Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Τμήμα Πληροφορικής & Τηλεπικοινωνιών

Τεχνολογία Λογισμικού

8ο Εξάμηνο 2022-23

Αρχιτεκτονική λογισμικού (2 / 2)

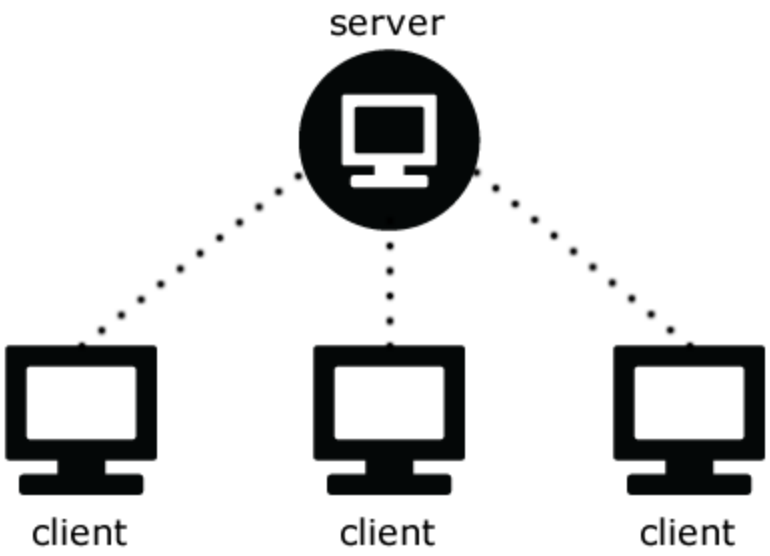
Δρ. Κώστας Σαΐδης (saiko@di.uoa.gr)

Αρχιτεκτονικά πρότυπα λογισμικού

- Client-Server
- Component-based
- Event-Driven
- Layered / N-tier
- Master-slave/Master-replica
- Message-driven/Publish-subscribe
- Microservices

- Model-View-Controller (MVC)
- Model-View-ViewModel (MVVM)
- Peer-to-peer (P2P)
- Pipeline / Pipe-filter
- Representation State Transfer (REST)
- Service-oriented
- Share-nothing
- World Wide Web

Client-Server

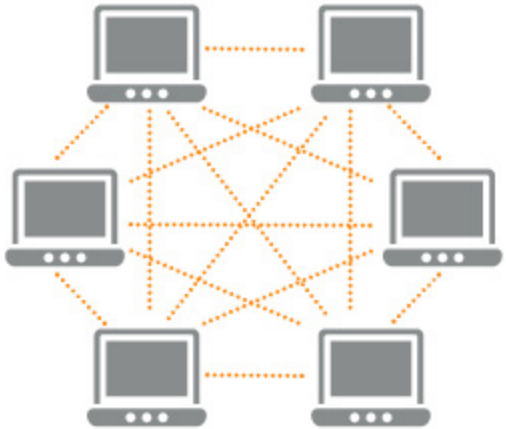


Χαρακτηριστικά

- Server-based
- N clients, 1 server
- Που επικοινωνούν μεταξύ τους με ένα συγκεκριμένο πρωτόκολλο για να υλοποιήσουν μια συγκεκριμένη "εφαρμογή"
- Παραδείγματα: WWW, IMAP, POP3, FTP, SSH, κ.ά

Peer-to-peer (P2P)

Server-Based

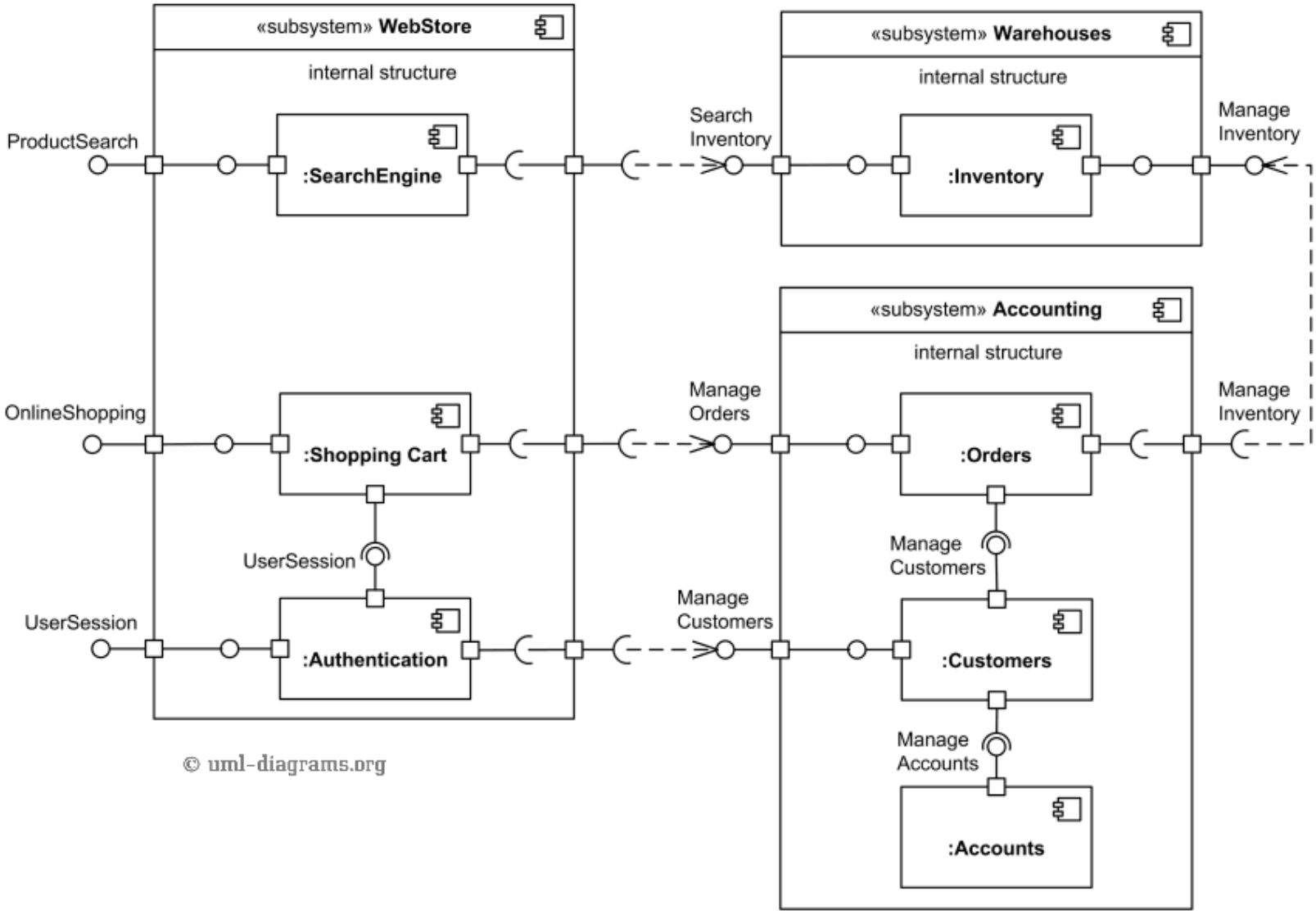


P2P

Χαρακτηριστικά

- Δίκτυο ομότιμων κόμβων
- Κάθε κόμβος είναι και client και server
- Οι κόμβοι επικοινωνούν μεταξύ τους με ένα συγκεκριμένο πρωτόκολλο για να υλοποιήσουν μια συγκεκριμένη "εφαρμογή"
- Παραδείγματα: File-sharing networks, Blockchain, Cryptocurrencies, κ.ά

Component-based



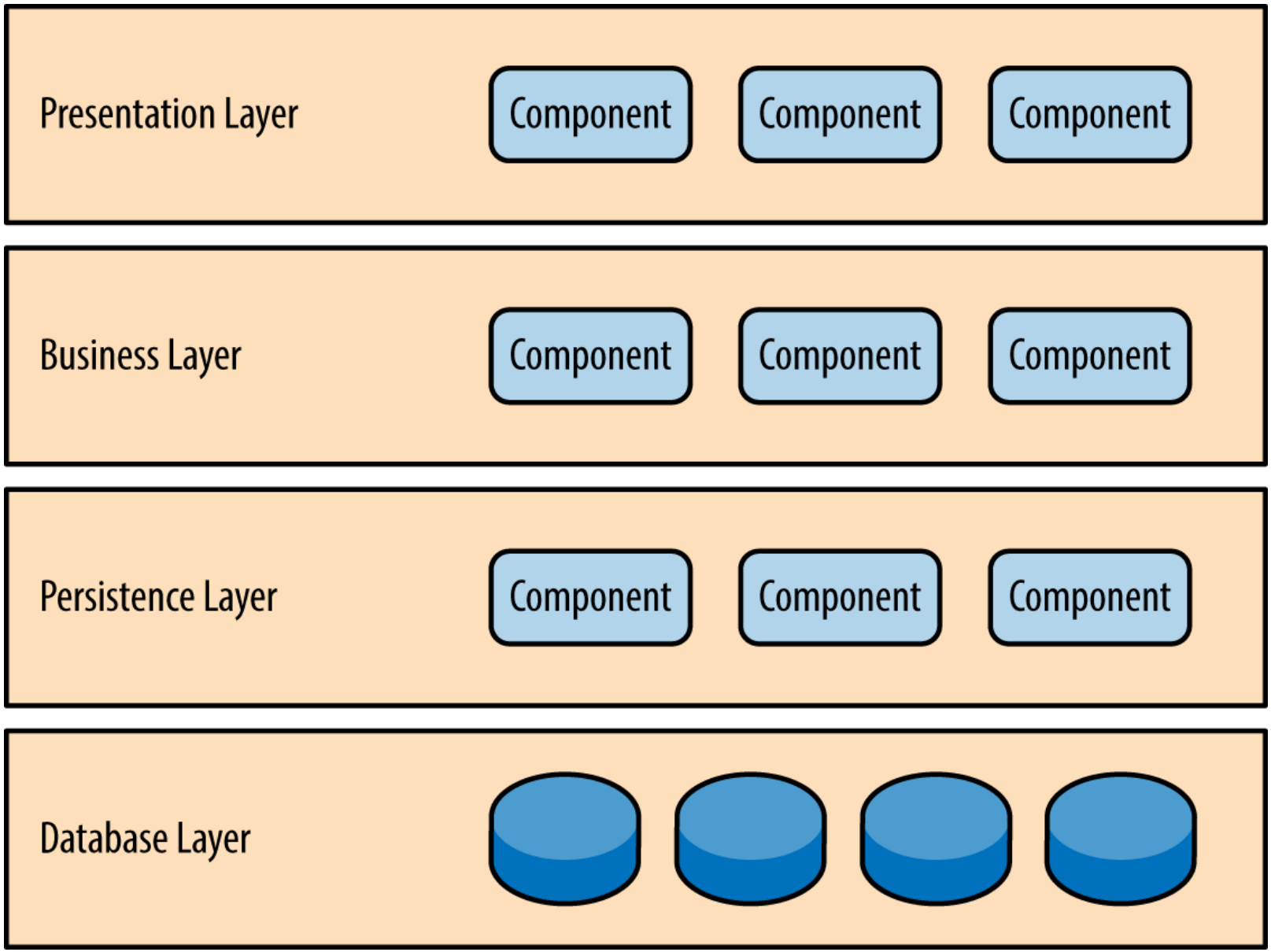
Χαρακτηριστικά

- Σχεδιασμός και αλληλεπίδραση των συστατικών του λογισμικού μέσω Interfaces
- Ένα component παρέχει/υλοποιεί ένα interface και απαιτεί την ύπαρξη/κάνει χρήση ενός άλλου
- Χαλαρή σύνδεση (loose coupling) και διαχωρισμός ενδιαφερόντων (separation of concerns)
- Application server: το λογισμικό που φιλοξενεί τα components

Κανόνας

Πάντα ξεκινάμε το σχεδιασμό του λογισμικού από τα Interfaces

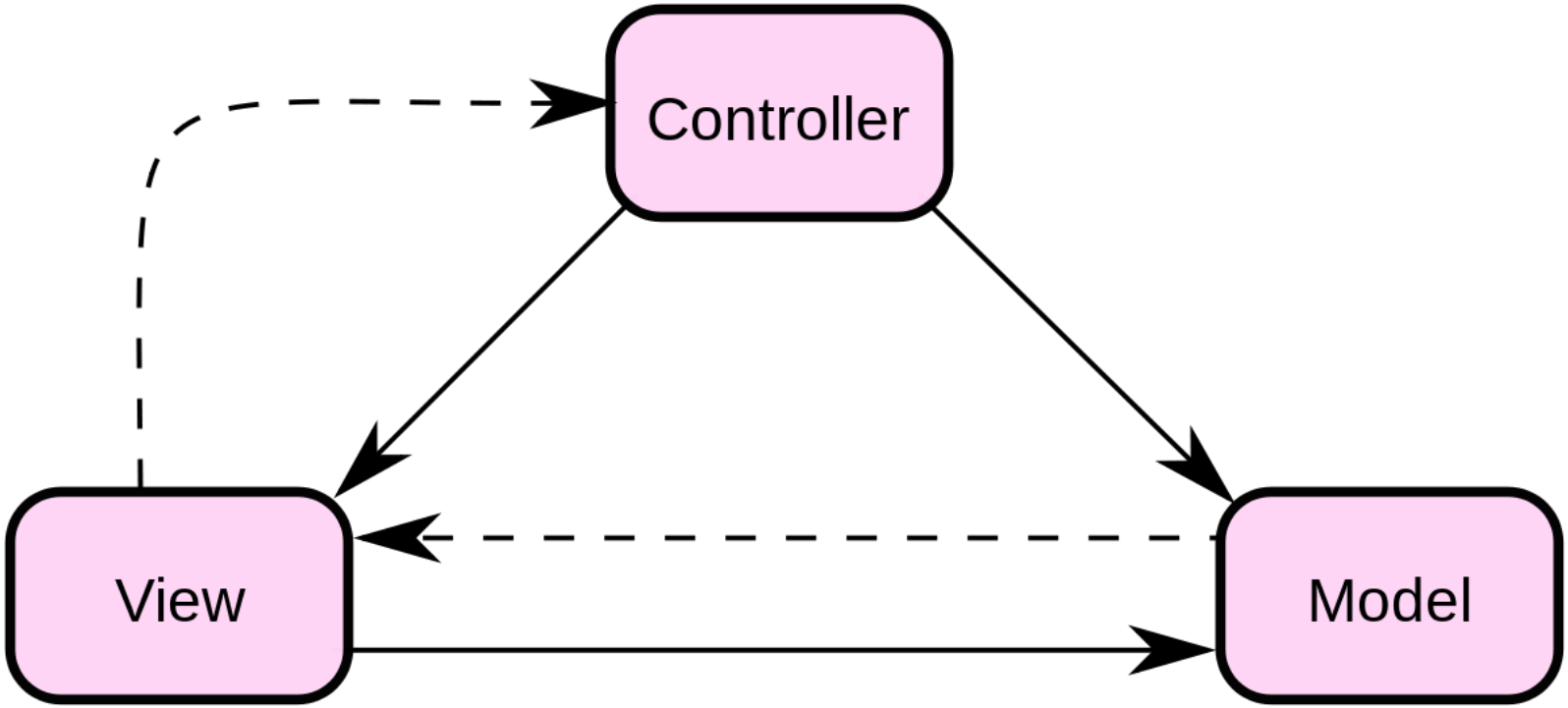
Layered/N-tier



Χαρακτηριστικά

- Server-based
- Λογική ή/και φυσική αρχιτεκτονική
- Ευρεία χρήση στις εφαρμογές διαδικτύου
- Frameworks: παρέχουν έτοιμα προς χρήση και παραμετροποίηση interfaces, components & layers

Model-View-Controller (MVC)



Χαρακτηριστικά

- Διαχωρισμός ενδιαφερόντων
- Controller
  - User input, request/response handling, επίβλεψη των Model, View
- Model
  - Data model, business logic
- View
  - Data display

Παράδειγμα

```
@Controller(url='/items')
class ItemController {

  void get(Request req, Response res) {
    //the view template
    Template t = loadTemplate('items')

    //the data model
    List<Item> items = store.loadItems()

    //the view context
    Context ctx = new Context()

    //add the data in the context
    ctx.put("items", items)

    //render the template using the context
    //and generate the response
    t.render(res.getWriter(), ctx)
  }

}
```

Παράδειγμα (συνέχεια)

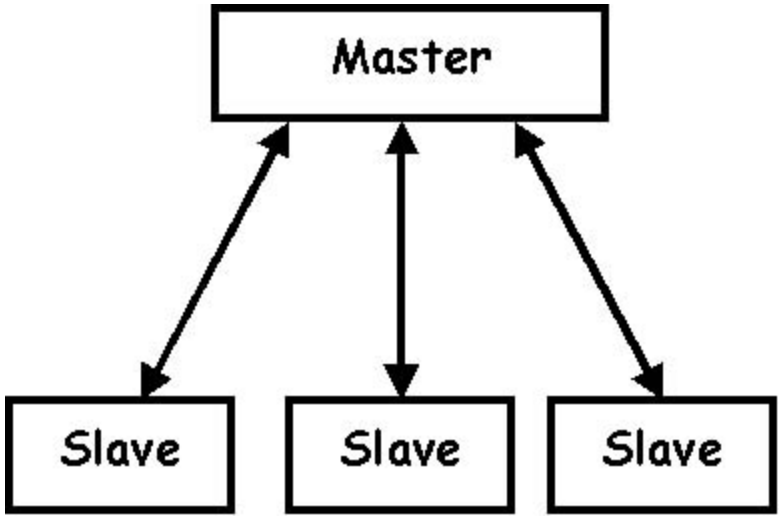
```
<table>
  { for item in items }
    <tr>
          <td> { item.title } </td>
          <td> { item.description } </td>
        </tr>
  { endfor }
</table>
```

MVC

Ευρεία χρήση στις εφαρμογές διαδικτύου

Πολλά (πάρα πολλά) frameworks

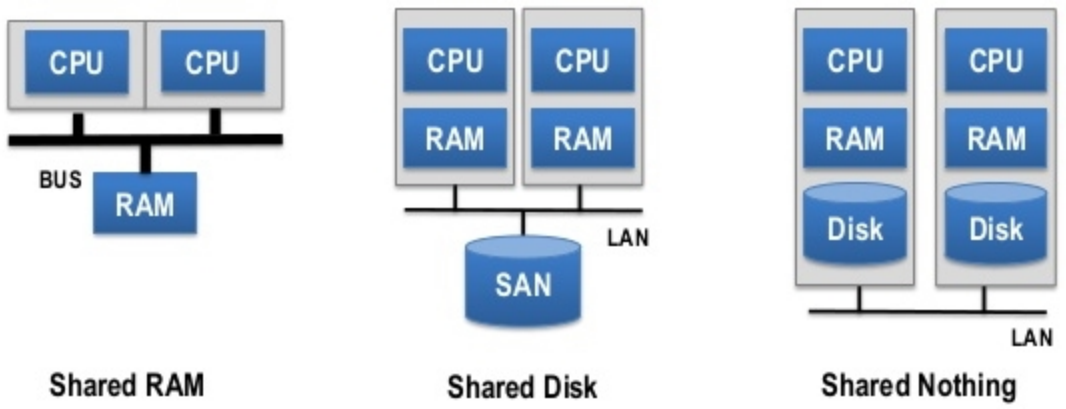
Master-Slave / Master-Replica



Χαρακτηριστικά

- N slaves, 1 master
- Master (authority), slaves (redundancy)
- Εφαρμογές: υψηλή διαθεσιμότητα, βελτίωση απόδοσης, επιμερισμός φόρτου, κ.ά
- Replication
  - Master copy of data, multiple replicas (slaves)
- Load balancing
  - Master dispatch logic, multiple "worker" nodes

Share-Nothing Architecture



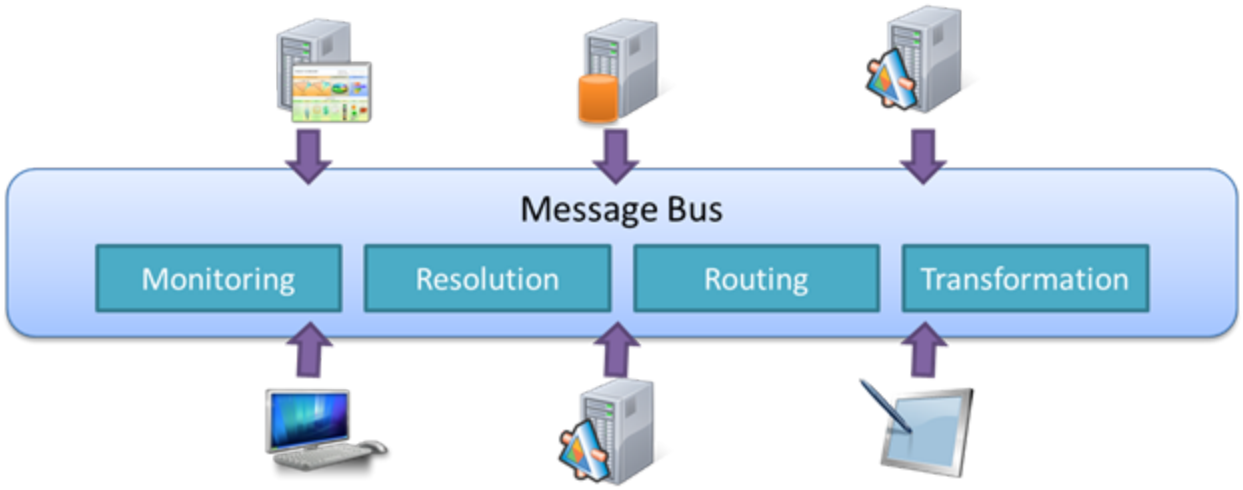
Χαρακτηριστικά

- Κάθε κόμβος είναι ανεξάρτητος και αυτοτελής.
- No single point of contention (δεν διαμοιράζονται πόροι, π.χ. μνήμη ή δίσκος).
- Sharding: οριζόντια επιμέρηση των δεδομένων.
- Οριζόντια κλιμάκωση (horizontal scalability) - απλή προσθήκη κόμβων.
- Η αρχιτεκτονική πολλών NoSQL συστημάτων.

Eventual Consistency

- **BASE Systems** (Basically Available, Soft state, Eventual consistency)
- Όταν πάψουν οι ενημερώσεις σε μια εγγραφή, τελικά (eventually) όλες οι αναγνώσεις της εγγραφής αυτής θα επιστρέψουν την πιο πρόσφατη ενημέρωση.
- Replica convergence (σύκλιση αντιγράφων)
- PA/EL (Επιλέγουν αυξημένη διαθεσιμότητα & μείωση καθυστέρησης αντί για συνέπεια)

Message-driven/Publish-subscribe



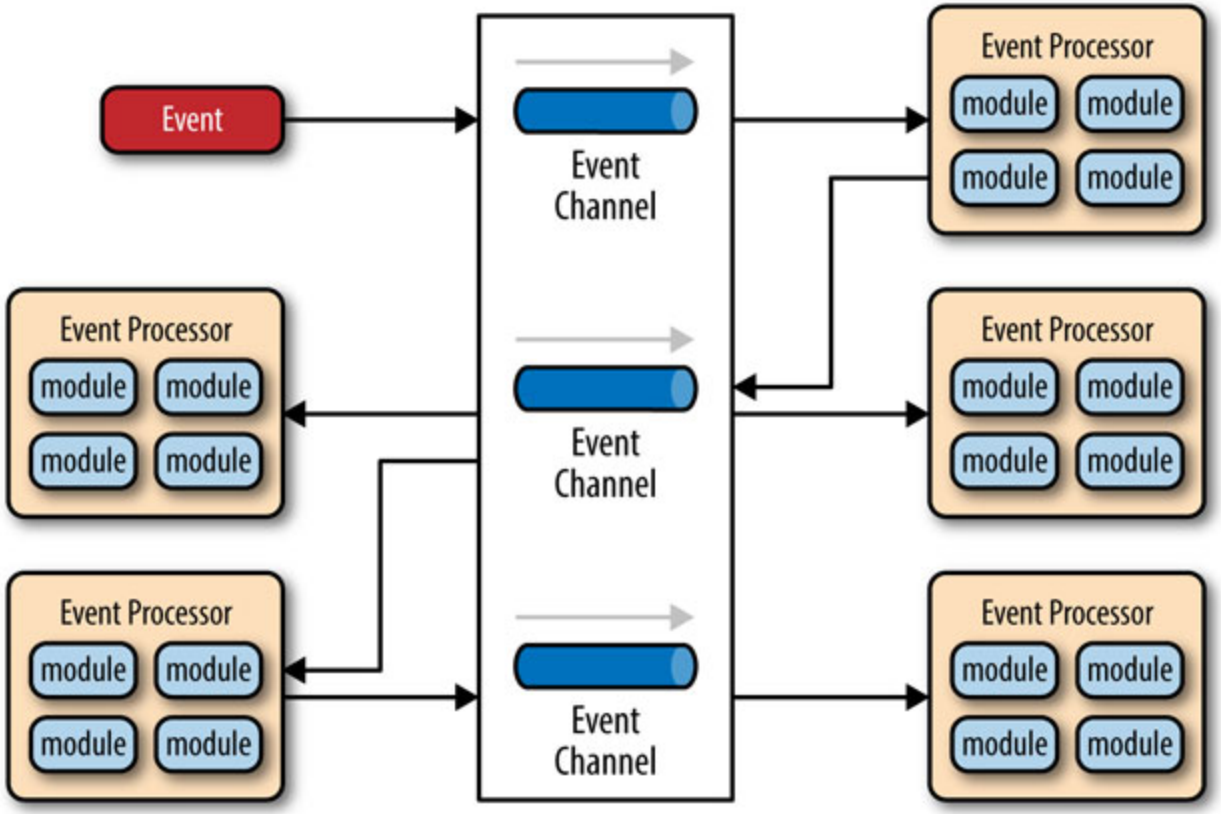
Χαρακτηριστικά

- Χαλαρή σύνδεση (loose coupling) μεταξύ συστατικών/εφαρμογών
- Publisher (producer): αποστολή μηνυμάτων
- Subscriber (consumer): λήψη μηνυμάτων
- Topics (channels): "κλάσεις/θέματα" μηνυμάτων
- Message Bus (broker): διαχερίση/δρομολόγηση μηνυμάτων σύγχρονα ή ασύγχρονα, με εγγυήσεις αποστολής ή όχι, με χρήση ουρών, με φιλτράρισμα ή όχι κτλ.

Εφαρμογές

- Middleware ολοκλήρωσης ετερογενών συστημάτων
- Επίτευξη υψηλής απόδοσης και κλιμάκωσης σε κατανεμημένα συστήματα
- Μειονέκτημα: δύσκολη η αλλαγή της δομής των μηνυμάτων

Event-driven



Χαρακτηριστικά

- Events & Event handlers (listeners, callbacks)
- Implicit invocation / Inversion of control
- Event thread / Event loop
- Εφαρμογές: γραφική διεπαφή χρήστη, server-side αρχιτεκτονική

Παράδειγμα (Javascript)

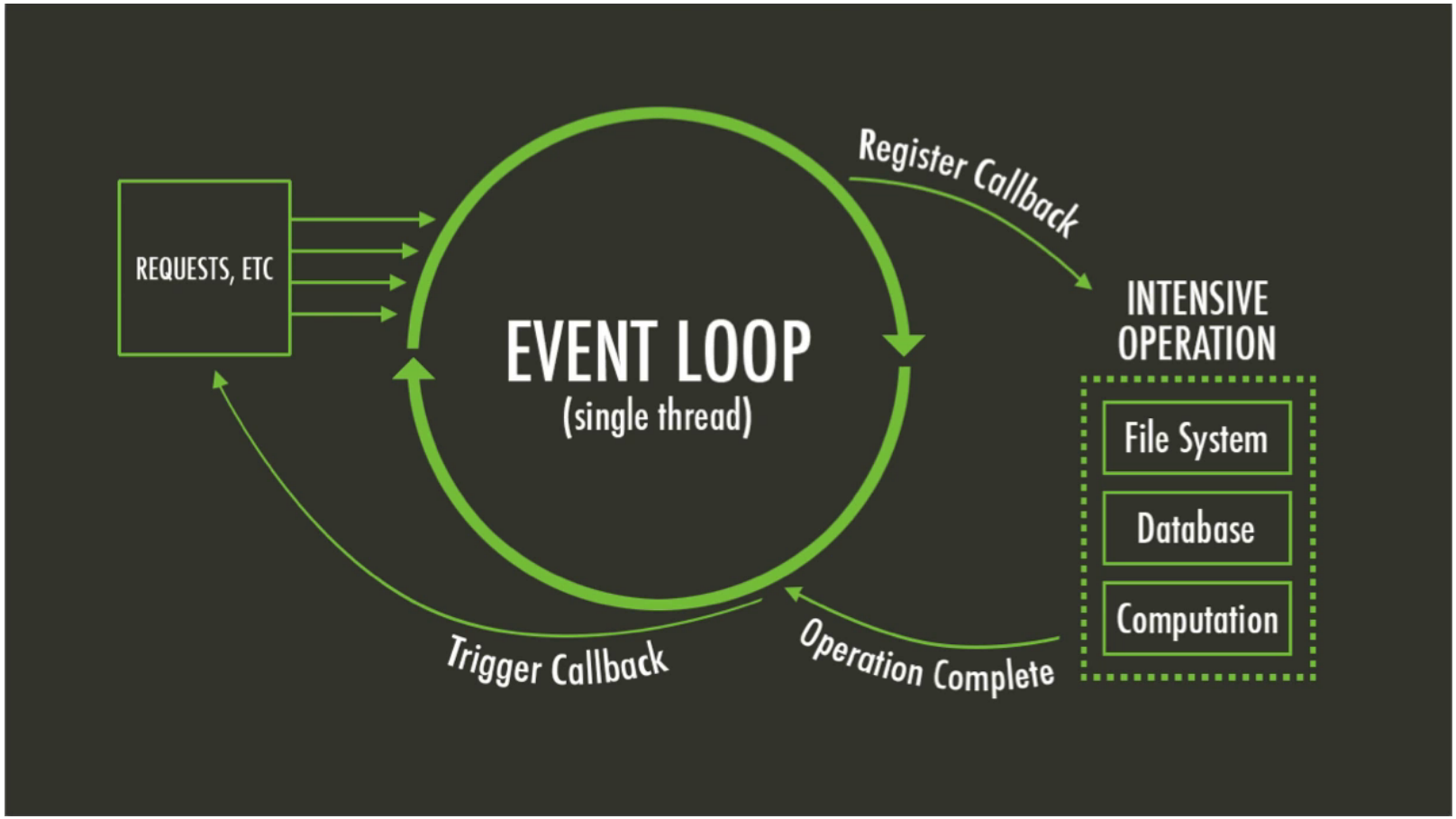
```
class EventEmitter {
  constructor() {
    this.events = new Map(); //Map<Event, Set<Listener>>
  }
  on(event, listener) {
    let listeners = this.events.get(event);
    if (!listeners) {
      listeners = new Set();
      this.events.set(event, listeners);
    }
    listeners.add(listener);
    return this;
  }
  emit(event, ...args) {
    const listeners = this.events.get(event);
    if (listeners) {
      for (let listener of listeners) {
        listener.apply(event, args);
      }
    }
    return this;
  }
}
```

```
const events = new EventEmitter();
events.on('foo', (e) => { console.log(e); });
events.emit('foo'); // Prints "foo"
```

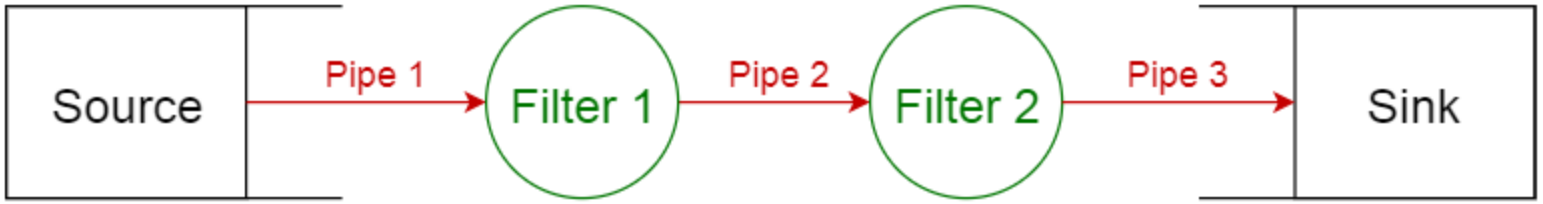
Παράδειγμα (Java)

```
public class MyPanel extends JPanel {
  public MyPanel() {
    JButton btn = new JButton("Do it");
    btn.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        //do it
      }
    });
    add(btn);
  }
}
```

Nodejs Event Loop



Pipeline / Pipe-filter



Χαρακτηριστικά

- Data streams, pipes and filters (data transformations)
- Συναρτησιακός προγραμματισμός
- Επαναχρησιμοποίηση, παραλληλισμός

Παράδειγμα (Java 8 streams)

```
List<String> l = Arrays.asList("a1","a2","b1","c2","c1");
l.stream()
 .filter(s -> s.startsWith("c"))
 .map(String::toUpperCase)
 .sorted()
 .forEach(System.out::println);
```

Output

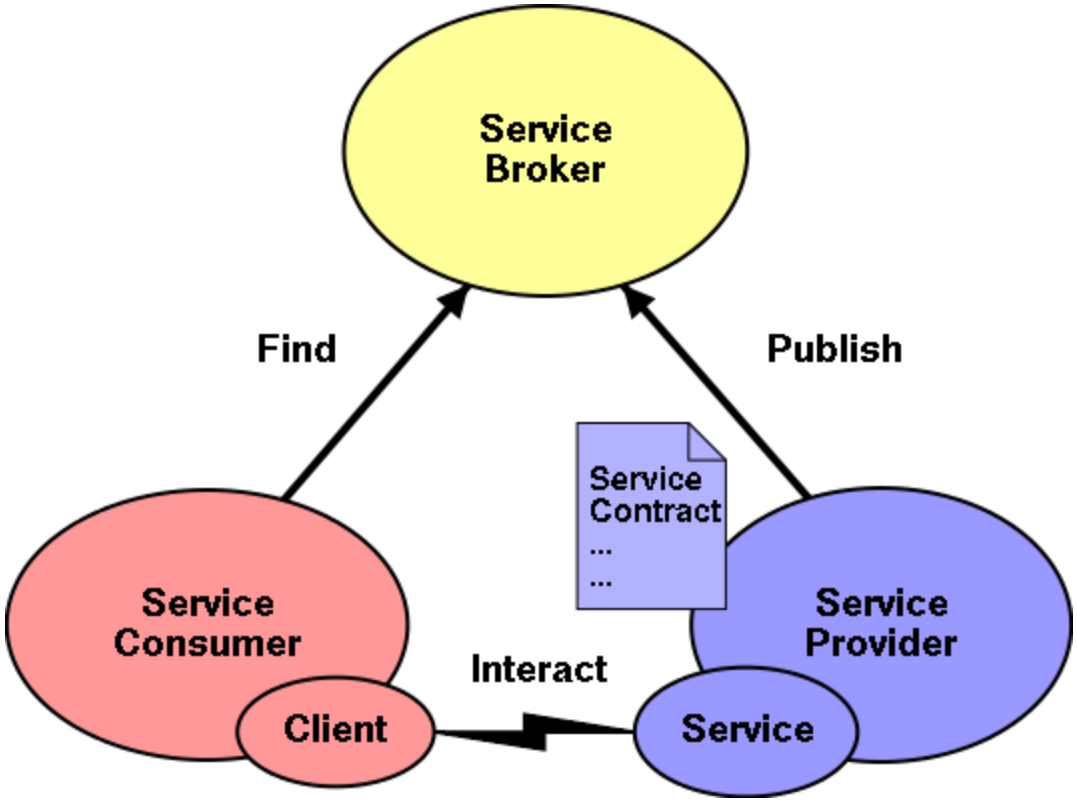
```
C1
C2
```

Service-oriented architecture

Η κεντρική ιδέα

- Υπηρεσίες που επικοινωνούν μέσω ενός πρωτοκόλλου επικοινωνίας και είναι:
  - κατανεμημένες,
  - αυτοτελείς (separately maintained & deployed)
  - χαλάρα συνδεδεμένες (loosely-coupled)
  - ανεξάρτητες της τεχνολογίας υλοποίησης (technology-neutral)
  - ανεξάρτητες του κατασκευαστή (no vendor lock-in)
- Σύνθεση της εφαρμογής μέσω της ολοκλήρωσης (integration) των υπηρεσιών.

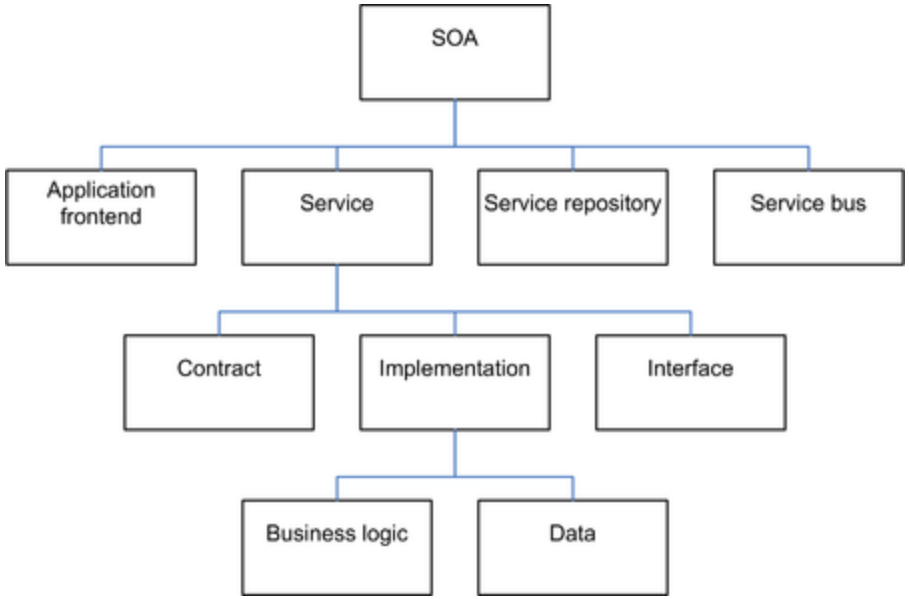
Ρόλοι



- Service consumer
- Service producer
- Service broker

SOA Αρχές

- Service contract
- Metadata
- Composition
- Autonomy
- Discovery
- Reusability


SOA
Application frontend
Service
Service repository
Service bus
Contract
Implementation
Interface
Business logic
Data

Υλοποίηση

- Web Services (SOAP, WSDL, UDDI)
- Remote-procedure call (RPC)
- Message-driven middleware
- RESTful APIs
- κ.ά

Ζητήματα

- Stateful vs Stateless
- Απόδοση
- Πολυπλοκότητα
- Έλεγχος και επαλήθευση (testing)

SOA & REST

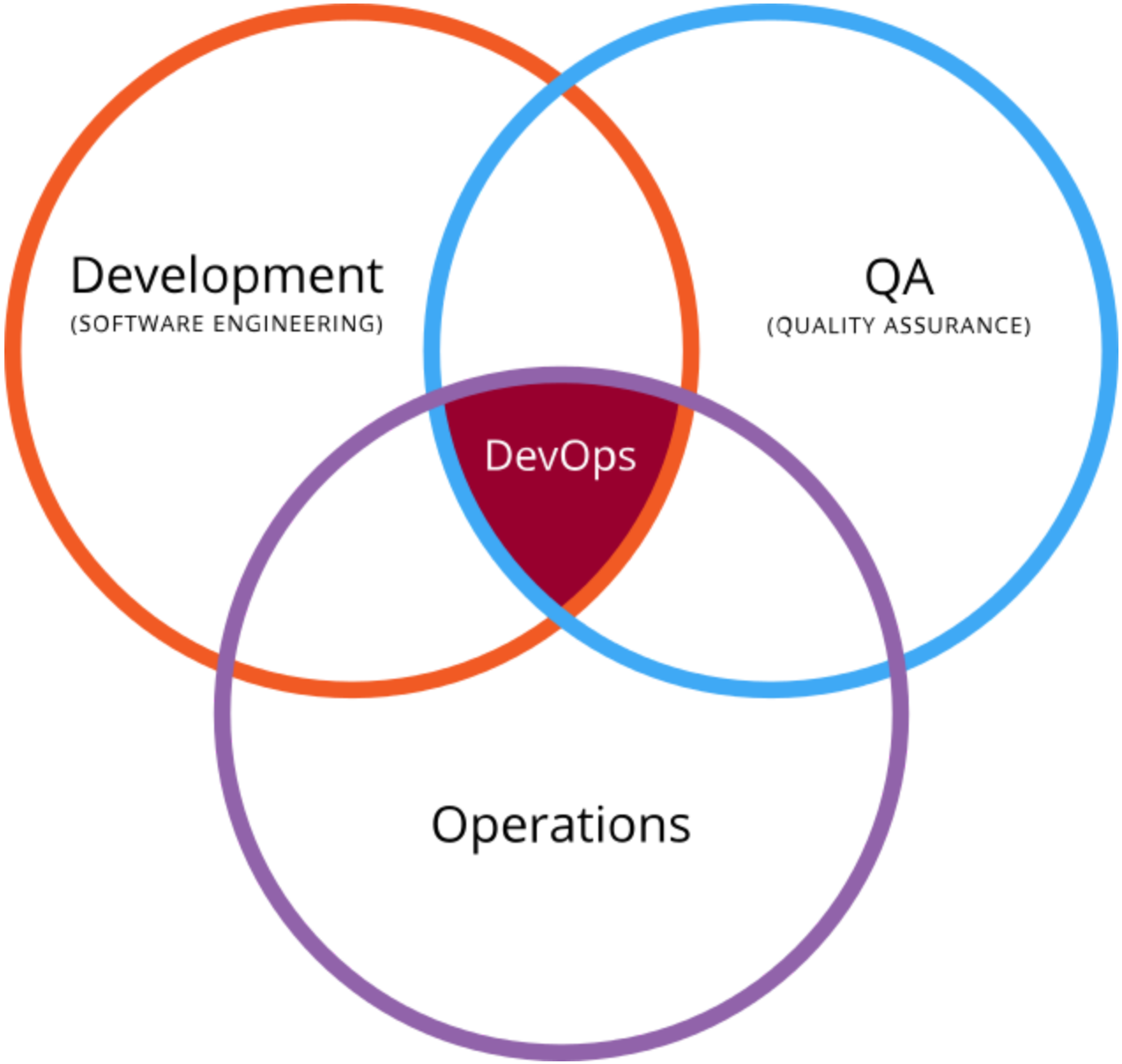
- Ένα σύνολο από RESTful HTTP end-points που ανταλλάσσουν δεδομένα σε μορφή JSON
- Ενδεχομένως ο απλούστερος τρόπος υλοποίησης μιας SOA

Microservices

Κωδικοποιημένα

Microservices = SOA + Unix Principles + Agile + DevOps

DevOps

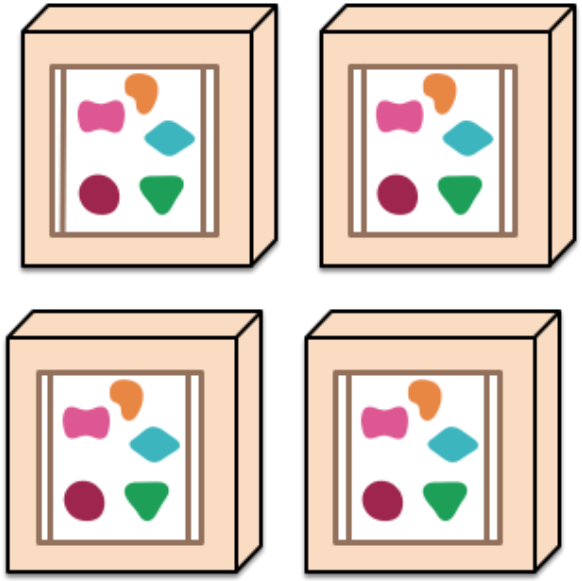

Development
(SOFTWARE ENGINEERING)
QA
(QUALITY ASSURANCE)
DevOps
Operations

Monolithic Server-side Applications

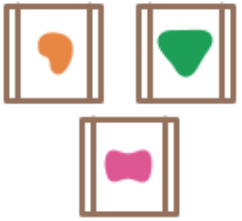
- A single logical executable
- Cloud deployment issues:
  - Small changes -> rebuild and redeploy the whole app
  - Scalability -> scale it all or not at all

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

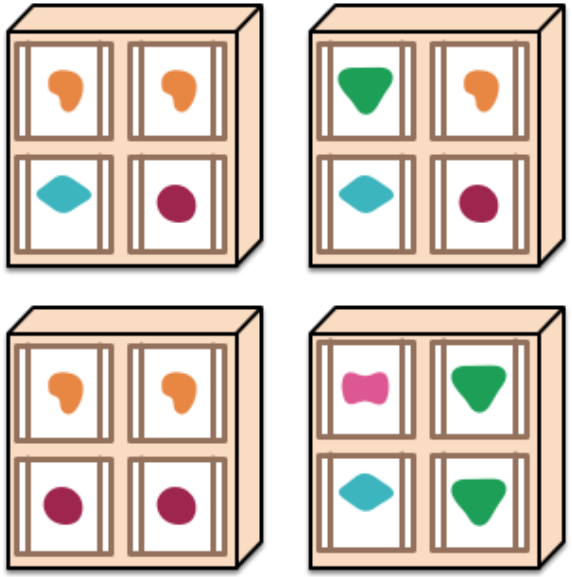
... and scales by replicating the monolith on multiple servers



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



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



Χαρακτηριστικά

- Componentization via services
  - As in SOA
- Organized around business capabilities
  - Cross-functional teams, as in Agile
- Products not projects
  - A team should own a product
  - You build it, you run it
- Smart end-points, dumb pipes
  - Simple communication (Unix-style)

- Decentralized governance
  - Each team governs the implementation details of its product
- Decentralized data managument
  - Different datastore per service/app
- Infrastructure automation
  - Build & test automation
  - Continuous delivery & deployment
- Design for failure
  - Sophisticated monitoring & logging

Devops toolchain for Microservices