# VSWA-Ausarbeitung

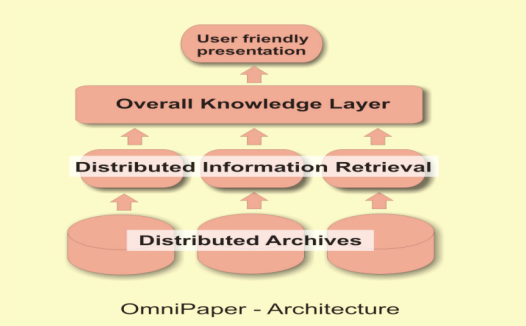
## Software Architektur

- Software Architektur stellt ein Grundgerüst zur Unterstützung von Design und Entwicklung von komplexen Systemen dar
- Software Architektur bildet eine Basis für aktuelle und künftige Adaptionen und Erweiterungen
- Software Architektur beinhaltet Schnittstellendefinitionen für die Zusammenarbeit aller beteiligten und künftigen Komponenten eines Systems

- Je umfangreicher Software wird, umso mehr Bedeutung gewinnt die Gesamtarchitektur gegenüber einzelnen Algorithmen und Datenstrukturen
- Darstellbar in Systemdiagrammen
  - Komponenten
  - Konnektoren
  - Bedingungen/Einschränkungen
  - Topologie/Hierarchie

## Systemdiagramme

- Komponenten (components)
  - Programme oder Prozesse, organisatorische oder technische Einheiten, Objekte, Funktionen, Tasks, Prozesse?
- Konnektoren (connectors)
  - Kommunikation, Synchronisation, Procedure Calls,…




- Wozu beschreibt man Software-Architektur?
  - Kommunikationsbasis für Software-Development (Design, Development, Testing, Reviews)
  - Dokumentation von einmal gesetzten Design-Entscheidungen
  - Modell des Gesamtsystems
    - Re-Analyse
    - Wiederverwendbarkeit
  - Dokumentation für alle Beteiligten
    - Entwickler, Auftraggeber, Kunden, Administratoren

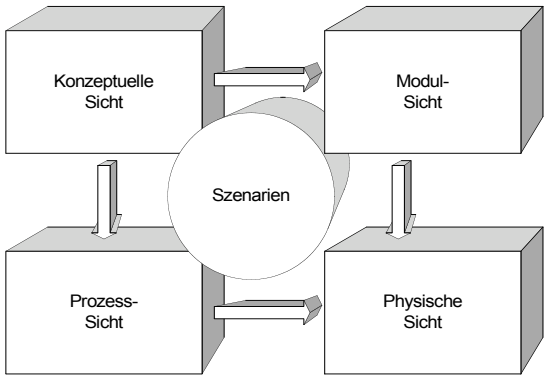
## Software Architektur Anforderungen

- Anforderungen an die SWA (messbar)
  - Erfüllung der Funktionalen Beschreibung
  - Erfüllung der Performance-Constraints
- Meta-Anforderungen
  - Adaptierbarkeit
  - Flexibilität
  - Interoperabilität
  - Wiederverwendbarkeit in anderen Systemen

## Software Architektur Beschreibungen

- Verschiedene Dokumentationsansätze
  - Komponenten, Konnektoren, Constraints, Funktionalität
  - Verschiedene Diagrammtypen
    - Unverständlich bzw. mehrdeutig
    - Nicht aussagekräftig bzw. überladen
- Lösung durch Integration verschiedener Sichtweisen

## Software Architekturen 4+1 views




## Konzeptuelle Sicht (Logiksicht)

- Funktionale Anforderungsbeschreibung
- Orientierung an der Problemstellung und dem Anwendungsbereich
- Kommunikation mit Experten
- Unabhängig von echten Implementierungsentscheidungen
- Gedankengerüste

## Modul Sicht (development view)

- Organisation der Module
  - Subsysteme
  - Zusammengehörige Teile der Entwicklungsarbeit
  - Allokation des Aufwands (Entwicklung, Wartung)
- Organisation in hierarchische Ebenen
  - Protokoll-Stacks, Applikationsebenen, etc.
- Statische Strukturdefinition
  - Datenstrukturen, memory allocation, etc.

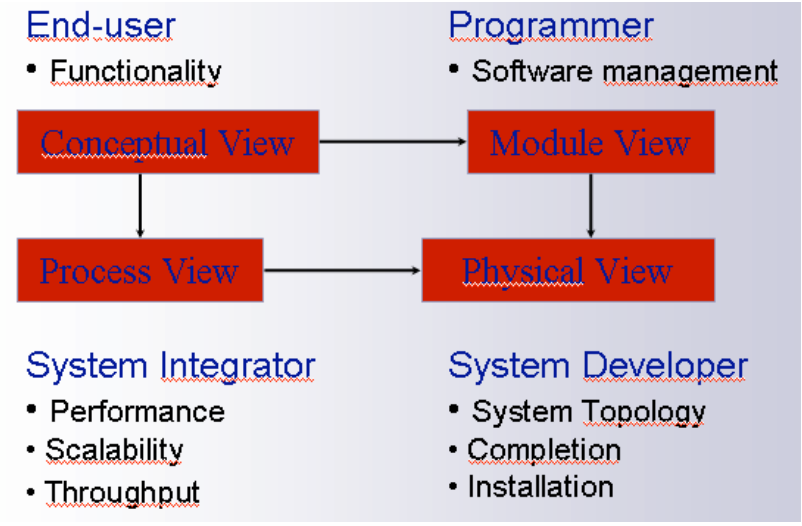
## Prozess Sicht (process and coordination view)

- Dynamische Aspekte der Prozesse zur Laufzeit
  - Process Generierung
  - Synchronisation
  - Concurrency
- Zentrale Komponenten dieser Sichtweise sind die Prozesse: Instruktionen und spezifische Ausführungslogik
- Abschätzungen/Festlegungen der Prozess-Allokationen etc.

## Physische Sicht

- Abbildung/Ablauf der Software auf real existierender Hardware
  - z.B. Verteilung der Aktivitäten und Prozessablaufe innerhalb eines vernetzten Systems
- Hat Auswirkungen auf
  - Verfügbarkeit, Zuverlässigkeit, Performanz und Skalierbarkeit
- Diese Strukturierung soll möglichst wenig Auswirkungen auf die Implementierung der einzelnen Komponenten haben (Transparenz)

## Integration der Sichten



# Software Architektur Stile

## Was ist ein Architektur-Stil?

- Eine Design-Sprache für eine Klasse von Systemen
  - Vokabular für Design-Elemente, z.B. Pipes, Filter, Server, DBs
  - Design-Regeln und Einschränkungen (constraints), z.B. Client/Server-Verhältnis n:1
  - Semantische Interpretation
  - Analysen zur Überprüfung der Entsprechung eines Entwurfs, z.B. Deadlock-Erkennung für concurrent systems, Schedulability-Analyse,etc.

## Definition eines Architektur Stils (Perry u. Wolf 1992)

- Ein architektureller Stil definiert eine Familie von Software-Systemen bezüglich ihrer strukturellen Organisation.
- Ein architektureller Stil drückt die Komponenten und Relationen zwischen diesen - gemeinsam mit den Einschränkungen ihrer Anwendung, der assoziierten Komposition und den Design-Regeln für deren Konstruktion - aus.

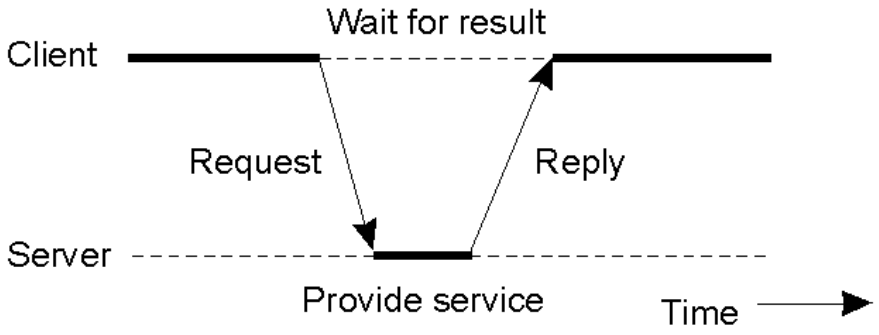
## Beispiele von Architektur Stilen

- Pipes & Filters
  - Sequentielle Abläufe, input/output streams
- Objects
  - Datenkapselung, Methoden
- Ereignisgesteuerte Systeme
  - Event handling, queuing
- Data sharing, Data Centering
  - Datenbankmanagementsysteme

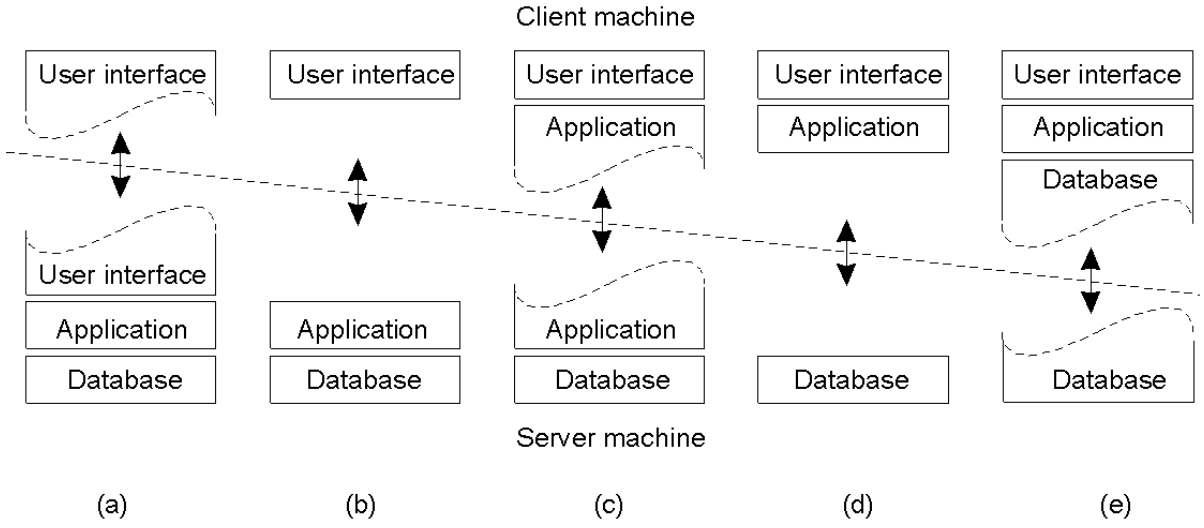
# Verteilte Systeme

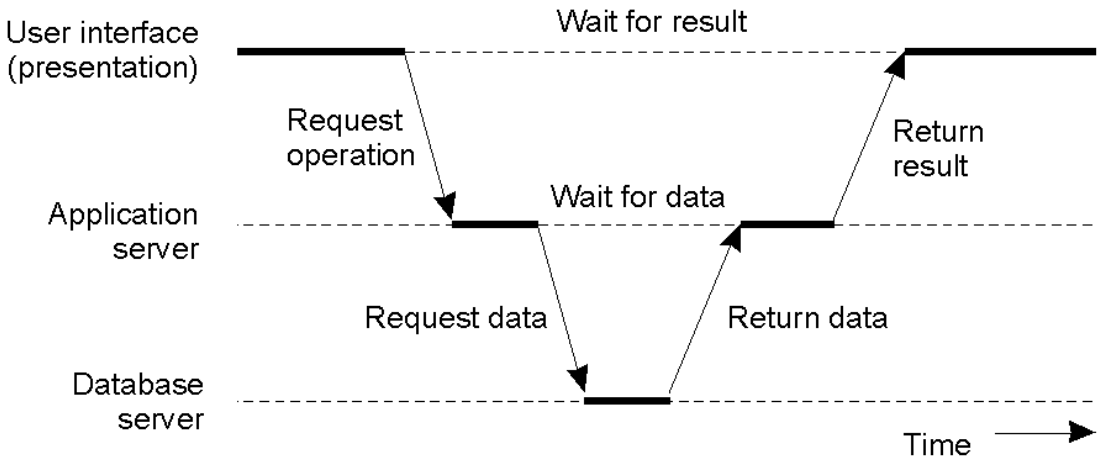
- Ein Verteiltes System
  - Ist eine Zusammenstellung von unabhängigen Computern, die dem Benutzer als eine Einheit erscheinen
  - Das „einheitliche“ Erscheinungsbild wird durch verschiedene Transparenzebenen erreicht
    - Access, Location, Migration, Relocation, Replication, Concurrency, Failure, Persistence

**Gebräuchlichster Ansatz: Client/Server**

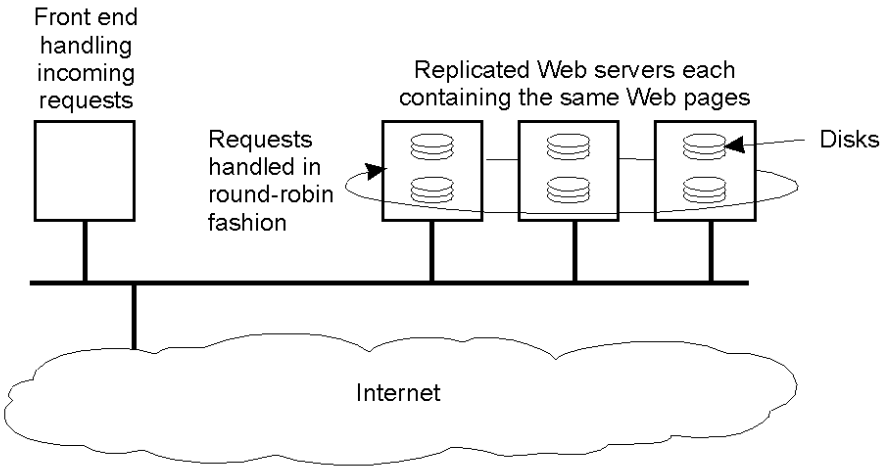



**Multi-Tiered Architectures**






**Aktueller Ansatz**

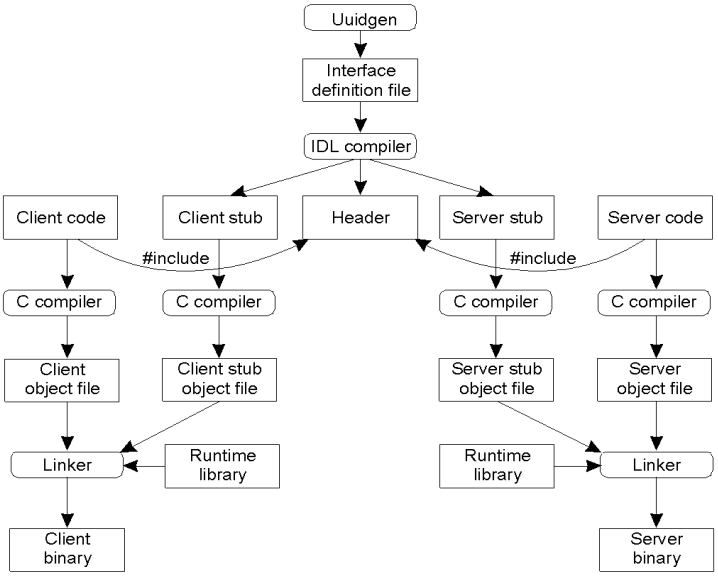



## Kommunikation in verteilten Systemen

- Kommunikationsparadigmen
  - Schichtenmodelle (OSI, TCP/IP)
  - Middleware Kommunikation (Abstraktion)
  - Remote Procedure Calls (SUN RPC)
  - Remote Object Invocation (RMI)
  - Messaging Systems (synchronous, asynchronous)
  - Streaming

- Remote Procedure Call
  - Paradigma geprägt von SUN Microsystems Ende 80er
  - Client/Server-Stubs
  - Gemeinsame XDR-Sprache / Marshalling
  - Programmiersprache C (C++ wrapper)
  - Aufruf wie lokale Prozedur

## Remote Procedure Call Generierung



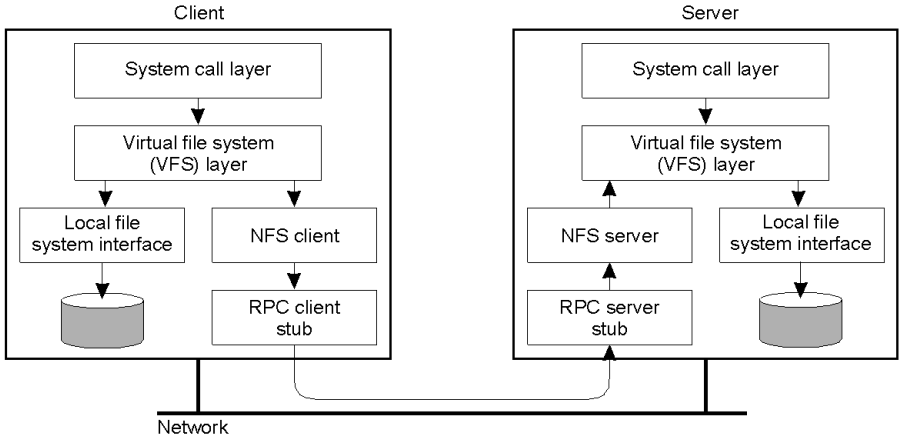

## Klassifikation verteilter Systeme

- Anwendungsbereiche Verteilter Systeme
  - Network File Systems
    - NFS, Coda, AFS
  - Distributed Information Systems
    - Data sharing, document-based, message-based
  - Distributed Coordination-based Systems
    - JavaSpace, Rendezvous
  - Peer-to-peer Systems
    - Client & Server in Doppelrolle

## Netzwerk Filesysteme

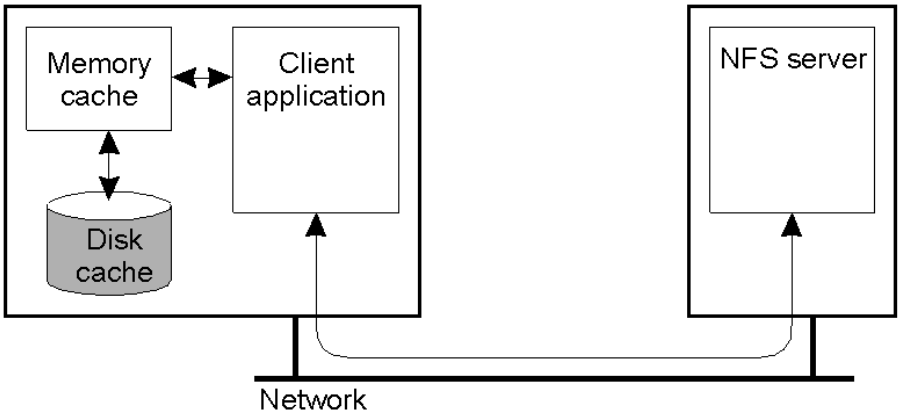
- Das SUN Network File System versucht, ein Filesystem transparent (für den Benutzer unsichtbar) den Klienten zugänglich zu machen
- Das NFS folgt dem remote access model
  - Die Daten bleiben auf dem entfernten Rechner, Verarbeitungsschritte werden über das Netzwerk transportiert
  - Im Gegensatz dazu gibt es upload/download Modelle, die die Daten tatsächlich bewegen und mit mehreren Kopien arbeiten (AFS)

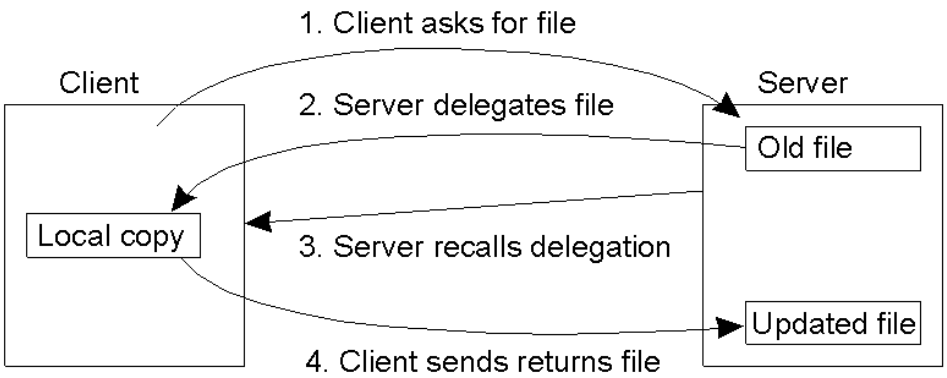
**NFS basiert auf dem Konzept des Virtuellen Filesystems (VFS)**



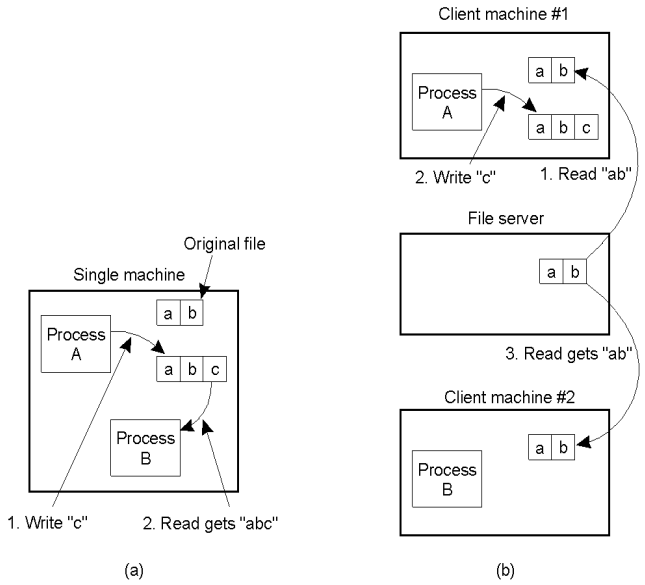

- Remote access model in NFS ist problematisch für Performance
- Lösungsansatz: Client-Side Caching
  - Eine lokale Kopie am Client wird beim Zugriff angelegt
  - Semantische Unterschiede bei Lese-/Schreibzugriffen
  - Server „delegiert“ Datei an den Client

## Client-Caching



- Client Caching verursacht Konsistenz-probleme für verteilte Filesysteme:
  - In lokalen Systemen bleibt eine sequentielle Abarbeitung von Schreib- und Lesezugriff unproblematisch
  - In einem verteilten System mit Caching können ungültige (d.h. nicht mehr gültige) Werte zurückgeliefert werden.

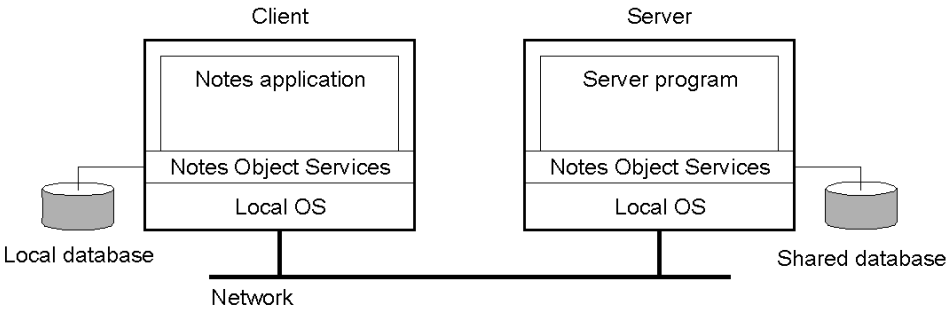



- Wie soll man mit Files in Verteilten Systemen umgehen?
  - Jede Operation auf einem File ist sofort für alle Prozesse sichtbar (UNIX-Semantik)
  - Keine Änderung am File ist sichtbar bis das Dokument geschlossen ist (Session-Semantik)
  - Änderungen sind von vornherein ausgeschlossen (immutable file semantic)
  - Alle Änderungen sind atomar (Transaktions-Semantic)
- Vor- und Nachteile für den jeweiligen Anwendungsbereich abwägen

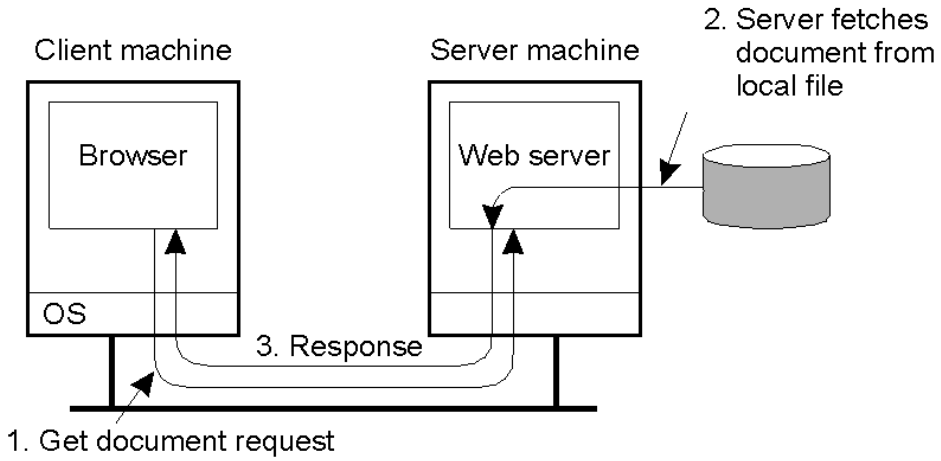
## Distributed Information Systems

- Verteilte Informationssysteme
  - Software Systeme, deren zentraler Fokus auf der Präsentation und Distribution von Informationen liegt
  - Dokumentbasierte Ansätze
    - WWW
    - Lotus Notes
  - Datenbankzentrierte Ansätze
    - Hyper-G, Hyperwave

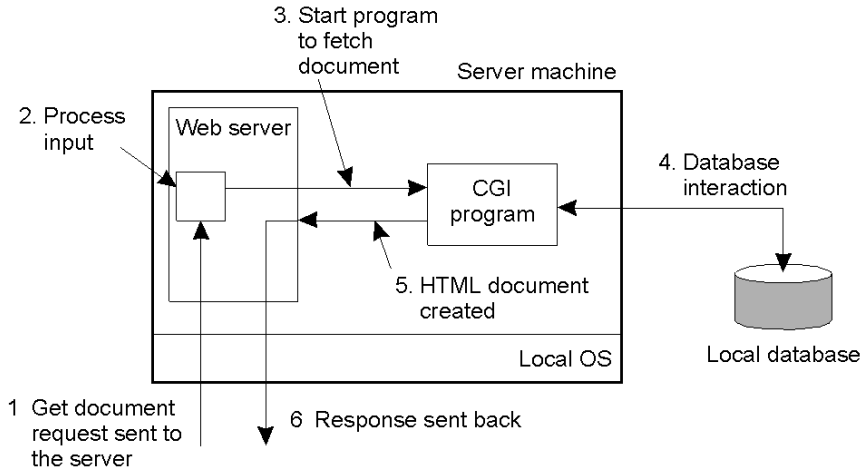
- Lotus Notes
  - Document replication
  - Document conflict resolution mechanisms
  - Least loaded server selection



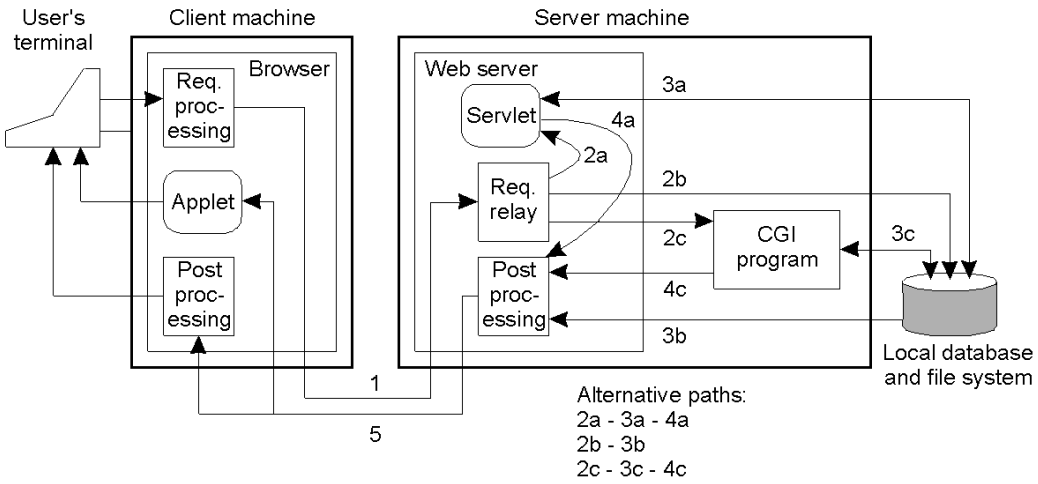

## The World Wide Web



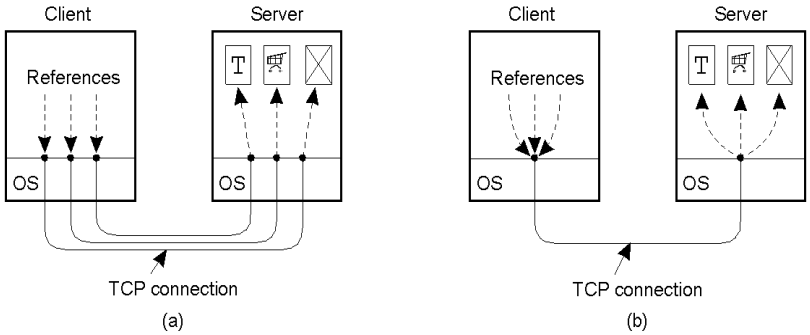

## Web Service Architektur mit simplem CGI



- Verschiedene Prozessabläufe im WWW
  - Basis: Client/Server Architektur

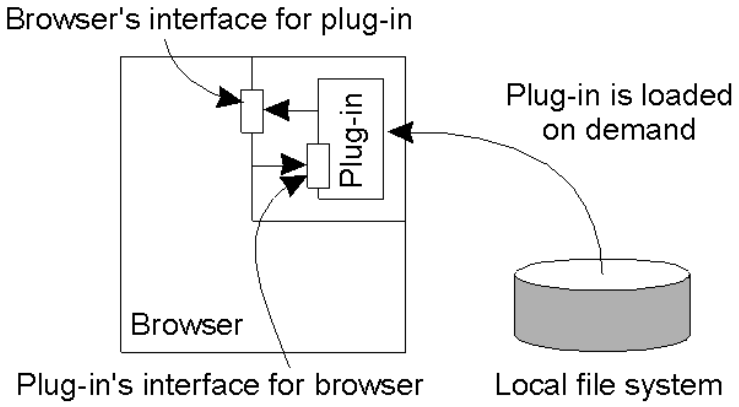



- Verbindungscharakteristiken
  - One-time connections
  - Persistente Verbindungen



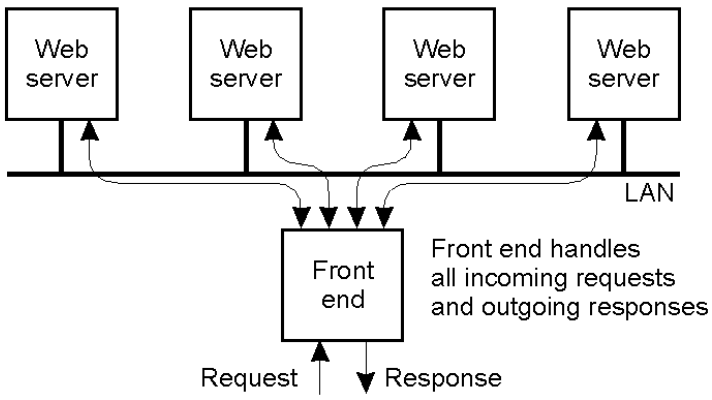

- Clientseitige Architektur
  - Die Software auf der Klientenseite ist meist monolithisch aufgebaut
  - Client dient als reines Instrument, vorgefertigte Information der Serverseite zu visualisieren
    - Optionale Erweiterungen:
      - Persistenz des Kommunikationsstatus
      - Integration von Clientseitiger Programmlogik (Applets, plugins, add-on features)

- Beispiel Client-Plugin Software
  - Etwa Shockwave
  - SVG-Viewer



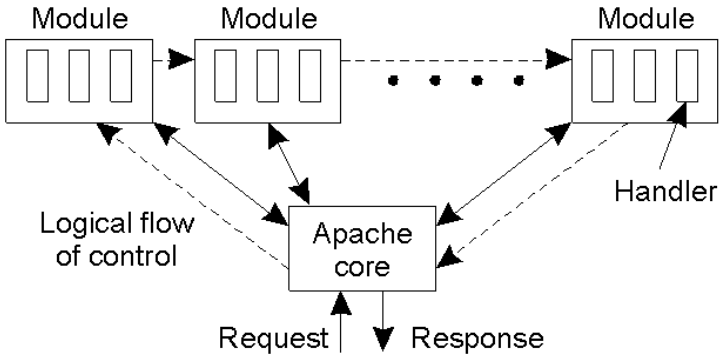

- Serverseitige Architektur
  - Der Webserver ist aufgrund der antizipierten Lastverteilung als simpler, aber leistungsfähiger Dienstleister ausgelegt
  - Ansätze
    - Monolithisch
    - Parallele Prozesse mit dispatcher
    - Light-weight/heavy-weight servers
    - Load balancing servers
    - Monitored servers

- Ein Server-Cluster
  - Mehrere gleichwertige Server werden von einem Frontend aus angesprochen
-



Modell des Apache Webservers

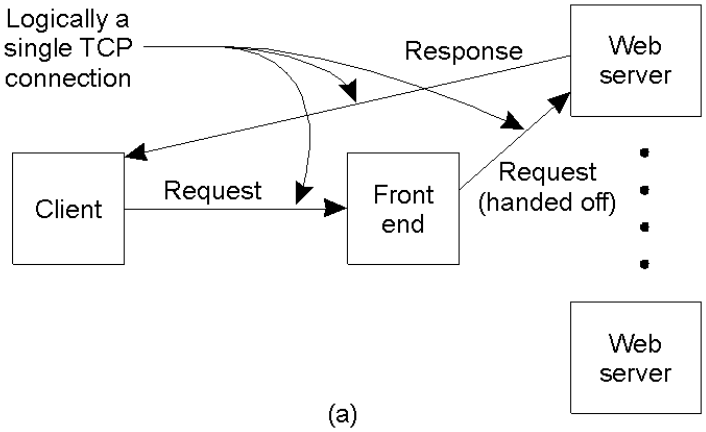
- Folgt dem Cluster-Modell (sowohl auf einem einzelnen als auch auf mehreren Rechner)
- Hat intern einen modularen Aufbau




- Server Clustering
  - Das Einbinden von mehreren Hardware-Servern in ein logisches Gesamtsystem
  - Mehrere Server können einen bestimmten Request beantworten
  - Ein dispatcher entscheidet, welcher Server den request erhalten soll
    - Zufälliges Aufteilen
    - Gesteuertes Aufteilen
      - Load balancing
      - Membership management
      - Round-Robin

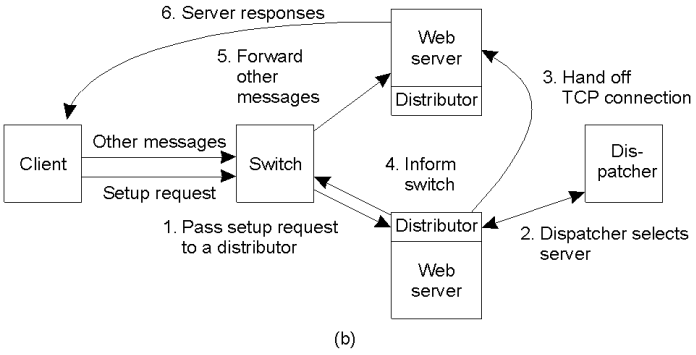
**Server Clustering**

-



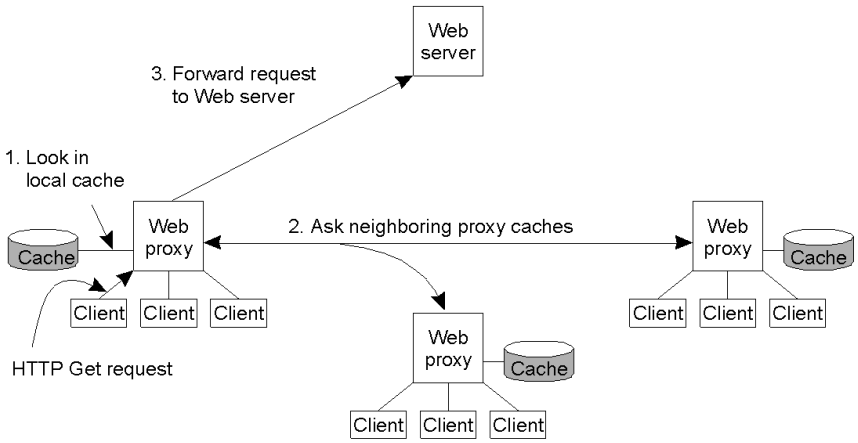
Intelligentes Server Clustering

- Für bestimmte Webservices ist es notwendig, einen Dispatcher einzusetzen, der mehr über die inhaltliche Leistungsfähigkeit der im Cluster enthaltenen Server weiß.
- Ein einfacher Switch teilt die Requests auf
- Ein Distributor-Prozess übernimmt den Setup-Request mit Hinweisen auf die Qualität der Verbindung
- Ein Dispatcher weist dem Switch die Information zu, an welchem Server künftig requests dieser Art bearbeitet werden (evtl. für Verbindungen persistenter Natur, session management, etc.)

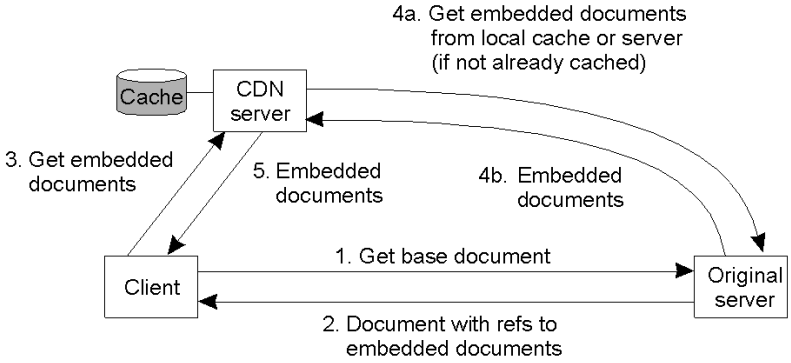



- Content Distribution
  - Oft ist es auch sinnvoll, Inhalte nicht nur innerhalb eines Servers auf mehrere Prozesse, oder innerhalb eines Clusters auf mehrere Server zu verteilen, sondern
  - Inhalte topologisch deutlich voneinander getrennt aufzubewahren
  - Ansätze
    - Web Proxying/Caching (low level control)
    - Content Distribution Networks (high level control)

- Web Proxy Servers, Web cache servers
  - Proxy servers agieren als Torwächter für Service-Zugriffe von außen
    - Kontrolle der Berechtigung
    - Schutz der internen Strukturen
    - Evtl. Cache-Funktionalität
  - Web Caching Servers speichern oft verwendete Inhalte zwischen
    - Rascherer Zugriff auf statische Dokumente
    - Kooperation mit anderen Cache-Servern um Netzwerklast zu verringern
    - Update-policies (FIFO, LRU)




- Content Distribution Networks (CDN)
  - Ein Content Distribution Network hat den Zweck, Servicelast vom Originalserver zu entnehmen
  - Anwendungsbeispiele: Downloadseiten von weltweiten Unternehmen, open source distribution
  - Möglichst hohe Transparenz für den Benutzer gefordert
    - URL rewriting
    - DNS redirection



## P2P Architekturen

- Was sind Peer-to-Peer Netzwerke?
  - Simplifizierung des Cient/Server-Architekturmodells für spezielle Anwendungsbereiche
- Client = Server?
  - Eine P2P Applikation zeichnet sich durch die Charakteristik aus, sowohl in die Rolle des Clients als auch in jene des Servers schlüpfen zu können
- Anwendung?
  - Rascher Datenaustausch, Anbieter und Empfänger gleichzeitig

- Installation
  - Wer eine P2P-Applikation installiert braucht kein Detailwissen zu VSWA, die Applikation selbst erkennt (transparent für den Anwender), in welcher Rolle sie eingesetzt wird
- Lastverteilung und Leistungssteigerung
  - P2P-Applikationen können durch ausgeklügelte verteilte Algorithmen Informationen aufeinander aufteilen. Jeder Peer, der einmal Daten besitzt, kann diese ab diesem Zeitpunkt auch anbieten. Anderen Peers bietet sich demnach ein immer größer werdendes Netzwerk an Anbietern
- Missbrauch?
  - Durch automatische Verteilung von wertbehafteten Informationen wird es rechtlich immer komplexer, Urheber von unerlaubten Informationsangeboten zu ermitteln bzw. die Verbreitung zu unterbinden

## Perl

- Die Programmiersprache PERL
  - Entwickelt von Larry Wall (1987)
  - Unterstützt von Tom Christiansen und Randal L. Schwartz
  - PERL ist die „Practical Extraction and Report Language“

- Warum PERL lernen?
  - Systemadministration mit einer Allzwecksprache
  - Unix-Anwendungen optimieren
  - CGI-Programme fürs WWW entwerfen
  - Prototypen für Programme entwerfen
  - Keine Entwicklungsumgebung notwendig
  - Bequem portierbar
  - Leistungsfähig
  - TMTOWTDI

- Grundlagen rund um PERL
  - PERL ist eine interpretierte Programmiersprache
  - Basiert auf C, unix tools (awk, sed)
  - Prozedural oder objektorientiert anwendbar
  - Grundsprache oder Bibliotheken (pm)
  - Open community unterstützt PERL
  - CPAN als riesige PERL-Ressource
    - www.cpan.org
  - Manpages rund um perl
    - Man perl, perldoc perlfaq, perldoc <Modulname>

## Perl Grundlagen

- Hello World in PERL
  - Dokument hello.pl editieren:
    #!/usr/bin/perl –w
    print "Hello World!\n";
- Programm ausführbar setzen
  Chmod +x hello.pl
- Programm ausführen

#> hello.pl

- Einführungskurs in PERL
  - Perl Datentypen
    - Skalare ($scalar)
    - Arrays (@array)
    - Hashes (%hash)
      - Beispiele als Demo
  - Perl Operationen
    - Arithmetische (+,-,*,/,**,%)
    - Comparative (==, !=, <, >,<=, >=, <=>, cmp)
    - Logische (&&, ||, ! Bzw. AND, OR, NOT)

- Einführungskurs in PERL
  - Perl Operationen (2)
    - Zuweisungen ( $lvalue = $rvalue )
    - Inkrement/Dekrement ( $i++, $i--, ++$i, --$i )
    - Zuweisungsoperatoren
      - Variabler Inkrement (+=, -=)

*=, /=, %=, **=

- Stringverkettung, -wiederholung ( . x)

- Rangfolge und Assoziativität

- Ein- und Ausgabe in PERL
  - Dateihandles
  - Standardein-/-ausgabe

STDIN, STDOUT, STDERR
FILEH

$zeile = <STDIN>;
Print STDOUT „hello“;

- Einsatz der Datentypen
  - Skalare

Skalare werden verwendet um Strings und Zahlen zu bearbeiten.

$zahl = 24;
$text = “das ist ein testtext“;
$zahl++;
$text .= “\n“;
Kontextsensitive Umwandlung in PERL

## Datentypen und -strukturen

- Einsatz der Datentypen
  - Arrays

In Arrays werden Listeninhalte verwaltet.

@liste = (´das´,´ist´,´eine´,´liste´);
@zahlen = (1..1000);
@nichts = ();
@zahlen = (@zahlen, 1001, 1002, 1003);
($a, $b) = (1,2);
($x, $y) = ($y, $x);

print $zahlen[3];
$#zahlen, scalar(@zahlen)

- Einsatz der Datentypen
  - Arrays bearbeiten
    - Sortieren

@geordnete_zahlen = sort { $a <=> $b } @zahlen;

    - Durchlaufen

foreach $x (@liste) { ... }

    - Listen einlesen, ausgeben

Chomp, chop, @zeilen = <STDIN>, print @zeilen;

    - Elemente hinzufügen/entfernen

push, pop, shift, unshift, splice

- Einsatz der Datentypen

Mit Hashes arbeiten

  - Hashes sind wie Arrays eine Sammlung von Daten (komplexe Datenstruktur)
  - Hashes sind assoziative arrays bestehen aus ungeordneten Paaren von Schlüsseln und Werten
  - Hashes können als Listen definiert werden (schlüssel1, wert1, schlüssel2, wert2,...)

- Einsatz der Datentypen

Mit Hashes arbeiten

```
%paare = (´rot´, 255, ´grün´,150,´blau´,100);
%paare = (rot => 255, gruen => 150, blau => 100);

%leer = ();
$paare{schwarz} = 50;
$rot_wert = $paare{rot};
```

- Einsatz der Datentypen

Auf Hashelemente zugreifen

```
Alle Werte
@werte = values(%paare);
@schluessel = keys(%paare);

Löschen: $geloeschter_wert = delete $paare{rot};
Durch-Iterieren: foreach $stadt (sort keys %paare) {...}
```

- Einsatz der Datentypen

Komplexere Datentypen

Durch die Kombination der angewandten Datentypen lassen sich komplexeste Datenstrukturen erstellen. Allerdings können in den Datenstrukturen nur Skalare verschachtelt werden. Über Referenzen können Skalare auf andere Datenstrukturen erzeugt werden:

```
$arrayref = \@werte;
$hashref = \%paare;
$komplex_list = ($arrayref, $hashref, 27, ´text´);
```

## Flusssteuerung

- Blöcke

Anweisungen werden mit ; abgeschlossen
Eine Gruppen von Anweisungen kann mit { und } zu einem Block zusammengefasst werden:

```
{
  $a = 1;
  $b = 2;
}
```

Blöcke können als bare blocks definiert werden oder im Rahmen der Flusssteuerung (Bedingungen, Schleifen) verwendet werden.

- Bedingungen

if, else, elsif, unless

```
if ($a < 0) {
  $a = 1;
} else {
  $a = -1;
}
```

Der Bedingungsoperator ?:

```
$a = ($a < 0)?1:-1; # können auch expressions sein
```

- Schleifen

While-Schleifen Until-Schleifen

```
while ($a > 0) {        #solange wahr / falsch
  print $a--;
  next if ($a == 100);
  last if ($a == 99);
}
```

Do-Schleifen

```
Do { ... } while ($a>0);  # do { ... } until ($a == 0);
```

- Schleifen

for-Schleifen

```
# for (Startwert; Bedingungsausdruck; Aenderung) {...}

for ($i = 1; $i < 10; $i++) {
  print $i;
}
```

**foreach** ist eine Spezialform, wo die Anzahl der Durchläufe auf die Anzahl der Elemente des angegebenen arrays beschränkt ist.

## Perl Pattern Matching

- Bisher gelerntes bildet die Basis vieler anderer Programmiersprachen
- Pattern Matching bildet in PERL den leistungsfähigsten und flexibelsten Aspekt
  - Was sind regular expressions (patterns)
  - Wie werden diese angewandt
  - Wozu dient pattern matching

- Pattern Matching Operatoren und Ausdrücke
  - Operator =~if ($string =~ m/muster/) { ... }
  - Einfache Muster:/foo//dies oder das//GrOsSklEineGAl/i
  - Sonderzeichen in Patterns:^, $, ., +, ?, *, {, }, (, ), \, /, |, [, ]  # müssen „escaped“ werden
  - Klassen von Muster:\d, \D, \w, \W, \s, \S

- Pattern Matching Spezialfälle
  - Mehrfaches Erkennen/.../g;
  - Negative Patterns!~ /.../;
  - Gruppierung und Alternativen/(H|M|S|Br)aus/
  - Prioritäten (lookahead)/abc(?=de)/, /abc(?!de)/

- Suchen und Ersetzen
  - Teile isolieren und speichern/abc(\w+)xyz/
  - Der Operator s///$text =~ s/abc(\w+)xyz/ABC$1XYZ/;
  - Pattern Matching über mehrere Zeilen s///s, s///m
  - Ersetzen durch einen PERL-Ausdruck s/abc(\w+)xyz/“ABC“.uc($1).“XYZ“/e;

## Perl Sub-Routinen

- Subroutinen erstellen und verwenden
  - Subroutinenaufruf $ergebnis = &addiere(2,3);
  - Werteübergabe, lokale Variable und Ergebnisrückgabe sub addiere {  my ($a, $b) = @_;  return $a+$b;

# Serverseitiges Programmieren

- Softwareentwicklung für client/server und multitier architectures
  - Schwerpunkt Web Applikationen
  - Embedded languages
  - CGI-Programming
  - Auszüge und Praxiserfahrungen mit
    - PHP, Zend
    - PERL

- Beide Ansätze (PHP,PERL) sind Teil der open source Philosophie
- Wichtige Bestandteile der LAMP-Architektur im open source Bereich:Linux, Apache, Mysql, PHP/PERL
- Scripting-Programmiersprachen, die ob der besonderen Eignung für rasche Web-Applikationserstellung als Programmiersprachen unterbewertet werden -> http://www.perl.com/pub/a/2000/01/10PerlMyths.html

- PHP (..rekursiv.. Hypertext Processor)
  - ist eine weit verbreitete und für den allgemeinen Gebrauch bestimmte Open Source Skriptsprache, welche speziell für die Webprogrammierung geeignet ist, und in HTML eingebettet werden kann.
  - Unterschied zu Skripts, welche in anderen Sprachen wie Perl oder C geschrieben wurden: anstatt ein Programm mit vielen Anweisungen zur Ausgabe von HTML zu schreiben, schreibt man einen HTML-Code mit einigen eingebetteten Anweisungen

- PHP
  - PHP ist hauptsächlich auf serverseitige Skripte fokussiert, weshalb Sie alles tun können, was auch ein anderes CGI Programm kann, wie z.B. Formulardaten sammeln, dynamische Inhalte für Websites generieren oder Cookies senden und empfangen. Aber PHP kann noch viel mehr.
  - Alternativen:
    - Skripte auf Kommandozeile (mit PHP-Parser)
    - GUIs mit standalone PHP (mit PHP-GTK)
- PHP
  - Auf allen gängigen Betriebssystemen lauffähig für die meisten Webserver in Modulform, für andere als CGI-Prozess
  - Prozedural, objektorientiert oder gemischt programmierbar
  - Keine Beschränkung auf HTML, etwa PDF, XHTML, XML, Bilder oder Flash Animationen
  - Integration vieler Datenbanken (u.a. mysql,Oracle)
  - Unterstützung vieler Web-Protokolle um andere Applikationen einzubinden
  - ...

- PHP
  - Systemvoraussetzungen
  - Webserver, der PHP unterstützt (PHP-Modul ist integriert oder PHP-Parser ist als CGI-Programm verfügbar und die Dateiendung .php ist der Applikation PHP zugeordnet), z.B. Apache
  - Php-Dateien werden wie Textdateien in für den Webserver lesbare Verzeichnisse gestellt und sind damit ausführbar
  - PHP bietet automatisch Zugriff auf wichtige Systemresourcen: phpinfo()

- PHP
  - Globale Variable in PHP
    - $_SERVER-Array, etwa
    - $_SERVER[“HTTP_USER_AGENT“]

  - Formularparameter
    - $_POST-Array, etwa
    - $_POST[“name“]

- PHP Zend
  - Zend steht für die Maschine, die als Kern die Anweisungen und Sprachkonstrukte abarbeitet
  - PHP selbst steht für das Gesamtsystem, so wie es von außer her erscheint (mit allen Features, Libraries, etc.)

  - Um einen Web-Skript Interpreter zu bauen, braucht man drei Komponenten:
    - (a) Den Interpreter, der den Code analysiert, übersetzt und ausführt
    - (b) Die Funktionalität, die durch die logischen Programme in den Modulen definiert ist
    - (c) Das Interface, durch das mit dem Webserver kommuniziert wird

- PHP Zend
  - Zend übernimmt dabei (a) und kleine Teile von (b)
  - PHP ergänzt dann (b) und (c)