Exam Preparation

Web Engineering

Dierk König

The Web

Publishing in an open way,

Sharing in a standard way,

Linking in an flexible way.

Collaboration, data submission

User Interfaces, applications

SGML Derivatives

Standard Generalized Markup Language

HTML - XML - DocBook - RDF

HyperText Markup Language

HTML

<!DOCTYPE html>

Elements: <tag> *content* </tag>

Attributes: <tag name=value>

Entities: ö

Readers

People with Browsers

Screen Readers, OS Integration

Search Engines

Web scrawlers, scrapers

Key Elements

HTML, HEAD, BODY
META, TITLE, LINK, BASE
H1, H2, H3, H4, H5, H6
P, A, DIV, SPAN, STYLE
SCRIPT

Media

FIGURE, FIGCAPTION
PICTURE, IMG, AUDIO, VIDEO
OBJECT, EMBED

CANVAS, MATH, SVG

Frames

IFRAME

FRAMESET, FRAME
no longer supported in HTML5

Presentational

EM, I, STRONG, B
INS, DEL, PRE,
SMALL, SUB, SUP

BR, HR

Collections

LI
OL, UL
DL, DT, DD
TABLE, COLGROUP, COL,
CAPTION, THEAD, TFOOT,
TBODY, TR, TH, TD

SEMANTIC

NAV, MENU, HEADER,
HGROUP, FOOTER, MAIN,
ASIDE, TIME, SECTION,
ARTICLE, ADDRESS, DFN,
ABBR, DETAIL, LEGEND,
PROGRESS, OUTPUT, SAMP

FORMS

FORM
INPUT, TEXTAREA, BUTTON,
SELECT, OPTION, OPTGROUP,
DATALIST
FIELDSET, LABEL, KEYGEN

Input types

button, checkbox, color, date, datetime-local, mail, file, hidden, image, month, number, password, radio, range, reset, search, submit, tel, text, time, url, week

Some more…

CITE, BLOCKQUTOTE, Q,
AREA, BDO, CODE, RUBY, VAR,
WBR, COMMAND, NOSCRIPT,
TEXT, …

Elements

Block elements: P, DIV,…

Inline elements: A, SPAN, …

Inline elements are not allowed to have block elements inside.

Elements

Some elements are self-closing:

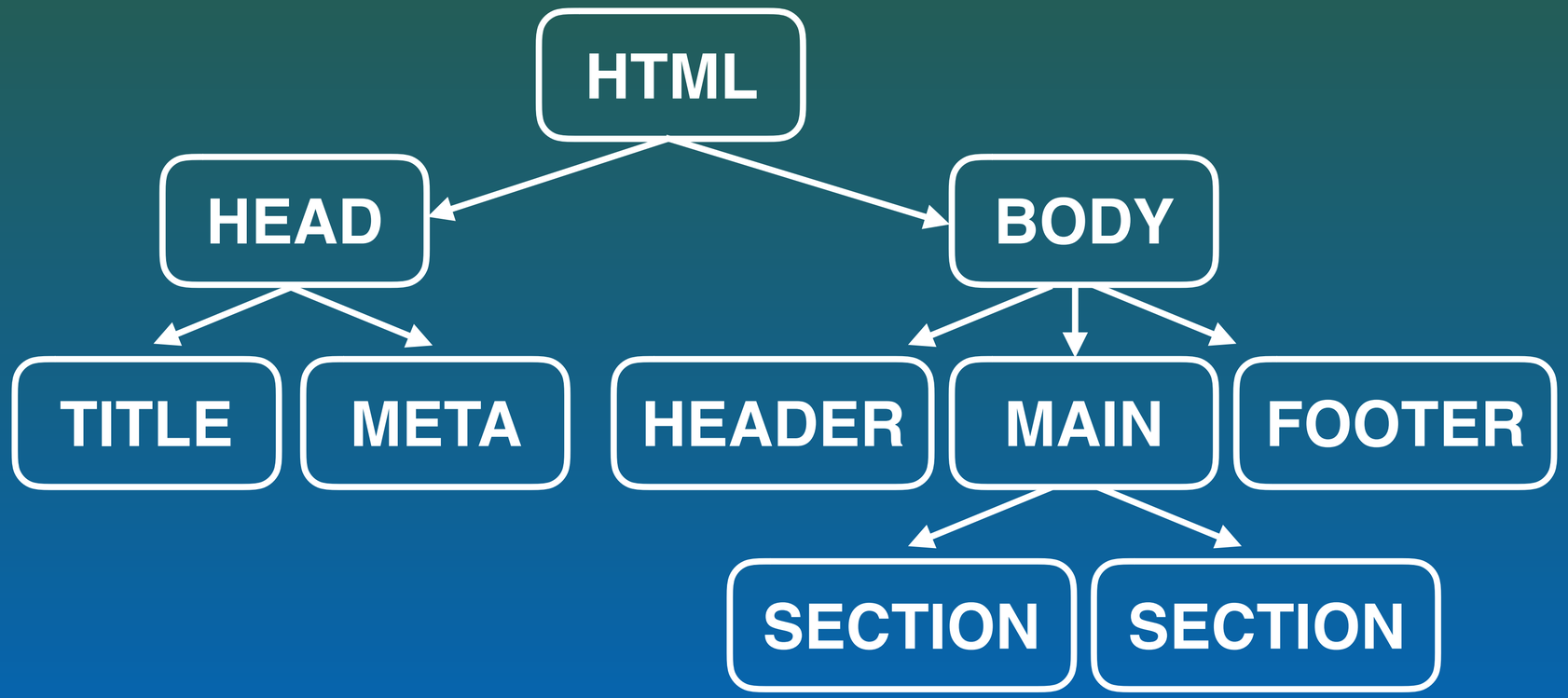
<IMG></IMG> (not allowed)

<IMG/> (ok)

<IMG> (ok)

DOM

Document Object Model



Well-formed vs Valid

Well-formed:

tags syntacticly correct

Valid:

tags nested according to DTD (document type definition)

Which is Which?

```
<html><body></html></body>
<html><body  </body></html>
<span><div></div></span>
<p><em>x</em></p>
<p><em>x<p>
```

HTML is forgiving

Unknown tags are ignored

Unknown attributes are ignored

… and browsers display almost anything.

Meta

charset

http-equiv, content

name, content

author, description, keywords, viewport

Global Attributes

id, class, title, lang,
accesskey, tabindex,
contextmenu, spellcheck,
draggable, dropzone, …

Event Attributes

onClick,

onMouseOver,

onChange, onInput, …

Attributes

are *sometimes* inherited
from parent to children
through the DOM

Attributes

User data

- data-*

Microformats

- itemscope, itemtype, itemprop

CSS

Web Engineering

Dierk König

CSS Rule

CSS Selectors

h1		element
#top	<.. id="top">	id
.cool	<.. class="cool">	class
*		all

can be a list like "cool drink"

Special Selectors

[title]	<.. title="..">	attribute
[title="x"]	<.. title="x">	attribute/value ^= $= *= ~=
p:hover	<p ...	pseudo-class

Special Selectors

h1+	next sibling
h1~	all siblings

P.S. there is much more that we don't cover

:not, :nth-child, :nth-child-of-type *pseudo-class-functions*

::first-letter *pseudo-elements*

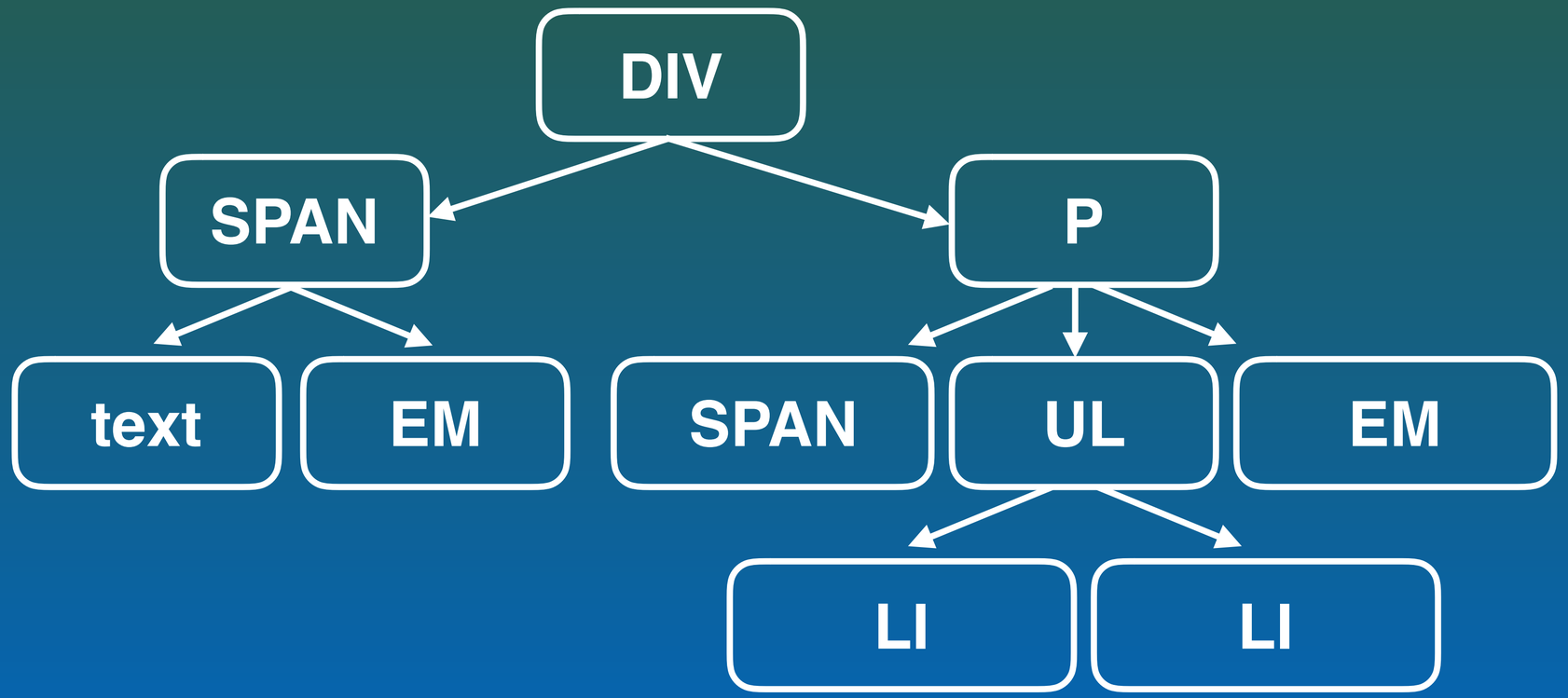
Selector Combinations

h1, h2	union
main p	descendant
ul > li	direct child

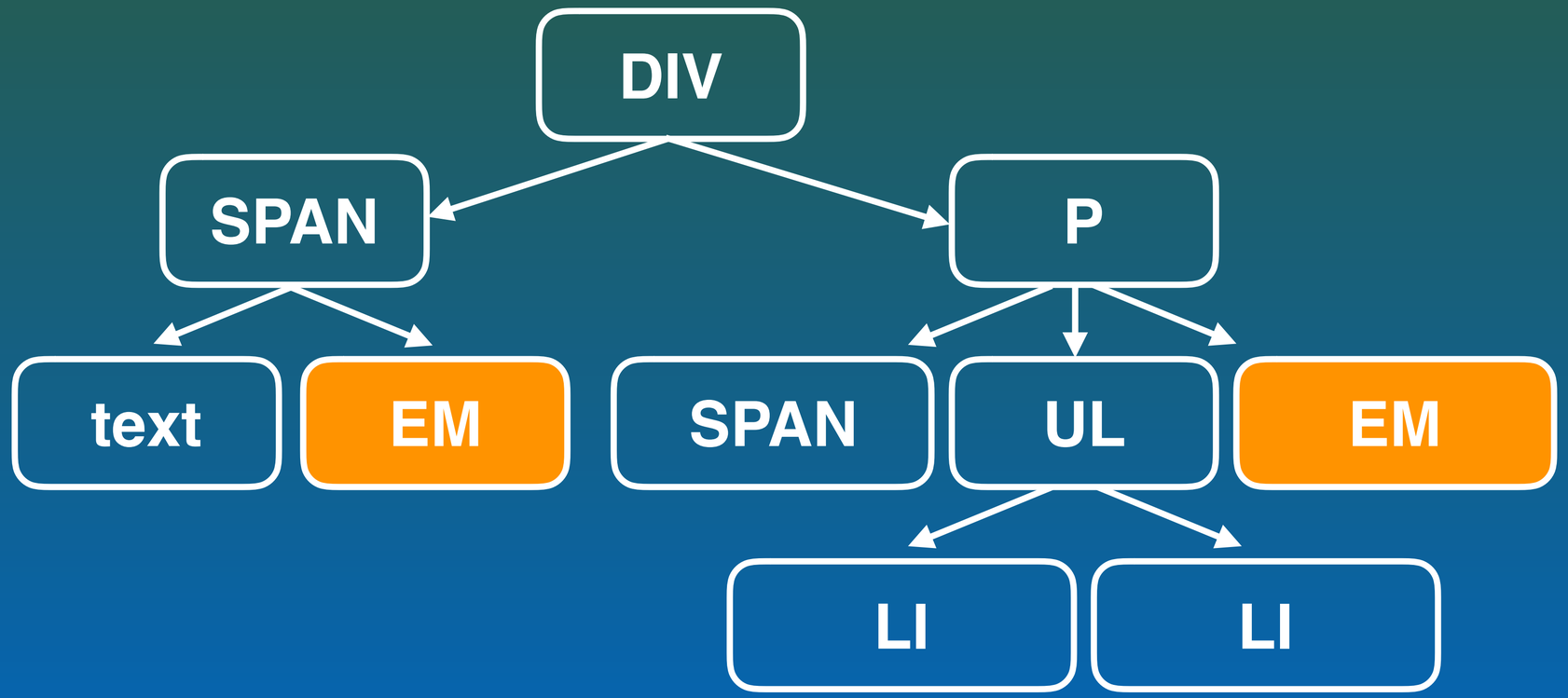
Selector Combinations

div.cool	element with class
div[cool]	element with attribute

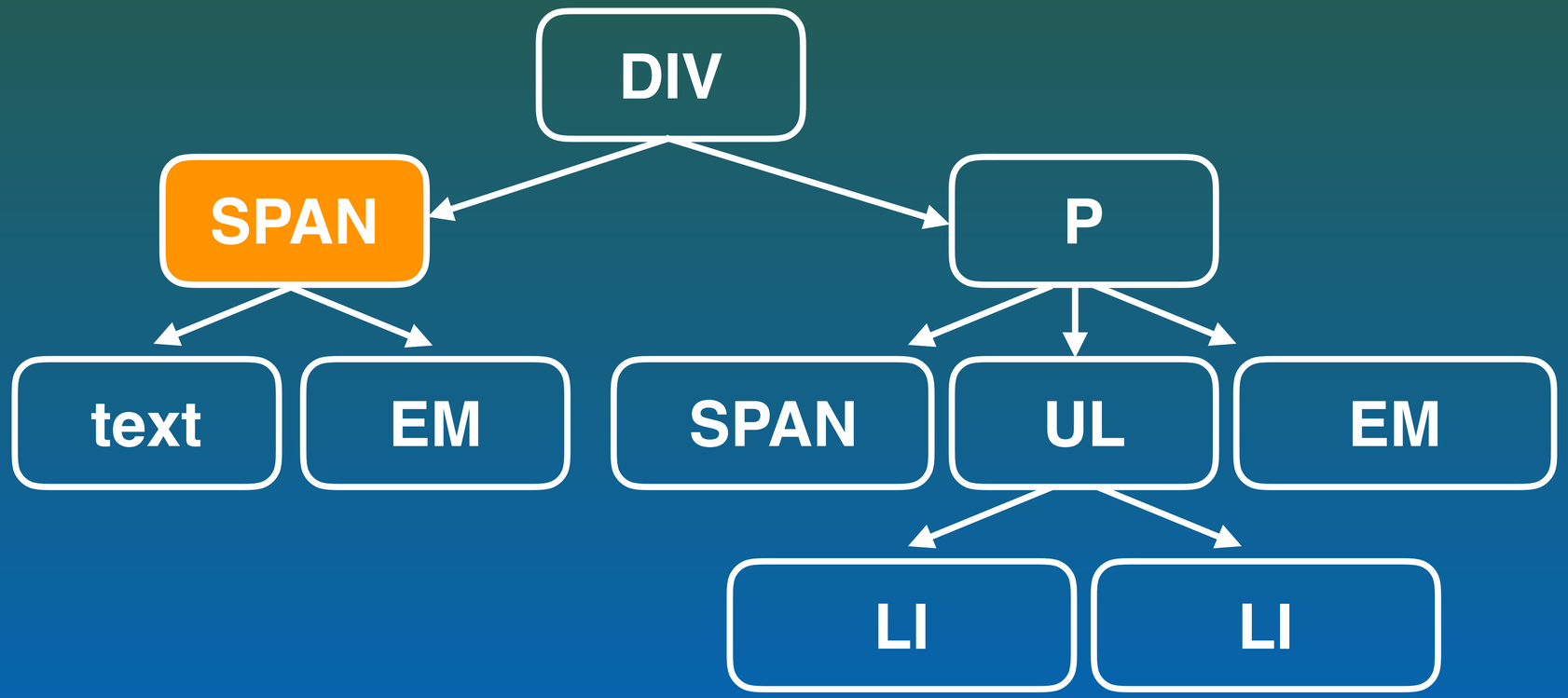
What selects: div em



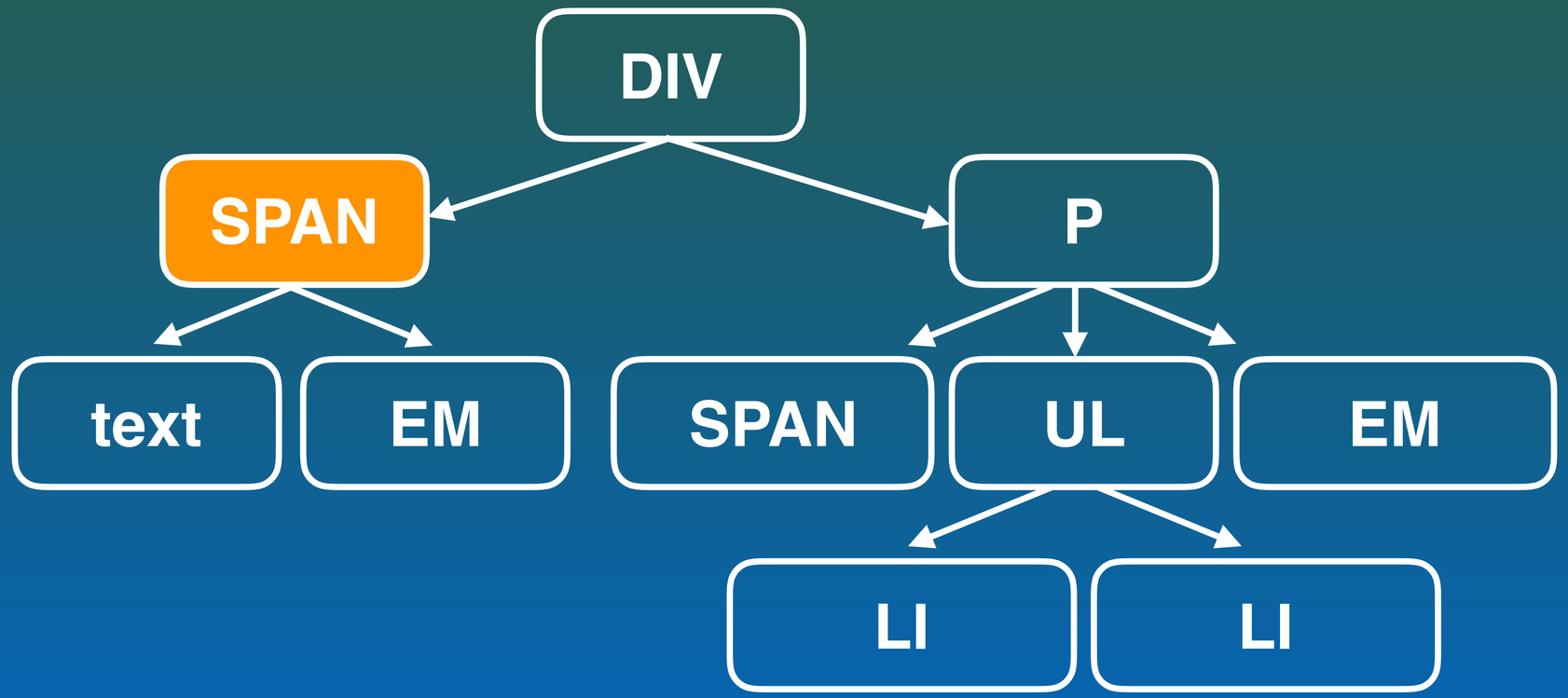
What selects: div em



What selects: _____



What selects: div > span



CSS Inline

HTML

```
..
<p style="color:blue;">
..
</p>
..
```

Style Element

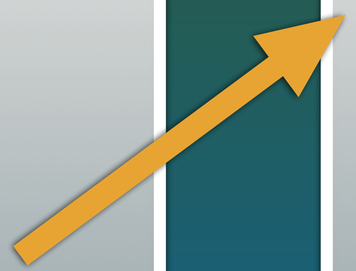
HTML

```
<head>
 <style>
  p { color:blue; }
 </style>
</head>
```

Link Element

HTML

```
<head>
 <link
   rel = "stylesheet"
   href = "style.css">
</head>
```



style.css

```
p { color:blue; }
```

CSS Import

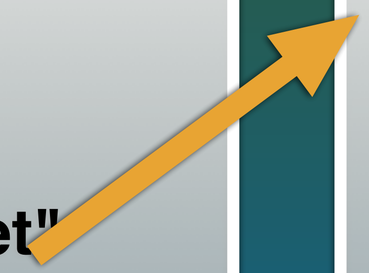
HTML
<head>
 <link
  rel = "stylesheet"
  href = "style.css">
</head>

style.css

@import
"common.css"

common.css

p { color:blue; }



Cascade Rules of Thumb

DOM inheritance *for some properties*

Sheet importance *author > user > browser*

Specificity *id > class > element*

Count ☐ ☐ ☐

Conflict Resolution

Source order

```
p { color: blue; }
p { color: red;  }
```

latest wins,
unless important

```
p { color: blue !important; }
p { color: red;  }
```

Cascade Rules of Thumb

Source order *latest wins*

Note:

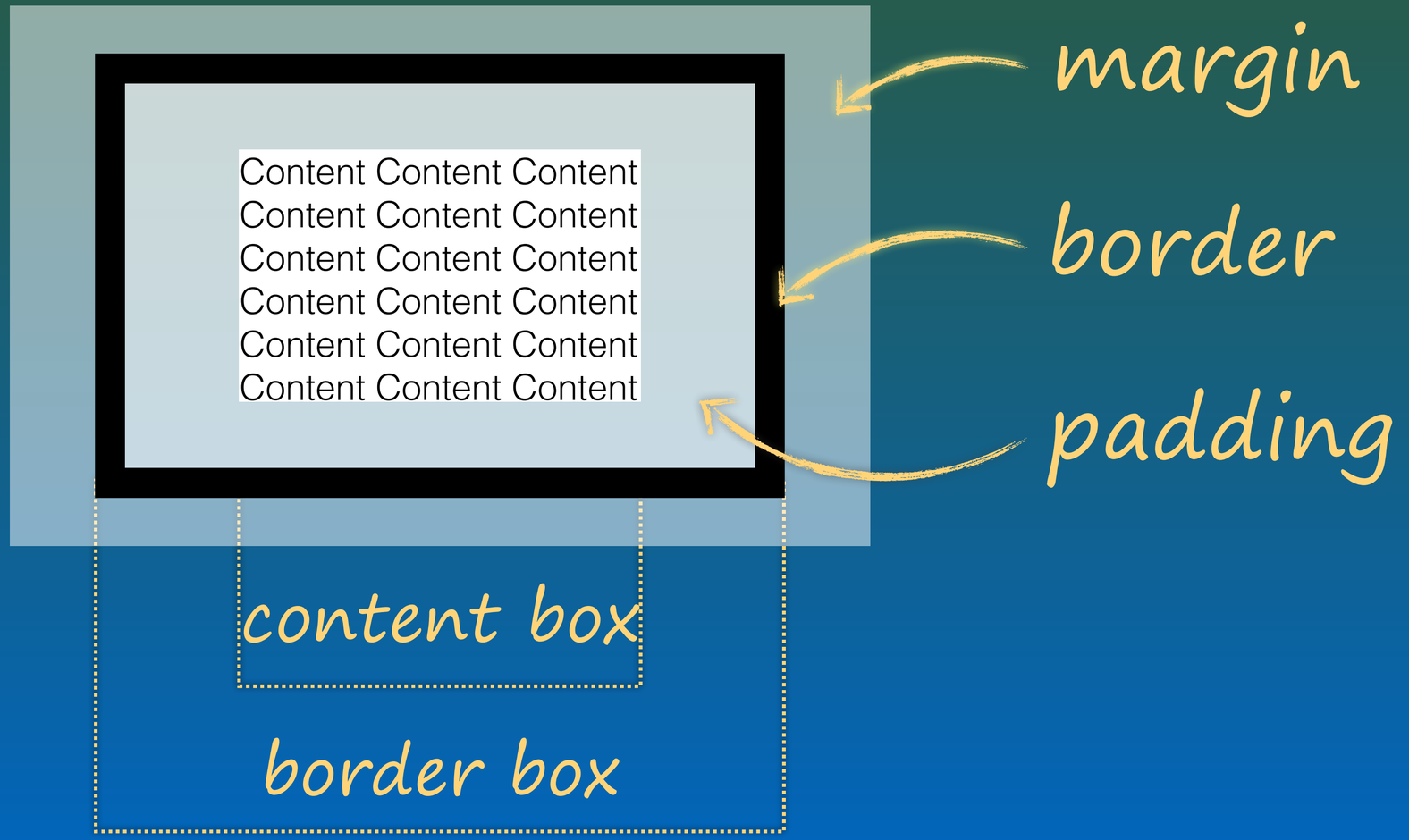
`p { color: blue !important; }` *overrides all the above!*

Properties

There are too many properties and possible values to list here.

We can only cover how it works *in principle*.

Box Model


Content Content Content
Content Content Content
Content Content Content
Content Content Content
Content Content Content
Content Content Content
margin
border
padding
content box
border box

Margins collapse

Content Content Content
Content Content Content
Content Content Content
Content Content Content
Content Content Content
Content Content Content

Content Content Content
Content Content Content
Content Content Content
Content Content Content
Content Content Content
Content Content Content

Content Content Content
Content Content Content
Content Content Content
Content Content Content
Content Content Content
Content Content Content

Units

px	*pixel*
cm, mm, in, pt	*absolute length*
em, rem	*size of "M"*
%	*relative to parent size*

Functions

calc(100% / 12)	*formula*
attr(*name*)	*attribute value*
url("backg.jpg")	*external reference*

Further Topics: layout

display	*inline, block, none,…*
position	*static, relative,*
	absolute, fixed
float, clear	*left, right*

Further Topics: color

red, blue, …	*color names*
#FF0000	*hex values*
rgba(255,0,0,0.5)	*opacity (alpha)*
linear-gradient()	*direction, color stops*
radial-gradient()	*shape, color, position*

MVC

Web Engineering

Dierk König

Testing

Is a core tenet of engineering.

Provides a validatable specification as opposed to "just give it a try".

grails test-app

Functional integration tests to validate the behavior as seen by the user.

Unit tests to validate controllers, models, and services in isolation.

Geb and Spock

Geb: user interactions with the HTML page

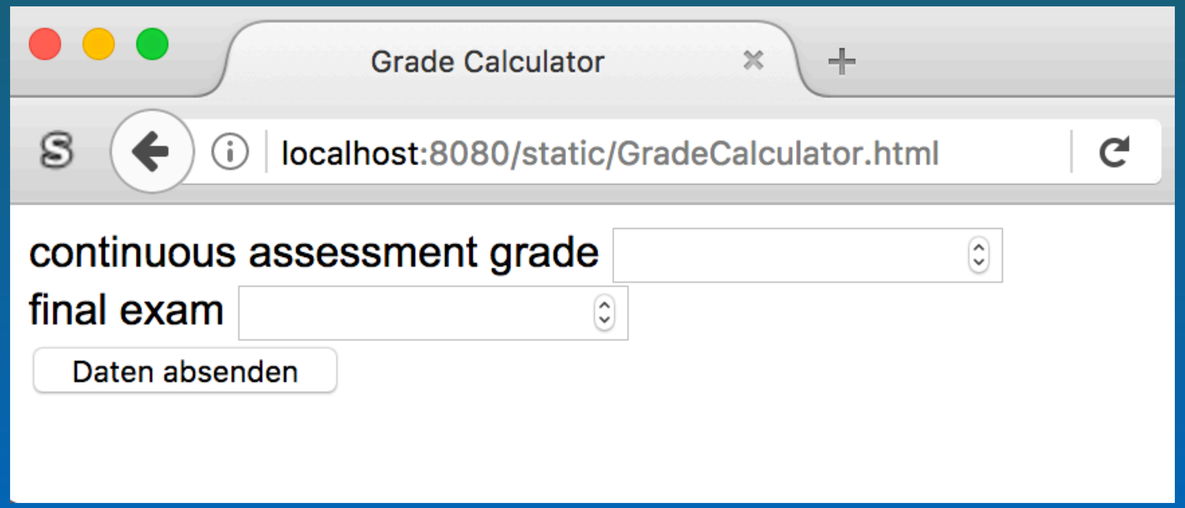
Spock: structure test cases in given - when - then - expect and allow data-driven tests

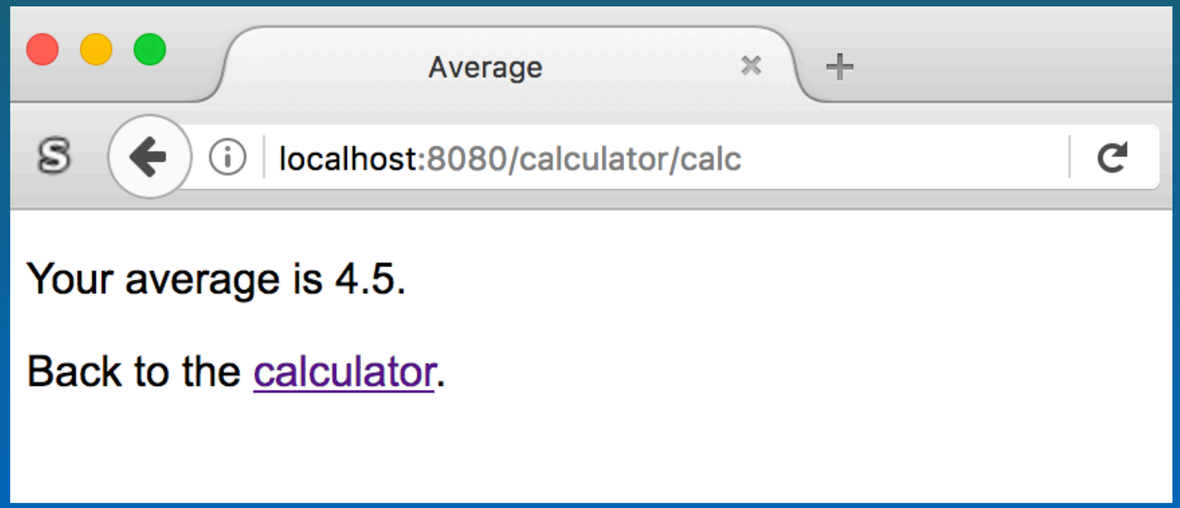
Geb selector $()

$("div.cool")	* *
$("a", href:"x")	**
$("a", text:"x")	*<a>x*
$("form").en	*<input name="en">*

Web View-Controller

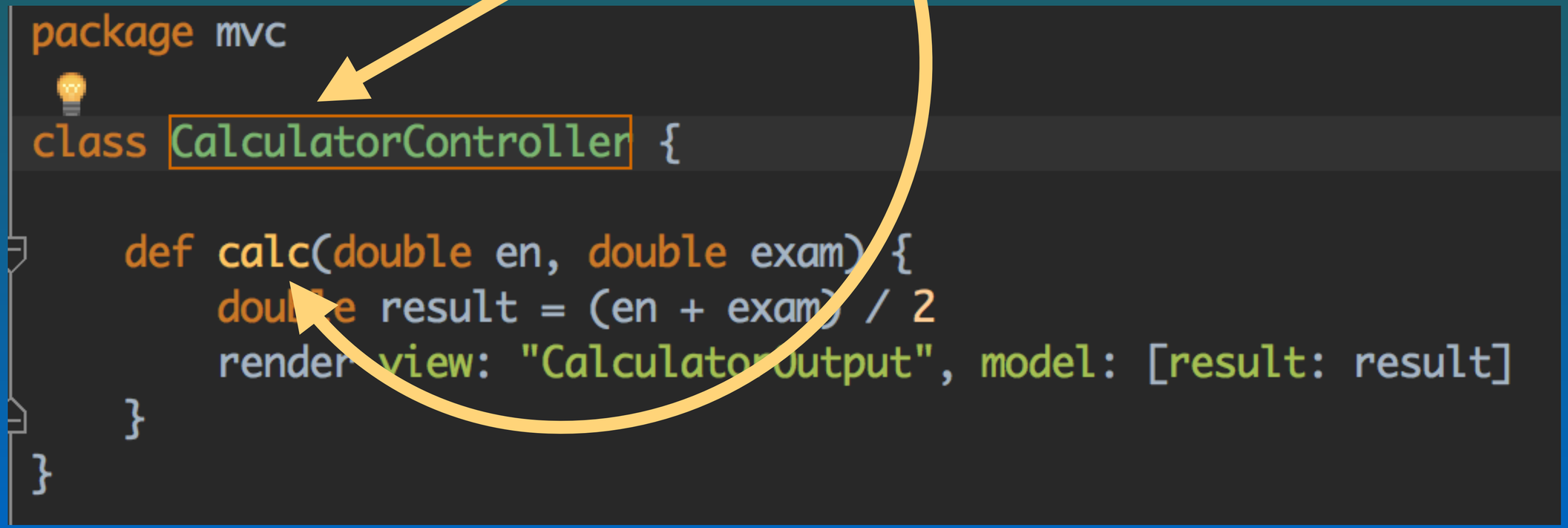
```
class MyController
  def myAction(en, exam)
```





Dispatch

/calculator/calc



View Selection

```groovy
package mvc

class CalculatorController {

    def calc(double en, double exam) {
        double result = (en + exam) / 2
        render view: "CalculatorOutput", model: [result: result]
    }
}
```

views/calculator/
CalculatorOutput.gsp

View Binding

```
package mvc

class CalculatorController {

    def calc(double en, double exam) {
        double result = (en + exam) / 2
        render view: "CalculatorOutput", model: [result: result]
    }
}
```

views/calculator/
CalculatorOutput.gsp

GSP View

use of view binding

```
<p> Your average is <output>${ result }</output>.</p>

<p> Back to the <a href="/static/GradeCalculator.html">calculator</a>.</p>
```

Web MVC

data binding

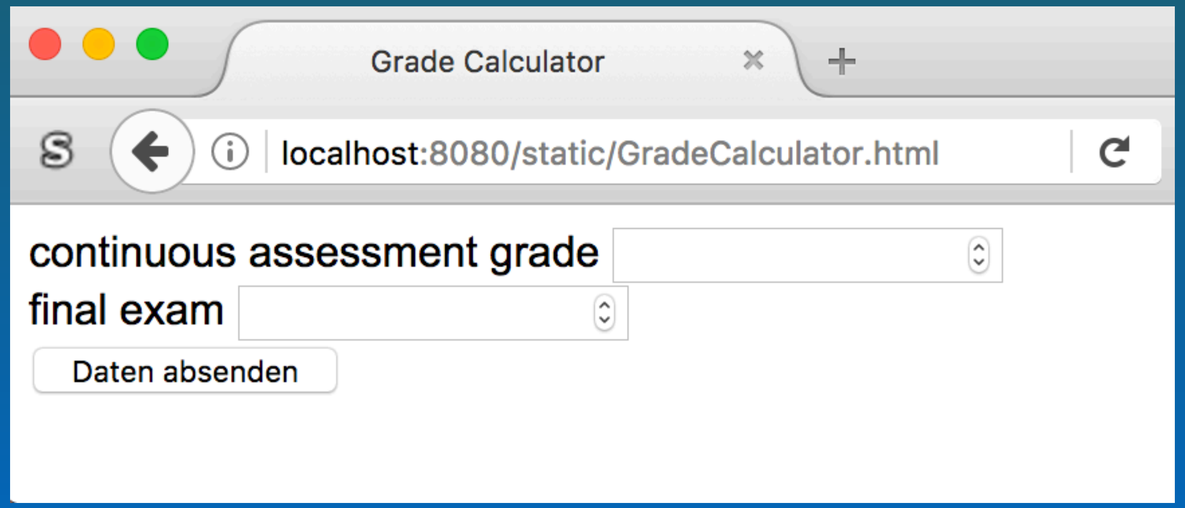
dispatch

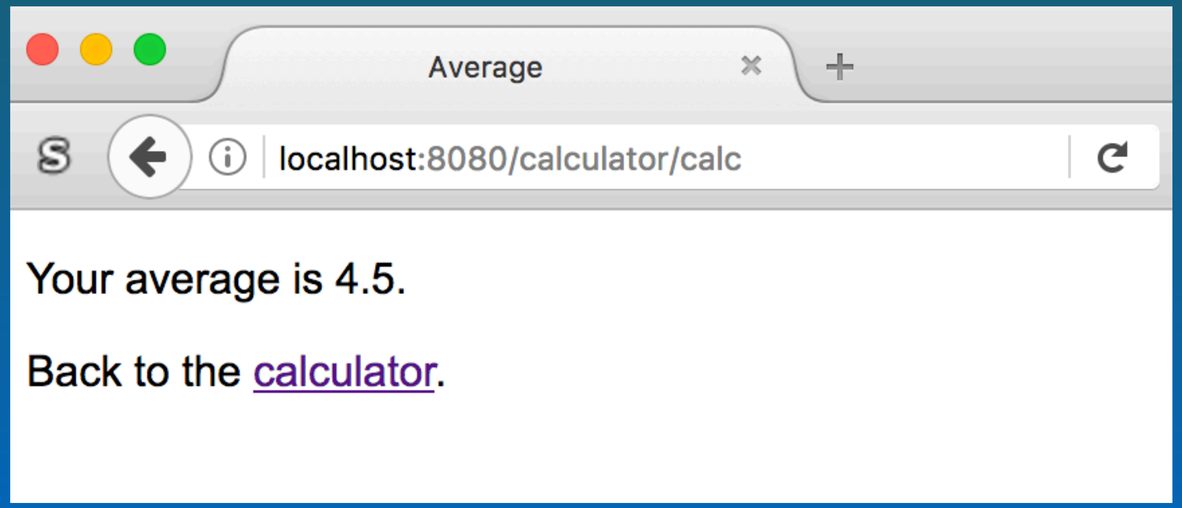
class MyModel
def en, exam, result

class MyController
def myAction(model)

view binding

view selection





Validation

In the view (trust: never!)

In the controller (imperative)

In the data binding (declarative)

Place of declaration: model constraints

Server Pages

Web Engineering

Dierk König

Technology Overview

Grails	***Web app platform***
Groovy	*"PHP" for Java*
Geb & Spock	*Test support*
Gradle	*Build system*
Spring Boot	*Enterprise Web*

The story so far

Static Pages - HTML, CSS

MVC - Model, View, Controller

Static Page

Server Page

Engineering Aspects

Well-formed & valid HTML, CSS

Avoid duplication (in CSS)

Testing pages and navigation

Validating Models (imp., decl.)

MVC separation of concerns

Request - Response

server

controller

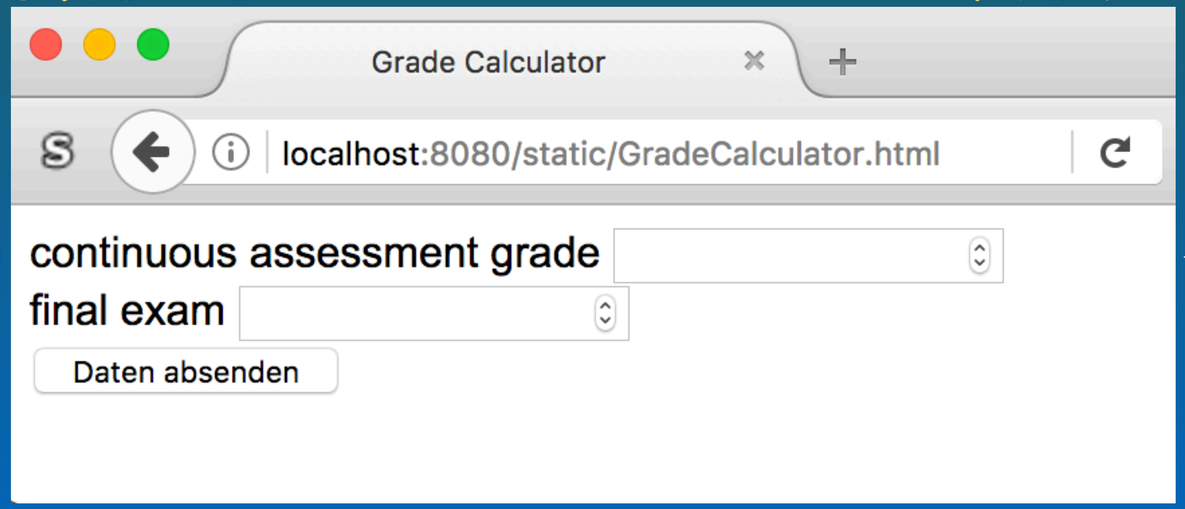
```
class MyController
  def myAction(en, exam)
```

request

response

client

view



Server Page 1: GSP

use of view binding

```
<p> Your average is <output>${ result }</output>.</p>

<p> Back to the <a href="/static/GradeCalculator.html">calculator</a>.</p>
```

Server Page 2: Template

gsp

```
<tmpl:form_row name="en"    label="En"    calculatorInstance="${calculatorInstance}" />
```

_form_row.gsp

```
<label for='${name}'>${label}</label>
<input type="number decimal" name="${name}" value="${calculatorInstance.getProperty(name)}"
       required="true" min="1.0" max="6.0" id="${name}"
```

Templates

static/structural

Parts of a page - as a page

Avoid in-page duplication

Sep. of concerns - abstraction

Compositional

Server Page 3: Taglib

gsp

```
<mvc:decorate grade="${calculatorInstance.result}">
  <output>${calculatorInstance.result}</output>
</mvc:decorate>
```

DecorationTagLib

```
def decorate = { attributes, body ->
    String grade = attributes.grade
    // ...
    out << body()
```

TagLibs

dynamic content

Parts of content - as a method

Can enclose a body

Can be used inside tags

All Grails tags are so defined

Templates & TagLibs

either can be used

as tag

as method

Server Page 4: Layout

gsp

```
<head>
  <meta name="layout" content="form"/>
```

layout/form.gsp

```
<h1><g:layoutTitle default="Form"/></h1>

<g:layoutBody/>
```

Layouts

Imposed embedding - as a page

Inverse composition (sitemesh)

Avoid duplication

Improve consistency

Allow multi-tenant apps

Page structure: four ways

GSP	*Server page with values*
Template	*Local composition*
TagLib	*Global composition*
Layout	*Inverse composition*

JavaScript

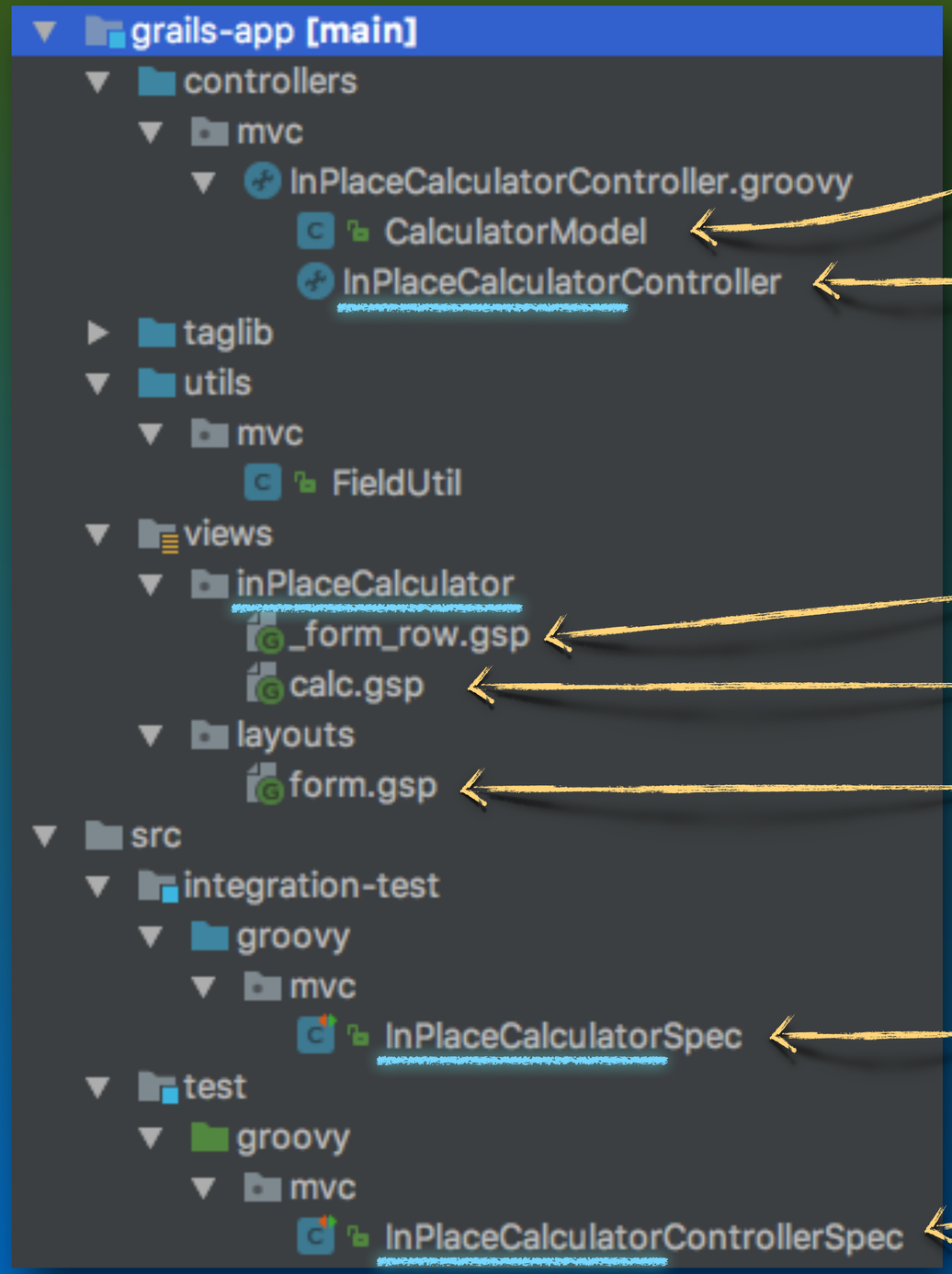
Web Engineering

Dierk König

Technology Overview

HTML	*Validator*
CSS	
Web MVC	*Unit Testing*
Server Pages	*UI Testing*
Javascript	*UI Testing*

Overall Structure



model
controller

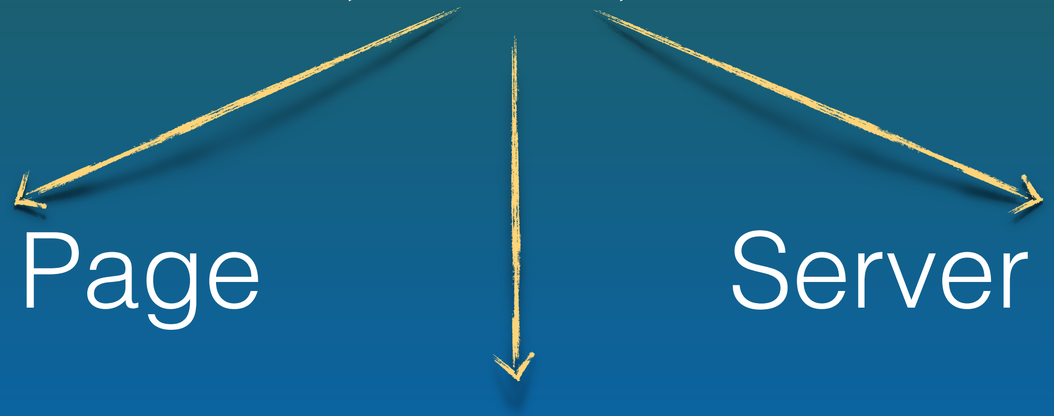
template
server page
layout

UI test
unit test

The story so far

Static Pages - HTML, CSS

MVC - Model, View, Controller



Static Page

Server Page

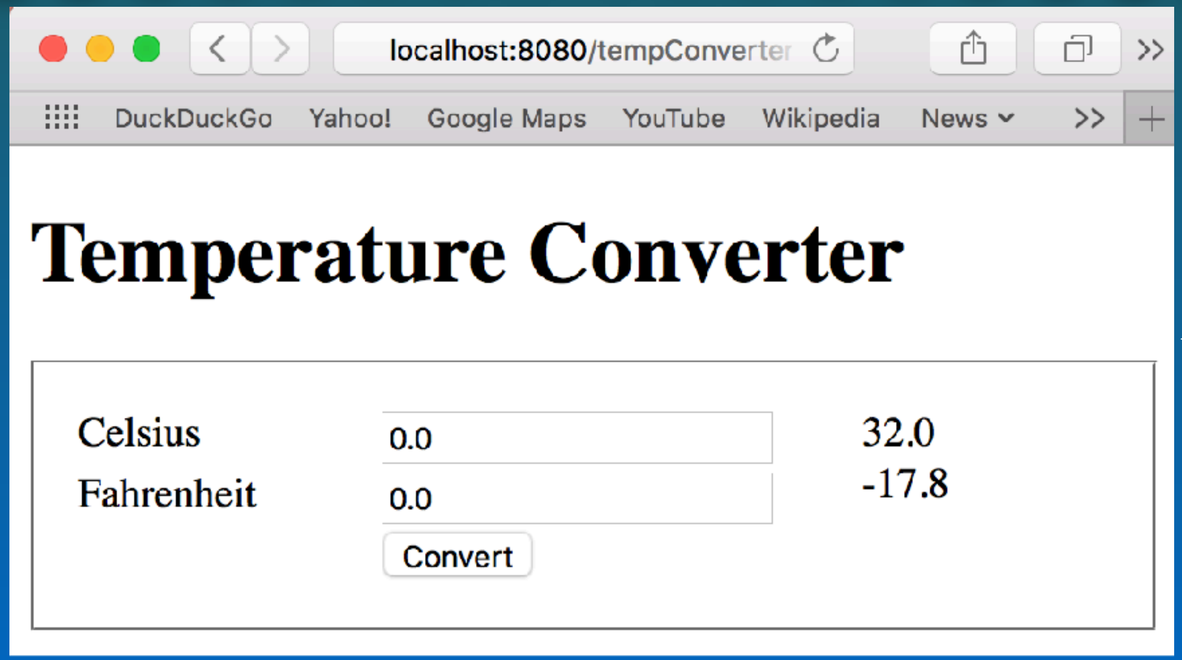
Dynamic Page

Request - Response

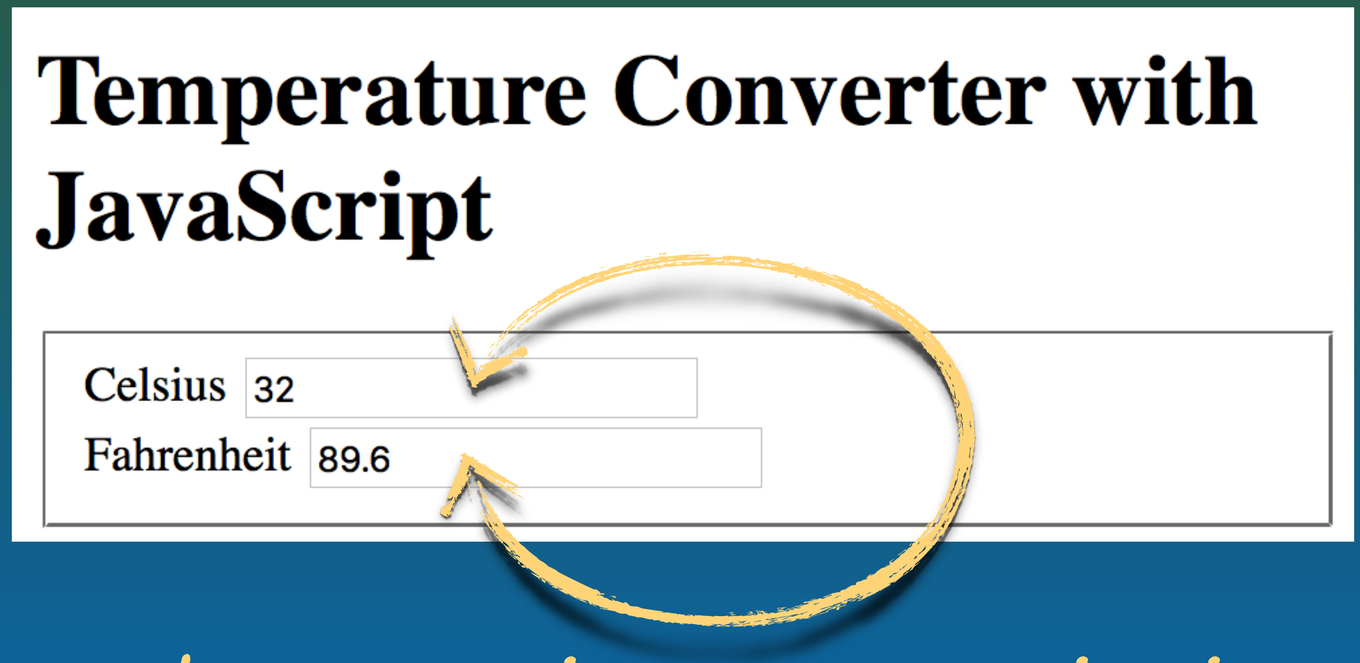
code as methods

```
class MyController
   def myAction(model)
```

strings



Direct Manipulation



code as strings == scripting

JavaScript

Code as string

HTML attribute value

Text content of <script> element

External .js file

Event Attributes

onClick,

onMouseOver,

onChange,

onInput,

…

JS Document

document.write(*html*);

document.getElementById(*id*);

document.querySelector(*selector*);

…

JS Element

element.id

element.value = *newValue*;

element.innerHTML = *newContent;*

element.classList.add(*newStyle*);

…

JS Function Declaration

keyword

function name

parameter names

```
function times(x, y) {
  return a * b;
}
```

value returned

Engineering Aspects

Where to put JS code:

in-line only for one-liners

in-page for local functions

.js files for cross-page sharing,
unit testing, linting, tool support, …

JS Topics not covered

classes, objects, types, literal decl.

scoping, prototype inheritance

function expressions, self-invocation

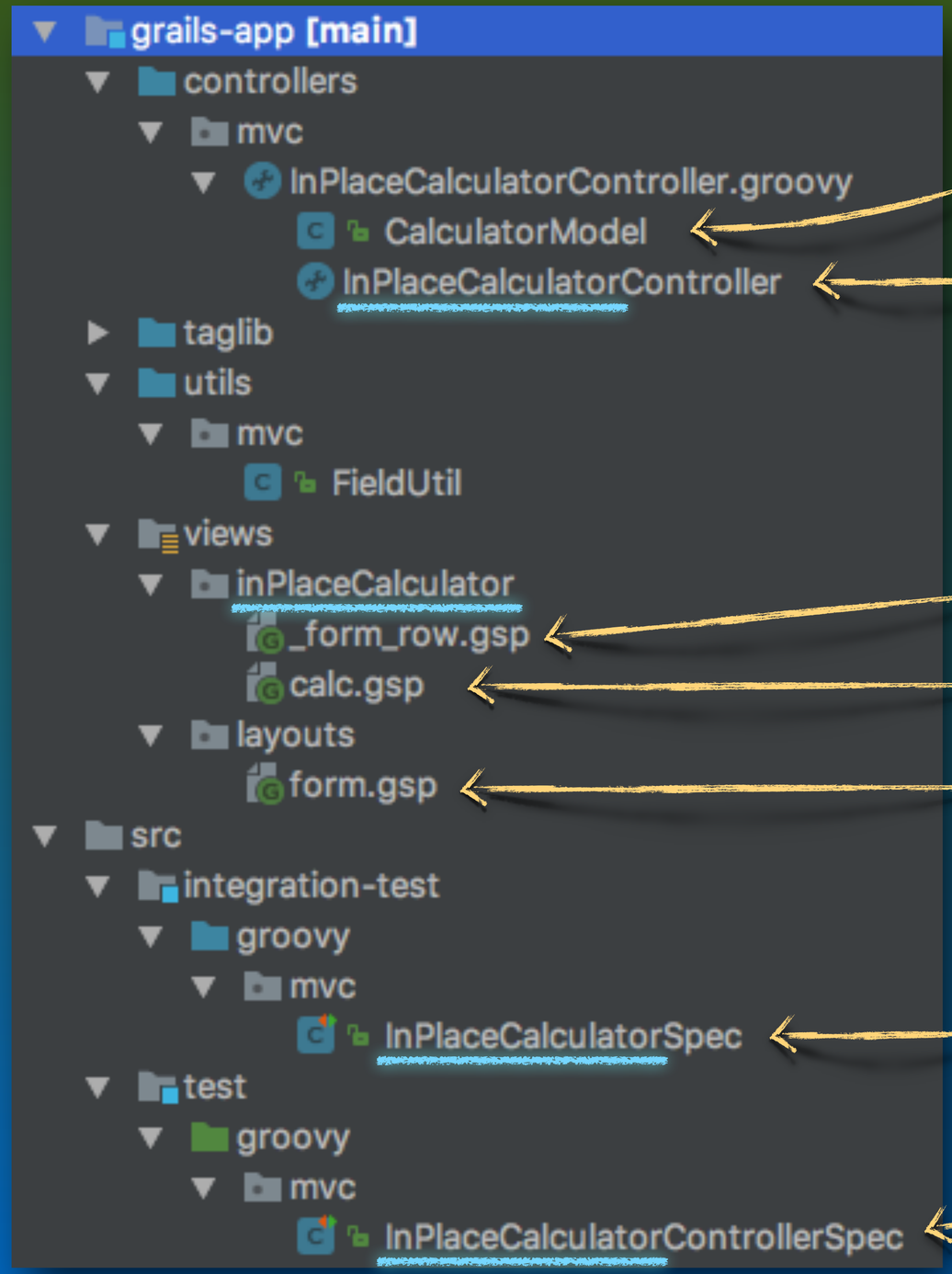
hoisting, linting, unit testing, …

Persistent State

Web Engineering

Prof. D. König

Overall Structure

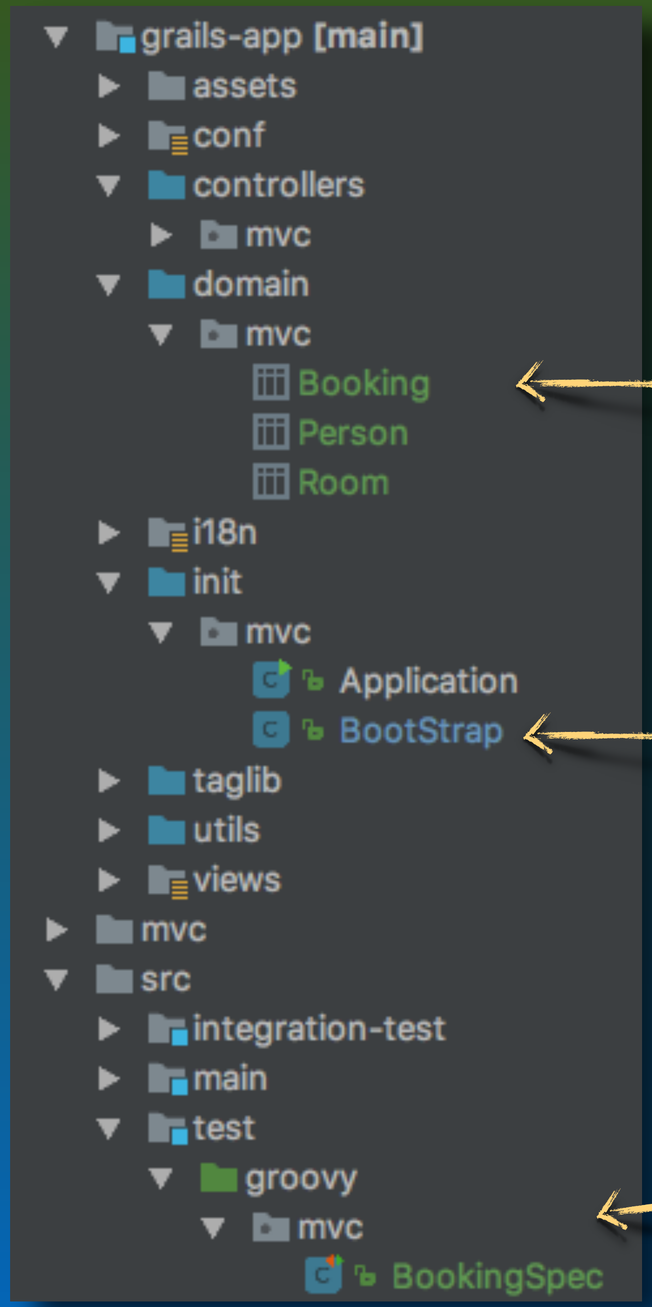


model
controller

template
server page
layout

UI test
unit test

Overall Structure



domain model

bootstrap

unit test

Domain Classes

```
package mvc

class Room {

    String description
    int    capacity

    String toString() { description + "(" + capacity + ")" }

    static constraints = {
    }
}
```

CRUD Methods

```
new Room(description: "5.3A17", capacity: 40).save()
Room.list()
Room.findAllByCapacityGreaterThan(20)
def firstRoom = Room.get(1)
firstRoom.delete()
```

Grails ORM

http://docs.grails.org/latest/guide/GORM.html

Domain classes serve as DAO/DTO/Model

DB is set up automatically

Dynamic finder methods simplify usage

Keep it simple

Scaffolding

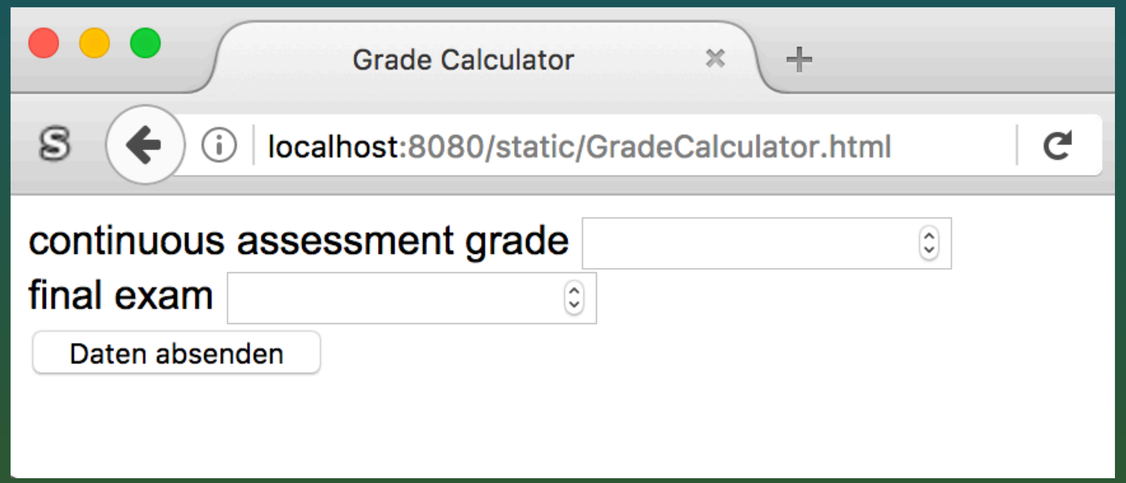
static scaffold = *domainClassName*

Controller actions and views are created transparently behind the scenes

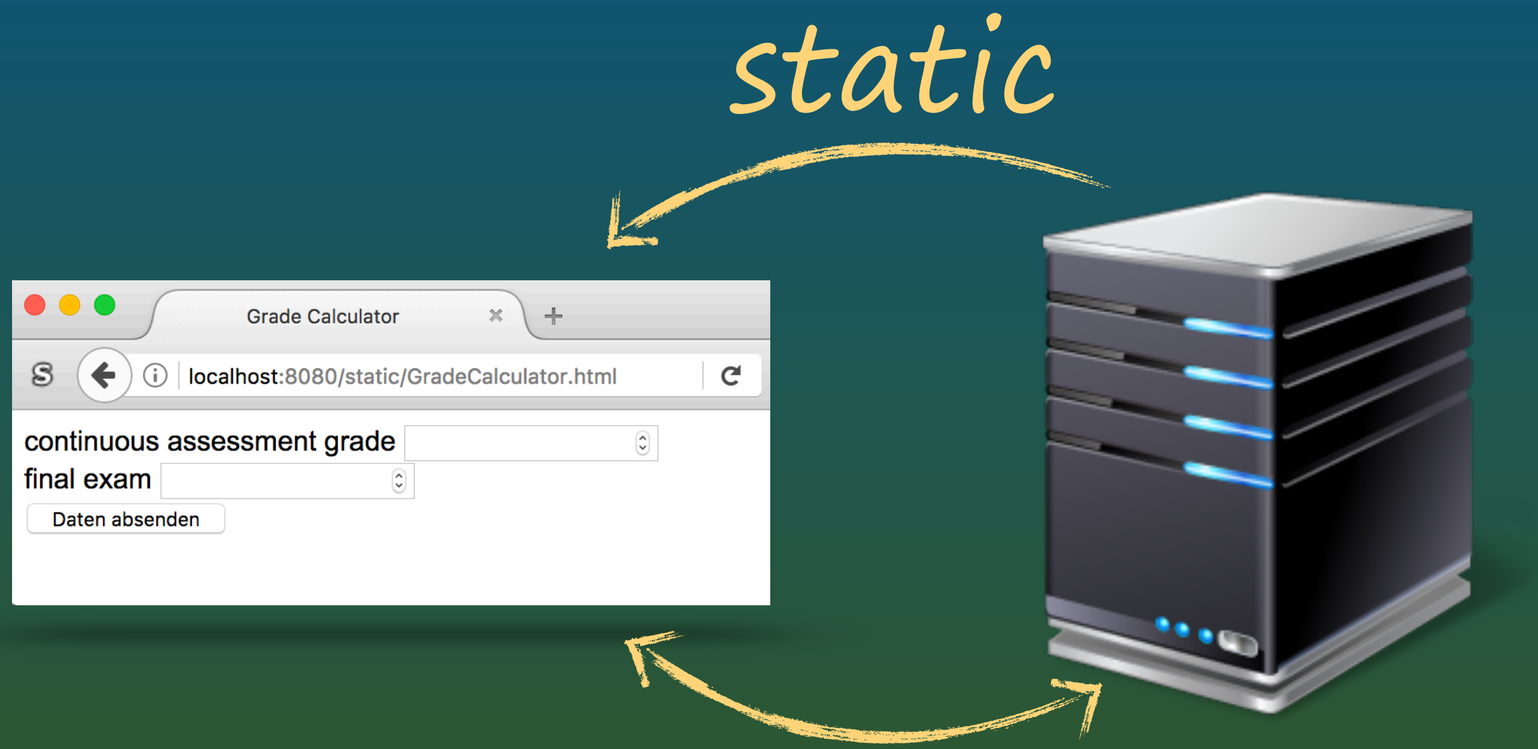
We can selectively override

HTML, CSS, JavaScript

static *dynamic*



Web MVC, Server Pages

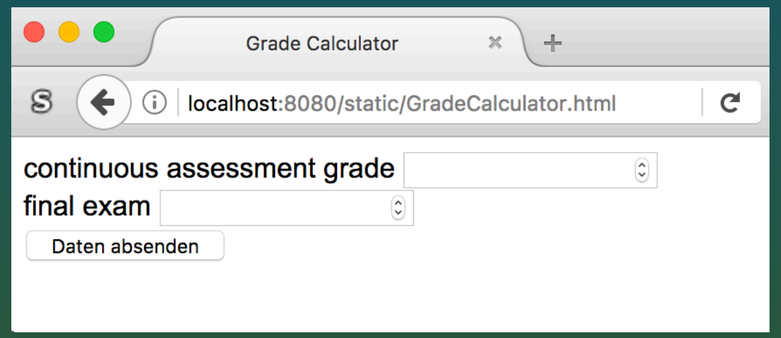


Three Tier Web Applications

Client

Server

DB



persist

Where State Lives

HTML static	*DOM, forms*
HTML dynamic	*JS, localstorage*
Server static	*Deployment*
Conversation	*Session*
Persistent	*Database*

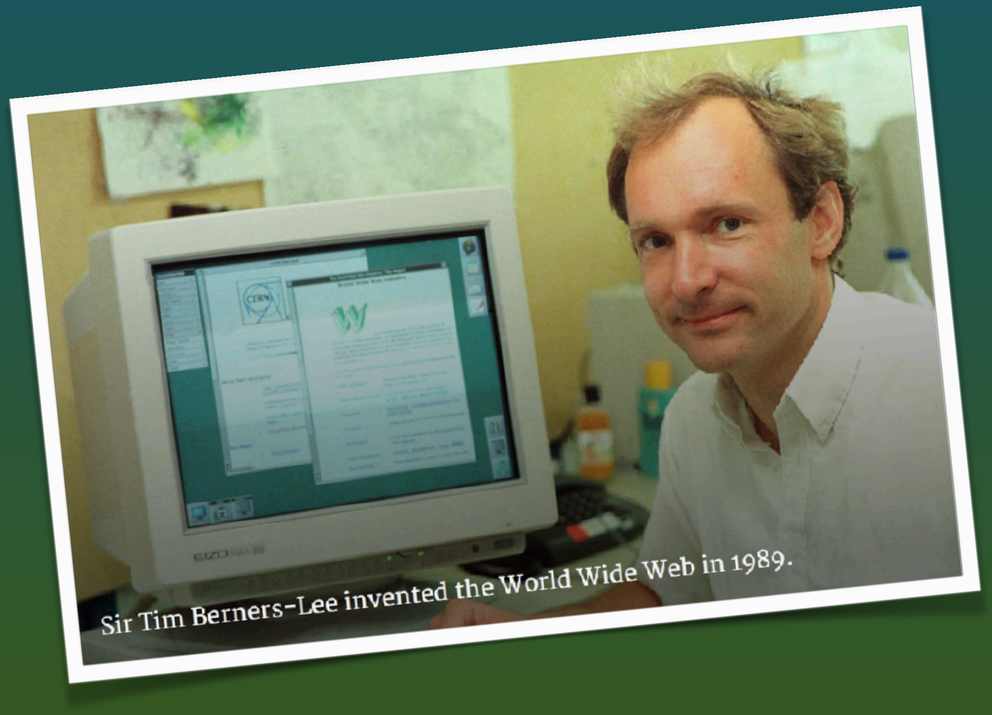
Services

Web Engineering

Prof. D. König

Web Resilience

The Web has never gone down
since its inception
in 1989.



Web Resilience

BE RESTRICTIVE
WHAT YOU EXPOSE

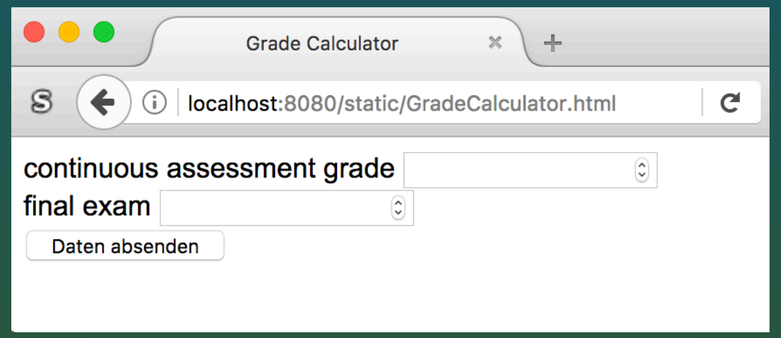
BE TOLERANT
WHAT YOU CONSUME

Three Tier Web Applications

Client

Server

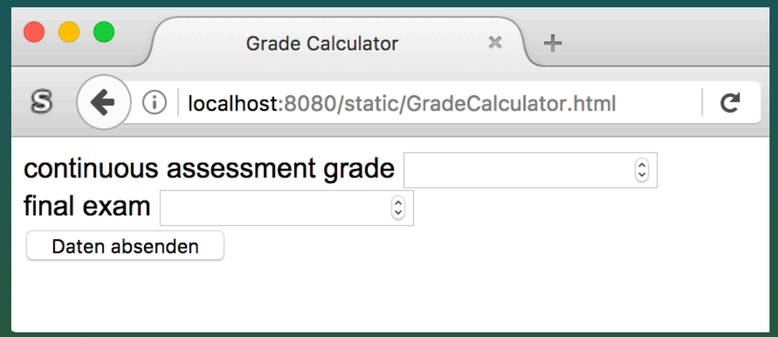
DB



persist

Server with Service

Client



Server

Service

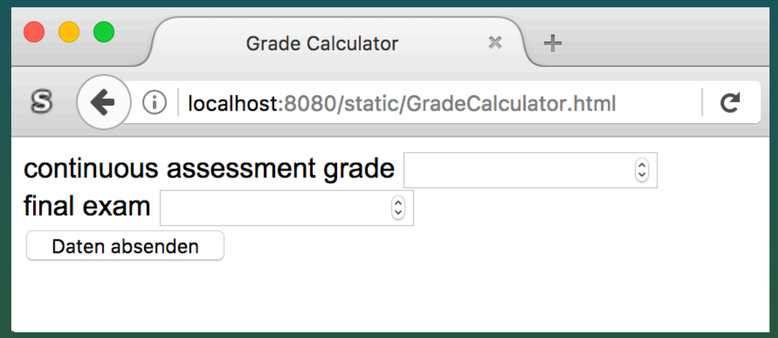
req/resp

"Smart UI" with Service

Client

Server

Service



req/resp

URL

Uniform Resource Locator

protocol, host, port, path, query, fragment

scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]

HTTP(s)

Hyper Text
Transfer Protocol

Very simple

Line/delimiter based

```
# First request
GET /quote?symb=YHOO HTTP/1.1
Host: www.example.org

HTTP/1.1 200 OK
Content-Type: text/plain;charset=UTF-8

15.96

# Second request 10 minutes later
GET /quote?symb=YHOO HTTP/1.1
Host: www.example.org

HTTP/1.1 200 OK
Content-Type: text/plain;charset=UTF-8

16.10
```

HTTP Verbs

Verb	Change	Use
GET	No	Single or collective read, may be cached
PUT	Yes	Modify resource in place
POST	Yes	Create new resource
DELETE	Yes	Delete resource
HEAD	No	
OPTIONS	No	

REST

Representational State Transfer

- Resource addressing by URL
- Uniform operations by HTTP verbs
- Self-containment (no conversational state)
- Choice of format (XML, JSON, …)

REST Issues

Easy to get started - easy to get lost.

Type safety, Versioning, Documentation

Addressing Operations as Resources

REST Alternatives

SOAP (WSDL),

CORBA (IDL),

EJB, *-RPC, RMI,

local services

REST in Grails

http://docs.grails.org/latest/guide/webServices.html

Note the usage of @Resource in domain classes and "respond" instead of "render"

in the controller actions.

The Mobile Web

Web Engineering

Prof. D. König

Responsive Design

Respond to varying screen sizes/resolutions

Prerequisite Knowledge:

HTML, CSS, JavaScript, Web-MVC

Testing the WebMile

The variety of devices, screen sizes, and resolutions makes any testing difficult.

Provider and products change quickly: google, microsoft, saucelabs, mobiready, …

Approaches

Flexible layout (CSS)

Media queries (CSS)

Dynamic in-page logic (HTML, JS)

Serve different views per capability (MVC)

Media Query (CSS)

```
#title { width: 50%; }

@media screen and (max-width: 800px) {
    #title { width: 100%; }
}
```

Media Query Attributes

max-width, max-device-width,
min-width, min-device-width, *(height)*
orientation (portrait, landscape),
[min-,max-,device-]aspect-ratio,
color, resolution, touch-enabled, …

Dynamic in-page logic

```
<body onresize="adapt()">


<script>
   function adapt() { …; screen.size … }
</script>
```

View per Capability

How to detect the capability?

How to change the view?

Request - Response

server

controller

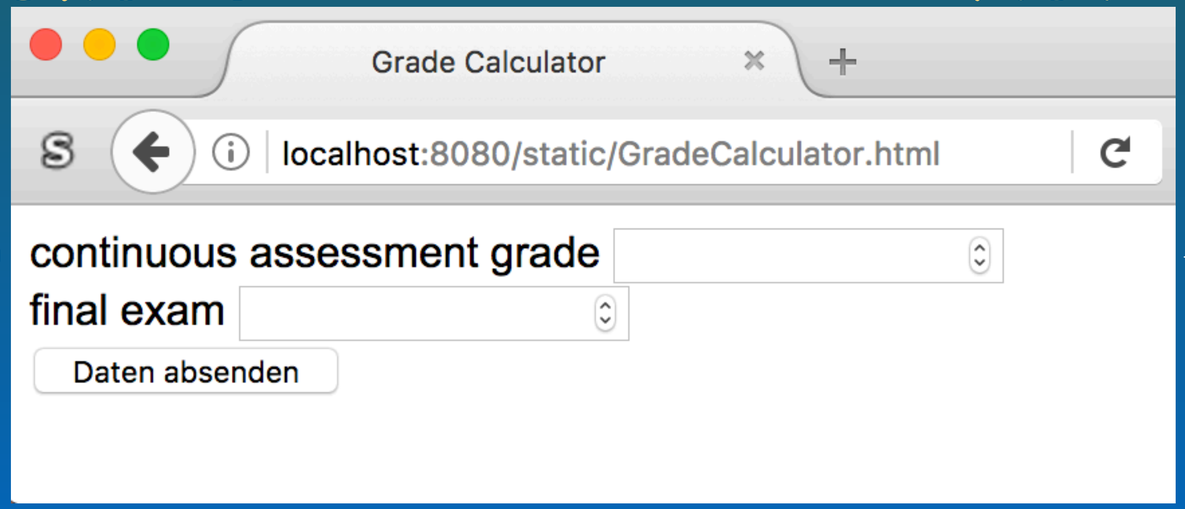
```
class MyController
  def myAction(en, exam)
```

request

response

client

view



How to change the View?

Use different CSS

Select a different layout

Rendering a different view

When to use: rule of thumb

CSS float	*Always consider*
@media	*Mostly static content*
onresize	*Fine-grained control*
MVC	*Default*

Web Deployment

Web Engineering

Prof. D. König

Deployment Option 1

grailsw war

upload war file to Java web server,
e.g. Tomcat

Deployment Option 2

gradlew stage

upload self-starting application to a Platform-as-a-Service (PaaS) provider,
e.g. Heroku

Web Atrocities

- Care for your data (download, backup)
- Data harvesting
- Data protection
- Forgetful data

Web Atrocities

- Denial of Service attacks, Spamming
- Proper Password handling, encryption
- Logging, Monitoring
- Localization
  formats, timezones, character encoding