

并行与分布式处理

直播/上课时间：周五6-8节课（12:55）

ZOOM直播

- 房间号：993562036
- 密码：02568320

微信群二维码



该二维码7天内(2月29日前)有效，重新进入将更新

温馨提示：

- 进入Zoom课堂后请将昵称改为**真实姓名**
- 提问发言请先“举手”
- 如遇系统卡顿，主动反馈并耐心等待
- 可在Zoom聊天室或微信群内留下与教学有关的问题或建议
- **请尽快熟悉Zoom、Canvas的操作方式**
- 不迟到、不早退，按时完成作业
- **尊重并保护知识产权，不随意传播视频等课程资料**

Parallel and Distributed Processing

Rong Chen

Institute of Parallel and Distributed Systems

Shanghai Jiao Tong University

http://ipads.se.sjtu.edu.cn/rong_chen

Outline

Course Information

- Faculty Info
- Why take this course?
- Course goal and methodology
- Grading

Intro to parallel/distributed systems

Faculty Information



陈榕 (CHEN Rong)

Email: rongchen@sjtu.edu.cn

Office phone: 13616861826

Assistant: **魏星达** (WEI XINGDA)

Email: wxdwfc@gmail.com

Course site: <http://ipads.se.sjtu.edu.cn/courses/ads>

CANVAS: <https://oc.sjtu.edu.cn/courses/20267>

Why take this Course ?

Huge amounts of computing are now **parallel** or **distributed** ...

- Intel threw its hands up in the air:

Couldn't increase GHz much more without CPU temperatures reaching solar levels

- But we can still stuff more transistors (Moore's Law)
- Results: **Multicore & NUMA**

Your computer has become a **parallel/distributed** system

Why take this Course ?

Whole Internet thing...

- Most things are distributed, and more each day

My **phone** syncs its calendar with iCloud, which I can get on my **mac air** with an **app** or **web browser**, ...

- *Internet* and *mobile* have made area much more attractive and timely
- **Hard/unsolved**: not many deployed sophisticated parallel and distributed systems

Course Goal

Introduce you to (parallel & distributed) system design concepts and principles

- Modularity, Layering, Reliability, ...

Cover the design and evolutions of many important (parallel & distributed) computer systems

- Industry and Academic

Course Goal

Introduce you to (parallel & distributed) system design concepts and principles

- Modularity, Layering, Reliability, ...

Cover the design and evolutions of many important (parallel & distributed) computer systems

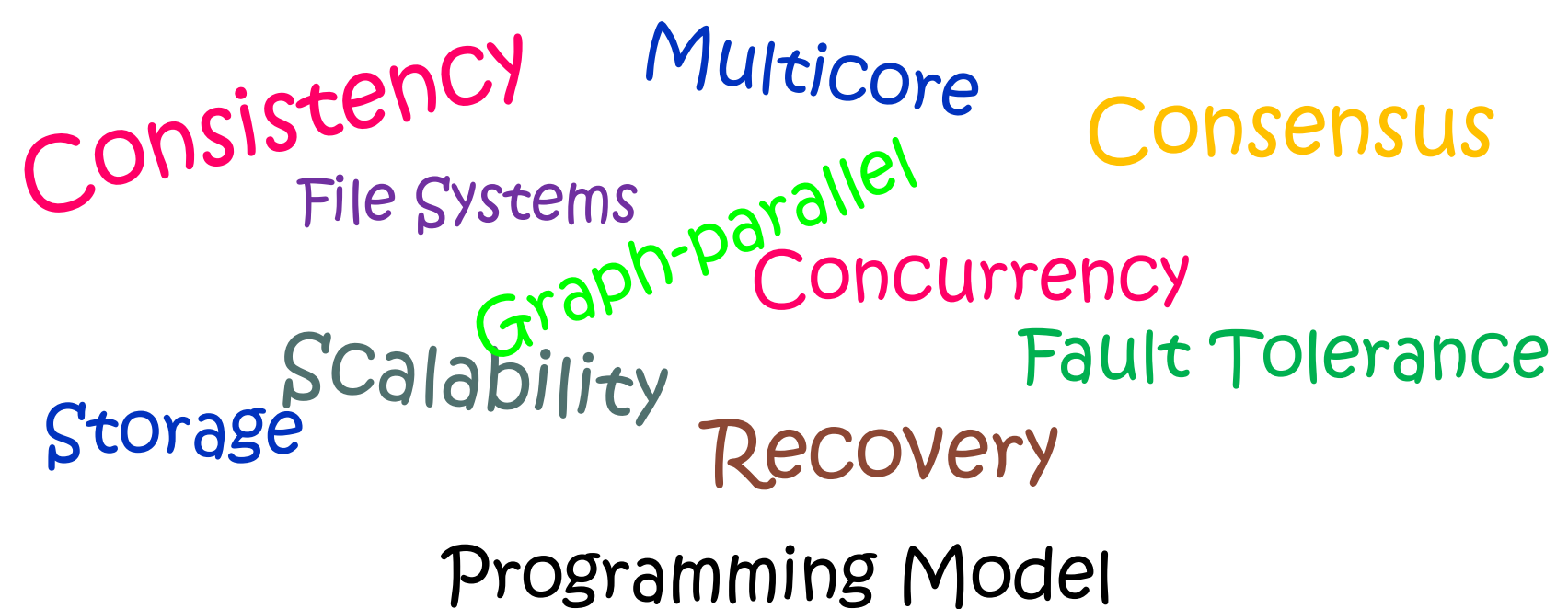
- Industry and Academic

Train yourself 😊

- ~~□ Present paper & ask question~~
- ~~□ Demo your idea & observation~~
- Paper review

Course Topics

Major: Distributed Systems



A word cloud of course topics. The words are arranged in a roughly circular pattern, with some overlapping. The colors of the words are: Consistency (pink), Multicore (blue), Consensus (yellow), File Systems (purple), Graph-parallel (green), Concurrency (pink), Fault Tolerance (green), Scalability (grey), Storage (blue), Recovery (brown), and Programming Model (black).

Consistency

Multicore

Consensus

File Systems

Graph-parallel

Concurrency

Fault Tolerance

Scalability

Storage

Recovery

Programming Model

Methodology

Courses

- Phase 1
 - Study classical systems (*not very old*)
 - Discuss practically-oriented concepts

Stunningly impressive capabilities now seem mundane. But lots of great stuff going on under the hood ...

– David Andersen (CMU)

Methodology

Courses

- Phase 1
 - Study classical systems (*not very old*)
 - Discuss practically-oriented concepts
- Phase 2
 - Present state-of-the-art systems (*not very new*)
 - Discuss new **trends** and **changes**

Methodology

Offline Tasks

- Pre-course
 - At a glance & Thinking questions
- Post-course
 - Paper review (800 words)
 - ~5 students per topic/paper

Methodology

Paper review (~800 words)

1. Summary of the paper
2. Pros: what you like about the paper
3. Cons: what you don't like about the paper
4. Criticism of the work
5. Comment paper section-by-section
6. Minor issues that should be fixed (grammar, typos, etc)
7. Questions to the authors
8. Future work

Course Schedule

Preliminary

- 3.06 Intro DS
- 3.13 Consistency: Sequential & Release
- 3.20 Consistency: Eventual & Causality
- 3.27 Crash Recovery & Logging
- 4.03 Concurrency Control
- 4.10 Two-phase Commit
- 4.17 Distributed Consensus: Paxos
- 4.24 DFS: NFS & GFS

Course Schedule

Preliminary

- 5.08 Data-parallel processing
- 5.15 Graph analytics
- 5.22 Data partitioning
- 5.29 Disk-based processing
- 6.05 Multicore & NUMA
- 6.12 Fault Tolerance
- 6.19 Review

Tentative Grading

Exam (85%)

Paper review (15%)

Outline

Course Information

- Faculty Info
- Why take this course?
- Course goal and methodology
- Grading

Intro to parallel/distributed systems

Building and Classifying Parallel and Distributed Systems

Flynn's Taxonomy (1966)

SISD

Number of **instruction** and **data** streams

- Traditional uniprocessor system

SIMD

- Array (vector) processor
- Examples: GPU, APU, SSE3, ...



Michael J. Flynn

MISD

- Generally not used and doesn't make sense

MIMD

- Multi-computers, each with PC, program, data
- **Parallel and Distributed systems**

Subclassifying MIMD

Memory

- Shared memory systems: **multiprocessors**
- No shared memory: **multicomputers**

Interconnect

- Bus
- Switch

Delay/Bandwidth

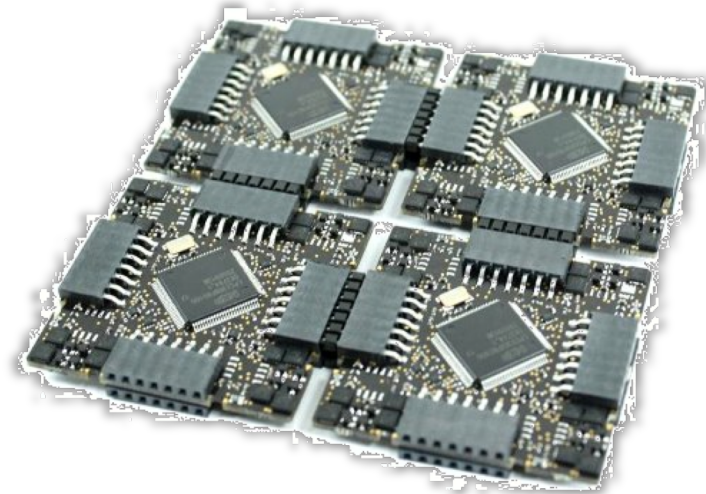
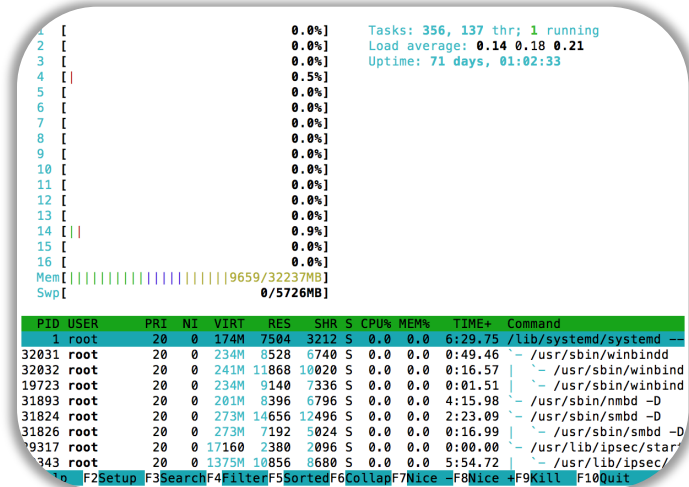
- Tightly coupled systems
- Loosely coupled systems

Parallel Systems: Multiprocessors

Shared memory

Shared clock

All-or-nothing failure



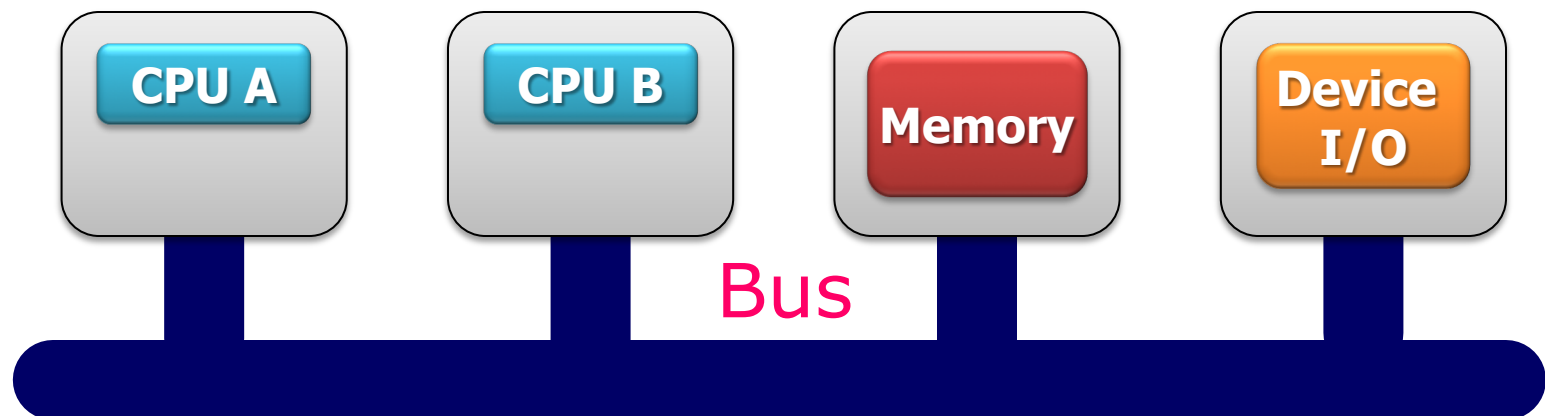
Bus-based Multiprocessors

SMP: Symmetric Multi-Processing

Characterized by

- UMA: Uniform Memory Access

Memory and peripherals are accessed via shared **bus**
System looks the same from any processor



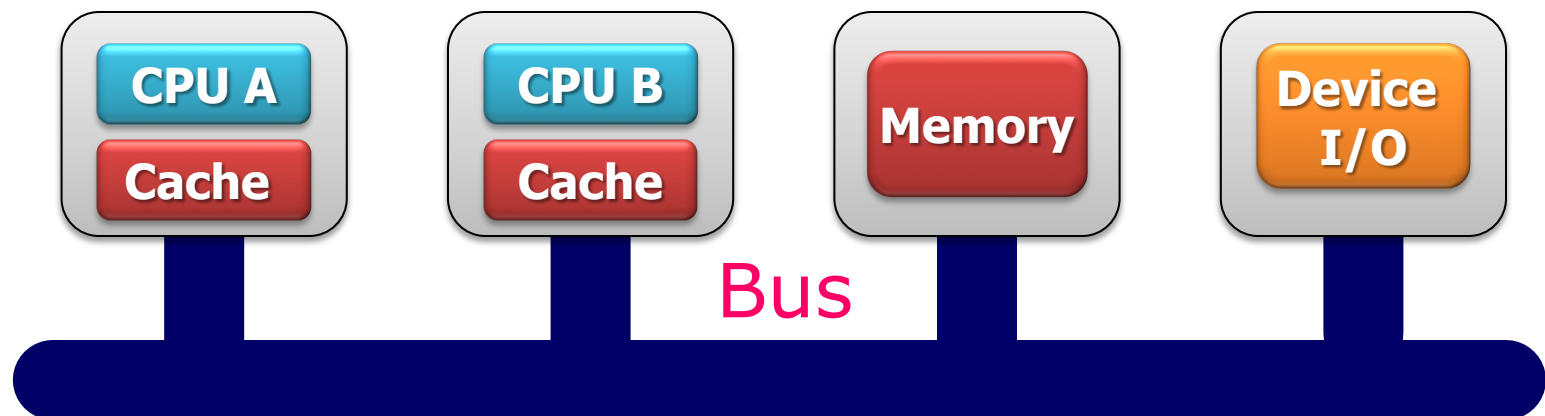
Bus-based Multiprocessors

How do we deal with bus overload & memory contention?

- Add local memory (cache)

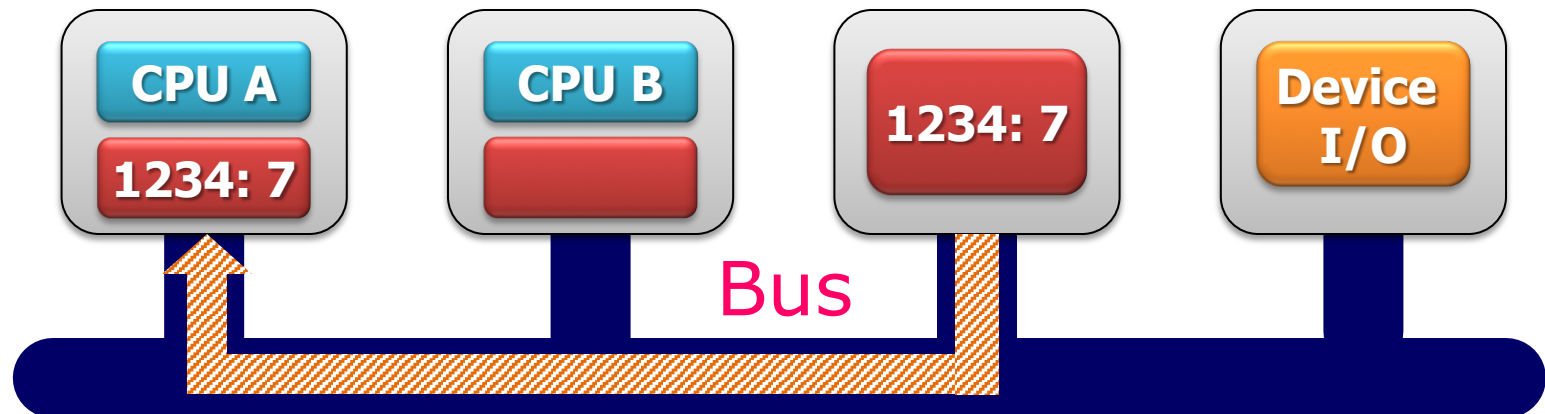
CPU reads/writes cache memory

- Access main memory only on cache miss



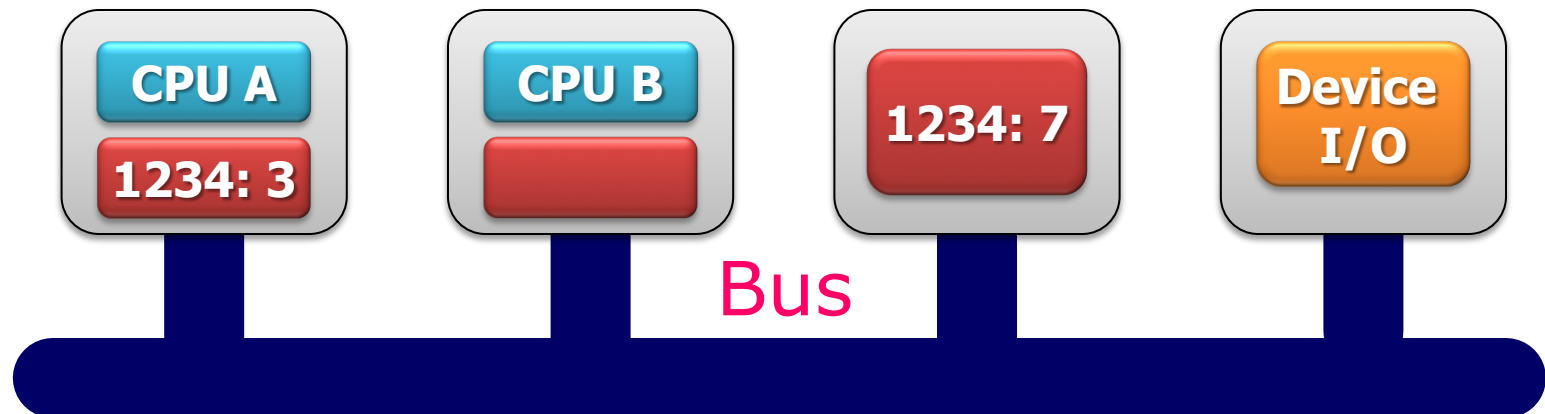
Working with a Cache

CPU A reads location 1234 from memory



Working with a Cache

CPU A modifies location 1234

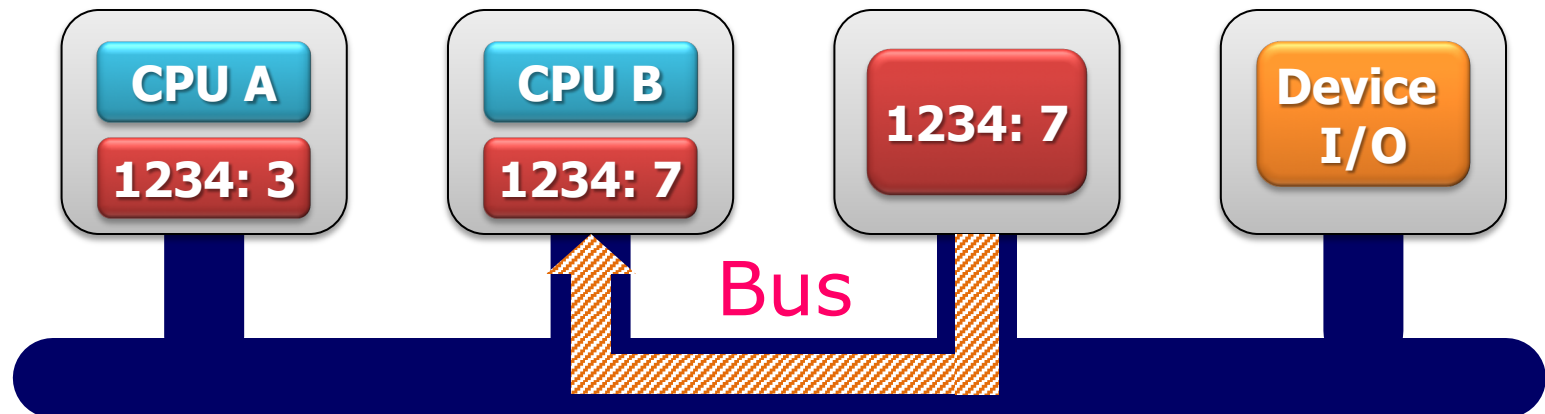


Working with a Cache

CPU B reads location 1234 from memory

Gets **stale** value

Memory not coherent!

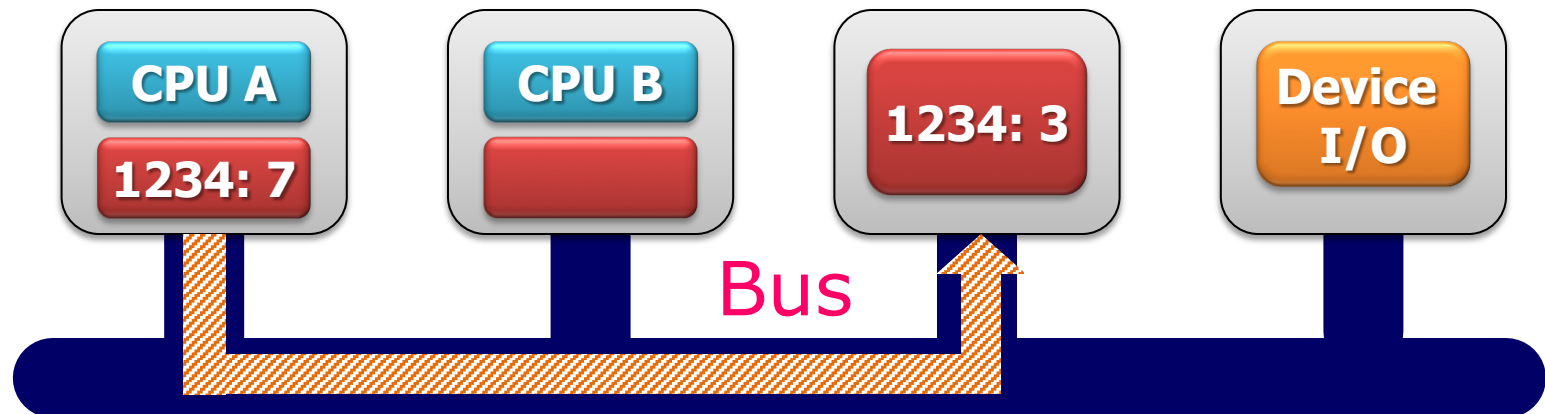


Working with a Cache

Fix coherency problem by writing all values through bus to main memory

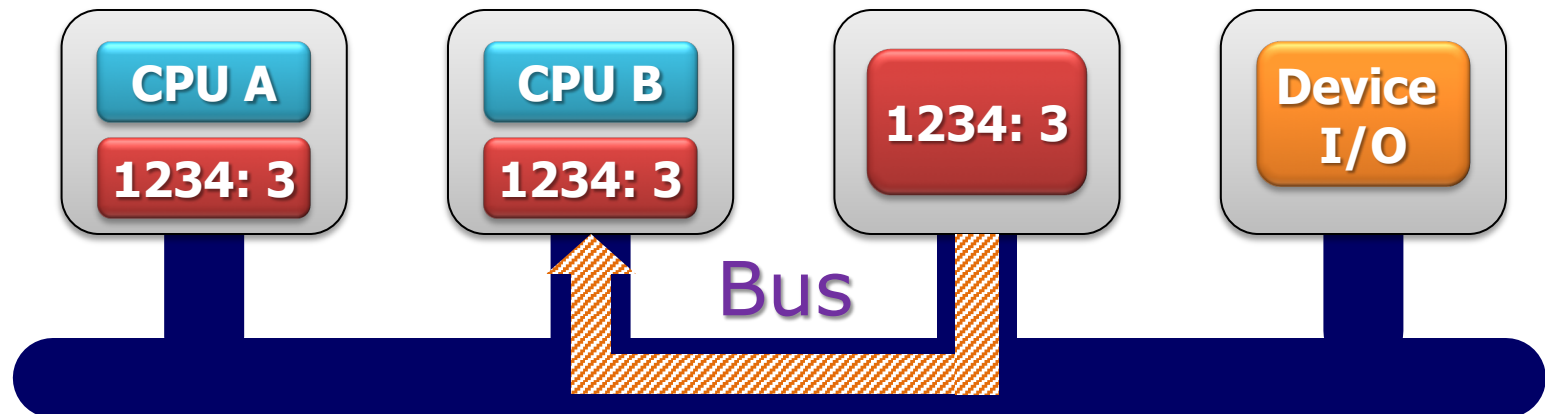
Write-through Cache

CPU A modifies location 1234, then **write-through** main memory is now coherent



Working with a Cache

CPU B reads location 1234 from memory

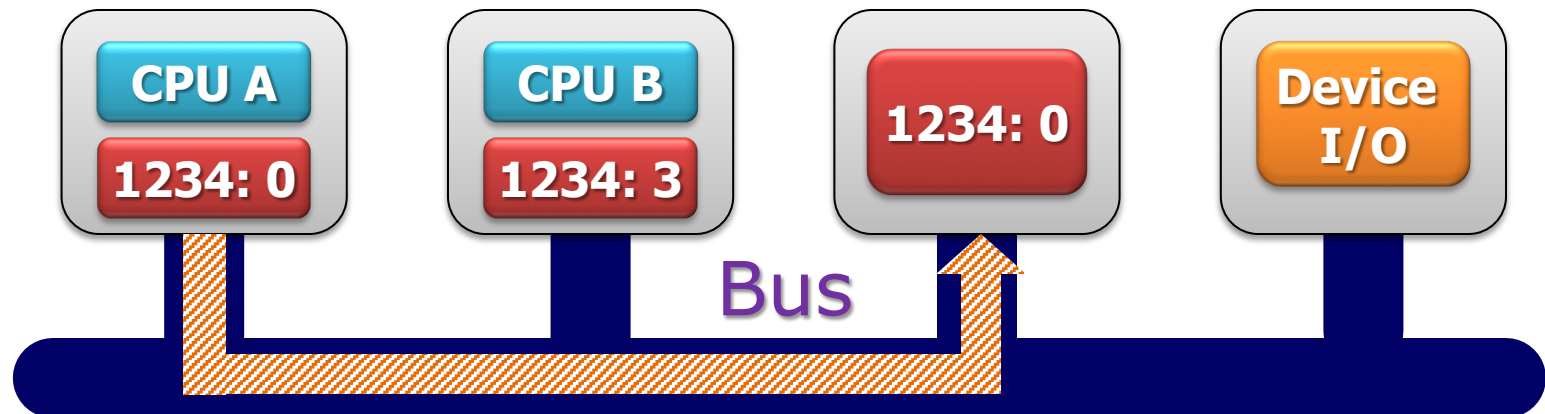


Working with a Cache

CPU A modifies location 1234 again

Cache on CPU B not **updated**

Memory not coherent!

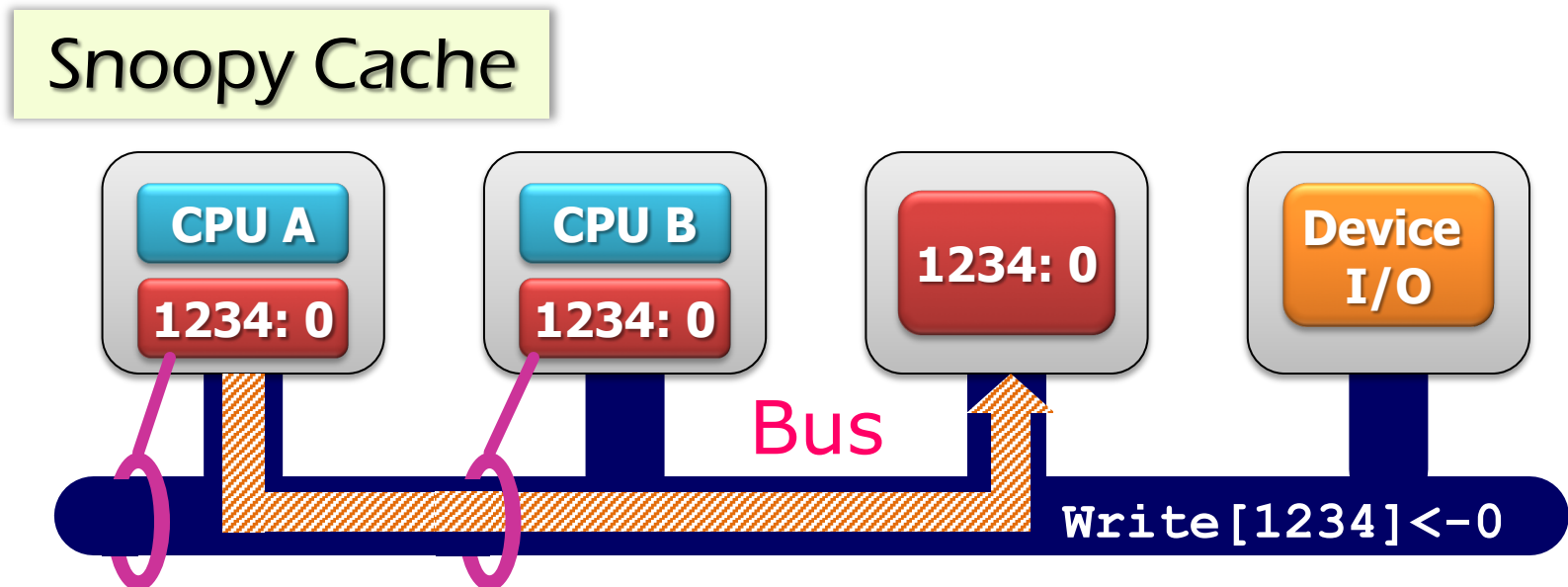


Working with a Cache

Add logic to each cache controller:

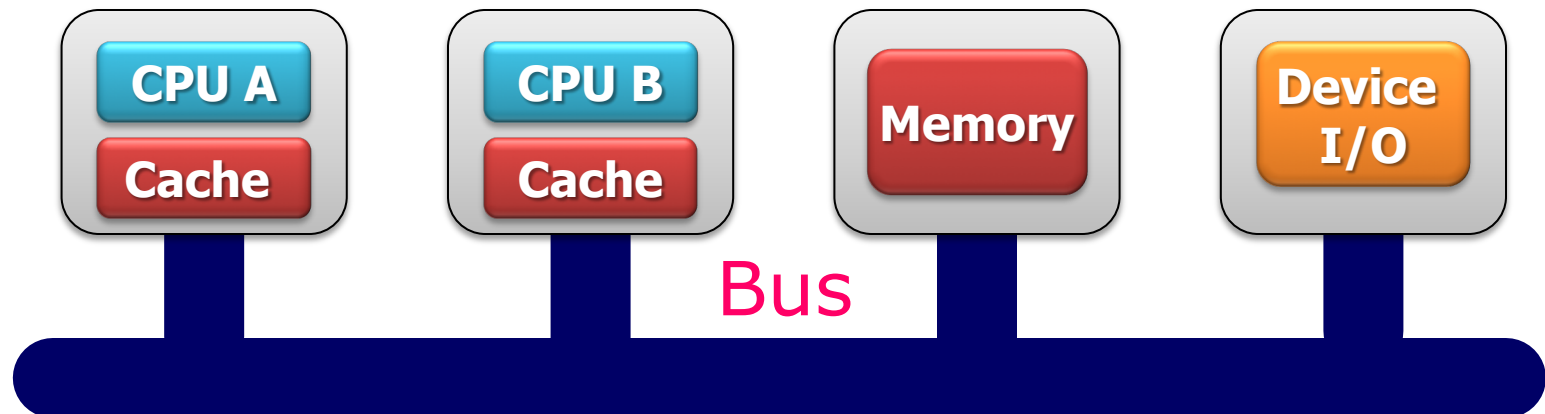
Monitor the bus for writes to memory

Virtually all bus-based architectures use a **snoopy** cache

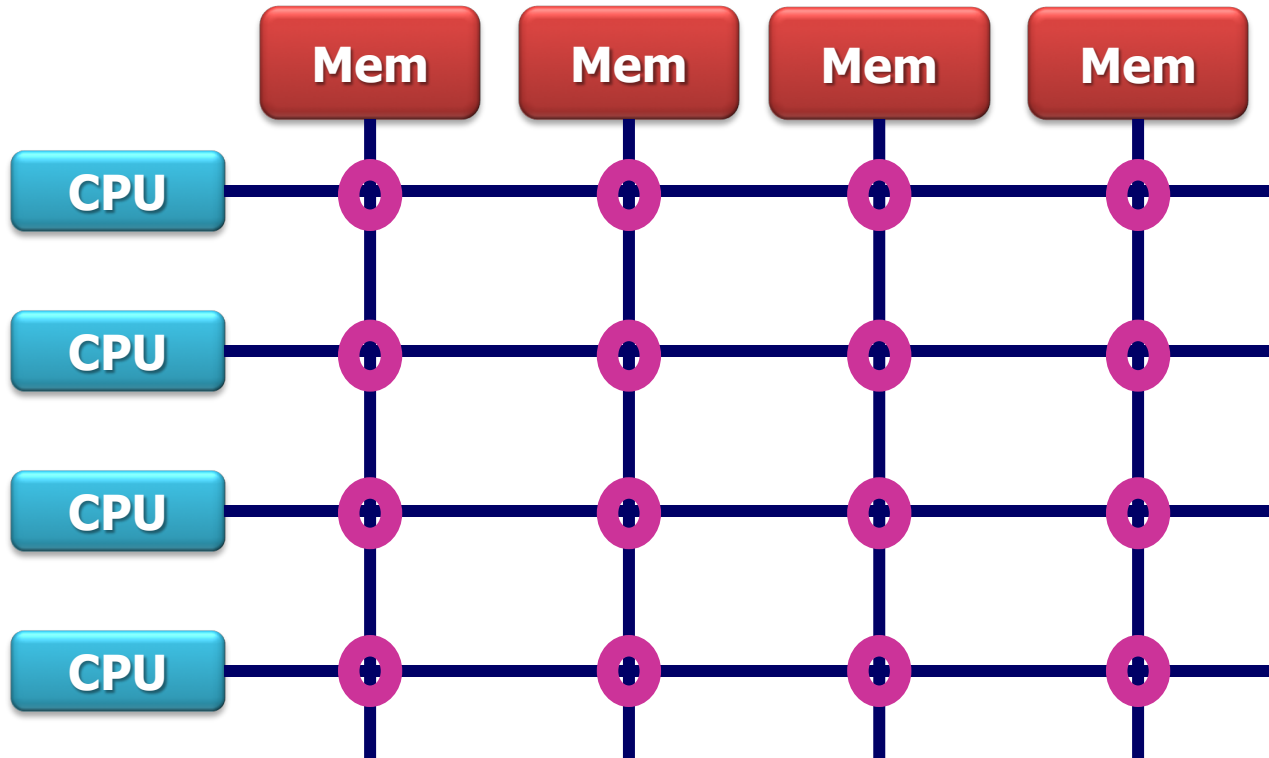


Switched multiprocessors

A bus-based architecture does not **scale** to a large number of CPUs (8+)



Switched multiprocessors



n^2 *crosspoint* switches

– expensive switching fabric

NUMA

Hierarchical Memory System

Each CPU has **local** memory: fast access

Other CPU's memory: slow access

Placement of code & data becomes difficult

- OS has to be aware of memory allocation and CPU scheduling

NUMA

SGI Origin's ccNUMA

AMD64 Opteron

- Each CPU gets a bank of DDR memory
- Inter-processor communications are sent over HyperTransport link

Intel Core i7

- Cores on the same package share memory via Integrated Memory Controller (IMC)
- Access memory from other nodes via Intel QuickPath Interconnect (QPI)

NUMA

Linux > 2.5 kernel

- Multiple run queues
- Structures for determining layout of memory and processors

Also supported in Windows, Oracle, ..

Distributed Systems

Multicomputers

or

Networked Computers

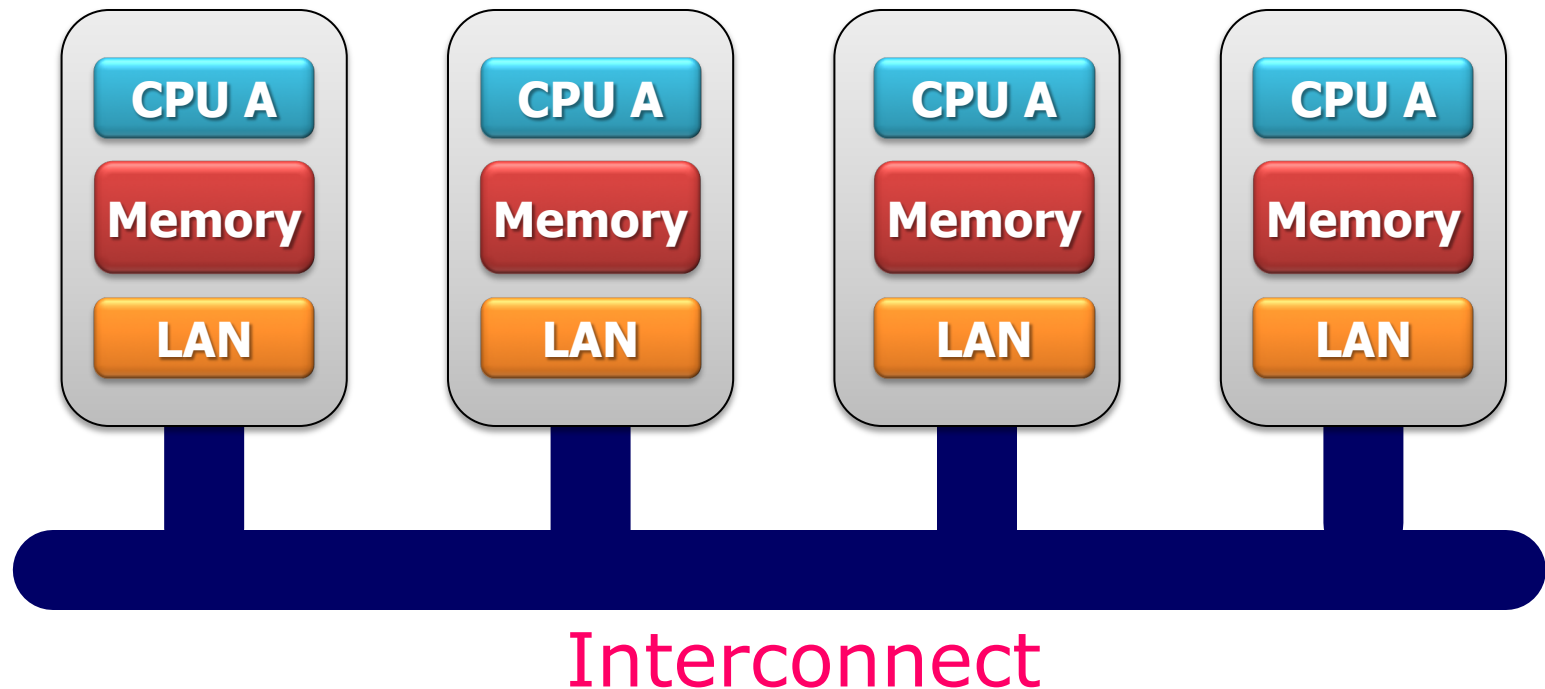
No shared clock

No shared memory

- Alternate communication mechanism needed
- Use network interconnect

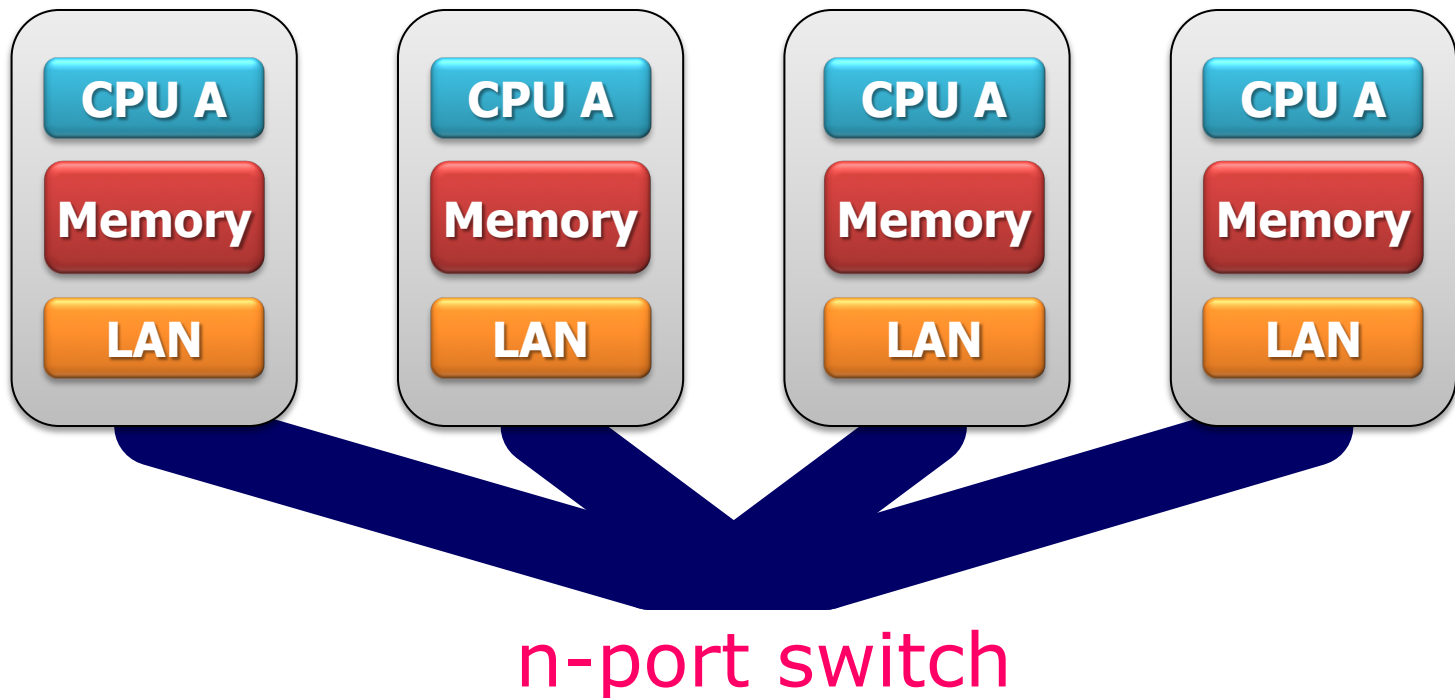
Bus-based Multicomputers

Collection of workstations on a LAN



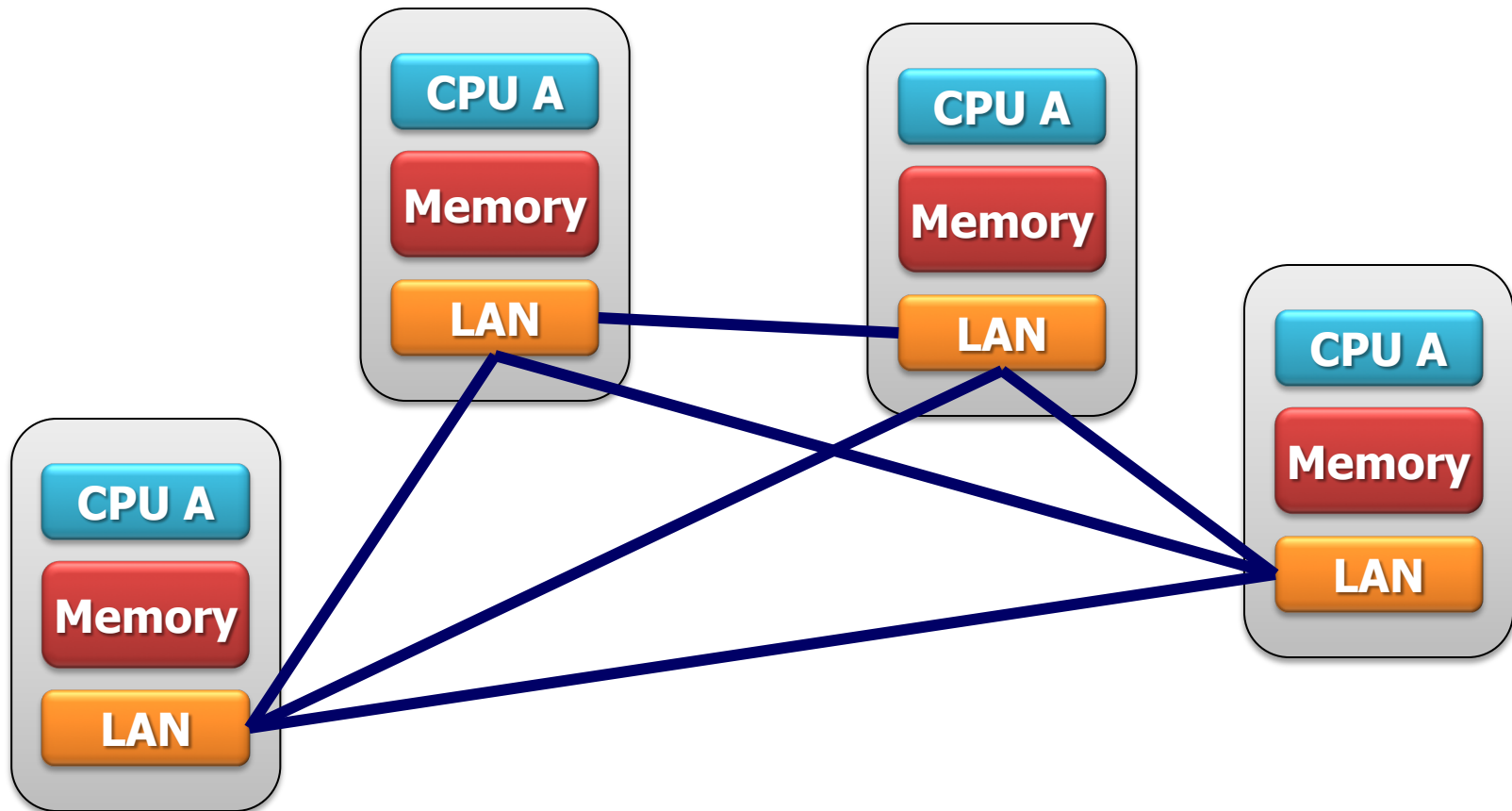
Switched Multicomputers

Collection of workstations on a LAN



Switched Multicomputers

Logical view: any-to-any communication



What is a Distributed Systems ?

A collection of **independent/autonomous** hosts connected through a communication network **working together** to perform **a service**

- No shared memory (must use the network)
- No shared clock

Examples

- Web servers
- Facebook
- Google search
- Producing an animated movie
- Amazon shopping cart
- Online game

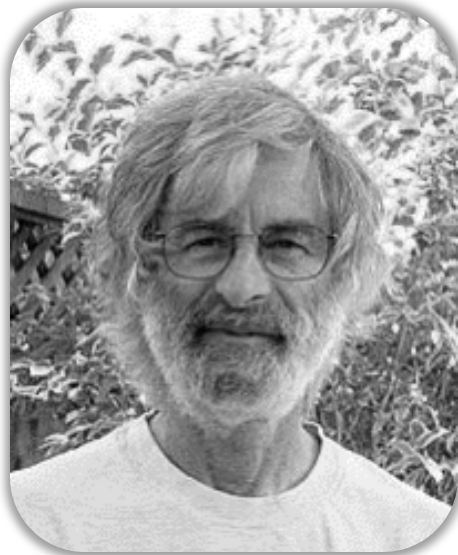
Single System Image

Collection of **independent** computers that appears as a single system to the user(s)

- Independent = autonomous
- Single system: user no aware of distribution

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



LESLIE LAMPORT

United States – 2013

CITATION

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.

Distributed Software Service Models

Multiprocessor/Multicore OS

Multiple CPUs: UMA/NUMA

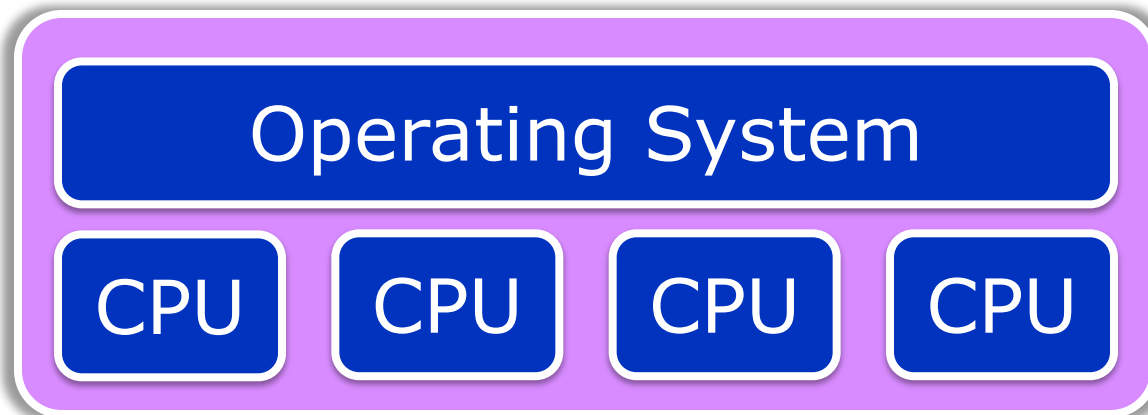
#Processors transparent to programs

Single OS and system image

Same communication mechanisms

- *Semaphores, shared memory, messages*

Networking not needed: we have shared memory



Network OS

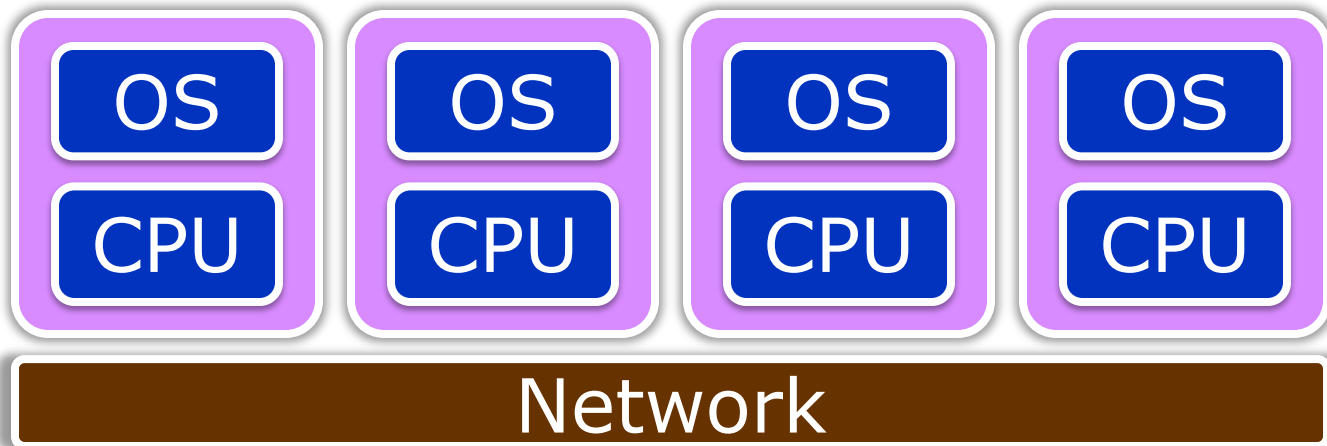
Autonomous nodes comm. via a network

Configurations may differ

Not a single system image

Distribution is explicit

Examples: Linux, Windows, OS X

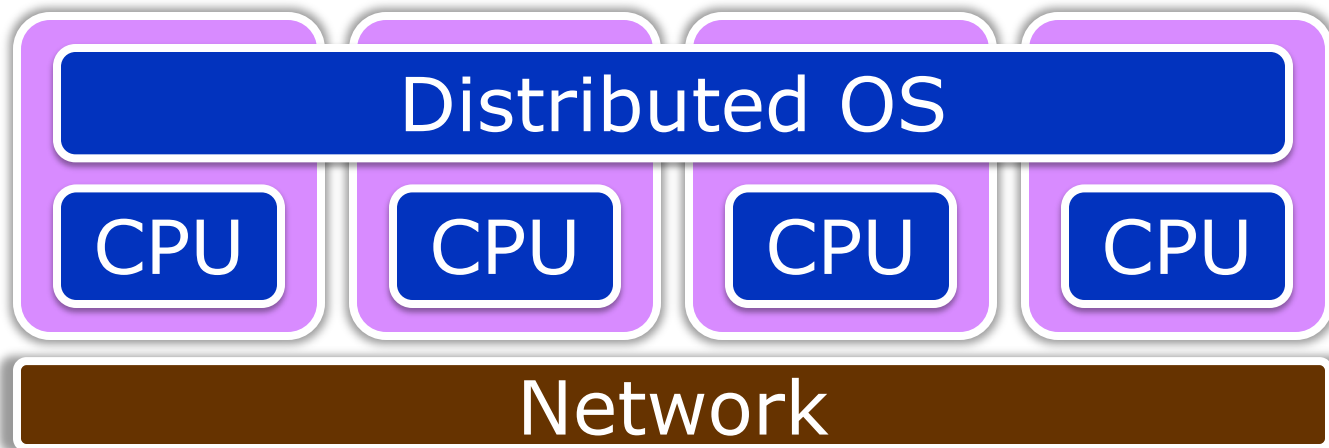


Distributed OS

Operating systems are aware of multiple systems and coordinate resource sharing

Single system image for file systems, processes, devices

Homogeneous hardware



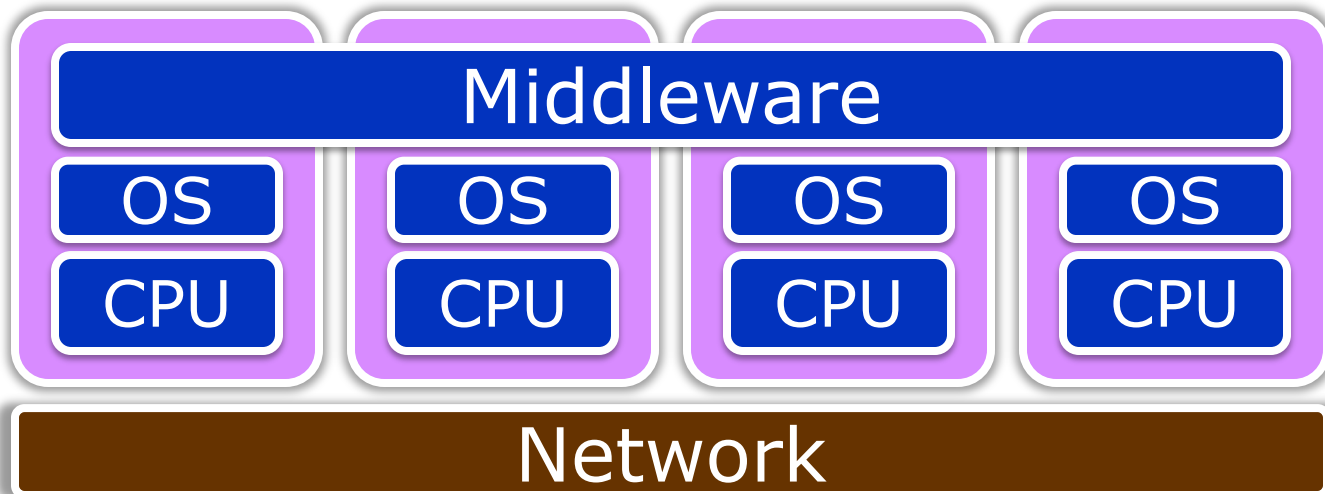
Middleware

User-level software (libraries): layer on top of network OS

Provides common interface for transparency of specific functions

Often OS-agnostic

Examples: PRC, RMI, .NET, MapReduce



Why build distributed systems?

Scalability

A system is said to be **scalable** if it can handle the **addition** of users and resources without suffering noticeable loss of **performance** or an increase in administrative **complexity**

– B. Clifford Neuman, 1994

Scalability considerations:

- ❑ #Components (users, data, CPUs): overloaded
- ❑ Geography: communication latency, bandwidth, reliability
- ❑ Administration: complexity

How can you get massive perf.?

Multiprocessor system don't scale

Disney's Cars 2 required 11.5 hours to render each frame (average) – some took 90 hours

- 12,500 cores on Dell render blades
- $\sim 152,640$ frames = 1,755,360 hours = 200ys

Google

- Approximately 1 billion queries per day
- Index > 25 billion web pages
- Hundreds of thousands of servers

Why build distributed systems?

Performance ratio

- Scaling multiprocessors may not be possible

Distributing applications may make sense

- ATMs, graphics, remote monitoring

Interactive communication & entertainment

- Work and play together: email, gaming, instant messaging, telephony

Remote content

- Web browsing, music & video downloads, IPTV

Mobility

Increased reliability

Transparency

High level: hide distribution from **users**

Low level: hide distribution from **software**

- **Location**: users don't care where resource are
- **Migration**: resources move at will
- **Replication**: users cannot tell whether there are copies of resources
- **Concurrency**: users share resources
- **Parallelism**: operations take place in parallel without user's knowledge

Problems

Designing dist. software can be difficult

- ❑ Operating systems handling distribution
- ❑ Programming language?
- ❑ Efficiency?
- ❑ Reliability?
- ❑ Administration?

Network

- ❑ Disconnect, loss of data, latency, bandwidth, ..

Security

- ❑ Want easy and convenient access

Design Issues

Reliability

- Availability: fraction of time system is usable
- Durability: data must not get lost

Performance

- Communication network may be slow and/or unreliable

Scalability

- Distributable vs. centralized algorithms
- Can we take advantage of having lots of computers?

Service Models

Centralized Model

No networking

Traditional time-sharing system

Direct connection of user terminals to system

One or several CPUs

Not easily scalable

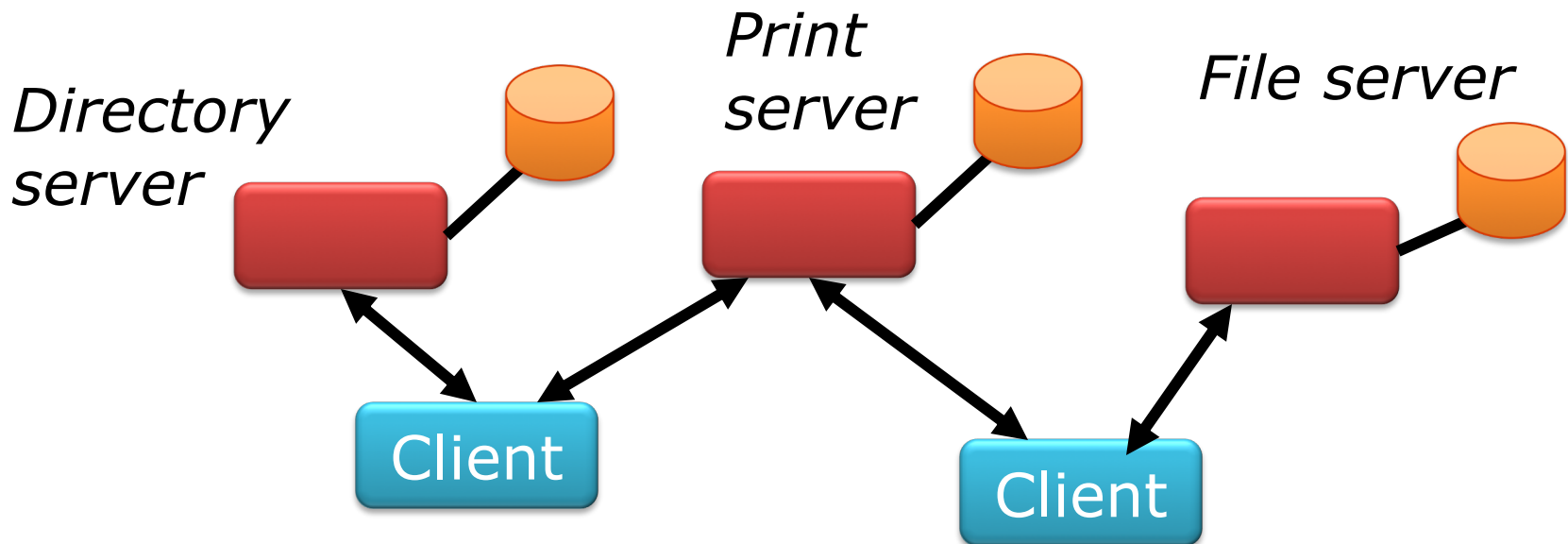
Limiting factor: #CPUs in system

- *Contention* for same resources

Client-server Model

Environment consists of clients and servers

- *Service*: task machine can perform
- *Server*: machine that performs the task
- *Client*: machine that is requesting the services



Peer-to-peer Model

Each machine on network has (mostly) equivalent capabilities

No machines are dedicated to serving others

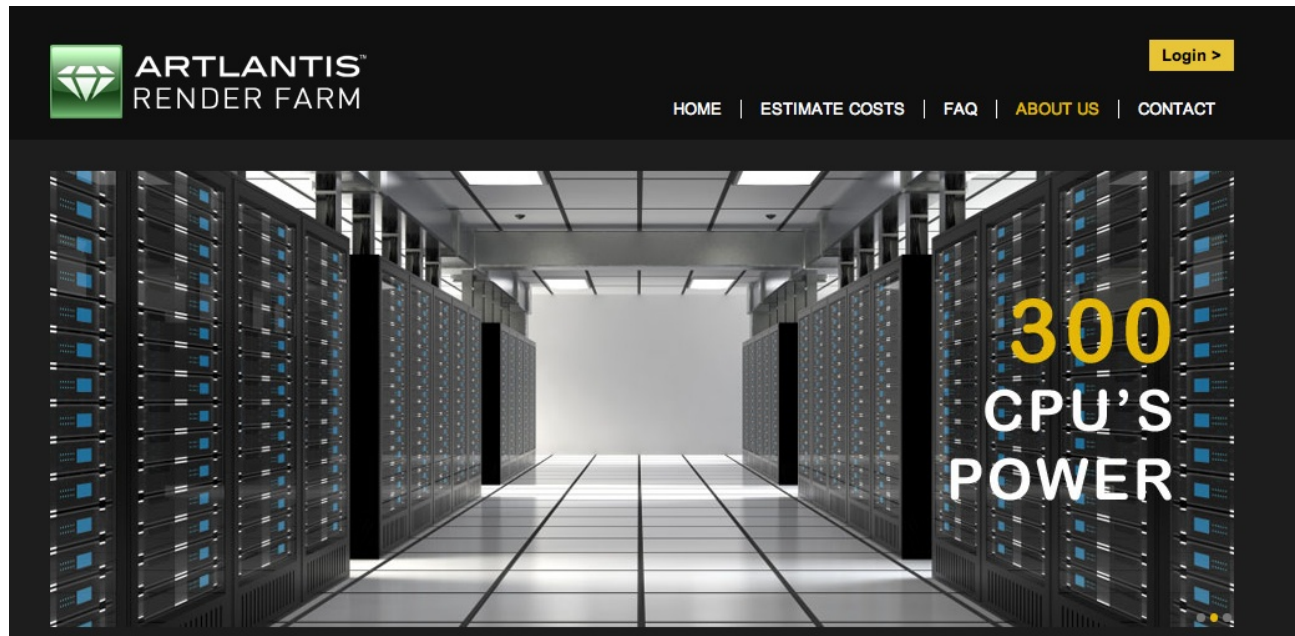
E.g. collection of PCs:

- Access other people's files
- Send/receive email (without server)
- Peer-to-peer file sharing
- SETI@home computation

Processor Pool Model

Collection of CPUs that can be assigned processes on demand

Example: Render farms



Cloud Computing Model

Provide users with **seamless** access to

- Computation
- Storage
- Communication

Heterogeneous and **geographically**
distributed systems

Build a “supercomputer” on the fly via
networked, loosely coupled computers

Cloud Computing Model

Offerings exist at various levels of abstraction

- Infrastructure as a Service (IaaS)
 - e.g., Amazon EC2, Rackspace, UCloud
- Platform as a Service (PaaS)
 - e.g., Windows Azure, Heroku
- Software as a Service (SaaS)
 - e.g., Google AppEngine, Salesforce.com

Case Study: Typical DataCenter

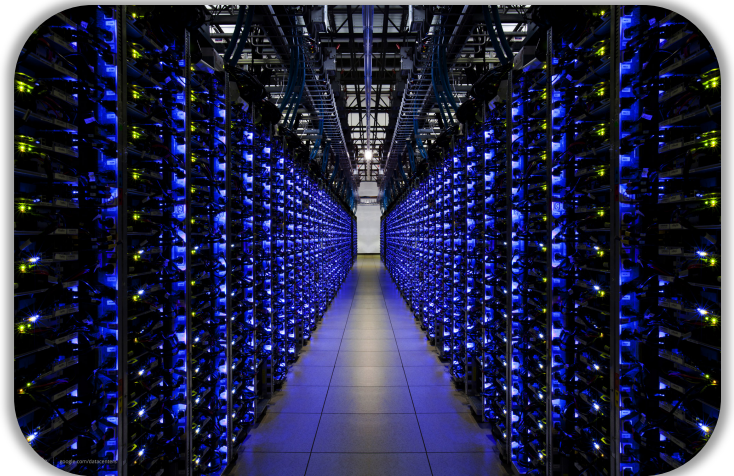
Typical DataCenter

10K-100K servers

40 servers per rack

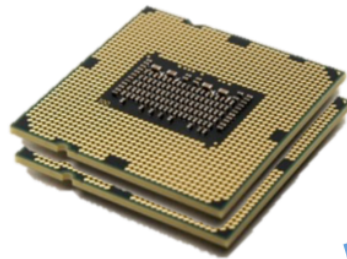
10 Gbps bandwidth in rack

10-100 MW of power



Typical DataCenter

CPUs:
16-32 cores

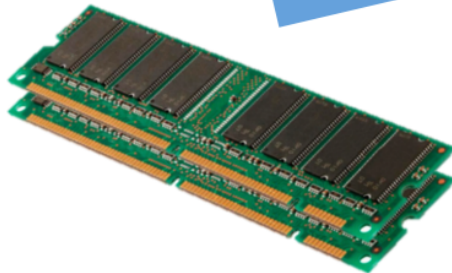


1 GB/s



NIC: 10 Gbps

50 GB/s



DRAM: 64-256 GB

0.1 GB/s
each



HDDs: 2-40 TB

1 GB/s each



SSDs?

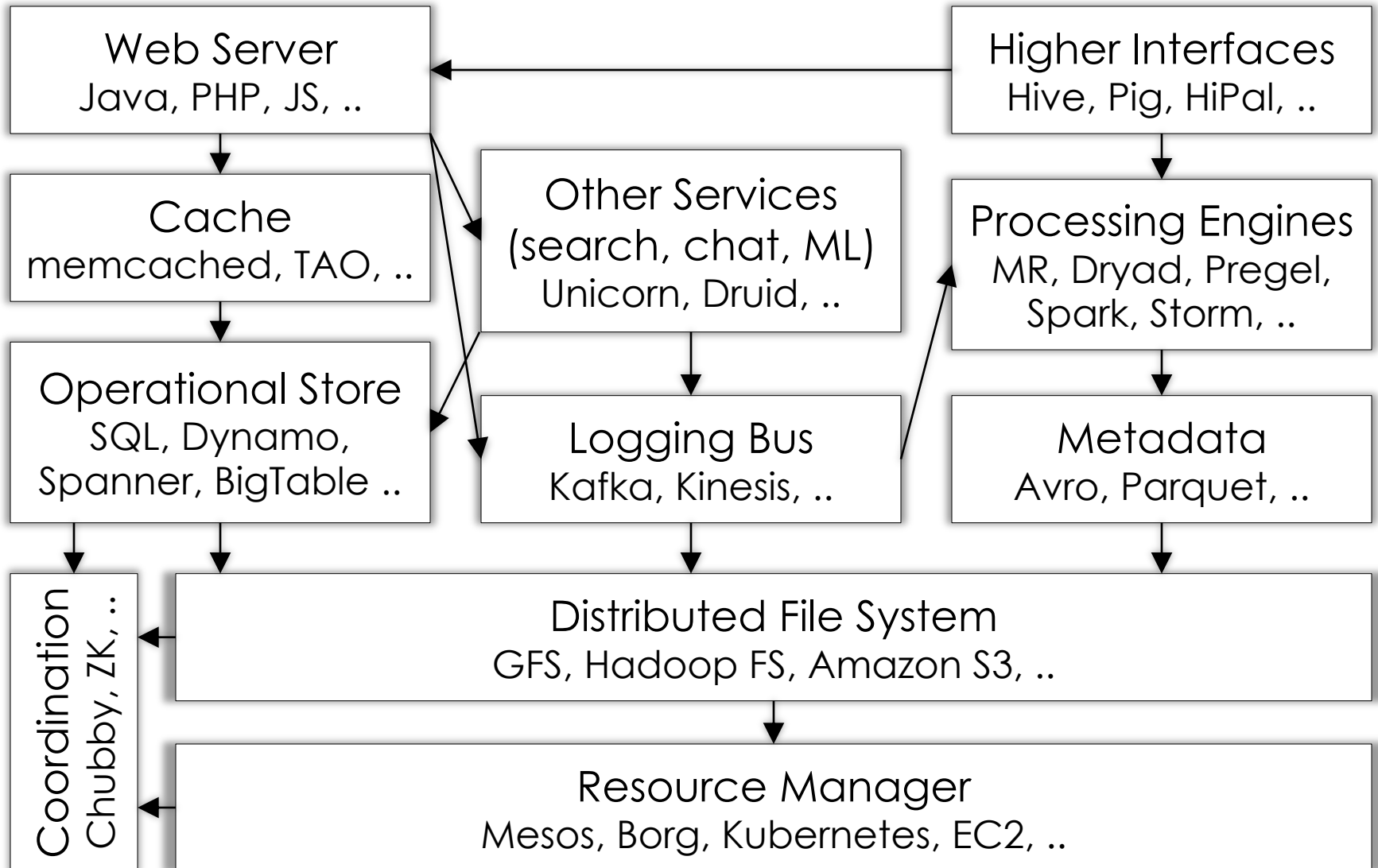
Types of Applications

Single big computation (e.g. big data)

Hosted apps for many tenants (e.g. Gmail, web hosting, cloud)

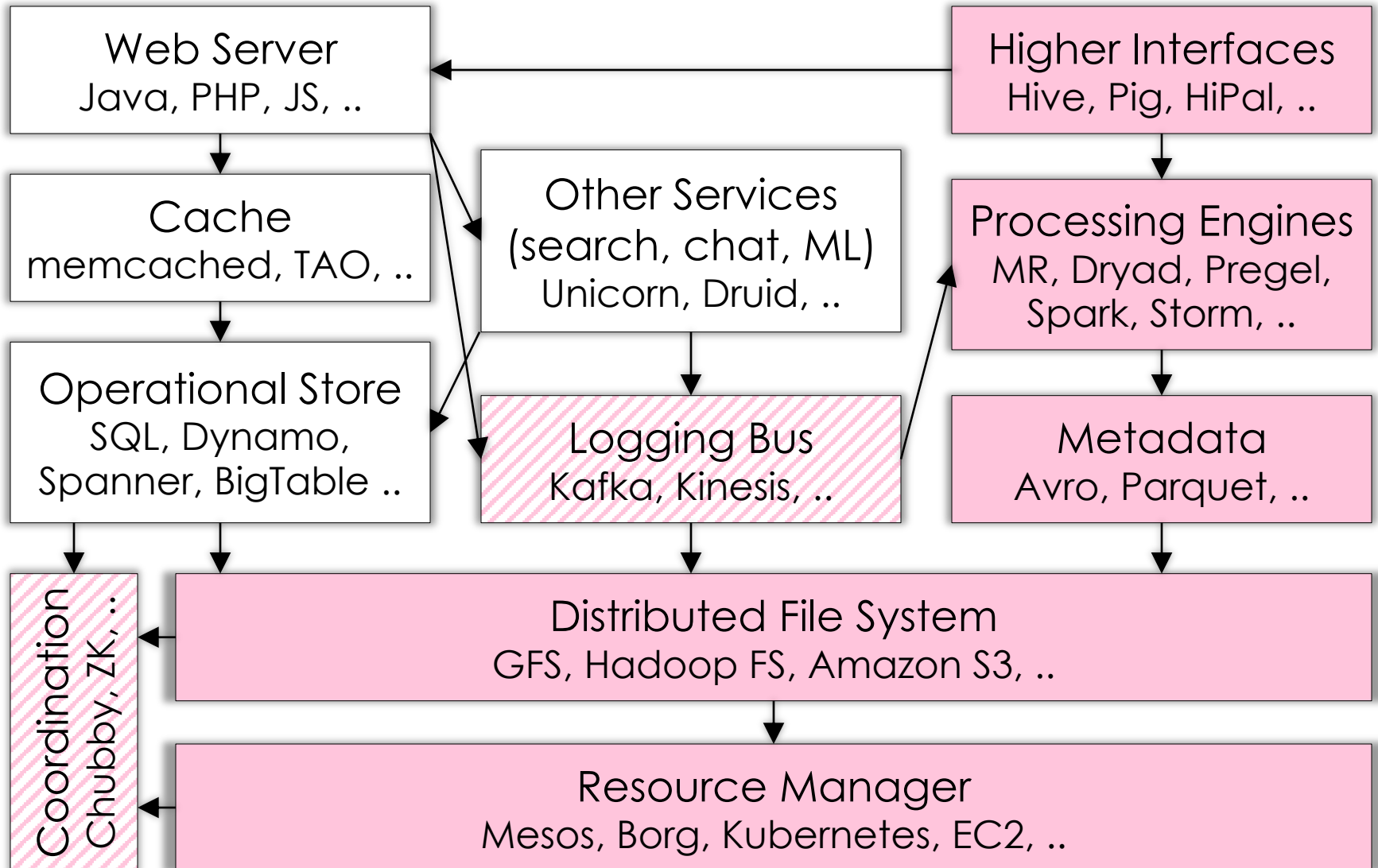
Single big multi-user app (e.g. Facebook)

Software Stack



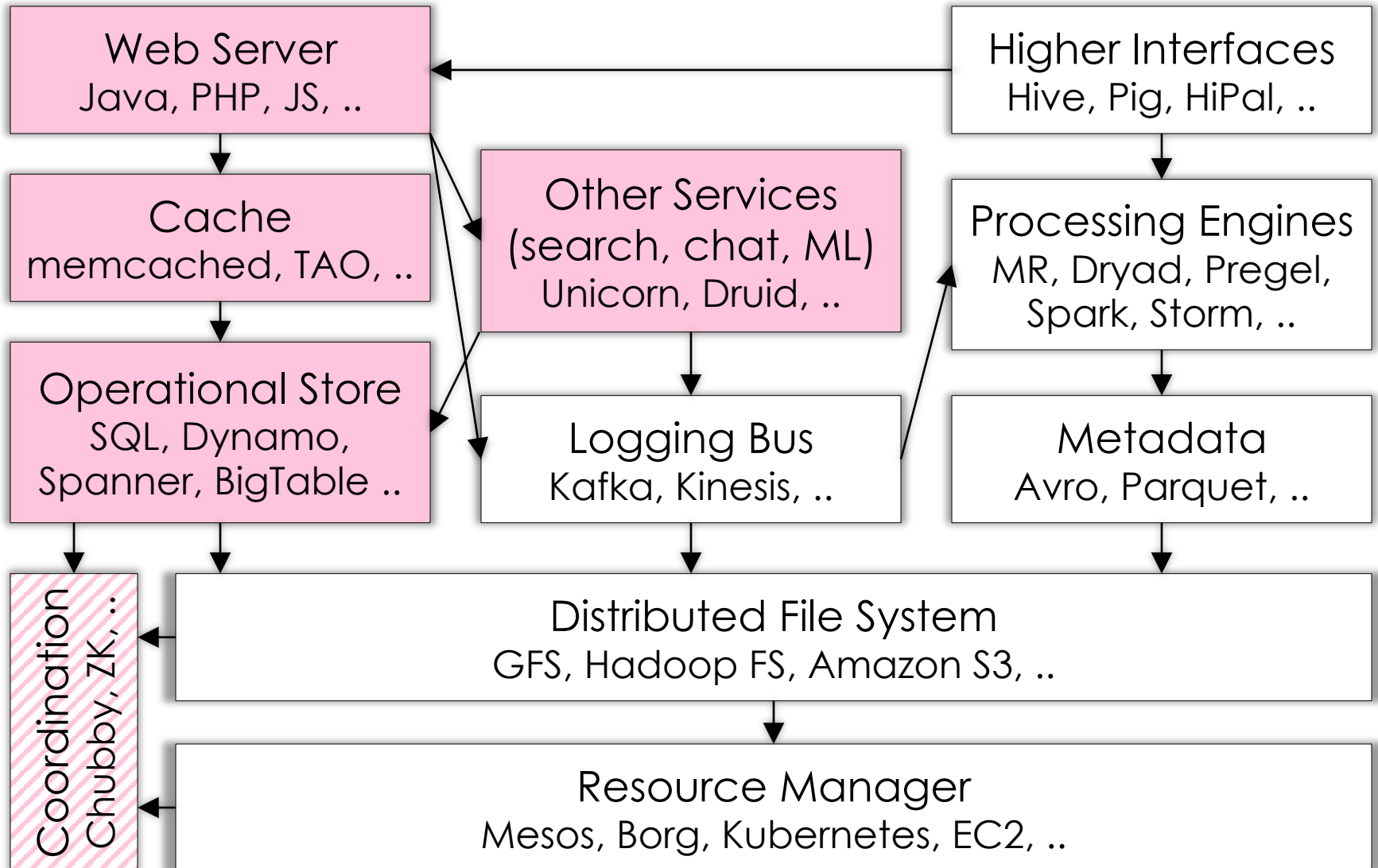
Software Stack

Dara Processing

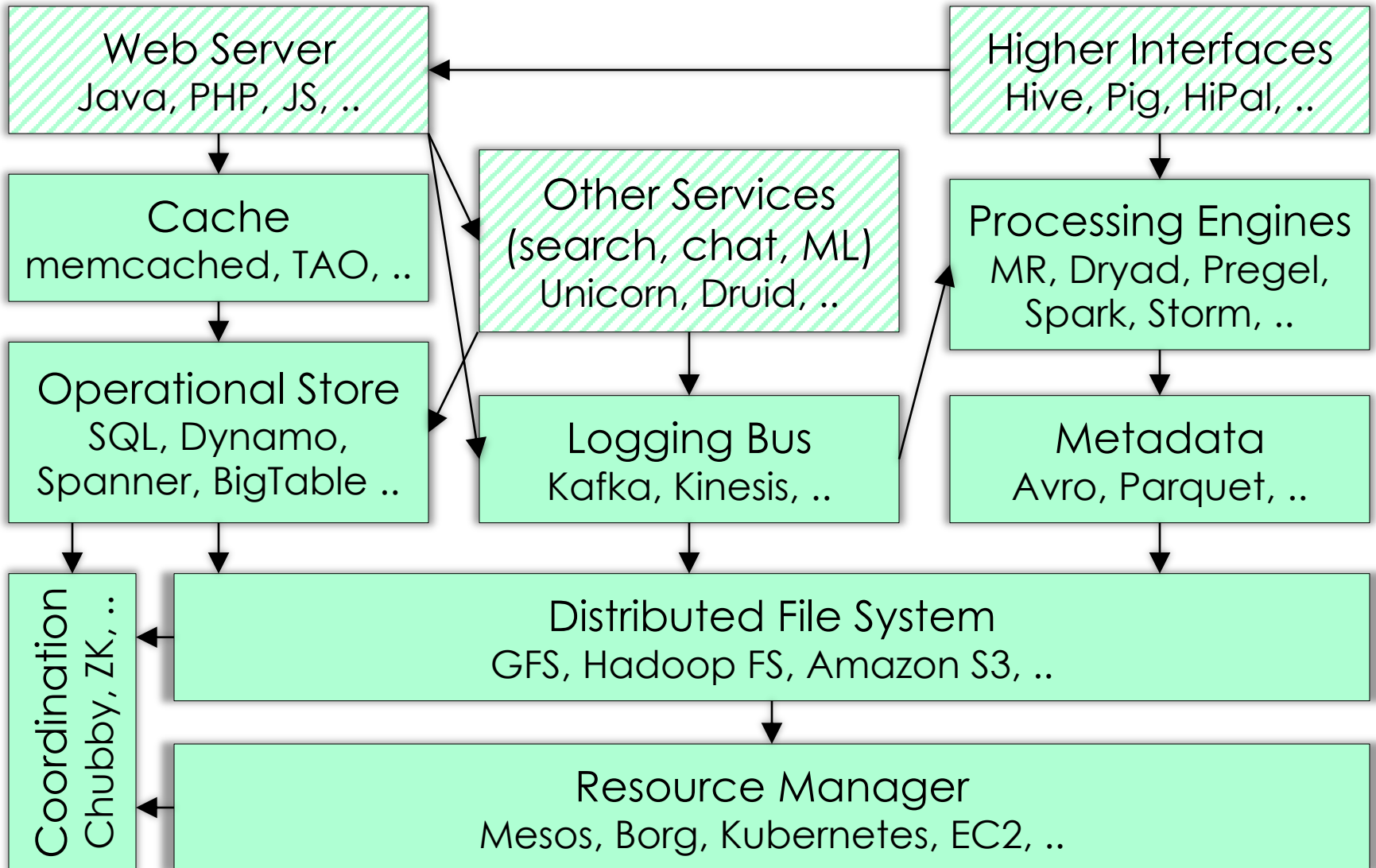


Software Stack

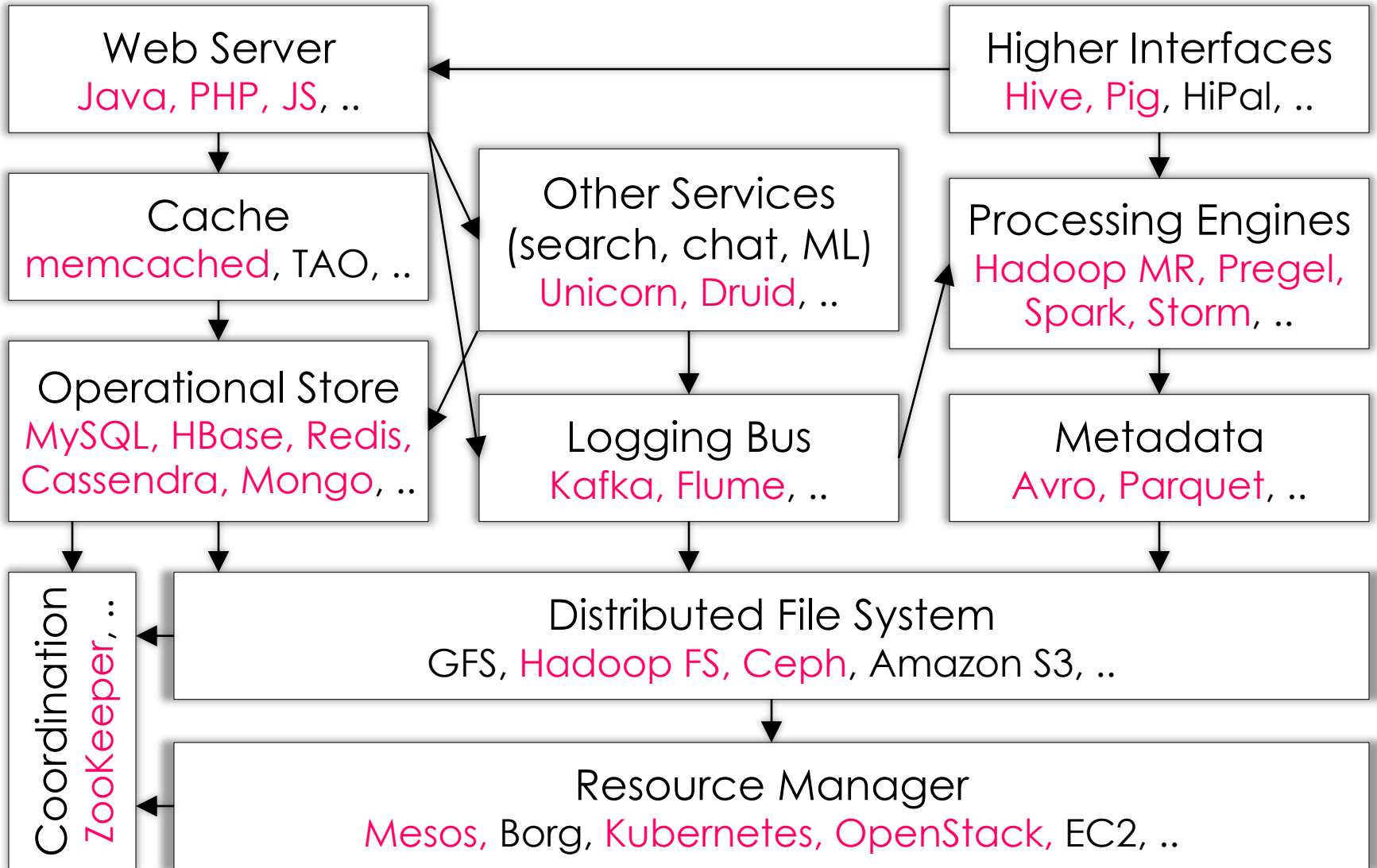
Online Apps



Offered by Cloud Services

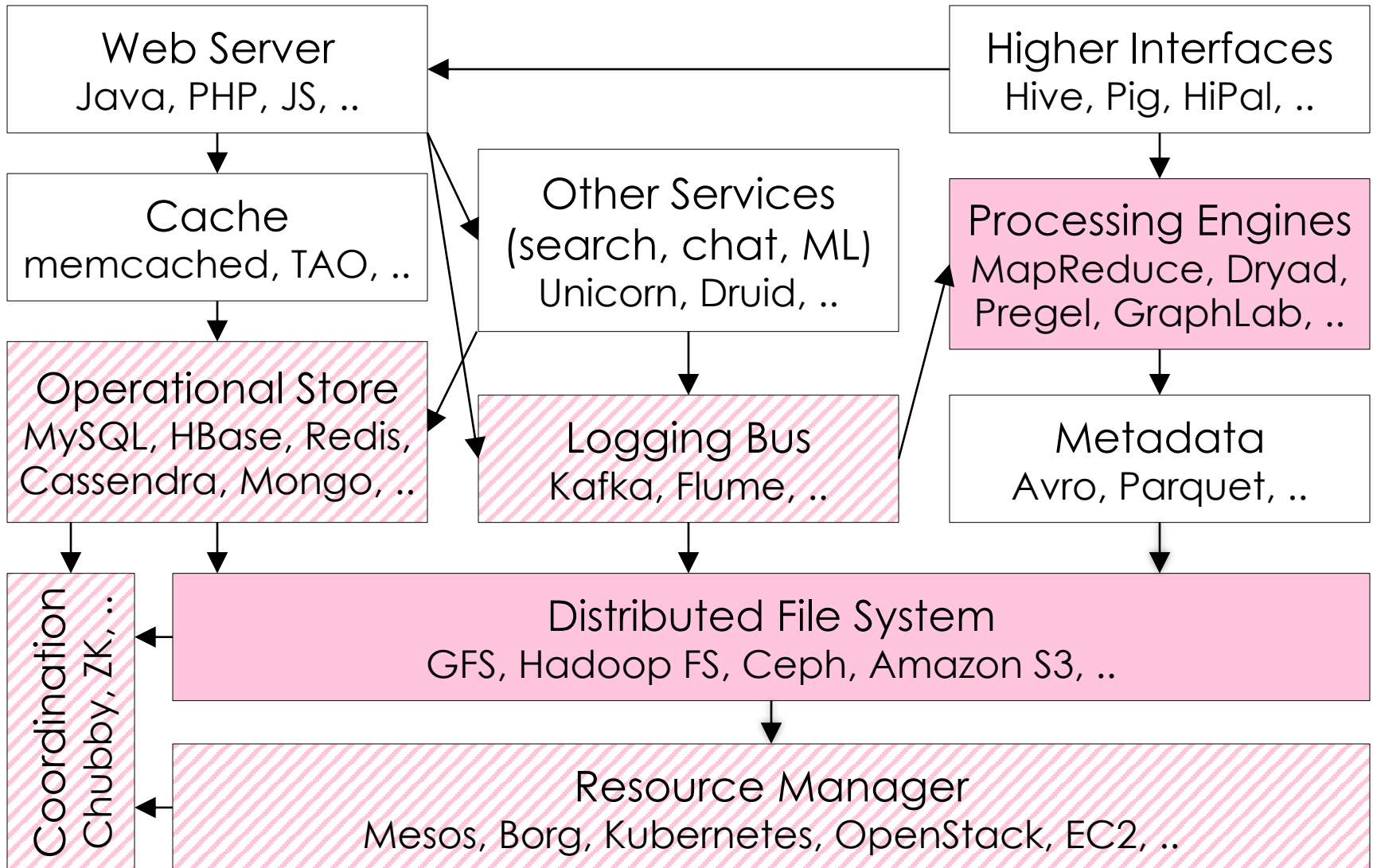


Available as Open Source



Software Stack

Appear in DS Course



Consistency

Multicore

Consensus

File Systems

Graph

Concurrency

Topics in this courses

Storage

Scalability

Fault Tolerance

Recovery

Programming Model

Next Lecture

Topic: Sequential/Release Consistency

THANKS