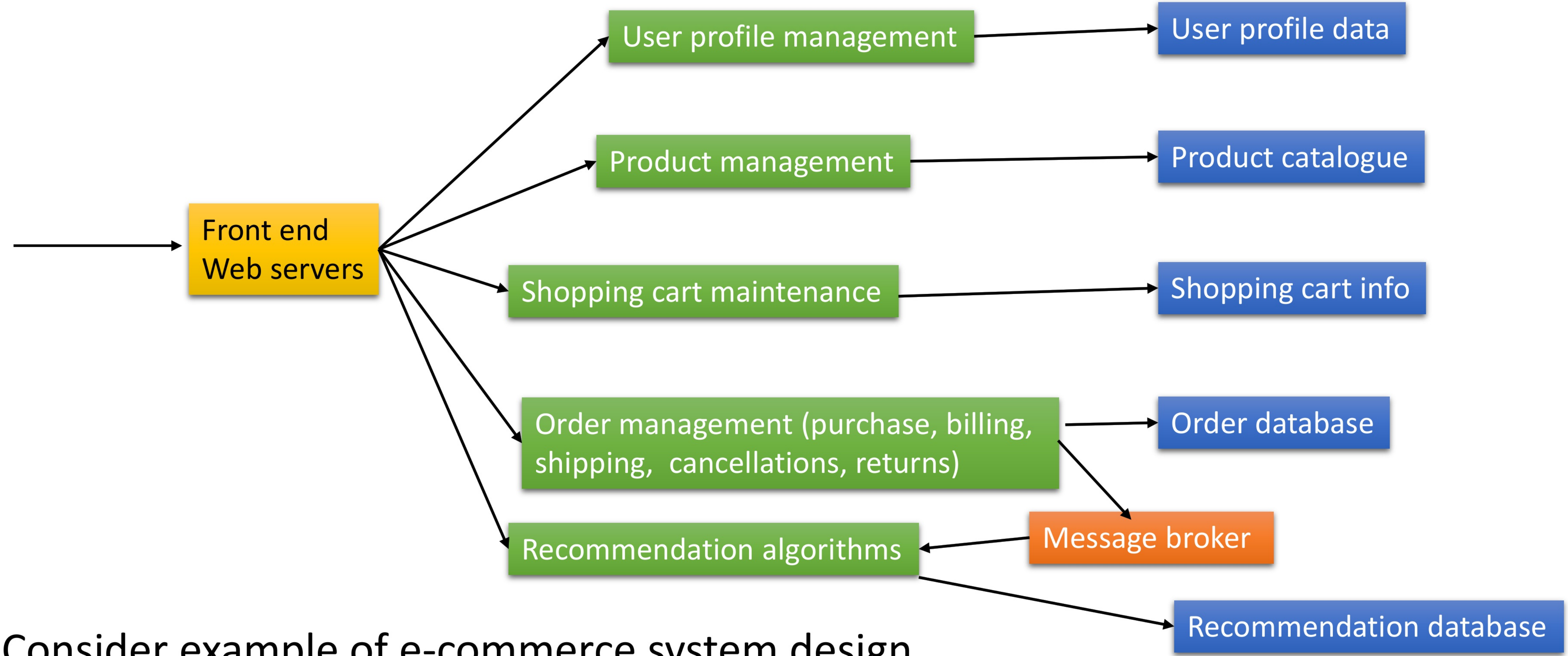
Design and Engineering of Computer Systems

Lecture 40: Case studies of distributed systems design

Mythili Vutukuru

IIT Bombay

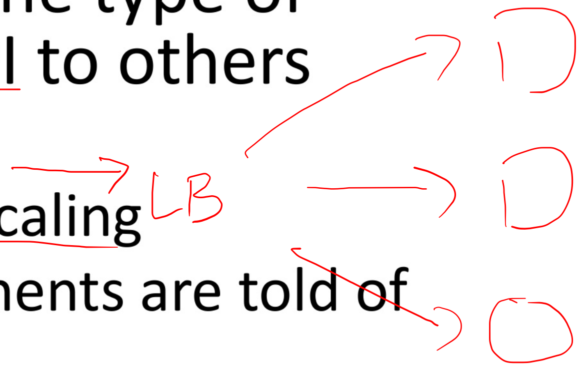
Reliability engineering: Putting it all together



- Consider example of e-commerce system design
- Multi-tier architecture of various front-end, application servers, data stores

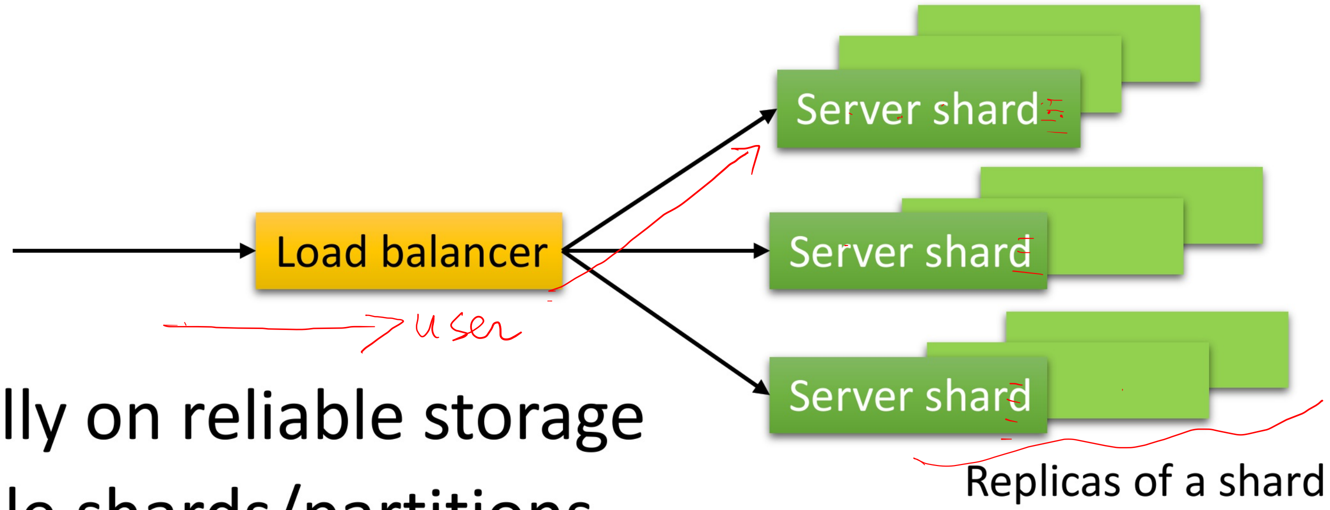
Fault-tolerant design of application servers

- Application server: system component that handles one type of requests, maintains one type of app data, exposes API to others
  - E.g., shopping cart server or order management server
  - Most servers have multiple replicas/shards for horizontal scaling
  - Load balancer distributes load to replicas, or other components are told of multiple replicas and distribute load themselves



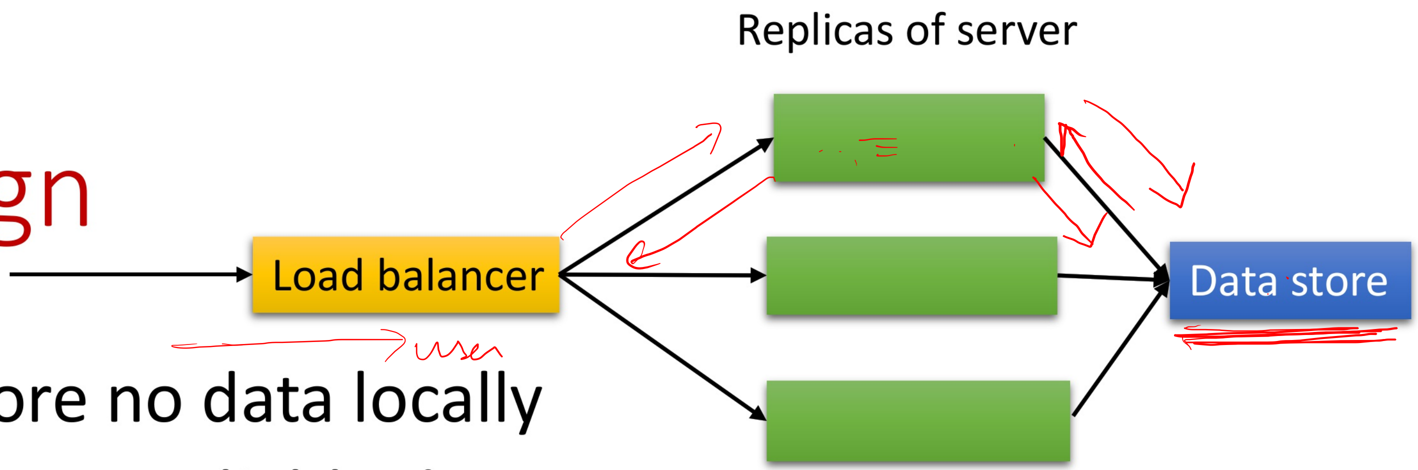
- How to make any app server reliable and robust to failures?
- Techniques for reliability depend on type of server design
  - Stateless server: app server stores all state (e.g., shopping carts of users) in a remote data store, each server replica is stateless
  - Stateful server: app data is stored and maintained within server replica itself

Stateful server design



- Application server stores data locally on reliable storage
- App data is partitioned into multiple shards/partitions
  - Load balancer needs to redirect requests of a user to the server shard that has the data of the user (user stickiness)
  - Key-server mappings explicitly stored, or use consistent hashing
- Within each partition, multiple replicas for fault tolerance
  - Active-active (replicated state machines) or active-passive (check-pointing)
  - Different replication logic based on consistency-availability tradeoff
- Each server replica implements mechanisms for atomicity (e.g., write ahead logging), for single server or distributed transactions
- Same server replica can be part of multiple shards (need not be disjoint)

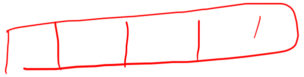
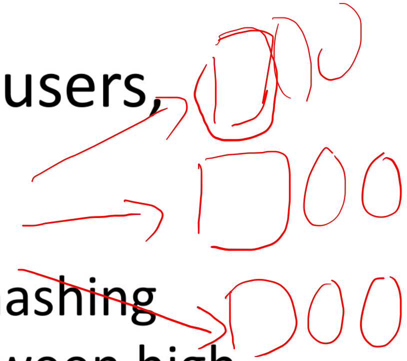
Stateless server design



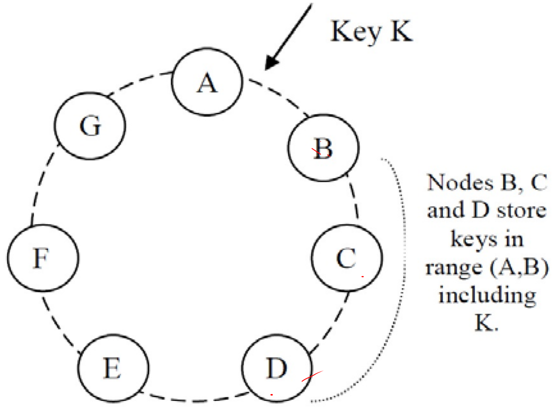
- Application server replicas store no data locally
- All app data is stored in a remote, reliable data store
- Load balancer can redirect any request of any user to any replica, no need of user stickiness, simpler load balancing
- Handling every request requires fetching data from data store, processing request, storing state back in data store (higher overhead)
  - Confirmation of request sent only after state has been put in data store
- No special mechanisms needed for reliability at server
  - All complexity now in the reliable data storage component
- Hybrid designs between fully stateful and fully stateless are possible

Distributed data stores

- Distributed data stores for reliable data storage in stateless applications
- Several high-performance data stores used in computer systems
  - Key-value stores for unstructured data: Amazon's Dynamo
  - Semi-structured data (column families exist but no fixed schema): Google's Bigtable, Apache Cassandra
  - Structured data: Google's Spanner supports SQL-like queries
- Data stores designed for Internet-scale applications, e.g., billions of users, millions of requests/second, low latency responses
  - Flexible data schema, not as structured as traditional RDBMS
  - Scalable design to handle large traffic, e.g., via partitioning with consistent hashing
  - Replication within each partition for fault tolerance, design choices vary between high availability and strong consistency
  - Varying amounts of support for distributed transactions
  - In-memory designs for quick access, disk-based storage for persistence

Distributed data stores: example



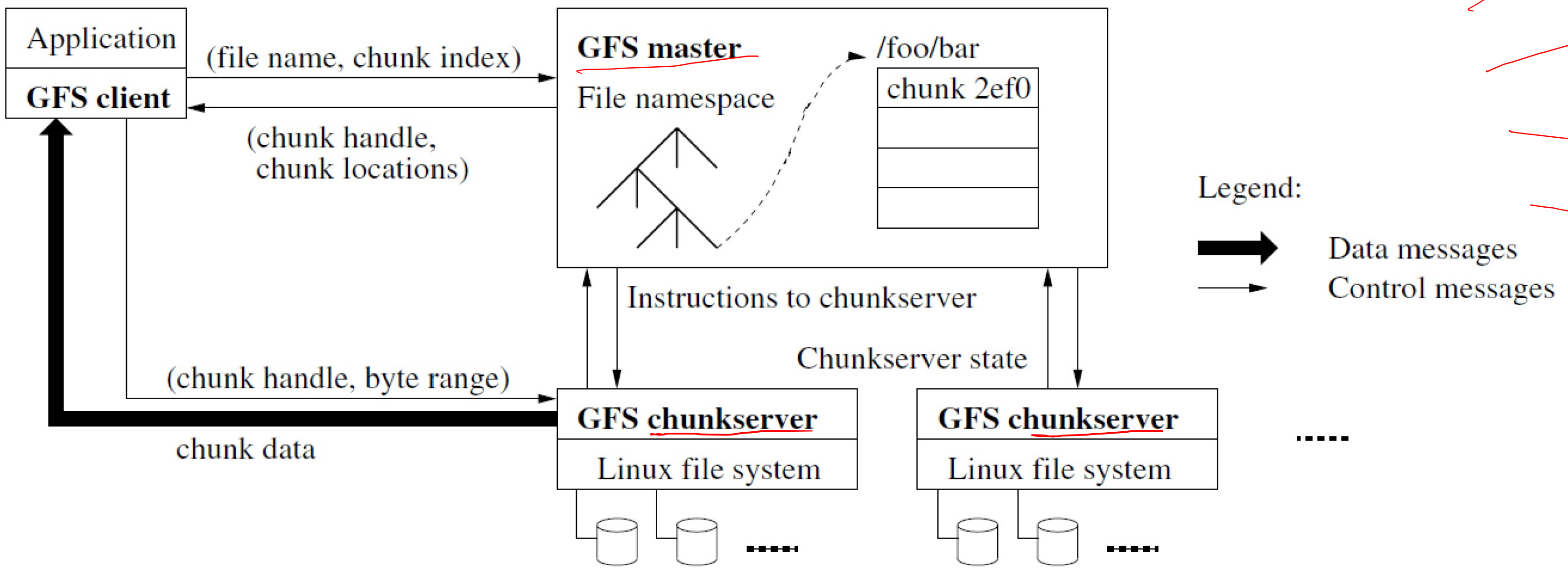
- Dynamo is a distributed key-value store
  - Simple get/put interface, unstructured data
- Partitions the keys over the set of nodes using consistent hashing
  - Every key is stored at N nodes following the key on the circular ring
- Shared-nothing architecture: each replica independently stores state
- Put operation: the key is written to a subset W of the N nodes
  - Succeeds even if some subset of nodes are unavailable
- Get operation: the key is read back from some subset R of the N nodes
  - Eventual consistency: get may not return latest put
  - Multiple values can be returned, application has to reconcile
- Dynamo chooses R,W,N such that R+W > N, so that the latest value can be returned most of the times, but no guarantee

Data storage in cloud systems

- Large scale computer systems use various types of storage systems to manage large amounts of application data
  - Network attached storage (NAS): reliable file storage appliances
  - Distributed file systems: persistent file storage, implemented not on single machine but across a cluster of file storage servers
  - Distributed configuration management: configuration and other important system information stored in a reliable service accessible to all servers
  - Distributed shared memory / remote memory: DRAM-like memory from a cluster accessible over the network
  - Traditional relational databases, with distributed and scalable designs

Distributed File Systems

- Google File System: distributed file system on commodity hardware
  - Designed to efficiently store a small number of large files (not POSIX API)
  - GFS cluster has one master and multiple chunk servers (Linux machines)
  - File divided into fixed size chunks, chunks replicated at multiple chunk servers
  - Chunks stored at chunk servers on local disk, identified by a unique handle
  - Master stores chunk handle → chunk server mapping

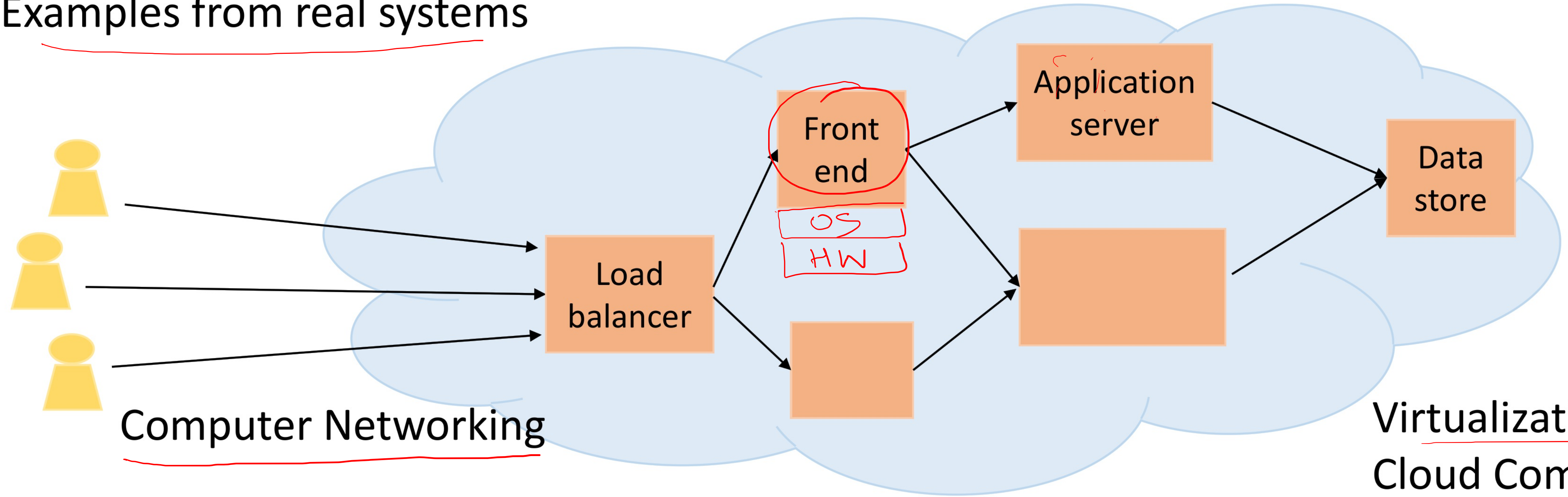


Summary: Design and Engineering of Computer Systems

Principles of Designing Computer Systems

End-to-end view of Computer Systems Design

Examples from real systems



Computer Networking

Operating Systems and Hardware

Virtualization and Cloud Computing

Performance Engineering

Reliability Engineering