Week 10

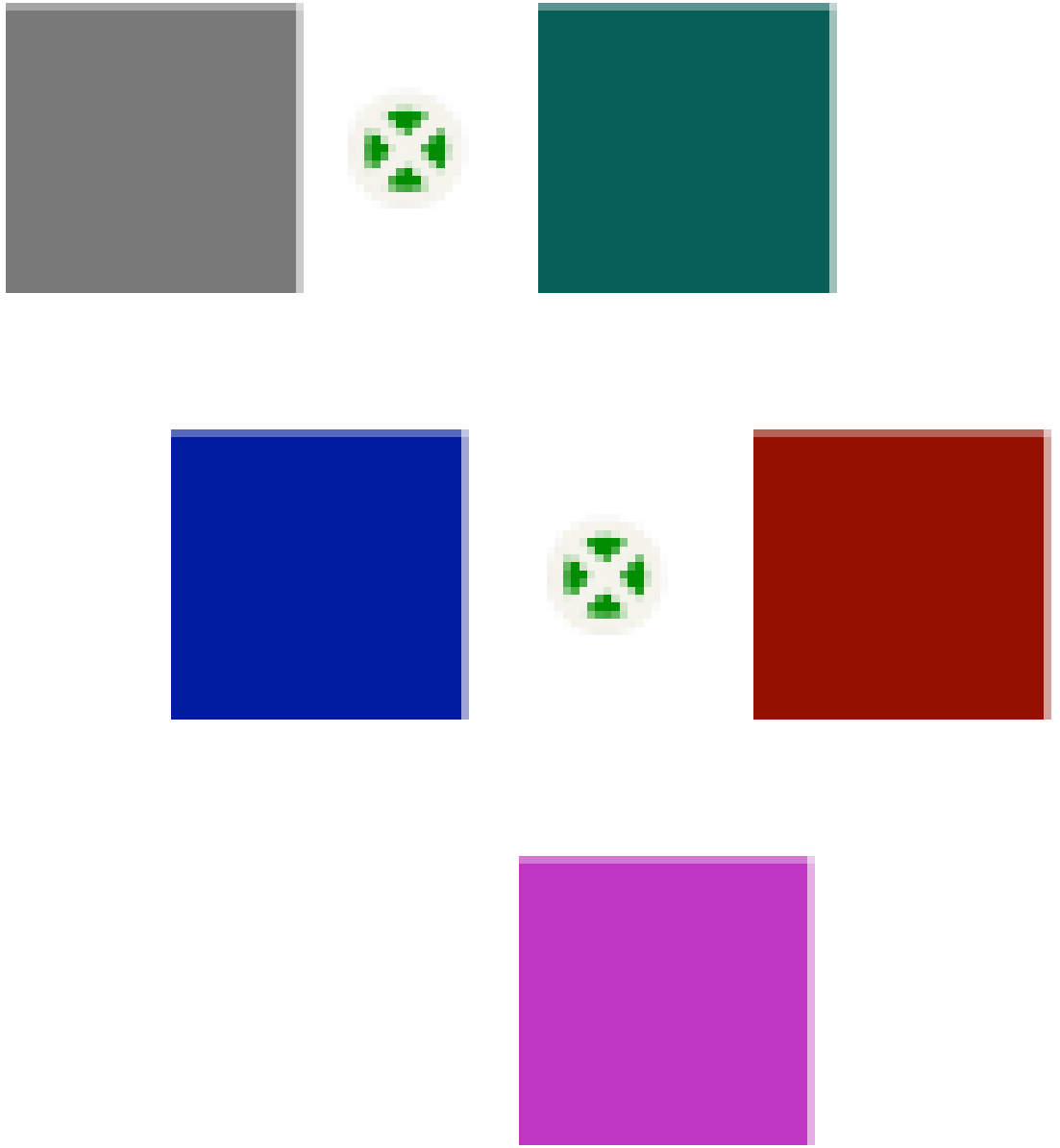
Working with databases

Fundamental Concept

- Tables are like variables
- But instead of containing one value they contain one set

KEY POINT: Tables are Variables

And with operations you can combine sets to obtain a result set and then may combine this with further operations until you get exactly the result set that you want…



Good Modelling is Needed

One thing that is very important is that tables should be well designed (they must follow a number of rules, called normal forms). You don't want for instance data to be repeated many times, because it would make changes difficult. You must also know precisely what uniquely identifies a row (it may be all columns, but most often it's one or a few columns)

Entities = “Existing Things”

- Student
- Advisor
- Course

When you design one, you look for “entities”, things that exist independently of the others (a course can be on a catalogue without anyone taking it or teaching it). You must know what identifies each item: a code, a student/employee id, mail address, phone number may be good identifiers. A persons name is not good because several people could have the same name. You then have attributes of the entities (such as name). One entity will be one table.

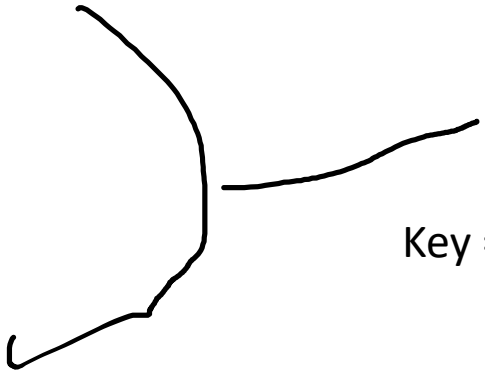
Relationships

Then you can have relationships between entities – that link entities together. If an entity can be linked to only one other entity (for instance Student → Dormitory) it can be an attribute of an entity.

Student

Advisor

Course



Session

Key = combination of entity keys (StudentID, Advisor ID, Course ID)

Often it will be a table relating identifiers because a student takes many course and there are many students in a course.

Normalization

- Distinguishing between what is an attribute and what should be in a relation is often difficult.
- All this business of organizing tables is rather difficult and often underrated.
- If poorly done it can cause many problems.

What identifies a customer?

- Name
- Email
- Phone number
- Generated customer ID

If you wanted to make a database of customers of an ecommerce site you could potentially use a name (but not unique), an email or a phone, but you are not supposed to modify an identifier. So it may be better to generate an ID.

SQL – Structured Query Language

SEQUEL: A STRUCTURED ENGLISH QUERY LANGUAGE

by

Donald D. Chamberlin
Raymond F. Boyce

IBM Research Laboratory
San Jose, California

Databases are usually associated with a query language called SQL. That was invented in the early 1970s.

It was originally proposed in the paper ”A Structured English Query Langauge” by Don Chamberlin and Ray Boyce in 1974: https://researcher.watson.ibm.com/researcher/files/us-dchamber/sequel-1974.pdf

ABSTRACT: In this paper we present the data manipulation facility for a structured English query language (SEQUEL) which can be used for accessing data in an integrated relational data base. Without resorting to the concepts of bound variables and quantifiers SEQUEL identifies a set of simple operations on tabular structures, which can be shown to be of equivalent power to the first order predicate calculus. A SEQUEL user is presented with a consis-

SQL

- SQL was designed to be a very simple language with a syntax similar to English that could be used by people who are not familiar with computers or programming.

```
SELECT ..
FROM …
WHERE …
```

- It became at the same time a major success and a major failure: today only computer programmers use SQL – but almost all of them have to use it, and sometimes very often.

Two Main Components

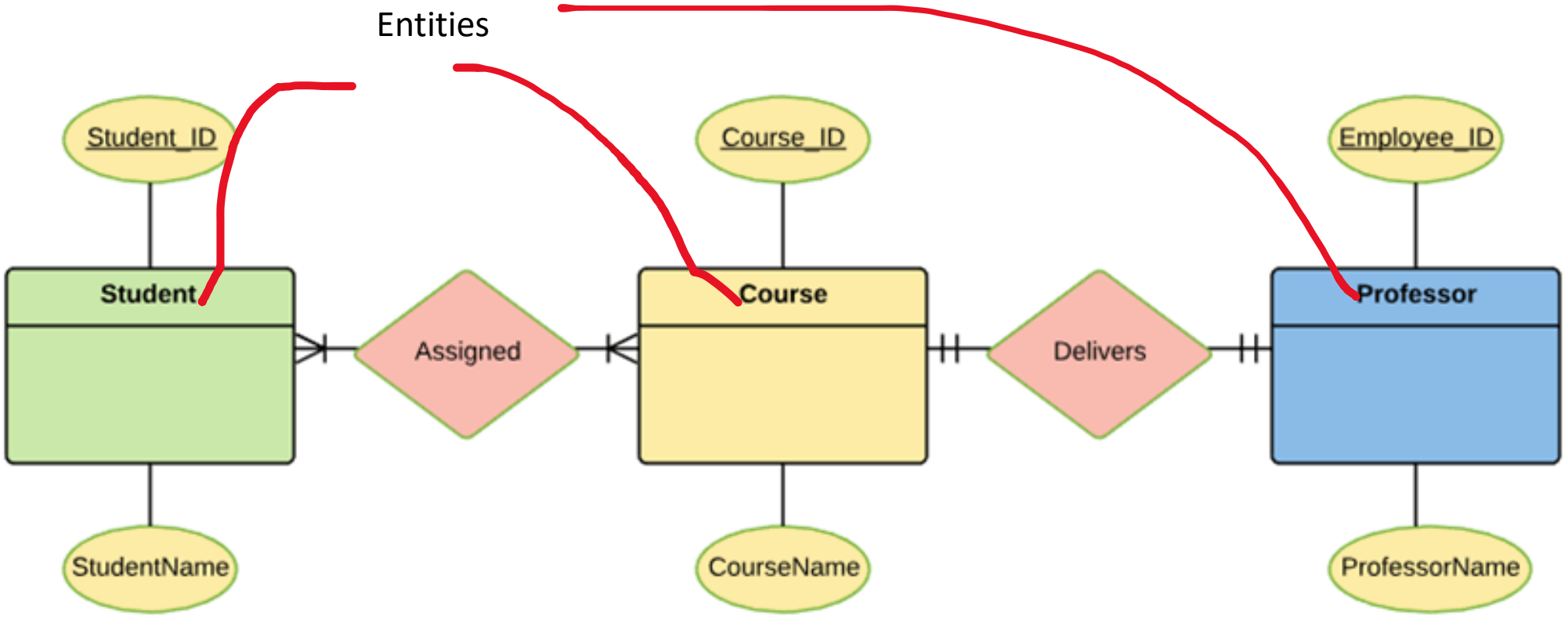
1. SQL provides commands for managing tables (creating, dropping, modifying them)
2. SQL provides commands for managing data (inserting new rows, updating or deleting existing ones – plus of course commands for retrieving data that satisfies some criteria).

You can learn SQL syntax in about the same time as you can learn how chess pieces move. As with chess, things however quickly become complicated.

Simple Database

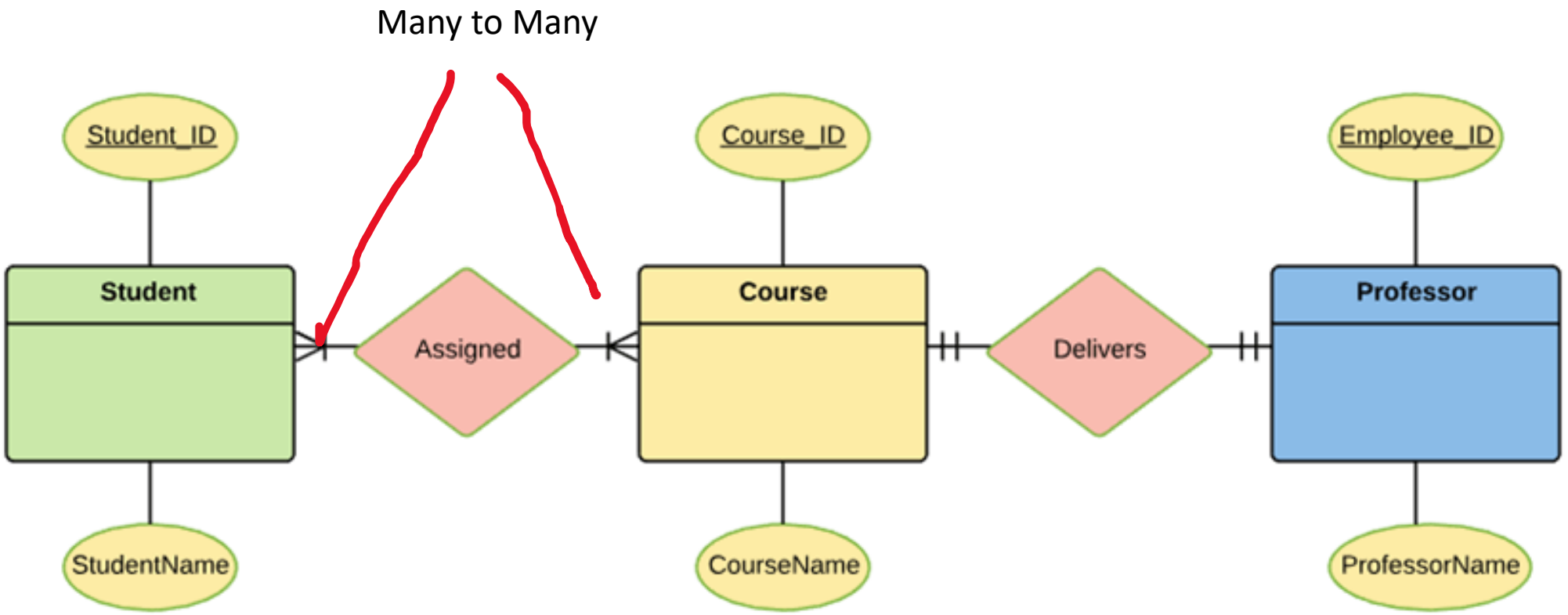
This small database represents Students, Courses and Professors.

Entity	Primary Key	Attribute
Student	Student_ID	StudentName
Professor	Employee_ID	ProfessorName
Course	Course_ID	CourseName



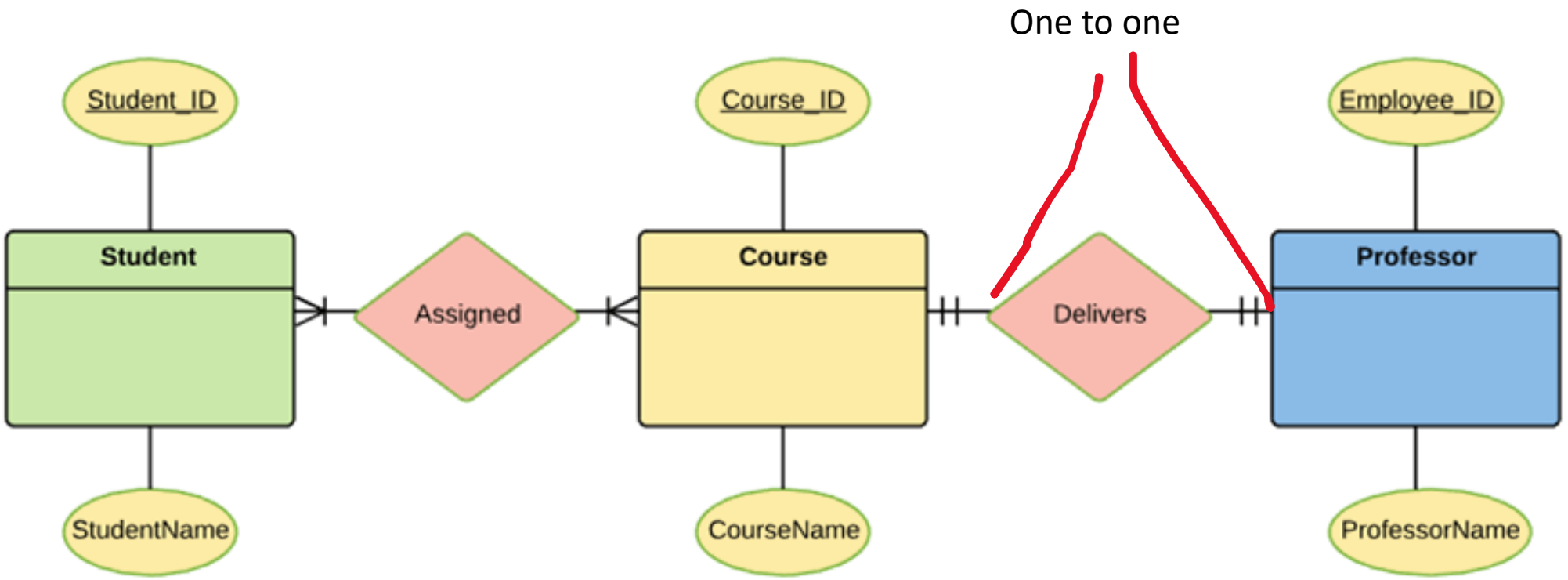
Simple Database

This small database represents
Students, Courses and Professors.



Simple Database

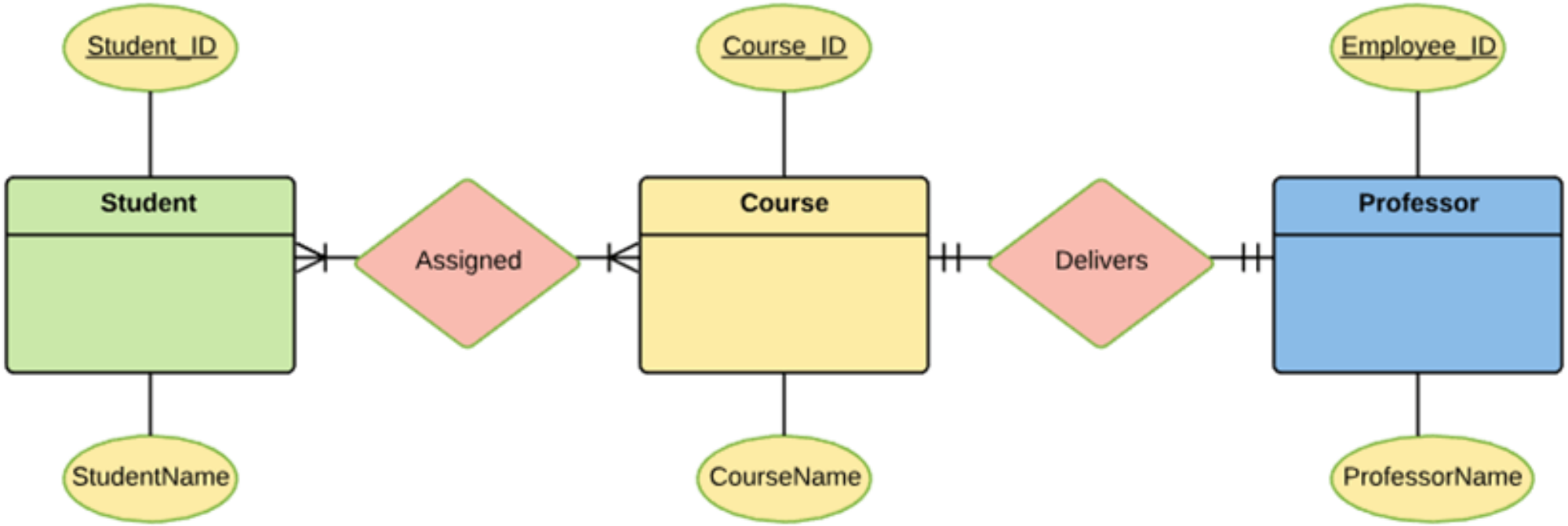
This small database represents
Students, Courses and Professors.

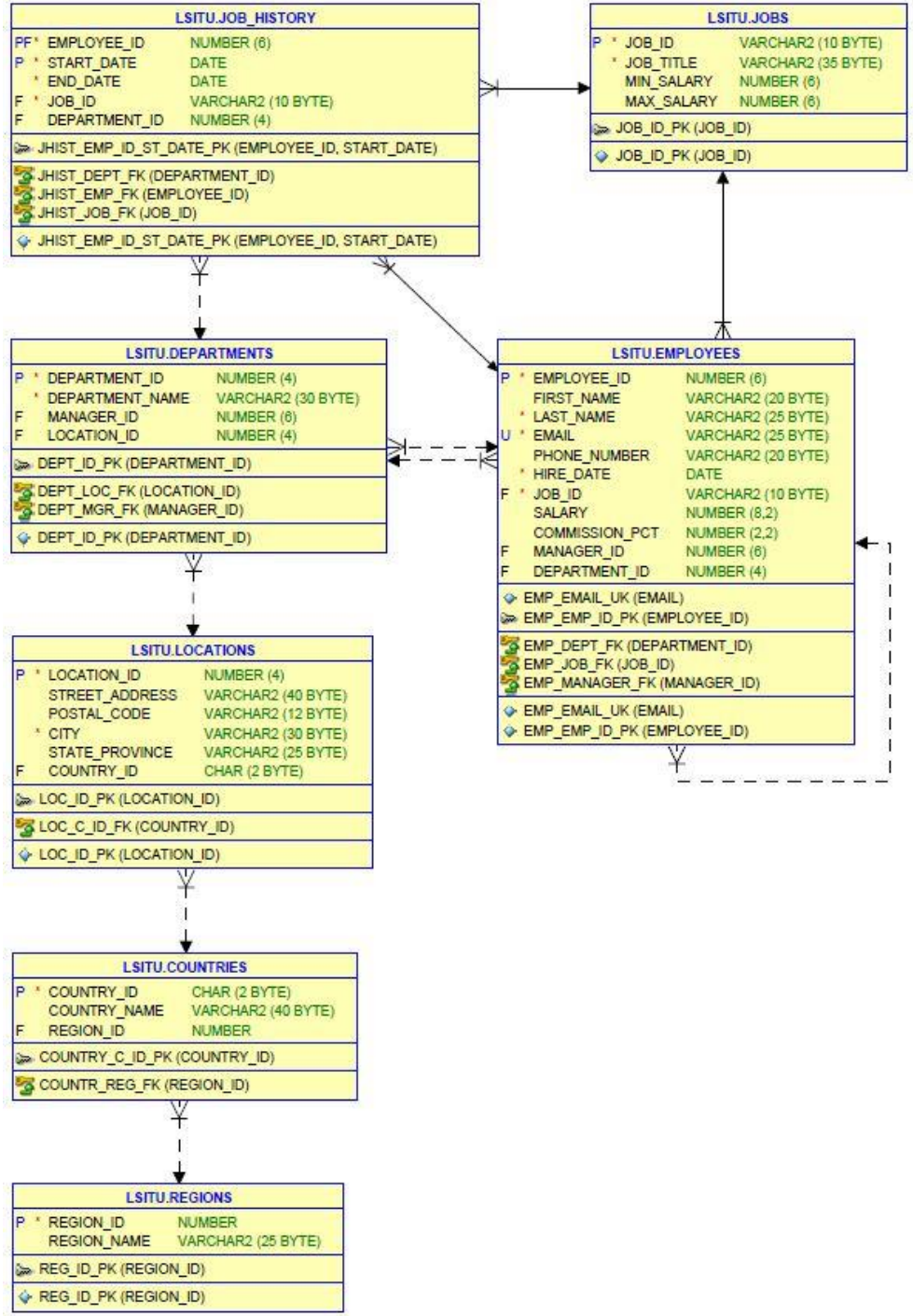


Simple Database

This small database represents Students, Courses and Professors.

Entity	Primary Key	Attribute
Student	Student_ID	StudentName
Professor	Employee_ID	ProfessorName
Course	Course_ID	CourseName



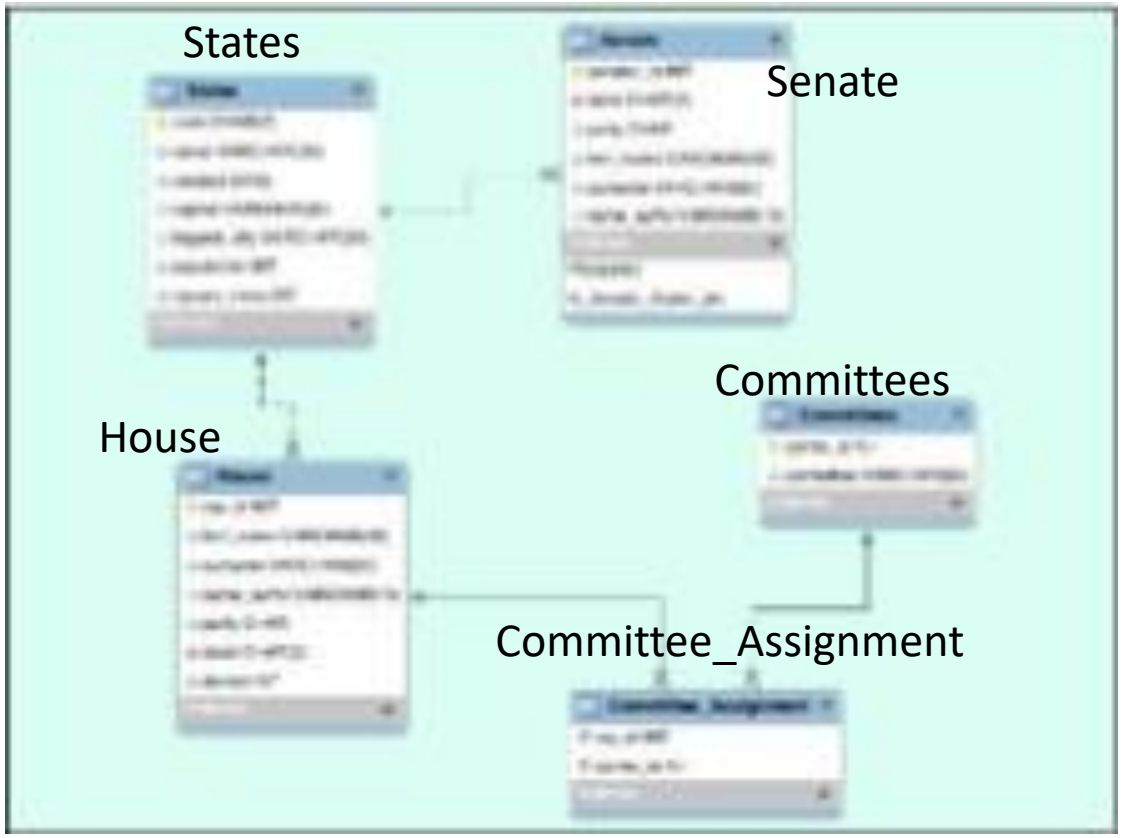


Here is another database which contains information about employees at a company. Not overly complex (7 tables with a few attributes in each).

And here is a query designed to fulfil a task that was "to display all employees and their related info even if some info is missing. Get as much information as you can about the employees".

```
SELECT
  e.employee_id AS "Employee #", e.first_name || ' ' || e.last_name AS "Name"
  , e.email AS "Email"  , e.phone_number AS "Phone"
  , TO_CHAR(e.hire_date, 'MM/DD/YYYY') AS "Hire Date"
  , TO_CHAR(e.salary, 'L99G999D99', 'NLS_NUMERIC_CHARACTERS = ''.,'' NLS_CURRENCY = ''$''') AS "Salary"
  , e.commission_pct AS "Comission %"
  , 'works as ' || j.job_title || ' in ' || d.department_name || ' department (manager: '
    || dm.first_name || ' ' || dm.last_name || ') and immediate supervisor: ' || m.first_name || ' ' || m.last_name AS "Curr
  , TO_CHAR(j.min_salary, 'L99G999D99', 'NLS_NUMERIC_CHARACTERS = ''.,'' NLS_CURRENCY = ''$''') || ' - ' ||
      TO_CHAR(j.max_salary, 'L99G999D99', 'NLS_NUMERIC_CHARACTERS = ''.,'' NLS_CURRENCY = ''$''') AS "Current Salary"
  , l.street_address || ', ' || l.postal_code || ', ' || l.city || ', ' || l.state_province || ', '
    || c.country_name || ' (' || r.region_name || ')' AS "Location"
  , jh.job_id AS "History Job ID"
  , 'worked from ' || TO_CHAR(jh.start_date, 'MM/DD/YYYY') || ' to ' || TO_CHAR(jh.end_date, 'MM/DD/YYYY') ||
    ' as ' || jj.job_title || ' in ' || dd.department_name || ' department' AS "History Job Title"
FROM employees e
-- to get title of current job_id
  JOIN jobs j
    ON e.job_id = j.job_id
-- to get name of current manager_id
  LEFT JOIN employees m
    ON e.manager_id = m.employee_id
-- to get name of current department_id
  LEFT JOIN departments d
    ON d.department_id = e.department_id
-- to get name of manager of current department
-- (not equal to current manager and can be equal to the employee itself)
  LEFT JOIN employees dm
    ON d.manager_id = dm.employee_id
-- to get name of location
  LEFT JOIN locations l
    ON d.location_id = l.location_id
  LEFT JOIN countries c
    ON l.country_id = c.country_id
  LEFT JOIN regions r
    ON c.region_id = r.region_id
-- to get job history of employee
  LEFT JOIN job_history jh
    ON e.employee_id = jh.employee_id
-- to get title of job history job_id
  LEFT JOIN jobs jj
    ON jj.job_id = jh.job_id
-- to get namee of department from job history
  LEFT JOIN departments dd
    ON dd.department_id = jh.department_id
ORDER BY e.employee_id;
```

If you want read more about this query and an explanation here (the purpose of the slide is just to give an example of how a query in SQL can become complex… https://dev.to/tyzia/example-of-complex-sql-query-to-get-as-much-data-as-possible-from-database-9he



is small database represents for instance part of the US Congress, with the Senate and House of Representatives.

Who participates in the greatest number of House Committees ?

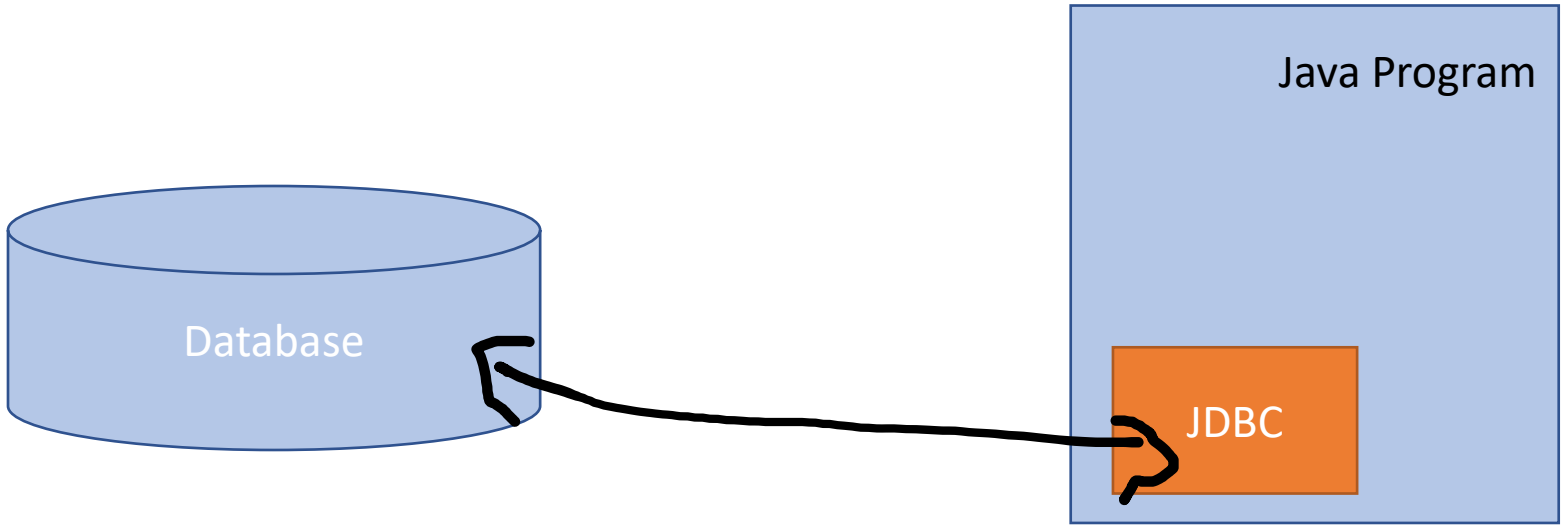
```sql
select h.first_name, h.surname,
            s.name as state, z.num_of_committees
from (select rep_id,
        count(*) as num_of_committees
    from committee_assignment group by rep_id
    having count(*) =
        (select max(num_of_committees) from (select rep_id,
        count(*) as num_of_committees from committee_assignment
        group by rep_id) x)) z
    join house h
        on h.rep_id = z.rep_id
    join states s
        on s.code = h.state
by h.surname, h.first_name, s.name

```

Next

Accessing databases in java

JDBC

Java Database Connectivity (JDBC)

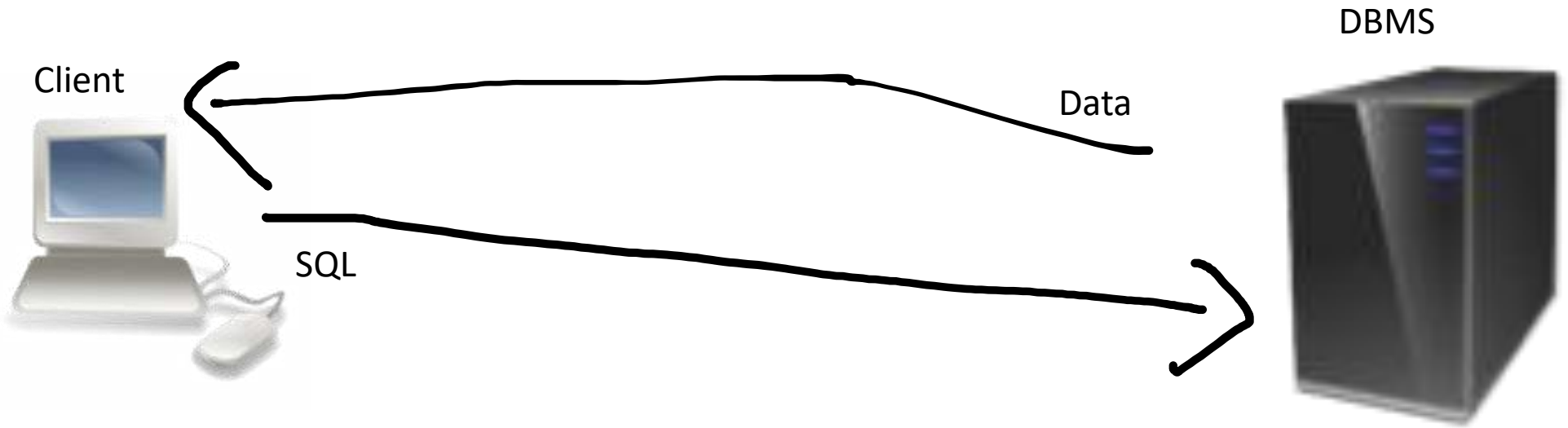


Accessing a database and sending/executing SQL from a Java program is easy, even without using some tools that try to write queries for you (sometimes these tools even write inefficient queries).

Accessing a database from a Java program

Classic case for a real database management server:

Your program must first connect to the database (which usually required supplying a username and a password) then will send strings that contain an SQL command and will be executed by the server. Some commands will only return success or failure (creating a table, for instance). A command such as an update will be able to tell you how many rows were changed. A query will return you rows, between 0 and n...



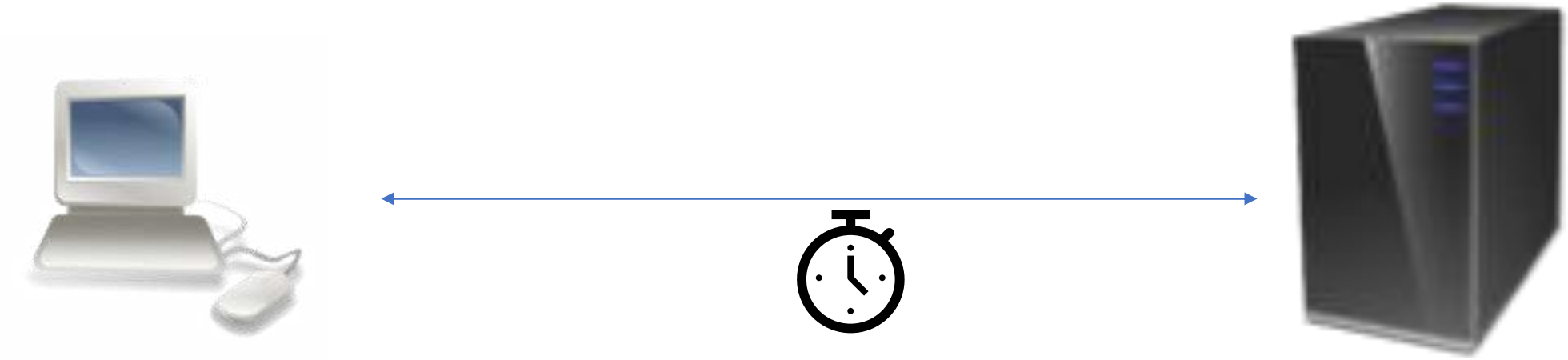
System Architecture: Classic Design Mistake

- Many Java developers do this wrong…
- Must SHARE the work between what the database does ...
  - Retrieve exactly what you need

  If you want a sum, don't retrieve all the data and iterate to sum it. Ask for the sum. The DBMS can compute it.

- And what the java program does
  - Presentation computations that cannot be performed by the DBMS

System Architecture: Consider Network Latency



Don't multiply exchanges between your application and the server. Travelling in networks takes time (it's called latency), even with a lot of bandwidth. Sometimes data centres are very far away.

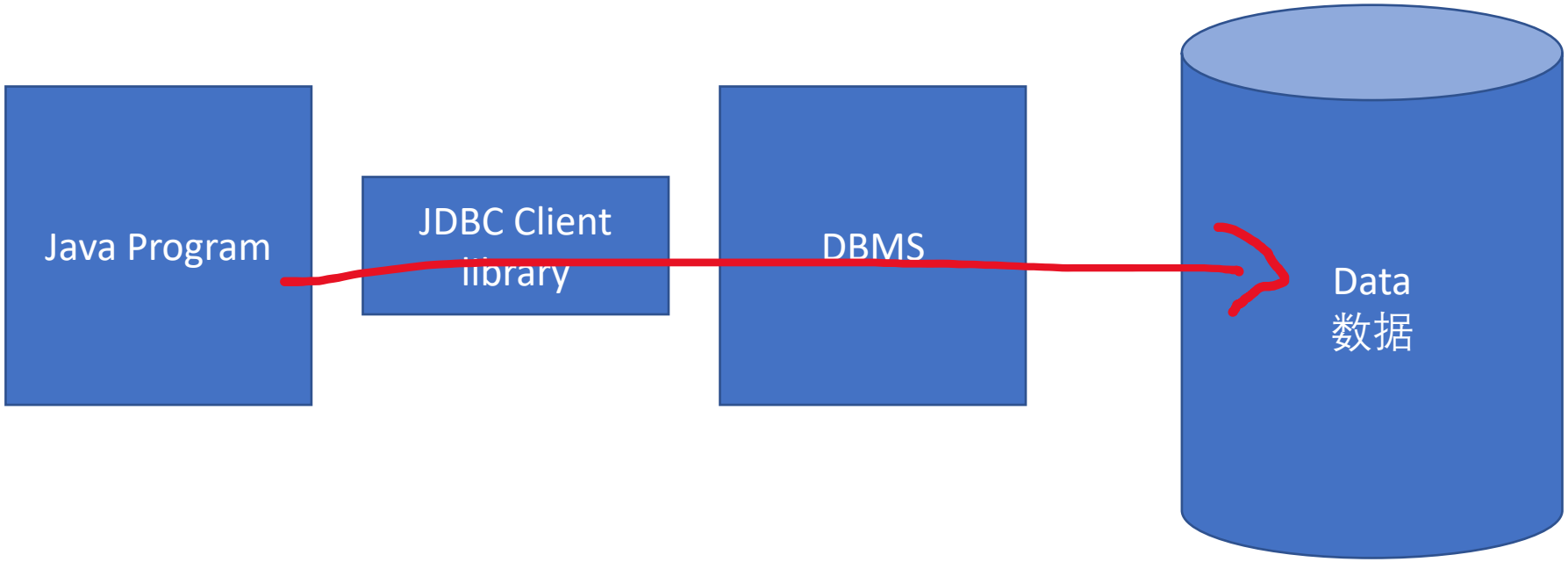
System Architecture: One INSERT is okay but 10K INSERTs is BAD

If you want to insert many rows into a remote database, send all the rows to the server and ask it to insert all the rows at once (this is called "batching"); it will do it very fast. If you insert rows one by one, you have to wait each time for the response from the server. A return trip between Singapore and Tokyo takes about 0.075s. If you assume that inserting 10,000 rows takes 0.050s* doing it as one batch between Singapore and Tokyo would take 0.125s. Inserting one row about 0.003s*. Inserting 10000 rows row by row would be (0.075 + 0.003) * 10000 = 780s, or 13 minutes ...

USE BATCHING

*tests by Stephane Faroult in 2017

Even when you are close…



Even when running on the same server, switches between a program and the DBMS are costly.

System Architecture Rules

- Share work between the DBMS and the program
- Consider network latency
- Use batching

Embedded Databases

An alternative to real database servers are "embedded databases", systems that let you use of file as if it were a database.

No server just for a single-user

"Connection" same as opening a file

Everything else like the real thing

Apache Derby

- Pure Java
- Part of the JDK
- Server or embedded

- Java is shipped with "Derby", sometimes called "Java DB"

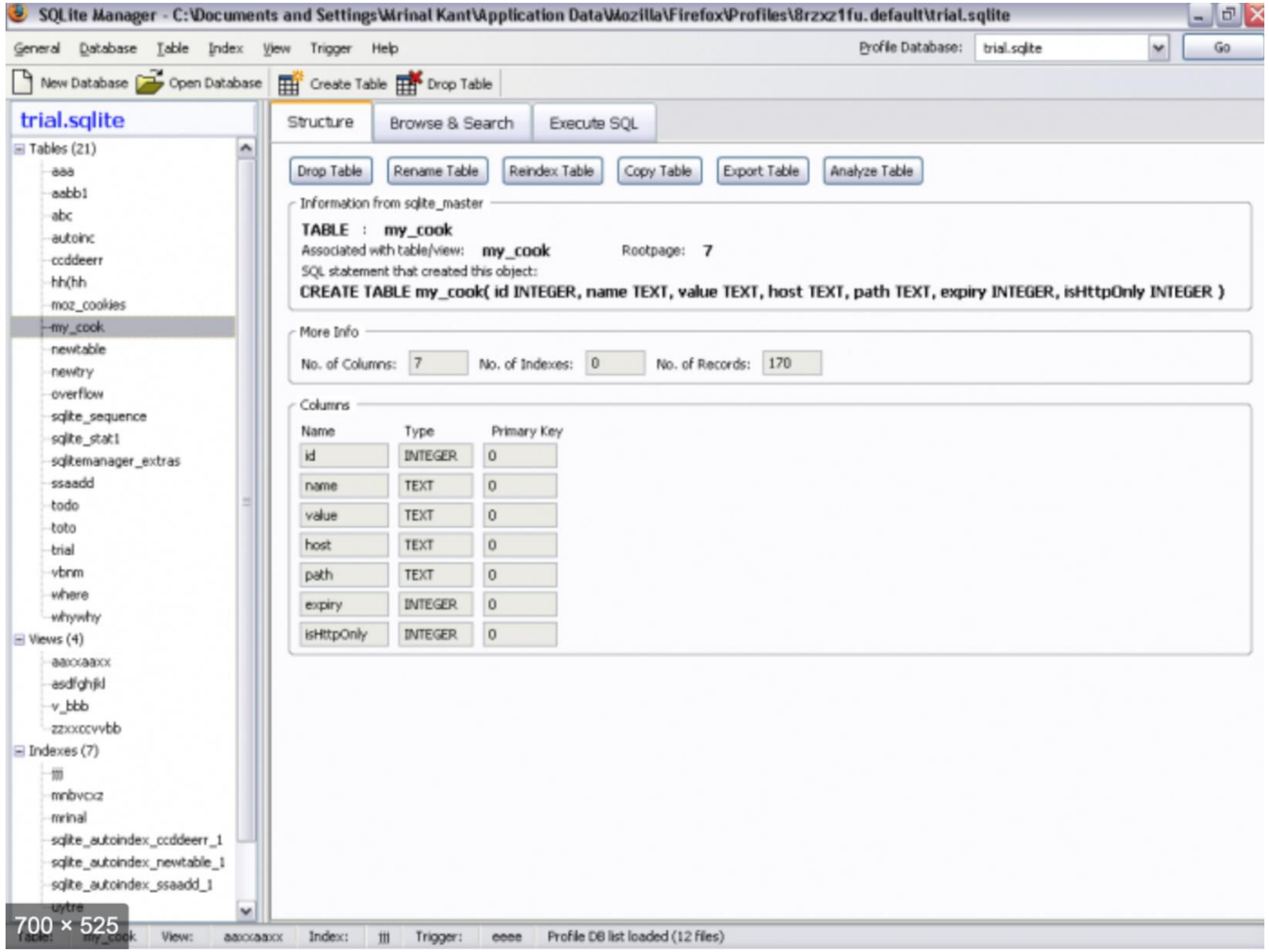
SQLite

- Public domain
  One file to download

- Used in mobile apps – and by Mozilla

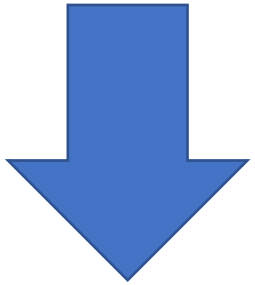
- Create tables in *SQLite Manager* (a firefox plugin)

The most popular embedded database is SQLite.



- Can directly execute the SQL from the manager

www.sqlite.org



Only for C programming language

JDBC: https://github.com/xerial/sqlite-jdbc

You just need a .jar file available from this address…

Java Database Connectivity (JDBC)

sqlite-jdbc refers to JDBC, which is a protocol allowing access to almost all databases from Java. SQL is often slightly different from database to database, not JDBC calls.

JDBC

- Load a specific connection driver (a .jar file)

  Database specific connection parameters

- Then java methods are the same with every database

- SQL is slightly different between databases

3 Main Objects

- **Connection**
  Link to a database
- **PreparedStatement**
  SQL commands:
- **ResultSet**
  Returned results

You can execute a query and do things like loop on rows returned. There is also a Statement object. A PreparedStatement can be re-executed with different parameters, but a Statement is only sent once.

Java/JDBC Example

```
import java.util.Properties;
import java.sql.*;
import java.util.Scanner;

class DBExample {
    static Connection con = null;

    public static void main(String arg[]) {
        Properties info = new Properties();
        String url = "jdbc:oracle:thin:@localhost:1521:orcl
        Scanner input = new Scanner(System.in);

        try {
            Class.forName("oracle.jdbc.OracleDriver");
        } catch(Exception e) {
            System.err.println("Cannot find the driver.");
            System.exit(1);
        }
```

- JDBC
- Caution: the driver must be in the classpath
- You connect to the database using a url string not a URL object…

Java/JDBC Example

```
try {
    Class.forName("oracle.jdbc.OracleDriver");
} catch(Exception e) {
    System.err.println("Cannot find the driver.");
    System.exit(1);
}

try {
    System.out.print("Username: ");
    String username = input.nextLine();
    System.out.print("Password: ");
    String password = input.nextLine();
    info.put("user", username);
    info.put("password", password);
    con = DriverManager.getConnection(url, info);
    con.setAutoCommit(false);
    System.out.println("Successfully connected.");
} catch (Exception e) {
        System.err.println(e.getMessage());
        System.exit(1);
}
```

- MySQL wants username and password independent parameters,
- Oracle passes them as Properties.
- Of course the connection can fail.

Java/JDBC Example

```
System.out.print("Date: ");
String utcDate = input.nextLine();
try {
    PreparedStatement stmt = con.prepareStatement("select region, count(*)"
            + " from quakes"
            + " where UTC_date >= ?" + " group by region");
    stmt.setString(1, utcDate);
    ResultSet rs = stmt.executeQuery();
    while (rs.next()) {
        System.out.println(rs.getString(1) + "\t" + rs.getString(2));
}
rs.close();
```

- Queries are simple. Placemarkers for parameters in a PreparedStatement are ? marks, implicitly numbered (from 1). Return columns are also numbered from 1. One can often only work with Strings, the DBMS can convert types.

Java/JDBC Example

```
            rs.close();
    } catch (Exception e) {
        System.err.println(e.getMessage());
        try {
            con.close();
        } catch (SQLException sqlE) {
        // Ignore
        }
        System.exit(1);
    }
    try {
        con.close();
    } catch (SQLException sqlE) {
    // Ignore
    }
```

Queries can fail too, but returning nothing or updating or deleting or inserting no rows doesn't throw any exception because the empty set is a valid set … However trying to insert the same key for instance will throw an exception

Common drivers

```
Class.forName("oracle.jdbc.OracleDriver");
Class.forName("com.mysql.jdbc.Driver");
Class.forName("org.postgresql.Driver");
Class.forName("org.sqlite.JDBC");
Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
```

These are JDBC driver names for commonly use Database Management systems (note that for Derby there are two modes, embedded – single user – and not embedded). Note also that although most tutorials use reflection to load the driver, you can also use import with it, especially with Derby, as the driver will always be in your class path.

Connection Examples

```
con = DriverManager.getConnection(url, info);
    "jdbc:oracle:thin:@hostname:port:dbname"
    "jdbc:postgresql://hostname:port/dbname"
     user password

 con = DriverManager.getConnection(url,
                                   username,
                                    password);
    "jdbc:mysql://hostname:port/dbname"

 con = DriverManager.getConnection(url);
    "jdbc:sqlite:filename"
```

- And here are connection examples.
- With Sqlite you just provide a filename. It will be created if it doesn't exist already.

Working with DB Systems

Relational Databases

Relational Databases Databases

- SQL databases
  - MySQL
  - PosgreSQL
  - Oracle
  - Apache Derby (packaged with Java)
- Embedded
  - These are highly integrated with an application (just a single user)
  - SQLite, Embedded Derby

Connection Examples

```
con = DriverManager.getConnection(url, info);
    "jdbc:oracle:thin:@hostname:port:dbname"
    "jdbc:postgresql://hostname:port/dbname"
     user password

 con = DriverManager.getConnection(url,
                                   username,
                                    password);
    "jdbc:mysql://hostname:port/dbname"

 con = DriverManager.getConnection(url);
    "jdbc:sqlite:filename"
```

- And here are connection examples.
- With Sqlite you just provide a filename. It will be created if it doesn't exist already.

Data Sources

- Configured by System Administrator

```
Context context = new InitialContext();
DataSource ds = (DataSource) context.lookup(DBname);
Connection con = ds.getConnection(username, password);
```

There is another way for getting JDBC connections, which is using a DataSource object. It abstracts the database (it uses a directory saying what is the servername and database name and database type for a given "service"), and in a big company it allows moving databases between servers transparently for users.

Also useful for switching between development and production environments…

Another JDBC Package

import javax.sql.*;

Data sources

Connection pools

Distributed transactions

DataSources and other advanced features used mostly in very big companies (connection pools are mostly useful when you have thousands of users, and distributed transactions are required to coordinate work across several databases) are inside an additional package called javax.sql (regular JDBC is java.sql)

SECURE database systems – login credentials

- Don’t hardcode username and password information
- Instead use a property file: more secure and can be updated

- Usernames and passwords should never be hardcoded in programs, because it's a common policy to force passwords to change every 3 or 6 months.
  - Users should be prompted for their credentials
  - Or if the program is run at 2am, read from a file. But then you should make sure that only the file owner can read it!

SECURE database systems – using SQL

- **DON'T** append user input to SQL statements in the program
- **DO** use PreparedStatements with parameters

It's very tempting to use "+" to build an SQL statement. The rule is that whenever there is a constant in the SQL command that was input by a user, it should be passed as a parameter to a PreparedStatement, not simply added.

SECURE database systems – using SQL Prepared Statements

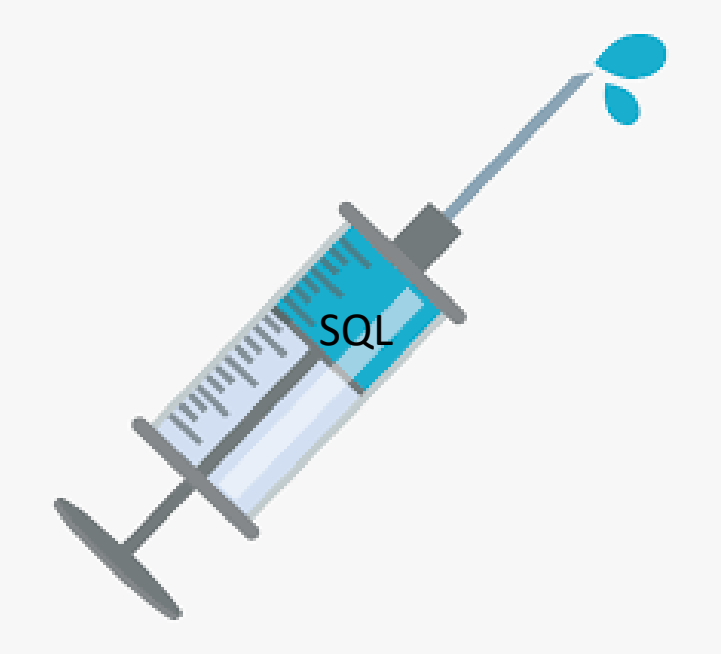
```
System.out.print("Date: ");
String utcDate = input.nextLine();
try {
    PreparedStatement stmt = con.prepareStatement("select region, count(*)"
            + " from quakes"
            + " where UTC_date >= ?" + " group by region");
    stmt.setString(1, utcDate);
    ResultSet rs = stmt.executeQuery();
    while (rs.next()) {
        System.out.println(rs.getString(1) + "\t" + rs.getString(2));
}
rs.close();
```

Making Queries with prepared statements is simple. Placemarkers for parameters in a PreparedStatement are ? marks, implicitly numbered (from 1). Return columns are also numbered from 1. One can often only work with Strings, the DBMS can convert types.

SECURE database systems – using SQL

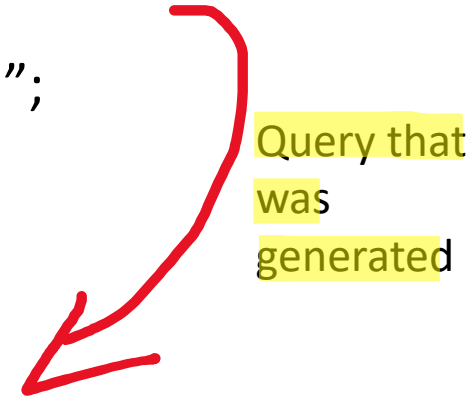
There are two reasons for this rule. One is linked to how a DBMS such as Oracle or SQL Server works, and is outside our scope (it gives better performance).

The second reason is a famous hacking technique known as "SQL injection"…



SQL Injection

```
query = "select memberid from members where username = ‘”
        + enteredUsername
        + "' and password = '" + enteredPassword . ”””;

        select memberid from members where
                username = 'Mickey’
                and password = '3#(ag89zhost]=89’
```



Most web applications store a table of members with the login name, and the crypted password. When users log in, the table is searched for a row containing their name and the encryption of the password they have entered is compared to the value stored. If the query above returns no rows, the user is rejected, otherwise s/he enters the site. When a developer builds the query as coded above, the type of expected input is shown in red

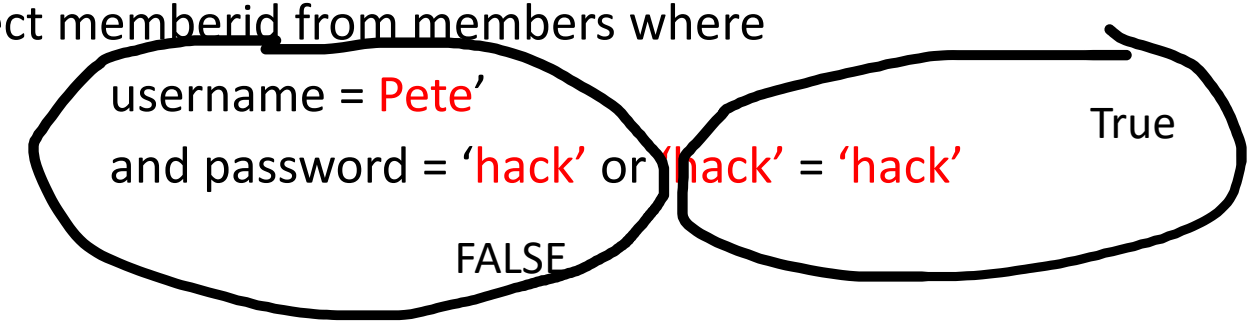
SQL Injection

query = "select memberid from members where username = ‘”

+ enteredUsername

+ "' and password = '" + enteredPassword . ”’”;

select memberid from members where



But if someone enters any name, let's say Pete, and something such as hack' or 'hack'='hack as password, when this is added to the statement it changes it. Because of the priorities rules between and and or, it will become a condition that is always false or a condition that is always true, and the query will always return rows, thus granting access to the site.

Two solutions:

Escape quotes in input
Use variables

To prevent such an attack, you can apply a function that escapes quotes to make sure that the quotes are understood as belonging to the constant (not as belonging to the SQL command), or to use variables in a PreparedStatement. The second is better because as said above it may make a performance difference with some database management systems.

IMPORTANT *friendly reminders*

MINIMIZE database accesses
DON'T do in Java what SQL does better!

Remember that if Object Oriented programming is mostly about exchanging messages between objects, talking to a remote database is completely different, and remember that database management systems process data better and faster than you can do – they were built for that.

It’s especially more important because many Java developers have it completely wrong, partly because of available tools.

Transactions and Batching

To simplify, by default every time you change something in the database JDBC requires the DBMS to write something to a file to make sure it's not lost (it's called "autocommit"). It's very bad for massive operations, and it's very bad for linked operations that should all succeed or all fail. You can also ask the database to perform several operations in one go (this is called "batching"), or adjust how much data is retrieved from the database in one return trip. All this is quite important when an application is heavily used.

JPA to Object Relational Mapping

JPA = Java Persistence API

- Because Java developers are assumed, the poor things, to be unable to code SQL, tools have been developed to allow them to focus on "business functions", and here things become quite worrisome ...
- JDBC allows you to write some good applications; you cannot say that it's very difficult to use, but it requires you to know some SQL, a language that is far less easy than it looks.
- Like everything, it takes time to learn. Tools have been designed to let anybody write database applications without knowing SQL. Unfortunately then anybody did write database applications, and the result was often not pretty.

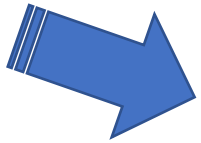
Persistence

- An important part of a software product
- Time spent searching and storing objects must be as short as possible
- Developers – focus on business functions 😉

Everybody acknowledges that persistence is very important, and that you cannot write a serious enterprise application without a database as backend. Given that many developers were writing terrible queries that were taking hours, the idea was to say "we'll generate database accesses for you and developers will no longer have to worry about the hard technical stuff but will able to focus on business functions". Which usually translates as "we'll be able to hire cheap beginners to do the job instead of expensive experienced people" (but they won’t tell you that).

Enterprise Java Bean

Increa$e productivity!
Focu$ on bu$ine$$!



Manage the business layer
(including persistence)

It all started with IBM, and it was incorporated later by Sun (then owner of Java) into an "Enterprise Edition" of Java for big companies.

How can we persist objects in a more advanced way than serialization?

A Trend in the 1990s – A new type of Database

- Object-Oriented Databases (OODBMS)

The big idea of the 1990s was the "Object Oriented Database", a kind of super-serialization. It never took off. Companies didn't want to take the risk of moving their data from relational databases that had been around for 10 to 15 years to untried products.

FAILED!

Learning to Live with Databases

Big database vendors paid lip service to object orientation, and Java developers had to look for bridges to the relational world.

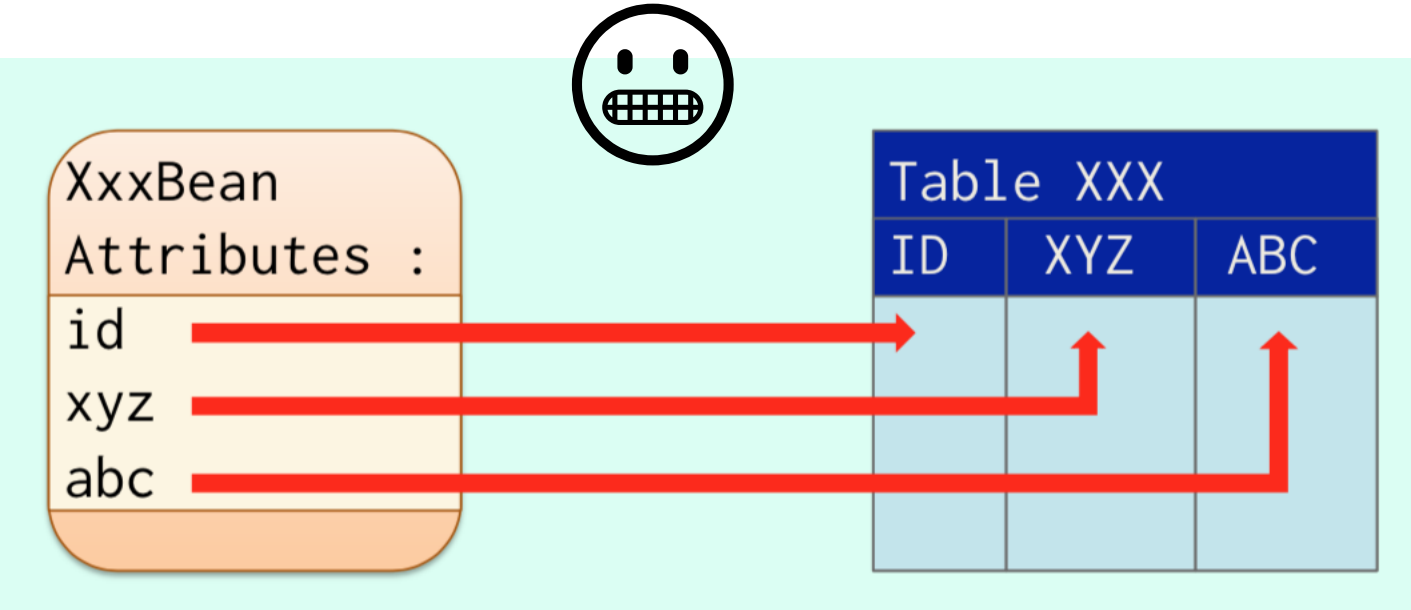
Object Relational Mapping

- One of the early products was Hibernate, which maps Java objects to database tables.

- Started early 2000s
- Simple to use
- Compatible with multiple database systems

- Good in simple cases

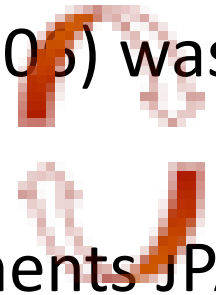
Object Relational Mapping



The idea is to have objects that correspond to tables, or tables that correspond to objects.

Object Relational Mapping

- **JPA** (defined in 2006) was inspired by Hibernate



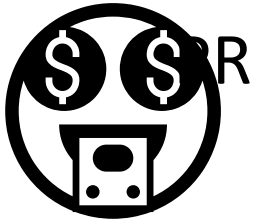
- Hibernate implements JPA

Uses Relational Databases: As a successful idea, it turned into a kind of standard for accessing relational databases without writing "**dirty**" SQL.

Java Data Objects **JDO** (2002) – is relic from OODBMS – it works with NoSQL

Object Relational Mapping

- Layer above JDBC
- No more SQL, no more ResultSet
- Improved connection 
- Caching



PRODUCTIVITY

Object Relational Mapping

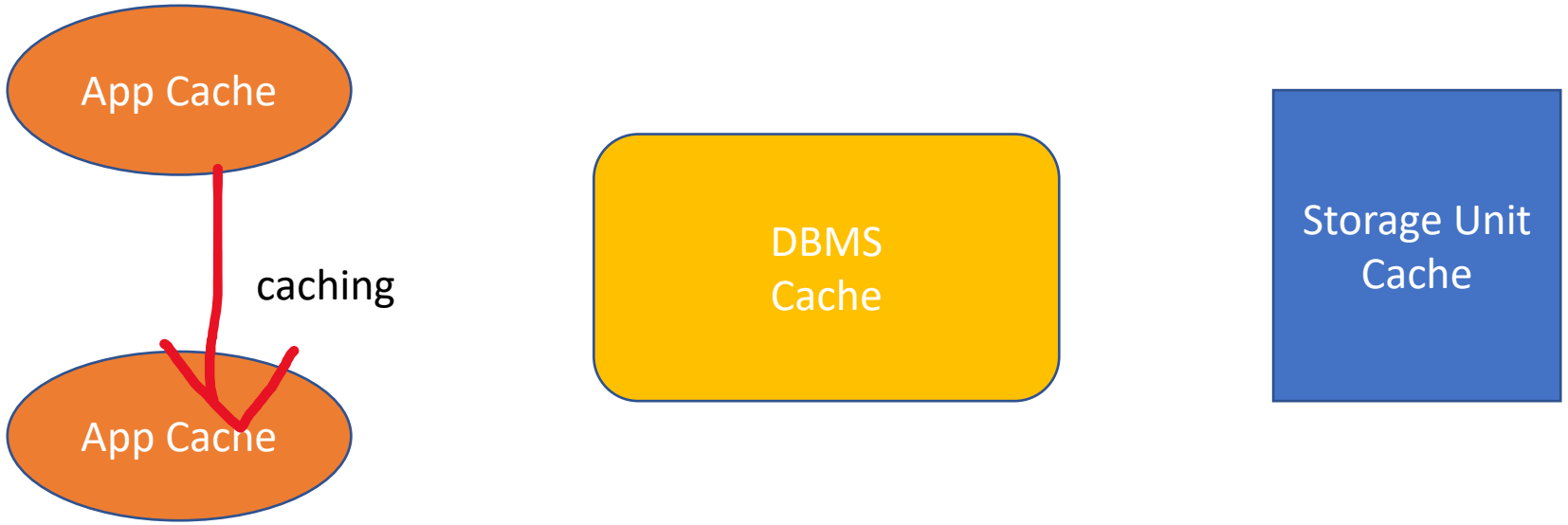
No more SQL, no more ResultSet

Instead directly get a collection of objects

But well written SQL can be very powerful and efficient

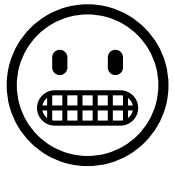
In practice, you do very efficiently in SQL operations that are very hard in Java (I'm not sure that iterating over Collections is a marked improvement over iterating over a ResultSet). In practice, people often by-pass ORM for what is critical.

Object Relational Mapping



Caching (comes from a French word meaning "to hide") can also be a problem. When you change data, you have to make sure that everybody is in synch and it can be very hard.

No more SQL, no more ResultSet



ORM with Hibernate

Object Relational Mapping - Entities

- **Entity** = lightweight persistence domain objects
- Java class that typically represents a table in a relational database, instances correspond to rows

- ORMs start with the premise "one class = one table, one object = one row" that is completely wrong. The relational logic is different.

Using ORM Entities

- Annotated with the **javax.persistence.Entity** annotation
- **public** or **protected**, no-argument constructor
- The class must not be declared **final**
- No methods or persistent instance variables must be declared **final**

ORM Entities

- Each entity must have a unique object identifier (persistent identifier)

```
@Entity
public class Student {
        @Id private int studentId;
        private String name;
        privateDate birthDate;
        public int getId() { return StudentId; }
        public void setId(int id) { this.studentId = id; }
        ...
}
```

Primary key

ORM Primary Key @id

- Simple id – single field/property

  @Id int studentId;

- Compound id – multiple fields/properties

  @Id String name;

  @Id Date birthDate;

  @Id String birthPlace;

- A key, which uniquely identifies an object is a row, can be one value or a combination of several values (each part may appear several times, but a combination appears only once)

ORM

Assumes that names in the class and names in the database match.

- If not it can be set in annotations:

```
@Entity(name = “SUSTC_STUDENTS")
public class Student{ ..... }
@Id @Column(name = “STUDENT_ID”, nullable = false)
private String studentId;
@Column(name = “FULL_NAME” nullable = true, length = 30)
private String name;
```

ORM → Relationships

@OneToOne

inheritance

Relationships describe how many rows in one table match how many rows in another table. OneToOne is typical inheritance. To use an example I have used several times, you can have a table of GeoPoints with a column that says what type of points this is, and a table of Cities with the reference to the GeoPoint plus City information. One row in one table matches one row in the other table, and only one.

ORM → Relationships

@OneToOne

@OneToMany

@ManyToOne

one-to-many and many-to-one “mirror” each other. You could for instance say that one section of a course is taught by one instructor, but that one instructor can teach many sections (example coming up)

ORM → Relationships

@OneToOne

@OneToMany

@ManyToOne

@ManyToMany

Example as a student, you take many courses and there are many students enrolled in each course.

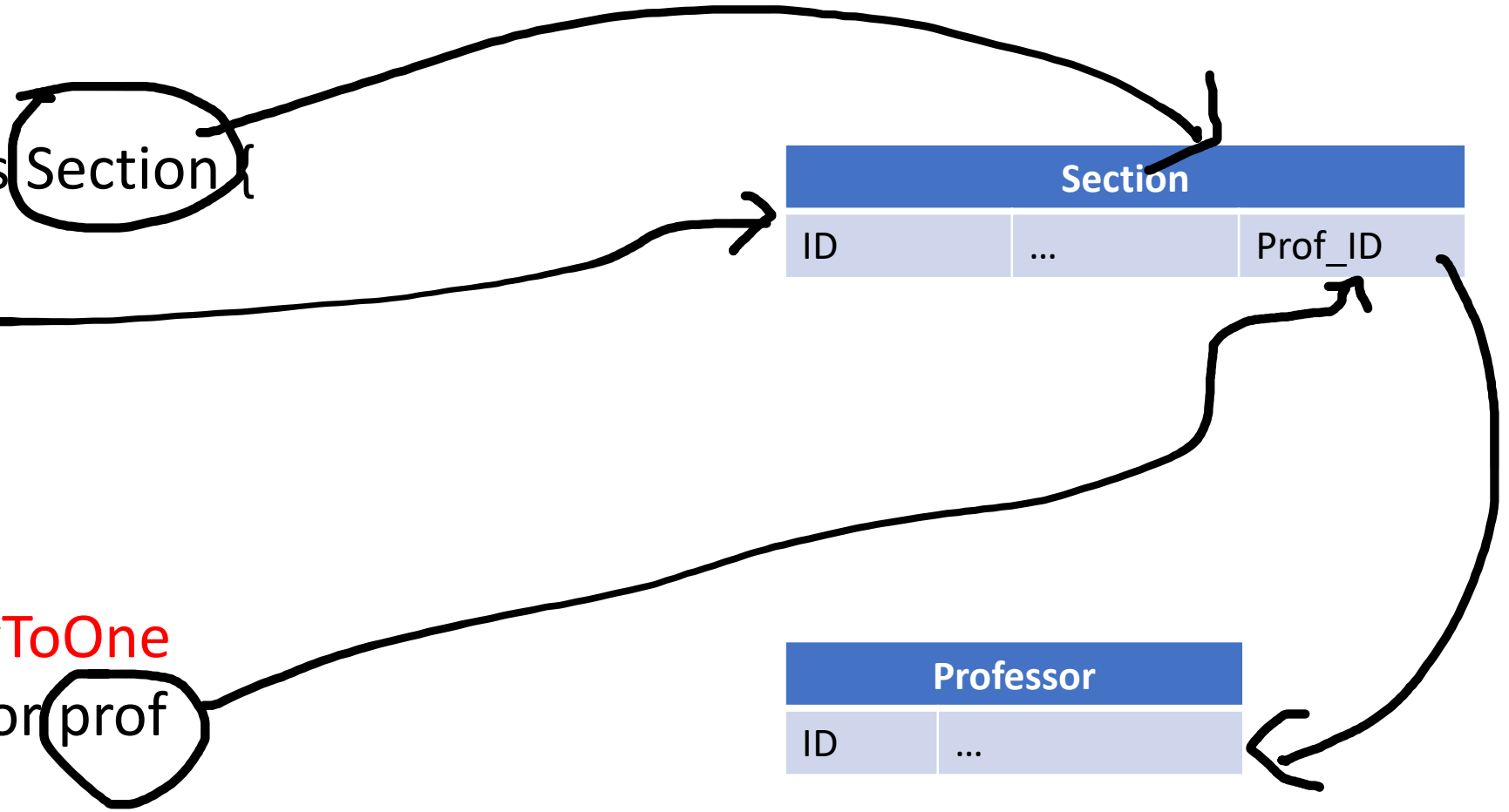
ORM Relation Attributes

```
@ManyToMany(
        cascade = {CascadeType.PERSIST, CascadeType.MERGE},
        fetch = FetchType.EAGER)
```

Relation attributes control what happens when you save and query objects related to each other. A tactic frequently used is a "lazy fetch", in which you fetch related data only when needed. Popular and bad for databases, because it generates zillions of queries to fetch what you might have returned in one operation (at the expense of using more memory)

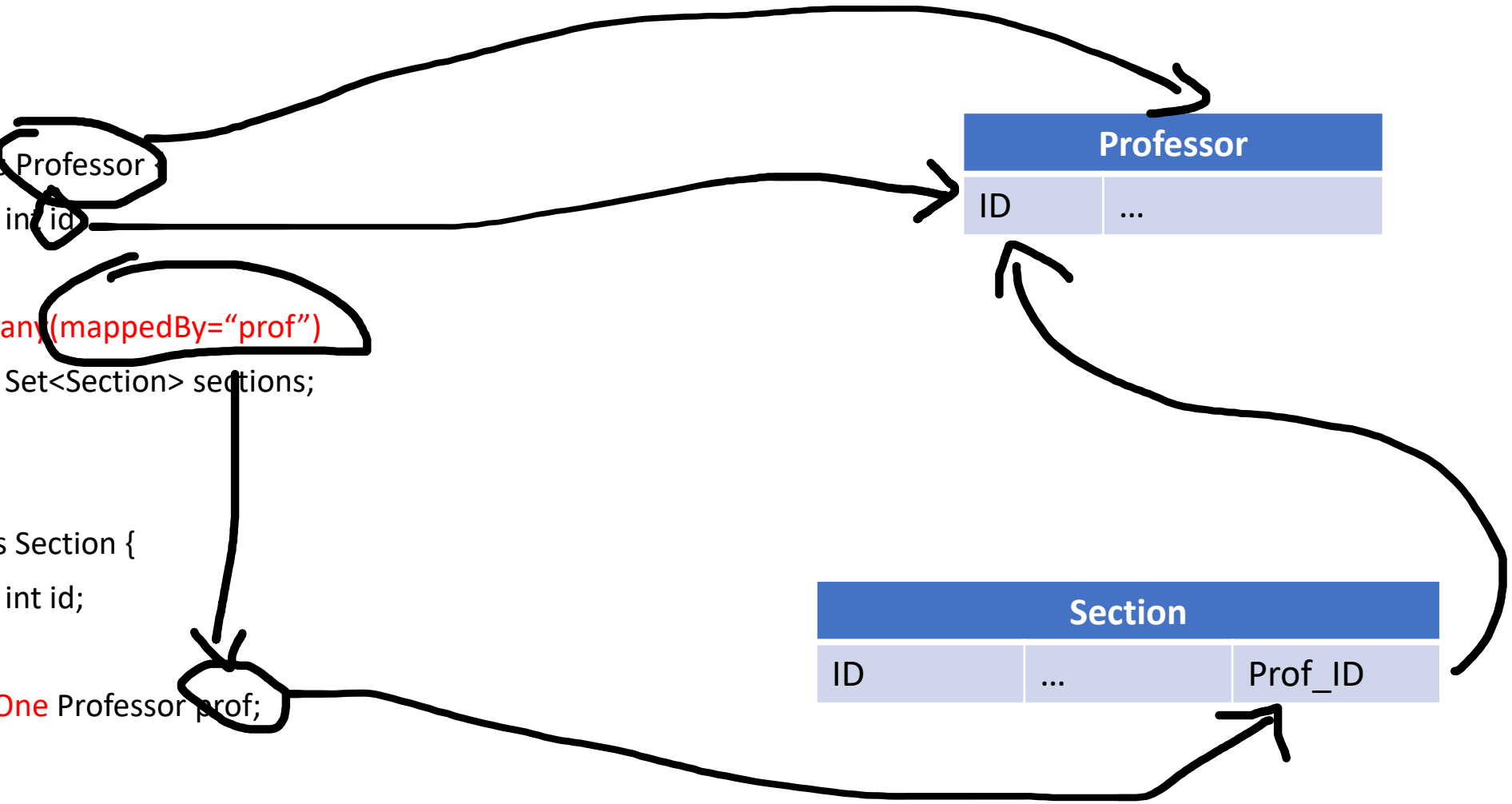
ORM Example

```
@Entity
public class Section {
    @Id
    int id;

    ...

    @ManyToOne
    Professor prof
}
```



ORM Example

```
@Entity
public class Professor {
@Id         int id;
…
@OneToMany(mappedBy="prof")
            Set<Section> sections;
}
@Entity
Public class Section {
@Id         int id;
…
@ManyToOne Professor prof;
}
```



ORM Example

```
@Entity
public class Student {
…
@JoinTable(      name = “STUDENT_SECTION”,
                 joinColumns=@JoinColumn(
                     name="STUDENT_ID",referencedColumnName="STUDENT_ID"),
                 inverseJoinColumns=@JoinColumn(
                      name="SECTION_ID", referencesColumnName="SECTION_ID")
                 Collection<Section> sections;
}

@Entity
   public class Section {
  …
  @ManyToMany(mappedBy="sections")
  Collection<Student> students;
}
```

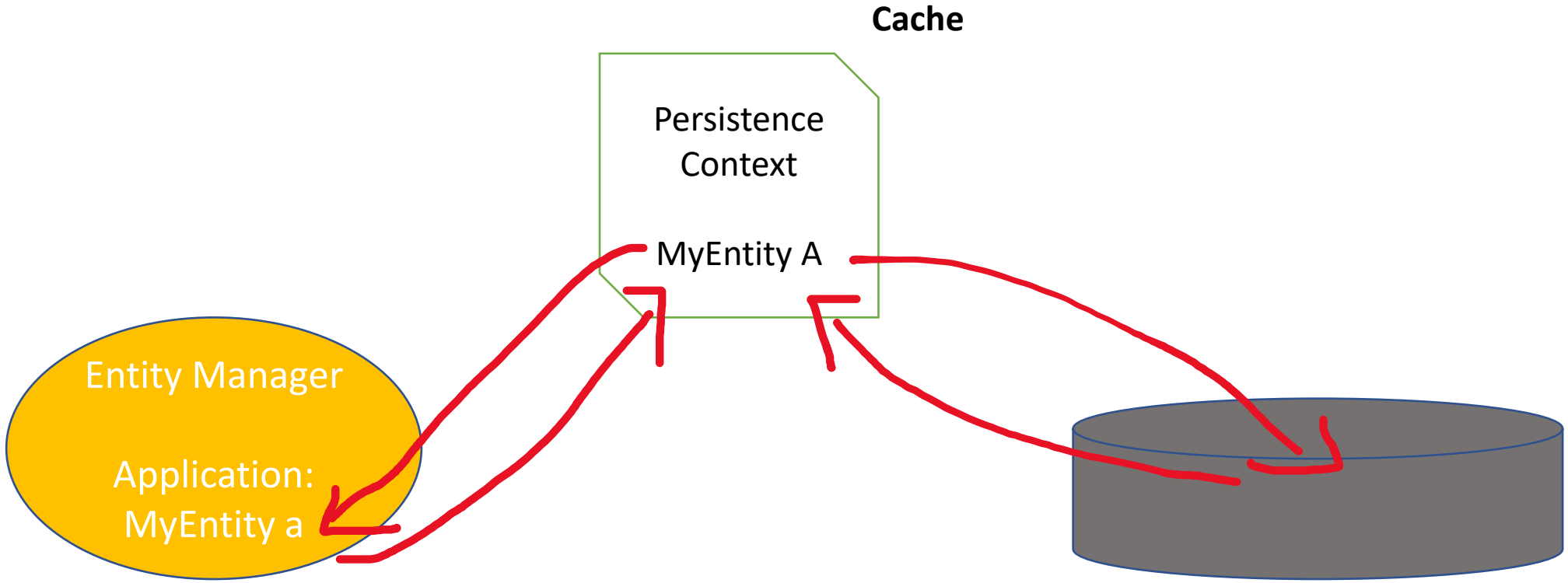
Object Relational Mapping

Inheritance and Polymorphism

The @OneToOne relationship is how to implement inheritance in a relational database, in java this results in a Java class inheriting fields and methods from a (single) superclass (inheritance) and the superclass reference can be used to refer to child classes (as in polymorphism).

ORM Caching

For all this to work you need to invoke an Entity Manager that will extend your entities and do the JDBC invisibly in the background.



ORM – Entity Manager API Operations

- **persist()-** Insert the state of an entity into the db
- **remove()-** Delete the entity state from the db
- **refresh()-** Reload the entity state from the db
- **merge()-** Synchronize the state of detached entity with the pc
- **find()-** Execute a simple PK query
- **createQuery()-** Create query instance using dynamic JP QL
- **createNamedQuery()-** Create instance for a predefined query
- **createNativeQuery()-** Create instance for an SQL quer
- **contains()-** Determine if entity is managed by pcy
- **flush()-** Force synchronization of pc to database

No SQL, no JDBC! Happy?

Object Relational Mapping

The sad truth is that as the set operations of SQL and the Object-by-Object approach of Object Oriented programming are two very different ways of solving problems, which both have their strengths and weaknesses, trying to match one to the other only works well in very simple cases. The incompability is known as "impedance mismatch’. Most hibernate developers end up having to write SQL sometimes.

Works well in simple cases. Drives people crazy in complicated cases.

Object Relational Mapping

Create, **r**ead, **u**pdate and **d**elete - CRUD

Basic assumption:

What you see is what is stored

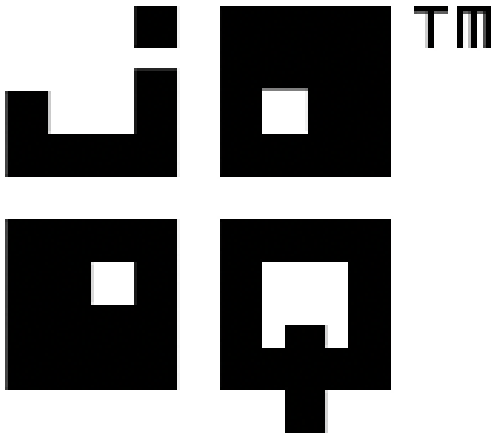
The basic ORM assumption that tables match objects is usually wrong. You need to be cleverer, usually. One way is to create "views" in the database, which are stored queries that may return something looking like your objects.

Way out: views in the database - but update may be tricky

Object Relational Mapping

Some ORM generate SQL code to create tables that match your objects. This can only generate a bad database, because objects should suit one application while the database should accommodate all cases. It's hard to design a good database. If you modify your objects, it may become difficult to change the database later; and another application may have very different requirements. ORMs can be useful, but don't expect miracles.

JOOQ



Standard SQL?

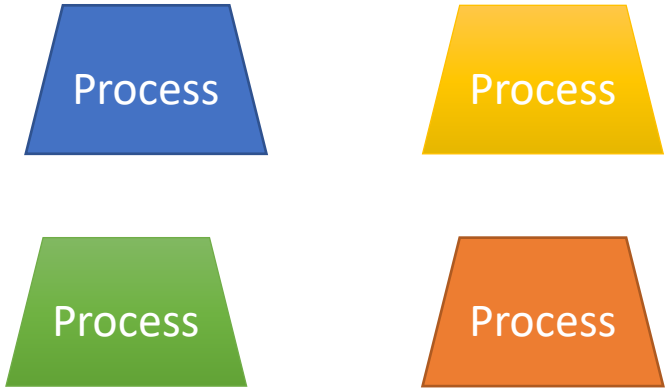
- A more recent product, JOOQ, takes the opposite approach. It is not certain that it gives better results, for the reverse reasons. Instead of having bad tables, you may get bad objects.
- "Database First”: creates classes from DBMS metadata
- They claim to generate standard SQL that is useable everywhere – but this sometimes leads to underusing a particular DBMS

Multiple Processes

Intro to networking

Processes

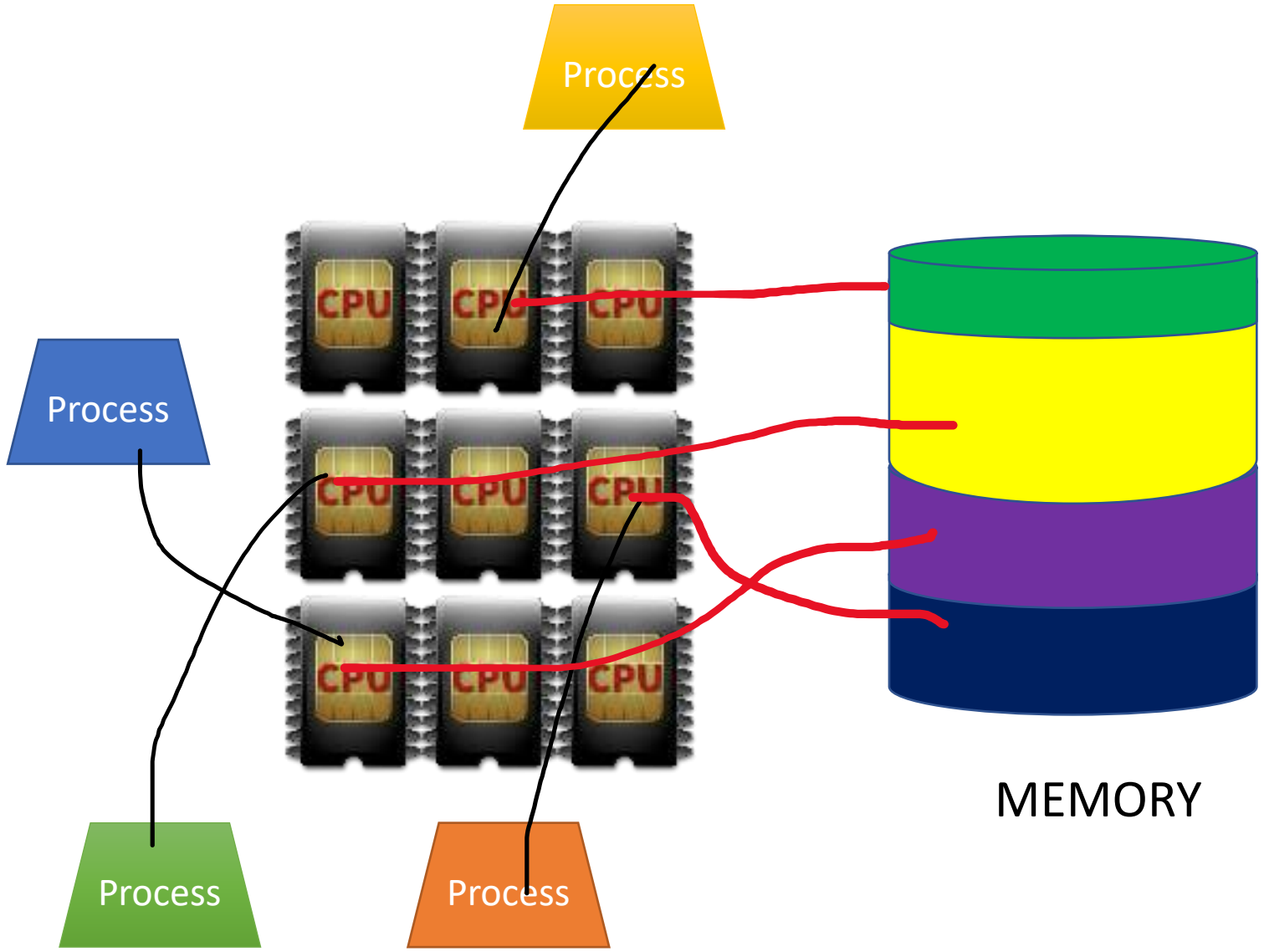
- More than one process runs at the same time.
- They might be using the same resources, each one has to take its turn…



In the 1960s researchers had the idea of simulating the simultaneous execution of programs on one processor by letting one program execute its instructions for a very short period of time or until it was needing to input/output data, whichever came first. As input/output has always been very slow compared to processor speed, then the context of the program was saved, the context of another program was restored, and this second program was running for another short period of time.

TIME SLICING

Multi-Tasking



As technology improved and computers could have several processors, then we could have programs executing instructions at exactly the same time. There is no problem as long as programs work independently.

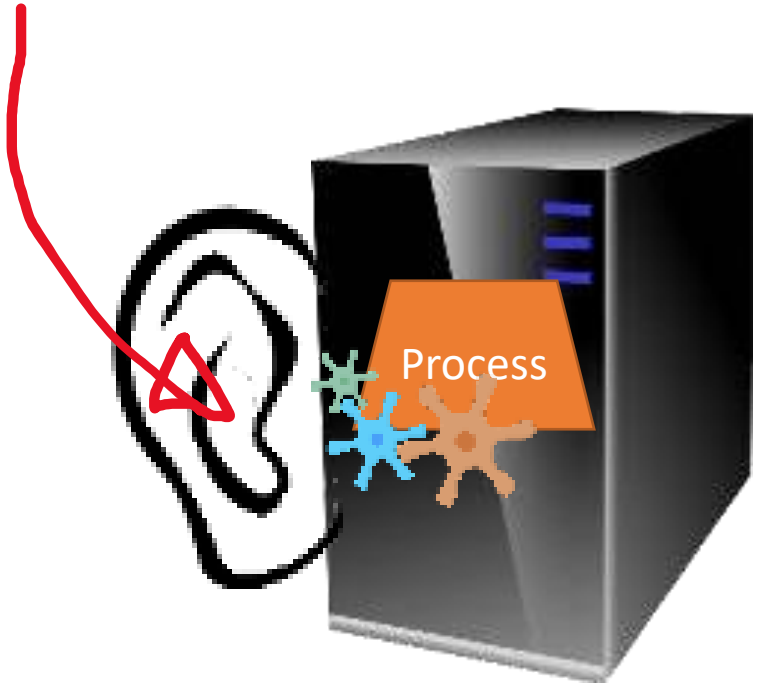
ONE PROCESS = ONE TASK

Sometimes you want to do several things at the same time.

There are many cases where you want independent tasks to be performed within the same program.

There are many cases where you want independent tasks to be performed within the same program.

Network Listener

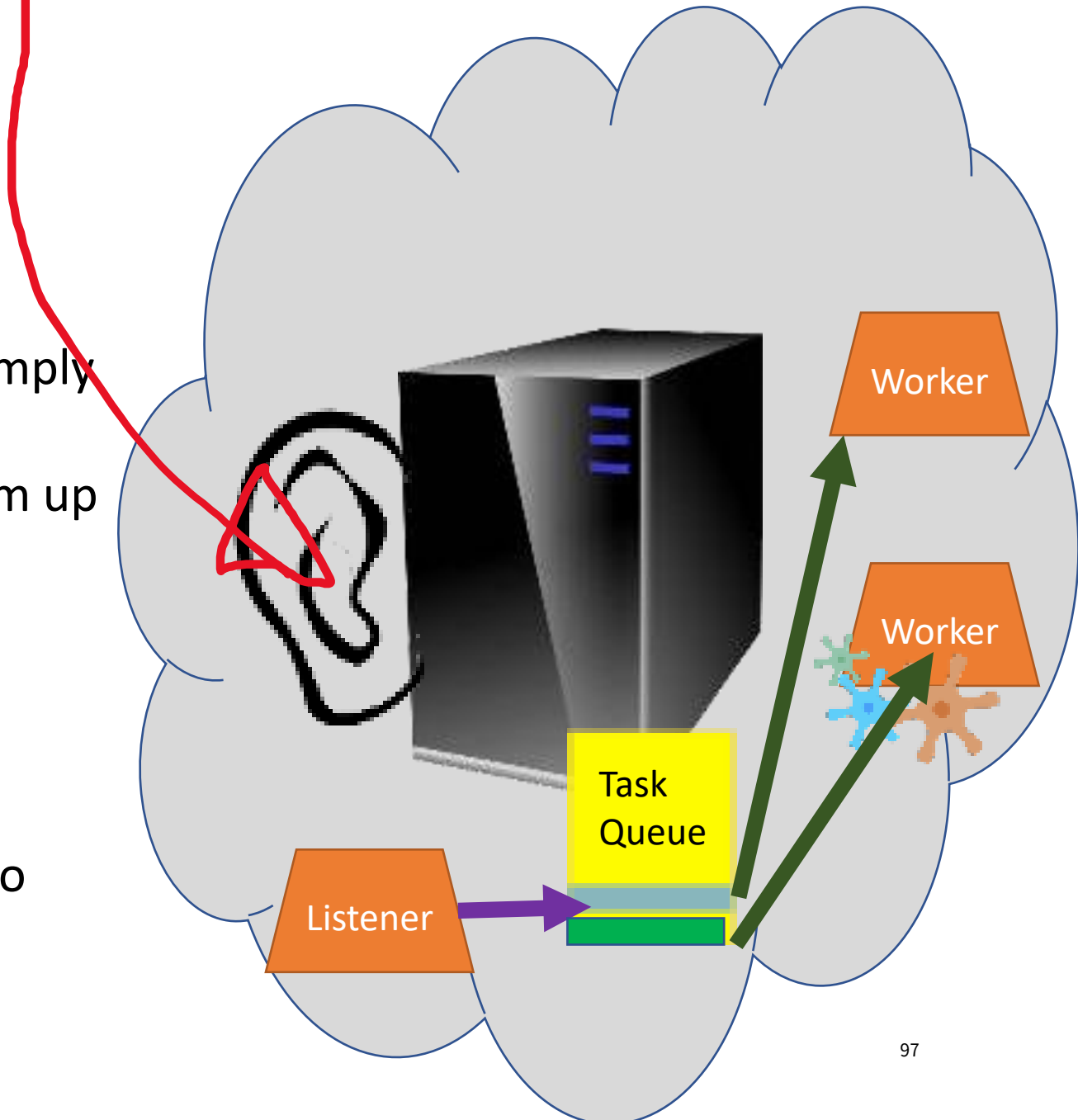


Let’s take the example of a network listener. If it processes a request (which may be something such as building a webpage from the result of a database query), it will be unable to handle requests arriving meanwhile.

These requests will probably time out if they don’t get a response from the busy listener within a short timeframe

Network Listener

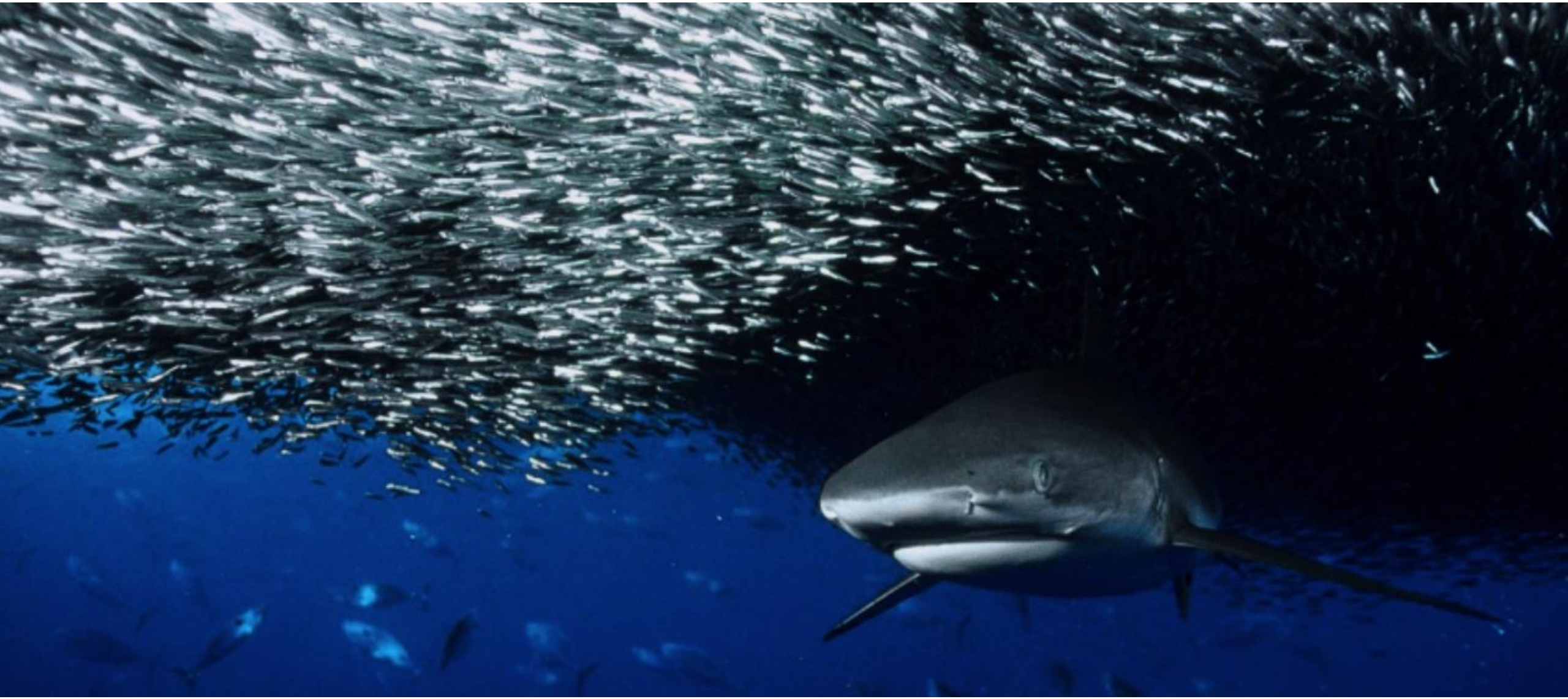
- So what is usually done is that the listener simply adds requests to the queue, where other processes running at the same time pick them up and process them.
- The listener will be able to acknowledge all requests in time, and we can have several programs processing tasks so that if one task takes a particularly long time to process, the other requests still won't have to wait for it to complete before being processed.



Another Example: Simulation

Object Oriented Programming was created (by Nygaard, with Dahl) with simulations in mind.

Kristen Nygaard
(1926-2002)



Simulating a crowd is difficult. But simulating one person (or fish, or bird) is relatively easy. If a lot of objects run at the same time, you have your crowd.

Processes working together are known as “multitasking or “multithreading”. Java was built from the start to support it easily.

Working together?

Need to coordinate

Need to communicate

Not very easy with processes if you think about it...

The system runs processes, but Java runs threads. A key difference is that a process runs a full program, and a thread is a part of a program.

Process

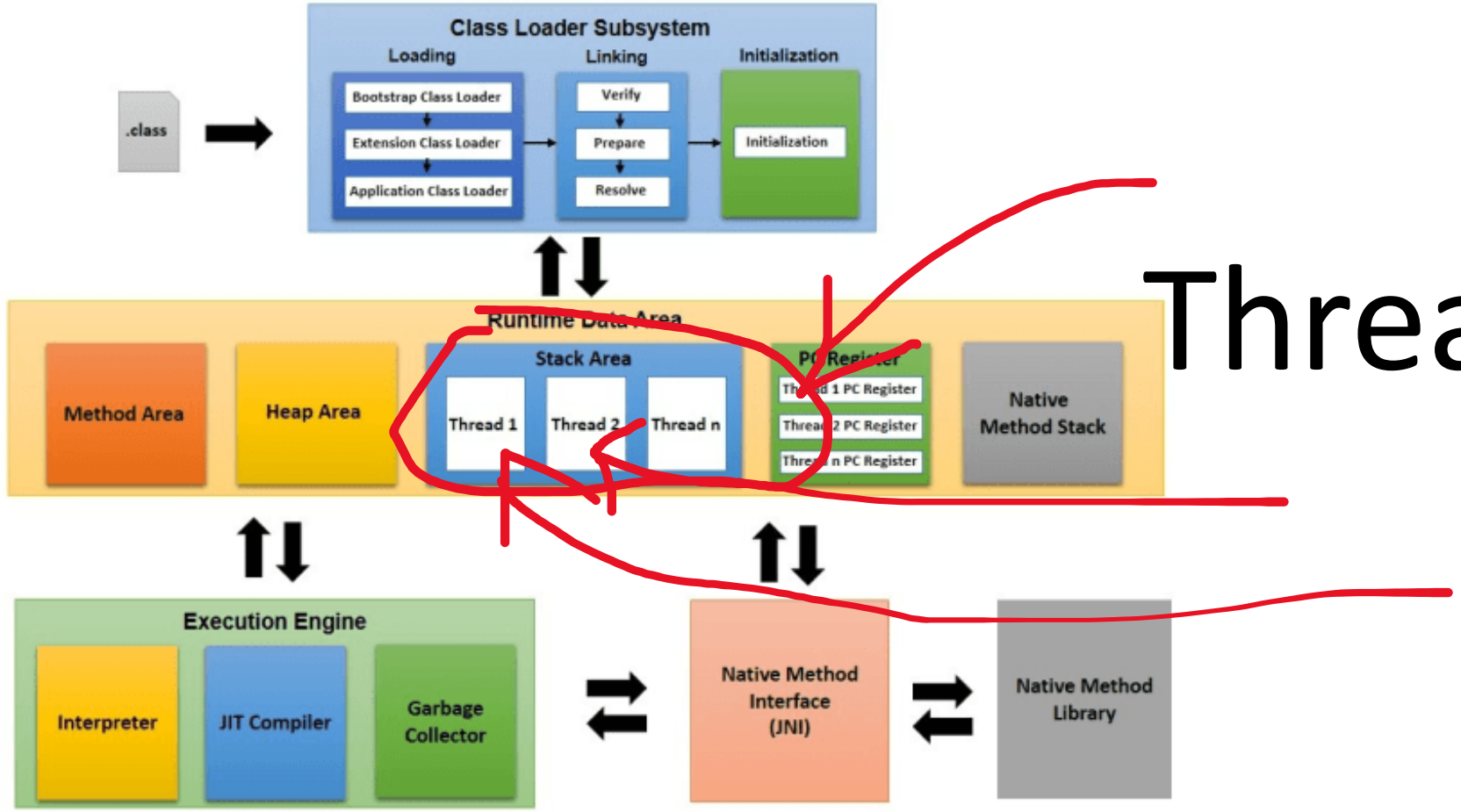
Runs a program

Thread

Runs independently in a method in a program

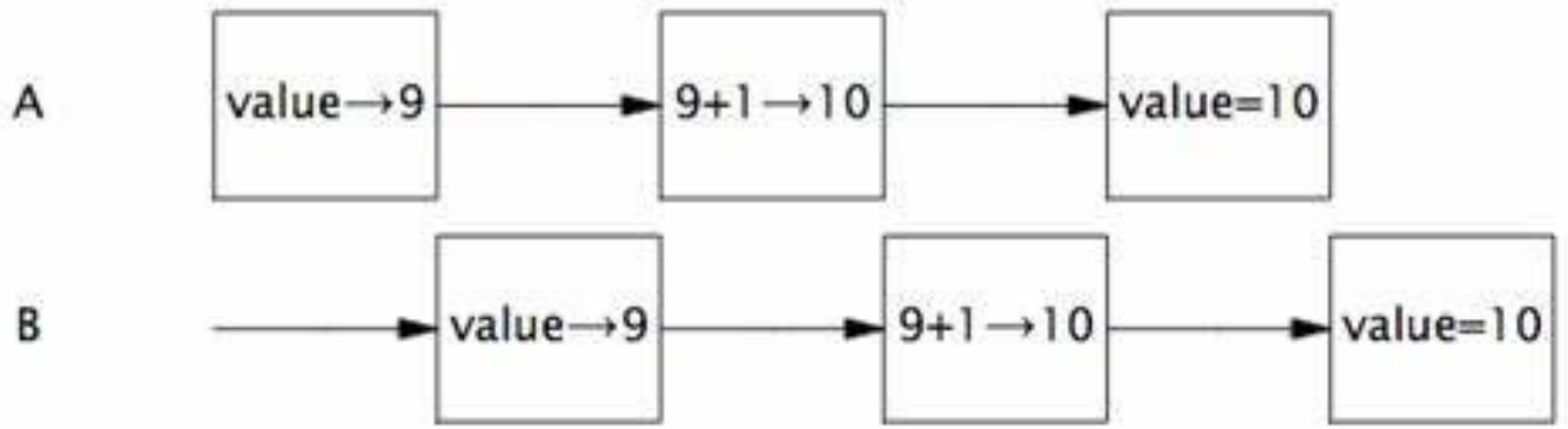
Threads are lightweight processes!

If you remember the JVM, it can support multiple threads. Not three or four like the diagram could suggest: it can support thousands of threads.



Java Threads

Week 9 - Presentation 3



Multi-tasking

Not very easy with processes if you think about it...

Need to coordinate

Need to communicate

Threads are lightweight processes!

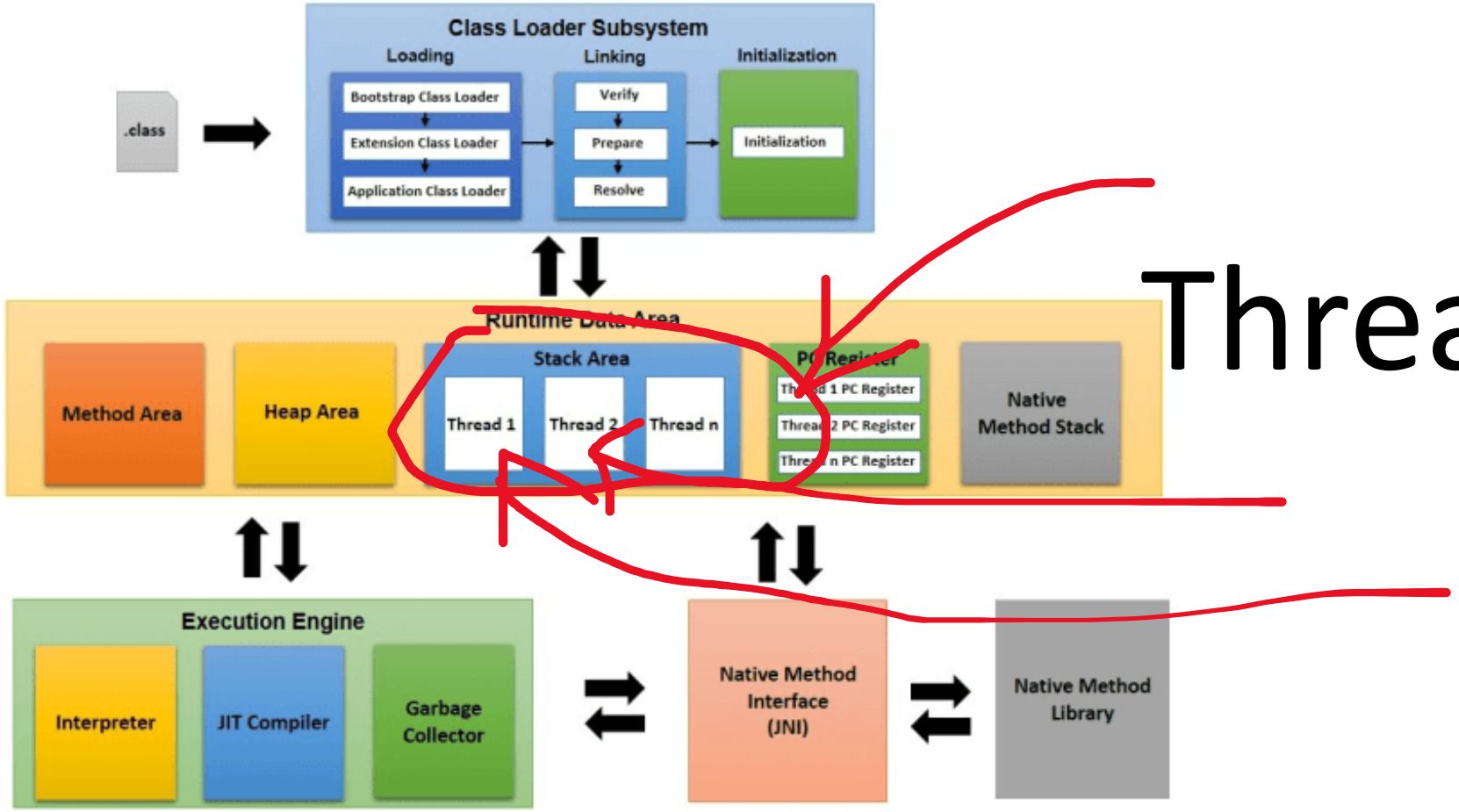
Process

Runs a program

Thread

Runs independently in
a method in a program

If you remember the JVM, it can support multiple threads. Not three or four like the diagram could suggest: it can support thousands of threads.



The Thread Class

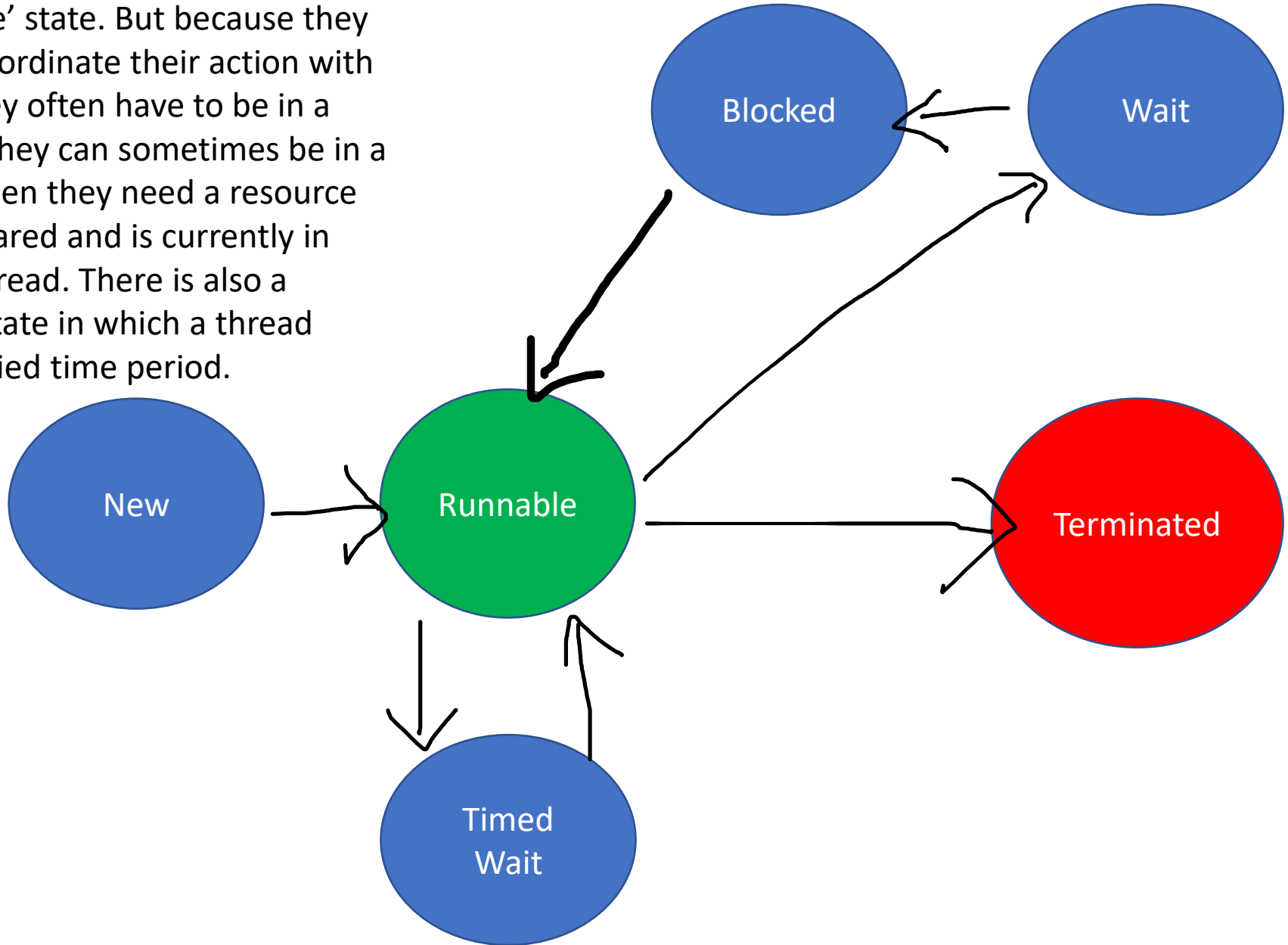
java.util.concurrent

```
class TaskToRun implements Runnable {

    public void run {


    }

}

...

    Thread t1 = new Thread(new TaskToRun( ... ));

    t1.start();

...
```

Calls .run() method

- Threads are java objects that can call the run() method of an object that implements Runnable.
- Once “run” the object works independently as a separate procedure.

Threads can be in multiple states. When you create them, they are in a ‘New’ state. When you call their start() method, they get into the ‘Runnable’ state. But because they usually have to coordinate their action with other threads, they often have to be in a ‘Wait’ state, and they can sometimes be in a ‘Blocked’ state when they need a resource that cannot be shared and is currently in use by another thread. There is also a “timed waiting” state in which a thread sleeps for a specified time period.



Thread States

- New
- Runnable
- Blocked
- Waiting
- Time waiting
- Terminated

- We will see a possible sequence of passing through the states in the server example from the previous presentation

There are many cases where you want independent tasks to be performed within the same program.

Network Listener (server)

Let’s take the example of a network listener. If it processes a request (which may be something such as building a webpage from the result of a database query), it will be unable to handle requests arriving meanwhile.

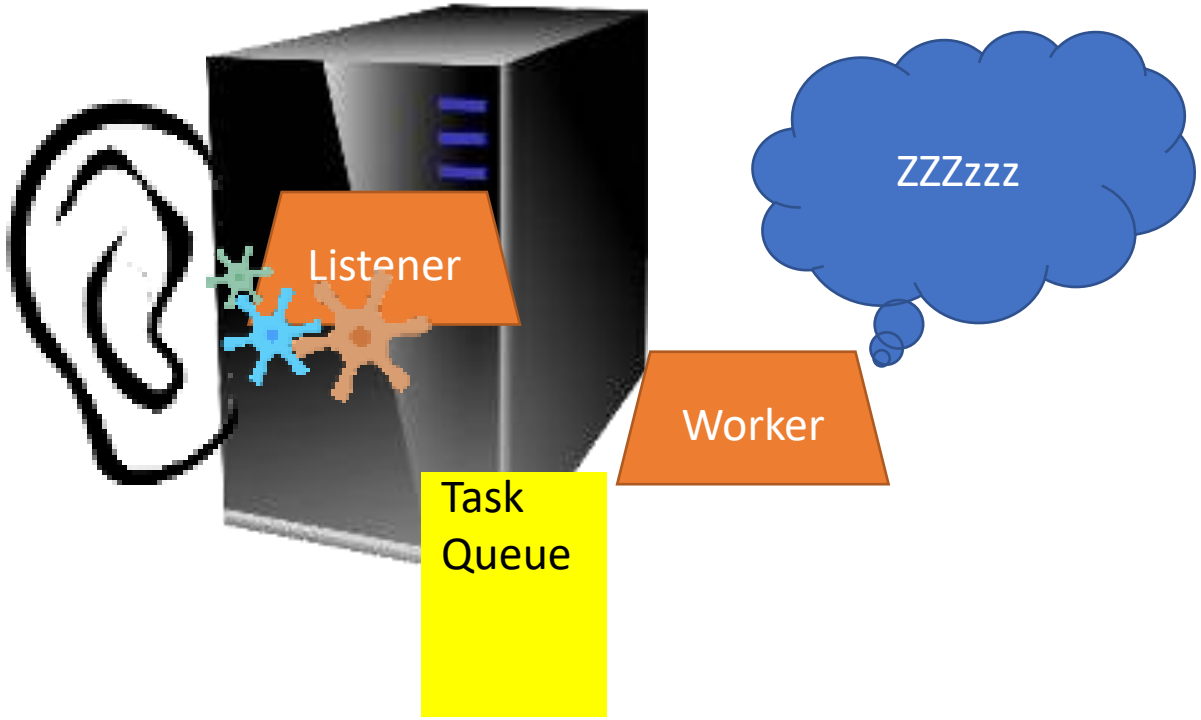
These requests will probably time out if they don’t get a response from the busy listener within a short timeframe

Listener and worker have nothing to do until a request arrives. They basically have two ways of waiting, calling a wait() method that makes them wait until something happens, or sleep for a fixed amount of time.

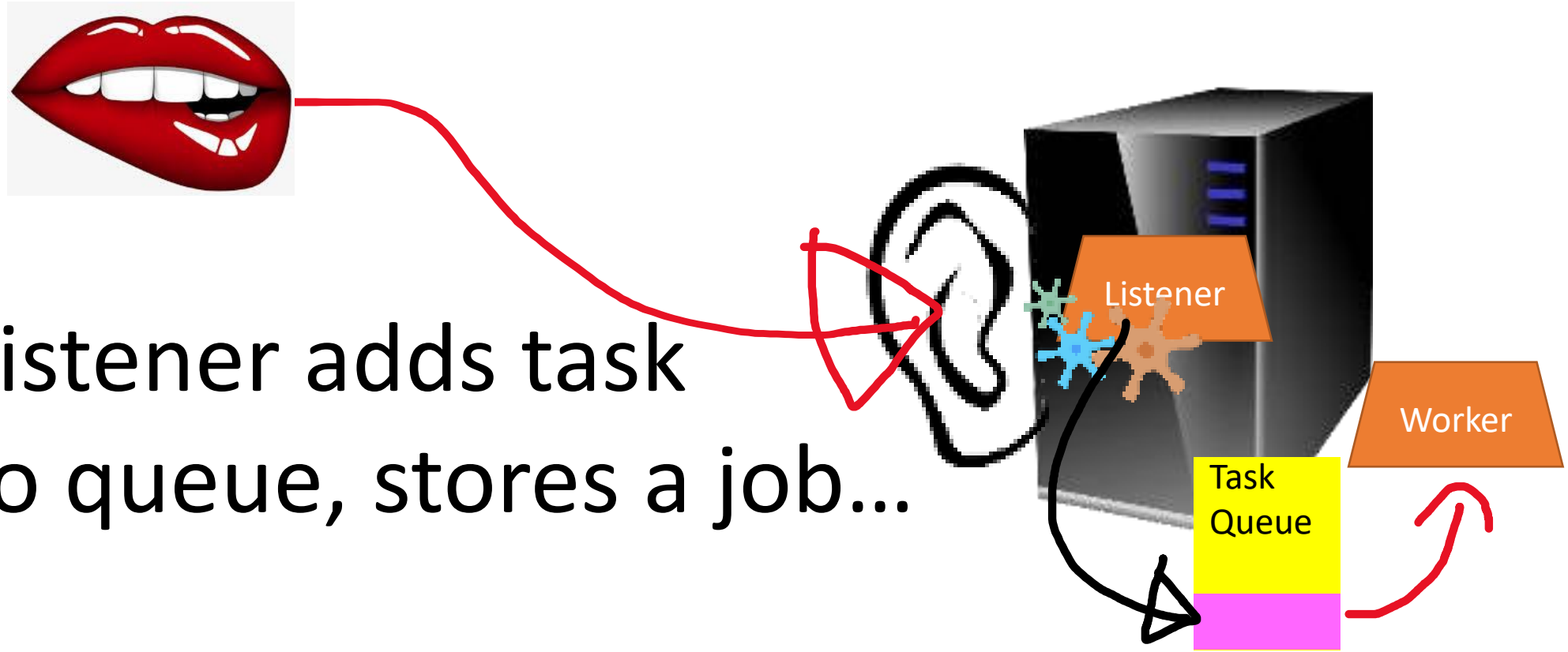
TimedWaiting

Runnable

Waiting



When the listener has received and stored something, then the worker thread must get to work.



Coordination patterns

Listener

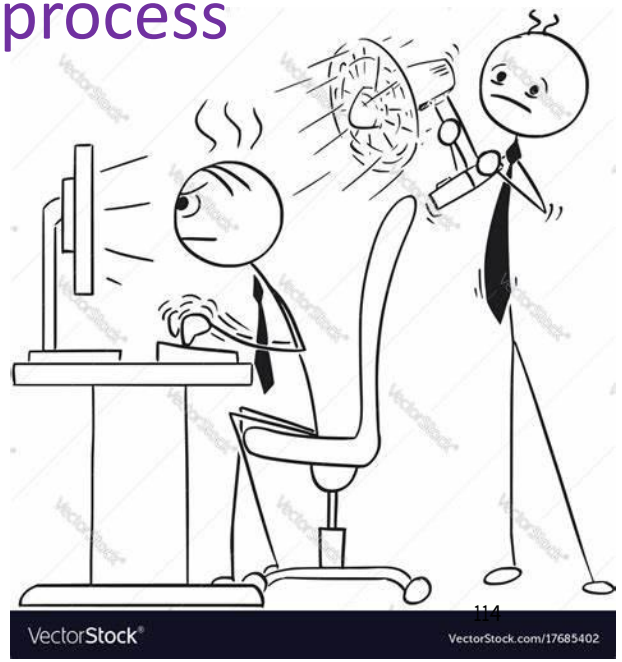
call worker.notify()

Worker

Calls listener.wait()

// do process

…



There are a few different ways for coordinating the processes.

- One is to make the Worker call the wait() method of the Listener. Then the worker will be made to wake up when the Listener calls its notify() method.

Coordination patterns

Listener

```
worker.interrupt();
```

Worker

```
Thread.sleep(3600000);

catch (InterruptedException e) {
        // Do Process
}
while (this.interrupted()) {
        // Do Process
}
```

There are a few different ways for coordinating the processes.

- One is to make the Worker call the wait() method of the Listener. Then the worker will be made to wake up when the Listener calls its notify() method.
- Or the worker can sleep (timed waiting) until the listener can throw a special exception to the worker thread and wake it up (interrupt). Because several interruptions could be generated, the “interrupted” state should be checked in a loop.

Coordination patterns

Listener

There are a few different ways for coordinating the processes.

- One is to make the Worker call the wait() method of the Listener. Then the worker will be made to wake up when the Listener calls its notify() method.
- Or the worker can sleep (timed waiting) until the listener can throw a special exception to the worker thread and wake it up (interrupt). Because several interruptions could be generated, the “interrupted” state should be checked in a loop.
- Or the Worker may ignore the Listener, check by itself, and keep on checking until it’s interrupted – which means no more work to expect.
- This is called “Polling”.

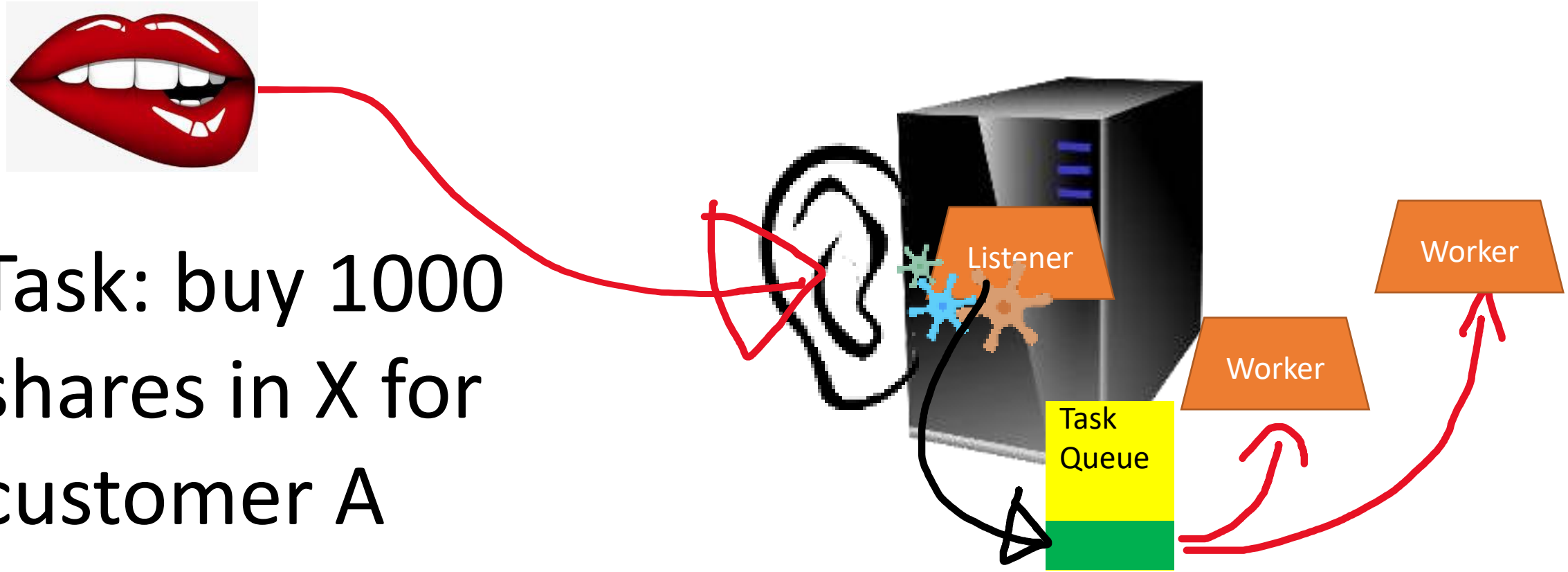
Worker

```
while (true) {
    try {
        Thread.sleep(1000);
        if ( /*something to do */ )
            // Just Do It
        }
    } catch (InterruptedException e) {
        break;
    }

}
```

`.interrupt()`

- This method is used to get the attention of another thread
- Very useful! You can interrupt a thread that works for far to long
- Usually the way exceptions work is that some method calls another one and then “throw” sends an exception to the caller.
- Here it is an exception that is “thrown” to a different thread - it comes from the outside (not from the call hierarchy)

However, coordination is a bit more complicated: we have a shared resource, which is the queue. You want every task to be processed once, not several times by Workers ignorant of each other.



Race Conditions

If we are not careful two or more workers may check the queue at the same time, see the same task and process it. This is called a race condition.

$ javac NaïveQueue
$ java NaiveQueue

```
T-3 processing task-1
T-2 processing task-1
T-4 processing task-10
T-5 processing task-2
T-1 processing task-3
T-4 processing task-4
T-3 processing task-6
T-5 processing task-5
T-2 processing task-4
T-1 processing task-7
T-4 processing task-8
T-2 processing task-9
T-3 processing task-9
T-5 processing task-8
```

Synchroniz

It works!

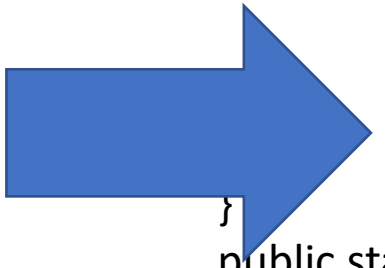
```
static PriorityQueue<String> tasks = new PriorityQueue<String>();


public static synchronized String checkTask() {
    String task = null;

    try {
        task = tasks.remove();
    } catch (Exception e) {
        task = null;
    }
    return task;
}
```

```
T-3 processing task-1
T-1 processing task-4
T-4 processing task-10
T-2 processing task-3
T-5 processing task-2
T-5 processing task-5
T-1 processing task-9
T-3 processing task-8
T-2 processing task-7
T-4 processing task-6
```

```
public static void main (String [] args) {
    for (int i = 1; i<11; i++) {
        tasks.add("task-" + i);
    }

    int numT = Integer.valueOf(args[0]);
    for (int i = 1; i <= numT ; i++) {
        Thread t = new Thread(new Worker ("T-"+i));
        t.start();
    }


}
```

```
    static class Worker implements Runnable{

        String myName;
        Random rand;
        public Worker  (String name) {
            myName = name;
            rand = new Random(28);
        }

        public void run () {

            String task;
            while (true) {
                try {
                    // sleep for 1/2 a second
                    Thread.sleep(500);
                    task = Synchro.checkTask();
                    if(task != null) {
                        System.out.println(myName + " processing " + task);
                        Thread.sleep(rand.nextInt(1000));
                    }

                } catch (InterruptedException e) {
                    break;
                }
            }
            System.err.println(myName + " stopping");
        }
    }
```

```
public class CriticalSectionDemo{
        private int i = 0;

        public int incrementValue() {
                System.out.println("Current Thread " +
Thread.currentThread().getName());
                try {
                        Thread.sleep(300);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
                return this.i++;
        }
        public static void main(String[] args) {
                CriticalSectionDemo demo = new CriticalSectionDemo();
                new Thread(() -> demo.incrementValue()).start();
                new Thread(() -> demo.incrementValue()).start();
                try {
                        Thread.sleep(1000);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
                System.out.println(demo.i);
        }
}
```



```
private int i = 0;

        public int incrementValue() {
                System.out.println("Current Thread " + Thread.currentThread().getName());
                try {
                        Thread.sleep(300);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
                return this.i++;
        }
```

Critical section of the code where the sequence of execution by different threads will change program behaviour

```
public synchronized int incrementValue() {
                System.out.println("Current Thread " + Thread.currentThread().getName());
                try {
                        Thread.sleep(300);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
                return this.i++;
        }
```

```
public int incrementValue() {
        System.out.println("Current Thread " + Thread.currentThread().getName());
        try {
                Thread.sleep(300);
        } catch (InterruptedException e) {
                e.printStackTrace();
        }
        synchronized (this) {
                return this.i++;
        }
}
```

The first arrived must **block** the others

`Synchronized`

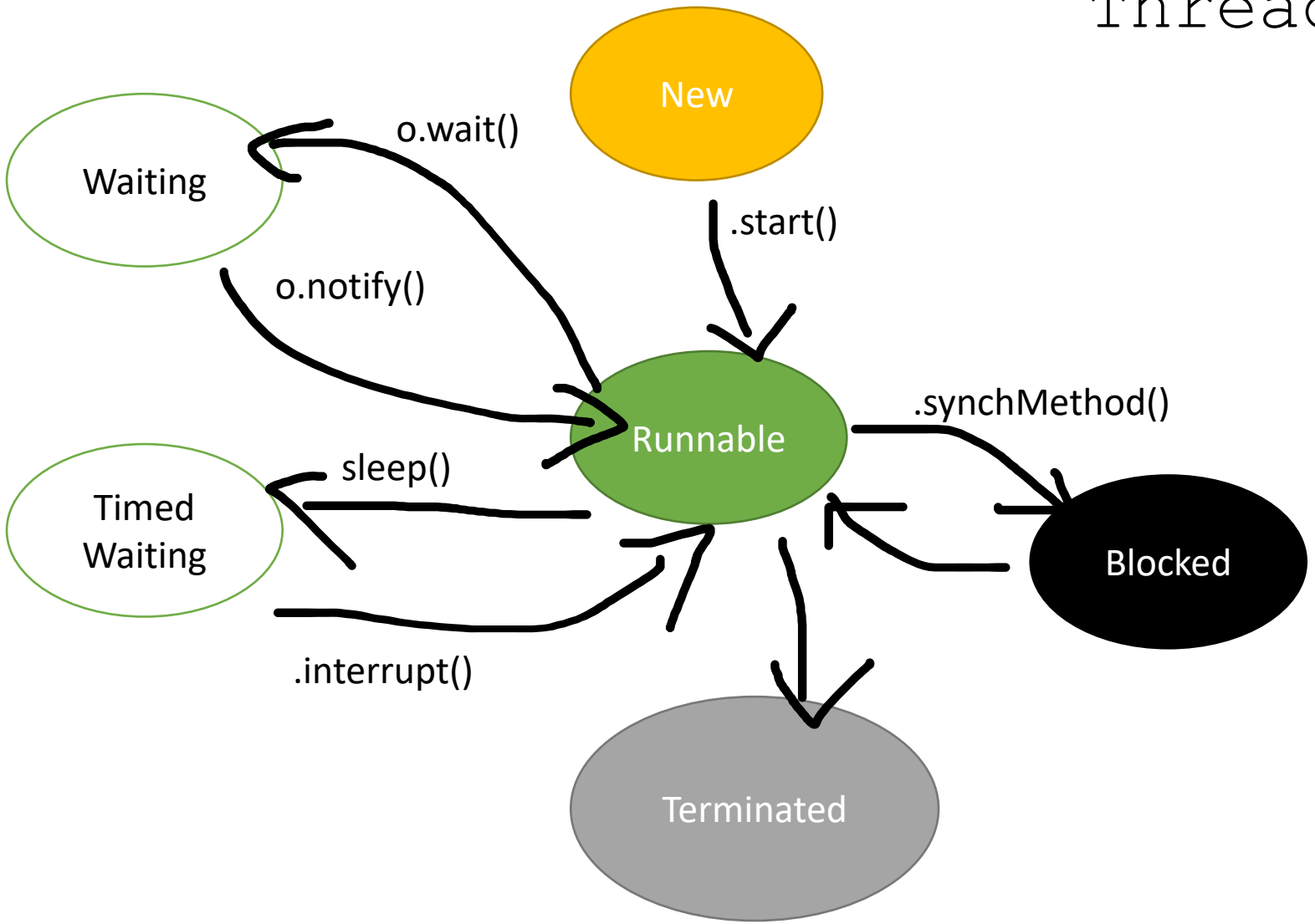
If we want to avoid this we want to make sure that on only one Worker reads from the queue at the same time (and in fact we should check that it’s not reading at exactly the same time as the Listener is writing, which could be messy too). In Java there is a mechanism called synchronization that guarantees that only one thread can access a synchronized resource at the same time.

The first arrived must **block** the others

synchronized *type* methodName(...) { }

You just declare the method to be synchronized. It means that the first thread to execute it will lock it. Other threads will block, and (each in turn) will be able to execute the method when the first thread returns from it. Needless to say, it's a bad idea to synchronize methods that could take hours to run. You should only synchronize pieces of code that should run fast.

Thread State Summary



Built-in Support for Concurrency

- You can also use synchronized collections in which the main methods (e.g. remove) are synchronized already. A similar concept to the observable lists we used with JavaFX.
- Java has built-in mechanisims for collections
  - Special wrappers designed for concurrency which you can call from the Collections class:
    - EG: List list = Collections.synchronizedList(new ArrayList());
  - OR you can use purpose built collections:
    - EG a FIFO queue: java.util.concurrent.ArrayBlockingQueue<E>

Built-in Support for Concurrency

- Using a synchronized collection means that YOU don’t need to write a synchronized method. There are however still plenty of cases where you might want to – think of a counter of tasks completed increased by one by every Worker that is done with a task ...
- You may also have synchronization or not with Strings, using either a StringBuilder (not synchronized) or a StringBuffer (synchronized) object. Avoid a synchronized version (slower) when not needed.

Thread 

WARNING : Deadlocks

- The fact that a thread can be blocked by another one is good, as long as one of the threads can, after a more or less long time, complete what it has to do, then release the locks and unblock any other thread waiting. But you can have problems, because of multiple locks that can be held at the same moment by different thread.
- Because locks are associated with objects, **different objects** can call the same synchronized method at the same time!
- There would be no problem with a synchronized static method, because then the (single) lock class would be held by only one thread. But when the methods are attached to different objects, the code can be executed at the same time and the locks held by two distinct threads.

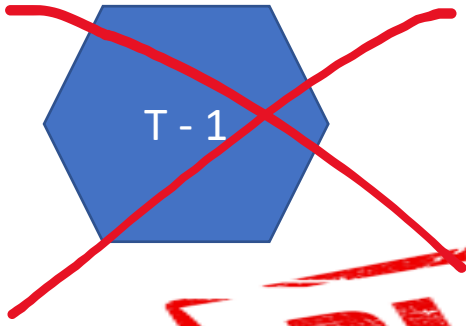
Deadlocks

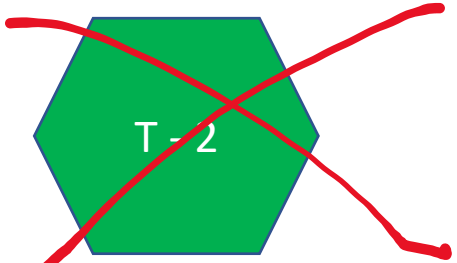
Gets the lock of o1

Gets the lock of o2

```
call synchronized o1.m1() {
   call synchronized o1.m2();
}
```

```
call synchronized o2.m1() {
   call synchronized o2.m2();
}
```





Example: Two people try to transfer money from their account to the other's account at the same time…

```
class Account {
    private int accountId;
    private float balance;

    public Account(int id, float amount) {
        accountId = id;
        balance = amount;
    }

    public synchronized void credit(float amount) {
        System.out.println("Thread "
                + Thread.currentThread().getName()
                + " crediting account #"
                + Integer.toString(accountId)
                + " of " + Float.toString(amount));
            balance += amount;
    }

}
```

I’m synchronizing every sensitive method.

The bank doesn’t want to see an account credited twice.

```
    }

    public synchronized boolean debit(float amount) {

        System.out.println("Thread "
        + Thread.currentThread().getName()
        + " debiting account #"
        + Integer.toString(accountId)
        + " of " + Float.toString(amount));

        if (amount < balance) {
            balance -= amount;
            return true;
        } else {
            return false;
        }
    }

}
```

I’m synchronizing every sensitive method.

The bank also doesn’t want to see an account debited twice.

When I’m transferring money from the current account to another account, I’m checking I still have enough on the account, then subtracting the amount from the current balance before crediting the other account with the amount. As I’m already in a synchronized method, there is no need to call this.debit(amount).

```
public synchronized void transfer(Account toAccount,float amount) {
    if (balance >= amount) {
        balance -= amount;
        toAccount.credit(amount);
    }
}
```

Here is my basic money transfer task which is run in a thread.

```
class MoneyTransfer implements Runnable {
    private Account fromAccount;
    private Account toAccount;
    private float moneyToTransfer;

    public MoneyTransfer(Account fromAcnt, Account toAcnt, float amount) {
        fromAccount = fromAcnt;
        toAccount = toAcnt;
        moneyToTransfer = amount;
    }

    public void run() {
        fromAccount.transfer(toAccount, moneyToTransfer);
    }
}
```

```
public class BankingTransaction {
    private final static Account account1 = new Account(1, 1000);
    private final static Account account2 = new Account(2, 500);

    public static void main(String[] args) {
        Thread t1 = new Thread(new MoneyTransfer(account1, account2, (float)500));
        Thread t2 = new Thread(new MoneyTransfer(account2, account1, (float)250));
        t1.start();
        t2.start();
    }
}
```

Now if you run this program, sometimes it will work fine, sometimes it will remain stuck. Random problem, and deadlocks can be hard to debug.

You normally avoid deadlocks by always locking resources in the same order. Here it's a bit different. You might imagine having a third thread watching and interrupting a stuck task ...

.isAlive() and .isJoin()

- Check for thread termination:

  thr.isAlive()

- Wait for thread termination:

  thr.join()

Sometimes the main thread (that started all the other ones running) must gather work done by the others and kind of finalize operations. It can check if a thread is still running, and if this is the case call its .join() method that blocks until the thread has terminated.

```
for (int i = 0; i < threadCount; i++) {
    thr = new Thread(...):
    threadList.add(thr);
    thr.start();

    Iterator iter = threadList.iterator();

    while (iter.hasNext()) {
         thr = iter.next();
    if (thr.isAlive()) {
         thr.join();
    }
}
```

// Now use what threads have done (combining their work)

Here is a simple example of a master thread starting children threads and waiting for the termination of each of them.

starvation

You can set thread priority but it is something to be done with caution.

If a thread is starved of CPU time it is not locked but it also cannot make progress.

Safe Threads

- It is better to use immutable objects
  - All attributes private and final
  - No setters
  - class final (means it cannot be extended)

- To avoid problems it is warmly recommended that you use immutable objects in threads

Client Server Intro

Network Programming

In Java (as in C) network communications are based on a "socket". In Java it's a class, and you use it like a file.

Client Socket ← → Server Socket

Stream = like a file

- java.net.*
- class Socket
- class ServerSocket

Class Socket

Talk to one server

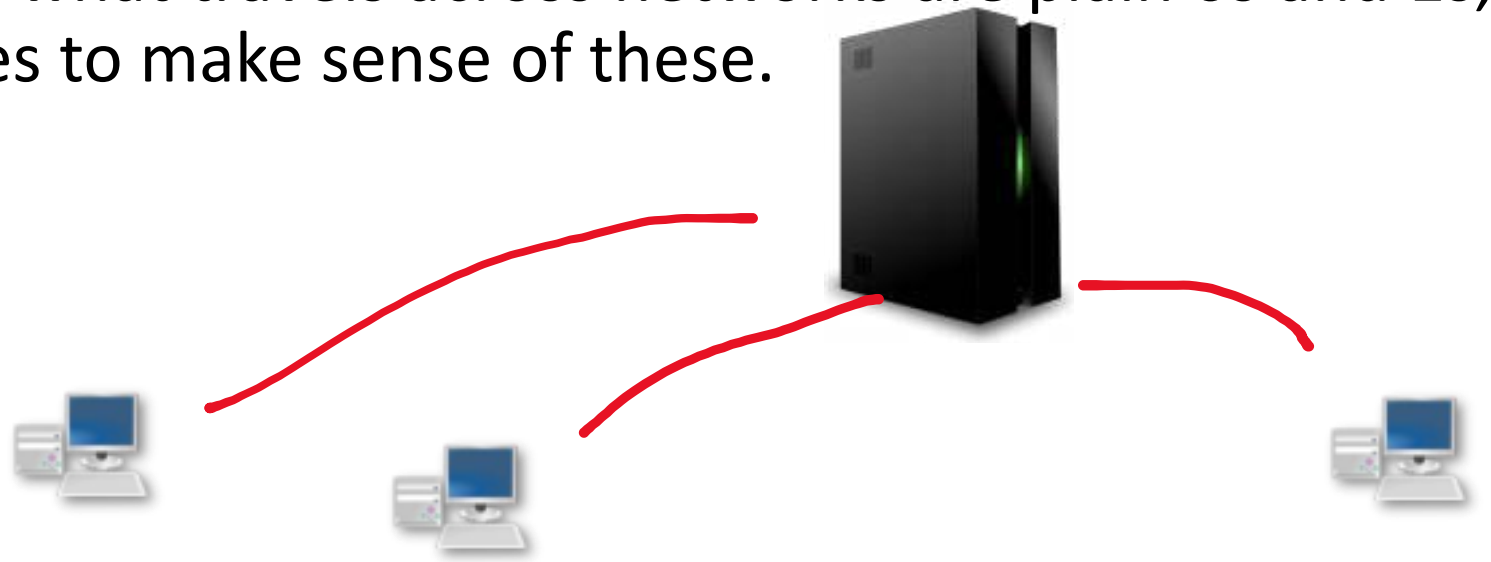
Class Server Socket

Talk to multiple servers – or used to serve multiple clients

Network Programming

Computers that talk to each other are one of the important features of the world we live in ...

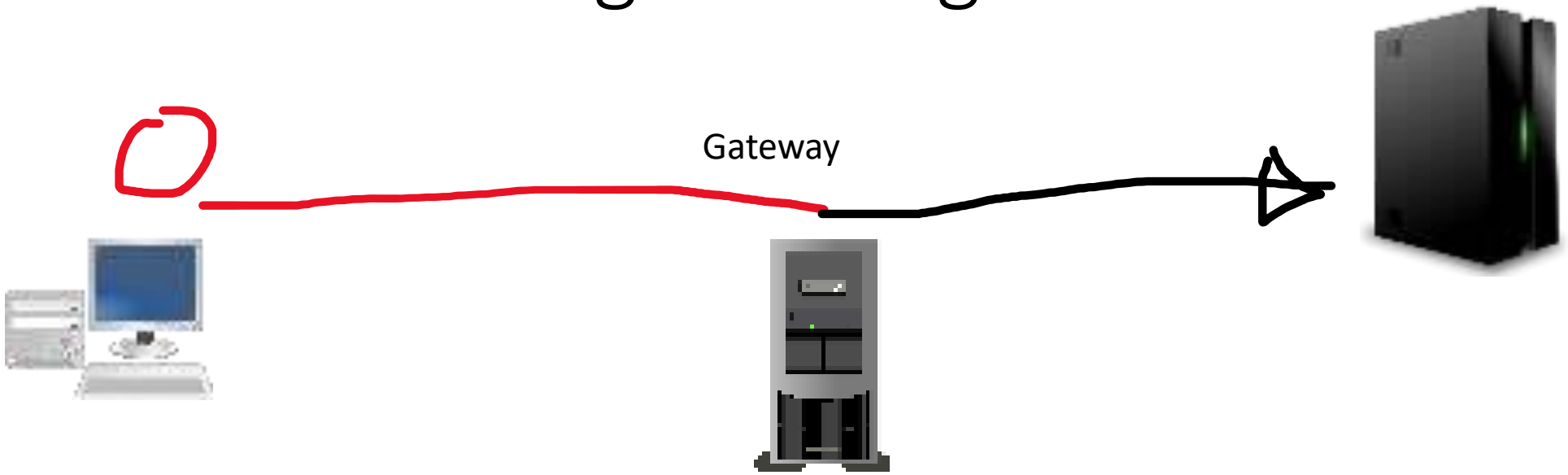
But as what travels across networks are plain 0s and 1s, we need a lot of rules to make sense of these.



Network Programming

Sending a message between computers is very much like sending a letter, say to the White House. You may drop it in a China Post mailbox, but it will not be delivered by China Post. China Post will take it by plane to the US, perhaps to Los Angeles, where the letter will be handled to another network (USPS, United States Postal Services). Additionally, none of them cares about what you wrote.

Network Programming



It's very much the same with computers. Your machine is on a network, and what you send must reach a special computer called a gateway that has two network cards connected to different networks and will send your "message" to another network (and possibly many gateways) until it reaches the target computer.

Rules = PROTOCOLS

In the same way that there are rules for sending a letter (address on the front, sender address on the back, country at the bottom when sending abroad, stamps...) there are rules for sending messages on a computer network. The word used isn't "rules" but "protocols", which basically means the same ("behavior rules").

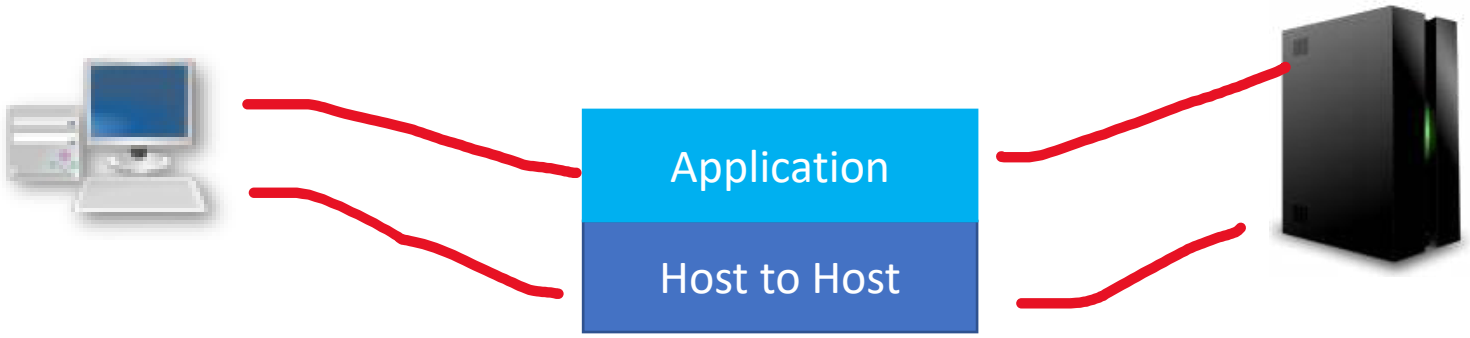
Rules = PROTOCOLS

In the same way that there are rules for sending a letter (address on the front, sender address on the back, country at the bottom when sending abroad, stamps...) there are rules for sending messages on a computer network. The word used isn't "rules" but "protocols", which basically means the same ("behavior rules").

Ports are like mail boxes in a building full of independent apartments…

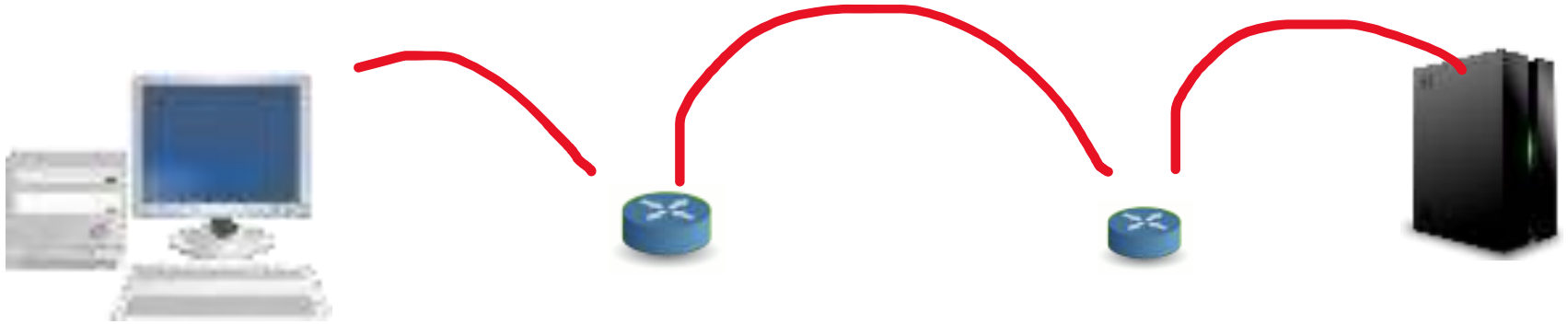
To be able to send a message, some program must be there to receive it. Some programs, collectively known as listeners or servers, do this. There may be several ones on one computer, they are listening for a "port" which is just a number that defines a service.

TCP



To send your message you must provide the port, but also the address or name of the computer (a name will be converted to an address). You may have seen IP address, they look like 192.254.23.127. Your message will be packaged with this information according to a set of rules known as the TCP (= Transmission Control Protocol).

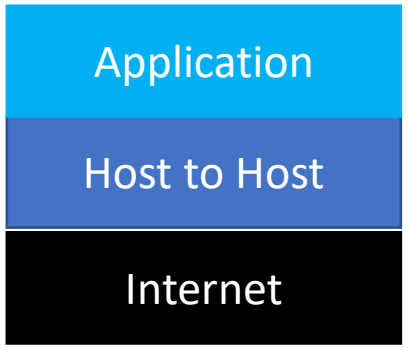
IP



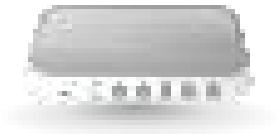
TCP relies on another set of rules that ensure that your message travels from your network to the target network:

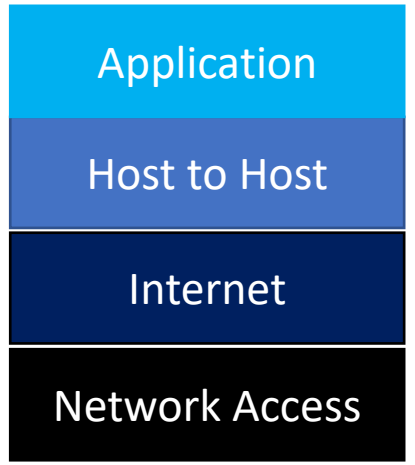
This is the IP or Internet Protocol

Internet just means between networks



Network Access





All this can only work if you can reach the network and the gateway, where machines known as "routers" will direct your message.

How does it work?

We have seen URI/URL (more or less the same already) a lot of operations in which networks are involved are transparent, and done by methods in libraries. However, if you have special needs, you may want to code a network application of your own.

Socket

```
Socket s = new Socket (hostname, port_number);

in = new BufferedReader(new
       InputStreamReader(s.getInputStream()));

out = new BufferedWriter(new
       OutputStreamWriter(s.getOutputStream()));
```

What travels along a network is nothing more than a byte stream. As a socket works in both directions, you have both an input and an output stream.

Sending and Listening for Messages

```
while ((fromServer = in.readLine()) != null) {
    // Analyze fromServer message
    // Prepare fromUser message
    out.write(fromUser);
}
in.close();
out.close();
s.close();
```

It's just a matter of reading and writing. All the lower level protocols are managed inside the socket object (much easier than in C). However YOUR application needs a “protocol": messages and answers probably should have a meaning!

More Examples from JavaNotes 8.1

Read section 11.4 Networking:
http://math.hws.edu/eck/cs124/javanotes8/c11/s4.html

Example of a trivial client server
http://math.hws.edu/eck/cs124/javanotes8/c11/s4.html#IO.4.4

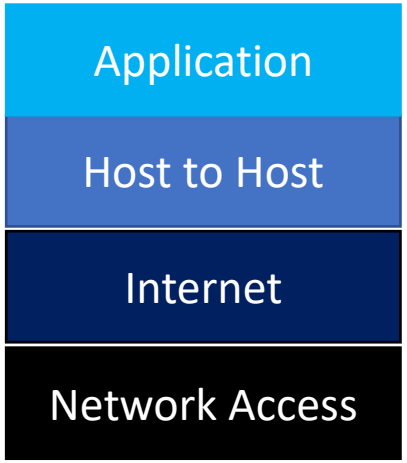
Example a simple network chat (exchanging messages between a client and a server):

http://math.hws.edu/eck/cs124/javanotes8/c11/s4.html#IO.4.5

Network Programming

So far…

- Computers that can “talk” to one another
- Stack of protocols (rules)



- Programming with Sockets

Sockets…

In Java (as in C) network communications are based on a "socket". In Java it's a class, and you use it like a file.

client socket

server socket

import java.net.*

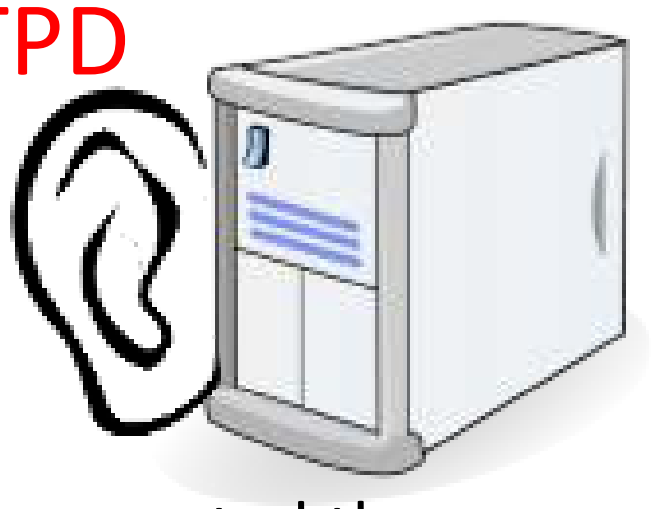
class Socket — talks to ONE server

class ServerSocket — talks to multiple clients

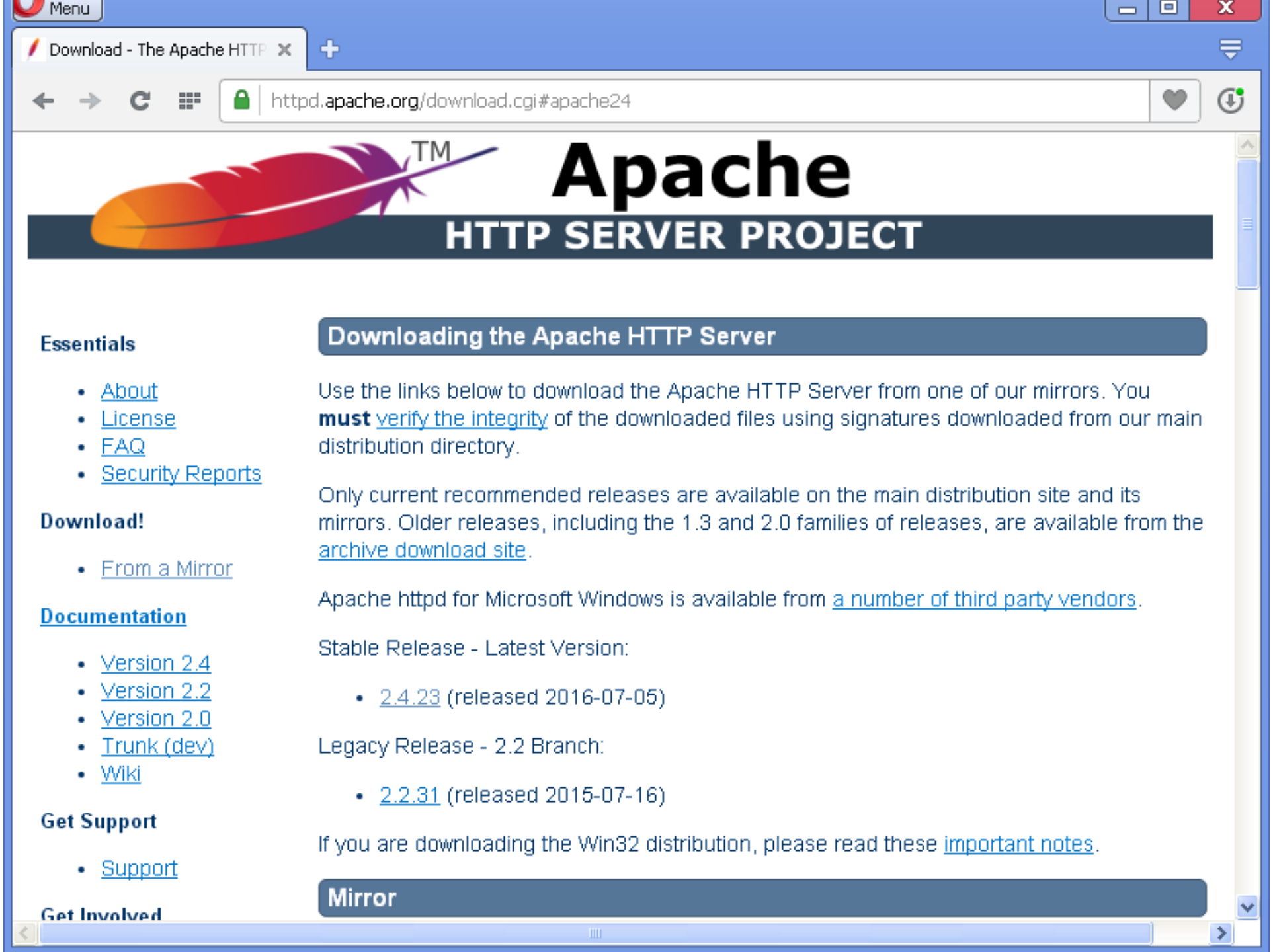
Example: Get the Home Page of a Website

A simple example is getting the home page of any website. Your browser uses a protocol the name of which must be familiar to you: HTTP, or Hyper-Text Transfer Protocol. It sends messages that must be understandable to a server, and the server sends back "web pages", using a protocol that must be understandable by the browser. As this protocol is publicly specified, anybody can write an HTTP server or a browser as long as you respect the rules.

Server Example with Apache Daemon

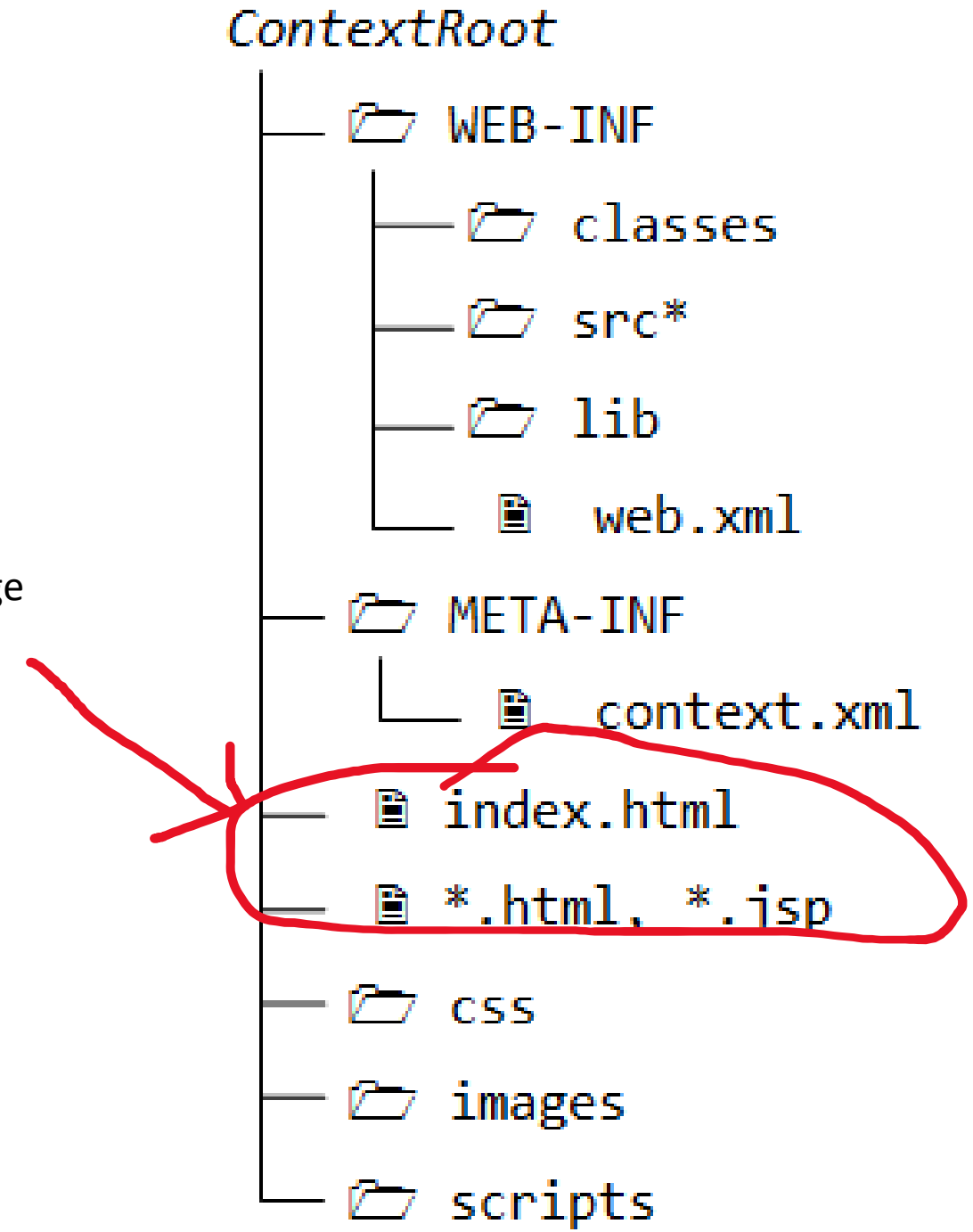


In the case of HTTP, I have represented the server by an Apache server ("Apache" is a very popular web server). The program that runs is called "httpd", the "d" is often used in Unix systems to indicate that a program runs in the background without interacting with the screen nor the keyboard (these programs are called "**d**aemons" in Unix systems)


Menu
Download - The Apache HTTP
httpd.apache.org/download.cgi#apache24
Apache
HTTP SERVER PROJECT
Essentials
About
License
FAQ
Security Reports
Download!
From a Mirror
Documentation
Version 2.4
Version 2.2
Version 2.0
Trunk (dev)
Wiki
Get Support
Support
Get Involved
Downloading the Apache HTTP Server
Use the links below to download the Apache HTTP Server from one of our mirrors. You must verify the integrity of the downloaded files using signatures downloaded from our main distribution directory.
Only current recommended releases are available on the main distribution site and its mirrors. Older releases, including the 1.3 and 2.0 families of releases, are available from the archive download site.
Apache httpd for Microsoft Windows is available from a number of third party vendors.
Stable Release - Latest Version:
2.4.23 (released 2016-07-05)
Legacy Release - 2.2 Branch:
2.2.31 (released 2015-07-16)
If you are downloading the Win32 distribution, please read these important notes.
Mirror

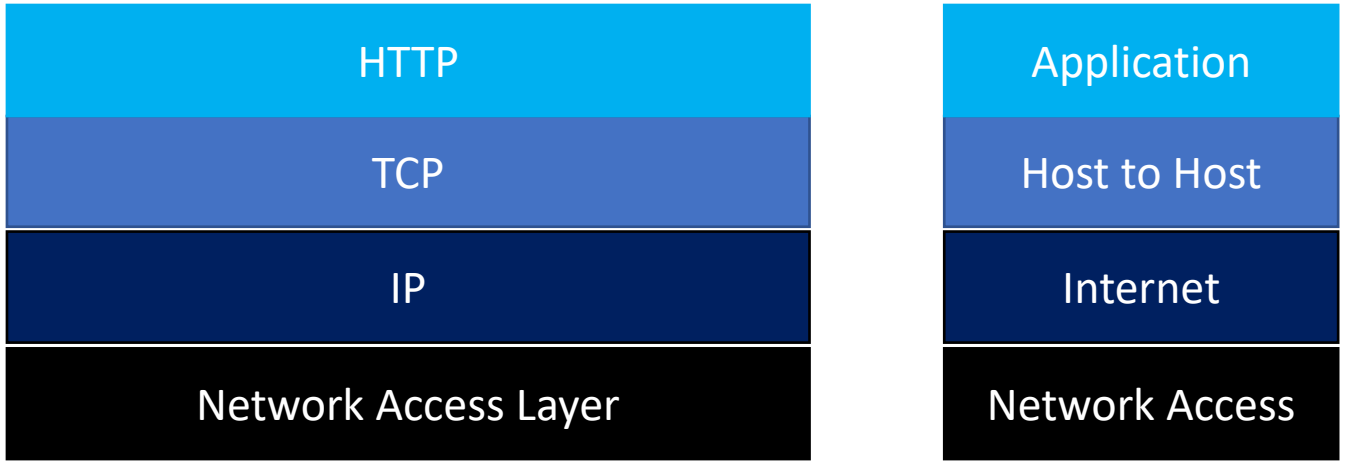
Apache Tomcat

Home Page

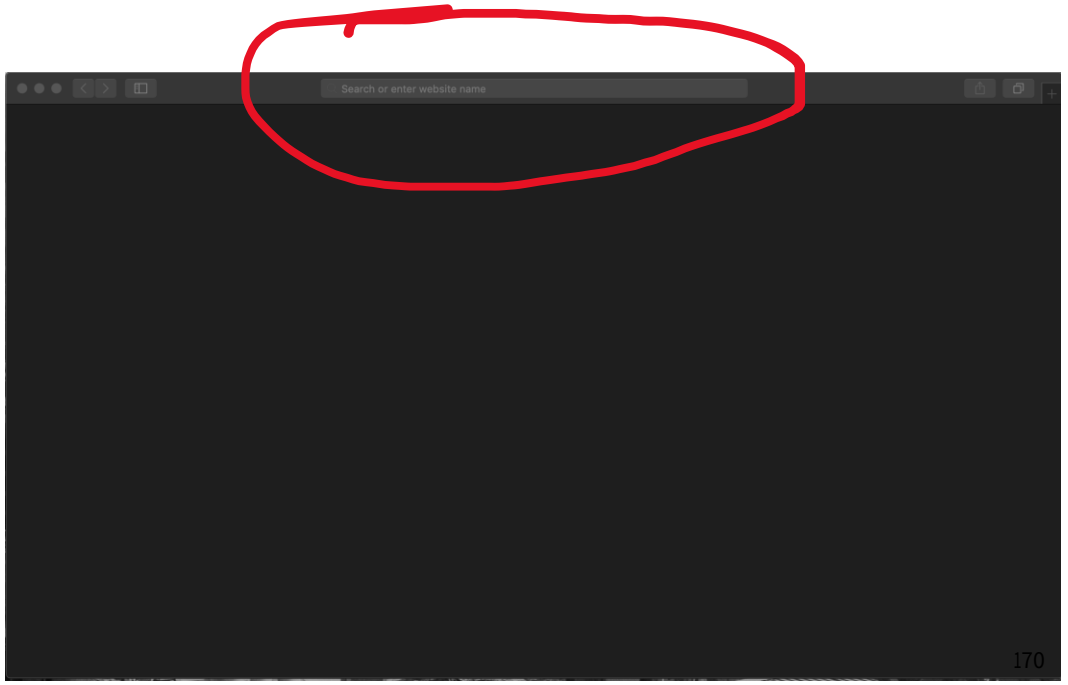


HTTP

HTTP is an application protocol on top of TCP: it consists of just text with special character sequences.



When you type an address in your browser, e.g. the local network address 192.168.254.100 ,the browser knows that this is supposed to be a HTTP server. HTTP servers listen on port 80 (reserved for them), so it will try to connect to port 80 on this machine. If the connection is successful (a program is listening and accepting the connection!) the browser automatically sends a message.



A GET Request

GET / HTTP/1.1

Host: 192.168.254.100

User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:2.0.1) [...]

Accept: text/html,application/xhtml+xml,application/xml;[...]

Accept-Language: en-gb,en;q=0.5

Accept-Encoding: gzip, deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7

Keep-Alive: 115

Connection: keep-alive

Cache-Control: max-age=0

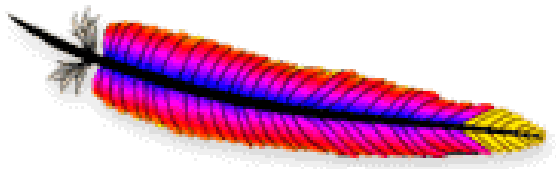
[empty line]

- This is what is sent; only the first two lines and the empty line at the end are mandatory.
- The first line says "send me your home page, I'm talking this version of HTTP". The second line repeats the server address for control.

Apache Server

- An Apache server will look into some special directories, will check for a .htaccess file that may require a password, and if everything is OK will send back file index.html.

- Directory structure:
  - var
    - www
      - htdocs
        - .htaccess
        - index.html



Apache Commons™
http://commons.apache.org/

permissions are set using the .htaccess

index.html

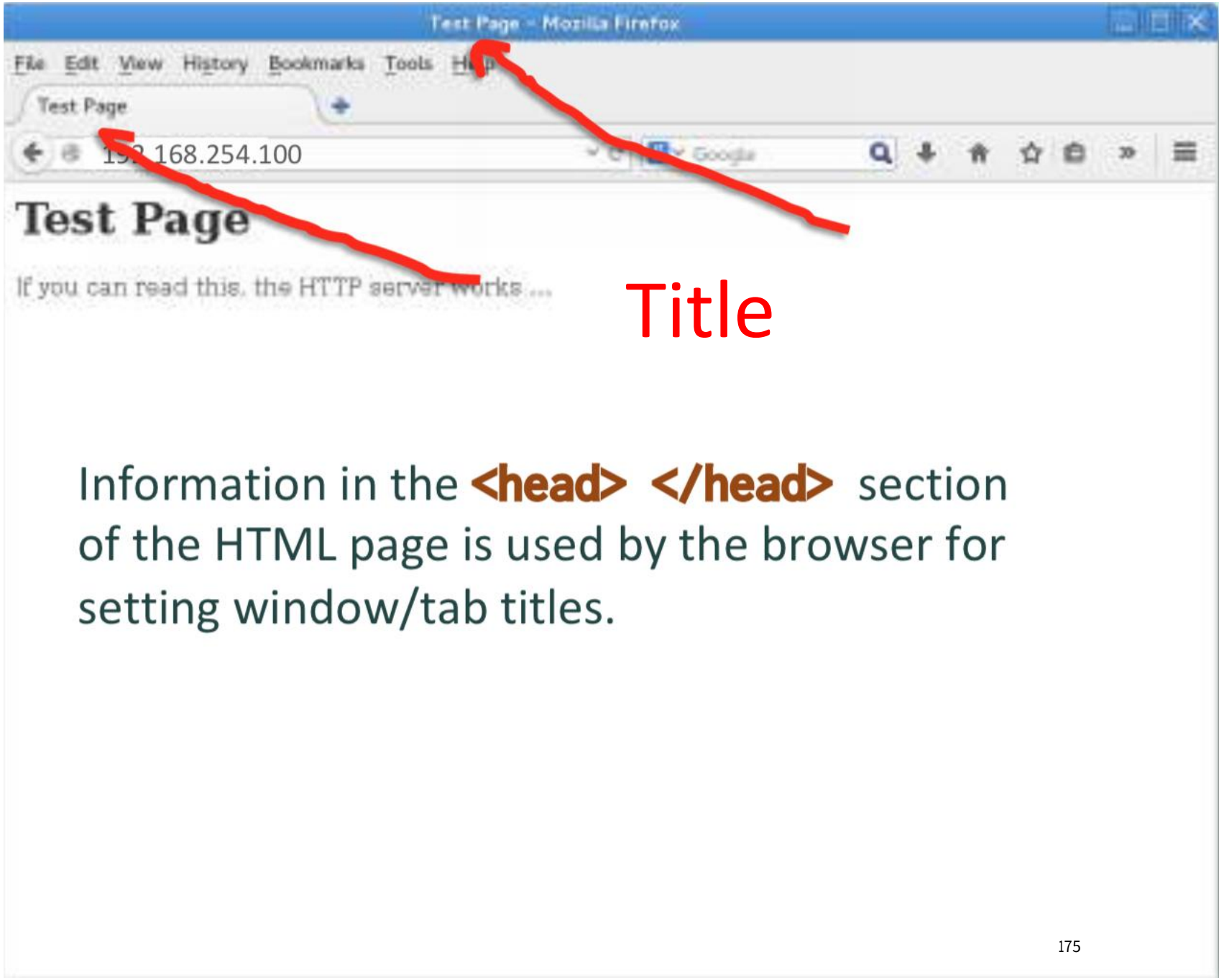
- index.html may look like this.
- What the browser will display is between <body> and </body>

```
<!doctype html>
<html>
  <head>
    <title>Test Page</title>
  </head>
  <body>
    <h1>Test Page</h1>
    <p>If you can read this, the HTTP server works ...</p>
  </body>
</html>
```

index.html

```
HTTP/1.1 200 OK
Date: [...]
Server: Apache/2/2/15 (Linux/SUSE)
Last-Modified: [date]
ETag: "2d7d-b7-4a3c52ef29046"
Accept-Ranges: bytes
Content-Length: 175
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
[empty line]
<html>
  <head>
    <title>Test Page</title>
  </head>
  <body>
    <h1>Test Page</h1>
    <p>If you can read this, the HTTP server works ...</p>
  </body>
</html>
```

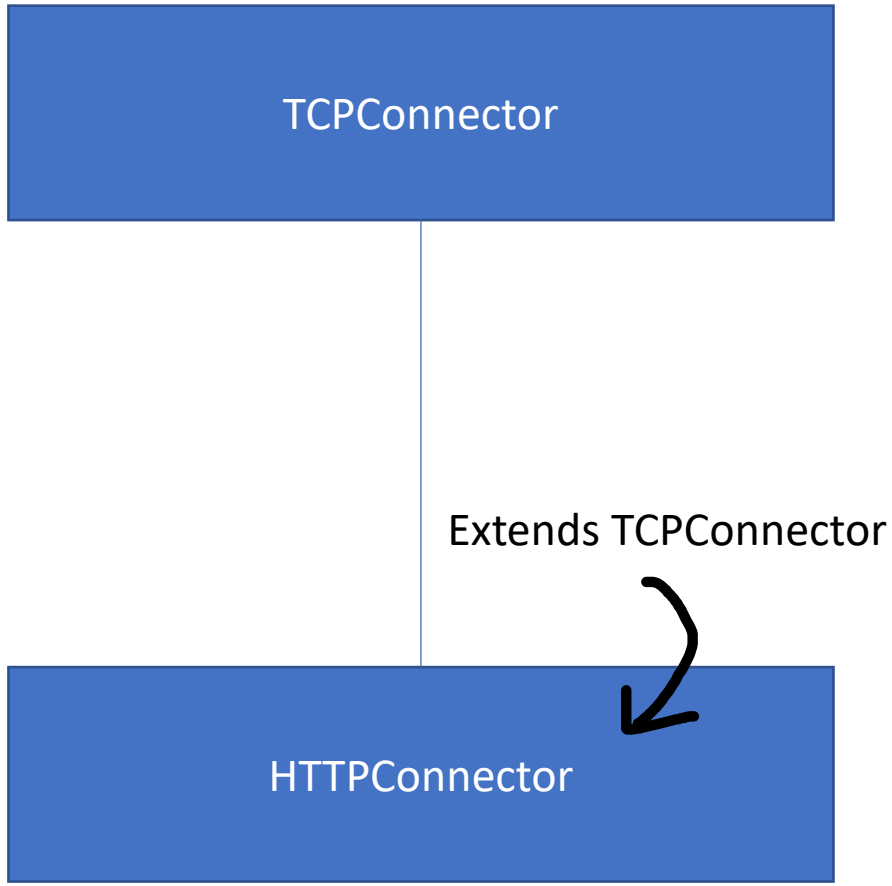
The Rendered (Generated) Page



Information in the **<head> </head>** section of the HTML page is used by the browser for setting window/tab titles.

Design*

- 2 Classes
- TCPConnector
  - Creates a socket
  - Set up streams
  - Send and receive messages
- HTTPConnector
  - Communicate HTTP
  - Always use port 80



(*credit Vic Hargrave)

TCPConnector Class

The TCPConnector class sets up a socket for communicating with a server.

Here are the fields defined in the class…

```
import java.net.Socket;
import java.net.SocketTimeoutException;
import java.io.*;

class TCPConnector {

private hostName;      www.example.com
private hostAddr;      123.123.123.123
private int port;

private Socket s = null;
private BufferedReader in = null;
private BufferedWriter out = null;
```

TCPConnector

- A constructor.
- getInetAddress() returns, when transformed into a string, a website name (possibly empty) followed by "/" and an IP address…
- Another method getHostAddress() returns the text resolution of the domain name (e.g. www.example.com).

```
1  import java.net.Socket;
2  import java.net.SocketTimeoutException;
3  import java.io.*;
4
5  class TCPConnector {
6
7  private hostName;
8  private hostAddr;
9  private int port;
10
11 private Socket s = null;
12 private BufferedReader in = null;
13 private BufferedWriter out = null;
14
15 public TCPConnector(String host, int portNum) throws IOException
16     hostName = host;
17     port = portNum;
18
19     s = new Socket(host, portNum);
20     s.setSoTimeout(1000);
21     hostAddr = s.getInetAddress().toString().split("/")[1];
22
23     in = new BufferedReader(new
24     InputStreamReader(s.getInputStream()));
25
26     out = new BufferedWriter(new
27     OutputStreamWriter(s.getOutputStream()));
28 }
```

```
28  }
29
30  public void close() throws IOException {
31      try {
32          in.close();
33          out.close();
34          s.close();
35      } catch (java.net.SocketException e) {
36          // Do nothing
37      }
38  }
39
40  public String getHostAddr() {
41      return hostAddr;
42  }
```

Closing connections properly is important ...

TCPConnector

- Finally another two simple methods to receive and send a message.
- You need to flush the message when sending, otherwise it won't go before buffers are full...

```
42  }
43
44  public void send(String msg) throws IOException {
45      out.write(msg);
46      out.flush(); // IMPORTANT
47  }
48
49  public String receive() throws IOException {
50      try {
51          String s = in.readLine();
52          return s;
53      } catch (SocketTimeoutException e) {
54          return null;
55      }
56  }
```

```
public class HTTPConnector extends TCPConnector {
    private StringBuffer header;
    private StringBuffer body;
```

```
  public HTTPConnector(String host) throws IOException {
      super(host, 80);
      header = new StringBuffer();
      body = new StringBuffer();
```

```
   public String get(String pagename) throws IOException {
       String msg;
       header.delete(0, header.length());
       body.delete(0, body.length());

       header.append("GET " + pagename + " HTTP/1.1\n");
```

- The HTTPConnector class extends the TCP class just defined. HTTP is an application protocol that operates ”on top” of TCP and consists of text including headers, body tags etc, it uses TCP to transfer the information across a network in packets (more in networking class)
- The get function handles header and body separately. The goal is to be able to store the length of the body in the header so that the other end can check that the full message was received.

HTTPConnector

```
public String get(String pagename) throws IOException {
    String msg;
    header.delete(0, header.length());
    body.delete(0, body.length());

    header.append("GET " + pagename + " HTTP/1.1\n");
    header.append("Host: " + getHostAddr() + "\n");
    header.append("User-Agent: Java test program\n");
    header.append("\n");
    send(header.toString());
    header.delete(0, header.length());

    boolean reading_header = true;
    int bytesToRead = -1;
    int read = 0;
    while ((bytesToRead != 0) && ((msg = receive()) != null)) {
        if (reading_header) {
        if (msg.trim().isEmpty()) {
            reading_header = false;
        }
```

- We send the basic message with a GET request header and then wait for the answer.
- We first read the header (stopping when hit an empty line)

HTTPConnector

condition: “bytesToRead” in body and message is not null

```
boolean reading_header = true;
int bytesToRead = -1;
int read = 0;
while ((bytesToRead != 0) && ((msg = receive()) != null)) {
    if (reading_header) {
        if (msg.trim().isEmpty()) {
        reading_header = false;
        } else {
            if (msg.toLowerCase()
                                .startsWith("content-length")) {

                bytesToRead = Integer
                                    .parseInt(msg.split(":")[1]
                                    .trim());
            }
            header.append(msg);
        }
    }
```

When reading the header, if the content length is given then we read this length (and assign to bytesToRead which then allows the loop to continue until all data is read)

```
                header.append(msg);
            }
        } else{
            read = 1 + msg.length();
            body.append(msg);
            body.append("\n");
            bytesToRead -= read;
        }
    }
return body.toString();
}
```

And we finally return the body (and only the body) that we have read when there are no more bytes to read.

```
public String get(String pagename) throws IOException {
    String msg;
    header.delete(0, header.length());
    body.delete(0, body.length());

    header.append("GET " + pagename + " HTTP/1.1\n");
    header.append("Host: " + getHostAddr() + "\n");
    header.append("User-Agent: Java test program\n");
    header.append("\n");
    send(header.toString());
    header.delete(0, header.length());

    boolean reading_header = true;
    int bytesToRead = -1;
    int read = 0;
    while ((bytesToRead != 0) && ((msg = receive()) != null)) {
        if (reading_header) {
            if (msg.trim().isEmpty()) {
            reading_header = false;
            } else {
                if (msg.toLowerCase()
                                    .startsWith("content-length")) {

                    bytesToRead = Integer
                                        .parseInt(msg.split(":")[1]
                                        .trim());
                }
                header.append(msg);
            }
        } else{
            read = 1 + msg.length();
            body.append(msg);
            body.append("\n");
            bytesToRead -= read;
        }
    }
return body.toString();
```

```
lic class GetHomePage {

 public static void main(String[] args) throws IOExcept
 if (args.length > 0) {
     HTTPConnector h = new HTTPConnector(args[0]);
     System.out.println(h.get("/"));
     h.close();
     }
 }
```

The class HTTPConnector can be used get any web page, to get the home page we are asking for the home-page specifically at the root “/” directory.

Network Features in Java Packages

- Many packages in Java have built-in networking capabilities

- All this will actually be automatically done for you if you pass anything that starts with "http:" to a method that takes a URI or URL parameter

URI Class: see java.net.URI
(https://docs.oracle.com/javase/7/docs/api/java/net/URI.html)

The URI class represents a Uniform Resource Identifier (URI) reference.

```
URI resourceName = new URI("...");
"file:... "
"http:... "
```

http://www.iana.org/assignments/uri-schemes/uri-schemes.xhtml

Designing a Client Server Application: Server

Week 13 – Presentation 2

Designing a Server…

Writing a server is hardly more difficult than writing a client (actually, it's easier, because a server doesn't need a nice interface).

1. Define a protocol for (client-server) communications
2. Server
3. Client

Protocol: *"what messages does the server understand?"*

- To write a client/server application the first thing to define was the protocol, which is basically defining the list of commands recognized by the server and how it should respond to each of them.
- You are going to send some commands to the server. What does it understand? How does it reply? What does it say when it receives a wrong command? what are the parameters associated with a command?
- Of course, all error cases should also be thought of! You can have several cases of errors, client side errors (the client sent either a command you couldn't make sense of, or misused a command), or server side errors (... for instance for some obscure reason you cannot connect to the database you need).

Example: Protocol for Queries of a Film Database

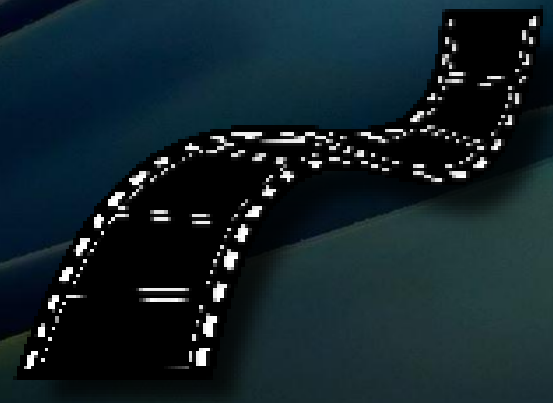
- Query the database for films that match certain conditions
  - Title
  - Director
  - Actors/Actresses
  - Year
  - Country

see www.imdb.com for instance for a database of films with a search feature.

Example: Protocol for Queries of a Film Database

Application Protocol (messages that the applications uses in queries for films)

It will run on top of other network protocols such as TCP or UDP



Such a protocol could be a keyword followed by a condition:

- **KEYWORD** cond.

This message would ask for the list of films in which Audrey Hepburn played:

- EG **ACTRESS** Audrey Hepburn

Example: Protocol for Queries of a Film Database

- I may want to implement "or"

KEYWORD cond[**|**cond ...]

- This would return films with either Audrey Hepburn or Ingrid Bergman:

ACTRESS Audrey Hepburn **|** Ingrid Bergman

Example: Protocol for Queries of a Film Database

- I may also want the films to match some other conditions

KEYWORD cond[|cond ...][, **OTHER_KEYWORD** cond[|cond ...] ...]

- This becomes films with either Audrey Hepburn or Ingrid Bergman, but directed by Hitchcock.

ACTRESS Audrey Hepburn **|** Ingrid Bergman**, DIRECTOR** Hitchcock

Example: Protocol for Queries of a Film Database

```
public class FilmProtocol {
    private static Connection con = null;

    public FilmProtocol(Connection cnx) {
        con = cnx;
    }

    public String processInput(String theInput) {
        ...
    }

    private String runQuery(String query) {
        ...
    }
}
```

calls a method to query a database

Code Outline for such a protocol:

A protocol parses (analyzes) the message, builds the suitable query and runs the query against the database.

Server

Just a **big** loop

The server just waits for queries, and executes them.

Film Server Example

Notice passing the port we are using on the command line (never hard-code it in the program, the port may already be taken)

```
import java.net.*;
import java.io.*;
import java.sql.*;

public class FilmServer {
    public static void main(String[] args) throws IOException {
    Connection con = null;
    if (args.length != 1) {
        System.err.println("Usage: java FilmServer <port number>");
        System.exit(1);
        }

    //
    // Here, connect to the database (not shown)
    //
```

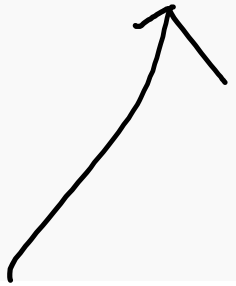
Film Server Example

We create a FilmProtocol (that does basically all the hard bits in the job) then create a socket and loop forever (should in a real app have a way to stop it nicely).

```
//
FilmProtocol    filmP = new FilmProtocol(con);
int             portNumber = Integer.parseInt(args[0]);
String          inputLine, outputLine;
PrintWriter     out = null;
BufferedReader  in = null;
ServerSocket    serverSocket = null;
Socket          clientSocket = null;


try {
    serverSocket = new ServerSocket(portNumber);
    System.err.println("Film server started on port "+ args[0]);
    while (true) {
        clientSocket = serverSocket.accept();
        System.err.println("Accepted connection");
        out = new PrintWriter(clientSocket.getOutputStream(), true);
        in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
```

autoflush = true

```
    try {
        serverSocket = new ServerSocket(portNumber);
        System.err.println("Film server started on port "+ args[0]);
        while (true) {
            clientSocket = serverSocket.accept();
            System.err.println("Accepted connection");
            out = new PrintWriter(clientSocket.getOutputStream(), true);
            in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

            // Wait for input
            if ((inputLine = in.readLine()) != null) {
                outputLine = filmP.processInput(inputLine); out.println(outputLine);
            }
            clientSocket.close();
        }
    }
    catch (...) {
            ...
    } finally {
            ...
    }

}
```



the FilmProtocol Object that does the tough bits (parsing the input line and generating the required SQL query)

Designing a Client Server Application: Client

Week 13 – Presentation 3

Client

- Can be very Basic:
  - command line tool
- Or much more complex:
  - e.g. a website or a GUI
- Don't underestimate ugly console applications – they are easier to integrate with other processes.

- Standard data exchange formats: JSON, XML, CSV

- The client might also use a bit of its processing power to try to present the result better. It would make the job easier if data returned by the server were better formatted.

Basic Film Client Example

Once the server is ready and that we know the protocol, time to write a (command-line here) client. It won't be a sexy program, but it will be functional.

Basic Film Client Example

```
import java.io.*;
import java.net.*;

public class FilmClient {

public static void main(String[] args) throws IOException {
    if (args.length != 2) {
        System.err.println("Usage: java FilmClient <host name> <port number>");
        System.exit(1);
    }
    String hostName = args[0];
    int portNumber = Integer.parseInt(args[1]);
    boolean loop = true;
```

The client has to be told where to connect (host and port)…

Basic Film Client Example

```
boolean loop = true;
while (loop) {
    try ( // try with resources
        Socket sock = new Socket(hostName, portNumber);
        PrintWriter out = new PrintWriter(sock.getOutputStream(), true);
        BufferedReader in = new BufferedReader(
        new InputStreamReader(sock.getInputStream()));
        ) {
            BufferedReader stdIn =
                    new BufferedReader(new InputStreamReader(System.in));
            String fromServer;
            String fromUser;
```

As every query is independent (there is no "session") here we create (and close) one connection for each query.

Basic Film Client Example

```
String fromUser;
System.out.print("Query> "); // read from input
fromUser = stdIn.readLine(); // send to server out.println(fromUser);
// read from server
while ((fromServer = in.readLine()) != null) {
    if (fromServer.length() > 0) {
        System.out.println(fromServer);
    }
if (fromServer.equals("Goodbye")) {
    loop = false;
    break;
    }
}
```

On exit the server will send an acknowledgement (“Goodbye”).

Any error messages could also be received from the server here and displayed by the client as unprocessed text data from the server in this example.

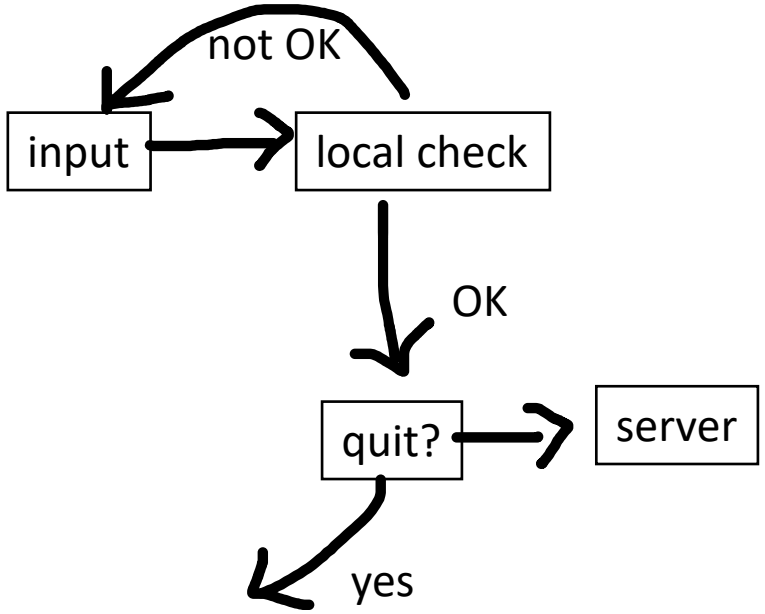
Basic Film Client Example

```
            break;
            }
        }
    } catch (UnknownHostException e) {
        System.err.println("Don't know about host " + hostName);
        System.exit(1);
    } catch (IOException e) {
        System.err.println("Couldn't get I/O to " + hostName);
        System.exit(1);
        }
    }
```

Handling connection errors in case client is unable to contact the server at all.

Client Design

- This client is quite simple and generally it is not efficient to let the server check everything
- It would be better if it knew the protocol and could check before sending if the message is correct.
- It would give less work to the server and require less network bandwidth.



Client Design

- Further potential improvements to client are in rendering and presenting the data
- Standard data exchange format such as CSV, XML, JSON ...
- The client might also use a bit of its processing power to try to present the result better. It would make the job easier if data returned by the server were better formatted.

Limitation: One Client at a Time

- The main weakness of previous examples we have seen of servers is that when it processes a query, it cannot check if there is another request. This was OK because there were not hundreds of requests arriving at the same time and queries execute fast. But think of the traffic on a popular website, for instance an ecommerce one, where a lot of users are asking for pages that are built on the fly ...

- Solution: multithreading.

Software System Design: Modular Systems

Week 13 – Presentation 4

Software is Complex

Complex ≠ complicated

Complex = composed of many simple parts

related to one another

Complicated = not well understood, or explained

Software Engineering is Complex and Complicated

- Large scale software systems have many components and this tends to lead to chaos
- It is difficult to define requirements as users cannot know what the precisely want or need and they evolve over time, the creeping change of requirements is a primary reason for software project failure (relationship of system function – behavior – structure)
- Also different designs can achieve similar functions and it is difficult to judge whether a design is good (Conway ’s law: The structure of software tends to be similar to the organizational structure of the development team)

Function-Behaviour-Structure Ontology

https://en.wikipedia.org/wiki/Function-Behaviour-Structure_ontology

Gero J.S. (1990) "Design prototypes: a knowledge representation schema for design", AI Magazine, 11(4), pp. 26–36.

- Function (F): the teleology (purpose) of the design object.
- Behavior (B): the attributes that can be derived from the design object's structure.
- Structure (S): the components of the design object and their relationships.
- The three ontological categories are interconnected:

Function is connected with behavior, and behavior is connected with structure. There is no connection between function and structure!

Conway’s Law

Conway's Law:

The rule that the organization of the software and the organization of the software team will be congruent; commonly stated as “If you have four groups working on a compiler, you'll get a 4-pass compiler”.

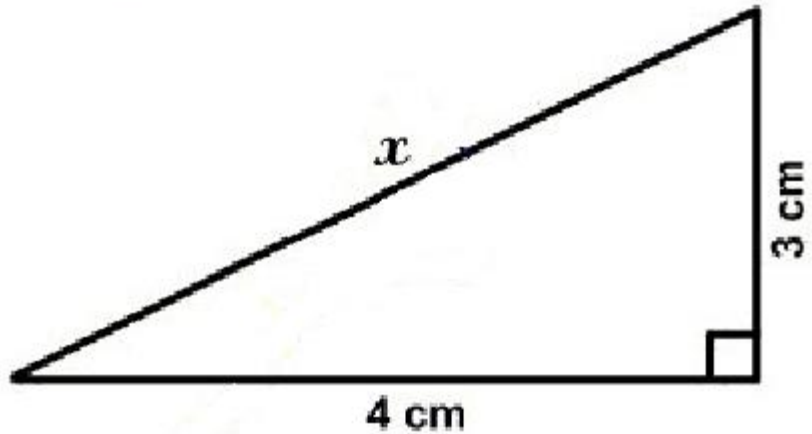
The original statement was more general, “Organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations.” This first appeared in the April 1968 issue of *Datamation*.

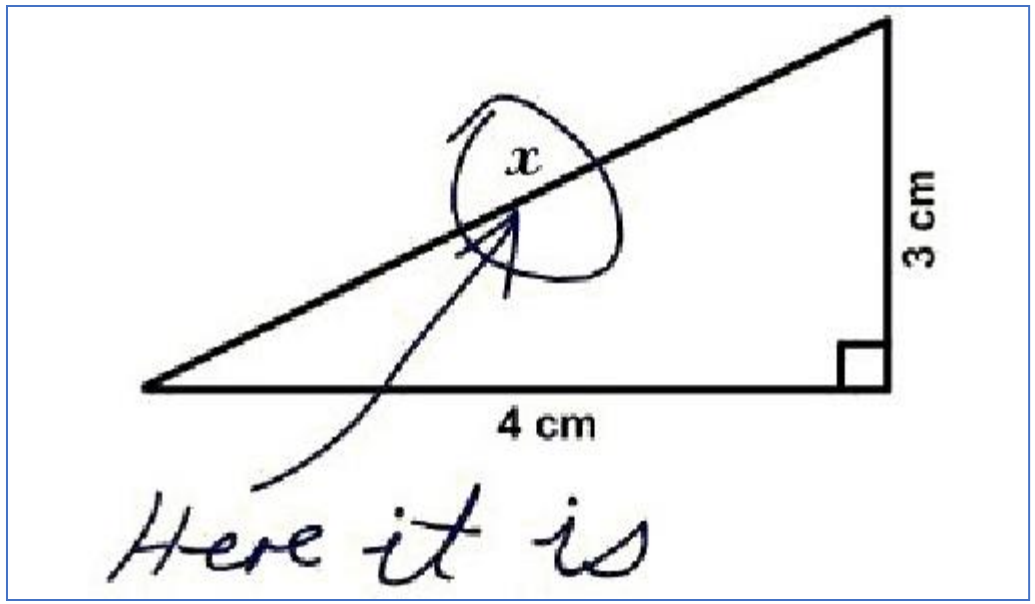
http://www.catb.org/~esr/jargon/html/C/Conways-Law.html

One of the Reason to explain Why Software Development is difficult:

How to judge whether a design is good or not is a big challenge, that is largely based on experience.

3. Find x.



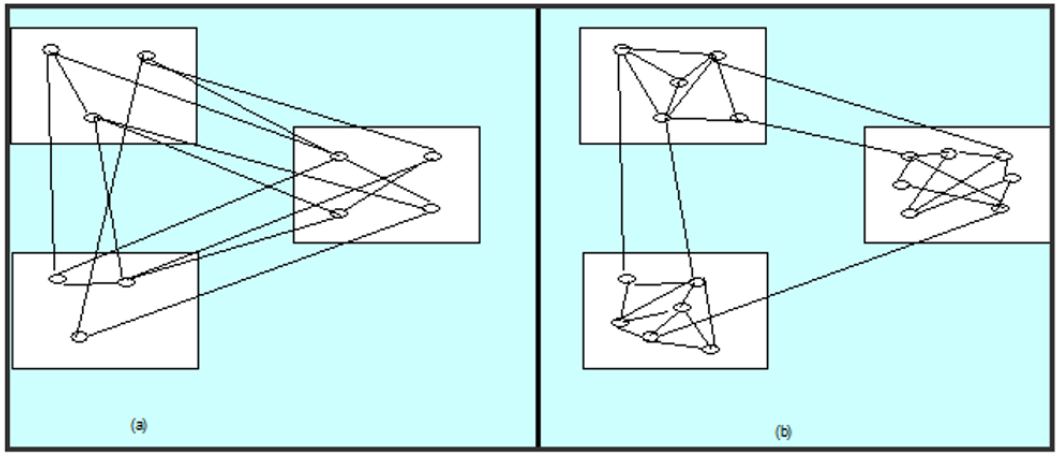


Decomposing Systems into Modules

- High cohesion within modules
- Loose coupling between modules

*Parnas, D.L. (Dec. 1972). "On the Criteria To Be Used in Decomposing Systems into Modules". Communications of the ACM. **15** (12): 1053–58.*

A visual representation



high coupling (low cohesion)

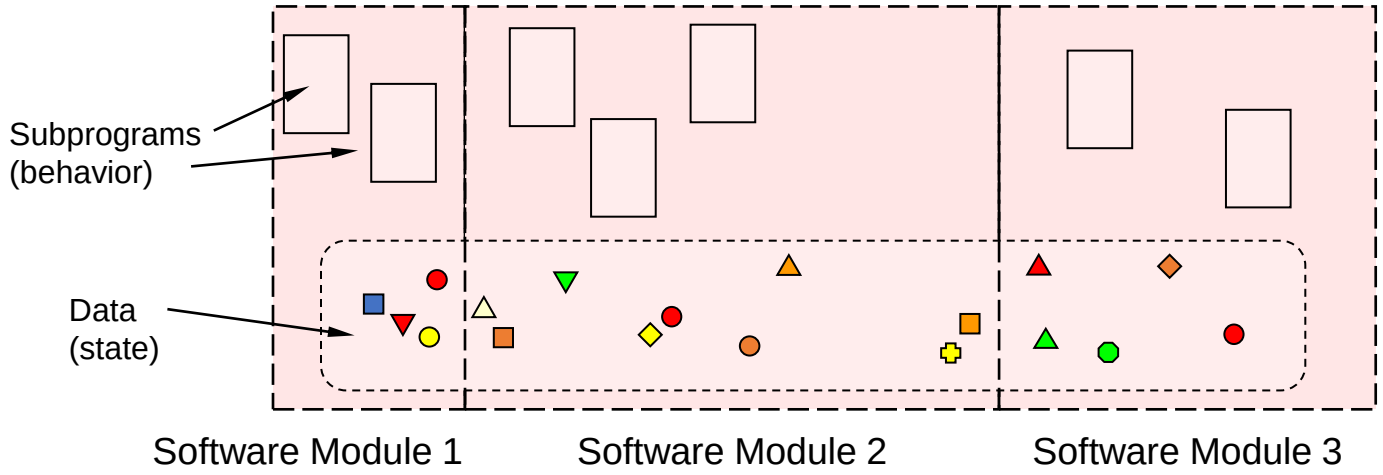
low coupling (high cohesion)

Modularity, Cohesion and Coupling

- A complex system may be divided into simpler pieces called *modules*
- A system that is composed of modules is called *modular*
- Supports application of separation of concerns
  - when dealing with a module we can ignore details of other modules
- Each module should be *highly cohesive*
  - module understandable as a meaningful unit
  - Components of a module are closely related to one another
- Modules should exhibit *low coupling*
  - modules have low interactions with others
  - understandable separately

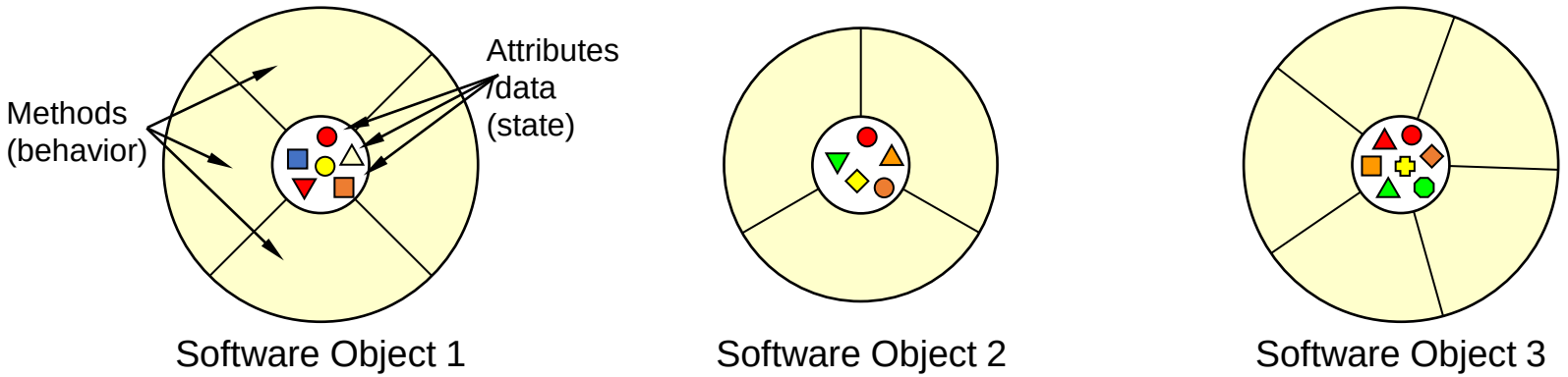
Modules versus Objects

Modules are loose groupings of subprograms and data



"Promiscuous" access to data often results in misuse

Objects ***encapsulate*** data

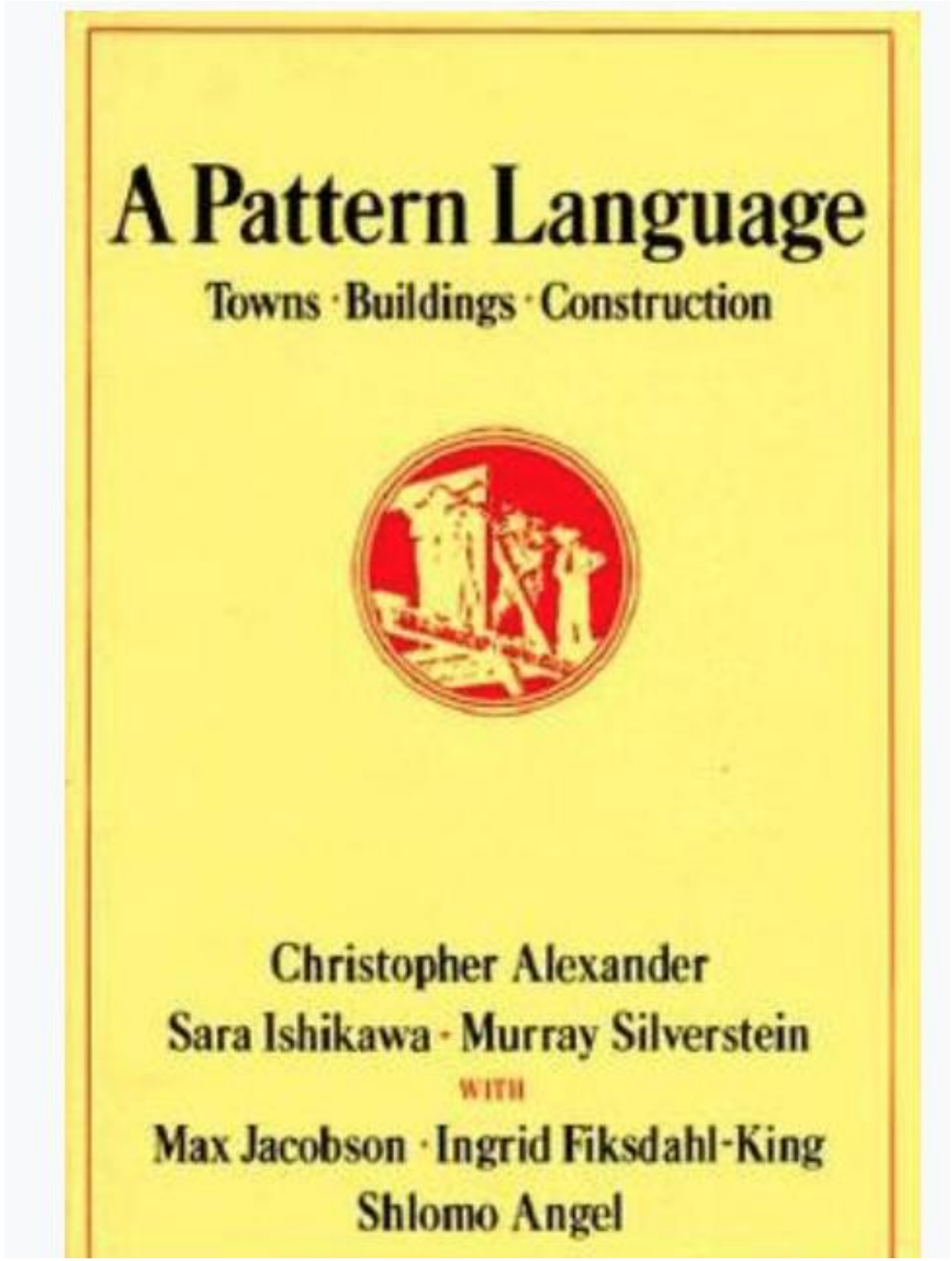


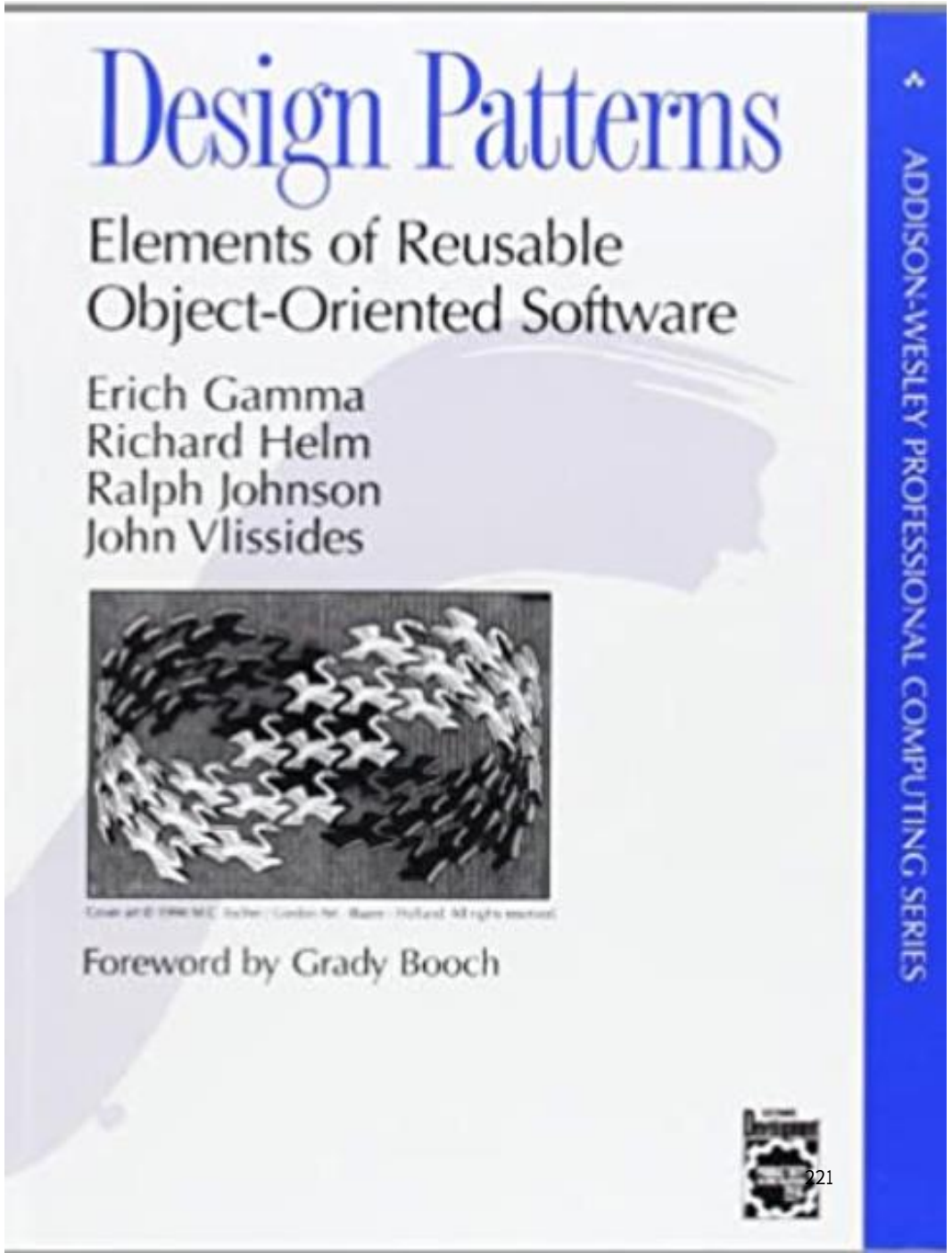
Software System Design: Patterns

Week 13 – Presentation 5

The Pattern Concept

- History: Architectural Patterns
- Christopher Alexander
- Each pattern has
  - a short *name*
  - a brief description of the *context*
  - a lengthy description of the *problem*
  - a prescription for the *solution*


A Pattern Language
Towns · Buildings · Construction
Christopher Alexander
Sara Ishikawa · Murray Silverstein
WITH
Max Jacobson · Ingrid Fiksdahl-King
Shlomo Angel


Design Patterns
Elements of Reusable
Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Short Passages Pattern

Short Passages Pattern

Context

"...Long, sterile corridors set the scene for everything bad about modern architecture..."

Problem

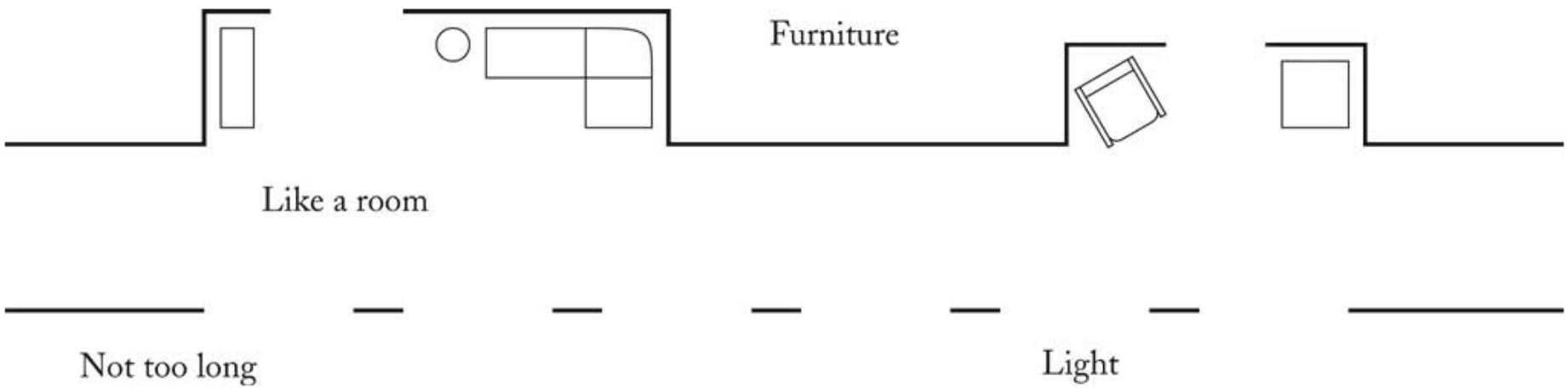
a lengthy description of the problem, including

- a depressing picture
- issues of light and furniture
- research about patient anxiety in hospitals
- research that suggests that corridors over 50 ft are considered uncomfortable

Short Passages Pattern

Solution

Keep passages short. Make them as much like rooms as possible, with carpets or wood on the floor, furniture, bookshelves, beautiful windows. Make them generous in shape and always give them plenty of light; the best corridors and passages of all are those which have windows along an entire wall.



Iterator Pattern

Context

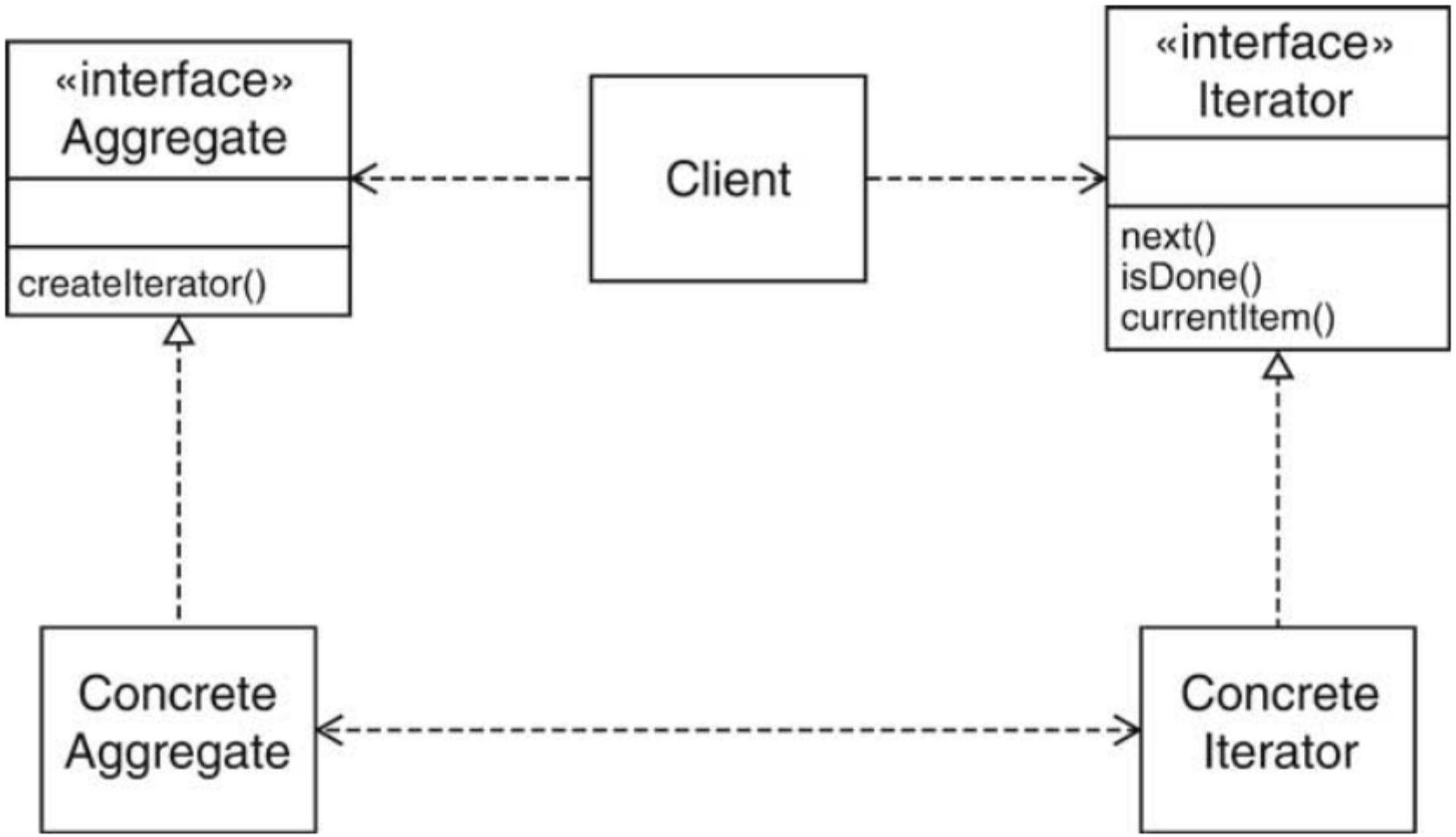
1. An aggregate object contains element objects
2. Clients need access to the element objects
3. The aggregate object should not expose its internal structure
4. Multiple clients may want independent access

Iterator Pattern

Solution

1. Define an iterator that fetches one element at a time
2. Each iterator object keeps track of the position of the next element
3. If there are several aggregate/iterator variations, it is best if the aggregate and iterator classes realize common interface types.

Iterator Pattern



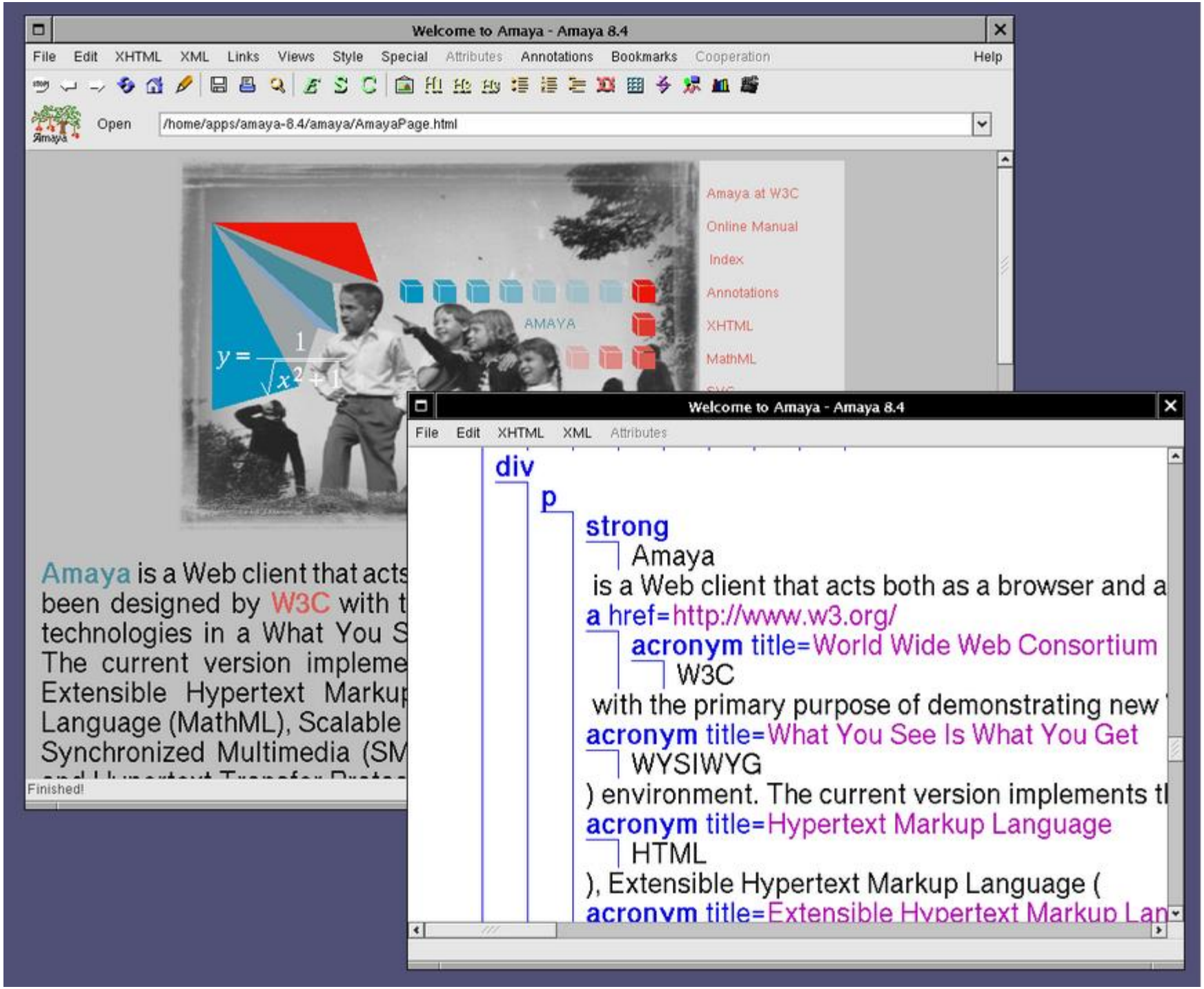
Iterator Pattern

- Names in pattern are *examples*
- Names differ in each occurrence of pattern

Name in Design Pattern	Actual Name (linked lists)
`Aggregate`	`List`
`ConcreteAggregate`	`LinkedList`
`Iterator`	`ListIterator`
`ConcreteIterator`	anonymous class implementing `ListIterator`
`createIterator()`	`listIterator()`
`next()`	`next()`
`isDone()`	opposite of `hasNext()`
`currentItem()`	return value of `next()`

Model/View/Controller

- Some programs have multiple editable views
- Example: HTML Editor
  - WYSIWYG view
  - structure view
  - source view
- Editing one view updates the other
- Updates seem instantaneous


Welcome to Amaya - Amaya 8.4
File Edit XHTML XML Links Views Style Special Attributes Annotations Bookmarks Cooperation Help
Open
/home/apps/amaya-8.4/amaya/AmayaPage.html
Amaya at W3C
Online Manual
Index
Annotations
XHTML
MathML
AMAYA
Amaya is a Web client that acts
been designed by W3C with t
technologies in a What You S
The current version impleme
Extensible Hypertext Markup
Language (MathML), Scalable
Synchronized Multimedia (SM
Finished!
Welcome to Amaya - Amaya 8.4
File Edit XHTML XML Attributes
div
p
strong
Amaya
is a Web client that acts both as a browser and a
a href=http://www.w3.org/
acronym title=World Wide Web Consortium
W3C
with the primary purpose of demonstrating new
acronym title=What You See Is What You Get
WYSIWYG
) environment. The current version implements t
acronym title=Hypertext Markup Language
HTML
), Extensible Hypertext Markup Language (
acronym title=Extensible Hypertext Markup Lan

Model/View/Controller

- Model: data structure, no visual representation
- Views: visual representations
- Controllers: user interaction

- Views/controllers update model
- Model tells views that data has changed
- Views redraw themselves

Model/View/Controller