

ELECTRONIC COMMERCE

MARCIN SKUBISZEWSKI

What this course teaches:

- ▷ To understand the technologies behind electronic commerce
- ▷ To design and build electronic commerce applications

CONTENT

Fundamentals

- ▷ Notions about computers
- ▷ Introduction to the Internet
- ▷ Software architecture of electronic commerce servers

Building an Internet-based store

- ▷ Web pages
 - HTML
 - HTML forms
 - CSS
- ▷ Programming
 - Java
 - modularity (object-oriented programming)
- ▷ Organizing business-related data
 - relational databases

- ▷ The generation of dynamic web pages
 - Java Servlet
a programming environment for web-based applications
 - Jakarta Struts
an extension to Java Servlet
a method of organizing web-based applications

Other topics (introduced briefly)

- ▷ Security
- ▷ Software market
 - the Microsoft monopoly
 - free (open source) software
- ▷ The worldwide name resolution infrastructure
- ▷ Project management
 - hiring developers
 - development methods
 - modularity and clarity
 - testing

HOW THE COURSE IS ORGANIZED

General organization

- ▷ Course: 17 hours
- ▷ Computer lab: approx. 26 hours, in two groups
 - under Linux

The project

- ▷ In small groups (up to 4 students)
- ▷ Individual defense, individual evaluation
 - the contribution of everyone needs to be defined clearly

Exams

- ▷ *Fundamentals* in February
 - understanding the first part of the course
- ▷ *Database schema design*
 - practical skills in data organization and database design
- ▷ A final exam
 - everything not covered in the other two exams

Questions are welcome anytime

Feedback is strongly desired

Note: *The content of slides 7–58 is more complete than the knowledge required from students.*

The required knowledge corresponds with slides 7, 17–20, 24–32, 36–37, 41–43, 45–46, 49–56, 59–end.

HARDWARE,

SECTION 1:

BITS

Definition

- ▷ A variable that can take two different values
 - **0** or **1**
 - **false** or **true**: bits are often called *logical values*

Examples

- ▷ Electric wire:
 - 0V: value 0
 - 3V: value 1
- ▷ Small fragment of a compact disc:
 - hole: value 0
 - no hole: value 1

Outline of this section

- ▷ Why bits are used in computers
- ▷ Orders of magnitude: how many bits a computer manipulates
- ▷ How computers operate on bits
 - MOS transistors and logical gates
- ▷ How bits are used to represent information
 - numbers, text, images

WHY BITS

Example alternative: analog electronics

- ▷ Any value between 0V and 3V is permitted
- ▷ Errors accumulate (example: 1.722V becomes 1.727V, then 1.735V etc.)
- ▷ A lot of information in every signal
 - ⇒ much less signals, much less components than with binary electronics
- ▷ Often used, but
 - not in computers
 - not in modern, complex systems

Example alternative: ternary system

- ▷ 0V: value 0
- ▷ 1.5V: value 1
- ▷ 3V: value 2
- ▷ In principle: as good as binary, errors do not accumulate
- ▷ In practice: hard to implement (for physical/electronic reasons)
- ▷ Never used

ORDERS OF MAGNITUDE

Storage

- ▷ RAM (fast storage):
 - one billion bits in a cheap computer (128 megabytes)
- ▷ Magnetic disk:
 - 700 billion bits in a magnetic disk worth 150 EUR (80 gigabytes)

Processing speed

- ▷ A clock of more than 1 GHz in a cheap computer
 - one billion consecutive operations on bits per second
 - hundreds of operations in parallel

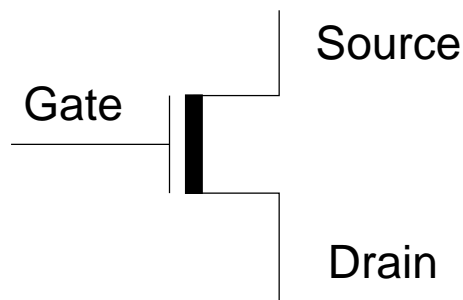
20 years ago

- ▷ RAM: 512 kilobits (64 kilobytes)
 - improvement factor: 2000
- ▷ Cheap magnetic disk (5½" floppy): 3 million bits (360 kilobytes)
 - improvement factor: 20 000
- ▷ A clock of 1 MHz in a cheap computer
 - improvement factor: better than 1000

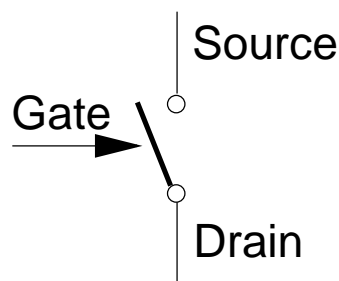
MOS (metal oxide semiconductor) TRANSISTORS

A voltage-controlled switch:

- ▷ connection between *source* and *drain* controlled by *gate* voltage
- ▷ the gate can
 - attract current carriers (electrons or holes)
- ⇒ create a conductive channel (a channel filled with carriers)
- ⇒ allow the current to pass



The symbol



What it does

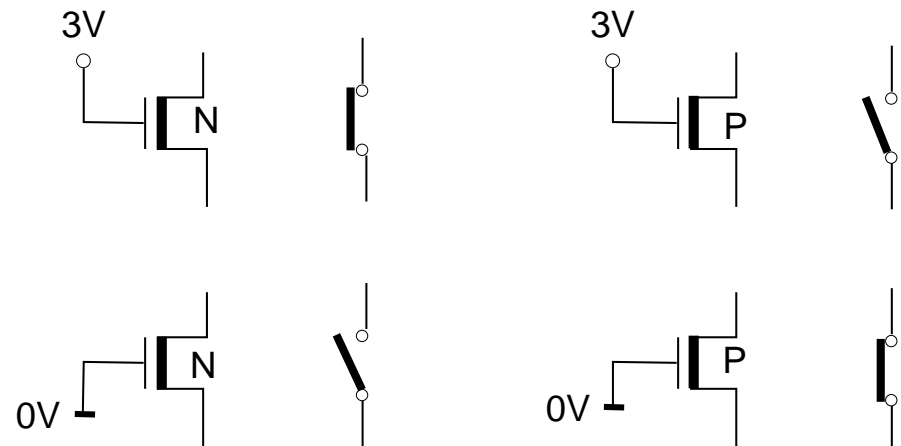
The two kinds of MOS transistors

Channel N:

- ▷ channel filled with negative carriers (electrons)
- ▷ positive voltage attracts electrons
- ▷ needs positive voltage (3V) to be conductive

Channel P:

- ▷ channel filled with positive carriers (holes)
- ▷ needs negative voltage (0V) to be conductive



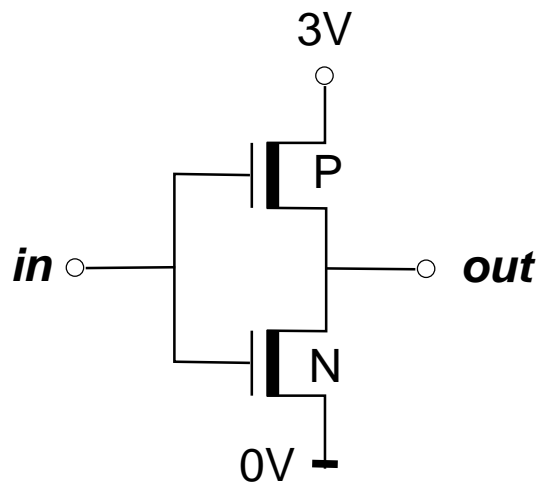
BOOLEAN OPERATIONS

- ▷ Operations on bits
- ▷ Bits are considered as truth values
 - 1: true
 - 0: false

NOT

p	$\neg p$
0	1
1	0

The NOT gate



BOOLEAN OPERATIONS

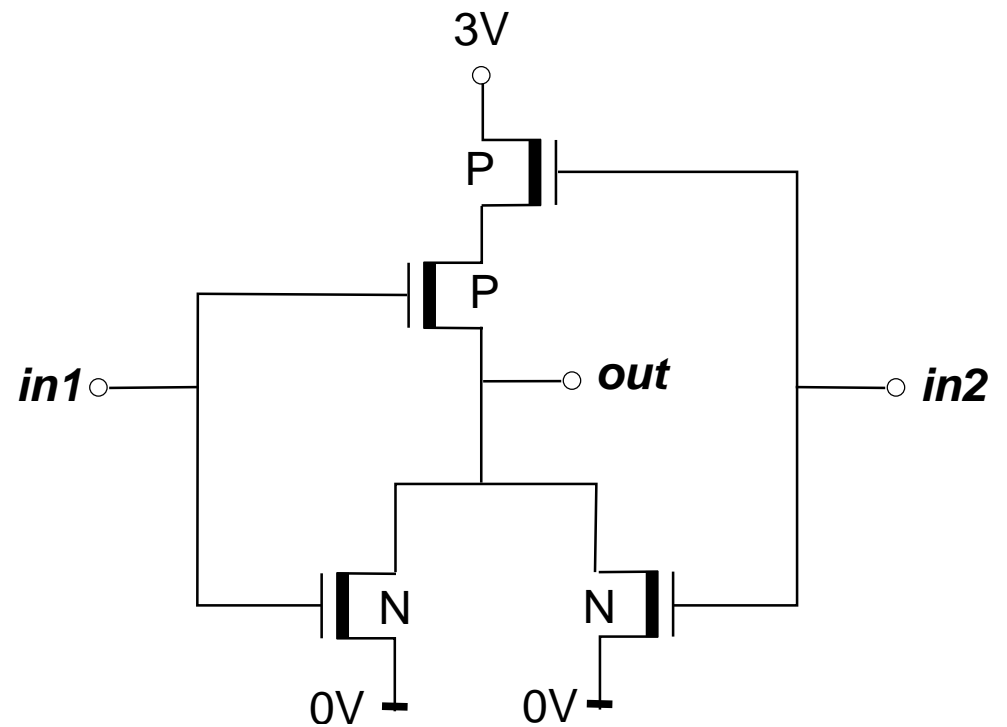
OR

p	q	$p \vee q$
0	0	0
0	1	1
1	0	1
1	1	1

NOR

p	q	$\neg(p \vee q)$
0	0	1
0	1	0
1	0	0
1	1	0

The NOR gate



REPRESENTING POSITIVE INTEGERS

Decimal representation

- ▷ Example: 2345
- ▷ Weights of digits:
 - $10 * 10 * 10 = 1000$
 - $10 * 10 = 100$
 - 10
 - 1
- ▷ Value: $2 * 1000 + 3 * 100 + 4 * 10 + 5 * 1 = 2345$
- ▷ Digits go from 0 to $10 - 1 = 9$

REPRESENTING POSITIVE INTEGERS

Binary representation

- ▷ Example: 1101
- ▷ Weights of digits:
 - $2 * 2 * 2 = 8$
 - $2 * 2 = 4$
 - 2
 - 1
- ▷ Value: $1 * 8 + 1 * 4 + 0 * 2 + 1 * 1 = 13$
- ▷ Digits go from 0 to $2 - 1 = 1$

Exercise: compute the value of 101010

THE KEY IDEA ABOUT BITS

Any kind of information processing can be expressed as a number of Boolean operations on bits

(often many billions of operations)

- ▷ Examples of information processing:
 - adding or multiplying numbers
 - selling through a website
 - playing music
 - movie editing

Example: image representation

- ▷ The image is decomposed into $n \times m$ dots (*pixels*, picture elements)
 - this image: 1024×768 pixels
- ▷ each pixel has three intensities: red, green, blue
- ▷ each intensity is stored as 8-bit number:

- 0 (min): no light
- 255 (max): max brightness

- ▷ this image: $1024 \times 768 \times 3 = 2359296$ numbers

Example: fading at the end of a movie

- ▷ a progressive reduction of brightness
- ▷ implemented as a multiplication of brightness by a decreasing number
 - first frame: 0.9
 - second frame: 0.8
 - last frame: 0
- ▷ the multiplication is implemented as a number of Boolean operations

ADVANCED TOPIC: ADDITION

The key idea

Example with 4 decimal digits: $415 + 777$

$$\begin{array}{r}
 \text{carry} \quad 1 \quad \quad 1 \\
 \quad \quad 0 \ 4 \ 1 \ 5 \\
 + \quad 0 \ 7 \ 7 \ 7 \\
 \hline
 = \quad 1 \ 1 \ 9 \ 2
 \end{array}$$

Example with 8 binary digits: $55 + 83$

$$\begin{array}{r}
 \text{carry} \quad \quad 1 \ 1 \ 1 \quad \quad 1 \ 1 \ 1 \\
 \mathbf{55} \quad \quad 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \\
 \mathbf{83} \quad + \quad 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \\
 \hline
 \mathbf{138} \quad = \quad 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0
 \end{array}$$

ADVANCED TOPIC: ADDITION

Adding two 4-bit numbers: $r := p + q$

$$\begin{array}{r}
 \text{carry} \quad \quad \quad c_3 \ c_2 \ c_1 \\
 p \quad \quad \quad p_3 \ p_2 \ p_1 \ p_0 \\
 q \quad \quad + \quad q_3 \ q_2 \ q_1 \ q_0 \\
 \hline
 r \quad \quad = \quad r_3 \ r_2 \ r_1 \ r_0
 \end{array}$$

$$r_0 := p_0 \oplus q_0$$

$$c_1 := p_0 \wedge q_0$$

$$r_i := (p_i \oplus q_i) \oplus c_i$$

$$c_{i+1} := (p_i \wedge q_i) \vee (p_i \wedge c_i) \vee (q_i \wedge c_i)$$

AND

p	q	$p \wedge q$
0	0	0
0	1	0
1	0	0
1	1	1

XOR

p	q	$p \oplus q$
0	0	0
0	1	1
1	0	1
1	1	0

$$p \oplus q = (p \wedge \neg q) \vee (\neg p \wedge q)$$

ADVANCED TOPIC: SIGNED INTEGERS

The most significant bit (MSB) is the sign bit

- ▷ MSB=1 when the number is negative
- ▷ this is not what you could think:
 - 11111111 represents –1
 - 11111110 represents –2
 - 11111101 represents –3
 - 10000000 represents –128

▷ details are not covered here

Addition and subtraction work as if the number were not signed

▷ Example: $13 + (-2)$

$$\begin{array}{r}
 \text{carry} \quad \quad \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \\
 \mathbf{13} \quad \quad \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \\
 \mathbf{-2} \quad \quad \mathbf{+} \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \\
 \hline
 \mathbf{11} \quad \quad \mathbf{=} \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1
 \end{array}$$

SIZES

#bits	#possible values
1	2
2	$2 * 2 = 4$
3	$2 * 2 * 2 = 8$
n	2^n
8	$2^8 = 256$
16	$2^{16} = 65536$
32	$2^{32} = 4294967296 \approx 4 \cdot 10^9$ (four billion)
64	$2^{64} = 18446744073709551616 \approx 1.8 \cdot 10^{19}$

The most common size: **32**

SIZES

8 bits:

- ▷ 256 values
- ▷ signed integers from -128 to 127

32 bits:

- ▷ 4294967296 values (around 4 billion)
- ▷ signed integers from -2147483648 to 2147483647

64 bits:

- ▷ 18446744073709551616 values
- ▷ signed integers from -9223372036854775808 to 9223372036854775807

The most common size: 32 bits

REPRESENTING CHARACTERS

Principle

- ▷ Characters are numbered
- ▷ Character numbers are represented using a specified number of bits
 - 7, 8, 16 or 21 bits per character

ASCII

- ▷ All characters common in North America
 - letters, digits, punctuation, dollar (\$) etc.
 - control characters
 - * formatting: linefeed, carriage return, form feed etc.
 - * bell
- ▷ 128 characters
- ▷ 7 bits (8 bits, with room left for *parity bit*)

REPRESENTING CHARACTERS

ISO-8859-*n*

- ▷ A family of encodings:
 - ISO-8859-15: for west-European languages
 - ISO-8859-2: for east-European languages with Latin alphabet

- ▷ Each encoding is a superset of ASCII:
256 characters, stored on 8 bits:
 - 128 ASCII characters
 - plus 96 extra characters specific to a group of languages
 - 32 character numbers are unused

The ISO 8859 alphabets:

- ISO 8859-1 west European languages (Latin-1) (old, deprecated)
- ISO 8859-2 east European languages (Latin-2)
- ISO 8859-3 southeast European and miscellaneous languages (Latin-3)
- ISO 8859-4 Scandinavian/Baltic languages (Latin-4)
- ISO 8859-5 Latin/Cyrillic
- ISO 8859-6 Latin/Arabic
- ISO 8859-7 Latin/Greek
- ISO 8859-8 Latin/Hebrew
- ISO 8859-9 Latin-1 modification for Turkish (Latin-5)
- ISO 8859-10 Lappish/Nordic/Eskimo languages (Latin-6)
- ISO 8859-11 Thai ISO 8859-13 Baltic Rim languages (Latin-7)
- ISO 8859-14 Celtic (Latin-8)
- ISO 8859-15 west European languages (Latin-9)

ISO-8859 Example

- ▷ **e**: number 101 in ASCII and in all ISO-8859 encodings
 - 8-bit encoding: 01100101

- ▷ **ê**
 - number 234 in ISO-8859-15
 - * 8-bit encoding: 11101010
 - cannot be expressed in ISO-8859-2

- ▷ **ë**
 - number 234 in ISO-8859-2,
 - * 8-bit encoding: 11101010
 - cannot be expressed in ISO-8859-15

ISO 8859: Discussion

- ▷ Advantages:
 - only 8 bits per character
 - superset of ASCII

- ▷ Problem: multiple encodings
 - ⇒ it is difficult to mix languages

- ▷ Current status:
 - widely used
 - declining

UNICODE

Principle

- ▷ More than $2^{20} \approx 1$ million character codes
- ▷ all characters in all languages are represented

Further encoding: UTF-8

- ▷ variable size
- ▷ Unicode codes are further encoded into bytes:
 - 8-bit codes from 0 to 127 stand for themselves, are interpreted in ASCII
 - 8-bit codes from 128 to 255 say
take into account this byte and a determined number of bytes that follow

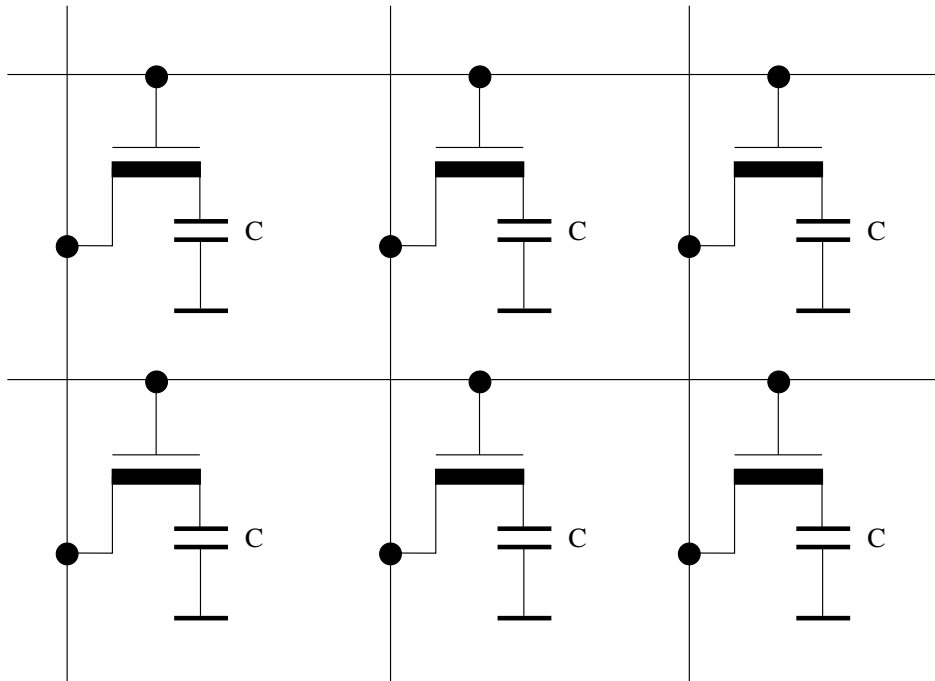
UTF-8 is the emerging standard on the Internet

HARDWARE COMPONENTS OF A COMPUTER

- ▷ Central memory (RAM)
- ▷ Central Processing Unit (CPU)
- ▷ Disks
- ▷ Cache memory
 - memory hierarchy involving disk, central memory, cache
- ▷ Other peripherals

DYNAMIC MEMORY

Dynamic memory cells



Reading a bit

- ▷ get row address
- ▷ get column address
- ▷ output a bit

DYNAMIC MEMORY

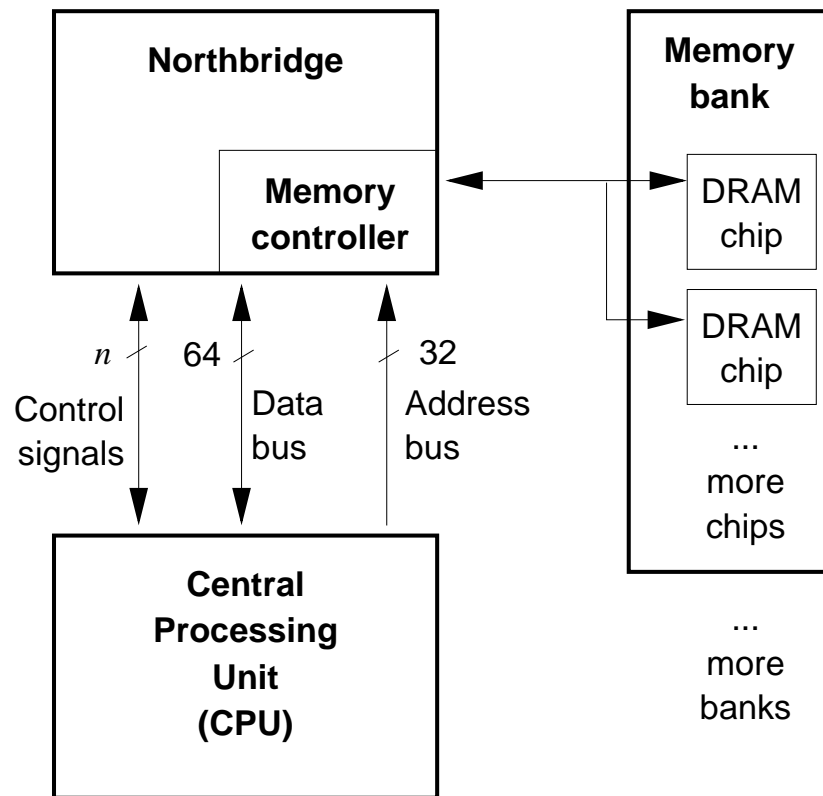
A modern chip

- ▷ 256 MBits (256 · 1024 · 1024 bits)
- ▷ 16 k columns, 16 k rows
- ▷ operation while reading:
 - get row address
 - then, get column address
 - then, output 8 bits
- ▷ clock: 400 MHz
 - 400 million elementary operations per second
 - 5 cycles to read 4 bits from a given address
 - 6 cycles to read 8 consecutive bits

A memory bank

- ▷ 8 memory chips working in parallel
 - 64 bits are delivered at once

THE OVERALL ARCHITECTURE



CENTRALIZED ARCHITECTURE

- ▷ Memory
 - stores massive amounts of information
 - does no processing
- ▷ CPU (central processing unit)
 - processes information
 - works sequentially
- ▷ Comparison: human brain
 - massively parallel
 - memory and processing are mixed
- ▷ Multiple CPUs
 - 2 to 4 CPUs in high-end PCs
- ▷ Other architectures exist
 - very expensive
 - special applications

CENTRALIZED ARCHITECTURE: EXAMPLE

Darken a 24-megapixel color image, stored using $3 \cdot 16$ bits per pixel:
for each pixel, divide by 2 three 16-bit numbers

- ▷ each of the $3 \cdot 24$ million numbers is
 - read from memory into the CPU
 - divided by 2
 - written back into memory
- ▷ the operations are
 - sequential
 - optimized
 - * 64 bits (4 numbers) at a time are read or written
 - pipelined
 - * one number is multiplied while another one is read or written

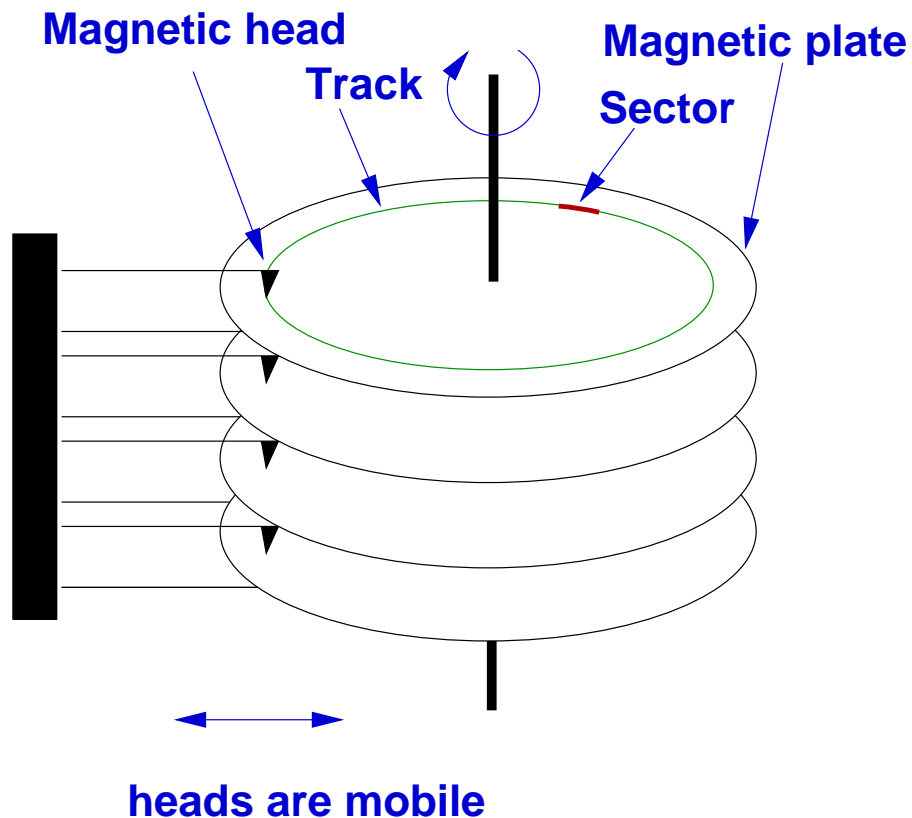
ADDRESSING IN A PC

We always address bytes (8-bit words)

Notation: we write in base 16 (hexadecimal)

- ▷ 4 binary digits: 16 possible values, from 0 to 15
 - ⇒ one digit in base 16
- ▷ The digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f
- ▷ Example: address 0x12345678
 - binary 00010010001101000101011001111000
 - decimal 305419896

MAGNETIC DISKS



Organization

- ▷ sectors of 512 bytes (4096 bits)
- ▷ each sector is read or written at once

SPEED OF MAGNETIC DISKS

Latency (example of a fast disk)

- ▷ 10000 rounds per minute (RPM)
- ▷ 166.7 rounds per second
- ▷ 6ms per round, up to 6 ms latency
(plus the time to move heads)

Comparison with fast RAM (PC3200)

- ▷ To access a long word in RAM
 - up to 6 cycles at 400 MHz
 - up to $6 \cdot 2.5\text{ns} = 15\text{ns}$
- ▷ the disk is up to 400000 times slower

SIZES OF MAGNETIC DISKS

- ▷ Typical disks:
 - 18 gigabytes (fast, 15000 RPM)
 - 120 gigabytes (ordinary, 7200 RPM)
- ▷ Typical memory modules:
 - 512 to 1024 megabytes (0.5 to 1 gigabyte)

HOW DISKS ARE USED

- ▷ virtual memory
- ▷ filesystems

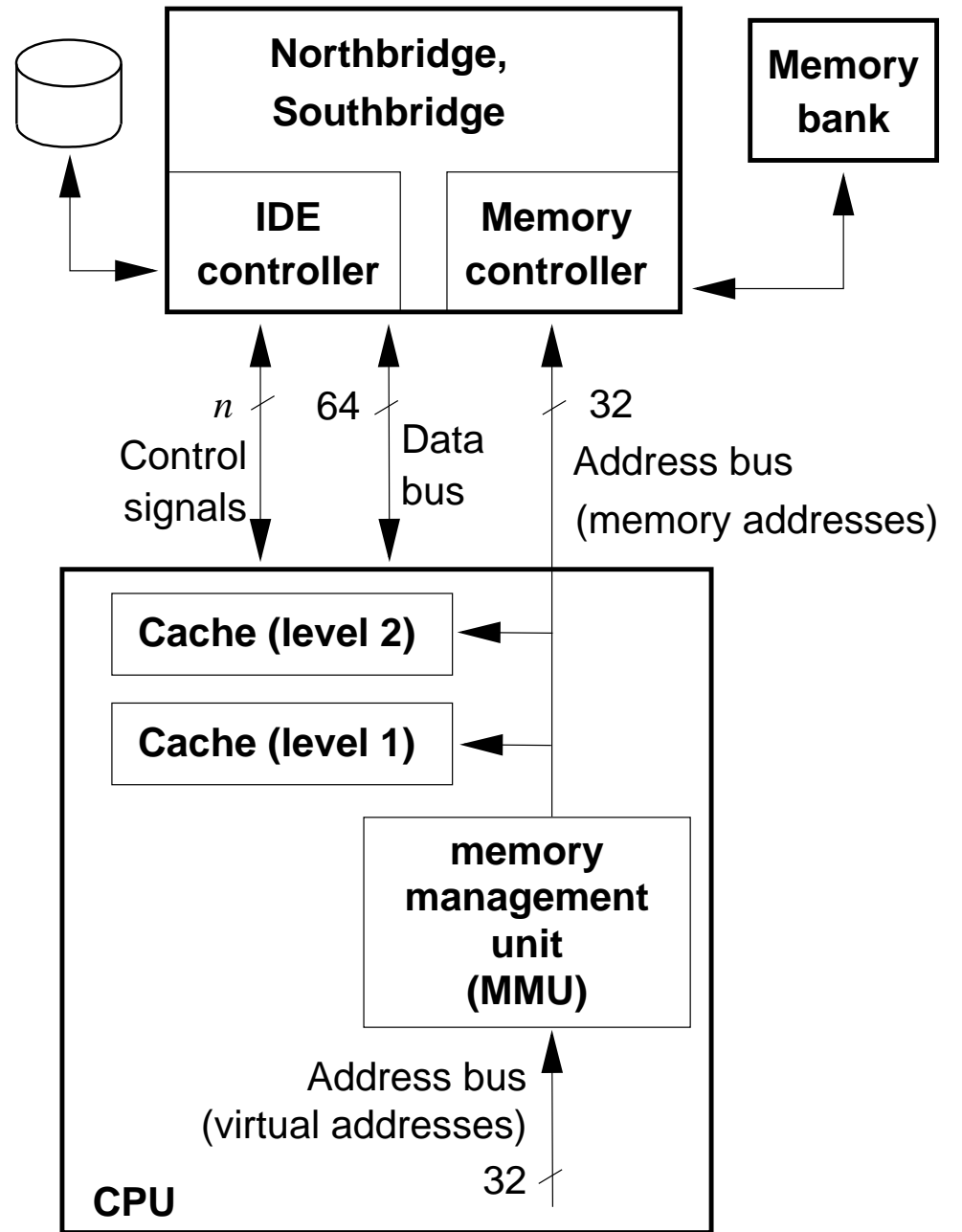
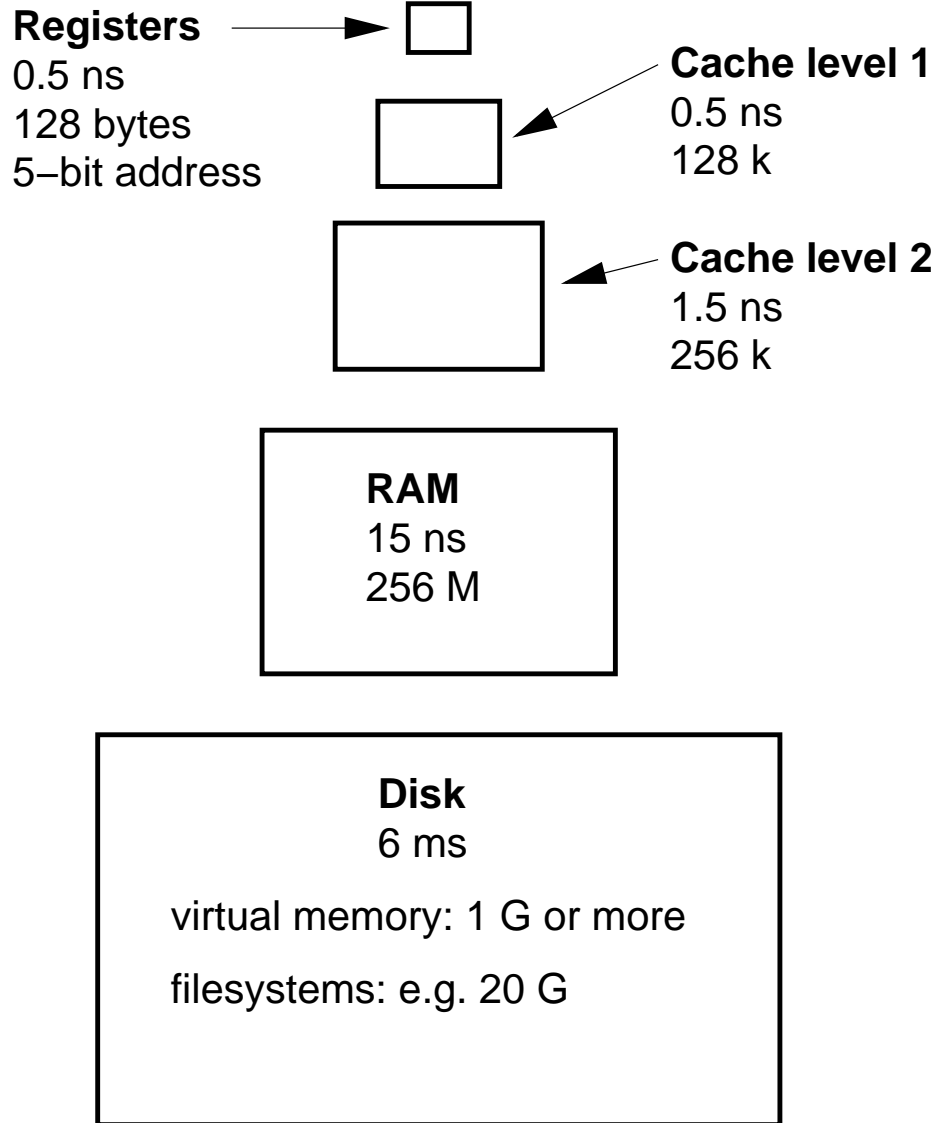
VIRTUAL MEMORY

Memory pages move between RAM and disk

- ▷ A page is 4 kilobytes
- ▷ Enough RAM \Rightarrow all pages stay in RAM
- ▷ Not enough RAM \Rightarrow pages move between RAM and disk
- ▷ Pages go from disk to RAM when needed for reading or writing
- ▷ Least recently used pages go from RAM to disk
- ▷ Virtual memory is invisible to the programmer

Common advice: “to accelerate your computer, add memory”

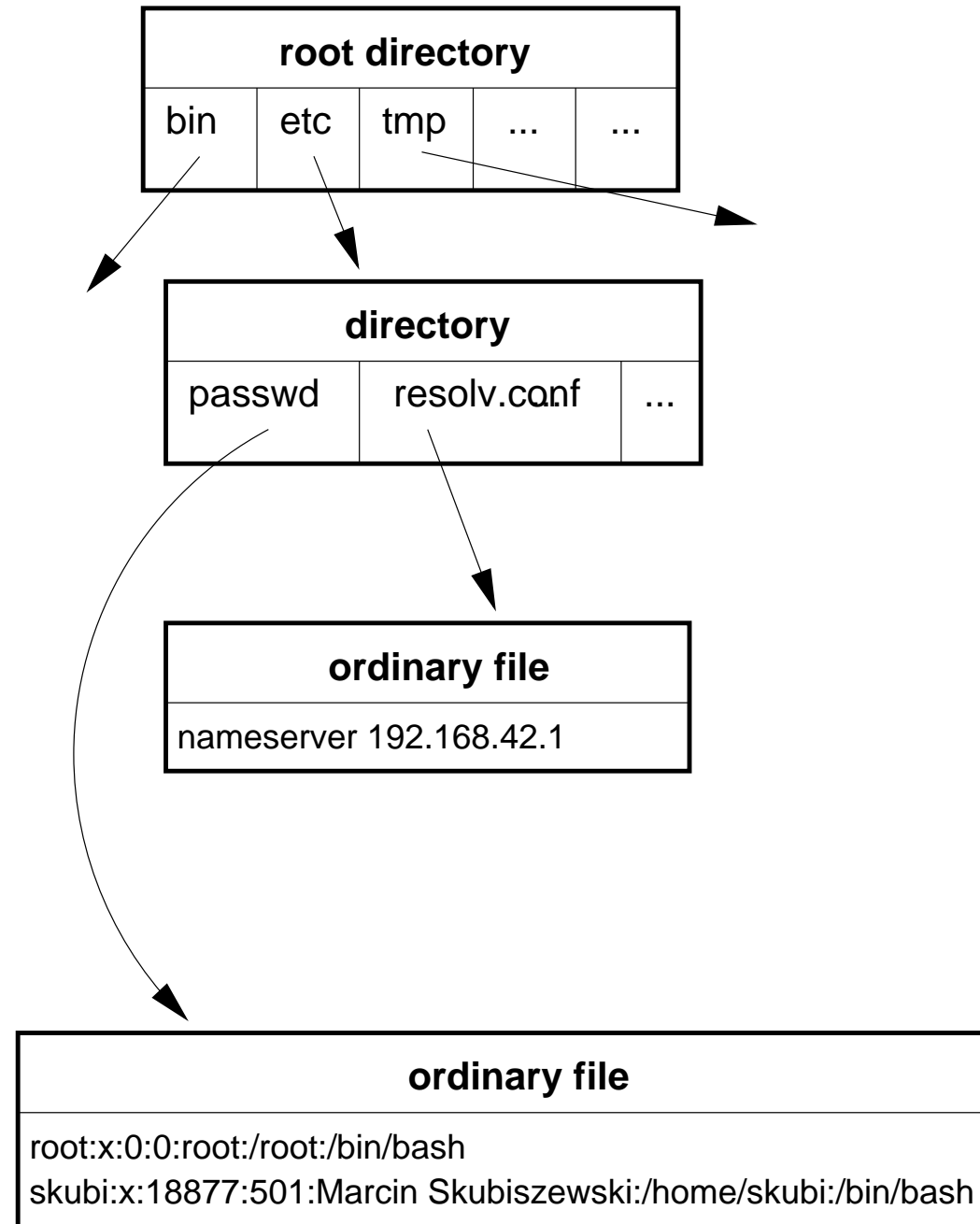
THE MEMORY HIERARCHY (simplified)



FILESYSTEMS

File

- ▷ an ordered sequence of bytes
 - Example: text
 - * bytes encode characters
 - Important example: a directory
 - * a file whose content consists of pairs <name, reference of file>
- ▷ has a name
 - Windows example: `C:\sys\autoexec.bat`
 - Unix/Linux example: `/etc/passwd`



HOW PROGRAMS ARE EXECUTED

WHAT A CPU DOES

Registers

- ▷ Very small, very fast storage inside the CPU
- ▷ Operations between memory and registers
- ▷ examples:
 - add to register EAX the number 5000
 - add to register EAX the content of memory at address 0x12345678
 - add to register EAX the content of memory at address pointed-to by register EBP

Operations

- ▷ arithmetic
- ▷ logical
 - example: AND, for 32 pairs of bits in parallel
- ▷ copying

MACHINE INSTRUCTIONS

An instruction tells

- ▷ what to do (example: add)
- ▷ which register to use
- ▷ which addressing mode to use (immediate, direct or indirect)
- ▷ which memory address to use

An instruction does not mention

- ▷ variables
- ▷ procedures

SOURCE CODE

Source code is for humans

- ▷ procedures
- ▷ variables

Example in C:

```
int doubleValue (int v) {  
    return 2*v;  
}  
main() {  
    volatile int a, b, c;  
    b = a + 5;  
    c = doubleValue(b);  
}
```

Compilation

Transforms source code (human-readable) into compiled code (computer-executable instructions)

EXAMPLE, COMPILED

```
.file "example.c"
.version "01.01"
gcc2_compiled.:
.text
    .align 4
    .globl doubleValue
    .type doubleValue,@function
doubleValue:
0000 55      pushl %ebp
0001 89E5     movl %esp,%ebp
0003 8B4508   movl 8(%ebp),%eax
0006 01C0     addl %eax,%eax
0008 89EC     movl %ebp,%esp
000a 5D      popl %ebp
000b C3      ret
    .Lfe1:
    .size doubleValue,.Lfe1-doubleValue
    .align 4
    .globl main
    .type main,@function
main:
000c 55      pushl %ebp
000d 89E5     movl %esp,%ebp
000f 83EC18   subl $24,%esp
0012 8B45FC   movl -4(%ebp),%eax
0015 83C005   addl $5,%eax
0018 8945F8   movl %eax,-8(%ebp)
```

```
001b 8B45F8   movl -8(%ebp),%eax
001e 01C0     addl %eax,%eax
0020 8945F4   movl %eax,-12(%ebp)
0023 89EC     movl %ebp,%esp
0025 5D      popl %ebp
0026 C3      ret
    .Lfe2:
    .size main,.Lfe2-main
0027 90      .ident "GCC: (GNU) 2.95.3 20010315 (rele
```

THE THREE WAYS OF EXECUTING PROGRAMS

Executing compiled code

- ▷ the fastest method

Executing source code

- ▷ parsing of source code too slow
- ▷ seldom used

Executing pseudocode in a virtual machine

- ▷ used for Java code
- ▷ universal, abstract machine
- ▷ indirect method: a program running on the PC reads pseudocode, understands it, and executes it
- ▷ not as fast as compiled code
 - the PC executes instructions that read and execute pseudocode instructions
- ▷ not as slow as source code
 - avoids complicated parsing

WHY PSEUDOCODE

Portability

- ▷ the same pseudocode runs
 - on Windows PC
 - on Linux PC
 - on Sun
 - on Macintosh

Security

- ▷ the virtual machine can have an arbitrarily sophisticated security model
(permit certain things, prohibit other things)

Security example: applets

- ▷ an applet is received from a server, runs on a client's computer
- ▷ problem: an applet may be a trojan horse
(harm the client on purpose)

- ▷ example: the applet could establish network connections using the client's identity
- ▷ solution one: the applet cannot communicate through network
- ▷ ... but often applets need to communicate
- ▷ solution two: the applet can only connect to the computer from which it comes
- ▷ this is only possible thanks to a virtual machine

MULTIPROCESSING

Example

- ▷ We want to do three things:
 - Prepare a document
 - * performance limited by human brain
 - Print a document
 - * performance limited by printer
 - Download a file
 - * performance limited by network connection

These three tasks can be done in parallel

- ▷ CPU and RAM time-shared between the three tasks
- ▷ Each task performs as well as if it were the only one

How this is done: multiprocessing

- ▷ process: a program being executed
 - several distinct executions of the same program are distinct processes
- ▷ a process is attached to every task
- ▷ each process has its own context
 - registers: what the CPU is doing
example: the program counter (PC) tells which instruction we execute
 - MMU content: which parts of RAM and virtual memory are used
- ▷ the operating system switches between processes
 - a process switch is a context switch

INTRODUCTION TO NETWORKING

A typical application

- ▷ the transmission of a web page

What is needed

- ▷ a page description language
example: `bold text`
- ▷ routing (deciding where to send data)
- ▷ a method to correct transmission errors
- ▷ a method to adjust transmission speed
- ▷ ... and more

RELIABLE STREAMS OF BYTES THE COMMON LAYER STRUCTURE

Upper layer (application-related)

Webpage description language (HTML)
Image compression method (JPEG)
Protocol for requesting pages
and sending information about pages (HTTP)

Lower layer (reliable streams of bytes)

Error correction, flow control
Addressing, routing

Hardware

Cables
Network interfaces

THE INTERNET-CENTRIC LAYER STRUCTURE

Upper layer (application-related)

Webpage description language (HTML)
Image compression method (JPEG)
Protocol for requesting pages
and sending information about pages (HTTP)

Lower layer (reliable streams of bytes)

Error correction, flow control (TCP)
Routing (BGP-4)
Domain Name Service (DNS)
Internet Protocol (IP)

Component networks

Cables + network interfaces
Ethernet
ATM

SIDE NOTE: THE OPEN SYSTEMS INTERCONNECTION MODEL

- ▷ 7. application
- ▷ 6. presentation
- ▷ 5. session
- ▷ 4. transport
- ▷ 3. network
- ▷ 2. data link
- ▷ 1. physical

This model is widely used

Why we do not use it:

- ▷ too complicated
- ▷ the Internet does not follow it

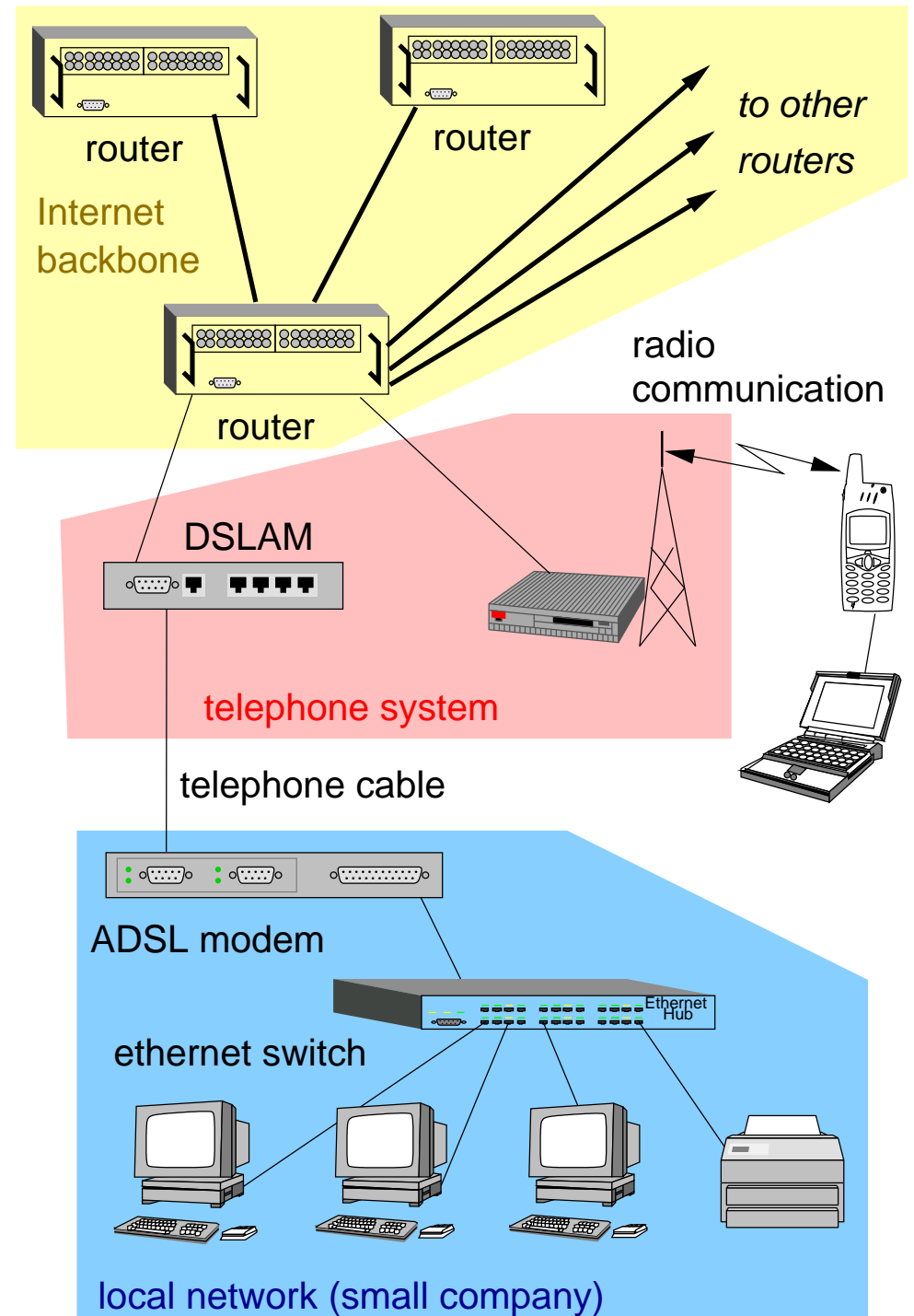
RELIABLE STREAMS OF BYTES

The fundamental principle

- ▷ packets

What must be accomplished

- ▷ addressing
- ▷ routing
- ▷ reliability
 - error detection
 - retransmission on error
- ▷ flow control



PACKETS

Definition

- ▷ a group of bytes that travel together in a network

Primary purpose

- ▷ to convey users' traffic
(convey a number of bytes belonging to a reliable stream)

Extra purpose

- ▷ to convey information/requests about users' traffic
 - example: to request the establishment of a connection
 - example: to acknowledge received bytes

An Internet packet contains

- ▷ a header
 - destination address (machine, port)
 - source address (machine, port)
 - checksum (for error detection)
 - sequence number (identification of the bytes contained)
 - length
 - acknowledgement of other packets
 - window (how many bytes we are able to receive)
 - flags (example: **FIN**—end of connection)
- ▷ bytes to convey

Length

- ▷ variable
 - minimum: header only, approx. 20 bytes
 - maximum: approx. 1500 bytes (depends on network)

CONNECTIONS

Definition

A connection is a pair of reliable streams of bytes between two processes.

- ▷ one stream in each direction
- ▷ streams always come in pairs

How a connection gets established

- ▷ A process running on computer *A* declares itself ready to receive connection requests
 - such a process is called a *server*
 - example: a web server
- ▷ A second process, running on another computer *B*, requests a connection to the server process
 - the second process is called a *client*
 - example: a web browser

- ▷ *B* sends to *A* a packet notifying the connection request
 - packets are built and parsed by the *kernel* of each computer
- ▷ *A* sends to *B* a packet notifying connection acceptance
- ▷ both processes are informed that the connection is established
- ▷ the two processes can send bytes to each other; the bytes are grouped into packets and sent over the network between *A* and *B*.

INTERNET ADDRESSING

Computer addresses

- ▷ Each computer connected to the Internet has a 32-bit address
- ▷ 4 billion addresses available
- ▷ Dial-up connections use temporary addresses

TCP ports

- ▷ The computer address is completed by a 16-bit *port number*
 - each connection is between two pairs (computer address, port number)
 - a computer can have many simultaneous connections
- ▷ 65000 port numbers

- ▷ Well-known port numbers used for servers
 - Webserver: port 80
 - SMTP server (mail): port 25
 - SSH server (remote command execution): port 22
- ▷ Arbitrary port numbers used by client programs
 - Example: two processes on my computer retrieve two photographs from a webserver
 - * Two distinct connections
 - * Webserver: port 80
 - * First client process: port 10550
 - * Second client process: port 10551
 - * Client port is used to distinguish between the two connections

PORT NUMBERS: PRACTICAL ORGANIZATION

- ▷ On every computer, a file contains well-known server port numbers: `/etc/services`
- ▷ An international agency attaches port numbers to services:

IANA: *Internet Assigned Numbers Authority*

`www.iana.org`

IANA works by consensus (has no official powers)

DOMAIN NAMES AND NAME SERVERS

How we name computers

- ▷ `www.wspiz.edu.pl`
 - top level domain: `pl`
 - inside `pl`, subdomain `edu`
 - inside `edu.pl`, subdomain `wspiz`
 - inside `wspiz.edu.pl`, subdomain `www`

How we convert a domain name into the corresponding 32-bit address

We ask *domain name servers*

- ▷ There are 11 name servers worldwide that know all top-level domains (*root name servers*)
 - The root servers are all identical (you can query any one of them)
 - Addresses of root servers are widely known

- ▷ Root servers know the addresses of the 11 `pl` name servers

(NS response)

- ▷ `pl` name servers know the addresses of the 2 `wspiz.edu.pl` name servers

(NS response)

- ▷ `wspiz.edu.pl` name servers know the address of `www.wspiz.edu.pl`

(A response)

When the conversion takes place

- ▷ Before every connection
- ▷ Conversion results are stored
 - locally
 - in caching name servers located at your ISP

NOTATION

32-bit computer addresses are noted with 4 dot-separated decimal numbers.

- ▷ Each number represents 8 bits
- ▷ Each number is between 0 and 255
- ▷ Example: 204.50.6.5
- ▷ This notation is only used for addresses

INTERNET ROUTING

Colocated computers have addresses sharing a common prefix

- ▷ Addresses sharing a common prefix form a **block**
 - example: the 202.55.0.0/16 block
all addresses whose first 16 bits are the same as the first bits of 202.55.0.0

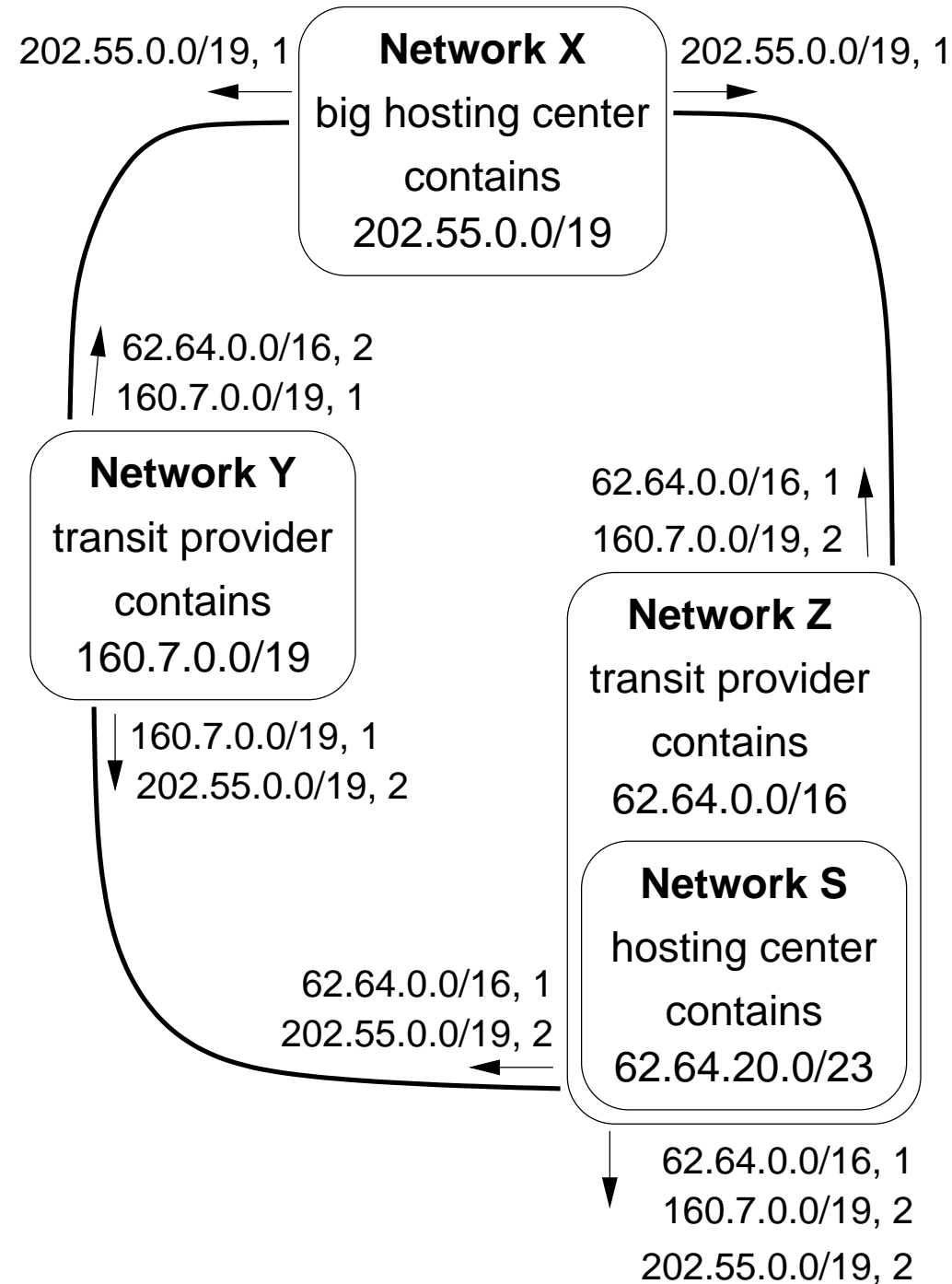
Every network in the Internet *announces* blocks that can be reached through it

- ▷ A distance (number of blocks) is provided with each block

For every address, we announce and use the shortest path

Name of this protocol:

BGP-4 (border gateway protocol, version 4)



INTERNET ROUTING, EXAMPLES

Major router in backbone

- ▷ routes to approx. 120 000 blocks are known and further announced

Router in a small company

- ▷ 2 blocks used for routing:
 - the local block 202.55.5.17/28, distance 1
 - rest of the world, block 0.0.0.0/0, distance 2

Routes are not always symmetric

Routes can change anytime

- ▷ routes change when a connection is established or cut

How addressing resources are allocated

- ▷ Four regional organizations allocate huge blocks to major networks (*autonomous systems*)
 - In Europe: RIPE (www.ripe.net)

Autonomous system—definition

A part of the Internet that participates in BGP-4 exchanges

Route announcements are policy-driven

- ▷ a hosting center only announces routes to itself
- ▷ a transit network will
 - announce to its customers all routes
 - announce to everyone routes to its customers

ERROR DETECTION

What we have

- ▷ transmission media that make errors

What we want

- ▷ no errors

What we do while sending a packet

- ▷ Compute a checksum of each packet
 - approximately as follows
 - * cut the packets into 16-bit chunks
 - * consider each chunk as a number
 - * add all these numbers together
- ▷ Send the checksum in the packet
 - The checksum depends on header and data
 - The checksum consists of 16 bits

What we do while receiving a packet

- ▷ Compute the checksum again
- ▷ Compare the checksum with the one received
 - Correct transmission
 - * both checksums identical
 - * the packet is taken into account
 - Some bits received incorrectly
 - * the checksums are different
 - * the packet gets thrown away

ERROR CORRECTION

Principle

- ▷ Retransmit bytes that did not arrive at destination

Details

- ▷ Inside a reliable stream, bytes are numbered
 - each packets bears the *sequence number* of its first data byte
- ▷ Receiver *acknowledges bytes*:
 - he sends the sequence number up to which everything has been received
- ▷ Sender *retransmits*:
 - if, after some specified time, certain bytes have not been acknowledged, they are transmitted again

- if, after an even longer time, certain bytes have still not been acknowledged, they are transmitted once more
- etc.
- after several minutes, the connection is cut

VOCABULARY

Communications protocol

Set of rules according to which information is exchanged

IP (Internet Protocol)

IP specifies rules that are common to all information transmissions in the Internet

TCP (Transmission Control Protocol)

TCP specifies rules specific to reliable streams of bytes

- ▷ Internet reliable streams of bytes follow the IP and TCP protocol

– thus the commonly-used name **TCP/IP**

UDP (User Datagram Protocol)

UDP specifies rules specific to transmitting unreliable packets

- ▷ not covered here

DNS (Domain Name Service)

DNS describes information exchanged between name servers and clients

BGP-4 (Border Gateway Protocol, version 4)

BGP-4 describes how networks exchange routing information

- ▷ the networks are called *autonomous systems*

ICMP (Internet Control Message Protocol)

ICMP certain kinds of control messages

- ▷ ICMP messages are not directly seen by the application layer

NETWORKING—SUMMARY

- ▷ Networking software and hardware is classified into *layers*
- ▷ Our description is based on three layers:
 - component networks
 - reliable streams of bytes
 - application
- ▷ Other systems of layers exist
- ▷ Bytes in the Internet are grouped into packets
- ▷ Each computer has a 32-bit address; inside a computer, each process can use one or more 16-bit port numbers

- ▷ Domain names (like `www.wspiz.edu.pl`) are converted into 32-bit addresses by domain name servers
 - domain name servers are hierarchical
- ▷ Routing is determined by exchange of routing information between neighboring autonomous systems
- ▷ For every packet, a checksum is computed, to check that the every bit in the packet is received correctly
- ▷ Bytes that were not received correctly are retransmitted, until correct reception

Everything we say here about networks is much simplified

OVERVIEW OF THE WORLD-WIDE WEB

Part 1: web pages considered as static entities

HTML (the hypertext markup language)

- ▷ Content and structure of a web page
 - *Markup* or *tags*: everything except the text to display

CSS (cascading stylesheets)

- ▷ Appearance directives, attached to various elements of web pages
- ▷ Example: titles of sections are in bold 18-point typeface

HTTP (the hypertext transmission protocol)

- ▷ Requests for pages
- ▷ Meta-information (information about pages)
 - language, character encoding
 - caching policy

HTML TAGS

What tags do

- ▷ Describe the structure of the document
 - sections, tables etc.
- ▷ Determine how to display text (boldface, italics etc.)
 - **this is deprecated**
- ▷ Describe hyperlinks
- ▷ Describe included documents
 - graphics, frames
- ▷ Declare keywords related to the web page

Note: the example that follows is

`../html/example.html`

```
<!DOCTYPE html PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <title>Example HTML document</title>
  <meta http-equiv="content-type"
    content="text/html; charset=ISO-8859-1">
</head>
<body>
<h1>Header</h1>
<p>Ordinary text
<a href="http://slashdot.org/"><br>
Hyperlink to a technology-oriented
news site</a></p>
<ul>
  <li>bulleted</li>
  <li>lines</li>
</ul>
<ol>
  <li>numbered</li>
  <li>lines</li>
</ol>
<table cellpadding="2" cellspacing="2"
border="1" width="100%"
```

```

style="text-align: left;">
  <tbody>
    <tr>
      <td valign="top">cell 1<br>
      </td>
      <td valign="top">cell 2<br>
      </td>
    </tr>
    <tr>
      <td valign="top">cell 3<br>
      </td>
      <td valign="top">cell 4<br>
      </td>
    </tr>
  </tbody>
</table>
<br>
<br>
<hr width="100%" size="2"><br>
<br>
</body>
</html>

```

RULES GOVERNING TAGS

General syntax

<tagname attribute1="value1" attribute2="value2" attribute3="value3">

Content:

text
nested tags

</tagname>

Example (note indentation)

```

<font size='+1'>
  <ul>
    <li>
      bulleted
    </li>
    <li>
      lines
    </li>
  </ul>
</font>

```

Syntax of tags without content

- ▷ XML (modern syntax)

`<tagname attribute1="value1" />`

- ▷ syntax that works everywhere (note the extra whitespace)

`<tagname attribute1="value1" />`

Lax (old-style) syntax

- ▷ it is often OK to
 - forget to close a tag
 - forget to mark a content-less tag as such
- ▷ no quotes when attribute is alphanumeric

Some tools accept the old syntax, others do not

The use of old syntax is prohibited in our project

Special characters

`&character-name;`

- ▷ Examples: `>`; `&`;

`&#hexadecimal-character-number;`

CASCADING STYLESHEETS (CSS)

Principle: separate presentation from structure

- ▷ In HTML, we describe content and structure
 - <h1> to <h3>: headers
 - <p>: paragraphs (ordinary text)
 - tables
 - hyperlinks
 - included graphics

- ▷ In CSS, we describe presentation
 - choice of fonts
 - size of characters
 - colors
 - indentation
 - placement of included graphics
 - ... and much more

ABOUT SEPARATION IN GENERAL

Why separate

- ▷ A computing project contains from 10 000 to 1000 000 lines of code
- ▷ Often, no one can understand a project in its entirety
- ▷ A project must be organized so that
 - you can work on a component, without understanding the rest of the project
 - you can modify a component, and the modification will have no impact on the rest of the project

Programming example

- ▷ The Netscape web browser runs on Microsoft Windows and on Linux/XWindows
 - need clear distinction between graphics-related components and the rest

- graphics-unrelated components
 - * just say “display this element”
 - * have no idea of *how* the display operation is done

Web-related separation

- ▷ Change of presentation should imply as little change as possible to a web project
- ▷ Example of reasons for changes of presentation:
 - new design for aesthetic reasons
 - addition of a WAP service
 - addition of printed documents
 - accessibility
 - * blind users
 - * slow connections

CSS SYNTAX

The best way of using CSS

- ▷ The stylesheet is a separate document (distinct from your HTML text)
- ▷ The same stylesheet is used for a large number of web pages
 - typically, just one stylesheet for your whole website

How to declare a stylesheet

In the content of the <head> tag of your web page, say

```
<head>
...
<link rel='stylesheet' type='text/css'
href='/mystylesheet.css' />
...
</head>
```

*Note: the example that follows is in
../html/style-example.html*

CSS SYNTAX

Example (see `mystylesheet.css`):

```
body {font-family: sans-serif;}
h1 {text-align:center;font-weight: bold;}
.example {background: #ffa;
          font-style: italic}
.important {background: #f88;
            font-weight: bold}
.logo {color: #00f}
```

How it works

Before each opening brace, you specify the tags to which the style applies

- ▷ no dot: all the tags bearing a given tag name
 - body for the whole document
- ▷ preceded with a dot: tags bearing a given class
 - .example for <div class='example'> ...
</div>

USING CSS RULES

The simple method for very simple documents

- ▷ The structure proposed by HTML suits your needs
 - sections `<h1>`, `<h2>`, `<h3>`
 - no special text (all text is ordinary and is contained in `<p>` tags)

In this case, styles (CSS rules) attached to tag names suffice

The flexible method for complicated documents: the `div` tag

- ▷ Certain parts of the document are special
 - Examples are in italics
 - Important paragraphs use bigger characters
 - etc.

- ▷ Each special part of the document is represented by a `div` tag

```
<div class='recipe'>
  <h1> My example </h1>
  <p> Take 3 eggs and 5 kg of sugar </p>
</div>
```

- ▷ The corresponding CSS rules are written with a dot

```
.recipe {background: #f88}
```

- ▷ You can totally abandon the usual structure-related tags (`<h1>`, `<p>` etc)

- ▷ Example:

```
<div class='s1'>Part One: CSS</div>
<div class='s2'>Chapter 1: syntax</div>
<div class='s3'>Section 1.1: braces</div>
<div class='p'>
  Braces are used as follows...
</div>
```

The effects of `div`

- ▷ To group together a number of paragraphs
 - titles count as paragraphs(a `div` may contain only one or several paragraphs)
- ▷ To allow the application of style rules for such a group
- ▷ No other effects
 - without a styleshee, `div` has no effect

The `span` tag

- ▷ `span` is included in a paragraph
- ▷ `span` is for formatting small elements

▷ example:

```
<p>  
  Our company,  
  <span class='logo'>  
    ACME Web design  
  </span>,  
  would be happy to design your web site  
</p>
```

Attaching a style to an individual tag

```
<p>  
  Our company,  
  <span style='color: #f00'>  
    ACME Web design  
  </span>  
  would be happy to design your web site  
</p>
```

- ▷ Good for testing
- ▷ Good when you need a different a style for each paragraph
 - see example below
- ▷ Do not do this in common situations

What about other possibilities

- ▷ There are many ways to use CSS
- ▷ The problem: you must follow a clear method, or else you will get lost
- ▷ I recommend the methods described here

WEB GRAPHICS

Including graphics in a web page

```

```

The two formats for photographs

- ▷ Principle: wavelet-based lossy compression
 - Instead of transmitting each pixel, we transmit,
 - * the average shade of an area
 - * how the shade changes in the area
 - ▷ example: darker on the left, lighter on the right
 - * small numbers are suppressed, replaced with 0

- this is hierarchical:
 - * for area = whole image
 - * for area = lower left quarter
 - * etc.

▷ Today's format: JPEG

- not really wavelets

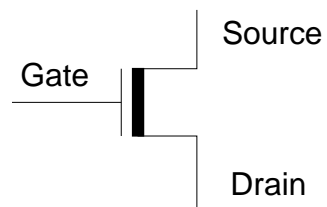
▷ In near future: JPEG 2000

- true wavelets
- vastly superior to JPEG
 - * better quality for equal file size
- not implemented in today's web browsers

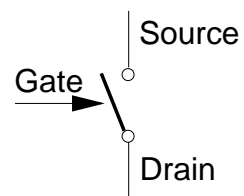


The two formats for drawings

- ▷ Principe:
 - almost the whole surface in a drawing is uniform
 - this leads to lossless compression
- ▷ The old format: GIF
 - limited to 256 colors in a figure
- ▷ Today's format: PNG



The symbol



What it does

EASE OF ACCESS TO WEB SITES

The case of Chirac and Jospin (French presidential election 2002)

- ▷ Many candidates, many websites
- ▷ Two candidates had huge budgets: Chirac and Jospin
- ▷ Two candidates had really slow websites: Chirac and Jospin
- ▷ Internet users do not want slow websites
 - Chirac and Jospin used their money to shoot themselves in the foot

Analogy

- ▷ You start a new daily newspaper
- ▷ You use many photographs and glossy paper
- ▷ Your newspaper is expensive
- ▷ Nobody buys your newspaper
- ▷ You fail

The problem

- ▷ Newspaper editors understand basic economics
- ▷ Many web designers do not understand "web economics" (making web sites accessible to many people)

KEY ISSUES CONCERNING EASE OF ACCESS

- ▷ Download time
- ▷ Accessibility
 - handicapped users, PDAs, slow connections
- ▷ Standard conformance
 - web browsers other than Internet Explorer

ACCESS TIMES

A modern telephone modem:

- ▷ 56 kbits/s

An older telephone modem:

- ▷ 33.6 kbits/s

An ADSL connection:

- ▷ 512 kbits/s

Photograph sizes: see example, file `sizes.html`

Where web designers waste download time:

- ▷ huge client-side scripts
- ▷ very complex web pages

ACCESSIBILITY

Example situations

- ▷ Blind users
 - the browser reads web pages aloud
- ▷ Color-blind users
- ▷ Other disabled users
 - no mouse
- ▷ Small screens
 - PDAs
- ▷ Very slow connections
 - ordinary GSM phone: 9600 bits/s
 - * 6 times slower than modern landline modem

Consequences

- ▷ Cannot see images

- ▷ Cannot see colors
- ▷ If the document has a complicated layout, cannot see it

Simple solutions

- ▷ Describe images textually
- ▷ Keep the graphical structure simple
 - the graphical structure of the document should follow its logical structure

Example (my eBay)

What we have:

```
<a href=" http://cgi.ebay.com/ws/
ebayISAPI.dll?ViewItem&category=
1244&item=2076358329#BID">
  
</a>
```



What we should have:

```
<a href=" http://cgi.ebay.com/ws/
ebayISAPI.dll?ViewItem&category=
1244&item=2076358329#BID">
  
</a>
```

Note

- ▷ The example was easy to find

WEB-RELATED STANDARDS

The problem

- ▷ 95 % of computers in usual situations run Internet Explorer on Microsoft Windows
 - **usual situation:** desktop or laptop computer, user has no disabilities
- ▷ Many web designers only consider Internet Explorer users

The low/high quality logos

This site is optimized for Internet Explorer 5.0



Who writes standards

The *World-Wide Web Consortium*

www.w3c.com

Recommended reading for everyone

Elizabeth Castro, *HTML for the World Wide Web with XHTML and CSS: Visual QuickStart Guide*

In Polish: *Po prostu HTML 4.0*

Recommended reading (advanced)

The standards

EASE OF ACCESS: SUMMARY AND ADVICE

Elementary level

- ▷ Standard conformance
 - Spend time reading documentation
 - Avoid Microsoft teaching material
 - Test your web pages using several web browsers

- ▷ Download times
 - Keep the stylesheet as a separate document
 - Keep extraneous content out of your web pages
 - * example: useless client-side scripts
 - Test your web pages through an ordinary modem (56 kbits/s)
 - Test your web pages through a slow modem (9600 bits/s)

▷ Accessibility

- Provide an ALT tag for every image
- Have HTML tags follow the logical structure of your document
- Test your web pages using a text-only browser (`links` or `lynx`)
- Test your web pages in a small window (VGA is 640×480)

Advanced level

▷ Standard conformance

- Use the W3C validator at
 - * `validator.w3c.org`
 - * `jigsaw.w3.org/css-validator/very strict`
only useful if you want to strictly follow standards

▷ Accessibility

- Read the W3C accessibility documents at `www.w3.org/WAI/`

THE HYPERTEXT TRANSMISSION PROTOCOL (HTTP)

▷ **Usually:**

- client sends a request
 - * tells what she wants
 - ▷ URI: universal resource identifier
 - * tells her preferences
 - * identifies previously open sessions (sends cookies)
- server responds by sending a document
 - * describes the document
 - * sets up cookies

▷ **Sometimes:**

- client sends a document

▷ **Vocabulary:**

- client sends *requests*, server sends *responses*

UNIVERSAL RESOURCE IDENTIFIERS (URIs)

Example:

```
http://sudety.net:8080/viewItem.jsp?id=45630&x=3
```

How URIs are used

- ▷ in HTTP requests
- ▷ in HTML documents, as references to resources
 - references to images, frames
 - hyperlinks

The scheme

- ▷ The access method for a given resource

The scheme—examples

- ▷ `http:` document available through HTTP
- ▷ `https:` document available through HTTP over a cryptographically secure connection
- ▷ `ftp:` document available through FTP (file transfer protocol)
- ▷ `email:` email address

The domain name (`sudety.net`)

- ▷ determines the address of the server computer
- ▷ often, is used to choose one of many web sites served by a computer
 - many low-traffic domains may be hosted by the same computer

The TCP port (8080)

- ▷ normally not used
 - the file `/etc/services` determines the port number of the server
- ▷ in experimental/temporary setups, used to select one of several web servers running on the same computer

The filename (`/viewItem.jsp`)

- ▷ selects the file to serve or the program to run

The arguments (`id=45630&x=3`)

- ▷ a number of variable-value pairs
- ▷ the arguments are provided to the program that will be run to fulfill the request

ABSOLUTE AND RELATIVE URLs

Relative URLs

- ▷ no schema, no domain name, no TCP port
- ▷ examples:

```
frogcamera.jpg  
../images/frogcamera.jpg  
/aux/style.css
```

- ▷ Without slash: relative to the directory of the document mentioning the URL
- ▷ With slash: relative to the domain name of the document mentioning the URL

The use of relative URLs

- ▷ In HTML documents
- ▷ Not in HTTP requests
- ▷ the client converts relative URLs to absolute ones

```
<img src='frogcamera.jpg' />
```

in `http://www.sudety.net/webcams/frogs.html`

results in a request for

```
http://www.sudety.net/webcams/frogcamera.jpg
```

The advantage of relative URLs

- ▷ You can move a complete website to another location
 - ... and URLs internal to the site will still work, with no change

DOCUMENT PROPERTIES IN HTTP

See *http-example.txt*

User preferences in request

- ▷ Natural languages accepted by the client
 - seldom used, and therefore often unusable
- ▷ Character encodings accepted
- ▷ Content types accepted
 - example: does the web browser accept JPEG 2000 ?
- ▷ ... and more

Document properties in response

- ▷ Content type
 - examples: `text/xhtml`, `image/jpeg`, `text/plain`
- ▷ Character encoding
 - example: `utf-8`
- ▷ ... and more

DOCUMENT CACHING

A document can

- ▷ come from the webserver
- ▷ come from an HTTP proxy run by your ISP
- ▷ already be present in client computer
- ▷ already be present, but

- before displaying the document, the client computer checks the identity of the server's current version

Examples

- ▷ stylesheet (CSS)
 - should only be downloaded once (this only works if the stylesheet is a separate document)
- ▷ company logo
 - should only be downloaded once (very important if the logo is used many times in the website)
- ▷ newspaper frontpage
 - should always come from webserver, or at least the cached version should be checked against the one in the server

What HTTP does

- ▷ Makes it possible to specify
 - a per-document caching policy
 - a per-document expiration date
- ▷ Allows conditional requests
 - get document if newer than...

Default policy

- ▷ URLs without arguments correspond with cachable documents
- ▷ URLs containing a '?' are considered as having arguments, and correspond with non-cachable documents
 - **Advice:** use '?' in references to webcam images

```
<meta http-equiv="refresh" content="30;
URL=index.html">
```

[...]

```
<IMG align="center" ALT="frog camera"
SRC="frogs3.jpg?">
```

COOKIES AND SESSIONS

Definition

Several consecutive and related requests/responses from a given user form together a **session**

Example: shopping cart

- ▷ a user progressively builds her shopping cart
- ▷ the shopping cart is attached to a session
- ▷ the page at

```
http://localhost:8080/sudety/viewCart.do
```

displays the content of the cart, which depends on the session

Example: non-secure authentication

- ▷ at some point
 - the user provides her account name and password
 - the server attaches the identity of the user to the session
- ▷ then, the user remains authenticated all the time

How this is done: cookies

- ▷ the server *sets a cookie* in the web browser
- ▷ then, the browser sends the cookie back with every request
- ▷ a cookie is always restricted to a second-level domain
 - `amazon.com` can see that you are the one who visited `amazon.com` yesterday
 - `amazon.com` does not know whether you also visited `ebay.com`

- ▷ a cookie can be restricted further
 - in time
 - to certain subdomains
 - to certain directories

ELECTRONIC COMMERCE APPLICATIONS

Part 1: the web interaction

Forms

- ▷ how the user tells what she wants

Server-side includes

- ▷ adding program fragments to web pages

Note: the form is at

`http://localhost:8080/sudety/form.jsp`

```
<form action="simple.jsp" method="get">
  <p>
    Item name:
    <br />
    <input type="text" name="item-name"
      value="Porsche 911" size="50"
      maxlength="1024" />
  </p>
  <p>
    Delivery mode:
    <br />
    <input type="radio" name="delivery"
      value="mail" checked="checked" />
    by mail
    <input type="radio" name="delivery"
      value="fedex" />
    by FedEx
    <input type="radio" name="speed"
      value="fast" checked="checked" />
    fast delivery
    <input type="radio" name="speed"
      value="economic" />
    economic delivery
  </p>
```

```
<input type="submit" name="action"
  value="Purchase" />
<input type="submit" name="action"
  value="Check price" />
</form>
```

Note: the JSP that follows is at
<http://localhost:8080/sudety/simple.jsp>

```

<%@ page language="java" import="java.util.*" %>
<% response.setContentType("text/html");
Enumeration attrEnum
    = request.getParameterNames();
boolean isEmpty = !attrEnum.hasMoreElements();
if (isEmpty) {
    response.setStatus
        (HttpServletResponse.SC_BAD_REQUEST);
}
%><!DOCTYPE (omitted)>
<html (omitted)>
<head>
    <title>Example Java Servlet Page (JSP)</title>
    <meta http-equiv="content-type"
content="text/html; charset=ISO-8859-1">
    <link rel='stylesheet' type='text/css'
        href='/style.css'>
</head>
<body>
<%
    if (isEmpty) {
        %>
        <div class="error">
            Error: you provided no arguments.
        </div>
        <%
    } else {
        %>
        <h1>Parameters received:</h1>

```

```

<table class="paramList" border="2">
    <thead>
        <tr>
            <td>Parameter name</td>
            <td>Parameter value</td>
        </tr>
    </thead>
    <tbody>
        <%
            while (attrEnum.hasMoreElements()) {
                String s
                    = (String)attrEnum.nextElement();
                %>
                <tr>
                    <td><%= s %> </td>
                    <td><%= request
                        .getParameterValues(s)[0] %>
                    </td>
                </tr>
                <%
            }
        %>
    </tbody>
</table>
<%
}
%>
</body>
</html>

```

JAVA SERVLET PAGES (JSP)

Mode change operators

<%	Java mode
<%=	Java expression (value to be inserted into the web page)
<%@	Java declaration mode
%>	back to HTML mode

Example

```
<%@ page language="java"
import="java.util.*" %>
<%
response.setContentType("text/html");
%><!DOCTYPE (omitted)>
<html (omitted)>
<head>
<title>Example counter</title>
<meta http-equiv="content-type"
```

```
content="text/html;
charset=ISO-8859-1">
<link rel='stylesheet' type='text/css'
href='style.css'> </head>
<body>
<h1> Let's count from 0 to 9 </h1>
<%
for (int i=0; i < 10; i++) {
%>
<p> Counter value <%= i %> </p>
<%
}
%>
</body>
</html>
```

Example available at

<http://localhost:8080/sudety/count.jsp>

OBJECTS IN JAVA

Definitions

- ▷ **An object:** a data structure
 - comprising a number of *fields* (values),
 - accompanied with a number of *methods* (program fragments).
- ▷ **A class:** the description of the fields and methods of an object

Example class

```
public class Person {
    String firstName;
    String familyName;
    int birthYear;
    public String getFirstName() {
        return firstName;
    }
    public String getFamilyName() {
        return familyName;
    }
}
```

```
public String getCompleteName() {
    return firstName + " " + familyName;
}
public int getAge (int currentYear) {
    if (currentYear < birthYear) {
        return 0;
    } else {
        return currentYear - birthYear;
    }
}
public void setName (String a, String b) {
    firstName = a;
    familyName = b;
}
}
```

Example of use

```
Person p;    /* reference to an object */
/* ... */
System.out.println ("Dear "
                    + p.getCompleteName()
                    + ",");
```

PREDEFINED OBJECTS IN JSP:

request

Object declaration (implicit)

```
HttpServletRequest request;
```

Scope

- ▷ common to all program elements that service a given request

Content

- ▷ complete information about the HTTP request
 - parameters
 - locale
 - client's IP address
 - ... and much more
- ▷ information added by the Servlet container
 - session information

- ▷ information added by the electronic commerce application
 - communication between program elements

```
Enumeration getParameterNames();
```

- ▷ returns the names of all the parameters supplied by the client

What we do:

```
Enumeration attrEnum  
    = request.getParameterNames();
```

What can be done with an enumeration:

- ▷ is there any more content?

```
boolean hasMoreElements();
```

- ▷ retrieve next element

```
Object nextElement();
```

What we do (initially):

```
boolean isEmpty  
    = !attrEnum.hasMoreElements();  
if (isEmpty) {  
    response.setStatus  
        (HttpServletResponse.SC_BAD_REQUEST);  
}
```

request (cont'd)

```
String[] getParameterValues(String);
```

Examples of use

```
if (getParameterValues("ship").length
    != 1) {
    /* do something */
}
String shipmentMode
    = getParameterValues("ship")[0];
```

The cast (type conversion)

- ▷ (String) checks that attrEnum.nextElement() really returned a String
 - otherwise, an exception is raised (the execution is interrupted)
 - in the general case, attrEnum.nextElement() could return any object

What we do with our Enumeration (cont'd):

```
while (attrEnum.hasMoreElements()) {
    String s
        = (String)attrEnum.nextElement();
    %>
    <tr>
        <td><%= s %> </td>
        <td>
            <%=
                request.getParameterValues(s)[0]
            %>
        </td>
    </tr>
    <%
}
}
```

WHAT WE HAVE AND WHAT WE NEED

We can

- ▷ generate web pages
- ▷ generate web-based forms
- ▷ retrieve values filled in forms
- ▷ program in Java

We still need

1. **program organization** (Jakarta Struts)

- ▷ in theory, you can write an application in JSPs
- ▷ in practice, confusion will kill you if you try

2. **data organization** (a database management system)

3. **convenience software** (Jakarta Struts)

example:

- ▷ all web-based application need to
 - prepopulate forms
 - retrieve information from forms
- ▷ program fragments that do this should be provided to us

PROGRAM ORGANIZATION (MODULARITY)

The objective

Each fragment of the project serves one well-defined purpose

- ▷ to write a fragment (a module) of the project you only need to know the corresponding fragment of the specifications
- ▷ a localized change to specifications will result in a localised change to the project
 - example: changing background color
 - example: changing criteria for article search
 - example: move
 - * from the notion of *manager* (can update product descriptions and customer accounts)
 - * to *assortment manager* (can update product descriptions) and *customer*

manager (can update customer accounts).

How we achieve modularity

- ▷ cascading stylesheets (CSS)
- ▷ model-view-controller separation (MVC2)
 - separation between
 - * business logic (model)
 - * displaying web pages (view)
 - * programs that organize everything (controller)
- ▷ object-oriented programming
 - build a class for every kind of real-world object, fact, or relationship
 - * examples: Article, ArticleList, Customer

THE MODEL-VIEW-CONTROLLER SEPARATION

The model (business logic)

- ▷ Business-related operations
 - search for articles, according to given criteria
 - purchase articles
 - update an article description
 - create a user account

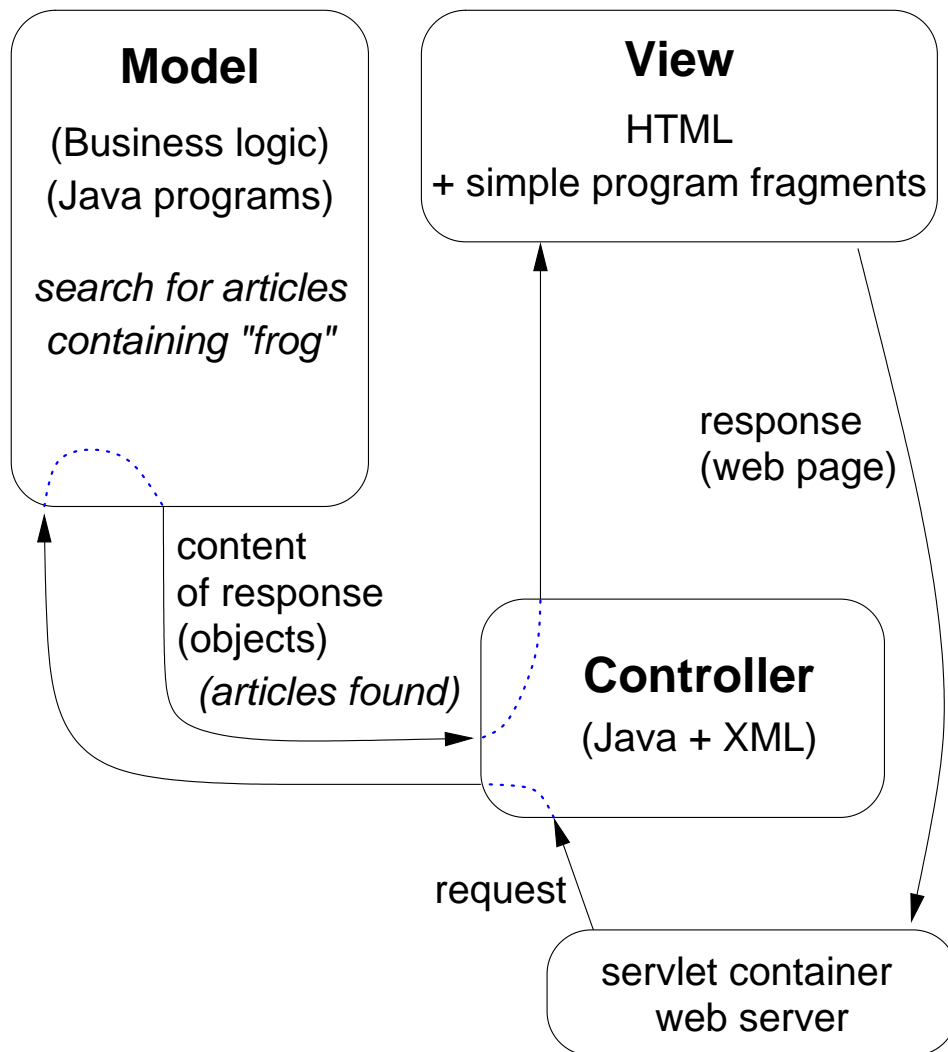
▷ Has nothing to do with

- displaying web pages
- debriefing forms

(ignores the very existence of the World Wide Web)

Example:

- generates a list of articles found, ignores how (and whether) the list is going to be displayed



- ▷ Written in Java (no need for JSP)

The view

- ▷ Displays web pages
- ▷ Has nothing to do with business logic

Example:

- displays a form to collect search criteria from the user
 - displays a list of articles representing the search results
 - ignores *how* the search is done
- ▷ Written in JSP (Java only used to retrieve results from the model)

The controller

- ▷ The glue that connects things together
 - tells which model program will process a given form
 - tells which JSP will display results from a given model operation

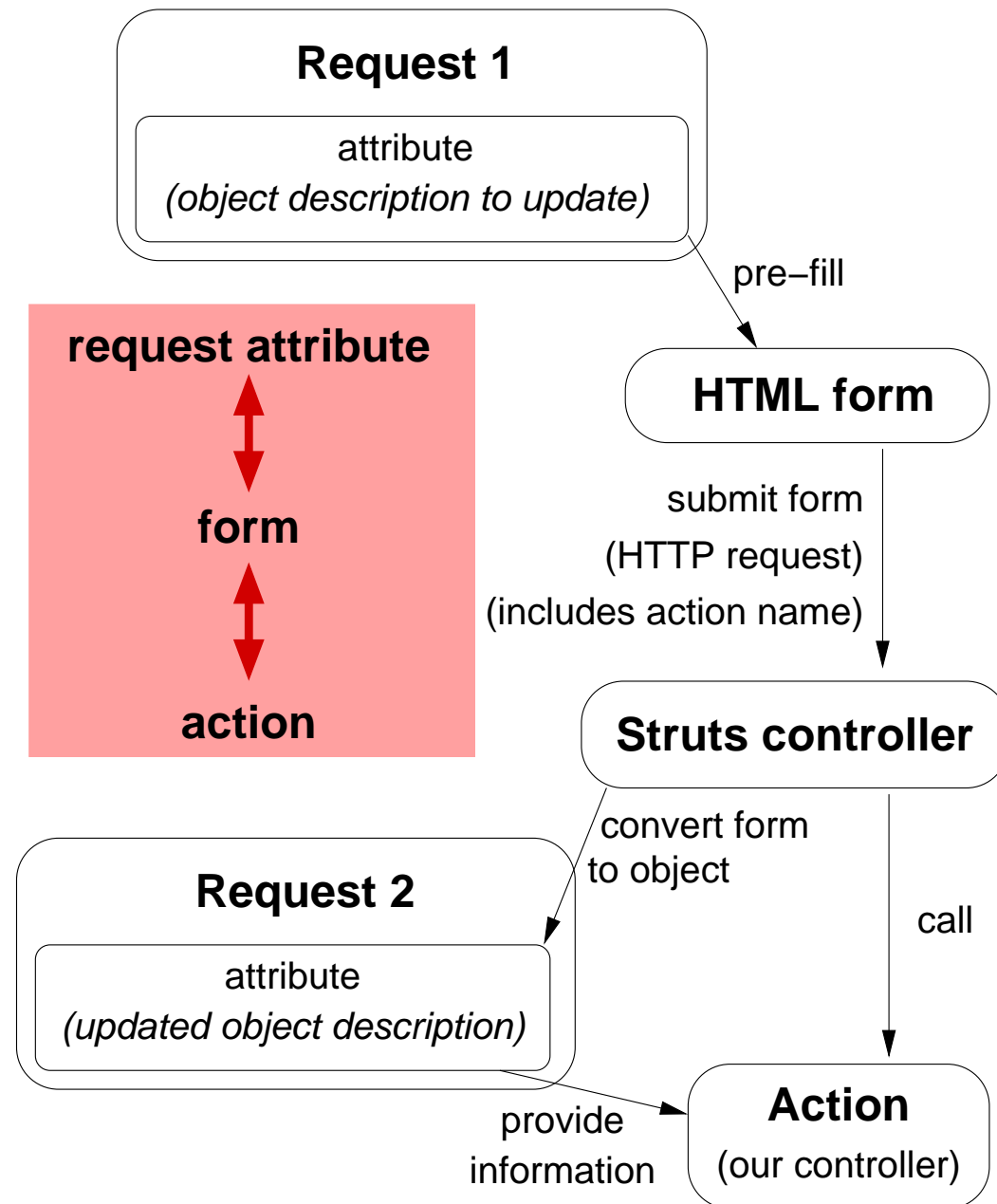
- ▷ Written in Java+XML

- Java: program fragments
- XML: declares model and view components to Struts

THE BASIC OPERATION OF STRUTS

Related elements

- ▷ **A form**
 - in HTML (almost)
- ▷ **An attribute of request**
 - corresponds with the form:
 - * the form is pre-filled with data from the attribute
 - * the attribute receives data filled into the form
- ▷ **An action**
 - is called by Struts when the form is submitted
 - receives the attribute



CONNECTION BETWEEN THE FORM AND THE ACTION

The submission of a form is a request for a resource

- ▷ a URI identifies the resource (i.e., the action that will act upon the form)
- ▷ the form contains the URI
 - i.e., the form says what to do with its content

Example:

```
<form:form method="GET"  
           action="updateArticle.do">  
...  
</form:form>
```

CONNECTION BETWEEN THE FORM OBJECT AND THE ACTION

A form is connected to a form object (form bean)

- ▷ from which the object is pre-filled
- ▷ into which the form is debriefed

We declare the connection between the form object and the **action**

- ▷ and not the form
 - looks strange, but is equivalent

Example:

In file ~/jsp/WEB-INF/struts-config.xml

```
<struts-config>
  ...
  <action-mappings>
    ...
    <action path="/search"
            type="sudety.SearchAction"
            name="searchForm"
            scope="request">
      ...
    </action-mappings>
  ...
</struts-config>
```

CONNECTION BETWEEN OBJECTS AND CLASSES

The need: to specify the class of every object

- ▷ every action
- ▷ every form object

Examples:

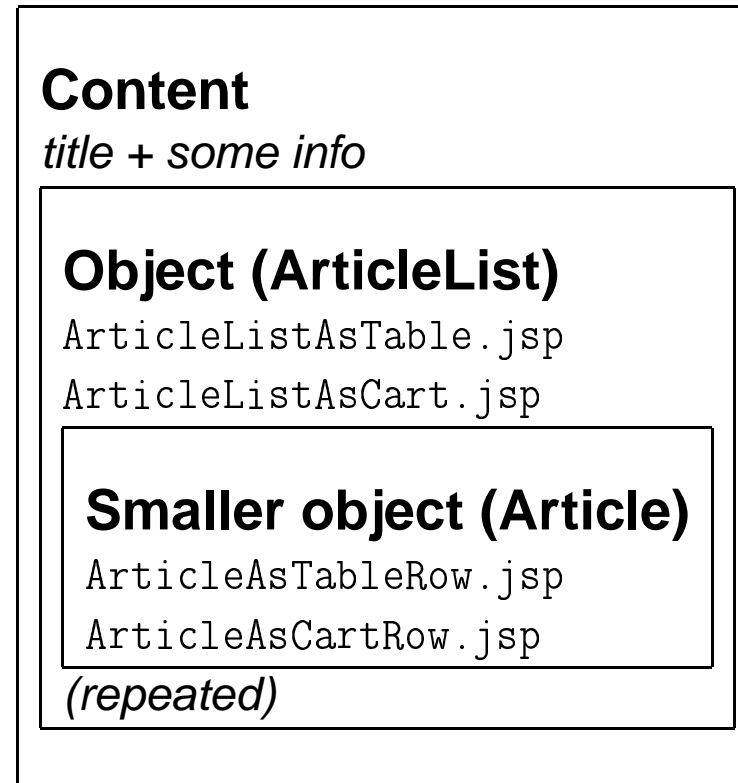
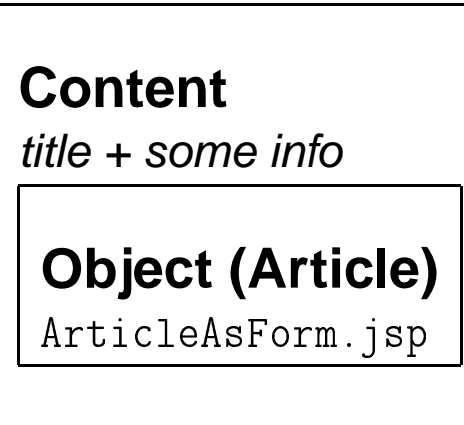
- ▷ for an action: see above
- ▷ for a form object (form bean):
 - In file `~/jsp/WEB-INF/struts-config.xml`

```
<struts-config>
...
<form-beans>
  <form-bean name="searchForm"
             type="sudety.SearchForm"/>
  <form-bean name="sudetyArt"
             type="sudety.Article"/>
</form-beans>
...
</struts-config>
```

VIEW: THE HIERARCHY OF A PAGE

1. **The whole page**
2. **Major component**
 - ▷ we always show 3 of them
 - ▷ (Struts framework; very common)
3. **Object** (personal choice)
4. **Smaller object** (personal choice)

Title	
	title.jsp
Menu choices.jsp	Content startup.jsp artSearchResults-content.jsp updateResults-content.jsp viewCart-content.jsp createArticle-content.jsp



HOW WE EXTEND THE JAVA OBJECT MODEL

The traditional model

- ▷ fields (data)
- ▷ methods (Java programs)

Our extension

- ▷ JSP webpage fragments

HOW FRAGMENTS ARE CHOSEN

- ▷ choice of action (our part of controller)
- ▷ choice of business logic called by action
- ▷ choice of webpage (main JSP file)
- ▷ choice of webpage fragments

Choice of action

- ▷ the URI identifies the action
- ▷ the method to execute is
<action's class>.execute()

remember the example:

```
<action path="/search"
        type="sudety.SearchAction"
        name="searchForm"
        scope="request">
```

Choice of business logic

- ▷ the action calls methods in various classes

example:

```
ArticleList.getCart  
    (request.getSession().getId())
```

Choice of webpage (main JSP file)

- ▷ each main JSP file is declared as a *forward* in file

```
~/jsp/WEB-INF/struts-config.xml
```

```
<global-forwards>  
    ...  
    <forward name="searchResults"  
            path="/artSearchResults.jsp"/>  
    ...  
</global-forwards>
```

- ▷ when an action finishes, it chooses a forward

example:

```
return mapping.findForward("update");
```

Choice of JSP fragments

- ▷ one JSP file includes JSP files

example:

```
<jsp:include page="ArticleAsTableRow.jsp" />
```

EXAMS

Written exams

- ▷ there are easy questions and hard questions
- ▷ easy questions compensate for hard ones
 - no adjustment of points (you win 0.7 to 3.1 points)
 - no leniency for question 1 (you lose up to 0.5 point)
- ▷ the BGP question: suppressed for those who answered (*b*):
 - in this case, exam graded using 15 questions
 - mathematically: grade multiplied by 16/15

Final exam

- ▷ at least two “very short essay” questions, graded without leniency
- ▷ multiple choice, at least four choices
- ▷ no negative points for wrong answers

The oral exam (“defense”):

- ▷ not really a “defense”
- ▷ you should have done the exercises
- ▷ you should understand the exercises
 - remember how they were done
 - be able to discuss them
 - be able to redo a small fragment
- ▷ you need not know unimportant details
 - an open-book oral exam (bring whatever notes you want)