Distributed Systems

A Primer
So you’re gonna build a Facebook killer...

- You can have some friends.
- You can post some photos.

friends[“Kenny”] = {“Stan”, “Kyle”}
photos[“Kenny”] = {p1, p2, p3, …}
Two Servers

Server 1

friends[“Kenny”].add(“Eric”)
photos[“Kenny”].add(p4)

network might deliver messages out of order

Server 2

photos[“Kenny”] = ?
friends[“Kenny”] = ?
In a distributed system, pick two of

1. Strong Consistency: Everybody should see the same ordering of events.
2. Availability: A meaningful response should be returned to clients right away.

CAP Theorem
REST API

**GET** http://api.fbkiller.com/user/Kenny/friends

```python
friends["Kenny"] = ?
```

=> \{"Stan", "Kyle"\}

**POST** http://api.fbkiller.com/user/Kenny/friends

```python
friends["Kenny"].add("Eric")
```

<= new_friend="Eric"

=> ok

**DELETE** http://api.fbkiller.com/user/Kenny/friends/Eric

```python
friends["Kenny"].del("Eric")
```

=> ok
RPC in Disguise

• Rigid Request-Reply pattern

• Poor performance on high-latency links (mobile)

• Clients must be connected to make API calls (not always possible, or waste battery by waking up wireless components)

• Mixed programming/networking concerns
Replicated Data Types

Client

friends[…]

photos[…]

sync

Server

friends[…]

photos[…]

sync
Decentralization