TRAPS IN DISTRIBUTED SYSTEMS

A STICH IN TIME SAVES NINE

Mariusz Krzanowski

ABOUT ME

- Employed in IT since 1999
  - Developer / Architect / Team Leader
  - Now: Senior Software Developer
    @ Demant Technology Centre
- Experienced in many technologies
  - .NET, SQL, SharePoint, BizTalk, Google Cloud, Android, SCCM
    Active Directory, Web Development ……
    20 years is long time ☺
- I like to share my knowledge with others –
  you can meet me at Warszawska Grupa .NET (WG-NET)

Demant Technology Centre

GOALS OF THE PRESENTATION

- To know the existing traps of distributed systems before designing them.
- To better understand distributed systems.

AGENDA - TRAPS IN:

IN SCOPE

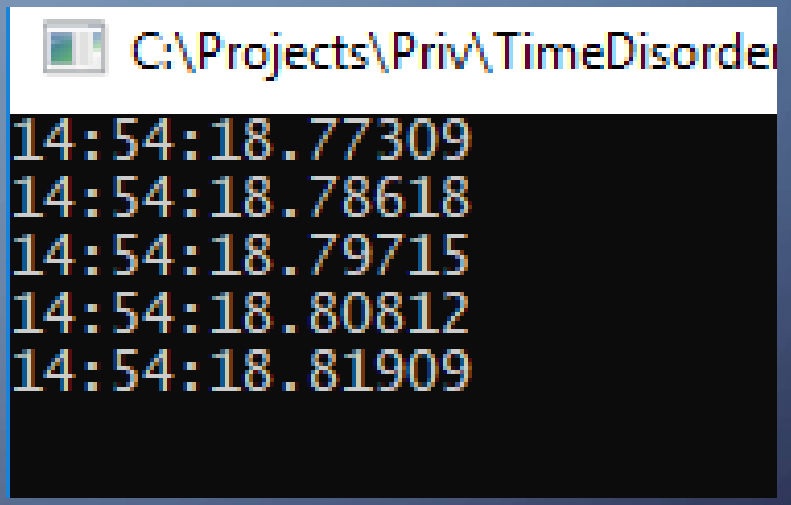
- Time
- Virtualization
- Consistency
- Communication
- Deployment

OUT OF SCOPE

- Security

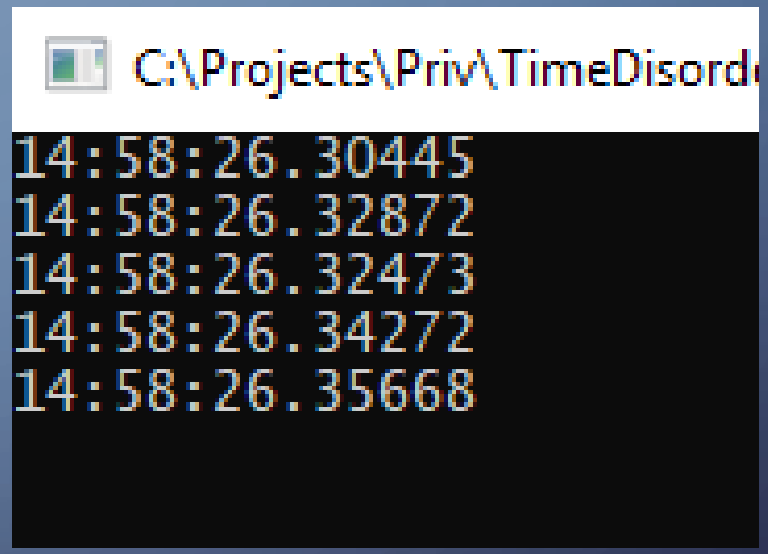
TIME ORDER

```
var t1 = DateTime.UtcNow;
Thread.Sleep(10);
var t2 = DateTime.UtcNow;
Thread.Sleep(10);
var t3 = DateTime.UtcNow;
Thread.Sleep(10);
var t4 = DateTime.UtcNow;
Thread.Sleep(10);
var t5 = DateTime.UtcNow;

Console.WriteLine($"{t1:HH:mm:ss.fffff}");
Console.WriteLine($"{t2:HH:mm:ss.fffff}");
Console.WriteLine($"{t3:HH:mm:ss.fffff}");
Console.WriteLine($"{t4:HH:mm:ss.fffff}");
Console.WriteLine($"{t5:HH:mm:ss.fffff}");
```



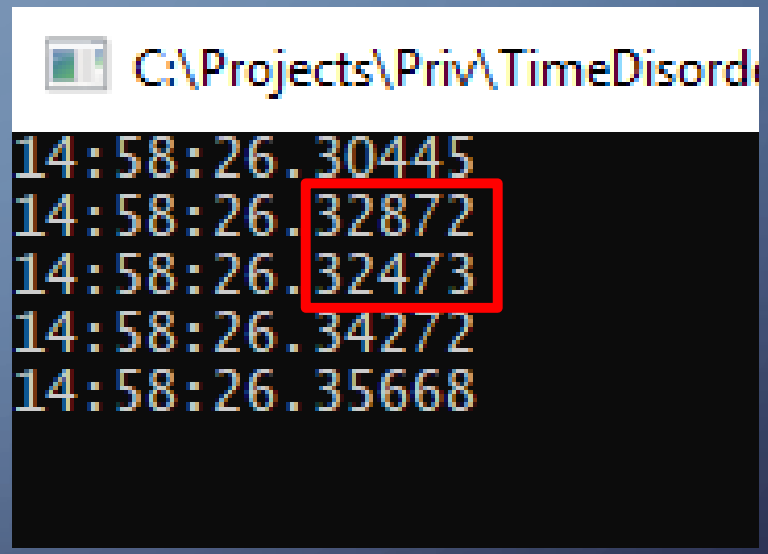
TIME ORDER

```
var t1 = DateTime.UtcNow;
Thread.Sleep(10);
var t2 = DateTime.UtcNow;
Thread.Sleep(10);
var t3 = DateTime.UtcNow;
Thread.Sleep(10);
var t4 = DateTime.UtcNow;
Thread.Sleep(10);
var t5 = DateTime.UtcNow;

Console.WriteLine($"{t1:HH:mm:ss.fffff}");
Console.WriteLine($"{t2:HH:mm:ss.fffff}");
Console.WriteLine($"{t3:HH:mm:ss.fffff}");
Console.WriteLine($"{t4:HH:mm:ss.fffff}");
Console.WriteLine($"{t5:HH:mm:ss.fffff}");
```

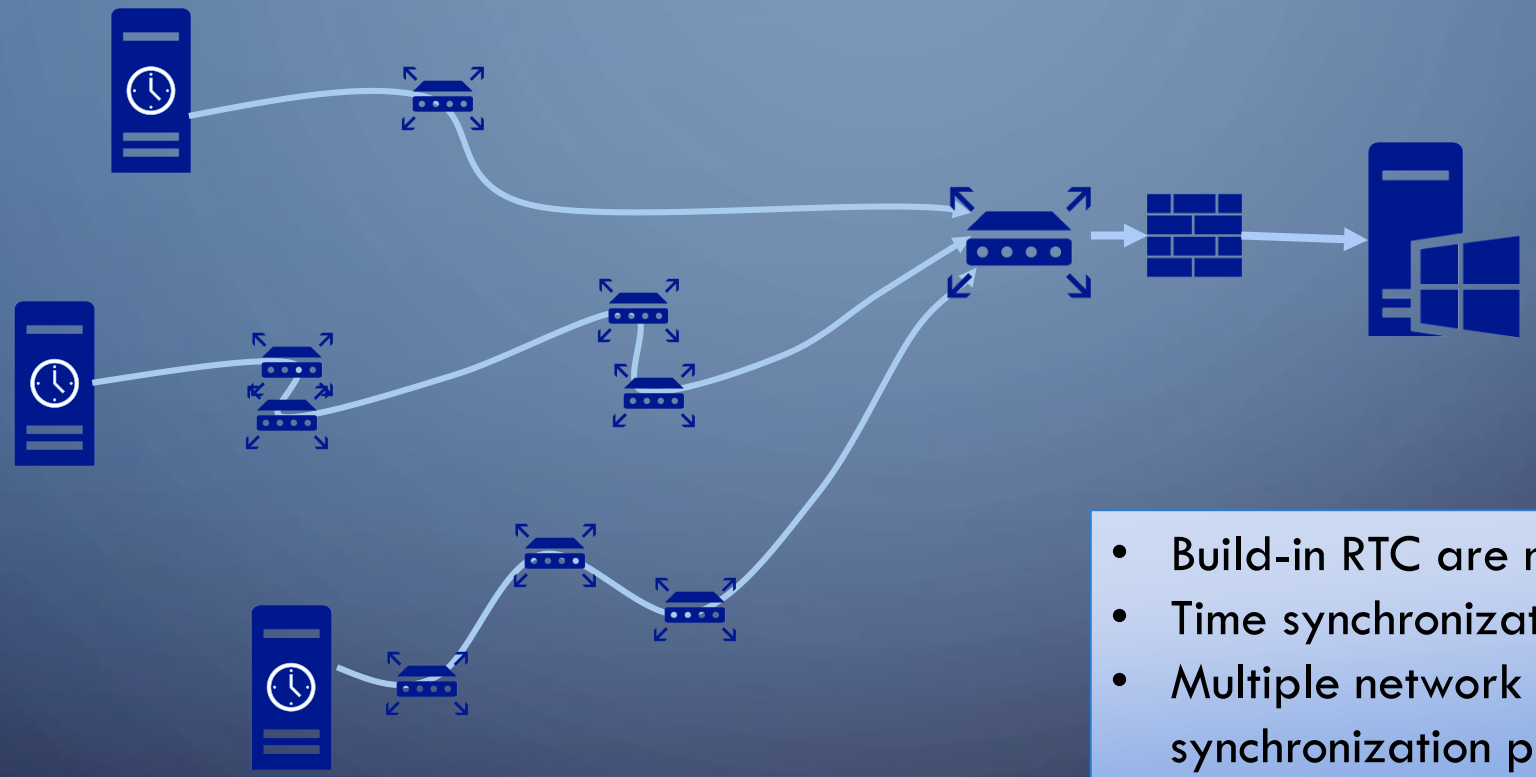


TIME ORDER

```
var t1 = DateTime.UtcNow;
Thread.Sleep(10);
var t2 = DateTime.UtcNow;
Thread.Sleep(10);
var t3 = DateTime.UtcNow;
Thread.Sleep(10);
var t4 = DateTime.UtcNow;
Thread.Sleep(10);
var t5 = DateTime.UtcNow;

Console.WriteLine($"{t1:HH:mm:ss.fffff}");
Console.WriteLine($"{t2:HH:mm:ss.fffff}");
Console.WriteLine($"{t3:HH:mm:ss.fffff}");
Console.WriteLine($"{t4:HH:mm:ss.fffff}");
Console.WriteLine($"{t5:HH:mm:ss.fffff}");
```

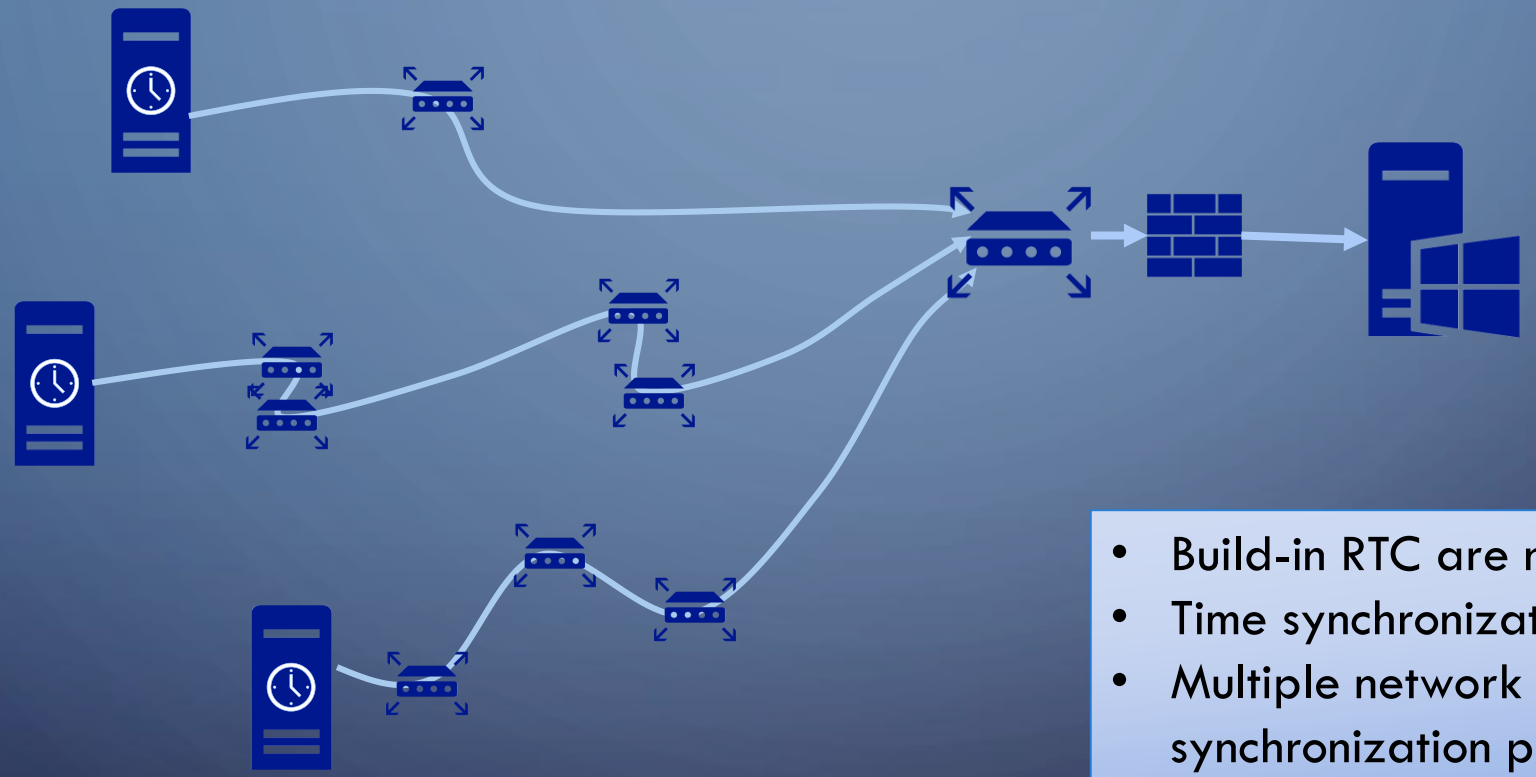


WRONG TIME ORDER – WHY?



- Build-in RTC are not accurate
- Time synchronization latency varies
- Multiple network devices in synchronization path

WRONG TIME ORDER – WHY?



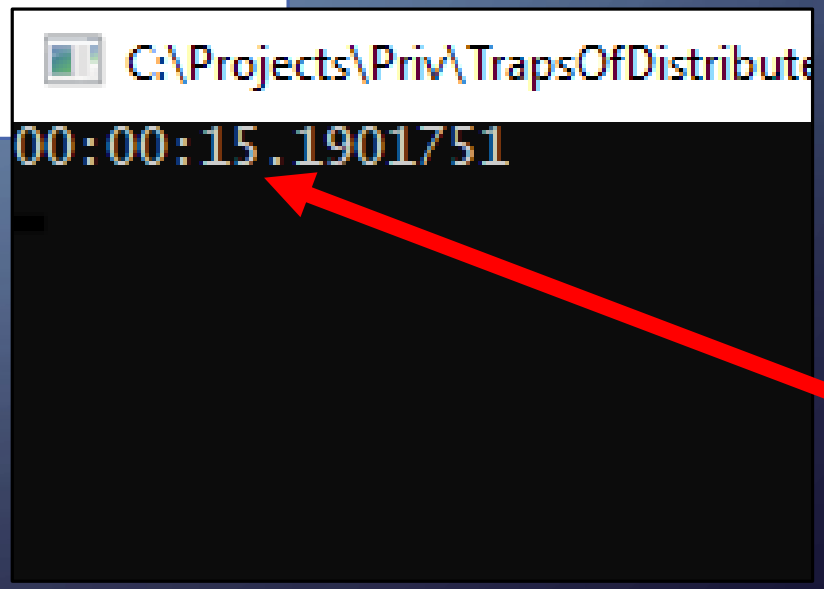
- Build-in RTC are not accurate
- Time synchronization latency varies
- Multiple network devices in synchronization path
- **NOTE! Do not use time only to sequence events.**

STOP THE WORLD

```
var t1 = System.DateTime.UtcNow;
Thread.Sleep(TimeSpan.FromSeconds(1));
var t2 = System.DateTime.UtcNow;
Console.WriteLine($"{t2-t1:c}");
Console.ReadLine();
```

STOP THE WORLD

```
var t1 = System.DateTime.UtcNow;
Thread.Sleep(TimeSpan.FromSeconds(1));
var t2 = System.DateTime.UtcNow;
Console.WriteLine($"{t2-t1:c}");
Console.ReadLine();
```



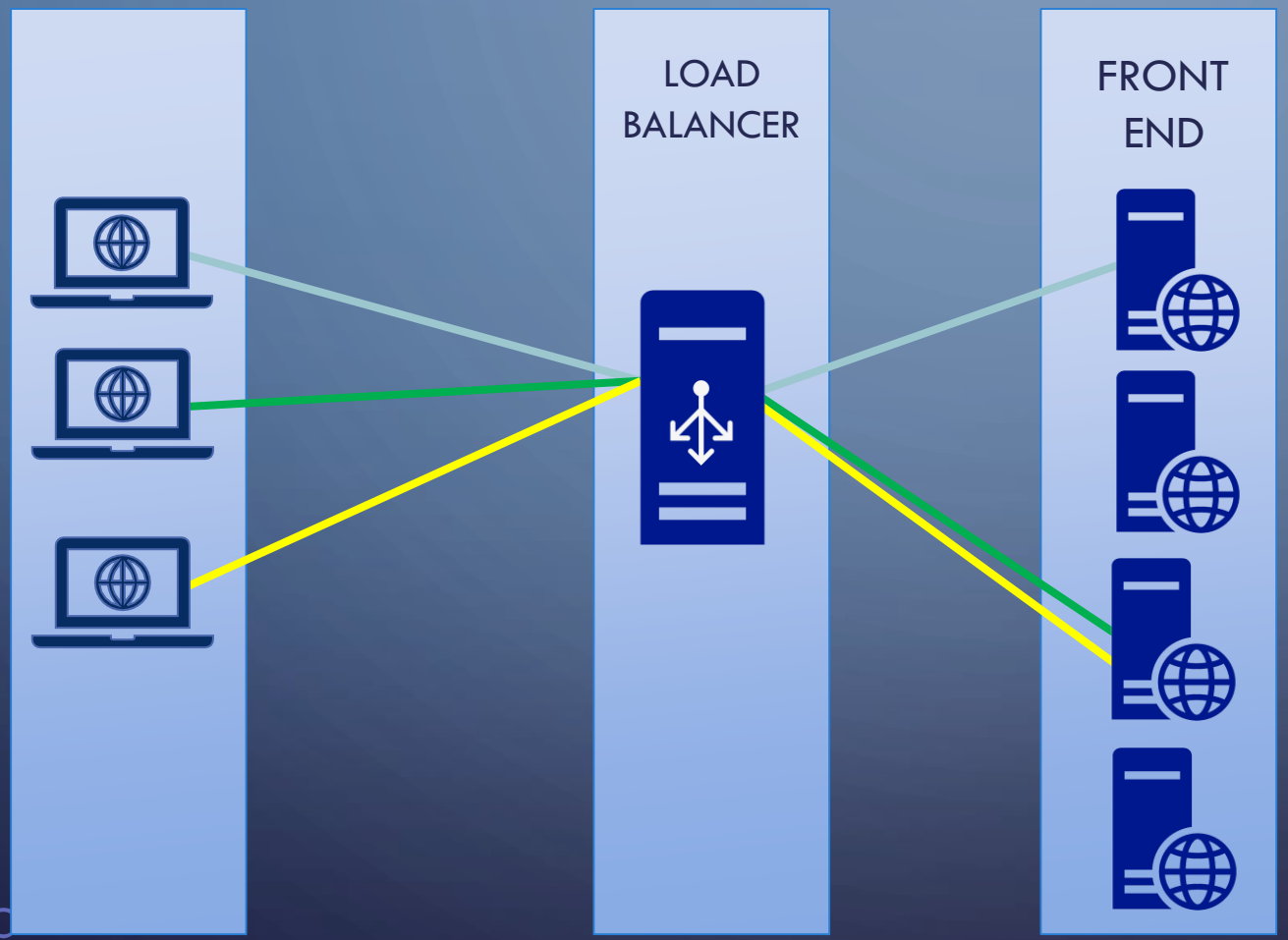
IN CLOUD EVERYTHING IS VIRTUAL

IN CLOUD EVERYTHING IS VIRTUAL

There is always risk that:

- host could be restarted
- guest process could be paused

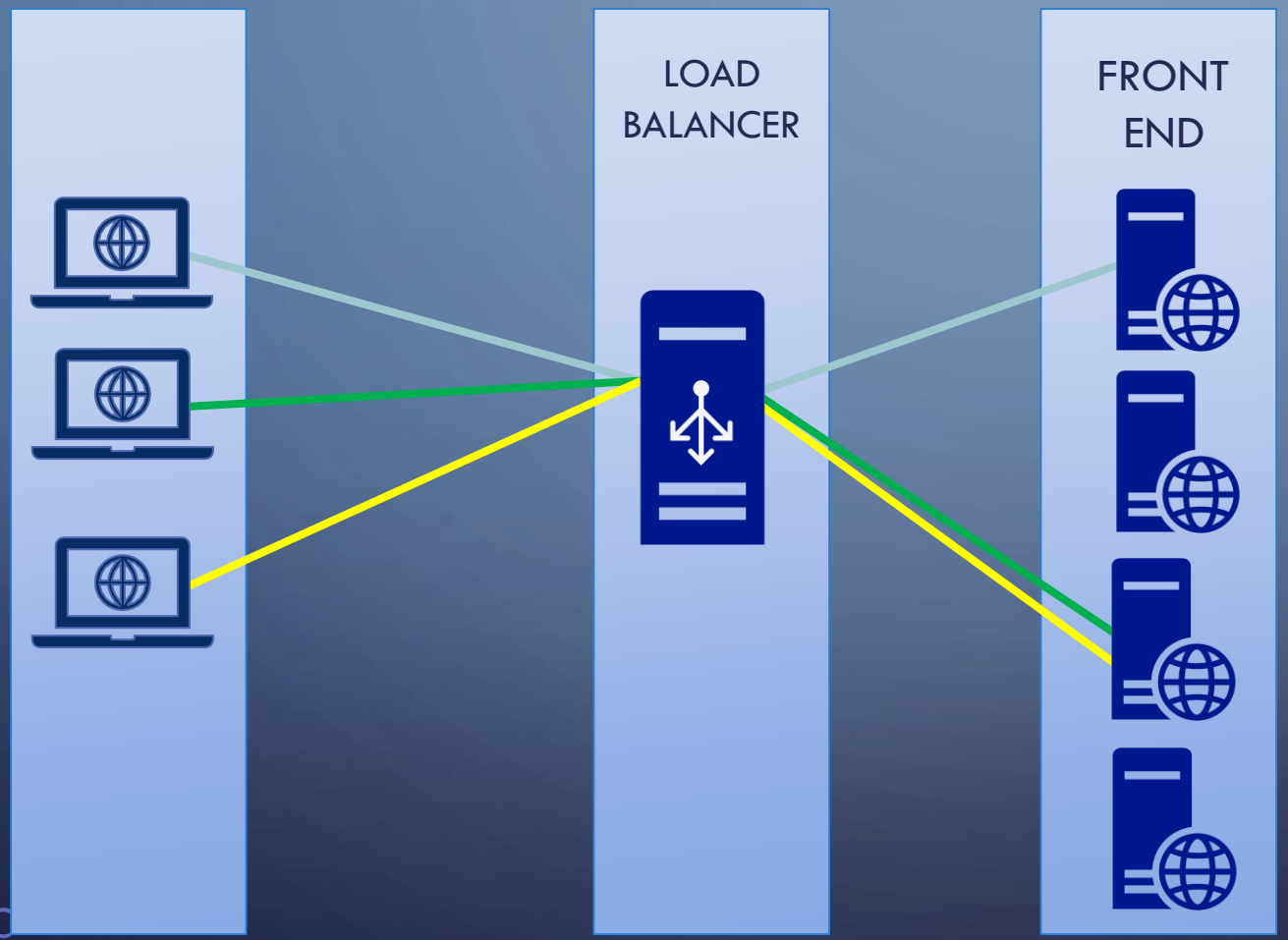
STICKY SESSION



Goal:

- Load balancer sends all traffic from a single client to single server.
- Load balancer can be distributed – e.g. Windows NLB service.

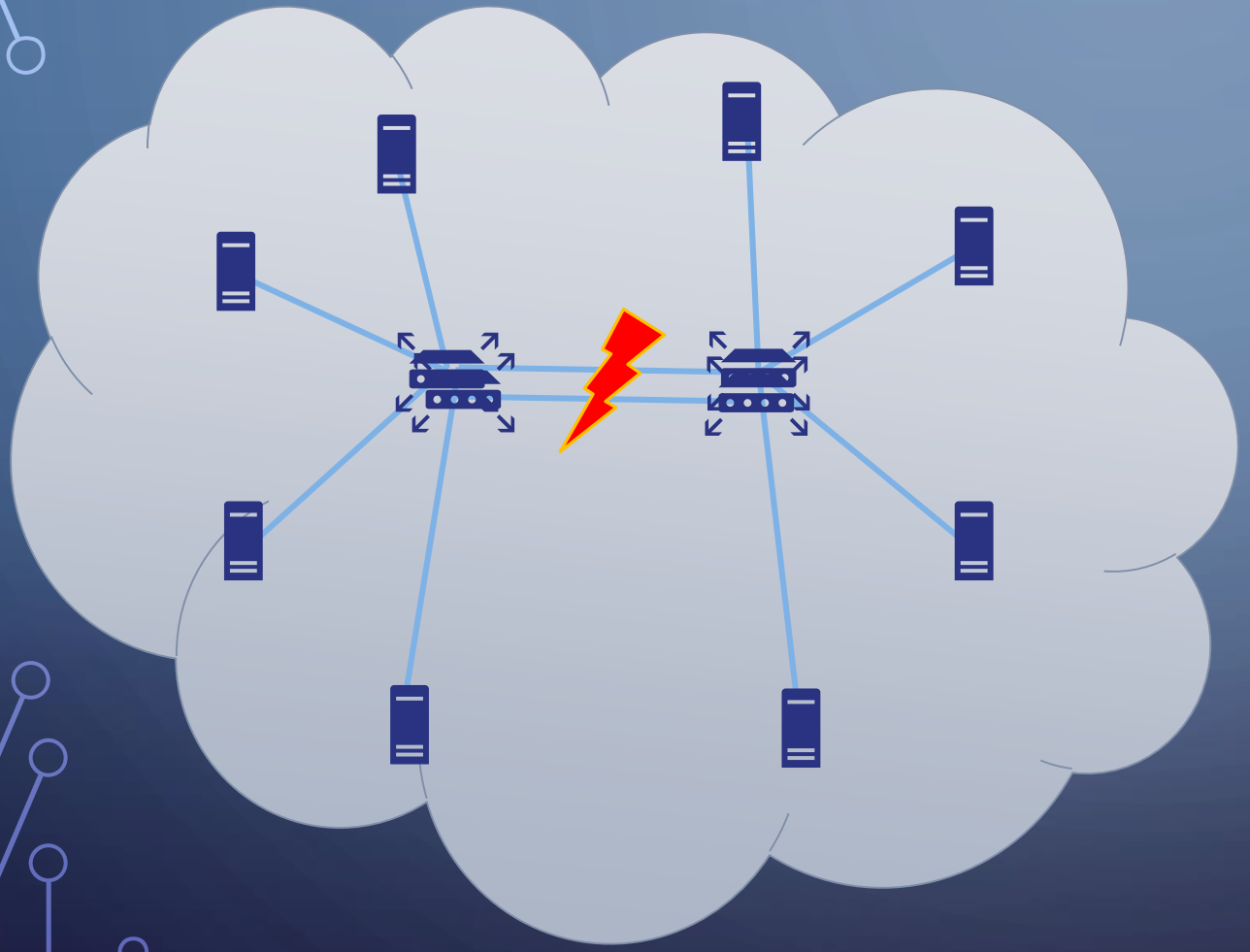
STICKY SESSION



Traps:

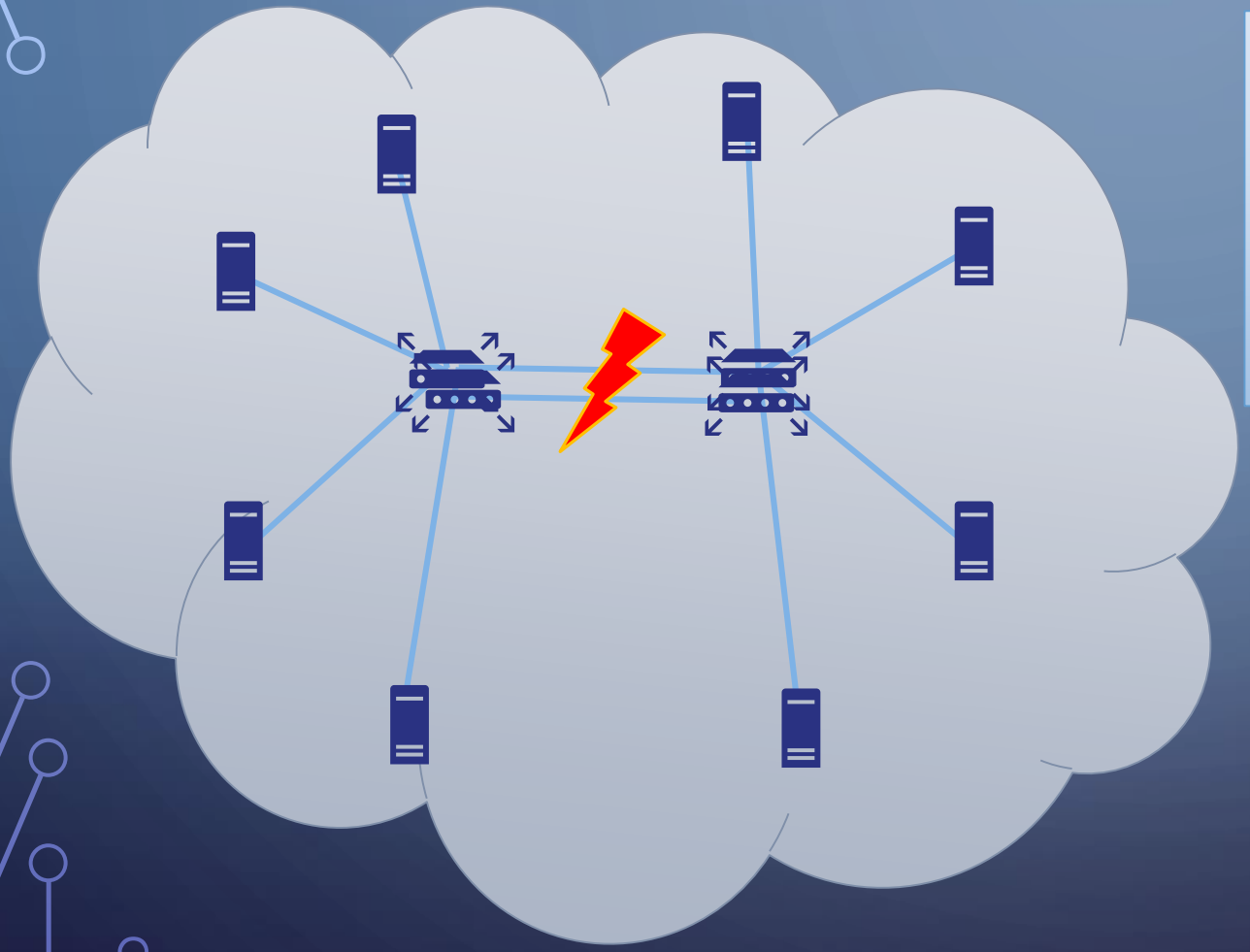
- Front-end server is unresponsive
- Multiple clients are directed to single machine
- Clients are accessing server farm via NAT
- Sticky token timeout. Two processes running in parallel:
  - clean-up on 1st server
  - new process on 2nd

BRAIN SPLIT



- The problem is not with the host, but the infrastructure
- Switch, router is disabled
- Shared communication channel is overloaded

BRAIN SPLIT

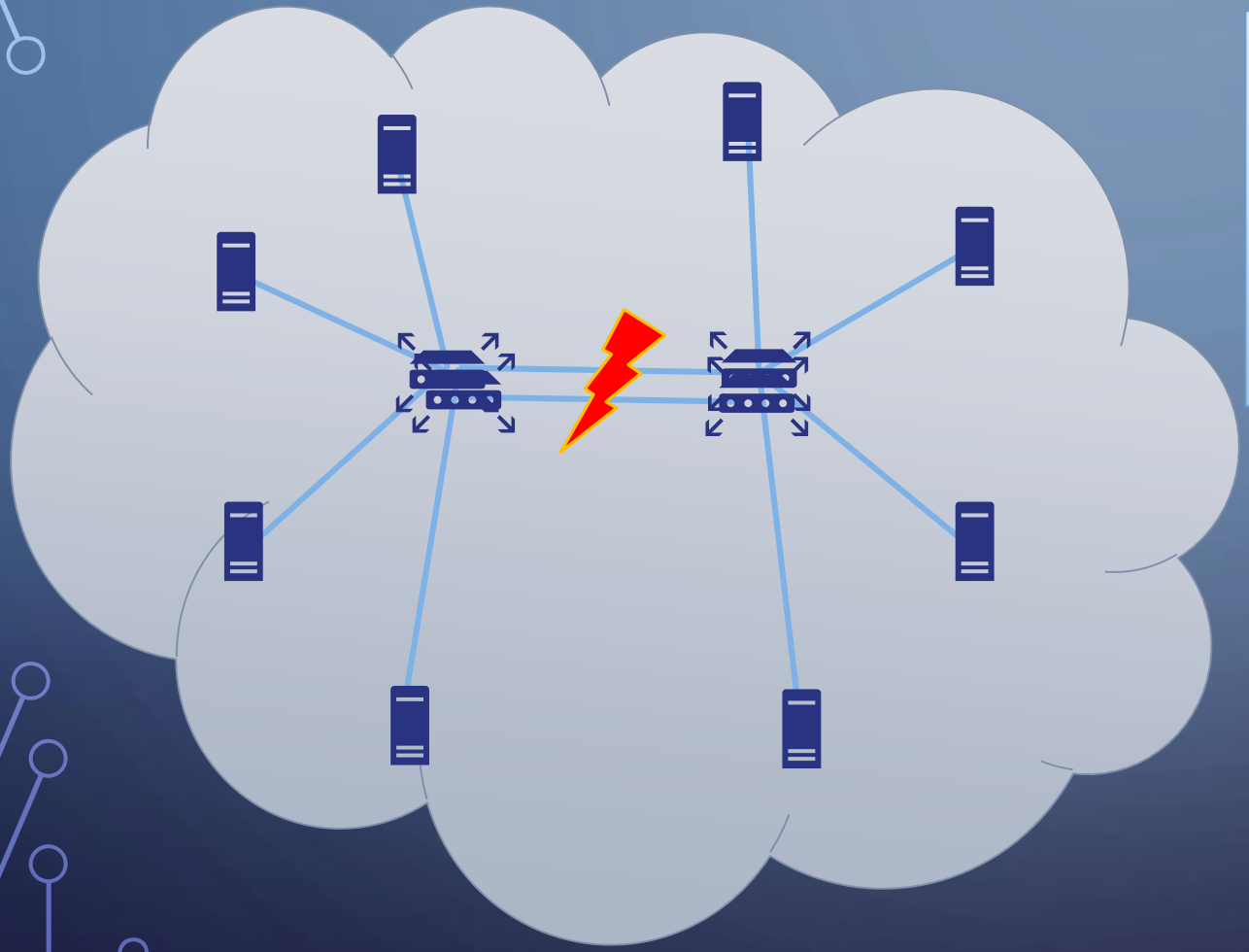


- Both sides think the other side is dead.

Problems to be solved:

- How to handle new client requests?
- How to merge potential conflicts, when connection is restored?

BRAIN SPLIT



- Both sides think the other side is dead.

Problems to be solved:

- How to handle new client requests?
- How to merge potential conflicts, when connection is restored?

- Consensus. Read only when there is $\leq \frac{(n)}{2}$ active nodes in group. Majority is not available.
- CRDT - Conflict-free replicated data types. Eventually consistent. Written data can be forgotten (wiped out) while merging.

WHAT YOU WROTE IS NOT WHAT YOU WILL READ

Scenario:

1. Create new Group in Active Directory.
2. Get the group from AD to verify that it was created.
3. Add Alice and Bob to the newly created group.

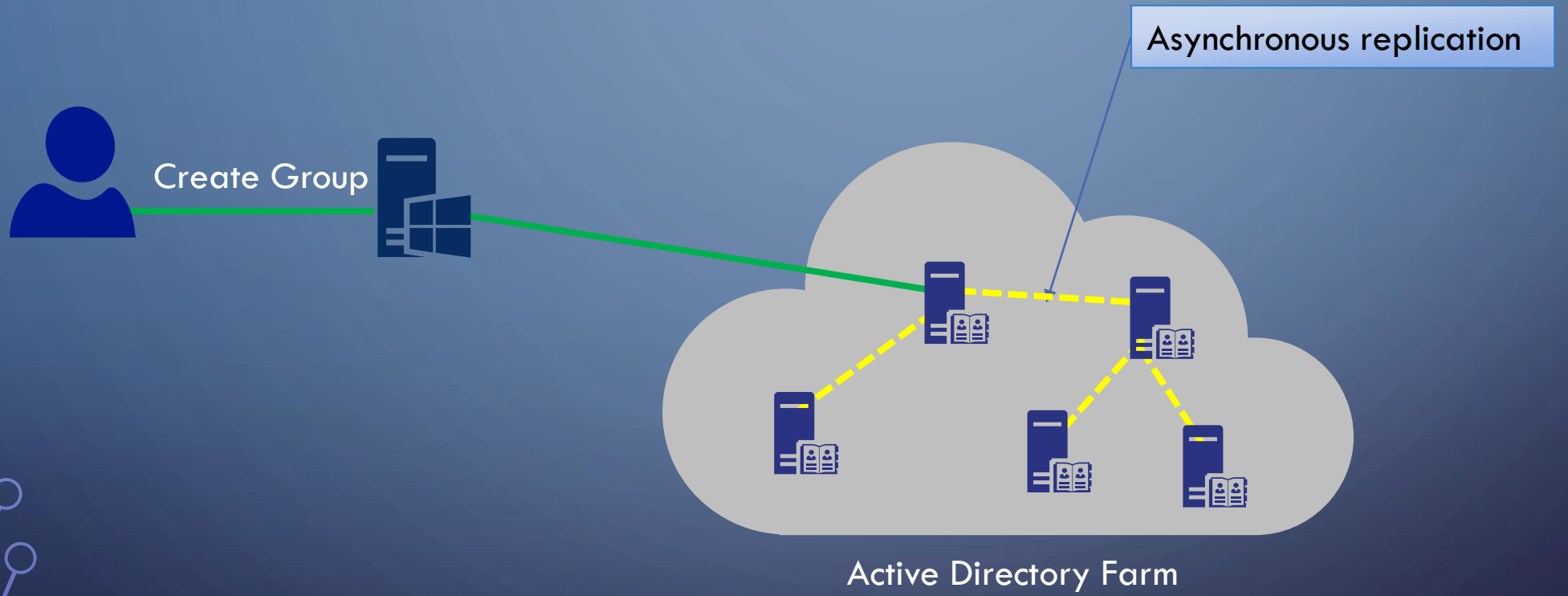
Expected result:

Group is created. Alice and Bob are assigned to group.

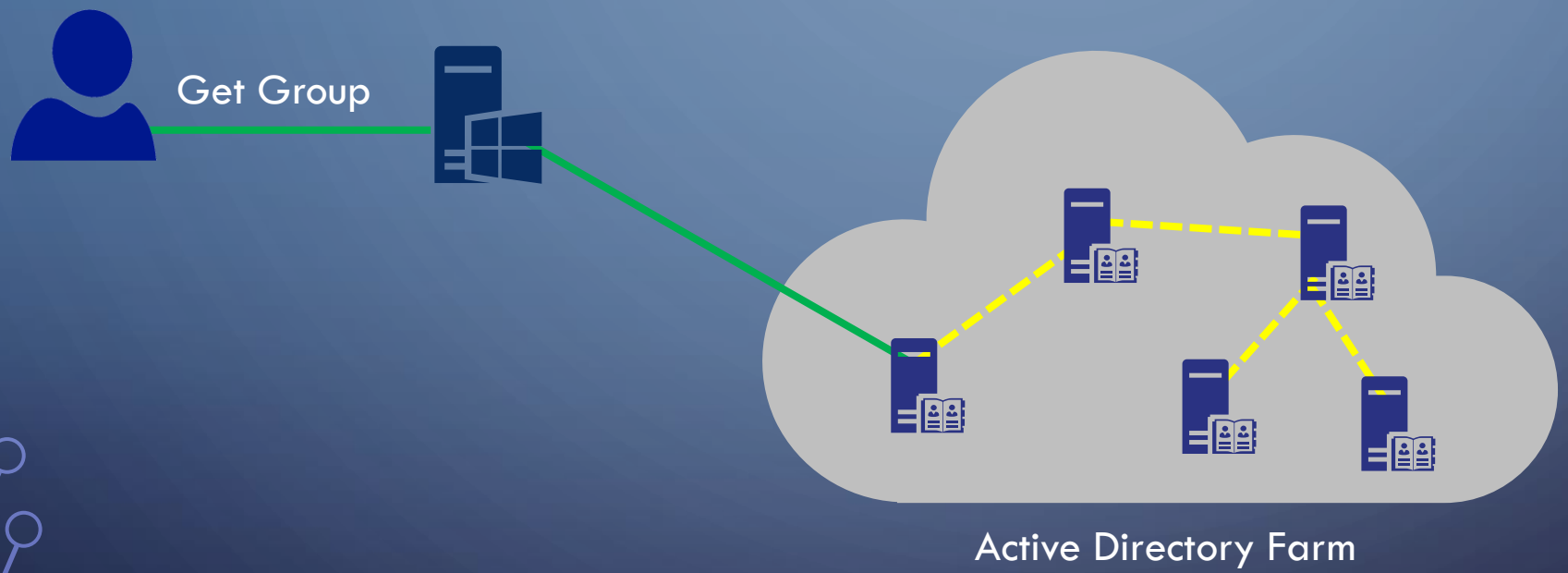
Reported error:

Group not found in **step 3** ☹

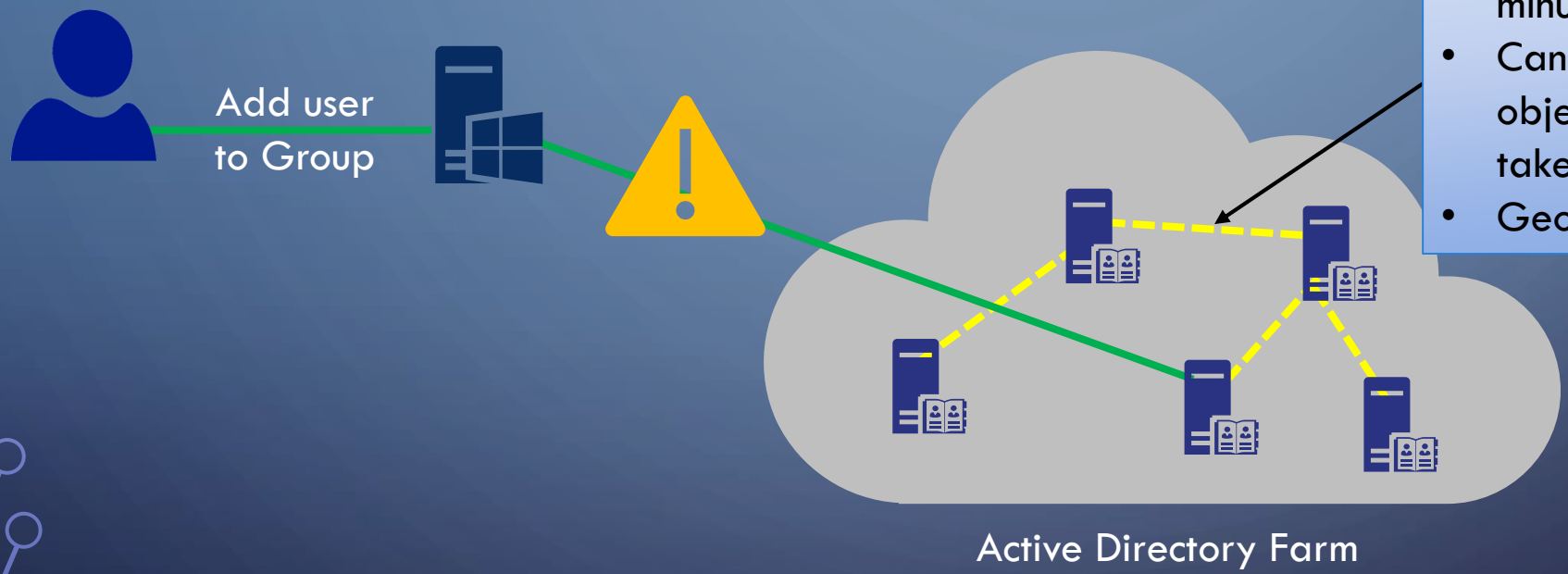
WHAT YOU WROTE IS NOT WHAT YOU WILL READ



WHAT YOU WROTE IS NOT WHAT YOU WILL READ

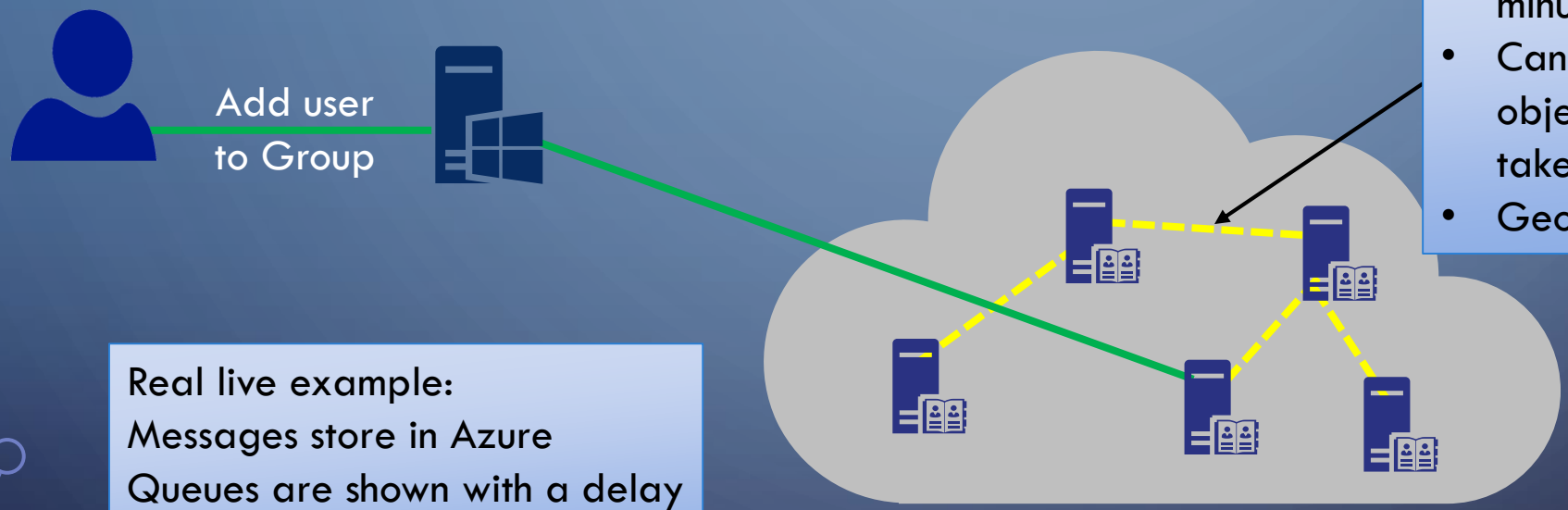


WHAT YOU WROTE IS NOT WHAT YOU WILL READ



- Replication can take up to 15 minutes
- Can be stopped on conflicted objects. Manual actions must be taken
- Geo-distributed replication

WHAT YOU WROTE IS NOT WHAT YOU WILL READ



- Replication can take up to 15 minutes
- Can be stopped on conflicted objects. Manual actions must be taken
- Geo-distributed replication

Real live example:
Messages store in Azure Queues are shown with a delay on the portal. Replication can be one of the reasons for that.

EVENTUAL CONSISTENCY

NOTE! System is never fully consistent when multiple processes are involved. Even communication with a single database fully supporting ACID is consistent only at the end.

CASE STUDY

- Single table
- Single command
- Single transaction

```
CREATE TABLE [dbo].[SqlCommitDemo](
[id] [int] NOT NULL PRIMARY KEY,
[data] [varchar](100) NOT NULL,
) ON [PRIMARY]
```

```
INSERT INTO [dbo].[SqlCommitDemo]
([id],[data])
VALUES (@id,@data)
```

(1) CREATE AND OPEN A CONNECTION

```
using (var con = new SqlConnection(Settings.Default.SqlConnection))
{
  con.Open();
  using (var trx = con.BeginTransaction())
  {
    using (var cmd = new SqlCommand(Resources.TestScript, con, trx))
    {
       cmd.Parameters.AddWithValue("@id", id);
       cmd.Parameters.AddWithValue("@data", data);
       cmd.ExecuteNonQuery();
     }
     trx.Commit();
  }
}
```

(2) BEGIN TRANSACTION

```
using (var con = new SqlConnection(Settings.Default.SqlConnection))
{
  con.Open();
  using (var trx = con.BeginTransaction())
  {
    using (var cmd = new SqlCommand(Resources.TestScript, con, trx))
    {
       cmd.Parameters.AddWithValue("@id", id);
       cmd.Parameters.AddWithValue("@data", data);
       cmd.ExecuteNonQuery();
     }
     trx.Commit();
  }
}
```

(3) EXECUTE SET OF COMMANDS

```
using (var con = new SqlConnection(Settings.Default.SqlConnection))
{
  con.Open();
  using (var trx = con.BeginTransaction())
  {
    using (var cmd = new SqlCommand(Resources.TestScript, con, trx))
    {
       cmd.Parameters.AddWithValue("@id", id);
       cmd.Parameters.AddWithValue("@data", data);
       cmd.ExecuteNonQuery();
    }
     trx.Commit();
  }
}
```

```
INSERT INTO [dbo].[SqlCommitDemo] ([id],[data])
VALUES (@id,@data)
```

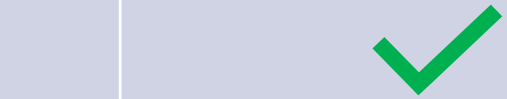
(4) COMMIT TRANSACTION

```
using (var con = new SqlConnection(Settings.Default.SqlConnection))
{
  con.Open();
  using (var trx = con.BeginTransaction())
  {
    using (var cmd = new SqlCommand(Resources.TestScript, con, trx))
    {
       cmd.Parameters.AddWithValue("@id", id);
       cmd.Parameters.AddWithValue("@data", data);
       cmd.ExecuteNonQuery();
     }
     trx.Commit();
  }
}
```

TEST - RESULTS

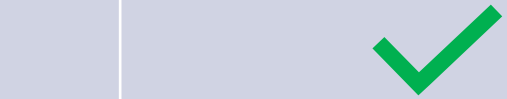
Test	Commit Results
Insert 1st	
Insert 2nd	
Insert 3rd	

QUIZ

Test	Commit Results
Insert 1st	
Insert 2nd	
Insert 3rd	

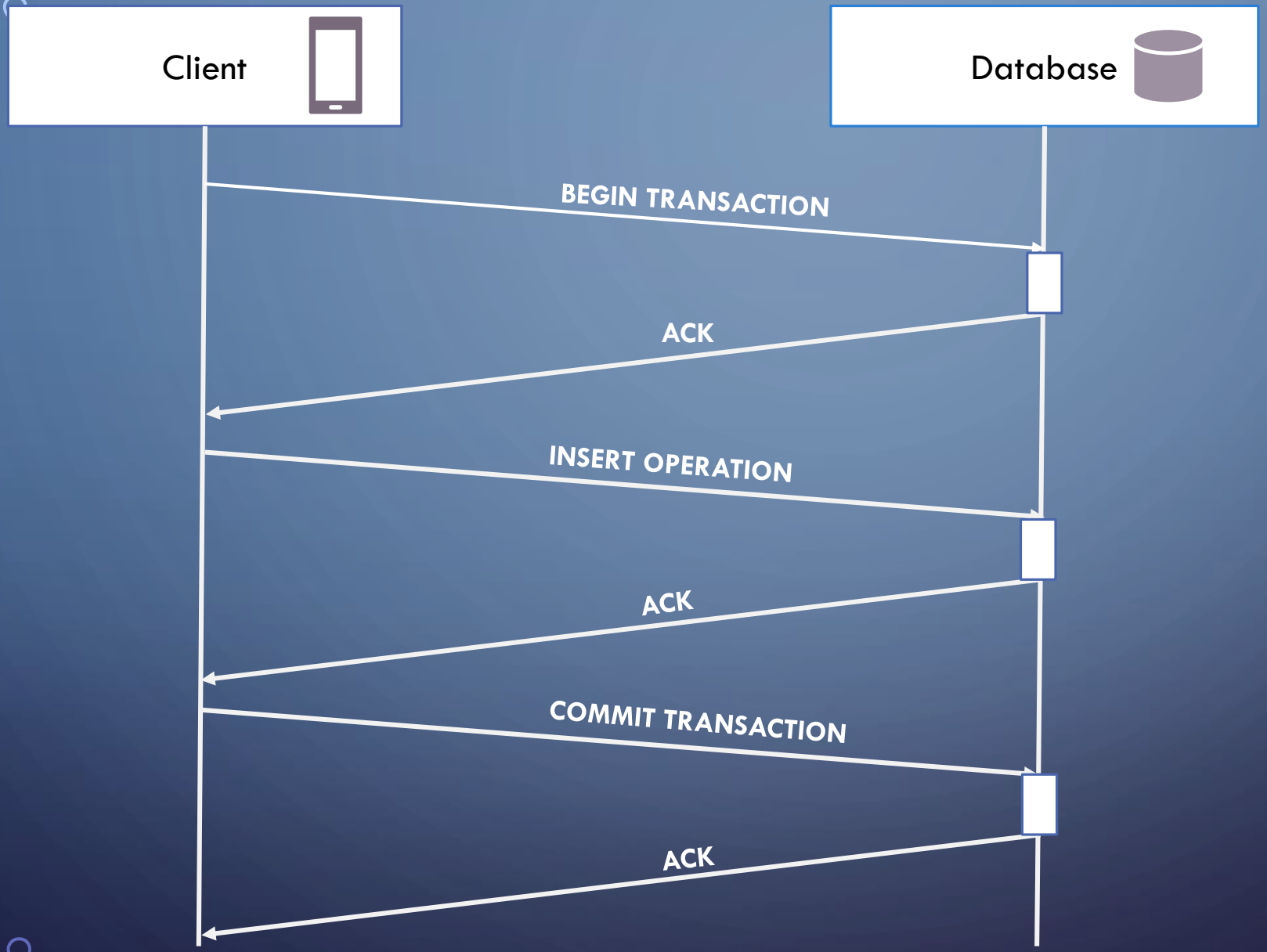
How many records are stored in the database?

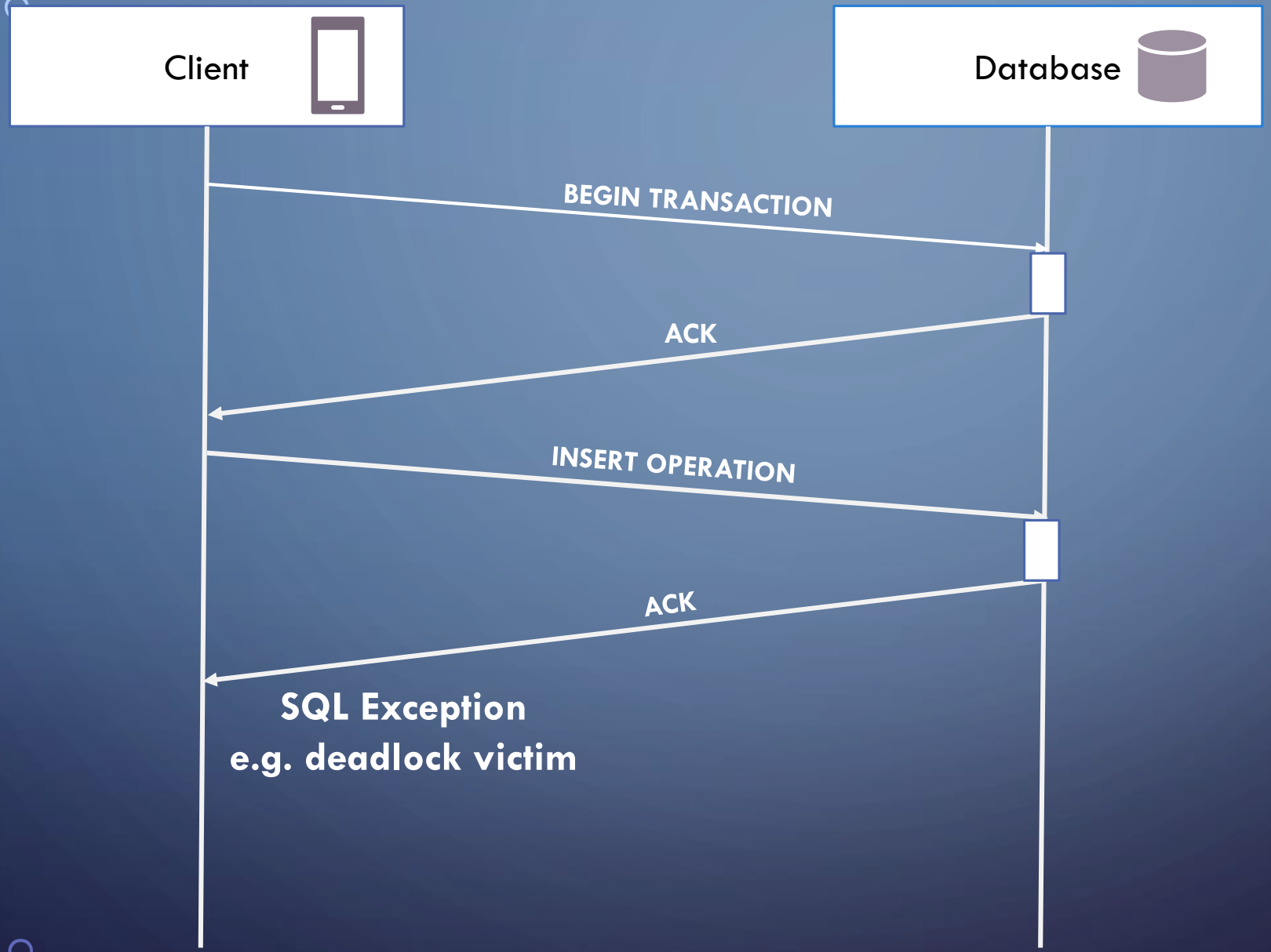
ANSWER

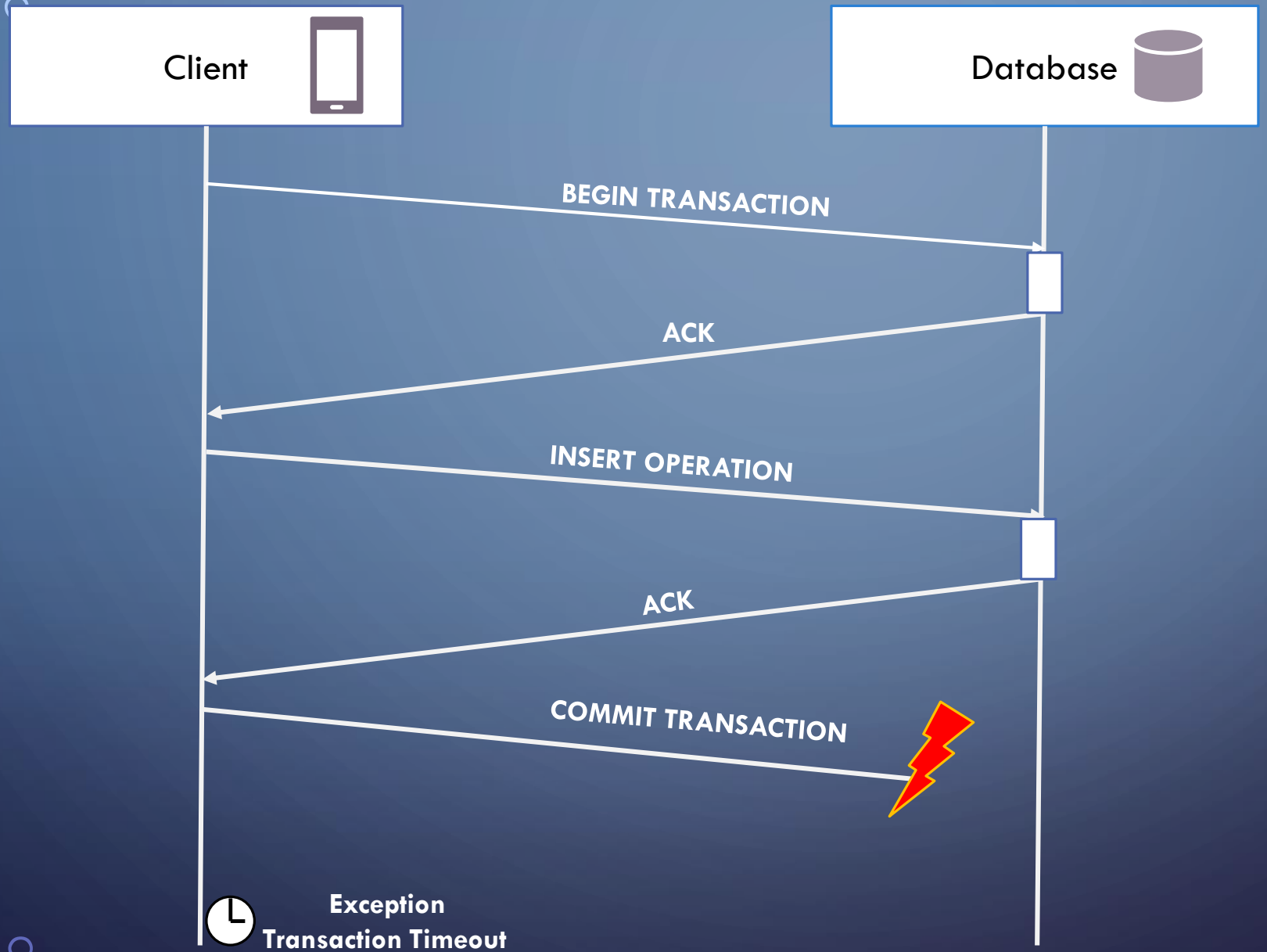
Test	Commit Results
Insert 1st	
Insert 2nd	
Insert 3rd	

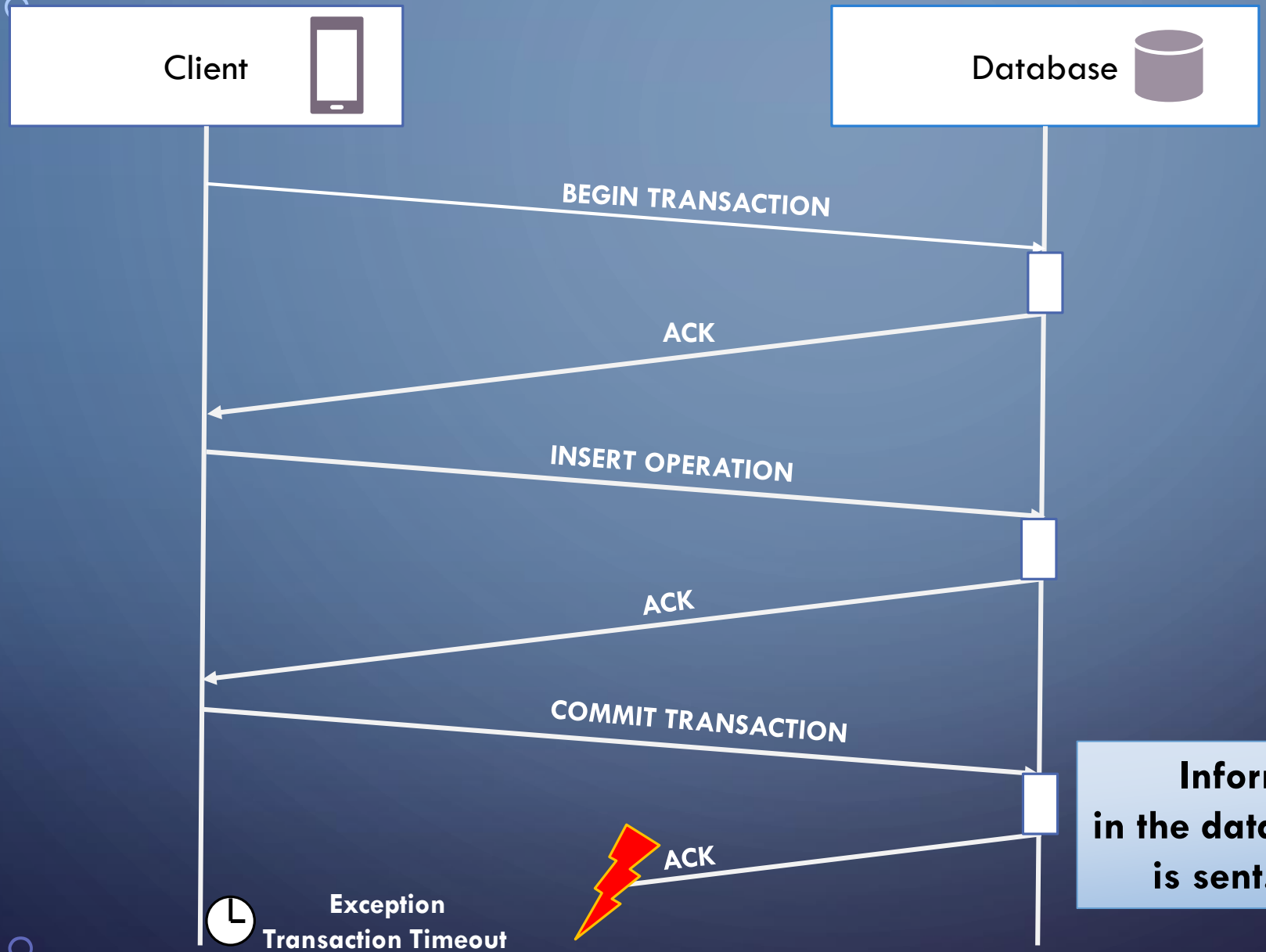
How many records are stored in the database?

Answer: Database has **at least 1** record.


Client
Database
BEGIN TRANSACTION
ACK
INSERT OPERATION
ACK
COMMIT TRANSACTION
ACK


Client
Database
BEGIN TRANSACTION
ACK
INSERT OPERATION
ACK
SQL Exception
e.g. deadlock victim


Client
Database
BEGIN TRANSACTION
ACK
INSERT OPERATION
ACK
COMMIT TRANSACTION
Exception
Transaction Timeout



Information MUST be saved in the database log before a response is sent. It cannot be rollbacked.

EVENTUAL CONSISTENCY - CONCLUSIONS

- System using single database sometimes is not consistent. Keeping consistency in distributed multi-database systems is more difficult.
- Each communication can be broken.
- Request or response packets can be lost.
- Simple retry does not solve all the problems.

MESSAGES NOT IN ORDER

- Knock – knock
- Message not in order
- Who is there?

MESSAGES NOT IN ORDER

- Servers have different processing speeds.
- Timeout/error can return message back to the queue
- Network latency can deliver messages in different moments
- Be careful of server responses not being in order. Create correlation id to match responses.

MESSAGES NOT IN ORDER

Real live example:
User double click a button in the browser. First received response can be an error 'record already exists'.

- Servers have different processing speeds.
- Timeout/error can return message back to the queue
- Network latency can deliver messages in different moments
- Be careful of server responses not being in order. Create correlation id to match responses.

DELIVERY OF EXACTLY ONE MESSAGE

- It is a myth. Network is not reliable.
- There are frameworks which emulate delivery of exactly one message. They implement this behavior:
  - by hiding usage of at least one message delivery pattern.
  - by involving Distributed Transaction Coordinator.

DELIVERY OF EXACTLY ONE MESSAGE

- It is a myth. Network is not reliable.
- There are frameworks which emulate delivery of exactly one message. They implement this behavior:
  - by hiding usage of at least one message delivery pattern.
  - by involving Distributed Transaction Coordinator.

- When you must deliver exactly one message, you have to build retry and de-duplication mechanism yourself.
- Keep in mind messages can be delivered not in order.

JSON EVERYWHERE

- Continuous serialization/deserialization process.
- Compressed data consumes processor time and memory. 1MB JSON compressed to 4kB still requires 1MB of RAM.
- Allocated memory must be released by Garbage Collector.

JSON EVERYWHERE

- Continuous serialization/deserialization process.
- Compressed data consumes processor time and memory. 1MB JSON compressed to 4kB still requires 1MB of RAM.
- Allocated memory must be released by Garbage Collector.

- Consider binary formatters.
- Redesign your data model.
- Zero Formatter – to minimize allocation.

RESOURCE ASYMMETRY

- When sharing multiple resources like disks, computing power, network bandwidth.
- Heavy load on processor or disk of single virtual machine can impact other virtual machines.
- Overloaded network, can make your services unavailable.

RESOURCE ASYMMETRY

- When sharing multiple resources like disks, computing power, network bandwidth.
- Heavy load on processor or disk of single virtual machine can impact other virtual machines.
- Overloaded network, can make your services unavailable.

Real live example:
Data scientist overloaded physical host by heavy disk load. All virtual machines running on this host became almost unresponsive.

DOMINO EFFECT

- What is the maximum throughput of your system?
- What happens when the system is overloaded?

DOMINO EFFECT

- What is the maximum throughput of your system?
- What happens when the system is overloaded?

- 503 Service Unavailable is not a solution for web application. Multiple requests of a browser are sent during a single page load.
- A wise retry policy must be implemented at the client side.
- Consider switching off less important systems.

EXTREME – THE ONLY VALID OPTION

- Reactive processing
- Batch processing

EXTREME – THE ONLY VALID OPTION

- Reactive processing
- Batch processing

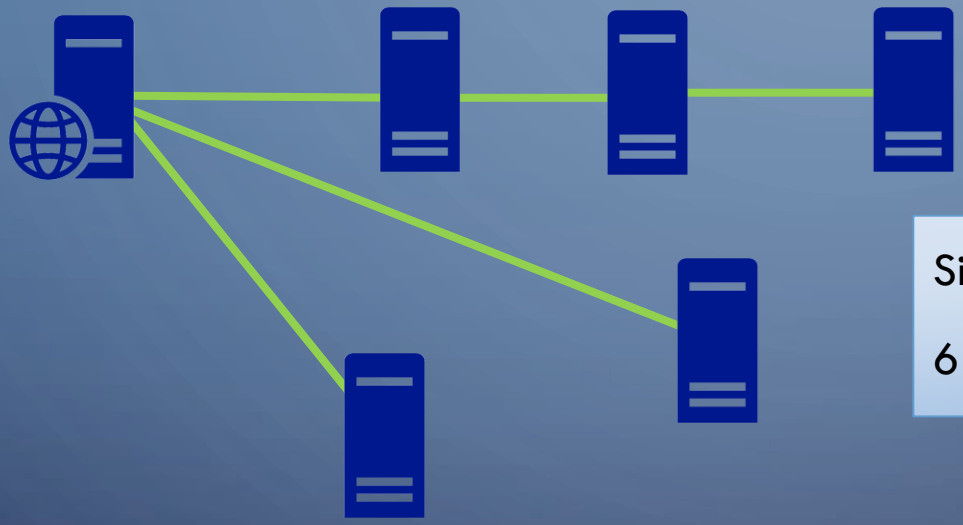
- Both have advantages and disadvantages. E.g. reactive has shorter response time, but its processing can generate $O(\log n)$ or $O(n)$ computational complexity.
- Micro-batching can increase global system performance, but can impact latency.

RELIABILITY

- Cloud provider
- Your local network provider

- Is your local network prepared for increased traffic? QoS.
- Sticky session and NAT problem. Limited port numbers to be used on client side.
- What is the cost of losing local connection?
- Is your system deployed in multiple zones?
- What is your backup plan?

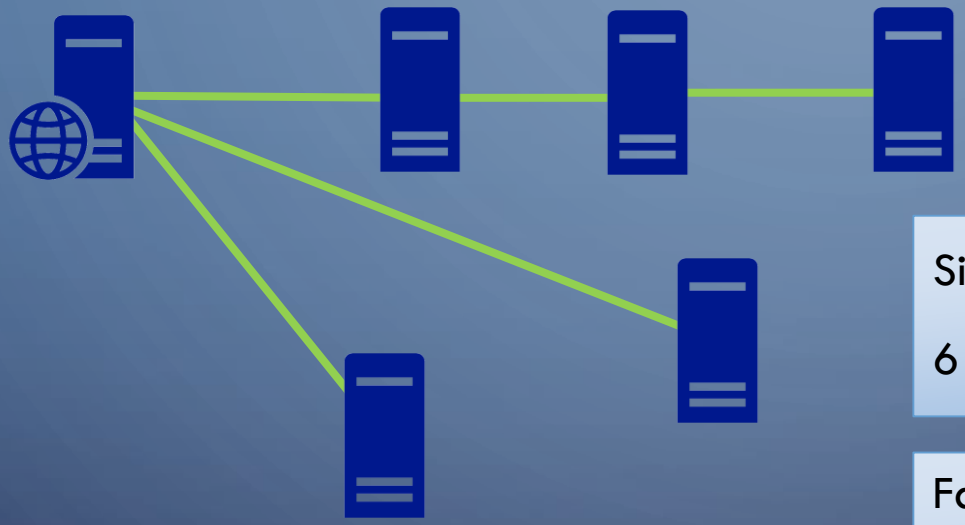
RELIABILITY – POINT OF FAILURE



Single server availability $99.99\% \approx 52 \frac{minutes}{year}$

6 servers $99.94\% \approx 5.25 \frac{hours}{year}$

RELIABILITY – POINT OF FAILURE



Single server availability $99.99\% \approx 52 \frac{minutes}{year}$

6 servers $99.94\% \approx 5.25 \frac{hours}{year}$

Fast failover mechanism is required. Each node must be duplicated. See Circuit Breaker approach used by Netflix.

RELIABILITY – HUMAN ERROR

- Do you remember to renew all your Security Certificates?
- Have you defined disaster recovery plan?
- Do you control your costs generated at your Cloud Provider?

RELIABILITY – HUMAN ERROR

- Do you remember to renew all your Security Certificates?
- Have you defined disaster recovery plan?
- Do you control your costs generated at your Cloud Provider?

- Business is not aware of SSL certificates validity.
- Cloud resources must be monitored.
- Usage forecast must be reported to business in advance. Remember about corporate procedures slowing down execution.

BIG BANG DEPLOYMENT

PROS

- All clients have the newest version.
- No need to maintain the old version.

CONS

- When the failure occurs?
  - You will get more disappointed users
  - Your support/OPS will be overloaded by duplicated bug reports.

BIG BANG DEPLOYMENT

PROS

- All clients have the newest version.
- No need to maintain the old version.

CONS

- When the failure occurs?
  - You will get more disappointed users
  - Your support/OPS will be overloaded by duplicated bug reports.

- Consider Canary Tests.
- Select a small group of end users for bringing a new release.
- Design how to switch on/off features for selected users.

NEW RELEASE (WEB SERVER)

- In the 24/7 distributed systems, there is no place for content replace.
- Deprecated version must be kept for a while.
- Consider that web browser has its own caching strategy.
- Consider that while a new version is being deployed, client application can download subcomponents prepared for previously released version.

NEW RELEASE (MOBILE APPLICATION)

- Real live example: Maps.me - over 1 millions downloads
- Map of European country contains about 300 MB of data.
- Single release cost $300 \cdot 10^6 \cdot 10^6 = 300 \cdot 10^{12} \approx 300TB$ to be transmitted to all clients.
  - $10\ \frac{GB}{s}$ average throughput $\approx$ 1 hours to update all devices
  - $1\ \frac{GB}{s}$ average throughput $\approx$ 10 hours to update all devices
- What will happen when an application has more downloads ?
- 4G network? 300 MB can be a killer for your clients budget. (Limits in New Zeeland, Australia, Galapagos)

PROCESSING TIME

- CPU ~ 1ns
- Memory ~100 us
- Local Network ~1ms
- External Network ~50ms
- Second side of the world ~0.5s
- When you add host restarts we can talk about minutes

Scale factor 10^9. Distributed system works well when workflows can be run in parallel. No synchronization points are involved.

SUMMARY

- Distributed programming has benefits when the solution requires large scale.
- Distributed systems are not a universal solution when your system is slow.
- If your application unable to work on a single machine it will die on servers farm.
- FAAS, PAAS is serverless, but logically only. The real server farm infrastructure hosts them.

ADDITIONAL RESOURCES

- Udi Dahan - NServiceBus
- Petabridge – Akka.NET
- Netflix – Chaos Engineering
- Martin Kleppmann

WHERE YOU CAN FIND ME?

- Web
  - https://lastboardingcall.pl
  - https://mrmatrix.net
- GitHub
  - https://github.com/MariuszKrzanowski
- Tweeter
  - @KrzanowskiM
- Meetups
  - Warszawska Grupa .NET

Mariusz Krzanowski

Q & A

Thank you

WHERE YOU CAN FIND ME?

- Web
  - https://lastboardingcall.pl
  - https://mrmatrix.net
- GitHub
  - https://github.com/MariuszKrzanowski
- Tweeter
  - @KrzanowskiM
- Meetups
  - Warszawska Grupa .NET

Mariusz Krzanowski