Cornell University
Computer Systems Laboratory

An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud and Edge Systems

Yu Gan, *Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayantara Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, Kelvin Hu, Meghna Pancholi, Yuan He, Brett Clancy, Chris Colen, Fukang Wen, Catherine Leung, Siyuan Wang, Leon Zaruvinsky, Mateo Espinosa, Rick Lin, Zhongling Liu, Jake Padilla and Christina Delimitrou*

Cornell University

ASPLOS 2019
Session Cloud I

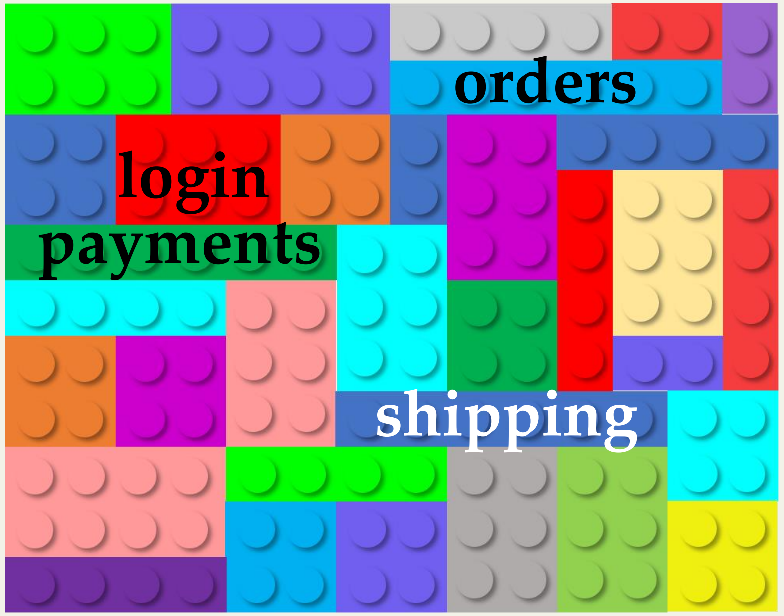
EXECUTIVE SUMMARY

CSL

- **Cloud applications migrating from monoliths to microservices**
  - Monoliths: all functionality in a single service
  - Microservices: many single-concerned, loosely-coupled services
  - Modularity, specialization, faster development
  - Datacenters designed for monoliths → microservices have different requirements
- **An end-to-end benchmark suite for large-scale microservices**
- **Architectural and system implications**
  - Hardware design
  - OS/networking overheads
  - Cluster management
  - Application & programming frameworks
  - Tail at scale

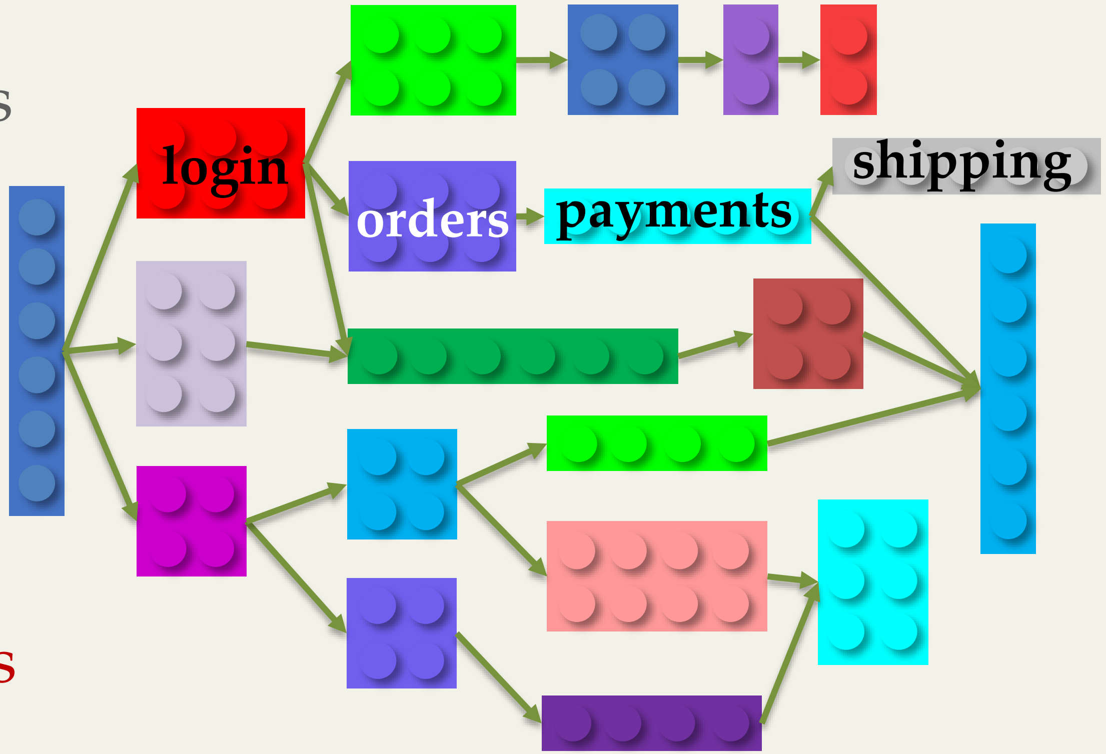
- **Monolithic applications**
  - Single binary with entire business logic

- **Limitations**
  - Too complex for continuous development
  - Obstacle to adopting new frameworks
  - Poor scalability & elasticity



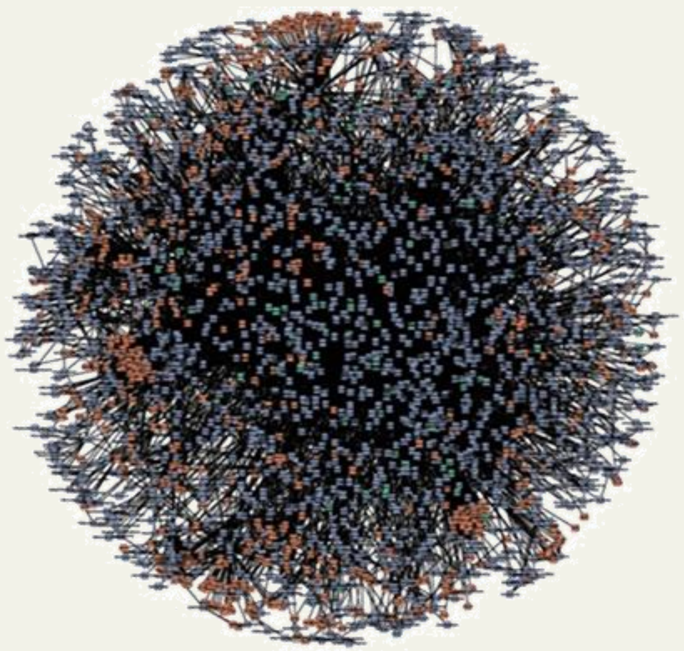
Monolith Application

- **Microservices**
  - Fine-grained, loosely-coupled, and single-concerned
  - Communicate with RPCs or RESTful APIs
- **Pros**
  - Agile development
  - Better modularity & elasticity
  - Testing and debugging in isolation
- **Cons**
  - Different hardware & software constraints
  - Dependencies → complicate cluster management

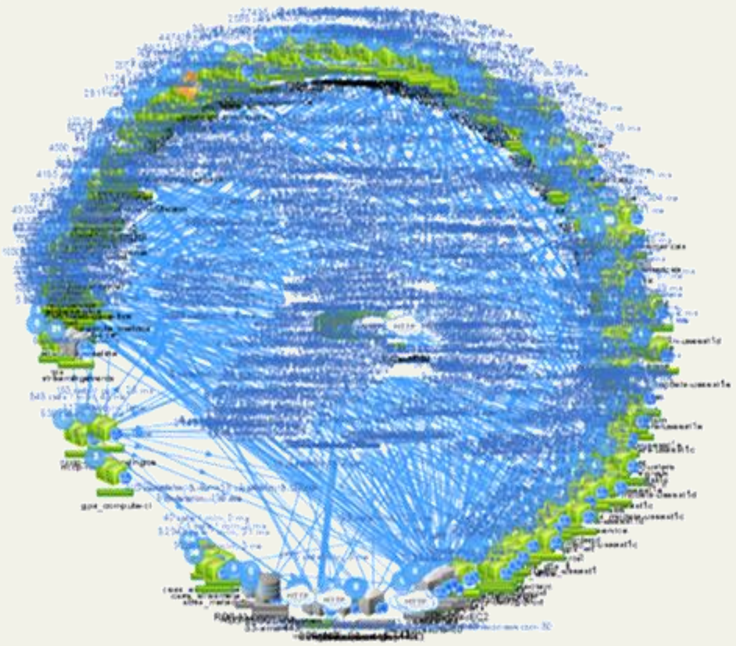


From Monoliths to Microservices

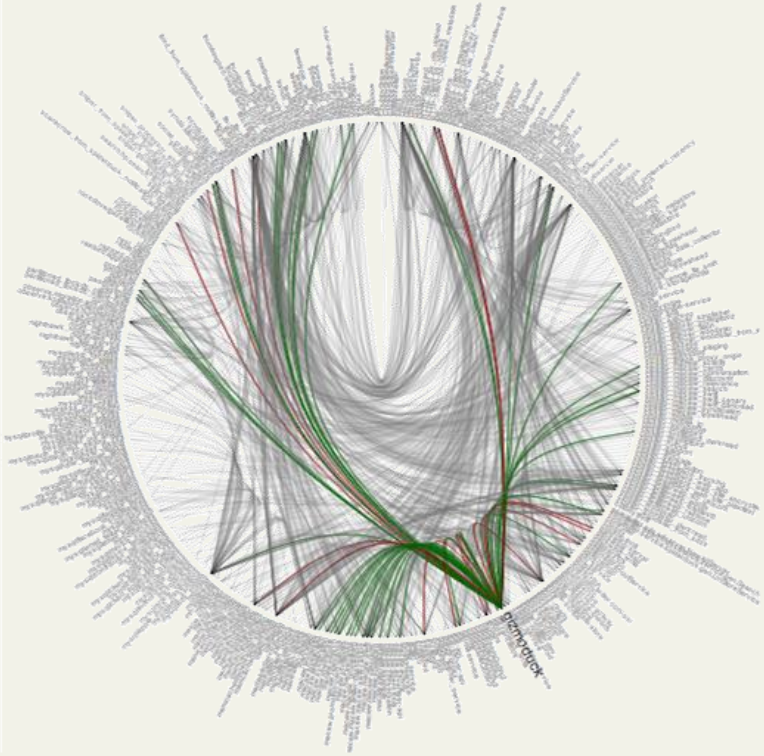
amazon.com



NETFLIX



twitter



- **Explore implications of microservices across the system stack**

5. Tail at scale

4. Application and frameworks

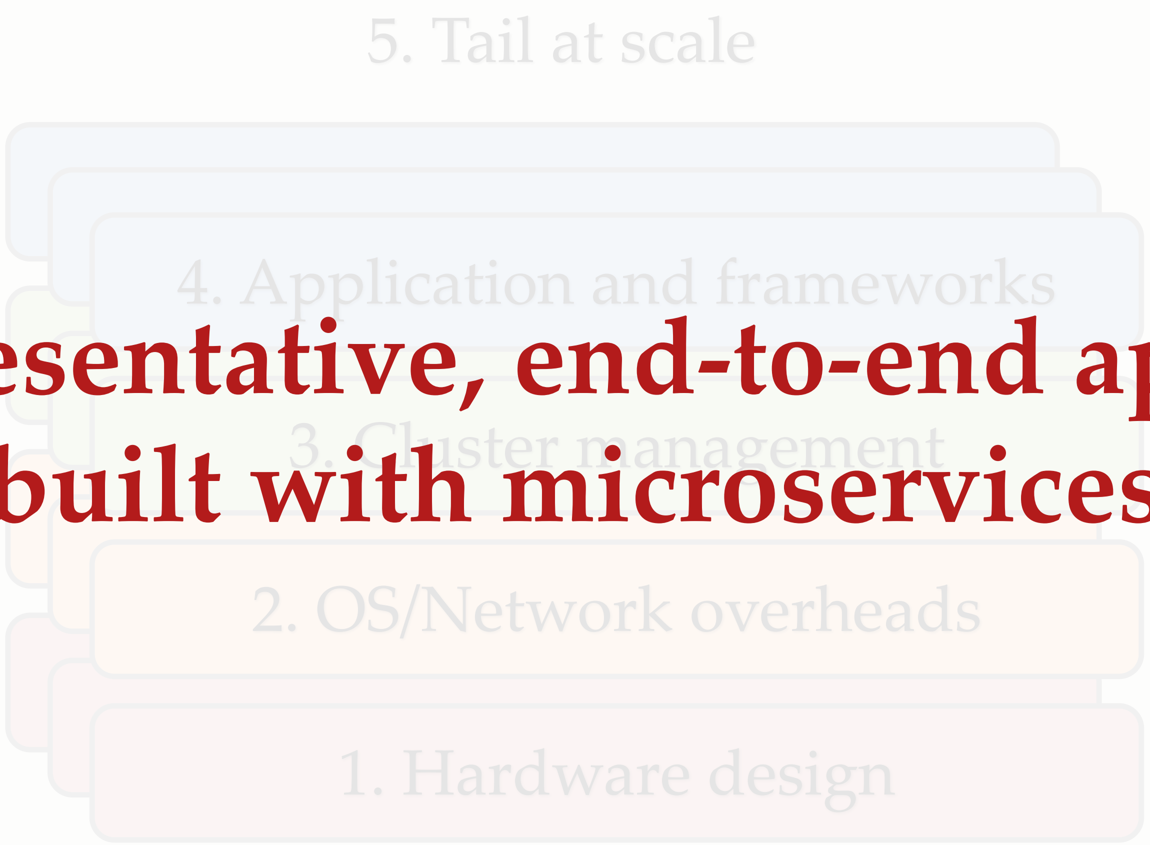
3. Cluster management

2. OS/Network overheads

1. Hardware design

Motivation

- Explore implications of microservices across the system stack

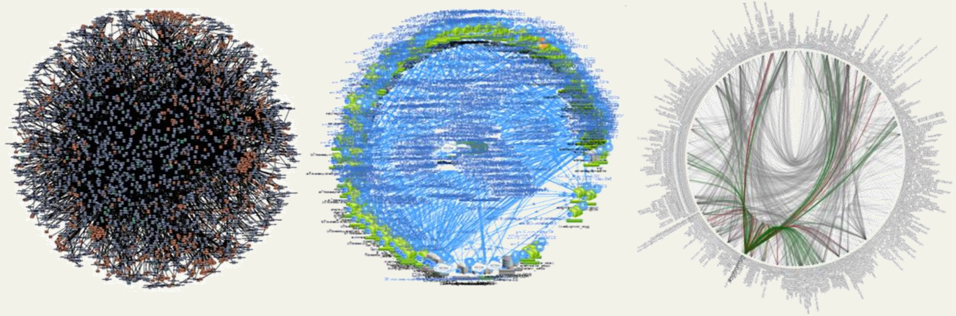


Need representative, end-to-end applications built with microservices

- **Previous work in cloud benchmarking**
  - CloudSuite [*ASPLOS'12*]
  - Sirius [*ASPLOS'15*]
  - TailBench [*IISWC'17*]
  - μSuite [*IISWC'18*]

Focus either on monolithic applications or applications with few tiers

- **DeathStarBench suite**
  - Focus on large-scale microservices that stress typical datacenter design



- **Design principles**
  - Representativeness
    - » Use of popular open-source applications and frameworks
    - » Service architecture following public documentation of real systems using microservices

- **Design principles**
  - Representativeness
  - End-to-end operation
    - » Full functionality using microservices

- **Design principles**
  - Representativeness
  - End-to-end operation
  - Heterogeneity
    - » Wide range of programming languages and microservices frameworks

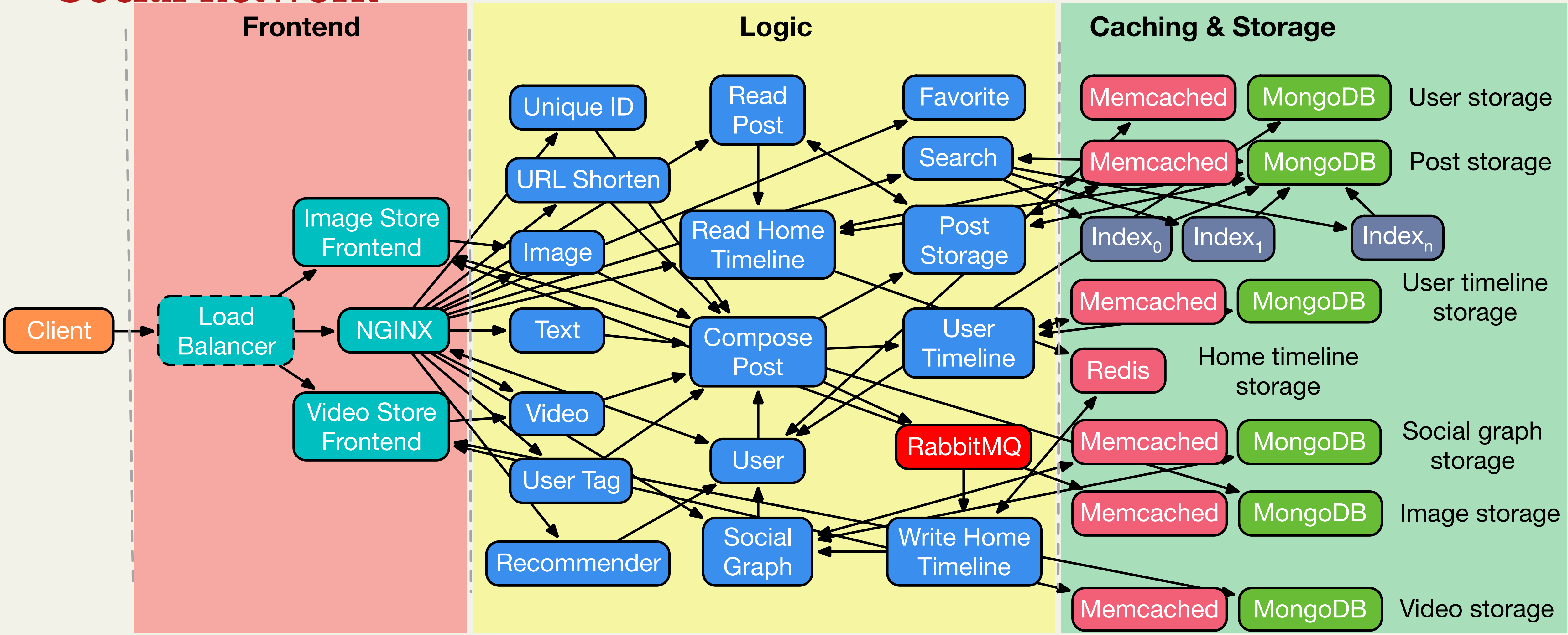
- **Design principles**
  - Representativeness
  - End-to-end operation
  - Heterogeneity
  - Modularity
    - » Single-concerned and loosely-coupled services

- **Design principles**
  - Representativeness
  - End-to-end operation
  - Heterogeneity
  - Modularity
  - Reconfigurability
    - » Easy to update or change components with minimal effort

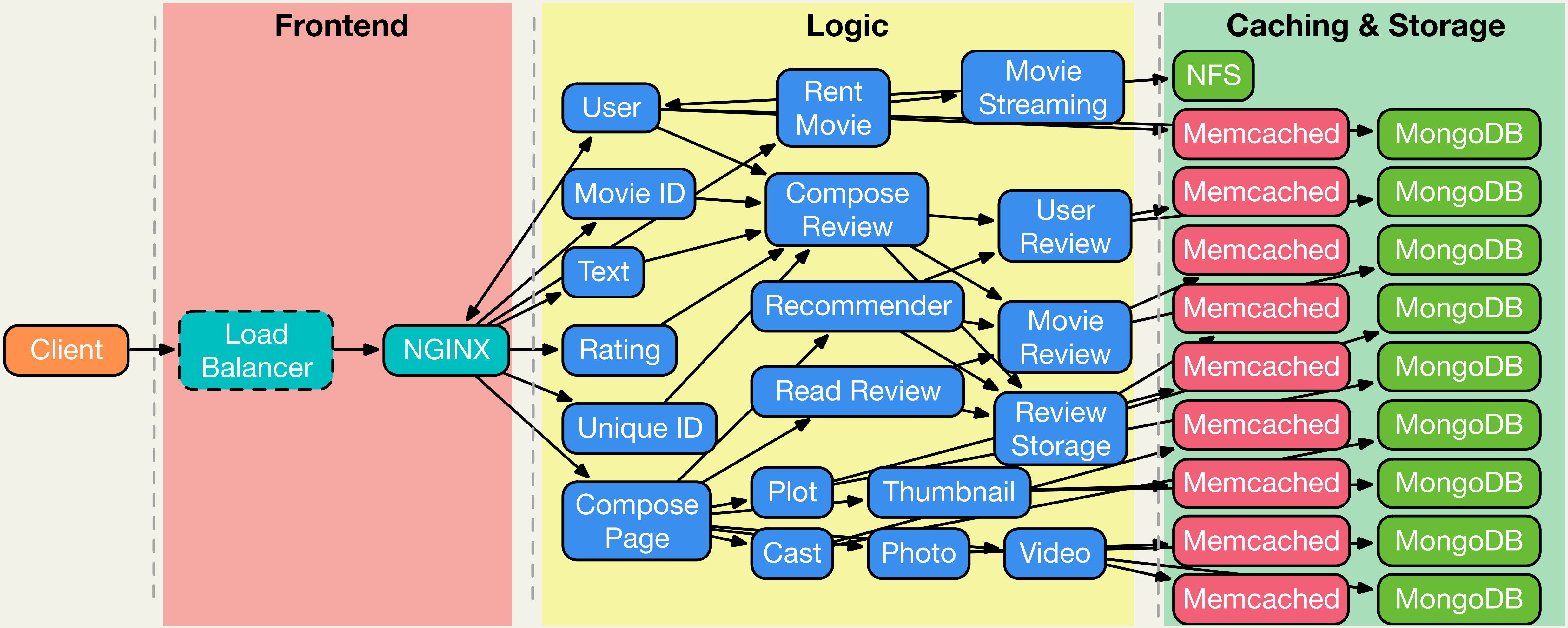
DeathStarBench Suite

- **5 end-to-end applications, tens of unique microservices each**
  - Social Network
  - Media Service
  - E-Commerce Service
  - Banking System
  - Drone Coordination System

- Social network

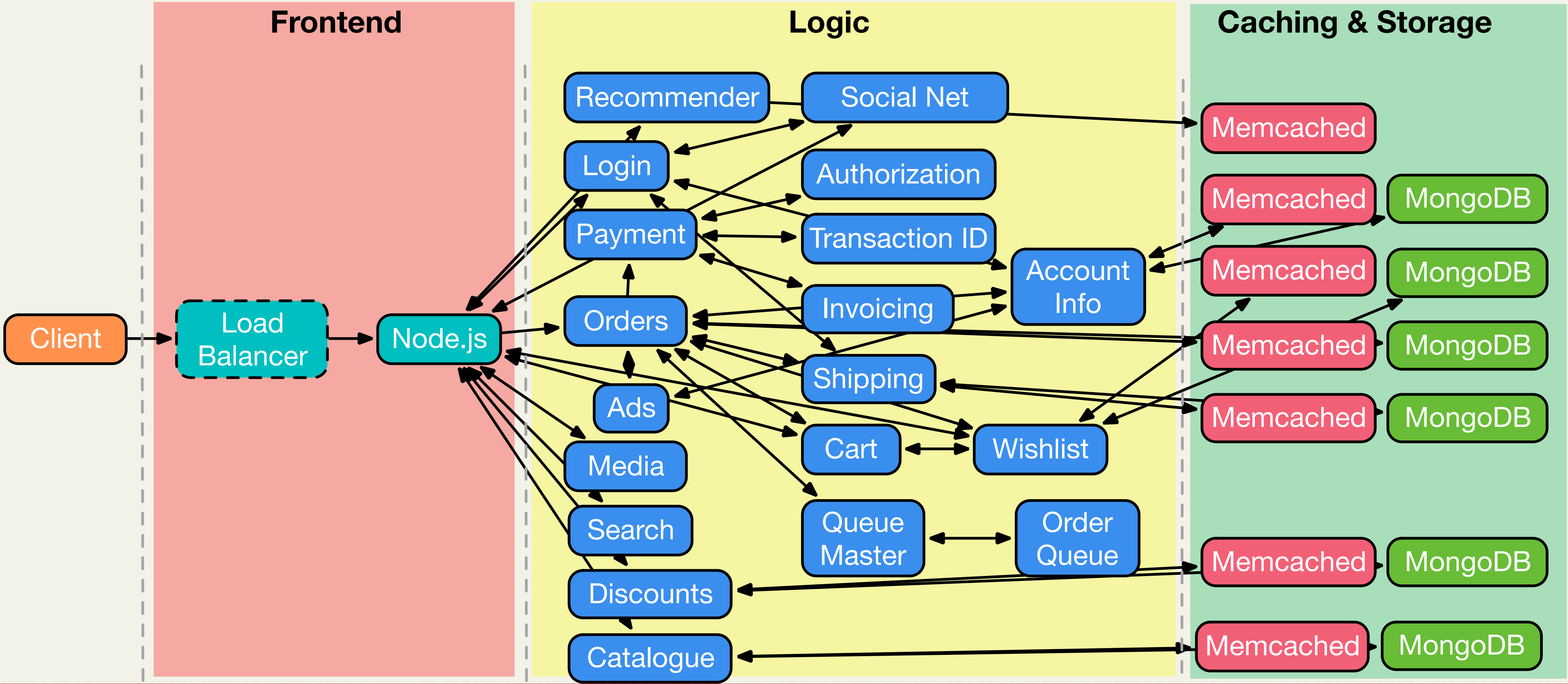


- Media service

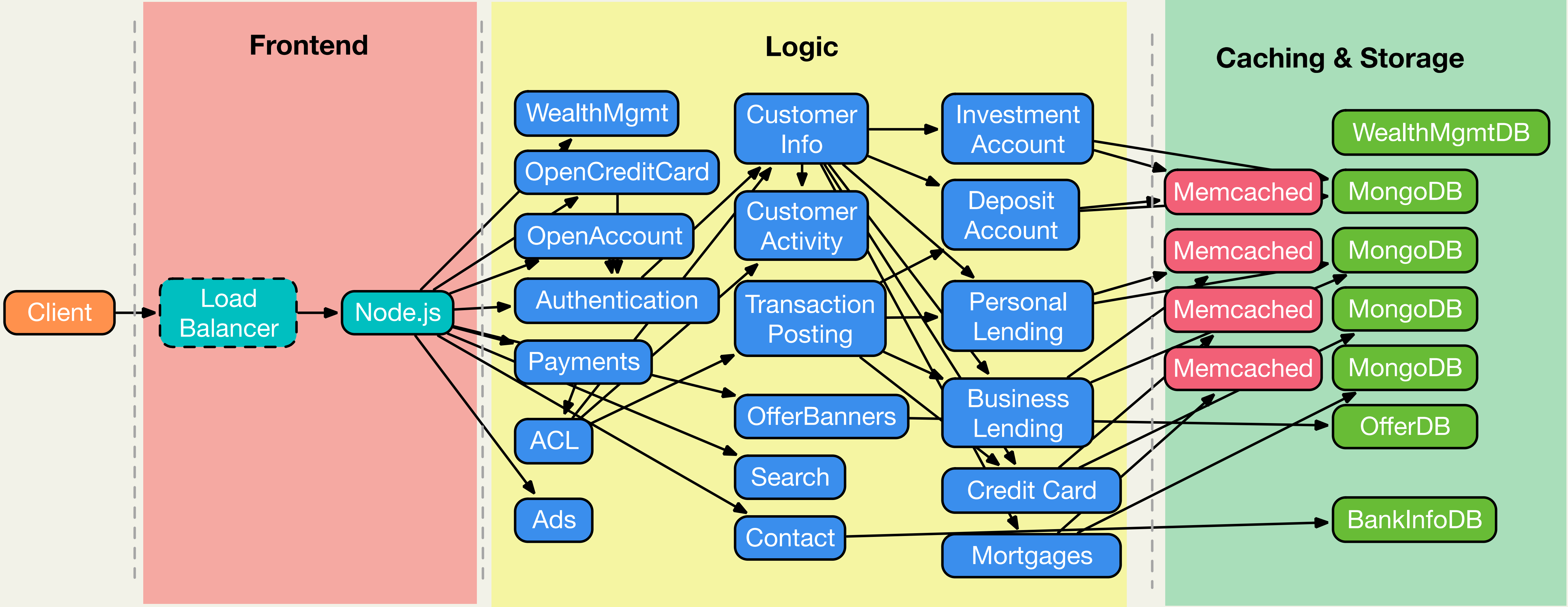


CSL

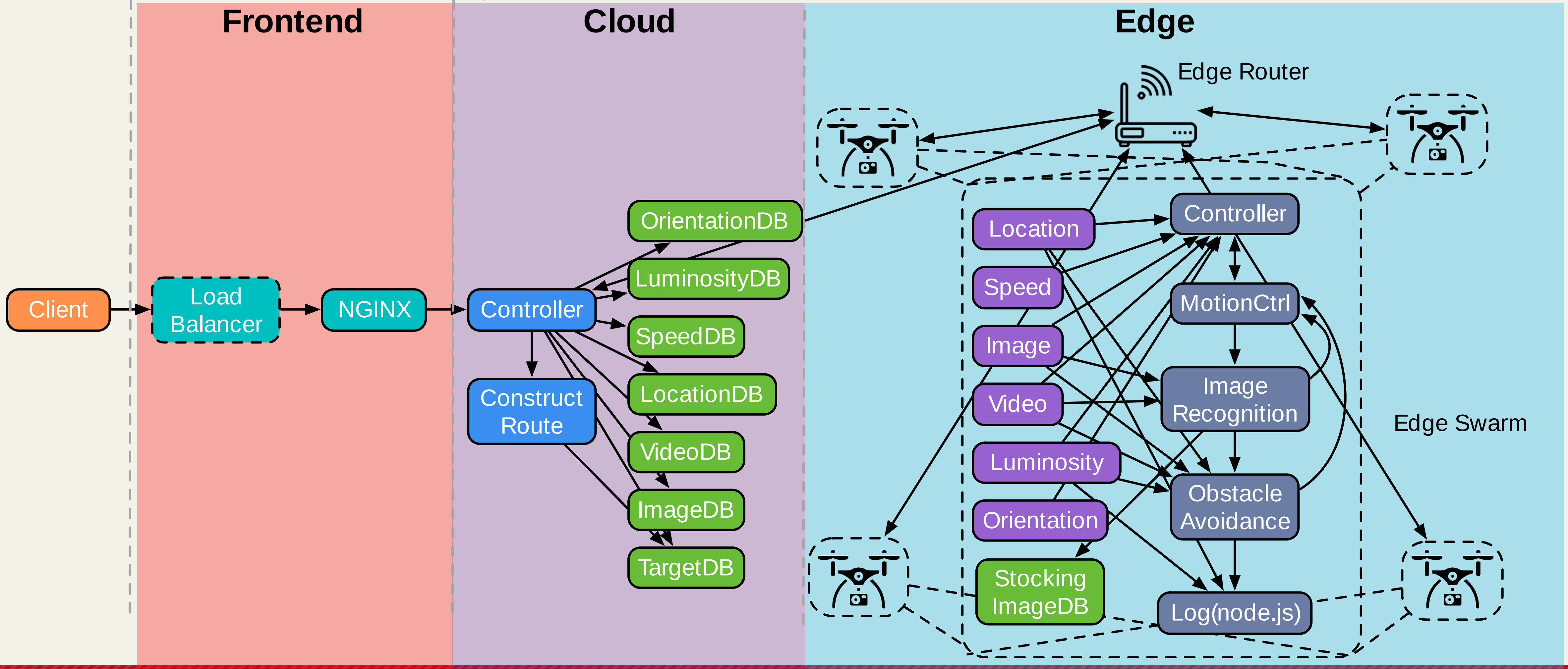
- E-commerce service



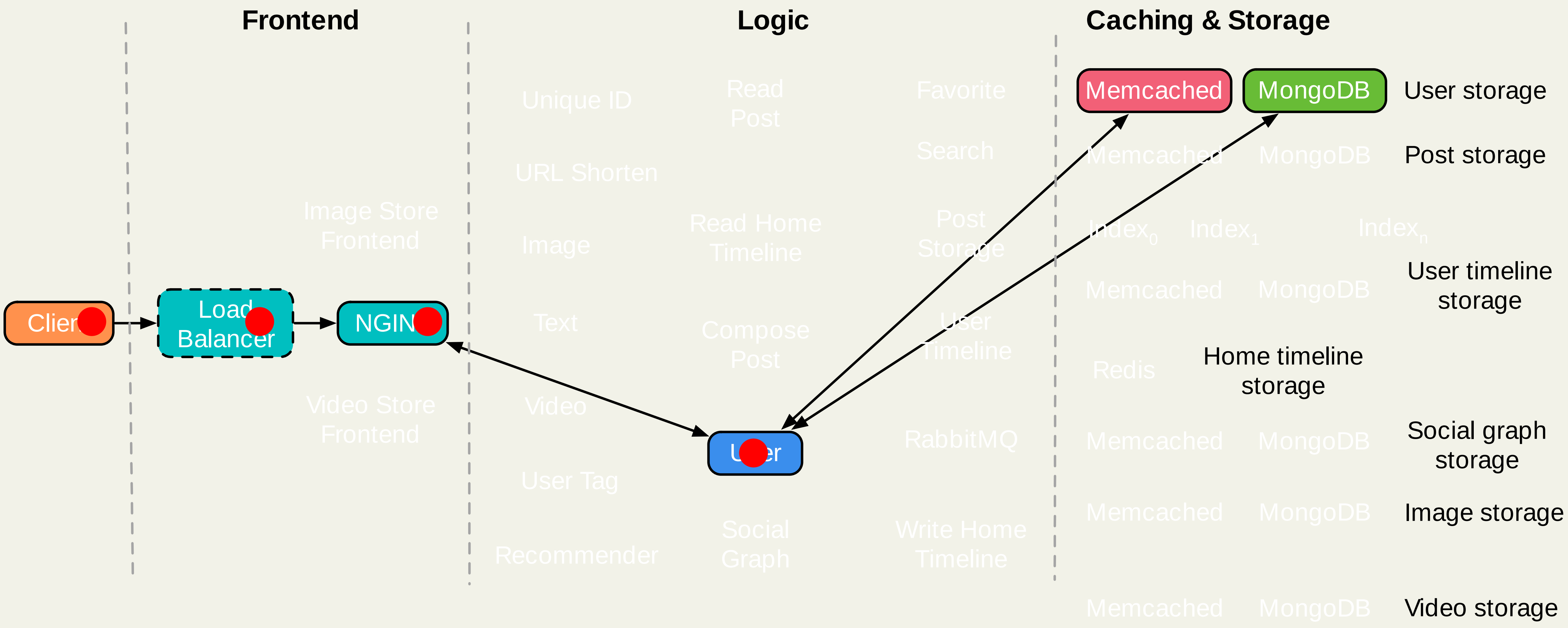
- Banking system



- **Drone coordination system**

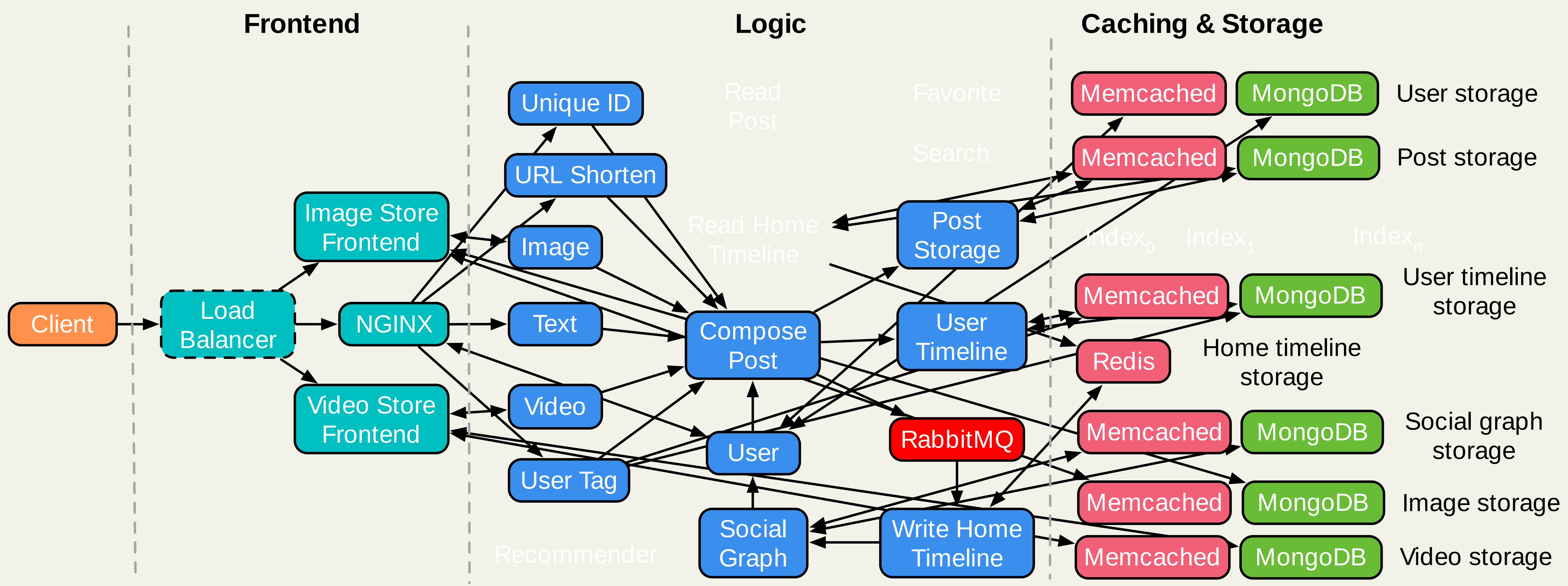


- User sign up/login

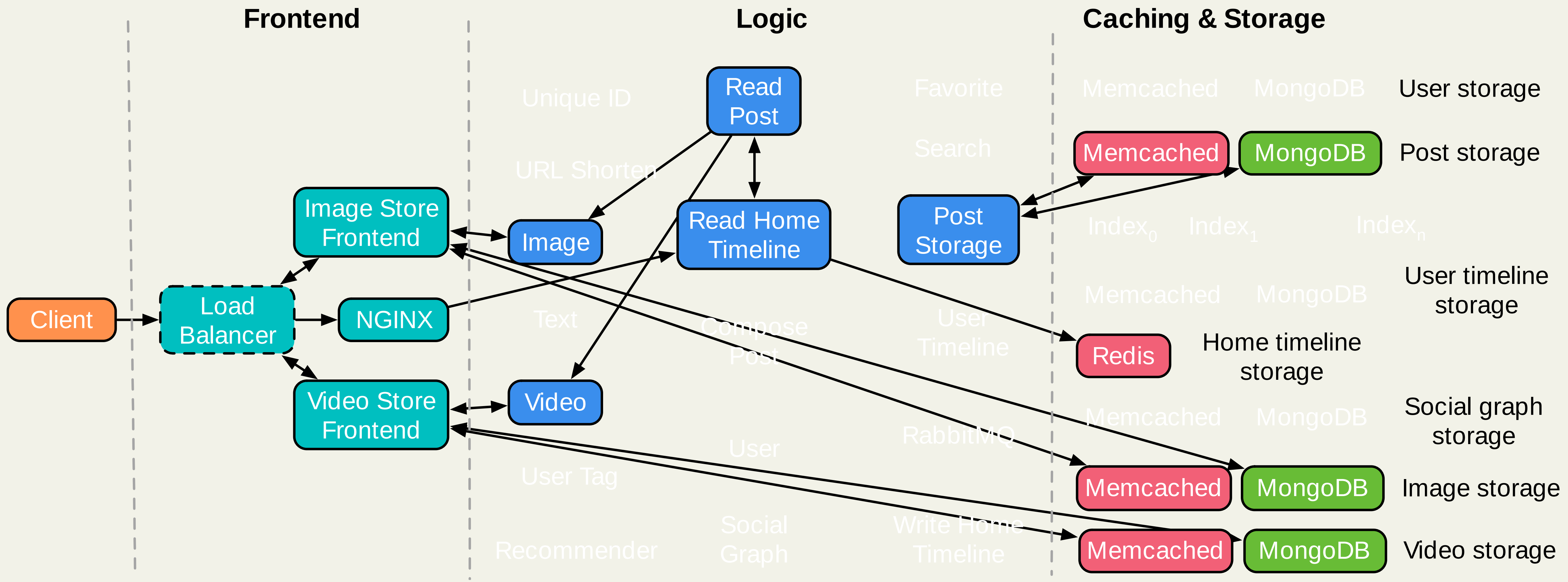


CSL

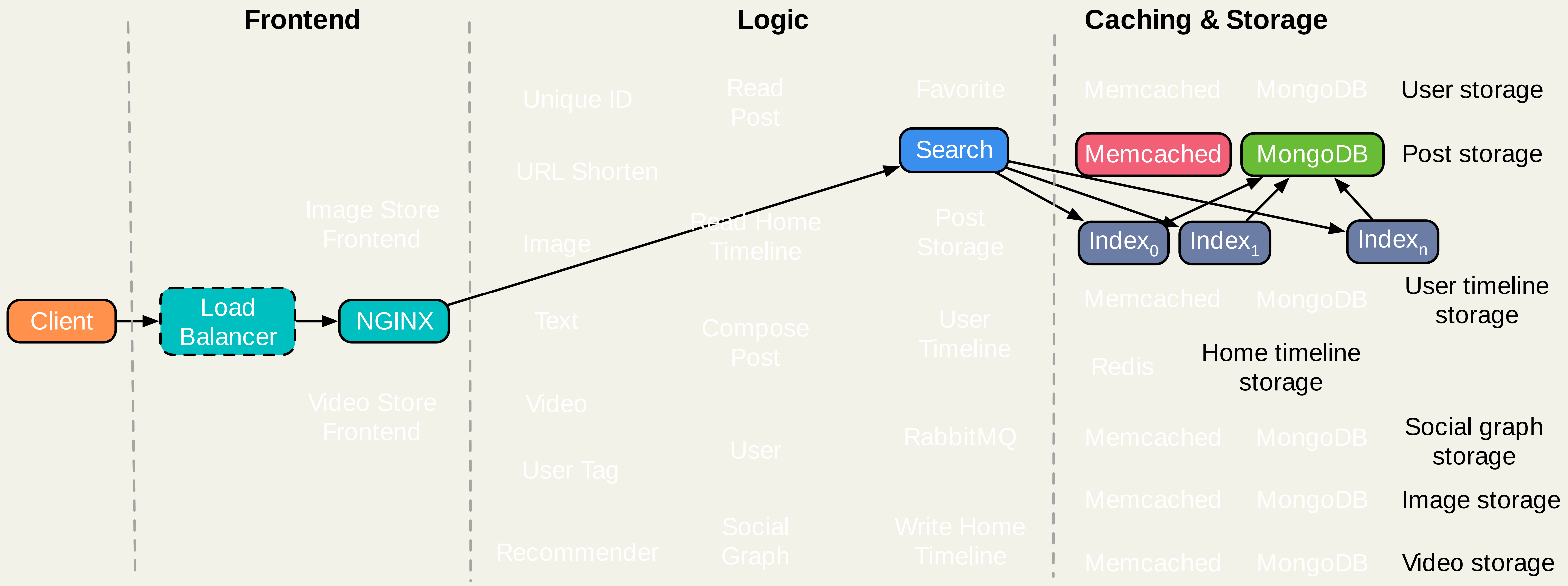
- Write posts



- ## Read home timeline

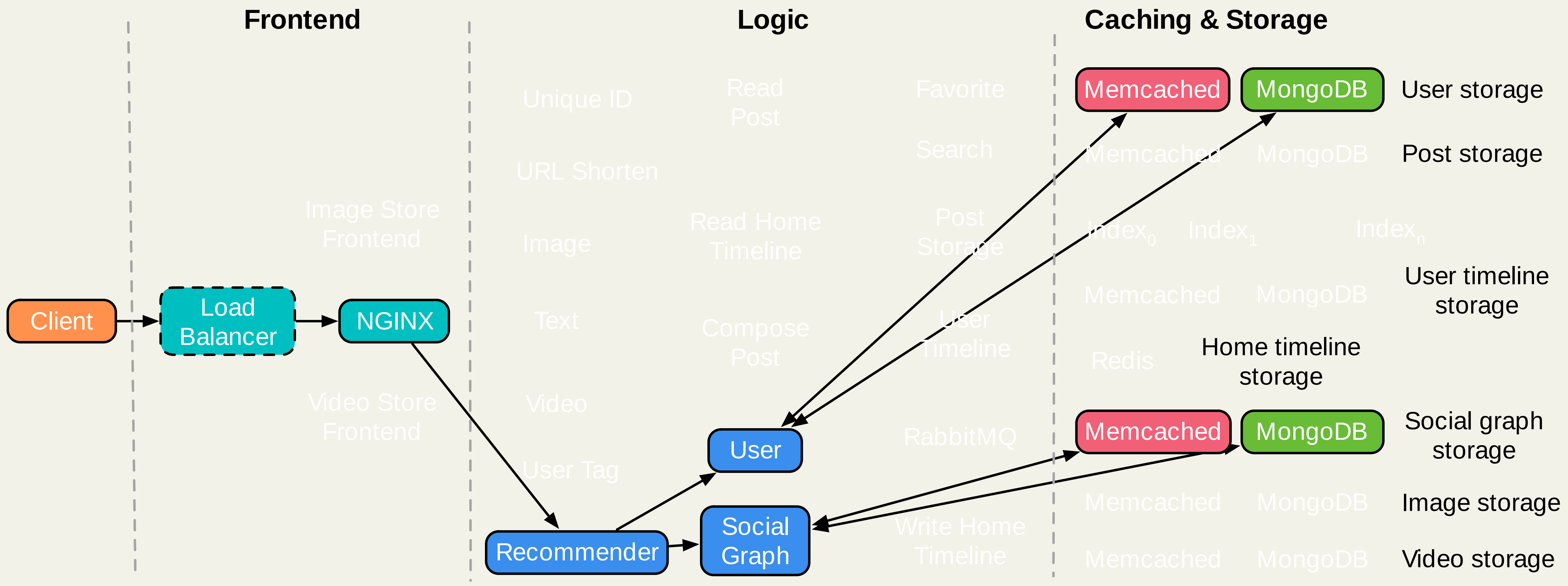


- Search



Frontend
Logic
Caching & Storage
Client
Load Balancer
NGINX
Image Store Frontend
Video Store Frontend
Unique ID
URL Shorten
Image
Text
Video
User Tag
Recommender
Read Post
Read Home Timeline
Compose Post
User
Social Graph
Favorite
Search
Post Storage
User Timeline
RabbitMQ
Write Home Timeline
Memcached
MongoDB
User storage
Memcached
MongoDB
Post storage
Index$_0$
Index$_1$
Index$_n$
Memcached
MongoDB
User timeline storage
Redis
Home timeline storage
Memcached
MongoDB
Social graph storage
Memcached
MongoDB
Image storage
Memcached
MongoDB
Video storage

- ## Recommendation



- Explore implications of microservices across the system stack

5. Tail at scale

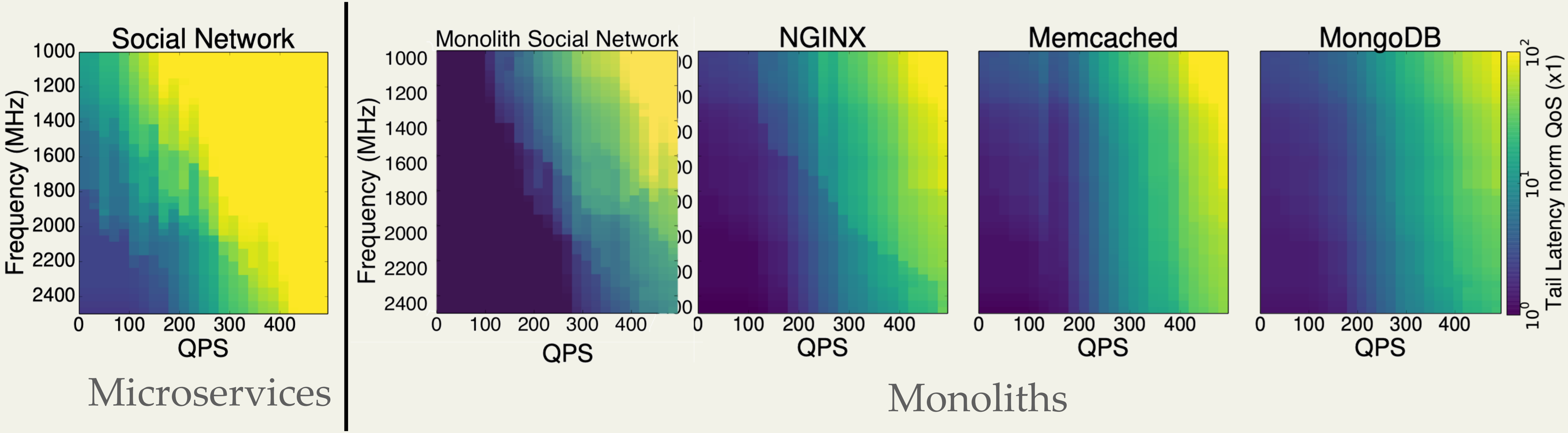
4. Application and frameworks

3. Cluster management

2. OS/Network overheads

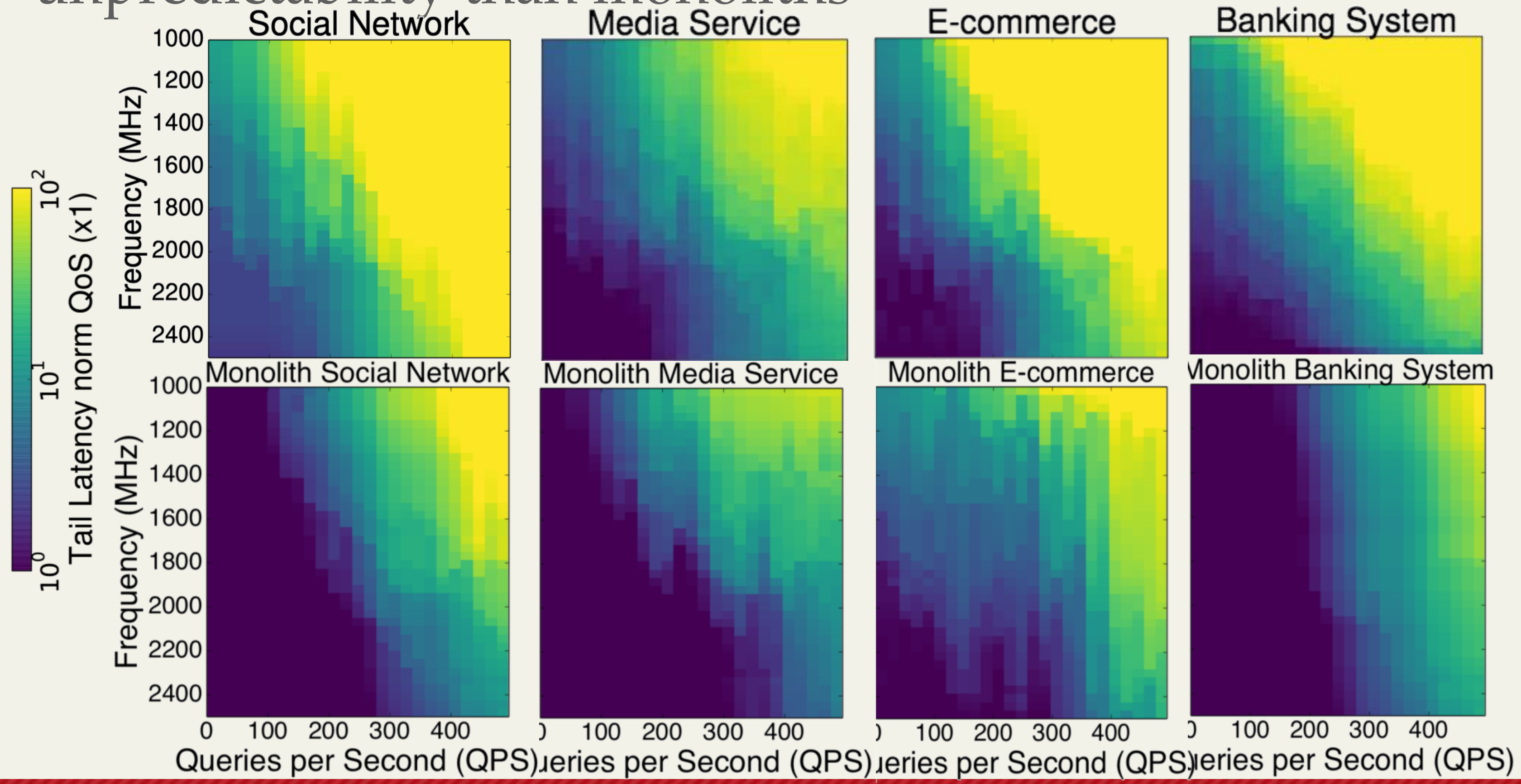
1. Hardware design

- **Brawny vs. wimpy cores**
  - Microservices are more sensitive to performance unpredictability than monoliths



CSL

- **Brawny vs. wimpy cores**
  - Microservices are more sensitive to performance unpredictability than monoliths



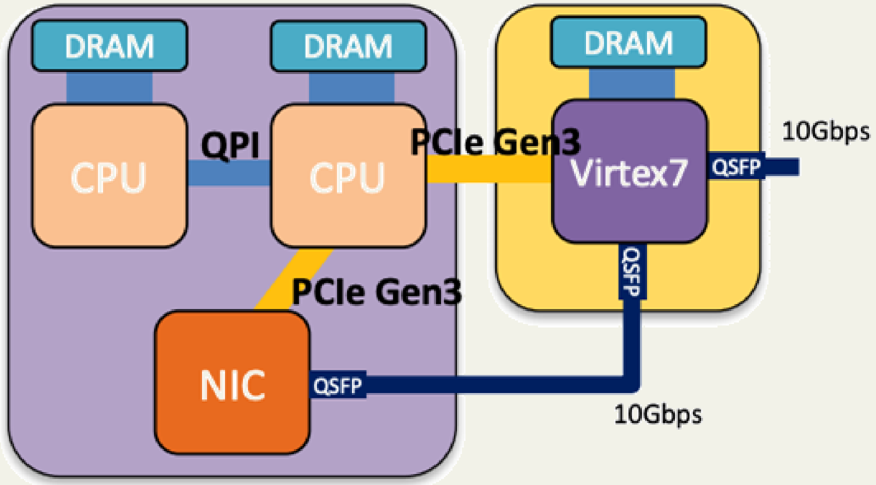
Microservices

Monoliths

- **Brawny vs. wimpy cores**
  - Microservices are more sensitive to performance unpredictability than monolithic apps
  - Xeon vs Cavium servers
- **Cycle breakdown of each microservice**
  - Smaller fraction of frontend stalls than monoliths
- **I-cache pressure**
  - Lower I-cache pressure than monoliths

OS/Network overheads

- **RPC overheads**
  - A large fraction of time spent in network stack
- **FPGA network acceleration**
  - Offload TCP stack on FPGA
  - 10 – 68x improvement on network processing latency
  - 43% - 2.2x improvement on end-to-end latency

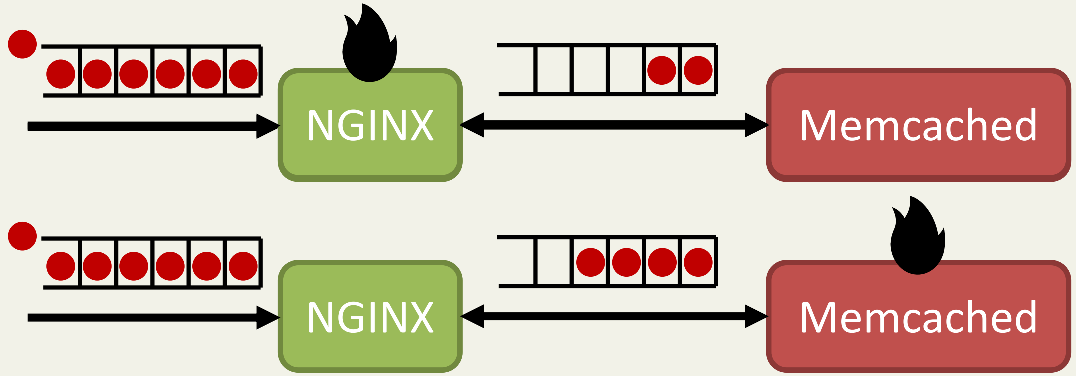


- **Latency back-pressure**
  - Bottleneck services pressure upstreaming services
  - Cause: Imperfect pipelining
    - HTTP/TCP HoL blocking
    - Limited number of worker threads/connections

Example: HTTP 1.1 HoL blocking

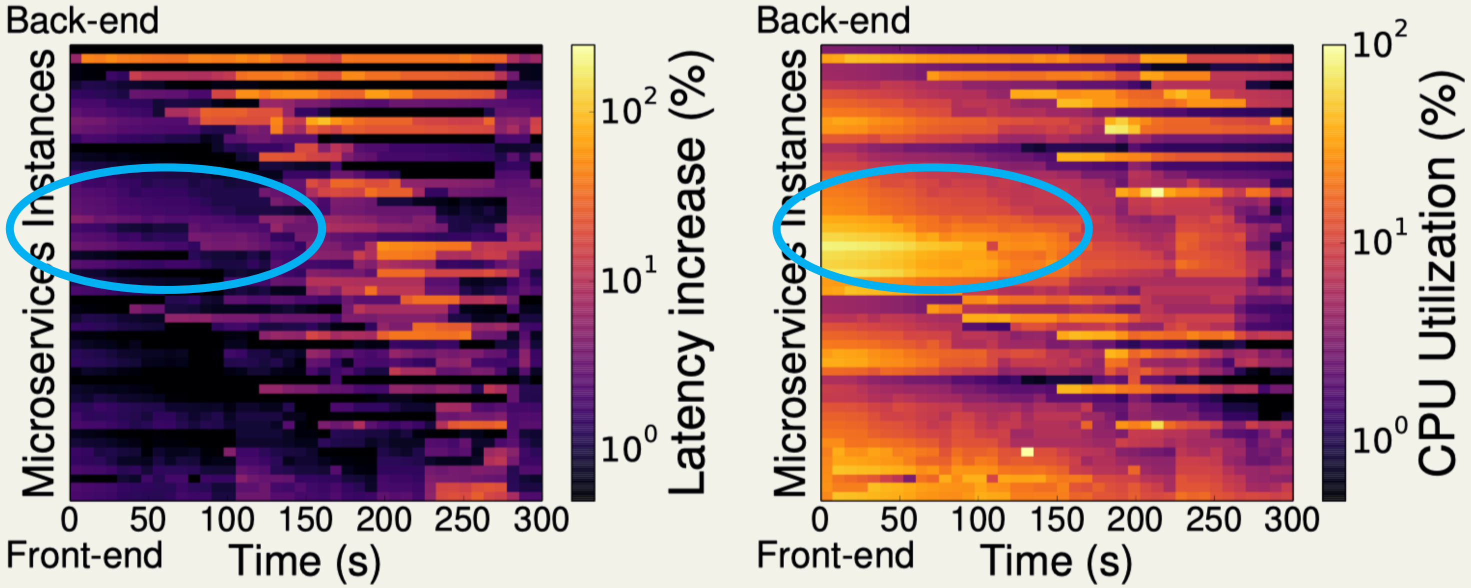
NGINX bottleneck

Memcached bottleneck



CSL

- **Cascading QoS violations**
  - Hotspots propagating along the dependency graph
  - No obvious correlation to CPU utilization
  - Difficulty in discovering the bottleneck and long time to recover from QoS violations



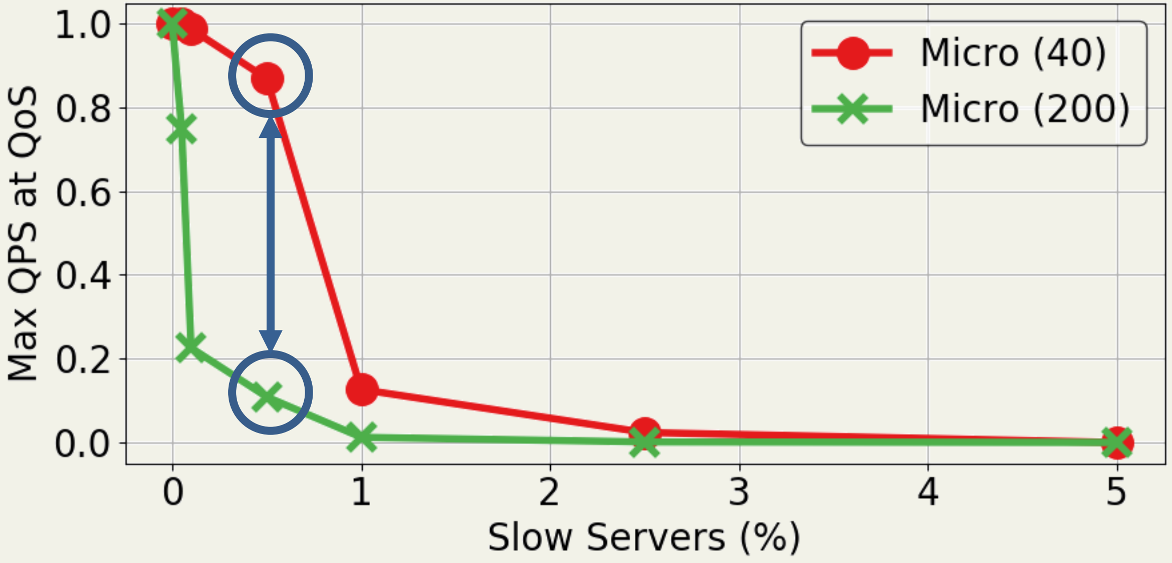
- **Serverless frameworks**
  - Compared long-running microservices on EC2 with short–running microservices on AWS Lambda
  - Agile resource adjustments with diurnal load pattern
  - Higher performance variability due to
    - » No control of lambda placement
    - » Communication through S3
    - » Loading of dependencies

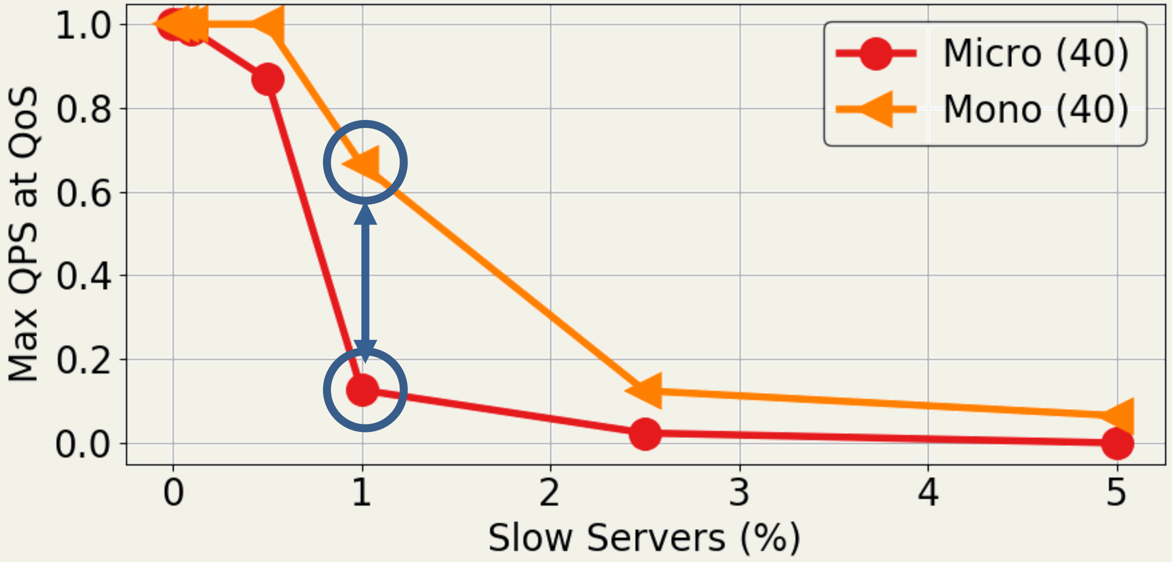
Impact of slow servers

- Larger cluster → larger impact of slow servers
- More severe tail latency increase compared to monoliths

▶ 5. Tail at Scale

4. Application and frameworks
3. Cluster management
2. OS/Network overheads
1. Hardware design





Conclusions

- **Cloud applications from monoliths to microservices**
- **Study implications of microservices across the system stack**
- **Open-source benchmark suite for cloud and IoT microservices**
- **Explored the implications of microservices**
  - More sensitive to performance unpredictability
  - Potential of hardware acceleration for networking
  - Need for cluster managers that account for dependencies
  - Tail at scale effects more prominent in microservices

QUESTIONS?

- **Cloud applications from monoliths to microservices**
- **Study implications of microservices across the system stack**
- **Open-source benchmark suite for cloud and IoT microservices**
- **Explored the implications of microservices**
  - More sensitive to performance unpredictability
  - Potential of hardware acceleration for networking
  - Need for cluster managers that account for dependencies
  - Tail at scale effects more prominent in microservices