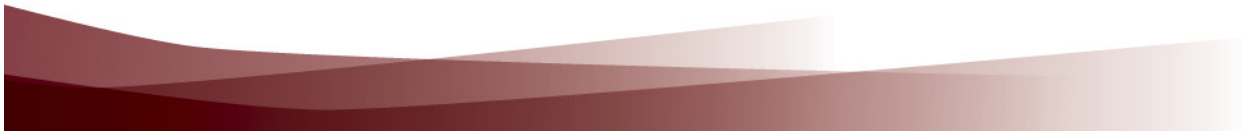
SOA and Virtualization Technologies

(ENCS 691K – Chapter 2)

Roch Glitho, PhD

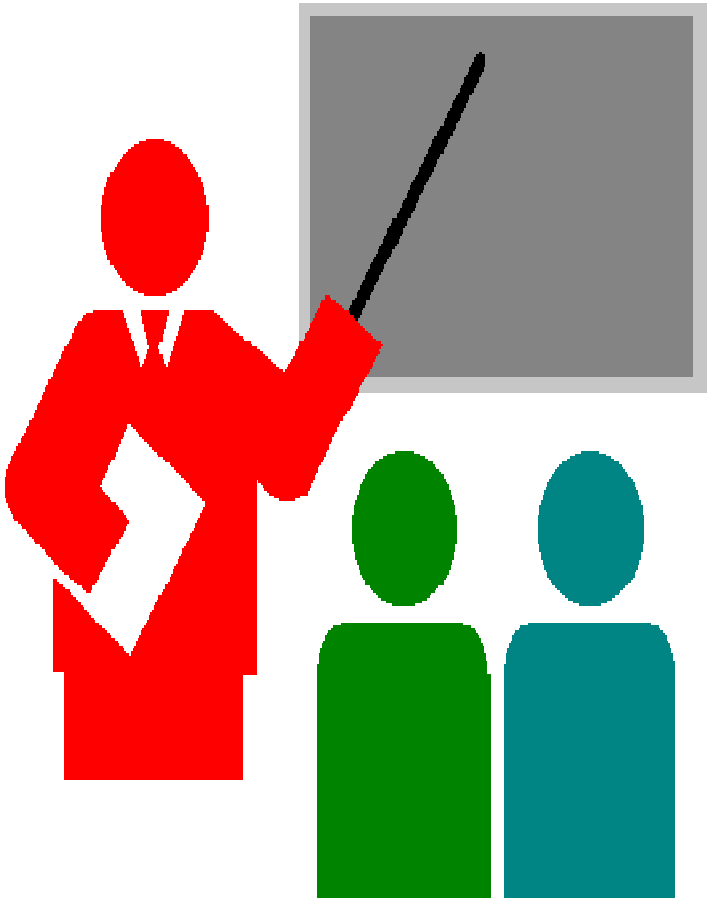
Associate Professor and Canada Research Chair

My URL - http://users.encs.concordia.ca/~glitho/



Concordia University
Engineering and Computer Science
Concordia Institute for Information Systems Engineering

The Key Technologies on Which Cloud Computing Relies



- Web Services
- Virtualization

References (Web Services)

1. F. Belqasmi, C. Fu, R. Glitho, Services Provisioning in Next Generation Networks: A Survey, IEEE Communications Magazine, December 2011

2. L. Richardson and S. Ruby, "RESTful Web Services", O' Reilly & Associates, ISBN 10: 0-596-52926-0, May 2007

3. Lightweight REST Framework, http://www.restlet.org/

4. C. Pautasso, O. Zimmermann, and F. Leymann, "RESTful Web Services vs. "Big"Web Services: Making the Right Architectural Decision", In Proceedings of the 17th International World Wide Web Conference, pages 805–814, Beijing, China, April 2008, ACM Press.

5. C. Pautasso and E. Wilde, "Why is the web loosely coupled? A multi-faceted metric for service design", in Proc. of the 18th World Wide Web Conference, Madrid, Spain (April 2009)

References (Virtualization)

6. M. Pearce et al., Virtualization: Issues, Security, Threats, and Solutions, ACM Computing Survey, February 2013

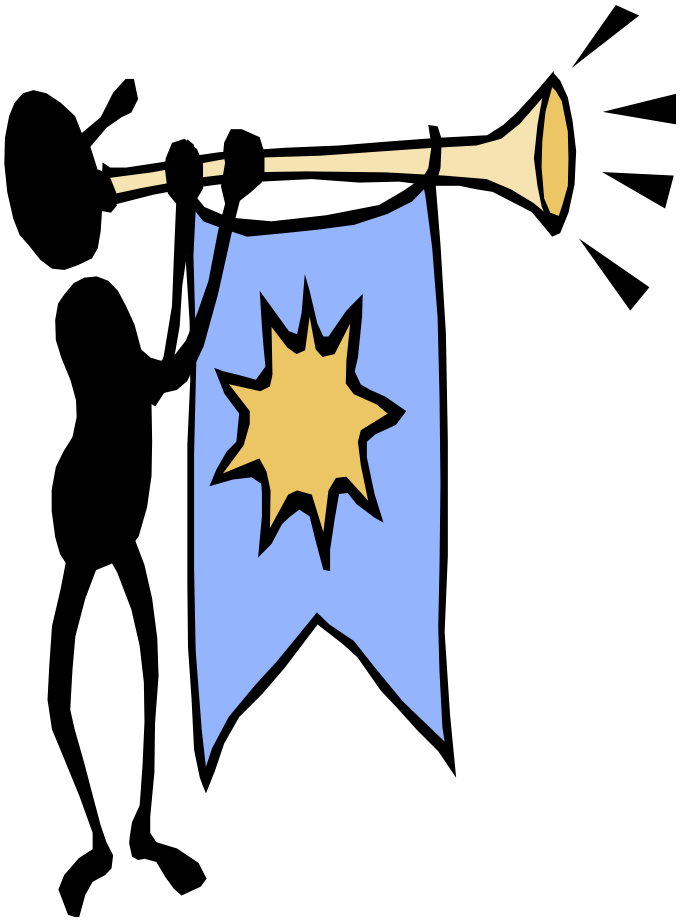
7. A. Khan et al., Network Vitrtualization: A Hypervisor for the Internet?, IEEE Communications Magazine, January 2012

8. N.M Chowdhury and r. Boutaba, Network Virtualization: State of the Art and Research Challenges, IEEE Communications Magazine, July 2009

9. J. Carapinha et al., Network Virtualization – A View from the Bottom, VISA '09 Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures, Pages 73-80

Web Services

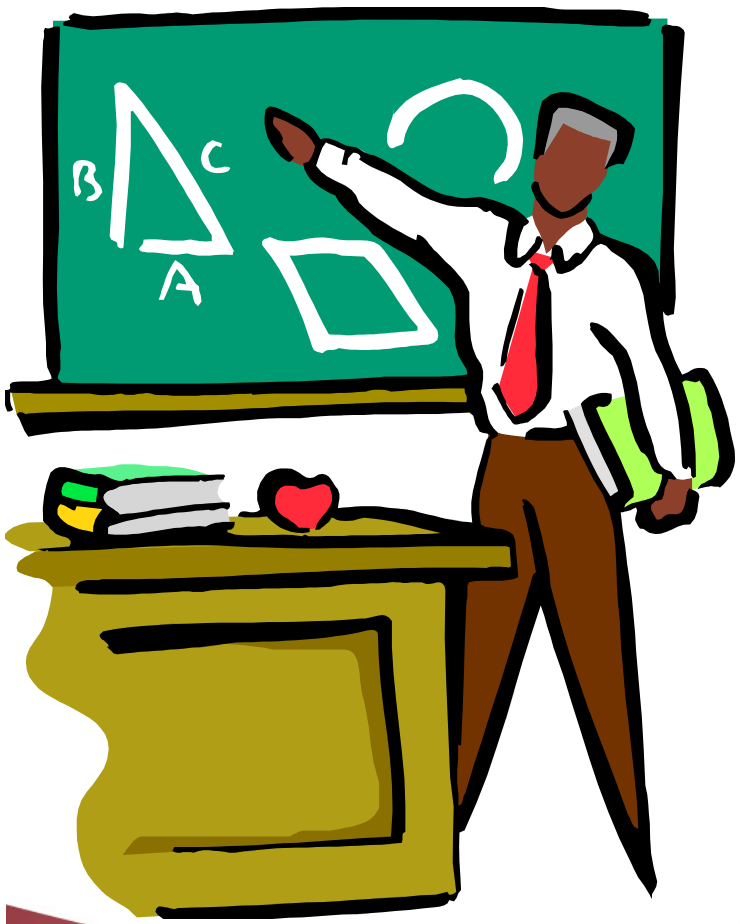
Outline



1. Introduction to Web Services

2. RESTful Web Services Overview

Introduction to Web Services



1. Definition and principles
2. Overall business model
3. Technologies

Web Services so far

- **SOAP – BASED WEB SERVICES**
- **RESTFul Web Services**

- This part of the course will discuss the general characteristics of Web services

Definitions and principles

Today		Tomorrow
• Publication of documents	→	• Publication of "reusable business logic"
• Human interaction	→	• Automated Program to program interaction
• Proprietary ad-hoc interfaces	→	• Industry standard interfaces

XML Technology

Note: There are other technologies such as JSON that may be used

Definitions and principles

"The term Web Services refers to an architecture that allows applications (on the Web) to talk to each other. Period. End of statement"

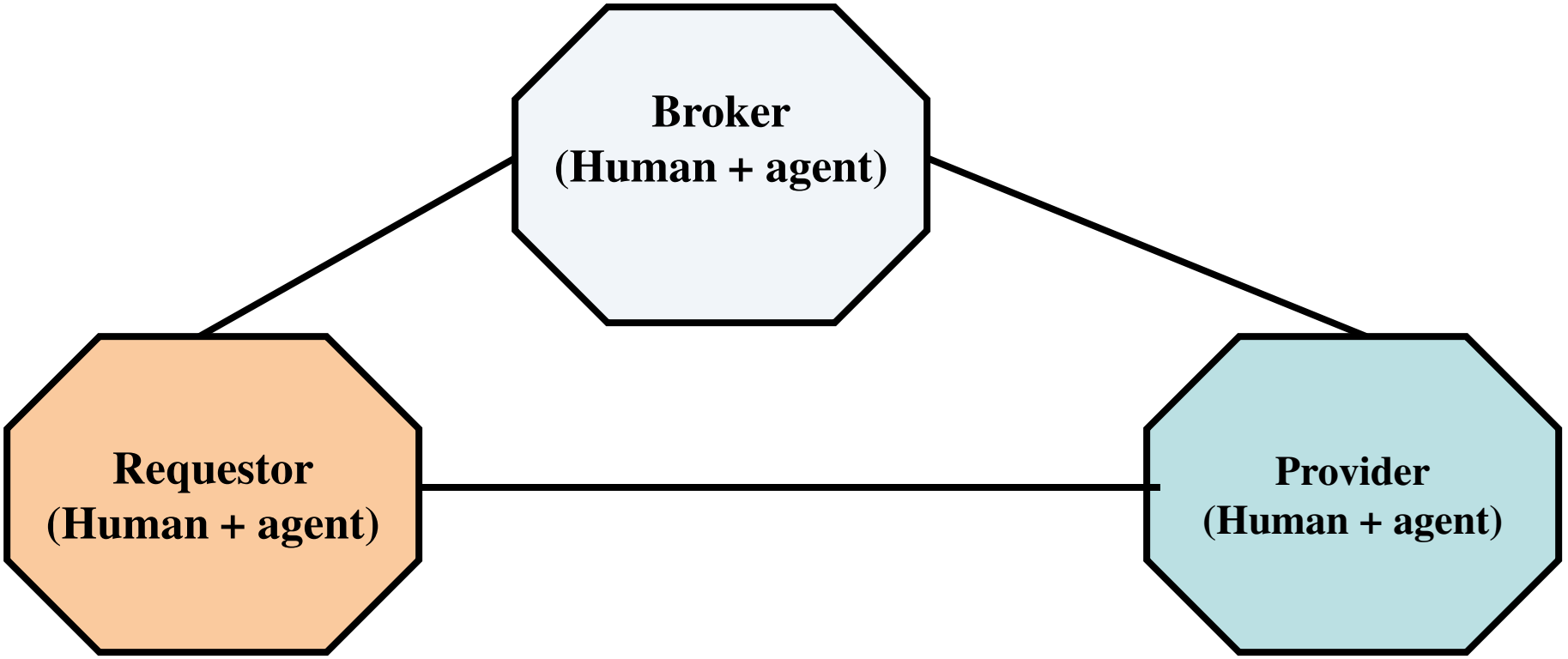
Adam Bobsworth in ACM Queue, Vol1, No1

Definitions and principles

The three fundamental principles, still according to Adam Bobsworth:

1. Coarse grained approach (I.e. high level interface)
2. Loose coupling (e.g. application A which talks to application B should not necessarily be re-written if application B is modified)
3. Synchronous mode of communication, but also asynchronous mode

Business model



Business model

Requestor

- Person or organization that wishes to make use of a Web service.
- Uses an agent (I.e requestor agent) to exchange messages with both broker agent and provider agent.

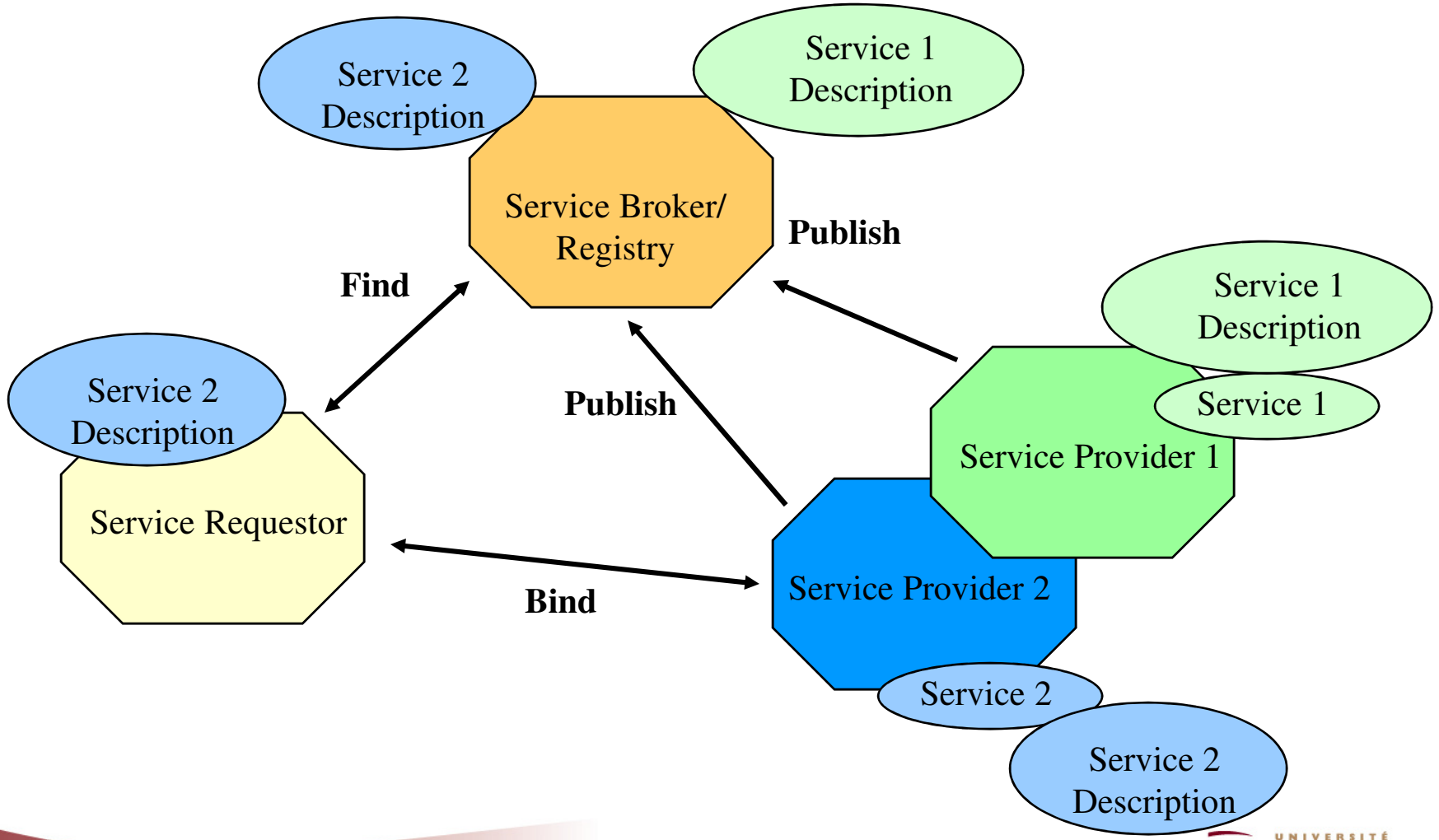
Provider

- Person or organization that owns a Web service it wants to make available for usage
- Use an agent (I.e provider agent) to exchange messages with broker agent and requestor agent.
- The provider agent is also the software piece which implements the Web service (e.g. mapping towards legacy)

Broker

- Person or organization that puts requestors and providers in contact
  - Providers use brokers to publish Web services
  - Requestors use brokers to discover Web services
- Use an agent (I.e broker agent) to exchange messages with requestor agent and provider agent

Business model



Technologies

Some of the technologies are mandatory for some Web services while optional for other Web services:

HTTP

- Mandatory for RESTful Web services but "optional" for SOAP Based Web services
- Note: In practice HTTP is also used for SOAP Based Web Services

XML

- Mandatory for SOAP Based Web Services but optional for RESTful Web services

HTTP

HTTP (HyperText Transfer Protocol)

- **Is an application-level protocol for distributed, collaborative, hypermedia information systems**
  - **HTTP has been in use since 1990**
  - **HTTP is a request-response protocol**
  - **HTTP requests relates to resources**
    - **A resource is any object or service network that can be identified by a URI (Universal Resource Identifier)**

HTTP

Client

- A program that establishes connections for the purpose of sending requests

User Agent

- The client which initiates a request (e.g. browser)

- Note
  - A request may pass through several servers

HTTP

Server

- An application program that accepts connections in order to service requests by sending back responses
- A given program may be capable of being both a client and a server
- The role depends on connections

HTTP

- Origin server
  - The server on which a given resource resides or is to be created
- Proxy server
  - An intermediary program which acts as both a server and a client for the purpose of making requests on behalf of other clients
- Gateway server
  - receives requests as if it were the origin server for the requested resource, and forwards the request to another server
  - Is transparent to the client

HTTP

HTTP-message = Request | Response

generic-message = start-line
*(message-header CRLF)
CRLF
[message-body]

start-line = Request-Line | Status-Line

HTTP

HEAD

- retrieve meta-information about a web page, without retrieving the page content (ex: get the date for last modification)

GET

- retrieve the page content

PUT

- store the enclosed content under the supplied Request-URI

POST

- add the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI
  - E.g.
    - Post a message to a mailinglist
    - Extend a database by appending information
    - Transfer a form data

HTTP

DELETE

- Deletes the page

TRACE

- Debug

OPTIONS

- Allows the client to discover the options supported by the server supporte

CONNECT

- Not used currently

HTTP

The built-in HTTP request methods.

Method	Description
GET	Request to read a Web page
HEAD	Request to read a Web page's header
PUT	Request to store a Web page
POST	Append to a named resource (e.g., a Web page)
DELETE	Remove the Web page
TRACE	Echo the incoming request
CONNECT	Reserved for future use
OPTIONS	Query certain options

HTTP

The status code response groups.

Code	Meaning	Examples
1xx	Information	100 = server agrees to handle client's request
2xx	Success	200 = request succeeded; 204 = no content present
3xx	Redirection	301 = page moved; 304 = cached page still valid
4xx	Client error	403 = forbidden page; 404 = page not found
5xx	Server error	500 = internal server error; 503 = try again later

XML

XML documents

Data objects made of elements

- <element> content </element>

Well-formed Documents

- If it obeys to the XML syntax
  - Exp: - All XML elements must have a closing tag
    - The name in an element's end-tag MUST match the element type in the start-tag.
    - All XML elements must be properly nested

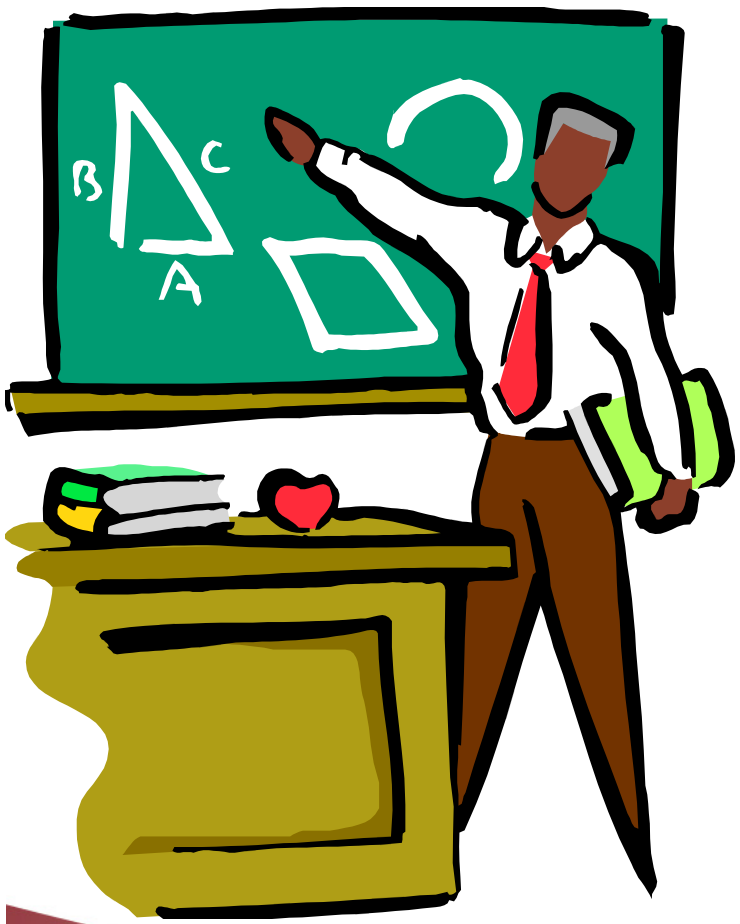
XML

XML processor

- Read XML documents
- Provide access to the content and the structure
- Behaviour described in the XML specifications
- Navigate XML document structure and add, modify, or delete its elements.

- Most popular programming APIs
  - Document Object Model (DOM) from W3C
  - Simple API for XML (SAX) – From XML-DEV mailing list

RESTFul Web Services



1. Introduction
2. Resource Oriented Architecture
3. Resources
4. Properties
5. Tool kits
6. Examples of RESTful Web services

Introduction

- What about using the Web's basic technologies (e.g. HTTP) as a platform for distributed services?
  - This is what is REST about.

Introduction

- REST was first coined by Roy Fielding in his Ph.D. dissertation in 2000
- It is a network architectural style for distributed hypermedia systems.

Introduction

- REST is a way to reunite the programmable web with the human web.
- It is simple
  - Uses existing web standards
  - The necessary infrastructure has already become pervasive
  - RESTFull web services are lightweight
  - HTTP traverse firewall

Introduction

- RESTFul web services are easy for clients to use
- Relies on HTTP and inherits its advantages, mainly
  - Statelessness
  - Addressability
  - Unified interface

Resource-Oriented Architecture

- The Resource-Oriented Architecture (ROA)
  - Is a RESTful architecture
  - Provides a commonsense set of rules for designing RESTful web services

Resource-Oriented Architecture

- Concepts
  - Resources
    - Resources names (Unified Resource Identifiers-URIs)
    - Resources representations
    - Links between resources
- Key properties:
  - Addressability
  - Statelessness
  - Uniform interface

Resources

- What's a Resource?
  - A resource is any information that
    - can be named
    - Is important enough to be referenced as a thing in itself
  - A resource may be a physical object or an abstract concept
  - e.g.
    - a document
    - a row in a database
    - the result of running an algorithm.

Resources

- Naming:
  - Unified Resource Identifier (URI)
    - The URI is the name and address of a resource
    - Each resource should have at least one URI
    - URIs should have a structure and should vary in predictable ways

Resource

Representation

- A representation is any useful information about the state of a resource
- Different representation formats can be used (Unlike SOAP based Web services)
  - *plain-text*
  - *JSON*
  - XML
  - XHTML
  - ….

Resource

…

- In most RESTful web services, representations are hypermedia
  - i.e. documents that contain data, and links to other resources.

Properties

- Addressability
  - An application is addressable if it exposes a URI for every piece of information it serves
  - This may be an infinite number of URIs
    - e.g. for search results
      - *http://www.google.com/search?q=jellyfish*

Properties

- Statelessness
  - The state should stay on the client side, and be transmitted to the server for every request that needs it.
    - Makes the protocol simpler
    - Ease load balancing

Properties

- Uniform interface
  - *HTTP GET:*
    - Retrieve a representation of a resource
  - *HTTP PUT*
    - Create a new resource, where the client is in charge of creating the resource URI: *HTTP PUT* to the new URI
    - Modify an existing resource: *HTTP PUT* to an existing URI
  - *HTTP POST:*
    - Create a new resource, where the server is in charge of creating the resource URI: *HTTP POST* to the URI of the superordinate of the new resource
  - *HTTP DELETE:*
    - Delete an existing resource:
  - HTTP HEAD:
    - Fetch metadata about a resource
  - HTTP OPTIONS:
    - Lets the client discover what it's allowed to do with a resource.

Examples of tool kits

- RestLet
- Jersey

Examples of RESTful Web Services

- Examples of existing RESTful web services include:
  - Amazon's Simple Storage Service (S3) (*http://aws.amazon.com/s3*)
  - Services that expose the Atom Publishing Protocol (*http://www.ietf.org/html.charters/atompub-charter.html*) and its variants such as GData (*http://code.google.com/apis/gdata/*)
  - Most of Yahoo!'s web services (*http://developer.yahoo.com/*)
  - Twitter is a popular blogging site that uses RESTful Web services extensively.

Examples of RESTful Web Services

Resources	URL Base URL: http://{serverRoot}/{apiVersion}/smsmessaging	HTTP action
Outbound SMS message requests	/outbound/{senderAddress}/requests	GET: read pending outbound message requests POST: create new outbound messages request
Outbound SMS message request and delivery status	/outbound/{senderAddress}/requests /{requestId}	GET: read a given sent message, along with its delivery status
Inbound SMS message subscriptions	/inbound/subscriptions	GET: read all active subscriptions POST: create new message subscription
Individual inbound SMS message subscription	/inbound/subscriptions/{subscriptionId}	GET: read individual subscription DELETE: remove subscription and stop corresponding notifications

Table 2. *A subset of ParlayREST SMS resources.*

Examples of RESTful Web Services

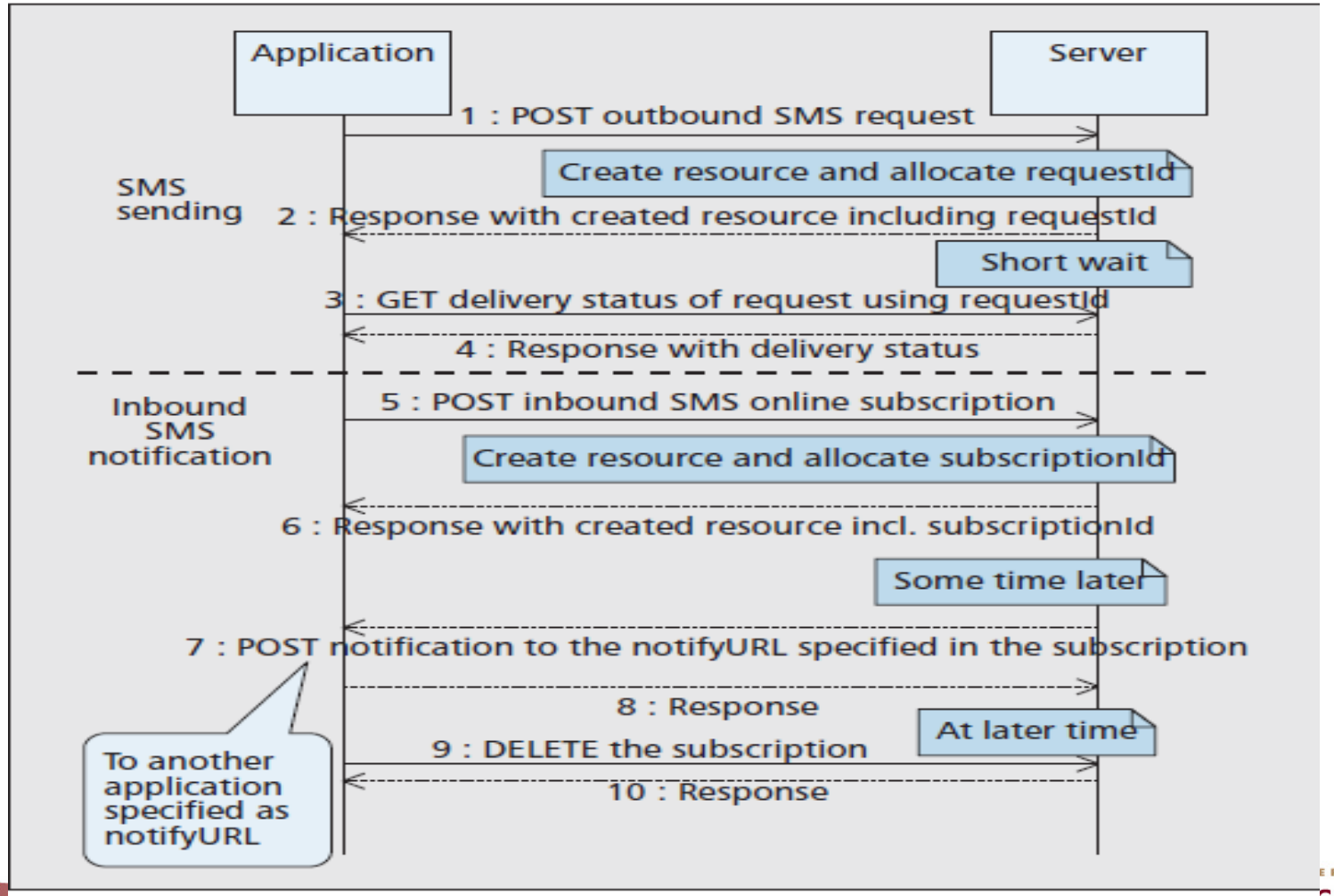
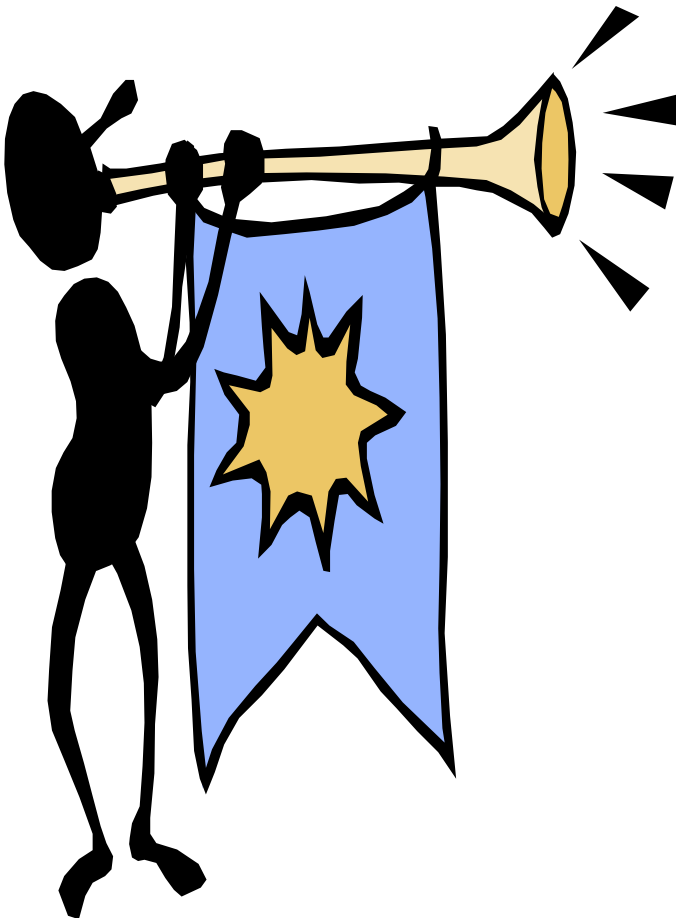


Figure 4. *Sample scenario for SMS handling.*

Virtualization

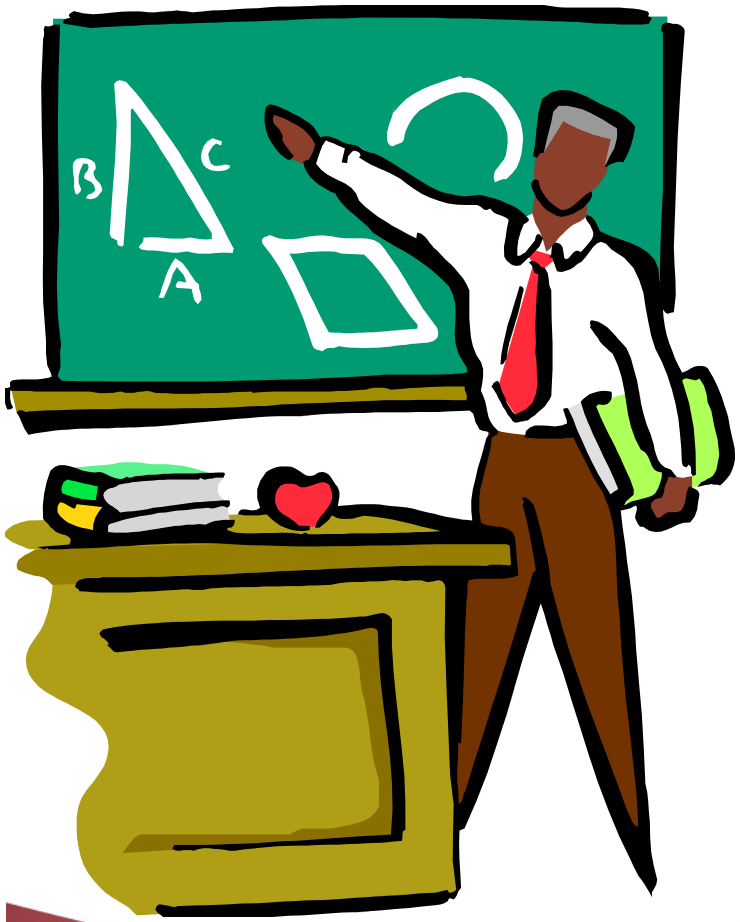
Outline



1. Systems virtualization

2. Network virtualization

Systems Virtualization



1. On operating systems
2. Brief history of systems virtualization
3. Key concepts (virtual machine, virtual machine monitor/hypervisor)
4. Examples of benefits

On operating systems

Some of the motivations

- Only one single tread of CPU can run at a time on any single core consumer machine
- Machine language is tedious

On operating systems

Operating systems bring a level of abstraction on which multiple processes can run at a time – Deal among other things with:

- Multiplexing
- Hardware management issues

However only one operating system can run on a bare single core consumer machine

Brief history

- Systems virtualization dates back to the 60s

- IBM experimentation with “time sharing systems”
  - Need for virtual machines to test how applications / users can time share a real machine

Key concepts

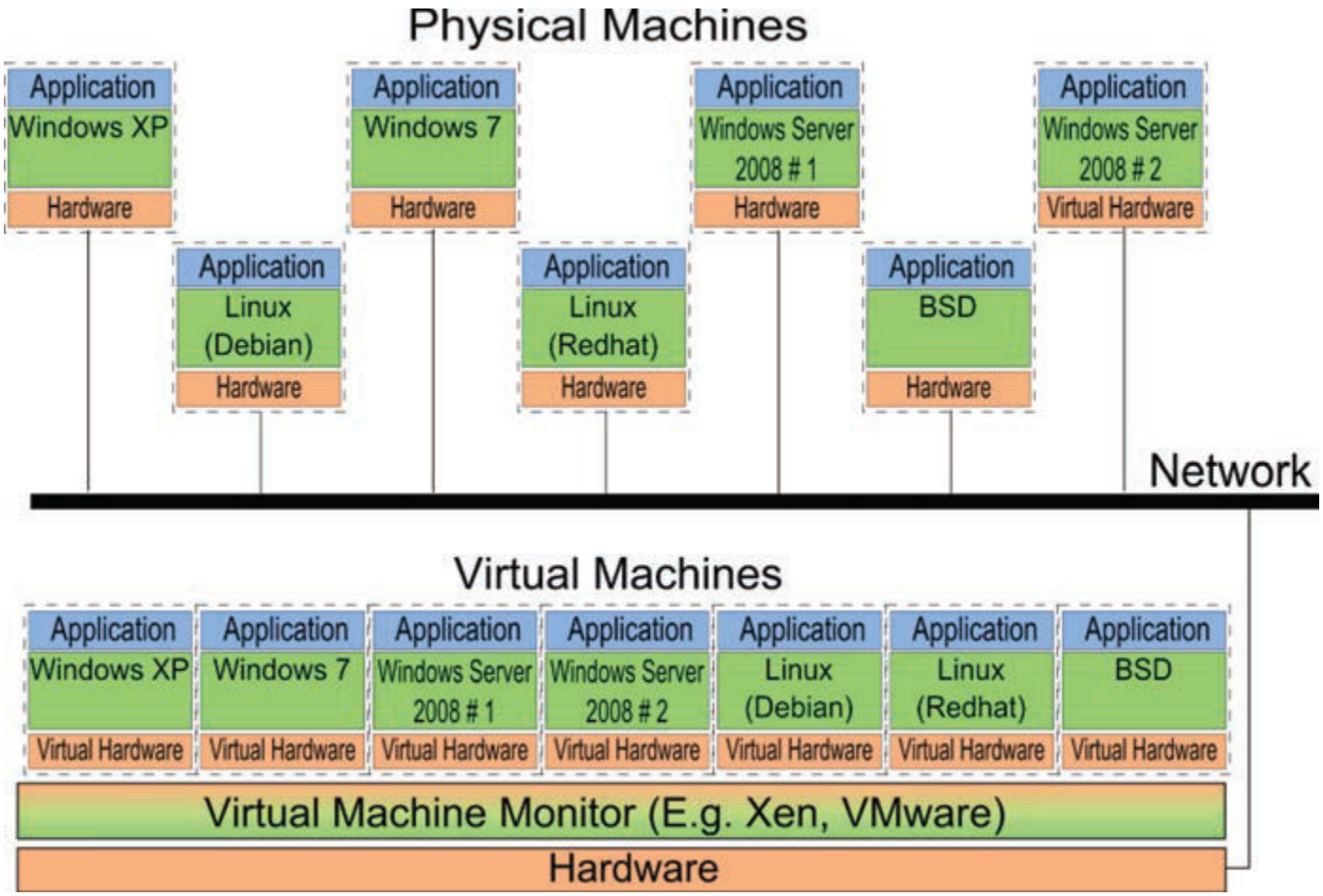
Virtual machine (VM) (sometimes called virtual hardware)

- Software that provides same inputs / outputs and behaviour expected from hardware (i.e. real machine) and that supports operations such as:
  - Create
  - Delete
  - Migrate
  - Increase resources

Virtual machine monitor (also called hypervisor)

- Software environment that enables operations on virtual machines (e.g. XEN, VMWare) and ensures isolation

Key concepts



From reference [6] – Note: There is a small in the figure

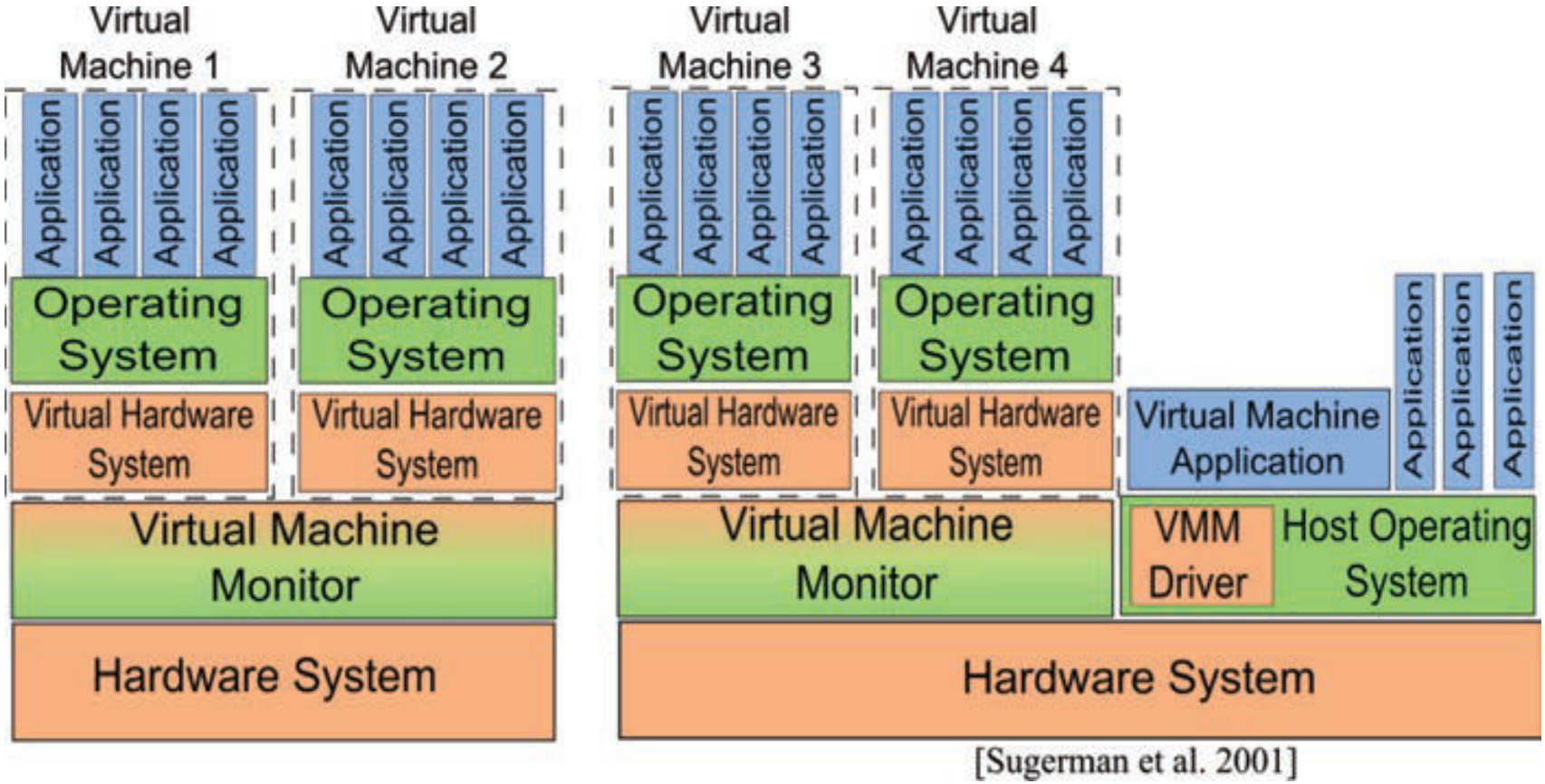
Key concepts

Types of hypervisor

- Type I – bare metal
- Type 2 - hosted

Key concepts

Types of hypervisor/virtual machine monitor



[Sugerman et al. 2001]

Key concepts

Full virtualization vs. para-virtualization

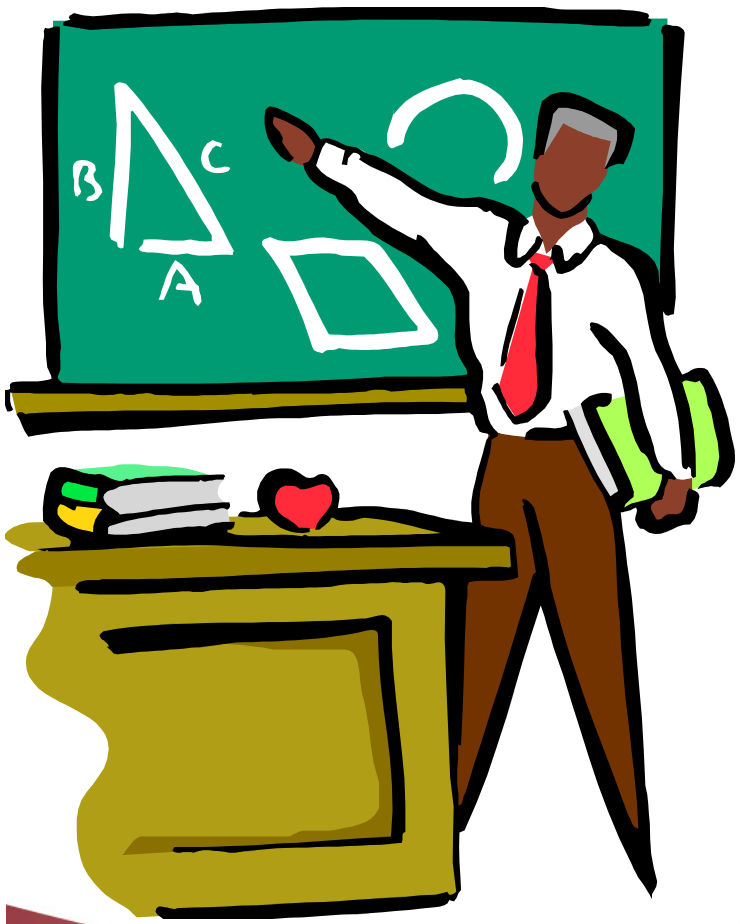
- Full virtualization
  - No need to modify guest operating system before installing it on top of hypervisor
- Para virtualization
  - Operating system needs to be modified
  - Note: Some hardware (e.g. X86) are not fully virtualisable

Examples of Benefits

All benefits are due to the possibility to manipulate virtual machine (e.g. create, delete, increase resources, migrate), e.g.

- Co-existence of operating systems
- Operating systems research
- Software testing and run-time debugging
- Optimization of hardware utilization
- Job migration

Network virtualization



1. Motivations

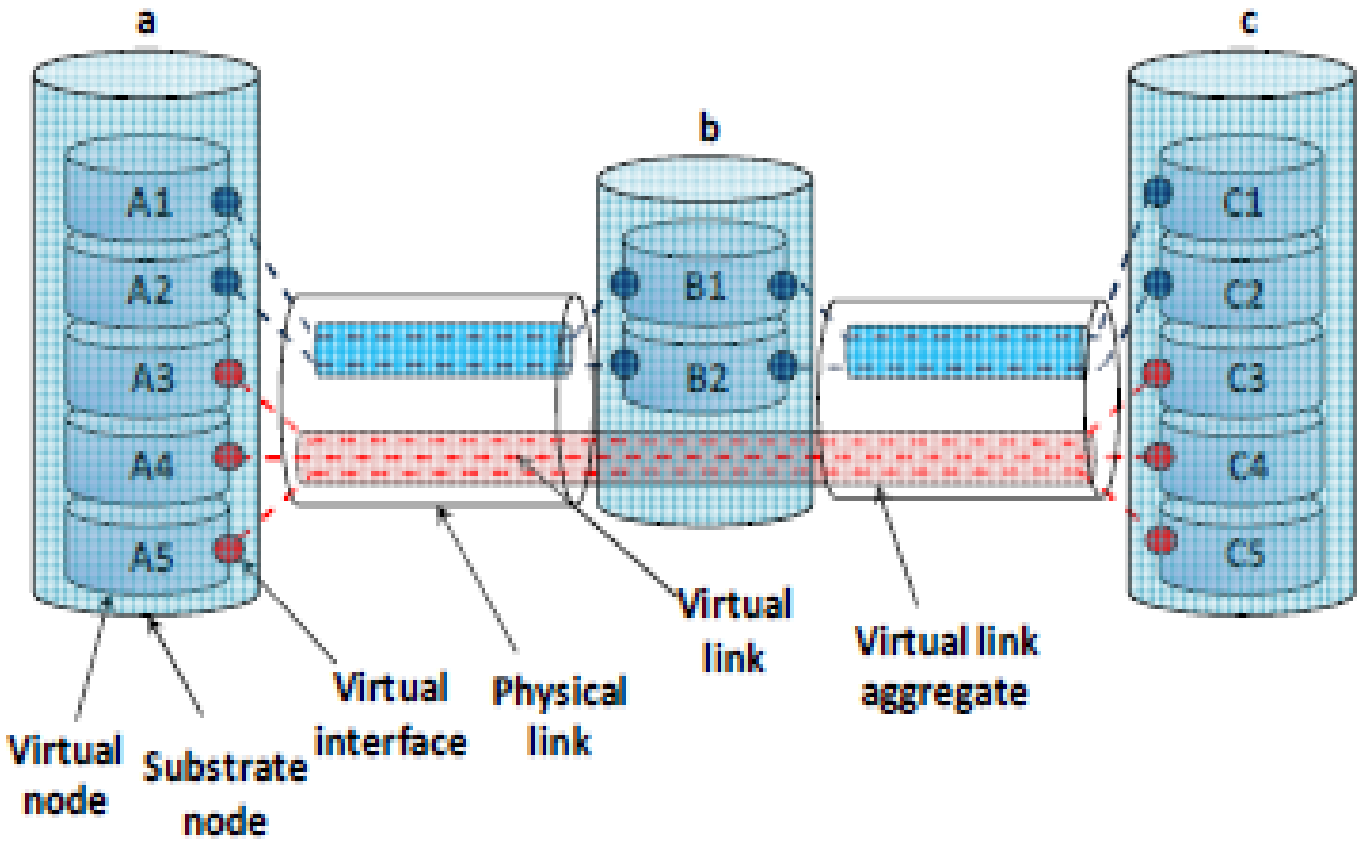
2. Basic components

Motivations

Bring the benefits of systems virtualization to the networking world, e.g.

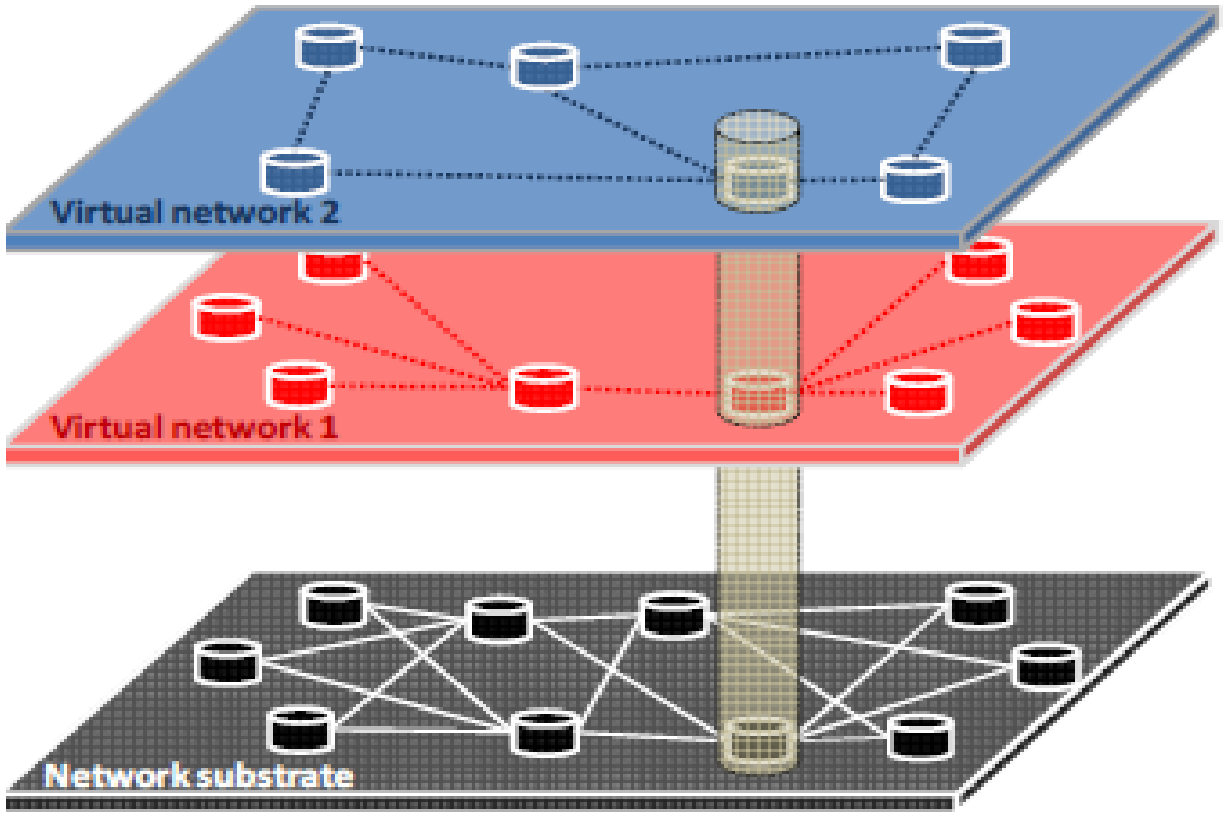
- Co-existence of virtual networks on top of a same real network
  - Note: Virtual Private Networks (VPNs) do not rely on virtualization and have several limitations
    - Different technologies and protocol stacks cannot be used for instance
- Networking research
- Optimization of networking resources utilization
  - Nodes
  - Links

Basic components



From reference 9

Basic components



From reference 9

The End