NoSQL vs. NewSQL

Demystifying the Zoo of Contemporary Database Systems

ComputeFest

Niv Dayan

12 January, 2017

DASlab

INSTITUTE FOR APPLIED
COMPUTATIONAL SCIENCE
AT HARVARD UNIVERSITY

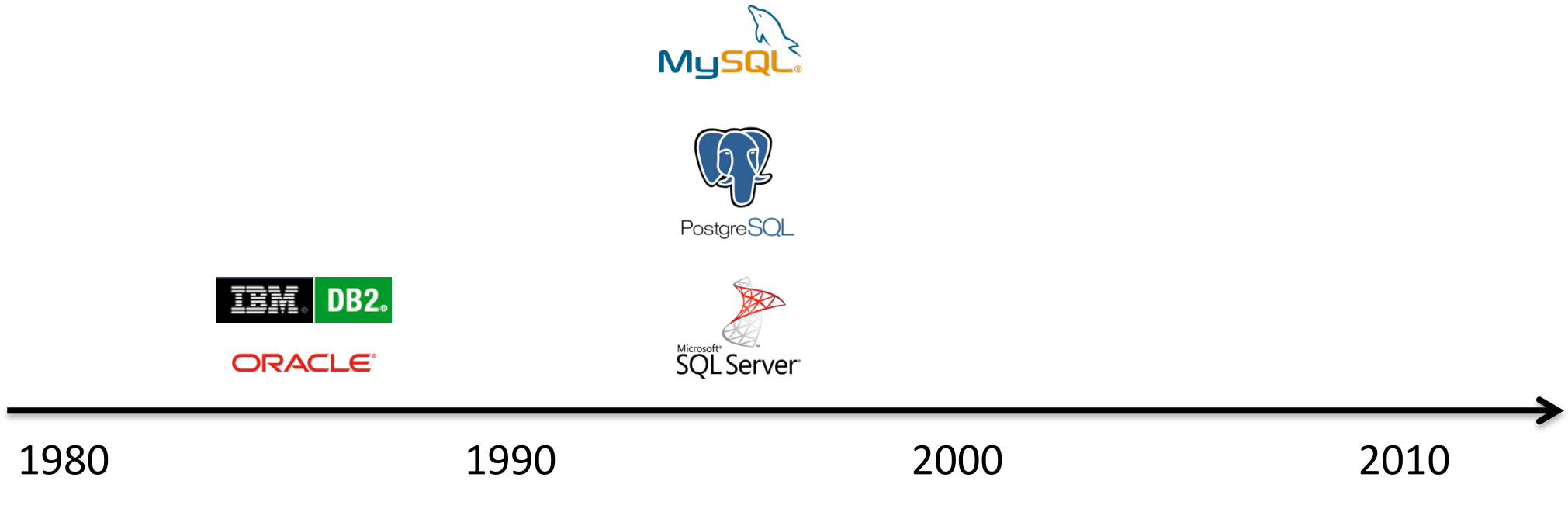
HARVARD
School of Engineering
and Applied Sciences

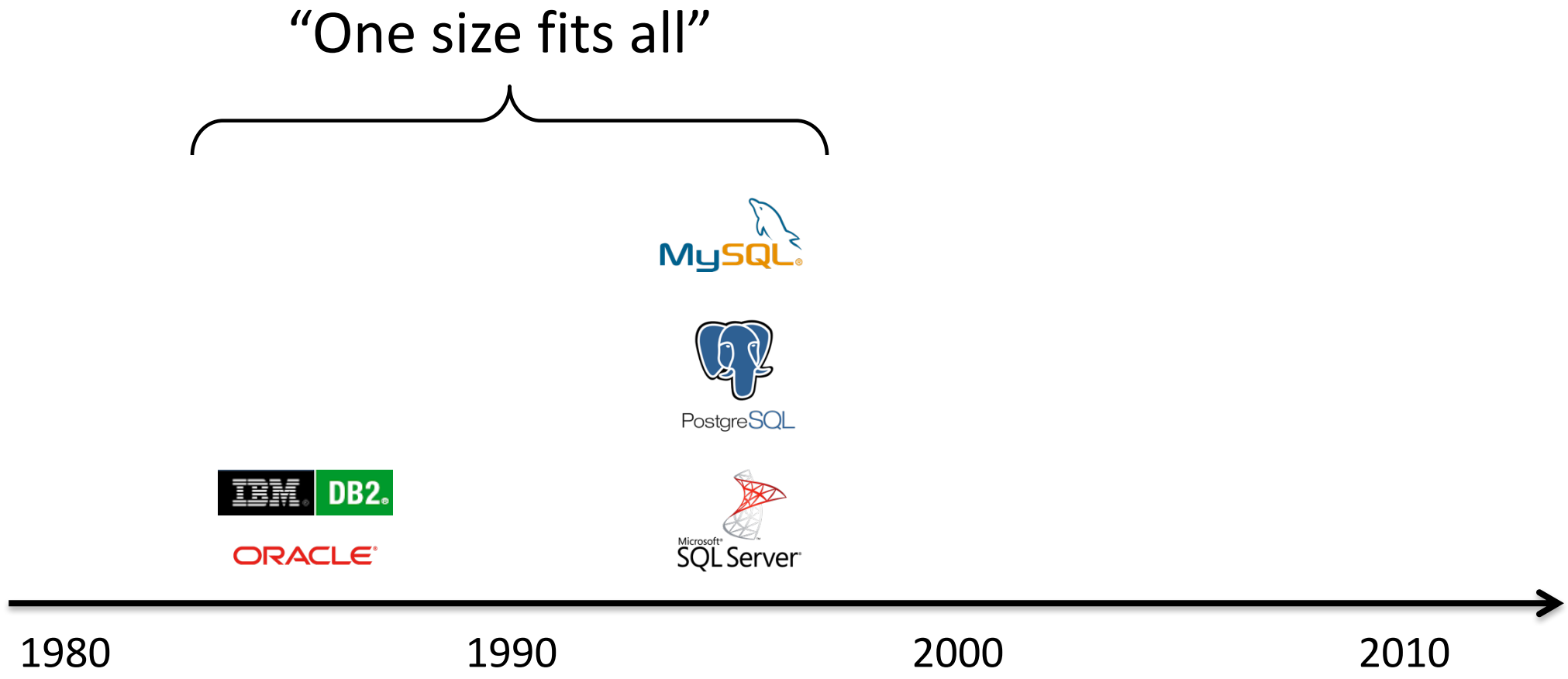

ALSO
BIG
DATA
TOOLS
DISK
TARGET
APPLIED
SENSOR
SHARED
DEFINITION
CURRENT
MAY
MOVING
WITHIN
EXAMPLES
EVERY
SET
USE
RELATIONAL
LARGER
CONTINUES
SOCIAL
COMPLEX
CITATION
TENS
ANALYTICS
PRACTITIONERS
CAPTURE
INTERNET
BUSINESS
SETS
DISTRIBUTED
GENOMICS
MANAGEMENT
TERABYTES
CASE
ABILITY
BIOLOGICAL
PROCESSING
CAPACITY
PETABYTES
INCLUDE
ELAPSED
SYSTEMS
TOLERABLE
HUNDREDS
NETWORKS
DATABASES
RECORDS
GARTNER
WORKING
STORAGE
SIZE
SEARCH
PARALLEL
SAN
MASSIVELY
GROW
DIFFICULTY

NoSQL NewSQL

NoSQL NewSQL

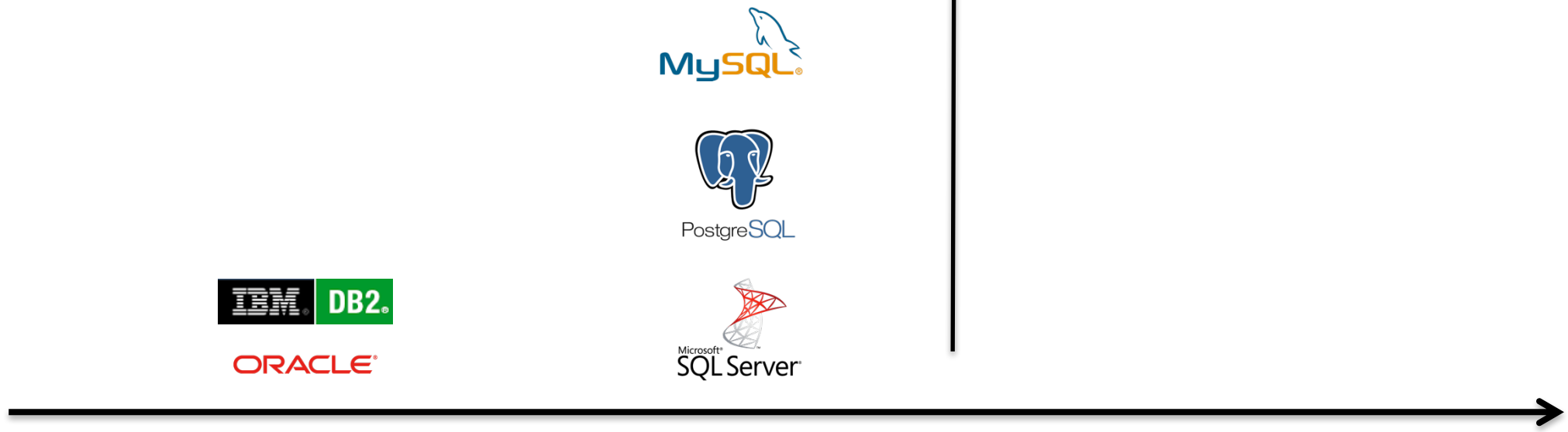
Buzzwords => **Principles**


MySQL
PostgreSQL
IBM DB2
ORACLE
Microsoft SQL Server
1980
1990
2000
2010
Time


"One size fits all"
MySQL
PostgreSQL
IBM DB2
ORACLE
Microsoft SQL Server
1980
1990
2000
2010
Time

"BigData" era commences

"One size fits all"



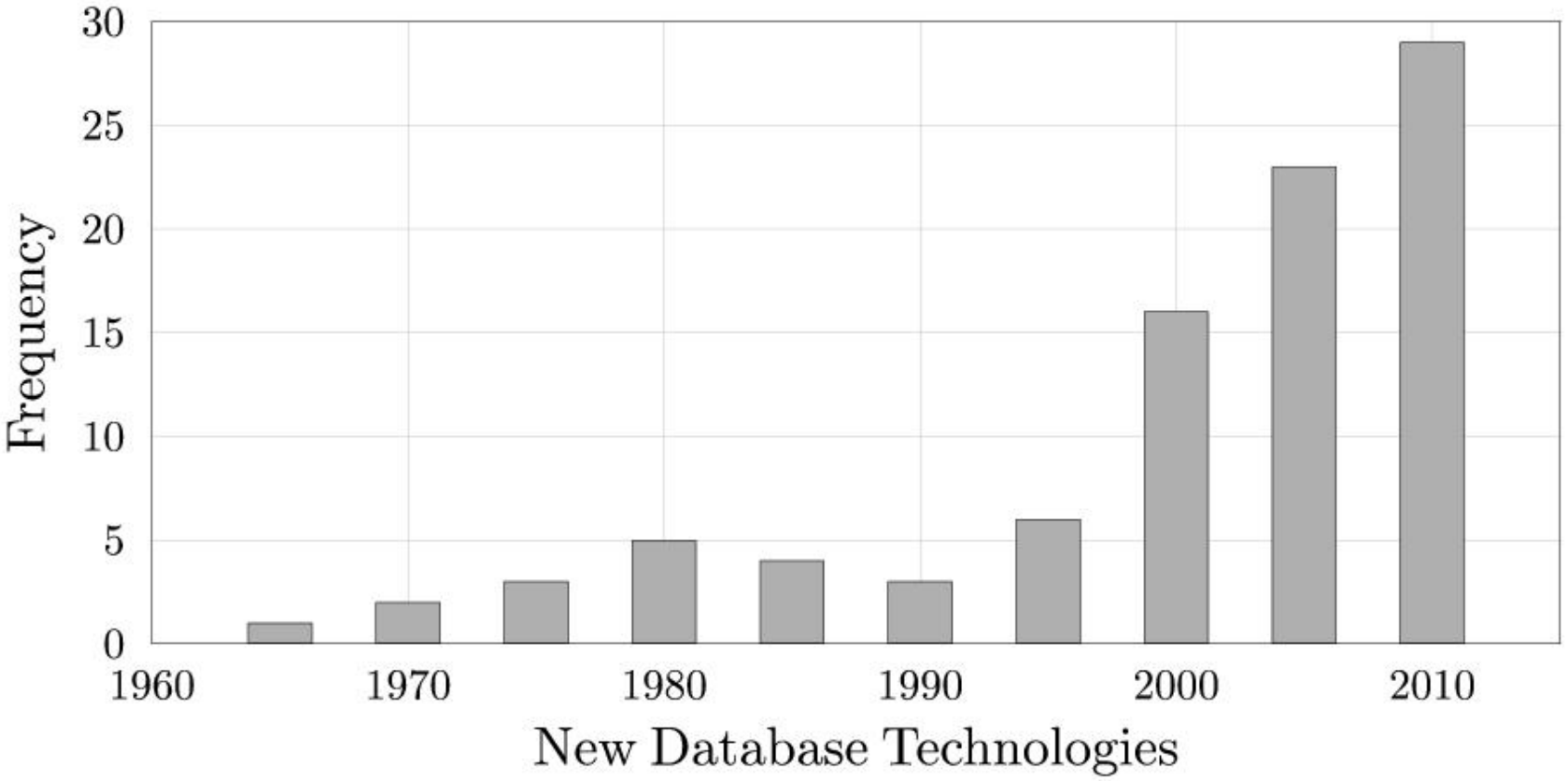
"BigData" era commences

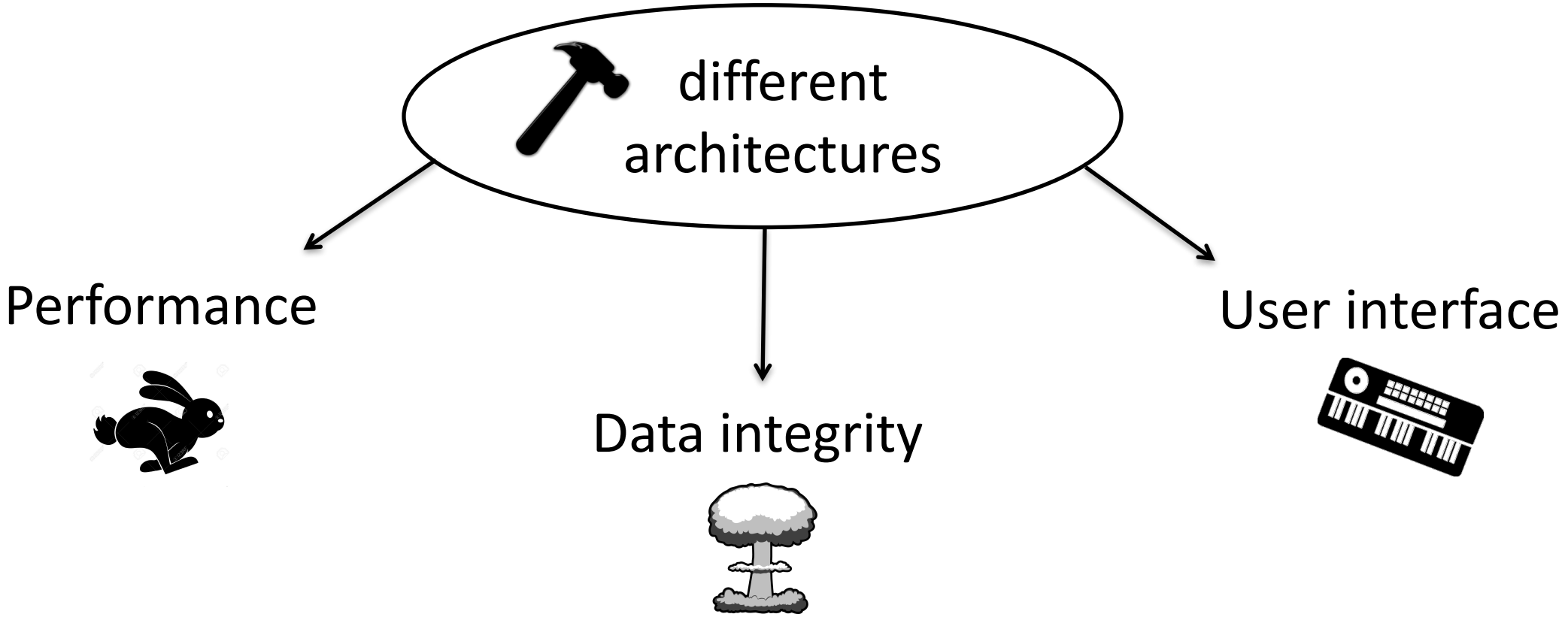
"One size fits all"

Specialization

Most database systems were created recently!

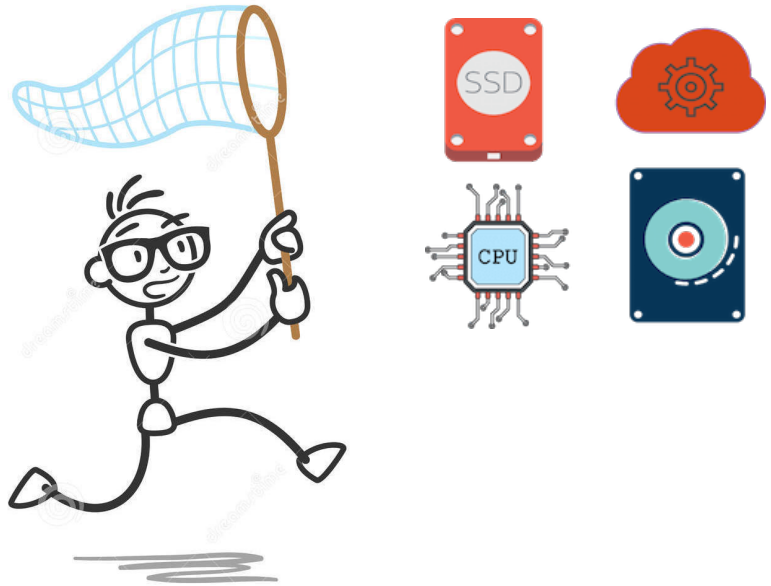
(db-engines.com)





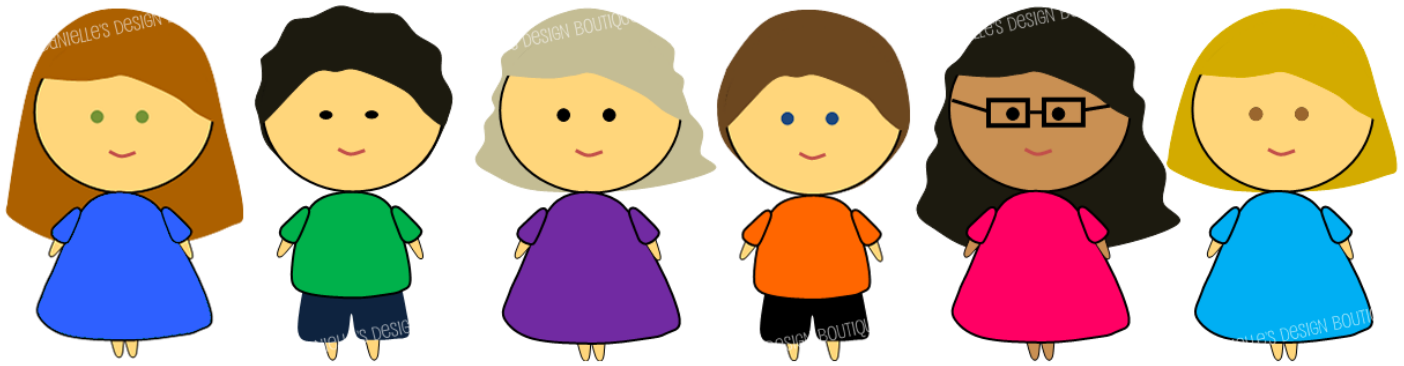
"Which database system is right for me?"

Theme: databases trends follow hardware trends



Claim: modern databases are adaptations

Different backgrounds, No Assumptions



Agenda

1. Traditional databases
2. BigData challenges
3. Analytical databases
4. NoSQL databases
5. NewSQL databases

tutorial

Total time: 90 minutes

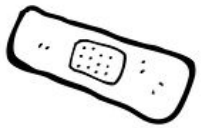
Why MongoDB?

Rank Jan 2017	Rank Dec 2016	Rank Jan 2016	DBMS
1.	1.	1.	Oracle
2.	2.	2.	MySQL
3.	3.	3.	Microsoft SQL Server
4.	↑ 5.	4.	MongoDB
5.	↓ 4.	5.	PostgreSQL
6.	6.	6.	DB2
7.	7.	↑ 8.	Cassandra
8.	8.	↓ 7.	Microsoft Access
9.	9.	↑ 10.	Redis
10.	10.	↓ 9.	SQLite

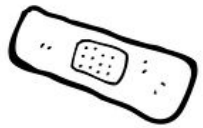
(db-engines.com)

Traditional Databases

Database Design Goals

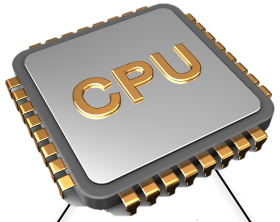
1. Speed 
2. Affordability $
3. Resilience to system failure 

Database Design Goals

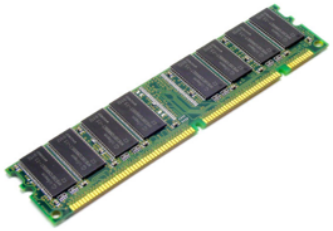
1. Speed 
2. Affordability $
3. Resilience to system failure 

How you achieve them depends on storage hardware

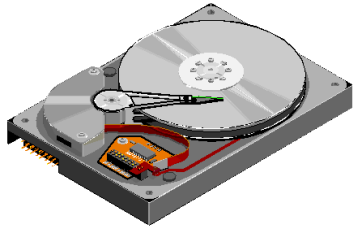
The Memory Hierarchy



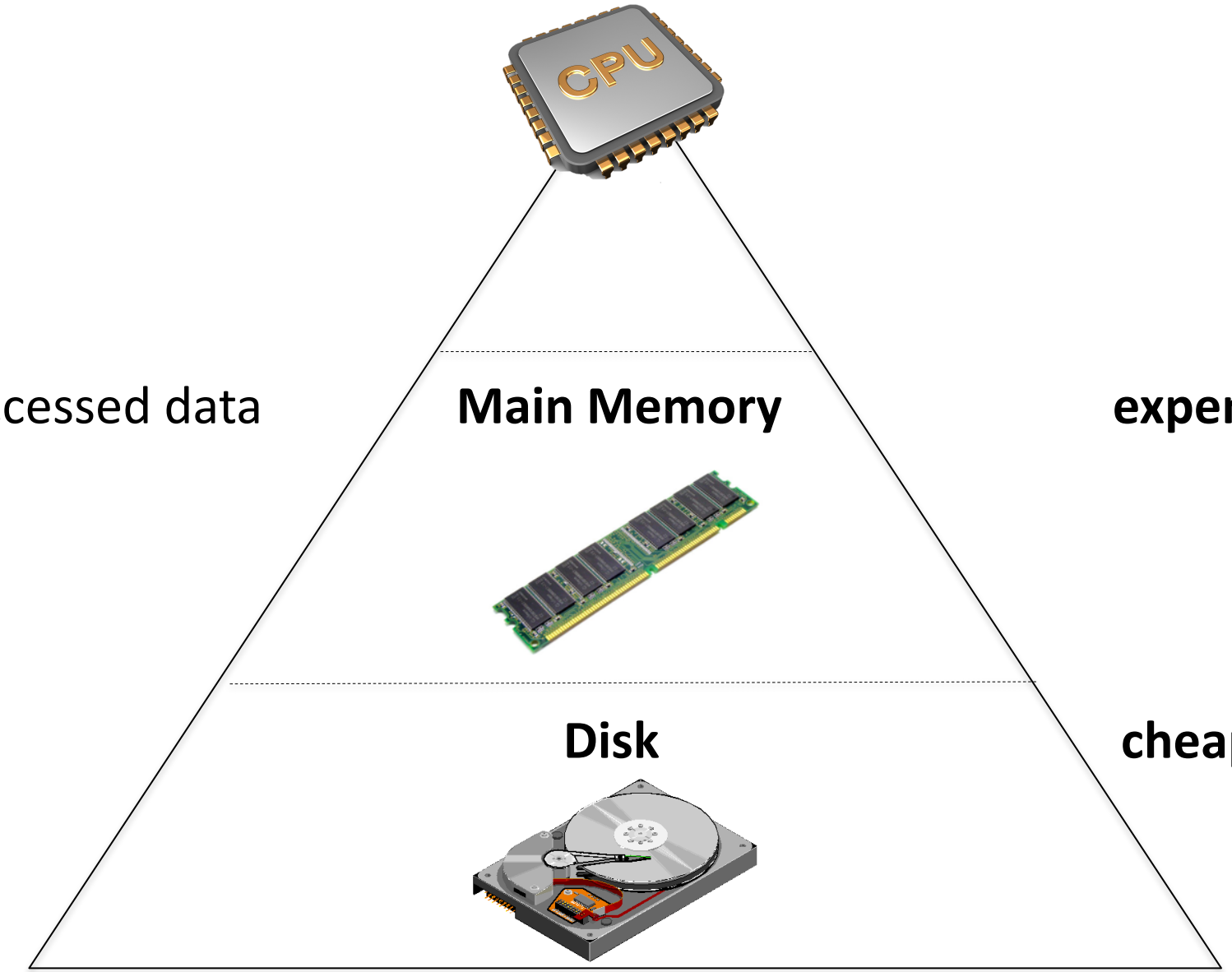
Main Memory — **expensive, fast, volatile**



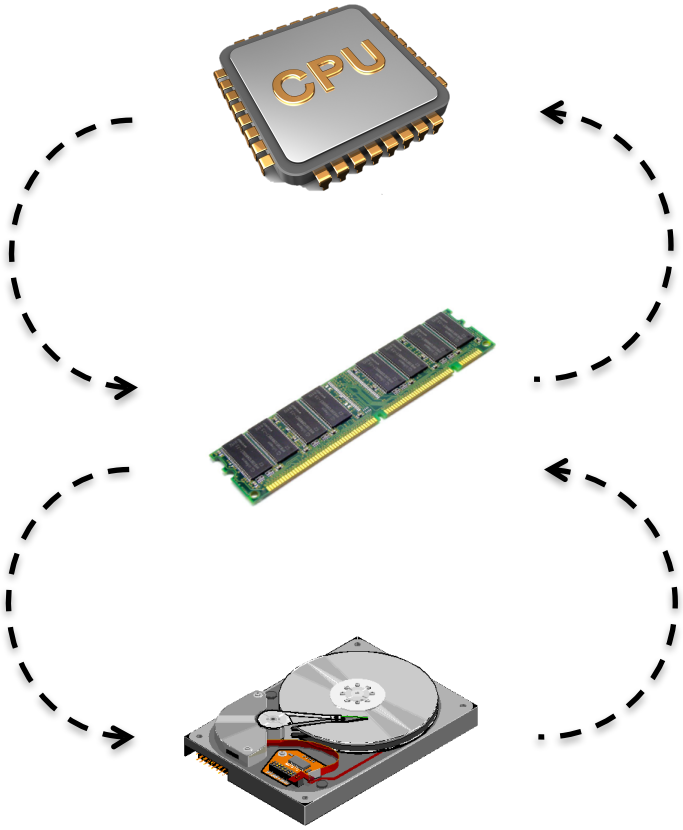
Disk — **cheap, slow, non-volatile**



The Memory Hierarchy



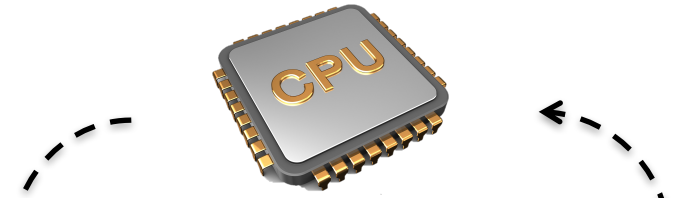
Disk is a bottleneck

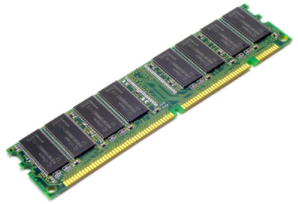


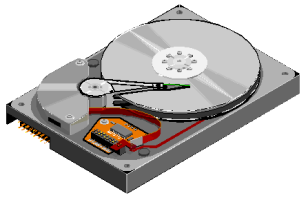
100 nanoseconds

10 milliseconds

Disk is a bottleneck

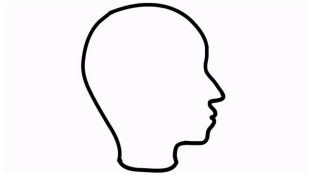




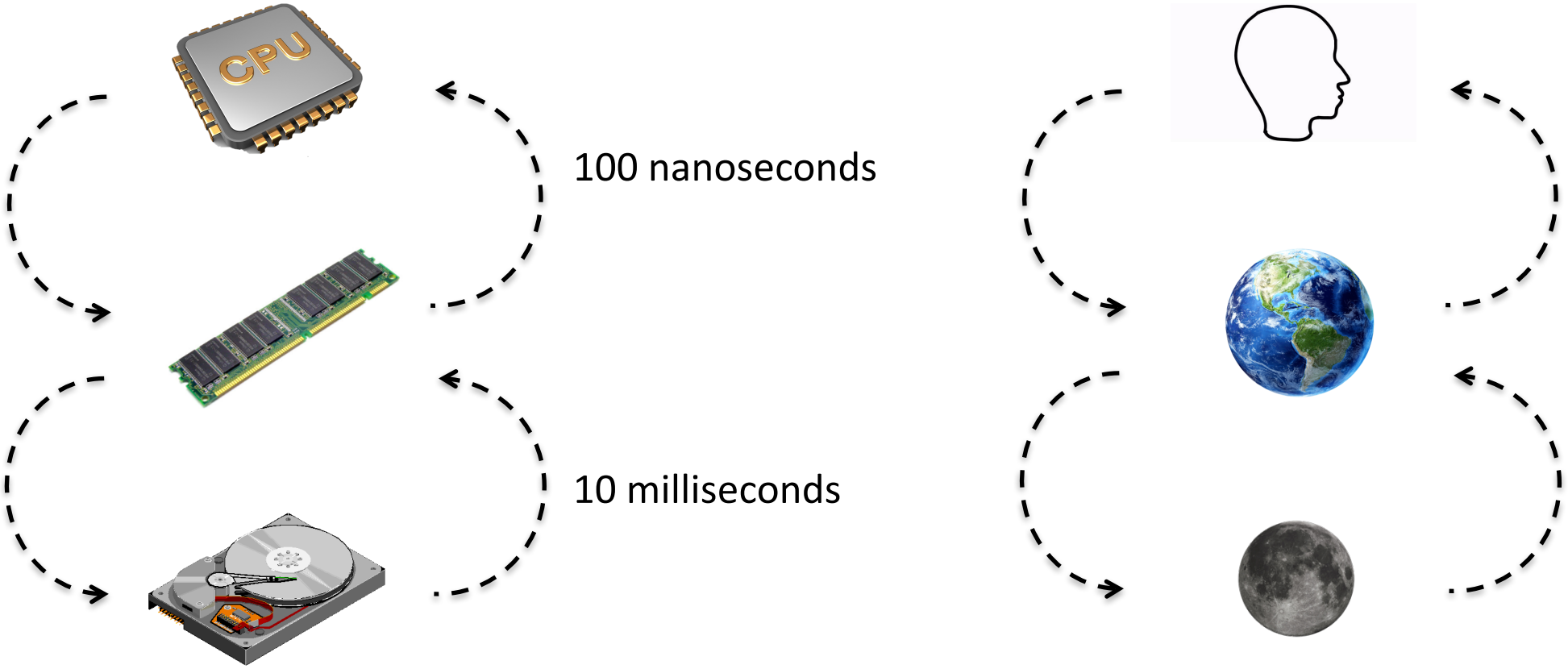


100 nanoseconds

10 milliseconds

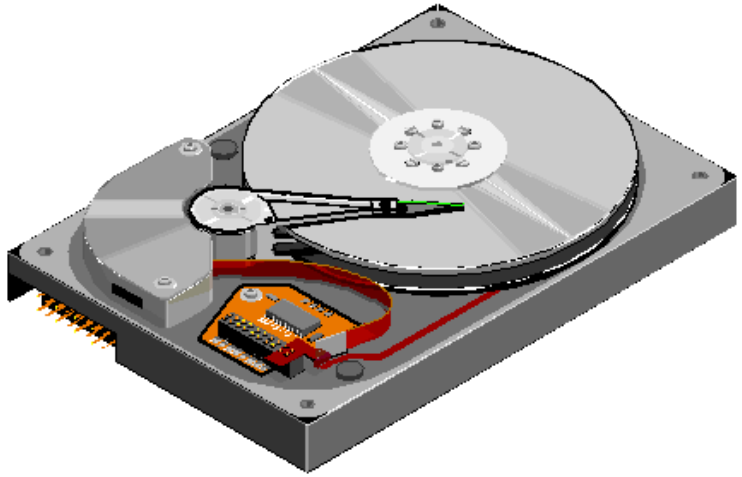


Disk is a bottleneck

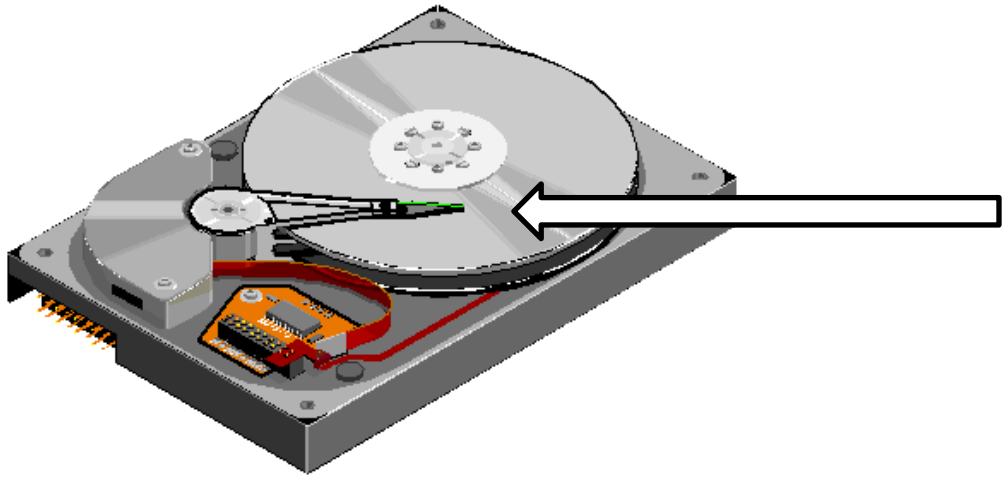


Goal: alleviate disk as the bottleneck

Why is disk slow?

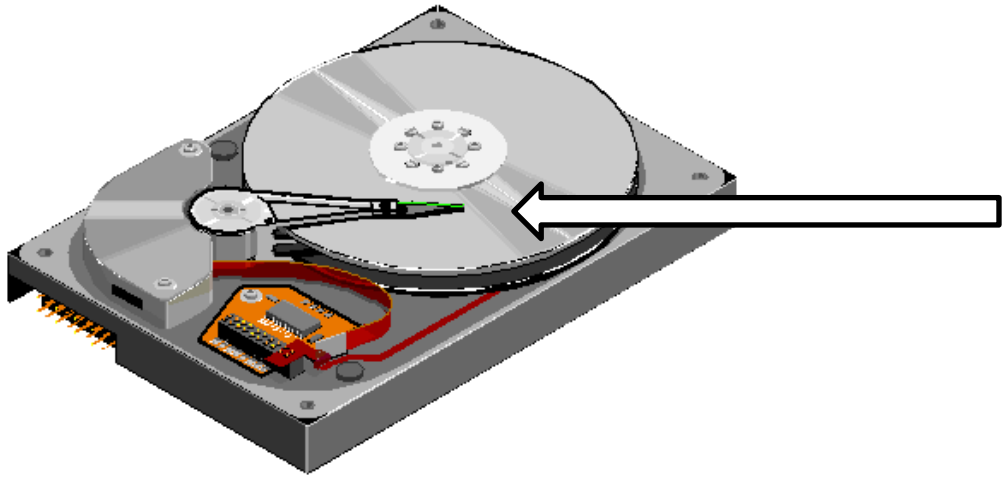


Why is disk slow?



Disk head

Why is disk slow?

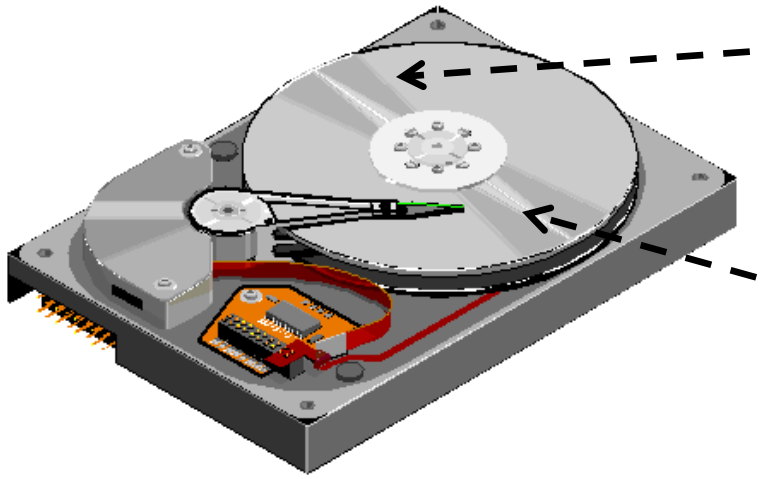


Question 1: How to minimize disk access?

Question 2: What to do during a disk access?

Minimizing Disk Access

Store together data that is frequently co-accessed



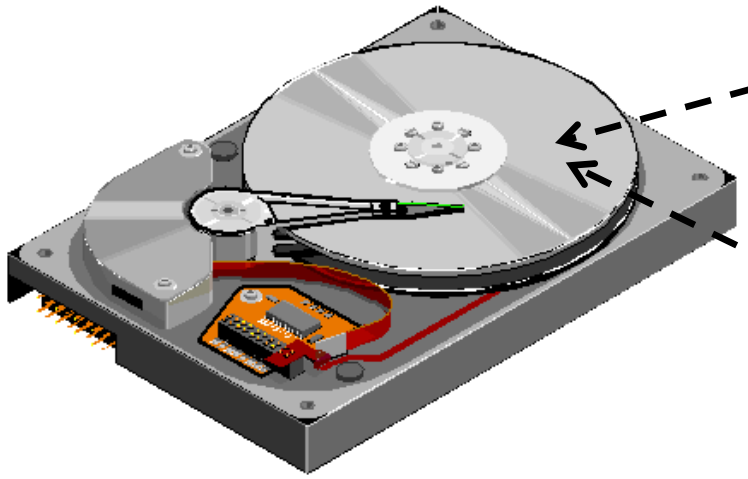
Data item 1

Data item 2

Bad

Minimizing Disk Access

Store together data that is frequently co-accessed



Data item 1

Data item 2

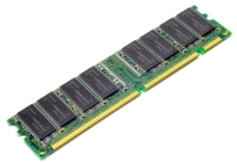
Good

Row-Store

Co-locate all information about data objects in rows

Example: Bank customer Sara deposits $100

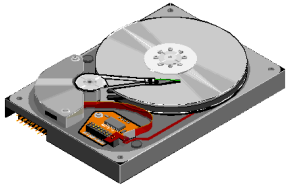
Main Memory



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

2 Sara 200 ← Add 100

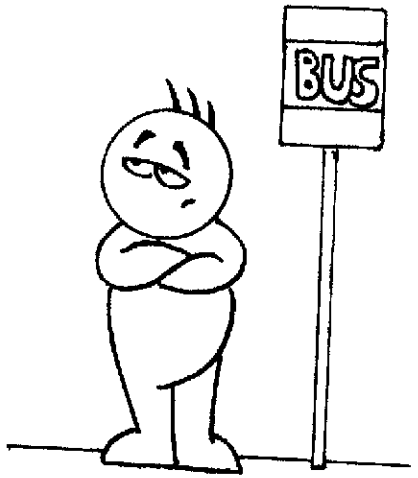
Disk



ID	Name	Balance
1	Bob	100
3	Trudy	450

What to do during a disk access?

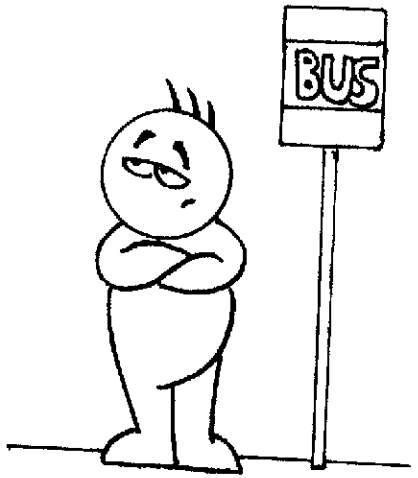
Don’t wait.



Start running the next operation(s)

What to do during a disk access?

Don’t wait.



Start running the next operation(s)

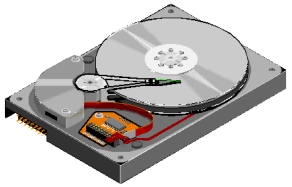
Improves performance

But data can get corrupted

Example of Data Corruption

Bob and Sara share a bank account

Both deposit $100 at same time

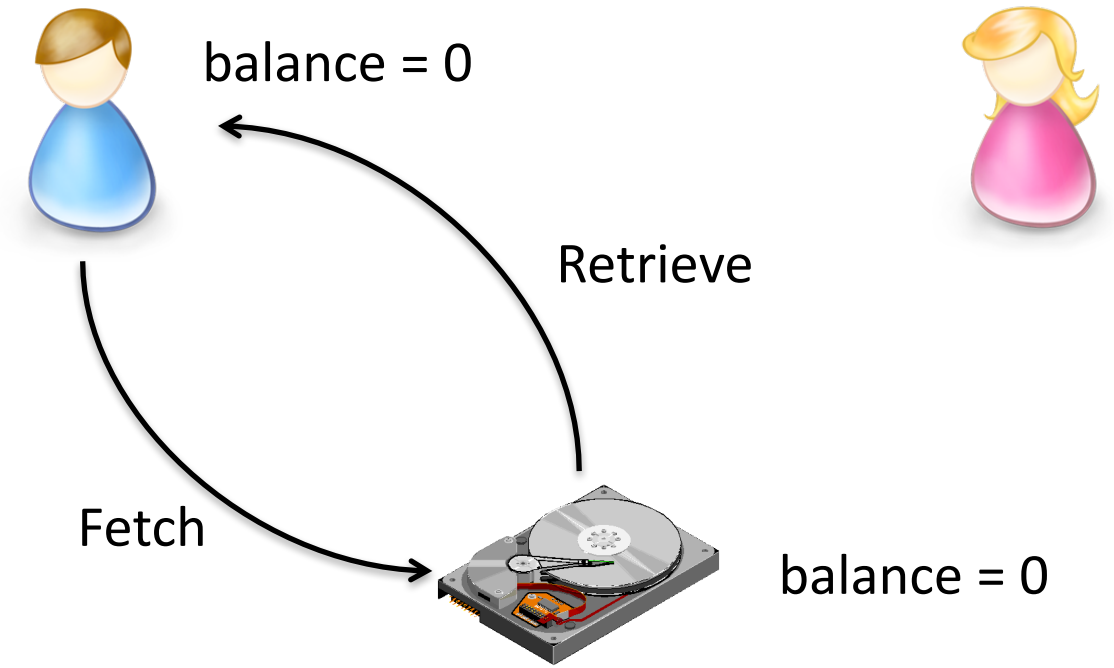


balance = 0

Example of Data Corruption

Bob and Sara share a bank account

Both deposit $100 at same time

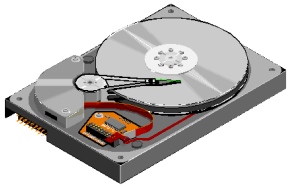


Example of Data Corruption

Bob and Sara share a bank account

Both deposit $100 at same time

balance = 0

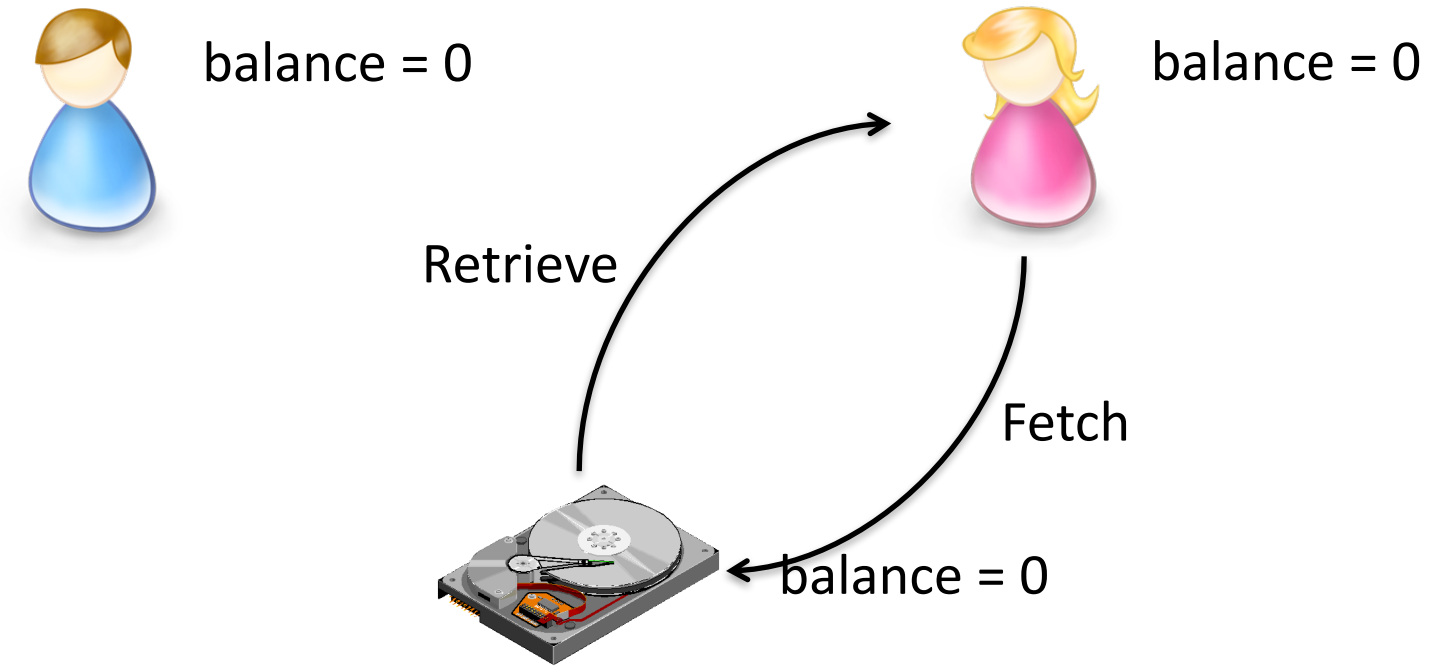


balance = 0

Example of Data Corruption

Bob and Sara share a bank account

Both deposit $100 at same time



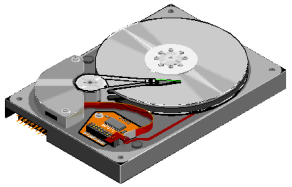
Example of Data Corruption

Bob and Sara share a bank account

Both deposit $100 at same time

balance = 0

balance = 0



balance = 0

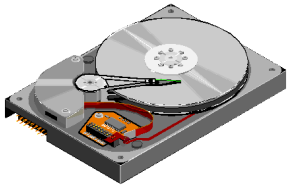
Example of Data Corruption

Bob and Sara share a bank account

Both deposit $100 at same time

balance = 100

balance = 100

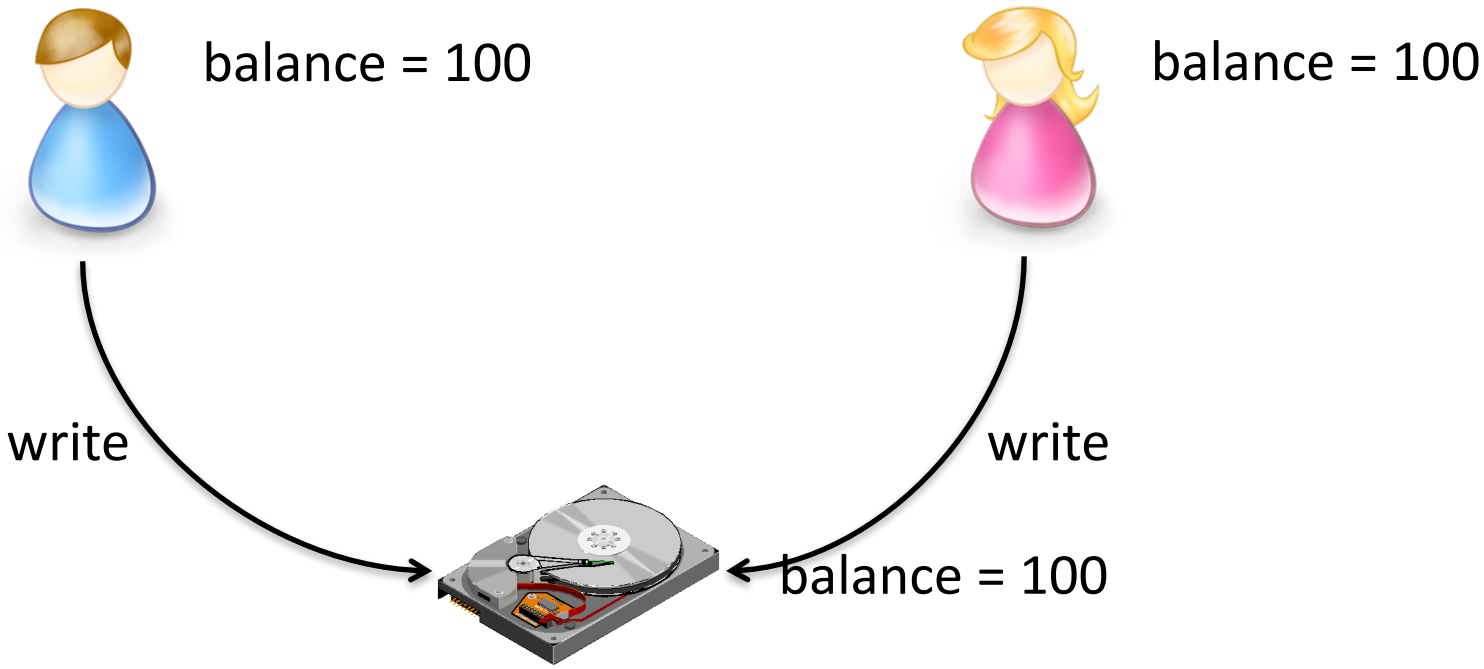


balance = 0

Example of Data Corruption

Bob and Sara share a bank account

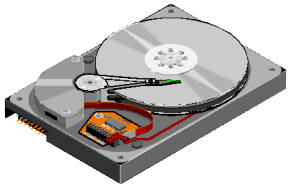
Both deposit $100 at same time



Example of Data Corruption

Bob and Sara share a bank account

Both deposit $100 at same time

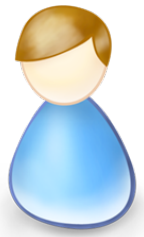


balance = 100

Example of Data Corruption

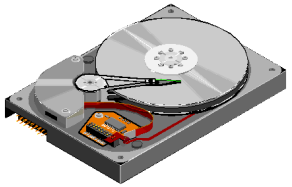
Bob and Sara share a bank account

Both deposit $100 at same time



Account balance should be 200!

Bob and Sara lost money.

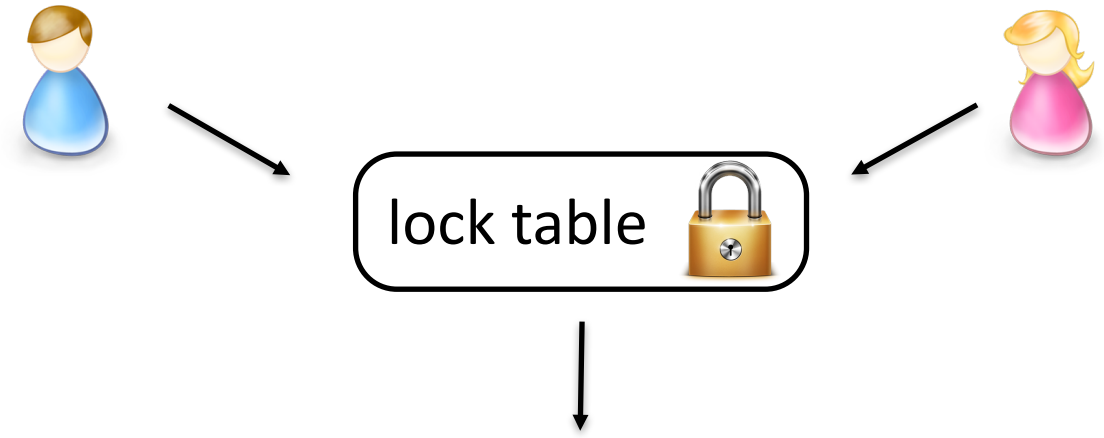


balance = 100

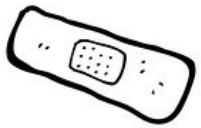
How to avoid data corruption?

Insight: operations can be concurrent, as long as they don't modify the same data

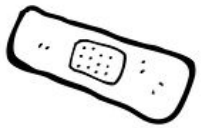
Solution: locking



Database Design Goals

1. **Speed** 
2. Affordability $
3. Resilience to system failure 

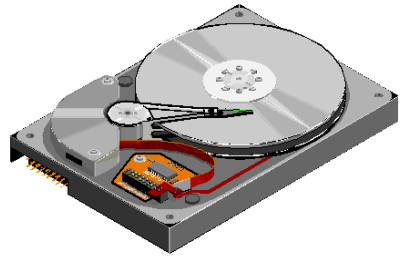
Database Design Goals

1. Speed 
2. **Affordability** **$**
3. Resilience to system failure 

Disk Was Expensive

1 gigabyte for $10000 in 1980

Avoid storing replicas of same data

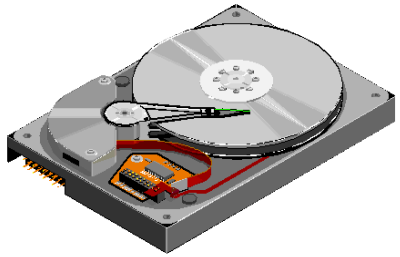


ID	name	account-ID	balance
1	Bob	1	100
2	Sara	1	100
3	Trudy	2	450

Disk Was Expensive

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

Customers

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

Accounts

ID	balance
1	100
2	450

Normalization

Saves storage space 

Helps data integrity 

Reads take longer 

Customers

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

Accounts

ID	**balance**
1	100
2	450

SQL

Getting Bob's account balance:

```
select balance
from Customers c, Accounts a
where c.account-ID = a.ID and c.name = "Bob"
```

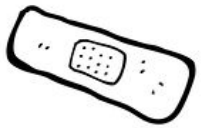
Customers

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

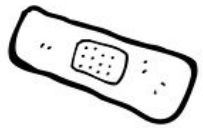
Accounts

ID	**balance**
1	100
2	450

Database Design Goals

1. Speed 
2. **Affordability** **$**
3. Resilience to system failure 

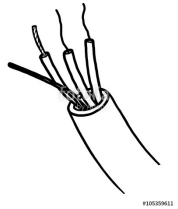
Database Design Goals

1. Speed 
2. Affordability $
3. **Resilience to system failure** 

Possible Failures

Power failure

Hardware failure



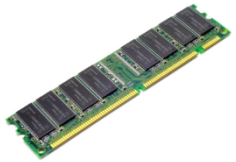
Natural disaster

Data is precious (e.g. bank)

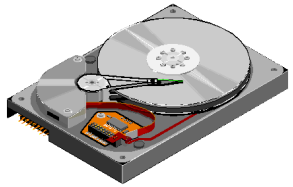
Provide recovery mechanism

Power Failure

Sara transfers $100 to Trudy



Sara's
balance = 0

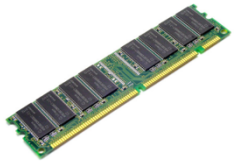


Sara's
balance = 100

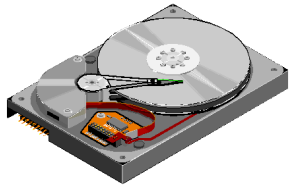
Trudy's
balance = 450

Power Failure

Sara transfers $100 to Trudy



Sara's
balance = 0

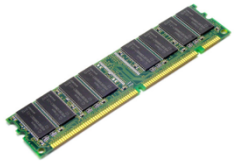


Sara's
balance = 100

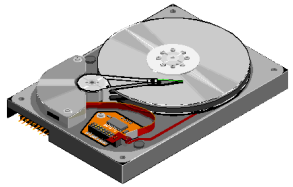
Trudy's
balance = 450

Power Failure

Sara transfers $100 to Trudy



Trudy's
balance = 550



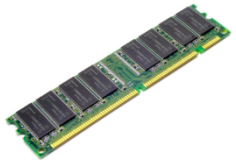
Sara's
balance = 0

Trudy's
balance = 450

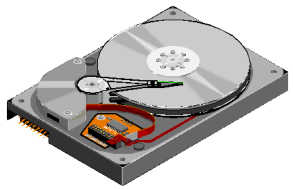
Power Failure

Sara transfers $100 to Trudy

power fails



Trudy's
balance = 550



Sara's
balance = 0

Trudy's
balance = 450

Transactions

Atomic sequence of operations (all or nothing)

full transaction {

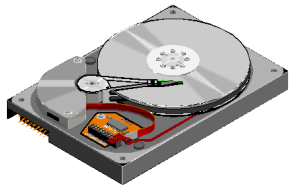
- Subtract 100 from Sara
- Add 100 to Trudy

Transactions

Atomic sequence of operations (all or nothing)

full transaction

Subtract 100 from Sara
Add 100 to Trudy



Name	balance
Sara	100
Trudy	450

Data

Changes

Log

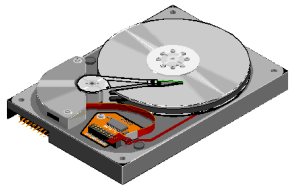
Transactions

Atomic sequence of operations (all or nothing)

full transaction

Subtract 100 from Sara

Add 100 to Trudy



Name	balance
Sara	100
Trudy	450

Data

Changes

Log

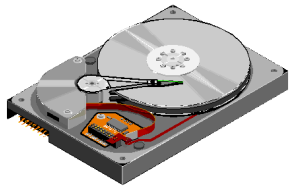
Transactions

Atomic sequence of operations (all or nothing)

full transaction

Subtract 100 from Sara

Add 100 to Trudy



Name	balance
Sara	0
Trudy	450

Data

Changes
Sara -100

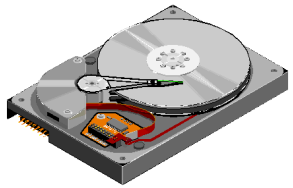
Log

Transactions

Atomic sequence of operations (all or nothing)

full transaction

Subtract 100 from Sara
Add 100 to Trudy



Name	balance
Sara	0
Trudy	450

Data

Changes
Sara -100

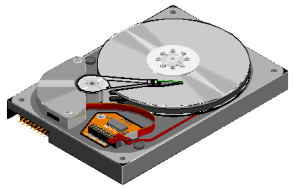
Log

Transactions

Atomic sequence of operations (all or nothing)

full transaction

Subtract 100 from Sara
Add 100 to Trudy



Name	balance
Sara	0
Trudy	450

Data

Changes
Sara -100

Log

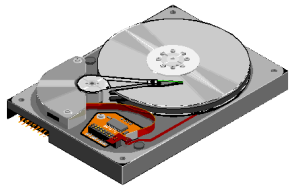
After failure, undo changes by unfinished transactions

Transactions

Atomic sequence of operations (all or nothing)

full transaction

- Subtract 100 from Sara
- Add 100 to Trudy



Name	balance
Sara	100
Trudy	450

Data

Changes
Sara -100

Log

After failure, undo changes by unfinished transactions

ACID

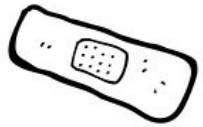
Data integrity pitfalls

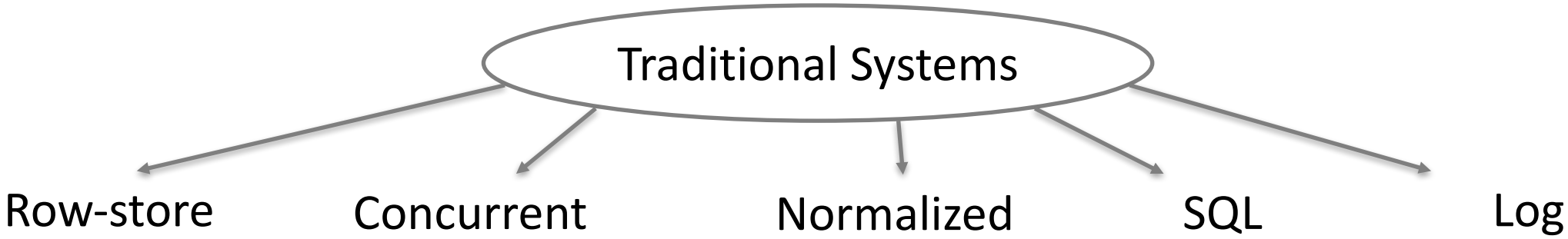
- Concurrency (fix with locking)
- Power failure (fix with logging)

Transactions should be

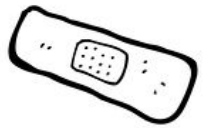
- **A**tomic
- **C**onsistent
- **I**solated
- **D**urable

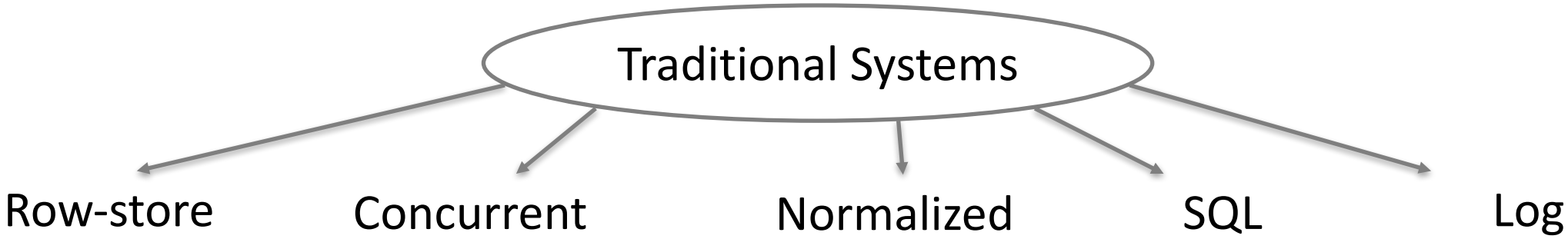
Database Design Goals

1. Speed 
2. Affordability $
3. Resilience to system failure 



Database Design Goals

1. Speed 
2. Affordability $
3. Resilience to system failure 



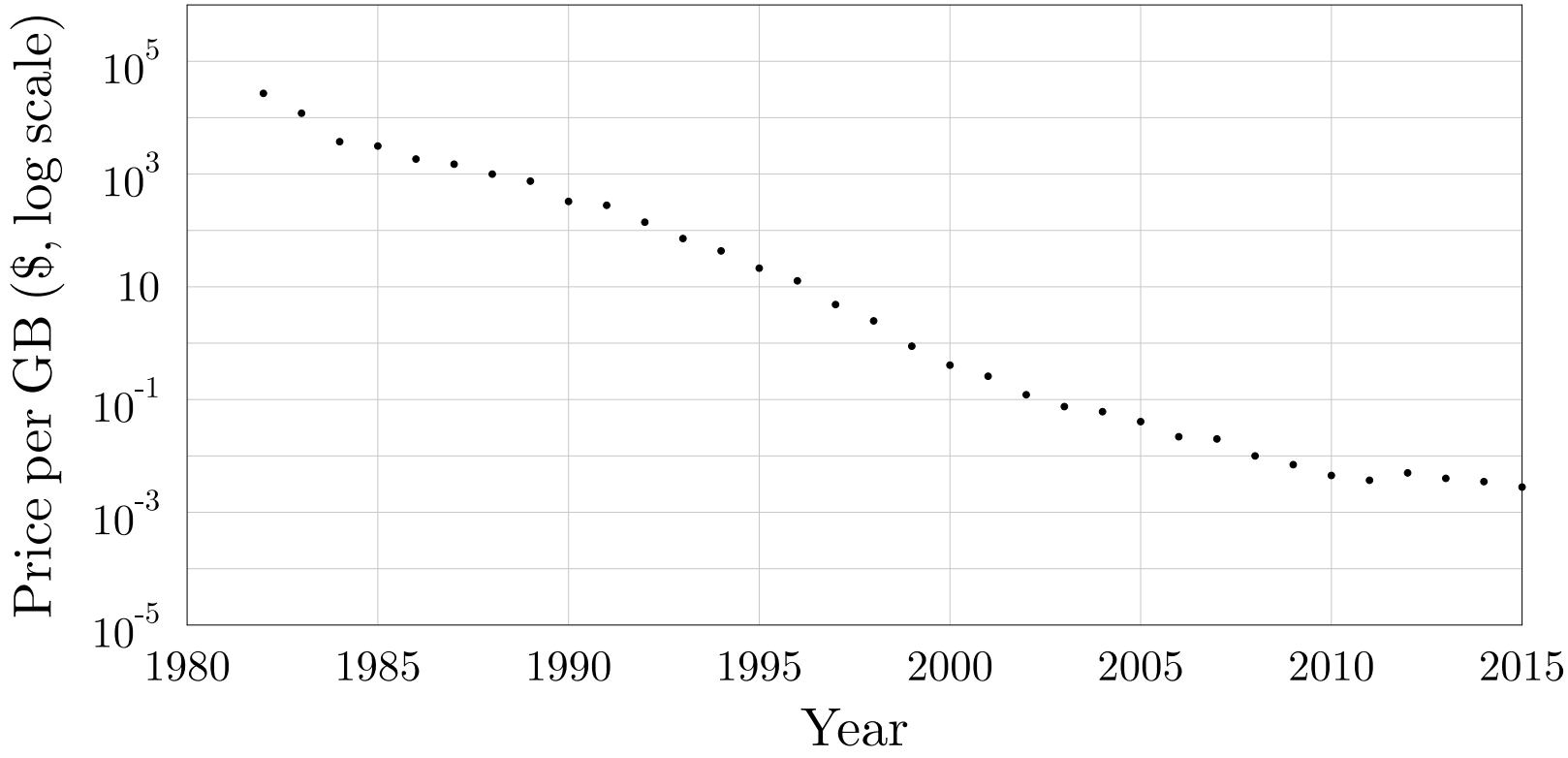
Everything is Motivated by hardware.

BigData Challenges

What changed in hardware?

Why can’t traditional systems cope?

Declining Storage Costs



10^{7} x cheaper

Problem 1: Analyzing Big Data

Everyone is hording data

End users

transactions

Transactional Database

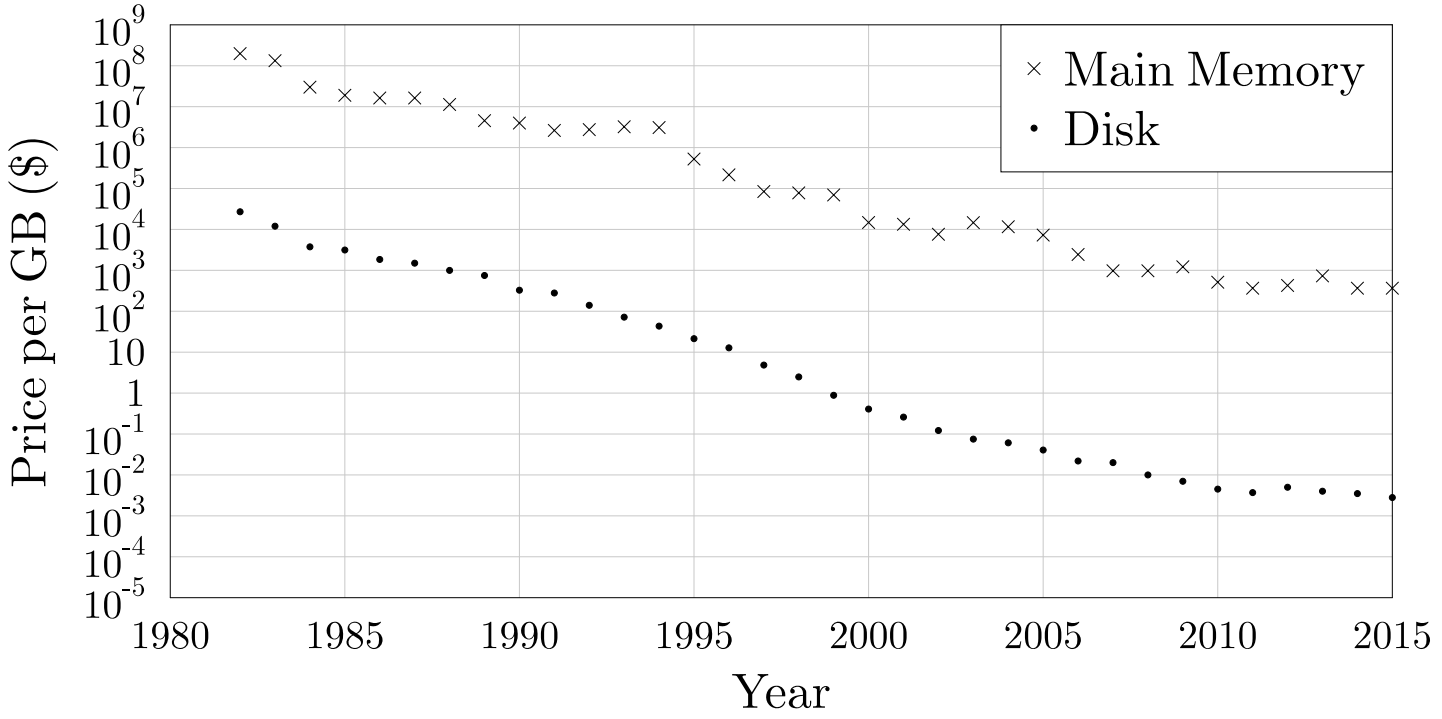
Analytical queries

Business Intelligence

Analytical queries are expensive

Worsens as data grows

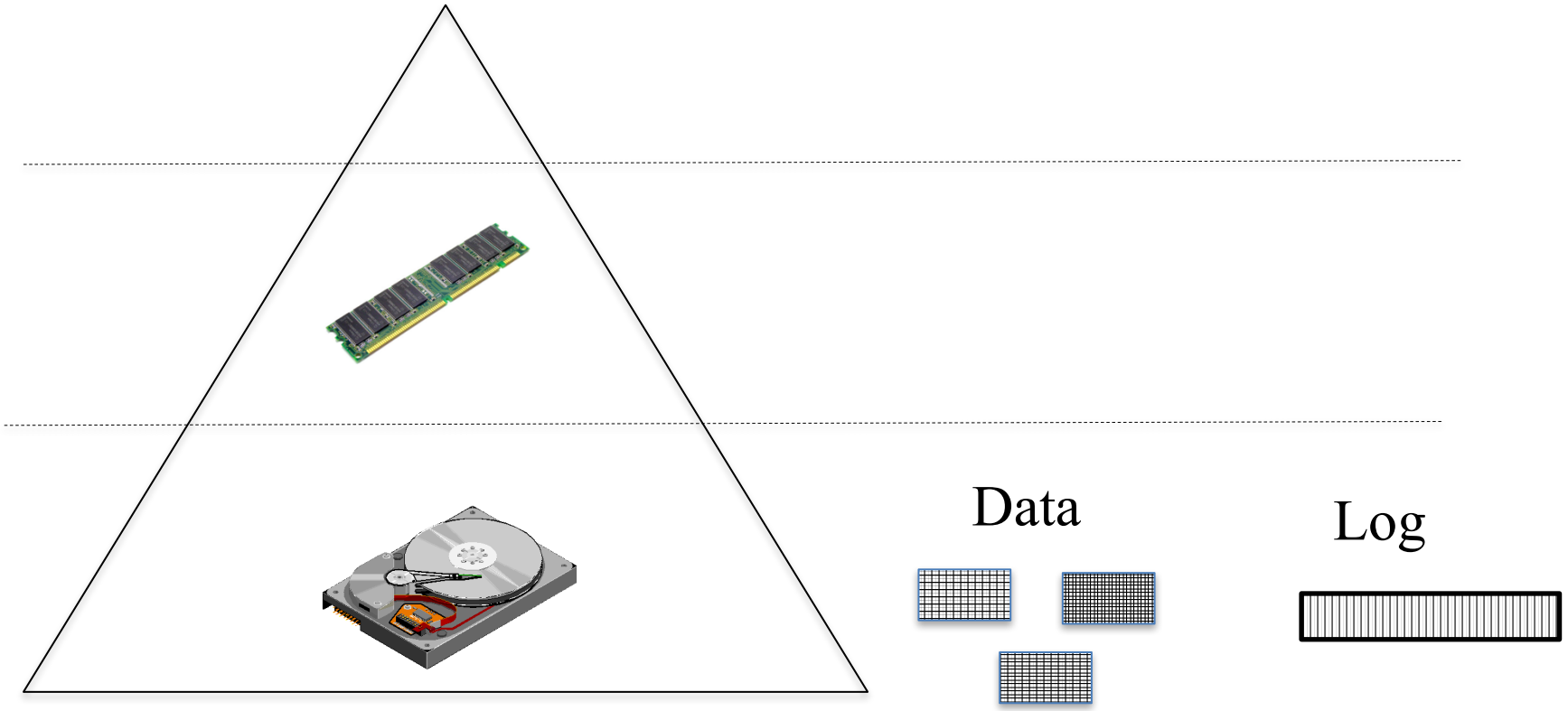
Declining Main Memory Costs



Main memory: 10^6 x cheaper

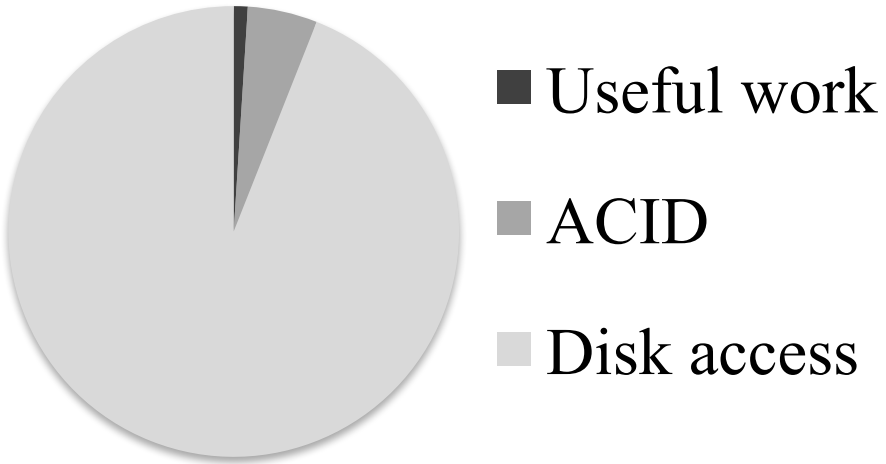
Disk: 10^7 x cheaper

Some Databases Fit in Memory



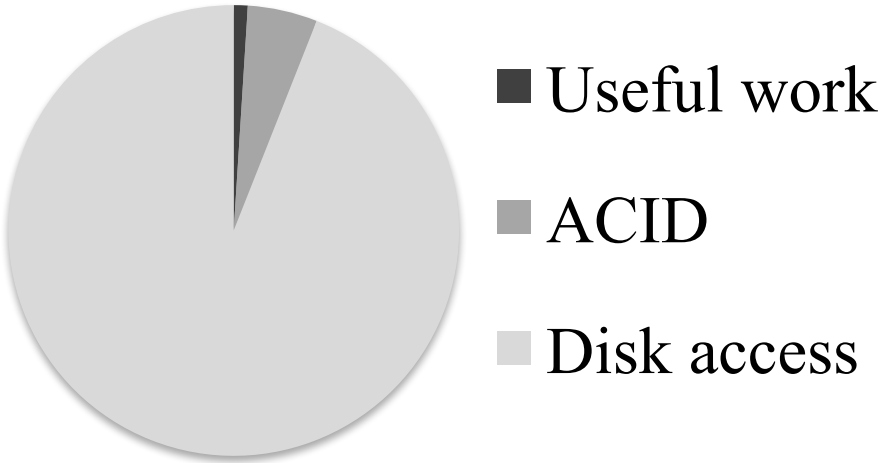
Problem 2: ACID is a new Bottleneck

Disk-based system

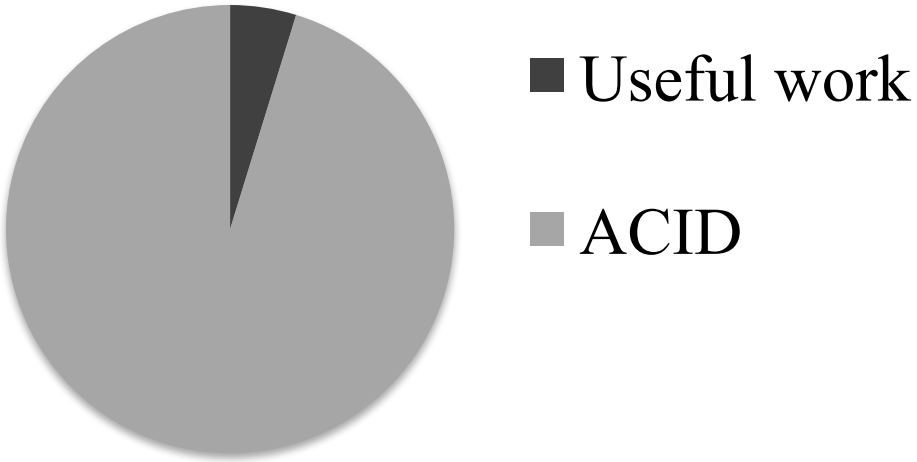


Problem 2: ACID is a new Bottleneck

Disk-based system



Memory-based system

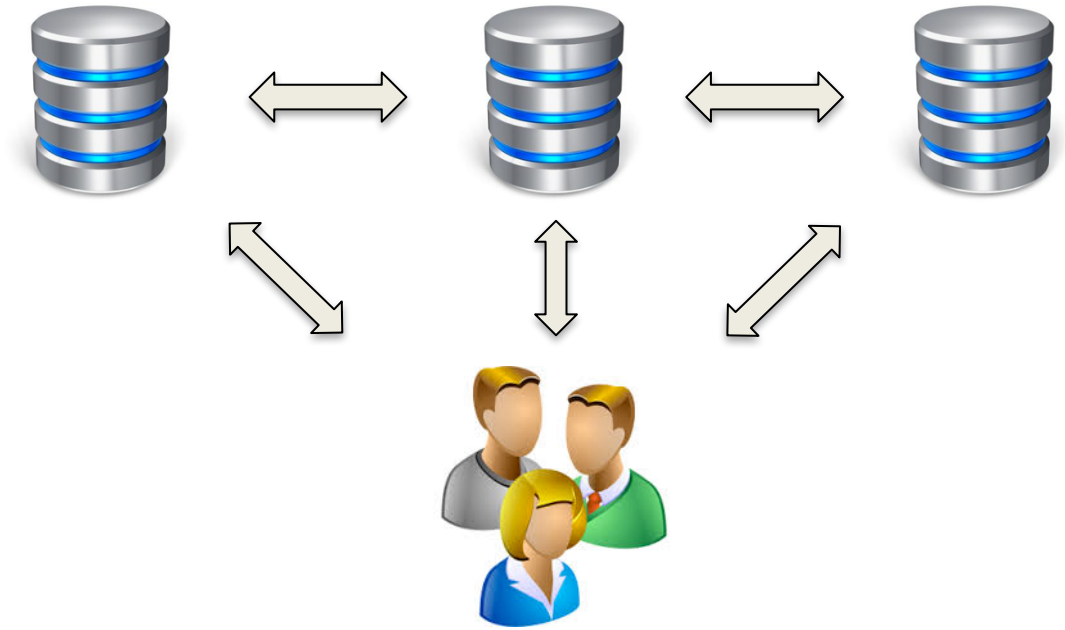


Locking, logging

Distribution Across Machines

Increase bandwidth (partitioning)

Remain available if one machine fails (replicating)



Problem 3: Consistency

Updates require synching across network

Accounts

ID	balance
1	100
2	450

Accounts

ID	balance
1	100
2	450

Problem 4: Joins

Take longer as tables grow

E.g. find Bob's account balance

Customers

ID	**name**	account-ID
	…	
2	Bob	X
	…	

Accounts

ID	**balance**
	…
X	100
	…

Problems

1. Analytical Queries
2. ACID
3. Consistency
4. Joins

Problems

1. Analytical Queries
2. ACID
3. Consistency
4. Joins

Solutions

Analytical Databases

NoSQL Databases

NewSQL Databases

Problems

1. **Analytical Queries**
2. ACID
3. Consistency
4. Joins

Solutions

Analytical Databases

NoSQL Databases

NewSQL Databases

Analytical Databases

End users

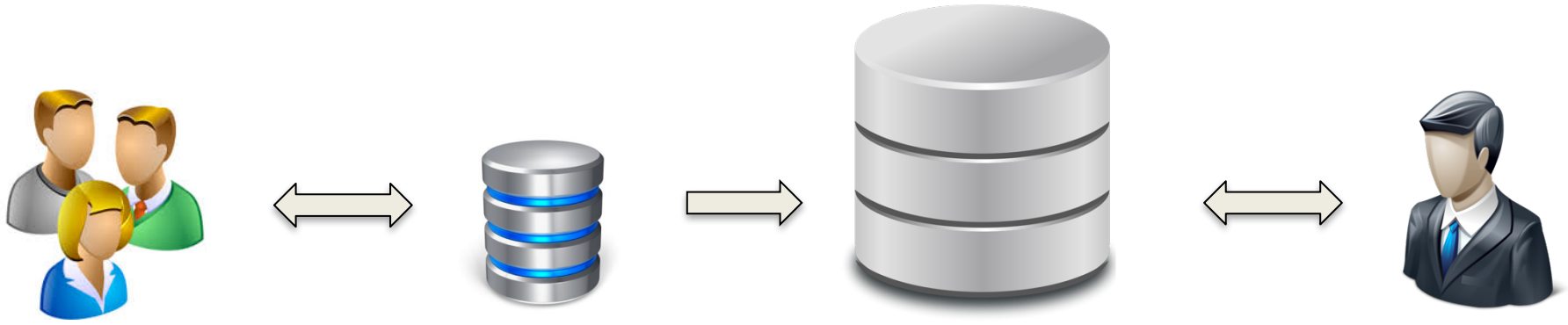
transactions

Transactional Database

Analytical queries

Business Intelligence

Analytical Databases



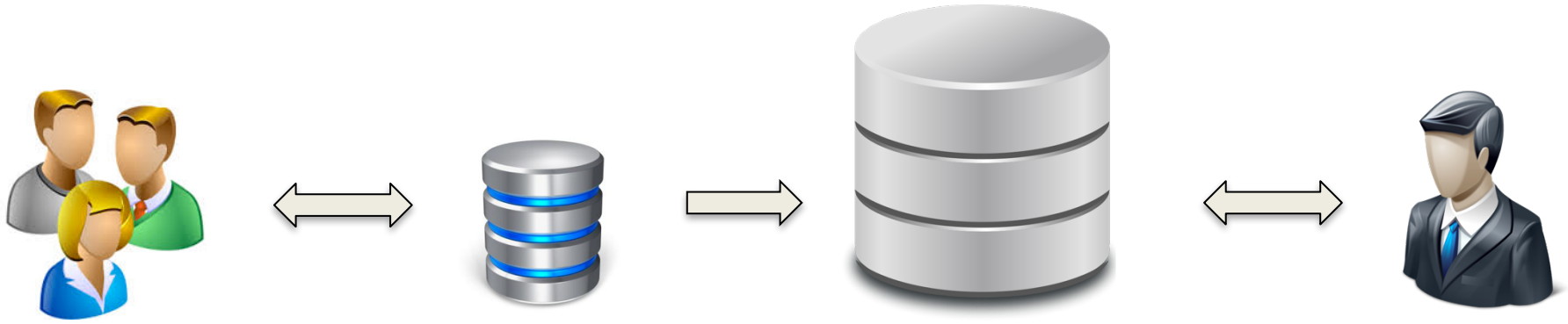
End users

Transactional Database

Analytical Database

Business Intelligence

Analytical Databases



Different internal design for analytics

Analytical Queries Examples

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

How long is delivery?

Revenue from product X?

Few columns, many rows

Problem: Data is stored row by row

Column-Stores

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

Each column is stored separately

Changes entire architecture

Examples: Vertica, Vectorwise, etc.

Problems

1. **Analytical Queries**
2. ACID
3. Consistency
4. Joins

Solutions

Analytical Databases

NoSQL Databases

NewSQL Databases

Problems

1. Analytical Queries
2. **ACID**
3. **Consistency**
4. **Joins**

Solutions

Analytical Databases

NoSQL Databases

NewSQL Databases

NoSQL

Named in 2009

Hashtag to advertise a conference

NoSQL

Problems

1. Joins
2. ACID
3. Consistency

NoSQL

Problems

1. Joins
2. ACID
3. Consistency

} De-normalization

Normalization Recap


Reduce data volume


Introduces joins


Locking & logging across

People

ID	name	addr-ID
1	Sara	1
2	Bob	1
3	Will	2

Addresses

ID	addr
1	33 Oxford st
2	45 Harvard st

Normalization Recap


Reduce data volume — smaller issue


Introduces joins — bigger issue


Locking & logging across — bigger issue

People

ID	name	addr-ID
1	Sara	1
2	Bob	1
3	Will	2

Addresses

ID	addr
1	33 Oxford st
2	45 Harvard st

Normalization Recap

Reduce data volume — smaller issue

Introduces joins — bigger issue

Locking & logging across — bigger issue

People

ID	name	addr-ID
1	Sara	1
2	Bob	1
3	Will	2

Addresses

ID	addr
1	33 Oxford st
2	45 Harvard st

NoSQL →

People

ID	name	Addr-ID
1	Sara	33 Oxford st
2	Bob	33 Oxford st
3	Will	45 Harvard st

Document-store

Each object is a document. Flexible format.

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

```
{ name: "Sara",
  balance:         100
  hobbies: ["rowing", "running"]
}
```

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

Downside: costly to search a document

Transaction spans Single Document

Cannot maintain integrity guarantees for changes across documents

Non-interacting objects

- No sharing (e.g. bank account)
- No exchanging (e.g. money transfers)

NoSQL

Problems

1. **Joins**
2. **ACID**
3. Consistency

} **De-normalization**

NoSQL

Problems

1. Joins
2. ACID

} De-normalization

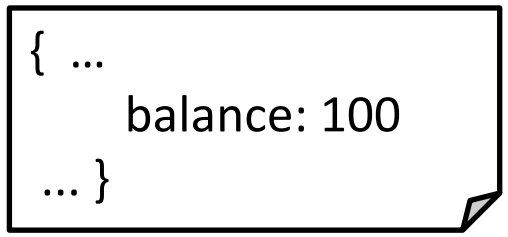
3. **Consistency** } **Relax**

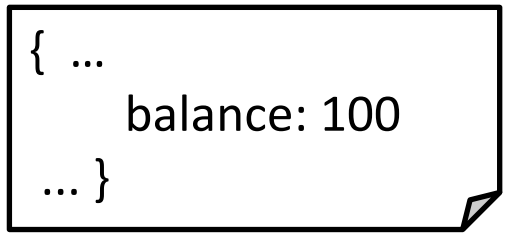
Consistency

Different operation order across replicas

Deposit
Add 100

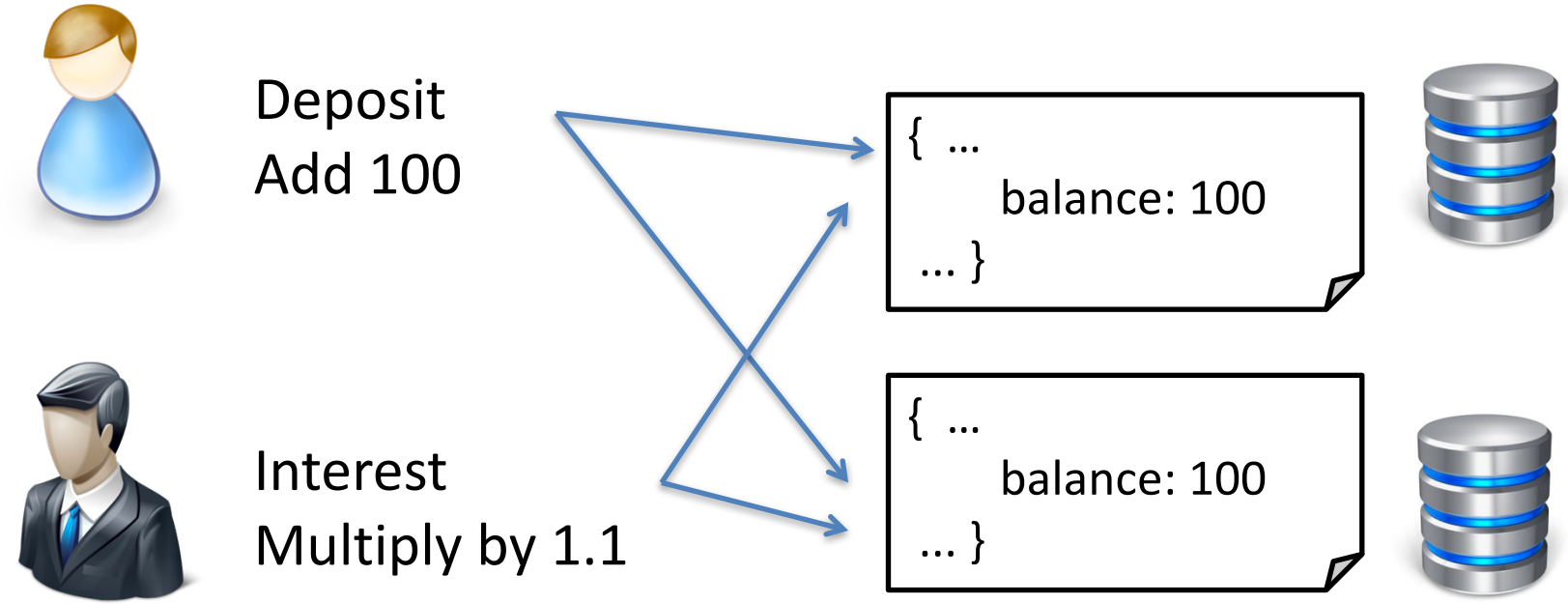
Interest
Multiply by 1.1





Consistency

Different operation order across replicas



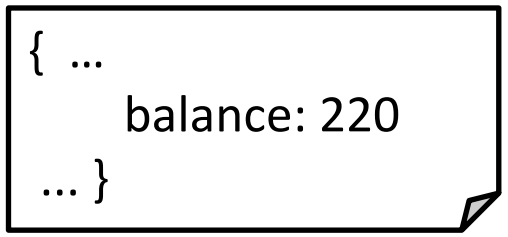
Consistency

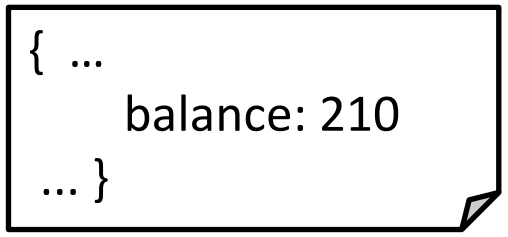
Different operation order across replicas

Deposit
Add 100

Inconsistent replicas

Interest
Multiply by 1.1





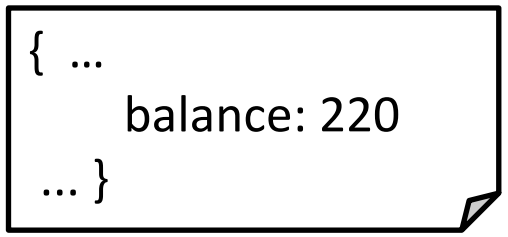
Consistency

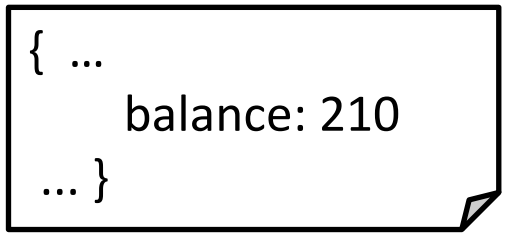
Different operation order across replicas

Deposit
Add 100

Inconsistent replicas

Interest
Multiply by 1.1





Eventually inconsistencies are fixed.

NoSQL

Problems

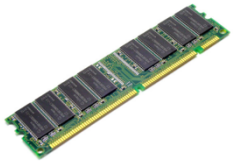
1. Joins
2. ACID

} De-normalization

3. **Consistency** } **Eventual-consistency**

Two NoSQL Types

Main memory



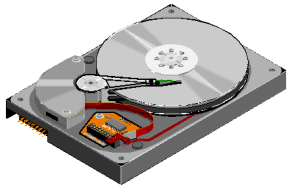
redis

TARANTOOL

MEMCACHED

Couchbase

Disk



HBASE

RocksDB

levelDB

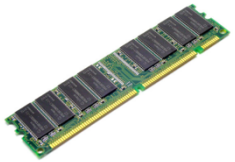
cassandra

mongoDB.

accumulo™

Two NoSQL Types

Main memory



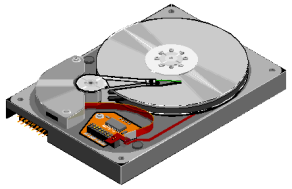
redis

TARANTOOL

MEMCACHED

Couchbase

Disk



HBASE

RocksDB

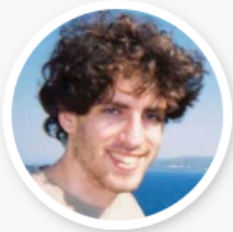
levelDB

cassandra

mongoDB.

accumulo™

My Area!



Niv Dayan

When to use NoSQL?

Non-interacting data objects

Weak consistency

Flexible data model

Quick & dirty & easy

Problems

1. Analytical Queries
2. **ACID**
3. **Consistency**
4. **Joins**

Solutions

Analytical Databases

NoSQL Databases

NewSQL Databases

Problems

1. Analytical Queries
2. **ACID**
3. **Consistency**
4. **Joins**

Solutions

Analytical Databases

NoSQL Databases

NewSQL Databases

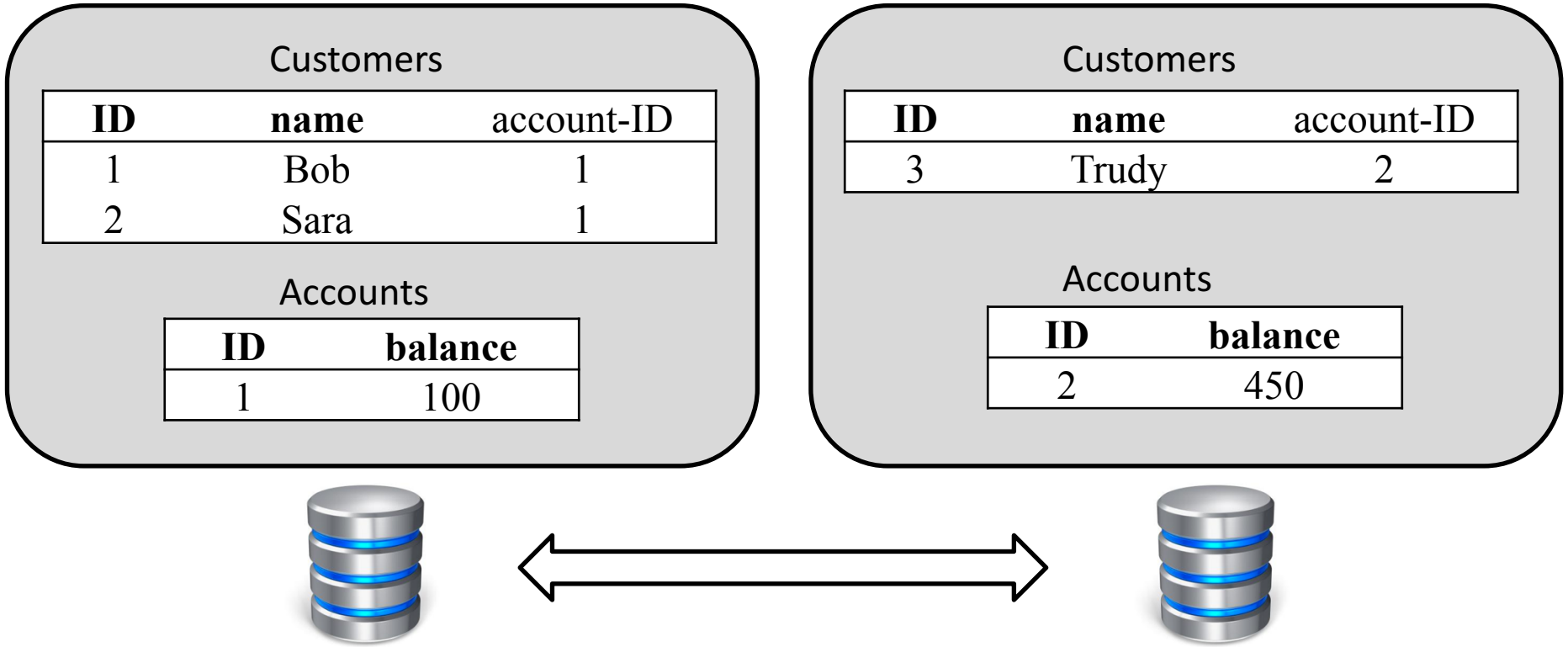
Address without compromising generality of application

e.g. applications involving money 

NewSQL

Data is normalized

Tables partitioned and replicated across machines



NewSQL

Problems

1. ACID
2. Consistency
3. Joins

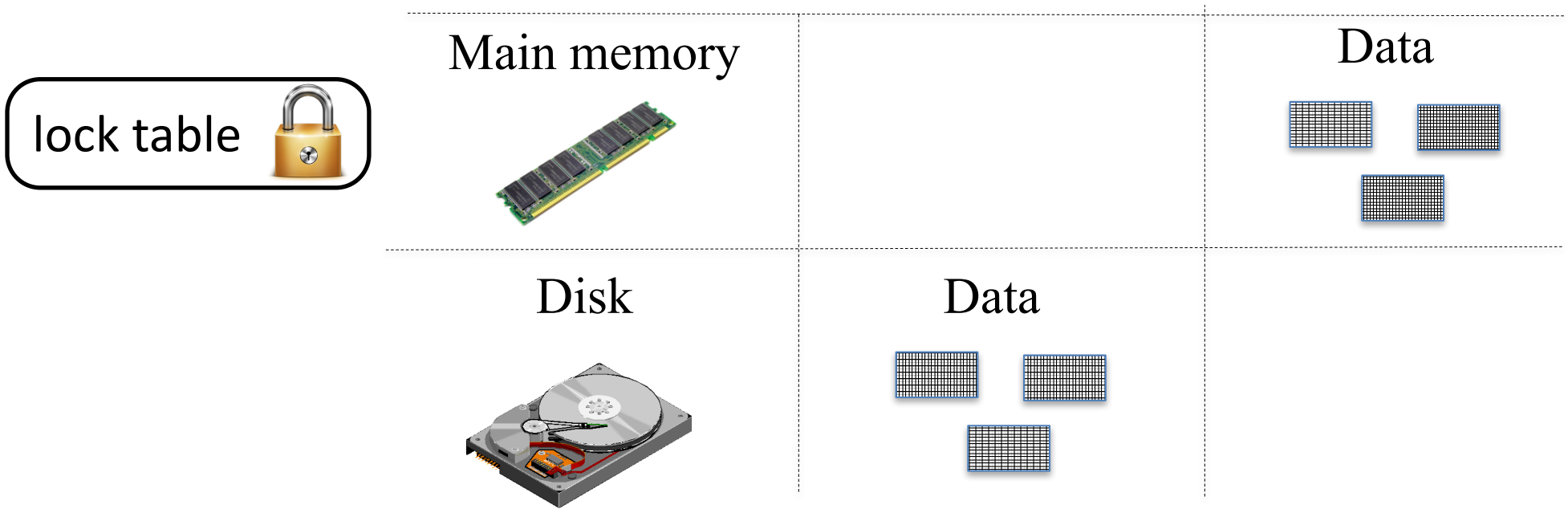
} Reengineering

Locking

concurrency introduced due to disk

Today data is in main memory

Remove concurrency and locking. Serial transactions

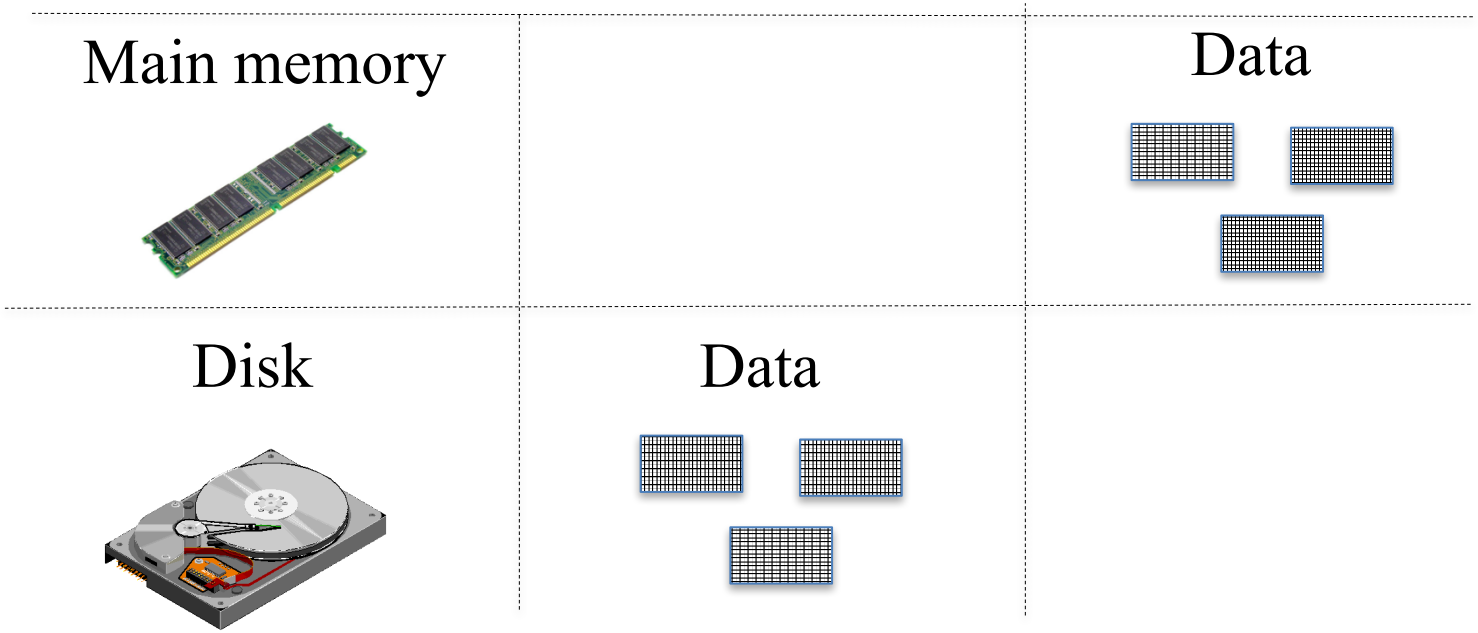


Locking

concurrency introduced due to disk

Today data is in main memory

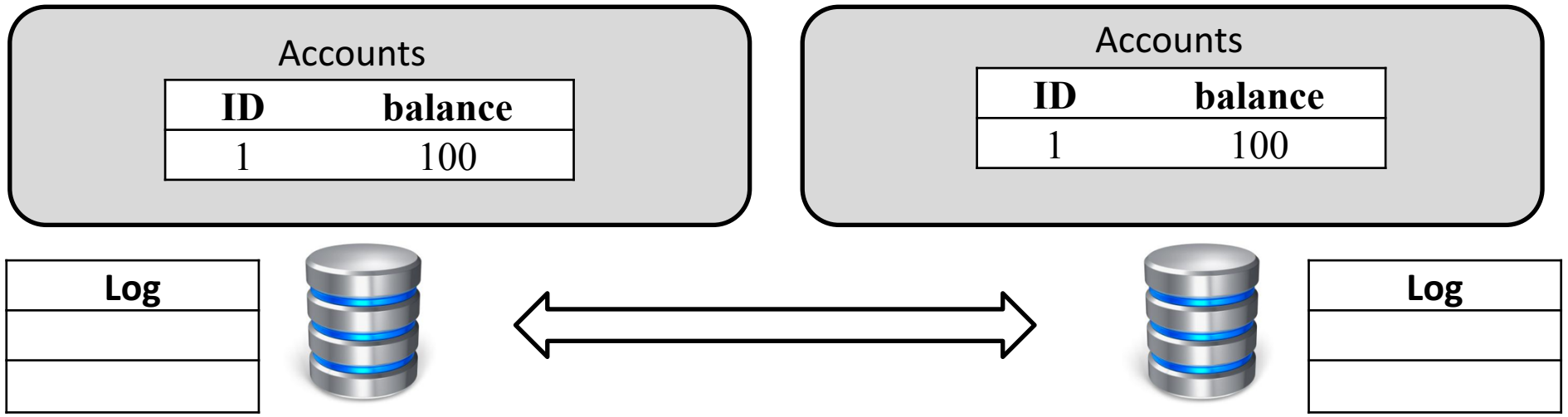
Remove concurrency and locking. Serial transactions



Logging

History: log introduced for recovery

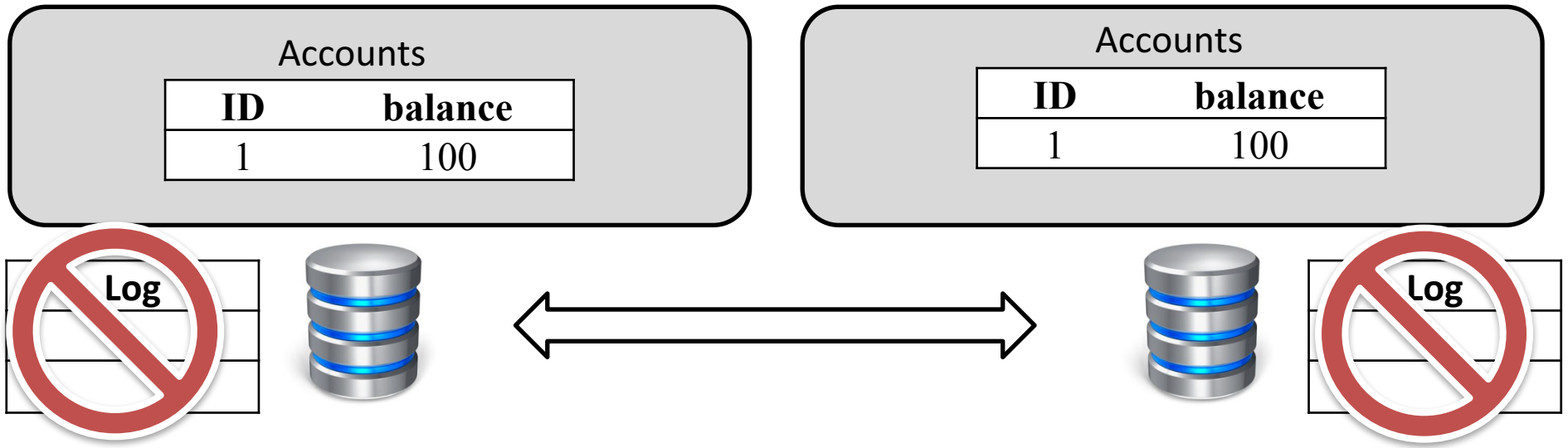
Today: replicas exist elsewhere



Logging

History: log introduced for recovery

Today: replicas exist elsewhere



NewSQL

Problems

1. **ACID** } **Reengineering**
2. Consistency
3. Joins

NewSQL

Problems

1. ACID } Reengineering
2. **Consistency** } **Tough luck**
3. Joins

NewSQL

Enforce operation order across replicas (expensive)

Deposit
Add 100

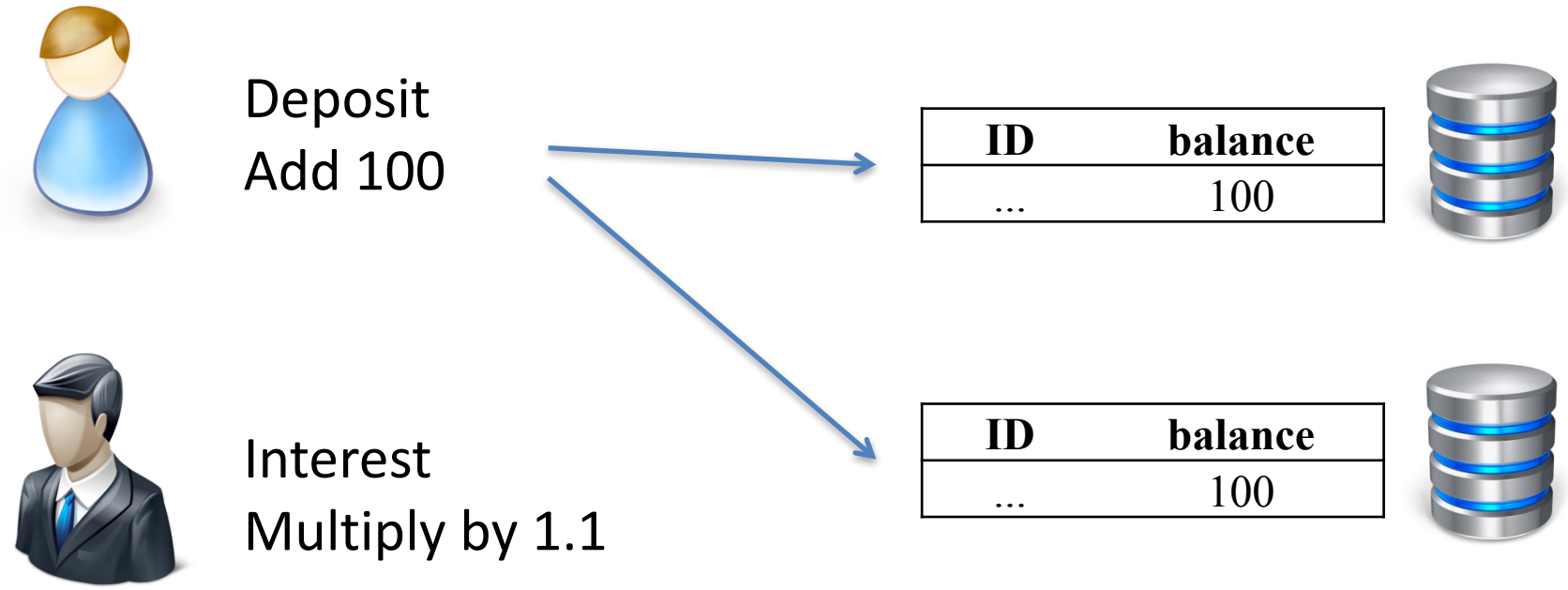
ID	balance
...	100

Interest
Multiply by 1.1

ID	balance
...	100

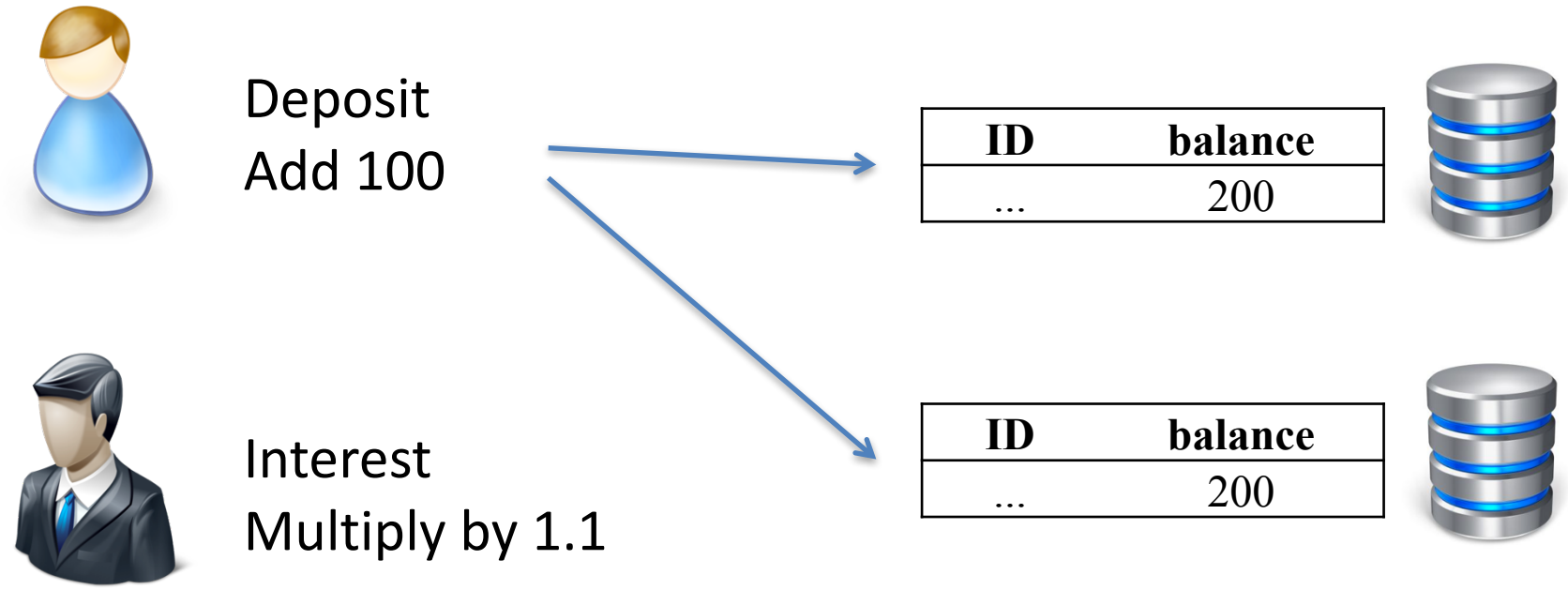
NewSQL

Enforce operation order across replicas (expensive)



NewSQL

Enforce operation order across replicas (expensive)



NewSQL

Enforce operation order across replicas (expensive)

Deposit
Add 100

ID	balance
...	200

Interest
Multiply by 1.1

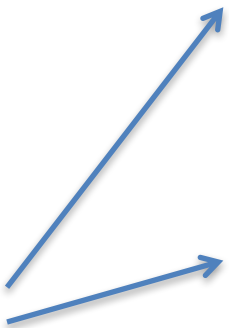
ID	balance
...	200

NewSQL

Enforce operation order across replicas (expensive)

Deposit
Add 100

Interest
Multiply by 1.1



ID	balance
...	200

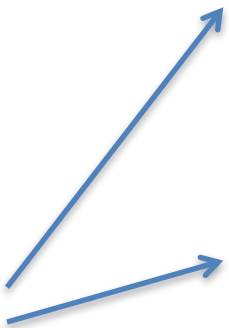
ID	balance
...	200

NewSQL

Enforce operation order across replicas (expensive)

Deposit
Add 100

Interest
Multiply by 1.1



ID	balance
...	220

ID	balance
...	220

NewSQL

Enforce operation order across replicas (expensive)

Deposit
Add 100

ID	balance
...	220

Interest
Multiply by 1.1

ID	balance
...	220

NewSQL

Problems

1. ACID } Reengineering
2. **Consistency** } **Tough luck**
3. Joins

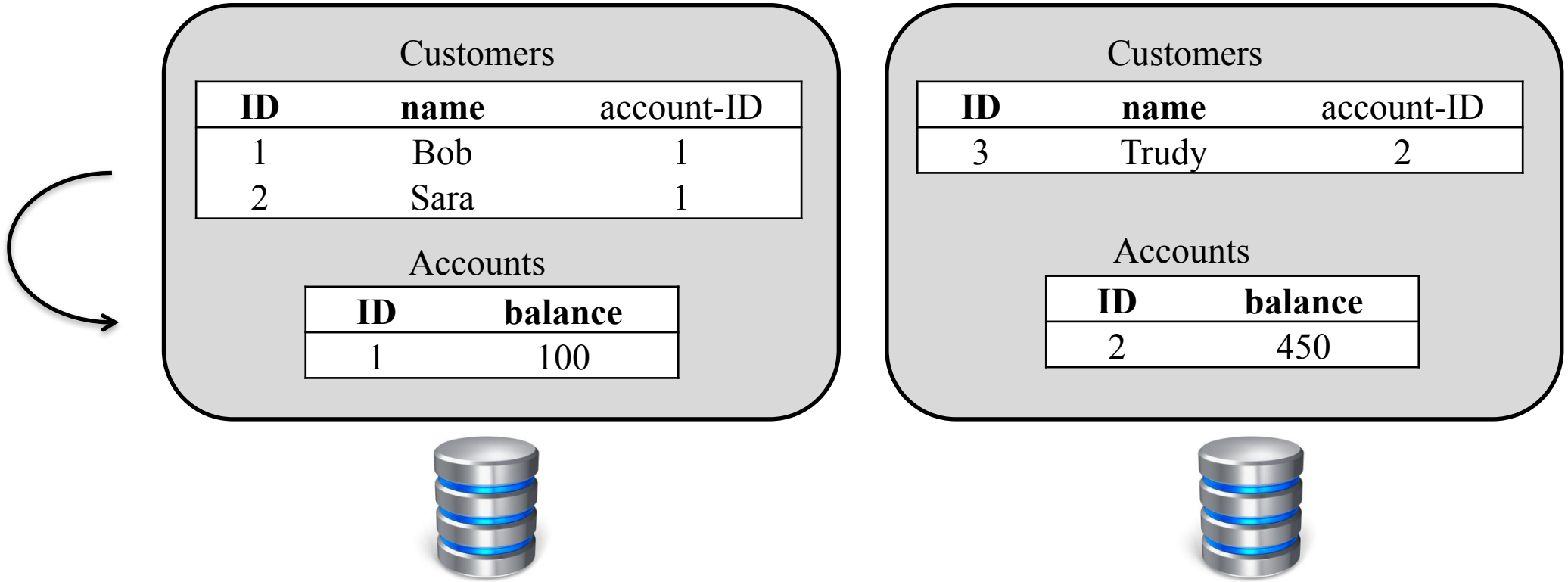
NewSQL

Problems

1. ACID } Reengineering
2. Consistency } Tough luck
3. **Joins** } **Tough luck**

NewSQL

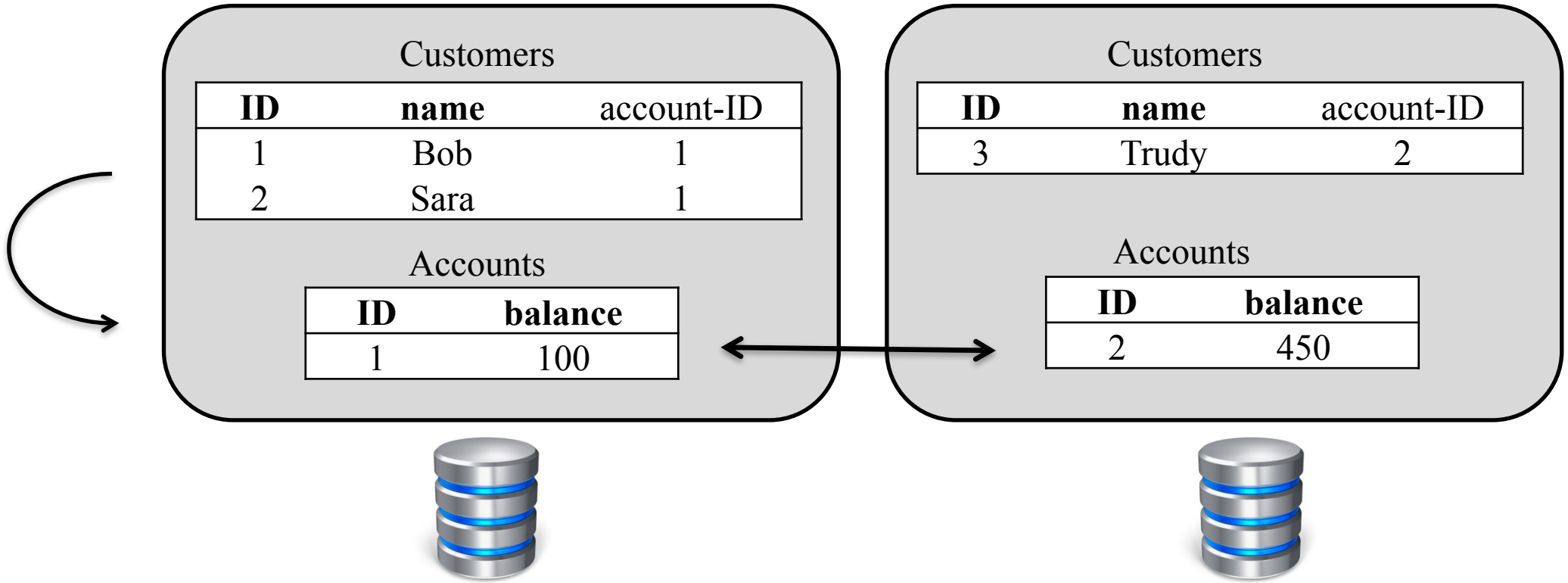
Data about same object on same machine



NewSQL

Data about same object on same machine

Transactions across machines cost (e.g. money transfers)



When to use NewSQL

Transactions span multiple data objects

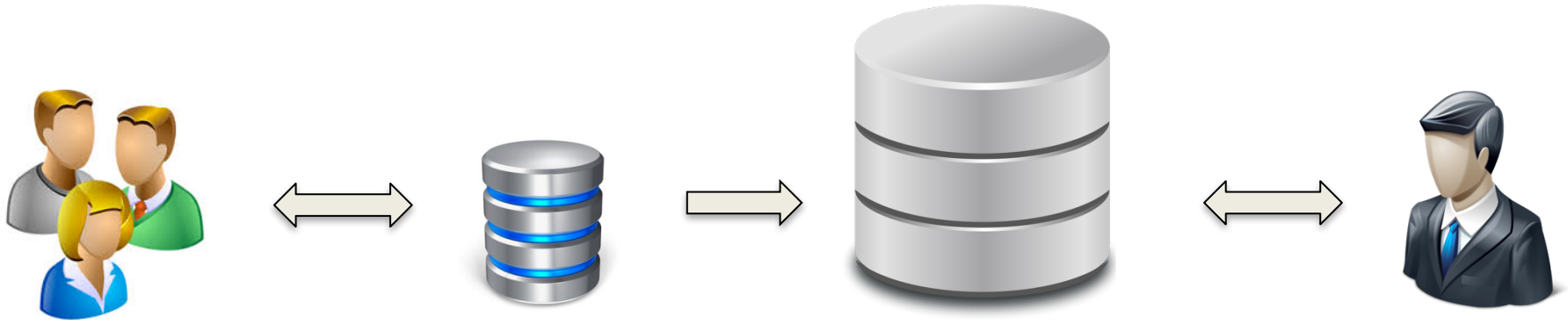
Strong consistency

Run at scale

More expertise

Conclusion

Analytical Databases



End users

Transactional Database

Analytical Database

Business Intelligence

Column-Stores

Joins

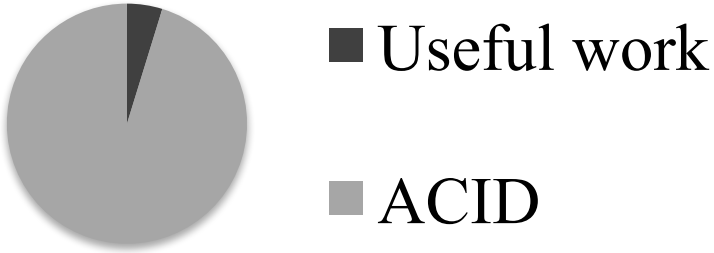
Customers

ID	name	acc-ID
	…	
2	Bob	X
	…	

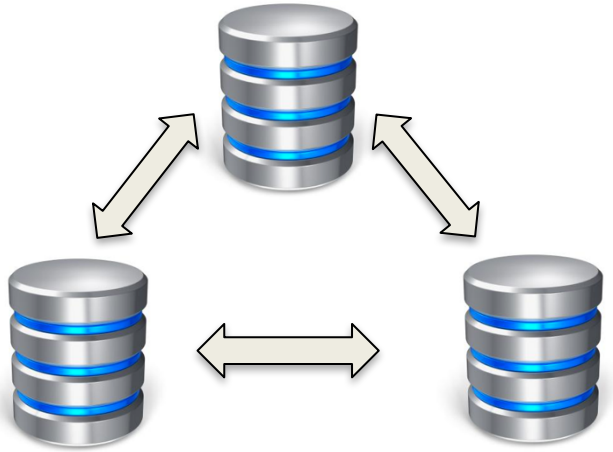
Accounts

ID	balance
	…
X	100
	…

Locking & Logging



Consistency



NoSQL Databases

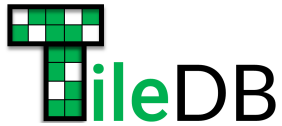
- Simplify
- De-normalize
- Weak consistency
- Non-interacting objects

NewSQL Databases

- Reengineer
- Normalize
- Strong consistency
- Interacting objects

There is more…

Scientific databases:



Graph databases

Considered NoSQL

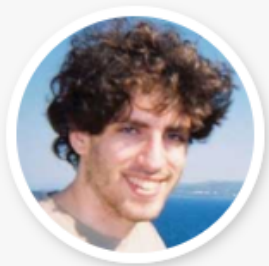
Hopefully now you have reasoning tools

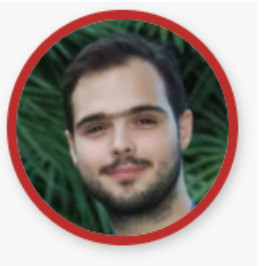
DATA SYSTEMS. LABORATORY

@ Harvard School of Engineering and Applied Sciences


Stratos Idreos


Manos Athanassoulis

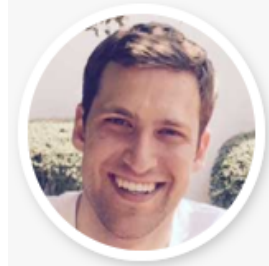

Niv Dayan


Kostas Zoumpatianos


Michael Kester


Lukas Maas


Abdul Wasay


Brian Hentschel

prof

post-docs

phd students

http://daslab.seas.harvard.edu/

Thanks!!!

Questions?

Presentation: nivdayan.github.io/db_pres.pdf

MongoDB tutorial: nivdayan.github.io/mongo.pdf