Very Large, Distributed Data Volumes

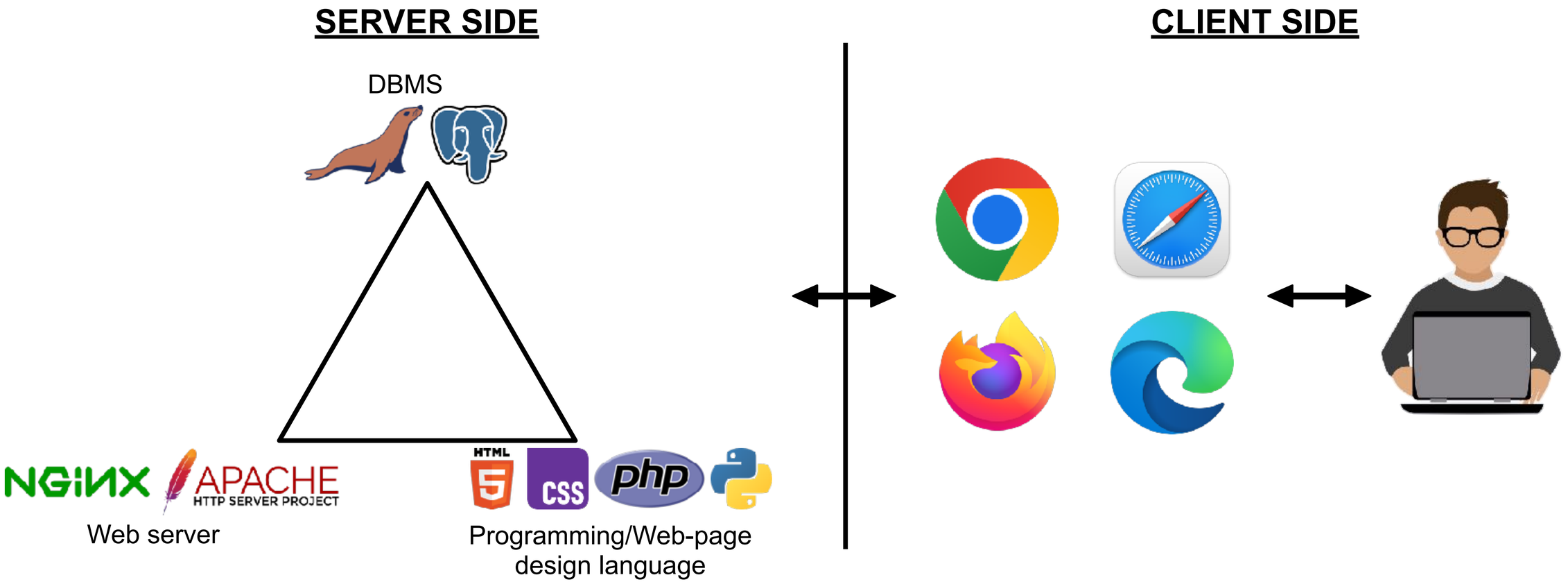
Database-as-a-service

Theodoros Chondrogiannis

October 23, 2025

Recall

- Simple way to create a website on your own server



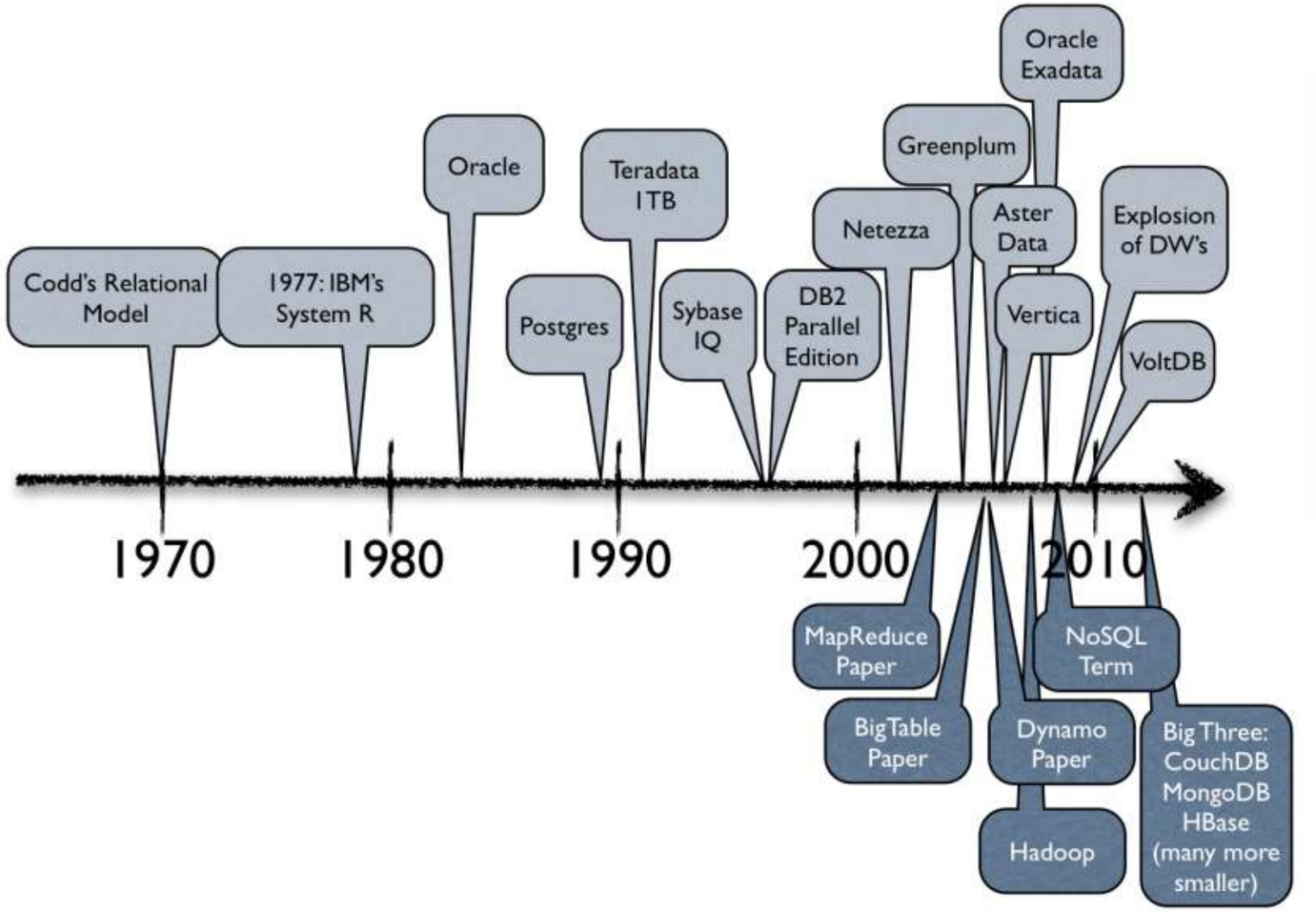
Outline

- Traditional databases
- Distributed databases
- Database-as-a-Service (DBaaS)
- Relational Cloud

Outline

- Traditional databases
- Distributed databases
- Database-as-a-Service (DBaaS)
- Relational Cloud

History of DBMS



Relational Databases

- ER modeling
- Relational schema
- Organize data in tables

Employee		
Name	**Age**	**Salary**
Alice	29	45000
Martin	26	38000
John	28	36000
Mario	35	58000

Department		
Director	**Name**	**Building**
Mario	IT	K
Alice	Finance	F

- Use indices to speed-up access

Relational Databases - Pros

- Flexible by design
- Familiar BCNF structure (strong mathematical background)
- Transactions & ACID
- Very “mature” & well tested (mostly)
- Easy adoption/integration

Relational Databases - Cons

- Large and unstructured data
- Lots of random I/Os and often write-heavy
- Not built for distributed applications/environments
- Single point of failure
- Speed (performance), i.e, not fast enough for specialized applications
- Scale up, not out
  - Scale up: grow capacity by replacing old machines with newer ones
  - Scale out: incrementally grow capacity by adding more COTS

Relational Databases - Cons

- Large and unstructured data
- Lots of random I/Os and often write-heavy
- **Not built for distributed applications/environments**
- **Single point of failure**
- Speed (performance), i.e, not fast enough for specialized applications
- **Scale up, not out**
  - Scale up: grow capacity by replacing old machines with newer ones
  - Scale out: incrementally grow capacity by adding more COTS

The shortcomings are not only of relational DBMS, but of many centralized systems in general

Outline

- Traditional databases
- Distributed databases
- Database-as-a-Service (DBaaS)
- Relational Cloud

Distributed Databases

- Heterogeneous and Homogeneous Databases
- Distributed Data Storage
- Distributed Transactions
- Commit Protocols
- Concurrency Control in Distributed Databases
- Availability
- Distributed Query Processing
- Heterogeneous Distributed Databases
- Directory Systems

Distributed Databases

- **Heterogeneous and Homogeneous Databases**
- **Distributed Data Storage**
- Distributed Transactions
- Commit Protocols
- Concurrency Control in Distributed Databases
- Availability
- **Distributed Query Processing**
- Heterogeneous Distributed Databases
- Directory Systems

Homogenous vs Heterogenous

- Homogeneous distributed database
  - All sites have identical software
  - Are aware of each other and agree to cooperate in processing user requests.
  - Each site surrenders part of its autonomy in terms of right to change schemas or software
  - Appears to user as a single system
- Heterogeneous distributed database
  - Different sites may use different schemas and software
    - Difference in schema is a major problem for query processing
    - Difference in software is a major problem for transaction processing
  - Sites may provide only limited facilities for cooperation in transaction processing

Distributed Data Storage

- Replication
  - System maintains multiple copies of data, stored in different sites, for faster retrieval and fault tolerance
- Fragmentation
  - Relation is partitioned into several fragments stored in distinct sites
- Replication and fragmentation can be combined
  - Relation is partitioned into several fragments: system maintains several identical replicas of each such fragment

Data Replication

- A relation or fragment of a relation is replicated if it is stored redundantly in two or more sites
- Full replication of a relation is the case where the relation is stored at all sites
- Fully redundant databases are those in which every site contains a copy of the entire database

Data Replication - Advantages

- Availability
  - If replicas exist, failure of the site containing relation r does not result in the unavailability of *r*
- Parallelism
  - Queries on *r* may be processed by several nodes in parallel
- Reduced data transfer
  - The relation *r* is available locally at each site, containing a replica of *r*

Data Replication - Disadvantages

- Increased cost of updates
  - Each replica of relation r must be updated
- Increased complexity of concurrency control
  - Concurrent updates to distinct replicas may lead to inconsistent data unless special concurrency control mechanisms are implemented
  - One solution
    - Choose one copy as the primary copy and apply concurrency control operations on the primary copy

Data Partitioning

- Division of relation r into fragments $r_1, r_2, \ldots, r_n$ that contain sufficient information to reconstruct r
- Horizontal partitioning
  - Each tuple of r is assigned to one or more fragments
- Vertical partitioning
  - The schema for the relation r is split into several smaller schemas
    - All schemas must contain a common candidate key (or superkey) to ensure the lossless join property
    - A special attribute, the tuple-id attribute, may be added to each schema to serve as a candidate key

Data Partitioning - Example

$account(branch, number, balance)$

branch	number	balance
Hillside	A-305	500
Hillside	A-226	336
Hillside	A-155	62
Valleyview	A-177	205
Valleyview	A-402	10000
Valleyview	A-408	1123
Valleyview	A-639	750

- Horizontal Partitioning

branch	number	balance
Hillside	A-305	500
Hillside	A-226	336
Hillside	A-155	62

branch	number	balance
Valleyview	A-177	205
Valleyview	A-402	10000
Valleyview	A-408	1123
Valleyview	A-639	750

$account_1 = \sigma_{brach='Hillside'}(account)$ $account_2 = \sigma_{brach='Valleyview'}(account)$

- Vertical Partitioning

branch	number	tid
Hillside	A-305	1
Hillside	A-226	2
Hillside	A-155	3
Valleyview	A-177	4
Valleyview	A-402	5
Valleyview	A-408	6
Valleyview	A-639	7

number	balance	tid
A-305	500	1
A-226	336	2
A-155	62	3
A-177	205	4
A-402	10000	5
A-408	1123	6
A-639	750	7

$account_1 = \pi_{brach,number,tid}(account)$ $account_2 = \pi_{number,balance,tid}(account)$

Advantages of Data Partitioning

- Horizontal:
  - allows parallel processing on fragments of a relation
  - allows a relation to be split so that tuples are located where they are most frequently accessed
- Vertical:
  - allows tuples to be split so that each part of the tuple is stored where it is most frequently accessed
  - tuple-id attribute allows efficient joining of vertical fragments
  - allows parallel processing on a relation
- Vertical and horizontal partitioning can be mixed

Distributed Query Processing

- For centralized systems, the primary criterion for measuring the cost of a particular strategy is the number of disk accesses
- In a distributed system, other issues must be taken into account:
  - The cost of a data transmission over the network
  - The potential gain in performance from having several sites process parts of the query in parallel

Query Tranformation

- Translating algebraic queries on fragments.
  - It must be possible to construct relation r from its fragments
  - Replace relation r by the expression to construct r from its fragments
- Consider the horizontal fragmentation of $account$

$$account_1 = \sigma_{brach='Hillside'}(account) \qquad account_2 = \sigma_{brach='Valleyview'}(account)$$

- Consider the following query

$$\sigma_{brach='Hillside'}(account) \rightarrow \sigma_{brach='Hillside'}(account_1 \cup account_2) \rightarrow$$

$$\sigma_{brach='Hillside'}(account_1) \cup \sigma_{brach='Hillside'}(account_2) \rightarrow \sigma_{brach='Hillside'}(account_1)$$

Simple Join

- Consider the following relational algebra expression in which the relations are neither replicated nor fragmented

$$account \bowtie depositor \bowtie branch$$

  - *account* is stored at site S_1
  - *depositor* at S_2
  - *branch* at S_3
- For a query issued at site S_i, the system needs to produce the result at site S_i

Possible Strategies

- Ship copies of all three relations to site S_i and choose a strategy for processing the entire locally at site S_i
- Ship copies between sites
  - Ship a copy of $account$ and compute $t_1 = account \bowtie depositor$ at S_2
  - Ship $temp_1$ to S_3, and compute $temp_2 = temp_1 \bowtie branch$ at S_3
  - Ship the result $temp_2$ to S_i
- You must consider some important factors
  - Amount of data being shipped, cost of transmitting a data block between sites, relative processing speed at each site

Outline

- Traditional databases
- Distributed databases
- Database-as-a-Service (DBaaS)
- Relational Cloud

Database-as-a-service

- What is DBaaS?
  - A cloud computing service that lets users access and use database software without purchasing and setting up hardware, installing software or managing the system themselves

Traditional vs Cloud Databases

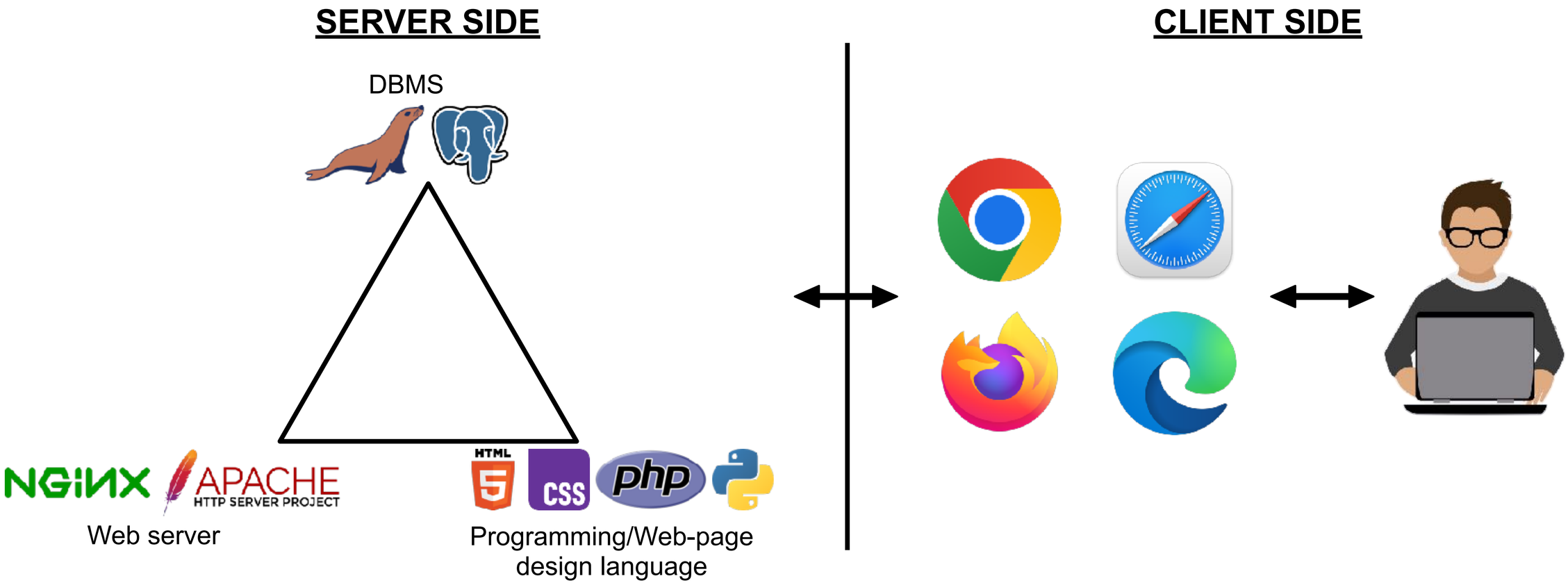
- Traditional databases
  - Physical servers managed and operated on a business's premises
  - Provide immediate access to data
  - The upfront costs can be very high
- Cloud-based databases
  - Operate in a cloud environment - information is stored in various private, public, or hybrid cloud off-site servers accessed by the internet
  - Provide all of the advantages of cloud computing, including speed, flexibility, scalability, and reduced cost
  - Don't require upfront costs - just a subscription

Traditional vs Cloud Databases

Traditional	Cloud
Ideal for structured data and direct access	Offer flexibility regarding data storage
Excel in speed but lack scalability	Provide easy scaling and global accessibility
Allow full control over security	Offer robust security managed by vendors but limit direct control
Higher initial costs	Reduce upfront expenses with pay-as-you-go pricing
Suited for businesses needing data control and instant access	Fits remote operations and those requiring scalability

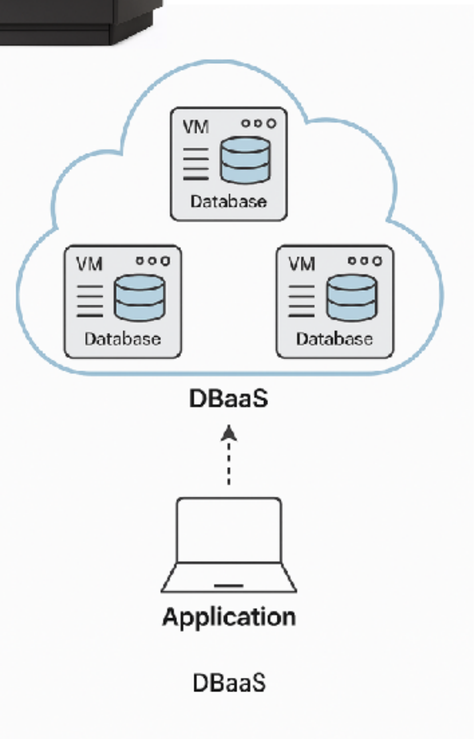
Recall

- Simple way to create a website on your own server

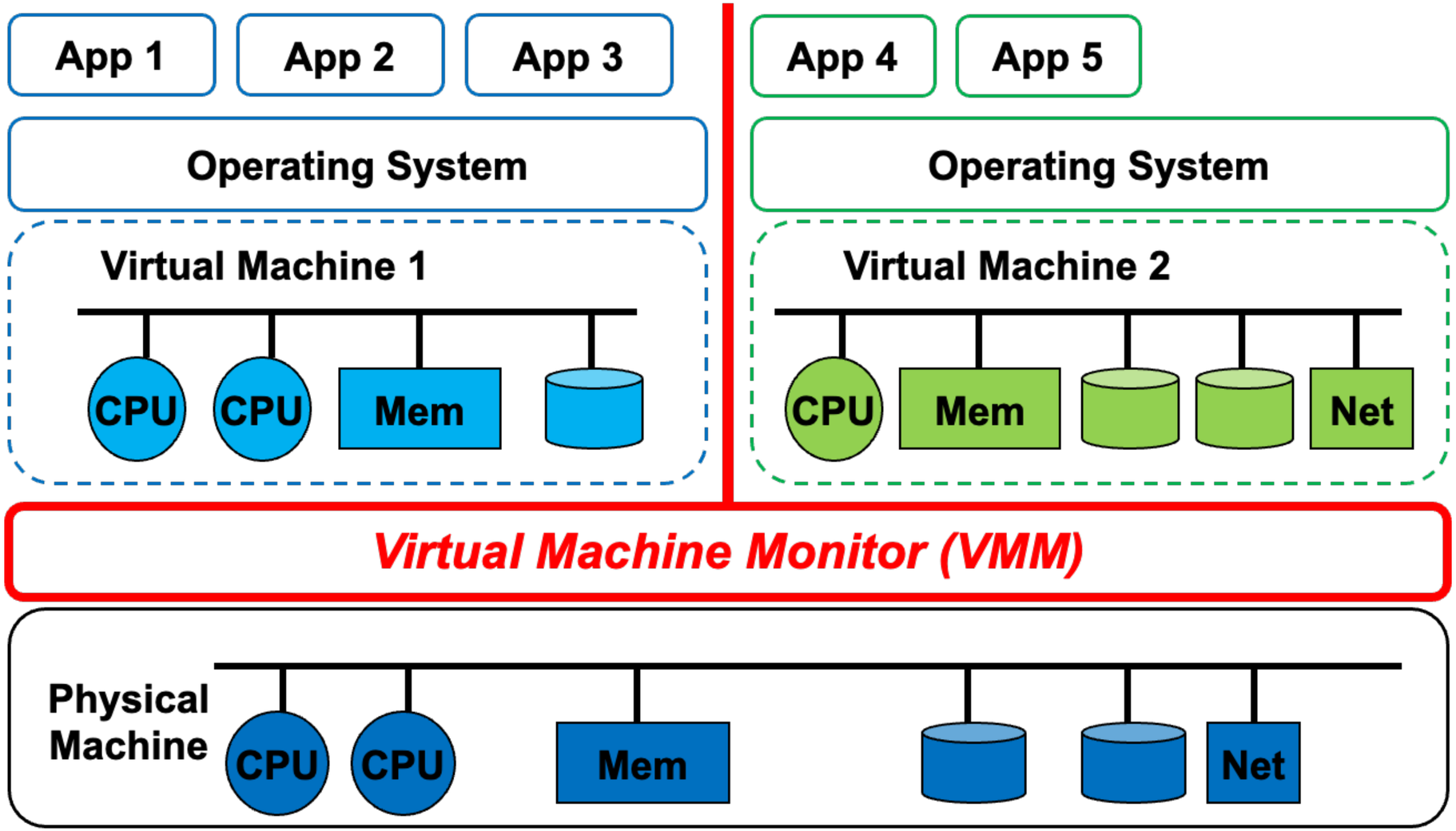


Database System Choice

- Buy a server
  - Requires the acquisition of a physical machine
  - DBA and maintenance are done by the client
- Rent a server
  - Fixed cost per month
  - DBA is done by the developers of the application, but maintenance is done by the service provider
- Rent an instance
  - Fixed cost per month
  - DBA and maintenance are done by the service provider



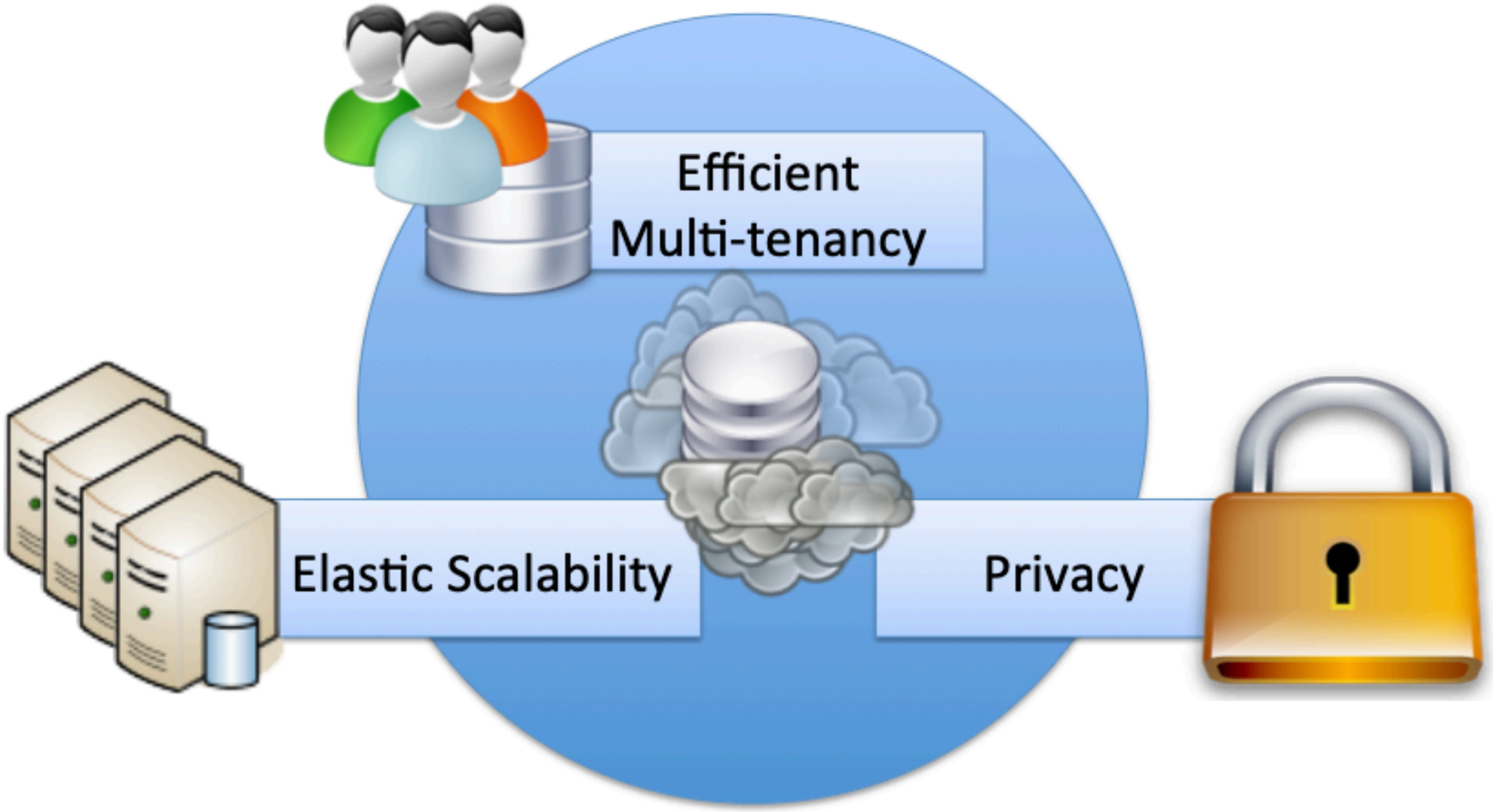
DBaaS


App 1
App 2
App 3
App 4
App 5
Operating System
Operating System
Virtual Machine 1
CPU
CPU
Mem
Virtual Machine 2
CPU
Mem
Net
Virtual Machine Monitor (VMM)
Physical Machine
CPU
CPU
Mem
Net

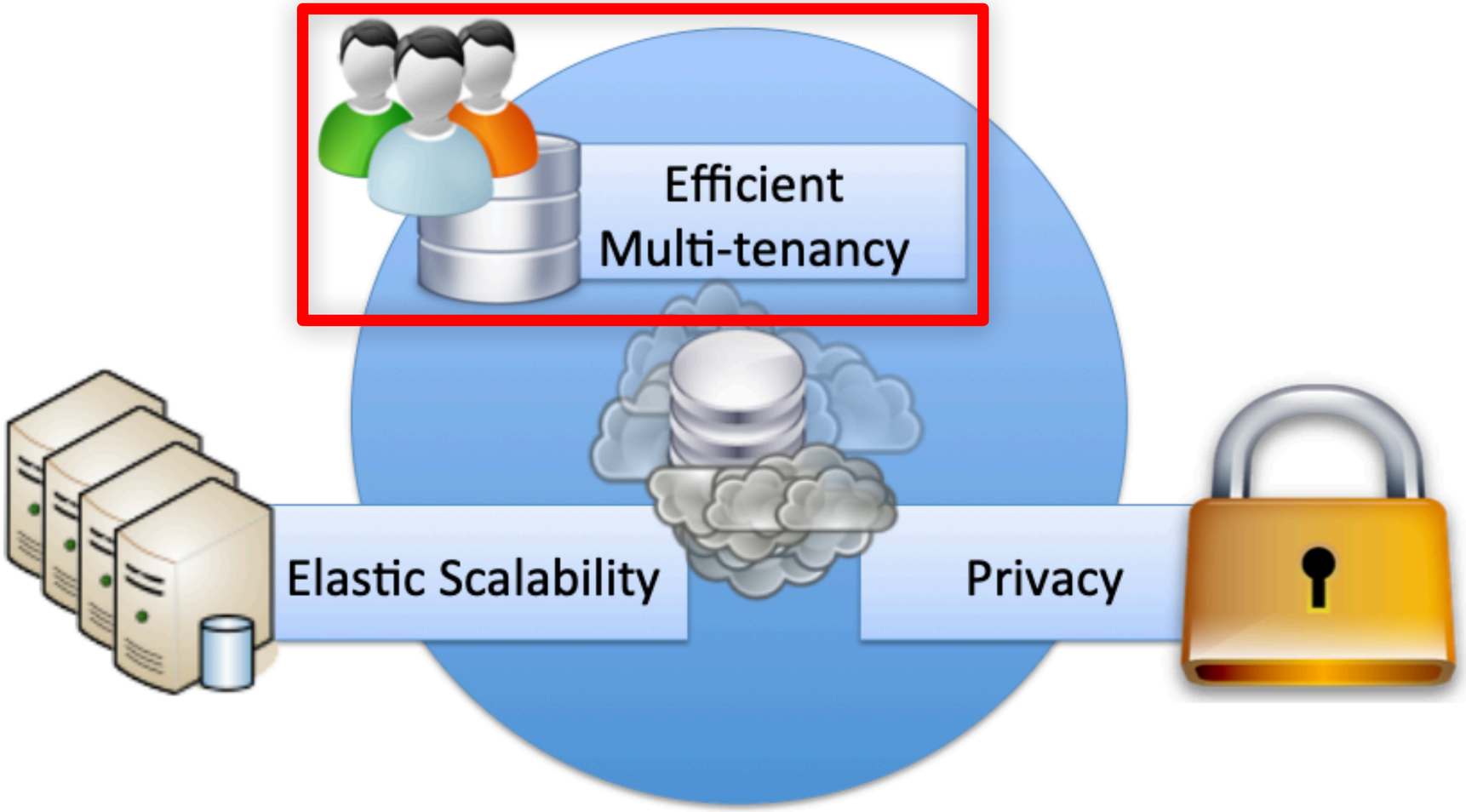
Outline

- Traditional databases
- Distributed databases
- Database-as-a-Service (DBaaS)
- Relational Cloud

Relational Cloud



Relational Cloud



Multi-tenancy

- Multitenancy means that multiple customers of a cloud vendor are using the same computing resources
  - Despite the fact that they share resources, cloud customers are not aware of each other, and their data is kept totally separate
  - Multitenancy is a crucial component of cloud computing; without it, cloud services would be far less practical
  - Multitenant architecture is a feature in many types of public cloud computing
    - IaaS, PaaS, SaaS, containers, and serverless computing

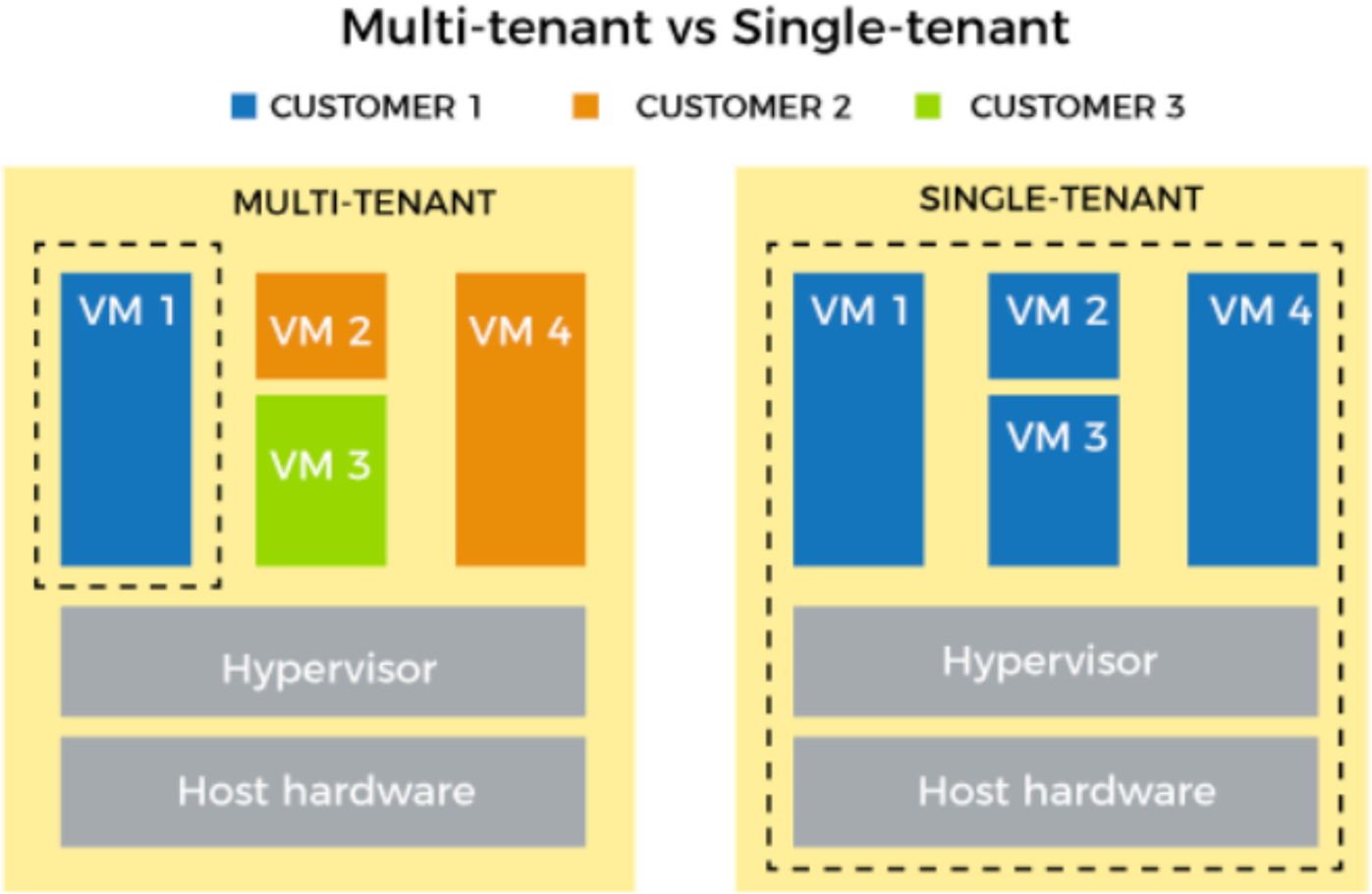
Multi-tenancy

- Many of the benefits of cloud computing are only possible because of multitenancy
- Better use of resources
  - One machine reserved for a single tenant is not efficient, as that tenant is unlikely to utilize all of the computing power. By sharing machines among multiple tenants, the use of available resources is maximized
- Lower costs
  - With multiple customers sharing resources, a cloud vendor can offer their services to many customers at a much lower cost than if each customer required their own dedicated infrastructure

Multi-tenancy Challenges

- Possible security risks and compliance issues
  - Some companies may not be able to store data within shared infrastructure, no matter how secure, due to regulatory requirements.
- Security problems or corrupted data from one tenant could spread to other tenants on the same machine
  - This is extremely rare and should not occur if the cloud vendor has configured their infrastructure correctly
- These security risks are somewhat mitigated by the fact that cloud vendors typically are able to invest more in their security than individual businesses can

Single-tenant vs Multi-tenant



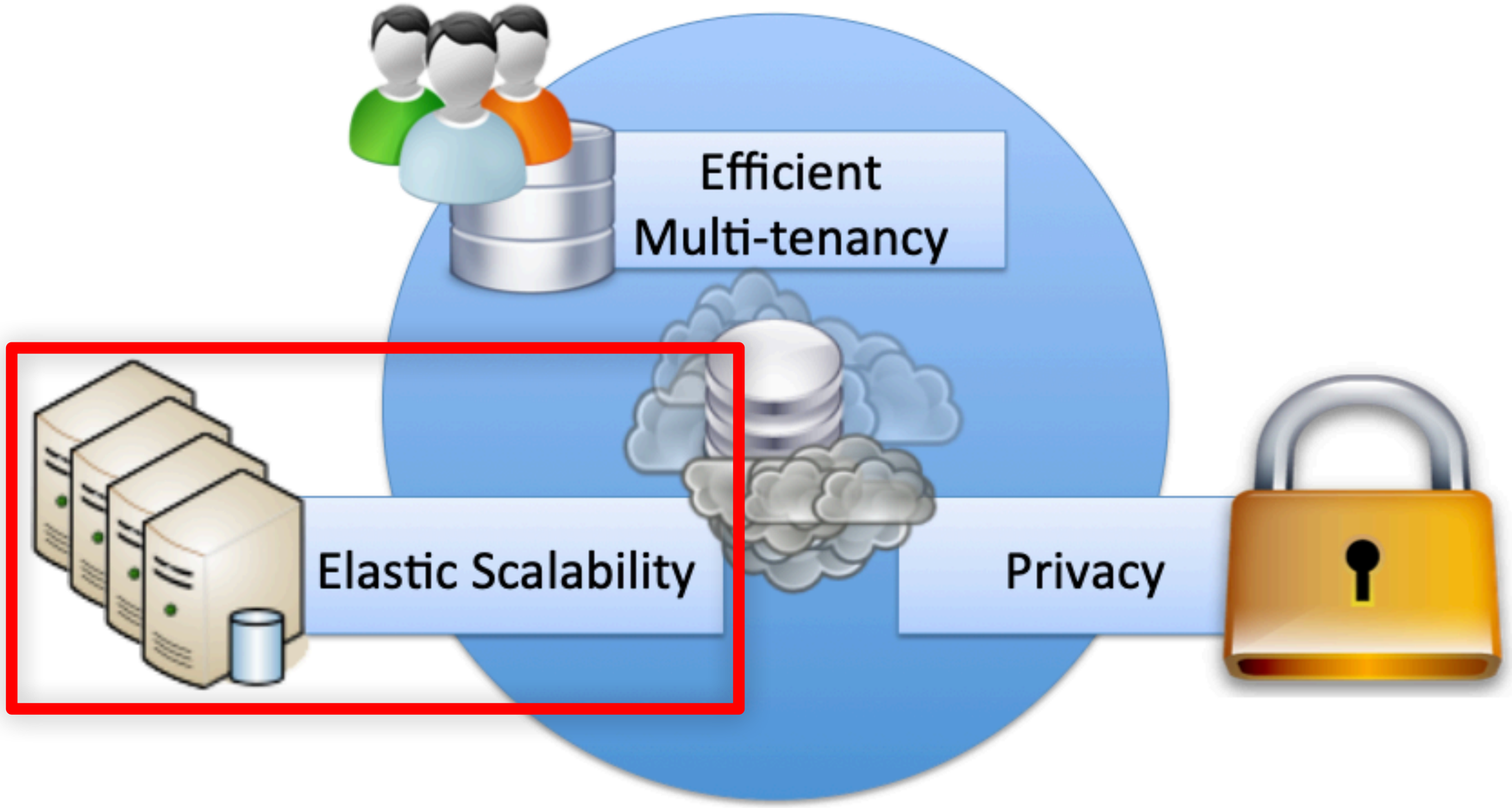
Efficient Multi-tenancy

- Problems
  - Monitoring resource requirements for workloads
  - Predicting the load generated
  - Assigning workloads to physical machines
  - Migrating workloads between nodes
  - Live migration

Efficient Multi-tenancy

- Kairos (monitoring and consolidation engine)
  - Resource Monitor
    - Disk activity and RAM requirements
  - Combined Load Predictor
    - CPU, RAM, and Disk model that predicts the combined resource requirements
  - Consolidation Engine
    - Non-linear optimization techniques to:
      - minimize the number of machines needed
      - balance the load between back-end machines

Relational Cloud



Elastic Scalability

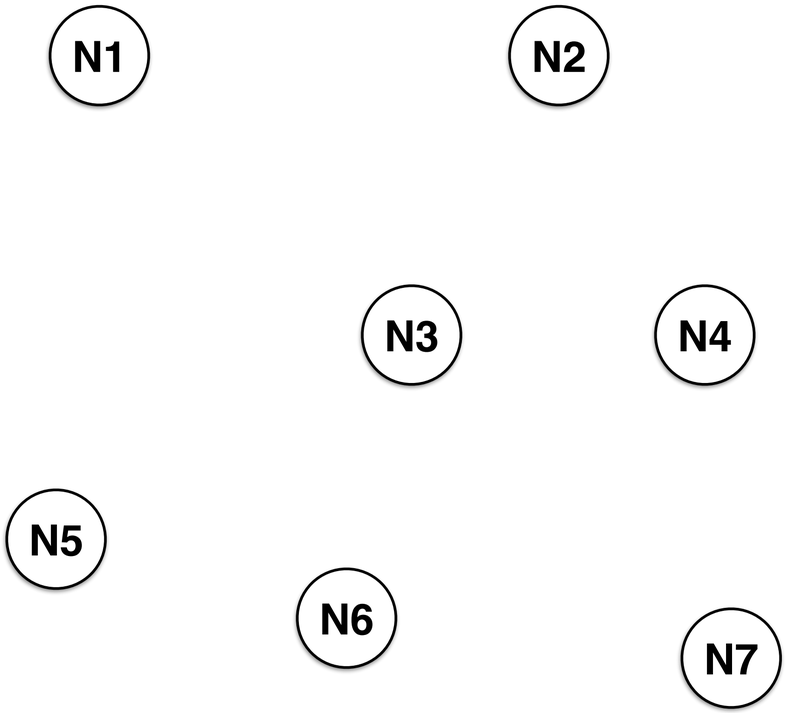
- Workload exceeds single machine capacity
  - Scale a single database to multiple nodes
  - Scale out by query processing partitioning
  - Granular placement and load balancing on backend

Elastic Scalability

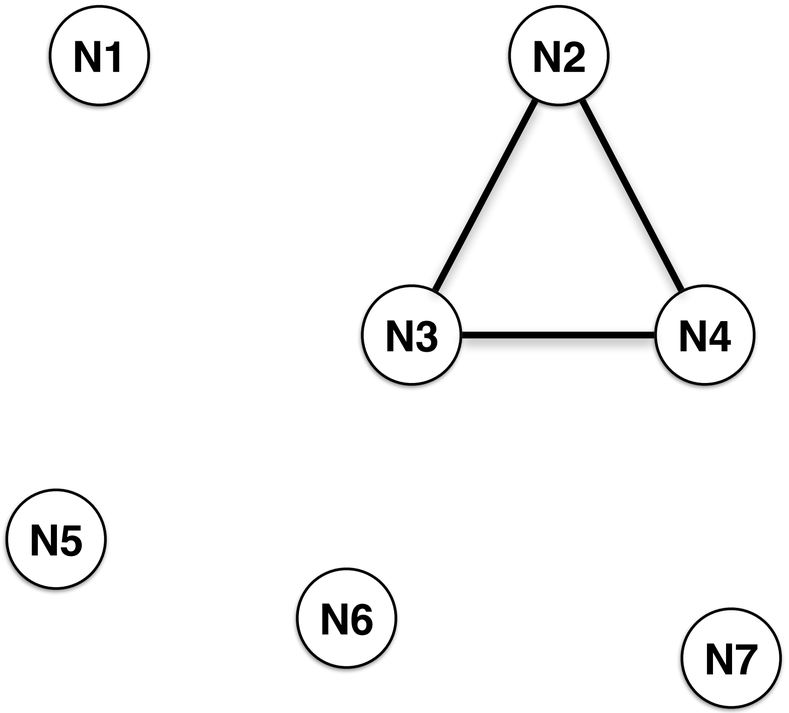
- Strategy well suited for OLTP and Web workloads… but can extend to OLAP
- Minimize cross-node distributed transactions
- Workload-aware partitioner
  - Partition data to minimize multi-node transactions
  - Front-end analyses execution traces represented as a graph
    - Each tuple is a node, and two nodes are connected when accessed together in a transaction

Data/Graph Partitioning

branch	number	balance	tid
Hillside	A-305	500	1
Hillside	A-226	336	2
Hillside	A-155	62	3
Valleyview	A-177	205	4
Valleyview	A-402	10000	5
Valleyview	A-408	1123	6
Valleyview	A-639	750	7



Data/Graph Partitioning

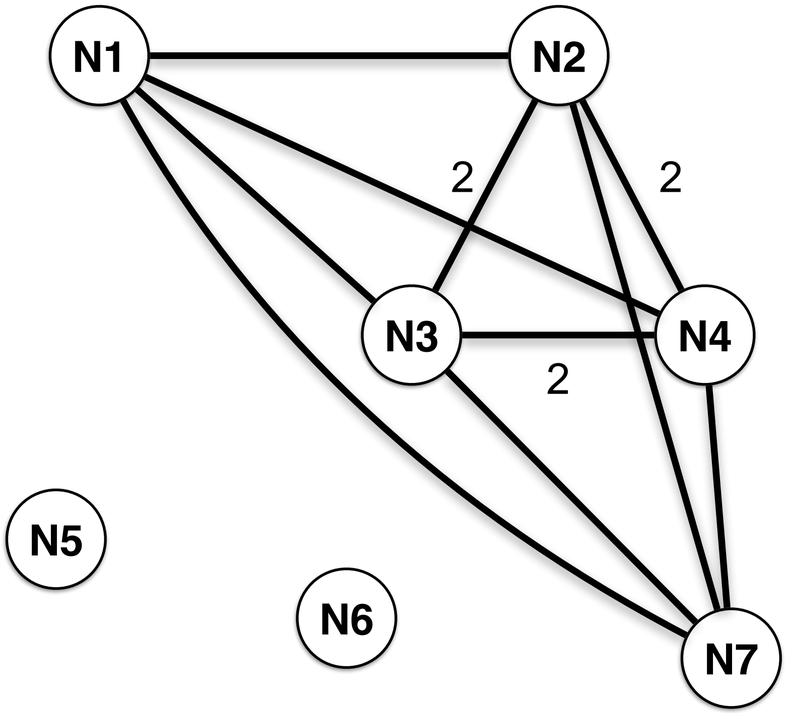


branch	number	balance	tid
Hillside	A-305	500	1
Hillside	A-226	336	2
Hillside	A-155	62	3
Valleyview	A-177	205	4
Valleyview	A-402	10000	5
Valleyview	A-408	1123	6
Valleyview	A-639	750	7

$$\sigma_{balance<500}$$

Data/Graph Partitioning

branch	number	balance	tid
Hillside	A-305	500	1
Hillside	A-226	336	2
Hillside	A-155	62	3
Valleyview	A-177	205	4
Valleyview	A-402	10000	5
Valleyview	A-408	1123	6
Valleyview	A-639	750	7

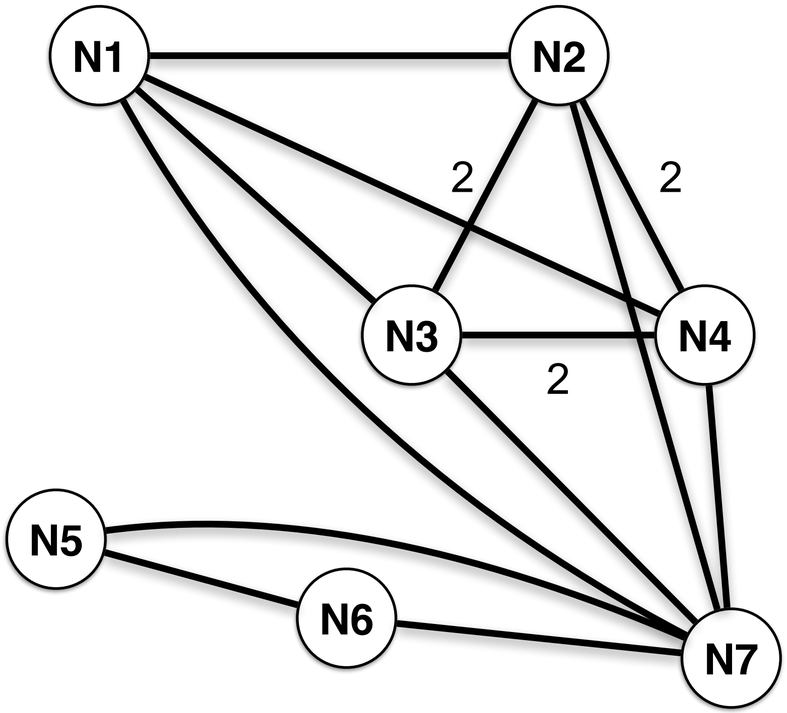


$$\sigma_{balance<500}$$

$$\sigma_{balance<800}$$

Data/Graph Partitioning

branch	number	balance	tid
Hillside	A-305	500	1
Hillside	A-226	336	2
Hillside	A-155	62	3
Valleyview	A-177	205	4
Valleyview	A-402	10000	5
Valleyview	A-408	1123	6
Valleyview	A-639	750	7



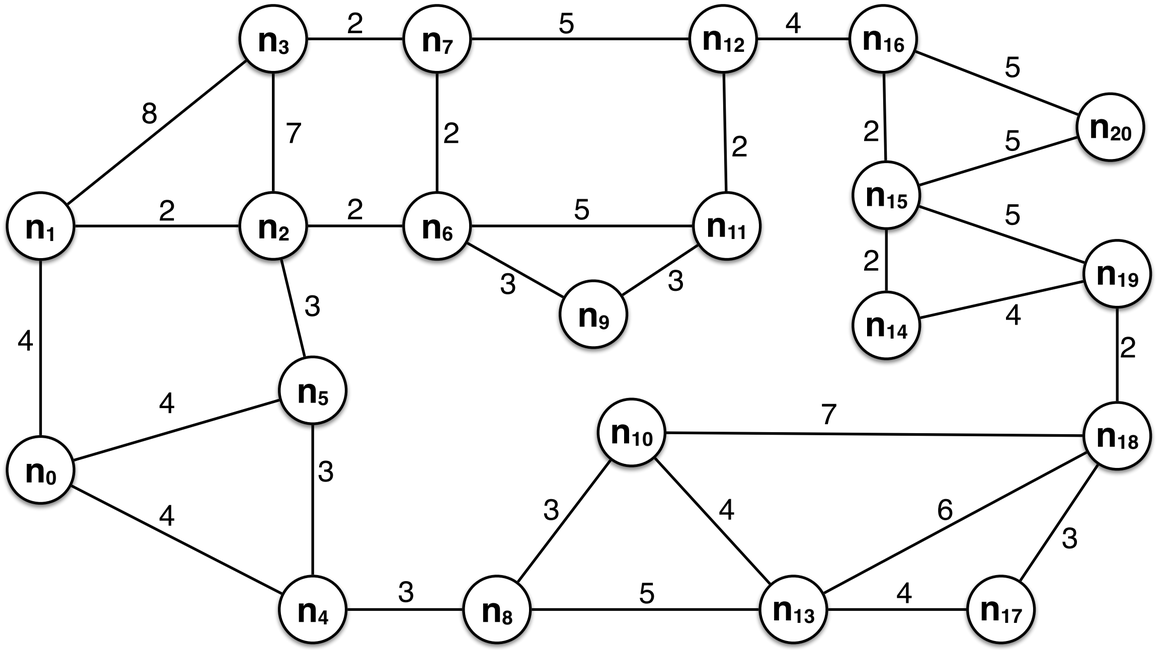
$$\sigma_{balance<500}$$

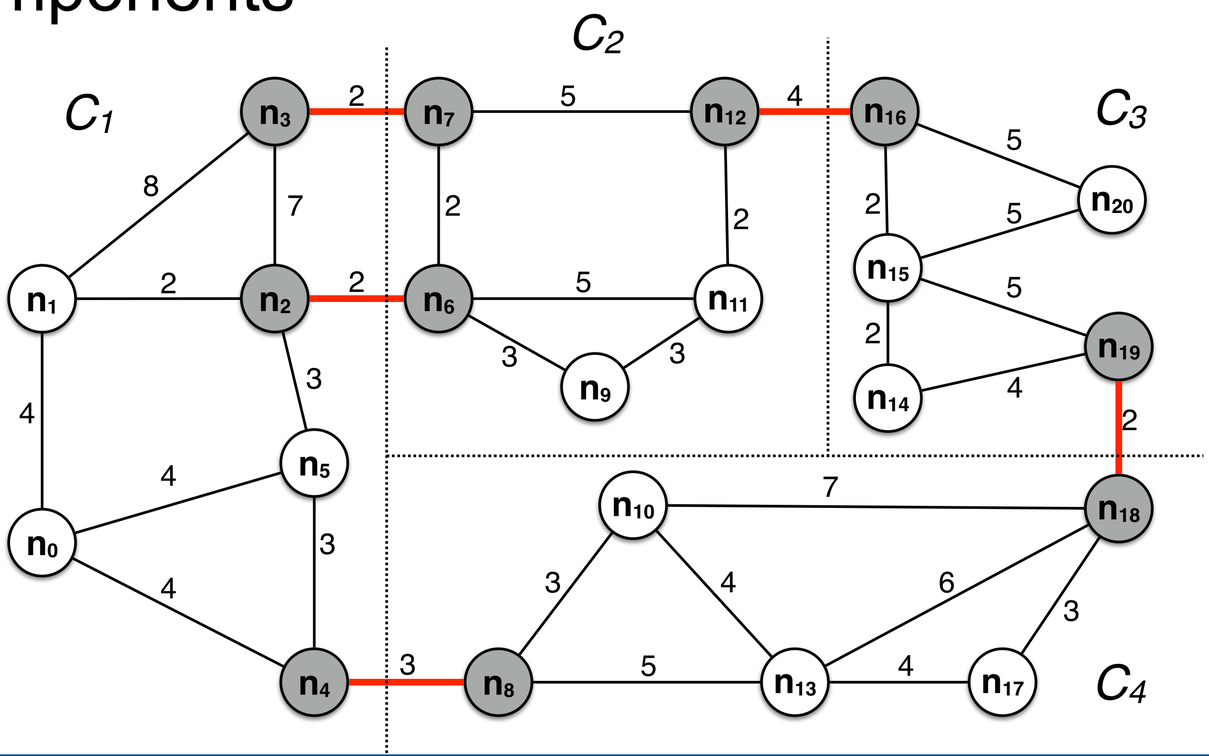
$$\sigma_{balance<800}$$

$$\sigma_{balance>7000}$$

Graph Partitioning

- Criteria
  - Minimize connections between components
  - Maximize total weight between components



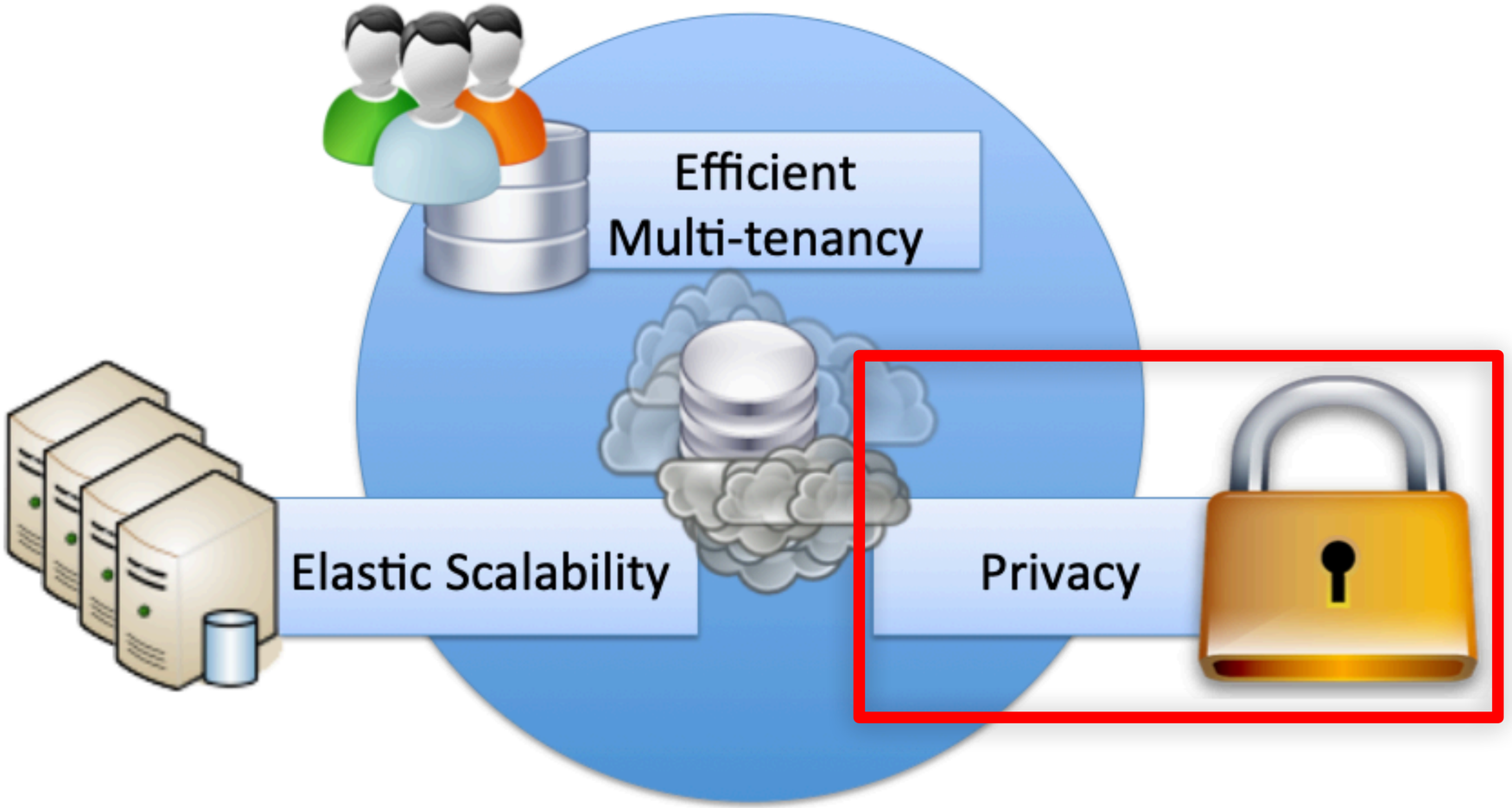


NTNU

Data Partitioning

- The output of the partitioner is an assignment of individual tuples to logical partitions
- The naïve approach leads to a graph with N nodes and up to N^2 edges for an N-tuple database
- Heuristics to reduce the size of the graph
  - blanket statement removal, i.e., the exclusion from the graph of occasional statements that scan large portions of the database
  - sampling tuples and transactions

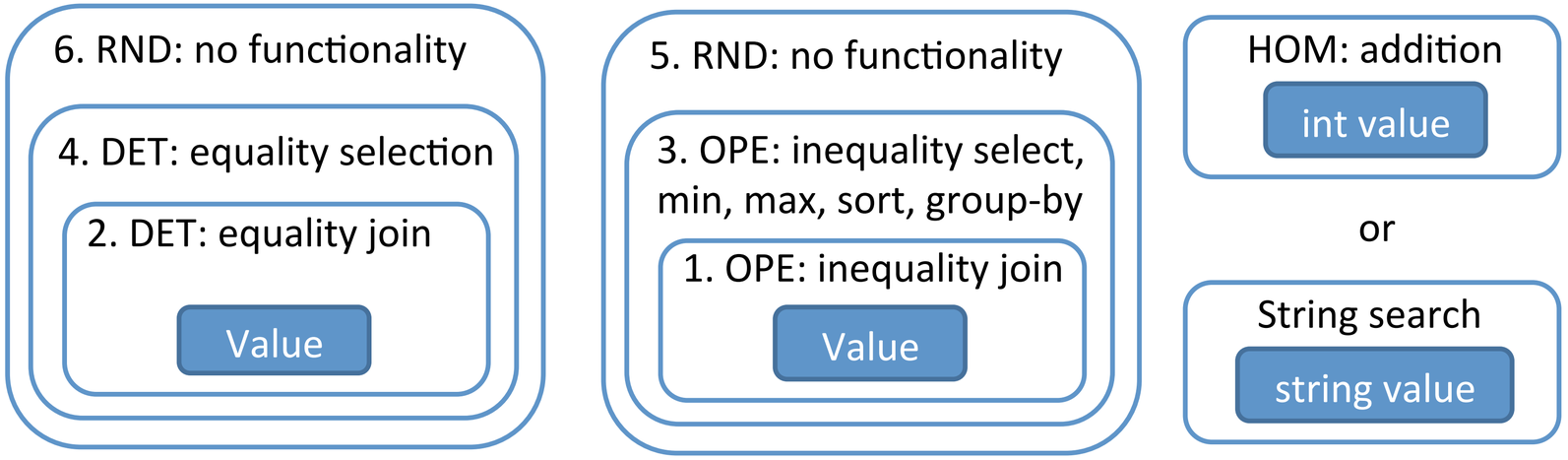
Relational Cloud

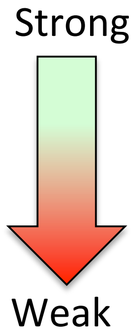


CryptDB

- Adjustable security
  - Onion ring encryption design
  - SQL query on encrypted data
- Security level is dynamically adaptive
  - Converge to an overall security level

CryptDB





RND = Randomized Encryption (no operations allowed)
DET = Deterministic Encryption
OPE = Order-preserving Encryption
HOM = Homomorphic Encryption (operations over encrypted data)

END OF LECTURE