05α Αρχιτεκτονικά πρότυπα λογισμικού

Τεχνολογία Λογισμικού

Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών
Εθνικό Μετσόβιο Πολυτεχνείο

Χειμερινό εξάμηνο 2017-18

Δρ. Κώστας Σαΐδης (saiko@di.uoa.gr)

Περιεχόμενα

- Αρχιτεκτονική λογισμικού
- Ποιοτικά χαρακτηριστικά κατανεμημένων συστημάτων
- Αρχιτεκτονικά στυλ και πρότυπα (architectural styles & patterns)

Αρχιτεκτονική λογισμικού

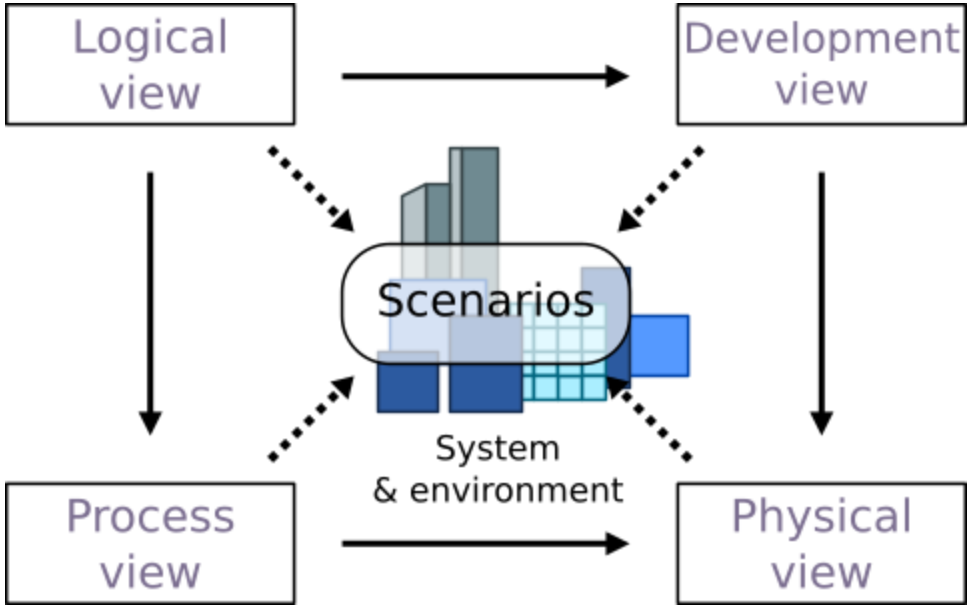
Περί τίνος πρόκειται

Η λήψη των θεμελιωδών δομικών και σχεδιαστικών αποφάσεων για το λογισμικό που είναι ακριβές να αλλάξουν αφού υλοποιηθούν.

Η αρχιτεκτονική πάει μαζί με

- Μεθοδολογία ανάπτυξης
- Ανάλυση απαιτήσεων
- Σχεδιασμό
- κ.ά (που έχουμε συζητήσει)

Πολλές οπτικές (4+1)



Ειδικότερα

- Logical view: έμφαση στη λειτουργικότητα του συστήματος σε υψηλό επίπεδο
- Physical view: έμφαση στην τοπολογία και διασύνδεση των συστατικών του σε φυσικό επίπεδο (deployment)
- Development view: έμφαση στην οπτική του προγραμματιστή
- Process view: έμφαση στη δυναμική συμπεριφορά του συστήματος κατά την εκτέλεσή του (απόδοση, κλιμάκωση, κτλ.)
- Scenarios - Use case view: έμφαση στη χρηστική πλευρά του συστήματος και στους σχετικούς ελέγχους αποδοχής

2ο Παραδοτέο

Πληρέστερο αν παρουσιάζει όλες τις οπτικές!

Αρχιτεκτονικά πρότυπα (architectural patterns)

Γενικές κι επαναχρησιμοποιήσιμες λύσεις σε κοινά προβλήματα αρχιτεκτονικής.

Αρχιτεκτονικά στυλ (architectural styles)

Όπως και στην "κανονική" αρχιτεκτονική, το στυλ είναι μια συγκεκριμένη μέθοδος κατασκευής που χαρακτηρίζεται από συγκεκριμένα αξιοπρόσεκτα χαρακτηριστικά.

An architectural style is a named collection of architectural design decisions that:

1. are applicable in a given development context,
2. constrain architectural design decisions that are specific to a particular system within that context, and
3. elicit beneficial qualities in each resulting system.

Λίστα αναγνωσμάτων

Richard N. Taylor, Nenad Medvidovic, Eric Dashofy, "Software Architecture: Foundations, Theory, and Practice", 2009, Wiley and Sons, ISBN: 0470167742

Αρχιτεκτονικά στυλ και πρότυπα

- Στο μάθημα δεν θα επιμείνουμε στη διάκριση.
- Είτε τα ονομάσουμε στυλ, είτε πρότυπα, μας εφοδιάζουν με μια κοινή γλώσσα ή λεξιλόγιο για να περιγράψουμε κατηγορίες συστημάτων.
- Είναι σύνηθες να συνυπάρχουν και να συνδυάζονται πολλά αρχιτεκτονικά πρότυπα και στυλ σε μια εφαρμογή.

Στο μάθημα θα δούμε (αλφαβητικά)

- 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

*** Σε επόμενη διάλεξη**

Ποιοτικά χαρακτηριστικά κατανεμημένων συστημάτων

Ορισμός

Ένα κατανεμημένο σύστημα αποτελείται από ξεχωριστά συστατικά που:

- Λειτουργούν σε ένα δίκτυο υπολογιστών.
- Επικοινωνούν μεταξύ τους μέσω ανταλλαγής μηνυμάτων.
- Αλληλοεπιδρούν για την επίτευξη ενός κοινού στόχου.

Πλάνες σχετικά με τα κατανεμημένα συστήματα (α)

1. Το δίκτυο είναι αξιόπιστο (reliable).
2. Η καθυστέρηση (latency) του δικτύου είναι μηδενική.
3. Το εύρος ζώνης (bandwidth) είναι άπειρο.
4. Το δίκτυο είναι ασφαλές (secure).

Πλάνες σχετικά με τα κατανεμημένα συστήματα (β)

5. Η τοπολογία (topology) του δικτύου δεν αλλάζει.
6. Υπάρχει μόνο ένας διαχειριστής (administrator).
7. Το κόστος μεταφοράς (transport) δεδομένων είναι μηδενικό.
8. Το δίκτυο είναι ομογενές (homogeneous).

Βασικά χαρακτηριστικά

- Συνέπεια δεδομένων (Consistency)
- Διαθεσιμότητα συστήματος (Availability)
- Αστοχία δικτύου (Network partition)
- Καθυστέρηση αίτησης/απόκρισης (Latency)
- Αιτήσεις ανά μονάδα χρόνου (Throughput)
- Κλιμάκωση (Scalability)

Consistency (C)

- Η συνέπεια των δεδομένων.
- Ζητούμενο: κάθε ανάγνωση (read) λαμβάνει την πιο πρόσφατη ενημέρωση (write) ή το σχετικό σφάλμα.

Η συνέπεια που υπόσχεται η αρχή ACID των δοσοληψιών στις βάσεις δεδομένων είναι πιο αυστηρή.

ACID Transactions

- Atomicity
  - Η δοσοληψία πετυχαίνει ή αποτυγχάνει πλήρως
- Consistency
  - Μετάβαση της βάσης σε πάντα έκγυρη κατάσταση
- Isolation
  - Απομόνωση της εκτέλεσης των δοσοληψιών
- Durability
  - Μονιμότητα των αποτελεσμάτων των δοσοληψιών

Availability (A)

- Η διαθεσιμότητα της εφαρμογής.
- Ζητούμενο: κάθε αίτηση (request) να λαμβάνει μια απάντηση (μη λάθους).
- Χωρίς να προσφέρονται πάντα εγγυήσεις ότι η απάντηση περιέχει την πιο πρόσφατη ενημέρωση (write).
- Η υψηλή διαθεσιμότητα απαιτεί αντιγραφές (replication).

Network Partition (P)

- Αστοχία δικτύου.
- Παράδειγμα: απώλεια σύνδεσης με τη βάση δεδομένων.

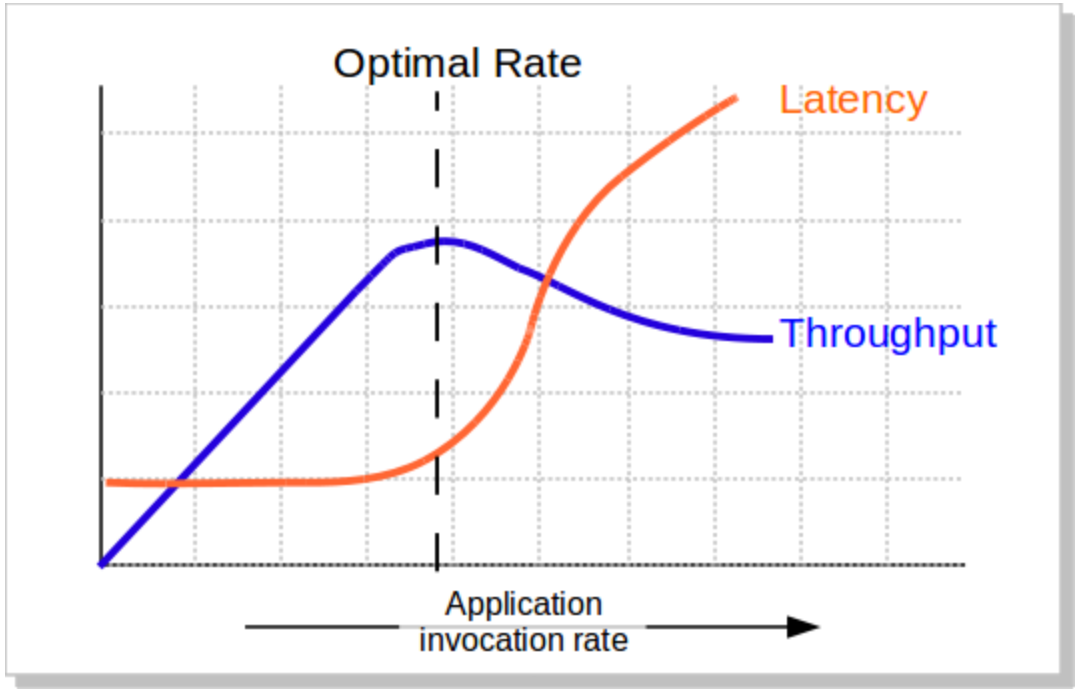
Latency (L)

- Η καθυστέρηση στην απόκριση του συστήματος.
- Ζητούμενο: να ελαχιστοποιηθεί ο χρόνος που απαιτείται για την ικανοποίηση μιας αίτησης.

Throughput

- Το πλήθος των αιτήσεων που ικανοποιούνται από το σύστημα ανά χρονική στιγμή.
- Ζητούμενο: να μεγιστοποιηθεί το πλήθος των αιτήσεων που μπορούν να ικανοποιηθούν ανά χρονική στιγμή.

Latency vs Throughput



docs.voltdb.com

Το θεώρημα CAP

Σε περίπτωση αστοχίας δικτύου (P), θα έχουμε είτε συνέπεια των δεδομένων (C) είτε διαθεσιμότητα της εφαρμογής (A), όχι και τα δύο.

```
if (P) { A or C }
```

Το θεώρημα PACELC

Επέκταση του CAP

Αν δεν υπάρχει P, θα έχουμε είτε συνέπεια των δεδομένων (C) είτε την ελάχιστη δυνατή καθυστέρηση (L), αλλά όχι και τα δύο.

```
if (P) { A or C }
else   { L or C }
```

Λίστα αναγνωσμάτων

Daniel Abadi, "Consistency Tradeoffs in Modern Distributed Database System Design", IEEE Computer, Volume 45, Issue 2, Feb. 2012.

Κατηγοριοποίηση κατανεμημένων συστημάτων

PA/EL

```
if P then A else L
```

PC/EC

```
if P then C else C //ACID databases
```

PC/EL

```
if P then C else L
```

Scalability

Η δυνατότητα ενός συστήματος να επαυξηθεί για να διαχειριστεί αυξημένο φόρτο.

Ειδικότερα

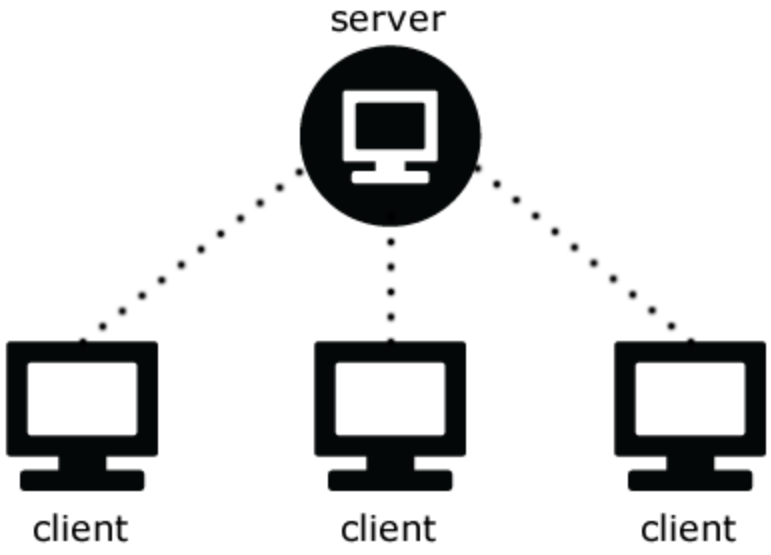
Αν η αύξηση της απόδοσης του συστήματος είναι αναλογική της αύξησης σε υπολογιστικούς πόρους (προσθήκη υλικού), τότε το σύστημα κλιμακώνεται (scales).

Οριζόντια και κάθετη κλιμάκωση

- Οριζόντια (scale out/in): αύξηση/μείωση των κόμβων.
- Κάθετη (scale up/down): αύξηση/μείωση των πόρων ενός κόμβου.

Αρχιτεκτονικά στυλ και πρότυπα

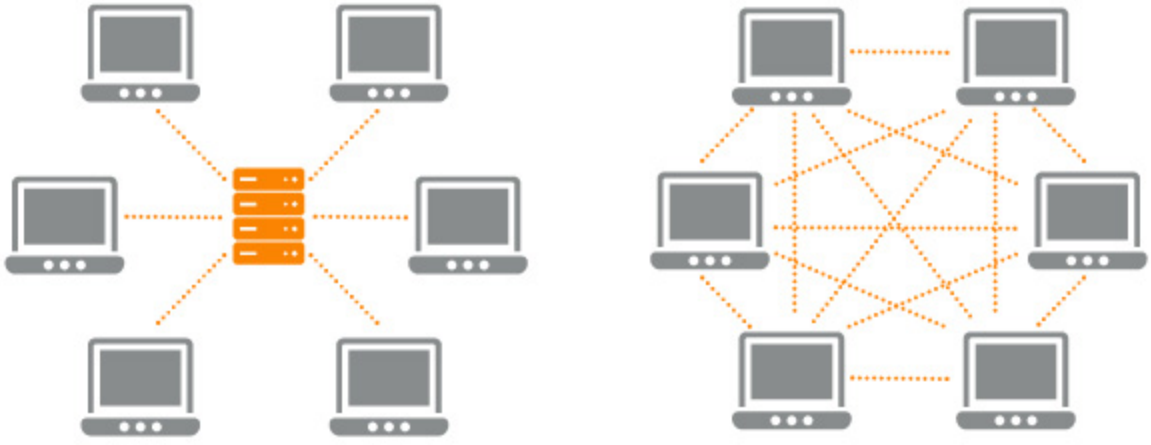
Client-Server



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

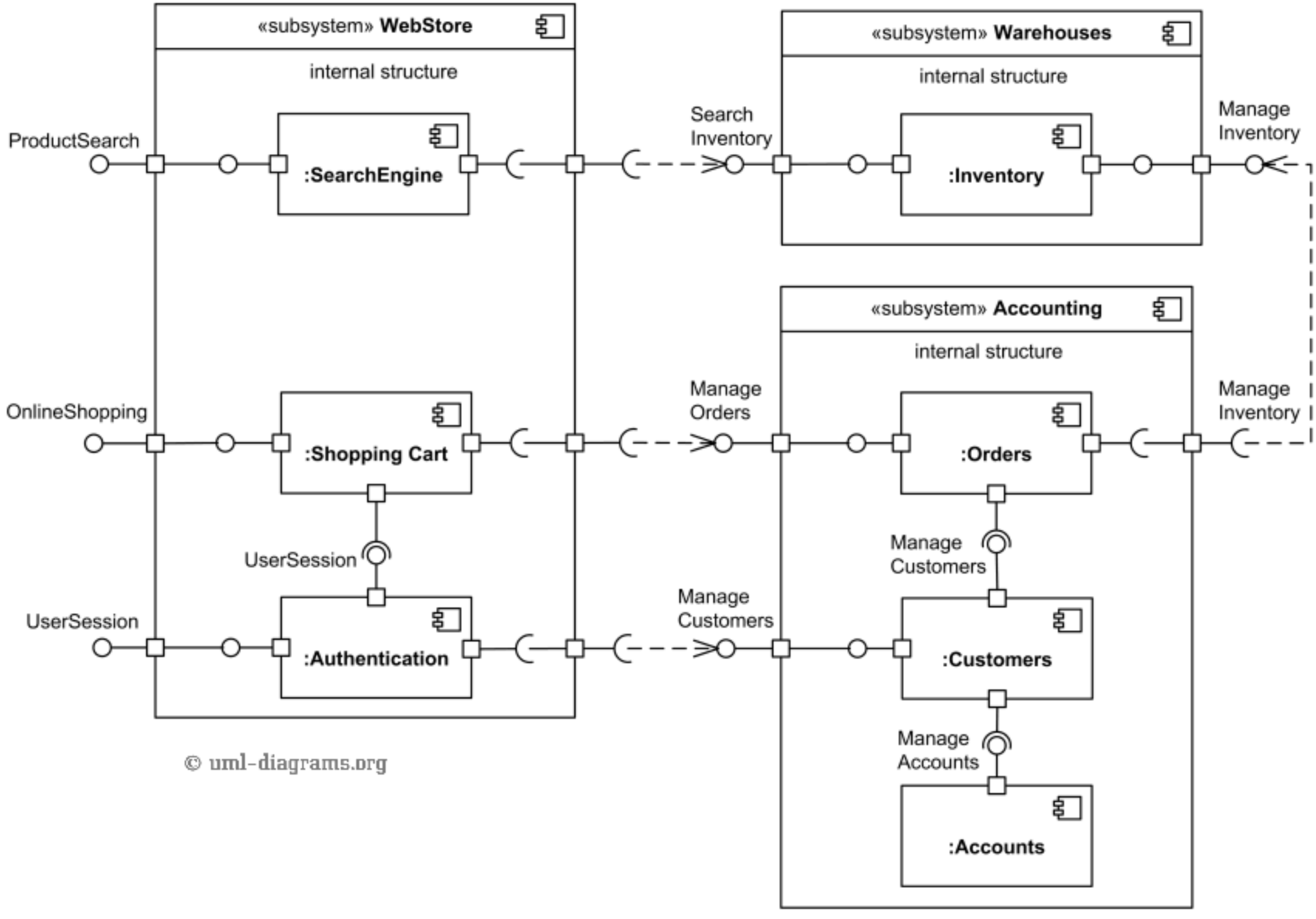
Peer-to-peer (P2P)



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, Cryptocurrencies, κ.ά

Component-based



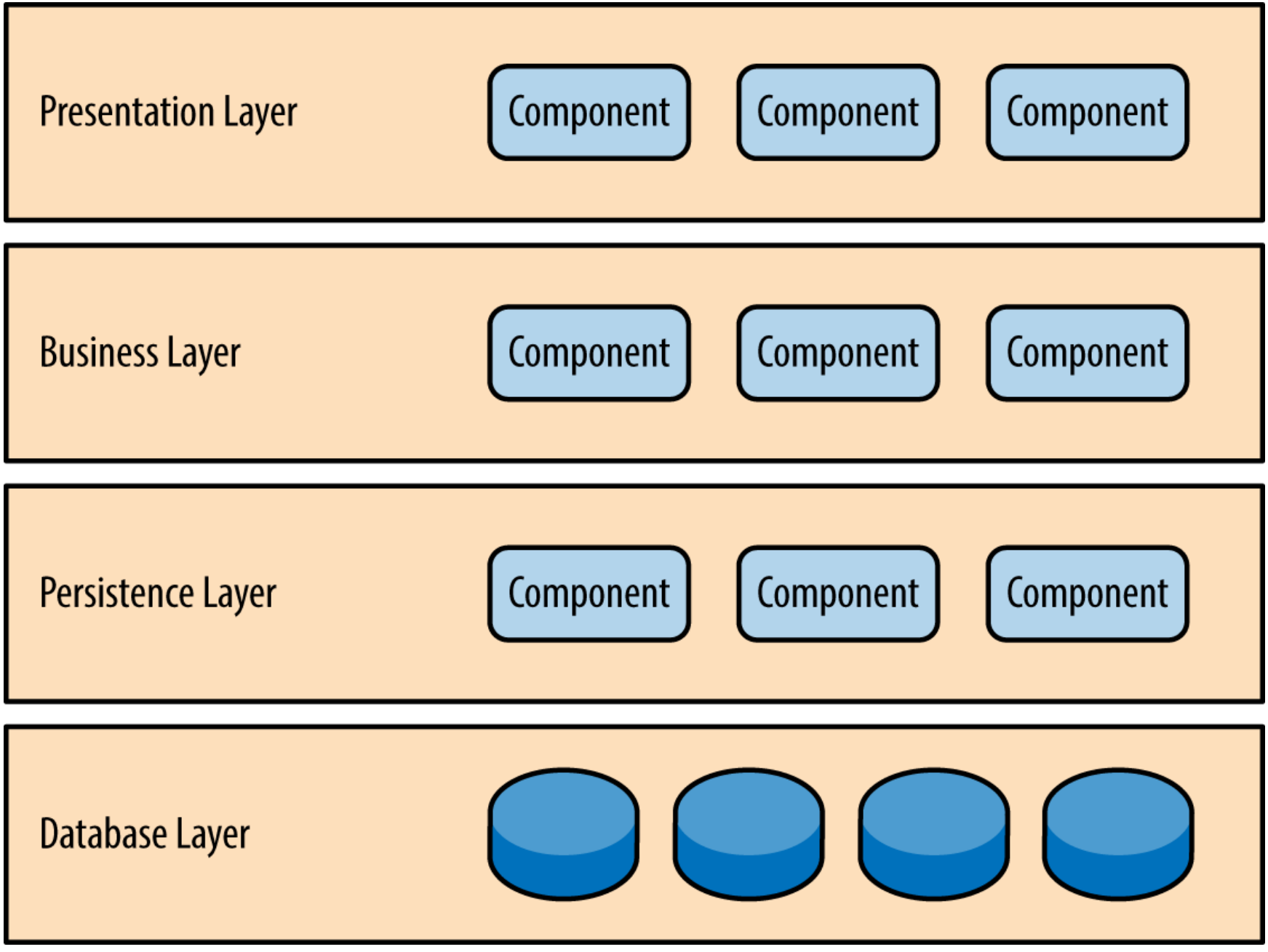
Χαρακτηριστικά

- 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

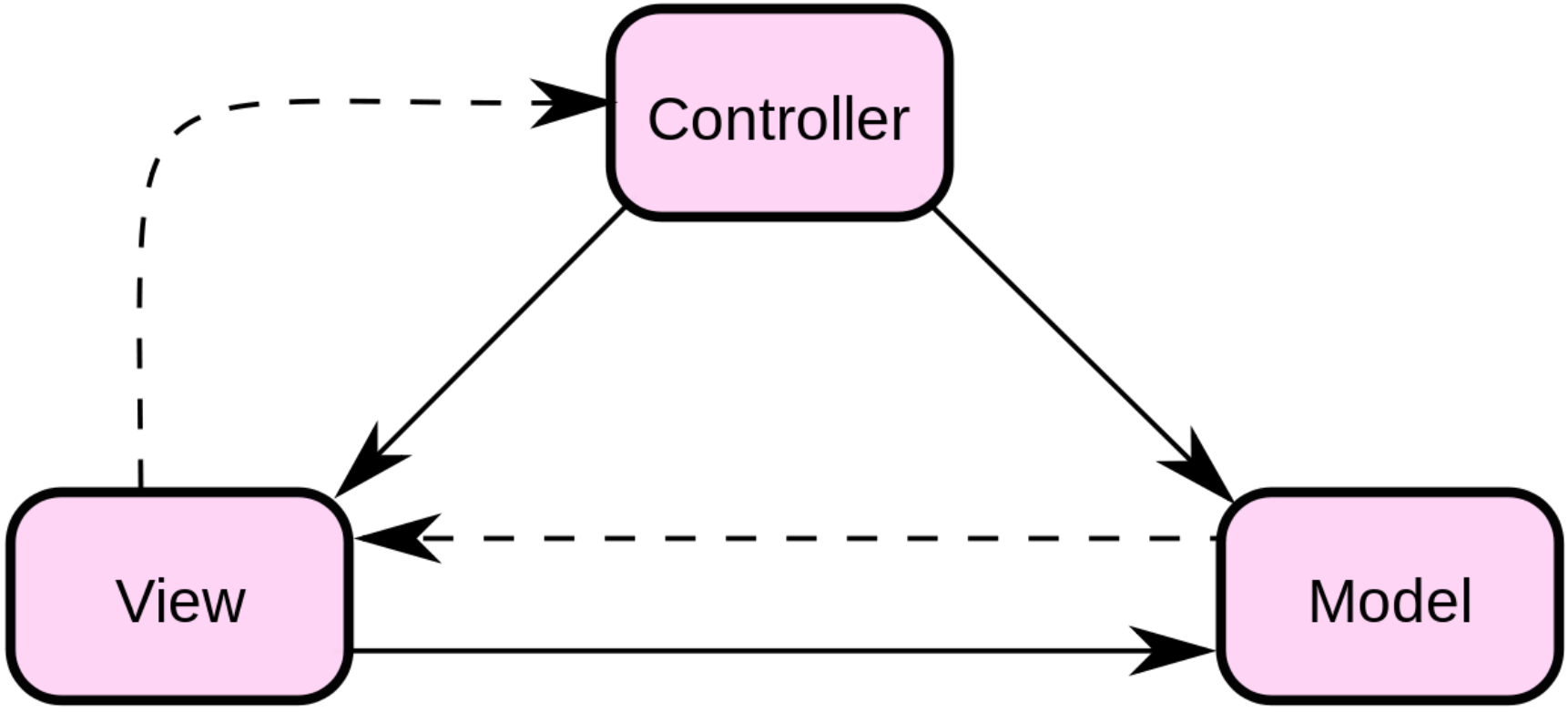
Layered/N-tier



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
- Ευρεία χρήση στις εφαρμογές διαδικτύου, πολλά frameworks

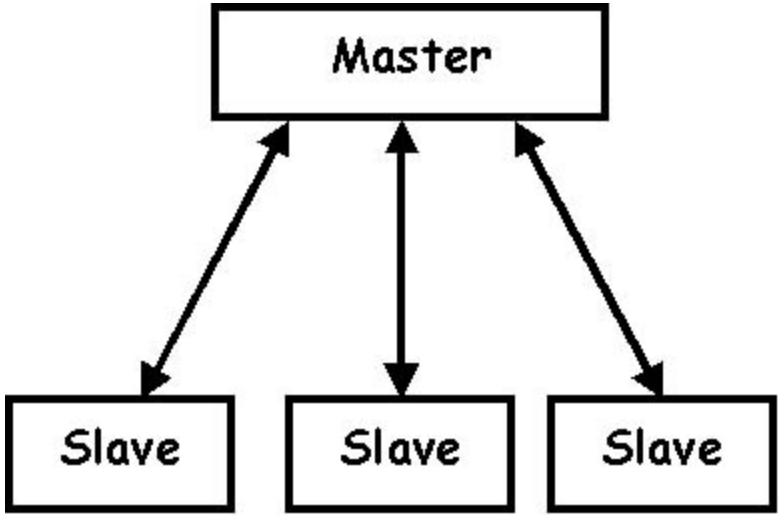
Παράδειγμα

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

  void get(Request req, Response res) {
    Template t = loadTemplate('items') //view
    List<Item> items = store.loadItems() //model
    Context ctx = new Context()
    ctx.put("items", items)
    t.render(res.getWriter(), ctx)
  }

}
```

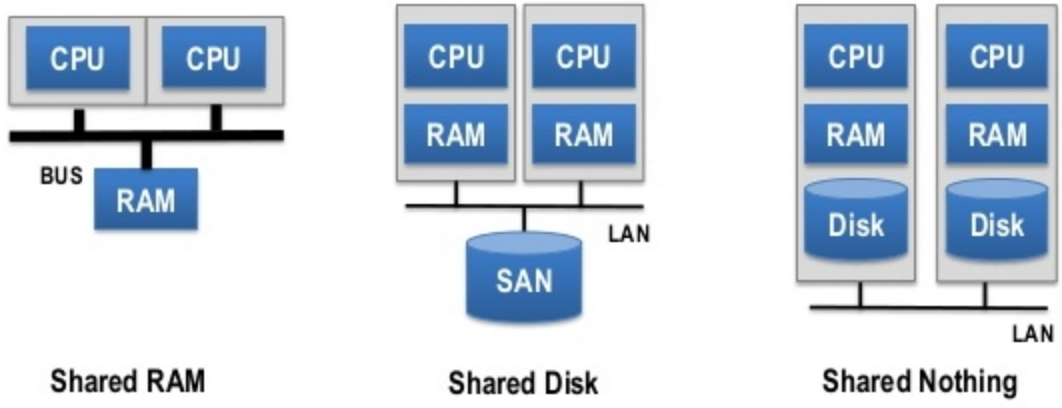
Master-Slave / Master-Replica



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



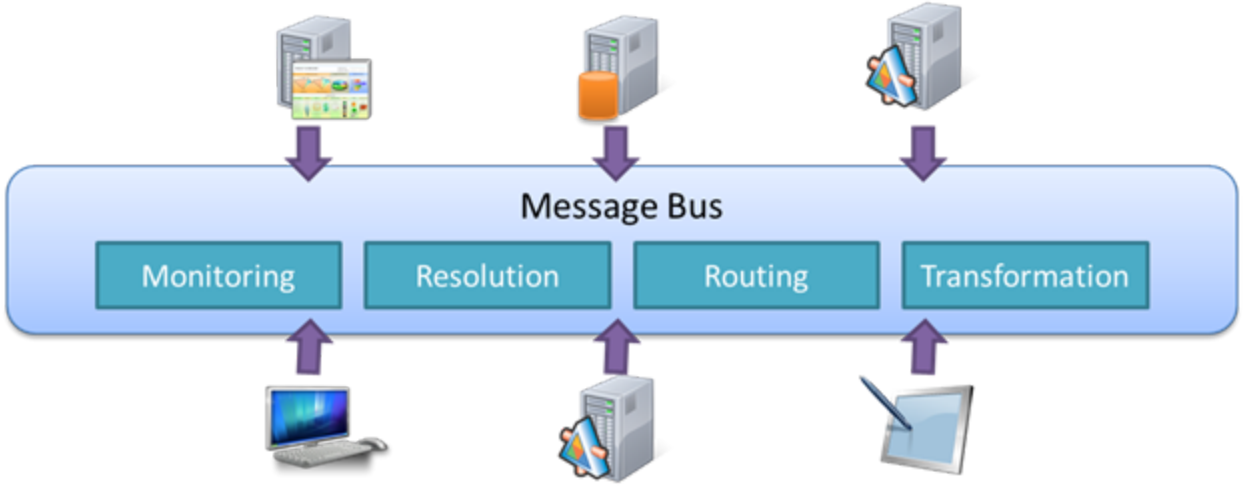
Χαρακτηριστικά

- 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 αύξηση διαθεσιμότητης & μείωση καθυστέρησης αντί για συνέπεια)

Message-driven/Publish-subscribe



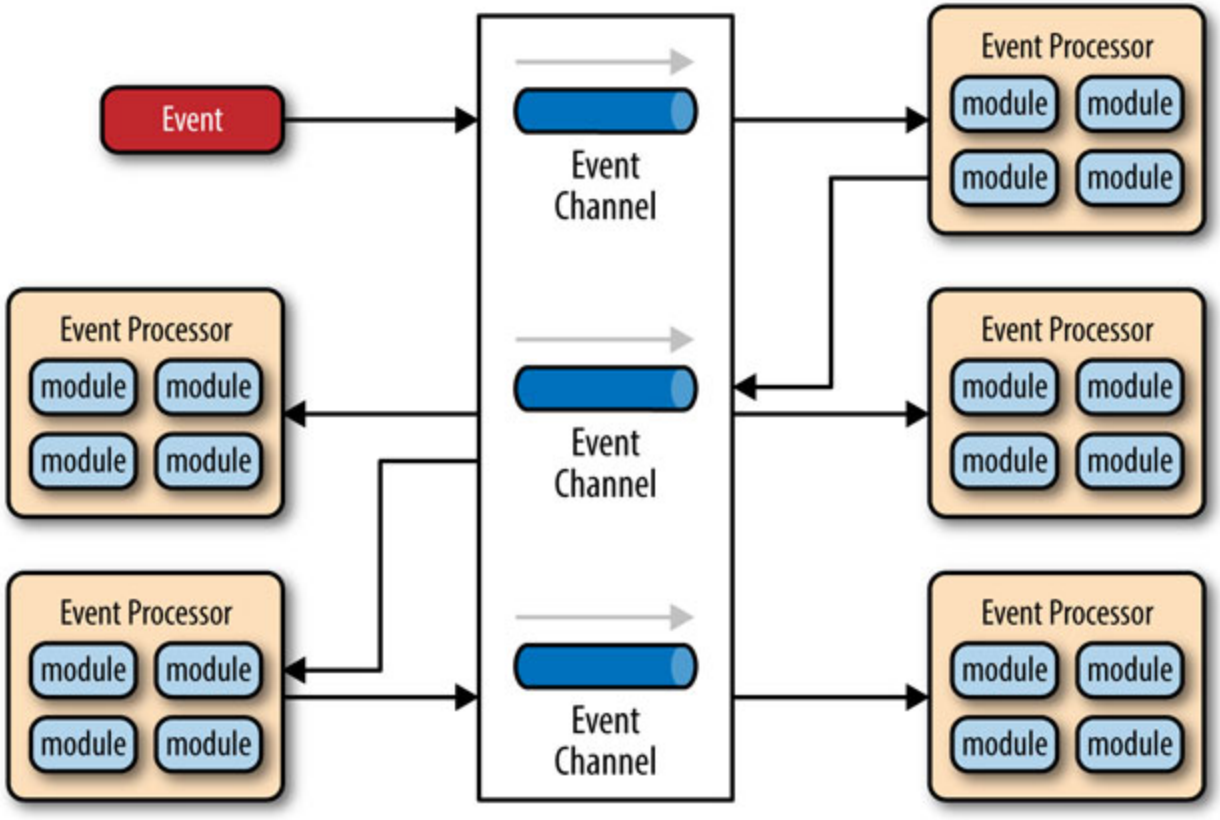
Χαρακτηριστικά

- 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)

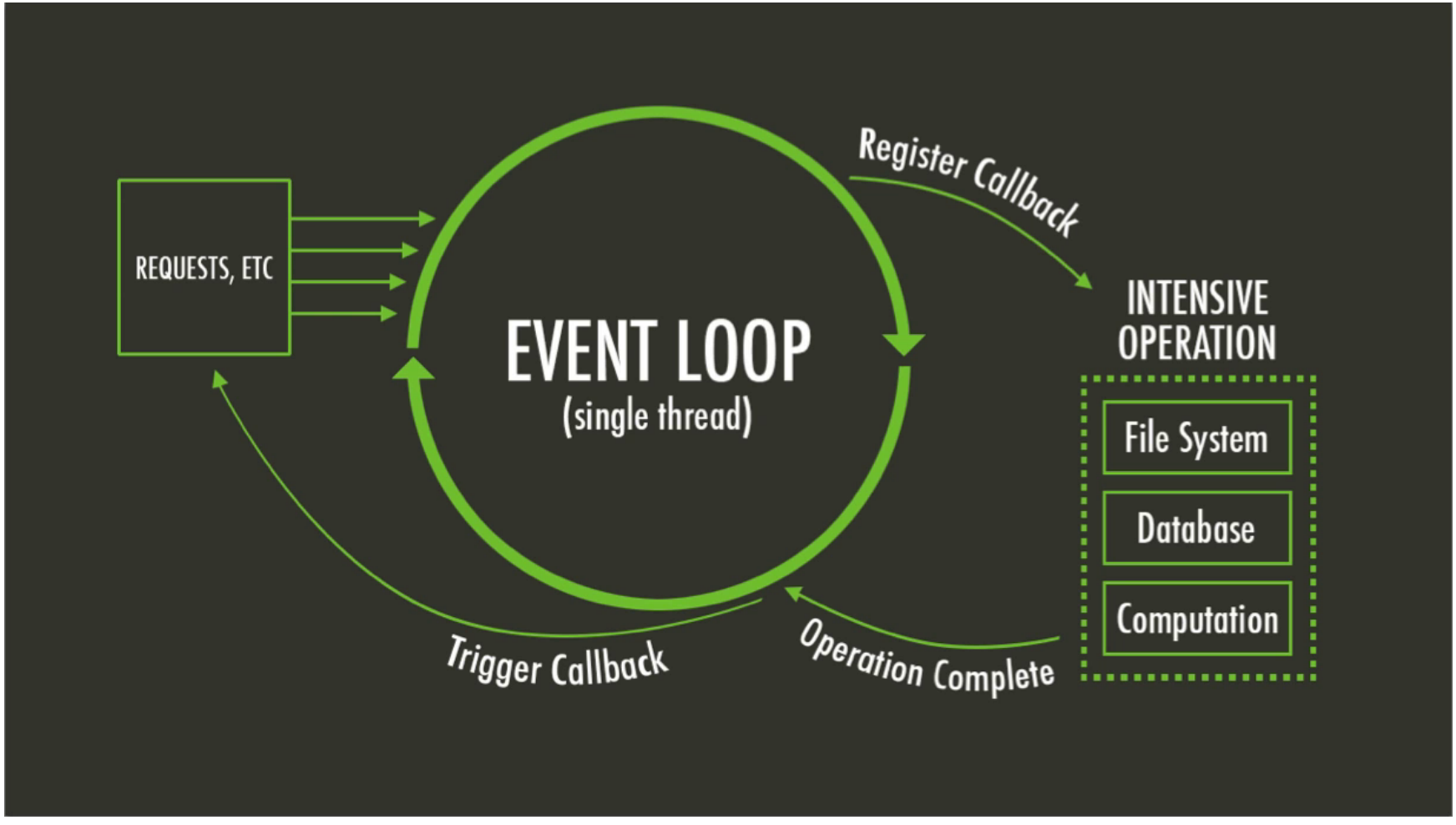
```
class EventEmitter {
  constructor() {
    this.events = new Map(); //Map<Event, Set<Listener>>
  }
  on(event, listener) {
    let listeners = this.events.get(event);
    if (!listeners) {
      listeners = new Set();
      this.events.set(event, listeners);
    }
    listeners.add(listener);
 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

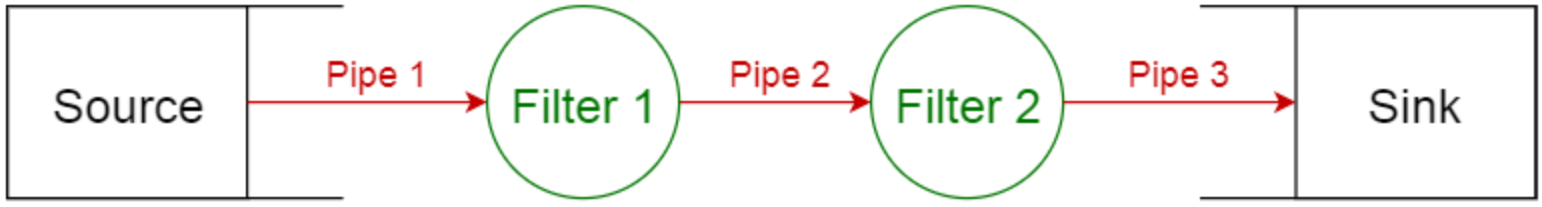
Παράδειγμα (Java)

```
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
```