Date:
M T W T F S S

Leader Elec^{n}

Gonna understand entire concept thru one of its actual use cases

- Case → Managing subscrip^{n}s and payments in some sort of service such as Amazon Prime, Netflix (subscrip^{n} on recurring basis)

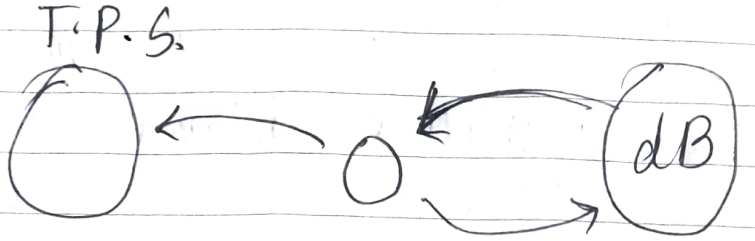
(1) dB → stores data pertainin to subscrip^{n} and users
- → is user curr. ly subscribed?
- → date for subscrip^{n} renewal
- → price

(2) 3rd party service (T.P.S.) → Service that conducts the payment transac^{n} (debit frm user → credit to you) eg. Paypal, Stripe
Needs to comm. with dB to know
- → when a user shud be charged again
- → what the charge?

No direct connec^{n} b/w dB and 3rd party service
- → diff. to implement
- → dB is pretty sensitive part of sys. Contains imp info. and its not wise to divulge all of it by allowin a T.P.S. to connect directly

Date:
M T W T F S S

(3) Service in middle → talks to dB periodically
→ figures out when a user's subscrip" is gonna renew
→ how much to charge the user
→ transmits only relevant info. to T.P.S.



Optimizing this setup.

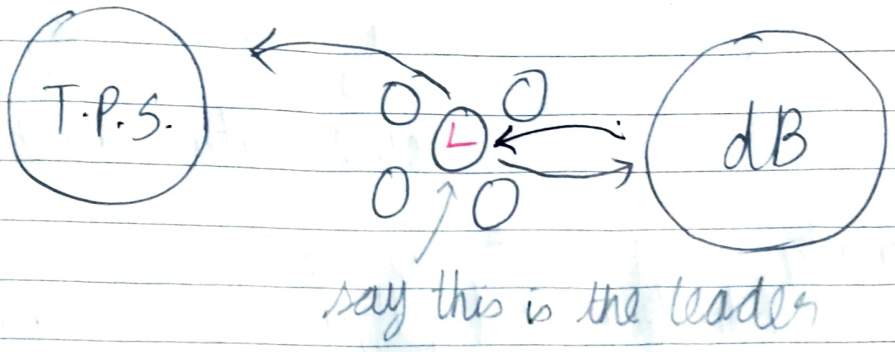
1) If the server in the middle fails, entire payment sys. collapses
We introduce 'redundancy' (passive) in our sys. by hori.ly scaling the service in the middle i.e. have 5 instead 1 server in the middle to do the transac" logic

2) All 5 servers are doin the same thing i.e. askin dB for transac" info and req. in T.P.S. to charge the user
We don't wanna duplicate the req. to T.P.S. to charge the user i.e. make that req. just once

3) Leader elec" → If you hv a grp of machine (in our case - servers) that are in charge of doin the same, instead of havin them all do that thing, we elect 1 of the machines as leader

Network parti^{n} → a some network failure that makes some machines no longer be able to comm. with other machines

and only that machine is gonna be responsib. for doin the opns. that all of the machines are built to do.



Other servers (followers) → just on standby to take over when leader fails i.e. a new leader is elected frm amongs these servers and it takes over.

4) Complexity of leader elec^{n}: Leader elec^{n}'s logic is complex cuz:

- → multip machines are made to elect a single leader (consensus - agree upon somethun together)
- → They all gotta be aware of who the leader is at any given time
- → They all gotta be capable of re-electin a new leader in case curr. leader

Difficulty of logic mainly lies in

- → multip machines that are distru.ed are sharin state. Diff. cuz ye dunno what might happen in a network (eg:- network parti^{n}
- → make multip machines gain consensus

Here, gainin consensus → agreein who's the curr. leader

5) Consensus Algo. → Complex, math heavy algos that allow multip nodes (servers) in a cluster (grp) to reach consensus i.e. agree upon some data-value.

eg:- Paxos, Raft

6) Usually, ppl just use some pre-existin 3rd party industry tool that offers them this logic whilst itself runnin a consensus algo under the hood

eg:- Zookeeper, Etcd → help implem. leader elec^n v. easily.

7) Leader elec^n usin Etcd:

(1) Etcd → k-v store
→ highly available
→ strongly consistent

Consistency → if you got multip machine (s), readin & writin to the same k-v pair in the k-v store, you're always gonna get the same, correct value irresp of when or from which machine the k-v pair is accessed.

(2) Etcd achieves high availability, strong consistency by implementin consensus algo - Raft

(3) There are gonna be multip machines that can read, write to the main k-v store

that Etcd supports. These machines need:

- high availability: To know which other machines are available after a leader dies
- strong consistency: single source of truth for all k-v pairs in store

(4) In Etcd each k-v pair can be visualised be of form:

key	value
status of machine (i.e. is it a leader or a follower)	name / IP address (basically a UID for a machine in grp i.e. node in cluster)

∴ we'll hv 1 k-v pair as follows

"leader": UID for this machine

↳ k-v pair represents curr "leader"

(5) All machines (in our case - servers) comm. with k-v store any given pt. in time we gotta hv a leader present as k-v pair in the store else we gotta elect a new one.

```
import etcd3
import time
from threading import Event

# The current leader is going to be the value with this key
LEADER_KEY = "/algoexpert/leader"

# Entrypoint of the program

def main(server_name):
    # Create a new client to etcd
    client = etcd3.client(host="localhost", port=2379)

    while True:
        is_leader, lease = leader_election(client, server_name)

        if is_leader:
            print("I am the leader.")
            on_leadership_gained(lease)
        else:
            print("I am a follower.")
            wait_for_next_election(client)

# This election mechanism consists of all clients trying to put their name into a single key, but in a way that only works if the key does not exist(or has expired before)

def leader_election(client, server_name):
    print("New leader election happening.")
 # Create a lease before creating a key. This way, if this client ever lets the lease expire, the keys associated with that lease will all expire as well.
 Here, if the client fails to renew lease for 5 seconds (network partition or machine goes down), then the leader election key will expire.
    lease = client.lease(5)

    # Try to create the key with your name as the value. If it fails, then another serer got there first
    is_leader = try_insert(client, LEADER_KEY, serve_name, lease)
    return is_leader, lease


def on_leadership_gained(lease):
    while True:
# As long as this process is alive and we're the leader, we try to renew the lease. We don't give up the leadership unless the process/machine crashes or some exception is raised
        try:
            print("Refershing lease; still the leader.")
            lease.refresh()
# This is where the business logic would go (eg: ask 3rd part service to charge user based on Db INFO)
            do_work()
        except Exception:
# Here we most likely got a client timeout (from network issue). Try to revoke the current lease so another member can get leadership
            lease.revoke()
        except KeyboardInterrupt:
```

```
        print("\n Revoking lease; no longer the leader")
        # Here we're killing the proccess. Revoke the lease and exit
        lease.revoke()
        sys.exit(1)


def wait_for_next_election(client):
    election_event = Event()

    def watch_callback(resp):
        for event in resp.events:
# For each event in the watch event, if the event is a deletion it means that the key expired / got deleted,
which means the leadership is up for grabs.
            if isinstance(event, etcd3.events.DeleteEvent):
                print("LEADERSHIP CHANGE REQUIRED")
                election_event.set()


watch_id = client.add_watch_callback(LEADER_KEY, watch_callback)

# While we haven't seen the that the leadership needs change, just sleep
try:
    while not election_event.is_set():
        time.sleep(1)
except KeyboardInterrupt:
    client.cancel_watch(watch_id)
    sys.exit(1)

# Cancel the watch; we see that the election should happen again.
client.cancel_watch(watch_id)

# Try to insert a key into etcd with a value and a lease. If the lease expires that key will get automatically
deleted behind mthe scenes. If that key was already present this will raise an exception.

def try_insert(client, key, value, lease):
    insert_succeeded, _ = client.transaction(
        failure=[],
        success=[client.transaction.put(key, value, lease)],
        compare=[client.transaction.version(key) == 0],
    )
    return insert_succeeded


def do_work():
    time.sleep(1)


if __name__ == "main":
    server.name = sys.argv[1]
    main(server_name)
```

leader_election — Python leader_election.py server1 — 93×24

~/Documents/Content/Design_Fundamentals/Examples/leader_election — Python leader_election.py server1

```
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
^C
Revoking lease; no longer the leader.
Clements-MBP:leader_election clementmihailescu$ python3 leader_election.py server1
New leader election happening.
I am a follower.
LEADERSHIP CHANGE REQUIRED
New leader election happening.
I am a follower.
```

leader_election — Python leader_election.py server2 — 93×24

~/Documents/Content/Design_Fundamentals/Examples/leader_election — Python leader_election.py server2

```
Clements-MBP:leader_election clementmihailescu$ python3 leader_election.py server2
New leader election happening.
I am a follower.
^CClements-MBP:leader_election clementmihailescu$ python3 leader_election.py server2
New leader election happening.
I am a follower.
LEADERSHIP CHANGE REQUIRED
New leader election happening.
I am a follower.
LEADERSHIP CHANGE REQUIRED
New leader election happening.
I am a follower.
```

leader_election — Python leader_election.py server3 — 93×24

~/Documents/Content/Design_Fundamentals/Examples/leader_election — Python leader_election.py server3

```
Clements-MBP:leader_election clementmihailescu$ python3 leader_election.py server3
New leader election happening.
I am a follower.
LEADERSHIP CHANGE REQUIRED
New leader election happening.
I am a follower.
LEADERSHIP CHANGE REQUIRED
New leader election happening.
I am the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
```

leader_election — -bash — 93×24

~/Documents/Content/Design_Fundamentals/Examples/leader_election — -bash

```
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
^C
Revoking lease; no longer the leader.
Clements-MBP:leader_election clementmihailescu$
```

leader_election — Python leader_election.py server1 — 93×24

~/Documents/Content/Design_Fundamentals/Examples/leader_election — Python leader_election.py server1

```
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
^C
Revoking lease; no longer the leader.
Clements-MBP:leader_election clementmihailescu$ python3 leader_election.py server1
New leader election happening.
I am a follower.
```

leader_election — Python leader_election.py server2 — 93×24

~/Documents/Content/Design_Fundamentals/Examples/leader_election — Python leader_election.py server2

```
Clements-MBP:leader_election clementmihailescu$ python3 leader_election.py server2
New leader election happening.
I am a follower.
^CClements-MBP:leader_election clementmihailescu$ python3 leader_election.py server2
New leader election happening.
I am a follower.
LEADERSHIP CHANGE REQUIRED
New leader election happening.
I am a follower.
```

leader_election — Python leader_election.py server3 — 93×24

~/Documents/Content/Design_Fundamentals/Examples/leader_election — Python leader_election.py server3

```
Clements-MBP:leader_election clementmihailescu$ python3 leader_election.py server3
New leader election happening.
I am a follower.
LEADERSHIP CHANGE REQUIRED
New leader election happening.
I am a follower.
```

leader_election — Python leader_election.py server4 — 93×24

~/Documents/Content/Design_Fundamentals/Examples/leader_election — Python leader_election.py server4

```
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
```

leader_election — -bash — 93×24

~/Documents/Content/Design_Fundamentals/Examples/leader_election — -bash

```
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
^C
Revoking lease; no longer the leader.
Clements-MBP:leader_election clementmihailescu$
```

leader_election — Python leader_election.py server2 — 93×24

~/Documents/Content/Design_Fundamentals/Examples/leader_election — Python leader_election.py server2

```
Clements-MBP:leader_election clementmihailescu$ python3 leader_election.py server2
New leader election happening.
I am a follower.
^CClements-MBP:leader_election clementmihailescu$ python3 leader_election.py server2
New leader election happening.
I am a follower.
LEADERSHIP CHANGE REQUIRED
New leader election happening.
I am a follower.
```

~/Documents/Content/Design_Fundamentals/Examples/leader_election — Python leader_election.py server3

```
Clements-MBP:leader_election clementmihailescu$ python3 leader_election.py server3
New leader election happening.
I am a follower.
LEADERSHIP CHANGE REQUIRED
New leader election happening.
I am a follower.
```

~/Documents/Content/Design_Fundamentals/Examples/leader_election — Python leader_election.py server4

```
Clements-MBP:leader_election clementmihailescu$ python3 leader_election.py server4
New leader election happening.
I am a follower.
LEADERSHIP CHANGE REQUIRED
New leader election happening.
I am the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
Refreshing lease; still the leader.
```