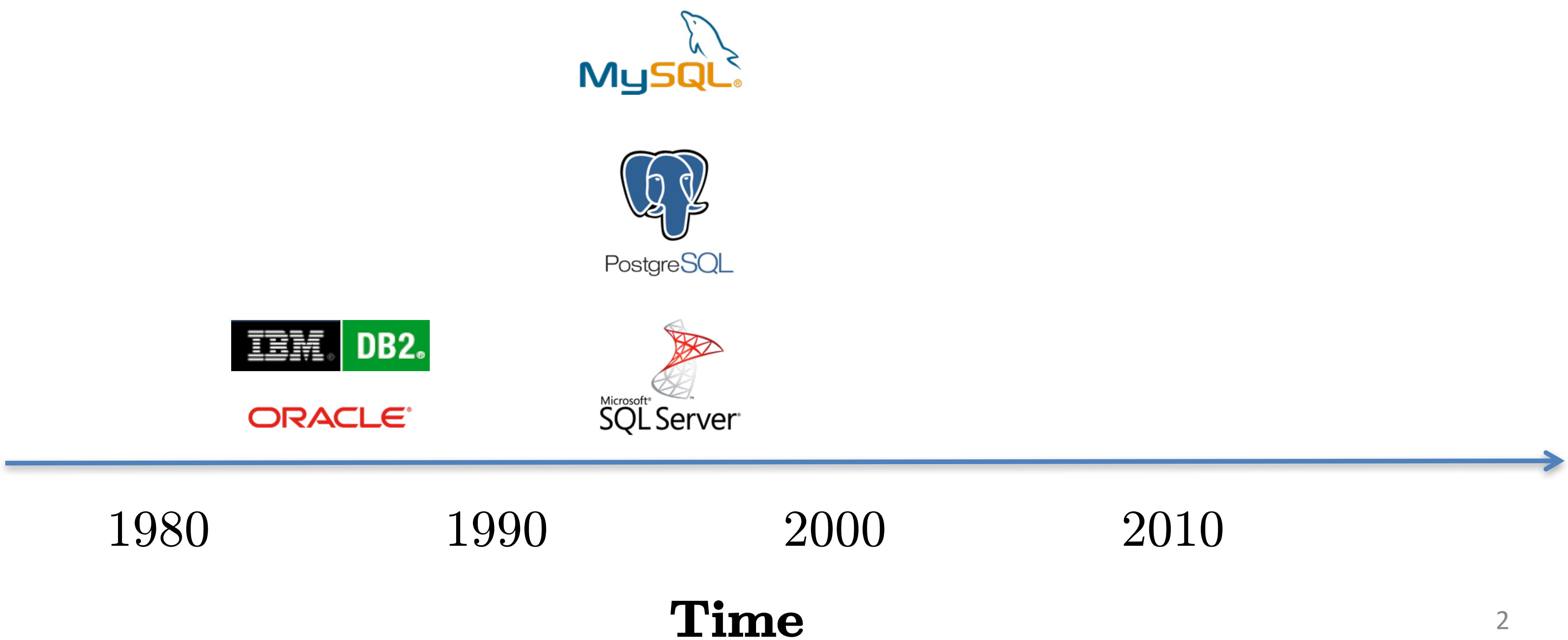
Demystifying the Zoo of Contemporary Database Systems

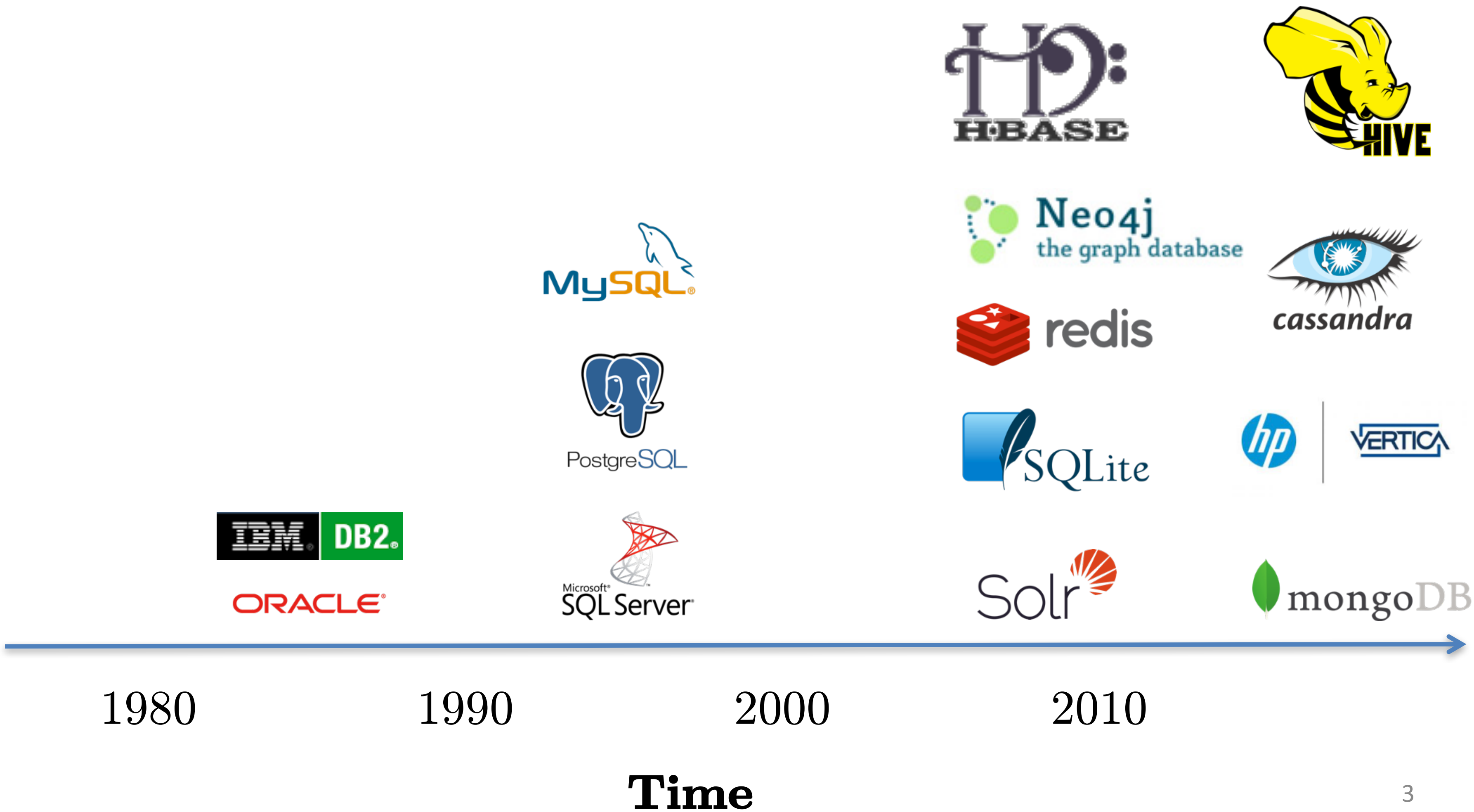
CS165 Section

Niv Dayan

Introduction



Introduction

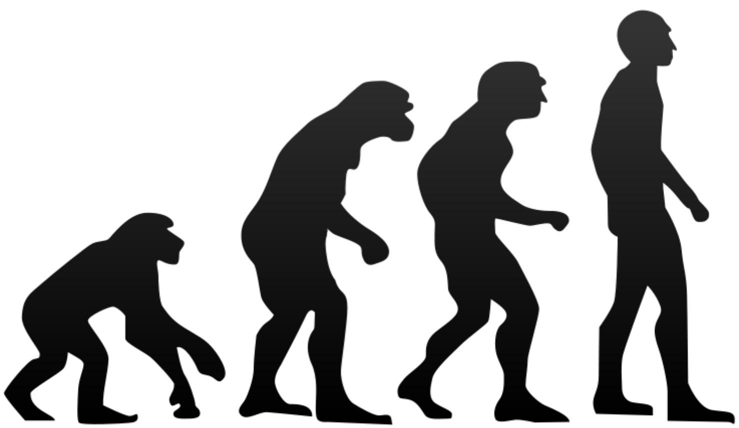


Introduction

- Different architectures
  - Performance
  - Data integrity
  - User interface

Introduction

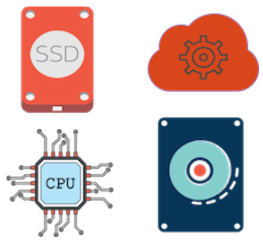
- Different architectures
  - Performance
  - Data integrity
  - User interface



Introduction

- **Theme:** any trend in database technology can be traced to a trend in hardware

Database designer



Hardware

- **Claim:** The new database technologies are adaptations to changes in hardware

DBHistory

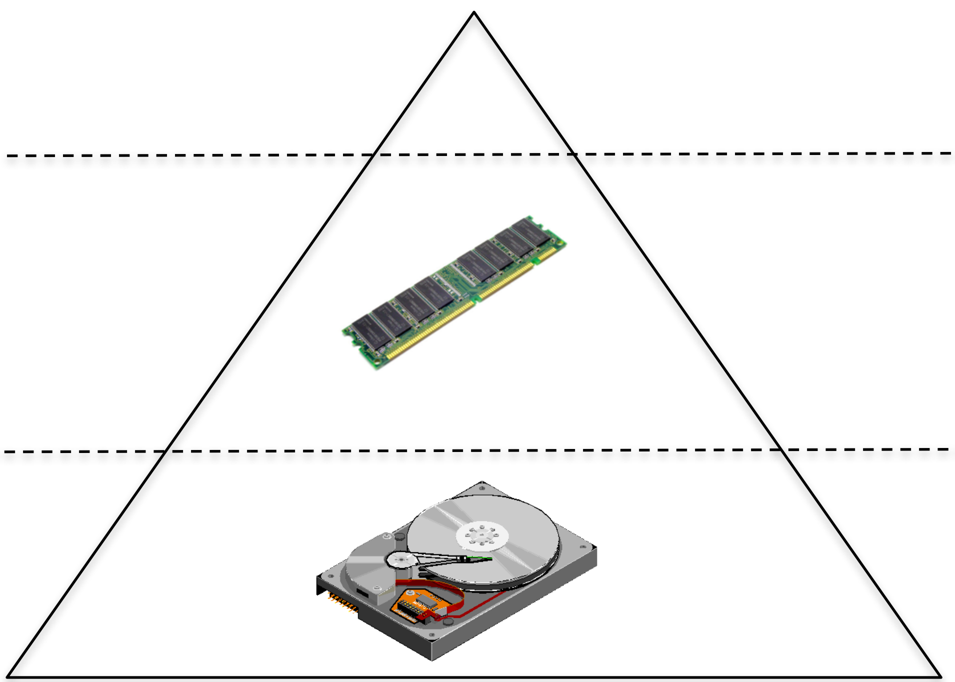
History

- 3 goals of database design
  - Speed
  - Affordability
  - Resilience to system failure

- How you achieve them depends on hardware

History

- Two storage media:

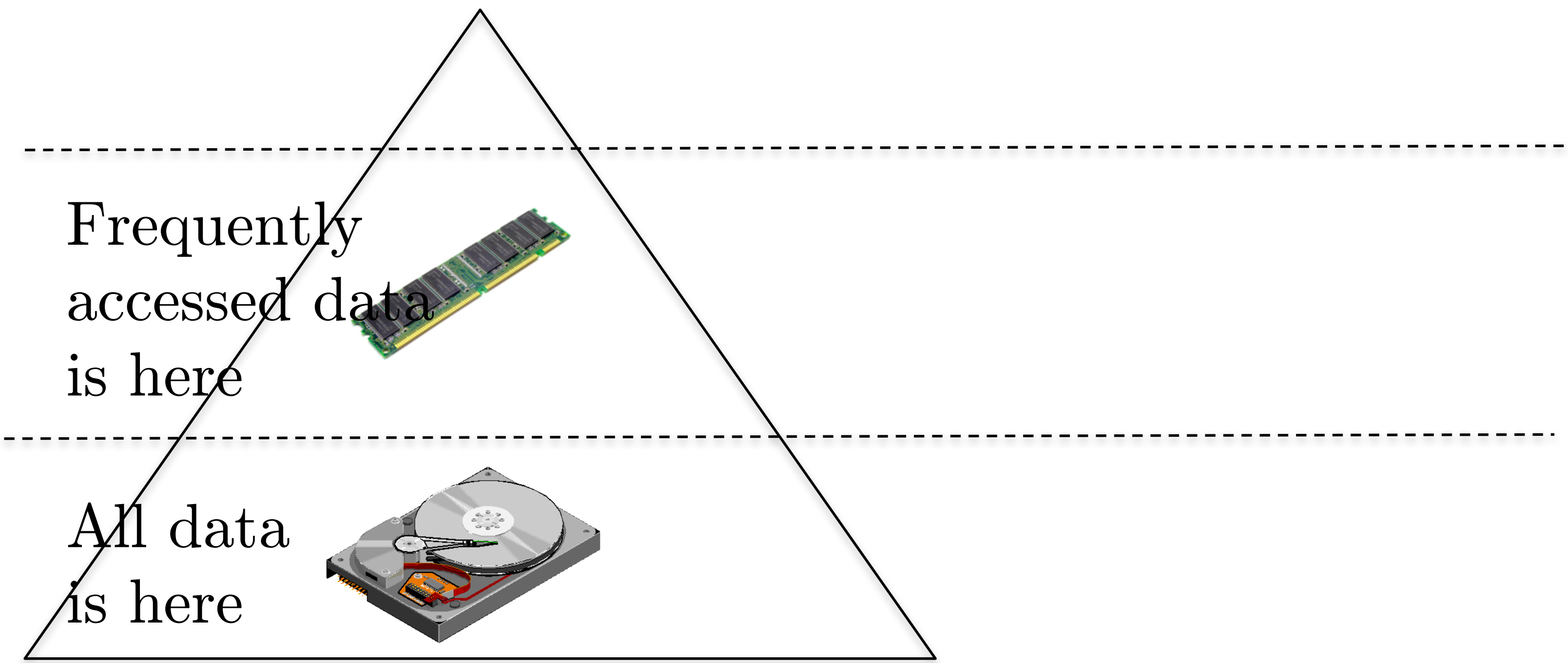


Main Memory
Fast, expensive, volatile

Disk
Slow, cheap, non-volatile

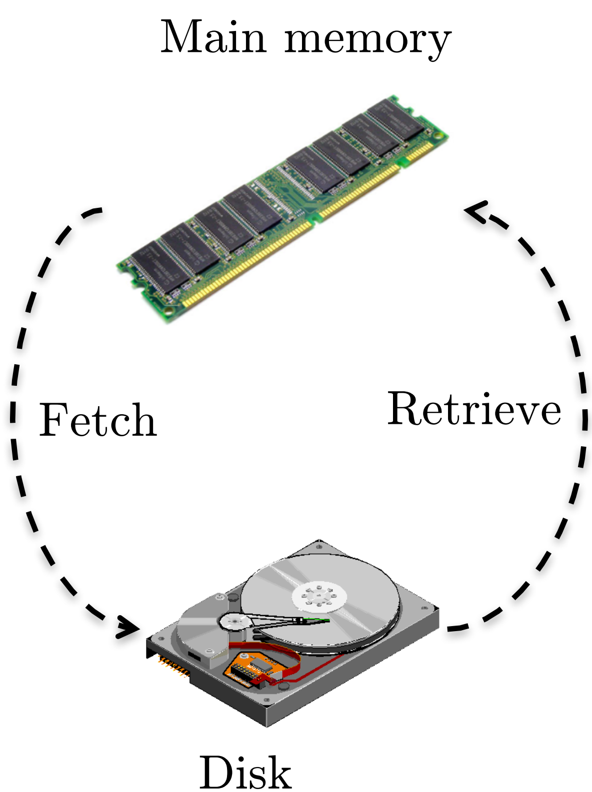
History

- How should data be stored across them?
- Main memory is volatile and expensive



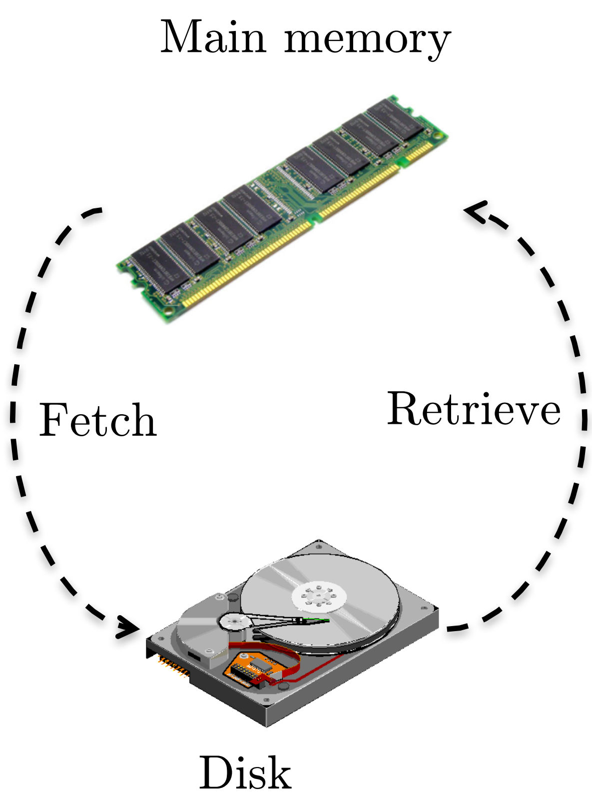
History

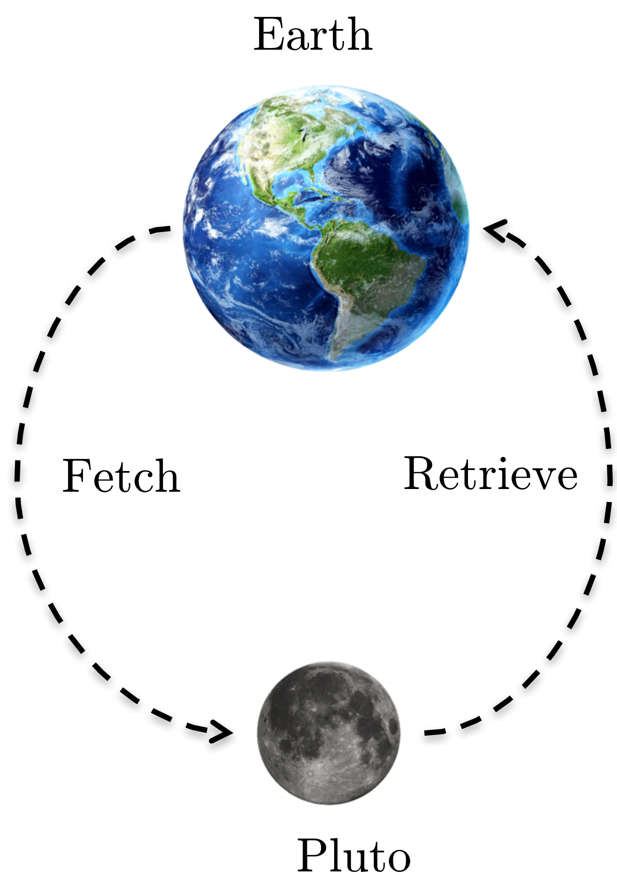
- To make a system fast, address bottleneck
- Disk is extremely slow



History

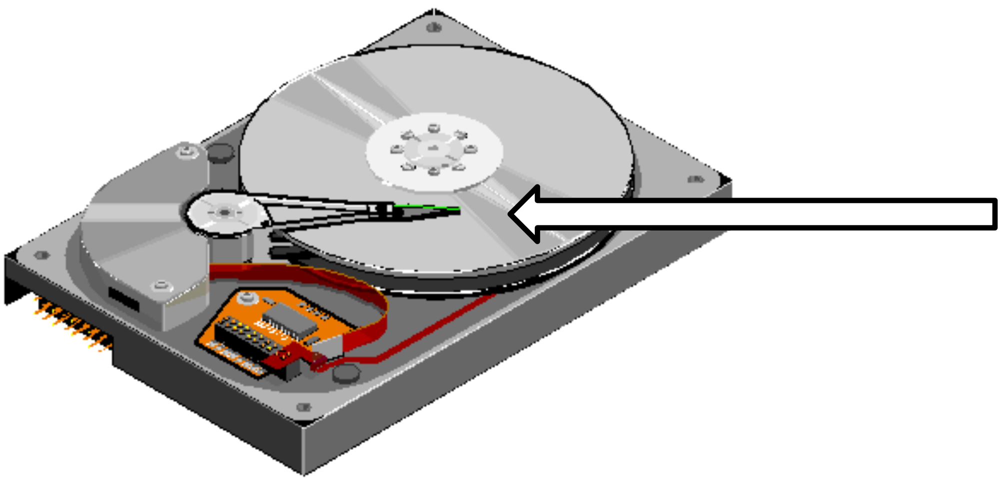
- To make a system fast, address bottleneck
- Disk is extremely slow





History

- Why so slow?

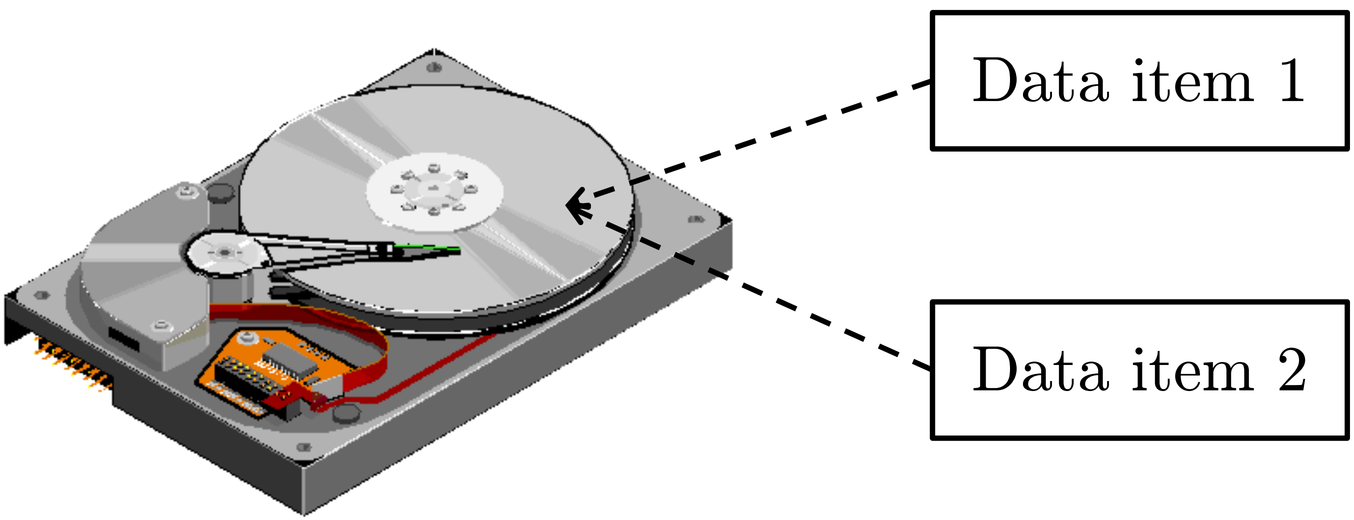


Disk hand moving

- Two questions:
  - Question 1: How to minimize disk access?
  - Question 2: What to do during a disk access?

History

- **Problem**: How to minimize disk accesses?
- **Solution**: Store data that is frequently co-accessed at the same physical location
- Consolidates many disk accesses to one

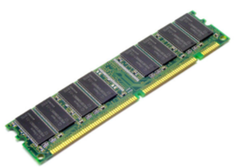


History

- **Example**: Bank
- Co-locate all information about each customer
- Customer Sara deposits $100

2 disk accesses, since data about sara is co-located

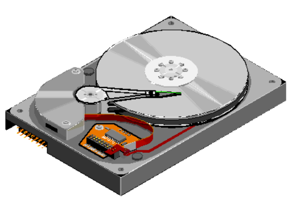
Main Memory



2	Sara	200

Add 100

Disk

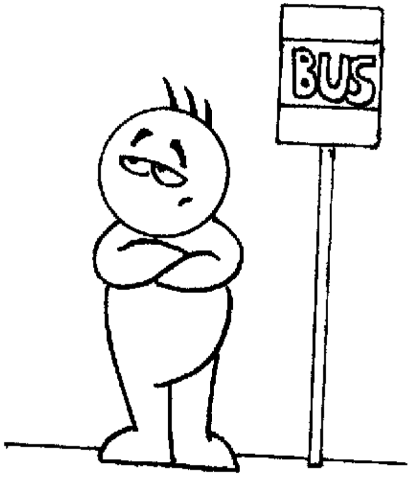


ID	Name	Balance
1	Bob	100
3	Will	450

Database

History

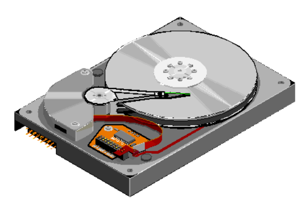
- What to do during a disk access?



- Start running the next operation(s)
- Improves performance
- But data can get corrupted

History

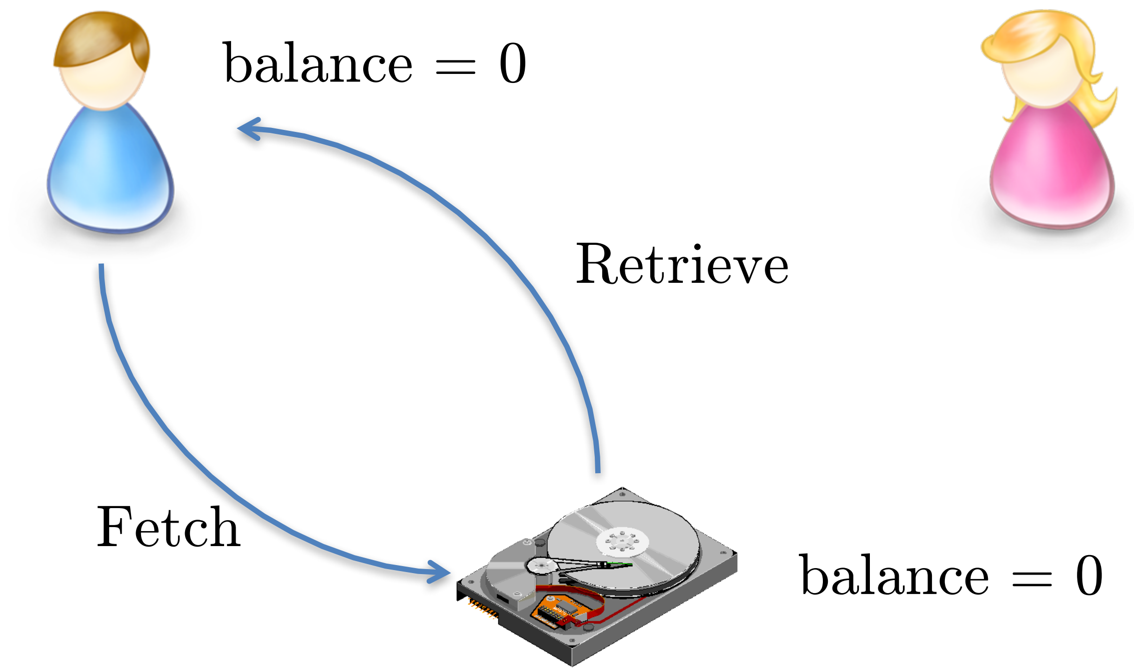
- A couple, Bob and Sara, share a bank account
- Both deposit \$100 at same time



balance $= 0$

History

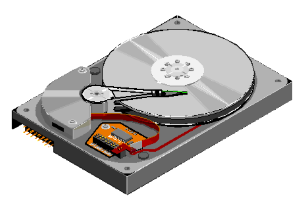
- A couple, Bob and Sara, share a bank account
- Both deposit \$100 at same time



History

- A couple, Bob and Sara, share a bank account
- Both deposit \$100 at same time

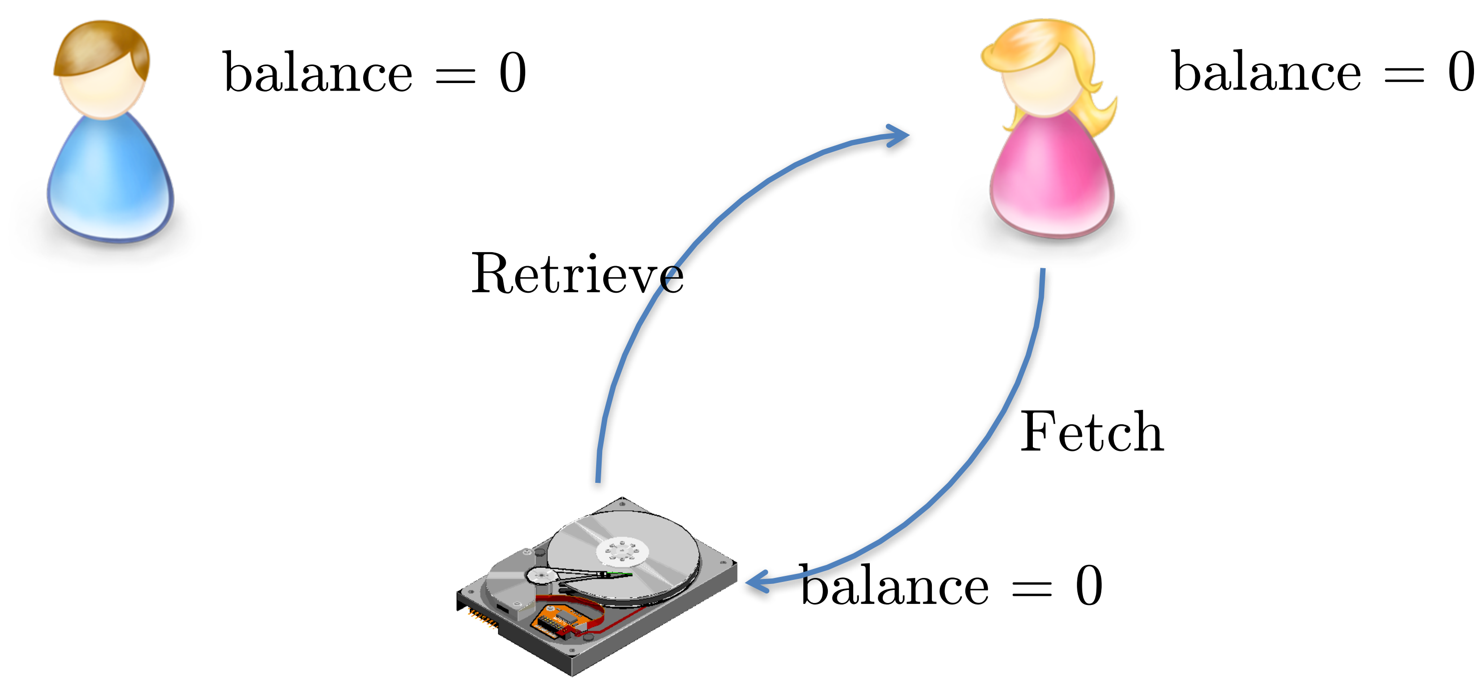
balance $= 0$



balance $= 0$

History

- A couple, Bob and Sara, share a bank account
- Both deposit $100 at same time

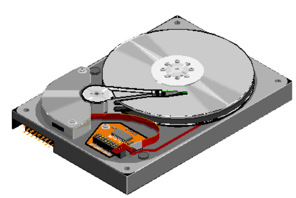


History

- A couple, Bob and Sara, share a bank account
- Both deposit \$100 at same time

balance $= 0$

balance $= 0$



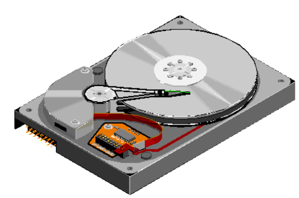
balance $= 0$

History

- A couple, Bob and Sara, share a bank account
- Both deposit $100 at same time

balance = 100

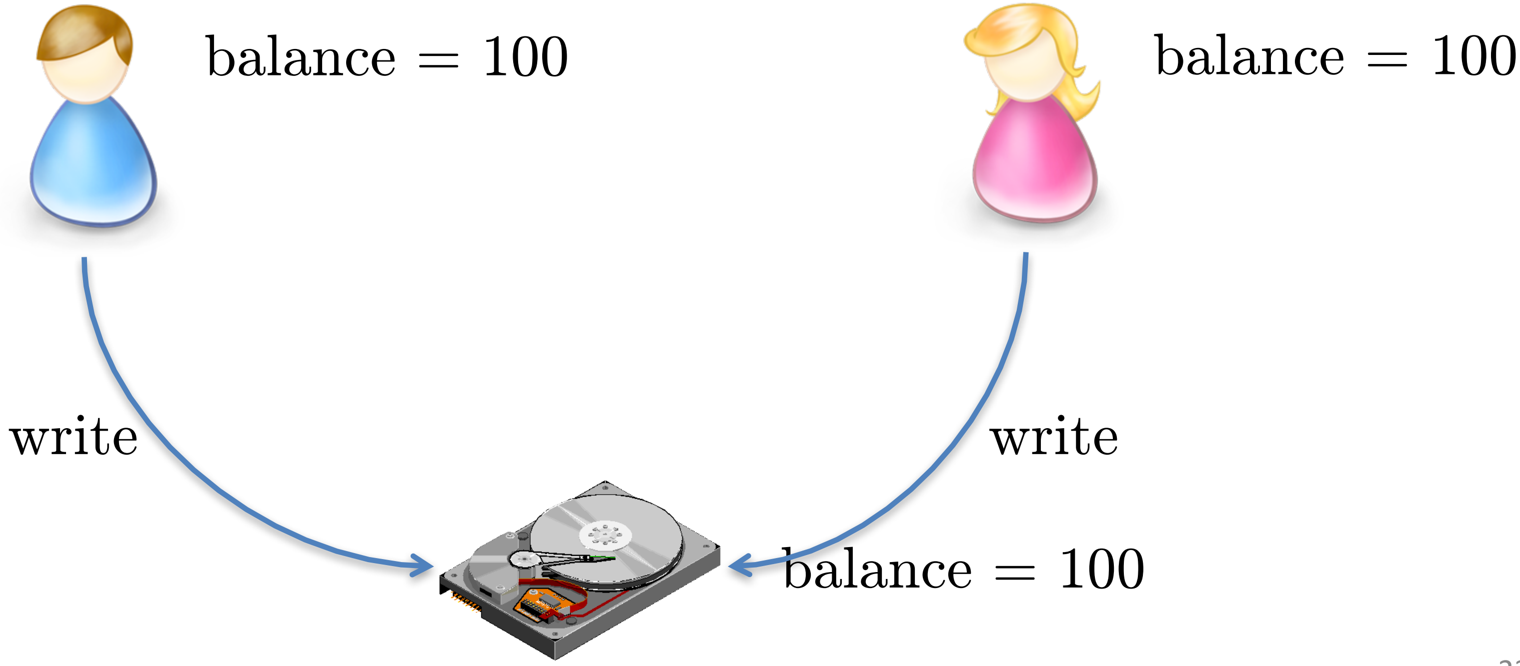
balance = 100



balance = 0

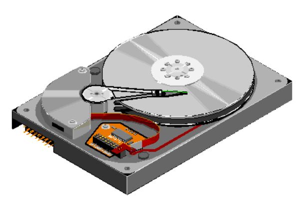
History

- A couple, Bob and Sara, share a bank account
- Both deposit $100 at same time



History

- A couple, Bob and Sara, share a bank account
- Both deposit \$100 at same time

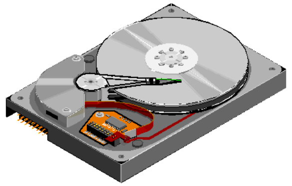


balance $= 100$

History

- A couple, Bob and Sara, share a bank account
- Both deposit $100 at same time

Account balance should be 200!
Bob and Sara lost money.



balance = 100

History

- **Question:** how to achieve concurrency while maintaining data integrity?
- **Insight**: transactions can be concurrent, as long as they don't modify the same data
- **Solution**: locking 
  - Bob locks data, modifies it, releases lock
  - Sara waits until lock is released
- **Downside**:
  - transactions may need to wait for locks.

History

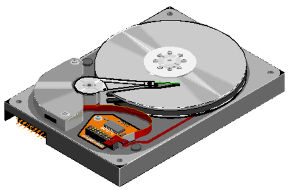
- 3 goals of database design
  - **Speed**
  - Affordability
  - Resilience to system failure

History

- 3 goals of database design
  - Speed
  - **Affordability**
  - Resilience to system failure

History

- Disk was cheap, but not so cheap
- 1 gigabyte for $10000 in 1980
- Avoid storing replicas of same data



ID	name	account-ID	balance
1	Bob	1	100
2	Sara	1	100
3	Trudy	2	450

History

- **Solution**: “Normalization”. Break tables.

ID	name	account-ID	balance
1	Bob	1	100
2	Sara	1	100
3	Trudy	2	450

Customers

ID	name	account-ID
1	Bob	1
2	Sara	1
3	Trudy	2

Accounts

ID	balance
1	100
2	450

- **Bonus**: Easier to maintain data integrity

History

- **Normalization**:
  - Saves storage space
  - Easier to maintain data integrity
- **Downside**: reads are more expensive
  - Need to join tables

Customers

ID	name	account-ID
1	Bob	1
2	Sara	1
3	Trudy	2

Accounts

ID	balance
1	100
2	450

History

- **Data is decomposed accross tables**
- **Query Language: SQL**
  - **select balance from Customers c, Accounts a where c.account-ID = a.ID and c.name = "Bob"**

Customers

ID	**name**	**account-ID**
1	Bob	1
2	Sara	1
3	Trudy	2

Accounts

ID	**balance**
1	100
2	450

History

- 3 goals of database design
  - Speed
  - **Affordability**
  - Resilience to system failure

History

- 3 goals of database design
  - Speed
  - Affordability
  - **Resilience to system failure**

History

- Many things can go wrong
  - Power failure
  - Hardware failure
  - Natural disaster

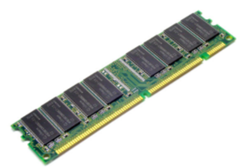
- Data is precious (e.g. bank)

- Provide recovery mechanism

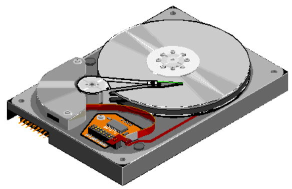
History

- **Example**: Sara transfers \$100 to Anna
- Power stops in the middle

Sara's
balance = 0



Sara's
balance = 100

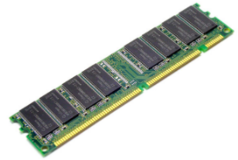


Anna's
balance = 450

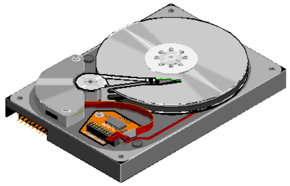
History

- **Example**: Sara transfers \$100 to Anna
- Power stops in the middle

Sara's
balance = 0



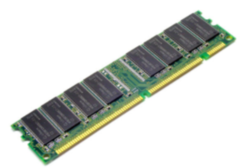
Sara's
balance = 100



Anna's
balance = 450

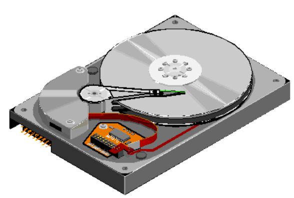
History

- **Example**: Sara transfers \$100 to Anna
- Power stops in the middle



Anna's
balance = 550

Sara's
balance = 0

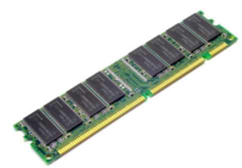


Anna's
balance = 450

History

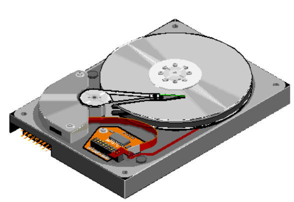
- **Example**: Sara transfers \$100 to Anna
- Power stops in the middle

At this point, power fails



Anna's
balance $= 550$

Sara's
balance $= 0$



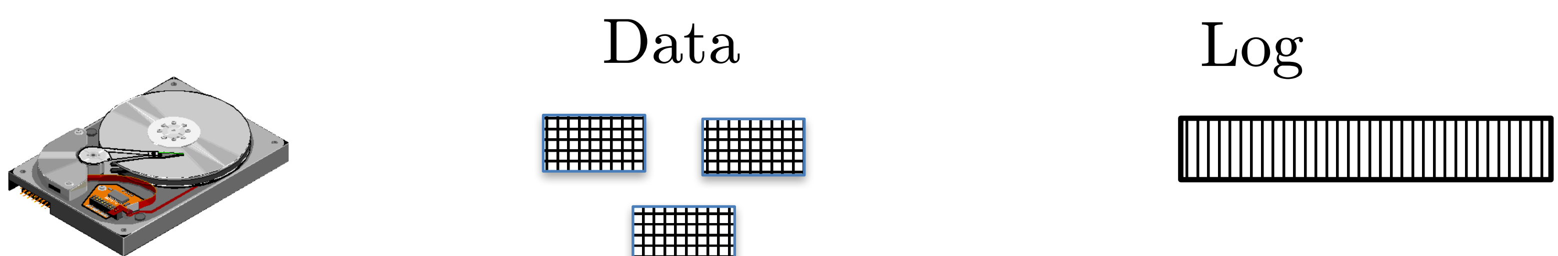
Anna's
balance $= 450$

History

- **Transaction**: a sequence of operations all takes place, or none take place.
- Transactions should be atomic

History

- **Problem:** how to guarantee atomicity?
- **Solution**: use a log (on disk)
- All data changes are recorded in the log
- After power failure, examine log
- Undo changes by unfinished transactions



History

- Data integrity
  - Concurrency (fix with locking)
  - System failure (fix with logging)

- **ACID**
  - Atomicity
  - Consistency
  - Isolation
  - Durability

History

- Summary
  - Speed
  - Affordability
  - Resilience to system failure

- Relational databases:
  - Normalize data into multiple tables
  - ACID (locking & logging)
  - SQL

- Design decisions are motivated by hardware

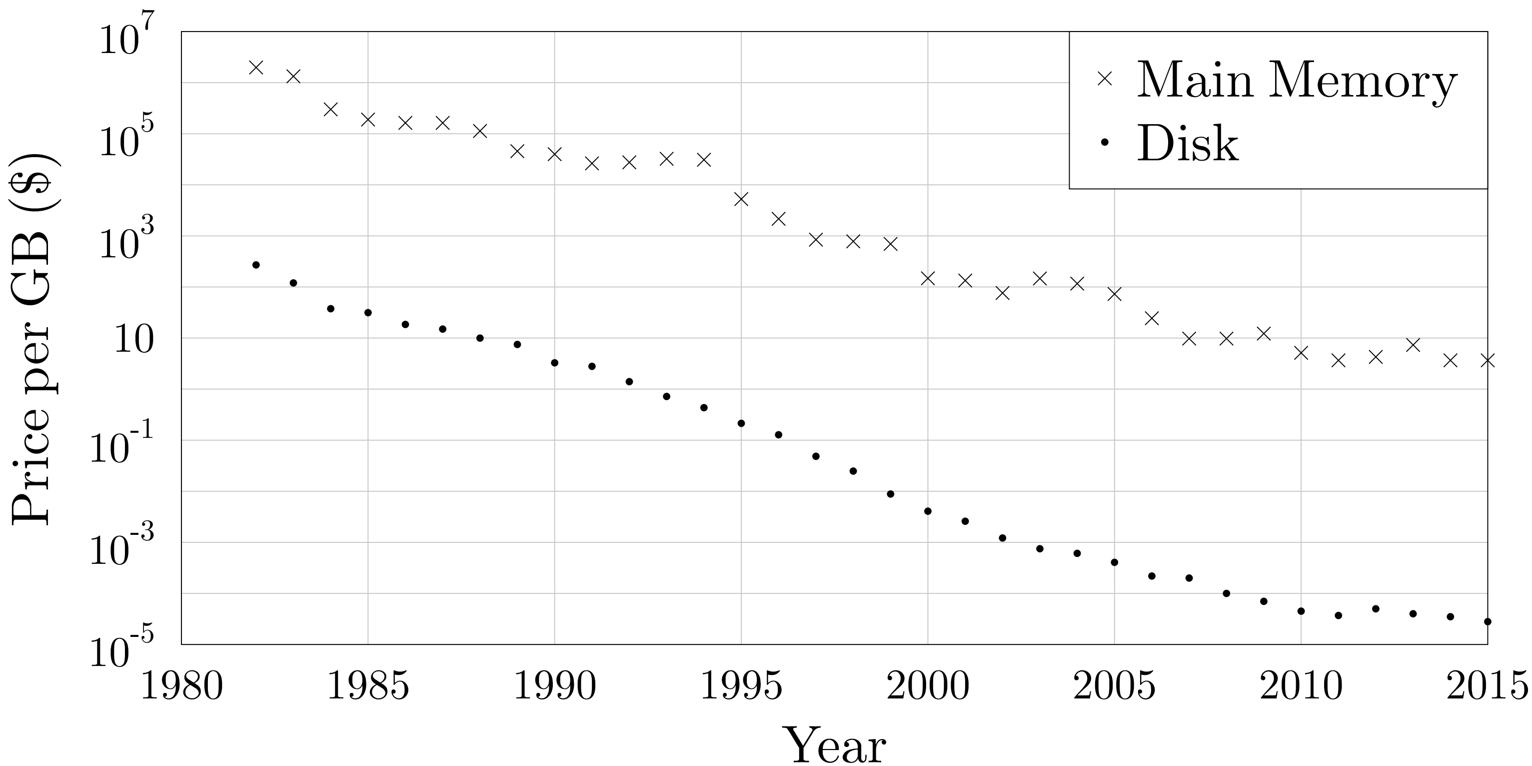
Today

Today

- **What changed in hardware?**
- **How does it affect database design?**

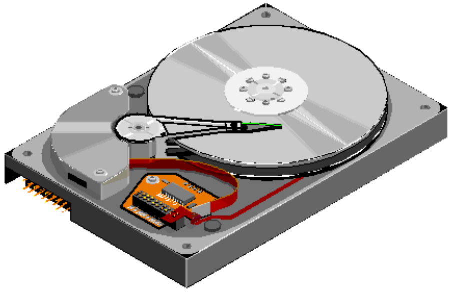
Today

- **Disk is 10^7 times cheaper**
- **Main memory is 10^6 times cheaper**



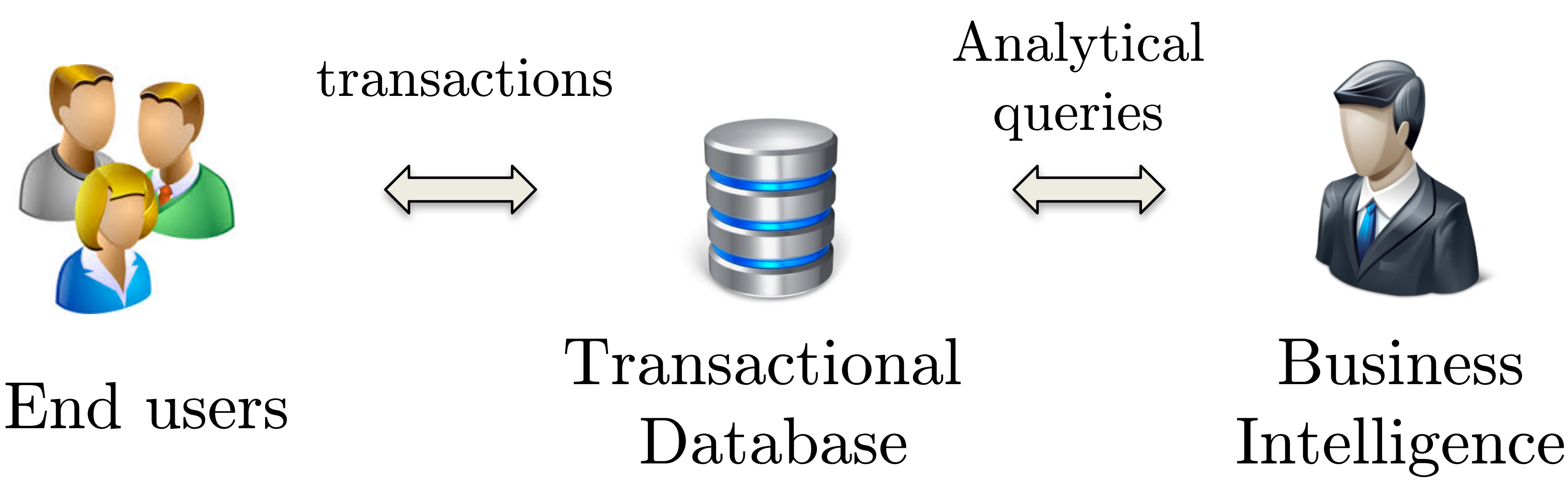
Today

- Disk is now dirt cheap
- Organizations keep all historical data
- Business intelligence
- E.g. Amazon
  - revenues from product X on date Y
  - which products are bought together



Today

- **Traditional system architecture:**

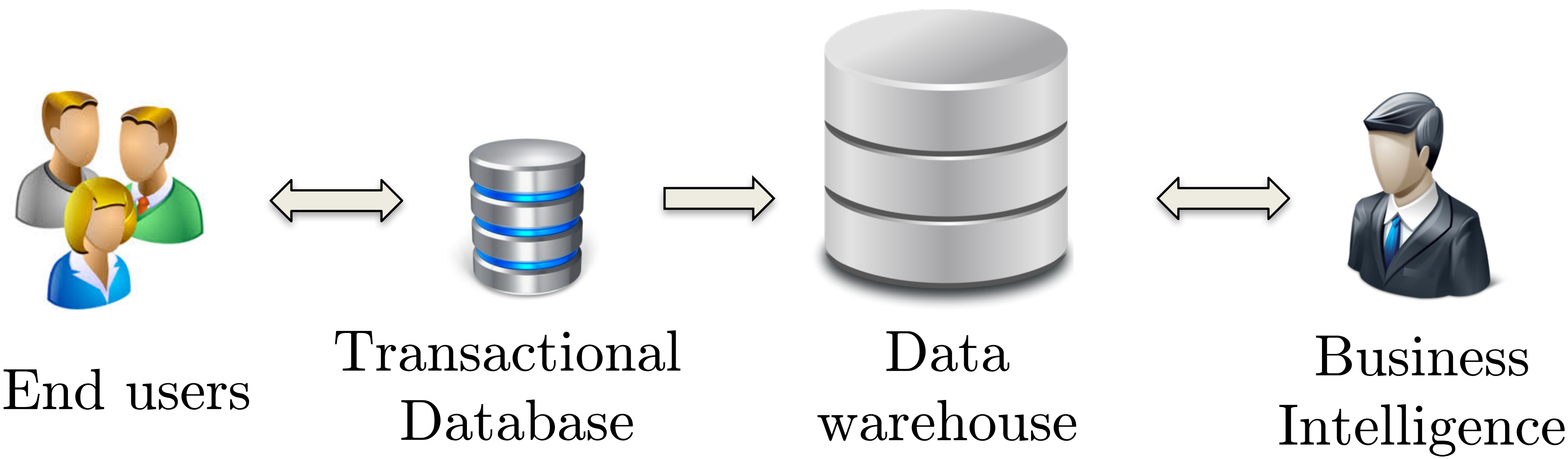


Today

- **Problem:**
  - Analytical queries are expensive
    - Touch a lot of data
    - Disk access
    - Locks
  - They slow down transactions.
  - End-users wait longer

Today

- **Solution:** split database



Today

- Different workloads
- Different internal design

Transactional Database

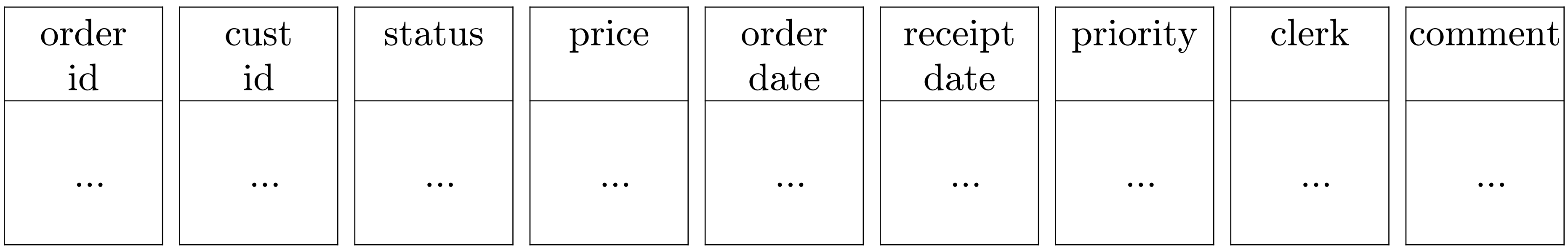
Data warehouse

Today

order id	cust id	product id	price	order date	receipt date	priority	status	comment
...	...	...	...	...	...	...	...	...

- **Example analytical queries**
  - How long is delivery? (2 columns, all rows)
  - Revenue from product X? (2 columns, all rows)
- **Problem:**
  - Data is stored row by row

Today



- **Solution: column-store**
  - Each column is stored separately
  - Good for analytical queries
  - Changes entire architecture
  - Examples: Vertica, Vectorwise, Greenplum, etc.

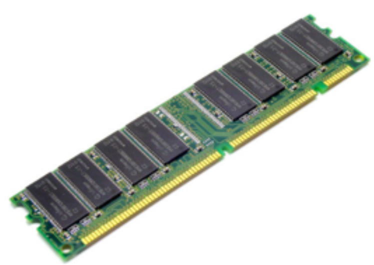
Today

- How are transactional databases affected by hardware changes?

Transactional Database

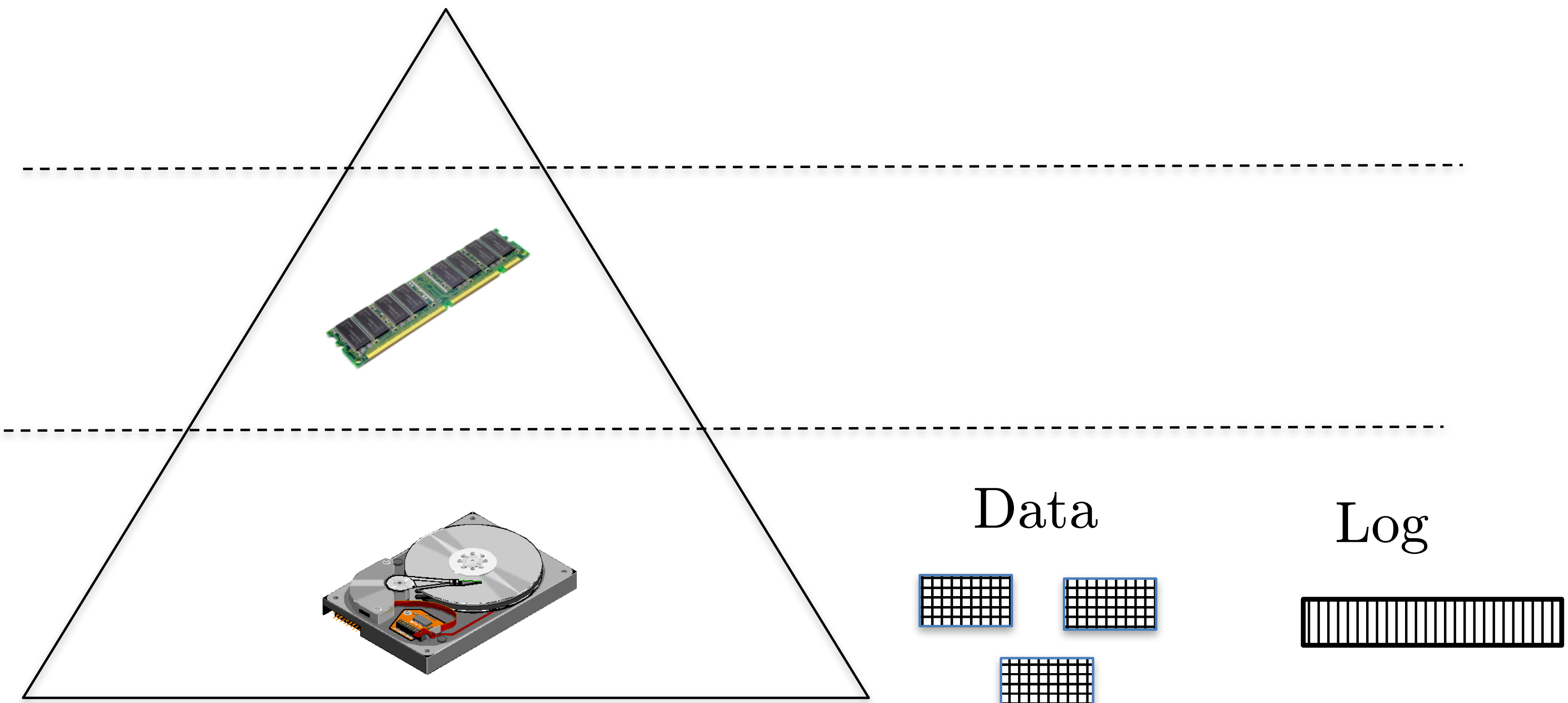
Today

- Main memory is cheaper
- Terabytes are affordable
- Enough to store all transactional data
- E.g. Amazon
  - Products list
  - User accounts



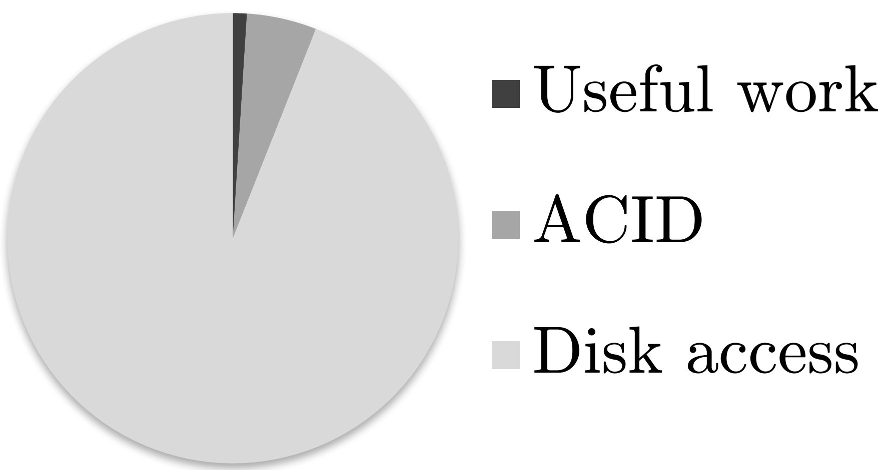
Today

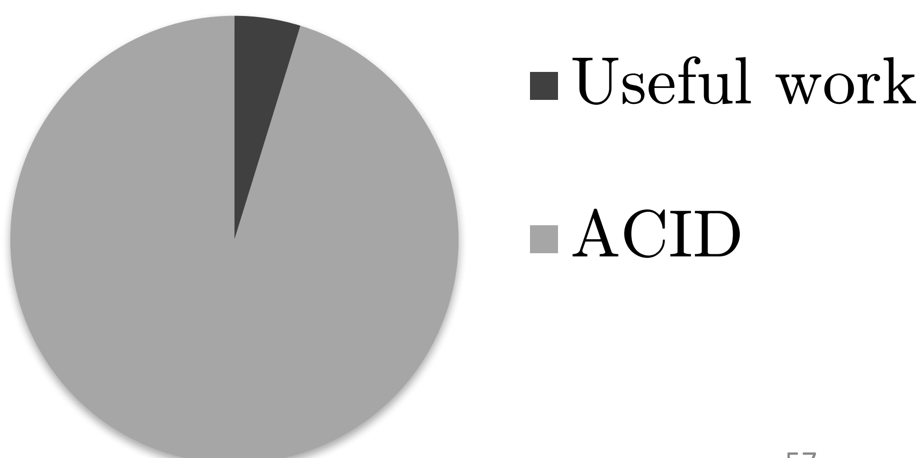
- Main memory was expensive
- Now it's cheaper



Today

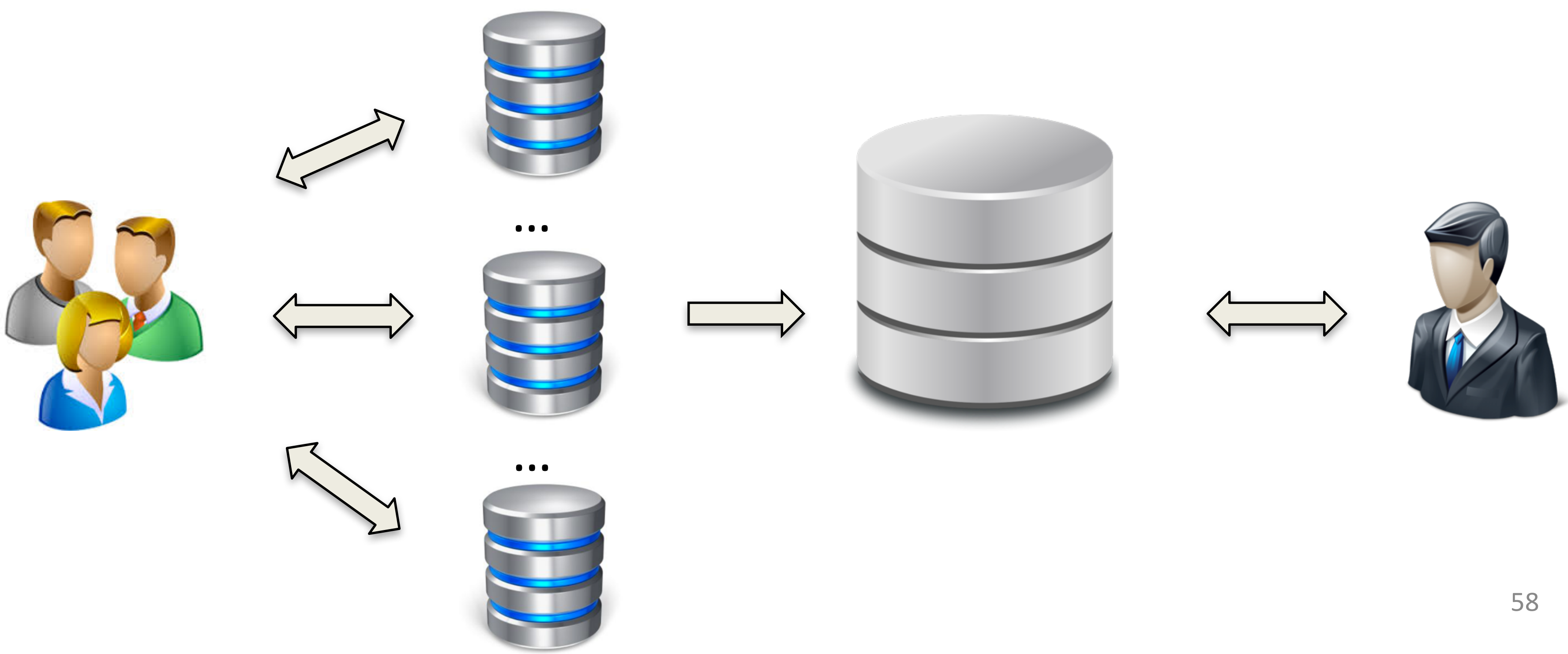
- Transactional databases are main memory databases
- Bottleneck used to be disk access
- The new bottleneck is ACID (logging, locking)





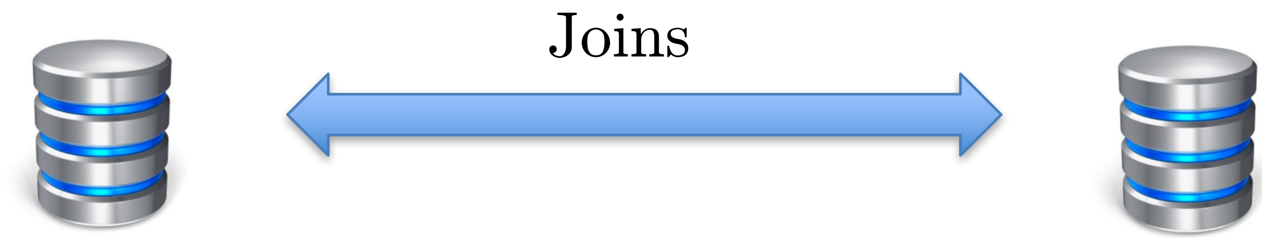
Today

- More challenges
- Due to internet, 100% availability is key
- Data is replicated



Today

- **Joins become more expensive**



Customers

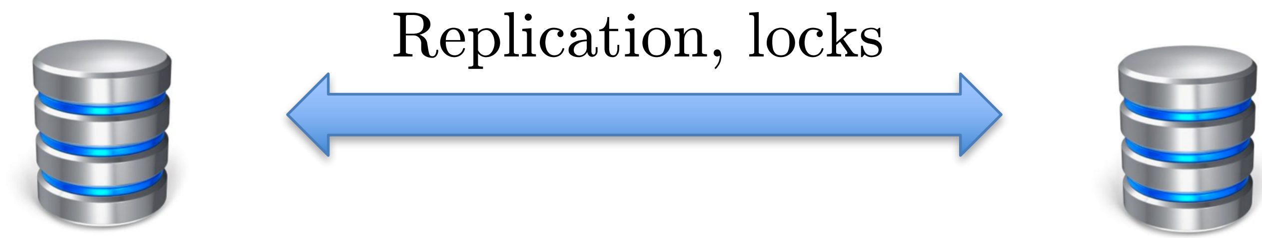
ID	name	account-ID
1	Bob	1
2	Sara	1
3	Trudy	2

Accounts

ID	balance
1	100
2	450

Today

- **Replication and locks become more expensive**



Accounts

ID	balance
1	100
2	450

Accounts

ID	balance
1	100
2	450

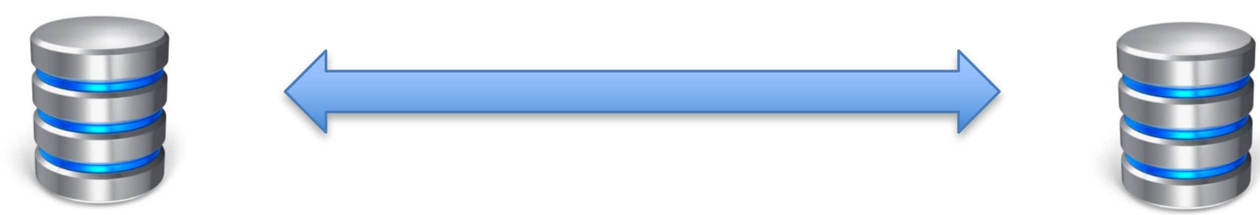
Today

- **Single machine bottlenecks:**

Logging & locking

- **Multiple machine bottlenecks:**

Replication, locks, joins



Today

- NoSQL and NewSQL address these
  - NoSQL simplifies
  - NewSQL engineers

NoSQL
(MongoDB)

NoSQL
(MongoDB)

Rank			DBMS	Database Model	Score		
Dec 2016	Nov 2016	Dec 2015			Dec 2016	Nov 2016	Dec 2015
1.	1.	1.	Oracle	Relational DBMS	1404.40	-8.60	-93.15
2.	2.	2.	MySQL	Relational DBMS	1374.41	+0.85	+75.87
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1226.66	+12.86	+103.50
4.	4.	↑ 5.	PostgreSQL	Relational DBMS	330.02	+4.20	+49.92
5.	5.	↓ 4.	MongoDB	Document store	328.68	+3.21	+27.29
6.	6.	6.	DB2	Relational DBMS	184.34	+2.89	-11.78
7.	7.	↑ 8.	Cassandra	Wide column store	134.28	+0.31	+3.44
8.	8.	↓ 7.	Microsoft Access	Relational DBMS	124.70	-1.27	-15.51
9.	9.	↑ 10.	Redis	Key-value store	119.89	+4.35	+19.36
10.	10.	↓ 9.	SQLite	Relational DBMS	110.83	-1.17	+9.98

http://db-engines.com/en/ranking

NoSQL

- Name popularized in 2009
- Conference on "open source distributed non-relational databases"
- NoSQL was a hashtag

NoSQL

- Different types
  - Document stores
  - Column-oriented
  - Key-value-stores
  - Graph databases

mongoDB
cassandra
redis

Similar

neo4j

Different

NoSQL

- MongoDB - Main decisions
  1. No joins
     - Aggregate related data into “documents”
     - Reduces network traffic
     - Data modeling is harder
  2. No ACID
     - Faster
     - Concurrency & system failure can corrupt data

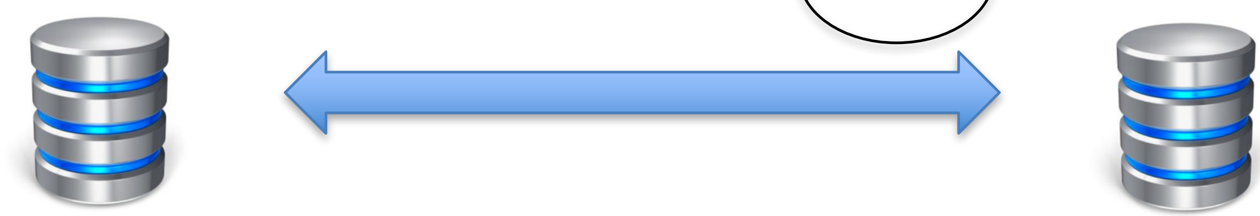
NoSQL

- **Single machine bottlenecks:**

Logging & locking

- **Multiple machine bottlenecks:**

Replication, locks, joins



NoSQL

- **To avoid joins, data is de-normalized**

Customers

ID	**name**	**account-ID**
1	Bob	1
2	Sara	1
3	Trudy	2

Accounts

ID	**balance**
1	100
2	450

ID	**name**	**account-ID**	**balance**
1	Bob	1	100
2	Sara	1	100
3	Trudy	2	450

NoSQL

- **In MongoDB**

Collection of customer Documents

```
{ name:        "Bob",
  account-ID:  1,
  balance:     100 }

{ name:        "Sara",
  account-ID:  1,
  balance:     100 }

{ name:        "Trudy",
  account-ID:  2,
  balance:     450 }
```

- **db.customers.find(name:"Sara")**

NoSQL

- Documents are flexible

```
{ name:            "Bob",
  account-ID:      1,
  balance:         100,
  favorite-color: "red"
  credit-score: 3.0
}
```

```
{ name:            "Sara",
  account-ID:      1,
  balance:         100
  hobbies: ["rowing", "running"]
}
```

NoSQL

- **Main point: no need for joins**
- **All related data is in one place**

```
{ name:           "Bob",
  account-ID:     1,
  balance:        100,
  favorite-color: "red"
  credit-score: 3.0
}
```

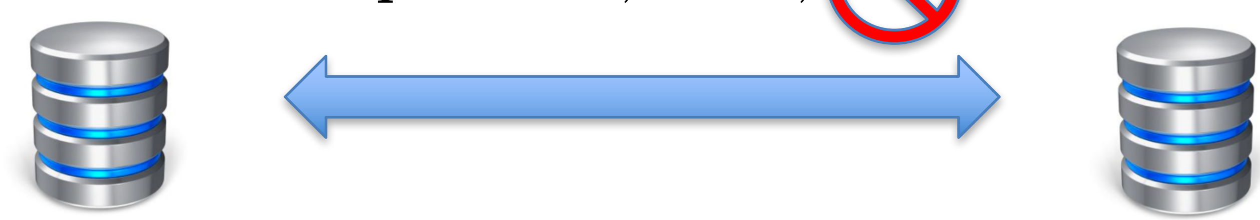
NoSQL

- **Single machine bottlenecks:**

Logging & locking

- **Multiple machine bottlenecks:**

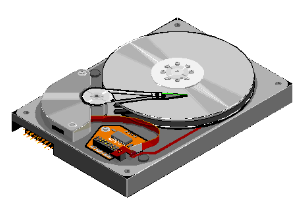
Replication, locks, ~~joins~~



NoSQL

- MongoDB does not lock
- Recall Bob and Sara
- Deposit $100 at same time to shared account
- Overwrite each other's update

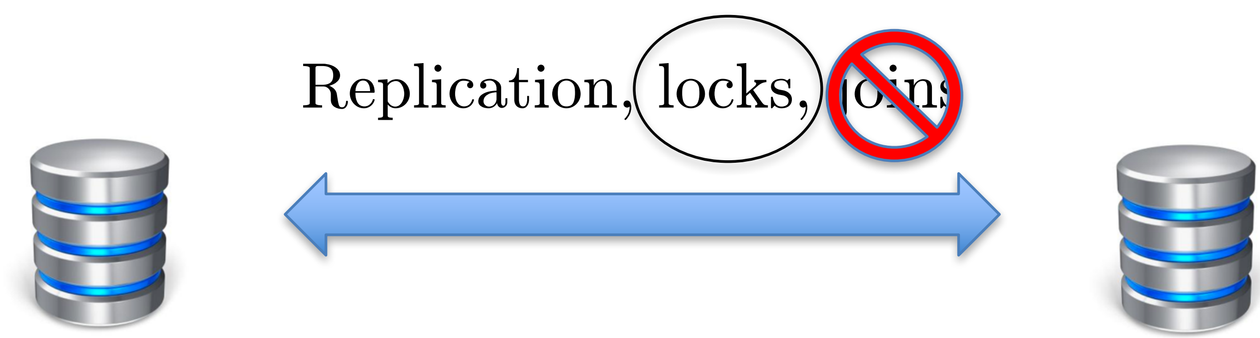
No general way to prevent this



NoSQL

- **Single machine bottlenecks:**

- **Multiple machine bottlenecks:**



NoSQL

- **Eventual consistency**
- **Different operation order across replicas**
- **E.g. concurrent addition and multiplication**

Deposit
Add 100

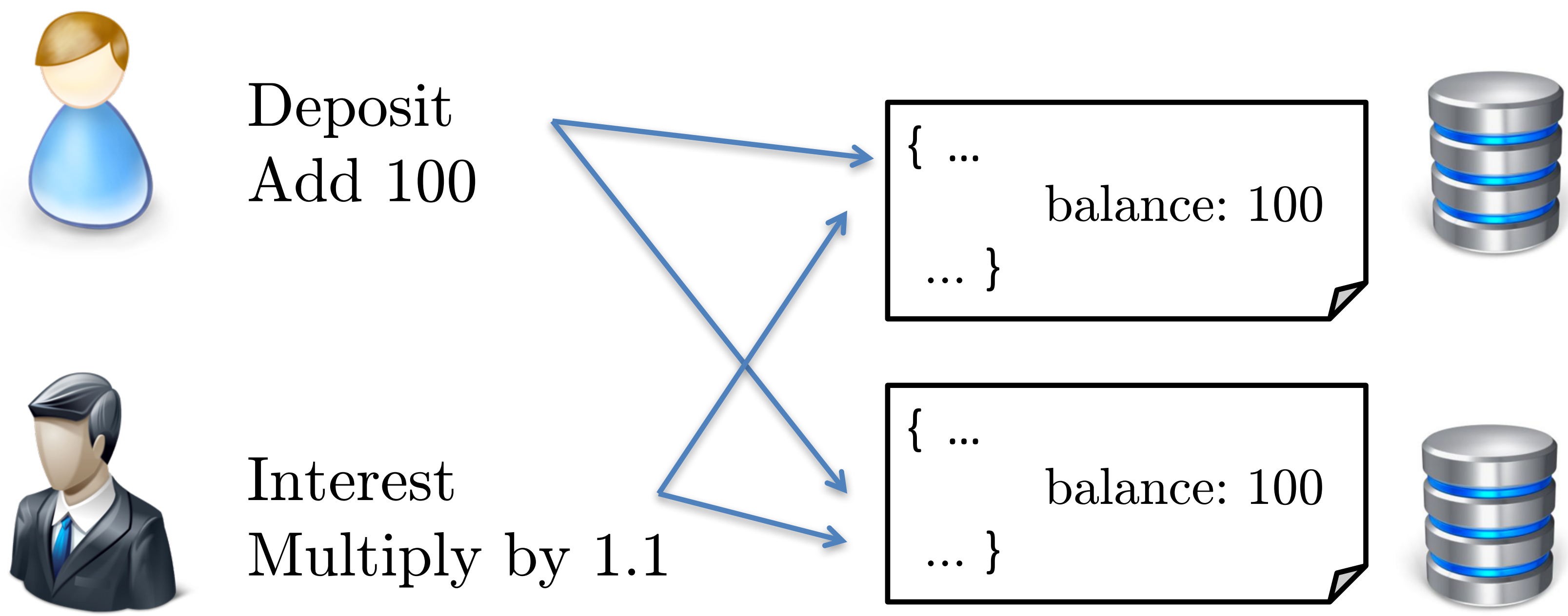
{ ...
balance: 100
... }

Interest
Multiply by 1.1

{ ...
balance: 100
... }

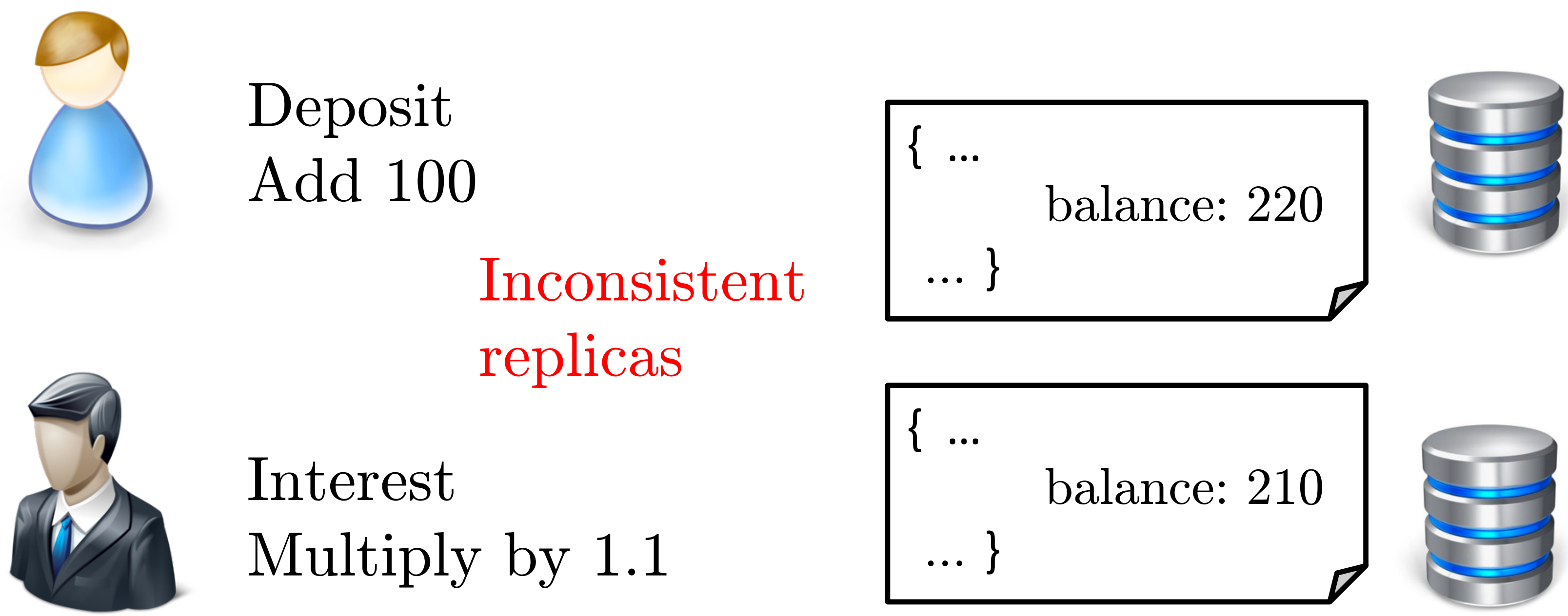
NoSQL

- **Eventual consistency**
- **Different operation order across replicas**
- **E.g. concurrent addition and multiplication**



NoSQL

- **Eventual consistency**
- **Different operation order across replicas**
- **E.g. concurrent addition and multiplication**



NoSQL

- **Single machine bottlenecks:**

Logging & locking

- **Multiple machine bottlenecks:**

Replication, locks, joins

NoSQL

- When to use MongoDB?
- Non-interacting entities
  - No sharing (e.g. bank account)
  - No exchanging (e.g. money transfers)
- Commutative operations on data
- You need a flexible data model

NewSQL

NewSQL

- ACID & good performance
- Redesign internal architecture.

NewSQL

- Data is normalized into multiple tables
- Tables partitioned and replicated across machines

Customers

ID	name	account-ID
1	Bob	1
2	Sara	1
3	Trudy	2

Accounts

ID	balance
1	100
2	450

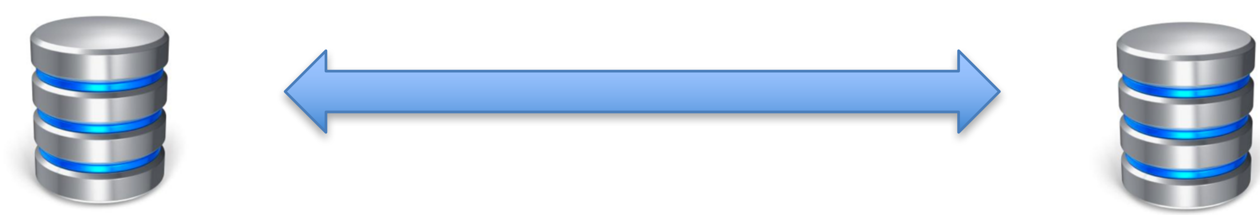
NewSQL

- Single machine bottlenecks:

Logging & locking

- Multiple machines bottlenecks:

Replication, locks, joins



NewSQL

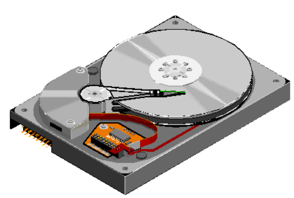
- **History**: concurrent transactions were introduced since disk was slow
- **Today**: Now all data is in main memory
- Transactions in main memory are fast
- Less need for concurrency

- VoltDB removes concurrency
- Thus, no need for locking

NewSQL

- Recall Bob and Sara
- Deposit $100 at same time to shared account
- Overwrite each other's update

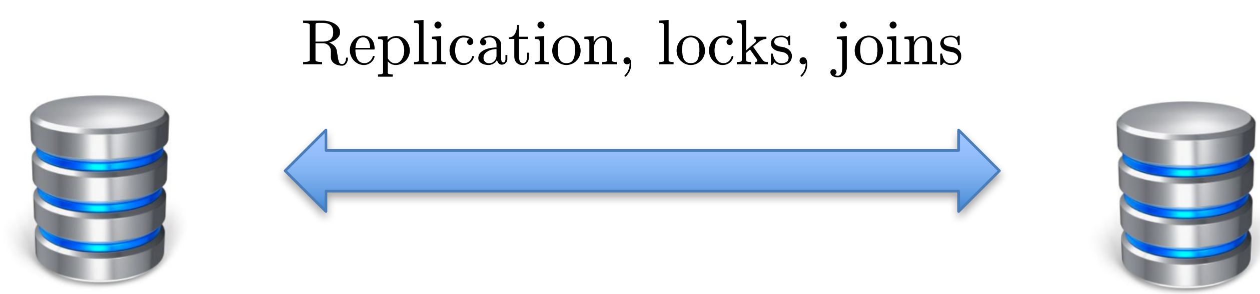
In VoltDB, this cannot happen



NewSQL

- **Single machine bottlenecks:**

- **Multiple machine bottlenecks:**



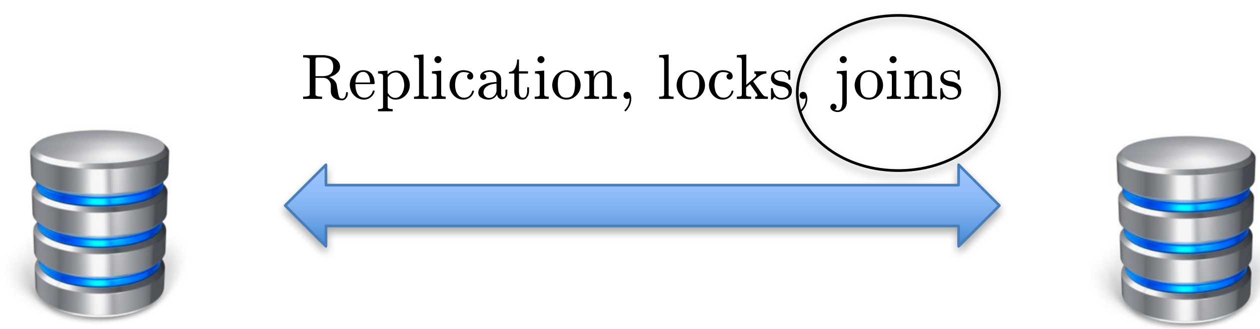
NewSQL

- **History**: log introduced for recovery
- **Today**: it takes too long to recover from log
- **Instead, replicate data across machines**
- **If one machine fails, others continue working**
- **Simplifies logging**

NewSQL

- **Single machine bottlenecks:**

- **Multiple machine bottlenecks:**



NewSQL

- Try to avoid joins across machines
- Store data that is commonly accessed at same time on same machine

Customers

ID	name	account-ID
1	Bob	1
2	Sara	1

Accounts

ID	balance
1	100

Customers

ID	name	account-ID
3	Trudy	2

Accounts

ID	balance
2	450

NewSQL

Customers

ID	**name**	**account-ID**
1	Bob	1
2	Sara	1

Accounts

ID	**balance**
1	100

Customers

ID	**name**	**account-ID**
3	Trudy	2

Accounts

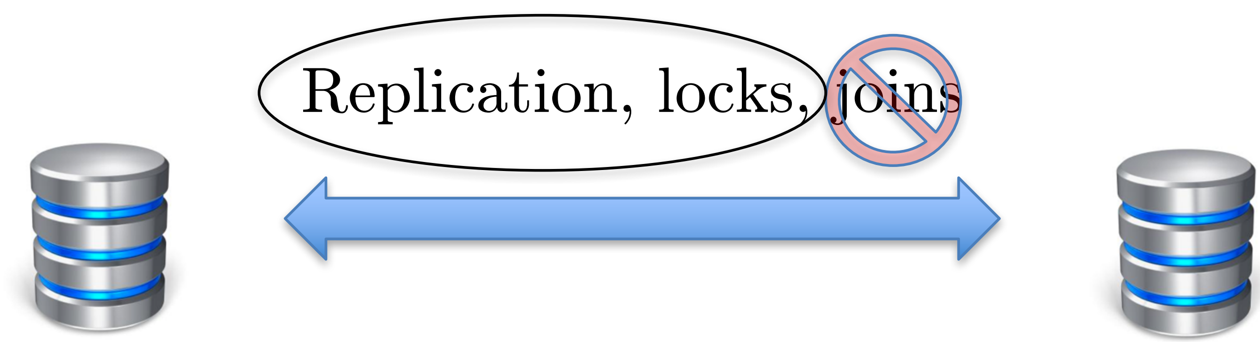
ID	**balance**
2	450

- Alleviates problem
- Does not solve it (e.g. money transfer)

NewSQL

- Single machine bottlenecks:

- Multiple machine bottlenecks:



NewSQL

- **Tables are replicated**
- **Enforce operation order across replicas**

Deposit
Add 100

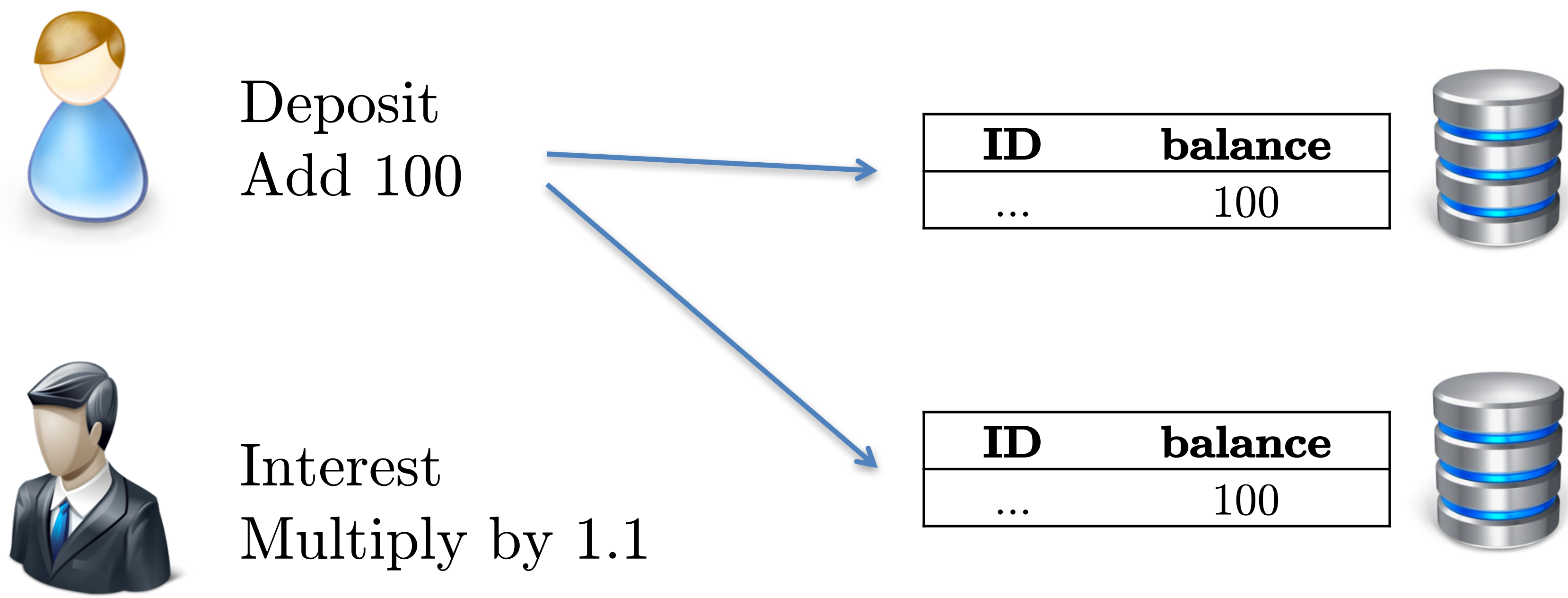
ID	**balance**
...	100

Interest
Multiply by 1.1

ID	**balance**
...	100

NewSQL

- **Tables are replicated**
- **Enforce operation order across replicas**



NewSQL

- **Tables are replicated**
- **Enforce operation order across replicas**

Deposit
Add 100

ID	balance
...	200

Interest
Multiply by 1.1

ID	balance
...	200

NewSQL

- **Tables are replicated**
- **Enforce operation order across replicas**

Deposit
Add 100

ID	balance
...	200

Interest
Multiply by 1.1

ID	balance
...	200

NewSQL

- **Tables are replicated**
- **Enforce operation order across replicas**

Deposit
Add 100

ID	**balance**
...	200

Interest
Multiply by 1.1

ID	**balance**
...	200

NewSQL

- **Tables are replicated**
- **Enforce operation order across replicas**

Deposit
Add 100

ID	balance
...	220

Interest
Multiply by 1.1

ID	balance
...	220

NewSQL

- **Tables are replicated**
- **Enforce operation order across replicas**

Deposit
Add 100

ID	**balance**
...	220

Interest
Multiply by 1.1

ID	**balance**
...	220

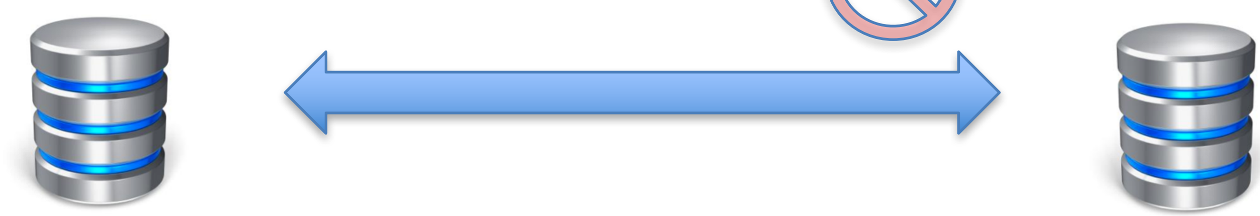
NewSQL

- Single machine bottlenecks:

Logging & locking

- Multiple machine bottlenecks:

Replication, locks, joins



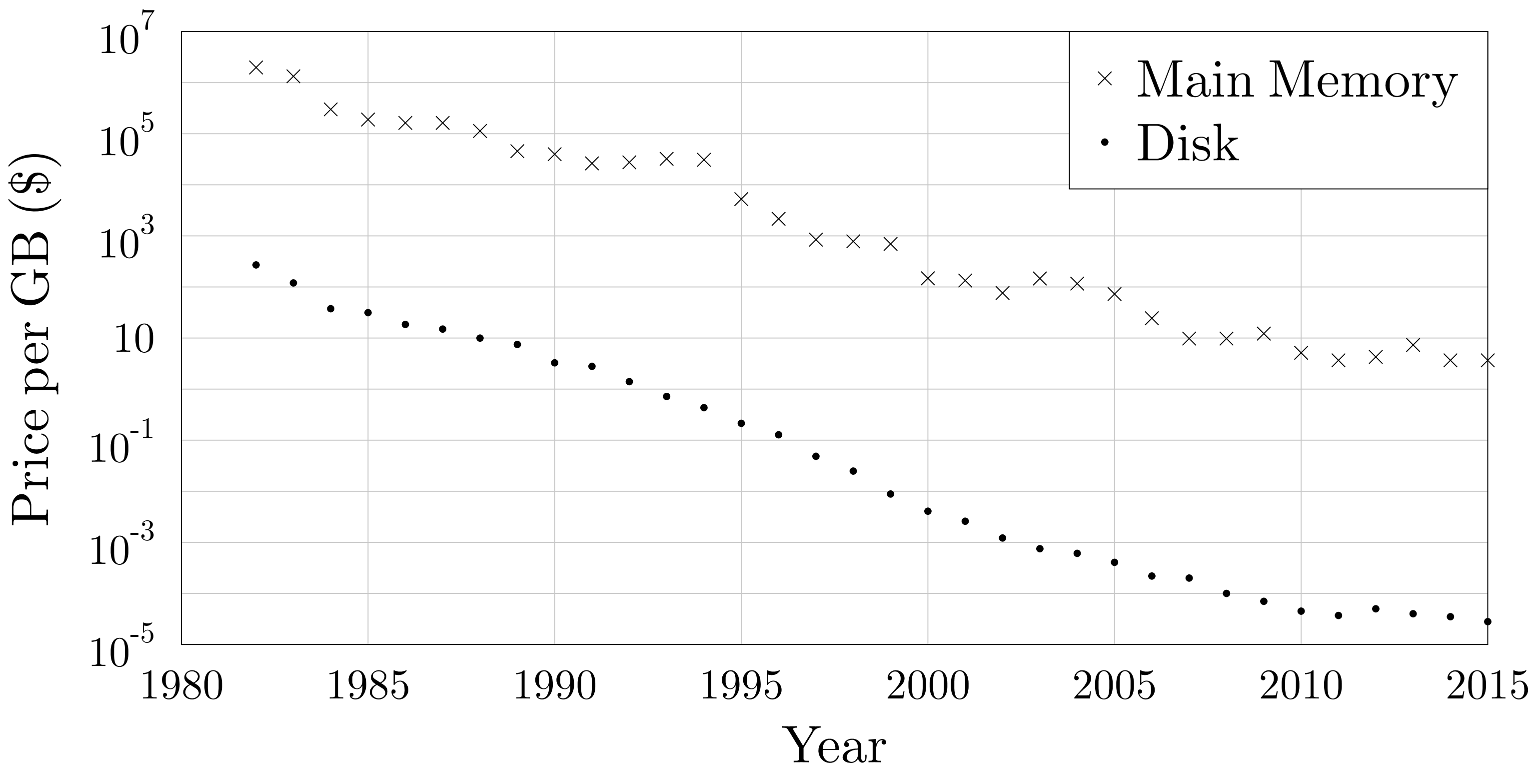
NewSQL

- When to use VoltDB?
  - run at scale
  - You need 100% availability
  - You need ACID

Conclusion

Conclusion

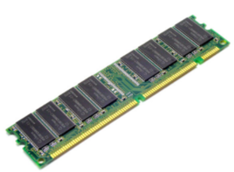
- **Hardware is cheaper**



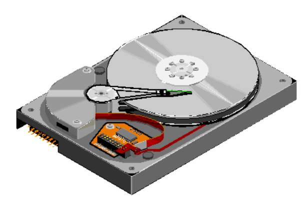
Conclusion

Transactional
Database

Data
warehouses



Row-store



Column-store

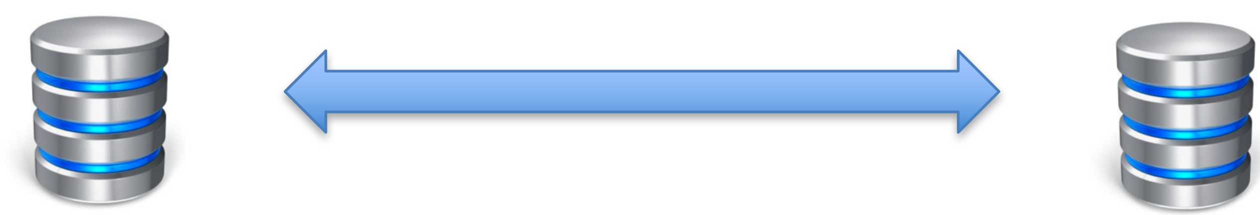
Conclusion

- Single machine bottlenecks:

Logging & locking

- Multiple machines bottlenecks:

Replication, locks, joins



Conclusion

- NoSQL adapts by simplifying
  - No ACID
  - No joins
- NewSQL adapts by reengineering
  - ACID
  - Removes concurrency
  - Simplifies logging
  - Smart but limited partitioning across servers

Conclusion

- **Disclaimer**: there is much more
  - Scientific databases:
  - Time series databases:
  - Graph databases
- Caveat: rapid changes
- But hopefully now you have reasoning tools

Conclusion

- **Thanks!**