SE124 (2023 Fall)

Scalability in Practice:
a Highly scalable web app

Yubin Xia

IPADS, Shanghai Jiao Tong University

https://www.sjtu.edu.cn

Scalable web apps are everywhere: pre-AI era

PayPal™
11.6 million payments/day

12306
9.56 million tickets/day

High request rate

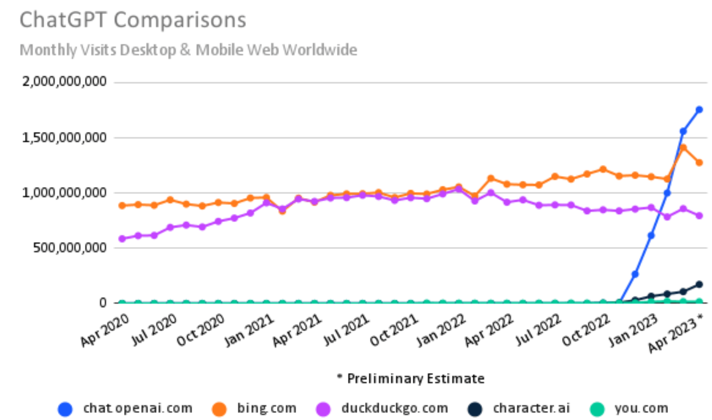
– 100 billions of requests **daily**

Massive data

– **Facebook** has more than **1 billion** of images uploaded **weekly**, **Baidu** stores **tens of billions** of web pages

Scalable web apps are everywhere: AI era

ChatGPT



Essentially, also a web APP

- Pre-AI era: request handling = order an item
- AI era: request handling = use model to do an in

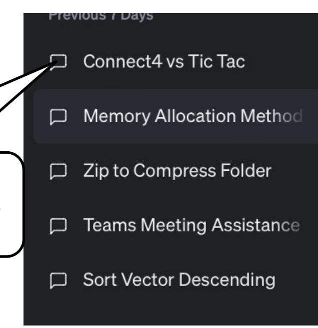
The same system requirements

- High request rate
- Massive data: both for training & for serving hist

New requirements

- Huge computation power required
- According to GPT (◀◀), an inference of GPT ~= sorting 100,000,000 numbers

Per-user sessions



The fundamental system techniques remain similar

AI is just another (challenging) workload

Case study:
a e-commerce website

Different storage systems to support different services

Database

File system

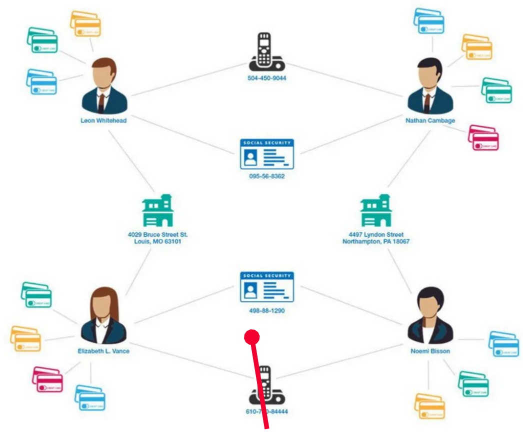
Key-value store

Computation frameworks to support different services

Hot topics

Batch processing system

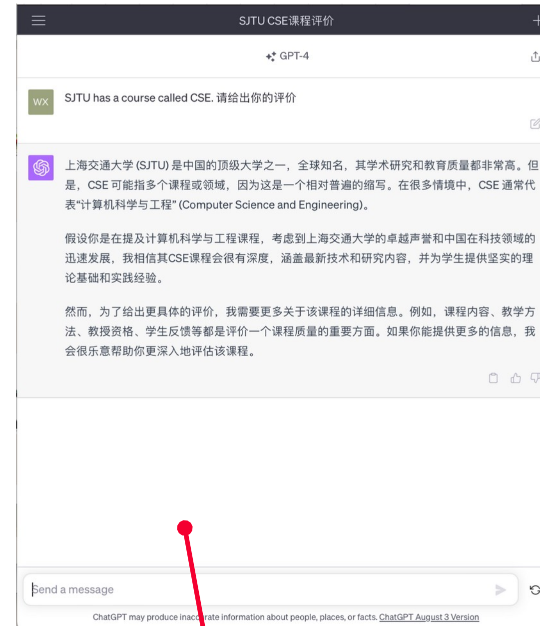
Fraud detection



Graph processing system

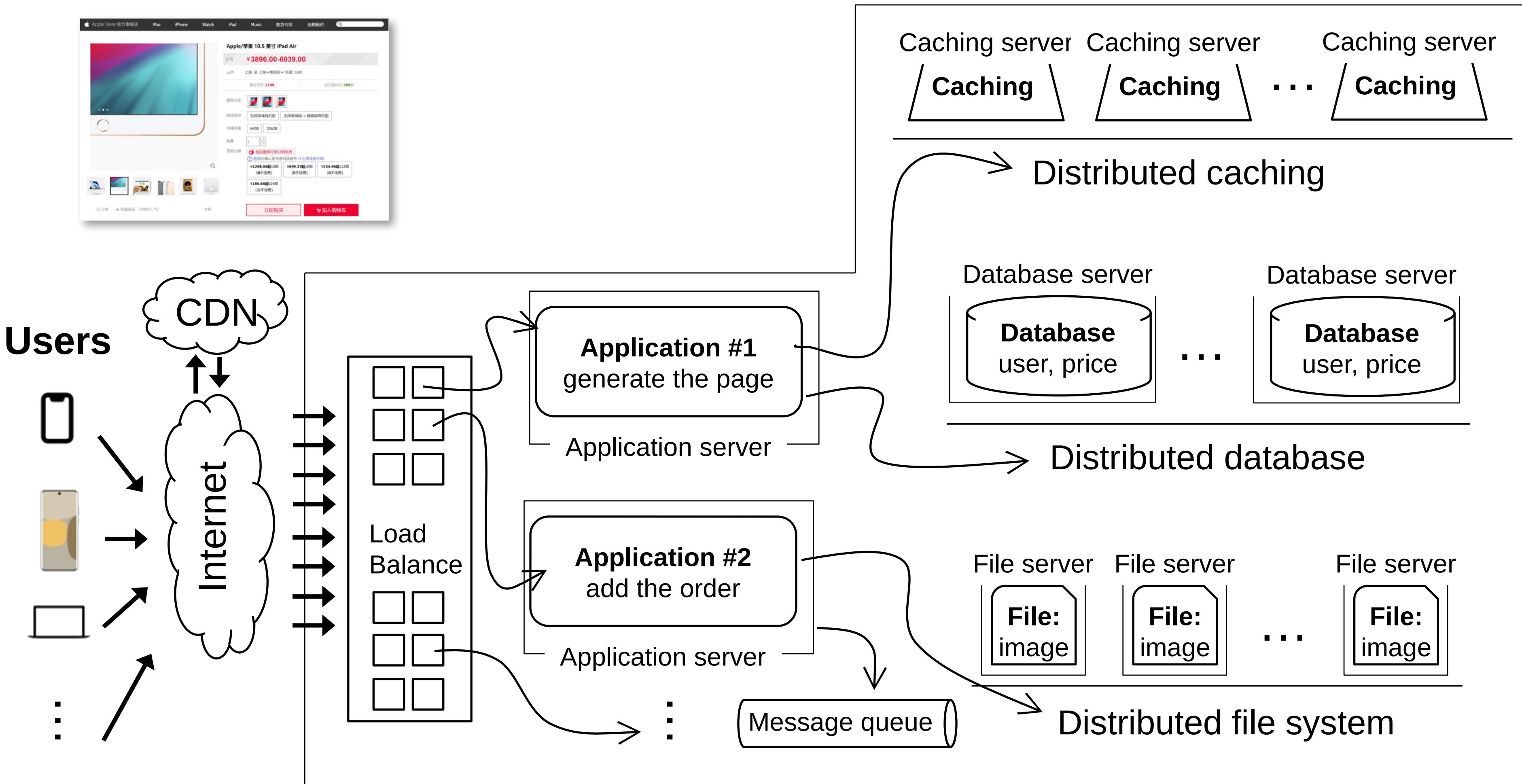
Source: https://www.graphable.ai/blog/graph-database-fraud-detection/

Chat Bot



ML systems

Each click needs thousands of servers to cooperate



How to build Taobao on a single machine (in old days)?

Operating system:

– **L**inux (in OS class)

Serving the requests: web server

– **A**pache, Nginx (in ICS)

Serving the data: file system & DB

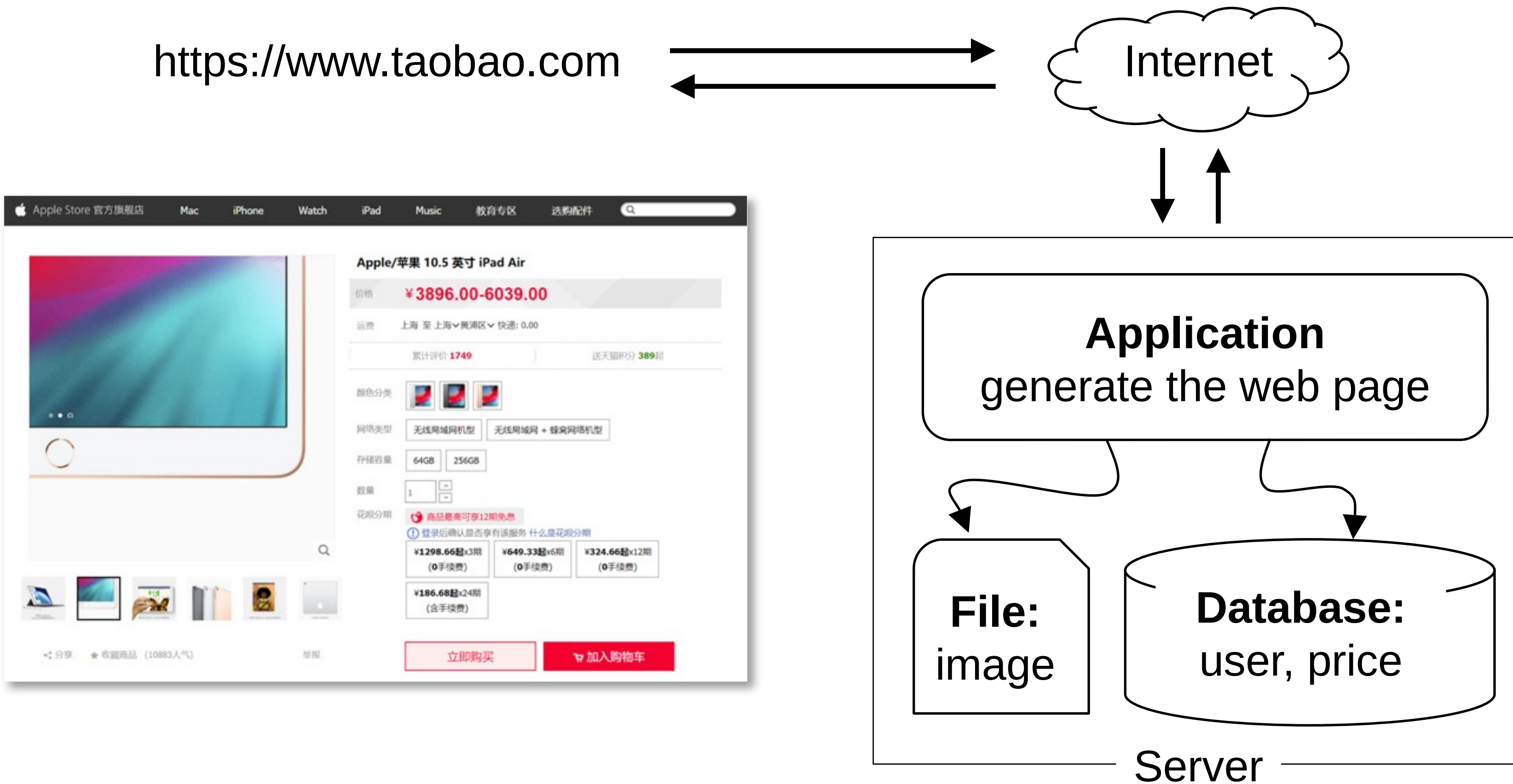
– **M**ySQL & inode file system (in CSE)

Displaying the page: HTML

– **P**HP (in Web class)

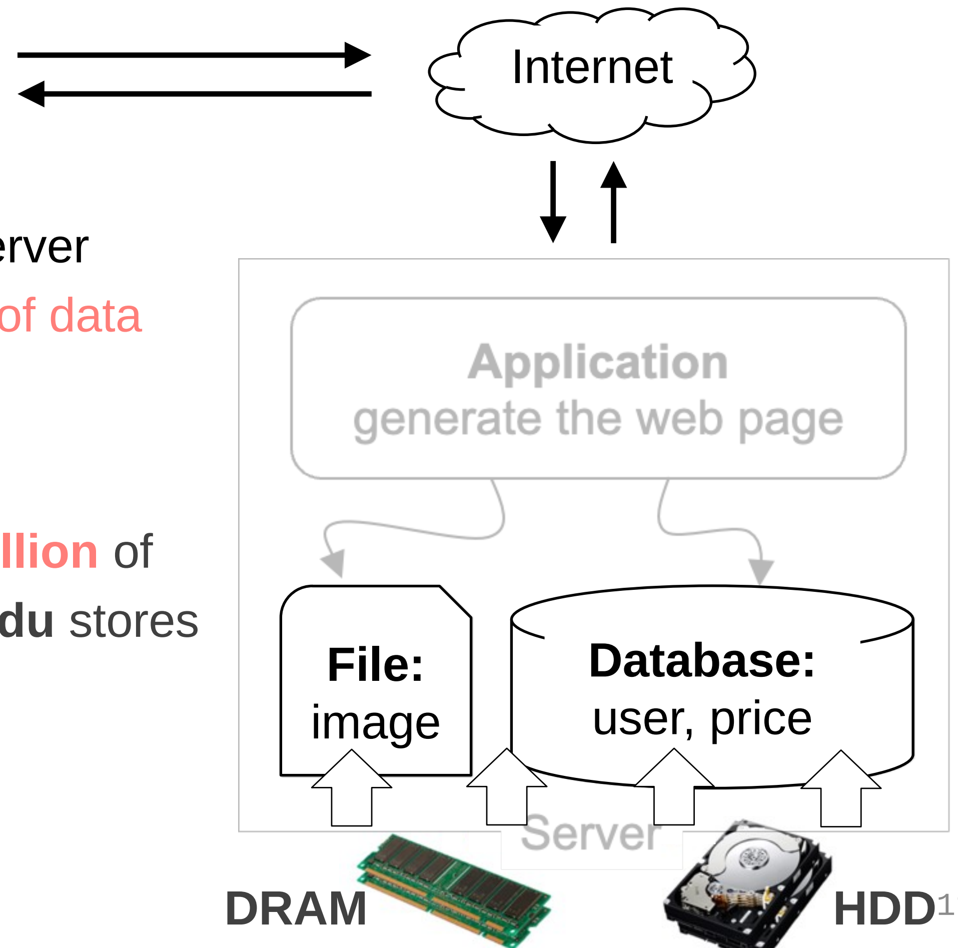
LAMP = Linux + Apache + MySQL + PHP

Using one server to build Taobao



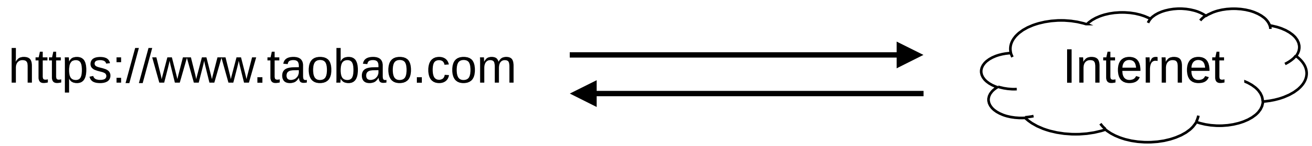
The architecture of LAMP cannot scale!

https://www.taobao.com



1. The **disk & memory** of one server cannot store massive amount of data
   - **DRAM**: 64 – 256 GB
   - **HDD**: 2 – 40 TB
   - **Facebook** has more than **1 billion** of images uploaded **weekly**, **Baidu** stores **tens of billions** of web pages

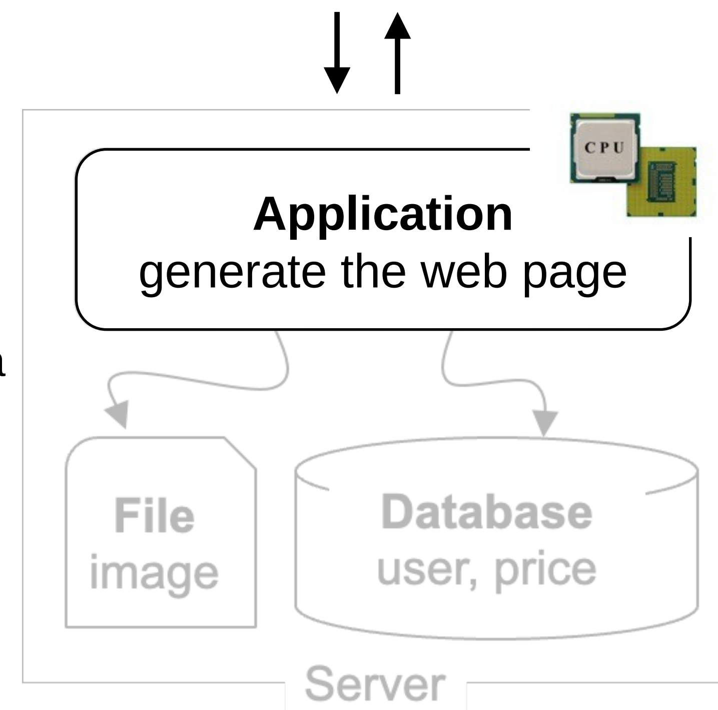
The architecture of LAMP cannot scale!



2. Application uses the CPU for processing, while **a single CPU can hardly scale** due to the end of **Moore's law & Dennard scaling**

– **Moore's law**: the number of transistors in a dense integrated circuit (IC) doubles about every two years.

– **Dennard scaling:** as transistors get smaller, their power density stays constant



Step #1 for scalability: disaggregating application & data

Application: handles application logic

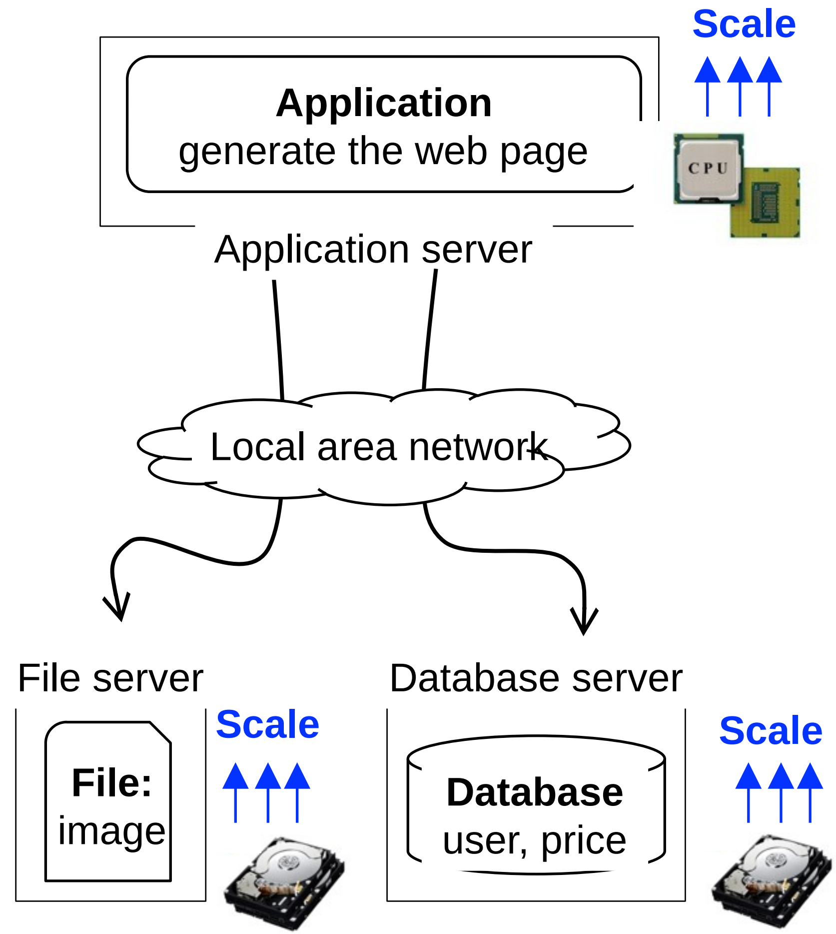
- Can be scaled with more **CPUs**

Database: requires reading/writing disk & cache

- Can be scaled with faster disks & larger memory

File system: store large bulks of data

- E.g., images, videos
- Can be scaled with faster disks



Step #2 Avoid the slow data accesses? Caching

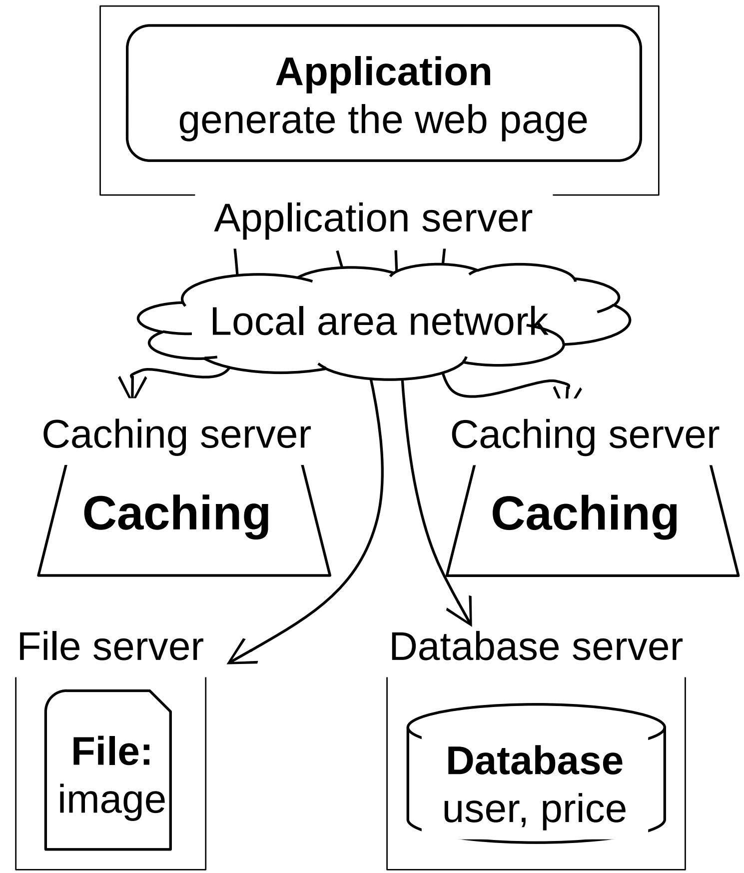
Observation: most requests access **a small portion** of the data (**locality**)

Caching system with a single node:

- E.g., page cache
- Drawbacks: **limited** DRAM capacity

Distributed caching server

- **Benefits**: huge DRAM capacity, e.g., deploy many caching servers



Case study of distributed cache server: Memcached

Distributed caching server

- Benefits: huge DRAM capacity

Memcached server

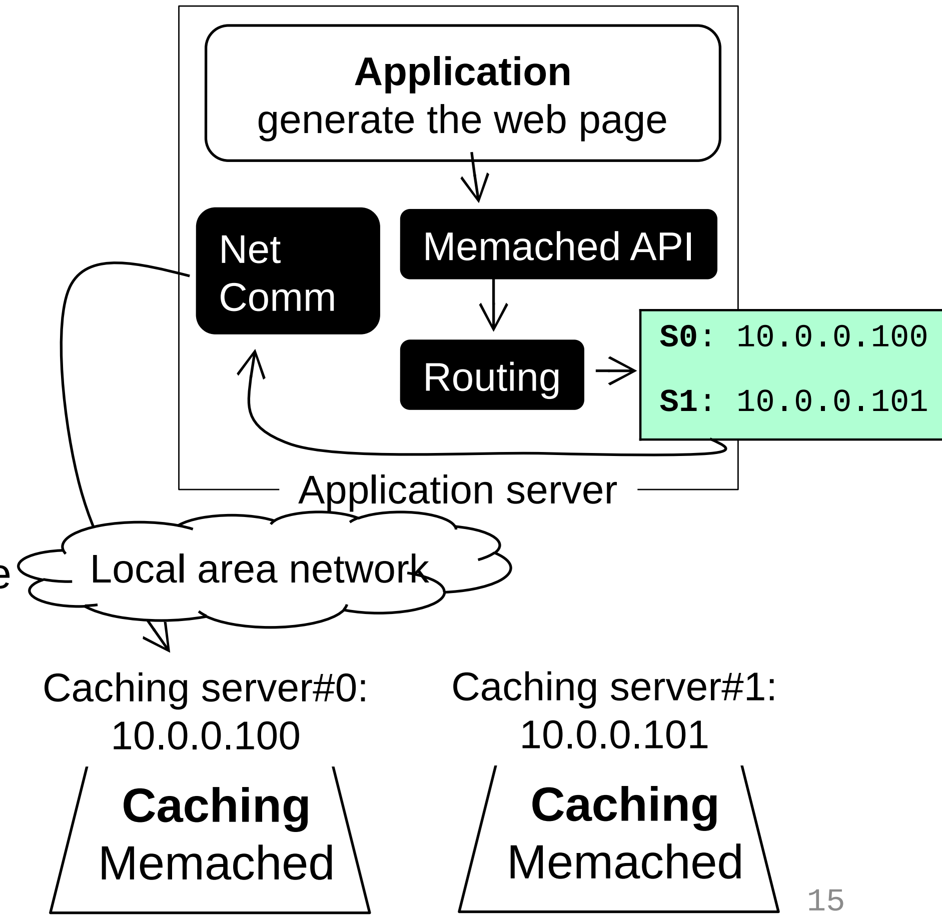
- Store all the cached data in memory
- Can scale out to use multiple servers

Memcached clients

- Check whether server has cached data
- On **cache miss**, **fallback** to database/file

Question:

- How to find which server has cached the data?



Case study of distributed cache server: Memcached

Naïve method, hashing:

– address = key / #server

Problem:

– What if we add a new server?

– Suppose that we add server from 3 → 4

– Then there would be **75% miss**!

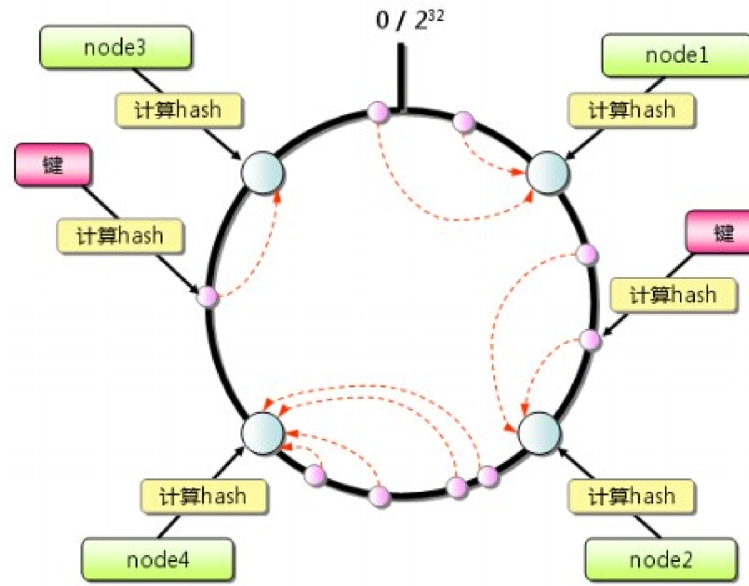
Solution: consistent hashing

– Suppose that we add server from 3 → 4

– It will only incur **25%** miss

– More details in later courses

How to find the data?



Consistent hashing

Step #3 for scalability: more servers

For **stateless** application servers

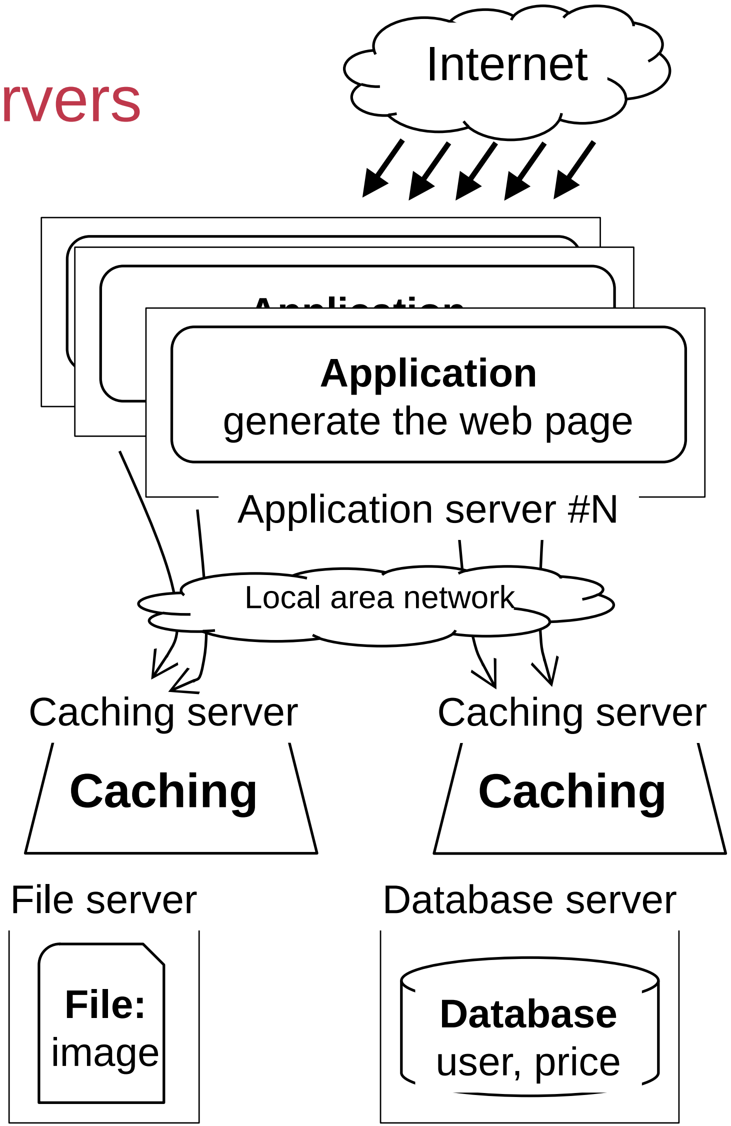
- E.g., web servers
- We can add more servers for scaling its performance

What is stateless?

- The server only executes the logic that only relies on input data but no long-term state

Benefits:

- Better fault-tolerance
- Better elasticity



Step #3 for scalability: How to do the load balance?

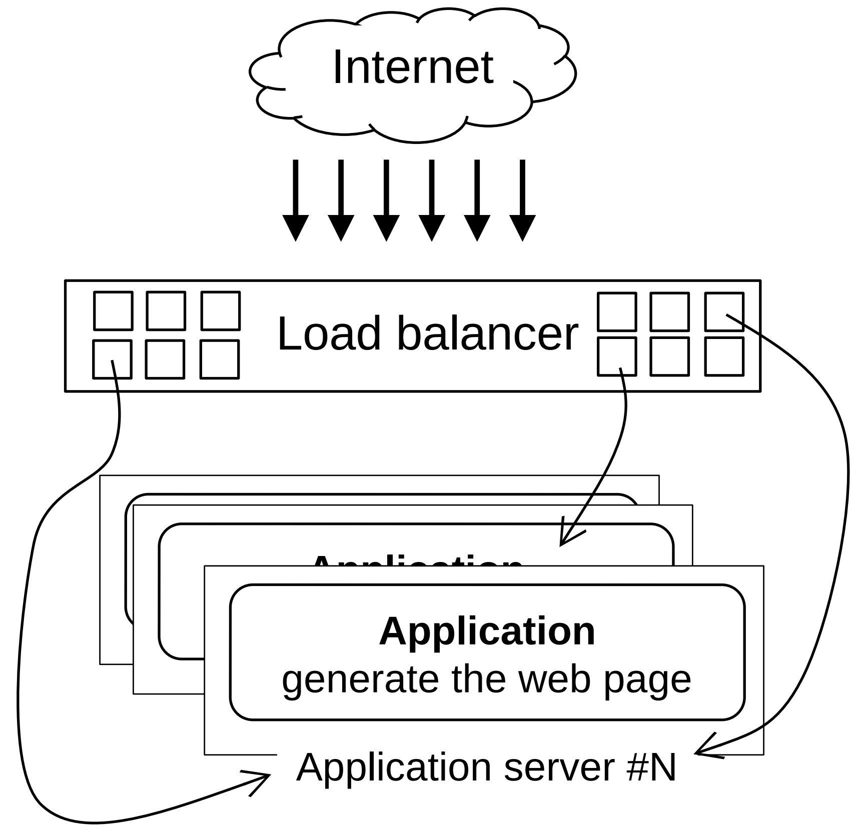
Load balance: how to route the user quests to the application servers?

Leverage the network layer

- HTTP redirection,
- reverse proxy,
- …

Load balance algorithms

- round-robin,
- random,
- hashing,
- …



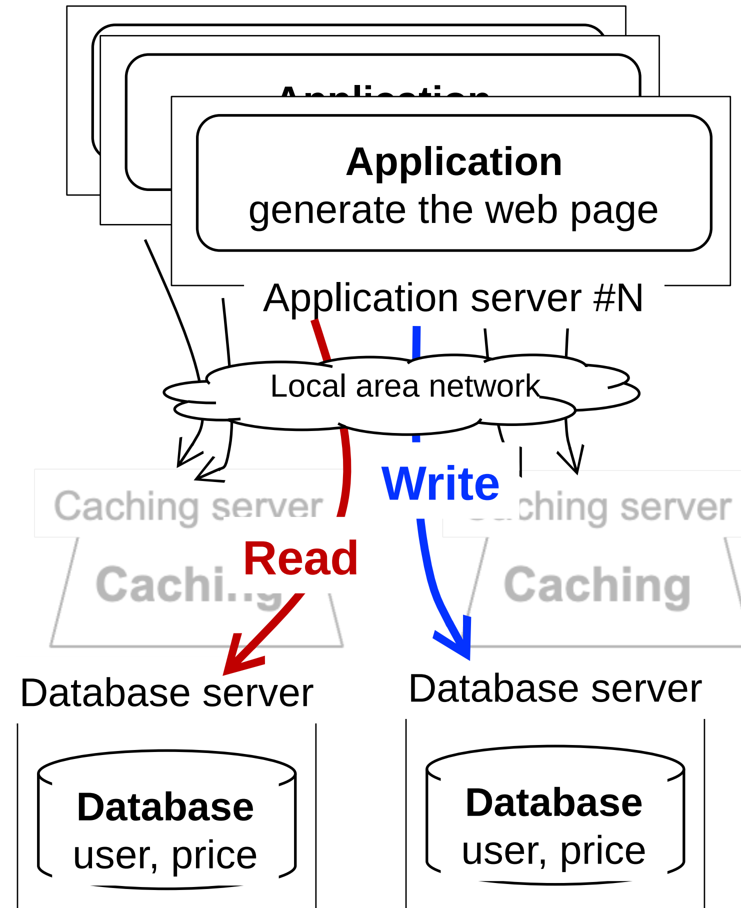
Step #4 for scalability: scaling database

#1. Separate the database for read/write

- E.g., use primary-backup replication
- The primary servers the writes and backup only serves reads

#2. Separate a table on multiple databases

- e.g., a bank has reported that a single table has **100,000,000 rows**
- However, split a table on multiple machines **complex consistency management**



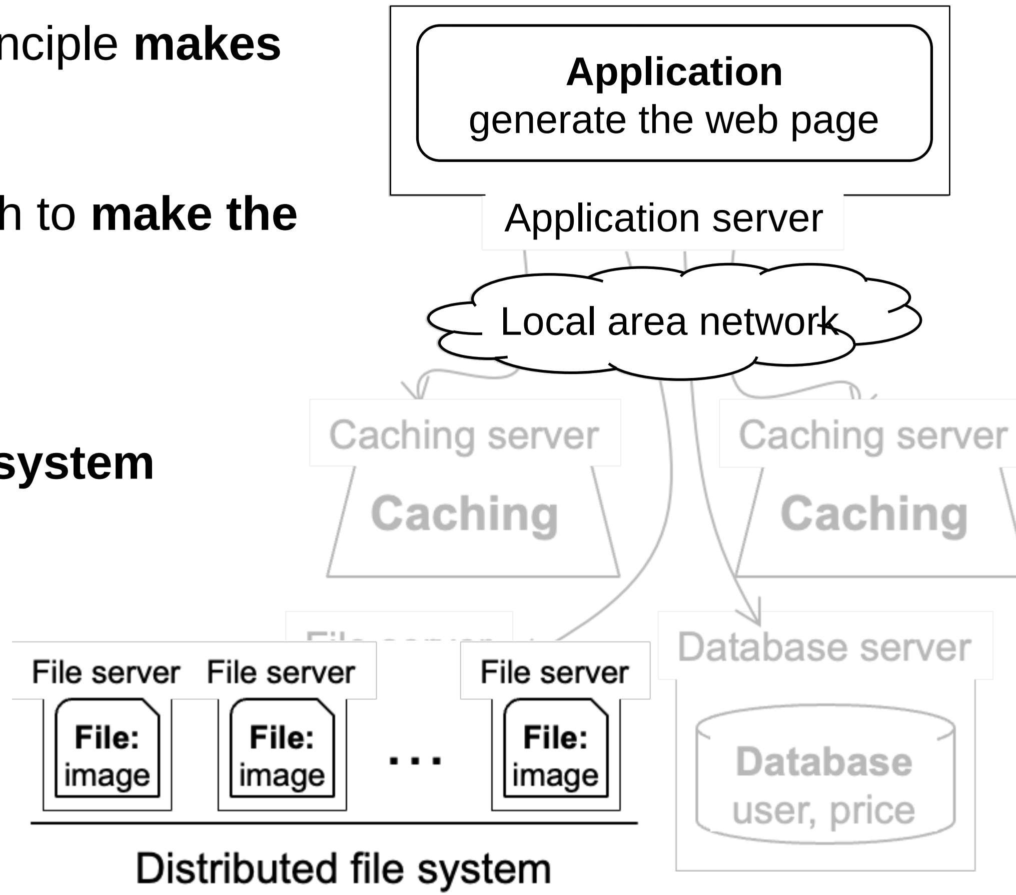
Step #5 for scalability: using distributed file system

Using multiple database in principle **makes the database distributed**

We can use a similar approach to **make the file system distributed !**

Well-known distributed file system

- NFS (Network file system)
- GFS (Google file system)
- HDFS



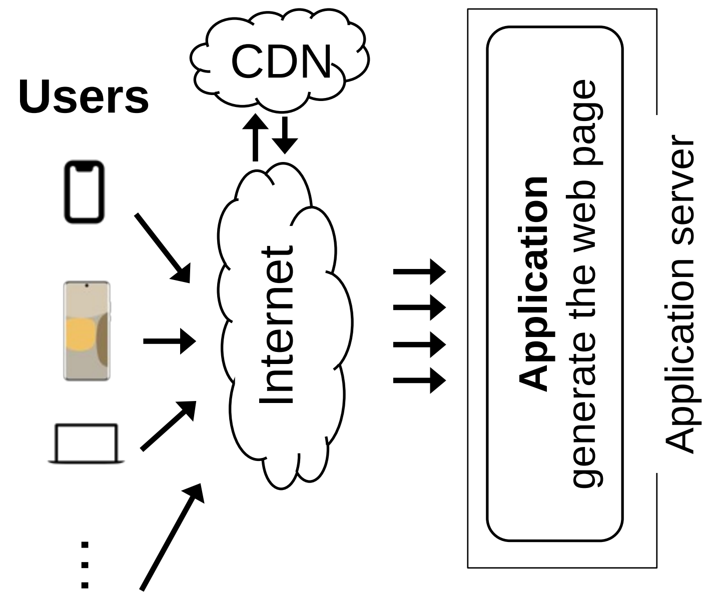
Step #6 for scalability: using CDN

Goal: return the results to user **as soon as possible**

- **Why**? Amazon has reported that **100ms** latency increase will cause **1%** financial **loss**

Core idea: caching (again)

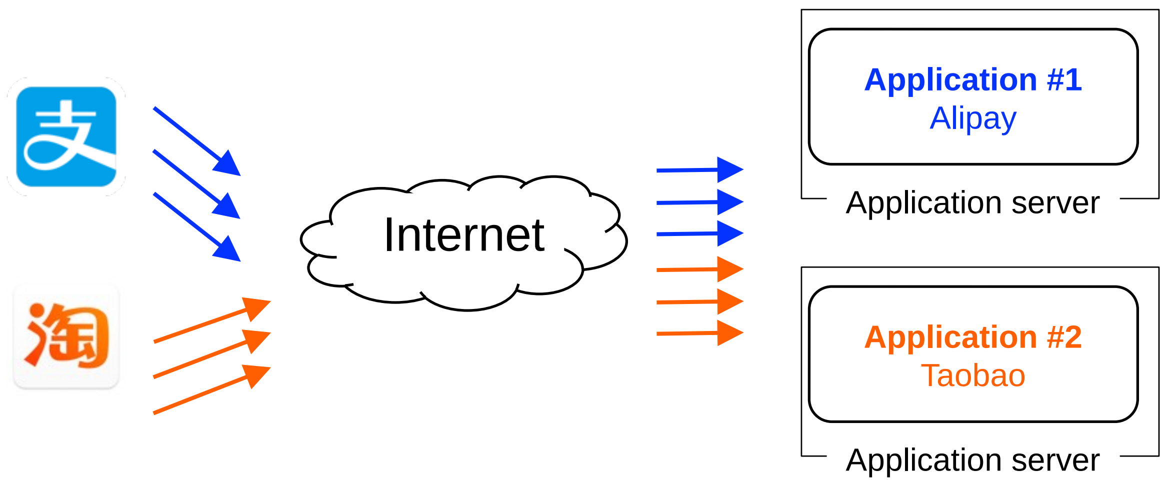
- E.g., **CDN** (content delivery network) caches the content at the network providers, which is closer
  to users
- Challenge: how to do it without users' awareness?



Step #7 for scalability: separate different applications

Use dedicated servers for different applications

– E.g., micro-services

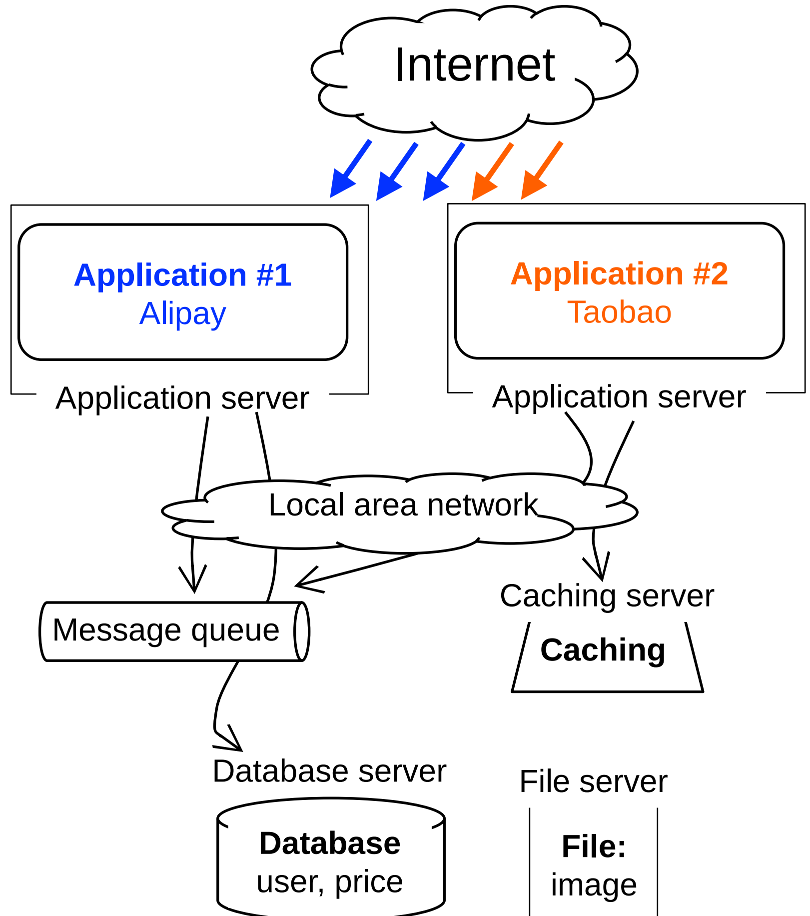


How different applications communicate? MQ or DB

Message queue (MQ): applications send the message the queue, and the queue can buffer the message (somewhere between RPC and database)

- E.g., Apache Kafka

Or, applications can directly use the **databases, caching** (e.g. KV) **or file system to communicate with shared data**



How to handle complex requests?

Example: after the website becoming larger, handling requests is far more than displaying a (static) web page

Alipay (支付宝)

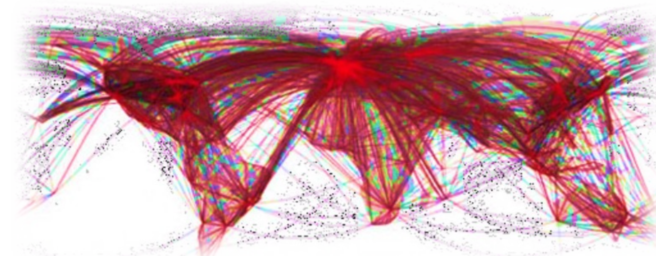
- E.g., fraud detection

Taobao

- Hot list (热榜)
- Dynamic product ads (千人千面)
- Recommendation (you might also like)
- …

Hot lists

Use distributed computing frameworks for complex queries



General batch processing systems

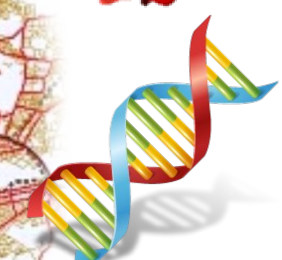
– MapReduce, Spark, etc.

Graph processing systems

– GraphLab, Pregel, etc.

Machine learning systems

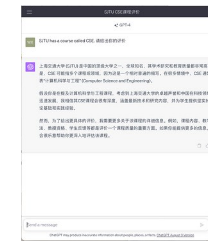
– Tensorflow, pytorch, etc.

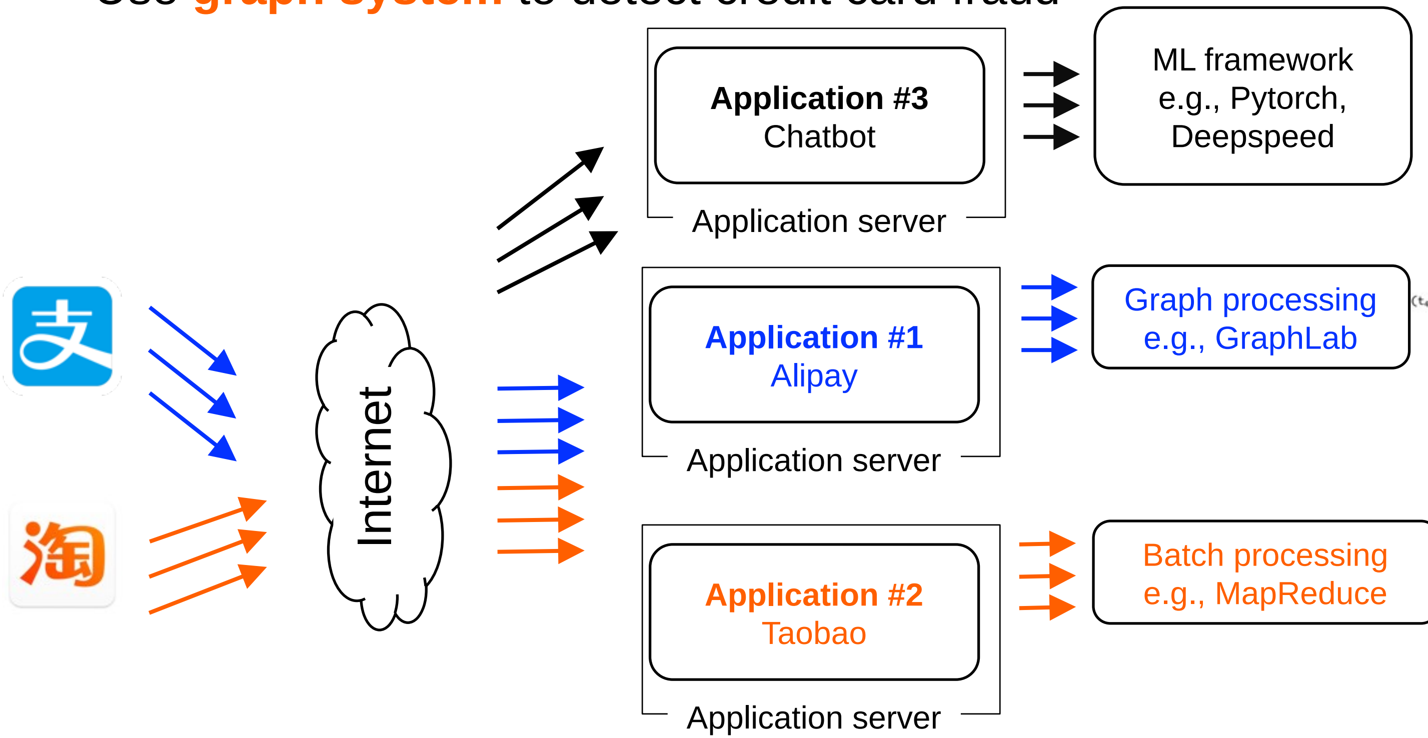


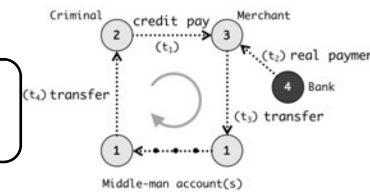
Step #7: separated applications + distributed computing

Example (Each system is backed by multiple machines)

- Use **general batch processing system** to generate hot list
- Use **graph system** to detect credit card fraud





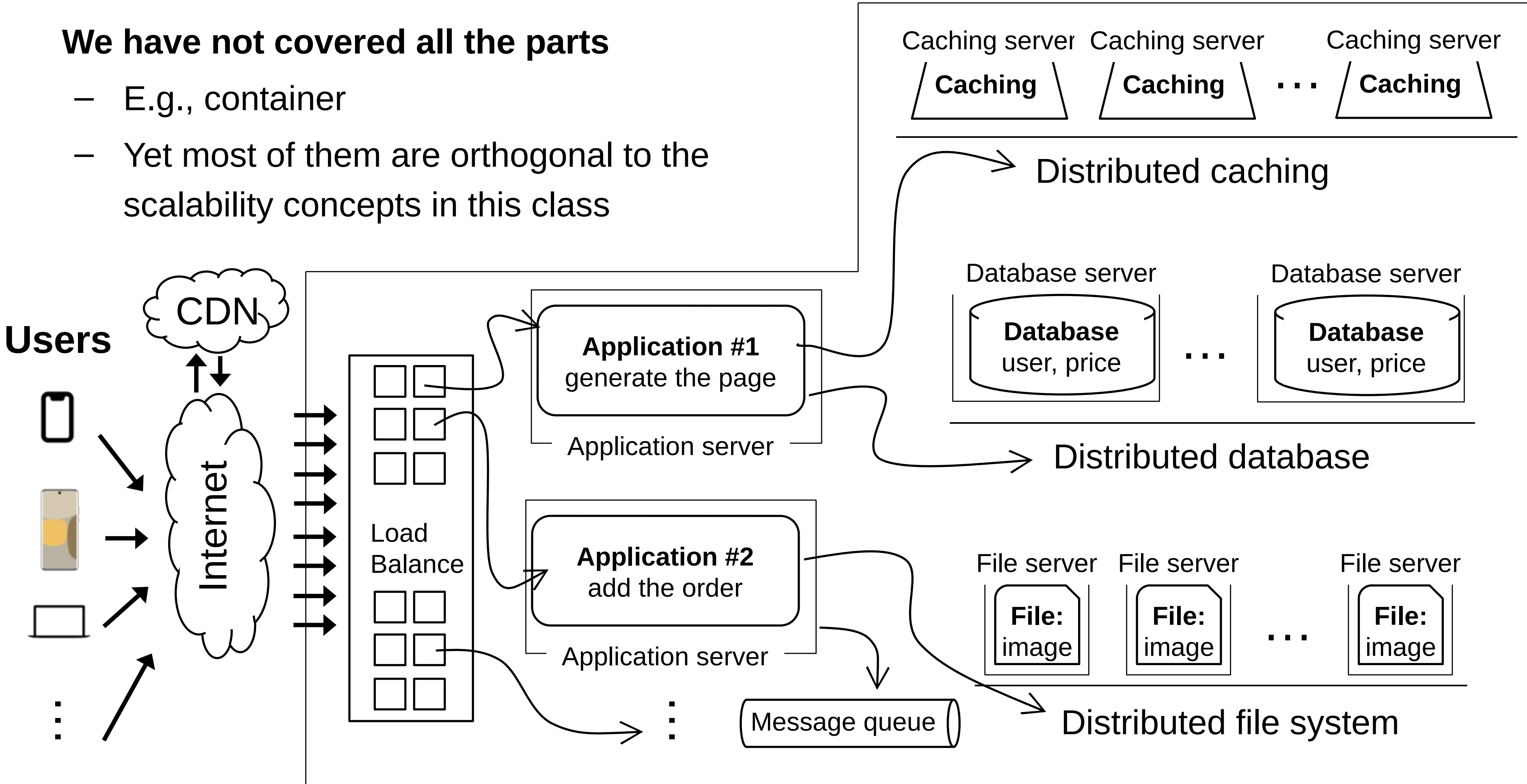


Generate hot lists

A scalable website: overall picture

We have not covered all the parts

- E.g., container
- Yet most of them are orthogonal to the scalability concepts in this class

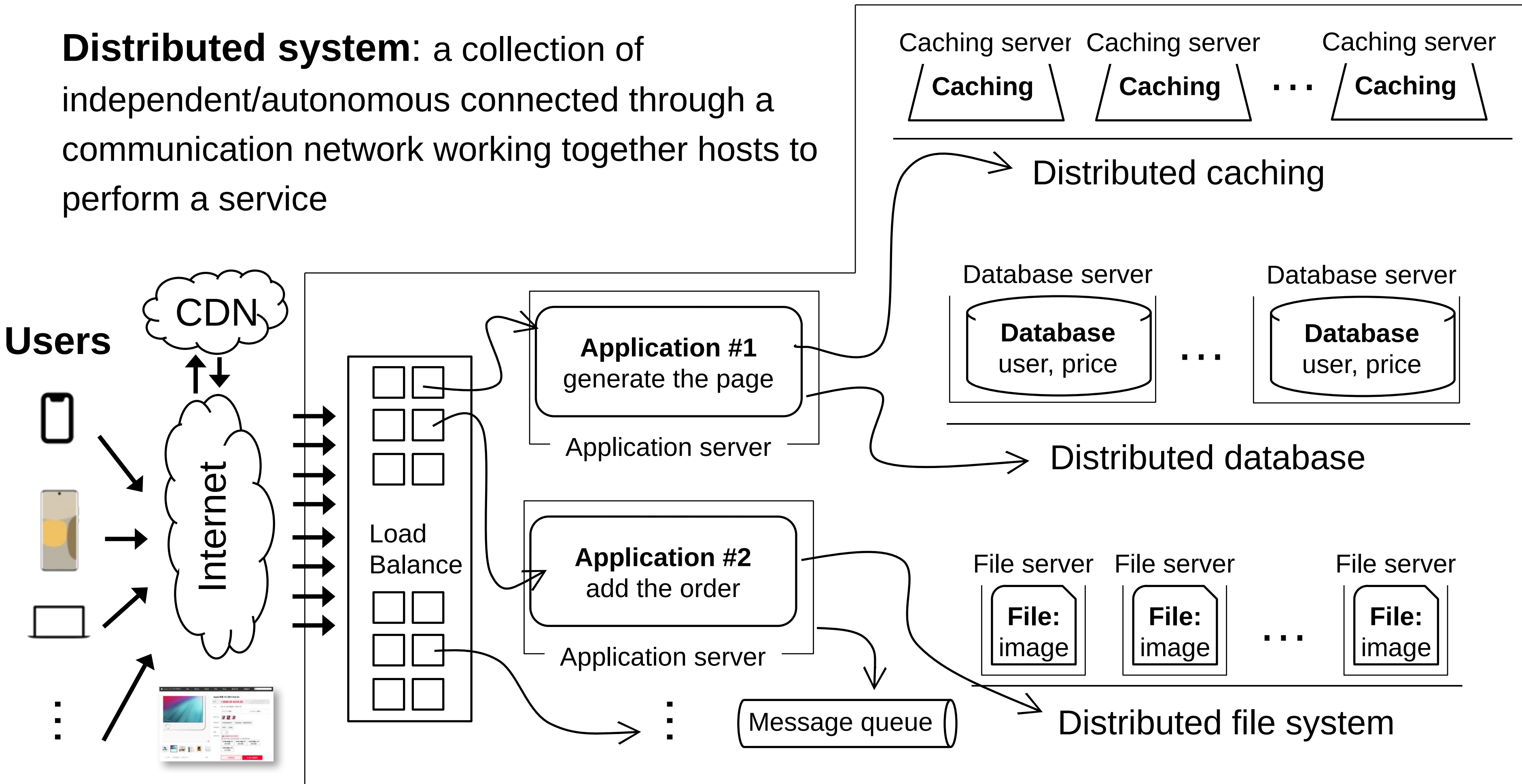


Recall: we have talked about **distributed system**

But what is a distributed system?

A scalable website: powered by distributed systems!

Distributed system: a collection of independent/autonomous connected through a communication network working together hosts to perform a service



Modern scalable system systems are

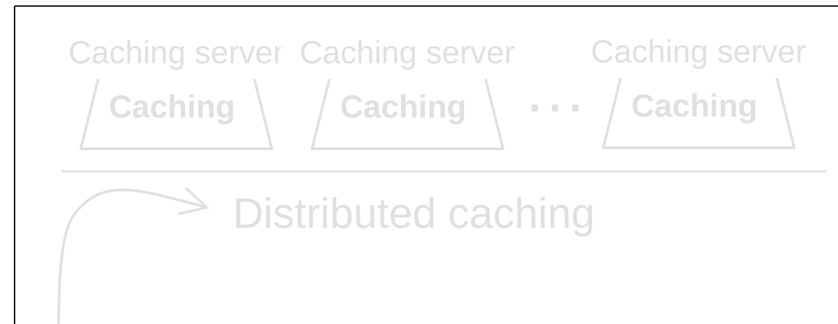
large & distributed

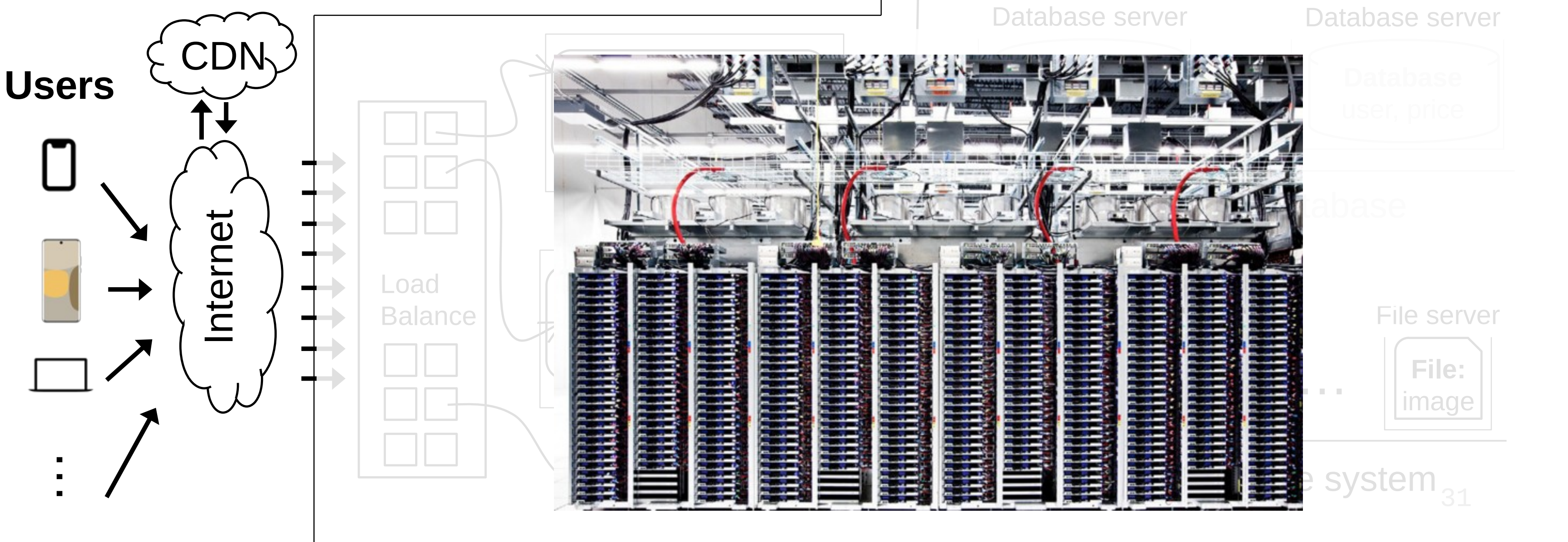
Powered by large-scale **datacenter**

Scalable websites are powered by modern datacenters

Large server and storage farms

- CPUs: > 20,000,000 cores
- Disk capacity: > 1,000,000 GB

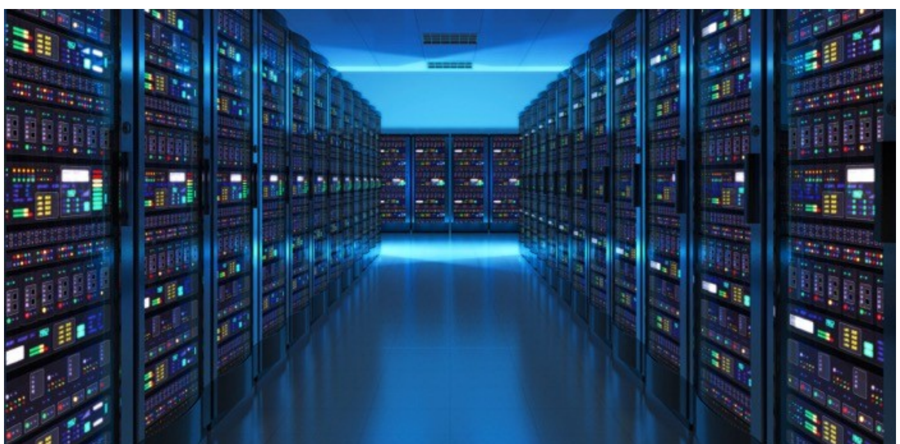


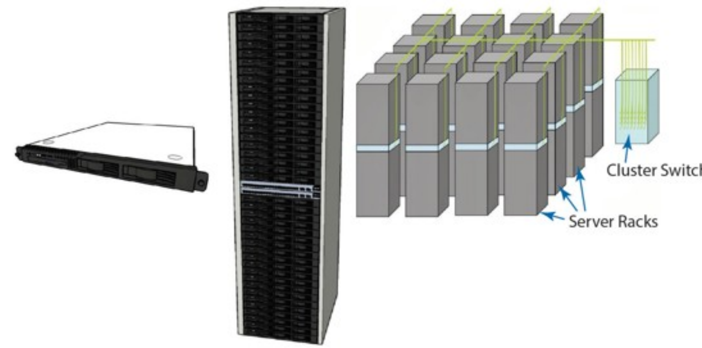


Datacenters that power the scalable website

Large-scale distributed systems: 10K – 100K servers

- Each rack: 40 servers
- Network: 10Gbps – 100Gbps in rack
- 10-100 MW of power

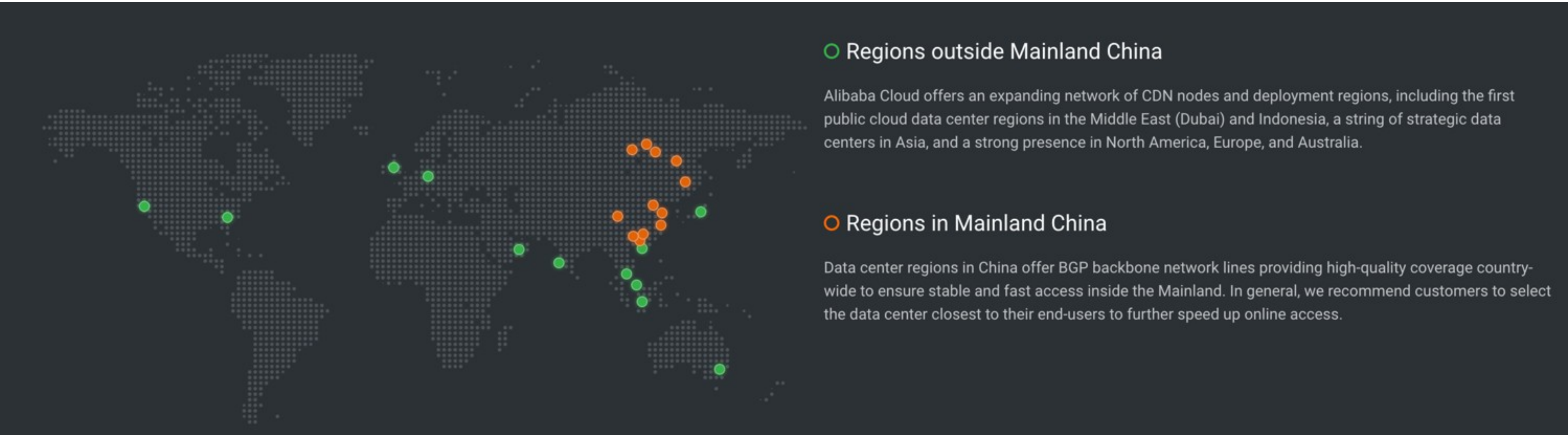




Source: "The Datacenter as a Computer --- An Introduction to the Design of Warehouse-Scale Machines"

Geo-replicated datacenters

The **locations** of Alibaba datacenters



Source: https://www.alibabacloud.com/zh/global-locations

Design large-scale distributed system is challenging

Fault is common, how to handle it?

Fault is common: fault, error, failure

Fault can be latent or active

- if active, get wrong data or control signals

Error is the results of active fault

- e.g. violation of assertion or invariant of spec
- discovery of errors is ad hoc (formal specification?)

Failure happens if an error is not detected and masked

- not producing the intended result at an interface

Fault is common , especially in **large distributed systems**

Why faults are common especially in distributed systems? Scale!

– “Suppose a cluster has ultra-reliable server nodes with a stellar mean time between failures (MTBF) of 30 years (10,000 days)—a cluster of 10,000 servers will see an average of one server failure per day. ”

Source: The Datacenter as a Computer
-- An Introduction to the Design of Warehouse-Scale Machines

Fault is common

especially in **large distributed systems**

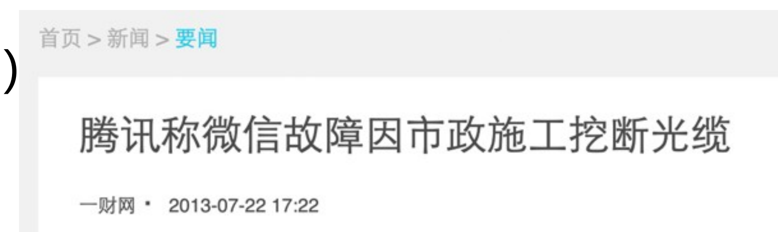
What are the causes?

Why faults are common especially in distributed systems? Scale!

- "Suppose a cluster has ultra-reliable server nodes with a stellar mean time between failures (MTBF) of 30 years (10,000 days)—a cluster of 10,000 servers will see an average of one server failure per day. "

Causes:

- Operation error (human, configuration, etc.)
- Software error (e.g., bug)
- Hardware error
- Power outage
- Natural disaster



Source: The Datacenter as a Computer
-- An Introduction to the Design of Warehouse-Scale Machines

Fault is common

especially in **large distributed systems**

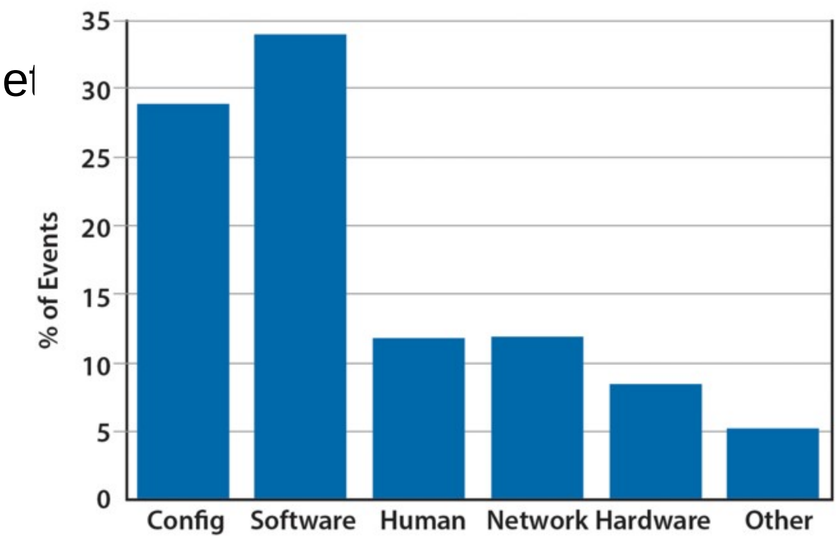
What are the causes?

Why faults are common especially in distributed systems? Scale!

- "Suppose a cluster has ultra-reliable server nodes with a stellar mean time between failures (MTBF) of 30 years (10,000 days)—a cluster of 10,000 servers will see an average of one server failure per day. "

Causes:

- Operation error (human, configuration, et
- Software error (e.g., bug)
- Hardware error
- Power outage
- Natural disaster



Source: The Datacenter as a Computer
-- An Introduction to the Design of Warehouse-Scale Machines

Faults and partial failures

On a single computer, **it either works or it doesn't**

But in a **distributed system**, **some parts of the system can be broken in some unpredictable way**

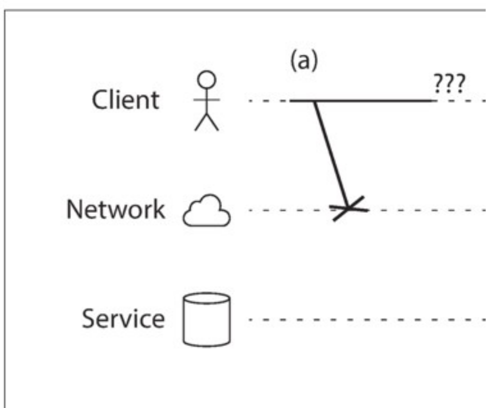
- Such failure is **partial** (aka., grey failure)

Since **most** other parts of the system are **OK**, we want the system **still working**!

Faults and partial failures

Example: unreliable network

- A client (e.g., Smartphone),
  sends requests to the service (e.g., Taobao),
  through network (5G, Wifi)
- The client gets: “ 网络竟然奔溃了” ，how can a network break?



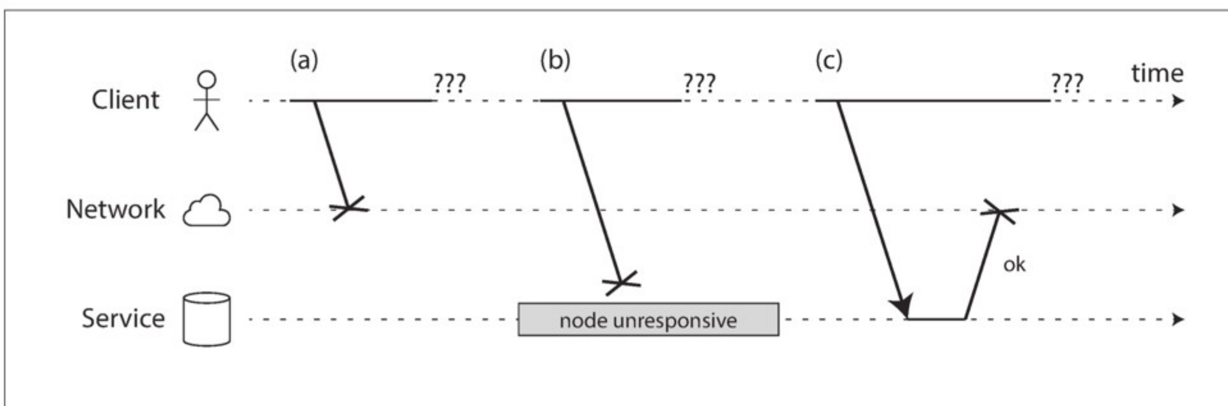
Example: **unreliable network**

A user sends a request but **the server does not reply**, possible reasons:

1. The request may have been **lost** (e.g., someone unplugged a network cable).

2. The request may be waiting in a queue and will be delivered **later** (e.g., the network or the recipient is overloaded).

3. The remote node may have **failed** (e.g., it crashed or it was powered down).



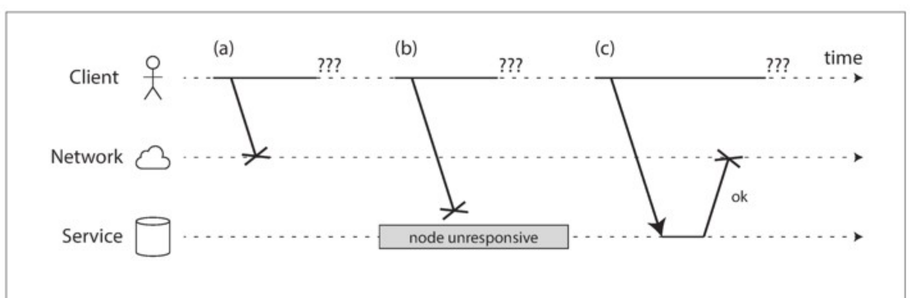
Example: **unreliable network**

A user sends a request but **the server does not reply**, possible reasons:

4. The remote node may **have temporarily stopped** responding (e.g., it is experiencing a long garbage collection pause)

5. The remote node may have processed your request, but the **response** has been **lost** on the network (e.g., a network switch has been misconfigured).

6. The remote node may have processed your request, but the response has been **delayed** and will be delivered later (e.g., the network or your own machine is overloaded).



Another common fault: network partition

A **network partition** refers to network decomposition into relatively independent subnets

- Can happen when a switch is being upgraded in a datacenter
- Can even happen when being attacked by sharks

Network partition is usually a reality

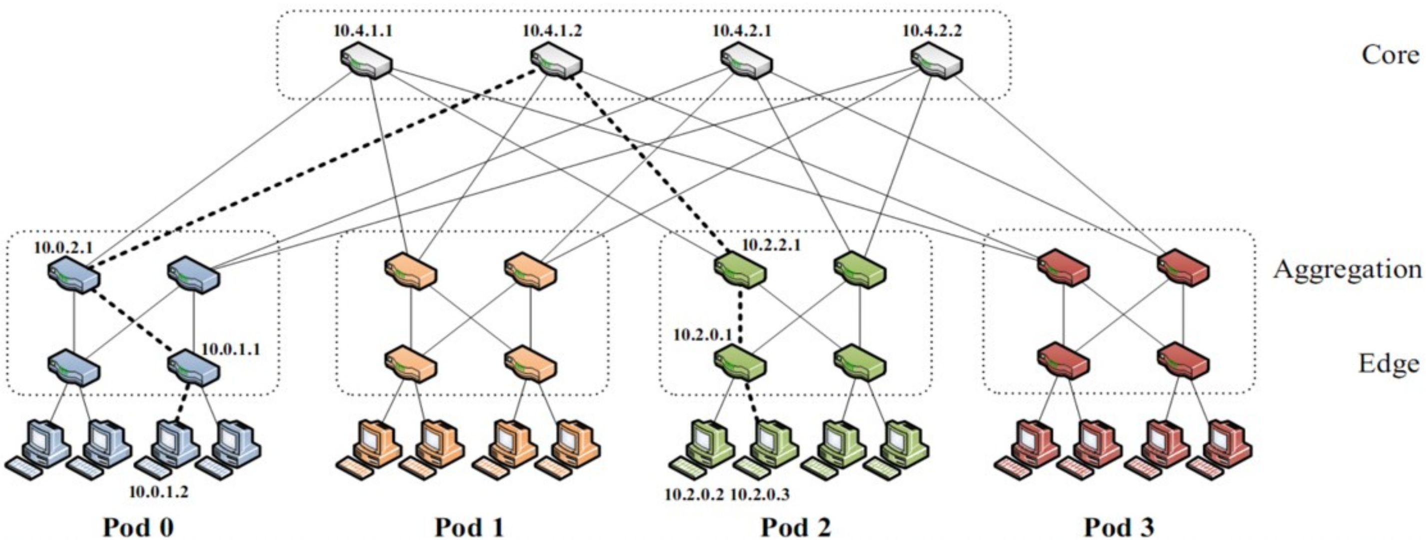
- You can never count on the network connectivity

Another common fault: network partition

A **network partition** refers to network decomposition into relatively independent subnets

- Can happen when a switch is being upgraded in a datacenter

For example, suppose the datacenter adopts a **fat-tree** topology



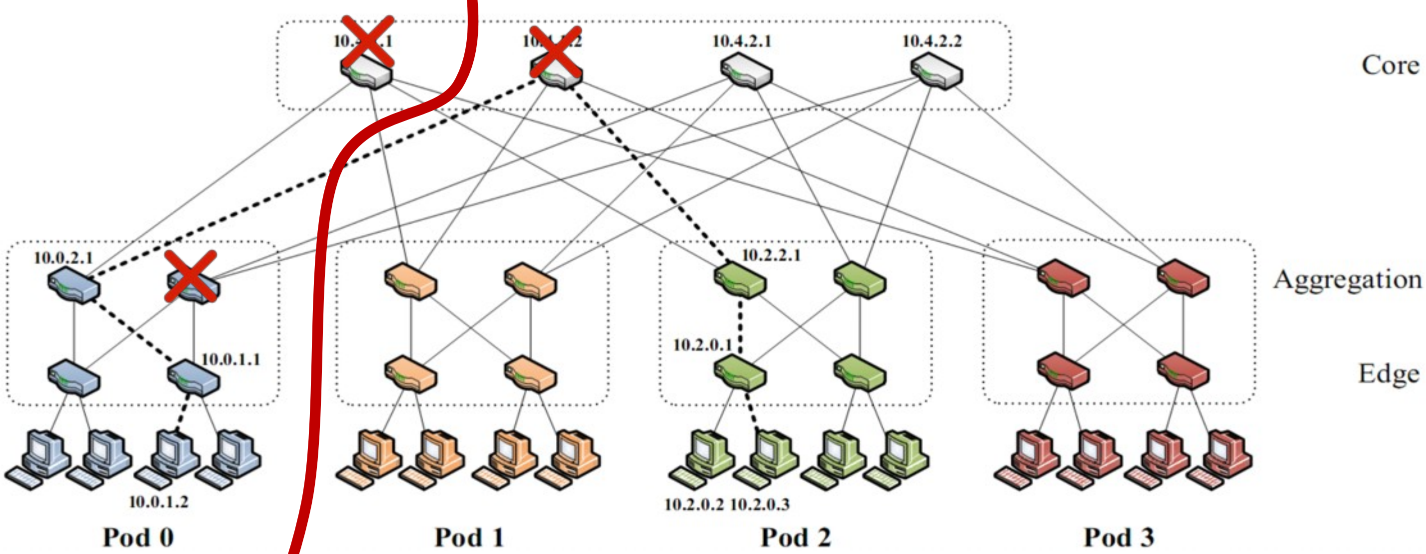
Source: https://www.cs.cornell.edu/courses/cs5413/2017sp/lectures/16-datacenter-topologies2.pdf

Another common fault: network partition

A **network partition** refers to network decomposition into relatively independent subnets

– Can happen when a switch is being upgraded in a datacenter

For example, suppose the datacenter adopts a **fat-tree** topology



Source: https://www.cs.cornell.edu/courses/cs5413/2017sp/lectures/16-datacenter-topologies2.pdf

Another common fault: network partition

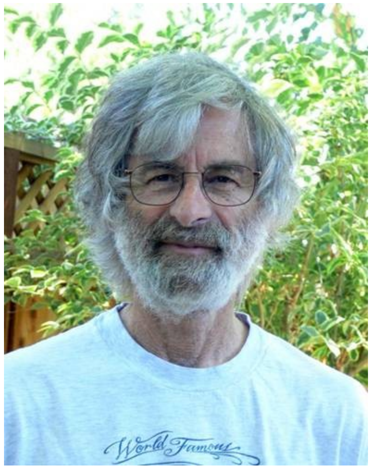
A **network partition** refers to network decomposition into relatively independent [subnets]

- Can happen when a switch is being upgraded in a datacenter
- Can even happen when being **attacked by sharks**

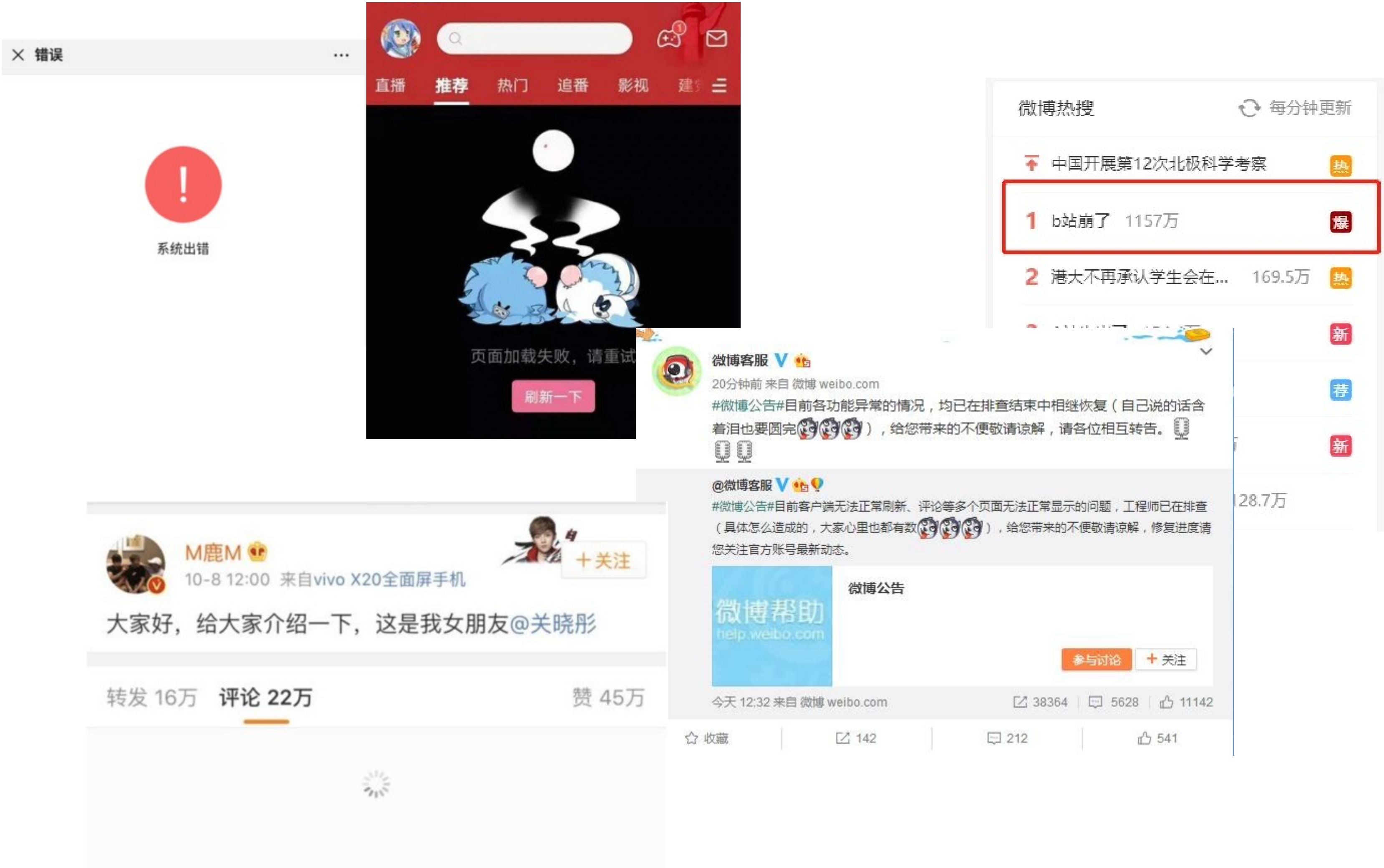
Source: https://slate.com/technology/2014/08/shark-attacks-threaten-google-s-undersea-internet-cables-video.html

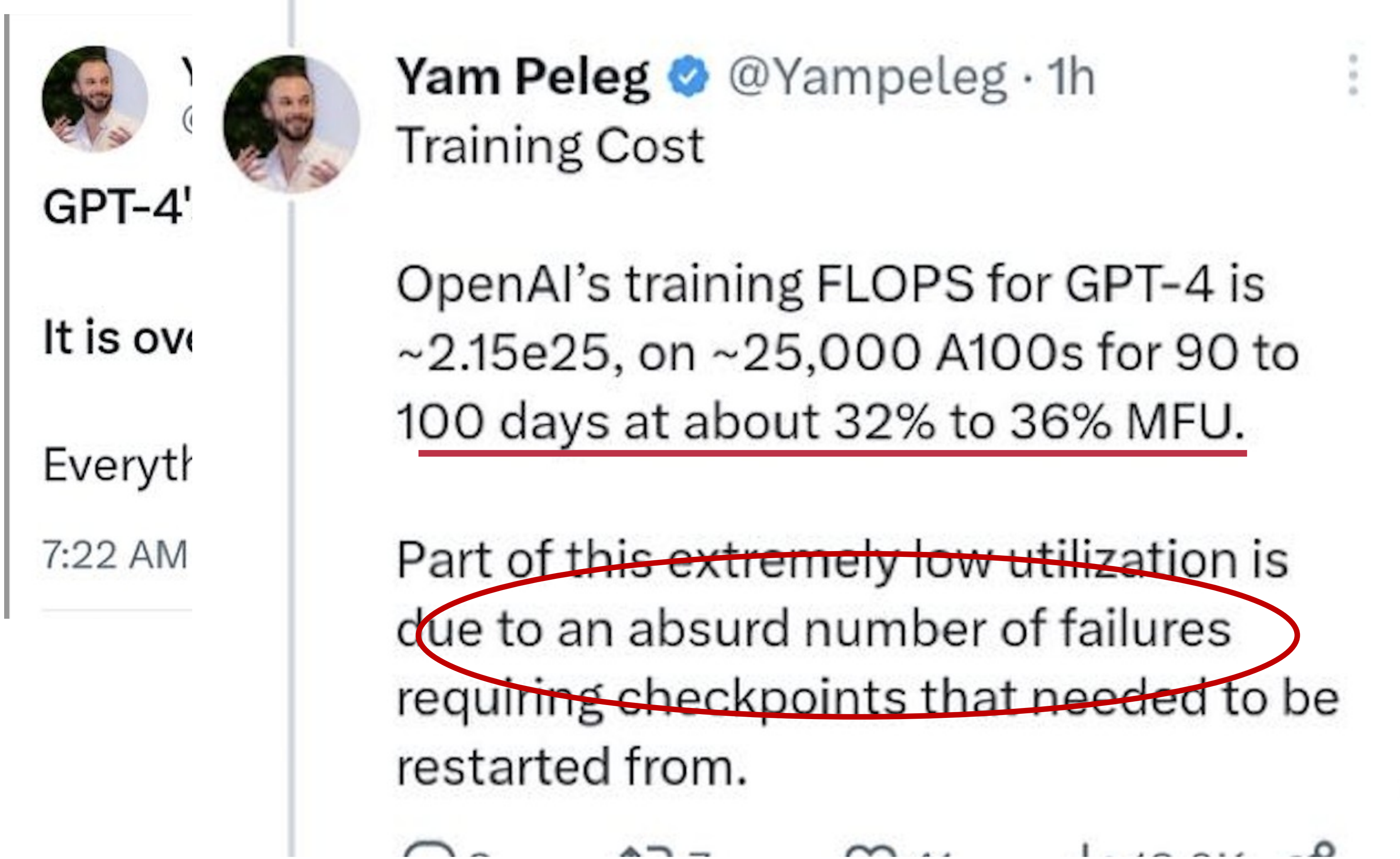
You know you have a distributed system when the crash of a computer you've never heard of stops you from getting any work done.

- Leslie Lamport



Lamport received the 2013 Turing Award for "fundamental contributions to the theory and practice of distributed and concurrent systems, notably the invention of concepts such as causality and logical clocks, safety and liveness, replicated state machines, and sequential consistency" in 2014.


错误
系统出错
直播 推荐 热门 追番 影视
页面加载失败，请重试
刷新一下
微博热搜 每分钟更新
中国开展第12次北极科学考察 热
1 b站崩了 1157万 爆
2 港大不再承认学生会在... 169.5万 热
新
荐
新
28.7万
微博客服
20分钟前 来自 微博 weibo.com
#微博公告#目前各功能异常的情况，均已在排查结束中相继恢复（自己说的话含着泪也要圆完），给您带来的不便敬请谅解，请各位相互转告。
@微博客服
#微博公告#目前客户端无法正常刷新、评论等多个页面无法正常显示的问题，工程师已在排查（具体怎么造成的，大家心里也都有数），给您带来的不便敬请谅解，修复进度请您关注官方账号最新动态。
微博帮助
help.weibo.com
微博公告
参与讨论
+ 关注
今天 12:32 来自 微博 weibo.com 38364 5628 11142
收藏 142 212 541
M鹿M
10-8 12:00 来自vivo X20全面屏手机
+ 关注
大家好，给大家介绍一下，这是我女朋友@关晓彤
转发 16万 评论 22万 赞 45万


GPT-4'
It is ov
Everyth
7:22 AM
Yam Peleg @Yampeleg · 1h
Training Cost
OpenAI's training FLOPS for GPT-4 is ~2.15e25, on ~25,000 A100s for 90 to 100 days at about 32% to 36% MFU.
Part of this extremely low utilization is due to an absurd number of failures requiring checkpoints that needed to be restarted from.

Availability and reliability

Availability: A measure of the time that a system was usable, as a fraction of the time that it was intended to be usable (x nines), corresponding downtime:

- e.g. 3-nines -> 8 hour/year
- e.g. 5-nines -> 5 min/year
- e.g. 7-nines -> 3 sec/year

Metrics to measure reliability

- MTTF: mean time to failure
- MTTR: mean time to repair
- MTBF: mean time between failure
- **MTBF = MTTF + MTTR**

$$MTTF = \frac{1}{N}\sum_{i=1}^{N} TTF_i$$

$$MTTR = \frac{1}{N}\sum_{i=1}^{N} TTR_i$$

Large-scale systems can be highly available

Example#1: Internet, the BGP (Border Gateway Protocol) is highly available

Example#2: WeChat, Taobao & Baidu rarely (but not never) become outage

Example #3: OpenAI successfully trained GPT despite many failures

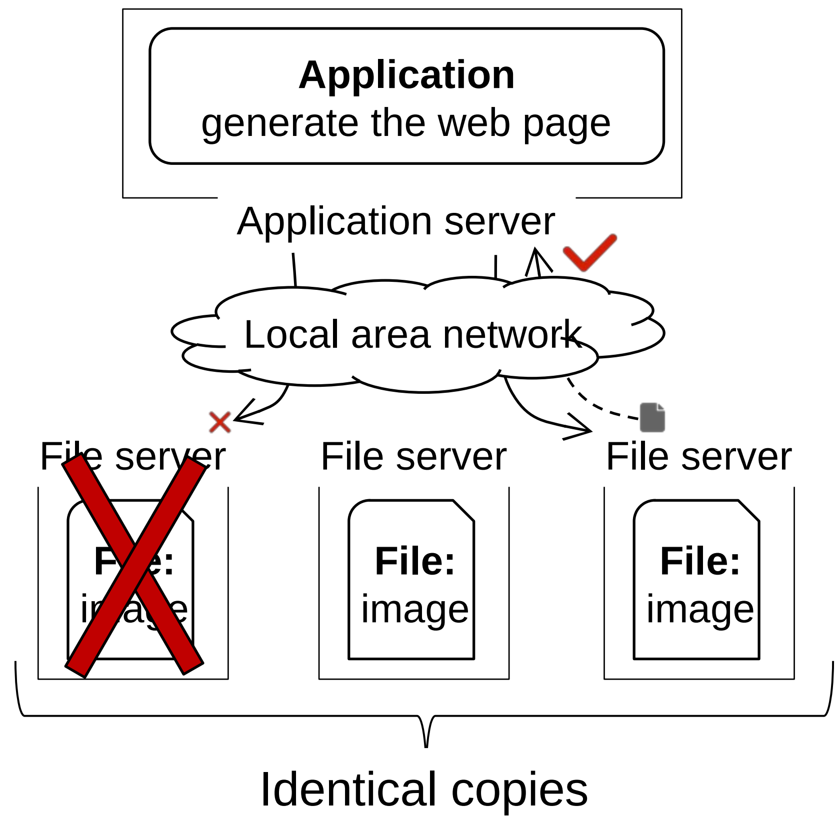
Achieving high availability: handling failures w/ replications

Replication

- replicas: identical multiple copies

Example: **replicated file servers**

- If one copy survives, the application is available



Challenge: consistency

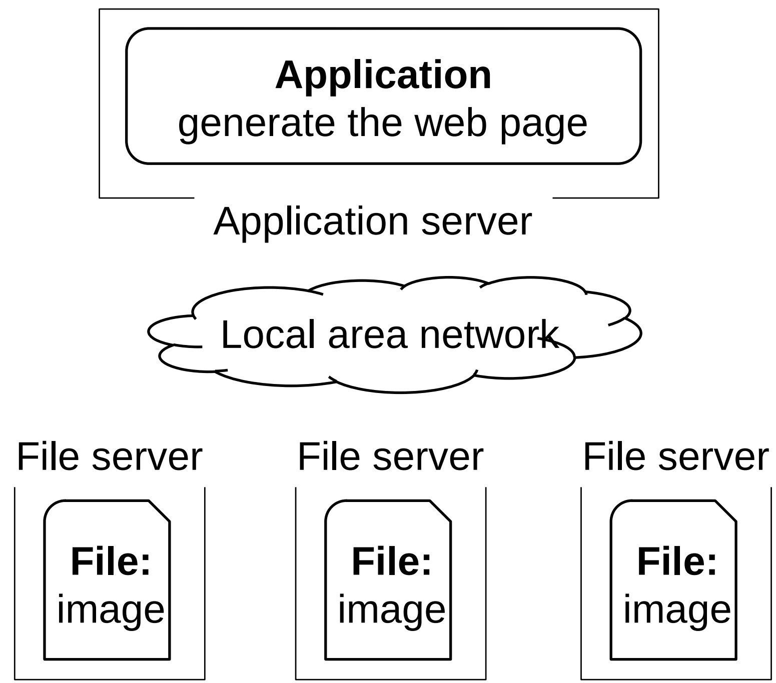
Replication

- replicas: identical multiple copies

Example: **replicated file servers**

- If one copy survives, the application is available

Challenge: consistency



Challenge: consistency

Replication

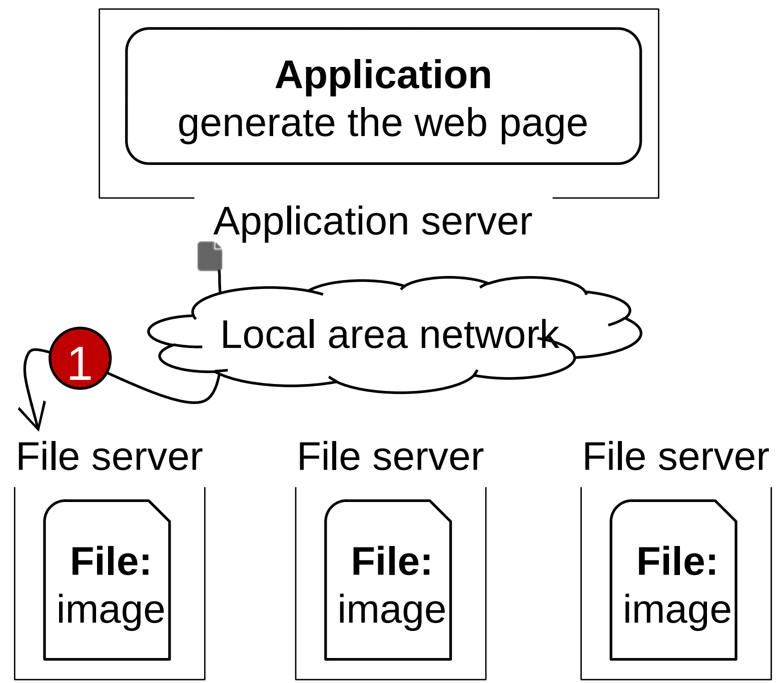
- replicas: identical multiple copies

Example: **replicated file servers**

- If one copy survives, the application is available

Challenge: consistency

1. Application put file A to one server



Challenge: consistency

Replication

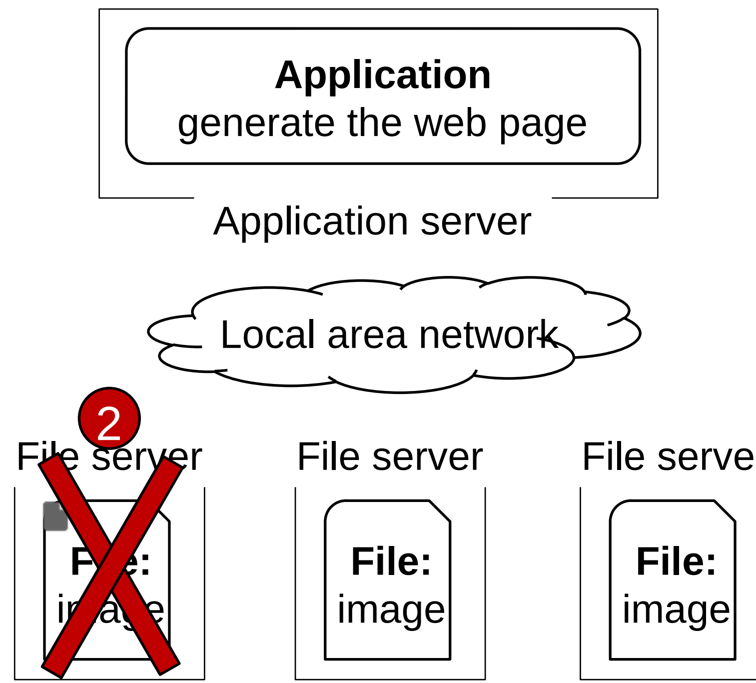
- replicas: identical multiple copies

Example: **replicated file servers**

- If one copy survives, the application is available

Challenge: consistency

1. Application put file A to one server
2. The server crashed



Challenge: consistency

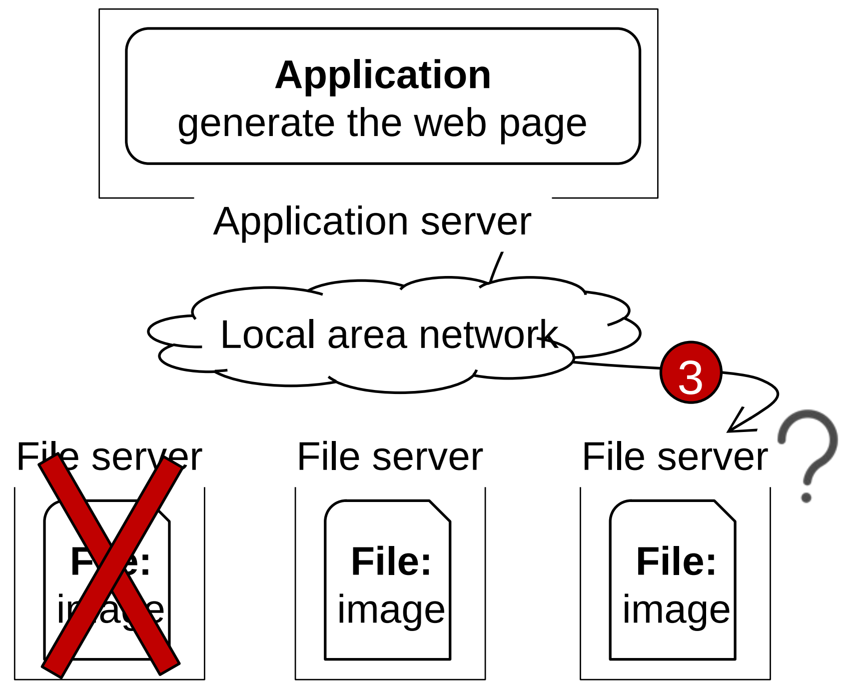
Replication

- replicas: identical multiple copies

Example: **replicated file servers**

- If one copy survives, the application is available

1. Application put file A to one server
2. The server crashed
3. What happens when read A again?



Achieving high availability: handling failures

Replication

- replicas: identical multiple copies

Techniques to cope with consistency:

- Primary-backup replication
- Replicated state machine
- …

Achieving high availability: handling failures via retry

Restart or **reconstruct**

- Monitoring and catching errors
- Restart or reconstruct the system (sub-system)
- E.g., restart the stateless application server is ok

What about consistency? Must made trade-off

- Stateless applications does not have consistency issue
- Some applications, like **Google search**, can even tolerate occasional inconsistency
    - Can you notice the inconsistency of search results?

The CAP theorem

- **C**onsistency, **A**vailability & **P**artition tolerance

CAP: an example

Amazon has two zones: US and Euro

- All US users connect to the US zone
- All Euro users connect to the Euro zone

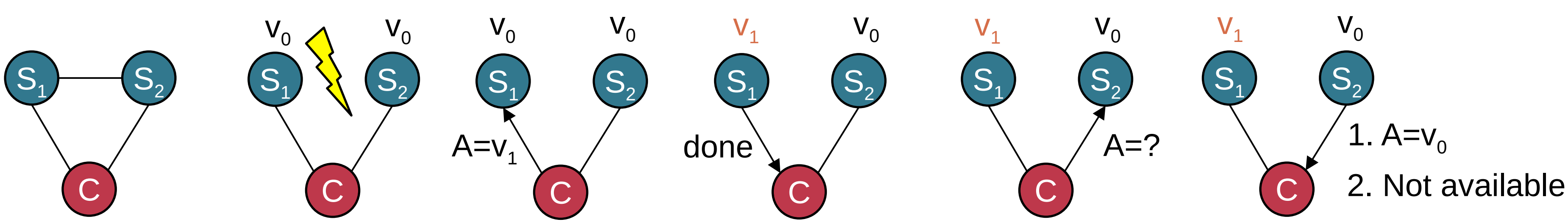
Define Availability and Consistency

- **C**: all users see the actual number of items
- **Not-C**: users see 1 item left, but actually is 0
- **A**: all users can buy items at any time (if there are some)
- **Not-A**: users might get "cannot buy now, please wait and retry"

The CAP theorem: 2 out of 3

It is impossible for a distributed computer system to simultaneously provide all three of the following guarantees

- **Consistency** (all nodes see the same data at the same time)
- **Availability** (a guarantee that every request receives a response about whether it succeeded or failed)
- **Partition tolerance** (the system continues to operate despite arbitrary message loss or failure of part of the system)



Partition Tolerance

"P" is usually a reality

– You can never count on the network connectivity

AP: sacrifice C

– If you have one book but sell it to two customers

– Maybe just an apology and a small gift coupon

CP: sacrifice A

– If you are selling train ticket but cannot deliver

– Customer may sue you

– But the user can fail to buy the ticket, e.g., 12306 during spring festival in the last few years

CAP: not a Binary Decision

Example: a CP system does not mean no "A" at all during network partition

- If the Euro zone and US zone are disconnected, the US zone can still keep available
- Only the Euro zone is not available
- When the network connects again, apply US zone data to Euro zone for consistency

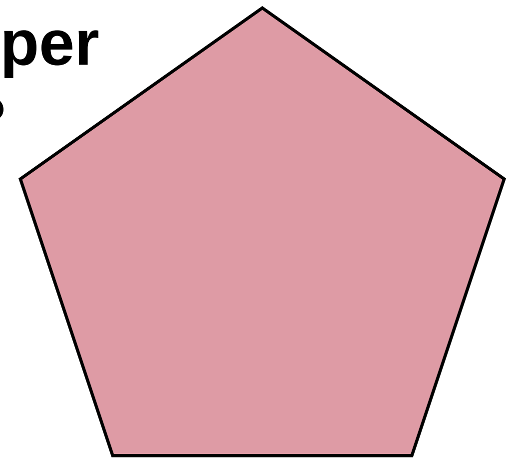
Summary of the (ideal) properties of distributed systems

Can our system handle a larger workload?

Scalability

How easy the developer can use our system?

Ease of programming



How fast is our system?

Performance

Consistency

Is our data/computation correct?

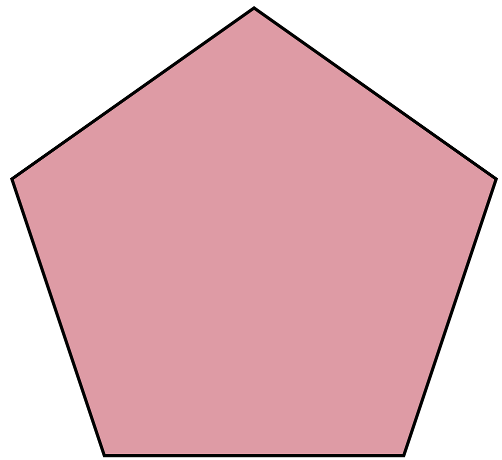
Fault tolerance

What if some component crash?

Summary of the (ideal) properties of distributed systems

Sharding Replication Load balancing New hardware

Scalability



Ease of programming

DSM, RPC, MapReduce, …

Performance

Caching

Consistency

Sequential, eventual, consensus, isolation, etc.

Fault tolerance

High availability: replication & checkpointing

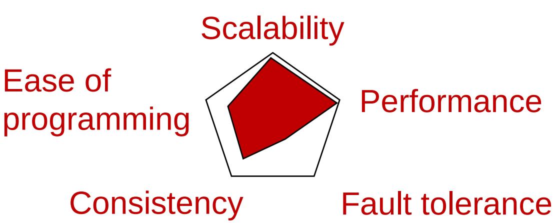
Durability: atomicity, logging, etc.

Summary of the (ideal) properties of distributed systems

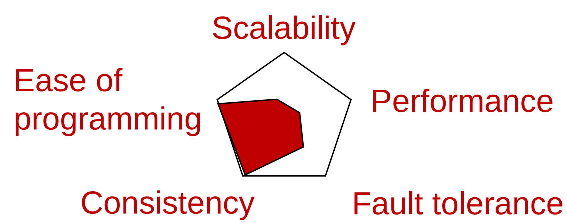
These properties typically cannot achieve at the same time

– The adults want them all, but the reality forces them to make trade-offs

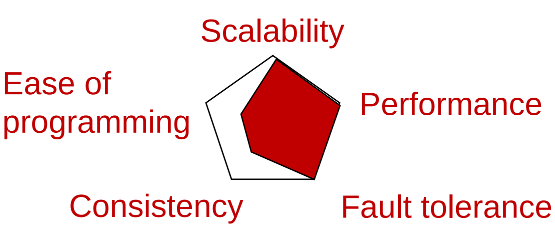
Remote Procedure Call (RPC)



Distributed Shared Memory (DSM)



NoSQL databases



NewSQL databases (e.g., Spanner)