System Architecture Part 1

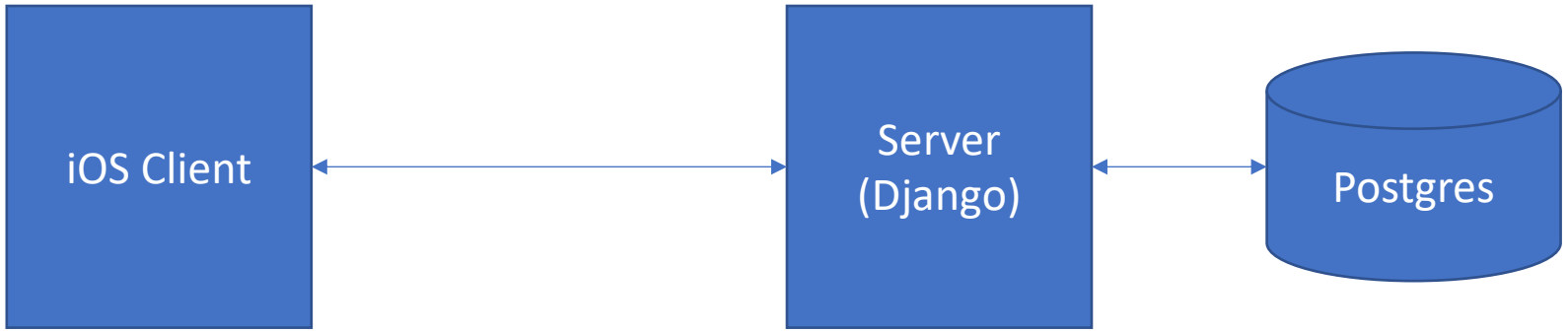
Lets scale that system!!

What We will be learning ?

1. Building a Client-Server Application
2. SwiftUI framework and MVVM design
3. Django Server and MVT design
4. Postgres(SQL dB)
5. Demo
6. Scalability and Benchmarking

1. Client-Server App

- Client Side: iOS Client that uses SwiftUI framework and MVVM
- Server Side: Django server using MVT in Python + Postgres Db



2. MVVM Design Paradigm

- Works in concert with the concept of “reactive” user-interfaces.
- Must be adhered to for SwiftUI to work
- MVVM != MVC(Model View Controller used by UIKit)

MVVM

ObservableObject
@Published
objectWillChange.send()
.environmentObject()

might "interpret"

ViewModel

publishes "something changed"

notices changes

automatically observes publications, pulls data and rebuilds

Binds View to Model
Interpreter

Model

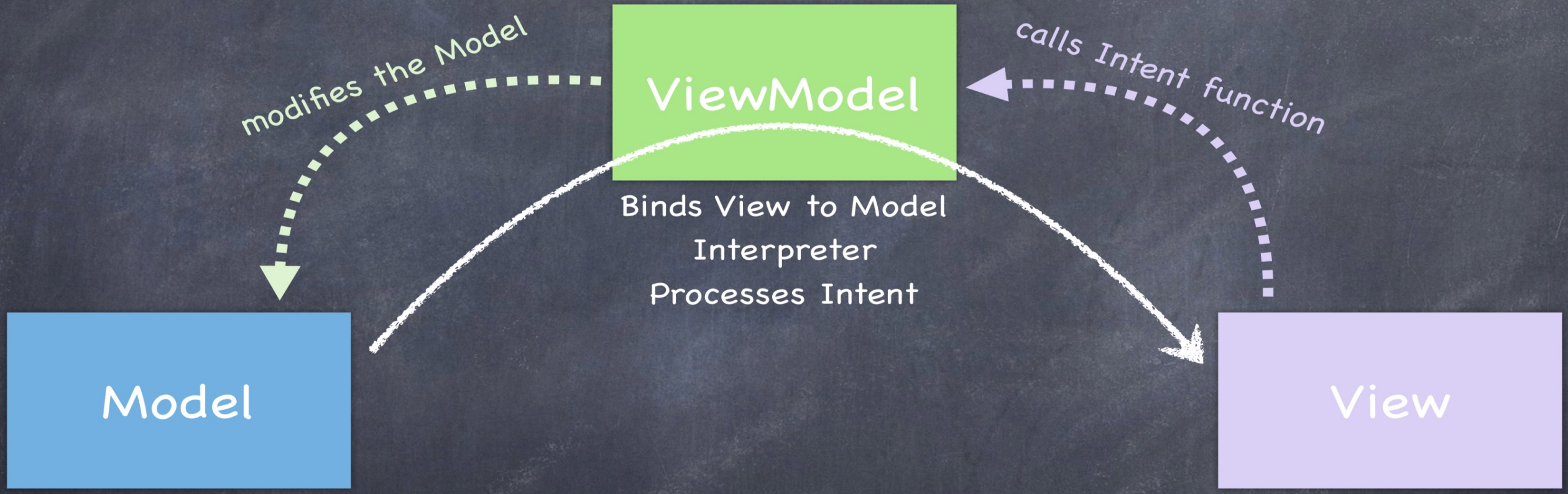
UI Independent
Data + Logic
"The Truth"

@ObservedObject
@Binding
.onReceive
@EnvironmentObject

View

Reflects the Model
Stateless
Declared
Reactive

MVVM



UI Independent
Data + Logic
"The Truth"

Reflects the Model
Stateless
Declared
Reactive

What about the other direction?

MVVM

ObservableObject
@Published
objectWillChange.send()
.environmentObject()

might "interpret"

publishes "something changed"

modifies the Model

calls Intent function

ViewModel

notices changes

automatically observes publications, pulls data and rebuilds

Binds View to Model
Interpreter
Processes Intent

Model

UI Independent
Data + Logic
"The Truth"

@ObservedObject
@Binding
.onReceive
@EnvironmentObject

View

Reflects the Model
Stateless
Declared
Reactive

3. Model View Template

urls.py: Determines what urls(routes) a client can go to

Views.py: Decides what should happen when the client makes a url request

Models.py: Define the types of data we store inside the database

Template: Template engine to manage Html files for a web App

4. PostgreSQL

- Object Relational Database system
- Settings.py: Choose the dB of your choice

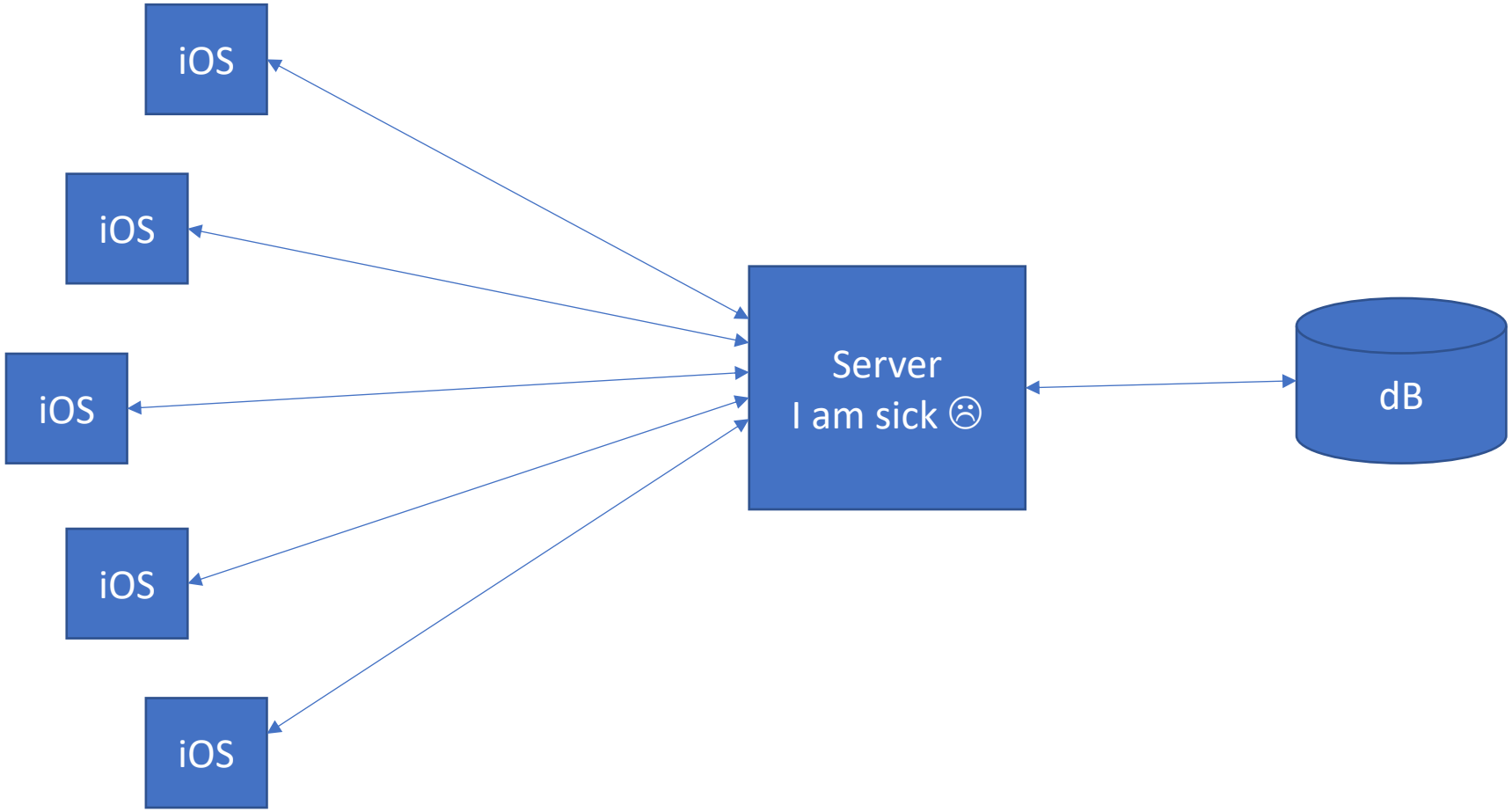
```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'postgres',
        'USER': 'amrbhardwaj',
        'HOST': '127.0.0.1',
        'PORT': '5432',
    }
}
```

Let’s see all this in action !! ☺

6. Scalability

- What happens after we have written our application ?
- What happens if the application gets viral and billions of people start to access the application data at the same time globally?
- What sort of considerations we need to take into account ?

6. Scalability



6. Benchmarking

- The server processes the client requests and delivers the response back to the user
- Server can do only a finite number of things
- How do we measure the number of requests/second a server can handle ?
- How do we measure latency response time(ms) for each new connection or request ?
- How do we measure throughput in bytes/second ?

6. Benchmarking Tools/Reports

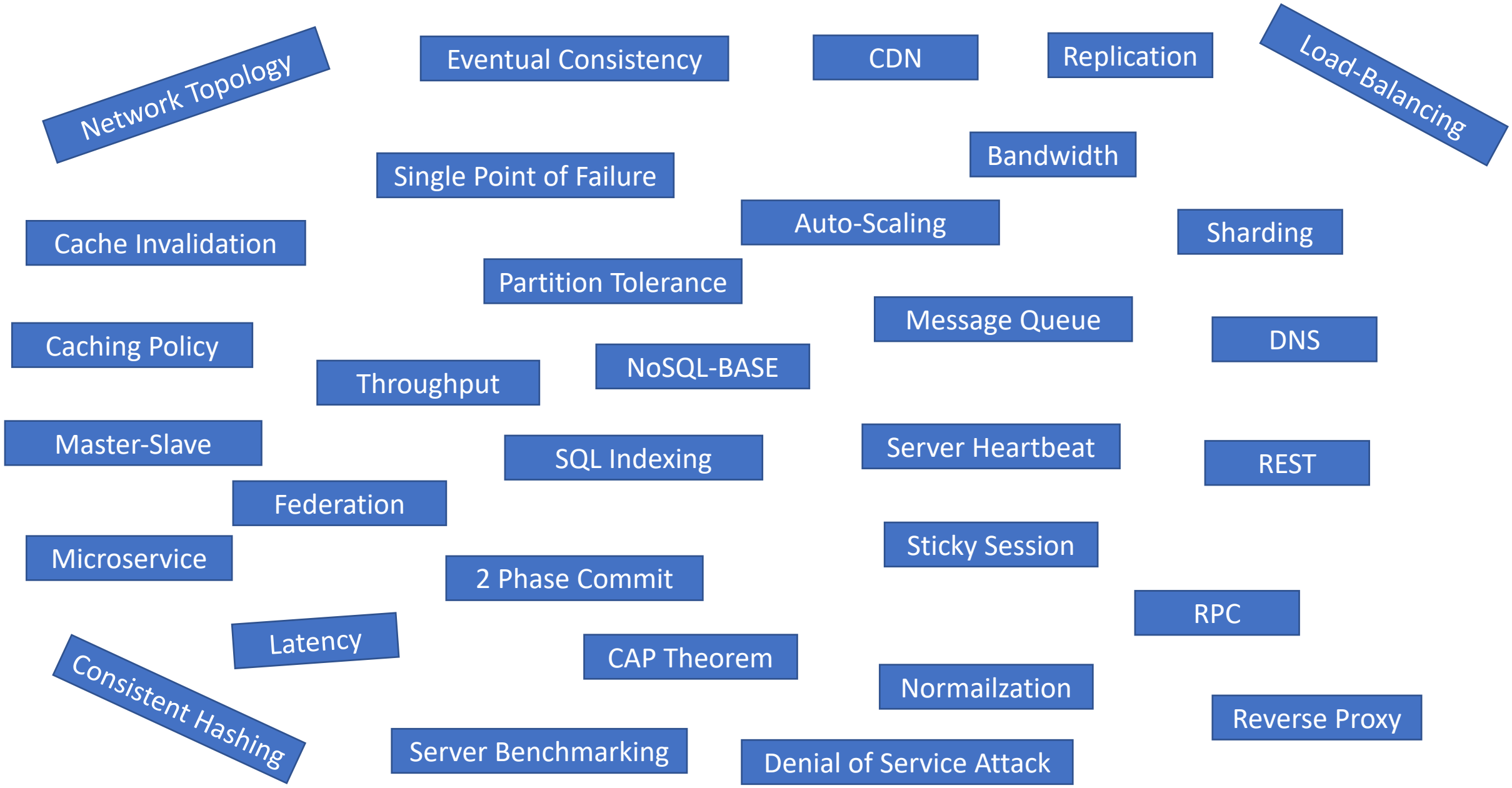
1. Apache Bench
2. Apache Jmeter
3. Django-performance-testing framework, etc

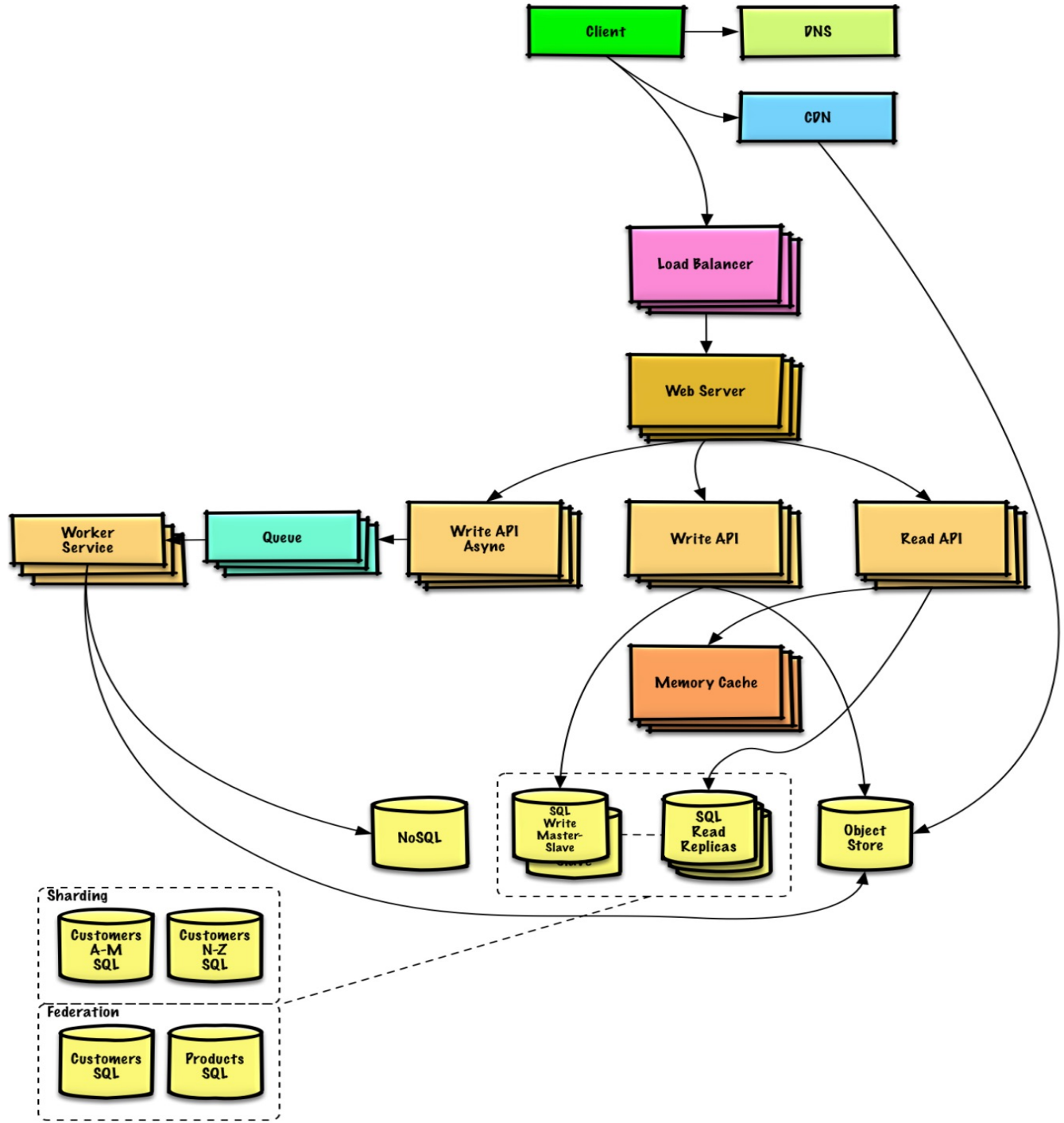
Lets Try:

$ ab -n 100 -c 10 http://127.0.0.1:8000/

-n = Number of requests = 100

-c = number of concurrent requests = 10


Network Topology
Eventual Consistency
CDN
Replication
Load-Balancing
Bandwidth
Single Point of Failure
Auto-Scaling
Sharding
Cache Invalidation
Partition Tolerance
Message Queue
DNS
Caching Policy
NoSQL-BASE
Throughput
Master-Slave
SQL Indexing
Server Heartbeat
REST
Federation
Sticky Session
Microservice
2 Phase Commit
RPC
Latency
CAP Theorem
Normailzation
Consistent Hashing
Reverse Proxy
Server Benchmarking
Denial of Service Attack


Client
DNS
CDN
Load Balancer
Web Server
Worker Service
Queue
Write API Async
Write API
Read API
Memory Cache
NoSQL
SQL Write Master-Slave
SQL Read Replicas
Object Store
Sharding
Customers A-M SQL
Customers N-Z SQL
Federation
Customers SQL
Products SQL

Stay Tuned for Next Session
Questions ?