ECS656U/ECS796P Distributed Systems

What we have seen so far

Consensus:

- Allows collection of machines to work as coherent group
- Continuous service, even if some machines fail

Paxos:

- Distributed consensus algorithm
- Safety
- Eventual liveness

What this lecture is about

- Raft
- Introduction to cloud computing

Raft

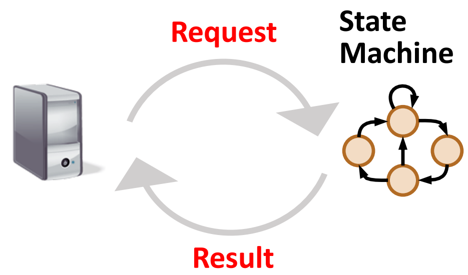
Many slides from Ion Stoica presentation:
(https://ucbrise.github.io/cs262a-spring2018/)

Introduction

- Raft is a consensus algorithm
  - Primary design goal: understandability (intuition, ease of explanation)
  - Complete enough that can be easily applicable in real implementations

  - This results in a different problem decomposition with respect to Paxos!

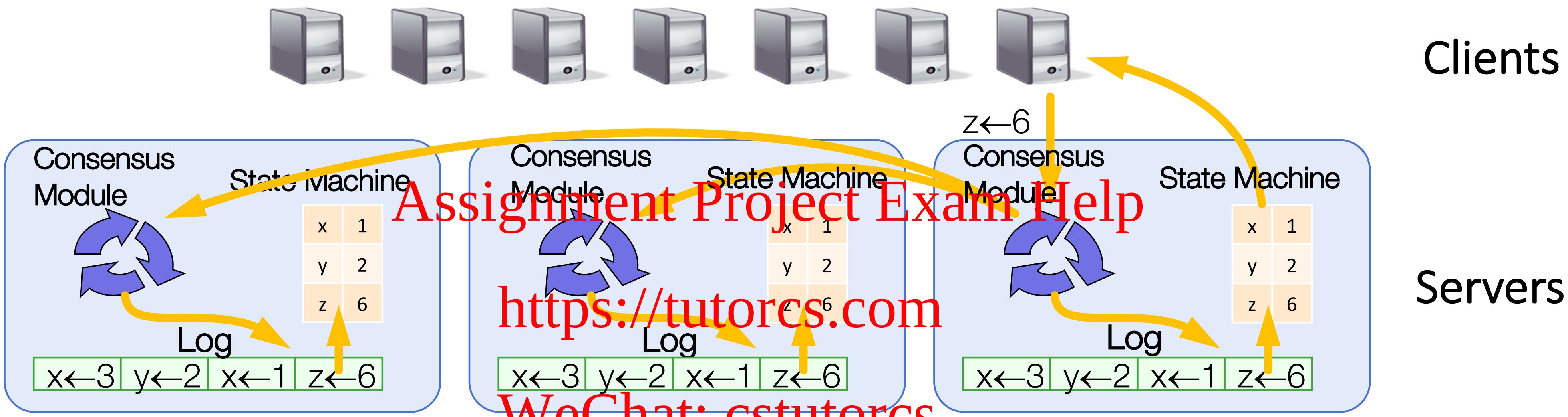
Introduction



- Consensus algorithms are commonly used in the context of "replicated state machines"
  - State machine: a program that respond to an external stimuli and manage an internal state
  - Most of today's services are based on state machines (Memcached, RAMcloud)

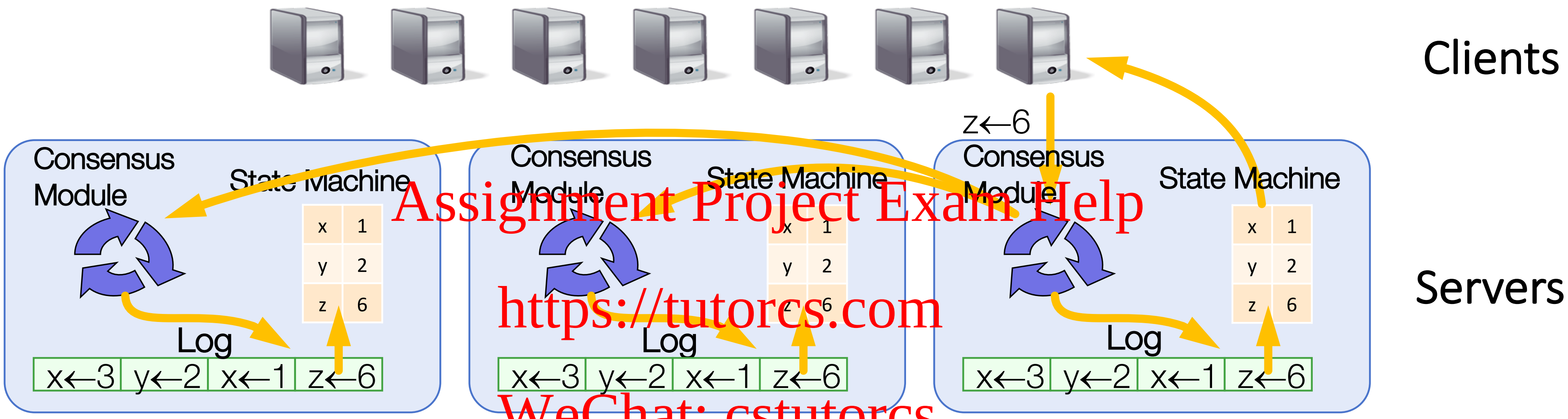
- How to build reliable state machines? You replicate them on different servers!

Replicated State Machines



- The idea: all the machines execute the same set of commands, with the same stimuli in the same order -> all they must produce the same result
- This shall be so reliable to survive the failure of some machines
- HOW? Keep a replicated log ⇒ replicated state machine

Replicated State Machines



- Consensus module ensures proper log replication
- System makes progress provided any majority of servers are up
- Failure model: fail-stop (not Byzantine), delayed/lost messages

How to do that? The Paxos answer..

1. Proposers: choose unique proposal number (Pn)
2. Acceptors: if Pn > any previous stored number (Ps), then reply back with Ps and the previously accepted value (V).
3. Proposer: if it gets a majority then select value V, if none choose own value, and send back "accept-request" (Pn,V)
4. Acceptor: is Pn > Ps? If so, reply with accept!

Before moving to Raft…

“There are significant gaps between the description of the Paxos algorithm and the needs of a real-world system…the final system will be based on an unproven protocol.”

– Google Engineers

Paxos Made Live - An Engineering Perspective (2006 Invited Talk)

Tushar Chandra, Robert Griesemer, and Joshua Redstone

Google Inc.

ABSTRACT

We describe our experience in building a fault-tolerant database using the Paxos consensus algorithm. Despite the existing literature in the field, building such a database proved to be non-trivial. We describe selected algorithmic and engineering problems encountered, and the solutions we found for them. Our measurements indicate that we have built a competitive system.

Categories and Subject Descriptors

D.4.5 [**Operating systems**]: Reliability—*Fault-tolerance*; B.4.5 [**Input/output and data communications**]: Reliability, Testing, and Fault-Tolerance—*Redundant design*

database is just an example. As a result, the consensus problem has been studied extensively over the past two decades. There are several well-known consensus algorithms that operate within a multitude of settings and which tolerate a variety of failures. The Paxos consensus algorithm [8] has been discussed in the theoretical [16] and applied community [10, 11, 12] for over a decade.

We used the Paxos algorithm (“Paxos”) as the base for a framework that implements a fault-tolerant log. We then relied on that framework to build a fault-tolerant database. Despite the existing literature on the subject, building a production system turned out to be a non-trivial task for a variety of reasons:

- While Paxos can be described with a page of pseudo-

Before moving to Raft…

"There are significant gaps between the description of the Paxos algorithm and the needs of a real-world system…the final system will be based on an unproven protocol."

– Google Engineers

- Paxos has dominated discussion for 25 years
  - Hard to understand (why does this work? what is the purpose of each phase?)
  - Incomplete (only agrees on single value, it does not guarantee that we converge on a value: if it converges, it will be just one value)

Before moving to Raft…

- Hard to implement reliably (how to choose proposal value?)

Hard to understand + Hard to implement reliably =

Not a good foundation for practical implementations!

Raft: designing for understandability..

- Main objective of RAFT
  - Which design decision is the easiest to understand?
- Techniques that were used include
  - Dividing problems into smaller problems (that are easier to understand)
  - Reducing the number of system states to consider (removing as much as possible “if statements”

Raft overview

1. Leader election
   - Select one of the servers to act as cluster leader
   - Detect crashes, choose new leader

2. Log replication (normal operation)
   - Leader takes commands from clients, appends them to its log
   - Leader replicates its log to other servers (overwriting inconsistencies)

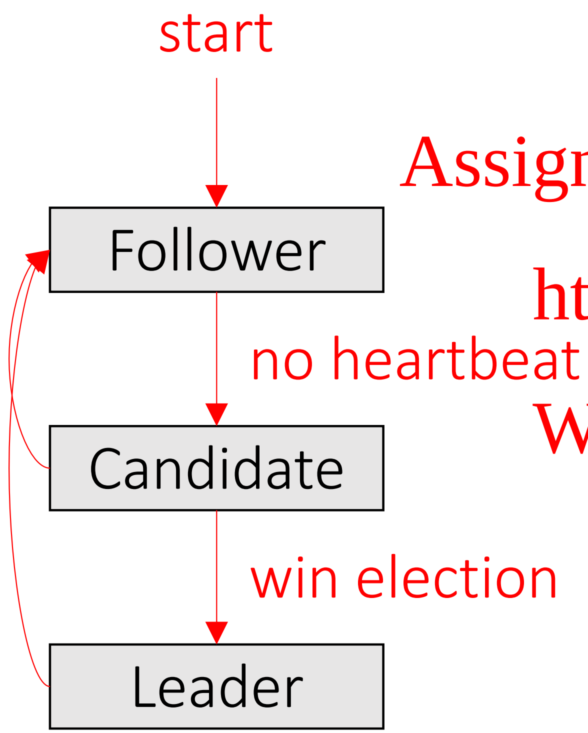
3. Safety
   - Only a server with an up-to-date log can become leader

Raft basics: servers

- A Raft cluster consists of several servers (remember the replicated state machine)

- Each server can be in one of three states:
  - Follower
  - Candidate
  - Leader

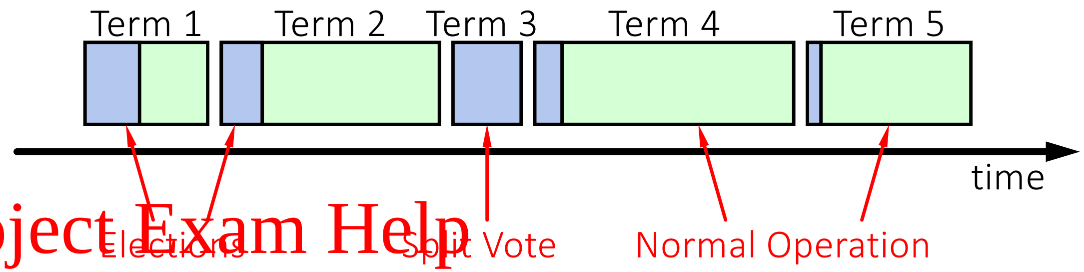
Raft basics: servers



- Follower is passive but expect regular heartbeats

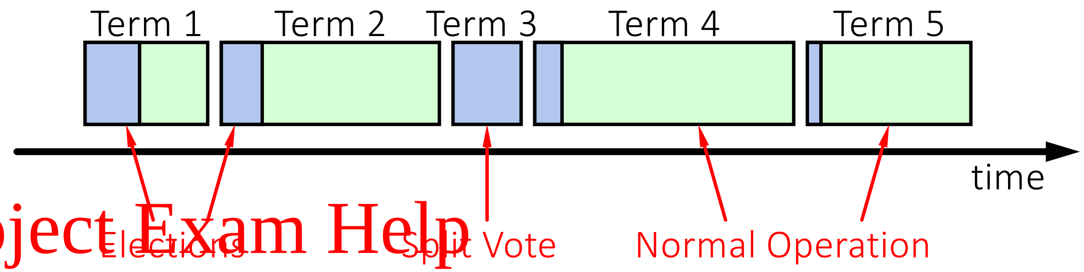
- Candidate tries to get elected as leader
- Leader replicates its log and send regular heartbeats to maintain leadership

Raft basics: terms



- Terms are epochs of arbitrary length
  - **Start** with the election of a leader
  - **End** when
    - Leader becomes unavailable
    - No leader can be selected (split vote)
- Servers do not have global view of the entire system. Servers do not have global view of terms and might be see the progressing of terms at different times. <u>How to deal with this?</u>

Raft basics: terms



- Each server:
  - Keeps what they think the current term is
  - Constantly exchange this information
  - Every Response-Request message include the Term the server thinks we are on
  - If a machine finds out that there is a more updated term, then it has an identity crisis and (1) updates its term and (2) become <u>**follower**</u>
  - If a machine receives a request with an old term, then it replies saying “dude, you are too old now!”

Raft basics: RPC

- RPC: Remote Procedure Call is the request-response protocol used in Raft

- Servers communicate though idempotent RPCs

- RequestVote

  - Initiated by candidates during elections

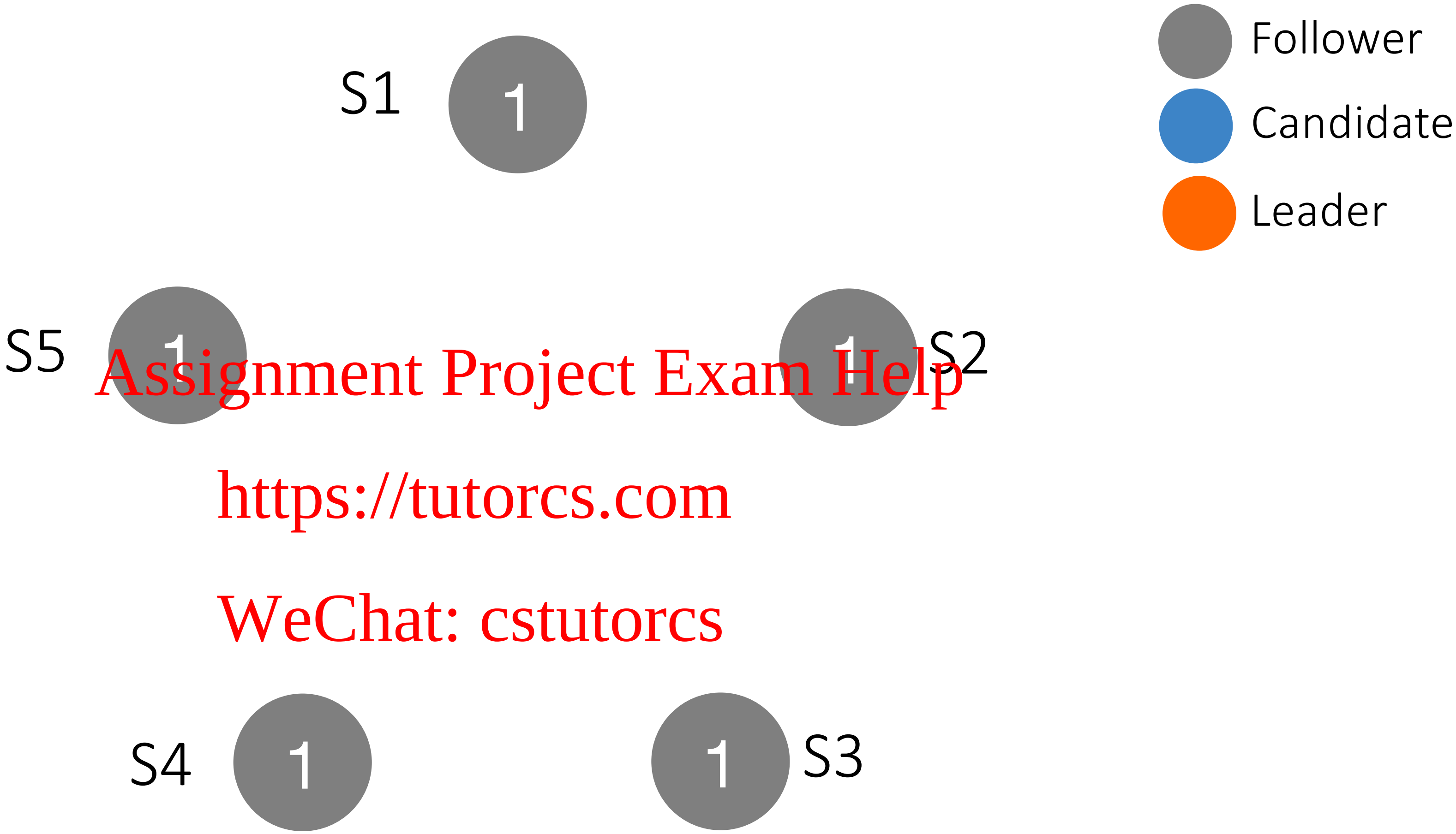
- AppendEntry: Initiated by leaders to
  - Replicate log entries
  - Provide a form of heartbeat
    - Empty AppendEntry( ) calls

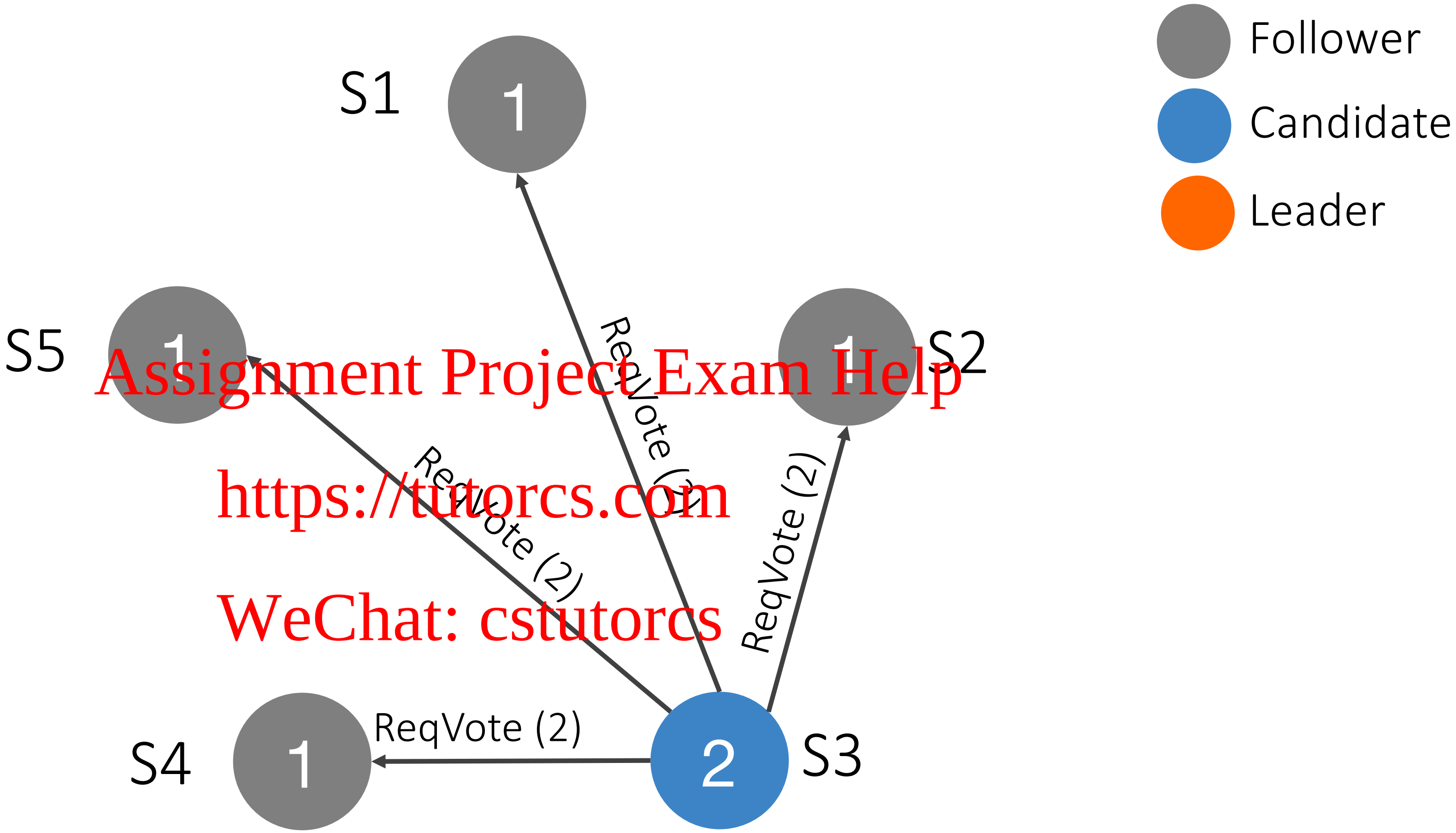
Leader election

- Servers start being ***followers***

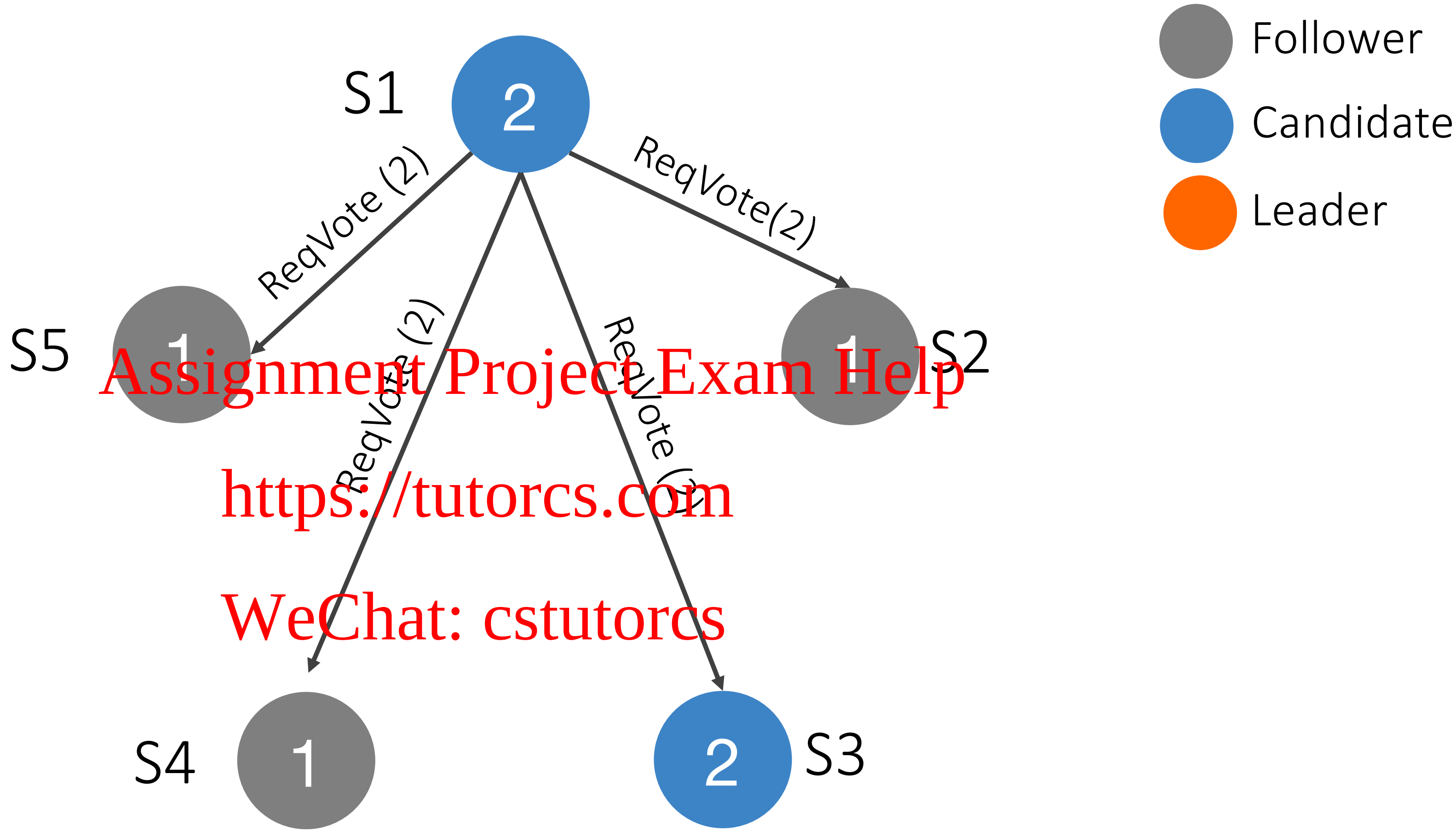
- Remain followers as long as they receive valid RPCs from a leader or candidate

- When a follower receives no communication over a period (the ***election timeout***), it starts an election to pick a ***new leader***

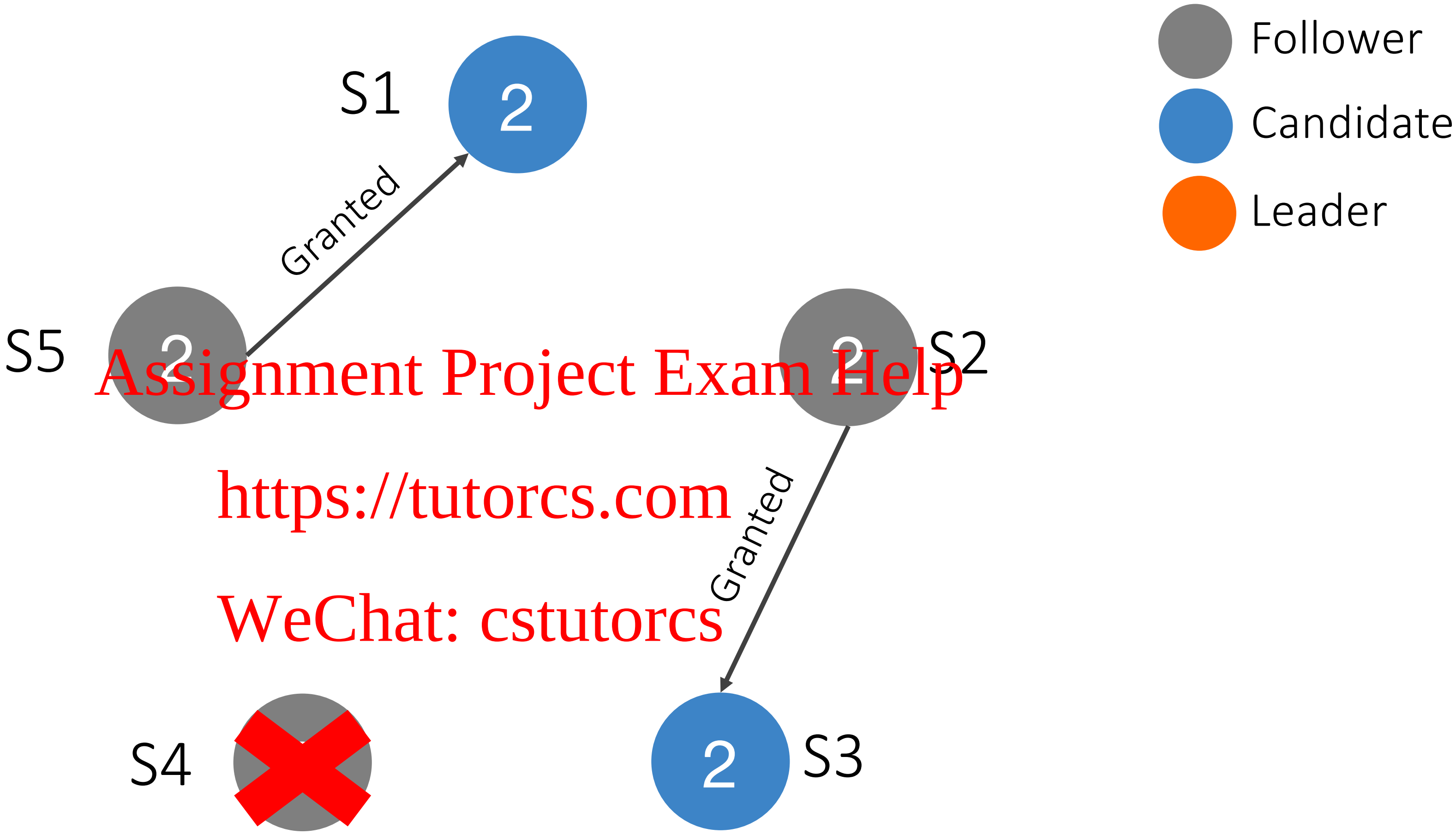

S1
1
S5
1
S2
1
S4
1
S3
1
Follower
Candidate
Leader



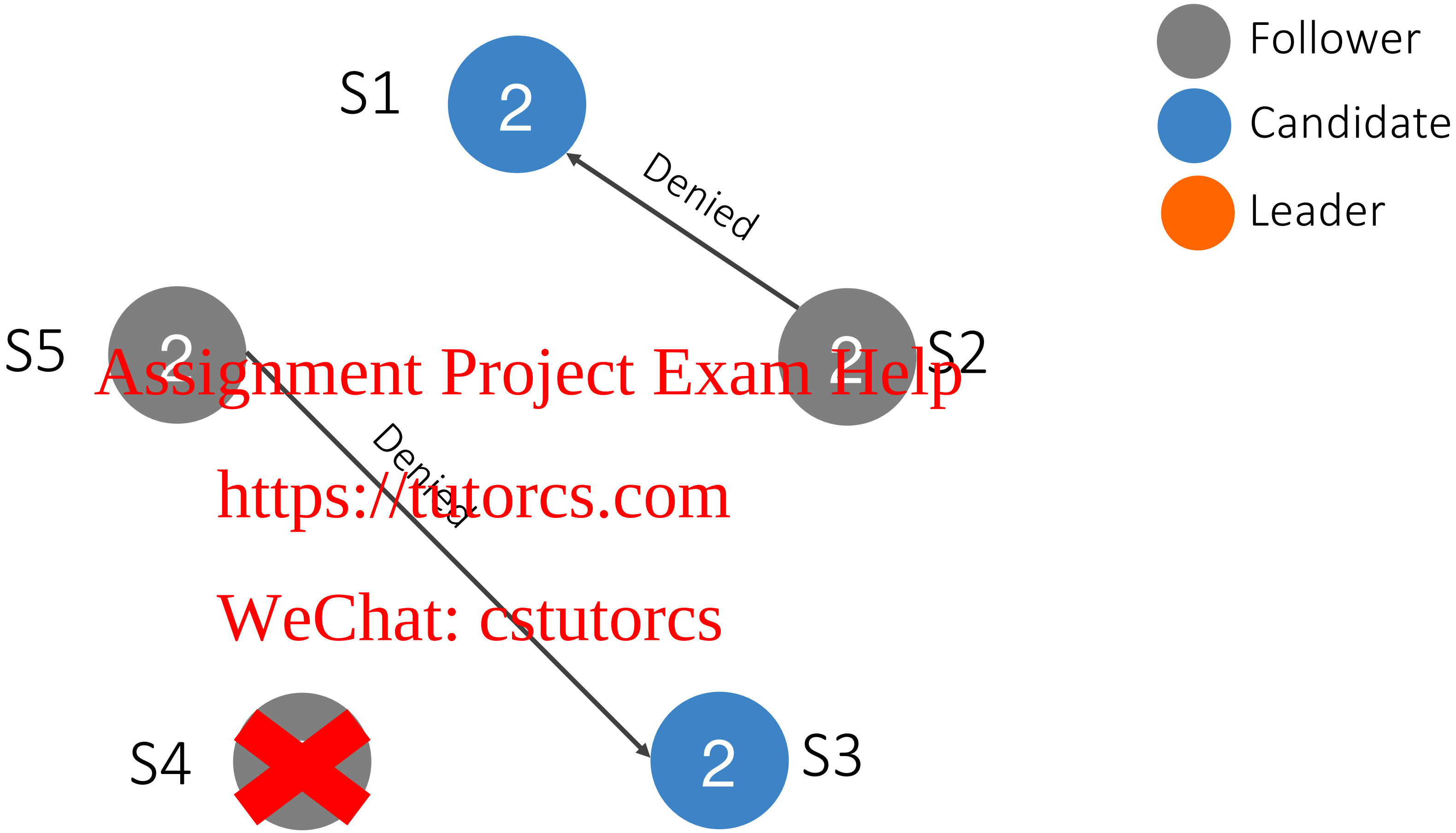
S3 timeouts, switch to candidate state,
increment term, vote itself as a leader and ask everyone else to confirm

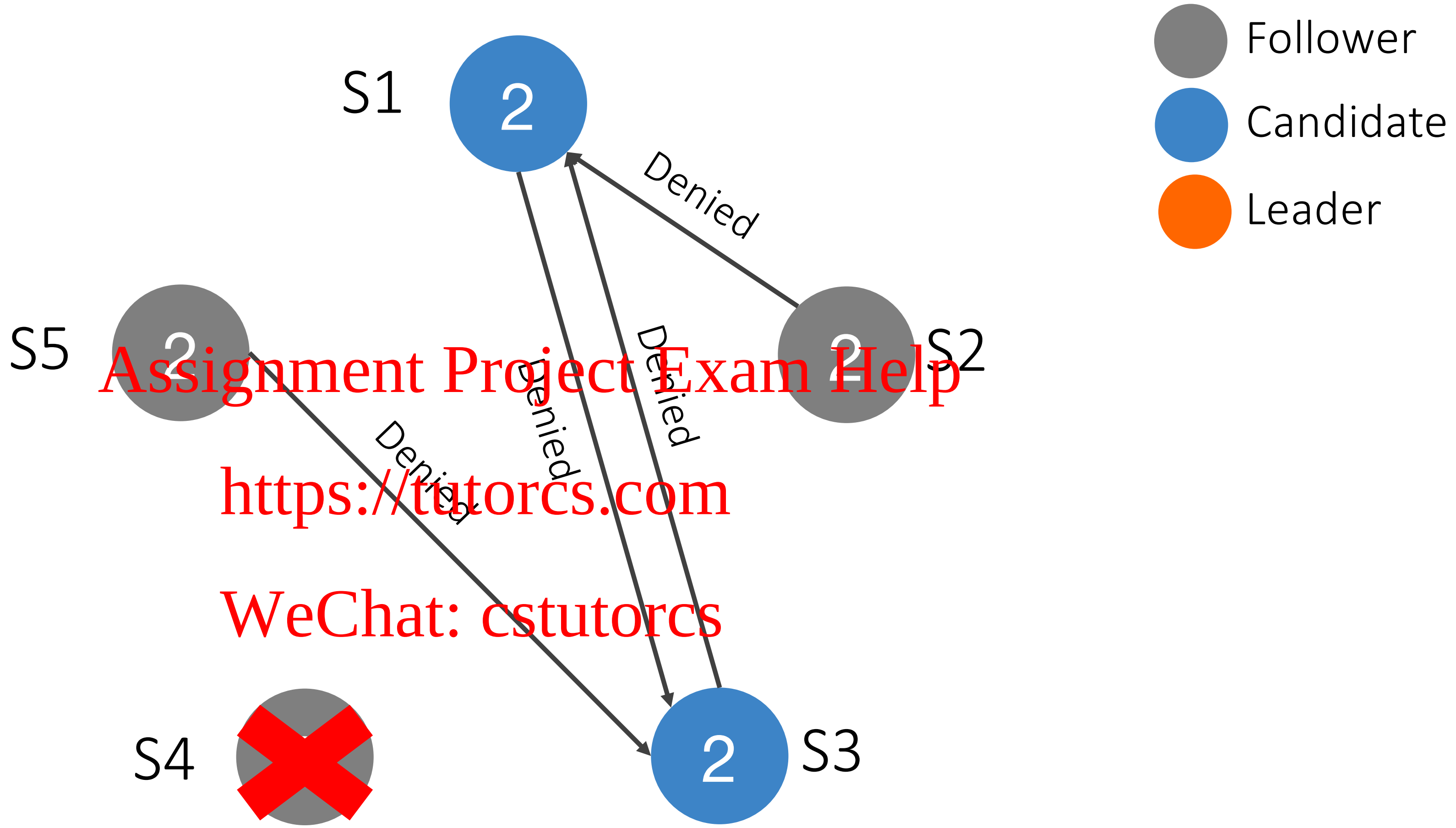


Concurrently S1 timeouts, switch to candidate state,
increment term, vote itself as a leader and ask everyone else to confirm

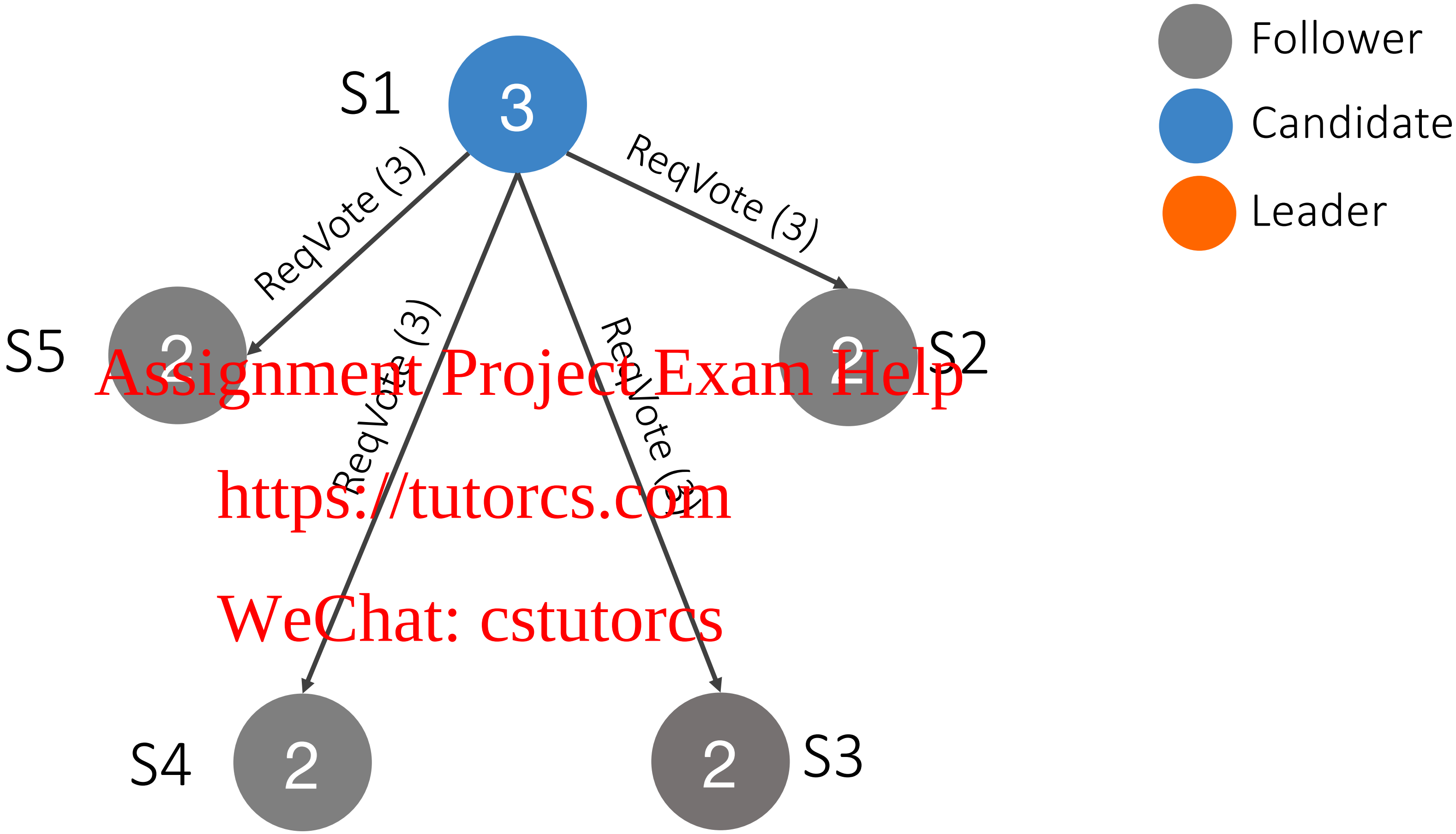


Let’s assume that S4 crashes and S5 grant vote to S1, while S2 grants vote to S3

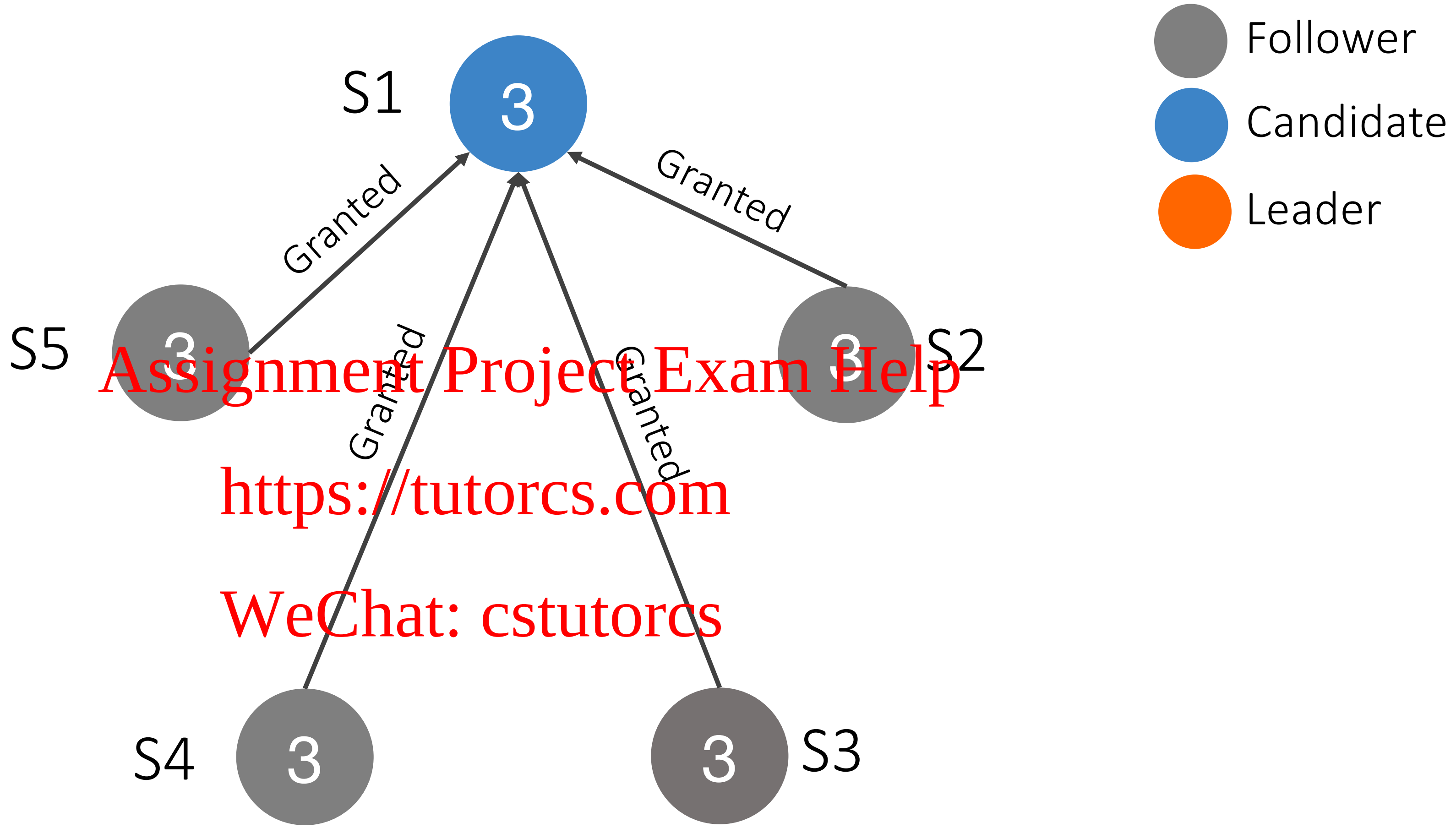

S1
2
Denied
S5
2
S2
2
Denied
S4
S3
2
Follower
Candidate
Leader



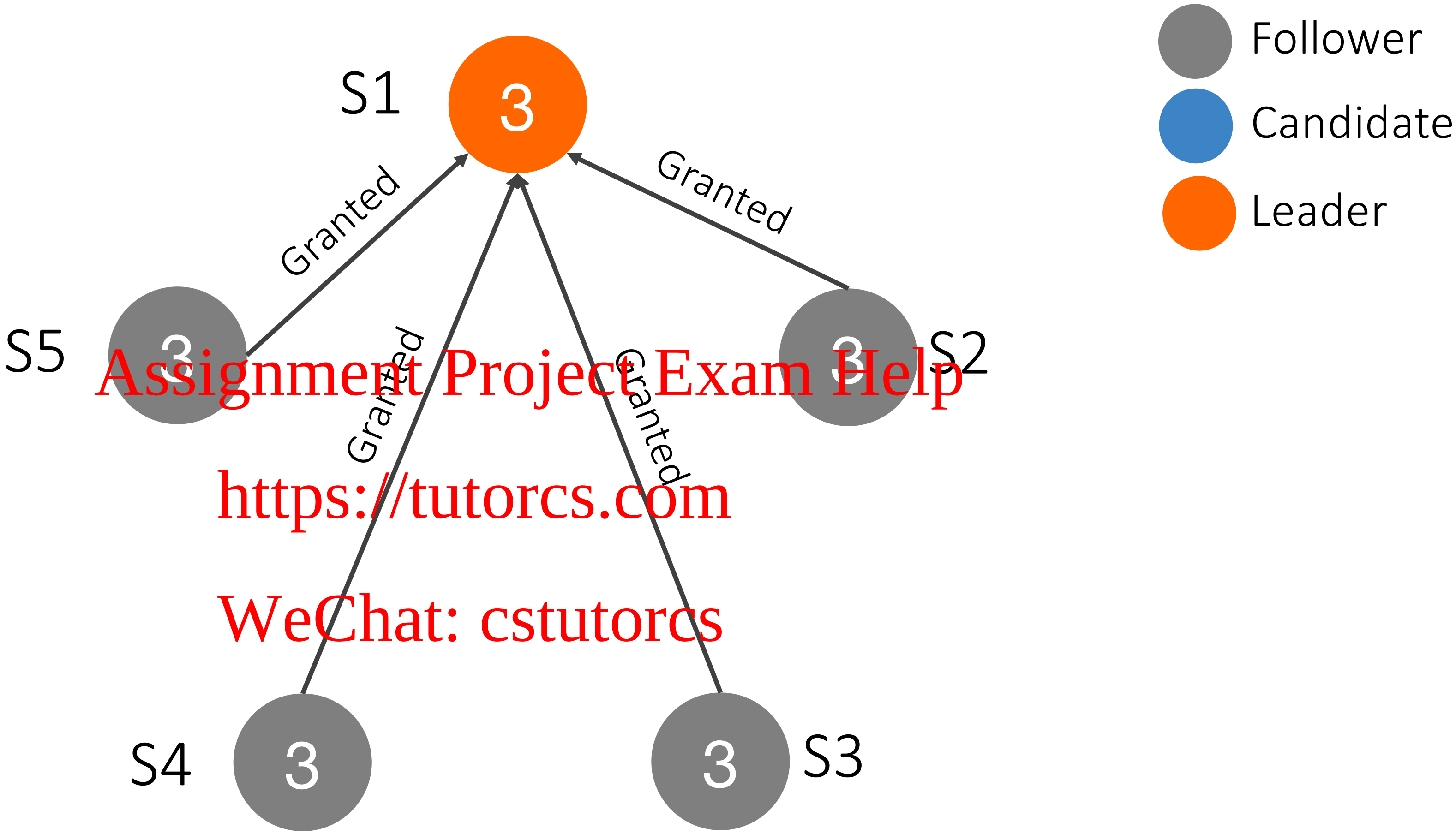
Neither candidate gets majority. After a random delay between 150-300ms try again.



S1 initiates another election for term 3

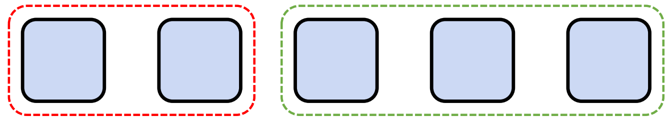


Everyone grants the vote to S1



S1 becomes leader for term 3

Election correctness



- Safety (nothing bad happen): allow at most one winner per term
  - Each server gives only one vote per term
  - Majority required to win election

- Liveness (something good happen): some candidate must eventually win
  - Choose election timeouts randomly in [T, 2T] (e.g., 150-300ms)
  - One sever usually times out and wins election before others time out
  - Works well if T >> broadcast time

So, what does a leader do?

Leader

- Accept client command

- Append them to their log (new entry)

- Issue **AppendEntry** RPCs in parallel to all followers

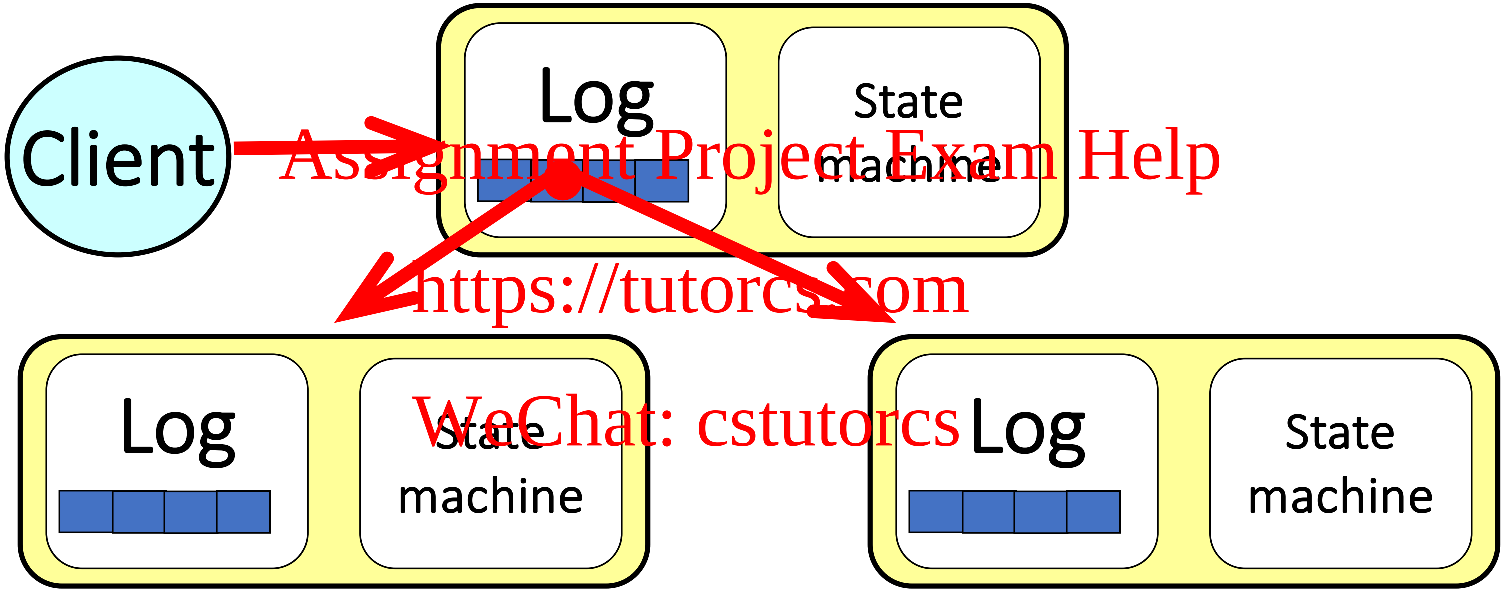
- Apply the entry to their state machine once it has been safely replicated

Leader

- Apply the entry to their state machine once it has been safely replicated. What does this means?

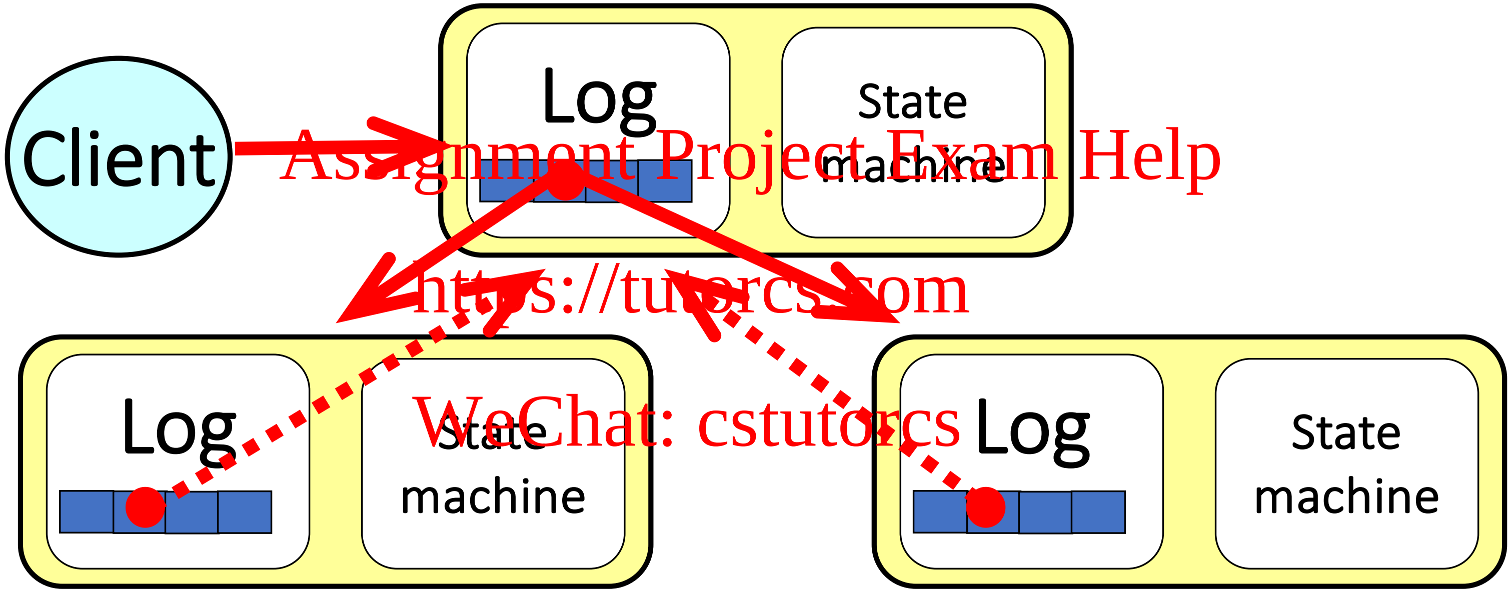
- Once new entry committed (safely replicated)
  - Leader executes command in its state machine, returns result to client
  - Leader notifies followers of committed entries in subsequent AppendEntries RPCs
  - Followers execute committed commands in their state machines

A client sends a request



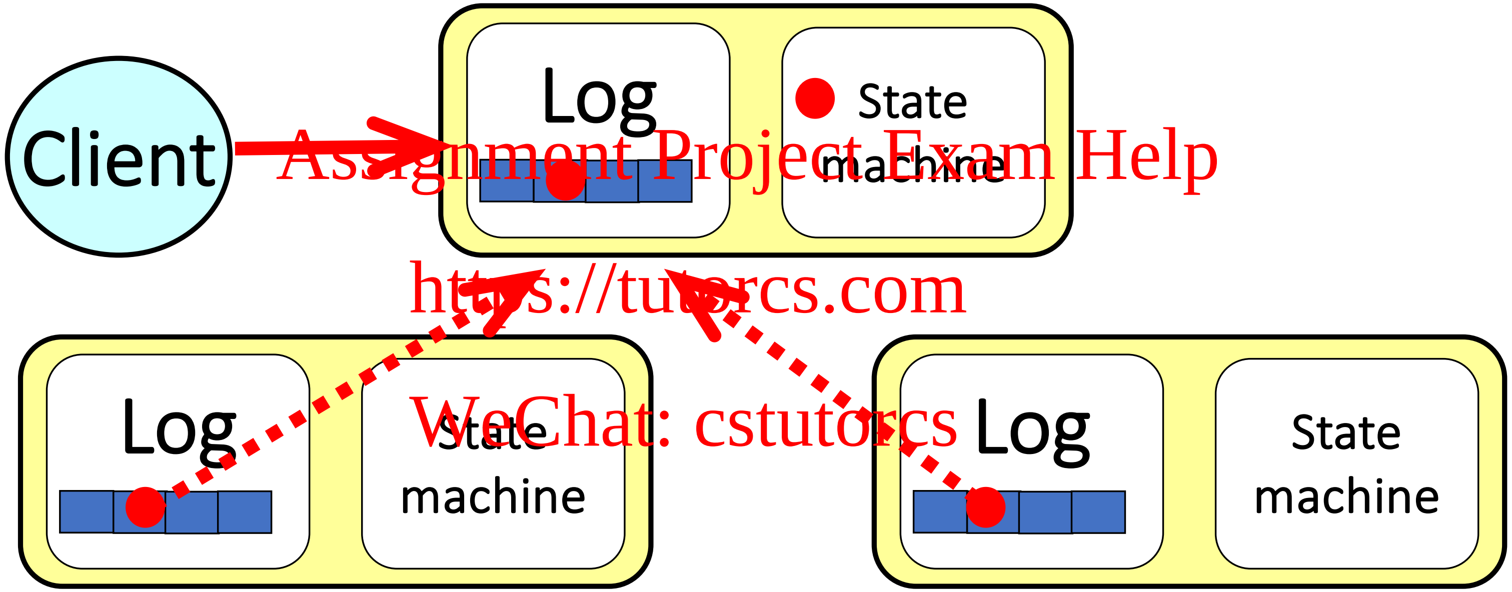
Leader stores request on its log and forwards it to its followers

The followers receive the request



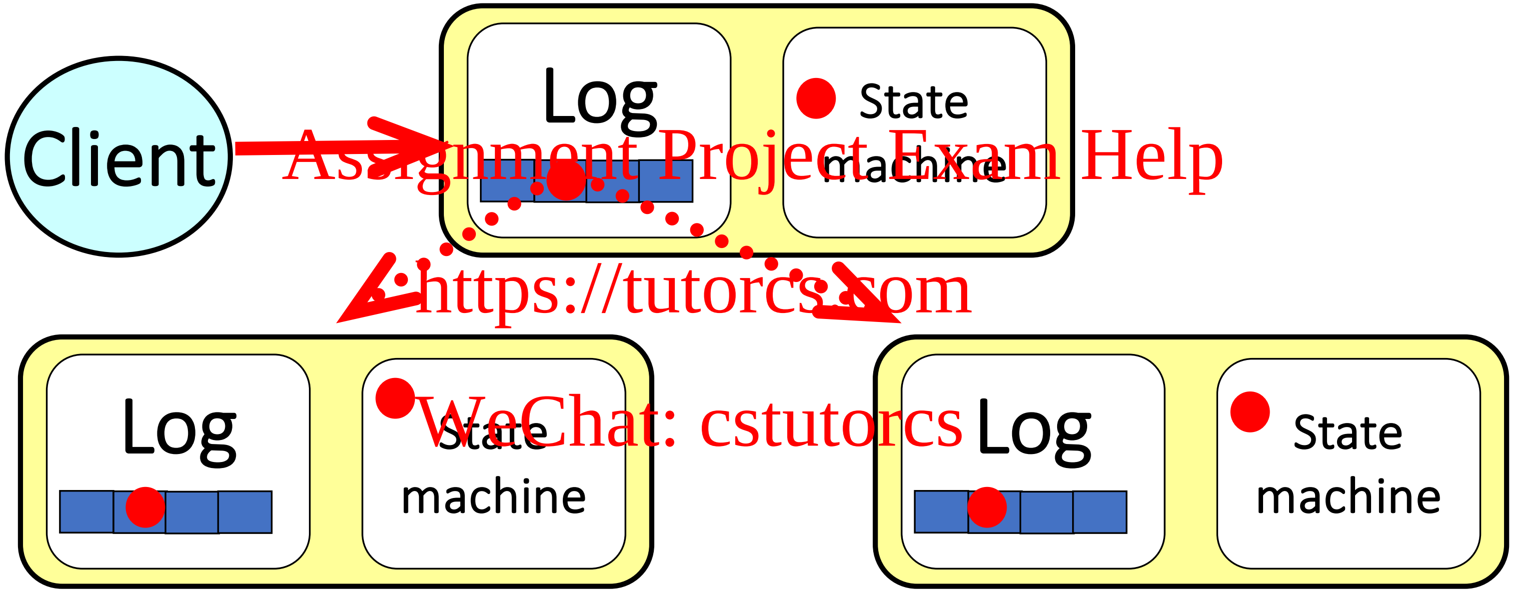
Followers store the request on their logs and acknowledge its receipt

The leader counts followers' ACKs



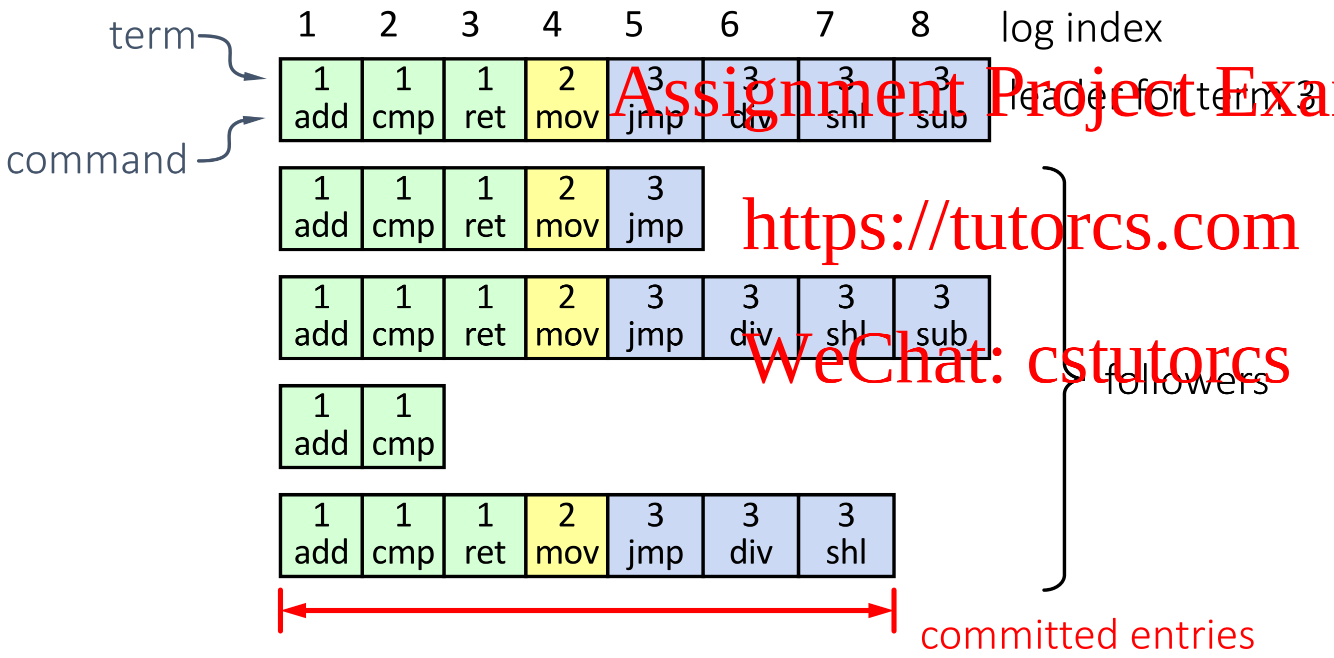
Once it ascertains the request has been processed by a majority of the servers, it consider the entry <u>committed</u> (replicated in enough logs). So, it execute the command in the state machine

The leader counts followers' ACKs



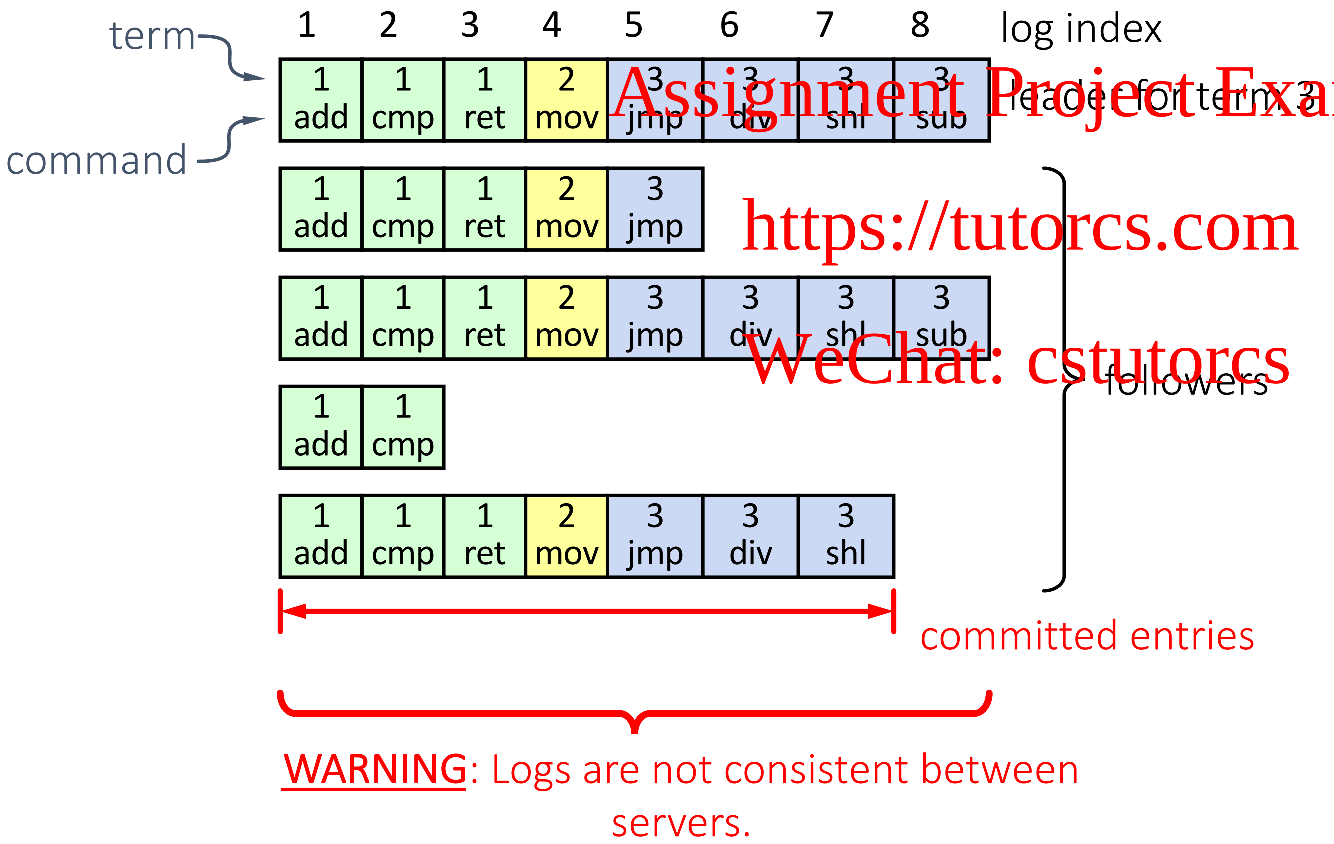
Leader's heartbeats convey the news to its followers: they update their state machines

Log structure



- Entry is <u>**committed**</u> only if it is stored in the majority of the servers (i.e., in this case index = 7)
- This is to guarantee that operations are executed in strictly <u>**the same sequential order**</u>

Log structure



- Entry is **committed** only if it is stored in the majority of the servers (i.e., in this case index = 7)
- This is to guarantee that operations are executed in strictly **the same sequential order**

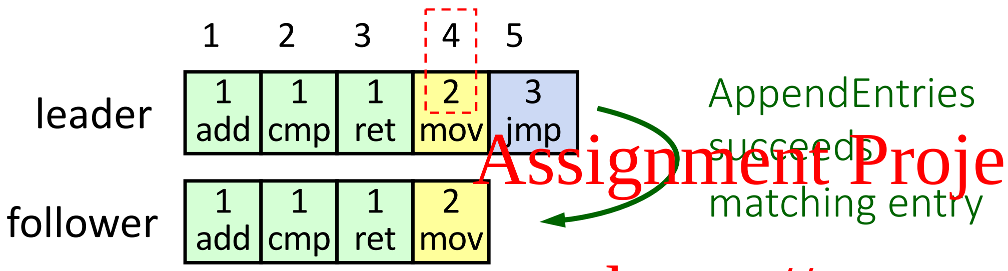
Log matching property

- <u>The goal</u>: high level of consistency between logs

1. If log entries on different servers have the same index and term
   - They store the same command
   - The logs are identical in all the preceding entries (they are committed)

2. If a given entry is committed, all preceding entries are also committed

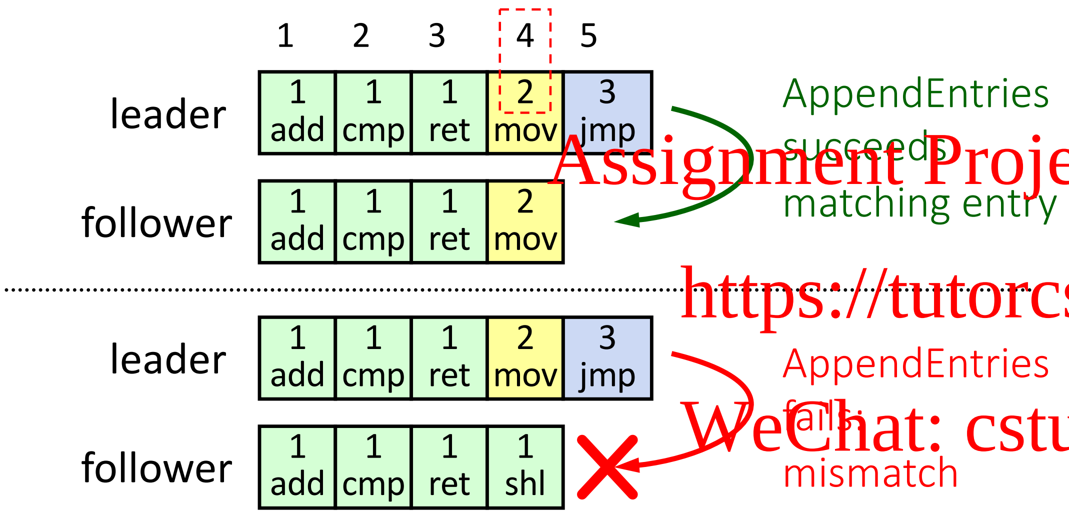
Consistency check



- AppendEntries RPCs include <index, term> of entry preceding new one(s)

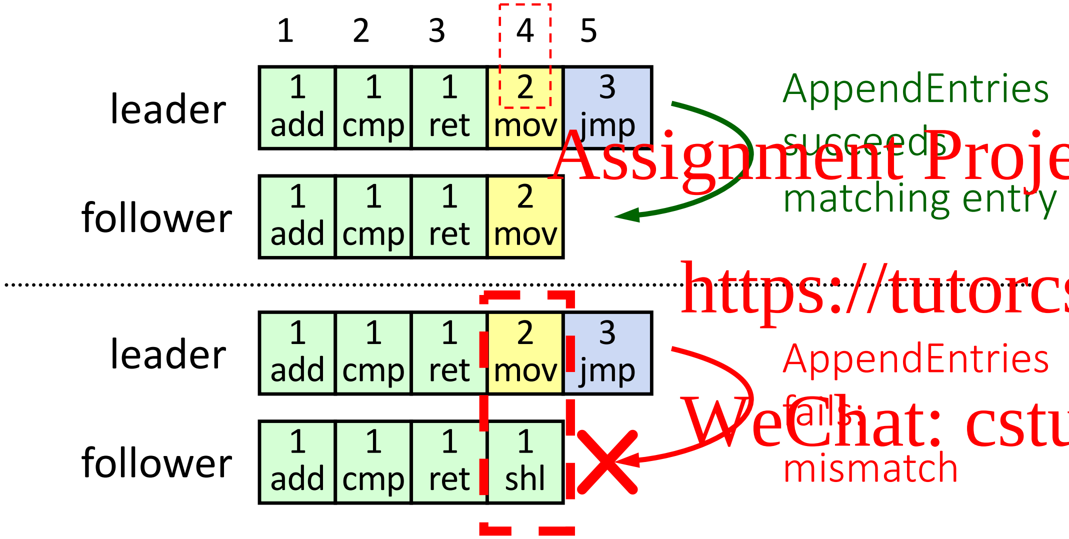
- Follower must contain matching entry;

Consistency check



- AppendEntries RPCs include <index, term> of entry preceding new one(s)
- Follower must contain matching entry; otherwise, it rejects the request
  - Leader retries with lower log index

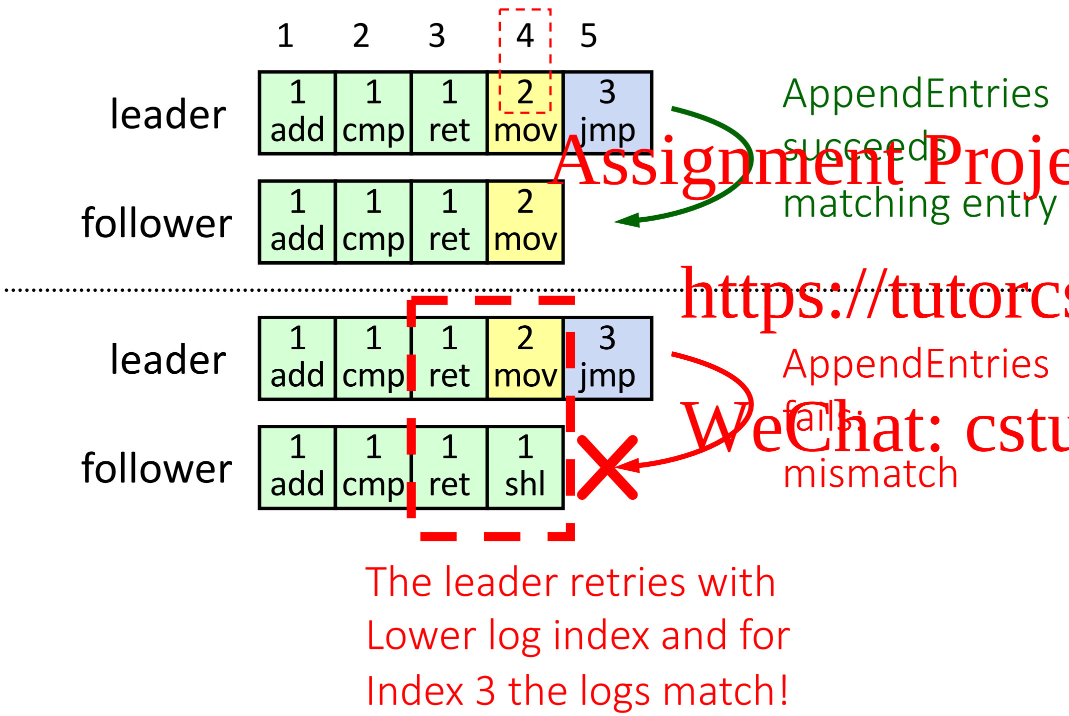
Consistency check



The leader cannot commit <5,3> because index 4 is different

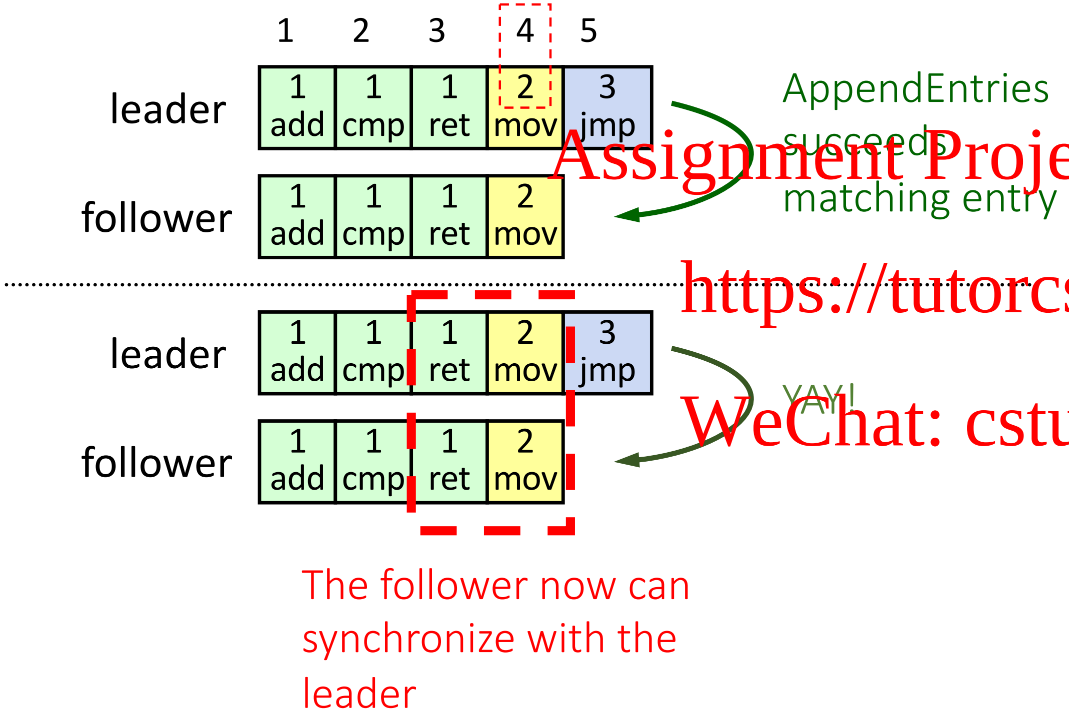
- AppendEntries RPCs include <index, term> of entry preceding new one(s)
- Follower must contain matching entry; otherwise, it rejects the request
  - Leader retries with lower log index

Consistency check



- AppendEntries RPCs include <index, term> of entry preceding new one(s)
- Follower must contain matching entry; otherwise, it rejects the request
  - Leader retries with lower log index

Consistency check



- AppendEntries RPCs include <index, term> of entry preceding new one(s)

- Follower must contain matching entry; otherwise, it rejects the request
  - Leader retries with lower log index

Safety: leader completeness

- This assumes that the leader is always right! (it has all the entry committed)

- Once log entry committed, all future leaders must store that entry

- Servers with incomplete logs must not get elected
  - Candidates include index and term of last log entry in RequestVote
  - Voting servers denies vote if its log is more up-to-date
  - Longs ranked by <lastTerm, lastIndex>

Eventual liveness

- Theoretically, competing candidates could cause repeated split votes

- Raft mitigates this by having each participating server individually choose a new random timeout within each given interval.

- This will lead to a situation, where usually there is only one server awake, which can then win the election while every other server is still asleep.

- This works best if the lower bound of the chosen interval is considerably larger than the broadcast time

Summary

- Consensus key building block in distributed systems

- Raft “similar to” Paxos
- Raft arguably easier to understand than Paxos
  - It separates stages which reduces the algorithm state space
  - Provides a more detailed implementation

Introduction to Cloud Computing

Disclaimer

So, what is it?

- <u>Cloud Computing</u> is a general term used to describe a class of network-based computing that takes place over the Internet

- Simply the renting of servers and/or storage as well as access to these resources via a network

- This an oversimplification but a good starting point

So, what is it? (cont'd)

- These platforms hide the complexity and details of the underlying infrastructure from users and applications by providing a very simple graphical interface or API (Applications Programming Interface)

- The illusion of infinite computing resources available on demand
  - on demand services, that are always on, anywhere, anytime and any place

So, what is it? (cont'd)

- The ability to use of computing resources on a short-term basis as needed (e.g., processors by the hour and storage by the day) and release them as needed

- Pay for use and as needed
  - scale up and down in capacity and functionalities

In summary

- Cloud computing is an umbrella term used to refer to Internet based development and services

- A number of characteristics define cloud data, applications services and infrastructure:
  - <u>Remotely hosted</u>: services or data are hosted on remote infrastructure
  - <u>Ubiquitous</u>: services or data are available from anywhere
  - <u>Commodified</u>: The result is a utility computing model similar to traditional that of traditional utilities, like gas and electricity - you pay for what you would want!

Motivating cloud computing

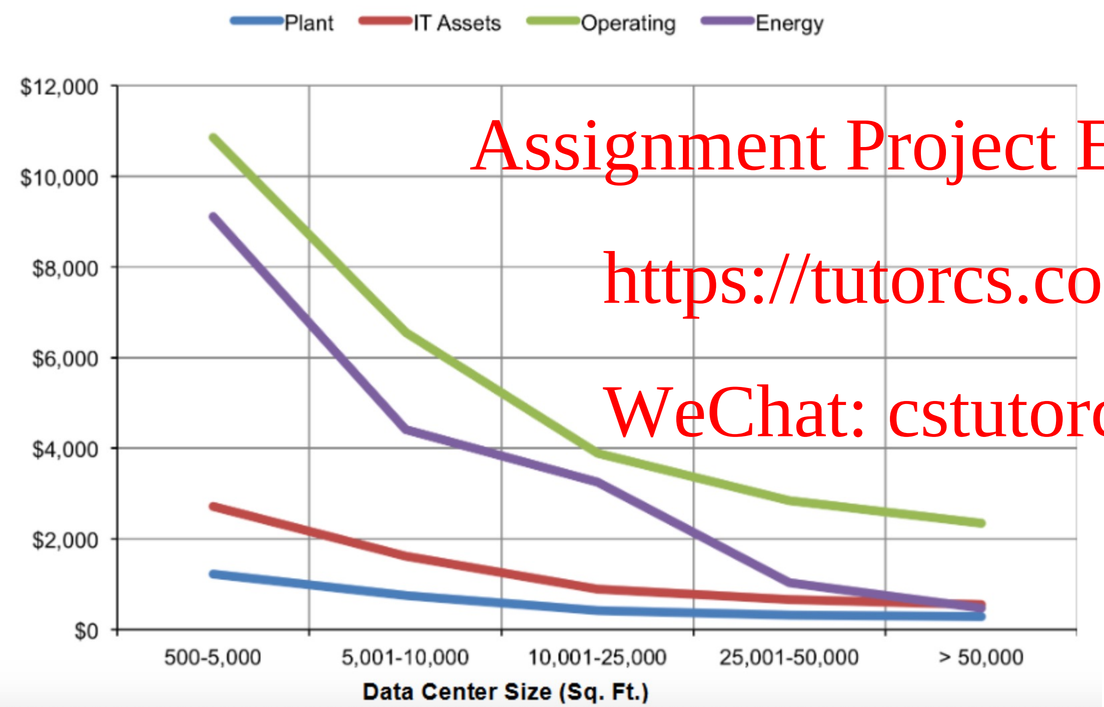
- Very large data centres can purchase hardware, network bandwidth and power for 1/5 to 1/7 the prices offered to a medium-sized data centre

Data Center Size (sq. ft.)	No. of Racks	Avg Compute Load (kW)	Avg Rack Density (kW)	Average Cost Per kW		Average Cost Per Rack	
				Annually	Monthly	Annually	Monthly
500 to 5,000	28.5	105	3.5	$26,495	$2,208	$97,614	$8,134
5,001 to 10,000	58	318	5.7	$13,662	$1,135	$74,689	$6,224
10,001 to 25,000	95	620	6.5	$8,464	$705	$55,242	$4,604
25,001 to 50,000	128.5	972	8	$6,734	$561	$50,841	$4,245
> 50,000	183	1,400	7.8	$5,467	$456	$41,825	$3,485

Data taken from a report made in 2016

https://www.datacenterdynamics.com/news/research-larger-data-centers-make-considerable-savings-on-operating-costs/

Economy of scale..



Data taken from a report made in 2016

https://www.datacenterdynamics.com/news/research-larger-data-centers-make-considerable-savings-on-operating-costs/

This is good for everyone

- Parallel batch processing

  - Batch processing and analytics jobs can analyse terabytes of data and take hours to finish

  - If there is enough parallelism, users can use hundreds of servers to complete the job quickly

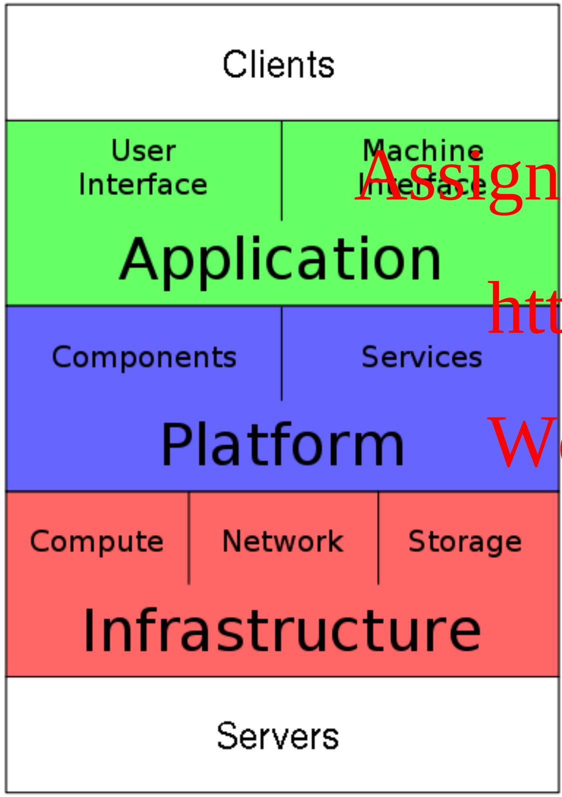
  - Tools such as Hadoop can be used to reduce the complexity of implementing these jobs

This is good for everyone

- <u>The rise of analytics</u>
  - A special case of batch processing is business analytics
  - A growing share of computing resources is now spent on understanding customers, supply chains and buying habits
  - Market Sentiment analysis using Twitter data is a good example of this

convinced?

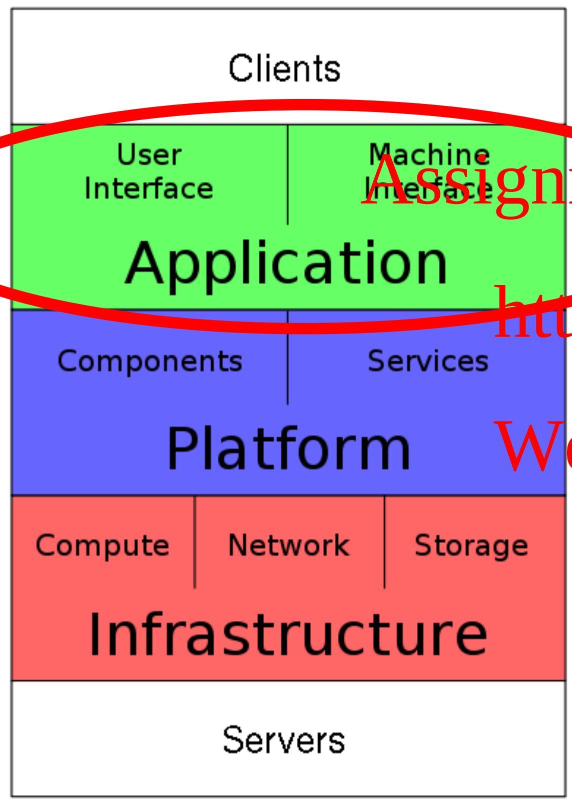
Cloud architecture



Cloud Computing Stack

- <u>Three main categories</u>
  - Software as a Service (SaaS)
  - Platform as a Service (PaaS)
  - Infrastructure as a Service (IaaS)
- Other services categories include
  - Storage as a Service (STaaS)
  - Database as a Service (DbaaS)

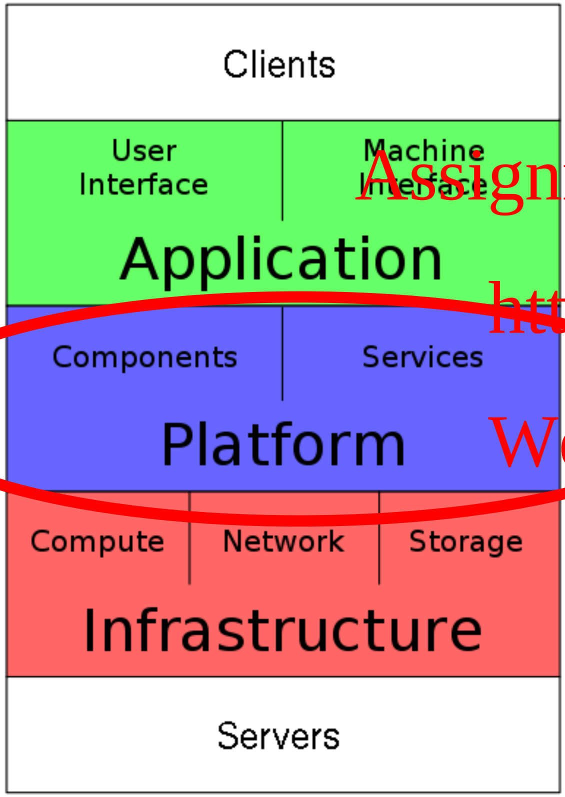
Software as a Service



Cloud Computing Stack

- Software is hosted on a cloud and clients typically access the software via a web browser
  - Examples: Facebook, Netflix, Youtube
- Software can be licensed on a subscription basis or supported via ad revenue and data services
- Migration of traditional software to SaaS model
  - Microsoft Office → Office 365
  - DVD Games → Steam

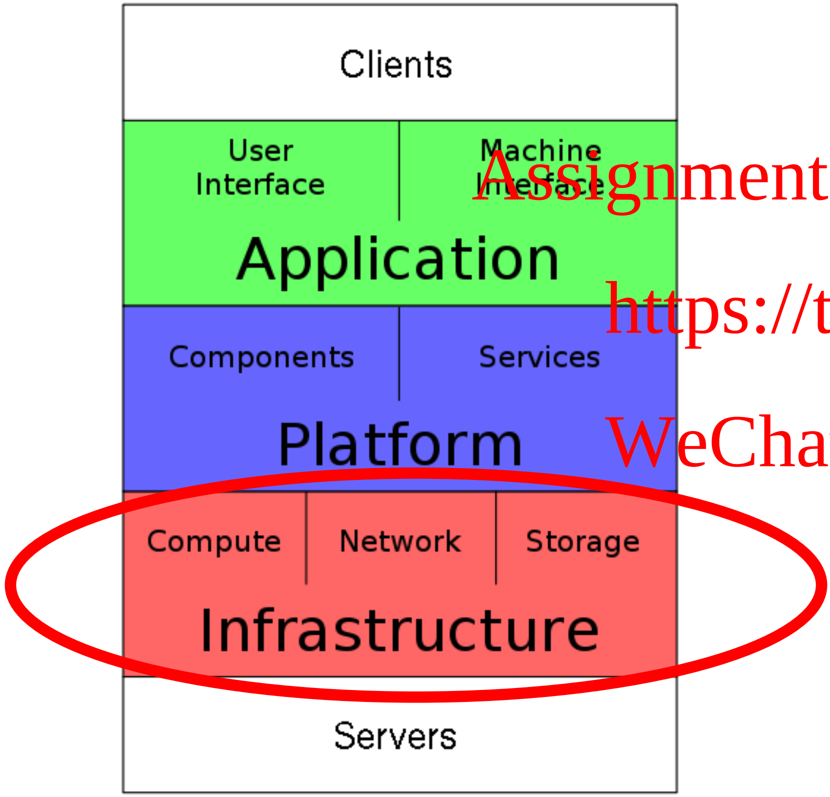
Platform as a Service



Cloud Computing Stack

- Cloud owners provide a platform for users to develop, run and manage web applications
  - Examples: IBM Bluemix, Googles AppEngine, Microsoft Azure
- Developers are restricted to a particular language (Javascript, Python, PHP) or framework (.NET)

Infrastructure as a Service



Cloud Computing Stack

- Cloud owners provide direct access to virtual (or in rare cases physical) machines which users can configure
  - Examples: IBM Bluemix Virtual, Machine and Amazon's EC2
- Users can select from a wide variety of operating systems and hardware configurations

Quick recap

- SaaS: provides access to application software. No need to worry about the installation, setup and running of the application.
  - Examples: Google Apps, Microsoft Office 365

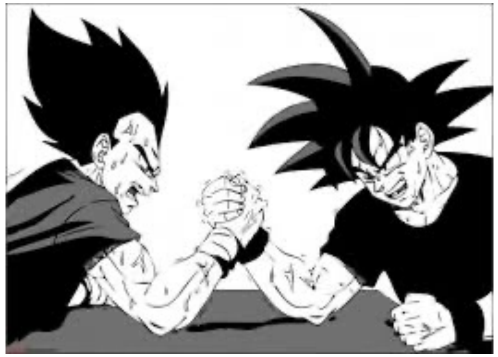
Quick recap

- SaaS: provides access to application software. No need to worry about the installation, setup and running of the application.
  - Examples: Google Apps, Microsoft Office 365
- PaaS: provides computing platforms which typically includes operating system, programming language execution environment, database, web server etc.
  - Examples: AWS Elastic Beanstalk, Windows Azure, Google App Engine

Quick recap

- SaaS: provides access to application software. No need to worry about the installation, setup and running of the application.
  - Examples: Google Apps, Microsoft Office 365
- PaaS: provides computing platforms which typically includes operating system, programming language execution environment, database, web server etc.
  - Examples: AWS Elastic Beanstalk, Windows Azure, Google App Engine
- IaaS: provides the computing infrastructure, physical or virtual machines and other resources like virtual-machine disk image library, block and file-based storage, firewalls, load balancers, IP addresses, virtual local area networks etc.
  - Examples: Amazon EC2, Windows Azure, Google Compute Engine.

IaaS vs PaaS



- IaaS is more powerful as more tools available and customization possible

- From great power comes great responsibility

  - User responsible for scaling applications (some tools like Amazon's Autoscaling can help but configuration required)

  - User responsible for updating OS and machine image (happens automatically on PaaS)

- In general PaaS less complex as many concepts are abstracted from the user

Example 1

- A biology lab creates 400GB of data for every experiment and wants to move its data processing to the cloud

- Choose a service model (IaaS, PaaS, SaaS) for the lab and explain why you chose this model?

Example 1

- IaaS is probably the most appropriate model

- The described data processing might require complex code that may not be easily integrated into a SaaS or even a PaaS service.

Example 2

- A web application wants to move its hosting to the cloud.

- Choose a service model (IaaS, PaaS, SaaS) for the lab and explain why you chose this model?

Example 2

- SaaS is probably the most suitable model

- A web application is usually a relatively simple application and the transference of the application into the SaaS cloud should be quite easy

- This reduces the labour costs associated with managing the hosting architecture

Cloud computing economics

- When deciding to migrate a service to the cloud there are a number of consideration to make:

1. The expected average and peak resource utilisation

2. Operational costs

Cloud Computing brings elasticity

- The pay as you go model brings elasticity

- The key advantage of elasticity is that it reduces the risk of overprovisioning (underutilisation) and under provisioning (saturation)

- Most users deliberately provision for the expected peak and allow resources to remain idle at non peak times

- The more pronounced the variation the greater the waste

Example

- A service has predictable daily demand where the peak require 500 servers at noon but then requires only 100 servers at midnight

- The average utilisation over a whole day is 300 servers and the actual utilization of the whole day is 300 x 24 = 7200 server hours

- If we buy servers we must provision for the peak of 500 servers so we pay 500 x 24 = 12000 server hours

Your turn!

- A service has a peak demand of 600 servers/day and a average utilisation of 200 servers/day. The pay as you go cost is £0.1 per server/hour and the buying cost is £0.05 per server/hour.

- Will we save money using Cloud Computing?

Your turn!

- A service has a peak demand of 600 servers/day and a average utilisation of 200 servers/day. The pay as you go cost is £0.1 per server/hour and the buying cost is £0.05 per server/hour.

- Will we save money using Cloud Computing? Yes
  - Buying utilisation = 600 x 24 = 14400 server hours
  - Buying cost = 14400 x £0.05 = **£720 per day**
  - Cloud utilisation = 200 x 24 = 4800 server hours
  - Cloud cost = 4800 x £0.1 = **£480**

Going forward..

- Of course, buying out a server is a one-shot cost, while cloud computing is a constant cost

HOWEVER

- These examples actually underestimate the benefits of cloud computing

- In addition to diurnal patterns most nontrivial service also experience seasonal or other periodic demands which need to be incorporated into the analysis, e.g., Ebay or Amazon in December

Not only that

- There can be unexpected demand bursts due to external events (news events) which need to be provisioned for when buying hardware

- It is much easier to adjust for these events with cloud computing

Under provisioning vs Overprovisioning

- It is difficult to predict peak utilisation

- **Under provisioning:** rejected user generate zero revenue and may not come back due to poor service

- **Overprovisioning:** you spend more for what you actually need

Transference of risk

- With cloud computing it is possible to remove of the risk of under provisioning for expected peak demand

- This is known as the ***transference of risk***

Factors to consider when migrating

- There are other factors which could potentially alter the economic argument to migrate to the cloud

- These include
  - Resource Utilisation
  - Power, cooling and physical plant costs
  - Operational costs

Factors to consider when migrating

- There are other factors which could potentially alter the economic argument to migrate to the cloud

- These include
    - Resource Utilisation
    - Power, cooling and physical plant costs
    - Operational costs

Resource utilization

- Applications might not make equal use of computation, storage and network bandwidth

- Some are CPU-bound while others are network bound etc.

- It is possible that an application will saturate one resource while underutilising another

- In pay as you go cloud computing an application can be charged separately for each type of resource, thereby, reducing the waste of underutilisation

Factors to consider when migrating

- There are other factors which could potentially alter the economic argument to migrate to the cloud

- These include
  - Resource Utilisation
  - Power, cooling and physical plant costs
  - Operational costs

Power, cooling and physical plant costs

- When buying hardware at scale other costs such as cooling, power and a physical building to house the servers needs to be considered
- Cloud computing frequently offers facilities such as data replication and backups at no additional cost
- These costs would have to be added when building a data centre to ensure durability and performance improvements

Factors to consider when migrating

- There are other factors which could potentially alter the economic argument to migrate to the cloud

- These include
  - Resource Utilisation
  - Power, cooling and physical plant costs
  - Operational costs

Operational costs

- Operate a datacentre is complex
- Troubleshooting: Hardware failures? Software failures?
- Cloud Computing: making operation someone else problem!

Example 1

- A biology lab creates 500GB of new data for every lab experiment. A computer of one EC2 instance takes 2 hours per GB to process the data

- The lab has the equivalent of 20 instances locally

- The time it takes to (locally) evaluate the experiment is therefore 500 x 2/20 = 50 hrs

Example 1

- They could process it in a single hour on 1000 instances at AWS. The cost to process one experiment would be 1000 x £0.10 in computation and another 500 x £0.10 in network transfer fees

- The network transfer rate from the lab to AWS is 20Mbit/second

- The transfer time is therefore (500GB x 1000MB/GB x 8bits/Byte)/20Mbits/sec = 4,000,000/20 = 200,000 seconds = approx 55hrs

- It takes 50 hours locally and 55+1=56 hours on AWS so they do not move to the cloud

Example 1: outcome

- No cloud for the lab! (at least in this scenario)

Example 2

- A biology lab generates 1TB of data per experiment

- One EC2 instance takes 2 hours per GB to process the data

- The are the equivalent of 25 instances locally
- The network transfer rate is 50Mbit/s
- Is it quicker in the cloud or locally?

Example 2

- Local Computation time is 1000x2/25 = 80 hrs

- Transfer time is (1000GB x 1000MB/GB x 8bits/Byte)/50Mbits/sec = 160,000 second or 45hrs

- Total Cloud time is 45+1= 46hrs

- May be advisable to move to cloud (Need to consider cost, age of local hardware and other factors)

So far so good

- It seems like cloud computing is (in most of the case) the way to go!

- Any obstacles?

Obstacles

- Privacy!

- The new cash is data!

- This is why there are different deployment models


THE END
JUST AHEAD