Databases, Networks & the Web

Week 1: Introductions and Workflow

Lecturers:

Term 1 - Sorrel Harriet (s.harriet@gold.ac.uk)

Term 2 – Dan McQuillan (d.mcquillan@gold.ac.uk)

Today's Agenda

- About the module
- Use of Gitlab
- Workflow
- Setting the scene: web application basics

About the module

Course information is available in the following places:

➔ On the department intranet: bit.ly/course-info

➔ In the lab exercises wiki on GitLab: bit.ly/dnw-wiki

Let's take a look...

Questions…

- What will we be learning?
- How much of the module is coursework assessed?
- How are the courseworks weighted?
- What will they involve?
- Which topics are covered in the exam?
- Where do I access lecture slides?
- Where do I access lab exercises?
- Where can I access past papers?

Any others?

What's new?

Not a massive amount…

- ✓ Some slight reordering of content
- ✓ An effort to make coursework freer and more fun
- ✓ Exposure to non-relational DBMS (MongoDB)
- ✓ A bit more depth (for those who need it)

What are we learning in term 1?

Primarily

- ✓ Structured Query Language (SQL)
- ✓ Relational databases (e.g. MySQL)

To a lesser extent...

- ✓ Application design
- ✓ Server-side scripting (e.g. PHP)

And touching on...

- ✓ Non-relational databases (e.g. MongoDB)

Why SQL?

Structured Query Language:

- Still widely used, and likely to be for a long time
- NoSQL databases often use `SQL-like' terminology
- Fundamental language for dealing with data

> *Will the underlying technology change?*
> *Yes, absolutely, but SQL itself will probably remain a really good way to interact with data for a long time to come.*

Admiralwaffles, 2015, reddit.com

But aren't relational databases dead?

Interesting question with no clear answer!

Let's revisit it later in the course...

For now, think of them as the `Do-Re-Me' of data science (a very good place to start)

> *A "data scientist" who isn't proficient in the concepts and execution of relational databases really doesn't possess even the minimum skills to work with complex datasets.*

Randy Zwitch, Data Scientist

GitLab: WT* is it?

First some definitions…

- ➔ GIT - open source distributed version control system
- ➔ GITLAB – application for git repository management
- ➔ SSH (Secure Shell) - an encrypted network protocol which allows secure access to a remote computer

Git: Why use it?

- Essential in most programming professions for…
  - Developing different features in parallel
  - Version management
  - Not losing/duplicating stuff
  - Collaboration
- Make your life much [harder/easier]
- Resistance is futile!

Use of GitLab

Departmental installation: gitlab.doc.gold.ac.uk

You use it for:

- ➔ Accessing course material & assignments (via wiki)
- ➔ Managing master versions of your web projects
- ➔ Submitting coursework

Tutors use it for:

- ➔ Sharing course material & assignments (via wiki)
- ➔ Viewing (and marking) your web projects
- ➔ Checking your commit history

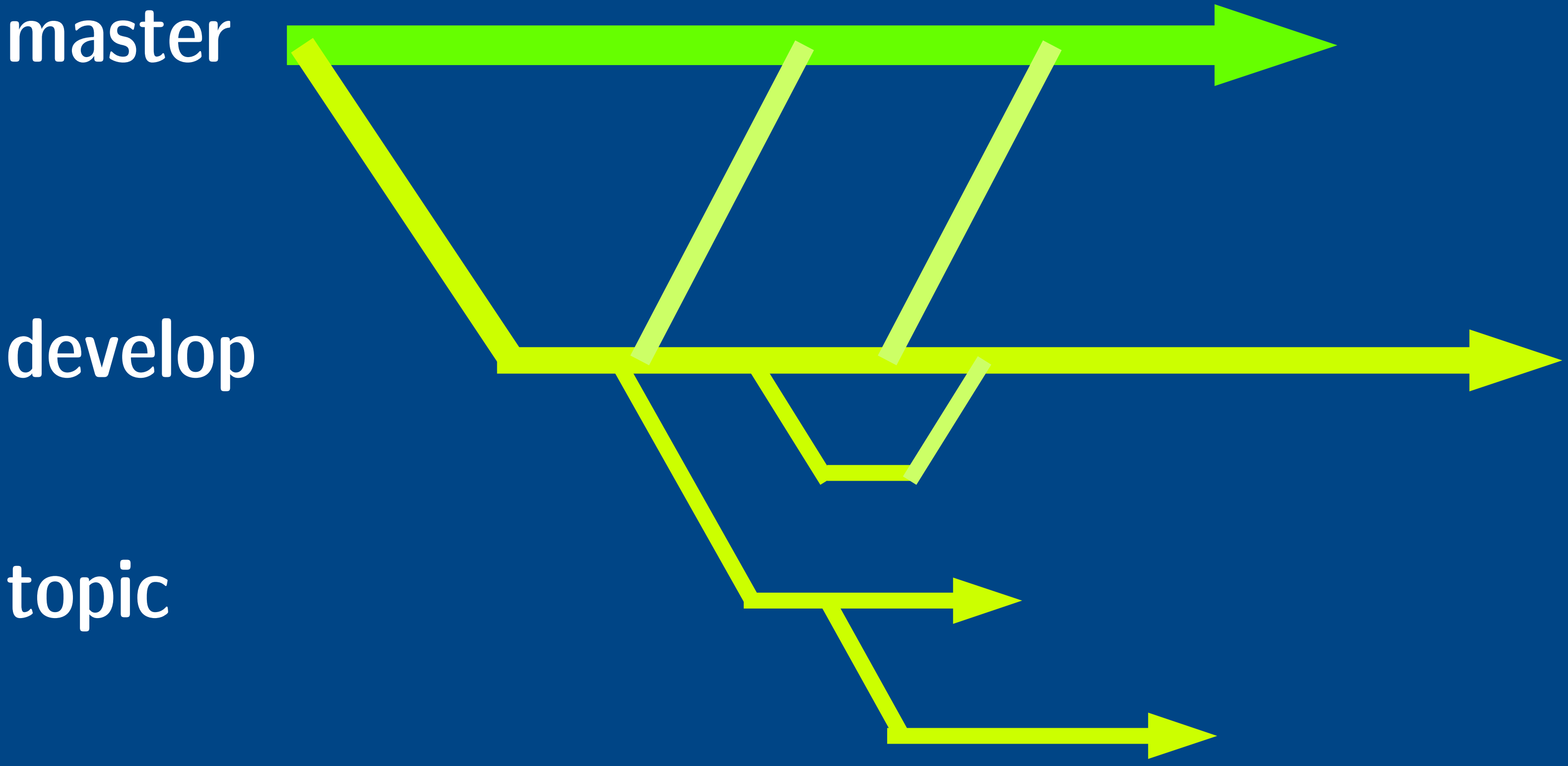
Git: How does it work?

Simple explanation:

- ➔ Project owner initialises a git `repo' (repository) for a code project
- ➔ A remote version of the repo may be hosted on the internet
- ➔ Permitted users can `clone' the remote repo to work on it
- ➔ When a user works on a file, they tell Git to track it
- ➔ When they finish working on it, they `commit' their changes
- ➔ Git records details about the changes. Each commit (or revision) is given a unique reference (SHA-1 hash)
- ➔ User may `push' their revisions back to the origin (the remote repo that they originally cloned)

In depth explanation: https://git-scm.com/

Git: How does it work?



master

develop

topic

GitLab: How to get started

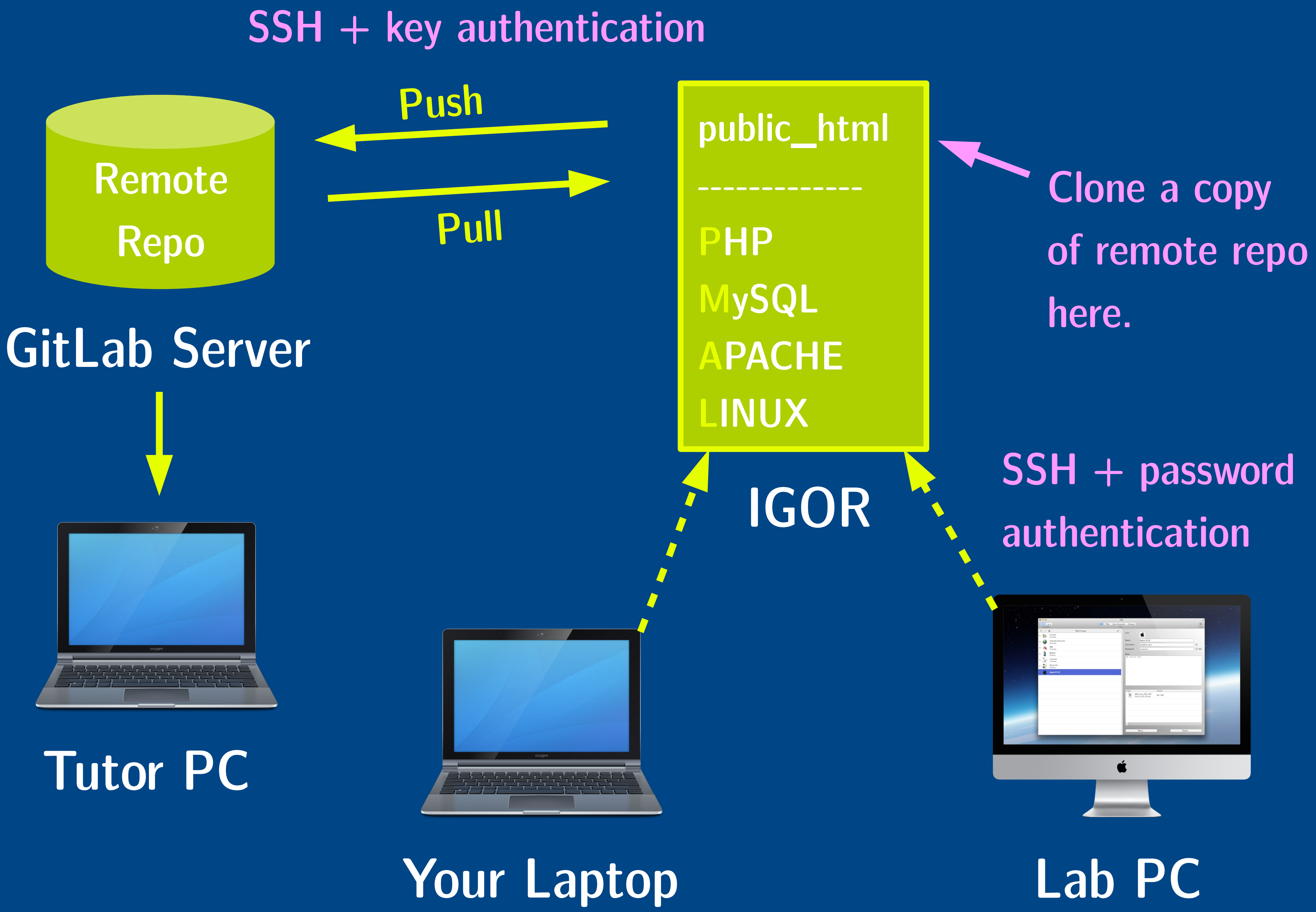
- ➔ SSH to IGOR Command-Line Interface (CLI)
- ➔ Generate an SSH key pair
- ➔ Share your public key with GitLab
- ➔ Create a remote repo on GitLab
- ➔ Clone the repo on IGOR
- ➔ Start developing!

Git: How do I use it?

Essential git commands:

- ➔ Add – to start tracking a file
- ➔ Commit – to commit a revision
- ➔ Pull – to get any new revisions
  - ➔ Merge any changes (if applicable)
  - ➔ Commit the result of a merge (if applicable)
  - ➔ Pull again (for good measure)
- ➔ Push – to push your revisions to the remote

Example Workflow (recommended)



SSH + key authentication

Push

Pull

Remote Repo

GitLab Server

public_html

PHP

MySQL

APACHE

LINUX

IGOR

Clone a copy of remote repo here.

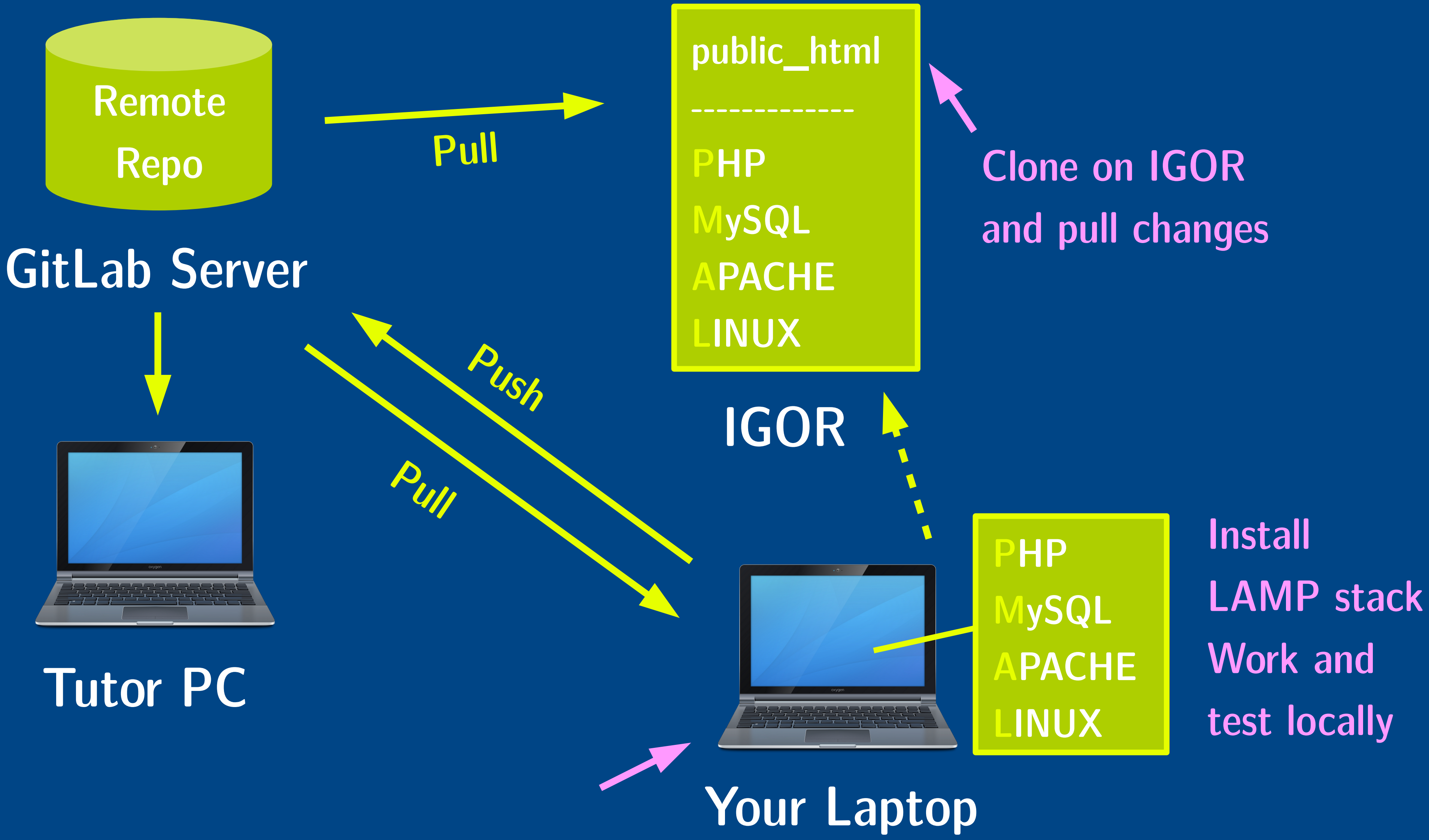
SSH + password authentication

Tutor PC

Your Laptop

Lab PC

Alternative Workflow (caution advised!)



Further Guidance & Support

- Accessing IGOR: bit.ly/access-igor
- Gold Overflow: bit.ly/goldoverflow
- Lab exercises
- The notes section of these slides (posted on the VLE)
- RTFM!
- Do some research! (then share on Gold Overflow...)
- Book me! sorrelharriet.youcanbook.me

BREAK!

- 7 mins
- Don't be late! :-)

And now we begin our foray in databases…

…somewhere a bit different!

What is a web application?

What is a web application?

“A client-server software application in which the user interface runs in a web browser” (Wikipedia)

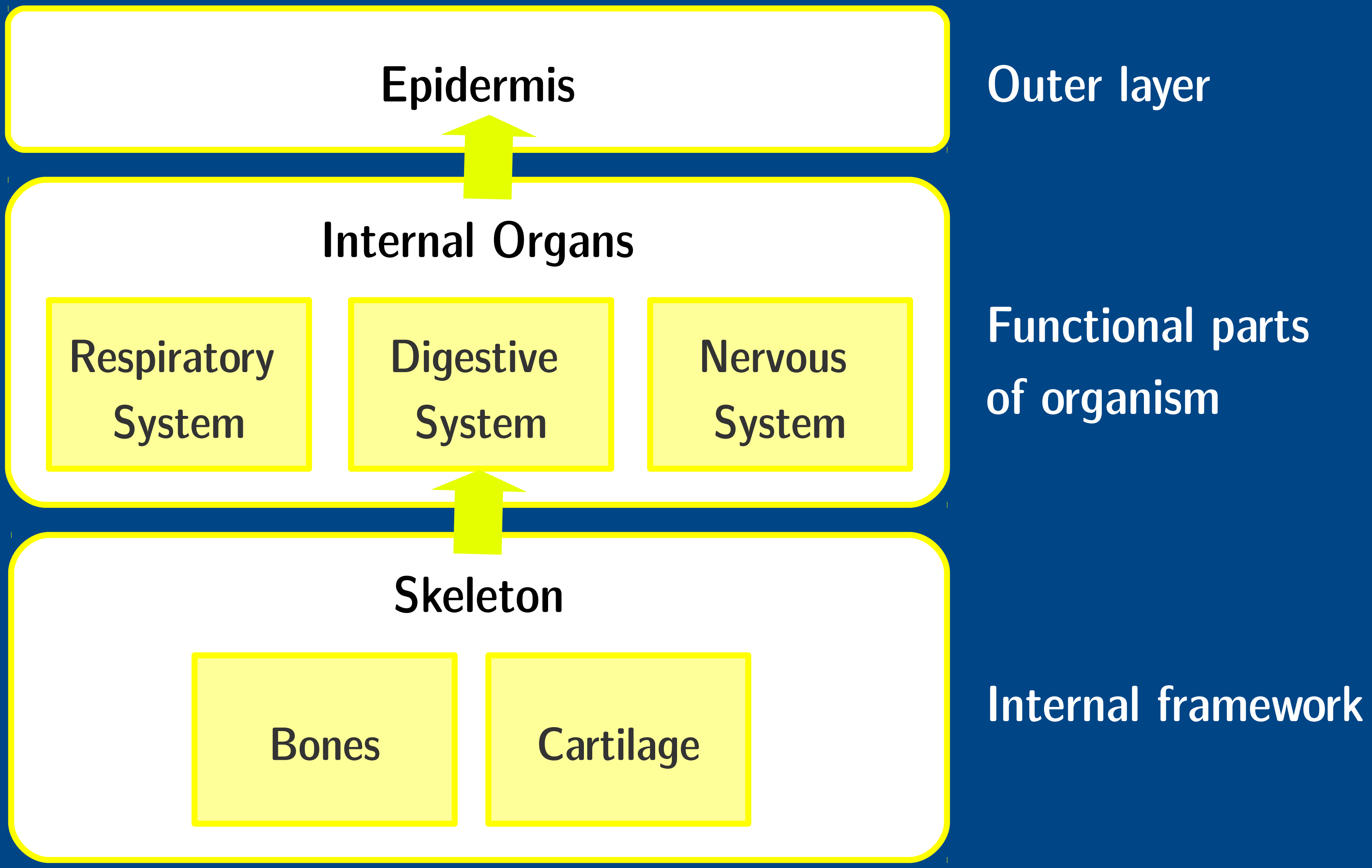
“A computer program which allows a user to submit and retrieve data to/from a database over the internet using their preferred browser” (www.acunetix.com)

“An application in which all or some parts of the software are downloaded from the Web each time it is run. It may refer to browser-based, 'rich client' and mobile web apps.” (PC Magazine)

Thinking Points:

- ➔ Is there a difference between a website and a web application?
- ➔ Can a web app exist without the internet?
- ➔ Does it matter what browser you use to access them?
- ➔ Do parts of the software get downloaded, or just the data?

Here is a layer diagram:



Epidermis

Outer layer

Internal Organs

Respiratory System

Digestive System

Nervous System

Functional parts of organism

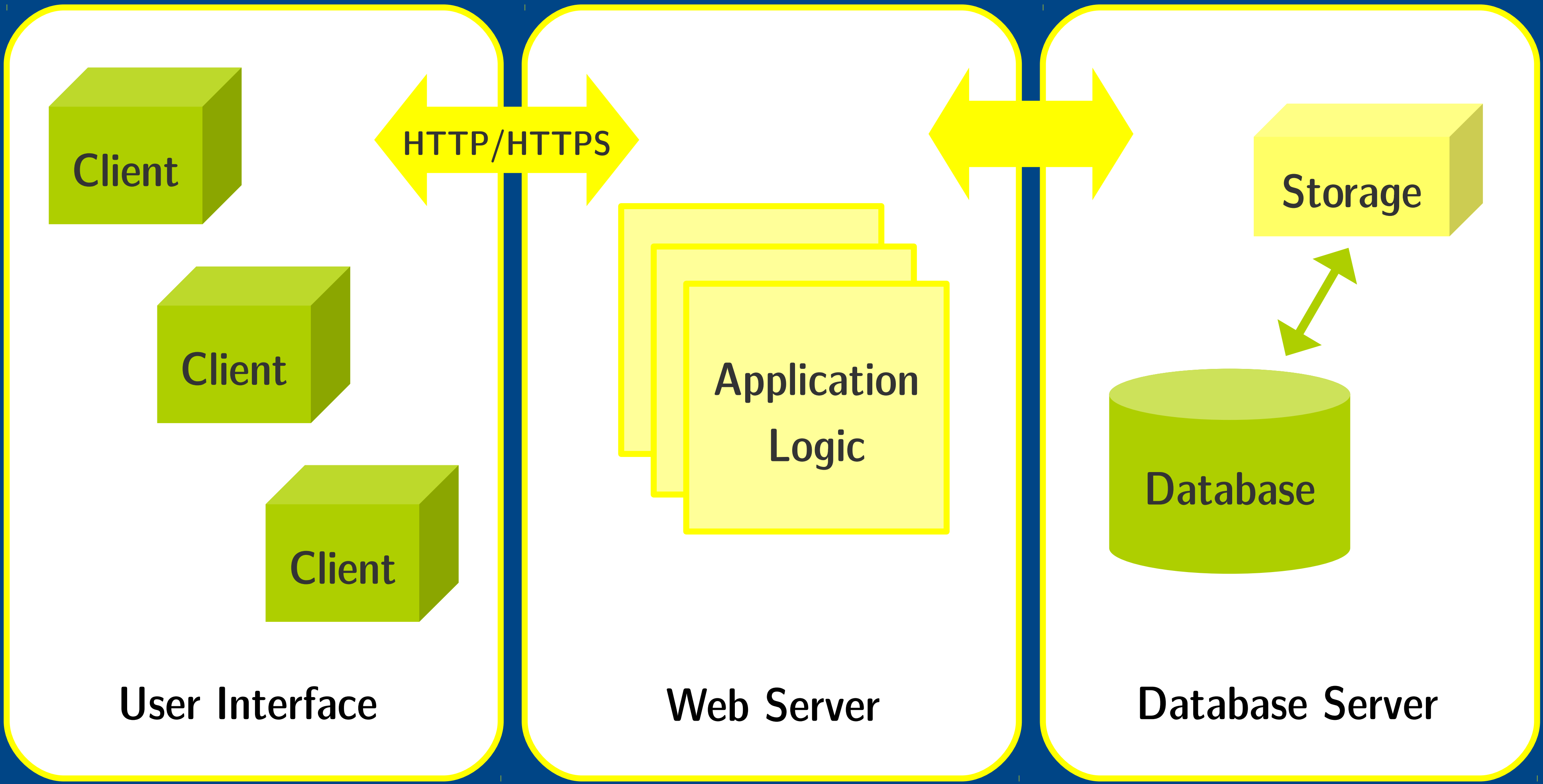
Skeleton

Bones

Cartilage

Internal framework

3 Tier Application Model

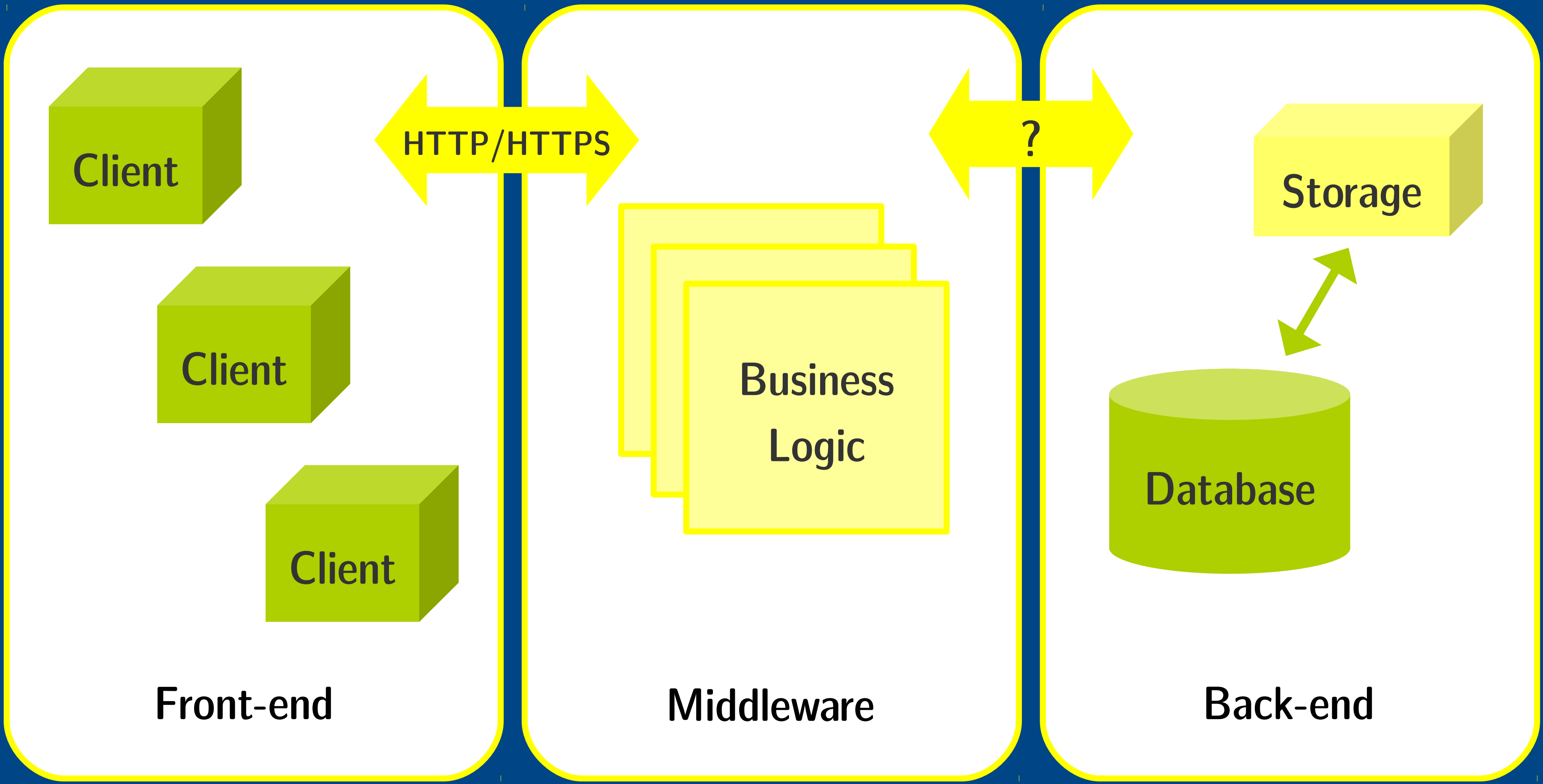


Presentation Tier

Application Tier

Data Tier

3 Tier Application Model

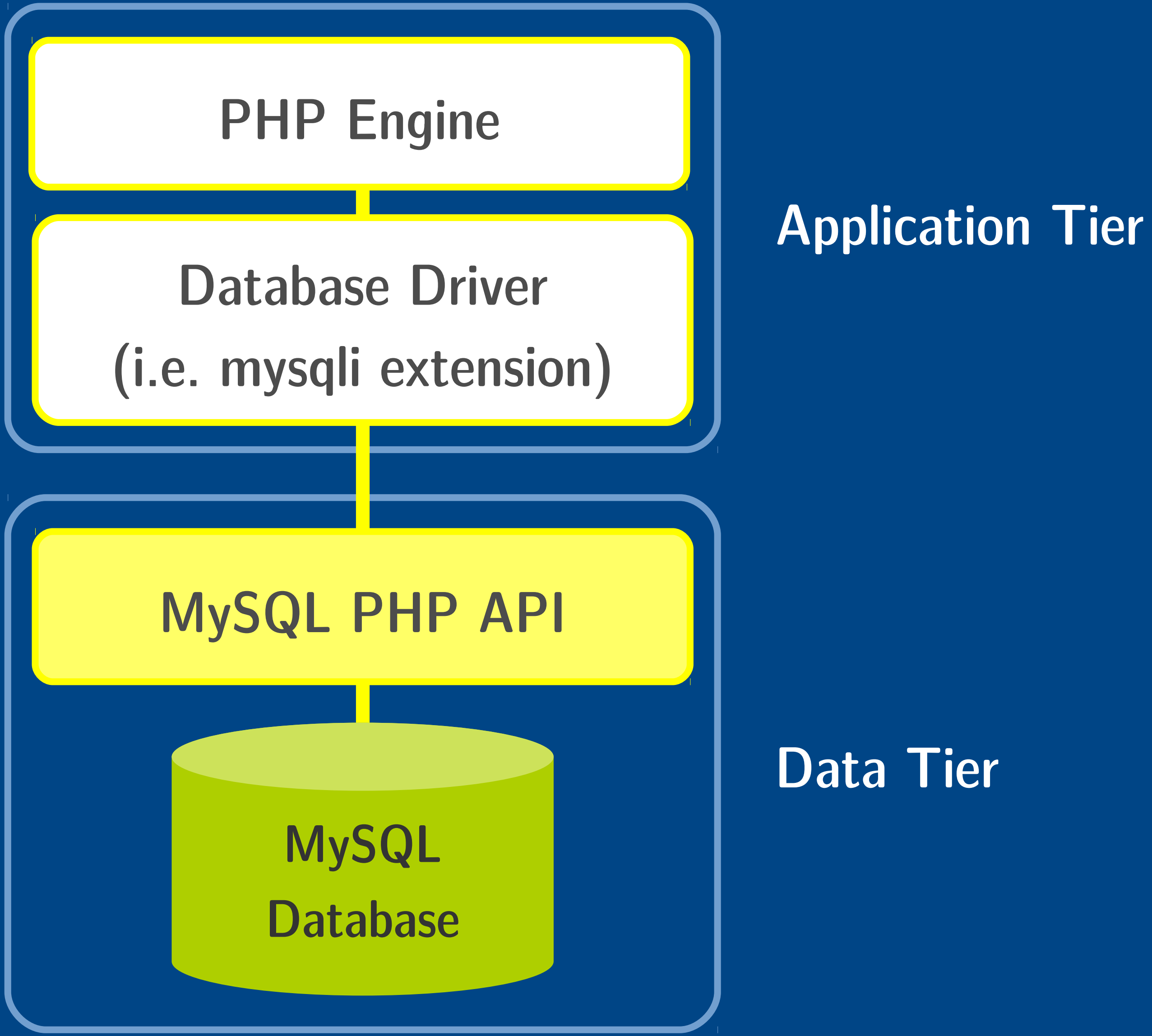


Presentation Tier

Application Tier

Data Tier

Our Solution Stack



PHP Engine

Database Driver
(i.e. mysqli extension)

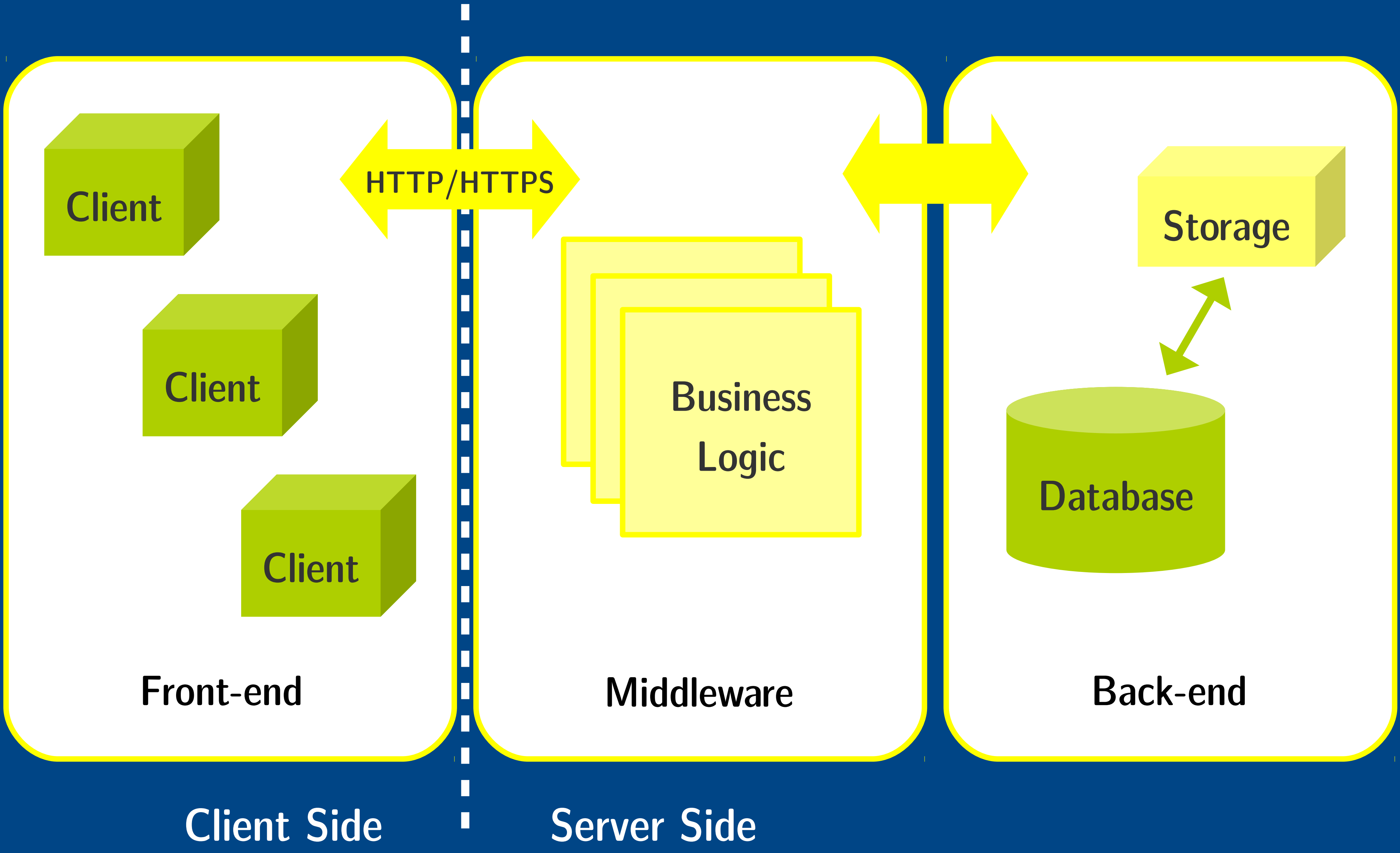
MySQL PHP API

MySQL
Database

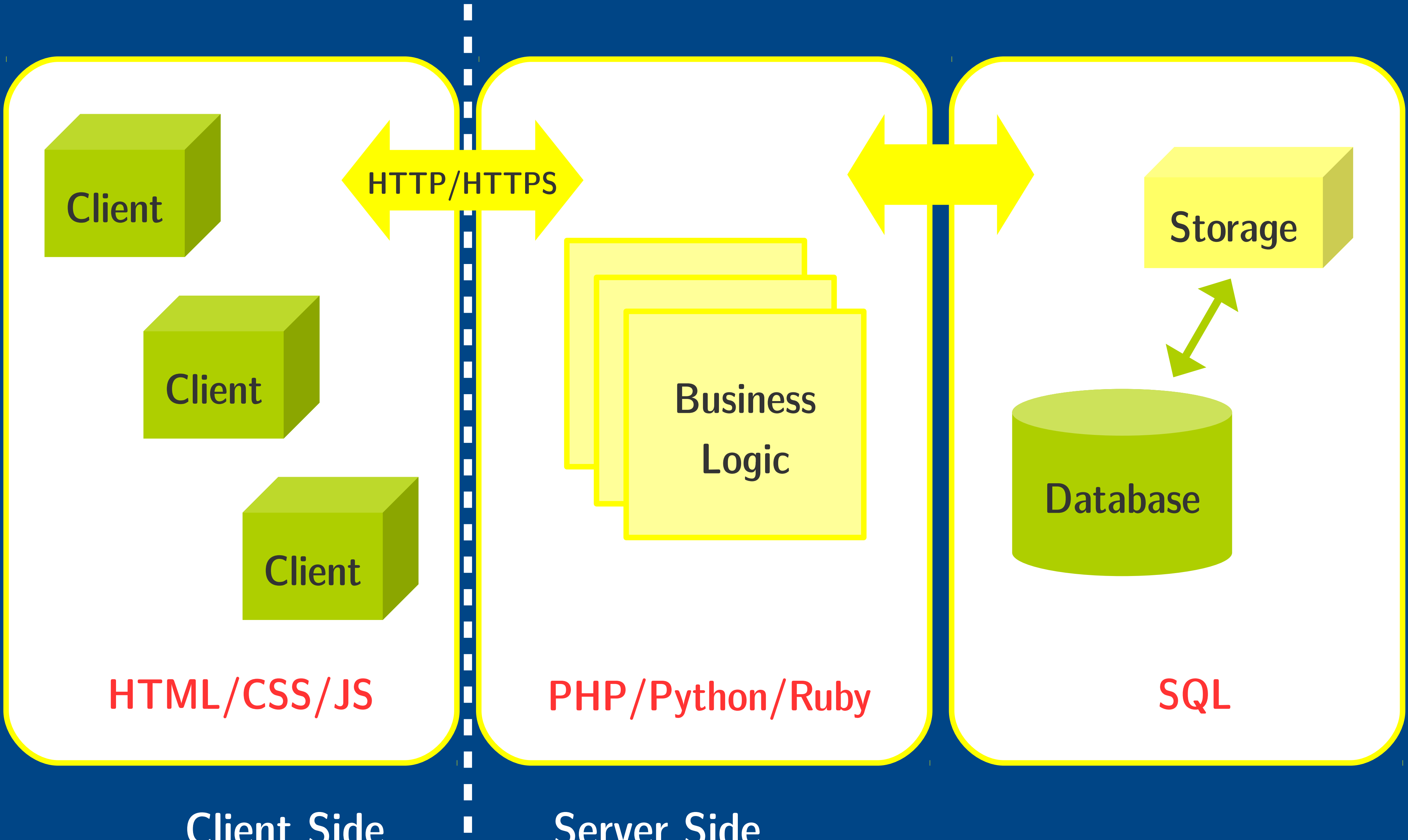
Application Tier

Data Tier

3 Tier Application Model

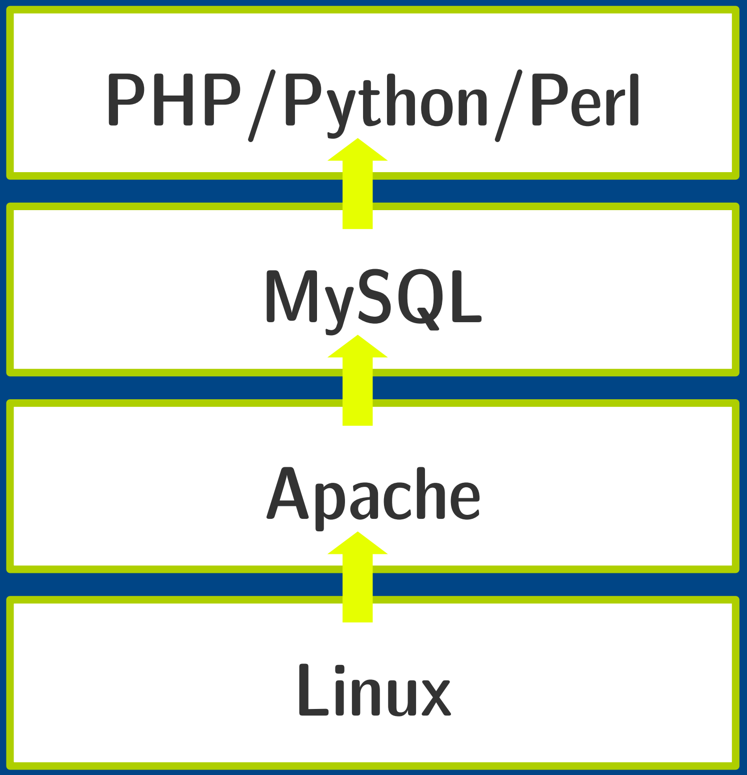


3 Tier Application Model



LAMP Stack

Generic software stack model used for developing and serving web applications



PHP/Python/Perl

MySQL

Apache

Linux

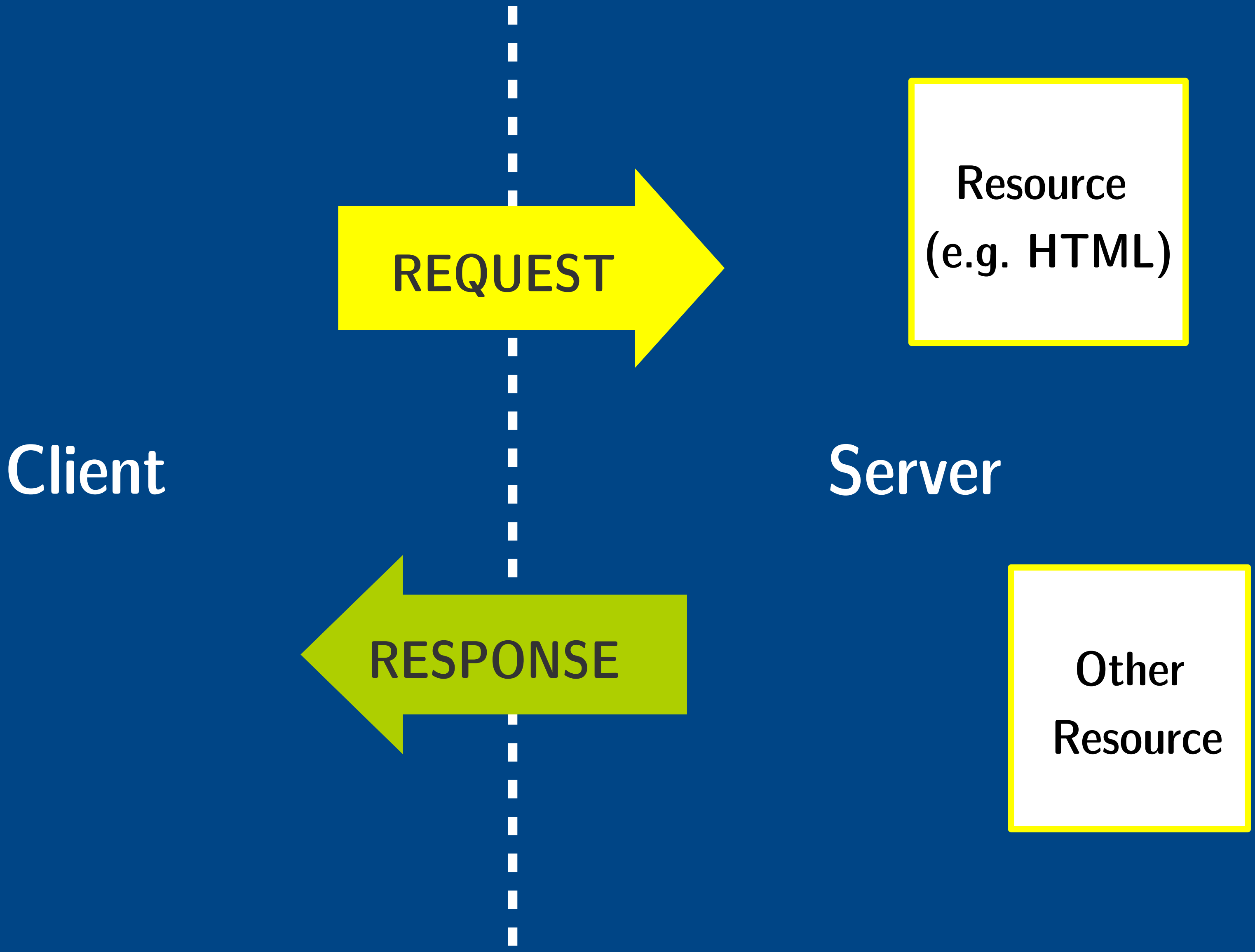
Scripts to handle user interaction

DBMS

Web server (handles HTTP)

Base/OS layer

HyperText Transfer Protocol



REQUEST

Resource (e.g. HTML)

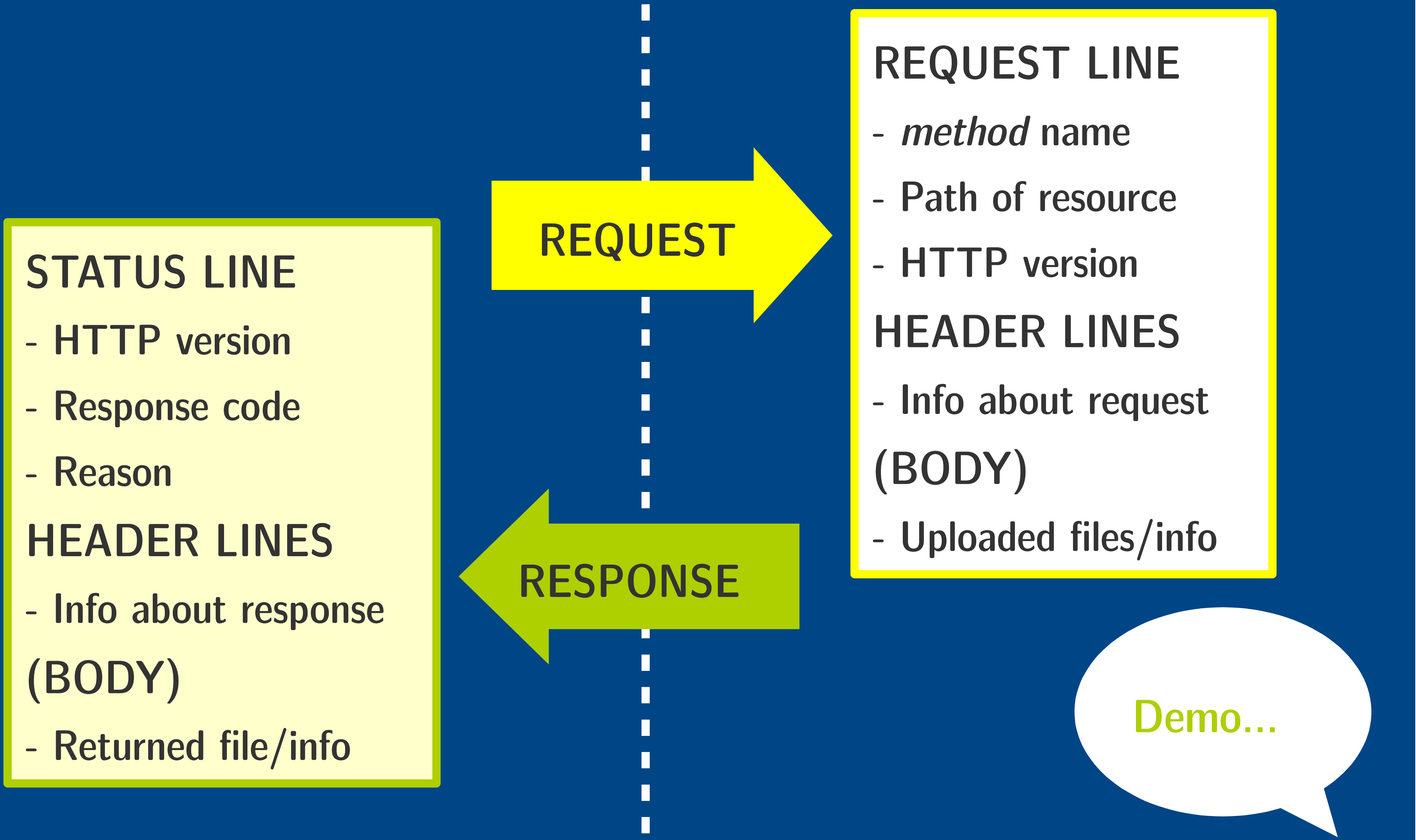
Client

Server

RESPONSE

Other Resource

HyperText Transfer Protocol



REQUEST

REQUEST LINE

- *method* name
- Path of resource
- HTTP version

HEADER LINES

- Info about request

(BODY)

- Uploaded files/info

RESPONSE

STATUS LINE

- HTTP version
- Response code
- Reason

HEADER LINES

- Info about response

(BODY)

- Returned file/info

Demo...

HTTP Request Methods

GET

Requests data

POST

Submits data to be processed (often from an HTML form)

PUT

DELETE

HEAD

CONNECT

OPTIONS

TRACE

More on this later!

Are you seeing it yet??

Coming up in Lab 1 (6/7 Oct)...

Setting up your development environment

➔ creating an SSH key-pair on GitLab

➔ creating a project repo on GitLab

➔ cloning your repo to IGOR with Git

➔ mounting IGOR (optional)

➔ exploring the LAMP stack

➔ cloning a test app to IGOR

Databases, Networks & the Web

Week 1: Introductions and Workflow

Lecturers:

Term 1 - Sorrel Harriet (s.harriet@gold.ac.uk)

Term 2 – Dan McQuillan (d.mcquillan@gold.ac.uk)

Today's Agenda

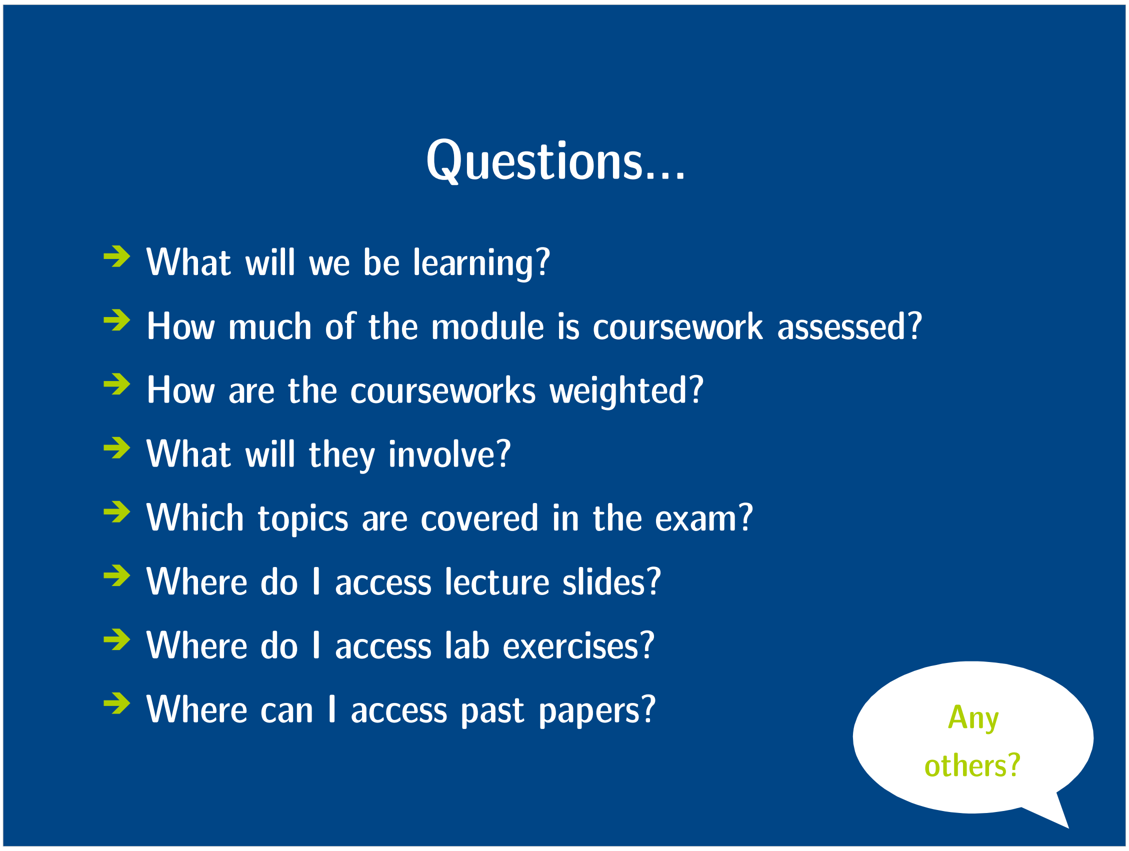
- ➔ About the module
- ➔ Use of Gitlab
- ➔ Workflow
- ➔ Setting the scene: web application basics

About the module

Course information is available in the following places:

- ➔ On the department intranet: bit.ly/course-info
- ➔ In the lab exercises wiki on GitLab: bit.ly/dnw-wiki

Let's take
a look...

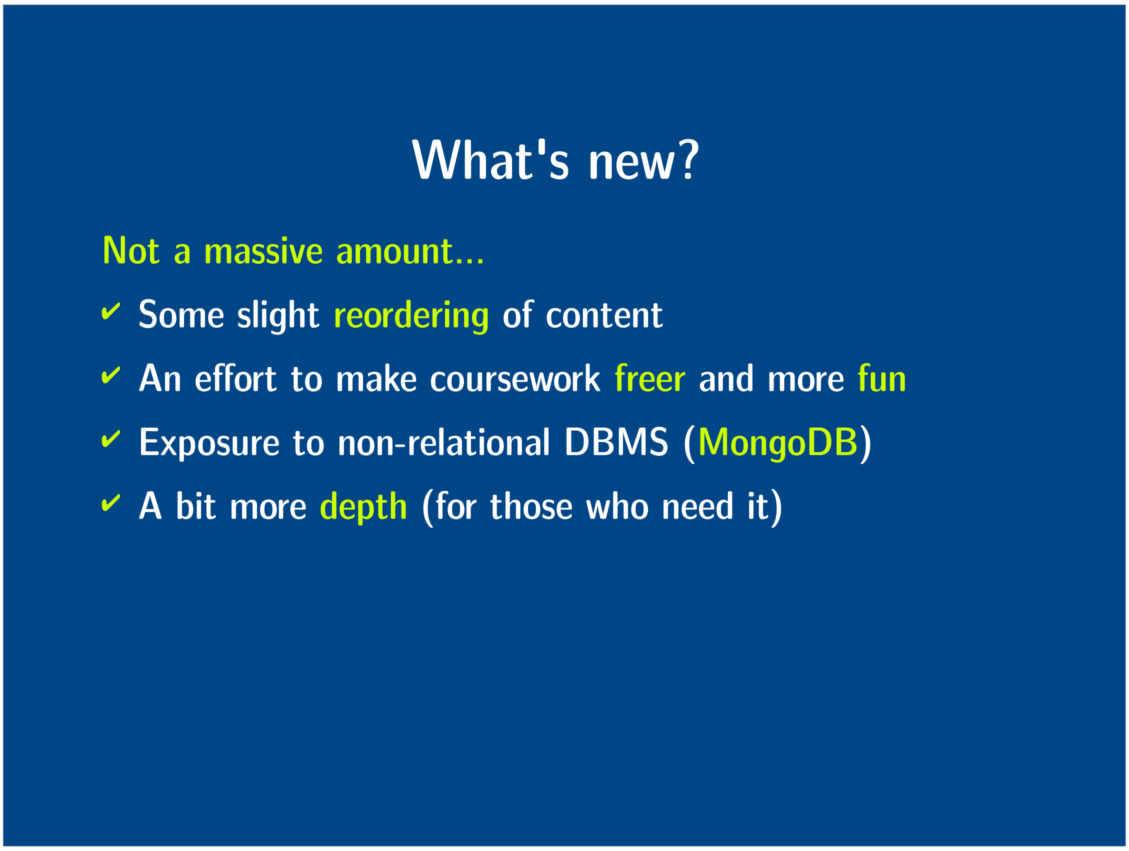


Questions…

- What will we be learning?
- How much of the module is coursework assessed?
- How are the courseworks weighted?
- What will they involve?
- Which topics are covered in the exam?
- Where do I access lecture slides?
- Where do I access lab exercises?
- Where can I access past papers?

Any others?

Familiarise yourself with the Wiki and VLE, and make sure you can answer these questions!



What's new?

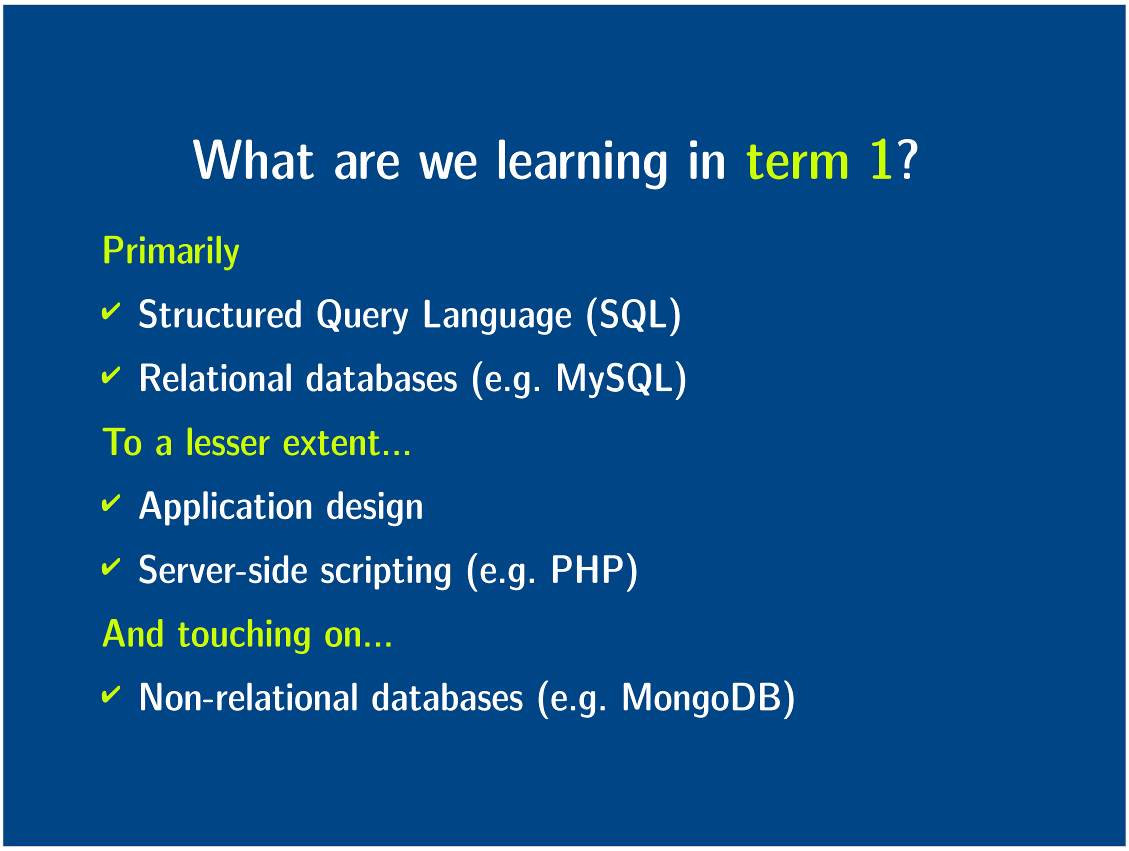
Not a massive amount...

- ✓ Some slight reordering of content
- ✓ An effort to make coursework freer and more fun
- ✓ Exposure to non-relational DBMS (MongoDB)
- ✓ A bit more depth (for those who need it)

After feedback from last year's cohort, we had a long hard think, and decided NOT to make major changes to the technologies used on the course (I'll justify those non-changes in a bit).

However, we are trying to make the lab work a bit **looser** and hopefully more **fun** for those in need of excitement! (and what could be more exciting than databases I wonder...)

You do need to make an effort though to engage with the material. You might not find it *all* scintilating, but you will hopefully find parts of it enjoyable and rewarding. **You get what you give!**



What are we learning in term 1?

Primarily

- ✔ Structured Query Language (SQL)
- ✔ Relational databases (e.g. MySQL)

To a lesser extent...

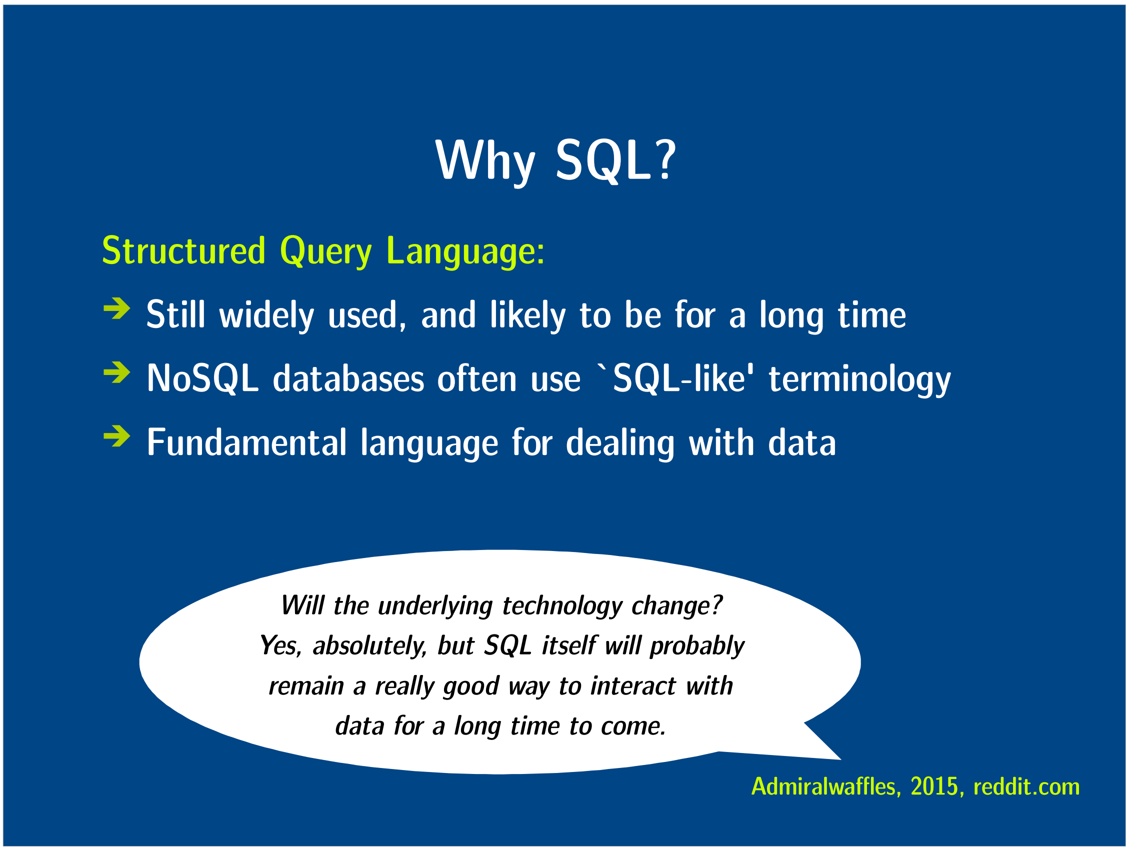
- ✔ Application design
- ✔ Server-side scripting (e.g. PHP)

And touching on...

- ✔ Non-relational databases (e.g. MongoDB)

Our main aim in term 1 is to develop a solid understanding of relational databases and SQL, in the context of developing dynamic web applications. This will mean having a pretty good grasp of what web applications are and how they work, although you won't go into as much detail on the implementation side of things until term 2.

Although we're using the Database Management System MySQL, and the server-side scripting language PHP, these are just tools. Its the underlying concepts we're most interested in, and in that sense, the software and language is somewhat bit arbitrary. However, PHP is great at allowing total beginners to develop whole applications from the bottom up, therefore getting to see `the whole picture', without the abstraction of a framework, or the complication of configuring Python servers etc.



Why SQL?

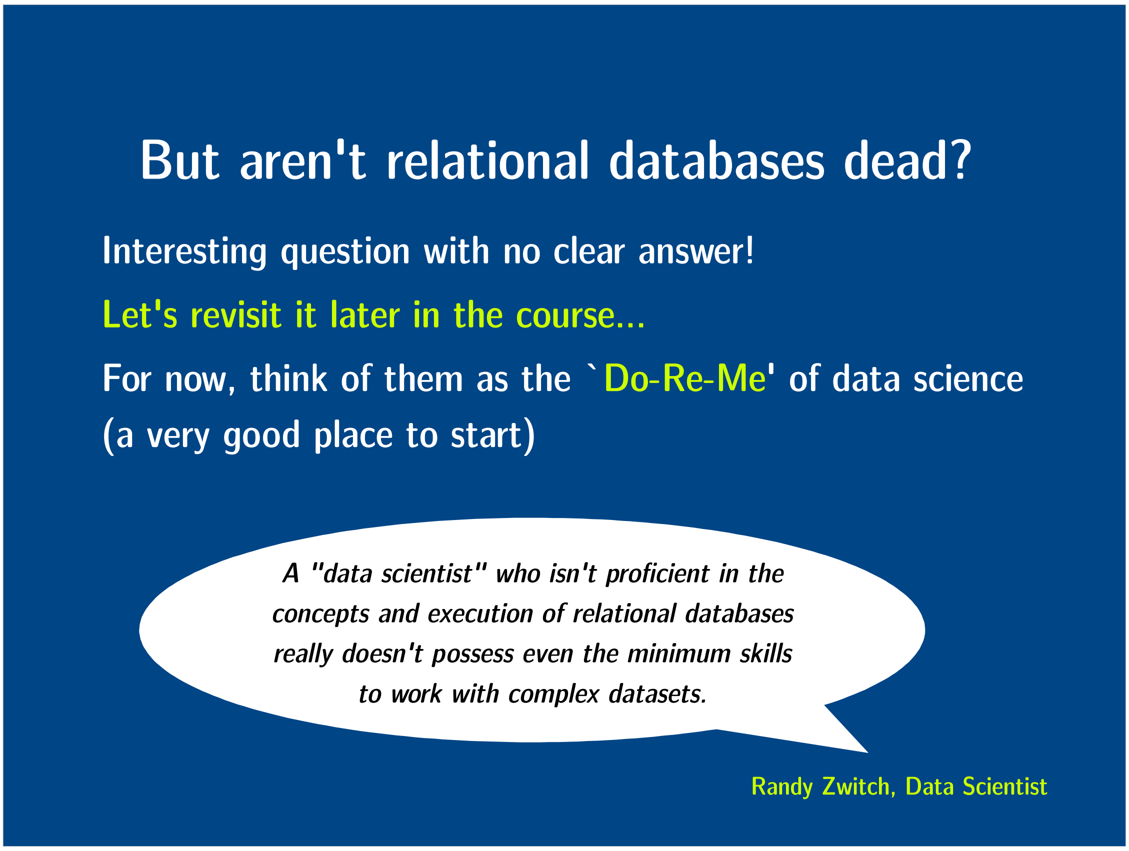
Structured Query Language:

- ➔ Still widely used, and likely to be for a long time
- ➔ NoSQL databases often use `SQL-like' terminology
- ➔ Fundamental language for dealing with data

Will the underlying technology change? Yes, absolutely, but SQL itself will probably remain a really good way to interact with data for a long time to come.

Admiralwaffles, 2015, reddit.com

Our main focus in term 1 will be on developing a solid understanding of basic SQL. SQL is a language for communicating with databases, typically of the relational type. However, as admiralwaffles quite rightly points out, even though the database technologies themselves are changing, SQL syntax is still prevalent.



But aren't relational databases dead?

Interesting question with no clear answer!

Let's revisit it later in the course...

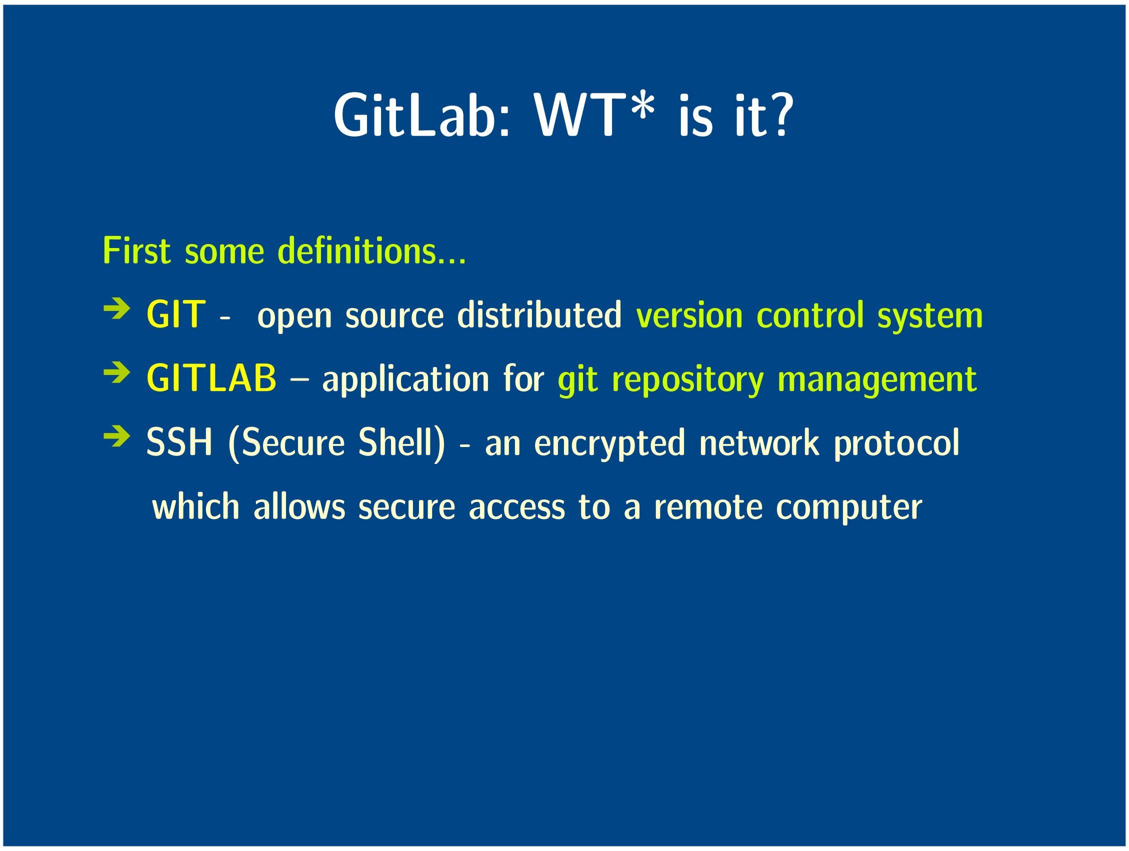
For now, think of them as the `Do-Re-Me' of data science (a very good place to start)

> *A "data scientist" who isn't proficient in the concepts and execution of relational databases really doesn't possess even the minimum skills to work with complex datasets.*

Randy Zwitch, Data Scientist

I hope you will keep this question in the back of your mind as we progress through term 1, and hopefully we can have a good debate about it later on!

However, I would say that, regardless of life expectancy, relational databases really are a good place to start learning the fundamental principles of data engineering. And as the old saying goes, `your database is only strong when you know its weaknesses'...



GitLab: WT* is it?

First some definitions…

➔ GIT - open source distributed version control system

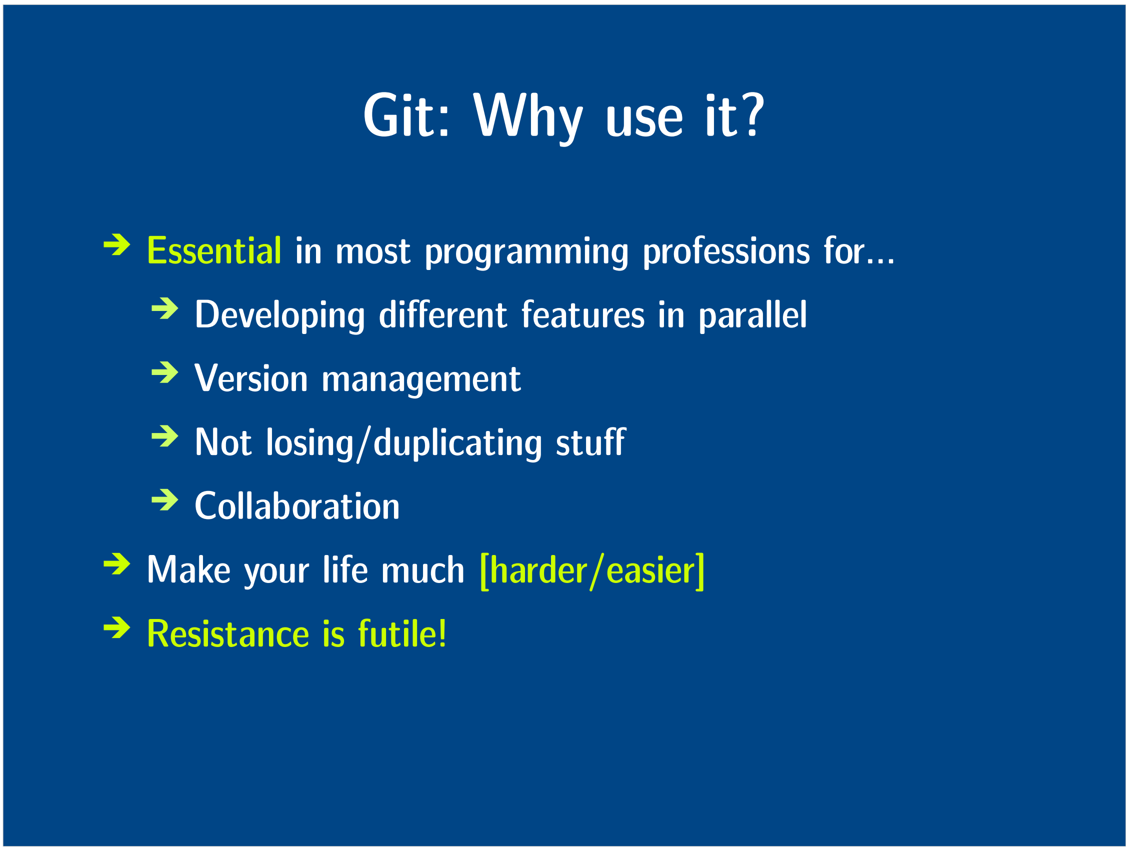
➔ GITLAB – application for git repository management

➔ SSH (Secure Shell) - an encrypted network protocol which allows secure access to a remote computer

The technology underlying GitLab is Git, a version control system which operates via a Command Line Interface (CLI).

GitLab is essentially a `pretty face' for Git, allowing browser-based access and a nice GUI.

We'll come to SSH later...it's something you need when using git

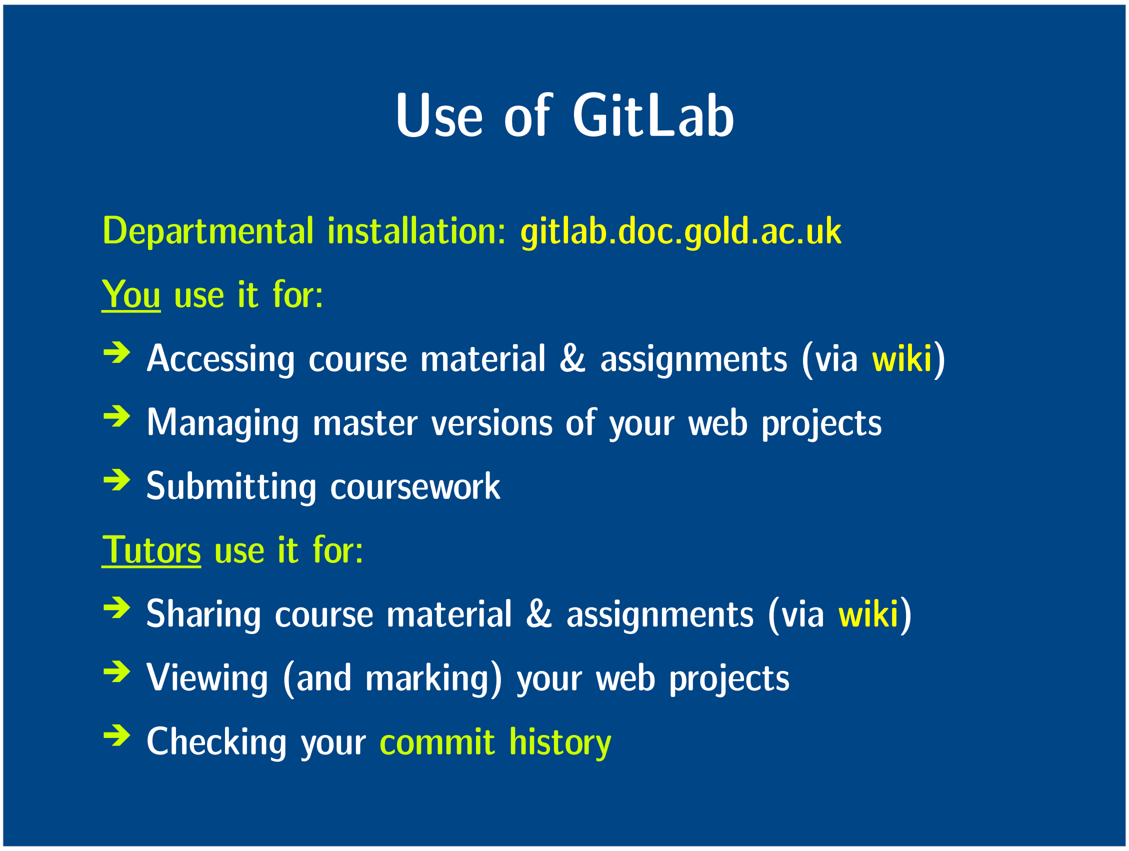


Git: Why use it?

- Essential in most programming professions for…
  - Developing different features in parallel
  - Version management
  - Not losing/duplicating stuff
  - Collaboration
- Make your life much [harder/easier]
- Resistance is futile!

Whatever computing profession you go into, if it involves code development of any kind, chances are you will be required to use a version management system. Git is open-source and widely used (others include subversion and mercurial, but all relatively similar).

Yes, Git can be a headache, but resisting it is not going to help you, now, or in your future career.



Use of GitLab

Departmental installation: gitlab.doc.gold.ac.uk

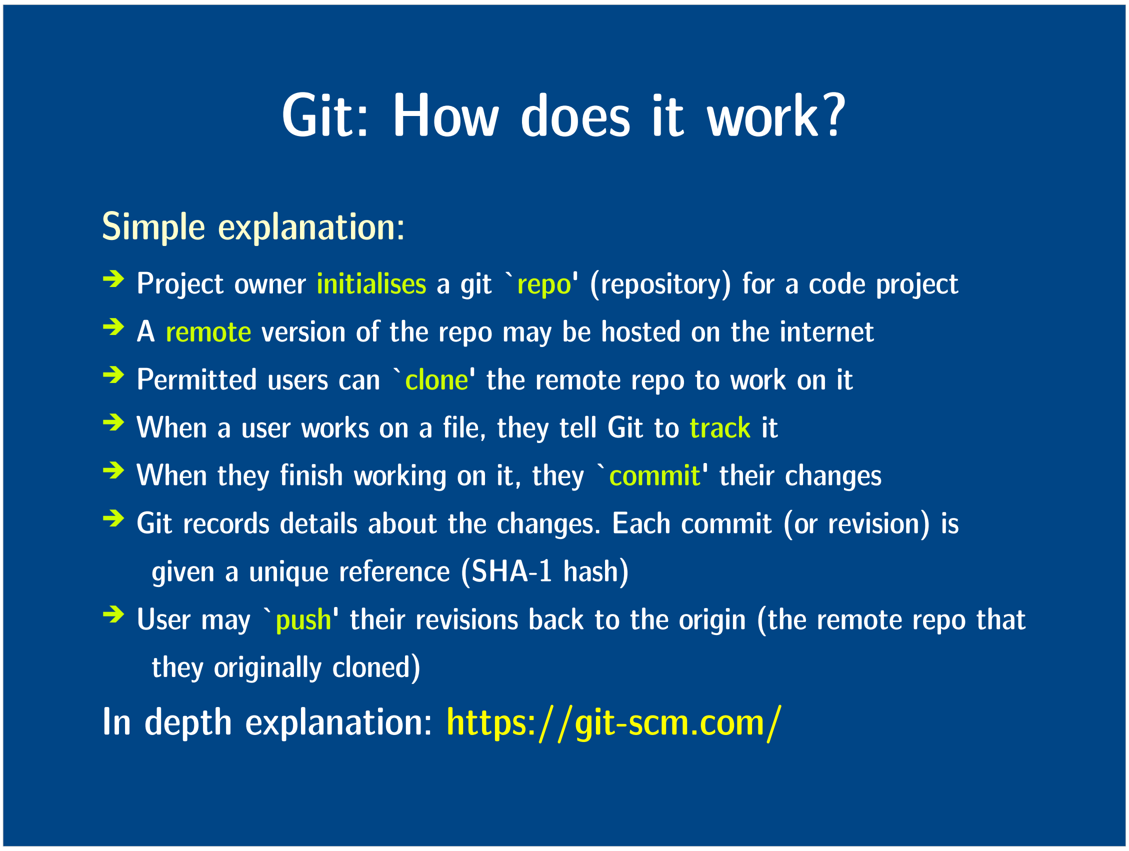
You use it for:

- ➔ Accessing course material & assignments (via wiki)
- ➔ Managing master versions of your web projects
- ➔ Submitting coursework

Tutors use it for:

- ➔ Sharing course material & assignments (via wiki)
- ➔ Viewing (and marking) your web projects
- ➔ Checking your commit history

Your GitLab account is already set up, and you should use you normal username and password to access it.



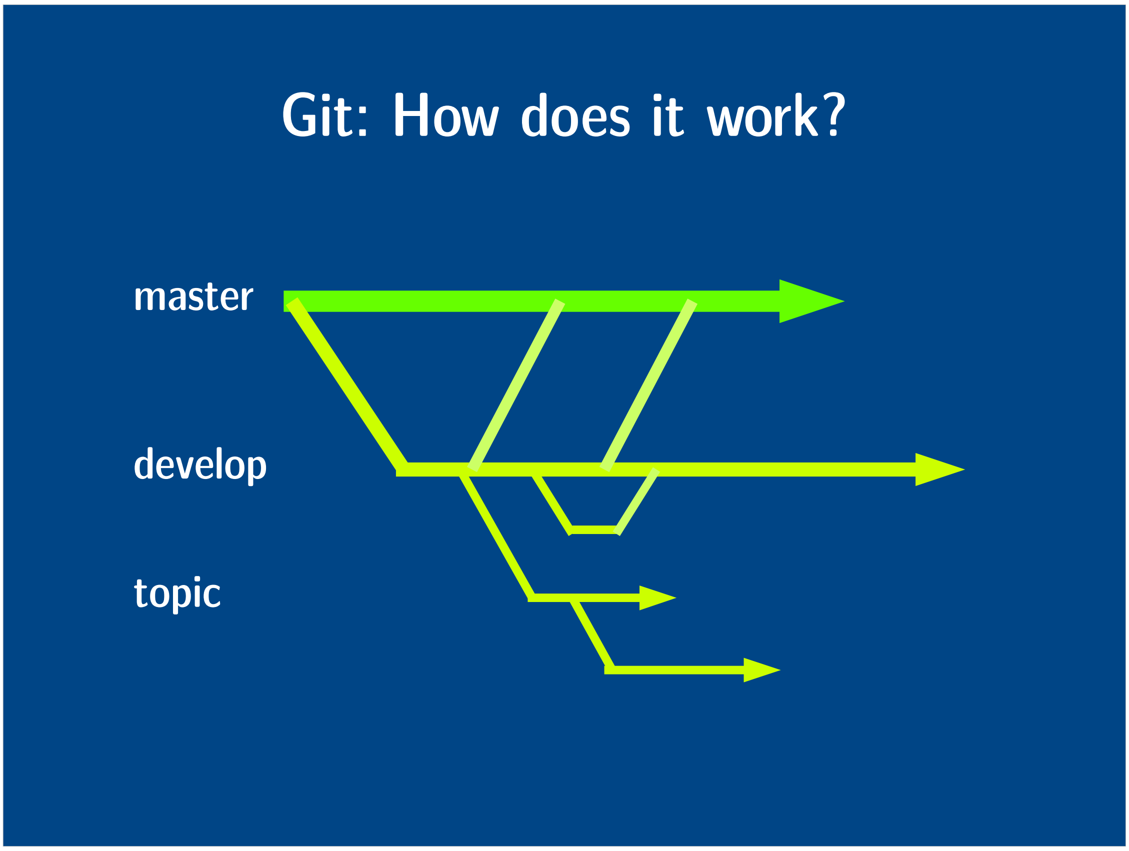
Git: How does it work?

Simple explanation:

- ➔ Project owner initialises a git `repo' (repository) for a code project
- ➔ A remote version of the repo may be hosted on the internet
- ➔ Permitted users can `clone' the remote repo to work on it
- ➔ When a user works on a file, they tell Git to track it
- ➔ When they finish working on it, they `commit' their changes
- ➔ Git records details about the changes. Each commit (or revision) is given a unique reference (SHA-1 hash)
- ➔ User may `push' their revisions back to the origin (the remote repo that they originally cloned)

In depth explanation: https://git-scm.com/

It's a bit more complicated than this...if multiple users are working on the same project, or a single user is working on a project from multiple locations, they must remember to PULL any changes to the remote repo, before they attempt to PUSH to it...but let's assume for now that you'll develop your projects independently, and in a single location.



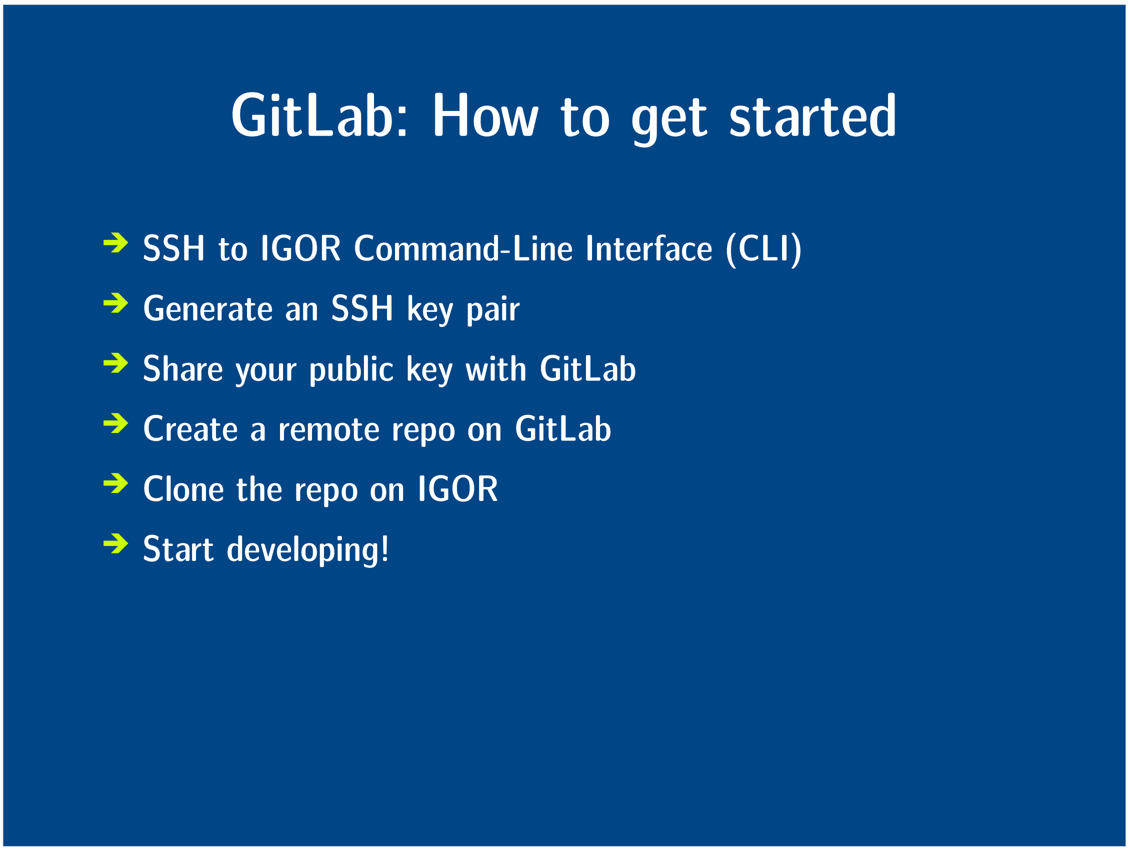
Git: How does it work?

It's also possible to BRANCH and MERGE in git.

Although, conceptually, it's easier to think of a branch as a copy of a repo, it's not a copy exactly...Git is still just keeping track of lots of revisions and piecing the `versions' together from this information.

Once a user is happy with a new feature they've developed on a particular branch, they can MERGE the branch back with the main development branch or master, so that everyone else gets to see their changes.

When merging, CONFLICTS can occur if the same code has been modified in different places. Sometime git can solve the conflicts itself, but other times it needs the developer to do this manually.



GitLab: How to get started

- SSH to IGOR Command-Line Interface (CLI)
- Generate an SSH key pair
- Share your public key with GitLab
- Create a remote repo on GitLab
- Clone the repo on IGOR
- Start developing!

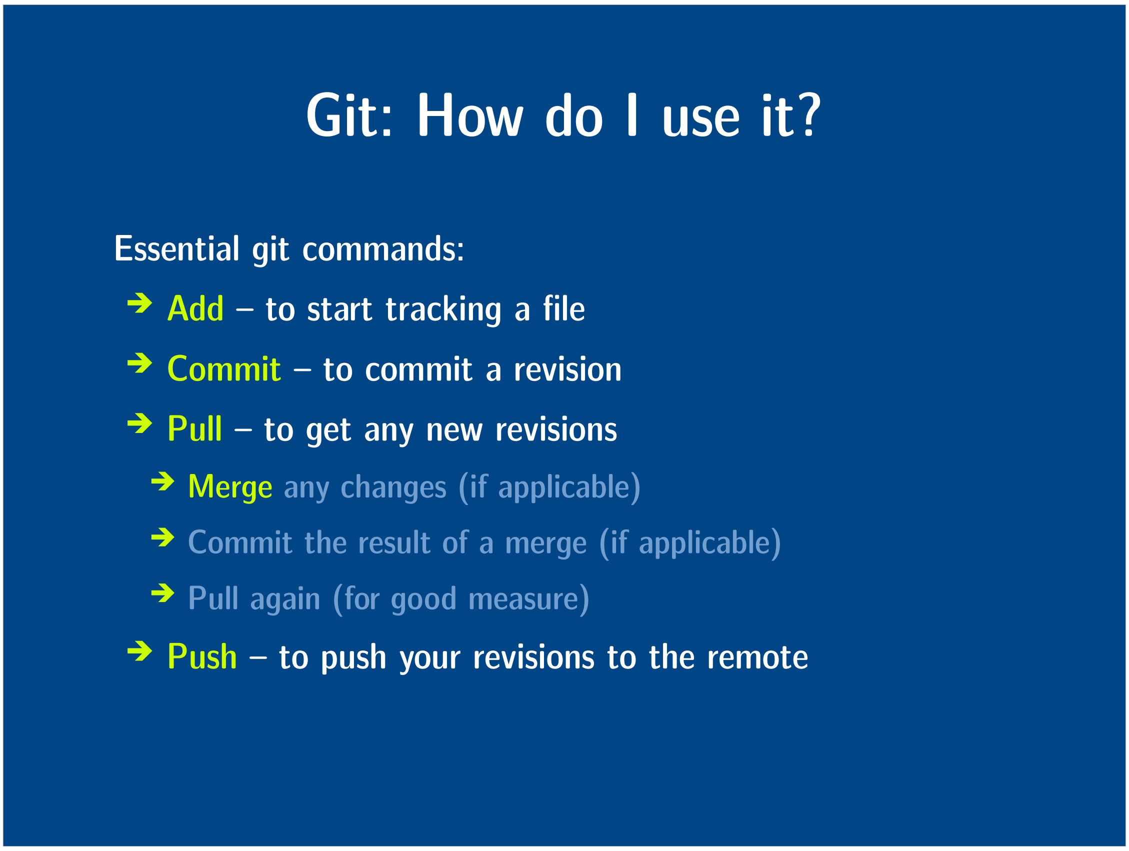
In **Lab 1** you will generate a public/private key pair to enable SSH key authentication by GitLab.

The private key is held on the machine that created it (IGOR), and the public key is shared with the remote machine (GitLab).

Now, each time you attempt to communicate with the remote repo using git*, GitLab will recognise you by your SSH public key (git uses SSH to connect to remote repositories).

*We recommend using GitLab's GUI for `one time only' configurations, such as adding a member to a project, but for day-to-day development, use git CLI.

Note: when accessing IGOR with SSH, you use password authentication, which is less secure than key authentication!



Git: How do I use it?

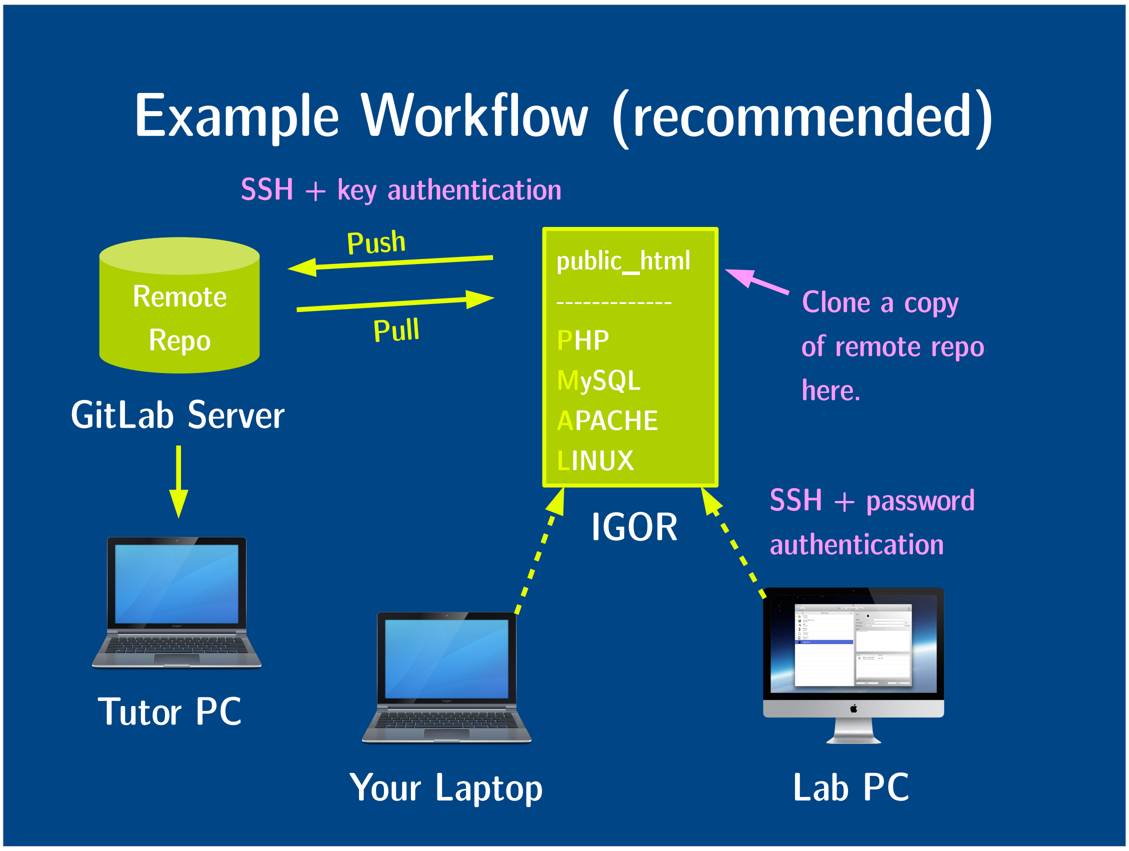
Essential git commands:

- ➔ Add – to start tracking a file
- ➔ Commit – to commit a revision
- ➔ Pull – to get any new revisions
  - ➔ Merge any changes (if applicable)
  - ➔ Commit the result of a merge (if applicable)
  - ➔ Pull again (for good measure)
- ➔ Push – to push your revisions to the remote

Of course, it can get much more complicated than this...but these are the basic commands you *should** be able to get by with on this module.

*Unless you do something odd...in which case you may need to investigate others!

You'll find instructions on how to use these commands in the lab exercises, but it might help to supplement these with your own reading.
For example: http://rogerdudler.github.io/git-guide/



Example Workflow (recommended)

SSH + key authentication

Remote Repo

Push

Pull

GitLab Server

public_html

PHP
MySQL
APACHE
LINUX

Clone a copy of remote repo here.

IGOR

SSH + password authentication

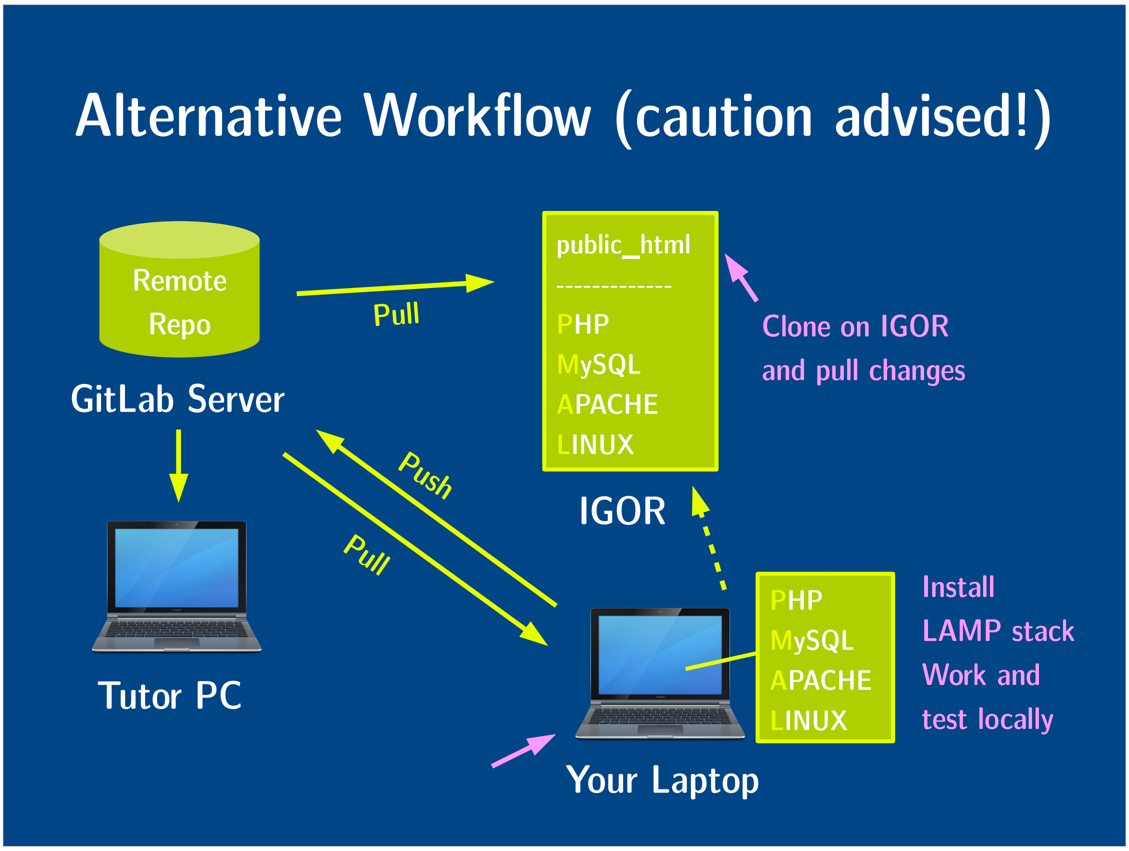
Tutor PC

Your Laptop

Lab PC

In this option, there is no need to install any extra software on your own machine (assuming it is Unix based OS*). You can simply connect to IGOR (or mount the IGOR file system on your computer), and work from there. Any pushing/pulling will be between IGOR and GitLab.

*If you have Windows, you can install Putty in order to connect to IGOR via SSH, or you can follow an alternative procedure to mount the IGOR file system



Alternative Workflow (caution advised!)

Remote Repo

GitLab Server

Pull

public_html

PHP
MySQL
APACHE
LINUX

Clone on IGOR
and pull changes

IGOR

Push

Pull

Tutor PC

PHP
MySQL
APACHE
LINUX

Install
LAMP stack
Work and
test locally

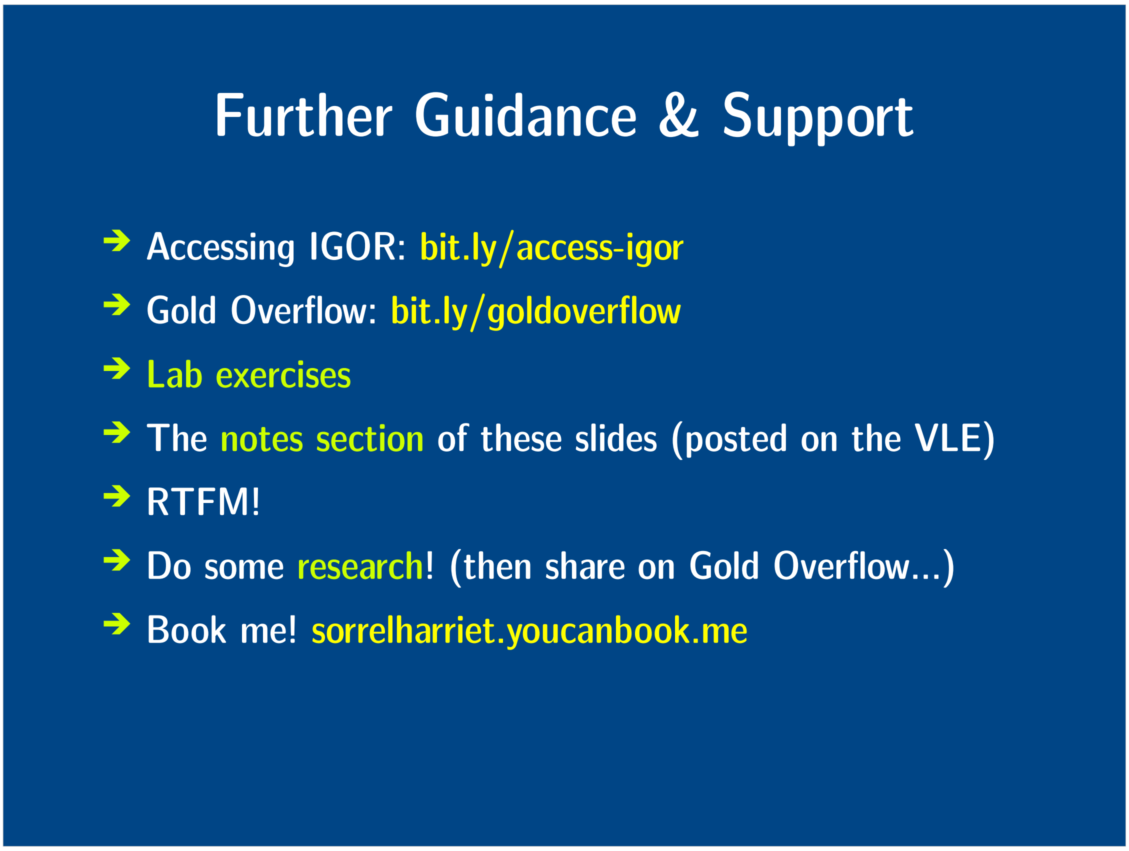
Your Laptop

In this option, you are developing locally, rather than on IGOR. This means that you will need a LAMP stack installed on your computer in order to view your application in a browser at http://localhost.

You would still need to clone a copy of your finished app to IGOR for the tutor to be able to test it.

To avoid the confusion that might arise from maintaining multiple working copies of your projects, we recommend developing your applications on IGOR. As previously mentioned, you can mount the IGOR file system on your home computer, which will allow you to use your preferred editor.

You could also use an FTP software such as FireFTP or FileZilla for transferring files from a local working copy, but this could potentially become confusing.

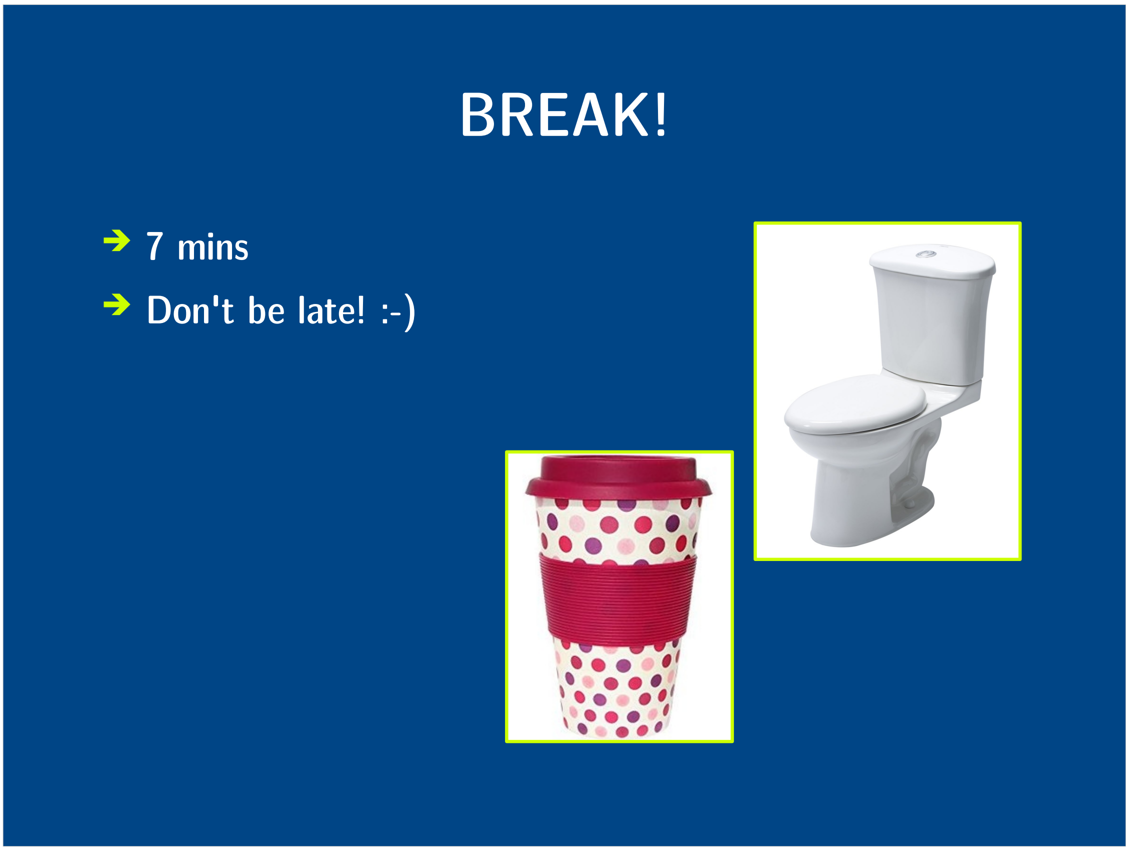


Further Guidance & Support

- Accessing IGOR: bit.ly/access-igor
- Gold Overflow: bit.ly/goldoverflow
- Lab exercises
- The notes section of these slides (posted on the VLE)
- RTFM!
- Do some research! (then share on Gold Overflow...)
- Book me! sorrelharriet.youcanbook.me

There's all sorts of ways you can help yourself/get help. Here are just a few places you can try. DO be proactive about finding information – my lecture slides are visual aids, they aren't a text book!

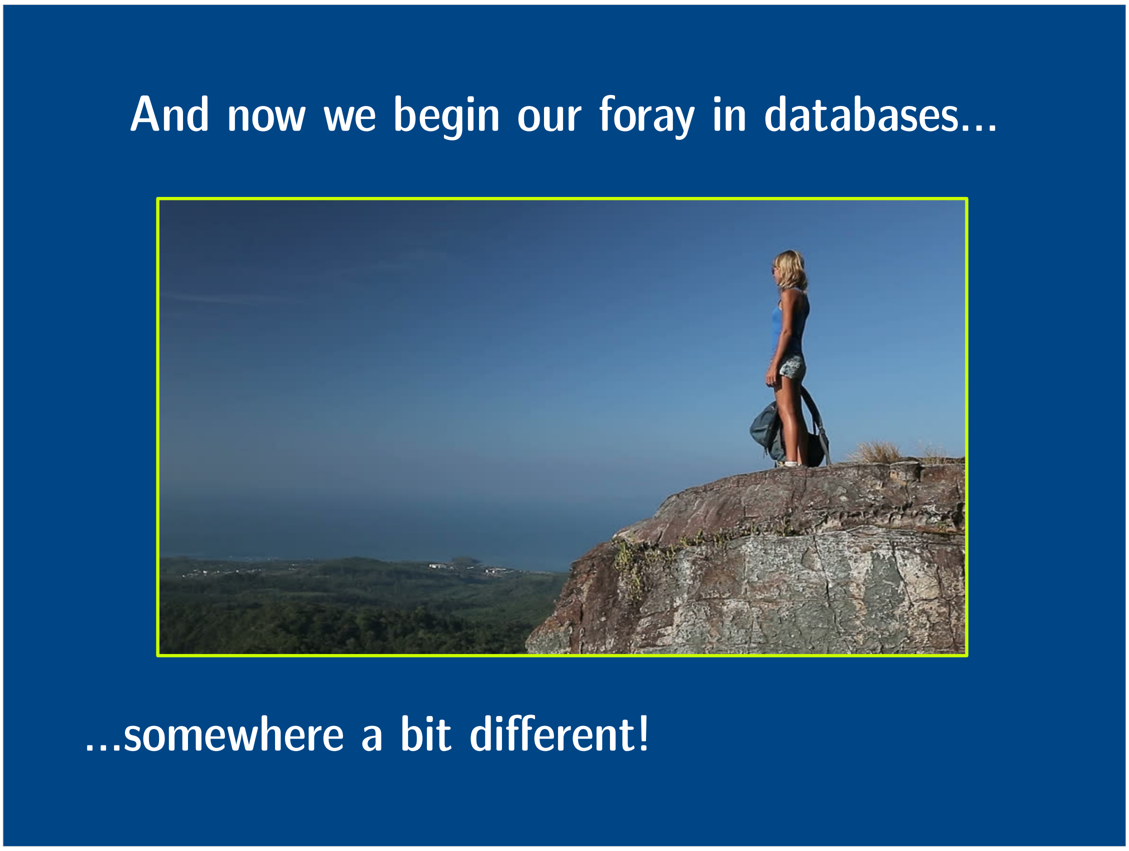
If you come unstuck, don't be shy about seeking help from myself, Dan, or another staff member.



BREAK!

- 7 mins
- Don't be late! :-)

I'll try to make sure you get 5 mins every half hour or so in lectures...in return, please be on time for lectures and after breaks.



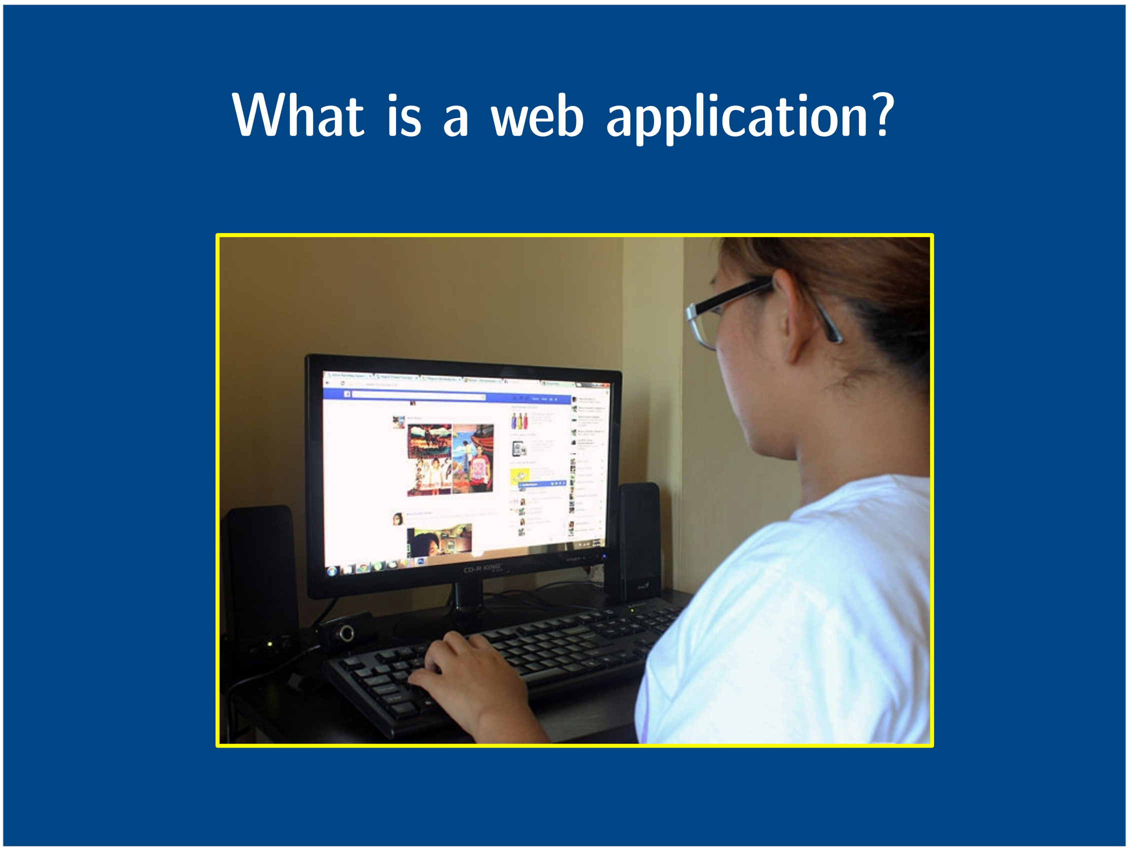
And now we begin our foray in databases...

...somewhere a bit different!

We're going to start our introduction to databases somewhere a bit different...the reason for this is that **you'll be learning databases in the context of building web applications**.

In this context, perhaps more than any other, **SECURITY** of the data is paramount (that's where all the stuff you'll be learning in term 2 about networks comes in). Data **INTEGRITY** and **EFFICIENCY** of retrieval are also pretty major.

Considering databases in this context forces us to look at **THE BIGGER PICTURE**, and that can be really useful, whatever you end up doing with databases in your future careers.

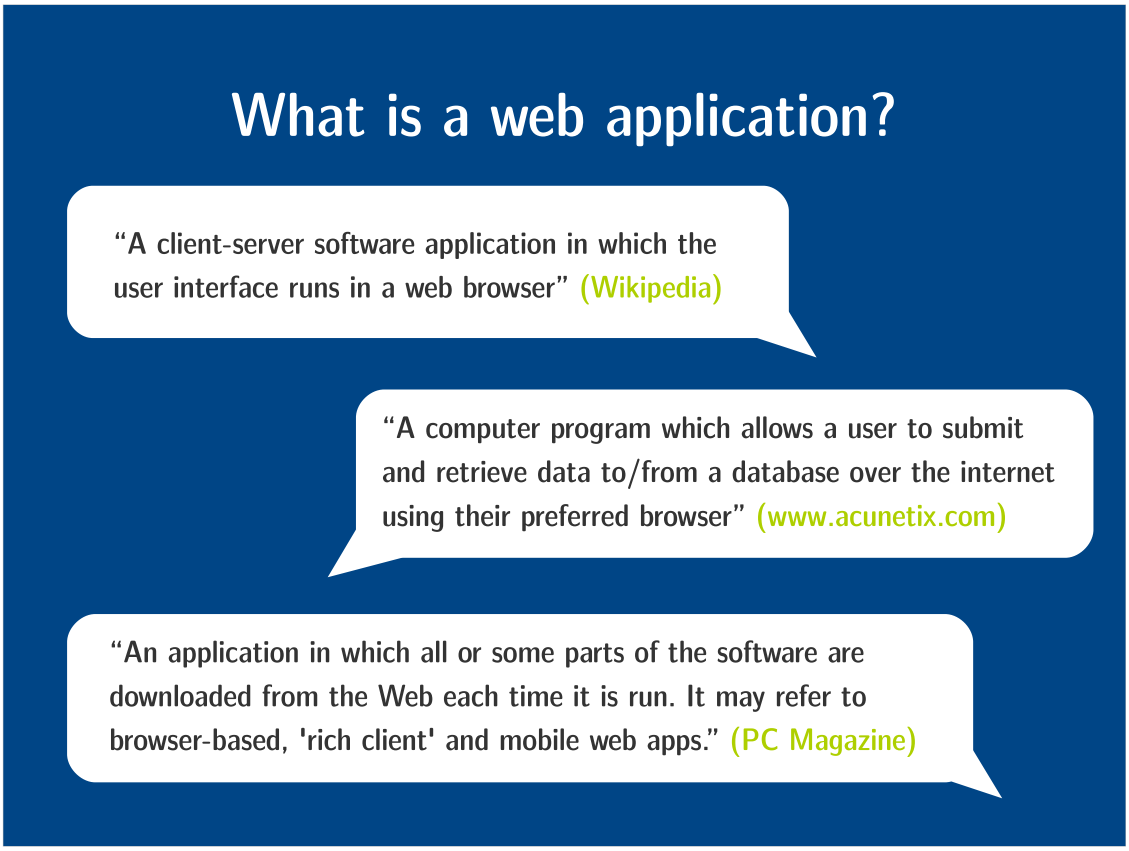


What is a web application?

Activity 1:

Think about the question...jot down anything that comes to mind.

Think also about the various terms you may have heard used in relation to web applications: i.e. static, dynamic, data-driven etc. Write them down too (and a definition, if you have one!)



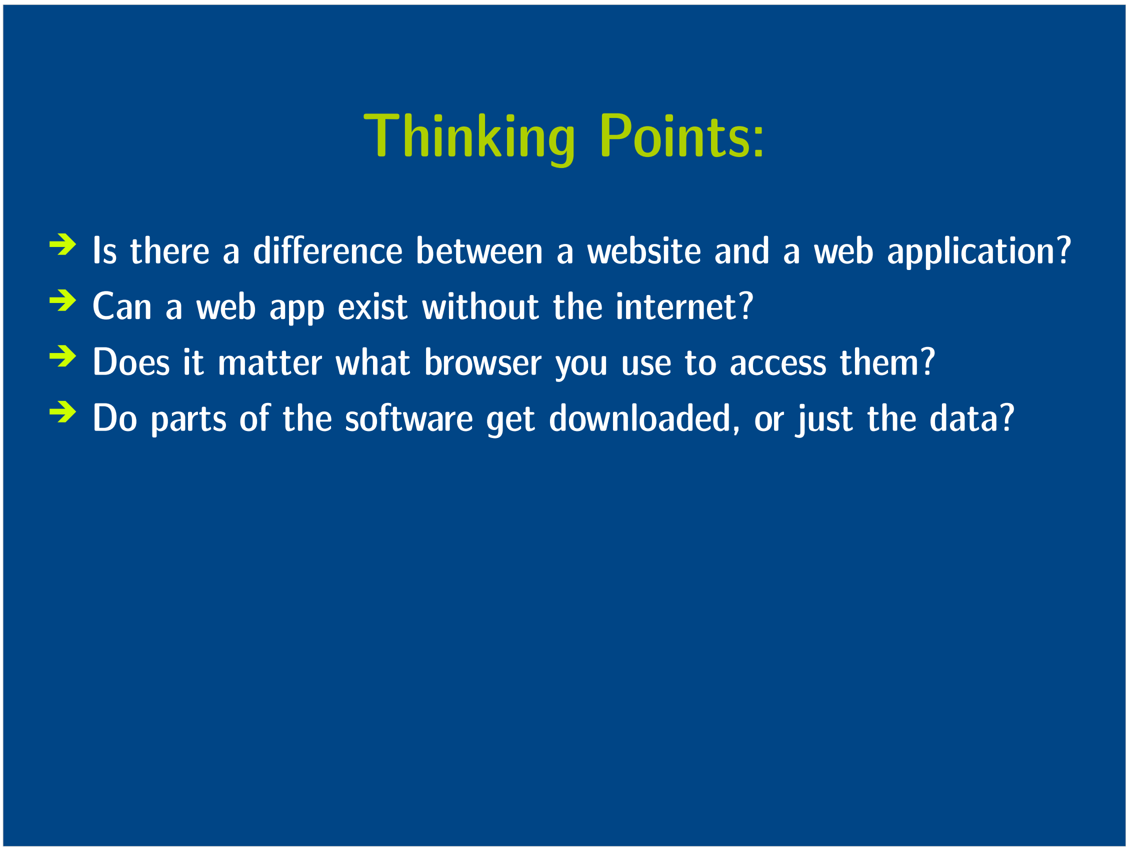
What is a web application?

"A client-server software application in which the user interface runs in a web browser" (Wikipedia)

"A computer program which allows a user to submit and retrieve data to/from a database over the internet using their preferred browser" (www.acunetix.com)

"An application in which all or some parts of the software are downloaded from the Web each time it is run. It may refer to browser-based, 'rich client' and mobile web apps." (PC Magazine)

These are just a few of the definitions I found...

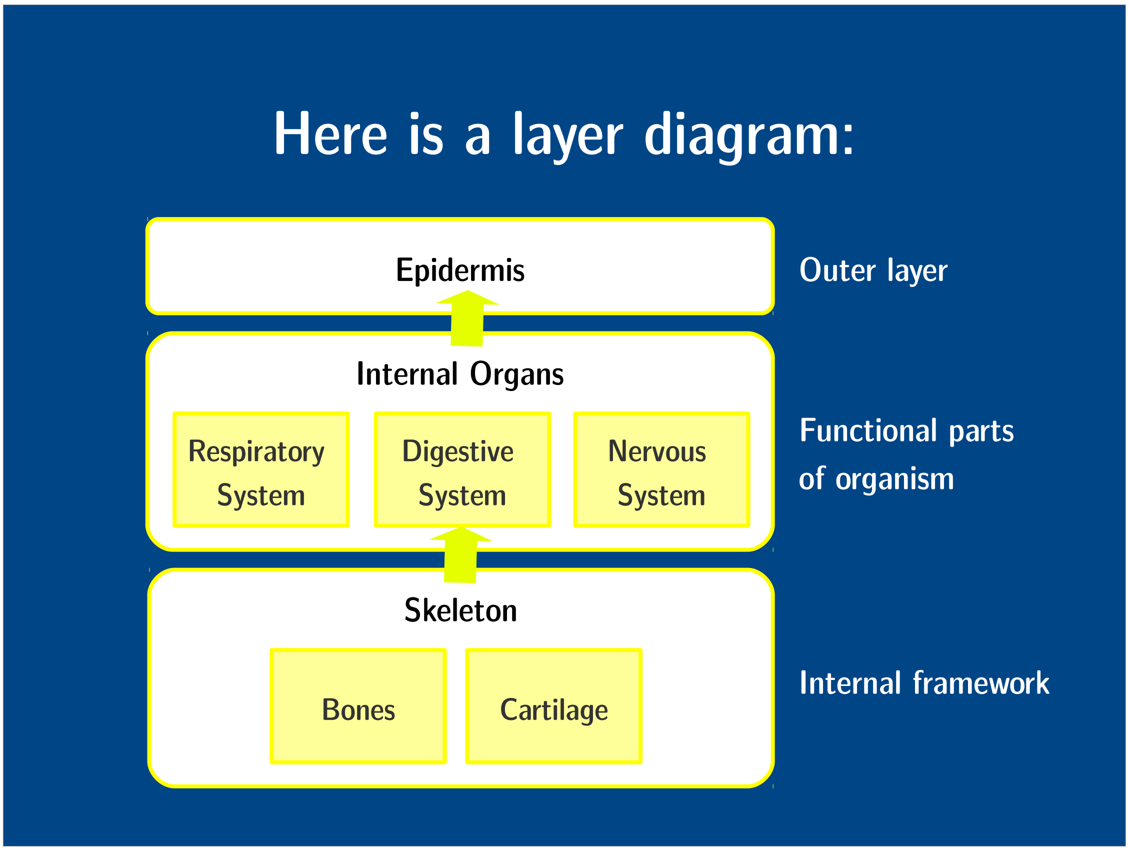


Thinking Points:

- Is there a difference between a website and a web application?
- Can a web app exist without the internet?
- Does it matter what browser you use to access them?
- Do parts of the software get downloaded, or just the data?

Activity 2:
Justify your answers to the person next to you...

Perhaps you've never really thought that much about how web applications work or the technology that is used to build them. From now on, or at least for the remainder of this course, I want you to try!



Here is a layer diagram:

Epidermis

Outer layer

Internal Organs

Respiratory System

Digestive System

Nervous System

Functional parts of organism

Skeleton

Bones

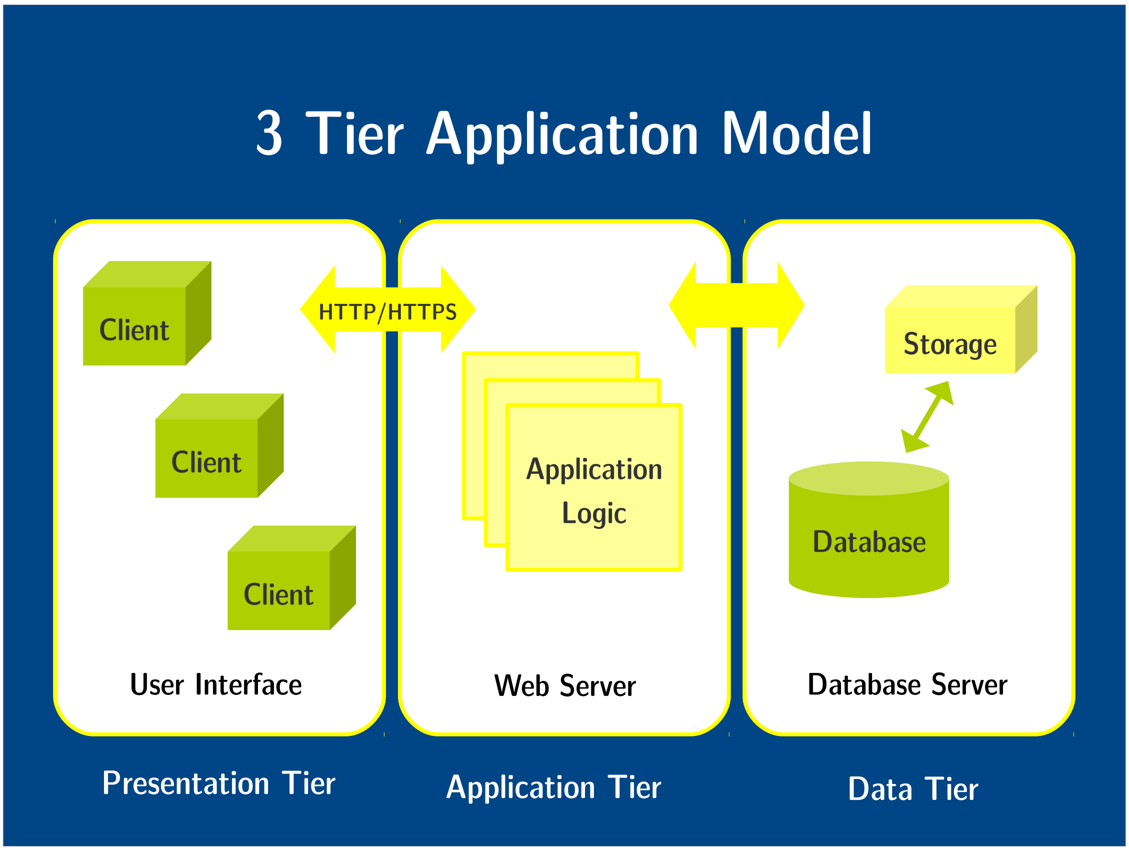
Cartilage

Internal framework

Here's an example of a sort of relation diagram/map of a mammalian body. **Could you come up with one for a web application?**

What would be in each layer?

What would each layer be called?



3 Tier Application Model

Client

Client

Client

HTTP/HTTPS

Application Logic

Storage

Database

User Interface

Web Server

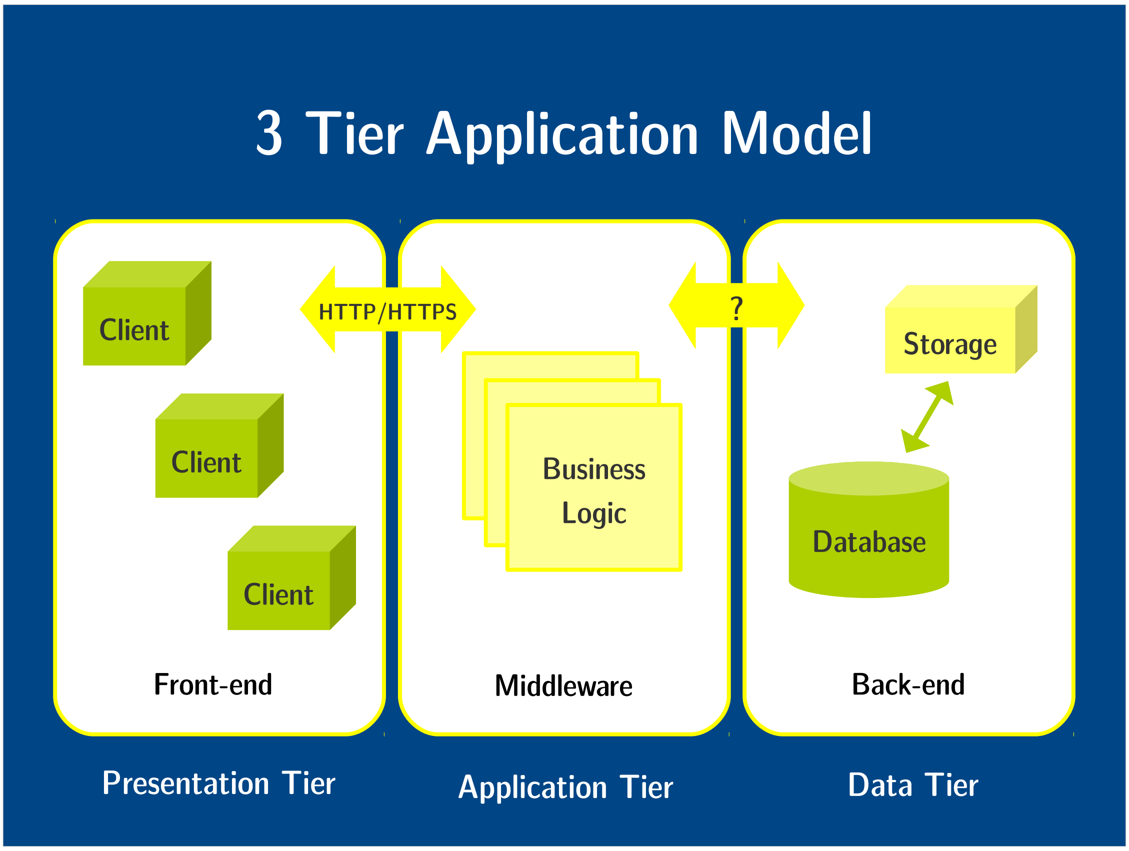
Database Server

Presentation Tier

Application Tier

Data Tier

A common architecture for web applications.



3 Tier Application Model

Client

Client

Client

HTTP/HTTPS

Business Logic

?

Storage

Database

Front-end

Middleware

Back-end

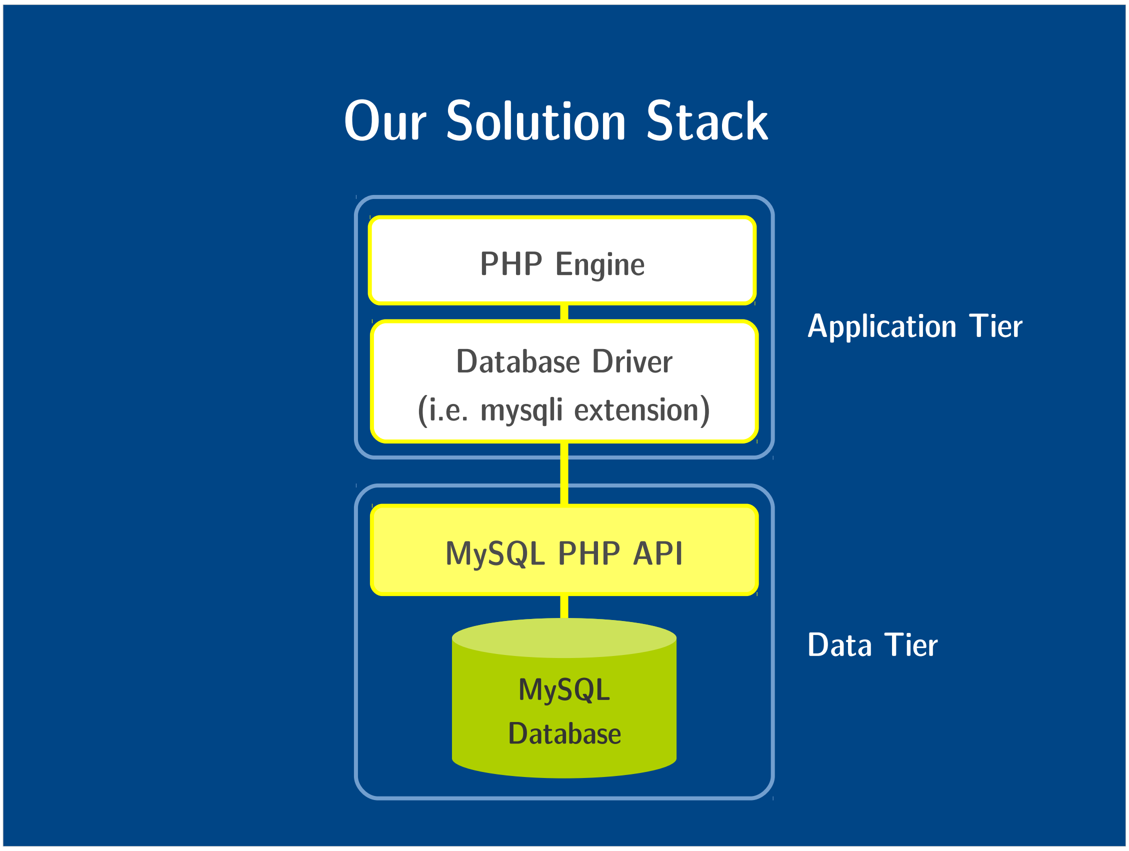
Presentation Tier

Application Tier

Data Tier

Terminology may vary a little, but essentially this slide shows the same thing as the previous one!

I've put a questionmark over the second set of arrows, because we haven't yet decided how the middleware will interface with the back-end.



Our Solution Stack

PHP Engine

Database Driver
(i.e. mysqli extension)

Application Tier

MySQL PHP API

MySQL
Database

Data Tier

Here we see parts 2 & 3 of the 3 tier model reflected in our particular **solution stack** (the one we'll mostly be using on this course).

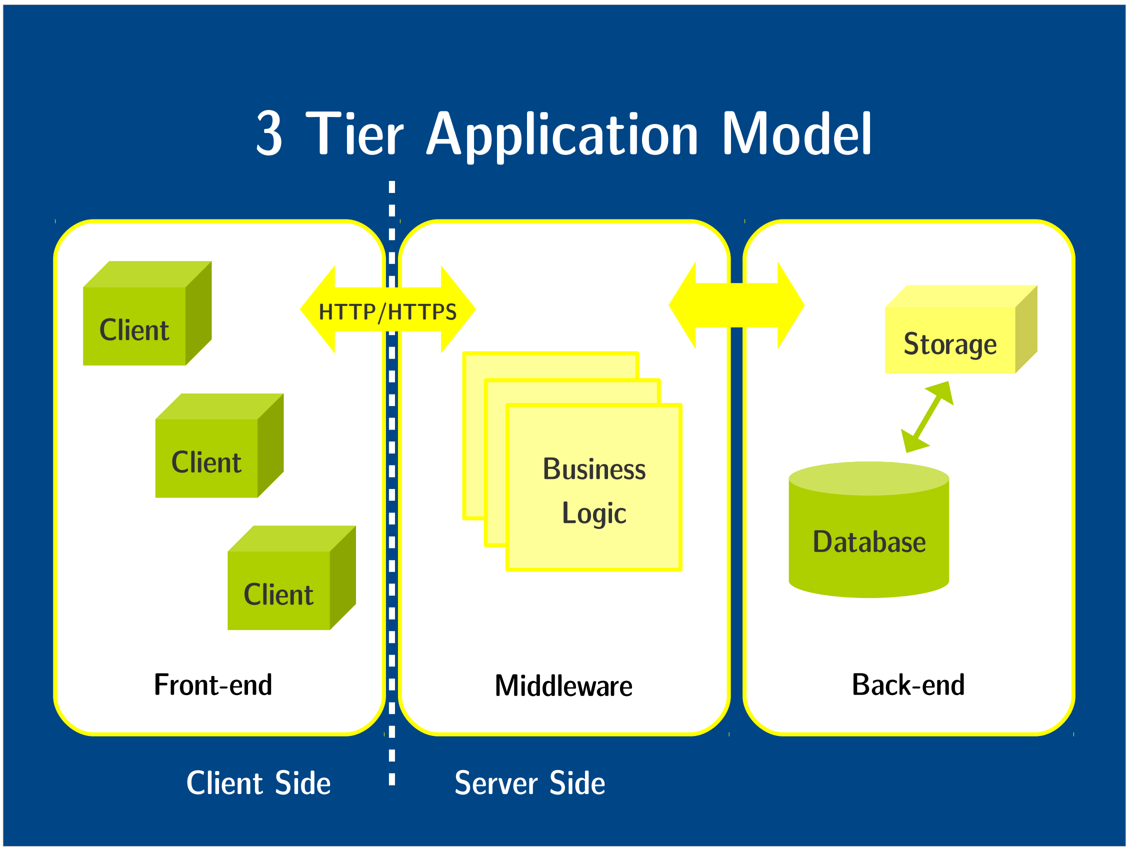
The **Database Management System** (DBMS) we'll use is MySQL; and we'll develop the application logic in PHP.

Note that the **PHP engine** executes the PHP code on the **web server**.

MySQL provides a particular **API** for applications to communicate with its databases via PHP. Different APIs are used by different languages and systems.

Here, the yellow box gives us an answer to the questionmark on the previous slide!

Also note, the software pictured could be co-located on the same physical machine, or it could be installed on separate web and data servers.



3 Tier Application Model

Client

Client

Client

HTTP/HTTPS

Business Logic

Storage

Database

Front-end

Middleware

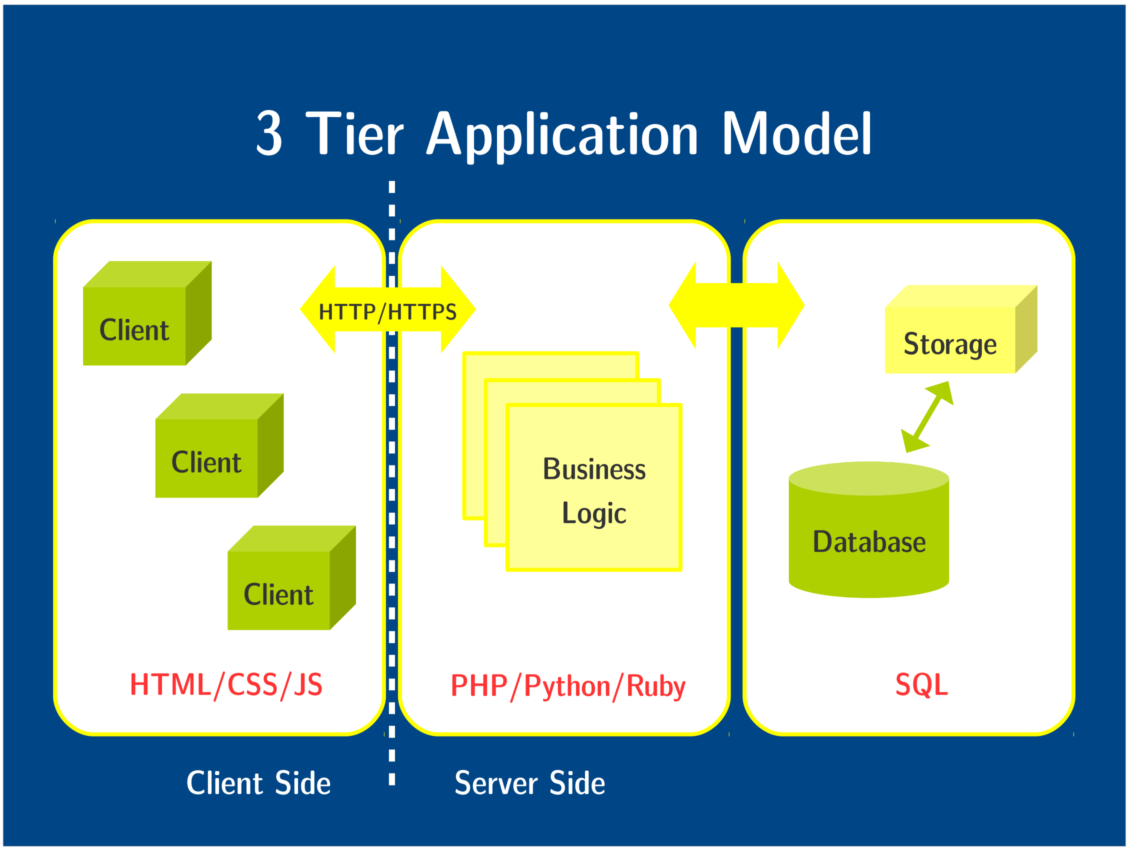
Back-end

Client Side

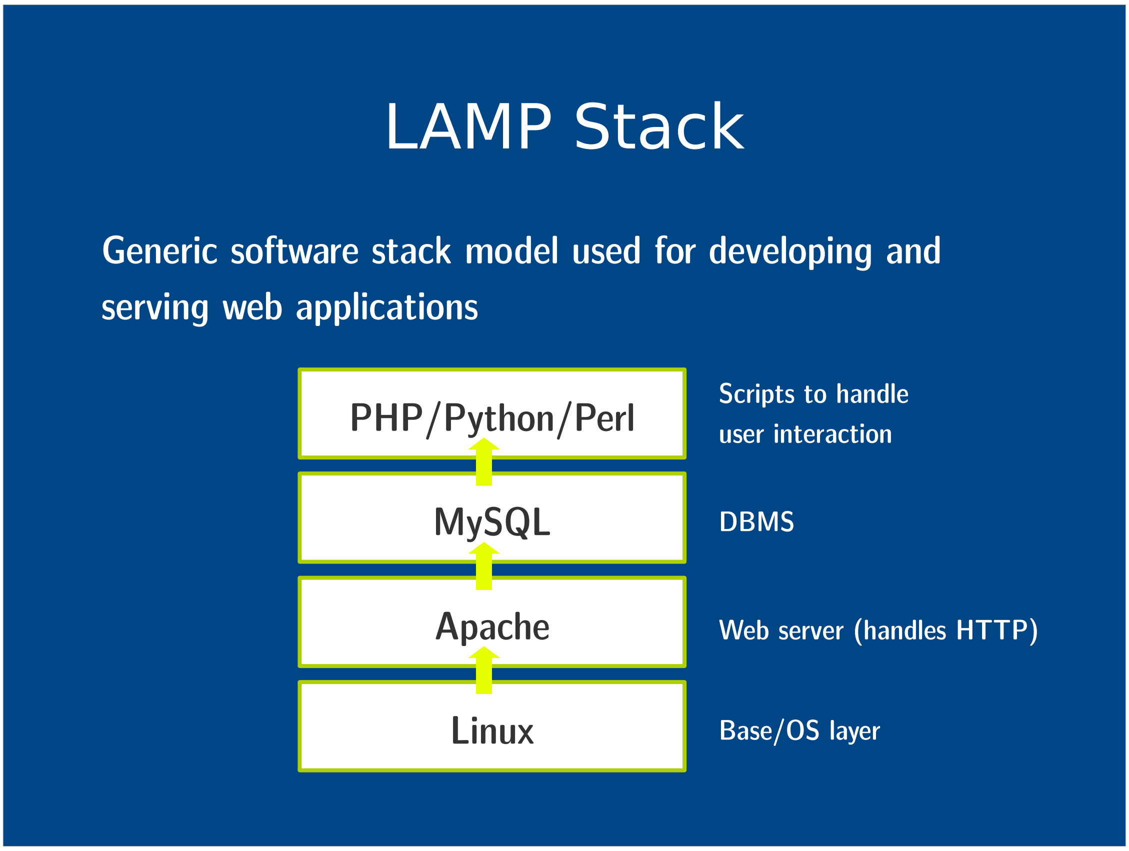
Server Side

Back to our model...

The part that was `missing' from our solution stack was the presentation tier. That's because that part is **client-side**. In other words, that part of the application code gets executed on the client's computer (usually by their browser).



Here you see some of the different languages that are executed on the client and server sides.

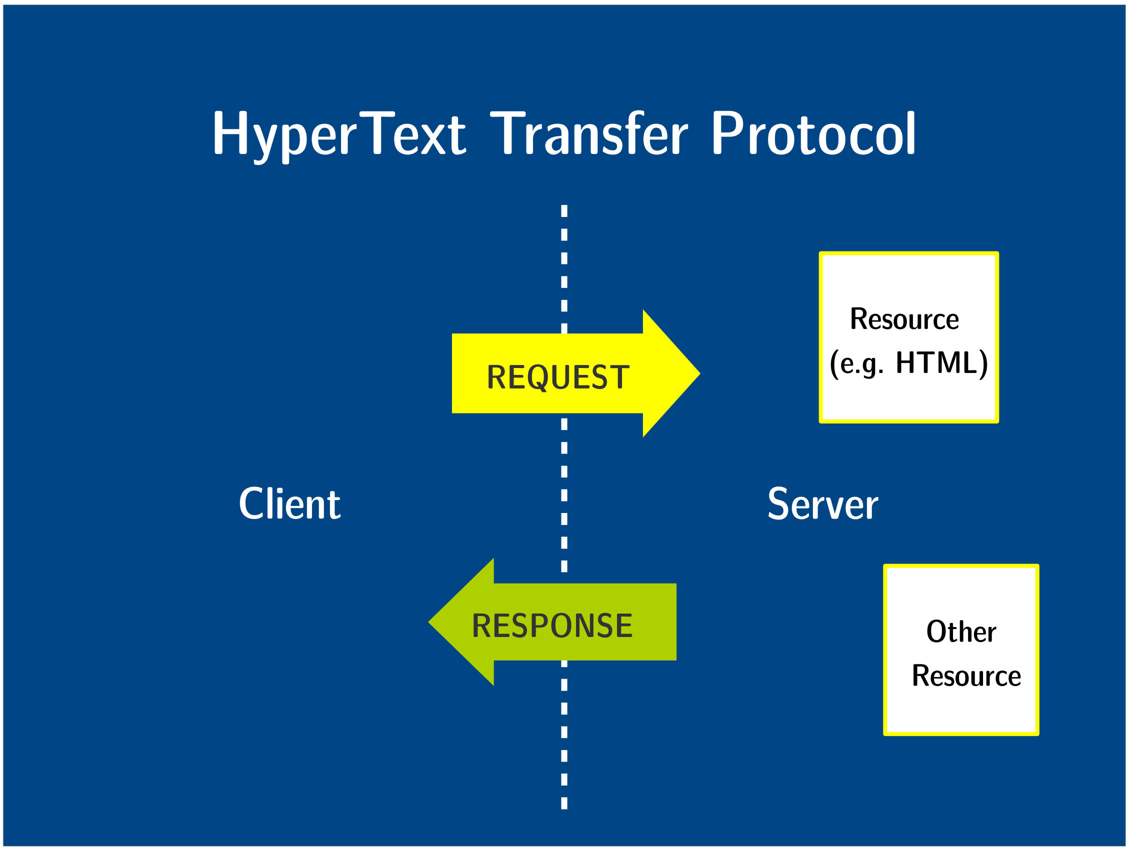


LAMP Stack

Generic software stack model used for developing and serving web applications

PHP/Python/Perl	Scripts to handle user interaction
MySQL	DBMS
Apache	Web server (handles HTTP)
Linux	Base/OS layer

In the first lab session, you'll be getting acquainted with the LAMP environment on IGOR. This is the set-up we will be using for most of the hands-on exercises on this module, although you are free to experiment with alternatives at home. **However**, should you wish to use anything other than a LAMP environment for either of the courseworks, you MUST consult one of the lecturers first.



HyperText Transfer Protocol

REQUEST

Resource (e.g. HTML)

Client

Server

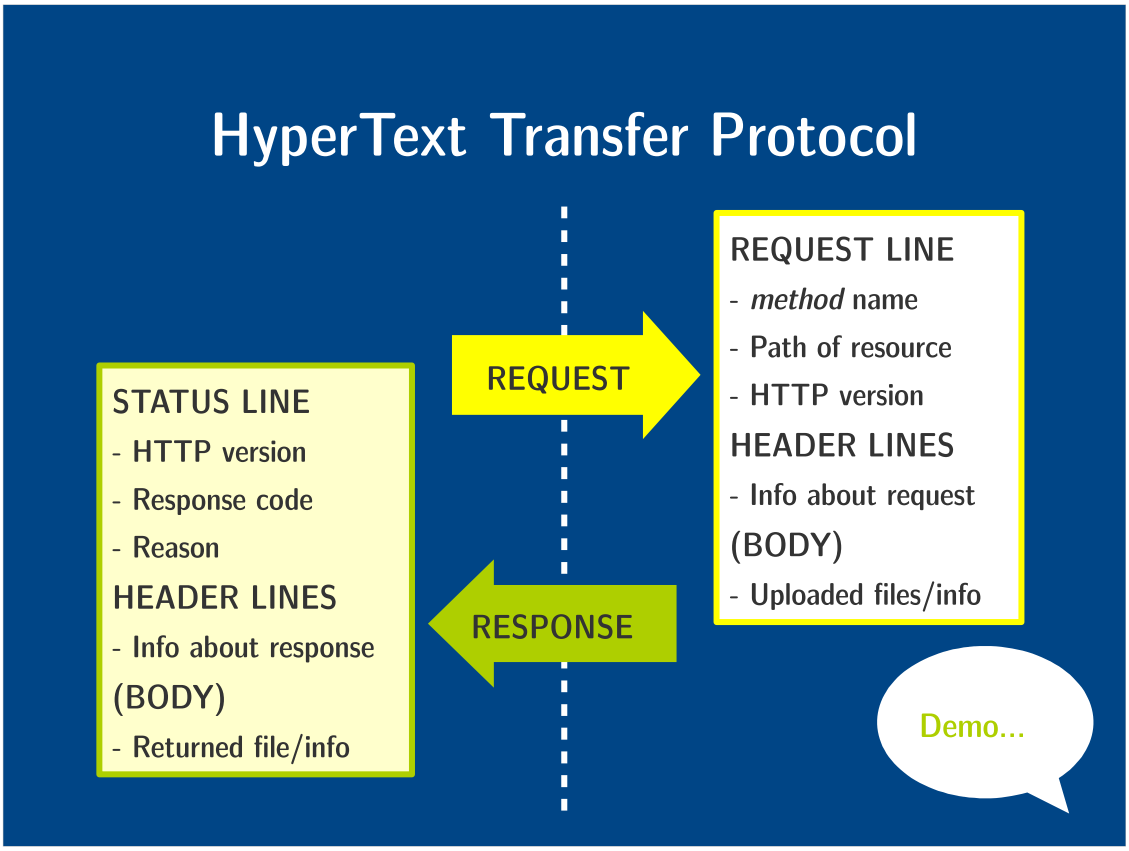
RESPONSE

Other Resource

The client (i.e. web browser) makes a **request** for a particular resource.

Scripts on the server evaluate the request and return a **response** (which may or may not include the requested resource, plus some other stuff...)

The web browser will evaluate the response and decide what to display and how to display it.



HyperText Transfer Protocol

REQUEST LINE
- *method* name
- Path of resource
- HTTP version

HEADER LINES
- Info about request

(BODY)
- Uploaded files/info

REQUEST

RESPONSE

STATUS LINE
- HTTP version
- Response code
- Reason

HEADER LINES
- Info about response

(BODY)
- Returned file/info

Demo...

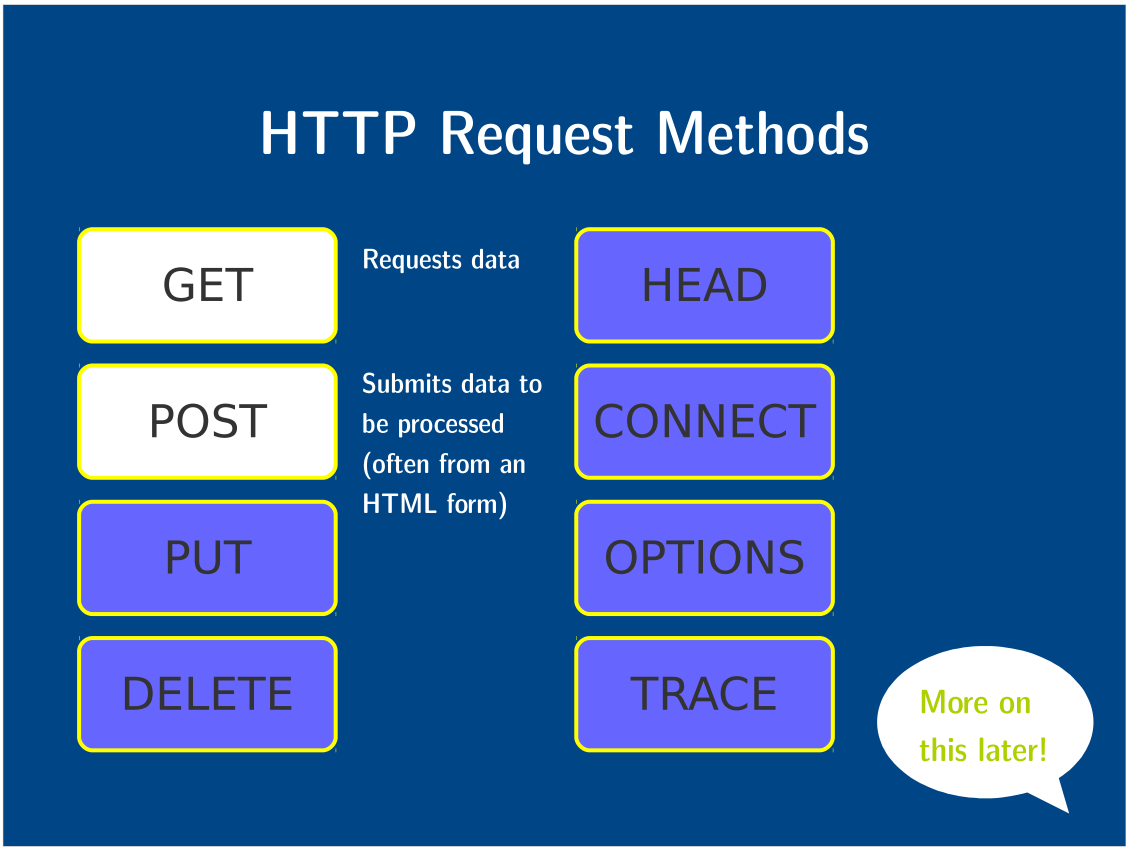
That `other stuff' mentioned in the previous slide will include a **status line**, and a number of **header lines** containing information about the response.

You can explore requests and responses using the **developer tools** in your web browser.

Press F12 to open the toolbar.

In Firefox, go to Net > All to browse headers and responses.

In Chrome, the list of requests is under the Console tab.



HTTP Request Methods

GET

Requests data

HEAD

POST

Submits data to be processed (often from an HTML form)

CONNECT

PUT

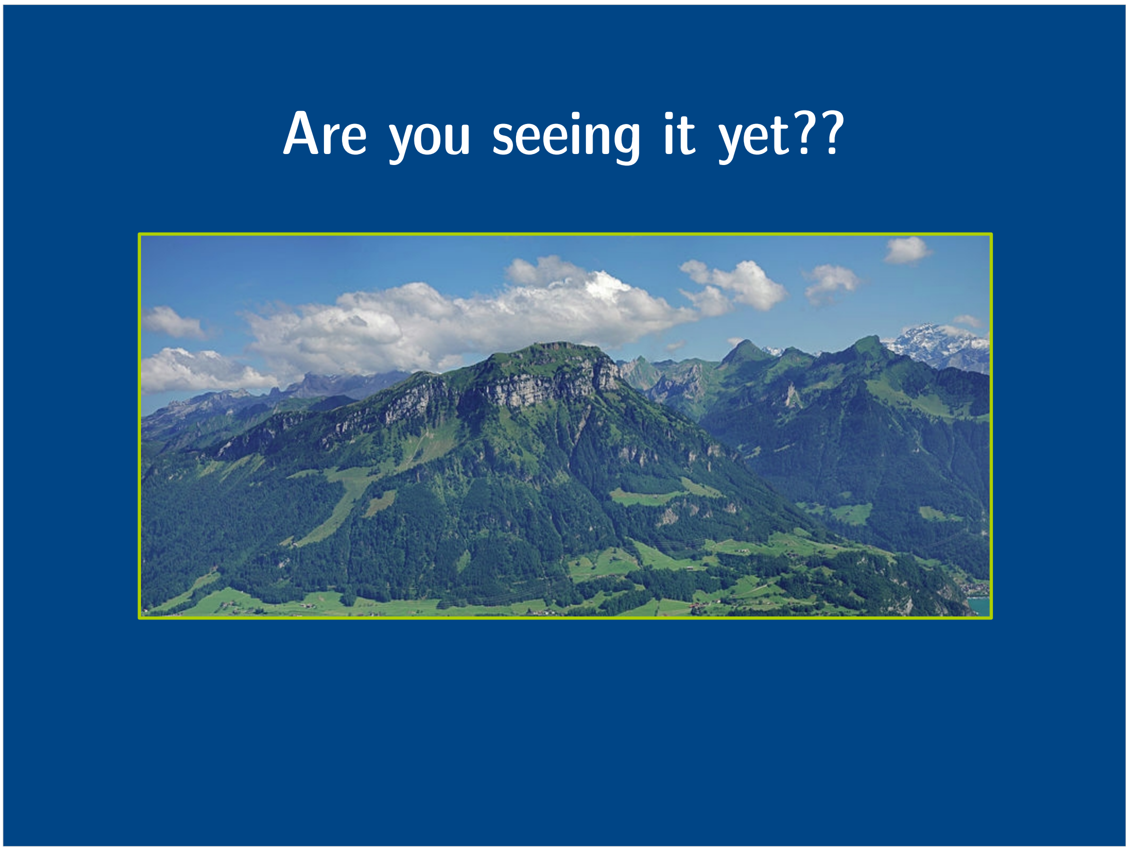
OPTIONS

DELETE

TRACE

More on this later!

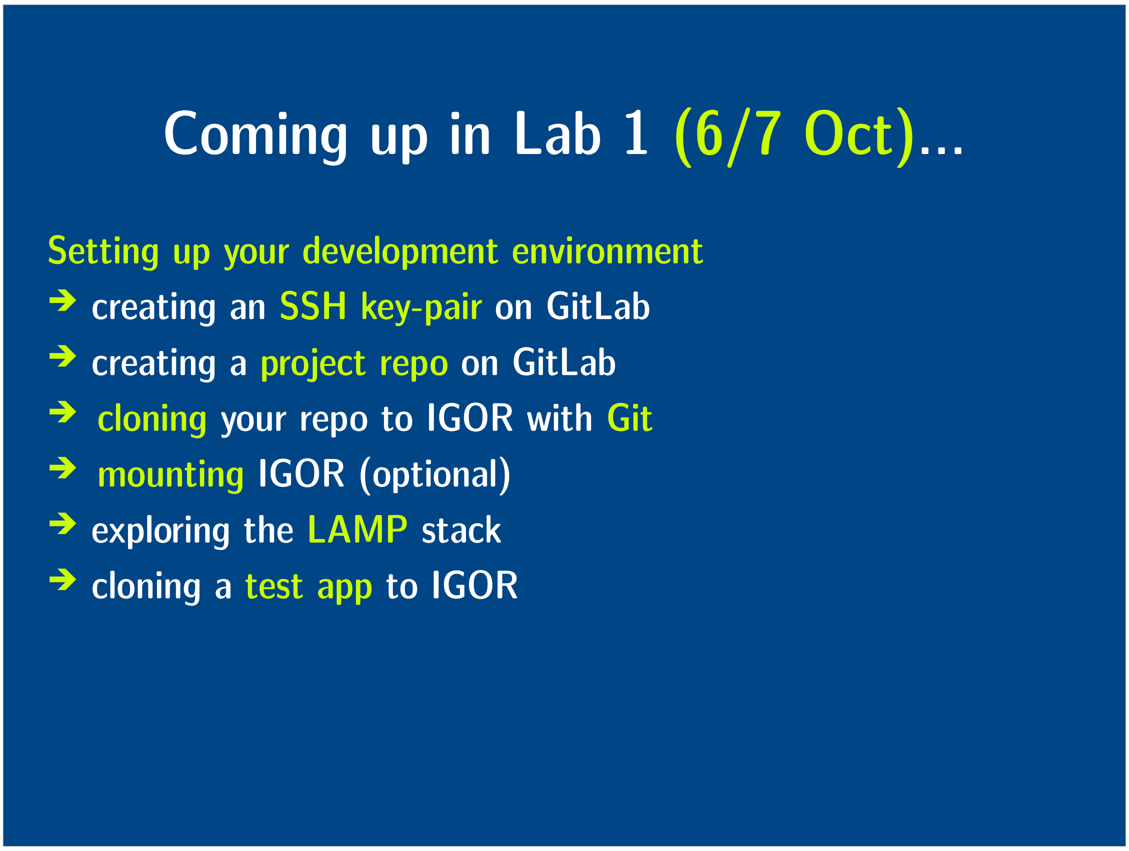
The main HTTP request methods to be aware of are GET and POST. These will be discussed in more detail at a later date!



Are you seeing it yet??

Hopefully you will now have a good understanding of WHAT you will be doing on this course and WHY you are doing it.

You should also have a basic grasp of how common web applications work.



Coming up in Lab 1 (6/7 Oct)...

Setting up your development environment

- ➔ creating an SSH key-pair on GitLab
- ➔ creating a project repo on GitLab
- ➔ cloning your repo to IGOR with Git
- ➔ mounting IGOR (optional)
- ➔ exploring the LAMP stack
- ➔ cloning a test app to IGOR

Lab 1 (in week 2) will focus on getting you up and running with GitLab, establishing your workflow, and finding your way around the LAMP stack.

You'll be creating an SSH key pair and initialising your repo for all term 1 lab work on GitLab; and then cloning and deploying a test application on IGOR.