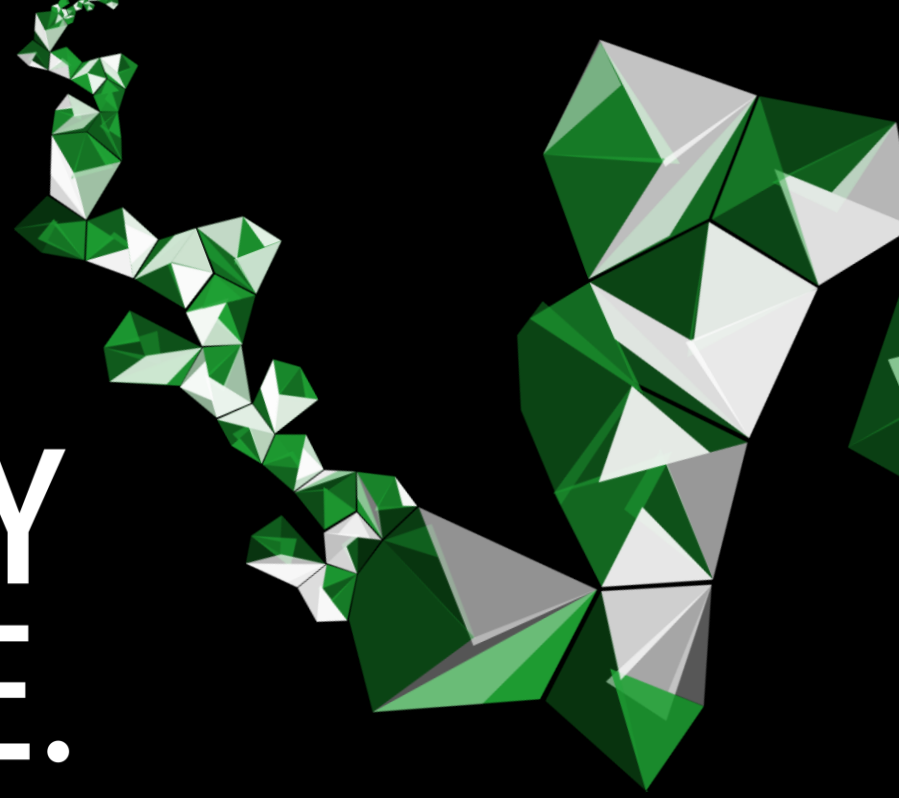


UNIVERSITY OF TWENTE.

ARCHITECTURE OF INFORMATION SYSTEMS (AIS)

LECTURE 2 – THE ROLE OF MIDDLEWARE FOR INTEROPERABILITY IN ENTERPRISE ARCHITECTURE

GOALS OF THIS WEEK

After this week you should be able to

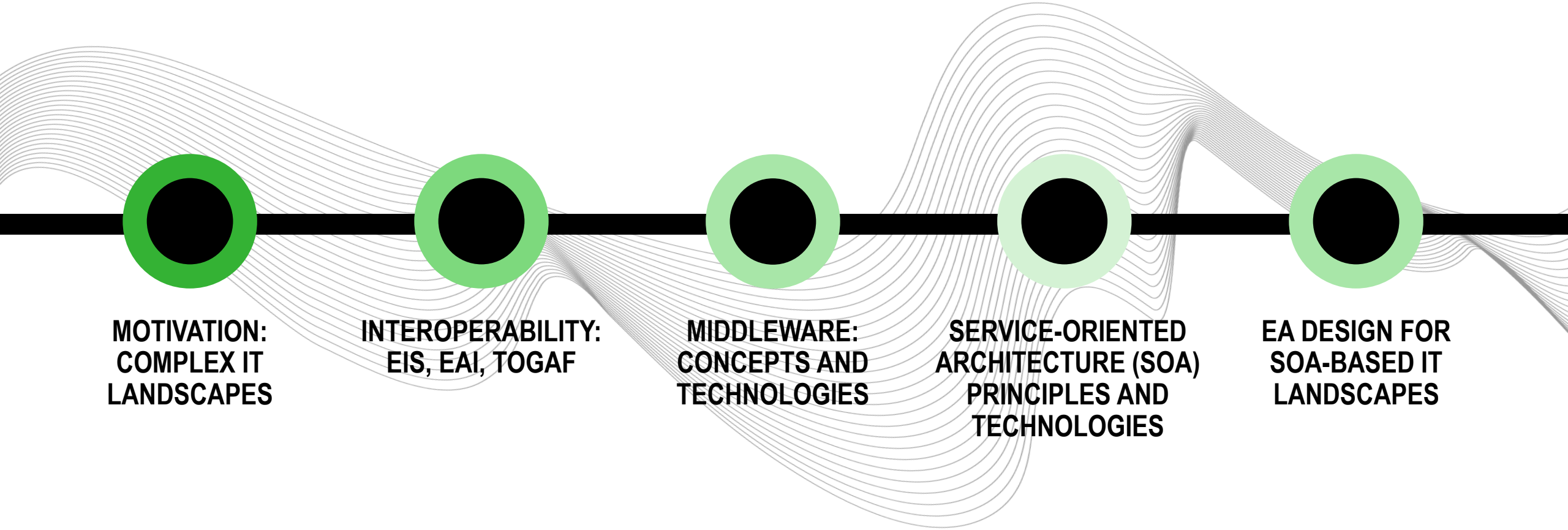
LG.2.1. *Explain* the main interoperability aspects of an EIS; the interoperability requirements, and *explain* the relations to the TOGAF Architecture Development Method (ADM), and to the Enterprise Application Integration (EAI) perspective

LG.2.2. *Explain why* middleware is important to address interoperability, *list* the most common types of middleware technology, and *explain* the underlying principles: tiers, layers, communication styles, and messaging)

LG.2.3. *Explain* SOA principles, web service technologies (e.g., RESTful), the microservices style, and messaging integration techniques, e.g., Enterprise Service Bus (ESB) and API Gateways

LG.2.4. *Design* SOA-based Archimate application integration viewpoints that cover business processes served by web services that are provided by applications and their technology services; along with the associated requirements realization

IN THIS PRESENTATION:



MOTIVATION: COMPLEX IT LANDSCAPES

INTEROPERABILITY: EIS, EAI, TOGAF

MIDDLEWARE: CONCEPTS AND TECHNOLOGIES

SERVICE-ORIENTED ARCHITECTURE (SOA) PRINCIPLES AND TECHNOLOGIES

EA DESIGN FOR SOA-BASED IT LANDSCAPES

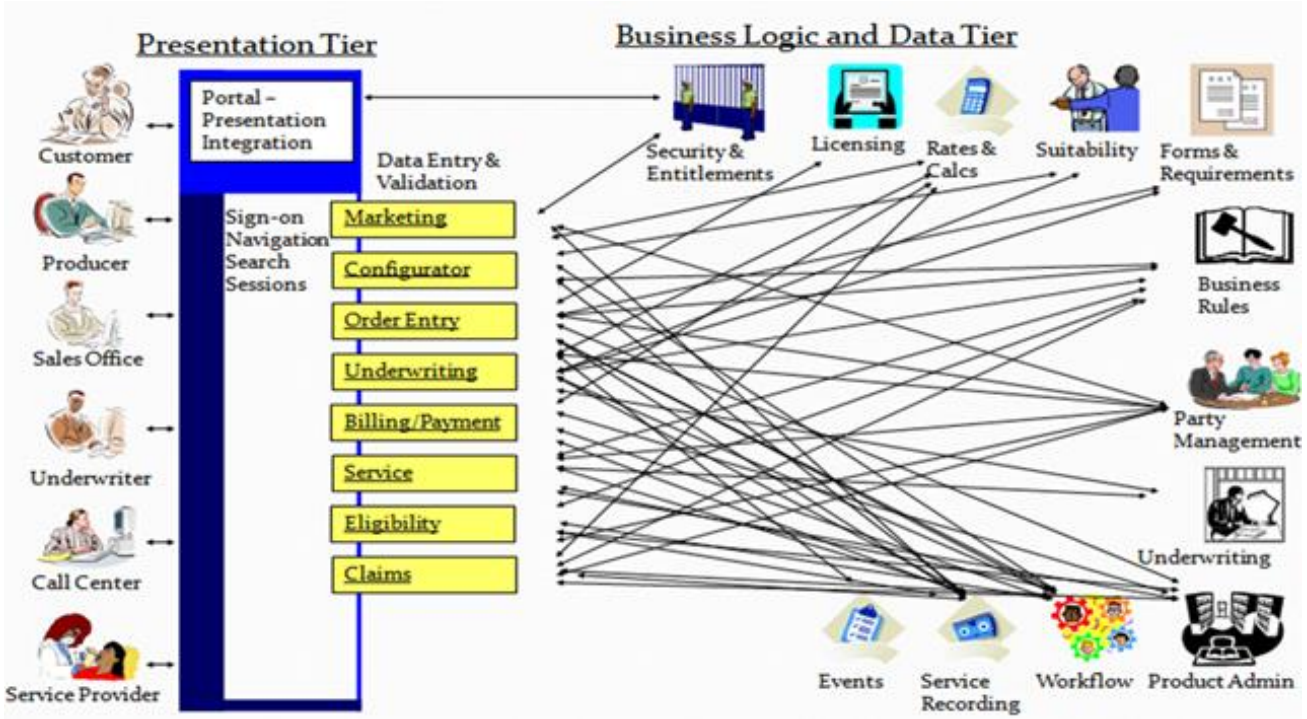
LAST LECTURE

- Project-based course: EA with focus on interoperability of complex IS architectures
- EIS improves business processes by integrating diverse systems, e.g., ERP, SCM, CRM, but may cause an "spaghetti architecture"
- Enterprise evolution depends on how to deal with integration and interoperability problems, which are associated to automation islands
- 5 main interoperability aspects that are related to application integration approach, which are implemented by some types of middleware
- Web services are technologies to solve integration problems at business and IT levels, supporting Service-Oriented Architecture (SOA) as a design principle
- The main elements of the architectural approach are: baseline and target architectures, migration roadmap, gap and impact analysis

COMPLEX IT LANDSCAPES

'SPAGHETTI ARCHITECTURE'

REMEMBER



EIS: Enterprise IS

- Point-to-point connections
- Unmanageable architecture

Software maintenance typically requires 40-60% (some cases 90%) of the total lifecycle effort devoted to a software. Typical effort devoted to maintenance

- 50% maintenance costs came from understanding code
- Amount of code to be maintained doubled every 7 years

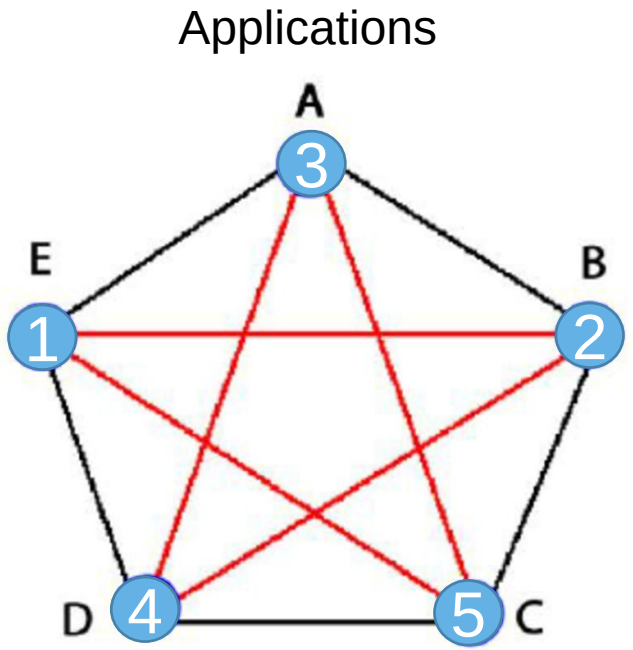
UNIVERSITY OF TWENTE.

ENTERPRISE APPLICATION INTEGRATION

- Each app may have different access mechanisms and data encoding rules
- These apps must be interconnected in order to communicate with each other
- A naïve approach is to connect apps one-by-one

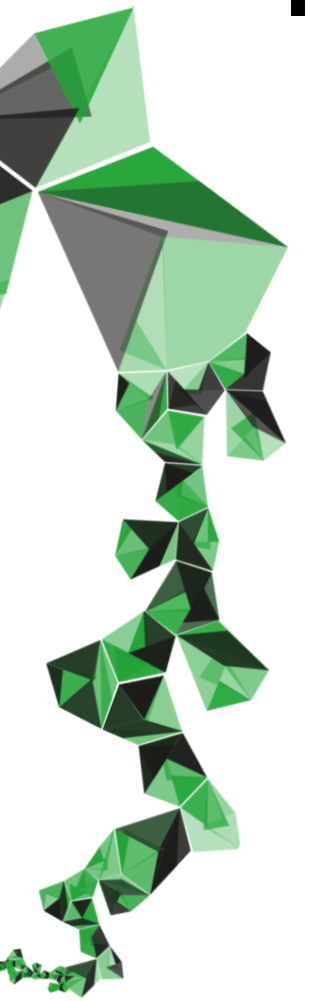
$$\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$$

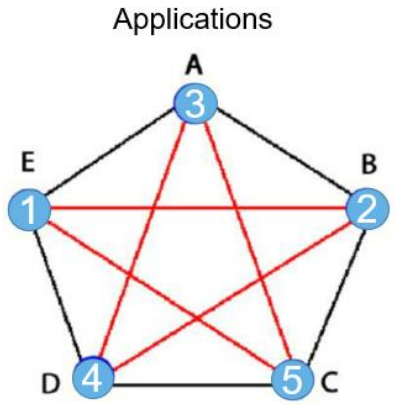
- For *n* apps we get *n*(*n*-1)/2 possibly different connections (k = 2)
  - 10 apps → 45 point-to-point connections
  - 100 apps → 4.950 connections…



UNIVERSITY OF TWENTE.

INTEROPERABILITY





"ability of two or more systems or components to exchange information and to use the information that has been exchanged" (IEEE, 1990)

Semantic interoperability
- Pragmatic interoperability
- Process interoperability
- Syntactic interoperability
- Physical interoperability

- Interoperability: systems A and B can exchange data, and they can interpret and use the received data
- Integration: uniform combining systems (A, B) into C (system) such that users of C can use functions and data of A and B

Enterprise applications often cannot directly exchange or share information as required by the business processes

- Manual mediation: tedious and error-prone
- Automated mediation: middleware solutions

UNIVERSITY OF TWENTE.

INTEROPERABILITY IN TOGAF

1. **Operational or Business:** how business processes are to be shared
2. **Information:** how information is to be shared
3. **Technical:** how technical services are to be shared (connect to one another)

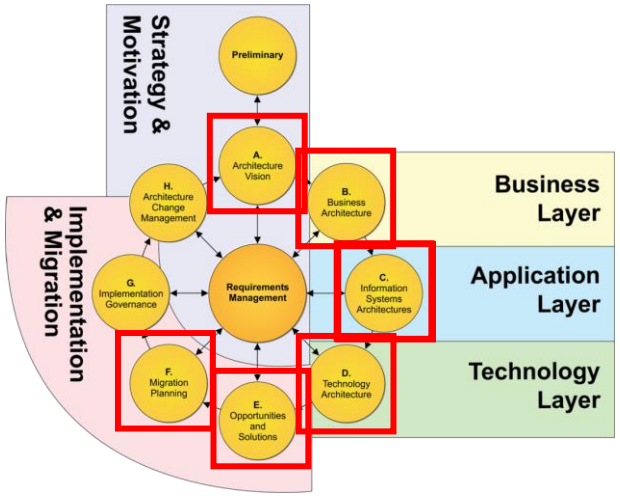
IT perspective → Enterprise Application Integration (EAI) is about middleware

A. **Presentation:** common look-and-feel, e.g., portal-like solution for processes
B. **Information:** corporate information shared among various applications to realize business processes, "based upon a commonly accepted corporate ontology"
C. **Application:** corporate functionality is integrated and shareable so that the applications are not duplicated and are seamlessly linked together
D. **Technical:** methods and shared services for the communication, storage, processing, and access to data (…) based on standards and/or IT platforms

INTEROPERABILITY IN TOGAF
ADM: ARCHITECTURE DEVELOPMENT METHOD

"Defining the degree to which the information and services are to be shared is a very useful architectural requirement, especially in a complex organization and/or extended enterprise"

"strongly recommended best practice": system of systems or federated systems



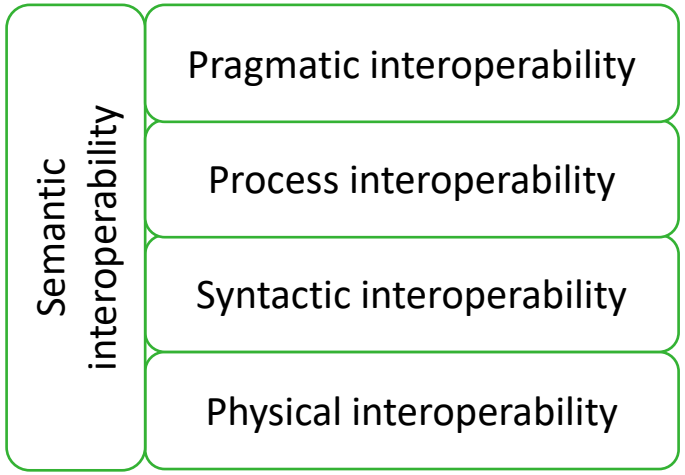
Information and service exchanges according to TOGAF ADM →

Phase A: nature and security (business scenarios)
Phase B: defined in business terms
Phase C (data): information content detailed using corporate data and information exchange model
Phase C (app): the way applications communicate to share the information and services
Phase D: technical mechanisms (middleware)
Phase E: Commercial Off-The-Shelf (COTS) packages
Phase F: Plan how interoperability is 'logically implemented'

UNIVERSITY OF TWENTE.

INTEROPERABILITY
CLASSIFICATIONS

Interoperability aspects



TOGAF

1. Operational or Business
2. Information
3. Technical

EAI

1. Presentation
2. Information
3. Application
4. Technical

European Interoperability Framework (EIF)

Interoperability Maturity Models…

Exercise: in pairs, discuss the relations among these (15min)

a) How syntactic interoperability is covered by TOGAF and EAI?
b) How about physical, process and semantic interoperability?
c) Can you map the interoperability aspects to EIF layers?

UNIVERSITY OF TWENTE.

INTEROPERABILITY
DISCUSSION AND BREAK

Exercise: in pairs, discuss the relations among these (15min):

- Interoperability aspects
- TOGAF, EAI and EAF classifications

a) How syntactic interoperability is covered by TOGAF and EAI?
b) How about physical, process and semantic interoperability?
c) Can you map the interoperability aspects to EIF layers?

✓ LG.2.1. *Explain* the main interoperability aspects of an EIS; the interoperability requirements, and *explain* the relations to the TOGAF Architecture Development Method (ADM), and to the Enterprise Application Integration (EAI) perspective

DISTRIBUTED IS ARCHITECTURE

TIERS AND LAYERS

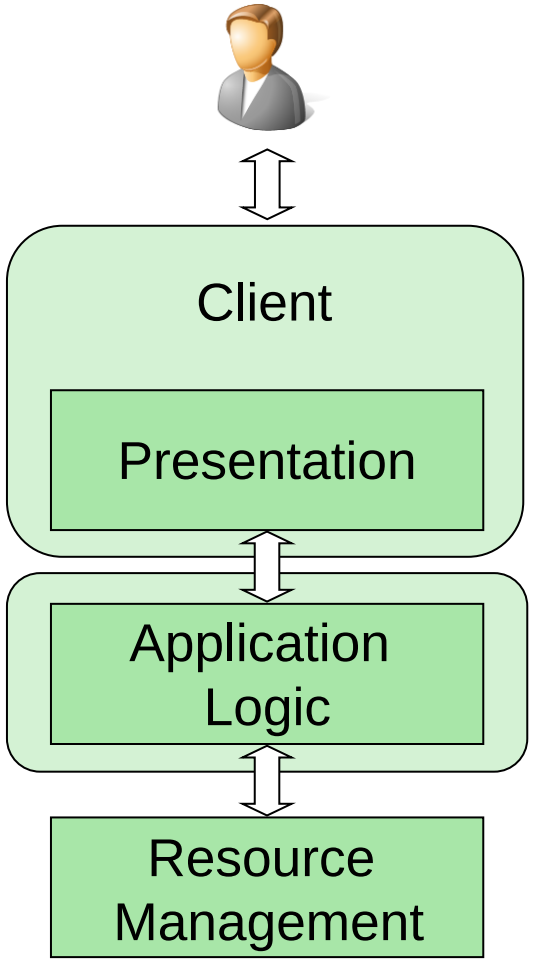
Architectures for distributed applications can be characterised by their number of tiers: 1-tier, 2-tier, 3-tier and N-tier

- Logical layers
  - Presentation
  - Application (or Integration) logic
  - Resource management
- A tier combines functionality of the logical layers, aiming at mapping it onto physically distributed parts

Any issues with the book terminology?

"Logical tier" or "logical layer" = layer

"one tier does not necessarily mean one physical system"



REMEMBER

- Evolution of 2-tier architecture
- Proper architecture for integration
- Avoids that integration occurs via the client

Middle tier: not only application logic, but also integration!

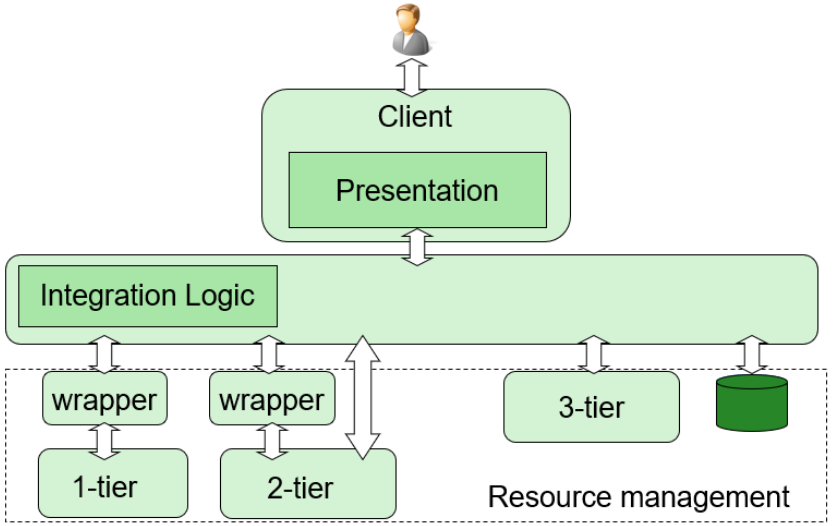
UNIVERSITY OF TWENTE.

DISTRIBUTED IS ARCHITECTURE

3-TIER AND N-TIER WITH WEB SERVER

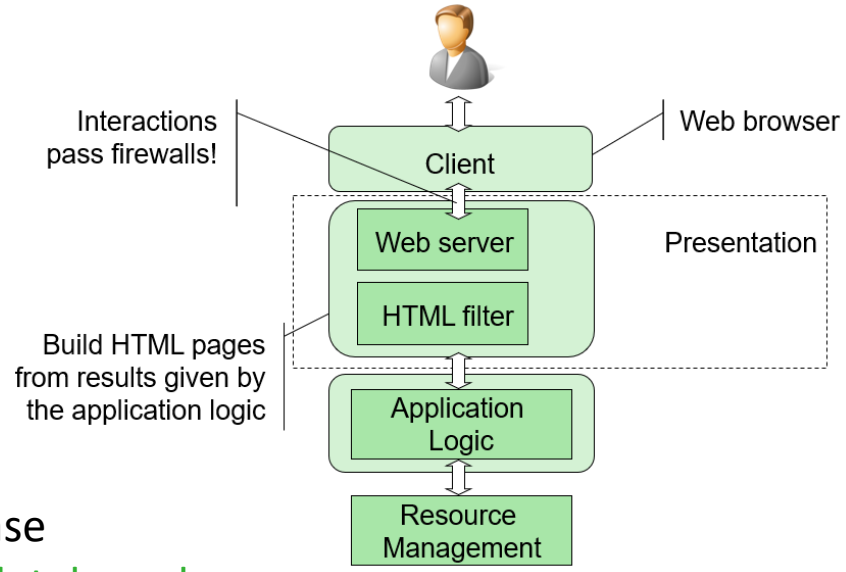
REMEMBER

Three-tier architecture: integration



N-tier architecture with web server

- Web server forms additional tier
- Evolution of 3-tier architecture
- Setting for introducing web



Transaction: unit of work that updates resurce(s), e.g., a database

- Transactions are either completed (committed) or are completely undone
- Transactions are important because organizational tasks are transactional

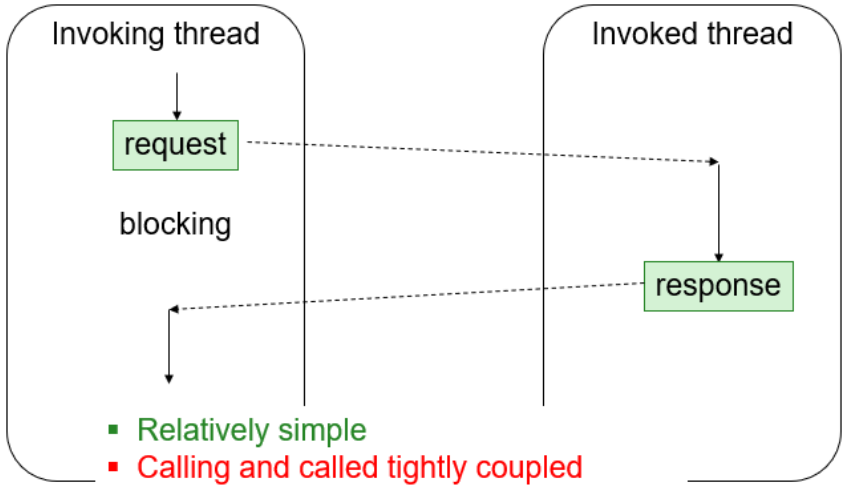
UNIVERSITY OF TWENTE.

DISTRIBUTED IS ARCHITECTURE

COMMUNICATION STYLES

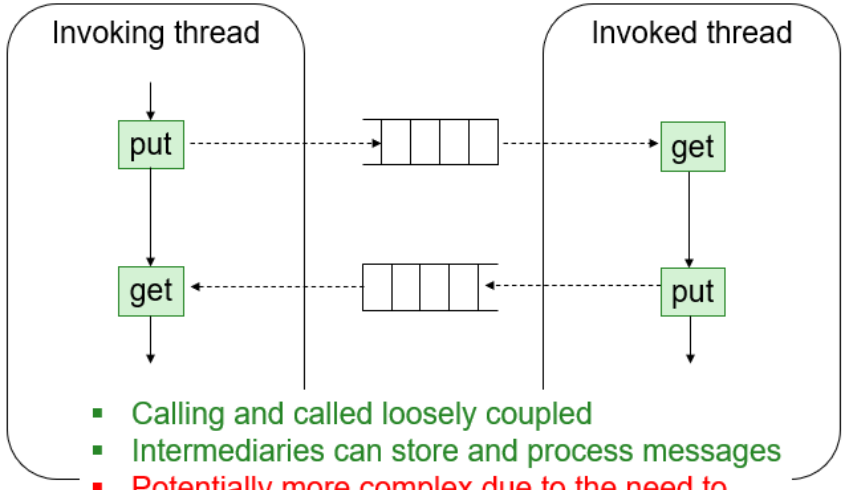
- Middleware infrastructures cope with the interactions between tiers
- Two classic forms of communication

Synchronous communication



- Relatively simple
- Calling and called tightly coupled
- Poor performance if processing takes too much time or for multiple tiers

Asynchronous communication

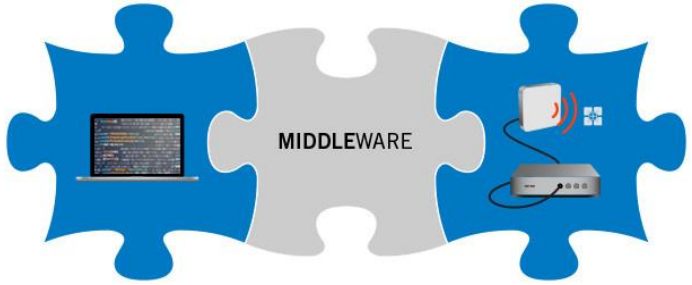


- Calling and called loosely coupled
- Intermediaries can store and process messages
- Potentially more complex due to the need to synchronise

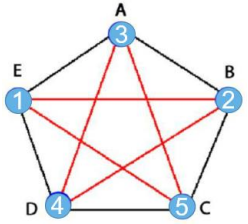
From programmer's perspective!

UNIVERSITY OF TWENTE.

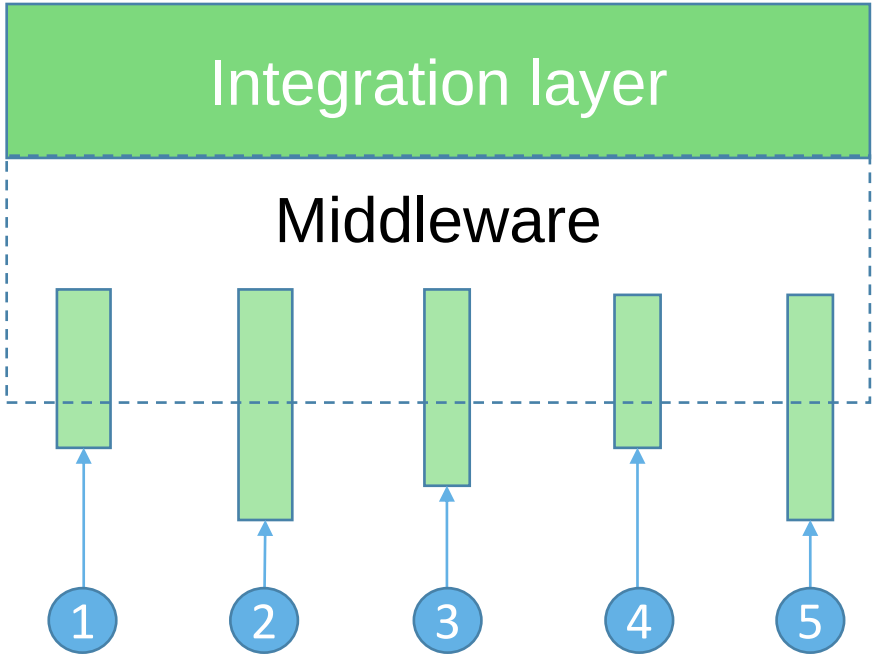
MIDDLEWARE
THE THING IN THE MIDDLE



Without middleware →



With middleware:

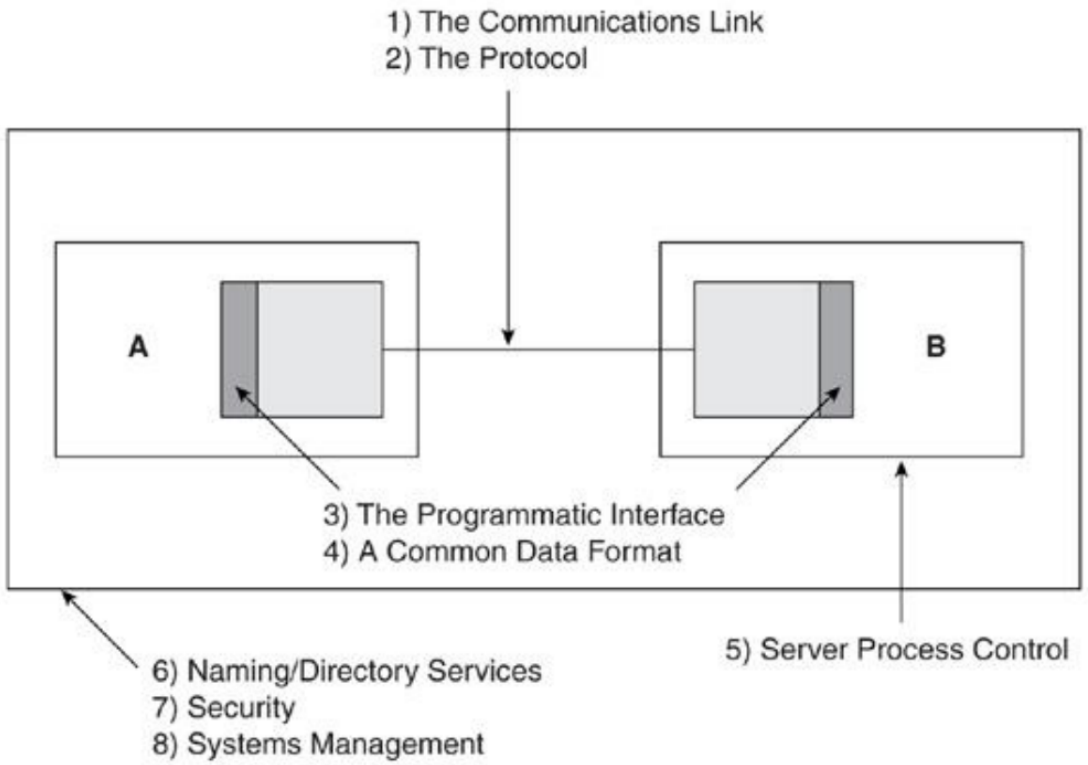


Middleware enables interaction between heterogeneous and distributed software applications

- Define common (standard) communication mechanism and encapsulate apps in this mechanism, so that these apps can communicate in the same way
- (Standard) communication mechanism forms a layer
- Most popular example: REST architectural style (RESTful services) that exchange data in JSON or XML formats

UNIVERSITY OF TWENTE.

MIDDLEWARE
INTEROPERABILITY



Example

- A: app1 or appX (presentation layer)
- B: app2 or appX (application layer)

Elements 1-4: sufficient to ensure the required communication between A and B

The programmatic interface and the common data format together define the way that A and B communicate with the middleware

The common data format describes how the data should be structured so that both A and B understand it

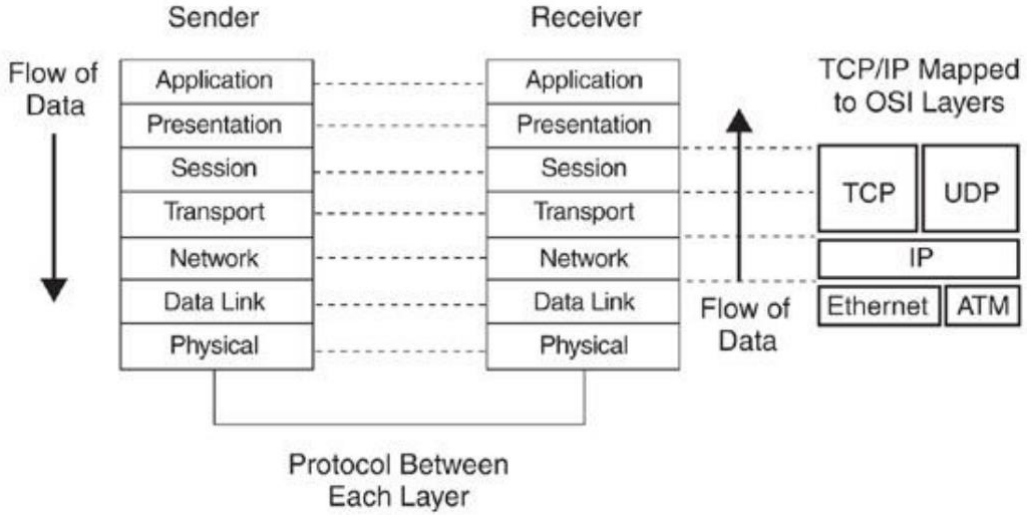
Programmatic interface specifies the way the data are presented to the middleware

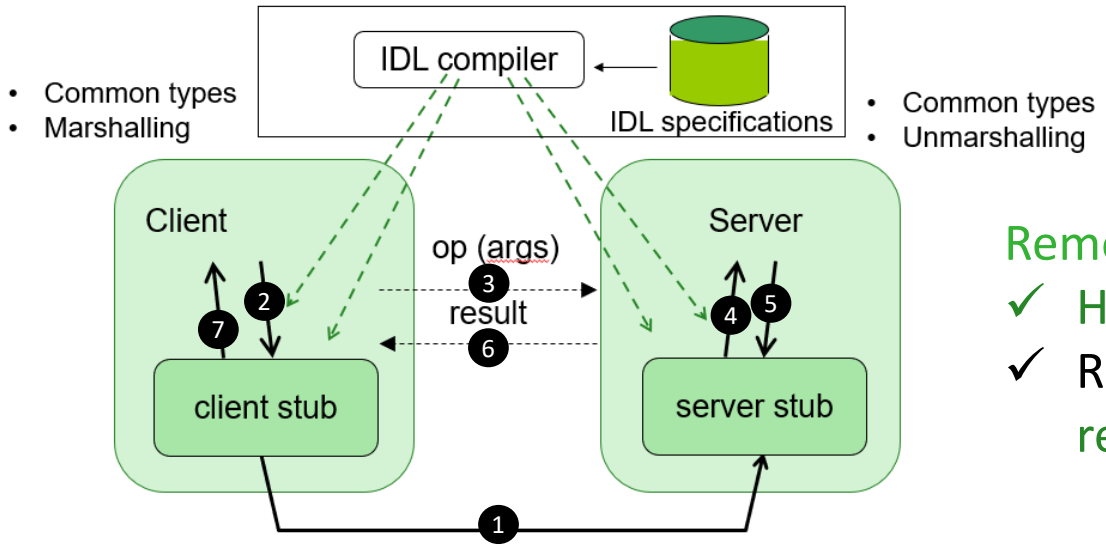
MIDDLEWARE
THE THING IN THE MIDDLE

REMEMBER

‘Network stack’: OSI conceptual model

Standard of communication functions of a telecommunication system or computing system, without any regard to the system's underlying internal technology and specific protocol suites





Remote procedure calls (RPC) principles

- ✓ Hide the complexity of the low-level interfaces
- ✓ RPC realises the abstraction of ‘calling a procedure remotely’

UNIVERSITY OF TWENTE.

MIDDLEWARE
TRADITIONAL APPROACHES AND TYPES

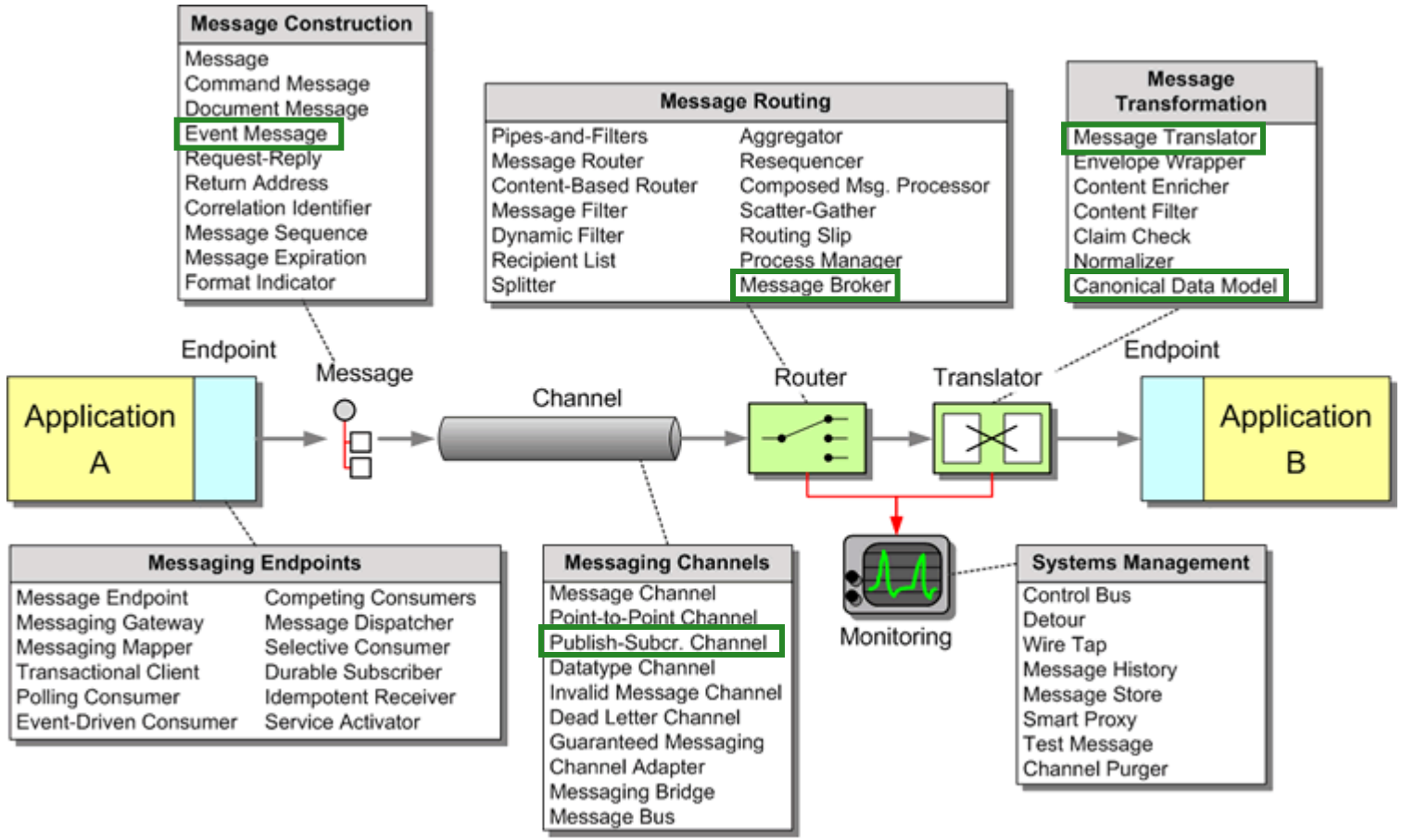
Application Integration

1. File Transfer
2. Shared Database
3. Remote Procedure Invocation
4. Messaging

Types of middleware

1. RPC-based middleware (Web services)
2. Message-oriented middleware / MQ / Message brokers / Enterprise service bus
3. Distributed Transaction Processing
4. Object request brokers / monitors
5. Remote Database middleware

(…)



65 messaging patterns

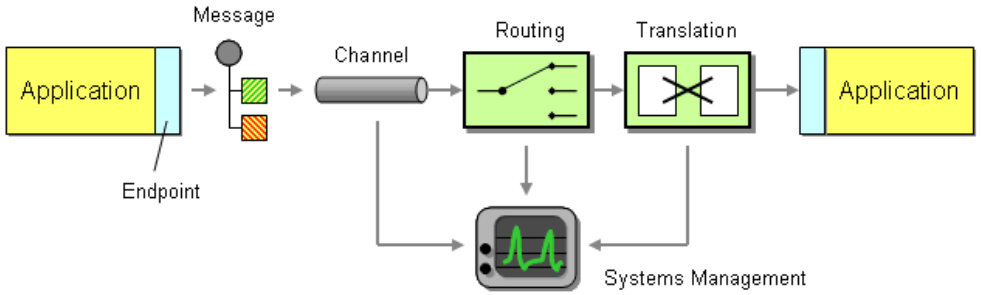
UNIVERSITY OF TWENTE.

MIDDLEWARE
LOOSE COUPLING

Loose coupling: Property of distributed systems: a change to one subsystem (application component, service) should not require a change to another

Integration style of chosen middleware impacts loose coupling, e.g.

- MQs handle messages as strings of bits (no IDL, no marshalling), and leaves loose coupling to applications



- WS/SOAP is tied to a specific networking protocol: prevents applications from using alternative network features
- RPC, CORBA, WS use separate interface definitions, which can make it easier to generate client and server stubs for different technology stacks
- Java RMI (a platform-specific RPC flavor) is heavily tied to the Java platform, which favors Java implementation of the client and server

MIDDLEWARE
ORCHESTRATION VS CHOREOGRAPHY

Orchestration: there is one central component that coordinates the work done by other parties

- May grow from client that calls multiple servers, e.g. using synchronous communication, according to some process logic in order to get work done
- Central "brain", any changes in distributed system need consideration of orchestrator

Choreography: parties react on each other using some scheme, but there is no central coordinator

- May be implemented with event-based communication, where events are announced by publisher(s) and reacted upon by subscribers
- More decoupling, parties can subscribe to events of interest and do their thing

MIDDLEWARE
REAL-TIME AND DEFERRABLE MESSAGES

Transactional services may process

1. Real-time
2. Deferrable messages

From business process perspective!

"The key difference between real-time and deferrable is what happens if the message cannot be sent now, not how long it takes."

Can you build real-time transaction calls with asynchronous calls?

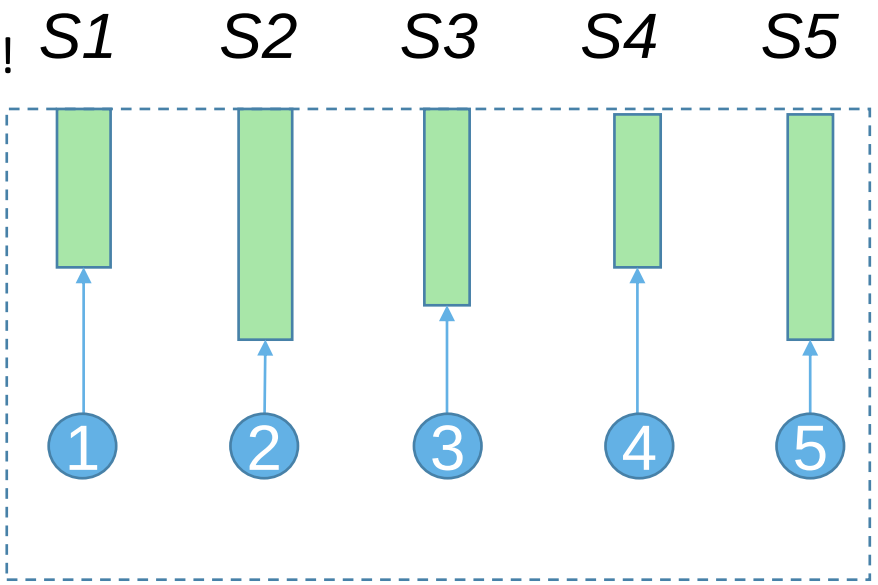
- ✓ LG.2.2. *Explain why* middleware is important to address interoperability, *list* the most common types of middleware technology, and *explain* the underlying principles: tiers, layers, communication styles, and messaging)

SERVICE-ORIENTED ARCHITECTURE
PRINCIPLES AND TECHNOLOGIES

REMEMBER

- Functionality is offered to potential users in terms of services
- Design principle: systems designed by composing services
- Built on top of a middleware layer (platform), i.e., standardised communication mechanisms
- SOA is design discipline (architectural solution) not a system!

Only services!



Flavors (technologies)

"SOA 1.0": SOAP-based
- Web services implemented on top of the SOAP protocol
- Often described using WSDL (language for describing services)
- Considered as 'heavyweight'

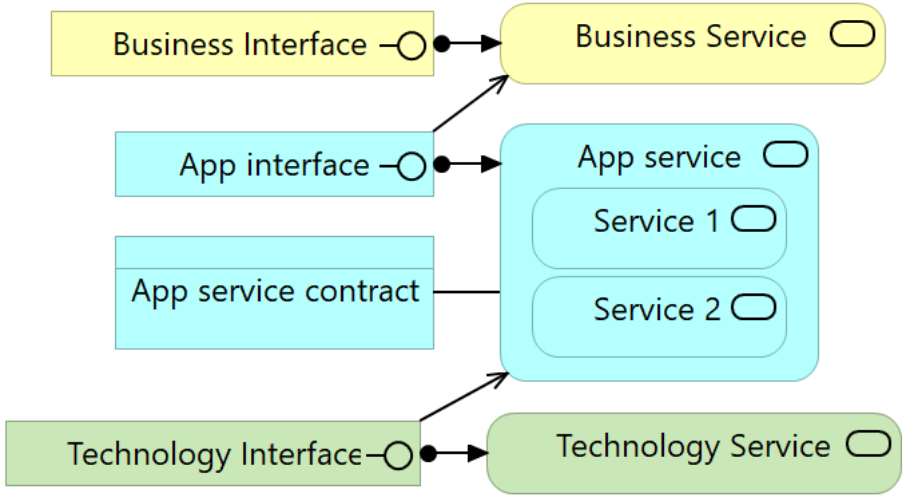
"SOA 2.0": RESTful
- Based on the REST principles (URLs, resources, HTTP methods, stateless communication)
- Increasingly popular, mainly for ad hoc service usage
- Considered as 'lightweight'

UNIVERSITY OF TWENTE.

SERVICE-ORIENTED ARCHITECTURE
SERVICE DEFINITION

A service

- is a logical representation of a repeatable business activity that has a specified outcome (e.g., check customer credit, provide weather data, consolidate drilling reports)
- is self-contained [complete in itself]
- may be composed of other services
- is a "black box" to consumers of the service (encapsulates its functionality)



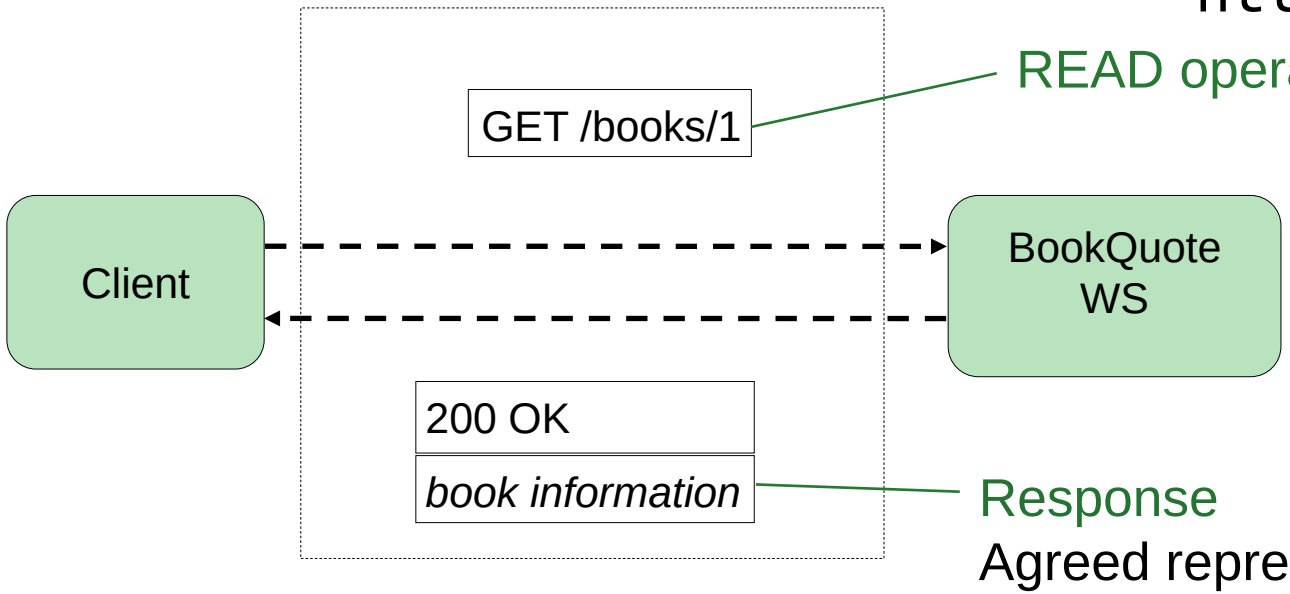
From http://opengroup.org/soa/source-book/soa/soa.htm

SERVICE-ORIENTED ARCHITECTURE
RESTFUL SERVICES

Web service ≠ web page!

identified by an URI

`http://somehost/books/1`



Self-descriptive messages → consequence of stateless communication

SERVICE-ORIENTED ARCHITECTURE
REST CHARACTERISTICS

- REST stands for REpresentational State Transfer
  > *Video: An Introduction to REST and JSON (Oracle)*
- Architectural style for invoking services over the Internet
  - Architectural style: set of design rules that identify the kinds of components and connectors that may be used to compose a system
- REST is therefore not a system that you can buy, but a style to build systems
- Defined in the PhD thesis of Roy Fielding (UCLA, 2000), still the standard reference for REST

SERVICE-ORIENTED ARCHITECTURE
REST PRINCIPLES

- Stateless client-server architecture, implying that request messages are self-contained (self-descriptive) → all necessary information for invoking a service is in the request message
- Web services are viewed as resources and are identified by their URIs
  → URIs offer a global addressing space for services
- Web service clients and providers choose a representation to send application content to the each other
  - Client and provider have a mutual understanding of the meaning of data since there is no formal way to describe web service interfaces

SERVICE-ORIENTED ARCHITECTURE
REST PRINCIPLES

- Use of a small globally defined set of remote methods that describe the actions to be performed on the resource
- Create-Read-Update-Delete (CRUD) actions
- In summary
  - Stateless client-server
  - Resources identified via URIs
  - State representation
  - CRUD actions
  - Natural match for HTTP

(no coincidence: Roy Fielding was one of developers of HTTP!)

REST action	HTTP method
Create	POST
Read	GET
Update	PUT
Delete	DELETE

SERVICE-ORIENTED ARCHITECTURE
DATA REPRESENTATION

- Different representations can be used for the information exchanged with a REST service
- Most popular are XML and JSON (JavaScript Object Notation)
- XML
  - Markup language used to represent data structures
  - Quite intuitive, but rather verbose
- JSON
  - Simplified format to represent data structures
  - Quite popular nowadays → less verbose, fits well with JavaScript

SERVICE-ORIENTED ARCHITECTURE
JSON-API

- JSON-API is a specification for how a client should request resources and how a server should respond to those requests
- Standardize implementation decisions of RESTful APIs, e.g., pagination, resources vs collections, metadata, related resources
- When? 2017 (Ember)
- Why was created? Standardize REST API requests and responses
- Relevance nowadays: convention widely adopted (e.g., Social Media)

https://jsonapi.org/

Content-Type: application/vnd.api+json

JSON API
EXAMPLE

```
{
  "links": {
    "self": "http://example.com/articles",
    "next": "http://example.com/articles?page[offset]=2",
    "last": "http://example.com/articles?page[offset]=10"
  },
  "data": [{
    "type": "articles",
    "id": "1",
    "attributes": {
      "title": "JSON:API paints my bikeshed!"
    },
    "relationships": {
      "author": {
        "links": {
          "self": "http://example.com/articles/1/relationships/author",
          "related": "http://example.com/articles/1/author"
        },
        "data": { "type": "people", "id": "9" }
      },
      "comments": {
        "links": {
          "self": "http://example.com/articles/1/relationships/comments",
          "related": "http://example.com/articles/1/comments"
        },
        "data": [
          { "type": "comments", "id": "5" },
          { "type": "comments", "id": "12" }
        ]
      }
    },
    "links": {
      "self": "http://example.com/articles/1"
    }
  }],
  "included": [{
    "type": "people",
    "id": "9",
    "attributes": {
      "firstName": "Dan",
      "lastName": "Gebhardt",
      "twitter": "dgeb"
    },
```

SERVICE-ORIENTED ARCHITECTURE
OPENAPI SPECIFICATION

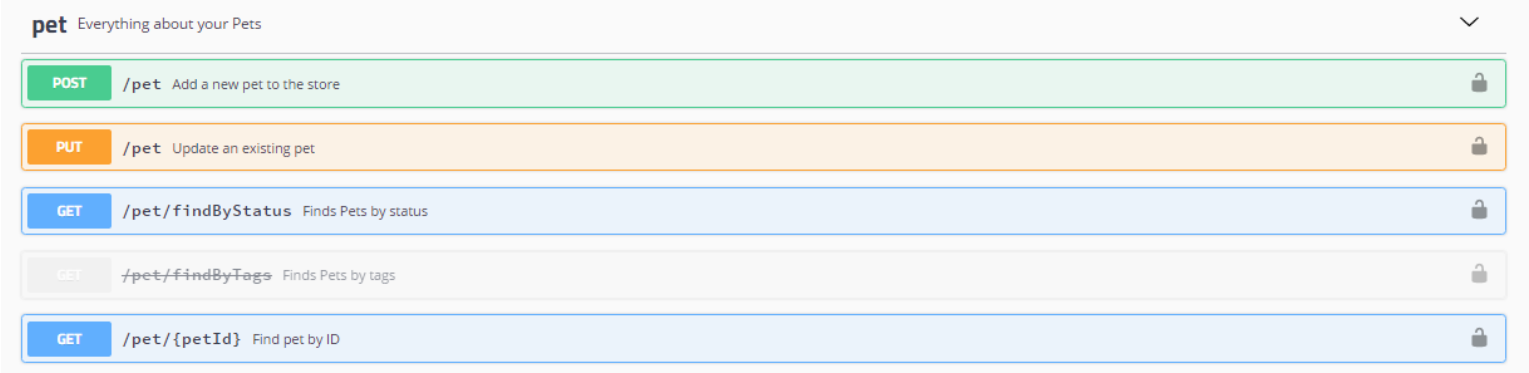
- OpenAPI Specification (OAS): language-agnostic interface to RESTful APIs that allows humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or network traffic inspection
- Consumer can understand and interact with the remote service with a minimal amount of implementation logic
- OpenAPI definition: used by documentation generation tools (to display the API), code generation tools (to generate servers and clients), testing tools, and other use cases

https://jsonapi.org/

http://spec.openapis.org/oas/v3.0.3

SERVICE-ORIENTED ARCHITECTURE
OPENAPI IMPLEMENTATION (SWAGGER)

- Swagger is a set of open-source tools built around the OAS for design, build, document and consume REST APIs
- Major Swagger tools
  - Swagger Editor: browser-based editor to write OpenAPI specs
  - Swagger UI: renders OpenAPI specs as interactive API documentation
  - Swagger Codegen: generates server stubs and client libraries



https://petstore.swagger.io/

https://jsonapi.org/

https://swagger.io/specification/

UNIVERSITY OF TWENTE.

SERVICE-ORIENTED ARCHITECTURE
MICROSERVICES

- SOA only prescribes the use of services, so it does not prescribe
  - Specific protocols (middleware) for the services to interact
    - Hence the different ‘service flavours’
  - The granularity (‘size’) of the services
    - Services can be as big as an application or as small as single data object
  - How the services are deployed (‘installed’ for execution)
    - Services deployed in-house or in the cloud, and user wouldn’t notice!

❑More guidelines necessary to achieve performance (elasticity) and fast deployment goals - streaming applications (Netflix, Spotify)

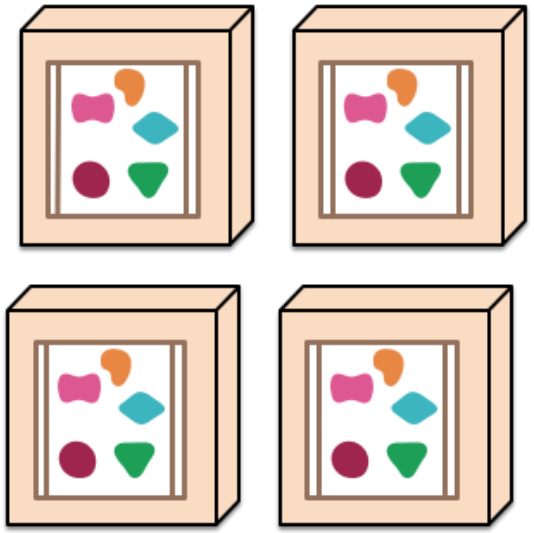
- Microservices
  - ‘Smaller’ service assigned to a single development team
  - Opposed to a Monolith (‘bigger’ application)

SERVICE-ORIENTED ARCHITECTURE
MICROSERVICES

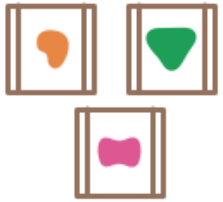
"the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API"
(Martin Fowler)

A monolithic application puts all its functionality into a single process...

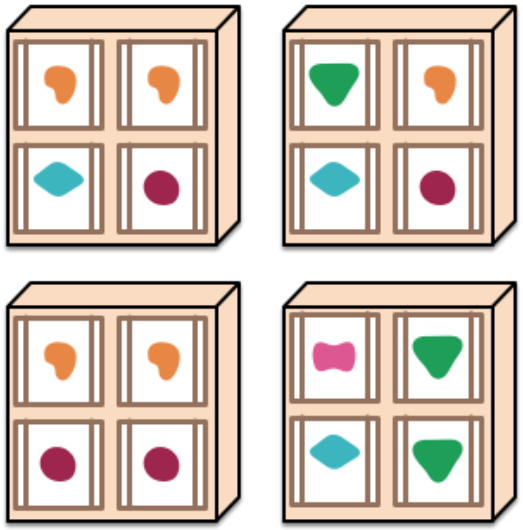
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...



... and scales by distributing these services across servers, replicating as needed.



MICROSERVICES
PRINCIPLES (SUMMARY FROM [ZIMMERMAN 2017])

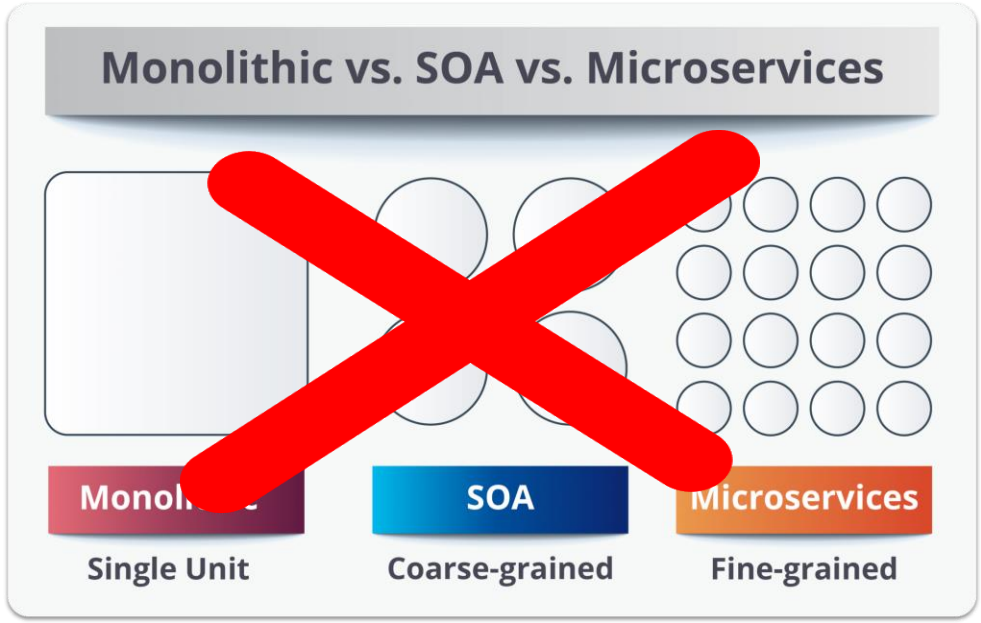
1. Componentised via services with own process and lightweight communication mechanisms (e.g., containers and REST)
2. Organised around business capabilities
3. Designed for failure (failures are isolated)
4. Designed with decentralisation in mind (for intelligence, governance and data management)
5. Profits from infrastructure automation (culture of automation)

MICROSERVICES
TYPICAL CHARACTERISTICS

- Is ‘small’ or ‘fine-grained’ (whatever that means!)
- Uses RESTful services
- Is deployed in a container, usually cloud environment
- Developed by a single development team, which is responsible for the whole lifecycle, including maintenance!
- Business-driven development according to the principle of Domain-Driven Design (microservices are identified by analysing domain models)

  Conclusion: special case of Service-Oriented Architecture!

SOA AND MICROSERVICES



That's… not really accurate!

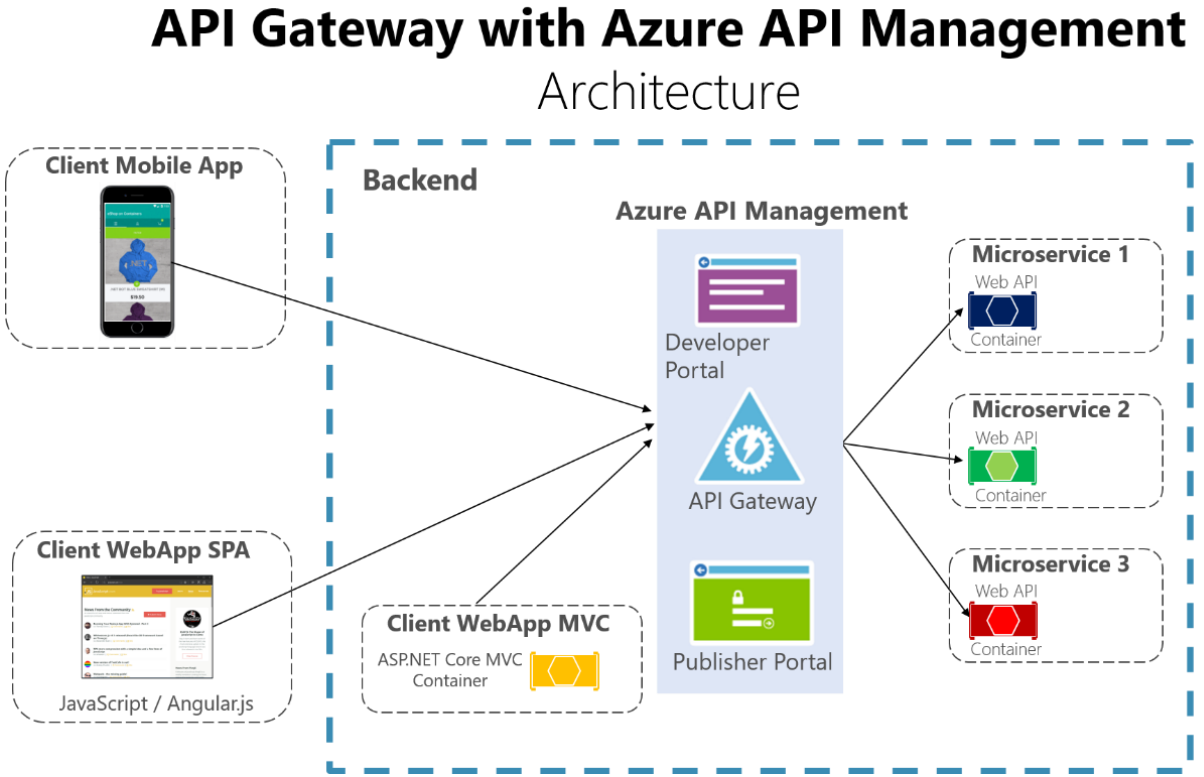
"The microservices approach has emerged from real-world use, taking our better understanding of systems and architecture to do SOA well. So you should instead think of microservices as a specific approach for SOA in the same way that XP or Scrum are specific approaches for Agile software development"

From [Newman 2015]

MICROSERVICES
API GATEWAYS: CONTROL ACCESS TO APIS

API Gateway

- Authenticator
  - Client authentication
  - Access control (permissions)
- Load balancing
- Request router
- Rate limiter
- Exception handler
- Centralized logging
- Request/response manipulation
- Service discovery of backends
- TLS termination

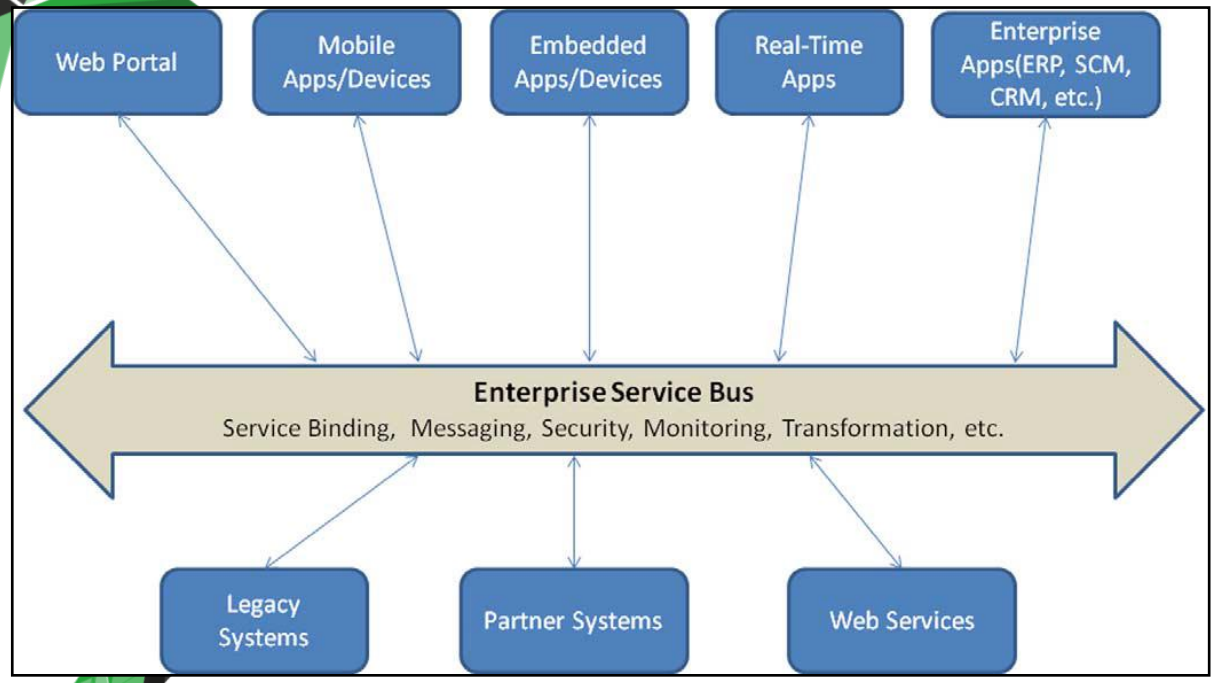


API Management

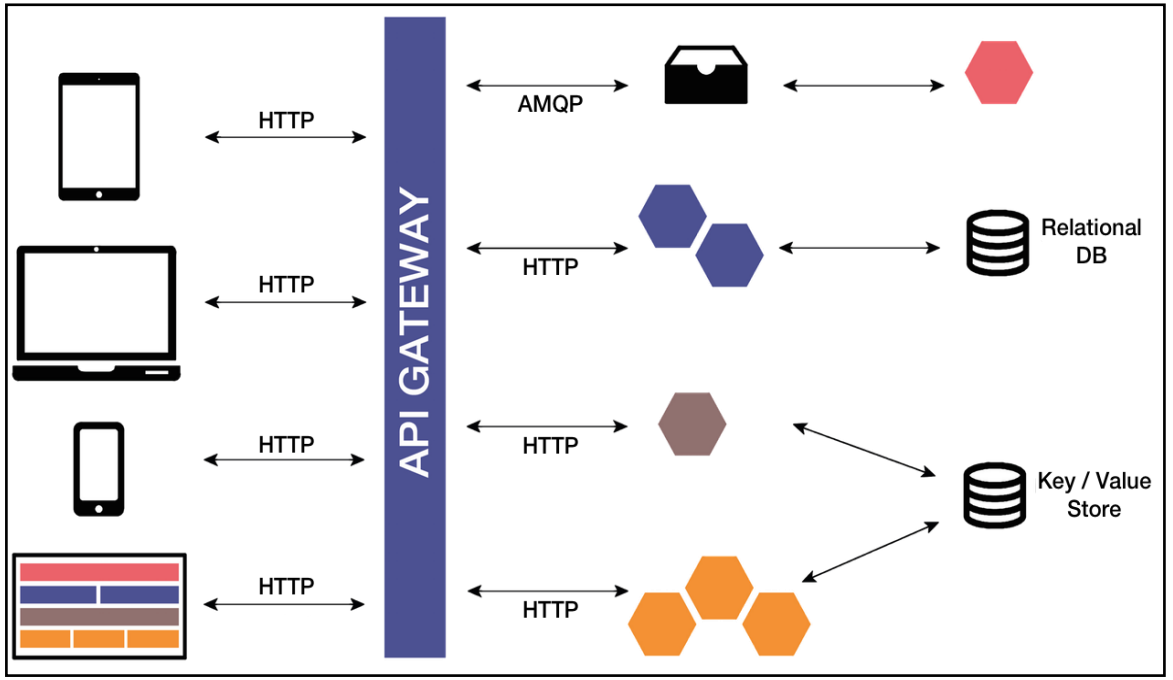
- Policy management
- Analytics and monitoring
- Developer documentation

UNIVERSITY OF TWENTE.

INTEGRATION TECHNIQUES
ENTERPRISE SERVICE BUS (ESB) X API GATEWAYS



Wu He and Li Da Xu, "Integration of distributed enterprise applications: a survey", IEEE Transactions on Industrial Informatics 10(1): 35-42, 2014.



Katuwal, K., Microservices: A Flexible Architecture for the Digital Age Version 1.0. American Journal of Computer Science and Engineering, 2016. (Microservices): p. 20-24.

✓ LG.2.3. *Explain* SOA principles, web service technologies (e.g., RESTful), the microservices style, and messaging integration techniques, e.g., Enterprise Service Bus (ESB) and API Gateways

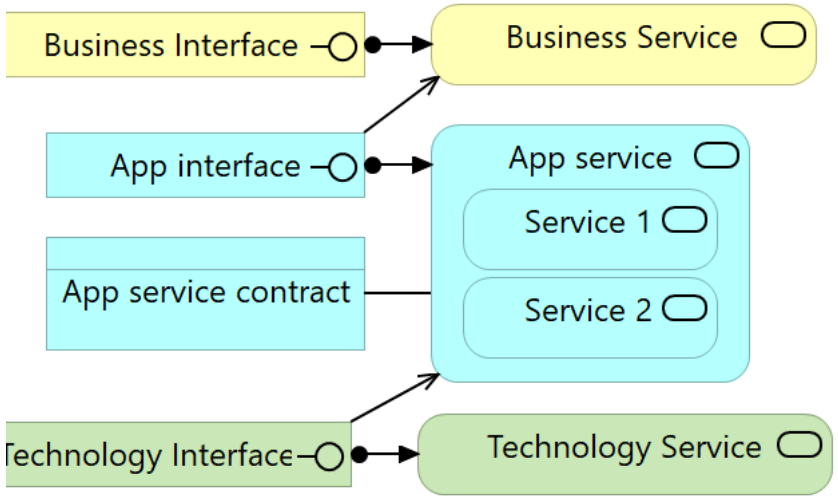
UNIVERSITY OF TWENTE.

SERVICE-ORIENTED ARCHITECTURE
EA: ARCHIMATE DEFINITIONS

Archimate 3.1

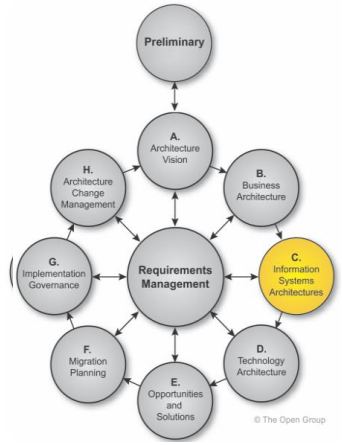
Behavior Elements: a service, represents an explicitly defined exposed behavior

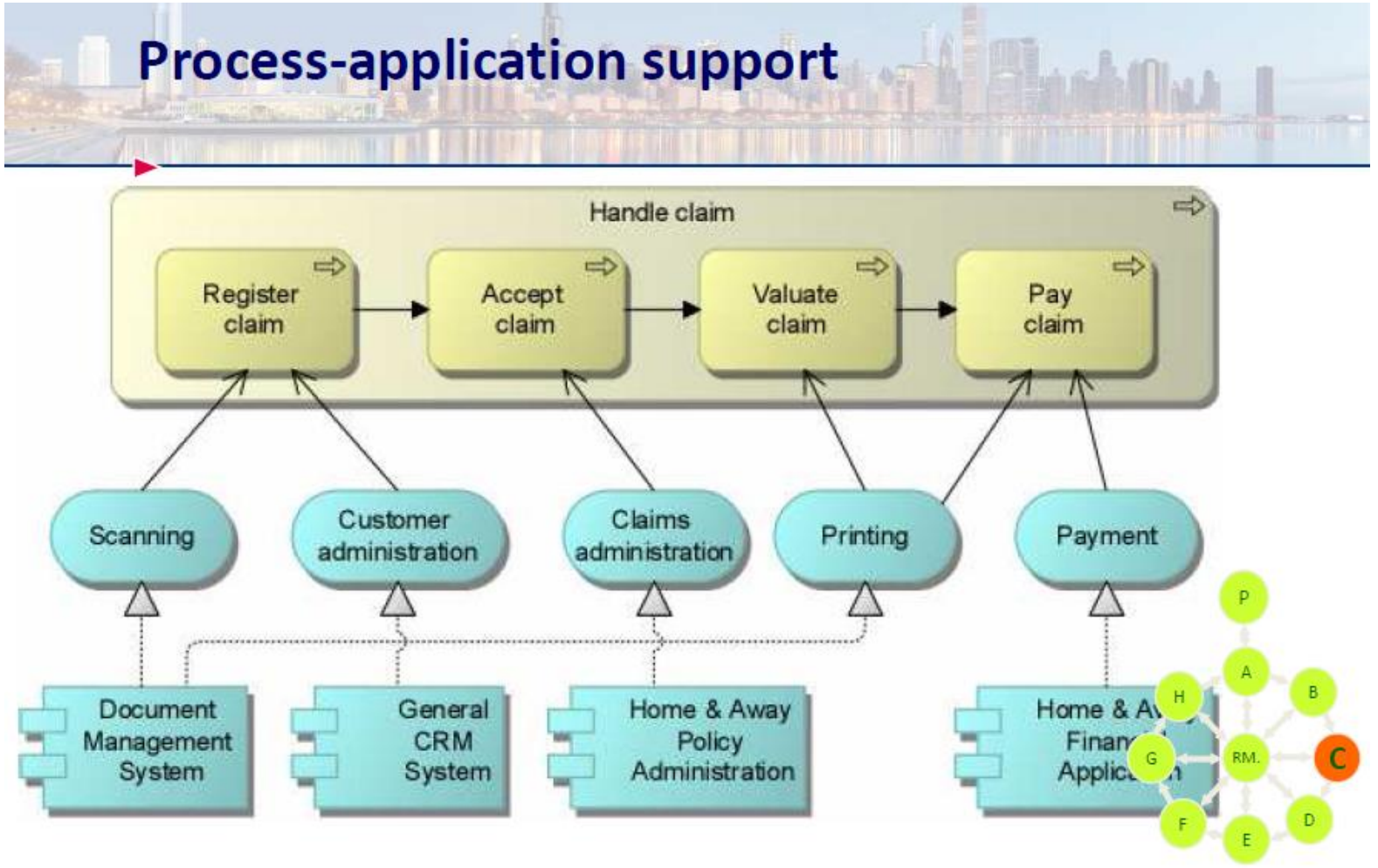
- Business Service: represents explicitly defined behavior that a business role, business actor, or business collaboration exposes to its environment
- Application Service: represents an explicitly defined exposed application behavior
- Technology Service: represents an explicitly defined exposed technology behavior



UNIVERSITY OF TWENTE.

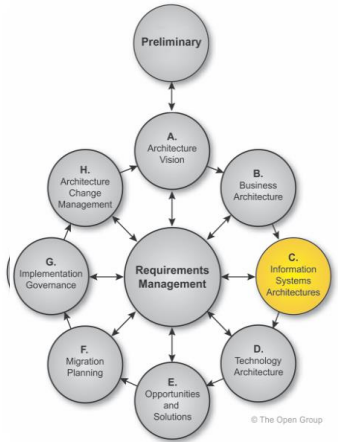
SERVICE-ORIENTED ARCHITECTURE
ARCHIMATE

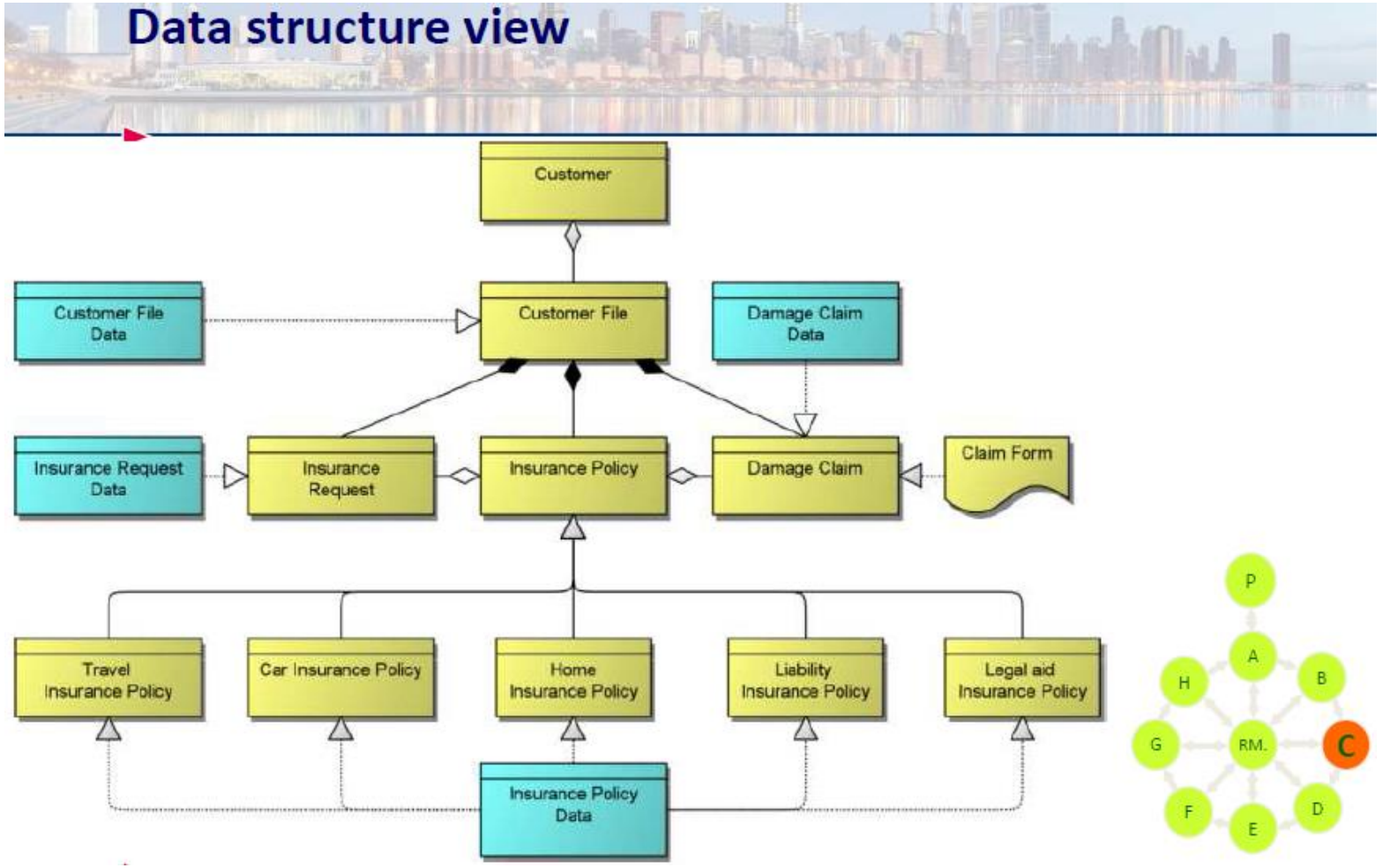




UNIVERSITY OF TWENTE.

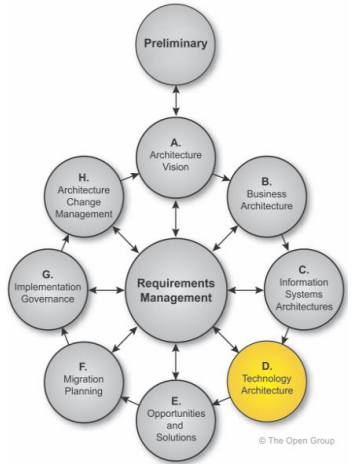
SERVICE-ORIENTED ARCHITECTURE
ARCHIMATE

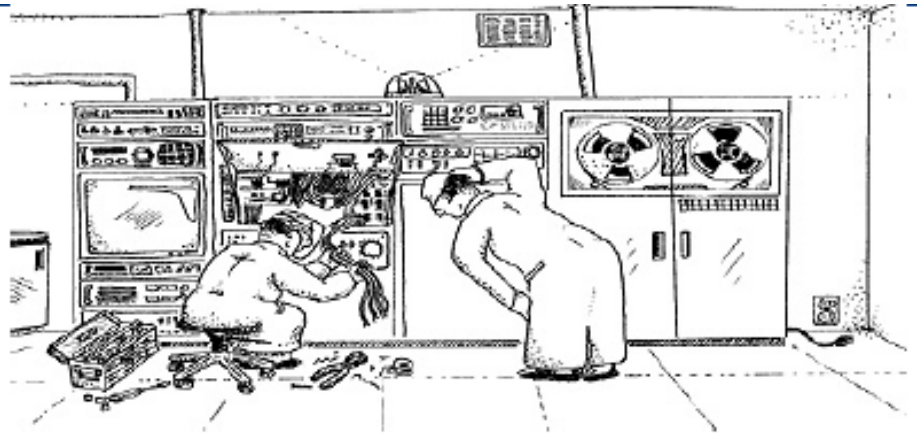




UNIVERSITY OF TWENTE.

ARCHIMATE: TECHNOLOGY (PHASE D)

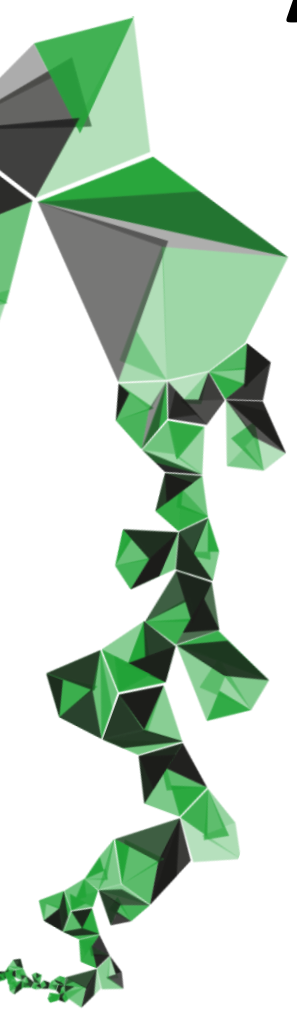


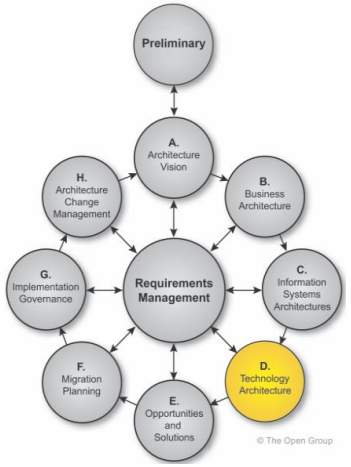


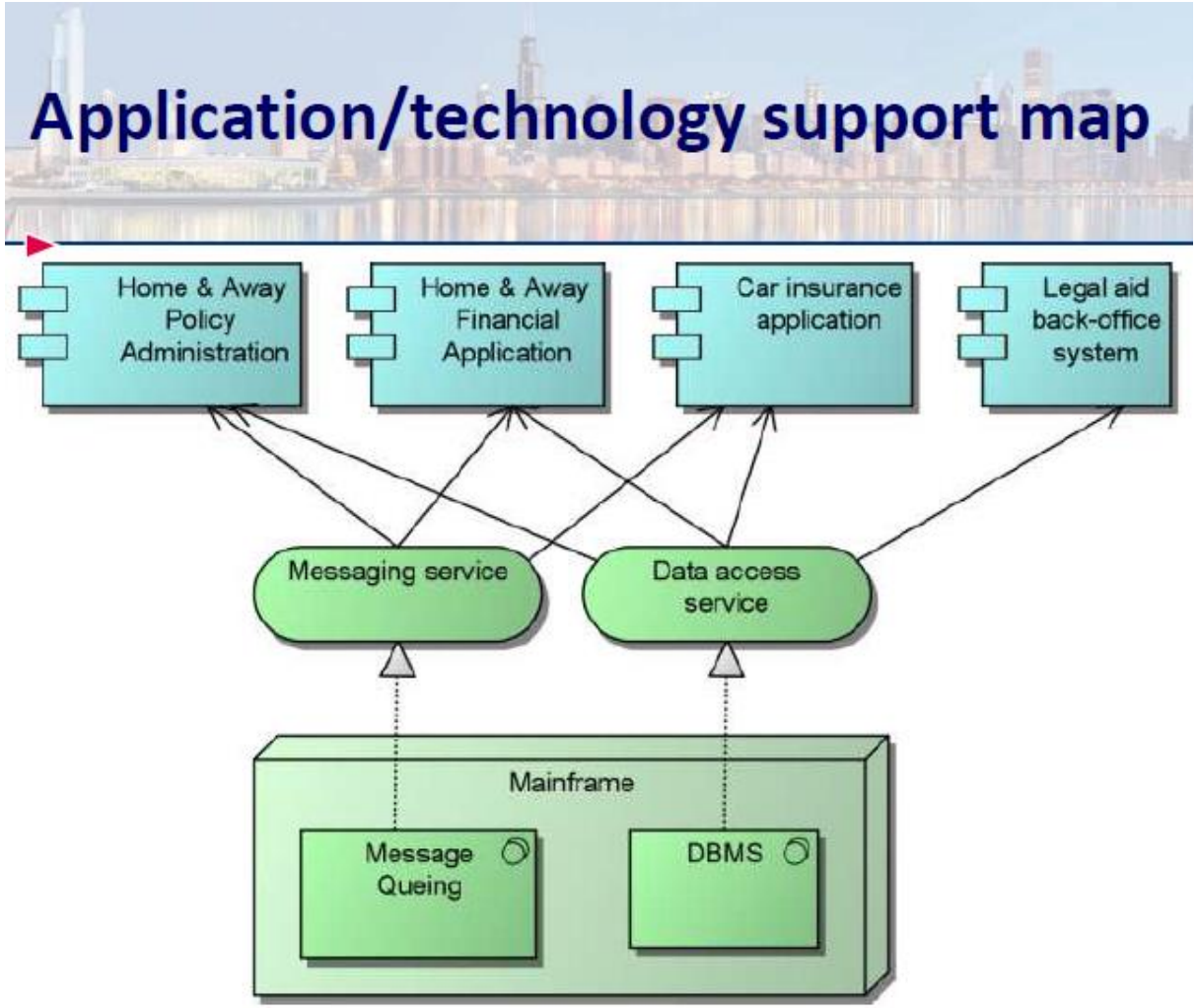
- Documenting the organisation of the IT systems:
  - Embodied in the hardware, software, and communications technology
  - Their relationship with each other and the environment
  - The principles governing its design and evolution

UNIVERSITY OF TWENTE.

ARCHIMATE: TECHNOLOGY (PHASE D)

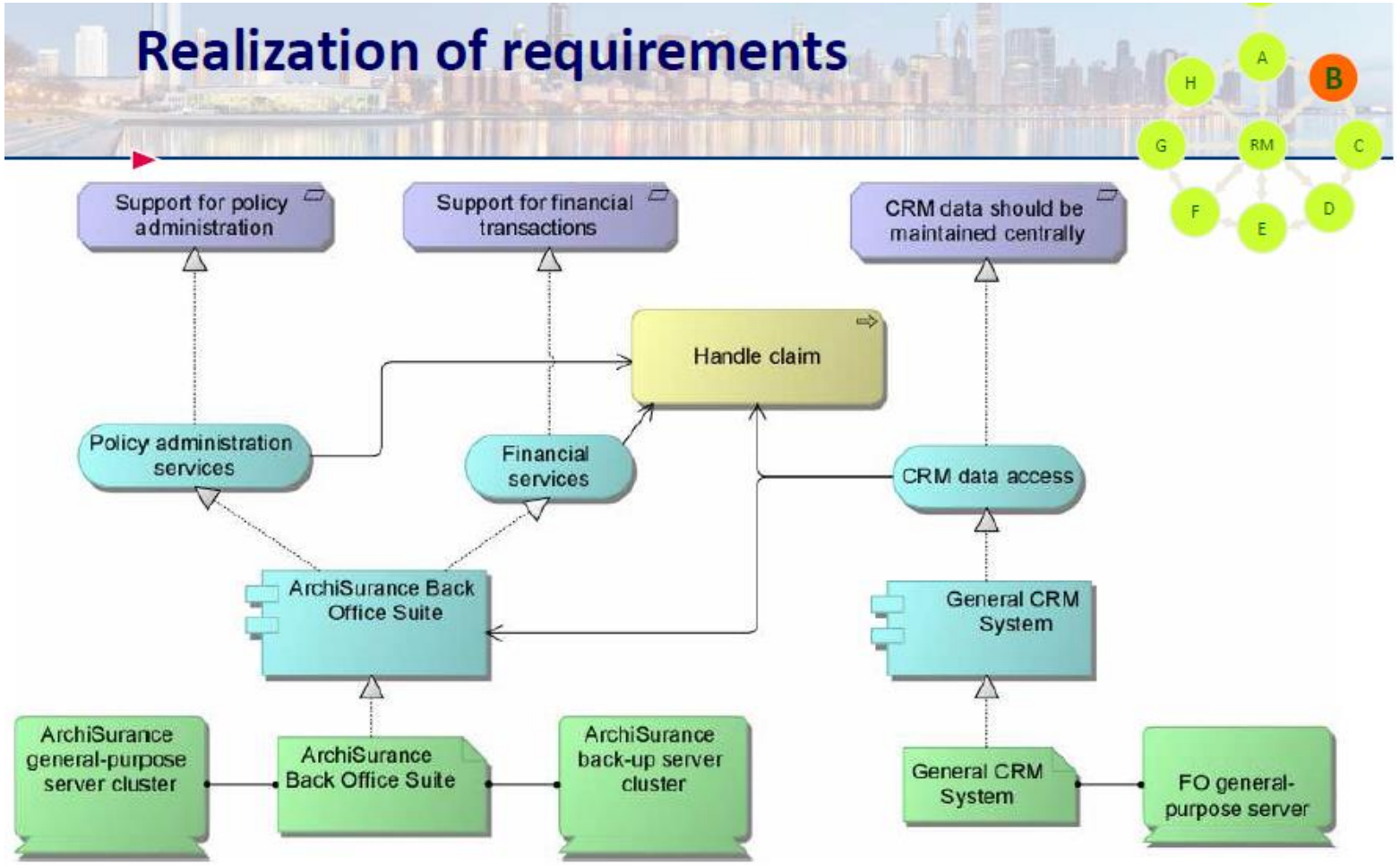


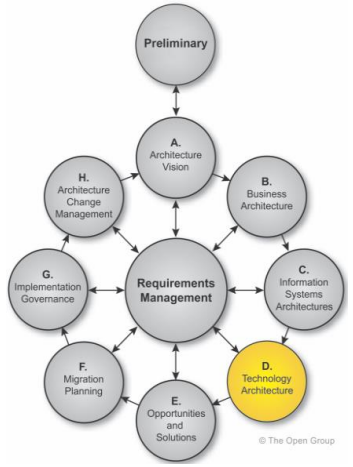




UNIVERSITY OF TWENTE.

ARCHIMATE: DESIGN SOA

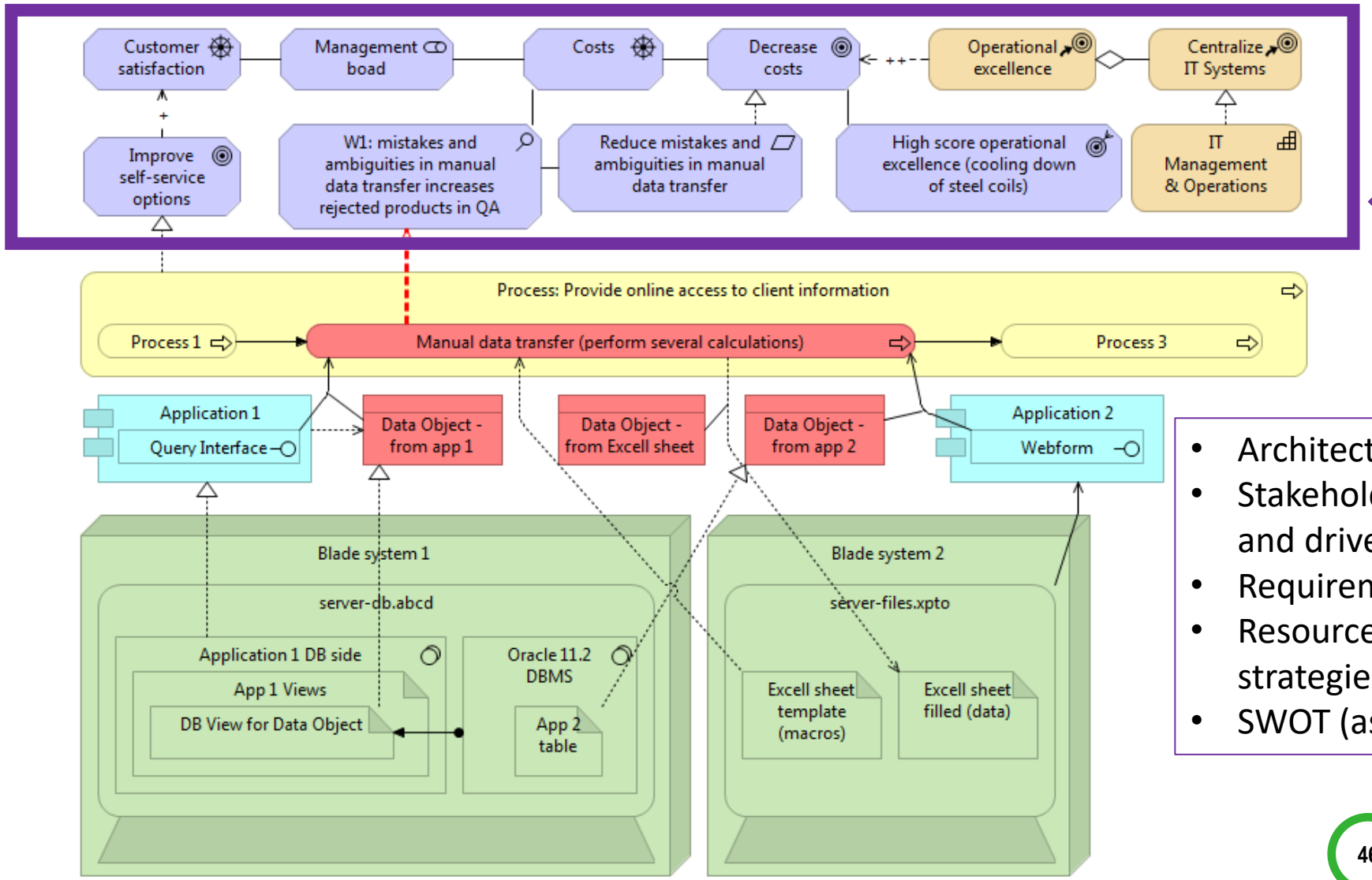




✓ LG.2.4. *Design* SOA-based Archimate application integration viewpoints that cover business processes served by web services that are provided by applications and their technology services; along with the associated requirements realization

UNIVERSITY OF TWENTE.

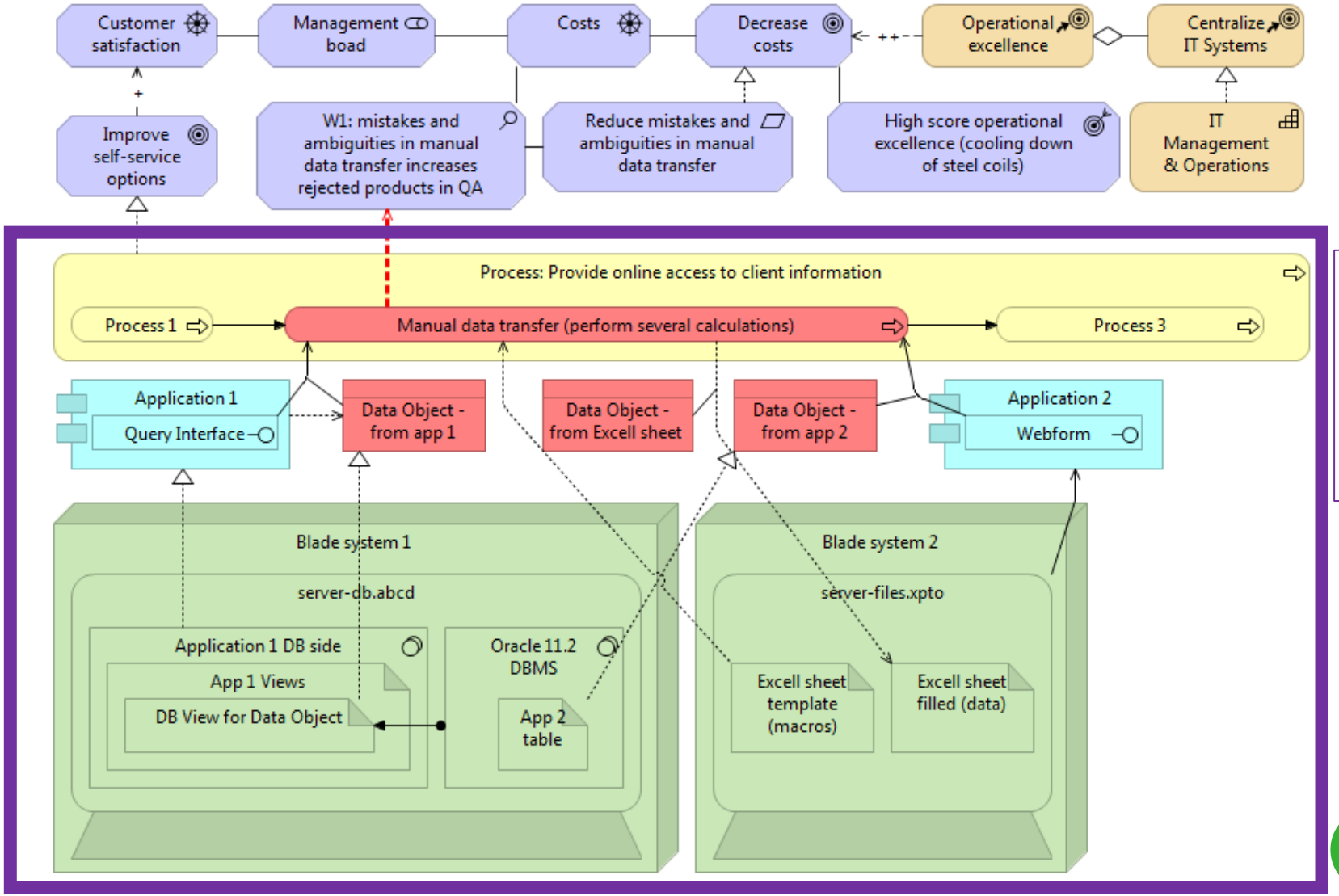
ARCHIMATE EXAMPLE



- Architecture principles
- Stakeholders, business goals and drivers
- Requirements and constraints
- Resources, capabilities, strategies and tactics
- SWOT (assessments)

UNIVERSITY OF TWENTE.

ARCHIMATE EXAMPLE



- Interoperability issues in the current architecture?
- SOA-based solution?

UNIVERSITY OF TWENTE.

TAKE-HOME MESSAGES

- Integrating diverse systems, e.g., ERP, SCM, CRM, may cause an "spaghetti architecture", which makes the IT landscape unmanageable
- There are 5 main interoperability aspects that are addressed (to some extent) by interoperability frameworks, which guide the application of middleware
- Service-Oriented Architecture (SOA) is a design principle, which may be implemented with different architectural styles like REST to design rules to compose a system
- REST principles: stateless client-server, resources identified via URIs, state representation, CRUD actions (HTTP)
- Microservice is an architectural style based on SOA to achieve performance (elasticity) and fast deployment goals
- The Archimate language can be used to design SOA, which is supported by TOGAF interoperability guidelines

PREPARATION NEXT LECTURE

Preparation

Reading main book

(Chapters 7-10)

W.	Activities
~~1~~	~~Lecture 1: Enterprise Information Systems~~
~~2~~	~~Project session 1: Organizational case analysis~~
~~3~~	~~Lecture 2: Role of middleware for interoperability in EA~~
4	**Workshop: non-functional requirements**
5	Workshop: Architectural patterns for integration design
6	Project session 2: Baseline architecture modelling
7	Workshop: Service-oriented applications and business processes
8	Project session 3: Target architecture modelling and migration
9	Exam preparation or guest lecture: reflect of EA for evolution
10	Project presentations; and Exam

Follow the steps in Canvas!