Software as a Service (SaaS)

Week 5

School of Software

Faculty of Engineering and Information Technology

University of Technology Sydney

SCHOOL OF SOFTWARE

Learning Objectives

- Understand Software applications
- Define Software-as-a-Service (SaaS)
  - Understand SaaS characteristics
  - Understand SaaS benefits and implications
- Comparison between on-premise applications & SaaS applications
- Integrating on-premise (in-house) applications with SaaS applications
- Example of SaaS Environments

Software Applications

- (Some) Software Applications in enterprises
  - Email Software Applications
  - Customer Relationship Management (CRM)
  - Human Resource Management (HRM)
  - Finance Management (planning, forecasting …etc.)
  - Patient records management (for hospitals)
  - Logistics/Supply Chain Management (for transportation companies)
  - …….

Using Software Application(s)

- Typical traditional mechanism of procuring and using software applications usually involves the following steps:
  - Identification of an appropriate software application (*depending on your business requirements*)
  - Purchasing the license of the software application (*How many users require access to the software application?*)
  - Installing the software application in-house (*Traditionally installation is done by the corporate in-house IT team*)
  - Deploying patches and upgrades on the procured software application (*An on-going activity and is traditionally done by the corporate in-house IT team*)
  - Reporting bugs to the software vendor (*An on-going activity*)

Using Software Application(s)

- Some disadvantages of procuring and using software applications using traditional mechanism are:
  - Need for up-front capital investment and provisioning
    - Determine the number of required licenses;
    - Purchase the appropriate licenses;
    - Requires (large) upfront capital investment in procuring software applications
  - Ongoing investment in the maintenance of the software applications (installation, upgrades, patches, compatibility… etc…)
    - Investment in manpower, training and in purchasing the patches or upgrades
  - New and better software product gets released in the market

How can cloud computing help?

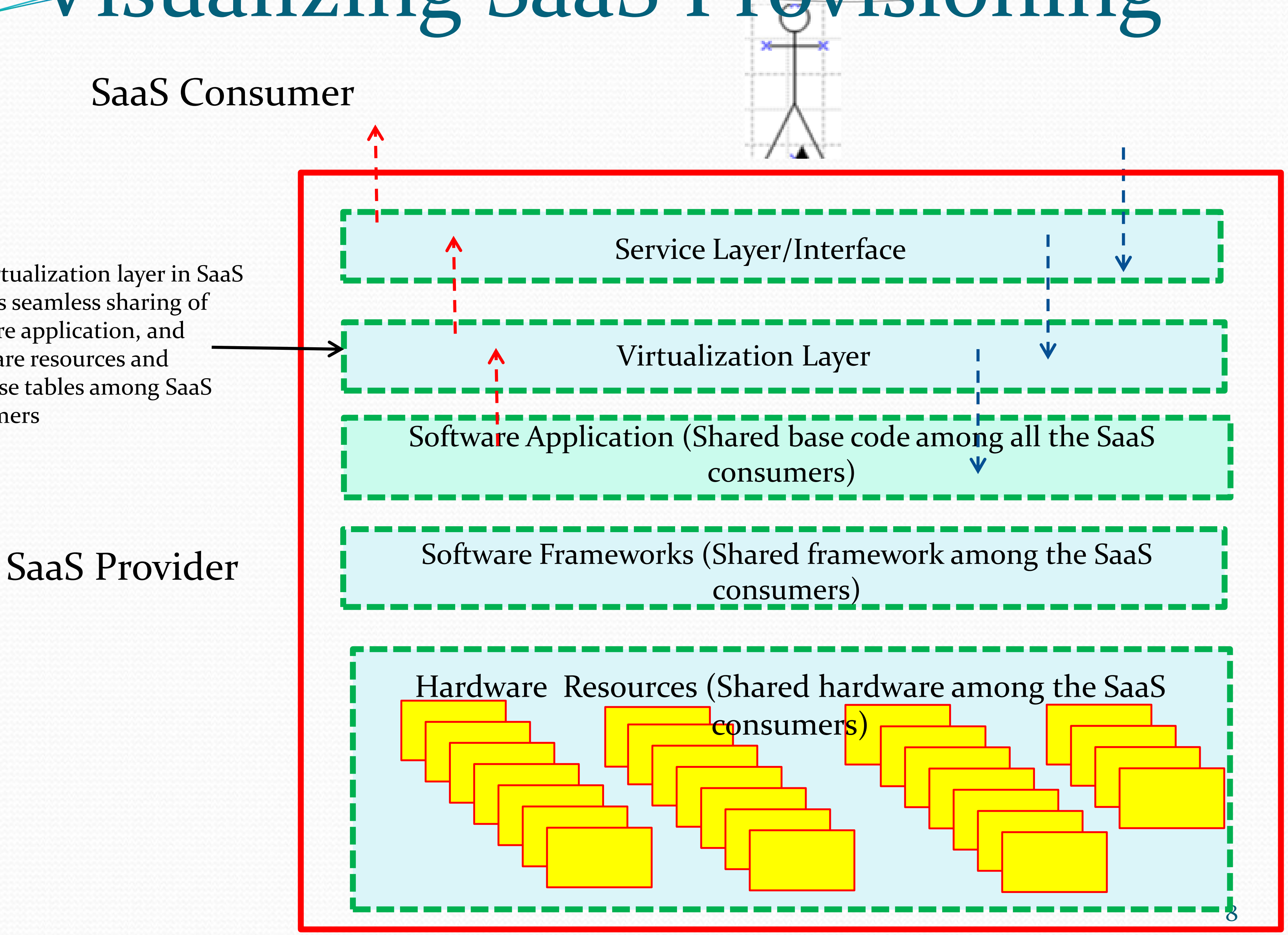
- IaaS Providers' provide and provision for infrastructure
- PaaS Providers' provide and provision computing platforms to engineer software applications
- Not of immediate value to a non-technical user wishing to consume software
- SaaS Providers' provide and provision for software applications
- Could be used directly by the end-user (no need to program the application, or configure it)
- Applications may be "*made available*" via a browser (e.g. Google Documents) and/or could be downloaded (e.g. Dropbox)

What does consuming "Software" mean?

- Software application is provided "as-a-utility" to the consumers
- Provided "on-demand" either by
  - the vendor of the software (or SaaS provider); or
  - (in some cases) by a third party (broker)
- Delivered over the internet to the consumers (known as SaaS Consumers)
- All the users share the same program code (code base)
- Deployed, upgraded, and maintained by the SaaS provider

- .....Software-as-a-Service (SaaS)

Visualizing SaaS Provisioning


SaaS Consumer
SaaS Provider
Service Layer/Interface
Virtualization Layer
Software Application (Shared base code among all the SaaS consumers)
Software Frameworks (Shared framework among the SaaS consumers)
Hardware Resources (Shared hardware among the SaaS consumers)

The virtualization layer in SaaS enables seamless sharing of software application, and hardware resources and database tables among SaaS consumers

Software-as-a-Service Definition

- The NIST Definition of SaaS

  "The capability provided to the consumer is to use the <u>provider's applications</u> running on a cloud infrastructure. The applications are accessible from various client devices through interfaces such as a web browser (e.g., web-based email), or a program interface".

  (Mell and Grance 2011)

- All the SaaS users share the same program code base
- All the SaaS users share the same computing platform
- This allows SaaS providers the ability to deliver services, on a scale, that is both economically profitable.

SaaS Characteristics

- The consumer <u>does not</u> manage or control the underlying computing platform (hardware resources and software frameworks) on which the SaaS is executing

- SaaS is always deployed outside the consumer's network or firewall
  - Managed by the SaaS provider, (usually) within their premises;
  - Security and Privacy of the stored data is provided by the SaaS provider ;
  - SaaS maintenance and updates are provided by the SaaS provider.

- SaaS is highly standardised
  - Customization is limited and completely controlled by the SaaS provider

Software-as-a-Service benefits

- Cost: Pricing model of SaaS is tied to its actual *"operational usage"*
  - In the traditional model, usually, most software applications are sold in terms of licenses. The payment (and investment) is "upfront".
  - In SaaS the user is charged on a "pay-as-you-go" basis. Some criteria for SaaS charging may include:
    - Time for which the SaaS application is being used;
    - Amount of data being stored in the SaaS application;
    - .....

- Reduction in the overall enterprise budget needed to purchase, operate and maintain software applications
  - No need to invest in procuring software licenses upfront. Limited or no upfront capital investment;
  - Reduction in the number of personnel needed to manage software applications (on an ongoing basis);
  - Makes software applications affordable for SME's, and cheaper for large organizations.

Software-as-a-Service benefits

- Rapid provisioning of (additional) SaaS Instances (compared with traditional software procurement and configuration practices):
  - Reduction in the time needed to acquire and wait for a software application to be deployed

Software-as-a-Service implications

- Limited customisation of SaaS functionality (One-size fits all)

- Overheads such as:
  - Costs associated with integrating with other in-house applications;
  - Costs and tasks associated with the migration-in and migration-out from SaaS.

- Security and Privacy of the data is managed by the SaaS provider:
  - SaaS provider's data centre may reside is different country;
  - Different data security (and privacy) laws are applicable depending on the host data centre country;
  - Local laws of the host data centre country are applicable

Comparison between on-premise applications and SaaS

- License
  - On-premise software applications are licensed with an up-front cost, depending on the number of users
  - SaaS applications are licensed based on usage ("usage-based-pricing" which is "pay-as-you-go"), such as:
    - Time-based usage;
    - Flat monthly billing
    - Data-based billing etc….

- Usage Constraints
  - SaaS applications impose restrictions on it usages to its consumers. Example usage constraints:
    - Maximum data that can be stored by a given user;
    - Size of each file being stored

SaaS and On-Premise Application Integration

- Difficult to move all the enterprise applications to SaaS, such as:
  - Large (and possibly recent) investment already made in procuring the on-premise applications;
  - Concerns on data security and/or data privacy ...etc.
- SaaS solution can be implemented in varying flavors:
  - Option 1: Complete migration to SaaS;
  - Option 2: Configuring a SaaS application to depend on data produced by on-premise applications (major applications have been moved to SaaS, and in-house applications provide some minor input to it);
    - Example: Making use of a CRM SaaS that references inventory or billing data that is residing on-premise
  - Configuring an on-premise application to depend on data produced by a SaaS application (major applications are kept in-house, and they invoke SaaS application for some minor activities)
    - Example: Making use of an on-premise payroll application that references HR data from a SaaS

SaaS and On-Premise Application Integration

- A balance between "consuming SaaS" and utilizing already existing on-premise (or in-house) applications
- This raises the need for integrating:
  - On-premise applications with SaaS; (and)
  - SaaS with On-premise applications.
- SaaS ←→ On-premise applications

Flavours of SaaS implementation

(a) Complete migration to SaaS Application

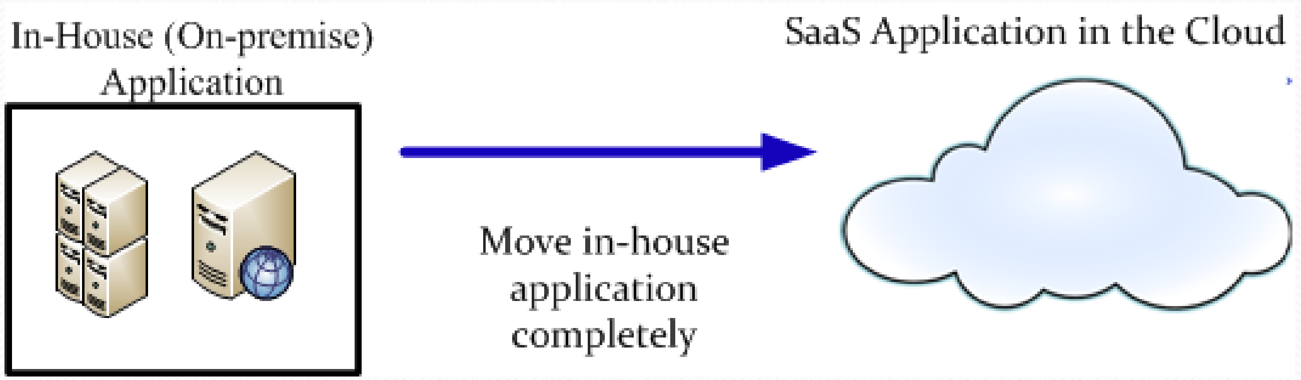
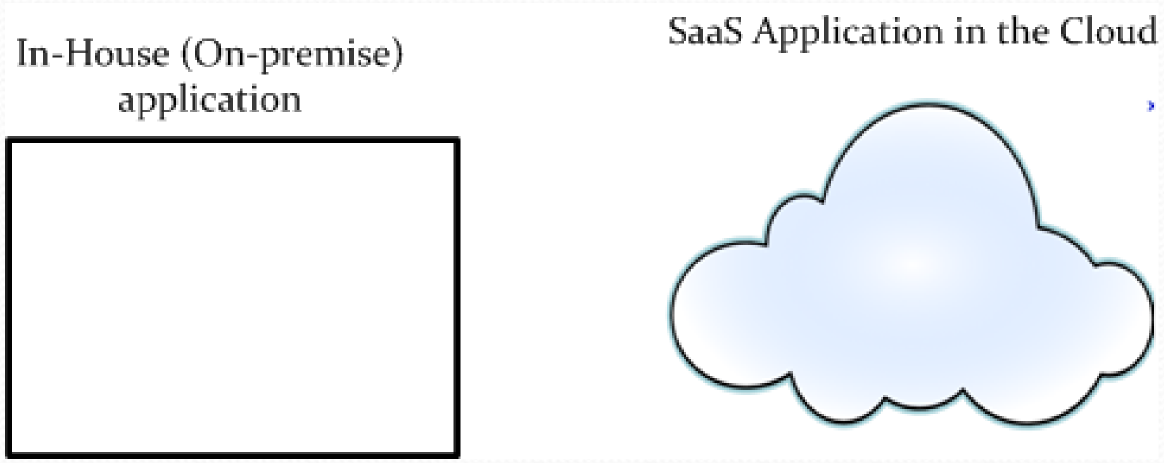

In-House (On-premise) Application
SaaS Application in the Cloud
Move in-house application completely

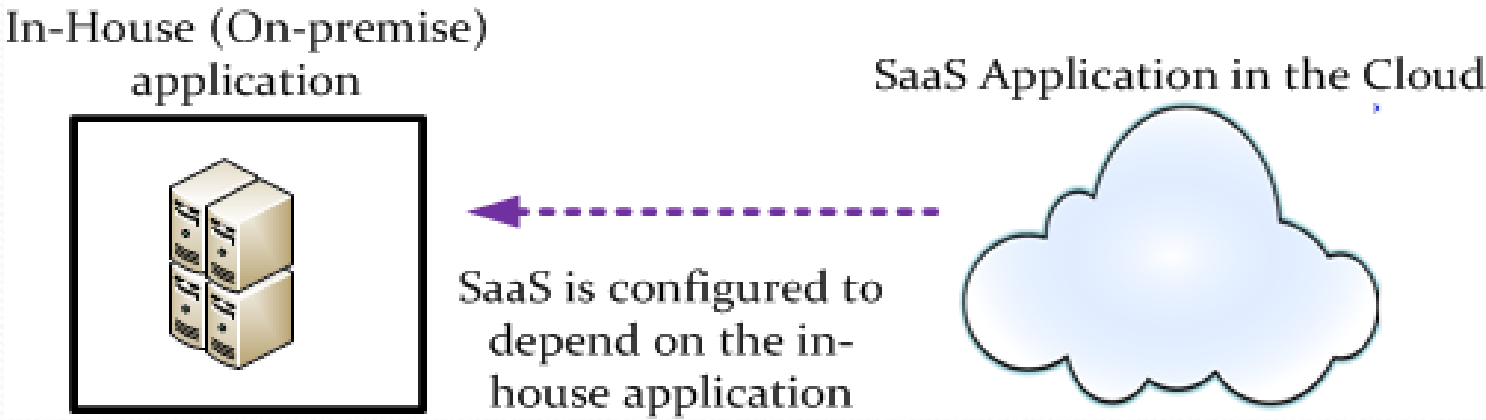
Figure (Above): Prior to complete migration to the SaaS Application

Figure (Below): After complete migration to SaaS Application

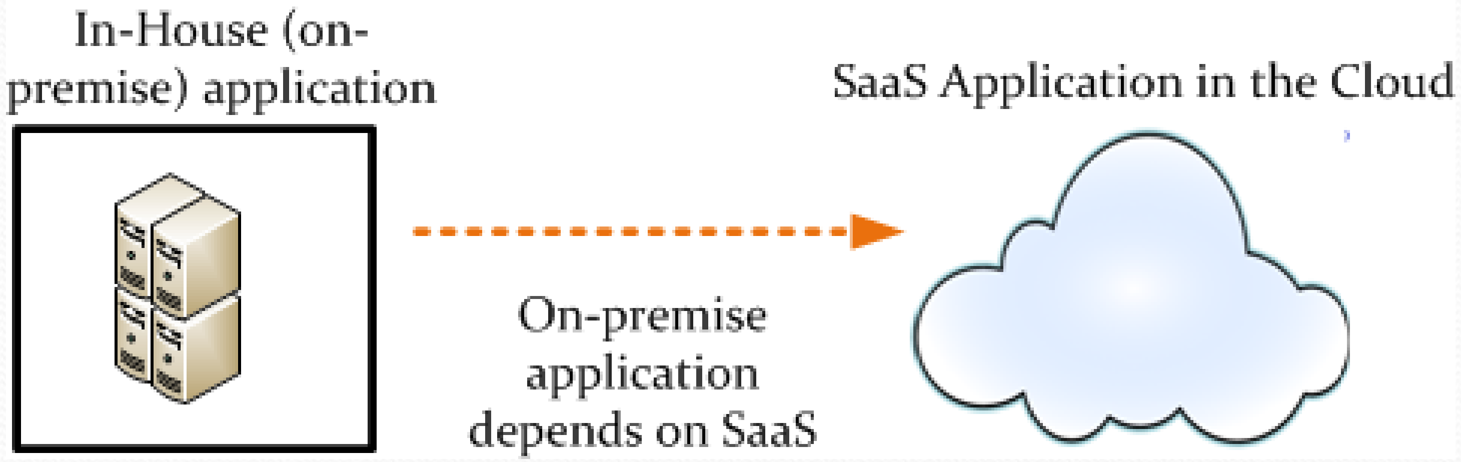

In-House (On-premise) application
SaaS Application in the Cloud

Flavours of SaaS implementation

(b) Configuring a SaaS application to depend on data produced by an on-premise applications


In-House (On-premise) application
SaaS Application in the Cloud
SaaS is configured to depend on the in-house application

(c) Configuring an on-premise application to depend on data produced by a SaaS application.


In-House (on-premise) application
SaaS Application in the Cloud
On-premise application depends on SaaS

Integration Broker

- Integration broker is responsible for integrating on-premise application with SaaS application (and vice versa).
- Responsible for managing and coordinating data dependencies and synchronization between the SaaS and the on-premise application (and vice versa)
- <u>Data Source</u>: Defined as the application where some changes to existing data have occurred, or new data has been created.
- <u>Data Sink</u>: Defined as the application where the corresponding change(s) from the data source, need to be synchronized.
- Integration broker is responsible for:
  - Receiving the data from the data source;
  - Determining where the corresponding data needs to be sent;
  - Routing the data to the appropriate application data sink;
  - Transforming the data to the appropriate format of the data sink.

Generic Architecture of an integration broker

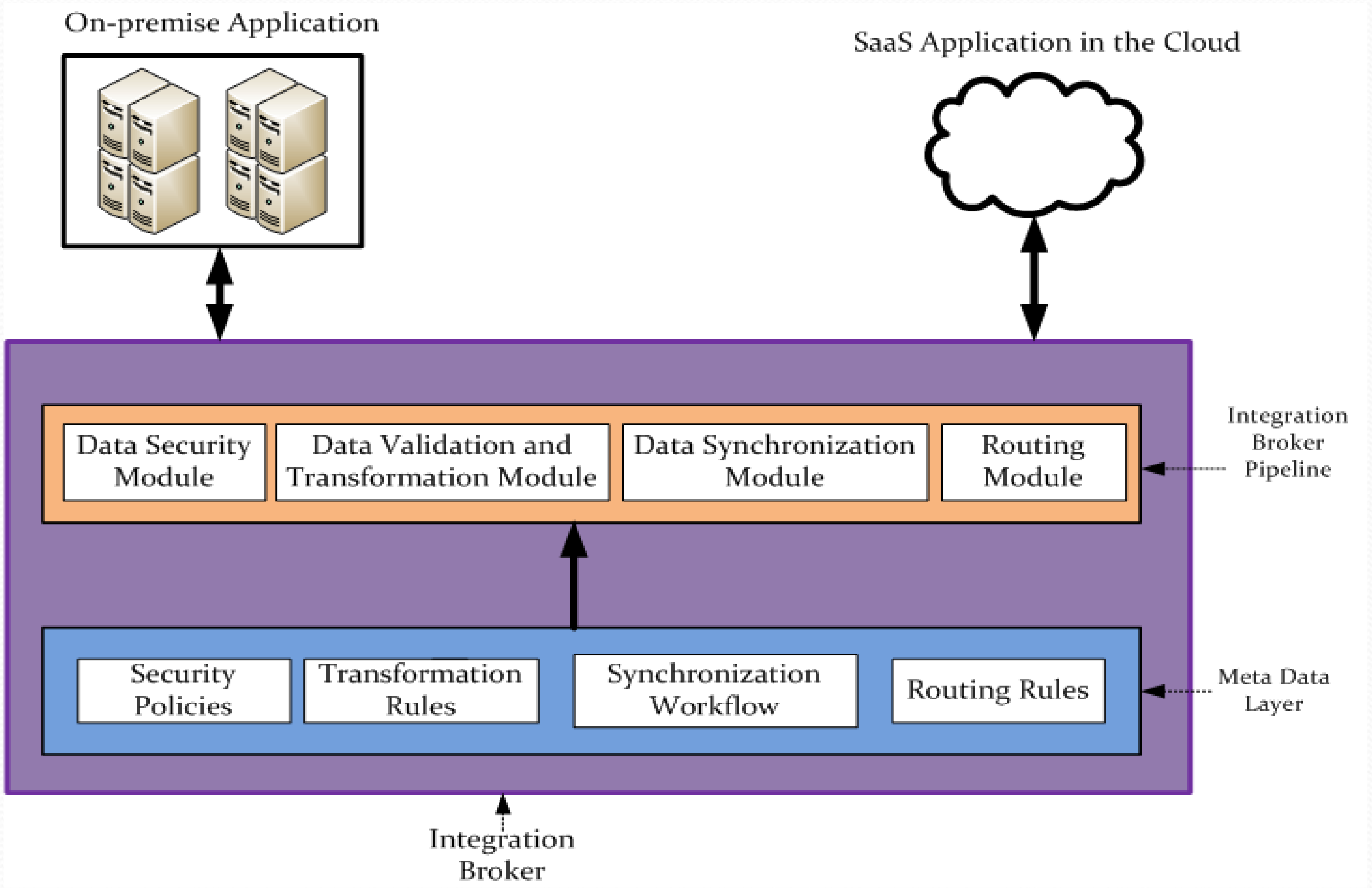
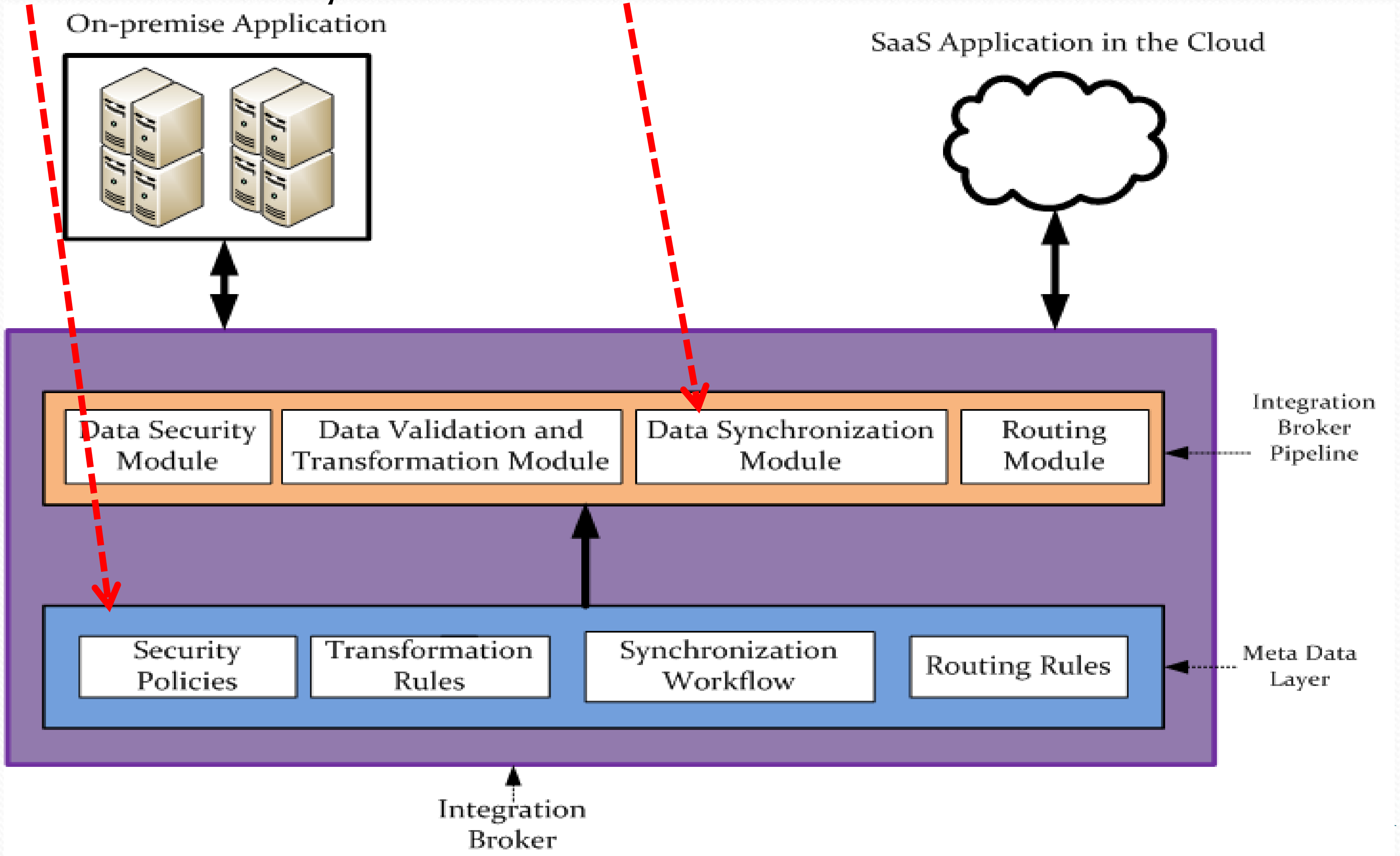


Figure : Generic Architecture of an integration Broker

Key elements of the integration broker

- Integration broker pipeline
- Meta Data Layer



Integration Broker Pipeline

- Performs all the actual integration activities between the data source and the data sink.
- It is modular in architecture, and is comprised of four modules.
- **<u>Module 1</u>**: Data Security Module
    - Responsible for carrying out security-related functions over the incoming data. The tasks performed by this module are:
        - Authenticating the data from the data source;
        - Check the (incoming) data for viruses or spam;
        - Apply decryption algorithms (if the incoming data is encrypted).

Integration Broker Pipeline

- **<u>Module 2</u>:** Data Validation and Transformation Module
  - Responsible for determining if the incoming data is in a valid format, and if needed transform it appropriately. The tasks performed by this module are:
    - Determine if the incoming data (from the data source) is in a valid format
    - Reject non-compliant data
    - Transform compliant data to the (corresponding) format of the data sink
    - Rules or standards govern the transformation process

Integration Broker Pipeline

- **<u>Module 3</u>:** Data Synchronization Module
  - Responsible for orchestrating the consistency of the data between the data source and data sink. The task performed by this module is to:
    - Inform the counterpart of new data or changes to the existing data
    - Three ways in which the synchronization could be achieved

Synchronization mechanisms

- "*Polling*" synchronization mechanism:
  - In this mechanism a node (SaaS application, or on-premise application) queries the other of any changes (a.k.a poll query);
  - The poll query is usually time-based, and is done at recurring intervals of time;
  - If the response to the query is "yes", then the polling node requests for the changes to the propagated;
  - Does not scale very well since each node needs to regularly keep on querying all the other nodes.
- Disadvantages of polling mechanism:
  - Not suitable for real-time applications;
  - A large number of messages are exchanged between the source and sink;
  - Does not scale very well.

Pictorial representation of polling synchronization mechanism

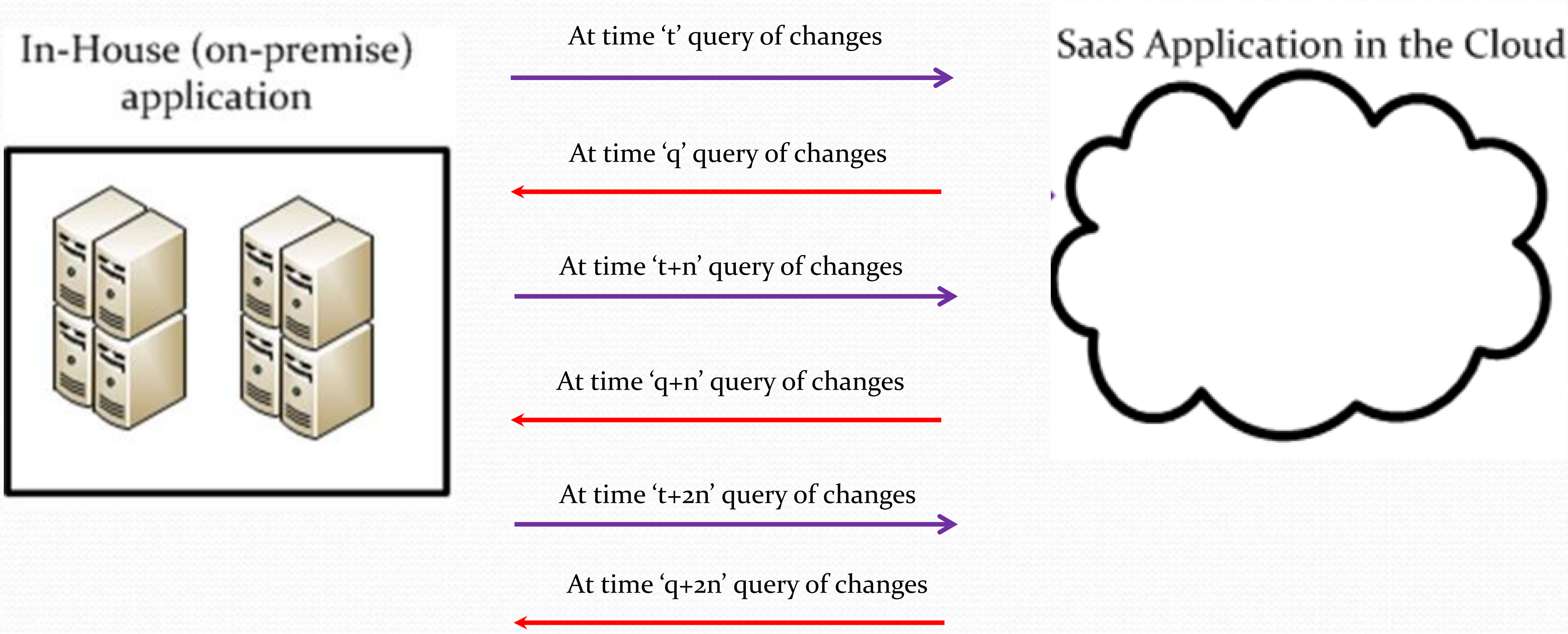

In-House (on-premise) application
At time 't' query of changes
At time 'q' query of changes
At time 't+n' query of changes
At time 'q+n' query of changes
At time 't+2n' query of changes
At time 'q+2n' query of changes
SaaS Application in the Cloud

Figure : Pictorial representation of polling synchronization mechanism

Synchronization mechanisms

- "*Push*" Synchronization mechanism
  - The data source notifies the data sink of changes ("push" the changes to the data sink)
  - No prompt or query (unlike polling)
  - The push is event-based (as soon as the change occurs)
  - Event-based push mechanism is good to ensure data consistency in real-time applications

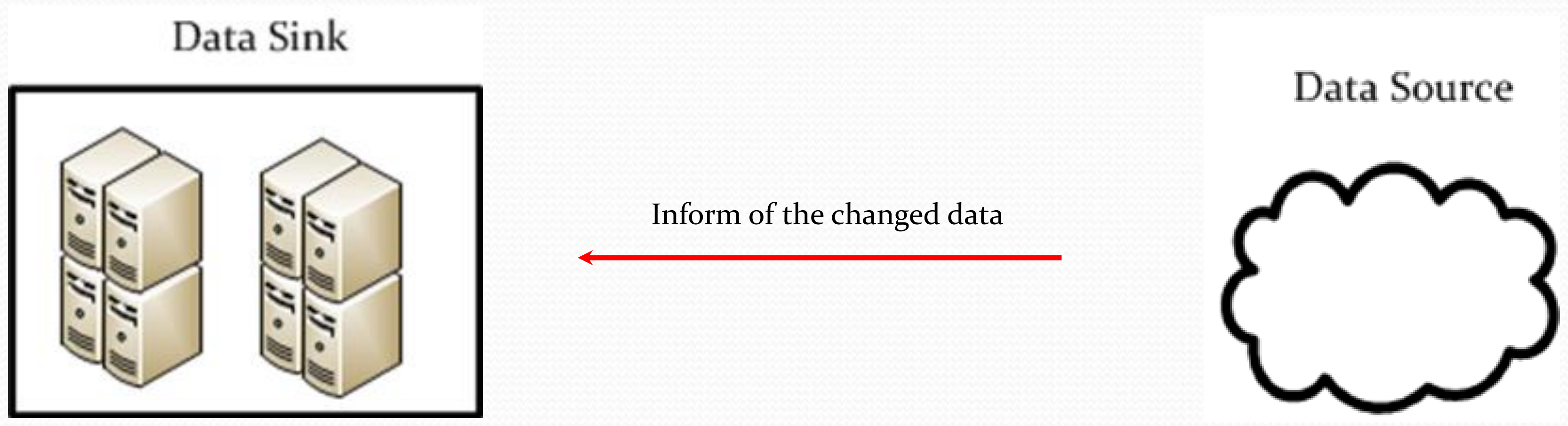
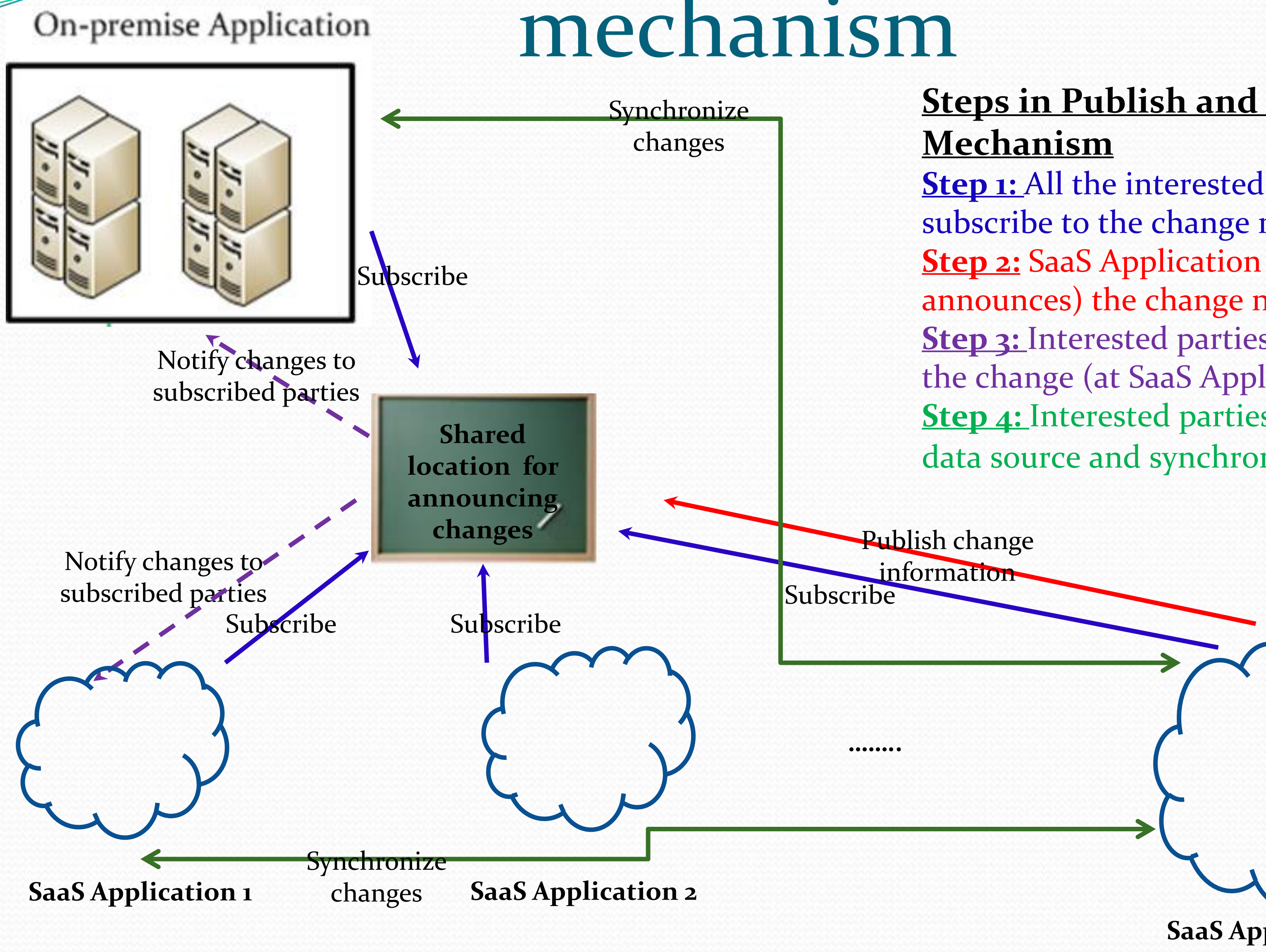


Figure : Pictorial representation of Push-based synchronization mechanism

Synchronization mechanisms

- “*Publish-and-Subscribe*” synchronization mechanism
  - The data source announces the change (“publishes the change”) at a commonly shared location;
  - Data sinks subscribe to changes from data source(s) (“subscribe to changes”)
    - The data sinks may subscribe to changes from selected data sources only;
    - Multiple data sinks may subscribe to changes from a given data source
  - When the data source announces the change, this information is correspondingly passed to all the subscribed data sinks
  - Interested data sinks contact the relevant data source and request for changes to be propagated
  - Scales very well to multiple data sinks

Publish and Subscribe Synchronization mechanism



Steps in Publish and Subscribe Mechanism

Step 1: All the interested applications subscribe to the change notification.

Step 2: SaaS Application N publishes (or announces) the change notification.

Step 3: Interested parties are informed of the change (at SaaS Application N)

Step 4: Interested parties contact the data source and synchronize the changes

Figure : Pictorial representation of Publish and Subscribe synchronization mechanism

Integration Broker

- **Module 4**: Routing Module
  - Responsible for routing the incoming data to the data sink;
  - Rules (pre-programmed rules or using intelligent mechanisms)

Meta Data Layer

- Specify and define “how” to perform integration activities
- Encompasses the logic to enable integration activities in the form of rules
  - Security operations
    - How to check for viruses in the incoming data?
    - How to cleanse data?
    - How to decrypt incoming data?
  - Validate and transform operations
    - Is the incoming data in a valid format?
    - Do we cater for the transformation of the incoming data?
    - How to transform data from one format to another?
  - Synchronization operations
    - Rules to determine how data changes are propagated
  - Routing operations
    - Routing rules specify the destination of the incoming data

Example of SaaS

- Example of web-based SaaS - Google Documents
  - Provide a number of office applications "*as-a-service*" to the end user.

- Example of download and install SaaS - Dropbox
  - Data storage, data sharing, and data synchronization SaaS application (https://www.dropbox.com/)
  - Could be used as a web-application, or a client installed locally

Summary

- Software as a Service (SaaS)
  - SaaS Capabilities
  - SaaS Benefits
  - SaaS Implications
- Traditional Applications vs. SaaS Applications
- Architecture for SaaS integration with in-house applications

Reading

Books

1. Rhoton, J. (2010), Cloud computing explained, Recursive Press, UK – Chapter 5

Reading

Papers, Website

1. Carraro, G., and Chong, F. (2006), Software as a Service (SaaS): An Enterprise Perspectives, Microsoft Corporation.

2. P. Mell and T. Grance. (2011), The NIST Definition of Cloud Computing.

3. Hai, H., and Sakoda, S. (2009), SaaS Integration Best Practices, Fujitsu Sci. Tech. J., vol 45, no. 3.