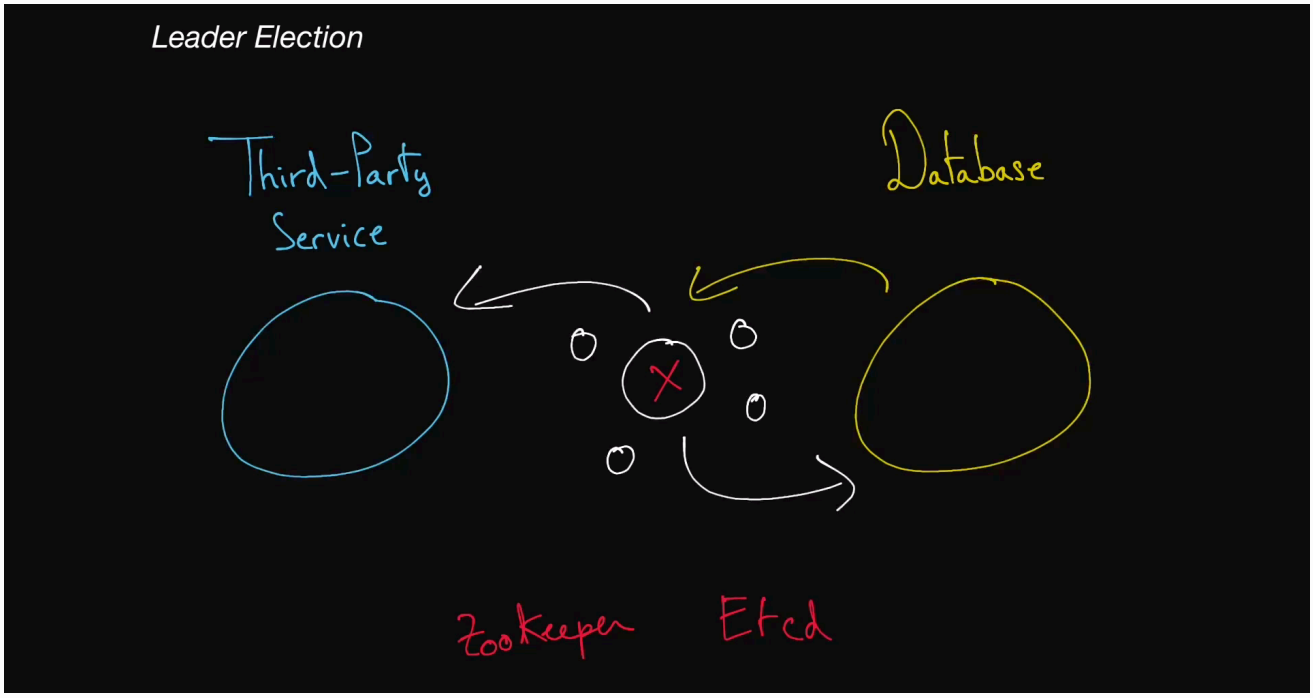


leader_election.py — leader_election

leader_election.py

```python
import etcd3

import sys
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.
    # https://help.compose.com/docs/etcd-using-etcd3-features#section-leases
    lease = client.lease(5)

    # Try to create the key with your name as the value. If it fails, then
```

leader_election — -bash — 93×24

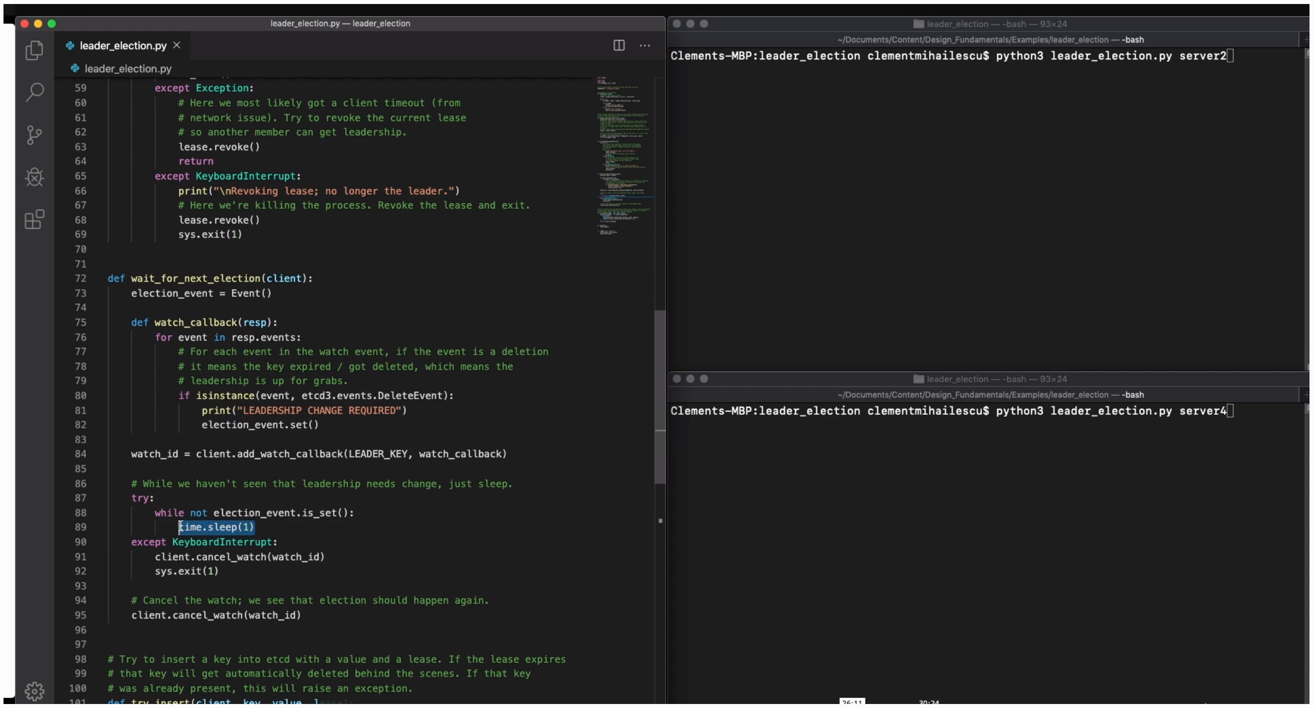
~/Documents/Content/Design_Fundamentals/Examples/leader_election — -bash

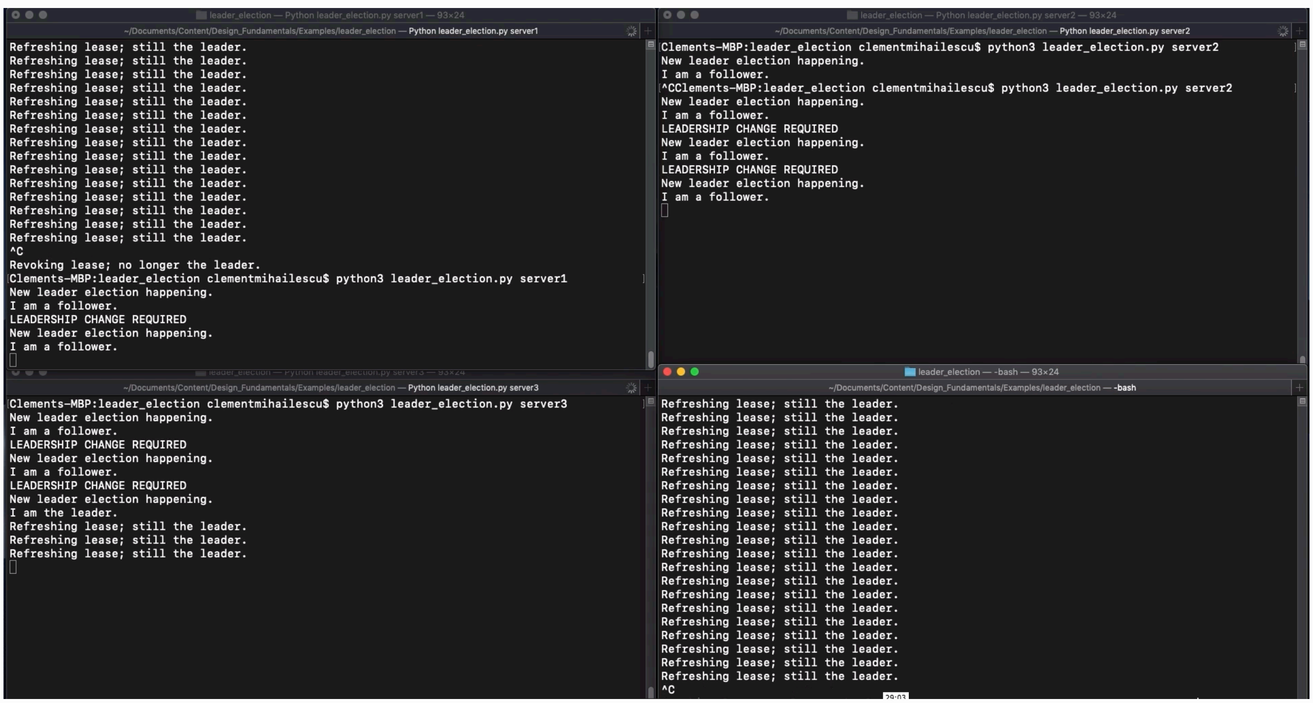
```
Clements-MBP:leader_election clementmihailescu$ python3 leader_election.py server2
```

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

```
Clements-MBP:leader_election clementmihailescu$ python3 leader_election.py server4
```

This code example was based off of the code found at:
https://www.sandtable.com/etcd3-leader-election-using-python/


leader_election.py — leader_election
leader_election.py
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()
return
except KeyboardInterrupt:
print("\nRevoking lease; no longer the leader.")
# Here we're killing the process. 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 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 that 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 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 the scenes. If that key
# was already present, this will raise an exception.
Clements-MBP:leader_election clementmihailescu$ python3 leader_election.py server2
Clements-MBP:leader_election clementmihailescu$ python3 leader_election.py server4


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.
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.
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.
^C

5 Prerequisites

Availability

The odds of a particular server or service being up and running at any point in time, usually measured in percentages. A server that has 99% availability will be operational 99% of the time (this would be described as having two **nines** of availability).

High Availability

Used to describe systems that have particularly high levels of availability, typically 5 nines or more; sometimes abbreviated "HA".

Redundancy

The process of replicating parts of a system in an effort to make it more reliable.

Strong Consistency

Strong Consistency usually refers to the consistency of ACID transactions, as opposed to **Eventual Consistency**.

Eventual Consistency

A consistency model which is unlike **Strong Consistency**. In this model, reads might return a view of the system that is stale. An eventually consistent datastore will give guarantees that the state of the database will eventually reflect writes within a time period (could be 10 seconds, or minutes).

5 Key Terms

Leader Election

The process by which nodes in a cluster (for instance, servers in a set of servers) elect a so-called "leader" amongst them, responsible for the primary operations of the service that these nodes support. When correctly implemented, leader election guarantees that all nodes in the cluster know which one is the leader at any given time and can elect a new leader if the leader dies for whatever reason.

Consensus Algorithm

A type of complex algorithms used to have multiple entities agree on a single data value, like who the "leader" is amongst a group of machines. Two popular consensus algorithms are **Paxos** and **Raft**.

Paxos & Raft

Two consensus algorithms that, when implemented correctly, allow for the synchronization of certain operations, even in a distributed setting.

Etcd

Etcd is a strongly consistent and highly available key-value store that's often used to implement leader election in a system.

ZooKeeper

ZooKeeper is a strongly consistent, highly available key-value store. It's often used to store important configuration or to perform leader election.