Architetture Software a Microservizi
Corso di Laurea Magistrale in Informatica

Introduction to microservice architectures

Davide Rossi
Dipartimento di Informatica – Scienze e Ingegneria
Università di Bologna

Rationale

- The two faces of microservices
- *Small* components of an enterprise SOA architecture
- Independent, modular components of a single, cohesive service-based application, each responsible for a specific business function or capability

© Classical Numismatic Group

What led us here?

- Enterpise SOA and *the new cloud*
- *Monolith refactoring*

Course aim

- Design and implementation of microservice architectures
- With strong emphasis on **design**
- Design what? Well, the usual suspects:
  - structural
  - behavioral

… aspects of the system and *its enviroment*

Service-Oriented Architectures

- One of the hottest buzzwords of the early 21st century
- A concept impacting both IT and management
- Several (broad) definitions: almost every distributed system can fit one of these

The SOA mandate

- All teams will henceforth expose their data and functionality through service interfaces.
- Teams must communicate with each other through these interfaces.
- There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team’s data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.

The SOA mandate

- It doesn’t matter what technology they use. HTTP, CORBA, Pub-Sub, custom protocols — doesn’t matter. [...]
- All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
- Anyone who doesn’t do this will be fired.

The SOA mandate

- It doesn’t matter what technology they use. HTTP, CORBA, Pub-Sub, custom protocols — doesn’t matter. [...]
- All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
- Anyone who doesn’t do this will be fired.

Jeff Bezos

The IT drivers

- **Inter-organization** issues: network standards and protocols
- **Intra-organization** issues: system integration

The economic drivers

- IT alignment
- Business agility
- Cross-industry collaboration and standardization
- Facilitating mergers and acquisitions

Services as a component model

A powerful **architectural concept** that promotes modularity, reusability, and flexibility in software systems.

It addresses: modularity and encapsulation, reusability, interoperability, scalability and flexibility, ...

Software architecture

Software architecture encompasses the following:

- The 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 collaboration among those elements
- The composition of these elements into progressively larger subsystems; the architectural style that guides this organization, these elements, and their interfaces, their collaboration and their composition

Software architecture is concerned with not only structure and behavior, but also also usage, functionality, performance, resilience, reuse, comprehensibility, economic and technology constraints and trade-offs, and aesthetic concerns.

Philippe Kruchten – The Rational Unified Process – An Introduction

Grady Booch’s take

All architecture is design but not all design is architecture. Architecture represents the significant design decisions that shape a system, where significant is measured by **cost of change**.

Needs and capabilities

SOA provides a mechanism for matching **needs** of service consumers with **capabilities** provided by service providers.

The granularity of needs and capabilities vary from fundamental to complex, and any given need may require the combining of numerous capabilities while any single capability may address more than one need.

Services characterization

In SOA, services usually are:

- Loosely coupled
- Reusable
- Autonomous
- Stateless : OGNI INTERAZIONE VA GESTITA COME INTER. INDIPEND. (ANCHE SE STESSO CLIENT)
- Discoverable
- Composable

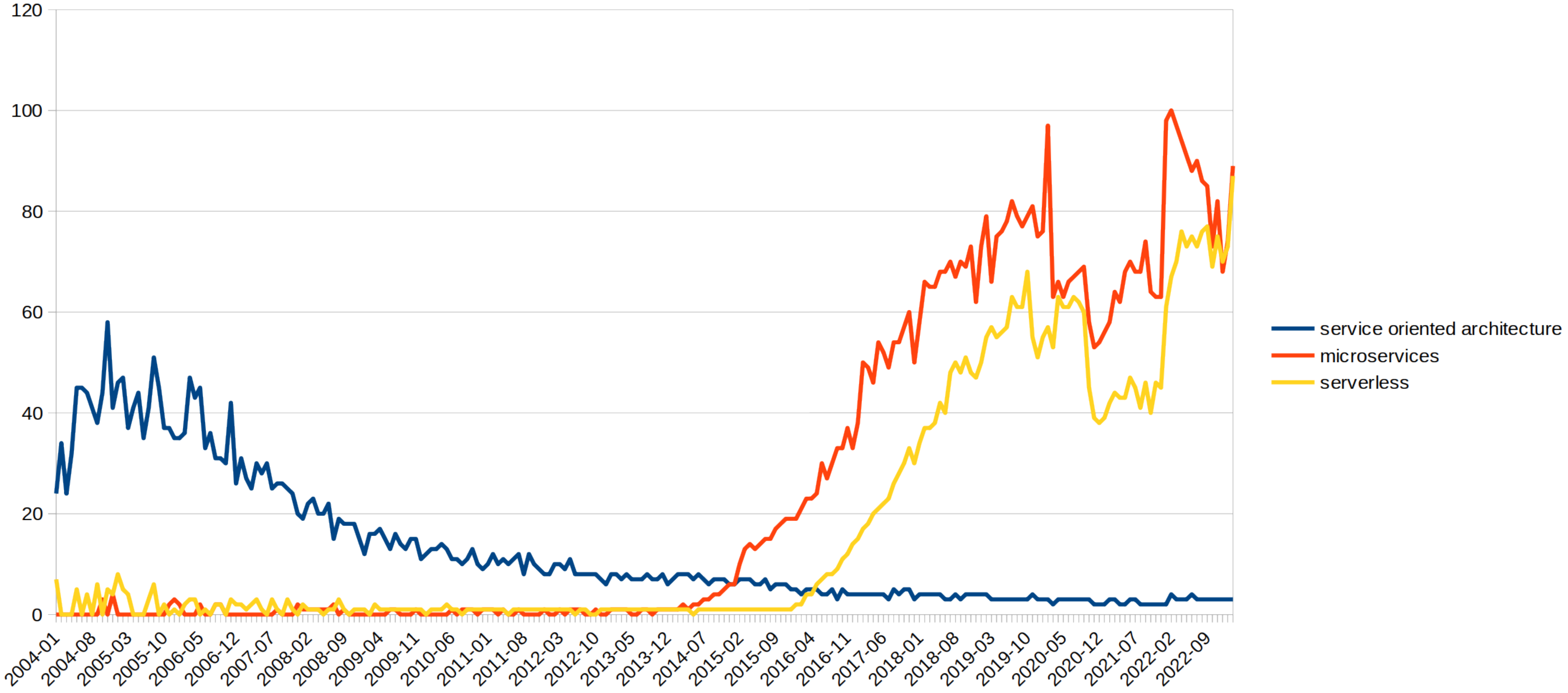
→ COMPOSIZIONE DINAMICA

- Based on standards (contracts, protocols, ...)

Here goes (again) the history class

- First generation (enterprise) SOA
- Microservices
- Serverless computing

The new wave



[Google trends]

First generation SOA

Main problems to solve:

- Inter-organization: network standards and protocols
- Intra-organization: system integration

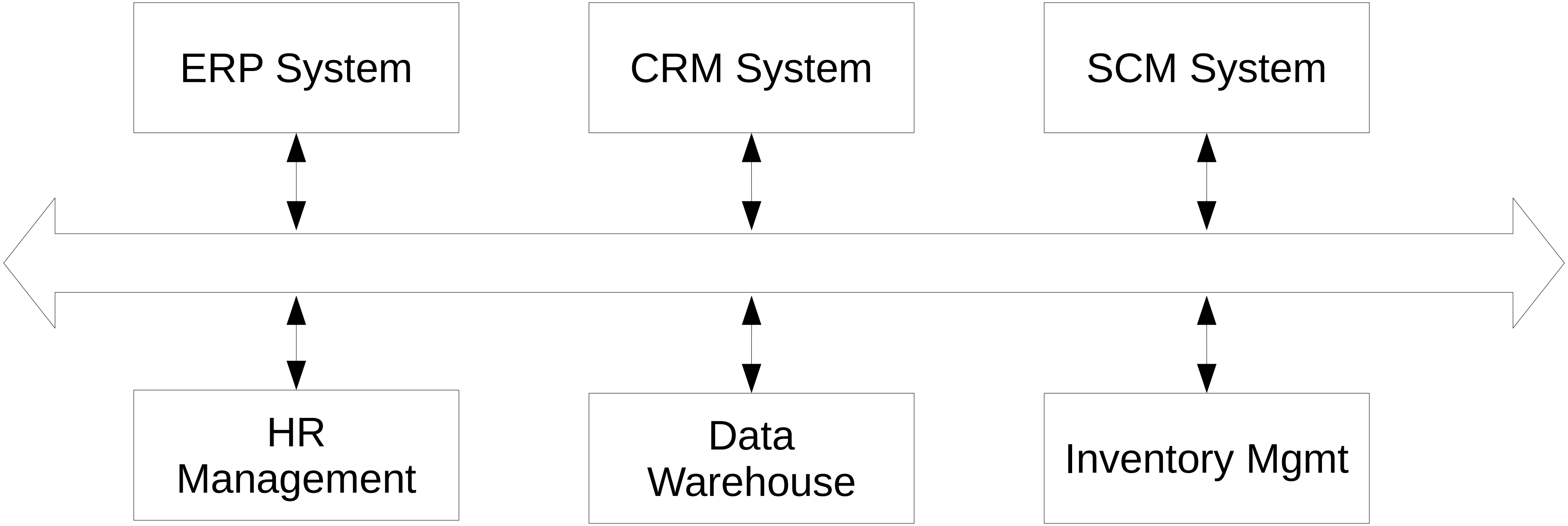
Typical solutions:

- Top-down, holistic modeling
- Clear governance - Governance is about policies. Policies are pervasive and can touch almost every aspect of software development and operations.
- Enterprise technology stack

Enterprise SOA

- Focus on consistency and correctness
- Top-down
- Single tech stack
- SOAP
- Clear governance
- Enterprise Service Bus (ESB)

Enterprise Service Bus



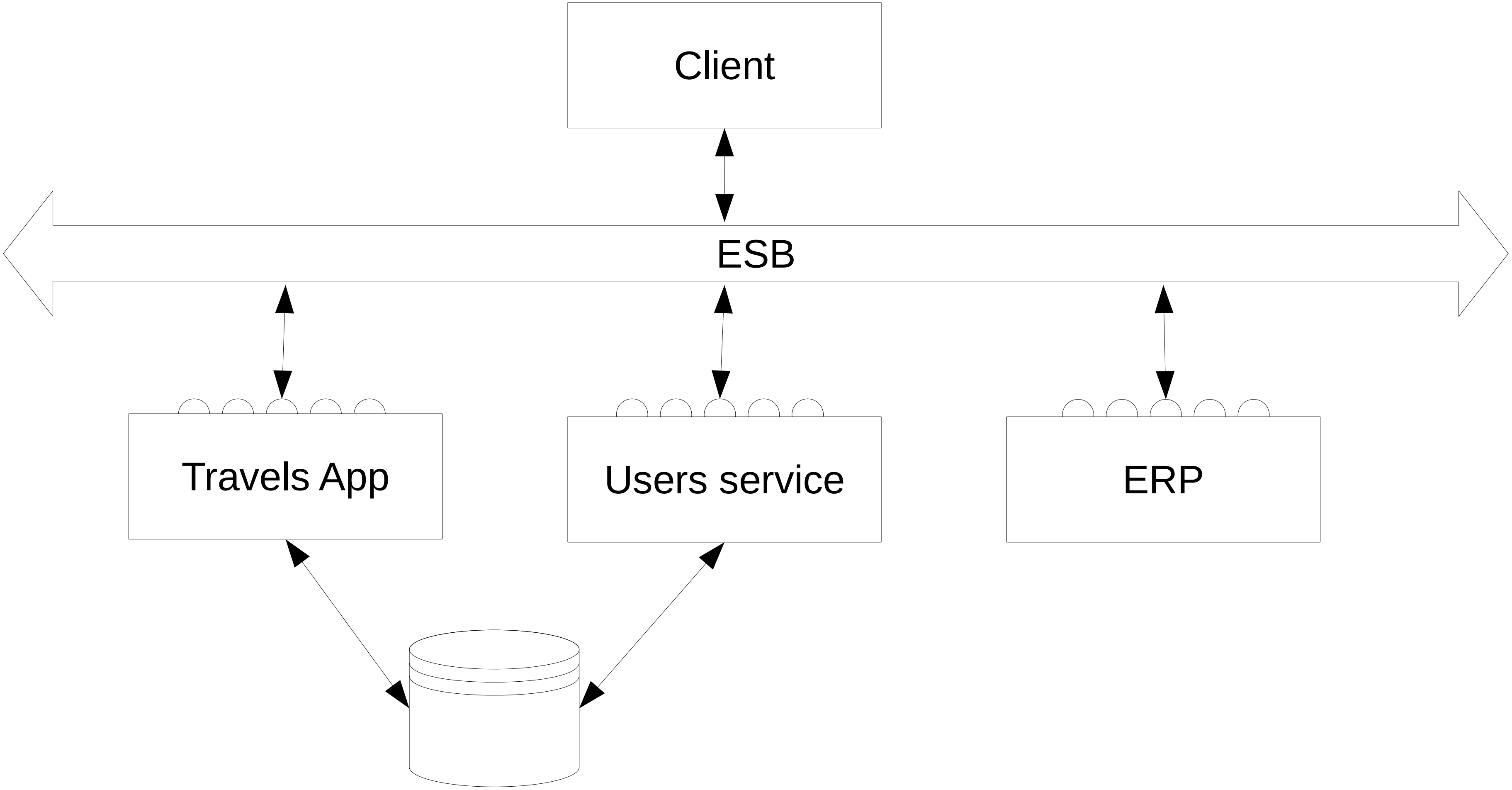
What went wrong?

- Business logic pushed into the ESB
- ESB just re-branding of EAI broker
- Services not really autonomous
- Centralized governance models that actively inhibited change

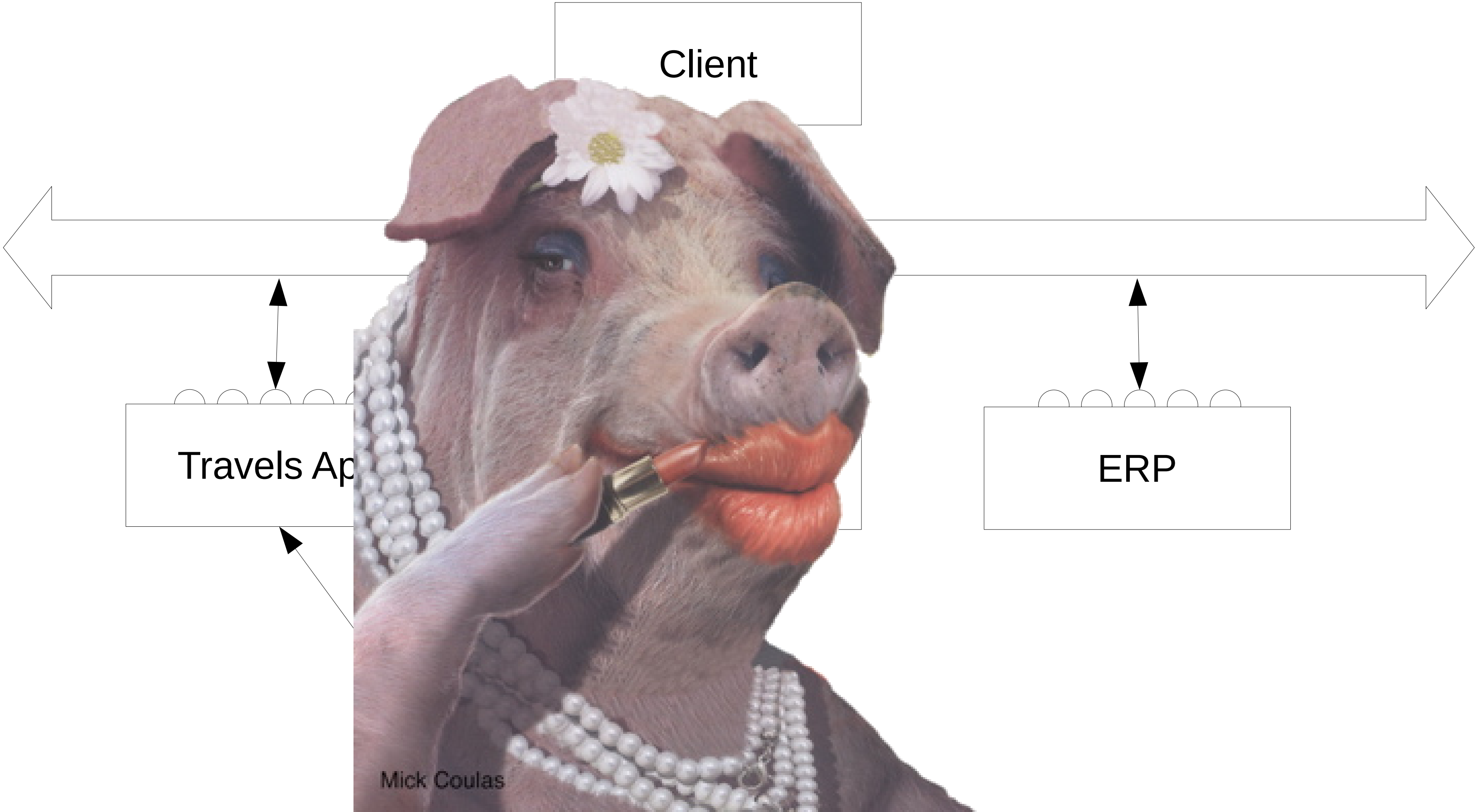
Conway’s Law

Organizations which design systems (in the broad sense used here) are constrained to produce designs which are copies of the communication structures of these organizations — Melvin E. Conway

Is this a SOA?



Is this a SOA?



The enterprise software zoo

Enterprise applications include:

Manufacturing: Engineering, bills of material, scheduling, capacity, workflow management, quality control, cost management, manufacturing process, manufacturing projects, manufacturing flow;

Supply chain management: Order to cash, inventory, order entry, purchasing, product configurator, supply chain planning, supplier scheduling, inspection of goods, claim processing, commission calculation;

Financials: General ledger, cash management, accounts payable, accounts receivable, fixed assets;

Projects: Costing, billing, time and expense, activity management;

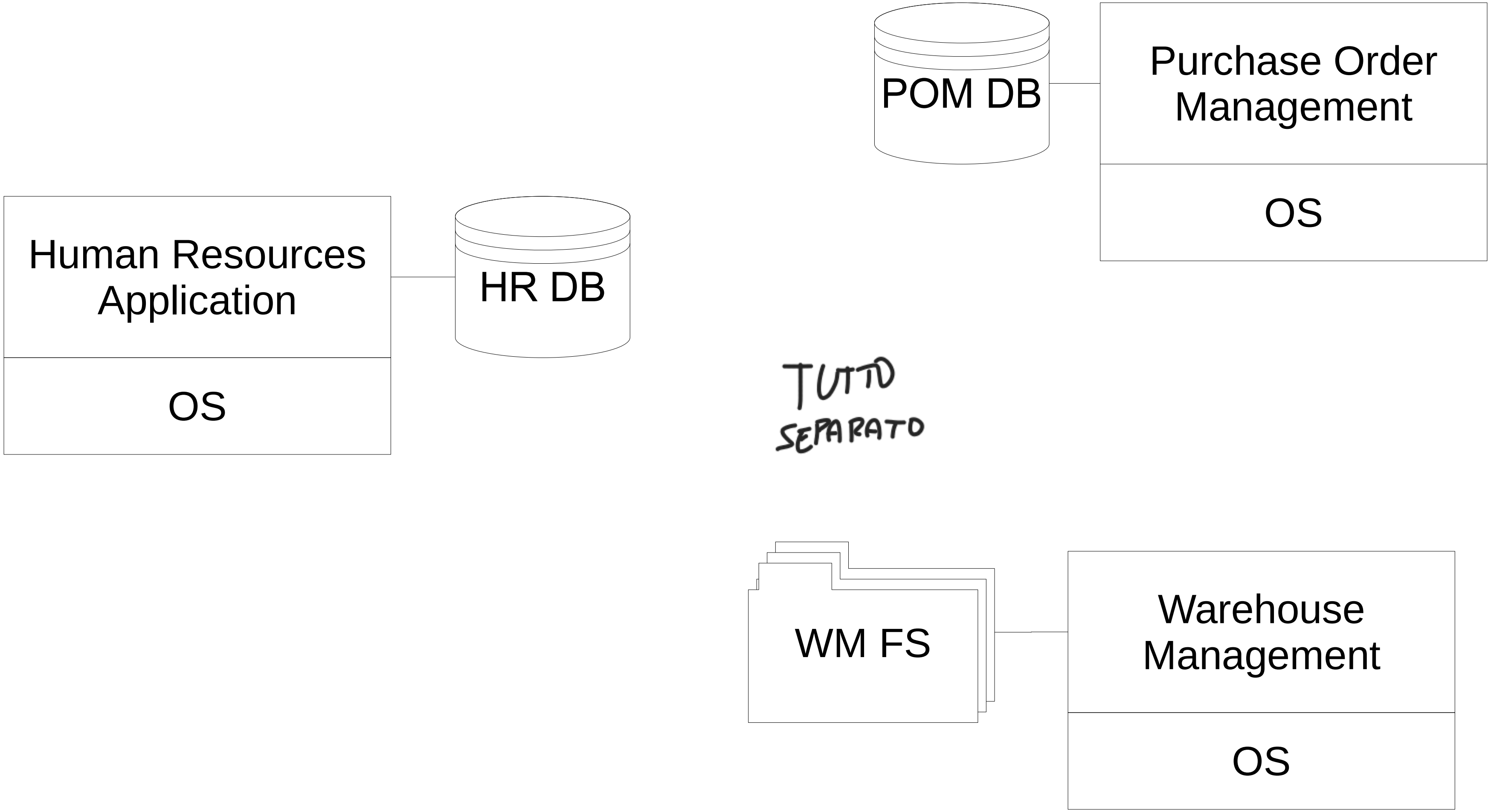
Human resources: Human resources, payroll, training, time and attendance, rostering, benefits;

Customer relationship management: Sales and marketing, commissions, service, customer contact and call center support;

Data warehouse and various self-service interfaces for customers, suppliers, and employees.

[Wikipedia]

Early architectures



Purchase Order Management

OS

POM DB

Human Resources Application

OS

HR DB

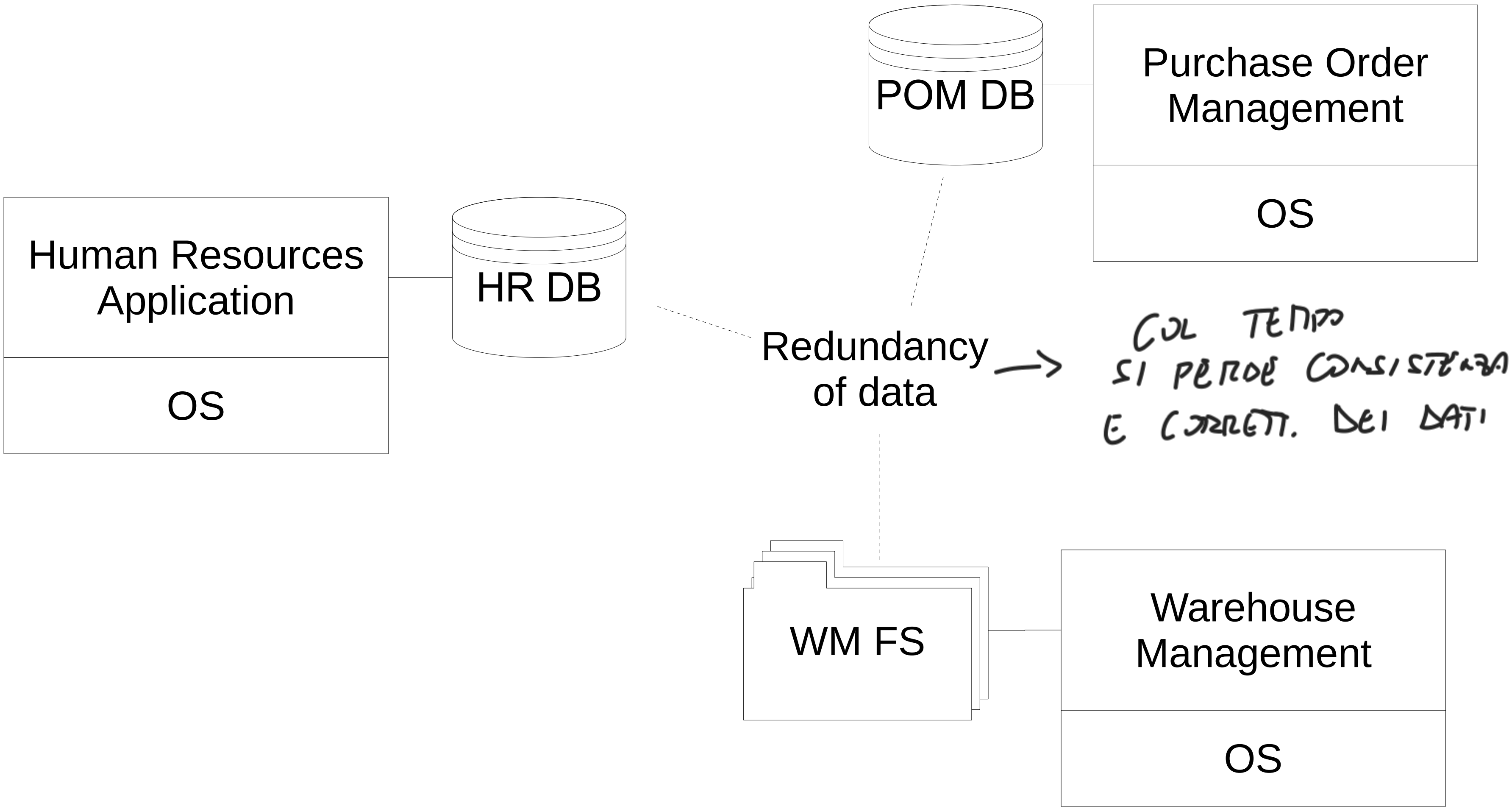
TUTTO SEPARATO

WM FS

Warehouse Management

OS

Early architectures



POM DB

Purchase Order Management

OS

Human Resources Application

OS

HR DB

Redundancy of data →

COL TEMPO SI PERDE CONSISTENZA E CORRETT. DEI DATI

WM FS

Warehouse Management

OS

The ERP

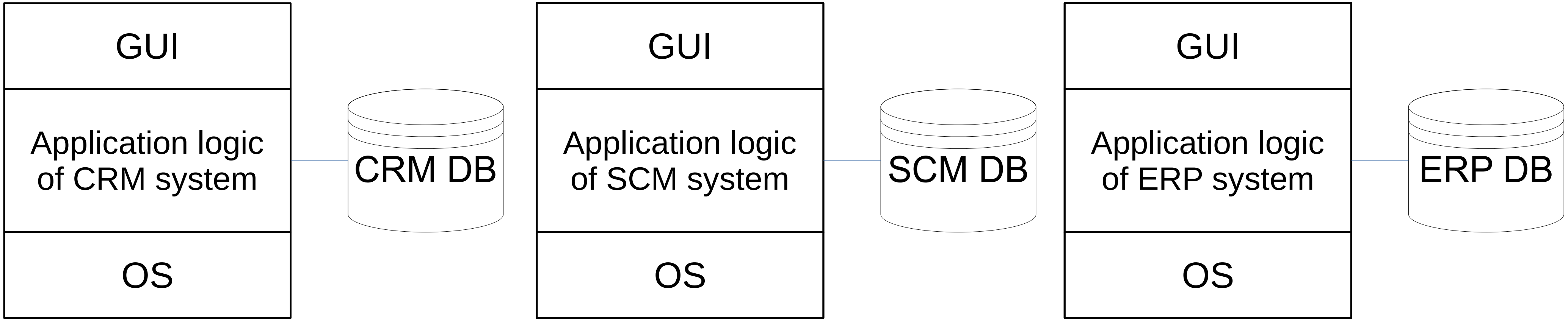
An Enterprise Resource Planning Systems is an enterprise-wide information system designed to coordinate all the resources, information, and activities needed to complete business processes such as order fulfillment or billing.

ERPs are commonly based on an integrated and consistent database.

Back to the future

At around year 2000, stand-alone Supply Chain Management Systems and Customer Relationship Management Systems start to be deployed along existing ERP. These new systems were not integrated with the ERP bringing back the old integration problems.

Siloed enterprise applications


GUI
Application logic of CRM system
OS
CRM DB
GUI
Application logic of SCM system
OS
SCM DB
GUI
Application logic of ERP system
OS
ERP DB

Enterprise Application Integration

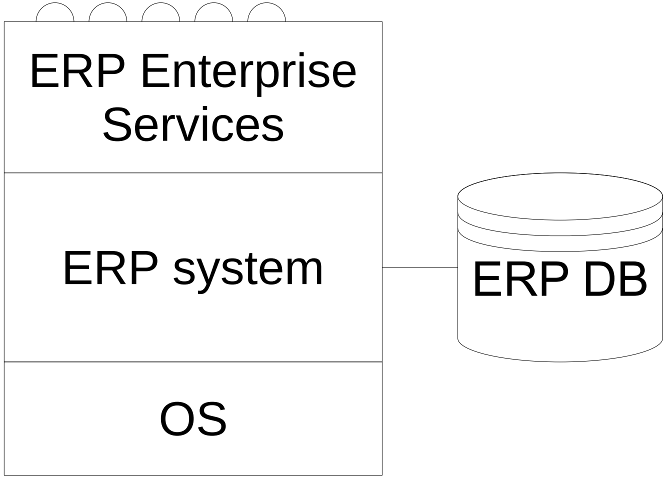
Enterprise application integration (EAI) is the process of linking siloed applications in order to simplify and automate business processes.

The main challenge is information exchange: data transfer and data transformation.

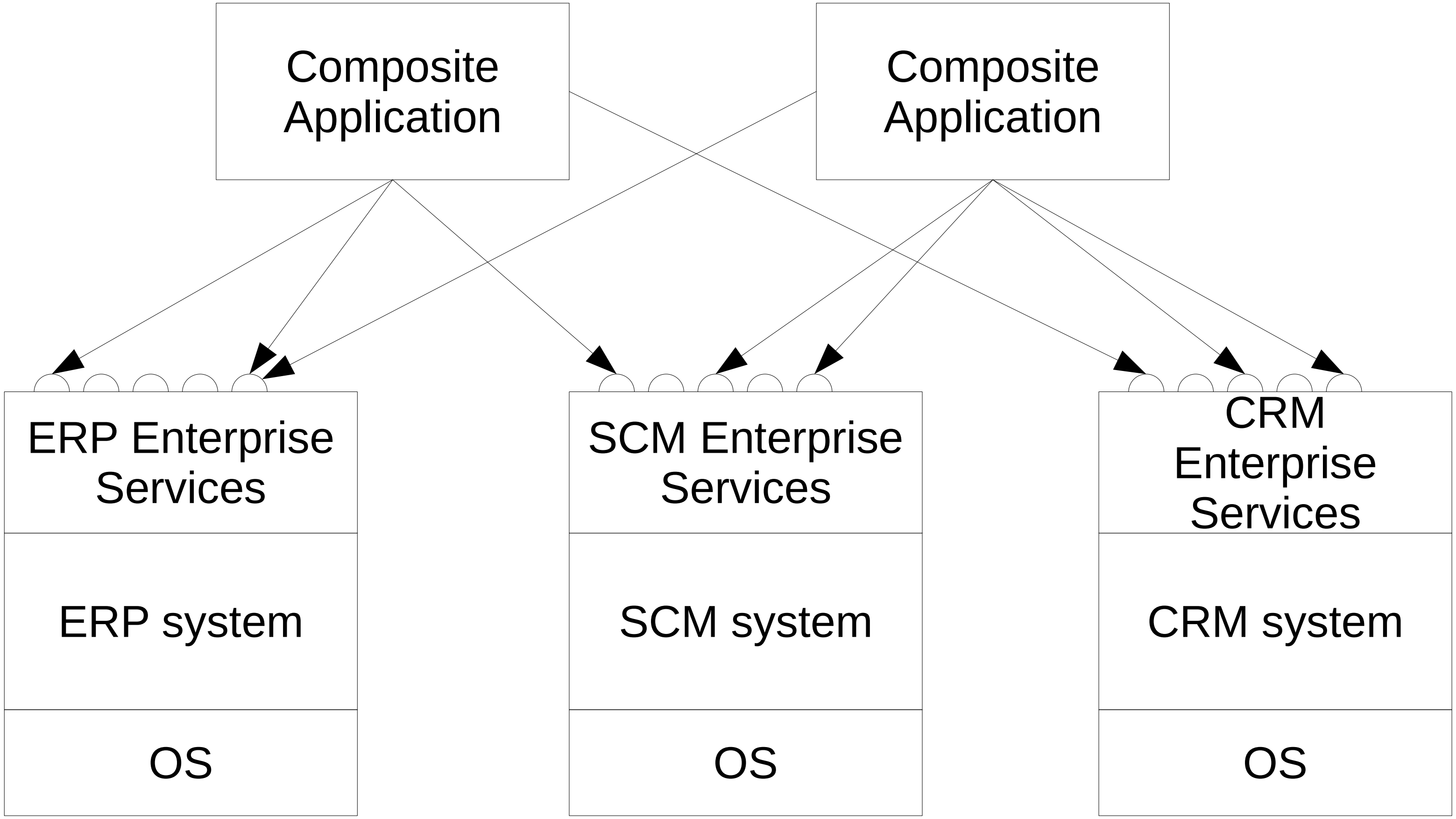
Two main integration patterns: mediation and federation.

Two main topologies: hub-and-spoke and bus.

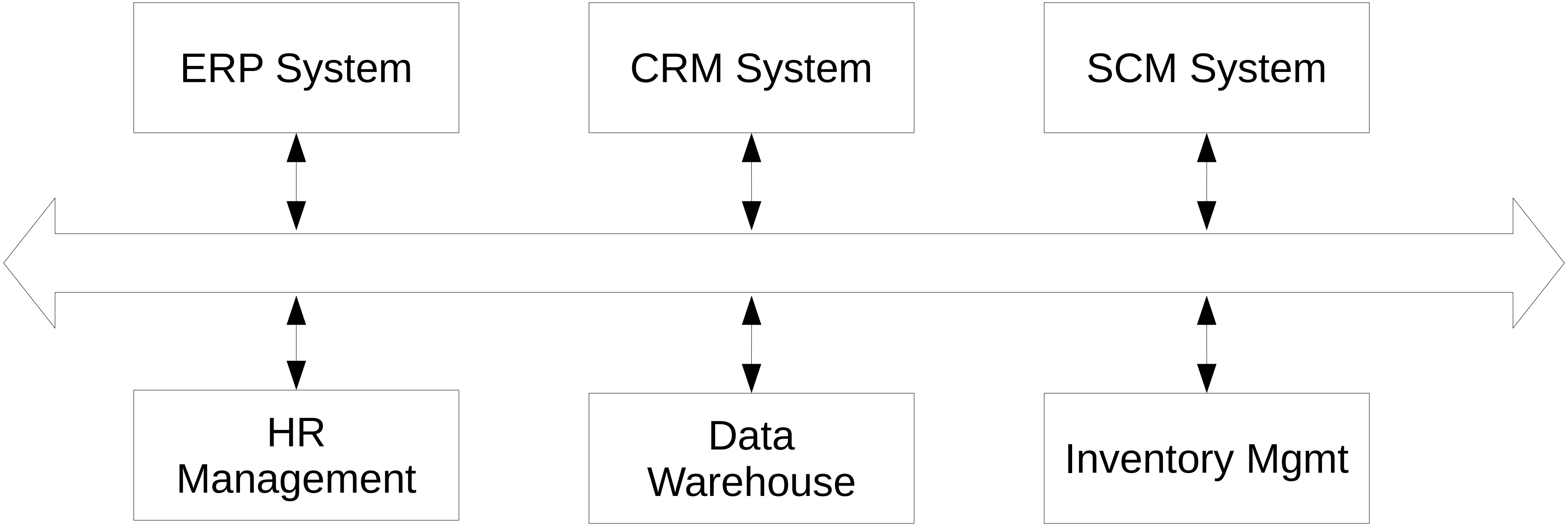
Service-enabled application



Composite Service-based Application



Enterprise Service Bus



Enterprise Service Bus

- Alternative to point-to-point integration
- Message routing
- Content adaptation
- Mediation (deployment and versioning)
- Transaction management, security, monitoring, error handling, …

The big picture

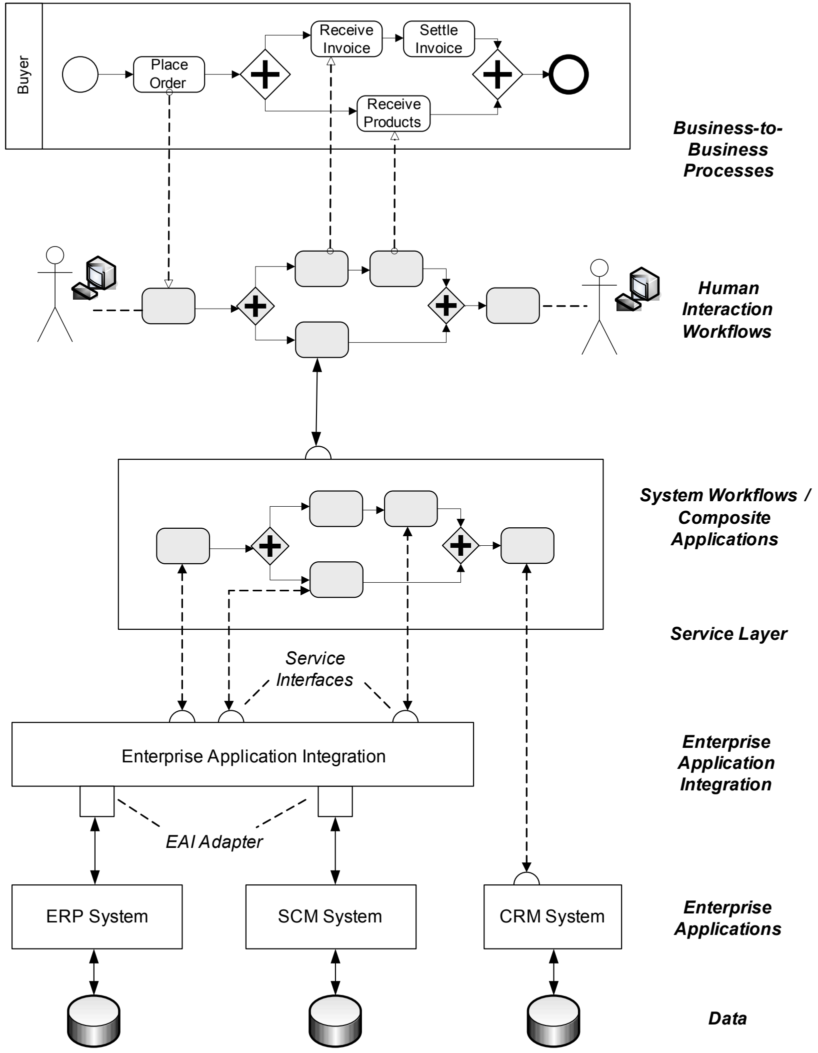
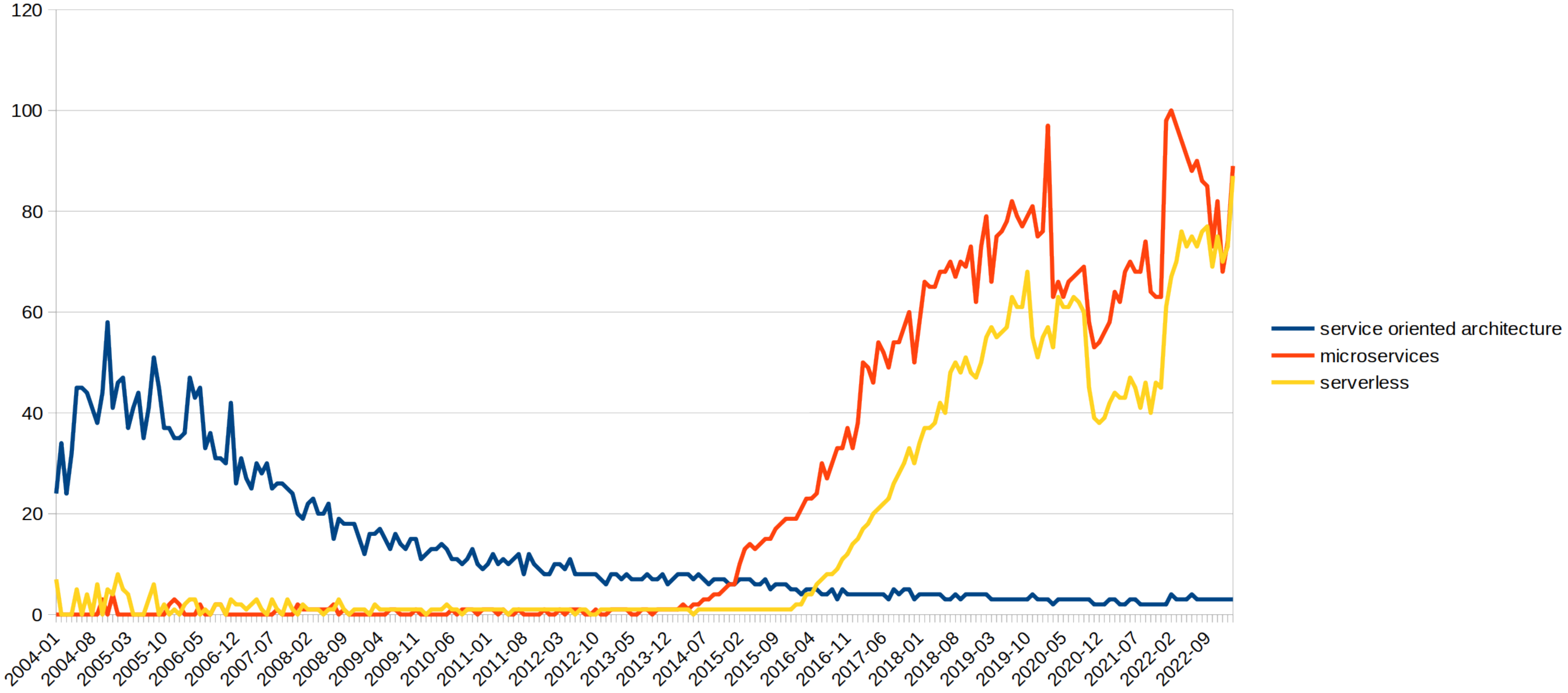

Buyer
Place Order
Receive Invoice
Settle Invoice
Receive Products
Business-to-Business Processes
Human Interaction Workflows
System Workflows / Composite Applications
Service Layer
Service Interfaces
Enterprise Application Integration
Enterprise Application Integration
EAI Adapter
ERP System
SCM System
CRM System
Enterprise Applications
Data

Fig 2.26. Business process management landscape

M. Weske: Business Process Management,
© Springer-Verlag Berlin Heidelberg 2007

The new wave (reprise)



[Google trends]

Why a new generation?

- To better address **existing** problems
- To address **new** problems

Software and quality

- External qualities
  - Qualities the end user can perceive
    - Functional
    - Non functional
  - Correctness, usability, efficiency, reliability, integrity, adaptability, accuracy, robustness
- Internal qualities
  - Qualities related to how the software is organized
  - Maintainability, flexibility, portability, re-usability, readability, testability, understandability

Steve McConnell - Code Complete

Evolving goals

- Focus shifts from inter-organization to intra-organization (not because of system integration needs)
- Focus shifts from consistency to availability
- Focus shifts from correctness to velocity: let the business evolve at its pace

Consistency and availability

- Consistency and availability are the most exemplary contrasting non-functional requirements that large, multi-user, distributed applications struggle with.

CONSISTENCY PARTITIONABILITY

- The CAP theorem (Gilbert and Lynch, 2002) states that in a partitionable system it is not possible to achieve full consistency and maximum availability.

AVAILABILITY

Microservices

Microservices as an architectural pattern

Forces:

- Services must be loosely coupled so that they can be **developed**, **deployed** and **scaled** independently

Enablers

- Agile software development
- Virtualization / containers
- Evolution in persistence management (NoSQL, messaging/events, …)
- Wide diffusion of enterprise-grade open source software

Microservices characterization

- Focus on availability and velocity
- Bottom-up
- Multiple tech stacks (polyglot programming, polyglot persistence)
- REST (not really...)
- DevOPs
- API Gateway

Microservices characteristics

Autonomous

- Single Responsibility Principle
- Distinct pipeline
- Own environment (process/host)
- Own data model

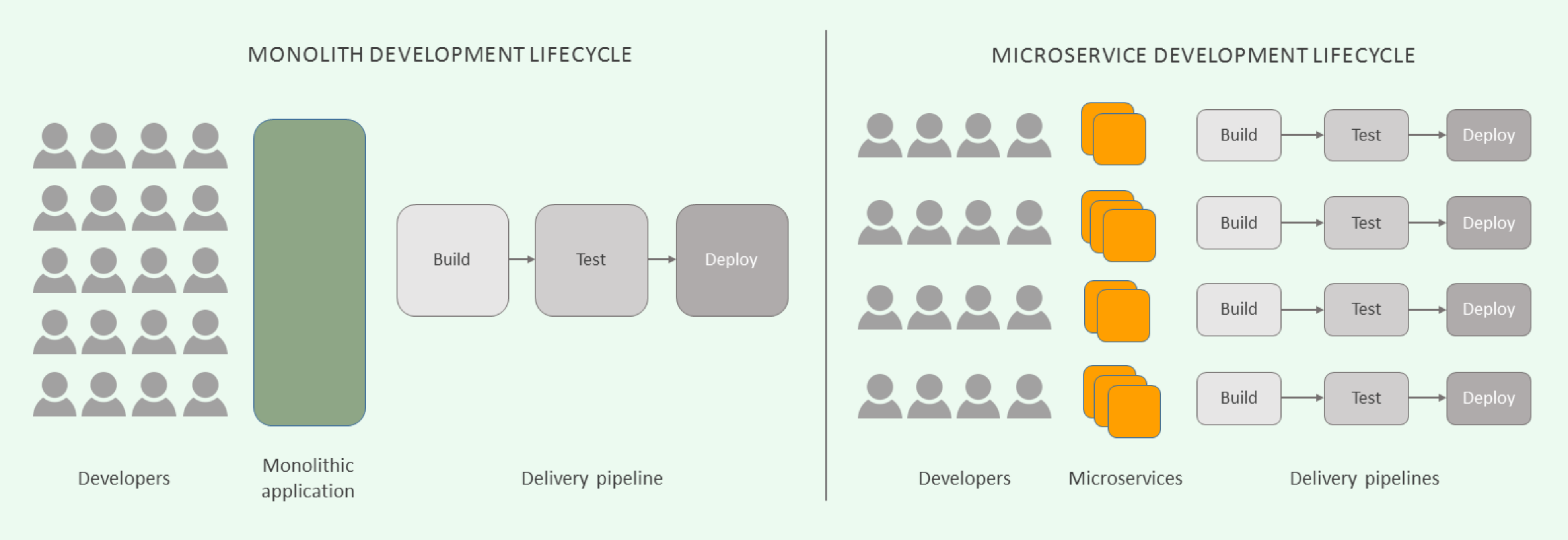
Microservices characteristics

- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for failure
- Evolutionary Design

NON VA
ESTERNALIZ.
LA LOGICA
DI BUSINESS

[James Lewis & Martin Fowler]

Microservices pipelines



[Kristijan Arsov]

API Gateway

→ SERVIZIO INFRASTRUTTURALE

An API Gateway is responsible for request routing, composition, and protocol translation

Responsibilities

- Facade + authentication
- Monitoring
- Load balancing
- Caching
- Request shaping and management
- Static response handling

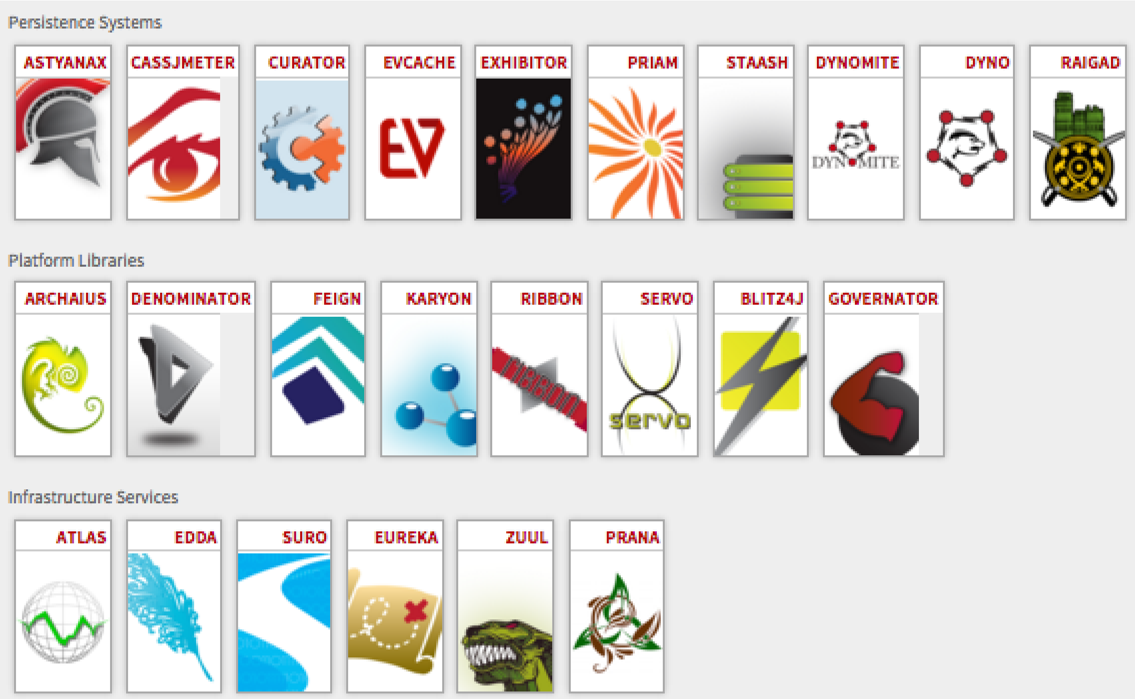
Criticism

From services hosted in containers as managed components (JEE) connected through a ESB to executables run in OS containers (Docker) and lots of infrastructure services (Api Gateway, ZooKeeper, Kafka, …)

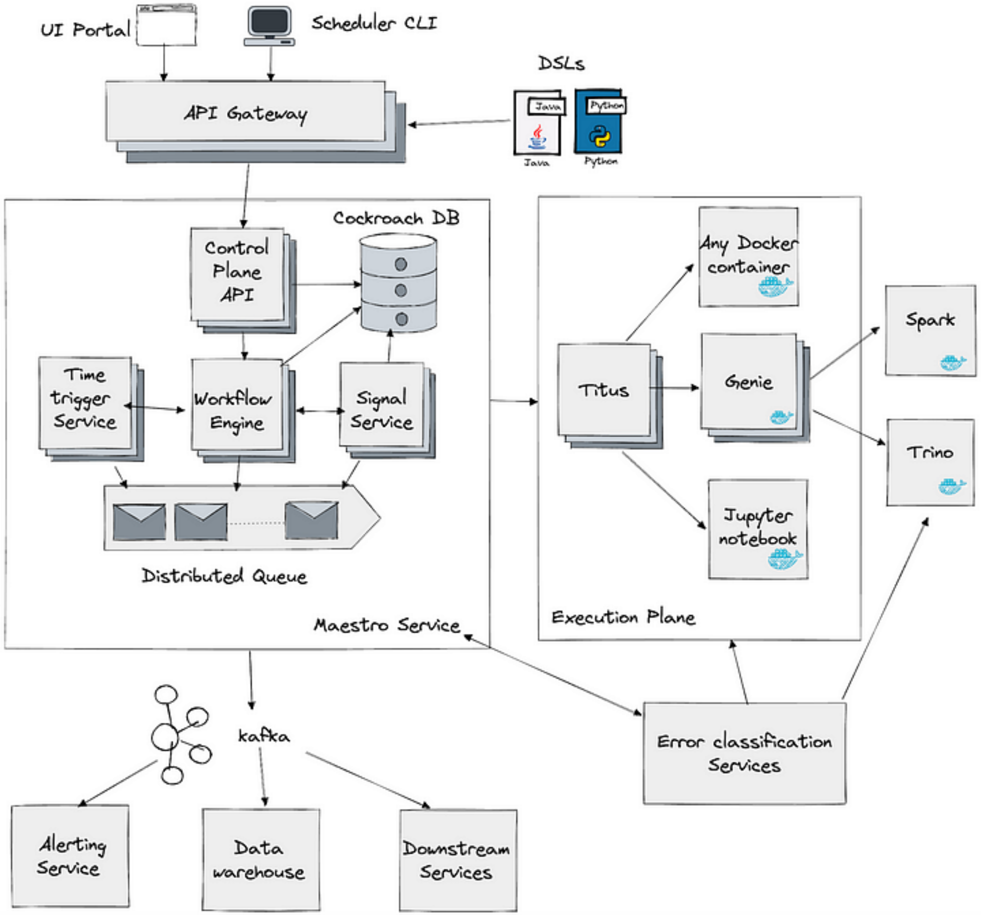
How much “infrastructure”?

Netflix’s case

- Eureka
- Ribbon
- Hystrix
- Zuul
- Curator
- Astyanax
- Conductor
- Memcached
- ...
- Which in turn depend upon ZooKeeper, Servo, Cassandra, ...



Old stuff? True. This is new.



Old wine in a new bottle?

- Not really
- Remember: microservices can be developed, deployed and scaled independently, it makes a lot of difference
- Yet, there is a price to pay

Complexity

- “The complexity of software is an essential property, not an accidental one.” — Fred Brooks, “The Mythical Man-Month”
- **When** it is correctly designed...

Serverless

Serverless computing refers to a model where the existence of servers is simply hidden from developers. I.e. that even though servers still exist developers are relieved from the need to care about their operation.

[Vidyasagar Machupalli – OpenWhisk]

- Pay-per-use business model
- Provider takes care of lifecycle, scaling, ...

Serverless providers

- Amazon Lambda
- Google Cloud Functions
- Azure Functions
- IBM Cloud Functions (Apache OpenWhisk)