Eine relationale Datenbank ohne SQL-Unterstützung ist eher eine Exotik.*

*Relationale Datenbanken sind für klassische Anwendungen mit strengen Anforderungen an Konsistenz und Sicherheit immer noch von Bedeutung.*



## Gründe für No-SQL-Datenbanken:

- *Auswertung von enorm großen Datenvolumen (in TiB- und PiB-Bereichen).*
- *Die Abfragen müssen kurze Laufzeit haben.*
- *Konsistenz der Daten ist nicht vom großen Interesse. Constraints und Transaktionen werden kaum eingesetzt, weil die Daten meistens nur abgefragt werden (keine INSERT/UPDATE/DELETE).*
- *Verfügbarkeit wird vor der Konsistenz bevorzugt, z.B. DNS.*
- *Speicher für relationale Datenbanken wird teuer.*

Warum No-SQL?



## *Eigenschaften für No-SQL-Datenbanken:*

- *Datenmodelle haben ein einfaches Schema, oder haben schwache Schemarestriktionen, oder sind überhaupt schemafrei.*
- *Horizontale und vertikale Skalierung ist einfach. Horizontale Skalierung der Datenbanken wird vor der vertikalen bevorzugt.*
- *Sie folgen dem BASE-Prinzip statt ACID, wie bei relationalen Datenbanken.*
- *Keine Normalformen, keine JOINS. Denormalisierter Zustand wird bevorzugt.*
- *Hohe Leistung und Verfügbarkeit dank Replikation der Daten.*



*Basically Available – Gewährleistung der Verfügbarkeit möglicherweise zum Nachteil der Konsistenz.*

*Soft State – Verantwortung für Konsistenz wird an den Entwickler delegiert.*

*Eventually Consistent – die Abfragen sind erlaubt, auch wenn der Zustand nicht konsistent ist.*

**BASE-Prinzip**

**ACID-Prinzip**

*Atomic – Transaktion kann von einer anderen Transaktion nicht unterbrochen werden, sie wird von Anfang bis zu Ende ausgeführt, wie eine atomare Operation.*

*Consistent – Transaktion gefährdet den konsistenten Zustand der Datenbank nicht.*

*Isolated – wenn Transaktionen mit einander agieren, dann verletzen sie einander nicht.*

*Durable – fällt eine Transaktion aus, dann gefährdet sie eigene Daten nicht.*



## *Vertreter der No-SQL-Datenbanken:*

- *Document-Stores*
- *Key-Value-Stores*
- *Wide-Column-Stores*
- *XML-Datenbanken*
- *Graph-Datenbanken*



## Eigenschaften:

- *Dokumentorientierte Datenbanken sind schemafrei.*
- *Daten werden in Dokumenten gespeichert, die jeweils eine eindeutige ID haben. Dokumente entsprechen den Datensätzen.*
- *Datensätze haben keine einheitliche Struktur und können verschachtelt sein.*
- *Die Felder (Spalten) in einem Datensatz können verschiedene Datentypen und sogar mehr als einen Wert haben (Arrays).*
- *Alle zusammenhängenden Daten sind in einem Dokument gespeichert.*
- *Hohe Skalierbarkeit.*
- *Operationen für die in den Dokumenten gespeicherten Daten müssen programmiert werden (Objektorientierung?).*



*Die aktuellen dokumentorientierten Datenbanken, wie MongoDB, CouchDB, verwenden für ihre Dokumente halb-strukturierte Datenformate, ähnlich wie XML und JSON.*

*Dadurch kann man für den Zugriff auf die Daten schon bewehrte Algorithmen verwenden, was eine Entwicklung der Anwendungen einfacher macht.*

*Andererseits ist Datenredundanz möglich, weil die Normalisierung fehlt. Die Datenbank kann auch keine Konsistenz der Daten gewährleisten – diese Funktionalität wird an die Anwendung übertragen.*



## Eigenschaften:

- *Im System werden nur die Schlüssel und dazu gehörige Werte gespeichert. Zu einem angegebenen Schlüssel wird entsprechender Wert geliefert. Das Schema ist ziemlich einfach.*
- *Schlüssel können sortiert werden, was geordnete Bearbeitung der Werte ermöglicht.*
- *Als Werte kommen Zahlen, Zeichenketten, Listen, Dokumente, Tabellen u.s.w. in einer Datenbank zum Einsatz, sodass die Werte der Datenbank nicht einheitlicher Natur sein müssen.*
- *Die Daten können sich im Hauptspeicher (In-Memory) befinden, was zur hohen Leistung führt, oder auch auf dem Datenträger (On-Disc).*



*Die aktuellen Key-Value-Datenbanken, wie Berkeley DB, Redis, Amazon DynamoDB, finden ihren Einsatz in Online-Geschäften und in Schemen, wo eine einfache Verbindung zwischen den Schlüsseln und Werten vorgesehen ist.*

*Einerseits erlauben die Key-Value-Stores sehr schnelle einfache Abfragen, andererseits sind die komplexen Beziehungen zwischen den Entitätstypen (und die komplexen Abfragen) recht schwer zu implementieren. Konsistenz wird durch die Anwendungen gewährleistet.*

*Die Key-Value-Datenbanken sind eigentlich mit dokumentenbasierten Datenbanken verwandt, weil Dokumente anstelle der Werte auftreten können.*

*Für Key-Value-Datenbanken ist die Sprache SQL verwendbar und manchmal auch realisiert (Berkeley DB).*



## Eigenschaften:

- *Datensätze in der Datenbank haben unterschiedliche Struktur, somit ist die Datenbank schemafrei.*
- *Innerhalb von einem Datensatz können Milliarden von Feldern existieren.*
- *Struktur eines Feldes: Name der Spalte (wird als Schlüssel verwendet und ist variabel vom Datensatz zu Datensatz!), Daten, Zeitstempel.*
- *Daten werden linear, Datensatz für Datensatz gespeichert.*
- *Es können zusammenhängende Spalten existieren (column family). Die Daten in solchen Spalten werden zusammen gespeichert, ähnlich wie in spaltenorientierten relationalen Datenbanken.*



Die aktuellen Wide-Column-Datenbanken, wie MS Azure Cosmos DB, Cassandra, sind gut skalierbar und deswegen eignen sich sehr gut für Verarbeitung der großen Datenvolumen in Big Data Analytics, wie in DWH.

Die Schreibzugriffe sind langsam, wenn mehr als eine Spalte geschrieben wird.

Die Wide-Column-Datenbanken darf man mit den spaltenorientierten relationalen Datenbanken nicht verwechseln. Die spaltenorientierten relationalen Datenbanken haben selbstverständlich ein festes Schema und speichern die Daten (physisch gesehen) spaltenweise und nicht zeilenweise, wie übliche relationale Datenbanken.

Sirius	3
Rigel	17
Altair	2

~~Sirius 3 Rigel 17 Altair 2~~

Sirius Rigel Altair 3 17 2



## Eigenschaften:

- *Daten werden in Dateien im XML-Format gespeichert.*
- *Als Abfragesprache wird XPath eingesetzt, um die XML-Dokumente auszuwerten.*
- *Für XML existieren viele gut entwickelte Algorithmen, um die gespeicherten Daten zu manipulieren.*

*Die aktuellen XML-Datenbanken, wie **BaseX**, stellen recht einfach und schnell die ganzen Dokumente zur Verfügung (z.B. Rechnungen). Andererseits aber zeigen sie deutliche Verzögerungen bei Verarbeitung der großen Mengen von Daten. Außerdem fehlen solche wichtige Operationen, wie COMMIT, ROLLBACK.*

*Die XML-Datenbanken sind eher keine Datenbanken, sondern Verarbeitungssysteme für XML-Dokumente, was eigentlich ihren Wert auf keinen Fall mindert.*



## Eigenschaften:

- *In diesen Datenbanken werden die Daten in drei Klassen unterteilt: Knoten, Attribut und Kante. Die Daten werden also als Graphen gespeichert.*
- *Ein Knoten hat beliebig viel Attributen.*
- *Eine Kante beinhaltet Information, welcher Knoten mit welchem verbunden ist.*
- *Einzelne Knoten realisieren die Tupel (Datensätze), und die Kanten stellen die Beziehungen dar.*

*Die Graph-Datenbanken stellen sehr einfach die Beziehungen dar und werden in den Anwendungen effektiv eingesetzt, wo Informationen "wer-kennt-wen" wichtig sind. Andererseits existiert keine einheitliche Abfragesprache für diese Datenbanken, und keine direkten Zugriffe auf Knoten mittels Attributen sind vorgesehen.*



Die No-SQL-Datenbanken sind sehr effektiv  
in ihrem Spezialgebiet.

Ende

