



Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Τμήμα Πληροφορικής & Τηλεπικοινωνιών

Προηγμένες Μέθοδοι Προγραμματισμού

ΠΜΣ (M135.CS1E, M135.CS23B, M135.IC1E, παλαιό: M117)

Πρότυπα σχεδίασης (1)

Δρ. Κώστας Σαΐδης (saiko@di.uoa.gr)

Πρότυπο σχεδίασης λογισμικού (Software design pattern)

- Ένας τρόπος δόμησης/διάρθρωσης του κώδικα, ώστε να επιτευχθεί μια -αποδεκτή ως- βέλτιστη επίλυση ενός σχεδιαστικού προβλήματος.
- Βέλτιστη: καθώς επεκτείνεται το λογισμικό, δεν θα χρειαστεί να επισκεφθούμε ξανά τη λύση (refine/revisit), αυτή πλέον θα αποτελεί "σταθερά" του σχεδιασμού μας.
- Η λύση παρέχει κάποιο "ποιοτικό" όφελος (μειώνει το τεχνικό χρέος): level of abstraction, separation of concerns, code reuse, κλπ.

Προσοχή: η υπερβολική χρήση των προτύπων μπορεί να κάνει τον κώδικο πιο δυσνόητο.

Λίστα αναγνωσμάτων

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides,
"Design Patterns: Elements of Reusable Object-Oriented
Software", Addison Wesley, 1994, 0-201-63361-2.

Παράδειγμα σχεδιαστικού προτύπου

Visitor

- Διαχωρισμός του αλγορίθμου "διάσχισης" από τη δομή δεδομένων.
- Η "λογική" της επίσκεψης στα στοιχεία της δομής (traversal) διαχωρίζεται από τη δομή αυτή καθ' αυτή.

Παράδειγμα

```
interface TreeVisitor<V> {  
    void visit(TreeNode<V> node);  
}  
  
abstract class TreeNode<V> {  
    TreeNode<V> getLeft();  
    TreeNode<V> getRight();  
    V getValue();  
    public void accept(TreeVisitor<V> visitor) {  
        visitor.visit(this);  
    }  
}  
  
interface TreeTraversal<V> {  
    void traverse(TreeNode<V> node);  
}
```

Υλοποίηση

```
class PreOrderTreeTraversal<V> implements TreeTraversal<V> {  
    private final TreeVisitor<V> visitor;  
  
    public PreOrderTreeTraversal(TreeVisitor<V> visitor) {  
        this.visitor = visitor;  
    }  
  
    public void traverse(TreeNode<V> node) {  
        //visit the node  
        node.accept(visitor);  
  
        //then traverse from left to right (pre-order)  
        TreeNode<V> left = node.getLeft();  
        if (left != null) traverse(left);  
  
        TreeNode<V> right = node.getRight();  
        if (right != null) traverse(right);  
    }  
}
```

Πραγματικό Παράδειγμα (Java 7+ FileVisitor)

java.nio.Files

```
static Path walkFileTree(  
    Path start,  
    FileVisitor<? super Path> visitor  
) throws IOException
```


java.nio.file.FileVisitor

```
interface FileVisitor<T> {  
    FileVisitResult postVisitDirectory(T dir, ...)  
    FileVisitResult preVisitDirectory(T dir, ...)  
    FileVisitResult visitFile(T file, ...)  
    FileVisitResult visitFileFailed(T file, ...)  
}
```

Κύρια είδη σχεδιαστικών προτύπων

- Αρχιτεκτονικά (architectural)
 - MVC, Multi-tier, κ.ά
- Κατασκευαστικά (creational)
 - Factory, Builder, Singleton, κ.ά
- Δομικά (structural)
 - Adapter/Wrapper, Bridge, Decorator, Proxy, κ.ά
- Συμπεριφοράς (behavioral)
 - Visitor, Chain of responsibility, Command, Mediator, κ.ά

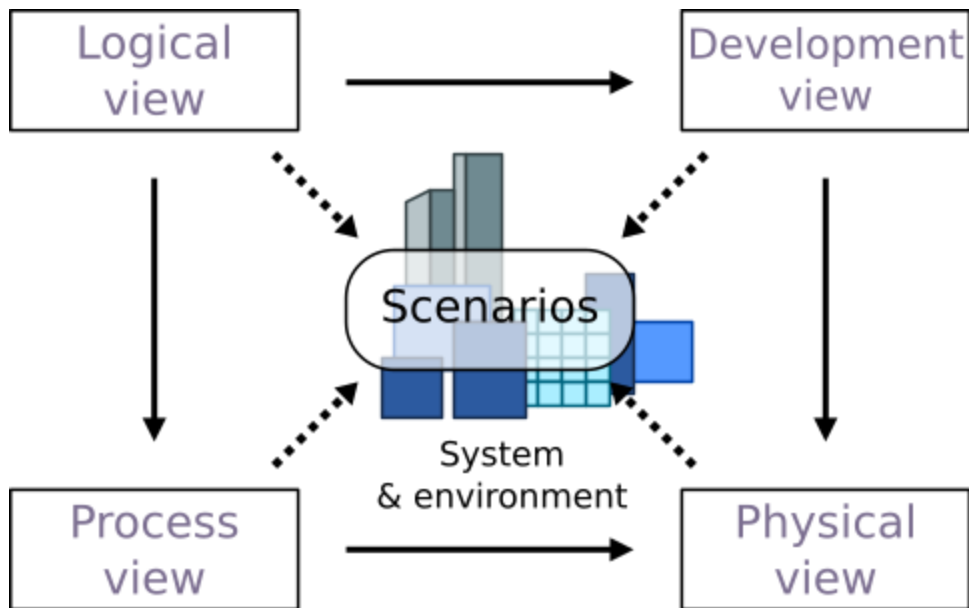
Αρχιτεκτονικά πρότυπα

- Ασχολούνται κυρίως με τη διάρθρωση του κώδικα σε υψηλό αρχιτεκτονικό επίπεδο (κόμβοι, υποσυστήματα, συστατικά, επικοινωνία μεταξύ τους).

Κατασκευαστικά, δομικά και πρότυπα συμπεριφοράς

- Ασχολούνται κυρίως με τη διάρθρωση του κώδικα στο χαμηλό επίπεδο των κλάσεων (κατασκευή, δομή και συμπεριφορά αντικειμένων).
- Θα τα δούμε σε επόμενες διαλέξεις.

Πολλές αρχιτεκτονικές οπτικές (4+1)



By mpan - Based on File:4+1 Architectural View Model.jpg by User:Mdd, CC BY-SA 3.0,
<https://commons.wikimedia.org/w/index.php?curid=50144028>

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.

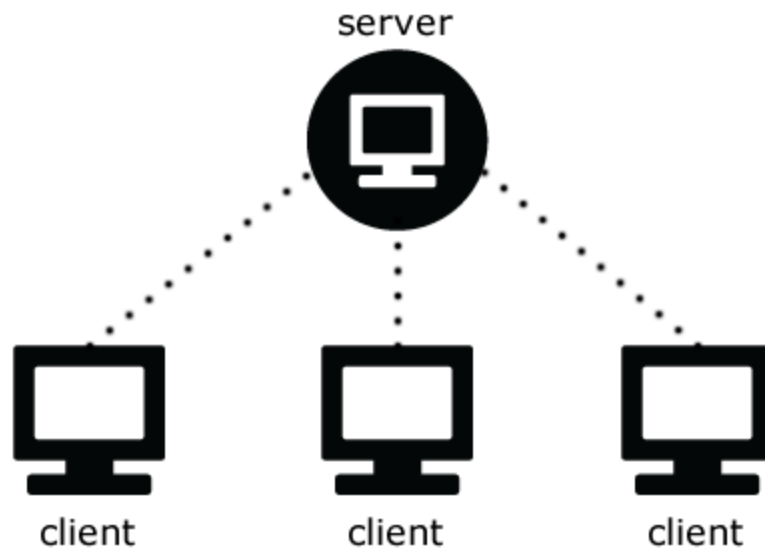
Αρχιτεκτονικά πρότυπα (architectural patterns)

Γενικές κι επαναχρησιμοποιήσιμες λύσεις σε κοινά προβλήματα αρχιτεκτονικής σχεδίασης λογισμικού.

Αρχιτεκτονικά πρότυπα (κάποια)

- Client-Server
- Component-based
- Event-Driven
- Layered / N-tier
- Message-driven/Publish-subscribe
- Model-View-Controller (MVC)
- Peer-to-peer (P2P)
- Pipeline / Pipe-filter

Client-Server



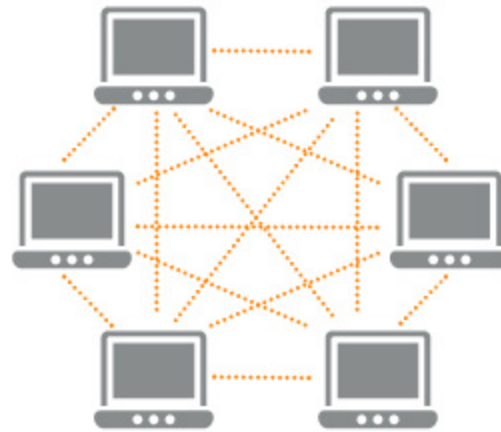
Χαρακτηριστικά

- 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
- Έμφαση: physical & development view

Peer-to-peer (P2P)



Server-Based

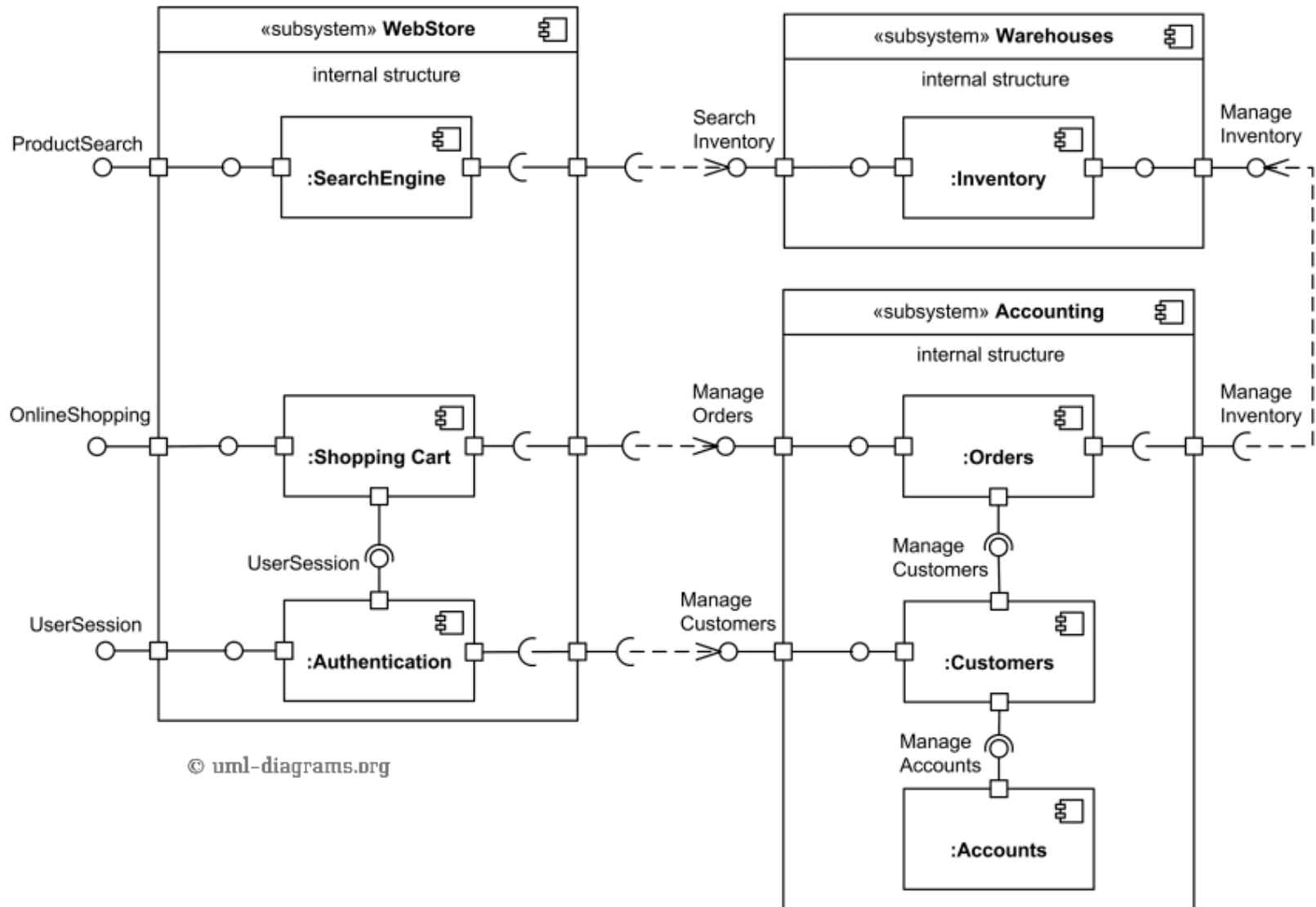


P2P

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
- Έμφαση: physical & development view

Component-based



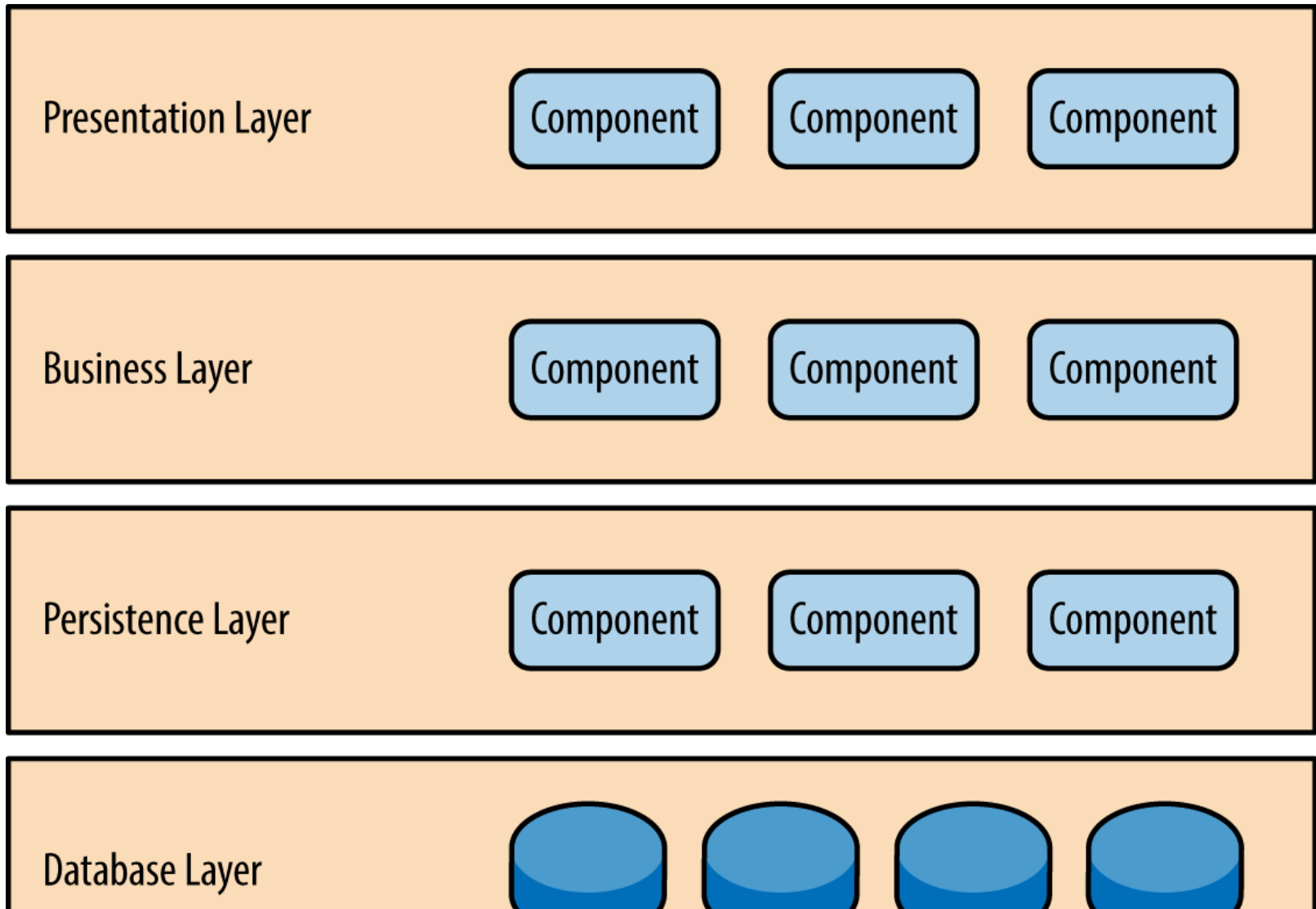
Χαρακτηριστικά

- 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
- Έμφαση: logical & development view

Κανόνας

Πάντα ξεκινάμε το σχεδιασμό του λογισμικού από τα Interfaces

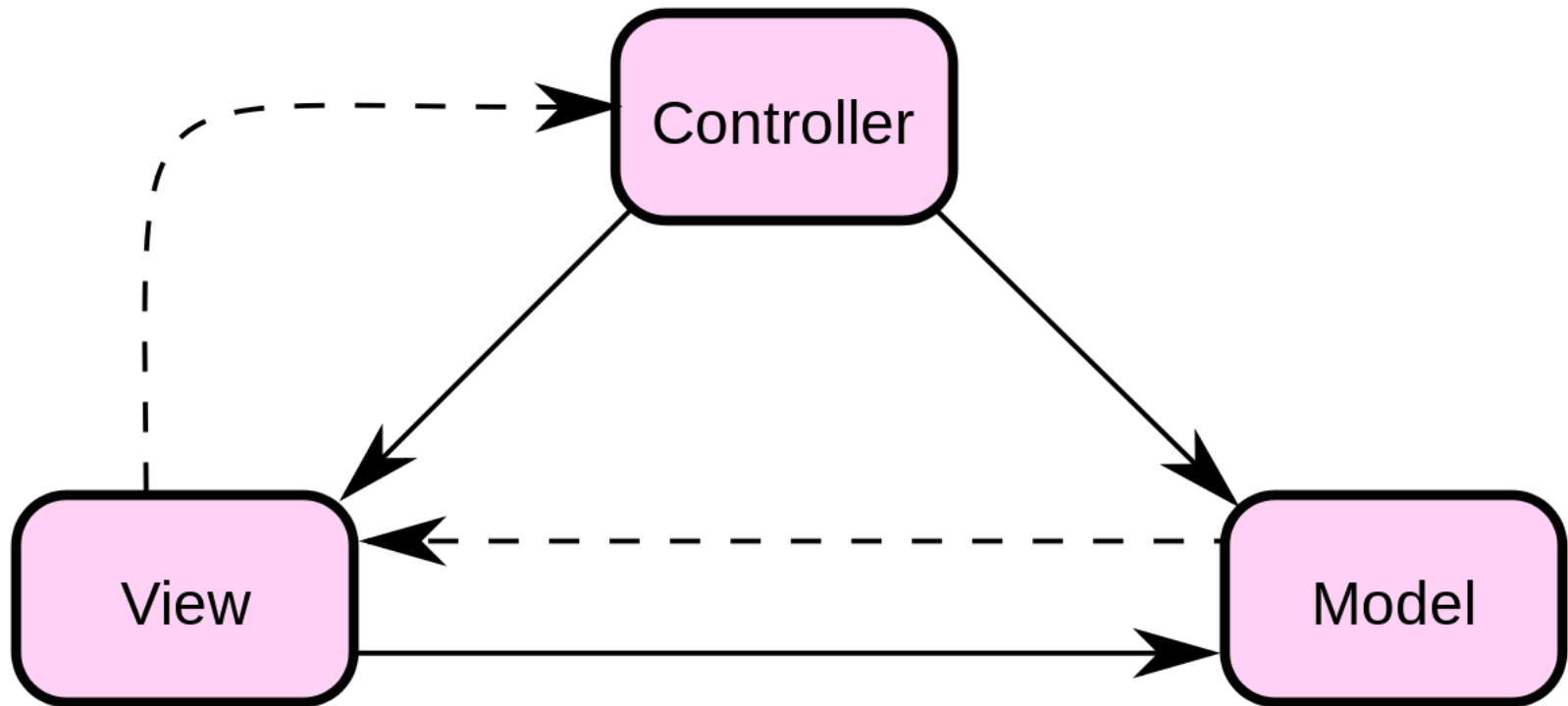
Layered/N-tier



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

- Επιμερισμός αρμοδιοτήτων (separation of concerns)
- 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) {
        //process the input request
        Filter filter = readFilter(req)

        //load the template (view)
        Template t = loadTemplate('items')

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

        //create a context (bindings)
        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)

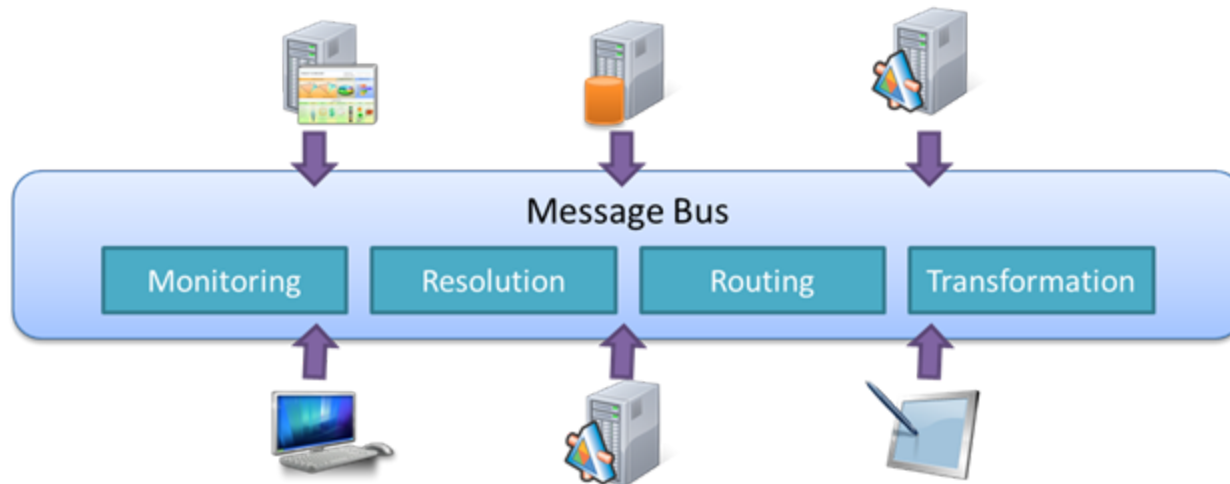
```
<table>
  { for item in items }
    <tr>
      <td> { item.title } </td>
      <td> { item.description } </td>
    </tr>
  { endfor }
</table>
```

MVC

Ευρεία χρήση στις εφαρμογές διαδικτύου

Πολλά (πάρα πολλά) frameworks

Message-driven/Publish-subscribe



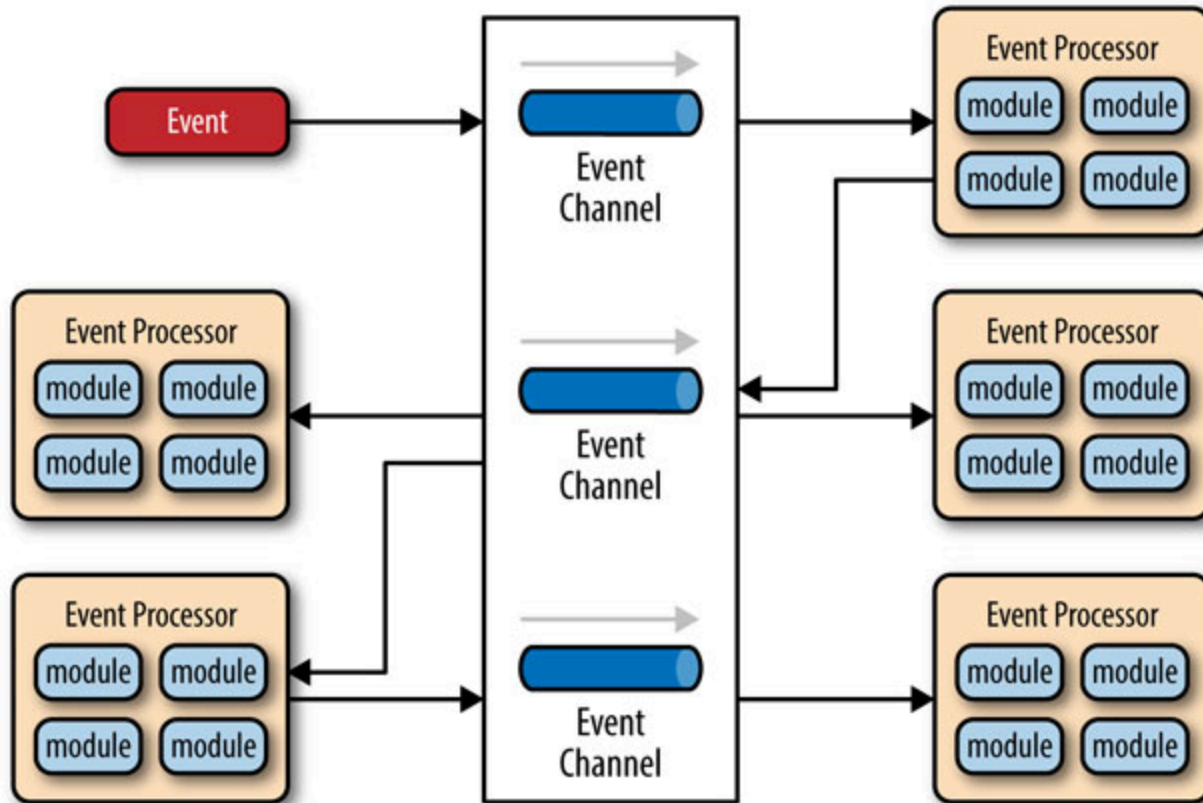
Χαρακτηριστικά

- Χαλαρή σύνδεση (loose coupling) μεταξύ συστατικών/εφαρμογών
- Publisher (producer): αποστολή μηνυμάτων
- Subscriber (consumer): λήψη μηνυμάτων
- Topics (channels): "κλάσεις/θέματα" μηνυμάτων
- Message Bus (broker): διαχείριση/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 (Typescript)

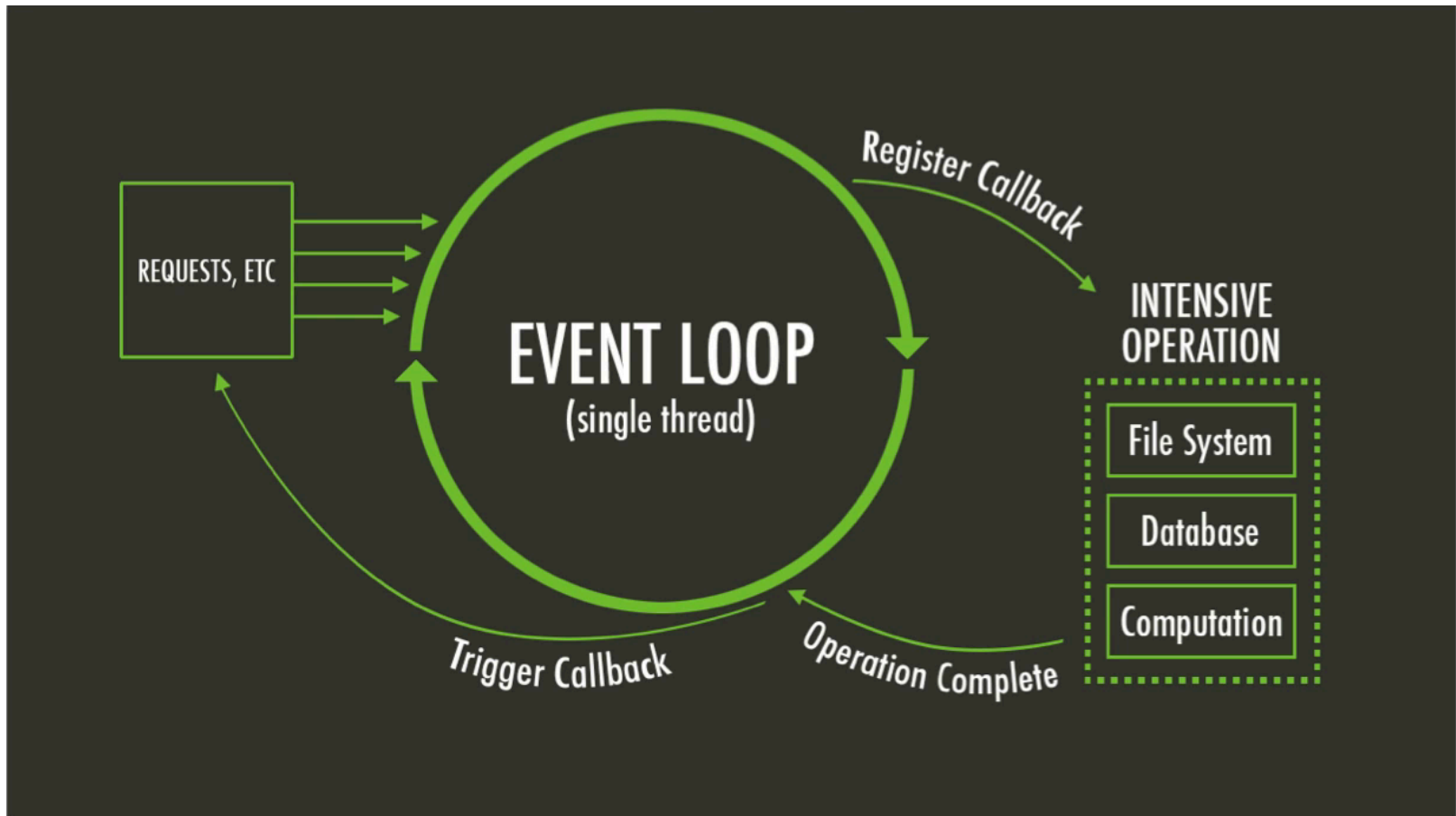
```
interface Listener {  
    listen(event:string, args:Array<any>):void;  
}  
class EventEmitter {  
    events: Map<string, Set<Listener>>;  
    constructor() {  
        this.events = new Map<string, Set<Listener>>();  
    }  
    on(event:string, listener:Listener) {  
        let listeners = this.events.get(event);  
        if (!listeners) {  
            listeners = new Set<Listener>();  
            this.events.set(event, listeners);  
        }  
        listeners.add(listener);  
        return this;  
    }  
    emit(event:string, args:Array<any>) {  
        const listeners = this.events.get(event);  
        if (listeners) {  
            for (let listener of listeners) {  
                listener.listen(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 & pipelines).
- 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
```

Παραλληλία (χρήση πολλών threads)

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

Δεν είναι πάντα τόσο απλό, ούτε θα είναι πάντα πιο αποδοτικό (γιατί και πώς, σε επόμενη διάλεξη).