Unit 7 Software Architecture

Unit Outcomes. Here you will learn

- what is meant by architecture in information systems development
- various architectural styles, including layers, MVC and SOA
- how to apply the Model-View-Controller (MVC) architecture

Further Reading: Bennett et al. Ch13 & Braude and Bernstein Ch18

based on Bennett, McRobb & Farmer (2010) and Braude & Bernstein (2011)

Contents

- Architecture
- Relationship between Architecture and Design
- Enterprise Architecture
- System Architecture
  - Key Definitions
  - The role of a System Architect
- Software Architecture
  - What is it?
  - Architecture $\neq$ Framework
  - Framework
- Subsystems
- Layering and Partitioning
- Layered Architecture
  - Schematic of a Layered Architecture
  - Example: Open Systems Interconnection model
  - Simple Layered Architecture
  - 3 or 4 Layers?
- Tiered Architecture
  - Layered architectures Vs Tiered architectures
  - Three-tier Architecture
- Partitioning
  - Partitioned Subsystems
- Model-View-Controller (MVC)
  - An issue with some architectures
  - A solution: the MVC architecture
  - Bennett et al. (2010)
  - Eckstein (2007)
- A simple calculator in MVC
  - UML Class Diagram
  - Roles of each component
  - CalcMVC (Swartz, 2004)
  - CalcModel
  - AbstractModel
  - CalcView
  - CalcController
  - Single Model, Multiple Views
- Subsystem Communications
  - Client-server communication
  - Peer-to-peer communication
- Service Oriented Architecture
  - Web Services
  - Operations in SOA
  - A Holiday Booking Scenario
  - Common Principles
  - Why SOA?
- References

Architecture
RIBA (2005)

"Architects are trained to take your brief and can see the big picture – they look beyond your immediate requirements to design flexible buildings that will adapt with the changing needs of your business.
Architects solve problems creatively – when they are involved at the earliest planning stage, they gain more opportunities to understand your business, develop creative solutions, and propose ways to reduce costs."

Royal Institute of British Architects (2005)

Architecture
What is architecture?

- *"Architecture is the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution."* (IEEE, 2000)

client-server

- An architectural style *"defines a family of systems in terms of a pattern of structural organization. More specifically, an architectural style defines a vocabulary of components and connector types, and a set of constraints on how they can be combined."* (Shaw and Garlan, 1996)

Architecture
What is architecture?

- *"Architecture is the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution."* (IEEE, 2000)

- An architectural style *"defines a family of systems in terms of a pattern of structural organization. More specifically, an architectural style defines a vocabulary of components and connector types, and a set of constraints on how they can be combined."*
  (Shaw and Garlan, 1996)

Architecture
An Architecture. . . (Eeles, 2006)

- defines structure
  - ➟ What are the core components?
  - ➟ How do they relate to each other?
- defines behaviour
  - ➟ E.g. How do the core components interact with each other?
- focuses on significant elements
  - ➟ E.g. elements associated with essential behaviour and/or address significant design qualities such as reliability and scalability.

Architecture
An Architecture. . . (Eeles, 2006)

- defines structure
  - ➟ What are the core components?
  - ➟ How do they relate to each other?
- defines behaviour
  - ➟ E.g. How do the core components interact with each other?
- focuses on significant elements
  - ➟ E.g. elements associated with essential behaviour and/or address significant design qualities such as reliability and scalability.

Architecture
An Architecture. . . (Eeles, 2006)

- defines structure
  - ➟ What are the core components?
  - ➟ How do they relate to each other?
- defines behaviour
  - ➟ E.g. How do the core components interact with each other?
- focuses on significant elements
  - ➟ E.g. elements associated with essential behaviour and/or address significant design qualities such as reliability and scalability.

Architecture
An Architecture. . . (Eeles, 2006)

- balances stakeholders' needs
  - ➟ Different types of stakeholder often have conflicting needs.
- embodies decisions based on rationale
- may conform to an *architectural style*
  - ➟ Recall that an architectural style "*defines a family of systems in terms of a pattern of structural organization*".
- has a particular scope
  - ➟ E.g. enterprise, system, software, hardware, organisational, information

Architecture
An Architecture. . . (Eeles, 2006)

- balances stakeholders' needs
  - ➠ Different types of stakeholder often have conflicting needs.
- embodies decisions based on rationale
- may conform to an *architectural style*
  - ➠ Recall that an architectural style "*defines a family of systems in terms of a pattern of structural organization*".
- has a particular scope
  - ➠ E.g. enterprise, system, software, hardware, organisational, information

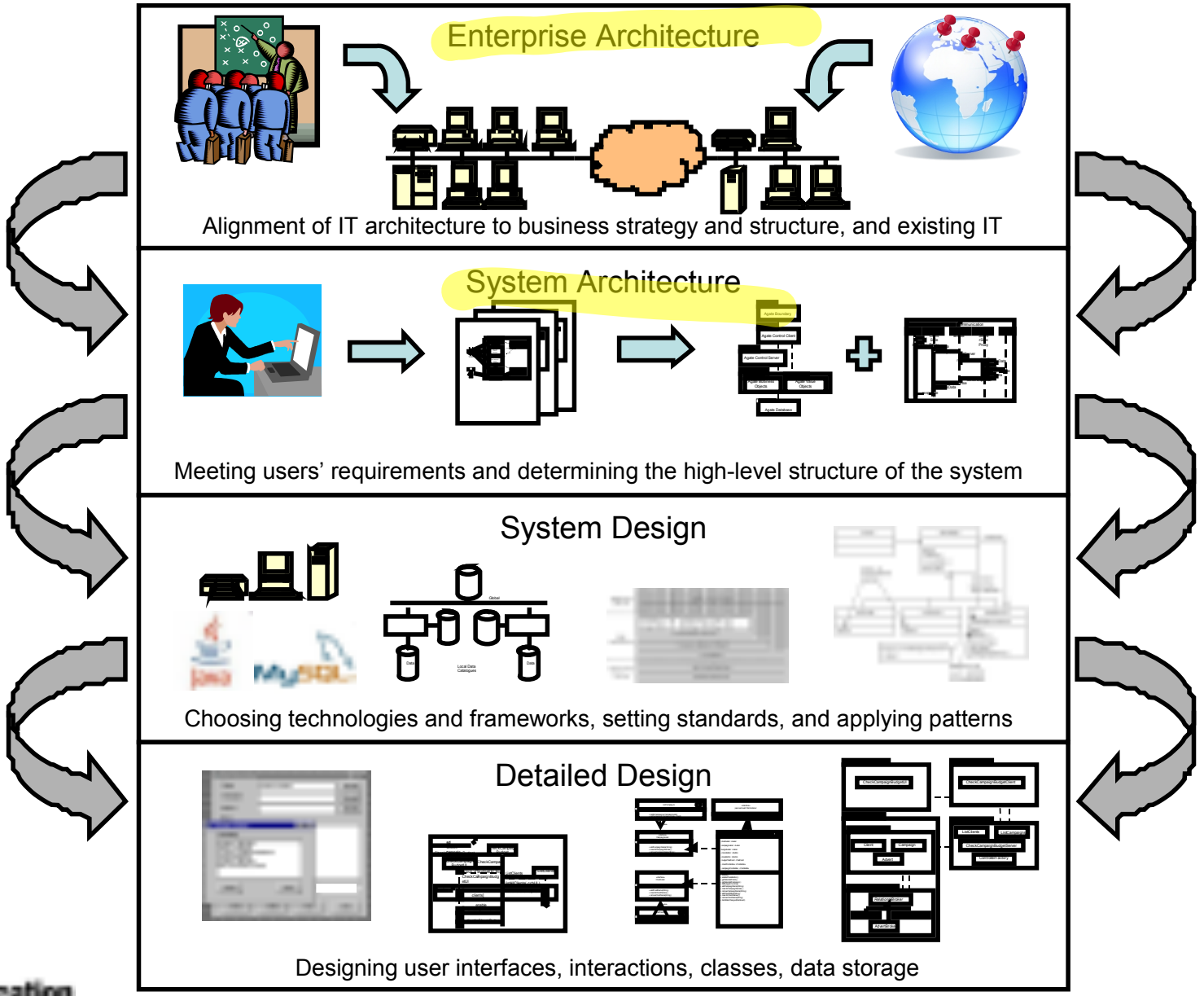
Architecture
An Architecture. . . (Eeles, 2006)

- balances stakeholders' needs
  - ➠ Different types of stakeholder often have conflicting needs.
- embodies decisions based on rationale
- may conform to an *architectural style*
  - ➠ Recall that an architectural style "*defines a family of systems in terms of a pattern of structural organization*".
- has a particular scope
  - ➠ E.g. enterprise, system, software, hardware, organisational, information

Architecture
An Architecture. . . (Eeles, 2006)

- balances stakeholders' needs
  - ➠ Different types of stakeholder often have conflicting needs.
- embodies decisions based on rationale
- may conform to an *architectural style*
  - ➠ Recall that an architectural style "*defines a family of systems in terms of a pattern of structural organization*".
- has a particular scope
  - ➠ E.g. enterprise, system, software, hardware, organisational, information

Relationship between Architecture and Design



Enterprise Architecture
Bennett et al. (2010, Section 13.4.2)

- **`Enterprise architecture`** concerns:
  - the modelling of the *business*,
  - the way the *enterprise* conducts business and
  - how the *information systems* are intended to support the business.
- *Typical questions* to be considered include:
  - How does the system overlap with other systems in the organisation?
  - How will the system need to interface with other systems?
  - Will the system help the organisation to achieve its goals?
  - Is the cost of the system justified?

Enterprise Architecture
Bennett et al. (2010, Section 13.4.2)

- **`Enterprise architecture`** concerns:
  - the modelling of the *business*,
  - the way the *enterprise* conducts business and
  - how the *information systems* are intended to support the business.
- *Typical questions* to be considered include:
  - How does the system overlap with other systems in the organisation?
  - How will the system need to interface with other systems?
  - Will the system help the organisation to achieve its goals?
  - Is the cost of the system justified?

System Architecture
Key Definitions : Garland & Anthony (2003) and IEEE (2000)

- *System* is a set of *co-operating components* organised to accomplish a specific function or a set of functions.
- *Architecture* is the *fundamental organization* of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.
- *Architectural Description* is a set of products that *document* the architecture.
- *Architectural View* is a *representation* of a particular system or part of a system from a particular perspective.

System Architecture
Key Definitions : Garland & Anthony (2003) and IEEE (2000)

- *System* is a set of *co-operating components* organised to accomplish a specific function or a set of functions.
- *Architecture* is the *fundamental organization* of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.
- *Architectural Description* is a set of products that *document* the architecture.
- *Architectural View* is a *representation* of a particular system or part of a system from a particular perspective.

System Architecture
Key Definitions : Garland & Anthony (2003) and IEEE (2000)

- *System* is a set of *co-operating components* organised to accomplish a specific function or a set of functions.
- *Architecture* is the *fundamental organization* of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.
- *Architectural Description* is a set of products that *document* the architecture.
- *Architectural View* is a *representation* of a particular system or part of a system from a particular perspective.

System Architecture
Key Definitions : Garland & Anthony (2003) and IEEE (2000)

- *System* is a set of *co-operating components* organised to accomplish a specific function or a set of functions.
- *Architecture* is the *fundamental organization* of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.
- *Architectural Description* is a set of products that *document* the architecture.
- *Architectural View* is a *representation* of a particular system or part of a system from a particular perspective.

System Architecture
Key Definitions

- *Architectural Viewpoint* is a *template* that describes how to create and use an architectural view. A viewpoint includes a name, stakeholders, concerns addressed by the viewpoint, and the modelling and analytic conventions.

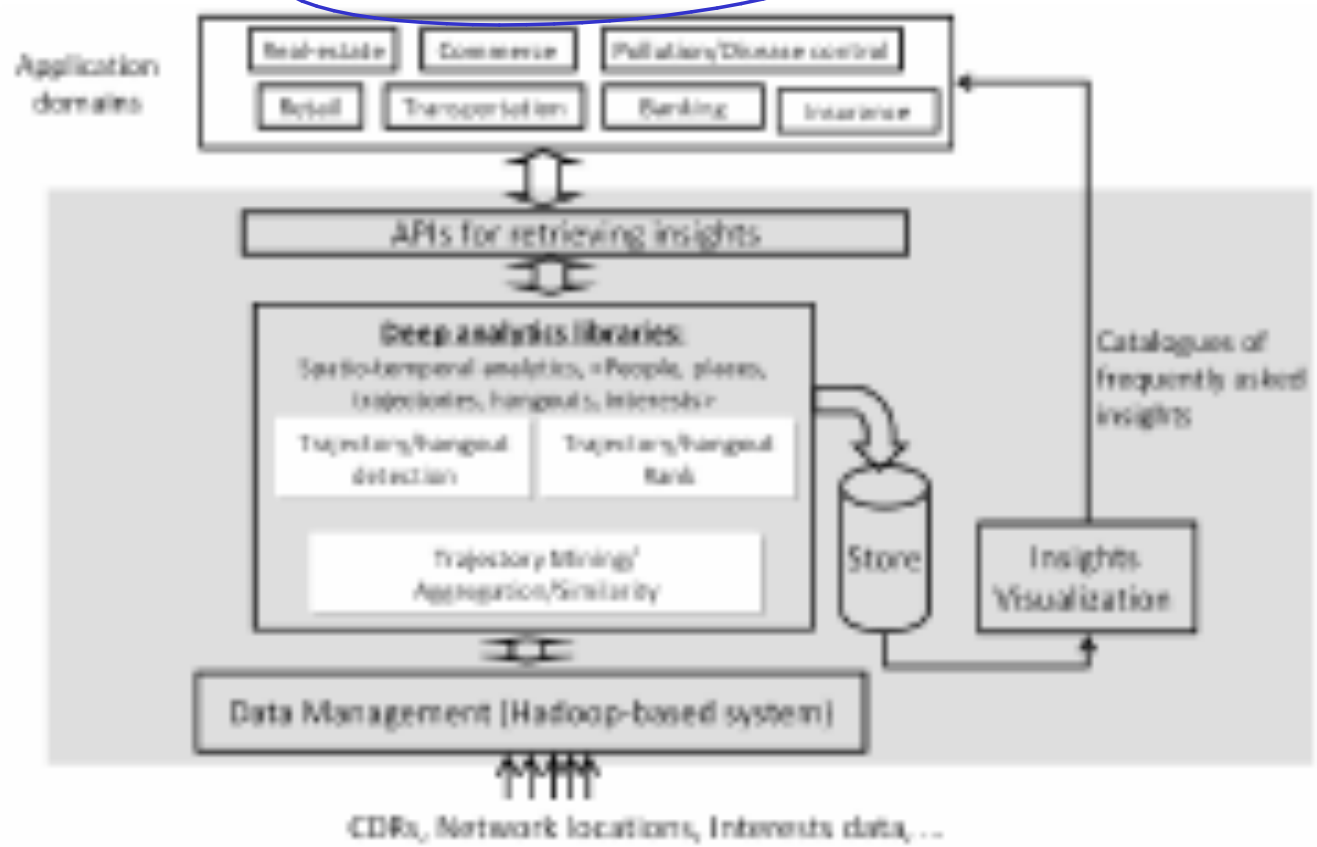
  ➠ System architecture describes the hardware and software elements and interactions between these elements within a system.

System Architecture
Key Definitions

- *Architectural Viewpoint* is a *template* that describes how to create and use an architectural view. A viewpoint includes a name, stakeholders, concerns addressed by the viewpoint, and the modelling and analytic conventions.

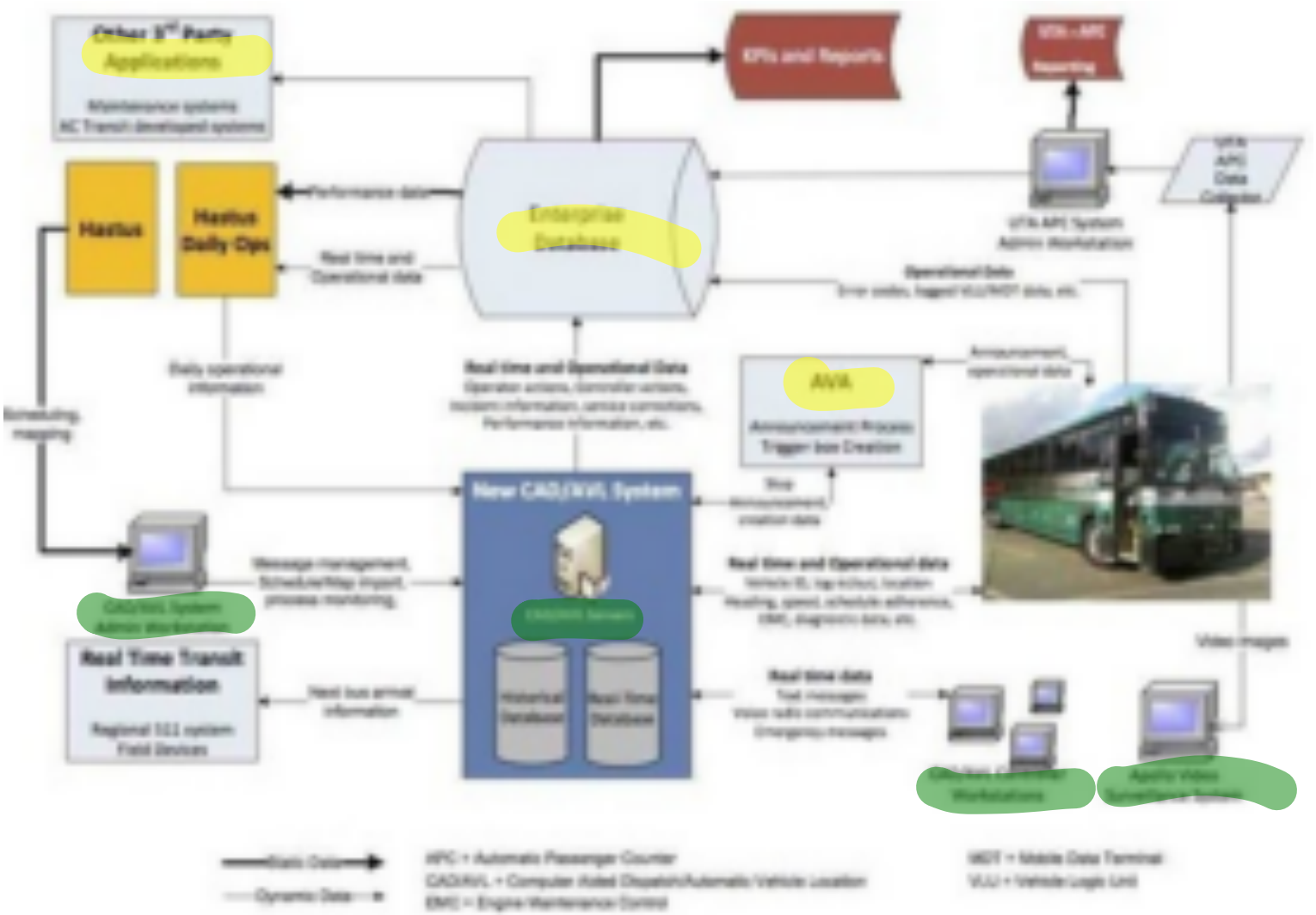
  ➠ System architecture describes the hardware and software elements and interactions between these elements within a system.

System Architecture
Key Definitions : an example



Source: People in motion: Spatio-temporal analytics on Call Detail Records - Scientific Figure on ResearchGate. Available from: `https://www.researchgate.net/System-Architecture_fig1_271555314` (accessed 27 Oct, 2018)

System Architecture
Key Definitions : an example



Source: `https://www.quora.com/What-is-an-architecture-diagram`

System Architecture

`System architects`:

- act on behalf of the client;
- address the *big picture*;
- ensure that the required *qualities* of the system are accounted for in the design;
- *solve* problems;
- ensure the required features are provided at the right cost.

System Architecture

`System architects`:

- act on behalf of the client;
- address the *big picture*;
- ensure that the required *qualities* of the system are accounted for in the design;
- *solve* problems;
- ensure the required features are provided at the right cost.

System Architecture

System architects:

- act on behalf of the client;
- address the *big picture*;
- ensure that the required *qualities* of the system are accounted for in the design;
- *solve* problems;
- ensure the required features are provided at the right cost.

System Architecture

System architects:

- act on behalf of the client;
- address the *big picture*;
- ensure that the required *qualities* of the system are accounted for in the design;
- *solve* problems;
- ensure the required features are provided at the right cost.

System Architecture

`System architects`:

- act on behalf of the client;
- address the *big picture*;
- ensure that the required *qualities* of the system are accounted for in the design;
- *solve* problems;
- ensure the required features are provided at the right cost.

Software Architecture
What is it?

- *"Software architecture is the organization of a system in terms of its software components, including subsystems and the relationships and interactions among them, and the principles that guide the design of that software system."*

  Bennett et al. (2010)

- *"A software architecture describes the overall components of an application and how they relate to each other. Its design goals, include sufficiency, understandability, modularity, high cohesion, low coupling, robustness, flexibility, reusability, efficiency, and reliability.*

  Braude and Bernstein (2011)

Software Architecture
What is it?

- *"Software architecture is the organization of a system in terms of its software components, including subsystems and the relationships and interactions among them, and the principles that guide the design of that software system."*

Bennett et al. (2010)

- *"A software architecture describes the overall components of an application and how they relate to each other. Its design goals, include sufficiency, understandability, modularity, high cohesion, low coupling, robustness, flexibility, reusability, efficiency, and reliability.*

Braude and Bernstein (2011)

Software Architecture
What is it?

- *An architecture is the set of significant decisions about the organization of a software system, the selection of the structural elements and their interfaces by which the system is composed, together with their behavior as specified in the collaborations among those elements, the composition of these structural and behavioral elements into progressively larger subsystems, and the architectural style that guides this organization—these elements and their interfaces, their collaborations, and their composition.*

Reproduced by Larman (2005, p.200) from Booch et al. (1997)

Software Architecture
What is it?

- Software architecture is something to do with the large scale *—the Big Ideas in the motivations, constraints, organization, patterns, responsibilities, and connections of a system (a system of systems).*

Larman (2005, p.201)

Software Architecture
Architecture $\neq$ Framework

- The terms *architecture* and *framework* have sometimes been used interchangeably, but that is *inappropriate*.
- *Architecture* $\neq$ *Framework*
- *"A framework, sometimes called a library, is a collection of software artifacts usable by several different applications. These artifacts are typically implemented as classes, together with the software required to utilize them. A framework is a kind of common denominator for a family of applications. The Java APIs (3D, 2D, Swing, etc.) are frameworks.*

  Braude and Bernstein (2011, p.358)

Can you name some Java frameworks?

Software Architecture
Architecture ≠ Framework

- The terms *architecture* and *framework* have sometimes been used interchangeably, but that is *inappropriate*.
- *Architecture ≠ Framework*
- *"A framework, sometimes called a library, is a collection of software artifacts usable by several different applications. These artifacts are typically implemented as classes, together with the software required to utilize them. A framework is a kind of common denominator for a family of applications. The Java APIs (3D, 2D, Swing, etc.) are frameworks.*

  Braude and Bernstein (2011, p.358)

Can you name some Java frameworks?

Software Architecture
Architecture $\neq$ Framework

- The terms *architecture* and *framework* have sometimes been used interchangeably, but that is *inappropriate*.
- *Architecture* $\neq$ *Framework*
- *"A framework, sometimes called a library, is a collection of software artifacts usable by several different applications. These artifacts are typically implemented as classes, together with the software required to utilize them. A framework is a kind of common denominator for a family of applications. The Java APIs (3D, 2D, Swing, etc.) are frameworks.*

Braude and Bernstein (2011, p.358)

Can you name some Java frameworks?

Software Architecture
Architecture ≠ Framework

- The terms *architecture* and *framework* have sometimes been used interchangeably, but that is *inappropriate*.
- *Architecture ≠ Framework*
- *"A framework, sometimes called a library, is a collection of software artifacts usable by several different applications. These artifacts are typically implemented as classes, together with the software required to utilize them. A framework is a kind of common denominator for a family of applications. The Java APIs (3D, 2D, Swing, etc.) are frameworks.*

Braude and Bernstein (2011, p.358)

Can you name some Java frameworks?
Java Collections Framework (JCF), Java Media Framework (JMF), JavaFX — GUI

↑ data structure

Framework : Examples

More examples of frameworks in Java:

- Apache Structs (`https://struts.apache.org/`)

  "*Apache Struts is a free, open-source, MVC framework for creating elegant, modern Java web applications.*"

- Spring (`http://spring.io/`)

  "*Spring helps development teams everywhere build simple, portable, fast and flexible JVM-based systems and applications.*"

Framework : Examples

More examples of frameworks in Java:

- Apache Structs (`https://struts.apache.org/`)

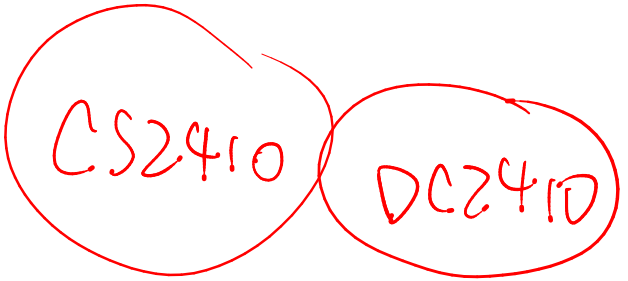
  "*Apache Struts is a free, open-source, MVC framework for creating elegant, modern Java web applications.*"

- Spring (`http://spring.io/`)

  "*Spring helps development teams everywhere build simple, portable, fast and flexible JVM-based systems and applications.*"

Framework : Examples

Some frameworks for PHP:



- Laravel (`http://laravel.com/`)
  "*The PHP Framework For Web Artisans*"

- Drupal (`https://www.drupal.org/`)
  "*Launch, manage, and scale ambitious digital experiences—with the flexibility to build great websites or push beyond the browser. Proudly open source.*"
  . . . a PHP based, open source, content management framework.

- CakePHP (`https://cakephp.org/`)
  "*A modern PHP 5.5+ framework offering a flexible database access layer and a powerful scaffolding system that makes building both small and complex systems simpler (and) easier.*"

Framework : Examples

Some frameworks for PHP:

- Laravel (`http://laravel.com/`)
  "*The PHP Framework For Web Artisans*"

- Drupal (`https://www.drupal.org/`)
  "*Launch, manage, and scale ambitious digital experiences—with the flexibility to build great websites or push beyond the browser. Proudly open source.*"
  . . . a PHP based, open source, content management framework.

- CakePHP (`https://cakephp.org/`)
  "*A modern PHP 5.5+ framework offering a flexible database access layer and a powerful scaffolding system that makes building both small and complex systems simpler (and) easier.*"

Framework : Examples

Some frameworks for PHP:

- Laravel (`http://laravel.com/`)
  "*The PHP Framework For Web Artisans*"

- Drupal (`https://www.drupal.org/`)
  "*Launch, manage, and scale ambitious digital experiences—with the flexibility to build great websites or push beyond the browser. Proudly open source.*"
  . . . a PHP based, open source, content management framework.

- CakePHP (`https://cakephp.org/`)
  "*A modern PHP 5.5+ framework offering a flexible database access layer and a powerful scaffolding system that makes building both small and complex systems simpler (and) easier.*"

Framework : Examples

Some frameworks for JavaScript, HTML, CSS:

- JQuery (`https://jquery.com/`)

  "*jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers.*"

- Bootstrap (`http://getbootstrap.com/`)

  "*the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web*"

- React (`https://facebook.github.io/react/`)

  "*A JavaScript library for building user interfaces.*"

Framework : Examples

Some frameworks for JavaScript, HTML, CSS:

- JQuery (`https://jquery.com/`)

  "*jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers.*"

- Bootstrap (`http://getbootstrap.com/`)

  "*the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web*"

- React (`https://facebook.github.io/react/`)

  "*A JavaScript library for building user interfaces.*"

Framework : Examples

Some frameworks for JavaScript, HTML, CSS:

- JQuery (`https://jquery.com/`)

  "*jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers.*"

- Bootstrap (`http://getbootstrap.com/`)

  "*the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web*"

- React (`https://facebook.github.io/react/`)

  "*A JavaScript library for building user interfaces.*"

Subsystems

- A *subsystem* typically groups together elements of the system that share some *common properties*.
- An object-oriented subsystem encapsulates a coherent set of responsibilities in order to ensure that it has integrity and can be maintained.
- The subdivision of an information system into subsystems has the following *advantages*:
  - It produces *smaller units* of development.
  - It helps to *maximize reuse* at the component level.
  - It helps the developers to *cope with complexity*.
  - It improves *maintainability*.
  - It aids *portability*.

Subsystems

- A *subsystem* typically groups together elements of the system that share some *common properties*.
- An object-oriented subsystem encapsulates a coherent set of responsibilities in order to ensure that it has integrity and can be maintained.
- The subdivision of an information system into subsystems has the following *advantages*:
  - It produces *smaller units* of development.
  - It helps to *maximize reuse* at the component level.
  - It helps the developers to *cope with complexity*.
  - It improves *maintainability*.
  - It aids *portability*.

Subsystems

- A *subsystem* typically groups together elements of the system that share some *common properties*.
- An object-oriented subsystem encapsulates a coherent set of responsibilities in order to ensure that it has integrity and can be maintained.
- The subdivision of an information system into subsystems has the following *advantages*:
  - It produces *smaller units* of development.
  - It helps to *maximize reuse* at the component level.
  - It helps the developers to *cope with complexity*.
  - It improves *maintainability*.
  - It aids *portability*.

Subsystems

- A *subsystem* typically groups together elements of the system that share some *common properties*.
- An object-oriented subsystem encapsulates a coherent set of responsibilities in order to ensure that it has integrity and can be maintained.
- The subdivision of an information system into subsystems has the following *advantages*:
  - It produces *smaller units* of development.
  - It helps to *maximize reuse* at the component level.
  - It helps the developers to *cope with complexity*.
  - It improves *maintainability*.
  - It aids *portability*.

Subsystems

- A *subsystem* typically groups together elements of the system that share some *common properties*.
- An object-oriented subsystem encapsulates a coherent set of responsibilities in order to ensure that it has integrity and can be maintained.
- The subdivision of an information system into subsystems has the following *advantages*:
  - It produces *smaller units* of development.
  - It helps to *maximize reuse* at the component level.
  - It helps the developers to *cope with complexity*.
  - It improves *maintainability*.
  - It aids *portability*.

Subsystems

- A *subsystem* typically groups together elements of the system that share some *common properties*.
- An object-oriented subsystem encapsulates a coherent set of responsibilities in order to ensure that it has integrity and can be maintained.
- The subdivision of an information system into subsystems has the following *advantages*:
  - It produces *smaller units* of development.
  - It helps to *maximize reuse* at the component level.
  - It helps the developers to *cope with complexity*.
  - It improves *maintainability*.
  - It aids *portability*.

Subsystems

- A *subsystem* typically groups together elements of the system that share some *common properties*.
- An object-oriented subsystem encapsulates a coherent set of responsibilities in order to ensure that it has integrity and can be maintained.
- The subdivision of an information system into subsystems has the following *advantages*:
  - It produces *smaller units* of development.
  - It helps to *maximize reuse* at the component level.
  - It helps the developers to *cope with complexity*.
  - It improves *maintainability*.
  - It aids *portability*.

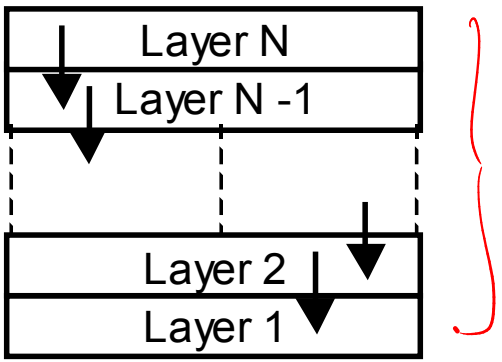
Subsystems

- A *subsystem* typically groups together elements of the system that share some *common properties*.
- An object-oriented subsystem encapsulates a coherent set of responsibilities in order to ensure that it has integrity and can be maintained.
- The subdivision of an information system into subsystems has the following *advantages*:
  - It produces *smaller units* of development.
  - It helps to *maximize reuse* at the component level.
  - It helps the developers to *cope with complexity*.
  - It improves *maintainability*.
  - It aids *portability*.

Layering and Partitioning

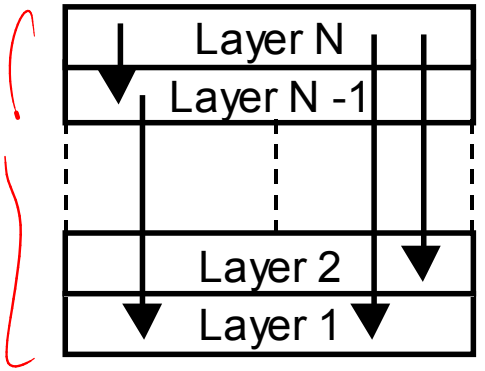
- Two general approaches to divide a software system into subsystems:
  - *Layering*: so called because the different subsystems usually represent different *levels of abstraction*.
  - *Partitioning*: usually means that each subsystem focuses on a different aspect of the *functionality* of the system as a whole.
- Both approaches are often used together on one system.

Layered Architecture
Schematic of a Layered Architecture



Closed architecture—messages may only be sent to the adjacent lower layer.

layers



Open architecture—messages can be sent to any lower layer.

Layered Architecture
Closed Vs Open

- A **closed** architecture:
  - *minimizes dependencies* between the layers, and
  - reduces the impact of a change to the interface of any one layer.
- An **open** layered architecture:
  - produces more compact code, the services of all lower level layers can be *accessed directly* by any layer above them without the need for extra program code to pass messages through each intervening layer,
  - but this breaks the encapsulation of the layers.

Layered Architecture
Closed Vs Open

- A **closed** architecture:
  - *minimizes dependencies* between the layers, and
  - reduces the impact of a change to the interface of any one layer.
- An **open** layered architecture:
  - produces more compact code, the services of all lower level layers can be *accessed directly* by any layer above them without the need for extra program code to pass messages through each intervening layer,
  - but this breaks the encapsulation of the layers.

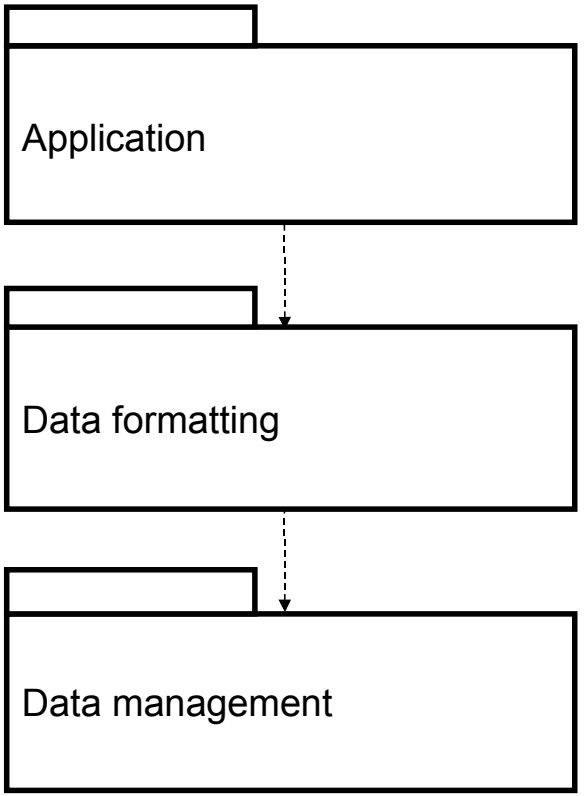
Layered Architecture
Example: Open Systems Interconnection model

OSI
7 Layer
Model

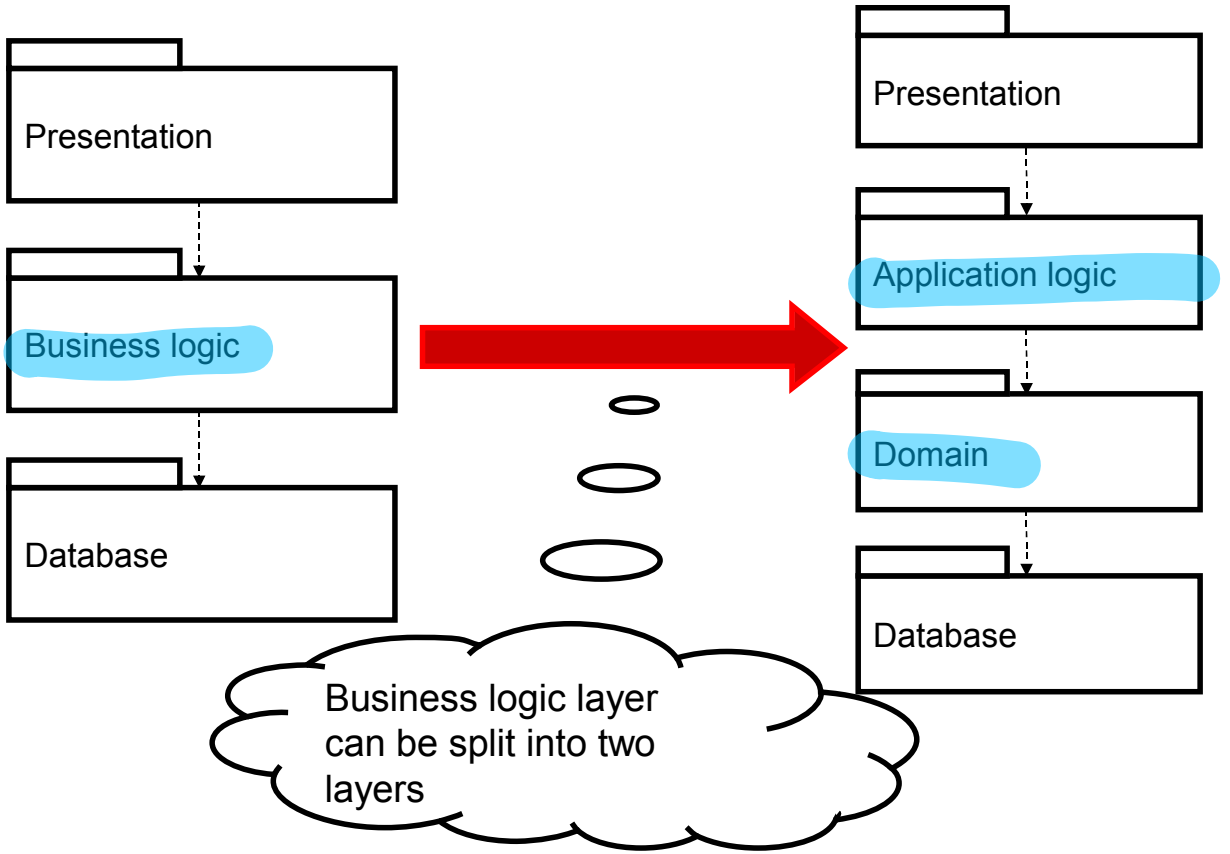
(Adapted from Buschmann et al., 1996)

Layer 7: Application Provides miscellaneous protocols for common activities.
Layer 6: Presentation Structures information and attaches semantics.
Layer 5: Session Provides dialogue control and synchronization facilities.
Layer 4: Transport Breaks messages into packets and ensures delivery.
Layer 3: Network Selects a route from sender to receiver.
Layer 2: Data Link Detects and corrects errors in bit sequences.
Layer 1: Physical Transmits bits: sets transmission rate (baud), bit-code, connection, etc.

Layered Architecture
Simple Layered Architecture



Layered Architecture
3 or 4 Layers?



Layered Architecture
3 or 4 Layers?

- Depending on the size and complexity of the intended system, it may be more appropriate to split the *business logic* layer into two further layers: *application logic* and *domain*.
- *Boundary* classes can be mapped onto the *presentation* layer.
- *Control* classes can be mapped onto the *application logic* layer.
- *Entity* classes can be mapped onto a *domain* layer.
- *Database* layer corresponds to the storage of data within the system.

Tiered Architecture
Layered architectures Vs Tiered architectures

"A layer is a logical structuring mechanism for the elements that make up your software solution; a tier is a physical structuring mechanism for the system infrastructure."

3-tiered architecture

MSDN Library (2011) (http://msdn.microsoft.com/en-us/library/ms998478.aspx)

- A *layered architecture*:
  - concerns the *logical* grouping of functionality / code.
  - The grouping is typically based on the level of abstraction.
  - All layers may be residing on the same computer.
- A *tiered architecture*:

Tiered Architecture
Layered architectures Vs Tiered architectures

"A layer is a logical structuring mechanism for the elements that make up your software solution; a tier is a physical structuring mechanism for the system infrastructure."

MSDN Library (2011) (http://msdn.microsoft.com/en-us/library/ms998478.aspx)

- A *layered architecture*:
  - concerns the *logical* grouping of functionality / code.
  - The grouping is typically based on the level of abstraction.
  - All layers may be residing on the *same* computer.
- A *tiered architecture*:
  - concerns the logical grouping of functionality / code.
  - Each tier resides on and executed by ... computer.

Tiered Architecture
Layered architectures Vs Tiered architectures

"A layer is a logical structuring mechanism for the elements that make up your software solution; a tier is a physical structuring mechanism for the system infrastructure."

MSDN Library (2011) (http://msdn.microsoft.com/en-us/library/ms998478.aspx)

- A *layered architecture*:
  - concerns the *logical* grouping of functionality / code.
  - The grouping is typically based on the level of abstraction.
  - All layers may be residing on the *same* computer.
- A *tiered architecture*:
  - concerns the logical grouping of functionality / code.
  - Each tier resides on and executed by a physical computer.

Tiered Architecture
Layered architectures Vs Tiered architectures

"A layer is a logical structuring mechanism for the elements that make up your software solution; a tier is a physical structuring mechanism for the system infrastructure."

MSDN Library (2011) (http://msdn.microsoft.com/en-us/library/ms998478.aspx)

- A *layered architecture*:
  - concerns the *logical* grouping of functionality / code.
  - The grouping is typically based on the level of abstraction.
  - All layers may be residing on the *same* computer.
- A *tiered architecture*:
  - concerns the logical grouping of functionality / code.
  - Each tier resides on, and executed by, a (remote) computer.

Tiered Architecture
Layered architectures Vs Tiered architectures

"A layer is a logical structuring mechanism for the elements that make up your software solution; a tier is a physical structuring mechanism for the system infrastructure."

MSDN Library (2011) (http://msdn.microsoft.com/en-us/library/ms998478.aspx)

- A *layered architecture*:
  - concerns the *logical* grouping of functionality / code.
  - The grouping is typically based on the level of abstraction.
  - All layers may be residing on the *same* computer.
- A *tiered architecture*:
  - concerns the logical grouping of functionality / code.
  - Each tier resides on, and executed by, *different* computer.

Tiered Architecture
Layered architectures Vs Tiered architectures

"A layer is a logical structuring mechanism for the elements that make up your software solution; a tier is a physical structuring mechanism for the system infrastructure."

MSDN Library (2011) (http://msdn.microsoft.com/en-us/library/ms998478.aspx)

- A *layered architecture*:
  - concerns the *logical* grouping of functionality / code.
  - The grouping is typically based on the level of abstraction.
  - All layers may be residing on the *same* computer.
- A *tiered architecture*:
  - concerns the logical grouping of functionality / code.
  - Each tier resides on, and executed by, *different* computer.

Tiered Architecture
Layered architectures Vs Tiered architectures

"A layer is a logical structuring mechanism for the elements that make up your software solution; a tier is a physical structuring mechanism for the system infrastructure."

MSDN Library (2011) (http://msdn.microsoft.com/en-us/library/ms998478.aspx)

- A *layered architecture*:
  - concerns the *logical* grouping of functionality / code.
  - The grouping is typically based on the level of abstraction.
  - All layers may be residing on the *same* computer.
- A *tiered architecture*:
  - concerns the logical grouping of functionality / code.
  - Each tier resides on, and executed by, *different* computer.

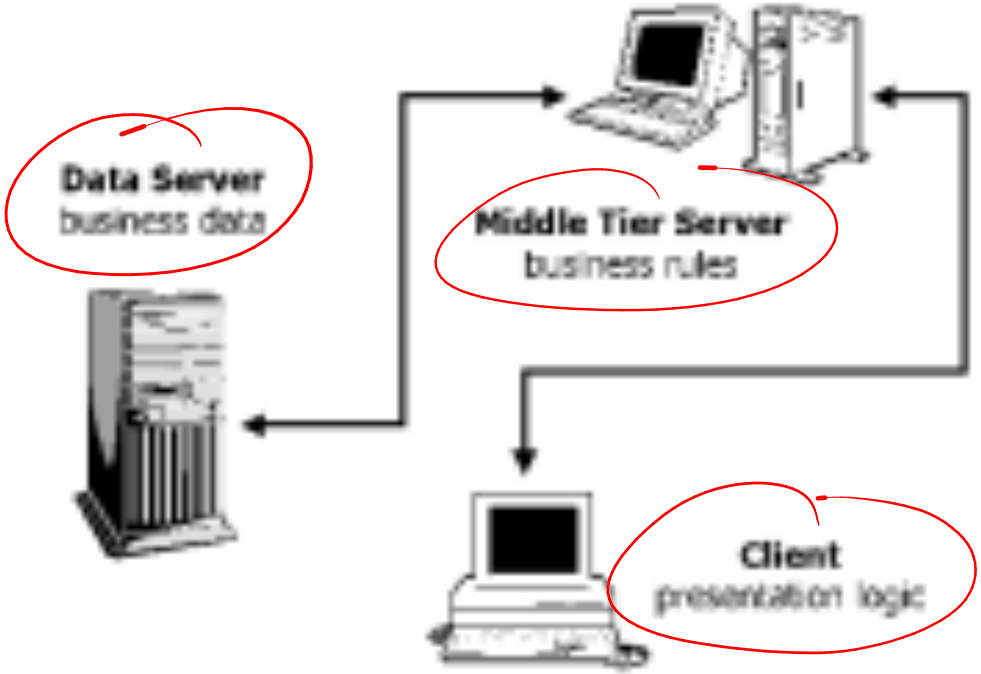
Tiered Architecture
Layered architectures Vs Tiered architectures

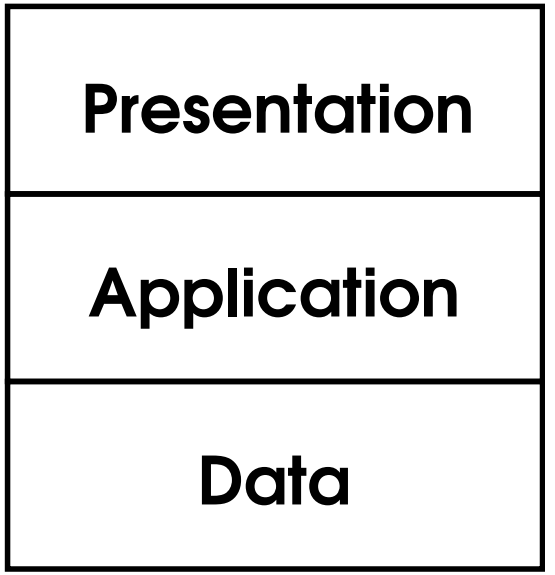
"A layer is a logical structuring mechanism for the elements that make up your software solution; a tier is a physical structuring mechanism for the system infrastructure."

MSDN Library (2011) (http://msdn.microsoft.com/en-us/library/ms998478.aspx)

- A *layered architecture*:
  - concerns the *logical* grouping of functionality / code.
  - The grouping is typically based on the level of abstraction.
  - All layers may be residing on the *same* computer.
- A *tiered architecture*:
  - concerns the logical grouping of functionality / code.
  - Each tier resides on, and executed by, *different* computer.

Tiered Architecture
Three-tier Architecture (Ramirez, 2000)





Source:

`http://www.linuxjournal.com/files/linuxjournal.com/linuxjournal/articles/035/3508/3508f3.jpg`

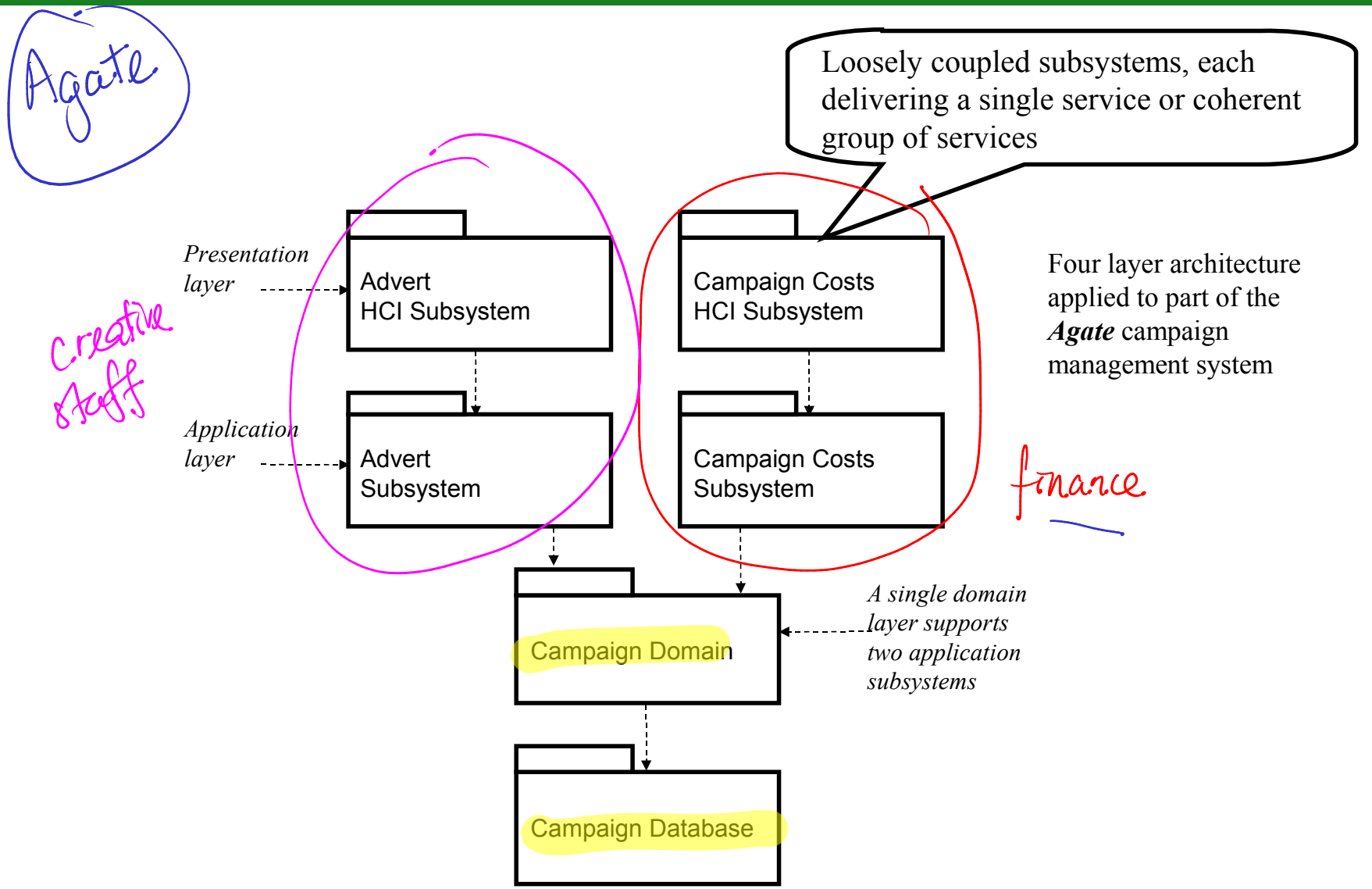
Tiered Architecture
Three-tier Architecture

- *Presentation Tier*:
  The **`client`** contains the presentation logic, including simple control and user input validation.

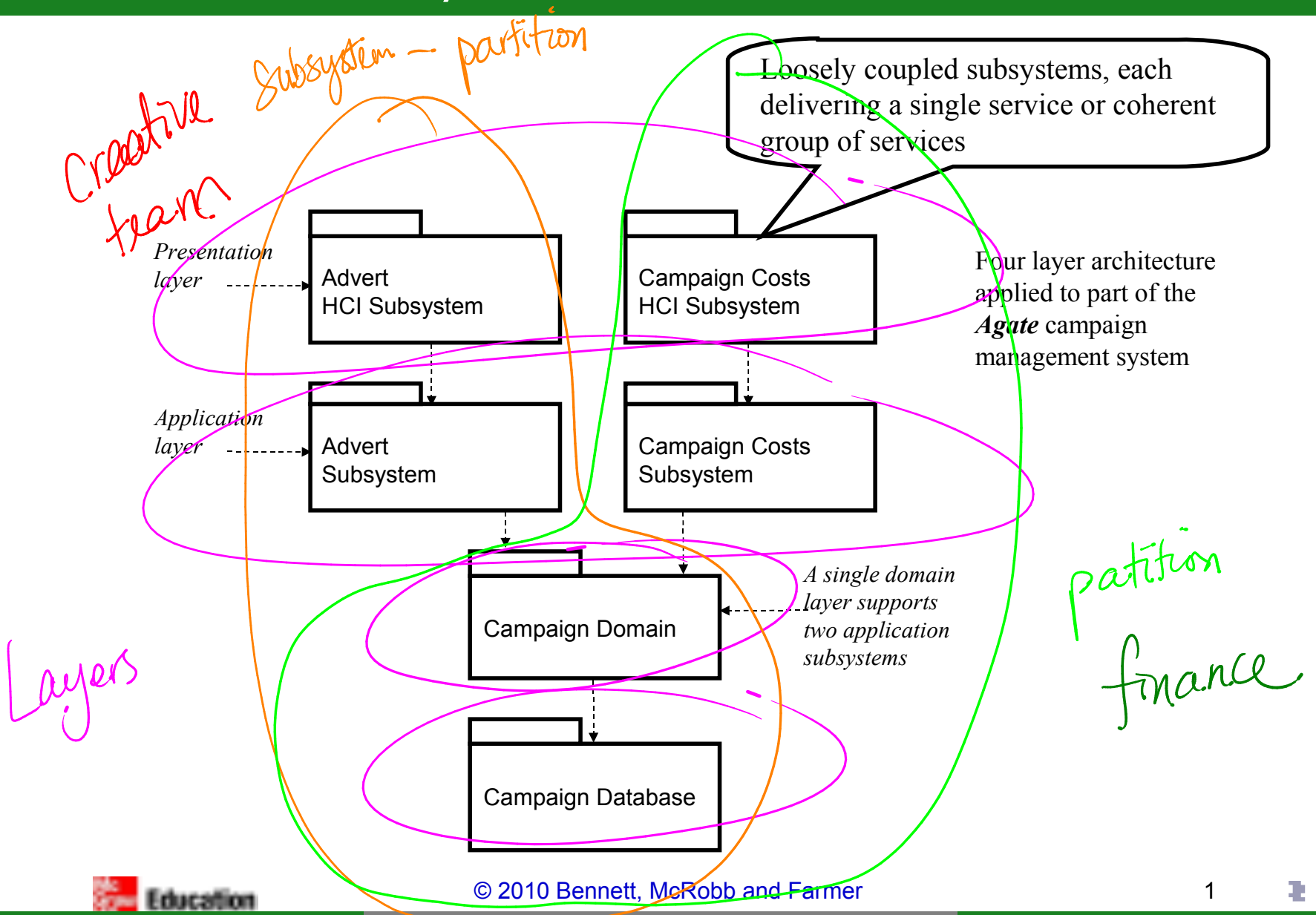
- *Application Tier*:
  The middle tier is also known as the **`application server`**, which provides the business processes logic and the data access.

- *Data Tier*:
  The **`database server`** provides the business data.

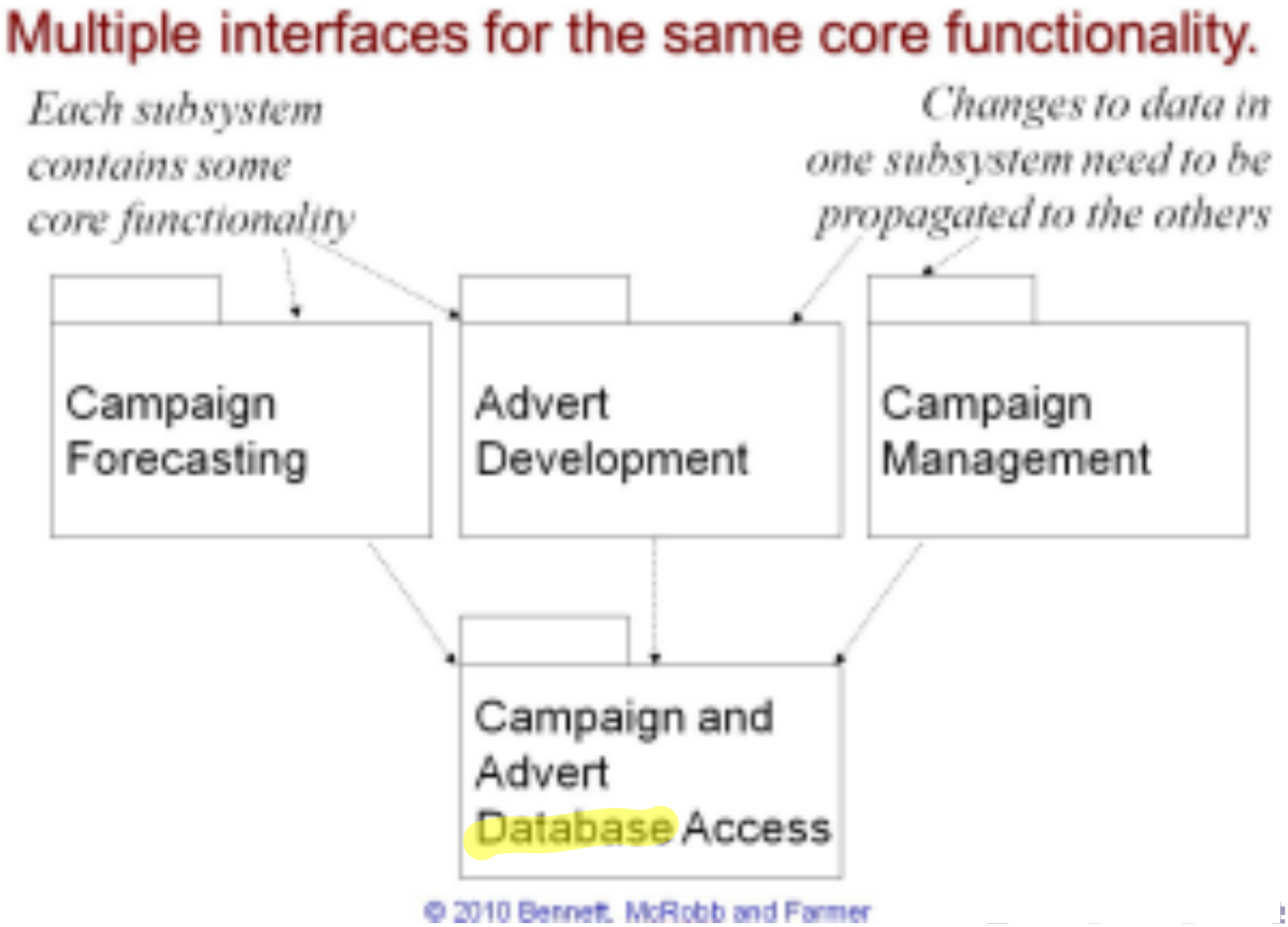
Partitioning
Partitioned Subsystems



Partitioning
Partitioned Subsystems



Model-View-Controller (MVC)
An issue with some architectures



Model-View-Controller (MVC)
An issue with some architectures : Difficulties

Some of the difficulties that need to be resolved for this type of application are:

- The *same information* should be capable of being presented in *different formats* in different windows.
- Changes made within one view should be *reflected immediately* in the other views.

  ➠ ILLUSTRATION: A simple multiplier...

- Changes in the user interface should be *easy to make*.
- *Core functionality* should be *independent* of the interface to enable multiple interface styles to co-exist.

  ➠ ILLUSTRATION: A simple Java application...

Model-View-Controller (MVC)
An issue with some architectures : Difficulties

Some of the difficulties that need to be resolved for this type of application are:

- The *same information* should be capable of being presented in *different formats* in different windows.
- Changes made within one view should be *reflected immediately* in the other views.
  - ➠ ILLUSTRATION: A simple multiplier...
- Changes in the user interface should be *easy to make*.
- *Core functionality* should be *independent* of the interface to enable multiple interface styles to co-exist.
  - ➠ ILLUSTRATION: A simple Java application...

Model-View-Controller (MVC)
An issue with some architectures : Difficulties

Some of the difficulties that need to be resolved for this type of application are:

- The *same information* should be capable of being presented in *different formats* in different windows.
- Changes made within one view should be *reflected immediately* in the other views.
  - ➠ ILLUSTRATION: A simple multiplier...
- Changes in the user interface should be *easy to make*.
- *Core functionality* should be *independent* of the interface to enable multiple interface styles to co-exist.
  - ➠ ILLUSTRATION: A simple Java application...

Model-View-Controller (MVC)
An issue with some architectures : Difficulties

Some of the difficulties that need to be resolved for this type of application are:

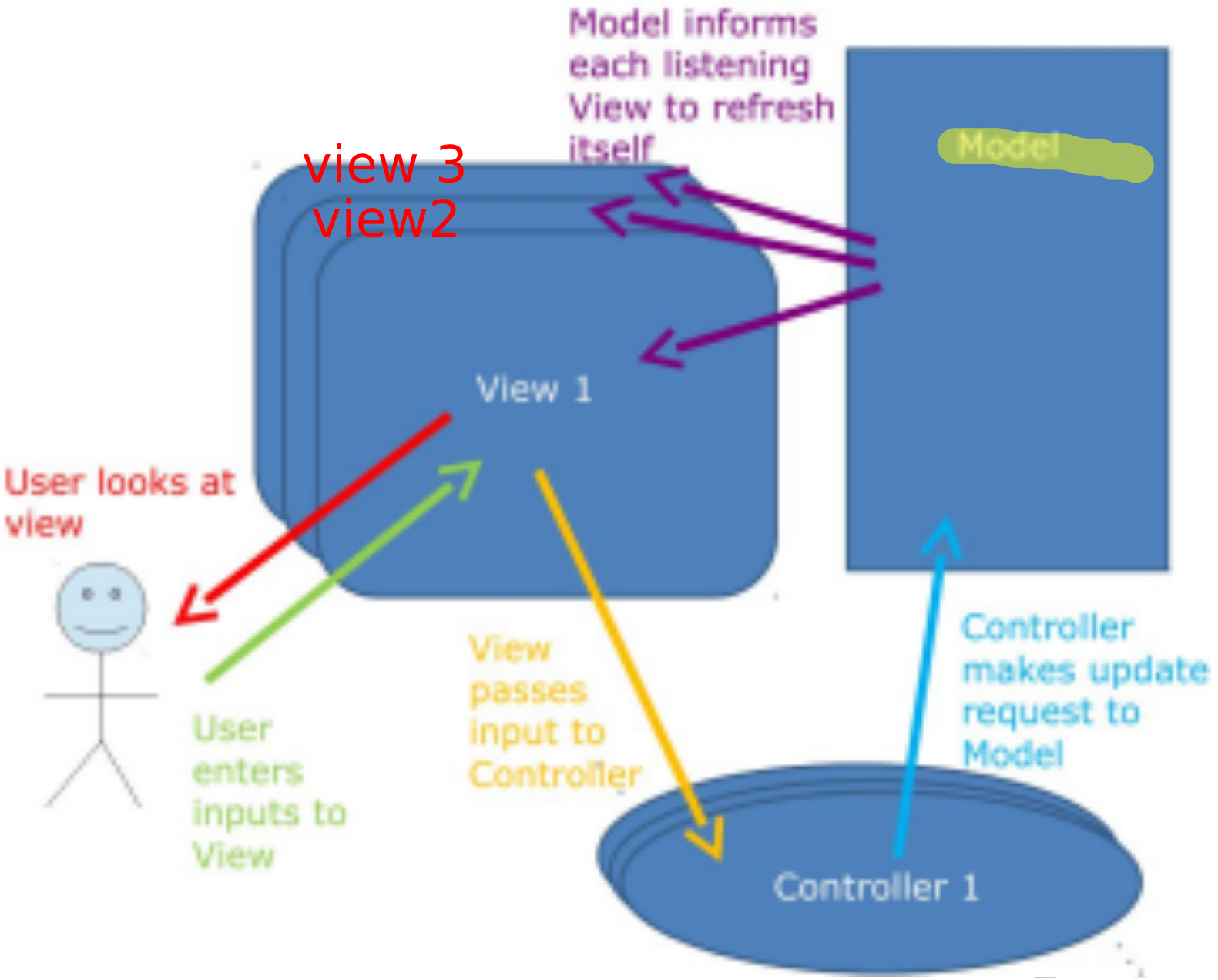
- The *same information* should be capable of being presented in *different formats* in different windows.
- Changes made within one view should be *reflected immediately* in the other views.
  - ➠ ILLUSTRATION: A simple multiplier...
- Changes in the user interface should be *easy to make*.
- *Core functionality* should be *independent* of the interface to enable multiple interface styles to co-exist.
  - ➠ ILLUSTRATION: A simple Java application...

Model-View-Controller (MVC)
A solution: the MVC architecture

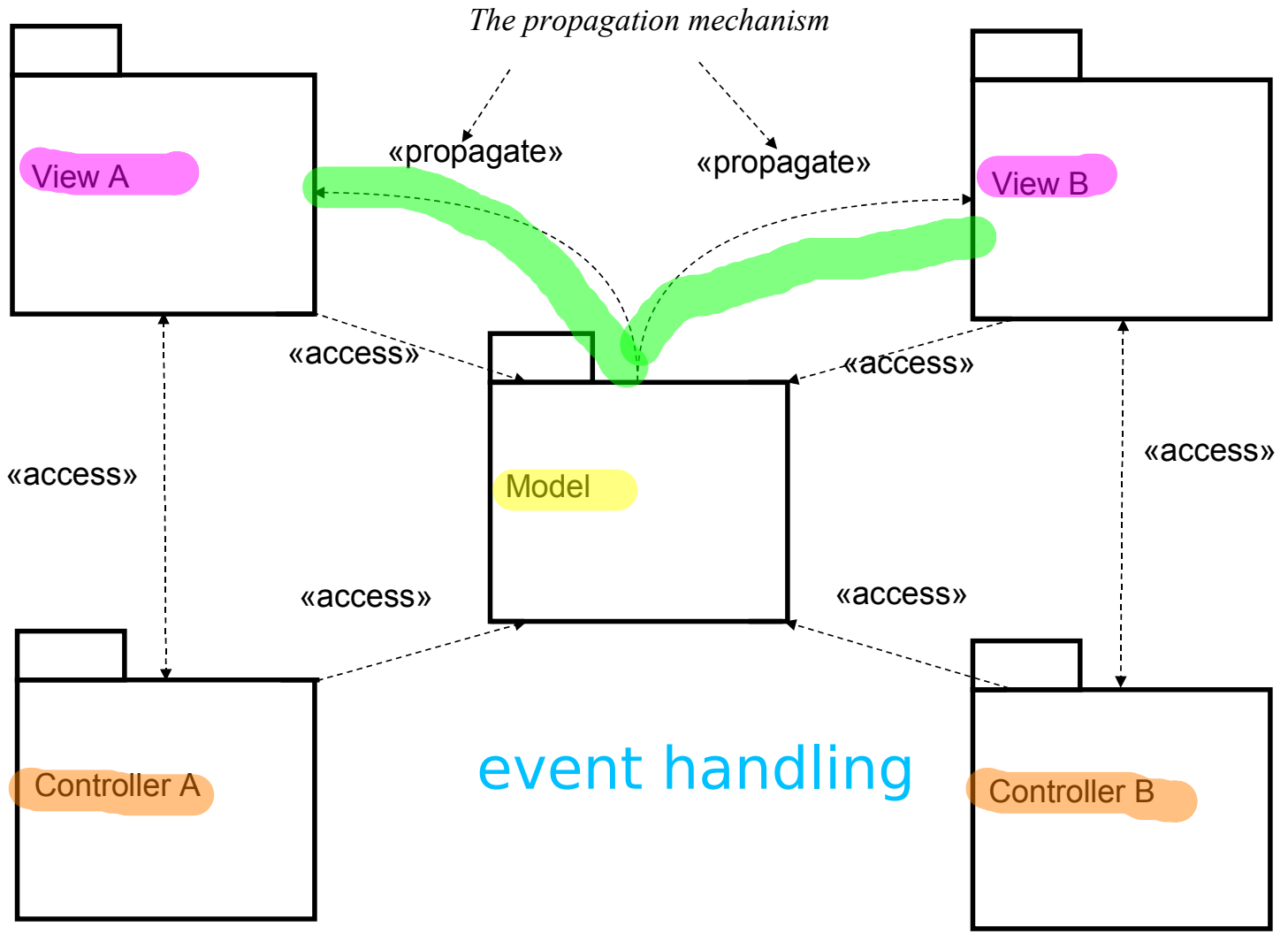
Model View Controller

comes to life...

Model-View-Controller (MVC)



Model-View-Controller (MVC)
Bennett et al. (2010)



(Adapted from Hopkins and Horan, 1995)

Model-View-Controller (MVC)
Bennett et al. (2010)

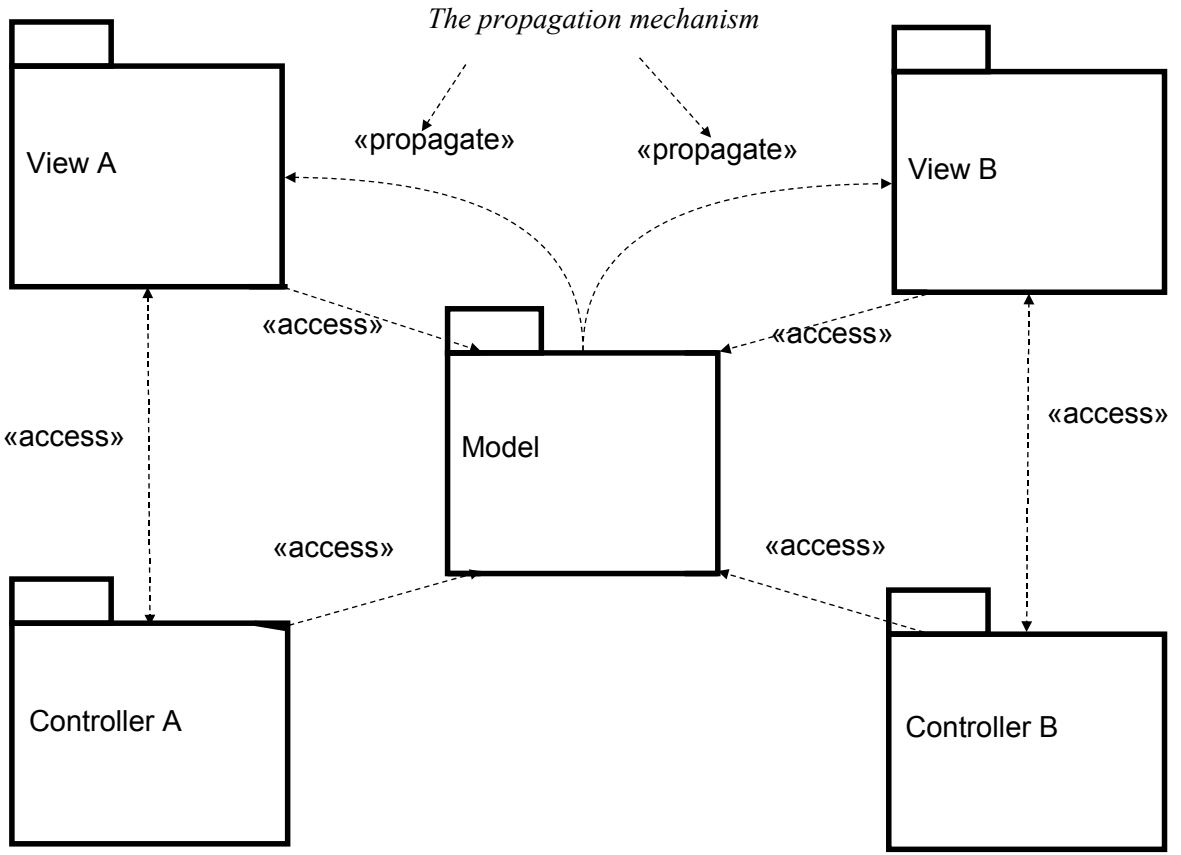
- *Model*: provides the *central functionality* of the application and is aware of each of its dependent view and controller components.

- *View*: corresponds to a *particular* style and format of *presentation of information* to the user.
  The view retrieves data from the **`model`** and updates its presentations when data has been changed in one of the other **`views`**.
  The view *creates* its associated **`controller`**.

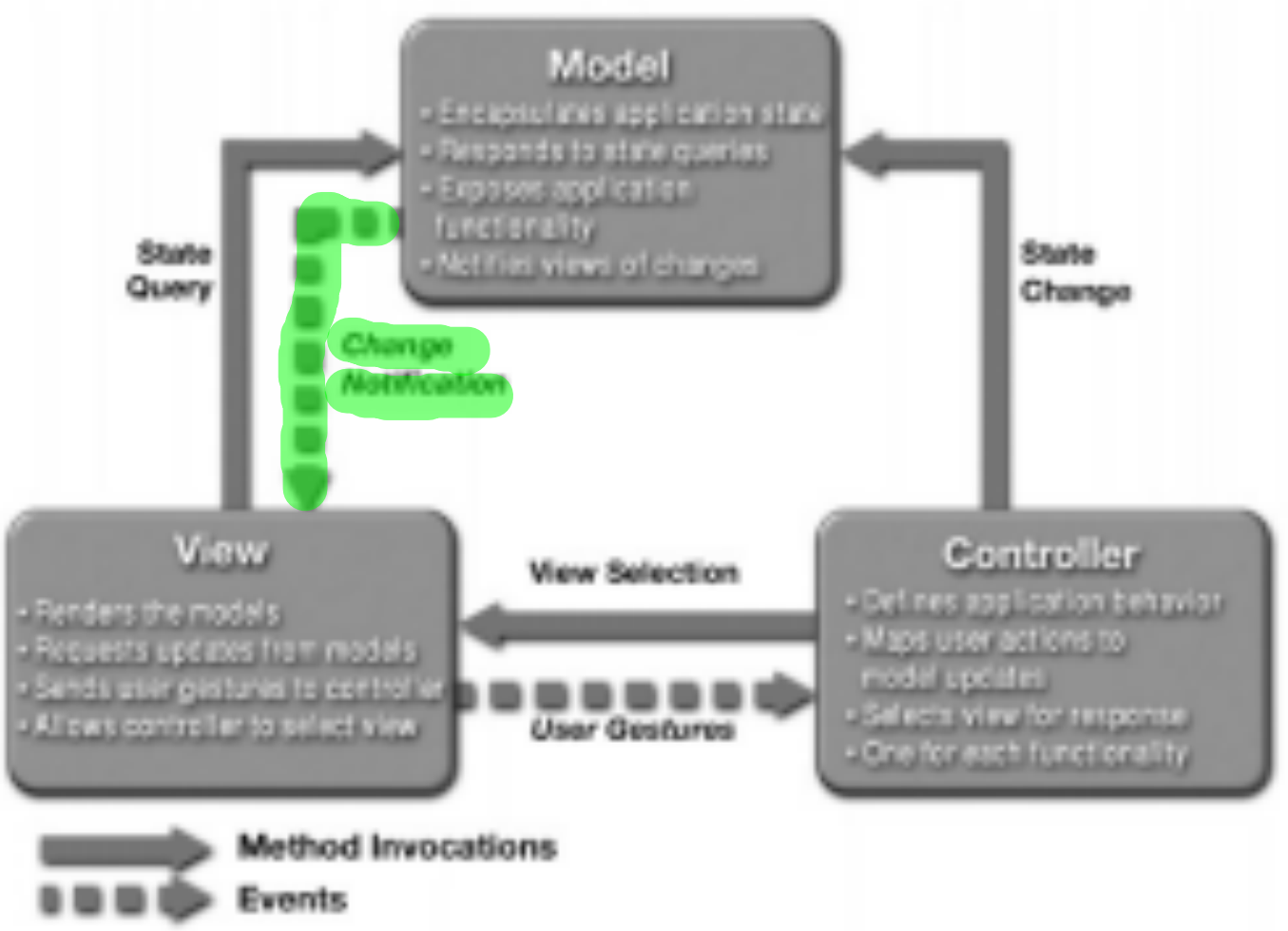
- *Controller*: accepts *user input* in the form of *events* that trigger the execution of operations within the **`model`**. These may cause changes to the information and in turn trigger updates in all the **`views`** ensuring that they are all up to date.

Model-View-Controller (MVC)
Bennett et al. (2010)

- *Propagation Mechanism*: enables the model to inform each view that the model data has changed and as a result the view must update itself.
  It is also often called the *dependency mechanism*.



Model-View-Controller (MVC)
Eckstein (2007)



Source: Figure 1, Eckstein (2007) http://www.oracle.com/technetwork/articles/javase/mvc-136693.html

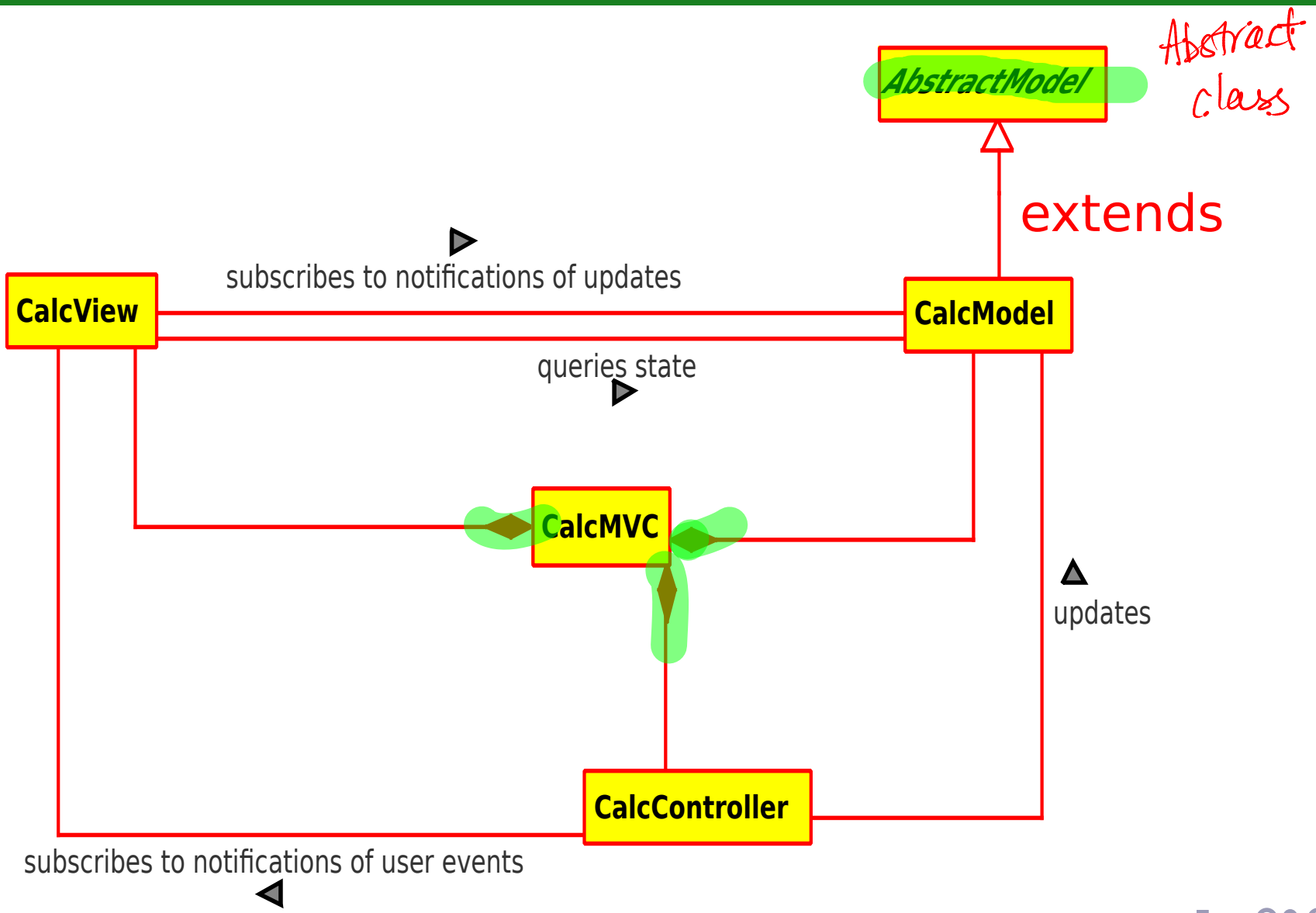
Model-View-Controller (MVC)
Eckstein (2007)

- *Model*: The model represents *data* and the rules that govern access to and updates of this data.

- *View*: The view *renders* the contents of a model.
  If the model data changes, the view must update its presentation as needed. This can be done by having the view *registering* itself with the model for change notifications or by making the view responsible for *calling* the model when it needs to retrieve the most current data.

- *Controller*: The controller *translates* the user's interactions with the view into actions that the model will perform.
  In a stand-alone GUI client, user interactions could be button clicks or menu selections, whereas in an enterprise web application, they appear as `GET` and `POST` HTTP requests.

A simple calculator in MVC
UML Class Diagram



A simple calculator in MVC
Roles of each component

- CalcMVC:
  - is the *top-level* class.
  - responsible for *creating* the **`model`**, **`view`** and **`controller`** objects.
- CalcModel:
  - does *not know* about the **`view`** nor the **`controller`**.
  - *fires* property change notifications to its subscribers, in this case, the **`view`**.

A simple calculator in MVC
Roles of each component

- CalcView:
  - *renders* the **`model`** using a GUI.
  - *subscribes* to the property change notifications from the **`model`**.
  - *passes* user input events to its subscribers.
- CalcController:
  - *subscribes* to all user input events.
  - *defines* the event handlers for GUI events.
  - *calls* the **`model`** to change its state based on user event.

CalcMVC (Swartz, 2004)


```
1  // structure/calc-mvc/CalcMVC.java - Calculator in MVC pattern.
2  // Fred Swartz - December 2004
3
4  import javax.swing.*;
5
6  public class CalcMVC {
7      //...  Create model, view, and controller.
8      public static void main(String[] args) {
9
10         CalcModel      model      = new CalcModel();
11         CalcView       view       = new CalcView(model);
12         CalcController controller = new CalcController(model, view);
13
14         view.setVisible(true);
15     }
16 }
```

Source: http://leepoint.net/notes-java/GUI/structure/40mvc.html

CalcModel (1 of 3)

```
1  // structure/calc-mvc/CalcModel.java
2  // Fred Swartz - December 2004
3  // Modified by M Konecny 21-01-2011
4  // Model
5  // This model is completely independent of the user interface.
6  // It could as easily be used by a command line or web interface.
7
8  import java.math.BigInteger;
9
10 public class CalcModel extends AbstractModel {
11     /** name used by property change notification mechanism */
12     public static final String TOTAL_PROPERTY = "Total";
13     private static final String INITIAL_VALUE = "1";
14
15     private BigInteger m_total; // The total current value state.
16
17     /** Constructor */
18     CalcModel() {
19         reset();
20     }
```

CalcModel (2 of 3)

```
1     /** Reset total to initial value. */
2     public void reset() {
3         BigInteger old_value = m_total;
4         m_total = new BigInteger(INITIAL_VALUE);
5         firePropertyChange(TOTAL_PROPERTY, old_value, m_total);
6     }
7
8     /**
9      * Multiply current total by a number.
10     * @param operand
11     *            Number (as string) to multiply total by.
12     */
13    public void multiplyBy(BigInteger operand) {
14        BigInteger old_value = m_total;
15        m_total = m_total.multiply(operand);
16        firePropertyChange(TOTAL_PROPERTY, old_value, m_total);
17    }
```

mutator method

mutator method

CalcModel (3 of 3)

```
1       /**
2        * Set the total value.
3        * @param value
4        *          New value that should be used for the calculator total.
5        */
6       public void setValue(BigInteger value) {
7           BigInteger old_value = m_total;
8           m_total = value;
9           firePropertyChange(TOTAL_PROPERTY, old_value, m_total);
10      }
11
12      /** Return current calculator total. */
13      public BigInteger getValue() {
14          return m_total;
15      }
16  }
```

Accessor method

AbstractModel (1 of 2)

```
1  import java.beans.PropertyChangeListener;
2  import java.beans.PropertyChangeSupport;
3
4  /**
5   * This class provides base level functionality for all models,
6   * including a support for a property change mechanism (using
7   * the PropertyChangeSupport class), as well as a convenience
8   * method that other objects can use to reset model state.
9   *
10  * @author Robert Eckstein
11  * from: http://www.oracle.com/technetwork/articles/javase/mvc-136693.html
12  */
13 public abstract class AbstractModel {
14
15     /**
16      * Convenience class that allow others to observe changes
17      * to the model properties
18      */
19     protected PropertyChangeSupport propertyChangeSupport;
20
21     /** Default constructor: Instantiates class PropertyChangeSupport. */
22     public AbstractModel() {
23         propertyChangeSupport = new PropertyChangeSupport(this);
24     }
```

AbstractModel (2 of 2)

```
1     /** Adds a property change listener to the observer list. */
2     public void addPropertyChangeListener(PropertyChangeListener l) {
3         propertyChangeSupport.addPropertyChangeListener(l);
4     }
5
6     /** Removes a property change listener from the observer list. */
7     public void removePropertyChangeListener(PropertyChangeListener l) {
8         propertyChangeSupport.removePropertyChangeListener(l);
9     }
10
11    /**
12     * Fires an event to all registered listeners informing them
13     * that a property in this model has changed.
14     */
15    protected void firePropertyChange(String propertyName,
16            Object oldValue, Object newValue) {
17        propertyChangeSupport.firePropertyChange(propertyName,
18            oldValue, newValue);
19    }
20 }
```

CalcView (1 of 2)

```
1  // structure/calc-mvc/CalcView.java - View component
2  // Presentation only.  No user actions.
3  // Fred Swartz - December 2004
4  // Modified by M Konecny 21-01-2011
5
6  import java.awt.event.*;
7  import java.beans.PropertyChangeEvent;
8  import java.beans.PropertyChangeListener;
9  // other import statements omitted
10 public class CalcView extends JFrame {
11
12     private CalcModel m_model;
13     // other field definitions and initialisation omitted.
14
15     /** Constructor */
16     CalcView(CalcModel model) {
17         // ...  Set up the logic
18         m_model = model;
19         // ...  Initialize components
20         m_totalTf.setText(m_model.getValue().toString());
21         m_totalTf.setEditable(false);
```

CalcView (2 of 2)

```
1           // ...  Setup automatic updates of the total from model:
2           m_model.addPropertyChangeListener(new PropertyChangeListener() {
3               // will be executed by the model when an event is fired there.
4               public void propertyChange(PropertyChangeEvent evt) {
5                   if (evt.getPropertyName().equals(m_model.TOTAL_PROPERTY)){
6                       m_totalTf.setText(evt.getNewValue().toString());
7                   }
8               }
9           });
10
11          // other GUI code omitted
12      }
13
14      // other GUI methods omitted
15
16      void addMultiplyListener(ActionListener mal) {
17          m_multiplyBtn.addActionListener(mal);
18      }
19
20      void addClearListener(ActionListener cal) {
21          m_clearBtn.addActionListener(cal);
22      }
23  }
```

CalcController (1 of 3)

```
1  // structure/calc-mvc/CalcController.java - Controller
2  // Handles user interaction with listeners.
3  // Calls View and Model as needed.
4  // Fred Swartz - December 2004, modified by M Konecny 21-01-2011
5
6  import java.awt.event.*;
7  public class CalcController {
8      // The Controller needs to interact with both the Model & View.
9      private CalcModel m_model;
10     private CalcView  m_view;
11
12     /** Constructor */
13     CalcController(CalcModel model, CalcView view) {
14         m_model = model;
15         m_view  = view;
16         //...  Add listeners to the view.
17         view.addMultiplyListener(new MultiplyListener());
18         view.addClearListener(new ClearListener());
19     }
```

CalcController (2 of 3)

```
1     ////////////////// inner class MultiplyListener
2     /** When a multiplication is requested.
3      *  1. Get the user input number from the View.
4      *  2. Call the model to multiply by this number.
5      * If there was an error, do nothing.
6      */
7     class MultiplyListener implements ActionListener {
8         public void actionPerformed(ActionEvent event) {
9             try {
10                m_model.multiplyBy(m_view.getUserInput());
11
12            } catch (Exception e) {
13                // ignore the exception, ie ignore the event
14            }
15        }
16    }//end inner class MultiplyListener
```

CalcController (3 of 3)

```
1
2       ////////////////////// inner class ClearListener
3       /**  Reset model. (View will update automatically)
4        */
5       class ClearListener implements ActionListener {
6           public void actionPerformed(ActionEvent e) {
7               m_model.reset();
8           }
9       }// end inner class ClearListener
10  }// end of class CalcController
```

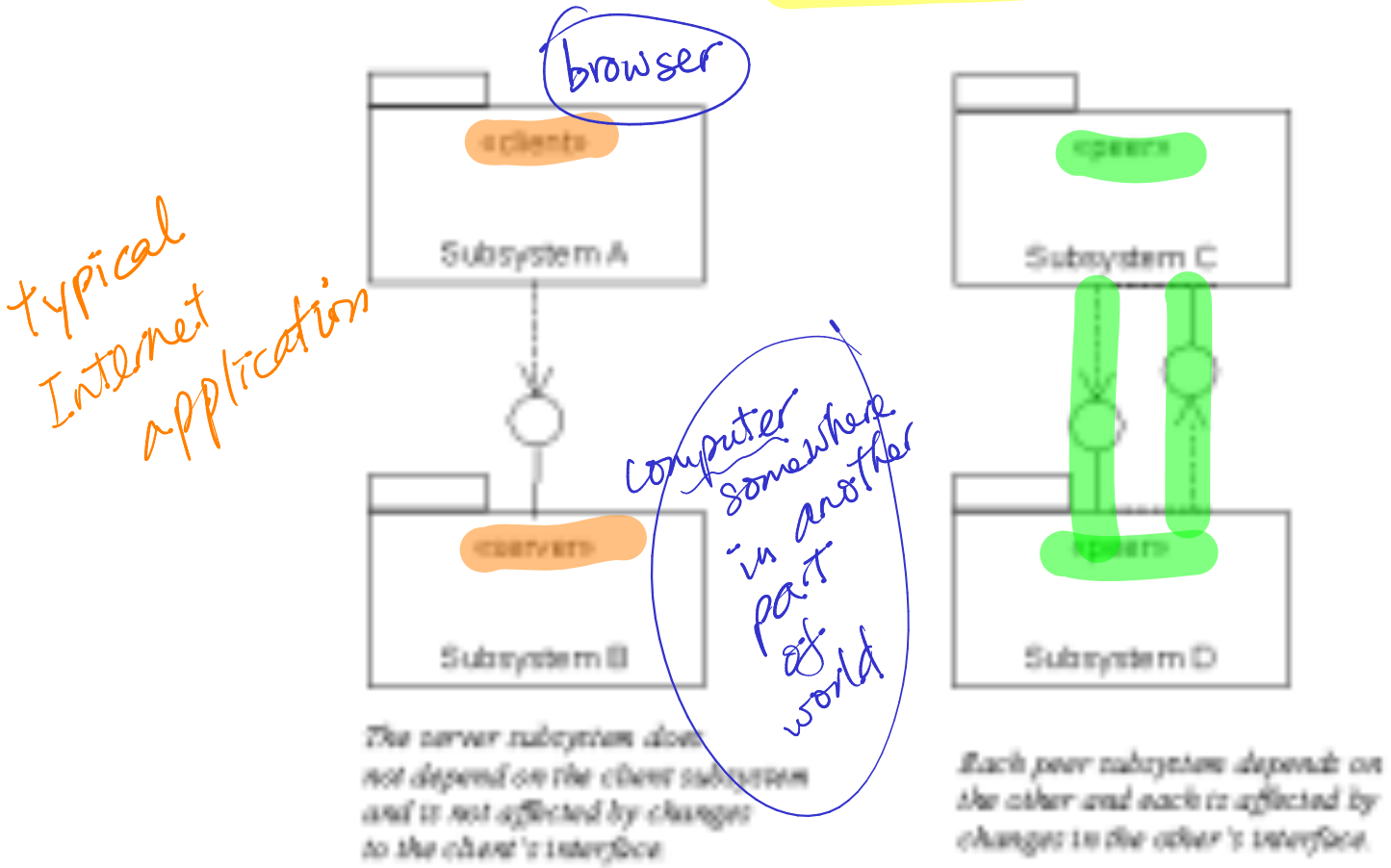
A simple calculator in MVC
Single Model, Multiple Views

- The MVC architecture is *flexible*.
- *Multiple* views can be created in the MVC architecture.
- Changes made to the model within one view is *reflected immediately* to *all* views which use the same model.

```
1  import javax.swing.*;
2  public class CalcMVC {
3      public static void main(String[] args) {
4          CalcModel      model       = new CalcModel();
5          CalcView       view1       = new CalcView(model);
6          CalcController controller1 = new CalcController(model, view1);
7          CalcView       view2       = new CalcView(model);
8          CalcController controller2 = new CalcController(model, view2);
9
10         view1.setVisible(true);
11         view2.setVisible(true);
12     }
13 }
```

Subsystem Communications Styles

- Each **`subsystem`** provides *services* for other subsystems.
- There are two different styles of *communication* that make this possible: *client-server* and *peer-to-peer* communication.



Subsystem Communications
Client-server communication

- *Client-server* communication requires the **`client`** to know the interface of the **`server`** subsystem.
  - ➠ The communication is only in *one direction*.
- The **`client`** subsystem requests *services* from the server subsystem and not vice versa.
  - ➠ The **`client`** plays the role of a *consumer*; while the **`server`** is considered to be the *supplier*.
- Examples of client-server communication can be found in most *network-based applications*.
  - ➠ E.g. email systems and most web applications.

Subsystem Communications
Client-server communication

- *Client-server* communication requires the **`client`** to know the interface of the **`server`** subsystem.
  - ➟ The communication is only in *one direction*.
- The **`client`** subsystem requests *services* from the server subsystem and not vice versa.
  - ➟ The **`client`** plays the role of a *consumer*; while the **`server`** is considered to be the *supplier*.
- Examples of client-server communication can be found in most *network-based applications*.
  - ➟ E.g. email systems and most web applications.

Subsystem Communications
Client-server communication

- *Client-server* communication requires the **`client`** to know the interface of the **`server`** subsystem.
  - ➟ The communication is only in *one direction*.
- The **`client`** subsystem requests *services* from the server subsystem and not vice versa.
  - ➟ The **`client`** plays the role of a *consumer*; while the **`server`** is considered to be the *supplier*.
- Examples of client-server communication can be found in most *network-based applications*.
  - ➟ E.g. email systems and most web applications.

Subsystem Communications
Peer-to-peer communication

- *Peer-to-peer* (P2P) communication requires each subsystem to know the interface of the other, thus *coupling* them more *tightly*.
- Each peer has to run exactly the *same program* and hence all peers provide *identical services*.
- *Peer-to-peer* communication is *two way* since either peer subsystem may request services from the other.

Subsystem Communications
Peer-to-peer communication

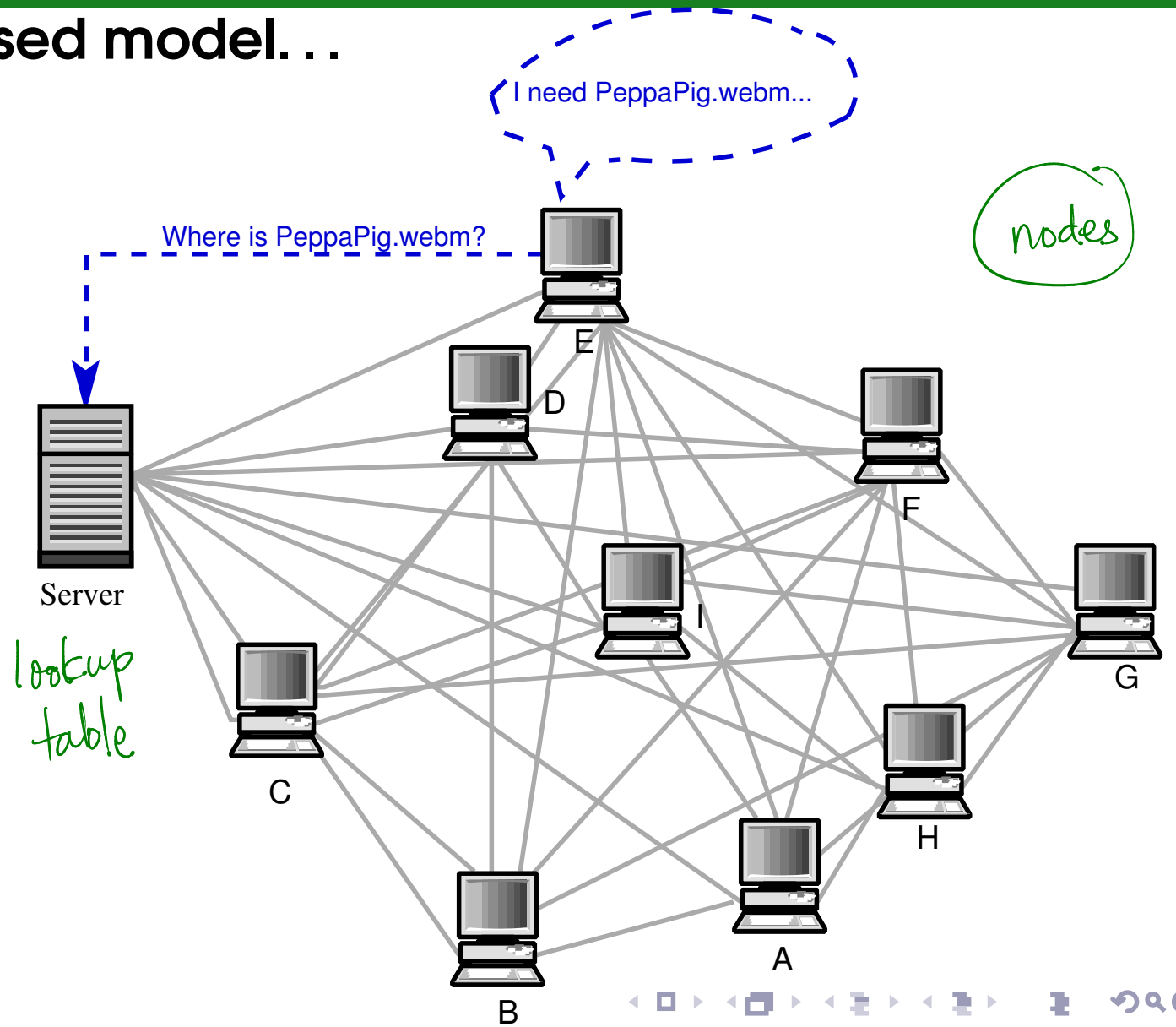
- *Peer-to-peer* (P2P) communication requires each subsystem to know the interface of the other, thus *coupling* them more *tightly*.
- Each peer has to run exactly the *same program* and hence all peers provide *identical services*.
- *Peer-to-peer* communication is *two way* since either peer subsystem may request services from the other.

Subsystem Communications
Peer-to-peer communication

- *Peer-to-peer* (P2P) communication requires each subsystem to know the interface of the other, thus *coupling* them more *tightly*.
- Each peer has to run exactly the *same program* and hence all peers provide *identical services*.
- *Peer-to-peer* communication is *two way* since either peer subsystem may request services from the other.

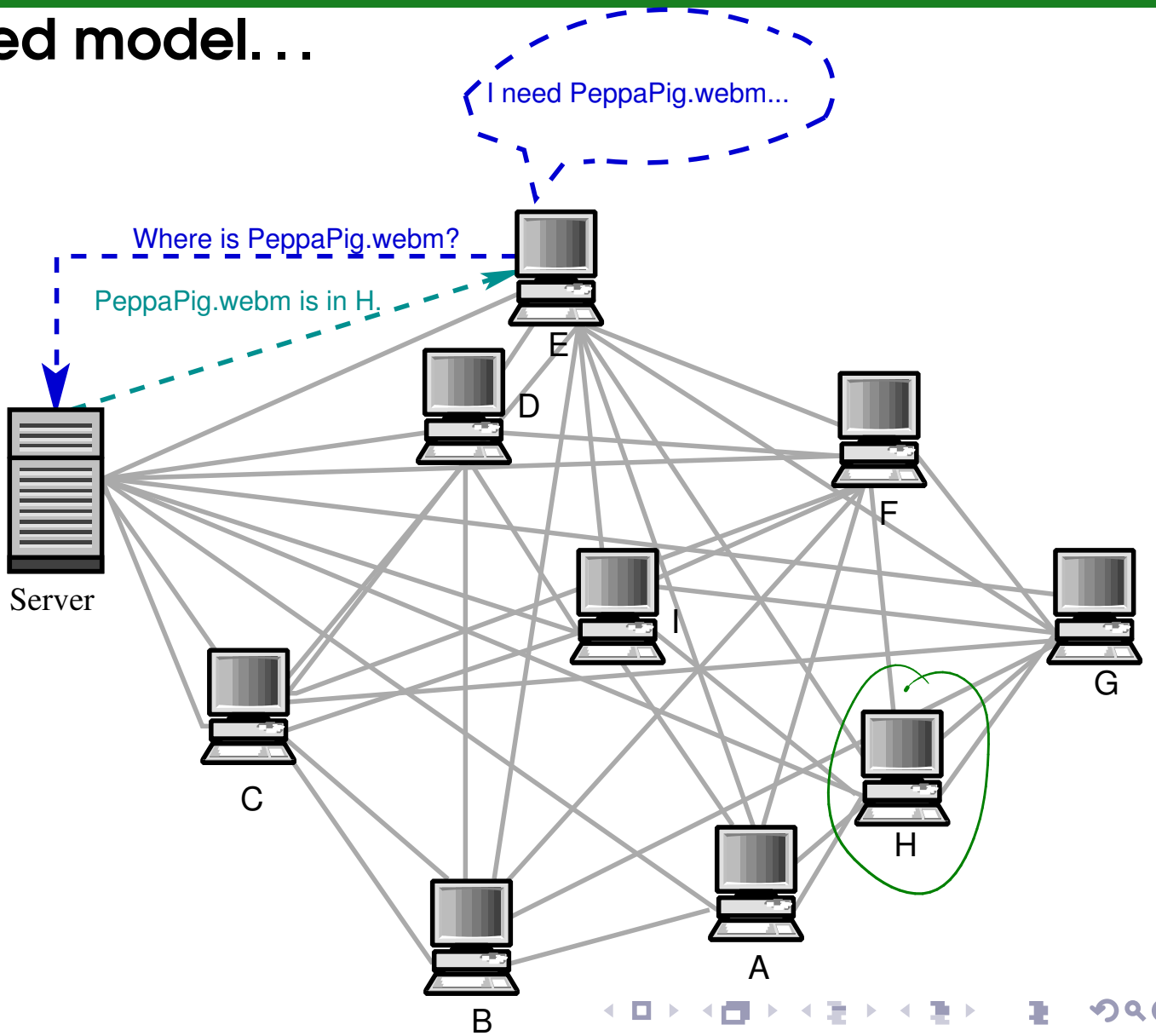
P2P Communication: How to find a peer which holds the required information?

Using a centralised model...



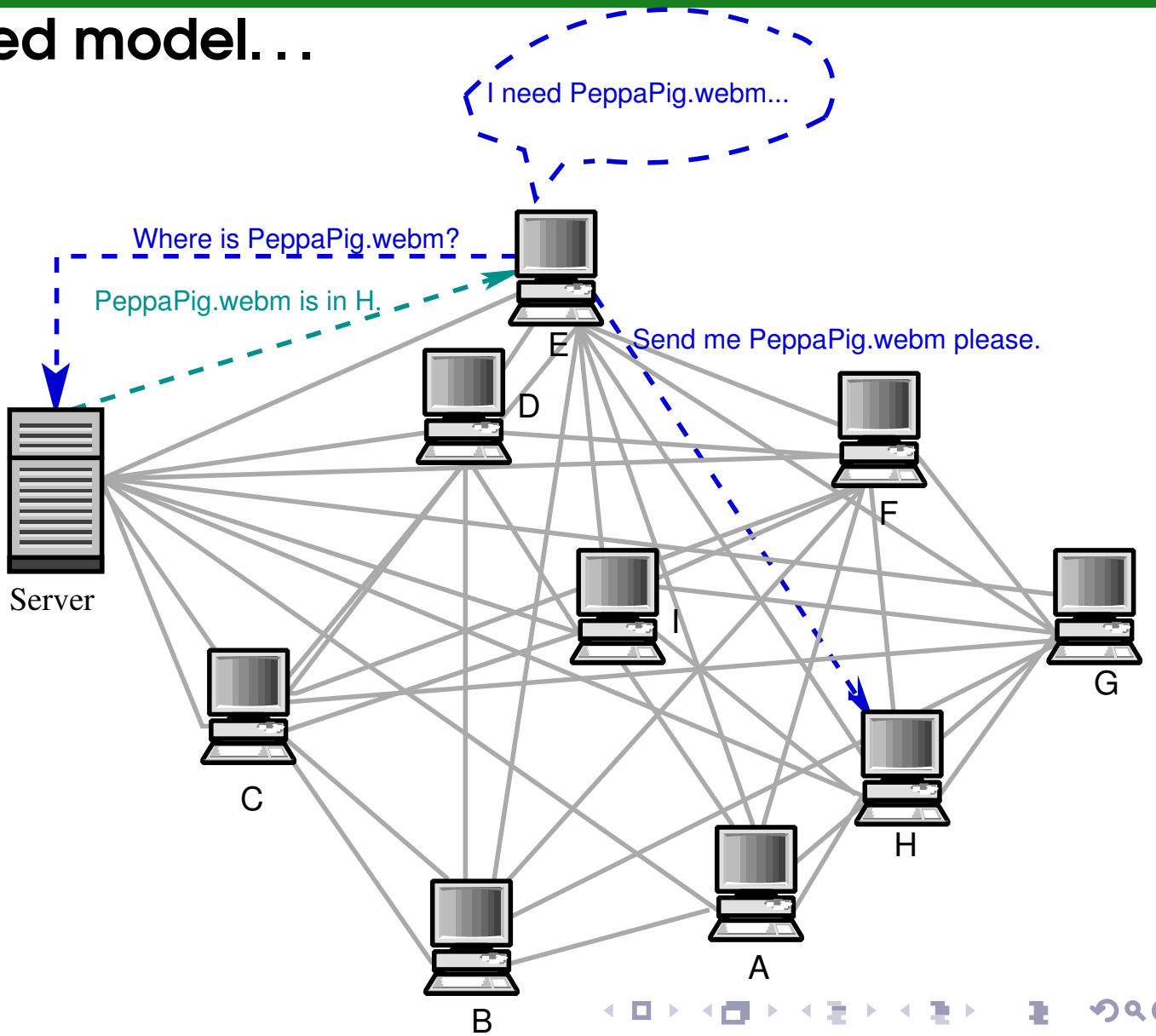
P2P Communication: How to find a peer which holds the required information?

Using a centralised model...



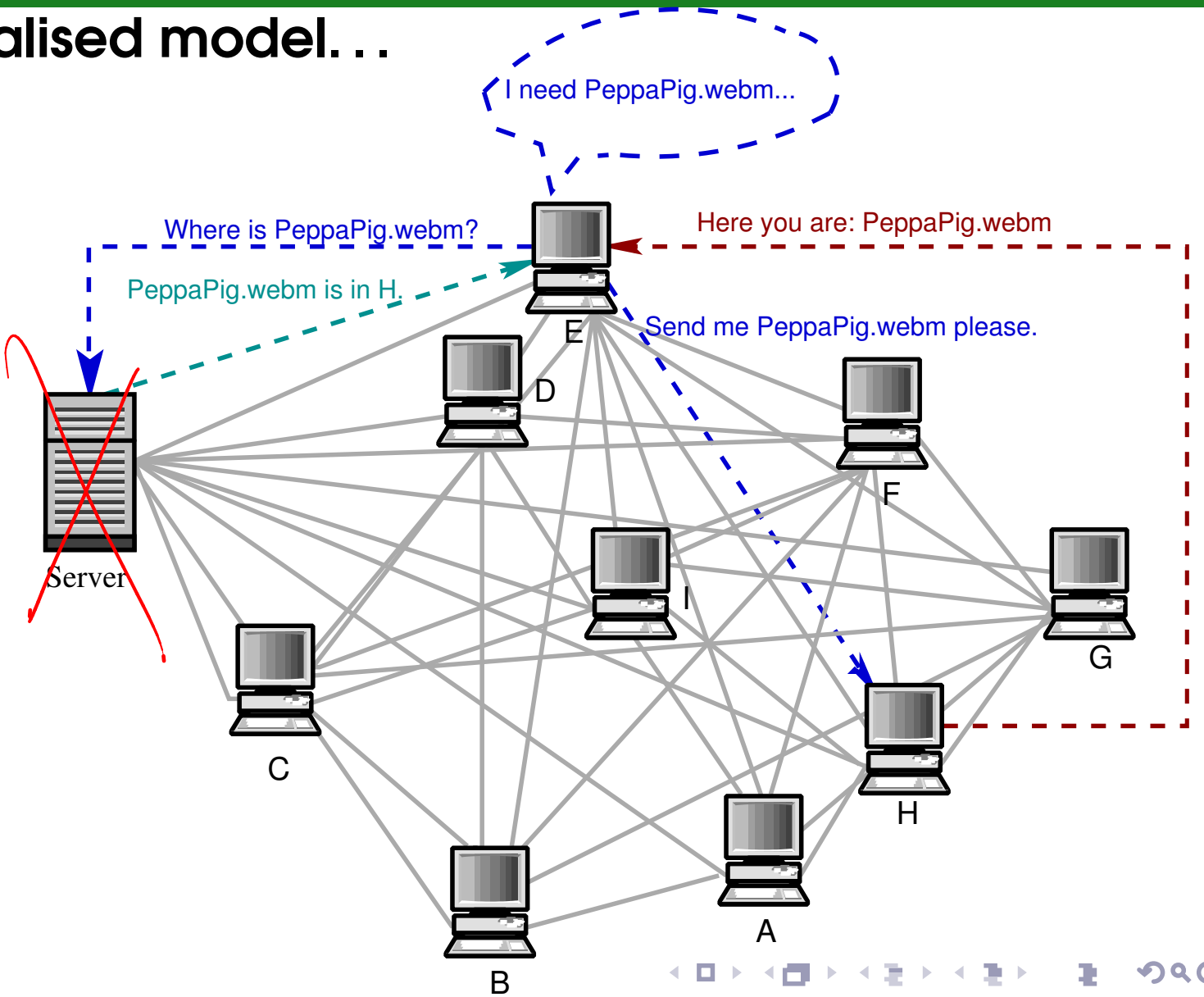
P2P Communication: How to find a peer which holds the required information?

Using a centralised model...



P2P Communication: How to find a peer which holds the required information?

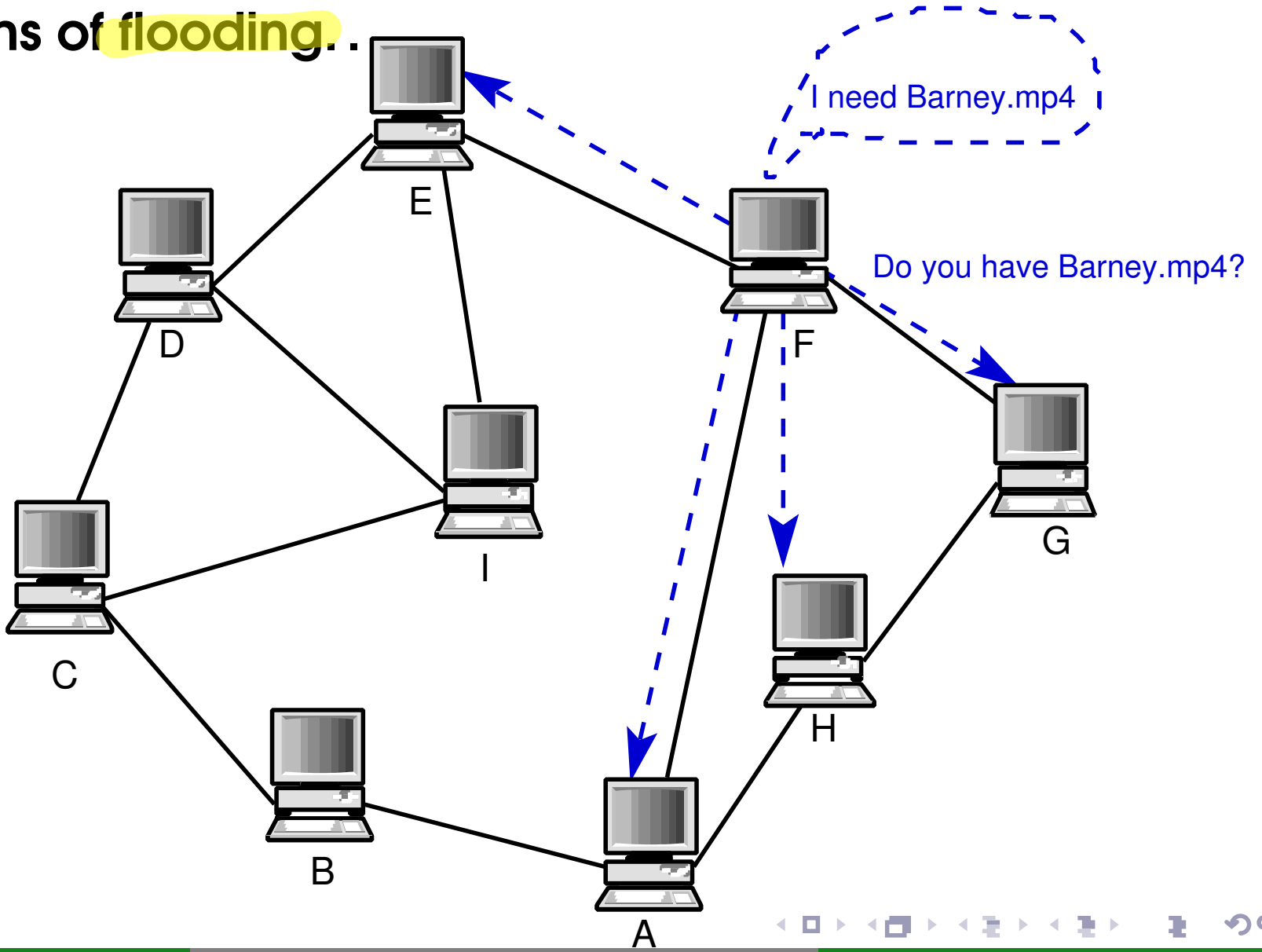
Using a centralised model...



Napster!

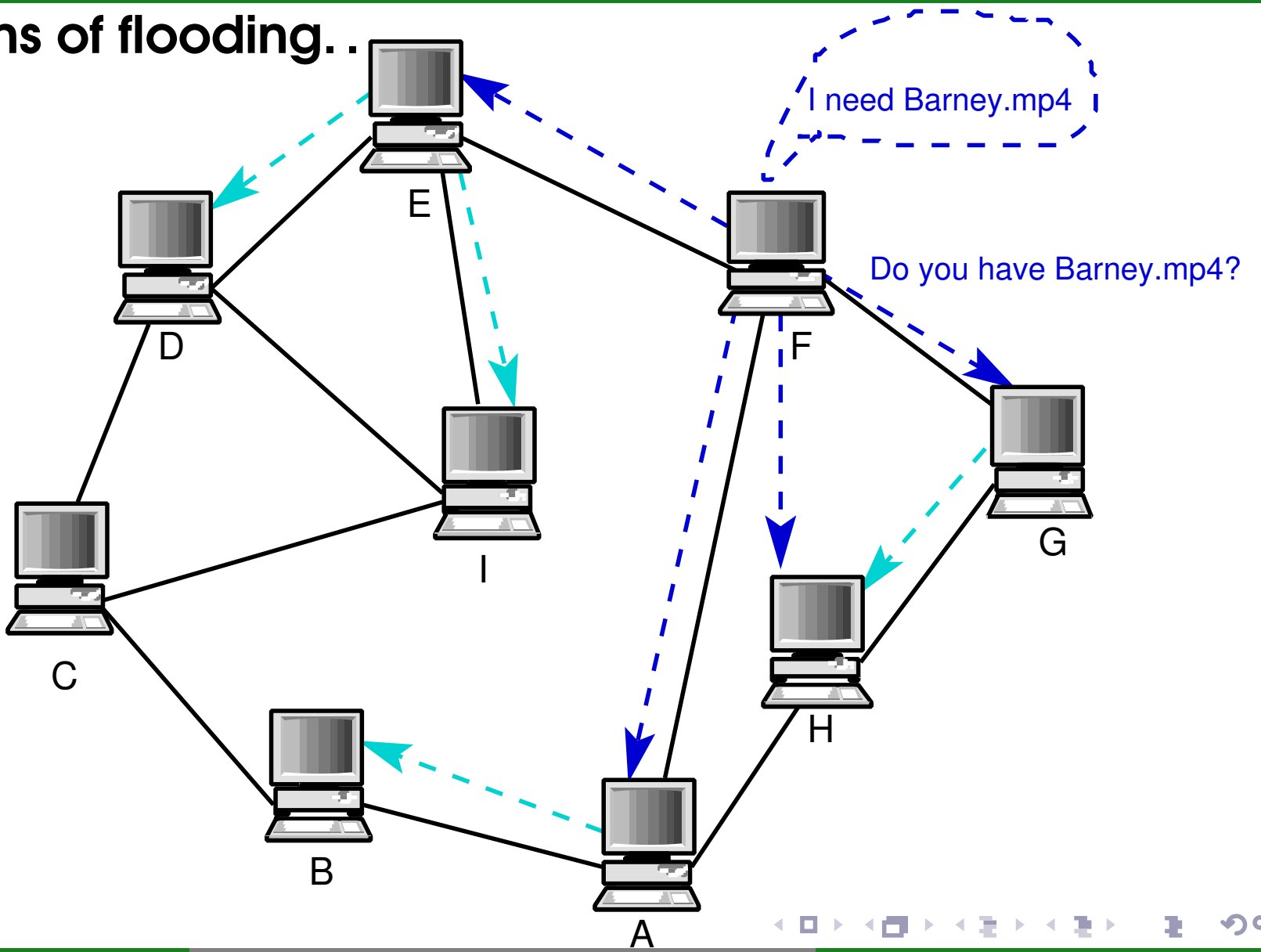
P2P Communication: How to find a peer which holds the required information?

By means of flooding...

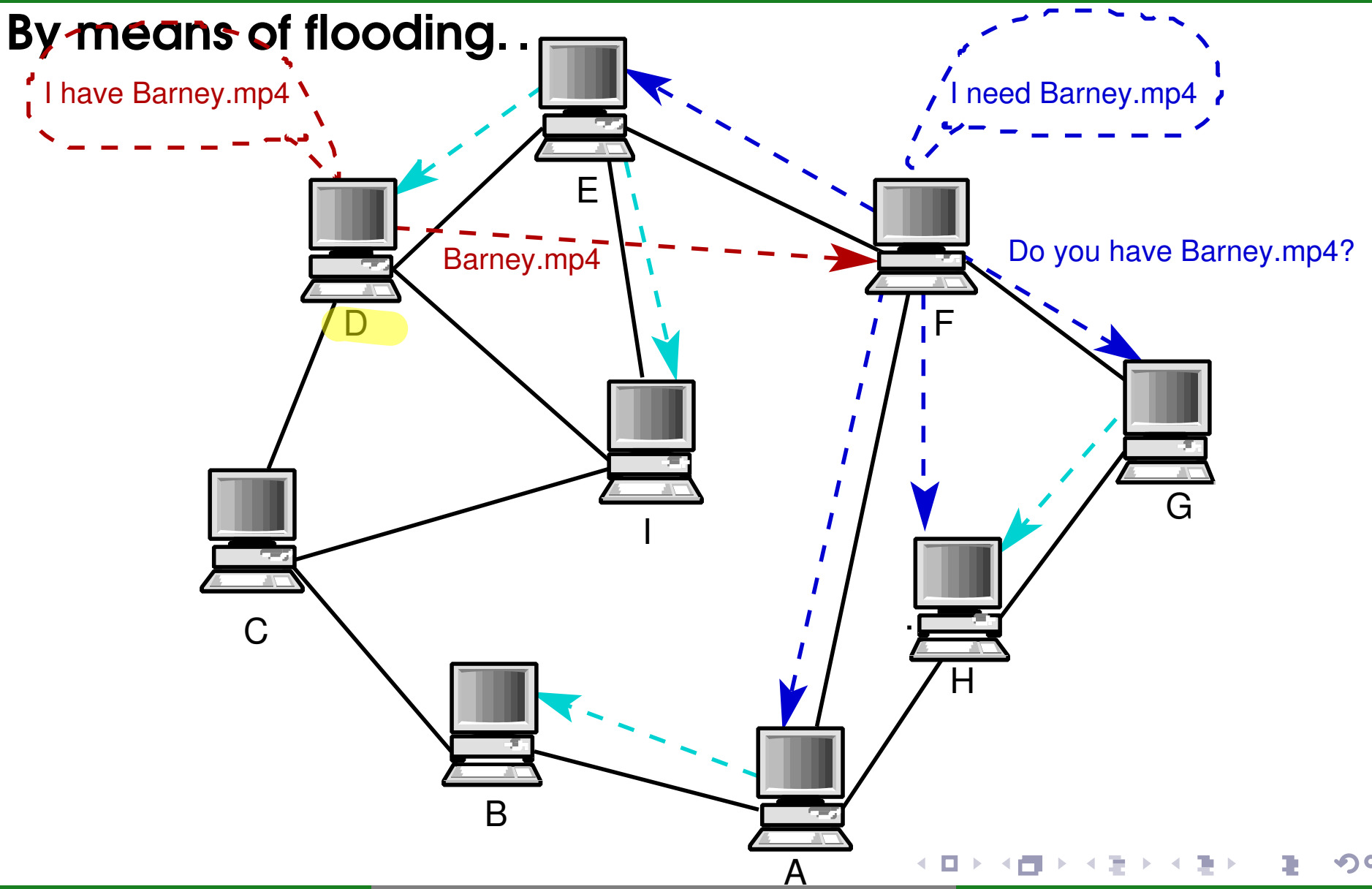


P2P Communication: How to find a peer which holds the required information?

By means of flooding..

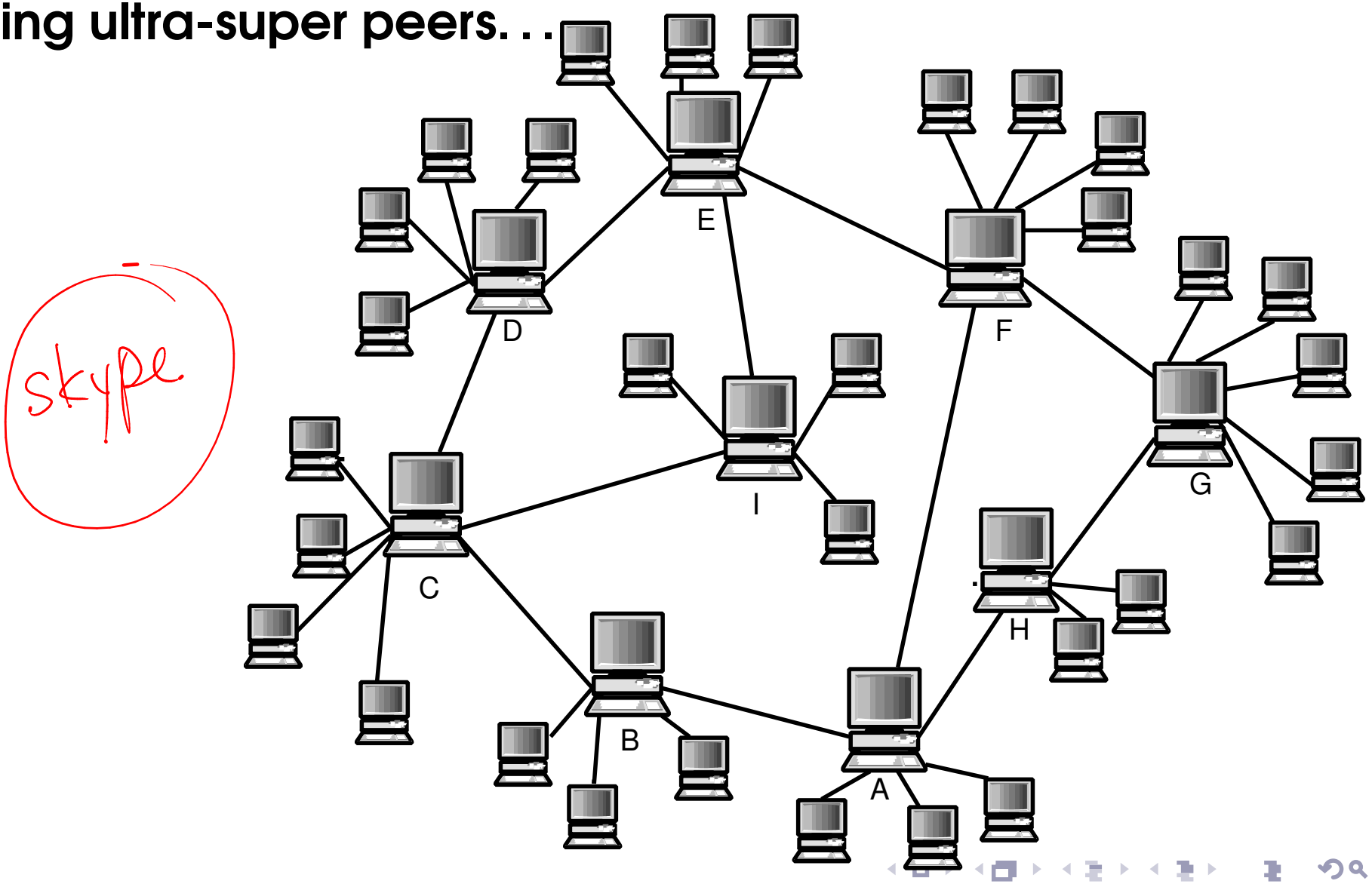


P2P Communication: How to find a peer which holds the required information?



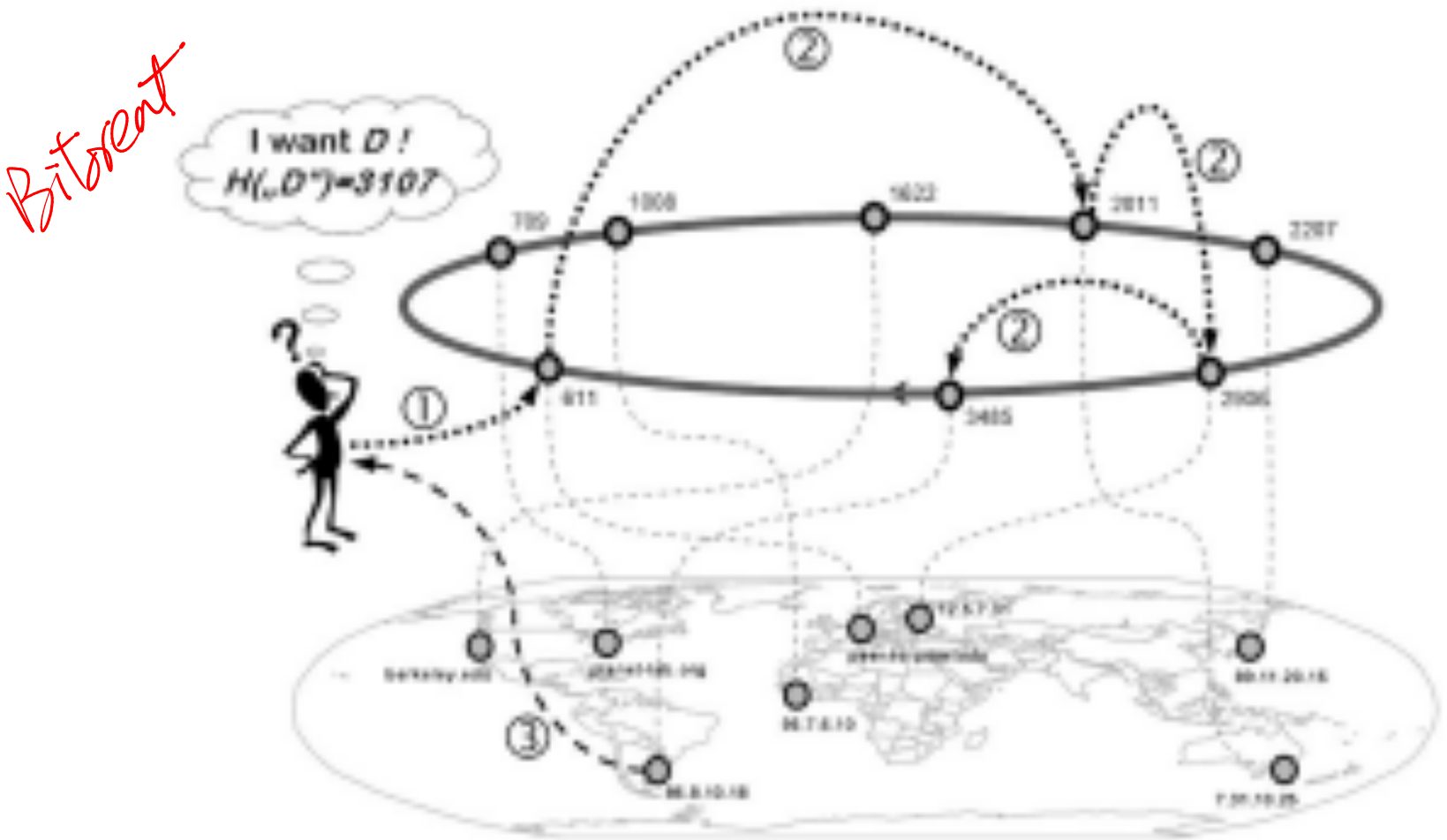
P2P Communication: How to find a peer which holds the required information?

Using ultra-super peers. . .



P2P Communication: How to find a peer which holds the required information?

Using distributed hash table...



Source: Figure 7.5 (Wehrle, Götz and Rieche, 2005)

Subsystem Communications
Peer-to-peer communication : An example

- The file sharing system *BitTorrent* is a well-known example of peer-to-peer system.
- *BitTorrent* uses a *distributed hash table* to facilitate storing and retrieving of large files.
  - ➠ Each peer knows which portions of files it keeps and also where to look for other chunks from other peers.

See pp.437–440 of Hawa (2010) and chapter 7 of Wehrle, Götz and Rieche (2005) for more detail.

- *Napster* is generally *not* considered as a *peer-to-peer* system.

  Napster uses *peer-to-peer* communication to share files amongst the peers. However, it uses a *centralised model* for locating files in peers.
  - ➠ A central server holds records of which files are stored in which peers.

Subsystem Communications
Peer-to-peer communication : An example

- The file sharing system *BitTorrent* is a well-known example of peer-to-peer system.
- *BitTorrent* uses a *distributed hash table* to facilitate storing and retrieving of large files.
  - ➠ Each peer knows which portions of files it keeps and also where to look for other chunks from other peers.

  See pp.437–440 of Hawa (2010) and chapter 7 of Wehrle, Götz and Rieche (2005) for more detail.
- *Napster* is generally *not* considered as a *peer-to-peer* system.

  Napster uses *peer-to-peer* communication to share files amongst the peers. However, it uses a *centralised model* for locating files in peers.
  - ➠ A central server holds records of which files are stored in which peers.

Subsystem Communications
Peer-to-peer communication : An example

- The file sharing system *BitTorrent* is a well-known example of peer-to-peer system.
- *BitTorrent* uses a *distributed hash table* to facilitate storing and retrieving of large files.
  - ➠ Each peer knows which portions of files it keeps and also where to look for other chunks from other peers.

  See pp.437–440 of Hawa (2010) and chapter 7 of Wehrle, Götz and Rieche (2005) for more detail.
- *Napster* is generally *not* considered as a *peer-to-peer* system.

  Napster uses *peer-to-peer* communication to share files amongst the peers. However, it uses a *centralised model* for locating files in peers.
  - ➠ A central server holds records of which files are stored in which peers.

Subsystem Communications
Peer-to-peer communication : An example

- The file sharing system *BitTorrent* is a well-known example of peer-to-peer system.
- *BitTorrent* uses a *distributed hash table* to facilitate storing and retrieving of large files.
  - ➠ Each peer knows which portions of files it keeps and also where to look for other chunks from other peers.

  See pp.437–440 of Hawa (2010) and chapter 7 of Wehrle, Götz and Rieche (2005) for more detail.
- *Napster* is generally *not* considered as a *peer-to-peer* system.

  Napster uses *peer-to-peer* communication to share files amongst the peers. However, it uses a *centralised model* for locating files in peers.
  - ➠ A central server holds records of which files are stored in which peers.

Service Oriented Architecture

- Large *distributed* systems may adopt *Service Oriented Architecture (SOA)* in its design.
- The basic unit of communication in SOA is the invocation of *remote services*.
- A *service* typically refers to an *XML Web service*, which:
  - communicates via Internet protocols (e.g. *HTTP*) and
  - sends and receives data in *XML* format.
- An *SOA* is a design model in which the application logic is encapsulated within *services* that interact via a common communications protocol.
  - ➠ The resulting system components are therefore *loosely coupled* with each other.

Service Oriented Architecture
Web Services

- A *service* suitable for use within SOA should be:
  - available *on demand* in the long term;
  - concisely, yet fully, described in a *standard* way, hence:
    - easy to connect to and use in any common programming environment;
    - having confined and predictable effects;
    - using standard formats for data exchange;
  - accessed in a stateless request-response manner
    - ➠ Each request is *self-contained*.
    - ➠ Normally, the history of *previous requests and responses* is not explicitly referred to.

Service Oriented Architecture
Web Services

- A *service* suitable for use within SOA should be:
  - available *on demand* in the long term;
  - concisely, yet fully, described in a *standard* way, hence:
    - easy to connect to and use in any common programming environment;
    - having confined and predictable effects;
    - using standard formats for data exchange;
  - accessed in a stateless request-response manner
    - ➠ Each request is *self-contained*.
    - ➠ Normally, the history of *previous requests and responses* is not explicitly referred to.

Service Oriented Architecture
Web Services

- A *service* suitable for use within SOA should be:
  - available *on demand* in the long term;
  - concisely, yet fully, described in a *standard* way, hence:
    - easy to connect to and use in any common programming environment;
    - having confined and predictable effects;
    - using standard formats for data exchange;
  - accessed in a stateless request-response manner
    - ➠ Each request is *self-contained*.
    - ➠ Normally, the history of *previous requests and responses* is not explicitly referred to.

Service Oriented Architecture
Web Services

- Typically, *services* expose business processes and *clients* use them to implement applications.

- The *client* of a *service* is itself a *service* (but for other clients).

  ➠ Hence, *Web services* fit in a *P2P* model, rather than a *client-server* model.

  E.g. in a holiday booking scenario, a client whose job is to book accommodation may use *services* from various hotel businesses. This client may also be a service provider for another client whose job is to act as a travel agent.

Service Oriented Architecture
Web Services

- Typically, *services* expose business processes and *clients* use them to implement applications.
- The *client* of a *service* is itself a *service* (but for other clients).

  ➠ Hence, *Web services* fit in a *P2P* model, rather than a *client-server* model.

  E.g. in a holiday booking scenario, a client whose job is to book accommodation may use *services* from various hotel businesses. This client may also be a service provider for another client whose job is to act as a travel agent.

Service Oriented Architecture
Web Services

- Typically, *services* expose business processes and *clients* use them to implement applications.
- The *client* of a *service* is itself a *service* (but for other clients).
  - ➠ Hence, *Web services* fit in a *P2P* model, rather than a *client-server* model.

E.g. in a holiday booking scenario, a client whose job is to book accommodation may use *services* from various hotel businesses. This client may also be a service provider for another client whose job is to act as a travel agent.

Service Oriented Architecture
Web Services

- Typically, *services* expose business processes and *clients* use them to implement applications.

- The *client* of a *service* is itself a *service* (but for other clients).

  ➠ Hence, *Web services* fit in a *P2P* model, rather than a *client-server* model.

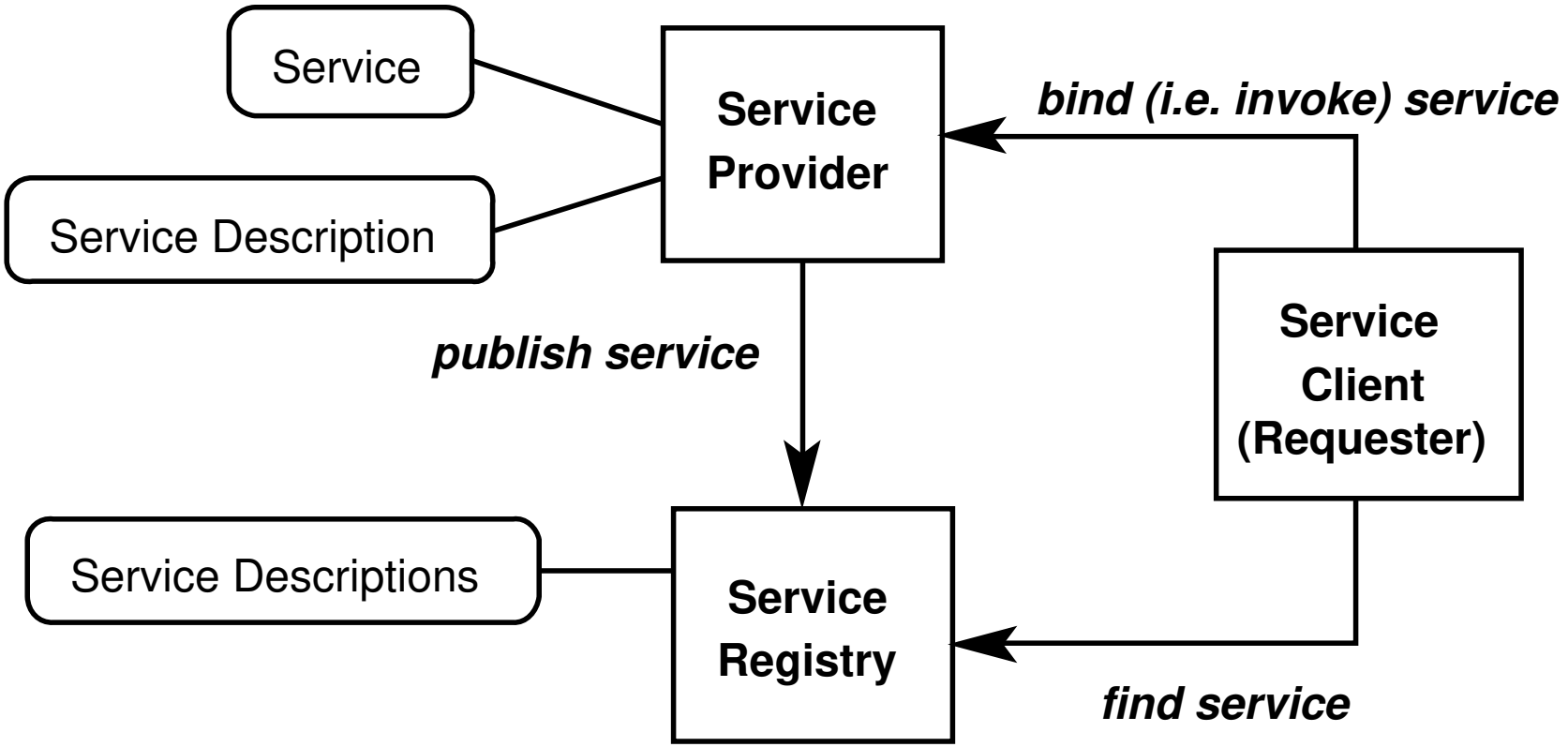
  E.g. in a holiday booking scenario, a client whose job is to book accommodation may use *services* from various hotel businesses. This client may also be a service provider for another client whose job is to act as a travel agent.

Service Oriented Architecture
Web Services

- Note that each *service* is a computing process which *does not* include human-computer interaction.
  E.g.

1. Service A requests available hotel rooms and prices for a set period.
2. Service B responses with the required info in XML format.
3. Service A then renders the info on its page for the user to view.

Service Oriented Architecture
Operations in SOA

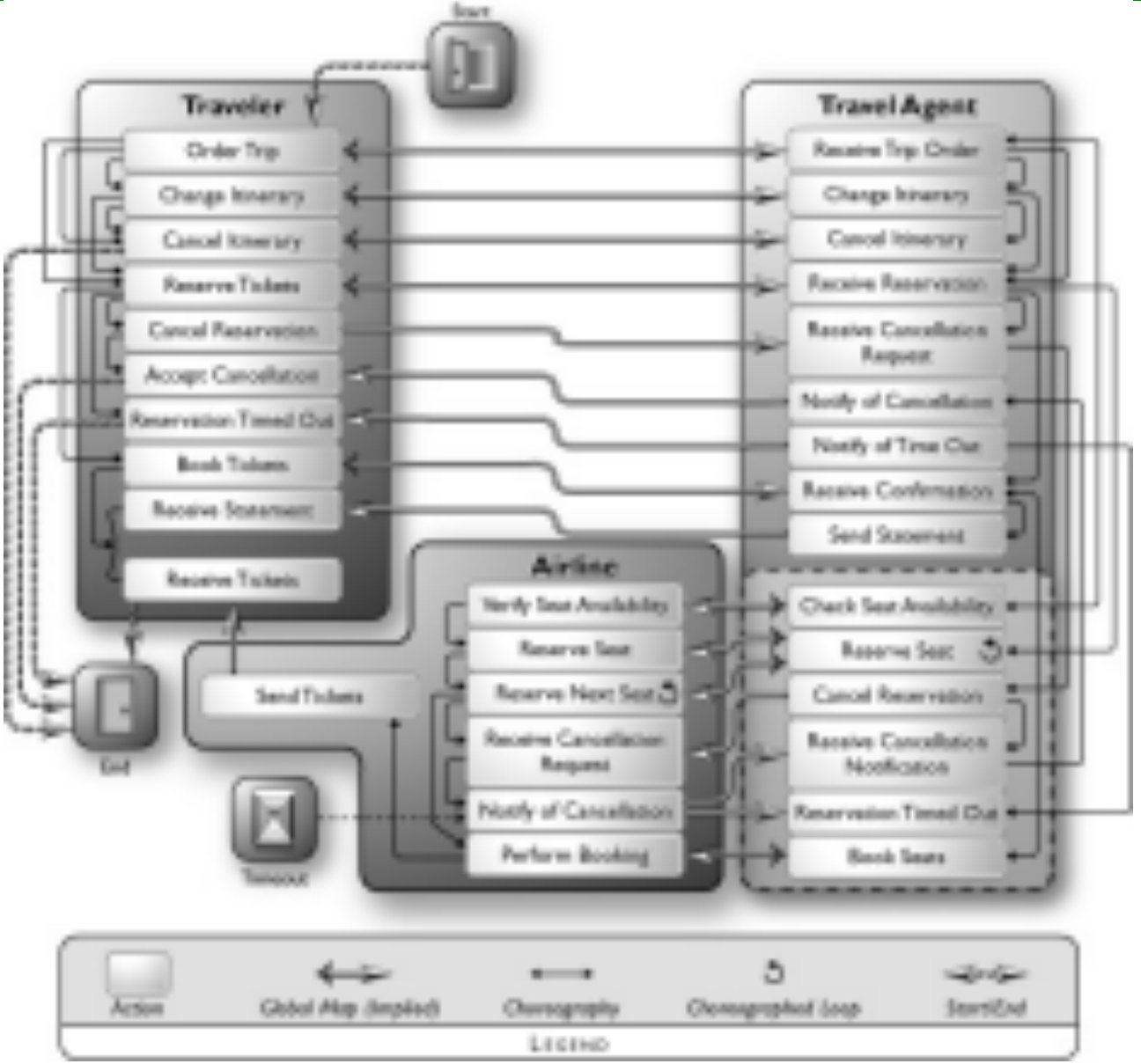


Papazoglou (2008, Section 1.6.2)

Service Oriented Architecture
A Holiday Booking Scenario

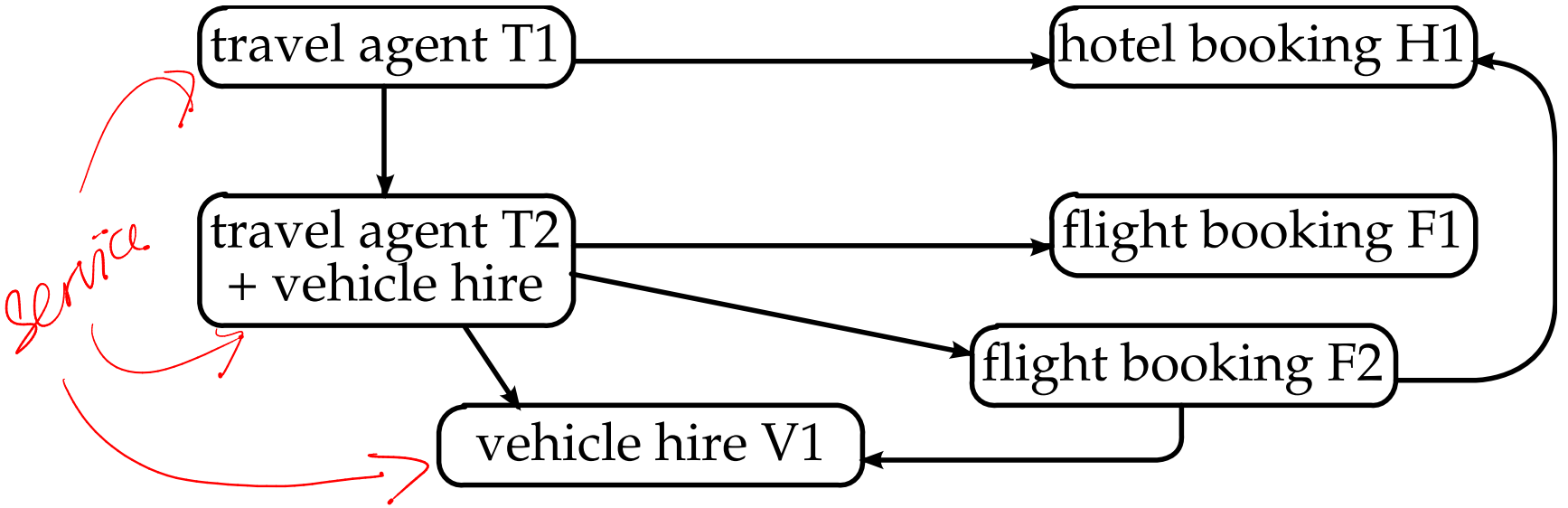
- An example of a SOA system is a network of *Web services* comprising and supporting *travel agents*.
- This system is composed of *services* for booking flights, hotels and car hire.
- Agent applications use these *services* to implement more sophisticated holiday-package services to its clients.

A Holiday Booking Scenario



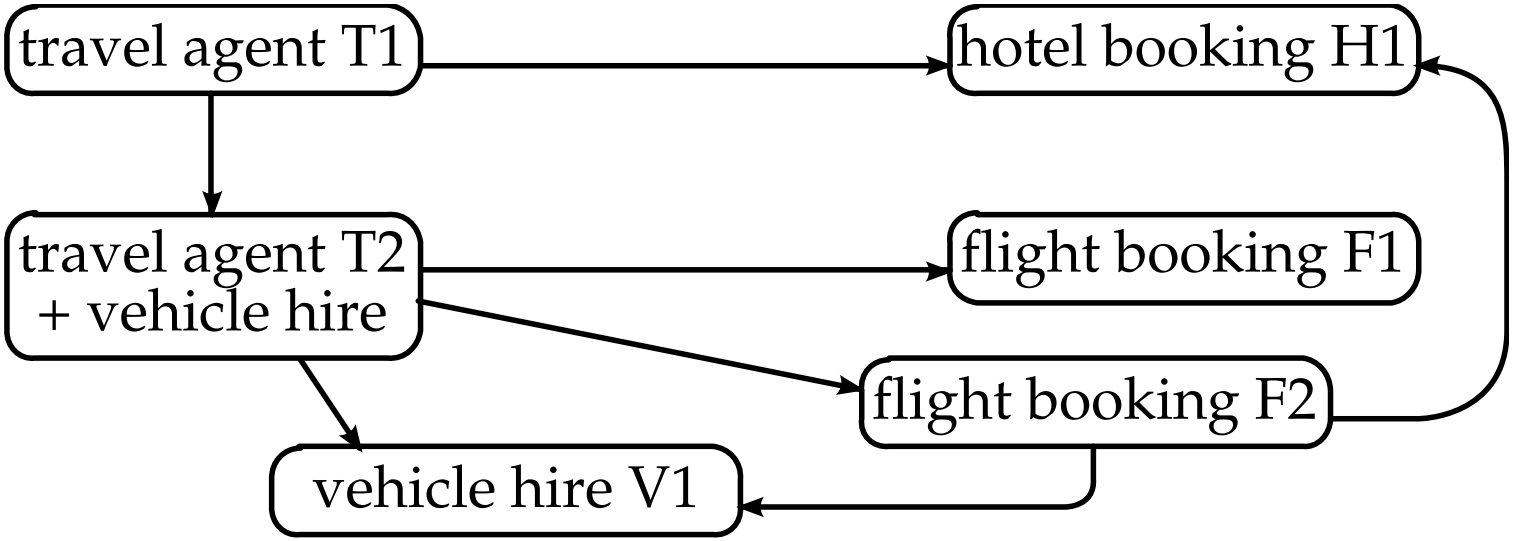
Source: Arkin et al. (2002, Figure 5-3)

Service Oriented Architecture
A Holiday Booking Scenario : a SOA example



- Each bubble is a *service* provided by potentially a *different* vendor.
- A *service* tend to use another service in order to complete its task.
- It should be *easy* to switch between the services of the same kind provided by different vendor.

Service Oriented Architecture
A Holiday Booking Scenario : a SOA example



- Each bubble is a *service* provided by potentially a *different* vendor.
- A *service* tend to use another service in order to complete its task.
- It should be *easy* to switch between the services of the same kind provided by different vendor.

Service Oriented Architecture
Common Principles (Erl 2004, Section 3.1.4)

- Reusable logic is divided into services.
- Services share a formal contract.
  - ➠ mainly regarding information exchange
- Services are *loosely coupled*.
- Services *abstract* underlying logic.
  - ➠ The only part of a service that is visible to the outside world is what is exposed via the service's description.
- Services are *composable*.
  - ➠ I.e. a service can be made up of other services.

Service Oriented Architecture
Common Principles (Erl 2004, Section 3.1.4)

- Services are *autonomous*.
- Services are *stateless*.
- Services are *discoverable*.
  - ➠ To discover a service means to locate "*a machine-processable description of a Web service that may have been previously unknown and that meets certain functional criteria*" (Haas & Brown, 2004).

Service Oriented Architecture
Why SOA?

Business use of SOA as opposed to *distributed objects* or *ad hoc Remote Procedure Call* (RPC) is justified by a promise of:

- *Easier reuse* of services for multiple purposes;
- *Better adaptability* to changing business environment and available technologies;
- Ability to integrate new and *legacy systems*;
- Ability to *cheaply* setup e-business links across the World.

References

- Hawa, M., "Cooperation Incentives: Issues and Design Strategies", in Antonopoulos, N., Exarchakos, G., Li, M. and Liotta, A. (eds), *Handbook of Research on P2P and Grid Systems for Service-Oriented Computing: Models, Methodologies, and Applications*, Volume 1, New York: Information Science Reference, 2010.

- Arkin, A., Askary, S., Fordin, S., Jekeli, W., Kawaguchi, K., Orchard, D., Pogliani, S., Riemer, K., Struble, S, Takacsi-Nagy, P., Trickovic, I. and Zimek, S., *Web Service Choreography Interface (WSCI) 1.0*, W3C Note, 8 August 2002, (Online). Available at: `http://www.w3.org/TR/wsci/` (02/11/2017).

References

- Bennett, S., McRobb, S. and Farmer, R., *Object-Oriented Systems Analysis and Design Using UML*, 4th ed, Maidenhead: McGraw-Hill, 2010.
- Booch, G., Rumbaugh, J. and Jacobson, I., *The UML specification documents*, Santa Clara, CA.: Rational Software Corp., 1997.
- Braude, E.J. and Bernstein, M.E., *Software Engineering: Modern Approaches*, 2nd ed, Hoboken, NJ: Wiley, 2011.
- Eckstein, R., *Java SE Application Design With MVC*, 2007, (Online). Available at: `http://www.oracle.com/technetwork/articles/javase/index-142890.html` (02/11/2017).

References

- Eeles, P., *What is a software architecture?*, 2006, (Online). Available at: `http://www.ibm.com/developerworks/rational/library/feb06/eeles/` (02/11/2017).
- Erl, T., *Service-Oriented Architecture: A field guide to integrating XML and Web Services*, Upper Saddle River, New Jersey: Pearson Education, 2004.
- Haas, H. and Brown, A., *Web Services Glossary*, W3C Working Group Note, 11 February 2004, (Online). Available at: `http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/` (02/11/2017).
- Jacobson, I., Booch, G. and Rumbaugh, J., *The Unified Software Development Process*, Reading, MA: Addison-Wesley; ACM Press, 1999.

References

- IEEE Computer Society, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems: IEEE Std 1472000, 2000.
- Kruchten, P., *The Rational Unified Process: An Introduction*, Reading, MA: Addison-Wesley, 2004.
- Larman, C., *Applying UML and patterns*, 3rd ed, Upper Saddle River, NJ: Prentice Hall, 2005.
- Lunn, K., *Software Development with UML*, London: Palgrave Macmillan, 2003.

References

- Microsoft MSDN Library, *Physical Tiers and Deployment*, chapter 19, 2011, (Online). Available at: `http://msdn.microsoft.com/en-us/library/ee658120.aspx` (02/11/2017).
- Papazoglou, M., *Web Services: Principles and Technology*, Harlow: Pearson Education, 2008.
- Ramirez, A.O., 'Three-Tier Architecture', *Linux Journal*, Issue 75, July, 2000.
- Reenskaug, T.M.H., *Model-View-Controller (MVC)*, (Online). Available at: `http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html` (03/11/2017).

References

- Wehrle, K., Götz, S. and Rieche, S.,"Distributed Hash Table", in Steinmetz, R. and Wehrle, K. (ed), *Peer-to-peer Systems and Applications*, LNCS 3485, Berlin: Springer-Verlag, 2005, pp. 79–93.

- Shaw, M. and Garlan,D., *Software Architecture – Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

- Swartz, F., *Model-View-Controller (MVC) Structure*, 2004, (Online). Available at: `http://leepoint.net/notes-java/GUI/structure/40mvc.html` (02/11/2017).

For more detail, see:
Chapter 13 of Bennett et al. (2010) and
Chapter 18 of Braude & Bernstein (2011).

Learning Outcomes

Learning Outcomes. You should now be able to

- explain what is meant by software architecture
- specify the role of a system architect
- describe the characteristics of a range of architectural styles
- identify if a simple software system follows the MVC architecture
- apply the MVC architecture