L19: NoSQL

CS3200 Database design (fa18 s2)

https://northeastern-datalab.github.io/cs3200/

Version 11/15/2018

Several slides courtesy of Benny Kimelfeld

Announcements!

		NoSQL		
19	R Nov 15	NoSQL		Q11 (Nov 29)
20	M Nov 19	NoSQL	**Harrison**	
	R Nov 22	No class: Thanksgiving		
21	M Nov 26	NoSQL	**Sadalage, Fowler**	
		Query Processing and Database Internals		
22	R Nov 29	Relational Algebra & Query Optimization		Q12, HW7
23	M Dec 3	Course Evaluation, Class Review		HW8, HW10 Optional PPTX
	R Dec 6	No class: Reading day		
	T Dec 11	**Exam 3**: 8am-10am, location: TBD		

- 4. NoSQL
  - **Sadalage, Fowler**: NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. 2012 [Safari books eBook (NEU free online access)]
  - **Harrison**: Harrison. Next Generation Databases: NoSQL, NewSQL, and Big Data. 2016 [Safari books eBook (NEU free online access)]

Announcements!

- HW7: extra time and merged (Thanksgiving!)
- Today
  - NoSQL (for 2-3 lectures): big pictures and hands-on example
  - Poll in class: install systems, or with Jupyter interface
    - Jupyter: easy to get started, same concepts, but no hands dirty
    - Real deal: takes time to setup, but you get hands-on experience
  - Monday: lessons learned from exam 2


11455

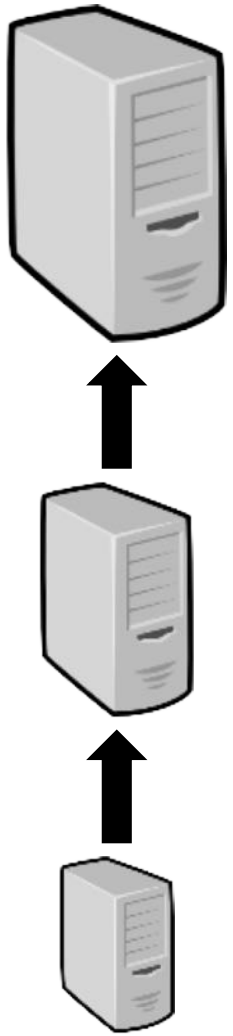
Outline

- Introduction
- Transaction Consistency
- 4 main data models
  - Key-Value Stores (e.g., Redis)
  - Column-Family Stores (e.g., Cassandra)
  - Document Stores (e.g., MongoDB)
  - Graph Databases (e.g., Neo4j)
- Concluding Remarks

SQL Means More than SQL

- SQL stands for the query language
- But commonly refers to the traditional RDBMS:
  - Relational storage of data
    - Each tuple is stored consecutively (per row, row-wise)
  - Joins as first-class citizens
    - In fact, normal forms prefer joins to maintenance
  - Strong guarantees on transaction management
    - No consistency worries when many transactions operate simultaneously (concurrently) on common data
- Focus on scaling up
  - That is, make a single machine do more, faster

Vertical vs. Horizontal Scaling



"scaling up"

- **Vertical scaling ("scale up")**: you scale by adding more power (CPU, RAM)
- **Horizontal scaling ("scale out")**: you scale by adding more machines

"scaling out"

Trends Drive Common Requirements

Social media + mobile computing

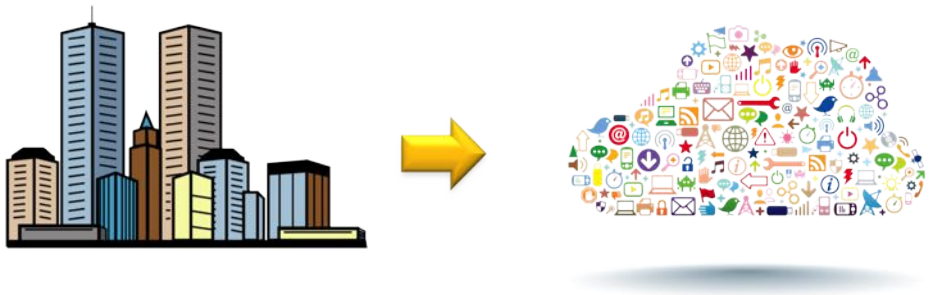
- Explosion in data, always available, constantly read and updated
- High load of simple requests of a common nature
- Some consistency can be compromised (e.g., 👍)

Cloud computing + open source

- Affordable resources for management / analysis of data
- People of various skills / budgets need software solutions for distributed analysis of massive data

Database solutions need to scale out
(utilize distribution, “scale horizontally”)

Compromises Required



What is needed for effective distributed, data- and user-intensive applications?

1. Use data models and storage that allow to avoid joins of big objects
2. Relax the guarantees on consistency

NoSQL

- Not Only SQL
  - May still support SQL-type languages
  - Term introduced by Carlo Strozzi in 1998 to describe an alternative database model
  - Became the name of a movement following Eric Evans's reuse for a distributed-database event
- Seminal papers:
  - Google's BigTable
    - Chang, Dean, Ghemawat, Hsieh, Wallach, Burrows, Chandra, Fikes, Gruber: Bigtable: A Distributed Storage System for Structured Data. OSDI 2006: 205-218
  - Amazon's DynamoDB
    - DeCandia, Hastorun, Jampani, Kakulapati, Lakshman, Pilchin, Sivasubramanian, Vosshall, Vogels: Dynamo: amazon's highly available key-value store. SOSP 2007: 205-220

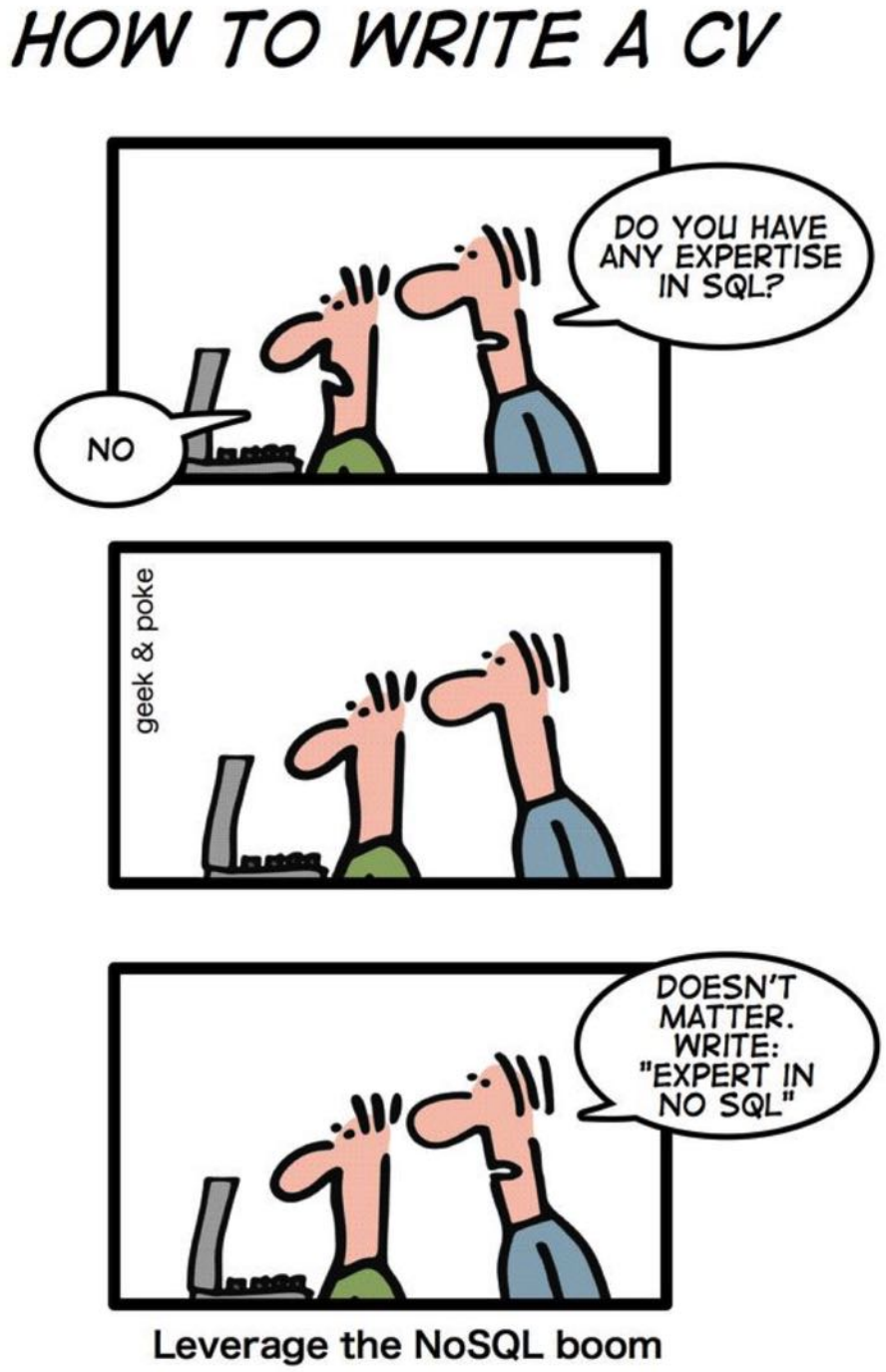
NoSQL from nosql-database.org

“

- Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontally scalable.
- The original intention has been modern web-scale databases. The movement began early 2009 and is growing rapidly. Often more characteristics apply such as: schema-free, easy replication support, simple API, eventually consistent / BASE (not ACID), a huge amount of data and more.
- So the misleading term “nosql” (the community now translates it mostly with “not only sql”) should be seen as an alias to something like the definition above.

”

What is NoSQL?



Source: Geek and Poke: http://geekandpoke.typepad.com/geekandpoke/2011/01/nosql.html

Common NoSQL Features

- Non-relational data models
- Flexible structure
  - No need to fix a schema, attributes can be added and replaced on the fly
- Massive read/write performance; availability via horizontal scaling
  - Replication and sharding (data partitioning, we'll discuss that next)
  - Potentially thousands of machines worldwide
- Open source (very often)
- APIs to impose locality (opposite of joins)

When the database grows: Partitioning Tables

Key	Product Name	Short Description	Review	Picture
01	Americano @ Starbucks	Black, no sugar	I'd buy again	
02	BB @ Seattle's Best	Black, no sugar	The best	
03	TB @ Zoka Coffee	Black, no sugar	It's okay	
04	BC @ Coffee	Black, no sugar	Never again	

Source: http://cloudgirl.tech/data-partitioning-vertical-horizontal-hybrid-partitioning/

Vertical Partitioning

Key	Product Name	Short Description	Review	Picture
01	Americano @ Starbucks	Black, no sugar	I'd buy again	
02	BB @ Seattle's Best	Black, no sugar	The best	
03	TB @ Zoka Coffee	Black, no sugar	It's okay	
04	BC @ Coffee	Black, no sugar	Never again	

Key	Product Name	Review
01	Americano @ Starbucks	I'd buy again
02	BB @ Seattle's Best	The best
03	TB @ Zoka Coffee	It's okay
04	BC @ Coffee	Never again

Key	Short Description	Picture
01	Black, no sugar	
02	Black, no sugar	
03	Black, no sugar	
04	Black, no sugar	

Source: http://cloudgirl.tech/data-partitioning-vertical-horizontal-hybrid-partitioning/

Horizontal Partitioning ("sharding")

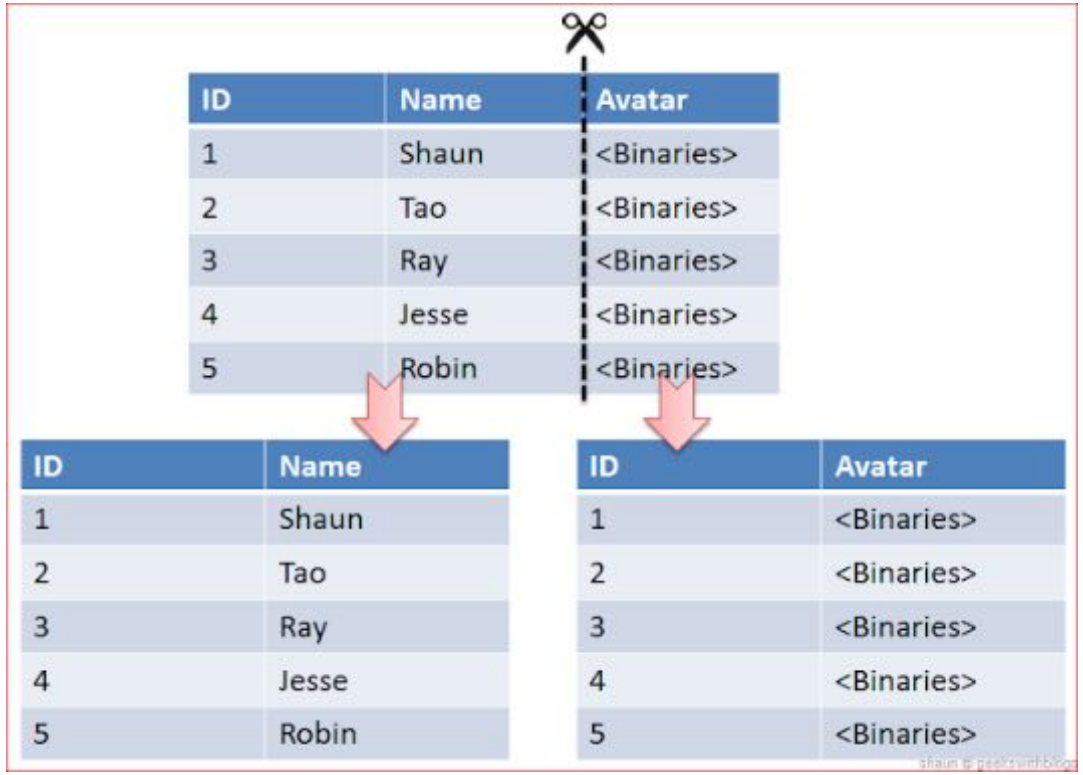
Key	Product Name	Short Description	Review	Picture
01	Americano @ Starbucks	Black, no sugar	I'd buy again	
02	BB @ Seattle's Best	Black, no sugar	The best	
03	TB @ Zoka Coffee	Black, no sugar	It's okay	
04	BC @ Coffee	Black, no sugar	Never again	

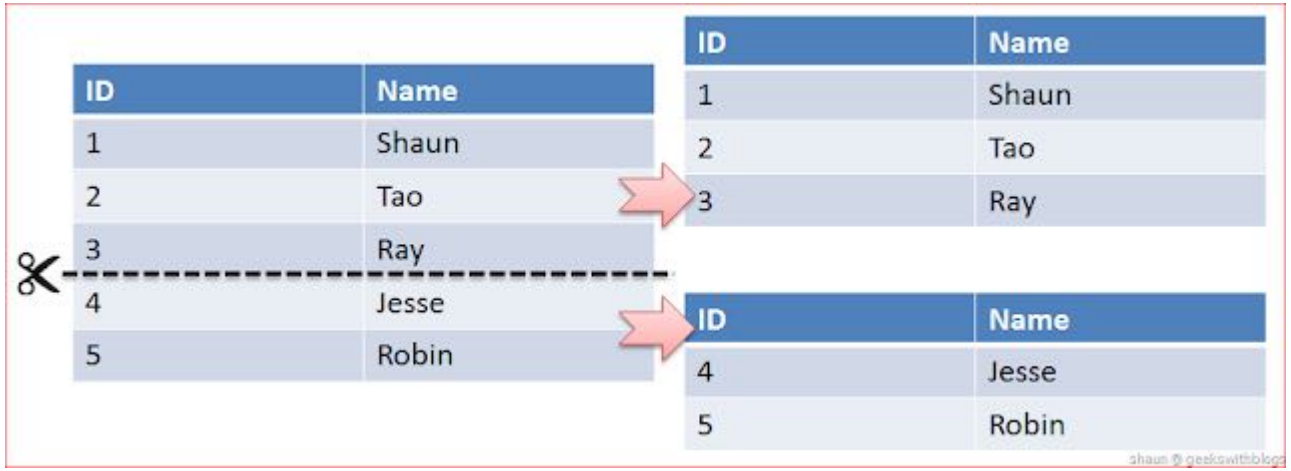
Key	Product Name	Short Description	Review	Picture
01	Americano @ Starbucks	Black, no sugar	I'd buy again	
02	BB @ Seattle's Best	Black, no sugar	The best	

Key	Product Name	Short Description	Review	Picture
03	TB @ Zoka Coffee	Black, no sugar	It's okay	
04	BC @ Coffee	Black, no sugar	Never again	

Source: http://cloudgirl.tech/data-partitioning-vertical-horizontal-hybrid-partitioning/

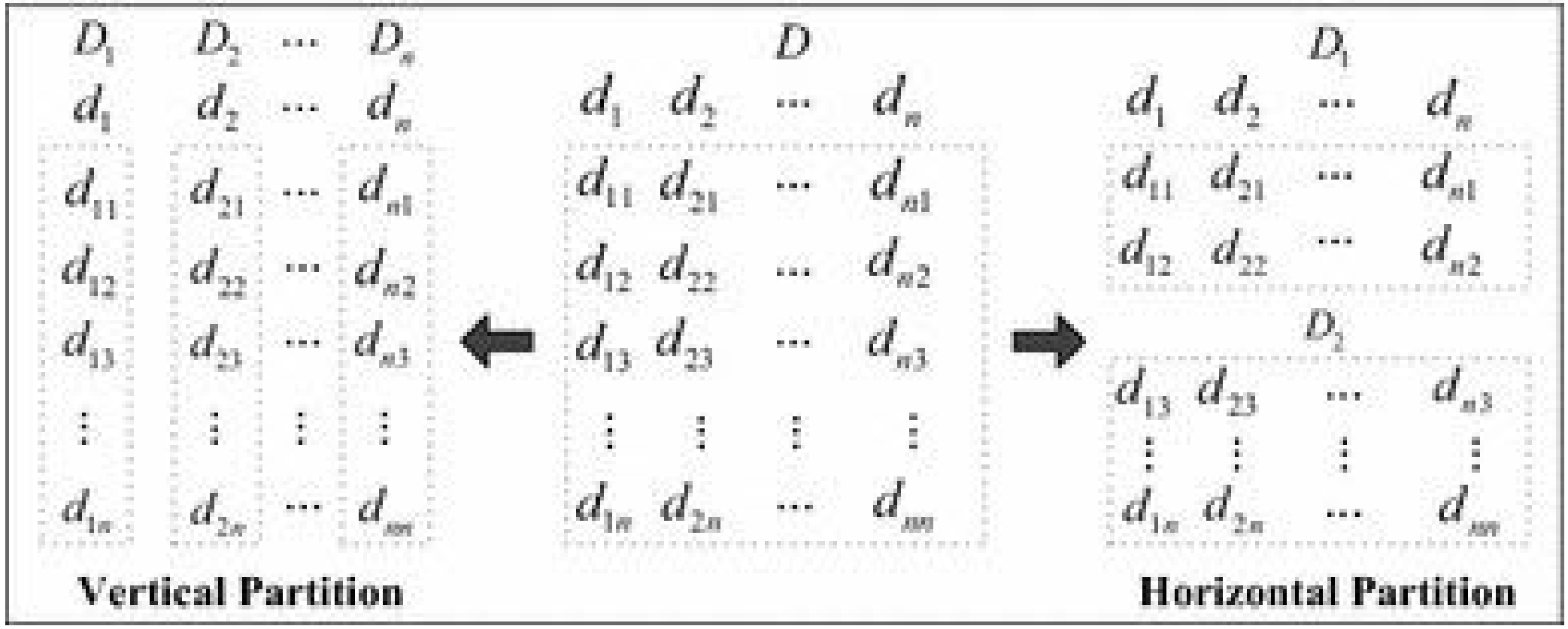
Vertical vs. Horizontal partitioning





Source: http://www.piyushgupta.co.uk/2016/04/database-scaling-jargons.html, http://slideplayer.com/slide/12131436/70/images/17/SQL+Azure+Azure+Custom+Sharding.jpg

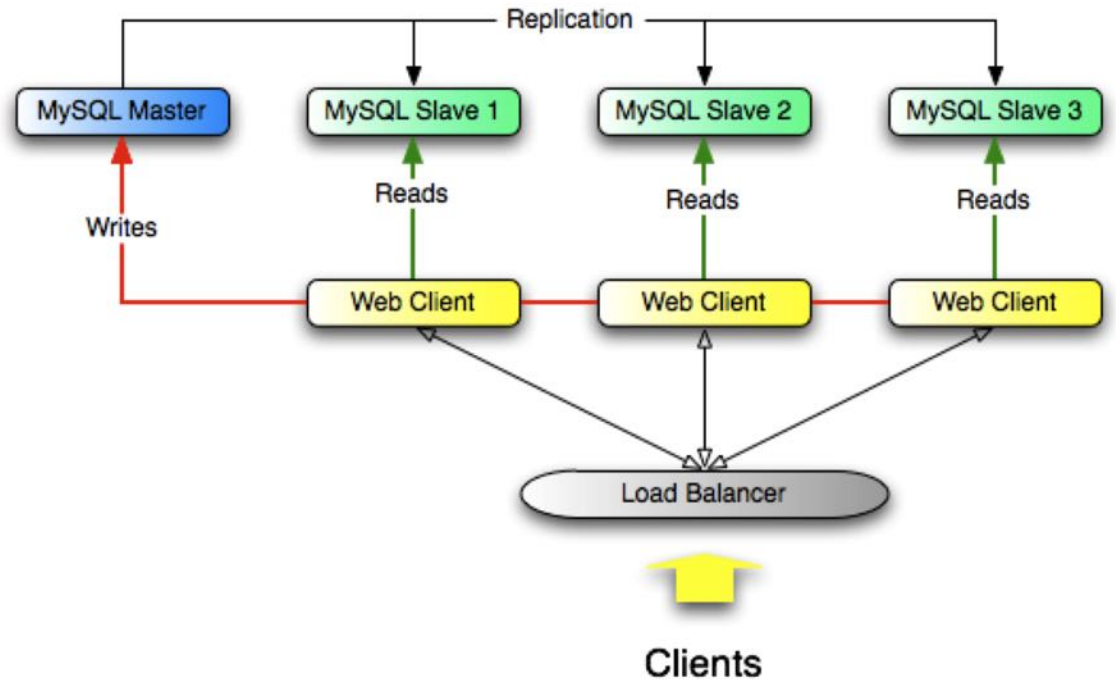
Cp. to concepts in Linear Algebra



Source: "An efficient scheme for probabilistic skyline queries over distributed uncertain data," Xiaoyong Li, Yijie Wang, Jie Yu, Telecommunication systems 2015.

Database Replication

- Data replication: storing the same data on several machines (“nodes”)
- Useful for:
  - Availability (parallel requests are made against replicas)
  - Reliability (data can survive hardware faults)
  - Fault tolerance (system stays alive when nodes/network fail)
- Typical architecture: master-slave



Replication example in MySQL (dev.mysql.com)

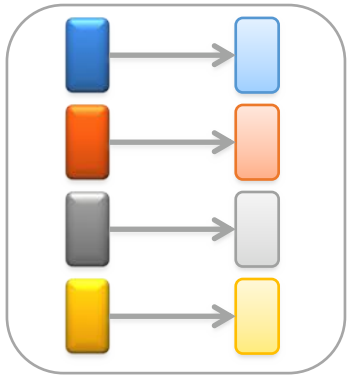
Open Source

- Free software, source provided
  - Users have the right to use, modify and distribute the software
  - But restrictions may still apply, e.g., adaptations need to be opensource
- Idea: community development
  - Developers fix bugs, add features, ...
- How can that work?
  - See [Bonaccorsi, Rossi, 2003. Why open source software can succeed. Research policy, 32(7), pp.1243-1258]
- A major driver of OpenSource is Apache

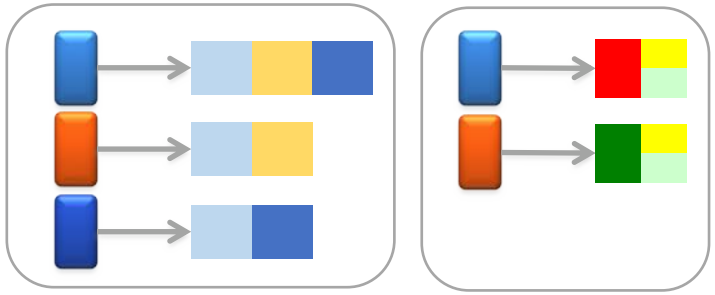
Apache Software Foundation

- Non-profit organization
- Hosts communities of developers
  - Individuals and small/large companies
- Produces open-source software
- Funding from grants and contributions
- Hosts very significant projects
  - Apache Web Server, Hadoop, Zookeeper, Cassandra, Lucene, OpenOffice, Struts, Tomcat, Subversion, Tcl, UIMA, ...

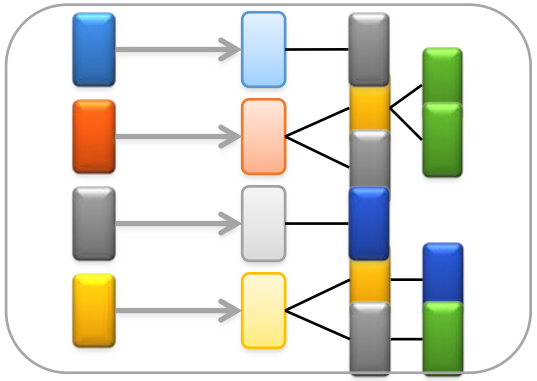
We Will Look at 4 Data Models



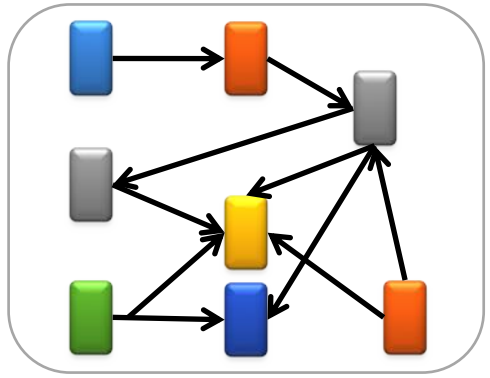
Key/Value Store
(e.g. REDIS)



Column-Family Store
(e.g. Cassandra)



Document Store
(e.g. MongoDB)



Graph Databases
(e.g. Neo4J

Source: Benny Kimelfeld

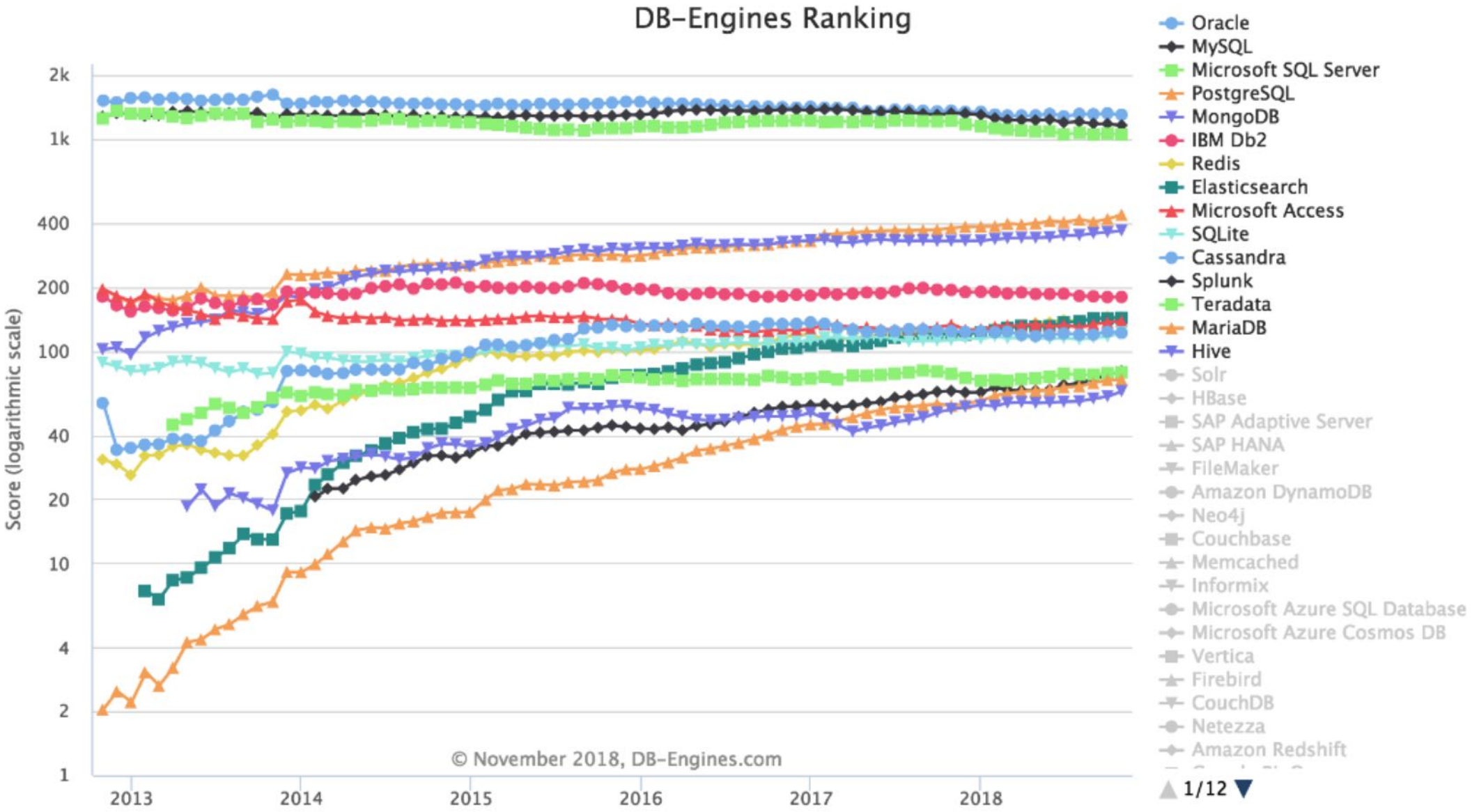
Database engines ranking by "popularity"

348 systems in ranking, November 2018

Rank			DBMS	Database Model	Score		
Nov 2018	Oct 2018	Nov 2017			Nov 2018	Oct 2018	Nov 2017
1.	1.	1.	Oracle	Relational DBMS	1301.11	-18.16	-58.94
2.	2.	2.	MySQL	Relational DBMS	1159.89	-18.22	-162.14
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1051.55	-6.78	-163.53
4.	4.	4.	PostgreSQL	Relational DBMS	440.24	+20.85	+60.33
5.	5.	5.	MongoDB	Document store	369.48	+6.30	+39.01
6.	6.	6.	IBM Db2	Relational DBMS	179.87	+0.19	-14.19
7.	7.	↑ 9.	Redis	Key-value store	144.17	-1.12	+22.99
8.	8.	↑ 10.	Elasticsearch	Search engine	143.46	+1.13	+24.05
9.	9.	↓ 7.	Microsoft Access	Relational DBMS	138.44	+1.64	+5.12
10.	↑ 11.	↑ 11.	SQLite	Relational DBMS	122.71	+5.96	+9.95
11.	↓ 10.	↓ 8.	Cassandra	Wide column store	121.74	-1.64	-2.47

Source: https://db-engines.com/en/ranking , 11/2018

Database engines ranking by "popularity"



Source: https://db-engines.com/en/ranking_trend , 11/2018

Highlighted Database Features

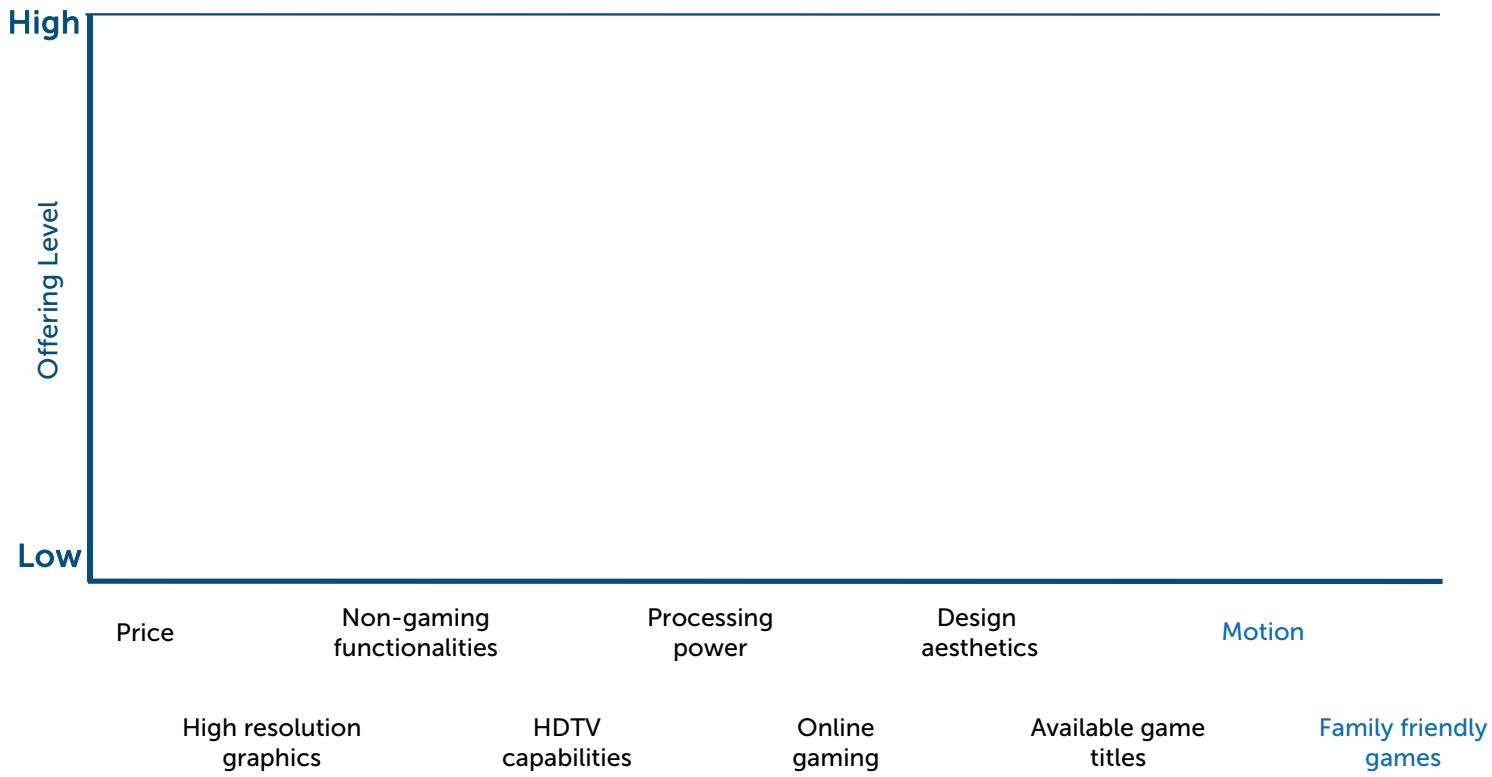
- Data model
  - What data is being stored?
- CRUD interface
  - API for Create, Read, Update, Delete
    - 4 basic functions of persistent storage (insert, select, update, delete)
  - Sometimes preceding S for Search
- Transaction consistency guarantees
- Replication and sharding model
  - What's automated and what's manual?

True and False Conceptions

- True:
  - SQL does not effectively handle common Web needs of massive (datacenter) data
  - SQL has guarantees that can sometimes be compromised for the sake of scaling
  - Joins are not for free, sometimes undoable
- False:
  - NoSQL says NO to SQL
  - Nowadays NoSQL is the only way to go
  - Joins can always be avoided by structure redesign

Strategy Canvas: Example Nintendo Wii (1/3)

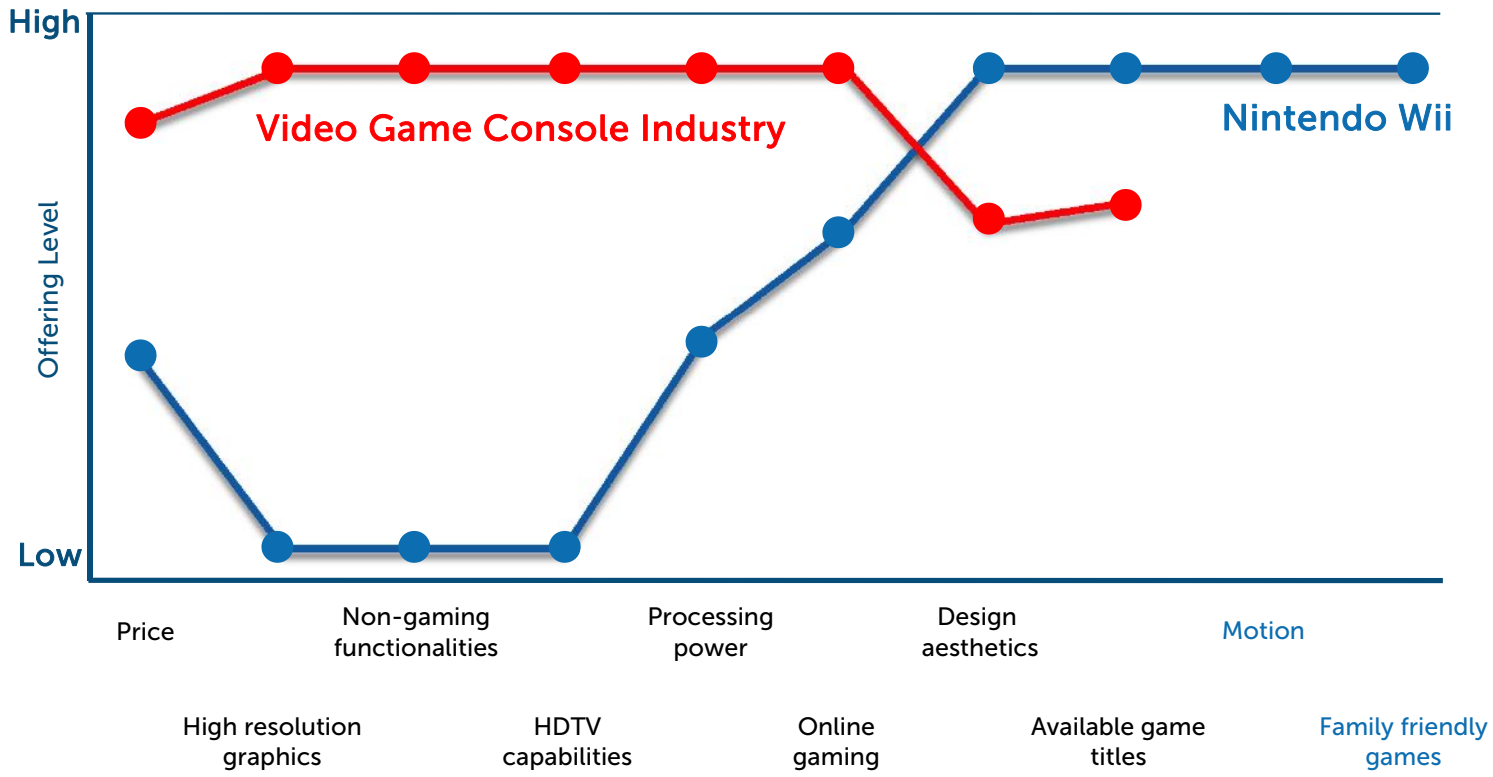
Nintendo Wii Strategy Canvas



Source: INSEAD, Blue Ocean Strategy Institute, 2013.

Strategy Canvas: Example Nintendo Wii (2/3)

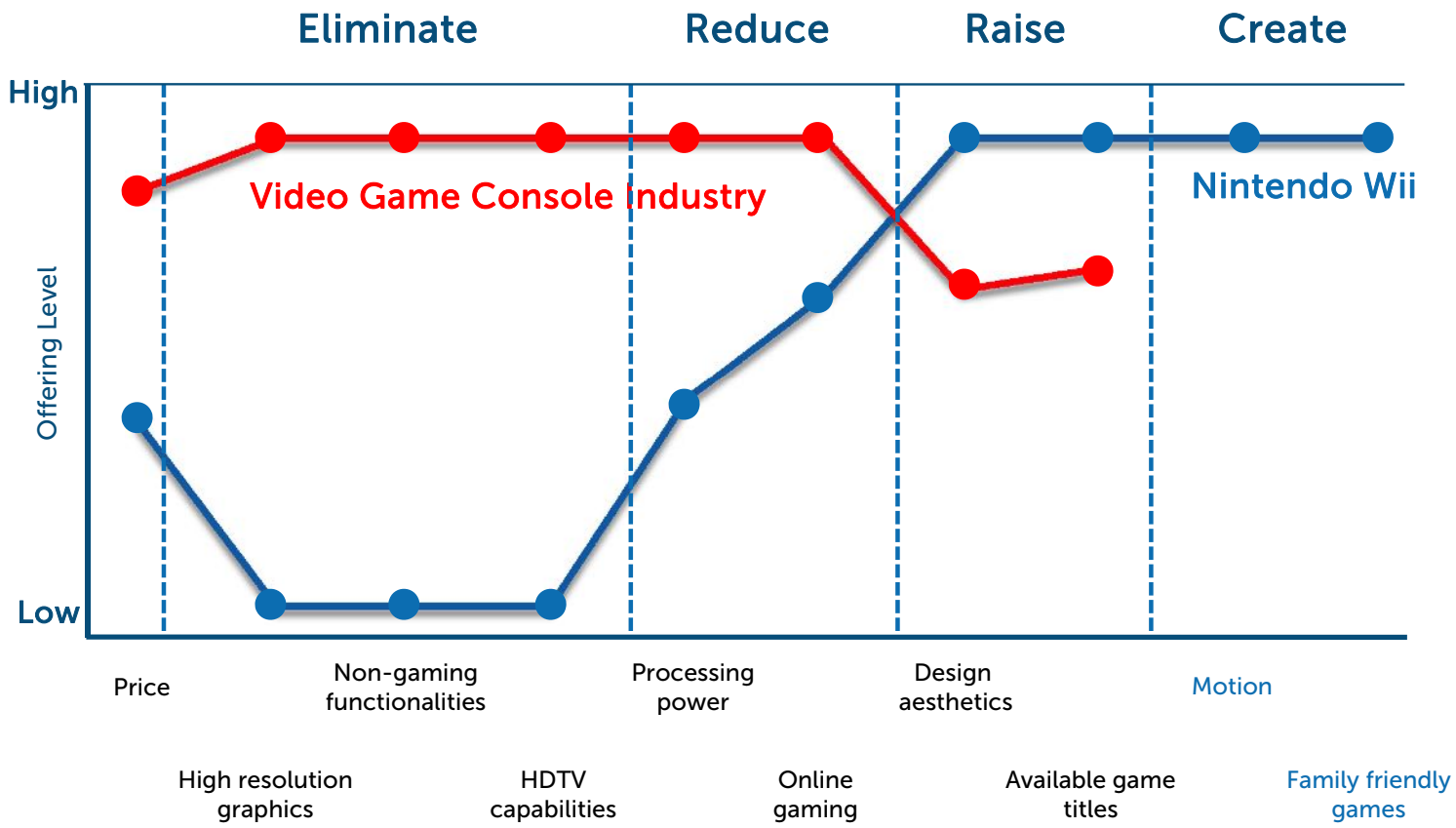
Nintendo Wii Strategy Canvas



Source: INSEAD, Blue Ocean Strategy Institute, 2013.

Strategy Canvas: Example Nintendo Wii (3/3)

Nintendo Wii Strategy Canvas



Source: INSEAD, Blue Ocean Strategy Institute, 2013.

Redefine the Market

SIDE TOPIC

Outline

- Introduction
- Transaction Consistency
- 4 main data models
  - Key-Value Stores (e.g., Redis)
  - Column-Family Stores (e.g., Cassandra)
  - Document Stores (e.g., MongoDB)
  - Graph Databases (e.g., Neo4j)
- Concluding Remarks

Transaction

- A sequence of operations (over data) viewed as a single higher-level operation
  - Transfer money from account 1 to account 2
- DBMSs execute transactions in parallel
  - No problem applying two “disjoint” transactions
  - But what if there are dependencies (conflicts)?
- Transactions can either commit (succeed) or abort (fail)
  - Failure due to violation of program logic, network failures, credit-card rejection, etc.
- DBMS should not expect transactions to succeed

Examples of Transactions

- Airline ticketing
  - Verify that the seat is vacant, with the price quoted, then charge credit card, then reserve
- Textbook example: bank money transfer
  - Read from acct#1, verify funds, update acct#1, update acct#2
- Online purchasing
  - Similar
- “Transactional file systems” (MS NTFS)
  - Moving a file from one directory to another: verify file exists, copy, delete

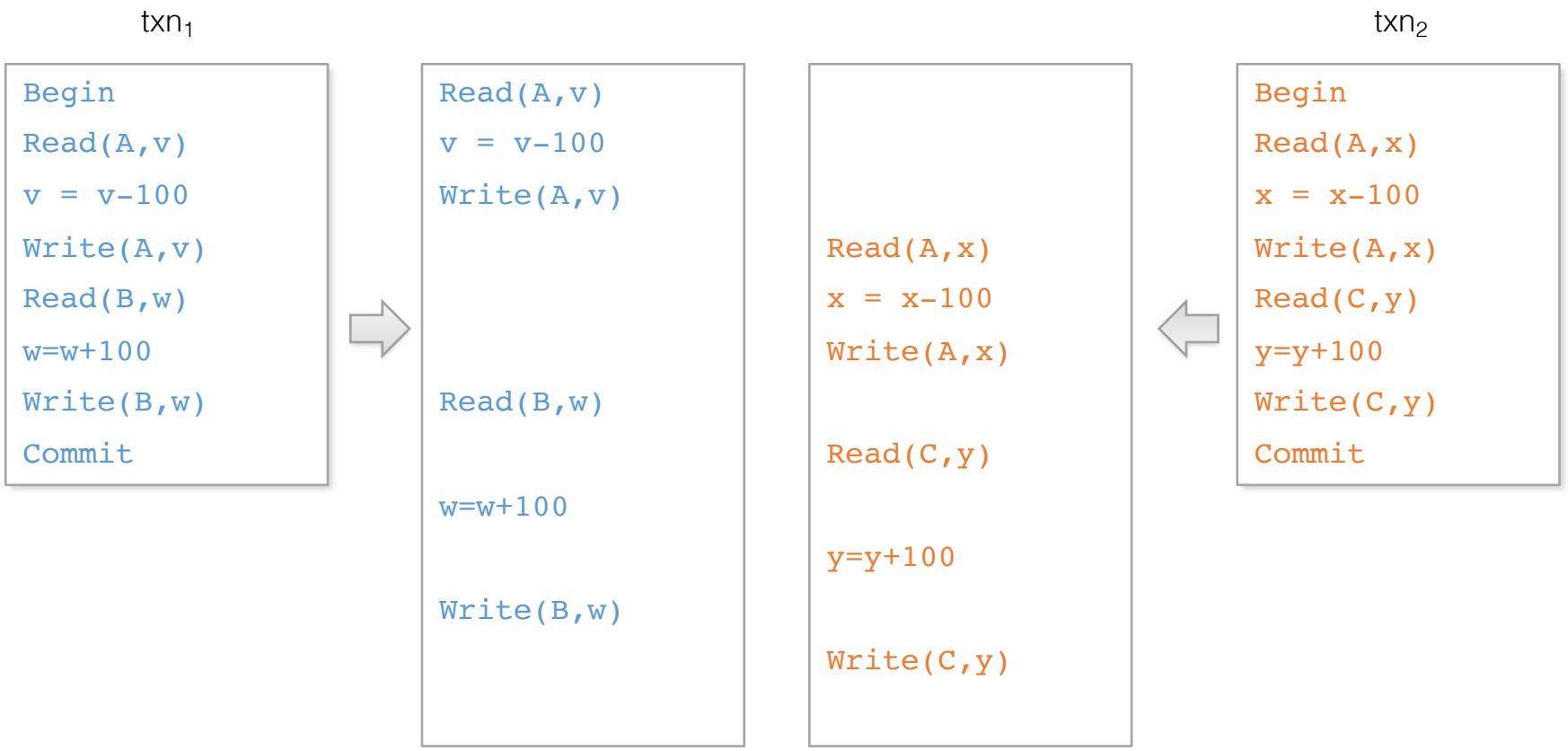
Transfer Example

```
Begin
Read(A,v)
v = v-100
Write(A,v)
Read(B,w)
w=w+100
Write(B,w)
Commit
```

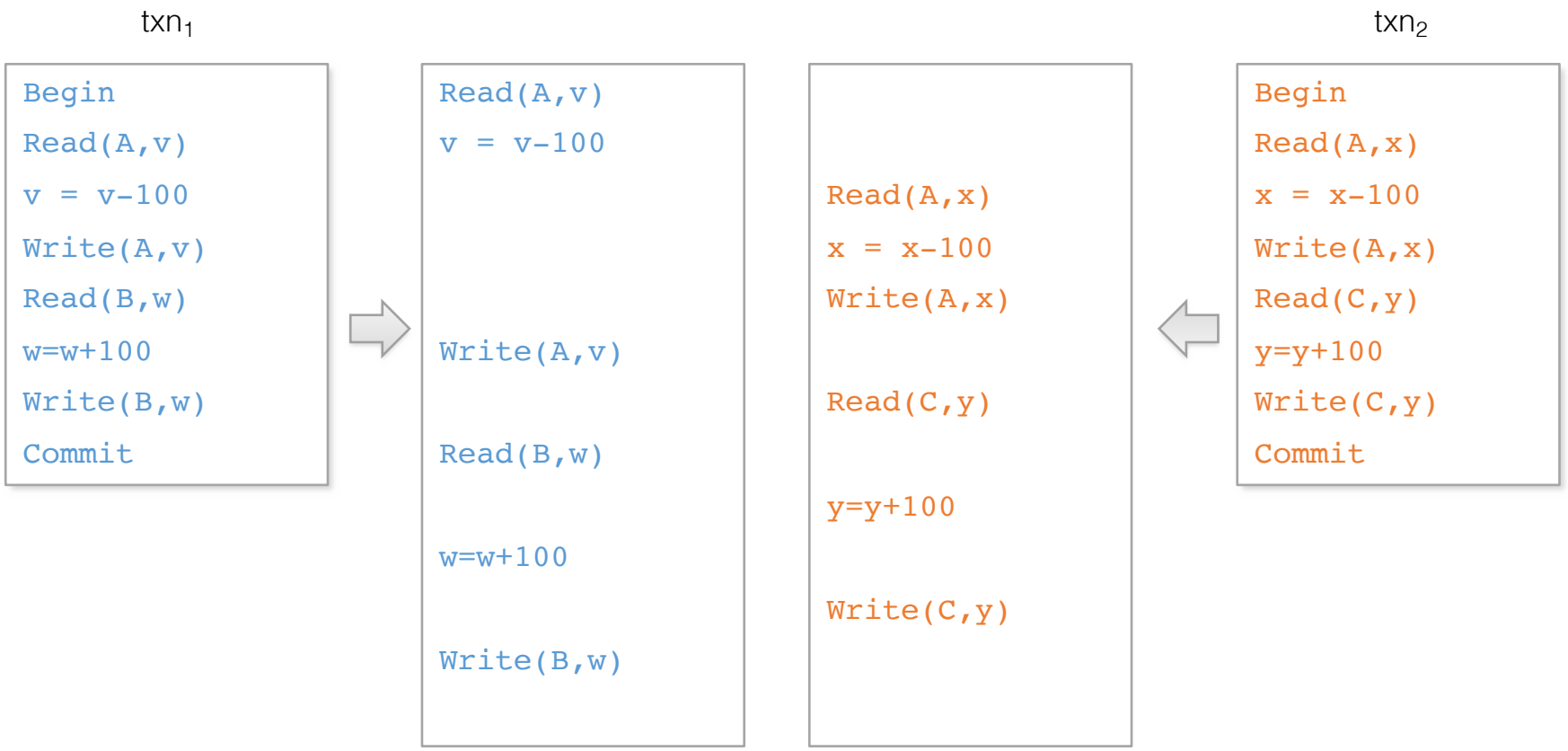
txn_1	txn_2
Begin	Begin
Read(A,v)	Read(A,x)
v = v-100	x = x-100
Write(A,v)	Write(A,x)
Read(B,w)	Read(C,y)
w=w+100	y=y+100
Write(B,w)	Write(C,y)
Commit	Commit

- *Scheduling* is the operation of interleaving transactions
  - *Why is it good?*
- A *serial schedule* executes transactions one at a time, from beginning to end
- A *good* ("serializable") scheduling is one that *behaves like some serial scheduling* (typically by locking protocols)

Scheduling Example 1



Scheduling Example 2



ACID

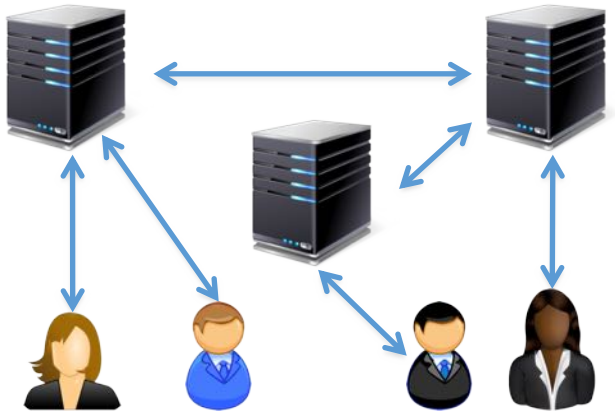
- **A**tomicity
  - Either all operations applied or none are (hence, we need not worry about the effect of incomplete / failed transactions)
- **C**onsistency
  - Each transaction can start with a consistent database and is required to leave the database consistent (bring the DB from one to another consistent state)
- **I**solation
  - The effect of a transaction should be as if it is the only transaction in execution (in particular, changes made by other transactions are not visible until committed)
- **D**urability
  - Once the system informs a transaction success, the effect should hold without regret, even if the database crashes (before making all changes to disk)

ACID May Be Overly Expensive

- In quite a few modern applications:
  - ACID contrasts with key desiderata: high volume, high availability
  - We can live with some errors, to some extent
  - Or more accurately, we prefer to suffer errors than to be significantly less functional
- *Can this point be made more "formal"?*

Simple Model of a Distributed Service

- Context: distributed service
  - e.g., social network
- Clients make get / set requests
  - e.g., setLike(user,post), getLikes(post)
  - Each client can talk to any server
- Servers return responses
  - e.g., ack, {$user_1$,....,$user_k$}
- Failure: the network may occasionally disconnect due to failures (e.g., switch down)
- Desiderata: **C**onsistency, **A**vailability, **P**artition tolerance

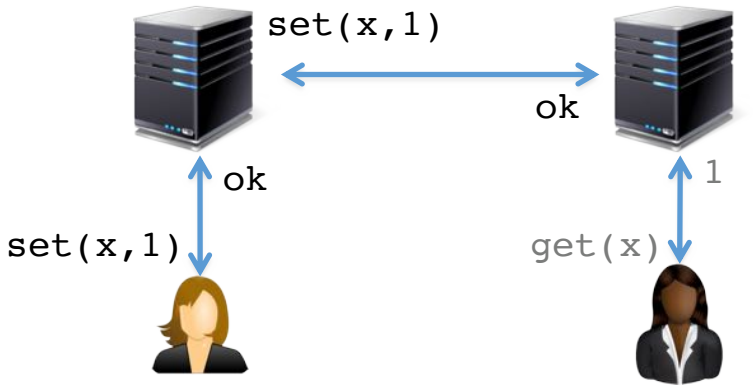


CAP Service Properties

- **C**onsistency:
  - every read (to any node) gets a response that reflects the most recent version of the data
    - More accurately, a transaction should behave as if it changes the entire state correctly in an instant, Idea similar to serializability
- **A**vailability:
  - every request (to a living node) gets an answer: set succeeds, get retunes a value (if you can talk to a node in the cluster, it can read and write data)
- **P**artition tolerance:
  - service continues to function on network failures (cluster can survive
    - As long as clients can reach servers

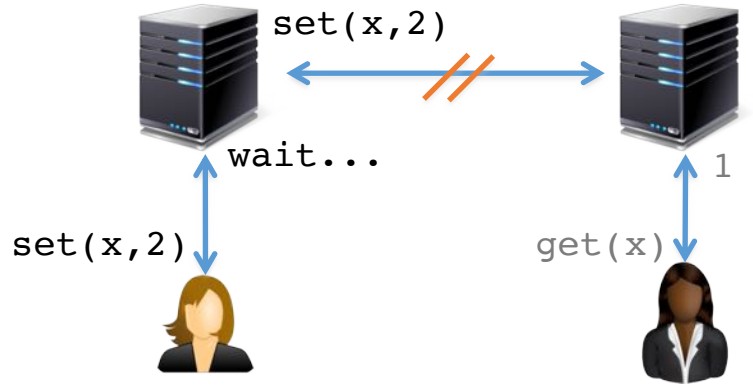
Simple Illustration

Our Relational Database world so far …



CA

Consistency, Availability

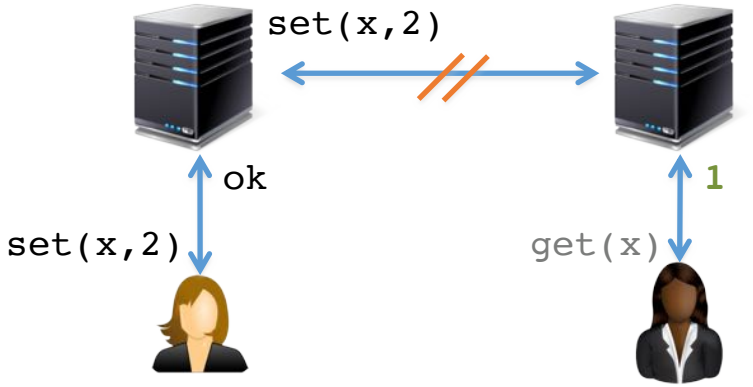


In a system that may suffer partitions, you have to trade off consistency vs. availability

CP

Consistency, Partition tolerance

~~Availability~~



AP

Availability, Partition tolerance

~~Consistency~~

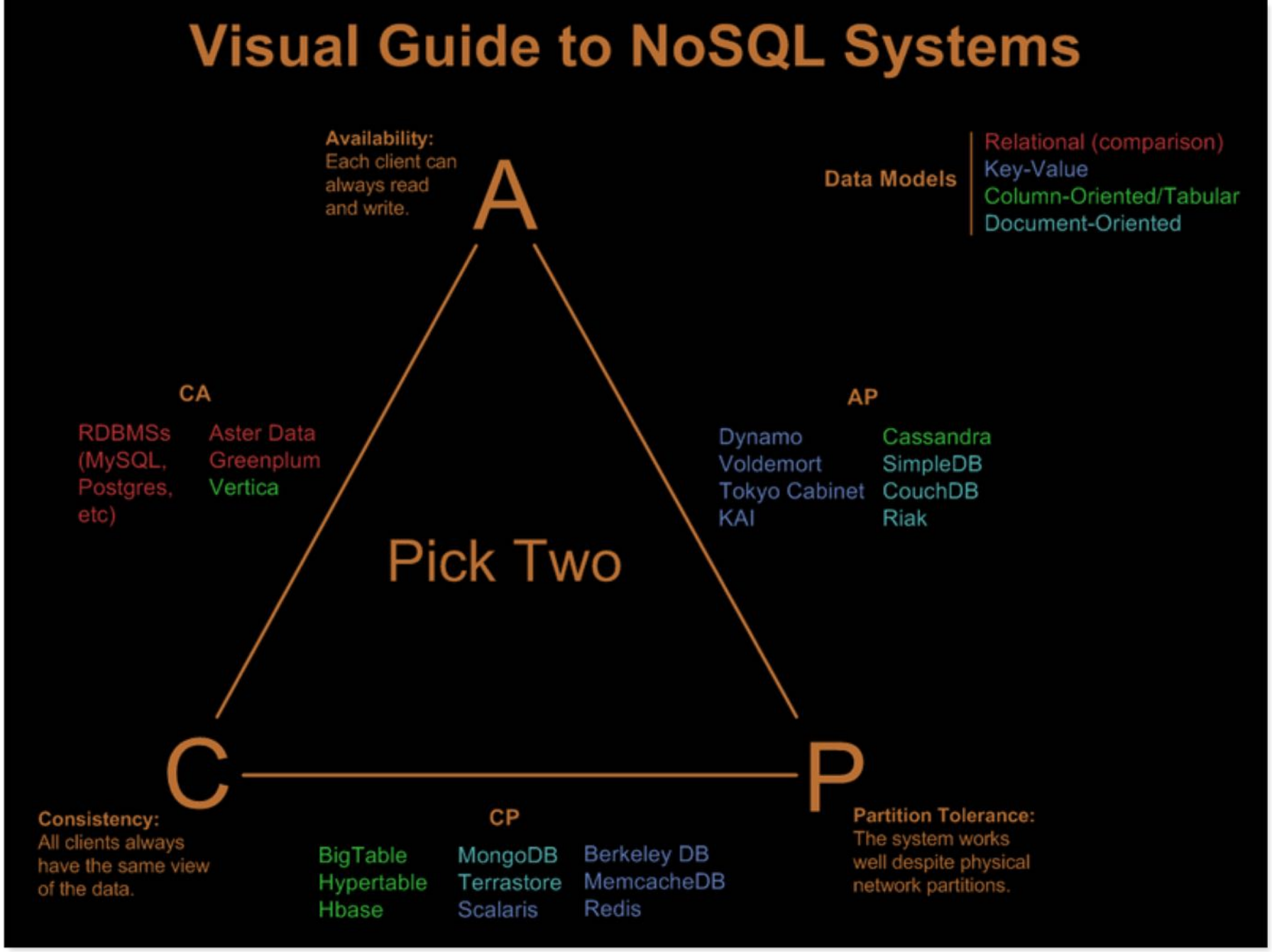
The CAP Theorem

Eric Brewer's CAP Theorem:

A distributed service
can support at most two
out of **C**, **A** *and* **P**

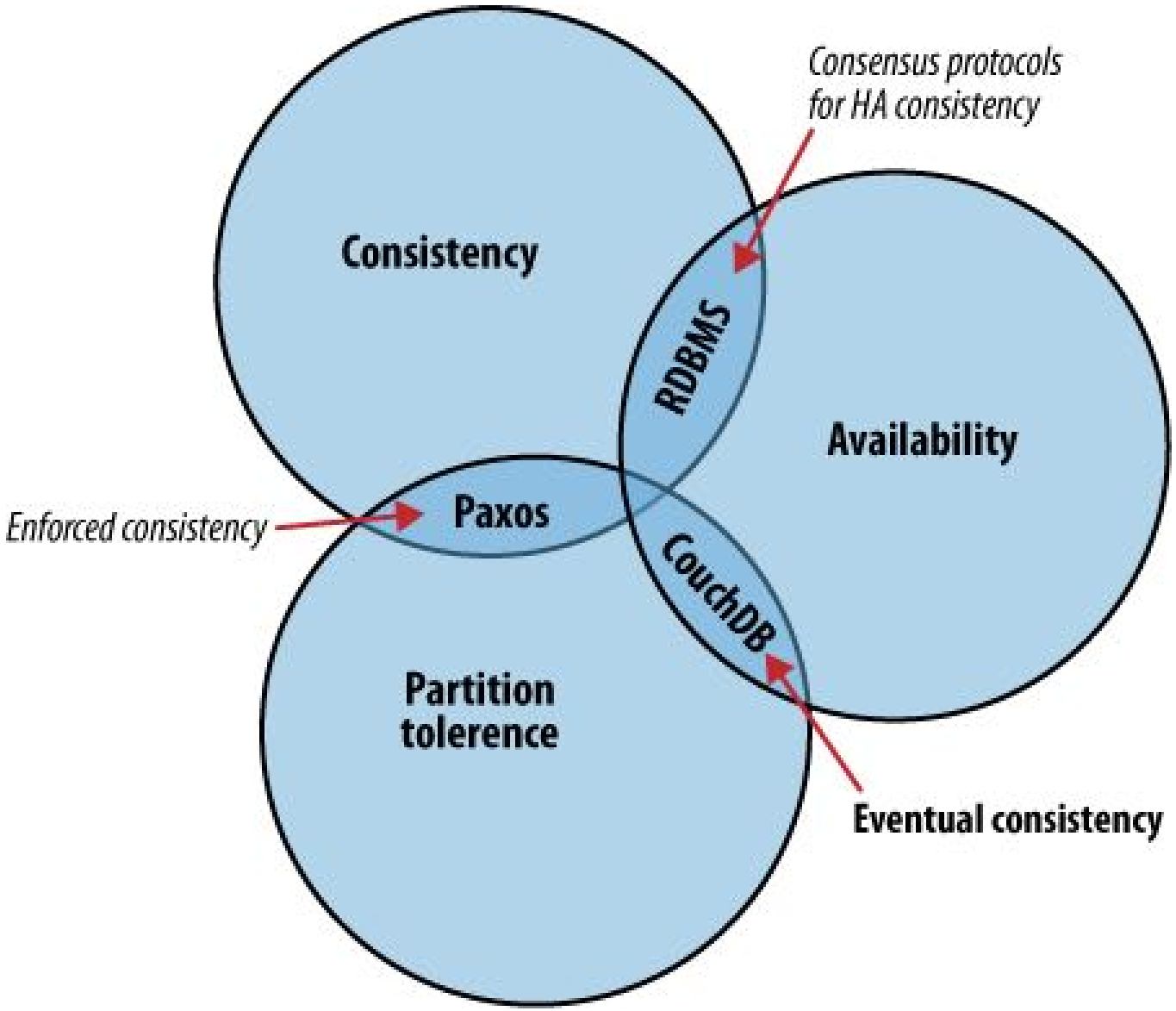
Historical Note

- Brewer presented it as the CAP principle in a 1999 article
  - Then as an informal conjecture in his keynote at the PODC 2000 conference
- In 2002 a formal proof was given by Gilbert and Lynch, making CAP a theorem
  - [Seth Gilbert, Nancy A. Lynch: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News 33(2): 51-59 (2002)]
  - It is mainly about making the statement formal; the proof is straightforward



Source: http://blog.nahurst.com/visual-guide-to-nosql-systems , 2010

CAP theorem



Source: http://guide.couchdb.org

The BASE Model

- Applies to distributed systems of type AP
- **B**asic **A**vailability
  - Provide high availability through distribution: There will be a response to any request. Response could be a ‘failure’ to obtain the requested data, or the data may be in an inconsistent or changing state.
- **S**oft state
  - Inconsistency (stale answers) allowed: State of the system can change over time, so even during times without input, changes can happen due to ‘eventual consistency’
- **E**ventual consistency
  - If updates stop, then after some time consistency will be achieved
    - Achieved by protocols to propagate updates and verify correctness of propagation (gossip protocols)
- Philosophy: best effort, optimistic, staleness and approximation allowed

Outline

- Introduction
- Transaction Consistency
- 4 main data models
  - Key-Value Stores (e.g., Redis)
  - Column-Family Stores (e.g., Cassandra)
  - Document Stores (e.g., MongoDB)
  - Graph Databases (e.g., Neo4j)
- Concluding Remarks

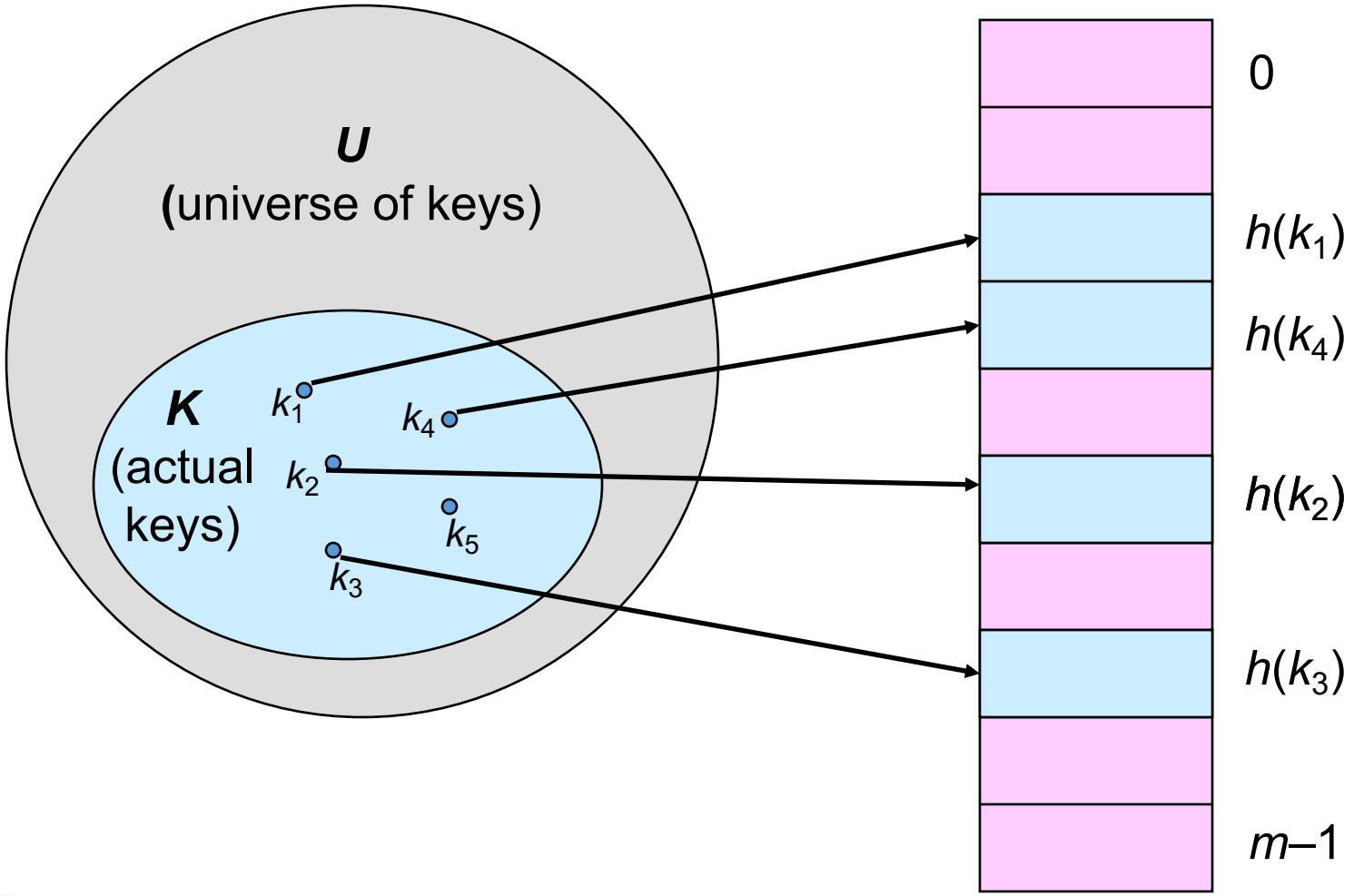
Key-Value Stores

- Essentially, big distributed hash maps
- Origin attributed to Dynamo – Amazon's DB for world-scale catalog/cart collections
  - But Berkeley DB has been here for >20 years
- Store pairs ⟨key, opaque-value⟩
  - Opaque means that DB does not associate any structure/semantics with the value; oblivious to values
  - This may mean more work for the user: retrieving a large value and parsing to extract an item of interest
- Sharding via partitioning of the key space
  - Hashing, gossip and remapping protocols for load balancing and fault tolerance

Hashing (Hash tables, dictionaries)

$h : U \to \{0,1,\ldots, m-1\}$

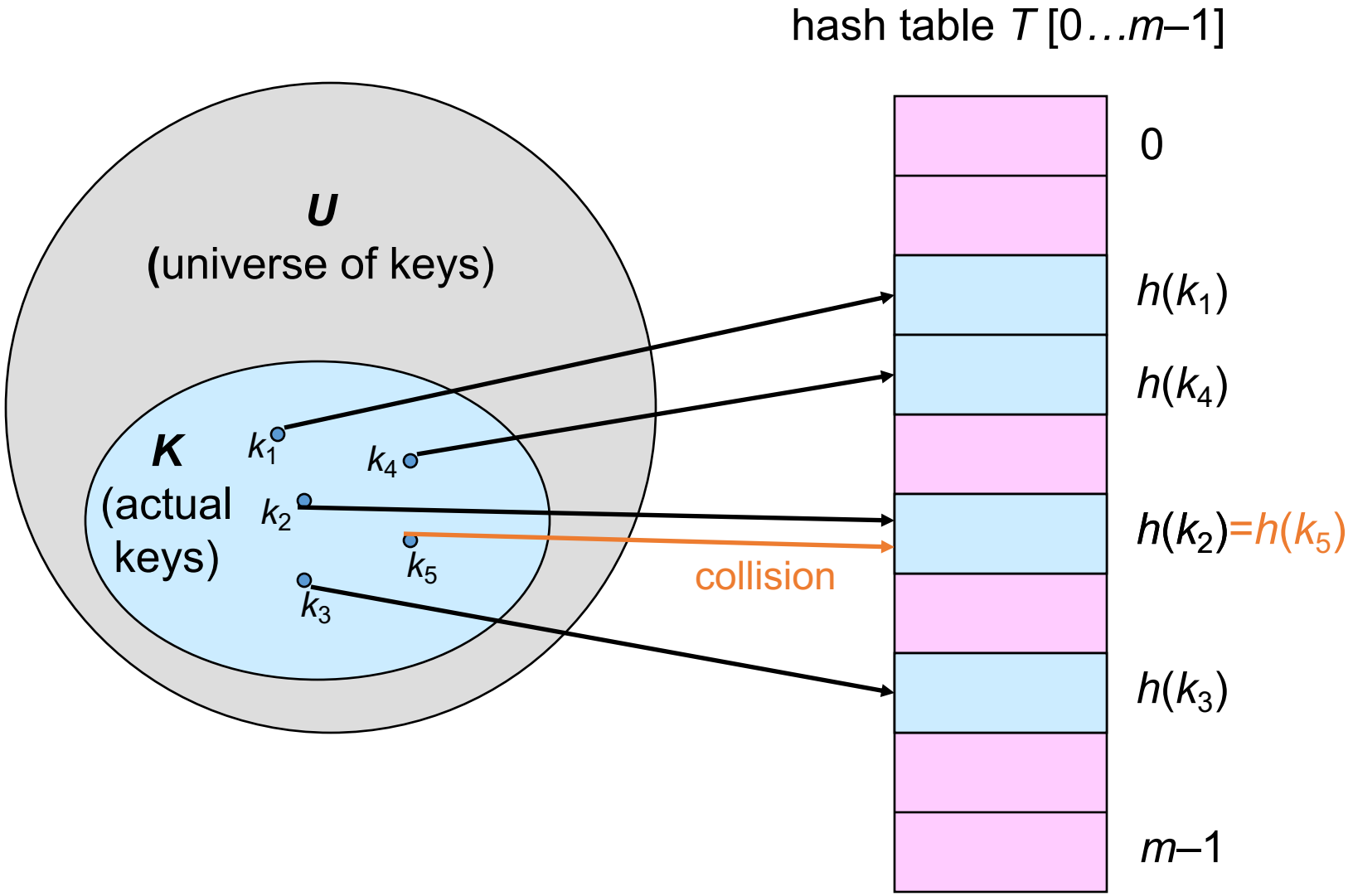
hash table $T\,[0\ldots m-1]$



$n = |K| << |U|$.

key k "hashes" to slot $T[h[k]]$

Hashing (Hash tables, dictionaries)



Example Databases

- Amazon's DynamoDB
  - Originally designed for Amazon's workload at peaks
  - Offered as part of Amazon's Web services
- Redis
  - Next slides and in our Jupyter notebooks
- Riak
  - Focuses on high availability, BASE
  - "As long as your Riak client can reach one Riak server, it should be able to write data."
- FoundationDB
  - Focus on transactions, ACID
- Berkeley DB (and Oracle NoSQL Database)
  - First release 1994, by Berkeley, acquired by Oracle
  - ACID, replication

Redis

- Basically a data structure for strings, numbers, hashes, lists, sets
- Simplistic “transaction” management
  - Queuing of commands as blocks, really
  - Among ACID, only Isolation guaranteed
    - A block of commands that is executed sequentially; no transaction interleaving; no roll back on errors
- In-memory store
  - Persistence by periodical saves to disk
- Comes with
  - A command-line API
  - Clients for different programming languages
    - Perl, PHP, Rubi, Tcl, C, C++, C#, Java, R, …

Example of Redis Commands

key maps to:


```
get x
>> 10
```

```
hget h y
>> 5
```

```
hkeys p:22
>> name , age
```

```
smembers s
>> 20 , Alice
```

```
scard s
>> 2
```

```
llen l
>> 3
```

```
lrange l 1 2
>> a , b
```

```
lindex l 2
>> b
```

```
lpop l
>> c
```

```
rpop l
>> b
```

Example of Redis Commands

key maps to:

(simple value)

(hash table)

(set)

(list)

```
set x 10
hset h y 5
hset h1 name two
hset h1 value 2
hmset p:22 name Alice age 25
sadd s 20
sadd s Alice
sadd s Alice
rpush l a
rpush l b
lpush l c
```

key	value
x	10
h	y→5
h1	name→two value→2
p:22	name→Alice age→25
s	{20,Alice}
l	(c,a,b)

```
get x
>> 10
```

```
hget h y
>> 5
```

```
hkeys p:22
>> name , age
```

```
smembers s
>> 20 , Alice
```

```
scard s
>> 2
```

```
llen l
>> 3
```

```
lrange l 1 2
>> a , b
```

```
lindex l 2
>> b
```

```
lpop l
>> c
```

```
rpop l
>> b
```