IS Architectures

SoftEng
http://softeng.polito.it

Architecture

- Components and their connections

IS architectures

- Three main options

- Functions and ‘islands’ (or silos)
  - 1960 →
- One data base
  - 1990 →
- Microservices
  - 2010 →

Conway's law

- The structure of an IT system mirrors the communication structure of the organization that produces it
  - [Melvin Conway, 1967]

Silos approach

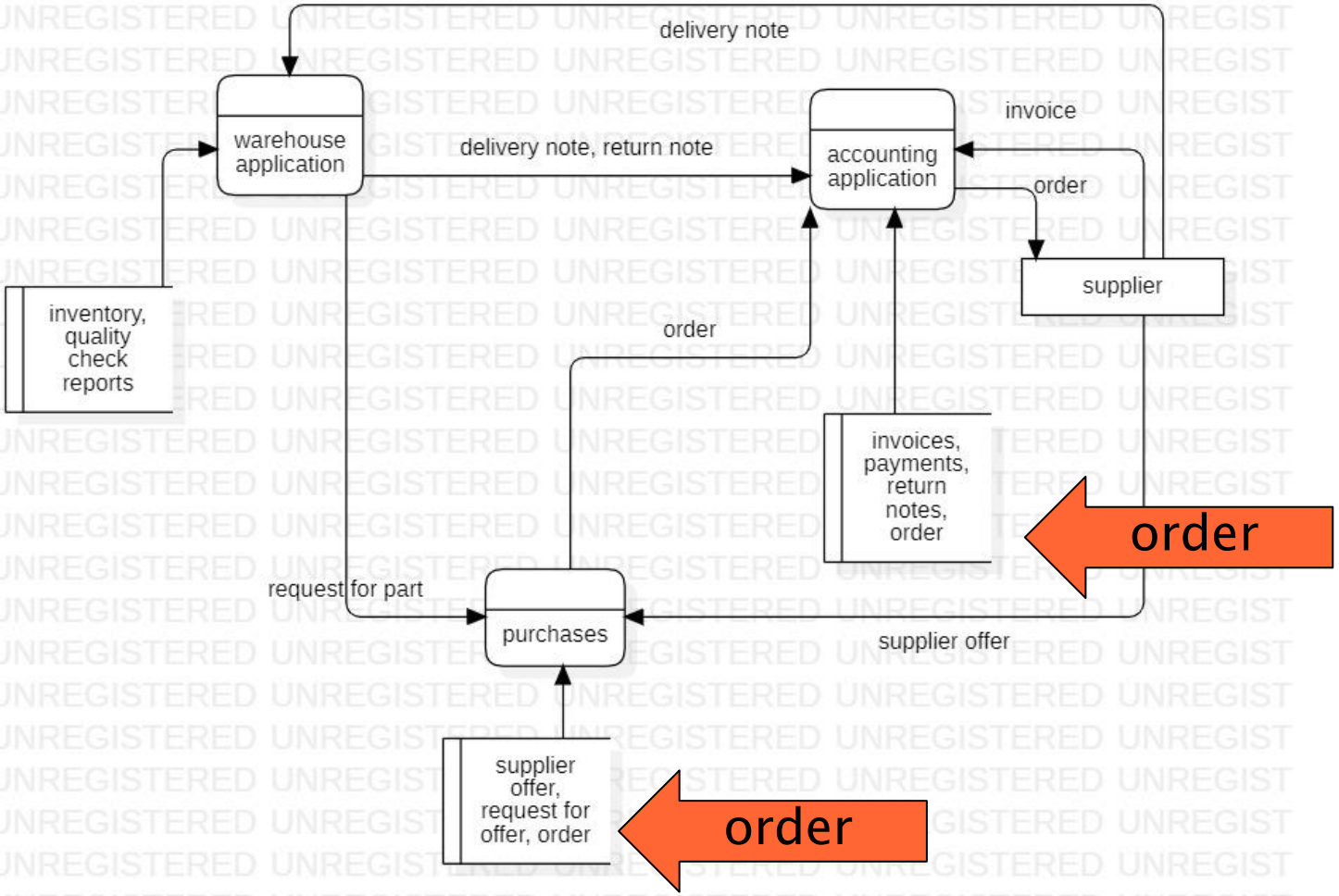
Silos approach

- Organization has several business functions
- ‘local’ IS are developed per business functions
- Data is replicated
  - Different semantics
  - Different values

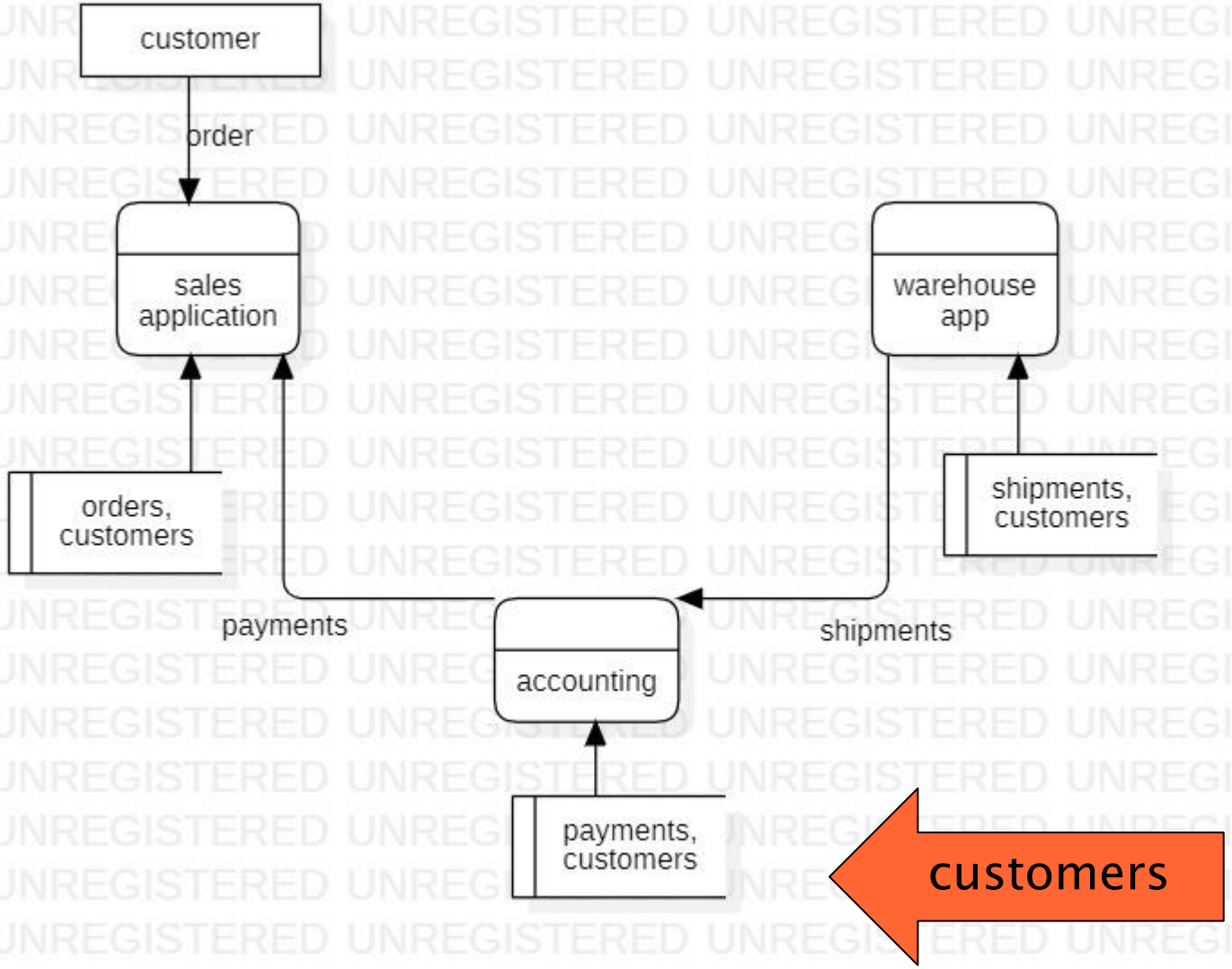
Ex

- Company has business functions
  - Warehouse / production
  - Purchase
  - Accounting / finance
- Each function develops, over time, different local IS
- Some data is local to BF, but some is common
  - Ex Order

Data replication: *legacy* islands



Data replication: *legacy* islands



Data replication

- Same data in several (legacy) systems
- Dedicated interfaces to synchronize (point to point)
  - Cost
  - Delays
  - Unfeasibility (of overnight synchronization)
  - Company must become system integrator

Data replication

- Each ‘data island’ typically matches a business function of the company
  - Accounting, warehouse, sales ..

- IS have a history, they are typically developed bottom up
- Unless a top down governance effort is made

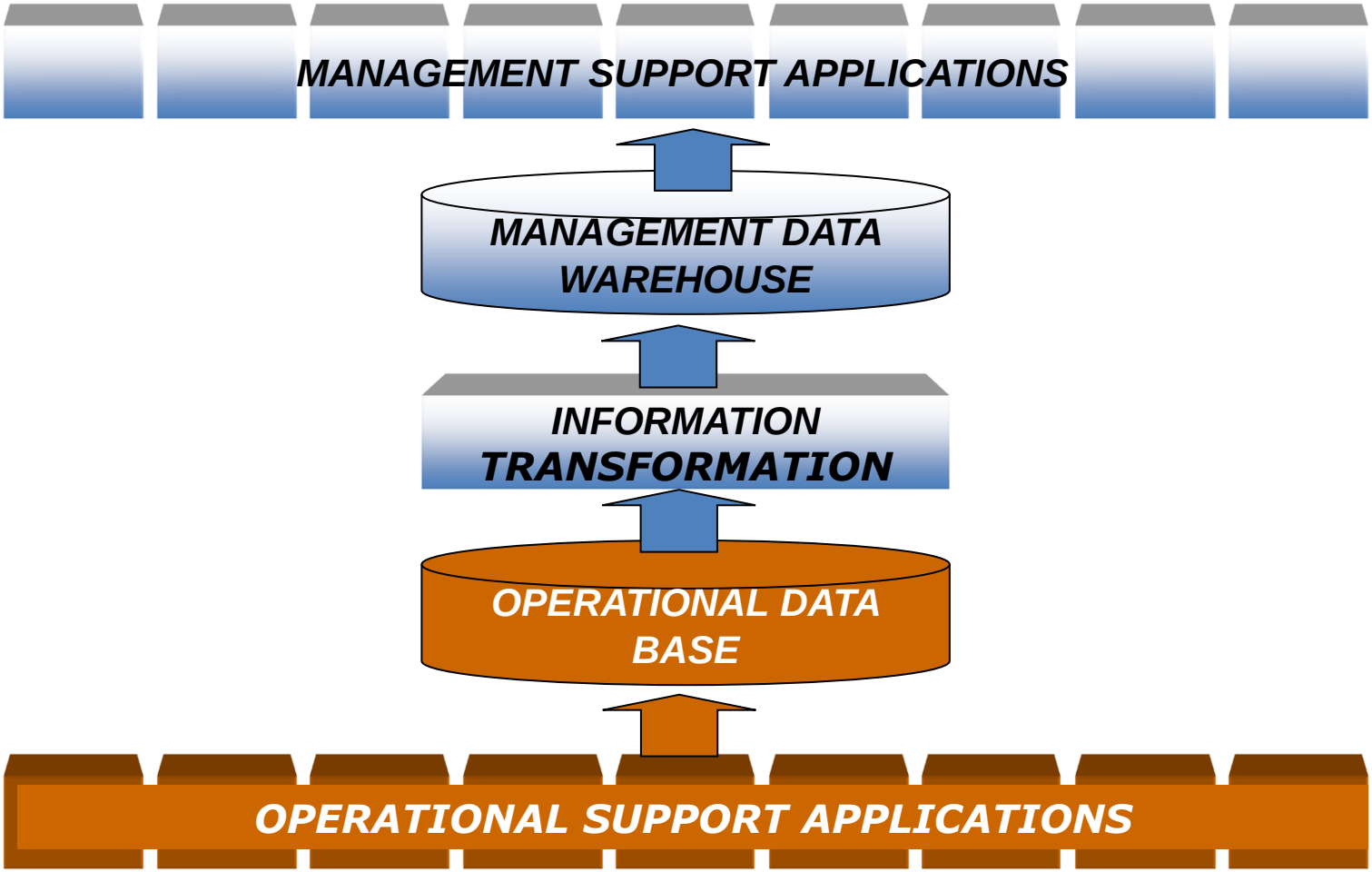
- Silos approach is typically not ‘designed’, but is the result of (ill governed) evolution over time
- Ex:
  - accounting IS developed at start of company
  - Warehouse IS added 10 years later
  - Purchase IS added 5 years later

Silos approach

- Issues of Silos approach are well known
- Silos approach often encountered because of resistance to change on companies, and costs + risks related to abandoning legacy IS

One database

ES: data sharing



ES: data sharing

- One DB or replicas with automatic synchronization
- One data model

- Horizontal integrity of data
  - All applications/modules share same data, with same data model
- Vertical integrity
  - From operation level to management level (aggregates of data)

One DB: issues

- Fits well if organization has one application only
  - Or a suite of applications developed by the same vendor
- In practice, organizations have many applications developed by different vendors
  - By default, each has an own DB

One DB: problems

- A certain data entity (ex table in RDB), is shared by many applications
- And becomes a coupling point
  - An application can modify the data entity without other application sknowing it, and can cause inconsistencies
- There is no ‘owner’ of a data entity capable of keeping the data entity consistent

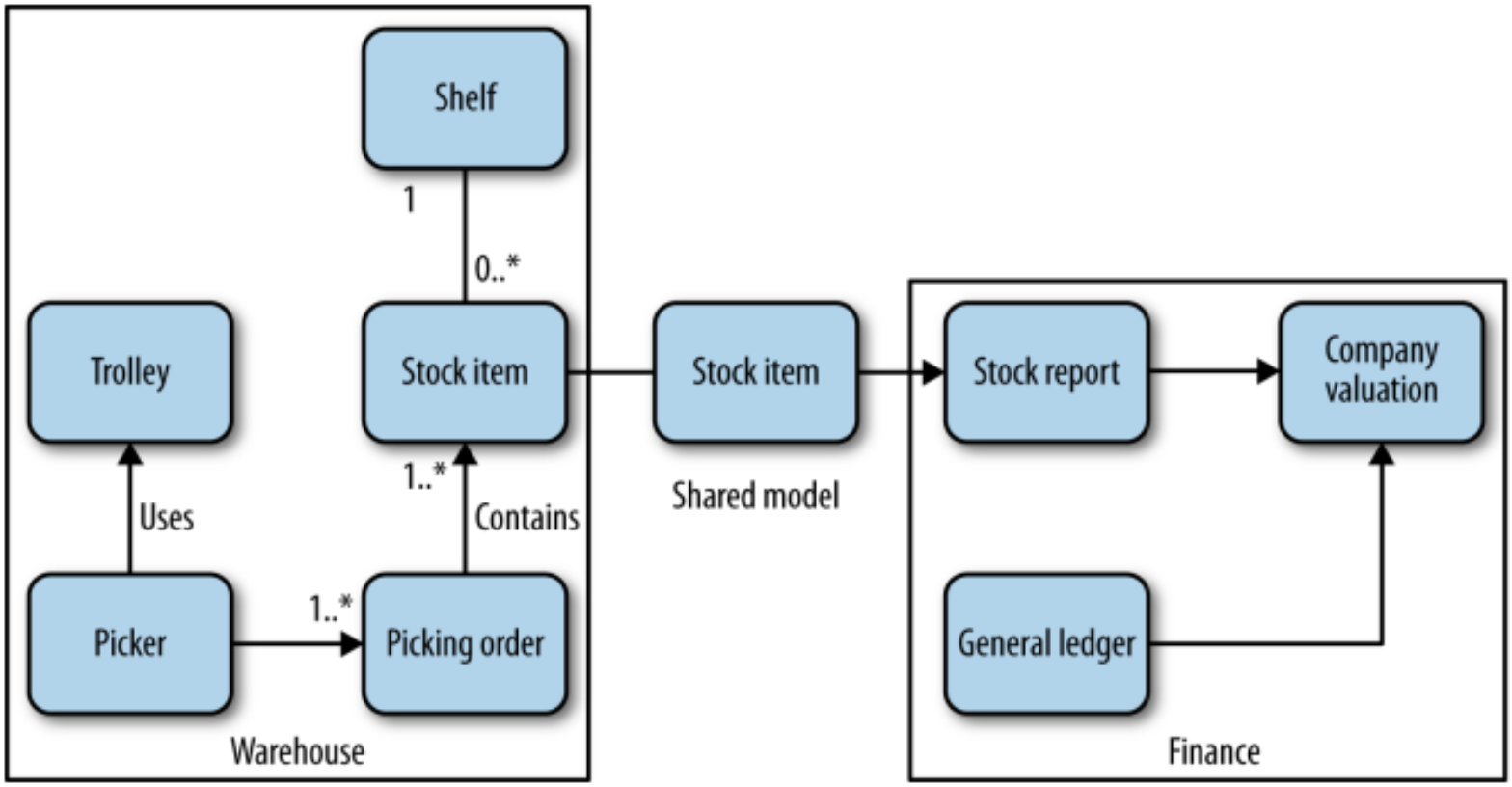
Microservices

Microservices

- Core idea
  - Modularity and encapsulation, as for object oriented languages
  - But raised to a higher level (many classes together) with an interface that is independent of a specific language or platform

Microservices

- Core idea:
  - Service is a group of classes with a defined interface
    - Consider a java package
  - Interface is independent of technology
    - Http REST (instead of Java interface as in java package)
  - Service runs and deploys independently


Shelf
1
0..*
Trolley
Stock item
Stock item
Stock report
Company valuation
1..*
Uses
Contains
Shared model
Picker
1..*
Picking order
General ledger
Warehouse
Finance

Protocol

- http
  - Get put post delete
- REST
  - What is exchanged is a textual representation of an object (XML or JSON), NOT the object itself
  - One service only has responsibility to manage the object and its state

Microservices

- Pro
  - Each service can use a different technology
    - Ex Oracle on one MS, mysql on other MS
    - Ex Linux on one MS, Windows on other MS
  - Each service can be developed and deployed by a different team, independently
  - Each service can focus on a specific business need (or group of functions)

Microservices

- Con
  - More complexity
  - More delays
  - Sometimes hard to split (data, functions)

Integration technologies

- Assuming an organization has several applications, how to integrate them?

API

- Application A exposes API
- Application B calls API

API

- Pros
  - The interaction follows a defined protocol
- Cons
  - Number of possible interactions is high (if number of applications is high)
  - Application may not offer API, or may not offer the needed function
  - APIs may use different technologies

Http REST APIs

- See microservices

- Wrap API of an application with http REST interface, repeat for all applications

Database

- Application A uses database DB_A
- Application B reads / writes on tables in DB_A

- Pros
- Cons
  - High coupling between A and B
  - Integrity of data may be lost (owner of data on common table is A or B?)

RPA Robotic Process Automation

- Application A and B have GUI interfaces
- RPA tool is introduced between A and B
  - RPA tool is capable of
    - Capturing data from GUI
    - Inserting data in GUI
    - Interacting with APIs, files system, databases

RPA

- Approach is very similar to capture and replay tools for GUI testing
  - Scrape a web page (or client interface) and extract data
  - Process data
  - Enter data in web page fields
- RPA tools do not need a real screen / keyboard / mouse, but work on virtual GUI
  - scalability

RPA

- Example –insurance
  - Email application A, customer sends a claim request via Email
  - ERP application B has GUI to enter manually data about the claim request (customer name and surname, motivation of claim etc)
  - RPA tool
    - Reads email, finds customer data in it
    - Enters data in GUI of application B

RPA

- Pros
  - Can be used on any application (especially the ones not exposing an API)
- Cons
  - Is subject to GUI analysis problems
  - Requires effort for evolution
    - GUIs of applications evolve faster than APIs

RPA vendors

- Automation anywhere
- Blue Prism
- UI Path
- Exilant