Parallel and Distributed Systems

Rong Chen

Institute of Parallel and Distributed Systems

Shanghai Jiao Tong University

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

Credits: some slides from Paul Krzyzanowski (Rutgers), Haibo Chen (SJTU) and et al.

Outline

Course Information

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

Intro to parallel/distributed systems

Faculty Information

陈榕 (Rong Chen)

Email: rongchen@sjtu.edu.cn

Office phone: 13616861826

Assistant: 丁鑫 (Xin Ding)

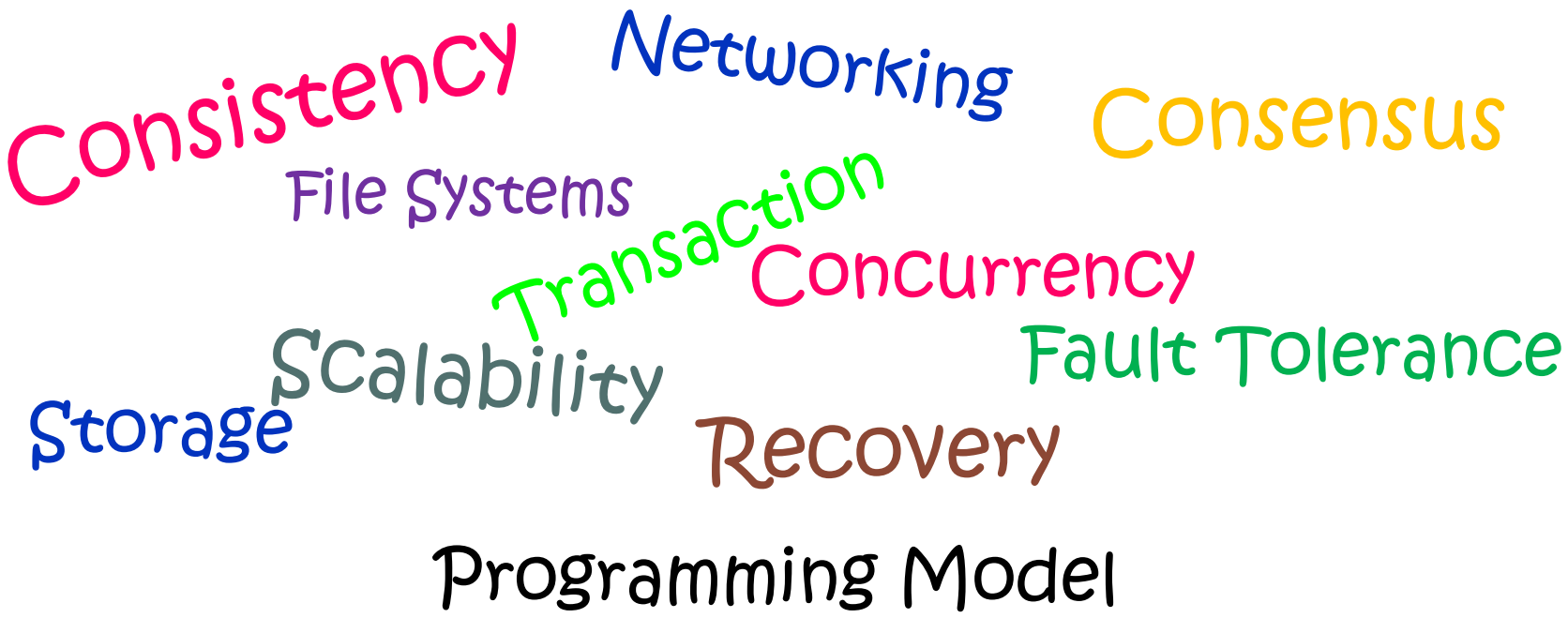
Email: sandy.dingxin@gmail.com

Course site:

http://ipads.se.sjtu.edu.cn/courses/ads

Course Topics

Major: Distributed Systems



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

  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*** has made area much more attractive and timely
- **Hard/unsolved**: not many deployed sophisticated **distributed** systems

Course Goal

Introduce you to (distributed) system **design concepts** and **principles**

- Modularity, Layering, Reliability, …

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

- Industry and Academic

Course Goal

- Introduce you to (distributed) system **design concepts** and **principles**
  - Modularity, Layering, Reliability, …
- Cover the design and evolutions of many important (distributed) computer **systems**
  - Industry and Academic
- Train **yourself** ☺
  - Presentation & ask question
  - Demo your idea & observation

Methodology

Courses

- Topic Introduction
  - 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

- Topic Introduction (90min)
  - Study **classical** systems (*not very old*)
  - Discuss **practically**-oriented concepts
- Team Show (45min)
  - Present **state-of-the-art** systems (*at present*)
  - Demonstrate **funny** features
  - Team: 3 persons

Tentative Grading

Exam 40%

Topic Introduction (10%)

- 1 paper before lecture, and hand in answers to paper questions before lecture

Team Show (50%)

- Presentation (20%)
- Demo (20%)
- Ask Questions (10%)

Tentative Grading

Presentation (25min+5min)

- Grading: Instructor (50%) + Classmates (50%)
- *Slides Presentation*
  - ✓ Basic(3): get the main idea cross
  - ✓ Organization(1): flow of slides, good timing
  - ✓ Style(1): easy to understand
- *Oral Presentation*
  - ✓ Basic(3): understandable, main idea comes
  - ✓ Clarity(1): clear explanation
  - ✓ Style(1): interesting, motivating

Tentative Grading

Demo (15min+5min)

- Grading: Instructor (50%) + Classmates (50%)
- *Innovation*
  - ✓ Basic(2): relevant, fully prepared
  - ✓ Attractive(2): irradiative, surprise
  - ✓ Robust (1): comprehensive, reproduction
- *Presentation*
  - ✓ Basic(3): understandable, main idea comes
  - ✓ Clarity(1): clear explanation
  - ✓ Style(1): interesting, motivating

Tentative Grading

Ask Questions

- After each *presentation* or *demo*
- Each valid question: 3~5 points
- Only consider best 2 questions

- *Valid question*
  - ✓ Relevant
  - ✓ Meaningful
  - ✓ Important

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)

Number of instruction and data streams

SISD

- Traditional uniprocessor system

SIMD

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

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

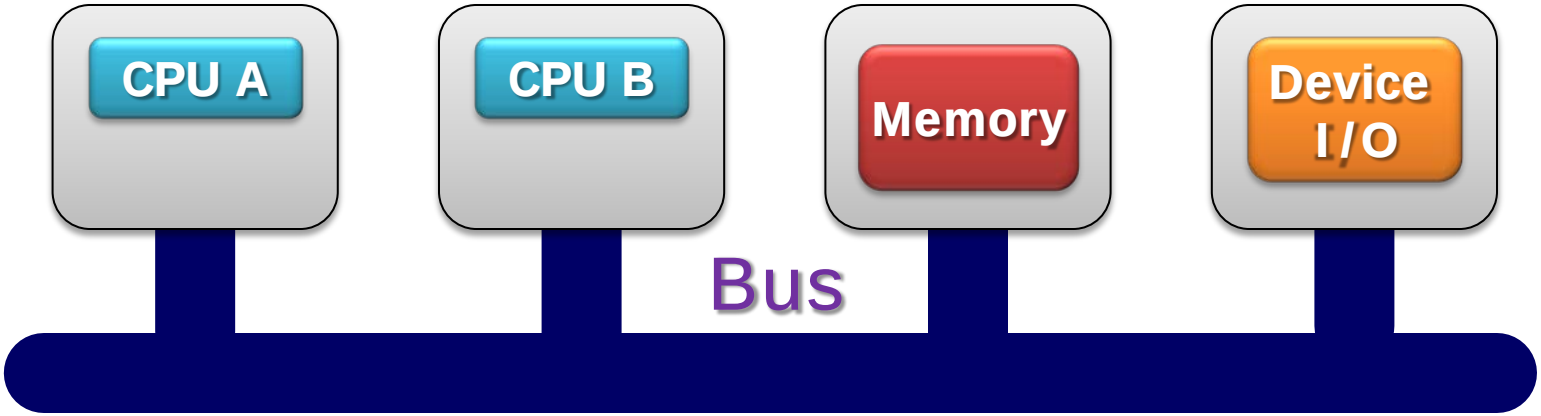
Bus-based Multiprocessors

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.



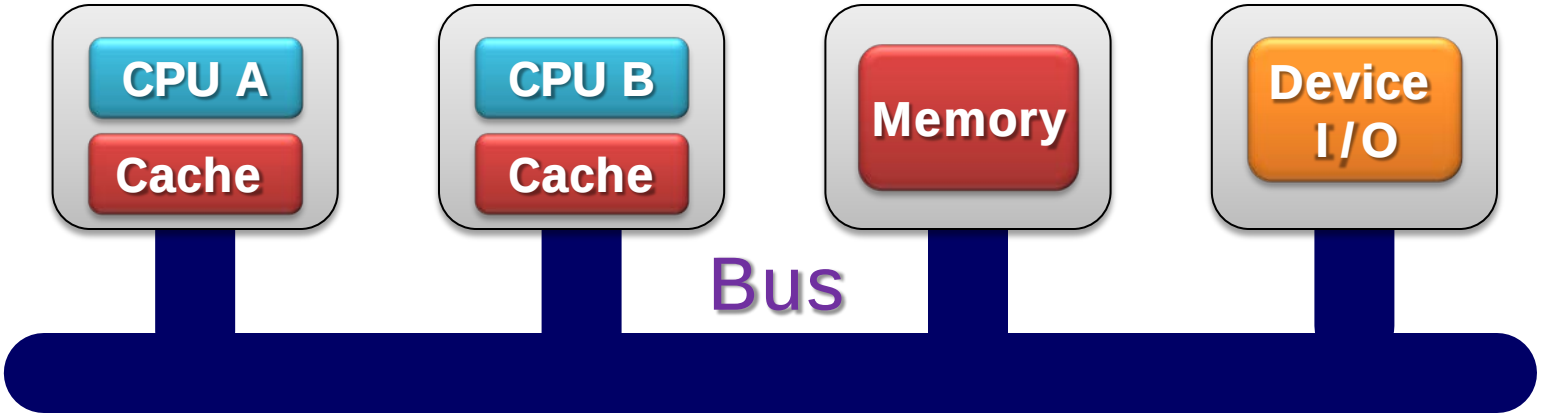
Bus-based Multiprocessors

How do we deal with bus overload & memory contention?

- Add local memory (cache)

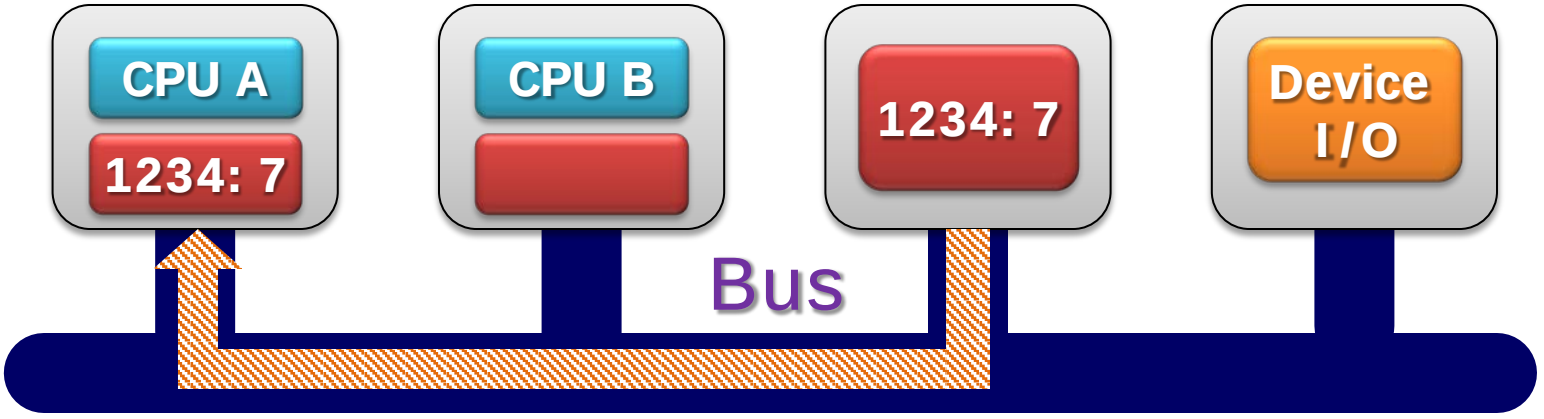
CPU reads/writes cache memory

- Access main memory only on cache miss



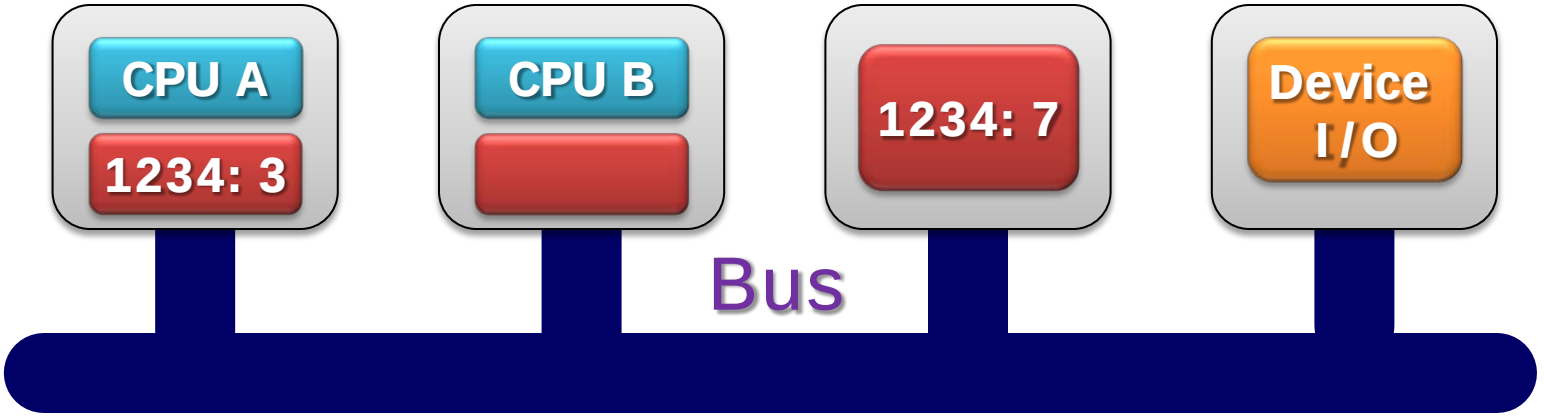
Working with a Cache

CPU A reads location 1234 from memory



Working with a Cache

CPU A modifies location 1234

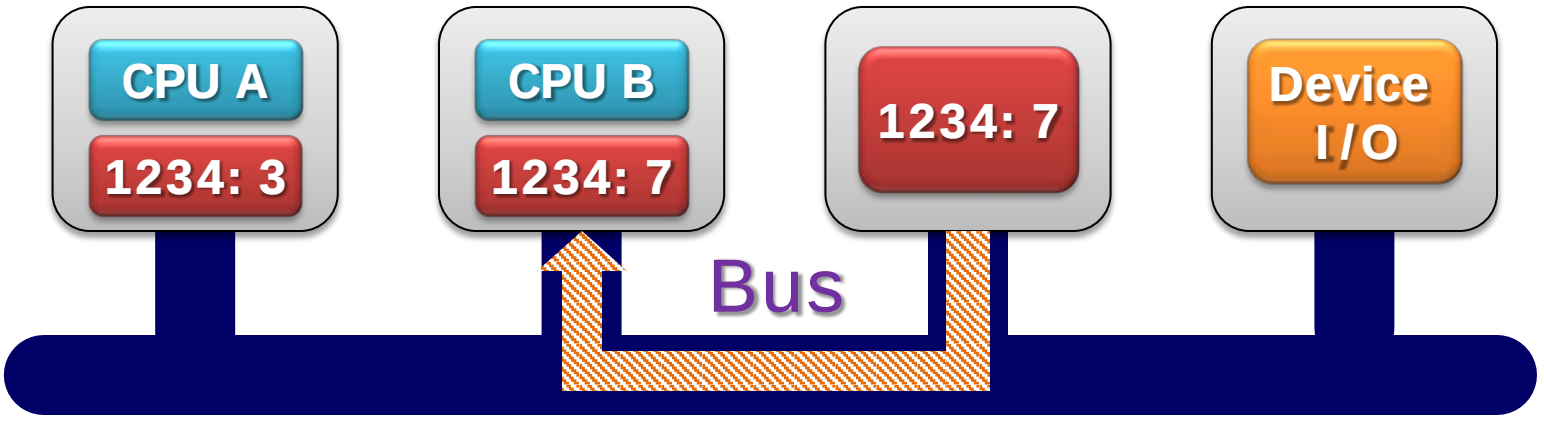


Working with a Cache

CPU B reads location 1234 from memory

Gets old value

Memory not coherent!

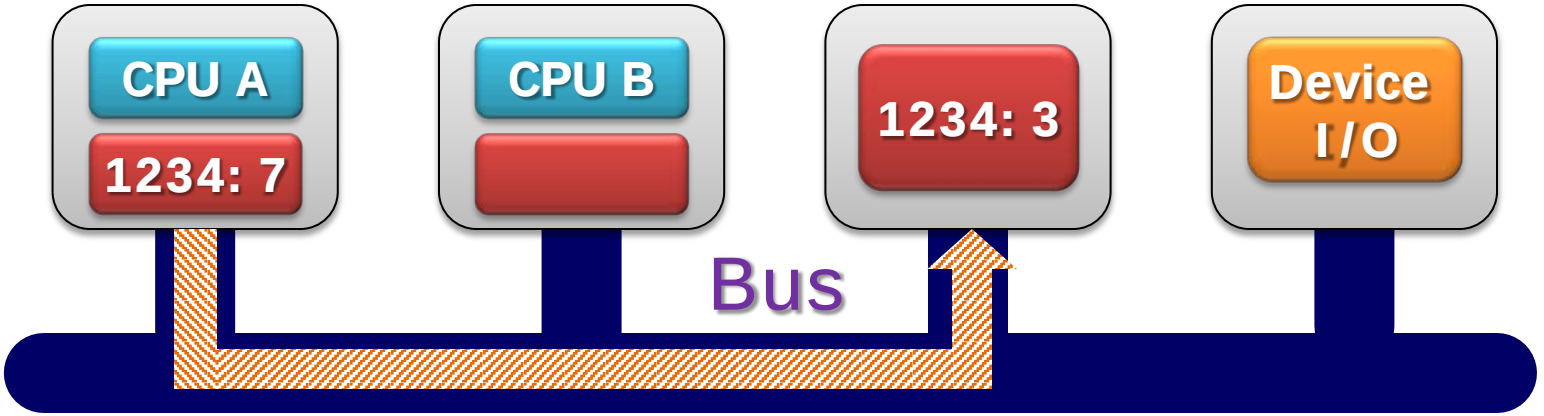


Working with a Cache

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

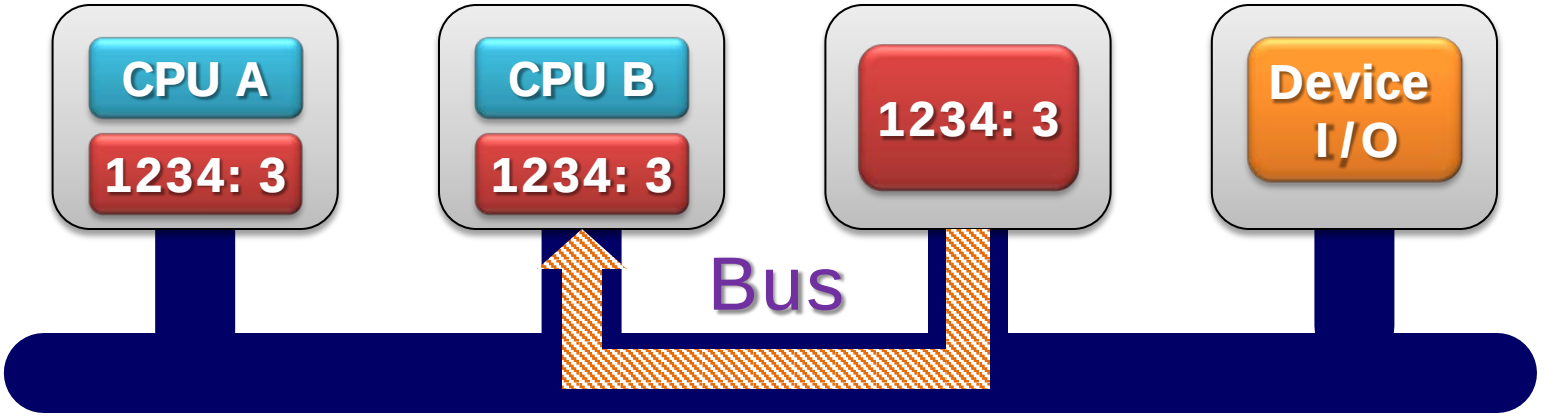
Write-through Cache

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



Working with a Cache

CPU B reads location 1234 from memory

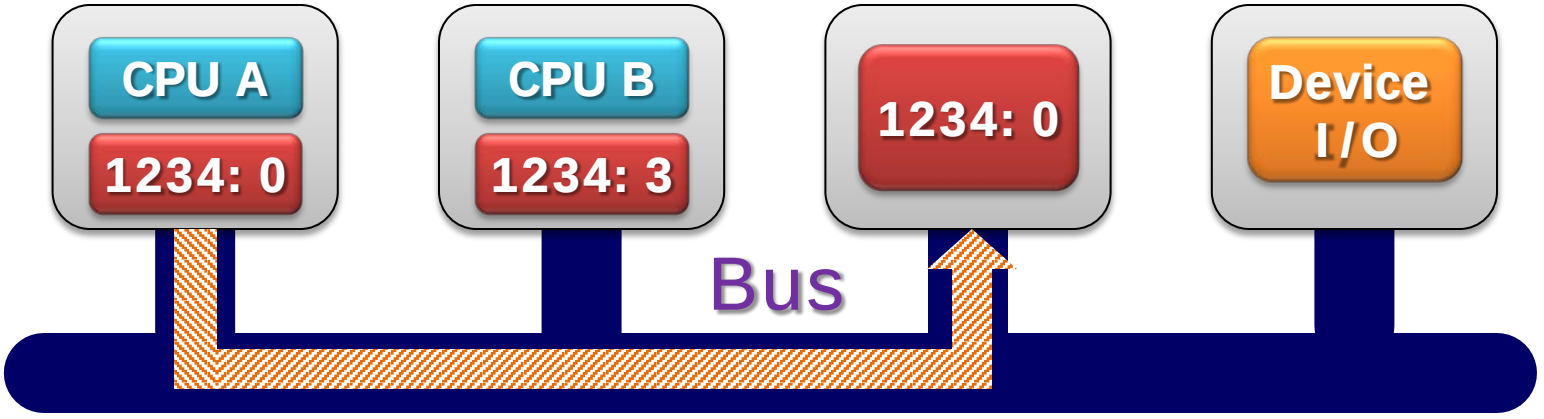


Working with a Cache

CPU A modifies location 1234 again

Cache on CPU B not updated

Memory not coherent!

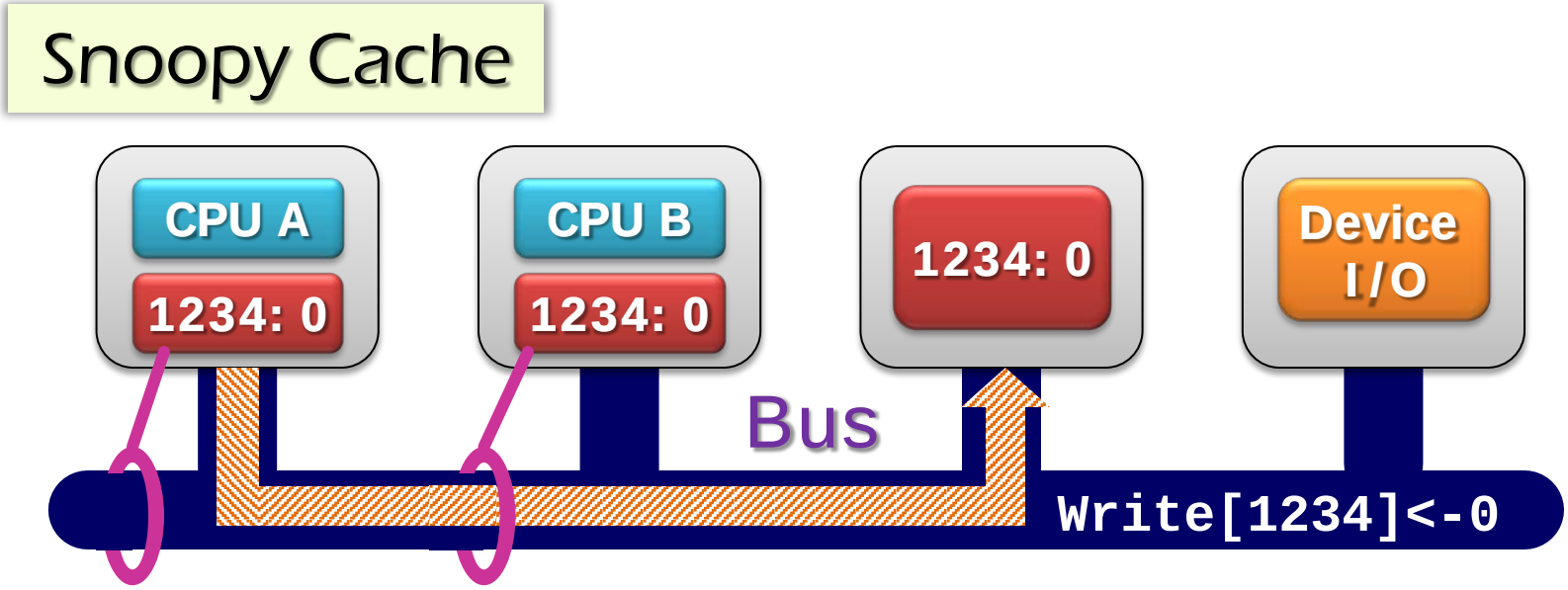


Working with a Cache

Add logic to each cache controller:

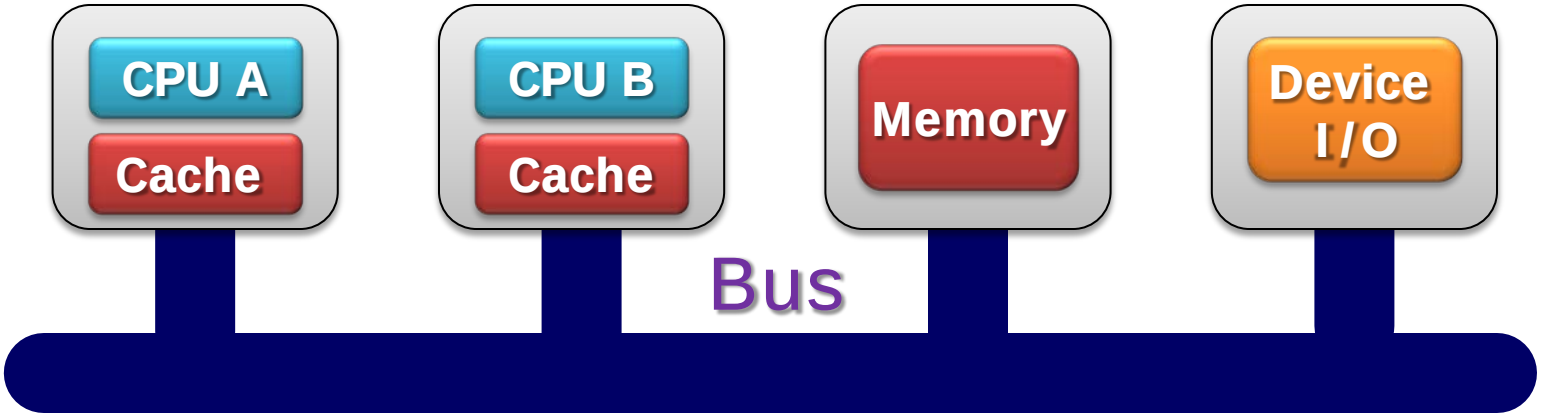
Monitor the bus for writes to memory

Virtually all bus-based architectures use a snoopy cache

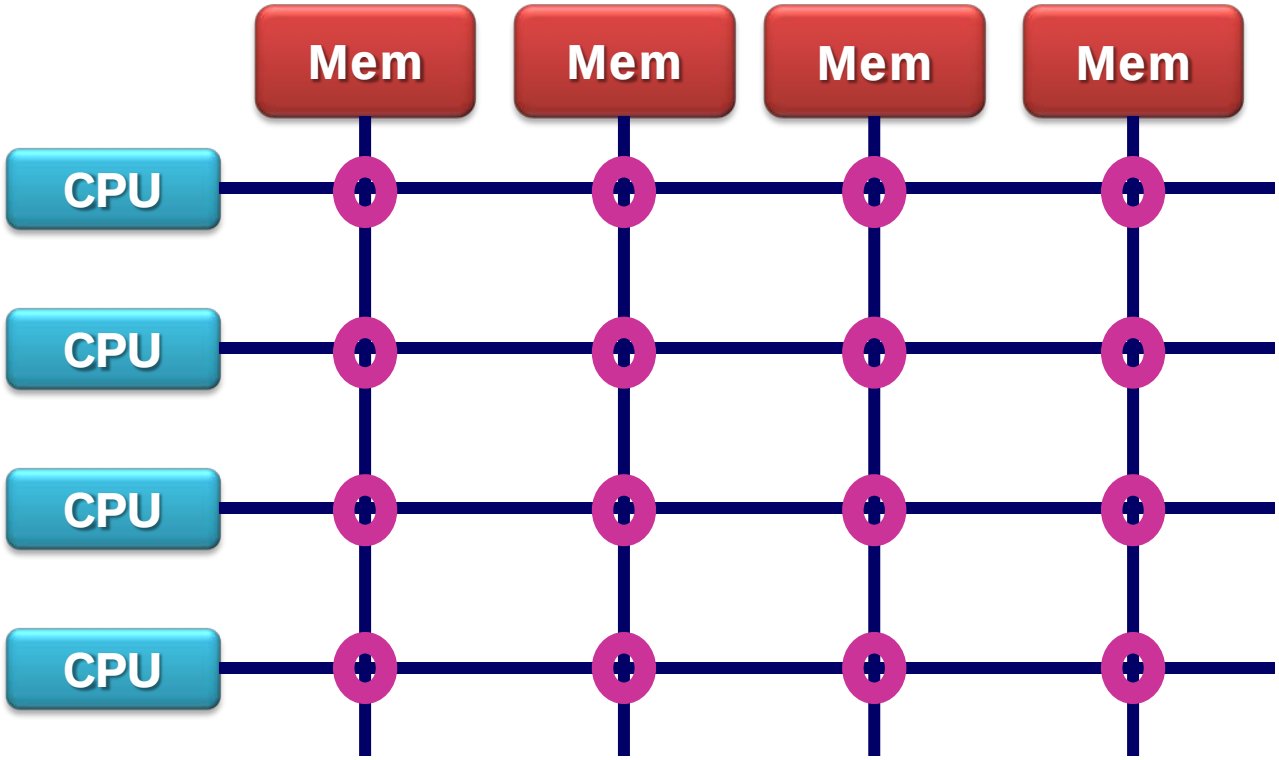


Switched multiprocessors

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 and data becomes difficult

- Operating system 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 (UMA) 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

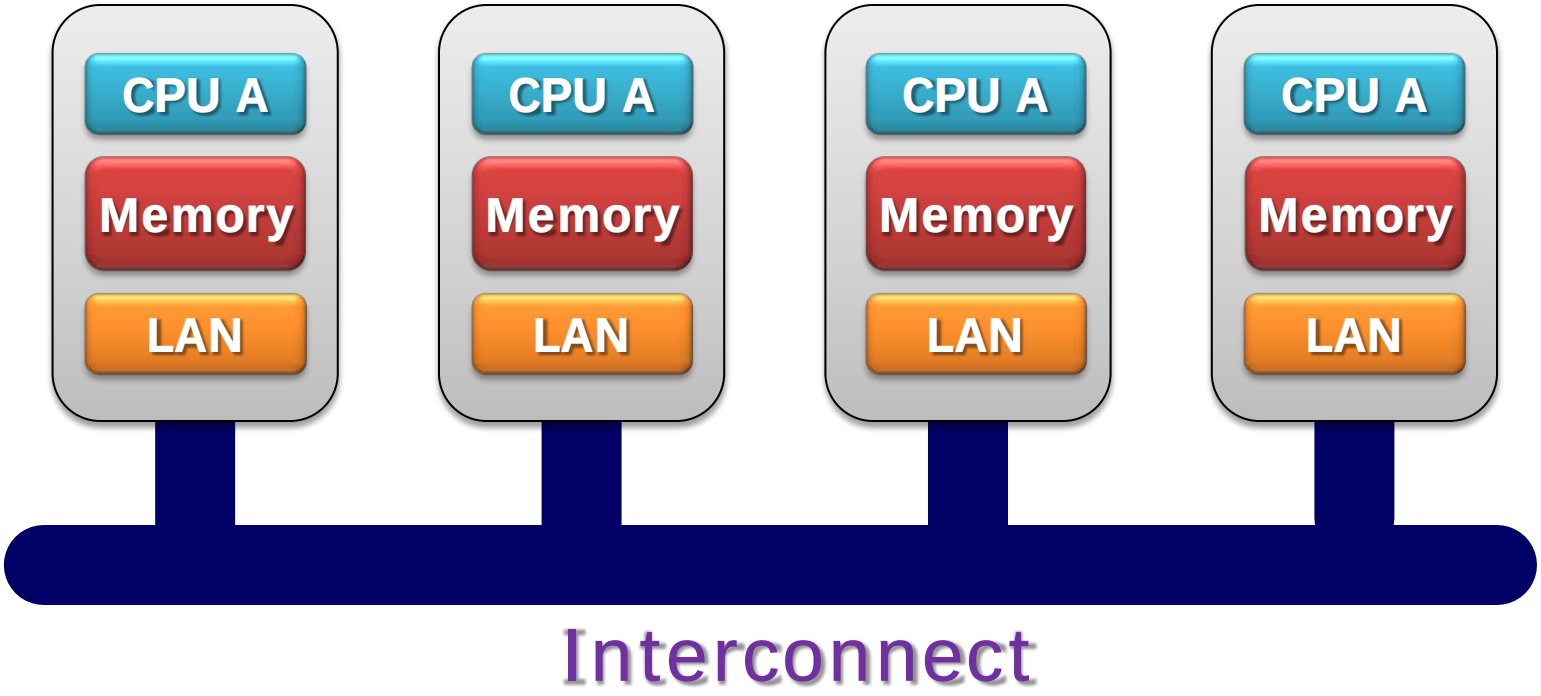
No shared clock

No shared memory

- Alternate communication mechanism needed
- Use network interconnect

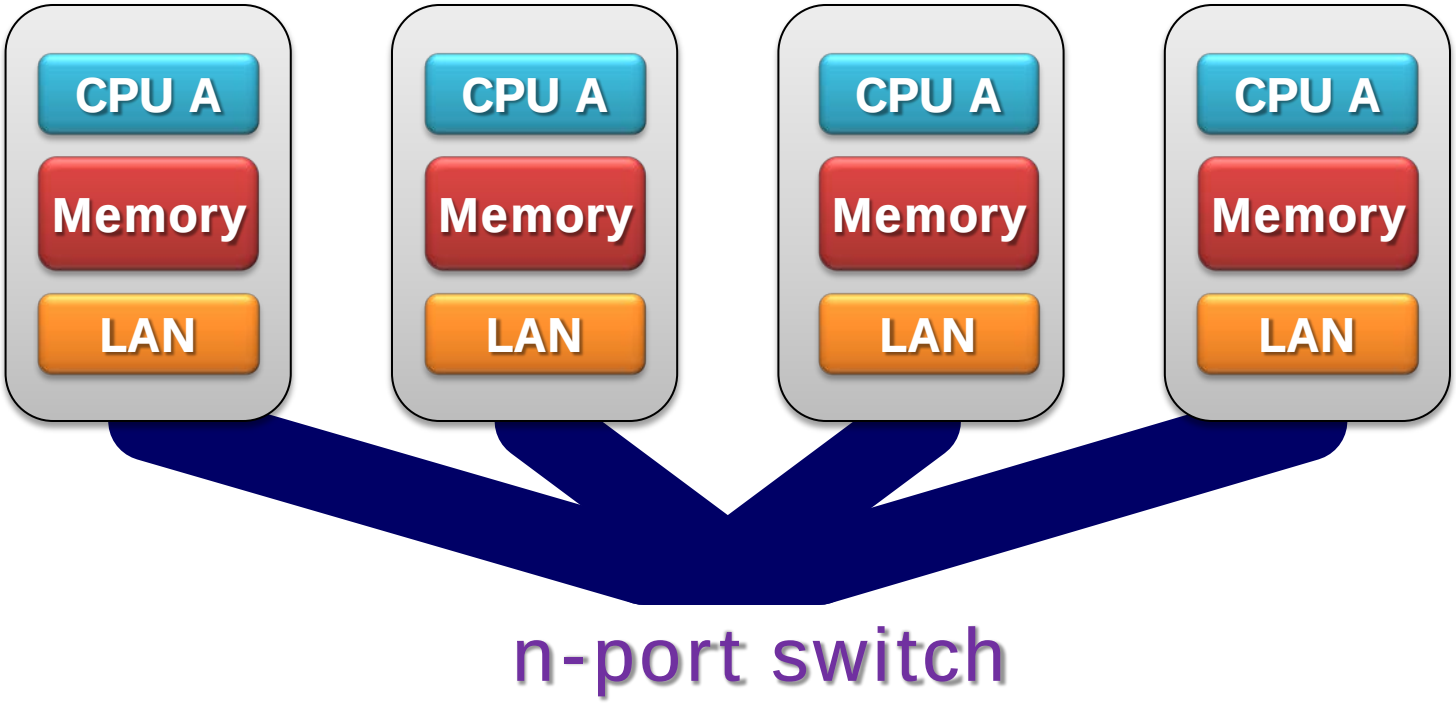
Bus-based Multicomputers

Collection of workstations on a LAN



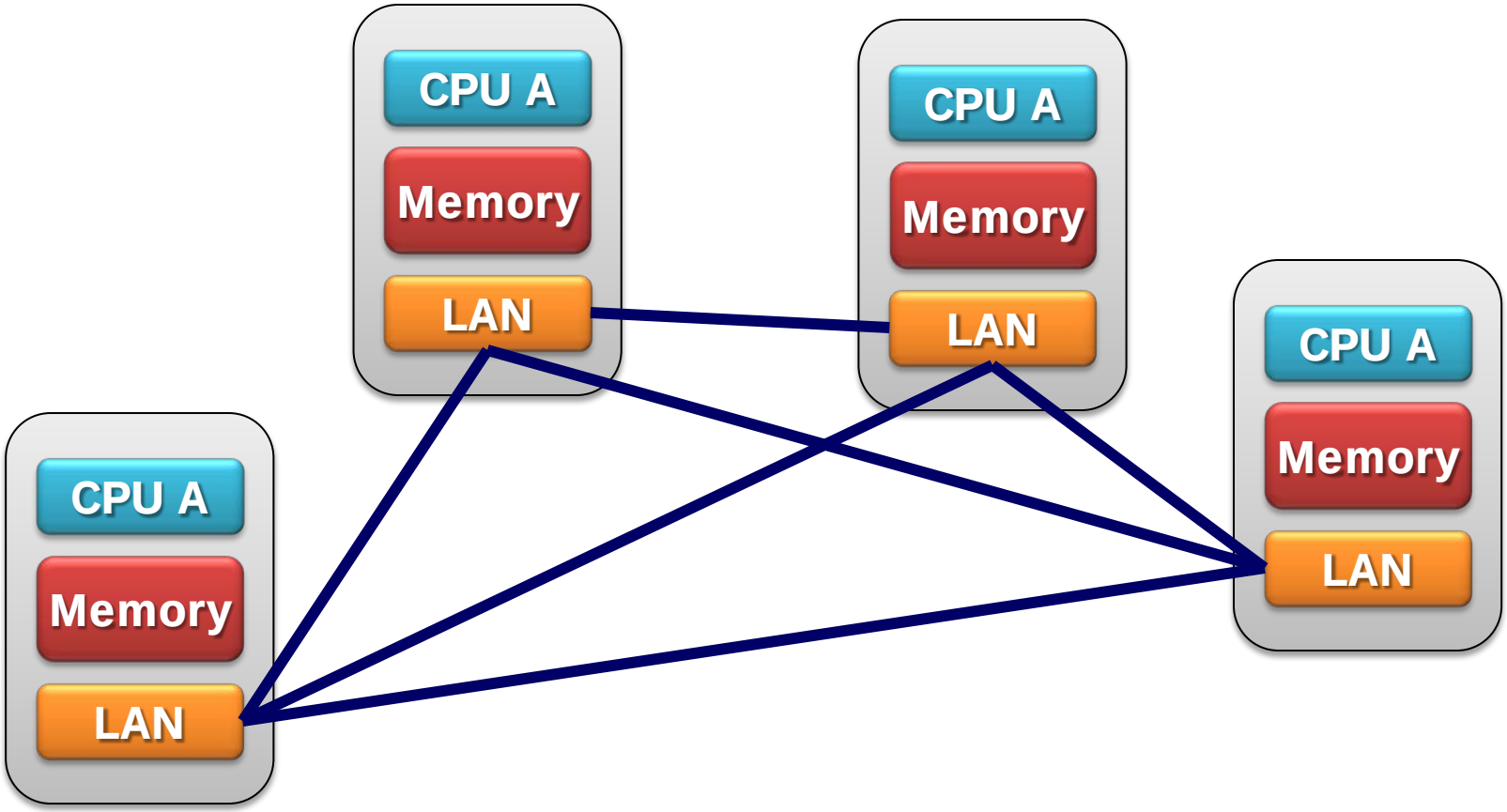
Switched Multicomputers

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

Distributed Software Service Models

Multiprocessor/Multicore OS

Multiple CPUs: UMA/NUMA

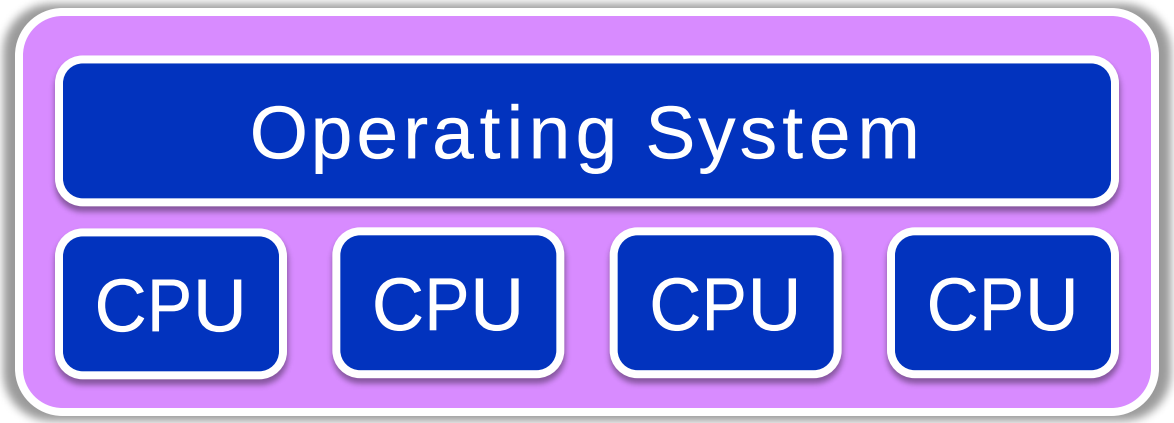
#Processors transparent to programs

Single OS and system image

Same communication mechanisms

- *Semaphores, shared memory, messages*

Networking not needed: we have shared memory



Network OS

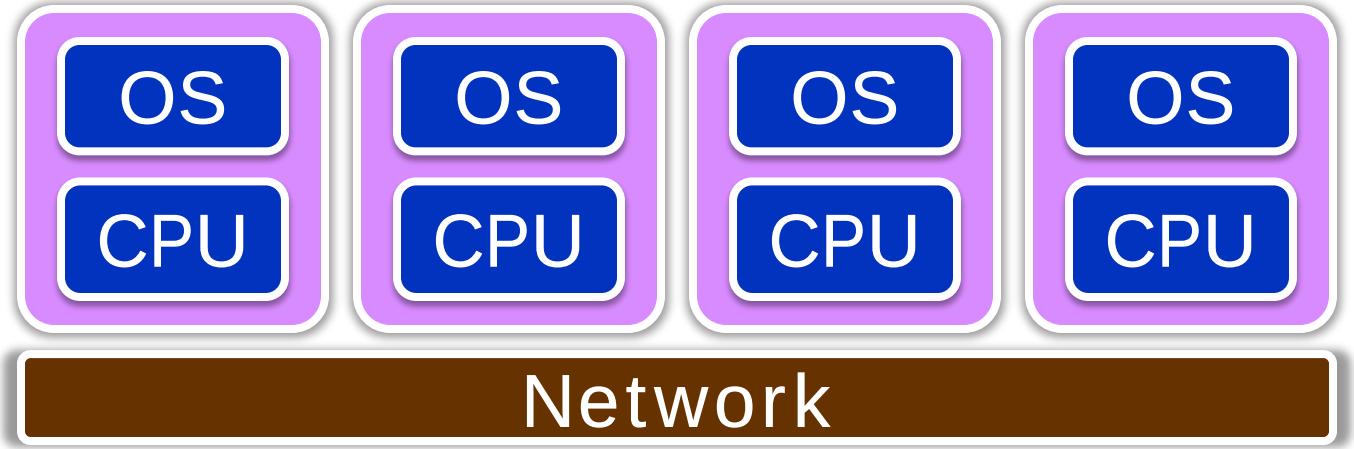
Autonomous nodes communicating via a network

Configurations may differ

Not a single system image

Distribution is explicit

Examples: Linux, Windows, OS X

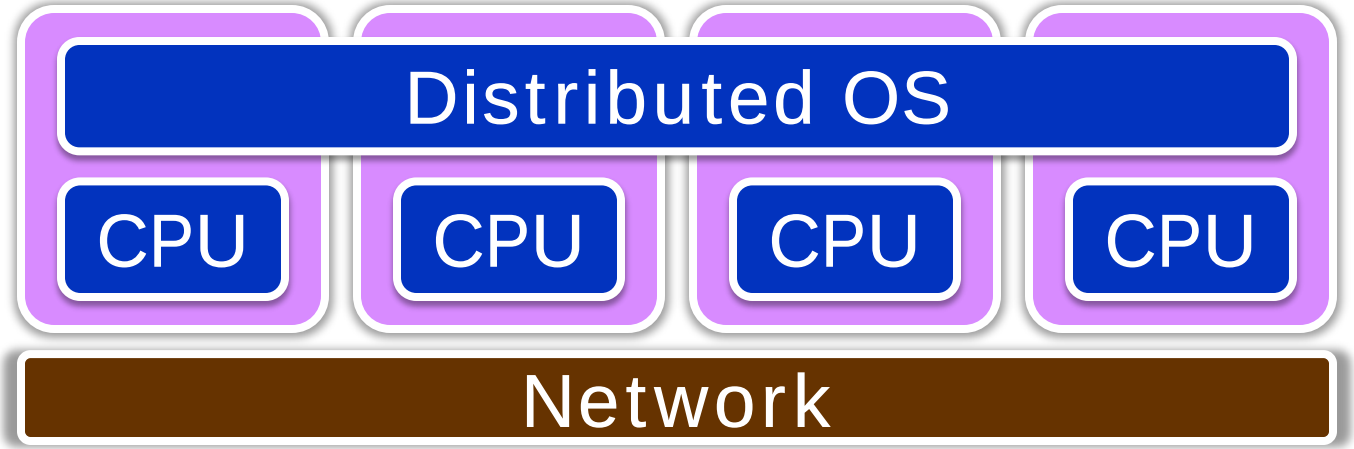


Distributed OS

Operating systems are aware of multiple systems and coordinate resource sharing

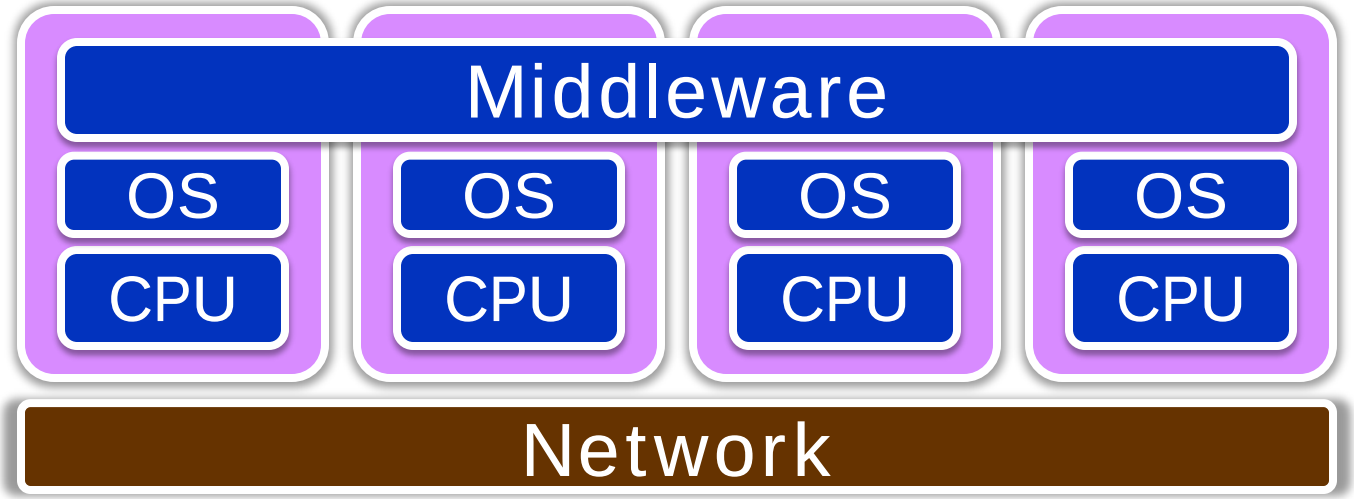
Single system image for file systems, processes, devices

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
- ~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 distributed 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
- Reliability: 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

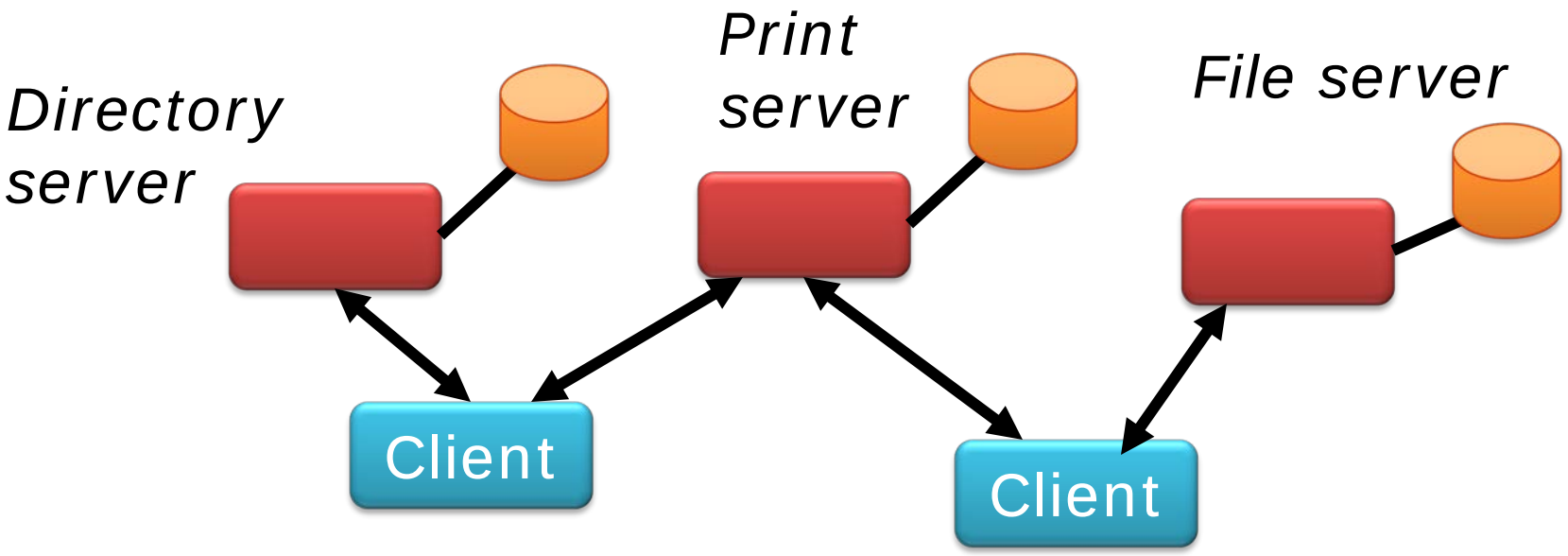
Limiting factor: #CPUs in system

- *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

- Storage capacity
- Processing
- Network bandwidth

Heterogeneous and geographically distributed systems

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

Cloud Computing Model

Resources are provided as a network (Internet) service

- Software as a Service
- Google Apps
- Salesforce.com
- Amazon Web Services

Multi-tier client-server architecture

Two-tier Architecture

Common from mid 1980's - early 1990's

- UI on user's desktop
- Application services on server

Three-tier Architecture

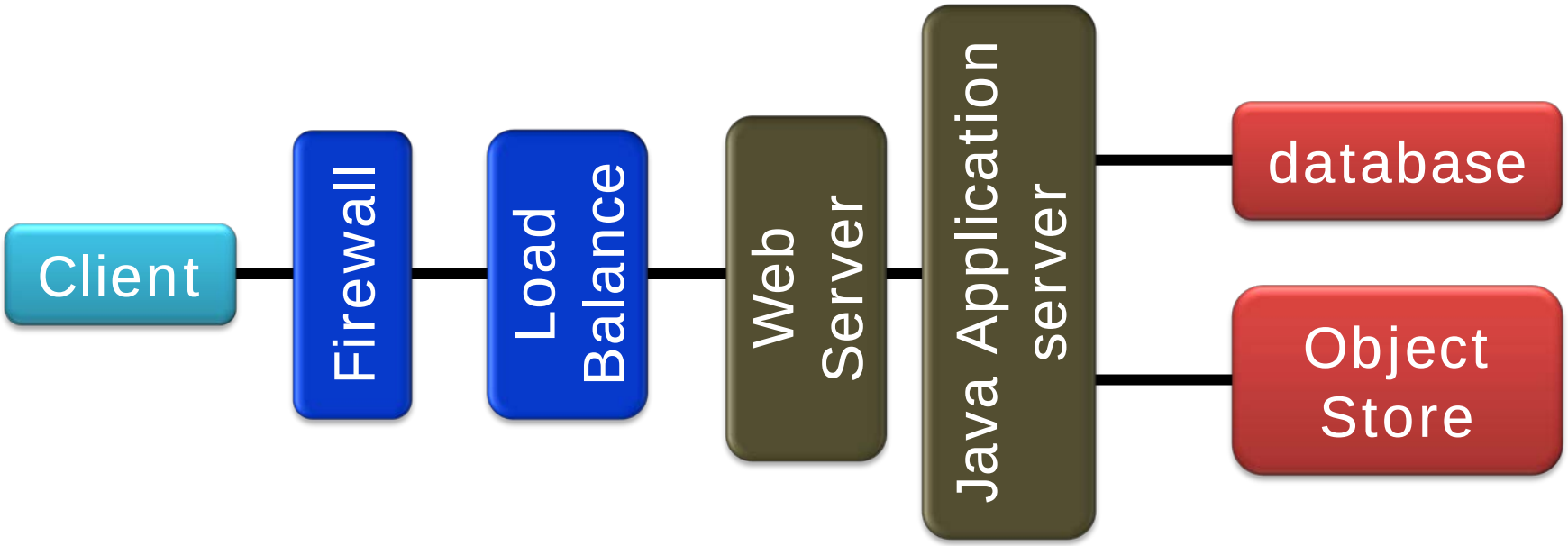
- User interface
- Data validation/ formatting

- Queuing/scheduling of user requests
- Transaction processor (TP)
- Connection mgmt.
- Format conversion

- Database
- Legacy application prcoessing

Beyond Three-tiers

Most architecture are multi-tiered



Consistency

Networking

Consensus

File Systems

Transaction

Concurrency

Topics in this courses

Fault Tolerance

Scalability

Storage

Recovery

Programming Model

Course Schedule

Preliminary

- 2.28 Intro
- 3.07 RPC & RMI
- 3.14 Distributed Programming *
- 3.21 Consistency: Sequential & Release
- 3.28 Consistency: Eventual *
- 4.04 *Tomb-sweeping Day*
- 4.11 Invitation: multicore scalability
- 4.18 Recovery *
- 4.25 Concurrency *

Course Schedule

Common from mid 1980's - early 1990's

- 5.02 Two-phase commit *
- 5.09 Consensus *
- 5.16 Complex data processing *
- 5.23 DFS *
- 5.30 NoSQL *
- 6.06 BFT *
- 6.13 Review

Next Lecture

Topic: Threads & Remote Procedure Calls

Paper

Andrew Birrell, Bruce Nelson, ***"Implementing Remote Procedure Calls"***. ACM Transactions on Computer Systems (TOCS), 2(1), **1984**

THANKS