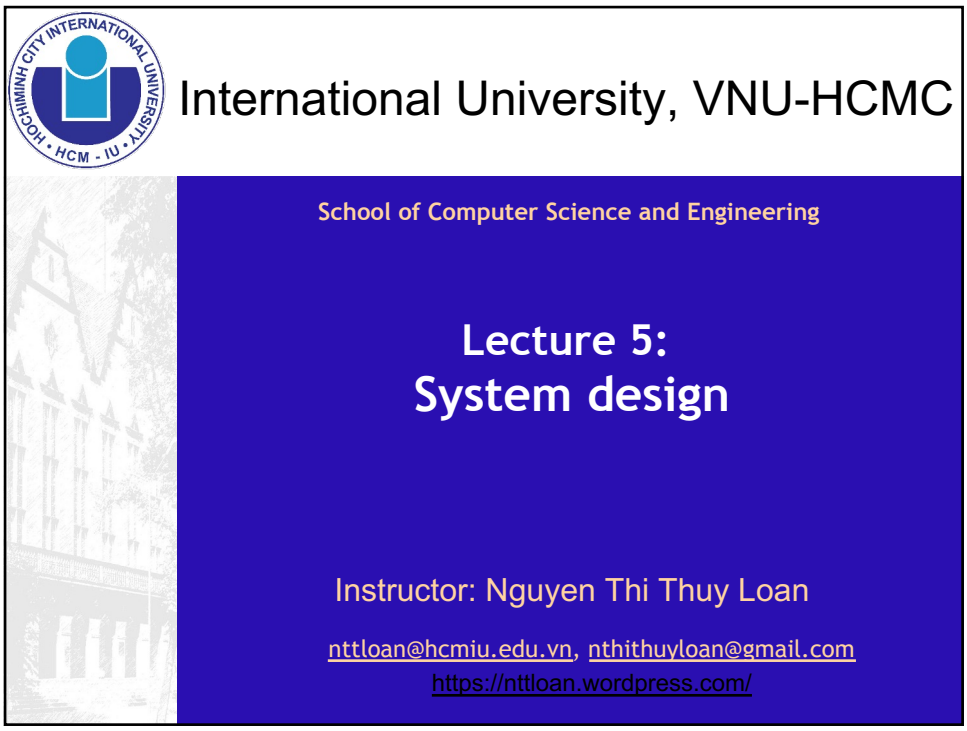
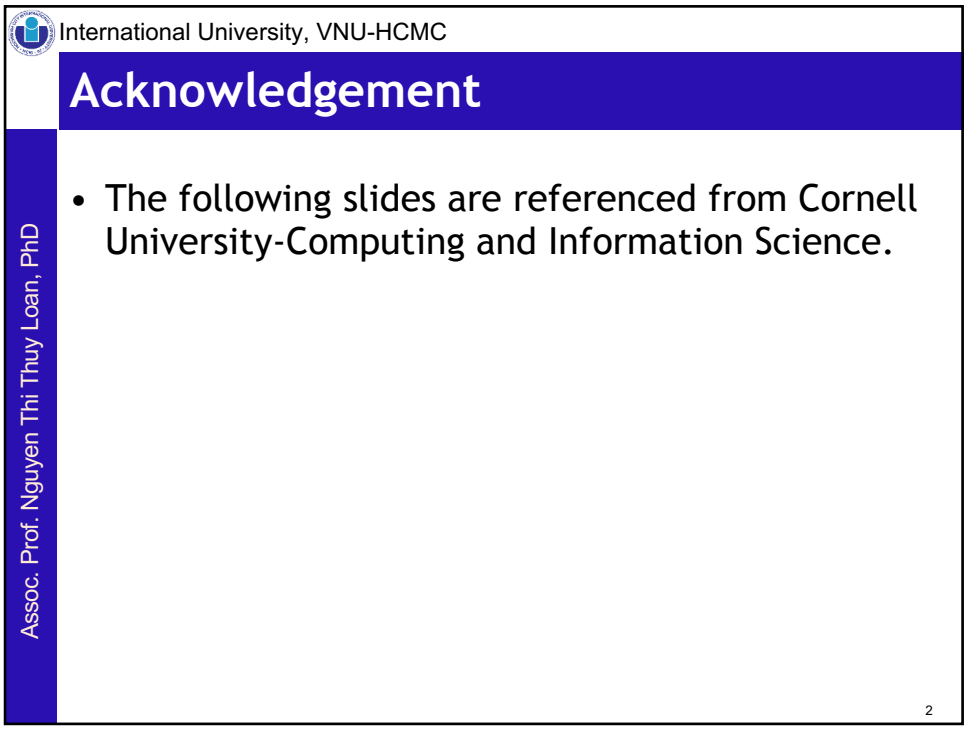

International University, VNU-HCMC
School of Computer Science and Engineering
Lecture 5:
System design
Instructor: Nguyen Thi Thuy Loan
nttloan@hcmiu.edu.vn, nthithuyloan@gmail.com
https://nttloan.wordpress.com/


International University, VNU-HCMC
Acknowledgement
Assoc. Prof. Nguyen Thi Thuy Loan, PhD
• The following slides are referenced from Cornell University-Computing and Information Science.
2

Outlines

- System architecture
- Three popular architectural styles
- Security
- Performance

Design

The requirements describe the function of a system as seen by the client.

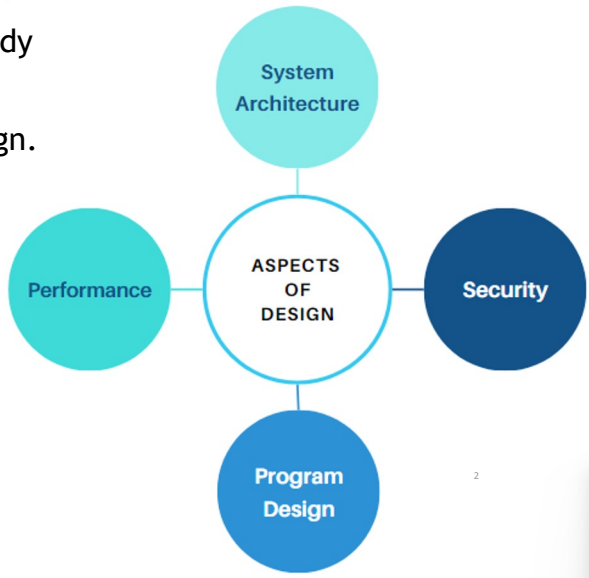
For a given set of requirements, the software development team must design a system that will meet those requirements.

In practice requirements and design are interrelated. In particular, working on the design often clarifies the requirements. This feedback is a strength of the iterative and agile methods of software development.

Design

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

We have already looked at user interface design.



Creativity and Design

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

Software development

Software development is a craft. Software developers have a variety of tools that can be applied in different situations.

Part of the art of software development is to select the appropriate tool for a given implementation.

Creativity and Design

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

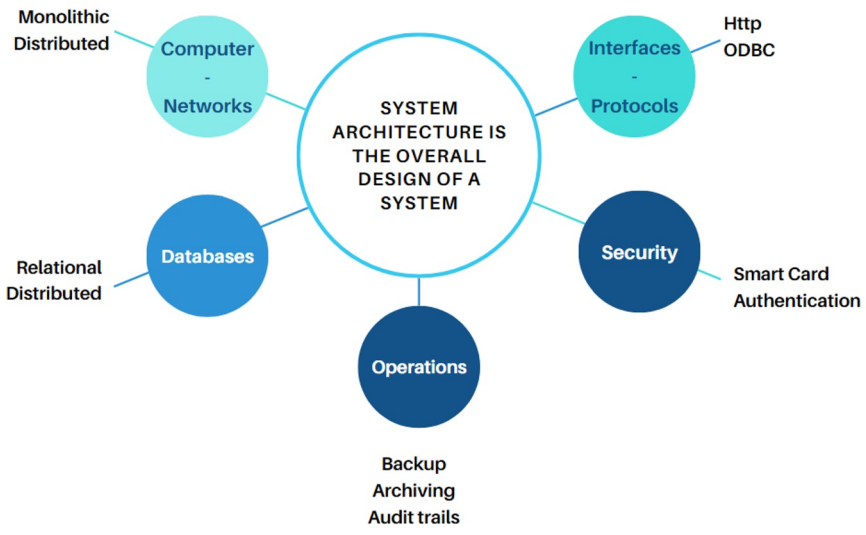
Creativity and design

System and program design are a particularly creative part of software development, as are user interfaces. You hope that people will describe your designs as "elegant", "easy to implement, test, and maintain."

Above all strive for simplicity. The aim is find simple ways to implement complex requirements.

System Architecture

Assoc. Prof. Nguyen Thi Thuy Loan, PhD



System Architecture

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

Software environments

Testing frameworks

Models for System Architecture

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

Our models for systems architecture are based on UML

The slides provide diagrams that give an outline of the systems, without the supporting specifications.

For every system, there is a choice of models

Choose the models that best model the system and are clearest to everybody.

When developing a system, every diagram must have supporting specification.

The diagrams shows the relationships among parts of the system, but much, much more detail is needed to specify a system explicitly.

Subsystems

A subsystem is a grouping of elements that form part of a system.

- **Coupling** is a measure of the dependencies **between two subsystems**. If two systems are strongly coupled, it is hard to modify one without modifying the other.
- **Cohesion** is a measure of dependencies **within a subsystem**. If a subsystem contains many closely related functions its cohesion is high.

An ideal division of a complex system into subsystems has low coupling between subsystems and high cohesion within subsystems.

Component

A **component** is a replaceable part of a system that conforms to and provides the realization of a set of interfaces.

A component can be thought of as an implementation of a subsystem.

UML definition of a component

"A distributable piece of implementation of a system, including software code (source, binary, or executable), but also including business documents, etc., in a human system."

Components as Replaceable Elements

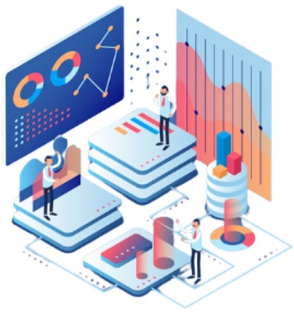
Assoc. Prof. Nguyen Thi Thuy Loan, PhD

COMPONENTS ALLOW SYSTEMS TO BE ASSEMBLED FROM BINARY REPLACEABLE ELEMENTS

- A component is bits not concepts
- A component can be replaced by any other component(s) that conforms to the interfaces
- A component is part of a system
- A component provides the realization of a set of interfaces

Components and Classes

Assoc. Prof. Nguyen Thi Thuy Loan, PhD



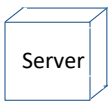
Classes represent logical abstractions. They have attributes (data) and operations (methods).

Components have operations that are reachable only through interfaces.

Package

A **package** is a general-purpose mechanism for organizing elements into groups.

Node



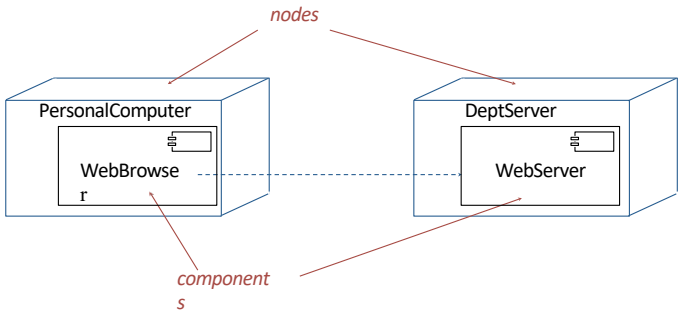
A node is a physical element that exists at run time and provides a computational resource, e.g., a computer, a smartphone, or a router.

Components may live on nodes.

Example: Simple Web System

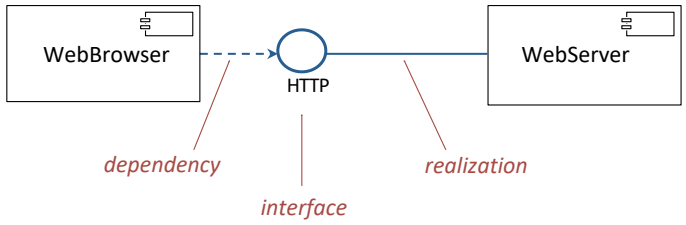
- Static pages from server
- All interaction requires communication with server

Deployment Diagram



Component Diagram: Interfaces

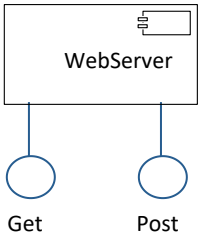
Assoc. Prof. Nguyen Thi Thuy Loan, PhD



Application Programming Interface (API)

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

An API is an interface that is realized by one or more components.



Architectural Styles

An **architectural style** is system architecture that recurs in many different applications.

See:

- https://www.castsoftware.com/glossary/what-is-software-architecture-tools-design-definition-explanation-best
- Mary Shaw and David Garlan, *Software architecture: perspectives on an emerging discipline*. Prentice Hall, 1996
- David Garlan and Mary Shaw, An Introduction to Software Architecture. Carnegie Mellon University, 1994 http://www.cs.cmu.edu/afs/cs/project/able/ftp/intro_softarch/intro_softarch.pdf

Architectural Style: Pipe

Example: A three-pass compiler

Output from one subsystem is the input to the next.

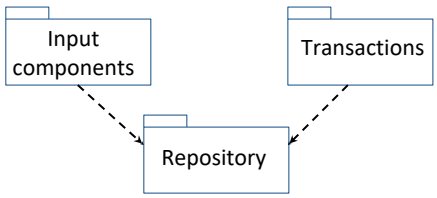
Architectural Style: Client/Server

Example: A mail system

The control flows in the client and the server are independent.
Communication between client and server follows a protocol.

In a peer-to-peer architecture, the same component acts as both a client and a server.

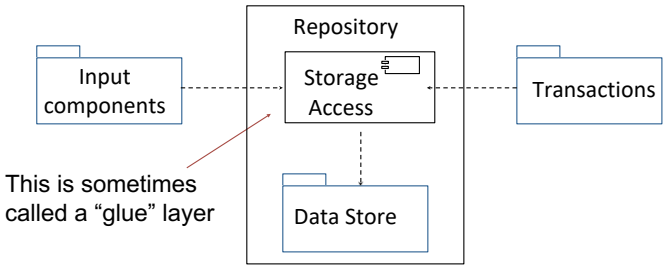
Architectural Style: Repository



Advantages: Flexible architecture for data-intensive systems.
Disadvantages: Difficult to modify repository since all other components are coupled to it.

Architectural Style: Repository with Storage Access Layer

Assoc. Prof. Nguyen Thi Thuy Loan, PhD



Advantages: Data Store subsystem can be changed without modifying any component except the Storage Access.

29

Time-Critical Systems

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

A time-critical (real time) system is a software system whose correct functioning depends upon the results produced and the time at which they are produced.

A hard real time system fails if the results are not produced within required time constraints

e.g., a fly-by-wire control system for an airplane must respond within specified time limits

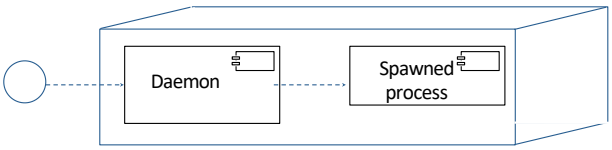
A soft real time system is degraded if the results are not produced within required time constraints

e.g., a network router is permitted to time out or lose a packet

31

Time Critical System: Architectural Style-Daemon

A **daemon** is used when messages might arrive at closer intervals than the time to process them.



Example: Web server
The daemon listens at port 80
When a message arrives it:
spawns a processes to handle the message returns to listening at port 80

Architectural Styles for Distributed Data

Replication:
Several copies of the data are held in different locations.
Mirror: Complete data set is replicated
Cache: Dynamic set of data is replicated (e.g., most recently used)
With replicated data, the biggest problems are concurrency and consistency.
Example: The Domain Name System
For details of the protocol read:
Paul Mockapetris, "Domain Names - Implementation and Specification". IETF Network Working Group, *Request for Comments*: 1035, November 1987.

An Exam Question

A company that makes sports equipment decides to create a system for selling sports equipment online. The company already has a product database with description, marketing information, and prices of the equipment that it manufactures.

To sell equipment online the company will need to create: a customer database, and an ordering system for online customers.

The plan is to develop the system in two phases. During Phase 1, simple versions of the customer database and ordering system will be brought into production. In Phase 2, major enhancements will be made to these components.

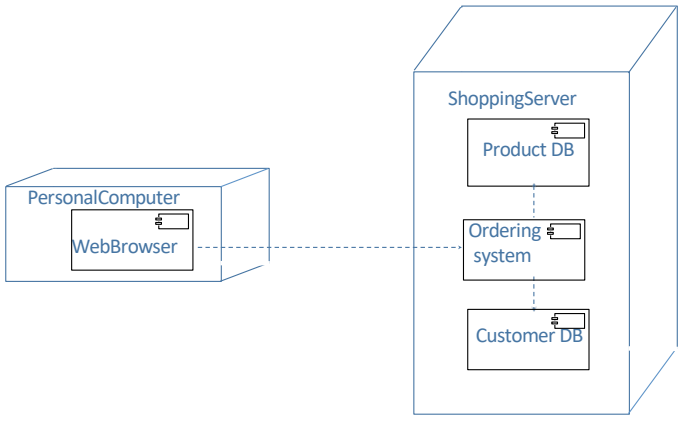
International University, VNU-HCMC

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

An Exam Question

a) For the system architecture of Phase 1:

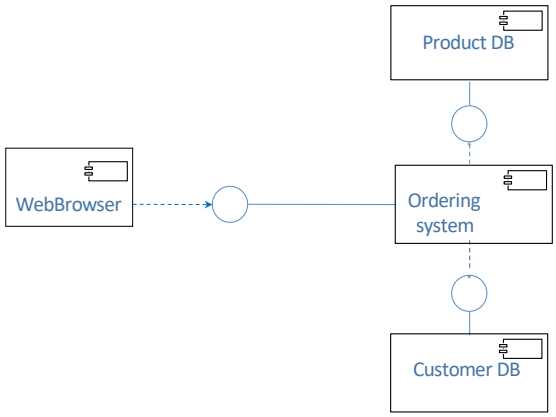
a.i) Draw a UML deployment diagram.



An Exam Question

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

(a) For the system architecture of Phase 1:
a.ii). Draw a UML interface diagram.



An Exam Question

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

(b) For Phase 1:

b.i). What architectural style would you use for the customer database?

Repository with Storage Access Layer

b.ii). Why would you choose this style?

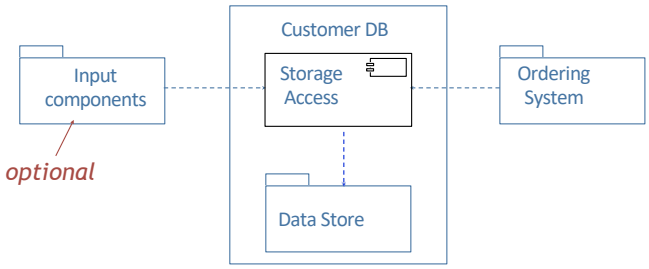
It allows the database to be replaced without changing the applications that use the database.

An Exam Question

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

(b) For Phase 1:

b.iii). Draw an UML diagram for this architectural style showing its use in this application.



Outlines

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

- System architecture
- Three popular architectural styles
- Security
- Performance

Three-Tier Architecture

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

What is three-tier architecture and describe?

Benefits of three-tier architecture

Three-tier application in web development

Example 1:

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

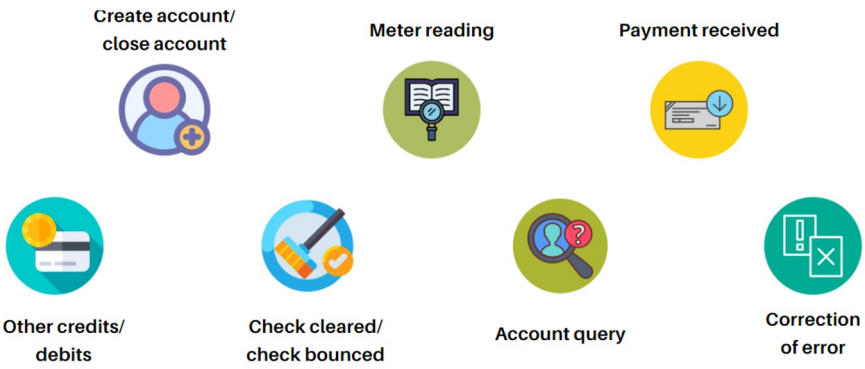
Batch Processing with Master File Update

- Electricity utility customer billing (e.g., NYSEG)
- Telephone call recording and billing (e.g., Verizon)
- Car rental reservations (e.g., Hertz)
- Bank (e.g., Tompkins Trust)
- University grade registration (e.g., IU)

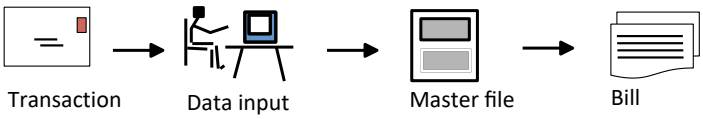
Master File Update

Example: Electricity Utility Billing

Requirements analysis identifies several transaction types:



First Attempt

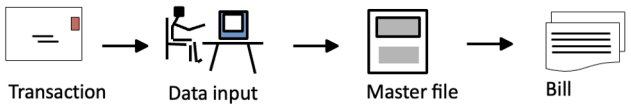


Each transaction is handled as it arrives.

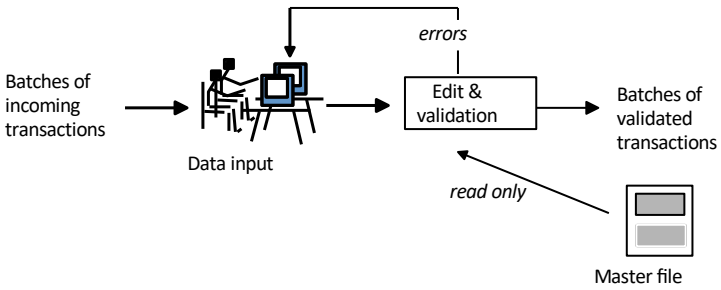
Criticisms of First Attempt

Where is this first attempt weak?

- All activities are triggered by a transaction.
- A bill is sent out for each transaction, even if there are several per day.
- Bills are not sent out on a monthly cycle.
- Awkward to answer customer queries.
- No process for error checking and correction.
- Inefficient in staff time.

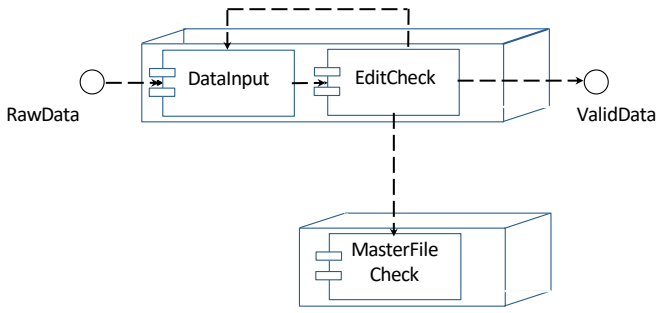


Batch Processing: Edit and Validation



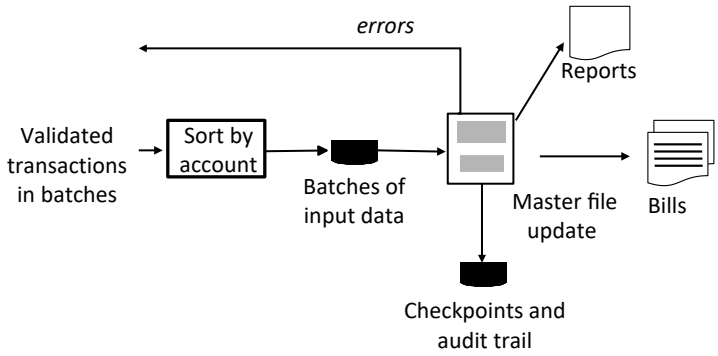
Deployment Diagram: Validation

Assoc. Prof. Nguyen Thi Thuy Loan, PhD



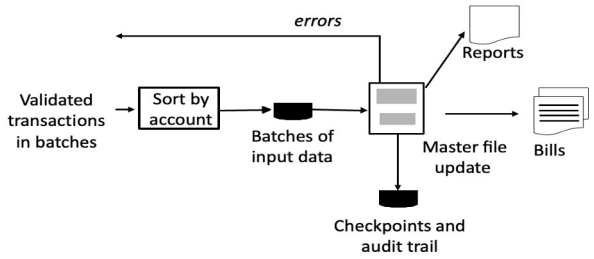
Batch Processing: Master File Update-MFU

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

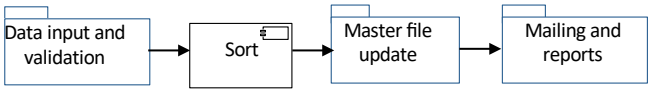


Benefits of Batch Processing with MFU

- All transactions for an account are processed together at appropriate intervals, e.g., monthly.
- Backup and recovery have fixed checkpoints.
- Better management control of operations.
- Efficient use of staff and hardware.
- Error detection and correction is simplified.



Architectural Style: MFU (Basic Version)



Advantages:

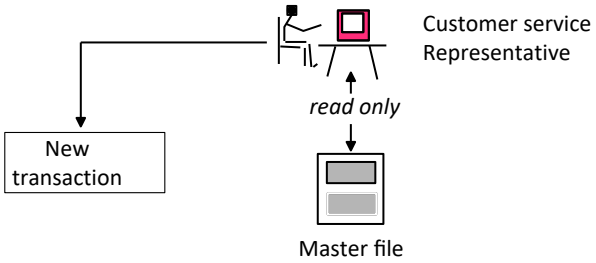
Efficient way to process batches of transactions.

Disadvantages:

Information in master file is not updated immediately. No good way to answer customer inquiries.

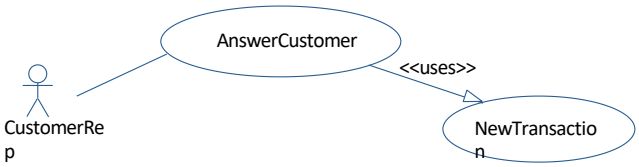
Online Inquiry

A customer calls the utility and speaks to a customer service representative.



Customer service department can read the master file, make annotations, and create transactions, but cannot change the master file.

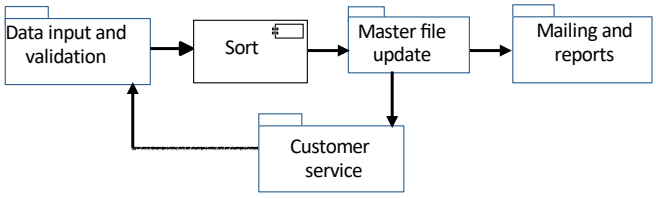
Online Inquiry: Use Case



The representative can read the master file, but not make changes to it.

If the representative wishes to change information in the master file, a new transaction is created as input to the master file update system.

Architectural Style: Master File Update (Full)



Advantage:

Efficient way to answer customer inquiries.

Disadvantage:

Information in master file is not updated immediately.

Example 2: Three Tier Architecture-TTA

The basic client/server architecture of the web has:

- a server that delivers static pages in HTML format
- a client (known as a browser) that renders HTML pages

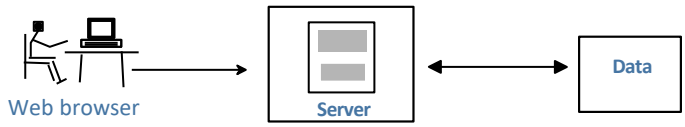
Both client and server implement the HTTP interface.

Problem

Extend the architecture of the server so that it can configure HTML pages dynamically.

Web Server with Data Store

Assoc. Prof. Nguyen Thi Thuy Loan, PhD



Advantage:

Server-side code can configure pages, access data, validate information, etc.

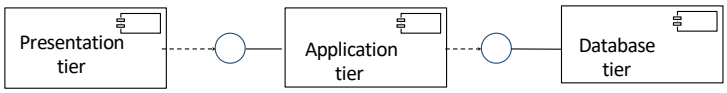
Disadvantage:

All interaction requires communication with server

55

Architectural Style: TTA

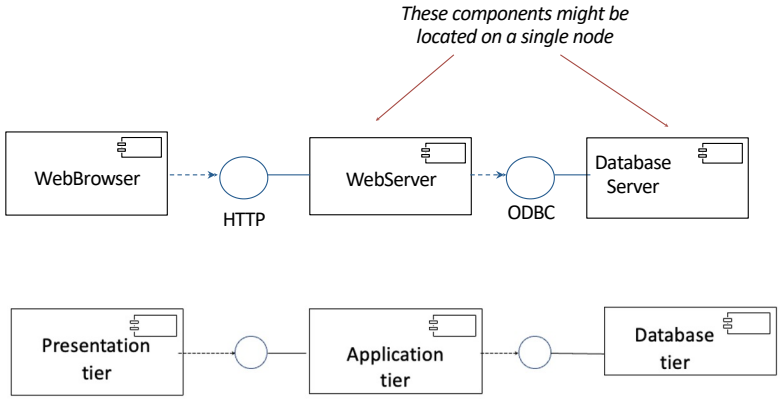
Assoc. Prof. Nguyen Thi Thuy Loan, PhD



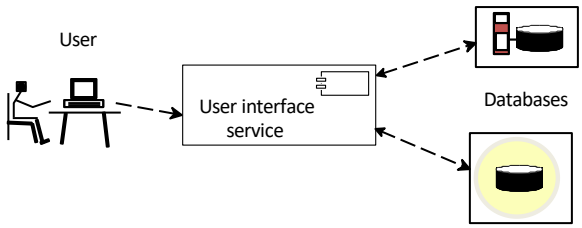
Each of the tiers can be replaced by other components that implement the same interfaces

56

Component Diagram



TTA: Broadcast Searching



This is an example of a multicast protocol.
The primary difficulty is to avoid troubles at one site degrading the entire system (e.g., every transaction cannot wait for a system to time out).

Extending the Architecture of the Web

Using a three tier architecture, the web has:

- a server that delivers dynamic pages in HTML format
- a client (known as a browser) that renders HTML pages

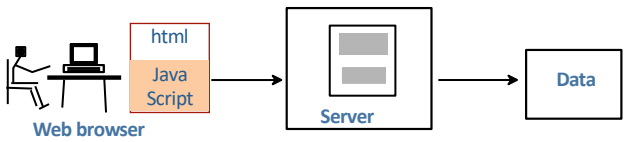
Both server and client implement the HTTP interface.

Every interaction with the user requires communication between the client and the server.

Problem 2

Extend the architecture so that simple user interactions do not need messages to be passed between the client and the server.

Extending the Web with Executable Code that can be Downloaded



Executable code in a scripting language such as JavaScript can be downloaded from the server

Advantage:

Scripts can interact with user and process information locally.

Disadvantage:

All interactions are constrained by web protocols.

Extending the Three Tier Architecture

In the three-tier architecture, a website has:

- a client that renders HTML pages and executes scripts
- a server that delivers dynamic pages in HTML format
- a data store

Further extensions

- The three-tier architecture with downloadable scripts is one of how the basic architecture has been extended. There are some more:
- Protocols: e.g., HTTPS, FTP, proxies
- Data types: e.g., helper applications, plug-ins
- Executable code: e.g., applets, servlets
- Style sheets: e.g., CSS

Example 3: Model/View/Controller (MVC)

The definition of Model/View/Controller (MVC) is in a state of flux. The term is used to describe a range of architectures and designs.

- Some are system architectures, where the model, view, and controller are separate components.
- Some are program designs, with classes called model, view, and controller.

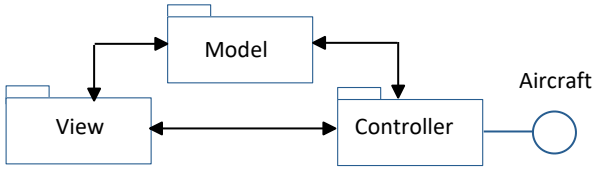
We will look at three variants:

- An MVC system architecture used in robotics.
- A general purpose MVC system architecture used for interactive systems.
- Apple's version of MVC as a program design for mobile apps.

Model/View/Controller in Robotics

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

Example: Control of an unmanned model aircraft



Controller: Receives instrument readings from the aircraft, updates the view, and sends controls signals to the aircraft.

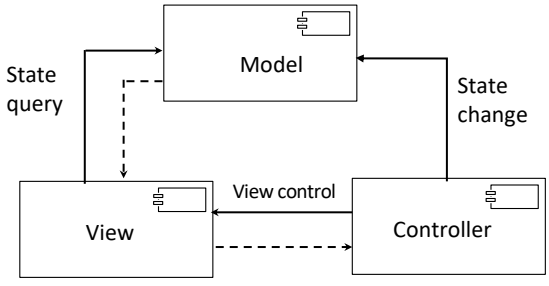
Model: Translates data received from and sent to the aircraft, and instructions from the user into a model of flight performance. Uses domain knowledge about the aircraft and flight.

View: Displays information about the aircraft to the user on the ground and transmits instructions to the model via the controller.

63

Example 3: MVC for Mobile Apps

Assoc. Prof. Nguyen Thi Thuy Loan, PhD



64

Model

The model records the state of the application and notifies subscribers. It does not depend on the controller or the view.

Stores the state of the application in suitable data structures or databases

Responds to instructions to change the state information

Notifies subscribers of events that change the state

May be responsible for validation of information

View

The view is the part of the user interface that presents the state of the interface to the user. It subscribes to the model, which notifies it of events that change the state.

- renders data from the model for the user interface
- provides editors for properties, such as text fields, etc.
- receives updates from the model
- sends user input to the controller

A given model may support a choice of alternative views.

Controller

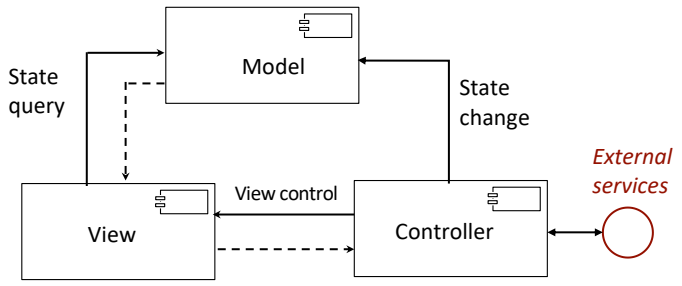
The controller is the part of the user interface that manages user input and navigation within the application.

- defines the application behavior
- maps user actions to changes in the state of the model
- interacts with external services via APIs
- may be responsible for validation of information

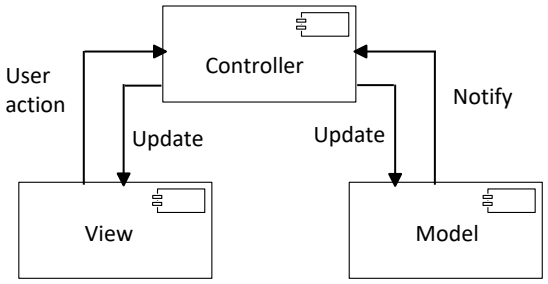
Different frameworks handle controllers in different ways. In particular there are several ways to divide responsibilities between the model and the controller, e.g., data validation, external APIs.

External Services for Mobile Apps

Mobile apps often make extensive use of cloud-based external services, each with an API (e.g., location, validation). These are usually managed by the controller.



Apple's Version of MVC



The diagram shows the model, view, and controller as components. In practice the Model-View-Controller is a program design with three major classes.

International University, VNU-HCMC

Apple's Version of MVC

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

Two challenges:

- A multi-screen app will have several views and controllers sharing the same model.
- It is easy to put too much code into the controller.

Architectural Styles and Design Patterns

There are many variants of the common architectural styles. Do not be surprised if you encounter a variant that is different from the one described in this course.

This is particularly true with the Model-View-Controller style. Several programming frameworks call classes that implement a variant of the Model-View-Controller architectural style a design pattern.

International University, VNU-HCMC

Architectural Styles and Design Patterns

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

In this course we distinguish carefully between architectural styles and design patterns.

- Architectural styles are part of system design. They are defined in terms of subsystems, components, and deployment.
- Design patterns are part of program design. They are defined in terms of classes.

Outlines

System architecture
Three popular architectural styles
Security
Performance

Security in the Software Development Process

The security goal

The security goal is to make sure that the agents (people or external systems) who interact with a computer system, its data, and its resources, are those that the owner of the system would wish to have such interactions.

Security considerations need to be part of the entire software development process. They may have a major impact on the system architecture chosen.

Security Needs and Dangers

Needs

- Secrecy: control of who gets to read information
- Integrity: control of how information changes or resources are used
- Availability: providing prompt access to information and resources
- Accountability: knowing who has had access to resources

Dangers

- Damage to information – integrity
- Disruption of service – availability
- Theft of money – integrity
- Theft of information – secrecy
- Loss of privacy – secrecy

Butler W. Lampson, Computer Security in the Real World
IEEE Computer, June 2004

The Economics of Security

How secure should your system be?

Building secure systems adds cost and time to software development

"Practical security balances the cost of protection and the risk of loss, which is the cost of recovering from a loss times its probability... When the risk is less than the cost of recovering, it's better to accept it as a cost of doing business ... than to pay for better security."

"Many companies have learned that although people may complain about inadequate security, they won't spend much money, sacrifice many features, or put up with much inconvenience to improve it."

Butler W. Lampson, 2004

The Economics of Security

Credit cards: Option A

- The card is a plastic card with all data (e.g., name, number, expiration date) readable by anybody who has access to the card. A copy of the signature is written on the card.
- This is a cheap system to implement but does little to discourage fraud.

Banks in the USA have traditionally used this system.

The Economics of Security

Credit cards: option B (chip and PIN)

- The card has an embossed security chip. To use the card, a particular reader must read the security chip, and the user must type in a confidential 4-digit number.
- This provides excellent protection against fraud but is more expensive and slightly less convenient for merchants and users.

For many years, banks in Europe have used this system.

The new system in the USA is less secure than option B but cheaper to install.

Security within an Organization: People

Many security problems come from people inside the organization

- In a large organization, there will be some dishonest and disgruntled employees.
  - -> Dishonest (e.g., stealing from financial systems)
  - -> Malicious
- Security relies on trusted individuals. What if they are dishonest?

People are intrinsically insecure

- Careless (e.g., leave computers logged on, share passwords)

Design for Security: People

- Make it easy for responsible people to use the system (e.g., make security procedures simple).
- Make it hard for dishonest or careless people (e.g., password management).
- Train people in responsible behavior.
- Test the security of the system thoroughly and repeatedly, particularly after changes.
- Do not hide violations.

External Intruders

All network systems are vulnerable to security breaches by external intruders:

- financial
- malicious
- secrets
- and worse

Modern software is so complex that it is impossible to eliminate all vulnerabilities.

Many skilled individuals and organizations are continually seeking to discover and exploit new vulnerabilities.

External Intruders

Examples of external security vulnerabilities:

- unauthorized access – modify software, install listening devices
- backdoors – bypass authentication
- denial of service – overload and other forms of blocking
- eavesdropping
- spoofing
- phishing etc., etc.

This list is derived from Wikipedia

External Intruders: Minimizing Risk

There is no way to guarantee security from external intruders, but careful software development can make a major difference.

How to minimize the risks:

- System design – secure protocols, authentication, barriers to access
- Programming – defensive programming and rigorous testing
- Operating procedures – backup, auditing, vulnerability testing
- Training and monitoring personnel

Minimizing Risks: System Architecture

The system architecture can minimize risks in various ways:

- Secure protocols, e.g., HTTPS encryption.
- Authentication, e.g., encryption of passwords in transmission and when stored, two factor authentication.
- Barriers, e.g., firewalls, private networks, and virtual private networks.
- Data security, e.g., encryption of stored data, backup.

Security Techniques: Barriers

Place barriers that separate parts of a complex system:

- Isolate components, e.g., do not connect a computer to a network
- Firewalls
- Require authentication to access certain systems or parts of systems

Every barrier imposes restrictions on permitted uses of the system.

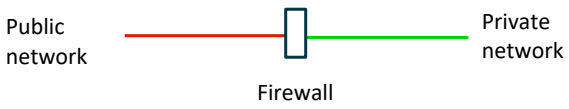
Barriers are most effective when the system can be divided into subsystems.

Example: Integration of Internet Explorer into Windows

International University, VNU-HCMC

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

Barriers: Firewall



A **firewall** is a computer at the junction of two network segments that:

- Inspects every packet that attempts to cross the boundary
- Rejects any packet that does not satisfy certain criteria, e.g.,
  - -> an incoming request to open a TCP connection
  - -> an unknown packet type

Firewalls provide increased security at a loss of flexibility, inconvenience for users, and extra system administration.

Security Techniques: Authentication & Authorization

Authentication establishes the identity of an agent:

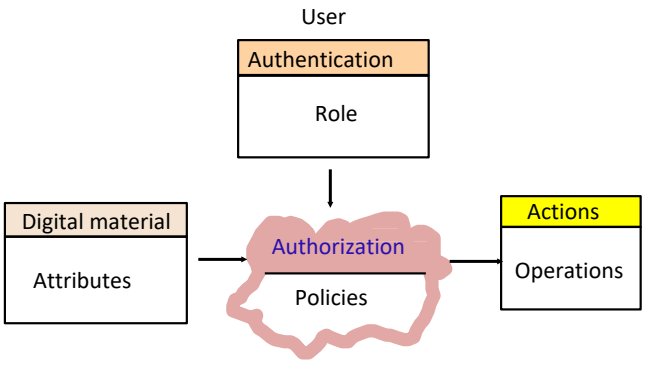
- What does the agent know (e.g., password)?
- What does the agent possess (e.g., smart card)?
- What does the agent have physical access to (e.g., crt-alt-del)?
- What are the physical properties of the agent (e.g., fingerprint)?

Authorization establishes what an authenticated agent may do:

- Access control lists
- Group membership

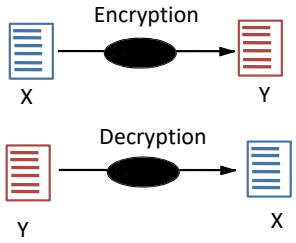
Example:

An Access Architecture for Digital Content



Security Techniques: Encryption

Allows data to be stored and transmitted securely, even when the bits are viewed by unauthorized agents and the algorithms are known.



- Private key and public key
- Digital signatures

Minimizing Risks: Programming

The software development challenge

- develop secure and reliable components
- protect whole system so that security problems in parts of it do not spread to the entire system

A large system will have many agents and components

- each is potentially unreliable and insecure
- components acquired from third parties may have unknown security problems

Minimizing Risks: Programming

The commercial off-the-shelf problem

- Developers of off-the-shelf software have considerable incentives to supply software with many options and features.
- In developing such software rapidly, they need more incentives to be thorough about security.

Programming Secure Software

Programs **that interface with the outside world** (e.g., web sites, mail servers) need to be written in a manner that resists intrusion.

For the top 25 programming errors, see: *Common Weakness Evaluation: A Community-Developed Dictionary of Software Weakness Types*.

http://cwe.mitre.org/top25/

- Insecure interaction between components
- Risky resource management
- Porous defenses

Project management and test procedures must ensure that programs avoid these errors.

Programming Secure Software

The following list is from the SANS Security Institute, *Essential Skills for Secure Programmers Using Java/JavaEE*, http://www.sans.org/

- Input handling
- Authentication & session management
- Access control (authorization)
- Java types & JVM management
- Application faults & logging
- Encryption services
- Concurrency and threading
- Connection patterns

Minimizing Risks: Procedures

The operating procedures must anticipate security problems.

A senior member of staff must have responsibility for security.

Equipment

- All system software should be kept up to date with latest security patches.
- All systems should run up to date virus checking software.
- Rules about passwords, personal equipment, and non-standard software should be explicit.

Minimizing Risks: Procedures

Routine checks

- Run network security tests regularly.
- Run password checkers.

Training

- Keep staff informed about security. Ask for their advice.

Operations: Recovery

Sooner or later every system fails because of hardware, software, operational, or security problems.

The operating procedures must anticipate loss of data and damage to systems, which can happen at any moment.

Backup techniques

- At regular intervals check point the system
- At regular intervals backup all data.
- Keep full audit trails of all important transactions

Recovery software

- Recovery software is complex. It needs to be tested regularly in realistic situations.
- A good practice is to rebuild the entire system, but this may not be possible with large collections of data.

Security in the Software Development Process

Conclusion

You can never guarantee that a system is completely secure, but you can do a great deal to minimize the risks and to be able to recover from problems.

Outlines

System architecture
Three popular architectural styles
Security
Performance

Performance of Computer Systems

In most computer systems

The cost of people is much greater than the cost of hardware

Yet performance is important

A single bottleneck can slow down an entire system

Future loads may be much greater than predicted

When Performance Matters

- **Real time systems** when computation must be fast enough to support the service provided, e.g., fly-by wire control systems have tight response time requirements.
- **Very large computations** where elapsed time may be measured in days, e.g., calculation of weather forecasts must be fast enough for the forecasts to be useful.
- **User interfaces** where humans have high expectations, e.g., mouse tracking must appear instantaneous.
- **Transaction processing** where staff need to be productive and customers not annoyed by delays, e.g., airline check-in.

High-Performance Computing

High-performance computing:

- Large data collections (e.g., Amazon)
- Huge numbers of users (e.g., Google)
- Large computations (e.g., weather forecasting)

Must balance cost of hardware against cost of software development

- Some configurations are very difficult to program and debug
- Sometimes it is possible to isolate applications programmers from the system complexities

Performance challenges for all software systems

Tasks

- Predict performance problems before a system is implemented.
- Design and build a system that is not vulnerable to performance problems.
- Identify causes and fix problems after a system is implemented.

Performance challenges for all software systems

Basic techniques

- Understand how the underlying hardware and networks components interact with the software when executing the system.
- For each subsystem calculate the capacity and load. The capacity is a combination of the hardware and the software architecture.
- Identify subsystems that are near peak capacity.

Example

Calculations indicate that the capacity of a search system is 1,000 searches per second. What is the anticipated peak demand?

Interactions between Hardware and Software

Examples

- In a distributed system, what messages pass between nodes?
- How many times must the system read from disk for a single transaction?
- What buffering and caching is used?
- Are operations in parallel or sequential?
- Are other systems competing for a shared resource (e.g., a network or server farm)?
- How does the operating system schedule tasks?

Look for Bottlenecks

Usually, CPU performance is not the limiting factor.

Hardware bottlenecks

- Reading data from disk
- Shortage of memory (including paging)
- Moving data from memory to CPU
- Network capacity

Inefficient software

- Algorithms that do not scale well
- Parallel and sequential processing

Look for Bottlenecks

CPU performance is a limiting constraint in certain domains, e.g.:

- large data analysis (e.g., searching)
- mathematical computation (e.g., engineering)
- compression and encryption
- multimedia (e.g., video)
- perception (e.g., image processing)

Timescale of Different Components

	Operations per second
CPU instruction:	2,000,000,000
Disk latency:	200
Disk read:	100,000,000 bytes
Network LAN:	10,000,000 bytes

Actual performance may be considerably less than the theoretical peak.

Look for Bottlenecks: Utilization

Utilization is the proportion of the capacity of a service that is used on average.

Utilization = proportion of capacity of service that is used

$$= \frac{\text{mean service time for a transaction}}{\text{mean inter-arrival time of transactions}}$$

When the utilization of any hardware component exceeds 0.3, be prepared for congestion.

Peak loads and temporary increases in demand can be much greater than the average.

Predicting System Performance

- Direct measurement on subsystem (benchmark)
- Mathematical models (queueing theory)
- Simulation

All require detailed understanding of the interaction between software and hardware systems.

Mathematical Models

Queueing theory

Good estimates of congestion can be made for single-server queues with:

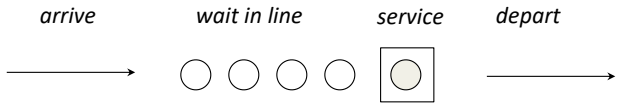
- arrivals that are independent, random events (Poisson process)
- service times that follow families of distributions (e.g., negative exponential, gamma)

Many of the results can be extended to multi-server queues.

Much of the early work in queueing theory by Erlang was to model congestion in telephone networks.

Mathematical Models: Queues

Single server queue

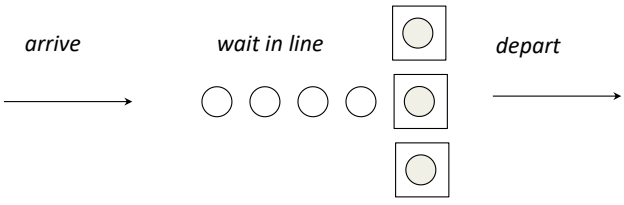


Examples

- Requests to read from a disk (with no buffering or other optimization)
- Customers waiting for check in at an airport, with a single check-in desk

Queues

Multi-server queue



Examples

- Tasks being processed on a computer with several processors
- Customers waiting for check in at an airport, with a several check-in desks

Techniques: Simulation

Build a computer program that models the system as set of states and events.

advance simulated time

determine which events occurred

update state and event list

repeat

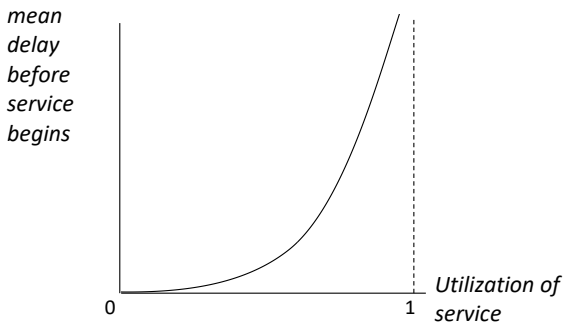
Discrete time simulation: Time is advanced in fixed steps (e.g., 1 millisecond)

Next event simulation: Time is advanced to next event

Events can be simulated by random variables (e.g., arrival of next customer, completion of disk latency), or by using data collected from an operational system.

Behavior of Queues: Utilization

The exact shape of the curve depends on the type of queue (e.g., single server) and the statistical distributions of arrival times and service times.



Measurements on Operational Systems

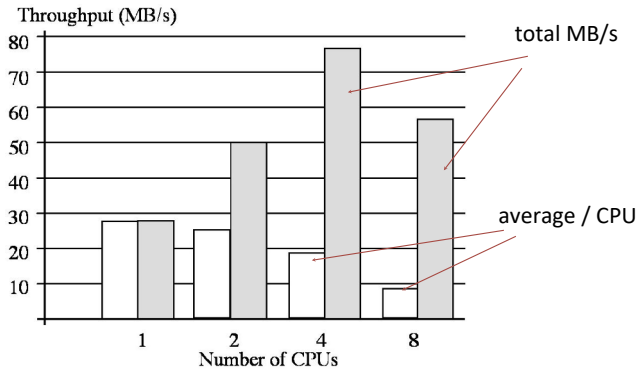
Measurements on operational systems

- Benchmarks: Run system on standard problem sets, sample inputs, or a simulated load on the system.
- Instrumentation: Clock specific events.

If you have any doubt about the performance of part of a system, experiment with a simulated load.

Example: Web Laboratory

Benchmark: throughput v. number of CPUs on a symmetric multiprocessor



Case Study: Performance of Disk Farm

When many transaction use a disk farm, each transaction must:

- wait for specific disk
- wait for I/O channel
- send signal to move heads on disk
- wait for I/O channel
- pause for disk rotation (latency)
- read data

Close agreement between: results from queuing theory, simulation, and direct measurement (within 15%).

Fixing Bad Performance

If a system performs badly, begin by identifying the cause:

Instrumentation. Add timers to the code. Often this will reveal that delays are centered in a specific part of the system.

Test loads. Run the system with varying loads, e.g., high transaction rates, large input files, many users, etc. This may reveal the characteristics of when the system runs badly.

Design and code reviews. Team review of system design, program design, and suspect sections of code. This may reveal an algorithm that is running very slowly, e.g., a sort, locking procedure, etc.

Find the underlying cause and fix it or the problem will return!

Predicting Performance Change: Moore's Law

Original version:

The density of transistors in an integrated circuit will double every year. (Gordon Moore, Intel, 1965)

Current version:

Cost/performance of silicon chips doubles every 18 months.

Moore's Law: Rules of Thumb

Planning assumptions

Silicon chips: cost/performance improves 30%/ year

- in 12 years = 20:1
- in 24 years = 500:1

Magnetic media: cost/performance improves 40% / year

- in 12 years = 50:1
- in 24 years = 3,000:1

These assumptions are conservative. During some periods, the increases have been considerably faster.

Recently, the rate of performance increase in individual components, such as CPUs, has slowed down, but the overall rate of increase has been maintained by placing many CPU cores on a single chip.

Moore's Law and System Design

Feasibility study: Production use: Withdrawn from production:	2013	2016	2026
Processor speeds Memory sizes: Disk capacity:	1 1 1	2.2 2.2 2.2	30 30 30
System cost:	1	0.4	0.03

Moore's Law Example

Will this be a typical laptop?

	2017	2027
Processors	2 x 2.5 GHz	8 x 10 GHz
Memory	8 GB	200 GB
Disc	500 GB	15 TB
Network	1 Gb/s	25 Gb/s

← or 100 processors?

Surely there will be some fundamental changes in how this this power is packaged and used.

Parkinson's Law

Original:

Work expands to fill the time available. (C. Northcote Parkinson)

Software development version:

(a) Demand will expand to use all the hardware available.

(b) Low prices will create new demands.

(c) Your software will be used on equipment that you have not envisioned.

False Assumptions from the Past

Be careful about the assumptions that you make

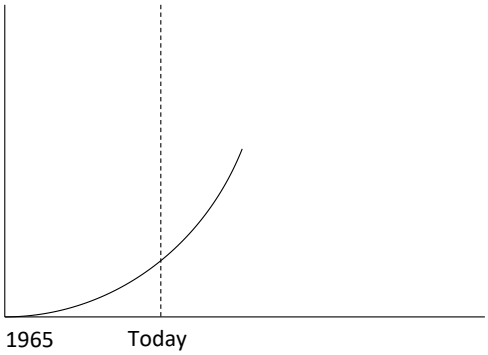
Here are some past assumptions that caused problems:

- Unix file system will never exceed 2 GB (2^{32} bytes).
- AppleTalk networks will never have more than 256 hosts (2^8 bits).
- GPS software will not last more than 1024 weeks.
- Two bytes are sufficient to represent a year (Y2K bug).

etc, etc,…

Moore's Law and the Long Term

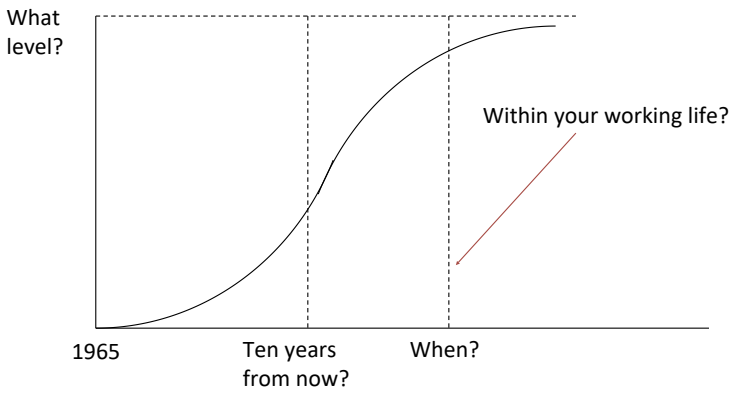
Assoc. Prof. Nguyen Thi Thuy Loan, PhD

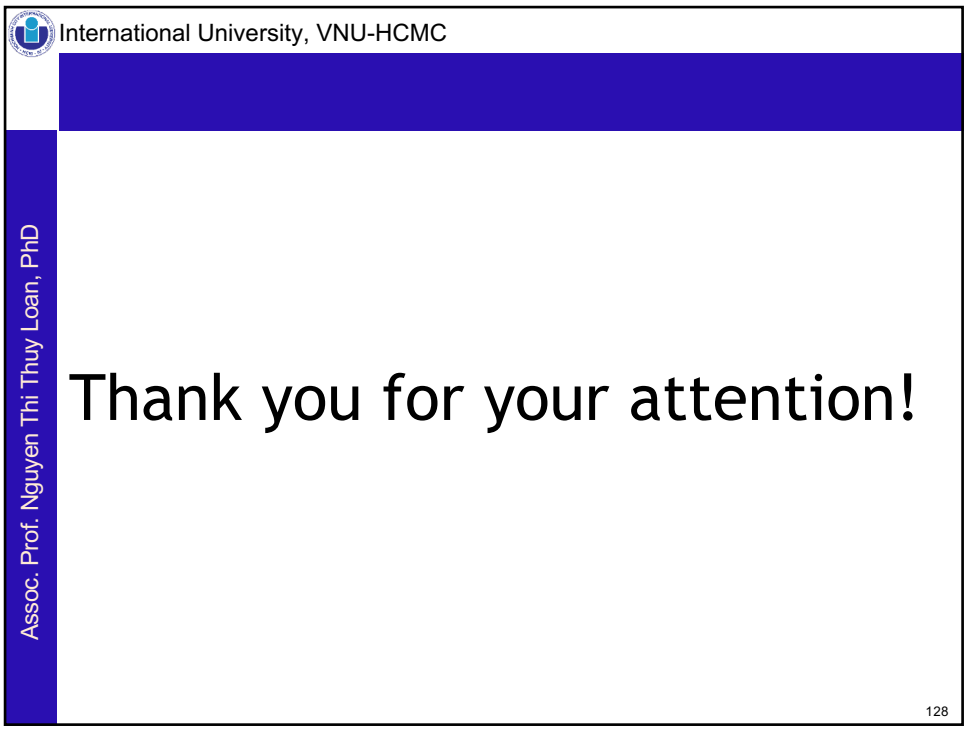


International University, VNU-HCMC

Moore's Law and the Long Term

Assoc. Prof. Nguyen Thi Thuy Loan, PhD




International University, VNU-HCMC
Assoc. Prof. Nguyen Thi Thuy Loan, PhD
Thank you for your attention!
128