

Architetture per applicazioni data-centric

Insegnamento di “Basi di Dati”

A.A. 2004/2005

Docente: Dott. Luca Pretto

Relatore: Dott. Lucio Benfante

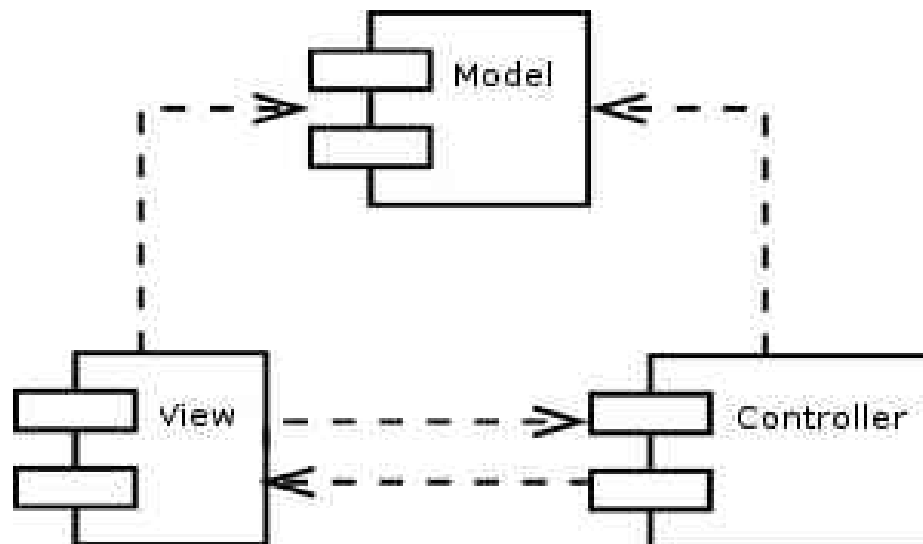
Suddivisione logica di un'applicazione

- Presentation logic: parte dell'applicazione che si occupa di determinare ciò che viene mostrato all'utente e di gestire l'input proveniente da esso (digitazione di testo, azioni col mouse, ecc.)
- Business logic: contiene le regole di “business” dell'applicazione, cioè come essa si comporta e le funzionalità che mette a disposizione
- Data management: gestisce i dati necessari all'applicazione. In un'applicazione data-centric queste funzionalità sono tipicamente fornite da un DBMS
- Servizi di infrastruttura: non fanno propriamente parte dell'applicazione, ma forniscono funzionalità aggiuntive alle sue parti (gestione di code di messaggi, supporto alle transazioni, elaborazione distribuita, ecc.)

Il pattern MVC

Per implementare un'applicazione in cui le varie parti funzionali siano suddivise correttamente, molto spesso si utilizza il pattern MVC (Model-View-Controller), che prevede i seguenti tre componenti:

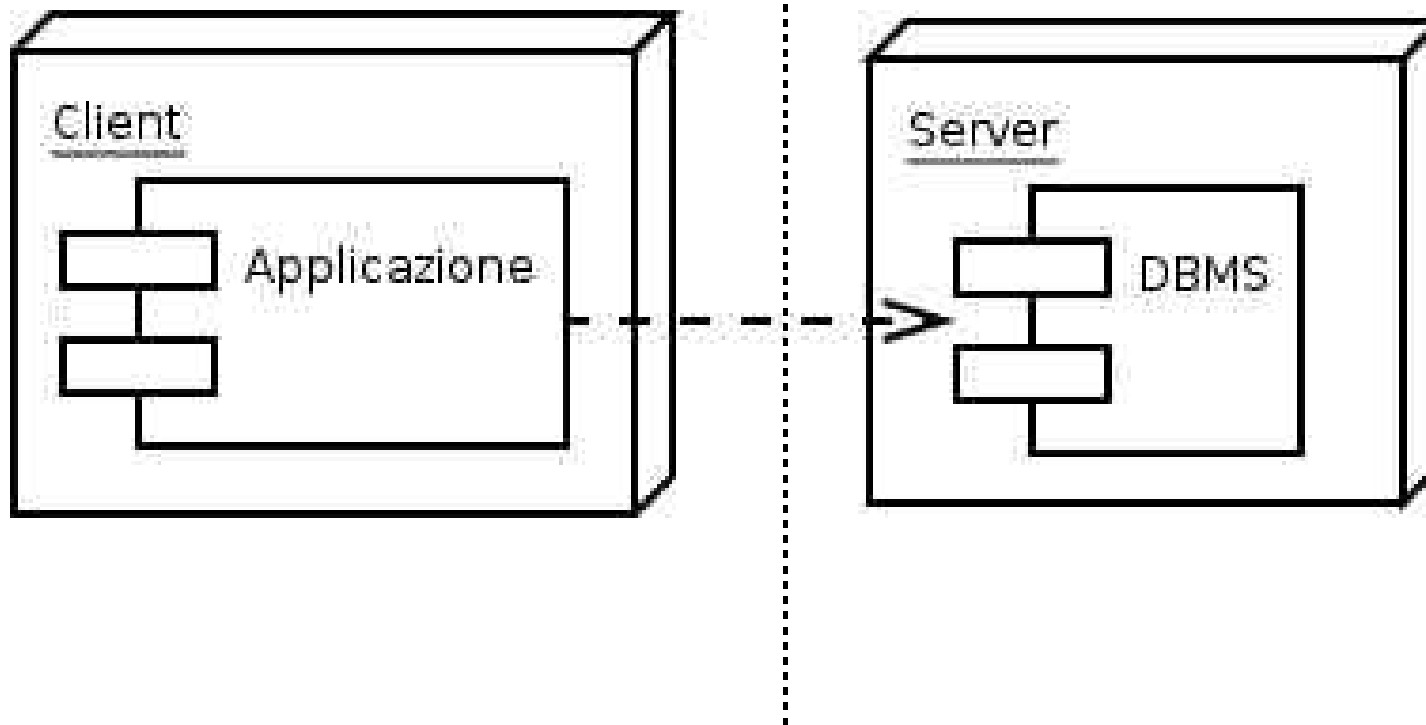
- Model: definisce e gestisce i dati su cui lavora l'applicazione
- View: si occupa della presentazione dei dati
- Controller: si occupa del flusso dell'applicazione



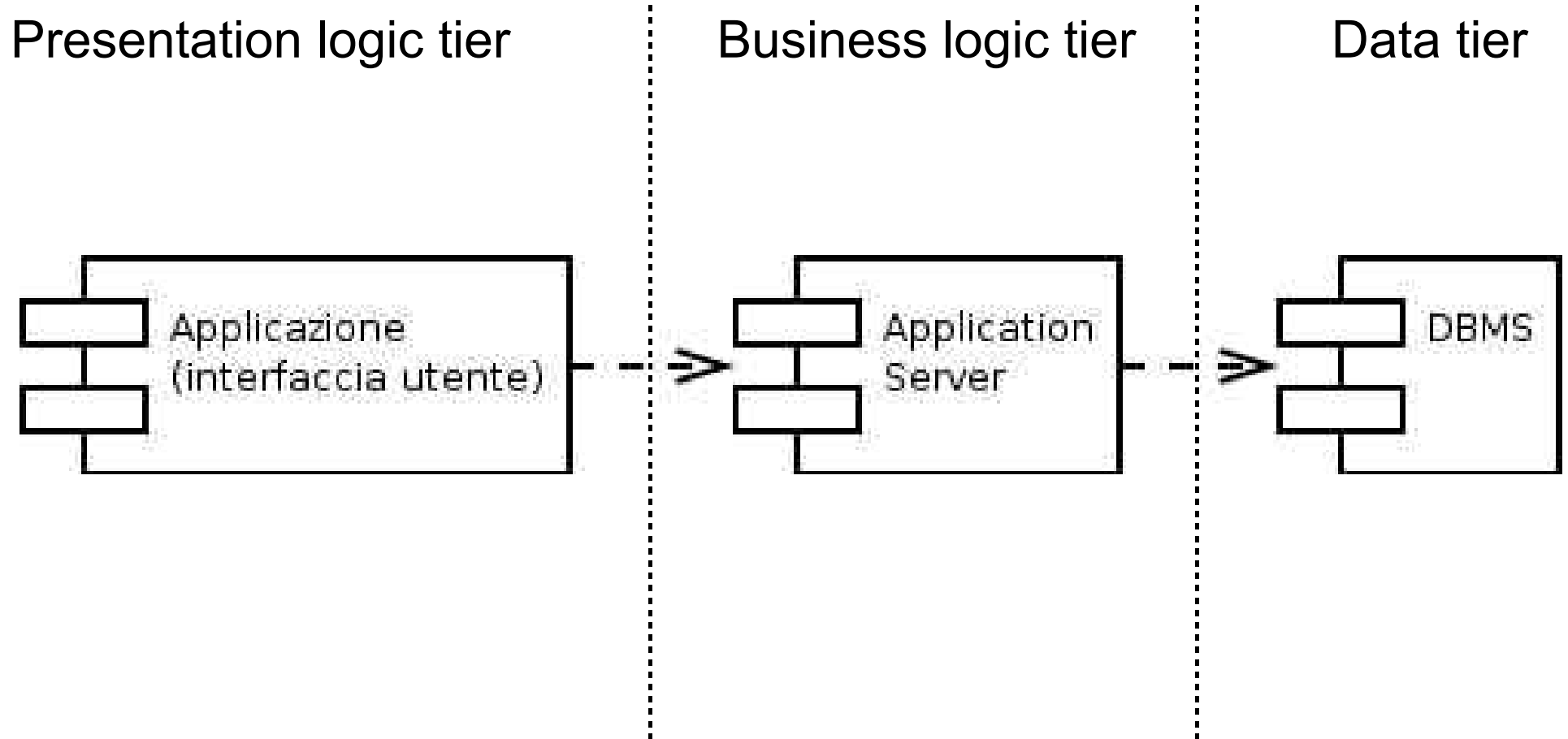
Applicazioni 2-tier

Presentation logic/Business logic tier

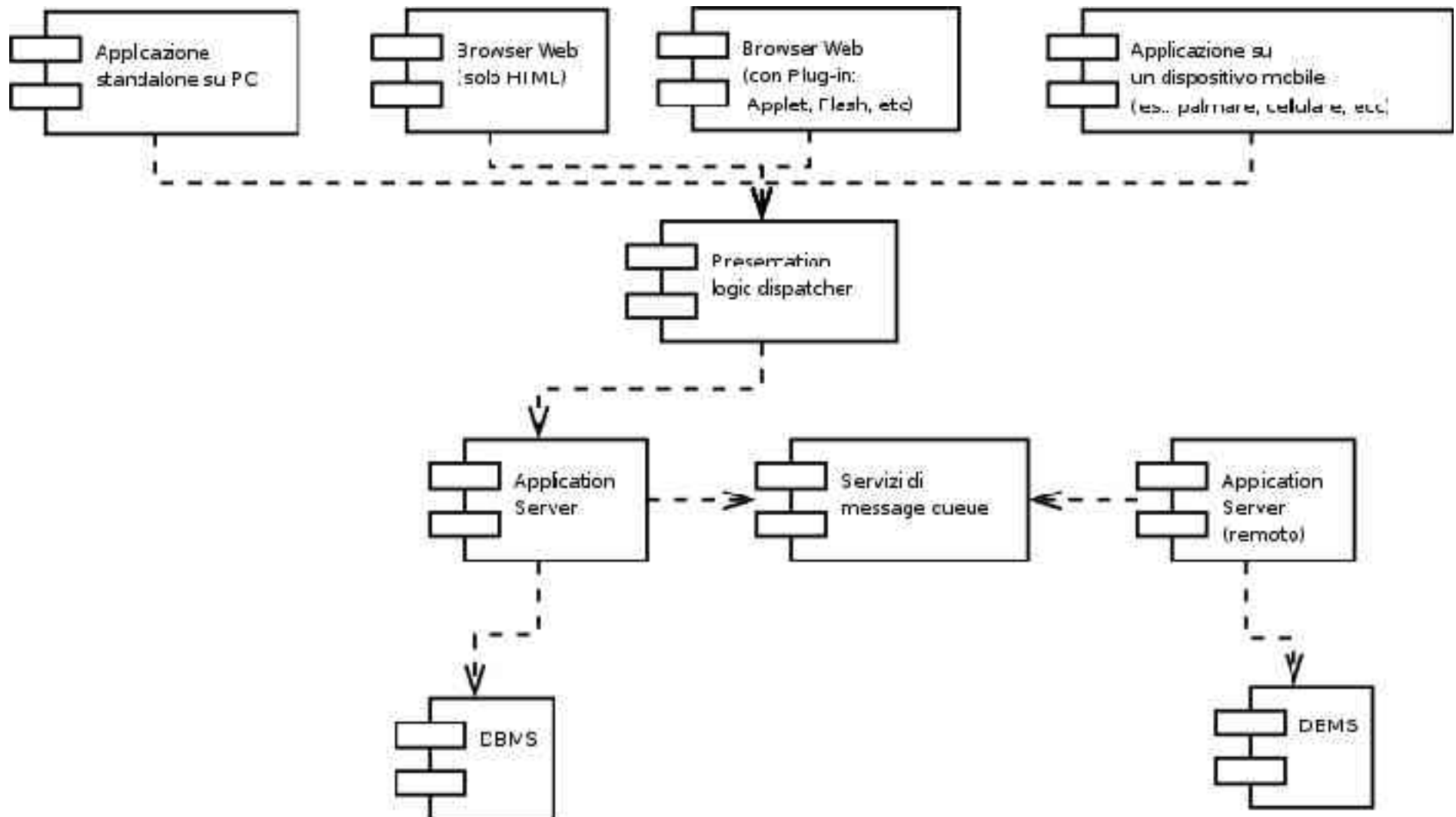
Data/Business logic tier



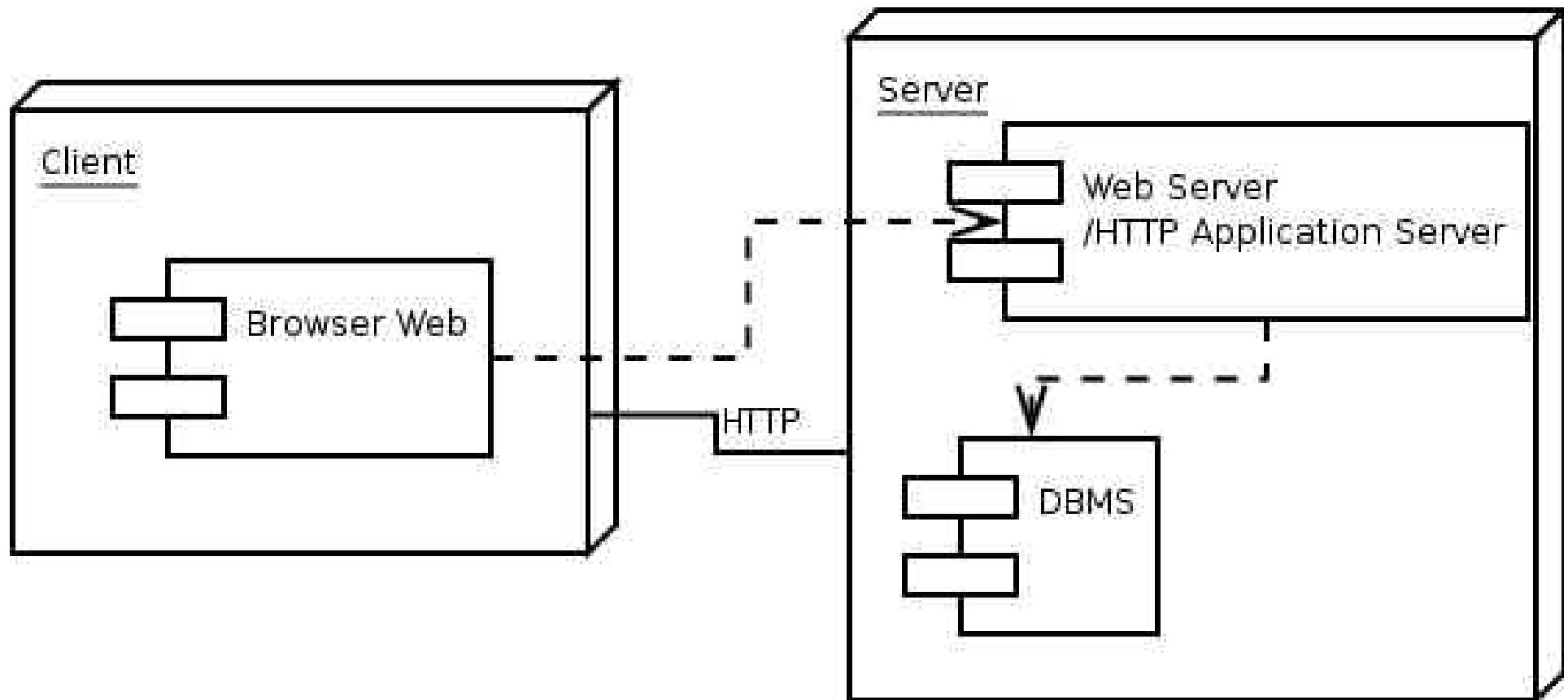
Applicazioni 3-tier



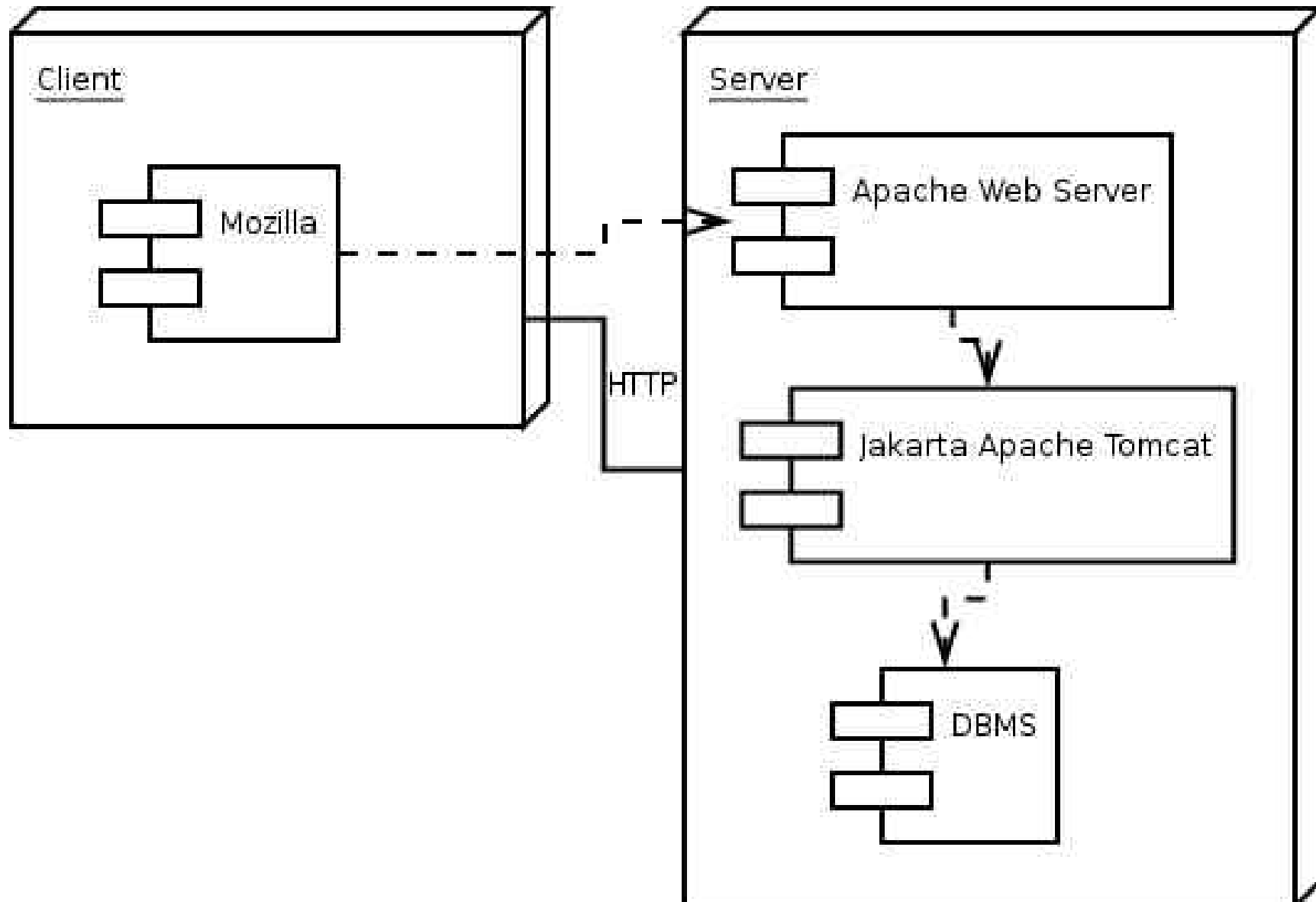
Applicazioni n-tier



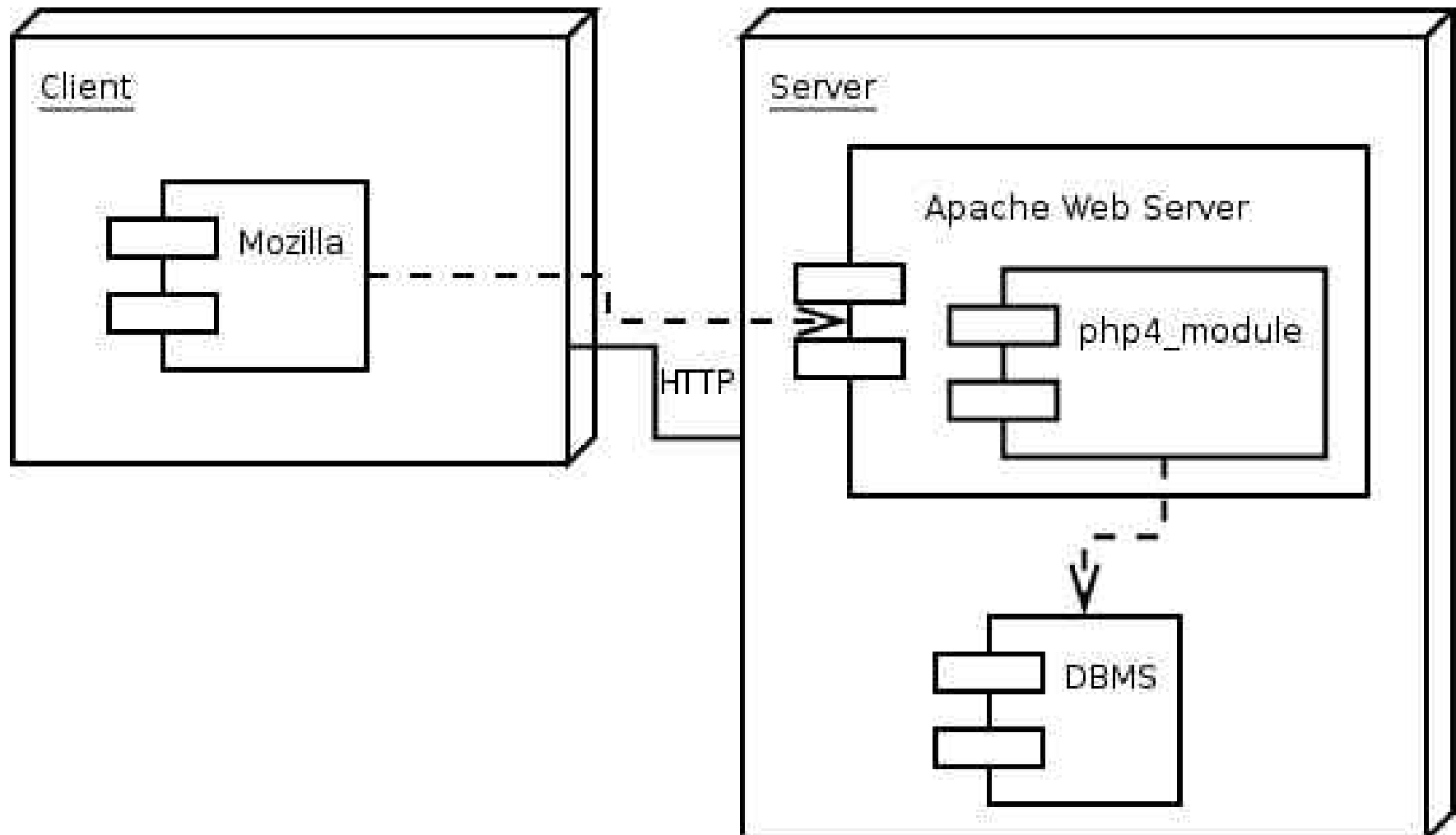
Applicazioni Web



Java Web Application



PHP Web Application



Tecnologie per sviluppare applicazioni Web

- CGI (usando vari linguaggi)
- Servlet
- Scripting server-side
 - ASP
 - JSP
 - PHP
- Estensioni client-side
 - Java Applet
 - Flash
 - (Java Web Start)

Scripting server-side (PHP)

- All'interno della pagina HTML vengono inseriti dei tag che vengono elaborati e sostituiti dal server prima che la pagina venga trasmessa al client.
- In PHP4 tali tag possono assumere una delle seguenti forme:
 - `<?php ...codice php... ?>`
 - `<? ...codice php... ?>`
 - `<script language="php"> ...codice php... </script>`
 - `<% ...codice php... %>`
- Dato che la pagina viene prima elaborata dal server, al client non arriverà mai il codice php, ma del semplice html

Esempio: ottenere l'indirizzo IP del client

showip.php

```
<html>
<head>
<title>PHP DEMO: CLIENT IP ADDRESS</title>
<body>
Ciao, il tuo indirizzo IP &grave;
<?php echo $_SERVER["REMOTE_ADDR"] ?><br>
<br>
(prima di PHP 4.1.0)<br>
Ciao, il tuo indirizzo IP &grave;
<?php echo $HTTP_SERVER_VARS["REMOTE_ADDR"] ?>
</body>
</html>
```



Esempio: esaminare la configurazione

info.php

```
<?php phpinfo(); ?>
```



Problemi comuni in un'applicazione Web

- Mantenimento dello stato
- Accesso ai database
- Validazione dei dati provenienti dal client
- SQL injection

Mantenere lo stato in una applicazione Web

- Memorizzazione **lato client**
 - Cookie
 - Campi nascosti in una form
 - Scrittura negli URL
- Memorizzazione **lato server**, con la tecnologia più adeguata per l'applicazione: DBMS, files, in memoria, ecc.
 - Il problema è mantenere l'associazione dello stato con l'utente.
 - Normalmente si assegna all'utente un identificatore di sessione che viene scambiato con una delle tecniche di memorizzazione lato client.
 - Di solito le varie tecnologie mettono a disposizione delle librerie che nascondono i dettagli implementativi del mantenimento della sessione.

PHP: memorizzazione di una variabile nella sessione

```
session_start();  
$_SESSION['nome_utente'] = $nome_from_form;
```

Esempio

countpage.php

```
<?php  
session_start();  
if (!isset($_SESSION['count'])) {  
    $_SESSION['count'] = 0;  
} else {  
    $_SESSION['count']++;  
}  
?>
```

<!-- HTML output -->

Ciao, hai visitato questa pagina
<?= \$_SESSION['count'] ?> volte



Accedere ad un database PostgreSQL

dbdemo.php

```
<?php $database =
    pg_connect ("host=localhost dbname=igsite user=lucio password=vxvdf");
$result = pg_query ($database, "SELECT * FROM exercise order by id;");
if (!$result) {
    echo "Errore interrogando il database.\n";
    exit;
}
if (pg_num_rows($result) != 0) { ?>
<table border="1">
<?php
    for ($i = 0; $i<pg_num_rows($result); $i++) {
        $row = pg_fetch_array($result); ?>
<tr>
    <td valign="top"><?= $row["id"] ?></td>
    <td valign="top"><?= $row["start_ex"] ?></td>
    <td valign="top"><?= $row["end_ex"] ?></td>
    <td valign="top"><?= $row["title"] ?></td>
    <td valign="top"><?= $row["description"] ?></td>
</tr>
<?php    } ?>
</table>
<?php }
pg_close($database); ?>
```



Indipendenza dal DBMS

dbxdemo.php

```
<?php $database =
        dbx_connect (DBX_PGSQL, "localhost", "igsite", "lucio", "vxvdf");
$result = dbx_query ($database, "SELECT * FROM exercise order by id;");
if (!is_object($result)) {
    echo "Errore interrogando il database, o nessun risultato.\n";
    exit;
} ?>
Recuperati <?= $result->rows ?> record(s).<br>
<table border="1">
<?php
foreach ($result->data as $row) { ?>
    <tr>
        <td valign="top"><?= $row["id"] ?></td>
        <td valign="top"><?= $row["start_ex"] ?></td>
        <td valign="top"><?= $row["end_ex"] ?></td>
        <td valign="top"><?= $row["title"] ?></td>
        <td valign="top"><?= $row["description"] ?></td>
    </tr>
<?php } ?>
</table>
<?php dbx_close($database); ?>
```

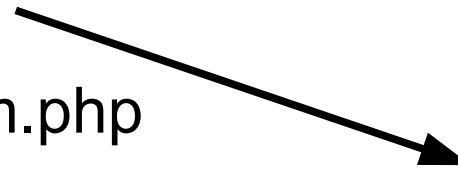
Validazione di ciò che arriva dal client

- In generale il client **deve essere considerato ostile**
- Ciò che proviene dal client quindi deve essere sempre verificato **dal server**
- Ciò che deve essere verificato è:
 - il valore dei campi di una form
 - il valore dei parametri dell'URL
 - il valore dei cookie
- Possono essere eseguite delle **verifiche client-side**, ad esempio implementate con del codice JavaScript, ma tali verifiche **non sono sufficienti** per garantire il buon funzionamento del server: devono essere sempre ripetute anche dal server.

Un form di esempio



simpleform.php



formresult.php

simpleform.php mostra la form e ne valida i campi. Nel caso in cui la validazione dei campi fallisca, rimostra la form, inserendo come valori dei campi quelli precedentemente digitati dall'utente.

Se la validazione viene eseguita con successo, mostra il risultato, prodotto dal file formresult.php.



SQL Injection

- Così come occorre sempre validare i dati provenienti dal client, a maggior ragione occorre assicurarsi che l'input proveniente dal client non modifichi il nostro codice SQL, facendogli eseguire operazioni diverse da quelle che avevamo previsto.
- Esempio: si supponga di avere una form in cui chiediamo all'utente il suo username e la sua password, e di usare autenticare l'utente usando uno statement SQL costruito in questo modo:

```
"SELECT nome, cognome FROM utenti WHERE username='" .  
$_REQUEST['username'] . "' AND password='" . $_REQUEST  
['password'] . '"
```

Cosa accade se l'utente scrive come username la seguente stringa (e come password qualunque cosa):

```
hi' or 1=1 --
```

Il risultato e' la seguente istruzione:

```
SELECT nome, cognome FROM utenti WHERE username='hi' or 1=1  
--' AND PASSWORD='hdjskuire'
```

Evitare SQL Injection

- Filtrare o sostituire come sequenze di escape i seguenti caratteri: apici singoli e doppi, slash e back-slash, punti e virgola, caratteri estesi e di controllo (NULL, CR, NL, ecc.). Deve essere fatto su tutto l'input dell'utente, sui parametri dell'URL e sul valore dei cookie.
- Convertire o verificare i valori numerici prima di usarli
- Se il DBMS definisce nel database delle stored procedure potenzialmente a rischio, ma che non si intende usare, eliminarle

Se si ha la possibilità di configurare il DBMS:

- Eseguire il DBMS al più basso livello di privilegi possibile (sicuramente non come root)

Bibliografia

- T. Ratschiller e T. Gerken, “PHP 4.0 – Applicazioni Web”, Addison-Wesley, 2001
- PHP Manual: <http://www.php.net/manual/it/index.php>
- Introduction to PHP - Stig Sæther Bakken with Zend staff
<http://www.zend.com/zend/art/intro.php>
- SQL Injection Walkthrough
<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>
- Jason Hunter e William Crawford, “Java Servlet Programming”, Second edition, O'Reilly. 2001
- Hans Bergsten, “Java Server Pages”, O'Reilly-Hops Libri, 2001
- E. Gamma, R. Helm, R. Johnson e J. Vlissides, “Design Patterns – Elements of Reusable Object-Oriented Software”, Addison-Wesley, 1995
- J. Rumbaugh, I. Jacobson e G. Booch, “The Unified Modeling Language Reference Manual”, Addison-Wesley, 1999
- E.J. Naiburg e R.A. Maksimchuk, “UML for Database Design”, Addison-Wesley, 2001