Chapter 2
Principles of Distributed Computing

Mastering Cloud Computing

Coleman Kane
(based on materials from Paul Talaga)

UNIVERSITY OF Cincinnati

Elements of Distributed Computing

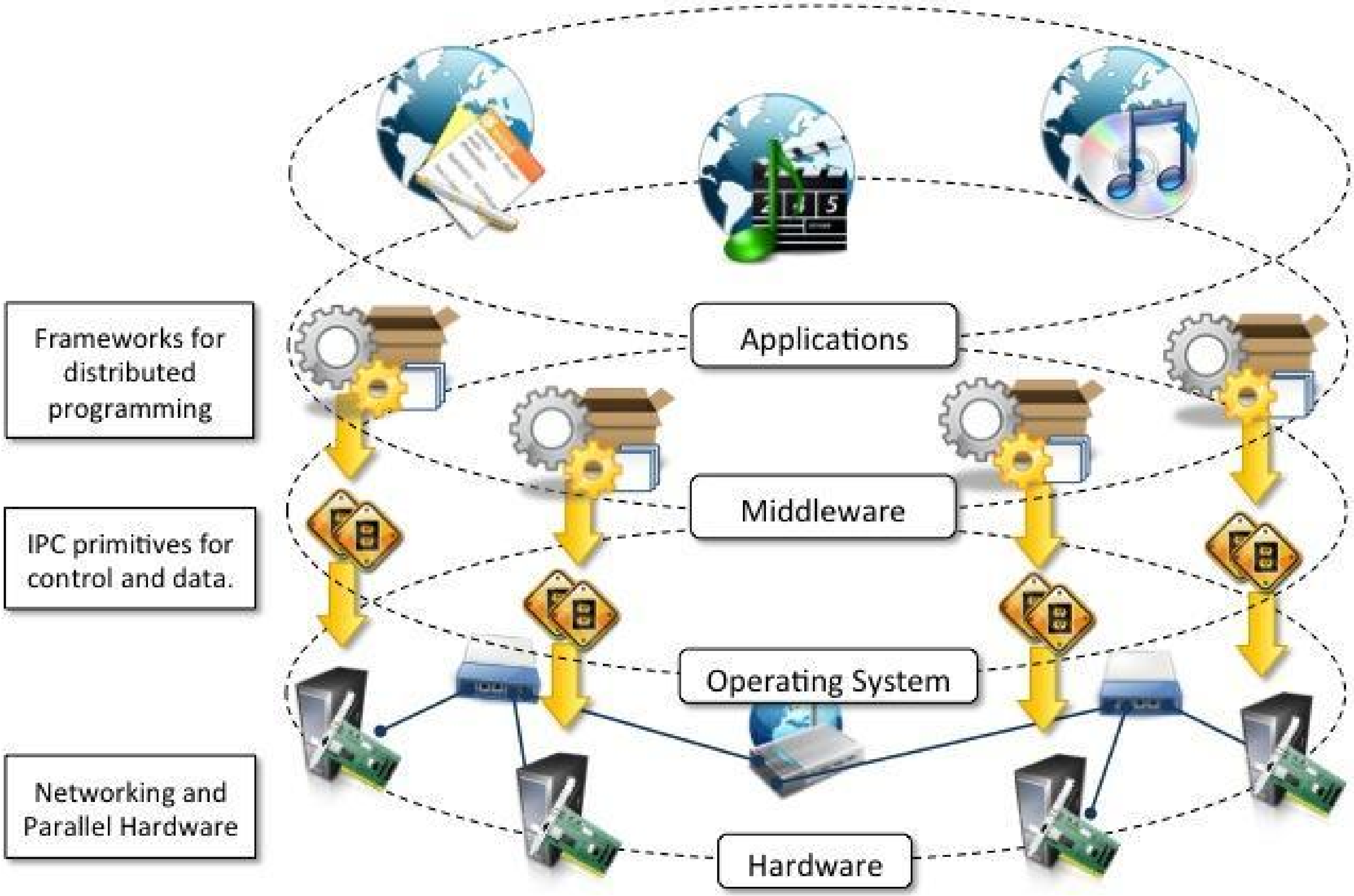
A distributed system is a collection of independent computers that appears to its users as a single coherent system - Tanenbaum et al

A distributed system is one in which components located at networked computers communicate and coordinate their actions by passing messages. - Coulouris et al

Message passing!!!!

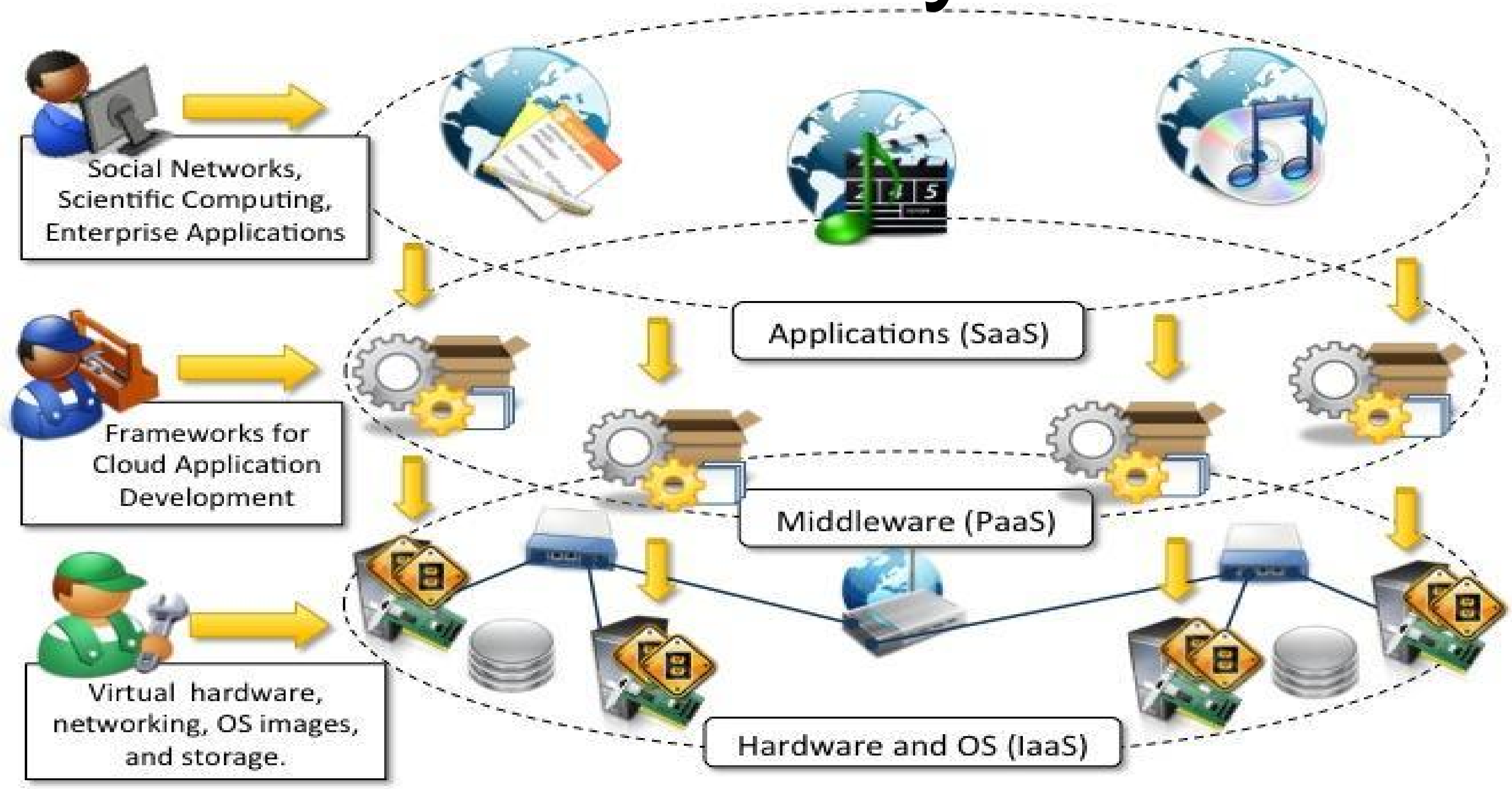
UNIVERSITY OF Cincinnati

Distributed System Layers



Abstraction!

Cloud Computing as a distributed system



Middleware enables distributed computing

UNIVERSITY OF Cincinnati

Architectural Styles for Distributed Computing

Architectural styles are mainly used to determine the vocabulary of components and connectors that are used as instances of the style together with a set of constraints on how they can be combined.

Like design patterns for a program, but for an entire software system.

- Software architectural styles (software organization)
- System architectural styles (physical organization)

UNIVERSITY OF Cincinnati

Components and Connectors

- Component - software that encapsulates a function or feature of the system: programs, objects, processes.
- Connector - communication mechanism between components, can be implemented in a distributed manner.

UNIVERSITY OF Cincinnati

Software Architectural Styles

Category	Common Architectural Styles
Data-centered	Repository
	Blackboard
Data flow	Pipe and filter
	Batch sequential
Virtual Machine	Rule-based system
	Interpreter
Call and return	Main program and subroutine/top down
	Object-oriented systems
Independent components	Communicating processes
	Even systems

Data centered architectures

- Data is core, and access to shared data
  - Data integrity is goal
  - Ex: Gmail, Flickr, Google search
- *Repository style -*
  - central data structure - current state
  - independent components - operate on data
  - 2 subtypes:
    - database systems - components called & act on data
    - blackboard systems - data-structure is trigger - if/then or expert-system feel - updates itself

UNIVERSITY OF Cincinnati

Software Architectural Styles

Category	**Common Architectural Styles**
Data-centered	Repository
	Blackboard
Data flow	Pipe and filter
	Batch sequential
Virtual Machine	Rule-based system
	Interpreter
Call and return	Main program and subroutine/top down
	Object-oriented systems
Independent components	Communicating processes
	Even systems

UNIVERSITY OF Cincinnati

Data-Flow architectures

- Availability of data controls, data flows through system
- For when data size exceeds storage capacities, or when long-term storage is not needed
- 2 styles:
  - Batch Sequential - sequence of programs - must wait for previous to finish before next
  - Pipe-and-Filter - sequence of programs, but FIFO queues to start processing before previous has finished.
  - Unix shell pipes and tools are good examples:
    - grep, sed, awk

UNIVERSITY OF Cincinnati

Batch Sequential vs. Pipe-and-Filter

Batch Sequential	**Pipe-and-Filter**
Coarse grained	Fine grained
High latency	reduced latency due to incremental processing
External access to input	localized input
No concurrency	Concurrency possible
Noninteractive	Awkward, but possible

UNIVERSITY OF Cincinnati

Software Architectural Styles

Category	Common Architectural Styles
Data-centered	Repository
	Blackboard
Data flow	Pipe and filter
	Batch sequential
Virtual Machine	Rule-based system
	Interpreter
Call and return	Main program and subroutine/top down
	Object-oriented systems
Independent components	Communicating processes
	Event systems

UNIVERSITY OF Cincinnati

Virtual Machine architectures

- Abstract execution environment - rule-based systems, interpreters, command-language processors. 2 types:
- **Rule Based:**
  - Inference engine - AI - process control - network intrusion detection
  - Examples includes some of the Predix IoT analytics systems - https://github.com/PredixDev/predix-analytics-sample
- **Interpreter:**
  - Interprets pseudo-program - abstracts hardware differences away - Java, C#, Perl, PHP
  - Examples include the "*try it*" features in the Python/Java tutorials I gave you

UNIVERSITY OF Cincinnati

Software Architectural Styles

Category	Common Architectural Styles
Data-centered	Repository
	Blackboard
Data flow	Pipe and filter
	Batch sequential
Virtual Machine	Rule-based system
	Interpreter
Call and return	Main program and subroutine/top down
	Object-oriented systems
Independent components	Communicating processes
	Event systems

UNIVERSITY OF Cincinnati

Call & Return architectures

- Components connected via method calls
- 3 styles:
  - **Top-Down**: imperative programming - tree structure - hard to maintain
  - **Object-Oriented**: coupling between data and manipulation operations - easier to maintain - method calling requires object - consistency an issue
  - **Layered Style**: Abstraction layers - modular design - hard to change layers
    - Ex: OS kernels, TCP/IP stack, web applications

UNIVERSITY OF Cincinnati

Software Architectural Styles

Category	Common Architectural Styles
Data-centered	Repository
	Blackboard
Data flow	Pipe and filter
	Batch sequential
Virtual Machine	Rule-based system
	Interpreter
Call and return	Main program and subroutine/top down
	Object-oriented systems
Independent components	Communicating processes
	Event systems

UNIVERSITY OF Cincinnati

Independent Components Architectures

- Life cycles to components
- 2 styles:
  - Communicating Processes: good for distributed systems - concurrent - service based - IPC
  - Event Systems: components have data and manipulation, but add event registering/triggering - callbacks - like layered, but looser connections - hard to reason about correctness of interactions

Apple's ObjectiveC/C++ is a good example

UNIVERSITY OF Cincinnati

System Architectural Styles

- Describe physical layout
- 2 styles:
  - Client/Server
  - Peer-to-peer

UNIVERSITY OF Cincinnati

Client/Server

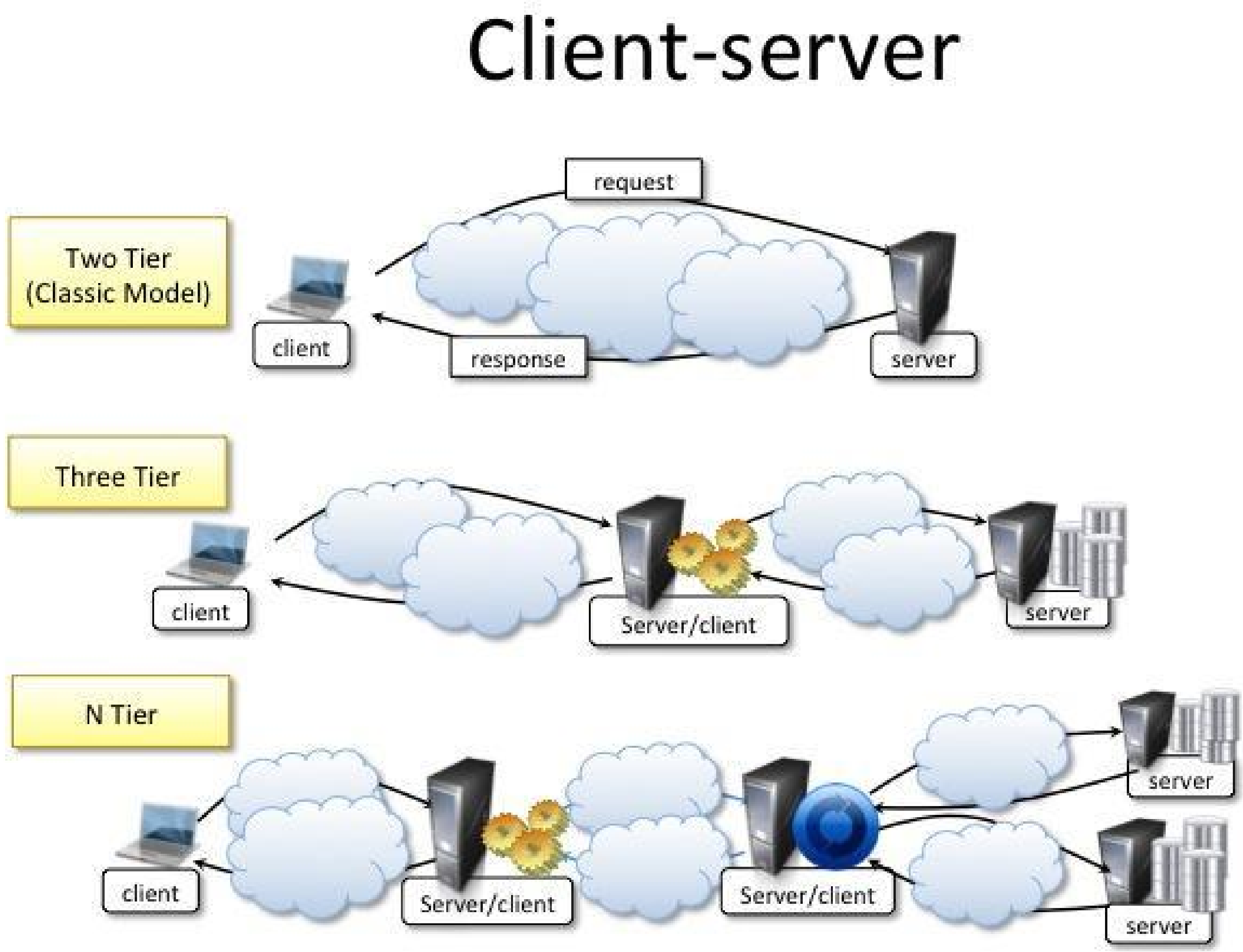
- Very Popular
- *request, accept (client)*
- *listen, response (server)*
- Suitable for many-to-one situations

Types of Clients

- Thin-client: (nearly) all data processing done on server (plain HTML)
  - presentation on client
  - app logic and data storage on server
- Fat-client: client processes and transforms data, server just gateway to access data (AJAX-like)
  - presentation and app logic on client
  - data storage on server

UNIVERSITY OF Cincinnati

Layered Approach



Layer Types

- Two Tier
  - Pros: Easier to build
  - Con: Doesn’t scale well
  - Ex: Small dynamic web applications
- Three Tier/N Tier
  - Pros: Scales better (add more servers to layer)
  - Con: Harder to maintain
  - Ex: Medium-Large dynamic web applications

Peer-to-Peer

- Symmetric - everyone client and server
- Scales very well!
- Hard to build
- Used in data centers to distribute data!
- Ex: Gnutella, BitTorrent, Kazaa, Skype, Tor
- Multi-level roles possible: Kazaa

UNIVERSITY OF Cincinnati